Incremental Learning for Semantic Segmentation of Large-Scale Remote Sensing Data

Onur Tasar, Yuliya Tarabalka, Pierre Alliez

Incremental Learning for Semantic Segmentation of Large-Scale Remote Sensing Data

Onur Tasar, *Student member, IEEE,* Yuliya Tarabalka, *Senior member, IEEE,* Pierre Alliez

***Abstract*—In spite of remarkable success of the convolutional neural networks on semantic segmentation, they suffer from catastrophic forgetting: a significant performance drop for the already learned classes when new classes are added on the data having no annotations for the old classes. We propose an incremental learning methodology, enabling to learn segmenting new classes without hindering dense labeling abilities for the previous classes, although the entire previous data are not accessible. The key points of the proposed approach are adapting the network to learn new as well as old classes on the new training data, and allowing it to remember the previously learned information for the old classes. For adaptation, we keep a frozen copy of the previously trained network, which is used as a memory for the updated network in absence of annotations for the former classes. The updated network minimizes a loss function, which balances the discrepancy between outputs for the previous classes from the memory and updated networks, and the mis-classification rate between outputs for the new classes from the updated network and the new ground-truth. For remembering, we either regularly feed samples from the stored, little fraction of the previous data or use the memory network, depending on whether the new data are collected from completely different geographic areas or from the same city.**

Our experimental results prove that it is possible to add new classes to the network, while maintaining its performance for the previous classes, despite the whole previous training data are not available.

***Index Terms*—Incremental learning, catastrophic forgetting, semantic segmentation, convolutional neural networks**

I. Introduction

RECENT improvements in satellite sensors have enabled us to capture massive amounts of remote sensing data with high spatial resolution, as well as rich spectral information. Generation of maps from such huge amounts of satellite images and updating them automatically have been long standing problems, as they are crucial for a wide range of applications in domains such as agriculture, navigation, environmental management, urban monitoring, and mapping. In this context, having a strong classification system, which performs a high-quality, pixel-wise, large-scale classification is the most essential step.

In the last decade, with the great advances in deep neural networks, notably convolutional neural networks (CNNs), it has been possible to obtain accurate segmentations [1]–[3]. Among the CNN-based approaches, U-net architecture [4] has gained a particular attention due to its success in various segmentation problems in different domains (e.g., medical imaging and remote sensing). This network architecture consists of a contracting path that captures the context and a symmetric expanding path, enabling accurate localization. In addition to traditional encoder-decoder layers, U-net architecture uses skip connections, which combine low level features with the high level ones in the expanding path to increase precision of localization. Variants of this network [5]–[7], (e.g., U-net, including VGG-11 [8] encoder and corresponding decoder) have been applied to remote sensing images and have shown a remarkable performance.

The major drawback of the recently proposed methodologies is their assumption that the whole training data are available in the beginning, which is not the case in real world remote sensing applications, as new images are collected from all over the world everyday. Besides, having large amounts of standard and unique label maps is almost impossible, because the label maps retrieved from different sources usually have distinct classes. In addition, it is not always possible to store enormous volume of training data. For the reasons described above, designing an incremental learning methodology, which can learn from the new training data while retaining performance for the old classes without accessing to the entire previous training data is crucial. Although a good solution for this problem is necessary to generate high-quality maps from satellite images that cover a large geographic extent, yet it has remained unexplored in remote sensing community.

Rather than assuming that we have all the training data initially, we aim to design an incremental learning methodology. Let us explain with an example of a real-world problem (see Fig. 1) where, in the beginning, we are provided with images from several cities in Austria with correspondent label maps for building and high vegetation classes. Later on, we are given other training data, having label maps for water class, collected from different areas in Germany. Finally, we receive new satellite images and their annotations for road and railway classes from certain cities in France. Every time when new data arrive, we assume that only a small portion of the previous data is stored. In such a scenario, our goal is to add segmentation capabilities for the new classes to the previously trained network without forgetting the already learned information so that maps for all the learned classes could be generated by the network. In addition to the described problem, because labeling satellite images covering a large geographic area requires a lot of manual work, it is quite common that annotations of different classes for the same images are provided sequentially in time. In this kind of situation,

O. Tasar, and P. Alliez are with Université Côte d'Azur and also with Inria, TITANE team, 06902 Sophia Antipolis, France. E-mail: onur.tasar@inria.fr

Y. Tarabalka is with LuxCarta Technology, Parc d'Activité l'Argile, Lot 119b, Mouans Sartoux 06370, France.

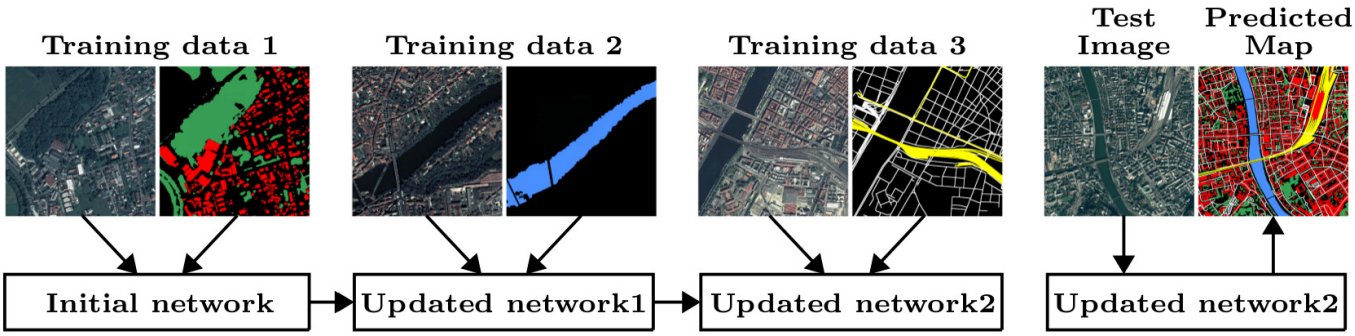


Fig. 1. An example incremental learning scenario. Firstly, satellite images as well as their label maps for building and high vegetation classes are fed to the network. Then, from the second training data, the network learns water class without forgetting building and high vegetation classes. Finally, road and railway classes are taught to the network. Whenever new training data are obtained, we store only a small part of the previous ones for the network to remember. When a new test image comes, the network is able to detect all the classes.

it is not feasible to train a new classifier from scratch every time new label maps are obtained. The limitations pointed out in this section motivated us to design an incremental learning methodology.

A. Related Work

The biggest challenge in incremental learning problem is that when new tasks are intended to be added to a classification system, performance of the system for the previously learned tasks degrades abruptly, which is referred as "catastrophic forgetting" in the literature [10], [11]. Incremental learning has been a historically important problem. Even before neural networks have become popular, researches had been studying this issue [12]–[15]. More recently, various convolutional neural network based methodologies have been proposed. There have been attempts, which change architecture of the neural network as new classes are added. In [16], the network is trained incrementally by sharing early layers and splitting later ones by adding new convolutional kernels. In [17], a tree-structured model, which grows hierarchically, is proposed. In [18] and [19], described approaches grow the network horizontally. The methodology described in [20] tries to solve the problem of determining the number of filters to be added to each layer by reinforcement learning. The major weakness of these approaches is that since the network grows during training, the number of parameters increases drastically as new tasks are added to the network. The methodologies proposed in [21]–[23] use not only the new training data but also a small portion of the old data. To determine the most important samples for the previous classes, the approach in [24] trains a Support Vector Machine (SVM) from the previous training data. The support vectors of the SVM correspond to the samples to be used for the former classes, while the network is adapted to the new training data. In [25], [26], instead of using the old data directly, fake previous data are generated by generative adversarial networks (GAN). It has been proven that many configurations of the network parameters may produce the same result [27]. Inspired by this idea, several works, which try to find a configuration of the network parameters that represents both the previous and the new training data well, have been published. The key idea behind these approaches is to find the important neurons for the old tasks and prevent these neurons from changing greatly or completely when the new tasks are added to the network. The proposed methodology explained in [28] is one of the approaches that falls into this category. In the loss function defined in the paper, there is an elastic weight consolidation (EWC) term, which is a multiplication of the importance value of parameters for the old tasks and quadratic penalty on difference between parameters of the previous and the updated networks. The importance value of the parameters is measured by the estimated diagonal Fisher information matrix. The same work has been extended in [29] by rotating the Fisher matrix. [30] is also quite similar to [28], but the elastic weight consolidation is performed in online fashion. In [31], importance of each neuron is determined by averaging gradients of the network output with respect to parameters of the neuron. In [32], in the training stage, features from the previous data are reconstructed in unsupervised manner using autoencoders. The features are then used to preserve information, which the old tasks rely on when the new tasks are added. [33] is another extension of [28], where trained models for all the tasks are combined via incremental moment matching (IMM). The proposed approaches in [34] and [35] try to learn a mask, which marks important neurons for the old tasks. When the new tasks need to be added, only the masked out neurons are updated. In [36], paths through the network, which represent a subset of parameters are determined by using tournament selection genetic algorithm. During the training stage, only the neurons that are located along the paths are updated. When the data come sequentially, the works explained in [37]–[39] optimize the parameters of the network by updating the posterior approximation by the Bayesian inference based methods. Distilling the knowledge approach proposed in [40], which enables to transfer the knowledge from a network or an assembly of several networks to a smaller network has inspired several works on incremental learning. The proposed methods in [41], [42] facilitate a similar distillation loss described in [40] to maintain performance on the previous tasks. The proposed approach in [43] uses a distillation loss function, which also uses samples for the previous classes in addition to samples for the new classes. Another knowledge distillation

