# Convolutional neural networks: towards less supervision for visual recognition

Maxime Oquab

# THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences Lettres
PSL Research University

## Préparée à l'École normale supérieure

# Convolutional neural networks: towards less supervision for visual recognition

Réseaux de neurones à convolution:
vers moins de supervision pour la reconnaissance visuelle

**École doctorale n°386**

ÉCOLE DOCTORALE DE SCIENCES MATHÉMATIQUES DE PARIS CENTRE

**Spécialité** INFORMATIQUE

**Soutenue par Maxime OQUAB**
**le 26.01.2018**

Dirigée par **Ivan LAPTEV**,
**Josef SIVIC**,
**Léon BOTTOU**.

**COMPOSITION DU JURY** :

M. Andrew Zisserman
University of Oxford
Rapporteur

M. Ross Girshick
Facebook AI Research
Rapporteur

M. Ivan Laptev
Inria Paris
Directeur de thèse

M. Josef Sivic
Inria Paris
Directeur de thèse

M. Léon Bottou
Facebook AI Research
Directeur de thèse

Mme. Cordelia Schmid
Inria
Membre du Jury

M. Francis Bach
Inria
Président du Jury

M. David Lopez-Paz
Facebook AI Research
Membre du Jury

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

## Abstract

This thesis investigates convolutional neural networks for visual recognition. Recent convolutional neural networks have demonstrated excellent performance for a variety of recognition tasks but typically require large amounts of manually annotated training data to perform well. This data is often costly to annotate and may introduce unwanted biases. In this thesis we investigate different ways how to reduce the amount and complexity of required training supervision.

In our first contribution, we propose a transfer learning approach with a convolutional neural network for object classification. We first learn mid-level features on the large ImageNet dataset during a pre-training phase, then we use the parameters to initialize another network designed for a smaller-scale task, where less training data is available. We show, first, that the image representations can be efficiently transferred to other visual recognition tasks, and second, that these representations lead to higher performance when more data is used for pre-training. We demonstrate that the proposed approach outperforms state-of-the-art on the Pascal VOC image classification task.

In our second contribution, we investigate weakly supervised learning for object recognition. We use the fact that for classification, convolutional neural networks tend to take decisions based on the most distinctive parts of objects. This allows us to build a network that can predict the location of objects, based on a weakly annotated dataset indicating only the presence or absence of objects but not their location in images. We demonstrate that our approach improves the state-of-the-art on the Pascal VOC image classification task, performing on par with methods requiring full object-level supervision.

In our third contribution, we look at possible paths for progress in unsupervised learning with neural networks. We study the recent Generative Adversarial Networks; these architectures learn distributions of images and generate new samples, but the evaluation which learned model is better than others is difficult. We propose a two-sample test method for this evaluation problem, allowing us to perform a first level of model selection. We investigate possible links between Generative Adversarial Networks and concepts related to causality, and propose a two-sample test method for the task of causal discovery, outperforming the state of the art. Finally, building on a recent connection with optimal transport, we investigate what these generative algorithms are learning from unlabeled data.

## Résumé

Dans cette thèse nous étudions les réseaux de neurones à convolution dans les systèmes de reconnaissance visuelle. Les réseaux de neurones à convolution récents ont d'excellentes performances pour une grande variété de tâches de reconnaissance, mais requièrent une grande quantité de données d'entraînement, annotées manuellement, pour révéler leur potentiel. Obtenir des données est une opération souvent coûteuse, et qui peut introduire des biais. Dans cette thèse nous étudions différentes manières de réduire la quantité et la compléxité de la supervision.

Notre première contribution est une méthode de transfert d'apprentissage dans les réseaux à convolution pour la classification d'image. Nous apprenons des représentations intermédiaires sur la base de données ImageNet pendant une phase de pré-entraînement, puis utilisons les paramètres appris pour initialiser un réseau concu pour une autre tâche avec moins de données. Nous montrons d'abord que ces représentations sont assez générales pour etre utilisées sur d'autres tâches, et meilleures lorsque le pré-entraînement est réalisé avec plus de données. Ceci nous a permis d'améliorer l'état de l'art en classification d'image sur la base de données Pascal VOC.

Notre deuxième contribution est une approche faiblement supervisé, tirant parti du fait que les réseaux à convolution prennent, pour la classification, des décisions basées sur les parties les plus informatives des objets. Ceci nous a permis de créer un système pouvant predire la localisation des objets en utilisant lors de l'entraînement, seulement l'indication de la présence ou l'absence des objets dans les images, et non leur position. Nous montrons que ce système améliore l'état de l'art en classification d'image sur Pascal VOC, avec des résultats comparables à ceux des systémes disposant de la position des objects.

Dans notre troisième contribution, nous cherchons des pistes de progression en apprentissage non-supervisé. Nous étudions l'algorithme récent des réseaux génératifs adversariaux; ces architectures apprennent des distributions d'images et génèrent de nouveaux exemples, mais l'évaluation d'un modèle appris est difficile. Nous proposons d'utiliser un test statistique pour ce problème, qui permet un premier filtrage des modèles. Nous étudions ensuite le problème de la causalité avec des réseaux génératifs, et proposons d'utiliser un test statistique pour la découverte causale. Finalement, grâce a un lien établi récemment avec les problèmes de transport optimal, nous étudions ce que ces réseaux apprennent des données dans le cas non-supervisé.

# Acknowledgments

Dear Ivan, Josef, Léon,

I would like to thank you for sharing your knowledge with me. When we first met in 2013, I didn't know this would be the turning point of my life. I can not thank you enough for your patience, your efforts, your attention and your time.

I am grateful to the MSR-Inria Joint Centre and to the ERC grants LEAP and ACTIVIA for their financial support during this thesis. I would like to thank Jean and Francis for setting up the Willow and Sierra research groups. If the goal was to build a stimulating environment, then I believe you have succeeded. I would like to thank the members of the jury for taking the time to read this thesis. I hope it was an enjoyable read.

I would like to thank Victor and Mathieu for proofreading this manuscript.

I would like to thank the researchers, assistants, engineers, post-docs, PhD students and interns, who all made this time enjoyable. I will keep fond memories of our lunches, coffee breaks, drinks and deadline pizzas.

I would like to thank the IT specialists for their availability and help.

I would like to thank the communication and contract specialists; I was lucky to have my office next to yours. I would like to thank Hélène in particular for always being there when I needed to be cheered up.

I would like to thank Isabelle, Catherine and Céline for running this place smoothly.

I would like to thank my co-authors Vadim, Minsu and David for their help.

I would like to thank Christophe and Guilhem for no particular reason.

I would like to thank my friends and my family for their support.

I would like to thank Gül for introducing me to İlayda.

I would like to thank Jacques-Alexis and Anne Lise for all the things they taught me, and Paul for sharing both the rent and his family for many years.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

The amount of visual data available online, thanks to readily available phone cameras, social networks, image and video hosting services, keeps growing faster and faster. Millions of pictures are uploaded by humans every single day.[1] The goal of visual recognition is to develop algorithms and methods able to understand the content of all the visual data surrounding us.

With people carrying powerful computers in their pockets, visual recognition has practical applications outside of research labs, in the everyday life. But while it is easy to find more information on something when we know its name thanks to Internet, what happens when we don't? Visual recognition tools give access to more knowledge, by allowing to recognize objects such as animals, plants, or buildings for example, only with pictures using tools such as Google Reverse Image Search on a smartphone.

Because it is fully automatized, visual recognition is also well-suited for security purposes. For instance, millions of hours of footage are produced daily by surveillance cameras, and there is not enough human capacity to monitor all of it and recognize sequences of interest. Similarly, automatizing baggage screening with computer vision

[1] In 2013, 350 million pictures were uploaded on Facebook every day. http://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9

could possibly speed up the whole process of checking for hazardous objects.

Moreover, computers are able to see things that human eyes sometimes cannot see. This has important consequences for medical applications, as an algorithm may be able to detect illnesses in medical images earlier in time than a doctor. This could lead to better treatment, and more lives saved. More generally, if we want to build an intelligent machine capable of reasoning about its surrounding environment, we want it to understand what it sees.

There are plenty of possible applications to visual recognition technology. But understanding the content of images is extremely challenging, as it requires such systems to deal with variations present in visual data, that we discuss next.

## 1.2 Challenges

When observing pictures, humans are able to understand their content and ignore imaging conditions such as changes in the point of view or lighting, occlusions or truncations, as well as cluttered environments. Our perception of an image tends to be *invariant* to these changes. In data, these changes are additional modes of variation against which robustness needs to be implemented in a recognition algorithm. When trying to recognize objects, an algorithm needs to address several problems such as illustrated in the examples below.

**Viewpoint and imaging conditions.** Depending on how a picture was taken, objects can have different appearance in an image. We show an example in Figure 1-1. A visual recognizer should be able to recognize a *chair* from different points of view, e.g. a close-up view, or from the top, or from afar.

**Lighting conditions.** If the light is a little dimmer in a picture, an algorithm should still be able to recognize the objects that are present. Recognition should be invariant to lighting conditions, ideally.


Figure 1-1: Variability in imaging conditions. These *chairs* can have unusual appearance, in particular the middle example seen from above. Images from the Pascal VOC dataset (Everingham et al. [2010]).


Figure 1-2: Variability in size, occlusion and truncation: in these images, examples of *bicycles* can appear partly hidden. Bounding boxes, describing the extent of objects in the images, are shown in yellow. Images from the Pascal VOC dataset (Everingham et al. [2010]).


Figure 1-3: Deformation variability. Animals such as *cats* can appear in various shapes and poses, adding more difficulty. Images from the Pascal VOC dataset (Everingham et al. [2010]).

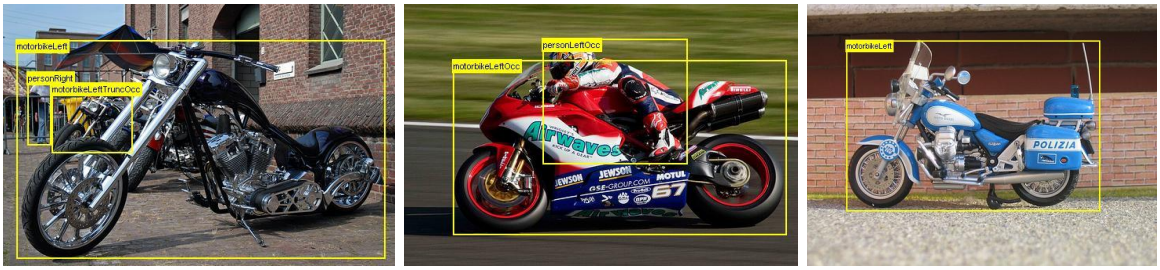



Figure 1-4: Intra-class variability. These examples of *motorbikes* seen from similar points-of-view, can present large differences in appearance while corresponding to the same class or category of objects. Images from the Pascal VOC dataset (Everingham et al. [2010]).

**Occlusions.** Some parts of objects may be hidden behind another object, as we show in Figure 1-2. These make objects more difficult to recognize, as only some parts of them are visible. When the configuration of an object is such that it partly occludes itself, we talk about self-occlusion.

**Background clutter.** In images, objects are not necessarily on a clean background, and other elements in the background may possibly distract a visual recognizer.

**Deformations.** This mode of variation happens more in the case of animals, that can be seen in a variety of poses. We show the example of cats in Figure 1-3, but this is especially true for humans as well.

**Intra-class variability.** In the data, objects belonging to a same category can still present large differences in appearance. While the nature of the objects can be the same, some parts may look different, as we show on motorbikes in Figure 1-4. This is known as intra-class variability and is a consequence of human decisions: it depends on how the data is labeled by annotators.

Some of the invariances mentioned above can be quite complex, and in order to build robustness against them, methods rely on empirical approaches, using *machine learning* as we will discuss next.

## 1.3 Visual recognition as a machine learning task

The different modes of variation in images can be difficult to express formally. Therefore, machine learning methods are used to learn these modes of variation from labelled data. In visual recognition, image classification is a base building block.

**Image classification.** Let us take a look at a simple dog vs. cat classification task. Given an image, the task consists of determining whether an image contains a cat or a dog. We can solve the task in Figure 1-5 because we can rely on appearance;

Figure 1-5: Example of a classification task : the goal here is to train an algorithm to predict whether an image contains a cat or a dog.

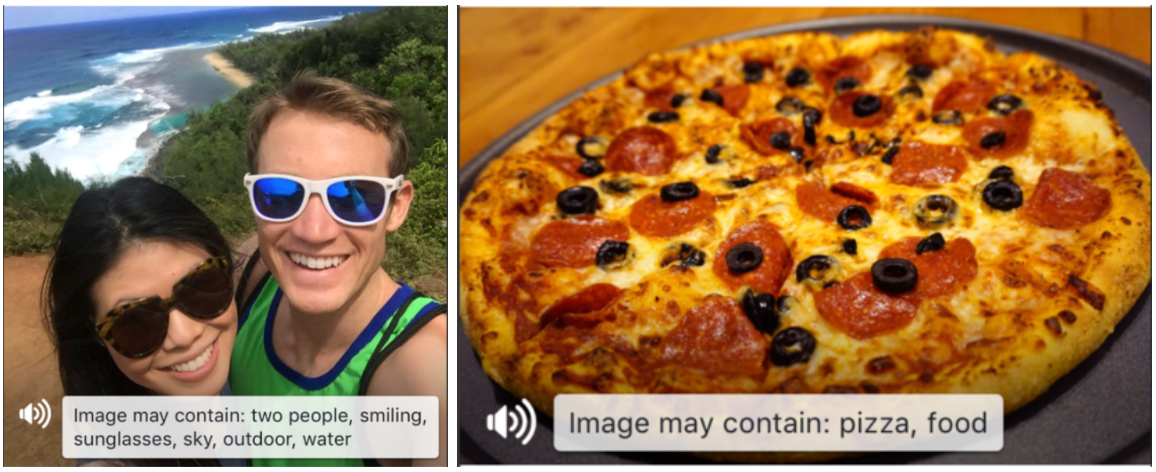



Figure 1-6: An application of classification: Automatic Alternative Text, describing images by listing what concepts are present, for the visually impaired.

intuitively, heads of dogs show similar patterns, and heads of cats show similar patterns, but dogs and cats look different. Learning algorithms, given enough examples of cats and dogs, will eventually become able to discriminate between the two. In 2017, data is plenty and algorithms for image classification are mature enough to run corresponding applications at a large scale. For example, Automatic Alternative Text[2], developed by Facebook and shown in Figure 1-6, uses classification to describe which objects or concepts are present in images, in order to improve the experience for the visually impaired.

[2] https://code.facebook.com/posts/457605107772545/

**Convolutional neural networks.** Among these machine learning approaches for computer vision, *convolutional neural networks* have recently attracted attention because of their great performance for visual recognition, outperforming all alternatives as we will describe in Chapter 2. These algorithms, invented in the '80s, will be the main focus of this thesis.

**The issue of interactions with objects.** Let's assume an image classification algorithm let us find images that contain a person and a phone. At that point it is quite possible that the person is using a phone, as we show in Figure 1-7. However, one may be interested in answering a more precise question, such as: is this person phoning, texting or taking a picture?

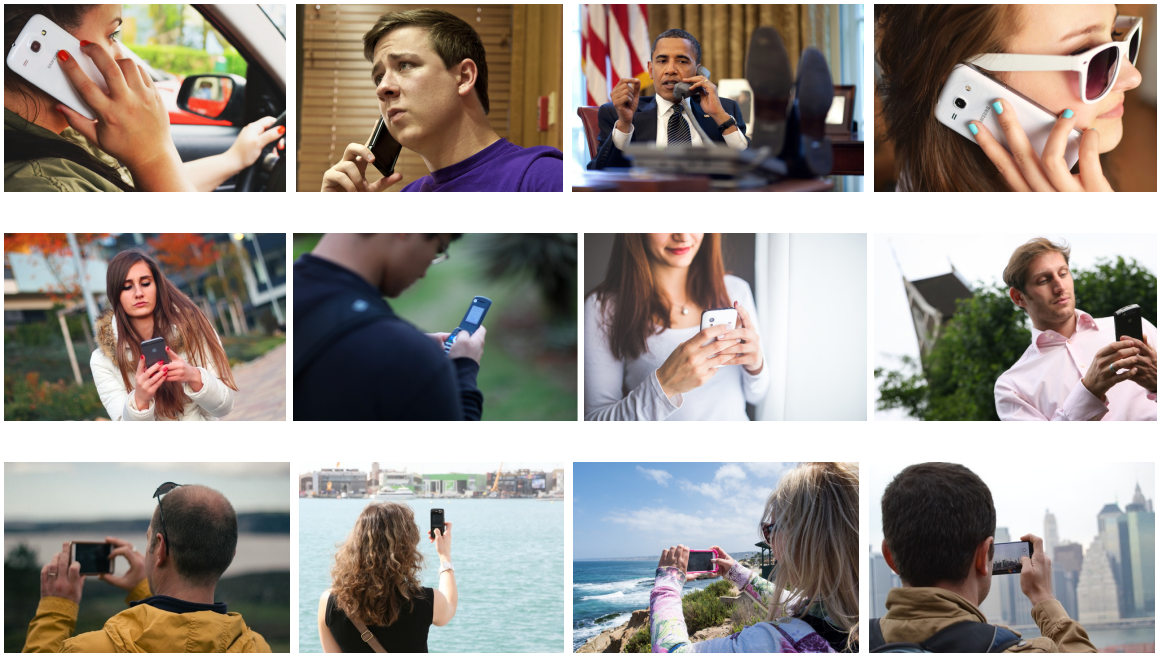


Figure 1-7: Recognizing interactions with objects is one step beyond object recognition, and we are now interested in *how* a person interacts with an object.

Solving this task should be possible as well, but may require interpreting more subtle cues, such as the positioning of the hands on the phone, the relative position of the phone to the head. Knowledge of the presence or absence of a phone is not

sufficient anymore to provide an answer, and therefore we need more data to capture the corresponding patterns. However, since it is possible for a person to interact in different ways with a given object, the number of possible actions grows quicker than the number of different objects: this means that solving this task may require more data. In general, algorithms deliver higher performance on tasks when trained with more data. And for more difficult tasks, more data is necessary to obtain good performance. But data is in general not readily available as we discuss next.

**The cost of data collection.** In order to be exploited by current algorithms, images need to be annotated to provide *supervision* to the machine. In the supervised learning context, an input is provided to an algorithm as well as the desired output. This means that the limit on the complexity of tasks that we can solve is set by how difficult it is to annotate data, keeping in mind that it gets harder to reach human agreement (*labeling consistency* across examples) for more abstract tasks. One possible issue is the granularity of labeling; for example, should an example be labeled as *cooking*, *chopping onions*, or *both* ?

While the presence of more data should be beneficial to cover the different modes of variability of images, obtaining data has a cost. In the current supervised learning paradigm, the only recognition tasks that can be addressed are the ones for which there is enough human agreement to crowdsource data collection efficiently. In crowdsourcing applications such as Amazon Mechanical Turk, an annotation task is proposed to a worker in exchange for a small amount of money. If the task is difficult then annotating an example costs more. To decrease that cost, one can consider tasks for which smaller amounts of data are necessary, or for which data collection is easier.

The cost of data is one of our main concerns; some tasks, such as recognizing the presence of objects, can be solved with high enough performance to be deployed at a large scale in industrial applications. But more difficult tasks such as recognizing interactions between humans with objects, may require too much data to be solved reliably. Therefore in this thesis we are interested in methods requiring less annotations in the data. The hope is that if algorithms were able to learn from less annotations

(or no annotations at all), they should have access to more data examples, and be potentially able to solve more difficult tasks.

## 1.4 Goals of this thesis

Even though annotating data is an expensive and limited process it is still reasonable in the case of objects, where human agreement is easy to obtain, allowing to build large databases such as ImageNet (Russakovsky et al. [2015]) which appeared in 2009. ImageNet provides a massive source of annotated data to the research community, exceeding in size everything that was available before, and paving the way for new algorithms. Building on this database, Krizhevsky et al. [2012] demonstrated in 2012 how to use a dataset of a million images to train the largest neural network ever built at that time, winning the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC-2012) by a large margin as we will describe in Chapter 2. Outperforming all other methods, this work revealed the full potential of *convolutional neural networks* for visual recognition, which will be the main focus of this thesis. But annotating and using one million images may be too costly to collect for any given task, and neural networks need a lot of data to perform well.

**Transferring learned image representations.** The first goal of this work is to find how to obtain the performance of large neural networks on other visual recognition tasks, without paying the cost of annotating large datasets, nor spending time building a working training setup from scratch.

**Learning from weak supervision.** The second goal of this work is to understand how precise annotations should be in order to solve a task with a convolutional neural network. The main question is whether data should be provided with the exact answers that are expected from the algorithm afterwards, or if it is possible to design a smarter algorithm, that can learn from less precise hints. Can we learn to localize objects only using less expensive image-level annotations?

**Learning without manual supervision.** Pushing this idea to the limit, we are interested in which regularities are natively present in data, if these can be retrieved only by observing data without any form of supervision, and if they can be useful for training algorithms. This *unsupervised learning* is the holy Grail of machine learning; our third goal is to explore this direction with convolutional neural networks.

## 1.5 Outline and contributions of this thesis

### 1.5.1 Outline

This manuscript contains six chapters including this introduction and a conclusion chapter, and two technical background appendices.

**Related work.** In Chapter 2, we will present an overview of previously published work related to our goals. The work of Krizhevsky et al. [2012] on ImageNet, published shortly before the beginning of this thesis, sparked great interest in neural networks from the computer vision community. Therefore, we will review the fields of visual recognition and neural networks, and illustrate the paradigm shift following the ImageNet event. We will point out that already in the 90's neural networks were very close to their current form, awaiting the technology and data made available only in the recent years to demonstrate their performance. Today, most state-of-the-art visual recognition systems involve a neural network component.

**Learning and transferring mid-level image representations.** In Chapter 3 we will present our first contribution, the study of a procedure called "pre-training" that leads to important improvements for many tasks in computer vision. The idea is the following: we first train a large neural network on a task for which we have plenty of data. Then, we show that the image representations learned by this network can be efficiently transferred to other visual recognition tasks where less data is available, leading to significant improvements. In particular, we show that the image representations lead to higher performance when the network is pre-trained with more data.

This technique allows leveraging the power of very large and powerful neural networks to smaller-scale tasks involving natural images. *Pre-training* is now a standard procedure in computer vision, eliminating the long process of training large networks from scratch.

**Weakly supervised learning.** In Chapter 4 we will present our second contribution, the study of the behavior of neural networks in the context of weak supervision. We observed that classification neural networks respond strongly to the *most distinctive parts* of objects, such as the heads of cats and dogs, or the wheels of a car. We extended this behavior to images and trained a neural network only with information on the presence or absence of objects in images; we observed that the algorithm was capable of retrieving the location of these objects, an information not available at training time, by relying on statistical regularities present in the data. This setup is an example showing it is possible for an algorithm to learn with less precise supervision than what was previously expected.

**Unsupervised learning.** In Chapter 5 the focus will be on unsupervised learning; first we will describe Generative Adversarial Networks (GANs, Goodfellow et al. [2014]), a recent class of generative models implemented as neural networks, that aim at learning the distribution of a set of images to generate new samples. This method, delivering appealing results, greatly increased the interest of the community on unsupervised learning, and the corresponding field is now very active. In this chapter, much more exploratory, our goal is to understand the underlying issues better and also expose trails worthy of study on this problem. We investigate the GAN evaluation problem and possible links with the concept of *causality*, for which we propose the use of classifier-based statistical two-sample tests. We also investigate the difficult subject of distances between distributions of images, related to GANs, and expose insights on what these algorithms are learning building on recent work on optimal transport.

**Technical background.** In addition to the main text, in Appendices A and B we provide technical background covering the mathematical and machine learning notions necessary for understanding the rest of the thesis. These appendices correspond to an effort to explain that neural networks are conceptually simple methods, despite the difficulties that are met in practice. We describe convolutional feed-forward neural networks and their associated learning algorithms. In particular, we present their core elements and explain how they leverage the power of GPUs through linear algebra-based operations. We show that neural networks correspond to a restricted class of parameterized functions, *following an architecture*, that fit into a simple gradient-based machine learning framework.

### 1.5.2 Publications

The material in Chapter 3 was published and selected for an oral presentation in CVPR 2014 under the title: *Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks*, Oquab et al. [2014].

The material in Chapter 4 was published in CVPR 2015 under the title: *Is object localization for free? - Weakly-supervised learning with convolutional neural networks*, Oquab et al. [2015].

In addition, an extension of Chapter 4 was published in ECCV 2016 under the title: "*ContextLocNet: Context-Aware Deep Network Models for Weakly Supervised Localization*", Kantorov et al. [2016].

Parts of Chapter 5 (Sections 5.3 and 5.4.2) were published in ICLR 2017 under the title: "*Revisiting Classifier Two-Sample Tests*", Lopez-Paz and Oquab [2017].

### 1.5.3 Software contributions

Apart from software to reproduce results of the published papers, during the course of this thesis, we also shared open-source GPU code packages for the Torch7 software (Collobert et al. [2011a]). These contributions include a GPU convolution code (`https://github.com/qassemoquab/nnbhwd`) and jittering code using texture

units (https://github.com/qassemoquab/texfuncs) for real-time data augmentation, that we used for our work in Chapter 4.

We also contributed an implementation of Spatial Transformer Networks (Jaderberg et al. [2015]) including a flexible and efficient bilinear interpolation code for images (https://github.com/qassemoquab/stnbhwd, starred 242 times). Our package was notably used in the `gvnn` package (Geometric Vision, Handa et al. [2016]), and various papers (e.g. Johnson et al. [2016], Reed et al. [2016]).

# Chapter 2

# Related work

In this chapter, we present early work on neural networks and computer vision separately, underlining important contributions that shaped the corresponding research along the years. Then, we describe the collision between these fields, that happened in 2012 with the success of Krizhevsky et al. [2012] in the ImageNet visual recognition challenge. Finally, we overview subsequences of this event that caused changes in the majority of areas of visual recognition.

In Section 2.1, we focus on the history of computer vision methods, underlining the importance of feature descriptors such as SIFT in vision pipelines. In Section 2.2, we focus on neural network methods, in particular showing that the algorithms used today are similar to the ones that have been developed in the '80s. In Section 2.3, we describe the ImageNet competition event, and the following unexpected increase of interest in neural networks from the computer vision community, notably through feature descriptors built from neural networks. Our work in Chapter 3 fits within that period and we provide more related references in this section. In Section 2.4 we then overview the current trend of integrating components of computer vision pipelines within neural network architectures with improved features and end-to-end differentiable setups. In Section 2.5 we discuss the problem of limited annotation and describe approaches leveraging weaker forms of supervision; we then review methods for weakly-supervised object localization, related to our work in Chapter 4. We

also discuss recent approaches on unsupervised learning with neural networks and in particular Generative Adversarial Networks, related to our work in Chapter 5.

## 2.1 Early days in vision

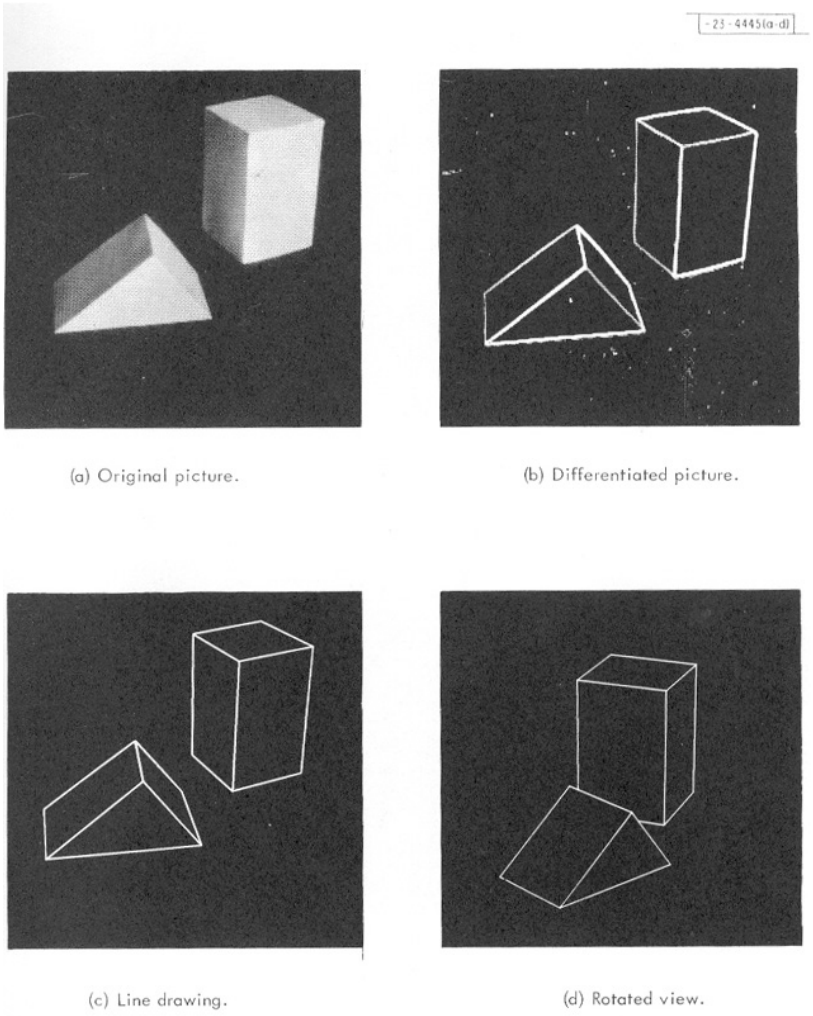



Figure 2-1: One of the earliest approaches in Computer Vision, by Roberts [1963]. Roberts attempts to reconstruct a three-dimensional shape from a picture.

**Geometrical models.** One of the first approaches to computer vision dates back to 1963 with the PhD thesis of L.G. Roberts (Roberts [1963]), attempting to reconstruct a three-dimensional shape from a two-dimensional picture, using the information of the edges, and assuming planar surfaces, as shown in Figure 2-1. Reconstruction of

simple geometric shapes was pushed further in the following decades. Duda and Hart [1972] introduce the Hough transform to detect lines. Canny [1986] proposes an edge detector to simplify the content of images. Huttenlocher [1987] presents a recognition approach by aligning an image of an object with a corresponding 3D CAD model.

As explicit 3D object reconstruction from images has shown to be difficult, another line of work focuses on studying so-called invariants, i.e. view-invariant object signatures in the image. Weiss [1988] looks for invariants by studying the projective nature of image formation. Rothwell et al. [1992] build on projective invariants to propose a method for recognizing planar objects from images. Planar objects are particularly interesting because the appearance variability, e.g. due to illumination, is much lower. Burns et al. [1993], however, establish that there are no general-case view-invariants for real three-dimensional objects.

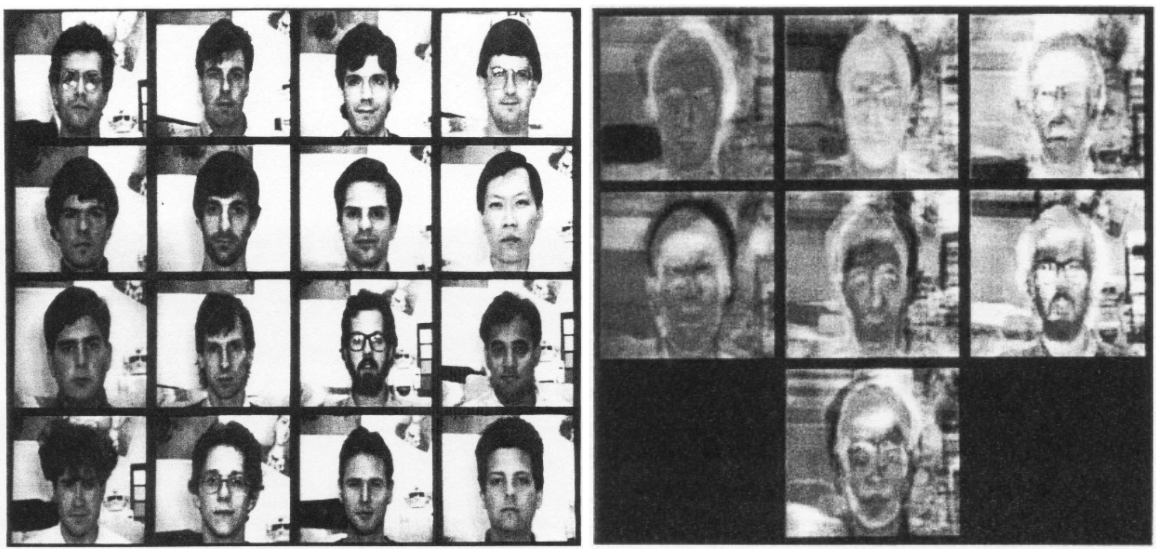


Figure 2-2: Eigenfaces (Turk and Pentland [1991]). The dataset (left) is used to compute the principal components (right): new faces are projected on the PCA subspace, and the resulting coordinates are used for recognition.

**Appearance models.** As the growing power of computers allows for more realistic images, the community becomes interested in more complicated three-dimensional objects, for which the variability in appearance becomes too complicated for mathematical study. Appearance-based methods start to emerge in the '80s, building object

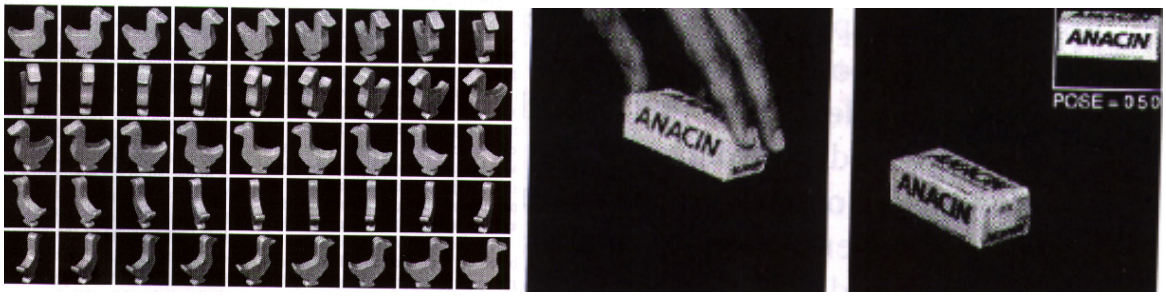



Figure 2-3: Empirical models of image variability, by Murase and Nayar [1995]. They introduce pose invariance in recognition by capturing multiple rotated views of objects. At run-time they recognize objects and the closest rotation angle for that view.

models from many image samples (image datasets). Early work in this direction has focused on face recognition. Sirovich and Kirby [1987] propose to decompose face images onto a low-dimensional space, applying Principal Component Analysis on a dataset of faces. Turk and Pentland [1991] improve on this approach by introducing Eigenfaces in 1991, a near-real-time system that can recognize faces using an Euclidean distance in the PCA space, see Figure 2-2. Murase and Nayar [1995] extend this approach to represent objects. They build a dataset of objects in different poses (rotated by different angles) on a uniform background. Using a PCA, as in Eigenfaces, allows them to recognize objects and poses simultaneously, effectively implementing pose invariance for objects as shown in Figure 2-3.

**Histograms and vocabularies.** The work on appearance models sets a trend for object recognition and in this context , Swain and Ballard [1991] propose associating objects to a *histogram* of their colors, and introduce a comparison operator called *histogram intersection*. This allows recognizing instances of objects robustly, as the distribution of colors remains unaffected by small viewpoint changes. But recognizing colors is difficult because of illumination changes, and Schiele and Crowley [1996] extend this histogram approach by considering the responses of an image to convolution filters (such as Gabor filters or Gaussian derivatives). This allows recognition based on the distribution of local features such as edges. Leung and Malik [2001] adopt a similar approach for recognizing textures, by considering histograms of local features called *3D textons*. These textons are built by concatenating the responses of

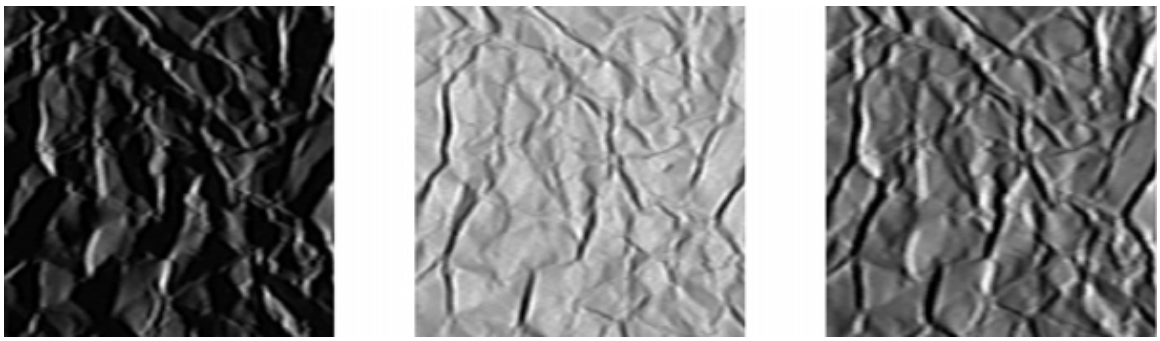


Figure 2-4: Example of the material "Crumpled paper"; Leung and Malik [2001] build a vocabulary of *3D textons* learned on patches of materials under different lighting conditions, inducing robustness to illumination changes.

textures in various lighting conditions (as shown in Figure 2-4), and clustering them using the K-means algorithm. These clusters define a *vocabulary*, that encodes the local geometric and photometric features of the material and allows building a setup robust to illumination changes. As an important property, this vocabulary of textons remains valid when used with unseen textures and materials.

**Object matching and local image descriptors.** In parallel, one of the tasks that led to many improvements in object recognition was image retrieval and matching. While the *eigen* line of work focused on global image representations, the work of Schmid and Mohr [1997] (Figure 2-5) popularized the idea of representing images with local image descriptors based on image gradients, as described by Koenderink and van Doorn [1987].

The evidence of the success of local descriptors for object recognition and image matching largely inspired the work of Lowe [1999] [1], leading to the highly influential SIFT descriptor (Scale Invariant Feature Transform, Lowe [1999]), shown in Figure 2-6. While computer vision methods have routinely used SIFT for visual recognition and object classification in the following decade, one can keep in mind that the initial purpose of this descriptor was to perform matching between different views of a given object or scene.

[1] Schmid and Mohr [1997] is the only reference in the introduction

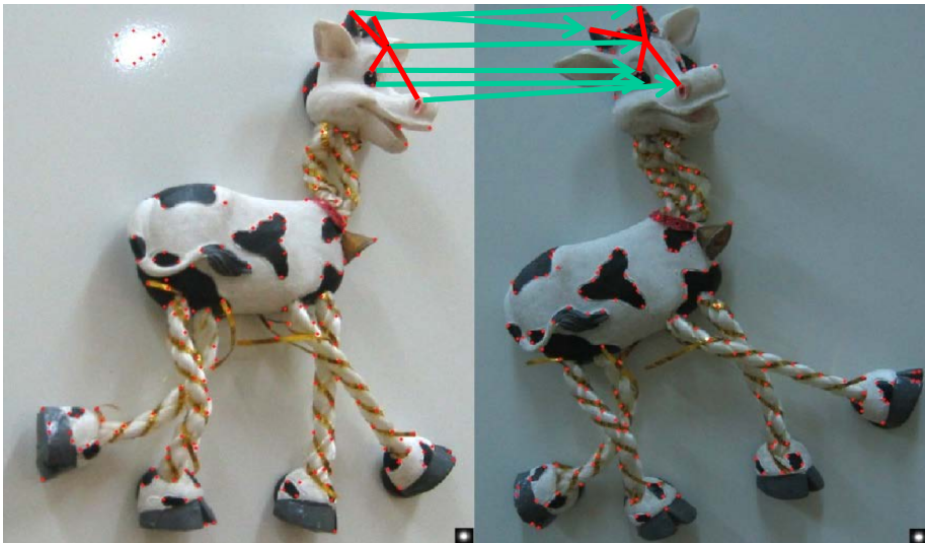


Figure 2-5: Schmid and Mohr [1997] propose to describe images using *local descriptors*, contrasting with global approches such as Eigenfaces. Considering local descriptors allows matching objects in different imaging conditions.

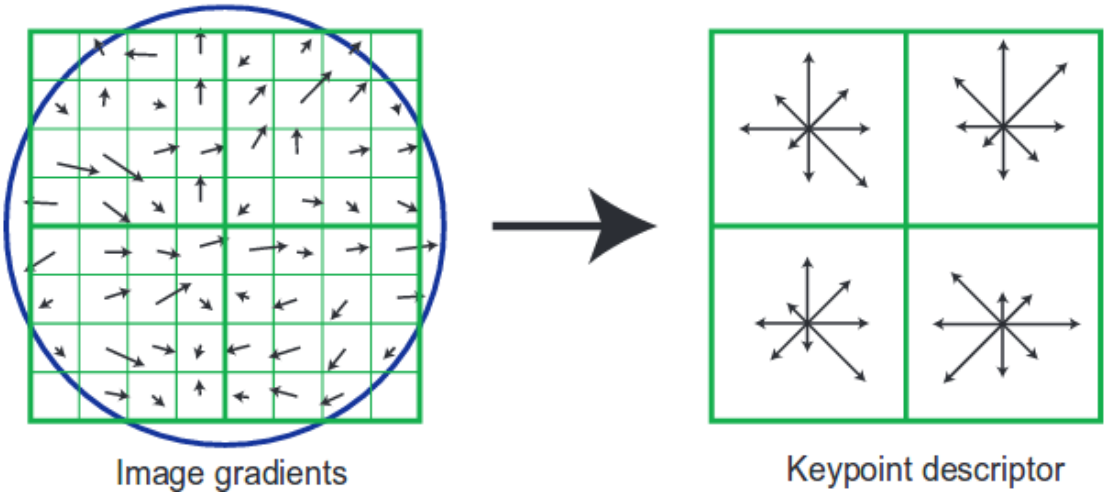



Figure 2-6: Given an input area, SIFT (Lowe [2004b]) computes gradients locally (left) then aggregates the information in a 128-bit descriptor (right). Aggregating the gradients from 16 cells provides robustness to small translations.

**Statistical representations and bagging.** With the success of SIFT in recognizing object instances, visual recognition entered a period of rapid progress. One important addition to the paradigm of visual recognition was the introduction of *bags* in statistical approaches. Sivic and Zisserman [2003], inspired by the success of the Google algorithm for text retrieval, propose an object matching method in videos by

Figure 2-7: Video Google (Sivic and Zisserman [2003]). First row: input image and selected query region. Second row: retrieved shot and region used for retrieval. Bottom: examples of patches for two visual words of the vocabulary. Video Google describes the selected region as a quantized orderless histogram of visual words, allowing efficient retrieval.
Demo : http://www.robots.ox.ac.uk/~vgg/research/vgoogle/

describing (groups of) frames as a *bag-of-visual-words* (shown in Figure 2-7). This approach consists of extracting SIFT descriptors in movie frames, then clustering them to build a vocabulary of *visual words*, similar to Leung and Malik [2001]. The visual words notably carry the invariance of SIFT to small shifts in position (see Figure

2-7, bottom). The frames are then described as a *bag* of these words, quantized in a histogram and used by a text retrieval algorithm to match relevant frames and video shots.

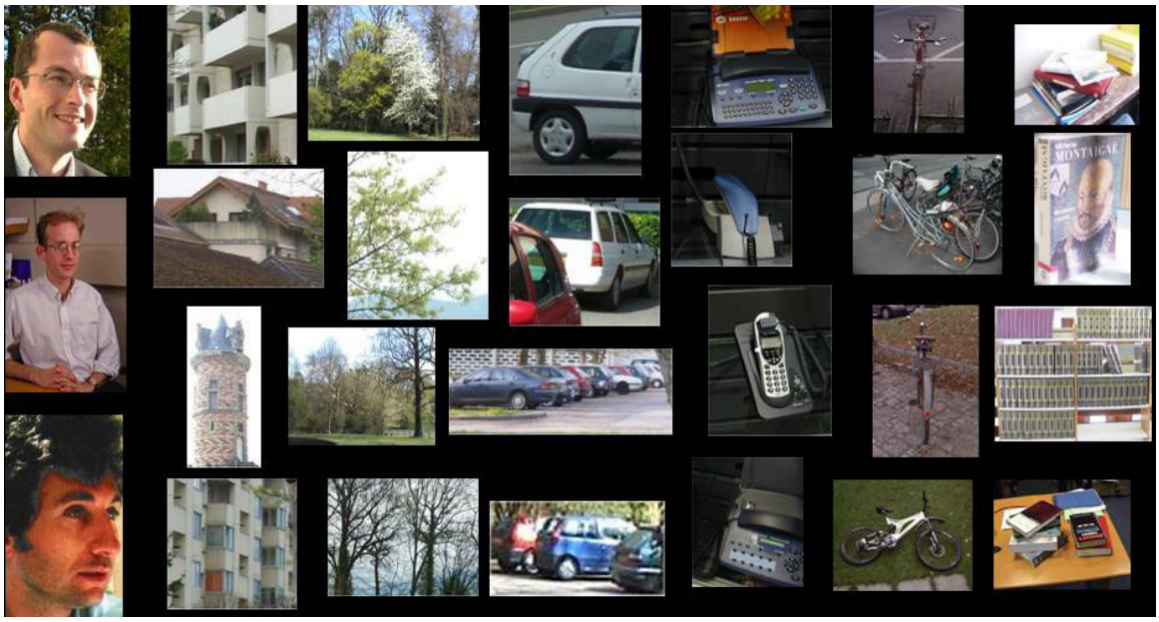


Figure 2-8: Dataset collected for visual categorization with 7 classes (Csurka et al. [2004]). This method combines the bag-of-visual-words approach with SVM classifiers to perform visual recognition.

**From instance-level to category-level recognition.** The location-agnostic bagging approach shows that recognition may succeed even if spatial information is lost. Csurka et al. [2004] adopt a similar approch with the *bags of keypoints*: constructing a vocabulary of patch descriptors extracted at interest points using the Harris Affine method (Mikolajczyk and Schmid [2002]) on a dataset of labeled images, describing images as a histogram of these words then classifying them into their corresponding classes using the SVM (Support Vector Machine, Schölkopf and Smola [2002]) algorithm. This method made visual categorization popular and robust by treating *statistically* concepts defined within a dataset only by examples, with large variations in viewpoints (Figure 2-8). This corresponds to a crucial step in computer vision, shifting the focus from instance-level recognition (recognizing a specific object) to category-level recognition (recognizing the nature of an object).

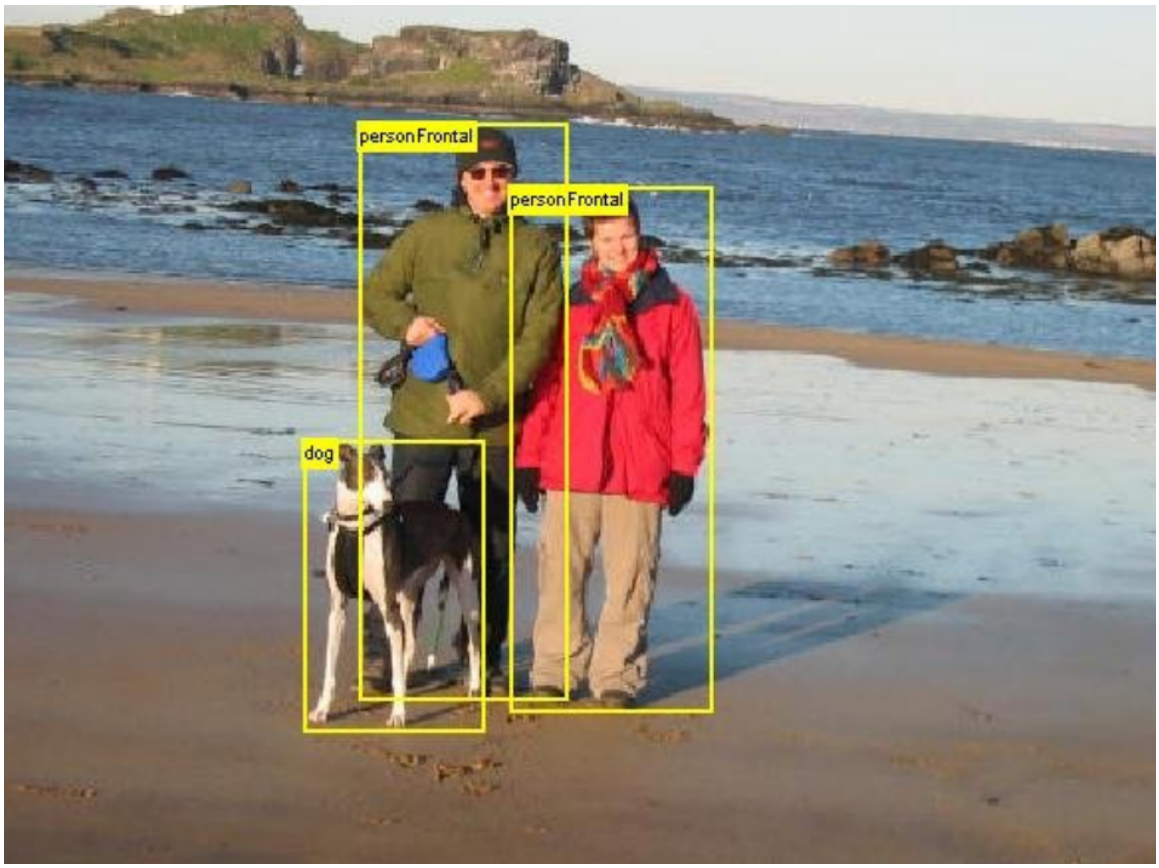



Figure 2-9: Example image with the Pascal VOC dataset (Everingham et al. [2010]), with *bounding boxes* localizing objects, popularizing the task of *object detection*: recognizing an object class and predicting its extent.

Around the same period, in 2005 the Pascal VOC challenge (Everingham et al. [2010]) appears along with its corresponding dataset, driving the research in computer vision towards better image classification and object detection (localizing objects and their extents) for the following decade, with a realistic and challenging natural image dataset updated every year to keep the difficulty high (example shown in Figure 2-9).

Building on the work of Grauman and Darrell [2005] on *pyramid matching*, Lazebnik et al. [2006] improve *bag-of-features representations* with spatial information, by applying the histogram computations to a pyramid of image regions (shown in Figure 2-10) at different scales, effectively introducing spatial cues in the pipeline, bridging the gap with global image representations.

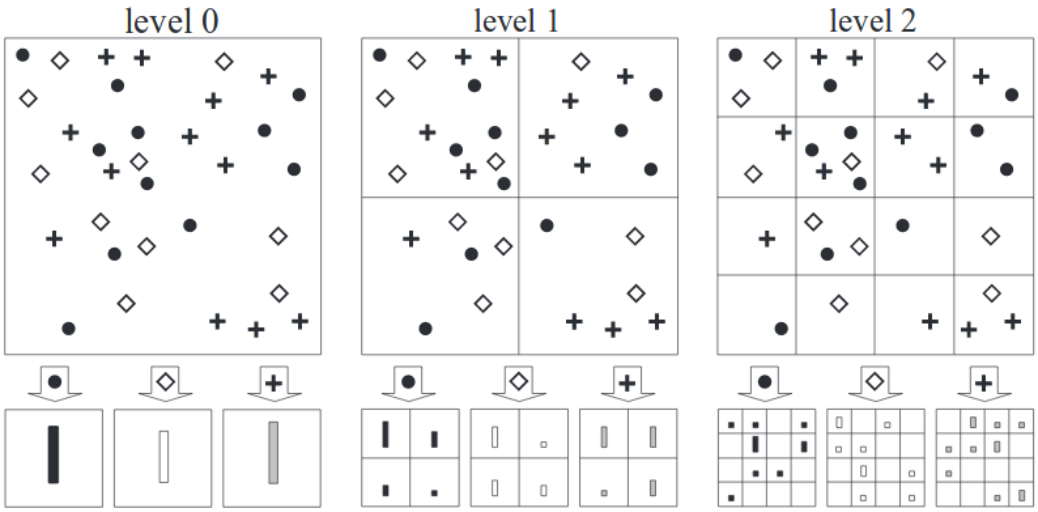



Figure 2-10: Spatial pyramids. Lazebnik et al. [2006] propose a pyramidal approach to histogram binning, introducing location-awareness and multi-scale processing in the bag-of-features pipeline.

**Structured models and constellations of parts.** Another class of approaches, related to the work on *pictorial representations* by Fischler and Elschlager [1973], consists of describing objects as a constellation of parts with constraints between them, as shown in Figure 2-11. By reasoning at the level of object parts, these methods have potential for much more precise localization of the objects that are recognized, and this idea proved fruitful. Marr and Nishihara [1978] propose to represent complex shapes as a hierarchy of geometric primitives such as cylinders, as shown in Figure 2-12. Ioffe and Forsyth [2001] build on a similar idea but adopting a bottom-up approach, by first detecting candidate *segments* corresponding to human limbs, and then considering the underlying kinematic constraints to assemble them using a human body model to perform person detection in images, as shown in Figure 2-13. Fergus et al. [2003b] propose an object model based on a *constellation* of parts, with built-in tolerance to deformations. This approach recognizes parts of objects then connects them together before evaluating them against a learned graph model of the classes of interest, as shown in Figure 2-14; this approach enables category-level recognition and localization of objects.

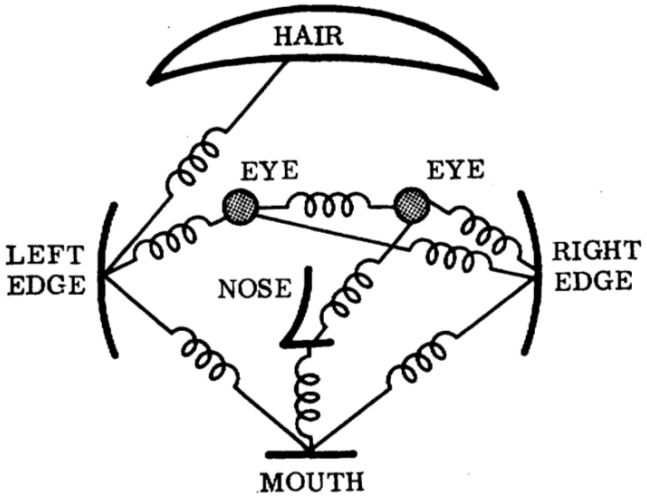



Figure 2-11: Example description of a face. Fischler and Elschlager [1973] propose representing objects as a structure, by assembling parts and connecting them in pairs with a set of "springs".

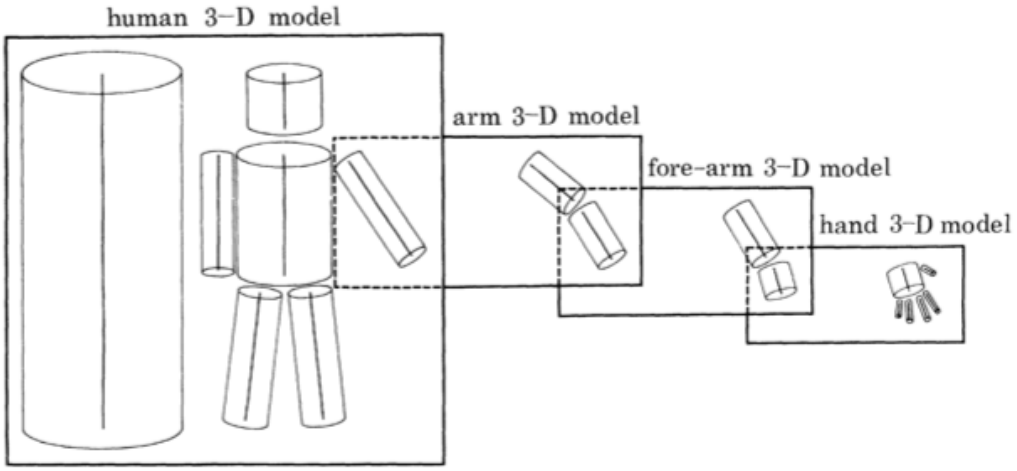



Figure 2-12: Marr and Nishihara [1978], inspired by human vision, propose to decompose objects in a hierarchy of geometric primitives.

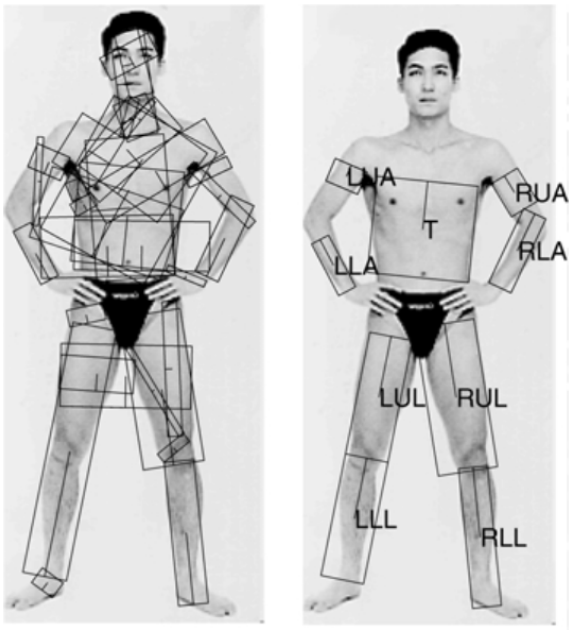



Figure 2-13: Ioffe and Forsyth [2001] propose detecting limbs as a first step before applying a kinematic model for person detection.

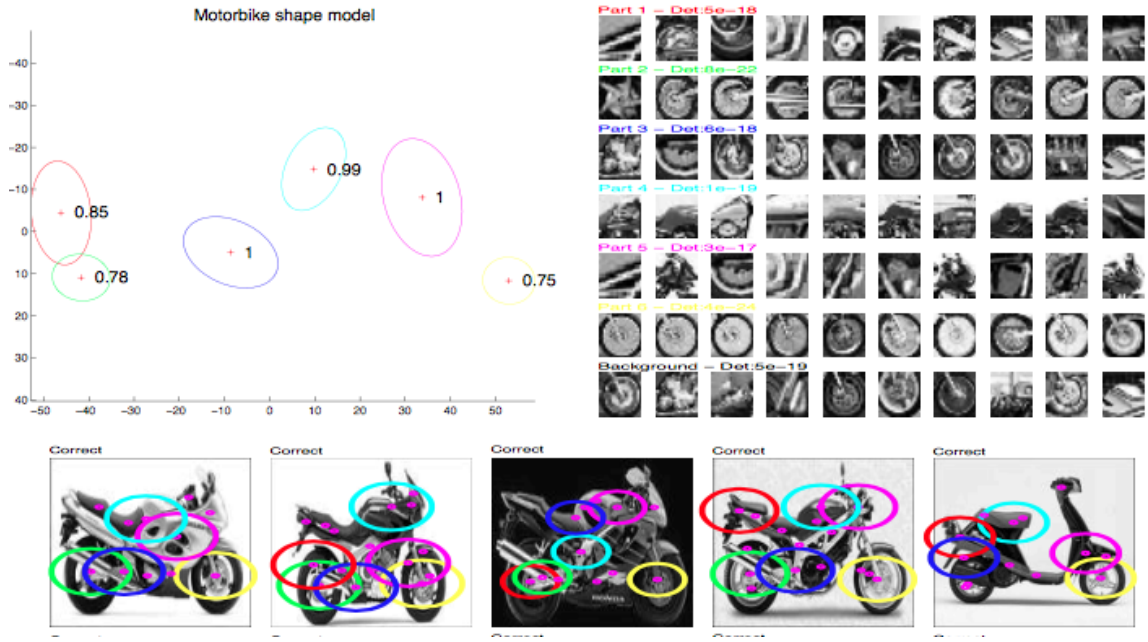



Figure 2-14: Fergus et al. [2003b] introduce *constellations* part models, recognizing objects by detecting visual words (top-right) then evaluating their relative positions against a learned shape model (top-left). Examples shown on the bottom.

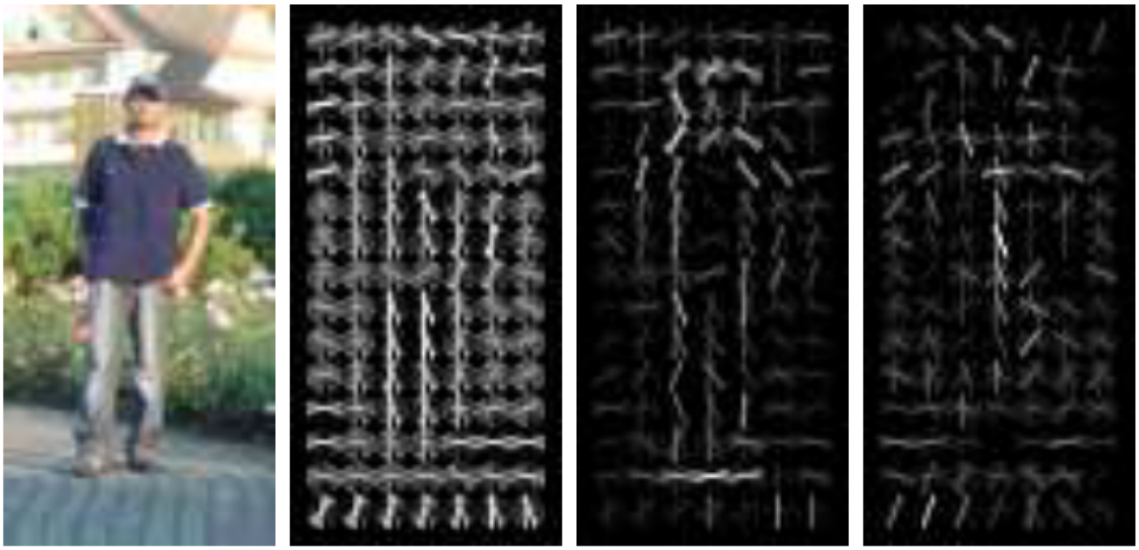


Figure 2-15: The Dalal & Triggs person detector (Dalal and Triggs [2005]), based on *Histograms of Gradients*. From left to right : (i) input image, (ii) image gradients computed within a grid, (iii) gradients reweighted by the SVM positive weights, (iv) by negative weights. The SVM weights effectively contain a person template for detection.

**Object detectors.** With the progress of visual recognition, object detection (recognizing an object category and predicting its extent with a bounding box, as shown in Figure 2-9) gains popularity, and one influential algorithm is the person detector of Dalal and Triggs [2005]; this method first collects gradient information within cells of a grid applied on an image at multiple scales using *Histograms of Gradients*, then matches these extracted features against a learned person model, represented by the SVM weights as shown in Figure 2-15.

Then, combining these Histograms of Gradients features with the constellation models described above and the spatial pyramid approach of Lazebnik et al. [2006], Felzenszwalb et al. [2008] introduce the *deformable part-models* (DPM, see Figure 2-16) using a new training method for SVM algorithms (the *Latent SVM*). This model is improved and evaluated in a follow-up paper (Felzenszwalb et al. [2010], Figure 2-17) with more flexible deformation models, setting the state of the art for object detection on the Pascal VOC challenge.

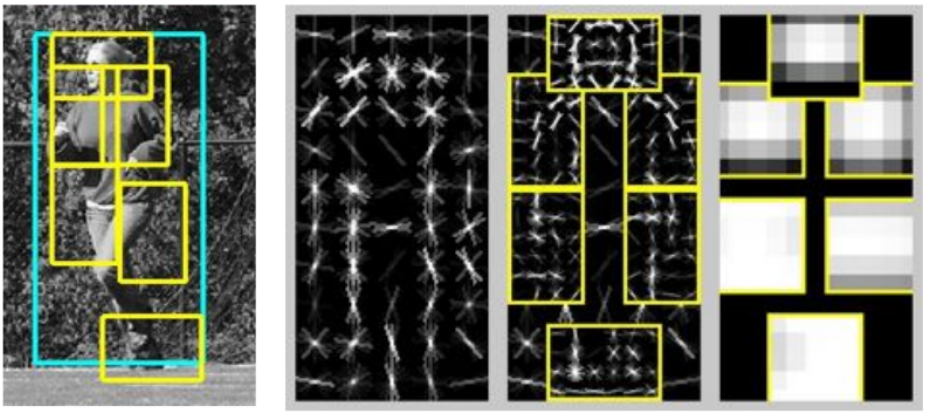


Figure 2-16: Original caption: *Example detection obtained with the person model. The model is defined by a coarse template, several higher resolution part templates and a spatial model for the location of each part.* (Figure and caption from the original paper Felzenszwalb et al. [2008].)
The *Deformable Part-Model* approach combines the constellation model with Histograms of Gradients features.

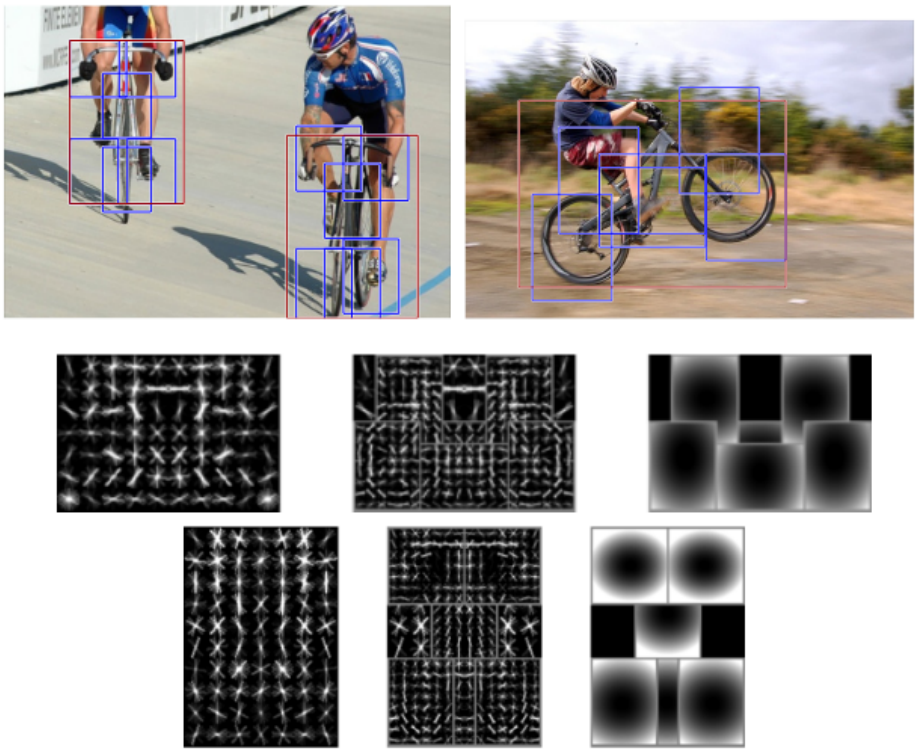


Figure 2-17: Original caption: *Detections obtained with a 2 component bicycle model. These examples illustrate the importance of deformations mixture models. In this model the first component captures sideways views of bicycles while the secondcomponent captures frontal and near frontal views. The sideways component can deform to match a "wheelie".* (Figure and caption from the original paper Felzenszwalb et al. [2010].)

**Large scale recognition.** With the growing computational power and amount of images on the Internet, interest emerges for recognition on a large scale. The ImageNet dataset (Russakovsky et al. [2015]) appears in 2009 with more exploitable data than ever available before for learning. In 2011, ImageNet contains 14 million labeled images corresponding to more than 21 thousand classes. The proposed challenge consists of performing image classification using a dataset of a million images from 1000 classes with varying levels of granularity (notably 120 different breeds of dogs).

With this new challenge, computer vision adapts new methods. Perronnin et al. [2010] introduce new techniques for describing features, in order to cope with large sets of images, using Fisher Kernels in a Gaussian Mixture Model framework with computationally efficient linear classifiers. Fisher Vectors were involved in many of best-performing state-of-the-art methods, in the ImageNet Challenge of 2011[2].

With the perspective of larger datasets, research in object detection witnessed the introduction of *objectness* methods for candidate windows: building on different image cues (e.g. color contrast or edge density) Alexe et al. [2010] (further improved in Alexe et al. [2012]) propose a generic objectness measure describing how likely it is for a window to contain an object rather than background or a small parts of objects, as shown in Figure 2-18. Following the objectness approach, van de Sande et al. [2011] propose *Selective Search* window proposals (see Figure 2-19) to build a state-of-the-art object detector.

These methods propose a small set of candidate bounding boxes for object detection; in contrast, a dense sliding-window approach would propose a much larger set of boxes. As a result, using more refined feature transforms becomes affordable on large-scale datasets, as there are fewer candidates to process. This approach is used in the more recent R-CNN object detector to be described in Section 2.3.

**Conclusion.** This section presents the history and some of the key paradigms in visual recognition. Starting with three-dimensional simple shapes, the community progressively introduced image datasets to learn models of visual appearance. Meth-

[2] http://image-net.org/challenges/LSVRC/2011/results

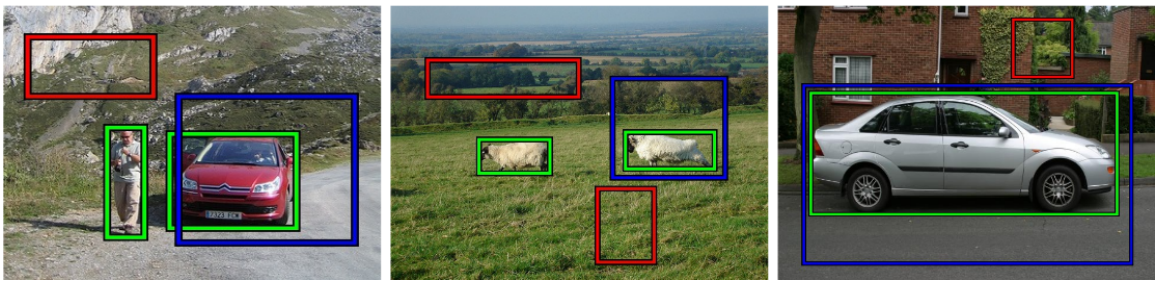


Figure 2-18: The objectness approach proposed by Alexe et al. [2010]. Original caption: *Desired behavior of an objectness measure. The desired objectness measure should score the blue windows, partially covering the objects, lower than the ground truth windows (green), and score even lower the red windows containing only stuff or small parts of objects.*

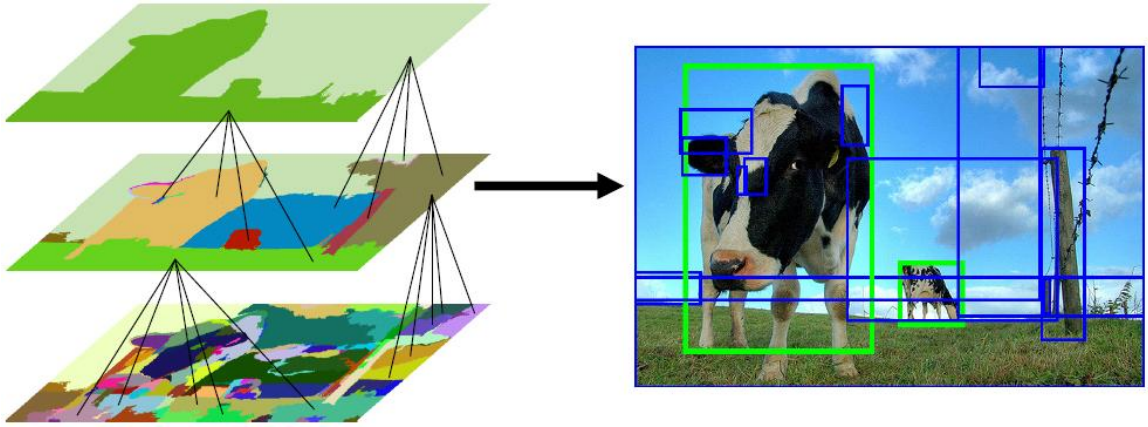


Figure 2-19: Selective search (van de Sande et al. [2011]): pixels in images are initially grouped together with the algorithm of Felzenszwalb and Huttenlocher [2004], then grouped hierarchically using a data-driven approach before proposing candidate object bounding boxes.

ods based on local appearance led to the introduction of local descriptors for matching. With the advent of statistical methods and classifiers using bags of these local features, visual recognition became operational. Refinements were then added with templates, part-based models and object proposals, pushing recognition to natural images at large scale.

