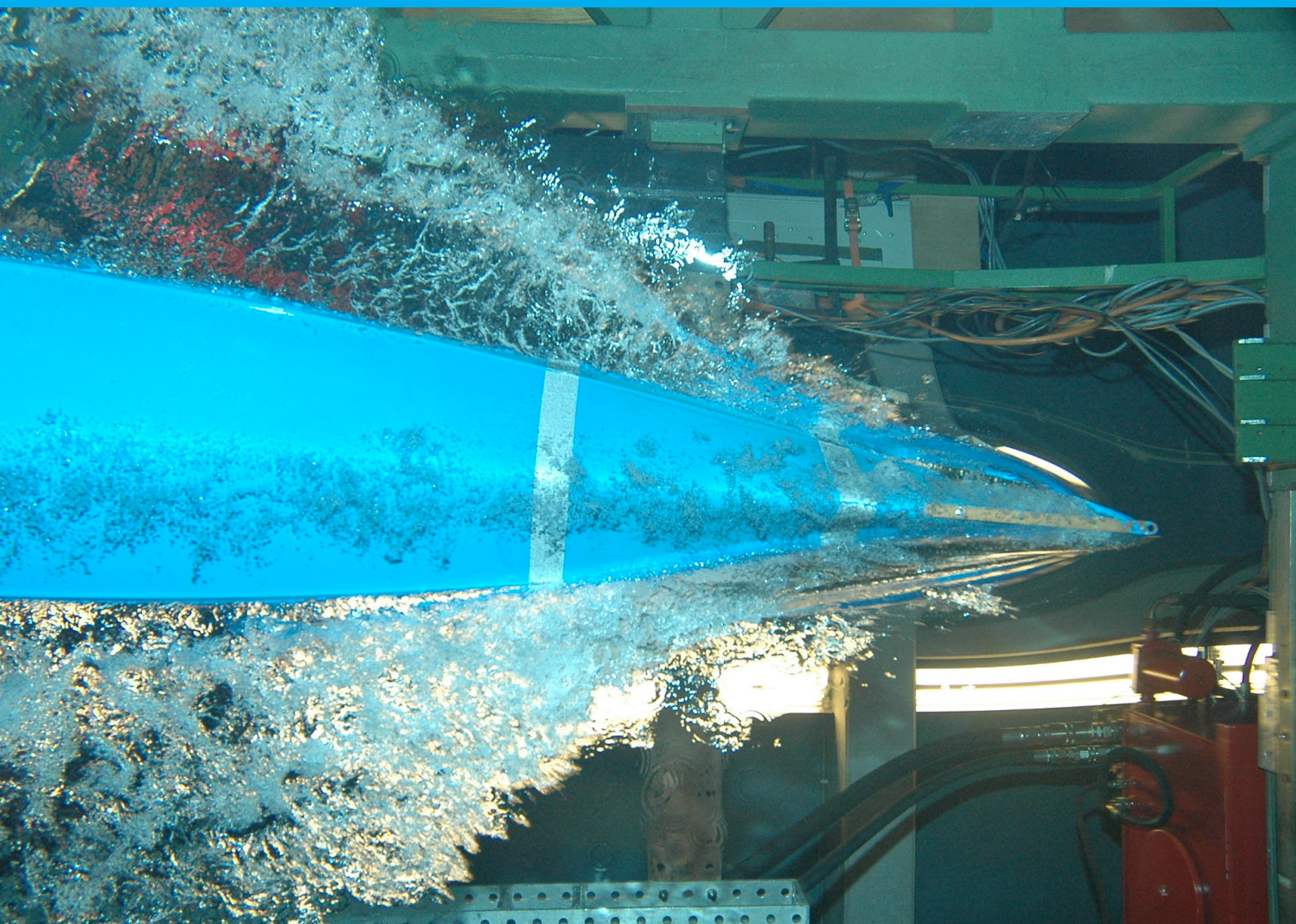
Frequency Learning for Structured CNN Filters

Nikhil Saldanha



Frequency Learning for Structured CNN Filters

by

Nikhil Saldanha

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday July 12, 2021 at 09:00 AM.

Student number:	4998707	
Project duration:	November 1, 2020 – July 15, 2021	
Thesis committee:	Prof. Dr. Jan van Gemert,	TU Delft, Supervisor and Chair
	Prof. Dr. Silvia Laura Pintea,	TU Delft, Supervisor and Committee Member
	Dr. Nergis Tömen,	TU Delft, Supervisor and Committee Member
	Prof. Dr. Frans Oliehoek,	TU Delft, Committee Member

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Preface

This report presents the work of my master's thesis project on the topic of *Frequency learning for structured CNN filters with Gaussian fractional derivatives*. The text is structured in 3 parts, a scientific article presenting our work as submitted to the British Machine Vision Conference (BMVC) 2021, the supplementary material to as provided to the conference, and an appendix that introduces some of the topics of the thesis.

This research was conducted at Computer Vision Lab of Pattern Recognition and Bioinformatics Group in TU Delft under the supervision of Dr. Silvia Pintea, Dr. Nergis Tömen, and Dr. Jan van Gemert.

I would firstly like to thank my daily supervisors Silvia and Nergis. Their guidance and valuable feedback in weekly meetings has helped shape this thesis. They have been a constant source of guidance without whose help I would not have been ambitious enough to submit my work to a conference. I am grateful to Jan for sharing his knowledge and feedback on my work and for helping support my thesis as the Head of the CV lab. I would also like to express my appreciation to Dr. Frans Oliehoek for taking an interest in my work and agreeing to be a part of the evaluation committee.

Nikhil Saldanha
Delft, July 2021

Contents

1 Scientific Article **1**

2 Supplementary Material **15**

3 Appendix **23**

3.1 Images . . . 23

3.2 Image Filtering . . . 23

3.2.1 Image Filtering in the Spatial Domain . . . 25

3.2.2 Image Filtering in the Frequency Domain . . . 26

3.3 Multi-Scale Image Representation and the Scale-Space . . . 28

3.3.1 Scale-Space Theory . . . 28

3.3.2 Gaussian Convolution Operator . . . 30

3.4 Machine Learning and Neural Networks . . . 32

3.4.1 Supervised Machine Learning . . . 32

3.4.2 Fully-Connected Neural Network . . . 33

3.4.3 Convolutional Neural Networks . . . 35

3.5 CNNs with Fixed Filter Banks . . . 36

3.5.1 Scattering Transform . . . 36

3.5.2 Structured Receptive Field (SRF) Network . . . 36

1

Scientific Article

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

A structured CNN filter basis allows incorporating priors about natural image statistics and thus require less training examples to learn, saving valuable annotation time. Here, we build on the Gaussian derivative CNN filter basis that learn both the orientation and scale of the filters. However, this Gaussian filter basis definition depends on a predetermined derivative order, which typically results in fixed frequency responses for the basis functions whereas the optimal frequency of the filters should depend on the data and the downstream learning task. 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 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 on.

1 Introduction

Convolutional neural networks (CNNs) learn filters from the data, where each per-pixel filter value represents a trainable weight. As a parameter-efficient alternative, convolutional filters can be defined as a combination of continuous functions containing certain desirable properties: the ability to learn the orientation selectivity [20, 39], or the scales of the filters [19, 32]. Such structured filters incorporate image priors into the CNN definition. This has the benefit that these priors no longer need to be learned from the data, leading to data efficiency. Here, we build on structured filters for added data efficiency, and specifically on the Gaussian derivative basis [14].

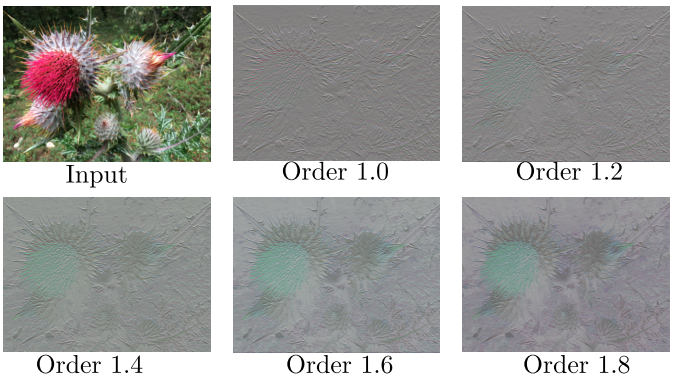


Figure 1: Filter responses when using fractional order Gaussian derivative filters (here x-order and y-order are equal). Defining the filters using fractional derivative orders adds flexibility in terms of the peak response frequency, and enables the use of standard gradient backpropagation for training.

The Gaussian derivative basis defines the CNN filters as a fixed linear combination of Gaussian derivatives up to a fixed hard-coded order. The choice of the Gaussian derivative order restricts the model in terms of the frequencies that it can learn from the data. Selecting a too small order would not allow the model to process high frequency responses. On the other hand, selecting a too high order may lead to over-parametrization if the data does not contain high frequencies, and it comes at large computational costs. In this work, we learn the frequencies present in the data during training.

We make the observation that the order of the Gaussian basis in the structured receptive fields (SRFs) [14] 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 [14], 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 3 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.* [30] define a steerbale pyramid using a set of wavelets that encode orientation and scale, while Mallat defines complex wavelet basis filters in [22]. These complex wavelets have been used in the Scattering transform [1, 23] which is later extended in [5, 25, 29, 31]. Other works consider PCA basis [6], Gabors [20, 26], circular harmonics [39], or simply learning the basis from the data [16]. A large amount of work has been focused on Gaussian derivatives basis [14] used for controlling the scale in deep networks [19, 27, 32] or for making the networks continuous over space and depth [35]. Here, we also build on the Gaussian derivative basis [14] 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 [9, 11, 41]. Alternatively, the use of 1×1 convolutions have significantly reduced the parameters in SqueezeNets [7, 13]. Depth-wise separable convolutions combined with 1×1 convolutions have shown parameter efficiency [3, 12, 21, 43]. More recently EfficientNet [33] 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 [34, 36, 42]. However they tend to learn low frequency signals first [28]. Rather than using frequency domain to analyze deep networks, the networks can actually be trained in the frequency domain [8, 37] or over inputs transformed to the frequency domain [40]. 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 [18, 38] enables the definition of filters as continuous functions [14, 32, 35]. 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 [18, 38] 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 [18]. The recursive formulation relying on Hermite polynomials [24] 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:

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

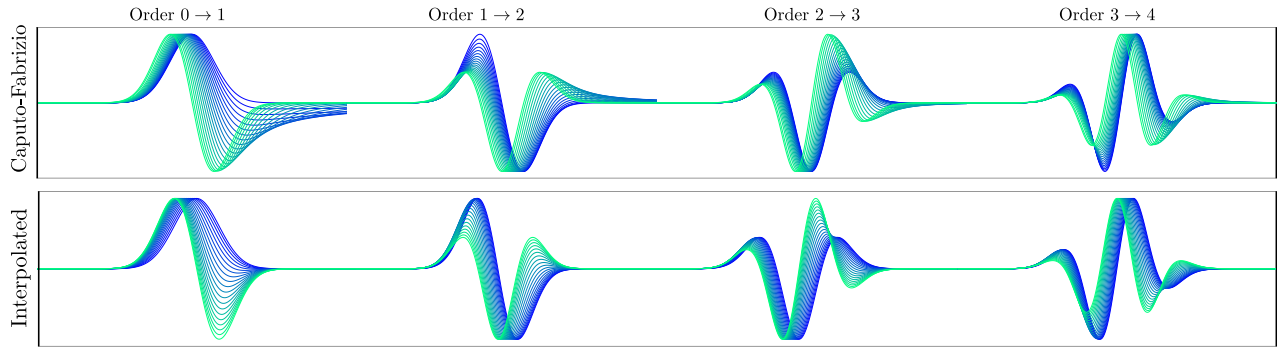


Figure 2: Top: Gaussian derivatives computed using Caputo-Fabrizio [2] fractional derivative form. Bottom: Fractional Gaussian derivatives computed via interpolation between integer orders. The error introduced by using the interpolation is small relative to the Caputo-Fabrizio form.

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

By simplifying Eq. (2) and incorporating the polynomial coefficients in a set of weights α, previous work [14, 35] 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 [27, 35].

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 [14], 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.

