Resolution learning in deep convolutional networks using scale-space theory

Silvia L. Pintea*, Nergis Tomen*, Stanley F. Goes, Marco Loog, and Jan C. van Gemert

Abstract—Resolution in deep convolutional neural networks (CNNs) is typically bounded by the receptive field size through filter sizes, and subsampling layers or strided convolutions on feature maps. The optimal resolution may vary significantly depending on the dataset. Modern CNNs hard-code their resolution hyper-parameters in the network architecture which makes tuning such hyper-parameters cumbersome. We propose to do away with hard-coded resolution hyper-parameters and aim to learn the appropriate resolution from data. We use scale-space theory to obtain a self-similar parametrization of filters and make use of the N-Jet: a truncated Taylor series to approximate a convolutional filter by a learned combination of Gaussian derivative filters. The parameter \( \sigma \) of the Gaussian basis controls both the amount of spatial extent of the filter. Since \( \sigma \) is a continuous parameter, we can optimize it with respect to the loss. The proposed N-Jet layer achieves comparable performance when used in state-of-the-art architectures, while learning the correct resolution in each layer automatically. We evaluate our N-Jet layer on both classification and segmentation, and we show that learning \( \sigma \) is especially beneficial when dealing with inputs at multiple sizes.

Index Terms—Scale-space theory, Gaussian basis approximation, resolution learning in deep networks.

I. INTRODUCTION

Resolution defines the inner scale at which objects should be observed in an image [1]. To control the resolution in a network, one can change the filter sizes or feature map sizes. Because there is a maximum frequency that can be encoded in a limited spatial extent, the filter sizes and feature map sizes define a lower bound on the resolution encoded in the network. CNNs typically use small filters of \( 3 \times 3 \) or \( 5 \times 5 \) px, where the first layers are forced to look at detailed, local image neighborhoods such as edges, blobs, and corners. As the network deepens, each subsequent convolution increases the receptive field size linearly [2], allowing the network to combine the detailed responses of the previous layer to obtain textures, and object parts. Going even deeper, strategically placed memory-efficient subsampling operations reduce feature maps to half their size which is equivalent to increasing the receptive field multiplicatively. At the deepest layers, the receptive field spans a large portion of the image and objects emerge as combinations of their parts [3]. The resolution, as controlled by the sizes of the receptive field and feature maps, is one of the fundamental aspects of CNNs.

In modern CNN architectures [4, 5], the resolution is a hyper-parameter which has to be manually tuned using expert knowledge, by changing the filter sizes or the subsampling layers. For example, the popular ResNeXt [5] for the ImageNet dataset starts with a \( 7 \times 7 \) px filter, followed by \( 3 \times 3 \) px and \( 1 \times 1 \) px convolutions where the feature maps are subsampled 6 times. The same network on the CIFAR-10 dataset exclusively uses \( 3 \times 3 \) px convolutions and the feature maps are subsampled 2 times. Hard-coding the resolution hyper-parameters in the network for different datasets affects the extent of the receptive field, and the specific choices made can be restrictive.

In this paper we propose the N-JetNet which can replace CNN network design choices of filter sizes by learning these. We make use of scale-space theory [6], where the resolution is modeled by the parameter \( \sigma \) of the Gaussian function and its derivatives. Gaussian derivatives allow a truncated Taylor series, called the N-Jet [7], to model a convolutional filter [8] as a linear combination of Gaussian derivative filters, each weighted by an \( \alpha \). We optimize these \( \alpha \) weights instead of individual weights for each pixel in the filter, as done in a standard CNN. The choice of the basis cannot be avoided. In standard CNNs the choice is implicit: an \( N \times N \) pixel-basis, whose size cannot be optimized, because it has no well-defined derivative to the error. In contrast, in the N-Jet model the basis is a linear combination of Gaussian derivatives where the \( \sigma \) parameter controls both the resolution and the filter size, and has a well-defined derivative to the effective filter and therefore to the error. This formulation allows the network to learn \( \sigma \) and thus the network resolution. We exemplify our approach in Fig. 1.

To avoid confusions, we make the following naming conventions: throughout the paper we refer to ‘resolution’ as the

* Shared first authorship with equal contributions.
inner scale as defined in [1]; ‘size’ as the outer scale [1]
denoting the number of pixels of a filter or a feature map;
and ‘scale’ as the parameter controlling the resolution, which
is the standard deviation σ parameter of the Gaussian basis
[7]. The scale is different from the size of a filter: one can
blur a filter and change its scale without necessarily changing
its size. However, they are related as increasing the scale of
an object (i.e. blurring) increases its size in the image (i.e.
number of pixels it occupies). Here we tie the filter size to the
scale parameter by making it a function of σ.

We make the following contributions. (i) We exploit the
multi-scale local jet for automatically learning the scale param-
eter, σ. (ii) We show both for classification and segmentation
that our proposed N-Jet model automatically learns the
appropriate input resolution from the data. (iii) We demonstrate
that our approach generalizes over network architectures and
datasets without deteriorating accuracy for both classification
and segmentation.

