Scale Learning in Scale-Equivariant Convolutional networks

by

M. J. Basting

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday October 11, 2024 at 10:00 AM.

Student number:	4904923	
Project duration:	November 29, 2022 – October 11, 2023	
Thesis committee:	R. Bruintjes,	TU Delft, daily supervisor
	Dr. J. C. van Gemert,	TU Delft, supervisor
	Dr. K. A. Hildebrandt	TU Delft, external Committee member

This thesis is confidential and cannot be made public until December 31, 2024.

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

TU Delft

Preface

This work represents the final work of my Master in Computer Science at Delft University of Technology. First, I would like to thank my daily supervisor Robert-Jan Bruintjes for the useful insights, motivation and great discussions we had. Additionally, I would like to thank Jan van Gemert, my thesis supervisor, for the valuable feedback. Finally, I would also like to thank my fellow student, Luuk Haarmarn, for our productive discussions and peer feedback.

M. J. Basting
Delft, October 2023

Contents

1 Introduction

The thesis is structured as follows: first, a general overview of the storyline will be given in Chapter 1. The main body of the thesis is presented in Chapter 2 in CVPR style article form. In Chapter 3, a short overview of Deep Learning methods can be found together with explanations of Convolutional Neural Networks, translation equivariance and scale equivariance.

1.1. General Storyline

Scale is an important factor to consider in digital image analysis. Objects appear at various distances from the camera resulting in a variable number of pixels being used to represent the same object in an image. Additionally, there is the notion of inter-class variation of scale which refers to the difference in scale between objects of the same class. An example of this is humans themselves; children are much smaller than adults, and recognising them based on features like their arms should consider the variation in size. Ideally, we want to be able to recognise objects in an image regardless of their scale or distance from the camera.

Convolutional Neural Networks (CNNs) have been widely popular in the field of computer vision but do not consider scale by default. With enough training examples, a CNN can recognise and classify differently-sized objects in an image. However, CNNs learn to classify differently-sized objects by learning different features corresponding to the same objects of a different scale. This leads to significant redundancy in the learned features.

Scale convolution methods, such as SESN [3], extend regular convolution in the scale domain to make them scale-equivariant. Scale-equivariance means that scaling the input scales the output of the operation that is scale-equivariant in the same way. Instead of learning differently sized versions of the same feature, scale convolution methods aim to share the features across a set of chosen scales S. This leads to an increase in parameter efficiency.

Scale convolution methods require the user to choose the set of scales S to share the features over. The set S is of limited size since it adds significant computational complexity for each additional scale. Thus, it is important to configure the set of scales S wisely. However, the current work does not explain in detail how to choose the set of scales S for different scale distributions.

In this work, we develop an empirical model which links the set of scales S to the data scale distribution and suggests how to choose the internal scales accordingly. The intuition behind the empirical model is that each filter at a certain scale s_i is activated for several data scales. We refer to this region as the tolerance of the scale $s_i \in S$. We combine the tolerances for all scales S into one mixture model and use optimisation to fit the mixture model to the known data scale distribution. The response of the optimisation is a suggestion on how to choose the set of scale S, narrowing down the search space and reducing the time needed for optional hyperparameter optimisation.

However, the empirical model cannot suggest how to choose the set of scales when the data scale distribution is unknown. We extend the work done by SESN [3] and parameterise the set of scales S by a learnable parameter. We adopt a similar approach to N-JetNet [2], directly learning the scales from the data using gradient optimisation. This allows the network to optimise the set of scales S instead of having to choose it before training.

2

Scientific Article

The main body of the thesis is presented in this chapter. The article is written in CVPR style form and thus starts on the next page.

Scale Learning in Scale-Equivariant Convolutional networks

Mark Basting
Delft University of Technology
Delft, 2628 CD, Netherlands
m.j.basting@student.tudelft.nl

Abstract

In real-life scenarios, there are many variations in sizes of objects of the same category and the objects are not always placed at a fixed distance from the camera. This results in objects taking up an arbitrary size of pixels in the image. Vanilla CNNs are by design only translation equivariant and thus have to learn separate filters for scaled variants of the same objects. Recently, scale-equivariant approaches have been developed that share features across a set of pre-determined fixed scales. We further refer to this set of scales as the internal scales. Existing work gives little information about how to best choose the internal scales when the underlying distribution of sizes, or scale distribution, in the dataset, is known. In this work, we develop a model of how the features at different internal scales are used for samples containing differently-sized objects. The proposed model return comparable internal scales to the best-performing internal scales for different data scale distribution of various width. However, in most cases, the scale distribution is not known. Compared to previous scale-equivariant methods, we do not treat the internal scales as a fixed set but directly optimise them with regard to the loss, removing the need for prior knowledge about the data scale distribution. We parameterise the internal scales by the smallest scale which we refer to as σ_{basis}, and the Internal Scale Range (ISR) that models the ratio between the smallest and largest scale. By varying the ISR, we learn the range of the scales the model is equivariant to. We show that our method can learn the internal scales on various data scale distributions and can better adapt the internal scales than other parameterisations. Finally, we compare our scale learning approach and other parameterisations to current State-of-the-art scale-equivariant approaches on the MNIST-Scale dataset.

1. Introduction

Scale is an important factor to take into account in image processing. The scale, or size in terms of pixels, of an object in an image can vary because of the distance to the camera or due to interclass variation. For example, imagine a golf ball and a volleyball being classified as balls but varying in size. Vanilla CNNs can learn differently-sized objects when presented with large amounts of data. However, since the CNN has no internal notion of scale, separate filters for differently scaled versions of the same objects are learned, leading to significant redundancy in the learned features.

Convolutional layers are by design translation equivariant. Translating an object in the image results in a similar transform applied to the output when convolved. Invariance is added in the form of a pooling operator and is used for tasks such as classification. Scale-equivariance and scale-invariance work the same way but for scaling transformations. The main benefit of equivariance to different transformations is computational efficiency since features can be reused across different transformations.

Scale-equivariant and scale-invariant Networks share features across a fixed set of chosen scales. We will refer to this set as the internal scales. Sharing the features across multiple scales increases parameter efficiency by removing the need to learn separate filters for differently-sized objects. DSS [17] make use of a Gaussian Scale-Space and filter dilation to evaluate the filter at different scale levels. Similarly, [2, 5, 11] use differently sized versions of the input to share features across different integer and non-integer scales. Alternatively, scale can be accounted for by using scale group-convolutions [4, 9, 12, 14, 15, 18, 21]. The general idea behind these scale convolution approaches is to share a kernel across chosen scales and add a scale dimension to regular convolutions. However, current scale-convolution approaches do not explicitly explain how to choose the internal scales. We present a model that demonstrates how to choose the internal scales when the underlying distribution of sizes, or scale distribution, in the dataset, is known.

In most cases, the data scale distribution is not known and configuring the internal scales becomes an even harder task. Recently, several methods to learn the scales in the dataset have been proposed. One approach learns the size

of the underlying continuous function that parameterises the kernel directly through gradient optimization [13, 20]. Augerino [1] estimates scale and other symmetries through learning the underlying distribution. SREN [16] instead approximates the local scale of the input image and transforms the kernel accordingly. These methods either require choosing constraints for each internal scale to prevent them from converging to the same value [20], are only limited to a single scale [13], require extensive use of transformation operations [1] or only cover small scale ranges [16]. We present a model capable of learning multiple scales simultaneously without defining any constraints. Our method is not reliant on many transformations of the input image and can cover large scale-ranges.

In this paper, we present a model of the relationship between the internal scales and the data scale distribution. We show empirically the parameters for which this model is most accurate. Furthermore, we define a parameterization of the internal scales and draw inspiration from NJet-Net [13] to learn the internal scales. Our method provides a way to learn the internal scales without the need for prior knowledge of the scale distribution of your data.

Our method can adjust the internal scales and achieve competitive performance. Especially in wide data scale distributions our Internal Scale Range based approach leads to more stable internal scales and achieves better performance than other parameterisations. Furthermore, our scale-learning-based approach performance is on par with its non-scale-learning counterpart SESN [14] on MNIST-Scale [5].

