Frequency learning for structured CNN filters with Gaussian fractional derivatives

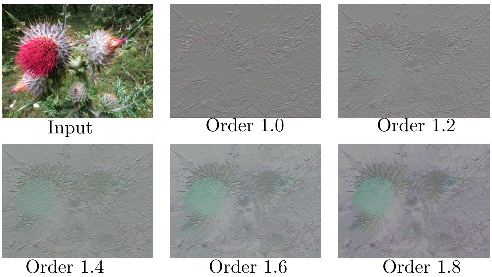
Nikhil Saldanha,
Silvia L. Pintea,
Jan C. van Gemert,
Nergis Tomen

Computer Vision Lab,
Delft University of Technology,
Delft, Netherlands

Abstract

Frequency information lies at the base of discriminating between textures, and therefore between different objects. Classical CNN architectures limit the frequency learning through fixed filter sizes, and lack a way of explicitly controlling it. Here, we build on the structured receptive field filters with Gaussian derivative basis. Yet, rather than using predetermined derivative orders, which typically result in fixed frequency responses for the basis functions, we learn these. We show that by learning the order of the basis we can accurately learn the frequency of the filters, and hence adapt to the optimal frequencies for the underlying learning task. We investigate the well-founded mathematical formulation of fractional derivatives to adapt the filter frequencies during training. Our formulation leads to parameter savings and data efficiency when compared to the standard CNNs and the Gaussian derivative CNN filter networks that we build upon.

1 Introduction



Input

Order 1.0

Order 1.2

Order 1.4

Order 1.6

Order 1.8

Figure 1: Filter responses when using fractional order Gaussian derivative filters (here x-order and y-order are equal). Defining the filters using fractional derivative orders adds flexibility in terms of the peak response frequency, and enables the use of standard gradient backpropagation for training.

The world comes in many frequencies, and we rely on frequency as encoded in texture to differentiate between different object types: a purple thistle flower versus a purple tulip flower. What's more, convolutional neural networks (CNNs) additionally use texture (e.g. 'fur' versus 'skin') for discriminating between dissimilar object categories [6]. Therefore, CNNs can reap benefits from an explicit lever for controlling the frequencies extracted from the data.

Current acclaimed CNNs architectures [11, 33, 36, 37] lack an explicit knob to control the frequencies learned from the data. These classical CNN architectures hard-code the filter sizes thus limiting the frequency resolution contained in the filters. Moreover, they learn each filter value separately at each featuremap location by treating the weights as independent, leading to data inefficiency. Here, we address both these issues, by proposing a way to explicitly control the frequency learning in a data-efficient continuous formulation using structured receptive fields with Gaussian basis.

We make the observation that the order of the Gaussian basis in the structured receptive fields (SRFs) [16] explicitly controls the maximum frequency of the filters, and therefore the maximum frequencies they can detect in the data. We, additionally, observe that when using SRFs [16], typically a few Gaussian basis functions are sufficient to extract useful information. However, while it may be adequate to use a single basis function out of the whole basis to define each kernel, selecting from a large range of derivative orders may be necessary. Putting together these observations, we aim to learn a single Gaussian derivative per kernel where the order of the Gaussian derivative is adapted during training to better represent the frequencies present in the data. Typically, the derivative order is an integer (e.g. first order derivative or second order derivative) which makes backpropagation difficult. However, the order of the Gaussian derivatives become differentiable when working within the domain of fractional calculus. In this work, we make use of the fractional derivatives of the Gaussian function to learn the derivative order. Fig. 1 shows examples of image responses when using fractional order Gaussian derivatives. Fractional orders add flexibility in terms of the frequencies that the model can encode and make the model easily trainable using standard gradient backpropagation methods.

This article makes the following contributions: (i) We propose a well-founded method for learning the filter frequencies from data, and demonstrate its effectiveness experimentally; (ii) To that end, we describe a mathematically solid approach to learning fractional order Gaussian derivatives; (iii) We demonstrate improved data efficiency and parameter savings across 4 datasets when comparing with existing standard CNNs and baselines with structured CNN filters.