Figure 2-20: Frank Rosenblatt with the huge *Perceptron* (Rosenblatt [1957]), one of the earliest machine vision systems in history.

## 2.2 Early days in neural networks

**Hebb's rule.** The history of Artificial Neural Networks goes back to 1949. Hebb, neuropsychologist, studies the human *biological brain* and connects it to the idea of the *mind* in his book *The Organization of Behavior* (Hebb [1949]). He popularizes the *Hebb's rule : neurons that fire together, wire together*, paving the way towards complex arrangements of neurons, which are the neural networks of today.

**Perceptron.** Rosenblatt [1957] builds what is known as one of the first computer vision systems: the Mark I *Perceptron* (shown in Figure 2-20). It can be described as a multi-layer network composed of an input layer composed of 20x20 photosensitive units, an association layer composed of 512 linear threshold units with fixed weights, and an output layer composed of 8 linear threshold units with adaptable weights implemented with stepping motors and trained with the Perceptron algorithm. The Perceptron algorithm allows training a binary classifier by adjusting the weights until finding a suitable hyperplane. This algorithm has been shown to converge on linearly separable data.

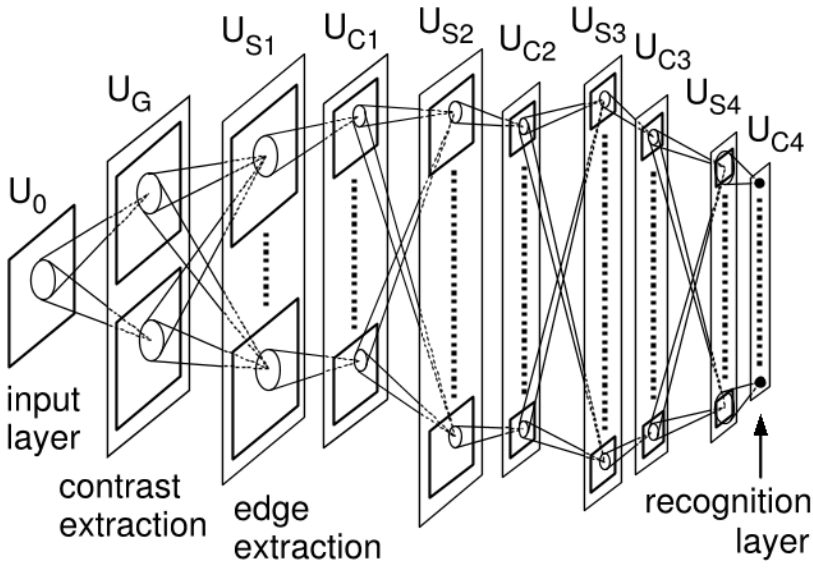



Figure 2-21: The *Neocognitron* architecture (Fukushima [1980]) introduces shift-invariance into neural networks together with pooling layers (as "complex units" $U_C$). This multi-layered architecture is very close to the modern neural network architectures that we use today.

**Neocognitron.** Inspired by the neural connectivity pattern discovered by Hubel and Wiesel [1959] in the cat's visual system, Fukushima's *Neocognitron* (Fukushima [1980], see Figure 2-21) replaces the association layer by alternating layers of simple and complex threshold units. The simple layers are trained using an unsupervised procedure related to clustering, the complex layers are fixed pooling transforms, and the whole architecture is motivated by its ability to compute features that are robust to changes in the position of the object in the receptive field. As a result, groups of S-cells in simple layers, arranged in two-dimensional arrays (*cell-planes*) respond to the same stimulus, while the C-cells aggregate the responses from neighboring S-cells in a plane.

The Neocognitron effectively implements *shift invariance*, which is known to be, today, an important property of vision systems: be it on the top-left or bottom-right of an image, a given object is still the same object. Similarly, the pooling transforms in the complex layers implement robustness to small deformations and are widely used in modern neural networks.

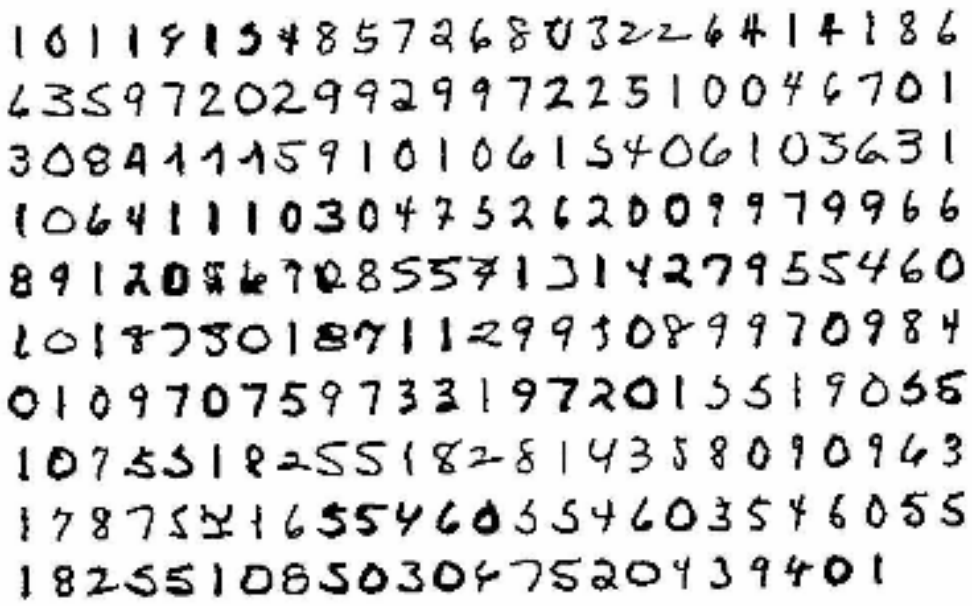


Figure 2-22: Normalized handwritten digits. This dataset, introduced in LeCun et al. [1989] with the convolution layer, corresponds to the first successful application of convolutional neural networks.

**Backprop.** Neocognitron was lacking a supervised learning algorithm, and therefore could not be used for a specific purpose. However, a couple years after, Rumelhart et al. [1986] introduce the *backpropagation* algorithm, which is a generalization of the chain rule for differentiating functions, making gradient computation possible in multi-layered systems. Using backprop, Lang and Hinton [1988] propose speech recognition using a translation-invariant neural network that convolves a set of weight patterns with the contents of a sliding window on one-dimensional acoustic signal.

**Convnets.** LeCun et al. [1989] extend the work of Lang and Hinton [1988] to the two-dimensional domain with the *multi-convolution layer* (see Figure 2-23), building on the *weight sharing* scheme of the S-cell planes of Fukushima's Neocognitron. This method quickly achieves excellent results in handwritten digit recognition (see Figure 2-22). Though processing two-dimensional signals (as images) with backprop and weight sharing is mentioned in Rumelhart et al. [1986], the digit recognition problem is the first successful and convincing incarnation of this idea. The setup is run on SN (*Simulateur de Neurones*, Bottou and LeCun [1988]), one of the earliest pieces

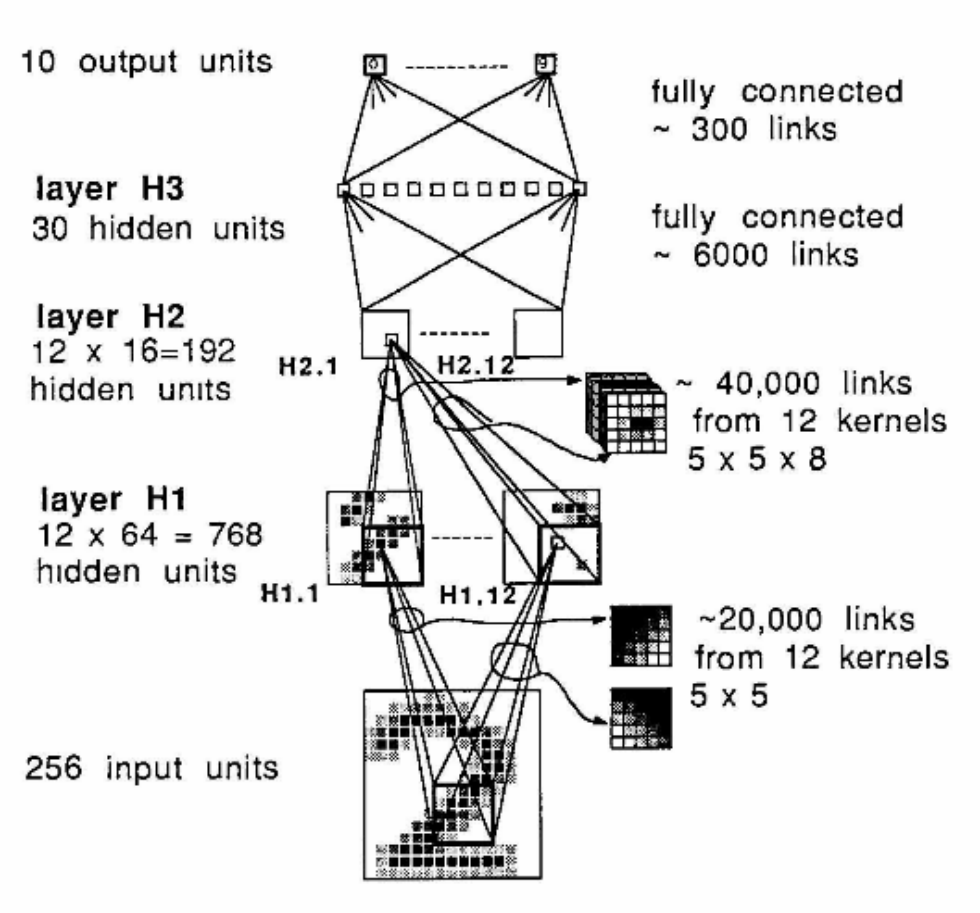



Figure 2-23: A digit recognition architecture using convolution layers, as proposed in LeCun et al. [1989]. The convolution kernels on the bottom (H1) allow for extracting edges in the initial image. The multi-convolution kernels (H2) find more complex patterns.

of software for neural networks, written in C. The operational nature of neural networks makes them engineering-intensive, and SN set a high standard in this domain, providing graphical output and a formalism close to those of the most popular recent frameworks.

**Stochastic gradient descent.** The speech and digit recognition networks mentioned above are trained by performing gradient descent on the error function. Given the high cost of computing the gradient with backpropagation on the whole dataset, Lang and Hinton [1988] propose an alternative where the weights of the networks are updated after presenting each pattern, "departing to some extent from a true gradient

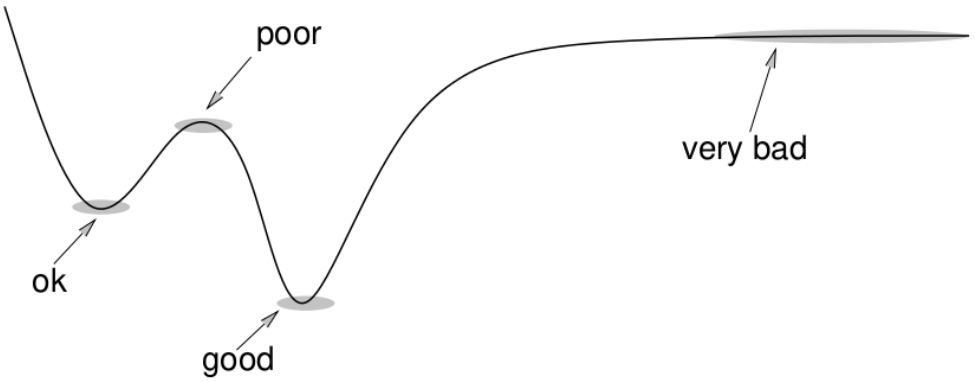



Figure 2-24: Bottou [1991] (updated in Bottou [1998]) provides a convergence proof of SGD towards extremal points (shown here), notably showing the issue of poor solutions like saddle points and asymptotic plateaus.

descent". This procedure, although not named as such in the paper, corresponds to *stochastic gradient descent* (SGD). The corresponding theory is formalized in 1991, and Bottou [1991] (updated in Bottou [1998]) provides a proof for the convergence of SGD towards extremal points (see Figure 2-24) in non-convex cases. The result of training with SGD depends on the initialization and the order in which the examples are presented.

**Graph Transformer networks and general architectures.** In 1997, neural networks have successful industrial applications in handwritten character recognition, and they are used for automatically reading checks in the USA with a system described in LeCun et al. [1997]. The *Graph Transformer Networks* (GTN) approach consists of building a whole system with arbitrarily complicated parameterized modules, as long as they are differentiable. We illustrate in Figure 2-25 the GTN architecture proposed for recognizing digit strings in checks, combining: a segmenter, a trainable neural network that recognizes digits, and graph-processing differentiable modules that allow propagating gradient to the neural network weights. With this type of approach, a complex heterogeneous system can be trained end-to-end with gradient descent methods.

In the GTN approach the input is first over-segmented, leading to a segmentation graph $G_{seg}$ where the graph edges correspond to possible character classes. These

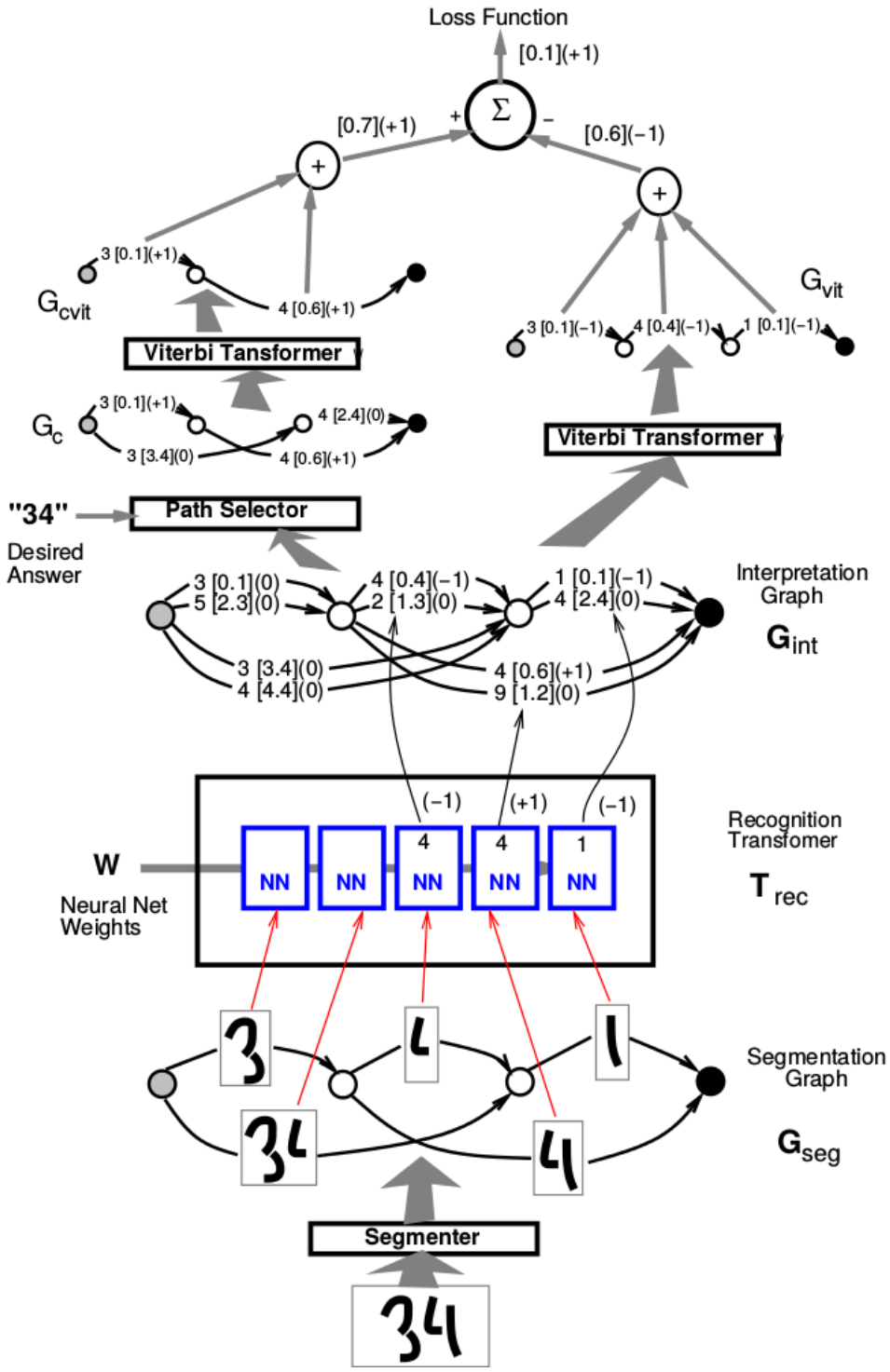



Figure 2-25: Original caption: *Discriminative Viterbi Training GTN Architecture for a character string recognizer based on Heuristic Over-Sergmentation. Quantities in square brackets are penalties computed during the forward propagation. Quantities in parentheses are partial derivatives computed during the backward propagation.*
Caption and Figure from LeCun et al. [1998a]. The Graph Transformer Networks approach allows for training arbitrarily complex systems in an end-to-end approach as long as the modules are differentiable, generalizing gradient-based learning.

candidates are then processed by a trainable neural network $T_{rec}$, returning a score for each class, building the interpretation graph $G_{int}$. Then, the left branch uses ground truth labels to keep only the correct paths in the constrained graph $G_c$, and searches the shortest path (smallest penalty) with the Viterbi algorithm, building $G_{cvit}$. In parallel, the right branch searches the shortest path from $G_{int}$ directly without the path selector supervision, and builds $G_{vit}$. The difference in penalties of the shortest paths returned in $G_{cvit}$ and $G_{vit}$ are then compared to obtain the value of the loss.

During training, the gradient is propagated through the active edges of these graphs. The left branch propagates positive gradient to correct candidates through the shortest path of the constrained graph $G_{cvit}$, and the right branch propagates negative gradient through the shortest path of the unconstrained graph $G_{vit}$. As a result of this procedure, the recognition network $T_{rec}$ receives gradient for learning. The left branch ensures that the correct candidates are learned, while the right branch penalizes all candidates. If a candidate is present in both branches (in $G_{cvit}$ and $G_{vit}$) then it does not receive learning signal as the contributions sum to zero.

At test time, we obtain the output of the system from the right branch. This GTN architecture shows that it is possible to train a neural network component inside a complex system of differentiable modules. We also note that the supervision for this setup is provided in an intermediate step, through the path selector module, and not directly by the loss function.

**Vanishing gradient and LSTM.** *Recurrent neural networks*, mentioned in Lang and Hinton [1988], are a special form of neural networks where *recurrent units* are connected to themselves, making them especially useful for learning models for sequential data. The presence of a *hidden state* reminds of Hidden Markov Models. Bengio et al. [1994] notice that, in longer sequences, neural networks have difficulties storing ("latching") information because of the *vanishing gradient problem*. This phenomenon, although common when processing data through many stages in an architecture, is even more important in the recurrent case where data can be processed

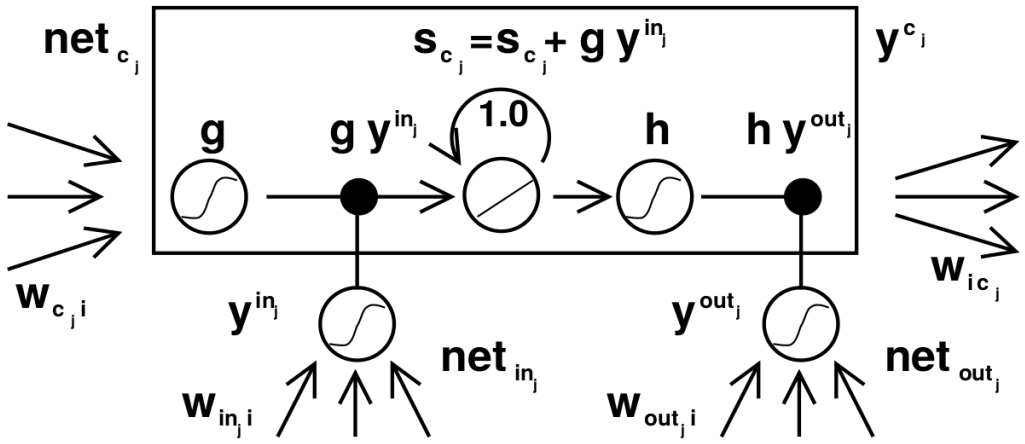



Figure 2-26: The original LSTM recurrent architecture as proposed in Hochreiter and Schmidhuber [1997]. The LSTM uses a gating approach to update or output the relevant contents of the hidden state (*the Constant Error Carrousel*) in the center.

an arbitrary number of times through small-slope non-linear functions such as the hyperbolic tangent, leading to small-magnitude gradients.

In order to solve this issue, Hochreiter and Schmidhuber [1997] propose Long-Short Term Memory (LSTM), a recurrent neural network architecture that learns when it should store and forget the data that it stores in its hidden state, as shown in Figure 2-26.

**Maturity.** The procedures are studied well, common pitfalls are listed, and tricks are proposed in LeCun et al. [1998b], which is very close to a user manual for training neural networks, containing many pieces of advice still useful today. One can notably observe the similarity between the modern AlexNet architecture and the LeNet-5 architecture in Figure 2-32. **In the end of the '90s, neural network methods are essentially mature**: they can be used as soon as there is enough data and processing power. However, at that time the data-hungry nature of neural networks limited their usage in computer vision, and the community obtained generally better performance on visual recognition with pipelines relying on the hand-made SIFT descriptor features and convex SVM classifiers allowing for easily reproducible results.

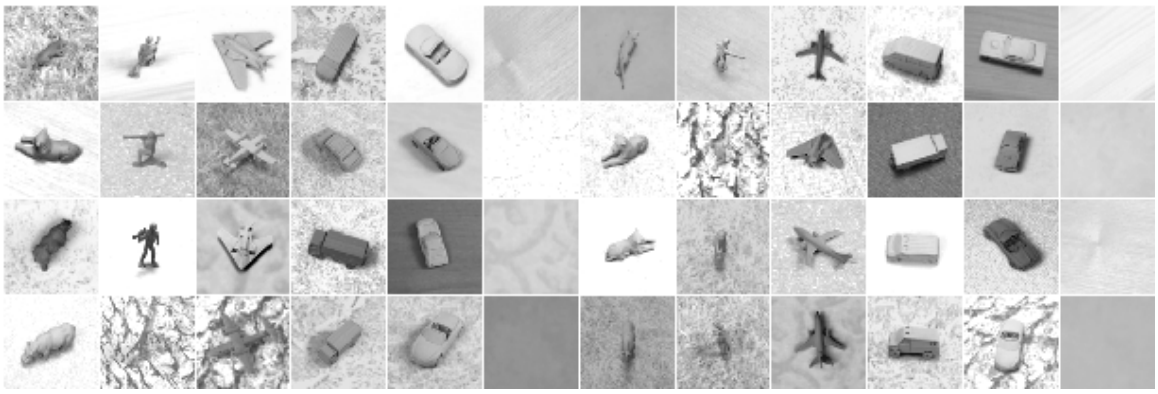


Figure 2-27: The NYU Object Recognition Benchmark (NORB) dataset introduced in LeCun et al. [2004], proposes a large-scale dataset of objects on different backgrounds. With few classes and many examples per class, this is a very favorable case for neural networks, and the unbeatable baseline did not allow the vision community to compete, explaining the little popularity of this dataset.

**"Winter" of neural networks.** The NORB dataset (LeCun et al. [2004], see Figure 2-27) is introduced in 2004, offering a large-scale dataset for visual recognition. With few classes and many examples, it is a very favorable case for neural networks. Given its difficulty, the strong baseline, and the introduction of the Caltech101 (in 2004, Fei-Fei et al. [2007]) and Pascal VOC (in 2005, Everingham et al. [2010]) datasets focused on more realistic images, NORB did not attract the vision community, and few results were published. Working on what seemed like a different kind of problems, the 2000s were difficult times for the neural network community in the vision conferences[3].

**Feature learning.** Feature extraction seemed to be the limiting factor to the success of neural networks in vision tasks for which the datasets were small, and trails have been explored to alleviate this problem based on unsupervised feature learning, hoping to compete with the SIFT descriptor. Restricted Boltzmann Machines (Hinton and Salakhutdinov [2006]) revived interest in deep architectures by providing a *pretraining initialization* method for weights in deep neural networks, leading to initial weights being neither too large (they lead to poor local minima during optimization) nor too small (gradients cannot be propagated efficiently with backprop),

[3]Yann Le Cun reports an unfair bias against neural networks methods in a withdrawal letter from the CVPR conference in 2012 https://plus.google.com/+YannLeCunPhD/posts/gurGyczzsJ7

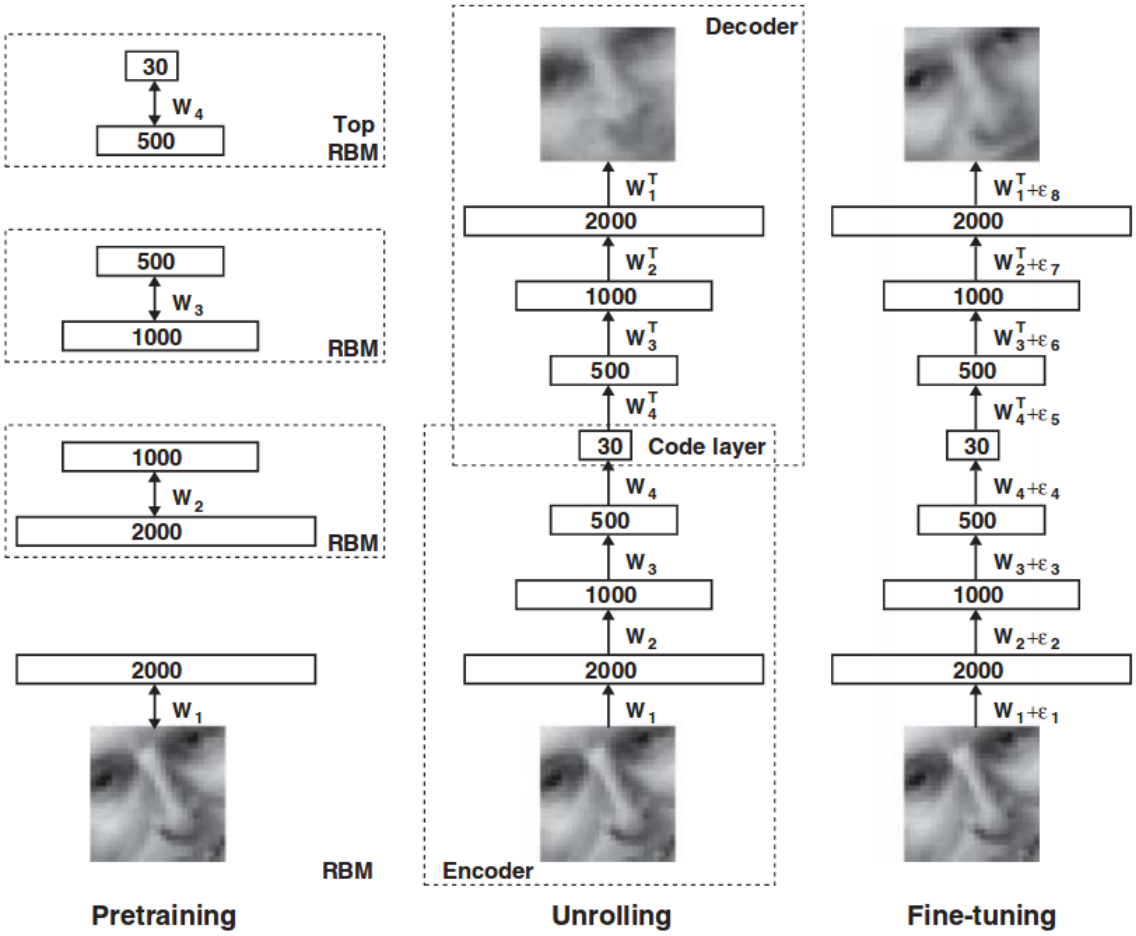



Figure 2-28: Original caption: *Pretraining consists of learning a stack of Restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.* Caption and Figure from Hinton and Salakhutdinov [2006]. The RBM approach revived interest in deep architectures by offering an initialization method for deep autoencoders.

allowing the use of deep auto-encoders (see Figure 2-28). Then, Ranzato et al. [2007a] propose to learn a hierarchy of shift-invariant features by training layers one after the other in an auto-encoder framework. The auto-encoder trail is further explored with sparsity regularizers in Ranzato et al. [2007b]. Kavukcuoglu et al. [2009] propose to pool groups of learned convolution filters according to topographic maps to build invariance explicitly (shown in Figure 2-29). But these methods struggle with stacking many of these layers and, while the results are promising, depth remains an issue.

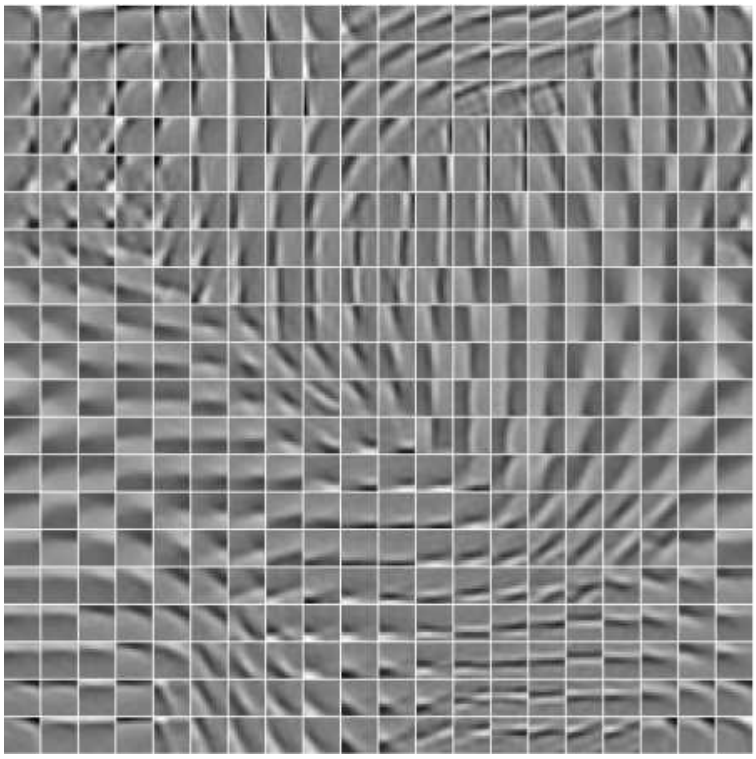


Figure 2-29: Kavukcuoglu et al. [2009] propose to learn a sparse set of feature detectors that are then arranged into a topographic map (shown here), where similar filters are pooled together to explicitly build invariance to small distortions of the input.

Figure 2-30: Le et al. [2012] train an autoencoder using a distributed system of 10000 CPUs, on frames extracted from YouTube, and observe what individual neurons react strongly to. The first author says: "*Perhaps in line with a stereotype about what goes on YouTube, we found a neuron that was highly selective to images of cat faces.*"

Later, Collobert et al. [2011b] show that an unsupervised approach for learning features (or *embeddings*) can be successful with neural networks in the case of Natural Language Processing; similarly, Le et al. [2012] manage to train a network with an unsupervised approach on large amounts of images using a distributed system of 10000 CPUs (see Figure 2-30), giving more hopes in this direction.

**ReLU.** One very important contribution during the neural network winter is the study of issues related to depth and vanishing gradient (mentioned in 1994 in Bengio et al. [1994]). Glorot and Bengio [2010] study the issue and propose an initialization scheme that allows for better gradient flow through saturating non-linearities (close to the one proposed in LeCun et al. [1998b]). Pushing the reasoning further, Glorot et al. [2011] stand back from the biologically-plausible saturating sigmoids, and introduce the unbounded ReLU non-linearity (see Figure 2-31), that greatly alleviates the vanishing gradient problem, paving the way for deeper architectures.

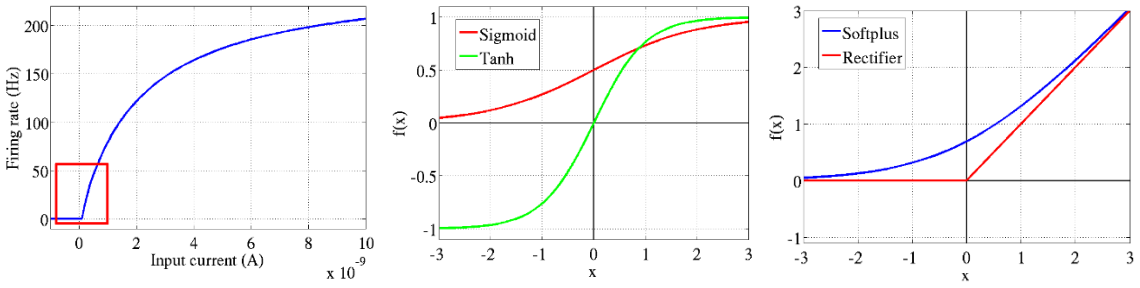



Figure 2-31: Biological justification for the Rectifier Linear Unit (ReLU), advocated in Glorot et al. [2011]. Left: biological neurons firing rate as a function of input current. We observe thresholding at the origin. Middle: saturating non-linearities popular at the time. Right: Rectifier Linear Unit and its smoothed version.

**Conclusion of this section.** In this section, we show that neural networks were extremely advanced and successful already in the '90s. Little has changed in their design when compared to the systems used today. Even though the work of Krizhevsky et al. [2012] has revived them and more improvements have happened since, we want to point out that the algorithms are mostly identical; the much larger scale of data and hardware has changed the game.

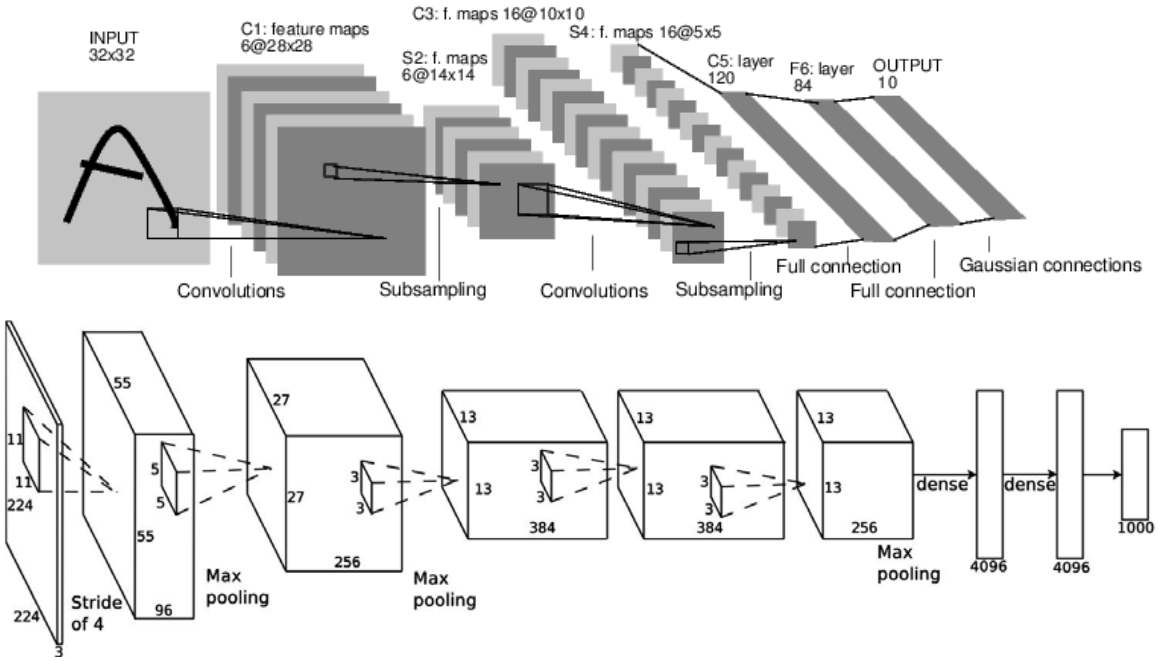



Figure 2-32: The LeNet-5 architecture (top, 4-layers deep), proposed in LeCun et al. [1998a], is very close to the more recent AlexNet (bottom, 8-layers deep) architecture, proposed in Krizhevsky et al. [2012].

## 2.3 The AlexNet breakthrough

### 2.3.1 Scaling up neural networks

**GPU convolution.** Krizhevsky [2009], explores methods to build better features using unsupervised methods and introduces the labeled CIFAR-10 dataset. It corresponds to a "natural image" version of the MNIST digits dataset introduced in LeCun et al. [1998a], with images of similar size (around $32 \times 32$), about the same number of samples (around 60000), also split in 10 classes but with RGB images (see Figure 2-33).

The standard MNIST digit classification dataset, commonly used for evaluating neural networks, was too easy, and error levels below 1% were already reached when it was introduced by LeCun et al. [1998a]. In contrast, the CIFAR-10 dataset offers a higher level of difficulty and is closer to natural images, but still allows quick experimentation on neural networks like MNIST.

In his work, A. Krizhevsky writes `cuda-convnet`, implementing the multi-convolution

Figure 2-33: Datasets commonly used for neural networks research.
Left: MNIST, introduced in LeCun et al. [1998a], 60000 training examples, 10000 testing examples, 28 × 28-pixels binary images, 10 classes.
Right: CIFAR-10, introduced in Krizhevsky [2009], 50000 training examples, 10000 testing examples, 32 × 32-pixels RGB images, 10 classes.

layer on GPUs at unprecedented speeds, around 100 times faster than CPU versions. As this type of layer accounts for the vast majority of computations in a CNN, a fast implementation is a key to process large natural images. This code, ready and public early 2012 [4], provides a crucial component to the success of CNNs: *computational power*.

**ImageNet.** As mentioned in Section 2.1, the ImageNet dataset (Russakovsky et al. [2015]) is proposed in 2009 and exceeds in size all previous manually labeled datasets. In 2011, ImageNet contains 14 million high-resolution images corresponding to more than 21 thousand classes. We show examples of these images in Figure 2-35. As the performance of neural networks tends to increase with the amount of training data, ImageNet provides a second crucial component: *large amounts of labeled data*.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is the corresponding annual competition; it consists of performing image classification using a dataset of a million images from 1000 classes. For ILSVRC-2012, Krizhevsky

[4] http://code.google.com/p/cuda-convnet/

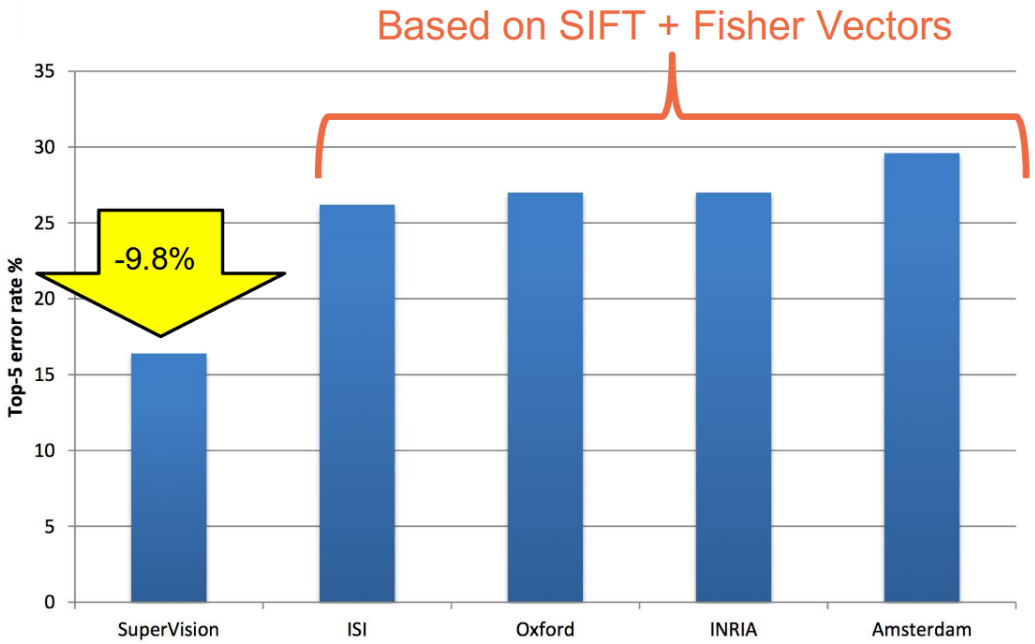



Figure 2-34: This bar graph shows the results of the 5 best performing teams on ILSVRC-2012. The methods perform classification on 1000 mutually exclusive classes, and five guesses are allowed for each test example (hence "top-5 error rate %"). We observe that the SuperVision team (Krizhevsky et al. [2012]) outperforms all others by a dramatic margin. Figure credit: Rob Fergus.

et al. [2012] propose using an 8-layer deep convolutional network later named *AlexNet* (shown in Figure 2-32) in a purely supervised manner. This network is very similar to the original neural networks of the '90s. A few important details are tuned for the problem: non-linearities are switched to ReLUs (Glorot et al. [2011]), and specific normalization layers and regularization methods (weight decay, data augmentation) are added to combat overfitting. This setup, powered by GPUs and a large amount of labeled data, outperforms all the state-of-the-art computer vision methods by a wide margin in the 2012 competition, as shown in Figure 2-34, inducing a major paradigm shift in computer vision that we discuss next.

### 2.3.2 Consequences

**Improved image descriptors.** While representations based on hand-crafted descriptors such as SIFT and HoG have been shown to work well in practice for vision, it is unclear whether they should be optimal for the task. This question raised con-

Figure 2-35: Original caption: *First column contains query images from ILSVRC-2010 test set, remaining columns contain retrieved images from training set.* Figure and caption courtesy of A. Krizhevsky.
We can observe in this figure the invariance that is encoded in the CNNs by looking at the nearest neighbors in the corresponding feature space, corresponding to the output of the second fully-connected layer FC7. One may compare this to the invariance encoded in the visual words of Sivic and Zisserman [2003] in Figure 2-7 (bottom) or Fergus et al. [2003b] in Figure 2-14 (top-right).

siderable interest in the subject of mid-level features Boureau et al. [2010], Juneja et al. [2013], Singh et al. [2012], and feature learning in general Le et al. [2011], Ren and Ramanan [2013], Taylor et al. [2010].

Since ILSVRC-2012, a large body of work (e.g. Girshick et al. [2014], Razavian et al. [2014], Jia et al. [2014]) has explored the properties of CNN features and arrived to similar conclusions: *Feature transforms learned on ImageNet by deep convolutional networks generalize well across vision tasks.* Zeiler and Fergus [2014] observe state-of-the-art performance for classification on other datasets such as Caltech-101, Caltech-256, simply by using CNN features from a similar network. Donahue et al. [2014] show similar results. Outperforming existing feature descriptors in vision pipelines

was a major step in popularizing neural networks within the vision community, and almost all visual recognition methods contain a neural network component nowadays as we will see in Section 2.4. Pre-trained CNN features have replaced the earlier SIFT and HoG features in the majority of modern vision pipelines.

To better describe the properties of CNN features, we show an example result in Figure 2-35, where $L^2$ nearest neighbors in the space of CNN features illustrate the invariance of these features to challenging variations in images.

**Relation to transfer learning.** During the course of this thesis, we have made a similar observation (see Chapter 3), and shown that the features are even more powerful when the network is trained with more data, especially if the data is related to the task we wish to solve. This let us build a state-of-the-art image classification setup on the Pascal VOC dataset.

In particular, we observed that *given the data-hungry nature of neural networks, their performance scales up with the amount of data provided for training.* This is important because once a neural-network-based system is operational, an easy way to increase its performance is simply to *collect and train with more data.*

Interestingly, building a feature transform on a task A (the *source* task) then using the learned knowledge on a task B (the *target* task) can also be seen as a form of *transfer learning*, as described in Pan and Yang [2010], to alleviate small amounts of training data. While Ahmed et al. [2008] propose this transfer in a CNN using unsupervised pseudo-tasks, other works in computer vision (Aytar and Zisserman [2011], Tommasi et al. [2010], Farhadi et al. [2009], Khosla et al. [2012], Saenko et al. [2010]) focus on classifiers rather than on descriptors.

Building on the AlexNet result, Razavian et al. [2014] advocate the usage of CNN features in an "off-the-shelf" manner, by using models readily available online to improve the performance of vision algorithms; open-source feature extractors based on CNNs (e.g. OverFeat Sermanet et al. [2014], Caffe Jia et al. [2014]) also appear along with trained models, standardizing this procedure and shifting the focus to better CNNs and CNN features.

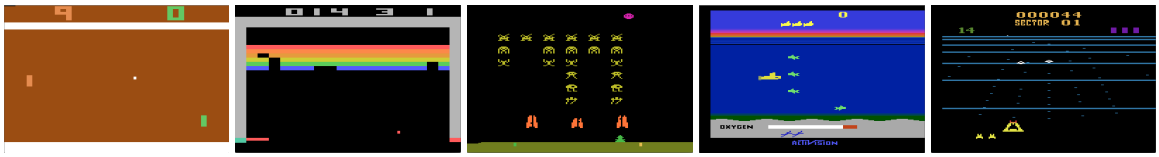


Figure 2-36: Screenshots from five Atari 2600 games used by the Deep Q-learning algorithm proposed in Mnih et al. [2013]: Pong, Breakout, Space Invaders, Seaquest, Beam Rider.

**The "AI" explosion.** With the success of neural networks on ILSVRC-2012, large investments were made by companies; we mention only three of them here. Google hired the creators of AlexNet (Krizhevsky et al. [2012]) early 2013 by acquiring their company [5]. Google uses neural networks in, among other applications, to its voice transcription engine [6] and its translation service (Wu et al. [2016]) Facebook opened FAIR (Facebook AI Research) and hired Yann Le Cun (author of the first Convolutional Neural Network recognizing digits LeCun et al. [1989]) as head of the lab late 2013 [7]. Facebook uses neural networks, among other applications, for translation [8], image description for visually-impaired users [9].

After building a setup capable of playing Atari games (see Figure 2-36), with a neural-network-based reinforcement learning algorithm (Mnih et al. [2013]), DeepMind was bought by Google early 2014 [10]. DeepMind, later, built the AlphaGo system (Silver et al. [2016]), beating world-class Go players for the first time, similar to the DeepBlue success (Campbell et al. [2002]) on the game of chess.

**Deeper architectures.** Given the crucial role of parallel computations in CNN architectures, the GPU industry (in particular nVidia, thanks to its ownership of the CUDA language for GPU programming) underwent quick progress on hardware, allowing more improvements on neural networks. Chetlur et al. [2014] notably introduce the *cuDNN* library as a standard set of subroutines for neural network computations

[5] https://www.wired.com/2013/03/google_hinton/
[6] https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html
[7] https://www.wired.com/2013/12/facebook-yann-lecun/
[8] https://code.facebook.com/posts/289921871474277
[9] https://code.facebook.com/posts/457605107772545
[10] https://techcrunch.com/2014/01/26/google-deepmind/

similar to the Basic Linear Algebra Subprograms formalism (BLAS, Lawson et al. [1979]).

More powerful GPUs and computation libraries set a trend for exploring deeper CNN architectures further decreasing error rates in the ILSVRC challenge:

- OverFeat (9 layers, Sermanet et al. [2014], 13% top-5 error)
- VGGNet (19 layers, Simonyan and Zisserman [2014b], 7% top-5 error)
- GoogLeNet (22 layers, Szegedy et al. [2015], 7% top-5 error)
- ResNet (152 layers, He et al. [2016], 4% top-5 error)

In particular, better performance on ImageNet classification leads to better CNN features, improving the performance on other visual recognition tasks.

**Democratization.** Neural networks are powerful, but they are known to be difficult to train properly, as mentioned in LeCun et al. [1998b] when discussing the backpropagation algorithm: *"getting it to work well, and sometimes to work at all, can seem more of an art than a science"*. As a consequence, along with the streak of successes starting with AlexNet, efforts were made to build tools to ease the task of training neural networks.

Among these tools, the important *batch-normalization layer* (Ioffe and Szegedy [2015]) was introduced: during mini-batch training, the output values of weighted layers are normalized with respect to the mean and standard deviation across examples in a differentiable manner; the authors report better, faster and easier training. The success and ease-of-use of this method led to its implementation in an optimized version in the cuDNN computation library.

New *adaptive learning rate* algorithms, variants of SGD, were also introduced to deal with the issue of setting learning rates properly in the networks, providing theoretical justifications in some cases (neural networks are mathematically difficult). Schaul et al. [2013] propose using second-order (Hessian) information. Tieleman and Hinton [2012] propose RMSprop, dividing the gradient by a running average of its

recent magnitude, justified by empirical evidence. In our work described in Chapter 4, we used the Adagrad algorithm proposed in Duchi et al. [2011b] that adjusts learning rates for each parameter, more effective when the gradients are sparse. Kingma and Ba [2015b] propose Adam, adjusting learning rates depending on the first-order and second-order moments of the sequence of gradients for each weight parameter in the network; we use Adam in our work in Chapter 5. With these new tools, neural networks have become easier to train, increasing their popularity further.

**Conclusion of this section.** In this section, we show that the goal of learning competitive image features, pursued by the neural network community using unsupervised methods, was finally met when the amount of data and the computational power reached a critical point in 2012 with AlexNet. Unexpectedly, these powerful CNN features were obtained with purely supervised methods, very similar to the state of the art in the '90s. Early evidence shows that they compare favorably to previous methods such as those based on SIFT, and their widespread success led to massive investments in the field, a larger community, better hardware and easier methods allowing progress towards more powerful models, effectively democratizing neural networks.

## 2.4 Integrating vision pipelines in neural networks

In this section we show the trend of integrating more complex vision systems within unified neural networks. Building heterogeneous systems is similar in spirit to the Graph Transformer Networks approach of LeCun et al. [1997] shown in Figure 2-25, Section 2.2. In contrast to the use of CNN features discussed in Section 2.3, it is possible to train end-to-end computer vision systems by expressing all processing steps as differentiable modules.

**Object detection.** Object detection is the computer vision task that consists of finding individual instances of objects in images, recognizing their classes and predicting their extents in terms of bounding boxes. Extracting the content of an image in this way should be useful for reasoning and understanding based on visual input. This task was popularized with the Pascal VOC challenge, and earlier methods solving this task, such as the *Deformable Part Models* algorithm, are described in Section 2.1.

With the appearance of more powerful feature transforms, object detection underwent a significant increase in performance with the R-CNN approach (Girshick et al. [2014], described in Figure 2-37), building on the Selective Search algorithm from van de Sande et al. [2011] (see Figure 2-19, Section 2.1). This method is further improved in a *Fast* version (Girshick [2015]), and a *Faster* version (Ren et al. [2015]) that proposes to integrate the object proposal step as a differentiable module within a more complex neural network.

In parallel to R-CNN, the OverFeat detector (Sermanet et al. [2014]) provides a different approach for object detection, by running a *sliding window classifier* densely on an image, using the convolutional properties of neural networks to share computation as explained in Figure 2-38, before feeding the result to a regressor network that provides bounding box coordinates as output.

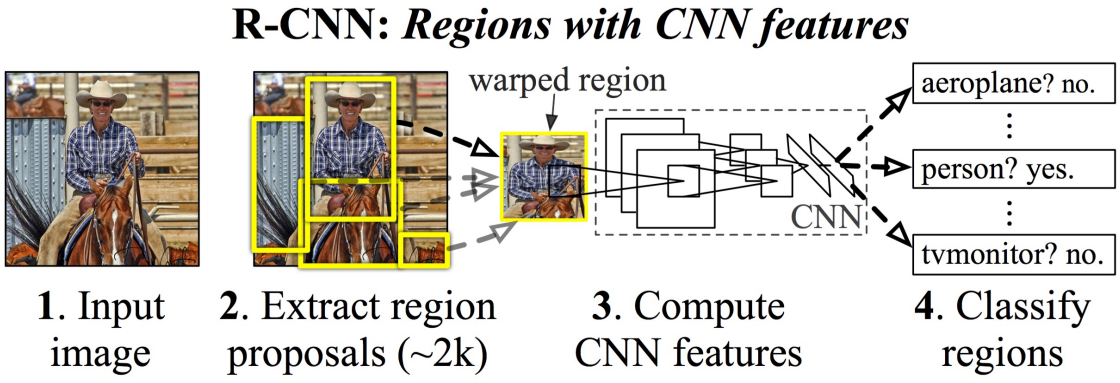



Figure 2-37: The original R-CNN (Regions with CNN features, Girshick et al. [2014]) algorithm for object detection. The setup first extracts object proposals using the Selective Search algorithm (van de Sande et al. [2011], Figure 2-19), computes a feature transform for each proposal using CNN features and trains a specific object detection classifier on top.
The *Faster-RCNN* (Ren et al. [2015]) variant of this algorithm integrates the object proposal step as a module within a unified network.

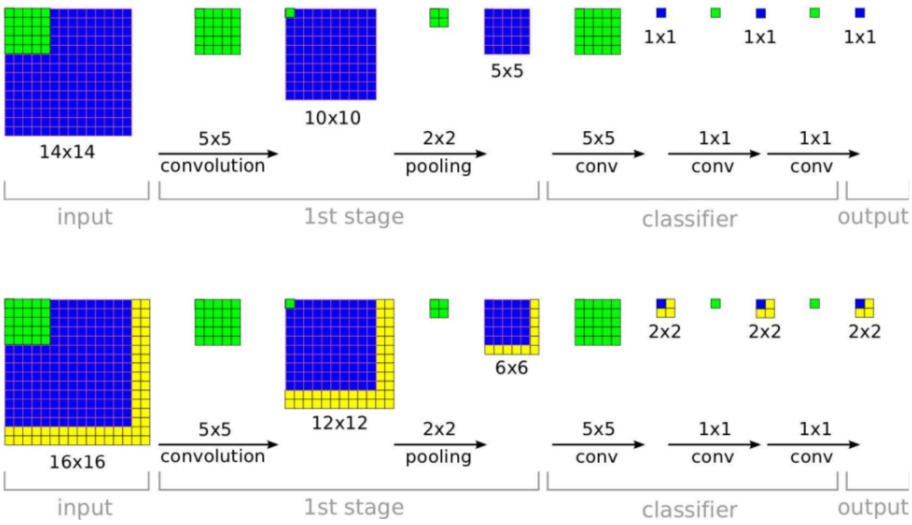



Figure 2-38: Original caption : ***The efficiency of ConvNets for detection.*** *During training, a ConvNet produces only a single spatial output (top). But when applied at test time over a larger image, it produces a spatial output map, e.g. 2x2 (bottom). Since all layers are applied convolutionally, the extra computation required for the larger image is limited to the yellow regions. This diagram omits the feature dimension for simplicity.* Figure and caption from Sermanet et al. [2014].

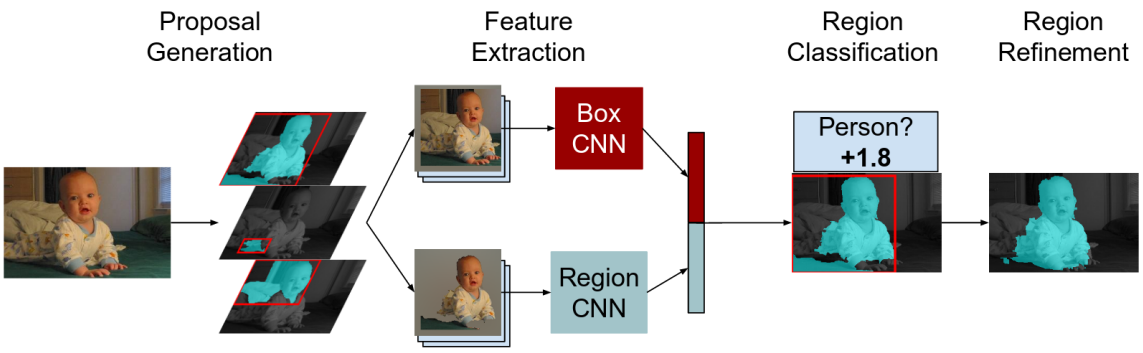



Figure 2-39: The Simultaneous Detection and Segmentation (SDS) algorithm proposed in Hariharan et al. [2014] is very similar to R-CNN (Girshick et al. [2014]) but uses object segmentation proposals to address the segmentation task.

**Object segmentation.** Object segmentation is the computer vision task that aims at partitioning an image into multiple sets of pixels corresponding to individual objects, separating them from the background. This refinement of object detection where the extent of objects is to be found at the pixel level, has also been converted to a neural network use-case, and setups progressively integrated the algorithms into single networks. One early iteration of neural networks for segmentation is the work of Hariharan et al. [2014], adopting an approach similar to R-CNN, but building on top of object segmentation proposals using the *Multiscale Combinatorial Grouping* algorithm of Arbeláez et al. [2014], shown in Figure 2-39.

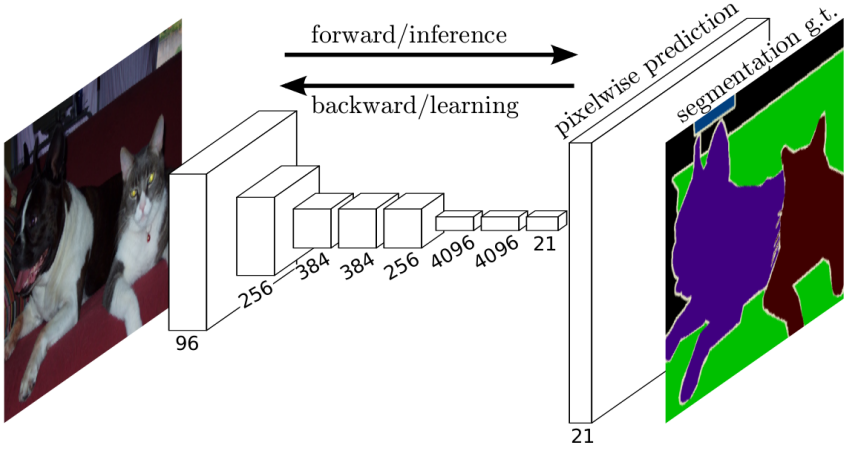



Figure 2-40: The Fully Convolutional Networks (FCN) approach proposed in Long et al. [2015] consists of assigning an object class to each pixel in the image, not using proposals anymore.

Further iterations in object segmentation research consist in removing the proposal step and integrating it within a neural network, as proposed in Long et al. [2015] (Figure 2-40) and in the DeepMask algorithm proposed in Pinheiro et al. [2015] (Figure 2-41). These methods leverage the sliding approach proposed in Sermanet et al. [2014], in order to generate pixel-level segmentation maps.

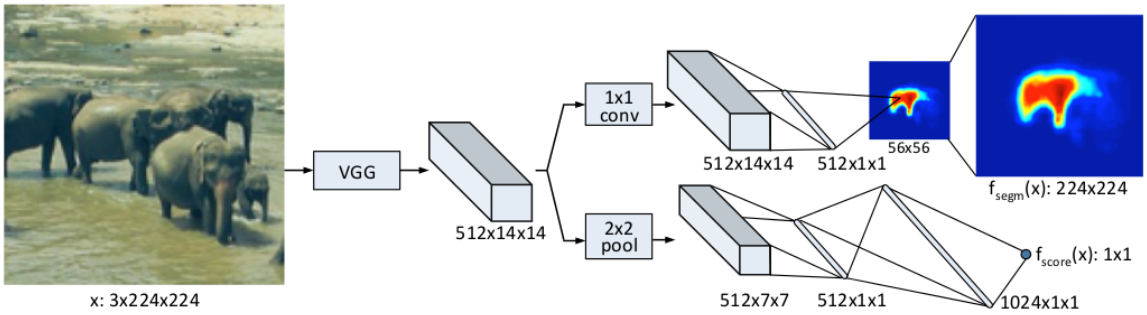



Figure 2-41: The DeepMask algorithm proposed in Pinheiro et al. [2015] consists of training a neural network to predict the segmentation mask of the object in the center of the image, without using proposals.

Figure 2-42: Example outputs of a pose estimation algorithm, DeepPose, proposed in Toshev and Szegedy [2014].

**Pose estimation.** Pose estimation is the computer vision task that consists of retrieving the pose of a person from an image by estimating the position of the limbs and their joints; we show examples in Figure 2-42. Before the AlexNet breakthrough, the state-of-the-art pose estimation method was the one of Yang and Ramanan [2011], consisting of building a flexible mixture of templates encoding spatial relations and capturing a notion of local rigidity, useful for dealing with the constraints of a human body, inspired by the *Deformable Part Models* of Felzenszwalb et al. [2010].

After the success of AlexNet in 2012, neural network methods appear for this task, building on their powerful features. Toshev and Szegedy [2014] introduce the usage of

neural networks in pose estimation with *DeepPose*. Chen and Yuille [2014] leverage the fact that the local appearance of a joint can help in predicting the appearance of neighboring joints, as described in Figure 2-43 and train a neural network for recognizing local positions. But Pfister et al. [2015] address this task in videos by proposing an integrated architecture building on the Two-Stream architecture (mentioned in the next paragraph), that directly outputs a pose estimate, as shown in Figure 2-44.

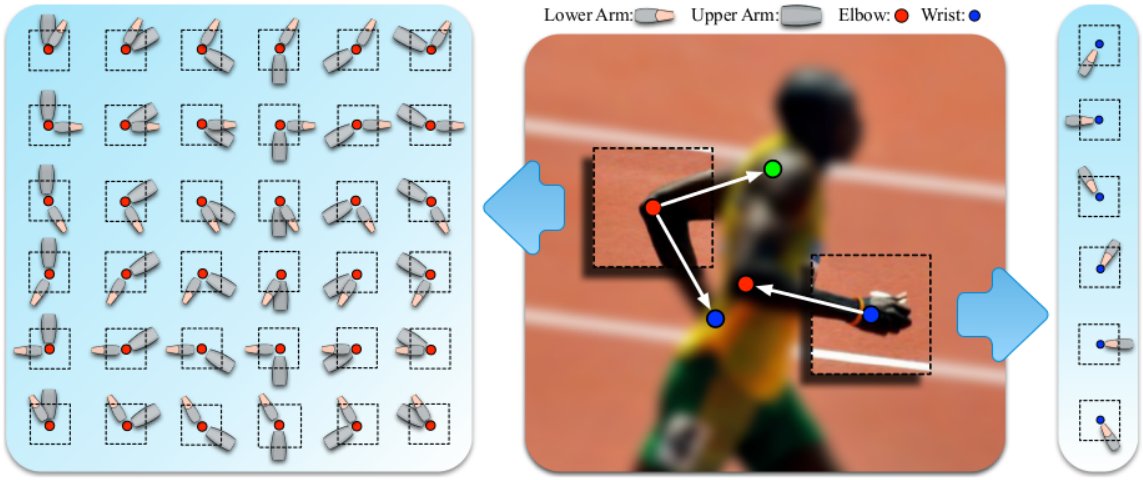



Figure 2-43: Motivation for the approach of Chen and Yuille [2014] for pose estimation. The local appearance of the patch surrounding the elbow can help in predicting the position of the shoulder and the wrist.

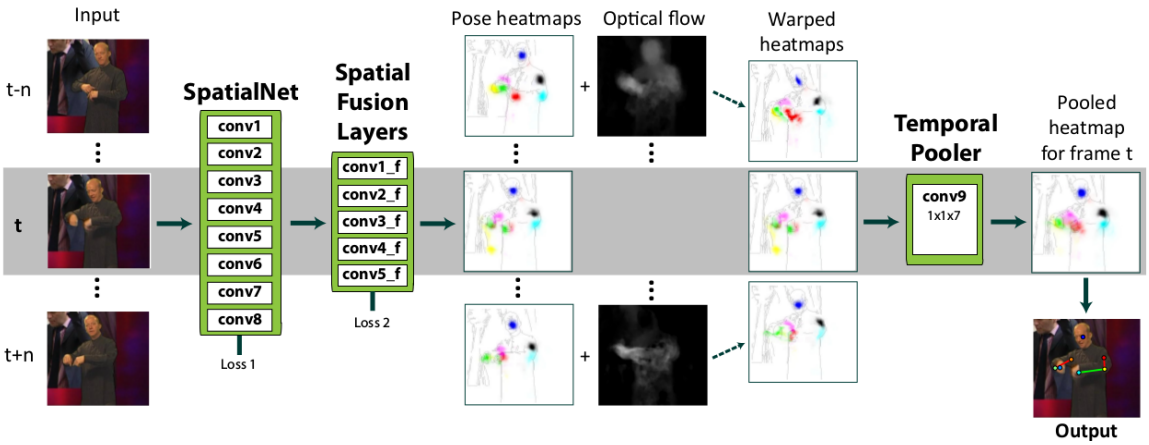



Figure 2-44: Pfister et al. [2015] adopt a more integrated approach for the task of pose estimation. They propose a neural network architecture that uses appearance and optical flow information to output a pose estimate.

In more recent work, Wei et al. [2016] introduce Convolutional Pose Machines, an improvement over the Pose Machines of Ramakrishna et al. [2014]. Pose Machines predict the pose of a person in an image, then use such initial predictions in subsequent processing stages to refine the location of the joints based on contextual features; each stage is trained separately. We illustrate the outputs of the Pose Machine in Figure 2-45. The Convolutional version proposed by Wei et al. [2016] integrates these different refinement stages within a single unified architecture, shown in Figure 2-46, and report large improvements by using end-to-end training. In concurrent work, Newell et al. [2016] propose Stacked Hourglass Networks for pose estimation, also integrating prediction and refinement stages within a unified architecture, shown in Figure 2-47.

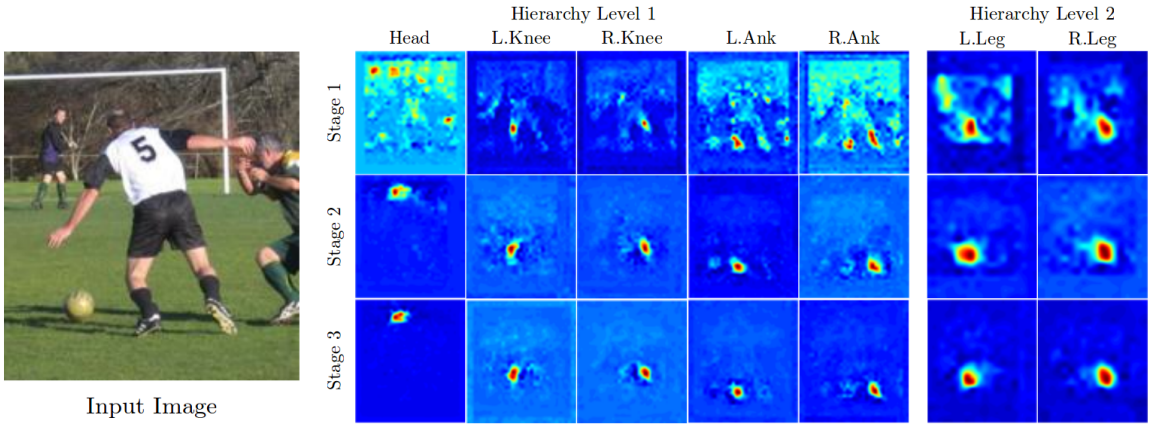



Figure 2-45: The original Pose Machines of Ramakrishna et al. [2014] first predict the location of joints from a single image, then refine the predictions in subsequent stages by using contextual features.

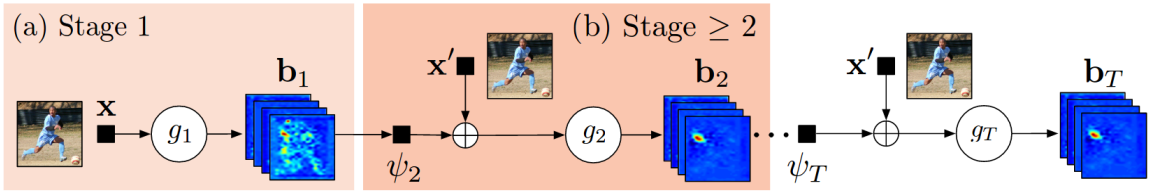



Figure 2-46: The improved Convolutional Pose Machines of Wei et al. [2016] integrate the prediction and refinement stages within a unified architecture that can be trained end-to-end, resulting in large performance improvements.

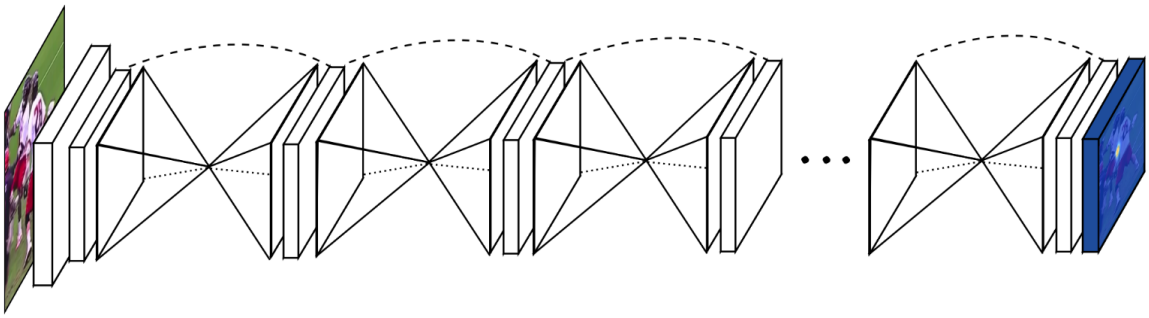


Figure 2-47: The Stacked Hourglass Network architecture of Newell et al. [2016] for pose estimation, integrating prediction and refinement steps in successive bottom-up/top-down steps.

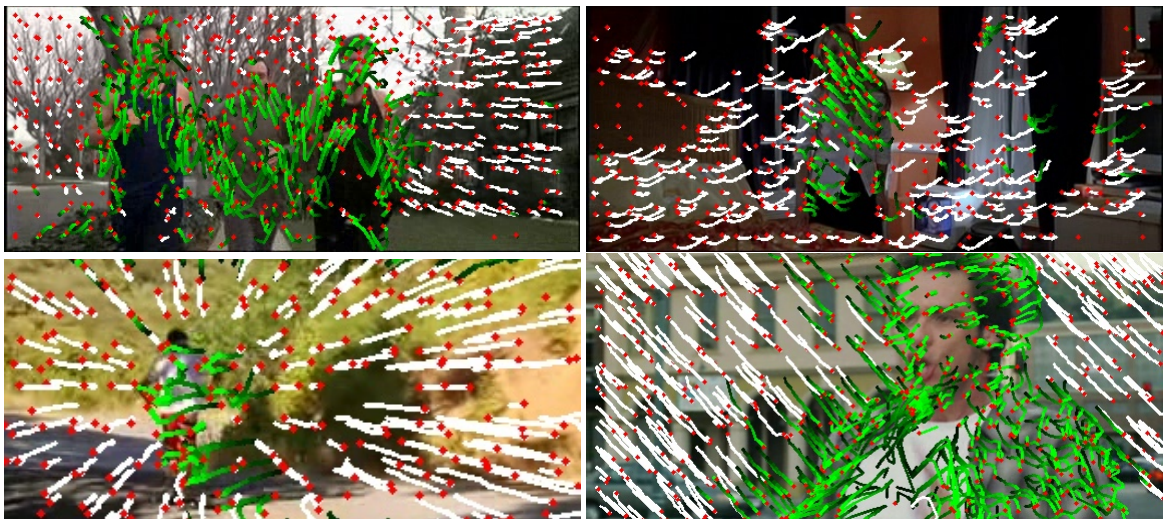


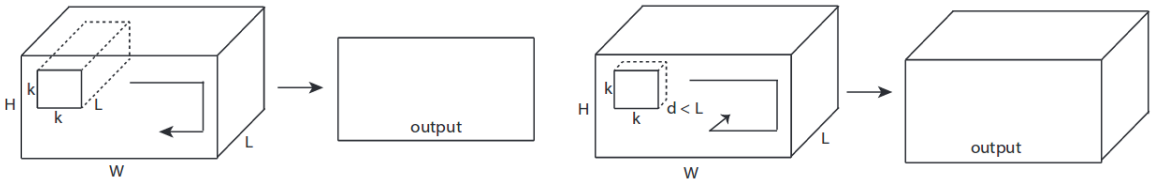
Figure 2-48: Wang and Schmid [2013] extract features in videos by following trajectories. In this figure, we can see, in white, trajectories that are considered due to camera motion.

