SCR: Smooth Contour Regression with Geometric Priors
Gaétan Bahl, Lionel Daniel, Florent Lafarge

To cite this version:
Gaétan Bahl, Lionel Daniel, Florent Lafarge. SCR: Smooth Contour Regression with Geometric Priors. 2022. hal-03889605

HAL Id: hal-03889605
https://inria.hal.science/hal-03889605
Preprint submitted on 8 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
While object detection methods traditionally make use of pixel-level masks or bounding boxes, alternative representations such as polygons or active contours have recently emerged. Among them, methods based on the regression of Fourier or Chebyshev coefficients have shown high potential on freeform objects. By defining object shapes as polar functions, they are however limited to star-shaped domains. We address this issue with SCR: a method that captures resolution-free object contours as complex periodic functions. The method offers a good compromise between accuracy and compactness thanks to the design of efficient geometric shape priors. We benchmark SCR on the popular COCO 2017 instance segmentation dataset, and show its competitiveness against existing algorithms in the field. In addition, we design a compact version of our network, which we benchmark on embedded hardware with a wide range of power targets, achieving up to real-time performance.
make use of Chebyshev or Fourier coefficients to explicitly regress an encoding that represents the shape of the object.

Our work proposes a Fourier shape decoder that brings several important improvements over previous methods. First, we give insights and mathematical arguments about the fact that Fourier coefficients constitute a better representation than the Chebyshev coefficients used in [37]. By careful design of the loss function, we also improve the visual quality of regressed contours with fewer Fourier coefficients, while also simplifying the training process. Moving to a complex representation for contours allows us half the number of calls to the Inverse Fast Fourier Transform (IFFT) in the final shape decoding stage, while allowing more freedom in shape representation by alleviating the limits of polar coordinates-based methods. Finally, we propose a compact version of our architecture based on a lightweight backbone [26] for use on low-power systems, typically found on board satellites and UAVs, targeting applications such as real time on-board instance segmentation and efficient transmission within constellations of satellites.

2. Related Work

Detection neural networks R-CNN [7] and its follow-ups [6, 27], were the original neural networks for object detection. They are two-stage approaches, where object locations are first proposed by a Region Proposal Network (stage one) and then classified by a subsequent CNN (stage two). Later, fully convolutional single-stage approaches were proposed [18, 21, 25], which were significantly faster than their two-stage counterparts. These methods rely on a set of “anchor” bounding boxes, to which the detected boxes are assigned and regressed relatively. The simplicity and efficiency of these single stage detectors have made them very popular over the years. A large number of improvements to this design have been proposed, such as the Feature Pyramid Network (FPN) [17] and subsequent work [5, 29, 32], which aggregates features from the backbone at different levels in order to create semantically rich feature maps at each level. Recently, FCOS [30], an anchor-free detection neural network architecture, has been developed. By leveraging the pyramid structure given by the FPN, FCOS removes the problem of anchor box assignment, and thus streamlines the object detection task with a one-stage, proposal-free, anchor-free framework.

Instance segmentation neural networks Instance segmentation neural networks were pioneered by Mask-RCNN [8], a two-stage approach based on [7]. A significant number of architectures have been developed [2, 3, 20, 33, 34], each bringing incremental improvements in popular benchmarks. Some methods save storage space by regressing masks as low-dimensional embeddings, which are decoded using either a learned decoder [11, 38] which complexes the model and its training, or fixed functions such as the Discrete Cosine Transform [28], in which case the encoding dimension is still quite large. Nonetheless, mask-based instance segmentation methods all encounter the drawbacks of the pixel-based representation, and are generally much slower than their bounding box-based counterparts.

Contour regression methods Recently, the community has explored new ways of regressing the boundaries of objects. These methods aim to alleviate the aforementioned issues of the pixel-based mask representation, while still offering better IoU with detected objects than bounding box based architectures, which most of them take as a starting point.

Snake-based methods [22, 23] represent shapes as polygons in pixel coordinates and deform them iteratively using circular convolutions. While these methods can reach good accuracy levels at reasonable speeds, they generally make use of a high number of contour points, which can be costly in terms of storage on embedded devices, especially compared to shape encoding methods described below.

ESE-Seg [37] is based on the YOLOv3 detection neural network [26] and regresses an explicit shape encoding for each detected object in the form of either Chebyshev or Fourier coefficients and a center. The contour of the object is modeled as a function $\rho(\theta), \theta \in [0, 2\pi]$, revolving around this center, as shown on the inset Figure. Corresponding ground truth coefficients have to be generated for each object of the training dataset. When used with Chebyshev coefficients, this representation is however not well suited to periodic functions, as we will demonstrate later. Moreover, this method is only able to fully represent star-shaped domains and thus cannot be used to regress less regular shapes.

In PolarMask [35], the authors modify the FCOS [30] architecture in order to regress the polar coordinates of the contour points directly, i.e. the values of the function $\rho$. Again, specific ground truth labels have to be created for each object, with a fixed number of rays cast from its center, and only star-shaped objects can be regressed, since only one coordinate is regressed for each ray.

