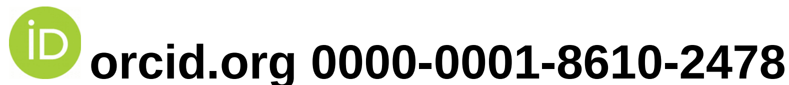
Contrasting Convolutional Neural Networks with alternative architectures for transformation invariance

Coenraad Mouton


orcid.org 0000-0001-8610-2478

Dissertation accepted in fulfillment 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: 25926713

Declaration

I, Coenraad Mouton hereby declare that the dissertation entitled "Contrasting Convolutional Neural Networks with alternative architectures for transformation invariance" is my own original work and has not already been submitted to any other university or institution for examination.

C Mouton

Student number: 25926713

Signed on the 8th day of December 2020 at Potchefstroom.

Acknowledgements

The author of this dissertation conducted the research contained within as a member of Multilingual Speech Technologies (MuST), a research group of the North-West University which specialises in deep learning. I would like to thank the following individuals for their support throughout the research process:

- Professor Marelie Hattingh Davel - Supervisor. The insights, exchange of ideas, reviews, corrections, and attention to detail provided by her were unparalleled, and I am very thankful to have had her guidance.
- Professor Etienne Barnard - Mentor. The technical (as well as philosophical) insight provided by him was invaluable, and I believe this research would've taken considerably longer without his guidance.
- Christiaan Myburgh - Good friend and co-student. Throughout the completion of our master's degrees we have exchanged several hypothesis, ideas, and a healthy amount of banter.
- Tian Theunissen - Good friend and MuST PhD student. His suggestions and mentoring was exceptionally helpful.
- Richard Zhang - Researcher at Adobe, San Francisco. Throughout this process he has answered several of my questions, for which I am very thankful.
- Ulrike Janke - Project manager of MuST. Her help in terms of administrative burdens, planning, and emotional support was very helpful, and made my work much, much easier (even when my response to emails was less than timely).

Finally, I would like to thank the MuST group at large. The exchange of ideas and principles within the group has shaped me into a better, more inquisitive researcher, and in fact a better person.

Abstract

Convolutional Neural Networks (CNNs) have become the standard for image classification tasks, however, they are not completely invariant to transformations of the input image. We empirically investigate to which degree CNNs can handle transformed input images, and also compare their abilities to multilayer perceptrons (MLPs) and spatial transformer networks (STNs). We measure invariance to three affine transformations, namely: translation, rotation and scale; and specifically focus on translation.

The lack of translation invariance in CNNs is attributed to the use of stride which subsamples the input, resulting in a loss of information, and fully connected layers which lack spatial reasoning. We first theoretically show that stride can greatly benefit translation invariance given that it is combined with sufficient similarity between neighbouring pixels, a characteristic which we refer to as *local homogeneity*. We then empirically verify this hypothesis, and also observe that this characteristic is dataset-specific, which dictates the required relationship between pooling kernel size and stride for translation invariance. Furthermore we find that a trade-off exists between generalization and translation invariance in the case of pooling kernel size and stride, as larger kernel sizes and strides lead to better invariance but poorer generalization.

We then compare the translation, scale, and rotation invariance of CNNs to STN-CNNs and MLPs. As expected, we find that MLPs fair far worse than CNNs and STN-CNNs in terms of transformation invariance and generalization. We find that STNs can improve the transformation invariance of a CNN architecture, given that it is exposed to enough transformed samples during the training process. Furthermore, we observe that without explicit regularization, STNs do not provide any benefits over CNNs in terms of generalization ability.

Keywords: *Convolutional Neural Network, Spatial Transformer Network, transformation invariance, scale invariance, rotation invariance, translation invariance, architectural comparison, subsampling*

Contents

List of Figures **ix**

List of Tables **xi**

List of Acronyms **xiii**

1 Introduction **1**

1.1 Background 1

1.1.1 Computer vision and machine learning 1

1.1.2 Transformation invariance 3

1.1.3 Translation equivariance 4

1.2 Problem statement 4

1.3 Project scope 5

1.4 Research questions 5

1.5 Objectives of the study 6

1.6 Research methodology 6

1.7 Dissertation overview 7

1.8 Publications 8

2 Background **9**

2.1 Introduction 9

2.2 Multilayer Perceptrons 10
2.2.1 Architecture 10
2.2.2 Training and data set processing 11
2.3 Convolutional Neural Networks 13
2.4 Spatial Transformer Networks 17
2.5 The importance of transformation invariance 20
2.6 Related work 22
2.6.1 Translation invariance 22
2.6.2 Rotation and scale invariance 23
2.6.3 Measuring invariance 24
2.7 Conclusion 25

3 Experimental Setup **26**

3.1 Introduction 26
3.2 Data set selection 27
3.3 Architecture selection and training 29
3.3.1 CNN architecture 29
3.3.2 Spatial transformer architecture 30
3.3.3 MLP architecture 31
3.3.4 Training protocol and hyperparameter selection 32
3.4 Measuring invariance 33
3.4.1 Metrics 34
3.4.2 Method of comparison 35
3.5 Verification 37
3.6 Conclusion 38

4 Translation Invariance - A theoretical perspective **39**

4.1 Introduction 39

4.2 Invariance and equivariance 40

4.2.1 Subsampling and downsampling 42

4.3 Signal movement, signal similarity, and local homogeneity 45

4.3.1 Shiftability 45

4.3.2 Signal similarity and signal movement 48

4.3.3 Local homogeneity 48

4.4 Conclusion 49

5 Translation Invariance - An empirical analysis **51**

5.1 Introduction 51

5.2 Experimental setup 52

5.3 Pooling and subsampling 53

5.4 Anti-aliasing 55

5.5 Global average pooling 57

5.6 Learned invariance 59

5.6.1 Data translation with MNIST 61

5.6.2 Data translation with CIFAR 63

5.7 Translation invariance and generalization 64

5.8 Architecture comparison 68

5.9 Conclusion 71

6 Spatial Transformer Networks **74**

6.1 Introduction 74

6.2 An overview of spatial transformer networks 75

6.3 Experimental setup 76
6.4 Translation invariance 78
6.5 Rotation invariance 81
6.5.1 Overview of rotation invariance 82
6.5.2 Rotation invariance - MNIST 82
6.5.3 Rotation invariance - CIFAR10 85
6.5.4 Conclusion 88
6.6 Scale invariance 89
6.6.1 Overview of scale invariance 89
6.6.2 Up-scaling 90
6.6.3 Down-scaling 92
6.6.4 Conclusion 94
6.7 Generalization of spatial transformer networks 95
6.8 Conclusion 98

7 Conclusion **100**

7.1 Introduction 100
7.2 Main findings 100
7.3 Addressing research questions 103
7.4 Practical implications 104
7.5 Future research 105
7.6 Generalized conclusion 106
References 108

List of Figures

2.1 Simple example MLP architecture for binary classification 12

2.2 Spatial transformer module (from [10]) 18

3.1 Typical examples of MNIST samples for each class 28

3.2 Typical examples of CIFAR10 samples for each class 28

4.1 2D Convolution of an untranslated and translated MNIST sample with a 3×3 convolution filter 42

4.2 Downsampling with four 5×5 convolution filters of a trained network for a translated and untranslated MNIST sample. 44

4.3 Shiftability maps for arbitrary MNIST sample: white indicates pixel positions that are shiftable, black positions are not. 47

5.1 MCS comparison for MNIST architectures with varying subsampling . . . 54

5.2 Anti-aliasing method preserving max pooling, from [43] 56

5.3 Weight visualization for three MNIST output nodes 60

5.4 Translated weight visualization for three MNIST output nodes 61

5.5 MCS comparison for MNIST architectures with data translation 62

5.6 MCS comparison for CIFAR architectures with data translation 63

5.7 PTop1 change for translation invariance of selected MNIST architectures . 69

5.8 PTop1 change for translation invariance of selected CIFAR architectures . 70

6.1 Spatial transformer network layout 76

6.2 PTop1 change translation invariance comparison for CIFAR10 architectures without train set translation 79

6.3 PTop1 change translation invariance comparison for CIFAR10 architectures with train set translation (5 pixel maximum train set shift) 80

6.4 PTop1 change rotation invariance comparison for MNIST architectures without train set rotation 83

6.5 Two class 6 MNIST samples angled differently 84

6.6 PTop1 change rotation invariance comparison for MNIST architectures with train set rotation (45 degrees maximum train set rotation) 84

6.7 PTop1 change rotation invariance comparison for CIFAR10 architectures without train set rotation 85

6.8 Two CIFAR samples from the “plane” class angled differently 86

6.9 PTop1 change rotation invariance comparison for CIFAR architectures with train set rotation (45 degree maximum train set rotation) 87

6.10 PTop1 change up-scale invariance comparison for CIFAR architectures without train set scaling 90

6.11 PTop1 change up-scale invariance comparison for CIFAR architectures with train set scaling (150% maximum up-scaling of the train set) 91

6.12 PTop1 change down-scale invariance comparison for CIFAR architectures without train set scaling 92

6.13 PTop1 change down scale invariance comparison for CIFAR architectures with train set scaling (75% minimum down-scaling of the train set) 93

List of Tables

3.1 MNIST and CIFAR10 data set parameters 27

3.2 Example CNN architecture with three convolution (Conv) and pooling (Pool) layers, as well as two fully connected (FC) layers 30

3.3 Spatial transformer localisation network for MNIST with convolution (Conv), pooling (Pool), and fully connected (FC) layers 31

3.4 Spatial transformer localisation network for CIFAR10 with convolution (Conv), pooling (Pool), and fully connected (FC) layers 31

4.1 Dense and strided convolution of a one-dimensional input with a kernel [1,0,1], and shifted variants 41

4.2 Subsampling factor of four for an arbitrary input 46

4.3 Subsampling of a locally homogeneous signal 49

5.1 Baseline CNN architecture w/o subsampling for MNIST and CIFAR with three convolution (Conv) and pooling (Pool) layers, and two fully connected (FC) layers 52

5.2 Stride of max pooling layers for MNIST architectures 53

5.3 Mean cosine similarity for CIFAR10 networks with varying subsampling and max pooling kernel sizes - 10 pixel range (SE < 0.04) 54

5.4 Mean cosine similarity for MNIST and CIFAR10 networks with and without anti-aliasing (AA) for a maximum shift of 10 pixels 57

5.5 Simple MNIST CNN architecture for weight distribution visualization . . . 60

5.6 Test accuracy for MNIST networks with varying subsampling factors trained on translated data, averaged over three initialization seeds 62

5.7 Test accuracy for CIFAR10 networks with varying kernel sizes trained on translated data, averaged over three initialization seeds 64

5.8 Test accuracy for CIFAR10 networks with varying subsampling and pooling kernel size (SE < 0.22) 65

5.9 Mean cosine similarity for CIFAR10 networks with varying subsampling and max pooling kernel sizes - 10 pixel range (Repeat of Table 5.3) 65

5.10 MNIST and CIFAR10 test accuracy with and without anti-aliasing (AA) . 67

5.11 Test accuracy for MNIST and CIFAR10 Models, averaged over three initialization seeds 70

6.1 CNN architecture for MNIST with three convolution (Conv) and pooling (Pool) layers, and two fully connected (FC) layers 77

6.2 CNN architecture for CIFAR with three convolution (Conv) and pooling (Pool) layers, and two fully connected (FC) layers 77

6.3 Standard test set accuracy for CIFAR10 models trained on different train sets, averaged over three initialization seeds (SE < 0.29) 95

6.4 Transformed test set accuracy for CIFAR10 models trained on transformed train sets, averaged over three initialization seeds (SE < 0.32) 96

6.5 Standard and transformed test set accuracy for CIFAR10 networks trained on data with mixed transformations, averaged over three initialization seeds 97

List of Acronyms

AA Anti-aliasing

ANN Artificial Neural Network

CIFAR Canadian Institute For Advanced Research

CNN Convolutional Neural Network

CV Computer vision

FC Fully connected

GAP Global average pooling

MCS Mean Cosine Similarity

MLP Multilayer Perceptron

MNIST Modified National Institute of Standards and Technology database

PTop1 Probability of top 1 change

ReLU Rectified Linear Unit

SS Subsampling

STN Spatial Transformer Network

Chapter 1

Introduction

"It's no use, Mr. James—it's turtles all the way down." - J. R. Ross [1]

1.1 Background

1.1.1 Computer vision and machine learning

Computer vision (CV) is a wide field of study concerned with, among others, creating meaningful and explicit descriptions of features in images [2]. From an engineering perspective, computer vision is focused on developing algorithmic methods and models for extracting information from two or three dimensional image data. Object recognition is a specific computer vision task that attempts to recognize and classify objects in an image according to a database of known object classes. Object classification is considered a non-trivial problem and many techniques have been developed in an attempt to solve this problem (with varying degrees of success). Traditional algorithms for object recognition focus on extracting and encoding key features from images which are then used for classification, this includes methods such as SIFT (Scale-Invariant Feature Transform), SURF

(Speeded-Up Robust Features), and other digital filtering techniques [3].

Tremendous progress has been shown in recent years in both object recognition and classification by making use of machine learning (ML) techniques. More specifically, artificial neural networks (ANNs) have greatly improved performance on benchmark datasets such as MNIST (Modified National Institute of Standards and Technology database) [4].

Artificial neural networks, or more specifically deep neural networks (DNNs), are trained for image classification by using a number of labeled examples (a dataset) to fit a large parameter model to a specific problem.

While standard deep neural networks, commonly referred to as multilayer perceptrons (MLPs), are capable of high accuracy on simple image classification problems (such as the aforementioned MNIST dataset), they are less successful with more difficult tasks such as the CIFAR10 (Canadian Institute For Advanced Research) dataset [5]. While there are varying reasons for this, the main problem is that MLPs are not capable of accurately encoding spatial information found in an image. This implies that these models do not thoroughly understand the location-based relation between separate pixels of an image. This spatial information is crucial to consider in any computer vision task.

To address this problem, a new neural network architecture was introduced in part by Yann LeCun [6], now commonly referred to as a Convolutional Neural Network (CNN). This architecture makes use of the well understood signal processing tools of 2D-convolution and downsampling. Whereas in traditional computer vision methods convolutional filters (kernels) are designed by hand for a specific purpose (such as edge detection), CNNs use kernel values as learnable parameters, which are trained to extract abstract features from images through gradient descent [7]. The use of these kernels in combination with pooling kernels, which apply a fixed localized operation, allow the model to more accurately encode spatial information in images.

CNNs have lead to a tremendous increase in classification accuracy on difficult datasets such as CIFAR10, CIFAR100, and ImageNet [8].

1.1.2 Transformation invariance

A transformation invariant function is a function whose output for a specific input remains unchanged when a transformation is applied to this input. In terms of object classification, a transformation invariant classification model would output the same prediction for a certain image regardless of any translation, shear, rotation or other transformations applied to it. In real world computer vision applications, transformation invariance can greatly improve the effectiveness of a model. Should such a model encounter images that do not strictly conform to the data on which it is trained (for example varying scale, rotation, or position) it can still perform as expected.

Currently the degree to which deep learning models are invariant to transformation is poorly understood. While convolutional neural networks do exhibit some invariance to small affine transformations, recent work has shown that state-of-the-art architectures are still very susceptible to minute changes in input. Azulay and Weiss [9] conclude that well-optimized CNN architectures exhibit dramatic drops in prediction confidence when certain inputs are translated by only a single pixel. In response to this apparent lack of invariance shown by deep learning models, a new architectural unit was developed by Jaderberg et al. (from Google Deepmind), known as Spatial Transformer Networks (STNs) [10].

STNs are neural networks that contain spatial transformer modules. A spatial transformer model can be inserted into any deep learning architecture as an intermediary unit that attempts to apply corrective transformations to inputs and/or intermediary feature maps. These transformer models are trained through the same process of backwards propagation (minimizing a specific loss function and updating parameters accordingly) as the rest of the network architecture. However, to date adequate comparisons between STNs, CNNs, and a combination of both has not been made in terms of their true transformation invariance. In conjunction with this, the specific parameters that contribute to transformation invariance in both architectures are not fully understood. This apparent lack of understanding can in part be attributed to the fact that it is difficult to quantify exactly how invariant a model is to a specific transformation.

Even if only affine transformations are considered (transformations where the collinearity property of an input is preserved) different transformations can have vastly different effects on the prediction accuracy of deep learning models. This greatly complicates the process of quantifying transformation invariance. Furthermore, merely observing accuracy does not provide any intuition as to why certain architectures perform better with regards to certain transformations than others.

1.1.3 Translation equivariance

In convolutional neural networks, convolution and pooling kernels are *equivariant* to translation. This property stems from the fact that convolution and pooling kernels are shifted over an input to provide an output. Translation equivariance implies that shifting the input results in an equal shift of the output. This unique property of translation in CNNs implies that the specific transformation of translation requires special consideration, as the equivariance property does not hold for other affine transformations such as rotation or scale.

1.2 Problem statement

Transformation invariance in object classification models such as convolutional neural networks and spatial transformer networks are poorly understood. The degree of invariance to transformations that a model exhibits is difficult to quantify. It is well known that invariance to transformation is very desirable in real world applications of computer vision, but no adequate comparison between different methods has been made, nor has it been established precisely which architectural elements contribute to invariance.

1.3 Project scope

Given the problem statement above, the initial scope of the research is restricted to a limited number of architecture types, datasets and transformations:

- **Architecture types:** Only deep learning models are considered, and it is restricted to:
  - Convolutional Neural Networks (CNNs)
  - Spatial Transformer Networks (STNs)
  - Multilayer Perceptrons (MLPs)
- **Datasets:** Only two publicly accessible datasets are considered, namely: MNIST and CIFAR10.
- **Transformations:** Only affine transformations are considered, meaning transformations where the collinearity of the image is preserved, and is limited to:
  - Translation
  - Rotation
  - Scaling

1.4 Research questions

With the aim to investigate the transformation invariance of different deep learning architectures the following research questions are posed:

- Which architectural elements are responsible for a given CNN's translation invariance (or lack thereof)?
- How do methods proposed by other authors fair in ensuring translation invariance?

- How does the transformation invariance of different deep learning architectures compare to one another?
- How do transformation invariance and generalization relate to each other?

1.5 Objectives of the study

Considering the research questions, the study objectives can be summarised as follows:

- Explore which architectural elements of a CNN contribute to translation invariance specifically.
- Compare the translation invariance of different CNN architectures.
- Compare the transformation invariance between selected deep learning architectures.
- Explore why certain architectures show more/less invariance than others.
- Explore the relationship between transformation invariance and generalization.

1.6 Research methodology

This study consists of:

- **Literature review:** An in-depth study of convolutional neural networks and other supporting architectures is necessary. In conjunction with this it is required to ascertain which aspects of transformation invariance in neural networks have already been well established and how they can be applied to this problem.
- **Experimental development:** Based on the literature review, a method for comparison and quantification of transformation invariance can be selected. This process

will be done by exploring possible comparison methods and running select experiments.

- **Codebase development:** Once a proper method has been found and is supported by anecdotal evidence, an experimental test bench can be derived and programmed.
- **Experimentation:** Given a well working system, architectures can be compared and several experiments assessing their transformation invariance can be conducted. This allows one to select candidate architectures for further assessment. Once a baseline has been established other factors such as prior data transformation can also be considered and assessed.
- **Assessment of experimental findings:** Once the experiments are completed the results can be interpreted, this would allow one to methodically increase the complexity of the experiments and guide the avenue of research. At the end of this process a conclusion will be formulated which answers the research questions posed earlier.

1.7 Dissertation overview

This dissertation aims to compare the transformation invariance of several deep learning architectures through the use of empirical experiments and relevant theory. The document is structured in the following manner:

- Chapter 2 provides relevant background information on deep learning, convolutional neural networks, spatial transformer networks, and transformation invariance. Along with this we provide a brief overview on related work regarding transformation invariance done by other authors.
- Chapter 3 describes the experimental setup that we use to measure transformation invariance, including the relevant metrics, training protocols, and architecture selections.

- Chapter 4 focuses on the specific affine transformation of translation. We study the relationship between translation equivariance and invariance, and show which architectural elements have a unique effect. We also introduce a novel characteristic of a dataset which affects translation invariance, *Local Homogeneity.*

- Chapter 5 verifies the theoretical understanding of translation invariance studied in the previous chapter through empirical study. We compare the translation invariance of many CNN architectures, and also compare their generalization ability.

- Chapter 6 measures the transformation invariance and generalization of several spatial transformer networks. In addition to this we compare these networks to normal CNN and MLP architectures.

- Chapter 7 concludes our research by summarising our findings and discussing their implications, along with avenues for future research.