**Video analysis.** Classification in the video domain also underwent a transition towards neural networks methods. The idea of combining appearance descriptors with optical flow[11] was explored with the *Dense Trajectories* (Wang et al. [2011]) then *Improved Dense Trajectories* (IDT) (Wang and Schmid [2013]) approaches for performing action classification on videos. IDT tracks regions using optical flow, creates trajectories, prunes trajectories due to camera motion (shown in Figure 2-48) and extracts descriptors along the trajectory to train a classifier on a Fisher Vector

[11] Optical flow encodes the underlying motion information between a pair of neighboring video frames.

encoding.

With neural networks, two important methods appear. One of them, proposed by Tran et al. [2015], adopts an approach similar to AlexNet, by learning a Convolutional Neural Network using spatio-temporal convolution (see Figure 2-49) on a large dataset of videos (Sports-1M, proposed by Karpathy et al. [2014]) in a supervised manner, then using the CNN features learned in this way to perform classification on other datasets.




(a) 2D convolution on multiple frames (b) 3D convolution

Figure 2-49: Tran et al. [2015] propose using spatial-temporal convolution in a neural network to process large datasets of videos and learn transferrable features, extending the approach of Krizhevsky et al. [2012] to the video domain. Original caption (adapted): *2D and 3D convolution operations. a) Applying 2D convolution on a video volume (multiple frames as multiple channels) results in an image. b) Applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.*

The other one, proposed by Simonyan and Zisserman [2014a] and shown in Figure 2-50, is close to the IDT approach in the sense that it merges appearance information with optical flow information in what is called a *Two-Stream architecture*. We can observe that this integrates the processing of these two modalities within a single network architecture. Following a similar trail, Arandjelović and Zisserman [2017] also propose using sound as an additional modality for recognition in videos.

Recent work by Carreira and Zisserman [2017] combines the two approaches by running the spatio-temporal convolution of Tran et al. [2015] on the image and flow modalities used in the Two-Stream architecture of Simonyan and Zisserman [2014a], and obtaining state-of-the-art results by pretraining the network on the Kinetics dataset (Kay et al. [2017]).

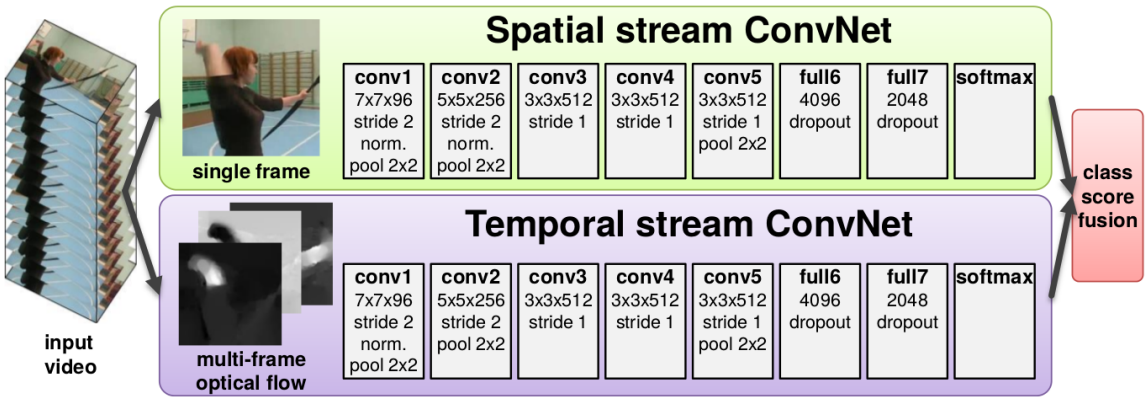



Figure 2-50: Simonyan and Zisserman [2014a] propose a *Two-Stream architecture* for video classification, combining appearance information (Spatial stream) and motion information (Temporal stream) within a single network.

**Conclusion of this section.** In this section, we describe a recent paradigm shift in computer vision, illustrated by various improvements in computer vision algorithms obtained by introducing CNN features in existing setups and then progressively building more complex networks from differentiable modules, integrating many processing steps. These integrated setups provide impressive results, but we want to point out that this progress is the result of a subtle balance between brilliant engineering and advances in the field of neural networks. The hype around *Artificial Intelligence*, fueled by the dramatic performance increases in recent applications, may mislead into believing in more fundamental advances than what has happened in the recent years.


Figure 2-51: Ordonez et al. [2011] propose learning from captioned images available on the Flickr image hosting website in their Im2Text approach. Top row: images and captions from the training set; bottom row: captions selected for query images at test time (good cases). The supervision takes the form of image captions.

## 2.5 Less supervision

The methods described in the previous section all rely on supervised datasets. In this section we are interested in ways to decrease the supervision necessary for a visual recognition setup to learn. Fully supervised methods require careful annotation of object location in the form of bounding boxes (Felzenszwalb et al. [2010]), segmentation (Yadollahpour et al. [2013]) or even location of object parts (Brox et al. [2011]), which is costly and can introduce biases. For example, should we annotate the dog's head or the entire dog? What if a part of the dog's body is occluded by another object?

### 2.5.1 Weak supervision

**Many ways to provide supervision.** Alternatively, it is possible to obtain weak image-level annotations, and they are less expensive to produce: for instance it is easier to annotate an image with the presence of an object than to draw a bounding

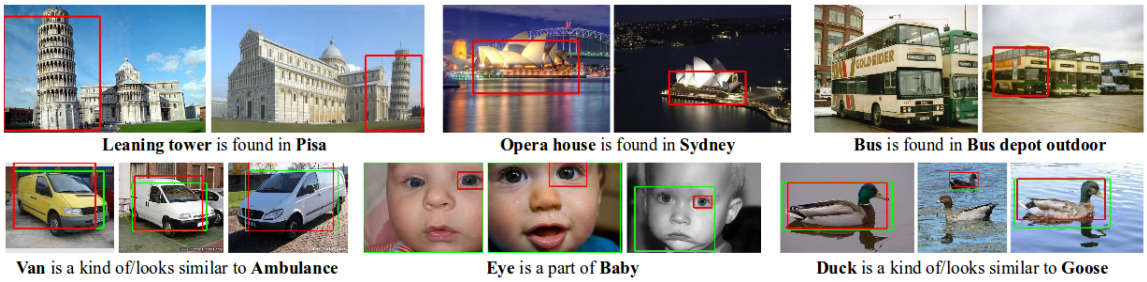



Figure 2-52: Examples of relationships obtained by the Never Ending Image Learner (NEIL) system of Chen et al. [2013], extracting knowledge from image queries on Google Image Search and clustering similar (according to image features) windows together to obtain bounding boxes for relevant objects. The supervision in this system is obtained by analyzing images corresponding to a similar search query.

box around it. This is an important setup for many practical applications as (weak) image-level annotations are often readily available in large amounts. Guillaumin et al. [2009] use text tags and propagate them to new images to perform automatic annotation; Ordonez et al. [2011] use image captions available on the Flickr image hosting website (see Figure 2-51) to match similar images and propose relevant captions; Joulin et al. [2016] propose learning the first layers of a CNN using also text annotations. Doersch et al. [2012] use geographical meta-data from pictures to recognize cities; Prest et al. [2012] use YouTube videos to collect frames containing objects of interest using tracking, and train object detectors; Hejrati and Ramanan [2012] annotate landmarks on car images to build a 3D car model then predict the pose of the car at test time; Shrivastava and Gupta [2013] build on RGBD (depth images) to improve the DPM model of Felzenszwalb et al. [2010]. In a more abstract line of work, Chen et al. [2013] and Divvala et al. [2014] attempt to learn relations between concepts with algorithms crawling the internet and extracting knowledge (see Figure 2-52).

To summarize, there are many different ways to obtain supervision, and the less precise annotations can sometimes be unexpensive and readily available. Therefore, there is interest in developing methods that solve tasks using as little supervision as possible, and the goal of weakly-supervised learning on images is to tackle the problem of expensive data collection.

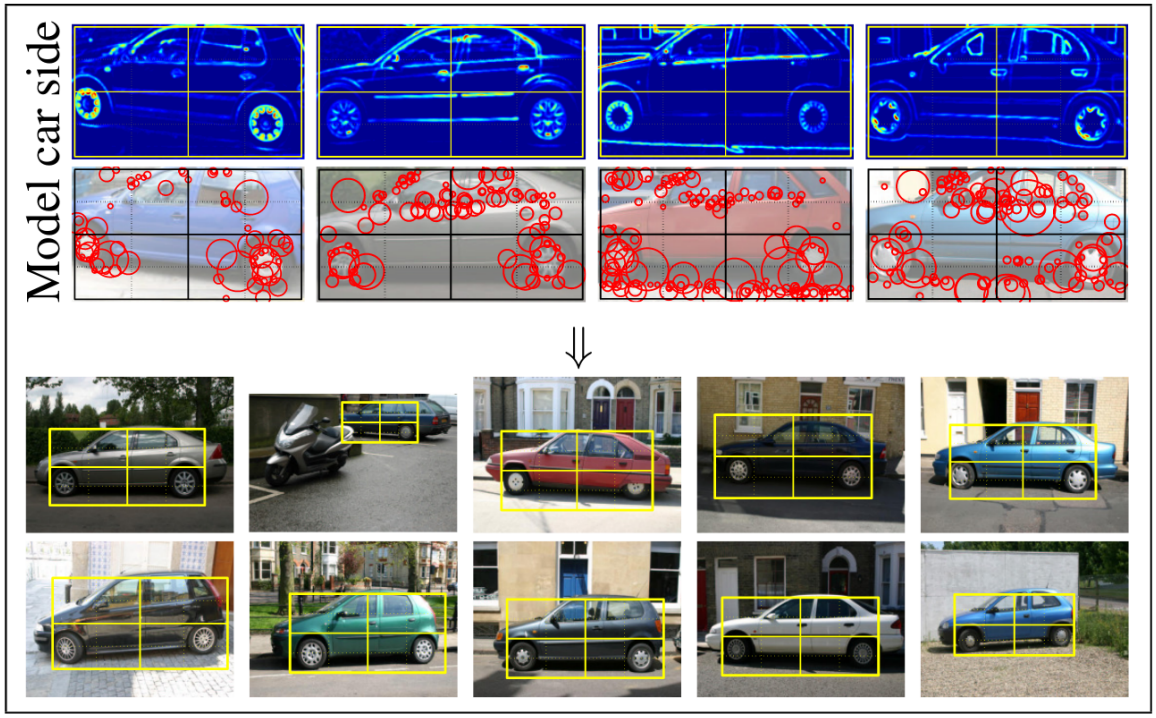



Figure 2-53: Exemplars by Chum and Zisserman [2007] learned on the Pascal VOC 2006 dataset without bounding box supervision. Original caption: *Examples of the exemplar representation for cars side [...]. Models show the spatial distribution of edges and appearance patches. [...] the images below the model show samples from the training images with the automatically learnt ROIs overlaid.*

**Early methods for weakly-supervised localization.** Early methods have focused on weakly-supervised object localization: given a dataset of images only knowing the presence or absence of a given class, the goal is to predict the location of the object. The work of Fergus et al. [2003a] (see Figure 2-14, Section 2.1), builds a graph model describing a constellation of parts specific to a category without using part-level supervision; by recognizing parts, the model effectively localizes the object. Similarly, Crandall and Huttenlocher [2006] learn models of objects by estimating the appearance of parts and the spatial relations between them. Winn and Jojic [2005] perform object segmentation using groups of images containing the same object. Chum and Zisserman [2007] address object detection on the Pascal VOC dataset by finding at test time regions of interest that are similar to *exemplars*, i.e. informative templates learned without location information by considering similarities across training examples (see Figure 2-53). In more recent work, Blaschko et al. [2010] propose using

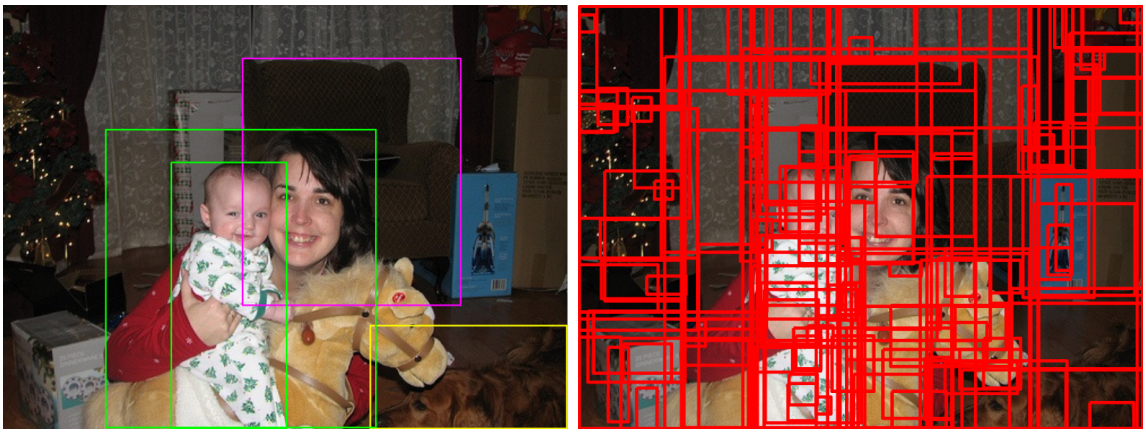


Figure 2-54: Using objectness measures to consider only a subset of the possible windows in an image greatly reduces the complexity of the weakly-supervised localization problem. Images by Song et al. [2014]. Left: ground-truth bounding box annotations; right: 100 candidate windows proposed by Selective Search (van de Sande et al. [2011]).

a combination of strongly annotated (with bounding boxes) and weakly annotated (image-level labels) training images and report that only few strong annotations are necessary to obtain good performance. Pandey and Lazebnik [2011] adapt the DPM algorithm of Felzenszwalb et al. [2010] to a weakly-supervised scenario.

**Objectness-based methods for weakly-supervised localization.** The introduction of *objectness* measures (Alexe et al. [2010], van de Sande et al. [2011], see Figures 2-18,2-19, Section 2.1) allows considering only a few candidate windows in images that are likely to contain an object instead of the dense set of all windows, greatly reducing the complexity of the problem as illustrated in Figure 2-54. As a result, most of the recent weakly-supervised localization methods build on objectness algorithmss.

Deselaers et al. [2010] propose a Conditional Random Field model with a unary potential based on the objectness measure of a window, and a pairwise potential describing the similarity between two windows in the dataset based on various image features. Song et al. [2014] use Selective Search (van de Sande et al. [2011]) and CNN features, inspired by the R-CNN approach (Girshick et al. [2014]). Similarly, the works of Wang et al. [2014], Cinbis et al. [2017] build on Selective Search and

CNN features to address the task, as well as Bilen and Vedaldi [2016] along with our extension in Kantorov et al. [2016][12].

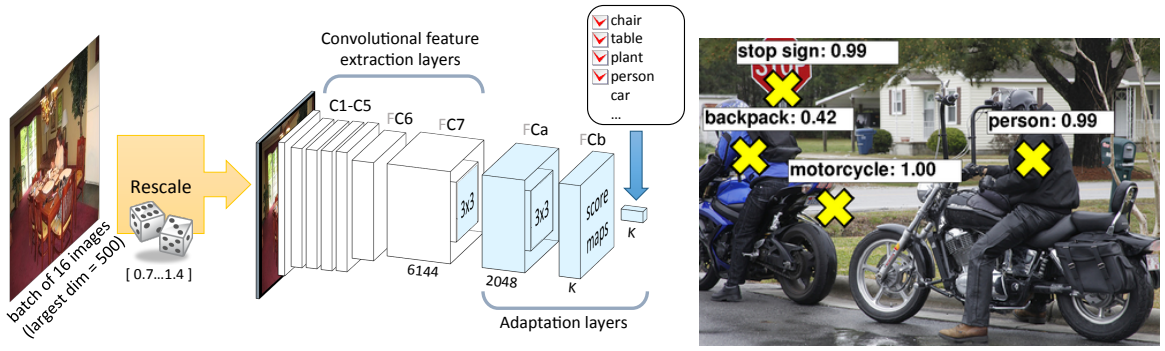
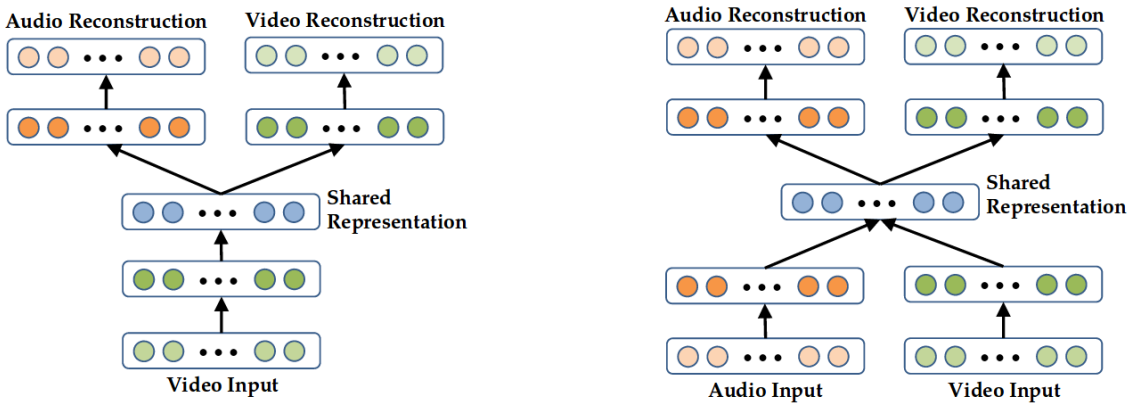



Figure 2-55: Building on the efficient sliding idea explained in Sermanet et al. [2014] (see Figure 2-38), we propose a weakly-supervised approach for multi-object classification, that allows predicting the location of objects coarsely in an image using classification score maps at test time. The method will be explained fully in Chapter 4.

**Approach in this thesis.** Our method uses supervision in the form of image-level labels indicating the presence or absence of objects in images, then coarsely predicts the locations of these objects in unseen images. Our CNN architecture relies on a global max-pooling operation which can be seen as a variant of Multiple Instance Learning (Foulds and Frank [2010], Kotzias et al. [2014], Viola et al. [2005]) if we refer to each image as a "bag" and treat each image window as a "sample". As a result, our method is able to predict the approximate location of objects in complex cluttered scenes as described in Chapter 4. With this procedure, one can efficiently retrieve the image region that leads to a classification decision. Using the location predictions that are obtained in this way, Sun et al. [2016] further extend our method with a verification network improving the classification performance on these datasets.

**Conclusions.** Various approaches show that it is possible to learn from different modalities, such as text, or even geographical meta-data. Such supervision is often readily available on the Internet, encouraging new methods using less fully-annotated training data. In this section, we discuss methods able to learn from weak forms of

[12]Not related to the next paragraph.




(a) Video-Only Deep Autoencoder (b) Bimodal Deep Autoencoder

Figure 2-56: Original caption by Ngiam et al. [2011]: *Deep Autoencoder Models. A "video-only" model is shown in (a) where the model learns to reconstruct both modalities given only video as the input. A similar model can be drawn for the "audio-only" setting. We train the (b) bimodal deep autoencoder in a denoising fashion, using an augmented dataset with examples that require the network to reconstruct both modalities given only one. [...]*

supervision, focusing on weakly-supervised object localization related to our work in Chapter 4.

### 2.5.2 Unsupervised learning with neural networks

In the following, we discuss approaches attempting unsupervised learning with neural networks. The autoencoder line of research notably fits in this category, and we refer to Section 2.2 for a description of related works. Below we focus on the more recent self-supervised learning and Generative Adversarial Network approaches.

**Self-supervision.** The work of Ngiam et al. [2011] proposes a multimodal autoencoder approach that given a video, attempts to reconstruct the sound from the image, or vice-versa (see Figure 2-56). This form of supervision is free, as it relies only on using correlated signal from different modalities. As a result, this approach allows pre-training layers of a neural network that can then be used in a transfer scenario (as in Chapter 3), effectively learning CNN features without annotations, with the system supervising itself.

Following this line of work, Agrawal et al. [2015] and Jayaraman and Grauman

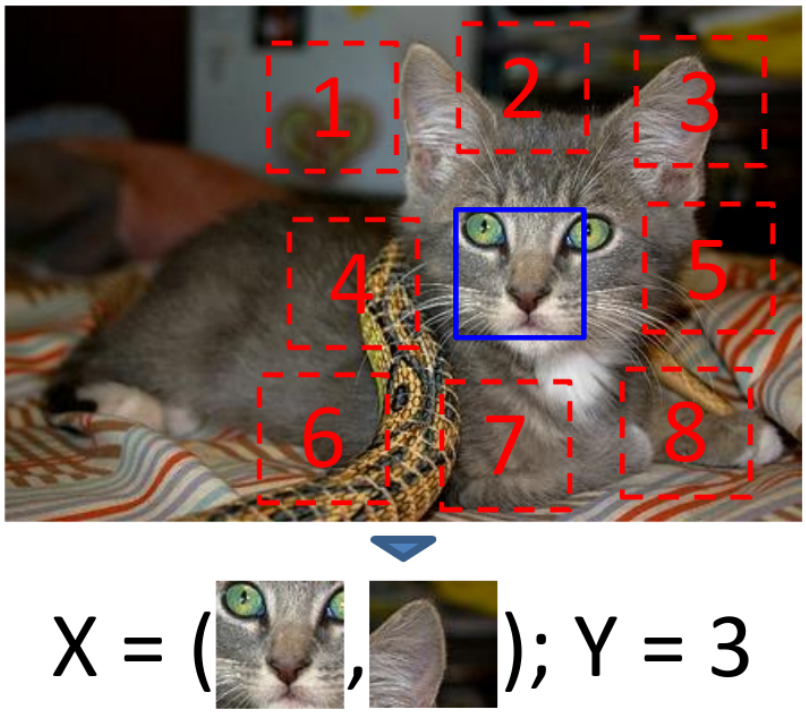



Figure 2-57: Doersch et al. [2015] propose training a neural network to predict the relative configuration of two image patches extracted from an image. Original caption: *The algorithm receives two patches in one of these eight possible spatial arrangements, without any context, and must then classify which configuration was sampled.*

[2015] propose using ego-motion as a form of correlated signal in videos: the motion of the camera is recorded along with the visual input, providing an additional modality for learning. Also with videos, Wang and Gupta [2015] propose tracking objects in an unsupervised way, and training a network to generate similar representations (in the $L^2$ feature space) for two regions containing the same object. Doersch et al. [2015] propose a different task with context prediction, where the task is to retrieve the spatial configuration of two patches extracted from a source image (see Figure 2-57). Zhang et al. [2016] propose using grayscale images to predict the colored versions, allowing them to build a colorization algorithm as well as learning features in a neural network (see Figure 2-58). These works build on the idea that it should be necessary for a network to understand the content of images to accomplish the self-supervised task.

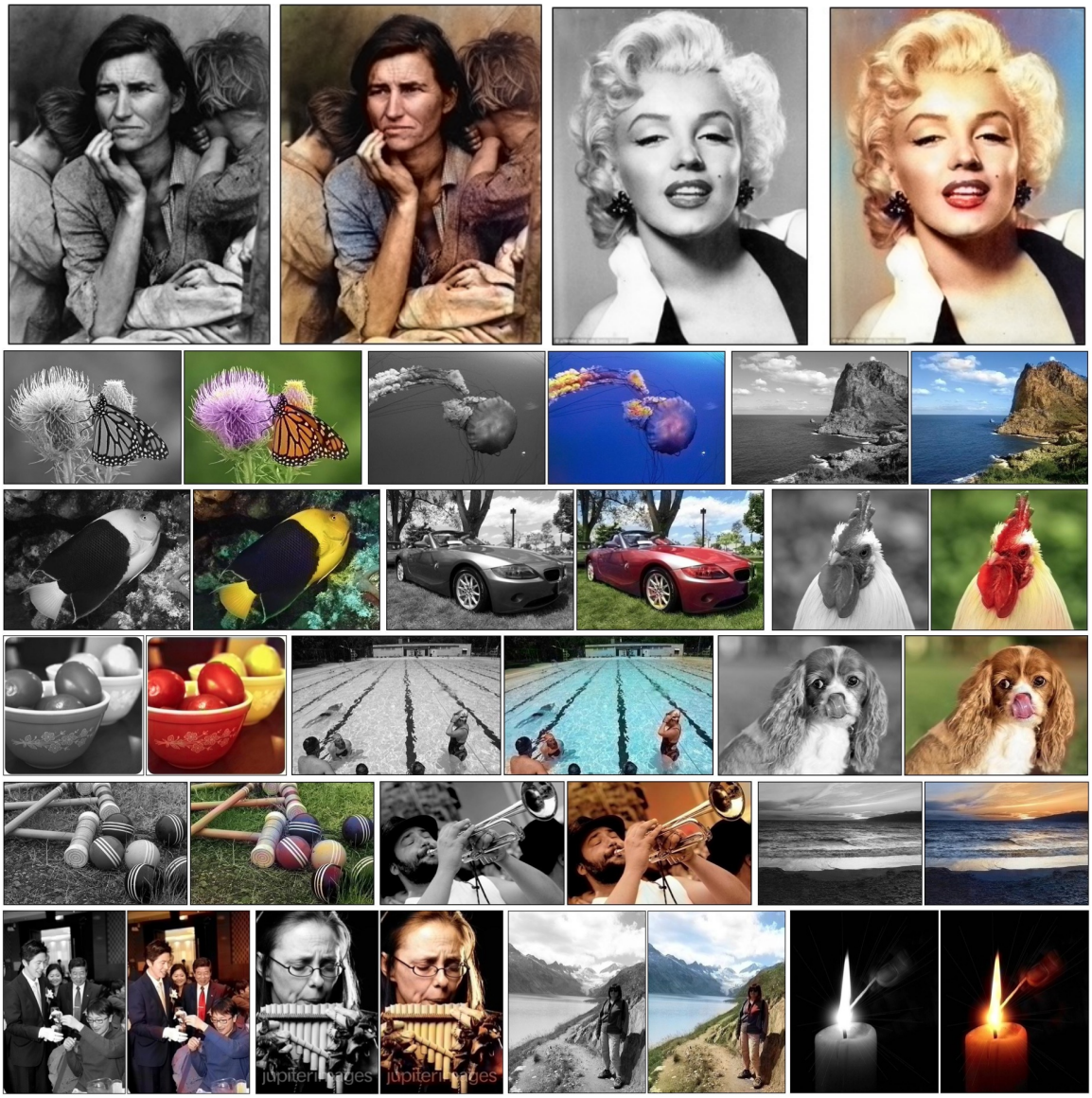


Figure 2-58: Zhang et al. [2016] propose training a neural network to predict the colors of grayscale images as a self-supervised task. At test time, the algorithm is able to colorize legacy pictures.

Figure 2-59: Adversarial examples proposed by Szegedy et al. [2013]. Original caption: *Adversarial examples generated for AlexNet.(Left) is a correctly predicted sample, (center) difference between correct image, and image predicted incorrectly magnified by 10x (values shifted by 128 and clamped), (right) adversarial example. All images in the right column are predicted to be an "ostrich, Struthio camelus". Average distortion based on 64 examples is 0.006508. Plase refer to* `http://goo.gl/huaGPb` *for full resolution images. The examples are strictly randomly chosen. There is not any postselection involved.*

**Generative Adversarial Networks.** Szegedy et al. [2013] study intriguing properties of neural networks and attract interest in the input gradient in these architectures. They notice that modifying an input image, with a distorsion imperceptible to the human eye, can lead to different classification results with a neural network if this distorsion is in the same direction as the gradient, as shown in Figure 2-59. The resulting images are called *adversarial examples* and illustrate the weaknesses of neural networks.

Goodfellow et al. [2014] use this input gradient in a different way and propose a new architecture for unsupervised training named Generative Adversarial Networks. The training procedure for GANs consists of letting two neural network adversaries - a *Generator* G and a *Discriminator* D - compete in the following game. Given an unlabeled training dataset:

- G generates a random sample $x$ from a random vector $z$ and passes it to D.
- D is trained to classify whether $x$ comes from the dataset or not.

- D provides gradient to G, communicating how G should modify its output to *decrease* the performance of D for classification.
- G adapts progressively, eventually providing more realistic samples.

This idea is applied to the image domain in Denton et al. [2015]. Radford et al. [2016], have popularized this idea with promising results on generating new human faces, shown in Figure 2-60. As a result of the procedure, GAN generators are able to generate random images resembling a training dataset.


Figure 2-60: Random faces generated by a Generative Adversarial Network architecture. Left: results obtained in early stages of training, right: later stages. Images courtesy of A.B.L. Larsen and S.K. Sønderby (http://torch.ch/blog/2015/11/13/gan.html).

This line of work is currently being actively explored in the learning community. In vision, successful applications have been proposed for other image generation tasks. For instance, related to the self-supervision approach of the previous paragraph, Pathak et al. [2016] notably use a discriminator as an additional *adversarial loss* term to perform inpainting and report better quality in generated patches (see Figure 2-61).

Despite various successful applications, the understanding of the properties of

(a) Input context (b) Human artist (c) Context Encoder (L2 loss) (d) Context Encoder (L2+Adversarial loss)

Figure 2-61: Pathak et al. [2016] propose using an adversarial loss based on a GAN discriminator to improve inpainting. Original caption: *Qualitative illustration of the task. Given an image with a missing region (a), a human artist has no trouble inpainting it (b). Automatic inpainting using our context encoder trained with L2 reconstruction loss is shown in (c), and using both L2 and adversarial losses in (d).*

GANs remains limited. For example, it is difficult to measure whether the generated distribution matches the reference data distribution. We discuss this subject further in our work in Chapter 5.

## 2.6 Conclusion of this chapter

In this chapter, we describe the progress of research in visual recognition and neural networks and the recent paradigm shift caused by the success of Krizhevsky et al. [2012] in the ImageNet classification challenge of 2012.

In Section 2.1, we review the literature of computer vision with a focus on visual recognition before 2012. Early attempts at solving the problem of computer vision consisted of pure geometric methods. Given the difficulty and the numerous modes of variation of image data, the field started relying on empirical models through datasets, prompting the use of machine learning techniques to tackle recognition. Then, looking for suitable ways to represent images in this context, global representations were replaced progressively by local image descriptors such as SIFT in the 2000s, paving the way for efficient statistical approaches to instance-level object matching, and later category-level object recognition with SVMs. In parallel, refined techniques based on structured models were proposed, improving the localization of objects in images and

eventually leading to object detection methods suitable for large-scale applications.

In Section 2.2, we review the literature related to neural networks before 2012. Inspired by studies of the human brain, early architectures were proposed for visual recognition, and the introduction of the backpropagation algorithm made neural networks trainable. Successful applications were proposed for character recognition using Convolutional Neural Networks (CNNs), and implemented in real-world situations such as check-reading in the '90s. The technology was essentially mature already and very close to its current state, but the lack of data and computational resources at that time held back applications on realistic images. During the 2000s, with the quick progress of other computer vision methods based on hand-crafted feature descriptors, neural networks could not compete with the state of the art in the field of visual recognition, despite many attempts at feature learning with unsupervised approaches: this was the "winter" of neural networks.

In Section 2.3 we review the ImageNet competition event that occurred in 2012. The growing scale of Internet in the 2000s and the wide spread of image capturing devices such as smartphones have made visual data available in large amounts. Driven by the perspective of large-scale recognition, millions of these images were annotated and stored in the ImageNet database, fueling the research in visual recognition in the early 2010s. The work of Krizhevsky et al. [2012], implementing CNNs on GPUs, increased the speed of these algorithms by two orders of magnitude, providing the computational power necessary to learn from large databases. The ImageNet challenge of 2012, where Convolutional Neural Networks outperformed all alternative methods by a wide margin, illustrated a critical point that was reached by data and computational power, inducing a paradigm shift in visual recognition.

In Section 2.4 we describe this paradigm shift that has resulted in the introduction of neural network components in most visual recognition algorithms. First, hand-crafted features such as SIFT and their variants were replaced by features extracted from intermediate layers in CNNs (see Chapter 3), leading to major performance improvements. Then, more complex purpose-built architectures were introduced, integrating various steps of processing within unified architectures trained end-to-end.

In parallel, advances in GPU hardware and corresponding software tools have led to increasingly powerful networks and higher performance in real-world applications, effectively democratizing the technology. While our discussion here is focused on visual recognition, these improvements have influenced many fields such as machine translation and speech recognition.

In Section 2.5 we discuss possible paths for future progress, and focus on weaker forms of supervision given the high cost of data annotation. While CNNs are suited for a large number of applications, there are challenges that cannot be addressed yet. More abstract cases such as image understanding and action recognition in videos remain difficult in real-world scenarios (see Chapter 1) where the complexity of the tasks may exceed our capacity at providing a suitable labeled dataset for learning. Therefore, we are interested in ways to learn from weak forms of supervision (see Chapter 4), using different supervision modalities and possibly, with recent algorithms, without supervision at all (see Chapter 5).

# Chapter 3

# Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks

In this chapter, we address the problem of feature learning and feature generalization across tasks in visual recognition. The procedure described here consists of using a large dataset (the *source task*) to build a powerful feature representation for images within a neural network. Then, once this representation is learned, it is applied on a smaller dataset (the *target task*) by keeping the learned parameters and the structure of the first processing modules. We bridge the performance gap between ImageNet classification (which contains more than one million training images) and other smaller-scale computer vision tasks, showing the generality of these pre-trained features for visual recognition. We show directions for improvement by running the pre-training procedure on image data related to the target task. We demonstrate the power of this technique by outperforming the state of the art on the Pascal VOC image classification dataset.

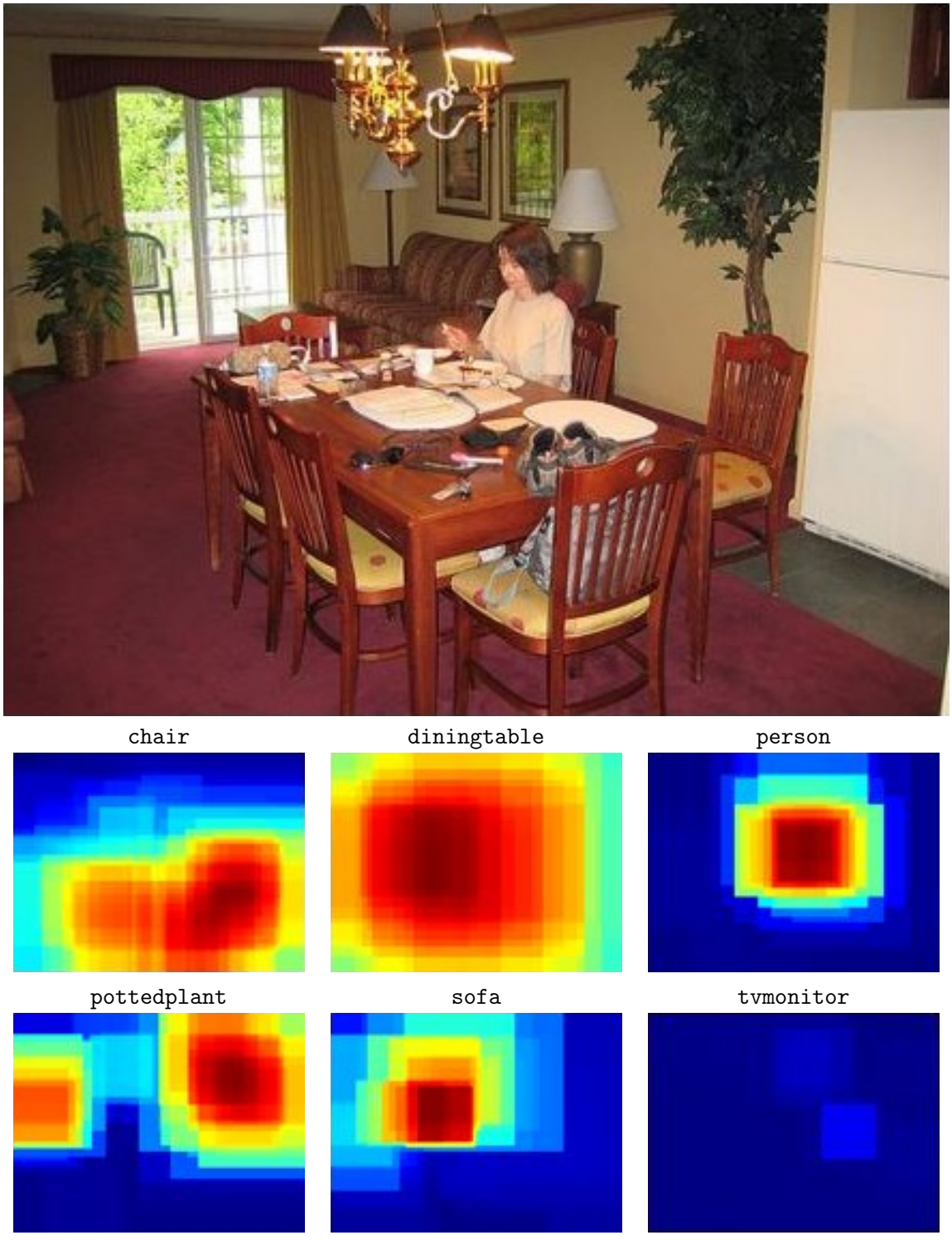



Figure 3-1: Recognition and localization results of our method for a Pascal VOC test image. Output maps are shown for six object categories with the highest responses. The classification heat maps demonstrate the potential of this setup for localizing objects in images.

## 3.1 Introduction

Object recognition has been a driving motivation for research in computer vision for many years. Recent progress in the field has allowed recognition to scale up from a few object instances in controlled setups towards hundreds of object categories in arbitrary environments.

Much of this progress has been enabled by the development of robust image descriptors such as SIFT (Lowe [2004a]) and HOG (Dalal and Triggs [2005]), bag-of-features image representations (Csurka et al. [2004], Lazebnik et al. [2006], Perronnin et al. [2010], Sivic and Zisserman [2003]) as well as deformable part models (Felzenszwalb et al. [2010]).

Another enabling factor has been the development of increasingly large and realistic image datasets providing object annotation for training and testing, such as Caltech256 (Griffin et al. [2007]), Pascal VOC (Everingham et al. [2010]) and ImageNet (Deng et al. [2009]).

**Neural networks for vision.** Although they were common before 2012, neural networks have a long history in visual recognition. Rosenblatt's Mark I Perceptron (Rosenblatt [1957]) arguably was one of the first computer vision systems.

Inspired by the neural connectivity pattern discovered by Hubel and Wiesel [1959], Fukushima's Neocognitron (Fukushima [1980]) extended earlier networks with invariance to image translations. But neither Rosenblatt nor Fukushima had the means to train the association layer weights in a supervised manner. This was achieved about a decade later.

Combining the back-propagation algorithm (Rumelhart et al. [1986]) with the Neocognitron architecture, convolutional neural networks (Lang and Hinton [1988], LeCun et al. [1989]) quickly achieved excellent results in optical character recognition leading to large-scale industrial applications (LeCun et al. [1998a], Simard et al. [2003]). Although convolutional networks have been advocated for other vision tasks (Vaillant et al. [1994], Osadchy et al. [2005]) including generic object recognition (LeCun et al. [2004]), their performance was limited by the relatively small size of

the standard object recognition datasets (such as the CalTech and Pascal datasets).

This situation changed with the appearance of the ImageNet dataset (Deng et al. [2009]) and the rise of GPU computing. Using a very efficient GPU implementation of convolutional neural networks, (Krizhevsky et al. [2012]) achieve a performance leap in image classification on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC-2012). Although the challenge calls only for the recognition of 1000 object categories, the best result was achieved using a network trained on the entire ImageNet Fall 2011 release (15 million images, 22000 categories).

**Goal: a generic feature transform.** The fact that Rosenblatt and Fukushima went ahead without a suitable supervised learning algorithm indicates that they hoped to engineer an effective association layer and reuse this work for a variety of problems.

The remarkable effectiveness of the SIFT features has certainly proven them right for the case of low-level image features. Recent works on scattering transforms (Bruna and Mallat [2013]) provide the means to build mid-level features from first principles. It is also telling to observe that the early deep learning papers (Hinton [2007], Hinton and Salakhutdinov [2006]) placed the focus on unsupervised learning, and that the convolution kernels learned by the first layer of a convolutional neural network are usually similar to manually engineered edge detectors (see Krizhevsky et al. [2012], Farabet et al. [2013] for instance).

From this perspective, it seems that supervised learning in a convolutional neural network simply customizes mid-level features for a particular task.

It is therefore natural to investigate under which conditions the features learned by a convolutional neural network on a large dataset can be reused for other computer vision tasks. This contribution reports on experiments carried out by training mid-level features on the ImageNet dataset, which currently is the largest publically available labeled image dataset, and using these features on the Pascal VOC data (Everingham et al. [2010]), which has sometimes been described as the most challenging current object recognition dataset and therefore the most worthy of interest (Torralba and Efros [2011]). We focus on the Pascal VOC classification task for ex-

pediency reasons: although the system clearly locates the recognized objects in the input image, it does so in a manner that is not immediately suitable for the Pascal VOC detection task.

**The issue of image statistics.** It has been argued that computer vision datasets have significant differences in image statistics (Torralba and Efros [2011]). For example, while objects are typically centered in Caltech256 and ImageNet datasets, other datasets such as Pascal VOC and LabelMe are more likely to contain objects embedded in a scene (see Figure 3-3). Differences in viewpoints, scene context, "background" (negative class) and other factors, inevitably affect recognition performance when training and testing across different domains (Pirsiavash and Ramanan [2012], Saenko et al. [2010], Torralba and Efros [2011]). Similar phenomena have been observed in other areas such as NLP (Jiang and Zhai [2007]). Given the "data-hungry" nature of CNNs and the difficulty of collecting large-scale image datasets, the applicability of CNNs to tasks with limited amount of training data appears as an important open problem.

**Contributions.** To address this problem, we propose to transfer image representations learned with CNNs on large datasets to other visual recognition tasks with limited training data. In particular, we design a method that uses ImageNet-trained layers of CNN to compute efficient mid-level image representation for images in Pascal VOC. We analyze the transfer performance and show significant improvements on the Pascal VOC object and action classification tasks, outperforming the state of the art. We also show promising results for object and action localization. Results of object recognition and localization by our method are illustrated in Figure 3-1.

## 3.2 Transferring CNN weights

The CNN architecture of Krizhevsky et al. [2012] contains more than 60 million parameters. Directly learning so many parameters from only a few thousand training images is problematic. The key idea of this work is that the internal layers of the

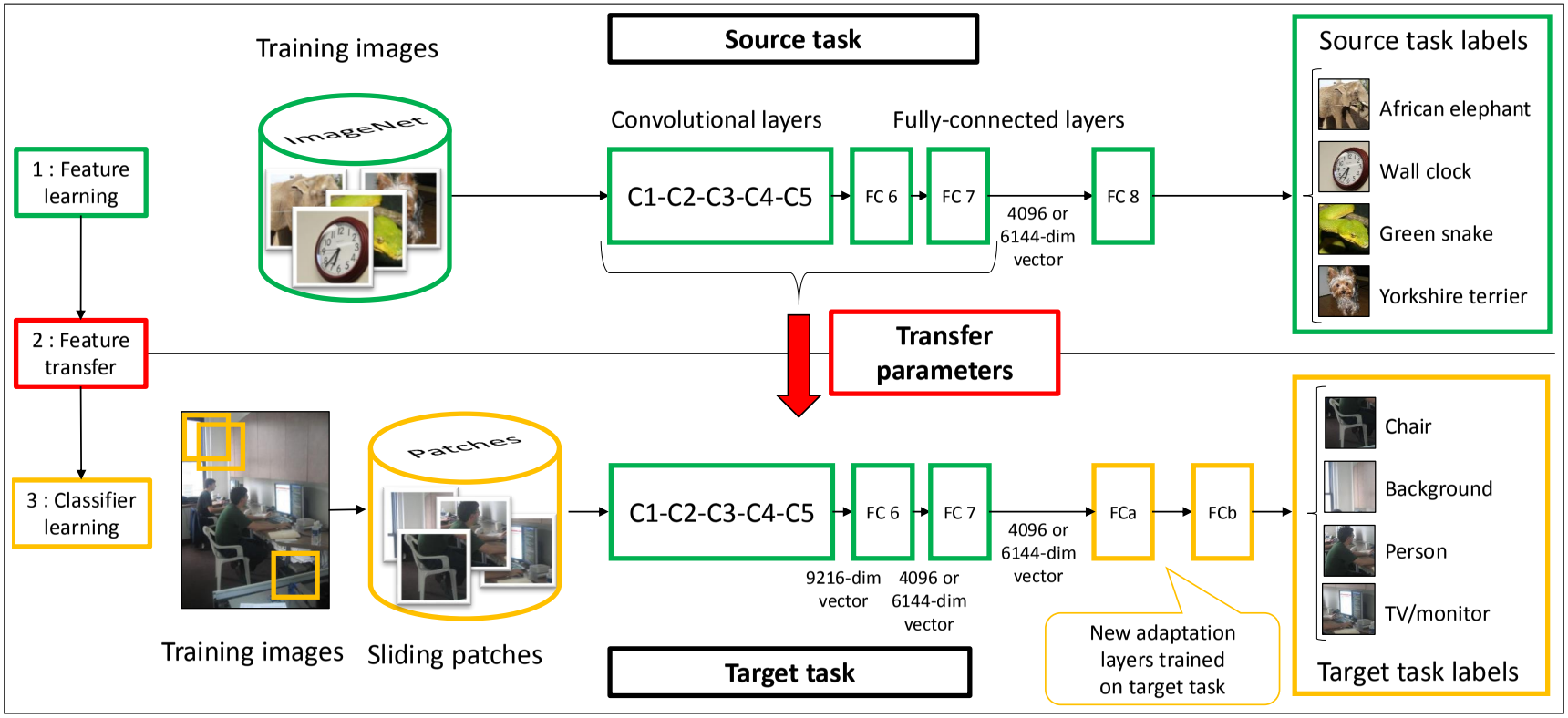



Figure 3-2: **Transferring parameters of a CNN.** First, the network is trained on the source task (ImageNet classification, top row) with a large amount of available labelled images. Pre-trained parameters of the internal layers of the network (C1-FC7) are then transferred to the target tasks (Pascal VOC object or action classification, bottom row). To compensate for the different image statistics (type of objects, typical viewpoints, imaging conditions) of the source and target data we add an adaptation layer (fully connected layers FCa and FCb) and train them on the labelled data of the target task.

CNN can act as a *generic extractor of mid-level image representation*, which can be pre-trained on one dataset (the *source task*, here ImageNet) and then re-used on other *target tasks* (here object and action classification in Pascal VOC), as illustrated in Figure 3-2. However, this is difficult as the labels and the distribution of images (type of objects, typical viewpoints, imaging conditions, etc.) in the source and target datasets can be very different, as illustrated in Figure 3-3. To address these challenges we (i) design an architecture that explicitly remaps the class labels between the source and target tasks (Section 4.2), and (ii) develop training and test procedures, inspired by sliding window detectors, that explicitly deal with different distributions of object sizes, locations and scene clutter in source and target tasks (Sections 3.2.2 and 4.5).

### 3.2.1 Network architecture

For the source task, we use the network architecture of Krizhevsky et al. [2012]. The network takes as input a square $224 \times 224$ pixel RGB image and produces a distribution over the ImageNet object classes. This network is composed of five successive convolutional layers C1...C5 followed by three fully connected layers FC6...FC8 (Figure 3-2, top).

The three fully connected layers compute

$$\mathbf{Y}_6 = \sigma(\mathbf{W}_6 \mathbf{Y}_5 + \mathbf{B}_6)$$
$$\mathbf{Y}_7 = \sigma(\mathbf{W}_7 \mathbf{Y}_6 + \mathbf{B}_7)$$
$$\mathbf{Y}_8 = \psi(\mathbf{W}_8 \mathbf{Y}_7 + \mathbf{B}_8)$$

where $\mathbf{Y}_k$ denotes the output of the $k$-th layer,

$\mathbf{W}_k, \mathbf{B}_k$ are the trainable parameters of the $k$-th layer,

$\sigma(\mathbf{X})[i] = \max(0, \mathbf{X}[i])$ is the ReLU non-linear activation function,

$\psi(\mathbf{X})[i] = \frac{e^{\mathbf{X}[i]}}{\sum_j e^{\mathbf{X}[j]}}$ is the SoftMax non-linear activation function.

For target tasks (Pascal VOC object and action classification) we wish to design a network that will output scores for target categories, or `background` if none of the categories are present in the image. However, the object labels in the source task can be very different from the labels in the target task (also called a "label bias" by Torralba and Efros [2011]). For example, the source network is trained to recognize different breeds of dogs such as `husky dog` or `australian terrier`, but the target task contains only one label `dog`. The problem becomes even more evident for the target task of action classification. What object categories in ImageNet are related to the target actions `reading` or `running`?

In order to achieve the transfer, we remove the output layer FC8 of the pre-trained network and add an adaptation layer formed by two fully connected layers FCa and FCb (see Figure 3-2, bottom) that use the output vector $\mathbf{Y}_7$ of the layer FC7 as input. Note that $\mathbf{Y}_7$ is obtained as a complex non-linear function of potentially all input pixels and may capture mid-level object parts as well as their high-level configurations (Le et al. [2012], Zeiler et al. [11]). The FCa and FCb layers compute:

$$\mathbf{Y}_a = \sigma(\mathbf{W}_a \mathbf{Y}_7 + \mathbf{B}_a)$$
$$\mathbf{Y}_b = \psi(\mathbf{W}_b \mathbf{Y}_a + \mathbf{B}_b)$$

where $\mathbf{W}_a$, $\mathbf{B}_a$, $\mathbf{W}_b$, $\mathbf{B}_b$ are the trainable parameters.

In all our experiments, FC6 and FC7 have equal sizes (either 4096 or 6144, see Section 3.3), FCa has size 2048, and FCb has a size equal to the number of target categories.

The parameters of layers C1...C5, FC6 and FC7 are first trained on the source task, then transferred to the target task and kept fixed. Only the adaptation layer is trained on the target task training data as described next.


Figure 3-3: Illustration of different dataset statistics between the source (ImageNet) and target (Pascal VOC) tasks. Pascal VOC data displays objects embedded in complex scenes, at various scales (right), and in complex mutual configurations (middle). Left: Image from ImageNet with label `maltese terrier`. Middle and right: Images from Pascal VOC with label `dog`.

### 3.2.2 Network training

First, we pre-train the network using the code of Krizhevsky et al. [2012] on the ImageNet classification source task. Each image typically contains one object centered and occupying significant portion of the image with limited background clutter as illustrated in Figure 3-3(left). The network is trained to predict the ImageNet object class label given the entire image as input. Details are given in Section 3.3.

**Dataset bias issues.** As discussed above, the network is pre-trained to classify source task images that depict single centered objects. The images in the target task, however, often depict complex scenes with multiple objects at different scales and orientations with significant amount of background clutter, as illustrated in Figure 3-3 (middle and right). In other words, the distribution of object orientations and sizes as well as, for example, their mutual occlusion patterns is very different between the two tasks. This issue has been also called "a dataset capture bias" by Torralba and Efros [2011]. In addition, the target task may contain many other objects in the background that are not present in the source task training data (a "negative data bias" in Torralba and Efros [2011]). To explicitly address these issues we train the adaptation layer using a procedure inspired by training sliding window object detectors (e.g. Felzenszwalb et al. [2008]) described next.

Figure 3-4: Example cars from the ImageNet dataset. The cars are generally centered and scaled to the size of the image.

Figure 3-5: Example cars from the Pascal VOC dataset, as well as bounding boxes (in yellow) describing the extent of different objects. The cars appear within complex scenes and at various scales.

**Patch extraction.** We illustrate this procedure with the example of cars. We can see in Figures 3-4 and 3-5 that the collection method of these datasets reflects on the appearance of objects in examples.

Therefore, we employ a sliding window strategy and extract around 500 square patches from each image by sampling eight different scales on a regularly-spaced grid with at least 50% overlap between neighboring patches. More precisely, we use square patches of width $s = \min(w, h)/\lambda$ pixels, where $w$ and $h$ are the width and height of the image, respectively, and $\lambda \in \{1, 1.3, 1.6, 2, 2.4, 2.8, 3.2, 3.6, 4\}$. Each patch is rescaled to $224 \times 224$ pixels to form a valid input for the network.

Tiling the image allows matching the average appearance of our source ImageNet

dataset for the cars, as shown in Figure 3-6. As a result, we obtain many patches, as shown in Figure 3-7 that we need to label appropriately.

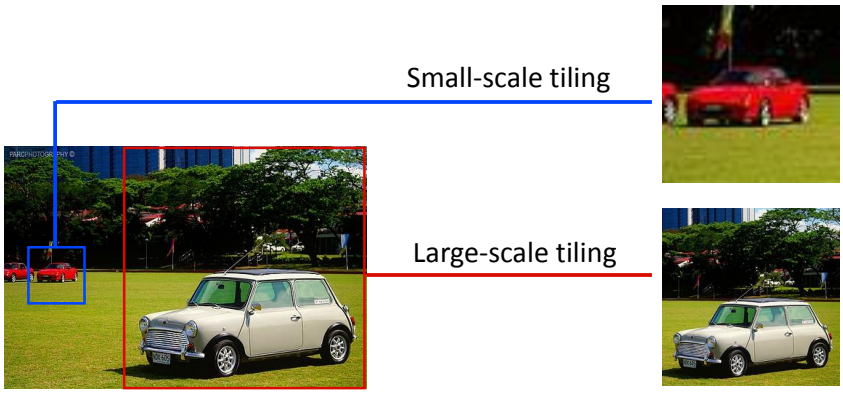



Figure 3-6: Tiling an image at multiple scales builds patches where objects are centered and scaled, closer to the average appearance in ImageNet.

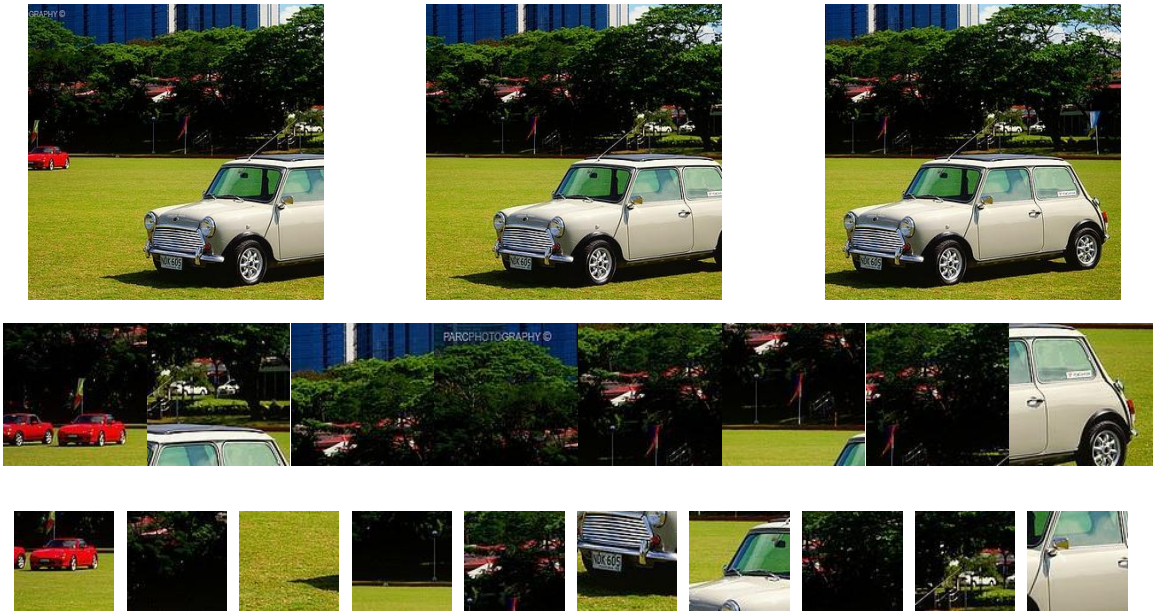


Figure 3-7: As a result of our tiling procedure, we obtain a large number of patches that we need to label appropriately.

**Patch labeling.** Sampled image patches may contain one or more objects, background, or only a part of the object. To label patches in training images, we measure the overlap between the bounding box of a patch $P$ and ground truth bounding boxes $B$ of annotated objects in the image. The patch is labelled as a positive training example for class $o$ if there exists a box $B_o$ corresponding to class $o$ such that:

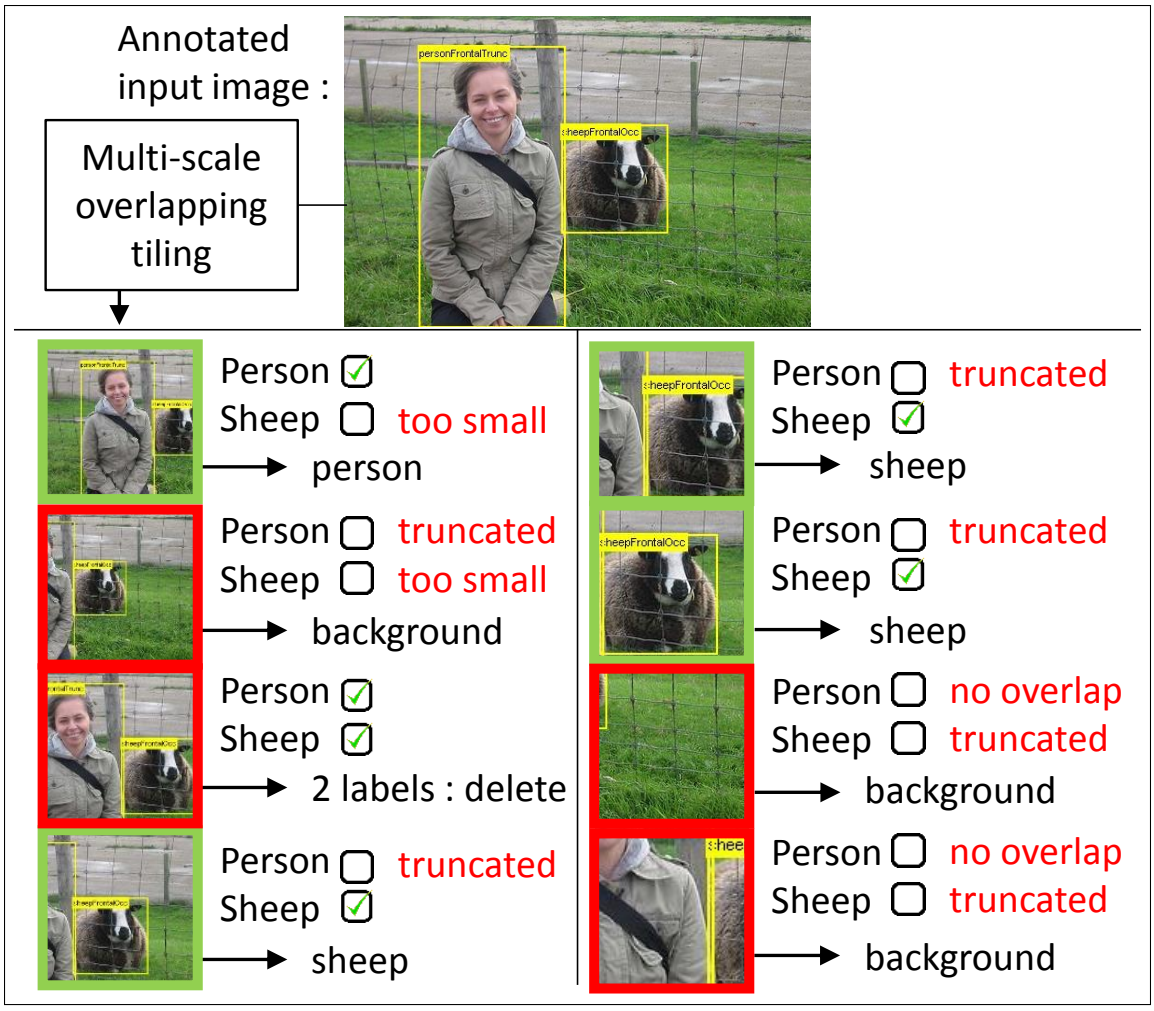



Figure 3-8: **Generating training data for the target task.** The input image (top) is divided into multi-scale overlapping patches (bottom). Each patch is labelled with an object label (green) or as background (red) depending on the overlap with object bounding boxes. Note that object patches are similar in appearance to the training data for the source task containing mostly centered objects.

- $B_o$ overlaps sufficiently with the patch $|P \cap B_o| \geq 0.2|P|$,
- the patch contains a large portion of the object $|P \cap B_o| \geq 0.6|B_o|$,
- the patch overlaps with no more than one object.

In the above definitions $|A|$ measures the area of the bounding box $A$. Our labeling criteria are illustrated in Figure 3-8.

**Dealing with background.** As discussed above, the target task has an additional `background` label for patches that do not contain any object. One additional difficulty is that the training data is unbalanced: most patches from training images come from background. This can be addressed by re-weighting the training cost function, which would amount to re-weighting its gradients during training. We opt for a slightly different procedure and instead re-sample the training patches to balance the training data distribution. This resampled training set is then used to form mini-batches for the stochastic gradient descent training. This is implemented by sampling a random 10% of the training background patches.

### 3.2.3 Classification

At test time we apply the network to each of the (approximately) 500 overlapping multi-scale patches extracted from the test image. Examples of patch scores visualized over entire images are shown in Figures 3-1 and 3-9 - 3-10. We use the following aggregation formula to compute the overall score for object $C_n$ in the image

$$\text{score}(C_n) = \frac{1}{M} \sum_{i=1}^{M} y(C_n|P_i)^k, \tag{3.1}$$

where $y(C_n|P_i)$ is the output of the network for class $C_n$ on image patch $P_i$, $M$ is the number of patches in the image, and $k \geq 1$ is a parameter. Higher values of $k$ focus on the highest scoring patches and attenuate the contributions of low- and mid-scoring patches. The value of $k = 5$ was optimized on the validation set and is fixed in our experiments.

Note that patch scores could be computed much more efficiently by performing large convolutions on adequately subsampled versions of the full image, as described for instance in Farabet et al. [2013]. This would permit a denser patch coverage at a lower computation cost.

## 3.3 Experiments

In this section we first describe details of training, and discuss pre-training results for the source task of ImageNet object classification. We next show experimental results of the proposed transfer learning method on the target Pascal VOC object classification task for both VOC 2007 and VOC 2012 datasets. We also investigate the dependency of results on the overlap of source and target tasks by object classes. Finally, we apply the proposed transfer learning method on a very different task of action recognition in still images.

**Training convolutional networks.** All our training sessions were carried out using the code provided by Krizhevsky et al. [2012] and replicating their exact dropout and jittering strategies. However, we do not alter the RGB intensities and we use a single GeForce GTX Titan GPU with 6GB of memory instead of the two GPUs of earlier generation as originally used. The training procedure periodically evaluates the cross-entropy objective function on a subset of the training set and on a validation set. The initial learning rates are set to 0.01 and the network is trained until the training cross-entropy is stabilized. The learning rates are then divided by 10 and the training procedure repeats. We stop training after three iterations. We have not tuned parameters for this part of the algorithm and we did not observe overfitting on the validation set.

**Image classification on ImageNet.** We first train a single convolutional network on the 1000 classes and 1.2 million images of the ImageNet 2012 Large Scale Visual Recognition Challenge (ILSVRC-2012). This network has exactly the same structure as the network described in Krizhevsky et al. [2012]. Layers FC6 and FC7 have 4096 units. Training lasts about one week. The resulting network achieves a 18% top-5 error rate[1], comparable to the 17% reported by Krizhevsky et al. [2012] for a single network. This slight performace loss could be caused by the absence of RGB intensity manipulation in our experiments.

[1] 5 guesses are allowed.

| | plane | bike | bird | boat | btl | bus | car | cat | chair | cow | table | dog | horse | moto | pers | plant | sheep | sofa | train | tv | **mAP** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INRIA | 77.5 | 63.6 | 56.1 | 71.9 | 33.1 | 60.6 | 78.0 | 58.8 | 53.5 | 42.6 | 54.9 | 45.8 | 77.5 | 64.0 | 85.9 | 36.3 | 44.7 | 50.6 | 79.2 | 53.2 | 59.4 |
| NUS-PSL | 82.5 | 79.6 | 64.8 | 73.4 | **54.2** | 75.0 | 77.5 | 79.2 | 46.2 | 62.7 | 41.4 | 74.6 | **85.0** | 76.8 | 91.1 | 53.9 | 61.0 | **67.5** | 83.6 | 70.6 | 70.5 |
| PRE-1000C | **88.5** | **81.5** | **87.9** | **82.0** | 47.5 | **75.5** | **90.1** | **87.2** | **61.6** | **75.7** | **67.3** | **85.5** | 83.5 | **80.0** | **95.6** | **60.8** | **76.8** | 58.0 | **90.4** | **77.9** | **77.7** |

Table 3.1: Per-class results for object classification on the VOC2007 test set (average precision %). INRIA relates to the work of Marszalek et al. [2007], and NUS-PSL to the work of Song et al. [2011].

| | plane | bike | bird | boat | btl | bus | car | cat | chair | cow | table | dog | horse | moto | pers | plant | sheep | sofa | train | tv | **mAP** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NUS-PSL | **97.3** | **84.2** | 80.8 | **85.3** | **60.8** | **89.9** | **86.8** | 89.3 | **75.4** | 77.8 | **75.1** | 83.0 | 87.5 | **90.1** | 95.0 | 57.8 | 79.2 | **73.4** | **94.5** | **80.7** | 82.2 |
| NO PRETRAIN | 85.2 | 75.0 | 69.4 | 66.2 | 48.8 | 82.1 | 79.5 | 79.8 | 62.4 | 61.9 | 49.8 | 75.9 | 71.4 | 82.7 | 93.1 | 59.1 | 69.7 | 49.3 | 80.0 | 76.7 | 70.9 |
| PRE-1000C | 93.5 | 78.4 | *87.7* | 80.9 | 57.3 | 85.0 | 81.6 | *89.4* | 66.9 | 73.8 | 62.0 | *89.5* | 83.2 | 87.6 | *95.8* | *61.4* | 79.0 | 54.3 | 88.0 | 78.3 | 78.7 |
| PRE-1000R | 93.2 | 77.9 | 83.8 | 80.0 | 55.8 | 82.7 | 79.0 | 84.3 | 66.2 | 71.7 | 59.5 | 83.4 | 81.4 | 84.8 | 95.2 | 59.8 | 74.9 | 52.9 | 83.8 | 75.7 | 76.3 |
| PRE-1512 | 94.6 | 82.9 | **88.2** | 84.1 | 60.3 | 89.0 | 84.4 | **90.7** | 72.1 | **86.8** | 69.0 | **92.1** | **93.4** | 88.6 | **96.1** | **64.3** | **86.6** | 62.3 | 91.1 | 79.8 | **82.8** |

Table 3.2: Per-class results for object classification on the VOC2012 test set (average precision %). NUS-PSL relates to the work of Yan et al. [2012].

**Image classification on Pascal VOC 2007.** We apply our mid-level feature transfer scheme to the Pascal VOC 2007 object classification task. Results are reported in Table 3.1. Our transfer technique (PRE-1000C) demonstrates significant improvements over previous results on this data outperforming the 2007 challenge winners Marszalek et al. [2007] (INRIA) by 18.3% and the more recent work of Song et al. [2011] (NUS-PSL) by 7.2%.

**Image classification on Pascal VOC 2012.** We next apply our method to the Pascal VOC 2012 object classification task. Results are shown in the row PRE-1000C of Table 4.1. Although these results are on average about 4% inferior to those reported by the winners of the 2012 challenge (NUS-PSL - Yan et al. [2012]), our method outperforms Yan et al. [2012] on five out of twenty classes. To estimate the performance boost provided by the feature transfer, we compare these results to the performance of an identical network directly trained on the Pascal VOC 2012 training data (NO PRETRAIN) without using any external data from ImageNet. Notably, the performance drop of nearly 8% in the case of NO PRETRAIN clearly indicates the positive effect of the proposed transfer.

**Transfer learning and source/target class overlap.** Our source ILSVRC-2012 dataset contains target-related object classes, in particular, 59 species of birds and 120 breeds of dogs related to the `bird` and `dog` classes of Pascal VOC. To understand the influence of this overlap on our results, we have pre-trained the network on a source task data formed by 1,000 ImageNet classes selected, this time, *at random* among all the 22,000 available ImageNet classes. Results of this experiment are reported in Table 4.1, row PRE-1000R. The overall performance has decreased slightly, indicating that the overlap between classes in the source and target domains may have a positive effect on the transfer. Given the relatively small performance drop, however, we conclude that our transfer procedure is robust to changes of source and target classes. As the number of training images in this experiment was about 25% smaller than in the ILSVRC-2012 training set (PRE-1000C), this could have been another reason for

the decrease of performance.

Conversely, we have augmented the 1,000 classes of the ILSVRC-2012 training set with 512 additional ImageNet classes selected to increase the overlap with specific classes in the Pascal VOC target task. We included all the ImageNet classes located below the `hoofed mammal` (276 classes), `furniture` (165), `motor vehicle` (48), `public transport` (18), `bicycle` (5) nodes of the WordNet hierarchy. In order to accommodate the larger number of classes, we also increased the size of the FC6 and FC7 layers from 4,096 to 6,144 dimensions. Training on the resulting 1.6 million images achieves a 21.8% top-5 error rate on the 1,512 classes. Using this pre-trained network we have obtained further improvements on the target task, outperforming the winner of Pascal VOC 2012 (Yan et al. [2012]) on average (row PRE-1512 in Table 4.1). In particular, improvements are obtained for categories (`cow`, `horse`, `sheep`, `sofa`, `chair`, `table`) related to the added classes in the source task. By comparing results for PRE-1000R, PRE-1000C and PRE-1512 setups, we also note the consistent improvement of *all* target classes. This suggests that the number of images and classes in the source task might be decisive for the performance in the target task. Hence, we expect further improvements by our method using larger source tasks.

**Varying the number of adaptation layers.** We have also tried to change the number of adaptation layers in the best performing PRE-1512 training set-up. Using only one fully connected adaptation layer FCb of size 21 (the number of categories) results in about 1% drop in performance. Similarly, increasing the number of adaptation layers to three (of sizes 2048, 2048 and 21, respectively) also results in about 1% drop in classification performance.