2 Related Work

Structured filters in CNNs. Influential prior work has investigated the usefulness of structured filters for image analysis. Simoncelli *et al.* [32] define a steerbale pyramid using a set of wavelets that encode orientation and scale, while Mallat defines complex wavelet basis filters in [24]. These complex wavelets have been used in the Scattering transform [1, 25] which is later extended in [5, 27, 31, 34]. Other works consider PCA basis [7], Gabors [22, 28], circular harmonics [43], or simply learning the basis from the data [18]. A large amount of work has been focused on Gaussian derivatives basis [16] used for controlling the scale in deep networks [21, 29, 35] or for making the networks continuous over space and depth [39]. Here, we also build on the Gaussian derivative basis [16] because it allows us to easily control the number of learnable parameters by directly learning the order of the Gaussian derivative basis. The order parameter controls the complexity of the patterns the filters can respond to, therefore by learning the order we learn how complex these filters need to be. While wavelets, such as Gabor filters, can directly learn the frequency response of the filters, the frequency parameter of the wavelet is coupled to its scale which relates to its spatial extent. Our representation decouples the frequency response and the scale/spatial extent of the filters, via two independently trained parameters: derivative order and scale-parameter σ.

Parameter efficiency and data efficiency in CNNs. CNNs come with large computational costs entailed by the large number of parameters to be learned on the training data. A new trend is emerging with focus on efficiency. Model compression has been the most intuitive manner of reducing computations and memory [10, 12, 45]. Alternatively, the use of 1×1 convolutions have significantly reduced the parameters in SqueezeNets [8, 15]. Depthwise separable convolutions combined with 1×1 convolutions have shown parameter efficiency [3, 13, 23, 47]. More recently EfficientNet [37] shows both accuracy improvement and parameter reduction by carefully scaling network width, depth and resolution. Similarly, here we also propose a model aimed at reduced parameters by learning how complex the filters need to be. Moreover, our proposed fractional structured filters can be used in combination with any efficient convolutional architecture.

Frequency learning in CNNs. Analyzing the deep networks in frequency domain has brought insights into how they work. Deep networks can fit, barely perceivable, high-frequency signals, thus leading to vulnerability to adversarial attacks [38, 40, 46]. However they tend to learn low frequency signals first [30]. Rather than using frequency domain to analyze deep networks, the networks can actually be trained in the frequency domain [9, 41] or over inputs transformed to the frequency domain [44]. Here, we also analyze which frequencies our model can fit well and where it makes errors. Our proposal learns the appropriate frequency of the filters by learning the order of the Gaussian basis.

3 Fractional structured filters

3.1 Review of Gaussian basis filters

Rather than representing filters as a discrete set of pixel values, the use of Scale-space theory [20, 42] enables the definition of filters as continuous functions [16, 35, 39]. And instead of learning the values of the individual pixels, one only needs to learn the parameters of these functions. The underlying idea is that a filter $F(x)$ can be approximated with a Taylor expansion around a point a, up to a certain order N:

$$F(x) \approx \sum_{i=0}^{N} \frac{F^i(a)}{i!}(x-a)^i. \tag{1}$$

Scale-space theory [20, 42] defines the filter derivatives F^i as the convolution $(*)$ of the filter F with Gaussian derivatives, G^i:

$$F(x) \approx \sum_{i=0}^{N} \frac{(G^i(.;\sigma) * F)(a)}{i!}(x-a)^i \tag{2}$$

where σ is the standard deviation of the Gaussian representing the scale parameter [20]. The recursive formulation relying on Hermite polynomials [26] allows to effectively compute the i^{th} Gaussian derivative G^i as a point-wise multiplication $(\circ)$ between the Gaussian G and the i^{th} Hermite polynomial, H_i:

$$G^i(x;\sigma) = \left(\frac{-1}{\sigma\sqrt{2}}\right)^i H_i\left(\frac{x}{\sigma\sqrt{2}}\right) \circ G(x;\sigma), \tag{3}$$

where the recursive definition of the Hermite polynomials is: $H_0(x) = 1$; $H_1(x) = 2x$; $H_i(x) = 2xH_{i-1}(x) - 2(i-1)H_{i-2}(x)$.