1.8 Publications

Some of this dissertation's content has been accepted for presentation at SACAIR2020, and selected for inclusion in Communications in Computer and Information Science (LNCS sub-series CCIS). Specifically, parts of Chapters 4 and 5 are included in "Stride and translation invariance in CNNs" [11]. This study also contributed to "Tracking translation invariance in CNNs" [12].

Chapter 2

Background

"I do not fear truth. I welcome it. But I wish all of my facts to be in their proper context."
– Gordon B. Hinckley [13]

2.1 Introduction

In this chapter we provide an overview of the different deep learning architectures that are investigated in this study, as well as background information required to understand the transformation invariance capabilities of these architectures. Furthermore, we also provide an overview of work done by other authors in terms of transformation invariance.

In Section 2.2 we explain the functioning of MLPs, as well as their training process. Following this we provide a brief overview of convolutional neural networks in Section 2.3, and examine the functional units that they are composed of. In Section 2.4 we explain the architectural elements of spatial transformer networks and their functioning. Furthermore, in Section 2.5, we discuss why transformation invariance is important, and how it occurs in real world applications of object recognition systems. Finally, in Section

2.6, we provide an overview of related work with regards to transformation invariance in convolutional neural networks.

2.2 Multilayer Perceptrons

Deep learning has revolutionized the field of machine learning, as artificial neural networks (ANNs) have shown tremendous advances in areas such as computer vision, speech recognition, language translation, and many others [14]. The core architecture of ANNs is that of the multilayer perceptron (MLP), also referred to as a feedforward neural network [15]. In this chapter we provide a very brief overview of the functioning of MLPs as used in image classification problems.

Neural networks attempt to approximate a function for a specific problem by defining a mapping in the form of $\hat{y} = f(x; \Theta)$, where Θ are the parameters learned from a given data set to provide the best approximation [15], and $\hat{y}$ is the approximated output given an input x. We first study the architecture of MLPs, before examining the training process.

2.2.1 Architecture

The core element of an MLP is that of the neuron. Neurons are computational units that perform a weighted summation of its inputs before adding a bias term, and several neurons are combined in what is known as a hidden layer [16]. Each neuron provides an output known as an *activation*, and each neuron in a given layer can be mathematically defined as follows (as adapted from [17]):

$$a_j = \sum_{i=1}^{N} w_{ji} x_i + b_j \tag{2.1}$$

Where a_j is the activation for node j, with input x_i, and corresponding weight w_{ji}, for a total number of inputs N. Along with the weighted summation, each node also has a bias

term, b_j. The activation is then transformed using a nonlinear activation function [17], denoted by σ, and z_j the output for node j:

$$z_j = \sigma(a_j) \tag{2.2}$$

While there are many activation functions which can be used, one example is the rectified linear unit (ReLU) [18], as shown in Equation 2.3 (where x is an arbitrary input).

$$\sigma(x) = max(0, x) \tag{2.3}$$

MLPs consist of several fully connected layers, meaning each node of one layer is connected to every node of the following layer. The first layer is known as the input layer, in terms of image classification this would be a two-dimensional image that is flattened into a one-dimensional vector. Following this there are one or multiple hidden layers consisting of several neurons (also referred to as nodes), where the output of each node serves as input to each node in the following layer. The number of neurons in a hidden layer is known as the width of the layer, while the number of hidden layers is known as the depth of the network. After the hidden layers a final layer known as the classification layer, or output layer, assigns a confidence value to each class for a given input. Each node in the output layer corresponds to a specific class, where higher values indicate a higher confidence that the given input belongs to the corresponding class. An example of a simple binary classification network with four input features and two hidden layers is shown in Figure 2.1.

2.2.2 Training and data set processing

In supervised image classification tasks, a data set contains several samples of each class, each labeled accordingly. Data sets are usually split into three separate sets: train, validation, and test. The train set is used to train the network, meaning to learn parameter values for accurate classification, while the validation set is used to evaluate the perfor-

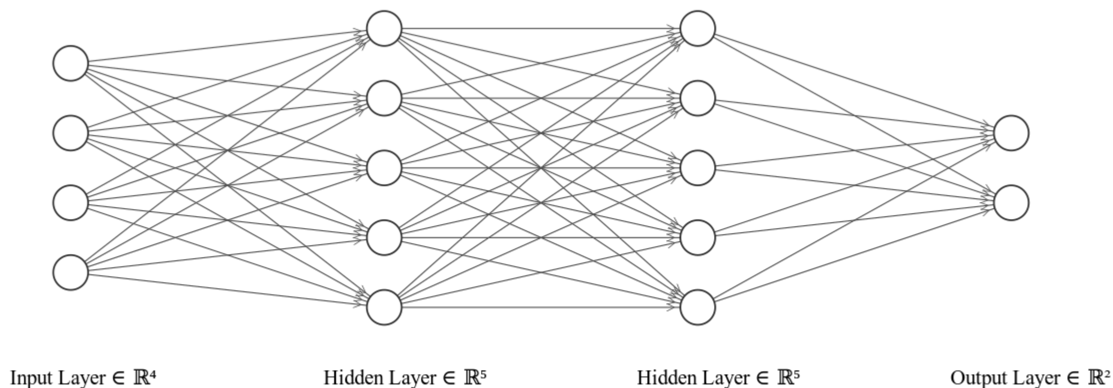


Figure 2.1: Simple example MLP architecture for binary classification

mance of the model on unseen samples, and hyperparameters are adjusted to ensure the highest validation accuracy. Finally, the test set is used to provide an unbiased performance evaluation.

The process of training (when using gradient descent [7] [19]) is done in two steps: 1) forward propagation and 2) backwards propagation. During forward propagation, a batch of samples is passed through the network, providing a prediction output vector $\vec{\hat{y}}$. The network's prediction is then compared to the ground truth value vector provided by the sample labels, $\vec{y}$, by using a cost function [7]. The cost function calculates the error between these two vectors, which is commonly referred to as a loss value. Whilst many different cost functions can be used, common examples are mean squared error [20], and cross entropy loss [21].

The following step in the training process is to minimize the loss value by adjusting the network parameters accordingly [22]. This is achieved through a process of gradient descent, where the gradient of each parameter is calculated with respect to the cost function [22] [7]. The term back propagation refers to the process whereby these gradients are calculated [19]. Each parameter value is then updated according to the gradient of the loss with regard to the parameter, to minimize the resulting loss value, multiplied by a scalar referred to as the learning rate. The process of forward propagation, back propagation, and updating the parameters is repeated for the entire training set and thus

completes one training cycle, or epoch.

In summary, MLPs are powerful architectures which learn a function approximation from labeled samples through a process of gradient descent. In the following section we examine convolution neural networks.

2.3 Convolutional Neural Networks

While MLPs are certainly very powerful, they struggle to achieve high performance on difficult image classification tasks. To address this problem, modern day machine learning image classification problems are usually solved through the use of convolutional neural networks (CNNs), originally attributed to Yann LeCun [6]. CNNs combine the traditional image classification method of identifying features through the use of convolutional kernels with that of gradient based learning [23]. In this section we explain the functioning of CNNs and which parameters they are subject to.

Broadly speaking, CNNs consist of two unique stages: (1) Convolution stage, which consists of convolution and pooling kernels; and (2) Fully connected stage, which does the final classification. We first turn our attention towards the convolution stage.

Firstly one must understand the convolution operation, which consists of an input, I, and a kernel, K, and is usually denoted with an asterisk [24]. The two dimensional discrete convolution operation for an input of size $m \times n$, with input features i and j is defined in equation 2.4.

$$(I \circledast K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{2.4}$$

In practice, convolution is usually implemented with a flipped kernel: The convolution operation shifts the kernel over the input, multiplies overlapping values with each other, and then takes the summation; the resulting output is commonly referred to as a feature map [24]. A well known example of using convolution for feature detection is that of the

Sobel–Feldman operator [25] which is used to detect edges in images. However, in the case of CNNs the kernel parameters, or *weights*, are learnable parameters, which along with the parameters of the fully connected stage, are learned from data through a process of gradient descent. The question then arises why convolutional kernels are generally better at feature detection than that of standard MLPs; we briefly explore these reasons as stated by Goodfellow et al. [26].

1. Sparse interactions - As the kernel is generally smaller than the input, every input feature does not interact with every parameter as is the case in MLPs [27]. This means that the size of the feature representations become smaller throughout the network. This greatly reduces the computational complexity involved.

2. Parameter sharing - Given that the kernel is shifted over the input, the same parameter is used multiple times for different inputs, whereas MLPs have a different parameter for every single input feature [28]. This means that memory requirements are drastically less, and the computation is more statistically efficient.

3. Equivariant representations - Convolution is equivariant to translation, meaning that translating the input causes an equal translation in the output. This is due to the fact that convolution is inherently translation, as kernels are shifted over an input to produce an output feature map [29].

While these qualities provide great advantages over MLPs, the greatest distinction is that CNNs capture spatial dependencies in an image through the application of filters [30]. Given that CNNs treat images as two dimensional matrices, spatial relations between pixels are kept when convolutional kernels identify features, whereas in the case of MLPs images are flattened into a one dimensional vector prior to any feature identification.

When using convolution kernels in a CNN they are combined in a *convolution layer*, which consists of several convolutional filters [31]. Convolutional layers stack several feature maps on top of each other, meaning that the filters being applied are not purely two dimensional but also have a specific depth [31]. The term *input channels* refers to the

depth of the filters being used, and *output channels* to the number of filters applied in a single convolutional layer [31]. For example, given input images with three color channels (RGB), the first convolution layer consists of convolution kernels of size $H \times W \times 3$, where H is height and W is width, but may apply several such three dimensional filters resulting in an output that is also three dimensional. Alternatively, each convolutional layer can also consist of a single filter which is applied independently to each channel, or alternate filters independently applied per channel. This further implies that each subsequent convolutional layer has as many input channels as the number of output channels of the previous. We summarise the hyperparameters which specify a convolutional layer below, as explained by Yamashita et al. [32]:

- Kernel Size - The size of the kernels, generally kernels are square ($m \times m$), but rectangular kernels ($m \times n$) are also possible.
- Kernel Stride - The distance between two successive kernel positions, or put otherwise the size of the shift of the kernel between each separate convolution operation.
- Padding - The application of a convolutional kernel downsamples the input, as information along the edges of the image is lost. If this is not desirable, padding is used to increase the size of the resulting feature map. While there are several forms of padding, that of *zero padding* is most common, which means that zero values are added to the edges of the image.
- Input Channels - The depth of the convolutional filters being applied.
- Output Channels - The number of convolutional filters applied.

To better understand the effects of kernel size and stride, consider the output width of a layer W_l given the previous layer width W_{l-1}, in the case of two-dimensional convolution:

$$W_l = \frac{W_{l-1} - k_w + 2p}{s} + 1 \tag{2.5}$$

where k_w is the kernel width, p is padding, and s is stride, and the same equation holds for height. During convolution, the kernel is never centered on the pixels located at the

edges of each feature map, therefore each convolution layer leads to a reduction in size, depending on the size of the kernel. Furthermore, a stride greater than one also greatly reduces the spatial dimensions. Padding is commonly used to offset size reduction, and the amount of padding used is usually described by one of three terms, as stated by Kayhan and Gemert [33]:

- Valid padding - No padding is applied.
- Same padding - The input is padded so that the output is equal in size to the input.
- Full padding - Sufficient padding to ensure that every parameter of the kernel is applied to every value of the input (disregarding stride).

In addition to convolution layers, the convolution stage of a CNN also typically makes use of pooling layers. Pooling kernels are similar to convolution kernels as they are also shifted across an input to calculate the output feature map, with two key exceptions: Firstly, pooling kernels apply a fixed operation to each region, meaning they contain no learnable parameters [32]. Secondly, each pooling operation is applied to each feature map (channel) separately, unlike convolution which can be applied across channels simultaneously [32]. Due to these reasons, pooling layers are specified by the same hyperparameters as convolution layers (kernel size, kernel stride, padding), but the number of channels remain unchanged [32]. Two common forms of pooling is that of max and average pooling, where max pooling outputs the maximum value of a given region and disregards other values, while average pooling outputs the average value of the region [34].

Pooling layers serve a dual purpose: Pooling reduces the spatial dimension of each feature map, therefore reducing the time complexity of each subsequent convolution operation, and also provides local invariance to small transformations [34]. Given that pooling provides a summary statistic for a region, it can ensure that a specific feature is detected without pinpointing its exact location [35]. We explore the nature of transformation invariance as related to pooling layers in later chapters.

In a CNN, convolution and pooling layers are combined along with a nonlinear activation

function, such as the rectified linear unit (ReLU) [18]. Generally an activation function is applied after a convolution layer, and after one or several convolution layers a pooling layer is applied. This forms the "convolution stage" of a CNN, which precedes the fully connected stage. After the convolution stage the resulting feature maps are flattened into a one-dimensional vector, which serves as input to fully connected layers. The fully connected layers (also known as dense layers [32]) are essentially a standard MLP, and are responsible for doing the final classification after the convolution stage has done the necessary feature detection.

In conclusion, we have explained the basic building blocks of convolutional neural networks, and their advantages over MLPs. In the following section we take a look at spatial transformer networks.

2.4 Spatial Transformer Networks

As discussed in Chapter 1, CNNs are not completely invariant to transformations of the input image. In response to this apparent lack of invariance, a new architectural unit was developed by Jaderberg et al., known as Spatial Transformer Networks (STNs) [10]. STNs are neural networks that contain spatial transformer modules. A spatial transformer module can be inserted into any deep learning architecture as an intermediary unit that attempts to apply corrective transformations to inputs and/or intermediary feature maps. In this section we provide an overview of the functioning of spatial transformer modules.

Spatial transformer modules consist of three distinct parts: 1) Localisation Network, 2) Grid generator, and 3) Sampler [10]. These three modules are shown in Figure 2.2, where U is an input feature map and V the corresponding output feature map.

We explain the purpose of each module, and its interaction with others.

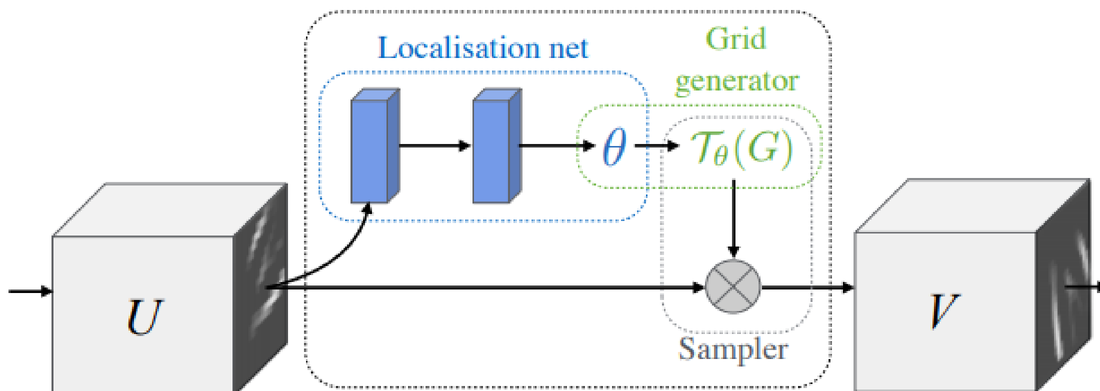


Figure 2.2: Spatial transformer module (from [10])

Localisation Network

The localisation network is responsible for determining the parameters of the transformation that must be applied to correct an input or feature map. This module can consist of any neural network architecture, such as an MLP or CNN, with a final regression layer of parameters $\vec{\theta}$ which specify the transformation [10].

Whilst this module can be configured to produce a wide range of transformation, we focus on affine transformations, which are geometric transformations that preserve collinearity, meaning that lines, planes, and parallelism are preserved. Put otherwise, all points that lie on a line prior to transformation will still lie on the line after transformation [36]. These transformations can be expressed as a matrix multiplication, where $\vec{y'}$ is the transformed output vector, given an input vector $\vec{y}$:

$$\vec{y'} = A\vec{y} \tag{2.6}$$

And A is of the form:

$$A = \begin{bmatrix} \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{21} & \Theta_{22} & \Theta_{23} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Given this, the localisation network outputs six Θ values specifying the affine transformation. The localisation network is trained in an end-to-end fashion along with the rest

of the network through the same process of backwards propagation [10]. Although A allows for three-dimensional transformations, only two-dimensional transformations are considered, as shown later in Equation 2.8. This implies that the same transformation is applied to each channel of the two-dimensional feature map separately.

Grid generator

The transformation specified by the localisation network is not directly applied to the input feature map, instead the grid generator is responsible for specifying the spatial locations where the input must be sampled to provide the transformed output [37].

Firstly, a normalized meshgrid is created, denoted as G_i, which is a set of target indices (x^t, y^t) which cover the entire input feature map [37] [10]. The source coordinates, (x^s, y^s) which specify the input sample points are then calculated using the following formula (for two-dimensional affine transformations, as defined in [10]):

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \tau_\theta(G_i) = A_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{21} & \Theta_{22} & \Theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{2.8}$$

The source coordinates column vector provides the coordinates where the input feature map must be sampled, and is then passed to the next module, the sampler.

Sampler

Given the set of sampling points provided by the grid generator, the input feature map, $U \in R^{H \times W \times C}$, is sampled to produce the transformed output feature map, $V \in R^{H' \times W' \times C}$ [10]. At each source coordinate a sampling kernel is applied on U to provide the pixel value for V. These source coordinates are not guaranteed to be integers, meaning a form of interpolation must be applied, such as bilinear interpolation [38]. Any form of sampling kernel can be applied, so long as it is differentiable, or put otherwise that

gradients can be found with respect to (x^s, y^s), to allow for backwards propagation [10]. The same sampling is applied to each channel, so that each channel is transformed identically [10], furthermore the number of channels for the input feature map stay constant in the output. While spatial transformer modules can be used for cropping, we focus on affine transformations, as such the height and width of the feature map also stays constant.

The three aforementioned units make up the architectural composition of a spatial transformer module, which attempts to learn invariant representations. In the following section, we explain why transformation invariance is necessary in object recognition systems.

2.5 The importance of transformation invariance

We have selected three affine transformations for analysis, namely: translation, rotation, and scale. In this chapter we motivate the importance of each transformation in terms of invariance and how they affect practical applications of image classification.

For the sake of perspective, one must first understand the concept of data set bias, as pointed out by Azulay and Weiss [9]. Commonly used data sets such as ImageNet [39], and especially tiny image data sets such as CIFAR10 and MNIST, are very biased in the spatial location, size, and orientation of objects within the image. This is attributed to photographer's bias, as images are captured with the subject centered within the frame, whilst facing and near to the camera [9]. Khosla et al. [40] states that these are biased samples of the greater visual world, and prevents machine learning architectures from generalizing to other unbiased samples.

In terms of translation invariance, it is of great importance that objects are recognizable regardless of their location within the image canvas. In real world applications it can not be expected that subjects would be centered within the image when classification is required, or alternatively there can exist multiple subjects within the image at different locations. To take a trivial example, consider a camera responsible for detecting wildlife.

It would be erroneous to assume that a deer and bird will usually occur at the same location within the image, as birds would generally be higher (e.g. sitting in a tree), while deer dwell on the ground. If the system is to be effective, it requires a degree of invariance to translation to accurately detect and classify wildlife that enter into view. Should the system purely be trained on biased data samples (e.g. purely close up centered images of birds), it is difficult to imagine it would generalize to other cases.

When considering scale invariance, the importance is very similar to that of translation. To re-use the previous example of wildlife detection, it would be logical to assume that an animal would not always be the same distance from the camera, and the relative size of each individual animal can differ greatly. In order for the system to generalize well, it is required that animals of different sizes, and more importantly animals at different distances from the view point be recognizable. Once again, should the system only be trained on biased samples where the subject is very close to the viewpoint, its ability to detect animals that are farther away can be greatly diminished. Noord and Postma [41] further point out that in the case of scale, the resolution of the image also plays a large role. The resolution of an image indicates it size in pixels, and sufficient resolution is required to represent the fine details of the image. It then follows that images at a lower scale are less detailed representations, and proves to be a formidable problem for image recognition systems [41].

Finally, we consider the case of rotation invariance. While it is true that in some applications the real world tends to conform to our photographer's bias, for example in our wildlife scenario birds tend to be upright, as opposed to lying on their side, this is not true in all cases. Follman and Bottger [42] point out that in biomedical and industrial environments objects can occur in arbitrary orientations (such as objects on a conveyor belt or microscopic images). Furthermore, smaller rotations are common in other applications (such as a deer tilting its head to graze) and must remain recognizable in these positions. Rotation is also distinct from scale and translation, as in some cases it can change the identity of a sample, for example, in digit recognition the digit '9' rotated by hundred-and-eighty degrees becomes the digit '6'.