**Object localization.** Although our method has not been explicitly designed for the task of localization, we have observed strong evidence of object and action localization provided by the network at test time. For qualitative assessment of localization results, we compute an output map for each category by averaging the scores of all the testing patches covering a given pixel of the test image. Examples of such output

| Action | jump | phon | instr | read | bike | horse | run | phot | comp | walk | **mAP** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| STANFORD pas [2012] | 75.7 | 44.8 | 66.6 | 44.4 | 93.2 | 94.2 | 87.6 | 38.4 | **70.6** | **75.6** | 69.1 |
| OXFORD pas [2012] | **77.0** | **50.4** | 65.3 | 39.5 | **94.1** | **95.9** | **87.7** | 42.7 | 68.6 | 74.5 | 69.6 |
| NO PRETRAIN | 43.2 | 30.6 | 50.2 | 25.0 | 76.8 | 80.7 | 75.2 | 22.2 | 37.9 | 55.6 | 49.7 |
| PRE-1512 | 73.4 | 44.8 | **74.8** | 43.2 | 92.1 | 94.3 | 83.4 | **45.7** | 65.5 | 66.8 | 68.4 |
| PRE-1512U | 74.8 | 46.0 | **75.6** | **45.3** | 93.5 | 95.0 | 86.5 | **49.3** | 66.7 | 69.5 | **70.2** |

Table 3.3: Pascal VOC 2012 action classification results (AP %).

maps are given in Figures 3-1 and 3-9 - 3-10. This visualization clearly demonstrates that the system knows the size and locations of target objects within the image. Addressing the detection task seems within reach, as has been, in parallel to this work, explored with the R-CNN algorithm (Girshick et al. [2014]).

**Action recognition.** The Pascal VOC 2012 action recognition task consists of 4588 training images and 4569 test images featuring people performing actions among ten categories such as `jumping`, `phoning`, `playing instrument` or `reading`. This fine-grained task differs from the object classification task because it entails recognizing fine differences in human poses (e.g. `running` v.s. `walking`) and subtle interactions with objects (`phoning` or `taking photo`). Training samples with multiple simultaneous actions are excluded from our training set.

To evaluate how our transfer method performs on this very different target task, we use a network pre-trained on 1512 ImageNet object classes and apply our transfer methodology to the Pascal VOC action classification task. Since the bounding box of the person performing the action is known at testing time, both training and testing are performed using a single square patch per sample, centered on the person bounding box. Extracting the patch possibly involves enlarging the original image by mirroring pixels. The results are summarized in row PRE-1512 (Table 3.3). The transfer method significantly improves over the NO PRETRAIN baseline where the CNN is trained solely on the action images from Pascal VOC, without pretraining on ImageNet. In particular, we obtain best results on challenging categories `playing instrument` and `taking photo`.

In order to better adapt the CNN to the subtleties of the action recognition task, and inspired by Collobert et al. [2011b], our last results were obtained by training

the target task CNN without freezing the FC6 weights. More precisely, we copy the ImageNet-trained weights of layers C1...C5, FC6 and FC7, we append the adaptation layers FCa and FCb, and we retrain layers FC6, FC7, FCa, and FCb on the action recognition data. This strategy increases the performance on all action categories (row PRE-1512U in Table 3.3), yielding, to the best of our knowledge, the best average result published on the Pascal VOC 2012 action recognition task.

To demonstrate that we can also localize the action in the image, we train the network in a sliding window manner, as described in Section 3.2. In particular, we use the ground truth person bounding boxes during training, but do not use the ground truth person bounding boxes at test time. Example output maps shown in Figures 3-9 - 3-10 clearly demonstrate that the network provides an estimate of the action location in the image.

**Failure modes.** Top-ranked false positives in Figures 3-9 - 3-10 correspond to samples closely resembling target object classes. Resolving some of these errors may require high-level scene interpretation. Our method may also fail to recognize spatially co-occurring objects (e.g., person on a chair) since patches with multiple objects are currently excluded from training. This issue could be addressed by changing the training objective to allow multiple labels per sample. Recognition of very small or very large objects could also fail due to the sparse sampling of patches in our current implementation. As mentioned in Section 4.5 this issue could be resolved using a more efficient CNN-based implementation of sliding windows.

## 3.4 Conclusion and discussion

Building on the performance leap achieved by Krizhevsky et al. [2012] on ILSVRC-2012, we have shown how a simple transfer learning procedure yields state-of-the-art results on challenging benchmark datasets of much smaller size. We have also demonstrated the high potential of the mid-level features extracted from an ImageNet-trained CNNs. Although the performance of this setup increases when we augment

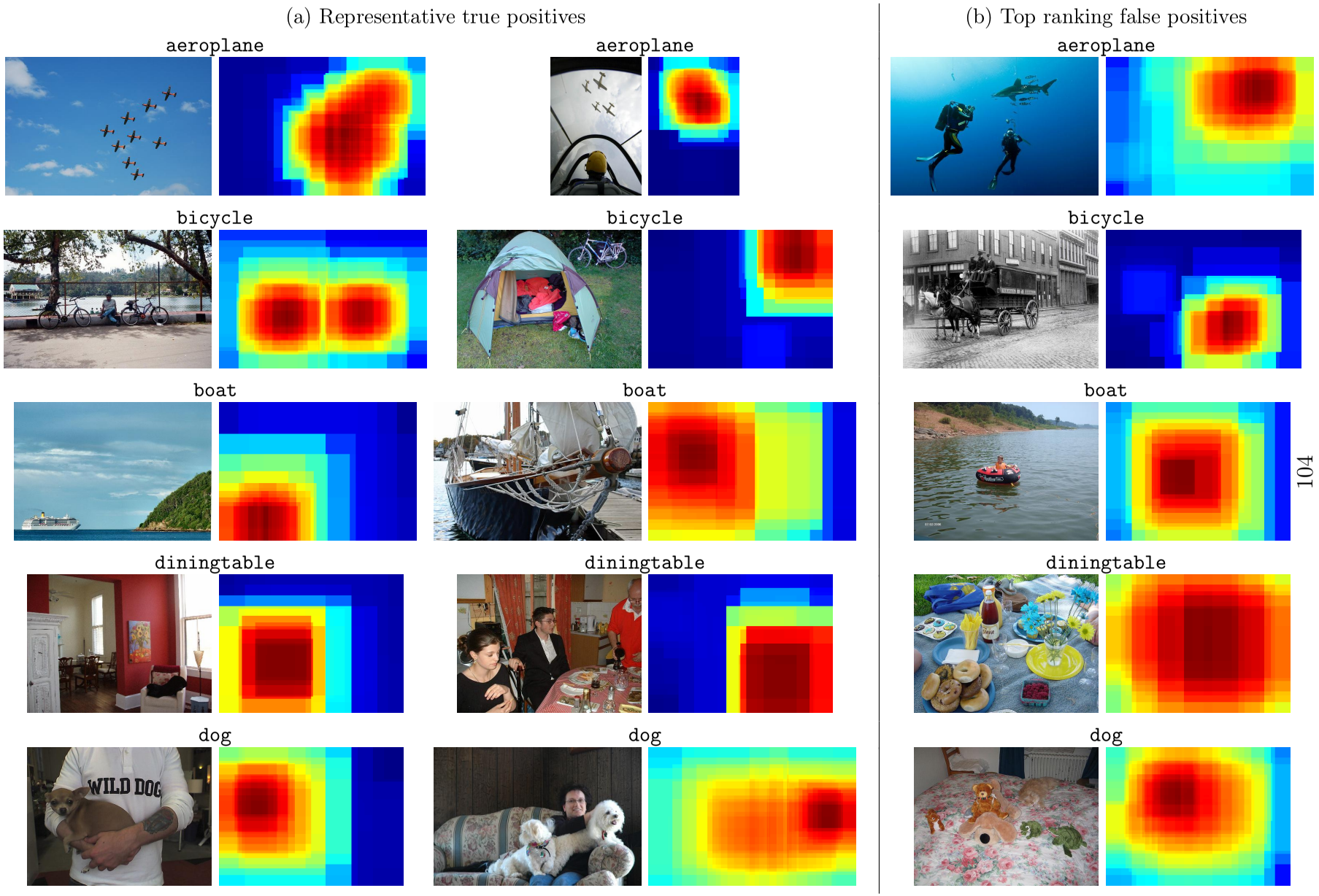



Figure 3-9: Response maps on representative images of several categories of the VOC 2012 object and action classification test set. The rightmost column contains the highest-scoring false positive (according to our judgement) for each of these categories. Note that correct estimates of object and action locations and scales are provided by the score maps.

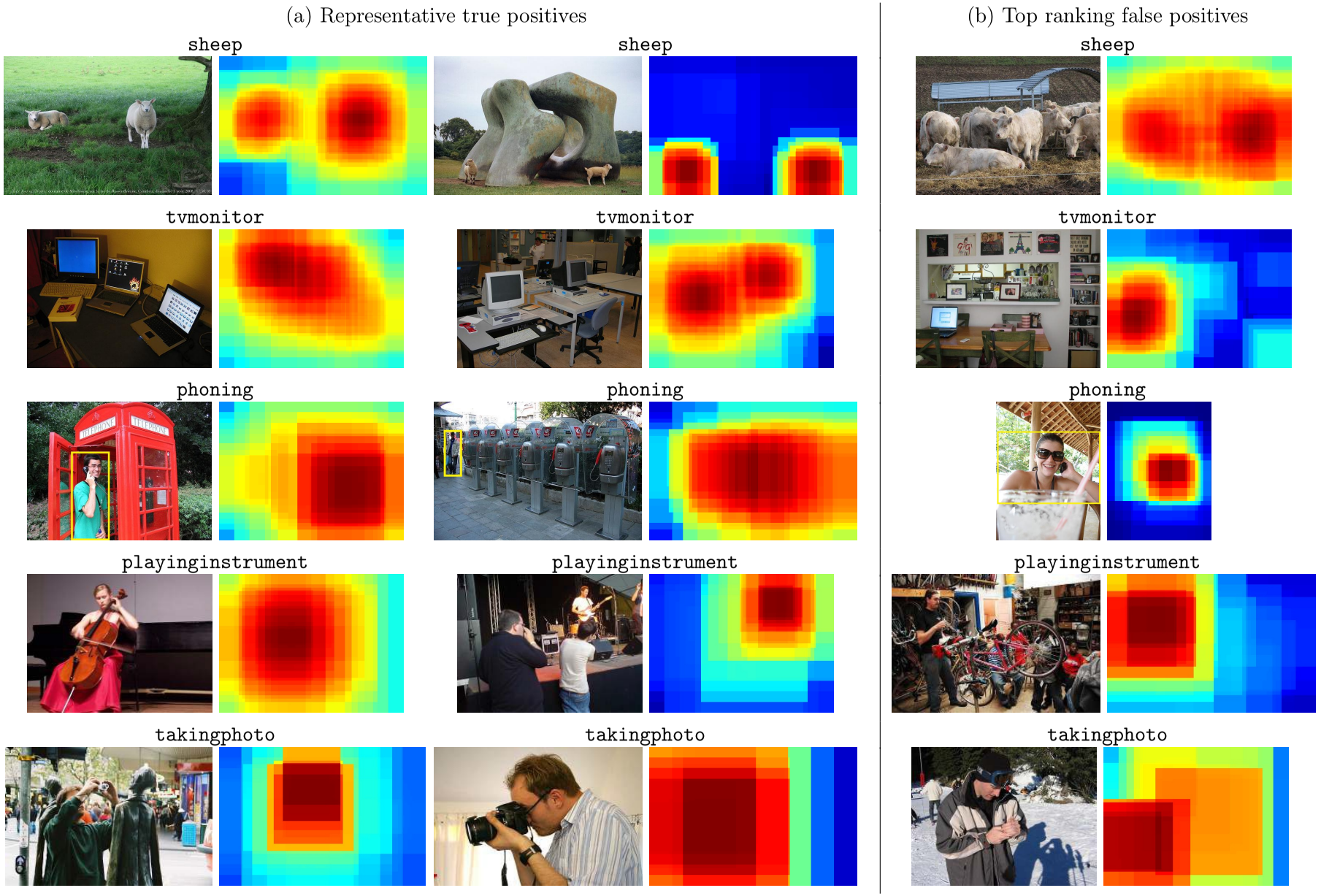



Figure 3-10: Response maps on representative images of several categories of the VOC 2012 object and action classification test set. The rightmost column contains the highest-scoring false positive (according to our judgement) for each of these categories. Note that correct estimates of object and action locations and scales are provided by the score maps.

the source task data, using only 12% of the ImageNet corpus already leads to the best published results on the Pascal VOC 2012 classification and action recognition tasks.

**Recent developments.** This work shows that manually engineered image representations can be outperformed with relative ease using neural networks. Moreover, the performance scales up with data.

Concurrently with this work, other papers showed the generality of CNN features, their potential for learning rich mid-level image features transferrable to a variety of visual recognition tasks, and advocated their wide usage in computer vision. Therefore, this chapter is part of a large body of work following the ImageNet breakthrough of Krizhevsky et al. [2012], modernizing the field of computer vision by intersecting it with the powerful features of neural networks.

1. Zeiler and Fergus [2014] do an early study of the AlexNet architecture and show the patterns that activate neurons in each layer, sharing their intuitions and understanding on CNNs for the vision community to catch up quickly with the techniques.

2. Girshick et al. [2014] use the output of the FC7 layer as a feature vector and connect it to an approach using Selective Search (van de Sande et al. [2011]) to outperform object detection algorithms.

3. Razavian et al. [2014] do a survey and advocate the use of CNNs as a feature transform available "off-the-shelf", allowing replacing SIFT/HoG and their Fisher Vector representations with CNN feature vectors collected in the FC layers.

4. Sermanet et al. [2014] propose open-source software for extracting features on images based on their ILSVRC-2013 model.

5. Jia et al. [2014] and Donahue et al. [2014] propose a framework for sharing pre-trained models and using them for feature extraction: the *Caffe Model Zoo* contains many pre-trained CNN models that helped popularize this technique.

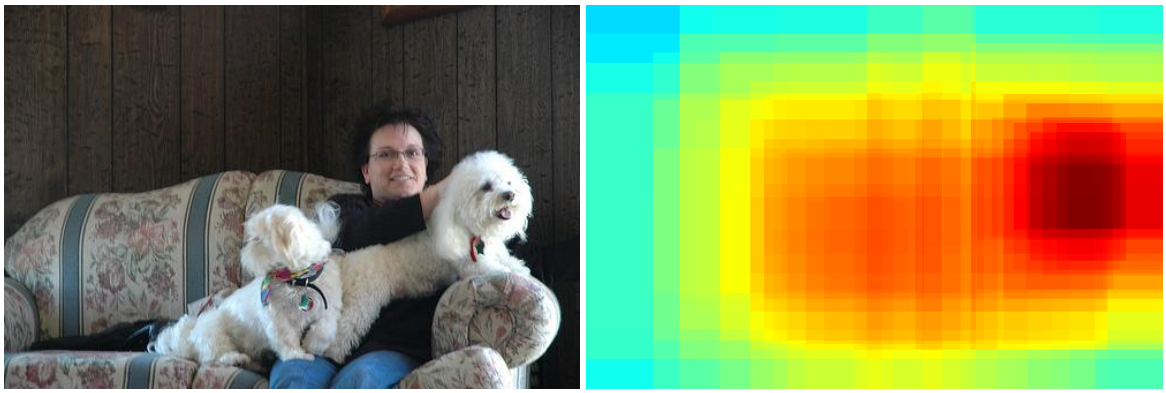


Figure 3-11: In this last example, we show the *dog* classification score map, and notice that the response is much stronger for a distinctive part such as a front-view of the head, than for the fur. We build on this observation in the next chapter.

# Chapter 4

# Is object localization for free? Weakly-supervised learning with convolutional neural networks

In this chapter, we build on an observation from the previous chapter on feature learning and transfer. Heat maps for the dog class in Figure 3-11 (Chapter 3) show that the neural network classifier is more sensitive to the head of a dog than to its fur. We conclude that patches used for training are not equally important for the classification. Based on this observation, in this chapter, we design a method that lets the neural network select that most discriminative patch that results in the best classification performance, demonstrating that the neural networks have naturally strong output on statistically relevant patterns for image classification. We exploit this mechanism to localize objects in images by retrieving the image patch that activates a classifier maximally. Adding this degree of liberty to the classifier lets us achieve state-of-the-art results on the Pascal VOC classification task. Moreover, we provide quantitative evidence that neural networks are sensitive to object location in an exploitable way. Notably, we train our architecture using only image-level annotations but show that the trained model can localize objects in images. The supervision in our setup is weaker than the output it provides: we demonstrate a form of weakly-supervised training; example results are shown at the end of this chapter, in Figure 4-6.

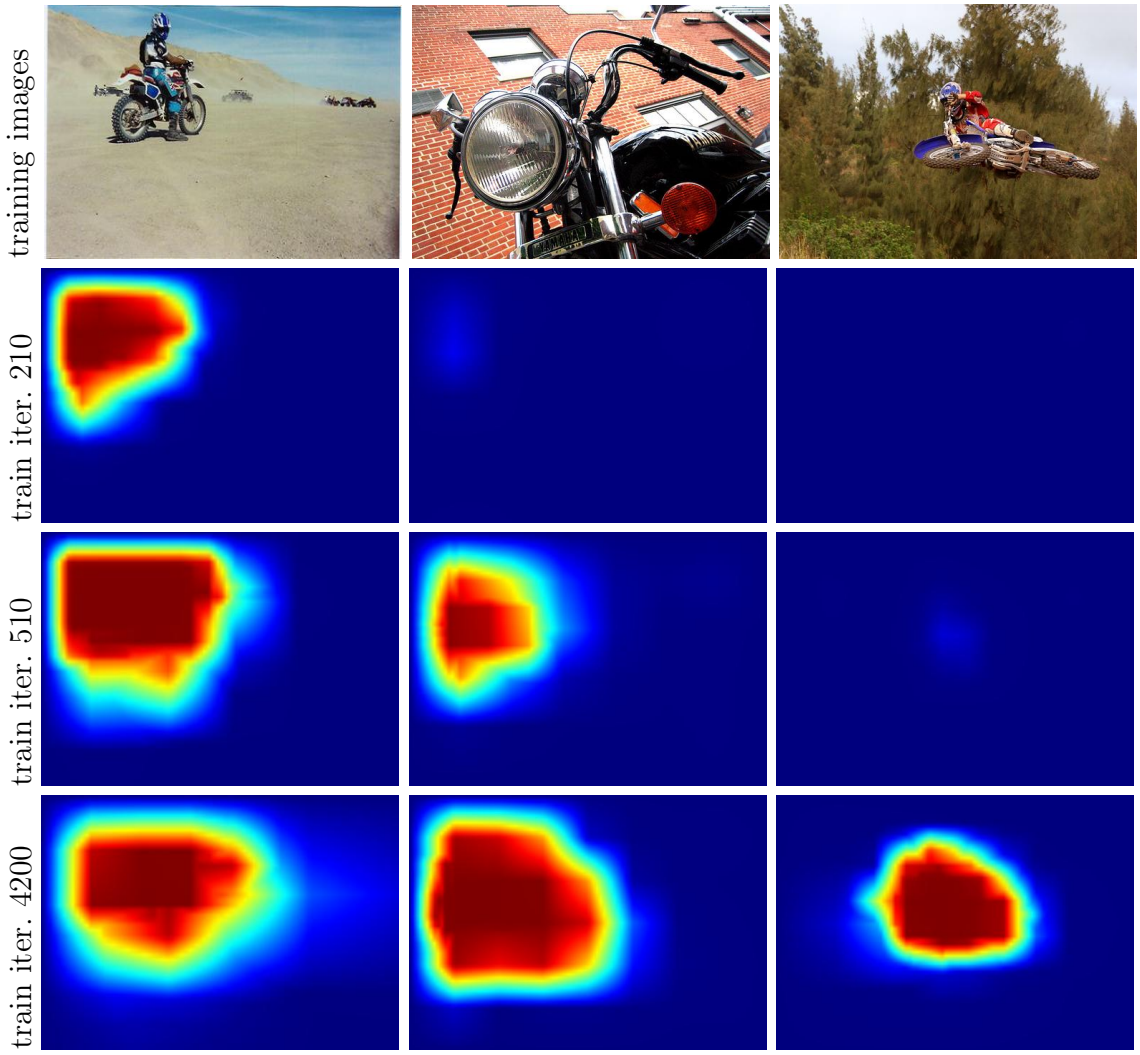



Figure 4-1: Evolution of localization score maps for the `motorbike` class over iterations of our weakly-supervised CNN training. Note that the network learns to localize objects despite having no object location annotation at training, just object presence/absence labels. Note also that locations of objects with more usual appearance (such as the motorbike shown in left column) are discovered earlier during training.

## 4.1 Introduction

Visual object recognition entails much more than determining whether the image contains instances of certain object categories. For example, each object has a location and a pose; each deformable object has a constellation of parts; and each object can be cropped or partially occluded.

Object recognition algorithms of the 2000s can roughly be categorized in two

styles. The first style extracts local image features (SIFT, HOG), constructs *bag of visual words* representations, and runs statistical classifiers( Csurka et al. [2004], Perronnin et al. [2010], Sivic and Zisserman [2003], Zhang et al. [2007]). Although this approach has been shown to yield good performance for image classification, attempts to locate the objects using the position of the visual words have been unfruitful: the classifier often relies on visual words that fall in the background and merely describe the context of the object.

The second style of algorithms detects the presence of objects by fitting rich object models such as *deformable part models*( Felzenszwalb et al. [2008], Yang and Ramanan [2011]). The fitting process can reveal useful attributes of objects such as location, pose and constellations of object parts, but the model is usually trained from images with known locations of objects or even their parts. The combination of both styles has shown benefits( Harzallah et al. [2009]).

A third style of algorithms, *convolutional neural networks* (CNNs, Lang and Hinton [1988], LeCun et al. [1989]) construct successive feature vectors that progressively describe the properties of larger and larger image areas. Recent applications of this framework to natural images (Krizhevsky et al. [2012]) have been extremely successful for a variety of tasks including image classification (Chatfield et al. [2014], Krizhevsky et al. [2012], Oquab et al. [2014], Razavian et al. [2014], Sermanet et al. [2014]), object detection (Girshick et al. [2014], Sermanet et al. [2014]), human pose estimation (Toshev and Szegedy [2014]) and others. Most of these methods, however, require detailed image annotation. For example bounding box supervision has been shown highly beneficial for object classification in cluttered and complex scenes (Chapter 3).

**The labelling issue.** Labelling a set of training images with object attributes quickly becomes problematic. The process is expensive and involves a lot of subtle and possibly ambiguous decisions. For instance, consistently annotating locations and

scales of objects by bounding boxes works well for some images but fails for partially occluded and cropped objects. Annotating object parts becomes even harder since the correspondence of parts among images in the same category is often ill-posed.

**Goal.** In this chapter, we investigate whether CNNs can be trained from complex cluttered scenes labelled only with lists of objects they contain and not their locations. This is an extremely challenging task as the objects may appear at different locations, different scales and under variety of viewpoints, as illustrated in Figure 4-1 (top row). Furthermore, the network has to avoid overfitting to the scene clutter co-occurring with objects as, for example, motorbikes often appear on the road. How can we modify the structure of the CNN to learn from such difficult data?

**Method.** We build on the successful CNN architecture of Krizhevsky et al. [2012] and the follow-up state-of-the-art results for object classification and detection, but introduce the following modifications.

1. We treat the last fully connected network layers as convolutions to cope with the uncertainty in object localization.

2. We introduce a max-pooling layer that hypothesizes the possible location of the object in the image, similar to [Lang et al., 1990, Section 4] and Keeler et al. [1991].

3. We modify the cost function to learn from image-level supervision.

Interestingly, we find that this modified CNN architecture, while trained to output image-level labels only, localizes objects or their distinctive parts in training images, as illustrated in Figure 4-1. *So, is object localization with convolutional neural networks for free?* In this chapter we set out to answer this question and analyze the developed weakly supervised CNN pipeline on two object recognition datasets containing complex cluttered scenes with multiple objects.

**Contributions.** The contributions of this work are twofold. First, we develop a weakly supervised convolutional neural network end-to-end learning pipeline that learns from complex cluttered scenes containing multiple objects by explicitly searching over possible object locations and scales in the image. Second, we perform an extensive experimental analysis of the network's classification and localization performance on the Pascal VOC 2012 and the much larger Microsoft COCO datasets. We find that our weakly-supervised network

- outputs accurate image-level labels,
- predicts approximate locations (but not extents) of objects, and
- performs comparably to its fully-supervised counterparts that use object bounding box annotation for training.

## 4.2 Architecture for weakly supervised learning

We build on the fully supervised network architecture described in Chapter 3, that consists of five convolutional and four fully connected layers and assumes as input a fixed-size image patch containing a single relatively tightly cropped object. To adapt this architecture to weakly supervised learning we introduce the following three modifications.

First, we treat the fully connected layers as convolutions, which allows us to deal with nearly arbitrary-sized images as input. Second, we explicitly search for the highest scoring object position in the image by adding a single global max-pooling layer at the output. Third, we use a cost function that can explicitly model multiple objects present in the image.

The three modifications are discussed next and the network architecture is illustrated in Figure 4-2.

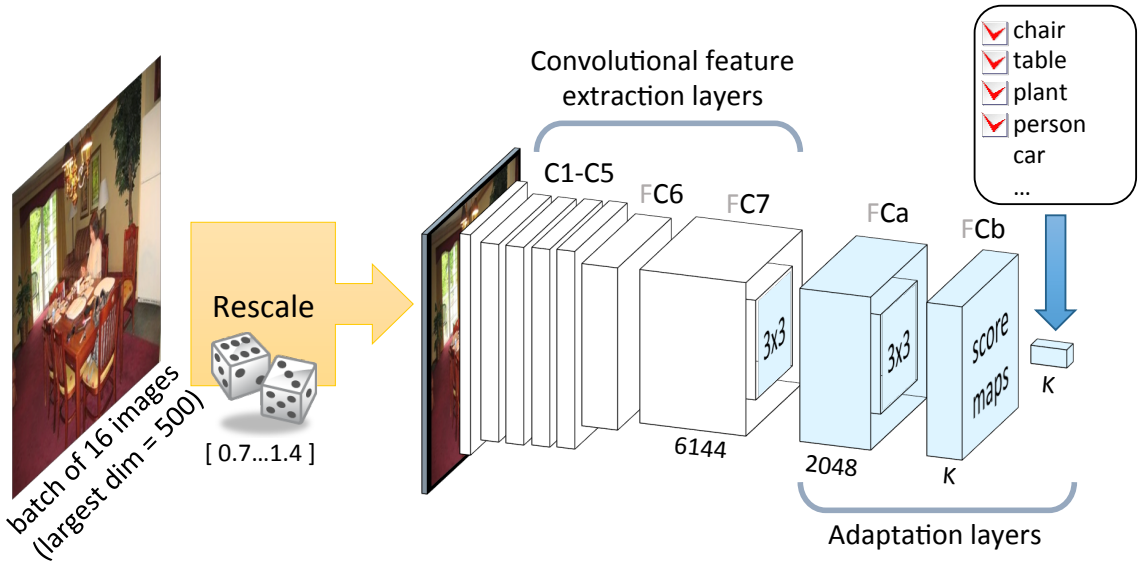



Figure 4-2: Network architecture for weakly supervised training. We convert the Fully Connected layers FCa and FCb to the corresponding Convolution layers Ca and Cb. The score maps are then globally pooled: only the largest output is kept as an image-level class score.

### 4.2.1 Convolutional adaptation layers

The network architecture of Chapter 3 assumes a fixed-size image patch of 224×224 RGB pixels as input and outputs a $1 \times 1 \times K$ vector of per-class scores as output, where $K$ is the number of classes. The aim is to apply the network to bigger images in a sliding window manner thus extending its output to $n \times m \times K$ where $n$ and $m$ denote the number of sliding window positions in the $x$- and $y$- direction in the image, respectively, computing the $K$ per-class scores at all input window positions.

While this type of sliding was performed in Chapter 3 by applying the network to independently extracted image patches, here we achieve the same effect by treating the fully connected adaptation layers as convolutions. For a given input image size, the fully connected layer can be seen as a special case of a convolution layer where the size of the kernel is equal to the size of the layer input. With this procedure the output of the final adaptation layer FC7 becomes a $2 \times 2 \times K$ output score map for a $256 \times 256$ RGB input image.

As the global stride of the network is 32[1] pixels, adding 32 pixels to the image

[1] or 36 pixels for the OverFeat network that we use on MS COCO

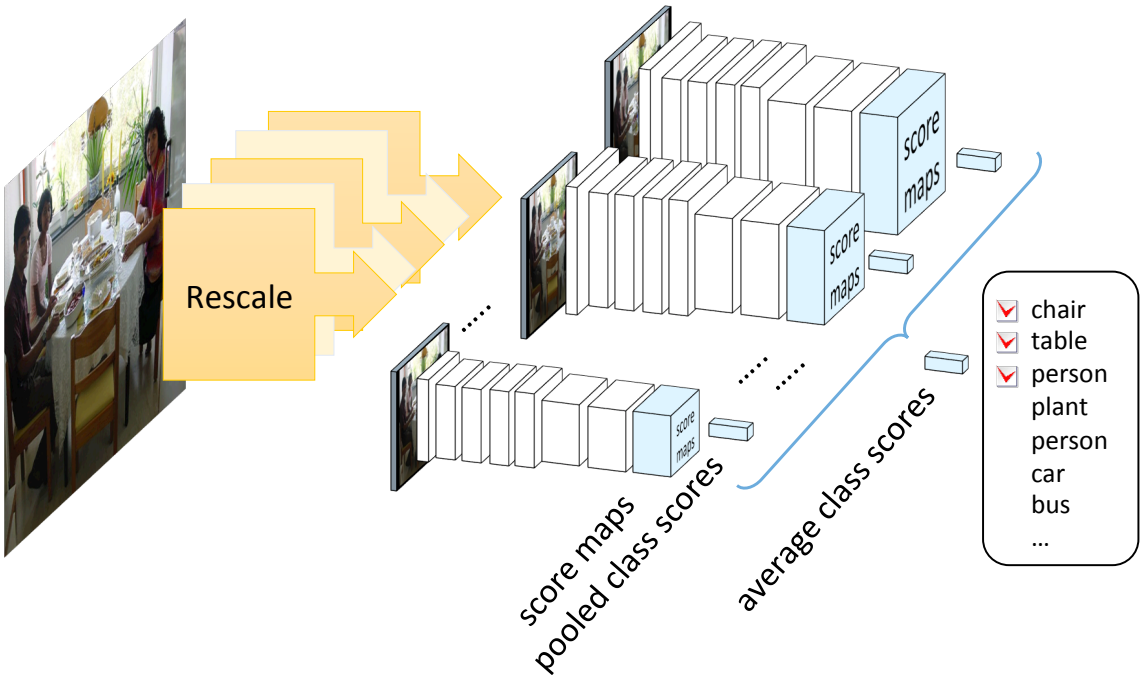



Figure 4-3: Multiscale object recognition.

width or height increases the width or height of the output score map by one. Hence, for example, a $2048 \times 1024$ pixel input would lead to a $58 \times 26$ output score map containing the score of the network for all classes for the different locations of the input $224 \times 224$ window with a stride of 32 pixels.

While this architecture is typically used for efficient classification at test time, see e.g. Sermanet et al. [2014], here we also use it at training time (as discussed in Section 4.3) to efficiently examine the entire image for possible locations of the object during weakly supervised training.

### 4.2.2 Explicit search for object's position via max-pooling

The aim is to output a single image-level score for each of the object classes independently of the input image size. This is achieved by aggregating the $n \times m \times K$ matrix of output scores for $n \times m$ different positions of the input window using a global max-pooling operation into a single $1 \times 1 \times K$ vector, where $K$ is the number of classes. Note that the max-pooling operation effectively searches for the best-scoring candidate object position within the image, which is crucial for weakly supervised learning

where the exact position of the object within the image is not given at training. In addition, due to the max-pooling operation the output of the network becomes independent of the size of the input image, which will be used for multi-scale learning in Section 4.3.

### 4.2.3 Multi-label classification loss function

The goal of object classification is to tell whether an instance of an object class is present in the image, where the input image may depict multiple different objects. As a result, the usual multi-class mutually exclusive logistic regression loss, as used in e.g. Krizhevsky et al. [2012] for ImageNet classification, is not suited for this set-up as it assumes only a single object per image. To address this issue, we treat the task as a separate binary classification problem for each class. The loss function is therefore a sum of K binary logistic regression losses, one for each of the K classes $k \in \{1 \cdots \mathrm{K}\}$,

$$\ell(\ f_k(\mathbf{x})\ ,\ y_k\ )\ =\ \sum_k \log(1 + e^{-y_k f_k(\mathbf{x})})\ , \tag{4.1}$$

where $f_k(\mathbf{x})$ is the output of the network for input image $\mathbf{x}$ and $y_k \in \{-1, 1\}$ is the image label indicating the absence/presence of class $k$ in the input image $\mathbf{x}$. Each class score $f_k(\mathbf{x})$ can be interpreted as a posterior probability indicating the presence of class $k$ in image $\mathbf{x}$ with transformation

$$P(k|\mathbf{x}) \approx \frac{1}{1 + e^{-f_k(\mathbf{x})}}\ . \tag{4.2}$$

Treating a multi-label classification problem as K independent classification problems is often inadequate because it does not model label correlations. This is not an issue here because the classifiers share hidden layers and therefore are not independent. Such a network can model label correlations by tuning the overlap of the hidden state distribution given each label.

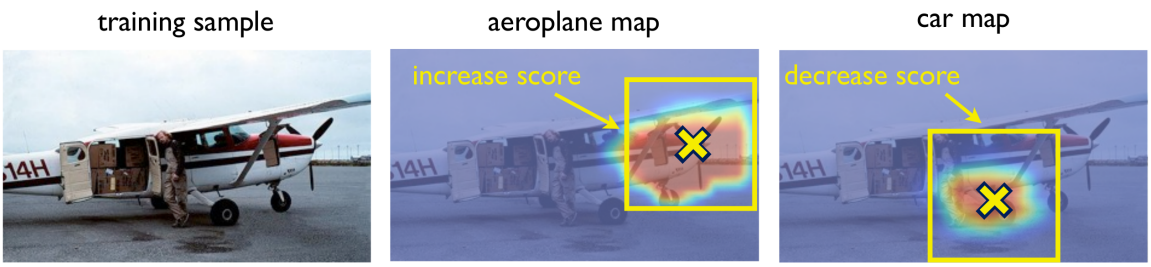



Figure 4-4: **Illustration of the weakly-supervised learning procedure.** At training time, given an input image with an aeroplane label (left), our method increases the score of the highest scoring positive image window (middle), and decreases scores of the highest scoring negative windows, such as the one for the car class (right).

## 4.3 Weakly supervised learning and classification

In this section we describe details of the training procedure. Similar to Chapter 3 we pre-train the convolutional feature extraction layers C1-C7 on images from the ImageNet dataset and keep their weights fixed. This pre-training procedure is standard and similar to Krizhevsky et al. [2012]. Next, the goal is to train the adaptation layers Ca and Cb using the Pascal VOC or MS COCO images in a weakly supervised manner, i.e. from image-level labels indicating the presence/absence of the object in the image, but not telling the actual position and scale of the object. This is achieved by stochastic gradient descent training using the network architecture and cost function described in Section 4.2, which explicitly searches for the best candidate position of the object in the image using the global max-pooling operation. We also search over object scales (similar to Papandreou et al. [2015]) by training from images of different sizes. The training procedure is illustrated in Figure 4-2. Details and further discussion are given next.

### 4.3.1 Stochastic gradient descent with global max-pooling

The global max-pooling operation ensures that the training error backpropagates only to the network weights corresponding to the highest-scoring window in the image. In other words, the max-pooling operation hypothesizes the location of the object in the image at the position with the maximum score, as illustrated in Figure 4-4. If the image-level label is positive (i.e. the image contains the object) the back-propagated

error will adapt the network weights so that the score of this particular window (and hence other similar-looking windows in the dataset) is increased. On the other hand, if the image-level label is negative (i.e. the image does not contain the object) the back-propagated error adapts the network weights so that the score of the highest-scoring window (and hence other similar-looking windows in the dataset) is decreased.

For negative images, the max-pooling operation acts in a similar manner to hard-negative mining known to work well in training sliding window object detectors (Felzenszwalb et al. [2010]).

Note that there is no guarantee the location of the score maxima corresponds to the true location of the object in the image. However, the intuition is that the erroneous weight updates from the incorrectly localized objects will only have limited effect as in general they should not be consistent over the dataset.

### 4.3.2 Multi-scale sliding-window training

The above procedure assumes that the object scale (the size in pixels) is known and the input image is rescaled so that the object occupies an area that corresponds to the receptive field of the fully connected network layers (i.e. 224 pixels).

In general, however, the actual object size in the image is unknown. In fact, a single image can contain several different objects of different sizes. One possible solution would be to run multiple parallel networks for different image scales that share parameters and max-pool their outputs. We opt for a different less memory demanding solution.

Instead, we train from images rescaled to multiple different sizes. The intuition is that if the object appears at the correct scale, the max-pooling operation correctly localizes the object in the image and correctly updates the network weights. When the object appears at the wrong scale the location of the maximum score may be incorrect. As discussed above, the network weight updates from incorrectly localized objects may only have limited negative effect on the results in practice.

In detail, all training images are first rescaled to have the largest side of size 500 pixels and zero-padded to $500 \times 500$ pixels. Each training mini-batch of 16 images is

then resized by a scale factor $s$ uniformly sampled between 0.7 and 1.4. This allows the network to see objects in the image at various scales. In addition, this type of multi-scale training also induces some scale-invariance in the network.

### 4.3.3 Classification

At test time we apply the same sliding window procedure at multiple finely sampled scales. In detail, the test image is first normalized to have its largest dimension equal to 500 pixels, padded by zeros to $500 \times 500$ pixels and then rescaled by a factor $s \in \{0.5, 0.7, 1, 1.4, 2.0, 2.8\}$. Scanning the image at large scales allows the network to find even very small objects. For each scale, the per-class scores are computed for all window positions and then max-pooled across the image.

These raw per-class scores (before applying the soft-max function (4.2)) are then aggregated across all scales by averaging them into a single vector of per-class scores.

The testing architecture is illustrated in Figure 4-3. We found that searching over only six different scales at test time was sufficient to achieve good classification performance. Adding wider or finer search over scale did not bring additional benefits in our experiments.

## 4.4 Implementation details

Our training architecture (Figure 4-2) relies on max-pooling the outputs of the convolutional network operating on a small batch of potentially large images. Several implementation details make this possible.

- In order to accomodate images of various sizes, all network layers are implemented as convolutions. Layers that were described as fully connected layers in Krizhevsky et al. [2012] are also viewed as convolutions (see Figure 4-2.)

- The GPU convolution code decomposes each convolution into an intricate sequence of cuBLAS[2] calls on adequately padded copies of the input image and

[2] http://docs.nvidia.com/cuda/cublas.

kernel weights. Unlike previous "unfolded" convolution approaches (Chellapilla et al. [2006]), our scheme does not make multiple copies of the same input pixels and therefore consumes an amount of GPU memory comparable to that of the image itself. This implementation runs at least as fast as that of Krizhevsky et al. [2012] without relying on large mini-batches and without consuming extra memory. This allows for larger images and larger networks. CuDNN, written by nVidia since, offers a faster and more memory efficient implementation of the convolution operation, rendering this code obsolete today.

- The training code performs fast bilinear image scaling using the GPU texture units[3]. This is used for resizing the input images with random scales at training time.

- All the adaptation layers use dropout (Krizhevsky et al. [2012]). However, instead of zeroing the output of single neurons, we zero whole feature maps with probability 50% in order to decorrelate the gradients across different maps and prevent the coadaptation of the learned features. This variant of Dropout is now known as *SpatialDropout* and commonly available in CNN frameworks.

Our implementation takes the form of additional packages for the Torch7 environment.[4]

## 4.5 Classification experiments

In this section we describe our classification experiments where we wish to predict whether the object is present / absent in the image. Predicting the location of the object is evaluated in section 4.6.

[3]Code package available at: https://github.com/qassemoquab/texfuncs
[4]http://torch.ch.

| **Object-level sup.** | plane | bike | bird | boat | btl | bus | car | cat | chair | cow | table | dog | horse | moto | pers | plant | sheep | sofa | train | tv | **mAP** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A.NUS-SCM | 97.3 | 84.2 | 80.8 | 85.3 | 60.8 | 89.9 | 86.8 | 89.3 | **75.4** | 77.8 | 75.1 | 83.0 | 87.5 | 90.1 | 95.0 | 57.8 | 79.2 | **73.4** | **94.5** | 80.7 | 82.2 |
| B.OQUAB | 94.6 | 82.9 | 88.2 | 84.1 | 60.3 | 89.0 | 84.4 | 90.7 | 72.1 | 86.8 | 69.0 | 92.1 | 93.4 | 88.6 | 96.1 | 64.3 | 86.6 | 62.3 | 91.1 | 79.8 | 82.8 |
| **Image-level sup.** | plane | bike | bird | boat | btl | bus | car | cat | chair | cow | table | dog | horse | moto | pers | plant | sheep | sofa | train | tv | **mAP** |
| C.Z&F | 96.0 | 77.1 | 88.4 | 85.5 | 55.8 | 85.8 | 78.6 | 91.2 | 65.0 | 74.4 | 67.7 | 87.8 | 86.0 | 85.1 | 90.9 | 52.2 | 83.6 | 61.1 | 91.8 | 76.1 | 79.0 |
| D.CHATFIELD | 96.8 | 82.5 | 91.5 | 88.1 | 62.1 | 88.3 | 81.9 | **94.8** | 70.3 | 80.2 | 76.2 | 92.9 | 90.3 | 89.3 | 95.2 | 57.4 | 83.6 | 66.4 | 93.5 | 81.9 | 83.2 |
| E.NUS-HCP | **97.5** | 84.3 | **93.0** | **89.4** | 62.5 | 90.2 | 84.6 | **94.8** | 69.7 | **90.2** | 74.1 | 93.4 | 93.7 | 88.8 | 93.2 | 59.7 | 90.3 | 61.8 | 94.4 | 78.0 | 84.2 |
| F.FULL IMAGES | 95.3 | 77.4 | 85.6 | 83.1 | 49.9 | 86.7 | 77.7 | 87.2 | 67.1 | 79.4 | 73.5 | 85.3 | 90.3 | 85.6 | 92.7 | 47.8 | 81.5 | 63.4 | 91.4 | 74.1 | 78.7 |
| G.WEAK SUP | 96.7 | **88.8** | 92.0 | 87.4 | **64.7** | **91.1** | **87.4** | 94.4 | 74.9 | 89.2 | **76.3** | **93.7** | **95.2** | **91.1** | **97.6** | **66.2** | **91.2** | 70.0 | **94.5** | **83.7** | **86.3** |

Table 4.1: Single method image classification results on the VOC 2012 test set. Methods A,B use object-level supervision. Methods C to G use image-level supervision only. The combination of methods A and E reaches 90.3% mAP (in Wei et al. [2014]), the highest reported result on this data at the time of these experiments. References: A.NUS-SCM: Song et al. [2011], B.OQUAB: Chapter 3, C.Z&F: Zeiler and Fergus [2014], D.CHATFIELD: Chatfield et al. [2014], E.NUS-HCP: Wei et al. [2014],

### 4.5.1 Experiments

**Experimental setup.** We apply the proposed method to the Pascal VOC 2012 object classification task and the more recent released Microsoft COCO dataset. The Pascal VOC 2012 dataset contains 5k images for training, 5k for validation and 20 object classes. The much larger COCO dataset contains 80k images for training, 40k images for validation and 80 classes. On the COCO dataset, we wish to evaluate whether our method scales-up to much bigger data with more classes.

We use Torch7 (Collobert et al. [2011a]) for our experiments. For Pascal VOC, we use a network pre-trained on 1512 classes of ImageNet following Chapter 3; for COCO, we use the Overfeat (Sermanet et al. [2014]) network. Training the adaptation layers was performed with stochastic gradient descent (learning rate 0.001, momentum 0.9).

**Pascal VOC 2012 classification results.** In Table 4.1, we provide classification scores on the Pascal VOC 2012 test set, for which many baseline results are available. Evaluation is performed via the Pascal VOC evaluation server. The per-class performance is measured using average precision (the area under the precision-recall curve) and summarized across all classes using mean average precision (mAP). Our weakly supervised approach (G.Weak sup) obtains the highest overall mAP among all single network methods outperforming other CNN-based methods trained from image-level supervision (C-G) as well as the comparable setup of Chapter 3 (B) that uses object-level supervision.

### 4.5.2 Discussion

**Benefits of sliding-window training.** Here we compare the proposed weakly supervised method (G. Weak sup) with training from full images (F. Full images), where no search for object location during training/testing is performed and images are presented to the network at a single scale. Otherwise the network architectures are identical. Results for Pascal VOC test data are shown in Table 4.1). The results clearly demonstrate the benefits of sliding window multi-scale training attempting

to localize the objects in the training data. The largest improvements are obtained for small objects, such as bottles and potted plants, where AP increases by 15-20%. Similar results on the COCO dataset are shown in the first row of Figure 4-5, where sliding window weakly supervised training (blue) consistently improves over the full image training (red) for all classes.

**Benefits of multi-scale training and testing.** On the COCO dataset, multi-scale training improves the classification mAP by about 1% when compared to training at a single-scale $s = 1$. The intuition is that the network gets to see objects at different scales, increasing the overall number of examples. Scanning at multiple scales at test time provides an additional 3% increase in classification mAP.

**Does adding object-level supervision help classification?** Here we investigate whether adding object-level supervision to our weakly supervised setup improves classification performance. In order to test this, we remove the global max-pooling layer in our model and introduce a "masked pooling" layer that indicates the location of individual objects during training. In detail, the masked pooling layer uses ground truth maps of the same size as the output of the network, signaling the presence or absence of an object class to perform the global max-pooling, but now restricted to the relevant area of the output. This provides learning guidance to the network as the max-scoring object hypothesis has to lie within the ground truth object location in the image. We have also explored a variant of this method, that minimized the object score outside of the masked area to avoid learning from the context of the object, but obtained consistently worse results. Classification results for the masked-pooling method (I. MASKED POOL) on both the Pascal VOC and COCO datasets are provided in Table 4.2 and show that adding this form of object-level supervision does not bring significant benefits over the weakly-supervised learning.

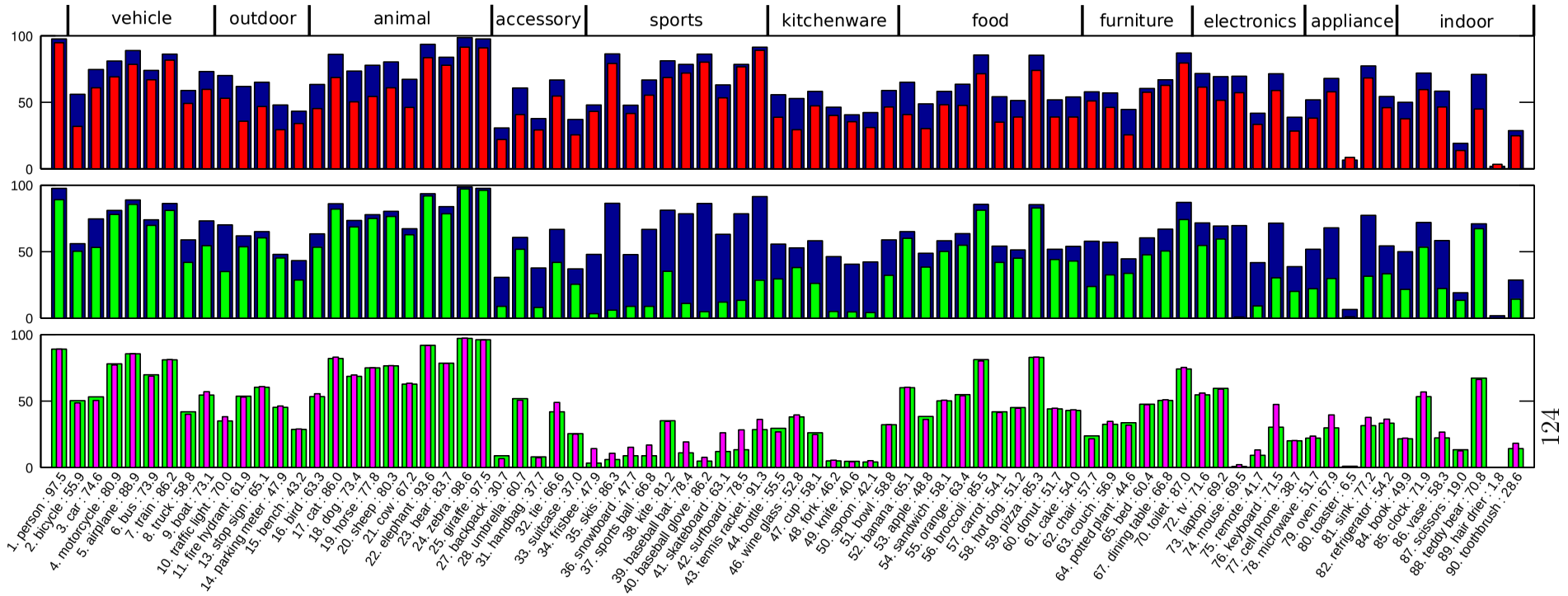





Figure 4-5: Per-class barplots of the output scores on the Microsoft COCO validation set. From top to bottom : (a) weakly-supervised classification AP (blue) vs. full-image classification AP (red). (b) weakly-supervised classification AP (blue) vs. weakly-supervised location prediction AP (green). (c) weakly-supervised location prediction AP (green) vs. masked-pooling location prediction AP (magenta). At the bottom of the figure, we provide the object names and weakly-supervised classification AP values.

| Setup | Classification | | Location Prediction | |
|---|---|---|---|---|
| Dataset | VOC | COCO | VOC | COCO |
| H.Full images | 76.0 | 51.0 | - | - |
| I.Masked pool | **82.3** | 62.1 | 72.3 | **42.9** |
| J.Weak sup | 81.8 | **62.8** | 74.5 | 41.2 |
| K.Center pred. | - | - | 50.9 | 19.1 |
| L.RCNN* | 79.2 | - | **74.8** | - |

Table 4.2: Classification and location prediction mean Average Precision on the validation sets for Pascal VOC and COCO datasets. *For R-CNN (Girshick et al. [2014]), which is an algorithm designed for *object detection*, we use only the most confident bounding box proposal per class and per image for evaluation.

## 4.6 Location prediction experiments

The proposed weakly supervised architecture outputs score maps for different objects. In the previous section we have shown that max-pooling on these maps provides excellent classification performance. However, we have also observed that these scores maps are consistent with the locations of objects in the input images. In this section we investigate whether the output score maps can be used to localize the objects.

### 4.6.1 Experiments

**Location prediction metric.** In order to provide quantitative evaluation of the localization power of our CNN architecture, we introduce a simple metric based on precision-recall using the per-class response maps. We first rescale the maps to the original image size[5]. If the maximal response across scales falls within the ground truth bounding box of an object of the same class within 18 pixels tolerance (which corresponds to the pooling ratio of the network), we label the predicted location as correct. If not, then we count the response as a false positive (it hit the background), and we also increment the false negative count (no object was found). Finally, we use the confidence values of the responses to generate precision-recall curves. Each p-r curve is summarized by Average Precision (AP). The perfect performance (AP=1) means that the network has indicated the presence / absence of the object correctly in all images and for each image containing the object the predicted object location

[5]We do simple interpolation in our experiments.

| | plane | bike | bird | boat | btl | bus | car | cat | chair | cow | table | dog | horse | moto | pers | plant | sheep | sofa | train | tv | **mAP** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I.Masked pool | 89.0 | 76.9 | **83.2** | 68.3 | 39.8 | **88.1** | 62.2 | 90.2 | 47.1 | 83.5 | 40.2 | 88.5 | **93.7** | 83.9 | 84.6 | 44.2 | 80.6 | **51.9** | **86.8** | 64.1 | 72.3 |
| J.Weak sup | 90.3 | 77.4 | 81.4 | **79.2** | 41.1 | 87.8 | 66.4 | **91.0** | 47.3 | **83.7** | 55.1 | **88.8** | 93.6 | 85.2 | 87.4 | 43.5 | **86.2** | 50.8 | **86.8** | 66.5 | 74.5 |
| K.Center pred. | 78.9 | 55.0 | 61.1 | 38.9 | 14.5 | 78.2 | 30.7 | 82.6 | 17.8 | 65.4 | 17.2 | 70.3 | 80.1 | 65.9 | 58.9 | 18.9 | 63.8 | 28.5 | 71.8 | 22.4 | 51.0 |
| L.RCNN* | **92.0** | **80.8** | 80.8 | 73.0 | **49.9** | 86.8 | **77.7** | 87.6 | **50.4** | 72.1 | **57.6** | 82.9 | 79.1 | **89.8** | **88.1** | **56.1** | 83.5 | 50.1 | 81.5 | **76.6** | **74.8** |

Table 4.3: Location prediction scores on the VOC12 validation set. Maximal responses are labeled as correct when they fall within a bounding box of the same class, and count as false negatives if the class was present but its location was not predicted. We then use the confidence values of the responses to generate precision-recall values.

fell inside one of the ground truth bounding boxes of that object (if multiple object instances were present). This metric differs from the standard object detection bounding box overlap metric as it does not take into account whether the extent of the object is predicted correctly and it only measures localization performance for one object instance per image. Note however, that even this type of location prediction is very hard for complex cluttered scenes considered in this work.

**Location prediction results.** The summary of the location prediction results for both the Pascal VOC and Microsoft COCO datasets is given in Table 4.2. The per-class results for the Pascal VOC and Microsoft COCO datasets, are shown in Table 4.3 (J.WEAK SUP) and Figure 4-5 (green bars), respectively.

**Center prediction baseline.** We compare the location prediction performance to the following baseline. We use the max-pooled image-level per-class scores of our weakly supervised setup (J.WEAK SUP), but predict the center of the image as the location of the object. As shown in Table 4.2, using the center prediction baseline (K.CENTER PRED.) results in a >50% performance drop on COCO, and >30% drop on Pascal VOC, compared to our weakly supervised method (J.WEAK SUP) indicating the difficulty of the location prediction task on this data.

**Comparison with R-CNN baseline.** In order to provide a baseline for the location prediction task, we used the bounding box proposals and confidence values obtained with the state-of-the-art object detection R-CNN (Girshick et al. [2014]) algorithm on the Pascal VOC 2012 validation set. Note that this algorithm was not designed for classification, and its goal is to find all the objects in an image, while our algorithm looks only for a single instance of a given object class. To make the comparison as fair as possible, we process the R-CNN results to be compatible with our metric, keeping for each class and image only the best-scoring bounding box proposal and using the center of the bounding box for evaluation. Results are summarized in Table 4.2 and the detailed per-class results are shown in Table 4.3. Interestingly, our weakly supervised method (J.WEAK SUP) achieves comparable location predic-

tion performance to the strong R-CNN baseline, which uses object bounding boxes at training time.

### 4.6.2 Discussion

**Does adding object-level supervision help location prediction?** Here we investigate whether adding the object-level supervision (with masked pooling) helps to better predict the locations of objects in the image. The results on the Pascal VOC dataset are shown in Table 4.3 and show a very similar overall performance for our weakly supervised (J.WEAK SUP) method compared to the object-level supervised (I.MASKED POOL) setup. This is interesting as it indicates that our weakly supervised method learns to predict object locations and adding object-level supervision does not significantly increase the overall location prediction performance. Results on the COCO dataset are shown in Figure 4-5 (bottom) and indicate that for some classes with poor location prediction performance in the weakly supervised setup (green) adding object-level supervision (masked pooling, magenta) helps. Examples are small sports objects such as frisbee, tennis racket, baseball bat, snowboard, sports ball, or skis. While for classification the likely presence of these objects can be inferred from the scene context, object-level supervision can help to understand better the underlying concept and predict the object location in the image. We examine the importance of the object context next.

**The importance of object context.** To better assess the importance of object context for the COCO dataset we directly compare the classification (blue) and location prediction (green) scores in Figure 4-5 (middle). In this setup a high classification score but low location prediction score means that the classification decision was taken primarily based on the object context. Fore example, the presence of a baseball field is a strong indicator for presence of a baseball bat and a baseball glove. However, as discussed above these objects are hard to localize in the image. The kitchenware (forks, knives, spoons) and electronics (laptop, keyboard, mouse) superclasses show a similar behavior. Nevertheless, a good classification result can still be informative

and can guide a more precise search for these objects in the image.

**Predicting extent of objects.** To evaluate the ability to predict the extent of objects (not just the location) we also evaluate our method using the standard area overlap ratio as used in object detection (Everingham et al. [2010]). We have implemented a simple extension of our method that aggregates CNN scores within selective search (van de Sande et al. [2011]) object proposals. This procedure obtains on the Pascal VOC 2012 validation set the mAP of 11.74, 27.47, 43.54% for area overlap thresholds 0.5, 0.3, 0.1, respectively. The relatively low performance could be attributed to (a) the focus of the network on discriminative object parts (e.g. aeroplane propeller, as in Figure 4-4) rather than the entire extent of an object and (b) no max-pooling over scales in our current training procedure. Similar behavior on discriminative parts was recently observed in scene classification by Zhou et al. [2015].

## 4.7 Conclusion of this chapter

So, is object localization with convolutional neural networks for free? We have shown that our weakly supervised CNN architecture learns to predict the location of objects in images despite being trained from cluttered scenes with only weak image-level labels.

We believe this is possible because of (i) the hierarchical convolutional structure of CNNs that appears to have a bias towards spatial localization combined with (ii) the extremely efficient end-to-end training that back-propagates loss gradients from image-level labels to candidate object locations.

While the approximate position of objects can be predicted rather reliably, this is not true (at least with the current architecture) for the extent of objects as the network tends to focus on distinctive object parts.

In particular, we show that for classification, taking a decision based on the *context* of an object (such as a *kitchen* for determining the presence of *knives, forks, spoons* without locating them properly) corresponds to the idea that statistically relevant

patterns are considered by the algorithm to provide a prediction.

We believe our results are significant as they open-up the possibility of large-scale reasoning about object relations and extents without the need for detailed object level annotations.

**Recent developments.** The work presented in this chapter is part of a body of work that attempts to run CNNs as sliding window classifiers on images larger than the base receptive field, collecting maps as output. One of the earliest popular incarnations of this idea is the work of Sermanet et al. [2014], integrating a localization and detection pipeline within a neural network for the ILSVRC-2013 competition using strong bounding-box supervision.

The sliding approach was later largely explored in the object segmentation problem, notably in the work of Pinheiro et al. [2015] and Long et al. [2015] where CNNs are used to compute segmentation maps for objects in images in end-to-end setups.

The work of Zhou et al. [2015] confirms our findings in this chapter in a scene classification setup, noticing that objects that are statistically relevant for a given scene consistently activate specific parts of the inner layers of the CNN allowing to perform interpretable object localization.

Weakly supervised object localization was further pursued notably in the work of Bilen and Vedaldi [2016] or our own work (Kantorov et al. [2016]) adressing the object detection task that requires also predicting the extent of objects.

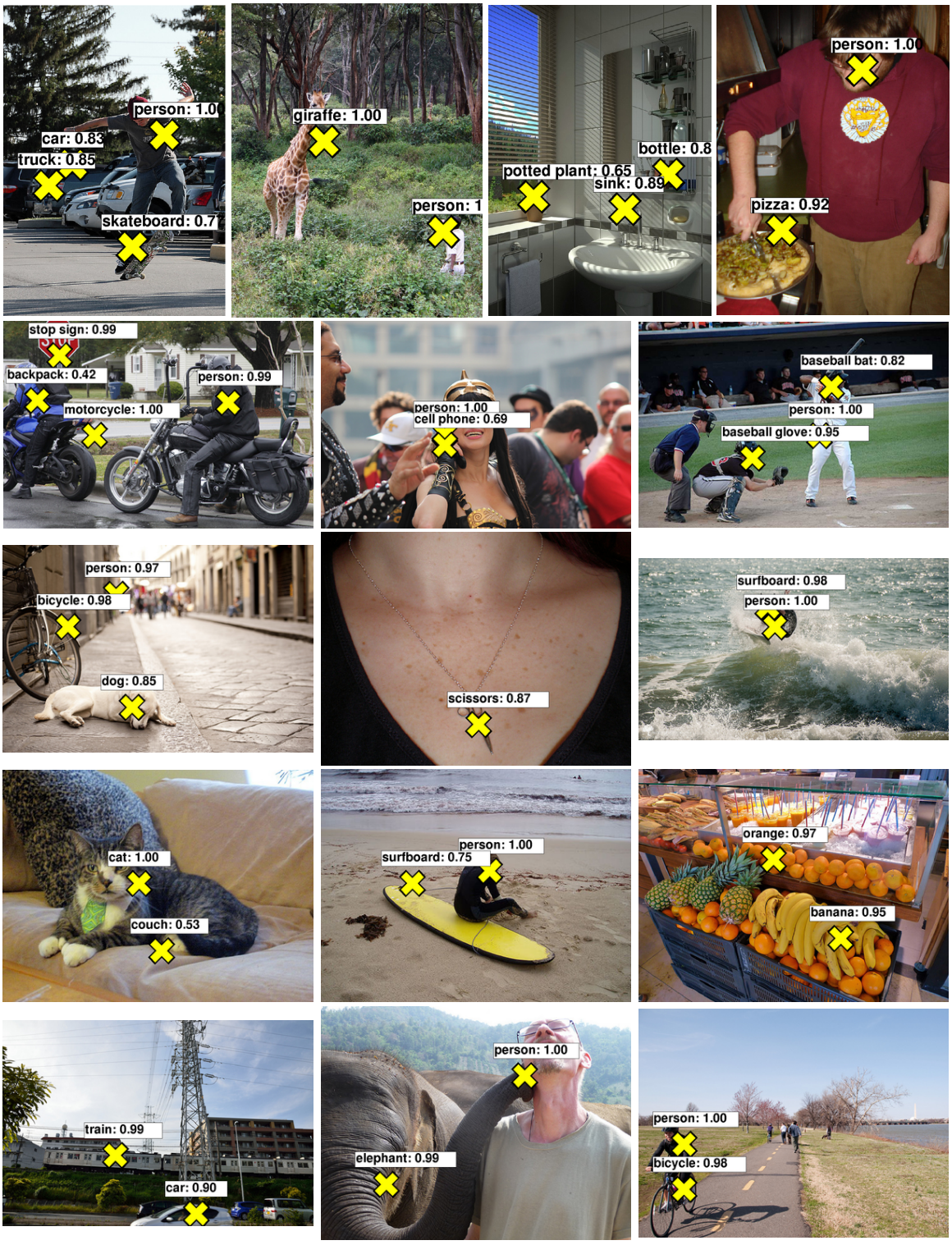



Figure 4-6: Example location predictions for images from the Microsoft COCO validation set obtained by our weakly-supervised method. Note that our method does not use object locations at training time, yet can predict locations of objects in test images (yellow crosses). The method outputs the most confident location per object per class.

# Chapter 5

# Unsupervised learning with CNNs

In the previous chapters, we presented methods for supervised learning of image representation for object classification. But as discussed in Chapter 1, if we want to build a general visual recognizer, then supervised learning may not be a good approach, as it is difficult to provide enough annotated data for training. We saw in Chapter 4 that neural networks will naturally focus on the statistical regularities that appear in the data, in particular patterns correlated with a given class, that we need to provide in quantities large enough for the algorithms to learn properly.

For supervised classification tasks, current methods typically deploy the following strategy: (i) collect as much data as possible, (ii) train a neural net. But if direct actions (such as *use+knife+onion*) are to be recognized by a pattern recognition system such as a neural network, then we have the following issue: while it is already difficult and costly to annotate thousands of object classes, the supervised approach scales even worse for combinations of objects, properties and their relations, such as "person on a chair", "girl playing tennis", or "blue van".

However, once we step away from the supervised path, it is not clear what method should be applied. Different directions are being explored, such as for example reinforcement learning (Sutton and Barto [1998]), which lets software agents interact with an environment giving it sparse rewards, and learn to take actions that maximize the reward over time; in this chapter we are interested in the path of unsupervised learning, which should let algorithms learn from large amounts of unlabeled data. Un-

supervised learning is the Holy Grail of machine learning: the idea that an algorithm could observe the world and capture its regularities on its own, without any help from humans, is a driving force for researchers eager to build Artificial Intelligence.

## 5.1 Introduction

Unsupervised learning as a way to extract knowledge from unlabeled data remains largely unsolved. At this time, the potential of unsupervised learning is not known, and it is a matter of belief whether this could help in solving hard problems such as those based on understanding the content of an image or video (as described in Chapter 1). The belief in this idea stems from the fact that the human brain is able to learn and study new things in an autonomous way, without a teacher explaining what is presented to us at every step and providing supervision. Therefore, there should be conditions where it should be possible for a machine to learn without external help, and we believe finding such conditions is an interesting long-term research goal.

One major issue in this research direction is that problems are not properly defined, as they are not related to a specific task or benchmark. Currently, unsupervised algorithms are indirectly evaluated (such as e.g. learning a feature transform from unsupervised data then evaluating its performance in an object detection pipeline as proposed by Doersch et al. [2015]), or sometimes not evaluated beyond qualitative observations. Therefore, in contrast with the previous chapters, the focus of this work on unsupervised learning is more on collecting knowledge and insights than delivering task-oriented results. This field is not mature yet and problems still need to be framed; so far the general idea is mostly to seek ways to extract knowledge from data by observing it.

In the following, we give a high-level overview of the content of this chapter, starting with (i) a review of generative adversarial networks (GANs) as a new approach for unsupervised learning, (ii) difficulties with their evaluation, (iii) links between GANs and causality, and finally (iv) understanding cost functions optimized by GANs.

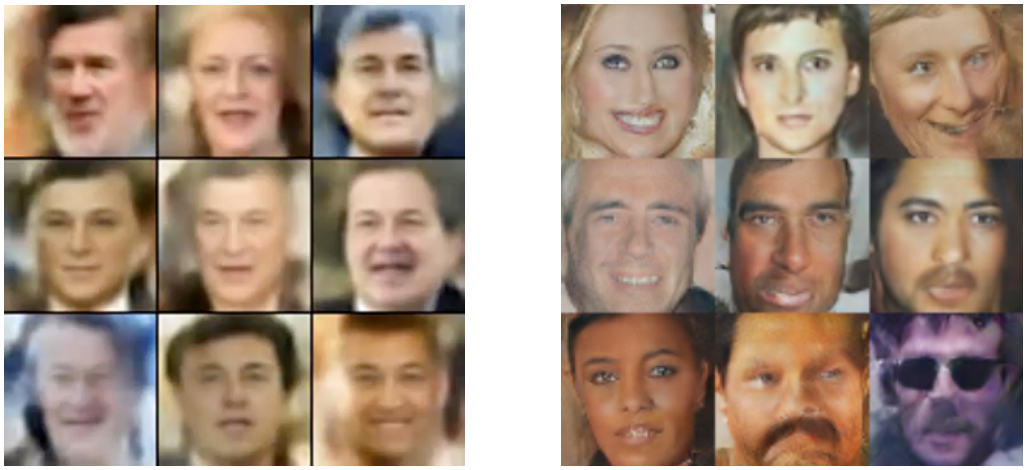


Figure 5-1: Left: Faces reconstructed using the variational autoencoder approach described in Kingma and Welling [2013]. Right: Faces generated by a Deep Convolutional Generative Adversarial Network (Radford et al. [2016]). Images credit: Alec Radford.

**Unsupervised learning with GANs.** Attempts at unsupervised learning in computer vision were made in earlier work, notably with the use of clustering algorithms such as K-means to build sparse dictionaries (Leung and Malik [2001], Sivic and Zisserman [2003]) or algorithms for topic modeling used in the text literature (Sivic et al. [2005]). In the particular case of neural networks, unsupervised learning was first addressed with autoencoder architectures, introduced by Rumelhart et al. [1986], where the goal is to first map an input to a lower-dimensional *code* (encoding phase) using an encoder network, then to reconstruct the initial input from the code (decoding phase) using a decoder network; the intuition is that frequent patterns in the data should be stored in a way similar to a dictionary in the networks, allowing a compressed representation of the input as a code[1].

Autoencoders are usually trained using an objective function (e.g. the squared difference) comparing the initial input and the reconstruction obtained by decoding. However, the training procedure can experimentally result in smooth blurry reconstructions, as shown in Figure 5-1. Recently, Generative Adversarial Networks (GANs) were proposed by Goodfellow et al. [2014], and are actively studied as of 2017. GANs are able to generate sharper, crisper images than autoencoders (see

[1] Additional references on autoencoders are provided in Chapter 2.

Figure 5-1), reviving interest in unsupervised learning using neural networks. Given the major successes achieved in the supervised scenario, hopes and expectations have been set high for this new unsupervised incarnation of neural networks.

In contrast to autoencoders, the GAN training procedure does not rely on reconstructing specific examples and therefore does not depend explicit formulation of the reconstruction loss; instead, a generator network generates random images, and a discriminator evaluates whether these generated images look similar to those of a reference dataset, using only a binary classification objective to provide gradient to the generator. This has practical applications in supervised variants of GANs, as the discriminator approach eliminates the need to specify a loss function: new tasks can be defined implicitly using only data (e.g. Isola et al. [2017], see Figure 5-4). We start this chapter by giving an overview of GANs in Section 5.2 to understand their underlying principle, and show examples of applications in vision.

**Evaluating GANs.** As a result of their training, GANs are able to generate images from a distribution $p_g$ learned by observing real reference data that follows a distribution $p_{data}$. Running a GAN generator provides new samples, but no estimate of the probability of a given sample: this excludes using maximum likelihood estimation of samples under a GAN model. How to evaluate GANs? In Section 5.3 we investigate this problem, and inspired by the role played by the discriminator we propose applying a statistical two-sample test approach to evaluate a GAN model and evaluate whether the distributions $p_g$ and $p_{data}$ match. While our method does not solve the general evaluation problem, it allows performing a first level of model selection.

**Links between GANs and causality.** Interested in what could be missing elements in order to engineer a setup capable of intelligent high-level reasoning, following the work of Lopez-Paz [2016] we are interested in links between GANs and causal reasoning. Can GANs learn causal relations from data? Harnessing causality would allow inferring the state of a system under hypothetical manipulations. Being able to answer a question such as "what would happen if... ?" seems closely related

to predicting possible future outcomes and understanding the links between different elements of a dynamic scene. This is better illustrated by the example in Bottou [2011]: rain and umbrellas are generally correlated, but causal reasoning adds the idea that banning umbrellas will not stop the rain.

While it seems possible to learn such links in an interventional scenario by *performing an action and observing consequences*, we are interested in understanding cause-effect relationships from only observational data, more readily available. To illustrate this, we take the example of a marionette, shown in Figure 5-9: the body of the marionette hangs below a plate, held by strings. Is it possible, using only observations, for a machine to understand that the position of the plate is the cause and the position of the body is the effect ?

In Section 5.4, we connect GANs to the subject of causality. First, we design a synthetic setup inspired by marionettes that we expect to contain a causal component. While we believe the problem is worthy of study, obtaining results proved difficult. Yet, looking for a way to extract causal information using GANs, we combine them with two-sample tests for the task of cause-effect discovery in a more successful approach, obtaining competitive results on the *Tüebingen cause-effect pairs dataset*.

**Cost functions of GANs.** In our last part, in Section 5.5, we are interested in the properties of the distribution $p_g$. What is learned by GANs? Building on recent contributions formalizing this issue and a connection to optimal transport, we expose the depth of this problem that appears, finally, to be much more subtle than minimizing a Jensen-Shannon divergence as suggested in the initial GAN paper by Goodfellow et al. [2014]. Although it relates to the difficult problem of defining a similarity function between images, the goal of this section is to advocate the study of such distance metrics in order to understand the behavior and the output of these algorithms more clearly.