II. RELATED WORK

Multiples scales and sizes in the network. Size plays an
important role in CNNs. The highly successful inception
architecture [9] uses two filter sizes per layer. Multiple input
sizes can be weighted per layer [10], integrated at the feature
map level [11], processed at the same time [12], [13], [14],
[15] or even made to compete with each other [16]. To process
multiple featuremap sizes, spatial pyramids are used [17],
[18], [19], [20], alternatively the best input size and network
resolution can be selected over a validation set [21]. Scale-
equivarainat CNNs can be obtained by applying each filter at
multiple sizes [22], or by approximating filters with Gaussian
basis combinations [23] where the set of scale parameters is
not learned, but fixed. Unlike these works, we do not explicitly
process our feature maps over a set of predefined fixed sizes.
We learn a single scale parameter per layer from the data.

Downsampling and upsampling can be modeled as a bi-
jective function [24], or made adaptive using reinforcement
learning [25] and contextual information at the object bound-
aries [26]. The optimal size for processing an input giving
the maximum classification confidence can be selected among
multiple sizes [27], [28], or learned by mimicking the human
visual focus [29], or minimizing the entropy over multiple
input sizes at inference time [30]. Network architecture search
can also be used for learning the resolution, at the cost of
increased computations [31]. Alternatively, the scale distribu-
tion can be adapted per image using dynamic gates [32], or
by using self-attentive memory gates [33]. The atrous [34],
[35] or dilated convolutions [36], [37] design fixed versions of
larger receptive fields without subsampling the image. These
are extended to adaptive dilation factors learned through a
sub-network [38]. Rather than only learning the filter size, we
learn both the filter shape and the size jointly, by relying on
scale-space theory.

Architectures accommodating subsampling. A pooling op-
eration groups features together before subsampling. Popular
forms of grouping are average pooling [39], and max pool-
ing [40]. Average pooling tends to perform worse than max
pooling [41], [42] which is outperformed by their combina-
tion [43], [44]. Other forms include pooling based on rank-
ing [45], spatial pyramid pooling [46], spectral pooling [47]
and stochastic pooling [48] and stochastic subsampling [49].
The recent BlurPool [50] avoids aliasing effects when sam-
ping, while fractional pooling [51] subsamples with a factor of
\(\sqrt{2}\) instead of 2 which allows larger feature maps to be
used in more network layers. All these pooling methods use
hard-coded feature map subsampling. Our work differs, as we
do not use fixed subsampling or strided convolution: we learn
the resolution.

Fixed basis approximations. Resolution in images is aptly
modeled by scale-space theory [52], [11], [53]. This is achieved
by convolving the image with filters of increasing scale,
removing finer details at higher scales. Convolving with a
Gaussian filter has the property of not introducing any arti-
facts [54], [55] and the differential structure of images can
be probed with Gaussian derivative filters [56], [6] which
form the N-Jet [7], a complete and stable basis to locally
approximate any realistic image. Scale-spaces model images
at different resolutions by a continuous one-parameter family
of smoothed images, parametrized by the value of σ of the
Gaussian filter [1]. In this paper we build on scale-space theory
and exploit the differential structure of images to optimize σ
and thus learn the resolution.

Various mathematical multi-scale image modeling tools
have been used in convolutional networks. The classical work
of Simoncelli et al. [57] proposes the steerable pyramid,
defining a set of wavelets for orientation and scale invariance.
Similarly, the seminal Scattering transform [58], [59] and its
extensions [60], [61] are based on carefully designed complex
wavelet basis filters [62] with pre-defined rotations and scales
giving excellent results on uniform datasets such as MNIST
and textures. Using the Scattering transform as initialization
for the first few layers of a CNN has recently [63], [64] been
shown to also lead to good results on more varied datasets.
Filters can also be approximated as a liner combination over
a set of learned low-rank filter basis [65]. Recent work
also starts with a filter basis and use a CNN to learn the
filter weights. Examples include a PCA basis [11], circular
harmonics [66], Gabors [67], and Gaussian derivatives [8].
In this paper we build on the Gaussian derivative basis [8]
because it directly offers the tools of Gaussian scale-space to
learn CNN resolution.

Learning kernel shape. Current methods investigate inherent
properties of CNN filters. Filters that go beyond convolution
include non-linear Volterra kernels [68], a learned image
adaptive bilateral filter [69] and learned image processing
operations [70]. For convolutional CNN filters, Sun et al. [71]
proposes an asymmetric kernel shape, which simulates hexag-
onal lattices leading to improved results. The active convo-
lution by Jeon and Kim [72] and the deformable CNNs by
Dai et al. [73] offer an elegant approach to learn a spatial
offset for each filter coefficient leading to flexible filters and
improved accuracy. [74] learns continuous filters as functions
over sub-pixel coordinates, allowing learnable resizing of the
feature maps. The hierarchical auto-zoom net [75], the scale
III. LEARNING NETWORK RESOLUTION