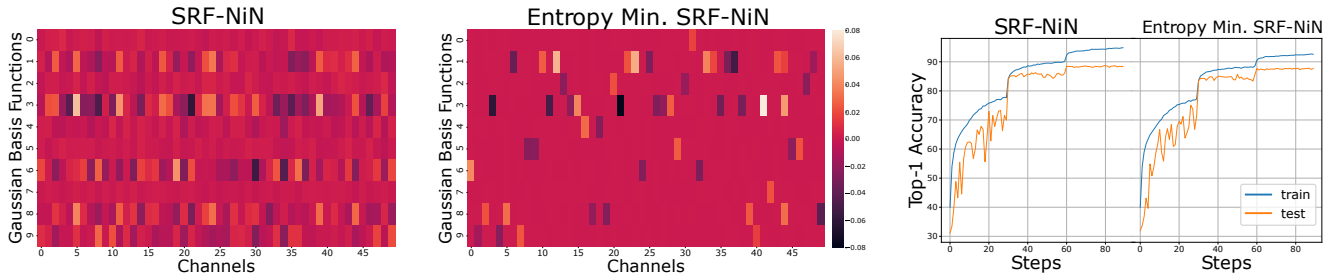
One possible way of dealing with fractional derivatives is the Caputo-Fabrizio [2] form, which in the 1D case is:

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

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



(a) (b) (c)

Figure 3: (a) The distribution of α weights (color bar) learned in a layer of the original SRF-NiN [14] 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.

integers of the fractional order:

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

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^x_{ck}, ν^y_{ck}) per kernel in the filter, and a scalar (α_{ck}) for each kernel:

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

where the scale parameter σ is shared among the kernels in the filter and can either be learned as in [27, 35], or fixed as in [14, 32]. Our method is more flexible than the structured receptive fields (SRF) [14], allowing for non-integer derivatives. We coin our filters FracSRF.

Is one Gaussian derivative sufficient? Unlike previous work [14, 35], 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 [14] over a NiN [17] 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

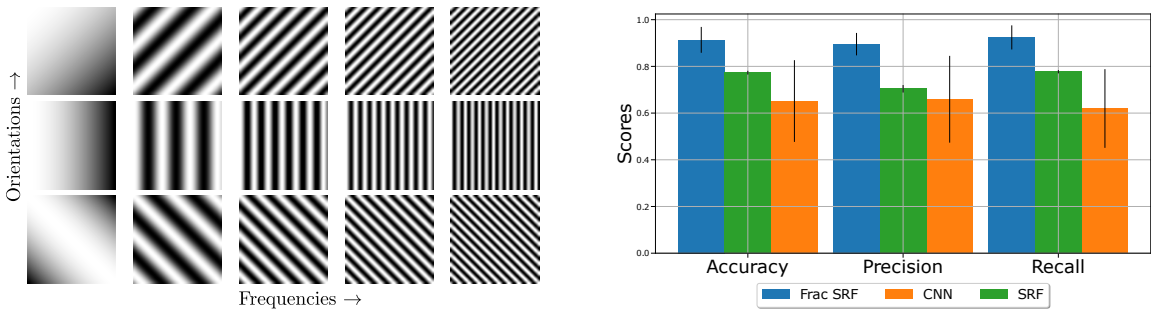
reduces the actual number of Gaussian derivatives used per filter. At no loss in accuracy (Fig. 3.(c)) the number of Gaussian derivatives can be reduced from 9 to 2 per channel. This supports our intuition that using one Gaussian derivative is sufficient, where we make it more flexible by learning its order from the data.

4 Experiments

4.1 Experimental setup

Datasets. We test our method across 3 datasets: *CIFAR-10*, *CIFAR-100* [15], and *STL-10* [4], 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) [17], Resnet-32 [10], EfficientNet-b0 [33]. We also compare with a few methods using structured filters: SRF [14, 27]. 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.



(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.

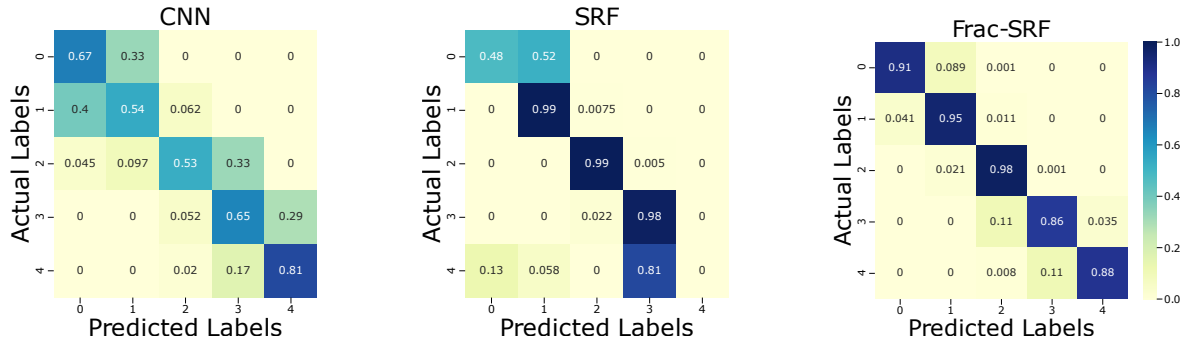


Figure 5: **Exp 1:** Confusion matrices for the CNN, SRF [14], and FracSRF small networks on the *Sinusoids* dataset. Our FracSRF can learn varying frequencies, and therefore it is better at distinguishing the 5 classes.

	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 $\sigma = 2^0$.

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 corresponds 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 $5\times$. 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 [14] 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 [14] 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 [35] 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

	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.

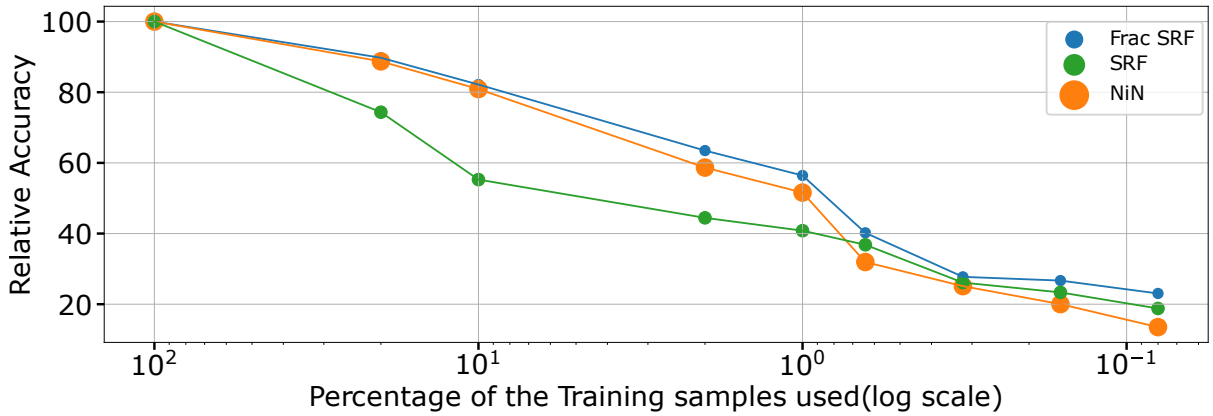


Figure 6: **Exp 3.(a):** Data efficiency in the FracSRF model. Relative accuracy of NiN, SRF-NiN [14, 27], and FracSRF-NiN on subsets of *CIFAR-10*. The dot size of each method indicates the relative number of parameters. The performance of each model are normalized as a percentage of their own accuracy at 100% training data. The scores of our FracSRF-NiN degrade less rapidly especially when compared to NiN and SRF-NiN.

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 $3\times$. Table 2 shows the optimal order initialization is found in the interval $[1,3]$. 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 sub-optimal 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 [14], 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

	NiN [17]	SRF		FracSRF (ours)	
		Fixed scale [14]	Learned scale [27]	Fixed scale	Learned scale
# Params	0.98M	0.5M	0.52M	0.33M	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 [10]	SRF-ResNet-32		FracSRF-ResNet-32 (ours)	
		Fixed scale [14]	Learned scale [27]	Fixed scale	Learned scale
# Params	0.47M	0.30M	0.31M	0.15M	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%
	EfficientNet-b0 [33]	SRF-EfficientNet-b0		FracSRF-EfficientNet-b0 (ours)	
		Fixed scale [14]	Learned scale [27]	Fixed scale	Learned scale
# Params	3.6M	3.47M	3.48M	3.43M	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* and *STL-10* datasets when comparing the baseline NiN, Resnet-32 and EfficientNet-b0 with their SRF variants [14, 27] 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.

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 [14] and learned scale [27], on *CIFAR-10*, *CIFAR-100* and *STL-10*. Table 3 reports accuracies and number of parameters. Our FracSRF layer achieves comparable performance to standard convolutional 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. 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 Conclusion

We propose structured basis filters based on Gaussian derivatives, where we learn the frequencies present in the data by learning the orders of the Gaussian derivatives. We show experimentally that our model can learn the correct frequencies from the data. Moreover, our model degrades gracefully with fewer training samples, and it can achieve good accuracy at large parameter reductions.

One of the limitations of our model is that computations drastically increase with order: because we rely on the recursive Hermite polynomials to define the Gaussian derivatives. 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 2 parameters per kernel: the scale and the order.

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

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

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

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

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

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

[21] 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.

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

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

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

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

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

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

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

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

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

[36] 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.

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

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

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

[40] 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.

[41] 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.

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

[43] 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.

2

Supplementary Material

Frequency learning for structured CNN filters with Gaussian fractional derivatives (Supplementary)

BMVC 2021 Submission # 1415

1 Fractional-order derivatives

This section provides a narrow introduction to fractional calculus as required for understanding our approach. Let us take a function $f(x)$ which we would like to differentiate with respect to the variable x. This can be written as follows:

$$\frac{d}{dx}f(x) = D[f(x)] = D_x f. \tag{1}$$

$\frac{d}{dx}$ is the derivative operator with respect to the variable x. This formulation is known as a first-order derivative of f with respect to x. $D_x f$ is a short-hand for the same. We can successively differentiate the function $f(x)$ n times to get an n^{th} order derivative. Here we are only concerned with positive integer orders, i.e. $n \in \mathbb{Z}^+$:

$$\frac{d^n}{dx^n}f(x) = D_x^n f. \tag{2}$$

The simplest definition of the fractional derivative operator would be one that can operate on integers as well as fractions. Let us use ν to represent real numbers that can take on integer or fractional values. Hence, in order for the fractional derivative operator $D^\nu(\cdot)$, $\nu \in \mathbb{R}$ to be properly defined, it should be accurate for integer values as well. One of the best and simplest tests for this is that the composition of derivatives of two fractional orders ν_1 and ν_2 should be equivalent to the derivative of the sum of the orders $\nu_1 + \nu_2$ of the composing derivatives:

$$D^{\nu_1}(D^{\nu_2} f) = D^{\nu_1+\nu_2} f. \tag{3}$$

In principle, fractional derivatives should be able to define normal derivatives Df and also integrals $\int f$:

$$D^{\frac{1}{2}}(D^{\frac{1}{2}} f) = Df \tag{4}$$

$$D^{-1} f(x) = \int f(x)dx. \tag{5}$$

Note that there is no single form of a fractional derivative and much like integrals or even regular differentials, there are different ways of defining fractional derivatives for different types of functions. In the next section, we review fractional derivatives of some common functions.

1.1 Fractional derivatives of polynomial functions

We can first break up polynomials as terms that look like ax^k and then each term can be differentiated separately using the chain rule of differentiation:

$$Dax^k = akx^{k-1} \tag{6}$$

$$D^2ax^k = ak(k-1)x^{k-2} \tag{7}$$

$$\vdots$$

$$D^nax^k = a\frac{k!}{(k-n)!}x^{k-n}, k \geq n. \tag{8}$$

When considering non-integer values of the order ν, the factorial function does not have solutions for all real numbers. To get around this issue, we can use the Gamma function $\Gamma(\cdot)$ which accepts as input all positive real numbers:

$$D^\nu ax^k = a\frac{\Gamma(k)}{\Gamma(k-\nu)}x^{k-\nu}. \tag{9}$$

1.2 Fractional derivatives of exponential functions

We can expand exponentials as an infinite series of polynomials and apply the same technique as in the previous section:

$$D^\nu e^x = D^\nu \sum_{i=0}^{\infty}\frac{x^i}{i!} = \sum_{i=0}^{\infty}\frac{x^{i-\nu}}{(i-\nu)!}, \tag{10}$$

where again for non-integer values of ν, we use the Gamma function $\Gamma(\cdot)$:

$$D^\nu e^x = \sum_{i=0}^{\infty}\frac{x^{i-\nu}}{\Gamma(i-\nu)}. \tag{11}$$

We can combine the definitions of fractional order derivatives of polynomials and exponentials to define the fractional order derivative for the Gaussian family of functions.