## 5.2 Review of Generative Adversarial Networks

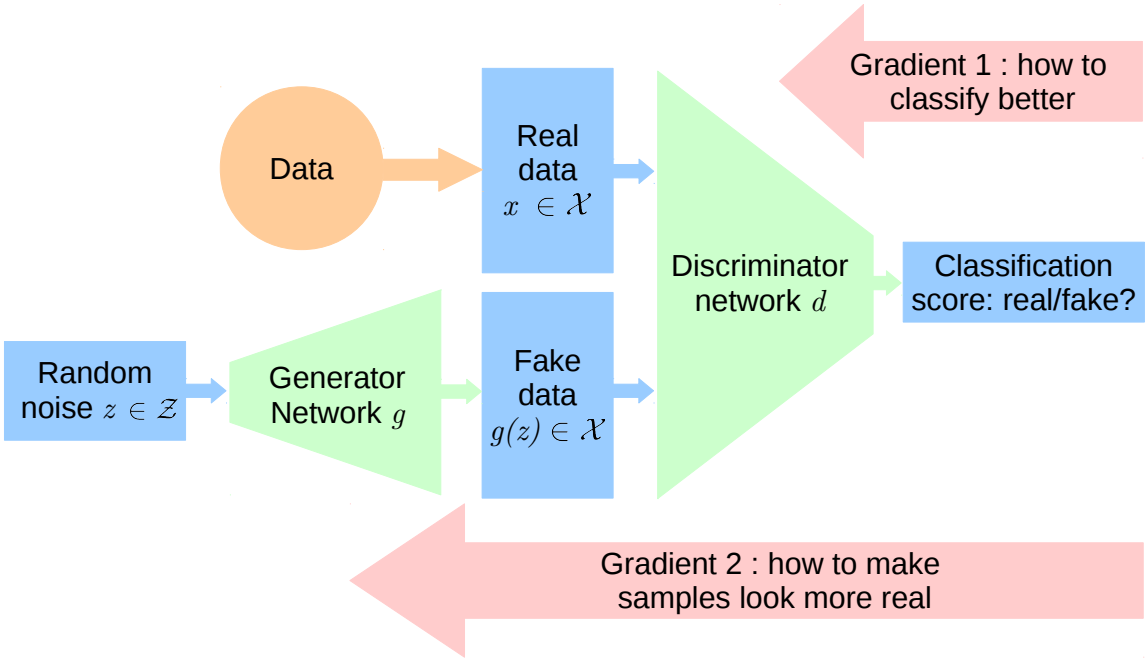



Figure 5-2: How GANs work: A generator $g$ maps a noise space $\mathcal{Z}$ to an image space $\mathcal{X}$ (usually RGB images). A discriminator $d$ is trained to classify between real samples and fake samples $g(\mathcal{Z})$. At each step, the model computes the gradient of the loss: the discriminator decreases it to classify better; the generator increases it to generate more real-looking samples, making the task of the discriminator harder.

### 5.2.1 Overview

Generative adversarial networks (Goodfellow et al. [2014]) are a recent idea in the field of neural networks, where the goal is to train a generative model allowing easy generation of new samples resembling a source dataset.

The GAN model, shown in figure 5-2 is composed of a *generator* neural network and a *discriminator* neural network working together on a reference (*real*) dataset. The generator takes random vectors as input and generates *fake* images using *deconvolution* layers that increase the resolution of data at each stage of the neural network. The discriminator (a common CNN) then tries to discriminate between fake images produced and real images by solving a binary classification problem.

Since the discriminator is differentiable, it can produce gradient indicating what small changes should be applied to the generated images for them to look more real to the discriminator. Using this gradient, the generator adapts its parameters. Training GANs helps the generator in producing increasingly realistic synthetic images which are harder to distinguish from real images by the discriminator.

Alternating between generation and discrimination appears like a *game* played by two adversaries, hence the name of the technique. GANs implement the adversarial loss function defined below:

$$\min_{g} \max_{d} \mathbb{E}_{x \sim P(\mathcal{X})} \left[\log(d(x))\right] + \mathbb{E}_{z \sim P(\mathcal{Z})} \left[\log(1 - d(g(z)))\right], \tag{5.1}$$

where $d(x)$ denotes the probability of the example $x$ following the data distribution $P(\mathcal{X})$ versus being synthesized by the generator. This is according to a trainable *discriminator* function $d$.

In the adversarial game, the generator $g$ plays to fool the discriminator $d$ by transforming noise vectors $z \sim P(\mathcal{Z})$ into real-looking examples $g(z)$. On the opposite side, the discriminator plays to distinguish between real examples $x$ and synthesized examples $g(z)$. To approximate the solution to (5.1), we alternate optimization of the two losses [Goodfellow et al., 2014] given by

$$\begin{aligned} L_d(d) &= \mathbb{E}_{x \sim P(\mathcal{X})} \left[\ell(d(x), 1)\right] + \mathbb{E}_{z \sim P(\mathcal{Z})} \left[\ell(d(g(z)), 0)\right], \\ L_g(g) &= \mathbb{E}_{x \sim P(\mathcal{X})} \left[\ell(d(x), 0)\right] + \mathbb{E}_{z \sim P(\mathcal{Z})} \left[\ell(d(g(z)), 1)\right]. \end{aligned} \tag{5.2}$$

With (5.2), the adversarial game reduces to the sequential minimization of $L_d(d)$ and $L_g(g)$ where $\ell$ is a binary classification loss function. The discriminator optimizes the loss $L_d(d)$, in order to predict a positive (1) label for real samples $x$ and negative (0) label for fake samples $g(z)$. The generator adapts accordingly by optimizing the loss $L_g(g)$. As a result, the generator trained with this procedure is able to sample from a distribution $p_g = g(\mathcal{Z})$ that is close to $p_{data}$ according to the discriminator network $d$.

The GAN approach proposed in 2014, therefore, describes a viable approach to

build a generative model that allows sampling from a distribution resembling a reference data distribution, without the need to write a loss function explicitly: the loss function between the generated samples and the real samples is implicitly defined by the discriminator. This method has been actively studied since 2014 as we show next.

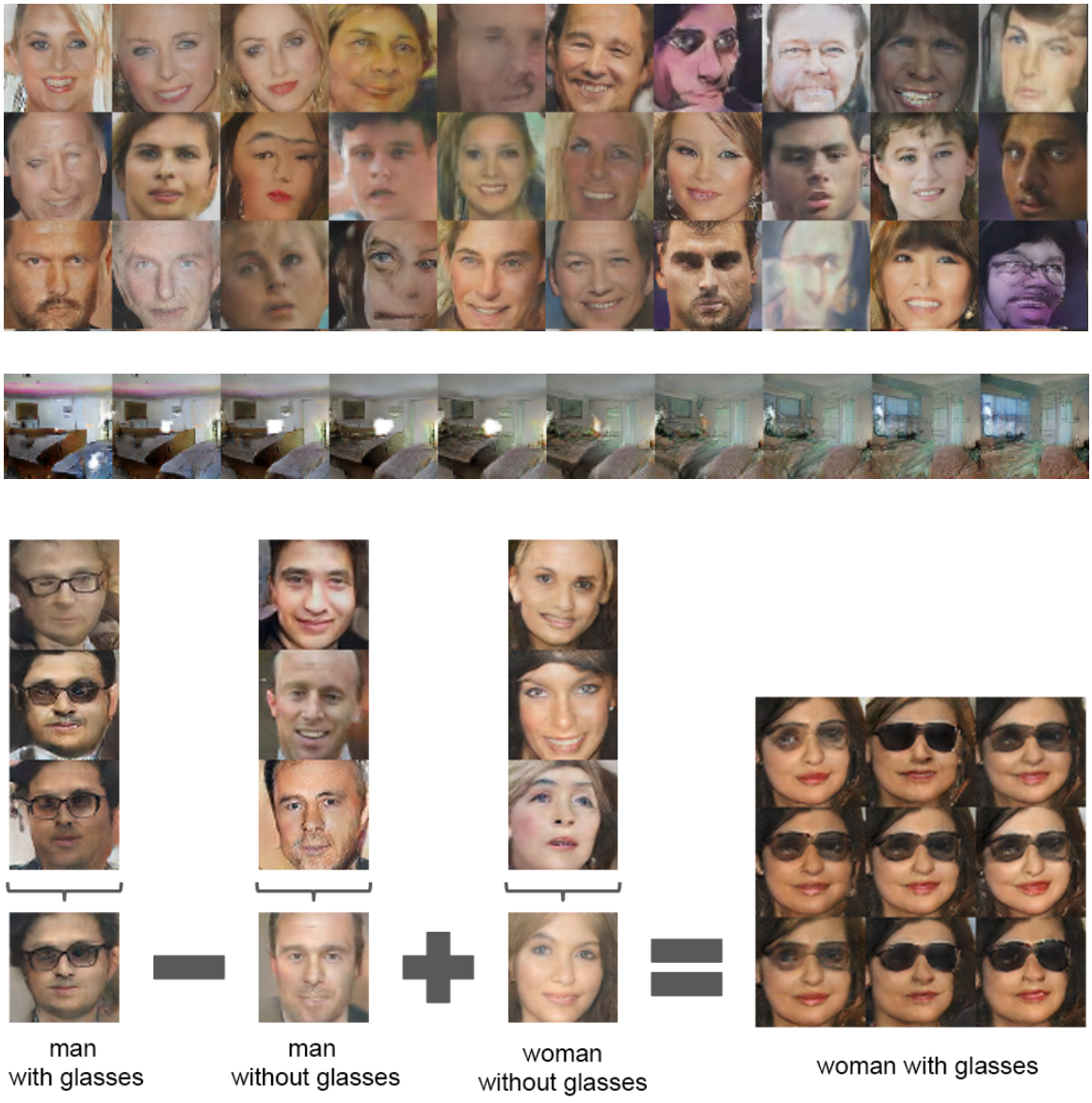



Figure 5-3: Examples of $64 \times 64$ pixel images from Radford et al. [2016]. Top: samples of new faces generated by a Deep Convolutional GAN. Middle: Result of interpolation in noise space $\mathcal{Z}$: we observe a transition from one bedroom sample to the next. Bottom: example of vector arithmetic. Authors perform arithmetic in the input space noise vectors and generate the examples on the right, hinting at a structured model of face pictures.

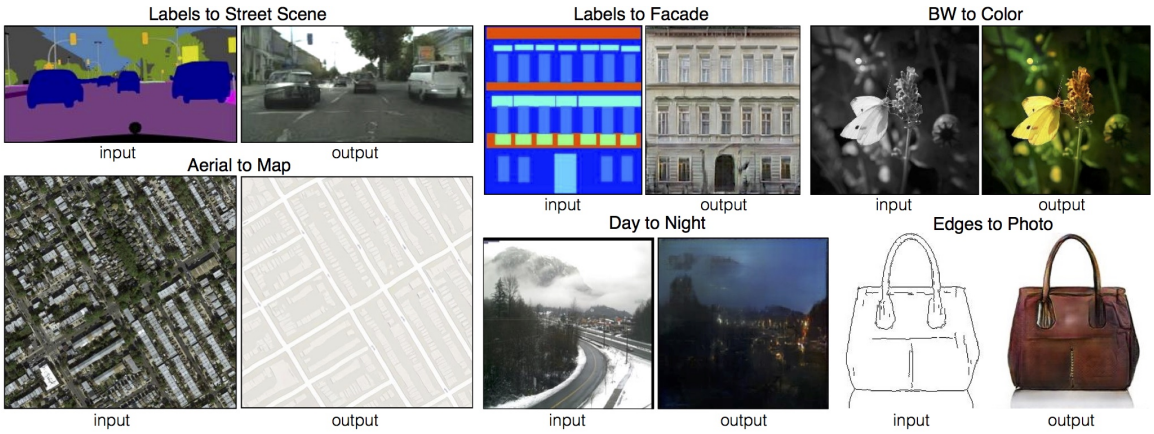



Figure 5-4: Isola et al. [2017] propose a method that maps input images to new generated images, allowing a GAN to reproduce the image processing task defined by the dataset.

### 5.2.2 Applications of GANs

As an example of the work on GANs, the LAPGANs[2] (Denton et al. [2015]) and DCGANs[3] (Radford et al. [2016]) propose models generating faces that look somewhat realistic, while not being always perfectly coherent. We show examples in Figure 5-3. In this figure, it is clear that the generated faces (top) are much sharper than those obtained with alternative approches such as Variational Autoencoders (Kingma and Welling [2013]), shown in Figure 5-1. Interestingly, the model shows evidence of a certain level of understanding of the data: it can perform arithmetic on the latent noise space and obtain meaningful outputs in terms of attributes, as shown in the bottom example where *man with glasses - man + woman = woman with glasses*.

While GANs use no meta-data or labels, Mirza and Osindero [2014] propose a variant named "*Conditional GAN*" that considers data pairs $(c, x)$ to generate new images following conditions: Conditional GANs use an input $c$ (e.g. an image) and a noise vector $z$ to generate a new image $x$ through a generator function $g : (c, z) \mapsto x$, while the discriminator $d : (c, x) \mapsto [0, 1]$ also has access to the conditioning variable. As a result, given a condition $c$ the generator produces a corresponding relevant image $x$. Many setups in computer vision (e.g. Isola et al. [2017], Zhu et al. [2016]) leverage

[2]Laplacian Pyramid of Generative Adversarial Networks
[3]Deep Convolutional Generative Adversarial Networks

this approach (see Figure 5-4) and show visually appealing generated samples. We will notably use this variant in Section 5.4 for causal discovery.

### 5.2.3 The challenge of evaluating GANs

In the context of GANs, the whole discriminator can be seen as a loss function providing gradient to the generative model; the binary classification loss of the discriminator is not very informative indeed. Not relying on explicit loss functions has an important consequence: there is no clear method for evaluation.

As a result of their training, GANs are able to generate samples following a a distribution $p_g$. The goal is for $p_g$ to be as close as possible to the data distribution $p_{data}$. But using only samples from these distributions, measuring the distance between $p_g$ and $p_{data}$ is difficult as mentioned in Theis et al. [2016]. In particular, the authors mention that there is no universal metric of how well a distribution fits, and each metric involves its tradeoffs, as illustrated in Figure 5-16 (Section 5.3). Because of this issue, it is difficult to measure improvements on this class of algorithms. However, alternatives were proposed; for example, Salimans et al. [2016] propose the *inception score*: using a classification network to measure the entropy of the distribution of the output predictions. While they report good correlation with human crowdsourced evaluation, the justification remains empirical. Alternatively, the performance of GANs can be evaluated indirectly on surrogate tasks, such as *super-resolution* which consists of increasing the resolution of an image, with corresponding performance metrics available, as proposed by Ledig et al. [2016].

In Section 5.3, we propose an evaluation procedure based on a statistical two-sample test with the following intuition: if the sample distribution produced by the generator is close to the reference data distribution, then the associated binary classification problem should be difficult. We explore this direction and obtain mixed results, revealing generation artifacts in the process.

## 5.3 Evaluating GANs

In this section, we propose to use a binary classification setup to compare whether two distributions are similar or not. The underlying intuition is: *if two distributions are close, then binary classification whether a new point is coming from one of the two distributions should be difficult.* We apply this approach to GANs that we trained using the procedure from Radford et al. [2016], in an attempt to evaluate the sample quality from our models.

Though one could use the binary classification result directly to provide a ranking of GAN models, in this section we provide a statistical interpretation of this result to connect it to two-sample tests between samples. Moreover, the statistical two-sample test point-of-view allows us to use the Maximum Mean Discrepancy two-sample test of Gretton et al. [2012] for this evaluation problem.

### 5.3.1 The evaluation problem with GANs

GANs build generator functions that are able to generate samples from random noise vectors. However, they do not allow retrieving the likelihood value $p(x)$ of a real data sample $x$, therefore usual approaches such as Maximum Likelihood Estimation can not be applied. Similarly, auto-encoder approaches can be evaluated on the quality of reconstructions for a given sample; but in the case of GANs, there is no specific target to reconstruct, excluding this approach as well.

The examples proposed in the DCGAN work of Radford et al. [2016] (see Figure 5-3) hint at three possible approaches for empirical evaluation of a generator.

The first approach is to assess the visual quality of generated samples. This is usually done by observing samples qualitatively, but there is currently no way to systematically evaluate this criterion except by crowdsourcing perceptual evaluation e.g. in Amazon Mechanical Turk, as done in Isola et al. [2017] and Zhu et al. [2016].

The second one consists of observing whether the generating function $g$ learned by the GAN corresponds to a continuous mapping. This can be done by generating samples from points along a segment $[z_1, z_2]$ in input space $\mathcal{Z}$ and observing whether

the resulting samples in image space $\mathcal{X}$ correspond to a meaningful interpolation. We illustrate this method with our *pendulums* experiment, as we describe in Section 5.4, Figure 5-14, confirming this property of the generator.

The third one is to verify whether the model allows for performing arithmetics on the problem, as illustrated in Figure 5-3 (bottom), which should be evidence for an even stronger model, where the representation in latent noise space disentangles modes of variation in the data. These two approaches, interpolation and arithmetics, are meaningful in specific cases like faces, for which we have attributes that are sometimes continuous (e.g. *smiling* or *frowning* have continuous ranges). In the general case of images where the presence or absence of an object is a discrete variable, these modes of variation are not clearly defined.

These three approaches require manual observation of the resulting images. In the following, we propose a two-sample testing approach on the real data and the generated distributions, and observe that the computed test statistic value correlates with the visual quality of images.

### 5.3.2 Two-sample tests

The goal of two-sample tests is to assess whether two samples, denoted by $S_P \sim P^n$ and $S_Q \sim Q^m$, are drawn from the same distribution [Lehmann and Romano, 2006]. More specifically, two-sample tests either accept or reject the *null hypothesis*, often denoted by $H_0$, which stands for "$P = Q$". When rejecting $H_0$, we say that the two-sample test favors the *alternative hypothesis*, often denoted by $H_1$, which stands for "$P \neq Q$". To accept or reject $H_0$, two-sample tests summarize the differences between the two *samples* (sets of identically and independently distributed *examples*):

$$S_P := \{x_1, \ldots, x_n\} \sim P^n(X) \text{ and } S_Q := \{y_1, \ldots, y_m\} \sim Q^m(Y) \tag{5.3}$$

into a statistic $\hat{t} \in \mathbb{R}$. Without loss of generality, we assume that the two-sample test returns a small statistic when the null hypothesis "$P = Q$" is true, and a large statistic otherwise. Then, for a sufficiently small statistic, the two-sample test will

accept $H_0$. Conversely, for a sufficiently large statistic, the two-sample test will reject $H_0$ in favour of $H_1$.

More formally, the statistician performs a two-sample test in four steps. First, decide a *significance level* $\alpha \in [0, 1]$, which is an input to the two-sample test. Second, compute the two-sample test statistic $\hat{t}$. Third, compute the *p-value* $\hat{p} = P(T \geq \hat{t}|H_0)$, the probability of the two-sample test returning a statistic as large as $\hat{t}$ when $H_0$ is true. Fourth, reject $H_0$ if $\hat{p} < \alpha$, and accept it otherwise.

**With classifiers.** Without loss of generality, we assume access to the two samples $S_P$ and $S_Q$ defined in (5.3), where $x_i, y_j \in \mathcal{X}$, for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$, and $m = n$. To test whether the null hypothesis $H_0 : P = Q$ is true, we proceed in five steps.

First, we construct the binary classification dataset

$$\mathcal{D} = \{(x_i, 0)\}_{i=1}^{n} \cup \{(y_i, 1)\}_{i=1}^{n} =: \{(z_i, l_i)\}_{i=1}^{2n}.$$

Second, we shuffle $\mathcal{D}$ at random, and split it into the disjoint *training* and testing subsets $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{te}}$, where $\mathcal{D} = \mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{te}}$ and $n_{\text{te}} := |\mathcal{D}_{\text{te}}|$.

Third, we train a binary classifier $f : \mathcal{X} \to [0, 1]$ on $\mathcal{D}_{\text{tr}}$; in the following, we assume that $f(z_i)$ is an estimate of the conditional probability distribution $p(l_i = 1|z_i)$.

Fourth, we return the classification accuracy on $\mathcal{D}_{\text{te}}$:

$$\hat{t} = \frac{1}{n_{\text{te}}} \sum_{(z_i, l_i) \in \mathcal{D}_{\text{te}}} \mathbb{I}\left[\mathbb{I}\left(f(z_i) > \frac{1}{2}\right) = l_i\right] \tag{5.4}$$

as our *C2ST statistic*, where $\mathbb{I}$ is the indicator function. The intuition here is that if $P = Q$, the test accuracy (5.4) should remain near chance-level. In opposition, if $P \neq Q$ and the binary classifier unveils distributional differences between the two samples, the test classification accuracy (5.4) should be *greater* than chance-level.

Fifth, to accept or reject the null hypothesis, we compute a p-value using the null distribution of the C2ST statistic $\hat{t}$ and compare it to the significance level $\alpha$. We study below the distribution of $\hat{t}$ under the null hypothesis $H_0 : P = Q$, and under

the alternative hypothesis $H_1 : P \neq Q$.

**Null Distribution.** Each term $\mathbb{I}[\mathbb{I}(f(z_i) > 1/2) = l_i]$ appearing in (5.4) is an independent Bernoulli($p_i$) random variable, where $p_i$ is the probability of classifying correctly the example $z_i$ in $\mathcal{D}_{\text{te}}$.

Under the null hypothesis $H_0 : P = Q$, the samples $S_P \sim P^n$ and $S_Q \sim Q^m$ follow the same distribution, leading to an impossible binary classification problem. In that case, $n_{\text{te}}\hat{t}$ follows a Binomial($n_{\text{te}}, p = \frac{1}{2}$) distribution. Therefore, for large $n_{\text{te}}$, we can use the central limit theorem to approximate the null distribution of (5.4) by $\mathcal{N}(\frac{1}{2}, \frac{1}{4n_{\text{te}}})$.

The cumulative distribution function (CDF) of the test statistic distribution then lets us compute the corresponding p-value to accept or reject the hypothesis from a statistical perspective by comparing it to the significance level $\alpha$. For ranking GAN models however, this thresholding step is not necessary.

### 5.3.3 Results

We obtain state-of-the-art performance for two-sample tests using our approach against alternative approaches such as the Maximum Mean Discrepancy (MMD) criterion proposed by Gretton et al. [2012], validating this approach as described in detail in Lopez-Paz and Oquab [2017]; for clarity, in this chapter we focus only on the use of two-sample tests for model selection in the context of GANs. To this end, we train a number of DCGANs on the *bedroom* class of LSUN (Yu et al. [2015]) and the Labeled Faces in the Wild (LFW) dataset (Huang et al. [2007]). We reused the Torch7 code of Radford et al. [2016] to train a set of DCGANs for $\{1, 10, 50, 100, 200\}$ epochs, where the generator and discriminator networks are 4-layer deep convolutional neural networks with variable numbers of filters; we describe the architectures in detail in Appendix C.

Then, in order to perform evaluation for each generator, we generate $10,000$ samples, and use $10,000$ held-out real data samples to build our two-sample test dataset. This dataset is then shuffled and split into a training set and a testing set for the

classification procedure.

**Artifacts.** Our first experiments reveal an interesting result. When performing two-sample tests directly on pixels using a classifier with an architecture identical to the one of discriminator, we obtain near-perfect test accuracy when distinguishing between real and synthesized (fake) examples. Such near-perfect accuracy happens consistently across DCGANs, regardless of the visual quality of their examples.

To explain this, we propose two possible hypotheses. The first is that there is overlap between the training and testing sets obtained with the generator, which would correspond to the issue of "mode collapse" reported in concurrent work (e.g. Metz et al. [2017]), where the generator outputs only a small discrete set of samples.

Our second hypothesis is that the generated samples contain a signature of the generation process such that they are easy to recognize by a classifier. This is because, albeit visually appealing, the fake examples contain checkerboard-like artifacts that may be sufficient for the tests to consistently differentiate between real and fake examples. Odena et al. [2016] observed this phenomenon concurrently with us, as they show in Figure 5-5. We show in Figure 5-6 that even when an image looks visually acceptable to the human eye, observing the output of the first layer of a CNN may still reveal heavy checkerboard artifacts.

**Results on featurized images.** In order to alleviate the possible issues mentioned above, on a second series of experiments, we featurize all images (both real and fake) using a deep convolutional ResNet (He et al. [2016]) pre-trained on ImageNet. In particular, we use the output of the last fully-connected layer of the `resnet-34` model from Gross and Wilber [2016] to obtain feature vectors of dimensionality 2048. We then perform the following two-sample tests:

- Linear-time estimate of the Maximum Mean Discrepancy (MMD, Gretton et al. [2012], Jitkrittum et al. [2016]) with Radial Basis Function kernels.
- C2ST-NN: this corresponds to the classifier two-sample test procedure described in 5.3.2, where the classifier is a two-layer fully-connected neural network with

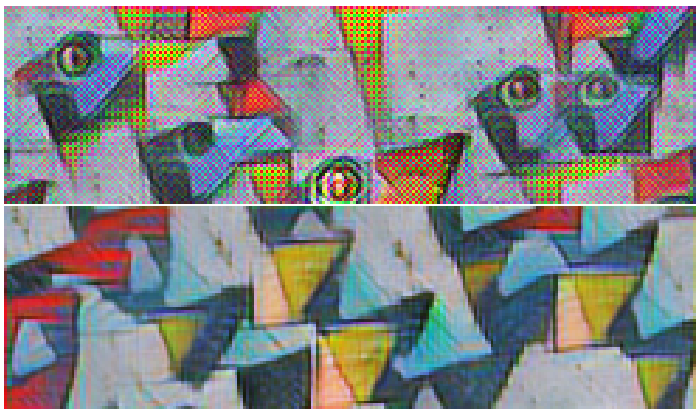


Figure 5-5: Checkerboard artifacts appear (top) as a result of an architecture based on *deconvolutions*. Using a different architecture alleviates this problem (bottom). Figure from Odena et al. [2016].

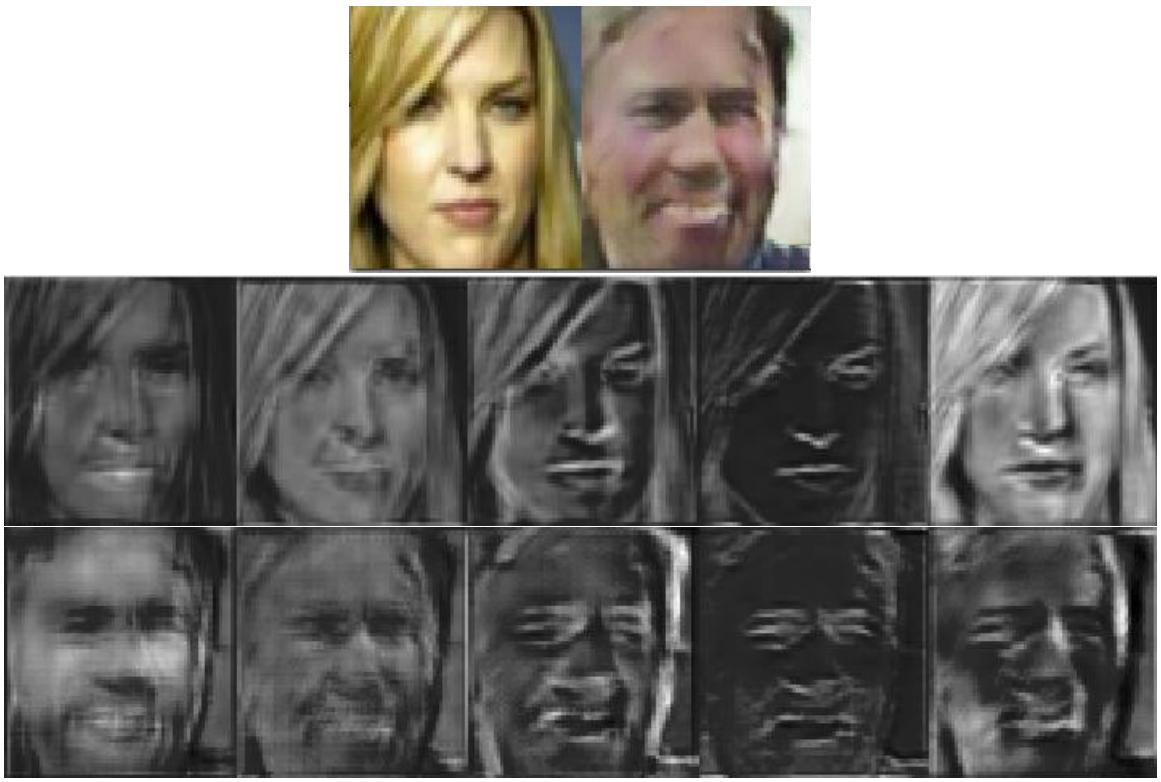


Figure 5-6: First row: Face from the training dataset (left) and face generated by a Deep Convolutional GAN (right). Second row: output maps of the first convolution layer for the real face. Third row: same output maps for the generated face. While the generated face does not show visible artifacts, some appear clearly as horizontal and vertical lines, especially on the two leftmost maps.

20 hidden units and a ReLU non-linearity. We choose a small number of units to have a classifier with reasonable training time, and use the Adam optimizer (Kingma and Ba [2015a]) for 100 epochs to reach convergence in all our experiments.

- C2ST-KNN: this classifier two-sample test is based on a K-nearest neighbor classifier with $K = \sqrt{|\mathcal{D}_{te}|}$, which is the standard "rule-of-thumb value" for these classifiers.

Reusing a CNN model pre-trained on natural images ensures that the test will distinguish between real and fake examples based only on natural image statistics, such as Gabor filters, edge detectors, and so on. Such a strategy is similar to perceptual losses (Johnson et al. [2016]) and inception scores (Salimans et al. [2016]). In short, in order to evaluate how natural the images synthesized by a DCGAN look, we employ a "natural discriminator". Figure 5-7 shows three GANs producing poor samples and three GANs producing good samples for the LSUN (Yu et al. [2015]) and Labeled Faces in the Wild (LFW) (Huang et al. [2007]) datasets, according to the MMD, C2ST-KNN, C2ST-NN tests on top of ResNet features.

Assessing the reliability of this test is difficult, as the quality of samples is prone to human judgement. However we believe this procedure can help in filtering out the worst cases, allowing a level of model selection in an automated way. In particular, "collapsed" generators, that only output a small discrete set of images, can be detected easily because of the overlap between the training and testing sets of the classifier two-sample test.

### 5.3.4 Conclusion of this section

In this section, we develop the idea of evaluating GAN models with two-sample tests, and propose an approach based on classifiers. We show that the test statistic distribution can be approximated under the null hypothesis $H_0 : P = Q$, making it a viable statistical two-sample test.

We apply this approach to sets of images generated by GANs with a classifier of

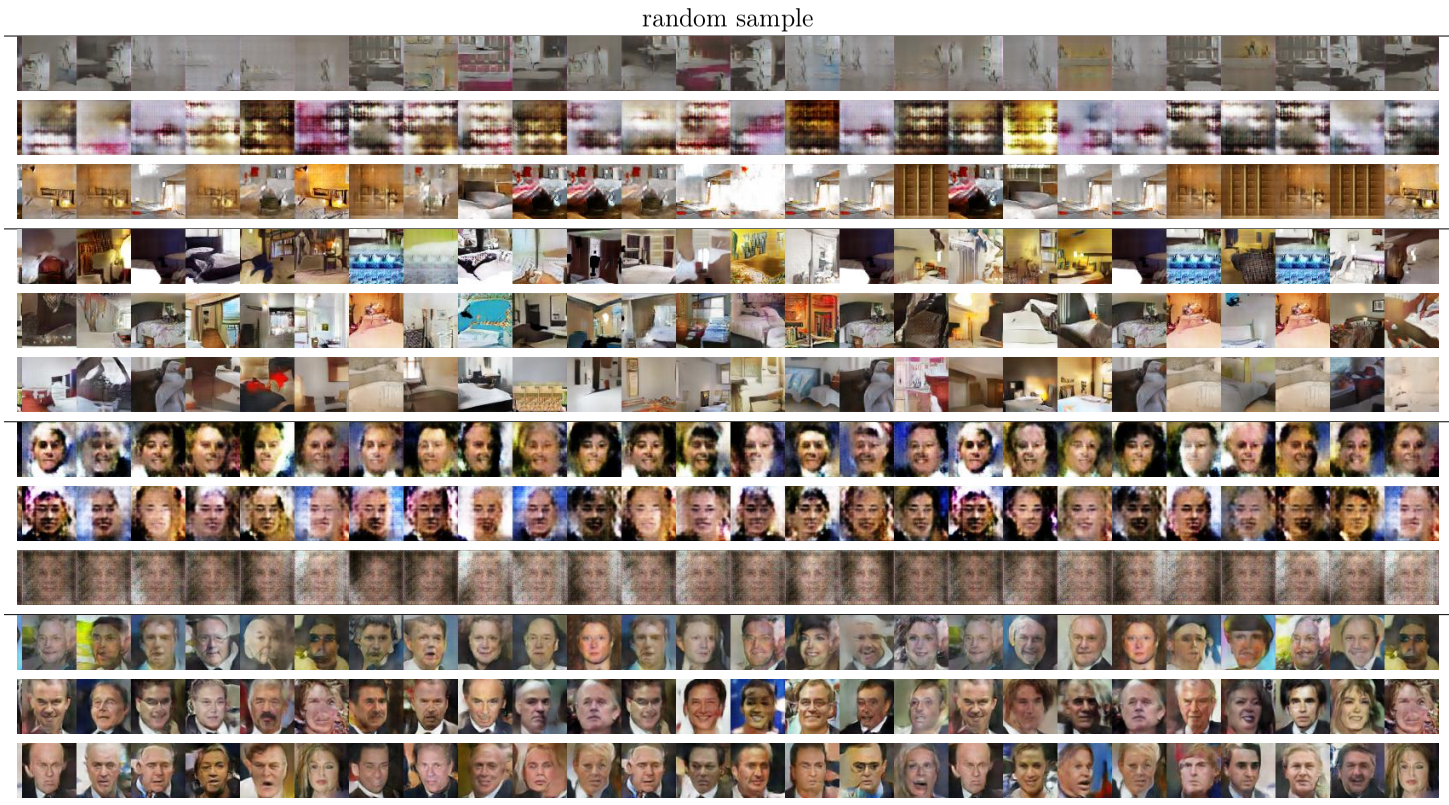



Figure 5-7: Results on GAN evaluation. The six first (resp. last) rows correspond to GAN models trained on the *LSUN bedrooms* (resp. *LFW*) dataset. We show samples from the models with the best and worst test statistic for each two-sample test that we used (MMD, C2ST-NN and C2ST-KNN). Lower test statistics are best and appear to correlate with better sample quality. We provide more detailed examples in the Appendix C. The typical values are specific to each two-sample test method; with C2ST (columns KNN and NN), the test statistic corresponds to the test set accuracy, and we note that the neural network classifier has very high performance in all cases: the task remains too easy for this classifier in this setup. The KNN classifier, on the other hand, has a wider range of accuracy values, making its results easier to use for model selection.

same architecture as the discriminator but systematically obtain perfect classification results. Further observation of images shows the presence of artifacts due to the network architectures that are involved, sometimes even invisible to the human eye. One hypothesis is that the presence of these artifacts makes it extremely easy for a classifier to discriminate between fake samples and real samples when trained directly on the pixels.

This also highlights that in GAN training, discriminators are in general powerful enough to win the adversarial game every time given enough training steps, and one key to the success of GAN training is to cripple the discriminator, either by not training it completely (which corresponds to the algorithm proposed initially) or regularizing it; one popular method, described in Arjovsky et al. [2017], consists of penalizing or constraining its Lipschitz constant to achieve this goal.

These artifacts also show that GANs are able to generate samples but with limitations due to the restriction of the function set defined by the architectures of the networks involved and/or the learning procedures; they carry (at least) a signature of the generation process.

In the case of images featurized by a transform learned on natural images, this issue is of much lesser extent, and this leads to an evaluation method with a test value that correlates with the visual quality of samples. The underlying justification is that a feature transform learned on natural images should not be sensitive to the generation artifacts observed in fake samples. This approach helps in filtering out the worst cases when training many GANs, allowing a level of model selection.

## 5.4 Causality and physics

In this section, we are interested in introducing concepts from *causality* to generative adversarial networks.

First, we illustrate this problem with the physical behaviour of a marionette. In this case, the problem has a causal structure: the position of the plate is the cause, and the position of the body below is the effect. The work of Lopez-Paz et al. [2017] advocates the presence of a causal signal in images, and we are interested in whether a GAN can capture this structure. The setup that we describe in 5.4.1, however, was not able to capture this global structure in our data. We performed an experiment on a simpler dataset generated with a similar procedure, and observed that a GAN is able to learn a good representation if the data is simpler; therefore we believe this marionette problem can be addressed in the future with better and more stable GAN algorithms.

Second, we investigate the related subproblem of causal discovery using two-sample tests in 5.4.2. Causal discovery aims at finding, given a set of pairs of data $(x_i, y_i)$ whether $x$ corresponds to the cause and $y$ to the effect, or vice versa. This subproblem is relevant to the marionette, as in this case one variable could correspond to the position of the plate, and the other variable to the position of a limb. We propose a setup based on the *conditional* variant of GANs, conditioning on one of the variables to generate the other, and build on the idea that it should be easier to generate an effect from a cause than the other way around. We evaluate this setup on the Tübingen cause-effect pairs dataset ([Mooij et al., 2016]).

**Statistical footprints of causality.** Lopez-Paz et al. [2017] build on the idea that causal signals carry a statistical footprint due to the assumption that for a causal process such that $X \to Y$ ("$X$ causes $Y$"), the cause (represented by the distribution $P(X)$) and the mechanism mapping the cause to the effect (represented by $P(Y|X)$) are independent. This is called the Independance between Cause and Mechanism (ICM) assumption and is described in Schölkopf et al. [2012].

This assumption is illustrated better in the case of an *Additive Noise Model*: we

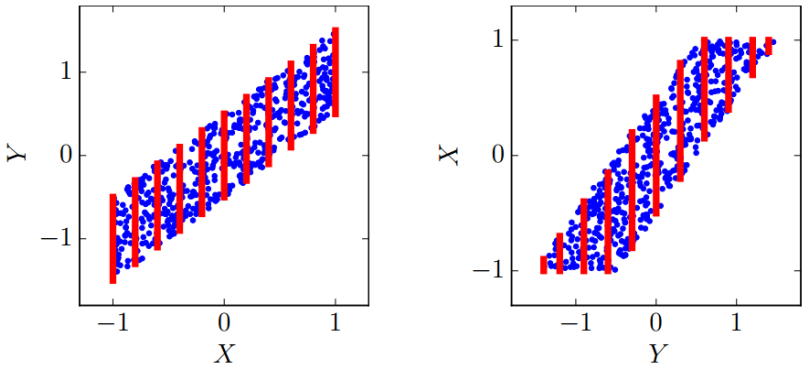



Figure 5-8: Statistical footprint of causality. We consider the case $Y \leftarrow f(X) + E$, where $X$ and $E$ are independent uniform variables, and $f$ is a linear function. We plot $(X, Y)$ on the left and $(Y, X)$ on the right and illustrate the variance of the noise with red bars. The noise has uniform variance only in the direction $X \rightarrow Y$, which is therefore the preferred causal direction. Plots obtained from: Lopez-Paz [2016].

consider the case $Y \leftarrow f(X) + E$, where $X$ and $E$ are independent uniform variables, and $f$ is a linear function. We show plots in Figure 5-8. In this case the causal direction $Y \rightarrow X$ would violate the ICM assumption: we observe that under this hypothesis the variance of the noise depends on $Y$. Therefore the causal direction $X \rightarrow Y$ is preferred.

These detectable footprints are likely to be many and in order to avoid building a corresponding catalog, Lopez-Paz et al. [2017] propose to learn these footprints from data, and predict a *neural causation coefficient* (NCC) to bags of samples $(x_i, y_i)_i$. One research direction of interest is recovering such a causal signal in observational data such as in the setup that we describe below.

### 5.4.1 Marionette physics

GANs are promising algorithms and examples hint that they are capturing structure in the data without supervision (e.g. Radford et al. [2016], shown in Figure 5-3 or Chen et al. [2016]). In this context, we made a first attempt at extracting physical structure from a synthetic dataset that we designed, inspired by *marionettes*, as shown in Figure 5-9. Concurrent work demonstrated interest in learning physics from visual observations (Mottaghi et al. [2016]) and in an unsupervised scenario with GANs

Figure 5-9: A *marionette*, with a plate controlling it. The position of the limbs depends on the position of the plate, up to noise and momentum from previous motion.

(Fragkiadaki et al. [2016]), which gave us inspiration for this setup.

The idea behind marionettes is that physical constraints apply to the position of the limbs depending on the position of the plate in a causal manner, up to some noise depending on the past positions and momentum. As such, we expect the set of appearances of a marionette and its plate to lie on a small subset of the image space, and we expect this set to be small enough for its structure to be captured by a generative model.

**Using a physics engine.** In concurrent work related to problems involving physics, the contribution of Lerer et al. [2016] proved particularly useful. The authors notably build an interface between a game engine (the open-source *Unreal Engine 4* (UE4)) and the machine learning framework Torch7 (Collobert et al. [2011a]). Game engines are useful in this context because they are optimized for rendering increasingly realistic images in real-time. Moreover, they usually include a physics simulation engine for scenes to look more natural, making them useful for synthetizing data.

We build a puppet mesh and *rig* it using the 3D design software *Blender*. A mesh defines the appearance of a 3D object. Rigging a mesh consists of associating subsets

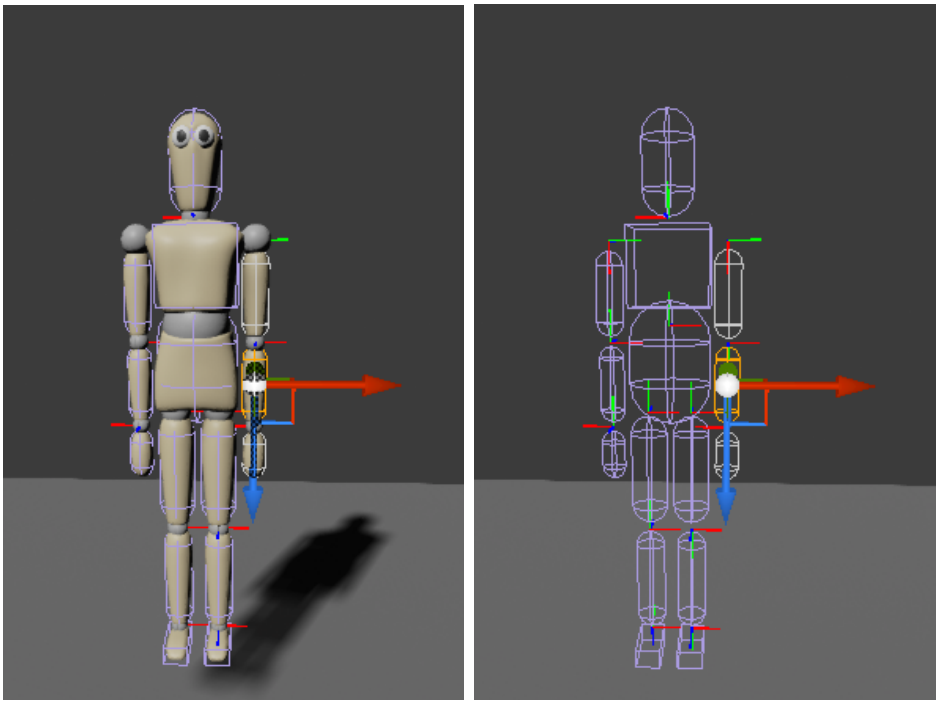


Figure 5-10: Left: our puppet model. Right: the corresponding solid *bodies* for physical simulation. Constraints appear between pairs of connected bodies, restricting the possible motions. Each bone in the armature is approximated with a primitive called *body* that has a simple shape, a mass and a volume; examples include spheres, cubes, cylinders and capsules. Bodies are connected to each other by physical *constraints* defining the range of motion that is permitted, and other physical properties such as friction.

of the mesh to bones connected within a skeleton, or *armature*. This allows animating a mesh; by simply moving the armature, the associated vertices of the mesh follow. Then, we add physical properties to our puppet in the Physics tool of UE4 as shown in Figure 5-10.

**Simulating the puppet.** We connect our puppet to a plate using strings, with the head fixed, and move the plate randomly in terms of angle and speed, collecting screenshots regularly. We obtain the dataset shown in Figure 5-11.

We expect the data to follow a model similar to an Additive Noise Model:

$$\text{body position} = f_{physics}(\text{plate}) + \text{noise}, \tag{5.5}$$

where the noise depends on past momentum and positions of the body (physical inertia). We run a DCGAN algorithm on this data using the code of Radford et al. [2016] and obtain the results described next.

Figure 5-11: Examples from the dataset we built from screenshots of our puppet simulator. The limb configuration of the puppet, attached to the plate with strings, depends on the position of the colored plate.

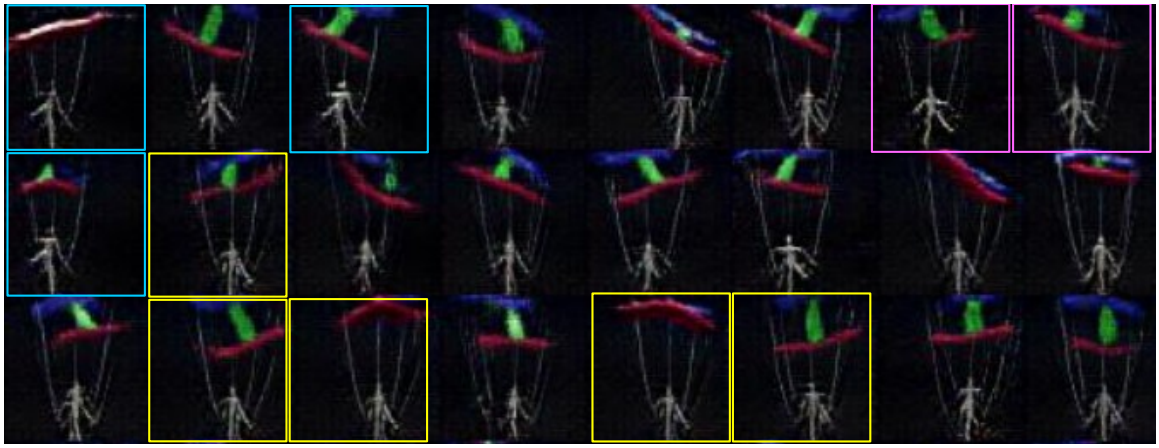


Figure 5-12: Samples generated by our DCGAN trained on our puppet dataset. In many cases, the position of the puppet does not depend on the position of the plate: we highlight with the same color samples where this phenomenon can be clearly observed.

**Results.** Capturing the underlying structure of the puppet dataset proved difficult. While the DCGAN was able to generate images that looked similar to the ones in the training data, as shown in Figure 5-12, it can be seen in a random set of generated samples that no global structure seems to be captured, as different positions of the plate correspond to the same exact position of the puppet (see highlighted examples in Figure 5-12). During the experiments, we observed that the puppet in the image was generated only within a discrete set of positions (which corresponds to what is called *mode collapse* e.g. in Metz et al. [2017]).

While the puppet experiment corresponds for now to a negative result, we believe the setup remains interesting because of the presence of a global structure in the example images. Moreover, this work may possibly be improved in the future with more stable GAN models and procedures.

**A simpler setup.** Here we do an additional experiment to show that a simpler synthetic dataset can be learned properly by a GAN. We build such a dataset using the physics simulator. In this data, two red pendulums hang below a blue bar, attached with green bars. The data is shown in Figure 5-13. A DCGAN generator trained on this simpler data learns the data in a way that allows meaningful interpolation in the latent space $\mathcal{Z}$ between samples, as we show in Figure 5-14. Therefore, a GAN is able to build a continuous generative model based on multiple observations of a given object, confirming that this work could be revisited with GAN algorithms able to deal with more complex details such as those of the puppet.

But it is yet unclear if our interest in GANs - whether we can extract the causal structure of this physical setup - is founded. We investigate this question next, and look for causal signals with GANs.

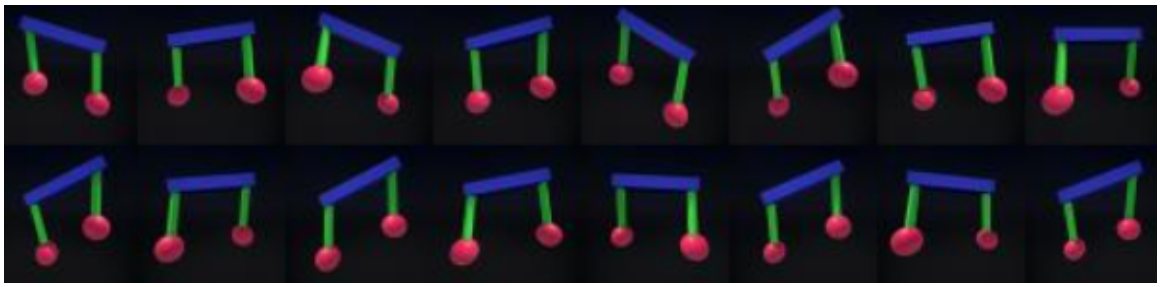


Figure 5-13: Our simpler *pendulums* dataset, built with the physics engine of UE4, consists of two red pendulums hanging below a randomly moving blue plate. The pendulums can move in all three dimensions.

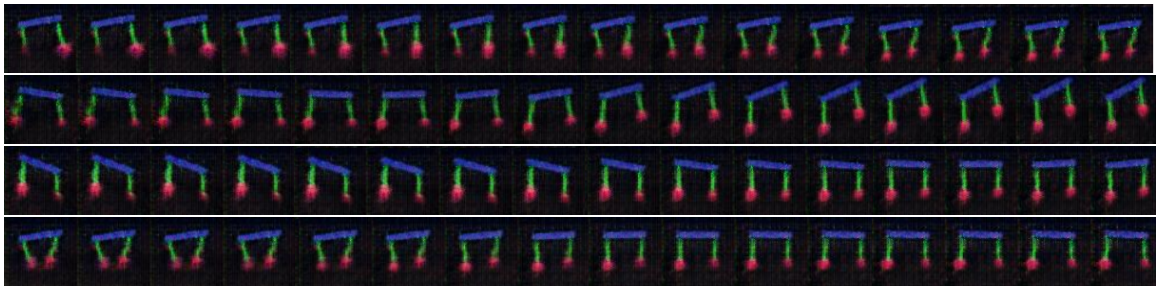


Figure 5-14: Results of interpolation in the noise space after training a DCGAN on the pendulums dataset. In each case, we interpolate between the leftmost and the rightmost samples. The model learns a continuous mapping between the latent space and the image space.

### 5.4.2 Cause-effect discovery with two-sample tests

The GAN approach to retrieve causal structure from our physics simulation has not been successful yet, but there is still hope, and we keep looking for a way to capture a causal signal from data using GANs. To this end, in this part we use GANs for the simpler setup of causal discovery, using the two-sample test procedure described in Section 5.3.

**Problem definition.** In causal discovery, we study the causal structure underlying a set of $d$ random variables $X_1, \ldots, X_d$. In particular, we assume that random variables $X_1, \ldots, X_d$ share a causal structure described by a collection of Structural Equations, or SEs (Pearl [2009]). More specifically, we assume that for all $i = 1, \ldots, d$,

the random variable $X_i$ takes values as described by the SE:

$$X_i = g_i(\text{Pa}(X_i, \mathcal{G}), N_i) \text{ for all } i = 1, \ldots, d \tag{5.6}$$

where $\mathcal{G}$ is a Directed Acyclic Graph (DAG) with vertices associated to each of the random variables $X_1, \ldots, X_d$,

$\text{Pa}(X_i, \mathcal{G})$ denotes the set of random variables which are parents of $X_i$ in the graph $\mathcal{G}$,

$N_i$ is an independent noise random variable that follows the probability distribution $P(N_i)$.

Then, we say that $X_i \to X_j$ if $X_i \in \text{Pa}(X_j)$, since a change in $X_i$ will *cause* a change in $X_j$, as described by the $j$-th SE.

The goal of causal discovery is to infer the causal graph $\mathcal{G}$ given a sample from $P(X_1, \ldots, X_d)$. For the sake of simplicity, we focus on the discovery of causal relations between two random variables, denoted by $X$ and $Y$. In particular, this setup excludes the case where $X$ and $Y$ are the effects of a common cause. Given the sample $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \sim P^n(X, Y)$, our goal is to conclude whether "$X$ causes $Y$", or "$Y$ causes $X$". We call this problem *cause-effect discovery* [Mooij et al., 2016]. In the case where $X \to Y$, we can write the cause-effect relationship as:

$$x \sim P(X), \quad n \sim P(N), \quad y \leftarrow g(x, n). \tag{5.7}$$

The current state-of-the-art in the cause-effect discovery is the family of Additive Noise Models, or ANM [Mooij et al., 2016]. These methods assume that the SE (5.7) allow the expression $y \leftarrow g(x) + n$, and exploit the independence assumption between the cause random variable $X$ and the noise random variable $N$ to analyze the distribution of nonlinear regression residuals, in both causal directions.

**Proposed method.** Assuming independent additive noise is often too simplistic (for instance, the noise could be heteroskedastic or multiplicative). Because of this

reason, we propose to use Conditional Generative Adversarial Networks, or CGANs [Mirza and Osindero, 2014] to address the problem of cause-effect discovery. Our motivation is the resemblance between the generator of a CGAN and the SE (5.7): the random variable $X$ is the conditioning variable input to the generator, the random variable $N$ is the noise variable input to the generator, and the random variable $Y$ is the variable synthesized by the generator. Furthermore, CGANs respect the independence between the cause $X$ and the noise $N$ by construction, since $n \sim P(N)$ is independent from all other variables. This way, CGANs bypass the additive noise assumption naturally, and allow arbitrary interactions $g(X, N)$ between the cause variable $X$ and the noise variable $N$.

To implement our cause-effect inference algorithm in practice, recall that training a CGAN from $X$ to $Y$ minimizes the two following objectives in alternation:

$$\begin{aligned} L_d(d) &= \mathbb{E}_{x,y}\left[\ell(d(x,y),1)\right] + \mathbb{E}_{x,z}\left[\ell(d(x,g(x,z)),0)\right], \\ L_g(g) &= \mathbb{E}_{x,y}\left[\ell(d(x,y),0)\right] + \mathbb{E}_{x,z}\left[\ell(d(x,g(x,z)),1)\right]. \end{aligned} \tag{5.8}$$

where $\ell$ is a binary classification loss function, $d$ is the discriminator function and $g(x,z)$ is a sample from the generator conditioned on $x$, using the latent vector $z$. The discriminator optimizes the loss $L_d(d)$, in order to predict a positive (1) label for real samples ($(x,y)$) and negative (0) label for fake samples $(x, g(x,z))$. The generator adapts accordingly by optimizing the loss $L_g(g)$.

Our recipe for cause-effect is to learn two CGANs: one with a generator $g_y$ from $X$ to $Y$ to synthesize the dataset $\mathcal{D}_{X\to Y} = \{(x_i, g_y(x_i, z_i))\}_{i=1}^n$, and one with a generator $g_x$ from $Y$ to $X$ to synthesize the dataset $\mathcal{D}_{X\leftarrow Y} = \{(g_x(y_i, z_i), y_i)\}_{i=1}^n$. Then, we prefer the causal direction $X \to Y$ if the two-sample test statistic (Equation (5.4), Section 5.3) between the real sample $\mathcal{D}$ and $\mathcal{D}_{X\to Y}$ is smaller than the one between $\mathcal{D}$ and $\mathcal{D}_{Y\to X}$. Thus, our method *CGAN-C2ST* is Occam's razor at play: declare the simplest direction (in terms of conditional generative modeling) as the true causal direction.

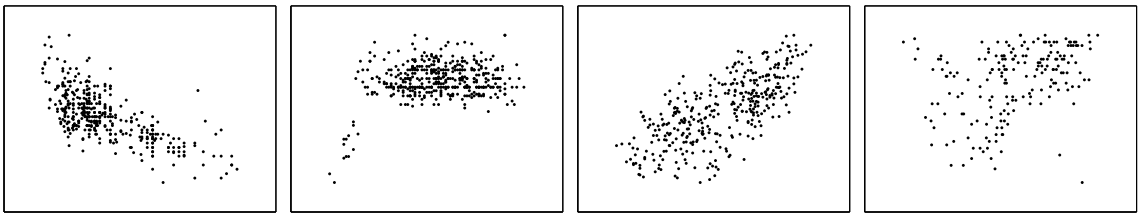


Figure 5-15: Example plots of the data in the Tüebingen cause-effect pairs dataset. From left to right: (i) x: horsepower, y: acceleration, x→y; (ii) x: age, y: height, x→y; (iii) x: ozone, y: temperature, x←y; (iv) x: life expectancy, y: latitude, x←y.

| Method | ANM-HSIC | IGCI | RCC | NCC | CGAN-C2ST | Ensemble | C2ST type |
|---|---|---|---|---|---|---|---|
| Accuracy | 67% | 71% | 76% | 79% | 73% | **82%** | KNN |
| | | | | | 70% | 73% | NN |
| | | | | | 58% | 65% | MMD |

Table 5.1: Results on cause-effect discovery on the Tübingen *cause-effect pairs* dataset. This dataset, and the ANM-HSIC and IGCI methods are provided by Mooij et al. [2016]. RCC corresponds to the work of Lopez-Paz et al. [2015] and NCC to Lopez-Paz et al. [2017].

**Results.** Table 5.1 summarizes the performance of this procedure when applied to the 99 Tübingen cause-effect pairs dataset, version August 2016 [Mooij et al., 2016]. This dataset corresponds to hand-collected real-world cause-effect samples from 99 different causal phenomena such as e.g. (altitude → temperature) or (age → salary) where the ground truth causal direction is decided by expert knowledge after studying the data-generating mechanism; this dataset of real samples is therefore small because of this tedious collection process. We show examples of this data in Figure 5-15.

The proposed method Ensemble-CGAN-C2ST trains, for each causal phenomenon, 100 CGANs in each direction as explained in the previous paragraph. Then, for each CGAN, we run a Classifier Two-Sample Test (C2ST, described in Section 5.3) to evaluate the quality of the new generated samples. Finally, we compare the C2ST statistics (equation (5.4)) of the best generator in each causal direction to predict the causal direction $X \to Y$ or $X \leftarrow Y$.

We obtain competitive results with the CGAN-C2ST approach and outperform the state-of-the-art result of Lopez-Paz et al. [2017] (NCC, 79%) by using an ensemble of 100 CGANs. The need to ensemble is a remainder of the unstable behaviour of generative adversarial training, but also highlights the promise of such models for

causal discovery.

### 5.4.3 Conclusion of this section

In this section, we were interested in the usage of Generative Adversarial Networks to recover causal signals from data. In the first part we described an experiment based on a synthetic dataset built using a physics engine, that, we expect, carries a causal component in its global structure. Learning this data using a GAN proved difficult and the generated samples did not carry the causal structure; the hope is that better GAN algorithms may be able to learn the underlying model of this data in the future.

In the second part we address the problem of cause-effect discovery; this simple case corresponds to a basic building block for causal inference, that consists of determining in a dataset of causally-related pairs $(x_i, y_i)_i$, whether X causes Y or Y causes X. We proposed the usage of the conditional variant of GANs combined with the two-sample tests described in Section 5.3, showing promising results on the Tübingen cause-effect pairs dataset.

## 5.5 Distances between distributions

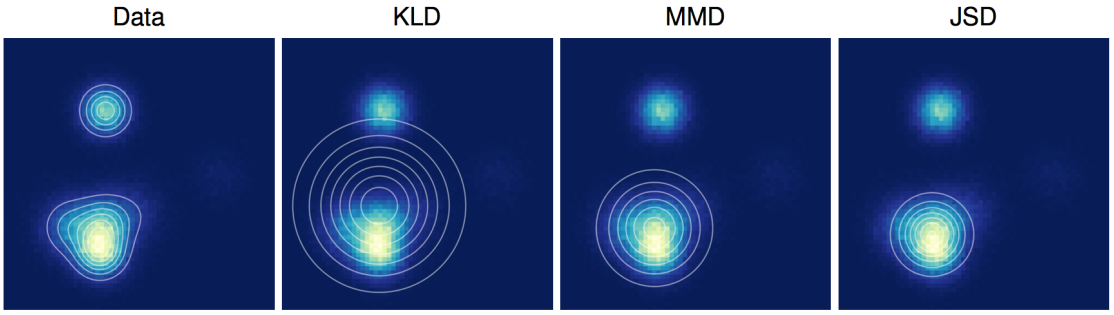



Figure 5-16: Original caption: *An isotropic Gaussian distribution was fit to data drawn from a mixture of Gaussians by either minimizing Kullback-Leibler divergence (KLD), maximum mean discrepancy (MMD), or Jensen-Shannon divergence (JSD). The different fits demonstrate different tradeoffs made by the three measures of distance between distributions.* Figure and caption from Theis et al. [2016]. MMD (Maximum Mean Discrepancy) refers to Gretton et al. [2007] and is a kernel-based method for measuring a distance between distributions (used in Section 5.3 as a two-sample test).

In this last section, our goal is to investigate different distance metrics between two distributions for GAN training. Matching two distributions, as in the GAN context, prompts for a distance to evaluate how well the generated distribution $p_g$ fits the data distribution $p_{data}$. Regarding this issue, Theis et al. [2016] report on an example, shown in Figure 5-16, that optimizing different distances leads to different solutions when trying to fit data with a model. In a related work, Nowozin et al. [2016] explore the optimization of "generative neural samplers" using different objectives. This shows that (i) it should be possible to optimize GANs in different ways and (ii) this should lead to different solutions.

Therefore we are interested in how GANs are optimized and look for insights on what is the generated distribution that is learned. In the following, we investigate three distance measures between distributions, namely the popular Kullback-Leibler divergence (KL), the Jensen-Shannon divergence (JS, which is connected to GANs) and the Earth Mover Distance (EMD, which is connected to the new Wasserstein-GAN algorithm of Arjovsky et al. [2017]).

### 5.5.1 Definitions

**Kullback-Leibler (KL) Divergence.** The KL divergence between distributions P (the data) and Q (the model) is the amount of information (from the information theory point of view) that is needed to rebuild the distribution P from the distribution Q. It is defined as :

$$KL(P||Q) = \int P(x) log\left(\frac{P(x)}{Q(x)}\right) dx. \tag{5.9}$$

If $P(x) > 0$ and $Q(x) = 0$ over a non-negligible region of the space (i.e. there is data that is not covered by the model) then $KL(P||Q) = +\infty$.

**Jensen-Shannon (JS) Divergence.** The JS divergence between P (the data) and Q (the model) is a symmetrical distance built using the KL divergence. It is defined as:

$$JS(P||Q) = JS(Q||P) = \frac{1}{2}\left(KL\left(P||\frac{P+Q}{2}\right) + KL\left(Q||\frac{P+Q}{2}\right)\right). \tag{5.10}$$

If there is no overlap at all between the distributions P and Q, then we obtain $JS(P||Q) = \log 2$, which is the maximal value of the JS divergence. In particular in the case of GANs, Goodfellow et al. [2014] show that under the assumption that the discriminator is perfectly trained at each step, the globally optimal solution for the training objective of the generator (Equation (5.1)) minimizes the JS divergence. In practice however, GAN implementations do not minimize the training objective (5.1) but instead alternately optimize of the two losses in (5.2) (see Section 5.2).

**Earth Mover Distance (Wasserstein-1).** The Earth Mover Distance is defined formally as the $p = 1$ case of the general p-th Wasserstein distance :

$$W_p(P,Q) = \inf_{\gamma \in \Gamma(P,Q)} \left(\int_{\mathcal{X}\times\mathcal{X}} d(x,y)^p \, \mathrm{d}\gamma(x,y)\right)^{1/p} \tag{5.11}$$

where $\Gamma(P, Q)$ is the set of all joint probability distributions (or *couplings*) that have P and Q as marginals, and $d(x, y)$ is a metric on the corresponding sample space $\mathcal{X}$.

The EMD is a distance between two distributions P and Q that can be intuitively seen in the following way: if P and Q correspond to piles of dirt in a space $\mathcal{X}$, and *work* corresponds to the mass of dirt $\times$ *how far* it is moved, then the EMD is the total work needed to transform P into Q, if dirt is moved optimally: it is the cost of the optimal transport plan. We build the intuition from the formal definition in (5.11) with the following scenario: let points $x$ be suppliers in a group $P$, each producing a mass $p(x)$ with a total sum of 1 and points $y$ consumers in a group $Q$, each requiring a mass $q(y)$, also summing to 1.

By definition, we have $\int_{\mathcal{X}} \gamma(x, y)\ \mathrm{d}x = q(y)$. This means that each point $y$ obtains its total mass $q(y)$ by receiving $\gamma(x, y)$ units from each point $x$. By definition, we also have $\int_{\mathcal{X}} \gamma(x, y)\ \mathrm{d}y = p(x)$. This means that each point $x$ distributes all its mass $p(x)$ by sending $\gamma(x, y)$ units to each point $y$.

This defines a transport plan $\gamma$ from $P$ to $Q$, mapping the suppliers $x$ to the consumers $y$. We now introduce the cost $d(x, y)$ of transporting a unit of mass from $x$ to $y$. Minimizing the integral in (5.11) searches the least expensive transport plan from $P$ to $Q$ according to the metric function $d$, i.e. how to move the mass optimally.

We show an example of EMD in Figure 5-17 below.

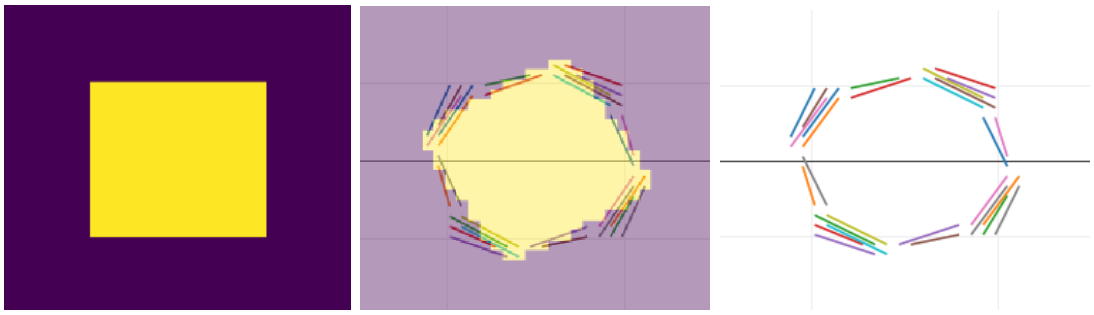


Figure 5-17: Example of optimal transport on a $32 \times 32$ grid. Left: density of the distribution P. Middle: density of Q with the optimal transport plan overlaid. Right: optimal transport plan. Each colored line corresponds to the displacement of the mass from one point to another. In this simple case, all points in the supports of P and Q have the same mass. The Earth Mover Distance corresponds to the sum of the cost of individual contributions to the transport plan.

The Wasserstein distance uses a metric function $d$ over pairs of points in $\mathcal{X}$, and generalizes it to pairs of probability distributions having support in $\mathcal{X}$. In contrast to KL and JS, it does not require overlap between the distributions but builds on the notion of ground distance instead.

In vision, Rubner et al. [2000] advocate its use for image retrieval, by computing Earth Mover Distances between distributions (histograms) of features (color, texture) extracted from images. In particular, the Wasserstein-GAN training algorithm of Arjovsky et al. [2017] builds on this distance to stabilize GAN learning.

### 5.5.2 Toy experiments

We conduct toy experiments in order to gain better understanding of the distances between distributions mentioned above. In these experiments the goal is to fit a model to a set of data points sampled from a square non-uniform distribution. Additional details are provided in Appendix D.

In Figure 5-19 we show the density of this square distribution (left); we sample points (middle) and estimate their density with a Kernel Density Estimation method (right). We then define a parameterized model as a uniform probability distribution, with the shape of a rectangle, with three parameters: height, width, and rotation angle, shown in Figure 5-18.

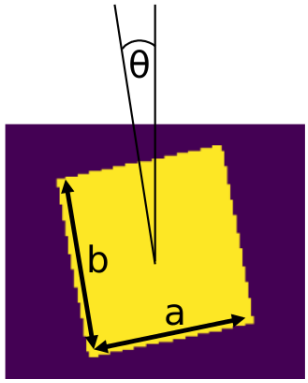



Figure 5-18: Our uniform rectangle distribution model. This model has three parameters, width (a), height (b) and rotation angle ($\theta$).

In Figure 5-20 we show the result of our first experiment: we fit the model to

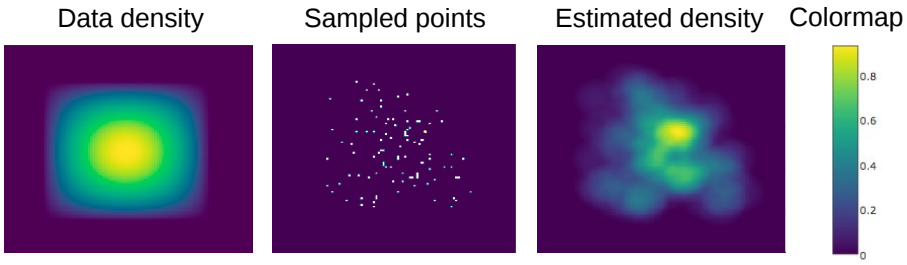



Figure 5-19: Using the data density we sample points, then estimate the density using Kernel Density Estimation (with a finite support kernel), introducing *sampling noise* in the problem.

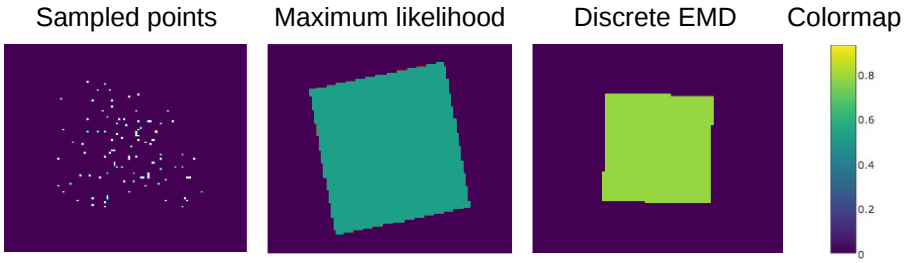



Figure 5-20: We fit the uniform rectangle to our sampled data points by optimizing the Maximum Likelihood of the points under the uniform distribution rectangle model, and the Earth Mover Distance (EMD) computed using Euclidean distance on the 2D plane. We observe that the Maximum Likelihood method attemps to cover all the points, where the EMD method is less sensitive to points with low density.

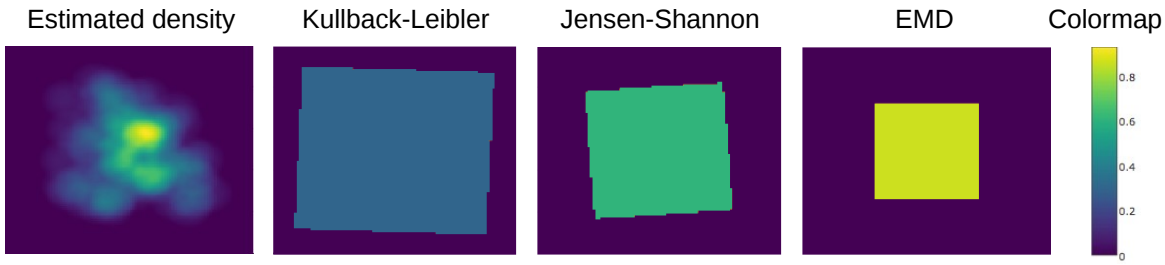



Figure 5-21: We fit the uniform rectangle to the estimated density by minimizing the KL, JS and EMD distances. EMD is computed using Euclidean distance on the 2D plane. The KL method will attempt to cover all the density, where the JS and EMD methods are able to ignore some points, fitting the dense areas with higher values.

the sampled points by optimizing the Maximum Likelihood under the model, and the Earth Mover Distance. In Figure 5-21 we show our second experiment: we fit the model to the density estimate, by optimizing the Kullback-Leibler divergence, the Jensen-Shannon divergence, and the Earth Mover Distance.