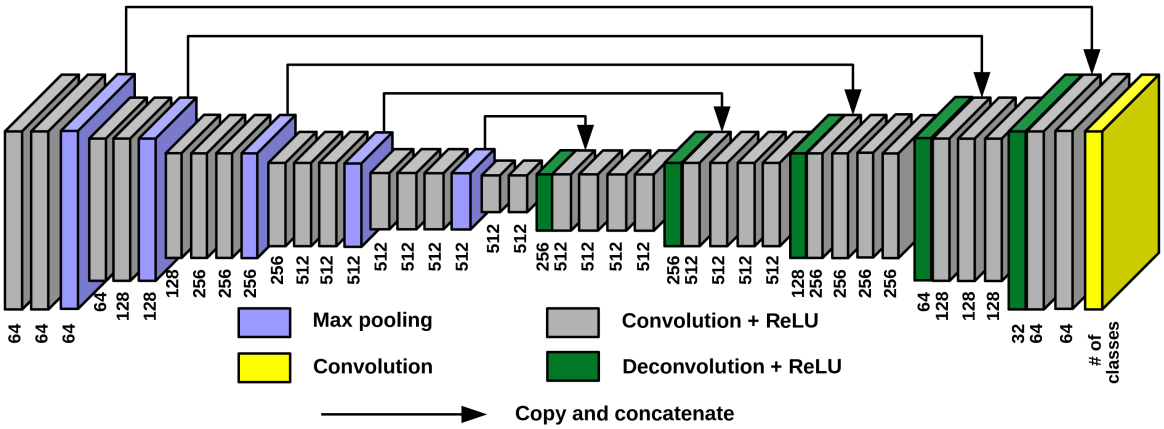


Fig. 2. The network structure. The number below each layer corresponds to number of filters. We refer the last layer, shown by yellow color as the classification, and the rest as the shared layers.

based approach has been proposed in [44]. They deal with incremental object detection and classification tasks at the same time. Although the incremental learning problem has been explored in depth in the literature, none of the works described in this section studies incremental learning for dense labeling.

B. Contributions

We propose a novel incremental learning methodology for semantic segmentation problem, where the network learns segmenting new classes without deteriorating performance for the previously learned classes, even when the entire previous training data are not stored[1].

We deal with two common real-world problems, in which the former is the situation of retrieving stream of training data, where at each time step, the data contain satellite images collected from different locations in the world and annotations for separate classes, the latter is the case, where label maps for the same geographic area are provided sequentially. To investigate how our methodology performs on the first problem, we test our approach on the Luxcarta dataset, consisting of the satellite images captured over different cities in France and Austria. For the second problem, we conduct experiments on the Vaihingen and the Potsdam benchmark datasets provided by the ISPRS [45]. The first problem is much more challenging, as the satellite images have high color variations and visual feature differences. Besides, for the first problem, by following a similar strategy described in [9], we test the trained models on the data collected from completely different geographic areas than the ones we use during training.

We provide rich experimental results for both problems by comparing our methodology with *static learning, multiple learning, fixed representation*, and *fine-tuning* (see Sec. III-A). Our experimental results prove that by training only one network, it is possible to learn new classes without catastrophically forgetting the previous classes. To the best of our knowledge, this is the first work, which proposes a solution for the incremental semantic segmentation problem.

[1]Project page: https://project.inria.fr/epitome/inc_learn

II. Methodology

A. Network Architecture

Our network (see Fig. 2) is a variant of U-net, which consists of an encoder that is architecturally the same as the first 13 convolutional layers of VGG16 [8], a corresponding decoder, mapping low resolution encoder feature maps to original input image size of outputs, and two center convolutional layers. We prefer to use VGG16 as the encoder, because it provides a good compromise between complexity and performance, as it is not as deep as e.g., VGG19 but still it is one of the best performers on famous benchmark challenges (e.g., ImageNet [46]).

The output of each pooling layer in the encoder is concatenated with the output of the symmetric deconvolutinal layer in the decoder through skip connections to combine higher level features with the lower ones. Kernel size and stride in all the convolutional layers are 3 and 1 respectively. Padding parameter in the convolutional layers is set to 1 so as to keep height and width of output the same as output of the previous layer. The max-pooling layers, having 2×2 window with stride 2 are used to halve width and height of the previous layer. In order to upsample output of the previous layer by factor of 2, both kernel size and stride parameters are set to 2 in deconvolutional layers. Except the last convolutional layer, all the convolution and deconvolution operations are followed by a ReLU. Since batch normalization uses the memory inefficiently, we prefer not to use it to add more patches in a batch.

Multi-task learning is the learning strategy which solves multiple problems at the same time by learning all the tasks jointly. In deep neural networks, bottom layers enable to share information for all the tasks, whereas the last layers are dedicated to provide a solution for each task [47], [48]. In incremental semantic segmentation problem, since the label maps of a remote sensing image for a class or several classes come sequentially, we consider the segmentation tasks as a multi-task learning problem, where performing a binary classification for each class corresponds to a different task. The output of our network is a 3D matrix that is a stack of

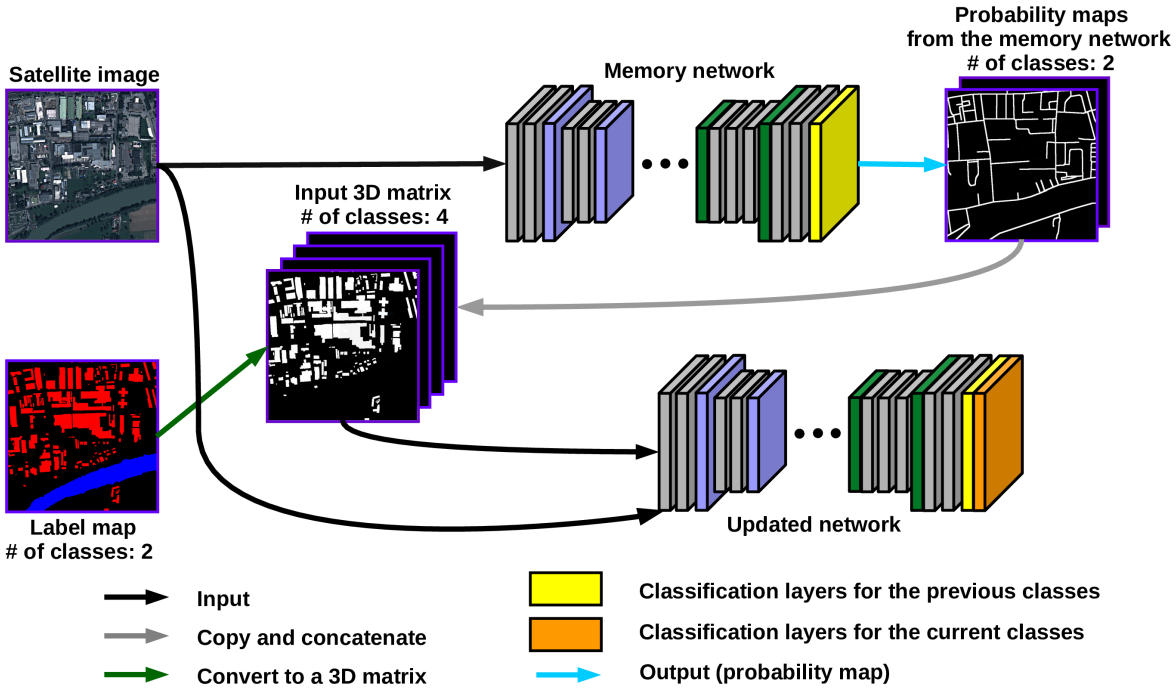


Fig. 3. Adapting the network to the new training data. Although annotations for only 2 classes are provided, the updated network is still able to learn current classes as well as the previously learned 2 classes with the help of the memory network.

binary predicted maps for all the classes. In the test stage, to generate a binary segmentation for each class, we first convert outputs of the final convolutional layers to probability maps using sigmoid; then, we threshold the probabilities at 0.5.

B. Adapting the Network to the New Training Data

To explain the adaptation phase, let us assume that the current training data are indicated by D_{curr}. We denote sets of the previously learned classes and the classes in D_{curr} by $\mathcal{L}_{prev}$ and $\mathcal{L}_{curr}$, where $\mathcal{L}_{prev} \cap \mathcal{L}_{curr} = \emptyset$. The main goals we try to achieve during adaptation are to update the formerly trained network so that segmentation capabilities for $\mathcal{L}_{curr}$ are added, and to fine-tune the network on D_{curr} for $\mathcal{L}_{prev}$, although annotations for $\mathcal{L}_{prev}$ are not available in D_{curr}. The output of the updated network is the matrix consisting of binary segmentations for $\mathcal{L}_{updated} = \mathcal{L}_{prev} \cup \mathcal{L}_{curr}$.

We use the knowledge distillation from the previously trained network, which we refer to as memory network, as a proxy in absence of the ground-truth for $\mathcal{L}_{prev}$ in D_{curr}. We create an updated network, having exactly the same structure except the last classification layer, which has $|\mathcal{L}_{updated}|$ filters instead of $|\mathcal{L}_{prev}|$. During creation of the updated network, additional $|\mathcal{L}_{curr}|$ filters in the last classification layer are initialized using Xavier initialization [49], and the rest of the parameters are loaded from the memory network. When D_{curr} arrive, the incoming label map is first converted to a 3D matrix, consisting of binary ground-truth for $\mathcal{L}_{curr}$. The probability maps generated by the memory network are concatenated with this 3D matrix to provide information about $\mathcal{L}_{prev}$ to the updated network. The final 3D matrix as well as the input image in D_{curr} are fed to the network as the new training data. While concatenating output of the memory network with the new ground-truth, we prefer to use soft probability maps generated by the memory network rather than hard classification maps in order to reduce the propagated error rate, caused by imprecision in output of the memory network, at each time step of incremental learning.