FourierNet [31], also based on FCOS, simplifies the contour regression problem by using the Inverse Fast Fourier Transform (IFFT) as a differentiable shape decoder. Thus, the neural network regresses Fourier coefficients, but the output is directly compared to ground truth polygons. The authors use either polar or Cartesian coordinates. In
the first case, one set of complex coefficients is used to find $\rho$, and the regressed shapes have to be star-shaped. In the Cartesian case, the contour is seen as two functions, $X(t)$ and $Y(t)$ as shown on the inset Figure. Two sets of complex coefficients are used (one for each) to model these functions. While the authors manage to get good looking results in the polar case, their Cartesian version does not perform as well quantitatively, and fails to produce coherent shapes when using a high number of Fourier coefficients.

### Contours as a complex function, and Fourier series

This drawback of Chebyshev polynomials steers us toward the Fourier decomposition, which is better suited to our purposes simply due to its conception, since any set of Fourier coefficients will always represent a continuous smooth periodic function.

Our single complex function representation has several advantages over using two real functions as in [31]: not only does it halve the number of IFFT calls in the shape decoding stage; it also ties these coefficients together as they represent a single function, which might help during the training of the neural network, as backpropagation occurs through a single IFFT call.

In order to evaluate how many complex coefficients are needed to represent shapes accurately, we interpolate all polygons of the COCO 2017 validation set using the scheme detailed in section 3, we then compute the FFT and its inverse after zeroing a certain number of coefficients, and finally compare the result to the original polygons using the Chamfer Distance as reconstruction error. The inset figure shows the quickly diminishing returns of increasing the number of coefficients, highlighting the fact that only a small number of them suffice to represent a shape accurately. We decide to run our experiments using 8 to 32 complex coefficients.

### SCR head

The SCR head, shown in Figure 2, is based on the FourierNet [31] head. For both the class and contour regression branches, we make use of 3 deformable convolutions [39] with shared weights among feature levels. Thanks to the use of the complex contour representation, we only require one call to the IFFT for shape decoding. We add two regularization losses which we will detail further below. The first one is applied directly to the regressed coefficients. The second one is applied to the decoded point returns of increasing the number of coefficients, the quickly diminishing returns of increasing the number of coefficients, highlighting the fact that only a small number of them suffice to represent a shape accurately. We decide to run our experiments using 8 to 32 complex coefficients.

### Loss function

For training SCR, our loss function is defined as a sum of different terms:

$$\mathcal{L} = \mathcal{L}_{\text{CD}} + \mathcal{L}_{\text{cent}} + \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{perm}} + \mathcal{L}_{\text{coeff}}$$  \hspace{1cm} (1)$$

where $\mathcal{L}_{\text{CD}}$ is a polygon regression loss based on a symmetrized Chamfer Distance (similarly to [31]), $\mathcal{L}_{\text{cent}}$ is the centerness loss from FCOS [30] (binary cross-entropy), and

---

### Method

Our network architecture is based on the FCOS detection neural network [30]. Like most recent detection architectures, FCOS is composed of a backbone, a neck (feature pyramid) and several detection heads with shared weights. As backbones, we use ResNet-50 [9], ResNetXt-101 [36] and the lightweight DarkNet-53 [26], in order to address different performance targets. We run experiments with both the classical FPN neck from [17], as well as the more recent FPN-CARAFAE [32], which provides a small accuracy improvement.

#### Choice of shape representation

In [31,35,37], the shape of an object is represented as a function $\rho(\theta)$ with $\theta \in [0, 2\pi]$, and thus revolve around a center, which is either regressed along with the coefficients in [37], or is the center of output “cells” from FCOS in [31,35]. $\rho$ is a $2\pi$-periodic function measuring the distance of each point of a contour to this center. By contrast, we choose to model the shape of objects with a more generic complex periodic function of the form $C(t) = X(t) + iY(t)$ with $t \in [0, 1]$.

#### On Chebyshev coefficients

The use of Chebyshev coefficients [37] is not well suited to the regression of continuous periodic functions through a neural network - in this case, the $\rho$ function. Indeed, we demonstrate in the appendix that $\rho$ is continuous periodic, i.e. $\rho(0) = \rho(2\pi)$, if and only if the regressed Chebyshev coefficients $\alpha_k$ satisfy $\sum_k 0 \alpha_{2k+1} = 0$. It is unlikely that a neural network can learn this exact relation, even if optimized as a loss during training. Hence, discontinuities can be observed in nearly every shape regressed in this manner, as seen in Figure 11 of [37] reproduced in appendix. A similar relationship can be found using the derivatives of Chebyshev polynomials if we also want shapes to be smooth, which should be the case for most objects.
\[ L_{\text{cls}} \] is the Focal Loss for classification from RetinaNet [18]. \( L_{\text{perim}} \) and \( L_{\text{coeff}} \) are regularization terms on the shape perimeter and on Fourier coefficient respectively. Both are detailed below.

In contrast to [31], note that we do not use any bounding box loss function as bounding boxes can be inferred from the regressed polygons and do not need to be learned separately.

**L2 Perimeter regularization** The use of the Chamfer distance loss as the main polygon regression loss has a major drawback that needs to be addressed: since ground truth points are only compared to the closest regressed point, the network does not necessarily learn to regress contours that go around the object only once. Indeed, circling the object multiple times is not penalized in \( L_{\text{CD}} \). One way to alleviate this problem, as done in [31], is to first perform a warm-up of the shape regression with an \( L^1 \) loss, so that each output point is already tied to specific ground truth points. However, after being trained with Chamfer loss, their network still tries to regress shapes in an overly complex way, which creates lots of self-intersections in the output polygon, as seen in our comparison in section 4 (Figure 4).