We get the following insights from running this experiment: KL and JS require overlap between densities (and therefore smoothing the data with density estimation); while optimizing KL will attempt to cover all the data, JS is able to ignore samples and make tradeoffs. This implicitly regularizes the solutions, preferring ones with better density estimates in the overlapping areas[4]. EMD, on the other hand, does not require overlap but a ground distance function; as a result, the penalty for not covering a sample is proportional to the distance, and our experiments suggest that this allows better capturing the general shape of the data, by spreading outliers around the model evenly according to the ground distance.

### 5.5.3 Discussion

**JS in original GAN.** The initial work of Goodfellow et al. [2014] uses the JS divergence to prove that the global minimum for its training procedure, when the generator distribution matches the data distribution, is $p_g = p_{data}$ under the assumptions that:

1. the discriminator is perfectly trained at each step; this poses a problem because the slope of the binary classification loss function should be zero under this assumption.

2. the underlying neural network architecture is able to represent $p_{data}$; the proposed justification is the excellent performance of neural networks in practice.

The GAN training procedure depends on gradient provided by the discriminator in order to train the generator, and the first assumption is violated in practice. As a result of this approximation, gradient is obtained, and the generator is adjusted

[4]The second term in the JS divergence (Equation 5.10) only considers the support of Q, and its value decreases when the density estimates match better.

according to this signal that we expect to depend on the architecture and state of the discriminator in unknown ways.

In Section 5.3 we observed that generated samples often contain artifacts, and this means that the second assumption could be wrong in the case of images. Moreover, while it is possible to generate samples that look good to the human eye, artifacts can still appear in the upper layers of a network, as we show in Figure 5-6; better architectures are likely to alleviate this issue, but it may be useful to keep in mind that the generation process can carry a signature from its synthetic nature.

**The neural net distance.** What is learned? Arora et al. [2017] propose to formalize this problem with the notion of $\mathcal{F}$-divergence. $\mathcal{F}$ corresponds to a restriction of the set of functions (which would correspond to a network architecture) and implicitly carries the properties of the discriminator:

Let $\mathcal{F}$ be a class of functions from $\mathbb{R}^d$ to $[0, 1]$ and $\phi$ be a concave measuring function from $[0, 1]$ to $\mathbb{R}$. Then the *$\mathcal{F}$-divergence with respect to $\phi$ between two distributions* $P$ and $Q$ supported on $\mathbb{R}^d$ is defined as

$$d_{\mathcal{F},\phi}(P, Q) = \sup_{f \in \mathcal{F}} |\mathbb{E}_{x \sim P}[\phi(f(x))] + \mathbb{E}_{x \sim Q}[\phi(1 - f(x))]| - 2\phi(1/2) \tag{5.12}$$

In Equation 5.12, $f \in \mathcal{F}$ corresponds to a classifier function discriminating between samples from $P$ and $Q$. If $\phi$ is the logarithm function, then the first (resp. second) term is the classification loss on samples from $P$ (resp. $Q$). A perfect classifier should output $f(x \sim P) = 1$ and $f(x \sim Q) = 0$. If no classifier $f \in \mathcal{F}$ can discriminate between samples from $P$ and $Q$, we have $f(x) = 1/2$, and the third term ensures that the $\mathcal{F}$-divergence becomes 0. The $\mathcal{F}$-divergence measures whether a set of functions $\mathcal{F}$ is "blind" to the differences between two distributions P and Q; it notably exposes the role of architecture, that defines the set $\mathcal{F}$.

When $\phi(t) = log(t)$ and $\mathcal{F}$ is the set of all functions from $\mathbb{R}^d$ to $[0, 1]$ then $d_{\mathcal{F},\phi}(P, Q) = 2\, JS(P||Q)$. And in this particular case, we can observe that minimizing $d_{\mathcal{F},\phi}(P, Q)$ with respect to Q corresponds to Equation 5.1, which is the initial

GAN objective of Goodfellow et al. [2014]. We derive this JS divergence below.

**Deriving the JS divergence.** We first note that for $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \mapsto a \log(y) + b \log(1 - y)$ is maximal for $y = \frac{a}{a+b}$. Then we continue with the following derivation [5]:

$$d_{\mathcal{F},\phi}(P, Q) = \sup_{f \in \mathcal{F}} [\mathbb{E}_{x \sim P}[\log(f(x))] + \mathbb{E}_{x \sim Q}[\log(1 - f(x))]] - 2 \log(1/2) \tag{5.13}$$

$$= \sup_{f \in \mathcal{F}} \left[ \int_x p(x) \log(f(x)) dx + \int_x q(x) \log(1 - f(x)) dx \right] - 2 \log(1/2) \tag{5.14}$$

$$= \int_x p(x) \log \left( \frac{p(x)}{p(x) + q(x)} \right) dx + \int_x q(x) \log \left( \frac{q(x)}{p(x) + q(x)} \right) dx - 2 \log(1/2) \tag{5.15}$$

$$= KL \left( P || \frac{P + Q}{2} \right) + \log(1/2) + KL \left( Q || \frac{P + Q}{2} \right) + \log(1/2) - 2 \log(1/2) \tag{5.16}$$

$$= 2 \, JS(P||Q) \tag{5.17}$$

We retrieve the JS divergence because $\mathcal{F}$ is the set of all functions and $f$ can therefore reach the optimal function $x \mapsto \frac{p(x)}{p(x) + q(x)}$.

However if $\mathcal{F}$ is a function set restricted by an architecture, such as in practice with GAN discriminators, then this optimal function cannot be reached in general; the properties of the corresponding $\mathcal{F}$-divergence are not clear and minimizing it, as done by the GAN procedure, should not correspond to minimizing the JS divergence in the general case. This notion of $\mathcal{F}$-divergence proposed by Arora et al. [2017] exposes the subtlety of the distance that is minimized between these distributions when training a GAN, as it depends on the architecture that restricts the set $\mathcal{F}$.

**Wasserstein-GAN and optimal transport.** Arjovsky et al. [2017] propose a different abstraction of this problem by building on a connection with optimal transport in an algorithm called Wasserstein-GAN (WGAN). This method consists of imposing a Lipschitz constraint on the discriminator, building on the Kantorovitch-Rubinstein

[5] This derivation is similar to the one proposed in Goodfellow et al. [2014].

dual formulation (Villani [2008]) of the $W_1$ distance:

$$W_1(P,Q) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim P}\left[f(x)\right] - \mathbb{E}_{x\sim Q}\left[f(x)\right]. \tag{5.18}$$

The formulation (5.18) fits in the $\mathcal{F}$-divergence formulation (5.12) as pointed out by Arora et al. [2017], it is the case where $\phi(x) = x$ and $\mathcal{F} = \{f : ||f||_L \leq 1\}$.

In this equation (5.18), $f$ corresponds to a *potential* function assigning values to the samples and varying slowly because of the 1-Lipschitz ($||f||_L \leq 1$) constraint. The intuition is that maximizing this expression with respect to $f$ consists of assigning higher values to samples of $P$ and lower values to those of $Q$; because of the upper-bounded variations (due to the Lipschitz constraint [6]), this maximal difference in potential is proportional to the distance between samples from the optimal coupling. Moreover, using expectations adds a linear dependency to the mass, prioritizing dense areas and allowing tradeoffs (being robust to outliers) during maximization. In the case where $P = Q$, we have $W_1(P,Q) = 0$. We retrieve the intuition on the Earth Mover Distance described previously, as this function grows linearly with the distance between samples of $P$ and $Q$, with a linear dependency to mass, and reaching 0 when the distributions are the same.

**Implementation in WGAN.** The WGAN algorithm implements the dual formulation (5.18) in the following loss function:

$$W_1(p_{data}, p_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim p_{data}}\left[f(x)\right] - \mathbb{E}_{z\sim \mathcal{Z}}\left[f(g(z))\right], \tag{5.19}$$

where $f$ is now called the "*critic*" (playing a role similar to the discriminator) and $g$ is the generator. The procedure then involves three operations.

First, the critic $f$ is restricted to be in a set of convolutional networks that are K-Lipschitz, by constraining all weights to a box $[-m, +m]$, with $m \in \mathbb{R}$ and approximately minimizing the equivalent objective $K.W_1$[7].

[6] See Equation (5.20) in the next paragraph.

[7] The value of $m$ determines the Lipschitz constant $K$, and this leads to minimizing $K.W_1$, or more precisely an approximation because of the restrictions on $f$.

Second, the gradient $\frac{\partial W_1}{\partial f}$ of the $W_1$ objective is a scalar with fixed unit-magnitude (it is $+1$ for *real* samples and $-1$ for *generated* samples) with respect to the potential $f(x)$ of a sample provided by the output of the discriminator. As a consequence, this procedure greatly stabilizes the training.

Third, in the adversarial training procedure, the critic $f$ is trained for 5 to 10 consecutive steps for each training step of the generator $g$, in order to approximately satisfy the supremum search in Equation 5.19.

Applying a restriction to the magnitude of the weight parameters ensures that the discriminator is Lipschitz-continuous with constant K (depending on the value of $m$). But we note that while the Kantorovitch-Rubinstein dual formulation (Equation 5.18) considers the set of all 1-Lipschitz functions, WGAN uses a set that depends on the discriminator architecture. As we saw previously with the derivation of the JS divergence from the $\mathcal{F}$-divergence, restricting the set of functions in WGAN also leads to an approximation to the optimal transport criterion being minimized.

**The point of view of optimal transport.** This approach gives valuable insights into the problem of matching two distributions by connecting it to optimal transport and distances. Distances are an important notion in the context of learning a distribution of images as we expect a meaningful semantic distance to be small for images containing the same object; for example, this is not true for the pixelwise $\mathbb{L}^2$ distance: slightly translating an object can significantly increase this distance between two images while the content remains almost identical because foreground pixels and background pixels do not have the same semantic importance.

We recall that the Lipschitz-continuity of a real-valued function $f$ in a metric space (with metric function $d$) corresponds to the following inequality:

$$\forall x, y, |f(x) - f(y)| \leq K d(x, y) \tag{5.20}$$

where K is the Lipschitz constant. Currently, the WGAN weight constraint ensures that the discriminator output varies slowly if the value of a pixel in the input is

changed: it is therefore Lipschitz-continuous wrt. the pixelwise $\mathbb{L}^2$ metric. The point of view of optimal transport hints at the following question: can we define and optimize with another metric function $d$, in order to control the learning better and extract information in an unsupervised way?

### 5.5.4 Conclusion of this section

In this section, we focus on the subject of distances between distributions, as GANs allow matching a generated distribution $p_g$ with a data distribution $p_{data}$. While the initial formulation of Goodfellow et al. [2014] proposes a link with the minimization of the JS divergence, recent insights from Arjovsky et al. [2017] and Arora et al. [2017] expose the problem of distances as a more complex issue than what was previously thought. We mention this as a possible future research direction for improving unsupervised learning techniques. While Arora et al. [2017] build a framework where the properties of the networks are implicitly carried through the notion of $\mathcal{F}$-distance, Arjovsky et al. [2017] propose a connection with optimal transport where the properties of the solution are exposed more clearly via the concept of ground distance. This is still an open problem that relates to the difficult issue of defining a similarity function between images, but we believe optimal transport and ground distances reflect an interesting abstraction to the problem of GANs.

## 5.6 Conclusion of this chapter

In this chapter, we described our work on the difficult subject of unsupervised learning, that underwent a surge in interest thanks to the recent advances in Generative Adversarial Networks (GANs).

In Section 5.2 we described the GAN algorithm proposed by Goodfellow et al. [2014] along with its shortcomings and issues; this method corresponds to a problem that is not defined as clearly as the ones that were addressed in Chapters 3-4 . Therefore, this part of our research is more exploratory, and we are seeking trails to improve our knowledge of unsupervised learning with the belief that this is important

for solving visual recognition tasks that are currently out of reach.

First, in Section 5.3, we investigated the problem of evaluating the quality of a learnt GAN. We proposed a method based on classifier two-sample tests, that let us perform a first level of model selection by eliminating worst cases. In particular, during the course of these experiments, we observed that GAN discriminators are usually good enough to discriminate reliably between real samples and generated samples once training is stopped. One of our hypotheses to explain this result is artifacts corresponding to a signature of the generation process. In future work, we could study whether this issue still holds with newer GAN algorithms; such a signature can have consequences when addressing tasks such as *domain adaptation*, as e.g. proposed by Ganin et al. [2016], if data is to be processed by a neural network further down in the pipeline.

Second, in Section 5.4 we investigated links between generative adversarial networks and the concept of causality. Inspired by Lopez-Paz et al. [2017] demonstrating the existence of a causal signal in images, we designed a synthetic experiment with a physics simulator to build a dataset that we expect carries a causal component: the plate of the marionette is the cause of the position of the body below which is the effect. While we weren't able to retrieve this causal structure using a GAN, we believe this task can be solved, and these experiments can be revisited with better training algorithms and possibly lead to better understanding of causal signals. Still, in order to illustrate the capabilities of GANs in the context of causality, we also proposed a two-sample test method to address the task of *causal discovery* on real data from the Tübingen cause-effect pairs dataset, with state-of-the-art results using ensembles, highlighting the promise of these methods for causal discovery.

Finally, in Section 5.5, we investigated different distances between distributions, that expose the following issue: we don't know what GANs are learning. Using recent insights from Arora et al. [2017] and Arjovsky et al. [2017], we get better understanding of the depth of this problem. What is learned is likely to depend on the architectures that are involved, and one interpretation (based on the idea of the *neural net distance* of Arora et al. [2017]) is that generators attempt to match the

data distribution up to the invariances carried by the discriminator and its convolutional neural network nature; however the dynamics of this learning process are still unknown. An alternative point of view described by Arjovsky et al. [2017] proposes to see GANs in an optimal transport optimization framework, and provides a stable GAN training algorithm. Further study with our toy experiment let us use an optimal transport criterion in practice, and observe that the core problem consists of building an appropriate *ground distance* to define more clearly how the distributions $p_g$ and $p_{data}$ are matched; this relates to the difficult issue of building a similarity function between images. In future work following this trail, we want to study what properties should be necessary in such metrics with the goal of building one that would allow extracting knowledge from unsupervised data.

# Chapter 6

# Discussion

In this chapter we review the contributions of this thesis, and discuss possible directions for future research.

## 6.1 Contributions of the thesis

Chapter 3 - **Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks** describes our first contribution to the field. We proposed a pre-training method that allows generalizing the performance improvements obtained by Krizhevsky et al. [2012] using the large ImageNet database, to smaller-scale tasks that are more common in computer vision, notably the popular Pascal VOC classification benchmark (Everingham et al. [2010]) where our method outperformed the state of the art at the time of publication. Our method consists of training a CNN on a large source dataset where the amount of data alleviates the overfitting issues, then use a subset of the learned network to initialize another one designed for a target task, where the amount of data is smaller.

In Chapter 4 - **Is object localization for free? Weakly-supervised learning with convolutional neural networks**, we improved the method described in Chapter 3 by decreasing the amount of supervision necessary for performing the classification task. In particular, we observed that for a classification task, a neural

network activates where relevant patterns are present, exposing the statistical learning nature of these methods; we built on this mechanism to predict the location of objects in images while only knowing at training time that they are present or absent, thus achieving a form of weak supervision in an end-to-end setup, also outperforming the state of the art at the time of publication for image classification on the Pascal VOC benchmark. This procedure relies on the fact that discriminative parts for classification are usually located on objects, although we show that the context may be sufficient for taking a decision sometimes because of the correlations in datasets: recognizing a kitchen is often enough to say that a fork and knife are present in the image.

In Chapter 5 - **Unsupervised learning with CNNs** - we followed the direction of less supervision and investigated unsupervised learning. The current successful methods in computer vision build on data and supervision to solve tasks, but providing enough annotated data to train an algorithm can be too costly and we look for alternatives. The field of unsupervised learning corresponds to a different challenge in the sense that there are no clear criteria or benchmarks for measuring progress. Therefore this research is much more exploratory, and we are seeking ways to correctly frame the problem of extracting knowledge from unlabeled data.

We studied the recent trend of Generative Adversarial Networks proposed in Goodfellow et al. [2014], that allow generating samples resembling a reference real dataset, and focused first on the issue of evaluating this procedure. We explored the usage of classifier-based statistical two-sample tests, and noticed in our experiments that such an approach, performing binary classification between real data and generated data, is likely to fail due to artifacts resulting from the constraints applied to the generator function, but that this issue can be alleviated by performing a CNN feature transform prior to the procedure, in order to rely only on "natural image" statistics. This allows us to perform a first level of model selection when training a set of GANs.

Then, building on the fact that causal links happen naturally in the real world, and work by Lopez-Paz et al. [2017] showing that these can be observed in the form of a statistical footprint in images, we proposed a synthetic experiment inspired by mar-

ionettes carrying a causal structure to illustrate this idea, and we proposed a method based on GANs and two-sample tests for performing causal discovery, ie. given a sample of data points $(x_i, y_i)$, determining whether $X$ causes $Y$ or $Y$ causes $X$ on the Tübingen cause-effect pairs dataset, with promising results. This method shows that a form of causal signal can be captured with neural network-based algorithms.

Finally, since Generative Adversarial Networks learn by minimizing a distance between generated and reference image distributionss, we focused on the subject of distances between distributions and conducted a toy experiment to understand the subtleties of different approaches such as the Kullback-Leibler and Jensen-Shannon divergences, with a focus on the Earth Mover Distance related to the new Wasserstein-GAN algorithm of Arjovsky et al. [2017]. Because of the restriction imposed by the usage of fixed neural network architectures, as pointed in the formalization of Arora et al. [2017], current algorithms minimize approximations to these distances, but the nature and the practical consequence of such approximations is not understood well.

## 6.2 Future work

**Built-in invariances in CNNs.** It is not clear what makes CNNs and their features so powerful. While we showed in Chapter 3 that using more annotated data results in higher performance, attempts were also made at pre-training networks from unsupervised data, as proposed by e.g. Doersch et al. [2015], Wang and Gupta [2015] or Bojanowski and Joulin [2017]. The high performance obtained in these works expose the role of the bottom-up architecture, and suggests that a well-initialized CNN provides already good feature descriptors for visual recognition. Therefore, it is possible that the performance is the result of (i) being able to train a network, initializing its parameters correctly, and (ii) having similar data in the source and target datasets thus making the network more sensitive to relevant common patterns. One further research direction would be to understand the balance between the importance of data and the invariances built in the architecture in designing high-performing descriptors. The *scattering networks* line of research, led by Bruna and Mallat [2013],

notably explores this direction to understand what are these invariances.

**Causality.** Unsupervised learning is a field of many open questions, and so far, the main one is *"how to do it?"*, in contrast with supervised learning, where we are interested in *"how to do it better?"*. For this reason, perspectives are wide open but still our work aims at proposing potential directions for future research. In this context, we believe that understanding the concepts of *causality* is important, as causal links express themselves naturally in nature, and harnessing these in an unsupervised manner would provide additional signal to learn from and improve our algorithms. For example, capturing the physical causal structure of a marionette using only observations would be a great step towards understanding links between objects. In this context, we showed that GANs were able to capture a causal signal in the particular setup of cause-effect discovery (Mooij et al. [2016]); one next step would be to investigate ways to map noise following a causal structure with data also following a causal structure. The goal would be to control separately at generation time, by varying the appropriate components of the generator input, parts of the resulting image that correspond to causes, and parts that correspond to effects. Achieving this would give further insights on the causal relationships between objects in scenes, and may lead to better usage of unlabeled data.

**Adversarial training.** Adversarial training has potential, as shown by their successful applications in computer vision. However we have no good understanding of the result of this training process. In Chapter 5, we investigate what GANs learn, and what kind of distance between the real data and the generated data is being optimized. The observations of Theis et al. [2016] show that there is no universal metric for measuring a distance between probability distributions, exposing the depth of this problem. With the insightful works of Arora et al. [2017], formalizing the distance problem better by introducing an architecture-dependant distance (the *neural net distance*), and Arjovsky et al. [2017] introducing a theoretical connection to optimal transport optimization, we advocate the study of distance metrics between images

and between distributions of images for a clearer understanding of the unsupervised learning problem.

# Technical background

We want to provide technical background for this thesis, that we split in two parts: Appendix A on Machine Learning, and Appendix B on Neural Networks. We will:

- show that many computer vision problems can be cast as classification problems (Appendix A).
- explain that classification corresponds to an optimization problem in machine learning, that can be addressed using first-order gradient descent methods on parameterized functions (Appendix A).
- describe neural networks as a particular restriction of the set of differentiable functions that fits in this learning framework (Appendix B).
- detail how backpropagation works in a neural network architecture and what are the necessary conditions (Appendix B).
- provide examples of important layers that are commonly used in neural networks (Appendix B).

# Notations

| symbol | meaning |
|---|---|
| $x$ | sample |
| $\mathcal{X}$ | sample space |
| $y$ | label |
| $\mathcal{Y}$ | label space |
| $\hat{y}$ | prediction |
| $\ell$ | loss function |
| $L$ | objective function |
| $\mathcal{F}$ | set of functions |
| $f(x, w)$ | parameterized function with input $x$ and parameters $w$ |
| $w^i$ | parameters for layer $i$ |
| $w = w^1, w^2, \ldots$ | set of parameters for a network |
| $x^i$ | input for layer $i$ |
| $y^i$ | output for layer $i$ |
| $A(., i)$ | column $i$ of matrix $A$ |
| $A(i, .)$ | row $i$ of matrix $A$ |
| $X$ | tensor |
| $x_{ijk}$ | element $(i, j, k)$ of tensor $X$ |
| $x \sim p(x)$ | $x$ is a random variable following the distribution $p(x)$ |

Table 1: Notations.

# Appendix A

# Technical background - Supervised Classification for visual recognition

## Outline

In this first part, we want to explain the classification problem, and an approach to solve it. The goal is to introduce in the context of machine learning:

- data distributions, that we want to understand,
- classifiers, that are algorithms trained to make sense of data,
- objective functions, that allow a learning setup to use data from the real world,
- parameterized classifier functions, of which neural networks are a special case,
- stochastic gradient descent, which is the algorithm used to train neural networks.

In Section A.1, we will first describe the data that we use for classification, and show that in supervised learning, the knowledge present in the data is an illustration of the procedure that was used to collect it. In particular, the data may contain biases, and requires human agreement on the labeling.

In Section A.2, we will describe the problem of classification, and show in particular that many tasks in computer vision can be formulated as a classification task.

In Section A.3, we get interested in the mathematical interpretation of classification, as an optimization problem where the goal is to minimize what is called the *empirical risk*, that describes the performance of a classifier function for a task and dataset.

In Section A.4, we will introduce continuous loss functions that approximate the empirical risk in a differentiable way, allowing the use of gradient descent methods. Gradient descent methods allow using parameterized differentiable functions as classifiers; neural networks fit in this class as we will see in Appendix B.

In Section A.5 we will mention convergence theorems for gradient descent algorithms, and also introduce *stochastic gradient descent*, which is the standard algorithm for training neural networks.

## A.1 Data

We would like to teach computers how to do reasoning. Many approaches are also being explored, such as *unsupervised learning* that we investigate in Chapter 5, or *reinforcement learning*, which consists of training an algorithm to perform the best actions in an environment to maximize a *reward*.

The approach that we describe in this Appendix, *supervised learning*, consists of providing examples along with our understanding of the examples, and training an algorithm to mimic human reasoning.

### A.1.1 Supervised learning with datasets

Let $\mathcal{X}$ be a set containing observations, where $x_i$ live. We call this the *input space*. Let $\mathcal{Y}$ be the set of possible labels for a sample $x_i \in X$. We call this the *label space*. Providing labels for inputs is called *supervision*. Each $y_i \in Y$ is a *ground truth label*. In the dataset paradigm, a machine learns by observing examples that are provided by the user, following a distribution $p(x, y)$. Sampling these observation

pairs $(x, y) \sim p(x, y)$ is called building a dataset. The goal is to build a function $f \in \mathcal{F} := \mathcal{X} \to \mathcal{Y}$ that provides predictions $\hat{y} = f(x)$. We want the machine to mimic human understanding and predict $\hat{y} = y$.

A *dataset* is a finite set of examples $(x_i, y_i)_i$ sampled from $p(x, y)$. For example, we can sample examples from the set $\mathcal{X}$ of images of pets, with corresponding labels in the label space $\mathcal{Y} = \{cat, dog\}$. For simplicity, we will assume that there are no other pets than cats and dogs.

Supervised learning consists of learning the conditional distribution $p(y|x)$ using the dataset: in this example, this means that given an image $x \in \mathcal{X}$, the machine should be able to output whether this image is of a cat or of a dog, on new samples from the input space $\mathcal{X}$ (images of pets that were not seen before). In order to do this, we expect a learning machine to learn the **redundant patterns in the data**, and associate them with the relevant label to provide a *prediction* on new samples. There is a catch though: the dataset distribution $p(x, y)$ carries a lot of hidden information (and possible patterns) related to the way it was collected, which may be ultimately absorbed by the machine during learning.

### A.1.2 Biases due to human intervention.

As mentioned above, building a dataset consists of collecting samples from the data distribution $p(x, y) = p(y|x)p(x)$. Human intervention occurs in the following steps:

- defining the input space $\mathcal{X}$: in our case, selecting only images of pets is a restrictive condition.

- Collecting the samples $x$: in our example, browsing the internet for collecting images implies that other people have put these images online in the first place, following unknown rules that are carried by the distribution $p(x)$.

- Defining the label space $\mathcal{Y}$: in our example, the {cat, dog} set implies that pets should be grouped in these two large categories, but it is also possible to use the set {{breeds of cats}, {breeds of dogs}} in this setting. The label space $\mathcal{Y}$

defines the chosen granularity of the predictions and thus the level of *invariance* that we want the machine to learn.

- Collecting the labels $y$ by asking human experts: while in our case, we expect human agreement on deciding whether an image is of a dog or of a cat, the consistency of labeling is not always verified. In the case of more subjective label spaces (e.g. {beautiful, not beautiful}), $p(y|x)$ carries information related to the human collection process.

Building general learning systems is very difficult because of these issues, and it was pointed out in Torralba and Efros [2011] that for a given recognition task, performance was not consistent across datasets when tested. This means that the behavior of a learning system depends on the dataset that was used for training. In supervised learning, a machine will only learn from given data; no more, no less. For example, if a poorly-balanced dataset contains 95% of samples corresponding to a same label, an easy solution for a learning algorithm will be to output that label and ignore the input, yielding 95% accuracy but not useful in practice.

### A.1.3 Training set, validation set, testing set

Datasets are usually split in three subsets:

- the training set, which is used for learning,
- the validation set, which is kept aside and used to check the performance on unknown examples and/or optimize hyperparameters such as the architecture of the classifier,
- the testing set, to allow final comparison with other research teams.

Good practice in machine learning requires the testing set to be completely unknown to the user building a learning setup. In the best case, this testing set is used only once for the final evaluation, for comparison to other setups by an external software: the correct labels are unknown to the user.

If the classification setup has any kind of hyper-parameters (such as "what kind of learning algorithm should we use?"), a subset of the training set is kept on the side along with the corresponding labels: we call it the validation set. This validation set will be used to tune the hyper-parameters for the best performance, hoping that it will lead to the best performance on the testing set. *Tuning* consists exactly of trying different values until finding the one that works best.

### Conclusion

We have discussed that learning based on data can be subject to various biases due to data collection and human decisions. A trained algorithm is meant to be run ultimately "in the wild", and when building a setup the goal is to reproduce these conditions as precisely as possible. This goal can only be reached up to some point, depending on the knowledge and the effort of the practitioner. We use datasets to define the *empirical risk* in Section A.3.

## A.2 Classification

We now dive deeper in learning, and study the important case of classification, which is the basic building block of visual recognition systems. Classification is a specific learning task, where the label space $\mathcal{Y}$ is discrete and finite. In this case, each element of $\mathcal{Y}$ is called a *class*, and we want to infer which class corresponds to a given sample $x \in \mathcal{X}$. For each $x$, there is one and only one corresponding label $y \in \mathcal{Y}$. We want to build a *classifier* $f \in \mathcal{F} = \mathcal{X} \to \mathcal{Y}$ such that $\hat{y} = f(x) = y$ for all $(x, y) \sim p(x, y)$.

Different classification tasks have different performance metrics. We mention three examples below.

### A.2.1 Different variants of classification

**Binary classification.** In binary classification, the goal is to build a function $f$ that will be able to answer a binary question given a data sample $x$, providing a

prediction $\hat{y} \in \mathcal{Y} = \{yes, no\}$.

$$\hat{y} = f(x) \in \{yes, no\} \text{ answers "is there a cat on this image?".}$$

We would like $f$ to provide the correct prediction on as many samples as possible. In this case, our performance metric is "how often does $f$ provide the correct prediction?". This is the most general case. In Section A.3 we will describe the *0-1 loss function* that reflects the performance of $f$ for this metric.

**Attribute / multi-label classification.** One simple extension of this would be to have many independent predictions for a given input.

$$\begin{aligned}
\hat{y}_1 = f_1(x) \in \{yes, no\} &\text{ answers "is there fur?",} \\
\hat{y}_2 = f_2(x) \in \{yes, no\} &\text{ answers "are there paws?",} \\
\hat{y}_3 = f_3(x) \in \{yes, no\} &\text{ answers "is it a cute animal?".}
\end{aligned}$$

In this case, our performance metric can be "how often do all $f_i$ provide a correct prediction". This is a more refined case than the binary, and is useful when many questions are considered simultaneously.

**Multi-class classification (forced-choice).** One more specific extension is the forced-choice classification problem. In that case, there are multiple possible answers to the question, but we add the constraint that only one of them can be correct at a time.

$$\hat{y} = f(x) \in \{cat, dog, cow, horse\}.$$

In this case, our performance metric is "how often does $f$ provide the correct prediction among the possible choices". This is useful in problems such as digit recognition for example or ImageNet classification (Russakovsky et al. [2015]), when we know in advance that the input is in a given finite set.

### A.2.2 Generality of classification in vision problems

Classification is ubiquitous in computer vision, especially whenever some form of recognition is involved: many tasks can be broken down and reduced to a combination of binary classification tasks. We review some examples below to illustrate that classification is a base building block for many computer vision setups.

**Example: Image classification.** The base task of image classification consists of predicting whether an image contains, or not, at least one object of a given class. In Figure A-1, we show the base task that consists of predicting whether an image contains a dog or not, a cat or not, a person or not: this can be reduced to a combination of three binary classification tasks. This corresponds to image classification as defined in the Pascal VOC dataset (Everingham et al. [2010]).

Figure A-1: Image classification task: the goal is to answer whether each image contains a dog, a cat or a person. This a simple combination of three binary classification tasks.

**Example: Object detection and segmentation.** Object detection is a more complicated version of the above: this task consists of finding an object of a given type, and drawing a tight rectangular box around it. We can see in Figure A-2 that the detection task can be reduced to a classification task, by first sampling possible locations, then running a trained classifier to answer whether the image contains completely and predominantly an object. This approach corresponds to the R-CNN algorithm (Girshick et al. [2014]) for object detection.

Similarly, object segmentation can be cast as a classification problem by considering all possible windows in an image and training a classifier to answer whether the

Figure A-2: Reducing the detection task to a classification problem. The goal is to find a tight bounding box around the bottle. We sample possible locations, reducing the task to binary classification predicting whether the input window contains *completely and predominantly* the object of interest.

center pixel is part of the object of interest.

**Example: Tracking.** Tracking consists of following an object in a video. Given an initial location of the object to follow (in general as a box), the task consists of finding the corresponding box in the next frames. We show an example in Figure A-3. The tracking task can be reduced to considering the neighborhood of the initial position, and predicting whether a box contains the same object as the initial one.

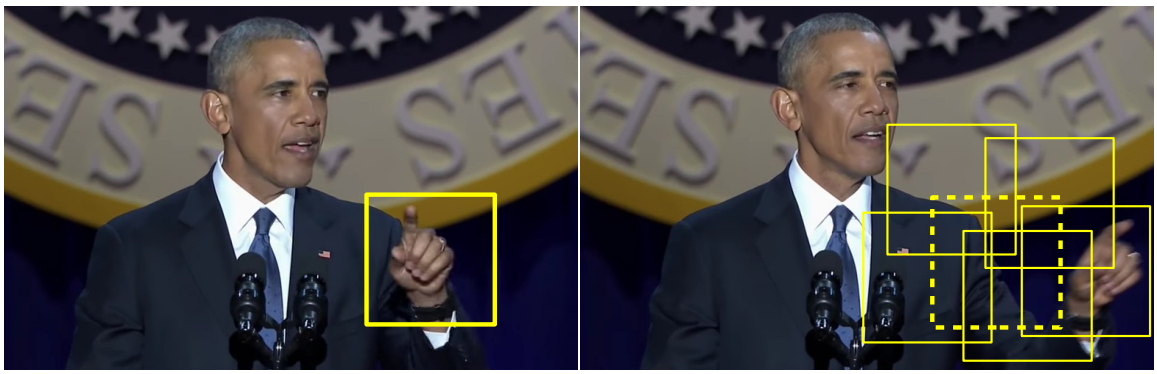


Figure A-3: Reducing the tracking task to a classification problem. The goal is to follow an object in a video. We sample windows in the neighborhood of the object of interest in the first frame, and train a classifier to predict whether one of the new windows *contains the same object.*

### Conclusion

Classification is a base building block in computer vision, that is ubiquitous when it comes to visual recognition. Problems can often be reduced to binary classification, making it a very general case.

## A.3 Optimization and learning

We described how general the classification problem is, and we are now interested in the method for solving it. The common approach is to have a dataset and optimize a classifier with respect to a continuous loss function, allowing the use of first-order (gradient) methods. In the following, we assume that we have a finite dataset $\mathcal{D}$:

$$\mathcal{D} = (x_i, y_i)_{1 \leq i \leq N}, \text{ where } (x_i, y_i) \sim p(x, y) \in \mathcal{X} \times \mathcal{Y}. \tag{A.1}$$

In this expression $(x_i, y_i)$ are sample/label pairs following a distribution $p(x, y)$.

### A.3.1 Empirical Risk Minimization.

We are interested in finding a function that will provide the correct prediction for any sample from the distribution $p(x)$. In order to evaluate this function, we will use a loss function $\ell$ to compare its predictions to the ground truth label. For instance, in binary classification, we can define the 0-1 loss function $\ell_{0-1}$ to reflect the performance metric (see Section A.2) we are interested in:

$$\ell_{0-1}(\hat{y}, y) = 0 \text{ if } \hat{y} = y, \tag{A.2}$$

$$\ell_{0-1}(\hat{y}, y) = 1 \text{ if } \hat{y} \neq y, \tag{A.3}$$

where $\hat{y}$ is a prediction ($\hat{y} = f(x)$), and $y$ is a ground truth label. A lower value of loss is better. This loss function indicates if the classifier made an error. Since we want to maximize the amount of correct predictions over the data, we want to minimize this loss over the data distribution. We define the *risk* associated to the classifier $f$

as the expectation of the loss:

$$R(f) := \mathbb{E}_{(x,y)\sim p(x,y)}[\ell(f(x), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y) dp(x, y). \tag{A.4}$$

The goal of learning is to find $f^* := \underset{f\in\mathcal{F}}{\operatorname{argmin}} R(f)$. However, in general, we don't have access to the whole distribution $p(x, y)$; in order to approximate this risk, we use a finite set of N samples from this distribution: the dataset D. We define the empirical risk associated to the classifier f using the dataset D:

$$R_{emp}(f) := \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), y_i) \tag{A.5}$$

A learning algorithm will search $f^*_{emp} := \underset{f}{\operatorname{argmin}} R_{emp}(f)$. Searching the minimizer of the function $R_{emp}$ is an optimization problem.

### A.3.2 Overfitting

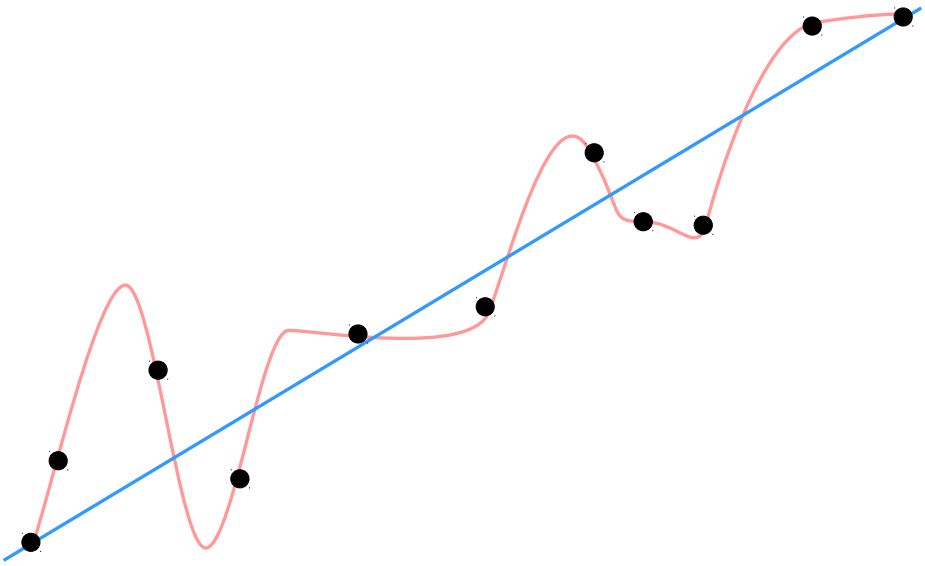


Figure A-4: Illustration of overfitting. In this case we build a function to match data points (in black) following a noisy linear distribution. The overfitted red curve matches the training points better but the less flexible blue curve seems to match the structure of the overall data better: we expect it to fit new points better.

Using an approximation of the empirical risk can cause issues during learning:

since the classifier is ultimately designed to minimize the loss only over the training set, it doesn't necessarily output the right prediction when it encounters an example from the data distribution that is unknown to the learning algorithm. In this case, the classifier does not *generalize* well.

This situation where the classifier knows the correct labels "by heart" over the dataset that it was trained on, without making sense of the underlying structure of the $p(x, y)$ distribution (which would lead to correct predictions on new examples), is called overfitting and illustrated in Figure A-4.

Overfitting is a major issue in learning setups, and can be detected on a classifier by observing that its performance on the validation set is significantly below its performance on the training set. While it can be easily detected, it is not easily fought, and many machine learning techniques - e.g. *regularization* as discussed in A.4.2 - are meant to alleviate this problem.

### A.3.3 Restricting to parameterized functions

As mentioned above, one trivial solution to the previous optimization problem is to build a function $f$ that knows the correct label $y_i$ for all the inputs $x_i$ in the dataset, and gives an arbitrary response everywhere else in the domain $\mathcal{X}$.

However we are interested in generalizing beyond the dataset: if two samples $x_a$ and $x_b$ are similar, then their corresponding labels $y_a$ and $y_b$ should be similar as well. In order to enforce a form of continuity on the predictions, there needs to be restrictions on the set of functions $\mathcal{F}$ that we will consider.

The general set of functions $\mathcal{F} = X \to Y$ is difficult to use because it is infinite. In order to perform practical operations, we restrict $\mathcal{F}$ to functions that can be completely defined by a set of continuous parameters $w \in \mathbb{R}^d$ such that:

$$f_w(x) = f(x, w) = \hat{y}, \tag{A.6}$$

with $f_w \in \mathcal{F}$. This makes the optimization problem more tractable because we are

now looking for a finite set of weights $w^*$ such that:

$$w^* := \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} R_{emp}(f(., w)). \tag{A.7}$$

In general we want $f$ to be differentiable w.r.t. $w$; this will allow the use of gradient descent algorithms as described in Section A.4.

### A.3.4 Capacity

The form of $f(., w)$ (how the parameters $w$ are used by the function) is a restriction that defines a subset of $\mathcal{F}$. For example we can require $f$ to be a *perceptron* or a *convolutional neural network* following a chosen architecture. Depending on the restrictions imposed on the set $\mathcal{F}$, classifiers are more or less prone to overfitting, depending on their ability to learn specific patterns in examples, which is called *capacity* and corresponds to the *flexibility* of a set of functions. The notion of capacity corresponds to an idea that is more precisely defined by measures such as the *VC-dimension* described in Vapnik and Chervonenkis [2015].

The capacity of a classifier can be easily adjusted by increasing or decreasing the number of parameters $d$ (dimensionality of $w$) that can be learned during training. There is a careful balance to find when it comes to capacity. If it is too small, then the classifier will not be able to learn the redunduncies in the dataset: the predictions will be erroneous even on the examples that were seen during training. On the other hand, if the capacity is too high, the classifier will learn the labels for the training examples by heart and will output random predictions on new unknown examples, while still minimizing the empirical risk measured on training data. A classifier will work properly if it has just the right amount of capacity: enough to learn the general redundant patterns in the data, but not enough to learn infrequent example-specific patterns irrelevant to the task.

### Conclusion

We show that learning is done by defining a loss (or *cost*) function over a training set, and finding the best function (the one with the lowest loss) within a restricted set of functions $\mathcal{F}$. Choosing this set of functions is done by imposing restrictions that have an effect on the capacity of the classifier. In order to use optimization methods such as gradient descent, we choose in practice a set of functions that are continuously differentiable with respect to their set of parameters $w$.

## A.4 Gradient-based algorithms

The ideal case for optimization is to have a functional $C$ to minimize (which corresponds to the empirical risk or the objective function), and a set of parameters $w$ such that $C$ is differentiable and convex w.r.t. $w$:

$$C(w) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i), \tag{A.8}$$

where $\ell$ is a loss function, and $f$ is a classifier function. In this case, it is possible to use gradient descent to find $w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} C(w)$.

The convexity assumption is generally not met in practice, especially with neural networks; yet we will see in Section A.5, that it is still possible to learn with a non-convex functional $C$ if it is differentiable and meets certain properties.

### A.4.1 Continuous loss functions

We can not perform gradient descent directly on the 0-1 loss function defined in Section A.3 because it is neither continuous nor differentiable. However, it is possible to use continuous approximations for learning.

**Surrogate loss functions.** We can approximate the 0-1 loss with one of the following functions (plots in Figure A-5) and obtain a classifier that produces correct

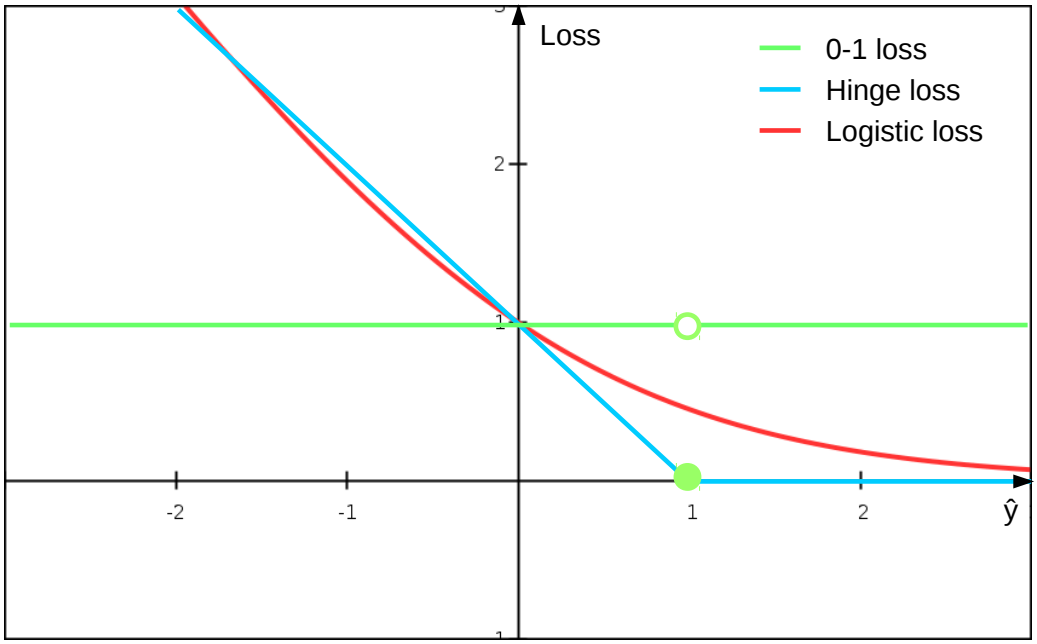



Figure A-5: Surrogate loss functions $\ell(\hat{y}, y)$ approximating the 0-1 loss function in green, for $\hat{y} = 1$. Blue is the hinge loss function $\ell_{hinge}(\hat{y}, y) = \max(0, 1 - y\hat{y})$. Red is the logistic loss function $\ell_{logistic}(\hat{y}, y) = \dfrac{1}{\ln 2} \ln\left(1 + e^{-y\hat{y}}\right)$.

predictions:

$$\text{Hinge loss: } \ell_{hinge}(\hat{y}, y) = \max(0, 1 - y\hat{y}), \tag{A.9}$$

$$\text{Logistic loss: } \ell_{logistic}(\hat{y}, y) = \frac{1}{\ln 2} \ln\left(1 + e^{-y\hat{y}}\right). \tag{A.10}$$

These surrogate loss functions are differentiable, which is useful for gradient descent. We choose one and call it $\ell$. Learning a classifier consists, then, in solving the following optimization problem:

$$w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}}\, C(w) = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) \tag{A.11}$$

Using a continuous surrogate loss function implies that the classifier $f$ outputs values in $\mathbb{R}$ instead of $\mathcal{Y}$ (in the case of binary classification). The classification decision can then be obtained via a post-processing step (such as thresholding the result).

### A.4.2 Regularization

**Regularization.** However, in order to suffer less from overfitting, it can be helpful to add regularization terms to the surrogate loss function. As an example, one common regularization term is the $L^2$ penalty:

$$Reg_{L^2}(w) = ||w||^2. \tag{A.12}$$

The optimization problem becomes:

$$w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}}\, C_{regularized}(w) = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}}\, \left[C(w) + \lambda Reg_{L^2}(w)\right]. \tag{A.13}$$

This new term will add a penalty to solutions where the norm of $w$ is high. It means that the learning algorithm will be more likely to sacrifice some prediction performance on the training set in order to get smaller values for the parameters in $w$. How "likely" depends on the strength of the regularization, which is the value of $\lambda$.

As a result, the function $f(., w)$ becomes smoother, reducing overfitting, and in some cases will generalize better to samples unseen by the learning algorithm. $\lambda$ is a hyper-parameter that can be tuned for optimal generalization performance on the validation set.

**Objective functions.** The sum of the surrogate loss and the (possibly many) regularization terms is called an *objective function*, that we minimize using an optimization algorithm. However, and especially in the context of neural networks, many regularization procedures have no closed-form expression (such as Dropout, described in Srivastava et al. [2014]). In these cases, learning is done by minimizing an objective function that can not be explicitly computed.

### Conclusion.

In order to solve the optimization problem of minimizing the empirical risk, we use a continuous approximation of the loss function described in Section A.3. This lets us build an objective function; adding terms to this objective function steers training towards a set of solutions. In particular, regularization penalties involve tradeoffs between the various terms in the objective function such as sacrificing accuracy on the training set in order to obtain better performance on unseen examples, which is effective in practice for reducing overfitting.

## A.5 Learning with gradient descent

The optimization problem described above can be solved using first-order (gradient descent) methods. We describe here the convergence theorems that we have and introduce *stochastic gradient descent*, which is the most common method for training neural networks. Gradient descent is an example of optimization algorithm that can be used to find a minimizer $w^*$ of a differentiable convex function $C(w)$. The underlying concept is simple:

- for any set of parameters $w_t$ in a sequence, we can compute the gradient $\nabla C(w_t)$ because $C$ is differentiable,
- the opposite of this gradient gives the direction of steepest descent,
- at every iteration $t$ we take a small step towards that direction:

$$w_{t+1} = w_t - \gamma_t \nabla C(w_t), \tag{A.14}$$

- if the steps are small enough, then

$$\forall i, C(w_{t+1}) < C(w_t), \tag{A.15}$$

- if the steps are also large enough, then

$$\lim_{t \to \infty} w_t \to w^* \text{ (optimization succeeds).} \tag{A.16}$$

We illustrate this procedure below, in Figure A-6:

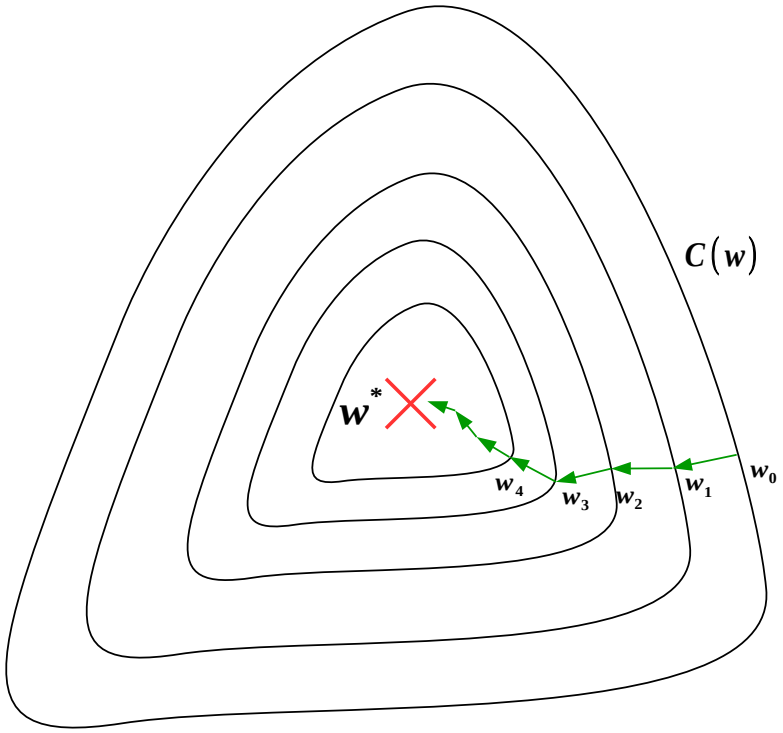



Figure A-6: Illustration of gradient descent. We show level sets of the convex function $C(w)$. We start with an initial parameter $w_0$ and iterate in the direction of steepest descent to progressively reach the minimizer $w^*$.

### A.5.1 Gradient descent convergence

**Theorem 1** (Gradient descent convergence)**.** *Let $C : \mathbb{R}^d \to \mathbb{R}$ be convex and differentiable. Let $\nabla C$ be Lipschitz-continuous with constant $K > 0$:*

$$\forall u, v \in \mathbb{R}^d, ||\nabla C(u) - \nabla C(v)|| \leq K||u - v||. \tag{A.17}$$

*Gradient descent with fixed step size $\gamma < 1/K$ satisfies:*

$$C(w_t) - C(w^*) \leq \frac{||w_0 - w^*||^2}{(2\gamma t)}. \tag{A.18}$$

*This means that gradient descent eventually converges given enough iterations.*

**Theorem 2** (Gradient descent convergence with milder assumptions)**.** *This theorem introduces the notion of general convexity, as proposed in Bottou [1998].*

*Let $C : \mathbb{R}^d \to \mathbb{R}$ be differentiable with the following general convexity properties:*

- *$C$ has a single minimum $w^*$.*
- *The opposite of the gradient points towards the minimum $w^*$:*

$$\forall \epsilon > 0, \inf_{(w-w^*)^2 > \epsilon} (w - w^*) \nabla C(w) > 0. \tag{A.19}$$

- *The gradient does not grow too fast when we move away from the minimum:*

$$(\nabla C(w))^2 \leq A + B(w - w^*)^2; A, B \geq 0. \tag{A.20}$$

*Let $(\gamma_t)_t$ be a sequence such that:*

$$\sum \gamma_t^2 < \infty, \qquad \textit{(step sizes decrease fast enough)}$$
$$\sum \gamma_t = \infty. \qquad \textit{(step sizes allow traveling arbitrary distances in } \mathbb{R}^d\textit{)}$$

*If the update rule for $(w_t)_t$ is:*

$$w_{t+1} = w_t - \gamma_t \nabla C(w_t), \tag{A.21}$$

*then*

$$\lim_t w_t \to w^* \textit{ (optimization succeeds).} \tag{A.22}$$

### A.5.2 Stochastic gradient descent (SGD)

There is, however, an issue when using gradient descent. Recall expression (A.8):

$$C(w) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i), \tag{A.23}$$

Differentiating that expression gives:

$$\nabla C(w) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial w}(x_i, w). \tag{A.24}$$

Computing this gradient has a complexity growing linearly with $N$: this means that for large datasets the cost of computation can become prohibitively expensive.

Stochastic gradient descent deals with that issue. This variant is very similar to gradient descent but only uses an approximation of the true gradient to perform optimization. In practice, this consists of computing the gradient for only a few ($B$) samples at a time then performing a gradient step at each iteration $t$. We define the following random variable, for $B \geq 1$:

$$H(z, w_t) = \frac{1}{B} \sum_{b=1}^{B} \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial w}(z, w_t), \text{ where } z \sim p(x). \tag{A.25}$$

Immediately we derive:

$$\mathbb{E}_z(H(z, w_t)) = \nabla C(w_t). \tag{A.26}$$

The resulting algorithm is similar to gradient descent, except that at every iteration i, we take a step towards a random direction $H(z, w_t)$ that, on average, corresponds to the true gradient. Sampling that function is done by computing the gradient $\frac{\partial C}{\partial w}$ only on a small subset of the dataset containing $B$ samples drawn randomly without replacement. A larger sample size $B$ leads to a better approximation of the gradient. If $B = N$, we obtain the previous gradient descent algorithm.

The following theorem ensures that the algorithm converges almost surely.

**Theorem 3** (Stochastic Gradient Descent convergence)**.** *This theorem is proposed in Bottou [1998] and proves almost-sure convergence in a convex case.*
*Let $C : \mathbb{R}^d \to \mathbb{R}$ be differentiable with the following general convexity properties:*

- *$C$ has a single minimum $w^*$,*
- *the opposite of the gradient points towards the minimum $w^*$:*

$$\forall w \in \mathbb{R}^d, \ \forall \epsilon > 0, \inf_{(w - w*)^2 > \epsilon} (w - w^*) \nabla C(w) > 0. \tag{A.27}$$

*Let $z \sim P$, where $P$ is a probability distribution. Let $H(z, w_t)$ satisfy:*

- *$H$ is a stochastic approximation of the true gradient $\nabla_w f$:*

$$\forall w \in \mathbb{R}^d, \ \mathbb{E}_z[H(z, w)] = \nabla C(w), \tag{A.28}$$

- *the second moment of $H$ has bounded growth:*

$$\forall w \in \mathbb{R}^d,\ \exists A, B \geq 0,\ \mathbb{E}_z[H(z,w)^2] \leq A + B(w - w^*)^2. \tag{A.29}$$

*Let $(\gamma_t)_t$ be a sequence such that:*

$$\sum \gamma_t^2 < \infty, \qquad \textit{(step sizes decrease fast enough)}$$
$$\sum \gamma_t = \infty. \qquad \textit{(step sizes allow traveling arbitrary distances in } \mathbb{R}^d\textit{)}$$

*If the update rule for $(w_t)_t$ is:*

$$w_{t+1} = w_t - \gamma_t H(z, w_t), \tag{A.30}$$

*then*

$$\lim_t w_t \overset{a.s.}{\to} w^* \ \textit{(optimization succeeds almost surely).} \tag{A.31}$$

**Extension to non-convex cases.** Bottou [1998] extends this theorem to prove almost-sure convergence to extremal points by showing that the magnitude of the gradient converges to zero. This implies convergence to extremal points, as shown in Figure A-7. This figure exposes the problem of local minima and asymptotic plateaus. This form of convergence implies that given an random initial set of parameters $w_0$, there is no guarantee that optimization will lead to a satisfying solution.

Neural networks are particularly affected by these issues, as SGD is the core algorithm for training them given the high cost of gradient computation, as we will see in Appendix B. As a consequence, producing and reproducing results is difficult, and much time is usually spent tuning the hyperparameters for training to succeed.

## Conclusion

We show convergence theorems for gradient-based methods, describe stochastic gradient descent, and mention the underlying limitations in the non-convex cases that include neural networks, in particular the convergence towards extremal points that,

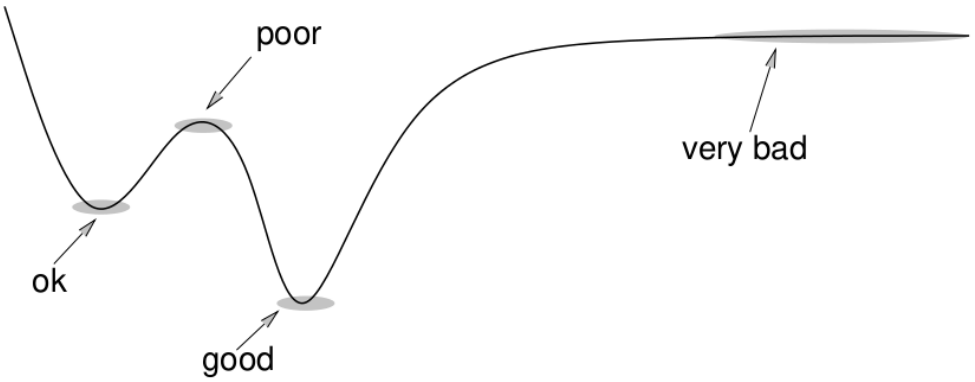



Figure A-7: Bottou [1991] (updated in Bottou [1998]) provides a convergence proof of SGD towards extremal points (shown here). We observe different kinds of extremal points; from left to right: local minimum, local maximum, global minumum, asymptotic plateau. Figure credit: Bottou [1998].

given a random initial point, can make the production and reproduction of results difficult.

## A.6 Conclusion

In this Appendix, we provided a short introduction to supervised classification and its basics. In Section A.1, we described the data that we can use, and showed that in supervised learning, the knowledge present in a dataset is an illustration of the procedure that was used to collect it. This is a core limitation of supervised learning, as in this scenario a machine can only learn concepts that are simple enough for humans to annotate and agree upon.

In Section A.2, we described the problem of *classification*, and showed that many tasks in computer vision can be engineered into one (or many) classification task(s). This special case of machine learning allows for many applications.

In Section A.3, we described the mathematical interpretation of classification as an optimization problem over parameterized functions, where the goal is to minimize what is called the *empirical risk*, which describes the performance of an algorithm on a training set. We mentioned one of the major issues in machine learning, *overfitting*, where a function learns to minimize this empirical risk, but not in a way that allows

satisfying predictions on new samples.

In Section A.4, we introduced continuous loss functions that approximate the empirical risk in a differentiable way, allowing the use of gradient descent methods on the parameters of the classifier functions for this optimization. This allows defining differentiable *objective functions* that notably contain *regularization* terms, sacrificing performance on the training set to obtain better *generalization* and reduce overfitting.

In Section A.5 we mentioned convergence theorems for gradient descent algorithms, and also introduced the very important *stochastic gradient descent* algorithm, which is the standard algorithm for training neural networks, along with its convergence properties and shortcomings in the non-convex case, notably the local extrema issue that makes training neural networks complicated.

The learning framework that we described in this Appendix builds on parameterized differentiable classifier functions; neural networks fit in this class as we will see in the second part of this technical background, in Appendix B.

# Appendix B

# Technical background - Neural networks

## Outline

In this second part, we focus on convolutional neural networks and describe their underlying principles. We saw in Appendix A that classification is a very important building block in computer vision that can be seen as an optimization problem on differentiable parameterized functions. The goal of this Appendix B is to show that neural networks are a particular restriction of the function set, that fits in the learning framework described in Appendix A. We first start with a short math preliminary, in Section B.1 to briefly discuss the dot-product.

Then, in Section B.2 we will describe architectures that define neural networks as parameterized functions and explain the elements that are necessary for executing backpropagation, which is an algorithm that computes gradient. We will see that neural networks are specific arrangements of modules called *layers*.

In Section B.3 we will describe some simple layers that are used in neural networks, notably the fully-connected layers and element-wise nonlinearities. In Section B.4 we will introduce tensors and pooling layers, that exploit their structure. In Section B.5 we will describe the important *convolution layer* which is the core element of convolutional neural networks. In particular we want to expose the heavy linear

algebra operations that are involved in this layer, explaining the great success of GPUs for computation in this context.

In Section B.6 we will provide more insights on the practical usage of these algorithms; we will also mention problems related to random initialization of weights and learning rates along with recent methods for alleviating these issues. We will also mention briefly the case of *recurrent neural networks*.

## B.1 Math preliminaries

Many of the computations in a neural network are based on dot-products. In this section, after introducing some notation on derivatives, we will briefly discuss this operation. We will in particular see that a set of dot-products can be seen as a matrix operation; this will be very important for implementation purposes because computers perform linear algebra very efficiently.

### B.1.1 Partial derivative notation

Neural network training relies heavily on gradient descent, therefore this appendix chapter involves partial derivatives. The representation of data is usually done with tensors - multi-dimensional arrays for which matrices are a special case of dimension 2, and we need to extend the chain rule of derivatives to tensors.

Let $d_1^x, d_2^x, d_3^x, d_1^y, d_2^y, d_3^y$ be positive integers. Given tensors $X \in \mathbb{R}^{d_1^x \times d_2^x \times d_3^x}$, $Y \in \mathbb{R}^{d_1^y \times d_2^y \times d_3^y}$, and a scalar $L \in \mathbb{R}$, we will use the partial derivative notation to extend Jacobian matrices to tensors, such that:

- $\dfrac{\partial L}{\partial X}$ is a gradient tensor $G \in \mathbb{R}^{d_1^x \times d_2^x \times d_3^x}$ with the following entrywise definition:

$$\forall i, j, k \;\; g_{ijk} = \frac{\partial L}{\partial x_{ijk}} \tag{B.1}$$

- $\dfrac{\partial Y}{\partial X} \in \mathbb{R}^{d_1^y \times d_2^y \times d_3^y \times d_1^x \times d_2^x \times d_3^x}$ is a "Jacobian" tensor $T$ with the following entrywise

definition:

$$\forall p, q, r, i, j, k, \quad t_{pqrijk} = \frac{\partial y_{pqr}}{\partial x_{ijk}} \tag{B.2}$$

- the chain rule holds using an appropriate tensor reduction:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial X} \tag{B.3}$$

### B.1.2 The dot-product

The dot-product is an operation in Euclidean spaces that plays a crucial role in neural networks. A large portion of the mathematical operations in neural networks (especially in the *convolution layer*) are more or less complicated combinations of dot-products. We take a look at how it works.

**Geometric view.** Let $W$ and $X$ be two non-zero vectors in $\mathbb{R}^d$. These two vectors define a two-dimensional subspace (at least one) that contains both. Let $(e_1, e_2)$ be an orthonormal basis of this plane, such that $W = ||W||\, e_1$, where $||.||$ is the Euclidean norm. In this case, the dot-product is equal to:

$$Y = (W|X) = ||W||\, ||X||\, \cos(\theta), \tag{B.4}$$

where $\theta$ is the angle in $]-\pi, \pi]$ between $W$ and $X$ in this 2D plane. The angle $\theta$ measures how much $W$ and $X$ point to the same direction, as shown in Figure B-1.
If the angle is large, directions are roughly opposite, $cos(\theta) < 0$ and $Y < 0$.
If the angle is small, the directions are similar, $cos(\theta) > 0$ and $Y > 0$.

**Algebraic view.** The equivalent algebraic view involves the coordinates of the vectors, which is much more convenient for implementation and differentiation. Let $W = [w_1, \ldots, w_d]$ and $X = [x_1, \ldots, x_d]$ be vectors in $\mathbb{R}^d$. Now, the same dot-product

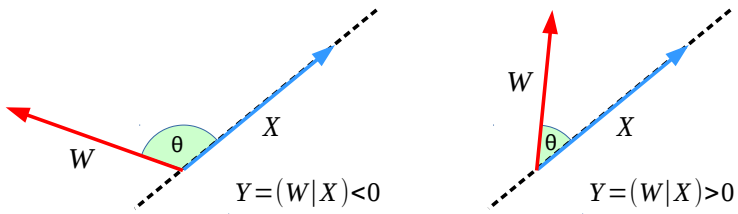



Figure B-1: The sign of the *dot-product* tells whether two vectors point roughly in a similar direction. Left: large angle $\theta$, negative dot-product $Y$; right: small angle $\theta$, positive dot-product $Y$.

can be rewritten :

$$Y = (W|X) = \sum_{i=1}^{d} w_i x_i \tag{B.5}$$

By using (B.5), one can derive the gradients:

$$\frac{\partial Y}{\partial X} = W \tag{B.6}$$

$$\frac{\partial Y}{\partial W} = X \tag{B.7}$$

**Matrix multiplication.** Let $(a_i)_{1 \leq i \leq M}$ and $(b_j)_{1 \leq i \leq N}$ be vectors in $\mathbb{R}^d$.
Let $A \in \mathbb{R}^{d \times M}$ be a matrix such that $A(., i) = a_i$ for all $i \leq M$.
Let $B \in \mathbb{R}^{d \times N}$ be a matrix such that $B(., j) = b_j$ for all $j \leq N$.
Let $C = A^T \times B$, where $\times$ is the matrix multiplication operation. We have $C(i, j) = (a_i|b_j)$ for all $i \leq M$ and $j \leq N$.

When we want to perform dot-products between many vectors $(a_i)$ and many vectors $(b_j)$, all computations can be done within a single matrix multiplication as shown in Figure B-2. It is important to spot matrix operations, because their implementations are **extremely optimized** in computers thanks to the Basic Linear Algebra Subprograms formalism (BLAS, Lawson et al. [1979]). This property is thoroughly used for efficiently computing the *convolution layer* that we describe later in Section B.5, and is a large component of the success of GPUs in neural networks.

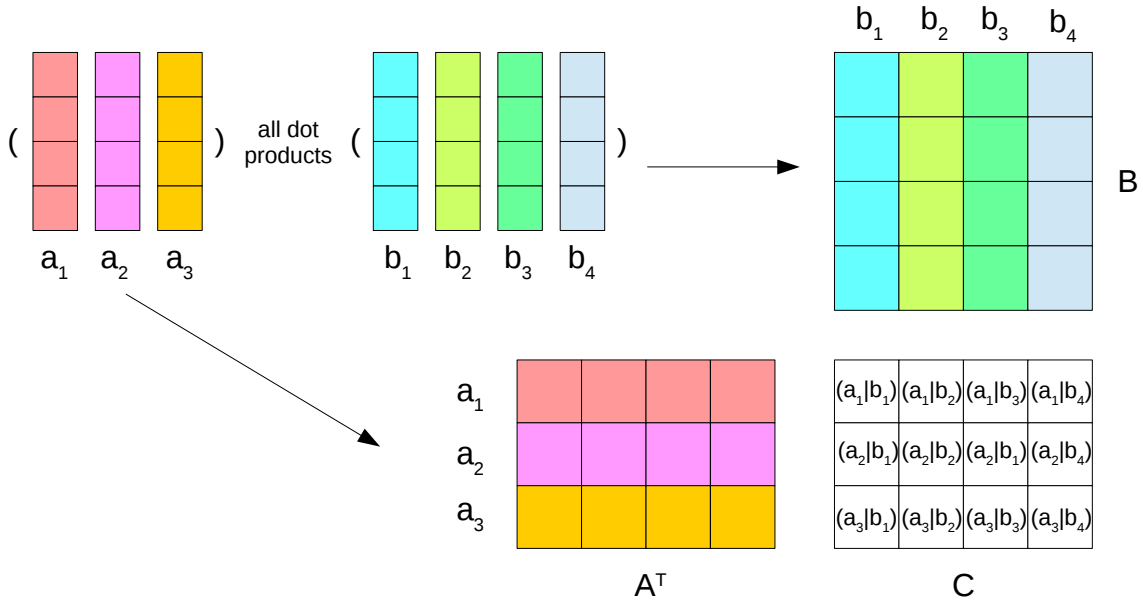



Figure B-2: Computing dot-products between many vectors $(a_i)_i$ and many vectors $(b_j)_j$ can be seen as a matrix multiplication operation for which there are extremely optimized algorithms available, e.g. OpenBLAS (`http://www.openblas.net`) on CPU or cuBLAS (`https://developer.nvidia.com/cublas`) on nVidia GPUs, often close to the limits of the hardware.

## B.2 Neural network architectures

We saw in Appendix A that the classification problem can be addressed using parameterized differentiable functions $f(x, w)$ (with input $x$ and parameters $w$) and optimizing a cost functional $C(w)$ with gradient descent. Given a loss function $L$ and a dataset $\mathcal{D} = \{x_i, y_i\}_{1 \leq i \leq N}$, we want to solve the following optimization problem, as defined in Section A.4, Equation (A.11):

$$w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} C(w) = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{N} L(f(x_i, w), y_i). \tag{B.8}$$

As we will use Stochastic Gradient Descent for this problem, (see Section A.5), we will need to compute $\dfrac{\partial L}{\partial w}$. Our goal in this section is to review the properties that are required to build suitable functions following a modular graph-based architecture, and more importantly, how to compute their gradient.

**Notations.** In this section, we will consider many layers, and therefore use superscripts to refer to the layer index. For a layer indexed with $i$, $f^i$ is the layer function, $x^i$ is the input, $w^i$ is the set of parameters, and $y^i$ is the output. When no superscript is used, we consider the whole network.

### B.2.1 Graph-based architectures

Neural networks are differentiable parameterized functions $f = (x, w) \mapsto \hat{y} \in \mathcal{F} = \mathcal{X} \times \mathcal{W} \to \mathcal{Y}$ defined by a directed acyclic graph (DAG) of nodes called *layers* going from the data input $x \in \mathcal{X}$ to the prediction output $\hat{y} \in \mathcal{Y}$, with a set of parameters $w$. In other words:

- We need to use differentiable and parameterized functions in order to use gradient descent algorithms to train them by optimizing their parameters, as explained in Section A.3. We have $\hat{y} = f(x, w)$.
- We build suitable functions by composing smaller functions following a DAG. This allows the use of the backpropagation algorithm (Rumelhart et al. [1986])

to compute gradients, that we will describe in B.2.2.

- Layers implement these smaller functions, and are the base nodes of this DAG. They can be of different types, as we will show in examples in Sections B.3-B.4-B.5. Each layer indexed by $i \leq n_{layers}$ has a set of parameters $w^i$ such that $w \in \mathbb{R}^d$ contains the elements of all the sets $w^i$. $w^i$ can be empty, which means layer $i$ has no parameters.

We will first describe layered architectures, that are used to define neural networks, and explain the properties that are necessary in layers for implementing backpropagation in a directed acyclic graph.

**Example of neural network architecture.** We start with an example shown in Figure B-3. A neural network is an arrangement of modules such that the input is processed, in a sequence, by a composition of functions. The example shown corresponds to the following function $f$, after adding the weights (Figure B-4). Given $f^1, f^2, f^{3a}, f^{3b}, f^4$, one defines $\hat{y}$ as follows:

$$\hat{y} = f(x, w) = f^4(f^{3a}(f^2(f^1(x, w^1), w^2), w^{3a}), f^{3b}(f^2(f^1(x, w^1), w^2), w^{3b}), w^4), \quad \text{(B.9)}$$

which is quite difficult to read. While this shows that the network effectively corresponds to a mathematical composition of functions, observing the graphs is much easier.