1.3 Fractional-order derivative of Gaussians

1.3.1 Using polynomial expansion

We first express the Gaussian as a Taylor Series (about $x = 0$) to get the polynomial terms:

$$\begin{aligned} G(x;\sigma=1,\mu=0) &= \frac{1}{\sqrt{2\pi}\sigma}exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}} - \frac{x^2}{2\sqrt{2\pi}} + \frac{x^4}{8\sqrt{2\pi}} - \frac{x^6}{48\sqrt{2\pi}} + \cdots \\ &= \sum_{k=0}^{\infty}\frac{(-1)^k}{2^k k!}\cdot\frac{x^{2k}}{\sqrt{2\pi}} \end{aligned} \tag{12}$$

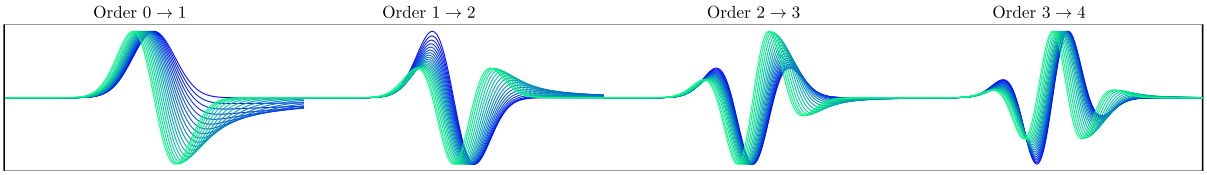


Figure 1: Fractional Order Derivatives of Gaussians, implemented using the Caputo-Fabrizio closed-form. Results are the same as those obtained by authors in [1]

.

Each term is now a polynomial term whose fractional order derivative can be described in terms of equation 9:

$$D^{\nu}G(x;\sigma=1,\mu=0) = \sum_{k=0}^{\infty} \frac{-1^k}{2^k k!} \cdot \frac{\Gamma(2k+1)}{\Gamma(2k-\nu+1)} \cdot \frac{x^{2k-\nu}}{\sqrt{2\pi}}. \tag{13}$$

The drawback of this approach is that it yields complex values for $x < 0$. This works perfectly fine at integer orders but does not give correct values for negative x for fractional orders, since there we cannot deal with the imaginary component. One possible solution is to shift the Gaussian in the positive direction so we do not encounter negative values. The resulting Gaussian derivative should be identical in shape to the one at $x = 0$ since the mean of a Gaussian only determines its shape. By expanding the series about some point $x = a$, we get the following expression:

$$\begin{aligned} G(x;\sigma=1,\mu=0) = {} & \frac{ae^{-a^2/2}(x-a)}{\sqrt{2\pi}} + \frac{(a^2-1)e^{-a^2/2}(x-a)^2}{2\sqrt{2\pi}} \\ & - \frac{(a(a^2-3)e^{a^2/2})(x-a)^3}{6\sqrt{2\pi}} + \cdots \end{aligned} \tag{14}$$

Unfortunately, there is no closed-form expression for this expansion which is compounded by the fact that each of the terms of the expansion contains an exponential in the form of the Gaussian which if expanded again will generate more exponential terms. Hence we abandon this approach entirely and consider next the Caputo-Fabrizio form of the fractional derivative.

1.3.2 Using Caputo-Fabrizio closed-form

One of the more popular forms of the fractional derivative is called Caputo-Fabrizio (CF) fractional derivative [3, 4, 6]. We are interested in this form since prior work has shown to be able to capture multiple types of functions. One of the advantages of this form is that it allows us to separate the fractional part of the derivative from the integer order derivative and compose them as required. It has also been shown in previous work that a closed-form expression for the Gaussian derivative is possible [1]. The final closed-form expression looks as follows:

$${}^{CF}D_x^{n+\nu}G(x) = \frac{1}{1-\nu}\sum_{k=1}^{n}\left[\left(-\frac{\nu}{1-\nu}\right)^{n-k}\partial_x^k G(x)\right] + \left(-\frac{\nu}{1-\nu}\right)^{n} {}^{CF}D_x^{\nu}G(x) \tag{15}$$

where ${}^{CF}D_x^{n+\nu}G(x)$ denotes the $n+\nu$ order Caputo-Fabrozio derivative of $G(x)$ with respect to x. n is the integer part of the order and ν is the fractional part. The first term on the right

hand side is the integer-order part, which is a straightforward integer-order derivative and can be computed using the Hermite polynomials. The second term is the fractional-order part of the derivative which can be expressed using the Caputo-Fabrizio derivative form [6]. The CF derivative of an arbitrary function $f(x)$ can be expressed as:

$$ {}^{CF}D_x^{\nu} f(x) = \frac{1}{1-\nu} \int_0^x \exp\left(\frac{\nu}{1-\nu}(x-\tau)\right) \partial_x f(x) \Big|_{x=\tau} d\tau \tag{16} $$

Therefore, the CF derivative of a Gaussian can be expressed as:

$$ \begin{aligned} {}^{CF}D_x^{\nu} G(x;\mu,\sigma) &= \frac{1}{1-\nu} \int_0^x (\mu-\tau) \exp\left(-\frac{\nu}{1-\nu}(x-\tau)\right) \left[\frac{\mu-\tau}{\sigma^2} G(\tau;\mu,\sigma)\right] d\tau \qquad (17) \\ &= \frac{1}{(1-\nu)\sqrt{(2\pi)\sigma^3}} \int_0^x (\mu-\tau) \exp(-\frac{1}{2\sigma^2}\left(\tau - \left(\frac{\nu}{1-\nu}\sigma^2 + \mu\right)\right)^2 \\ &- \frac{\nu}{1-\nu}\left[x - \mu - \frac{\sigma^2}{2}\left(\frac{\nu}{1-\nu}\right)\right]) d\tau \end{aligned} $$

To make the equation simpler to read, we can separate out the integral into a new term that we call $\zeta_{\mu,\sigma,\nu}(x)$

$$ {}^{CF}D_x^{\nu} G(x;\mu,\sigma) = \frac{1}{1-\nu} \cdot \frac{1}{\sqrt{2\pi}\sigma^3} \exp\left(-\frac{1}{1-\nu}\left(x - \mu - \frac{\sigma^2}{2} \cdot \frac{1}{1-\nu}\right)\right) \zeta_{\mu,\sigma,\nu}(x) \tag{18} $$

$$ \zeta_{\mu,\sigma,\nu}(x) = \int_0^x (\mu - x) \exp\left(-\frac{\left(\tau - \mu + \frac{1}{1-\nu}\sigma^2\right)^2}{2\sigma^2}\right) d\tau \tag{19} $$

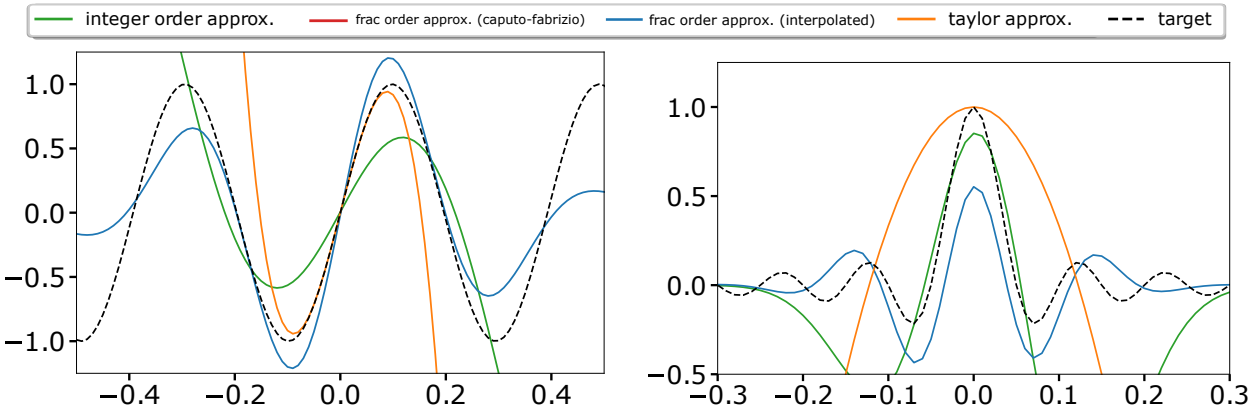
To implement the fractional derivative in closed-form, we need to approximate the integral $\zeta_{\mu,\sigma,\nu}(x)$ using cumulative trapezoid or adaptive quadrature integral approximation methods [2]. The fractional derivatives of Gaussians computed using the CF form can be seen in figure Fig. 1 which are the same as results obtained by the authors. We can observe that the fractional-order smoothly transition between the integer orders which is a crucial requirement.

2 Additional experiments

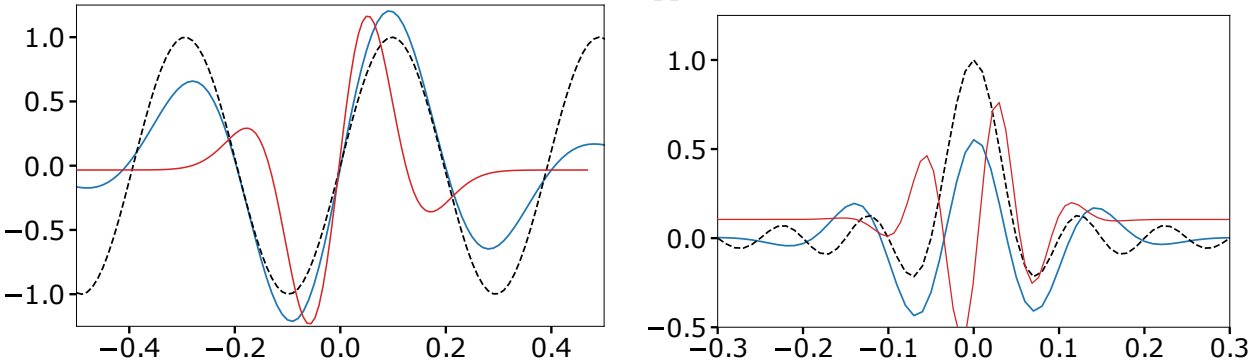
2.1 Approximation of 1D functions

Experimental Setup. Before we embed our method into a CNN, we would like to test our proposed scaled fractional Gaussian derivative in a simpler 1D setting. To do this, we create a test scenario where we have 1D target functions which we would like to approximate with a fractional Gaussian basis of a single learned order (our approach) and compare this with the standard Taylor approximation of a function and previously proposed linear combination of integer order Gaussian derivatives as used in [5]. While the 'integer order' method learns the optimal coefficients for the fixed basis set required to fit the function, the 'fractional order' method learns a single order. Both methods learn via gradient descent to optimize a squared error loss. The Taylor and the 'integer order' method approximate the function with 3 terms.

We hypothesize that our approach is better at approximating the functions than other approaches because the other methods rely on a large number of terms (in the case of Taylor



(a) 1D Function approximation



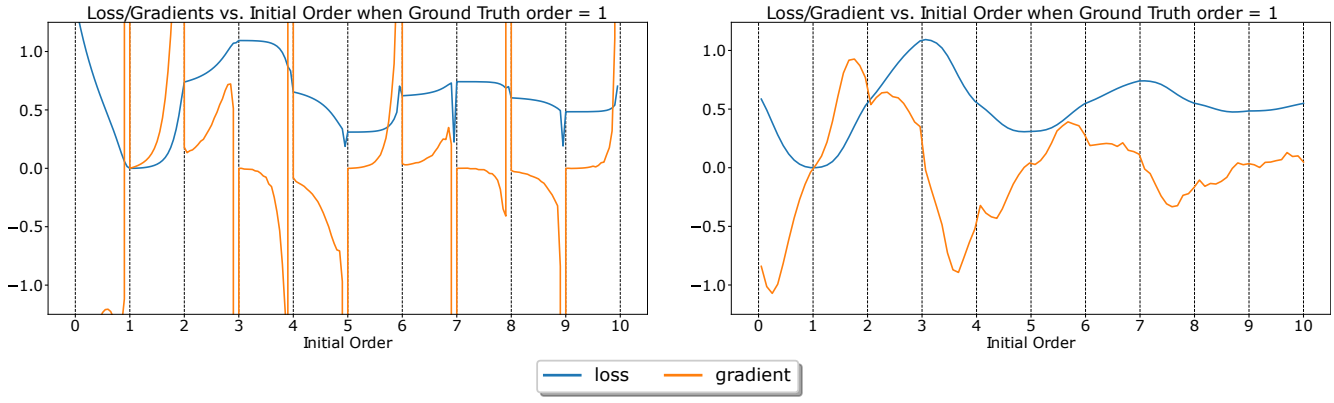
(b) Caputo-Fabrizio vs. Interpolated Fractional Derivative

Figure 2: **1D function approximation.** (a) Approximating two target functions (dashed lines) when using: a Taylor approximation, a linear combination of integral Gaussian derivatives, and a scaled fractional Gaussian derivative (computed by interpolating between integer orders). Our fractional Gaussian derivatives can accurately approximate the function. (b) Comparison between the approximations by scaled fractional Gaussian derivative computed via the Caputo-Fabrizio formulation and the interpolated method shown in (a).