In short, our **contributions** are:

- We demonstrate that the best internal scales are related to the used data scale distribution.
- We derive an empirical model that shows approximately how we should choose the internal scales when the data scale distribution is known.
- We remove the need for prior knowledge about the data scale distribution by making the internal scales learnable.

2. Related Work

2.1. Scale spaces

Unlike translation, which uses a linear axis, scale is naturally defined on a logarithmic axis [3]. [10] use this property and the notion of scale-spaces to define the scale-space primal sketch. This method forms the basis of many future work by providing an explicit description of the relation between structures at different scales for continuous and discrete signals. More recently, DSS [17] use a scale space in combination with semi-group theory to create a scale-equivariant model. While scale-space approaches achieve a very low equivariance error, they are limited to integer scale factors which drastically limits the applicability when more fine-grained control is necessary. We do use theory on the logarithmic nature of scale to define the internal scale tolerance model and in the parameterisation of the internal scales.

2.2. Pyramid Networks

Pyramid-based networks are closely related to scale space approaches since they also use differently scaled versions of the input image to share features across different scales. These approaches differ in the sense that they are not limited to only integer scale factors. Hierarchical ConvNets [2] used Laplacian pyramids to evaluate the filters at each scale disjointly. At the end of the network, the responses for each scale are aligned and concatenated. SI-ConvNet [5] instead uses a pooling operation to combine the scale responses after each convolution. Hierarchical ConvNets and SI-ConvNet are both scale-invariant, SEVF [11] argues that this is not always wanted behaviour, for example when the size of the object encodes important information. SEVF achieves scale-equivariance by injecting the maximum activated scale after each pooling layer into the next layer. Pyramid-based networks are equivariant over fixed chosen scales and require many expensive interpolation operations. Contrarily, our approach can learn the scales without extensive use of interpolations.

2.3. Scale-Convolution approaches

An alternative way to achieve scale-equivariance or scale-invariance is through the use of group convolution. One of the earliest networks to achieve local scale-invariance is SiCNN [18]. SiCNN uses interpolation to share filters across different scales. Other methods [4, 9, 12, 14, 21] have proposed to, instead of expensive interpolation, use a parameterised kernel. The kernel of these methods consists of a weighted sum of fixed multi-scale continuous basis functions. The resulting continuous kernel is discretized before the convolution operation. DISCO [15] argues that the discretisation of the underlying continuous basis functions leads to increased scale-equivariance error and therefore leads to worse performance. Instead, they opt to use dilation for integer scale factors and directly optimise basis functions for non-integer scales using the scale-equivariance error [15]. While all methods allow for non-integer scale factors, the scales over which the network is equivariant are fixed and they provide little instructions on how to best choose the internal scales.

2.4. Learnable Scale

Continuous kernel parameterisation forms the basis of methods that aim to learn the scale or scales of the dataset. NJet-Net [13] learned the scale of the dataset by making the σ parameter of the Gaussian derivative basis function

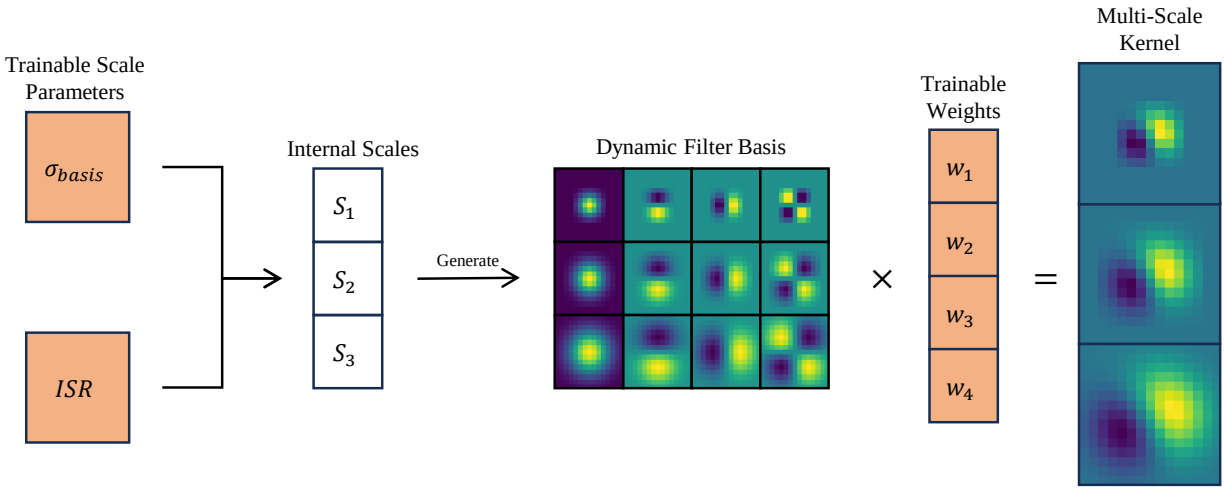

Figure 1. Dynamic Multi-Scale Kernel generation pipeline. Filter basis is parameterised by a discrete set of scales which in turn are generated from learnable parameters, controlling both the size of the first scale σ_{basis} and the range the internal scales span (ISR). Linear combination of the Dynamic Filter Basis functions with trainable weights form Multi-Scale Kernel.

learnable. While effective, this method is limited to learning a single scale. SEUNET [20] builds upon this work by learning the multiple scales that parameterise the underlying multi-scale Gaussian basis functions. However, it requires choosing disjoint intervals for each internal scale to lie between otherwise the σ values collapse. Augerino [1] takes a more approach and aims to discover symmetries present in the dataset through estimating the underlying distribution. At test time, multiple augmentations are then sampled from the learned distribution which are used as input to the model. While Augerino achieves invariance and equivariance it requires many transformations of the input image which is an expensive operation. A different approach is taken by SREN [16] which instead approximates the local scale, and rotation, and adapts the continuous kernel accordingly in a modified convolution operator. Our method can learn multiple internal scales simultaneously without the use of expensive transformations of the input and can cover large scale-ranges.

3. Method

3.1. Preliminaries

Before going into detail about our internal scale tolerance model and scale-learning approach we first discuss equivariance, basis functions and scale-convolutions.

3.1.1 Scale-Translation-Equivariance

A function is equivariant under a transformation L if there exists a transformation L' if for an arbitrary input x it holds that:

$$L(f(x)) = f(L'(x)) \tag{1}$$

in other words, transforming the output of the function is the same as transforming the input and then applying the function. For an invariant mapping, L' is the identity function.

Our method is equivariant to both translation and scale transformations. Like SESN [14], our method achieves scale-equivariance through an inverse mapping of the kernel:

$$f(L_s[x]) * \kappa = L_s[f(x) * L_{s^{-1}}[\kappa]] \tag{2}$$

where L_s represents a scaling transformation, $f(x)$ represents the value of the input image at position x and κ is the kernel. Thus, a scaled input convolved with a kernel is the same as first convolving the original input with an inversely scaled kernel and then applying the same scaling.

3.1.2 Fixed Filter Basis

Due to the discrete nature of images, we need to approximate the equivariance to translation and scaling by a discrete group. The translation group is approximated by taking into account all discrete translations. The scaling group is discretised by N_S scales with log-uniform spacing as follows:

$$S = [\sigma_{basis} \times ISR^{(\frac{i}{N_S - 1})} \text{ for } i \text{ in } 0..N_S - 1] \tag{3}$$

where σ_{basis} is a learnable parameter that defines the smallest scale, and the ISR defines the range between the largest and smallest scale, also known as the Internal Scale Range.