Let us denote the binary target label vectors for n training samples $i = 1 \dots n$ in a batch from D_{curr} by $\mathbf{y}^{(i)}_{curr}$ and the predicted probabilities for $\mathcal{L}_{prev}$ from the memory network by $\hat{\mathbf{y}}^{(i)}_{mem}$. We denote by $\hat{\mathbf{y}}^{(i)}_{up\_curr}$ and $\hat{\mathbf{y}}^{(i)}_{up\_prev}$, the predicted probabilities for $\mathcal{L}_{curr}$ and $\mathcal{L}_{prev}$ from the updated network. The classification loss L_{class} quantifies mismatch between $\mathbf{y}^{(i)}_{curr}$ and $\hat{\mathbf{y}}^{(i)}_{up\_curr}$. In order to compute L_{class}, since we deal with generation of a binary segmentation for each class as a separate task, we use sigmoid cross entropy loss defined as:

$$L_{class} = -\frac{1}{n|\mathcal{L}_{curr}|} \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{L}_{curr}|} \Big[y^{(i)}_{curr(k)} \log \Big(\hat{y}^{(i)}_{up\_curr(k)} \Big) + \Big(1 - y^{(i)}_{curr(k)} \Big) \log \Big(1 - \hat{y}^{(i)}_{up\_curr(k)} \Big) \Big]. \tag{1}$$

In order for the updated network to learn $\mathcal{L}_{prev}$ on D_{curr}, we try to keep discrepancy between $\hat{\mathbf{y}}^{(i)}_{up\_prev}$ and $\hat{\mathbf{y}}^{(i)}_{mem}$ as small as possible. The distillation loss L_{distil}, which measures this disparity is defined as:

$$L_{distil} = -\frac{1}{n|\mathcal{L}_{prev}|} \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{L}_{prev}|} \Big[\hat{y}^{(i)}_{mem(k)} \log \Big(\hat{y}^{(i)}_{up\_prev(k)} \Big) + \Big(1 - \hat{y}^{(i)}_{mem(k)} \Big) \log \Big(1 - \hat{y}^{(i)}_{up\_prev(k)} \Big) \Big]. \tag{2}$$

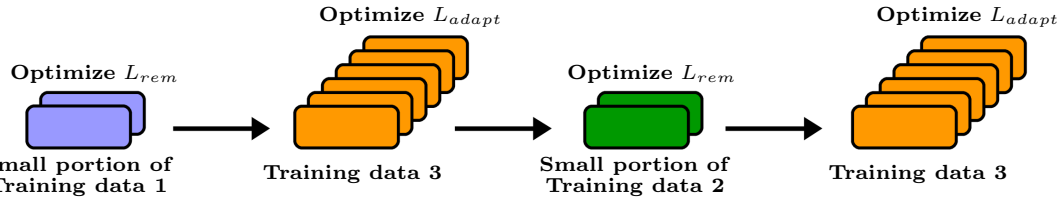


Fig. 4. An example optimization sequence. The new classes are added on Training data 3 to the network, which was already trained on Training data 1 and Training data 2. The optimization sequence is as follows: L_{rem} on Training data 1, L_{adapt} on Training data 3, L_{rem} on Training data 2, and L_{adapt} on Training data 3 again.

The overall adaptation loss L_{adapt} that is optimized during adaptation is computed by adding these two terms:

$$L_{adapt} = L_{class} + L_{distil}. \tag{3}$$

Fig. 3 depicts how the network is adapted to the new data.

C. Remembering From the Previous Training Data

We denote the previous training data by $D_{prev} = D_{prev}^{(1)} \cup D_{prev}^{(2)} \cup \ldots \cup D_{prev}^{(m)}$, where $D_{prev}^{(1)}$ corresponds to the first data, $D_{prev}^{(2)}$ is the second data, and so forth. If the training data are captured sequentially from different geographic locations, in order for the network not to overfit on D_{curr} for $\mathcal{L}_{prev}$, we remind the previously learned information by systematically showing patches from the stored, small portion of D_{prev}. Since in most of the cases classes in the training data are highly imbalanced, when determining which training patches to store in $D_{prev}^{(j)}$, random selection may cause storing no samples for less frequent classes. For this reason, we take the class imbalance problem into account. We first compute weight w_c of each class $c \in \mathcal{L}_{prev}^{(j)}$ in $D_{prev}^{(j)}$ as:

$$w_c = \frac{\text{median}(f_c | c \in \mathcal{L}_{prev}^{(j)})}{f_c}, \tag{4}$$

where f_c denotes frequency of the pixels that are labeled as class c. We then assign an importance value $I^{(l)}$ to the l^{th} training patch in $D_{prev}^{(j)}$ as:

$$I^{(l)} = \sum_{c \in \mathcal{L}_{prev}^{(j)}} w_c f_c^{(l)}, \tag{5}$$

where $f_c^{(l)}$ denotes the number of pixels, belonging to c in the patch. We store certain number of patches that have the highest I value, which we denote by $D_{prev\_imp}^{(j)}$. In order to diversify the patches that are fed to the network, we randomly select a small fraction of the remaining patches. We denote the randomly chosen patches by $D_{prev\_random}^{(j)}$. The data to be stored from $D_{prev}^{(j)}$ for remembering are $D_{prev\_rem}^{(j)} = D_{prev\_imp}^{(j)} \cup D_{prev\_random}^{(j)}$. The number of patches that is selected randomly and using the importance value needs to be determined by the end user.

Let us denote the target vector for the i^{th} sample among n samples in a batch from $D_{prev\_rem}^{(j)}$ by $\mathbf{y}_{prev}^{(j)(i)}$. We denote by $\hat{y}_{up\_prev}^{(j)(i)}$ the predicted vector from the updated network for the same sample. The remembering loss L_{rem} is calculated as:

$$L_{rem} = -\frac{1}{n|\mathcal{L}_{prev}^{(j)}|} \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{L}_{prev}^{(j)}|} \Big[y_{prev(k)}^{(j)(i)} \log\left(\hat{y}_{up\_prev(k)}^{(j)(i)}\right) + \left(1 - y_{prev(k)}^{(j)(i)}\right) \log\left(1 - \hat{y}_{up\_prev(k)}^{(j)(i)}\right) \Big]. \tag{6}$$

During remembering from $D_{prev}^{(j)}$, we freeze the classification layers that are responsible for $c \notin \mathcal{L}_{prev}^{(j)}$ and optimize the rest of the network. The user needs to determine how often and on which data L_{rem} is optimized. An example optimization sequence is depicted in Fig. 4.

III. Experiments

A. Methods Used for Comparison

Table I compares our methodology with the following approaches:

Static learning: This is the traditional learning approach, where we assume that all the training images and annotations for the same classes are available at the time of training. In real-world segmentation problems, this condition is extremely hard to meet. This method does not support learning new classes continually.

Multiple learning: In this learning strategy, we train an additional classifier whenever the new training data are obtained. The number of classifiers that needs to be stored increases linearly. In addition, because the test images have to be segmented using all the trained classifiers to generate a map for each class, the test stage might be extremely long. Therefore, this approach is extremely expensive in terms of storage and segmentation efficiency.

Fixed representation: To learn new classes, we remove the classification layers, which were optimized for the previous classes, and plug in new classification layers dedicated for the new classes. The newly added classification layers are initialized with Xavier method [49]. When new training data arrive, we optimize only the newly added classification layers and freeze the rest of the network. Hence, training is very fast. During testing, we append the formerly trained classification layers back to the network to generate label maps for all the classes. The major issue is that although performance for the initial classes is preserved, the network struggles in learning new classes, because the previously extracted features are not optimized to represent the new classes.

TABLE I
ADVANTAGES AND DISADVANTAGES OF OUR APPROACH WITH RESPECT TO THE COMPARED METHODS.

Method	Training Time (1 iteration)	Test Time	Performance for the new classes	Performance for the old classes	Convergence time for the new classes	Number of Classifiers
static learning	fast	fast	continual learning is not supported	continual learning is not supported	continual learning is not supported	1
multiple learning	fast	very slow	good	good	medium	N
fixed representation	very fast	fast	very bad	good	cannot learn	1
fine-tuning	fast	fast	good	very bad	very fast	1
incremental learning	medium	fast	good	good	very fast	1

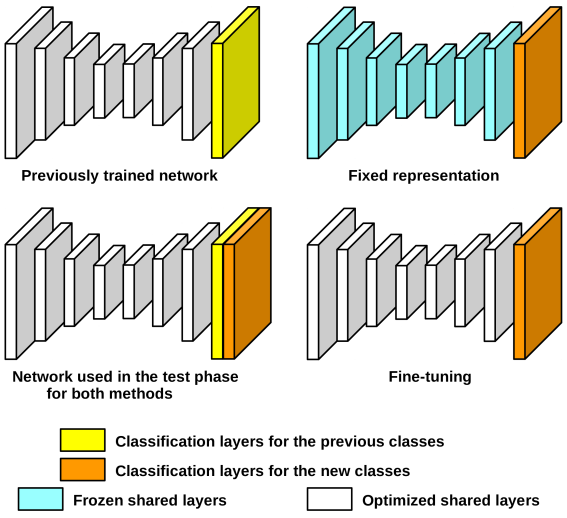


Fig. 5. Example network structures for *fixed representation* and *fine-tuning*. During the test stage, classification layers for the previous classes are appended to the network to generate label maps for all the classes.