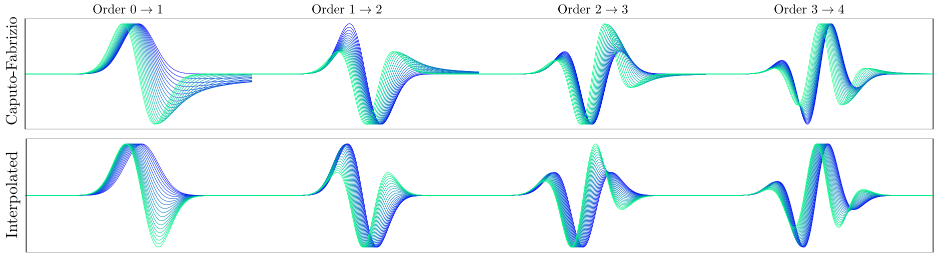


Figure 2: Top: Gaussian derivatives computed using Caputo-Fabrizio [2] fractional derivative form. Bottom: Fractional Gaussian derivatives computed via interpolation between integer orders. The error introduced by using the interpolation is small relative to the Caputo-Fabrizio form.

By simplifying Eq. (2) and incorporating the polynomial coefficients in a set of weights α, previous work [16, 39] defines the filter approximation F as a linear combination of Gaussian derivatives up to order N:

$$F(x,\sigma) \approx \sum_{i=0}^{N} \alpha_i G^i(x;\sigma), \tag{4}$$

where both the weights α and the scale parameter σ are can be learned from data [29, 39].

3.2 Fractional structured filters: Learning the basis order

We propose to learn the frequency of the filters by making the order of the Gaussian basis a learnable parameter. Instead of defining the filter as a linear combination of Gaussian derivatives up to order N, as previously done [16], we approximate the filter with only one weighted Gaussian derivative, where the order of the derivative ν is a learnable parameter:

$$F(x;\sigma) \approx \alpha G^{\nu}(x;\sigma). \tag{5}$$

When using this filter definition in a deep network, we can obtain the gradients of the loss function with respect to ν through the standard network backpropagation. One caveat of learning the order of the Gaussian derivative is that a gradient descent step will always result in real (fractional) order updates. Since the Gaussian derivatives are traditionally only defined for integer orders, we need to account for orders in between two integers.

One possible way of dealing with fractional derivatives is the Caputo-Fabrizio [2] form, which in the 1D case is:

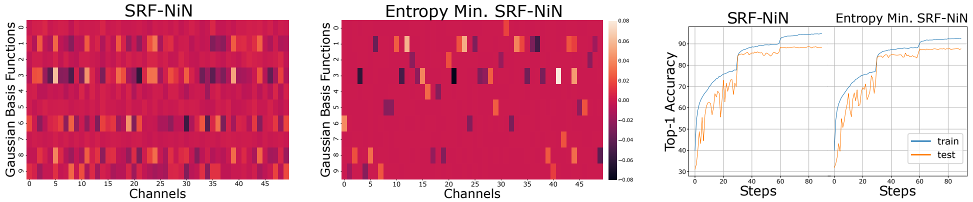
$$G^{\nu}_{CF}(x;\sigma) = \frac{1}{1-\nu}\cdot\frac{1}{\sqrt{2\pi\sigma^3}}\exp\left(-\frac{1}{1-\nu}\left(x-\frac{\sigma^2}{2}\cdot\frac{1}{1-\nu}\right)\right)\zeta_{\sigma,\nu}(x) \tag{6}$$

where $\zeta_{\sigma,\nu}(x)$ is an integral of the form:

$$\zeta_{\sigma,\nu}(x) = \int_0^t (\mu - x)\exp\left(-\frac{\left(\tau+\frac{1}{1-\nu}\sigma^2\right)^2}{2\sigma^2}\right)d\tau \tag{7}$$

However, when using this formulation, we observed exploding gradients due to the non-linear terms. A more straight-forward approach is to interpolate between the two closest integers of the fractional order:

$$G^{\nu}_{Iter}(x;\sigma) = (\lceil\nu\rceil-\nu)G^{\lfloor\nu\rfloor}(x;\sigma)+(\nu-\lfloor\nu\rfloor)G^{\lceil\nu\rceil}(x;\sigma), \tag{8}$$



(a) (b) (c)

Figure 3: (a) The distribution of α weights (color bar) learned in a layer of the original SRF-NiN [16] model on *CIFAR-10*. (b) The distribution of α-s when minimizing their entropy. (c) Training/test accuracies for the original SRF-NiN and the entropy-minimized version. We can safely reduce the number of Gaussian derivatives defining the filters (i.e. set most basis coefficients α to zero), at no cost to validation accuracy.

where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are the ceil and floor roundings of ν. This formulation permits us to keep the gradients in check, due to linear nature of interpolation used. Fig. 2 shows a number of fractional order Gaussian derivatives when going from order 0 to 1, 1 to 2, 2 to 3, and 3 to 4. On the top row the Caputo-Fabrizio form (Eq. (6)) is used for computing the 1D derivatives, while on the bottom row the interpolation method (Eq. (8)) for estimating fractional Gaussian derivatives. There is on average less than 0.22 root mean squared error between these two estimations. In all our experiments we use the linearly interpolation method to compute the fractional Gaussian derivatives.