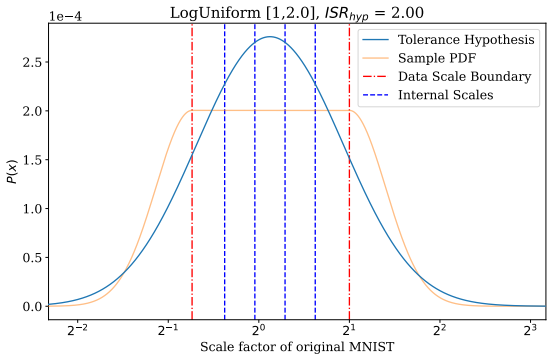

Figure 2. Example of possible internal scale tolerance model for the log-uniform data scale distribution over the range [0.6, 2.0] with ISR_{hyp} fixed to 2 to reflect internal scales choice in SESN [14] on MNIST-Scale.

The logarithmic spacing can be attributed to the logarithmic nature of the scale.

The kernels of the model consist of a weighted sum of basis functions that are defined at each scale in the internal scales S. Following [14], we use a basis of 2D Hermite polynomials with a 2D Gaussian envelope. This basis is pre-computed at the start of training for all pre-determined scales if scale learning is disabled. Otherwise, the basis functions are recomputed at each forward pass.

3.1.3 Scale-Convolution

Scale-convolution is a standard convolution extended by incorporating an additional scale dimension [14]. Without taking into account interscale interactions we define the following two types of scale convolutions:

1. Conv $T \rightarrow H$: In this scenario, the input of the scale-convolution is a tensor without any scale-dimension, or $|S'| = 1$. The output, defined over the internal scales S stems from the convolution of the input with scaled kernels $\kappa_{s^{-1}}$ s.t. $s \in S$:
$$[f *_H \kappa](s,t) = f(\cdot) * \kappa_{s^{-1}}(\cdot) \tag{4}$$
where κ_s is a kernel scaled by s, $*_H$ is the scale convolution and $*$ is a standard convolution.
2. Conv $H \rightarrow H$: The input is now defined over the internal scales S, the resulting output at scale s is the convolution of the input at scale s with the scaled kernel $\kappa_{s^{-1}}$:
$$[f *_H \kappa](s,t) = f(s,\cdot) * \kappa_{s^{-1}}(\cdot) \tag{5}$$

These methods are designed to adhere to the scale-equivariance equation highlighted in Eq. 2.

3.2. Internal Scale Tolerance Model

We define an empirical tolerance model to estimate which internal scales to choose when the scale distribution is known. The tolerance describes the region of data scales the kernel can generalise to. Previous papers have shown that the generalisation error to unseen scales follows an approximate log-normal distribution [5, 9]. Therefore, we use a Normal distribution on a logarithmic scale to model the tolerance for each kernel at a certain internal scale. The log-normal distributions of each internal scale are then combined into one mixture model. An example of a possible configuration can be seen in Fig. 2.

The internal scale tolerance model has the following parameters:

1. Reference Internal Scale: defines the relationship between the position of the internal scales and the data scales.
2. ISR_{hyp}: range over which the internal scales are defined, this is the factor between the largest and the smallest scale.
3. τ_{tol}: standard deviation of the underlying log-normal distribution that is placed on each internal scale.

The reference scale and ISR are specific to each tolerance model of a data scale distribution while τ_{tol} is independent of the data scale distribution and a property of a kernel.

We make the assumption here that we do know the data scale distribution. We extend the data scale distribution at the boundaries by a half-log-normal distribution with $\sigma = 0.4$ to model the generalisation to unseen scales. The Kullback-Leibler [7] is used to fit the tolerance model on the data scale sampling distributions.

3.3. Moving away from fixed Scale Groups

To make the scales of the network learnable we move away from fixed multi-scale basis functions and make the scales of the basis functions dynamic. The scales that parameterise the basis functions are continuous and have a gradient with regard to the loss allowing for direct optimisation. This allows us to simultaneously learn the kernel shape and scales, see Fig. 1. Unlike SEUNET [20], we do not parameterise the scales directly but parameterise the internal scales by σ_{basis} and the ISR using Eq. 6.

We observe that a value for the ISR lower or equal to 1 is unwanted as this would result in kernels at the same scale or a subsequent scale smaller than the base scale defined by σ_{basis}. We do not use a ReLU activation as this can lead to a dead neuron and zero gradient. We parameterise the ISR using the following formula:

$$ISR = 1 + \gamma^2 \tag{6}$$

where γ is the learnable parameter. We will only mention the ISR since it is closely related to the learnable parameter and more intuitive to understand.

Various size basis functions lead to difficulties in choosing the best kernel size before training. We use the method

Conv $T \rightarrow H$, Hermite, $N_S = 3$, 16 filters
scale-projection
batch norm, relu
42 x 42 max pool
fully-connected, softmax

Table 1. Toy Architecture used in Experiments 4.1 and 4.2 to show the impact of choosing the internal scales and scale-learning abilities.

by Pintea et al. [13] to learn the size of the kernel based on the ISR:

$$l = 2\lceil k(\sigma_{basis} \times ISR)\rceil + 1 \tag{7}$$

where k is a hyperparameter that determines the extent of the approximation of the continuous basis functions. Thus, the kernel size used in convolution is determined by the largest scale in the set of internal scales which is directly parameterised by σ_{basis} and the ISR.

4. Experiments

In this section, we first validate the importance of matching the internal scales with the data scale distribution and compare the results with the suggested values of the Internal Scale Tolerance model. Secondly, we verify that we can indeed learn the internal scales by evaluating the models on the same data scale distribution and comparing the results. Next, we test the effect of initialisation and parameterisation of the scale parameter on the performance and stability of the model. Lastly, we compare our scale-learning method with existing scale-equivariant and scale-invariant approaches with fixed internal scales on the commonly used MNIST-Scale dataset [5].

In Section 4.1 and 4.2, we use the toy architecture shown in Tab. 1 on the Dynamic Scale MNIST dataset with a Logarithmically Uniform data scale distribution with a range of 1 to $2^{1.5}, 2^{2.25}$ and 2^3 corresponding to $2.83, 4.76$, and 8 scale factors of MNIST respectively. Appx. A gives a complete description of all datasets used in the experiments. A description of the training procedures can be found in Appx. B.

4.1. Validation

4.1.1 Do Internal scales really matter?

We test our assumption that the internal scale range (ISR), the factor between the largest and smallest internal scale influences the performance. Furthermore, we compare the optimal ISRs we discovered with the suggested values of the Internal Scale Tolerance model. The ISRs are chosen on a logarithmic scale in the range of [1.5, 7.65].

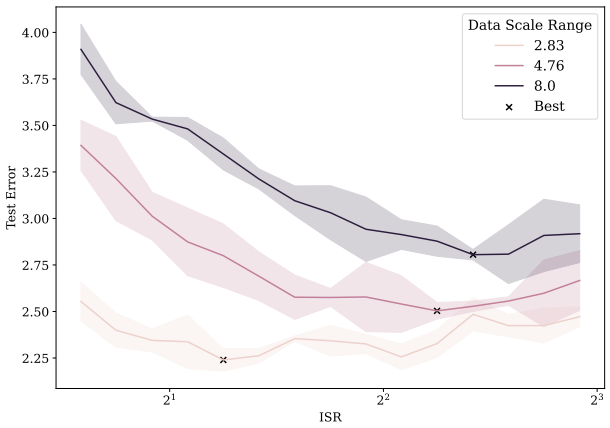

Figure 3. Impact on Test Error when varying Internal Scale Range (ISR) for different log-uniform data scale distributions from 1 scale factor of MNIST to 2.83, 4.76 and 8 scale factors MNIST. The value of the Internal Scale Range (ISR) for the best-performing model increases together with the width of the model.

The results in Fig. 3 indicate that smaller ISRs are better for narrow data scale distributions, while larger ISRs perform better for wider data scale distributions. Thus, for a log-uniform distribution that spans a small scale range narrow internal scales are preferred. Conversely, for a log-uniform distribution that spans a large scale range, wide internal scales are preferred. Looking at the test error of individual data scales in Fig. 4, we see that at the boundary regions of wide scale distributions narrow internal scales achieve significantly higher test error than wider internal scales indicating that the model cannot share features across the whole data scale distribution. Narrow internal scales perform slightly better than wider internal scales when evaluated on narrow distributions. This statement aligns with results found in Fig. 3, which indicates less test error variation between ISR values for the log-uniform distribution between 1 and 2.83.