Fine-tuning: We use a similar strategy that we follow in *fixed representation*. The only difference is that while training the network, instead of only the classification layers, we optimize the whole network using only the new training data. In this methodology, although the network performs a remarkable performance for the new classes, it suffers from catastrophic forgetting. Example network structures for *fixed representation* and *fine-tuning*, for both training and test phases, are illustrated in Fig. 5.

For *incremental learning*, it is required for the memory network to generate probability maps from the training patches to optimize L_{distil}. Therefore, training time for our approach is slightly longer than the others. This can be considered as the only disadvantage of the proposed methodology.

B. Datasets and Evaluation Metrics

The first data we use are the Luxcarta dataset, containing 8 bit satellite images collected from 22 different cities in Europe. 11 of these cities are located in France and the other 11 are in Austria. The cities cover the total area of approximately 1367 km^2. The images were collected from the following cities: *Amstetten*, *Enns*, *Leibnitz*, *Salzburg*, *Villach*, *Bad Ischl*, *Innsbruck*, *Klagenfurt*, *Osttirol*, *Sankt Pölten*, *Voitsberg* in *Austria*, and *Albi*, *Angers*, *Bayonne-Biarritz*, *Béziers*, *Bourges*, *Douai*, *Draguignan*, *Lille*, *Lyon*, *Nîmes*, *Roanne* in *France*. The spectral bands of the images are composed of Red (R), Green (G), and Blue (B) channels. The spatial resolution is 1 m. Since the images were captured over different geographic locations, they have different color distributions and visual features. The annotations for *building*, *road*, *high vegetation*, *water*, and *railway* classes are provided.

The other two datasets, on which we conduct our experiments are the Vaihingen and the Potsdam benchmarks provided by the ISPRS [45]. Both datasets contain 8 bit aerial images. The Vaihingen dataset consists of 33 image tiles (of average size 2494 $\times$ 2064), where 16 of them are provided as training and the rest as test. The images comprise 3 spectral bands: Near Infrared (NIR), R, and G. The spatial resolution is 9 cm. The Potsdam dataset includes 38 tiles (of size 6000 $\times$ 6000), out of which 24 are dedicated for training and the remaining for test. The images contain 5 channels: NIR, R, G, B, and the normalized DSM (nDSM) data. The resolution of the images in this benchmark is 5 cm. Both datasets contain full annotations for 6 classes: *impervious surfaces*, *building*, *low vegetation*, *high vegetation*, *car*, and *clutter*. However, since only 0.78% of the pixels in the Vaihingen dataset is labeled as *clutter*, we ignore this class in the experiments on this benchmark. As of 2018 summer, the competition for these benchmarks is over, and all the reference data are publicly available. Hence, we use all the training tiles for training, and test tiles for validation. To account for the labeling mistakes while the datasets are annotated, the eroded ground-truth is also provided. We use this ground-truth to assess the performance on the benchmarks.

To quantitatively assess the performance for each class, we compare the binary predicted map and the binary ground-truth using two evaluation metrics: intersection over union (IoU) [50] and F1-score [3].

C. Experiments on the Luxcarta Dataset

In this experimental setup, we suppose that the training data are obtained sequentially in time, and every snapshot of the streaming training data contains the satellite images from different cities and label maps for separate classes. As the training data, we use 18 cities, out of which 9 are located in *Austria* and the other 9 are in *France*. We use 2 cities from each country for validation. We split the training cities into three sets as reported in Table II by paying attention that the cities in each set are the ones, which contain a reasonable amount of samples for the given annotations, and whose color distributions are as diverse as possible. We assume that

TABLE II
TRAINING AND VALIDATION CITIES OF THE LUXCARTA DATASET.

Data Type and Classes for				**City (Country)**	**Area (km²)**	**Class Frequency (%)**				
mult. learning fixed rep. fine-tuning inc. learning		*static learning*				building	high veg.	road	railway	water
Train1	building	Train	building	*Bad Ischl* (*Austria*)	27.71	5.51	35.38	5.87	0.16	2.58
				Osttirol (*Austria*)	28.38	6.96	15.37	7.72	0.44	0.84
				Voitsberg (*Austria*)	28.70	7.47	30.21	6.54	0.44	0.98
				Bayonne-Biarritz (*France*)	66.58	15.21	19.45	12.66	0.45	1.26
				Bourges (*France*)	72.20	9.81	14.83	10.10	0.42	0.92
	high veg.		high veg.	*Draguignan* (*France*)	25.54	9.64	34.99	10.24	0.00	0.08
				Nîmes (*France*)	26.62	21.78	19.65	14.10	1.30	0.04
Train2	road		road	*Enns* (*Austria*)	64.49	6.25	12.54	6.81	1.36	2.82
				Innsbruck (*Austria*)	132.50	8.92	22.78	7.00	0.90	2.97
				Klagenfurt (*Austria*)	67.73	10.96	18.89	9.05	0.65	1.20
				Sankt Pölten (*Austria*)	87.17	6.68	25.13	5.40	0.99	1.70
	railway		railway	*Béziers* (*France*)	25.75	19.09	10.91	16.10	1.52	0.78
				Lyon (*France*)	187.14	18.48	16.82	12.59	1.41	2.83
Train3	water		water	*Albi* (*France*)	25.76	17.20	15.19	13.93	0.55	1.65
				Villach (*Austria*)	43.59	9.26	19.91	9.61	1.02	2.69
				Salzburg (*Austria*)	134.71	9.44	23.88	7.90	0.79	2.41
				Angers (*France*)	74.16	15.78	15.97	10.40	0.63	1.39
				Douai (*France*)	58.10	13.31	14.62	8.63	0.93	2.09
Validation	building high veg. road railway water			*Amstetten* (*Austria*)	14.26	11.11	15.61	9.67	1.85	1.72
				Leibnitz (*Austria*)	32.72	6.96	16.84	6.99	0.34	3.30
				Lille (*France*)	117.58	18.36	15.40	11.39	1.32	1.02
				Roanne (*France*)	25.84	18.44	8.33	14.00	0.78	0.95

the training cities are streamed in this order: Train1, Train2, Train3. For *multiple learning*, *fixed representation*, and *fine-tuning* we assume that the previous data are not accessible. For *incremental learning*, we store only 30% of the training patches in the previous data, out of which 15% are selected using the importance value and 15% are chosen randomly, as explained in section II-C. We also test our approach without accessing to the previous data (i.e., without optimizing L_{rem}), which we refer as *incremental learning w/o* L_{rem}. Since *static learning* does not support adding new classes continually, for this approach, we use all the training images from 18 different cities and label maps of all 5 classes for each image when training a network. For this reason, we expect it to be an obvious upper bound of the other methods.

During the pre-processing step, we split all the training images into 384 $\times$ 384 patches with an overlap of 32 $\times$ 32 pixels between the neighboring patches. The validation images are divided into 2240 $\times$ 2240 patches with 64 $\times$ 64 pixels of overlap. After all the validation patches are classified, they are combined back to get the original size classification maps. Because the satellite images arrive sequentially (except for *static learning*), it is not possible to compute mean values for the image channels. Hence, for the normalization, we subtract 127 from all the pixels, as the images are 8 bit.

We train a single model for *static learning* using the whole training data for 500 epochs, in which each epoch has 100 iterations. For *multiple learning*, we train 3 separate models from scratch on Train1, Train2, and Train3 with the same hyper-parameters. For *fixed representation*, *fine-tuning*, and the proposed *incremental learning* methodologies, every time when the new classes are added from the new data, we optimize the network for the same number of epochs and iterations as for *static learning* and *multiple learning*. In every 5 training iterations of the network for *incremental learning* approach on Train2, we optimize L_{rem} on Train1 for 1 iteration and L_{adapt} for the next consecutive 4 iterations. During the training on Train3, since the network has already learned information from both Train1 and Train2, we prefer to remind the network the previously learned information more often. On Train3, the optimization sequence as follows: L_{rem} on Train1 for 1 iteration, L_{adapt} for 2 iterations, L_{rem} on Train2 for 1 iteration, and L_{adapt} for 2 iterations again.

(a) $k = 1.4$ (b) $k = 0.6$ (c) Image (d) $\gamma = 0.6$ (e) $\gamma = 1.4$

Fig. 6. Illustration of the contrast change (a - b) and the gamma correction (d - e) for an example input image (c).

To update parameters of the network, we use *Adam optimizer*, where the learning rate is 0.0001, exponential decay rate for the first and the second moment estimates are 0.9 and 0.999, respectively. In every training iteration, a mini-batch of 12 patches is used for the optimization. When sampling a patch, we first select a random country (i.e., *Austria* or *France*). We then sample a random patch belonging to the city, which is also randomly chosen from the selected country. While training the network we apply online data augmentation to enrich the training data. The patches are augmented

TABLE III
F1 SCORES ON THE LUXCARTA DATASET.