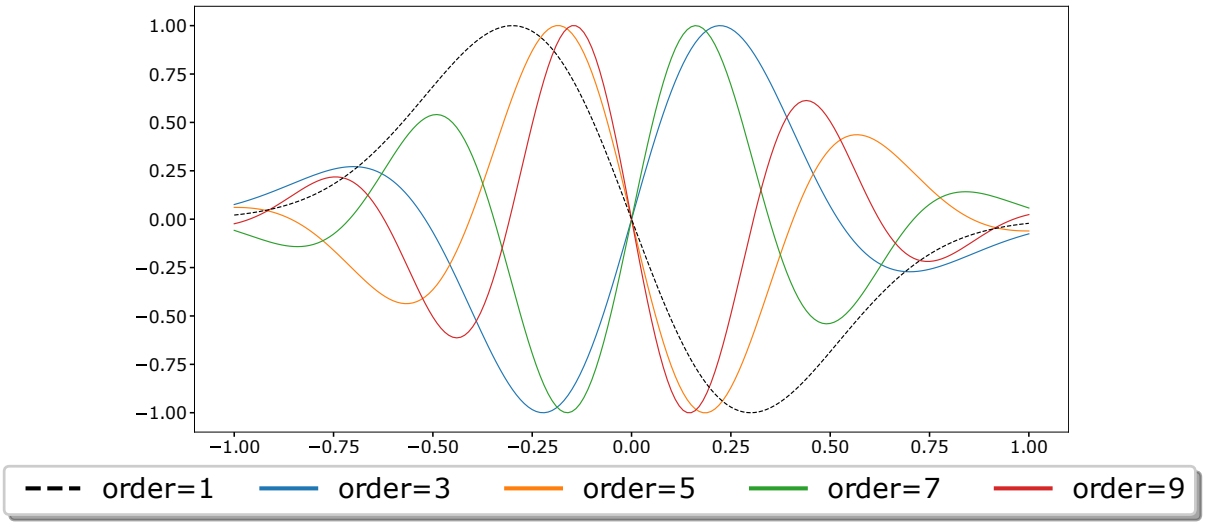
approximation) or orders (in the case of the linear combination of Gaussian derivatives), while our method is can learn the correct order via gradient descent. This is more true for functions with a higher frequency which can still be approximated with a single learned order rather than relying on a large number of fixed terms. We experiment with two different variations of our approach. The first by making the fractional-order derivatives using the Caputo-Fabrizio form and the next by the interpolating between integer orders.

Accuracy of Approximation In Fig. 2. (a) can be seen how the interpolated variant of our method performs against the integer-order and Taylor approximation. We show the results here for the sinusoid and the sinc function where our method can better approximate the functions than the other methods. While the other methods are accurate only towards the centre (which is expected), our method also performs relatively well a few standard deviations away. This result is promising as we expect this to translate into better results than SRF networks in 2D where the filters made of our Fractional Gaussian basis will try to approximate local image functions.

Caputo-Fabrizio (CF) vs interpolated fractional derivatives In Fig. 2. (b), we show some comparisons between the approximation of the CF and the Interpolated Fractional Derivative. We consistently noticed that the performance of the CF variant is lower than the interpolated one. Additionally, we noticed that the CF Derivative is highly sensitive to the initial conditions and would sometimes never update the order. Sometimes, we observe a numerical underflow of the gradients, and so we needed to clip the gradients. We investigate why such



(a) CF Fractional Derivative (b) Interpolated Fractional Derivative



(c) Aligning of peaks of odd-ordered Gaussian Basis to the Ground-Truth order (dashed line)

Figure 3: **Gradients analysis.** Loss and gradient landscape of CF fractional derivative: (a) and 'interpolated fractional derivative' method; (b) methods for different initialization of order when approximating a Gaussian basis of order = 1 (ground truth order). The reason for the local minima in odd orders in the loss landscape is that they are aligned with the peaks and troughs of the odd order Gaussian derivative as shown in (c).

errors happen in the next section.

2.2 Gradients of fractional derivative methods

Experimental Setup. In this section, we analyse the gradients and loss landscape of the CF-Derivative and 'interpolated derivative' methods. This is in an effort to understand why the CF Derivative method is so sensitive to the initial order and the final approximation is far below the 'interpolated derivative' method. This is even more surprising considering that both derivatives smoothly transition between orders of Gaussian basis functions.

We initialize both methods at different fractional orders ranging from 0 to 10. At each of these initial conditions for both methods, we record the value of the loss function and the gradient of the loss with respect to the order. Importantly, we do not perform any gradient descent steps. The function that we task each method with approximating is simply a Gaussian derivative of arbitrary order which we call the Ground Truth order. In doing this, we would like to eliminate all external factors that could influence the difference in performance between the two methods. We also do not learn the coefficients of the basis for the 'integer order' model and make sure that the target Gaussian is scaled appropriately. We plot the gradient and loss values for each initial order condition. We expect that when the initial order

matches the ground-truth order, the loss goes to zero. Additionally, to the left and right of the optimal order, the loss should be convex. We would hope to see smooth gradients that do not grow suddenly to infinity or diminish to zero.

Analysis of Gradient and Loss Landscape. In figure Fig. 3, (a) and (b) we see the plots of the loss and gradients with respect to different initial orders. The gradients and loss landscape of the 'interpolated order' method are as described and expected for a well-behaved method, while the gradients and loss functions of the CF fractional derivative method are not so well behaved. We notice exploding gradients while the initial order increases and approaches an integer order. This lends an explanation for the numerical underflow explained in the previous section. The loss function is not exactly smooth but has several peaks and sharp drops which could explain why the method was so sensitive to initial conditions. Another interesting phenomenon is the presence of the sharp drops close to odd integer locations which can be explained by the fact that the peaks of odd-ordered Gaussian derivatives align to reduce the loss. This can be seen illustrated in Fig. 3.(c).

References

[1] Jorge M. Cruz–Duarte, Juan Rosales–Garcia, C. Rodrigo Correa–Cely, Arturo Garcia–Perez, and Juan Gabriel Avina–Cervantes. A closed form expression for the gaussian–based caputo–fabrizio fractional derivative for signal processing applications. *Communications in Nonlinear Science and Numerical Simulation*, 61:138–148, 2018. ISSN 1007-5704. doi: https://doi.org/10.1016/j.cnsns.2018.01.020.

[2] Walter Gander and Walter Gautschi. Adaptive quadrature—revisited. 40(1):84–101. ISSN 1572-9125. doi: 10.1023/A:1022318402393. URL https://doi.org/10.1023/A:1022318402393.

[3] J.F. Gómez-Aguilar, Teodoro Cordova, Jesús Escalante-Martínez, C.M. Calderon-Ramon, and R. Escobar Jiménez. Electrical circuits described by a fractional derivative with regular kernel. *Revista Mexicana de Fisica*, 62:144–154, 03 2016.

[4] Jordan Hristov. Transient heat diffusion with a non-singular fading memory: From the cattaneo constitutive equation with jeffrey's kernel to the caputo-fabrizio time-fractional derivative. *Thermal Science*, 20:19–19, 07 2016. doi: 10.2298/TSCI160112019H.

[5] 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.

[6] R. Khalil, M. Al Horani, A. Yousef, and M. Sababheh. A new definition of fractional derivative. 264:65–70. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2014.01.002. URL https://www.sciencedirect.com/science/article/pii/S0377042714000065.

3

Appendix

3.1. Images

From a layman's perspective, an image is an artificial representation of something perceived through the visual senses, most commonly as a photograph, painting, or sculpture. In this text, we define an image from the perspective of image processing. In image processing, specifically digital image processing, we are concerned with using computers to performing operations on digital representations of images without interpreting them in a semantic way. In the context of image processing, an image is also a representation of visual perception but modelled as a mathematical function over a continuous spatial domain. The mapping onto a spatial domain is important to take full advantage of special properties of signals which we can use to manipulate images in information systems.

We mathematically define an image I as a function that maps a spatial domain S to a range of intensity values V, $I : S \rightarrow V$. In theory, the spatial domain S does not have to be restricted to just 2D images as we commonly know them. The mathematical formulation allows us to represent higher dimensional images such as CT scans that capture 3D space. But in this text, we will consider the 2D continuous Euclidean space $\mathbb{R}^2$ as our domain. Depending on type of sensor that captured the image or the image processing task to be performed, there are multiple ways to define the range of intensity values. In some cases you could consider multiple ranges for the same image in order to apply different algorithms. The two most common ranges are:

Grayscale: As discussed above, the range depends on the type of sensor that captures the image. For grayscale values, the sensors directly map their measurements at each point to a continuous real scale $\mathbb{R}$. The lowest and highest sensor reading value is mapped to the lowest and highest number on the intensity scale. For representational purposes, the highest intensity is mapped to white and lowest to black with different shades of gray in between. This is shown in figure 3.1. (a)

Color: For color images, each point on the spatial domain typically corresponds to 3 intensity measurements, red, green, and blue. The three measurements can be combined by displays to try and match the colors of the scene that was originally captured by the sensor. This is illustrated in figure 3.1. (b).

3.2. Image Filtering

We can broadly categorize the operations that can be performed on images as *filtering* and *warping* based on what part of the image the operation is performed on.

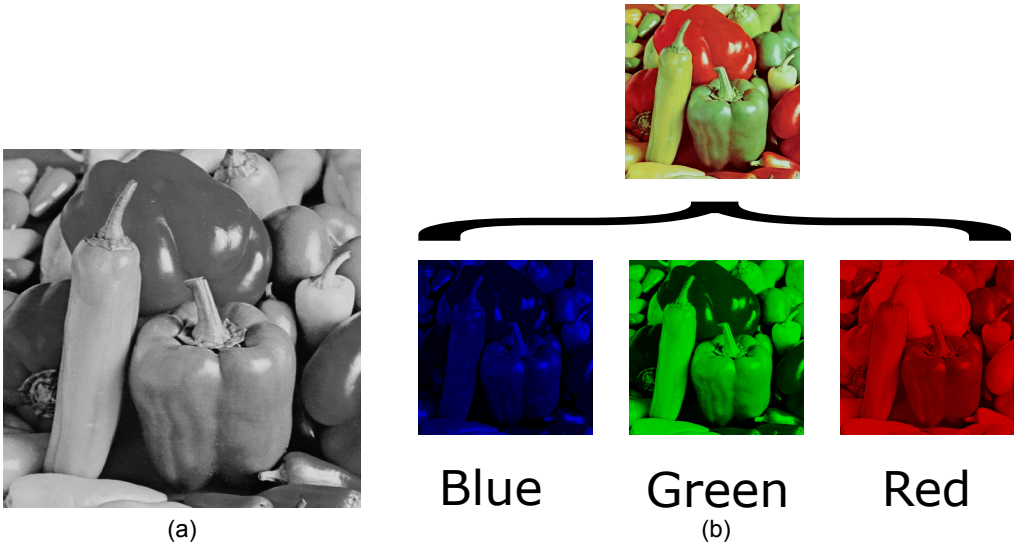


Figure 3.1: grayscale images (a) have a single range of values while RGB Color Images have 3 measurements — red, green and blue — which are combined to create the color image.

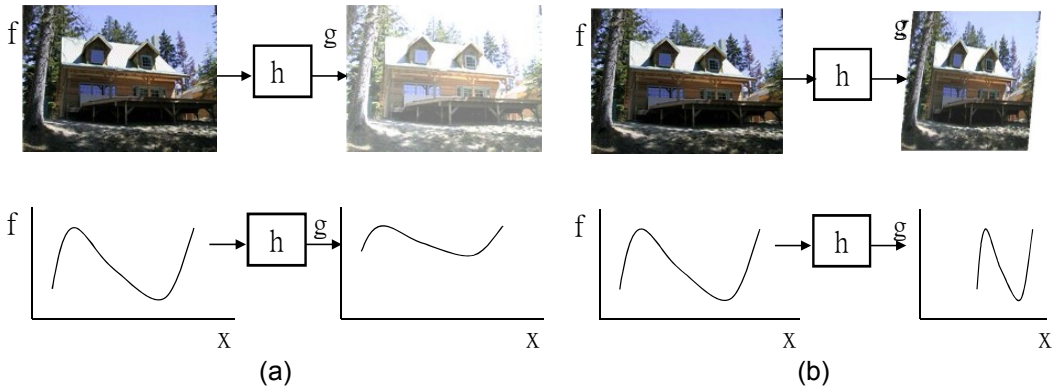


Figure 3.2: Filtering (a) operates on the range of intensities while warping (b) modifies the image domain. Image adapted from Szeliski [23].