The results of the combined optimisation of the tolerant model and the three sampling distributions can be found in Fig. 5. The optimisation leads to $\tau_{tol} = 0.459$. The optimisation fits ISR values approximately similar to the best ISRs depicted in Fig. 3. Again, the ISR values follow an increasing pattern when the data scale distribution gets wider. Additionally, Fig. 5 shows increasing gaps in the tolerance hypothesis between internal scales for wide distributions.

4.1.2 Can we learn the internal scales?

To test our scale-learning capabilities, we evaluate our scale-learning on three log-uniform data scale distributions. The ISR is parameterised according to Eq. 6 and the scale parameters are initialised with $\sigma_{basis} = 2$ and $ISR = 3$.

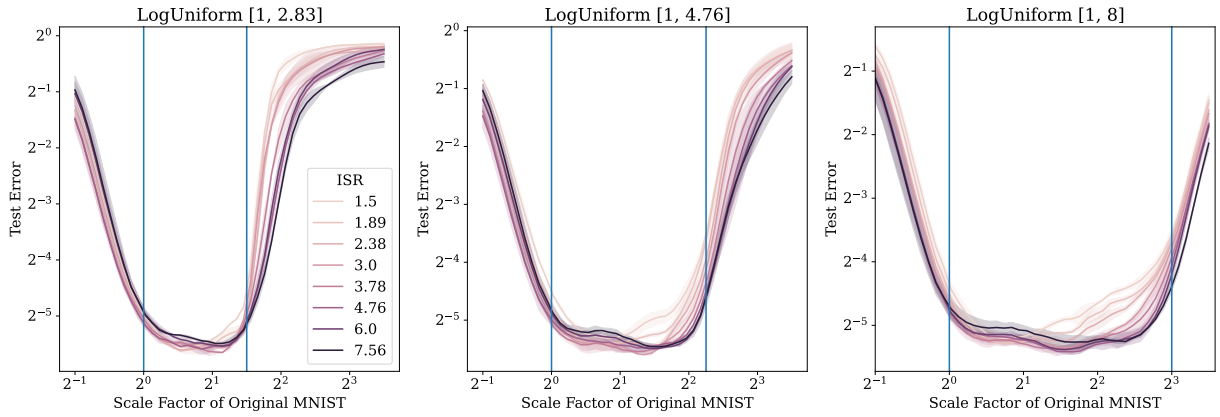

Figure 4. Test Error per data scale for multiple data scale distributions and values for the Internal Scale Range (ISR). Models with narrow internal scales especially deteriorate in performance in the large-scale region for wide distribution.

Data Scale Range	Scale Learning	σ_{basis}	ISR	Test Error
[1,2.83]	✓	1.96 ± 0.081	3.390 ± 0.545	2.291 ± 0.067
	✗	2	2.34	2.239 ± 0.060
[1,4.76]	✓	2.001 ± 0.063	3.321 ± 0.024	2.510 ± 0.084
	✗	2	4.76	2.503 ± 0.045
[1,8]	✓	1.943 ± 0.064	4.196 ± 0.159	2.872 ± 0.070
	✗	2	5.35	2.805 ± 0.028

Table 2. Learned parameters for the Basis Min Scale σ_{basis} and Internal Scale Range (ISR) compared to the configuration of the best-performing model with fixed internal scales on different ranges of the log-uniform data scale distribution. Apart from the log-uniform distribution with boundaries $[1, 2.83]$, the learned scale parameters σ_{basis} and ISR follow a similar pattern as the manually found scale parameters. The test error of our scale-learning method is also comparable to the best-performing models with fixed scales.

The results of learning the ISR and σ_{basis} compared to the best-performing ISRs without scale learning enabled can be found in Table 2. Apart from the narrowest scale distribution, the ISR values learned increase when enlarging the range of the data scale distribution. Our scale learning method gives comparable performance to the baselines while not using hyperparameter optimisation to determine the best ISR.

4.2. Model Choices

4.2.1 How does initialisation of the scales scale learning behaviour?

We test the importance of the initialisation of the internal scales by varying the starting values of σ_{basis} and ISR and report the classification error and variation in learned scale parameters. We vary the σ_{basis} between 1 and 4 and the ISR between 1.5 and 6 in a logarithmic fashion.

Table 3 show the results for the log-uniform distributions between $[1, 2.83]$, $[1, 4.76]$ and $[1, 8]$. The results indicate

Data Scale Range	Init σ_{basis}	Init ISR	Learned σ_{basis}	Learned ISR	Test Error
[1,2.83]	1	1.5	1.268 ± 0.061	2.609 ± 0.269	2.487 ± 0.108
		3.0	1.236 ± 0.139	3.300 ± 0.082	2.357 ± 0.024
		6.0	1.313 ± 0.102	4.350 ± 0.396	2.309 ± 0.021
	2	1.5	$\mathbf{1.810 \pm 0.057}$	$\mathbf{2.405 \pm 0.167}$	$\mathbf{2.260 \pm 0.025}$
		3.0	1.973 ± 0.079	3.635 ± 0.647	2.368 ± 0.055
		6.0	1.994 ± 0.012	5.336 ± 0.283	2.359 ± 0.097
	4	1.5	2.778 ± 0.092	2.521 ± 0.199	2.421 ± 0.050
		3.0	2.703 ± 0.081	3.817 ± 0.245	2.483 ± 0.089
		6.0	2.832 ± 0.001	5.211 ± 0.416	2.420 ± 0.124
[1, 4.76]	1	1.5	1.294 ± 0.110	3.462 ± 0.410	3.033 ± 0.130
		3.0	1.253 ± 0.070	3.829 ± 0.126	2.762 ± 0.087
		6.0	1.331 ± 0.091	4.612 ± 0.188	2.727 ± 0.101
	2	1.5	1.931 ± 0.068	2.932 ± 0.169	2.767 ± 0.128
		3.0	1.975 ± 0.060	3.309 ± 0.092	2.527 ± 0.120
		6.0	$\mathbf{2.041 \pm 0.092}$	$\mathbf{4.882 \pm 0.007}$	$\mathbf{2.501 \pm 0.157}$
	4	1.5	2.515 ± 0.038	3.093 ± 0.157	2.648 ± 0.073
		3.0	2.587 ± 0.040	3.638 ± 0.169	2.575 ± 0.070
		6.0	2.803 ± 0.098	5.662 ± 0.204	2.709 ± 0.078
[1, 8]	1	1.5	1.450 ± 0.130	3.767 ± 0.171	3.087 ± 0.173
		3.0	1.275 ± 0.076	4.158 ± 0.123	3.131 ± 0.179
		6.0	1.331 ± 0.097	5.423 ± 0.529	3.087 ± 0.178
	2	1.5	1.755 ± 0.156	3.444 ± 0.025	3.010 ± 0.198
		3.0	1.982 ± 0.068	4.095 ± 0.078	2.921 ± 0.082
		6.0	$\mathbf{2.079 \pm 0.061}$	$\mathbf{5.053 \pm 0.375}$	$\mathbf{2.745 \pm 0.012}$
	4	1.5	2.453 ± 0.151	3.466 ± 0.216	2.935 ± 0.036
		3.0	2.607 ± 0.078	4.401 ± 0.343	2.803 ± 0.085
		6.0	2.655 ± 0.044	5.268 ± 1.071	2.957 ± 0.140

Table 3. Mean and standard deviation of learned scale parameters (σ_{basis}, ISR) and Test Error for different initialisation of σ_{basis} and ISR for log-uniform data scale distribution between $[1, 2.83]$, $[1, 4.76]$ and $[1, 8]$. Learned σ_{basis} and ISR values are highly dependent on the values they are initialised on. Initialisation with a large ISR for a wide data scale distribution leads to significantly lower test error than initialisation at a low ISR.

that the learned σ_{basis} and ISR values can be adapted to fit the data scale distribution but the initialisation of the values has a big impact on the learned scales and thus also the test error. Initialisation with a large ISR for a wide data scale distribution leads to significantly lower test error. Fig. 6 shows the ISR over time during training, indicating the ISR stabilises after around 20 epochs while the best-performing model has a significantly larger ISR.