A. Local image differentials at given scale

Scale-spaces [56, 6, 53] offer a general framework for modeling image structures at various scales. The resolution, or the inner scale 1 of an image is modeled by a convolution with a 2D Gaussian. The 1D Gaussian at scale is given by 
\[ G(x, \sigma; \sigma) = \frac{1}{\sqrt{\pi} \sigma} e^{-x^2/2\sigma^2} \]
which is readily extended to 2D as 
\[ G(x, y; \sigma) = G(x; \sigma) \cdot G(y; \sigma) \] 
The local structure learned in deep networks 11 is linked to the image derivatives. Image pixels are discretely measured, and do not directly offer derivatives. The linearity of the convolution operator allows [31] to take an exact derivative of a slightly smoothed function \( f \) with a Gaussian kernel \( G(\cdot; \sigma) \) with scale \( \sigma \):
\[ \frac{\partial (f * G(x; \sigma))}{\partial x} = \frac{\partial G(x; \sigma)}{\partial x} * f(x), \]
where \( * \) denotes a convolution. This allows taking image derivatives by convolving the image with Gaussian derivatives. Gaussian derivatives in 1D at order \( m \) and scale \( \sigma \) can be defined recursively using the Hermite polynomials [82]:
\[ G^m(x; \sigma) = \frac{\partial^m G(x; \sigma)}{\partial x^m} \] 
\[ = (-1)^m \frac{m!}{\sigma^m \sqrt{2^m}} H_m \left( \frac{x}{\sigma \sqrt{2}} \right) G(x; \sigma), \] 
where \( G(x; \sigma) \) is the Gaussian function and \( H_m(x) \) the \( m \)-th order Hermite polynomial, recursively defined as \( H_0(x) = 2x H_{i-1}(x) - 2(i-1) H_{i-2}(x); H_0(x) = 1; H_1(x) = 2x \). We define 2D Gaussian derivatives by the product of the partial derivatives on \( x \) and on \( y \):
\[ G^{i,j}(x, y; \sigma) = \frac{\partial^{i+j} G(x, y; \sigma)}{\partial x^i \partial y^j} \] 
\[ = \frac{\partial^i G(x; \sigma)}{\partial x^i} \frac{\partial^j G(y; \sigma)}{\partial y^j}. \]

B. Multi-scale local N-Jet for modeling local image structure

A discrete set of Gaussian derivatives up to \( n \)-th order, \( \{G^{i,j}(x, y; \sigma) \mid 0 \leq i + j \leq n \} \), can be used in a truncated local Taylor expansion to represent the local scale-space near any given point with increasing accuracy 12. This allows us to approximate a filter \( F(x) \) around the point \( a \) up to order \( N \) as:
\[ F(x) = \sum_{i=0}^{N} \frac{\alpha^i}{i!} F(a) (x-a)^i + \frac{R(a)}{(N+1)!} (x-a)^{N+1}, \]
where \( R \) is the residual term that corresponds to the approximation error. By absorbing the polynomial coefficients into a value \( \alpha \), we arrive at a linear combination of Gaussian derivative basis filters which can be used to approximate image filters, as illustrated in Fig. 2. For filter \( F(x, y; \sigma) \) at position \( (x, y) \) and color channel \( c \) the approximation is:
\[ F(x, y, c; \sigma) = \sum_{0 \leq i, 0 \leq j, 0 \leq c} \alpha_{i,j,c} \frac{\partial^{i+j}}{\partial x^i \partial y^j} G^{i,j}(x, y; \sigma) + R(x, y, c; \sigma), \]
where \( R \) is the residual error, ignored here. Optimizing the \( \alpha \) parameters allows us to switch from learning pixel weights as commonly used in CNNs, to learning the weights of the Gaussian basis filters. We show some examples in Fig. 3 where we optimize the \( \alpha \) parameters of an order-3 RGB Gaussian derivative basis with \( \sigma = 5 \) to least squares fit an \( 11 \times 11 \) px patch. Results show that the fit can approximate well a slightly blurred version of the original patch. Because \( \sigma = 5 \) we cannot recover a perfectly sharp faithful copy of the original patch.

C. Learning receptive field size

We have all ingredients to learn the resolution in a convolutional deep neural network (CNN). Resolution is bounded by the size of the CNN filters. We can now dynamically adapt the resolution during training.
Safely subsampling for Image Classification.

**Scale-invariant basis normalization.** The filter responses of Gaussian derivatives decay with order, as depicted in figure 3(a). Following [7], we make the Gaussian derivatives scale-independent by multiplying each \( i \)-th order partial derivative by \( \sigma^i \). This brings the magnitude of basis filters in approximately the same range, as illustrated in Fig. 3(b).

**Learning scale and filter size.** The network resolution depends on the parameter \( \sigma \), determining the inner scale of the Gaussian derivative basis. The chain-rule for differentiation allows to express the derivative of the error \( J \) with respect to \( \sigma \) as the product of two terms:

\[
\frac{\partial J}{\partial \sigma} = \frac{\partial J}{\partial F} \cdot \frac{\partial F}{\partial \sigma}.
\]

The first term is the derivative of the error with respect to the filter and it is found by error-backpropagation, as standardly done. The second term is the derivative of the filter with respect to \( \sigma \) and can be found by differentiating Eq. (5) with respect to \( \sigma \). Similarly, the value of the Gaussian basis mixing coefficients, \( \alpha_{i,j,c} \), can be found by differentiating the filter \( F \) with respect to the coefficients \( \alpha \).

In practice we cannot work with continuous filters. Therefore, we need to clip the filters to a finite size to perform the convolution. The size \( s \) of the filter follows the formula:

\[
s = 2 \lceil k \sigma \rceil + 1,
\]

where \( k \) determines the extent of the local N-Jet approximation and is experimentally set. By tying the filter size to the scale parameter, we only need to change \( \sigma \) and adapt both the scale controlling the network resolution, and the size defining the spatial extent of the filters.