Filtering: Let us recall the mathematical definition of an image: $I : S \rightarrow V$. Filtering operations change the range (or intensity values) V of the image. This can be defined mathematically as a mapping, h from the range of values V of the original image I to a new range of values V' to obtain the filtered image I'. An example of this can be seen illustrated in figure 3.2 (a). Filtering operations are usually done to modify image properties such as contrast, sharpness, brightness, and so on. Additionally, filtering also allows the extraction of information from images such as key points, edges, and corners.

Warping: Operations that change the domain S (or pixel positions) of the image. An example of a warping operation is shown in figure 3.2 (b) where the values of the image do not change but the domain is shifted.

In this section, we will only focus on image filtering. We can further classify filtering operations based on the domain in which it is performed. Filtering can be performed in either the spatial or the frequency domain.

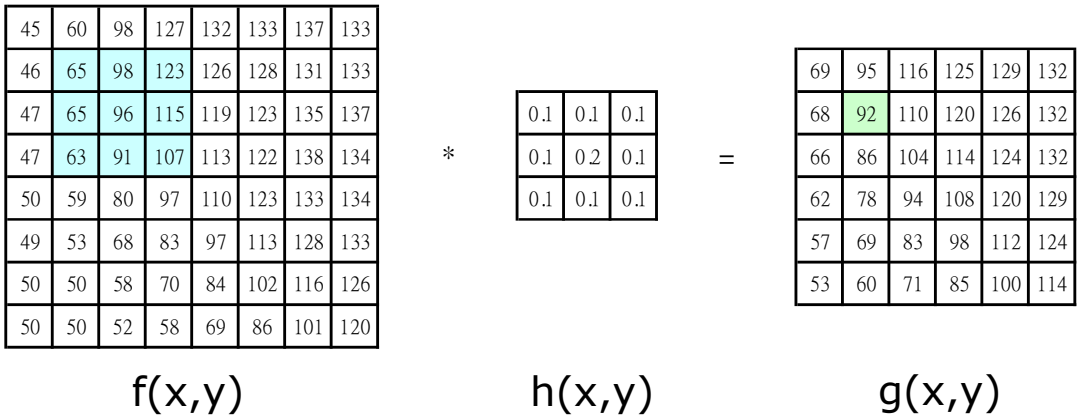


Figure 3.3: Convolution: image on the left $f(x,y)$ is the input image which is convolved with the filter $h(x,y)$ in the center that results in the image on the right $g(x,y)$. The filter operates on a neighbourhood of the image as shown in blue to produce an output value shown in green.

3.2.1. Image Filtering in the Spatial Domain

When filtering images in the spatial domain, we are essentially modifying the values of the image at each position as a function of a local neighborhood of values surrounding the position. Filtering operations that involve fixed weighted combinations of pixels in the local neighborhood are called *linear filtering operations*. The linear filtering operation can be formulated as follows:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{3.1}$$

where the input image is f, the filter is h whose values are known as the filter coefficients. The filtered result is given by g. This can be seen illustrated at 3.3.

A popular variant of this filter operation, known as convolution can be described as follows:

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l) \tag{3.2}$$

here we have just offset all of the coefficients by k and l. This is the standard convolution operator which can be simply written as $g = f * h$, where h is the impulse response function.

Convolution has additional properties that it is commutative and associative. Another useful property is that the Fourier transform of the result of convolution is equivalent to the product of the Fourier transform of the original images. Convolution is also linear shift-invariant(LSI) due to which it obeys the superposition principle allowing the commutation of convolution across a weighted sum of images:

$$h * (f_1 + f_2) = h * f_1 + h * f_2 \tag{3.3}$$

Linear filters are quite easy to compose due to this property and are suitable for frequency response analysis as we will discuss in the next subsection. Additionally, convolution follows the shift-invariance principle due to it being an LSI operator. This allows the shifting of the signal to commute with convolution:

$$g(i,j) = f(i+k, j+l) \iff (h*g)(i,j) = (h*f)(i+k, j+l) \tag{3.4}$$

The shift-invariance property of convolution means that convolution behaves the same at every point in the image. We can imagine that this is useful when, for example, we apply a filter on images to detect a particular object, and the response is similar anywhere in the image.

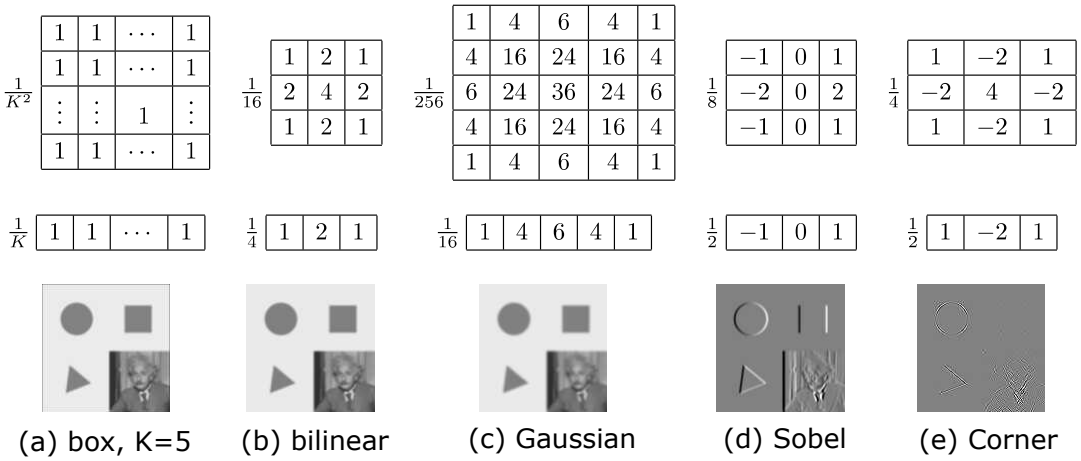


Figure 3.4: The 2D filter kernel (top), the corresponding 1D kernel (middle) and the filtered image (bottom) for 5 different separable linear filters. Adapted from Szeliski [23]

.

In the figure 3.4, we see some examples of the simplest linear filters. The **box filter**, also known as the *average filter*, computes the mean of the pixel values in every $K \times K$ window to get a smoother image than the original. We can achieve a smoother result if we convolve with a tent function (Bartlett filter) [18]. The **bilinear filter** is a 3x3 version of the tent function. For an even smoother result, we can use the **Gaussian Kernel** which is the result of convolving the linear tent function with itself or iterated convolution of the box filters [7, 5]. These are all examples of smoothing filters, also called low-pass filters because they pass the low frequencies to the output but remove high frequencies up to a certain extent. We visualize this idea in the next section.

As mentioned earlier, linear filtering can be used for feature extraction. These perform a kind of image derivative to find where the image value changes sharply. One of the simplest edge extracting filters is the **Sobel Filter** [22], which is composed of a separable combination of a horizontal central difference filter (to compute the horizontal derivative) and the vertical linear tent filter (to smooth the derivative). It makes sense that vertical edges are extracted with the Sobel filter. The **corner detector** computes both a horizontal and vertical derivative to extract corners and also diagonal edges.

As a note, we can also have non-linear filtering operators, which perform a non-linear combination of the neighborhood of pixels. This often leads to superior performance for specific tasks such as edge-preserving filters. The downside of these filters is that they are not easy to compose and computationally expensive. We will not be going into the details of non-linear filters in this text. This is explained in detail in other works [23].

3.2.2. Image Filtering in the Frequency Domain

In this section, we show how Fourier transforms can be used to analyze the characteristics of filters by looking at the frequencies present in an image. This is not by any means an in-depth introduction to Fourier transforms. Refer to other works that cover this topic more comprehensively [9, 23].

To analyse how a filter affects the spatial frequencies present in an image, we use a sinusoid of known frequency is affected by the filter. The reason we use sinusoids is because according to Fourier, ”Any univariate function can be written as a sum of sines and cosines of different frequencies”. Let $s(x)$ be the input sinusoid of frequency f, angular frequency ω and phase ϕ_i:

$$s(x) = sin(2\pi f x + \phi_i) = sin(\omega x + \phi_i) \tag{3.5}$$

If we convolve the input sinusoid with a filter, we get as an output a sinusoid with a different phase

and magnitude:

$$o(x) = h(x) * s(x) = Asin(\omega x + \phi_o) \tag{3.6}$$

where A is the magnitude or gain of the filter and $\phi_i - \phi_o$ is the phase shift of the filter.

This relationship can be explain due to the fact that convolution is defined as a linear combination of shifted inputs. The linear combination of shifted sinusoids of equal frequencies is equivalent to a single sinusoid of the same frequency. Complex number theory is used to more compactly represent this same relationship:

$$s(x) = e^{jwx} = cos(\omega x) + jsin(\omega x)$$
$$o(x) = h(x) * s(x) = Ae^{jwx+\phi}$$

The Fourier transform $\mathcal{F}\{\cdot\}$ is simply a way to measure the phase and magnitude response for different spatial frequencies of the input:

$$H(\omega) = \mathcal{F}\{h(x)\} = Ae^{j\phi} \tag{3.7}$$

where $H(\omega$ is the response of the filter $h(x)$ to a complex sinsuoid of frequency ω.

A closed-form solution for the Fourier transform is given for both discrete

$$H(\omega) = \int_{-\inf}^{\inf} h(x)e^{-j\omega x}dx \tag{3.8}$$

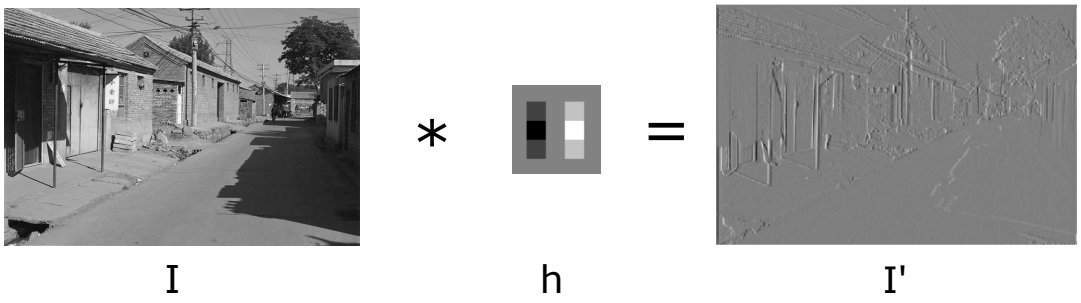
and continuous setting.

$$H(k) = \frac{1}{N}\sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi kx}{N}} \tag{3.9}$$

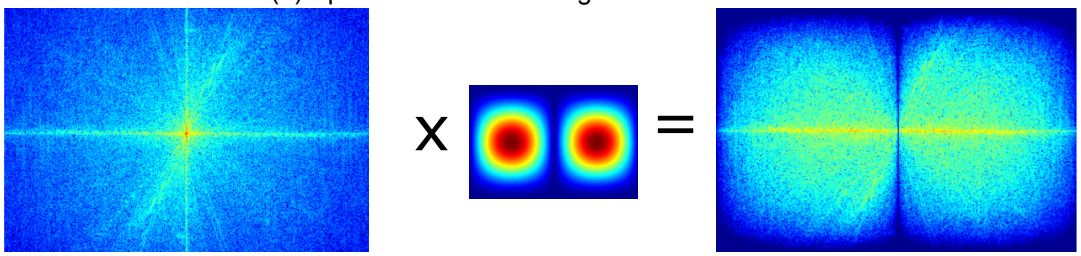
Apart from the mathematical formulation, we would like to get an intuition for the Fourier Domain and the Fourier Transform. In an attempt to do this, we first show a comparison between filtering operations done in the spatial domain and the Fourier Domain. Filtering in both domains is equivalent as it is possible to transform the result from one domain to the other. Filtering in the frequency domain can be performed by a point-wise multiplication between the Fourier transformed input image and the Fourier transformed kernel. This is a well-established result in image processing [23].