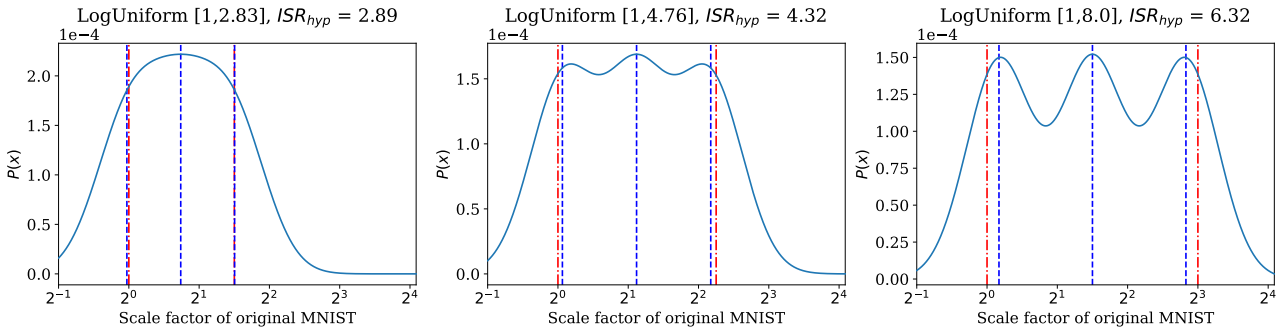

Figure 5. Results of combined optimisation of tolerance hypothesis models and three log-uniform data scale distributions. The blue dashed line indicates the proposed internal scales, while the continuous blue line represents the tolerance hypothesis for a specific data scale distribution. The red dashed lines indicate the boundaries of the loguniform distribution. The result of fitting the tolerance model results in increasing ISR_{hyp} similar to results best performing ISRs found for each data scale distribution in Fig. 3.

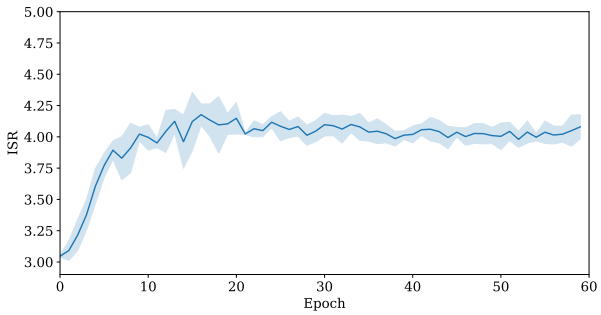

Figure 6. ISR parameter overtime for run initialised with $\sigma_{basis} = 2, ISR = 3$ on log-uniform distribution with boundaries [1,8]. After around 20 epochs, the learnable ISR stabilises while the value for the ISR of the best-performing model is significantly larger.

4.2.2 How does parameterisation of learnable scales affect learnability?

We test the importance of our scale learning parameterisation method on the stability and classification performance against other possible parameterisation methods. We initialise all scale learning approaches with the internal scales: $[2.0, 3.47, 6]$. The parameterisation methods that we compare are:

1. Learning the first scale (σ_{basis}) and the ISR using the parameterisation from Eq. 6 (Ours)
2. Learning the first scale (σ_{basis}) and the individual spacings between subsequent scales
3. Learning the individual scales directly, based on [20] but without defining intervals the internal scales adhere to.

Table 4 shows the parameterisation methods, the classification error and the variation in the learned internal scales. Unlike shown in [20] directly learning the scales without constraints between the internal scales does not lead to internal scales converging to the same value. The methods do not vary significantly in performance for the log-uniform distribution between [1, 2.83] and [1, 4.76] but this changes when training on wider distributions. All methods adjust the scales somewhat to account for the wider scale distribution but our method of learning the Internal Scale Range (ISR) is more stable and achieves significantly better test Error.

Data Scale Range	Parameterisation	Scale 1	Scale 2	Scale 3	Test Error
[1, 2.83]	Direct	1.965 ± 0.047	3.500 ± 0.193	6.235 ± 0.497	2.321 ± 0.095
	Individual Spacing	$\mathbf{1.967 \pm 0.079}$	$\mathbf{3.591 \pm 0.329}$	$\mathbf{6.930 \pm 1.374}$	$\mathbf{2.285 \pm 0.038}$
	ISR	1.960 ± 0.081	3.608 ± 0.435	6.672 ± 1.311	2.291 ± 0.067
[1, 4.76]	Direct	1.865 ± 0.046	3.357 ± 0.105	6.450 ± 0.049	2.554 ± 0.093
	Individual Spacing	1.996 ± 0.013	3.626 ± 0.158	6.830 ± 0.167	2.565 ± 0.061
	ISR	$\mathbf{2.001 \pm 0.063}$	$\mathbf{3.647 \pm 0.127}$	$\mathbf{6.647 \pm 0.255}$	$\mathbf{2.510 \pm 0.084}$
[1, 8]	Direct	1.689 ± 0.109	3.262 ± 0.107	6.997 ± 0.282	3.057 ± 0.015
	Individual Spacing	1.902 ± 0.085	3.648 ± 0.165	8.093 ± 0.229	3.007 ± 0.049
	ISR	$\mathbf{1.943 \pm 0.063}$	$\mathbf{3.977 \pm 0.053}$	$\mathbf{8.145 \pm 0.057}$	$\mathbf{2.872 \pm 0.070}$

Table 4. Mean and standard deviation of learned scales and Test Error of different parameterisations for multiple log-uniform distributions with internal scales initialised as [2.0, 3.46, 6.0]. Learning the ISR leads to more stable learned internal scales and better performance for wide distributions further away from the initialised scales.

4.3. Does Learnable scales improve baseline on popular scale-equivariant image classification baselines?

We compare our scale-learning ability against existing scale-equivariant baselines by evaluating on the MNIST-Scale [5] dataset. We reuse the code provided in DISCO [15] to compare our results to a baseline CNN and other methods that take into account scale variations such as SI-ConvNet [5], SS-CNN [4], SiCNN [19], SEVF [11], DSS [17], SESN [14] and DISCO [15]. All methods adopt the same training strategy apart from our scale learning method having a different learning rate scheduler for its scale parameters (Appx. B).

Model	MNIST-Scale	MNIST-Scale+	# Params.
CNN	2.02 ± 0.07	1.60 ± 0.09	495k
SiCNN	2.02 ± 0.14	1.59 ± 0.03	497k
SI-ConvNet	1.82 ± 0.11	1.59 ± 0.10	495k
SEVF	2.12 ± 0.13	1.81 ± 0.09	495k
DSS	1.97 ± 0.08	1.57 ± 0.09	475k
SS-CNN	1.84 ± 0.10	1.76 ± 0.07	494k
SESN (Hermite)	1.68 ± 0.06	1.42 ± 0.07	495k
DISCO	1.52 ± 0.06	1.35 ± 0.05	495k
Ours (Learn ISR)	1.72 ± 0.05	1.44 ± 0.09	495k
Ours (Learn Spacings)	1.70 ± 0.10	1.50 ± 0.08	495k
Ours (Learn Scales Directly)	1.74 ± 0.06	1.50 ± 0.08	495k

Table 5. Classification error of Vanilla CNN and other methods that take into account scale variations in the data. The error is reported for runs with and without data scale augmentation, the "+" denotes the use of data scale augmentation. Learnable scale approaches perform on par with the non-learnable scale baseline SESN [14].