We instead choose to apply a \( L^2 \) perimeter-based regularization to every output contour, even the ones that have not been assigned to any object. This, while also simplifying the training process, encourages simpler shapes that go around the object once and have a minimal number of self-intersections:

\[
L_{\text{perim}} = \lambda_{\text{perim}} \sqrt{\sum_{i=0}^{N} (x_i - x_{i+1})^2}
\]  

(2)

The \( \lambda_{\text{perim}} \) coefficient can be set to a relatively low number in order to avoid influencing training too much. Setting it too high will result in small round-ish shapes. It can be decayed over time or set to zero after \( L_{\text{perim}} \) reaches a steady state, in order to retrieve more complex shapes while retaining the effect of regularization, as shown in our experimental results (Figure 3).

**L1 Fourier coefficient regularization** While representing contours using Fourier coefficients is already more storage-efficient than using masks, we may want to further reduce the storage space needs by ignoring very small coefficients (i.e. setting them to zero after regression). Thus, it would be interesting to enforce sparsity in the coefficients, in order to have as many of them be close to zero as possible.

In order to favor sparsity of the Fourier coefficients \( F_i \), we use an \( L^1 \) regularization term:

\[
L_{\text{coeff}} = \frac{\lambda_{\text{coeff}}}{N_c} \sum_{i=-n, i \notin \{-1,0,1\}}^{n} |F_i|
\]  

(3)

Where \( \lambda_{\text{coeff}} \) is an adjustable parameter (500.0 in our experiments). It has to be noted that we do not apply this penalty to the -1, 0 and 1 frequency coefficients, since these coefficients will have to be quite large in most cases. Having lots of Fourier coefficients close to zero, especially high
frequency ones, also helps the network create simpler and smoother contours, as shown in section 4 (Figure 5).

**Polygon ground truth processing** Existing works [31, 35, 37] typically convert the ground truth polygons to polar coordinates with respect to a center. This is done by casting a number of rays at regular angular intervals from this center to the farthest contour point. This process can hardly be done on-the-fly without slowing the data loading pipeline. We also note that the distance between two adjacent points created in this manner can vary greatly (e.g. in oblong objects). We believe that having a high number of evenly spaced out ground truth points is better for the stability of the Chamfer Distance loss. Thus, our interpolation is done in a “constant spacing” fashion for each ground truth polygon. Since the original ground truth polygons vary in number of points, we interpolate them such that the number of ground truth points \( N_p \) matches the number of regressed points. This is easily done on the fly in the data loading pipeline and does not incur any performance loss.

### 4. Experiments

We implement our ResNet-50, ResNeXt-101 and DarkNet-53 based models using the MMDetection 2.7.0 object detection toolbox [4], based on MMCV 1.2.1 for easy and fair comparison with other methods. The underlying framework is PyTorch 1.7.0 with CUDA 10.1. We make use of the updated FFT package from PyTorch 1.7.0 which allows the use of complex numbers. We use the Chamfer Distance implementation from PyTorch3D 0.3.0 [24]. The interpolation scheme described in 3 is implemented as a module in the MMDetection data loading pipeline using the “interp” module from Numpy. Notably, due to software incompatibilities during the conversion of our DarkNet-53 model for low-power hardware targets, we had to simplify the architecture by removing deformable convolutions and CARAFE. In order to evaluate our method and easily compare it to previous works, we choose to work on the widely used COCO 2017 Instance Segmentation dataset [19].

Our networks are trained for 38 epochs using SGD with momentum (0.9), with an initial learning rate of 0.0003, batch-size of 16 (2 images per GPU), gradient clipping at a maximum \( L^2 \) norm of 45, and employ of a 500 step warm-up with a ratio of 0.001 applied to the learning rate. We use the multiscale training feature found in MMDetection to train with image heights of 640 and 800 pixels. We balanced our loss function terms through trial-and-error, and settled on the following values: \( \lambda_{cls}=1.0, \lambda_{cent}=1.0, \lambda_{CD}=1.0, \lambda_{perim}=0.01, \lambda_{coeff}=500.0. \)

<table>
<thead>
<tr>
<th>Name-Backbone</th>
<th>DCNv2</th>
<th>CARAFE</th>
<th>( N_{feats} )</th>
<th>( N_{feats} )</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCR-D53</td>
<td>No</td>
<td>No</td>
<td>128</td>
<td>128</td>
<td>20h</td>
</tr>
<tr>
<td>SCR-R50</td>
<td>Yes</td>
<td>Yes</td>
<td>256</td>
<td>256</td>
<td>40h</td>
</tr>
<tr>
<td>SCR-X101</td>
<td>Yes</td>
<td>Yes</td>
<td>256</td>
<td>256</td>
<td>80h</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of our main networks. Variations are further explored in our ablation study. DCN refers to the use of Deformable Convolution v2 [39] layers in the head. CARAFE refers to the use of the FPN-CARAFE [32] neck. Training time on 8 Tesla V100 GPUs.

