ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning For Computer Vision

DOGNEXT: CONVOLUTIONAL NEURAL NETWORK WITH DIFFERENCE OF GAUSSIAN KERNELS

CANDIDATE
Davide Gardenal

SUPERVISOR
Prof. Salti Samuele

CO-SUPERVISOR
Prof. Akihiro Sugimoto

Academic year 2023-2024

Session 1st

Abstract

Convolutional Neural Networks have seen a lot of advancements over the years. All modern CNNs moved from using classical convolutions, that process the spatial and channel dimension simultaneously, to depth-wise separable convolutions, effectively separating the spatial aggregation with the channel mixing. This has enabled to get a better picture on how kernels are processing the images, in fact, recently it was discovered that many of them share a set of common patterns, resembling the Difference of Gaussian (DoG) function. This has led us into formulating a new DoG-parametrized convolutional kernel called DoGConv that aims at recreating the common kernel patterns observed in the wild. With this we created DoGNeXt, a CNN based on the ConvNeXt V2 architecture. We test DoGNeXt on ImageNet1K and MedMNIST achieving remarkable results, beating ConvNeXt V2 when the training data is scarce. Thanks to the DoGConv parametrization we are able to resize the kernels without retraining. This property is used to improve the performance of the network when used on images with small resolutions. In this setting DoGNeXt is able to perform remarkably well. Finally, we highlight a lack of augmentation for small images in the classic ImageNet1K training recipe. We propose an additional augmentation that is able to outperform, across a wide range of image resolutions, the vanilla training, performing on par with DoGNeXt kernel resizing technique.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) are one of the building blocks of many modern computer vision architecture. With the advent of AlexNet [17] a lot of focus has been put on deep convolutional neural networks, and thanks to the ILSVRC (ImageNet) competition [26], image classifiers became more and more powerful, even surpassing the human performance [27]. It became clear with Inception [30] that the convolutional part of the model was responsible for the majority of the performance, since the hard part of the problem is the feature extraction, that consists in transforming an image in a dense, semantically meaningful representation. Most of the backbones that different architecture use to accomplish their task, mostly object detector or segmentation models, are pre-trained on ImageNet or similar datasets. This is because the representation that image classifiers learn is generic enough to improve the performance on these tasks, even if the training data is scarce and the objective is particularly different. Therefore, improving the performances backbones is crucial for many downstream tasks. The overall paradigm to build feature extractors shifted over the years from classic convolutions to group convolutions with ResNeXt [35] and depth-separable convolution with MobileNet [12]. The main drive for this was primarily to reduce the computational cost of the operation, therefore making faster models. This conception shifted with ConvNeXt[22] that used depth-separable convolutions in

combination with point-wise convolutions to outperform Swin Transformers [21], a special type of Vision Transformer (ViT), on key benchmark like ImageNet1K, ImageNet22K, COCO object detection [20] and ADE20K image segmentation [39].

The work of Babaiee et al. [3] showed that many CNNs that use depth-separable convolutions share a unique hidden characteristic. In fact, the authors noticed that, by plotting the kernels of trained models, a lot of similar patters were repeating, even between models with vastly different architectures. They observed a strong resemblance between these patterns and the Difference of Gaussian (DoG) function and it's derivatives. This has led them to create a clustering solution that could enumerate the most popular patterns in order to have a quantitative measure to compare different models. With this, they observed a strong correlation between the clusterability of the kernels and the performance of the network, leading to speculations regarding the nature and role of the patterns. This has inspired us to build DoGNeXt, a CNN based on the ConvNeXt V2[33] architecture that uses DoG-parametrized kernels. In this work we provide a complete formulation of the DoG-parametrized kernels, called DoGConv, to best resemble the empirical observations.

Originally, our work was more focused on the problem of classification at different resolutions. The objective was to develop a novel technique that could better withstand scale changes. This is a closely related problem found in object detection, where small objects are usually detected with less accuracy. Many different techniques have been developed to mitigate the issue, most notably Feature Pyramid Network (FPN) [18], but little was done to improve scale robustness on the backbones directly. In the literature we can find many examples of scale equivariant networks [23][34][40][28][29], developed with different approaches, but none of them have been trained and tested on challenging benchmarks, therefore it is hard to assess whether they are useful in real world applications or not. In our work we put the main focus on getting good performances on the challenging ImageNet1K benchmark, to

then shift our attention to some possible application of our resizable kernel parametrization.

Our experiments achieve a remarkable level of accuracy, near the one of ConvNeXt V2 [33]. We tested DoGNeXt also in the medical domain with the MedMNIST [36] classification task, managing to beat ConvNeXt V2 by a small margin. Unfortunately, our quest for improving scale robustness did not produce the results we had hoped for, showing that the approach is viable only when training in a specific and articulated way. A noteworthy discovery was the lack of a meaningful scale augmentation for small objects in the classic ImageNet1K training recipe. In fact, we need to remember that the training recipe was developed over the years to get the best possible accuracy on the standard validation resolution, and not across a diverse spectrum. We propose an additional step during training that boosted the performance on small images by significant margins, suggesting the fact that the limitation observed is due to a lack of exposure during training to small images and not inherit with the architecture itself.

Chapter 2

Background

In this chapter we are going to present the datasets used in this work and the original inspiration that led to the development of DoGNeXt.

2.1 Datasets

2.1.1 ImageNet1K

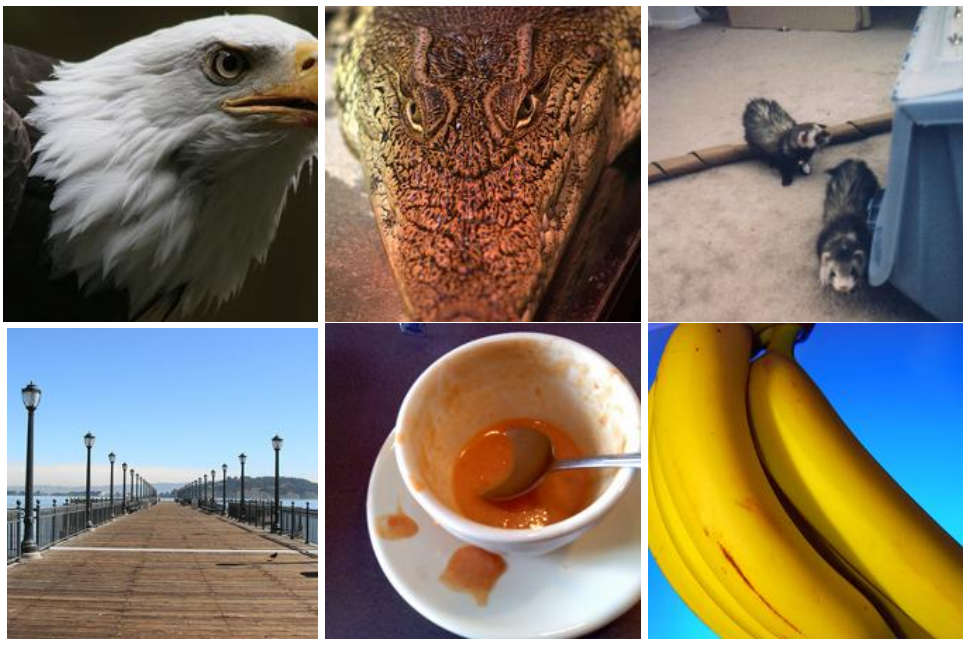


Figure 2.1: Example images from ImageNet1K.

ImageNet1K [26] is the most famous image classification problem to this day. Many major computer vision architecture like ResNet[10], EfficientNet[31] and ConvNeXt[22][33] are trained or pre-trained on it, creating foundational work for many models with countless applications. The dataset is a collection of 1.3 million natural images divided into 1000 different classes. The images display different resolutions and aspect ratio, therefore the convention is to use $224px \times 224px$ crops when training and evaluating. In Figure 2.1 we can see a collection of some random crops drawn from ImageNet1K.

2.1.2 MedMNIST

MedMNIST [36] is a large-scale MNIST-like dataset collection of standardized biomedical images, consisting of 12 datasets in 2D and of 6 in 3D. All datasets are available with resolution 28×28 or 224×224. For the purpose of this work we are going to use the 2D datasets at a resolution of 224×224 to resemble ImageNet. In Table 2.1 we have a summary of each of the task in the dataset. As we can see the datasets vary in size and in the number of classes. In Figure 2.2 we can see an example image from each of the task, that compared to ImageNet's, exhibits difference features. This is the main reason we chose MedMNIST as a supplementary benchmark to evaluate DoGNeXt.

It is important to note that every dataset is balanced, therefore we don't have any class imbalance that could skew the results and would require further attention from our side. Moreover, this enables us to compare the different models using just the class accuracy and not other metrics, like F_1, that are less sensitive to the class distribution.

2.2 CNNs and the scale problem

With the advent of object detection models a problem presented: scale variation. Small objects are notably harder to detect due to the absence of scale equivariance in convolutions. Many different techniques over the years have

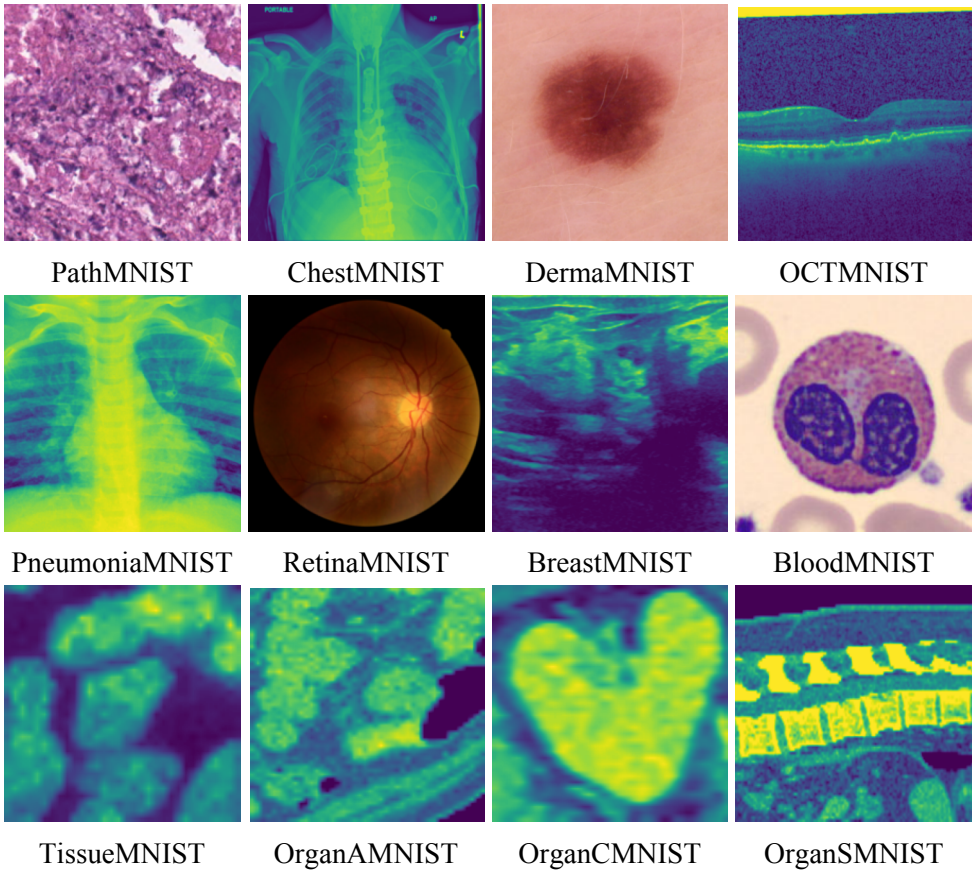


Figure 2.2: Example images from MedMNIST datasets.

been developed to mitigate this, most notably Feature Pyramid Network (FPN)[18], but the problem remains. FPN is now used in object detectors like R-CNN [24] or RetinaNet [19] style networks as it improves performance on a wide range of scales. We argue that one reason for low performance with small objects has to do with the backbones of the detectors. Our primary objective is to develop an architecture that is more resistant to scale changes, or alternatively, that can be adapted to work at different scales without retraining.