Method	**Epoch**	**Building**	**High veg.**	**Road**	**Railway**	**Water**	**Overall**
static learning	**500**	80.74 (Ref.)	71.26 (Ref.)	66.21 (Ref.)	61.72 (Ref.)	82.74 (Ref.)	72.54 (Ref.)
multiple learning	**500**	71.25	68.88	59.28	**55.65**	79.83	66.98 (-5.56)
fixed representation	1000	71.25	68.88	2.71	0.00	—	
	1500	71.25	68.88	2.71	0.00	0.11	28.59 (-43.95)
fine-tuning	1000	28.91	0.17	59.30	60.06	—	
	1500	27.90	7.71	0.14	0.01	**90.20**	25.19 (-47.35)
incremental learning w/o L_{rem}	1000	74.19	66.32	56.57	50.87	—	
	1500	74.91	66.87	58.14	51.70	82.32	66.79 (-5.75)
incremental learning	1000	75.98	72.38	57.29	50.18	—	
	1500	**76.78**	**72.06**	**59.58**	53.07	78.94	**68.09 (-4.45)**
		Training Set 1		**Training Set 2**		**Training Set 3**	

by random vertical/horizontal flips, 0/90/180/270 degrees of rotations, and distorting their radiometry by random contrast change and gamma correction. Contrast of each channel in the image is changed as:

$$x_{curr} = (x_{prev} - \mu) * k + \mu, \tag{7}$$

where x_{prev} and μ are the pixel value and mean of all the pixels before the change, x_{curr} is the pixel value after the change, and k is the distortion factor, for which we generate a random value between 0.75 and 1.5. Gamma correction is formulated as:

$$x_{curr} = {x_{prev}}^{\gamma}, \tag{8}$$

where γ is the correction factor, which is drawn uniformly between 0.75 and 1.25. In Eqs. (7) and (8), we assume that the pixel values range between [0-1]. Fig. 6 illustrates the effect of gamma correction and the contrast change.

The overall F1-scores of all the classes on the Luxcarta dataset for each method are reported in Table III. The method, which achieves the most similar performance with *static learning* is highlighted. Fig. 7 depicts the change of IoU values on the validation cities as the training progresses. Visual close-up results for *static learning*, *multiple learning*, *incremental learning w/o rem* and *incremental learning* generated by the final models are shown in Fig. 8. Although our network generates a binary label map for each class, for the sake of compact and better visualization, we provide multi-class predicted maps obtained by assigning each pixel to the class, for which the highest probability is produced. In the figure, the pixels, having no probability higher than or equal to 0.5 are labeled as background.

As expected, *static learning* outperforms the other approaches on the Luxcarta dataset (see Table III), because in the training stage, we feed much more and diverse training data to the model compared to the other approaches. Although *static learning* is superior to the other approaches on the Luxcarta dataset, it is applicable only if the data are static and the annotations are unique, which is almost never the case in real-world applications. In *multiple learning*, even if the previous data are not accessible, predicted maps for all the presented classes can be generated. However, because of the growing number of classifiers, this approach is inefficient in terms of test efficiency and storage. In addition, for each individual classifier, learning is limited to the data, on which the classifier was initially trained. For instance, *building - high vegetation* classifier trained on Train1 can not be fine-tuned on Train2, as annotations for these classes are not available on Train2.

In *fixed representation* methodology, the exact performance for the initially introduced classes is retained as neither the shared nor the classification layers for these classes change. On the other hand, the network performs extremely poorly for the new classes as shown in Fig. 7c and reported in Table III. All in all, we conclude that shared layers of the network have to be adapted to the new training data. When we apply *fine-tuning*, since instead of initializing all the parameters randomly, the extracted features for the previous classes are used, performance for the new classes is remarkable, especially when there is only one class to be added. For instance, it is the best performer for *water* class. However, the results justify that the network catastrophically forgets the previously learned information.

As reported in Table III, *incremental learning* exhibits the closest performance to *static learning*. Since our approach enables the network to learn the old classes on the new data and remember them from the previous data, performance for the previous classes gets better over time. If the previous data are never shown, performance for the old classes may decrease as a result of adapting the network to the new data completely and imprecision of output of the memory network on the new data for the previous classes. Fig. 9 compares *incremental learning* and *incremental learning w/o* L_{rem} for *high vegetation* before and after adding *road* and *railway* classes on Train2 (i.e., before and after the 500^{th} epoch) to the *building* & *high vegetation* classifier trained on Train1. The close-ups from *Roanne* in Fig. 8 show that *incremental learning w/o* L_{rem} fails in detecting a lot of *high vegetation*, whereas *incremental learning* exhibits a good performance. We also observe that *incremental learning* significantly outperforms *multiple learning* for *building* class. The reason is that the network in *multiple learning* learns *building* only on Train1, while *incremental learning* facilitates learning the same class from all the training data sequentially. Although when buildings are small and regular shaped as in *Leibnitz* and *Roanne*, both approaches generate similar outputs, *multiple learning* is not able to delineate the borders very well when buildings cover a large area as in *Amstetten*. *Road* and *Railway* classes turn out to be the most difficult classes, as the numeric results for them are much lower than the others. As can be seen in the

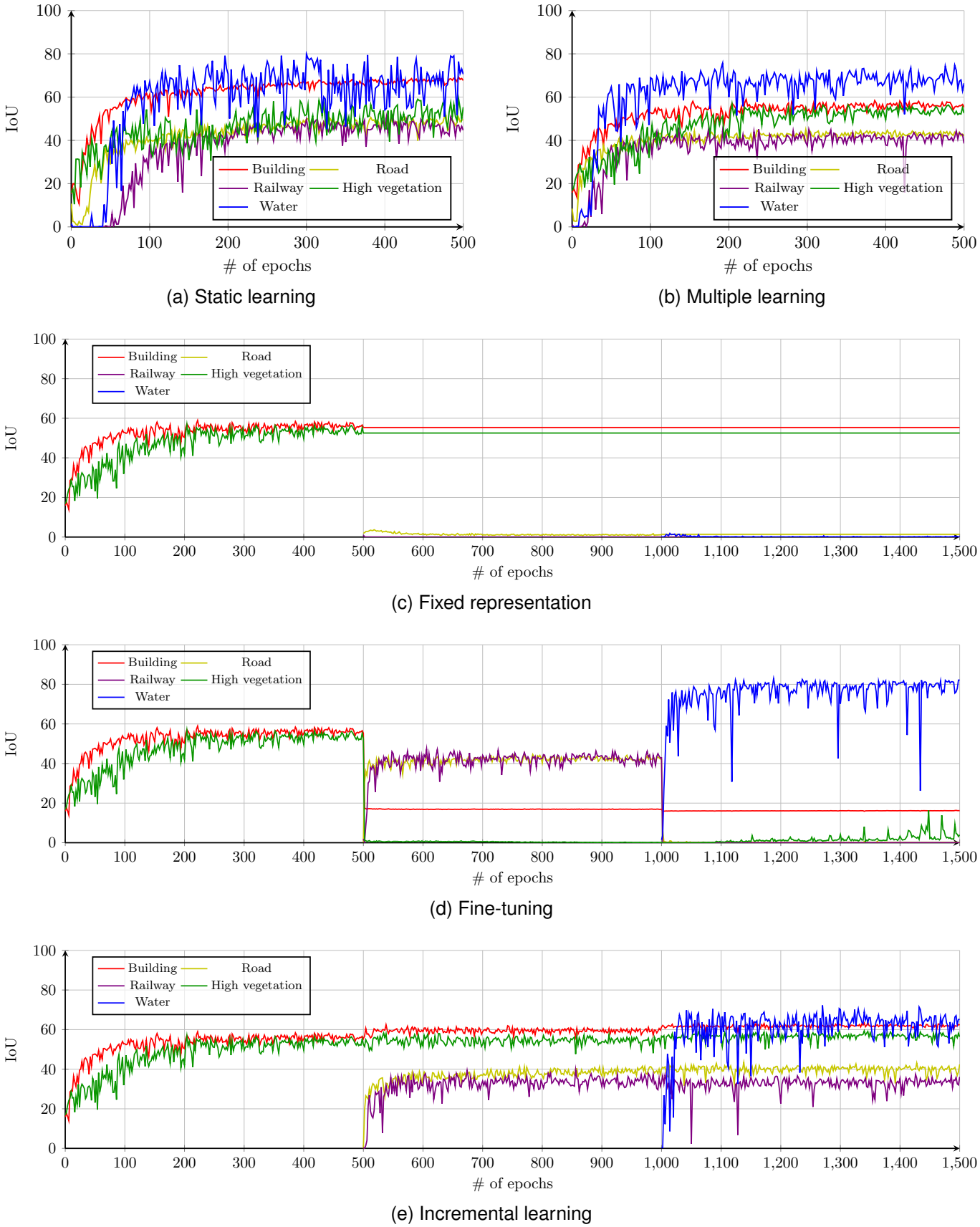


Fig. 7. Plots for the overall IoU values on the 4 validation cities of the Luxcarta dataset.