Because we are working with images, we use 2D Gaussian derivatives. The outer product ($\otimes$) of 1D Gaussian derivatives along the x- and y-direction defines the 2D Gaussian derivative: $G^{i+j}(x,y;\sigma) = G^{i}(x;\sigma) \otimes G^{j}(y;\sigma)$.

3.3 Deep networks with fractional structured filters

Each 2D Gaussian derivative requires two order parameters: the order on the x-axis, ν_x, and the order on the y-axis, ν_y. When considering a filter F of size $[C,K,W,H]$ with C input channels and K output channels, we learn in practice two order parameters (ν_{ck}^{x}, ν_{ck}^{y}) per kernel in the filter, and a scalar (α_{ck}) for each kernel:

$$F(c,k,x,y;\sigma) = \alpha_{ck}(G^{\nu_{ck}^{x}}(x;\sigma) \otimes G^{\nu_{ck}^{y}}(y;\sigma)), \tag{9}$$

where the scale parameter σ is shared among the kernels in the filter and can either be learned as in [29, 39], or fixed as in [16, 35]. Our method is more flexible than the structured receptive fields (SRF) [16], allowing for non-integer derivatives. We coin our filters FracSRF.

Is one Gaussian derivative sufficient? Unlike previous work [16, 39], we do not use a linear combination of Gaussian derivatives up to a fixed order. We use a single Gaussian derivative, whose order can be learned. To check whether using a single Gaussian derivative is sufficient, we do a small test on the *CIFAR-10* dataset, using SRF filters [16] over a NiN [19] backbone. In the SRF model the α weights control how much a certain integer-order Gaussian derivative contributes to the final filter. Fig. 3.(a) shows the distribution of the α-s in a layer of the original SRF-NiN model, compared to the same model in Fig. 3.(b) where we normalize the α values and we minimize their entropy. Minimizing the entropy of α-s reduces the actual number of Gaussian derivatives used per filter. At no loss in accuracy (Fig. 3.(c)) the number of Gaussian derivatives can be reduced from 9 to 2 per channel. This supports our intuition that using one Gaussian derivative is sufficient, where we make it more flexible by learning its order from the data.

4 Experiments

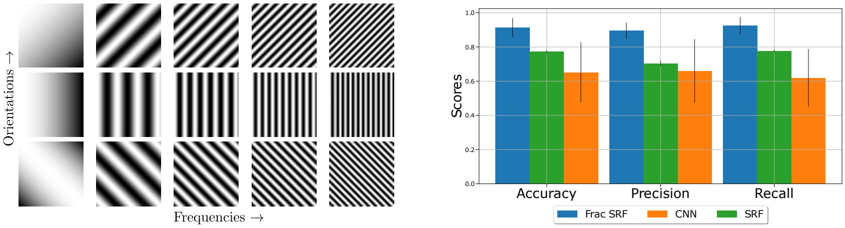
4.1 Experimental setup

Datasets. We test our method across 4 datasets: *CIFAR-10*, *CIFAR-100* [17], and *STL-10* [4] and ImageNette [14], having low and high resolution images, respectively. Additionally, to test the method's ability to learn the correct data frequency, we created a dataset called *Sinusoids* containing 2D sinusoids of various orientations and 5 spatial frequencies defining the 5 classes. We also test our method's accuracy in few-data samples regime by sub-sampling the *CIFAR-10* dataset between 40 and 0.04% of the original number of images.