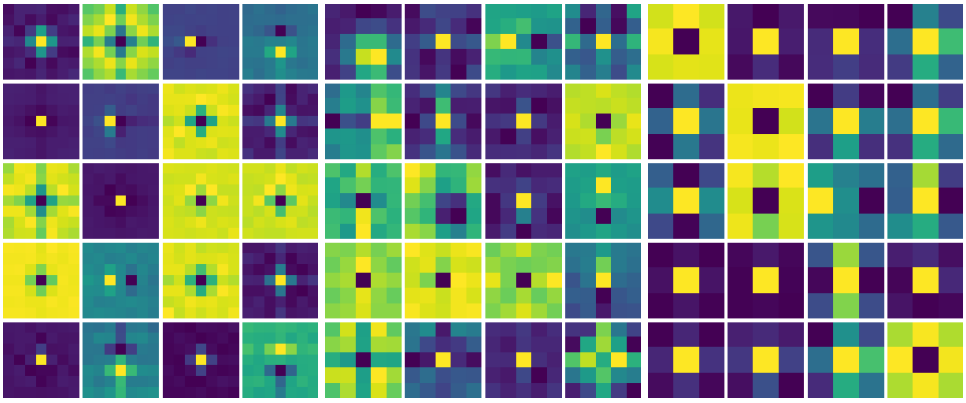
There are many different attempts to build models that are scale invariant and equivariant. For clarity, we define a network invariant to scale where given the same input image at different scales it produces the same exact result for each of them. Meanwhile, we have equivariance, with regard to scale, when the output is proportional to the input scale. In the case of image classifiers, we are working with discrete classes and there is not a difference between

Table 2.1: MedMNIST sub-dataset summary.

Name	Data Modality	Number of Classes	Number of Images			
			Total	Train	Validation	Test
PathMNIST	Colon Pathology	9	107,180	89,996	10,004	7,180
ChestMNIST	Chest X-Ray	2	112,120	78,468	11,219	22,433
DermaMNIST	Dermatoscope	7	10,015	7,007	1,003	2,005
OCTMNIST	Retinal OCT	4	109,309	97,477	10,832	1,000
PneumoniaMNIST	Chest X-Ray	2	5,856	4,708	524	624
RetinaMNIST	Fundus Camera	5	1,600	1,080	120	400
BreastMNIST	Breast Ultrasound	2	780	546	78	156
BloodMNIST	Blood Cell Microscope	8	17,092	11,959	1,712	3,421
TissueMNIST	Kidney Cortex Microscope	8	236,386	165,466	23,640	47,280
OrganAMNIST	Abdominal CT	11	58,850	34,581	6,491	17,778
OrganCMNIST	Abdominal CT	11	23,660	13,000	2,392	8,268
OrganSMNIST	Abdominal CT	11	25,221	13,940	2,452	8,829

invariance and equivariance if we look at the class output, instead, we should consider the final activation of the model, just before the classification head. Arguably, equivariance has more applications, for example in object detection, therefore we will focus more on it. One possible approach is to work with Fourier or Riesz transforms [23][5][15], in this case the idea is to use the transform to build the kernels or to transform the activation and perform the operation in the Fourier domain. While this approaches often achieve an excellent level of equivariance, they are often limited in efficiency due to the usage of the DFT (Discrete Fourier Transform), which is not well parallelize for GPU computation. A different approach is based on building a convolution operation that is scale equivariant by processing the input at different scales [34][40][28][14][16][9][29], this most of the time has the massive drawback of increased computational cost. The idea that many use to process a different scale is to build convolutional kernel with some function basis that the network optimize and fuse. This idea is used also in DoGNeXt with the difference that we don't aim at complete scale equivariance. All of them share the same evaluation procedure that consists in the use of Scale-MNIST[29] and STL-10[8] to assess equivariance. While these benchmarks are useful for the purpose, we argue that it is hard to understand the real world performance of the techniques in challenging situations like ImageNet1K. For example, a scale equivariant model that performs worse than one that was simply scale-augmented does

not have a meaningful relevance in practice.



(a) ConvNeXt V2 Tiny (b) EfficientNet (c) MobileNet V3

Figure 2.3: Random set of kernels drawn from different architectures. Source [3]

DoGNeXt draws inspiration from them with the idea to build a kernel basis that works in the challenging ImageNet1K with good performances.

A different approach comes from FlexiViT [6]. In fact, ViT are limited in the patch size they can use, changing it results is a noticeable performance decrease. FlexiViT uses some clever weight interpolation techniques to be able to perform training at different path resolutions without the need of different patchifiers for each path dimension. This enables, at inference, to decide the patch size depending on the level of accuracy desired and computational resources available. We argue that their multi-patch training hides an implicit multi-scale augmentation, therefore similar results can be achieved with CNNs by increasing stride or dilating kernels. DoGNeXt has the added benefit of being able to fully utilize the input image thanks to resizable kernels, therefore, we will cover this setting in chapter 4.

2.3 DoGConv inspiration

In [3] the authors analyzed many different CNNs that use depth-separable convolutions, in figure 2.3 we can see some kernels drawn at random. We

can clearly distinguish some common patters among them, even if the kernel dimension is different. Moreover, Babaiee et al. [3] developed a clustering technique in order to systematically enumerate the patters; the results can be seen in figure 2.4. Finally, they observed an interesting correlation shown in table 2.2. In fact, we can notice how the greater the percentage of clustered kernel the better the performance. It is important to point out that training with more data (ImageNet22K instead of ImageNet1K) has a great influence in both accuracy (on ImageNet1K) and on the amount of clustered kernels.

Table 2.2: ImageNet1K Accuracy@1 and percentage of clustered kernels in different network architectures and training modalities. Source [3].

Model	Parameters	Accuracy@1	Kernels Clustered
ConvMixer 768_32	28 M	**83.9%**	**98.16%**
ConvNextV2 tiny 22k	28 M	83.0%	97.33%
ConvNextV2 tiny	28 M	82.1%	95.21%
ConvNeXt tiny	22 M	82.8%	80.71%
HorNet tiny	25 M	83.4%	78.87%
MogaNet small	21 M	80.1%	56.64%
ConvNextV2 huge 1k_224	660M	86.3%	82.08%
ConvNextV2 huge 22k_384	660 M	**88.7%**	**92.41%**
ConvNextV2 large 1k_224	198 M	84.3%	90.82%
ConvNextV2 large 22k_224	198 M	**86.6%**	**96.63%**

This striking correlation inspired us to build a network where all spatial aggregation was done using parametrized kernels that best resemble what they observed. In figure 2.5 we can see the pattern distribution of ConvNeXt V2 (tiny), as we can see only few patterns observed in figure 2.4 are relevant, therefore we will limit the formulation to only those.

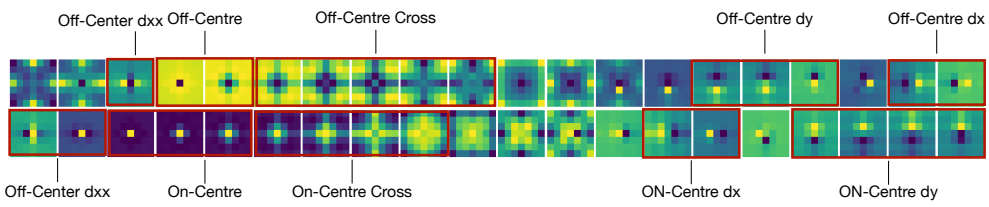


Figure 2.4: Identified kernel patters from [3].

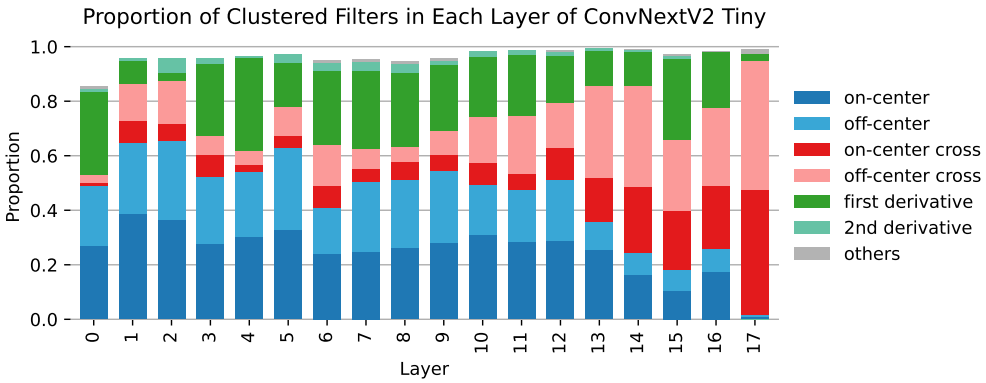


Figure 2.5: Distribution of kernel patters in ConvNeXt V2 Tiny. ImageNet22K pre-training. Source [3].

Chapter 3

Architecture

DoGNeXt is a Convolutional Neural Network (CNN) where each convolutional kernel is sampled from a parametrized Difference of Gaussians (DoG) function and its derivatives. In this chapter, we will explain how the kernels are created and what changes to the ConvNeXt V2 architectures have been made in order to improve the performances when using them.

3.1 DoG Kernels

The DoGConv is the building block of DoGNeXt, in this section we will analyze how it is composed and built. We start by defining the Difference of Gaussians function as follows

$$DoG(\sigma_1, \sigma_2) = \mathcal{N}(0, \sigma_1 I) - \mathcal{N}(0, \sigma_2 I) \tag{3.1}$$

where $\mathcal{N}(0, \sigma I)$ is a bivariate Gaussian function with diagonal covariance matrix

$$\Sigma = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix} = \sigma I$$

As we can see, the mean of both functions is set to 0 since it represents just an offset from the center, and thus it doesn't influence the shape of the

function.

We can rewrite explicitly the function and take the assumption of zero mean and constant diagonal covariance to simplify the expression. Let's start from the generic bivariate Gaussian function

$$f_{\mathcal{N}(\mu,\Sigma)}(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} e^{\left(-\frac{1}{2\left[1-\rho^2\right]}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]\right)} \tag{3.2}$$

where ρ is the correlation between X and Y, that in our case is 0 since the non-diagonal elements of the covariance matrix are 0. Moreover, we have $\sigma_X = \sigma_Y$, therefore we can simplify the expression

$$f_{\mathcal{N}(\mu,\Sigma)}(x,y) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{1}{2}\left[\left(\frac{x-\mu_X}{\sigma}\right)^2 + \left(\frac{y-\mu_Y}{\sigma}\right)^2\right]\right)} \tag{3.3}$$

We observe that $\mu_x = \mu_y = 0$, this gives us the final expression

$$f_{\mathcal{N}(\mu,\Sigma)}(x,y) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)} \tag{3.4}$$

In order to create the kernels we also need the first order derivatives $\left(\frac{\partial}{\partial x} \text{ and } \frac{\partial}{\partial y}\right)$ and second order derivative $\left(\frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2} \text{ and } \frac{\partial^2}{\partial x \partial y}\right)$ of the DoG function (see Figure 3.1 for the 3D plot of the functions).

$$\frac{\partial}{\partial x} = -\frac{x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.5}$$

$$\frac{\partial^2}{\partial x^2} = \frac{x^2}{2\pi\sigma^6} e^{-\frac{x^2+y^2}{2\sigma^2}} - \frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.6}$$

$$\frac{\partial^2}{\partial x \partial y} = \frac{x \cdot y}{2\pi\sigma^6} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.7}$$

It is important to highlight the fact that we do not have to explicitly compute $\frac{\partial}{\partial y}$ and $\frac{\partial^2}{\partial y^2}$, this is because we can just take the derivative along x and transpose the output, this will be explained in more detail later with the kernel building process.