**IV. Experiments**

**A. Exp (A): N-Jet for Image Classification.** The receptive field size is also altered through subsampling, pooling, or strided convolution. For classification models we remove all subsampling operations in the network and add a safe-subsampling operation. If the resolution is low (i.e., the \( \sigma \) value is high) then there is no need to keep the feature map at full size, and it can safely be subsampled, to improve memory and speed. For a feature map of size \( s \), we subsample the feature map to a new size \( \bar{s} \), where we half its current size as a function of \( \sigma \) as:

\[
\bar{s} = s \left( \frac{1}{2} \right)^{\sigma/r},
\]

where \( r \) is the safe-subsampling hyper-parameter. We apply safe-subsampling for all models except for the very deep networks: Resnet-110 and EfficientNet, where it is reducing the feature map sizes too much.

**Experimental setup.** We validate our approach on three standard datasets: CIFAR-10, CIFAR-100 [83] and SVHN [84], and multiple network architectures: NIN (Network in Network) [85], ALLCNN [86], and Resnet-32, Resnet-110 [87], as well as the recent EfficientNet [88]. To derive our N-Jet models, we replace all the normal convolutional layers with variants of our N-Jet convolutional layers. When using safe-subsampling we remove all pooling layers, and set the stride to 1 in all network layers. In all our N-Jet models we set the order of the Taylor series approximation to 3, unless otherwise specified. For all the models reported, we add a batch normalization layer after the convolutional layers, for robustness, and use the momentum SGD with the momentum set to 0.9. We train for the number of epoch reported in the literature. For the NIN baseline model we found the best starting learning rate to be 0.5, while for the ALLCNN baseline 0.25. We use the same starting learning rates in our N-Jet models. For our N-Jet-NIN model we use an \( L_2 \) regularization weight over the Gaussian mixing coefficients, \( \alpha \), set to 0.01, while for N-Jet-ALLCNN we regularize the \( \alpha \)-s with a weight decay of 0.001. When training our N-Jet-Resnet models we use an \( L_2 \) regularization weight over the Gaussian mixing coefficients, \( \alpha \), set to 0.0001, and a starting learning rate of 0.1 as indicated in [87]. We evaluate on relatively small datasets, and therefore we use the lightweight version of Resnet where the first block has 16 channels and the last block 64, while for the deeper Resnet-110 models we use bottleneck blocks with a \( 4 \times \) channel expansion. For both the baseline EfficientNet and our N-Jet-EfficientNet we train the models from scratch, and given the small datasets we use the smallest model B0 [88]. For the EfficientNet baseline we use a 0.01 learning rate and a batch size of 32 and we rescale the inputs to \( 224 \times 224 \).
px, since otherwise the model performs poorly, maybe due to the large subsampling. In our N-Jet-EfficientNet we keep the input images to their original size. For our deeper models N-Jet-EfficientNet and N-Jet-ResNet-110 we use batches of 16 and a learning rate of 0.001.1

Experiment 1(A): Validation

Experiment 1.1(A): Do resolution hyper-parameters really matter? We test our assumption that filter sizes and feature map sizes affect accuracy. For this we use the NIN baseline trained on CIFAR-10. We vary the filter sizes in the layers of the NIN which are not 1 × 1 convolutions, and we reset the strides to 1 in all layers, to remove the feature map subsampling. Fig. 6 shows the impact of changing the filter sizes and removing the subsampling. The smaller filter sizes, as in the case when all filter sizes are set to 3, are affected to a greater degree by the removal of the subsampling because they have a smaller receptive field. Selecting the correct filter sizes impacts the overall classification accuracy, and an exhaustive search over all possible filter size combinations is not feasible. This validates the need for learning filter sizes.

Experiment 1.2(A): Can the image resolution be learned?

To test resolution learning, we create a toy network architecture depicted in Fig. 5(a). We train the toy architecture on MNIST when resizing the images 1 × 1.5 × 2 and 2 ×. Fig. 5(b) shows the learned Gaussian basis scale, σ, per setup. The σ values learned for the images resized by 1.5 and 2 do not directly correspond to these values because the operations of sampling and resizing are not commutative: we first discretized the continuous signal into an image and subsequently subsampled it. However, the relative ratio between the learned scales is close: (2.0/1.5)σ₁₁.₅ = 2.81 ± 0.04 ≈ σ₂₀ = 2.82 ± 0.04. The learned σ values follow the input resizing, thus the correct filter scales and sizes can be learned from the input.

Experiment 2(A): Model choices

Experiment 2.1(A): Learning sigma. We test the effect of σ on the performance on the CIFAR-10 dataset, using the NIN backbone. We fix the spatial extent, k, to 2 and vary sigma in the set {0.5, 1.0, 2.0}. Tab. II shows that a wrong setting of σ can influence the classification accuracy up to 3%. The safe-sub-sampling is slightly more sensitive to σ than the baseline sampling.

| Sampling       | σ = 0.5 | σ = 1.0 | σ = 2.0 |
|----------------|---------|---------|---------|
| Baseline       | 88.76%  | 90.25%  | 87.39%  |
| Safe-subsampling | 86.29%  | 89.50%  | 87.26%  |

1 We will provide the N-Jet code at http://github.com/SilviaLauraPintea/N-JetNet

The learned σ on multi-size MNIST.