**Qualitative results** Figure 1 shows a selection of results from the COCO 2017 test-dev dataset, computed using our smallest DarkNet-53 based network. Our method properly regresses smooth artifact-free contours in these scenes varying in number of objects, classes and shapes. In particular, non-star-shaped objects can be accurately represented. An extended selection of results is provided in appendix.

**Quantitative comparison** We compare our method to existing shape encoding methods [31, 35, 37], as well as state-of-the-art snake-based [22, 23] and mask-based methods [8, 20, 34]. The evaluation is done using three metrics: mAP for accuracy (as defined in [19]), FPS (frames per second) reported on a single Nvidia GTX 1080Ti GPU for speed and memory usage per regressed object in bits, which we call SEC (for Shape Encoding Complexity). Run time and object size in memory or storage are important factors for embedded hardware applications, such as real-time on-board change detection. Thus, we propose OES (Overall Efficiency Score), which is simply the multiplication of these three scores and thus defined as \( OES = 100 \times mAP \times FPS \times SEC^{-1}. \) This metric represents the trade-off between having slow but accurate models outputting detailed shapes, and having faster models outputting simpler shapes.

The comparisons against other shape encoding methods are shown in Table 2 for the ResNeXt-101 backbone and 3 for the DarkNet-53 backbone. We observe that with the same backbone and at a similar number of coefficients, our method achieves good accuracy compared to other shape encoding methods.

In Table 2, our method has a competitive efficiency score (OES). While [31] is a little bit faster, our method reaches a better accuracy. By contrast, PolarMask [35] can achieve higher mAP at the cost of a higher number of coefficients, and thus has a lower overall efficiency.

Table 3 shows that our method is faster and more efficient than ESE-Seg [37]. In addition, even though its mAP
score is a little bit lower, our method will not visually suffer from the drawbacks of Chebyshev coefficients detailed in section 3.

<table>
<thead>
<tr>
<th>Method</th>
<th>$N_{\text{coeff}}$</th>
<th>mAP</th>
<th>FPS</th>
<th>SEC</th>
<th>OES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolarMask [35]</td>
<td>36</td>
<td>32.9</td>
<td>4.1</td>
<td>1152</td>
<td>11.7</td>
</tr>
<tr>
<td>FourierNet-Cartesian [31]</td>
<td>16</td>
<td>22.9</td>
<td>4.9</td>
<td>512</td>
<td>21.9</td>
</tr>
<tr>
<td>FourierNet [31]</td>
<td>16</td>
<td>23.3</td>
<td>4.9</td>
<td>512</td>
<td>22.3</td>
</tr>
<tr>
<td><strong>Ours (SCR-RX101)</strong></td>
<td>16</td>
<td>27.3</td>
<td>4.2</td>
<td>512</td>
<td><strong>22.4</strong></td>
</tr>
</tbody>
</table>

Table 2. COCO 2017 test-dev results against other shape encoding methods based on ResNeXt-101 with an image height of 800 pixels. $N_{\text{coeff}}$ in real numbers for shape encodings: one complex coefficient counts as two. SEC: size of a single detected object in memory (bits). OES: Overall Efficiency Score.

<table>
<thead>
<tr>
<th>Method</th>
<th>$N_{\text{coeff}}$</th>
<th>mAP</th>
<th>FPS</th>
<th>SEC</th>
<th>OES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESE-Seg [37]</td>
<td>20</td>
<td>21.6</td>
<td>38.5</td>
<td>640</td>
<td>130</td>
</tr>
<tr>
<td><strong>Ours (SCR-D53)</strong></td>
<td>16</td>
<td>21.2</td>
<td>39.1</td>
<td>512</td>
<td><strong>162</strong></td>
</tr>
</tbody>
</table>

Table 3. COCO 2017 val results for models based on DarkNet-53 with an image height of 416 pixels. $N_{\text{coeff}}$ in real numbers for shape encodings: one complex coefficient counts as two. SEC (Shape Encoding Complexity): size of a single detected object in memory (bits). OES: Overall Efficiency Score.

The comparison against other types of approaches is shown in Table 4. Snake-based methods typically offer a better accuracy than shape encoding methods, however their overall efficiency suffers from a high shape encoding complexity and are slower than our method when using the same processing size and backbone. Mask-based methods reach very high mAP scores, but are also hindered by their speed and SEC and thus generally have a lower overall efficiency.

<table>
<thead>
<tr>
<th>Method</th>
<th>$N_{\text{coeff}}$</th>
<th>mAP</th>
<th>FPS</th>
<th>SEC</th>
<th>OES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepSnake [23]</td>
<td>256</td>
<td>31.0</td>
<td>6.68*</td>
<td>4096</td>
<td>5</td>
</tr>
<tr>
<td>DANCE [22]</td>
<td>196</td>
<td>34.6</td>
<td>7.6</td>
<td>3136</td>
<td>8</td>
</tr>
<tr>
<td>Mask R-CNN [3]</td>
<td>784</td>
<td>33.6</td>
<td>10.8</td>
<td>784</td>
<td>46</td>
</tr>
<tr>
<td>PANet [20]</td>
<td>784</td>
<td>38.2</td>
<td>4.5</td>
<td>784</td>
<td>22</td>
</tr>
<tr>
<td>SOLOv2 [34]</td>
<td>64000</td>
<td>38.8</td>
<td><strong>12.4</strong></td>
<td>64000</td>
<td>0.75</td>
</tr>
<tr>
<td><strong>Ours (SCR-R50)</strong></td>
<td><strong>16</strong></td>
<td><strong>24.2</strong></td>
<td>10.7</td>
<td><strong>512</strong></td>
<td><strong>51</strong></td>
</tr>
</tbody>
</table>