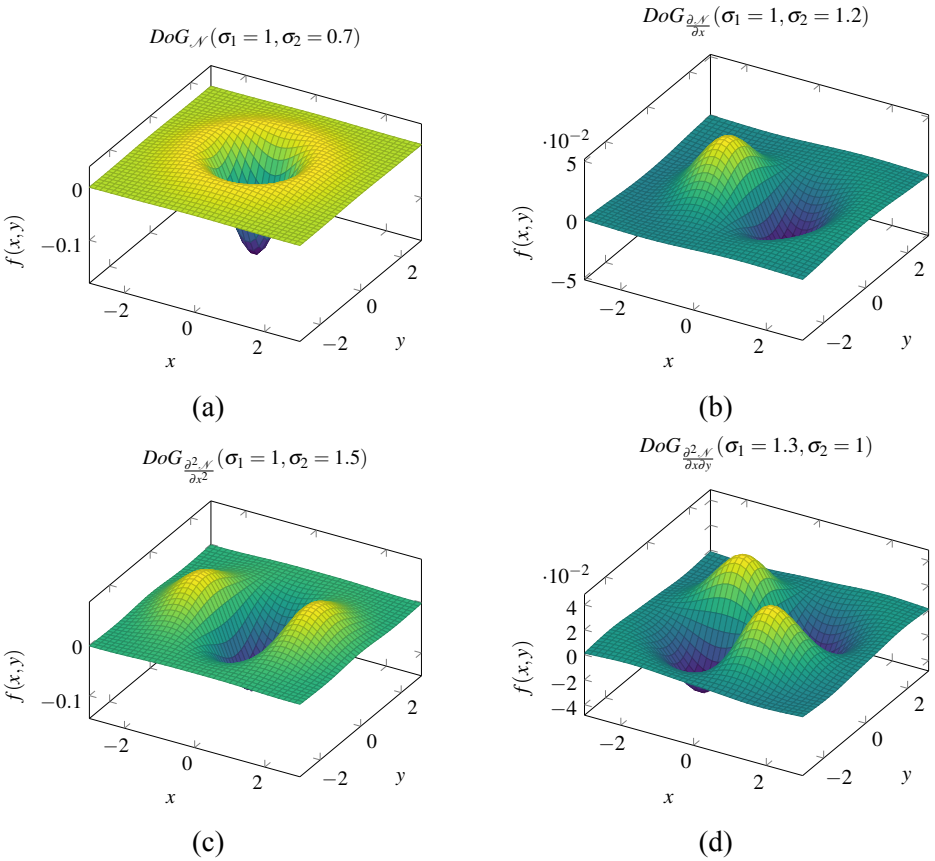
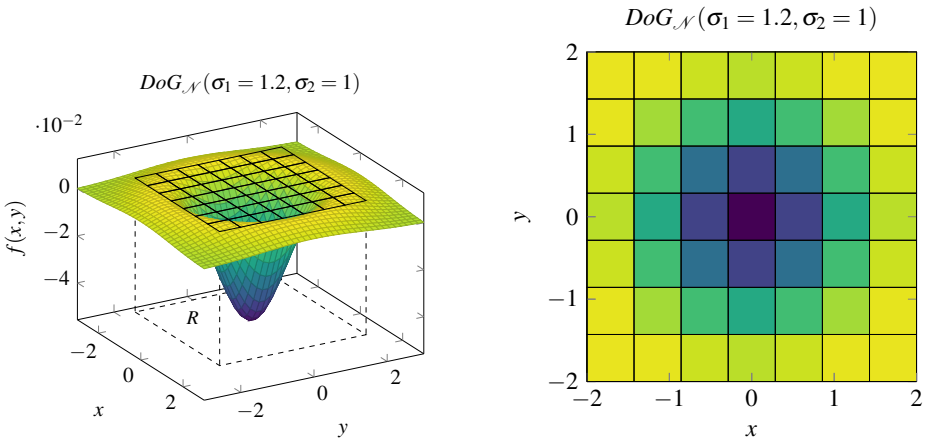


Figure 3.1: 3D plot of the DoG functions used by DoGNeXt.

Now that we have all the functions needed to build the kernel, we need to understand how to perform the sampling in order to move from continuous to discrete space. The simplest solution would be to define a region, discretize it, and finally compute the function on the discrete points (that correspond to the kernel entries). This is a possible solution that lacks a key property, the ability to resize the kernels after the training is complete. This can come handy in some specific experiments and it will be tested in chapter 4. Very intuitively we can think of a 7×7 kernel that represents a fixed function, now imagine we want to increase the dimension to 15×15 or 21×21. We can immediately notice a problem, the more we increase the 'resolution' of the sampling, the more elements are going to get summed by the convolution operator, this is clearly not acceptable since we are technically changing the output distribution of the operation, thus invalidating any training done to the network. Imaging we have a 1×1, the kernel will be computed by sampling the center point of the kernel $(0,0)$ and thus become $f(0,0)$. Now, suppose to transform it to a 3×3 by increasing the sampling rate. This resulting kernel would still compute the function at $(0,0)$ since is the center point of the central entry of the kernel. But the difference this time is that we are also incorporating into the kernel all other surrounding entries, therefore distorting the original.

The solution to this problem involves dividing the function in K^2 different regions of the $xy-$plane, where K is the kernel dimension. This effectively can be seen as overlapping the $xy-$plane of the function with a 2D representation of the kernel. Each entry of the kernel will then hold the integral of the function computed on the corresponding region (Figure 3.2). Doing this will result in a change of the values of the kernels, compare to the naive technique. This is not a problem, in fact, the main objective is to replicate the empirical findings of [3]. This gives us the freedom of using a transformation of the original function, granted it keeps the same patters, which is the case for the integral.

To prove that this sampling methods will give us the desired resizing property, we first need to derive the formula to integrate a function over a given



(a) (b)

Figure 3.2: 3D representation of the kernel sampling process. In figure (a) we can imagine the kernel overlaying the DoG function. This delimits a region R on the $xy-$plane and each entry delimits K^2 subregions. Meanwhile. figure (b) shows the result of the sampling process, where each entry will hold the integral of the function computed over that specific subregion.

interval R

$$\bar{f}_R = \iint_R f(x,y)\,dx\,dy = \int_{a_y}^{b_y}\int_{a_x}^{c_x} f(x,y)\,dx\,dy \tag{3.8}$$

where R is the integration interval (see Figure 3.3). Now, if we take R and split it in 4 different regions $R'_{1\ldots4}$ of the same exact size, we need to prove the following

$$\bar{f}_R = \sum_{1\leq i\leq 4} \bar{f}_{R'_i} \tag{3.9}$$

Proof. Let's start by expanding the formula

$$\int_{a_y}^{b_y}\int_{a_x}^{c_x} f(x,y)\,dx\,dy = \sum_{1\leq i\leq 4}\int_{a_{y_i}}^{b_{y_i}}\int_{a_{x_i}}^{c_{x_i}} f(x,y)\,dx\,dy \tag{3.10}$$

This can be easily proved by using the following property

$$\int_a^b + \int_b^c = \int_a^c \tag{3.11}$$

We notice that by definition the integration limits defined by $R'_{1\ldots4}$ are adjacent (see Figure 3.3). Therefore, we can rewrite the equation and merge the

different terms as follows

$$\int_{a_y}^{b_y}\int_{a_x}^{c_x} = \sum_{1\le i\le 4}\int_{a_{y_i}}^{b_{y_i}}\int_{a_{x_i}}^{c_{x_i}} = \int_{a_{y_1}}^{b_{y_1}}\int_{a_{x_1}}^{c_{x_1}} + \int_{a_{y_1}}^{b_{y_1}}\int_{c_{x_1}}^{c_{x_2}} + \int_{b_{y_1}}^{b_{y_3}}\int_{a_{x_3}}^{c_{x_3}} + \int_{b_{y_1}}^{b_{y_3}}\int_{c_{x_3}}^{c_{x_4}} =$$

$$\int_{a_{y_1}}^{b_{y_1}}\int_{a_{x_1}}^{c_{x_2}} + \int_{b_{y_1}}^{b_{y_3}}\int_{a_{x_3}}^{c_{x_4}} = \int_{a_{y_1}}^{b_{y_1}}\int_{a_{x_1}}^{c_{x_2}} + \int_{b_{y_1}}^{b_{y_3}}\int_{a_{x_1}}^{c_{x_2}} = \int_{a_{y_1}}^{b_{y_3}}\int_{a_{x_1}}^{c_{x_2}} = \int_{a_y}^{b_y}\int_{a_x}^{c_x} \tag{3.12}$$

□

For simplicity R was divided into just 4 subregions, but it is easy to see that the technique we used is generalizable, therefore the proof can be extended to any arbitrary split.

Intuitively, if we keep constant the interval in which we compute the function, we can imagine that this sampling method is just taking the mass of the function and discretizing it. In this way the resolution at which this operation is performed will not interfere with the total mass of the function, therefore producing a kernel that can be resized without the problem of changing the output distribution when applied to a tensor via convolution.

We now provide the formula to compute the average of the DoG and its derivatives

$$\begin{aligned}\bar{f}_{\mathcal{N}(0,\sigma I)}(x,y) &= \int_{a_y}^{b_y}\int_{a_x}^{c_x}\frac{1}{2\pi\sigma^2}e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}\,dx\,dy = \\ &= \frac{1}{4}\left(\operatorname{erf}\left(\frac{b_y}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{a_y}{\sqrt{2}\sigma}\right)\right)\cdot\left(\operatorname{erf}\left(\frac{c_x}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{a_x}{\sqrt{2}\sigma}\right)\right)\end{aligned} \tag{3.13}$$

$$\begin{aligned}\frac{\partial}{\partial x}\bar{f}_{\mathcal{N}(0,\sigma I)}(x,y) &= \int_{a_y}^{b_y}\int_{a_x}^{c_x}-\frac{x}{2\pi\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}\,dx\,dy \\ &= \frac{\sqrt{2\pi}}{4\pi\sigma}\left(e^{-\frac{c_x^2}{2\sigma^2}}-e^{-\frac{a_x^2}{2\sigma^2}}\right)\cdot\left(\operatorname{erf}\left(\frac{b_y}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{a_y}{\sqrt{2}\sigma}\right)\right)\end{aligned} \tag{3.14}$$

$$\begin{aligned}\frac{\partial^2}{\partial x^2}\bar{f}_{\mathcal{N}(0,\sigma I)}(x,y) &= \int_{a_y}^{b_y}\int_{a_x}^{c_x}\frac{x^2}{2\pi\sigma^6}e^{-\frac{x^2+y^2}{2\sigma^2}} - \frac{1}{2\pi\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}\,dx\,dy \\ &= \frac{\sqrt{2\pi}}{4\pi\sigma^5}\left(\frac{a_x}{2}e^{-\frac{a_x^2}{2\sigma^2}} - \frac{c_x}{2}e^{-\frac{c_x^2}{2\sigma^2}} + \left(\operatorname{erf}\left(\frac{c_x}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{a_x}{\sqrt{2}\sigma}\right)\right)\right)\cdot \\ &\cdot\left(\operatorname{erf}\left(\frac{b_y}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{a_y}{\sqrt{2}\sigma}\right)\right) - \frac{1}{\sigma^2}\bar{f}_{\mathcal{N}(0,\sigma I)}(x,y)\end{aligned} \tag{3.15}$$

$$\begin{aligned}\frac{\partial^2}{\partial x\partial y}\bar{f}_{\mathcal{N}(0,\sigma I)}(x,y) &= \frac{1}{A}\int_{a_y}^{b_y}\int_{a_x}^{c_x}\frac{x\cdot y}{2\pi\sigma^6}e^{-\frac{x^2+y^2}{2\sigma^2}}\,dx\,dy = \\ &= \frac{1}{2\pi\sigma^4}\left(e^{-\frac{a_y^2}{2\sigma^2}} - e^{-\frac{b_y^2}{2\sigma^2}}\right)\cdot\left(e^{-\frac{a_x^2}{2\sigma^2}} - e^{-\frac{c_x^2}{2\sigma^2}}\right)\end{aligned} \tag{3.16}$$

where $\operatorname{erf}(\cdot)$ is the error function defined as

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\,dt \tag{3.17}$$

this is built in all major deep learning frameworks, therefore we do not care about the implementation of it.