### B.2.2 Backpropagation

The backpropagation algorithm is simply a generalization of the chain rule for composed functions. It allows computing the gradient of a function with respect to each element in a DAG that defines the function. Layers are the building block elements of neural networks that will be laid out on a DAG. Each layer $i$ contains a function $f^i$ and a set of learnable parameters $w^i$ called *weights*, which can be the empty set. The goal is to compute the gradient $\dfrac{\partial L}{\partial w^i}$ for all the layers in the graph, in order to obtain

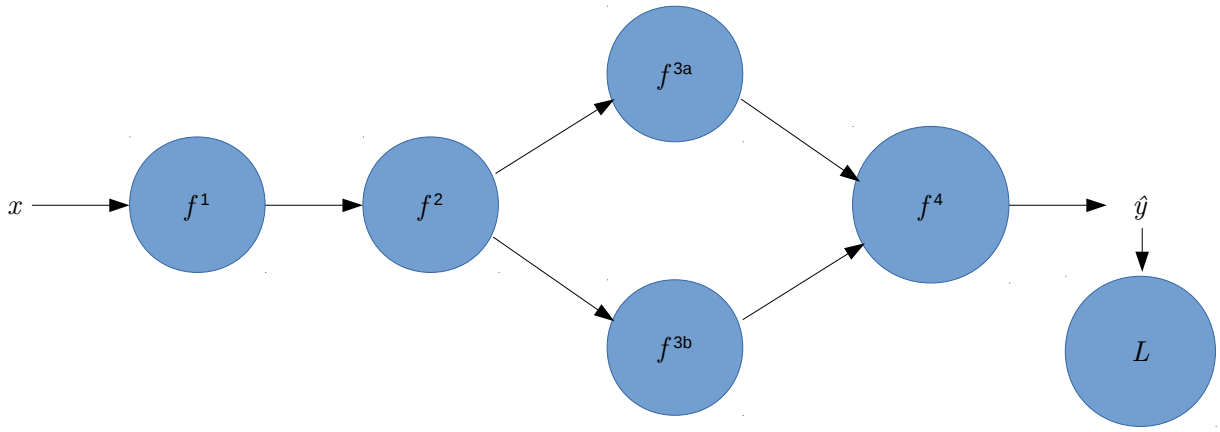



Figure B-3: An example architecture made of 5 nodes laid out on a Directed Acyclic Graph (DAG).

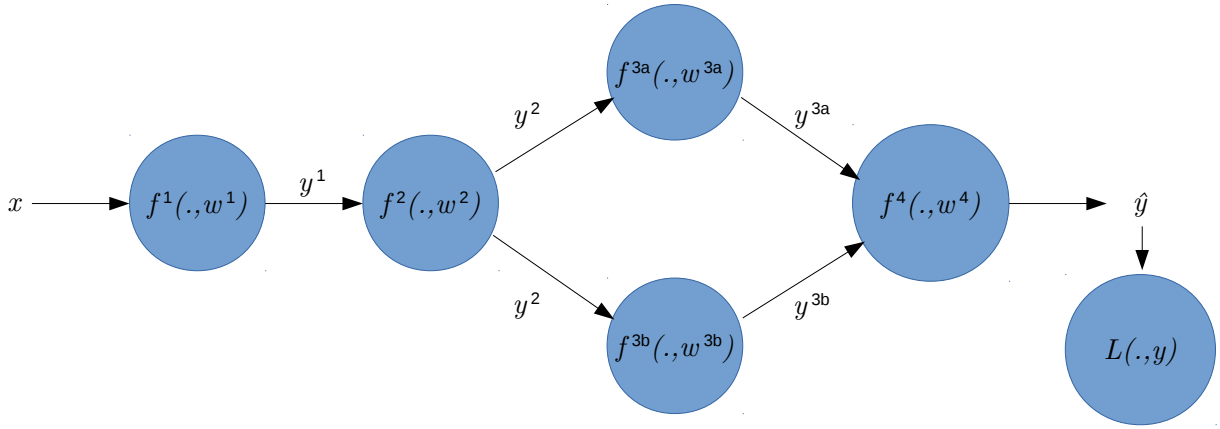



Figure B-4: The same architecture as in Figure B-3, with details exposed, in particular the parameters $w^i$.

$\frac{\partial L}{\partial w}$ during training. For this, a layer $i$ needs to be able to execute three operations, as shown in Figure B-5:

- compute its output $y^i = f^i(x^i, w^i)$,
- propagate the loss gradient to its input: compute $\frac{\partial L}{\partial x^i}$, given $\frac{\partial L}{\partial f^i}$. This is the chain rule for composed functions:

$$\frac{\partial L}{\partial x^i} = \frac{\partial L}{\partial f^i}\frac{\partial f^i}{\partial x^i}, \tag{B.10}$$

- propagate the loss gradient to its weights: compute $\frac{\partial L}{\partial w^i}$, given $\frac{\partial L}{\partial f^i}$. This is the chain rule for composed functions:

$$\frac{\partial L}{\partial w^i} = \frac{\partial L}{\partial f^i}\frac{\partial f^i}{\partial w^i}. \tag{B.11}$$

**Directed and Acyclic.** A directed acyclic graph ensures that the network can be traversed in order, such that when we want to evaluate a layer, we know its inputs are available. We show a topologically sorted version of our example architecture in Figure B-6. Similarly, when evaluating gradients for a layer, we know that the gradient of the loss with respect to the outputs is available.

**Forward propagation.** We can evaluate the prediction $\hat{y} = f(w, x)$ by traversing the graph in the forward direction. The layers in this graph use the outputs of other layers as inputs (except for the first). Since the graph is directed and acyclic, layers can be topologically sorted (see Figure B-6) and evaluated individually in a sequence until the output is reached, where we collect the prediction $\hat{y}$.

**Backpropagation.** Backpropagation is the algorithm that allows computing the gradients within a DAG of layers. Given the loss gradient $\frac{\partial L}{\partial \hat{y}}$, we traverse the sorted graph backwards, propagating the loss gradient at each step to the layers, then to their weights, as we illustrate in Figure B-7.

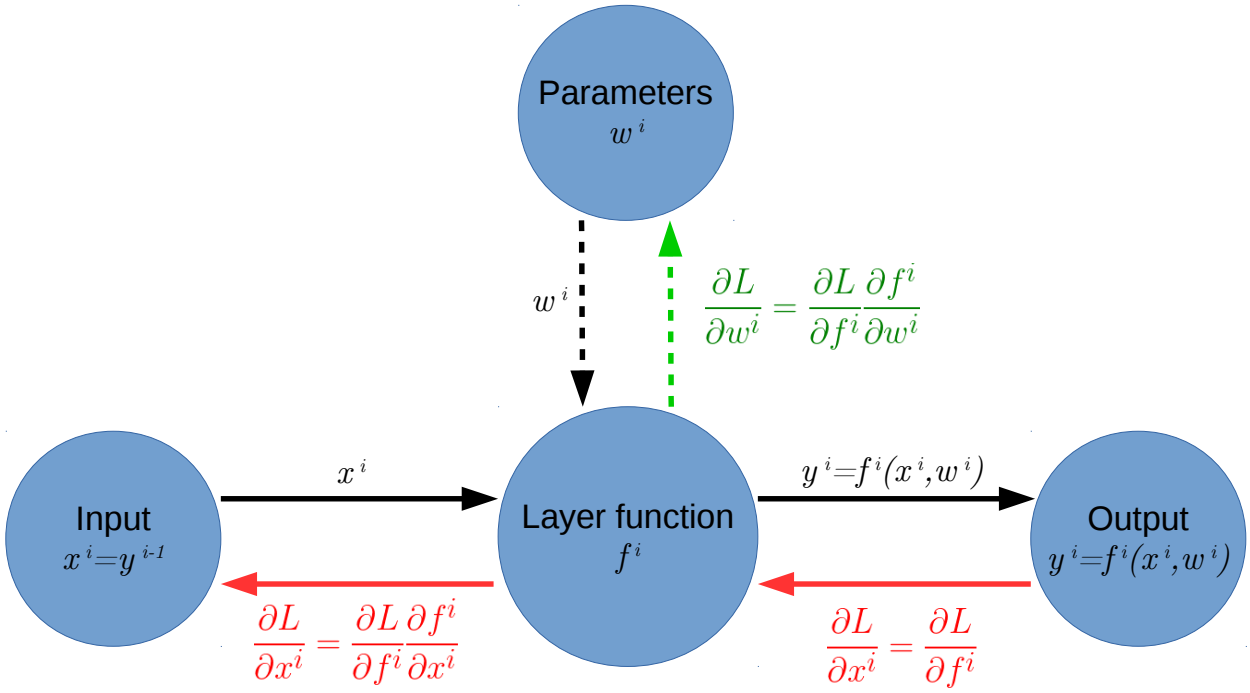



Figure B-5: The operations performed by a layer. In black, the forward operations for evaluating the output of the layer. In red and green, operations related to gradient computation. $\frac{\partial f^i}{\partial w^i}$ and $\frac{\partial f^i}{\partial x^i}$ can be used in the chain rule because the function $f^i$ is known.

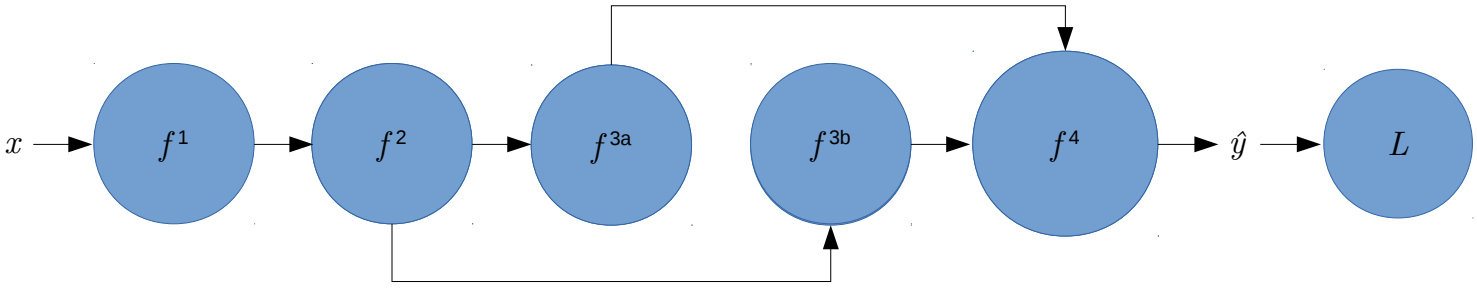



Figure B-6: Topologically sorted graph corresponding to the example in Figure B-3. When the graph is directed and acyclic, it can be traversed in order, which allows forward (going right) and backward propagation (going left). Topological sorting ensures that all the outputs of the *parents* are available to their *children* when they are evaluated.

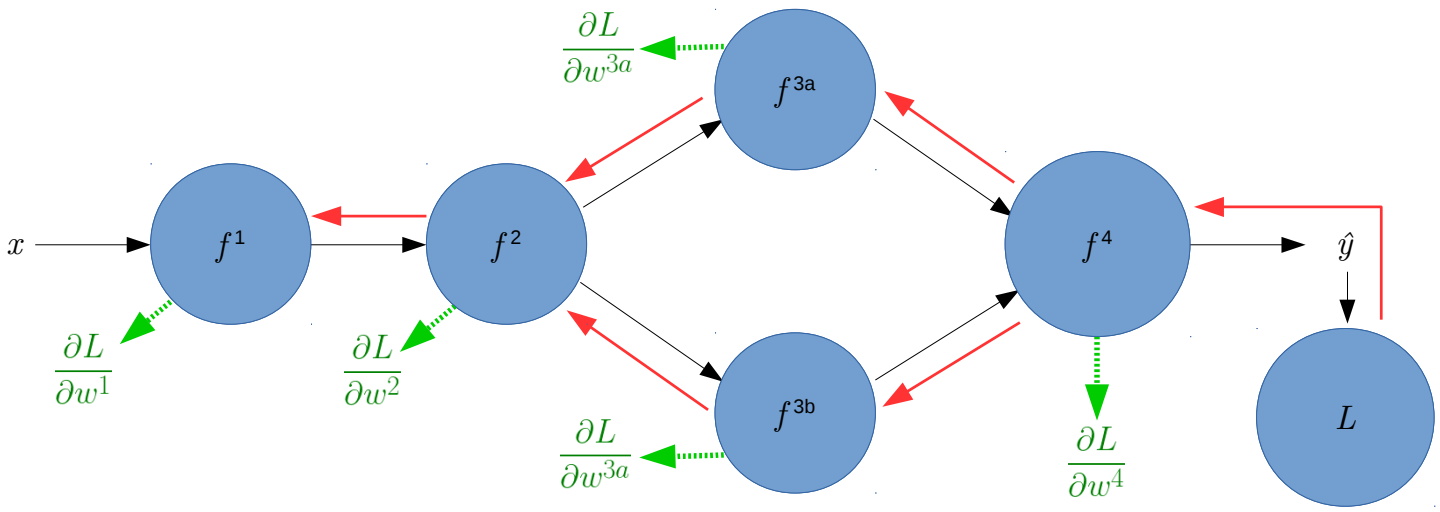



Figure B-7: Backpropagation. The loss function L uses ground truth annotations to provide gradient with respect to the predictions $\hat{y}$. This gradient is then propagated (in red) in the network. Each layer can then compute the gradient with respect to its weights (in green) as shown in Figure B-5.

**Summary.** A neural network is an arrangement of layers in a graph representing a differentiable parameterized function $f$ such that $\hat{y} = f(x, w)$ that fits in the framework described in the previous section.

If this graph is directed and acyclic, it can be sorted topologically, enabling forward propagation and backpropagation, which is an algorithm for computing the gradient of a graph-based function, generalizing the chain rule.

Each individual layer needs to be able to execute the three following functions for the procedure to succeed:

- Compute its output given an input: $y^i = f^i(w^i, x^i)$.
- Compute the gradient with respect to its input, given the gradient with respect to its output: $\frac{\partial L}{\partial x^i} = \frac{\partial L}{\partial f^i}\frac{\partial f^i}{\partial x^i}$.
- Compute the gradient with respect to its parameters if any, given the gradient with respect to its output: $\frac{\partial L}{\partial w^i} = \frac{\partial L}{\partial f^i}\frac{\partial f^i}{\partial w^i}$.

These are the only requirements, and they allow building setups with arbitrarily complex architectures. With the gradient available, the neural network can be trained

to optimize the objective using gradient descent methods. In Sections B.3-B.4-B.5 we will introduce example layers that have all the required properties, and can be used as building blocks for designing neural networks.

**Jacobians.** In this section, the partial derivatives $\frac{\partial f^i}{\partial x^i}$ and $\frac{\partial f^i}{\partial w^i}$ are in fact Jacobian matrices. In some cases, such as the matrix-vector multiply ($Y = WX$, see below), the Jacobian matrix (here, $W^T$) is easy to obtain in closed form. But in more complicated cases, this Jacobian is only useful for theory; in practice $\frac{\partial L}{\partial x^i}$ and $\frac{\partial L}{\partial w^i}$ are computed directly from the gradient $\frac{\partial L}{\partial y^i}$ without using Jacobians.

## B.3 Simple layers

We now describe some important layers that are used in neural networks, as well as their gradient computation.

**Notations.** In this section we consider only a single layer at a time, and subscripts will only be used to index elements in vectors. A vector $X \in \mathbb{R}^p$ is described by a sequence of elements $(x_i)_{1 \leq i \leq p}$.

### B.3.1 Fully-connected layer

This layer takes a vector $X \in \mathbb{R}^p$ and outputs a vector $Y \in \mathbb{R}^n$ such that:

$$Y = WX + B \tag{B.12}$$

where $W \in \mathbb{R}^{n \times p}$ (the weight matrix) and $B \in \mathbb{R}^n$ (the bias matrix) are the parameters for this layer[1]. The gradient can be propagated as follows:

$$\frac{\partial L}{\partial X} = W^T \frac{\partial L}{\partial Y} \tag{B.13}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} X^T \tag{B.14}$$

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial Y} \tag{B.15}$$

This layer has $(p+1)n$ parameters in total. This number of parameters is defined when defining the layer (and the size of the matrix $W$). The input to this layer can be a *flattened* tensor, which is the same tensor but viewed as a vector with the same number of elements.

We illustrate the operations in Figure B-8.

[1] With the notations of Section B.2, if layer $i$ is a fully-connected layer, then $w^i$ is a vector containing all the values in $W$ and $B$.

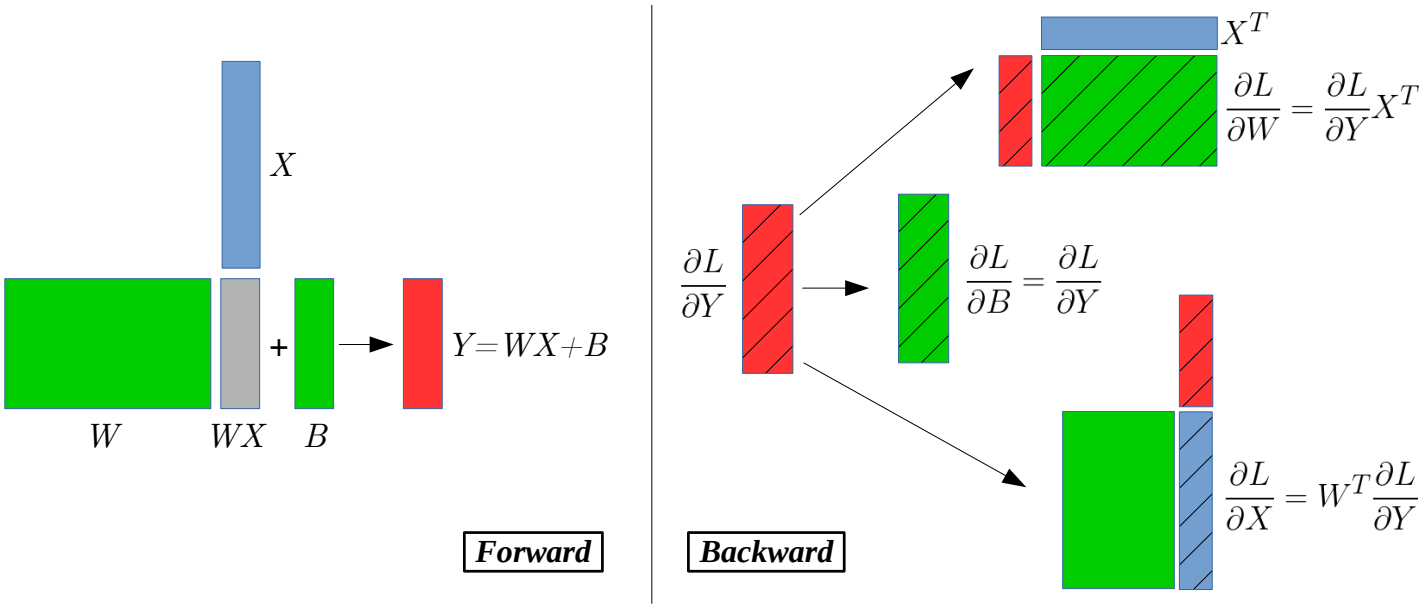



Figure B-8: The *fully-connected layer* takes its input $X$ and applies a matrix-vector operation with the weight matrix $W$ and adds a bias vector $B$. We show the corresponding operations for computing the gradients on the right. Hatched rectangles are gradients of the loss. Each value in $Y$ depends on all the values of $X$, hence *fully-connected*.

### B.3.2 Elementwise nonlinearity

This layer takes a vector $X \in \mathbb{R}^p$ and outputs a vector of same dimensionality $Y \in \mathbb{R}^p$, by processing each coordinate independently with a non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$ such that:

$$\forall i \in [1, p], y_i = \sigma(x_i) \tag{B.16}$$

The gradient can be propagated as follows:

$$\forall i \in [1, p], \frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y_i} \frac{\partial \sigma}{\partial x_i} \tag{B.17}$$

where $\frac{\partial \sigma}{\partial x_i}$ corresponds to the slope of the function $\sigma$ at $x_i$. This layer has no weights.

In common network architectures, many nonlinearity layers are present systematically after each weighted layer except for the last one. During backpropagation,

the gradient values are multiplied by the slope, which can be small especially in the case of saturating functions. With many nonlinearities, this can have a multiplicative effect: the gradient magnitude can become very small, making optimization difficult. This is known as the *vanishing gradient problem*.

Popular non-linear functions (also called *activation functions*) include the Rectified Linear Unit (ReLU), Hyperbolic tangent (tanh) and Sigmoid; we show plots in Figure B-9. Nonlinearities are the main reason why neural networks are not convex functions[2]; for example, a sequence of two fully-connected layers is an affine transform, which is a convex function. But a nonlinearity between these layers makes it a non-convex function, giving it higher capacity and flexibility, and higher potential for classification purposes.

### B.3.3 Softmax.

The softmax function is a particular non-linear function that transforms an input such that the coordinates are positive and sum to one; the transformed vector can then be interpreted as a probability distribution. It is the function $\psi$ such that $Y = \psi(X)$ with:

$$\forall i \in [1, p]\,,\ y_i = \frac{\exp(x_i)}{\sum_{j=1}^{p} \exp(x_j)}. \tag{B.18}$$

This function builds a probability distribution such that higher values of the input are exponentially preferred. This can be used in particular as a final layer for multi-class forced-choice classification, as done by Krizhevsky et al. [2012] (see Section B.6).

[2] A max-pooling layer, which will be described in the next section, is also non-linear.

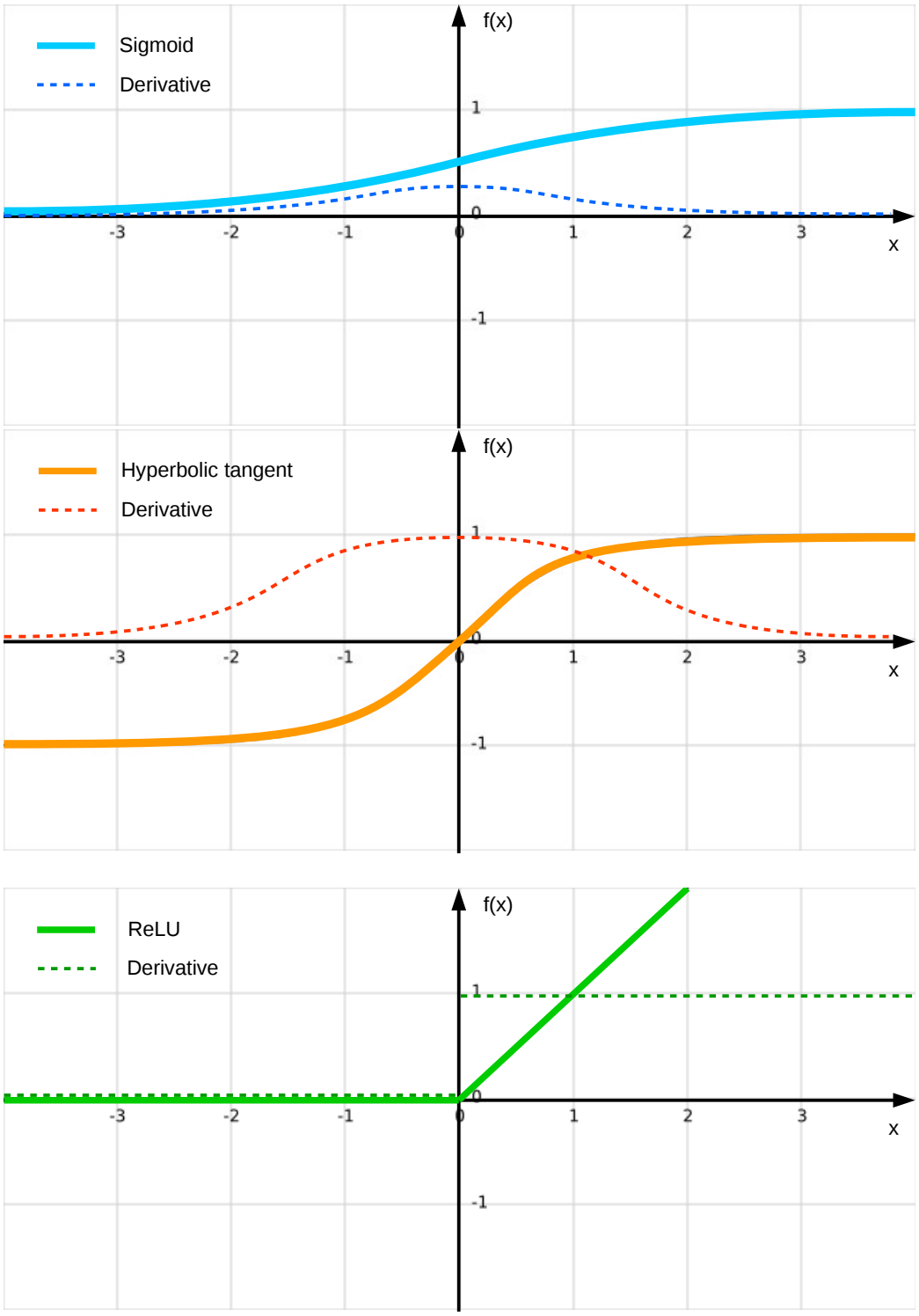



Figure B-9: Non-linear activation functions used commonly in neural networks.
Top: Sigmoid: $\sigma(x) = \dfrac{1}{1+e^{-x}}$. Middle : Hyperbolic tangent: $\tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.
Bottom : Rectified Linear Unit: $\mathrm{ReLU}(x) = \max(0, x)$.
ReLU is non-saturating and allows propagating gradient easily at the cost of unbounded activation. Sigmoid and Tanh can make gradient vanish if the activation is in the flat gradient regime.

## B.4 Tensor layers

In convolutional neural networks, in order to preserve the spatial information present in images, data is often represented as a *tensor*. Tensors are a generalization of *matrices*. One can see matrices as a special case of tensors, where the tensor rank is 2 (rows, columns).

**Notations.** In this section we consider only a single layer at a time, and subscripts will only be used to index elements in tensors. A three-dimensional tensor $X$ has an element $x_{ijk}$.

### B.4.1 Feature maps

We start with the example of RGB (Red, Green, Blue) color images. Each pixel is represented by 3 values R, G, B. As such, an image can be represented by a tensor of size $3 \times height \times width$. Each matrix of size $height \times width$ corresponds to a *feature map*: it represents the value of the corresponding feature (e.g. "how red ?"). We show an example of a $5 \times 5$ pixel RGB image in Figure B-10 to illustrate the corresponding $3 \times 5 \times 5$ tensor. We define the indexing of tensors elements as shown in that same figure.

Convolution layers generalize this by representing data over more feature maps according to patterns called *convolution filters*; we will describe this after the pooling layers below.

### B.4.2 Pooling layers

Pooling layers, such as the *max-pooling* or the *average-pooling*, are special types of layers that are applied, in the case of images, to each feature map separately. They consist of aggregating neighboring elements in order to obtain an output of smaller height and width.

The sizes of the output map, $h_{out}$ and $w_{out}$, depend on the definition of the grid, usually smaller than those of the input $h_{in}$ and $w_{in}$. Pooling performs subsampling.

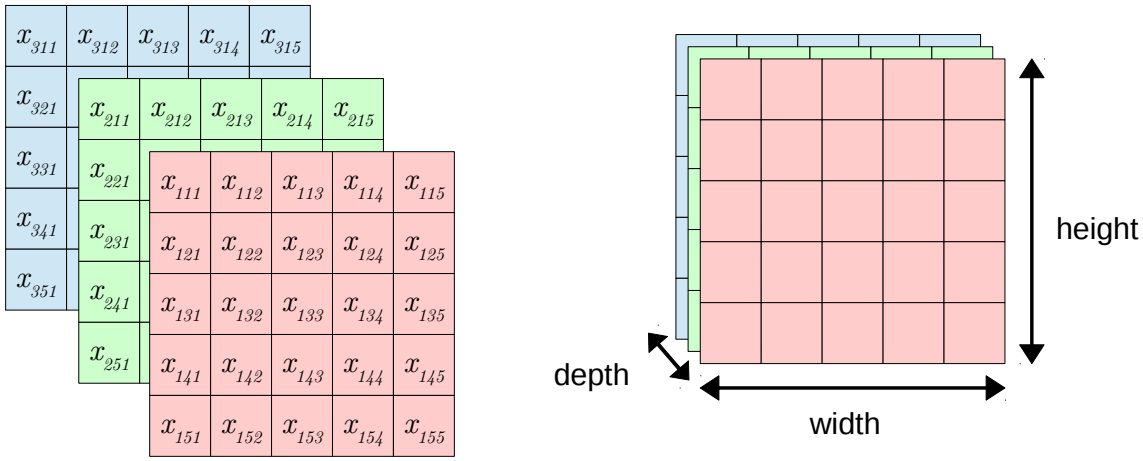



Figure B-10: A $5 \times 5$ pixel RGB image (left) can be seen as a $3 \times 5 \times 5$ tensor (right). We call the sizes of the tensor width, height, and depth as shown. Feature maps are concatenated over the depth dimension.
Indexing of tensor elements is done such that for $x_{abc}$, $a$ corresponds to the depth dimension, $b$ to the height dimension, $c$ to the width dimension.

We illustrate this procedure in Figure B-11.

**Defining a grid with sizes and strides.** In order to pool, one needs to define the arrangement of the *pooling cells* $C(i, j)$ indexed by their position in the grid: these define the neighborhood structure of the pooling operation.

The *pooling size* defines the size of the cell, and the *pooling stride* defines the distance between two consecutive cells.

Figure B-12 shows a simple cell grid, of size 2 and stride 2. Figure B-13 shows a more complicated *overlapping pooling* grid, of size 3 and stride 2.

**The pooling function.** The pooling function is then applied to each of the cells defined by the size and stride. Popular pooling functions include:

- *max*, returning the maximum element of the cell.
- *average*, returning the average value of the cell.
- $L^p$*-norm*, returning the $L^p$ norm of the cell.

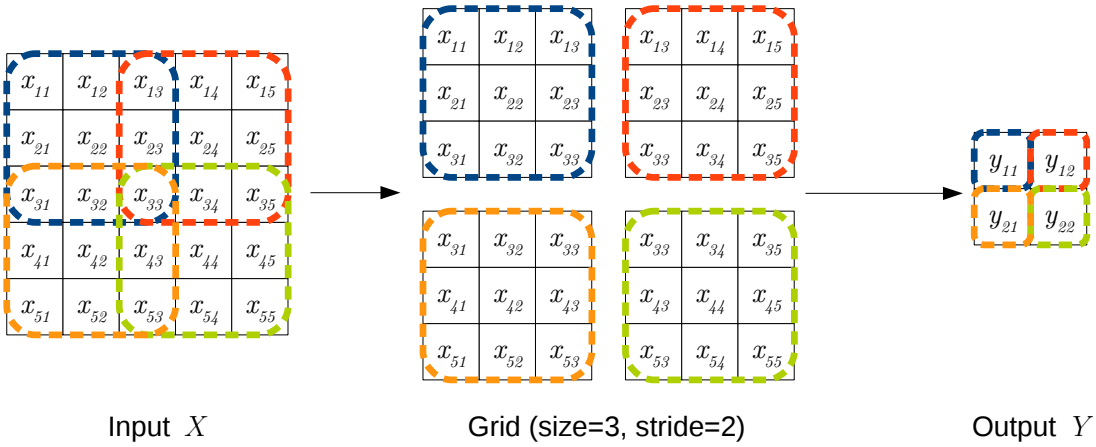



Figure B-11: Pooling operations consist of two steps: decomposing the input in a grid depending on parameters called the *size* (here 3) and the *stride* (here 2), then processing each cell with a function $f$ to obtain an element of the output. In this illustration, we have $y_{11} = f(x_{11} \dots x_{33})$, $y_{12} = f(x_{13} \dots x_{35})$, and so on.

We show an example of a max-pooled image and an average-pooled image in Figure B-14; the differences are subtle but the max-pooling operation appears to subsample with sharper details.

**Backpropagation.** In a first step, the gradient is computed separately for each cell, then the contents of the cells are arranged back into a matrix of same size as the input, to obtain the input gradient $\dfrac{\partial L}{\partial X}$. In the case of overlapping grids, the overlapping portions are summed together as illustrated in Figure B-15.

**Max-pooling case.** While the average and the $L^p$-norm functions are immediately differentiable, this is not the case for the max function. For the max-pooling operation, computing the gradient is done in the following way:

- As a result of the forward operation, we have $y_{ij} = x_{k^* m^*}$, where $k^*, m^*$ are the coordinates of the maximal element.
- Within the corresponding cell, we set $\dfrac{dL}{dx_{km}} = \dfrac{dL}{dy_{ij}}$ if $(k, m) = (k^*, m^*)$, and 0 otherwise.

This means that the gradient is backpropagated through the maximal element of each

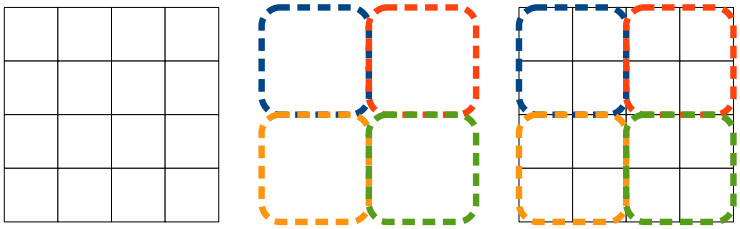


Figure B-12: We show an example of a grid of size 2 and stride 2 used to cover an input of size $4 \times 4$.

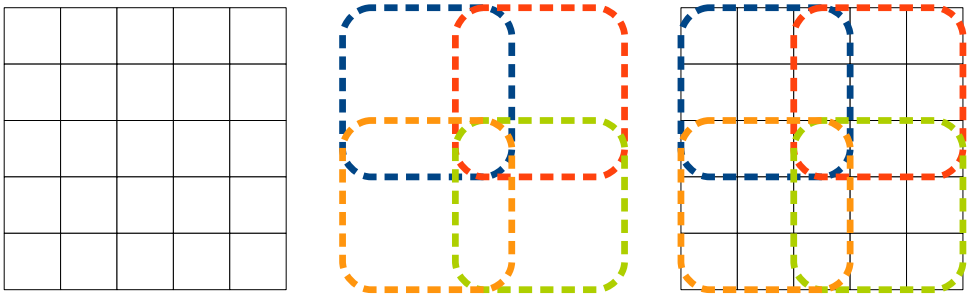


Figure B-13: We show an example of an *overlapping* grid of size 3 and stride 2 used to cover an input of size $5 \times 5$.

cell. This is consistent because the output of the *max* function only depends on the largest element, locally. We show an example of a max-pooling operation in Figure B-15.

**Corner cases.** It is possible that for a given pooling size and stride, the grid does not cover the input exactly. Deciding what happens is a choice in the implementation. Usually, the input is padded with zero elements (explicitly or implicitly) in order to avoid this case.

In the case of an overlapping grid, it is possible that an element backpropagates gradients from many cells. In this case, the gradient corresponding to an element corresponds to the sum of the gradient contributions of the cells in which it is involved. *Grid-summing* is illustrated in the example of Figure B-15 (see blue and red cells). If within a cell, two elements have the maximum value, then choosing which element $(k^*, m^*)$ backpropagates the gradient is also an implementation choice. However, this collision case does not happen often with floating-point numbers in practice as those

Figure B-14: Different pooling operations. Left: Original Grace Hopper image. Middle: result of max-pooling with grid of size 5, stride 4. Right: result of average-pooling with the same grid. We upscale the results to allow for comparison with the original image. Some details, such as the contours of the coat, appear sharper in the max-pooling case and blurrier with average-pooling.

commonly used in neural networks[3].

[3]It can happen when a ReLU nonlinearity outputs 0, but in this case the propagated gradient is multiplied by 0 and doesn't cause issues.

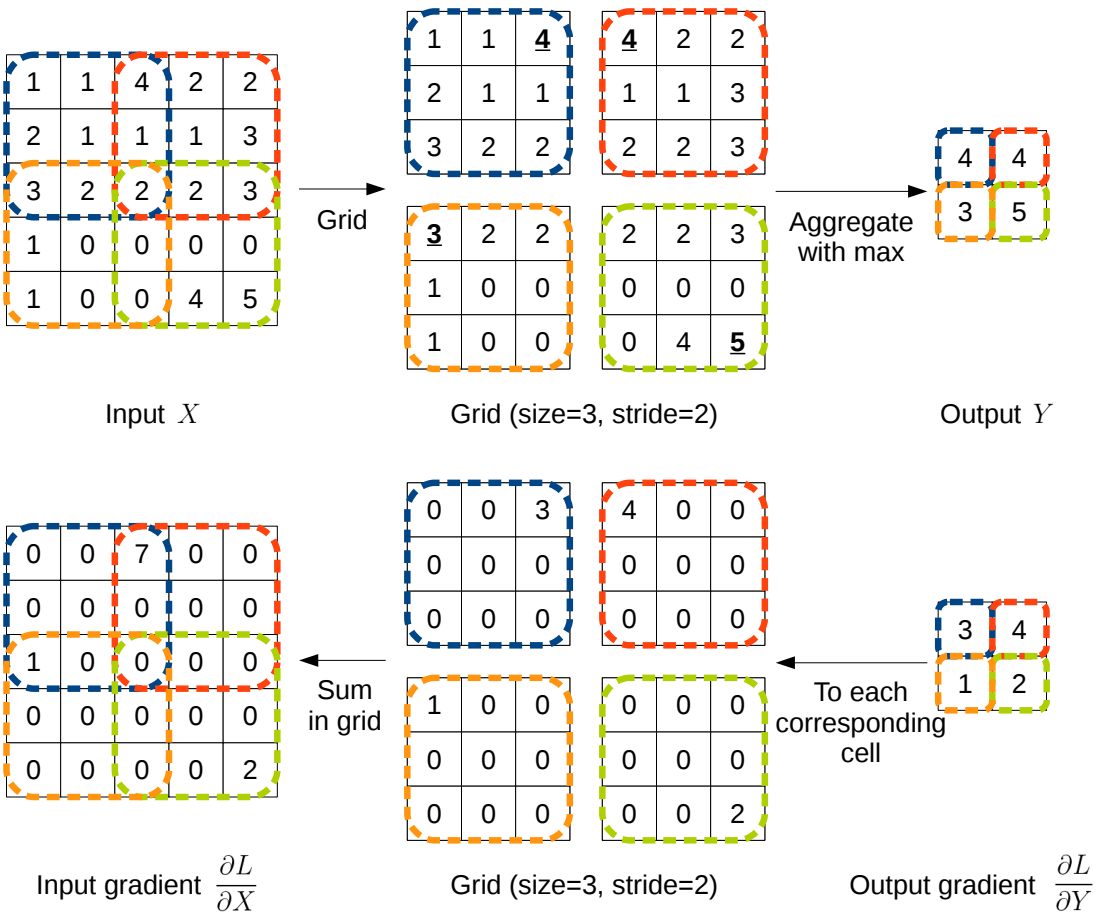



Figure B-15: Example of max-pooling in a size 3, stride 2 case.

Top: forward propagation. For each cell we extract the largest element (bold underlined) of the cell and put it in the output.

Bottom: gradient backpropagation. We compute the gradient corresponding to each cell. In the max-pooling case we copy the value of the output gradient $\frac{\partial L}{\partial Y}$ to the position of the largest element within each cell, then sum them according to the grid to obtain the input gradient $\frac{\partial L}{\partial X}$.

The red and blue cells overlap, propagating a value of $7 = 3 + 4$ in the first row.

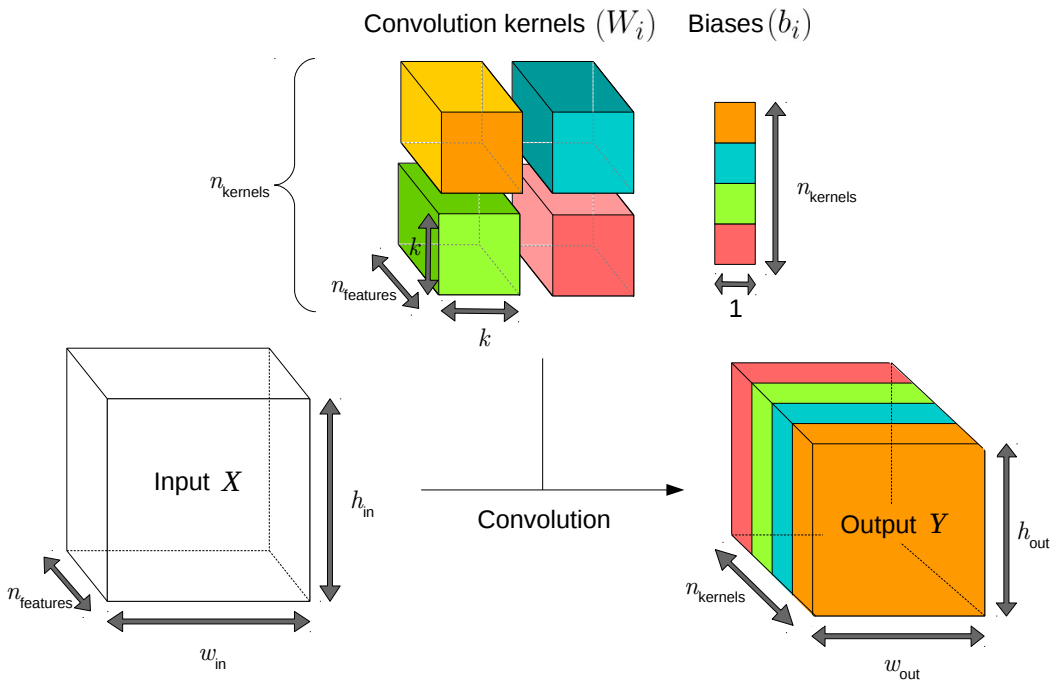



Figure B-16: A convolution layer takes an input tensor $X$, consisting of $n_{features}$ concatenated feature maps, and processes it with $n_{kernels}$ kernels $W_i$ and biases $b_i$ to obtain independent feature maps, concatenated into an output $Y$.

## B.5 Convolution layers

### B.5.1 Overview.

Convolution layers are applied to data tensors containing arbitrarily many feature maps concatenated over the depth dimension. A convolution layer processes a tensor $X$ that consists of $n_{features}$ input feature maps of size $(h_{in} \times w_{in})$ concatenated over the depth dimension.

Each of the $n_{kernels}$ convolution kernels $W_i$, of size $(n_{features} \times k \times k)$ (with $k > 0$) outputs a single feature map by performing a convolution operation and adding a bias value $b_i$ shared across the map. The maps are then concatenated over the depth dimension to output a tensor $Y$ of size $n_{kernels} \times h_{out} \times w_{out}$, as shown in Figure B-16.

The convolution operation consists of sliding a filter tensor of same depth dimensionality over the width and height dimensions, applying dot-products on each cell (sometimes with a stride between consecutive cells). We will describe it next.

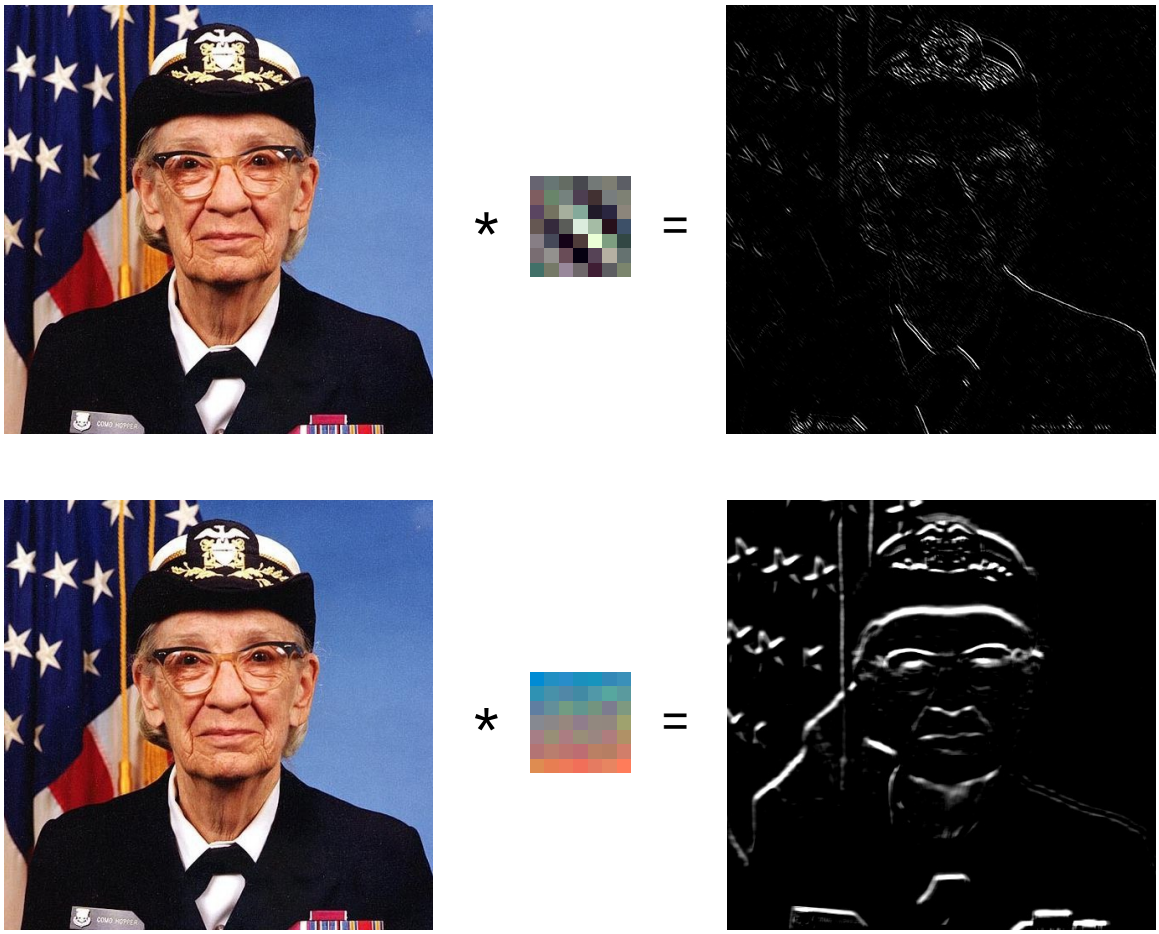


Figure B-17: Example output of a convolution layer with ReLU nonlinearity applied on an RGB image. On each row: input image, convolution kernel and output. We see that the top kernel extracts diagonal sharp edges (top-left to bottom-right). The bottom kernel extracts smoother horizontal edges.

**Convolving an image with a filter** We show in Figure B-17 an example output of a convolution layer applied on an image, to build intuition. Applying a convolution kernel over an image extracts features. In this very simple case, the input is an 3-channel RGB image, and we can observe that the output maps react to edges of different orientations. In the upper layers of a network, however, it is more complicated to obtain an interpretable visualization, as the number of channels does not allow it. However, Zeiler and Fergus [2014] provide a visualization in Figure B-24, that we will discuss in Section B.6.

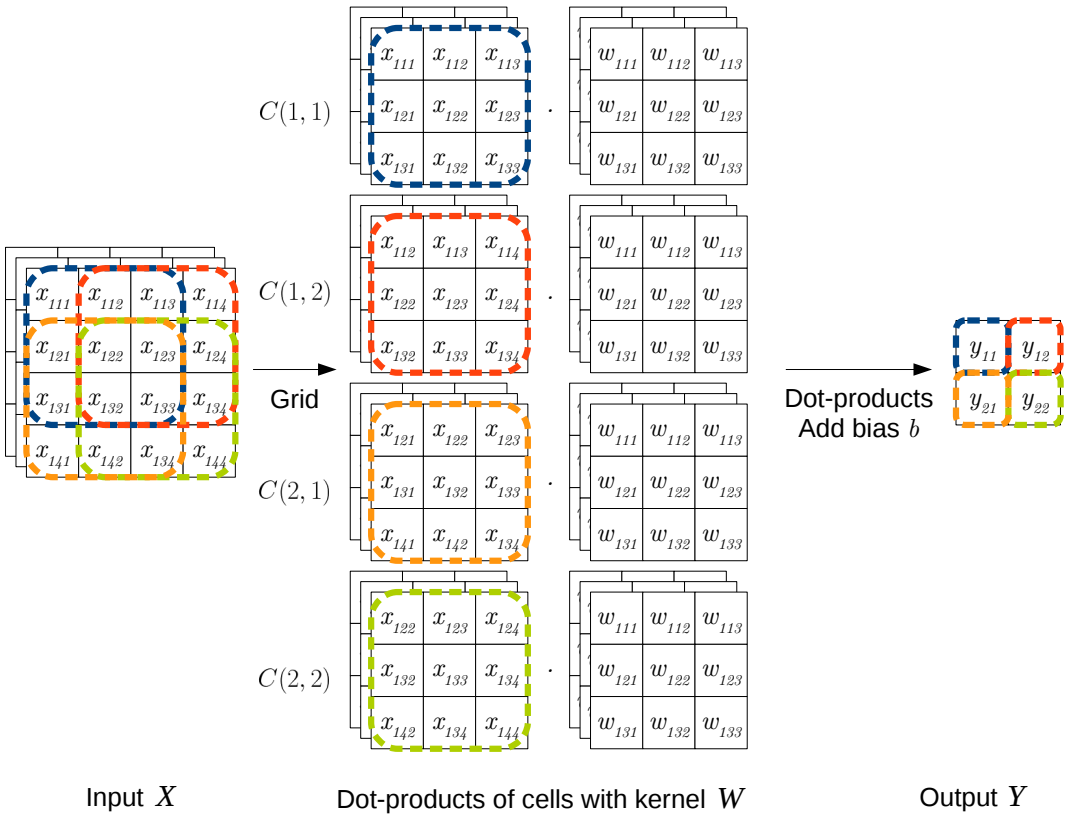



Figure B-18: **Forward computation of the convolution layer** for a single kernel $W$. We extract the cells corresponding to the grid, similarly to the pooling operations. Then, for each position $(i, j)$, the cell $C(i, j)$ is used in a dot-product with the kernel $W$; we add a bias $b$ to obtain the output value $y_{ij}$.

**Tensor dot-product.** We first define the dot-product between two tensors of same size U and V $\in \mathbb{R}^{a \times b \times c}$, similarly to a vector dot-product:

$$(U|V) = \sum_{i,j,k} u_{ijk} v_{ijk} \tag{B.19}$$

## B.5.2 Single kernel forward computation.

We first start with the single-kernel case, where the convolution layer applies a single kernel to its input.

In the convolution layer, the input $X$ is a tensor of size $n_{features} \times h_{in} \times w_{in}$. A (square) weighted convolution kernel $W$ is a tensor of size $n_{features} \times k \times k$. We show the convolution of a single kernel $W$ with an input $X$ in Figure B-18.

First, the input $X$ is cut into a grid over the spatial dimensions (height and width), such that each cell tensor $C$ matches the size of the kernel tensor $W$. We index $C(i, j)$ by its spatial position $(i, j)$ in the grid.

Then, each cell $C(i, j)$ is used in a dot-product with $W$ to obtain the value $y_{ij}$ in the output feature map $Y$. A bias value $b$, shared across the feature map for this kernel, is then added to all elements. This gives:

$$\forall i, j,\ y_{ij} = (W|C(i, j)) + b \tag{B.20}$$

**Remarks.** We note that the output height $h_{out}$ and width $w_{out}$ are smaller than those of the input as a consequence of the cell grid definition. When necessary, this issue can be addressed by padding the input tensor with enough zeros.

As a result of the convolution operation, the input is processed *locally*: each cell corresponds only to a small spatial portion of the input. In contrast, fully-connected layers process their input globally, as they have no notion of sparsity or spatial distance.

The convolution operations usually account for the large majority of floating-point operations in a neural network (as we will see in B.6.2), therefore optimizing their execution is a critical issue that was addressed using GPU implementations by Krizhevsky et al. [2012], explaining part of their current success.

### B.5.3 Single-kernel backpropagation to inputs.

We saw in Section B.1 that for a dot-product $y_{ij} = (W|C(i,j))$, deriving the gradients gives: $\frac{\partial y_{ij}}{\partial C(i,j)} = W$, and $\frac{\partial y_{ij}}{\partial W} = C(i,j)$. For backpropagation, we first compute the gradient for each cell ($\frac{\partial L}{\partial y_{ij}}$ is a scalar):

$$\frac{\partial L}{\partial C(i,j)} = \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial C(i,j)} = \frac{\partial L}{\partial y_{ij}} W \tag{B.21}$$

then sum the contributions of each cell within a tensor $\frac{\partial L}{\partial X}$ of same size as the input, by following the grid, adding up the overlapping portions, as shown in Figure B-19:

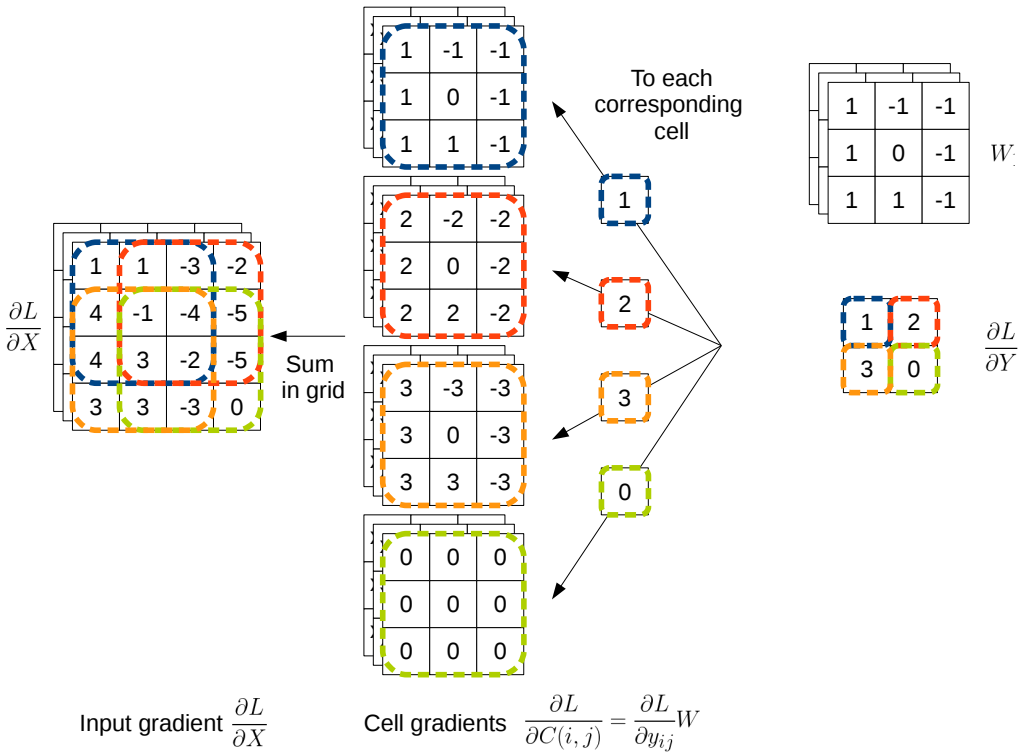



Figure B-19: **Input gradient** computation of the convolution layer for a single kernel $W$. We use number values for clarity in the grid-summing operation.
We first compute the gradient corresponding to each cell $\frac{\partial L}{\partial C(i,j)} = \frac{\partial L}{\partial y_{ij}} W$.
Then, we sum the contributions back into the input gradient $\frac{\partial L}{\partial X}$ following the grid.

### B.5.4 Single-kernel backpropagation to weights and biases.

Similarly, for backpropagating to the kernel weights, we compute the contributions of each cell $C(i,j)$ such that:

$$\frac{\partial L}{\partial W} = \sum_{i,j} \frac{\partial L}{\partial y_{ij}} C(i,j). \tag{B.22}$$

For the bias:

$$\frac{\partial L}{\partial b} = \sum_{i,j} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial b} = \sum_{i,j} \frac{\partial L}{\partial y_{ij}} \times 1 = \sum_{i,j} \frac{\partial L}{\partial y_{ij}}. \tag{B.23}$$

We illustrate the operations in Figure B-20 below:

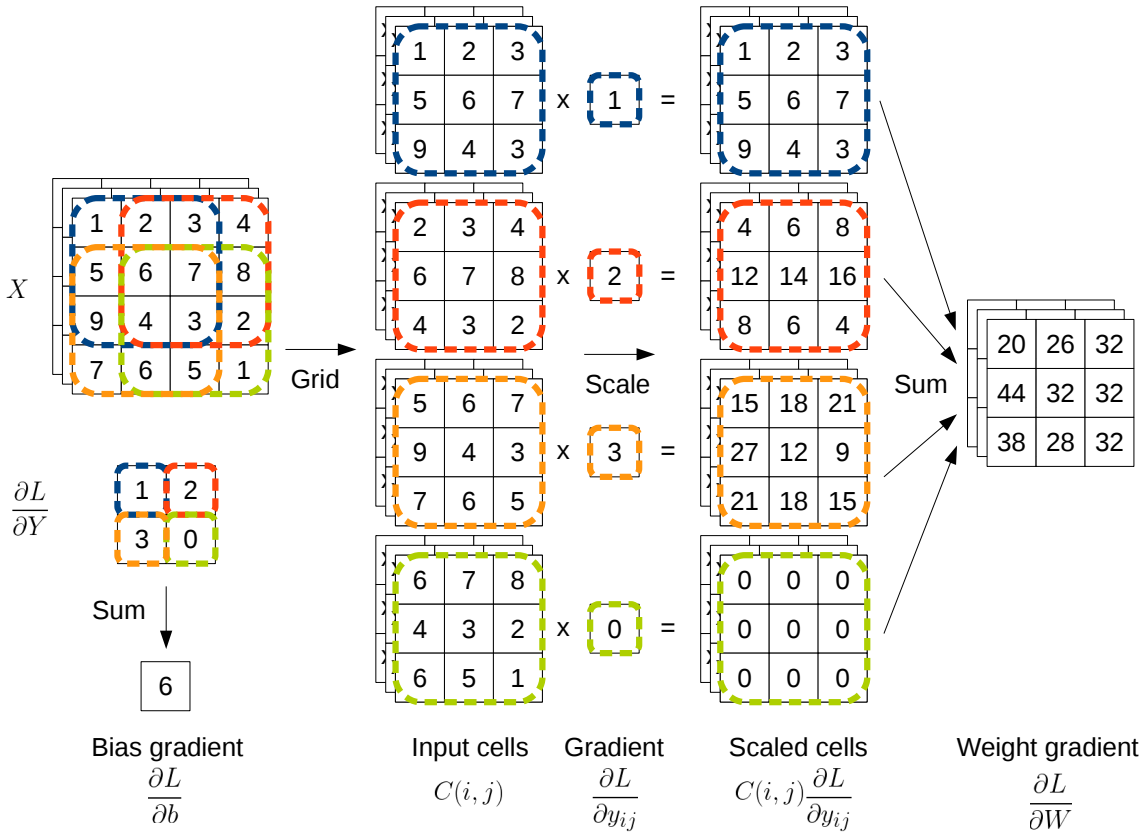



Figure B-20: **Weight and bias gradient** computation of the convolution layer for a single kernel $W$ and bias $b$. We use number values for clarity. The bias gradient is the following sum: $\frac{\partial L}{\partial b} = \sum_{i,j} \frac{\partial L}{\partial y_{ij}}$. To compute the weight gradient, we decompose the input $X$ into a grid, scale each element with the corresponding gradient coordinate $y_{ij}$ and sum the result to obtain $\frac{\partial L}{\partial W}$.

### B.5.5 Remarks

**Extension to multiple kernels.** When using multiple kernels (and their corresponding biases), each of them results in a single independent feature map, and the output becomes a three-dimensional tensor of size $n_{kernels} \times h_{out} \times w_{out}$. For backpropagation to the input, the contributions of individual kernels and their corresponding map are summed to compute the input gradient.

**Number of parameters.** A convolution layer has $n_{kernels} \times n_{features} \times k \times k$ free parameters in the kernels, and $n_{kernels}$ parameters in the biases. The number of free parameters can therefore be adjusted by choosing the number of kernels and their size.

**Implementation.** A convolution with a single kernel consists of dot-products between a kernel and many cells. This can be seen as a matrix-vector multiply operation. In practice, a convolution layer consists of multiple kernels against many cells, which corresponds to a matrix-matrix multiply operation. We illustrate this in Figure B-21.

This linear algebra operation (Single-precision General Matrix Multiply - SGEMM) is extremely optimized in computers thanks to the BLAS formalism (Lawson et al. [1979]), and can be computed by decomposing the output matrix in blocks and parallelizing computations. GPUs are designed for parallel processing, and allow running the convolution layer much quicker than on CPUs, granting the processing power necessary for convolutional neural networks research. Similar optimizations can be obtained for the backward operations. In particular the gradient computations can also be cast as convolution operations.

*Overall, convolutional neural networks greatly benefit from the capabilities of GPUs for linear algebra.*

**Mini-batch parallelization.** Using minibatches consists of executing the forward and backward operations in parallel across $B$ examples. This increases the total number of cells that are processed in the matrix multiplication described in Figure

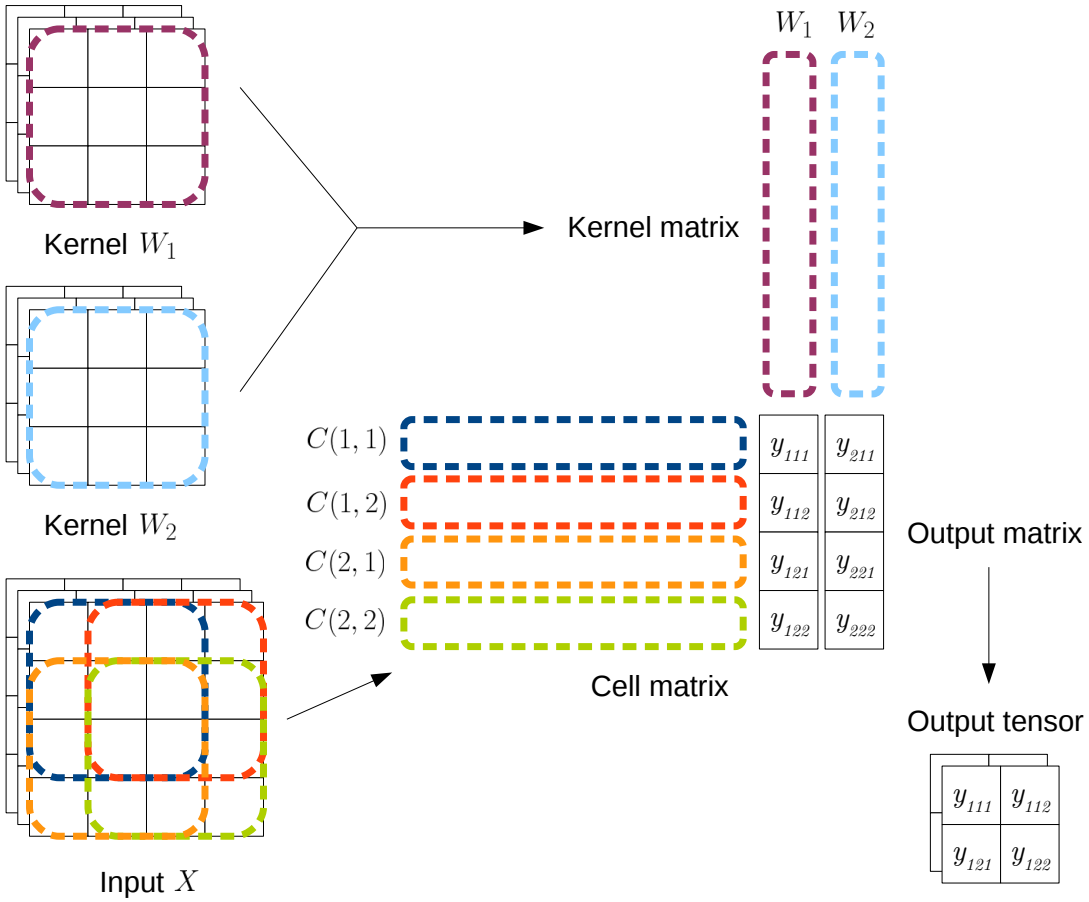



Figure B-21: Seeing the convolution operation as a matrix multiplication. The convolution layer is based on dot-products between many cells and many weighted kernels. Arranging the cells into a matrix (of size ($n_{cells}$ × cell dimensionality)), and the kernels into another matrix (of size ($n_{kernels}$ × cell dimensionality)) allows using efficient generic linear algebra routines such as the matrix multiplication. The result of this matrix multiplication is then rearranged into a tensor, exposing concatenated feature maps.

B-21. As a result, we obtain:

- a matrix multiply operation with larger matrices; SGEMM performance is closer to the hardware limits if matrices are large. More generally, parallelization is more efficient when the problem is large and made of independent subproblems (e.g. different images).

- A better estimate of the weight and bias gradient for the stochastic gradient descent learning, as it is averaged across samples.

## B.6 In practice

### B.6.1 Summary so far

**Layered architectures** We showed in Section B.2 that neural networks are arrangements of layers with specific properties within a directed acyclic graph, that allow running the backpropagation algorithm to compute gradient. This makes these functions fit in the machine learning framework that we described in Appendix A.

**Core layers.** In Sections B.3-B.4-B.5 we defined the *core* layers of neural networks, in particular the convolution, pooling, nonlinearity and fully-connected layers. Many more layers and variants exist, but the ones presented here are present in almost all successful applications. In particular, we show that the convolution layer can be easily reduced to linear algebra operations given the numerous dot-products involved. Moreover, with mini-batch processing, many samples can be treated in parallel, which translates to larger matrices at computation time. Given that GPUs are especially efficient at processing large matrices (matrix multiplication can be processed in parallel easily by dividing the problem into submatrices), this explains why their introduction in machine learning improved neural networks significantly.

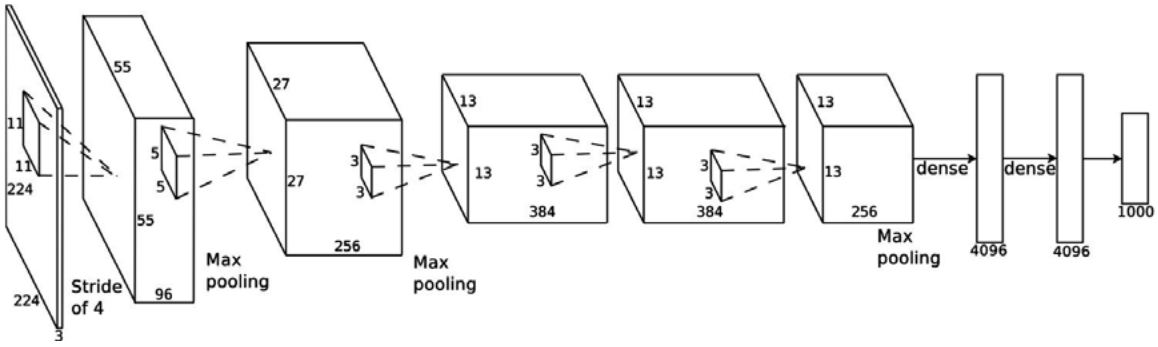



Figure B-22: Reading the AlexNet architecture (figure from Krizhevsky et al. [2012]). The blocks correspond to the data tensors processed by the network. The convolution and fully-connected (*dense*) layers are the operators between the blocks. The network contains a total of 60 million parameters.

### B.6.2 A famous architecture

**AlexNet.** We will take a look at the very important AlexNet architecture, which was proposed by Krizhevsky et al. [2012] for the ILSVRC-2012 competition, and caused the shift to neural network methods in computer vision as we described in Chapter 2. We show the architecture in Figure B-22. The sequence of layers is as follows:

| Blocks shown on figure | Data tensor size |
|---|---|
| 1. Data tensor: $224 \times 224 \times 3$ (RGB input image) | |
| (a) Convolution conv1: 96 kernels of size $11 \times 11$, stride 4 | $55 \times 55 \times 96$ |
| (b) ReLU nonlinearity | $55 \times 55 \times 96$ |
| 2. Data tensor: $55 \times 55 \times 96$ | |
| (a) Max-Pooling pool1: size 3, stride 2 | $27 \times 27 \times 96$ |
| (b) Convolution conv2: 256 kernels of size $5 \times 5$, stride 1 | $27 \times 27 \times 256$ |
| (c) ReLU nonlinearity | $27 \times 27 \times 256$ |
| 3. Data tensor: $27 \times 27 \times 256$ | |
| (a) Max-Pooling pool2: size 3, stride 2 | $13 \times 13 \times 256$ |
| (b) Convolution conv3: 384 kernels of size $3 \times 3$, stride 1 | $13 \times 13 \times 384$ |
| (c) ReLU nonlinearity | $13 \times 13 \times 384$ |
| 4. Data tensor: $13 \times 13 \times 384$ | |
| (a) Convolution conv4: 384 kernels of size $3 \times 3$, stride 1 | $13 \times 13 \times 384$ |
| (b) ReLU nonlinearity | $13 \times 13 \times 384$ |
| 5. Data tensor: $13 \times 13 \times 384$ | |
| (a) Convolution conv5: 256 kernels of size $3 \times 3$, stride 1 | $13 \times 13 \times 256$ |
| (b) ReLU nonlinearity | $13 \times 13 \times 256$ |
| 6. Data tensor: $13 \times 13 \times 256$ | |

(a) Max-Pooling pool5: size 3, stride 2 — $6 \times 6 \times 256$
(b) View as vector — 9216
(c) Fully connected FC6: $9216 \rightarrow 4096$ — 4096
(d) ReLU nonlinearity — 4096

7. Data tensor: 4096
   (a) Fully connected FC7: $4096 \rightarrow 4096$ — 4096
   (b) ReLU nonlinearity — 4096
8. Data tensor: 4096
   (a) Fully connected FC8: $4096 \rightarrow 1000$ — 1000
   (b) Softmax nonlinearity — 1000
9. Data tensor: 1000 (one score per ImageNet class)

We omit the regularization layers (local response normalization and dropout for clarity) and refer to Krizhevsky et al. [2012] for more details. We can observe in this architecture, which corresponds to a standard network in computer vision, that convolution layers are stacked on top of each other. We explain the consequence next.

**Growing receptive fields with convolution layers.** Each value in a feature map depends on a subset of the initial input image that is processed by the network. This subset is called the *receptive field*. Stacking convolution layers increases the receptive field of elements of feature maps as we progress in the network, as we show in Figure B-23. As a result, the input image to a network is first processed *locally* then more globally by the further layers. This gives convolutional neural networks a bottom-up structure that may be related to their success.

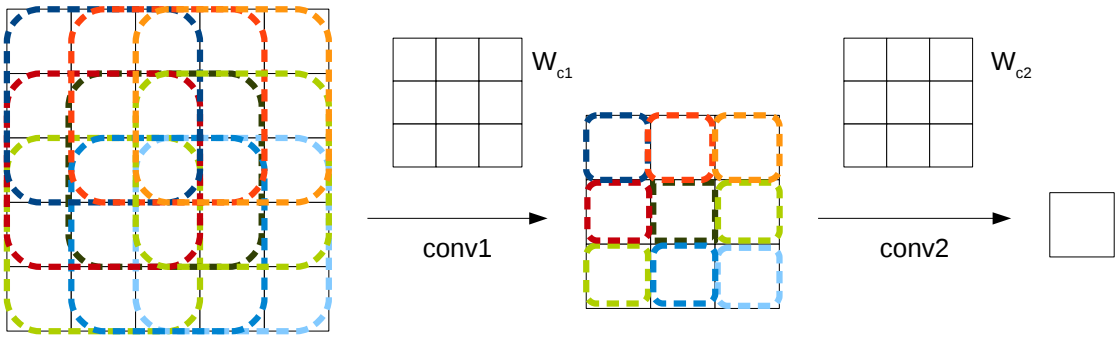



Figure B-23: Stacking two convolution layers increases the receptive field of an element of the feature map. In this example, each output element of the first layer has access to a $3 \times 3$ area, but the second layer has access to a $3 \times 3$ square of these elements, giving it access to a $5 \times 5$ area of the input image.

