Parametric studies of translation invariance and distortion robustness in Convolutional Neural Networks

Johannes Christiaan Myburgh

 **orcid.org 0000-0002-7378-4796**

Dissertation accepted in fulfilment of the requirements for the degree Master of Engineering in Computer and Electronic Engineering at the North-West University

Supervisor: Prof. M.H. Davel

Graduation: June 2021

Student number: 25878360

Declaration

I, Johannes Christiaan Myburgh hereby declare that the dissertation entitled "Parametric studies of translation invariance and distortion robustness in Convolutional Neural Networks" is my own original work and has not already been submitted to any other university or institution for examination.

J.C. Myburgh

Student number: 25878360

Signed on the 5th day of December 2020 at Potchefstroom.

Abstract

Although Convolutional Neural Networks (CNNs) are widely used, their translation invariance (ability to deal with translated inputs) is still subject to some controversy. We explore this question using translation-sensitivity maps to quantify how sensitive a standard CNN is to a translated input. We propose the use of cosine similarity as sensitivity metric over Euclidean distance, and discuss the importance of restricting the dimensionality of either of these metrics when comparing architectures.

Our main focus is to investigate the effect of different architectural components of a standard CNN on that network's sensitivity to translation. To study the effects of max-pool kernel size on translation invariance, we train several CNN architectures with differently shaped max-pool kernels and compare their translation invariance. The results indicate that larger max-pool kernels result in more translation invariance than smaller max-pool kernels.

By varying convolutional kernel sizes and amounts of zero padding, we control the size of the feature maps produced, allowing us to quantify the extent to which these elements influence translation invariance. We also measure translation invariance at different locations within the CNN to determine the extent to which convolutional and fully connected layers, respectively, contribute to the translation invariance of a CNN as a whole. Our analysis indicates that both convolutional kernel size and feature map size have a systematic influence on translation invariance. We also see that convolutional layers contribute less than expected to translation invariance, when not specifically forced to do so.

The effects of various CNN components on distortion-sensitivity is also analysed in this study. We analyse the differences between how CNNs deal with translation and distortion. Using distortion-sensitivity functions that we define, we are able to quantify how sensitive a system is to distorted inputs. The results indicate that larger max-pool kernels result in more distortion-sensitivity for CNNs trained on MNIST, similar to translation invariance. Convolutional kernel size has less of an effect on distortion sensitivity for CNNs trained in MNIST while all networks, regardless of their architectural variations, learn to be less

sensitive to distortion when trained on CIFAR10.

All in all, it seems that convolutional layers are not fully utilised to deal with spatial information if the training task is not difficult enough, forcing the fully connected layers (spatially unaware) to compensate by learning similar elements at different inputs. By reducing feature map size to 1, forcing the convolutional layers to better deal with translation, we obtain the most translation invariant system studied, when evaluated on the unseen test set. Also, we observe a few similarities in how CNNs deal with translation and distortion and attribute these similarities to the network's ability to pinpoint important features in general, rather than specifically the movement of kernel across the input during convolution.

Keywords: *Convolutional Neural Networks, Translation invariance, Distortion robustness, Translation-sensitivity map, Data augmentation*

Contents

List of Figures

List of Tables

List of Acronyms

ReLU rectified linear unit

MLPs multilayer perceptrons

CNNs convolutional neural networks

Chapter 1

Introduction

1.1 Background

For some time it was believed that Convolutional Neural Networks (CNNs) are able to accommodate translated inputs easily because of the inherent abilities of their architectures, with later work [1]–[4] showing this not to be the case. A system is translation-invariant when it produces the same output, regardless of input shift. CNNs are commonly used for classification tasks requiring the classification of multiple images. These CNNs then learn to detect specific features in the input that allow the CNN to classify an image. Although CNNs are able to work quite well with complex and large inputs with multiple different classes, they are still sensitive to small translations as shown by Azulay and Weiss [4].

In work done by Kayhan and van Gemert [1] they show that CNNs are not translation-invariant and that modern CNNs can exploit absolute spatial location by learning filters (convolutional kernels) that respond to particular absolute locations. These CNNs learn to abuse boundary effects when the convolutional kernels are applied to an input.

As is evident from earlier work on CNN translation invariance [1]–[4], it is clear that modern CNNs are not completely translation invariant. We thus use the term "translation-invariance" to refer to a system's sensitivity to translated inputs, rather than strictly

to whether there is a change or not. This means that a system can be more or less translation-invariant. To quantify a CNN's translation invariance, we use translation-sensitivity maps and radial translation-sensitivity functions similar to those proposed by Kauderer-Abrams. [2].

New CNN architectures are constantly produced that advance the state-of-the-art in terms of performance, but how these CNNs deal with translated input samples is still not fully understood. The different components of a CNN each plays a specific role in the functionality of a CNN, but how each of the different components contributes to translation invariance is still not clear.

While translation moves images within a canvas, elastic distortion warps and shifts features within an image as explained by Simard et al. [5], where they use elastic distortions to expand the existing MNIST dataset. Even though translation is a form of distortion, it is a very specific form that should be easy for CNNs to handle. We are therefore interested in comparing how different components of a CNN affect translation and distortion robustness, respectively.

1.2 Problem statement

It is widely believed that CNNs are capable of learning translation-invariant representations, but the mechanism behind this translation invariance is still not fully understood. A standard CNN architecture consists mainly of convolutional kernels, pooling kernels and fully connected layers. These different components each play a key role in how CNNs function, but their individual contributions to translation invariance is still unclear. While translation shifts an input in a canvas, distortion shifts and warps features in an image. How different standard CNN components influence the network's overall sensitivity to distortion is still unclear.

1.3 Project scope

Given the problem statement above, we restrict the scope of the research to a limited number of datasets and architecture types:

Datasets: The datasets selected for our research are MNIST [6] and CIFAR10 [7]. These datasets are popular in the field of CNN research as they contain large amounts of accurately labelled data with varying levels of complexity.

Architecture types: The study mainly focuses on standard convolutional neural networks and components such as pooling layers, convolutional layers and fully connected layers. With the rapid increase in the popularity of CNNs, many different architectural variations have arisen in the past few years. As it is near impossible to study all these architectural variations of CNNs, we focus on the components of standard CNNs as they are present in nearly all state-of-the-art CNNs.

Translation: This study investigates how sensitive CNNs are to translated inputs. Translations of between 0 and 10 pixels in the x-axis and the y-axis are used to study this.

Distortion: The study investigates the effects of various CNN components on distortion robustness. Elastic distortions of magnitudes between 0 and 9 generated with the Augmentor library [8], as discussed by Simard et al. [5], are used to study distortion sensitivity.

1.4 Research questions

With the aim of investigating the components that contribute to CNN translation invariance and distortion sensitivity, the following research questions are formulated:

- How do some of the main design choices of a standard CNN, such as max-pool kernel sizes, convolutional kernel sizes and feature map size, affect translation invariance?

- To what extent do convolutional layers and fully connected layers, respectively, contribute to the translation invariance of a standard CNN?
- How is translation invariance of CNNs similar to standard distortion sensitivity?

1.5 Objectives of the study

The objectives of the study include the following, considering the research questions:

- To determine how standard CNN design choices such as max-pool kernel sizes, convolutional kernel sizes and feature map sizes affect how sensitive a CNN is to translated inputs.
- To determine to what extent convolutional layers and fully connected layer, respectively, contribute to the translation invariance of a CNN as a whole.
- To better understand the difference between how standard CNNs deal with distortion and translation.

1.6 Research methodology

This study consists of applied, qualitative and exploratory research based on previous studies on CNN invariance. Additionally the study consists of experiments and the analysis thereof using a codebase developed in the research period. The following forms part of the study:

Literature review: Gain a thorough understanding of convolutional neural networks, focussing on invariance to translated data. Investigate recent work done on the factors contributing to invariance and the capability of CNNs to learn translation-invariant representations.

Codebase development: We develop a codebase that allows us to efficiently train and analyse CNNs on multiple datasets.

Experimental development: Using the developed codebase we train and analyse CNNs to achieve the following:

- Gain better understanding of the effects on various CNN components on translation invariance and distortion robustness.
- Characterise the effect and contribution of each component.

Results: The empirical results obtained from the experimental analyses are shown and discussed in detail.

1.7 Dissertation overview

This dissertation aims to identify and analyse different components of the standard CNN architecture that affect translation invariance. The dissertation consists of the following sections:

- Chapter 1 contains a brief introduction of the work done in the dissertation. The problem statement and research objectives are stated here as well.
- Chapter 2 discusses the relevant background information and related work done in the field.
- Chapter 3 explains the experimental setup, datasets and CNN architectures used throughout this study.
- Chapter 4 describes the translation invariance quantification metric we use to analyse CNN translation invariance.

- Chapter 5 investigates the effects of max-pooling and max-pooling kernel size on translation invariance.
- Chapter 6 investigates the effects of different convolutional kernel sizes on CNN translation invariance.
- Chapter 7 investigates how convolutional layers and fully connected layers, respectively, contribute to translation invariance.
- Chapter 8 explores how max-pool and convolutional kernel size affect how sensitive a CNN is to distorted data. Comparisons between distortion-sensitivity and translation invariance are also made in this sections.
- Chapter 9 summarises the research done in this study and discusses how the research objectives are met in this dissertation.

1.8 Publications

This study is based on work published by Myburgh et al. (*Tracking translation invariance in CNNs*) [9] accepted for presentation at SACAIR 2020 and publication in Communications in Computer and Information Science (LNCS sub-series CCIS), Volume 1342. Chapters 4, 5, 6 and 7 follow work presented in this article.

In addition, the current work also contributed to a second study: *Stride and translation invariance in CNNs* by Mouton et al. [10], also accepted for presentation at SACAIR 2020 and publication in Communications in Computer and Information Science (LNCS sub-series CCIS), Volume 1342.

Chapter 2

Background

2.1 Introduction

Convolutional Neural Networks (CNNs) have become the industry standard for many image classification tasks. In this section we aim to investigate the relevant background of CNNs and translation invariance. We discuss why CNNs have grown so much in popularity and list a few important milestones of the CNN timeline. We then focus on specific work done on translation invariance in CNNs and discuss aspects that we feel have not yet been addressed.

2.2 Literature study

With the focus of our study being CNN components and their contributions to translation invariance, we review relevant work done in the field as a start. In Section 2.2.1 we discuss important milestones in CNN history and discuss their contributions to CNNs as we know them today. In Section 2.2.2 we do an overview of work done on translation invariance related to CNNs.

2.2.1 CNN evolution

Although CNNs and machine learning in general have become very popular in the past decade, they have been around for quite some time. Due to technological advances in the past few years, we have been able to explore the true potential of CNNs.

Early CNNs

In 1968, Hubel and Wiesel released a paper [11] of their work done on the visual cortexes of cats and monkeys. They show that these cortexes contain neurons that respond to smaller regions of the visual field. They refer to these regions as receptive fields and find that neighboring neurons have overlapping receptive fields. Their work inspired the creation of the Neocognitron [12] introduced by Fukushima in 1980. The Neocognitron consists of convolutional layers containing filters and down-sampling layers that calculate the average of the activations in a region. With its hierarchical architecture containing convolutional and down-sampling layers, the Neocognitron is widely considered as one of the first variations of CNNs as we know them today.

LeNet5 [13], developed in 1989 by LeCun et al., was one of the first CNNs trained with backpropagation. The backpropagation algorithm trains the convolutional kernels from given input samples and was shown to perform much better than convolutional kernels designed by hand. Due to the fact that these convolutional kernels could be automatically learnt from input samples, the system was suited to a large range of image classification tasks. Although the work done with LeNet5 showed great promise, due to lacking computational power and data availability at the time, deep learning did not get much attention until 2010.

Increased computational power

In 2010 Ciresan et al. [14] were able to implement and train several multilayer perceptrons (MLPs) on GPUs. Using parallel processing, they found that they were able to speed up

the training process by a factor of 40, allowing them to train networks with much more parameters than previously possible.

Due to increased processing power and GPU implementation, in 2012 Krizhevsky et al. were able to train AlexNet [15], a large CNN with multiple convolutional and fully connected layers, and achieve state-of-the-art performance on the ImageNet [16] benchmark dataset. AlexNet has an architecture very similar to that of LeNet5, but contains many more layers and kernels. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [17] where the network achieved a top-5 error of 15.3%. The very next year, Zeiler and Fergus won the ILSVRC 2013 [18] with ZFNet [19] using a similar architecture to AlexNet with more fine tuned hyper-parameters. This highlighted the true potential of CNNs, but also the importance of hyper-parameter choice.

With the increase in popularity of CNNs, Simonyan et al. investigate the effect of depth on CNNs. They introduce VGGNet [20], a CNN with small 3x3 convolutional kernels and multiple consecutive convolutional layers. In their work they vary the number of convolutional layers (depth) over several VGGNet architectures and find that increased depth results in better general performance. One of the big contributions to the field by VGGNet is the insight that multiple smaller convolutional kernels in sequence can emulate the effect of larger receptive fields.

CNN architectural variations

As CNN performance continuously increased, a trend started to appear: if improved network performance is desired, increase their size. This would almost always result in better performance, but came with a major drawback. Increasing the size of a network generally increases the number of parameters which leads to overfitting and drastically increased computational cost. To address this problem, C. Szegedy et al. propose the use of inception modules in GoogleLeNet [21]. Although these inception modules increase the architectural complexity of a CNN, they drastically reduce the amount of required computation while still obtaining state-of-the-art performance. These inception modules can

be seen as a combination of parallel convolutional kernels, containing a 1x1 convolutional kernel that reduces the amount of parameters of the system. These 1x1 convolutional kernel layers are referred to as "bottleneck" layers.

With VGGNet that showed how more convolutional layers result in better performance and the introduction of bottleneck layers by GoogleLeNet, in 2015 K. He et al. introduced ResNet [22] that combines these ideas to produce a CNN 8 times deeper than VGGNet that is able to achieve state-of-the-art performance for lower complexity. These CNNs with depths of up to 152 layers won first place at the ILSVRC 2015 [23] achieving an impressive 3.57% error on the ImageNet test set. The ResNet architectures use residual blocks and skip connections to allow outputs to bypass two layers before being reintroduced. The intuition behind bypassing two consecutive layers is that two layers can be thought of as a small classifier within the network. Within these residual blocks they use bottleneck layers to reduce network complexity and the total number of parameters. In their work they show that the ResNet architecture can gain considerable accuracy from increased depth and is easier to optimise than traditional CNN architectures of similar size.

In 2016 G. Huang et al. introduced the DenseNet [24] architecture that contains multiple dense blocks. As with ResNet, these dense blocks have a sort of skip connection between layers in a block, but unlike ResNet, each layer within a dense block receives an input from each previous layer in the block. This means that each layer within a block is receiving a sort of "collective knowledge" from all preceding layers in the block. DenseNet architectures consist of several of these dense blocks and implement bottleneck layers that reduce the amount of parameters in the system.

Difficulty of modern-day CNN research

The race to achieve state-of-the-art performance has produced so many CNN architectural variations that it is near impossible to perform an in-depth study on each of them. Without fully understanding their functionality or contribution, components that achieved state-of-the-art performance at one point in time are included in nearly all future varia-

tions in the hope that they will help achieve the next state-of-the-art CNN performance. In our work we aim to take a step back and focus on the base components of CNNs. Although the architectures mentioned in this section have various levels of complexity, they all contain the same basic components of a CNN: convolutional layers, pooling layers and (in most cases) fully connected layers. Thus in our study we focus on these components in hope that our work can contribute to a better understanding of CNNs in general.

2.2.2 Translation invariance

For a system to be completely translation invariant, its output must not be influenced by any translation of the input [25]. The output of a translation-invariant system must thus remain identical for translated and untranslated inputs. Knowing that complete translation-invariance is not an inherent characteristic of standard CNN architectures [1], [2], we redefine the term "translation-invariance" to refer to a system's sensitivity to translated inputs. This means that a system can be more or less translation-invariant based on the values received from our translation sensitivity quantification metric.