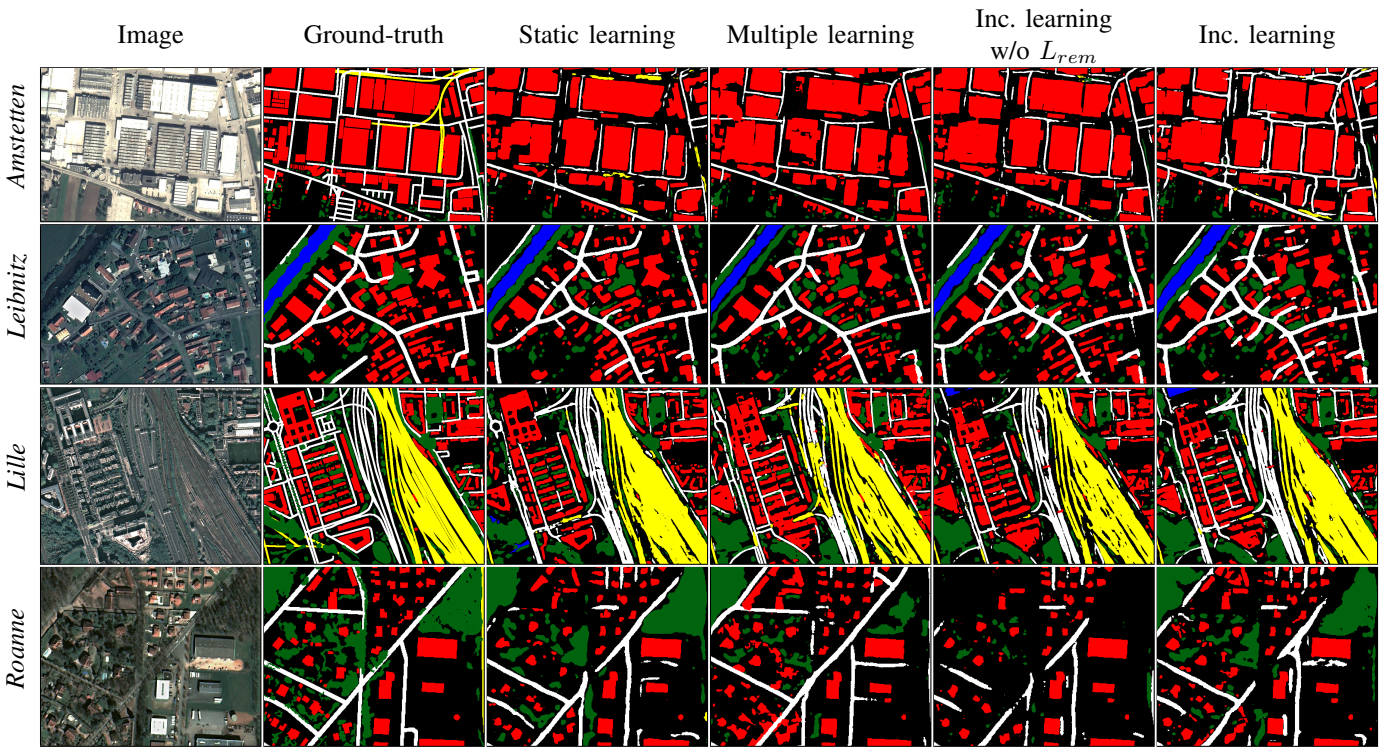


Fig. 8. Close-ups from validation cities of the Luxcarta dataset. Classes: background (black), building (red), road (white), railway (yellow), high vegetation (green), and water (blue).

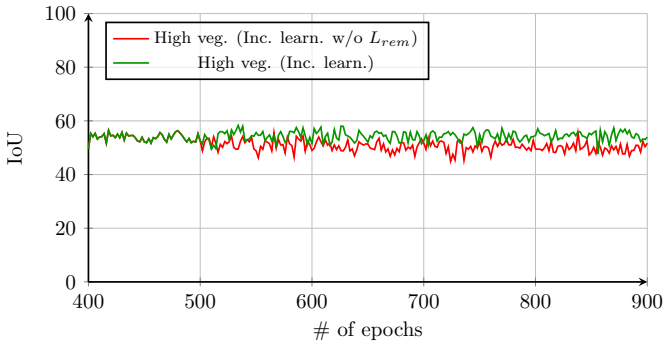


Fig. 9. Comparison of *incremental learning* and *incremental learning w/o* L_{rem} for *high vegetation* class.

close-up from *Lille*, they visually look quite similar, which makes the classifiers confuse between them in some cases. *Incremental learning* seems detecting the roads and railways that are mis-classified by *incremental learning w/o* L_{rem}.

D. Experiments on the Benchmark Datasets

In the experiments on the benchmarks, we assume that we have access to the whole training tiles, but we are provided the annotations sequentially. We suppose that every time a new set of annotations are retrieved, the previous one is not accessible. On the Vaihingen dataset, we consider that we retrieve label maps for *building* and *high vegetation* classes in the beginning. We are then given the ground-truth for *impervious surfaces* and *low vegetation*. Finally, we receive the annotations for *car* class. On the Potsdam dataset, since there is an additional *clutter* class, we assume that the label map for this class is also available in the initial training data. For our approach, since we always use the same training images, we remind the network the old classes using output of the memory network (i.e., we only optimize L_{adapt}). On contrary the other approaches, for *static learning*, we use all training tiles as well as annotations for all the classes at once in the training stage.

Because the images in the benchmarks are of much higher resolution than the satellite images in the Luxcarta dataset, the patches need to be larger to cover a reasonable area. Therefore, we divide the training tiles into 512 $\times$ 512 patches. The validation tiles are split into 2000 $\times$ 2000 patches. The training and validation tiles have 64 $\times$ 64 and 120 $\times$ 120 pixels of overlap, respectively. We compute a global mean for each channel from the training tiles and subtract it from all the pixels.

For each approach, we train the same number of models for the same number of epochs and iterations using the same optimizer with the same parameters as in the experiments on the Luxcarta dataset. As size of the training patches is larger than in the previous experiments, we randomly sample 8 patches instead of 12. Another difference is that since both training and validation patches are from the same city, we augment the patches by only random flips and rotations.

We present the qualitative and quantitative experimental results on the benchmarks in a similar way described in Sec. III-C. We report F1-score for each class in Tables IV and V, illustrate the plots of IoU vs. number of epochs on the Vaihingen benchmark in Fig. 10, and show close-ups from both benchmarks in Fig. 11. As we use all the annotations at once for *static learning*, we again choose this approach as the

TABLE IV
F1 SCORES ON THE VAIHINGEN BENCHMARK DATASET.

Method	Epoch	Building	High veg.	Imper. surf.	Low veg.	Car	Overall
static learning	**500**	93.61 (Ref.)	87.87 (Ref.)	91.55 (Ref.)	81.05 (Ref.)	82.83 (Ref.)	87.38 (Ref.)
multiple learning	**500**	**94.43**	**88.12**	90.71	80.41	**87.90**	**88.31 (+0.93)**
fixed representation	1000	94.43	88.12	87.09	76.39	—	
	1500	**94.43**	**88.12**	87.09	76.39	13.37	71.88 (-15.5)
fine-tuning	1000	52.40	0.03	91.83	80.99	—	
	1500	0.02	0.00	43.81	0.01	86.18	26.00 (-61.38)
incremental learning w/o L_{rem}	1000	94.34	88.02	91.42	81.65	—	
	1500	94.31	88.07	**91.51**	**81.60**	81.69	87.44 (+0.06)
		Training Set 1		**Training Set 2**		**Training Set 3**	

TABLE V
F1 SCORES ON THE POSTDAM BENCHMARK DATASET.

Method	Epoch	Building	High veg.	Clutter	Imper. surf.	Low veg.	Car	Overall
static learning	**500**	96.83 (Ref.)	85.04 (Ref.)	54.57 (Ref.)	92.62 (Ref.)	85.69 (Ref.)	94.84 (Ref.)	84.93 (Ref.)
multiple learning	**500**	96.59	85.25	50.82	92.07	84.82	**95.36**	84.15 (-0.78)
fixed representation	1000	96.59	85.25	50.82	86.76	79.98	—	
	1500	96.59	85.25	50.82	86.76	79.98	72.14	78.59 (-6.34)
fine-tuning	1000	0.00	44.53	3.23	92.13	85.45	—	
	1500	1.62	24.73	0.00	65.00	0.01	94.60	30.99 (-54.94)
incremental learning w/o L_{rem}	1000	96.91	86.12	50.23	92.20	85.64	—	
	1500	**96.86**	**85.28**	**51.56**	**92.10**	**85.28**	94.43	**84.25 (-0.68)**
		Training Set 1			**Training Set 2**		**Training Set 3**	

reference method.

From the plots in Figs. 7 and 10, our first observation is that IoU values for each model, as the training iterations continue, fluctuate much more on the Luxcarta dataset than on the Vaihingen benchmark. We also observe that models, trained from the Vaihingen dataset converge faster. The reason for these two conclusions is that in the Vahingen dataset, a single aerial image was split into smaller tiles, while images in the Luxcarta dataset were taken from different cities at different dates; therefore, they have distinct color variations and visual features. Furthermore, the Luxcarta images are of much lower resolution, and the validation set consists of the cities that are not seen by the network during training. Because of all these reasons, accuracies for the same classes (i.e., *building* and *high vegetation*) are significantly lower in the experiments on the Luxcarta dataset than on the benchmarks.

Our observation for *fixed representation* and *fine-tuning* is similar to the experiments on the Luxcarta dataset. As can be seen in Fig. 10c, for *fixed representation*, although some classes such as *impervious surface* and *low vegetation* can be learned relatively well, the network performs poorly if the newly added class represents small objects like *car*.