Model	Scale 1	Scale 2	Scale 3	Scale 4
SESN	1.5	1.89	2.38	3
DISCO	1.8	2.27	2.86	3.6
Ours (Learn ISR)	1.390 ± 0.016	1.890 ± 0.061	2.572 ± 0.164	3.503 ± 0.338
Ours (Learn Spacing)	1.390 ± 0.011	1.930 ± 0.099	2.614 ± 0.300	3.776 ± 0.600
Ours (Learn Scales Directly)	1.375 ± 0.008	1.889 ± 0.054	2.383 ± 0.044	3.297 ± 0.086
Ours (Learn ISR)+	1.381 ± 0.013	2.066 ± 0.012	3.092 ± 0.038	4.627 ± 0.095
Ours (Learn Spacing)+	1.373 ± 0.013	1.859 ± 0.069	2.847 ± 0.091	4.646 ± 0.306
Ours (Learn Scales Directly)+	1.360 ± 0.014	1.741 ± 0.062	2.485 ± 0.050	3.775 ± 0.083

Table 6. Learned scale parameters of our model on MNIST-Scale compared to values chosen in SESN [14]. The "+" denotes the use of data scale augmentation. The learned scales of all scale learning methods are much wider than the initialised scales in SESN [14] and DISCO [15], especially when scale data augmentation is used.

We also compared our Internal Scale Range based parameterisation with other parameterisations such as: learning the individual spacings between internal scales and learning the scales directly. Learning the scale directly is similar to the approach taken by [20] but without defining intervals the internal scales have to adhere to.

As can be seen from Table 5, the performance of the scale learning approaches are very comparable with SESN [14] without learnable scales. All three scale-learning approaches achieve test error performance within 1 standard deviation of SESN with fixed scales. The learned scales, found in Table 6, are consistently more spread out than the default scales used in SESN [14] especially when scale data augmentation is used.

5. Discussion

We have shown to be able to learn the internal scales, but the problem of choosing the number of internal scales remains an issue. For wide scale distribution, wide internal scales achieve the best performance. However, if the models were truly scale-equivariant, the resulting test error would be similar to the test error for the log-uniform data scale distribution between 1 and 2.83. More specifically, if the spacing between the internal scales is too large the implied scale-equivariance over the entire range of the internal scales does not hold up. The model again needs to learn duplicate filters to cover the entire data scale range. This hypothesis also matches up with our Internal Scale tolerance model seen in Fig. 5, which shows dips in between internal scales. We expect that increasing the number of internal scales restores the scale equivariance over the entire scale group with the downside of reduced computational efficiency.

Another difficulty of learning the scales is the initialisation of the internal scales. We have found that the initialisation of the internal scales has a large impact on the learned scales and as a result the performance. However, we do expect that this can be resolved by tuning the training procedure.

In addition, our method adds a significant computational overhead since it has to reconstruct the dynamic filter basis functions on each step instead of being able to reuse the fixed multi-scale basis. However, hyperparameter optimisation of the scale parameters would take significantly longer.

To conclude, we have shown to be able to make a good approximation of good scale parameters if the data scale distribution is known using our Internal Scale tolerance model. In the more realistic scenario of not knowing the data scale distribution, we have developed a method to learn the internal scales instead. Our results show that we can indeed learn the internal scales but the initialisation plays an important role in the process. We demonstrate that all scale-learning parameterizations perform on par with state-of-the-art scale-equivariant approaches that have been configured on MNIST-Scale. However, our method of learning the internal scale range is able to adapt to a wider data scale distribution.

Learnable scales did not add significant gains in classification tasks but for other tasks with larger scale-variations the importance of choosing internal scales becomes more important. We leave this as future work. Furthermore, we anticipate that the ability to learn the internal scales is especially beneficial in more complicated scenarios with more complicated data scale distributions, like a Normal distribution. To learn the internal scales for more advanced data scale distributions it might be essential to find a way to additionally learn or adapt the number of internal scales based on a heuristic.

References

[1] Gregory W. Benton, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. Learning invariances in neural networks. *CoRR*, abs/2010.11882, 2020. 4, 5

[2] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene la-

beling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, 2013. 3, 4

[3] Luc M.J Florack, Bart M ter Haar Romeny, Jan J Koenderink, and Max A Viergever. Scale and the differential structure of images. *Image and Vision Computing*, 10(6): 376–388, 1992. Information Processing in Medical Imaging. 4

[4] Rohan Ghosh and Anupam K. Gupta. Scale steerable filters for locally scale-invariant convolutional neural networks. *CoRR*, abs/1906.03861, 2019. 3, 4, 9, 11

[5] Angjoo Kanazawa, Abhishek Sharma, and David W. Jacobs. Locally scale-invariant convolutional neural networks. *CoRR*, abs/1412.5104, 2014. 3, 4, 6, 7, 9

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 12

[7] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. 6

[8] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. 11

[9] Tony Lindeberg. Scale-covariant and scale-invariant gaussian derivative networks. *CoRR*, abs/2011.14759, 2020. 3, 4, 6

[10] Tony Lindeberg and Jan-Olof Eklundh. Scale-space primal sketch: construction and experiments. *Image and Vision Computing*, 10(1):3–18, 1992. 4

[11] Diego Marcos, Benjamin Kellenberger, Sylvain Lobry, and Devis Tuia. Scale equivariance in cnns with vector fields. *CoRR*, abs/1807.11783, 2018. 3, 4, 9

[12] Hanieh Naderi, Leili Goli, and Shohreh Kasaei. Scale equivariant cnns with scale steerable filters. In *2020 International Conference on Machine Vision and Image Processing (MVIP)*, pages 1–5, 2020. 3, 4

[13] 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. *CoRR*, abs/2106.03412, 2021. 4, 7

[14] Ivan Sosnovik, Michal Szmaja, and Arnold W. M. Smeulders. Scale-equivariant steerable networks. *CoRR*, abs/1910.11093, 2019. 3, 4, 5, 6, 9, 10, 11, 12

[15] Ivan Sosnovik, Artem Moskalev, and Arnold W. M. Smeulders. DISCO: accurate discrete scale convolutions. *CoRR*, abs/2106.02733, 2021. 3, 4, 9, 10, 11

[16] Zikai Sun and Thierry Blu. Empowering networks with scale and rotation equivariance using a similarity convolution, 2023. 4, 5

[17] Daniel Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems 32*, pages 7364–7376. Curran Associates, Inc., 2019. 3, 4, 9

[18] Yichong Xu, Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, and Zheng Zhang. Scale-invariant convolutional neural networks. *CoRR*, abs/1411.6369, 2014. 3, 4

[19] Yichong Xu, Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, and Zheng Zhang. Scale-invariant convolutional neural networks. *CoRR*, abs/1411.6369, 2014. 9

[20] Yilong Yang, Srinandan Dasmahapatra, and Sasan Mahmoodi. Scale-equivariant unet for histopathology image segmentation, 2023. 4, 5, 6, 9, 10

[21] Wei Zhu, Qiang Qiu, A. Robert Calderbank, Guillermo Sapiro, and Xiuyuan Cheng. Scale-equivariant neural networks with decomposed convolutional filters. *CoRR*, abs/1909.11193, 2019. 3, 4

A. Dataset Description

A.1. Dynamic Scale MNIST

The Dynamic Scale MNIST pads the original 28x28 images from the MNIST dataset [8] to 168x168 pixels and then on initialisation of the dataset, an independent scale for each sample is drawn from the chosen scale distribution. Only scales e larger than 1 are sampled during training time to prevent the influence of information loss which occurs when downsampling the data. Since each digit is upsampled upon accessing no additional storage is needed to use this dataset for various scale distributions. After initialisation the dataset is normalised.

Additionally, this dataset can also be used to evaluate across a range of scales by sampling each test digit individually on multiple scales. The scales to evaluate are rounded to the nearest half-octave of 2. The number to evaluate is determined by the range of octaves times 10. Thus for Fig. 4, 45 scales are sampled between $2^{-0.5}$ and $2^{3.5}$ in a logarithmic manner. The underlying MNIST dataset [8] is split into 10k training samples, 5k validation samples, and 50k test samples and 3 different realisations are generated and fixed.