Quantifying translation invariance

When investigating translation invariance, we require a performance metric that measures the magnitude of the effect when a sample is translated in a given direction: both direction and magnitude are important. To address this, Kauderer-Abrams [2] developed translation-sensitivity maps that can be used to visualise and quantify exactly how sensitive a network is to shifted inputs. These translation-sensitivity maps resemble a "heat-map" with brighter pixel representing high translation invariance and darker pixels representing low translation invariance. The main mechanic used to quantify translation invariance is comparing output vectors of a network given a translated and untranslated sample. The direction and amount of translation in represented by each pixel's location in the map. These translation-sensitivity maps are the main translation invariance quantification metric we use in our study and are discussed in depth in Section 4.2.1.

Kauderer-Abrams use these sensitivity maps to analyse translation invariance in CNNs given augmented training data. Although they study architectural components such as pooling and convolution, they find that these architectural choices have a secondary effect and that augmented training has the biggest influence on increasing translation invariance.

Inherent invariance

Due to moving convolutional kernels, many believe that convolutional layers are able to accommodate translated inputs easily. In work done by Kayhan et al. [1] they challenge this assumption by showing that CNNs have the capability to learn filters that use absolute spatial location. In their work they illustrate how convolutional kernels are able to manipulate border effects to ignore certain parts of an image, thus allowing them to encode absolute spatial location. Due to the large receptive fields of CNNs, they are able to exploit boundary effects quite far from the image border. Kayhan et al. propose the use of specific forms of padding that remove spatial location encoding which then increases translation invariance. They state that if spatial location is truly discriminative between classes, it should be used and not removed. This relates to photographer's bias [4], a phenomenon where objects appear in specific locations within an image due to the bias of the photographer, and motivates that it could aid in classification tasks. They also state that it is difficult to discern when CNNs exploit spurious location correlations due to a lack of data.

In work done by Azulay and Weiss [4] they show that standard CNN architectures do not achieve complete translation invariance, and they attribute it to CNN architectures ignoring the classical sampling theorem [26] during downsampling [4]. They also show that augmented training data does not lead to true translation invariance as the CNNs only learn to be invariant for samples very similar to the training set. To illustrate this they use several popular CNN architectures and test their sensitivity to 4 different, single pixel perturbation protocols (shift, embedding, embedding with inpainting, scale) on the ImageNet dataset. In all four protocols, the effects on the perturbation protocols are imperceptible to the human eye, yet fools modern CNNs 30% of the time. This

illustrates how brittle and sensitive these networks are to extremely small variations in input data. They also find that training a network on augmented training data only motivates the training algorithm to learn features that are invariant to translations present in the training set. In an attempt to achieve true translation invariance, they list 3 possible solutions: anti-aliasing, increased data augmentation and reduced subsampling.

Following the work done by Azulay and Weiss, Zhang [3] introduces anti-aliasing into the CNN architecture. They identify that max-pooling consists of two operations: the max operation and subsampling. They propose the use of a low-pass filter between the two operations and observe improved accuracy across several popular CNN architectures.

Effect of learning strategy on translation invariance

With many groups focussing their work on CNN architectures, Srivastava and Grill-Spector [27] focus their attention on the effect of learning strategy on translation invariance. They utilise transfer learning to study the effects of three different learning strategies: training from scratch, training after being initialised with a pre-trained model, and a pre-trained model with the last layer retrained keeping all other layers frozen. They study these learning strategies with 7 different CNN architectures over various amounts of variation in sample position, size, rotation and resolution. From their results they find that the most significant contributor to trasformation invariance is pre-training on a large and diverse image dataset. They also find that the initial kernels of the pre-trained models are seemingly unchanged by the fine-tuning process and resemble general filters such as edge-detectors. This suggest that unless the network is exposed to various spatially transformed samples in the training process, the network learns to identify local features over general features.

2.3 Conclusion

In this brief overview of the field we see an impressive increase in performance and accuracy of CNNs in the past decade. The first CNNs developed in 1989 showed great promise but it wasn't until 2010 when GPU implementation allowed us to witness the true power and potential of CNNs. As CNNs have become more popular over the past few years, we see more and more architectural variations arise each year. While these variations constantly achieve state-of-the-art performance, it becomes increasingly difficult to understand the contributions of the different components in these architectures.

One of the reasons for the immense growth in the popularity of CNNs is their capability to deal with transformed input data. Many studies have focused specifically on translation invariance, most of them being done on the popular, but more complex CNN architectures, such as VGGNet, ResNet, etc. The translation-sensitivity maps introduced by Kauderer-Abrams [2] allow us to study the effects that specific CNN components have on translation invariance. They also study the effects of augmented training data and find that it significantly increases translation invariance. This is then contradicted by the work of Azulay and Weiss [4] that indicates that CNNs simply memorise augmented samples and do not become completely translation invariant. Their results motivate us to study how CNN components contribute to inherent translation invariance of CNNs trained on non-augmented data. In the work done by Kayhan et al. [1] they show that CNNs are not completely translation invariant, motivating us to study general translation invariance in CNNs. They also find that convolutional kernels learn to use their absolute location during classification, motivating us to study how convolutional kernel size affects translation invariance.

Although these studies produce relevant and insightful results, the architectures used can make it difficult to truly identify components that contribute to translation invariance. Thus in our work we focus purely on the components of a standard CNN architecture, more specifically: convolutional layers, max-pool layers and fully connected layers. We choose these architectures as they are present in most CNN architectures, regardless of

complexity.

Chapter 3

Experimental setup

3.1 Introduction

In this section we discuss the datasets and architectures used in our work. We give a brief description of each and then elaborate on why we have decided to use them in our work. We briefly discuss the three main components of Convolutional Neural Networks (CNNs) and identify architecture patterns present in many popular and influential CNN architectures.

We first describe our experimental setup and the datasets we use in Sections 3.2 and 3.3, before discussing CNN components and architectures in Sections 3.4. Our optimisation protocol is explained in Section 3.5.

3.2 Experimental setup

This study consists of the analysis of CNNs and their components trained and tested on different datasets with varying levels of complexity. We first identify a component of a CNN that we wish to investigate. We then set up an experiment in such a manner that would allow us to investigate the chosen component over multiple parameters. Vari-

ations of a standard CNN architecture is then designed and trained using the datasets described in Section 3.3, and the protocol described in Section 3.5. The fully trained and optimised CNNs are then used to generate translation-sensitivity maps and radial translation-sensitivity functions as described in Sections 4.2.1 and 4.2.3. From these results we are able to determine the effects of the chosen component on the CNN's sensitivity to translated input samples. Once the results are thoroughly analysed, we discuss and report our findings.

3.3 Dataset preparation

In our work we use two common image classification datasets: MNIST and CIFAR10. As is common practice in supervised machine learning research [1]–[4], [27], we split our datasets into three parts, namely the train set, validation set and the test set. The train set is usually a large portion of the original dataset and is used to fit the model. These are the samples given to the network over several training epochs and are generally not used to evaluate the model's performance after training. The validation set is a small subset of the dataset that is frequently used to evaluate the model's performance. Although the samples in the validation set are used for model selection and evaluating the effect of hyperparameters, these samples are not present when training takes place. Hence the network does not "learn" these samples. The test set is a subset of samples used to generate a final evaluation of the model's performance after training. Similarly to the samples in the validation set, the test set samples are never present during training. The test set is only used when the model is fully trained. It is important to note that samples from the train, validation and test set are not translated during training or testing.

3.3.1 MNIST

The MNIST dataset [6] contains 70,000 samples of singular handwritten digits between 0 and 9 centred in a 28×28 pixel canvas. As previously mentioned, the dataset is ran-

domly split into a train set containing 55,000 samples, a validation set containing 5,000 samples and a test set containing 10,000 samples. In our analysis, we generate translation-sensitivity maps as discussed in Section 4.2.1. To generate these sensitivity maps, individual samples have their features shifted by a set amount of pixels within a canvas. To avoid the loss of features due to features being shifted outside the canvas, we zero pad all samples in the dataset with a 6-pixel border, as introduced by Kauderer-Abrams [2]. Example samples of MNIST can be seen in Figure 3.1.

3.3.2 CIFAR10

The CIFAR10 dataset [7] contains 32×32 pixel samples of different class colour images. The 10 classes contain various images of different animals and vehicles. The dataset is randomly split into a train set containing 45,000 samples, a validation set containing 5,000 samples and a test set containing 10,000 samples. As in Section 3.3.1, all samples are zero padded by a 6-pixel border to avoid the feature loss when generating sensitivity maps. Example samples of CIFAR10 can be seen in Figure 3.1.

3.3.3 Dataset complexity

Both the MNIST and CIFAR10 datasets have proven to be very popular in many image classification research problems. Although both these datasets contain a similar amount of samples with similar sizes, they have different levels of complexity. In Figure 3.1

Figure 3.1: CIFAR10 class samples compared to MNIST class samples. The top row contains ten samples of one class of the CIFAR10 dataset while the bottom row contains 10 samples of one class of the MNIST dataset.

one can see ten samples of a single class of both datasets. Whilst the MNIST samples (bottom row) have high similarity across different samples in a class, it is clear to see that the samples of CIFAR10 (top row) do not have the same level of inter-class similarity. The main features within the MNIST samples tend to be centred on the canvas while in the case of the CIFAR10 samples, the features seem to be much more distributed across the canvas. Due to these distributed features, CNNs trained on CIFAR10 learn to be more translation-invariant. Not only does the positioning of features within the CIFAR10 samples provide an extra level of complexity, background noise and multiple color-channels also make it more difficult for CNNs to fit this dataset than MNIST.

Due to these reasons, we tend to increase the number of channels per convolutional layer for CNNs trained on CIFAR10 to allow for extra capacity within the CNN. Although train set accuracies tend to be similar across both datasets, there is a difference of around 25% in validation set and test set accuracy between MNIST and CIFAR10. This gap in accuracy is desired as it allows us to investigate our various hypotheses on both a simple and a somewhat more complex problem.

3.4 CNN architecture configuration

In preparation for this study, we analyse a few influential and commonly used CNN architectures such as: LeNet [13], AlexNet [15], ZFNet [19], VGGNet [20], GoogleLeNet [21], ResNet [22] and DenseNet [24]. In our analysis we identify a few architectural patterns that seem to be present in many of the listed architectures. We list these patterns and their applicable architectures in Table 3.1.

In our research we study standard CNNs that consist of multiple convolutional layers, each followed by a pooling layer, connected to a set of fully connected layers. A convolutional layer consists of multiple convolutional kernels that are fine-tuned during the training process to identify and extract specific features from an input. We refer to the number of convolutional kernels per convolutional layer as the number of channels per layer. These convolutional kernels are convolved with an input, producing output feature maps.

Each convolutional layer is followed by a pooling layer containing pooling kernels. Pooling kernels have a fixed function that does not change during the training process. We specifically focus on max-pooling, as it has become the standard pooling operation in many modern CNNs (explained in Section 5.2), that identifies the largest value in a region. These max-pool layers reduce feature map size and extract dominant features identified by the convolutional layers. Following the pairs of convolutional and max-pool layers, CNNs contain fully connected layers. These fully connected layers receive their input from the final convolutional and max-pool layer pair.

As previously mentioned, the incredible performance and popularity of CNNs have resulted in the rise of many different architectural variations. Although some of these variations tend to outperform standard CNN architectures on specific tasks, it would be impossible to analyse all of them. Thus, in our work we focus on selected components of the standard CNN architecture as described above.

Although most of the analysed architectures were trained and tested on the ImageNet [16] dataset, we find that these patterns also produce good performance on both the MNIST and CIFAR10 datasets. Throughout our work we attempt to ensure that the architectures we use contain these patterns, except for the cases were we explicitly investigate certain components such as varying convolutional kernel sizes.

3.5 Optimisation protocol

We train a large number of models, following the exact same optimisation protocol. All networks are initialised with He initialisation [28] over 3 different seeds. Adam is used to optimise Cross-Entropy Loss with a mini-batch size of 128. We use the Adam optimisation protocol as it is able to converge rapidly, has little memory requirements, is computationally efficient [29], and is a popular choice for modern CNNs [2], [27]. Mishkin et al. [30] study the effects of various hyper-parameters on CNN performance and find that while mini-batch sizes larger than 512 decrease performance, very small mini-batch sizes slow down training significantly. We thus select a mini-batch size of 128, which

Table 3.1: Identified popular architectural patterns and their applicable CNN architectures.

CNN architectural pattern	**Applicable architecture**
Multiple convolutional layers with following pooling layers.	LeNet, AlexNet, ZFNet, VGGNet, ResNet, GoogleLeNet, DenseNet
Convolutional kernels tend to decrease in size in consecutive layers.	AlexNet, ZFNet, ResNet, GoogleLeNet
Convolutional kernels tend to only have a stride of one after the first convolutional layer.	LeNet, AlexNet, VGGNet, GoogleLeNet
Most CNNs tend to have at least one fully connected layer.	LeNet, AlexNet, ZFNet, VGGNet, ResNet, GoogleLeNet, DenseNet
Pooling kernels tend to have constant size over multiple layers.	LeNet, AlexNet, ZFNet, VGGNet, GoogleLeNet
Convolutional kernels tend to be between 7×7 and 3×3 pixels.	LeNet, ZFNet, VGGNet, GoogleLeNet, ResNet, DenseNet
Max-pool tends to have a stride of 2.	AlexNet, ZFNet, VGGNet, GoogleLeNet

achieves good performance and efficient training in our initial tests.

All networks are trained to near-perfect train accuracy ($> 98\%$) and are optimised on validation accuracy. Networks are initially trained for 100 epochs, when the best performing network is found in the last few epochs (last 30 epochs for MNIST, last 50 epochs for CIFAR10), training is extended by a set amount of epochs (30 epochs for MNIST, 50 epochs for CIFAR10) until the best performing network is not found in the last few epochs. Four initial learning rates are used: when the best performing learning rate is found at the edge of the learning rate sweep, the learning rate is varied by 0.001 outside the sweep range to ensure that only fully optimised networks are used to generate results. All results shown are averaged over 3 seeds.

3.6 Conclusion

With the varying levels of complexity between CIFAR10 and MNIST, we are able to ensure that our results are a good representation of the general performance of CNNs on similar image classification tasks. We also identify certain CNN architectural patterns that are present in most common and influential CNN architectures. We include these architectural patterns in our experiments with the aim of making our results more broadly applicable. We discuss the optimisation protocol we follow in subsequent chapters, in order to generate results from CNNs that are well optimised, and to avoid reporting on the results of outliers.

Chapter 4

Quantifying invariance

4.1 Introduction

In this section we define what we refer to as "translation-invariance" in Section 4.2. Translation-sensitivity maps and radial translation-sensitivity functions are used to quantify and compare how sensitive a network is to translated input samples. We also discuss how we generate these quantification metrics in Sections 4.2.1 and 4.2.3 and propose a small change in the method of calculating the similarity of two vectors in Section 4.2.2.

4.2 Translation invariance

As explained in Section 2.2.2, we refer to translation invariance as a system's sensitivity to translated inputs. This means that a system can be more or less translation-invariant based on the values received from the translation sensitivity quantification metric.

4.2.1 Translation-sensitivity maps

To quantify and measure translation sensitivity, we use translation-sensitivity maps and radial translation-sensitivity functions, as introduced by Kauderer-Abrams [2], with a slight change to the sensitivity metric. In the original paper by Kauderer-Abrams [2], they use Euclidean distance that produces a distance measurement to calculate the similarity of two vectors. As discussed in Section 4.2.2, we instead use Cosine similarity that produces a similarity measure (rather than a distance). With Euclidean distance, smaller values indicate high translation invariance and with Cosine similarity, larger values indicate high translation invariance. All results from Chapter 5 onwards consistently use the latter approach.

Translation-sensitivity maps are two-dimensional graphs that consist of multiple pixels. Each pixel has a value that represents the network's sensitivity to a specific shift in the input. To calculate the values of these pixels, we determine the similarity between two vectors, namely the base output vector and the translated output vector. These vectors are generated by passing an input sample through a network, with the output of the final fully connected layer being referred to as either the base output vector or the translated output vector. The base output vector is generated by passing a non-translated input to the network. The translated output vector is generated by passing a translated input image to the network, with the x-axis and y-axis shifts corresponding to the pixel's location in the translation-sensitivity map. The similarity between these two vectors is then used as the translation-sensitivity metric. If there is high similarity between the base output vector and the translated output vector, the network is less sensitive to that specific translation. This insensitivity is then represented by a brighter pixel in the translation-sensitivity map. We generate translation-sensitivity maps from each sample in the respective test sets. To generate a translation-sensitivity map, we calculate this similarity for each sample in the test set with 441 different translations (-10 pixel to 10 pixel shift in the x-axis and -10 pixel to 10 pixel shift in the y-axis) and calculate the average translation-sensitivity map over all samples in a class. In cases where we directly compare architectures, we obtain the average translation-sensitivity map of a network by

calculating the average over all samples of all classes of the test set.