Table 4. COCO 2017 test-dev results against mask and snake-based methods with a ResNet-50 backbone and an image height of 800 pixels. $N_{\text{coeff}}$ in real numbers for shape encodings: one complex coefficient counts as two. SEC: size of a single detected object in memory (bits). For snake-based methods, we assume pixel coordinates are stored as 16-bit integers. For mask-based methods, we assume masks are binary. OES: Overall Efficiency Score. *: Number extrapolated from [22]

While our method does not necessarily give the best score for each of the evaluation metrics, it offers a good compromise between them. In particular, the OES scores obtained by our method is competitive compared to existing methods under the same backbone.

**Effect of perimeter penalty decay** As discussed in section 3, it is possible to set $\lambda_{\text{perim}}$ to zero after $L_{\text{perim}}$ stabilizes. Removing the perimeter regularization after a few iterations (2000 here) leads to more complex and accurate contours, while retaining the effect of regularization. On figure 3, the network learns to properly follow both front and back legs of the giraffe (left), which is not the case without perimeter penalty decay (right). This is also a typical example of non-star-shaped contour that cannot be regressed by previous works that opt for a single polar function. Setting $\lambda_{\text{perim}} = 0$ right away, i.e. removing $L_{\text{perim}}$ entirely, will let network circle objects multiple times, leading to poor results, as shown in our ablation study and on Figure 5(a).

**Fourier coefficient regularization** The benefits of our coefficient regularization can be observed on figure 4, where our network (left) learned to regress smooth contours compared to the erratic ones regressed the cartesian version of [31] (right) when using a high number of coefficients (here 32).
Figure 5. Disabling our perimeter regularization (a) lets the network circle the object multiple times, while disabling the coefficient regularization (b) leads to erratic shapes with lots of self-intersections. When both regularizers are enabled (c), the network outputs smooth and accurate shapes with a low number of self-intersections, here zero.

<table>
<thead>
<tr>
<th>Exp.</th>
<th>Backbone</th>
<th>Neck</th>
<th>Head</th>
<th>Ncoeff</th>
<th>Npts</th>
<th>H</th>
<th>Perim. reg</th>
<th>Coeff. reg</th>
<th>mAP</th>
<th>AP50</th>
<th>AP75</th>
<th>APs</th>
<th>APm</th>
<th>APl</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>ResNet-50</td>
<td>FPN</td>
<td>DCN</td>
<td>20</td>
<td>60</td>
<td>800</td>
<td>✓</td>
<td>✓</td>
<td>22.4</td>
<td>43.0</td>
<td>21.0</td>
<td>10.8</td>
<td>24.2</td>
<td>29.7</td>
</tr>
<tr>
<td>#2</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>20</td>
<td>128</td>
<td>800</td>
<td>✓</td>
<td>✓</td>
<td>21.4</td>
<td>44.2</td>
<td>19.0</td>
<td>10.9</td>
<td>22.9</td>
<td>28.4</td>
</tr>
<tr>
<td>#3</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>20</td>
<td>60</td>
<td>800</td>
<td>✓</td>
<td>✓</td>
<td>23.8</td>
<td>45.0</td>
<td>22.4</td>
<td>11.6</td>
<td>25.6</td>
<td>31.9</td>
</tr>
<tr>
<td>#4</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>40</td>
<td>60</td>
<td>800</td>
<td>✓</td>
<td>✓</td>
<td>23.8</td>
<td>46.4</td>
<td>21.8</td>
<td>11.7</td>
<td>25.4</td>
<td>32.3</td>
</tr>
<tr>
<td>#5</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>64</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>25.2</td>
<td>48.1</td>
<td>23.6</td>
<td>13.1</td>
<td>26.9</td>
<td>33.6</td>
</tr>
<tr>
<td>#6</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>20</td>
<td>60</td>
<td>360</td>
<td>✓</td>
<td>✓</td>
<td>19.1</td>
<td>36.2</td>
<td>18.1</td>
<td>4.0</td>
<td>19.9</td>
<td>31.3</td>
</tr>
<tr>
<td>#7</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>60</td>
<td>60</td>
<td>360</td>
<td>✓</td>
<td>✓</td>
<td>18.9</td>
<td>37.3</td>
<td>17.0</td>
<td>4.3</td>
<td>19.5</td>
<td>30.9</td>
</tr>
<tr>
<td>#8</td>
<td>DarkNet-53</td>
<td>FPN</td>
<td></td>
<td>16</td>
<td>60</td>
<td>320</td>
<td>✓</td>
<td>✓</td>
<td>17.6</td>
<td>33.8</td>
<td>16.4</td>
<td>1.5</td>
<td>17.0</td>
<td>32.1</td>
</tr>
<tr>
<td>#9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>416</td>
<td>✓</td>
<td>✓</td>
<td>21.2</td>
<td>39.3</td>
<td>20.4</td>
<td>3.7</td>
<td>22.6</td>
<td>35.5</td>
</tr>
<tr>
<td>#10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>608</td>
<td>✓</td>
<td>✓</td>
<td>23.9</td>
<td>42.2</td>
<td>24.0</td>
<td>6.2</td>
<td>27.5</td>
<td>36.3</td>
</tr>
<tr>
<td>#11</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td>20</td>
<td>128</td>
<td>800</td>
<td>✓</td>
<td>×</td>
<td>10.8</td>
<td>22.6</td>
<td>9.3</td>
<td>6.3</td>
<td>12.5</td>
<td>14.2</td>
</tr>
<tr>
<td>#12</td>
<td>ResNet-50</td>
<td>CARAFE</td>
<td>DCN</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>5.8</td>
<td>14.5</td>
<td>3.5</td>
<td>3.1</td>
<td>6.6</td>
<td>7.8</td>
</tr>
<tr>
<td>#13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
<td>5.4</td>
<td>13.9</td>
<td>3.1</td>
<td>2.9</td>
<td>6.1</td>
<td>7.3</td>
</tr>
<tr>
<td>#14</td>
<td>ResNeXt-101</td>
<td>CARAFE</td>
<td>DCN</td>
<td>20</td>
<td>60</td>
<td>800</td>
<td>✓</td>
<td>✓</td>
<td>27.2</td>
<td>50.3</td>
<td>26.4</td>
<td>14.0</td>
<td>29.4</td>
<td>35.4</td>
</tr>
<tr>
<td>#15</td>
<td>ResNeXt-101</td>
<td>CARAFE</td>
<td>DCN</td>
<td>64</td>
<td>60</td>
<td>800</td>
<td>✓</td>
<td>✓</td>
<td>27.2</td>
<td>50.9</td>
<td>26.1</td>
<td>13.9</td>
<td>29.5</td>
<td>35.9</td>
</tr>
</tbody>
</table>