Spatial vs. Frequency Domain Filtering In the figure 3.5 (a) we can see how the Sobel kernel filters the image in the spatial domain. It extracts the vertical edges in the image which are denoted by high intensity (white) along the electricity poles and doorposts. In figure 3.5 (b) we see how the same filtering is performed in the frequency domain. We first take the Fourier transform of the input image and the filter. The highest intensities in the frequency domain images are represented in red, while the lowest are in blue. The lowest frequencies are present towards the center of the frequency domain image, while the highest frequencies are present toward the edges. As we move outwards from the center in a horizontal direction in a Fourier transformed image, we encounter increasing frequencies in the horizontal direction. Correspondingly, we encounter an increase in vertical frequencies as we move vertically out from the center. We can see that the output image in the frequency domain has its vertical frequencies modified. Additionally, many high-frequency details, corresponding to the outer edges of the frequency domain, are suppressed. This tracks well with our observations of the result of filtering in the spatial domain.

High, Low and Band-pass filtering in the Frequency Domain As a final exercise, let us see how high-pass, low-pass, and band-pass filters look in the Fourier domain so we can get an intuition for



(a) Spatial Domain Filtering with a Sobel Filter



(b) Frequency Domain Filtering with a Sobel Filter

Figure 3.5: Shows the comparison between filtering in the spatial domain and filtering in the frequency domain (both using the sobel filter). Adapted from [15]

why they operate the way that they do. Figure 3.6 shows this comparison. Low-pass filters let low-frequency details pass through to the result and remove the high-frequency noise. This corresponds to high-intensity values towards the center of the Fourier Transform of the low-pass filter while the edges are low intensity (black) as shown in the center image of figure 3.6 (a). A high-pass filter behaves in the opposite way. It removes details corresponding to low frequencies from the image while high frequencies are let through. This corresponds to low intensity in the center of the Fourier transformed filter, as can be seen in the center image of figure 3.6 (b). Figure 3.6 (c) shows the results of a band-pass filter. Here a small section of frequencies is let through, corresponding to a ring of horizontal and vertical frequencies around the center.

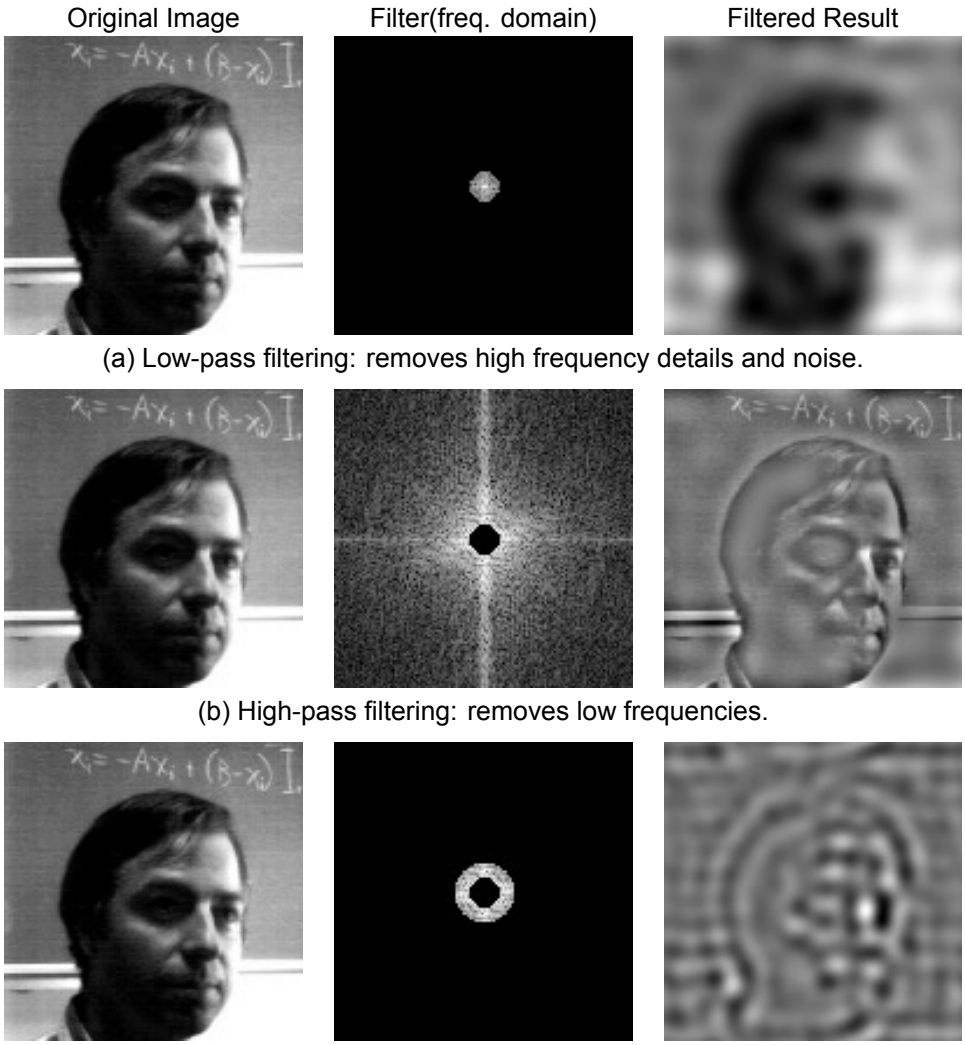
3.3. Multi-Scale Image Representation and the Scale-Space

Images are usually described by local features and texture details: points, lines, edges, etc i.e. the elemental features that make up all kinds of things that can be present in images. But due to the nature of the 3D world we live in and the way that cameras work, these local features are of different scales. Features further away from the camera appear smaller than closer ones. In figure 3.7, we see details of the sunflowers which are the same size in real life and have the same general features but are of different sizes in the image. To be able to find these objects, we would need to represent their features agnostic of their size.

There is not always prior knowledge about the scale of these features and therefore we need to be able to process them irrespective of scale. A system that takes visual input of the physical 3D world should be able to be size/scale-invariant. Additionally, the local details of an object which are prominent at nearby distances appear smaller and smaller as distance increases until they disappear.

3.3.1. Scale-Space Theory

Koenderink [11] mathematically defined scale and structure of images showing how image structure can be seen at different scales simultaneously. It describes one of the most basic way to represent



(a) Low-pass filtering: removes high frequency details and noise.

(b) High-pass filtering: removes low frequencies.

(c) Band-pass filtering: retains only certain spatial frequencies.

Figure 3.6: Shows the comparison between results of low, high and band-pass filtering and the corresponding filters in the frequency domain. Adapted from [15]

image at different scales or resolutions simultaneously which is to embed the image into a one parameter family of images $f(x, \sigma)$ as follows:

$$f(x, y, \sigma) = (f_0 * G(\cdot; \sigma))(x, y) \tag{3.10}$$

where f_0 is the original image at 'zero scale", $G(\cdot; \sigma)$ is the Gaussian of a particular standard deviation σ:

$$G(x, y; \sigma) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{3.11}$$

The function f is a single parameter family of images that encodes the scale, i.e. for all positive scales $\sigma > 0$, it gives an image that is smoother than the original. At $\sigma \to 0$, $(f_0 * G(\cdot; \sigma))(x, y) \to f_0$ and at $s \to \infty$, $(f_0 * G(\cdot; \sigma))(x, y) \to constant$ which tends to the average value of the original image. This function f is known as the scale-space. An example of a scale-space representation for an image is shown in figure 3.8. In the original image, we can see the details of the building and the trees but

Figure 3.7: Image of a field of sunflowers illustrating varying scale of objects and their features in images. The flowers closer to the camera appear larger with more prominent features like the separation between petals, carpels(brown) and stamen(green) while the ones further away, less so.

when increasing σ and hence scale, these become hard to see. Correspondingly, the shape of the objects is more apparent.

3.3.2. Gaussian Convolution Operator

At first, it may seem as if there are other better ways to build the scale-space with a different kernel other than the Gaussian, but setting some simple basic requirements means that there is only 1 scale-space construction possible, i.e. with the Gaussian scale-space operator. Koenderink [11] constructed the scale-space by adding certain constraints and defined the scale-space operator as follows. A family of image operators L^σ where s refers to the scale such that given the 'image at zero scale' f_0 we can construct the family of images $L^\sigma * f_0$ such that:

- L^σ is a linear operator
- L^σ is translation invariant, which means that the operator behaves in the same way at every point of the image
- L^σ is rotation invariant, which means that all orientations or directions are treated equally
- L^σ is separable by dimension
- L^σ is scale-invariant, which means that we should be able to create an equivalent scale-space with a scaled version of the image by possibly re-parameterizing the scale.

In addition to this, causality in scale space must be satisfied. This prevents the use of linear operators that can introduce spurious detail at coarse scales that are due to structures at finer scales. It can be shown that the scale-space operator satisfying these requirements can only be a convolution with a Gaussian. But we do not prove it in this text. Refer to Koenderink [11] for more details. Next, we discuss some important properties of Gaussian convolution, which are important in the context of scale-space.

Scale Space Diffusion Equation

The scale space function is the solution to the diffusion equation:

Figure 3.8: Scale-Space representation of an image from the original at σ=0 (bottom-most) to $\sigma = 8$(topmost).

$$f_\sigma = \sigma\nabla^2 f = \sigma f_{xx} + \sigma f_{yy} \tag{3.12}$$

with the initial condition $f(\cdot, 0) = f^0$. This was the starting point of the derivation of linear scale space. It gives the relationship between spatial derivatives and convolution with Gaussian in scale-space. It states that the rate of change of f with respect to an infinitesimal change in scale is proportional to the differential structure of f at scale σ.

Scale Space Derivatives

Since the spatial derivative is translation invariant which commutes with Gaussian Convolution operation, we can observe the following property:

$$\partial f^0(x) * G(x;\sigma) = \partial(f^0(x) * G(x;\sigma)) = f^0(x) * \partial G(x;\sigma) \tag{3.13}$$

we can either construct the scale-space from the differentiated image or take the derivative at each level of the scale-space. Additionally, we can also compute the Gaussian derivative at all scales. The derivative of a Gaussian is available in closed-form, and is much easier to formulate when compared to the derivative of an image. Therefore, this property allows mathematical convenience.

Scale-Normalized Gaussian Derivatives From figure 3.9, it is evident that the derivatives decrease in amplitude with increasing orders. When comparing derivatives across scale this is problematic. To solve this problem, we will use scale normalized derivatives. If $G^n(x;\sigma)$ denotes the n^{th} order spatial derivative of a Gaussian with standard deviation of σ, the scale-normalized Gaussian derivative is given as:

$$G^n_{norm} = \sigma^n G^n \tag{3.14}$$

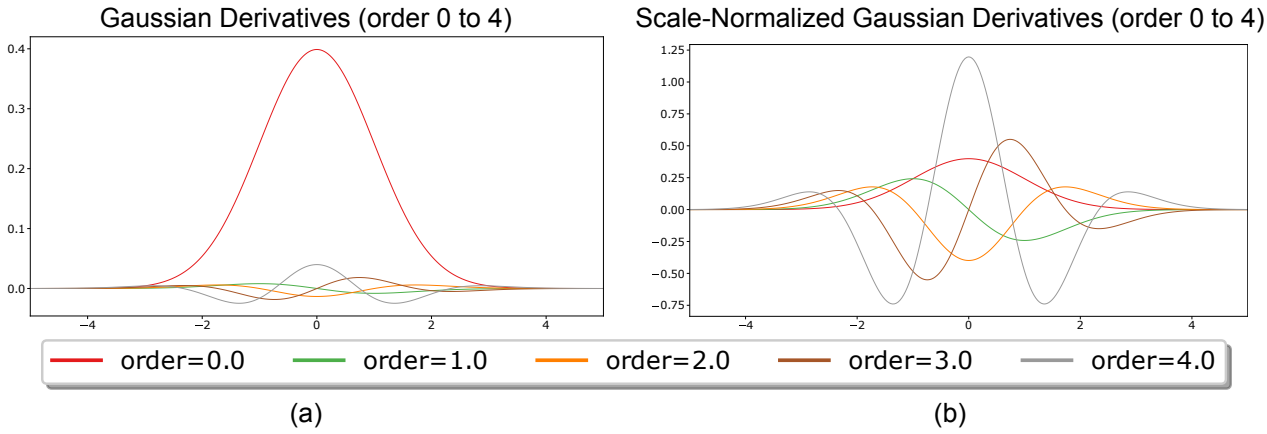


Figure 3.9: Gaussian derivatives decrease in amplitude with increasing orders (a) and Scale-Normalized Gaussian Derivatives (b) prevents this reduction in amplitude.