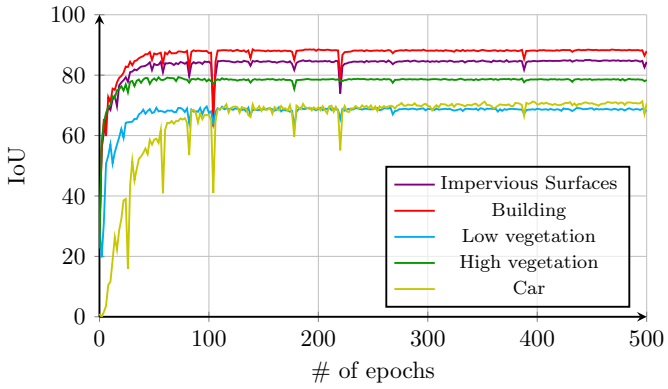
Since training as well as test tiles are from the same city, output of the memory network becomes almost the ground-truth for the previous classes. As a result, even if annotations for the previous classes are not accessible, new classes can be learned while exhibiting a similar performance for the former classes. We justify this claim in Fig. 10e, in which it is demonstrated that IoU plots for the previously learned classes remain quite flat over time. The predicted maps of the close-ups from Vaihingen in Fig. 11 for 3 approaches look very similar. The advantage of our approach is that with the help of the features for the previous classes, the network converges very fast for the new classes. For instance, as illustrated in Fig. 10a, it takes roughly 50 epochs in order for the network to converge for *low vegetation* class when *static learning* is applied, whereas with the proposed approach, a similar accuracy for the same class can be achieved in only a few epochs, as depicted in Fig. 10e.

In this experimental setup, if the classes have distinct visual appearance and features like in the Vaihingen benchmark, as the classification tasks are shared between several classifiers, *multiple learning* performs better especially when the class has a low number of samples such as *car*. As training tiles of the Potsdam dataset contain the nDSM data, detecting *car* class is easier on this dataset than on the Vaihingen benchmark. As reported in Table V, the gap between *multiple learning* and the other approaches is smaller for this class. On the contrary, as can be seen in the last row in Fig. 11, *clutter* class has high visual similarities with some pixels labeled as *impervious surfaces* or *low vegetation*. Hence, a single classifier that is trained jointly for all the classes, performs better in distinguishing these classes. Unlike *multiple learning*, where several isolated classifiers are trained, our approach allows joint training via the memory network. Therefore, our approach performs better for these classes, as confirmed by Table V. The last row in Fig. 11 exemplifies some misclassified *clutter* pixels by *multiple learning* but correctly detected by our approach.

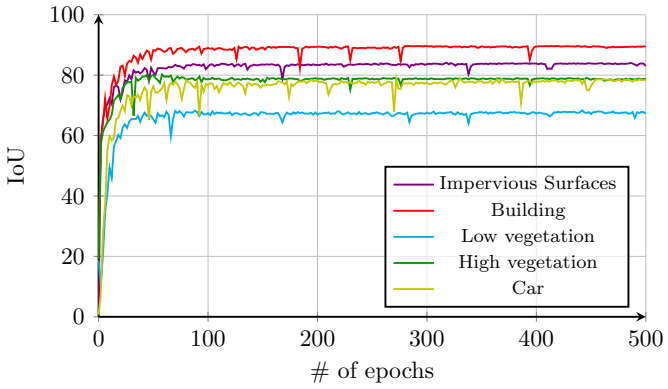
E. Running Times

We have implemented all the approaches in Tensorflow[2], and conducted all the experiments on an Nvidia Geforce GTX1080 Ti GPU with 11 GB of RAM. Table VI reports the training times for *incremental learning*, *fixed representation*, and the others. Let us remark that the training times for

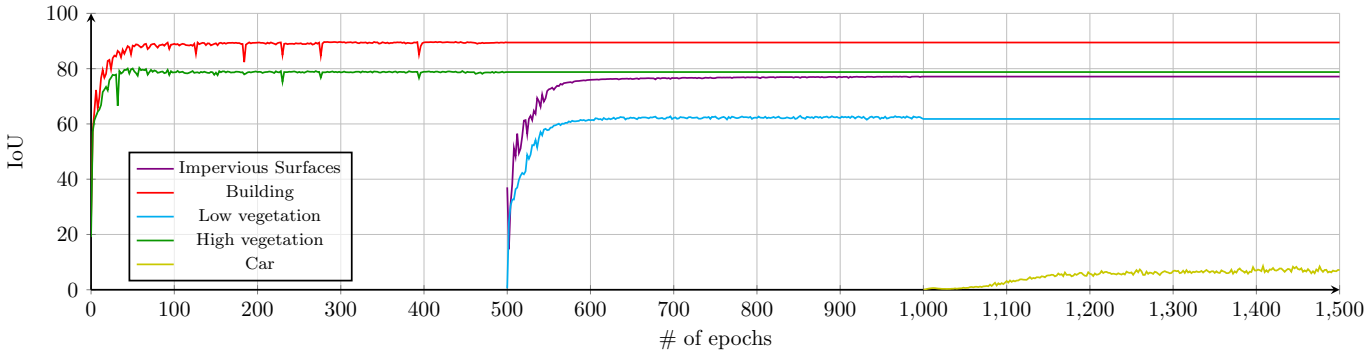
[2]https://www.tensorflow.org



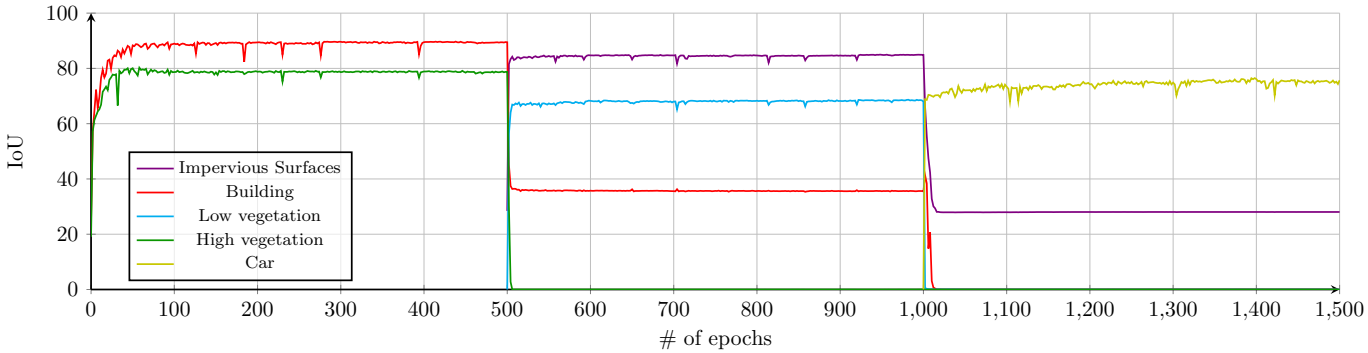
(a) Static learning



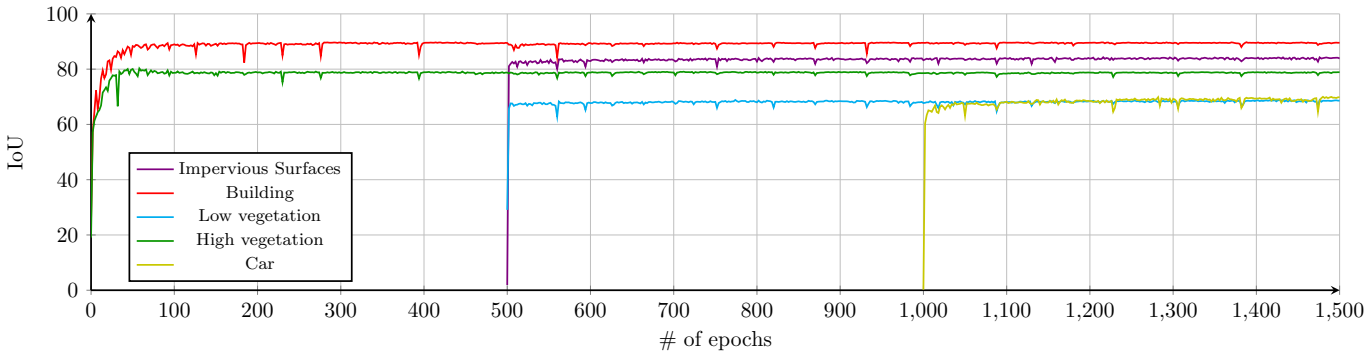
(b) Multiple learning



(c) Fixed representation



(d) Fine-tuning



(e) Incremental learning w/o L_{rem}

Fig. 10. Plots for the overall IoU values on validation data of the Vaihingen benchmark dataset.

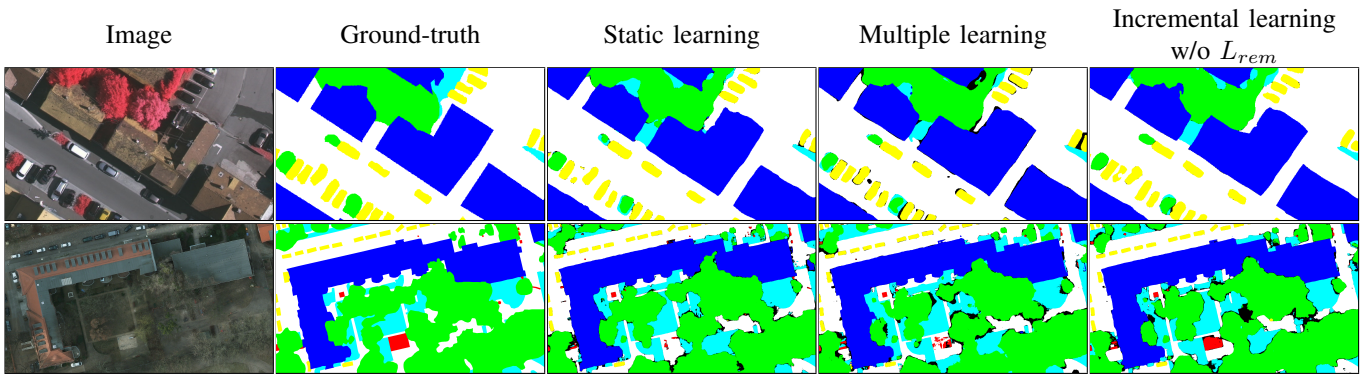


Fig. 11. Close-ups from the benchmarks. Classes: background (black), impervious surfaces (white), building (blue), low vegetation (cyan), high vegetation (green), car (yellow), and clutter (red). The results in the upper row are from the Vaihingen dataset, and the bottom row are from the Potsdam benchmark.