Table 5. Impact of different design choices on our models on COCO test-dev. H = image height. Ncoeff in real numbers (one complex coefficient counts as two). Npts = number of contour points in ground truth and after IFFT. DCN = Deformable Convolution v2.

Ablation study We now evaluate the impact of our design choices through an ablation study. The results are summarized in Table 5. Numbers in parentheses in the following paragraph refer to corresponding lines of that Table.

(#2,#3) We trained the ResNet-50 based model with 60 and 128 contour points, and found that for this set of hyperparameters, there was a performance degradation when using 128 points. We thus kept this parameter at 60, a finding consistent with the literature. (#1,#3) We show that the new FPN-CARAFE from [32] improves the scores of SCR compared to the vanilla FPN, for which it is a drop-in replacement. (#3,#4,#5; #6,#7; #14,#15) Increasing the number of complex coefficients from 10 (20 real values) to 20 made no significant difference, while increasing it further to 32 yielded a small performance increase of 1.4 mAP. Depending on the requirements of the application, this increase in detail might be worth the extra storage space. Note that it is still possible to regress a high number of coefficients, then decide to set some of them to zero after the fact. (#6,#7,#8,#9) We find that the models used on smaller image resolutions do not perform very well on small objects (APs). (#14,#15) Using a larger ResNeXt-101 backbone yields a small accuracy improvement, but not to the point where we would think it is worth the decrease in FPS and the up to two times longer training times. (#10) DarkNet-53, on the other hand, seems to perform as well as ResNet-50 at high resolutions, especially on larger objects. (#2,#11,#12,#13) Disabling the L1 coefficient penalty leads to a very significant drop in mAP. This is probably due to the erratic shapes regressed without it, as shown on Figure 5. Disabling the L2 perimeter penalty also significantly affects the scores, as the contours circle the objects multiple times and thus miss a lot of detail.
Limitations  Our regularization terms tend to direct the training of our network towards simple shapes without actively preventing self-intersections. Thus, such errors might appear occasionally. On figure 6, for instance, we see that while the two leftmost horse contours seem properly regressed, they each present one self-intersection. Nevertheless, in that particular case, the masks given by these contours are still usable. One other limitation is that by design, our method is more tailored towards free-form objects than the regression of piece-wise linear shapes, for which polygon-based methods are better suited.

![Figure 6. Examples of self-intersections which can appear occasionally as they are not actively prevented. The contours still follow the shape of the animals and are thus still somewhat usable.](image)

Performance benchmarks on low-power hardware  We evaluate the speed of our fastest DarkNet-53 based network on a variety of devices, ranging from powerful discrete GPUs to low-power edge devices (Table 6).

Our first test platform is the Nvidia Jetson Nano, which is a popular, affordable and widely available yet reasonably powerful family of development kits. Our method runs at 9.5 FPS on these boards, which is close enough to realtime to be usable in many applications. We also run the same benchmarks on the more powerful Jetson Xavier NX, featuring INT8 compute capabilities, thus yielding a significant performance boost by bringing the throughput up to 115 FPS. It achieves 5.75 FPS/Watt, making it by far the most efficient combination. The Intel Movidius Myriad X VPU is an ultra-low-power AI accelerator with a power consumption of only 1W. The Myriad VPUs have been used in ESA’s φ-sat-1 CubeSat, as well as in smart security cameras, UAVs and industrial machine vision equipment. We benchmark SCR on the Myriad X using the imposed OpenVino toolkit and achieve 2.8 FPS, which is very satisfactory for such a low-power device. As a point of comparison with other works, we also run this network on an Nvidia GeForce GTX 1080Ti GPU, which is still one of the most popular discrete GPUs. For the sake of completeness, we also provide numbers on an Intel Xeon Silver 4114 CPU.