In summary, data sets that are commonly used to train image recognition systems are biased towards a specific viewpoint, position, and orientation of a subject, and can prevent generalization to cases that do not conform to these biases. We have explained the importance of translation, scale, and rotation invariance, and how these transforms can commonly occur in real world applications. Finally, we emphasise that it is critical that these transformations be taken into account when designing systems for use in the real world.

2.6 Related work

In this section we provide a brief overview of related studies of transformation invariance in convolutional neural networks.

2.6.1 Translation invariance

In convolutional neural networks, convolution and pooling layers are equivariant to translation, implying that a shift of the input results in an equal shift of the output [29]. This property stems from the fact that convolution and pooling kernels are translational in nature, as they are shifted over the image to provide an output feature map. However, Azulay and Weiss [9] point out that this property only applies to dense convolution and pooling, and not to kernels with a stride greater than one. They attribute this to subsampling - by using larger strides information is disregarded, and the equivariance property no longer holds for translation of the input. However, even if no subsampling is used, translation equivariance does not imply translation invariance. Translation invariance refers to the ability of a model to be unaffected by shifts of the input. Azulay and Weiss proposes a solution to this in the form of global average pooling (GAP), which applies an averaging kernel over the entire output feature map.

Zhang [43] points out that when subsampling, the textbook signal processing solution is to apply an anti-aliasing filter prior to subsampling. He then proposes combining

pooling kernels with an intermediary anti-aliasing filter. Zhang states that by filtering before subsampling, the model's equivariance to translation is improved, and thereby also increases its invariance.

In addition to this, Lenc and Vedaldi [44] mathematically define the properties of invariance and equivariance, and further study how visual information is captured and represented in a CNN.

Clearly, there is a relationship between a model's translation equivariance and invariance. However, the precise nature of this relationship is not made clear, and none of the aforementioned authors address how these properties, or their proposed modifications, affect the fully connected layers of a CNN.

Furthermore, besides stride and subsampling, it would appear that downsampling can also have a large effect on translation invariance. Kayhan et al. [33] show that CNNs can learn the specific location of features in an image through using "edge effects" caused by the size of a kernel during convolution or pooling. Or put otherwise, by exploiting the difference in the loss of information that occurs between a translated and untranslated sample.

In Chapter 4 we carefully examine the theory behind translation equivariance and invariance, and how these properties are related. In addition to this, we also address the effects of downsampling. In Chapter 5, we measure the translation invariance of several CNN architectures, and provide empirical results to support our theoretical understanding.

2.6.2 Rotation and scale invariance

Convolutional neural networks are not equivariant to rotation and scale transformations [42], unlike the case of translation. Given that convolution and pooling kernels are not rotated or scaled during the convolution operation, the same kernel parameters are not applied to the same input features when the input is transformed. This implies that the output of a transformed input is distinct from the output of an untransformed

input.

Follmann and Bottger [42] proposes a solution to this lack of equivariance to rotation, by introducing rotational convolution and pooling. In addition to the standard, shifted convolution, each kernel is also rotated. These kernels produce separate feature maps for each angle by which it is rotated, and in doing so are able to learn semi-rotation-invariant representations. Similarly, Xu et al. [45] addresses the problem of scale invariance by creating transformed copies of a standard size convolution kernel, either by up- or down-scaling. They then combine these different scale kernels in a single multi-column CNN.

Other solutions also exist in the form of ensemble techniques. Noord and Postma [41] combine several CNNs, each trained on data with different scales and with unique kernel sizes, and average their predictions for final classification.

Instead of resorting to such "artificial" methods, we measure the rotation and scale invariance of normal CNNs, and compare them to the performance of spatial transformer networks in Chapter 6.

2.6.3 Measuring invariance

There are several methods and metrics that can be used to measure a model's invariance to transformation. While some authors simply measure the error rate of a model on a transformed test set [42] [10], this method is more aligned with a measure of generalization, than invariance specifically.

Kauder-Abrams [46] proposes an alternative method for quantifying the translation invariance of deep learning models. This method involves comparing the output vectors of a specific model for a standard and translated input by means of euclidean distance. By comparing the output vector of a standard image to many translated versions of it, one is able to generate what is known as a translation heat map [46]. These heat maps show the average euclidean distance between the output vectors of translated and untranslated samples in a ten-pixel radius. This method provides great intuition into how deep learning

models are affected by translation, and is much more descriptive than merely comparing accuracy.

Azulay and Weiss [9] use “mean absolute change” - the average change in the top class prediction for translated samples in comparison to their untranslated counterparts. This method is useful as it provides a single scalar value specifying the model’s invariance for a specific translation. Furthermore, they also make use of “probability of top 1 change”, which is a the probability of a change in classification following a one-pixel shift of the input samples.

In Chapter 3 we explain the methods and metrics we employ to measure invariance to transformations, which are closely aligned with the aforementioned techniques.

2.7 Conclusion

In this chapter we have reviewed the three different architectures we use for analysis, namely: multilayer perceptrons, convolutional neural networks, and spatial transformer networks. We provided an explanation of their basic functioning and also briefly explained the training process behind deep learning. We have shown how convolutional neural networks have certain advantages over MLPs, and how spatial transformer networks attempt to learn transform invariant representations when combined with one of the aforementioned architectures.

Following this we pointed out the bias that exists in commonly used image recognition data sets, and how this bias can impede generalization. We examined three affine transformations and explained how these transformations mimic cases that can occur in real world applications of object recognition systems, and why invariance to these transformations are of utmost importance.

Finally we had provided a brief overview of related research done in terms of transformation invariance, and also how invariance can be measured.

Chapter 3

Experimental Setup

"If it disagrees with experiment, it's wrong. In that simple statement is the key to science." - Richard Feynman [47]

3.1 Introduction

In this chapter we explain our selection of data sets and architectures, and also our methods for measuring transformation invariance. We develop a test bed for comparing different architectures, and thoroughly explain which metrics are employed, along with protocols for training and verification.

We select two data sets which are suitable for analysis, in Section 3.2. Following this, in Section 3.3, we identify architectural layouts for all three of our architectures (MLP, CNN, STN) for each data set. Furthermore, we develop a training protocol which ensures that these architectures are adequately and efficiently trained, and we provide motivation for our specific choice of hyperparameters. In Section 3.4 we explain how transformation invariance is measured, and identify which metrics are used, before showing the specific

process that is followed to apply each transformation. Finally, we explain the multiple methods of verification that we use to ensure that our results are accurate and repeatable, in Section 3.5.

3.2 Data set selection

When selecting data sets for our specific analysis we use the following criteria:

1. Commonly used. We wish to compare our results to that of other authors, and also ensure that they are easy to interpret, therefore we limit our research to data sets which are commonly used for analysis purposes.
2. High accuracy. To ensure valid results, we focus on tasks that CNN architectures perform relatively well on. Additionally, for commonly used data sets there are a lot of available literature describing well performing hyperparameters, this allows us to save time on extensive grid searches to ensure our models are well optimized.

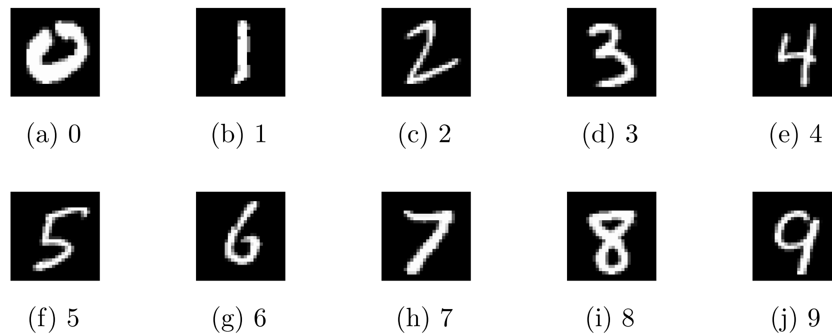
Given this criteria, we opt to use two well-known data sets: MNIST (Modified National Institute of Standards and Technology database) [4] and CIFAR10 (Canadian Institute For Advanced Research) [5]. These data sets are commonly used in deep learning research and well documented, therefore we do not have to do extensive hyperparameter and architecture searches to achieve high performance. The specifics of each data set is summarised in Table 3.1.

Table 3.1: MNIST and CIFAR10 data set parameters

Parameter	**MNIST**	**CIFAR**
Train Set Size	60 000	50 000
Test Set Size	10 000	10 000
Image Size	28x28	32x32
Color Channels	1	3
Classes	10	10

Each data set contains samples which belong to one of ten classes: MNIST consists of samples of handwritten digits, ranging from 0 to 9, whilst CIFAR10 consists of samples of

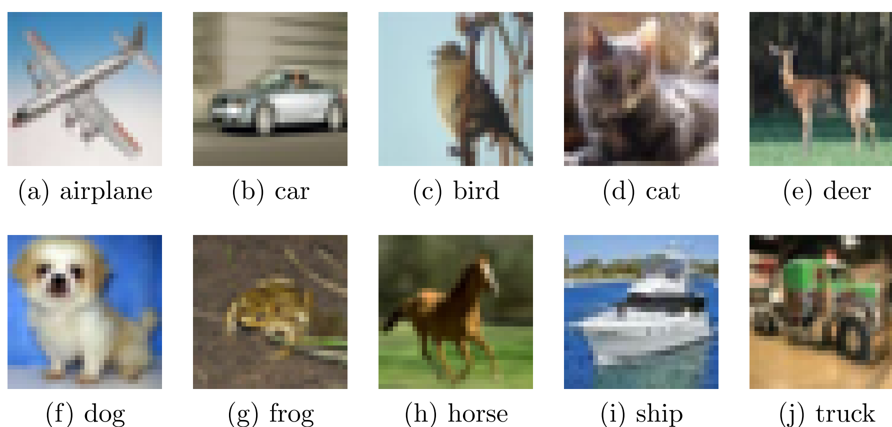
different objects and animals. A typical example of each class for each data set is shown in Figures 3.1 and 3.2.



(a) 0 (b) 1 (c) 2 (d) 3 (e) 4

(f) 5 (g) 6 (h) 7 (i) 8 (j) 9

Figure 3.1: Typical examples of MNIST samples for each class



(a) airplane (b) car (c) bird (d) cat (e) deer

(f) dog (g) frog (h) horse (i) ship (j) truck

Figure 3.2: Typical examples of CIFAR10 samples for each class

Observing the illustrated examples, it is clear that CIFAR10 consists of samples that are much more detailed and complex, and it is generally regarded as a more difficult problem than MNIST. This is advantageous to our goals, as each data set serves a unique purpose. MNIST provides us with a relatively simple problem on which deep learning architectures reach very high performance, and therefore serves as a clean test bed for comparison of different architectures in terms of transformation invariance. Conversely, the greater difficulty of CIFAR10 allows us to compare the generalization ability of different architectures to their transformation invariance, and further verify whether results on MNIST are still applicable. Additionally, by making use of two data sets we are able to compare the general pattern of the results on each, which further allows us to make more informed

conclusions. Finally, as these data sets are widely used, they are well curated, implying that we need not concern ourselves with errors commonly present in real world data.

3.3 Architecture selection and training

In this section we describe the basic network architecture layouts we use for MLPs, CNNs, and STNs. Following this we explain our training protocol and hyperparameter selection.

3.3.1 CNN architecture

We use various CNN architectures throughout this document, however, the general layout of our models remains constant. We explicitly try to keep our CNN architectures simple, as more complex models are more difficult to interpret and also require extensive resources for training. We limit our architectures to three convolution-pool blocks, each of which consists of a convolution layer followed by a max pooling layer, as are used in many state-of-the-art architectures such as AlexNet [39] and ZFNet [48]. We incrementally double the amount of channels for each convolution layer, which is also common. After the convolution stage, final classification is performed by two fully connected layers. Throughout our analysis, we vary the number of channels of each layer, the size of the pooling kernels, as well as their stride. An example of one such an architecture is shown in Table 3.2.

Table 3.2: Example CNN architecture with three convolution (Conv) and pooling (Pool) layers, as well as two fully connected (FC) layers

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**
Conv	1	32	3x3	1	1
Pool	32	32	2x2	2	0
Conv	32	64	3x3	1	1
Pool	64	64	2x2	2	0
Conv	64	128	3x3	1	1
Pool	128	128	2x2	2	0
FC	3 200	400	N/A	N/A	N/A
FC	400	200	N/A	N/A	N/A
FC	200	10	N/A	N/A	N/A

We elaborate on the hyperparameter selection and training protocol for these CNN architectures later in this section (Section 3.3.4). We also report on the test accuracy of each model when performing analysis in later chapters.

3.3.2 Spatial transformer architecture

Given that we insert spatial transformer modules into existing CNN architectures, we try to keep these architectures simple to prevent excessive increases in complexity. After experimenting with different setups, we identify an architecture for each data set which is able to adequately apply corrective transformations when combined with a CNN, we elaborate on this in Chapter 6. The localization network architecture for MNIST is shown in Table 3.3 and in Table 3.4 for CIFAR10. As we focus on affine transformations, each localisation network outputs six theta values which are used by an affine grid sampler to perform a corrective transformation. The sampler module of the spatial transformer makes use of bilinear interpolation [38]. The test accuracy of each model is reported and explained after analysis in Chapter 6.

Table 3.3: Spatial transformer localisation network for MNIST with convolution (Conv), pooling (Pool), and fully connected (FC) layers

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**
Conv	1	8	7x7	1	3
Pool	8	8	2x2	2	0
Conv	8	16	5x5	1	2
Pool	16	16	2x2	2	0
FC	1 600	1 024	N/A	N/A	N/A
FC	1 024	6	N/A	N/A	N/A

Table 3.4: Spatial transformer localisation network for CIFAR10 with convolution (Conv), pooling (Pool), and fully connected (FC) layers

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**
Conv	3	32	3x3	1	1
Conv	32	32	3x3	1	1
Pool	32	32	2x2	2	0
Conv	32	32	3x3	1	1
Pool	32	32	2x2	2	0
Conv	32	32	3x3	1	1
Pool	16	32	2x2	1	0
FC	4 608	1 024	N/A	N/A	N/A
FC	1 024	6	N/A	N/A	N/A

3.3.3 MLP architecture

Standard MLPs serve as simpler models with which to compare other, more advanced architectures, but we must still ensure that we select architectures that perform well. We identify an architecture for both MNIST and CIFAR respectively which performs exceptionally well for a standard feedforward neural network, and is the result of an extensive search across architectures, batch sizes, and learning rates, as conducted by fellow group member Heymans [49]:

- For MNIST, we use 4x400 hidden layers, a batch size of 128, and learning rate of 0.0005. We find that this network achieves a test accuracy of 98.44%, averaged over three initialization seeds.
- In the case of CIFAR, our architecture consists of 2x4000 hidden layers, batch size of 128, and learning rate of 0.0001. After training, we achieve an average test accuracy

of 57.66%, which is remarkably good for an MLP architecture.

For both architectures, all other hyperparameters and protocols used are as explained in the following section.

3.3.4 Training protocol and hyperparameter selection

In this section we provide an overview on our design choices and protocols for training architectures for both our selected data sets. When comparing different models of the same type, i.e. the same sort of architecture (e.g. CNN models), we keep all hyperparameters apart from batch size and learning rate constant; this ensures that our analysis are valid and not subject to differences in training. The following describes our default setup and hyperparameters, and it should be assumed that they are used unless specified otherwise.

Firstly, we pre-process the data by zero padding. MNIST networks are padded by 6 (meaning 6 rows of zeros are added to every edge of the image) and 10 for CIFAR10, resulting in 40x40 and 52x52 sized images respectively. This allows space on the image canvas for transformations such as scaling or translation. For data set partitioning, we make use of three sets: train, validation, and test. For both CIFAR and MNIST we make use of the standard test set, which is 10 000 samples in size for each. Furthermore we split the standard train set into a train and validation set, where 5 000 samples are reserved for the validation set. This implies 55 000 train samples for MNIST and 45 000 train samples for CIFAR10.

Considering other hyperparameters, batch size is kept constant at 128 for MNIST networks, and CIFAR10 networks make use of either 64 or 128 sized batches. We sometimes employ the smaller batch size of 64 due to a lack of video memory to train exceptionally large networks. Step-wise learning rate decay is also used - every 10 epochs the learning rate is multiplied by a scalar, in the range of 0.95 to 0.99, depending on the specific model. The starting learning rate for a group of networks is chosen empirically based on validation set performance, within the range of 0.001 to 0.0001 (with the exception of the

two MLP architectures specified earlier).

In terms of optimization, all models use cross-entropy loss and ReLU activation functions [18], along with the Adam optimizer [50]. This optimizer is especially useful as it generally provides more rapid convergence for MNIST and CIFAR10 than other popular optimizers [50], and is less sensitive towards initial learning rate selection than the standard stochastic gradient descent (SGD). All MNIST networks are trained for a minimum of 100 epochs and 200 epochs for CIFAR10; furthermore, if a network has shown improvement in validation or training accuracy within the last 10 epochs an additional 15 epochs are added to training, this allows both validation and train accuracy to suitably converge. It is further confirmed that all networks are trained to 100% train accuracy, with the exception of models trained on transformed data, which generally converge very near to 100% train accuracy (we report on the accuracy of the models used for analysis throughout the document).

In terms of regularization, early stopping is used: the model at the epoch exhibiting the highest validation accuracy is selected. Apart from this, we do not make use of any explicit regularization methods such as dropout or batch-norm, as these methods could potentially affect the transformation invariance of a given architecture. Finally, we train each network with three different initialization seeds, and report on the average test accuracy, as well as standard error across seeds.

3.4 Measuring invariance

The term *transformation invariance* implies that a system in completely unaffected by transformations of the input, however, as previously discussed this is not the case for deep learning architectures. Therefore, we require a method to quantify transformation invariance.

As we focus on image classification problems, a simple solution would be to simply measure classification accuracy on transformed inputs. However, this method has two drawbacks:

1. There are differences in generalization ability between models. If one model generalizes better, it is difficult to compare its transformation invariance to other models that do not generalize as well if only classification accuracy is measured. Furthermore, we would also expect that a relation exists between generalization and transformation invariance, meaning a model could exhibit better transformation invariance simply because it generalizes better. If accuracy is the only metric employed, we can not separate the one from the other.
2. Classification accuracy is not a precise measurement. Accuracy does not quantify the extent to which the model output is affected when subjugated to transformed inputs, if the classification result is not changed.

Given these flaws, we opt to use an alternative method, as employed by several other authors [9] [43] [46]. We measure transformation invariance across different architectures by comparing the activation value vectors at the network output layer for an original and transformed sample. By comparing entire output vectors we can more accurately measure invariance. This method requires suitable comparison metrics, as discussed next.

3.4.1 Metrics

We use the following metrics for comparing network output vectors for transformed and untransformed samples:

1. Mean Cosine Similarity (MCS). A commonly used similarity measure which follows directly from the definition of the dot product:

$$a \cdot b = \|a\|\|b\| cos(\theta) \tag{3.1}$$

$$cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|} \tag{3.2}$$

Cosine similarity uses only the angle between any two vectors a and b as a similarity measure, meaning differences in vector magnitudes are ignored. We use this metric

as a sensitive measurement to quantify small changes in invariance, and further allows us to make normalized comparisons. The cosine similarity between two vectors is within the range of +1 (most similar) to -1 (most dissimilar).

2. Probability of top 1 change (PTop1): The probability of the top class prediction of a given network changing after an image is transformed, as originally proposed by Azulay and Weiss [9]. While Azulay and Weiss only employ this metric for translations of one pixel, we use this metric to measure invariance for other, and larger, transformations as well. This allows us to determine an exact probability of a sample being incorrectly classified given a specific range of transformation. This is useful as it purely measures a *change* in prediction accuracy, and is therefore not as sensitive a measure as cosine similarity.

When comparing models that are very similar in architecture, for example CNN architectures with different kernel sizes, we employ the mean cosine similarity metric to measure the smaller differences in invariance. When comparing models that are very different, e.g. a CNN to an MLP, we make use of the less sensitive PTop1 metric, as we require an absolute indication of the models' comparative invariance when confronted with transformed inputs.

3.4.2 Method of comparison

For each measurement, we first find the test set samples that are correctly classified by each model in the comparison prior to transformation. The correctly classified samples are then transformed by randomly sampling from a specified range of transformation. We then do two sets of forward passes, one for the original test set and one for the transformed test set. This provides us with two sets of output vectors for each sample, we then compare each transformed output vector to its unaltered counterpart using one of the metrics specified earlier (MCS or PTop1). Finally, we take the average of these values which provides us with a single scalar value per metric, specifying the network's invariance for a specified maximum range of transformation. Stated otherwise, we report

on the average value of the metric, across test samples that are transformed according to randomly sampling from a specified range. This is our standard protocol and we do not deviate from this, it should thus be assumed (unless explicitly stated otherwise) that MCS and PTop1 are always evaluated on the *test set* of each data set, and only the samples that are correctly classified prior to transformation are used.