Convolution with Gaussian Derivative at different scales As discussed earlier, changing the scale parameter causes a scale-space image to become smoother and allows the representation of coarser features. This can be directly represented by the gradient vector at different points in the scale-space images. We illustrate this in figure 3.10 which shows image derivatives with Gaussians of different scales. Figure 3.10 (a) shows the original, while 3.10 (b) and (c) shows the image derivatives at scale 1 and 4 respectively. 3.10 (d), (e) and (f) shows the direction of the gradient at same point in space for scales 0, 1 and 4 respectively. The direction of the gradient vector corresponds to the level of detail at that scale.

3.4. Machine Learning and Neural Networks

This text assumes that the reader has some basic understanding of neural networks. In this section, we give a brief recap and introduce the notions that are needed to later introduce the math behind the convolutional neural networks.

3.4.1. Supervised Machine Learning

The typical formulation of a machine learning system is the transformation of inputs x into desired outputs y with a fixed function called the hypothesis h_p which is parameterized by some parameters p. The system must optimize/learn these parameters:

$$\hat{y} = h_p(x) \tag{3.15}$$

Let us consider a system for the classification of images of handwritten digits as available in the MNIST dataset. Here, we consider the input x is a vector containing the pixel values of the images. The dataset can be represented as a set of tuples $(x^{(i)}, y^{(i)})$, where the vector $x^{(i)}$ is an instance of the input and $y^{(i)}$ is the label or the ground-truth value of the corresponding input. The goal of the classification task is to classify each of the input images, $x^{(i)}$ correctly into one of 10 classes $0, 1, 2, \cdots, 9$. This classification is the index of the element with the largest value in the output vector $\hat{y}$. The system aims at finding the correct values for the parameter vector p that will minimize the total loss (error) made for all examples in the data set. This paradigm of machine learning is called supervised learning.

$$L = \sum_{i=1}^{m} l(h_p(x^{(i)}, y^{(i)}) \tag{3.16}$$

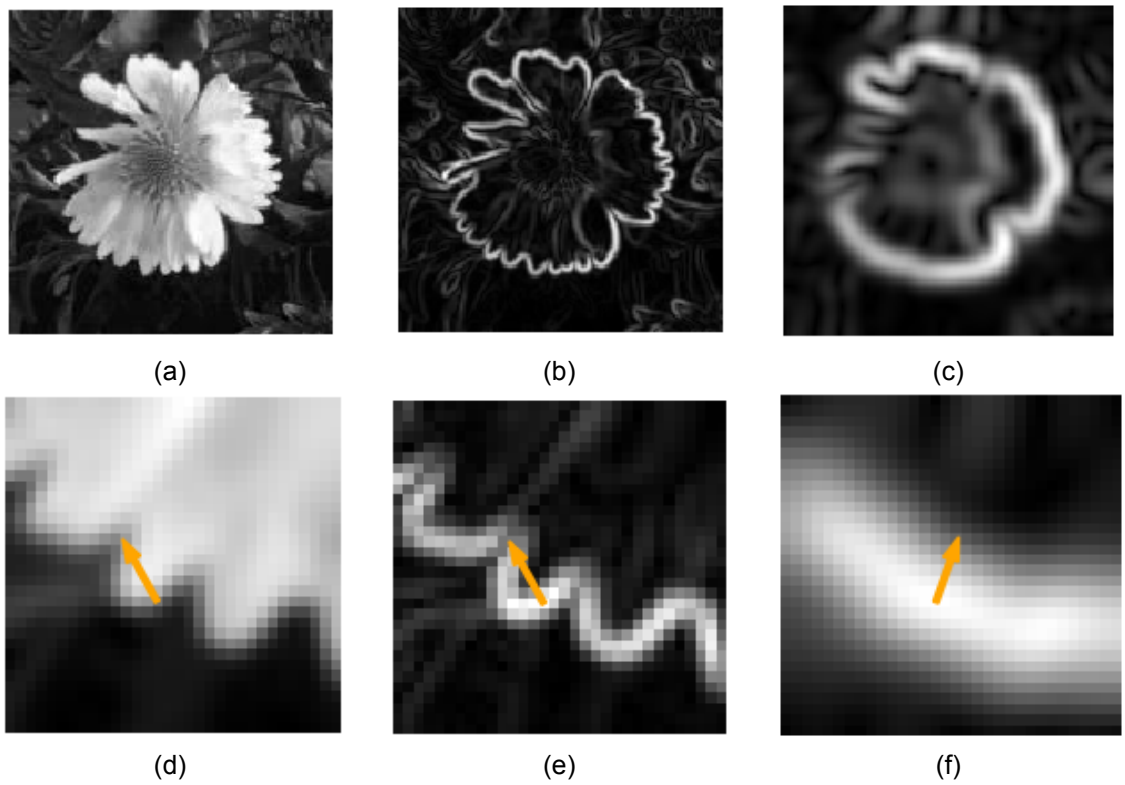


Figure 3.10: Image of a flower (a) convolved with a first-order Gaussian of scale = 1.0 (b) and scale = 4.0 (c). Images in the second row (d, e, and f) show a zoomed-in image of the corresponding image above along with a super-imposed gradient vector. Note the change in direction of the gradient vector for a larger sigma in the direction of the coarser feature of the flower. Adapted from [1]

where l is the loss contribution for a single example-target pair. There are multiple formulations of loss functions as desired for the task to be performed. We can state the objective of machine learning in this context as:

$$p^* = \arg\min_p L(p)$$
$$= \arg\min_p \sum_{i=1}^{m} l(h_p(x^{(i)}), y^{(i)})$$

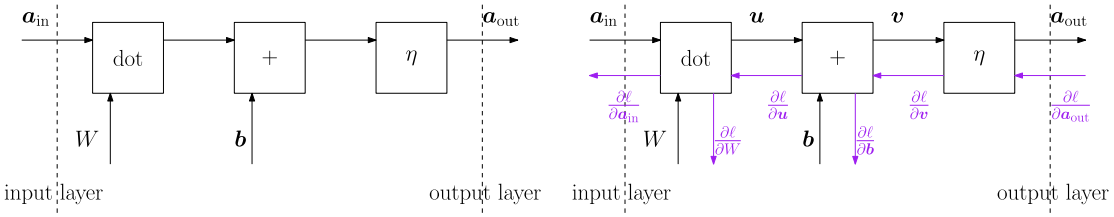
Most machine learning systems employ numerical optimization techniques to find the optimal parameters p^* because the loss functions are quite complex and difficult to solve in a closed form. One of the most common techniques used now is the iterative gradient descent which updates the values of the parameters in the direction of the gradient of the loss function with respect to the parameters:

$$p := p - \alpha \frac{\partial L(p)}{\partial p} \tag{3.17}$$

3.4.2. Fully-Connected Neural Network

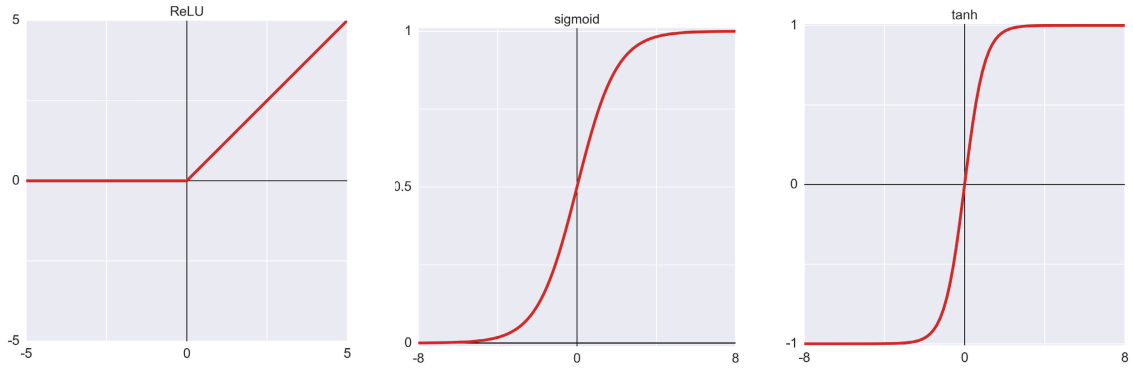
A fully connected neural network is a sequence of processing modules that take an input vector a_{in} and produces an output vector a_{out} as a linear combination of the inputs:

$$a_{out} = \alpha(W a_{in} + b) \tag{3.18}$$



(a) Forward Propagation (b) Backward Propagation

Figure 3.11: Fully-Connected Neural Network Module during forward propagation (a) and the same network with the backward propagation depicted in purple. Taken from [1].



(a) (b) (c)

Figure 3.12: Common activation functions used in Neural Networks. ReLu (a), Sigmoid (b) and Tanh (c). Adapted from [8].

The weights matrix W and bias vector b are the parameters of each module and α is known as the activation function. Sigmoid and the ReLU functions are well known activation functions. Some common activation functions are illustrated in figure 3.12. A Fully Connected Neural Network is a sequence of these modules. A single module is illustrated in figure 3.11 (a). The derivatives of the loss function with respect to each of the parameters in the chain of modules is computed using the chain-rule of differentiation. The chain-rule is computed from the end of the last layer known as backpropagation or backprop [20, 19]. This is shown for one layer in figure 3.11 (b) in purple. This is shown in more detail below:

$$
\begin{aligned}
\frac{\partial l}{\partial v} &= \alpha'(v) \odot \frac{\partial l}{\partial \hat{a}_{out}} \\
\frac{\partial l}{\partial u} &= \frac{\partial l}{\partial v} \\
\frac{\partial l}{\partial a_{in}} &= W^{\top} \frac{\partial l}{\partial u} = W^{\top} \left(\alpha'(v) \odot \frac{\partial l}{\partial \hat{a}_{out}} \right) \\
\frac{\partial l}{\partial b} &= \frac{\partial l}{\partial v} = \alpha(v) \odot \frac{\partial l}{\partial a_{out}} \\
\frac{\partial l}{\partial W} &= \frac{\partial l}{\partial u} x^{\top} = \left(\alpha(v) \odot \frac{\partial l}{\partial a_{out}} \right) x^{\top}
\end{aligned}
$$

We have discussed what a neural network does, at least for 1 block. If we connect multiple blocks together with the loss function, we can use the chain rule of differentiation to compute the gradients of loss functions. Once we have the gradients of the loss function, we do the same steps as discussed in the previous section to optimize parameters using iterative gradient descent.

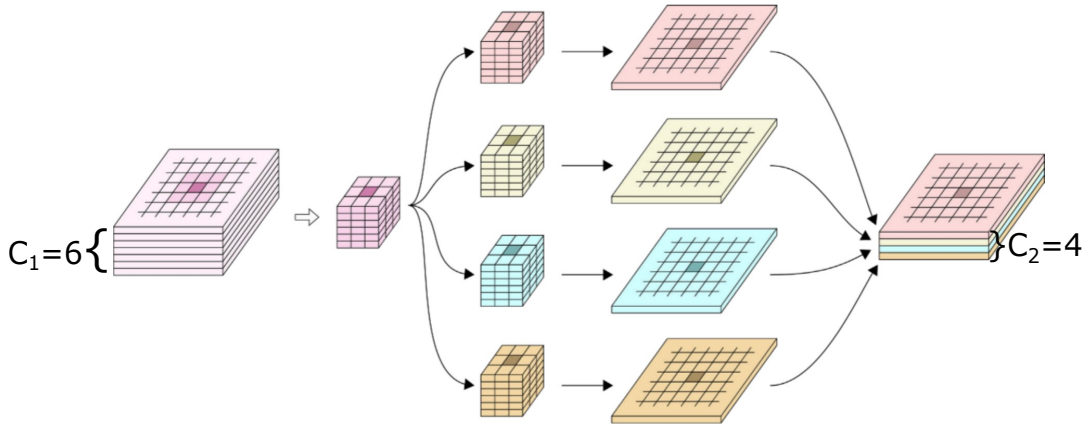


Figure 3.13: 2D convolution with multiple input and output channels. Each convolutional kernel takes the C_1 input channels as input to produce one of the values — after convolution and non-linear activation — into one of the C_2 output channels. This is performed in a windowed fashion, keeping each pixel of the input at the center of the window. Here, we have $C_1 = 6$, $C_2 = 4$ and convolution window $S = 3$ (same as size of the kernel).

3.4.3. Convolutional Neural Networks

The basic principle behind the working of a CNN [14] is extracting local image structure at different scales, which is the same as what we discussed in section 3.3 on Scale-Space and 3.2.1 on Spatial Domain Filtering. This is not surprising as CNNs are made up of layers of convolutions in a range of receptive field sizes. Since we need only a local structure, a fully connected block would have superfluous parameters which would lead to overfitting and generalize poorly to new data. Hence, the inputs and layers are organized such that each processing module of a CNN considers only a small neighborhood as input.