Using the definition of all the function needed to build our DoG kernels we can give the complete formulation that the network will optimize during training. In this formulation, the *y* derivatives are not computed directly because our output is a matrix that can be transposed to simulate a 'swap' between the *x* and *y* axes; therefore, we can use the transposed *x* derivatives. First we define the kernel using just second order derivative DoG

$$\begin{aligned}&\mathbf{K}''_{\sigma_1,\sigma_2,\mathbf{p}_{d2}} = \operatorname{softmax}_{T=0.01}(\mathbf{p}_{d2})\cdot \\ &\cdot\left[DoG_{\frac{\partial^2\mathcal{N}}{\partial x^2}}(\sigma_1,\sigma_2), DoG_{\frac{\partial^2\mathcal{N}}{\partial x^2}}(\sigma_1,\sigma_2)^T, DoG_{\frac{\partial^2\mathcal{N}}{\partial x\partial y}}(\sigma_1,\sigma_2)\right]^T\end{aligned} \tag{3.18}$$

We define softmax_T as

$$\text{softmax}_{T=\tau}^{(i)}(\mathbf{x}) = \frac{e^{\frac{x_i}{\tau}}}{\sum_{j=1}^{l} e^{\frac{x_j}{\tau}}} \tag{3.19}$$

where $\mathbf{x} \in \mathbb{R}^l$ and τ is the temperature. Notice that in our formulation the temperature value is close to zero, this produces an output distribution with lower entropy compared to the 'vanilla' softmax where $T = 1$, therefore, it acts more like a hard selection. This, in combination with $\mathbf{p}_{d2} \in \mathbb{R}^3$, which is optimized by the network, controls which of the function is dominant in the kernel.

The notation $DoG_{\frac{\partial^2 \mathscr{N}}{\partial x^2}}(\sigma_1, \sigma_2)$ indicates that the base function used to compute the difference of Gaussians is the second-order derivative $\left(\frac{\partial^2 \mathscr{N}}{\partial x^2}\right)$. Intuitively, this creates a mix of DoGs thanks to $\mathbf{p}_{d2}$. The same approach is applied also to the first order derivatives as follows

$$\mathbf{K}'_{\sigma_1,\sigma_2,\mathbf{p}_{d1}} = \sigma\left(\frac{\mathbf{p}_{d1}}{0.01}\right) \cdot \left[DoG_{\frac{\partial \mathscr{N}}{\partial x}}(\sigma_1, \sigma_2), DoG_{\frac{\partial \mathscr{N}}{\partial x}}(\sigma_1, \sigma_2)^T\right]^T \tag{3.20}$$

where $\mathbf{p}_{d1} \in \mathbb{R}^2$ and has the same purpose as $\mathbf{p}_{d2}$.

The final kernel is than given by mixing again all the functions

$$\begin{aligned}\mathbf{K}_{\sigma_1,\sigma_2,\mathbf{p_d},\mathbf{p_{d1}},\mathbf{p_{d2}},\rho,b} = \text{softmax}\left(\frac{\mathbf{p_d}}{0.01}\right) \cdot \\ \cdot \left[DoG_{\mathscr{N}}(\sigma_1, \sigma_2), \mathbf{K}'_{\sigma_1,\sigma_2,\mathbf{p}_{d1}}, \mathbf{K}''_{\sigma_1,\sigma_2,\mathbf{p}_{d2}}\right]^T \cdot e^{\rho} + b\end{aligned} \tag{3.21}$$

where b is the bias, ρ is a scaling term and $\mathbf{p}_d \in \mathbb{R}^3$ is the mixing parameter. Note that ρ is used as exponent; this is done for the sole purpose of having a better numerical range of the parameter during optimization and to avoid negative or zero values that could destabilize the training process.

Lastly, we need to address the integration intervals to use when computing the DoG functions. Until now all the formulations used a_x, a_y, b_y, c_x as extremes, but we need real values to compute the kernels. In fact, all of them

are fixed when deciding what region of the $xy-$plane we want to represent in the kernels. We empirically define this space as $R = [-2,2]$ to be a good enough region to display good diversity in the patterns, we then define the dimension of the kernel $k = 7$, therefore, we can divide in 7×7 squared regions R obtaining the integration intervals (see Figure 3.3).

This completes the definition of the DoG based convolutions, that from now on are going to be called DoGConv.

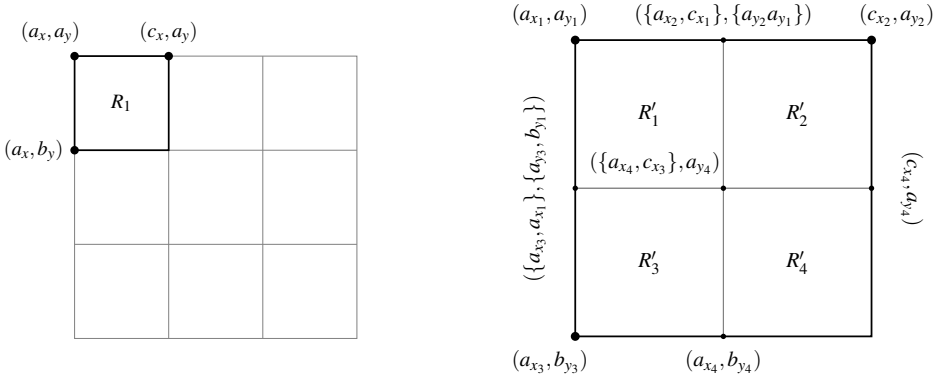


Figure 3.3: Panel (a) shows a 3×3 kernel and the relative integration extremes of the top left region R_1. Panel (b) meanwhile takes R_1 and split it into four subregions $R'_{1\ldots4}$ each with the relative extremes. From (b) we can notice how many of the coordinates overlap, for example while considering the x axis we observe that $a_{x_1} = a_{x_3}$, $a_{x_2} = c_{x_1} = c_{x_3} = a_{x_4}$ and that $c_{x_2} = c_{x_4}$. The same reasoning can be applied to y axis.

3.2 DoGNeXt and Downsample Blocks

With the definition of the DoGConv operator we can analyze the differences between ConvNeXt V2[33] and DoGNeXt, but first we need to establish some design criteria. First, we want all spatial aggregation of the network to be done exclusively with DoGConv; the reason behind this is having the option to rescale the kernels with the objective of modifying the scale of at which the network performs the best, this will be used in chapter 4. Second, we want to keep approximately the same amount of FLOPs of the original network; doing

this will ensure a fair comparison between the different architectures.

Let's start by analyzing the differences between the stem layer and downsampling blocks of the two architectures. In Figure 3.4a we can see the first problem with ConvNeXt, the stem layer is a 4×4 classic convolution. This is problematic for several reasons: first, our DoG-based kernels are depth-wise only, therefore cannot be used to replace it; second, the size of the kernel is too small to have a meaningful representation of the underlying function if we were to use a DoGConv. The solution is unconventional but it solves the two issues, we can simply reduce the stem layer to a 1×1 convolution (Figure 3.4b). The idea is that we keep the same amount of FLOPs, this is because the stem layer of ConvNeXt has a stride of 4, therefore it processes the same spatial location only once, making it almost equivalent to ours (we are missing a summation that is not making a meaningful impacting on the overall FLOPs count). Moreover, we avoid the spatial aggregation without DoGConv, which is one of our design principles.

However, this approach introduces a problem, the activation after the stem layer is not correctly strided. The solution that DoGNeXt uses is to add stride to the first block of the network, doing this ensures that the convolution receives in input a more dense input compared to pooling techniques. But as we can see in Figure 3.5a, the ConvNeXt block has a residual connection, as it is usual to have in modern architectures, this creates a problem similar to the one seen in ResNet[11]. When a block has stride greater than 1 the shape of the input and output activation do not match, this makes impossible to apply the residual connection unless the input tensor is downsampled. Several solutions have been explored with ResNet, like 1×1 convolution with stride 2, removing the residual connection or using an average pool operator followed by a 1×1. DoGNeXt uses just an $s \times s$ average pool, where s is the stride of the block (Figure 3.5c).

The downsampling block is very similar to the stem layer, with the difference that the stride is 2 instead of 4. Note that while the first block has stride

4 to compensate the missing stride in the stem, all the blocks after the donwsample blocks have stride 2 (Figure 3.4d). The reason behind keeping this block even if is not effectively donwsampling is that it increases the number of channels and it regularizes the network, thanks to the LayerNorm[2].

In Figure 3.5a we can see the structure of the main block that composes the ConvNeXt V2 architecture. The design principle comes from the transformer block, with an MLP with expansion ratio of 4, but with the substitution of the attention with convolutions. The main innovation proposed in the V2 revision compared to the original ConvNeXt[22] is the addition of the GRN (Global Response Normalization) layer that aims at promote feature diversity in between the two point-wise convolutions, refer to the original work for a more in depth explanation. DoGNeXt build upon this and retains the same exact structure, with the substitution of the classic convolutions with DoGConvs.

It is important to highlight some differences between the two architecture from a computational cost perspective. Following our design principle we were able to keep approximately the same amount of FLOPs as the original network. It has to be noted that the computational cost of the DoGConv is slightly different from a normal convolution. This is because we need to consider that the kernel needs to be sampled after each parameter update, this introduces a bit of overhead to the forward pass during training. However, note that it is completely independent of the batch size and image dimension, therefore practically negligible. Additionally, there is a slight parameter reduction with DoGConvs compared to normal convolutions, but since the parameter budget is completely allocated to the MLPs, the reduction is also negligible. Lastly, using stride has a drawback that could hurt performance when training on memory constrained settings. The stem layer in ConvNeXt V2 has the job of both increasing the number of channels while reducing the spatial dimension, this results in an output activation with shape $\frac{H}{4} \times \frac{W}{4} \times C$. DoGNeXt does not reduce the spatial dimension, producing an activation that is 16 times larger after the stem layer, increasing the memory consumption.