**Study of convolution filters by Zeiler & Fergus.** In Zeiler and Fergus [2014], the authors provide an insightful study of the role played by various filters in AlexNet, by studying what patterns in images activate filters the most in a convolutional network. We show their visualization in Figure B-24. We can observe that layers in the network activate against progressively more semantically meaningful patches, corresponding to larger receptive fields because of the stacked convolutions and poolings.

**Rules for building an architecture.** Unfortunately, there are no provably true rules for how an architecture should be designed for best performance. However, practice led to some heuristics on this difficult issue:

- There should be a nonlinearity layer after each weighted layer.
- Deeper networks (longer sequences of layers) are better.

Designing a good architecture is a long process of trial-and-error.

**The balance of computation in layers.** In an architecture such as AlexNet, processing a single image (forward) corresponds to 725 million floating-point operations (FLOPs). The architecture contains 60 million parameters. Convolution layers perform roughly 95% of the FLOPs and contain 5% of the parameters. The fully-connected layers perform 5% of the FLOPs and contain 95% of the parameters. Max-pooling and nonlinearity account for less than 1% of the total number of operations. Therefore, faster implementations of the convolution layers are critical to the success of these algorithms, and improvements have appeared over time, with the GPU convolution of Krizhevsky et al. [2012], the Fourier Transform based algorithm of Mathieu et al. [2014], the Winograd algorithm of Lavin and Gray [2016] and the optimized implementations by nVidia in cuDNN (Chetlur et al. [2014]).


Figure B-24: Original caption: *Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are not samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.* Figure and caption from Zeiler and Fergus [2014].

### B.6.3 Training a network

Most of the advice mentioned here comes from the work of LeCun et al. [1998b], as these algorithms have been studied for a long time in the 90s, but this knowledge was refined during the years.

The major issue when training a deep multi-layered network is that the signal can vanish. In the forward phase, if the weights in layers are small, then the magnitudes of the output values decrease; with many layers, this has a multiplicative effect, and the signal can vanish entirely. Conversely, if the weights are large, then the values grow fast, and this can lead to numerical instabilities with unbounded activations such as the ReLU.

In the backward phase, in the case of saturating nonlinearities, the gradient is multiplied by the slope: if nonlinearities are mostly saturated, then this gradient can become too small to allow learning. Therefore, a couple settings were proposed.

**Weight initialization.** LeCun et al. [1998b] propose that data should first be *normalized*. The mean values of variables should be close to zero, their variances should be about the same (in general, 1), and in the best case they should be decorrelated.

With normalized inputs, data can be treated as a random variable with known variance. In neural networks, weights are initialized randomly; in general by following a centered gaussian or uniform distribution with variance $\sigma^2$. This allows choosing the magnitudes of the weights (through $\sigma$) and therefore controlling the variance of the output of the a layer. Given a normalized centered and scaled input, the weights for a given neuron (e.g. for a convolution, its kernel parameters) should be drawn from a centered distribution with $\sigma = m^{-1/2}$ where m is the dimensionality of the kernel (the number of input variables to the dot-product or *fan-in*), if the desired output variance is 1.

This rule was refined over the years, notably with the formula proposed by Glorot and Bengio [2010], known as the "Xavier" initialization, drawing weights from a uniform distribution with variance $\frac{2}{n_i + n_{i+1}}$ where $n_i$ corresponds to the number of input variables (*fan-in*) for layer $i$. This ensures that the variances of the forward and

backward signal remain approximately the same throughout the network with hyperbolic tangent activations. As a result, the neurons activate in areas where the slope of the saturating nonlinearity is high (the *linear regime*, corresponding to the central part of the plots in Figure B-9 (top and middle)). Current frameworks implement such initialization schemes natively.

**Batch normalization.** The *batch normalization* layer was proposed in Ioffe and Szegedy [2015] and alleviates issues related to these weight scaling issues. When a mini-batch of inputs is processed, the statistics of activations are computed across the mini-batch and the data representation is centered on-the-fly (zero-mean, unit-variance across the mini-batch) before the nonlinearity, within a differentiable procedure. In the saturating cases, this ensures that activations lie mostly on the linear regime, avoiding vanishing and exploding signals in the forward and backward computations. In all cases, authors report faster, easier and better training; this technique is very popular as of 2017.

The effect of this layer on the optimization procedure is not perfectly understood yet, but some insights were provided in Lafond et al. [2017].

**Learning rates.** When running stochastic gradient descent for optimizing the values of the weights, picking a good *learning rate* can become an issue. This is pretty much an unsolved problem in the difficult non-convex case of neural networks (although this issue was studied in LeCun et al. [1998b]), and practice suggests that values should be tried and cross-validated.

One commonly used practice, used in Krizhevsky et al. [2012], consists of setting a global learning rate (initially $10^{-2}$) on all the weight parameters, then decreasing it by a factor of 10 every time the validation loss reaches a plateau. We show a sketch of typical loss curves in Figure B-25.

**Momentum.** An alternative update rule for SGD (described in A.5) that is frequently used (in particular by Krizhevsky et al. [2012]) is the momentum acceleration method of Polyak [1964] (also known as *heavy ball method*) or its variant by Nesterov

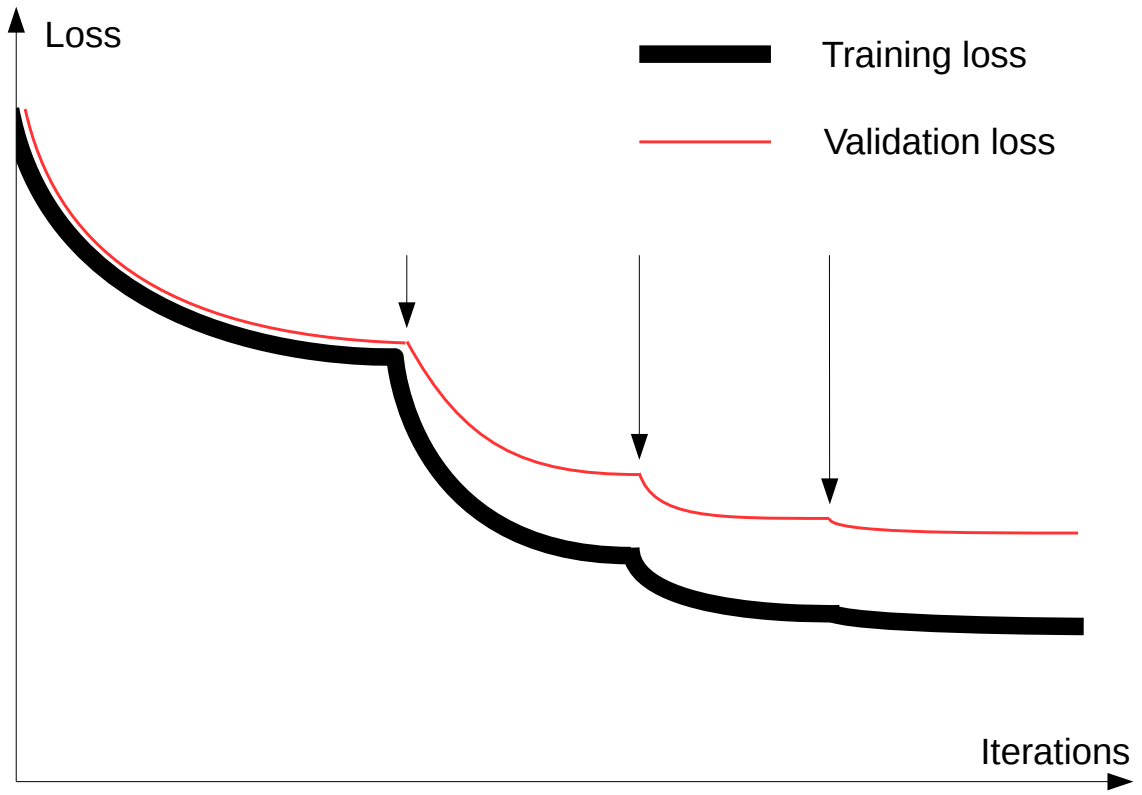



Figure B-25: Illustration of typical loss curves for an AlexNet ImageNet training. The training and validation losses decrease over iterations, and when they reach a plateau the learning rate (initially $10^{-2}$) is manually decreased by a factor 10 (pointed with arrows), until the procedure converges.

[1983]. Both versions enjoy improved convergence rates compared to the plain version of SGD that we will not detail here; Sutskever et al. [2013] also report the importance of these methods for training in practice.

Let $C(w)$ be a functional that we seek to minimize. The update rule for the momentum method is:

$$v_{t+1} = \mu v_t - \epsilon \nabla C(w_t), \tag{B.24}$$

$$w_{t+1} = w_t + v_{t+1}, \tag{B.25}$$

where $\mu \in [0, 1]$ is the momentum coefficient and $\epsilon > 0$ is the learning rate.

The update rule for Nesterov's Accelerated Gradient method is:

$$v_{t+1} = \mu v_t - \epsilon \nabla C(w_t + \mu v_t), \tag{B.26}$$

$$w_{t+1} = w_t + v_{t+1}, \tag{B.27}$$

where $\mu \in [0, 1]$ is the momentum coefficient and $\epsilon > 0$ is the learning rate.

The intuition, in both cases, is that the weights are updated according to an exponential average of previous weights updates accumulated in a momentum vector. In the version of Polyak, the new gradient is computed before the momentum is applied, in contrast to Nesterov's method that computes it after.

**Adaptive optimizers.** For the issue of learning rates, adaptive learning-rate optimizers such as RMSprop (Tieleman and Hinton [2012]), AdaGrad (Duchi et al. [2011a], used in Chapter 4) or Adam (Kingma and Ba [2015b], used in Chapter 5), were proposed to avoid setting learning rates manually. The general intuition behind these adaptive algorithms is to scale the step size by the inverse of a running average of the gradient magnitude: when the gradient is small the step sizes are larger, and vice-versa.

Again, the effect of these methods on the optimization procedure is not understood as well as SGD yet, but their ease of use have popularized their usage.

**Data augmentation.** A popular regularization method for reducing overfitting in neural networks is data augmentation: this consists of applying a label-preserving transformation to input samples in order to artificially increase the size of the dataset. An exemple used notably in Krizhevsky et al. [2012] is the crop/flip scheme, that extracts patches from an input image by cropping the borders and horizontally flipping them with probability 0.5. We illustrate this scheme in Figure B-26. An experiment in Zhang et al. [2017] shows that without data augmentation, it is possible to train a CNN on the ImageNet dataset (1 million images) with random labels, showing the very high capacity of these algorithms for memorizing a training set, and hinting at the usefulness of the data augmentation scheme.

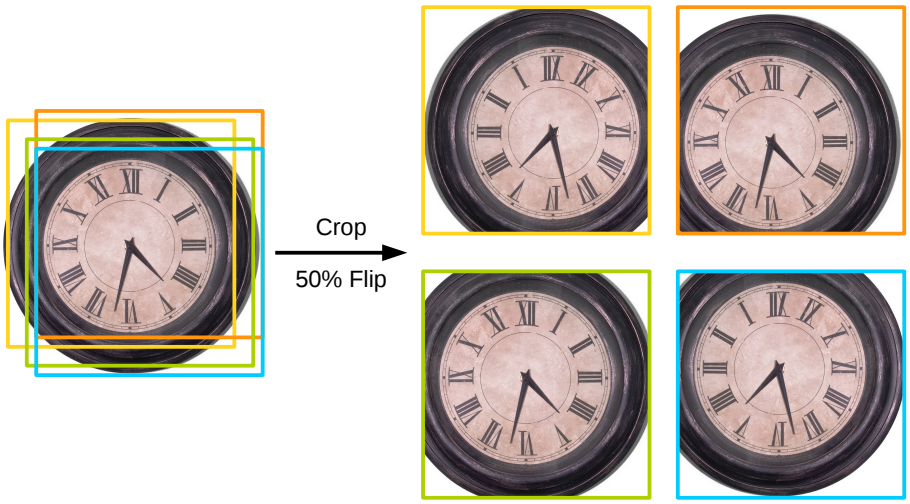



Figure B-26: Data augmentation. An input image undergoes a label-preserving transformation: we crop pixels in the border of the image and flip the result horizontally with 50% probability during trianing, creating more examples for the neural network to learn. This technique does not correspond to an explicit regularization term in the objective function but reduces overfitting.

**Recurrent neural networks.** These architectures are frequently used in machine translation (e.g. in the Google Neural Machine Translation system, Wu et al. [2016]) with their Long-Short Term Memory variant (LSTM, Hochreiter and Schmidhuber [1997]) mentioned in Chapter 2. These networks are useful in these cases because they are able to process sequences as input, and provide sequences as output.

Recurrent Neural Networks contain a hidden state that is updated with each element of the input sequence, and this state is used to produce elements in the output following, for instance, equations of the form:

$$h_0 = 0, \tag{B.28}$$

$$h_{i+1} = f_h(x_{i+1}, h_i, w_h), \tag{B.29}$$

$$y_i = f_{out}(h_i, w_{out}) \tag{B.30}$$

where $w_h$ and $w_{out}$ are the parameters of the recurrent unit, $(h_i)_i$ is a sequence of hidden states, $(x_i)_i$ is an input sequence, and $(y_i)_i$ is an output sequence. We show in Figure B-27 an illustration of a recurrent unit.

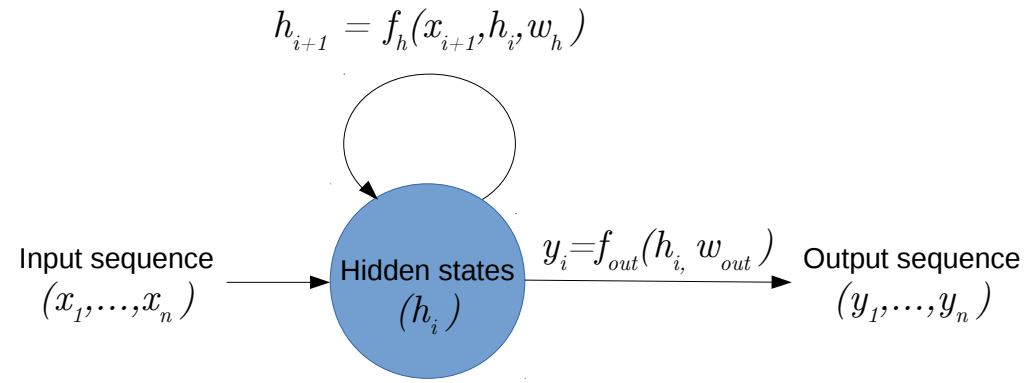



Figure B-27: A recurrent unit processing an input sequence $(x_i)_i$, updating a hidden state sequence $(h_i)_i$ with a recurrent connection, producing an output sequence $(y_i)_i$.

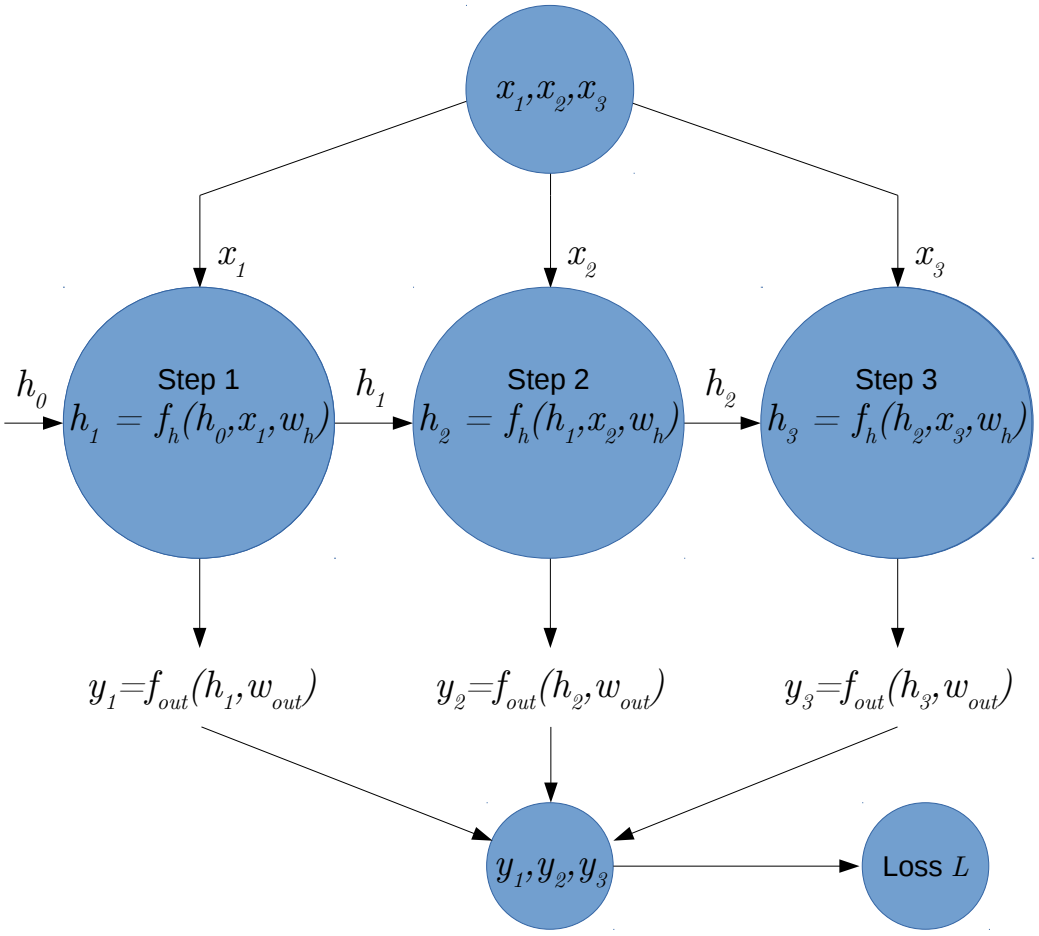



Figure B-28: The same recurrent unit as in Figure B-27, unrolled for 3 steps. The backpropagation through time training algorithm for recurrent networks consists of unrolling the network and converting it to a Directed Acyclic Graph in order to use the standard backpropagation algorithm described in Section B.2.

But the training algorithm for these recurrent networks, known as *backpropagation through time* (BPTT, Rumelhart et al. [1986]), consists of unrolling the network for a fixed number of iterations, in order to build a Directed Acyclic Graph and execute the usual backpropagation algorithm for feedforward networks, except the parameters $w_h$ and $w_{out}$ are shared between unrolled nodes (this doesn't contradict the directed acyclic assumption); the gradient for the recurrent unit is the sum of the contributions of all nodes. We show a sketch of the same recurrent unit in Figure B-28 after unrolling it for 3 steps to give intuition on the procedure.

Even though RNNs appear to be a particular case of the standard neural networks, the training techniques and corresponding issues are different: when unrolling for a large number of steps (hundreds or thousands), the vanishing and exploding gradient problems can be more difficult to deal with due to the multiplicative effects, but we leave these out of the scope of this text.

## B.7 Summary and conclusion

**Summary.** In this Appendix B, we show that (convolutional) neural networks are a particular set of parameterized functions that fit in the machine learning framework described in Appendix A. In Section B.2 we described the properties of the architectures that define these functions, as layers laid on a directed acyclic graph, enabling the use of the backpropagation algorithm. We also showed which operations each layer needs to run, for the procedure to succeed.

In Sections B.3-B.4-B.5, we described a set of important layers that are present in most neural network architectures, in particular the convolution, pooling, fully-connected and nonlinearity layers, as well as methods for computing the appropriate gradients in the backward operations. In particular, we show that the numerous dot-products involved in computations can be seen as linear algebra operations on matrices, for which efficient algorithms are available.

In Section B.6 we focused on practice, and presented the network of Krizhevsky et al. [2012] and, the study of its convolution filters provided in Zeiler and Fergus

[2014]. We also mentioned some of the issues related to weights initialization (alleviated with the recent batch-normalization procedure of Ioffe and Szegedy [2015]) and issues related to setting learning rates in the stochastic gradient descent, prompting the usage of adaptive optimizers. Many of the tricks and hacks used for improving the training of neural networks are reported to work well in particular research cases, but the general effectiveness of these methods is not proved by any theory yet, and developing the right intuitions to build high-performance setups is what makes these methods difficult: it is a matter of practice, trial-and-error analysis, and patience. The introduction of LeCun et al. [1998b] describes this situation in a paragraph that is still true today: *"Backpropagation is a very popular neural network learning algorithm because it is conceptually simple, computationally efficient, and because it often works. However, getting it to work well, and sometimes to work at all, can seem more of an art than a science. Designing and training a network using backprop requires making many seemingly arbitrary choices such as the number and types of nodes, layers, learning rates, training and test sets, and so forth. These choices can be critical, yet there is no foolproof recipe for deciding them because they are largely problem and data dependant. However, there are heuristics and some underlying theory that can help guide a practitioner to make better choices."*

**Conclusion.** In Appendix A, which is the first part of the technical background of this thesis, we showed that the basic building block of machine learning, classification, can be reduced to an optimization problem on a set of parameterized functions, that can be addressed using first-order gradient descent methods.

In this Appendix B, which is the second part, we made an effort to break neural networks down to smaller pieces, pointing out that these methods are conceptually simple, but their practice more difficult due to the high number of hyper-parameters (e.g. number of layers, layer sizes, learning rates) at training time. Our study is of course not exhaustive, but our goal is merely to reveal the true nature of these neural networks: complex and powerful algorithms built from (mostly) simple linear algebra.

# Appendix C

# Results of Evaluation of Generative Adversarial Networks with Classifier Two-Sample Tests

**Architectures and filter parameters.** We detail here the architectures of the generator and discriminator, that depend on the parameters gf and df.

DC-GAN generator architecture with parameter gf:

1. Noise input: $1 \times 1 \times 100$
   (a) Deconvolution dc1: (8×gf) kernels of size $4 \times 4$, stride 1
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
2. State dimensionality: $4 \times 4 \times (8 \times \text{gf})$
   (a) Deconvolution dc2: (4×gf) kernels of size $4 \times 4$, stride 2
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
3. State dimensionality: $8 \times 8 \times (4 \times \text{gf})$
   (a) Deconvolution dc3: (2×gf) kernels of size $4 \times 4$, stride 2
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
4. State dimensionality: $16 \times 16 \times (2 \times \text{gf})$
   (a) Deconvolution dc4: (gf) kernels of size $4 \times 4$, stride 2
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
5. State dimensionality: $32 \times 32 \times \text{gf}$
   (a) Deconvolution dc5: 3 kernels of size $4 \times 4$, stride 2
   (b) Hyperbolic Tangent nonlinearity

6. Output RGB image tensor: $64 \times 64 \times 3$

DC-GAN discriminator architecture with parameter df:

1. Image input: $64 \times 64 \times 3$
   (a) Convolution c1: (df) kernels of size $4 \times 4$, stride 1
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
2. State dimensionality: $32 \times 32 \times$ df
   (a) Convolution c2: (2×df) kernels of size $4 \times 4$, stride 2
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
3. State dimensionality: $16 \times 16 \times (2 \times$ df)
   (a) Convolution c3: (4×df) kernels of size $4 \times 4$, stride 2
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
4. State dimensionality: $8 \times 8 \times (4 \times$ df)
   (a) Convolution c4: (8×df) kernels of size $4 \times 4$, stride 2
   (b) Spatial Batch Normalization
   (c) ReLU non-linearity
5. State dimensionality: $4 \times 4 \times (8 \times$ df)
   (a) Convolution dc5: 1 kernel of size $4 \times 4$
   (b) Sigmoid nonlinearity
6. Output score: 1 scalar value

In the following figures, we show random samples from generators trained with the GAN procedure using the code of Radford et al. [2016], varying the filter parameters gf and df. We display samples at different epochs (ep) of training. Table C.1 contains the results for the LSUN-Bedrooms dataset (Yu et al. [2015]), and Table C.2 for the Labeled Faces in the Wild (LFW, Huang et al. [2007]) dataset.

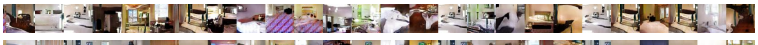
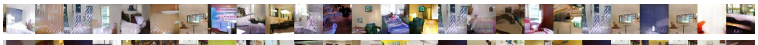
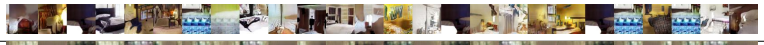
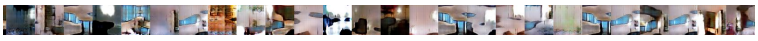
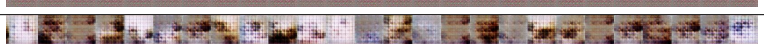
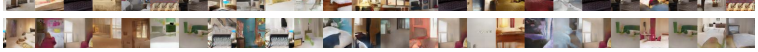
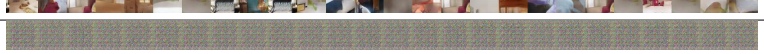
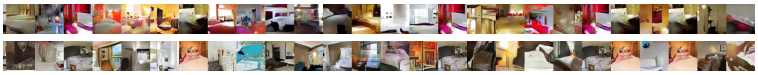
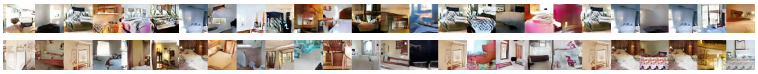
| gf | df | ep | random sample | MMD | KNN | NN |
|---|---|---|---|---|---|---|
| - | - | - |  | - | - | - |
| 32 | 32 | 1 |  | 0.154 | 0.994 | 1.000 |
| 32 | 32 | 10 |  | 0.024 | 0.831 | 0.996 |
| 32 | 32 | 50 |  | 0.026 | 0.758 | 0.983 |
| 32 | 32 | 100 |  | 0.014 | 0.797 | 0.974 |
| 32 | 32 | 200 |  | **0.012** | 0.798 | 0.964 |
| 32 | 64 | 1 |  | 0.330 | 0.984 | 1.000 |
| 32 | 64 | 10 |  | 0.035 | 0.897 | 0.997 |
| 32 | 64 | 50 |  | 0.020 | 0.804 | 0.989 |
| 32 | 64 | 100 |  | 0.032 | 0.936 | 0.998 |
| 32 | 64 | 200 |  | 0.048 | 0.962 | 1.000 |
| 32 | 96 | 1 |  | 0.915 | 0.997 | 1.000 |
| 32 | 96 | 10 |  | 0.927 | 0.991 | 1.000 |
| 32 | 96 | 50 |  | 0.924 | 0.991 | 1.000 |
| 32 | 96 | 100 |  | 0.928 | 0.991 | 1.000 |
| 32 | 96 | 200 |  | 0.928 | 0.991 | 1.000 |
| 64 | 32 | 1 |  | 0.389 | 0.987 | 1.000 |
| 64 | 32 | 10 |  | 0.023 | 0.842 | 0.979 |
| 64 | 32 | 50 |  | 0.018 | 0.788 | 0.977 |
| 64 | 32 | 100 |  | 0.017 | 0.753 | 0.959 |
| 64 | 32 | 200 |  | 0.018 | 0.736 | 0.963 |
| 64 | 64 | 1 |  | 0.313 | 0.964 | 1.000 |
| 64 | 64 | 10 |  | 0.021 | 0.825 | 0.988 |
| 64 | 64 | 50 |  | 0.014 | 0.864 | 0.978 |
| 64 | 64 | 100 |  | 0.019 | 0.685 | 0.978 |
| 64 | 64 | 200 |  | 0.021 | 0.775 | 0.980 |
| 64 | 96 | 1 |  | 0.891 | 0.996 | 1.000 |
| 64 | 96 | 10 |  | 0.158 | 0.830 | 0.999 |
| 64 | 96 | 50 |  | 0.015 | 0.801 | 0.980 |
| 64 | 96 | 100 |  | 0.016 | 0.866 | 0.976 |
| 64 | 96 | 200 |  | 0.020 | 0.755 | 0.983 |
| 96 | 32 | 1 |  | 0.356 | 0.986 | 1.000 |
| 96 | 32 | 10 |  | 0.022 | 0.770 | 0.991 |
| 96 | 32 | 50 |  | 0.024 | 0.748 | **0.949** |
| 96 | 32 | 100 |  | 0.022 | 0.745 | 0.965 |
| 96 | 32 | 200 |  | 0.024 | 0.689 | 0.981 |
| 96 | 64 | 1 |  | 0.287 | 0.978 | 1.000 |
| 96 | 64 | 10 |  | **0.012** | 0.825 | **0.966** |
| 96 | 64 | 50 |  | 0.017 | 0.812 | 0.962 |
| 96 | 64 | 100 |  | 0.019 | **0.670** | 0.983 |
| 96 | 64 | 200 |  | 0.020 | 0.711 | 0.972 |
| 96 | 96 | 1 |  | 0.672 | 0.999 | 1.000 |
| 96 | 96 | 10 |  | 0.671 | 0.999 | 1.000 |
| 96 | 96 | 50 |  | 0.829 | 0.999 | 1.000 |
| 96 | 96 | 100 |  | 0.668 | 0.999 | 1.000 |
| 96 | 96 | 200 |  | 0.849 | 0.999 | 1.000 |

Table C.1: GAN evaluation results on the LSUN dataset, for different epochs (ep), and numbers of filters (gf, df). Different models are separated by a horizontal line. We show different test statistics (for MMD, C2ST-KNN, C2ST-NN) with the lowest statistics in bold; a lower test statistic estimates that the GAN produces better samples. Best viewed with zoom.

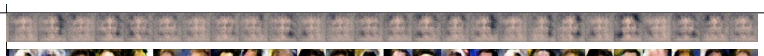
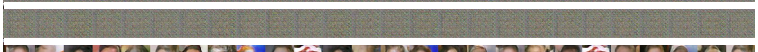
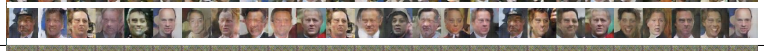
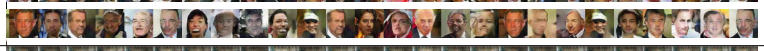
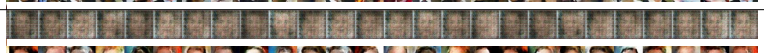
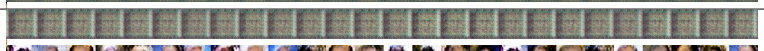
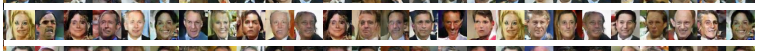
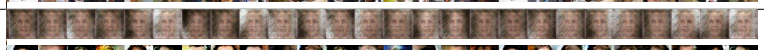
| gf | df | ep | random sample | MMD | KNN | NN |
|---|---|---|---|---|---|---|
| - | - | - | | - | - | - |
| 32 | 32 | 1 |  | 0.806 | 1.000 | 1.000 |
| 32 | 32 | 10 |  | 0.152 | 0.940 | 1.000 |
| 32 | 32 | 50 |  | 0.042 | 0.788 | 0.993 |
| 32 | 32 | 100 |  | 0.029 | 0.808 | 0.982 |
| 32 | 32 | 200 |  | 0.022 | 0.776 | 0.970 |
| 32 | 64 | 1 |  | 0.994 | 1.000 | 1.000 |
| 32 | 64 | 10 |  | 0.989 | 1.000 | 1.000 |
| 32 | 64 | 50 |  | 0.050 | 0.808 | 0.985 |
| 32 | 64 | 100 |  | 0.036 | 0.766 | 0.972 |
| 32 | 64 | 200 |  | **0.015** | 0.817 | 0.987 |
| 32 | 96 | 1 |  | 0.995 | 1.000 | 1.000 |
| 32 | 96 | 10 |  | 0.992 | 1.000 | 1.000 |
| 32 | 96 | 50 |  | 0.995 | 1.000 | 1.000 |
| 32 | 96 | 100 |  | 0.053 | 0.778 | 0.987 |
| 64 | 96 | 200 |  | 0.037 | 0.779 | 0.995 |
| 64 | 32 | 1 |  | 1.041 | 1.000 | 1.000 |
| 64 | 32 | 10 |  | 0.086 | 0.971 | 1.000 |
| 64 | 32 | 50 |  | 0.043 | 0.756 | 0.988 |
| 64 | 32 | 100 |  | 0.018 | 0.746 | 0.973 |
| 64 | 32 | 200 |  | 0.025 | 0.757 | 0.972 |
| 64 | 64 | 1 |  | 0.836 | 1.000 | 1.000 |
| 64 | 64 | 10 |  | 0.103 | 0.910 | 0.998 |
| 64 | 64 | 50 |  | 0.018 | 0.712 | 0.973 |
| 64 | 64 | 100 |  | 0.020 | 0.784 | **0.950** |
| 64 | 64 | 200 |  | 0.022 | 0.719 | 0.974 |
| 64 | 96 | 1 |  | 1.003 | 1.000 | 1.000 |
| 64 | 96 | 10 |  | 1.015 | 1.000 | 1.000 |
| 64 | 96 | 50 |  | 1.002 | 1.000 | 1.000 |
| 64 | 96 | 100 |  | 1.063 | 1.000 | 1.000 |
| 64 | 96 | 200 |  | 1.061 | 1.000 | 1.000 |
| 96 | 32 | 1 |  | 1.022 | 1.000 | 1.000 |
| 96 | 32 | 10 |  | 0.222 | 0.978 | 1.000 |
| 96 | 32 | 50 |  | 0.026 | 0.734 | 0.965 |
| 96 | 32 | 100 |  | 0.016 | 0.735 | 0.964 |
| 96 | 32 | 200 |  | 0.021 | 0.780 | 0.973 |
| 96 | 64 | 1 |  | 0.715 | 1.000 | 1.000 |
| 96 | 64 | 10 |  | 0.042 | 0.904 | 0.999 |
| 96 | 64 | 50 |  | 0.024 | **0.697** | 0.971 |
| 96 | 64 | 100 |  | 0.028 | 0.744 | 0.983 |
| 96 | 64 | 200 |  | 0.020 | 0.697 | 0.976 |
| 96 | 96 | 1 |  | 0.969 | 1.000 | 1.000 |
| 96 | 96 | 10 |  | 0.920 | 1.000 | 1.000 |
| 96 | 96 | 50 |  | 0.926 | 1.000 | 1.000 |
| 96 | 96 | 100 |  | 0.920 | 1.000 | 1.000 |
| 96 | 96 | 200 |  | 0.923 | 1.000 | 1.000 |

Table C.2: GAN evaluation results on the LSUN dataset, for different epochs (ep), and numbers of filters (gf, df). Different models are separated by a horizontal line. We show different test statistics (for MMD, C2ST-KNN, C2ST-NN) with the lowest statistics in bold; a lower test statistic estimates that the GAN produces better samples. Best viewed with zoom.

# Appendix D

# Toy experiment with distances

In this appendix, we build a controlled setup and observe the behavior of different distance measures on distributions that we compute numerically on a simple problem, in order to understand these distances better.

## D.1 Experiment

Here we investigate the Kullback-Leibler Divergence (KL), Jensen-Shannon Divergence (JS) and Earth Mover Distance (EMD), defined in 5.5.1 in the main text. In order to understand better the differences and issues related to the distances, we design a simple data-fitting problem inspired by Independant Component Analysis (ICA, Hyvarinen et al. [2001]) and observe the assumptions required as well as the results obtained with KL, JS and EMD.

We describe a simple case of ICA where we have access to samples $(u, v) = T(x, y)$, where $x$ and $y$ are independent random variables following some distributions. The goal is to find $T^{-1}$ and thus retrieve the $(x, y)$ distribution. In our experiments, $T$ corresponds to a rotation transformation. ICA is useful, for example, for blind source separation.

#### Data and model

We describe the data and model that we use for this ICA-inspired experiment below.

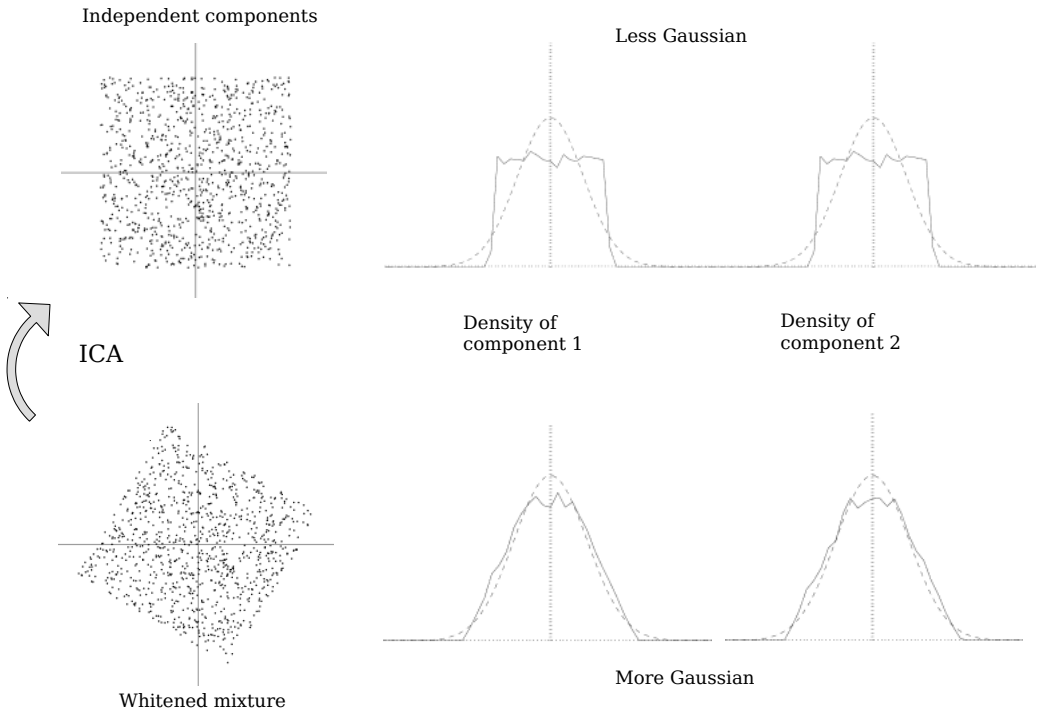



Figure D-1: The central limit theorem says that sums of independent variables tend to be gaussian. One way of finding the independent components is to find a direction such that the sample distribution is maximally non-gaussian. The graphs are taken from the ICA book (Hyvarinen et al. [2001]).

**Data generation.** We generate two-dimensional data following the density shown in Figure D-2:

$$p(x, y) \propto (1 - x^2)_+ \times (1 - y^2)_+.$$

In order to compute distances between densities, we estimate the density of our data using a Parzen Window approach with Epanechnikov kernel: around each point that we sample from the original distribution (see Figure D-3), we add a finite circular parabola-shaped blob of density with radius 0.3. We avoid the Gaussian kernel, as the KL divergence enforces solutions that cover the whole support of the data density as we will see in the experiments (KL has infinite value otherwise). This procedure adds *sampling noise* to our data, to account for the fact that datasets are made of discrete samples of the *real data* distribution, as shown in Figure D-4.

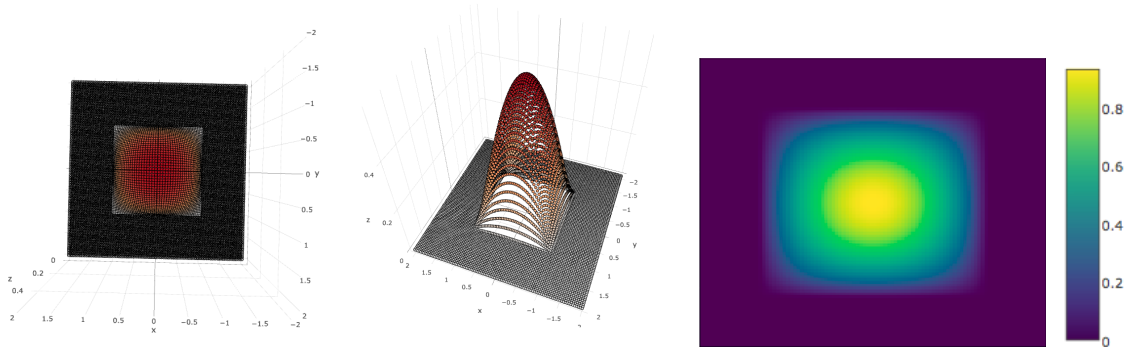



Figure D-2: Our two-dimensional parabola-based data distribution. Left: 3D plot seen from above, middle: from the side, right: heatmap and colormap.

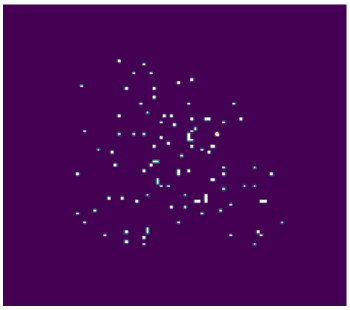


Figure D-3: Points sampled from the parabola distribution shown above.

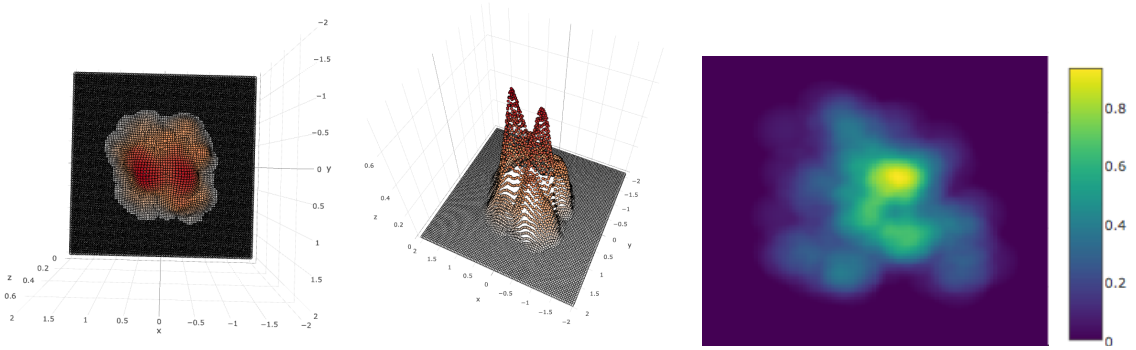



Figure D-4: Our data distribution with sampling noise. Left: 3D plot seen from above, middle: from the side, right: heatmap and colormap.

**Variable-size rotated rectangle model.** We fit the following model to our data:

$$q_{\theta,a,b}(x,y) = R_{\theta,a,b}(x,y)$$

$R_{\theta,a,b}$ is a uniform probability distribution rectangle of size $a \times b$ rotated by an angle $\theta$, summing to one, shown in Figure D-5.

While the goal is to retrieve the rotation transform that was applied to the data,

we opt for a simpler equivalent case where we do not rotate the data, but allow rotation in the model instead. This corresponds to a change of coordinates in the problem, and provides a simpler set of target solutions $\theta^* = 0 \left[\frac{\pi}{2}\right]$.

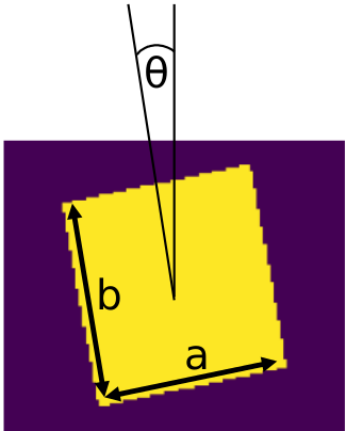



Figure D-5: Our model in the experiment. This model corresponds to a uniform probability distribution with the shape of a $\theta$-rotated rectangle of size $a \times b$. The edges are jagged due to the discretization of the space for computations.

**What we solve.** In this ICA-inspired problem, the goal is to retrieve the rotation angle $\theta$ that corresponds to the best fit.

In order to use the KL and JS divergences that require density overlap, we use density estimation using a Parzen Window estimator as described above. To add this circular blob of noise, we do an important assumption: the $x-$ and $y-$ directions are equivalent in this problem, implicitly defining a metric over the 2D space through the addition of isotropic noise.

In the case of EMD, we do the same assumption but explicitly, and define the *ground metric* (the distance between pairs of samples) as the Euclidean distance over $\mathbb{R}^2$. We discuss this assumption after the experimental results.

## D.2 Implementation details

**Overlap-based metrics (KL and JS).** For our experiments, we restrict ourselves to a low-dimensional setup, where we can compute KL and JS using numerical integration with Simpson's method (explained in Press et al. [1992]), as numerical

integration suffers from the curse of dimensionality.

**Sinkhorn distances.** The Sinkhorn Algorithm, described in Cuturi [2013], Schmitzer [2016], Chizat et al. [2017], lets us compute an "entropy-regularized" version of the EMD in an efficient way. As the regularization parameter becomes smaller, the Sinkhorn distance should converge to the EMD. We use code from the Python Optimal Transport library[1] that we ported to Torch.

The example shown in Figure 5-17 (main text) computes optimal transport from a grid of dimensionality $32 \times 32 = 1024$ to another grid of dimensionality 1024. Even though we use a GPU-based implementation of Sinkhorn Distances, computation requires several seconds in this low-resolution case. Optimal transport is computationally heavy, but in the GAN scenario, using the Kantorovitch duality and optimizing through 1-Lipschitz functions appears tractable as shown by Arjovsky et al. [2017], explaining our interest in this method.

In our experiments, computations are done on discrete grids of size $128 \times 128$ representing the square $[-2, 2] \times [-2, 2] \subset \mathbb{R}^2$, in order to deal with computational constraints in EMD computation.

**Finding the solution.** Our model has few parameters, making it possible to perform a grid search over parameters to find the solution. The goal is to find the best rotation angle $\theta$. As described previously, without loss of generality, we use $\theta^* = 0$ for simplicity when generating the data. We observe the tradeoffs due to the different distances being optimized next.

## D.3 Results

The obtained solutions, shown in Figures D-6 and D-7, demonstrate that KL and Maximum Likelihood will stretch the rectangle to cover all the data distribution, because the penalty for having a data point not covered by the model is infinite.

[1] https://github.com/rflamary/POT

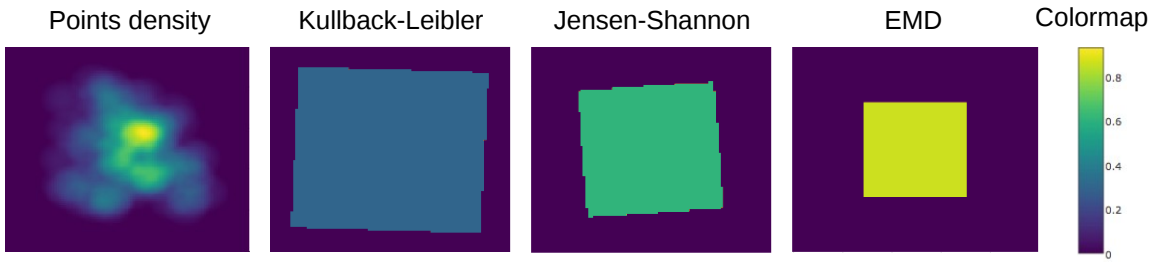



Figure D-6: We fit the uniform rectangle to our data by minimizing the KL, JS and EMD distances. EMD is computed using Euclidean distance on the 2D plane.

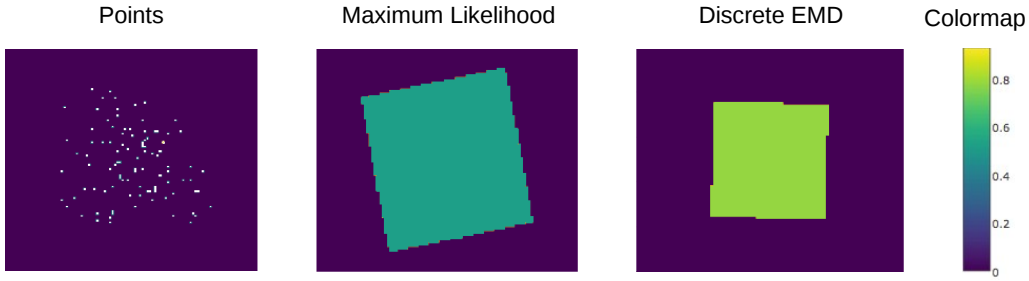



Figure D-7: We fit the uniform rectangle to our data by optimizing the Maximum Likelihood of the points under the rectangle model, and the EMD distance. EMD is computed using Euclidean distance on the 2D plane.

The JS divergence is more tolerant and will drop some samples and shrink the rectangle, thus increasing its density (it sums to 1 in all cases). The tradeoff in JS consists of balancing the penalty for dropping a sample vs. the penalty for having a smaller density within the rectangle. This tradeoff defines an implicit regularization, and solutions with overlapping areas that have better density estimates are preferred, in contrast to KL which will attempt to create as much overlap as possible between the data and the model.

With EMD, there is now a notion of distance between points of the space, and overlap is not taken into account anymore: the penalty for ignoring samples is now proportional to the ground distance. Our experiments suggest that EMD captures the general shape of the data better (as the rotation angle $\theta$ for the rectangle model is closer to 0). This behavior is controlled by the ground distance function (here Euclidean), providing an additional parameter to control the solution that is found.

Our observations are similar to those of Theis et al. [2016] as they show different solutions obtained by optimizing different objectives.

# Bibliography


http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2012/, 2012. 102

Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 37–45, 2015. 77

A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *ECCV*, 2008. 59

Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. What is an object? In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 73–80. IEEE, 2010. 41, 42, 75

Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2189–2202, 2012. 41

R. Arandjelović and A. Zisserman. Look, listen and learn. In *ICCV*, 2017. 70

Pablo Arbeláez, Jordi Pont-Tuset, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping. In *CVPR*, pages 328–335, 2014. 65

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *To appear in ICML*, 2017. 151, 163, 166, 170, 173, 174, 175, 179, 180, 261

Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *To appear in ICML*, 2017. 169, 170, 171, 173, 174, 179, 180

Y. Aytar and A. Zisserman. Tabula rasa: Model transfer for object category detection. In *ICCV*, 2011. 59

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on neural networks*, 5(2):157–166, 1994. 49, 54

Hakan Bilen and Andrea Vedaldi. Weakly supervised deep detection networks. In *CVPR*, pages 2846–2854, 2016. 76, 130

M. Blaschko, A. Vedaldi, and A. Zisserman. Simultaneous object detection and ranking with weak supervision. In *NIPS*, 2010. 74

P. Bojanowski and A. Joulin. Unsupervised Learning by Predicting Noise. *ArXiv e-prints*, Apr 2017. 179

L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL http://leon.bottou.org/papers/bottou-98x. revised, oct 2012. 47, 203, 205, 206, 207

L. Bottou. From machine learning to machine reasoning. Technical report, arXiv:1102.1808, February 2011. URL http://leon.bottou.org/papers/tr-bottou-2011. 137

L.-Y. Bottou and Y. LeCun. Sn: A simulator for connectionist models. In *Proceedings of NeuroNimes 88*, Nimes, France, 1988. 45

Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2. URL http://leon.bottou.org/papers/bottou-91c. 47, 207

Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*, 2010. 58

T. Brox, L. Bourdev, S. Maji, and J. Malik. Object segmentation by alignment of poselet activations to image contours. In *CVPR*, 2011. 72

J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE PAMI*, 35 (8):1872–1886, 2013. 88, 179

J Brian Burns, Richard S. Weiss, and Edward M Riseman. View variation of point-set and line-segment features. *IEEE PAMI*, 15(1):51–68, 1993. 29

Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002. 60

John Canny. A computational approach to edge detection. *IEEE PAMI*, (6):679–698, 1986. 29

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *CVPR*, 2017. 70

K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *Proc. BMVC.*, 2014. 111, 121

K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006. URL http://hal.inria.fr/inria-00112631/en. 120

X. Chen, A. Shrivastava, and A. Gupta. Neil: Extracting visual knowledge from web data. In *ICCV*, 2013. 73

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016. 153

Xianjie Chen and Alan L Yuille. Articulated pose estimation by a graphical model with image dependent pairwise relations. In *NIPS*, pages 1736–1744, 2014. 67

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 60, 242

L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard. Scaling Algorithms for Unbalanced Transport Problems. *To appear in Mathematics of Computation*, 2017. 261

O. Chum and A. Zisserman. An exemplar model for learning object classes. In *CVPR*, 2007. 74

Ramazan Gokberk Cinbis, Jakob Verbeek, and Cordelia Schmid. Weakly Supervised Object Localization with Multi-fold Multiple Instance Learning. *IEEE PAMI*, 39 (1):189–203, Jan 2017. 75

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011a. 25, 122, 154

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537, 2011b. 54, 102

D. Crandall and D. Huttenlocher. Weakly supervised learning of part-based spatial models for visual object recognition. In *ECCV*, 2006. 74

G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *ECCV Workshop*, 2004. 34, 87, 111

M. Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances. *NIPS*, 2013. 261

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 39, 87

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 87, 88

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a[U+FFFC] laplacian pyramid of adversarial networks. In *NIPS*, pages 1486–1494, 2015. 81, 141

T. Deselaers, B. Alexe, and V. Ferrari. Localizing objects while learning their appearance. In *ECCV*, 2010. 75

S. Divvala, A. Farhadi, and C. Guestrin. Learning everything about anything: Webly-supervised visual concept learning. In *CVPR*, 2014. 73

C. Doersch, S. Singh, A. Gupta, J. Sivic, and A.A. Efros. What makes paris look like paris? *ACM Transactions on Graphics (TOG)*, 31(4):101, 2012. 73

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, pages 1422–1430, 2015. 78, 134, 179

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014. 58, 107

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, 2011a. 247

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011b. 62

Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. 29

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *IJCV*, 88(2):303–338, Jun 2010. 17, 35, 51, 87, 88, 129, 177, 193

C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE PAMI*, 2013. 88, 97

A. Farhadi, M. K. Tabrizi, I. Endres, and D. Forsyth. A latent model of discriminative aspect. In *ICCV*, 2009. 59

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *CVIU*, 106(1):59–70, 2007. 51

P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008. 39, 40, 93, 111

P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE PAMI*, 32(9):1627–1645, 2010. 39, 40, 66, 72, 73, 75, 87, 118

Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004. 42

R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR*, 2003a. 74

R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR*, volume 2, pages 264–271, Jun 2003b. URL http://www.robots.ox.ac.uk/~vgg. 36, 38, 58

Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, 100(1):67–92, 1973. 36, 37

J. Foulds and E. Frank. A review of multi-instance learning assumptions. *The Knowledge Engineering Review*, 25(01):1–25, 2010. 76

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *ICLR*, 2016. 154

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 44, 87

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 17(59):1–35, 2016. 174

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 58, 63, 64, 65, 75, 102, 106, 111, 125, 127, 193

Ross Girshick. Fast R-CNN. In *ICCV*, pages 1440–1448, 2015. 63

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010. 54, 244

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. 54, 57

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS*, 2014. 24, 80, 135, 137, 138, 139, 164, 168, 170, 173, 178

Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1458–1465. IEEE, 2005. 35

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola. A kernel two-sample test. *JMLR*, 2012. 143, 146, 147

Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pages 513–520, 2007. 163

G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical report, CalTech, 2007. 87

S. Gross and M. Wilber. Training and investigating residual nets, 2016. URL http://torch.ch/blog/2016/02/04/resnets.html. 147

M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *CVPR*, 2009. 73

Ankur Handa, Michael Bloesch, Viorica Pătrăucean, Simon Stent, John McCormac, and Andrew Davison. gvnn: Neural network library for geometric computer vision. In *ECCV Workshop on Geometry Meets Deep Learning*, 2016. 26

Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *ECCV*, pages 297–312. Springer, 2014. 65

H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *CVPR*, 2009. 111

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 61, 147

Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. 1949. 43

M. Hejrati and D. Ramanan. Analyzing 3d objects in cluttered images. In *NIPS*, 2012. 73

G.E. Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007. 88

G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 51, 52, 88

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 50, 248

G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, University of Massachusetts, Amherst, 2007. 146, 149, 254

D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959. 44, 87

Daniel P Huttenlocher. Object recognition using alignment. *ICCV*, pages 102–111, 1987. 29

Aapo Hyvarinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*. 2001. 257, 258

Sergey Ioffe and David A. Forsyth. Probabilistic methods for finding people. *IJCV*, 43(1):45–68, 2001. 36, 38

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015. 61, 245, 251

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017. 136, 141, 143

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015. 26

Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1413–1421, 2015. 77

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014. 58, 59, 107

J. Jiang and C. Zhai. Instance weighting for domain adaptation in NLP. In *ACL*, 2007. 89

W. Jitkrittum, Z. Szabo, K. Chwialkowski, and A. Gretton. Interpretable distribution features with maximum testing power. *NIPS*, 2016. 147

J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *ECCV*, 2016. 149

Justin Johnson, Andrej Karpathy, and Li Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 26

Armand Joulin, Laurens van der Maaten, Allan Jabri, and Nicolas Vasilache. Learning visual features from large weakly supervised data. In *European Conference on Computer Vision*, pages 67–84. Springer, 2016. 73

M. Juneja, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *CVPR*, 2013. 58

Vadim Kantorov, Maxime Oquab, Minsu Cho, and Ivan Laptev. Contextlocnet: Context-aware deep network models for weakly supervised localization. In *ECCV*, pages 350–365. Springer, 2016. 25, 76, 130

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014. 70

Koray Kavukcuoglu, Rob Fergus, Yann LeCun, et al. Learning invariant features through topographic filter maps. In *CVPR*, pages 1605–1612. IEEE, 2009. 52, 53

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 70

J. D. Keeler, D. E. Rumelhart, and W. K. Leow. Integrated segmentation and recognition of hand-printed numerals. In *NIPS*, 1991. 112

A. Khosla, T. Zhou, T. Malisiewicz, A. A. Efros, and A. Torralba. Undoing the damage of dataset bias. In *ECCV*, 2012. 59

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015a. 149

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015b. 62, 247

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 135, 141

Jan J Koenderink and Andrea J van Doorn. Representation of local geometry in the visual system. *Biological cybernetics*, 55(6):367–375, 1987. 31

D. Kotzias, M. Denil, P. Blunsom, and N. de Freitas. Deep multi-instance transfer learning. *NIPS Deep Learning Workshop*, 2014. 76

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 22, 23, 27, 54, 55, 56, 57, 60, 70, 82, 83, 88, 89, 91, 93, 98, 103, 106, 111, 112, 116, 117, 119, 120, 177, 223, 234, 240, 241, 242, 245, 247, 250

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 55, 56

Jean Lafond, Nicolas Vasilache, and Léon Bottou. Diagonal rescaling for neural networks. Technical report, arXiV:1705.09319, 2017. URL http://leon.bottou.org/papers/lafond-vasilache-bottou-2017. 245

K.J. Lang and G.E. Hinton. A time delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, CMU, 1988. 45, 46, 49, 87, 111

K.J. Lang, A.H. Waibel, and G.E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3(1):23–43, 1990. 112

Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016. 242

Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979. 61, 212, 237

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 35, 36, 39, 87

Q. Le, W. Zou, S. Yeung, and A. Ng. Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011. 58

Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *ICML*, 2012. 53, 54, 92

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989. 45, 46, 60, 87, 111

Y. LeCun, L. Bottou, and Y. Bengio. Reading checks with graph transformer networks. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 151–154, Munich, 1997. IEEE. 47, 63

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *PIEEE*, 86(11):2278–2324, 1998a. 48, 55, 56, 87

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998b. 50, 54, 61, 244, 245, 251

Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*, volume 2, pages II–104. IEEE, 2004. 51, 87

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *CVPR*, 2016. 142

E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. Springer, 2006. 144

Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016. 154

Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International journal of computer vision*, 43(1):29–44, 2001. 30, 31, 33, 135

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015. 65, 66, 130

D. Lopez-Paz. *From dependence to causation*. PhD thesis, Univ. of Cambridge, 2016. 136, 153

D. Lopez-Paz, K. Muandet, B. Schölkopf, and I. Tolstikhin. Towards a learning theory of cause-effect inference. In *ICML*, pages 1452–1461, 2015. 161

David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *ICLR*, 2017. 25, 146

David Lopez-Paz, Robert Nishihara, Soumith Chintala, Bernhard Schölkopf, and Léon Bottou. Discovering causal signals in images. *CVPR*, 2017. 152, 153, 161, 174, 178

D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2): 91–110, 2004a. 87

David G Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2, pages 1150–1157. Ieee, 1999. 31

David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60 (2):91–110, 2004b. 32

David Marr and Herbert Keith Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London B: Biological Sciences*, 200(1140):269–294, 1978. 36, 37

M. Marszalek, C. Schmid, H. Harzallah, and J. van de Weijer. Learning object representations for visual object class recognition. In *Visual Recognition Challenge workshop, ICCV*, 2007. 99, 100

Michaël Mathieu, Mikael Henaff, and Yann Lecun. Fast training of convolutional networks through ffts. In *ICLR*, 2014. 242

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *ICLR*, 2017. 147, 157

Krystian Mikolajczyk and Cordelia Schmid. An affine invariant interest point detector. *ECCV*, pages 128–142, 2002. 34

M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv*, 2014. 141, 160

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 60

J. M. Mooij, J. Peters, D. Janzing, J. Zscheischler, and B. Schölkopf. Distinguishing cause from effect using observational data: methods and benchmarks. *JMLR*, 2016. 152, 159, 161, 180

Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *CVPR*, pages 3521–3529, 2016. 153

Hiroshi Murase and Shree K Nayar. Visual learning and recognition of 3-d objects from appearance. *IJCV*, 14(1):5–24, 1995. 30

Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983. 245

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, pages 483–499. Springer, 2016. 68, 69

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011. 77

S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. *NIPS*, 2016. 163

A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. http://distill.pub/2016/deconv-checkerboard/, 2016. 147, 148

M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, pages 1717–1724, 2014. 25, 111

Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. Is object localization for free?-weakly-supervised learning with convolutional neural networks. In *CVPR*, pages 685–694, 2015. 25

V. Ordonez, G. Kulkarni, and T. Berg. Im2text: Describing images using 1 million captioned photographs. In *NIPS*, 2011. 72, 73

R. Osadchy, M. Miller, and Y. LeCun. Synergistic face detection and pose estimation with energy-based model. In *NIPS*, 2005. 87

S. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. 59

M. Pandey and S. Lazebnik. Scene recognition and weakly supervised object localization with deformable part-based models. In *ICCV*, 2011. 75

G. Papandreou, I. Kokkinos, and P.-A. Savalle. Untangling Local and Global Deformations in Deep Convolutional Networks for Image Classification and Sliding Window Detection. In *CVPR*, 2015. 117

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016. 81, 82

J. Pearl. *Causality*. Cambridge University Press, 2009. 158

F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *ECCV*, 2010. 41, 87, 111

Tomas Pfister, James Charles, and Andrew Zisserman. Flowing convnets for human pose estimation in videos. In *ICCV*, pages 1913–1921, 2015. 67

Pedro O Pinheiro, Ronan Collobert, and Piotr Dollar. Learning to segment object candidates. In *NIPS*, pages 1990–1998, 2015. 66, 130

H. Pirsiavash and D. Ramanan. Detecting activities of daily living in first-person camera views. In *CVPR*, 2012. 89

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. 245

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992. ISBN 0-521-43108-5. 260

A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari. Learning object class detectors from weakly annotated video. In *CVPR*, 2012. 73

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016. 81, 135, 140, 141, 143, 146, 153, 156, 254

Varun Ramakrishna, Daniel Munoz, Martial Hebert, James Andrew Bagnell, and Yaser Sheikh. Pose machines: Articulated pose estimation via inference machines. In *ECCV*, pages 33–47. Springer, 2014. 68

MA Ranzato, Fu Jie Huang, Y-L Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *CVPR*, 2007a. 52

Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *NIPS*, pages 1185–1192. Curran Associates Inc., 2007b. 52

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR DeepVision workshop*, pages 806–813, 2014. 58, 59, 106, 111

Scott Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In *Advances in Neural Information Processing Systems*, 2016. 26

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015. 63, 64

X. Ren and D. Ramanan. Histograms of sparse codes for object detection. In *CVPR*, 2013. 58

Lawrence Gilman Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963. 28

F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Project PARA, Cornell Aeronautical Lab, 1957. 43, 87

Charles Rothwell, Andrew Zisserman, David Forsyth, and Joseph Mundy. Canonical frames for planar object recognition. In *ECCV*, pages 757–772. Springer, 1992. 29

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *IJCV*, 40(2):99–121, 2000. 166

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 45, 87, 135, 214, 250

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 2015. 22, 41, 56, 192

K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *ECCV*, 2010. 59, 89

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. *NIPS*, 2016. 142, 149

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *ICML*, pages 343–351, 2013. 61

Bernt Schiele and James L Crowley. Object recognition using multidimensional receptive field histograms. In *European Conference on Computer Vision*, pages 610–619. Springer, 1996. 30

Cordelia Schmid and Roger Mohr. Local grayvalue invariants for image retrieval. *IEEE PAMI*, 19(5):530–535, 1997. 31, 32

B. Schmitzer. Stabilized Sparse Scaling Algorithms for Entropy Regularized Transport Problems. *ArXiv e-prints*, 2016. 261

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002. 34

Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning. *ICML*, 2012. 152

P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *ICLR*, 2014. 59, 61, 63, 64, 66, 76, 106, 111, 115, 122, 130

A. Shrivastava and A. Gupta. Building part-based object detectors via 3d geometry. In *ICCV*, 2013. 73

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 60

P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003. 87

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, pages 568–576, 2014a. 70, 71

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014b. 61

S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*, 2012. 58

Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987. 30

J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, 2003. 32, 33, 58, 87, 111, 135

J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering object categories in image collections. In *ICCV*, 2005. 135

H. Song, R. Girshick, S. Jegelka, J. Mairal, Z. Harchaoui, and T. Darrell. On learning to localize objects with minimal supervision. In *ICML*, 2014. 75

Z. Song, Q. Chen, Z. Huang, Y. Hua, and S. Yan. Contextualizing object detection and classification. In *CVPR*, 2011. 99, 100, 121

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014. 201

Chen Sun, Manohar Paluri, Ronan Collobert, Ram Nevatia, and Lubomir Bourdev. Pronet: Learning to propose object-specific boxes for cascaded neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3485–3493, 2016. 76

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013. 246

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998. 133

Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991. 30

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 80

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 61

G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *ECCV*, 2010. 58

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *ICLR*, 2016. 142, 163, 180, 262

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012. 61, 247

T. Tommasi, F. Orabona, and B. Caputo. Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *CVPR*, 2010. 59

A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *CVPR*, 2011. 88, 89, 92, 93, 190

A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, 2014. 66, 111

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, pages 4489–4497, 2015. 70

Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991. 29, 30

R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localisation of objects in images. *IEE Proc on Vision, Image, and Signal Processing*, 141(4): 245–250, 1994. 87

K. van de Sande, J.R.R. Uijlings, T. Gevers, and A.W.M. Smeulders. Segmentation as Selective Search for Object Recognition. In *ICCV*, 2011. 41, 42, 63, 64, 75, 106, 129

Vladimir Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer, 2015. 198

Cédric Villani. *Optimal transport: old and new*. 2008. 171

P. Viola, J. Platt, C. Zhang, et al. Multiple instance boosting for object detection. In *NIPS*, 2005. 76

C. Wang, W. Ren, K. Huang, and T. Tan. Weakly supervised object localization with latent category learning. In *ECCV*. 2014. 75

Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, pages 3551–3558, 2013. 69

Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *CVPR*, pages 3169–3176, 2011. 69

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, pages 2794–2802, 2015. 78, 179

Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *CVPR*, pages 4724–4732, 2016. 68

Y. Wei, W. Xia, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan. Cnn: Single-label to multi-label. *arXiv:1406.5726*, 2014. 121

Isaac Weiss. Projective invariants of shapes. In *CVPR*, pages 291–297, 1988. 29

J. Winn and N. Jojic. Locus: Learning object classes with unsupervised segmentation. In *ICCV*, 2005. 74

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 60, 248

P. Yadollahpour, D. Batra, and G. Shakhnarovich. Discriminative re-ranking of diverse segmentations. In *CVPR*, 2013. 72

S. Yan, J. Dong, Q. Chen, Z. Song, Y. Pan, W. Xia, Z. Huang, Y. Hua, and S. Shen. Generalized hierarchical matching for sub-category aware object classification. In *Visual Recognition Challenge workshop, ECCV*, 2012. 99, 100, 101

Y. Yang and D. Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR*, 2011. 66, 111

Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. 146, 149, 254

M. Zeiler, G. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, 11. 92

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pages 818–833, 2014. 58, 106, 121, 232, 242, 243, 250

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2017. 247

J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *IJCV*, 73(2): 213–238, jun 2007. URL http://lear.inrialpes.fr/pubs/2007/ZMLS07. 111

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016. 78, 79

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object Detectors Emerge in Deep Scene CNNs. *ICLR*, 2015. 129, 130

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016. 141, 143

## Résumé

Les réseaux de neurones à convolution sont des algorithmes d'apprentissage flexibles qui tirent efficacement parti des importantes masses de données qui leur sont fournies pour l'entraînement. Malgré leur utilisation dans des applications industrielles dès les annees 90, ces algorithmes n'ont pas été utilisés pour la reconnaissance d'image à cause de leurs faibles performances avec les images naturelles. C'est finalement grâce a l'apparition d'importantes quantités de données et de puissance de calcul que ces algorithmes ont pu révéler leur réel potentiel lors de la compétition ImageNet, menant à un changement de paradigme en reconnaissance d'image.

La première contribution de cette thèse est une méthode de transfert d'apprentissage dans les réseaux à convolution pour la classification d'image. À l'aide d'une procédure de pré-entraînement, nous montrons que les représentations internes d'un réseau à convolution sont assez générales pour etre utilisées sur d'autres tâches, et meilleures lorsque le pré-entraînement est réalisé avec plus de données.

La deuxième contribution de cette thèse est un système faiblement supervisé pour la classification d'images, pouvant predire la localisation des objets dans des scènes complexes, en utilisant, lors de l'entraînement, seulement l'indication de la présence ou l'absence des objets dans les images.

La troisième contribution de cette thèse est une recherche de pistes de progression en apprentissage non-supervisé. Nous étudions l'algorithme récent des réseaux génératifs adversariaux et proposons l'utilisation d'un test statistique pour l'évaluation de ces modèles. Nous étudions ensuite les liens avec le problème de la causalité, et proposons un test statistique pour la découverte causale. Finalement, grâce a un lien établi récemment avec les problèmes de transport optimal, nous étudions ce que ces réseaux apprennent des données dans le cas non-supervisé.



## Mots Clés

Vision artificielle, réseaux de neurones à convolution, apprentissage profond, apprentissage faiblement supervisé, apprentissage non-supervisé, réseaux génératifs adversariaux.



## Abstract

Convolutional Neural Networks are flexible learning algorithms for computer vision that scale particularly well with the amount of data that is provided for training them. Although these methods had successful applications already in the '90s, they were not used in visual recognition pipelines because of their lesser performance on realistic natural images. It is only after the amount of data and the computational power both reached a critical point that these algorithms revealed their potential during the ImageNet challenge of 2012, leading to a paradigm shift in visual recogntion.

The first contribution of this thesis is a transfer learning setup with a Convolutional Neural Network for image classification. Using a pre-training procedure, we show that image representations learned in a network generalize to other recognition tasks, and their performance scales up with the amount of data used in pre-training.

The second contribution of this thesis is a weakly supervised setup for image classification that can predict the location of objects in complex cluttered scenes, based on a dataset indicating only with the presence or absence of objects in training images.

The third contribution of this thesis aims at finding possible paths for progress in unsupervised learning with neural networks. We study the recent trend of Generative Adversarial Networks and propose two-sample tests for evaluating models. We investigate possible links with concepts related to causality, and propose a two-sample test method for the task of causal discovery. Finally, building on a recent connection with optimal transport, we investigate what these generative algorithms are learning from unlabeled data.



## Keywords

Computer vision, convolutional neural networks, deep learning, weakly supervised learning, unsupervised learning, generative adversarial networks.