Models. We consider several backbone architectures: Network in Network (NiN) [19], Resnet-32 [11], EfficientNet-b0 [37]. We also compare with a few methods using structured filters: SRF [16, 29]. To obtain the SRF and our FracSRF variants, we replace all the non 1×1 convolutional layers either with SRF layers or with FracSRF layers. For the SRF networks, we always set the Gaussian basis orders to 2. For our models we initialization of the orders uniformly between $[1,6]$, set the spatial filter extent to 2σ around the center and initialize $\sigma = 1$, unless stated otherwise. We train using SGD with momentum of 0.9 and L_2 regularization of 5e-4. For FracSRF-NiN, FracSRF-Resnet32, FracSRF-Efficientnetb0 we use a learning rate of 0.1, 0.05, 0.001 and batch sizes of 128, 256, and 16. When enabling σ learning in FracSRF, we use a different learning rate and weight decay for σ of 0.001 and 0.01 on FracSRF-Resnet-32, while on FracSRF-EfficientNet-b0 we use 0.001 and 0.05 for σ learning. We keep learning rates and batch sizes fixed across datasets except for FracSRF-Efficientnet-b0 on *STL-10* where due to memory limitations, we use a batch size of 4 and learning rate proportionally increased to 0.05. For the baselines NiN, Resnet-32 and EfficientNet-b0 we use learning rates of 0.1, 0.01, 0.01 and batch sizes of 128, 128 and 16, respectively. Given the relatively small dataset sizes, we use the lightweight version of *Resnet-32* where the first block has 16 channels and the last block 64. For the SRF-NiN, SRF-Resnet-32 and SRF-EfficientNet-b0 we use learning rates of of 0.1, 0.05, 0.001 and batch sizes of 128, 256, and 16 respectively.

4.2 *Exp 1:* Does FracSRF learn the correct data frequency?

We test the hypothesis that our FracSRF is more flexible in learning a large range of frequencies, by learning the Gaussian derivative order. For this we create a synthetic toy dataset coined the *Sinusoids* dataset. Fig. 4.(a) shows a few examples from this dataset. The dataset contains 5 classes, each with 600 training examples and 200 test examples. Each class cor-



(a) *Sinusoids* dataset

(b) *Sinusoids* scores

Figure 4: **Exp 1:** (a) Examples from the toy *Sinusoids* dataset. We vary the number of frequencies and the orientations. (b) Accuracy / Precision / Recall results on the *Sinusoids* dataset. For a baseline CNN, its SRF equivalent, and FracSRF. Our FracSRF is more suitable for learning varying frequencies.

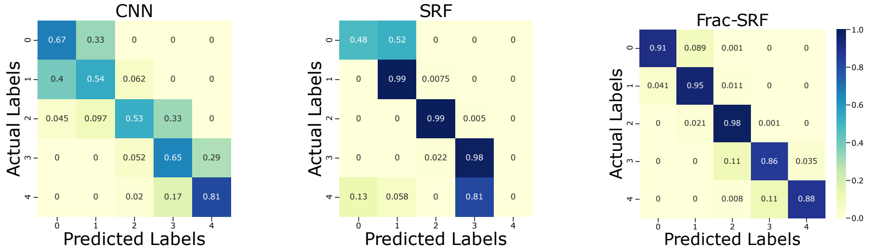


Figure 5: **Exp 1:** Confusion matrices for the CNN, SRF [16], and FracSRF small networks on the *Sinusoids* dataset. Our FracSRF can learn varying frequencies, and therefore it is better at distinguishing the 5 classes.

	Filter scale initialization				
	$\sigma = 2^{-2}$	$\sigma = 2^{-1}$	$\sigma = 2^{-0}$	$\sigma = 2^{1}$	$\sigma = 2^{2}$
Top-1 Accuracy (%)	90.59 ± 0.2	90.65 ± 0.04	90.90 ± 0.05	90.68 ± 0.25	90.26 ± 0.29
Initial Filter Size	3×3	5×5	7×7	9×9	11×11
Training Time (sec/epoch)	79.2s	79.2s	79.8s	79.2s	81.0s

Table 1: **Exp 2.(a):** Impact of initializing the filter scale on the performance and training time of the FracSRF-NiN on *CIFAR-10*. The network can adapt the scale parameter σ even when initialized far from the optimum. The best initialization seems to be $\sigma = 2^0$.

responds to a different frequency, where we vary the orientations of the sinusoids across examples. For this experiment we use a small 2-layer network where the first layer has 32 output channels and the second 5 output channels. We repeated the experiments 5×. For the normal CNN we learn the filters the traditional way, for the SRF we replace the filters with a linear combination of Gaussian derivatives as in [16] with $\sigma = 1$, and for FracSRF we use a single weighted Gaussian derivative with $\sigma = 1$. All filters are 5×5 px.