<table>
<thead>
<tr>
<th>Device</th>
<th>Power</th>
<th>Net.</th>
<th>H</th>
<th>Tool</th>
<th>Prec.</th>
<th>FPS</th>
<th>fps/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nano</td>
<td>10W</td>
<td>Ours</td>
<td>320</td>
<td>TensorRT</td>
<td>FP32</td>
<td>5.4</td>
<td>0.54</td>
</tr>
<tr>
<td>1080Ti</td>
<td>250W</td>
<td>Ours</td>
<td>320</td>
<td>MMDet</td>
<td>FP32</td>
<td>40.6</td>
<td>0.16</td>
</tr>
<tr>
<td>1080Ti</td>
<td>250W</td>
<td>Ours</td>
<td>608</td>
<td>TensorRT</td>
<td>FP32</td>
<td>186.6</td>
<td>0.74</td>
</tr>
<tr>
<td>Xavier NX</td>
<td>30W</td>
<td>Ours</td>
<td>320</td>
<td>TensorRT</td>
<td>FP32</td>
<td>17.2</td>
<td>0.57</td>
</tr>
<tr>
<td>Nano</td>
<td>1W</td>
<td>Ours</td>
<td>320</td>
<td>OpenVino</td>
<td>FP32</td>
<td>2.8</td>
<td>1.87</td>
</tr>
<tr>
<td>Myriad X</td>
<td>1.3W</td>
<td>Ours</td>
<td>320</td>
<td>OpenVino</td>
<td>FP32</td>
<td>2.8</td>
<td>1.87</td>
</tr>
</tbody>
</table>

Table 6. Throughput of SCR with Darknet-53 backbone on a wide range of devices with varying levels of power. H = image height.

5. Conclusion

In this paper, we proposed SCR, a method that captures object contours with a compact resolution-free representation based on a low number of Fourier coefficients. Using a complex representation and geometric priors, we were able to observe qualitative and quantitative improvements in the regressed shapes over previous methods.

We also implemented a downsized version of our model which, while retaining competitive accuracy, is able to run on a wide range of low-power hardware at very reasonable speeds (up to 115 FPS on a 30W device), and is therefore suitable for edge computing applications such as on-board processing of UAV or satellite images, autonomous vehicles and robotics.

Future work will include testing and evaluating SCR on other applicative scenarios, such as autonomous driving and remote sensing. Matching the "shape signature" of detected objects to an on-board database is an example of how our work might be used for embedded change detection. This work could be also extended by adding a polygon simplification scheme that takes advantage of properties of the Fourier decomposition, such as ease of differentiation.

Acknowledgements

This project is funded by the CIAR project at IRT Saint Exupéry. The authors are grateful to the OPAL infrastructure from Université Côte d’Azur for providing resources and support. This work was granted access to the HPC resources of IDRIS under the allocation 2020-AD01101311R2 made by GENCI. We thank Maxime Nabon for running our network on Jetson Xavier NX.
A. Chebyshev coefficients

We demonstrate why Chebyshev coefficients, as used in [37], are not well suited to the regression of continuous periodic functions through a neural network.

Chebyshev polynomials are obtained through the following recurrence relation:

\[
T_0(x) = 1 \\
T_1(x) = x \\
T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x).
\]  

(4)

Let \( N \) be the number of degrees used for interpolation of \( \rho \) using the truncated Chebyshev series. Then for \( x \in [-1, 1] \) we have:

\[
\rho(x) = \sum_{n=0}^{N} \alpha_n T_n(x)
\]  

(5)

Since the contour \( \rho \) is a continuous periodic function on \([-1, 1]\), in order to perfectly interpolate it, the Chebyshev coefficients \( \alpha_i \) must satisfy:

\[
\sum_{n=0}^{N} \alpha_n T_n(-1) = \sum_{n=0}^{N} \alpha_n T_n(1)
\]  

(6)

We also know that Chebyshev polynomials satisfy the following properties, for all \( n \in \mathbb{N} \):

\[
T_n(1) = 1 \\
T_{2n}(-1) = 1 \\
T_{2n+1}(-1) = -1
\]  

(7)

Then, we have the following:

\[
\sum_{k=0}^{N/2} \alpha_{2k} - \sum_{k=0}^{N/2} \alpha_{2k+1} = \sum_{n=0}^{N} \alpha_n
\]  

(8)

Thus, we have shown that in order to perfectly interpolate a continuous periodic function, the Chebyshev coefficients must satisfy the following relationship:

\[
\sum_{k=0}^{N/2} \alpha_{2k+1} = 0
\]  

(9)

Figure 7 shows the discontinuities created when this relationship is not strictly enforced, e.g. when the Chebyshev coefficients are regressed by a neural network. They appear at \( \theta = 2\pi \) on every regressed shape.