![Image](https://example.com/image.png)

Fig. 6. Exp 1.1(A): The impact on accuracy when varying filter sizes and feature map sizes for the NIN baseline on CIFAR-10. Smaller filter sizes are more affected by the removal of the subsampling. Setting the resolution hyper-parameters wrong can deteriorate the accuracy.
TABLE II

| r | Accuracy | Training time  |
|---|----------|---------------|
| 2.0 | 91.59% | ≈129.91 min. |
| 4.0 | 91.60% | ≈179.43 min. |
| 6.0 | 91.63% | ≈193.48 min. |

TABLE III

| Model | NIN [85] | N-Jet-NIN (Ours) |
|-------|----------|-----------------|
| SVHN  | 98.17%   | 97.55% (±0.08)  |
| CIFAR-10 | 90.89% | 91.60% (±0.08)  |
| CIFAR-100 | 66.14% | 68.42% (±0.31)  |

For an approximation of order 3 in the N-Jet, there is a small increase in the number of parameters compared to the baseline, as well as the number of parameters. Using our proposed N-Jet architectures, for the baseline model we only report the best performance we obtain rerunning the models, while for our N-Jet models we report mean and standard deviations over 3 runs. We achieve comparable classification accuracy with the baseline.

### Experiment 3.1(A): Generalization to other datasets.

We compare our N-Jet-NIN method with the baseline NIN [85]. We test the generalization properties of our method by also reporting scores on two other datasets: CIFAR-100 and SVHN.

Table III shows the classification results of our N-Jet-NIN when compared with the baseline NIN. We report mean and standard deviations over 3 runs for our method. We show in Fig. 7 the hard-coded sizes of the baseline feature maps, versus the sizes learned by our N-Jet-NIN on CIFAR-10. The performance of N-Jet-NIN is comparable with the baseline performance, while dynamically learning the appropriate feature map size.

### Experiment 3.2(A): Generalization to other models.

To test the generalization of our N-Jet convolutional layer to different network architectures, we use the ALLCNN [86], Resnet [87], and the recent EfficientNet [88] backbone network architectures. For N-Jet-ALLCNN we use safe-subsampling at every layer, while for N-Jet-Resnet-32 only at the layers where the original network subsamples. Table IV shows the classification accuracy of the baseline models tested by us on CIFAR-10, CIFAR-100, when compared with our N-Jet models. We report mean and standard deviation over 3 repetitions for our models, as well as the number of parameters. Using our proposed N-Jet layers gives similar accuracy to the standard convolutional layers, while avoiding the need to hard-code the filter sizes. For an approximation of order 3 in the N-Jet, there is a small increase in the number of parameters compared to the baseline, except for the EfficientNet which uses also kernel sizes larger than 3 × 3 px. When employing larger models – N-Jet-ResNet-110 and N-Jet-EfficientNet – a Gaussian basis combination of order 2 is sufficient to obtain an accuracy comparable to the baseline models, while reducing the number of parameters. In Fig. 8 we show the baseline ALLCNN feature map sizes when compared to the N-Jet-ALLCNN learned feature map sizes on CIFAR-10. The N-Jet model has similar classification accuracy to the ALLCNN baseline, while learning at every layer the befitting feature map size. Applied at every layer, the safe-subsampling makes the subsampling continuous and smooth, compared to the baseline.

### Experiment 4(A): Comparison to scale-invariant methods.

We evaluate on the normal sized 28 × 28 px MNIST and MNIST resized by a factor of 4 with a size of 112 × 112 px. We compare against a standard CNN with varying filter sizes, and against the Deformable CNN [73], as well as Atrous (dilated) convolutions [33]. We consider 2 and 4-layer toy architectures containing only convolutional layers followed by ReLU activations. Results in Table V show that

---

1. We use torchinfo (https://github.com/tyleryep/torchinfo) to enumerate all the parameters.
Experimental Results: Multiscale Image Segmentation

**Table V:** Comparison of scale-invariant methods. We evaluate on MNIST and MNIST resized 4 × our N-Jet, standard convolutions of varying filter sizes, as well as using atrous convolutions [43] and deformable convolutions [73]. Our N-Jet performs well on MNIST × 4 despite the increase in size.

| CNN         | 2-layer architecture | MNIST 4 × MNIST |
|-------------|----------------------|-----------------|
| Standard 3 × 3 | 97.04% (± 0.22) | 86.27% (± 3.44) |
| Standard 5 × 5 | 98.62% (± 0.08) | 88.67% (± 5.97) |
| Standard 9 × 9 | 98.93% (± 0.08) | 95.84% (± 0.76) |
| Standard 11 × 11 | 98.72% (± 0.20) | 95.75% (± 1.50) |
| Atrous [43]   | 98.47% (± 0.25) | 89.87% (± 3.67) |
| Deformable [73] | 97.54% (± 0.46) | 84.09% (± 0.84) |
| N-Jet (Ours)  | 99.05% (± 0.04) | 98.11% (± 0.16) |

| CNN         | 4-layer architecture | MNIST 4 × MNIST |
|-------------|----------------------|-----------------|
| Standard 3 × 3 | 98.49% (± 0.20) | 86.53% (± 7.15) |
| Standard 5 × 5 | 98.54% (± 0.37) | 97.47% (± 0.41) |
| Standard 9 × 9 | 98.91% (± 0.18) | 94.23% (± 4.01) |
| Standard 11 × 11 | 98.81% (± 0.20) | 96.21% (± 1.40) |
| Atrous [43]   | 98.40% (± 0.38) | 91.28% (± 1.88) |
| Deformable [73] | 98.91% (± 0.27) | 86.45% (± 1.32) |
| N-Jet (Ours)  | 99.37% (± 0.05) | 98.87% (± 0.21) |

The standard CNN performs well on MNIST, yet results are sensitive to the filter size for 4 × MNIST. The Deformable CNN [73] is also affected by the change in image size. Our intuition is that the Deformable CNN still relies on the initial 3 × 3 convolutions and optimizing the offsets is difficult under large size changes in the input. For Atrous CNN the dilation factor has to be hard-coded, and we use a dilation factor of 2, as using 4 would imply including prior knowledge. The Atrous performance is also affected by the change in input size. In contrast, our N-Jet model is able to learn the correct resolution and is more accurate.