A.2. MNIST-Scale

The images in the MNIST-Scale dataset are rescaled images of the MNIST dataset [8]. The scales are sampled from a Uniform distribution in the range of 0.3 - 1.0 of the original size and padded back to the original resolution of 28x28 pixels. The dataset is split into 10k training samples, 2k validation samples and 50k test samples and 6 realisations are made.

B. Experimental Details

B.1. Dynamic Scale MNIST

For all experiments on the Dynamic Scale MNIST dataset, we use the toy architecture highlighted in Table 1. Each filter is parameterised by 5x5 parameters but the extent varies according to Eq. 7 with $k = 4$. If not otherwise specified the scales are chosen or initialised according to Eq. 6 with $\sigma_{basis} = 2, ISR = 3, N_S = 3$.

B.2. MNIST-Scale

Following SS-CNN [4], SESN [14] and DISCO [15] we use 1 upsample layer which upsamples the digits by 2, 3 convo-

Model	MNIST-Scale
SiCNN	[0.3, 0.45, 0.67, 1.0, 1.49, 2.23, 3.33]
SI-ConvNet	[0.3, 0.45, 0.67, 1.0, 1.49, 2.23, 3.33]
DSS	[1,2]
SS-CNN	[1.0, 1.33, 1.67, 2.0, 2.33]
SESN (hermite)	[1.0, 1.26, 1.59, 2.0]
DISCO	[1.0, 1.26, 1.59, 2.0]

Table 7. Internal Scales configuration for different models on the MNIST-Scale dataset.

lutional layers, scale projection layer if necessary, followed by 2 fully connected layers. Each filter is parameterised by 7x7 parameters. In the case of CNN, this means a filter of 7x7 pixels while for kernel parameterisation methods the actual kernel size can vary depending on the scales. The number of parameters are kept roughly the same, around 495k using different number of hidden channels in the convolution used. The fully connected layers have 256 and 10 hidden channels respectively. The internal scale factors for the models and their values are summarised in Table 7 for each model. Following SESN [14] the internal scales for our method are initialised according to Eq. 6 with $\sigma_{basis} = 1.5, ISR = 2, N_S = 4$. The kernel size is learnable according to Eq. 7 with $k = 4$

We use an Adam optimizer [6] with default settings $\beta_1 = 0.9, \beta_2 = 0.999$ and train for 60 epochs with a batch size of 128. The learning rate for scale learning parameters is initialised at 0.005 and a warmup is used of 10 epochs. The learning rate for other parameters is initialised at 0.01 and both learning rates are divided by 10 every 20 epochs. Additionally, if scale learning is enabled we use a warmup of 10 epochs for the scale learning parameters with a learning rate of 0.01. The learning rate for the scale learning parameters is also divided by 10 every 20 epochs.

3

Supplemental Materials

In this chapter, the technical required knowledge will be explained. First, a general overview of deep learning in computer vision will be given. Secondly, Convolutional Neural Networks and their translation-equivariance property will be described in more detail. Additionally, we will discuss how we achieve scale-equivariance using a different kind of convolution.

3.1. Deep Learning for Images

Deep learning refers to a subset of methods in machine learning whose aim is to learn useful representations in training data. The representations are learned using an architecture called Neural Networks. Neural Networks consist of interconnected nodes, or neurons, divided into layers. This layer of neurons is referred to as a fully connected layer since every neuron in the layer is connected to all other neurons in the previous layer. Each neuron takes in several inputs, performs a weighted sum, and outputs a single value. The computation of the output of the network is referred to as the forward pass since information passes through each layer. A helpful way to visualise this forward pass is to think about the weights being on the line between nodes. The goal of the network is to change the weights of all of these neurons in such a way as to output the correct value. The main task we are dealing with in our paper is classification and thus the correct value would be the label that corresponds to the input image. A visual representation of a network can be seen in Fig. 3.1.

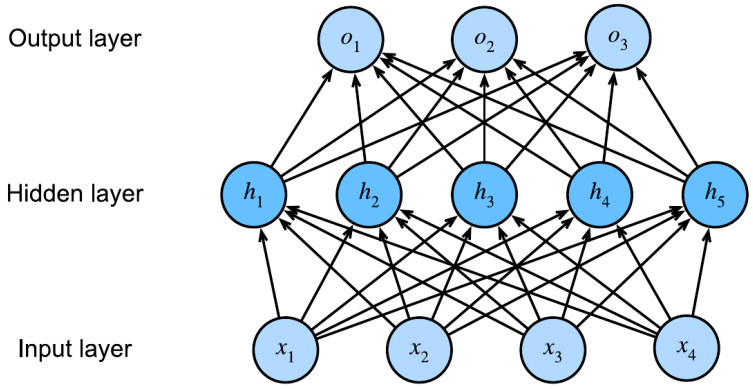

Figure 3.1: A visual representation of a (fully connected) Neural Network with 4 inputs, a hidden layer with 5 units and 3 outputs. From [4].

3.1.1. Backpropagation

Deep learning methods use the backpropagation algorithm to efficiently calculate the derivatives of the weights with respect to the loss function. The loss function calculates the distance between the prediction of the Neural Network and the wanted output. Computing the gradients of each weight with respect to the loss function individually is a very costly operation. The backpropagation algorithm uses the chain chain rule to efficiently compute the derivatives of each parameter. The chain rule makes it possible to reuse already calculated derivatives. The final derivatives are later used to update the weights of the network and train the network.

Suppose we have the network highlighted in Fig. 3.2 consisting of two input pixels, two (hidden) nodes, and one output node. The forward pass is calculated as $y' = w_y^1(w_f^1 x_1 + w_f^2 x_2) + w_y^2(w_g^1 x_1 + w_g^2 x_2)$. The prediction is compared to the actual label of the sample and the loss L is calculated. The backward pass algorithm starts at the last node with $\frac{dL}{dy'}$ and $\frac{dL}{dw_o^a} = \frac{dL}{dy'}\frac{dy}{dw_o^a}$ for $a \in [0, 1]$. Deeper in the network we get $\frac{dL}{dw_g^1} = \frac{dL}{dy}\frac{dy}{dw_g^1}$. Notice that we can reuse the previously calculated value for $\frac{dL}{dy}$ here. If we compute the topological ordering of the graph and evaluate the derivatives in the reverse order we can, much more efficiently, compute the derivatives with respect to the weights.

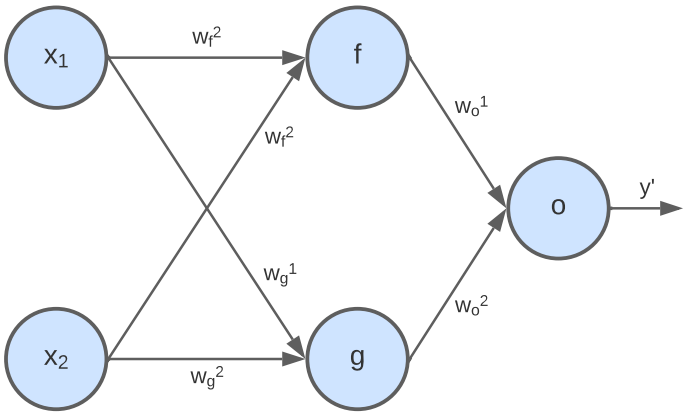

Figure 3.2: An example Neural Network to help visualize the backpropagation algorithm. It consists of 2 input nodes 2 x_1 and x_2, 2 hidden nodes f and g and an output node y.

3.1.2. Optimization

We have shown how to compute the forward pass of the network to compute the prediction of the network and the backward pass to calculate the gradients of the network but have not shown how to update the weights. The weights of the network are updated as follows: $w' = w - \epsilon\frac{dL}{dw}$. ϵ represents the learning rate and it controls the magnitude of how much to update the weights. The gradient with respect to the loss is calculated with backpropagation and is often not calculated over the whole dataset but over a small subset or batch of the data. This results in slightly more noisy gradients but leads to faster convergence in the end. By repeated updating of the weights eventually, the network is trained to perform the task.