We explain the specific transformation process for each transformation below.

Translation

In the case of translation, there are four directions of movement that must be taken into account to accurately measure a model's invariance to shifts of the input. We translate each sample by randomly sampling from a given maximum range for both horizontal and vertical shift separately, as shown in Equations 3.3 and 3.4 where t_x and t_y is the change in horizontal and vertical position in pixels, respectively. This allows us to expose a model to a wide variety of translations.

$$t_x \in [-max, +max] \tag{3.3}$$

$$t_y \in [-max, +max] \tag{3.4}$$

After translation, the resulting "empty" areas on the image canvas are filled with zero values.

Rotation

When measuring rotation invariance, there is only one axis of movement that needs to be considered. We rotate each sample by randomly sampling from a maximum range specified in degrees. This is shown in Equation 3.5, where Θ is the change in angle.

$$\Theta \in [-max, +max] \tag{3.5}$$

A negative angle indicates a counter-clockwise rotation, while a positive angle indicates a clockwise rotation. After rotation, should the resulting transformed pixel locations not be integers, we make use of nearest integer interpolation. Furthermore, as with translation, the resulting "empty" areas on the image canvas are filled with zero values.

Scale

For scale transformations, we handle down-scaling and up-scaling separately. We measure the change in scale as a fraction of the size of the original. We randomly scale each sample by sampling from a range of size increase or decrease factors, as shown in Equation 3.6 for down-scaling, and in Equation 3.7 for up-scaling, where s is the scale factor.

$$s \in [min, 1] \tag{3.6}$$

$$s \in [1, max] \tag{3.7}$$

As with rotation, we use nearest integer interpolation. With down-scaling, the "empty" areas on the image canvas are filled with zeros. It is also important to note that we do not change the physical size of the image canvas, and merely scale the contents, our images retain their original size in pixels.

3.5 Verification

It is important that our results are repeatable and verifiable. We therefore take the following measures to verify our own results: Firstly, we take care during the training process to ensure that any group of networks of the same architecture type that are compared with one another are subject to the same hyperparameters (besides learning rate and batch size), and the same training protocol applies to all models. Additionally, we train each network with three different initialization seeds, to ensure that a model's performance is not merely a product of a particularly good or bad initialization. We report

on the average result of all three measurements, for both invariance as well as accuracy measurements. For results displayed with graphs we include error bars which indicate the standard error (SE), calculated using Equation 3.8, where n is usually 3. We utilise standard error as a confidence interval, which allows us to determine the significance of comparative results. Finally, for notable results we also do our experiments on both data sets; this ensures that our findings are not specific to that single task. Furthermore, by comparing the general pattern of results on each data set, we are able to make more informed conclusions.

$$SE = \frac{\sigma}{\sqrt{n}} \tag{3.8}$$

3.6 Conclusion

In this chapter we have developed a testbed for measuring and comparing transformation invariance of different architectures. We identify MNIST and CIFAR10 as our main data sets for analysis, and variations of MLP, CNN, and STN architectures for each task. Furthermore we explain our hyperparameter selection and training protocol, and confirm in Chapters 5 and 6 that this protocol allows our networks converge to 100% train accuracy.

In addition to this, we have identified a method for measuring invariance for each of our selected affine transformations, by employing the cosine similarity and probability of top one change metrics. Finally, we have reviewed the steps taken to ensure that our results are accurate, repeatable, and verifiable.

Chapter 4

Translation Invariance - A theoretical perspective

"All is flux, nothing stays still." - Heraclitus, as recounted by Plato [51]

4.1 Introduction

In this chapter we provide a theoretical overview of translation invariance and identify which parameters play a role. In the case of CNNs, translation is distinct from other affine transformations as CNNs make use of convolution and pooling operations which are inherently translational, as filter kernels are shifted over an image to provide an output. Due to this, the translation transformation requires special consideration.

In Section 4.2 we first define the mathematical properties that pertain to translation, namely invariance and equivariance, and show that they generally do *not* hold in the case of CNNs. We show how the use of subsampling, commonly referred to as "stride" [52], leads to a loss of information and breaks equivariance. Following this, in Section 4.3, we

analyse the effects of stride further and find that it can be greatly beneficial to achieve translation invariance, due to a property we call *shiftability*. Given this information we then show what is required for shiftability to hold when subsampling, and analyse its interplay with filtering (Section 4.3.2). Finally, in Section 4.3.3, we show that an inherent property of a data set, which we call *local homogeneity*, is the deciding factor for achieving translation invariance when combined with subsampling in CNNs.

4.2 Invariance and equivariance

The inherent translation of CNN filters leads to the erroneous assumption that these systems are invariant to translations of the input image, or that the spatial location of internal features are irrelevant for classification. However, Azulay and Weiss [9] show that even for state of the art CNN architectures (VGG16, ResNet50, InceptionResNetV2), a single pixel shift of the input image can cause a severe change in the prediction confidence of the network. Surprisingly the shift of the input is imperceptible to the human eye, but drastically impedes the network's classification ability. In an attempt to understand why this is the case, we start by defining the relevant terms.

Translation invariance can be described as follows (as adapted from [44]):

Definition 4.2.1. A function f can be said to be invariant to a group of translations G if for any g element of G:

$$f(g(I)) = f(I) \tag{4.1}$$

This implies that g has no effect on the output of function *f*, and the result remains equal whether g is applied or not.

A second misconception is that whilst modern CNNs are not translation *invariant*, they are translation *equivariant*. Translation equivariance (also referred to as covariance by some authors) is the property by which internal feature maps are shifted in a one-to-one ratio along with shifts of the input. We define translation equivariance as follows (as adapted from [44]):

Definition 4.2.2. A function f is equivariant to the translations of group G if for any g element of G:

$$f(g(I)) = g(f(I)) \tag{4.2}$$

This implies that the output is shifted in accordance with the shift of the input, or in other terms that the output of the function f can be translated to produce the same result as translating the input I before f is applied.

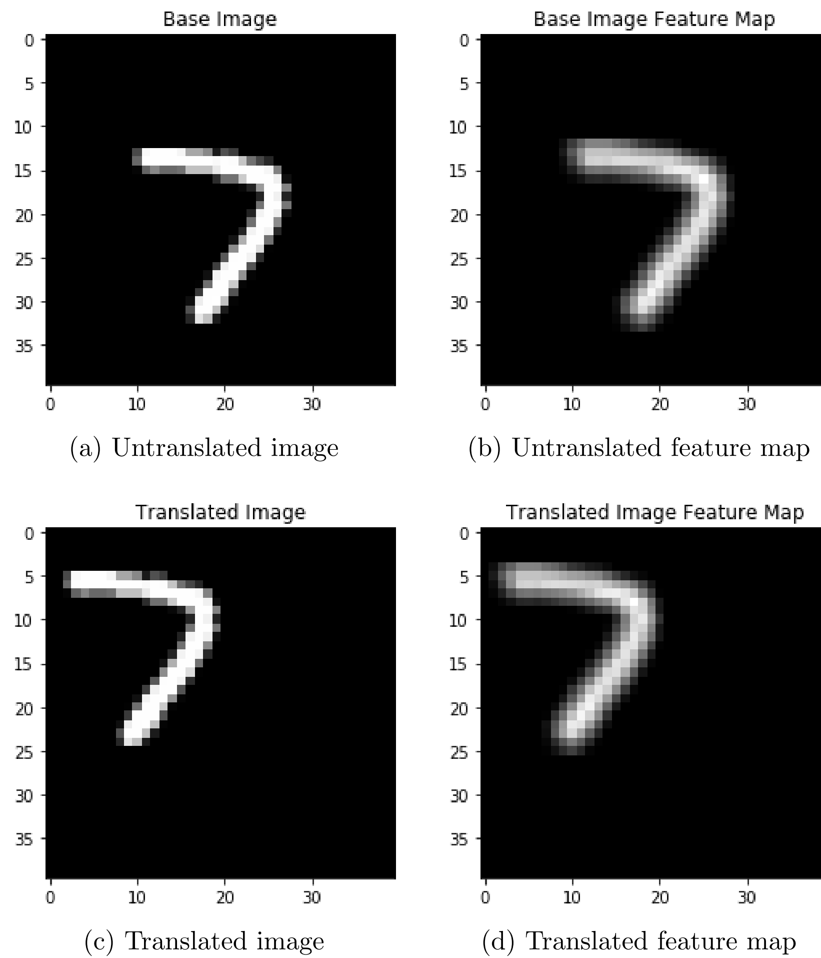
Intuitively it would be expected that translation equivariance holds for both convolution and pooling layers, and this intuition proves correct for dense convolution and pooling, if edge effects are ignored. To illustrate this property, an arbitrary one-dimensional filter is applied to a one-dimensional input, as well as to shifted values of this input. Consider an input signal $I[n] = [0, 0, 0, 0, 1, 2, 0, 0, 0, 0]$ and a kernel $K[n] = [1, 0, 1]$: the result of the convolution $I[n] \circledast K[n]$ is shown in the second column of Table 4.1.

As per this example, the equivariance property holds, as a shift of the input results in an equal shift of the output, meaning $f(g(I)) = g(f(I))$. It is important to note that this property also holds in the case of 2D convolution. We illustrate this by using an arbitrary sample from the MNIST data set and the result of two dense 3x3 convolution layers. An input and convolved output is shown in Figure 4.1, for both an untranslated and translated input.

This example shows that in the two-dimensional case the equivariance property also holds, as both convolved outputs are equivalent. However, the intuition that CNNs are equivariant fails when considering subsampling.

Table 4.1: Dense and strided convolution of a one-dimensional input with a kernel [1,0,1], and shifted variants

Input	**Dense Convolution**	**Strided Convolution**
[0,0,0,0,1,2,0,0,0,0]	[0,0,1,2,1,2,0,0]	[0,1,1,0]
[0,0,0,0,0,1,2,0,0,0]	[0,0,0,1,2,1,2,0]	[0,0,2,2]
[0,0,0,0,0,0,1,2,0,0]	[0,0,0,0,1,2,1,2]	[0,0,1,1]



(a) Untranslated image (b) Untranslated feature map

(c) Translated image (d) Translated feature map

Figure 4.1: 2D Convolution of an untranslated and translated MNIST sample with a 3×3 convolution filter

4.2.1 Subsampling and downsampling

In CNNs, subsampling occurs when a convolution or pooling layer is used with a kernel stride greater than one. Using strided filters results in intermediary samples present in the input being skipped over and disregarded. Strided filters are widely used in state-of-the-art architectures, which has both benefits and drawbacks.

We first consider the disadvantages of subsampling. Azulay and Weiss [9] correctly show that subsampling breaks translation equivariance, as subsampling causes a loss of information. In the case of CNNs, an input is sampled at a rate dictated by the kernel stride, implying that information is disregarded if a stride greater than 1 is used. If information

is lost, shifts of the input are not guaranteed to result in equivalent responses.

This effect can be shown by using the previous example and a stride of 2 when calculating the convolution (thus a subsampling factor of 2) as the third column of Table 4.1 illustrates.

Two important characteristics of stride are illustrated by this example:

1. **Signal information is lost.** Compared to the output of dense convolution, it is clear that subsampling disregards intermediary values. It is logical that this property can have an adverse effect on a CNN's ability to detect features in an image, as translating the input signal can cause features present in the input not to line up with the stride of the kernel.

2. **Translation Equivariance is lost.** The output of the convolution operation is no longer shifted in a one-to-one ratio with the input, meaning $f(g(I)) \neq g(f(I))$, as can be seen by comparing the output of $I[n] \circledast K[n]$ to that of $I[n-1] \circledast K[n]$. This implies that shifts of the input signal result in outputs that are not equivalent. The result of the strided filtering of a translated input is not guaranteed to be sufficiently similar to that of its untranslated counterpart if equivariance does not hold; this partly explains why CNNs are so susceptible to small shifts.

The main benefit of subsampling is that it can **greatly reduce the training time** of CNNs. As He and Sun [53] show, the time complexity of convolution layers in a CNN is given by:

$$O(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2) \tag{4.3}$$

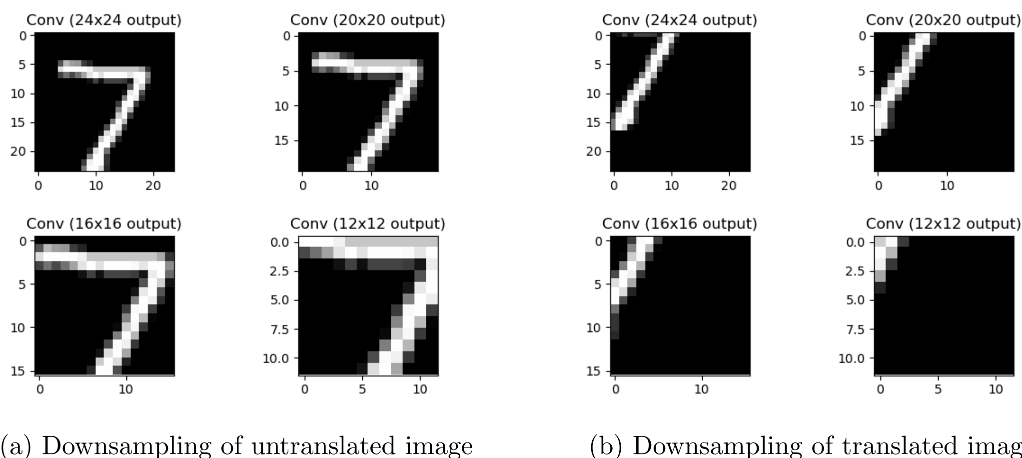
where l is the index of the convolutional layer, d is the number of convolutional layers, and for any layer l, n_l is the number of output channels, s_l the spatial size of the filter, and m_l the size of the output feature map. The subsequent output size of any given layer has a large effect, given that m_l is squared. It can also be shown that the output width of a layer W_l given the previous layer width W_{l-1} is specified by:

$$W_l = \frac{W_{l-1} - k_w + 2p}{s} + 1 \tag{4.4}$$

where k_w is the kernel width, p is padding, and s is stride (the same equation holds for height), as explained earlier in Section 2.3. Given that stride acts as a divisor, we conclude that subsampling drastically reduces the size of the output, and in doing so substantially reduces the time complexity. Furthermore, through spatial reduction, the size of the input layer to the fully connected layers is greatly reduced, implying there are less learnable parameters which further reduces training time and memory consumption [53].

Whilst subsampling reduces spatial dimensions, it is worth noting that this can also be achieved by downsampling in CNNs. We define downsampling as the reduction in spatial size caused by the size of a kernel during a convolution/pooling operation, where information along the edges of an input is disregarded (commonly referred to as "edge effects"). Conversely, subsampling causes spatial reduction by explicitly disregarding intermediary samples.

In terms of translation invariance, downsampling can have a large effect by disregarding parts of an image. We illustrate this phenomenon by using the translated and untranslated sample of Figure 4.1 and passing it through a trained CNN that has four 5×5 convolutional layers without any additional padding, as shown in Figure 4.2.



(a) Downsampling of untranslated image (b) Downsampling of translated image

Figure 4.2: Downsampling with four 5×5 convolution filters of a trained network for a translated and untranslated MNIST sample.

Observing this example it is clear that downsampling causes a large loss in information, which can drastically affect the translation invariance of the network. We do not attempt

to quantify the influence of downsampling on translation invariance, rather, we mitigate this effect by using adequate padding, during both pre-processing and each convolution or pooling operation.

Given subsampling and downsampling's relation to equivariance, a logical conclusion is that a CNN which is fully equivariant (no subsampling is used) and with sufficient padding (to mitigate downsampling) will be translation invariant. This conclusion proves to be false when also considering the fully connected layers. As each resulting feature map of the convolution stage is flattened into a one-dimensional array, spatial information is lost. Fully connected layers possess no spatial reasoning abilities and therefore shifts of the input can drastically alter the resulting classification accuracy of the network. However, subsampling can also be beneficial to this problem, as we explain in the following section.

4.3 Signal movement, signal similarity, and local homogeneity

In this section we show how subsampling plays a role in achieving translation equivariance, and identify the characteristics required for it to be present.

4.3.1 Shiftability

Whilst subsampling breaks the equivariance property, we propose that it can greatly benefit translation invariance under certain circumstances due to a third property we define as shiftability. Shiftability holds for systems that make use of subsampling and is defined as follows (note: this definition of shiftability is distinct from that provided in [9]):

Definition 4.3.1. A function f with subsampling factor s is shiftable for a given translation if

$$f(g(I)) = g'(f(I)) \tag{4.5}$$

where g is a translation function with translation vector $\vec{u}$ and input X

$$g(X) = t(\vec{u}, X) \tag{4.6}$$

and

$$g'(X) = t(\frac{\vec{u}}{s}, X) \tag{4.7}$$

Put otherwise, shiftability holds for translations that are factors of the subsampling factor s of f. When subsampling shifted inputs, equivalence will hold if a given translation is in accordance with the stride. Where equivalence implies that the two resulting feature maps are equal (the global average is the same), except for the difference in position. To illustrate this property, consider an arbitrary input signal that is subsampled by a factor of four and various shifts of the signal, as shown in Table 4.2.

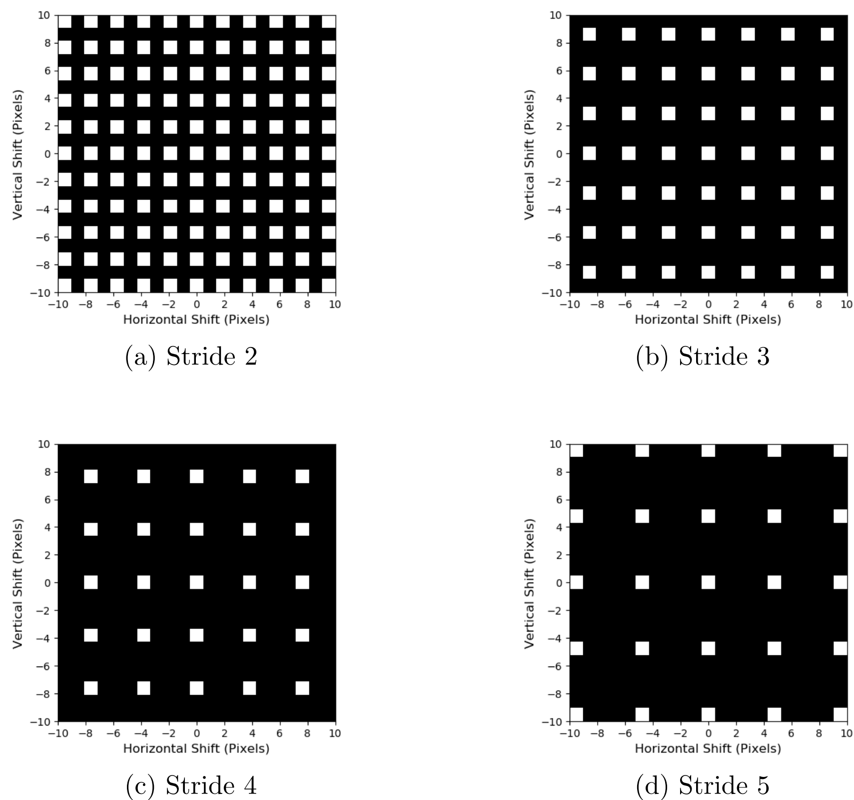
In this example, shiftability holds for translations that are factors of the subsampling factor (shifts of 4 and 8), and so a scaled form of equivariance is kept. It is further evident that subsampling scales shifts of the input signal: in this example, a shift of four in the input results in only a shift of one in the output.

The subsampling factor dictates how many versions of the output signal can potentially exist after translation (again ignoring edge effects): In this example four discrete outputs are present, where all other outputs are merely shifted variants. In the case of two-dimensional filtering, inputs are subsampled both vertically and horizontally, meaning s^2 output signals can exist given a single input and a bounded translation. This further

Table 4.2: Subsampling factor of four for an arbitrary input

Input	0 0 0 0 0 0 0 0 0 0 3 2 5 2 4 1 6 3 4 6 5 5 0 0 0 0 0 0 0 0 0
Shift	**Subsampled Output**
0	**0 0 0 2 3 5 0 0**
1	0 0 0 5 6 5 0 0
2	0 0 0 2 1 6 0 0
3	0 0 0 3 4 4 0 0
4	**0 0 0 0 2 3 5 0**
5	0 0 0 0 5 6 5 0
6	0 0 0 0 2 1 6 0
7	0 0 0 0 3 4 4 0
8	**0 0 0 0 0 2 3 5**

implies that a given input will only be shiftable for $\frac{1}{s^2}$ of possible translations [9]. We demonstrate this by using the sample from Figure 4.1 and passing it through a subsampling filter with varying stride, we then measure for which translations the global average is equal to that of the untranslated response, over a 10 pixel radius. This is shown in Figure 4.3. (Note: we pad the original sample to allow space on the canvas area for translation.)