B. Exp (B): N-Jet for Image Segmentation

Learning the receptive field size for segmentation. Multiscale information processing is heavily used in modern segmentation architectures, and seems to be an important performance booster [10]. Here, we focus on two popular mechanisms for multiscale processing, namely the merging of information at different scales via skip connections in U-Net architectures [89], and the pooling of information at different scales via atrous spatial pyramid pooling (ASPP) layers in DeepLab architectures [53], [90].

Similar to the classification experiments (Section IV-A), we replace the fixed-size convolutional filters of baseline networks with the N-Jet definition, where we learn the size and scale of the filters in the convolution operations during training.

Experiment 1(b). Segmentation with U-Net

Experiment 1.1(b): Segmentation of multiscale inputs. We first evaluate the performance of N-Jet models on a small toy dataset, where each input image is formed by concatenating 4 images (objects) from the Fashion MNIST dataset (Fig. 9). The corresponding segmentation masks are generated by assigning the class label of the corresponding object to pixels whose input grayscale values are above the threshold \( h_{x,y} \geq h_\theta = 0.2 \).

Each object is assigned a random scale \( s \), which determines the factor by which we upsample the original Fashion MNIST image, via bilinear interpolation. The scale affects the object sizes — the number of pixels occupied by the object in the image. We construct four different training sets: three where the scale of each object is homogeneous: the discrete variable \( s \) has the probability mass functions \( P(s = 1) = 1 \), or \( P(s = 2) = 1 \), or \( P(s = 4) = 1 \); and one where the image contains objects on multiple scales: \( s \) has the probability mass function \( P(s) = 0.25 \) for values \( s \in \{1, 2, 3, 4\} \). After rescaling, each object is placed in one quadrant of the input image, centered at a uniformly sampled random location. The corresponding ground truth segmentation masks are created by assigning the class label \( (1 \ldots 10) \) of the corresponding object to pixels whose input grayscale values \( h_{x,y} \) are above the threshold \( h_\theta = 0.2 \), by assigning the background label \( 0 \) to pixel locations where \( h_{x,y} = 0 \) and by assigning an ignore index to undetermined pixel locations where \( 0 < h_{x,y} < h_\theta \). The ignored pixel locations do not contribute to the loss during training and do not contribute to the accuracy at test time.

Due to the simple nature of the training set, we use a small U-Net architecture, where the encoding network has three levels, as opposed to five in the original U-Net [89]. This corresponds to two downsampling layers. Each level is composed of two convolutional layers, followed by the ReLU activation layer. The channel dimension is 64 at the first level, and doubles with every downsampling, performed via \( 2 \times 2 \) max pooling. In the decoding network, we use bilinear upsampling to increase feature map size, and in all convolutional layers we use ‘same’ padding. We train all networks (N-Jet and baseline) for 50 epochs, using the ADAM...
Experiment 1.2(B): Learning the receptive field size.

While \( \sigma \) optimization is successful for different basis orders, we note that the N-Jet model with basis order 4 has a larger number of free parameters than the baseline U-Net. Nevertheless, we observe on our toy dataset that the validation mIoU depends only weakly on the number of parameters, beyond a certain network size. For the multi-scale segmentation task with \( s = 4 \), where the scale of objects is increased by a factor of 4, we find that the receptive field size at the end of the encoding network largely determines the validation mIoU (Fig. 11). To demonstrate this, we vary the number of parameters and the receptive field size at the end of encoding in the baseline U-Net models, until we match the N-Jet performance: we increase the kernel size \( k \) from 3 to 4 and 5, and expand the depth of the baseline network by increasing the number of encoding and decoding levels from 3 (10 convolutional layers) to 4 (14 convolutional layers) and 5 (18 convolutional layers). To keep the number of trainable parameters at a reasonable level, for networks with 4 and 5 levels we also decrease the channel width of the layers (by halving or quartering the number of channels in each layer, as given in the legend of Fig. 11).

We find that the N-Jet models can outperform baseline U-Net models while using a much smaller number of free parameters, due to \( \sigma \) optimization. In addition, we show that while the receptive field size is a good predictor of performance, it cannot be learned during training for the baseline U-Net, and would need to be optimized via hyper-parameter scans. This can potentially mean increasing the depth of the network to match the input resolution, which cannot be parallelized. Finally, we observe that slightly better validation mIoU can be obtained by baseline models, with almost 7 times the number of parameters and double the number of layers. We attribute this slight performance boost to the much larger depth, and thus increased number of nonlinearities in the network.

Experiment 2(B): Image segmentation using DeepLabv2.