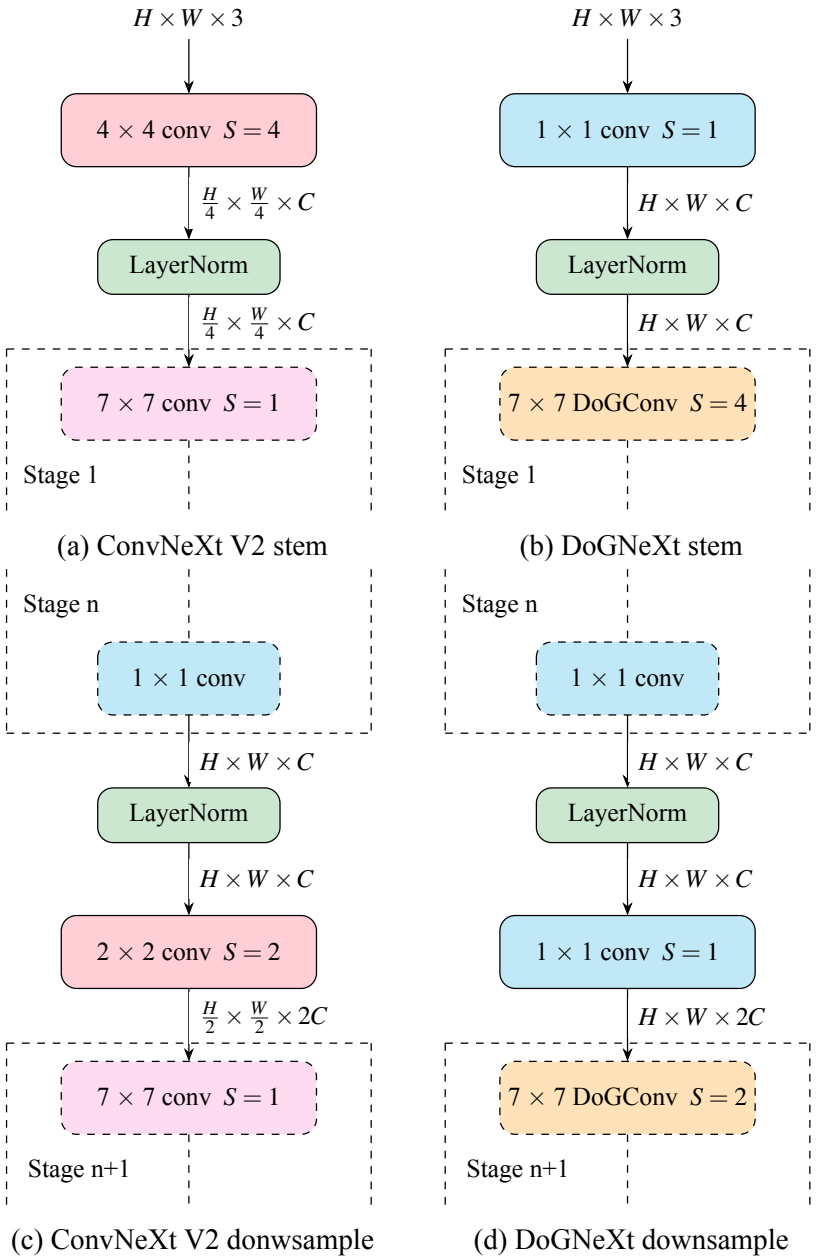
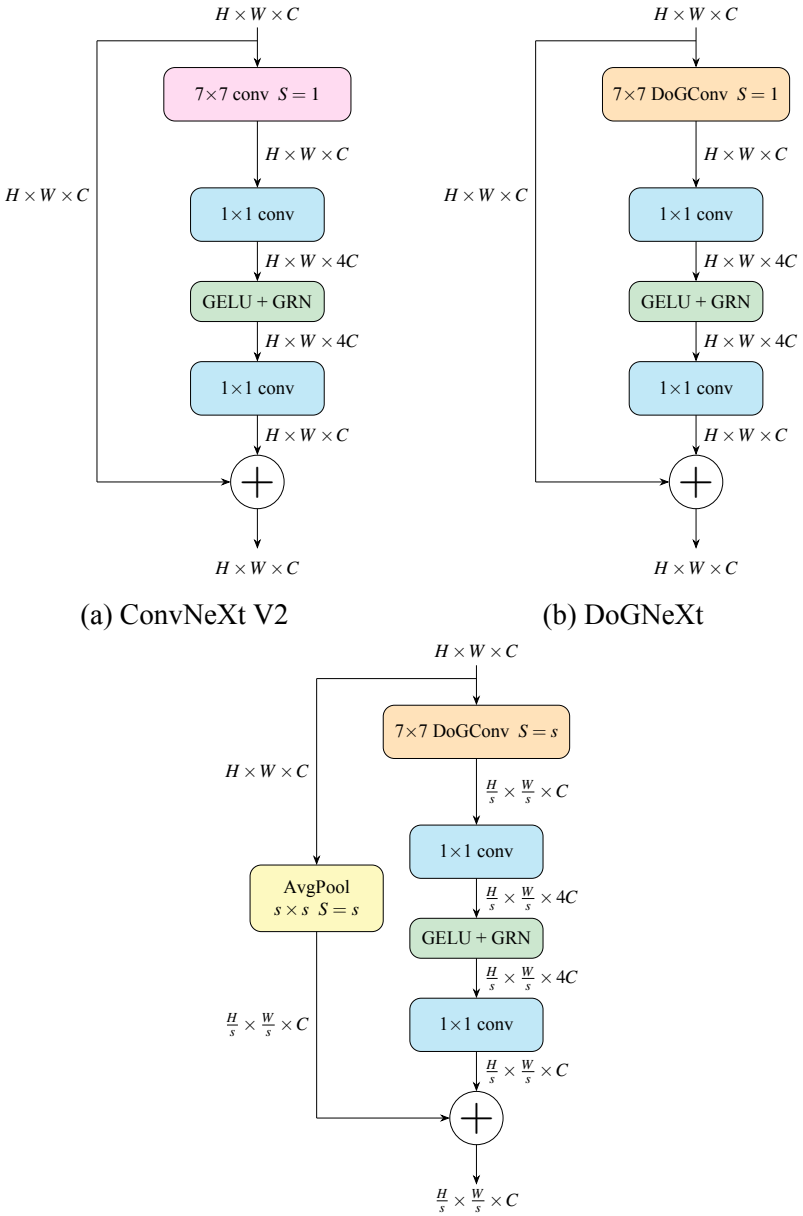


Figure 3.4: ConvNeXt V2 and DoGNeXt stem and donwsample blocks. From (a) and (b) we can see how DoGNeXt differs since it uses a simple 1×1 convolution instead of the 4×4 of ConvNeXt V2 meanwhile adding stride to the subsequent block. From (c) and (d) we can see a similar approach to the previous example.



(a) ConvNeXt V2

(b) DoGNeXt

(c) DoGNeXt w/stride

Figure 3.5: ConvNeXt V2 and DoGNeXt blocks. As we can see from (a) and (b) the only difference between ConvNeXt V2 and DoGNeXt is the use of DoGConvs in the latter. In (c) we can see the solution used by DoGNeXt to use stride in conjunction with residual connections.

Chapter 4

Experiments

Now that we have a complete overview of the DoGNeXt architecture we can move to the experiments conducted using the network. We present some ablations on the network architecture itself to assess the validity of our design choice and to discard some possible alternatives. We then move on some more specific benchmark, medical images, to assess how DoGNeXt performs in other types of image domains. Lastly, by using the property of resizing kernels, we are going to experiment with scale robustness, and we are going to see how we can reduce the number of FLOPs of the model by changing the total stride of the network.

4.1 Experimental setup

DoGNeXt takes the work of ConvNeXt V2 and builds upon it as they share the same underlying network structure. The codebase is developed in PyTorch [1] and for most of the experiments we used 2×L40 NVIDIA GPUs as our accelerators.

The training details for ImageNet1K of DoGNeXt can be seen in Table 4.1, unless specified we used the same for all our experiments. ConvNeXt V2 is a family of different sized models, due to our limited training resources we

took as reference the smallest, named *atto*, moreover, we reduced the training epochs from 600 to 50. ConvNeXt V2 atto did not use MixUp[38] or CutMix[37] due to the network capacity being particularly low, therefore we kept this also in the training of DoGNeXt. One notable difference from classic CNNs is the absence of model Exponential Moving Average (EMA)[13], this was forced since the EMA checkpoint of DoGNeXt never converged for some unknown reasons. Our best hypothesis is that making an EMA of the parameters of the DoGConv doesn't result in a meaningful set of kernels. We also avoided to use weight decay for the DoGConv parameters as they are quite sensitive to changes and they do not have overfitting problems; this is because the capacity is significantly lower than a classic convolution. Lastly, ConvNeXt uses layer-wise learning rate decay[4][7], this is a technique that reduces the learning rate based on the layer depth (starting from the last layer) with the theoretical objective of reducing the magnitude of updates in the first layers to help convergence. We found that it was reducing the convergence speed for DoGNeXt, therefore we kept a constant learning rate throughout the network. In Table 4.1 we specify the 'Base LR' that is not the true learning rate used by the network but is the learning rate tuned for a batch size of 256. To get the true value we multiply the base learning rate with the ratio between the new batch size and 256, in our case the result would be $\frac{1024}{256} \cdot (2 \cdot 10^{-4}) = 4 \cdot (2 \cdot 10^{-4}) = 8 \cdot 10^{-4}$.

4.2 Network ablations

The objective of the ablation on the network architecture is to show the performance of different configurations, validating the design choice explained in Chapter 3. In table 4.2 we can see the results of different DoGNeXt and ConvNeXt V2 variation. We refer to 3 different versions of DoGConv: base, $d1$ and $d2$. These are networks where we keep the same exact architecture of ConvNeXt V2 but we substitute the kernel with DoGConv. The *base* uses just

Table 4.1: DoGNeXt Hyper-Parameters for ImageNet1K training.

Training hyper-parameter	Value
Optimizer	AdamW
Base LR	$2 \cdot 10^{-4}$
Weight Decay	0.3
Optimizer momentum	$\beta_1 = 0.9,\ \beta_2 = 0.999$
Batch size	1024
LR Schedule	Cosine Decay
Warmup epochs	0
Training epochs	50
Label smoothing	0.2
Drop path	0.1

the base DoG function, the $d1$ and $d2$ use first and second order derivatives respectively. *Stride downsample* is the true DoGNeXt as outlined previously, it uses stride to downsample instead of 4×4 and 2×2 classic convolutions.

We then tried to make small changes to improve further the architecture with poor results. DoGNeXt uses a kernel dimension of 7, this derives from an extensive ablation done in the original ConvNeXt[22] paper. We tried to enlarge the kernel to assess the validity of that ablation also with DoGConv finding that in fact 7 is the best kernel dimension. Note that enlarging the kernel from 7 to 9 or 11 implies a careful consideration on the initialization and scale. In fact, DoGNeXt uses a careful initialization that will be explained in section 4.2.1. If we increase the sampling rate of the function, we incur a side effect. In fact, we would like to keep constant the ratio between size of the patterns displayed in the DoGConv and image resolution. If we were to just increase the kernel dimension the formulation will create a higher resolution version of the kernel, therefore enlarging the pattern relative to the image, an effect that will be used to control the optimal scale of detections in the next sections. To avoid this problem we enlarge the integration limits of the function instead, so that the inner 7×7 square of the kernel is kept equal regardless of the kernel dimension. We can imagine this as taking the original kernels and adding one or two outer rings around each kernel. Even with this extra care

Table 4.2: Network ablations results for ImageNet1K.

Model	Accuracy@1	Accuracy@5
ConvNeXt V2 atto	**70.87**	**89.97**
DoGConv (base)	68.40	88.53
DoGConv (d1)	70.22	89.73
DoGConv (d2)	69.56	89.26
Stride downsample (d1)	70.15	89.63
Stride downsample (d2) - DoGNeXt	**70.59**	89.76
Kernel size 9	70.43	**89.79**
Kernel size 11	70.31	89.68
Double Residual	69.80	89.29
No GRN	68.83	88.50
ConvNeXt V2 atto (600 epochs)	*76.2*	-

we see degrading performance the larger the kernel. Future work could assess the relationship between kernel dimension and image resolution, theoretically images with large resolutions could benefit from larger kernels without the convergence problems that classic convolution may have, as shown in [3].

In Chapter 3 we have seen how the residual connection get in the way of strided convolutions in some blocks. Taking inspiration from the transformer block we tried to create a double residual connection, one that would skip the convolutional part and one for the MLP. The results are underwhelming, therefore this solution was not used.

Finally, ConvNeXt V2 introduces the Group Response Normalization (GRN) layer. We assessed that it is essential in order to get good performances out of the network.

Note that all the networks have been trained with just 50 epochs, meanwhile the original training recipe uses 600 epochs. We included the results as reported in the ConvNeXt V2 paper[33] for the *atto* variant (without pretraining). We can see that training for longer gives around 5.3 points of accuracy.

4.2.1 DoGConv initialization

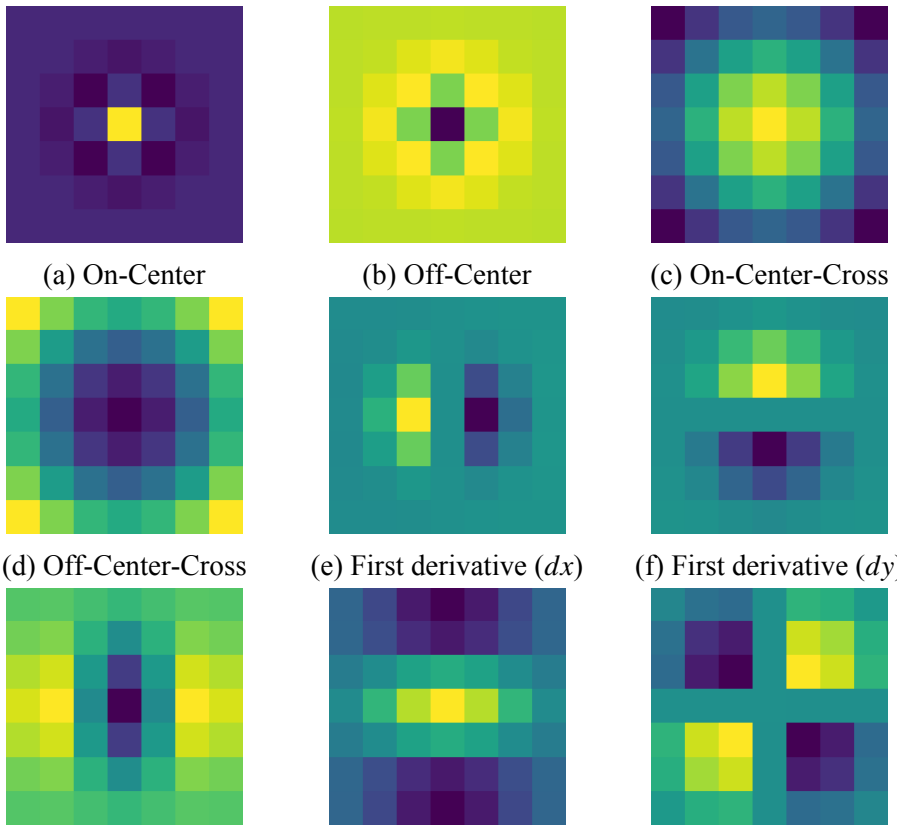
Until now, we did not mention how the parameters of the DoGConv are initialized. This is a very important topic because we need a careful consideration due to their unique usage in the network. In Figure 4.1 we can see the different patterns that a DoGConv can exhibit. We define a range of values for $\sigma_{1,2}$ that produce each patter:

- *On/Off Center*: $\sigma_1, \sigma_2 \in [-1.1, -0.7]$
- *On Center Cross*: $\sigma_1 \in [0, 0.8],\ \sigma_2 \in [1, 1.1]$
- *First derivative*: $\sigma_1, \sigma_2 \in [-1, 1]$
- *Second derivative*: $\sigma_1, \sigma_2 \in [-0.5, 0.5]$

All the values have been chosen empirically in order to have good variety in kernel patterns. Note that, in order to produce the *Off Center Cross*, we swap the two sigma of the *On* variant described above, this results in a sign flip of the function.

In addition to $\sigma_{1,2}$, we have all the parameters that determine the function mix, these are used to let just a single function emerge while suppressing the others. To initialize this, we decided to give equal space for each patter, prioritizing the base function. More precisely: *On/Off Center* and *On/Off Center Cross* have allocated 25% of the kernels each (per DoGConv module); *First derivative* have 25%, comprised of a $12.5\% + 12.5\%$ split between dx and dy; the remaining 25% is used by *Second derivative* in a $10\% + 10\% + 5\%$ split between dx^2, dy^2 and dxy. This was determined empirically with some quick experiments on the different proportions.

To summarize the initialization process, firstly, we determine which kernels need to exhibit which pattern and therefore we fix the mixing parameters, lastly we draw at random the values for $\sigma_{1,2}$ from the respective ranges based on the pattern.



(a) On-Center (b) Off-Center (c) On-Center-Cross

(d) Off-Center-Cross (e) First derivative (dx) (f) First derivative (dy)

(g) Second derivative (dx^2) (h) Second derivative (dy^2) (i) Second derivative (dxy)

Figure 4.1: DoGConv kernel patterns

Table 4.3: Comparison of initialization strategies (ImageNet1K performance).

Init Strategy	Accuracy @1	Accuracy @5
Random	67.65	87.85
Custom	70.59 (+2.94)	89.76 (+1.91)

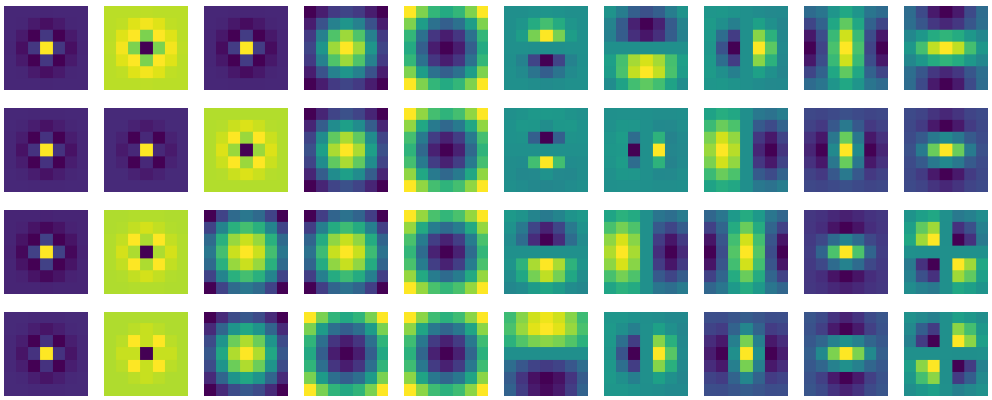
In Table 4.3 we provide a quantitative measure of the impact of our initialization. We put it against a strategy that uses a truncated normal distribution with $\sigma = 0.02$. This is the same type of initialization that ConvNeXt V2 uses for the convolution's and linear layer's weights. As we can see the degradation in performance is significant, validating our technique.

In Figure 4.2 and Figure 4.3 we can see all the kernels of the first block of DoGNeXt before training and after training. Note that the colors in the image are scaled on a kernel base, this is done to have a consistent coloring and avoid deleting details due to very diverse value range between kernels. We can use these images to assess the patterns but not the values of the kernels. From Figure 4.2 it is clear that the difference between the initialized (before training) and after training are very subtle, with some kernels changes and minor shape differences. This unexpected behavior might have different causes. We can argue that the initialization is good enough that the optimizer is not pushed to substantially change the kernels, but on the other hand, we can suppose that such strong initialization is inducing the network to reach a local minima that could be avoided with a different one. These are just suppositions, future work might give more plausible reasons with a more in depth analysis.

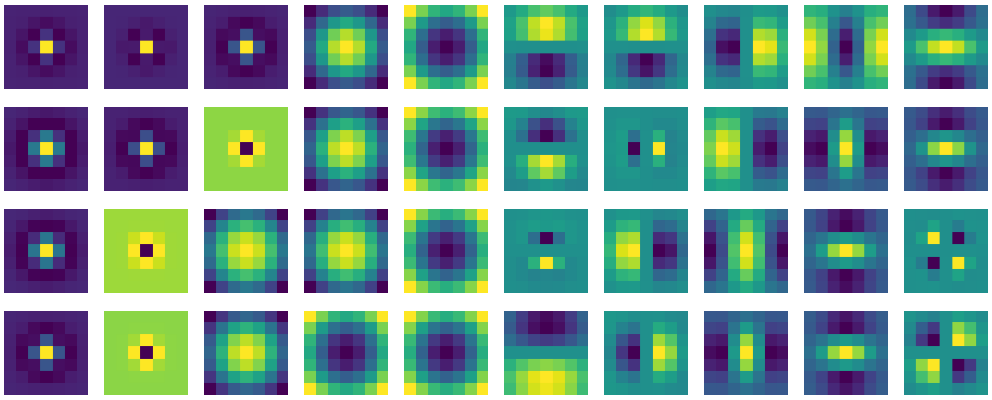
4.2.2 Performance with data scarcity

We showed how DoGNeXt kernels do not require a major tuning from the training to be meaningful, this can result in better performance when data is scarce. In Table 4.4 we can see the results from DoGNeXt and ConvNeXt V2 atto on different versions of ImageNet1K, namely:

- Full ImageNet1K. This is the full version of the dataset trained for a

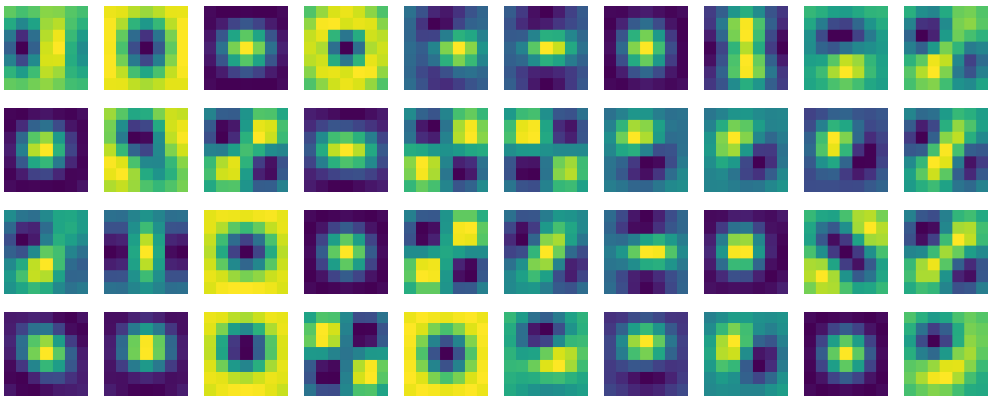


(a) Custom Initialized Kernels

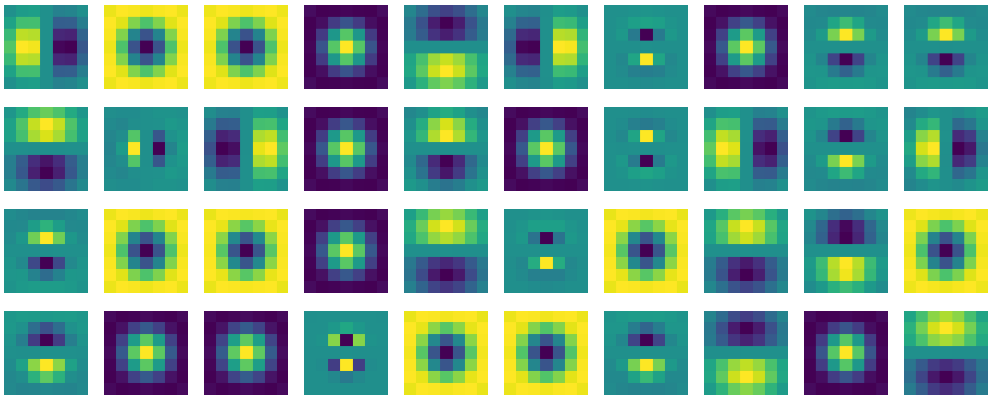


(b) Trained Kernels

Figure 4.2: DoGNeXt first block kernels before and after training using the custom initialization.



(a) Random Initialized Kernels



(b) Trained Kernels

Figure 4.3: DoGNeXt first block kernels before and after training using the random initialization.

Table 4.4: ConvNeXt V2 and DoGNeXt performance on different data regimes.

Model	ImageNet1K		ImageNet1K (200)		ImageNet50	
	Acc@1	Acc@5	Acc@1	Acc@5	Acc@1	Acc@5
ConvNeXt V2	**70.87**	**89.97**	59.56	81.27	81.26	95.71
DoGNeXt	70.59	89.76	**60.02**	**81.72**	**83.58**	**96.49**

limited number of epochs due to the computational cost. Recall that the dataset consists of 1000 classes, each with 1300 images, for a total of approximately 1.3 million images.

- Small ImageNet1K (200). This is a cut down version of the dataset that uses just 200 images per class, for a total of 200K images. To compensate the small amount of data the number of epochs has been increased to 200 epochs.

- ImageNet50. This is a random set of just 50 classes of ImageNet1K. This is the smallest dataset of the three with a total image count of around 65K. To further compensate the limited number of data we increased the number of epochs to 600.

Our hypothesis is confirmed by the result in Table 4.4. In fact, we see how DoGNeXt is able to perform better than ConvNeXt V2 when we reduce the amount of data. When training on the full dataset we are not able to replicate or surpass ConvNeXt V2, this could be a ground for future work to explore with some changes to the architecture or formulation of the DoGConv. Overall the results are remarkable when we consider that our kernels are extremely constrained and that they work as a drop-in replacement almost without any tuning.

4.3 MedMNIST benchmark

In section 2.1.2 we described the MedMNIST dataset and here we are presenting the results for ConvNeXt V2 and DoGNeXt. Recall from Table 2.1 that each sub-dataset in MedMNIST has a limited number of images compared to the size of ImageNet1K, this is a unique situation where data efficiency is key. Moreover, the distribution of the images is quite different compared to ImageNet, and it will be interesting to assess if the DoG kernels are good also in this setting.

Table 4.5 shows the accuracy of the various models on each task. We included the average accuracy as a single comparison metric, this will summarize the performance on the entire benchmark. The original MedMNIST paper [36] provides the results for ResNet-18 and ResNet-50. We included those with an asterisk in the table as well, even though they have 11.7M parameters with 1.8G FLOPs and 25.6M parameters with 4.1G FLOPs respectively, which is considerably more than the 3.7M and 3.4M parameters with 550M FLOPs of ConvNeXt V2 and DoGNeXt. In addition to this, we provide the results with fine-tuned models that we pre-trained on ImageNet1K for 50 epochs. Moreover, as a reference, we include a ConvNeXt V2 pre-trained for 600 epochs, that is publicly available from the original authors, to show what performance could be archived with an extensive pre-training.

We trained all the models for 100 epochs and batch size of 128, all other training details are equal to the recipe for ImageNet1K of Table 4.1, with the exception of the data augmentation. In fact, we use just a random horizontal flip to avoid disrupting the images and therefore the ability to classify them. Usually fine-tuning doesn't require an extensive training of this many epochs, but we wanted to picture the best possible scenario for each model, without the risk of under-training and without the problem of tuning the recipe for each separate task.

When training from scratch, DoGNeXt performs remarkably well, beating

Table 4.5: Test Accuracy on the MedMNIST benchmark.

Model	Path	Chest	Derma	OCT	Pneumonia	Retina	Breast	Blood	Tissue	OrganA	OrganC	OrganS	*Average*
ConvNeXt V2	88.83	94.78	76.77	84.80	83.97	55.25	82.05	96.61	68.10	92.99	88.97	72.58	82.14
DoGNeXt	89.29	94.81	76.67	86.90	85.41	54.00	82.05	98.45	72.72	94.42	91.74	79.42	83.82
ResNet-18*	90.9	94.7	75.4	76.3	86.4	49.3	83.3	96.3	68.1	95.1	92.0	77.8	82.13
ResNet-50*	89.2	94.8	73.1	77.6	88.4	51.1	84.2	95.0	68.0	94.7	91.1	78.5	82.14
ConvNeXt V2 (pretranied)	95.33	**94.83**	87.18	89.00	**93.10**	**65.50**	**89.74**	**99.15**	75.14	97.12	**95.09**	83.17	88.69
DoGNeXt (pretrained)	**95.58**	**94.83**	**88.88**	**91.50**	90.86	64.75	88.46	98.71	**75.42**	**97.22**	95.02	**83.64**	**88.73**
ConvNeXt V2 (pretrained 600)	*96.74*	*94.85*	*88.63*	*90.30*	*90.70*	*66.75*	*90.38*	*99.09*	*75.40*	*97.38*	*95.47*	*84.05*	*89.14*

ConvNeXt V2 on the majority of the tasks. On average, it is able to score better than all other models, showing the benefit of having the DoGConv. When fine-tuning from a pre-trained checkpoint the performance increases substantially. In this regime ConvNeXt V2 is able to perform as good as DoGNeXt and in some cases better, the reason is plausibly that the bias that the DoGConv provides is less impactful in this instance due to the pre-training.

4.4 Scale Adaptation and Augmentation

DoGNeXt has the property of resizing the kernels at inference thanks to the formulation of the DoGConv. In this section we want to assess if it is feasible to change the scale at which the accuracy is best via a kernel resizing.

Let's start by outlining the situation with a default train on ImageNet. In Table 4.6 we can see how decreasing the evaluation resolution, therefore reducing the scale of the images, results in a sharp drop in accuracy for both DoGNeXt and ConvNeXt V2. Our objective is improving them by shrinking the kernels of each DoGConv. Note that in this section we train DoGNeXt with a kernel size of 9 instead of 7, this is because we need bigger kernels to avoid a collapse when we will shrink them.

In Table 4.7 we have the performance of DoGNeXt when evaluating at kernels size 9, 7 and 5. As we can see the results are not good unfortunately. The reason for this decay in performance is not completely clear, but we can suppose that it is due to some distribution shift on the activation of the kernels, caused by the discretization of the function, in particular, kernel 5 is completely collapsed. We don't have a clear solution to this other than to train the

network with the different kernel dimensions. This is not the solution we were hoping for but it is the best compromise. Many recipes have been tested, we will just show the best in order to avoid presenting useless information.

Since we need to modify the training process we should include a comparison with a scale augmented network. This is done in order to assess if DoGNeXt exhibit some additional benefits thanks to the DoGConv or not. The training recipe for ImageNet1K already includes a scale augmentation, called `RandomResizedCropAndInterpolation`, included in the Python library `timm`. ConvNeXt V2, and many other CNNs trained on ImageNet, uses it with parameters `scale=(0.08, 1.0)` and `ratio=(0.75, 1.3333)`. The `scale` argument defines the scale of the crop on the original image, for example, if we have scale 0.5 and an image with dimension 1000×500, it will take a random crop of size 500×250. Meanwhile, `ratio` defines the aspect ratio of the crop, that then will be resized to a $1:1$ for the network, producing a vertical or horizontal stretch of the crop. From this we can understand the source why the performance on smaller resolutions is bad, in fact, the scale is at maximum 1.0, this means that, during training, the images are never downsampled but we always zoom-in a bit. This is also the main cause of the FixRes effect [32]. We define a different form of scale augmentation that does not interfere with the one already implemented. There are multiple factors in play behind this decision. First, if we want to resize the kernels when the resolution is in a certain range we need the entire batch to have the same exact resolution. Second, we want to have a better control over the distribution of the augmentation, without the limitation of a random scale drawn from a uniform distribution. The solution we propose is to place a donwsampling operation before the forward pass. This will reduce the resolution of the input batch with a scale factor drawn at random in the range $s \in [0.35, 0.9]$ with a probability $p = 0.2$ of activating.

In Table 4.8 we find the results of ConvNeXt V2 and DoGNeXt trained with this technique. Both network improved massively on small images with

gains of more than 25 points of accuracy when evaluating 64 px $\times$ 64 px images. Both network did not suffer from a significant loss in performance, with ConvNeXt V2 performing better across the board. This is extremely unexpected since we are focusing on smaller resolution, but we observe better performances also on $2\times$ the training resolution. Future work could focus on this augmentation with more extensive testing, including different architectures, evaluating the viability and impact for object detectors.

In Table 4.9 we find the results for DoGNeXt when training with different kernels. We train the network with kernel size 9, 7, 5 and for each we define a scale range to use. Kernel 9 is the base and it will be used every time we don't change the scale, therefore $s = 1.0$. If $s \in [0.6, 0.9]$ we use kernel 7, meanwhile, if $s \in [0.35, 0.6]$ we will use kernel 5. These ranges have been chosen in order to keep the ratio between kernel size and resolution constant. The results are marginally better when compared to the custom scale augmentation we defined, but a clear improvement over a normal train recipe. In Table 4.10 we trained DoGNeXt with a hybrid approach. In fact, we use the train with multiple kernel dimensions but with a probability of 0.5 of using kernel 9 even if the $s < 1.0$, thus performing half custom scale augmentation and half multi-kernel. In this case we see better results over the baseline compared to the previous approach, especially in lower resolution the gain is significant and the loss in larger ones is marginal. Kernel 5 did not manage to get better results than kernel 7 even in very small resolution. We can suppose that the additional scale augmentation was more beneficial to the latter, producing a big improvement.

In summary, DoGNeXt is able to marginally surpass the remarkable results we can obtain with the custom scale augmentation we implemented. Unfortunately, the DoGNeXt approach has the crucial problem of having to define resolution range for each kernel, this is something that could be done empirically but is an added complication.

Table 4.6: Accuracy@1 on ImageNet1K at different resolutions. Vanilla training.

Resolution (px)	64	96	128	156	192	224	312	384	448
DoGNeXt (K9)	2.48	27.06	50.95	61.92	68.03	70.44	71.65	70.31	68.99
ConvNeXt V2	3.92	33.57	54.04	56.72	68.57	70.84	71.85	70.79	69.55

Table 4.7: DoGNeXt accuracy@1 on ImageNet1K at different resolutions for the different kernel sizes. Vanilla training without scale augmentation.

Resolution (px)	64	96	128	156	192	224	312	384	448
Kernel 9	2.48	27.06	**50.95**	**61.92**	**68.03**	**70.44**	**71.65**	**70.31**	**68.99**
Kernel 7	**4.99**	**31.11**	51.80	59.95	64.40	65.50	62.64	58.78	55.39
Kernel 5	1.62	8.03	14.49	15.91	16.55	14.86	9.16	7.19	5.92
Δ w.r.t. Kernel 9	+2.51	+4.05	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0	+0.0

4.5 Adapting Model FLOPs

This experiment aims at partially replicating the work done in the FlexiViT paper [6]. The idea is to have a single model, in their case a ViT, in ours a CNN, that is able to adapt the FLOPs count at inference. This has several benefits that were highlighted in the original work like: quickly adapting at inference the performance of the network depending on the hardware resources available, but also being able to leverage this for faster pre-training and transfer learning purposes. To ”flexify” a ViT the idea was to create a way to resize and interpolate the patchifier so that an arbitrary patch size could be used, therefore enlarging the patch size would reduce the FLOPs count. In case of a CNN it is easier due to the equivariance of the convolution operator, in fact, we can change the stride of the network at any time. Increasing the stride of a convolution has a problem, the kernel size. In the case the stride is larger than half the kernel dimension, we would occur in a situation where part of the activation is unused. DoGNeXt, by having the ability to resize the kernels, can avoid this undesirable effect. The objective is to keep the ratio between kernel size and stride constant, for example, going from stride 4 to 8 will result in the kernel size going from 7 to 15. This ensures that the whole activation

Table 4.8: Accuracy@1 on ImageNet1K at different resolutions. Custom scale-augmented training.