B. Implementation details

We use the MM Detection 2.7.0 object detection toolbox [4], based on MMCV 1.2.1 for easy and fair comparison with other methods. The underlying framework is PyTorch 1.7.0 with CUDA 10.1. We make use of the updated FFT package from PyTorch 1.7.0 which allows the use of complex numbers. We use the Chamfer Distance implementation from PyTorch3D 0.3.0 [24]. The interpolation scheme described in section 3 of the main text is implemented as a module in the MM Detection data loading pipeline using the interp module from Numpy.

<table>
<thead>
<tr>
<th>Name-Backbone</th>
<th>DCNv2</th>
<th>CARAFE</th>
<th>( N_{in} )</th>
<th>( N_{feat} )</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCR-D53-320</td>
<td>No</td>
<td>No</td>
<td>128</td>
<td>128</td>
<td>20h</td>
</tr>
<tr>
<td>SCR-R50-800</td>
<td>Yes</td>
<td>Yes</td>
<td>256</td>
<td>256</td>
<td>40h</td>
</tr>
<tr>
<td>SCR-X101-800</td>
<td>Yes</td>
<td>Yes</td>
<td>256</td>
<td>256</td>
<td>80h</td>
</tr>
</tbody>
</table>

Table 7. Characteristics of our main networks. Variations are further explored in our ablation study. DCN refers to the use of Deformable Convolution v2 [39] layers in the head. CARAFE refers to the use of the FPN-CARAFE [32] neck. Training time on 8 Tesla V100 GPUs.

Our networks are trained for 38 epochs using SGD with
momentum (0.9), with an initial learning rate of 0.0003, batch-size of 16 (2 images per GPU), gradient clipping at a maximum $L^2$ norm of 45, and employ of a 500 step warm-up with a ratio of 0.001 applied to the learning rate. We use the multiscale training feature found in MMDetection to train with image heights of 640 and 800 pixels. We balanced our loss function terms through trial-and-error, and settled on the following values: $\lambda_{cls}=1.0$, $\lambda_{cen}=1.0$, $\lambda_{CD}=1.0$, $\lambda_{perim}=0.01$, $\lambda_{coeff}=500.0$.

Due to software incompatibilities during the conversion of our DarkNet-53 model for low-power hardware targets, we had to simplify the architecture by removing deformable convolutions and CARAFE, as shown in Table 7.

C. Hardware configuration

This section contains details regarding the embedded hardware configuration used for the throughput benchmarks presented in section 4 of the main text.

Our Nvidia Jetson Nano 2GB, Nano 4GB and Xavier NX development kits are running JetPack 4.5, ONNXRuntime 1.4.0 and TensorRT 7.1.3. The Nano SoC features a quad-core ARM A53 CPU and a 128 CUDA core Maxwell GPU. In order to obtain consistent results, we lock the power consumption at 10W for benchmarking using the jetson_clocks tool and cool the heatsink with a Noctua A4x20 5V PWM fan using the default fan curve. We also ran our benchmarks on a Jetson Nano 4GB model and found no performance difference with the 2 GB model. Thus, Table 6 of the main text refers to both of them as "Nano".

The Jetson Xavier NX features a 6-core ARM processor, a 384 CUDA core Volta GPU and 8GB RAM, with a power consumption of 30W and is running the same setup.

The Intel Movidius Myriad X Vision Processing Unit is an ultra low-power AI accelerator with a power consumption of only 1W. It has been used in ESA's φ-sat-1 CubeSat, as well as in smart security cameras, UAVs and industrial machine vision equipment. We benchmark SCR on the Myriad X using OpenVino and FP16 precision.

Nvidia GeForce GTX 1080Ti (11GB VRAM, 250W) benchmarks were run using either the same MMDetection setup used for training, or TensorRT 7.1.3.

Intel Xeon Silver 4114 CPU benchmarks were run using the Intel OpenVino toolkit.

D. Carbon Impact Statement

While the focus of our work is to create efficient neural networks that are able to run on very low power devices, we cannot help but notice that these networks are still created and trained using power-hungry multi-GPU machines and wonder about the environmental impact. For this paper, we missed the opportunity to track the real power consumption of our experiments, and to estimate its global environmental footprint, but we are trying as much as we can to publish these estimations, as recommended in [1, 10, 14].

In order to run the experiments required for our main results and ablation study, we have used 5864.55 GPU hours on Nvidia Tesla V100 GPUs, which are rated for a power consumption of 300W. This, not counting CPUs, cooling, PSU efficiency, storage of datasets and results, as well as different trials or hyper-parameter searches on workstations, amounts to 1759kWh. Since the carbon intensity of our electricity grid is 10 gCO2/kWh, we estimate an emission of 17590 gCO2, which is equivalent to 146 km traveled by car according to [1].

E. Qualitative results

In this section, we provide an extended selection of qualitative results that could not be part of the main paper for space reasons. Figures 9 and 8 show results from our smallest and fastest DarkNet-53-based model.
Figure 8. More results from our DarkNet-53 based model selected from COCO 2017 test-dev.
Figure 9. More results from our DarkNet-53 based model selected from COCO 2017 test-dev.
References

[31] Hamd ul Moqeet Riaz, Nuri Benbarka, and Andreas Zell. Fouriernet: Compact mask representation for instance segmentation using differentiable shape decoders. In ICPR, 2020. 1, 2, 3, 4, 5, 6, 7, 8
[34] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Hujun Bao. Solo: Segmenting objects by locations. In ICCV, 2019. 2, 5, 6