Next, we consider a more realistic segmentation task on the Pascal VOC (SBD) dataset [23], [24] using the DeepLabv2 architecture [34]. Modern DeepLab models take advantage of dilated convolutions to aggregate information from multiple

---

**Fig. 10.** Exp 1.1(B): mIoU scores averaged over 5 repetitions with different random seeds on multi-scale Fashion MNIST. Error bars denote the standard deviation. Validation mIoU decreases dramatically for baseline networks with fixed filter sizes (orange) for larger objects. In comparison, N-Jet models are robust against scale changes, as they can learn the filter size and scale, without changing the architecture or hyper-parameters.

**Fig. 11.** Exp 1.2(B): The effect of the number of parameters (x-axis) and receptive field size (round marker size) on the mIoU scores in the validation set of multi-scale Fashion MNIST dataset with \( s = 4 \). We observe that the number of model parameters is a weak predictor of segmentation performance compared to receptive field size which can be learned by N-Jets during training. We note that N-Jet models (in black squares) can perform dramatically better than the baseline model with the same architecture (orange circle), and overall display high mIoU with a low number of parameters.
To that end, we employ the DeepLabv2 model with a ResNet-101 backbone pretrained on the 20 class subset of the MS COCO dataset [93], corresponding to the Pascal VOC classes. We retain all the network and training hyper-parameters of the original DeepLabv2 model and finetune the baseline network with an ASPP output layer on Pascal VOC with the batch normalization layers frozen. For the N-Jet network, we replace the 4 convolutional layers of the ASPP layer with different dilation rates with 4 N-Jet layers with independent, learnable scales $\sigma$ (during finetuning) and we impose weight sharing between the different scales (i.e. same $\alpha$ values). On top of eliminating the need to manually tune the dilation rates, N-Jet models with weight sharing also have the potential to dramatically reduce the number of parameters in ASPP layers.

We find that DeepLabv2 with N-Jet output layers indeed allows for parameter reductions (Tab. VI). Using an N-Jet output layer with basis order 3 and weight sharing, we achieve validation mIoU values within 1% of the baseline network, while reducing the number of parameters by nearly a factor of 4. As an additional control, we also train a baseline network with weight sharing within the ASPP layer. Interestingly, we observe that our N-Jet models attain on par or better performance than the weight-tied baseline network, even when we only use a basis order of 1 (each kernel is defined by only 3 free parameters).

It is worth noting that these validation mIoUs are achieved with no hyper-parameter tuning for the N-Jet models, and despite not using N-Jet layers in the pretraining of the DeepLab backbone. As it is, we believe N-Jet output layers may be used for multi-scale processing applications with further hyper-parameter tuning of learning rates and regularization parameters, or can be used out of the box to estimate the optimal scale or dilation rates for other architectures.

### V. Discussion and Limitations

To illustrate the differences between standard convolutional layers and N-Jet convolutional layers, we visualize a set of trained baseline filters compared to the equivalent N-Jet filters (Fig. 12). We find that in many models earlier layers will converge to smaller $\sigma$ values during training (Fig. 12 (top)), while deeper layers are prone to learning larger filter sizes (Fig. 12 (bottom)).

In addition, the strength of the N-Jet representation lies in that it can learn filter sizes, and thus the receptive field size, during training. However, recent work has demonstrated that the effective receptive field (eRF) size of networks can be considerably smaller than what would be expected from the kernel size [96]. We investigate the change in eRF size in our N-Jet models by visualizing the gradients with respect to the input image in our models trained on the multiscale Fashion-MNIST dataset (Fig. 13). We find that, as expected, the eRF size of N-Jet models grows with the size of the training images, proportionally to the growth of filter sizes. The baseline U-Net model with $3 \times 3$ kernels cannot learn to adapt its receptive field size during training, its eRF size remains relatively constant as a function of the input image scale.

One of the limitations of our proposed kernels is that they are typically larger than the standard $3 \times 3$ px, and therefore the convolutions take longer to compute. This comes at no cost in parameters as the size of the N-Jet filters is only affected by the scale parameters, $\sigma$. Additionally, computing the Gaussian basis is more expensive because it involves more operations: computing the Hermite polynomials, and obtaining the individual Gaussian basis from these, followed by estimating their linear combinations with the weights $\alpha$. For the NiN architecture, our model is $\approx 2 \times$ slower than the baseline model. As the network depth increases, so do the computations. However, manual architecture search takes a lot of manual architecture search takes a lot of...
Effective receptive field sizes. Effective receptive field (eRF) size for the N-Jet models trained on the multiscale Fashion-MNIST dataset. Columns ($s = \{1, 2, 4\}$) denote models trained with input images scaled up by a factor of $s$. eRF visualizations are obtained by the gradients backpropagated to input pixel space [96]. We find that the eRF size of N-Jet models scale up with the size of the input, while the eRF size of baseline U-Net models stay constant.

longer for finding the appropriate resolution hyper-parameters, because it requires a grid search over all possible filter sizes given a specific network depth and sub-sampling strategy.

VI. CONCLUSION

We learn the resolution in deep convolutional networks. Learning the resolution frees the network architect from setting resolution related hyper-parameters such as the receptive field size and subsampling layers, which are dataset and network dependent. While we learn the receptive field size and the feature map subsampling for classification, the resolution is also determined by the depth of the network, as each layer increases the resolution linearly. Network depth is not something we learn, and thus we do not learn all resolution hyper-parameters. In addition to hard-coded filter sizes and subsampling layers, current CNN architectures are also designed to share the same filter size in a single layer. Due to computational restraints, our implementation does not make it possible to learn a $\sigma$ for each filter, rather than per layer. We leave this as potential future work. To conclude, by replacing pixel-weights convolutional layers with our N-Jet convolutional layers we show that we can learn, and thus we do not learn all resolution hyper-parameters.