TABLE VI
TRAINING TIMES FOR EACH APPROACH

Method	Patch Size	Batch Size	Time for 1 Iter. (seconds)
Inc. learn.	384	12	1.03
Fixed. Rep.	384	12	0.31
The Others	384	12	0.72
Inc. learn.	512	8	1.21
Fixed. Rep.	512	8	0.32
The Others	512	8	0.87

fine-tuning, *multiple learning*, and *static learning* is almost the same; therefore, we refer to these approaches as *the others* and report the average of their training time in the table. Since we optimize only the classification layer for *fixed representation*, its training is extremely fast. For *incremental learning*, the current training patch needs to be segmented by the memory network for the knowledge transfer. Hence, the training time in each iteration for *incremental learning* is about 0.3 seconds longer than for *fine-tuning*, *multiple learning*, and *static learning*.

IV. Concluding Remarks

We proposed a novel incremental learning methodology, which enables the neural network to learn segmentation capabilities for the new classes while retaining dense labeling abilities for the formerly trained classes without using the entire previous training data.

In our experiments, we first showed that the common learning approaches are extremely inefficient or inapplicable to learn from streaming data. We then demonstrated why using only the features extracted for the previous classes is inefficient to learn new classes. We also provided the results, showing that when the network is trained using only the new data without having any regularization, the learned information for the previous classes is catastrophically forgotten. Finally, on three different datasets we proved that the proposed approach achieves a high performance for the new classes without forgetting the old classes.

As the future work, we plan to explore how to incorporate domain adaptation techniques to our incremental learning methodology so that the trained network could better generalize to the data collected from new geographic locations.

Acknowledgment

The authors thank ACRI-ST and CNES for initializing and funding this study.

References

[1] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "High-resolution semantic labeling with convolutional neural networks," *IEEE TGRS*, vol. 55/12, 2017.

[2] M. Volpi and D. Tuia, "Dense semantic labeling of subdecimeter resolution images with convolutional neural networks," *IEEE TGRS*, vol. 55/2, 2017.

[3] N. Audebert, B. Le Saux, and S. Lefèvre, "Semantic segmentation of earth observation data using multimodal and multi-scale deep networks," in *ACCV*. Springer, 2016, pp. 180–196.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*. Springer, 2015, pp. 234–241.

[5] B. Huang, K. Lu, N. Audebert, A. Khalel, Y. Tarabalka, J. Malof, A. Boulch, B. Le Saux, L. Collins, K. Bradbury *et al.*, "Large-scale semantic classification: outcome of the first year of inria aerial image labeling benchmark," in *IEEE IGARSS*, 2018.

[6] V. Iglovikov and A. Shvets, "Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation," *arXiv*, 2018.

[7] V. Iglovikov, S. Seferbekov, A. Buslaev, M. R. Center, and A. Shvets, "Ternausnetv2: Fully convolutional network for instance segmentation," *arXiv*, 2018.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv*, 2014.

[9] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *IEEE IGARSS*, 2017.

[10] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[11] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv*, 2013.

[12] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *Advances in neural information processing systems*, 1996, pp. 640–646.

[13] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on systems, man, and cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.

[14] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Advances in neural information processing systems*, 2001, pp. 409–415.

[15] L. Bruzzone and D. F. Prieto, "An incremental-learning neural network for the classification of remote-sensing images," *Pattern Recognition Letters*, vol. 20, no. 11-13, pp. 1241–1248, 1999.

[16] S. S. Sarwar, A. Ankit, and K. Roy, "Incremental learning in deep convolutional neural networks using partial network sharing," *arXiv*, 2017.

[17] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, "Error-driven incremental learning in deep convolutional neural network for large-scale image classification," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 177–186.

[18] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv*, 2016.

[19] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," 2018.

[20] J. Xu and Z. Zhu, "Reinforced continual learning," *arXiv*, 2018.

[21] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *CVPR*, 2017.

[22] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 6467–6476.

[23] S. Hou, X. Pan, C. Change Loy, Z. Wang, and D. Lin, "Lifelong learning via progressive distillation and retrospection," in *ECCV*, 2018, pp. 437–452.

[24] Y. Li, Z. Li, L. Ding, P. Yang, Y. Hu, W. Chen, and X. Gao, "Supportnet: solving catastrophic forgetting in class incremental learning with support data," *arXiv*, 2018.

[25] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, Z. Zhang, and Y. Fu, "Incremental classifier learning with generative adversarial networks," *arXiv*, 2018.

[26] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 2990–2999.

[27] H. J. Sussmann, "Uniqueness of the weights for minimal feedforward nets with a given input-output map." *Neural networks*, vol. 5, no. 4, pp. 589–593, 1992.

[28] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, p. 201611835, 2017.

[29] X. Liu, M. Masana, L. Herranz, J. Van de Weijer, A. M. Lopez, and A. D. Bagdanov, "Rotate your networks: Better weight consolidation and less catastrophic forgetting," *arXiv*, 2018.

[30] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," *arXiv*, 2017.

[31] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," *arXiv*, 2018.

[32] A. R. Triki, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," *arXiv*, 2017.

[33] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in Neural Information Processing Systems*, 2017, pp. 4652–4662.

[34] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," *arXiv*, 2017.

[35] ——, "Piggyback: Adding multiple tasks to a single, fixed network by learning to mask," *arXiv*, 2018.

[36] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv*, 2017.

[37] C. Zeno, I. Golan, E. Hoffer, and D. Soudry, "Bayesian gradient descent: Online variational bayes learning with increased robustness to catastrophic forgetting and weight pruning," *arXiv*, 2018.

[38] H. Ritter, A. Botev, and D. Barber, "Online structured laplace approximations for overcoming catastrophic forgetting," *arXiv*, 2018.

[39] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, "Variational continual learning," *arXiv*, 2017.

[40] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv*, 2015.

[41] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE TPAMI*, 2017.

[42] T. Furlanello, J. Zhao, A. M. Saxe, L. Itti, and B. S. Tjan, "Active long term memory networks," *arXiv*, 2016.

[43] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," *arXiv*, 2018.

[44] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *ICCV*, 2017, pp. 3420–3429.

[45] ISPRS. 2d semantic labeling contest. [Online]. Available: http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html

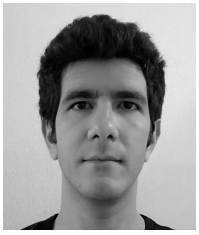
[46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[47] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[48] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv*, 2017.

[49] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[50] G. Csurka, D. Larlus, F. Perronnin, and F. Meylan, "What is a good evaluation measure for semantic segmentation?" in *BMVC*, vol. 27. Citeseer, 2013, p. 2013.



Onur Tasar (S'18) received the B.S. degree from Hacettepe University, Ankara, Turkey, in 2014, and the M.S. degree from Bilkent University, Ankara,Turkey, in 2017, both in computer engineering. He is currently working toward the Ph.D. degree with the TITANE team of Inria Centre de Recherche Sophia Antipolis Méditerranée, Sophia Antipolis, France.

His research interests include computer vision, machine learning, and computational geometry with applications to remote sensing

Yuliya Tarabalka (S'08 – M'10 – SM'18) received the B.S. degree in computer science from Ternopil IvanPul'uj State Technical University, Ternopil, Ukraine, in 2005, the M.Sc. degree in signal and image processing from the Grenoble Institute of Technology (INPG), Grenoble, France, in 2007, and the joint Ph.D. degree in signal and image processing from INPG and in electrical engineering from the University of Iceland, Reykjavik, Iceland, in 2010.

From July 2007 to January 2008, she was a Researcher with the Norwegian Defense Research Establishment, Kjeller, Norway. From September 2010 to December 2011, she was a Postdoctoral Research Fellow with the Computational and Information Sciences and Technology Office, NASA Goddard Space Flight Center, Greenbelt, MD, USA. From January to August 2012, she was a Postdoctoral Research Fellow with the French Space Agency (CNES) and Inria Centre de Recherche Sophia Antipolis Méditerranée, Sophia Antipolis, France. From 2012 to 2019, she was a Researcher with the TITANE team of Inria Centre de Recherche Sophia Antipolis Méditerranée. She is currently the Research Director of LuxCarta Technology, Mouans-Sartoux, France. Her research interests include image processing, pattern recognition, and development of efficient algorithms.

Pierre Alliez received the Ph.D. degree from Telecom ParisTech, Paris, in 2000.

He was a Postdoctoral Fellow at the Universityof Southern California in 2001. He is a Senior Researcher and Team Leader with the TITANE team of Inria Centre de Recherche Sophia Antipolis Méditerranée, Sophia Antipolis, France. He has authored scientific publications and several book chapters on mesh compression, surface reconstruction, mesh generation, surface remeshing, and mesh parameterization.

Dr. Alliez was the recipient of the Eurographics Young Researcher Award for his contributions to computer graphics and geometry processing in 2005. He was a Co-Chair of the Symposium on Geometry Processing in 2008, of Pacific Graphics in 2010, and of Geometric Modeling and Processing in 2014. In 2011, he was awarded a Starting Grant from the European Research Council on Robust Geometry Processing. He is an Associate Editor for the *ACM Transactions on Graphics*.