Fig. 5 shows confusion matrices for the CNN, SRF [16] and FracSRF 2-layer networks on the *Sinusoids* dataset. Fig. 4.(b) reports accuracy, precision and recall scores for these three methods. SRF cannot predict the highest frequency classes, being limited by its fixed order in the Gaussian basis. The CNN is not able to resolve between similar frequencies and tends to confuse neighboring classes. Our FracSRF can learn the varying frequencies and therefore is able to better separate the 5 frequency classes.

4.3 *Exp 2:* Model choices analysis

***Exp 2.(a):* Impact of scale initialization.** We test the effect of the initialization of the scale parameter (σ) of the Gaussian derivatives, in our FracSRF filters. Following [39] we learn the σ and initialize it as a power of 2, which avoids dealing with negative σ gradients during training. And we initialize the order uniformly in $[1,6]$. Table 1 shows results across 3 repetitions when varying σ for the FracSRF-NiN on *CIFAR-10*. The initialization of the scale parameter shows minors variations, with $\sigma = 2^0$ being the best. The network can correct for the scale well even when initialized far away from the optimum. Additionally, using larger scales impacts the training time.

Exp 2.(b): Impact of order initialization. We test the effect of initializing the Gaussian derivative order on the *CIFAR-10* dataset using FracSRF-NiN. We vary the initialization of the order by uniformly sampling in the ranges: $[1,3]$, $[3,6]$, and $[6,10]$. We repeated the experiments 3×. Table 2 shows the optimal order initialization is found in the interval $[1,3]$.

	Filter order initialization		
	order $\in \mathcal{U}_{[1,3]}$	order $\in \mathcal{U}_{[3,6]}$	order $\in \mathcal{U}_{[6,10]}$
Top-1 Accuracy (%)	90.62 ± 0.20	90.34 ± 0.12	89.52 ± 0.13
Training Time (sec/epoch)	74.4s	74.5s	79.2s

Table 2: **Exp 2.(b):** Impact of order initialization on CIFAR-10 using FracSRF-NiN. There is not a large difference in performance between different order ranges used for initialization. The model can learn to adapt the order to the best one. Higher orders require more computations.

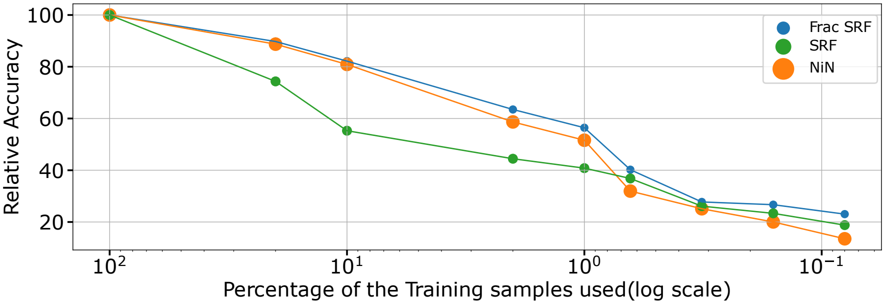


Figure 6: **Exp 3.(a):** Data efficiency in the FracSRF model. Relative accuracy of NiN, SRF-NiN [16, 29], and FracSRF-NiN on subsets of *CIFAR-10*. The dot size of each method indicates the relative number of parameters. The performance of each model are normalized as a percentage of their own accuracy at 100% training data. The scores of our FracSRF-NiN degrade less rapidly especially when compared to NiN and SRF-NiN.

There is not a large difference between the different initialization ranges, suggesting that the model can learn the correct orders for task. Starting from larger order range is suboptimal as the training time increases: the Hermite polynomial computations requires more time at higher orders. *CIFAR-10* does not contain many high frequencies and therefore it is reasonable that orders up to 3 are able to capture the information.

4.4 *Exp 3:* FracSRF performance analysis

Exp 3.(a): Accuracy in few-samples regime. We test our method in the few-training samples regime. We train on different sub-sets of the *CIFAR-10* dataset and evaluate on the full test set. We compare our FracSRF-NiN with the baseline NiN and other models using structured filters such as the SRF-NiN [16], which also have been shown to generalize well with few training examples. Fig. 6 shows the relative accuracy of each model as a percentage of its own top-1 accuracy when trained with 100% of the data: therefore all models start at 100% and scores decrease with the decrease in training samples. We also indicate through the dot size in the plot the relative number of parameters of each model. Our FracSRF has the smallest number of parameters. This plot shows the expected degradation of the performance of the networks as training data decreases. The scores of our FracSRF-NiN degrade less rapidly, especially when compared to the SRF-NiN and the original NiN model.