Instead of connecting every module of a layer to every module of another layer, CNN layers are organized into multiple *channels* which is also known as the depth of the image. In the Convolutional layer, a linear sum is performed in a local window, similar to shift-invariant convolution from equation 3.2. Though, there is a key difference between the convolution from image processing and convolution in CNNs. In image convolution, the filter applied to each of the color channels of the images is the same. But in CNNs, a different kernel is used for each of the channels and the activated output from convolution is linearly combined to determine the output for the next layer, the weights for which are learned via backpropagation as seen earlier. This is illustrated in figure 3.13. This is ideal as it allows the creation of local features which can be combined to learn meaningful features as required for the downstream task.

We can mathematically formulate the operation performed on each pixel of the input in a convolutional layer as:

$$s(i,j,c_2) = \sum_{c_1 \in C_1} \sum_{k,l \in \mathcal{N}} w(k,l,c_1,c_2) x(i-k, j-l, c_1) + b(c_2) \tag{3.19}$$

where i and j are the current pixel, $x(i,j,c_1)$ is the input, $\mathcal{N}$ is the local neighbourhood of pixels, c_1 is the input channel and c_2 is the output channel. Since the weights in a filter kernel are shared across the pixels within a layer and channel, they learn a representation that connects these pixels. Additionally, they learn fewer weights than fully connected layers. There are some additional terms we must define to fully determine the convolutional layer. We briefly go over them.

Padding: We cannot perform convolution at the outermost pixels of input since the kernel would then be placed outside the image. Hence, the size of the output keeps shrinking as we do more convolutions. Modern networks extend the image on all sides (padding) to prevent this.

Stride: Until now, we have assumed that the convolution is evaluated at every pixel of the input. It is also possible to evaluate it at every n^{th} row and n^{th} column. The number of pixels skipped in between is known as the stride of the convolutional layer.

Grouping: In the example we have seen, all input channels are used to create each of the output channels. It is also possible to group the channels into independent groups where the input channels in a group are convolved separately from channels in other groups. This is known as depthwise convolution.

Dilation: This is the concept of non-zero stride applied in the output. Rows and columns are skipped in the output channels which allows effective pooling with fewer parameters and over a larger region of the input.

This section only provides a brief outline of CNNs and convolutional arithmetic. Other better articles exist that explain these concepts in greater depth [6]

3.5. CNNs with Fixed Filter Banks

The power of CNNs comes from the ability to learn rich local feature representations and combining them optimally as required for the downstream learning task. When this is satisfied, they can solve problems that are extremely complex for humans [13, 24]. When data is limited, this is quite difficult due to a large number of parameters. One solution is to design priors that are carefully tuned so that learning is much easier. In this section, we look at 2 related ideas, the Scattering Transform [4, 17, 21]: a fully engineered representation which uses a wavelet filter bank with fixed filter parameters, and Structured Receptive Fields(SRF) [10]: representation composed of filters structured as a combination of fixed orders of Gaussians whose weights are learned. The SRF is the work on which we build our method.

3.5.1. Scattering Transform

The Scattering Transform is a CNN variant that uses filters composed of a fixed set of wavelets. It does not have any learnable filter parameters, and the choice of filters must be tailored to the kinds of images present in the dataset for the task. The scattering transform computes a local translation-invariant representation of the input which can recover wavelet coefficients in succeeding layers to avoid loss of information. Since the wavelet filters are fixed, the extracted features are fed into a Support Vector Machine classifier [2] to perform the downstream learning task. This method works especially well in the small data regime since there is no feature learning, and the effectiveness of the filters does not depend on the number of data points

To design the scattering transform, one must have a good idea of the translational, rotational, and scale-related variability present in the target domain. Scattering Transforms designed with the knowledge of this variability results in highly effective filters. This kind of hard-coding of the invariances and subsequently the filters can be effective when the domain is known precisely, which is rare for a majority of applications. In the case of an unknown domain, it may be required to exhaustively compute scattering paths. This leads to a high-dimensional parameter space which is difficult to learn.

As we will see in the next section, the Structured Receptive Field Network, unlike the Scattering Transform, directly learns effective filter combinations allowing a compact representation. This avoids the issues of hard-coding discussed.

3.5.2. Structured Receptive Field (SRF) Network

SRF Networks are a class of CNNs where the Receptive Field is formulated as a weighted sum of a fixed basis, similar to the Convolutional Scattering Network [3]. The key motivation of the authors

was to create a model which can generalize well in the case of limited training data. Solutions such as pre-training and transfer learning provide an understanding of the characteristics or features of images. If these properties were to be built into the network then it should allow for better generalizeability.

SRF Networks formulate the convolutional layer as a linear combination of Gaussian derivatives known as a Structured Receptive Field. The Gaussian family of basis functions is chosen for their ability to represent local image features [12]. Additionally, Scale-Space theory guarantees that an image function can be expanded as a Taylor function with a Gaussian kernel. These two together mean that the SRF is capable of forming any filter that is needed for learning effective image representations.

In SRF Networks, images are represented as functions in scale-space unlike in CNNs which treat images as pixel values. Representing the image in scale-space gives us two advantages. First, that we can change the resolution to operate on a continuous linear scale. Secondly, we can use differential operators on image functions.

To approximate an arbitrary CNN filter F as a sum of N Gaussian derivatives, we first approximate it as a Taylor Expansion centered at a.

$$F(x) = \sum_{i=0}^{N} \frac{F^i(a)}{i!}(x-a)^i \tag{3.20}$$

We can compute the derivative of an image I by convolving with a Gaussian kernel $G(\sigma)$

$$G(\sigma) * F(x) = \sum_{i=0}^{N} \frac{(G(\sigma) * F)(a)}{i!}(x-a)^i \tag{3.21}$$

This sum of weighted Gaussian kernels of different orders convolved with the image function is a good functional approximation, to the standard convolutional filtering. In theory, a CNN filter is exactly equal to this formulation when you use a basis set of orders up to infinity, but it has been shown that lower orders are sufficient to capture details needed for most tasks, depending on the resolution of the images [11, 16].

As an aside, a 1D Gaussian can be created using the Hermite formulation. It allows for the computation of Gaussian basis function of arbitrary order m:

$$G^m(\cdot;\sigma) = (-1)^m \frac{1}{\sigma^{m/2}} H_m\left(\frac{x}{\sigma\sqrt{(2)}}\right) \circ G(x;\sigma) \tag{3.22}$$

where H_m is the mth Hermite polynomial.

A single SRF block is illustrated in figure 3.14. Each filter F is a linear combination of gaussian derivatives ϕ_m, each weighted by α_{ij} where i and j index the input and output feature maps:

$$F(x,y) = \alpha_1\phi_1(x,y) + \alpha_2\phi_2(x,y) + \cdots + \alpha_n\phi_n(x,y) \tag{3.23}$$

The α_{ij} model the coefficients of the Taylor terms and the ϕ_i represent the Gaussian kernel. The filters formed from the SRF block are then convolved with the input image to give a feature map.

An SRF-CNN optimizes the filters as required for the downstream learning tasks via gradient descent backpropagation, not unlike traditional CNN. It is this similarity that makes the SRF block easy to in-

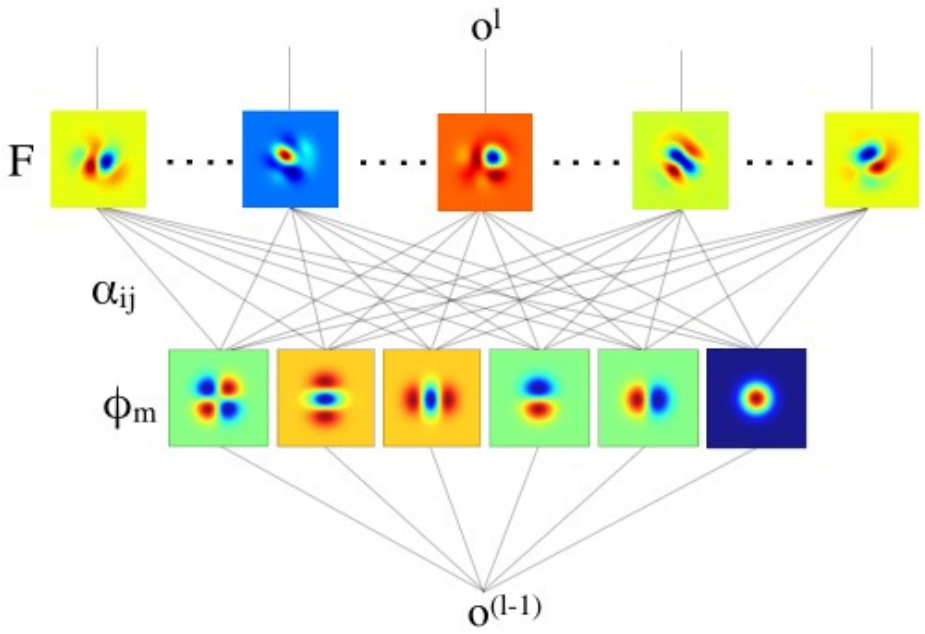


Figure 3.14: Illustration of the Structured Receptive Field block taken from Jacobsen et al. [10]. The kernels of filter F are created as a linear combination of basis ϕ_m weighted by learned weights α_{ij}.

tegrate into any standard Convolutional Network. As mentioned earlier, the orders of the Gaussian set are fixed, and the network only optimizes the weights α_i of the Gaussian basis functions.

Bibliography

[1] *7.1. Linear Scale Space — Image Processing and Computer Vision 2.0 documentation*. URL: https://staff.fnwi.uva.nl/r.vandenboomgaard/ComputerVision/LectureNotes/IP/ScaleSpace/linear_scale_space.html (visited on 06/27/2021).

[2] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[3] Joan Bruna and S. Mallat. "Invariant Scattering Convolution Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1872–1886. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.230. URL: http://ieeexplore.ieee.org/document/6522407/ (visited on 10/31/2020).

[4] Joan Bruna and Stephane Mallat. "Invariant Scattering Convolution Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1872–1886. DOI: 10.1109/TPAMI.2012.230.

[5] Peter J. Burt and Edward H. Adelson. "The Laplacian Pyramid as a Compact Image Code". In: *IEEE TRANSACTIONS ON COMMUNICATIONS* 31 (1983), pp. 532–540.

[6] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[7] Pascal Getreuer. "A Survey of Gaussian Convolution Algorithms". In: *Image Processing On Line* 3 (2013). https://doi.org/10.5201/ipol.2013.87, pp. 286–310.

[8] Andrew Glassner. *Deep Learning, Vol. 2: From Basics to Practice*. Kindle Edition. www.glassner.com, 2018. URL: http://gen.lib.rus.ec/book/index.php?md5=19ffded62d24612451194b8564186c64.

[9] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc., 2006. ISBN: 013168728X.

[10] Jorn-Henrik Jacobsen et al. "Structured Receptive Fields in CNNs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[11] Jan J Koenderink. "The structure of images". In: *Biological cybernetics* 50.5 (1984), pp. 363–370.

[12] JJ Koenderink and AJ van Doorn. "Representation of local geometry in the visual system". In: *Biological cybernetics* 55.6 (1987), pp. 367–375. ISSN: 0340-1200. DOI: 10.1007/bf00318371. URL: https://doi.org/10.1007/bf00318371.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[14] Y. Le Cun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Proceedings of the 2nd International Conference on Neural Information Processing Systems*. NIPS'89. Cambridge, MA, USA: MIT Press, 1989, pp. 396–404.

[15] Steven Lehar. *Frequency Domain Filtering*. URL: http://www.cs.cmu.edu/~16385/s15/lectures/Lecture3.pdf.

[16] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Boston, MA: Springer US, 1994. ISBN: 978-1-4419-5139-7 978-1-4757-6465-9. DOI: 10.1007/978-1-4757-6465-9. URL: http://link.springer.com/10.1007/978-1-4757-6465-9 (visited on 11/06/2020).

[17] Stéphane Mallat. "Group Invariant Scattering". In: *Communications on Pure and Applied Mathematics* 65.10 (2012), pp. 1331–1398. DOI: https://doi.org/10.1002/cpa.21413. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.21413. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.21413.

[18] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing (2nd Ed.)* USA: Prentice-Hall, Inc., 1999. ISBN: 0137549202.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. ISBN: 026268053X.

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1, 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[21] Laurent Sifre and Stephane Mallat. "Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013.

[22] Irwin Sobel. "An Isotropic 3x3 Image Gradient Operator". In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[23] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345.

[24] Yaniv Taigman et al. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708. DOI: 10.1109/CVPR.2014.220.