(a) Stride 2 (b) Stride 3

(c) Stride 4 (d) Stride 5

Figure 4.3: Shiftability maps for arbitrary MNIST sample: white indicates pixel positions that are shiftable, black positions are not.

This example illustrates that shiftability only holds for translations where both the vertical and horizontal shifts are factors of the subsampling factor.

To explain how shiftability benefits translation invariance, we must first define two distinct characteristics that must be accounted for when comparing outputs of translated inputs to that of untranslated inputs.

4.3.2 Signal similarity and signal movement

We propose two characteristics that influence translation invariance when subsampling is present:

- **Signal Similarity**: How much of the untranslated signal's output information is preserved after translation.
- **Signal Movement**: How far the translated output has been moved from the original position of the untranslated output.

As an example, consider an output signal of [0,0,1,2,0,0] and another of [0,0,0,1,2,0]. These signals are exactly similar except that the second is shifted by one. Conversely, an input of [0,0,1,2,0,0] and [0,0,2,3,0,0] are not exactly similar, but do not exhibit any shift (signal movement).

It is evident that subsampling reduces signal movement, but leads to the loss of signal similarity (for shifts that are not factors of the subsampling factor), conversely using no subsampling leads to perfect signal similarity but allows the output to shift in a one-to-one ratio with the input. For translation invariance in CNNs signal similarity must be kept but signal movement must also be reduced to the extent possible. Whilst the reduction of signal movement during translation is fully dependent upon the subsampling factor, we propose that the degree of signal similarity that is preserved during subsampling is dependent upon *local homogeneity*.

4.3.3 Local homogeneity

As subsampling disregards intermediary pixels in a given feature map, translated versions will result in more equivalent outputs if neighbouring pixels are more similar to each other in a given region. Put otherwise, preserving signal similarity requires that the variance between intermediary elements in a given window are sufficiently low, the size of which is dictated by the subsampling factor. We refer to this property as “local homogeneity”.

To illustrate the effect of local homogeneity, consider an input that is fully locally homogeneous in accordance with the subsampling factor and the resulting output when shifting the signal. This is shown in Table 4.3 where a subsampling factor of 2 is used.

Table 4.3: Subsampling of a locally homogeneous signal

Input	0 0 0 0 0 0 2 2 3 3 1 1 2 2 0 0 0 0 0 0
Shift	**Subsampled Output**
0	0 0 0 2 3 1 2 0 0
1	0 0 0 2 3 1 2 0 0
2	0 0 0 0 2 3 1 2 0
3	0 0 0 0 2 3 1 2 0
4	0 0 0 0 0 2 3 1 2

Given that successive elements are similar in accordance with the subsampling factor, subsampling results in equivalent responses, yet the initial shift of the input is also scaled; implying that signal similarity is preserved and signal movement is negated. Our hypothesis surrounding local homogeneity brings about two logical conclusions:

- **Pooling improves local homogeneity**: Dense pooling (max and average pooling alike) reduces the variance of any given input, therefore providing more similarity between neighbouring pixels.
- **Strided pooling can benefit translation invariance**: Combining a pooling window of sufficient size (and therefore sufficient local homogeneity) with subsampling leads to the reduction of signal movement. This implies that translated samples will result in outputs that are more equivalent to untranslated samples.

4.4 Conclusion

In this chapter we studied how the different parameters of convolutional networks relate to translation invariance. We defined invariance and equivariance, and found that subsampling breaks equivariance by leading to a loss of information. We then analysed the benefits and drawbacks of subsampling, and also how downsampling plays a distinctly

different role from subsampling. Given this information we introduced shiftability, and explained how it relates to signal movement and signal similarity. Finally we stated that given sufficient local homogeneity, subsampling can be very beneficial to translation invariance, and motivated why we believe this to be the case.

In Chapter 5 we empirically explore our hypothesis surrounding local homogeneity by observing the effects of stride and filtering on translation invariance, as well as the role of local homogeneity in solutions to translation invariance proposed by others.

Chapter 5

Translation Invariance - An empirical analysis

"What can be asserted without evidence can be dismissed without evidence." - Christopher Hitchens [54]

5.1 Introduction

In this chapter we empirically explore our hypotheses surrounding subsampling and local homogeneity, and how they relate to translation invariance and generalization.

Firstly, we describe the experimental setup and architecture selection in Section 5.2. In Section 5.3, we measure the effects of varying pooling kernel sizes and strides, repeated across several different architectures and datasets. Following this we examine the efficacy of solutions to translation invariance proposed by other authors, namely anti-aliasing (Section 5.4) and global average pooling (Section 5.5). We then measure the effects of learned invariance, by translating the data set prior to training, in Section 5.6. We

then turn our attention towards the generalization of these different methods, and how invariance and generalization is related (Section 5.7). Finally, in Section 5.8, we do a comprehensive comparison between architectures and draw conclusions from these and the preceding results.

5.2 Experimental setup

For experiments surrounding translation invariance, we select a three layer architecture, without subsampling, as baseline, as shown in Table 5.1. This architecture consists of 3x3 convolution kernels, 2x2 max pooling kernels, and three fully connected layers. We use the same architecture for both MNIST and CIFAR10 - generally we would use an architecture with more capacity for CIFAR10 (more filters or larger fully connected layers), but as we do not subsample the computational cost is too great. The training protocol and all other hyperparameters are as explained previously in Section 3.3, except that we use a batch size of 64 and 128, and learning rate of 0.0001 and 0.001, for CIFAR and MNIST respectively. Despite its minimized capacity, this architecture still performs very well on both data sets, and is therefore a suitable choice for a baseline. The MNIST baseline achieves a test accuracy of 99.36%, while CIFAR10 an acceptable 72.33%.

Table 5.1: Baseline CNN architecture w/o subsampling for MNIST and CIFAR with three convolution (Conv) and pooling (Pool) layers, and two fully connected (FC) layers

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**
Conv	1	32	3x3	1	1
Pool	32	32	2x2	1	0
Conv	32	64	3x3	1	1
Pool	64	64	2x2	1	0
Conv	64	128	3x3	1	1
Pool	128	128	2x2	1	0
FC	175 232/307 328	400	N/A	N/A	N/A
FC	400	200	N/A	N/A	N/A
FC	200	10	N/A	N/A	N/A

To measure translation invariance, we translate the test set samples by randomly sampling from a given range, as explain in Section 3.4, and then compare output classification

vectors to that of the untranslated samples. As explained earlier, we only use the samples that are correctly classified (prior to translation) by all the models contained within the experiment. We make use of both the mean cosine similarity (MCS) and probability of top 1 change (PTop1) metrics, and average all results over three initialization seeds.

5.3 Pooling and subsampling

Given the baseline architecture of the preceding section, we explore the effects of subsampling and pooling on translation invariance. Three other networks are trained with subsampling factors of 2, 4, and 8 respectively for the MNIST dataset. The subsampling factor is varied by consecutively setting the stride of the 2x2 max pooling filters in the network to 2, starting at the first layer, as shown in Table 5.2. The comparative MCS for these models is shown in Figure 5.1.

Table 5.2: Stride of max pooling layers for MNIST architectures

Pooling Layer	**Stride**			
	SS Factor 1	SS Factor 2	SS Factor 4	SS Factor 8
1	1	2	2	2
2	1	1	2	2
3	1	1	1	2

This result shows that subsampling improves translation invariance for MNIST, where networks using subsampling show a substantially higher MCS. This further implies that 2x2 max pooling is sufficient to provide local homogeneity given a stride of 2 for each layer, however samples in the MNIST dataset are generally inherently highly homogeneous.

CIFAR10 is a more complex and detailed dataset, and is therefore generally less homogeneous and would require a larger degree of filtering to ensure translation invariance. To ascertain the required max pooling size for this dataset, the previous experiment is repeated for kernel sizes ranging from 2x2 to 5x5 for each layer. This is shown in Table 5.3, which shows the mean cosine similarity for a maximum shift of 10 pixels for each network. We observe that in the case of CIFAR10 2x2 max pooling is not sufficient, and a substantial increase in translation invariance following subsampling is only observed at

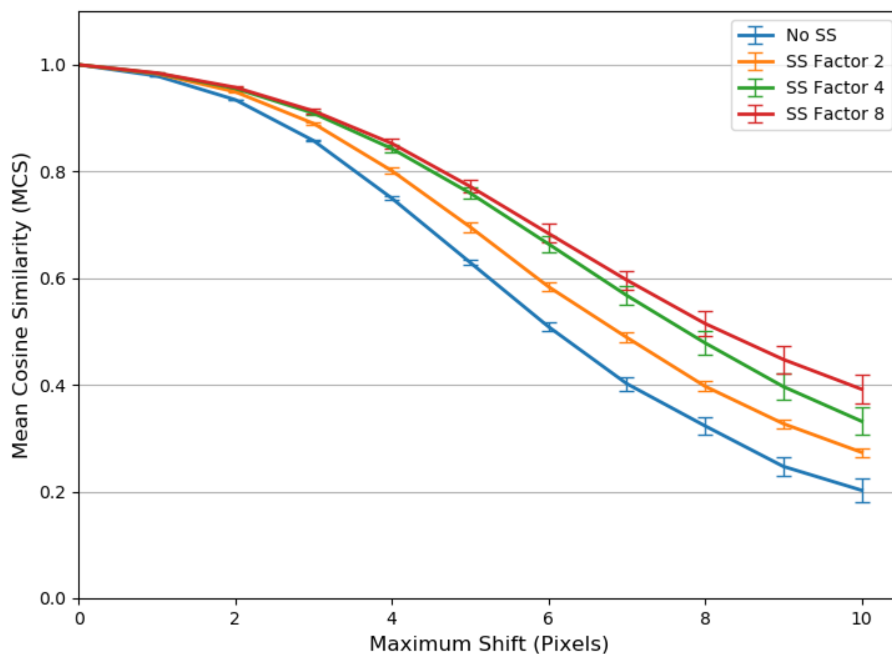


Figure 5.1: MCS comparison for MNIST architectures with varying subsampling

Table 5.3: Mean cosine similarity for CIFAR10 networks with varying subsampling and max pooling kernel sizes - 10 pixel range (SE < 0.04)

Subsampling Factor	**Kernel Size**			
	2x2	3x3	4x4	5x5
1	0.630	0.598	0.595	0.618
2	0.554	0.635	0.683	0.731
4	0.622	0.674	0.759	0.789
8	0.610	0.660	0.762	0.791

3x3 pooling and larger.

For networks that make use of subsampling, we observe that larger kernel sizes always result in greater invariance. Conversely, for networks that do not make use of subsampling (the first row of Table 5.3) we observe decreased translation invariance for larger kernels. Intuitively one would expect larger kernels to always provide greater translation invariance, but this intuition fails since these networks are fully translation equivariant, as no subsampling is applied. Finally, we observe that greater subsampling always results in greater invariance when adequately sized kernels are used (as in the case of 4x4 and

5x5 pooling) which are aligned with our findings on MNIST. The test accuracy of these models (both CIFAR and MNIST) is shown and analysed in Section 5.7 (Tables 5.8 and 5.10).

In summary, these results support our proposal that stride can significantly increase the translation invariance of a network, given that it is combined with sufficient local homogeneity. Furthermore we also find that the inherent homogeneity of a given dataset dictates the required filtering for subsampling to be effective. In Section 5.7 we revisit these architectures, and evaluate how classification performance relates to translation invariance.

Now that we have shown how subsampling can benefit translation invariance when it is combined with sufficient local homogeneity, we can explore how this idea relates to other methods such as anti-aliasing.

5.4 Anti-aliasing

In traditional signal theory, the term *aliasing* refers to the distortion of a signal when it is sampled at a frequency lower than that of the Nyquist frequency [55], meaning that the original signal is no longer reconstructable from its samples. The standard solution when subsampling is to low-pass filter the input prior to sampling, as this removes high frequency components and prevents aliasing when the signal is sampled, this process is known as *anti-aliasing* [56].

This method of low-pass filtering before subsampling is mostly ignored in conventional CNNs, with the exception of average pooling. Averaging filters, also referred to as box filters, serve as low-pass filters by smoothing out the input signal [57] prior to subsampling. However, Scherer et al. [34] show that max pooling generally results in better generalization than that of average pooling when used in CNNs.

Zhang [43] proposes a solution to this problem which allows the generalization benefits of max pooling without compromising translation invariance. The author alters strided

max pooling by separating it into three distinct layers (this is also illustrated in Figure 5.2):

1. Dense Max Pooling (max pooling with a stride of 1)
2. Anti-Aliasing
3. Subsampling

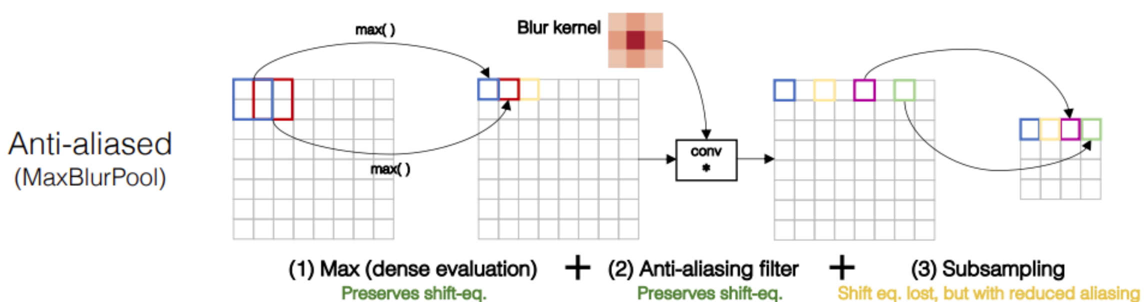


Figure 5.2: Anti-aliasing method preserving max pooling, from [43]

By adding in an intermediary filter, subsampling results in a more equivalent response (with regards to an untranslated sample), but the generalization benefits of max pooling is not compromised. To use our terminology, by applying an anti-aliasing filter local homogeneity is ensured and the subsequent subsampling operation's effect on signal similarity is strongly mitigated, which results in a more translation invariant network.

Whilst Zhang demonstrates the efficacy of several low-pass filters, we opt to use the best performer (shown by Zhang) - the 5x5 binomial filter. Binomial filters provide an approximation of the discretized Gaussian [58], and is the result of convolving a mean filter with itself [43] which serves as a blurring (anti-aliasing) filter. The use of anti-aliasing filters have the added benefit that they do not make use of any learnable parameters, as such the increase in time complexity is negligible.

The efficacy of this method is explored for both the MNIST and CIFAR10 datasets using the three layer 2x2 pooling networks from Section 5.3. Each pooling layer present in the network is replaced with a dense max pooling layer and a 5x5 binomial anti-aliasing filter.

These networks are then compared to their baseline counterparts that do not make use of anti-aliasing. The comparative MCS for a maximum shift of 10 pixels is shown in Table 5.4.

For MNIST, anti-aliasing seems to always provide better translation invariance regardless of whether subsampling is used or not. However, the greatest increase in translation invariance occurs when subsampling is applied, as it is not solely anti-aliasing that provides invariance, but its combination with stride. For CIFAR10 the results are slightly different, anti-aliasing greatly reduces the invariance of the architecture without subsampling. However, as is the case with MNIST, a large increase is evident when it is combined with subsampling.

In conclusion, these results confirm that both signal similarity must be preserved and signal movement must be reduced to increase the network's invariance to translation, and that anti-aliasing is an effective solution for ensuring local homogeneity. However, anti-aliasing is still not a perfect solution, as it does not guarantee complete translation invariance; in the next section we examine a complete solution in the form of global average pooling. We also revisit these anti-aliasing models and how they relate to generalization in Section 5.7.

5.5 Global average pooling

Whilst we have shown that the use of subsampling can greatly benefit translation invariance, Azulay and Weiss [9] propose a different approach that makes use of global

Table 5.4: Mean cosine similarity for MNIST and CIFAR10 networks with and without anti-aliasing (AA) for a maximum shift of 10 pixels

Subsampling Factor	**AA**	**MNIST**	**CIFAR**
1	No	0.248	**0.630**
	Yes	**0.329**	0.518
4	No	0.383	0.620
	Yes	**0.654**	**0.710**
8	No	0.447	0.611
	Yes	**0.638**	**0.690**

average pooling (GAP) and avoids any subsampling/downsampling throughout the network. Global pooling performs an operation over the entire feature map simultaneously, unlike conventional pooling which makes use of discrete windows which are shifted over the image [59]. This implies that the GAP operation takes the average of an entire feature map, and reduces it to a single scalar value. By withholding the use of any subsampling in the network, complete equivariance is kept, and as global pooling is not influenced by signal movement, the result is a perfectly translation invariant system. However, the effects of downsampling must also be accounted for, as any loss of information throughout the network would cause a difference in the global average of any given feature map.

We verify this by constructing a simple two layer model with no subsampling, and add a final global average pooling operation, which is applied independently per channel. Furthermore, we use full padding for each convolution and pooling layer to prevent any spatial reduction through downsampling, and train this architecture on the MNIST dataset. We find that it is completely invariant to translation, as we measure a 0% probability of classification change for shifts up to a maximum of 10 pixels.

Although this might seem to be a complete solution, the combination of GAP and no subsampling is not without its drawbacks. The GAP operation disregards a tremendous amount of information simultaneously, as opposed to subsampling which incrementally disregards samples, and could therefore lower the classification ability of a given architecture. Furthermore, as discussed earlier, the main benefit of subsampling is the reduction in training time. When training large networks on large images, withholding subsampling drastically increases the computational complexity. Due to these reasons we do not believe this method is necessarily a suitable solution for any given dataset or architecture, as it could potentially hamper generalization and/or training efficiency.

In summary, global average pooling and no subsampling (with adequate padding) is an acceptable method to provide complete translation invariance, but has the drawback of potentially hampering generalization ability, and increasing complexity. In the following section we experiment with learned invariance by translating the data set prior to training.

5.6 Learned invariance

We have thus far focused on *inherent invariance*, meaning invariance that is obtained from specific architectural elements within a model and not wholly dependent on data. We now turn our attention towards *learned invariance*, which is invariance that is obtained by virtue of the data set.

In the case of translation invariance, it would be expected that a data set which contains diverse samples of different spatial locations would force a model to learn translation invariant representations. However, tiny image datasets such as MNIST and CIFAR are biased in the sense that features of interest are centered within the canvas area, exhibiting only slight deviations from this position. A common solution to this is transforming the data set prior to training (or augmenting the standard train set with transformed data), meaning (in this case) translating each sample to diversify the examples presented to the network during training.

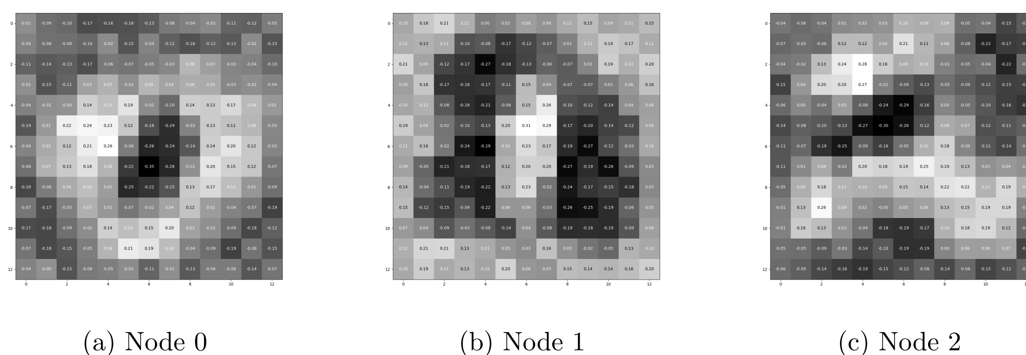
This pre-processing method then raises the question of which learnable parameters are responsible for the resulting invariance of the model. As already established, CNNs make use of convolutional kernels whose parameters are learned from data, but also make use of fully connected layers. Given that the convolutional kernels are inherently translational, we would expect that pre-translating data would have the largest effect on the parameters of the fully connected layers.

We devise an experiment to visualize how weights in fully connected layers are affected by data translation: A simple CNN is trained on a subset of the MNIST dataset, consisting of a one channel convolution and pooling filter, whilst the fully connected stage has no hidden layer - only an input and output layer. This architecture is illustrated in Table 5.5. As capacity is severely limited, we restrict the network to only three classes, namely: class 0, 1, and 2. The convolution stage outputs a 13x13 feature map, or 169 features, and as no hidden layer is present each pixel has only three weights associated with it for classification.

Table 5.5: Simple MNIST CNN architecture for weight distribution visualization

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**	**Activation**
Conv	1	1	3x3	1	0	ReLu
Pool	1	1	2x2	2	0	None
FC	169	3	N/A	N/A	N/A	Softmax

By restricting the dimensionality of the network, we can then visualize the weights associated with each output node, or put otherwise, the weights associated with each class. We reshape these classification parameters into 13x13 images (as each node has 169 inputs, and therefore 169 weights). Finally, we apply a color gradient to the resulting image where higher values are brighter and lower values are darker. The weight visualisation for all three nodes after training on untranslated data is shown in Figure 5.3.

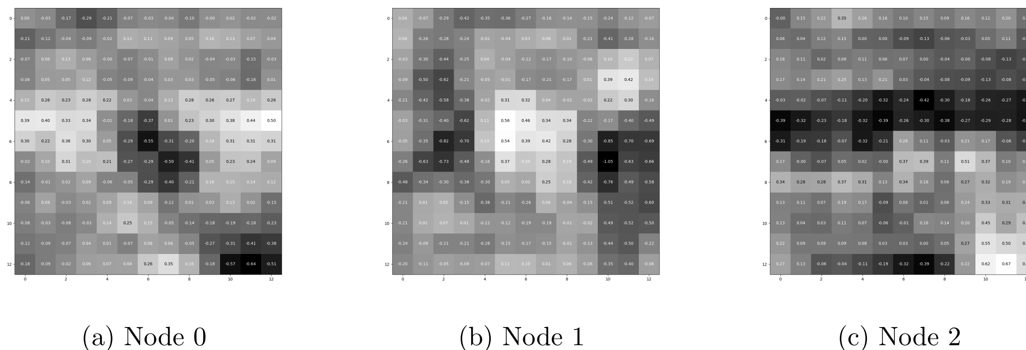


(a) Node 0 (b) Node 1 (c) Node 2

Figure 5.3: Weight visualization for three MNIST output nodes

Observing these visualizations, a clear pattern emerges where higher value weights show a stereotypical example of its corresponding class, and what is more we see that these visualization are very location specific and centered within the image. Put otherwise, certain areas have clearly been associated with a specific class. We now repeat this experiment by training the architecture on translated data, where each sample is randomly shifted up to a maximum of 4 pixels. The resulting weight visualizations are shown in Figure 5.4.

Comparing the translated visualizations to their untranslated counterparts, we observe the weight distributions are more scattered, and the previously observed patterns are less



(a) Node 0 (b) Node 1 (c) Node 2

Figure 5.4: Translated weight visualization for three MNIST output nodes

clear. This indicates that the network has learned to be less location specific with regards to each class, and should therefore be more invariant to translations of the input.

Given these intuitions, two natural questions arise:

1. How beneficial is data translation to translation invariance?
2. Do our previous findings surrounding subsampling and local homogeneity still hold?

5.6.1 Data translation with MNIST

In an attempt to address the previously posed questions, we explore the effects of learned invariance by training our previous MNIST architectures as in Figure 5.1 on a translated data set. We keep the size of the train set constant and randomly translate each sample up to a maximum of 8 pixels, furthermore we apply the same translation to the validation set (the test set remains untranslated). In this way we explicitly optimize our models for translation invariance, and then examine whether our previous findings surrounding subsampling still hold true. The comparative MCS of these networks is shown in Figure 5.5. For the sake of perspective, we also show the test accuracy of each model in Table 5.6.

Observing this result, we find that these networks are much more translation invariant

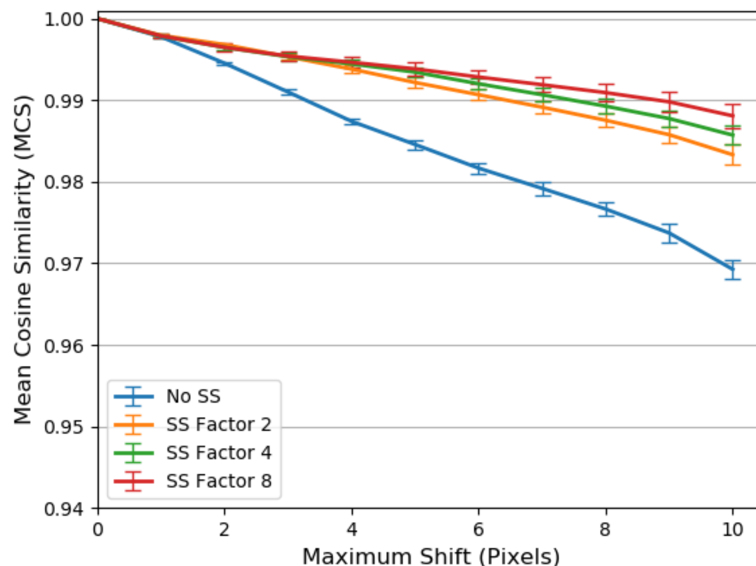


Figure 5.5: MCS comparison for MNIST architectures with data translation

than those not trained on translated data, with the lowest MCS at a staggering 0.97. We attribute this to the fully connected layers learning location invariant representations, allowing the network to be almost completely translation invariant. While it could also be possible that the convolutional filters have learned more invariant representations, due to their equivariant nature we would expect that the fully connected stage contributes the most to invariance (as shown in our earlier experiment of Figures 5.3 and 5.4, the spatial distribution of the fully connected layers are crucial). We also find that the same pattern emerges as that of Figure 5.1, where greater subsampling leads to greater translation invariance, but this effect is greatly minimized (note the scale of the y-axis). While

Table 5.6: Test accuracy for MNIST networks with varying subsampling factors trained on translated data, averaged over three initialization seeds

Subsampling Factor	Test set accuracy
1	99.27
2	99.41
4	99.46
8	99.50

this result is very promising, we must also keep in mind that MNIST classification is a relatively easy problem for neural networks. In the following section we explore the effects

of learned invariance on the more complex CIFAR10 data set.

5.6.2 Data translation with CIFAR

Following the spectacular result of data translation on MNIST, we wish to measure the effects on CIFAR10 architectures, and further verify whether our results regarding pooling size (from Section 5.3) are still valid. We re-purpose three networks with a subsampling factor of 8, and varying pooling sizes, from the last row of Table 5.3. As with MNIST, we randomly translate each sample in the training set up to a maximum shift of 8 pixels before training. The comparative MCS of these models is shown in Figure 5.6. Given that these models do not converge to 100% train accuracy, we also show the test accuracy for each model in Table 5.7.

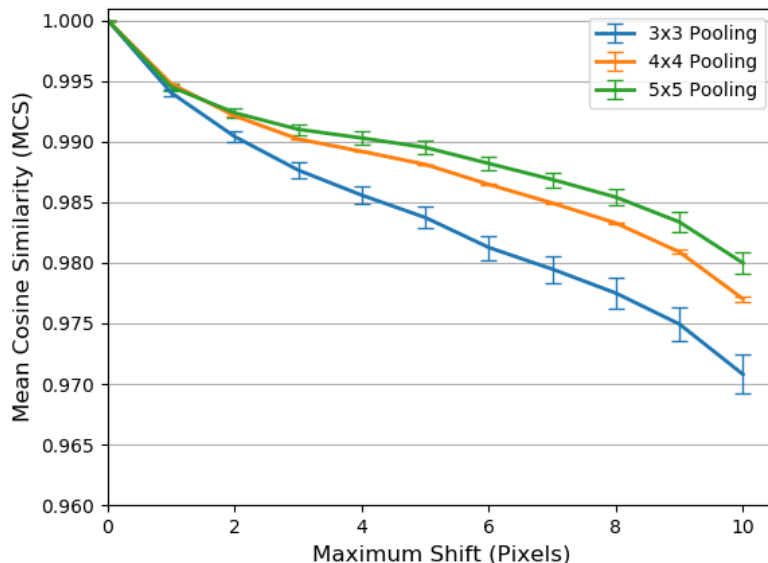


Figure 5.6: MCS comparison for CIFAR architectures with data translation

Once again, this result shows that these architectures are highly invariant to translation, and greatly outperform their counterparts not trained on translated data. We also observe that larger kernel sizes result in better invariance, which is aligned with our previous results, but in this case the effect is nearly negligible (note the scale of the y-axis).

Table 5.7: Test accuracy for CIFAR10 networks with varying kernel sizes trained on translated data, averaged over three initialization seeds

Kernel Size	**Test set accuracy**
3x3	80.28
4x4	79.14
5x5	78.16

While learned invariance is certainly a powerful tool, Azulay and Weiss [9] point out that this method can potentially result in models that are overly biased to translations of the train set and it can not be expected to generalize well to translations of unseen data in all cases. We also point out that MNIST and CIFAR10 are relatively simple data sets, compared to more complex problems such as CIFAR100 or ImageNet [39], and usually data augmentation would be required for these networks to achieve good performance. Data augmentation is the act of explicitly adding transformed training examples to the existing train set, which result in a much larger data set, and has been shown to be very beneficial for generalization [60] [61]. However, this implies that the required training time for a model is also substantially increased.

We further explore learned invariance, along with other transformations, in Chapter 6.

5.7 Translation invariance and generalization

In this section we explore how translation invariance relates to generalization ability. A natural assumption to make is that models that generalize better would exhibit better invariance to transformation. To ascertain whether our results are merely the consequence of differences in generalization, and how invariance relates to generalization, we compare the test accuracy of our models to their MCS.

The test accuracy of our CIFAR10 networks of varying kernel size and subsampling from Table 5.3 is shown in Table 5.8. We also repeat Table 5.3 here (shown in Table 5.9) for easier comparison of the accuracy of the models with their mean cosine similarity.

Table 5.8: Test accuracy for CIFAR10 networks with varying subsampling and pooling kernel size (SE < 0.22)

Subsampling Factor	**Kernel Size**			
	2x2	3x3	4x4	5x5
1	72.33	75.00	76.10	76.00
2	74.43	77.00	77.57	76.69
4	73.94	76.72	77.25	76.76
8	72.53	75.31	76.69	75.95

Table 5.9: Mean cosine similarity for CIFAR10 networks with varying subsampling and max pooling kernel sizes - 10 pixel range (Repeat of Table 5.3)

Subsampling Factor	**Kernel Size**			
	2x2	3x3	4x4	5x5
1	0.630	0.598	0.595	0.618
2	0.554	0.635	0.683	0.731
4	0.622	0.674	0.759	0.789
8	0.610	0.660	0.762	0.791

We observe that larger kernel sizes generally generalize better, but also that kernels that are too large (such as 5x5 in this case) lead to a reduction in test set accuracy. Conversely, 5x5 kernels provide the best translation invariance (highest mean cosine similarity). Furthermore, some subsampling seems to always provide better generalization regardless of kernel size, but too much subsampling leads to a reduction in model performance. This is contrasted with the cosine similarity results, where we see the greatest translation invariance for networks with a subsampling factor of 8. The results of Table 5.8 suggest the following:

1. Over/under filtering hampers generalization ability. We observe that 5x5 pooling kernels perform significantly worse than 4x4 kernels, and similarly kernels that are too small (2x2) also perform worse than larger kernels.

2. Over/under subsampling hampers generalization ability. Subsampling factors of 1 (no subsampling), and similarly subsampling factors of 8, always perform worse than milder subsampling (subsampling factors of 2 and 4).

3. There is a slight trade-off between translation invariance and generalization. While over subsampling and filtering reduces generalization ability, the previous translation

invariance MCS results in Table 5.3 show that the models with greater subsampling and filtering perform better in terms of translation invariance.

Logically this is an expected result: larger kernels lower the variance of a given sample and result in more locally homogeneous regions, but also implies that more information is disregarded which negatively impacts the model's ability to generalize to samples not seen during training, in the same sense large subsampling reduces signal movement but also leads to a loss of information that could be necessary for accurate classification.

These conclusions then raise the following question: Why do *some* subsampling and filtering **improve** generalization? Whilst exploring the reasons for the generalization ability of CNN architectures is somewhat outside our scope, we do offer a possible explanation for why this appears to be the case. When considering subsampling, we hypothesise that stride acts as an explicit regularizer by disregarding unnecessary information contained within the train set. Put otherwise, by incrementally reducing spatial size, only that which is paramount to classification is kept. Similarly, by applying a max pooling operation after convolution, only features of significance remain and propagate further through the network. In this sense, the combination of filtering and subsampling forces greater sparsity upon the model - as much of the irrelevant, train-set-specific data is disregarded, and so the bias-variance trade-off is reduced. This implies that the model does not overfit its function approximation to the small details of the train set. This then leads to better generalization on unseen data. Whilst our provided explanation is certainly logical, there can potentially be other reasons (that are not immediately clear to us) for why some architectures generalize better than others; as such further study would be required to confirm that this hypothesis holds true.

We now turn our attention towards the anti-aliasing models of Section 5.4, whose mean cosine similarity is shown earlier in Table 5.4. We observe a very small overall effect on generalization: Table 5.10 shows the test accuracy of these models with and without the use of anti-aliasing filters.

The combination of subsampling with anti-aliasing actually improves generalization for

Table 5.10: MNIST and CIFAR10 test accuracy with and without anti-aliasing (AA)

Subsampling Factor	**AA**	**MNIST**	**CIFAR10**
1	No	**99.36**	72.33
	Yes	99.18	**73.62**
4	No	**99.38**	73.94
	Yes	99.35	**74.71**
8	No	**99.30**	72.53
	Yes	99.21	**73.40**

the CIFAR10 dataset, and only slightly hampers accuracy for that of MNIST. As we had shown earlier in Table 5.4, the combination of anti-aliasing and subsampling also improves translation invariance. These results are aligned with that of Zhang [43] which show a slight improvement in generalization for state-of-the-art ImageNet networks using anti-aliasing.

Given the results of this section, we conclude the following:

1. Greater subsampling and filtering lead to better translation invariance, but too aggressive filtering or subsampling leads to a reduction in generalization ability. This implies that there exists a trade-off between translation invariance and generalization. (Compare Tables 5.3 and 5.8).

2. No subsampling or filters that are too small negatively affect generalization (Table 5.8).

3. Anti-aliasing improves translation invariance, and does not lead to a reduction in generalization (Tables 5.4 and 5.10).

4. Our previous results were not simply due to the differences in generalization ability among the networks compared.

In the following section we do a comprehensive translation invariance comparison between different deep learning architectures, and also review their generalization abilities.

5.8 Architecture comparison

In the preceding sections we have identified which parameters play a role in translation invariance for the case of CNNs, which other techniques and methods are available, and what their respective strengths and weaknesses are. Given this information, we can now select which models we wish to use for a more comprehensive comparison between architectures. We had previously only compared specific CNN architectures to similar models, e.g. models with and without anti-aliasing, models with different max pooling kernel sizes, and models with different subsampling factors. We now compare these different types of architectures to each other, and also to MLPs. Furthermore, in this section we only focus on the comparison of inherent translation invariance, we further investigate learned invariance in Chapter 6, which implies that all models are trained on the standard, unaltered train sets.

We select the following models for the MNIST and CIFAR10 dataset:

- Baseline MLPs, as discussed in Section 3.3.3, consisting of 4x400 hidden layers for MNIST and 2x4000 for CIFAR. We use this model to compare the translation invariance of any given CNN architecture to a standard neural network.
- CNN with no subsampling and 2x2 max pooling, from Figure 5.1 for MNIST and from Table 5.3 for CIFAR. As this model had previously served as a baseline architecture, we include it to serve as a comparison reference for any other architecture that does make use of subsampling.
- CNN with 2x2 max pooling kernels and a subsampling factor of 8, from Figure 5.1 for MNIST and from Table 5.3 for CIFAR. This architecture has only slight filtering, but does make use of subsampling, as such we include it for comparison with other models that make use of stronger filters.
- CNN with 5x5 binomial anti-aliasing filters and a subsampling factor of 8, from Table 5.4 for both data sets. We choose this model as it had shown the best performance for model's that make use of Zhang's anti-aliasing method [43], and therefore we

wish to compare it to other architectures.

- CNN with 5x5 max pooling kernels and a subsampling factor of 8, from Table 5.3 for CIFAR, and we also train this architecture on the MNIST data set. We had not previously analysed this model for MNIST, but as it makes use of excessive filtering we include it in the comparison.

For this comparison we make use of the probability of top 1 change (PTop1) metric. Whilst other metrics such as cosine similarity provide finer measurements, for this comparison we require a definitive indicator of classification ability to accurately ascertain which models perform better than others. The comparative PTop1 for our selected MNIST models is shown in Figure 5.7 and in Figure 5.8 for CIFAR10.

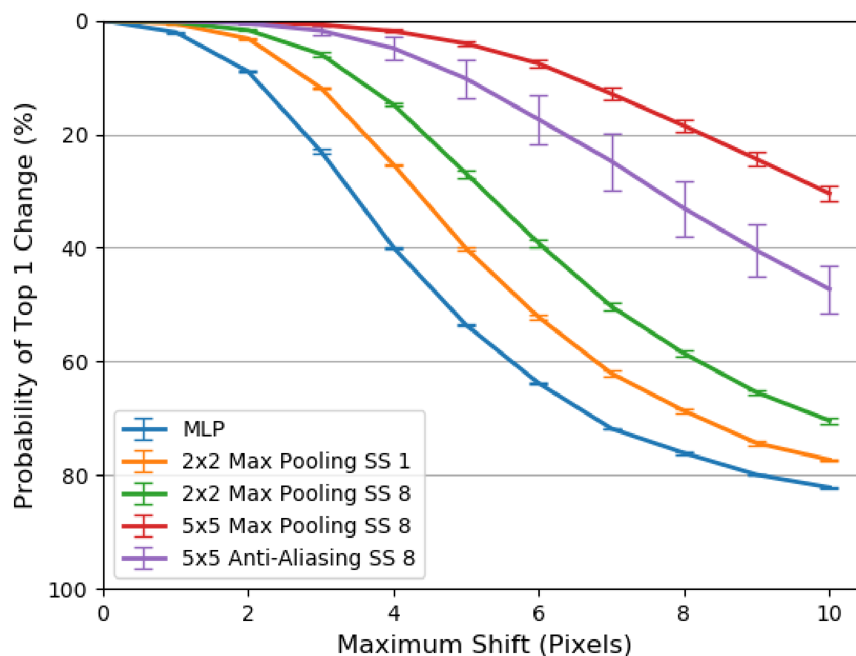


Figure 5.7: PTop1 change for translation invariance of selected MNIST architectures

Viewing the result of this architecture comparison for MNIST, we observe the following: Firstly, all of the selected CNN architectures outperform that of the MLP, regardless of other parameters. Furthermore, when comparing the networks that make use of stride to the model that does not, we see that subsampling is very beneficial. Even in the case

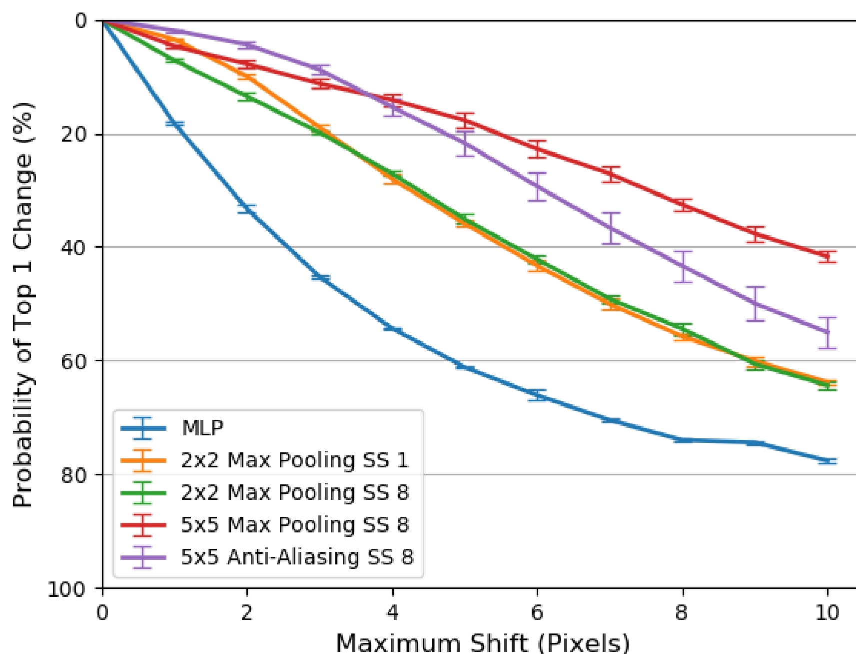


Figure 5.8: PTop1 change for translation invariance of selected CIFAR architectures

of light filtering, such as 2x2 max pooling kernels, the addition of stride can significantly improve the model's invariance to translation. Finally, we observe that the use of anti-aliasing further improves translation invariance when it is combined with subsampling, as this model is only outperformed by large 5x5 max pooling kernels.