4.2.2 Cosine similarity

In the introductory paper [2], the Euclidean distance (the length of a line segment between the two points) between the two vectors is used as the similarity metric. The outputs from the Euclidean distance calculation are non-normalised, restricting comparisons at different locations within a network. (Even if two layers have the same dimensions, the activation values at different layers may have different size distributions.) To address this normalisation issue, we propose the use of Cosine similarity (Eq. 4.1) to calculate the similarity between the two vectors. Cosine similarity measures the cosine of the angle between two vectors $\vec{a}$ and $\vec{b}$ in a multi-dimensional space, producing a similarity value between 1 (high similarity) and -1 (high dissimilarity).

$$cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} \tag{4.1}$$

To ensure that the Cosine similarity measurement produces comparable results to the Euclidean distance measurement, we use the classification accuracy of a network as a baseline sensitivity measurement to compare the two metrics. We use an architecture that is very similar to most Convolutional Neural Network (CNN) architectures used in the rest of this study: a standard CNN architecture with three convolutional layers, each followed by a max-pooling layer, a hidden layer and an output layer. Similarly to how we generate translation-sensitivity maps, we measure the accuracy of the CNN given samples with various amounts of translation. Classification accuracy is used as a baseline measurement as a drop in accuracy should correlate with less translation invariance. We then generate translation-sensitivity maps using Cosine similarity and Euclidean distance, respectively, and calculate Pearson Correlation Coefficients of the two measurements with the baseline to determine how correlated they are with classification accuracy. High Cosine similarity, indicating high invariance, correlates with high classification accuracy. The opposite is true for distance values from Euclidean distance, as small distances indicate

high invariance and correlation with high classification accuracy. We calculate how correlated the Cosine similarity measurements are to classification accuracy measurements and how correlated the negative of the Euclidean distance measurements are to classification accuracy.

The results in Figure 4.1 show the Cosine similarity and Euclidean distance Pearson correlation coefficients with classification accuracy of a standard CNN trained on MNIST. Architecture details can be found in Table A.1. Both sensitivity metrics (Cosine similarity and the negative of Euclidean distance) show a very strong positive correlation with classification accuracy (and with each other). From the results it seems that Cosine similarity does provide similar information to Euclidean distance with the added benefit of the results being normalised, allowing for comparisons across network architecture layers.

In Figure 4.2 we see two examples of translation-sensitivity maps. As previously explained, each pixel in the sensitivity map represents the network's sensitivity to a specific translation corresponding to the pixel's location in the map. This means that a pixel at position (2,2) indicates how insensitive the CNN is to an input shift of 2 pixels in the x-axis and y-axis. The pixel at position (0,0) is always very bright (indicating very high similarity) as it represents the similarity between the base vector and a vector produced from the same sample with a 0 pixel shift. In this example, the CNN is much less sensitive to input shifts of samples in class 1 compared to samples in class 0.

Dimensionality

Although Cosine similarity has the advantage of producing layer-normalised results, it does not allow for the direct comparison of vectors with different dimensions. When using either Euclidean distance or Cosine similarity the dimensions of the vectors greatly influence the calculated metric. As dimension increases, the average Euclidean distance value tends to increase while the average Cosine similarity value decreases.

To illustrate the size of this effect, we use Cosine similarity and Euclidean distance to

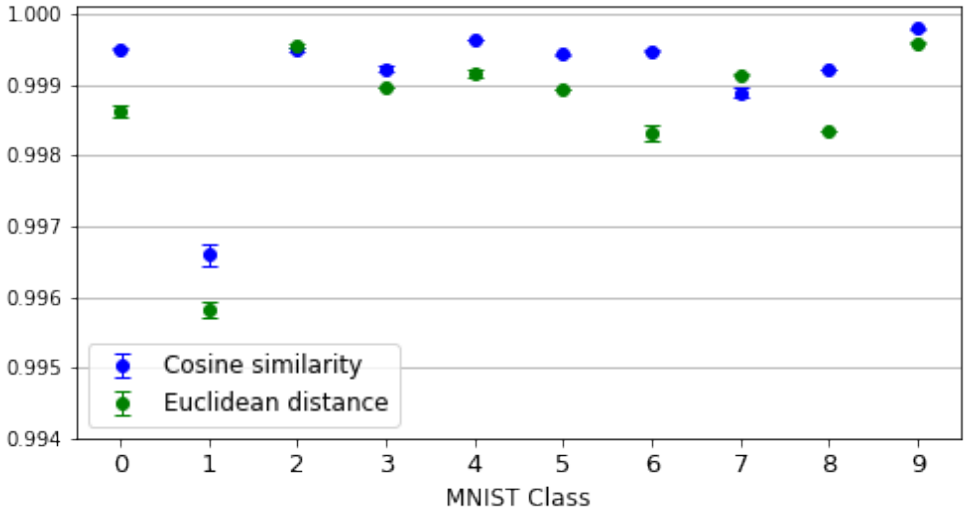

Figure 4.1: Pearson correlation of Cosine similarity and Euclidean distance with classification accuracy, calculated for each class of the MNIST dataset. Results are averaged over three seeds. CNN architecture details can be found in Appendix Table A.1.

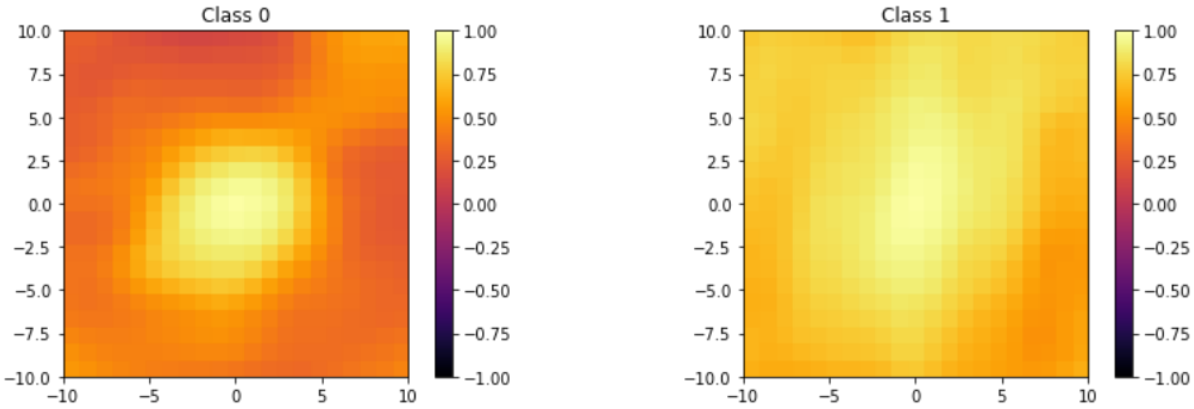

Figure 4.2: Example translation-sensitivity maps. These translation sensitivity maps are generated from a CNN trained on MNIST. The sensitivity map of class 0 (left) is darker than the sensitivity map of class 1 (right) indicating less translation invariance. These translation sensitivity maps are generated from the final fully connected layer outputs.

calculate the mean similarity of random vectors with different lengths. For each vector length we calculate the similarity of two vectors for 2000 sample vectors with random values between 0 and 1. From the results in Table 4.1 it is clear that the mean Cosine similarity decreases as vector dimension increases. We can also see that mean Euclidean distance increases as vector length increases.

This effect produces warped results when comparing either Euclidean distance or Cosine similarity values using different dimensions. As we use different output vectors to generate our results, we do not directly compare results generated from vectors with different lengths.

Table 4.1: Mean Cosine similarities and mean Euclidean distances of different length uniform random vectors.

Uniform random vector length	**Mean Cosine similarity**	**Mean Euclidean distance**
5	0.624	0.870
10	0.206	1.260
100	0.022	4.081
500	0.004	9.132

4.2.3 Radial translation-sensitivity functions

Radial translation-sensitivity functions are used to compare translation-sensitivity maps. These functions are generated by calculating the radial mean of a translation-sensitivity map at different radii. The output is a set of radial mean values that illustrate how sensitive a network is to the average translation that occurs at a specific radius. To generate these radial translation-sensitivity functions, all samples of the corresponding test set are used to generate an average translation-sensitivity map per class. These translation-sensitivity maps are used to generate radial translation-sensitivity functions for each class.

The final radial translations-sensitivity functions shown in our results are the average

sensitivity functions for all classes of a network. We also average all results over three initialisation seeds, showing the standard error of the three seeds with error bars.

In Figure 4.3 we see an example of a radial translation-sensitivity function. These functions are generated from the sensitivity maps in Figure 4.2. From this example one can see how the brighter, more translation-invariant sensitivity map of class 1 produces a radial translation-sensitivity function that also indicates its high translation invariance. In the figure we observe how the lines meet at a radius of zero, representing the centre pixel of the translation-sensitivity maps.

4.3 Conclusion

Using the proposed translation-sensitivity maps and radial translation-sensitivity functions we can quantify and compare results allowing us to study the effects of various CNN components on translation-invariance. We propose the use of Cosine similarity as the similarity metric when producing translation-sensitivity maps, and demonstrate that it provides similar results as Euclidean distance, with the advantage that it produces layer-normalised results. Although Cosine similarity produces layer-normalised results, we discuss the dangers of comparing the invariance of vectors with different dimensions.

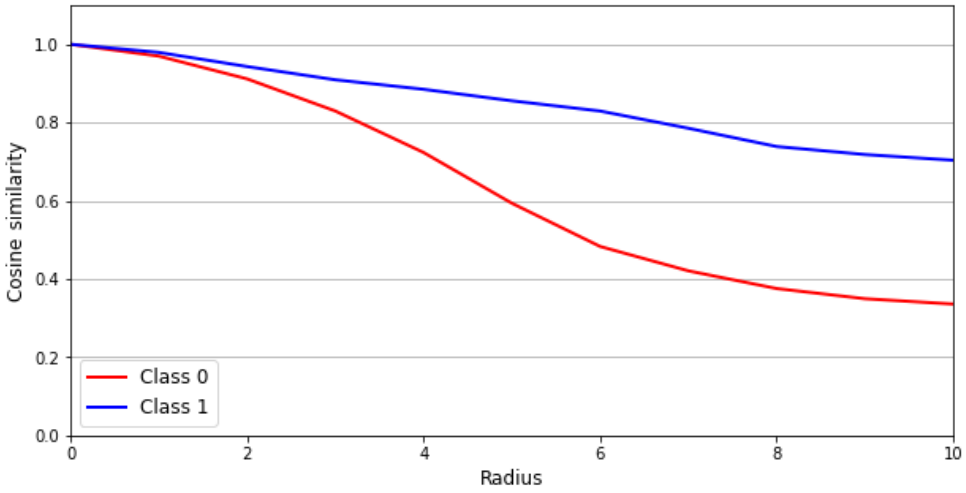

Figure 4.3: Example radial translation-sensitivity functions. These radial translation-sensitivity functions are generated from a CNN trained on MNIST. The blue line (top) is generated for class 1 and the red line (bottom) is generated for class 0. These translation sensitivity maps are generated from the final fully connected layer outputs. CNN architecture details can be found in Appendix Table A.1.

Chapter 5

Max-pooling

5.1 Introduction

In this section we investigate the role of pooling - more specifically, max-pooling. First, the way in which the pooling operation functions and its role in a standard Convolutional Neural Network (CNN) are discussed. We determine whether max-pooling layers themselves contribute to translation invariance and also analyse how different max-pool kernel sizes affect the translation invariance of a CNN. We start off by discussing the function of max-pooling in Section 5.2. We then investigate the effects of different max-pool kernel sizes in Section 5.3 and find that larger kernels result in more translation-invariance.

5.2 Max-pooling

We include max-pooling layers in the architectures, as they have become the standard pooling operation in most state-of-the-art CNN architectures [15], [19]–[21], [31]. The objective of max-pooling is to down-sample an input while retaining the important features in the given input. This results in reduced dimensionality of the sample, reducing the number of parameters that need to be learned, while still retaining the important fea-

tures of the sample. In the case of standard CNN architectures, max-pool layers always follow convolutional layers. This means that max-pool layers always receive the outputs from convectional layers as their input. We refer to these convolutional layer outputs as "feature maps" throughout the study. The max-pooling operation applies a max filter to sub-regions of the input. The max filter identifies the single largest value within the sub-regions and disregards all other values. As the max filter moves over the entire input space, these largest values are grouped and seen as the output of the max-pool operation. These max filter sub-regions are determined by the size of the max-pool kernels. Although the size of these kernels may vary, the max-pool operation only identifies a single greatest value per region. Figure 5.1 illustrates a simple example of the max-pool operation as a max-pool kernel with size 2×2 and stride 1 and 2 is applied to an input sample. There is a noticeable reduction in sample dimensionality as only the singular greatest value in each of the four colour-coded regions are not disregarded.

Max-pooling kernels can be thought of as convolutional kernels that have a fixed function and do not "learn" from given inputs. This fixed function of max-pool kernels is to morph and extract important features from input samples. One hypothesis is that because of the down-sampling caused by pooling layers, CNNs are forced to learn more invariant representations. To test this hypothesis we train two CNN architectures: one without max-pool layers and one with max-pool layers. To mitigate the effect of variations in feature map size, zero padding is used to ensure that feature maps produced by both networks have exactly the same size for each corresponding convolutional layer. To be able to keep the size of the feature maps constant across both networks, the stride of the pooling kernels is limited to 1. Although it is stated in Section 3.4 that pooling kernels tend to have a stride of larger than 1, it would be unrealistic to compare these two networks if the max-pool layers had a larger stride. To keep feature map size constant across both networks, absurd amounts of zero padding would be required. Hence in this investigation we purely focus on the effects of the max-pool kernels on translation invariance.

As previously discussed, max-pool kernels highlight and morph important features. This process down-samples the input samples, which should result in a less translation-sensitive CNN architecture. Although the amount of down-sampling is reduced due to the max-

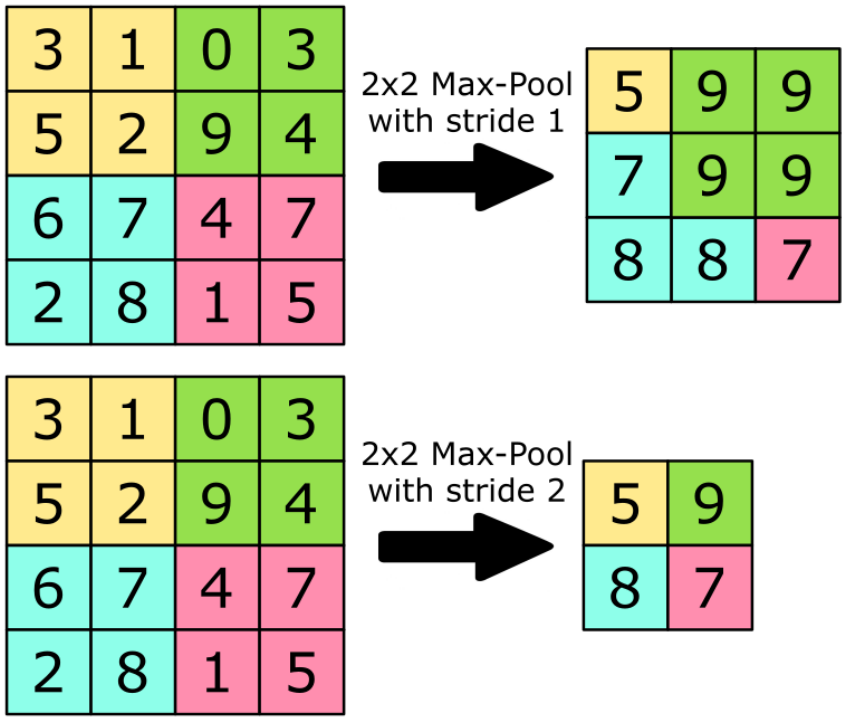

Figure 5.1: Max-pool operation example. In the above figure one can see a 2×2 max-pool kernel with a stride of 1 and a stride of 2 being applied to a 4×4 input sample.

pool kernels having a stride of 1, it is still expected that the max-pool layers will result in more translation invariance than a CNN without max-pool layers.

As previously stated, we investigate two architectures: one CNN without max-pool layers and one with max-pool layers with 2×2 max-pool kernels with stride 1 after each convolutional layer. Both architectures contain three convolutional layers with convolutional kernels that decrease in size per layer. We fully train and optimise the networks on the MNIST dataset with the optimisation protocol discussed in Section 3.5. Using the translation-sensitivity maps discussed in Section 4.2.1, we measure the translation invariance of both CNNs at their final fully connected layer outputs. From the translation-sensitivity maps we then generate the radial translation-sensitivity functions of both networks, as discussed in Section 4.2.3, and compare them in Fig 5.2.

Figure 5.2 shows the radial translation-sensitivity functions of two CNNs: one with max-pool kernels and one without, trained on MNIST. Architecture details can be found in Table A.2. From the results in Figure 5.2 one can see that the CNN with max-pooling

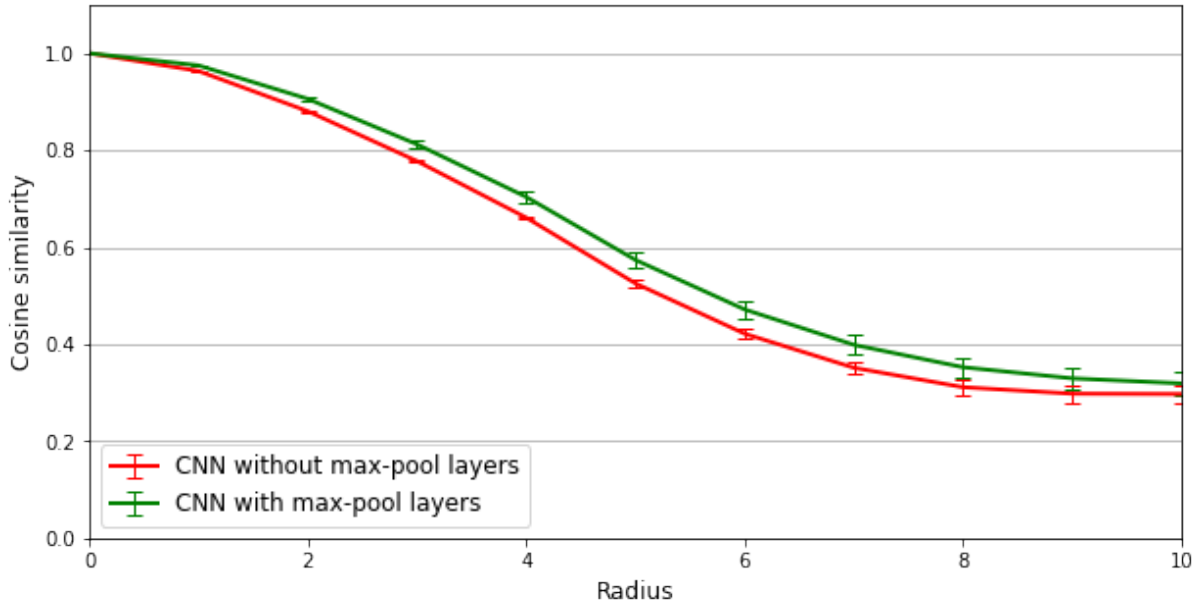

Figure 5.2: The effect of max-pool layers on translation invariance. In the above figure one sees the radial translation-sensitivity functions generated from a CNN with max-pool layers and a CNN without max-pool layers on MNIST. These radial translation-sensitivity functions are generated from the final fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.2.

kernels is more translation-invariant than the CNN without max-pool kernels as the results indicate a difference larger than the confidence interval (standard error). This confirms the hypothesis that down-sampling due to max-pooling does force a CNN to learn more invariant representations, for the architectures studied. It is evident that including max-pooling layers in a CNN, even with a max-pool kernel stride of just 1, reduces the CNN's sensitivity to translated inputs. In the rest of the study we use a standard max-pool kernel stride of 2, as motivated in Section 3.4.

5.3 Pooling kernel size

The previous section indicated that including max-pooling does increase translation invariance. In this section we explore the effects of max-pool layers in CNNs further. As previously said, max-pool layers consist of max-pool kernels that are applied to sub-sections of the input. Regardless of the number of values in each sub-section, only the

single largest value is not disregarded. As the size of these sub-sections increases, more and more information is disregarded, while important features are emphasised even more. To investigate this effect we compare three CNN architectures each with different max-pool kernel sizes. The CNNs are set up to have the same max-pool kernel size within each network, but different max-pool kernel sizes across the architectures. To ensure that only the effects of the max-pool kernel size are investigated, we keep all other components identical and use zero padding to ensure that the feature maps produced by each layer have the same size across the three architectures.

As previously explained, the larger the max-pool kernel size, the more information is disregarded while the important features are emphasised. On a simple task such as MNIST we expect to see that larger max-pool kernels result in more translation invariance, while on a more complex task such as CIFAR10 we expect to see a similar pattern but to a more limited extent.

Figure 5.3 shows the radial translation-sensitivity functions, as discussed in Section 4.2.3, of three CNNs containing max-pool kernels with different sizes. Translation-invariance is measured at their final fully connected layer outputs. Architecture details can be found in Table A.3. We fully train and optimise all three architectures on the MNIST dataset with the optimisation protocol discussed in Section 3.5. As expected, in Figure 5.3 one sees that larger max-pool kernels produce more translation-invariant CNNs.

Since MNIST is a more simple task, very large max-pool kernels do not seem to hinder the CNNs' capability to fit the dataset. On more complex classification tasks it is expected that very large max-pool kernels might inhibit the CNN from fitting the dataset, as they might disregard too much information from the original input sample. We repeat out analysis of the three CNNs with different max-pool kernel sizes on the CIFAR10 dataset. As explained in Section 3.3.3, we increase the number of channels per convolutional layer to ensure that the CNNs have sufficient capacity to fit the CIFAR10 dataset. Although CIFAR10 is considered to be much more complex than MNIST, the CNNs are still able to achieve near 100% train-accuracy.

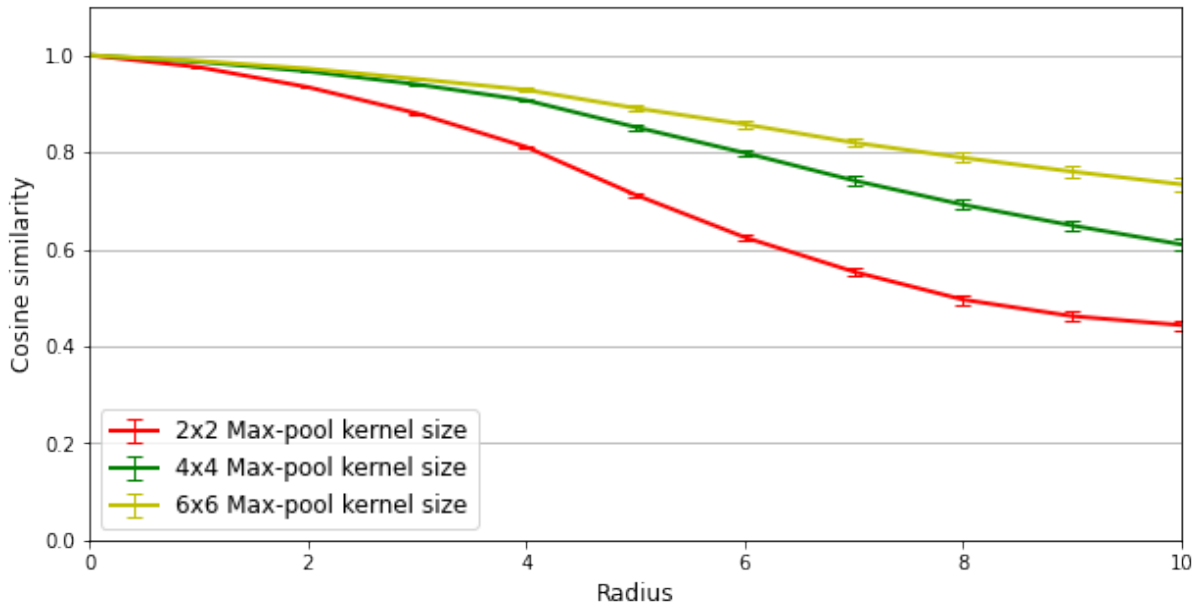

Figure 5.3: The effect of max-pool kernel size on CNNs trained on MNIST. The figure above shows the radial translation-sensitivity functions generated from three CNNs with different max-pool kernel sizes all with a stride of 2. These radial translation-sensitivity functions are generated from the final fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.3.

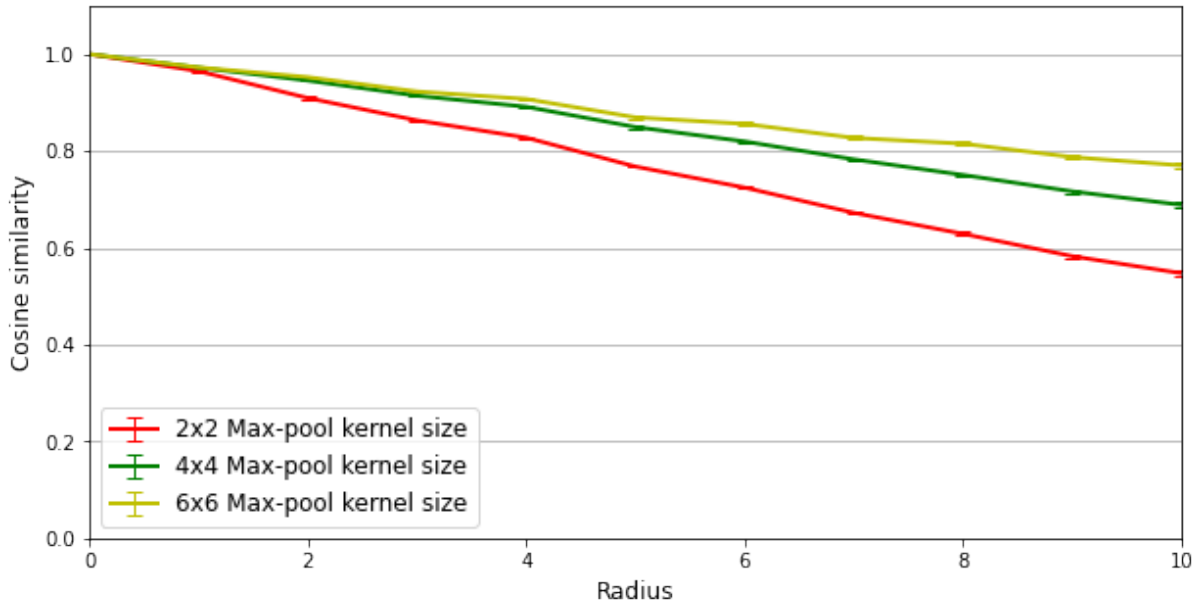

Figure 5.4: The effect of max-pool kernel size on CNNs trained on CIFAR10. The figure above shows the radial translation-sensitivity functions generated from three CNNs with different max-pool kernel sizes all with a stride of 2. These radial translation-sensitivity functions are generated from the final fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.4.

Figure 5.4 shows the radial translation-sensitivity functions of three CNNs containing max-pool kernels with different sizes on CIFAR10. Translation-invariance is measured at their final fully connected layer outputs. Architecture details can be found in Table A.4. The results in Figure 5.4 show the same pattern seen in the results of the CNNs trained on MNIST in Figure 5.3: larger max-pool kernels decrease translation sensitivity more than smaller max-pool kernels. Although these results are in line with our expectations, it is surprising to see that CNNs are able to fit the more complex dataset even with quite large max-pool kernels.

5.4 Conclusion

We studied the industry standard pooling operation, max-pooling, and found that including max-pool layers in a CNN does decrease the network's sensitivity to translated inputs. Focused on max-pooling, the effects of max-pool kernel size on translation invariance were investigated and it was found that larger kernels result in more invariant CNNs. We expected to find that larger max-pool kernels inhibit CNNs from fitting the more complex CIFAR10 dataset, but was proven wrong in the case of this analysis.

Chapter 6

Convolutional kernel size

6.1 Introduction

In this section we study the effects of convolutional kernel size on translation invariance. We discuss our expectation that larger convolutional kernels would result in less translation invariance, but find it difficult to make a definitive statement from our results. In Section 6.2 we discuss the functionality of convolutional kernels and the changes we make in Convolutional Neural Network (CNN) architectures for different datasets. We then explain our hypothesis in Section 6.3 and attempt to empirically prove or disprove it.

6.2 Convolutional kernels

Convolutional layers usually contain multiple convolutional kernels. These kernels are initialised with some initialisation-scheme before they are trained with backpropagation. The convolutional kernels are applied at various locations of an input sample, acting as filters that are fine-tuned to identify and "highlight" certain features. Initially these filters react to random features but during the training process these kernels are trained to identify and react to very specific features. A large number of these kernels are used in

each convolutional layer and are referred to as "channels". When we say that we increase the number of channels per layer to allow for sufficient capacity, we refer to an increase in the number of convolutional kernels per convolutional layer to allow the CNN to fit more complex datasets.

6.3 Convolutional kernel size

As discussed in Section 3.4, convolutional kernels tend to have a stride of 1 after the first layer. This means that most convolutional kernels (larger than 1×1 pixel) have overlapping sub-regions of the input samples they are applied to. As the size of convolutional kernels increases, the amount of overlap also increases. Since, unlike max-pool kernels, convolutional kernels each learns to act as a feature detector, it is difficult to predict how an increase in kernel size and sub-region overlap will affect translation invariance. One hypothesis is that larger convolutional kernels would result in poor translation invariance, as their increased size would allow them to capture more location information at each input sub-section to which they are applied.

To investigate the effects of convolutional kernel size on translation invariance, we train three different CNNs with different convolutional kernel sizes. All CNNs have the same convolutional kernel size across their own convolutional layers, but different convolutional kernel sizes across the three networks. Although this setup of constant convolutional kernel size across convolutional layers does not follow the pattern of decreasing kernel size identified in Section 3.4, it allows one to focus purely on the effects of convolutional kernel size. As in the earlier analysis, zero padding is used to ensure that feature map sizes are identical across the three CNNs. The first analysis is performed on the MNIST dataset.

Figure 6.1 shows the radial translation-sensitivity functions, as discussed in Section 4.2.3, of three CNNs with varying convolutional kernel size. Translation-invariance is measured at their final 10-dimensional fully connected layer outputs. Architecture details can be found in Table A.5. We fully train and optimise all three networks on the MNIST dataset with the optimisation protocol discussed in Section 3.5. From the varying ker-

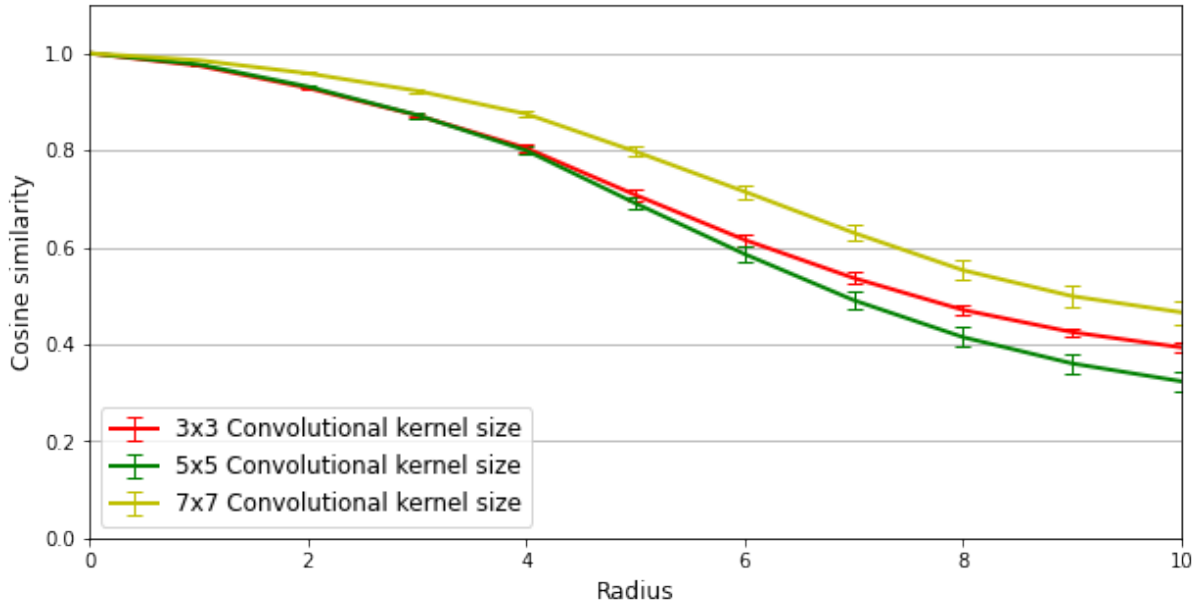

Figure 6.1: Influence of convolutional kernel size on translation invariance: radial translation-sensitivity functions generated from CNNs with differently sized convolutional kernels. Trained on the MNIST dataset. Functions are generated from the final 10-dimensional output layer. CNN architecture details can be found in Table A.5.

nel size results in Figure 6.1 it is difficult to identify a clear pattern. It seems that the largest convolutional kernel (7x7) provides more overall translation invariance than the two smaller kernels. This is somewhat unexpected, as we believe that larger kernels are able to capture and encode more location information than smaller kernels. Compared to earlier results, these radial translation-sensitivity maps have quite large standard error bars, indicating a large variation in the results produced by each CNN over their three different initialisation seeds. As MNIST is a very simple task and it is very easy to train these convolutional kernels to detect enough features to achieve near perfect test set accuracy, we expect to obtain more meaningful results on a more complex problem.

To investigate how convolutional kernel size affects translation invariance on a more complex task, we fully train and optimise three CNNs on the CIFAR10 dataset. Similar to the CNNs trained on MNIST, these networks all have the same convolutional kernel size across their convolutional layers, but different convolutional kernel sizes across the three networks. Zero padding is used to ensure that feature map sizes are constant across the networks.

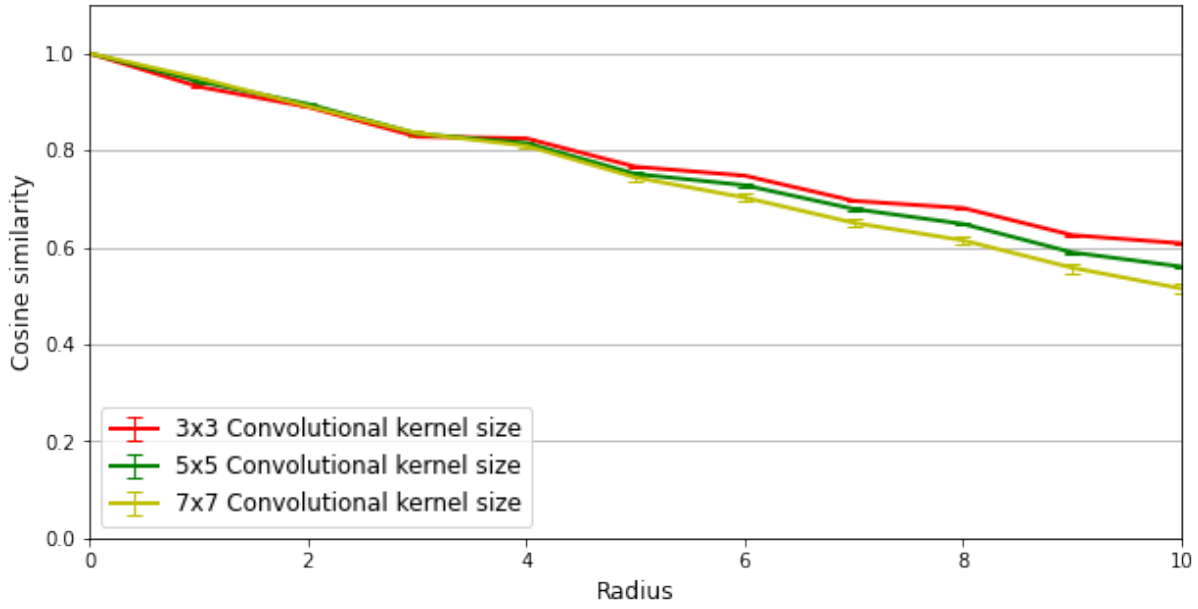

Figure 6.2: Influence of convolutional kernel size on translation invariance: radial translation-sensitivity functions generated from CNNs with differently sized convolutional kernels. Trained on the CIFAR10 dataset. Functions are generated from the final 10-dimensional output layer. CNN architecture details can be found in Appendix Table A.6.

Figure 6.2 shows the radial translation-sensitivity functions, of the three CNNs with varying convolutional kernel size. Architecture details can be found in Table A.6. Translation-invariance is measured at their 10-dimensional output layer. From the results in Figure 6.2 a pattern is identified that is more in line with our expectations. For translations of around 5 pixels, convolutional kernel size seems to have a negligible effect on translation invariance. For translations of more than 5 pixels, smaller convolutional kernels seem to produce slightly more translation invariance. Although these results seem to show that larger convolutional kernels encode more location information, the translation invariance of these three CNNs are so similar, even at 10 pixel shifts, that it is difficult to attribute these results solely to convolutional kernel size.

6.4 Conclusion

We studied the effects of convolutional kernel size on translation invariance. The initial expectation to find that smaller kernels produce more translation invariance was proven incorrect by the results from the CNNs trained on MNIST. The investigation was repeated on the more complex CIFAR10 and the results were more in line with the initial expectations. Because of the contrasting nature of the results, we believe that convolutional kernels have a more limited effect on translation invariance and that further analysis is required to make a more definitive statement on the role of convolutional kernel size in CNN translation invariance.

Chapter 7

Achieving translation invariance

7.1 Introduction

In this section we explore how the different sections of a Convolutional Neural Network (CNN), namely the convolutional and fully connected layers, respectively, contribute to translation invariance. We also investigate how the size of the feature maps produced by the convolutional layers affects how translation is encoded. In Section 7.2 we investigate the translation invariance contributed by the convolutional layers and the fully connected layers, respectively. We obtain interesting results regarding the role of fully connected layers in CNN translation invariance. Our results then lead us to investigate the feature maps produced by the final convolutional layer. In Section 7.3 we study the effects of feature map size on translation invariance and find that 1 pixel feature maps seem to produce more translation invariance than all other feature map sizes studied in our work.

7.2 Tracking translation invariance in a CNN

As previously stated, standard convolutional neural networks consist mainly of convolutional layers, separated by pooling layers, followed by a final set of fully connected layers.

The convolutional layers can be seen as encoders that morph and highlight important features in the input. The fully connected layers then use these encoded inputs to perform certain tasks. In essence, convolutional layers learn to identify certain features and their characteristics within the input and then pass these features to the following layers in a more efficient representation. As mentioned in Section 5.2, these efficient representations are referred to as "feature maps" and their size depends on several variables such as kernel size, stride, pooling, padding and input size.

Since the fully connected layers of a CNN act as the classifier, it is desired that the inputs they receive be unaffected by input shifts. Although it is known that complete translation invariance is unlikely with a standard CNN architecture, it is still expected that the convolutional layers will compensate for most of the translation in the input, as they are better equipped (with moving kernels and spatial awareness) to deal with translation. These kernels are convolved over the entire input space, allowing them locate specific features at nearly any location in the input.

To investigate this expectation, we directly compare translation sensitivity measurements generated from a CNN's final convolutional layer outputs with sensitivity measurements generated from fully connected layer outputs. In Section 4.2.2 we explain why comparing vectors with different dimensions will produce warped results. To avoid this phenomenon, we set up the CNN architecture to contain multiple fully connected hidden layers that match the dimensions of the final convolutional layer outputs when flattened into a one-dimensional vector. This setup allows us to compare the convolutional layer outputs directly with the final fully connected hidden layer outputs, with the added benefit of not "bottlenecking" the flow of information within the CNN.

Figure 7.1 shows the radial translation-sensitivity functions, as discussed in Section 4.2.3, generated from the convolutional layer outputs and the fully connected hidden layer outputs of a CNN. Thus translation-invariance is measured after the convolutional layers and after the hidden fully connected layers to study their individual contribution to translation invariance. Architecture details can be found in Table A.7. We fully train and optimise the network on the MNIST dataset with the optimisation protocol discussed in Section

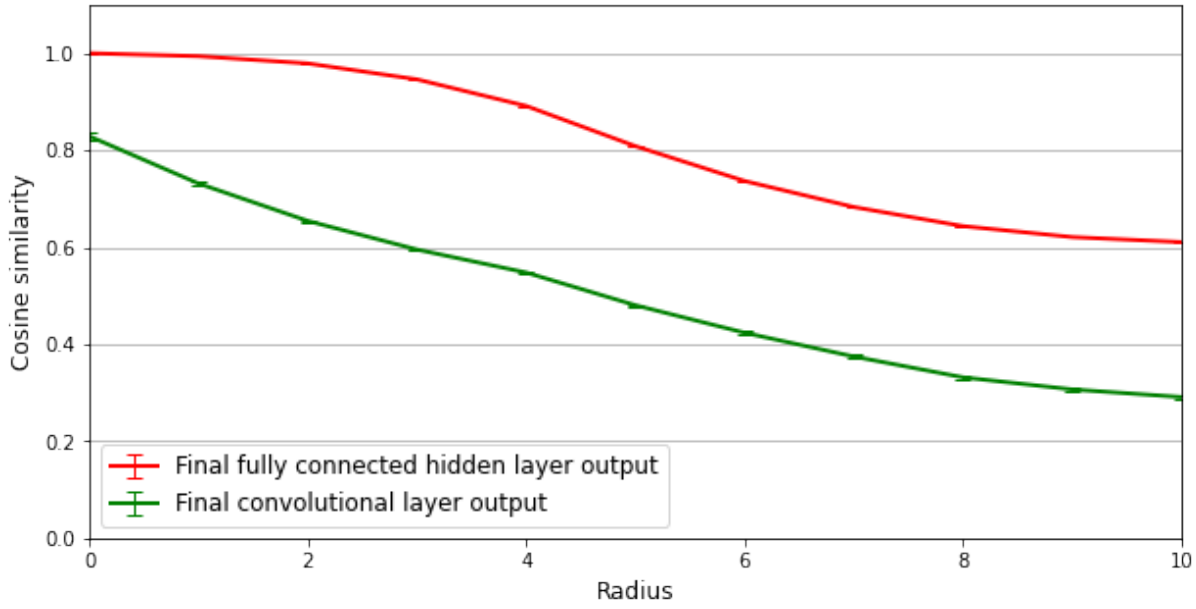

Figure 7.1: Radial translation-sensitivity functions generated from the final convolutional layer output compared to output generated from the final fully connected hidden layer output. These graphs are generated from a CNN trained and optimised on MNIST. CNN architecture details can be found in Table A.7.

3.5. Although the convolutional layers are expected to be responsible for most of the translation invariance of the network, the fully connected layers seem to contribute significantly to translation invariance. This is somewhat counter-intuitive, as it is expected that the convolutional layers would be able to compensate for the translated inputs far better than the fully connected layers. Having seen that the fully connected layers of a CNN are responsible for a large portion of the translation invariance observed, we next examine the feature maps produced by the final convolutional layer. Before obtaining these results we assume that the feature maps produced by the final convolutional layer would be nearly identical for the same input sample, regardless of input translation.

In Figure 7.2 one sees the feature map outputs of the final convolutional layer of a CNN given a non-translated and translated input samples of MNIST. From these results one can see that a shift in the input has an extremely large influence on the final feature maps that are passed to the fully connected layer. Since it seems that the convolutional layers have not been able to encode the shifted input effectively, the fully connected layers are forced to compensate for the translation by memorising this new representation of

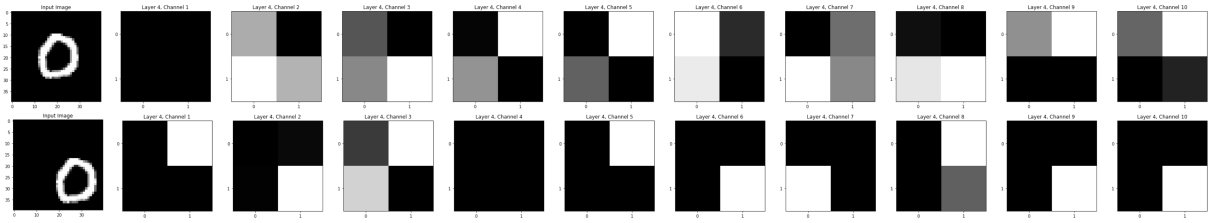

Figure 7.2: Feature maps produced by the final convolutional layer of a single CNN given an unshifted (top) and shifted (bottom) version of the same input sample. The final convolutional layer of this CNN contains 10 channels which produce 10 2×2 feature maps. The leftmost sample in each row is the input sample passed to the CNN.

the same sample. One hypothesis is that, since these experiments were performed on a very simple task (MNIST), the convolutional layers were never forced to learn a better encoding scheme, since the fully connected layers were easily able to memorise all of the transformed samples. This would explain the results in Figure 7.2, and why it seems that the fully connected layers significantly contribute to translation invariance. One way to test this hypothesis would be to limit the size of the convolutional layer output feature maps, limiting the number of output pixels available to the convolutional layers, and forcing them to learn a more effective encoding scheme. Reducing the feature map size to 1×1 pixel would remove translation from them completely and force them act as a single-valued feature.

7.3 Feature map size and translation invariance

Having previously explored the effects of convolutional kernel size on translation invariance using zero padding to keep feature map size constant, we now omit zero padding to analyse how varying feature map size affects translation invariance.

To have varying feature map sizes we implement convolutional kernels with different sizes similar to what we do in Section 6. Because of varying convolutional kernels sizes and no zero padding, one is able to study a range of feature map sizes and see how they affect translation invariance. As discussed in Section 7.2, we expect to see a noticeable increase in translation invariance when feature map size is reduced to 1×1 pixel, as it removes all

room for movement within the feature map itself.

To investigate the effects of feature map size on translation invariance we fully train and optimise three CNN architectures on MNIST. These CNNs are set up to have the same convolutional kernel size across their convolutional layers, but different convolutional kernel sizes across the three networks. No zero padding is used, as the aim of this analysis is to study varying feature map size. In an attempt to ensure that varying amounts of network capacity (due to different sized convolutional kernels) do not affect the results, we add channels to ensure that the number of effective nodes (kernel size $\times$ number of channels) per layer is comparable across networks.

Figure 7.3 shows the radial translation-sensitivity functions, as discussed in Section 4.2.3, generated from CNNs with different feature map sizes. Their translation-invariance is measured at their final fully connected layer outputs. Architecture details can be found in Table A.8. We fully train and optimise the networks on the MNIST dataset. From the results in Figure 7.3 it is clear that there is an increase in translation invariance when the feature map reaches a size of 1×1 pixel. Since there is no room for movement in a 1×1 pixel feature map, it seems that the convolutional layers are forced to better deal with input translations. Although it is clear that 1×1 pixel feature maps produce a higher level of translation invariance, it is expected that fully connected layers still contribute to invariance, as the CNNs are still not completely translation-invariant. Apart from 1×1 pixel feature maps, feature map size seems to have a more limited effect on translation invariance in CNNs trained on MNIST.

With the benefit of less translation-sensitive networks, why not reduce all convolutional layer output feature maps to 1×1 pixel? This is indeed possible, as is the case with some fully convolutional networks. However, these networks limit capacity in a way that additional fully connected layers do not. Neural networks trained for a classification task require adequate capacity to be able to learn the characteristics and features of the different classes. Using small feature map sizes is possible for tasks such as MNIST, but reducing feature map size can become a bottleneck in larger networks trained for more demanding classification tasks.

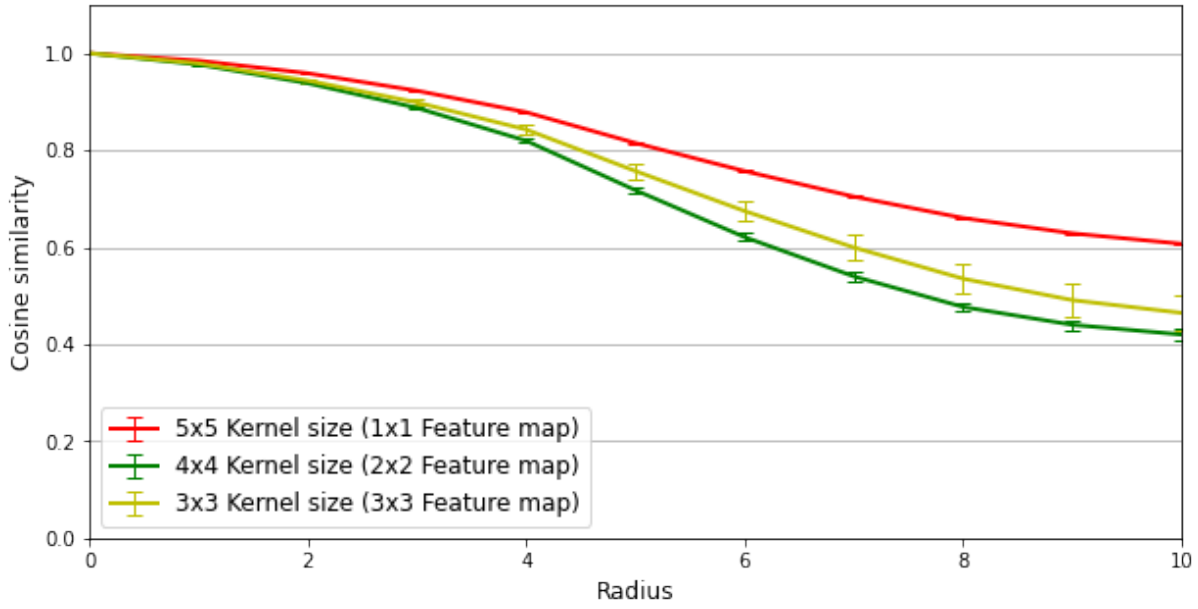

Figure 7.3: Radial translation-sensitivity functions of CNNs with different convolutional kernel sizes, resulting in different feature map sizes, on MNIST. These results are generated from the 10-dimensional fully connected layer outputs of the CNNs. CNN architecture details can be found in Table A.8.

To investigate how 1×1 pixel feature maps would affect translation invariance on a more complex dataset, we fully train and optimise four CNNs on the CIFAR10 dataset. Using differently sized convolutional kernels, the CNNs produce differently sized feature maps allowing us to study their effects on translation invariance. As previously mentioned, zero padding is not used as we wish to study the effects of feature map size. We also add channels to ensure that network capacity is comparable across the four CNNs. We expect to find similar results as in the case of the MNIST analysis in that feature map size has little influence on translation invariance apart from 1×1 pixel feature maps.

Figure 7.4 shows the radial translation-sensitivity functions, as discussed in Section 4.2.3, generated from CNNs with different feature map sizes. Their translation-invariance is measured at their final fully connected layer outputs. Architecture details can be found in Table A.9. We fully train and optimise the networks on the CIFAR10 dataset. From the results in Figure 7.4 one can see a slight increase in translation invariance in the CNN with 1×1 pixel final feature maps. As in our analysis of feature map size on MNIST, it seems that feature map size, apart from 1×1 pixel feature maps, have little influence on

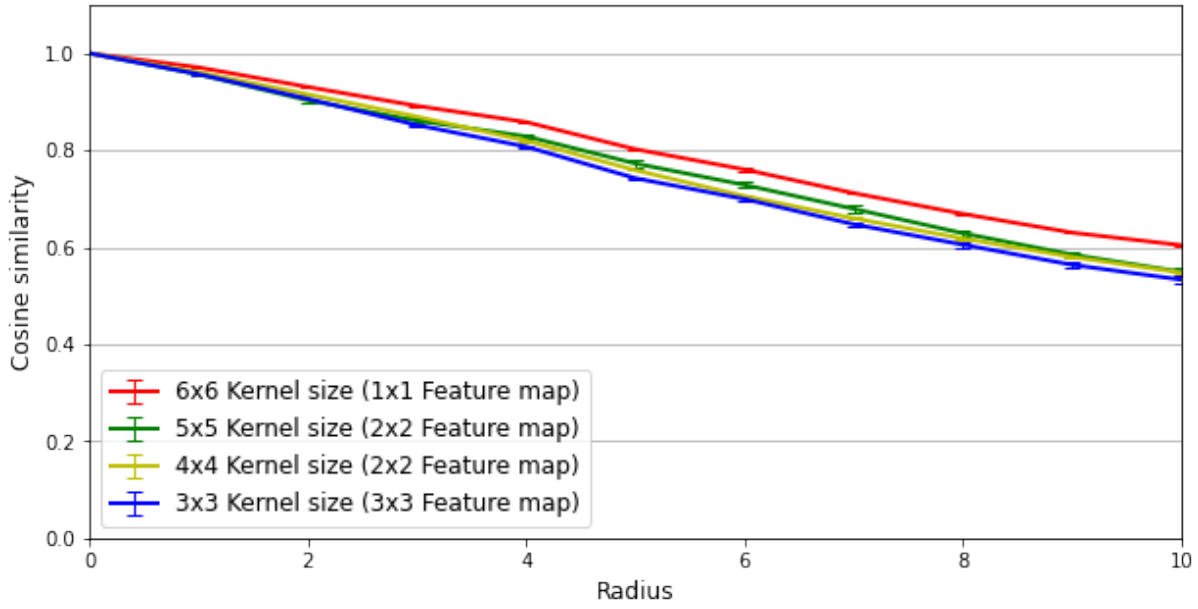

Figure 7.4: Radial translation-sensitivity functions of CNNs with different convolutional kernel sizes, resulting in different feature map sizes, on CIFAR10. These results are generated from the fully connected layer outputs of the CNNs. CNN architecture details can be found in Table A.9.

translation invariance in CNNs trained on CIFAR10.

7.4 Conclusion

In this section we explored how the different sections of a CNN, namely, the convolutional and fully connected layers, respectively, contribute to translation invariance. We measured translation invariance at different locations within a CNN and determined that fully connected layers are responsible for added translation invariance. To further investigate this effect, we studied the feature maps produced by the final convolutional layer of a CNN and found that input translation was still present in the feature maps even after three convolutional layers. This forces the fully connected layers to compensate for the translated inputs.

We then investigated whether reduced feature map size would result in more translation invariance, as it limits the amount of possible translation in the feature maps. In our

results of different CNNs trained on MNIST and CIFAR10, we found that feature map size has little influence on translation invariance apart from 1×1 pixel feature maps. We found that reducing feature map size to 1×1 pixel, somewhat eliminating the potential for translation in the feature maps, forces the convolutional layers to better deal with translated inputs before the fully connected layers. This increases the overall translation invariance of the CNN as the convolutional layers produce less translation sensitive feature maps that get passed to the fully connected layers. It seems that CNNs do not fully utilise convolutional layers to deal with spatial information if the problem they are trained on is not difficult enough.

Chapter 8

Distortion robustness

8.1 Introduction

In this section we explore how different Convolutional Neural Network (CNN) components affect a network's sensitivity to distortion, and investigate how these components interact with distortion compared to translation. To measure how sensitive a CNN is to distorted inputs, we define and generate distortion-sensitivity functions from the final fully connected layer output of the CNN. Using these distortion-sensitivity functions we investigate the effects of max-pool kernel size and convolutional kernel size on distortion-sensitivity. We first discuss the distortion algorithm and sensitivity functions in Section 8.2, followed by our analysis of max-pool kernel size and convolutional kernel size in Sections 8.3 and 8.4.

8.2 Distortion-sensitivity functions

Having focused on translation that shifts the image within the canvas, we now investigate distortion that warps and shifts the features within the image. We investigate elastic distortion as explained by Simard et al. [5] and analyse the differences between how

CNNs deal with distortion and translation. These elastic distortions are achieved by first generating random displacement fields with random values between -1 and +1. The fields are then convolved with a Gaussian of standard deviation σ, with a large σ value having little effect on the field and a small σ value having a greater effect on the field. To control the intensity of the distortion, the displacement fields are multiplied by α that acts as a scaling factor. These displacement fields are then applied to an input, determining the new location of each pixel in the input.

To be able to quantify distortion-sensitivity in CNNs, we define a distortion-sensitivity quantification metric: distortion-sensitivity functions. Distortion-sensitivity functions are generated using an algorithm very similar to the algorithm we use to generate translation-sensitivity maps. Distortion-sensitivity functions are one-dimensional functions that consist of multiple values. Each value represents the network's sensitivity to a distortion of a specific magnitude. To calculate the values of these functions, we determine the similarity between two vectors, namely the base output vector and the distorted output vector. These vectors are generated by passing an input sample through a network with the output of the final fully connected layer being referred to as either the base output vector or the distorted output vector. The base output vector is generated by passing a non-distorted input to the network and the distorted output vector is generated by passing a distorted input image to the network. The similarity between these two vectors is then used as the distortion-sensitivity metric. If there is a high similarity between the base output vector and the distorted output vector, the network is less sensitive to that specific distortion. We use Cosine similarity to measure the similarity of two vectors with a value of 1 indicating high similarity and a value if -1 indicating high dissimilarity.

We apply 10 distortions of different magnitude (0 to 9) to each sample in the test set, producing 10 similarity values that we show in the distortion-sensitivity functions. We apply distortions to input samples using the Augmentor [8] image augmentation library for machine learning with a grid size of 3×3. To generate translation-sensitivity maps, all samples of the dataset are zero padded with a 6 pixel border as explained in Section 3.3. Since we aim to compare distortion-sensitivity and translation invariance on a few architectures we have previously used, the 6 pixel zero padded border is still present in

the dataset samples.The 6 pixel border is not required to generate distortion-sensitivity functions; we only add the border here in order to ensure the experimental setups are directly comparable.

Figure 8.1 shows the effects of distortions with magnitudes ranging from 0 to 9 applied to a sample of both the MNIST and CIFAR10 datasets. The number above each distorted sample represents the distortion magnitude of the distortion applied to that sample. Figure 8.1 illustrates the distortions we use to generate distortion-sensitivity functions. Distortions with magnitudes of between 0 and 3 have little effect on the image and can be difficult to identify with the human eye. Distortions of magnitudes between 4 and 6 are more noticeable while distortions of magnitudes between 7 and 9 seem to drastically change the image, especially in the case of CIFAR10 samples. We feel that 0 to 9 is a good range for distortion magnitude as it will allow us to study how sensitive a CNN is to very slight distortion that changes only a few pixels as well as extreme distortion that completely warps the input.

8.3 Max-pool kernel size and distortion

As discussed in Section 5.2, max-pool kernels extract the largest value in a region, disregarding all other values. The size of this region is determined by the max-pool kernel size. As max-pool kernel size increases, important features become more emphasized and

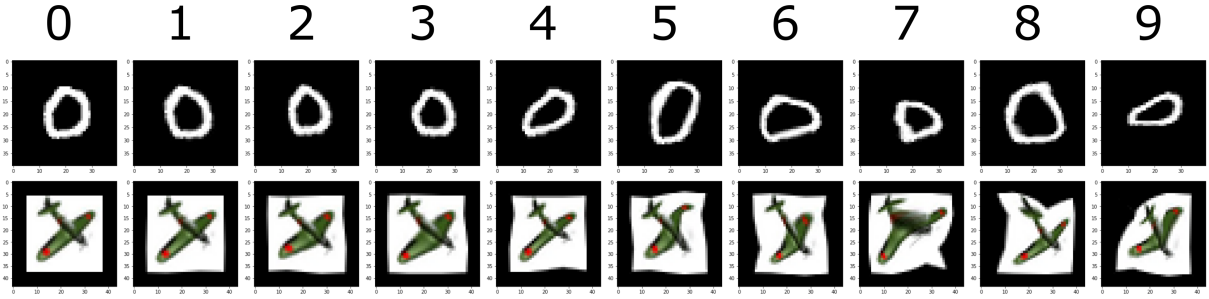

Figure 8.1: Distortions of magnitudes between 0 and 9 applied to a sample of both MNIST and CIFAR10. The number above each distorted sample represents the distortion magnitude of the distortion applied to that sample. These distorted samples are generated using the Augmentor library [8] with a grid size of 3×3.

a larger amount of "less important" information gets disregarded. Due to this, we expect to find that larger max-pool kernels result in less distortion-sensitivity as they are able to detect and extract important features in a larger region of the input.

To investigate the effects of max-pool kernel size on distortion-sensitivity we compare three CNNs each with different max-pool kernel sizes. The CNNs are set up to have the same max-pool kernel size within each network, but different max-pool kernel sizes across the three networks. To ensure that we only investigate the effects of the max-pool kernel size, we keep all other components identical and use zero padding to ensure that the feature maps produced by each layer have the same size across the three CNNs.

Figure 8.2 shows the distortion-sensitivity functions, as discussed in Section 8.2, of three CNNs containing max-pool kernels with different sizes. Distortion-sensitivity is measured at their final fully connected layer outputs. Architecture details can be found in Table A.3. We fully train and optimise all three networks on the MNIST dataset with the optimisation protocol discussed in Section 3.5. From the results in Figure 8.2 we can see that larger max-pool kernels do result in less distortion-sensitivity for CNNs trained on MNIST. It seems that larger max-pool kernels decrease distortion sensitivity and increase translation invariance for CNNs trained in MNIST as seen in Figure 5.3. In Figure 8.1 we can see the effects of distortion magnitude on a sample of MNIST and CIFAR10. Although the effects of distortion are noticeable in the MNIST sample, the effects seem much more intense in the CIFAR10 sample. Due to the increased complexity of CIFAR10 compared to MNIST, as explained in Section 3.3.3, we believe that CNNs trained on CIFAR10 learn to be inherently less sensitive to distortion. We thus expect to find that max-pool kernel size has a lesser effect on distortion-sensitivity for CNNs trained in CIFAR10 compared to CNNs trained on MNIST.

As in the analysis of max-pool kernel size on MNIST, we repeat the analysis on CNNs trained on CIFAR10. The three CNN architectures have differently sized max-pool kernels with all other components kept identical across networks.

Figure 8.3 shows the distortion-sensitivity functions, as discussed in Section 8.2, of three

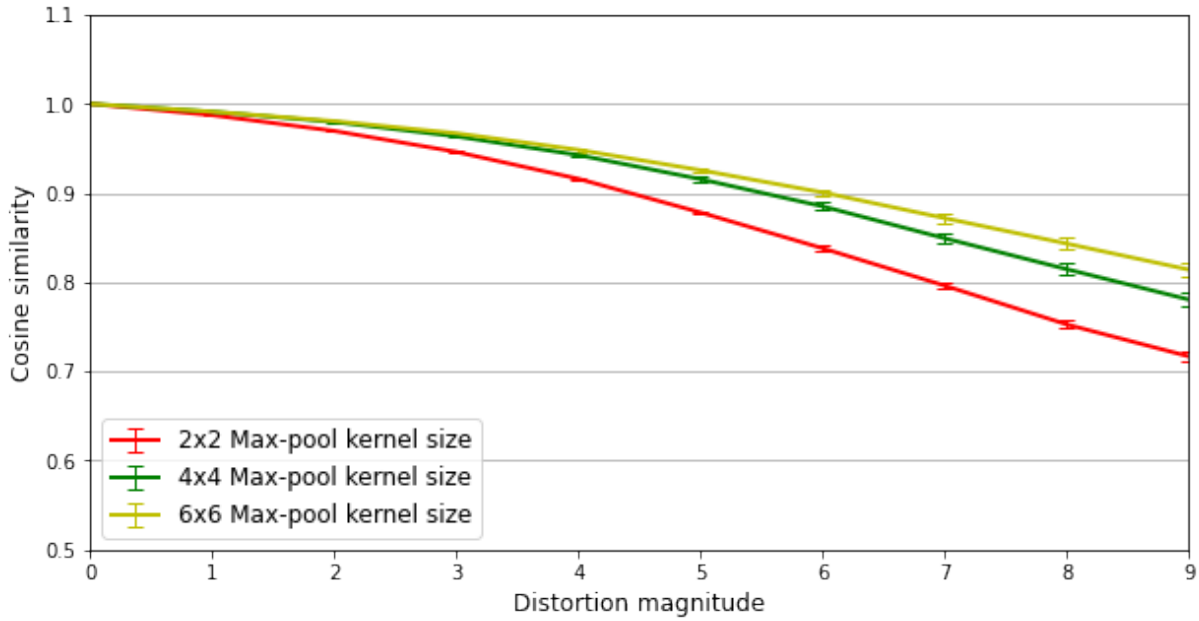

Figure 8.2: The effect of max-pool kernel size on distortion-sensitivity of CNNs trained on MNIST. The figure above shows the distortion-sensitivity functions generated from three CNNs with different max-pool kernel sizes all with a stride of 2. These results are generated from the 10-dimensional fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.3.

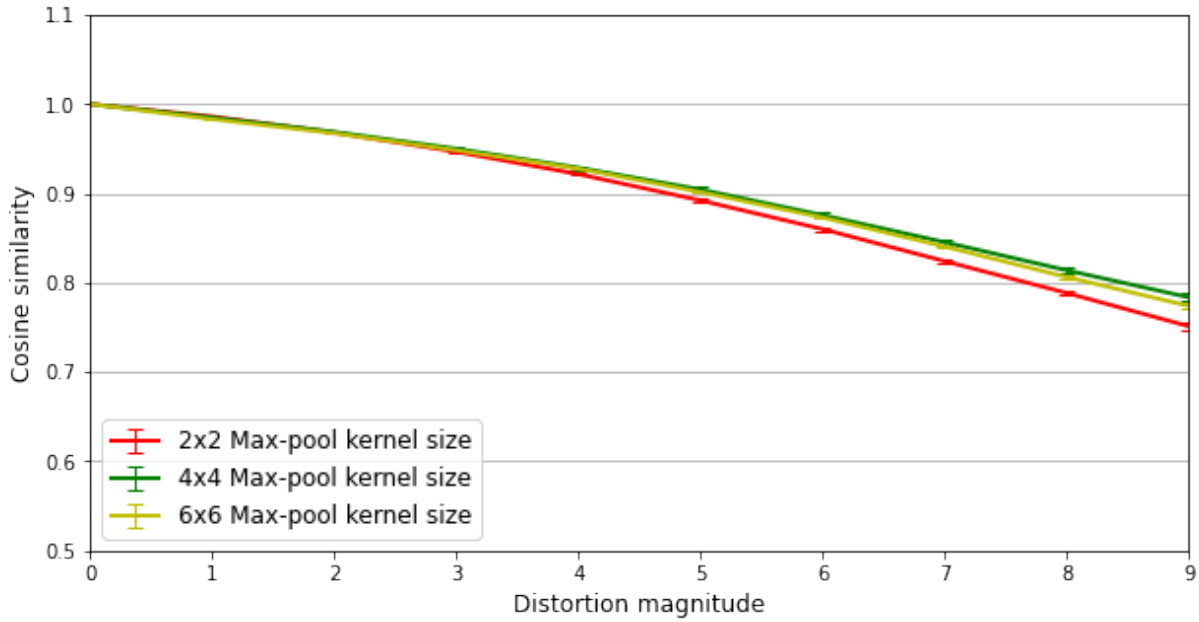

Figure 8.3: The effect of max-pool kernel size on distortion-sensitivity of CNNs trained on CIFAR10. The figure above shows the distortion-sensitivity functions generated from three CNNs with different max-pool kernel sizes all with a stride of 2. These results are generated from the fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.4.

CNNs containing max-pool kernels with different sizes. Distortion-sensitivity is measured at their final fully connected layer outputs. Architecture details can be found in Table A.4. We fully train and optimise all three networks on the CIFAR10 dataset. From the results in Figure 8.3 we find that all three CNNs have very similar sensitivity to distortion regardless of max-pool kernel size. These results somewhat confirm our expectation and show that the more complex CIFAR10 samples force CNNs to learn to be less sensitive to distortion. While max-pool kernel size greatly influences the translation invariance of a CNN trained on CIFAR10, as seen in Figure 5.4, it seems to have much less of an effect on distortion-sensitivity.

8.4 Convolutional kernel size and distortion

As explained in Section 6.2, convolutional kernels learn to detect specific features. This learning process makes it difficult to predict how a change in convolutional kernel size would affect the distortion-sensitivity of the CNN. One hypothesis is that since larger kernels are able to capture more information at each location they are applied to, they are expected to result in less distortion sensitivity. Since distortion warps and shifts features within the input, larger kernels are expected to better capture features that have been warped or shifted by a few pixels. Since larger convolutional kernels capture more location information, the position of a feature relative to another might still be captured by larger kernels when the input is distorted.

To investigate the effects of convolutional kernel size on distortion-sensitivity, we train three different CNNs with different convolutional kernel sizes. All CNNs have the same convolutional kernel size across their convolutional layers, but different convolutional kernel sizes across the three networks. Although this setup of constant convolutional kernel size across convolutional layers does not follow the pattern of decreasing kernel size identified in Section 3.4, it allows us to focus purely on the effects of convolutional kernel size. As in our earlier analysis, we use zero padding to ensure that feature map sizes are identical across the three CNNs. We perform our first analysis on the MNIST dataset.

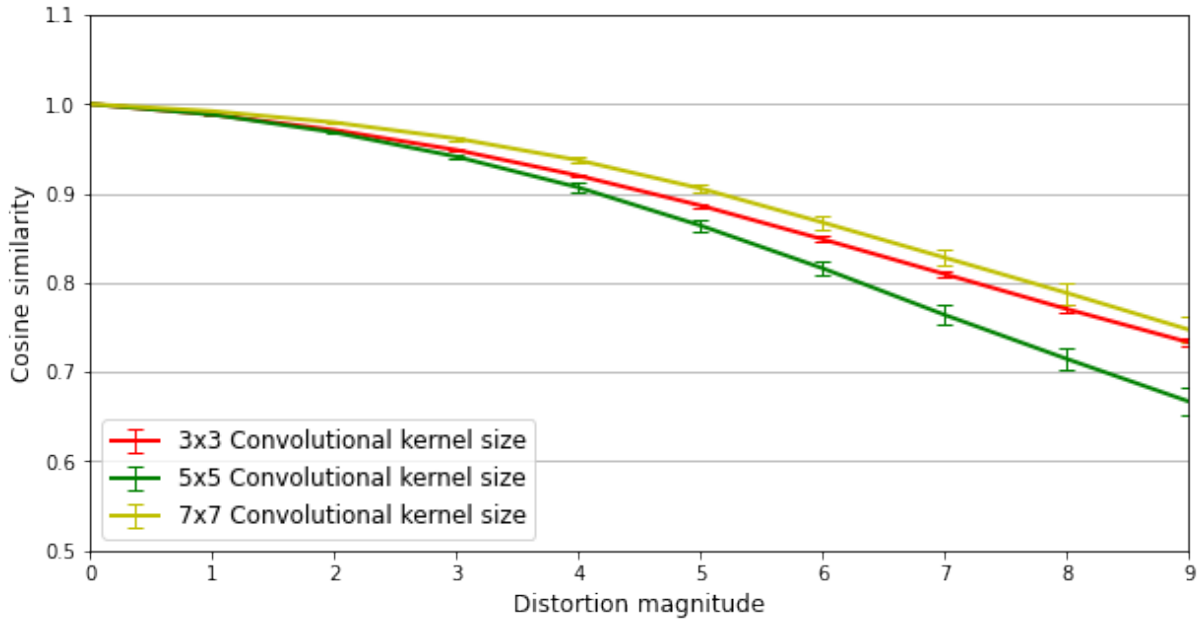

Figure 8.4: The effect of convolutional kernel size on distortion-sensitivity of CNNs trained on MNIST. The figure above shows the distortion-sensitivity functions generated from three CNNs with different convolutional kernel sizes. These results are generated from the final fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.5.

Figure 8.4 shows the distortion-sensitivity functions, as discussed in Section 8.2, of three CNNs containing convolutional kernels with different sizes. Distortion-sensitivity is measured at their final fully connected layer outputs. Architecture details can be found in Table A.5. We fully train and optimise all three networks on the MNIST dataset. From the results in Figure 8.4 it is difficult to identify a definite pattern. The CNN containing the largest convolutional kernels (7×7) is the least sensitive to distortion with the CNN with the smallest convolutional kernels (3×3) being just slightly more sensitive. Our hypothesis of larger convolutional kernels resulting in less distortion-sensitivity is proven to be somewhat correct with the 7×7 pixel CNN being the least sensitive to distortion, but also proven to be incorrect by the 5×5 pixel CNN being the most sensitive to distortion. It seems that convolutional kernel size has less of an effect on distortion-sensitivity, similar to translation invariance in Figure 6.1. To investigate this effect on a more complex task, we repeat our analysis on CNNs trained on CIFAR10.

Similar to the analysis of convolutional kernel size on MNIST, we repeat the analysis on

CNNs trained on CIFAR10. The three CNN architectures have differently sized convolutional kernels with all other components kept identical across networks.

Figure 8.5 shows the distortion-sensitivity functions, as discussed in Section 8.2, of three CNNs containing convolutional kernels with different sizes on CIFAR10. Distortion-sensitivity is measured at their final fully connected layer outputs. Architecture details can be found in Table A.6. The results in Figure 8.5 seem to support our expectations that convolutional kernel size has less of an effect on distortion-sensitivity. Similarly to the results we obtain in our analysis of the effects of max-pool kernel size on distortion robustness in Section 8.3, it seems that CNNs trained on CIFAR10 learn to be inherently less sensitive to distortion compared to networks trained on MNIST. Compared to the translation invariance results of convolutional kernel size on CNNs trained on CIFAR10 in Figure 6.2, it seems that convolutional kernel size has less of an effect on distortion-sensitivity than translation invariance.

8.5 Conclusion

In this section we study the effects of max-pool kernel size and convolutional kernel size on distortion-sensitivity and compare these effects to the results obtained for translation invariance. We first discuss the distortion-sensitivity functions used to quantify distortion robustness. Using the proposed distortion-sensitivity functions we analyse the effects of max-pool kernel size on distortion-sensitivity and find that larger max-pool kernels result in less distortion-sensitivity for CNNs trained on MNIST, similar to the high translation invariance produced by larger max-pool kernels. We then analyse how convolutional kernel size affects distortion-sensitivity and find that convolutional kernel size has less of an effect on distortion-sensitivity, similar to translation invariance, for CNNs trained on MNIST. In our analysis of max-pool kernel size and convolutional kernel size we find that CNNs trained on CIFAR10 learn to be inherently less sensitive to distortion, regardless of variations in max-pool or convolutional kernel size. We also find that max-pool and convolutional kernel size have a much larger effect on translation invariance than

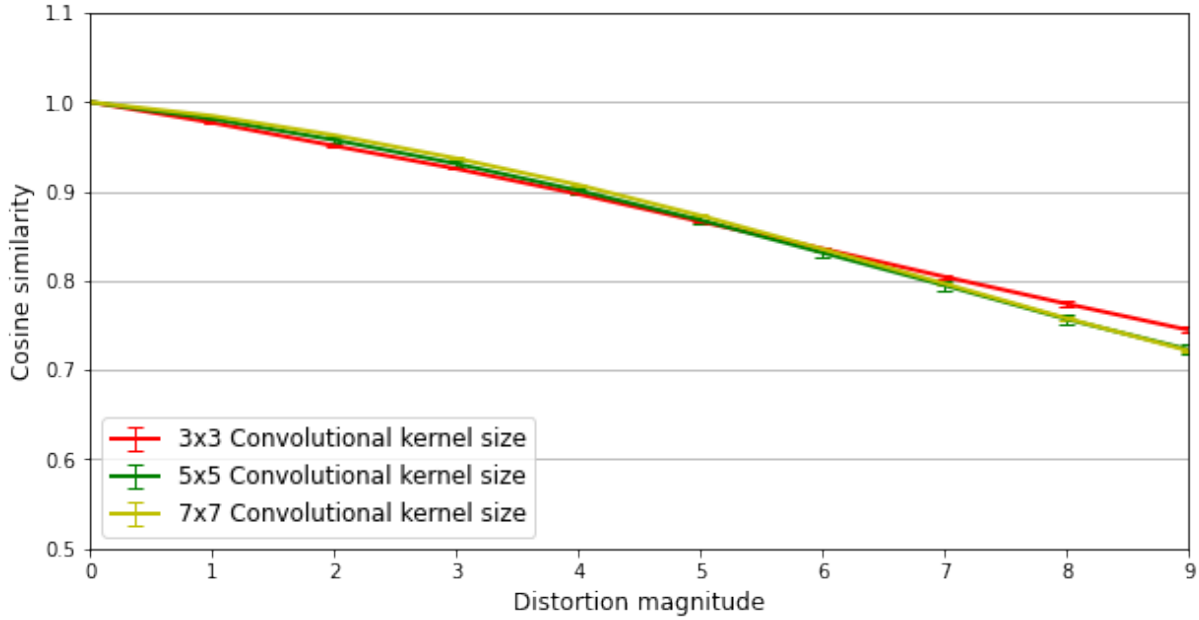

Figure 8.5: The effect of convolutional kernel size on distortion-sensitivity of CNNs trained on CIFAR10. The figure above shows the distortion-sensitivity functions generated from three CNNs with different convolutional kernel sizes. These results are generated from the final fully connected layer outputs of the CNNs. CNN architecture details can be found in Appendix Table A.6.

distortion-sensitivity for networks trained on CIFAR10.

Chapter 9

Conclusion

9.1 Introduction

In this chapter we summarize our findings and give an overview of our results. We discuss the translation-sensitivity maps used and how the similarity metric was changed. From the results obtained, Convolutional Neural Network (CNN) components and their contributions to translation invariance are discussed. The differences between how CNNs deal with translated and distorted inputs are investigated in Section 8. We also highlight the main contributions of our study and discuss possible future work that we feel might produce relevant insights. We start by discussing our key findings and implications in Section 9.2 followed by our contributions in Section 9.3. As there are still questions remaining, we discuss possible future work in Section 9.4 and conclude our study in Section 9.5.

9.2 Key findings and implications

To quantify translation invariance we use translation-sensitivity maps and radial translation-sensitivity functions introduced by Kauderer-Abrams [2]. We illustrate the functionality

of these metrics and propose a change to the similarity metric used to generate these maps. To evaluate if Cosine similarity is a suitable replacement for Euclidean distance, we calculate the Pearson correlation coefficients of both these metrics with classification accuracy: Results indicate that Cosine similarity is highly correlated with classification accuracy, motivating the use of Cosine similarity as replacement similarity metric.

The main focus of our study is to analyse components of the standard CNN architecture that might influence translation invariance. The initial analysis is performed on max-pooling and the results indicate:

- Including max-pool layers in a CNN increases translation invariance, even with a pool kernel stride of 1.
- Larger max-pool kernels result in more translation invariance than smaller max-pool kernels.
- CNNs are able to fit MNIST and CIFAR10 even with large (6×6) max-pool kernels.

Following the theme of the study, we investigate the effects of convolutional kernel size on translation invariance. Zero padding is used to ensure that feature map size is constant regardless of the convolutional kernel size and does not affect the results. The expectation was that larger convolutional kernels would result in less translation invariance, as they have large receptive fields and are able to encode large amounts of location information. From the analysis we found:

- Although it seems that convolutional kernel size has little effect on translation invariance, we feel that more analysis is required to make a definite statement on the influence of convolutional kernel size on translation invariance.

We also investigated how convolutional kernels and fully connected layers, respectively, contribute to the translation invariance of a CNN as a whole by comparing the translation invariance of the final convolutional layer output and the final fully connected layer output.

The feature map outputs of the final convolutional layer were studied and reduced to a single pixel in an attempt to remove possible translation in the feature maps. From the results we found:

- Fully connected layers are responsible for a large portion of the CNN's translation invariance.
- Shifted input samples have a drastic effect on the feature maps that are passed to the fully connected layers. This forces the fully connected layers to learn several different representations for a single sample, explaining the high levels of translation invariance these add to a CNN.
- Reducing feature map size to 1 pixel, eliminating possible translation in the feature map, does increase translation invariance.

Finally we focus on distortion-sensitivity and repeat some of our earlier analyses to investigate the effects of max-pool kernel size and convolutional kernel size on distortion-sensitivity. We propose the use of distortion-sensitivity functions to quantify how sensitive a network is to distorted inputs. We then compare the results of distortion-sensitivity with the results obtained for translation invariance to investigate the differences between how CNNs deal with these transforms. Using the proposed distortion-sensitivity functions we find that: Larger max-pool kernels reduce distortion-sensitivity for CNNs trained on MNIST, with an effect very similar to that observed when studying translation invariance. This effect is less visible for networks trained on CIFAR10, that learn to be inherently less sensitive to distortion.

9.3 Contributions

In view of the growth in popularity of CNNs, research done in the field mainly focuses on achieving the next state-of-the-art performance on various benchmarks. In this study the focus is on gaining better understanding of the base components of CNNs and their

individual contributions to translation invariance. We redefine the term "translation invariance" and use it to refer to a system's sensitivity to translated input data; using translation sensitivity-maps enables the quantification of the translation invariance of a system. We investigate specific standard architectural components of CNNs that might influence translation invariance and characterise their contribution. In addition, the effect of CNN components on elastic distortion robustness are studied and summarised. We develop distortion-sensitivity functions that act as a quantification metric for distortion-sensitivity in CNNs. Thus our research contributes to gaining better understanding of components implemented in most CNN architectures by characterising their individual contributions to translation invariance and distortion robustness.

9.4 Future work

Because of the high computational requirements of translation-sensitivity map generation, the number of architectures and layer outputs that could be analysed in this study is limited. We only study standard CNN architectures with three convolutional layers. Architectures containing more convolutional layers with forms of regularization such as Dropout or Batchnorm could provide interesting results. Since residual blocks and inception modules have become so popular in the past few years, we believe that they might greatly affect the translation invariance of a CNN.

Since the results indicate that fully connected layers make a large contribution to translation invariance, we believe future work that studies the extensions to the CNN architecture that force the convolutional layers to deal with all spatially aware information, rather than allowing the fully connected layers to compensate unnecessarily, may provide valuable insights to understanding translation invariance in CNNs.

9.5 Conclusion

The initial aim of this study was to gain better understanding of some of the components of CNNs that affect translation invariance. The focus is specifically on max-pooling layers, convolutional layers and fully connected layers using translation-sensitivity maps to quantify their individual contributions to translation invariance. We find that including max-pooling layers in a CNN produces more translation invariance and that larger max-pool kernels result in even more translation invariance. Moreover, although convolutional layers are better equipped to deal with translated inputs, fully connected layers seem to be responsible for a large amount of translation invariance when the final convolutional layer feature maps can contain movement. In our analysis of distortion-sensitivity we find that larger max-pool kernels seem to result in less distortion-sensitivity while convolutional kernel size has a limited effect on how sensitive a CNN is to distorted inputs. We also find that networks trained on CIFAR10 seem to learn to be inherently less sensitive to distortion than networks trained on MNIST.

All in all, it seems that convolutional layers are not fully utilised to deal with spatial information if the training task is not difficult enough, forcing the fully connected layers (spatially unaware) to compensate by learning similar elements at different inputs. By reducing feature map size to 1, forcing the convolutional layers to better deal with translation, we obtain the most translation invariant system studied, when evaluated on the unseen test set. Also, we observe a few similarities in how CNNs deal with translation and distortion and attribute these similarities to the network's ability to pinpoint important features in general, rather than specifically the movement of kernel across the input during convolution.

Bibliography

[1] O. S. Kayhan and J. C. van Gemert, "On translation invariance in CNNs: Convolutional layers can exploit absolute spatial location," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] E. Kauderer-Abrams, "Quantifying translation-invariance in Convolutional Neural Networks," *CoRR*, vol. abs/1801.01450, 2018. arXiv: 1801.01450.

[3] R. Zhang, "Making Convolutional Networks shift-invariant again," *International Conference on Machine Learning*, vol. abs/1904.11486, 2019. arXiv: 1904.11486.

[4] A. Azulay and Y. Weiss, "Why do deep Convolutional Networks generalize so poorly to small image transformations?" *Journal of Machine Learning Research 20*, 2019. arXiv: 1805.12177.

[5] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for Convolutional Neural Networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition*, 2003, pp. 958–963. DOI: 10.1109/ICDAR.2003.1227801.

[6] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[7] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[8] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An image augmentation library for machine learning," *The Journal of Open Source Software*, vol. abs/1708.04680, 2017. arXiv: 1708.04680.

[9] J. C. Myburgh, C. Mouton, and M. H. Davel, "Tracking translation invariance in CNNs," *Communications in Computer and Information Science (LNCS sub-series CCIS)*, vol. 1342, Accepted for publication.

[10] C. Mouton, J. C. Myburgh, and M. H. Davel, "Stride and translation invariance in CNNs," *Communications in Computer and Information Science (LNCS sub-series CCIS)*, vol. 1342, Accepted for publication.

[11] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology (London)*, vol. 195, pp. 215–243, 1968.

[12] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple Neural Nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010. DOI: 10.1162/neco_a_00052.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., 2012, pp. 1097–1105.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR09*, 2009.

[17] *Large scale visual recognition challenge*, 2012. [Online]. Available: http://image-net.org/challenges/LSVRC/2012/results.html.

[18] *Large scale visual recognition challenge*, 2013. [Online]. Available: http://www.image-net.org/challenges/LSVRC/2013/results.php.

[19] M. D. Zeiler and R. Fergus, "Visualizing and understanding Convolutional Networks," in *Computer Vision – ECCV*, Springer International Publishing, 2014, pp. 818–833, ISBN: 978-3-319-10590-1.

[20] K. Simonyan and A. Zisserman, "Very deep Convolutional Networks for large-scale image recognition," *3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, vol. abs/1409.1556, 2015.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with Convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1409.4842, 2015. arXiv: 1409.4842.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1512.03385, 2016. arXiv: 1512.03385.

[23] *Large scale visual recognition challenge*, 2015. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/results.

[24] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected Convolutional Networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1608.06993, 2017. arXiv: 1608.06993.

[25] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1411.5908, 2015. arXiv: 1411.5908.

[26] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949. DOI: 10.1109/jrproc.1949.232969.

[27] M. Srivastava and K. Grill-Spector, "The effect of learning strategy versus inherent architecture properties on the ability of Convolutional Neural Networks to develop transformation invariance," *CoRR*, vol. abs/1810.13128, 2018. arXiv: 1810.13128.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," *IEEE International Conference on Computer Vision (ICCV)*, vol. abs/1502.01852, 2015. arXiv: 1502.01852.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2015. arXiv: 1412.6980.

[30] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of Convolution Neural Network advances on the ImageNet," *Computer Vision and Image Understanding*, vol. 161, pp. 11–19, 2017. DOI: 10.1016/j.cviu.2017.05.007.

[31] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in Convolutional architectures for object recognition," 2010, pp. 92–101. DOI: 10.1007/978-3-642-15825-4_10.

Appendix A

Architectures

In this section we list all the Convolutional Neural Network (CNN) architectures we use in our study, as well as their individual train, validation and test accuracies.

Table A.1: Standard CNN architecture used in Cosine similarity and Euclidean distance analysis on MNIST.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	1	40×40	-	-	-	-
1	Conv	10	36×36	5×5	1	ReLU	0
-	Max-pool	10	18×18	2×2	2	-	0
2	Conv	20	15×15	4×4	1	ReLU	0
-	Max-pool	20	7×7	2×2	2	-	0
3	Conv	30	5×5	3×3	1	ReLU	0
-	Max-pool	30	2×2	2×2	2	-	0
4	FC	-	500	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.1 achieved an average train-accuracy of 100%, validation-accuracy of 99.52% and a test-accuracy of 99.27% on MNIST.

Table A.2: CNN architectures with max-pool layers and without max-pool layers on MNIST.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	1	40×40	-	-	-	-
1	Conv	10	36×36	5×5	1	ReLU	0
-	Max-pool	10, 0	37×37, 36×36	2×2, 0×0	1	-	0, 1
2	Conv	20	34×43, 33×33	4×4	1	ReLU	0
-	Max-pool	20, 0	33×33, 33×33	2×2, 0×0	1	-	0
3	Conv	30	31×31, 31×31	3×3	1	ReLU	0
-	Max-pool	30, 0	32×32, 31×31	2×2, 0×0	1	-	0, 1
4	FC	-	500	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.2 achieved an average train-accuracy of 100% on MNIST. The CNNs with max-pool kernels achieved an average validation-accuracy of 99.45% and test-accuracy of 99.39%. The CNNs without max-pool kernels achieved an average validation-accuracy of 99.31% and test-accuracy of 99.18%.

Table A.3: CNNs with differently sized max-pool kernels on MNIST.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	1	40×40	-	-	-	-
1	Conv	10	36×36	5×5	1	ReLU	0
-	Max-pool	10	18×18	2×2, 4×4, 6×6	2	-	0, 1, 2
2	Conv	20	15×15	4×4	1	ReLU	0
-	Max-pool	20	7×7	2×2, 4×4, 6×6	2	-	0, 1, 2
3	Conv	30	5×5	3×3	1	ReLU	0
-	Max-pool	30	2×2	2×2, 4×4, 6×6	2	-	0, 1, 2
4	FC	-	500	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.3 achieved an average train-accuracy of 99.98% on MNIST. The CNNs with 2×2 max-pool kernels achieved an average validation-accuracy of 99.34% and test-accuracy of 99.29%. The CNNs with 4×4 max-pool kernels achieved an average validation-accuracy of 99.48% and test-accuracy of 99.24%. The CNNs with 6×6 max-pool kernels achieved an average validation-accuracy of 99.41% and test-accuracy of 99.14%.

Table A.4: CNNs with differently sized max-pool kernels on CIFAR10.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	3	44×44	-	-	-	-
1	Conv	50	40×40	5×5	1	ReLU	0
-	Max-pool	50	20×20	2×2, 4×4, 6×6	2	-	0, 1, 2
2	Conv	100	17×17	4×4	1	ReLU	0
-	Max-pool	100	8×8	2×2, 4×4, 6×6	2	-	0, 1, 2
3	Conv	150	5×5	6×6	1	ReLU	0
-	Max-pool	150	3×3	2×2, 4×4, 6×6	2	-	0, 1, 2
4	FC	-	500	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.4 achieved an average train-accuracy of 99.24% on MNIST. The CNNs with 2×2 max-pool kernels achieved an average validation-accuracy of 75.52% and test-accuracy of 74.58%. The CNNs with 4×4 max-pool kernels achieved an average validation-accuracy of 78.38% and test-accuracy of 77.41%. The CNNs with 6×6 max-pool kernels achieved an average validation-accuracy of 77.21% and test-accuracy of 76.19%.

Table A.5: CNNs with differently sized convolutional kernels on MNIST.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	1	40×40	-	-	-	-
1	Conv	10	40×40	3×3, 5×5, 7×7	1	ReLU	1, 2, 3
-	Max-pool	10	20×20	2×2	2	-	0
2	Conv	20	20×20	3×3, 5×5, 7×7	1	ReLU	1, 2, 3
-	Max-pool	20	10×10	2×2	2	-	0
3	Conv	30	10×10	3×3, 5×5, 7×7	1	ReLU	1, 2, 3
-	Max-pool	30	5×5	2×2	2	-	0
4	FC	-	500	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.5 achieved an average train-accuracy of 100.00% on MNIST. The CNNs with 3×3 convolutional kernels achieved an average validation-accuracy of 99.51% and test-accuracy of 99.19%. The CNNs with 5×5 convolutional kernels achieved an average validation-accuracy of 99.42% and test-accuracy of 99.12%. The CNNs with 7×7 convolutional kernels achieved an average validation-accuracy of 99.52% and test-accuracy of 99.27%.

Table A.6: CNNs with differently sized convolutional kernels on CIFAR10.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	3	44×44	-	-	-	-
1	Conv	50	44×44	3×3, 5×5, 7×7	1	ReLU	1, 2, 3
-	Max-pool	50	22×22	2×2	2	-	0
2	Conv	100	22×22	3×3, 5×5, 7×7	1	ReLU	1, 2, 3
-	Max-pool	100	11×11	2×2	2	-	0
3	Conv	150	11×11	3×3, 5×5, 7×7	1	ReLU	1, 2, 3
-	Max-pool	150	5×5	2×2	2	-	0
4	FC	-	500	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.6 achieved an average train-accuracy of 99.43% on CIFAR10. The CNNs with 3×3 convolutional kernels achieved an average validation-accuracy of 75.36% and test-accuracy of 73.02%. The CNNs with 5×5 convolutional kernels achieved an average validation-accuracy of 75.64% and test-accuracy of 73.78%. The CNNs with 7×7 convolutional kernels achieved an average validation-accuracy of 74.26% and test-accuracy of 73.12%.

Table A.7: Standard three convolutional layer CNNs with added fully connected layers on MNIST.

Layer		**Channels**	**Size**	**Kernel Size**	**Stride**	**Activation**	**Padding**
Input	Image	1	40×40	-	-	-	-
1	Conv	10	36×36	5×5	1	ReLU	0
-	Max-pool	10	18×18	2×2	2	-	0
2	Conv	20	15×15	4×4	1	ReLU	0
-	Max-pool	20	7×7	2×2	2	-	0
3	Conv	30	5×5	3×3	1	ReLU	0
-	Max-pool	30	2×2	2×2	2	-	0
4	FC	-	120	-	-	-	-
5	FC	-	120	-	-	-	-
Output	FC	-	10	-	-	-	-

The CNNs in Table A.7 achieved an average train-accuracy of 99.99%, validation-accuracy of 99.40% and a test-accuracy of 99.26% on MNIST.

Table A.8: CNNs with differently sized feature maps on MNIST.

Layer		Channels	Size	Kernel Size	Stride	Activation
Input	Image	1	40×40	-	-	-
1	Conv	28, 16, 10	38×38, 37×37, 36×36	3×3, 4×4, 5×5	1	ReLU
-	Max-pool	28, 16, 10	19×19, 18×18, 18×18	2×2	2	-
2	Conv	56, 31 ,20	17×17, 15×15, 14×14	3×3, 4×4, 5×5	1	ReLU
-	Max-pool	56, 31 ,20	8×8, 7×7, 7×7	2×2	2	-
3	Conv	83 , 47, 30	6×6, 4×4, 3×3	3×3, 4×4, 5×5	1	ReLU
-	Max-pool	83 , 47, 30	3×3, 2×2, 1×1	2×2	2	-
4	FC	-	500	-	-	-
Output	FC	-	10	-	-	-

The CNNs in Table A.8 achieved an average train-accuracy of 99.96% on MNIST. The CNNs with 3×3 convolutional kernels achieved an average validation-accuracy of 99.28% and test-accuracy of 99.14%. The CNNs with 4×4 convolutional kernels achieved an average validation-accuracy of 99.26% and test-accuracy of 99.22%. The CNNs with 5×5 convolutional kernels achieved an average validation-accuracy of 99.3% and test-accuracy of 99.18%. We hide the Padding column to fit the table on the page, the CNNs use no padding.

Table A.9: CNNs with differently sized feature maps on CIFAR10.

Layer		Channels	Size	Kernel Size	Stride
Input	Image	3	44×44	-	-
1	Conv	320, 180, 115, 80	42×42, 41×41, 40×40, 39×39	3×3, 4×4, 5×5, 6×6	1
-	Max-pool	320, 180, 115, 80	21×21, 20×20, 20×20, 19×19	2×2	2
2	Conv	520, 293, 188, 130	19×19, 17×17, 16×16, 14×14	3×3, 4×4, 5×5, 6×6	1
-	Max-pool	520, 293, 188, 130	9×9, 8×8, 8×8, 7×7	2×2	2
3	Conv	800, 450, 288, 200	7x7, 5×5, 4×4, 2×2	3×3, 4×4, 5×5, 6×6	1
-	Max-pool	800, 450, 288, 200	3×3, 2×2, 2×2, 1×1	2×2	2
4	FC	-	500	-	-
Output	FC	-	10	-	-

The CNNs in Table A.9 achieved an average train-accuracy of 98.47% on CIFAR10. The CNNs with 3×3 convolutional kernels achieved an average validation-accuracy of 77.2% and test-accuracy of 76.74%. The CNNs with 4×4 convolutional kernels achieved an average validation-accuracy of 76.28% and test-accuracy of 75.94%. The CNNs with 5×5 convolutional kernels achieved an average validation-accuracy of 75.28% and test-

accuracy of 74.72%. The CNNs with 6×6 convolutional kernels achieved an average validation-accuracy of 75.19% and test-accuracy of 74.68%. We hide the Padding and the Activation column to fit the table on the page, the CNNs use no padding and all use ReLU activation functions.