Optimizers in deep learning are responsible for tuning the learning rate during training. This is an important aspect since later optimisation steps in the training phase require precise updates of the weights and thus a lower learning rate. Furthermore, isotropic parameter spaces can lead to large oscillations in the gradient steps for certain parameters. Optimizers leverage past gradient update statistics to update the learning rate, aiming for faster and better convergence.

The optimizer we use in our experiments is called the Adam Optimizer. First, Adam uses momentum that smooths the average of the gradient using exponentially weighted moving averaging (EWMA). Momentum uses the EWMA of past gradient updates to increase the step size in the directions the gradients are moving. Secondly, it makes use of an algorithm called RMSprop. RMSprop also uses the

EWMA of the zero-centred variance of the gradients to normalise the learning rate for each parameter. This has the effect of stabilising the optimisation process and reducing oscillation.

3.1.3. Other Operations

Fully connected layers are not the only type of layers or operations commonly used in Neural Networks. In this section, we highlight other popular layers and operations we use in Chapter 2 apart from Convolution Layers that will be explained in the next section.

Activation functions are used at each neuron output to add non-linearity. Without activating functions, a fully connected Neural Network would be purely linear and capable of only modelling other linear functions. Activation functions are used to extend the capacity of the network to non-linear functions. A common activation function is the rectified linear unit or ReLu defined by $g(x) = \max(0, x)$. ReLU remains close to linear, preserving many properties of linear models that are helpful, such as easy optimisation and good generalizability.

Another operation which is commonly used in practice is pooling. Pooling functions summarise the responses over a certain neighbourhood. Pooling can be used for multiple purposes such as reducing the number of output nodes of a layer, allowing the network to take in variable-size inputs and achieving invariance. Translation invariance means that if we move the object in the image, the output of the pooling layer does not change.

3.2. Convolutional Neural Networks

Convolutional Neural Networks are another type of neural network which are especially successful on grid-like data where patterns play an important role, such as time series data or images. Convolution in the image domain consists of sliding a flipped kernel of weights over the input of the layer. The output of each node is the multiplication of the flipped kernel entries with the entries in the surrounding input data of the same size. Deep Learning libraries used in practice often do not flip the kernel. An example of a convolution without a flipped kernel can be seen in Fig. 3.3. In this example, no padding is applied so that the resulting output is smaller than the input. The output of a convolutional layer is often called a feature map since it is a mapping of how much a certain feature or kernel is found in the input. Using multiple kernels in a convolution layer allows us to detect and locate multiple features in the input.

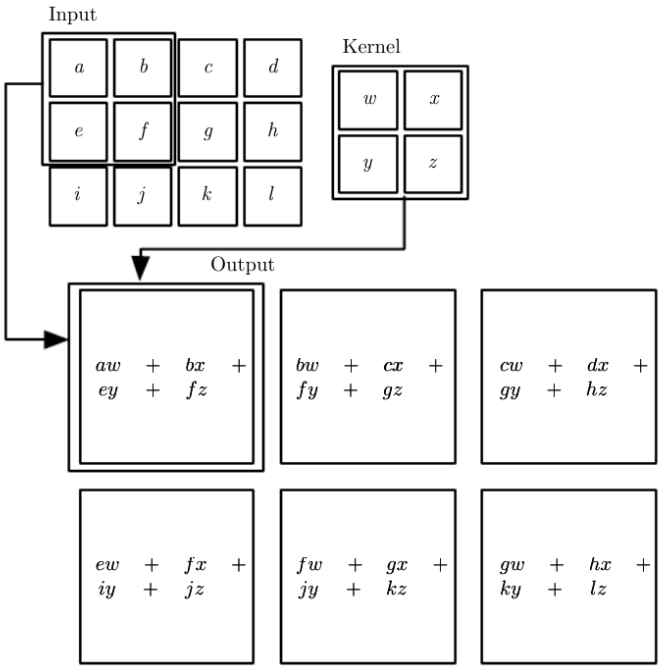

Figure 3.3: Example of a Convolution Operation on an image without flipped kernel and no padding of the input. From [1]

Convolutional layers allow for many useful properties that can improve the network. The kernel used in convolutional layers is often significantly smaller than the input feature map and thus the output

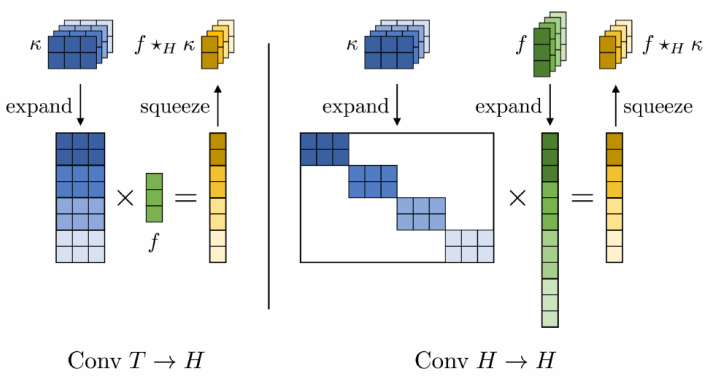

Figure 3.4: Scale-Convolution operation in practice. Left: the input feature map f does not contain scale dimension. Right: the input feature map f contains a scale dimension. For the sake of simplicity, the spatial component of the input and kernel is left out. From: SESN [3].

takes significantly less time to compute. Moreover, the kernel is used at various locations making it possible to locate features at multiple locations without having to learn separate weights reducing the number of parameters needed.

3.2.1. Translation-Equivariance

An important property of convolution is translation-equivariance. Equivariance to certain transformations means that transforming the input transforms the output similarly. In the case of convolution, moving a certain object in the image changes the output feature map in the same way. Translation equivariance is especially helpful in detection tasks where the objective is both classifying the object and locating it.

3.2.2. Scale-Equivariance

Convolution is by default not equivariant to scaling or other transformations. A common way to extend the equivariance property to other transformations is through the use of group convolutions. Group convolutions apply multiple convolutions of the image with transformed versions of the same kernel.

In practice, group convolutions are implemented in a slightly different manner. We will only explain the case where the group is the scale-translation group, but it works for other group transformations too. Assume you have a kernel size of 5 by 5 pixels and have 32 features in the convolution layer and an input feature map of 16 channels. A regular convolution would perform a convolution with the input feature map and the tensor of size $[16, 32, 5, 5]$. For the scale convolution, the kernels are first scaled according to a set of scales S. This has the additional effect of changing the kernel size since only upscaling is used. The resulting tensor of shape $[16, 32, S, V', V']$ is then expanded to size $[16, 32 \times S, V', V']$. The output feature map is then of size $[16 \times S, U, U]$, this is expanded to $[16, S, U, U]$. The output feature map consists of all feature maps of the transformed kernels stacked together.

A special case of scale convolution arises when the input feature comes from another scale convolution layer. In this case, the kernel is transformed similarly and again has a tensor shape in the form of $[C_out, C_in, S, V', V']$ but now, to preserve the scale equivariance property, each kernel with scale index s_i is convolved with the input feature map with the same scale index s_i. A visualization of the two cases can be seen in Fig. 3.4.

The benefits of scale-equivariance are similar to the benefits of translation-equivariance. Sharing kernels across various scales reduces the number of parameters we need since the network does not have to learn a separate representation for a scaled object.

References

[1] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[2] Silvia L. Pintea et al. “Resolution learning in deep convolutional networks using scale-space theory”. In: *CoRR* abs/2106.03412 (2021). arXiv: 2106.03412. URL: https://arxiv.org/abs/2106.03412.

[3] Ivan Sosnovik, Michal Szmaja, and Arnold W. M. Smeulders. “Scale-Equivariant Steerable Networks”. In: *ArXiv* abs/1910.11093 (2019). URL: https://api.semanticscholar.org/CorpusID:204852197.

[4] Aston Zhang et al. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG].