CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

# Adding Context into Convolutional Neural Networks

**Sebastian Štefko**
Artificial Intelligence and Computer Science

May 2022
Supervisor: Mgr. Jan Šochman, Ph.D.

# BACHELOR'S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Štefko Sebastian** Personal ID number: **492294**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

## II. Bachelor's thesis details

Bachelor's thesis title in English:

**Adding Context into Convolutional Neural Networks**

Bachelor's thesis title in Czech:

**Přidání kontextu do konvolučních sítí**

Guidelines:

Propose and implement a modification of a standard CNN architecture which gathers contextual information from the input image faster than the original method. Get inspired by the recent success of Vision Transformers [1] architecture, where the additional context early on in the decision process together with a positional encoding leads to better results on many standard tasks.
1. Get familiar with the current CNN and Transformer architectures and their use for vision tasks.
2. Get familiar with CNN-based architectures which try to incorporate additional context early on in the processing (e.g. DeepLab [2]).
3. Modify a commonly used CNN architecture (e.g. ResNet [3]) to incorporate addition context cues into lower spatially-narrow-focused layers. Get inspired by the Transformer's attention module.
4. Compare the method on a standard dataset (e.g. COCO dataset) with the original CNN and other similar approaches.

Bibliography / sources:

[1] Kolesnikov et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, ICLR 2021.
[2] Chen et al., DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, TPAMI 2017.
[3] He et al., Deep Residual Learning for Image Recognition, CVPR 2016.

Name and workplace of bachelor's thesis supervisor:

**Mgr. Jan Šochman, Ph.D. Visual Recognition Group FEE**

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **05.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

| Mgr. Jan Šochman, Ph.D. | prof. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| Supervisor's signature | Head of department's signature | Dean's signature |

## III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

| Date of assignment receipt | Student's signature |
|---|---|

# Acknowledgement / Declaration

I would like to express my gratitude to Mgr. Jan Šochman, Ph.D. for his guidance and feedback during the writing of this thesis. My thanks also go to Ing. Jonáš Šerých and Ing. Michal Neoral, who provided me with many valuable advice. Lastly, I would like to thank my family for supporting me and thus allowing me to work on things I enjoy. Thank you.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 20th May 2022

.....

# Abstrakt / Abstract

Nejnovější články [1, 2, 3] ukazují, že neuronové sítě mohou těžit z využití širšího receptivního pole (kontextu) již na nižších vrstvách. Naopak běžně používaná architektura ResNet má pouze lokální kontext a její receptivní pole se zvětšuje pomalu. Trvá několik vrstev, než se její kontext rozšíří. Navrhujeme několik metod, jak přidat kontext do architektury ResNet. Konkrétně jsme zkoušeli: (i) vygenerovat náhodný kontext, (ii) rozšířit náhodný kontext o relativní poziční kódování a (iii) naimplementovat self-attention s náhodným kontextem a pozičním kódováním. Prozkoumali jsme také možnosti použití dilatované konvoluce a její modifikaci s náhodným tvarem konvolučního jádra.

Navržená rozšíření testujeme na úloze klasifikace do deseti tříd. Měříme, jak neuronová síť využívá kontext pomocí receptivního pole a efektivního receptivního pole. Naše metody porovnáváme s architekturami jako je HRNet, Vision Transformer a ConvNext.

Výsledky ukazují, že síť ResNet 50 využívá rozšířeného kontextu již na nižších vrstvách, nárůst výkonu však nebyl zaznamenán. Použití dilatované konvoluce s náhodným tvarem jádra konvoluce přineslo malé zlepšení ve výkonu. I přes výsledky našich experimentů je rozšířený kontext považován za přínosný ve snaze zlepšit přesnost neuronových sítí.

**Klíčová slova:** kontext, receptivní pole, konvoluční neuronová síť, Vision Transformer, ResNet 50

Recent papers [1, 2, 3] show that neural networks can benefit from using a wider receptive field (context) already in lower layers. On the contrary, a commonly used ResNet has a local context, and the receptive field grows slowly. It takes multiple layers before the ResNet reaches a broader context.

We propose several methods how to add context into ResNet architecture. In particular, we experimented with: (i) generating random context, (ii) enhancing random context with the relative positional encoding, and (iii) implementing self-attention with the random context and relative positional encoding. Alternatively, we tried using dilated convolution and its alteration with a randomised shape of a convolutional kernel.

We test the proposed ResNet extensions on the task of classification into ten classes. We measure how well the networks utilise the context using the receptive field and the effective receptive field. Our methods are compared with HRNet, Vision Transformer and ConvNext architectures.

Results show that the utilisation of the context was increased already in the lower layers of the ResNet 50 network. However, the performance increase was not registered. Only using randomised dilated convolution has shown a slight performance increase. Despite the results of our experiments, the extended context is still considered beneficial in the pursuit of better performance.

**Keywords:** context, receptive field, convolutional neural network, Vision Transformer, ResNet 50

# / Contents

# Tables / Figures

# Chapter 1
# Introduction

Convolutional neural networks (CNN) are widely used in computer vision. In problems such as image classification or segmentation, a neural network (NN) receives an image on the input, processes it and performs a given task. The way an image is processed depends on the architectural structure of the network. CNNs are organised into layers, most of them are convolutional, and some are pooling layers or a fully-connected layers. The output of each layer is a feature tensor. In computer vision, a feature tensor has typically four dimensions, two of which are the spatial dimensions, one is the channel dimension determining the number of individual learned feature maps, and one is the batch size dimension. The network gathers signals from an area on the lower layer and applies the convolutional filter to learn a new feature. The area in the input image is referred to as a receptive field (RF). The example in Figure 1.1 shows a single feature as one green square in the second layer. It was influenced by an area of 3 by 3 in the first layer, further influenced by an area of 5 by 5 in the input image. It is apparent that the RF increases by every layer. The feature tensor shrinks in spatial dimensions and increases the number of its channels as it passes through the network.

We are talking about the context available to the network when we consider the total area of the RF at any given layer of the network. RF serves as a good indicator of the size of the context in the network. In CNN s like ResNet [4], the context is expanded relatively slowly. That is because the convolutional kernel is relatively small compared to the size of the image. It takes several layers before the network holds broader contextual information. There are, however, architectures which manage to provide wider context faster and achieve better performance results.

One of such approaches is to modify the standard convolution. By doing so, we provide a larger context which promises better performance. As mentioned, convolutional kernels have a narrow scope. To make the scope wider while keeping the number of parameters the same, dilated convolution (see Section 3.4.1) was proposed, reaching further from the convolution's centre. Such modification is implemented, e.g. in DeepLab architectures [2, 5] as one of the main modifications for bringing wider context. Combining the approach of dilated convolutions in several layers has shown significant merit.

A different way of ensuring broader context was proposed in the high-resolution network (HRNet) [1] architecture, which performs unmodified convolutions. An HRNet keeps the high-resolution image representation and,

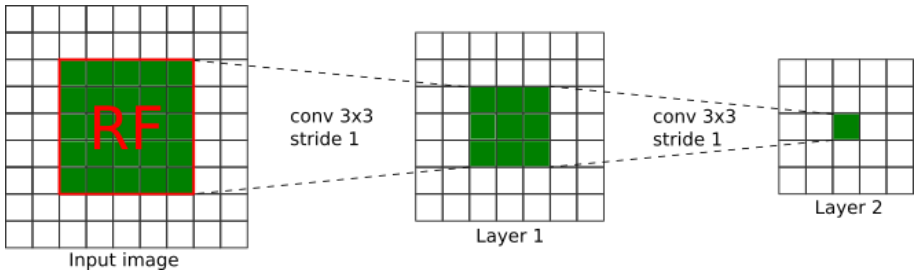
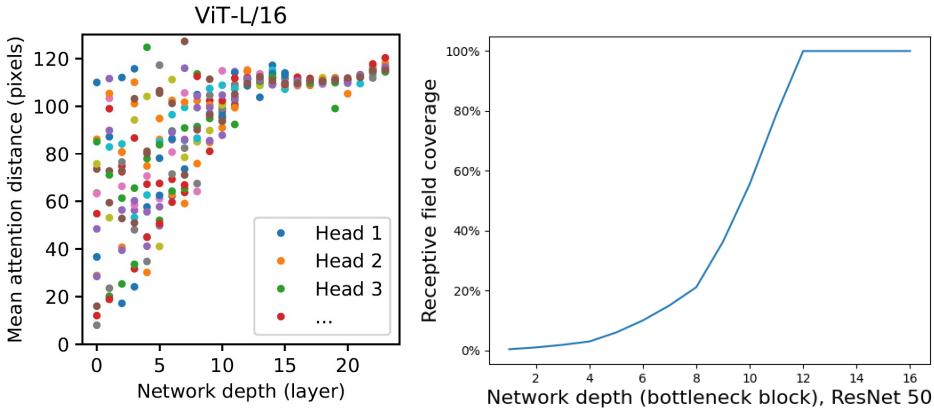



**Figure 1.1.** An example of the receptive field. A 3×3 convolution has been done on the input image and on layer 1. Receptive field of a single feature (depicted as a single green square) in layer 2 is 9 and 25 for layer 1 and the input image respectively.

along with that, computes several lower resolutions. On all of those resolutions, it computes a 3×3 convolution. Feature tensors of individual resolutions are combined together. The context increases when the low-resolution feature tensor to which the 3×3 convolution was applied is upscaled and concatenated to the high-resolution feature tensor. That is because when a 3×3 area gets upscaled, it covers a way larger area. This way of obtaining contextual information from the input data has proven to be particularly beneficial in the problem of semantic segmentation.

An inspiration for the expansion of context for this thesis was the recent success of the Transformer architecture [6] in natural language processing and later the Vision Transformers (ViT) [7] in computer vision. In the last year, they outperformed classical convolutional neural networks in terms of computational efficiency and accuracy in many tasks [7]. ViT brought many interesting ideas into computer vision as they completely substitute convolutions for the self-attention mechanism. Self-attention builds an image representation by relating all parts of the image. This operation allows the network to look anywhere in the image. It rapidly increases the receptive field giving the network a high context already in the early stages (see Figure 1.2a). In ViT, instead of the RF , a mean attention distance is used to quantify the amount of context. It is the average distance spanned by attention weights at different layers and is analogous to the receptive field size in CNNs ([7], Section D7). In contrast to the ViT, the receptive field grows much slower in CNN s (see Figure 1.2b). The whole context of the input image is first retrieved earliest at the latest layers of the network.

Providing a broader context to the neural network by the abovementioned architectures is one of the aspects which helps to increase their accuracy. We are aware of other options for making the network perform better, for example, changing the number of stages in every layer, reducing the number of activation functions and normalization layers or processing the input image differently at the beginning of the network [3]. We will, however, focus on adding context in order to increase the performance. Our goal is to modify a standard CNN architecture to make a broader context available to it. We




a) b)

**Figure 1.2.** The figure shows a growth of the receptive field (analogous term in ViT is the attention distance) for the a) Vision Transformer [7] and b) ResNet 50 [4]. ViT-L/16 contains 16 self-attention heads at every layer. The individual dots in a) show the mean attention distance for every head at a particular layer. The size of the input image for both ViT and for ResNet was 224×224 pixels.

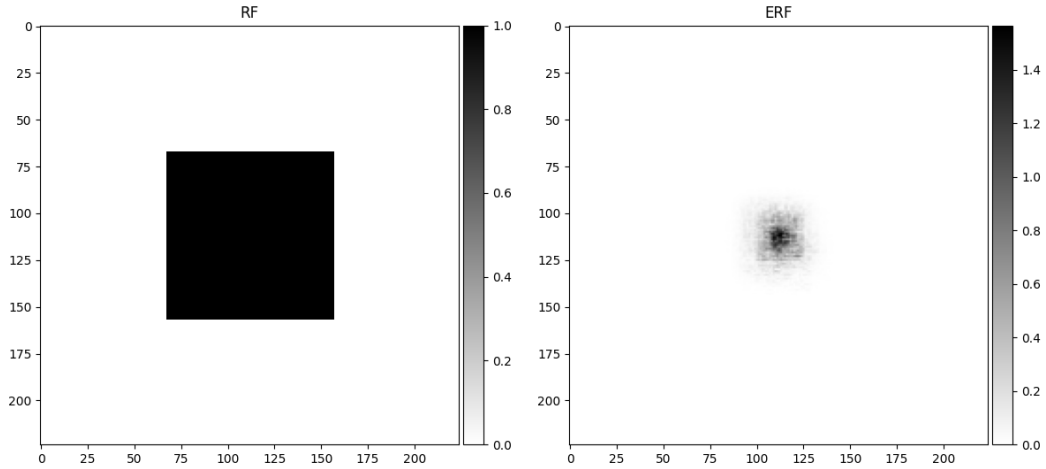
will use innovative methods as well as get inspiration from already verified architectures.

RF is usually used to evaluate the amount of context. It turns out that when creating a new feature, not every part of the RF area contributes the same. A method for such measurement has been proposed by [8], and it is called the effective receptive field (ERF). To calculate the impact of input on one feature vector in the feature tensor, a partial derivative $\partial y_p / \partial x_{i,j}$ is taken, where $y_p$ is the feature vector and $x_{i,j}$ represents the input pixel of an image on position $(i, j)$. This partial derivative can be computed using backpropagation. To compute the ERF, we set the error gradient with respect to an auxiliary loss $l$ as $\partial l / \partial y_p = 1$ and $\partial l / \partial y_{i,j} = 0$ for $i, j \neq p$. The error gradient $\partial l / \partial x_{i,j}$ on the first layer then equals to the desired $\partial y_p / \partial x_{i,j}$ [8]. This shows us how much $x_{i,j}$ contributes to $y_p$.

An illustration of the difference between RF and ERF is shown in Figure 1.3. The computation of the derivative was taken from the centre of the feature tensor at the end of layer 2 in the ResNet 50 network. It can be visually seen that the ERF occupies only a fraction of a theoretical RF and resembles the Gaussian distribution.

## 1.1 Contributions

For our research, we have chosen ResNet 50 [4] for its straightforward implementation, which was revolutionary at the time. It is also still being used for solving many computer vision tasks, so it is worth trying to improve its performance and accuracy.




a) b)

**Figure 1.3.** A difference between receptive field and effective receptive field on ResNet 50, computed from the middle feature on the second layer.

We propose the following contributions in this thesis:

- We made the following modifications to the original architecture of the ResNet 50 network to provide a higher context in the learning process.
  - Initially, we randomly generated a context, potentially spanning the whole image. The context is random but fixed in each layer where it is added to the ResNet 50 network. (Section 3.1)
  - We extended the randomly generated context by adding relative positional encoding. (Section 3.2)
  - A part of the self-attention mechanism for comparing the similarity between any two feature vectors was implemented in the ResNet 50. It was then combined with the randomly generated context method. (Section 3.3)
  - We used the dilated convolution in the architecture in two ways. First, we replaced the original 3×3 convolution with the dilated one, and second, we inserted the dilated convolution into the architecture. (Section 3.4)
  - We randomized the regular pattern of the dilated convolution and inserted it into ResNet 50. (Section 3.5)
- We tested our implementations on the ten class classification problem from the Imagenette dataset [9]. We compared our methods with the original ResNet 50 and other well-known architectures like Vision Transformers [7], HRNet [1] or ConvNext [3].
- To study the size of the context used by each method, we studied the receptive field and the effective receptive field [8]. We quantified the amount of context using the mean distance measure. (see experiments in Chapter 4)
- We managed to verify that our proposed methods increased the context in the network. However, no significant performance boost was observed for our methods.

# Chapter 2
# Architectures

Since the immense success of AlexNet [10] in 2012, convolutional neural networks became a popular architecture. The only limitation of these networks was initially their maximum depth, i.e. how many convolutional layers could be stacked on top of each other. The deeper the net was, the more difficult it was to optimise it. A sudden degradation of performance was a common occurrence. Another challenge was vanishing gradients, which occurred during the training and practically prevented the learning process. Most of these problems were mitigated by introducing the deep residual networks [4], known as ResNets.

As mentioned in the Introduction, ResNets are a popular backbones used plainly or with modifications [11, 12] in many neural networks. We will be exploring options for adding context to the ResNet 50 architecture in our experiments. Some of the principles we are inspired by were developed in ViT, which abandon the principle of convolutions and instead use self-attention. To better understand these architectures, we will describe them in more detail in this chapter.

## 2.1 ResNet 50

ResNet 50 is a variant of an architecture proposed by a paper [4], introducing a new concept of residual learning for image recognition. Individual variants (see Figure 2.1) differ in the total number of convolutions ranging from 18 all the way up to 152. The networks can be virtually arbitrarily deep, thanks to the skip connections, which is the most significant contribution of the mentioned paper. Skip connections allow the network to learn only the difference between the blocks into which the convolutions are organised (see Figure 2.2). It is done by summing an output from a previous block to the output of a current block. These blocks are of two types in ResNets. Variants with 18 and 34 convolutions use the basic block, consisting of two consecutive 3×3 convolutions, and the other variants use a more complex bottleneck block (see Section 2.2.1).

We will be dealing with the ResNet 50 version, which consists of 50 convolutions in 16 bottleneck blocks further organised into four layers, as shown in Figure 2.1. The input image is resized to a uniform shape of 224×224 pixels. First, a 7×7 convolution with stride 2 followed by 3×3 max-pooling is applied

| layer name | output size | 18-convolutions | 34-convolutions | 50-convolutions | 101-convolutions | 152-convolutions |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | [3×3, 64; 3×3, 64] ×2 | [3×3, 64; 3×3, 64] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 |
| conv3_x | 28×28 | [3×3, 128; 3×3, 128] ×2 | [3×3, 128; 3×3, 128] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×8 |
| conv4_x | 14×14 | [3×3, 256; 3×3, 256] ×2 | [3×3, 256; 3×3, 256] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×23 | [1×1, 256; 3×3, 256; 1×1, 1024] ×36 |
| conv5_x | 7×7 | [3×3, 512; 3×3, 512] ×2 | [3×3, 512; 3×3, 512] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

**Figure 2.1.** Table showing different configurations of residual networks. 18/34 convolution variants are built from basic blocks in 4 layers. 50/101/152 convolution variants are made of bottleneck blocks also in 4 layers. In the column output size are listed spatial dimensions of the image representation in every layer. Residual connections are around every block [4].

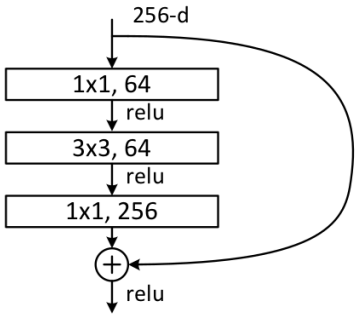



**Figure 2.2.** An example of a bottleneck block with the residual connection. Output from the previous block with 256 channels is squashed to 64 channels by the first 1×1 convolution; the context is expanded by the 3×3 convolution, and the last 1×1 convolution outputs back 256 channelled data to which a residual connection data is added. Last, a ReLU is applied [4].

before the image continues to the residual connections section formed out of the bottleneck blocks. The spatial dimensions of the input image stay the same in the entire layer and get halved at the end of it while increasing the number of channels by a factor of four. The final shape of the feature map is 7×7×2048 before global average pooling produces a vector of 2048 dimensions, which is inputted into a 1000-way fully-connected layer with softmax [4].

### 2.1.1 Bottleneck block

The bottleneck block is the main building block of the ResNet 50 architecture. It consists of three convolutions on top of each other. It starts with a 1×1 convolution, which reduces the number of channels to a quarter of the original size. This is followed by a 3×3 convolution that incorporates contextual information from the picture. Lastly, another 1×1 convolution restores

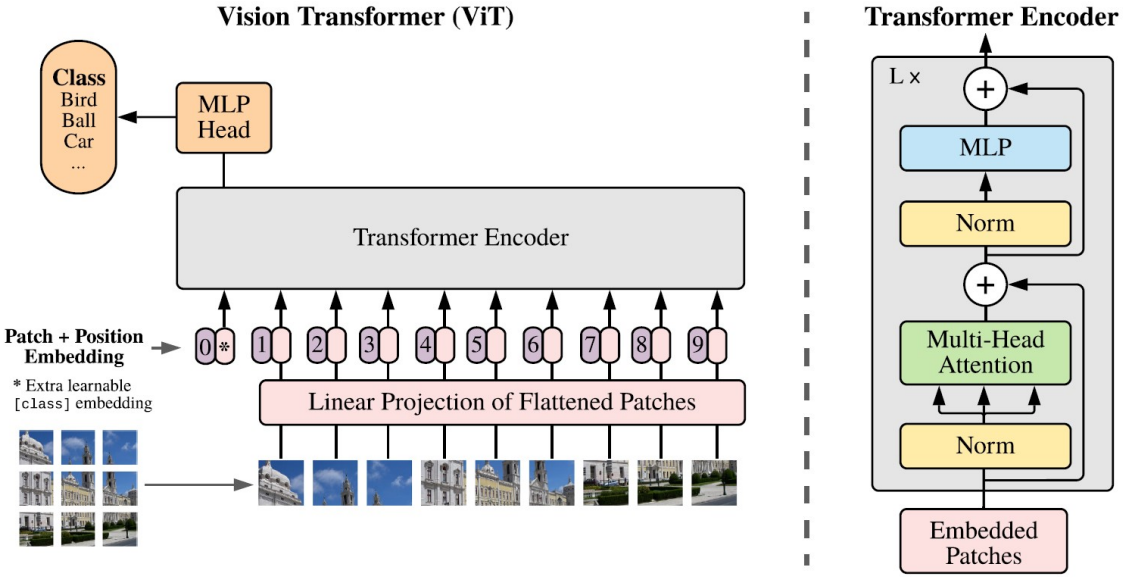



**Figure 2.3.** Architecture of the Vision Transformer model [7].

the number of channels to the original size. After every convolution, there is a ReLU activation function. An identity feature vector, which is the output feature map from the previous block, is added at the very end to the output of the last 1×1 convolution. This is the trick of residual learning — a shortcut connection (also skip or residual connection). The bottleneck block only learns the difference from the original data on the input to the block. The idea is that if the identity mappings were optimal, the network would learn the weights such that they were close to zero, easier than learning the function to be an identity.

## 2.2 Vision Transformer

Since the Transformers [6] revolutionised the natural language processing field of artificial intelligence, the ideas behind this architecture have also been adopted in computer vision. Such modification has been proposed in [7] and is called the Vision Transformer (ViT). It is similar to the standard CNN architecture, but it uses a self-attention mechanism (see Section 2.2.1) instead of convolutional layers. The ViT has several advantages over the standard CNN architectures. First, it lacks image-specific inductive biases, which are inherent to the CNNs. Second, the ViT is more accurate, when trained on large datasets [7]. This is because the self-attention mechanism can learn the same [13] and even more complex relationships than the convolutional layers in a CNN due to the global attention of all parts of the input image.

Transformer is a model which operates on sets of tokens. These tokens were originally words; however, in the vision, we are dealing with 2D images.

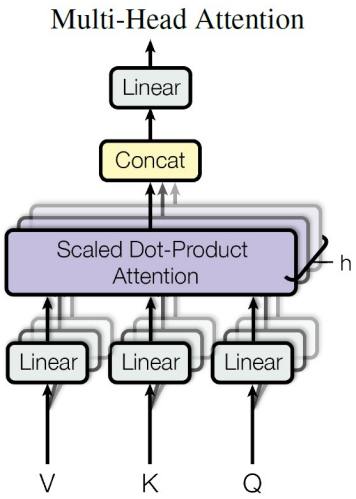



**Figure 2.4.** Multi-Head Self Attention with $h$ layers of self-attention function [6].

Naturally, the possibility of feeding a sequence of individual pixels to the network is an option but computationally unfeasible. Instead, an input image of size 224×224 pixels is divided into small patches, typically 14×14 pixels in size, resulting in a 16×16 grid of patches. Every patch is linearly projected by a trainable projection matrix to get the required sequence vector for the input called the *patch embeddings*. At this moment, positional relationships are lost as the embeddings are arranged in parallel to the next layer. In order to retain the positional information, learnable positional encoding (see Section 2.2.2) is added to every patch. On top of that, there is also a learnable embedding (see patch zero in Figure 2.3) added to the sequence at the beginning of the input. The first token gathers relational information from all the other patches during training and serves as the image representation used by the classification head. This process is visualised in the left-hand side of Figure 2.3.

The encoder consists of two sublayers, each having a layer norm applied before the next block. The first sublayer contains the core, computationally most expensive operation, the multi-head self-attention (see Figure 2.4). It consists of linear input projections preparing input data for several layers of the self-attention function, whose output is concatenated and again linearly projected. The second layer is a simple, fully connected feed-forward network. Residual connections are applied after every sublayer. See right-hand side of Figure 2.3.

### 2.2.1 Self-Attention

The self-attention mechanism is a type of neural network layer that allows the model to learn relationships between features in an image. It operates on sequences, which in our case are the patch embeddings. It relates every patch to all the others, computing similarities between them. Self-attention can be described as mapping a query, and a set of key-value pairs to an output [7]. The computation is done simultaneously as the data are packed into matrices.

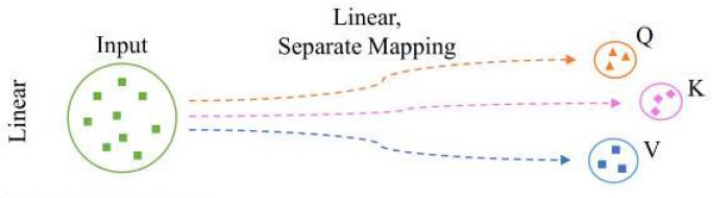



**Figure 2.5.** Linear mapping of the input embeddings to matrices $Q$, $K$, $V$ [14].

To obtain query $Q$, key $K$ and value $V$ matrices (see Figure 2.5), input data are passed through linear projection matrices, which have learnable parameters $[q, k, v] = zU_{qkv}$, where $z$ is the input vector of patch embeddings and $U_{qkv}$ is the projection matrix. The attention operation itself is computed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

First, a dot product of queries and keys is calculated and reweighted by a $\sqrt{d_k}$, where $d_k$ is the dimensionality of the keys. This gives us similarities between all patches. A distribution output (also called attention weights) from the softmax determines the amount of attention every patch should pay to all other patches by multiplying the weights with the matrix of values $V$. In other words, we obtain a matrix (attention scores) which tells us how significant are all the other tokens for every token in a sequence.

### 2.2.2 Positional Encoding

In natural language processing, a disarranged order of words in a sentence can cause a loss of meaning or even a change of meaning. For recurrent neural networks, which take one word after another, this is not a concern. However, the original Transformer architecture processes whole sentences all at once, which causes the word order to be lost. To retain it, a positional encoding is added as it returns positional information to the network. The same principle applies in vision, where we work with image embeddings instead of words. There are many options for how the positional encoding can be implemented.

In the original Transformer architecture, there is a fixed function that generates positional information depending on the position in the sequence. It must meet the following requirements:

- Each position in the sequence should have the same value regardless of the total length of the sequence or the input itself
- To avoid pushing the embeddings into very distinct subspaces, they need to be from a limited range of values

One such function was proposed in [6],

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d}) \tag{2}$$

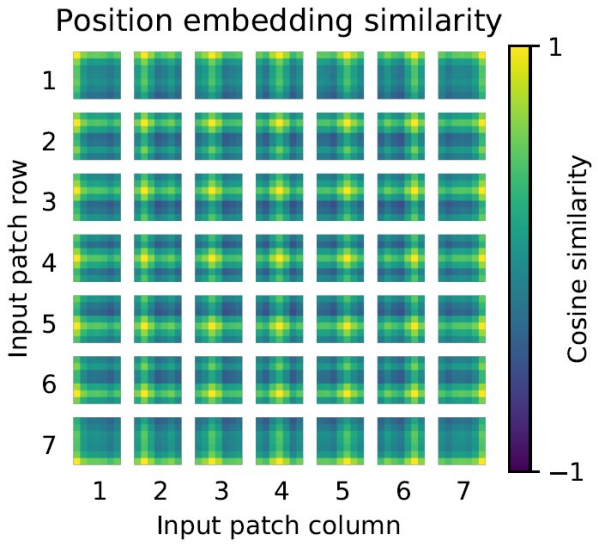



**Figure 2.6.** Example of learnable positional encodings after the training in Vision Transformer architecture reshaped and arranged so that every image patch corresponds to its original position. [7].

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d}) \tag{3}$$

where $pos$ is a position of the word or patch in the sequence and $i$ is the $i$th dimension of the $d$-dimensional positional embedding. Even dimensions are encoded by a $sin()$ and odd by a $cos()$ functions.

The equations (3) and (4) define positional encoding for a 1-dimensional sequence. There is also a 2-dimensional positional encoding, which is useful when dealing, for example, with images. It is defined as follows,

$$PE_{(x,y,i)} = sin(x/10000^{4i/d}) \tag{4}$$
$$PE_{(x,y,i+d/4)} = cos(x/10000^{4i/d}) \tag{5}$$
$$PE_{(x,y,j+d/2)} = sin(y/10000^{4i/d}) \tag{6}$$
$$PE_{(x,y,j+3d/4)} = cos(y/10000^{4i/d}) \tag{7}$$

where $(x, y)$ is a point in a 2d space and $i, j \in \langle 0, d/4)$, where $d$ is the number of channels.

ViT [7] approaches the issue in a different way. Instead of adding precomputed positional encodings, only 1-dimensional trainable encodings are added to the patch embeddings. These get trained during the learning process. Figure 2.6 shows that the network actually learns sensible positional relationships. The paper also mentions the difference between adding 1-dimensional or 2-dimensional positional embeddings concluding there is no significant difference in performance as opposed to no encodings at all.

Means of attaching the positional embeddings to the data also differ as they can be either summed or concatenated with the input. Summing retains the computational complexity while concatenating increases the dimensionality of data and, therefore, increases hardware resource demands.

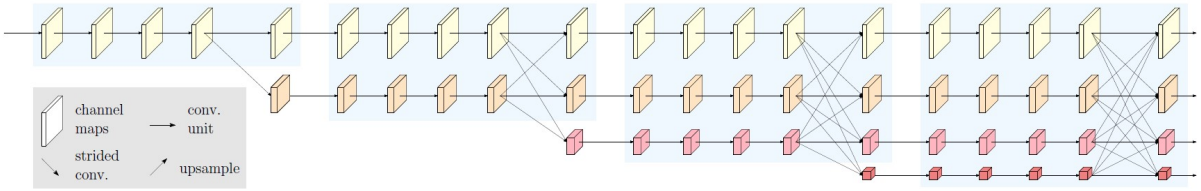



**Figure 2.7.** HRNet architecture, showing separation of lower resolutions after every layer in the whole network. [1]

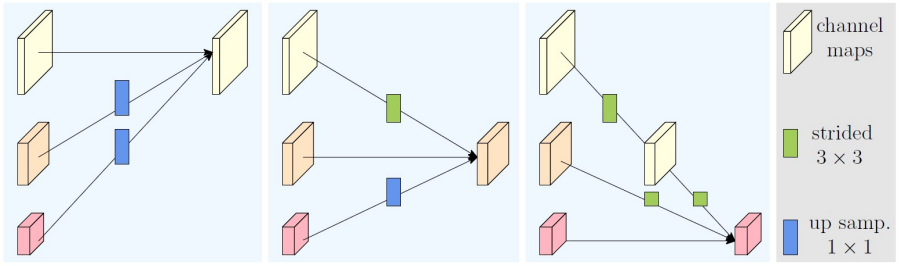



**Figure 2.8.** Changes in the resolution at the end of every layer are performed by 3×3 convolution with stride 2 for downsampling and by bilinear upsampling followed by 1×1 convolution for upsampling. [1]

## 2.3 High-Resolution Network

The High-Resolution Network (HRNet) [1] is a CNN architecture used mainly for tasks like semantic segmentation, object detection or human pose estimation, where high-resolution representations are needed. This architecture is interesting for us as it expands its context while performing only convolutions without any modifications. We will be using it for comparison with our modified architectures.

The network architecture consists of 4 stages, starting with only a high-resolution representation from the input. After every stage, a new lower resolution is separated from the higher ones (see Figure 2.7). The new lower resolution is half of the one separated last but has twice as many channels. There is an operation between stages called multi-resolution group convolution [1], where every resolution contributes to all the others. A resolution is downsampled (strided convolution) or upsampled (Figure 2.8) so it can be added to the rest of the resolutions.

# Chapter 3
# Method

In this chapter, we introduce the proposed methods for increasing the receptive field in convolutional neural networks. The goal is to extend the contextual information available to a neural network. We explored several approaches to architectural changes. As suggested by ([8], Section 4), instead of connecting each feature value to a local convolution window, sparse connections to a larger area in the lower layer of the network should be beneficial for increasing RF and thus ERF. We propose a method for achieving the sparse connections by permuting the feature tensor in the spatial dimensions. We also modify dilated convolution to a not grid-like shape, also suggested by [8]. For our method with the attention mechanism, we were inspired by the ViT, where it is used.

We build on top of the original residual CNN, ResNet 50 [4]. We implemented our methods into the architecture to add context and tested it against other well-known architectures in terms of the size of the RF, ERF and its performance.

## 3.1 Random sparse context

The task is to increase the amount of information processed by a convolution kernel at a single position. Under normal circumstances, a 3×3 kernel sees only the neighbouring eight feature values, which are all local. We propose extending the context of the convolution by considering a fixed number of other randomly selected positions from the feature tensor. We obtain a dense interconnection of individual features in the feature tensor when repeating this process on multiple consecutive convolutional layers. Figure 3.1 visualises this process for one feature value.

We modify a standard bottleneck block (see Section 2.1.1) in ResNet 50. We implement this by appending the same data tensor permuted along the spatial dimensions to the original tensor (see Figure 3.2). Formally, we have a data tensor $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$, where $C$ is a number of channels and $(H, W)$ are the spatial dimensions. In ResNet 50, $\mathbf{x}$ is the output of a 3×3 convolution in a bottleneck block. Let $k \in \mathbb{N}$ be the number of permutations, i.e. a fixed number of random neighbours which get associated with each spatial position in $\mathbf{x}$. The neighbours are formed by $k$ times permuting the tensor $\mathbf{x}$, producing $\mathbf{x}'_i$ where $i = 1, \ldots, k$ and then constructing tensor $\mathbf{x}_C$. Concatenating the tensor $\mathbf{x}$ and tensors $\mathbf{x}'_i$: $\mathbf{x}_C = [\mathbf{x}, \mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_k]$. Tensor $\mathbf{x}_C$ is the

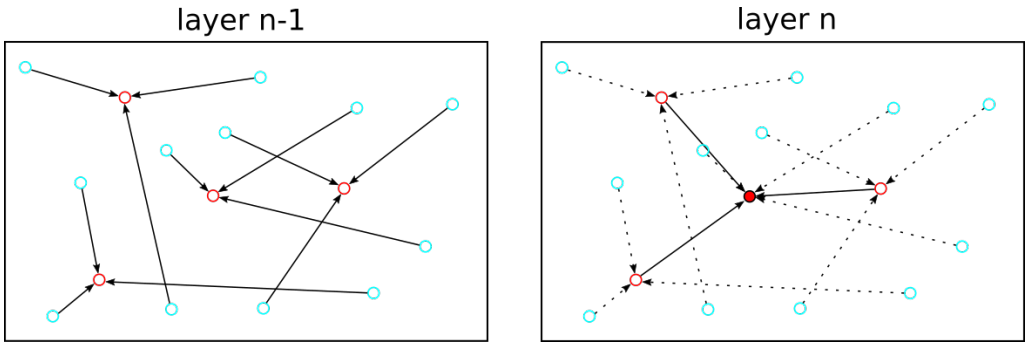



**Figure 3.1.** An illustration of the growth of the context with random sparse connections. To simplify the scheme, a 1×1 convolution is used. Also, the circles represent only some features in the feature tensor. In layer $n-1$, every red circle has its context extended by three random blue circles. After applying the convolution in layer $n$, the full red circle receives context from three empty red circles, which already had their context extended in layer $n-1$. This results in a total of 15 connections for the full red circle after the second convolution.

new input for the last 1×1 convolution in the bottleneck block with shape of $(k+1)C \times H \times W$. It has $k$ times more channels than $\mathbf{x}$. This, however, does not pose a problem as the 1×1 convolution shrinks or extends the channels (depending on $k$) to the original value of $4C$, so no other modifications of the architecture are necessary. That is a standard size of the output from the bottleneck block.

As a result, any position on the feature map is now able to attend $k$ other random positions. The subset of attainable values is quickly growing; therefore, the context was extended. Adding a random context is repeated on multiple convolutional layers in every bottleneck block or only on the higher spatially-narrow layers. This and the value $k$ control the amount of context supplied to the network.

## 3.2 Augmenting the context with relative positional encoding

Permuting data tensors and feeding them back into the network is a very straightforward method of increasing the attention distance already in the lower layers of the network. However, as the individual feature vectors in the feature tensor were randomly permuted, the original positional relations were lost. A feature vector in the permuted tensor can be close or far from its original position before the permutation. We want to retain this positional information as we think it is beneficial for the network to know which features are more local and which are more distant. This is similar to ViT architecture, where in order to keep positional information, a positional encoding is added. We proceeded in an analogous manner. In our case, a relative positional encoding is used.

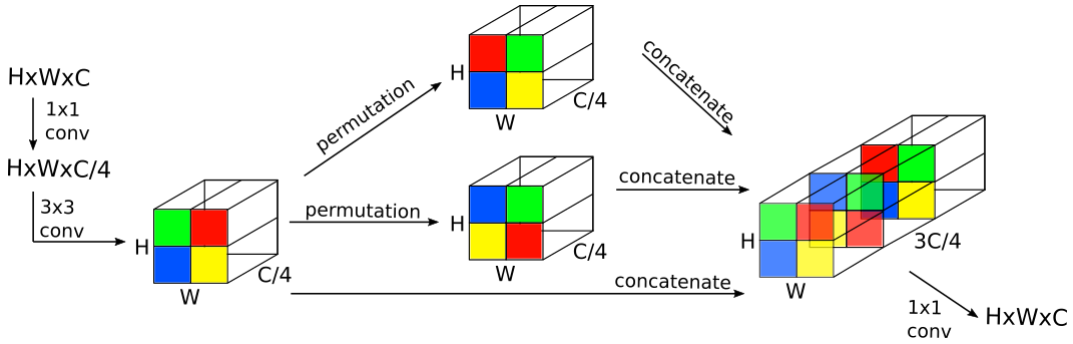



**Figure 3.2.** Modified bottleneck block with extended context. The input is a tensor with the shape H×W×C (in this scheme $H = 2, W = 2$). The cuboid on the left is the output of the 3×3 convolution with spatial dimensions $H, W$ and $C/4$ channels. In this case, it gets permuted twice (the cuboids in the middle). The original tensor is then concatenated with the permuted ones resulting in a tensor with $3C/4$ channels, which is the input a for 1×1 convolution. The 1×1 convolution is original in the bottleneck block, now with more channels on input.

Relative positional encoding is a subtle modification of the classical positional encoding [15, 6]. It considers relative distances between any two positions on a grid of spatial dimensions. The implementation details closely follow equations (4-7) in Section 2.2.2.

Let $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ be the original data tensor. The tensor $\mathbf{x}$ gets $k$-times permuted producing $\mathbf{x}'_i$, where $i = 1, \ldots, k$. For every permuted tensor $\mathbf{x}'_i$, we want to produce a relative positional encoding $\mathbf{E}_i \in \mathbb{R}^{C \times (HW)}$, which contains encoded relative distances between spatial positions in $\mathbf{x}'_i$ and $\mathbf{x}$. Let $\mathbf{R}^i$ be the matrix of relative row distances between elements in the original data tensor $\mathbf{x}$ and the $i$-th permuted tensor $\mathbf{x}'_i$. Let $\mathbf{C}^i$ be the matrix of relative column distances between elements in the original data tensor $\mathbf{x}$ and the $i$-th permuted tensor $\mathbf{x}'_i$. From equations (4-7) in Section 2.2.2, we obtain four vectors of frequencies $\mathbf{f}_j \in \mathbb{R}^{C/4}$; $j = 1 \ldots 4$. $\mathbf{E}_i$ encodes relative distances between rows in the first half of the channels and relative distances between columns in the second half of the channels. Each of those halves is encoded by the $sin()$ function in the first half and by the $cos()$ function in the second half. Hence the final relative positional embedding $\mathbf{E}_i$ is created as follows, $\mathbf{E}_i = [\widehat{\mathbf{R}^i}\mathbf{f}_1^T, \widehat{\mathbf{R}^i}\mathbf{f}_2^T, \widehat{\mathbf{C}^i}\mathbf{f}_3^T, \widehat{\mathbf{C}^i}\mathbf{f}_4^T]$, where $\widehat{\mathbf{A}}$ is flattened tensor. Lastly we add the reshaped relative positional encodings $\mathbf{E}_i$ to the data $\mathbf{x}'_i$.

## 3.3 Adding Attention

The attention mechanism computes how similar are two vectors from a given set. The similarity is evaluated as a magnitude of a dot product. The larger the dot product is, the more similar the two vectors are.

New connections between individual feature vectors are created as a data tensor $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ is permuted, producing $\mathbf{x}'_i$; $i = 1, \ldots, k$. However, not

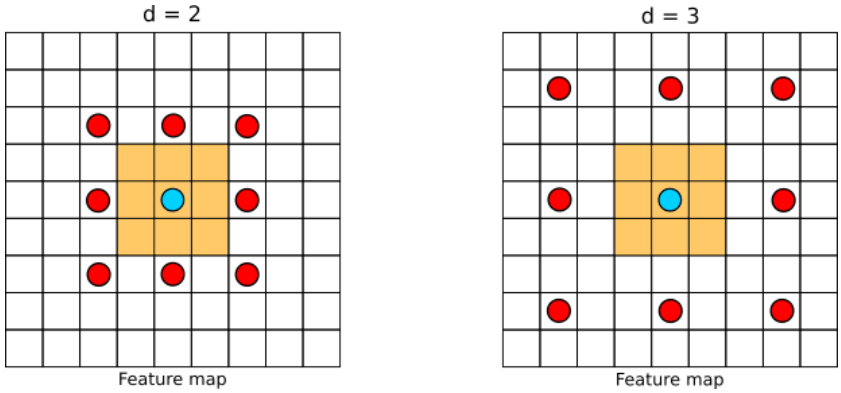



**Figure 3.3.** Examples of a 3×3 dilated convolution for $d = 2$ and $d = 3$. The centre of the convolutional kernel is depicted by a blue dot. Orange tiles show, where a standard convolution would have been. The red dots show the dilated positions of the kernel values.

every new connection with the original data tensor has the same significance as others. We use a modified attention function [6] to evaluate which feature vector should be paid more attention to and which not. We implement the attention as follows. First a relative positional encoding is added to all data tensors (according to Section 3.2). We denote $\mathbf{x}'_0 = \mathbf{x}$. For all $i = 0 \dots k$, we construct a matrix of the attention weights $\mathbf{s}_i \in \mathbb{R}^{H \times W}$, such that $\mathbf{s}_i(l, m) = \mathbf{x}^T(l, m, :)\mathbf{x}'_i(l, m, :)$, $\forall l, m$. There are $k+1$ matrices of the attention weights, each for the original tensor and all permuted tensors. A softmax is computed, $softmax(\mathbf{s}_0, \dots, \mathbf{s}_k)$ to get a distribution from the weights. Then the data is reweighted for $i = 0 \dots k$, $\mathbf{x}'_i(l, m, k) = \mathbf{s}_i(l, m)\mathbf{x}'_i(l, m, k)$; $\forall k, l, m$. We then sum all tensors $\sum_0^k \mathbf{x}'_i$ to produce an input for the following 1×1 convolution.

## 3.4 Dilated convolution

It was mentioned in the Introduction that dilated convolution is a successful method of increasing context. We also use dilated convolution in the ResNet 50 architecture.

Dilated convolution [16] (see Figure 3.3), also known as atrous convolution, is a convolution which splits the normally coherent convolutional kernel into discontinuous pieces, creating holes between the individual filter values. Dilated convolution is defined by the parameter $d$, which specifies the distance between individual values, which form a regular grid. The kernel is larger, having wider outreach while keeping the same number of parameters. A standard convolution has $d = 1$ as the distance between two filter values in a horizontal or vertical direction. Figure 3.3 shows an example of dilated convolution for different values of parameter $d$.

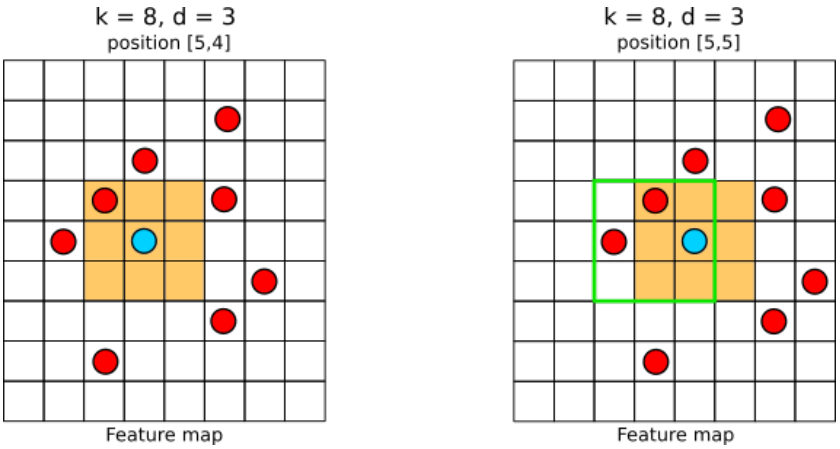



**Figure 3.4.** Example of a random dilated convolution performed on the feature map on two consecutive positions with stride 1. Blue dots represent the centre of the convolutions, and red dots the positions of the convolution filter values. Orange tiles show a standard 3×3 convolution. For better orientation, a green square shows the previous position of the standard convolution.

### 3.4.1 Inserted dilated convolution

In ResNet 50, the context is expanded only by the 3×3 convolutions, which takes many layers for the network to achieve a broader context. We inserted a dilated convolution into every bottleneck block. It is located right after the 3×3 and before the 1×1 convolutions, taking and outputting the same number of channels, so no other changes to the architecture are required.

## 3.5 Randomized dilated convolutions

The standard dilated convolution has individual filter values positioned in a regular grid with holes between them. We remove the regular pattern and introduce a random positioning to the filter values in the convolutional kernel. This is because choosing a larger parameter $d$ for a regular dilated convolution causes artefacts — isolated areas of the receptive field without any local context.

Two parameters define random dilated convolution. The first parameter $k$ specifies how many different positions besides the centre of the convolution a kernel will look at. Parameter $d$ defines a maximum vertical or horizontal distance of the filter value from the kernel centre. Visualization of this method is shown in Figure 3.4

Let $\mathbf{M} = [-d, d] \times [-d, d]$ be a set of possible offsets from the centre of the kernel, where $d$ is the maximum distance in any direction. Let $\mathbf{H}$ be a subset of $k$ randomly chosen offsets from $\mathbf{M}$. The random dilated convolution operation is defined as:

$$g(x, y) = \omega * f(x, y) = \sum_{(dx,dy)\in\mathbf{H}} \omega(dx, dy) f(x - dx, y - dy) \qquad (1)$$

where $g$ is the output image, $f$ is the original image and $\omega$ is the randomized filter kernel.

# Chapter 4
# Experiments

The previous chapter introduced new ways for adding context into the ResNet 50 backbone. Now we evaluate the proposed approaches both quantitatively and qualitatively by visualising the size of the context at different positions in the network. Then we check whether the methods led to a performance increase. We compare modified ResNet 50, Vision Transformers, HRNet and ConvNext to see the overall impact of our methods.

## 4.1 Methods of measurement

For measuring the amounts of context, we use the receptive field. It shows areas in the input image a network can query information from. The actual impact of the theoretically attended areas is measured using the effective receptive field [8]. The RF and ERF can be measured for any spatial position on the feature map in any network layer. Our experiments show the situation only for the centre most position in the feature map on various layers as it mitigates the effects at the edges of the image. It is also a well comparable position to do so. To quantify the size of the attended area, we introduce a *mean distance* $m$ measured from the position of ERF measurement (centre point $c$). Let $\mathbf{F} \in \mathbb{R}^{H \times W}$ be the normalised feature map, where $H, W$ is typically equal to 224 in our experiments. We have an Euclidean distance function $d(x)$, giving us the distance between $c$ and any $x \in \mathbf{F}$. The mean distance $m_c$ at a position $c$ is a sum of weighted distances from the centre point to all points in the feature map, computed as follows:

$$m_c = \sum_{i}^{H} \sum_{j}^{W} d(\mathbf{F}_{i,j}) \mathbf{F}_{i,j}$$

The greater the mean distance, the more important pixels further from the centre were for the neural network.

The results of the experiments are visualised in such a way that one image simultaneously shows RF (any non-white areas), ERF, the centre point, and the size of the mean distance depicted by a circle with a radius of the mean distance. The visualised data are averages over a batch of ten images.

| Architecture | Top 1 accuracy (in %) | params $\times 10^6$ |
|---|---|---|
| ResNet 50 | 90.42 | 23.5 |
| ResNet 50, 1 Permutation | 90.52 | 28.5 |
| ResNet 50, 3 Permutations | 89.63 | 38.6 |
| ResNet 50, 5 Permutations | 89.81 | 48.7 |

**Table 4.1.** Performance results on the validation set of random sparse context compared with the ResNet 50.

## 4.2 Dataset and training

The evaluation was performed on the task of classification into ten classes from a dataset Imagenette [9]. It is a subset of the full ImageNet 1k dataset [17], containing a thousand classes. The Imagenette contains approximately 9.4 thousand images in a training set and 3.9 thousand in a validation set.

In our training setup for ResNet 50, we followed the original paper [4] as closely as possible. We used random weights initialisation according to [18]. The models were trained for 100 epochs using an SGD optimiser with a batch size of 32. The learning rate starts at 0.1 and is divided by ten every 30 epochs. Standard image augmentations like random flipping and rotating were used, and the images were resized and cropped to be precisely 224×224 pixels in size.

## 4.3 Random sparse context

We performed the first experiment using a random sparse context implemented by permuting the feature map described in Section 3.1. Individual runs differed in the number of permuted tensors. We tested one, three and five permutations in every bottleneck block of the network. Performance results are shown in Table 4.1. The graph of the mean distance at every block is in Figure 4.1. Expansion of the RF and ERF for individual runs with different numbers of permutations is shown in Figure 4.2.

**Discussion.** It was expected to see a rapid increase of the RF already on lower layers, and it was confirmed that the higher number of permutations, the faster RF increases. Graph 4.1 also correspond with our hypothesis. The context was added, RF increased, and the network attended a wider area. Performance, however, did not increase. With more permutations, the performance was slightly getting worse. It follows that the network was probably overloaded with too much context. On top of that, no positional information was supplied in this experiment.

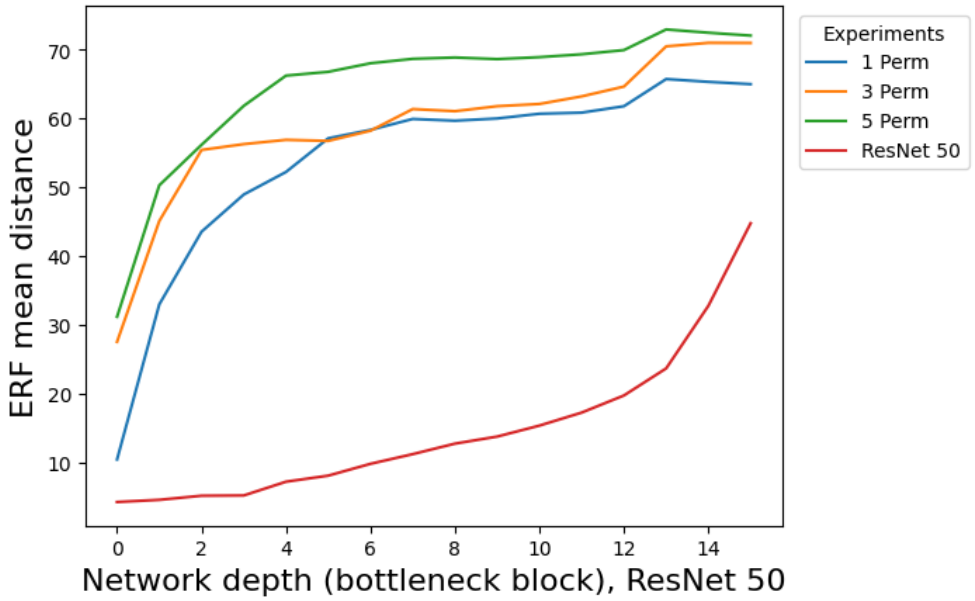



**Figure 4.1.** Mean distance at every bottleneck block for random sparse context method.

## 4.4 Random sparse context on higher layers

To prevent possible network overloading by context, we conducted a short experiment supplying considerably less context. Only one permutation was used and was added just in the third and fourth layers. This leaves the network to focus on the local features in the early stages. Once it already attends a wider area, a context is added. This can be seen in Figure 4.4 that the RF is starting to get extended first in the eighth block. Performance results are in Table 4.2. The results of the mean distance growth compared with plain ResNet 50 and one permutation from the previous experiment are plotted in graph 4.3.

**Discussion.** Based on a slight performance increase, we can say that adding context only on higher spatially-narrow layers is beneficial. The network was attending more limited areas until the eighth block, which nicely corresponds with the mean distance of ERF shown in Figure 4.3. It started to increase just as the context was supplied. The positional encoding still was not added.

## 4.5 Augmenting the context with relative positional encoding

This experiment extends the very first experiment with the random sparse context. We tested the influence of positional encoding on the added context to the NN. A relative positional encoding is used according to Section 3.2. It contains the encoded relative distances of the permuted data tensor from the

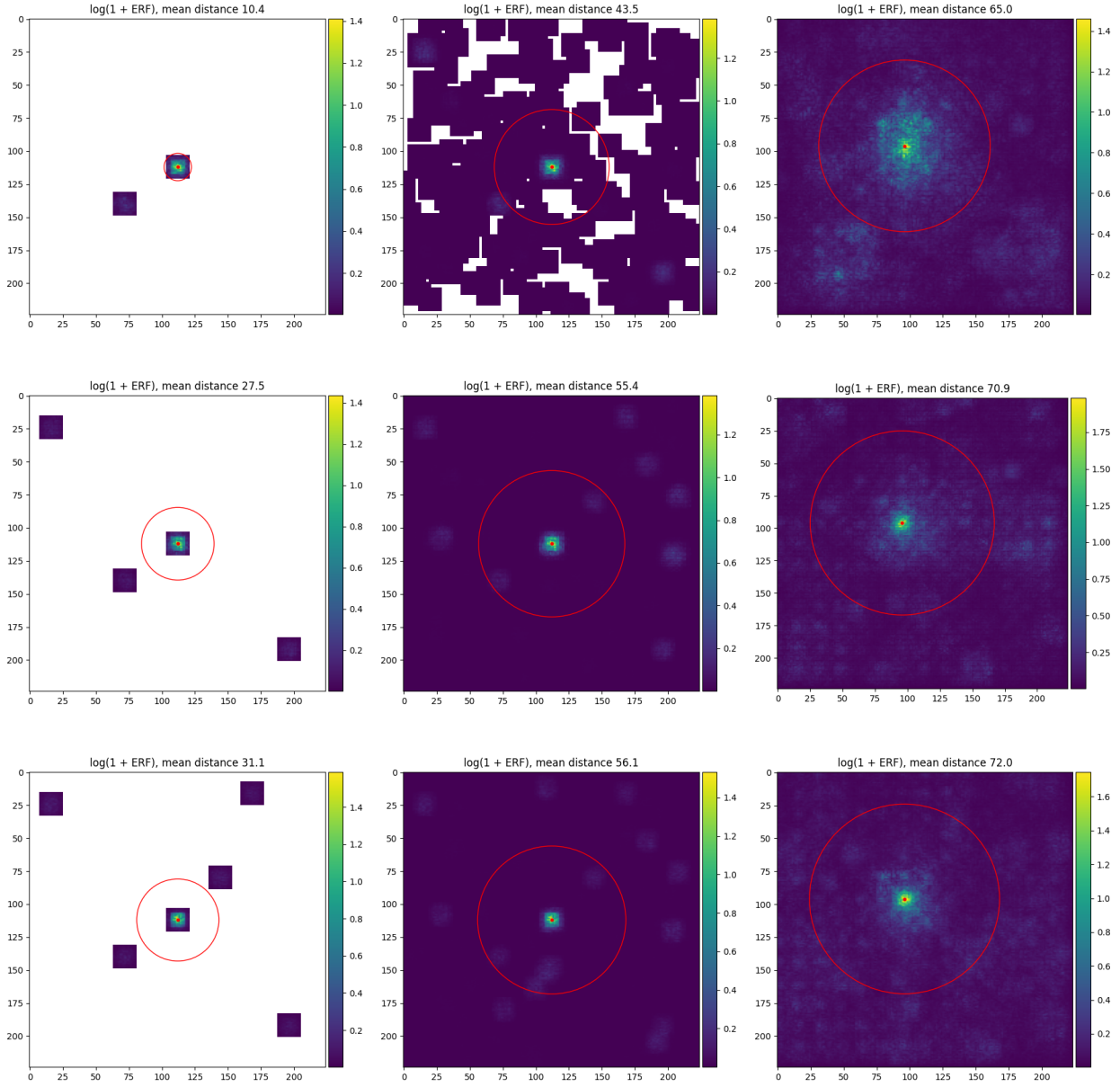



**Figure 4.2.** RF and ERF on the first (1st col.), third (2nd col.) and the last (3rd col.) bottleneck block with 1/3/5 permutations (in rows respectively).

| Architecture | Top 1 accuracy (in %) | params $\times 10^6$ |
|---|---|---|
| ResNet 50 | 90.42 | 23.5 |
| ResNet 50, 1 Permutation all layers | 90.52 | 28.5 |
| ResNet 50, 3 Permutations layers 3,4 | 90.85 | 28.3 |

**Table 4.2.** Performance results on the validation set of the random sparse context with one spatial permutation at different layers compared with the ResNet 50.

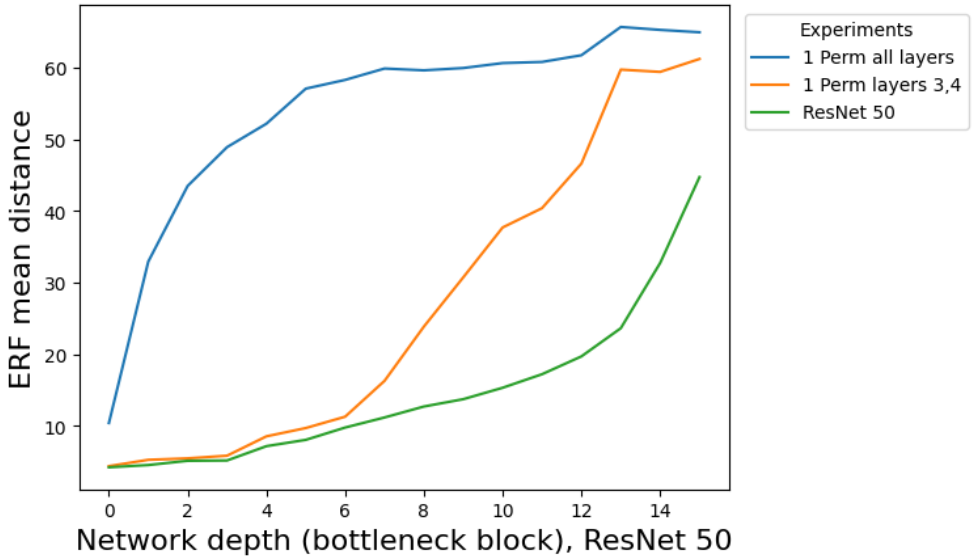



**Figure 4.3.** Mean distance at every bottleneck block for one spatial permutation at different layers.

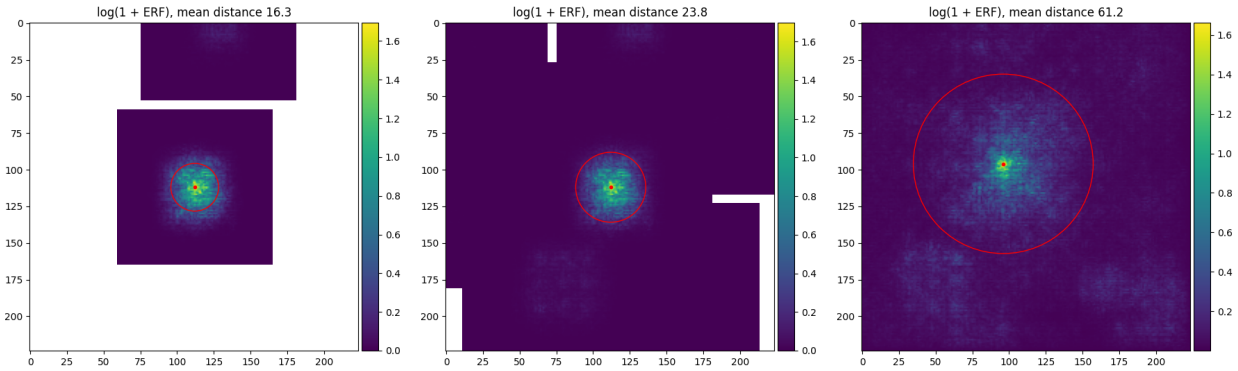



**Figure 4.4.** RF and ERF on the eighth, ninth and the last bottleneck block with only one permutation.

original data tensor giving the network a clue, which we expect to be beneficial. The tensors with relative positional encodings are accordingly summed with the permuted tensors. We again tried one, three and five permutations. Table 4.3 shows the performance results and the graph 4.5 growth of the mean distance compared with the plain ResNet 50.

**Discussion.** We observed that even if positional encoding was added, it was no help in terms of performance. When compared with permutations without encoding, there was even a slight performance decrease. We concluded that positional encoding for this method is more confusing than beneficial to the neural network. The course of the mean distance values shows a similar trend as in previous experiments. Figure 4.6 shows RF and ERF at different blocks of the network for various numbers of permutations, which is very similar to the Figure 4.2, showing only permutations without encoding. It can be inferred that the network tries to ignore the positional information.

| Architecture | Top 1 accuracy (in %) | params $\times 10^6$ |
|---|---|---|
| ResNet 50 | 90.42 | 23.5 |
| ResNet 50, 1 Permutation + Enc | 90.14 | 28.5 |
| ResNet 50, 3 Permutations + Enc | 89.81 | 38.6 |
| ResNet 50, 5 Permutations + Enc | 89.43 | 48.7 |

**Table 4.3.** Performance results on the validation set of the context with relative positional encoding added, compared with the ResNet 50.

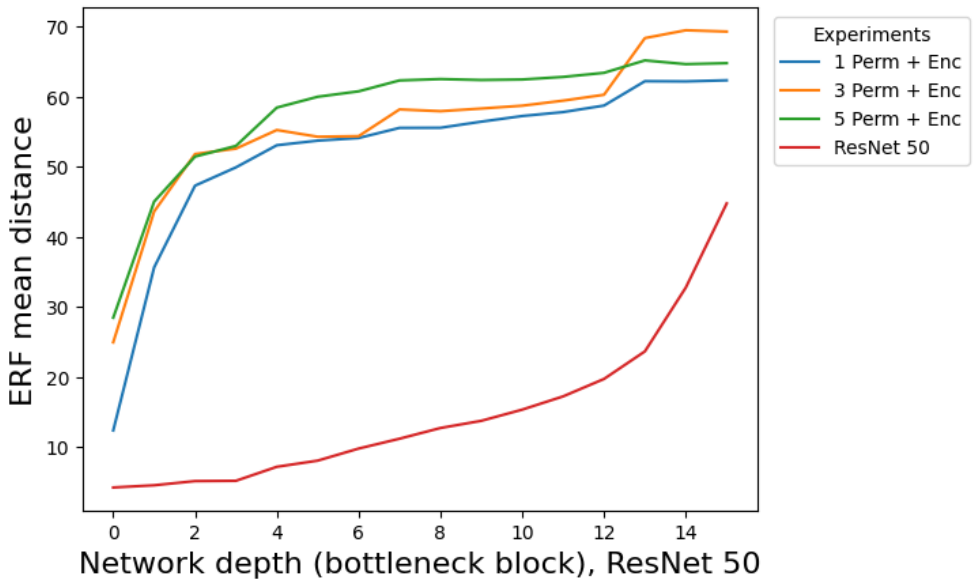



**Figure 4.5.** Mean distance at every bottleneck block for augmenting the context with relative positional encoding.

## 4.6 Adding Attention

Not every feature value contributes the same to the overall context and vice versa. This experiment evaluates the approach to the task of adding context inspired by the attention function used in ViT [7]. The method of the modification is described in Section 3.3. The performance results of the experiment are in Table 4.4, and the plotted course of the mean distance is in Figure 4.7.

**Discussion.** The results showed the same trend as in previous experiments. Increasing the number of permutations generally provided a higher mean distance, even though the one permutation reached a little lower value at the end of the network compared to the ResNet 50. However, regarding performance, it decreased with more permutations. Figure 4.8 shows that the ERF was more suppressed, meaning it did not spread as much compared to the experiment with only spatial permutations. On the other hand, the network in this experiment had the same number of trainable parameters as the original ResNet 50.

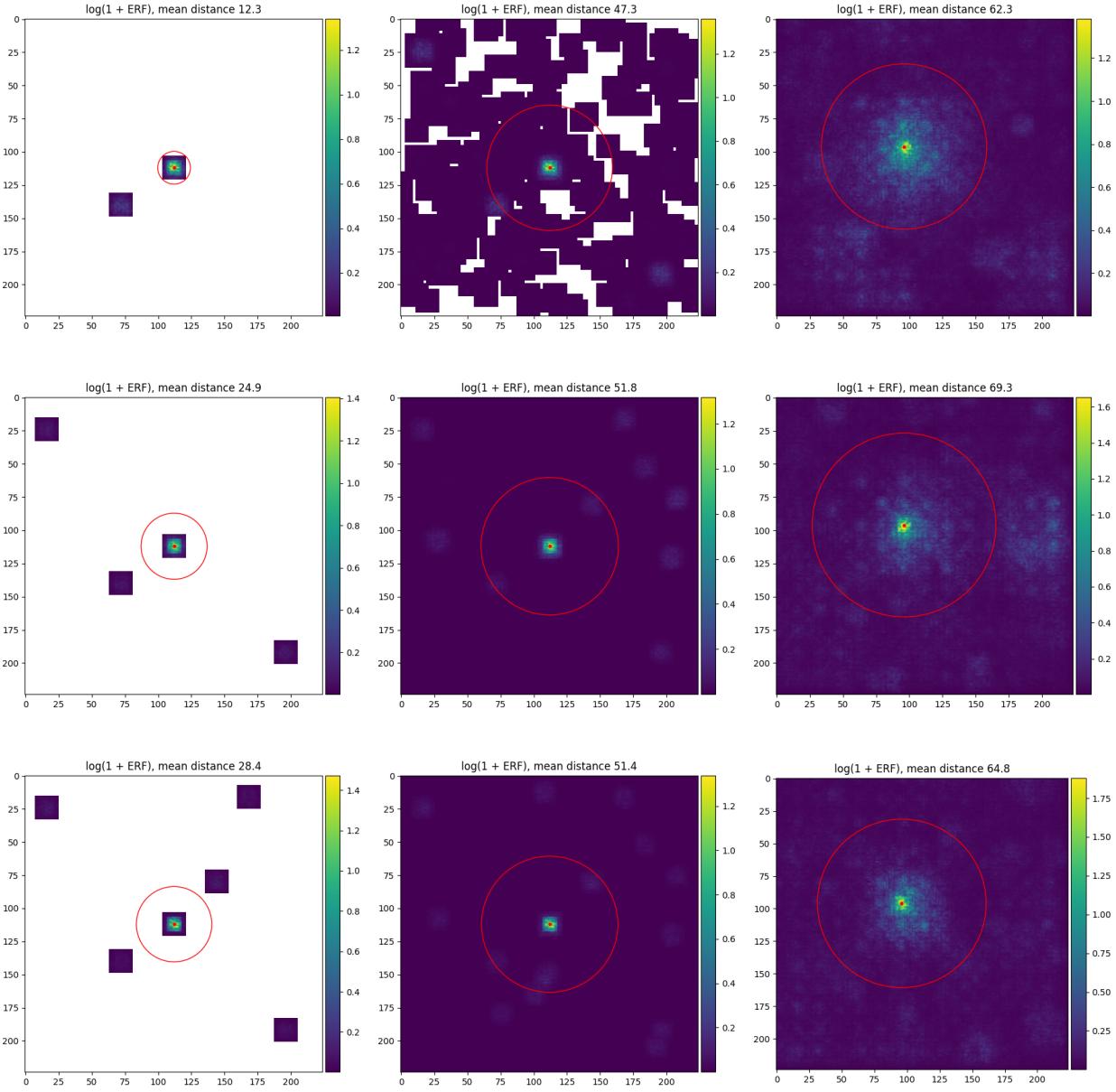



**Figure 4.6.** RF and ERF on the first (1st col.), third (2nd col.) and the last (3rd col.) bottleneck block with 1/3/5 permutations with relative positional encoding (in rows respectively).

| Architecture | Top 1 accuracy (in %) | params $\times 10^6$ |
|---|---|---|
| ResNet 50 | 90.42 | 23.5 |
| ResNet 50, 1 Permutation with Attention | 90.14 | 23.5 |
| ResNet 50, 3 Permutations with Attention | 89.83 | 23.5 |
| ResNet 50, 5 Permutations with Attention | 89.76 | 23.5 |

**Table 4.4.** Performance results on the validation set of the added attention function, compared with the ResNet 50.

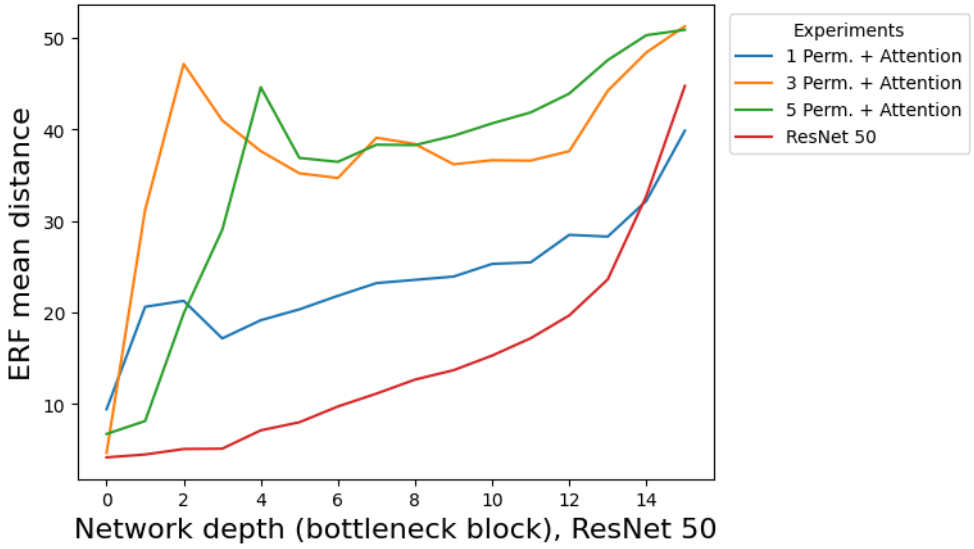



**Figure 4.7.** Mean distance at every bottleneck block for the added attention function.

| Architecture | Top 1 accuracy (in %) | params $\times 10^6$ |
|---|---|---|
| ResNet 50 | 90.42 | 23.5 |
| ResNet 50, modified dilated $d = 4$ | 87.26 | 23.5 |
| ResNet 50, inserted dilated $d = 4$ | 90.78 | 34.8 |
| ResNet 50, inserted dilated $d = 8$ | 89.81 | 34.8 |
| ResNet 50, random dilated $d = 16$ | 90.55 | 63.7 |
| ResNet 50, random dilated $d = 8$ | 91.09 | 63.7 |

**Table 4.5.** Performance results on the validation set of different experiments with dilated convolution.

## 4.7 Experiments with dilated convolution

In this set of experiments, the dilated convolution was used in its standard form and also modified and implemented in ResNet 50. Several approaches were tested. First, we modified the 3×3 convolution in every bottleneck block by dilating it with $d = 4$.

The next two runs also tested a dilated convolution; however, instead of replacing/modifying the original one, we inserted the 3×3 dilated convolution into the architecture according to Section 3.4.1. For the first run of the two, $d = 4$ and for the second, $d = 8$.

Lastly, randomised dilated convolution was tested (see Section 3.5). Same as previously, two runs were done, first with parameters $d = 16$ and $k = 8$ and the second with $d = 8$ and $k = 8$. The random dilated convolution was placed again after the 3×3 and before the 1×1 convolutions in every bottleneck block. For the results to be comparable, we always used $k = 8$

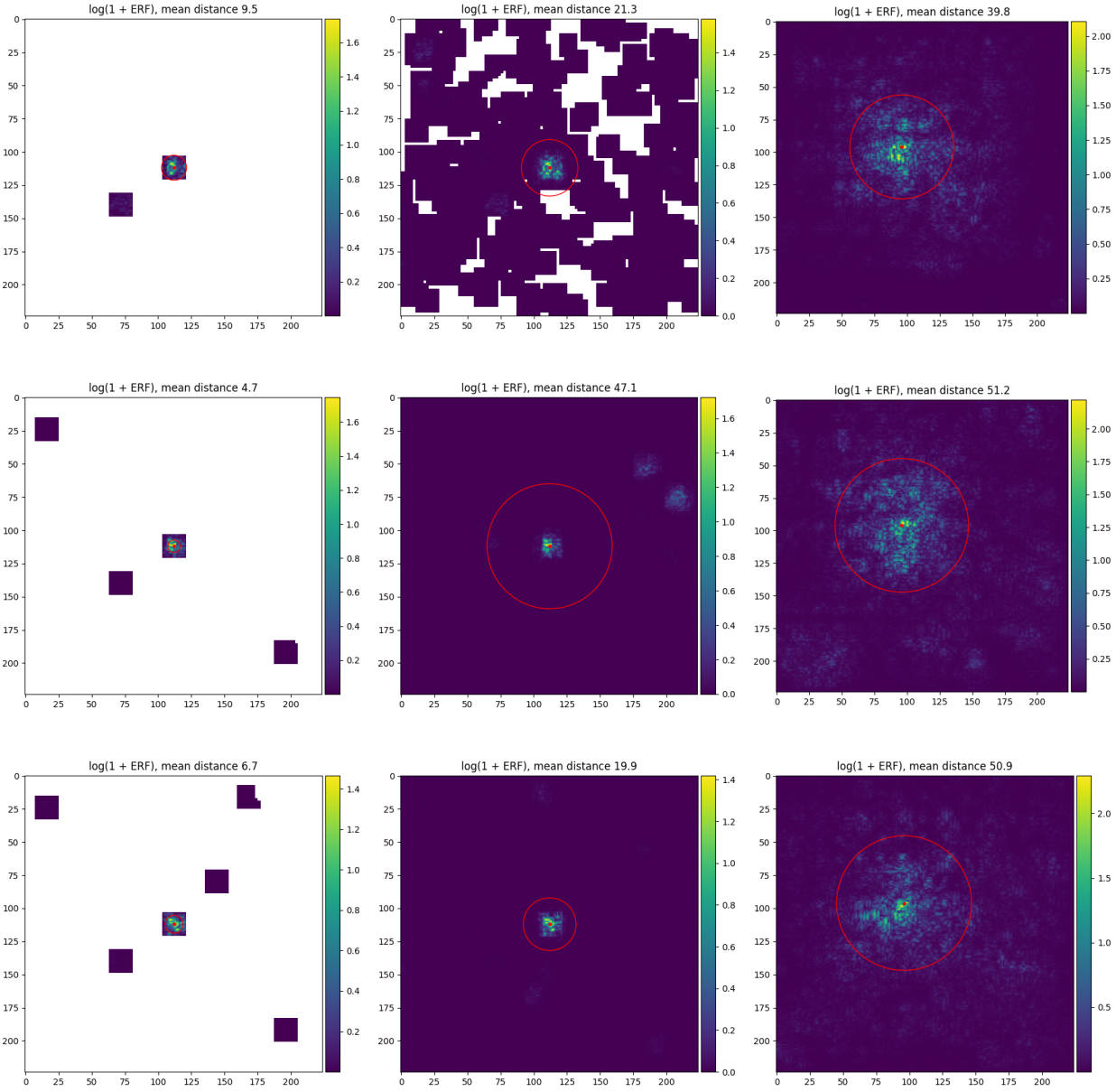



**Figure 4.8.** RF and ERF on the first (1st col.), third (2nd col.) and the last (3rd col.) bottleneck block with 1/3/5 permutations with the attention function (in rows respectively).

so that the convolutional kernel had the same number of filter positions as the 3×3 convolution. Performance results of individual runs are in Table 4.5, compared with the plain RasNet 50. Graphs of the mean distances are in Figure 4.9.

**Discussion.** Only dilating the 3×3 convolution has proven to decrease the performance, which was expected. This experiment was shown to demonstrate that even though the context was added and the mean distance of the ERF was also significantly higher than the original architecture, it is still crucial to incorporate local context. Figure 4.10 shows the isolated RF areas which prevented the incorporation of the local features. The inserted dilated convo-

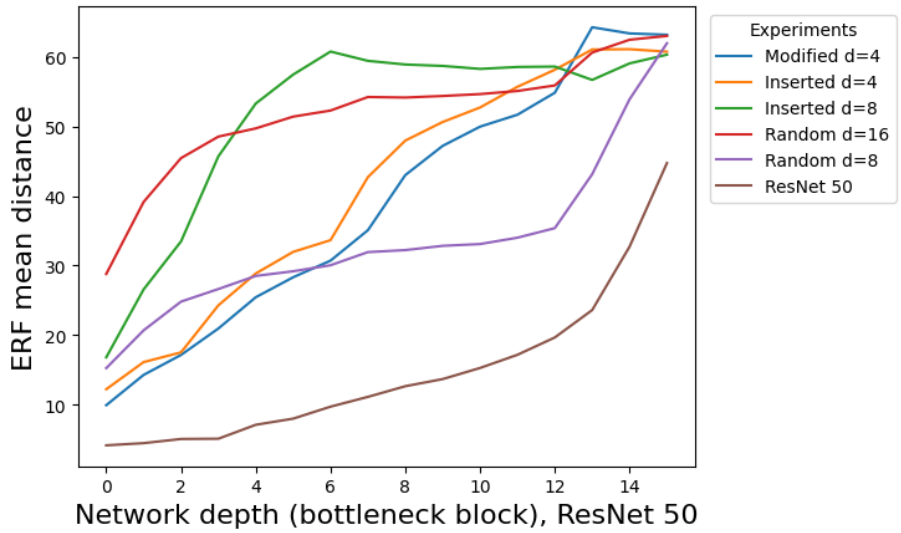



**Figure 4.9.** Mean distance at every bottleneck block for various experiments with dilated convolution.

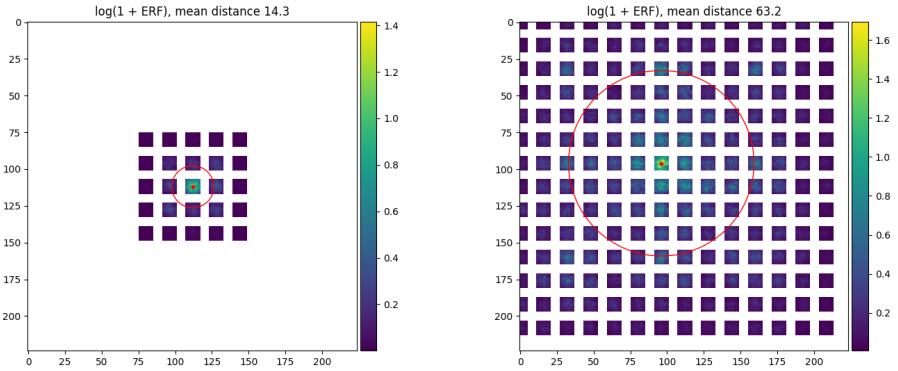
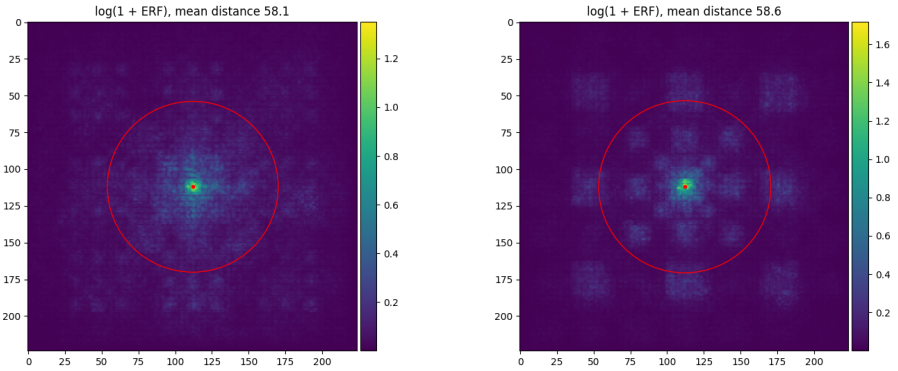



**Figure 4.10.** Isolated areas of the RF are shown at the second and the last block in ResNet 50, caused by dilating the 3×3 convolution with $d = 4$.




a) b)

**Figure 4.11.** Comparison of the artefacts of the inserted dilated convolution. a) shows the dilated convolution with $d = 4$ and b) with $d = 8$. It is apparent that the bigger parameter $d$, the more are the artefacts visible.

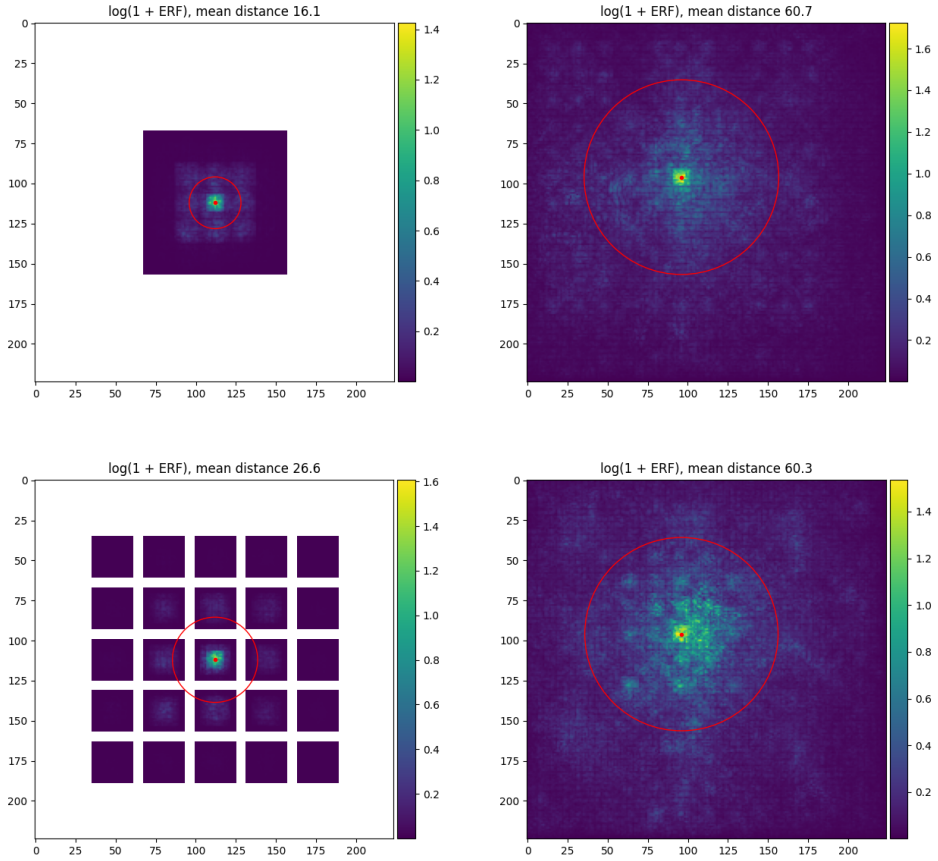



**Figure 4.12.** RF and ERF of the inserted dilated convolution on the second (1st col.) and the last (2nd col.) bottleneck block. First row shows dilation with $d = 4$ and the second row $d = 8$.

lution has shown a slight performance increase for smaller $d$. For the higher dilation factor, performance dropped. It was probably caused by the isolated artefacts created by the too dilated convolution. The artefacts can be well seen in Figure 4.11. Randomised convolution with $d = 8$ performed the best out of all methods at the cost of a large increase in the number of parameters. Again dilating too far has shown a performance decrease. Figure 4.12 visualises RF and ERF for the inserted and Figure 4.13 for the randomised convolutions.

## 4.8 Comparison of the ResNet 50 with other architectures

In the last experiment, we compared several neural network architectures in terms of ERFs with the ResNet 50 network. We selected the HRNet [1], described in Section 2.3, and the ConvNext [3], which both performed better than ResNet 50 in the classification task on the Imagenette dataset [9]. These are all CNN architectures, ConvNext being a network introduced this year. We also included the Vision Transformer [7] in the experiment. While not

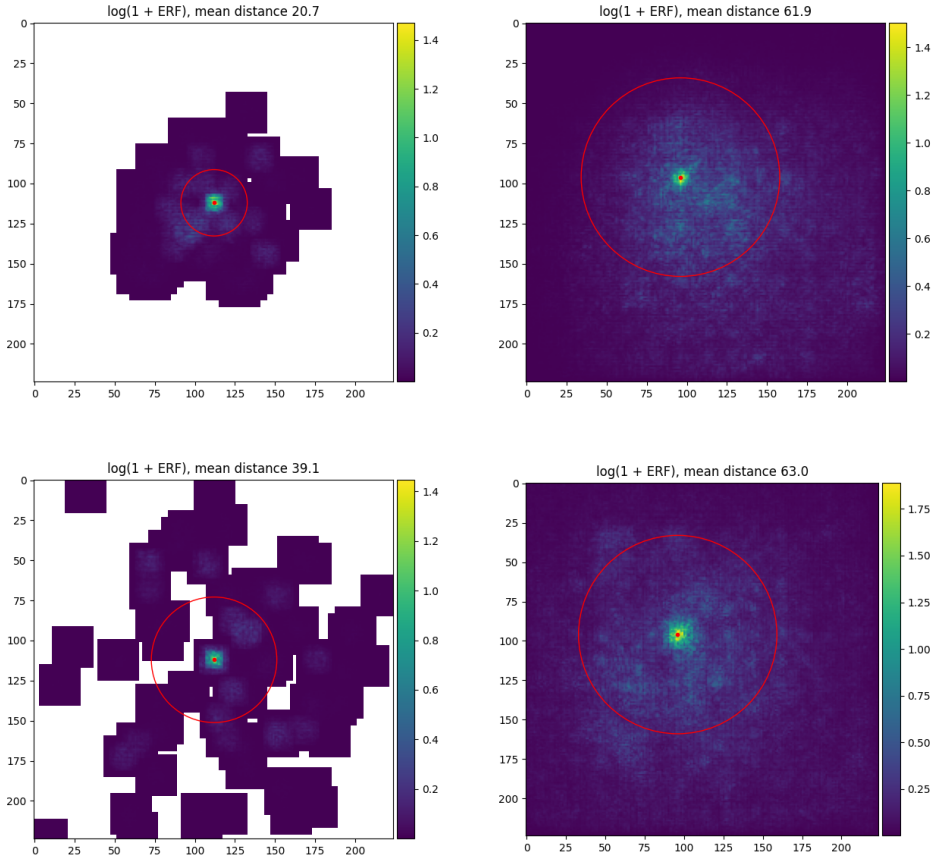



**Figure 4.13.** RF and ERF of the random dilated convolution on the second (1st col.) and the last (2nd col.) bottleneck block. First row shows dilation with $d = 8$ and the second row $d = 16$. In both cases was $k = 8$.

| Architecture | Top 1 accuracy (in %) | params $\times 10^6$ |
|---|---|---|
| ResNet 50 | 90.42 | 23.5 |
| HRNet | 91.62 | 21.3 |
| ConvNext | 94.80 | 28.6 |
| Vision Transformer | 84.99 | 85.6 |

**Table 4.6.** Performance results on the validation set of the Imagenette dataset.

convolutional, it has been an inspiration in our context expansion methods and has also performed well in benchmarks on datasets like Imagenet [17].

The performance results are listed in Table 4.6. Mainly we were interested in how these architectures use the context. Therefore the ERFs were plotted from the centre of the final feature map for every architecture. These can be seen in Figure 4.14. Finally, we visualised the sum of ERFs over all the spatial positions in the feature map for all architectures. These can be seen in Appendix B in Figures B.1-4, showing which areas of the input image contributed to the final feature map and by what amount.

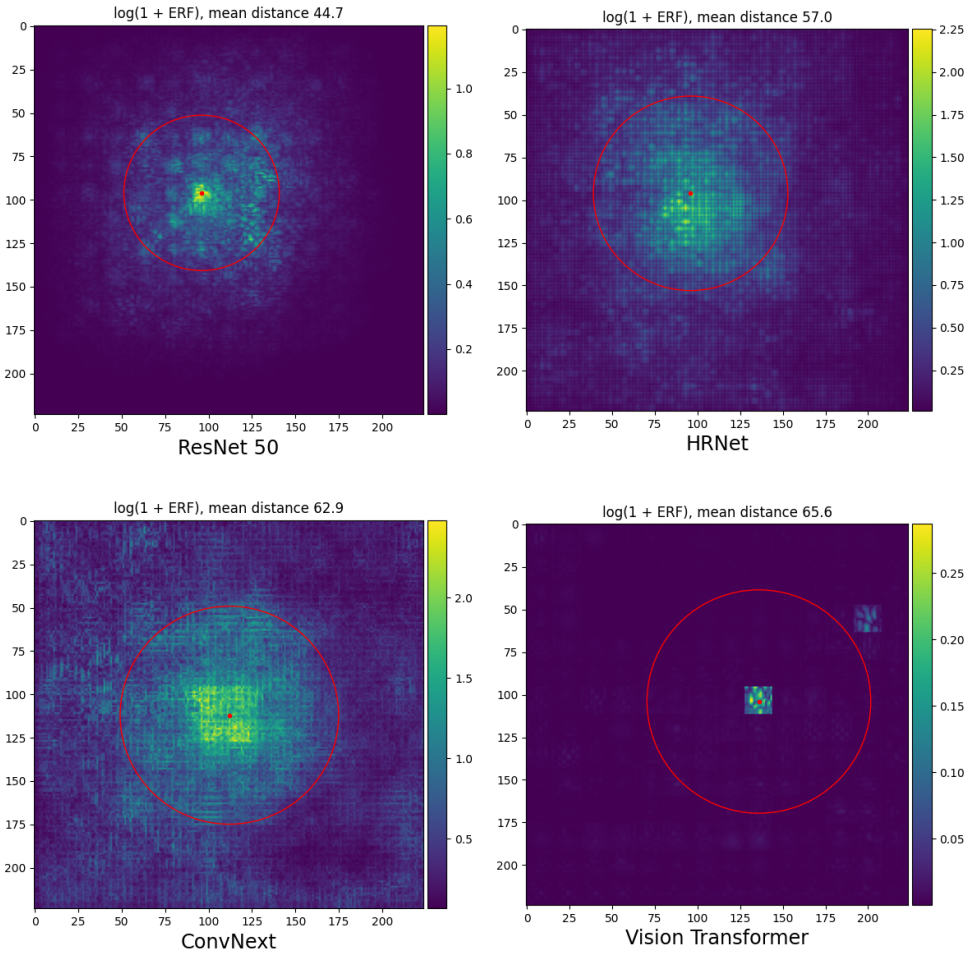



**Figure 4.14.** Visualization of the ERF for the ResNet 50, HRNet, ConvNext and the Vision Transformer.

**Discussion.** The performance results clearly show that ConvNext outperformed all the other architectures while having only slightly more trainable parameters than the other CNNs. Regarding the ViT, the accuracy was relatively low. ViT can and does outperform most of the CNN architectures [7]. However, it needs to be trained on very large datasets as it lacks the image-specific inductive biases inherent to the CNNs. When we compare the ERFs shown in Figure 4.14, it can be seen that ConvNext and HRNet attend way larger areas than the ResNet 50. Still, most focus is concentrated around the centre point (red dot in Figure 4.14) of the ERF in CNNs; however, it shows that the broader outreach is beneficial. The ViT shows very little ERF, even though the mean distance is significant, except for a single square from which the ERF was computed. The square corresponds to a single patch which attends to a few others just sparsely but mainly to itself.

# Chapter 5
# Conclusion

In this thesis, we proposed several methods for increasing context in convolutional neural networks. We tried two main approaches. The first was based on the generation of random context in order to achieve sparse connections between individual feature vectors. The second was utilising the dilated convolution.

Using the random context was a relatively simple method that led to increased context. This approach was then extended by using relative positional encoding for identifying new connections between feature vectors as local or distant. For the last method with random context, we implemented the attention mechanism to assess and reweight the significance of the newly created relationships. The inspiration came from the Vision Transformer, where self-attention is the main building block of the architecture.

Dilated convolution is defined by the dilation parameter, which states how far away the individual filter values are from each other in the grid. It reaches further away from the centre of the convolution, which extends the context. The downside of the dilated convolution is the high dilation parameter as it no longer incorporates local context and creates artefacts in the input image. We proposed a dilated convolution with a randomised kernel shape to prevent the artefacts while extending the context.

We have shown how to measure the context used by the network. In convolutional networks, we used the receptive field as a quantitative indicator, showing the area in the input image attended for creating a new feature. Further, we said that not every part of the receptive field contributes the same. To evaluate the context qualitatively, we introduced an effective receptive field. As the effective receptive field is computed for a particular feature, we also came up with the mean distance measure. It showed how far the feature, for which the effective receptive field was computed, attends in the input image.

We evaluated our methods implemented in the ResNet 50 network on the problem of classification into ten classes. We compared the use of the context with the Vision Transformer, HRNet and ConvNext architectures.

The results showed that all our methods successfully added context into the network. The receptive field contained the whole input area already after a few convolutional layers. Performance-wise however, no significant increase was registered.

The approach with the random sparse context showed that the more permutations (context) we added, the worse the performance was. This was the same for the random sparse context with relative positional encoding and also the attention approach. It was likely that the network was overloaded with context. So we tried adding the context later in the network on higher layers. It showed a slight increase in performance, however again, not significantly. We listed the experiment only with random sparse context as it showed the most promising improvement. Though we ran the experiment for the relative positional encoding and the attention, the improvement was insignificant.

The experiments with dilated convolution showed that the local context is still more important because the performance drops rapidly without it. Combining the standard and dilated convolutions saw a slight increase in performance, though only for smaller dilation parameter d. The best performance increase was achieved by the randomised convolution, in which accuracy was 0.67 % higher than the plain ResNet 50 at the cost of several times more parameters.

Lastly, the plain ResNet 50 was compared with Vision Transformer, HRNet and ConvNext architectures in terms of accuracy and effective receptive field. It showed that the better performing convolutional architectures, which were HRNet and ConvNext, could utilise the context better. The context utilisation for Vision Transformer was surprising as it showed very little attention to other parts of the image.

We still consider the context to be an essential part of increasing the network's performance. Potential research could focus on finding the right amount of context so the network is not overloaded. Also, adding the context into different parts of the architecture could lead to better results.

# References


[1] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. *Deep High-Resolution Representation Learning for Visual Recognition*. 2019. `https://arxiv.org/abs/1908.07919`.

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *CoRR*. 2016, abs/1606.00915

[3] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. *CoRR*. 2022, abs/2201.03545

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

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*. 2017,

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*. 2017, 30

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and others. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*. 2020,

[8] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*. 2016, 29

[9] Jeremy Howard, and Kerem Turgutlu. *FASTAI/imagenette: A smaller subset of 10 easily classified classes from Imagenet, and a little more French*. `https://github.com/fastai/imagenette`.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, 25

[11] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *CoRR*. 2016, abs/1611.05431

[12] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander J. Smola. ResNeSt: Split-Attention Networks. *CoRR*. 2020, abs/2004.08955

[13] Shanda Li, Xiangning Chen, Di He, and Cho-Jui Hsieh. Can Vision Transformers Perform Convolution?. *CoRR*. 2021, abs/2111.01353

[14] Jaesin Ahn, Jiuk Hong, Jeongwoo Ju, and Heechul Jung. *Rethinking Query, Key, and Value Embedding in Vision Transformer under Tiny Model Constraints*. 2021.
`https://arxiv.org/abs/2111.10017`.

[15] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*. 2018,

[16] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. *A real-time algorithm for signal analysis with the help of the wavelet transform*. 1990.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. *ImageNet: A large-scale hierarchical image database.* In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009. 248-255.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*. 2015, abs/1502.01852

# Appendix A
# Abbreviations

| | | |
|---|---|---|
| CNN | ■ | Convolutional Neural Network |
| ERF | ■ | Effective Receptive Field |
| HRNet | ■ | High-Resolution Network |
| MLP | ■ | Multilayer Perceptron |
| NN | ■ | Neural Network |
| ReLU | ■ | Rectified Linear Unit |
| RF | ■ | Receptive Field |
| SGD | ■ | Stochastic Gradient Descent |
| ViT | ■ | Vision Transformer |

# Appendix B
# ERF visualizations

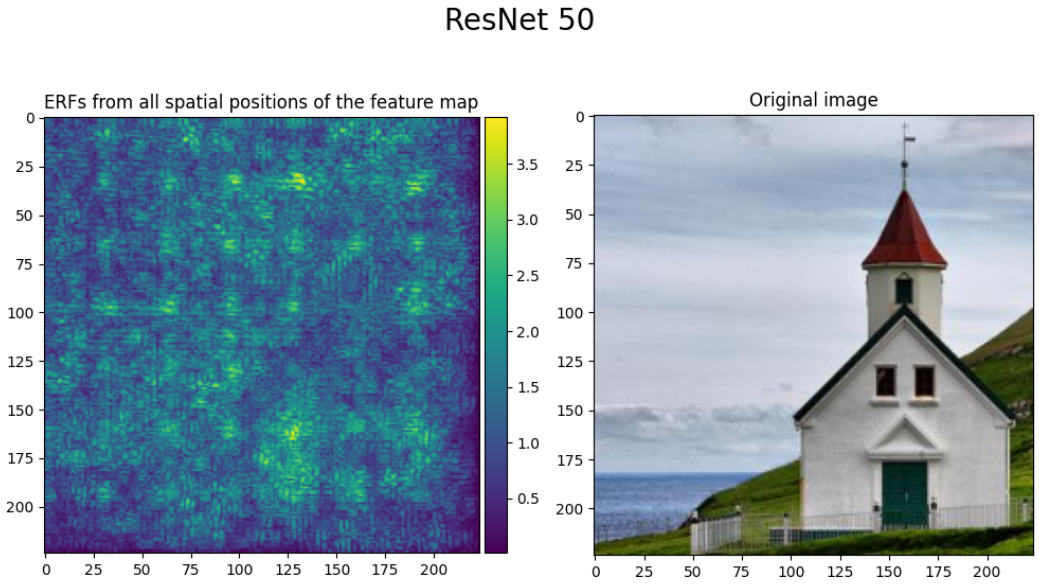



**Figure B.1.** Summed ERFs over all spatial positions of the final feature map for the ResNet 50.

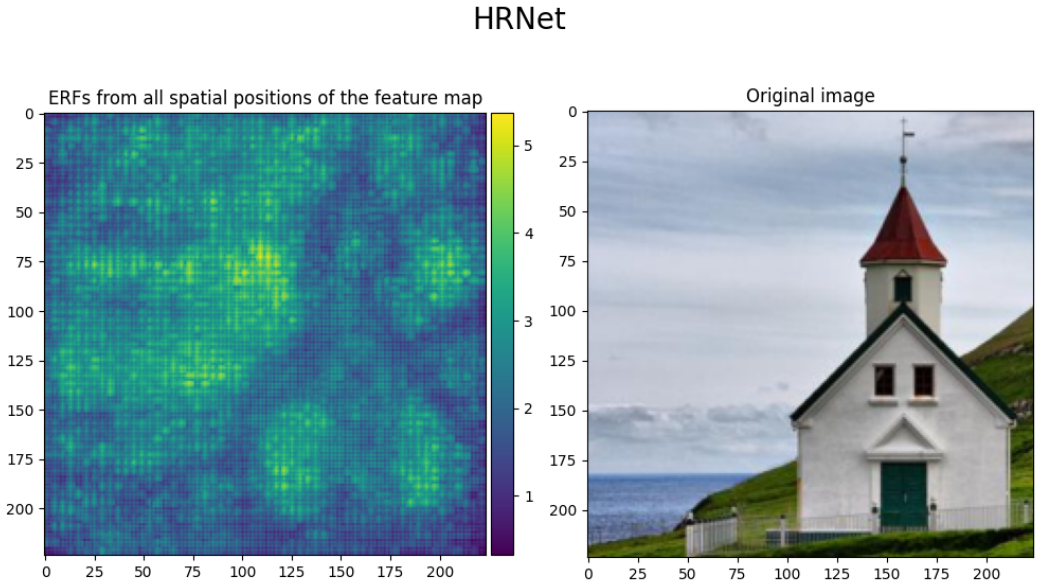



**Figure B.2.** Summed ERFs over all spatial positions of the final feature map for the HRNet.

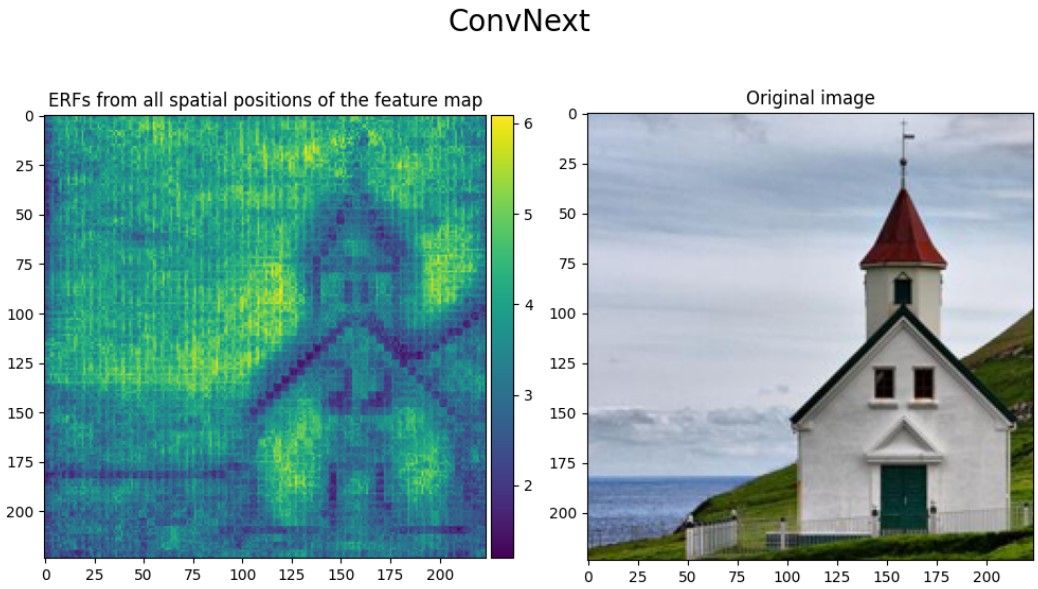



**Figure B.3.** Summed ERFs over all spatial positions of the final feature map for the ConvNext.

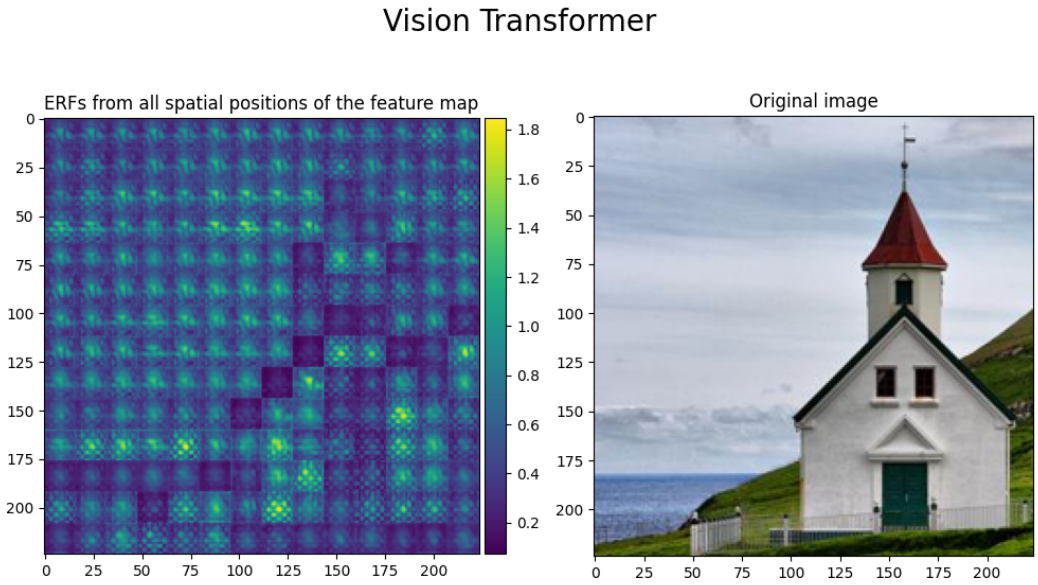



**Figure B.4.** Summed ERFs over all spatial positions of the final feature map for the Vision Transformer.