In the case of CIFAR10, we see a very similar pattern as observed on MNIST, with one key exception: We observe that for the models that make use of 2x2 max pooling, the model without subsampling performs slightly better than its subsampled counterpart. This is expected, as we had already established that this is the case in Section 5.3. Before drawing any concrete conclusions from these results, we also include the test accuracy (averaged over three different initialisation seeds) for each model, this is shown in Table 5.11.

Table 5.11: Test accuracy for MNIST and CIFAR10 Models, averaged over three initialization seeds

	MLP	**2x2 Max SS 1**	**2x2 Max SS 8**	**5x5 AA SS 8**	**5x5 Max SS 8**
MNIST	98.44	99.34	99.27	99.42	99.43
CIFAR	57.66	72.33	72.53	73.4	75.95

Given these results, we can now draw conclusions surrounding the inherent translation invariance of different architectures:

1. CNNs outperform MLPs in translation invariance, regardless of other parameters. In the case of both data sets, MLPs are significantly more susceptible to shifts of the input data than convolutional networks (shown by Figures 5.7 and 5.7). This reinforces the idea that CNNs are much better at handling spatial data in general than MLPs, and are therefore more invariant to transformations of its input. Whilst the argument could be made that this is simply the case of poor generalization, we find that in the case of MNIST the MLP achieves similar test accuracy to that of the CNN architectures (Table 5.11).

2. The use of subsampling after adequate filtering improves translation invariance. These results (Figures 5.7 and 5.7) verify our hypothesis surrounding local homogeneity and shiftability, and we find that the degree of filtering required is dependent on the inherent homogeneity of a given data set. As mentioned earlier, we find that 2x2 max pooling is sufficient for MNIST (shown in Figure 5.1), but in the case of CIFAR10 greater filtering such as 5x5 anti-aliasing/max-pooling is required for subsampling to be beneficial (shown in Tables 5.3 and 5.4).

3. Anti-aliasing prior to subsampling benefits translation invariance. We have verified the efficacy of Zhang's anti-aliasing method [43], and find that our results consistently show that 5x5 binomial filters greatly improve translation invariance (Figures 5.7 and 5.7), and do not reduce generalization ability (Table 5.11). Which is consistent with what is shown earlier in Tables 5.4 and 5.10.

5.9 Conclusion

In this chapter we investigated the following:

- The effect of stride and filtering on translation invariance (Section 5.3)

- The effect of anti-aliasing on translation invariance (Section 5.4)
- The effect of no subsampling and global average pooling on translation invariance (Section 5.5)
- The effects of data translation prior to training, or learned invariance (Section 5.6)
- The comparison of MLPs and several CNN architectures in terms of translation invariance (Section 5.8)
- The relation of the above mentioned methods to generalization (Sections 5.7 and 5.8)

Our main findings are summarised below:

- Subsampling can greatly benefit translation invariance, given that it is combined with sufficient local homogeneity. Local homogeneity is the property where neighbouring pixels in a given window are similar to each other:
  - Greater filtering leads to greater local homogeneity.
  - If sufficient filtering is used, which provides sufficient local homogeneity, greater subsampling leads to greater translation invariance. This is due to subsampling reducing signal movement when an input is translated. Local homogeneity is required to ensure signal similarity, meaning translated outputs are still sufficiently similar to their untranslated counterparts when subsampling is used.
  - The amount of filtering required depends on the inherent homogeneity of a given data set.
  - Too much filtering or subsampling negatively affects generalization, as such, a trade-off exists between translation invariance and generalization.
- Anti-aliasing filters increase an architecture's translation invariance.
  - Anti-aliasing performs well in providing better local homogeneity, and when combined with subsampling it can greatly increase the model's translation invariance.

  - Anti-aliasing does not lead to any significant reduction in generalization capability.

- Global average pooling with no subsampling can ensure translation invariance, but can negatively affect generalization and leads to an increase in complexity.

- Data translation prior to training, or learned invariance, is very effective, but can not be expected to perform well in all cases.

- CNNs outperform MLPs in terms of translation invariance, even when no subsampling is used.

In the following chapter we further analyse translation invariance, along with rotation and scale invariance, through the use of spatial transformer networks.

Chapter 6

Spatial Transformer Networks

"Autobots, transform and roll out!" - Optimus Prime, a fictional character in the *transformers universe* [62]

6.1 Introduction

In this chapter we measure the transformation invariance of spatial transformer networks, and also compare their results to that of MLPs and CNNs.

Firstly, in Section 6.3, we explain the architecture configuration we use for each model, their relevant hyperparameters, and the protocol we use for testing. We first measure the translation invariance of these architectures, in Section 6.4, before moving on to rotation invariance (Section 6.5) and scale invariance (Section 6.6). Finally, in Section 6.7, we study the generalization abilities of spatial transformer networks and other architectures.

6.2 An overview of spatial transformer networks

Spatial transformer networks attempt to learn a corrective transform from data in order to make a network more invariant to transformation (as explained in Section 2.4). We classify spatial transformers as a form of *learned invariance*, as they do not apply a set, predefined operation to input features or feature maps. It would be expected that a spatial transformer module must be exposed to diverse (variability in spatial location, angle, and scale) samples in order to learn a corrective affine transformation. While the two data sets we have selected do contain some slight variations between samples in terms of spatial location, scale, and angle, it is unclear whether these slight differences would be sufficient for a spatial transformer to learn an adequate, corrective affine transformation. Furthermore, we have thus far only focused on translation invariance, and have not provided any empirical measurements surrounding scale and rotation invariance. We aim to address the following questions in this chapter:

- How do STNs compare to CNNs and MLPs in terms of transformation invariance, when trained on untransformed data?
- How do STNs compare to CNNs and MLPs in terms of transformation invariance, when trained on transformed data?
- How much data transformation is required for STNs to learn corrective transformations?
- How well do STNs generalize?

In the following sections, we compare STNs, CNNs, and MLPs in terms of their transformation invariance - for architectures trained on both transformed and untransformed data. Finally, we analyse and compare the generalization capabilities of these models.

6.3 Experimental setup

Spatial transformer modules can be inserted into any deep learning architecture, and can be applied to the initial input image and/or intermediary feature maps [10]. We combine these modules with well-performing CNNs, which allows us to make comparisons between STN and CNN architectures. For each spatial transformer network (STN), the spatial transformer module is inserted as an initial input transformer before its output is passed to the CNN model, as illustrated in Figure 6.1. Furthermore, we use the same CNN architectures (without the added spatial transformer module) as models with which to compare the performance of the STN-CNNs. We refer to the spatial transformer network models as either "STN-CNN", or the shortened "STN", and the convolutional neural network architectures as either "CNN" or the more descriptive "normal CNN". The localisation network of each spatial transformer module is specified earlier in Section 3.3.2, and the sampler module of each spatial transformer uses bilinear interpolation [38]. We now describe the specifics of each CNN architecture for each data set.

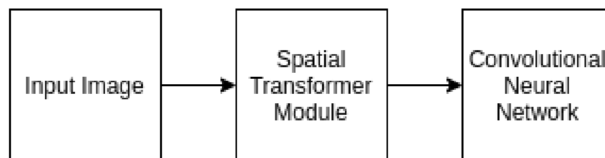


Figure 6.1: Spatial transformer network layout

For the MNIST CNN, we use one of the architectures illustrated in Figure 5.1, with a subsampling factor of 8. This architecture is shown in Table 6.1. We train this network with a learning rate of 0.0001, and batch size of 128, for both the CNN and STN-CNN. All other hyperparameters and training parameters are as specified in Section 3.3.4. The CNN achieves a test accuracy of 99.30%, while its spatial transformer counterpart (STN-CNN) achieves 99.00%, with the accuracy averaged over three initialization seeds (in both cases). Both models also converge to 100% train accuracy.

In the case of CIFAR10, we use an architecture similar to that used in Table 5.3, with a subsampling factor of 8, and pooling kernel size of 4. However, we alter the original architecture and double the number of channels of each convolution layer. This configu-

Table 6.1: CNN architecture for MNIST with three convolution (Conv) and pooling (Pool) layers, and two fully connected (FC) layers

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**
Conv	1	32	3x3	1	1
Pool	32	32	2x2	2	0
Conv	32	64	3x3	1	1
Pool	64	64	2x2	2	0
Conv	64	128	3x3	1	1
Pool	128	128	2x2	2	0
FC	3 200	400	N/A	N/A	N/A
FC	400	200	N/A	N/A	N/A
FC	200	10	N/A	N/A	N/A

ration is shown in Table 6.2. We make use of a learning rate of 0.0001, and a batch size of 128, while the other hyperparameters are as specified in Section 3.3.4. While it might appear strange that the learning rate and batch size are the same for both the MNIST and CIFAR10 architectures, this is purely coincidental - both networks were thoroughly tested with several sets of hyperparameters to find those that perform best. Averaged over three initialization seeds, the CNN performs exceptionally well with a test accuracy of 79.04%, while the STN-CNN achieves 77.27%.

Table 6.2: CNN architecture for CIFAR with three convolution (Conv) and pooling (Pool) layers, and two fully connected (FC) layers

Layer	**Input Channels**	**Output Channels**	**Kernel Size**	**Kernel Stride**	**Padding**
Conv	1	64	3x3	1	1
Pool	64	64	4x4	2	1
Conv	64	128	3x3	1	1
Pool	128	128	4x4	2	1
Conv	128	256	3x3	1	1
Pool	256	256	4x4	2	1
FC	9 216	400	N/A	N/A	N/A
FC	400	200	N/A	N/A	N/A
FC	200	10	N/A	N/A	N/A

For MLP architectures, we make use of the models specified in Section 3.3.3, that we had previously used in Chapter 5. As mentioned earlier, the CIFAR MLP has an average test accuracy of 57.66%, while the MNIST MLP 98.44%.

In the case of each transformation, we transform the samples in the test set (those correctly classified prior to transformation) by randomly sampling from a given range, as explained in Section 3.4, and then compare the output classification vectors to that of the untransformed outputs. In this chapter we only make use of the probability of top 1 change metric (PTop1 change). We use this metric as we are not concerned with secondary invariance effects, and aim only to compare absolute performance in terms of invariance. As explained in Section 3.4, the PTop1 metric is useful when comparing architectures with vastly different configurations, and provides us with an absolute comparative indication of their transformation invariance.

For each type of transformation, we first measure the invariance of the architectures specified in this section trained on the standard, unaltered train set. We then repeat the analysis for the same architectures trained with transformed data, where the train set is transformed according to the specific transformation being investigated. We sometimes refer to the architectures specified in this section as "baseline architectures", as they are trained on the standard training set. Once again, all results are averaged over three initialization seeds, and error bars are included indicating the standard error in each figure.

6.4 Translation invariance

We have extensively studied translation invariance in Chapters 4 and 5, and therefore find it fitting to assess the performance of spatial transformer networks on the translation transformation first. We analyse STNs, CNNs, and MLPs on the CIFAR10 data set. Given that the CNN we use has a subsampling factor of 8, and 4x4 max pooling kernels, it should fair relatively well in terms of translation invariance (as shown earlier in Chapter 5). While CIFAR10 does have some slight variability between samples in terms of spatial locations, it is unclear whether these small variations would be sufficient for the STN-CNN to learn a corrective translation. Furthermore, as this specific CNN architecture (the normal CNN architecture) is then expected to perform well, we must ascertain whether

the STN-CNN can improve its performance.

We measure the translation invariance of our specified MLP, CNN, and STN CIFAR10 models, without any prior translation of the training set. The PTop1 change for all three models are shown in Figure 6.2, up to a maximum translation range of 10 pixels.

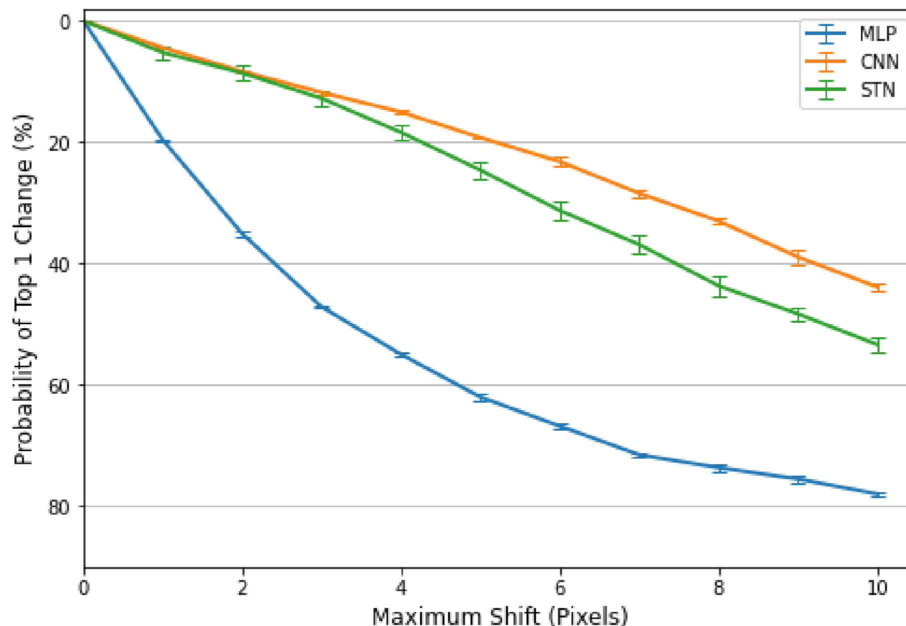


Figure 6.2: PTop1 change translation invariance comparison for CIFAR10 architectures without train set translation

Surprisingly, this result shows that the STN performs *worse* than that of the normal CNN, and quite significantly. At a maximum shift of 10 pixels, the CNN has a 43.90% probability of changing its classification for a sample, whilst the STN has a probability of 53.41%. Both models still greatly outperform the MLP (78.02% probability), but there is nearly a 10% discrepancy between the CNN and STN. This counter-intuitive result suggests that the STN has not been able to learn a corrective translation from the data set, and seems to be doing the opposite (not correcting the translation, but perhaps erroneously transforming the sample).

While the STN shows sub-par performance, one must recall that tiny images data sets, such as CIFAR10, are biased in terms of the spatial location of objects, with little variability between samples. We have previously mentioned and explained the bias of these

data sets in Section 2.5. To better measure the translation invariance capabilities of this STN, we retrain each architecture shown in Figure 6.2 with translated data. Each sample within the train set is randomly translated up to a maximum of 5 pixels, and the previous experiment is repeated. The PTop1 change of these models are shown in Figure 6.3.

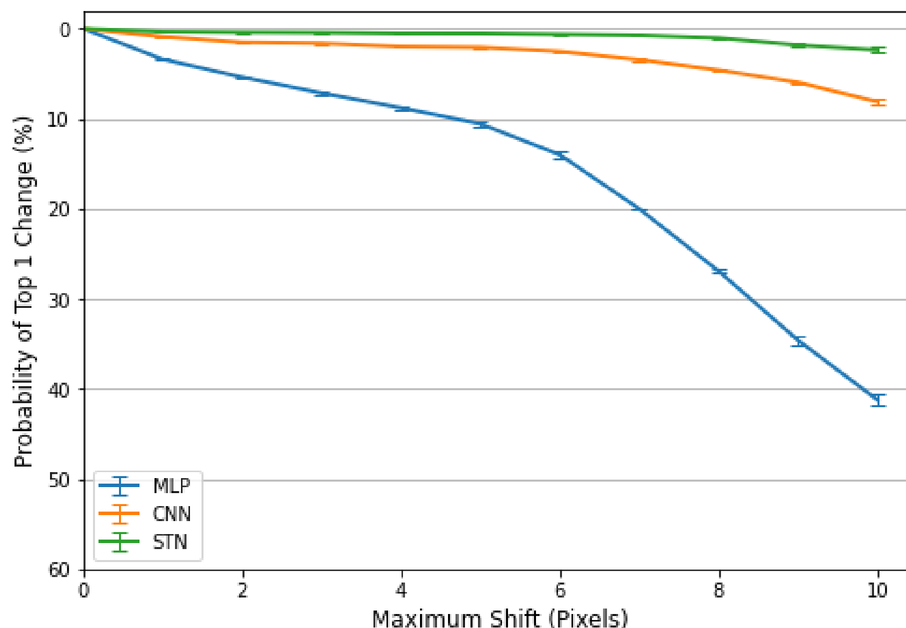


Figure 6.3: PTop1 change translation invariance comparison for CIFAR10 architectures with train set translation (5 pixel maximum train set shift)

In this case, the STN performs much better, and outperforms the normal CNN (and MLP) over the full range of translation tested. The STN model exhibits almost perfect translation invariance, with a mere 0.52% PTop1 change at a maximum shift of 5 pixels, and 2.36% at 10 pixels. We observe that the normal CNN model deteriorates much quicker at shifts greater than 5 pixels (the maximum shift that the training set was subjugated to) than its STN counterpart. The normal CNN has a 8.10% PTop1 change at 10 pixels, which is significantly worse than the STN. This result then suggests that the STN has learned to apply a corrective translation, and is able to significantly increase the model's invariance to translation.

Given these two results, we can now draw conclusions surrounding the translation invariance of the three architectures presented:

- In the case of CIFAR10, the standard, unaltered training set does not contain sufficient variability in spatial positioning for a spatial transformer module to learn a corrective translation. We observe that the STN performs significantly worse ($\sim$ 10% higher PTop1 at 10 pixels) than its counterpart without a spatial transformer module.

- When training on translated data, the STN is able to learn a corrective translation. We observe that the STN outperforms both the CNN and MLP in terms of translation invariance in this case, and delivers near perfect performance. Compared to the STN, the CNN and MLP have $\sim$ 6% and $\sim$ 40% higher PTop1 at a maximum translation of 10 pixels, respectively.

- CNNs and STN-CNNs both outperform MLPs in terms of translation invariance. These results clearly show that MLPs are significantly less invariant to translation in both experiments than the other models, whether trained on data translation or not. This result further emphasises our conclusion from Chapter 5, which argues that CNNs are naturally better at handling spatially related data than standard MLPs.

The results of this section indicate that spatial transformers require a significant amount of exposure to a specific transformation in order to learn a corrective one. In the following section, we explore how this concept relates to rotation invariance.

6.5 Rotation invariance

In this section we turn our attention towards rotation invariance, and compare the performance of the STN, CNN, and MLP architectures.

6.5.1 Overview of rotation invariance

When considering rotation invariance in CNNs, rotation is distinct from translation in the sense that convolutional filters are not equivariant to rotation [42]. The translation equivariance property stems from the fact that the convolution operation is translational in nature. While convolution filters are shifted during the convolution operation, they are not rotated. This implies that the same weights of the convolution filters are no longer applied to the same input features when rotating the input (compared to that of an unaltered sample). It then follows that the convolved output of a rotated sample is not equivalent to its upright counterpart.

The lack of rotation equivariance in CNNs poses a particularly difficult problem, as this implies that different filter parameters must be learned for samples which are rotated differently. We use both the MNIST and CIFAR10 data sets to analyse rotation invariance. These data sets do contain variations in angle between samples, but it is unclear whether this would be sufficient for any of the architectures to learn representations which are invariant to rotation. In the following sections we compare the rotation invariance of these architectures for models trained on standard and rotated data.

6.5.2 Rotation invariance - MNIST

We first turn our attention towards the MNIST data set. We use the three architectures (MLP, CNN, and STN) specified earlier in Section 6.3, trained on an unaltered train set. We measure the PTop1 change up to a maximum range of 70 degrees rotation (clockwise and counter-clockwise, as the range is for both positive and negative rotation) on the MNIST test set. The result of this analysis is shown in Figure 6.4.