Resolution (px)	64	96	128	156	192	224	312	384	448
DoGNeXt (K9)	28.69	48.47	59.30	65.15	68.61	70.39	72.03	71.69	71.12
Δ w.r.t. vanilla	+26.20	+21.41	+8.34	+3.22	+0.59	-0.04	+0.38	+1.38	+2.13
ConvNeXt V2	32.25	51.47	61.13	63.69	69.20	71.05	72.44	72.51	72.11
Δ w.r.t. vanilla	+28.33	+17.90	+7.09	+6.97	+0.64	+0.21	+0.59	+1.73	+2.56

Table 4.9: DoGNeXt accuracy@1 on ImageNet1K at different resolutions for the different kernel sizes. Multi-kernel scale-augmented training. Comparison with custom scale-augmented training.

Resolution (px)	64	96	128	156	192	224	312	384	448
Kernel 9	12.64	39.34	54.54	62.34	67.24	69.45	**71.35**	**70.72**	**69.75**
Kernel 7	22.56	47.27	59.59	**65.43**	**68.49**	**69.86**	69.44	67.99	66.06
Kernel 5	**29.08**	**49.67**	**59.62**	63.61	65.22	65.51	62.52	59.68	56.35
Δ w.r.t. DoGNeXt (K9)	+0.39	+1.20	+0.32	+0.28	-0.12	-0.53	-0.67	-0.97	-1.36
Δ w.r.t. ConvNeXt V2	-3.17	-1.80	-1.50	+1.74	-0.71	-1.19	-1.08	-1.79	-2.36

is used. Another possible way to reduce the FLOPs count is to simply reduce the input resolution, this is strongly related to the experiments done in section 4.4, related to scale, moreover, it is not dependent on a specific architecture.

For all our experiments we computed the FLOPs of the forward pass using the tool provided in the Python library `fvcore`[25].

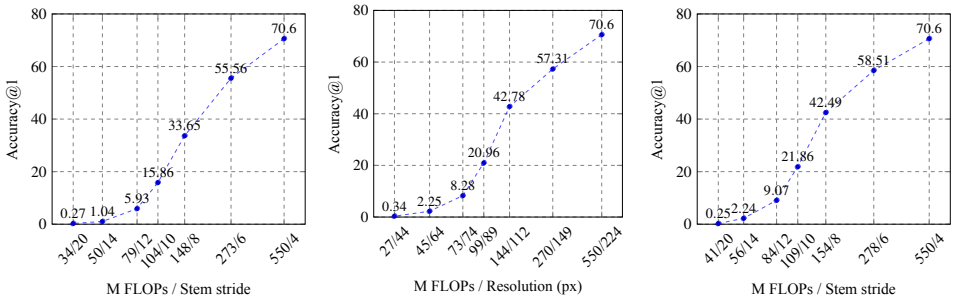
In Figure 4.4 we have a comparison between three different techniques: stride increase, resolution decrease and stride increase with kernel size increase. Note that we alter just the stride of the first DoGConv, the same applies for the kernel size. This will produce the desired FLOPs reduction. As we can see from the results we observe a sharp decrease in accuracy the more the number of FLOPs decreases. This is somewhat expected but not to this extent. The reason is simple and it lays in what we can call the ”effective resolution”. Doubling the stride of the first DoGConv produces a donwsampling effect for all subsequent layers. When we combine this information with the finding of section 4.4, we can see that a plausible problem can be the lack of resistance to scale decrease. In Figure 4.5 we perform the same comparison but with the scale-augmented DoGNeXt. We can see how the results are much

Table 4.10: DoGNeXt accuracy@1 on ImageNet1K at different resolutions for the different kernel sizes. Hybrid multi-kernel scale-augmented training. Comparison with custom scale-augmented training.

Resolution (px)	64	96	128	156	192	224	312	384	448
Kernel 9	26.45	46.74	58.01	63.98	67.93	**70.09**	**71.94**	**71.37**	**70.80**
Kernel 7	**30.72**	**50.20**	**60.40**	**65.42**	**68.54**	70.02	70.17	69.02	67.47
Kernel 5	29.76	48.22	57.66	62.37	64.44	65.12	63.05	60.30	57.22
Δ w.r.t. DoGNeXt (K9)	+2.03	+1.73	+1.1	+0.27	-0.07	-0.3	-0.09	-0.32	-0.32
Δ w.r.t. ConvNeXt V2	-1.53	-1.27	-0.73	+1.73	-0.66	-0.96	-0.5	-1.14	-1.31

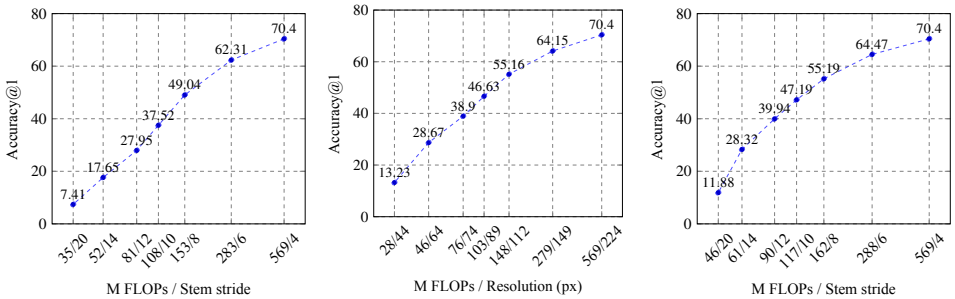
better with significant gains across the board.

This approach on CNNs has a major limiting factor, the stride. With ViTs one could theoretically choose an arbitrary patch size, but we cannot. For example, if we move from stride 4 to 6 we have a decrease of $1.5\times$ in the activation size, meanwhile, a ViT could change the patch size from 14×14 to 16×16 with a decrease of $\sim 1.14\times$ in the activation size. This reduction is proportional to the FLOPs of the subsequent operations. This lack of granularity, especially with small strides, is a problem that we can only fix with a resolution reduction. In summary, unfortunately the results are not convincing enough to justify the DoGNeXt approach. Resizing the kernel does not degrade the performance, something that is only possible thanks to the DoGConv, but is not making a major difference, scale augmentation is the key. Another point against the kernel resizing is the FLOPs penalty that comes with it. In Figure 4.5c we can see how the FLOPs count doesn't decrease a lot compared to 4.5a, especially when using a large stride. The resolution approach is the most promising. Future work could focus more on it, trying to replicate some benefits observed in FlexiViT with simply a resolution change.



(a) Stride (b) Resolution (c) Kernel and Stride

Figure 4.4: DoGNeXt accuracy @1 on ImageNet1K at different FLOPs levels. No scale augmentation. Each plot uses a different technique for reducing the FLOPs: Stride (a) increase, Resolution (b) decrease, Stride increase with Kernel adaptation (c).



(a) Stride (b) Resolution (c) Kernel and Stride

Figure 4.5: DoGNeXt accuracy @1 on ImageNet1K at different FLOPs levels. Scale augmented network. Each plot uses a different technique for reducing the FLOPs: Stride (a) increase, Resolution (b) decrease, Stride increase with Kernel adaptation (c).

Chapter 5

Conclusions

With the advent of deep learning computer vision moved from complicated and handcrafted techniques to a learning-based approach. Convolutional Neural Networks took the spotlight as one of the main architecture used to solve easily a variety of different tasks. For long, considering to bake into the kernels parametrized functions was thought to be extremely inefficient and useful only in very specific and limited scenarios due to the decrease in performance. In this work we showed that this is not anymore the case. The DoGNeXt architecture, thanks to the DoGConv kernel parametrization, is in fact able to perform similarly to ConvNeXt V2, one of the best performing models for image classification. We further proved the quality of our solution in a completely different domain with the MedMNIST dataset, a collection of different classification tasks on medical images. Moreover, we tried to use the kernel resizing properties of DoGConv to adapt the accuracy curve with respect to input scale, a problem that could benefit object detectors that struggle with small objects. The results obtained are encouraging but also mixed. This is because we created an alternative training recipe, that focuses more on small images, that is able to improve substantially the accuracy across different scales, beating the DoGNeXt approach. Finally, we showed how the model can be adapted on-the-fly to reduce the computational cost, founding that having a resizable stem provides a small, and inconsistent, improvement over naive techniques,

however, scale augmentation makes a significant difference.

This work wants to be a foundation and a first attempt towards high performing parametrized CNNs. We believe this to be extremely useful for specific applications, like scale equivariant networks, but also for advancements in AI explainability. Knowing the exact formulation of the kernels could spark new ideas to explain how CNNs learn to solve complicated vision tasks. Moreover, future work could focus on better evaluating and improving the proposed scale augmentation, with a special focus on object detection tasks.

Bibliography

[1] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, et al. Pytorch 2: faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 929–947, 2024.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. arXiv: `1607.06450 [stat.ML]`. URL: `https://arxiv.org/abs/1607.06450`.

[3] Z. Babaiee, P. Kiasari, D. Rus, and R. Grosu. Unveiling the unseen: identifiable clusters in trained depthwise convolutional kernels. In *The Twelfth International Conference on Learning Representations*.

[4] H. Bao, L. Dong, S. Piao, and F. Wei. Beit: bert pre-training of image transformers. In *International Conference on Learning Representations*.

[5] T. Barisin, K. Schladitz, and C. Redenbach. Riesz networks: scale-invariant neural networks in a single forward pass. *Journal of Mathematical Imaging and Vision*, 66(3):246–270, 2024.

[6] L. Beyer, P. Izmailov, A. Kolesnikov, M. Caron, S. Kornblith, X. Zhai, M. Minderer, M. Tschannen, I. Alabdulmohsin, and F. Pavetic. Flexivit: one model for all patch sizes. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, pages 14496–14506, 2023.

[7] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

[8] A. Coates, A. Ng, and H. 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.

[9] R. Ghosh and A. K. Gupta. Scale steerable filters for locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1906.03861*, 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[13] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: train 1, get m for free. In *International Conference on Learning Representations*, 2017.

[14] Y. Jansson and T. Lindeberg. Scale-invariant scale-channel networks: deep networks that generalise to previously unseen scales. *Journal of Mathematical Imaging and Vision*, 64(5):506–536, 2022.

[15] R. Joyseeree, S. Otálora, H. Müller, and A. Depeursinge. Fusing learned representations from riesz filters and deep cnn for lung tissue classification. *Medical image analysis*, 56:172–183, 2019.

[16] A. Kanazawa, A. Sharma, and D. Jacobs. Locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1412.5104*, 2014.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[18] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[20] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: common objects in context. In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, pages 740–755. Springer, 2014.

[21] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[22] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[23] M. A. Rahman and R. A. Yeh. Truly scale-equivariant deep nets with fourier layers. *Advances in Neural Information Processing Systems*, 36:6092–6104, 2023.

[24] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[25] F. A. Research. Fvcore: a collection of core functionalities for computer vision research. `https://github.com/facebookresearch/fvcore`, 2024. Accessed: 2025-02-20.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[27] V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, and L. Schmidt. Evaluating machine accuracy on imagenet. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8634–8644. PMLR, 13–18 Jul 2020.

[28] I. Sosnovik, A. Moskalev, and A. Smeulders. Disco: accurate discrete scale convolutions. *arXiv preprint arXiv:2106.02733*, 2021.

[29] I. Sosnovik, M. Szmaja, and A. Smeulders. Scale-equivariant steerable networks. In *International Conference on Learning Representations*.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[31] M. Tan and Q. Le. Efficientnet: rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[32] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou. Fixing the train-test resolution discrepancy. *Advances in neural information processing systems*, 32, 2019.

[33] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie. Convnext v2: co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

[34] D. Worrall and M. Welling. Deep scale-spaces: equivariance over scale. *Advances in Neural Information Processing Systems*, 32, 2019.

[35] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[36] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.

[37] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[38] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. Mixup: beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[39] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127:302–321, 2019.

[40] W. Zhu, Q. Qiu, R. Calderbank, G. Sapiro, and X. Cheng. Scaling-translation-equivariant networks with decomposed convolutional filters. *Journal of machine learning research*, 23(68):1–45, 2022.