Exp 3.(b): Accuracy versus parameter reduction. We test the accuracy versus parameter efficiency for our FracSRF models when compared to a set of baseline CNNs and their SRF versions with fixed scale [16] and learned scale [29], on *CIFAR-10*, *CIFAR-100*, *STL-10* and *ImageNette* for the ResNet-32 backbone. Table 3 reports accuracies and number of parameters. Our FracSRF layer achieves comparable performance to standard convolutional

	NiN [19]	SRF		FracSRF (ours)	
		Fixed scale [16]	Learned scale [29]	Fixed scale	Learned scale
% Params (count)	100% (0.98M)	51% (0.5M)	53% (0.52M)	33% (0.33M)	35% (0.35M)
CIFAR-10	90.90%	85.30%	91.48%	86.60%	91.30%
CIFAR-100	67.80%	61.50%	68.30%	61.90%	67.80%
STL-10	80.13%	59.40%	70.00%	71.00%	77.75%
	ResNet-32 [11]	SRF-ResNet-32		FracSRF-ResNet-32 (ours)	
		Fixed scale [16]	Learned scale [29]	Fixed scale	Learned scale
% Params (count)	100% (0.47M)	63% (0.30M)	65% (0.31M)	31% (0.15M)	34% (0.16M)
CIFAR-10	92.28%	88.33%	92.20%	87.99%	91.60%
CIFAR-100	67.90%	65.82%	67.61%	63.00%	67.50%
STL-10	72.30%	68.40%	70.30%	67.40%	72.00%
ImageNette	86.37%	78.57%	81.24%	80.23%	83.57%
	EfficientNet-b0 [37]	SRF-EfficientNet-b0		FracSRF-EfficientNet-b0 (ours)	
		Fixed scale [16]	Learned scale [29]	Fixed scale	Learned scale
% Params (count)	100% (3.6M)	96% (3.47M)	96% (3.48M)	95% (3.43M)	95% (3.45M)
CIFAR-10	92.31%	89.37%	93.50%	84.50%	90.23%
CIFAR-100	76.20%	67.50%	75.81%	66.89%	72.50%
STL-10	73.20%	67.50%	71.78%	65.83%	71.81%

Table 3: **Exp 3.(b):** Classification accuracies versus number of parameters on *CIFAR-10*, *CIFAR-100*, *STL-10* and *ImageNette* datasets when comparing the baseline NiN, Resnet-32 and EfficientNet-b0 with their SRF variants [16, 29] and our FracSRF variants. Our method has comparable accuracy with the baselines while largely reducing the number of parameters. On the high resolution, encoding more frequencies, *STL-10* dataset our method consistently outperforms the other models.

networks, while reducing the number of parameters 2 to 3 times on NiN and Resnet-32. On the EfficientNet-b0 we do not see large parameter reductions because the model heavily relies on 1×1 convolutions which are not replaced with our FracSRF layers. On *STL-10* our model with learned σ and learned Gaussian derivative order consistently outperforms the other models. On the *ImageNette* dataset our method outperforms the baseline SRF while reducing the number of parameters, as it does not limit the maximum filter frequency. The *STL-10* dataset contains high resolution images (96×96 px) allowing for higher frequencies to be present in the data. While the other methods cannot adapt to varying data frequencies, our models learn this information through the order parameter of the Gaussian derivatives.

5 Discussion

One of the limitations of our model is that computations increase with derivative order, because we rely on the recursive Hermite polynomials to define the Gaussian derivatives. However, while being computationally more expensive than standard CNNs, we find that FracSRF models are 25% faster during training compared to baseline SRF models (time estimates averaged over the complete training epochs) which also rely on the Hermite polynomials. The training time speedup comes from only computing 2 Gaussian derivative basis functions per filter.

Another limitation is that the scale learning is fairly unstable and it needs proper regularization and careful learning rate selection. Additionally, we notice that the orders have the tendency to go towards negative values, requiring clipping during training. However, our model greatly reduces the number of parameters when compared to standard 3×3 convolutional layers where instead of learning 9 parameters per kernel, it only needs to learn 3 parameters per kernel: the scale σ, and the orders ν_x and ν_y.