Acknowledgements. This publication is part of the project "Pixel-free deep learning" (with project number 612. 001.805 of the research programme TOP which is financed by the Dutch Research Council (NWO).

REFERENCES

[1] J. J. Koenderink, “The structure of images,” Biological cybernetics, vol. 50, no. 5, pp. 363–370, 1984.
[2] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” in NIPS, pp. 4898–4906, 2016.
[3] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in ECCV, pp. 818–833, Springer, 2014.
[4] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely connected convolutional networks,” in CVPR, 2017.
[5] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, “Aggregated residual transformer deep neural networks,” in CVPR, pp. 5987–5995, IEEE, 2017.
[6] T. Lindeberg, Scale-space theory in computer vision, vol. 256. Springer Science & Business Media, 2013.
[7] L. Florack, B. T. H. Romeny, M. Viergever, and J. Koenderink, “The gaussian scale-space paradigm and the multiscale local jet,” IVC, vol. 18, no. 1, pp. 61–75, 1996.
[8] J.-H. Jacobsen, J. C. van Gemert, Z. Lou, and A. W. M. Smeulders, “Structured receptive fields in CNNs,” in CVPR, 2016.
[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in CVPR, 2015.
[10] L.-C. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille, “Attention to scale: Scale-aware semantic image segmentation,” in CVPR, 2016.
[11] G. Ghiasi and C. C. Fowlkes, “Laplacian pyramid reconstruction and refinement for semantic segmentation,” in ECCV, 2016.
[12] S. Bai, V. Koltun, and J. Z. Kolter, “Multiscale deep equilibrium models,” ICLR workshop, 2018.
[13] A. Kanazawa, A. Sharma, and D. Jacobs, “Locally scale-invariant convolutional neural networks,” NIPS workshop, 2014.
[14] T.-W. Ke, M. Maire, and X. Y. Stella, “Multigrid neural architectures,” in CVPR, 2017.
[15] Y. Li, Z. Kuang, Y. Chen, and W. Zhang, “Data-driven neuron allocation for scale aggregation networks,” in CVPR, 2019.
[16] Z. Liao and G. Carneiro, “A deep convolutional neural network module that promotes competition of multiple-size filters,” Pattern Recognition, 2017.
[17] E. Agustsson, D. Minnen, N. Johnston, J. Balle, S. J. Hwang, and G. Toderici, “Scale-space flow for end-to-end optimized video compression,” in CVPR, pp. 8503–8512, 2020.
[18] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in CVPR, pp. 2117–2125, 2017.
[19] Z. Liu, G. Gao, L. Sun, and L. Fang, “Ipg-net: Image pyramid guidance network for small object detection,” in CVPR Workshops, pp. 1026–1027, 2020.
[20] X. Wang, S. Zhang, Z. Yu, L. Feng, and W. Zhang, “Scale-equalizing pyramid convolution for object detection,” in CVPR, pp. 13359–13368, 2020.
[21] T. Yang, S. Zhu, C. Chen, S. Yan, M. Zhang, and A. Willis, “Mutualnet: Adaptive convnet via mutual learning from network width and resolution,” in ECCV, 2020.
[22] D. Marcos, B. Kellenberger, S. Lobry, and D. Tuia, “Scale equivariance in cnns with vector fields,” ICML workshop, 2018.
[23] I. Sosnovik, M. Sznaja, and A. Smeulders, “Scale-equivariant steerable networks,” ICLR, 2020.
[24] M. Xiao, S. Zheng, C. Liu, Y. Wang, D. He, G. Ke, J. Bian, Z. Lin, and T.-Y. Liu, “Invertible image rescaling,” ECCV, 2020.
[25] B. Uzkent and S. Ermon, “Learning when and where to zoom with deep reinforcement learning,” in CVPR, pp. 12345–12354, 2020.
[26] D. Marin, Z. He, P. Vajda, P. Chatterjee, S. Tsai, F. Yang, and Y. Boykov, “Efficient segmentation: Learning downsampling near semantic boundaries,” in ICCV, pp. 2131–2141, 2019.
[27] Y. Li, D. M. Tax, and M. Loog, “Scale selection for supervised image segmentation,” Image and Vision Computing, vol. 30, no. 12, pp. 991–1003, 2012.
[28] Y. Li, D. M. Tax, and M. Loog, “Supervised scale-invariant segmentation (and detection),” in International Conference on Scale Space and Variational Methods in Computer Vision, pp. 350–361, 2011.
[29] G. A. Sigurdsson, A. Gupta, C. Schmid, and K. Alahari, “Beyond the camera: Neural networks in world coordinates,” CoRR, 2020.
[30] D. Wang, E. Shelhamer, B. Olshausen, and T. Darrell, “Dynamic scale inference by entropy minimization,” CoRR, 2019.
[31] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 82–92, 2019.
[32] Y. Li, L. Song, Y. Chen, Z. Li, X. Zhang, X. Wang, and J. Sun, “Learning dynamic routing for semantic segmentation,” in CVPR, pp. 8553–8562, 2020.
[33] G.-J. Qi, “Hierarchically gated deep networks for semantic segmentation,” in CVPR, pp. 2267–2275, 2016.