6 Conclusion

We propose to explicitly learn the frequencies present in the data by encoding these in a trainable network parameter. We start from the structured filters based on Gaussian derivative basis and make the observation that by learning the order of the Gaussian derivative we can learn to control the filter frequencies. We show experimentally that our model can learn the correct frequencies from the data on a synthetic dataset and test the abilities of our model on standard benchmark datasets when compared to NiN, ResNet and EfficientNet backbone architectures. Our model degrades gracefully with fewer training samples, and it can achieve good accuracy at large parameter reductions.

Acknowledgement 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] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *TPAMI*, 35(8):1872–1886, 2013.

[2] Michele Caputo and Mauro Fabrizio. A new definition of fractional derivative without singular kernel. *Progr. Fract. Differ. Appl*, 1(2):1–13, 2015.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[4] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[5] Fergal Cotter and Nick Kingsbury. Visualizing and improving scattering networks. In *MLSP*, 2017.

[6] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *ICLR*, 2019.

[7] Golnaz Ghiasi and Charless C Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. In *ECCV*, 2016.

[8] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1638–1647, 2018.

[9] Kfir Goldberg, Stav Shapiro, Elad Richardson, and Shai Avidan. Rethinking fun: Frequency-domain utilization networks. *arXiv preprint arXiv:2012.03357*, 2020.

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, 2017.

[14] Jeremy Howard and Sylvain Gugger. Fastai: a layered api for deep learning. *Information*, 11(2):108, 2020.

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *CoRR*, 2016.

[16] Jorn-Henrik Jacobsen, Jan van Gemert, Zhongyu Lou, and Arnold W. M. Smeulders. Structured receptive fields in cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[18] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *ICCV*, pages 5623–5632, 2019.

[19] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, 2013.

[20] Tony Lindeberg. *Scale-space theory in computer vision*, volume 256. Springer Science & Business Media, 2013.

[21] Tony Lindeberg. Scale-covariant and scale-invariant gaussian derivative networks, 2020.

[22] Shangzhen Luan, Chen Chen, Baochang Zhang, Jungong Han, and Jianzhuang Liu. Gabor convolutional networks. *IEEE Transactions on Image Processing*, 27(9):4357–4366, 2018.

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[24] Stéphane Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[25] Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

[26] J-B Martens. The hermite transform-theory. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9):1595–1606, 1990.

[27] Edouard Oyallon, Eugene Belilovsky, and Sergey Zagoruyko. Scaling the scattering transform: Deep hybrid networks. In *ICCV*, 2017.

[28] Juan C Pérez, Motasem Alfarra, Guillaume Jeanneret, Adel Bibi, Ali Thabet, Bernard Ghanem, and Pablo Arbeláez. Gabor layers enhance network robustness. In *European Conference on Computer Vision*, pages 450–466. Springer, 2020.

[29] Silvia L Pintea, Nergis Tomen, Stanley F Goes, Marco Loog, and Jan C van Gemert. Resolution learning in deep convolutional networks using scale-space theory. *arXiv:2106.03412*, 2021.

[30] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[31] Laurent Sifre and Stéphane Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *CVPR*, 2013.

[32] Eero P Simoncelli, William T Freeman, Edward H Adelson, and David J Heeger. Shiftable multiscale transforms. *IEEE transactions on Information Theory*, 38(2):587–607, 1992.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[34] Amarjot Singh and Nick Kingsbury. Efficient convolutional network learning using parametric log based dual-tree wavelet scatternet. In *CVPR workshop*, 2017.

[35] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Scale-equivariant steerable networks. *ICLR*, 2020.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[37] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[38] Nergis Tomen and Jan van Gemert. Spectral leakage and rethinking the kernel size in cnns. *arXiv preprint arXiv:2101.10143*, 2021.

[39] Nergis Tomen, Silvia Laura Pintea, and Jan van Gemert. Deep continuous networks. In *International Conference on Machine Learning (ICML)*, 2021.

[40] Haohan Wang, Xindi Wu, Zeyi Huang, and Eric P Xing. High-frequency component helps explain the generalization of convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8684–8694, 2020.

[41] Thomio Watanabe and Denis F Wolf. Image classification in frequency domain with 2srelu: a second harmonics superposition activation function. *CoRR*, 2020.

[42] Andrew P Witkin. Scale-space filtering. In *Readings in Computer Vision*, pages 329–332. Elsevier, 1987.

[43] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *CVPR*, July 2017.

[44] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1740–1749, 2020.

[45] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[46] Dong Yin, Raphael Gontijo Lopes, Jonathon Shlens, Ekin D Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision. *NeurIPS*, 2019.

[47] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.