This result shows that all three networks are relatively rotation invariant up to a maximum rotation of 15 degrees, but performance quickly degrades for a larger ranges of rotation. Furthermore, we observe that the STN outperforms the normal CNN, albeit only slightly. At a maximum rotation of 70 degrees, the STN has a PTop1 change of 30.03%, and the

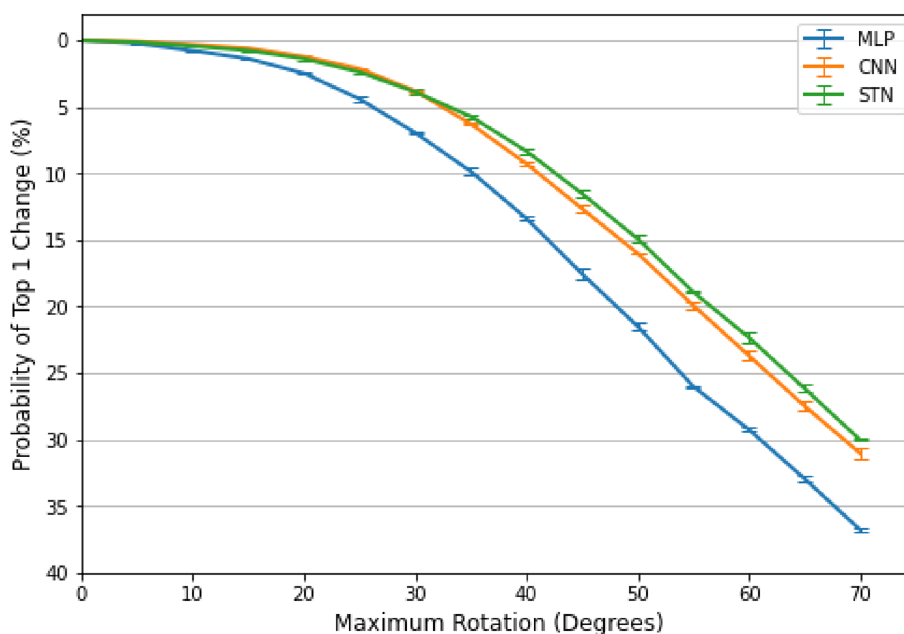
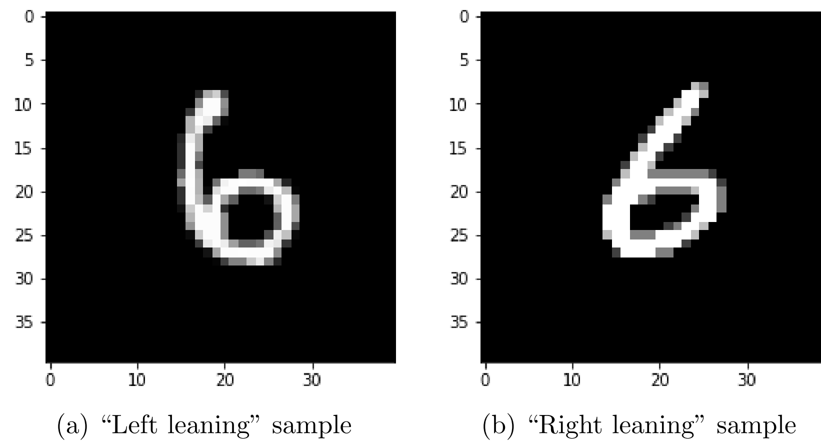


Figure 6.4: PTop1 change rotation invariance comparison for MNIST architectures without train set rotation

CNN a slightly higher 31.03%.

We have mentioned that CNNs are not equivariant to rotation, but this result could be explained by the fact that in the case of MNIST, many samples do exhibit slight rotation. Different people naturally angle each digit slightly differently due to differences in handwriting. This is illustrated in Figure 6.5, which shows two unaltered digits from class 6 of the MNIST train set, each angled differently. Furthermore, for MNIST class 0, the samples are (mostly) inherently rotation invariant, as rotating a zero still results in an approximate zero.

This inherent variability in angle of the data set further explains why the MLP models achieves comparable performance, with a 36.78% PTop1 change at a maximum of 70 degrees of rotation. These differences then imply that a significant amount of rotation invariance is learned from the data set, and also that the STN is able to learn a slight corrective rotation transformation. It would appear that the convolutional kernels of the CNN have also learned representations which are more rotation invariant, but this assumption is difficult to verify.



(a) “Left leaning” sample (b) “Right leaning” sample

Figure 6.5: Two class 6 MNIST samples angled differently

To further assess the rotation invariance capabilities of these architectures, we repeat the previous experiment, and retrain all three models on a rotated train set. Each sample in the train set is randomly rotated up to a maximum of 45 degrees. The PTop1 change for all three models, up to a maximum rotation of 70 degrees, is shown in Figure 6.6. Note: the scale of the y-axis is much smaller than that of Figure 6.4.

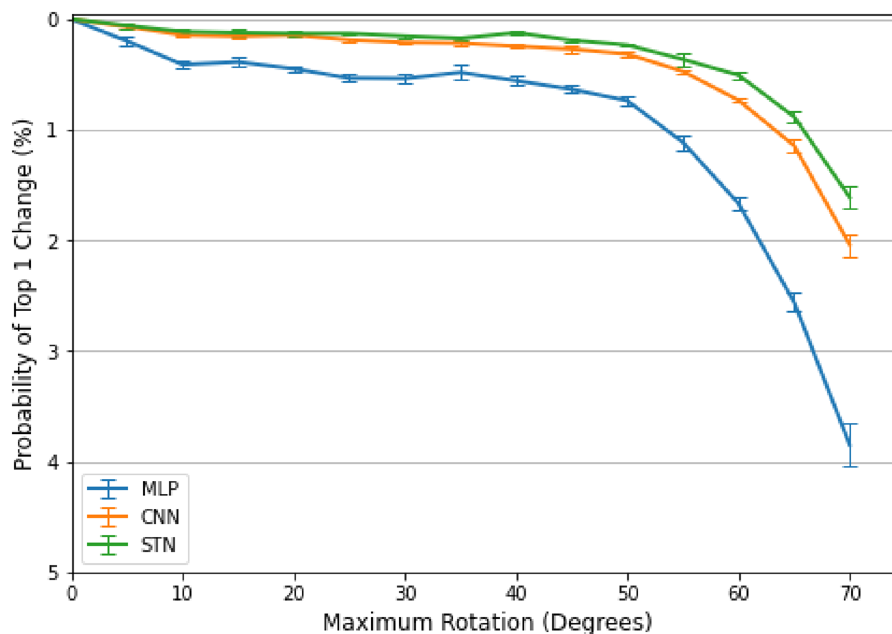


Figure 6.6: PTop1 change rotation invariance comparison for MNIST architectures with train set rotation (45 degrees maximum train set rotation)

We observe that when trained on rotated data, these architectures are highly invariant to

rotation. Once again, the STN slightly outperforms the normal CNN, but the difference is almost negligible - only a difference of $\sim 0.4\%$ at a maximum rotation of 70 degrees. In this case, the MLP is also highly invariant, with a PTop1 change of less than 4% for a rotation range of 70 degrees. Furthermore, all three networks decrease in performance when confronted with rotations greater than 45 degrees.

While these results are certainly informative, we repeat these experiments on the CIFAR10 data set in the following subsection before drawing any conclusions.

6.5.3 Rotation invariance - CIFAR10

We use the three baseline CIFAR10 architectures specified earlier, with no prior rotation of the train set, and measure their rotation invariance up to a maximum range of 70 degrees. The result of this analysis is shown in Figure 6.7, once again using the PTop1 change metric.

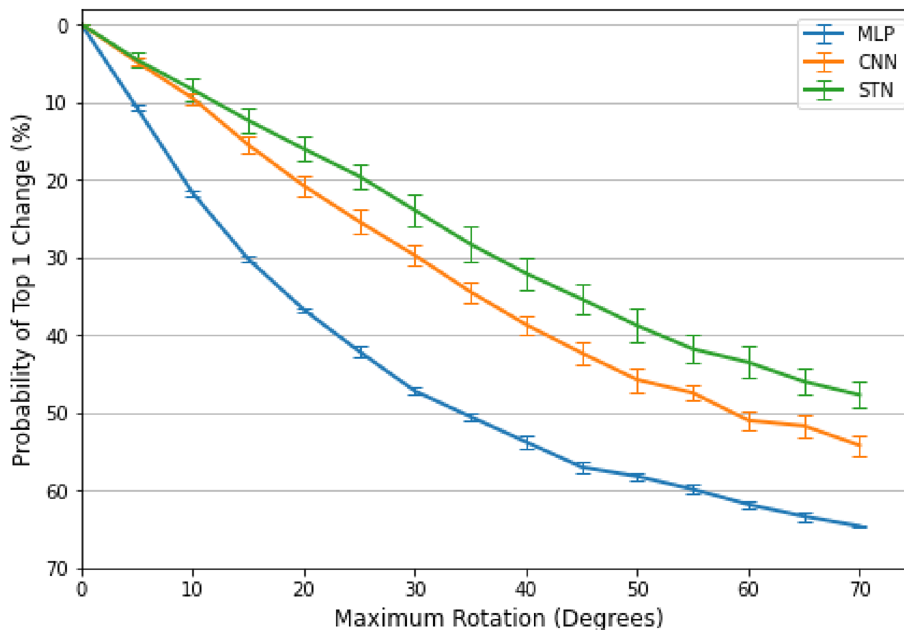
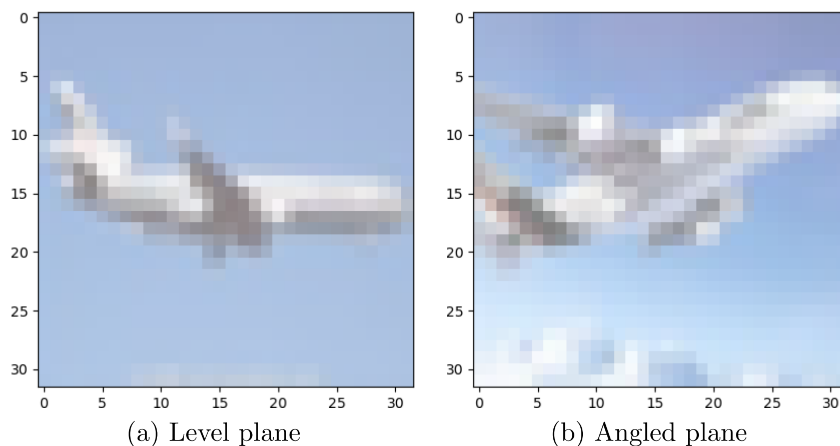


Figure 6.7: PTop1 change rotation invariance comparison for CIFAR10 architectures without train set rotation

We observe a pattern similar to that seen on MNIST. The STN model performs better than its CNN counterpart, and in this case the difference is more significant ($\sim$ 6.5%

difference in PTop1 change). Once again, both the STN and CNN outperforms the MLP architecture. The performance of the STN seems to suggest that it has learned a slight corrective rotation, implying that CIFAR10 also has some variability in angle between samples. Figure 6.8 illustrates two samples from the "plane" class, each angled differently, which supports this assumption.



(a) Level plane (b) Angled plane

Figure 6.8: Two CIFAR samples from the "plane" class angled differently

As a final assessment of rotation invariance, we repeat the previous experiment, but train each model on a rotated train set. Each train set sample is randomly rotated up to a maximum of 45 degrees. The PTop1 change, up to a maximum rotation of 70 degrees, for these models are shown in Figure 6.9.

The result of training with rotated data with CIFAR10 is very surprising, and markedly different from that observed on MNIST. While the STN is able to correct rotations up to 45 degrees, its performance quickly degrades for larger rotations. While the normal CNN exhibits a similar pattern, the deterioration in performance is much more gradual, and outperforms the STN for rotations greater than 45 degrees.

While it is difficult to provide and verify a specific explanation for the behaviour of the STN, we do offer a hypothesis. It is possible that the spatial transformer module has learned to apply a corrective transformation up to a maximum of 45 degrees, and adequately corrected each sample during the training process. This would then imply that

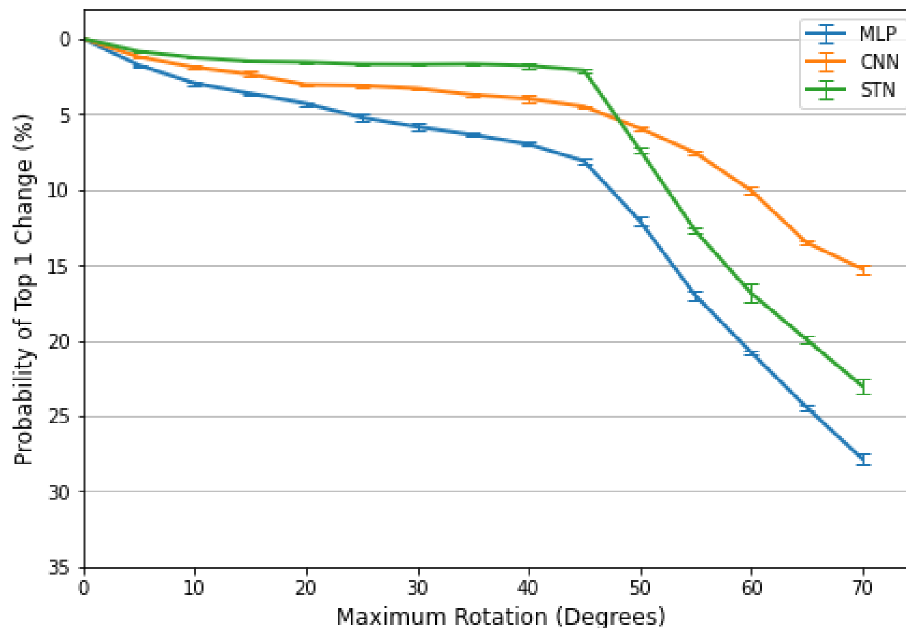


Figure 6.9: PTop1 change rotation invariance comparison for CIFAR architectures with train set rotation (45 degree maximum train set rotation)

the convolutional filters and fully connected layers that follow, were not exposed to rotation during training. Whereas in the case of the normal CNN, both the convolutional and fully connected parameters seem to have learned more rotation invariant representations. It then follows that for rotations greater than 45 degrees, the spatial transformer module is no longer able to completely correct the sample, and its subsequent convolution and fully connected parameters have not learned an invariance to rotation.

If this hypothesis is correct, why do we not see a similar result on the MNIST data set? This could possibly be due to the manner in which we rotate each sample. Given that we rotate the entire image, it implies that the object of interest within the image is rotated along with its environment (e.g. a cat is rotated along with the floor it sits on). In the case of MNIST, there is no "environment" to consider, as the samples have an empty (zero value) background. Conversely, for CIFAR10 the entire square color image (the object of interest and its environment) is rotated within a black (zero value) background. This implies there is a large color differential between the actual "image" and its background, which could "fool" the spatial transformer module, and not allow adequate correction for larger rotations. For a true measure of the rotation invariance of a STN, we would

require a data set that contains objects rotated relative to their environment. Given that such a data set is not readily available, we are resigned to simulating a realistic rotation by rotating the entire image. Additionally, another solution might be to use a form of "color averaging" to fill in the background with pixel values that more closely resemble the images' environment.

6.5.4 Conclusion

In this section we have measured the rotation invariance of STNs, CNNs, and MLPs. We have conducted experiments for both the MNIST and CIFAR10 data sets, with rotated as well as unaltered training data. Our main findings with regard to these two tasks are summarised below.

- Both the MNIST and CIFAR10 data sets contain sufficient variability in the angle of its samples, and spatial transformer networks are able to learn a slight corrective rotation from these data sets. Furthermore, STNs slightly outperform CNNs when trained on a normal train set. We observe a difference, compared to the STN, of $\sim 1\%$ and $\sim 6.5\%$ in PTop1 change for the MNIST and CIFAR10 CNN respectively, at a maximum rotation of 70 degrees.
- All three MNIST architectures considered are highly invariant to rotation when trained on rotated data, and the STN slightly ($\sim 0.4\%$ lower PTop1 change) outperforms a normal CNN.
- In the case of CIFAR10, the spatial transformer module is only able to adequately correct rotations up to the maximum rotation of the train set. Additionally, CNNs outperform STNs for larger rotations, we observe that the CNN has a $\sim 8\%$ lower PTop1 change at a maximum rotation of 70 degrees. We attribute this to our data set rotation protocol, which is not able to accurately simulate realistic rotation. However, we do not investigate this hypothesis further.
- MLPs fair the worst with regards to rotation invariance of all three architectures,

in all cases for both data sets.

In the following section, we repeat similar experiments in order to measure the scale invariance of STNs, CNNs, and MLPs.

6.6 Scale invariance

In this section we analyse scale invariance, and do a comprehensive comparison between architectures for both down and up-scaling.

6.6.1 Overview of scale invariance

Convolutional neural networks are not equivariant to scale transformations [29]. The size of the convolutional filters are not varied throughout the convolution operation, as such a scaling of the input implies that the resulting output would not be equivalent to its normal-scale counterpart.

While scale invariance is of great importance to image recognition (as described in Section 2.5), it is a particularly difficult problem in CNNs. However, some solutions have been demonstrated to be effective. Xu et al. [45] makes use of a multi-column CNN, were each column contains up or down-scaled versions of standard sized convolution filters. Similarly, Noord and Postma [41] use an ensemble technique, where each CNN within the ensemble has different convolution kernel sizes. These solutions then suggest that the size of the convolutional filter is critical, but that this size is specific to a certain image scale. Given this intuition, we choose to analyse down and up-scaling separately. This provides us with a clearer indication of the abilities of CNNs and STNs as pertaining to scale invariance.

6.6.2 Up-scaling

We measure the up-scale invariance of our three baseline CIFAR10 architectures (trained on an unaltered train set), up to a maximum of 200% the scale of the original test set sample. The PTop1 change for these models are shown in Figure 6.10.

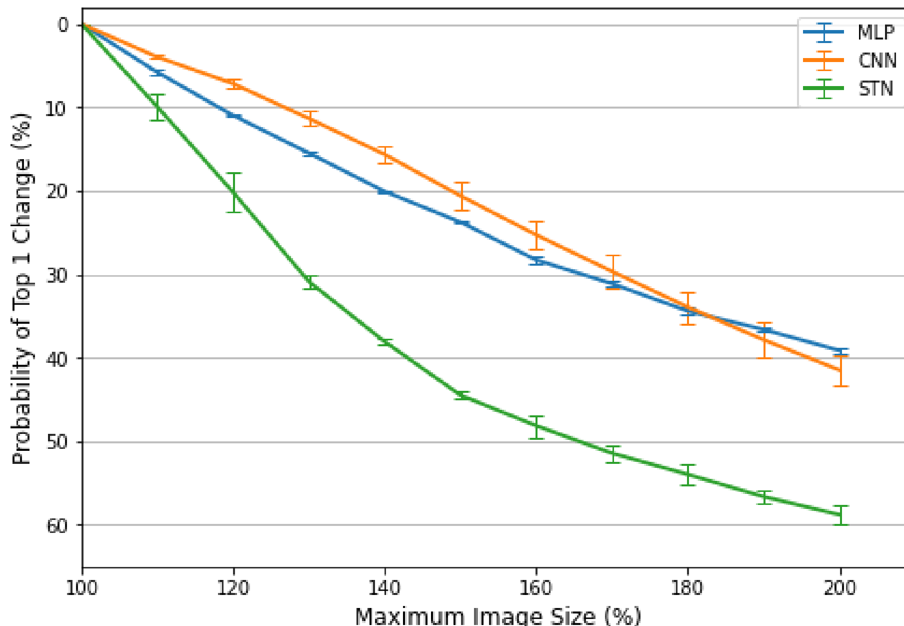


Figure 6.10: PTop1 change up-scale invariance comparison for CIFAR architectures without train set scaling

The result of this comparison is surprising. We observe that the STN performs significantly worse than the CNN, and in fact, worse than the MLP architecture ($\sim$ 20% higher PTop1 change at maximum scale of 200%). Furthermore, the MLP seems to deliver the best results for larger scale transformations. The sub-par performance of the STN model indicates that the CIFAR10 train set does not contain enough variability in scale to learn any useful corrective transformation (with regards to larger scales). As was the case with the translation invariance analysis of Figure 6.2, it appears that the spatial transformer module is not correcting the transformation, and is in fact doing the opposite (perhaps incorrectly transforming the input, worsening its invariance). The comparable performance of the CNN and MLP could be attributed to the fact that the convolutional filters are rather small (3x3 kernels) and do not seem to have learned scale invariant representations. As such they do not seem to be invariant with regards to larger scale images, and are

not able to provide any additional benefits in comparison to an MLP. However, the previous statement is merely a hypothesis, and further experimentation would be required to adequately motivate the reason for this discrepancy.

Given the lackluster performance of the CNN and STN models, we repeat this experiment after retraining these architectures on a scaled data set. Each training set sample is randomly scaled up to a maximum of 150% its original scale. The PTop1 change of these models on the CIFAR10 test set is shown in Figure 6.11. Note: the scale of the y-axis is much smaller than that of Figure 6.10.

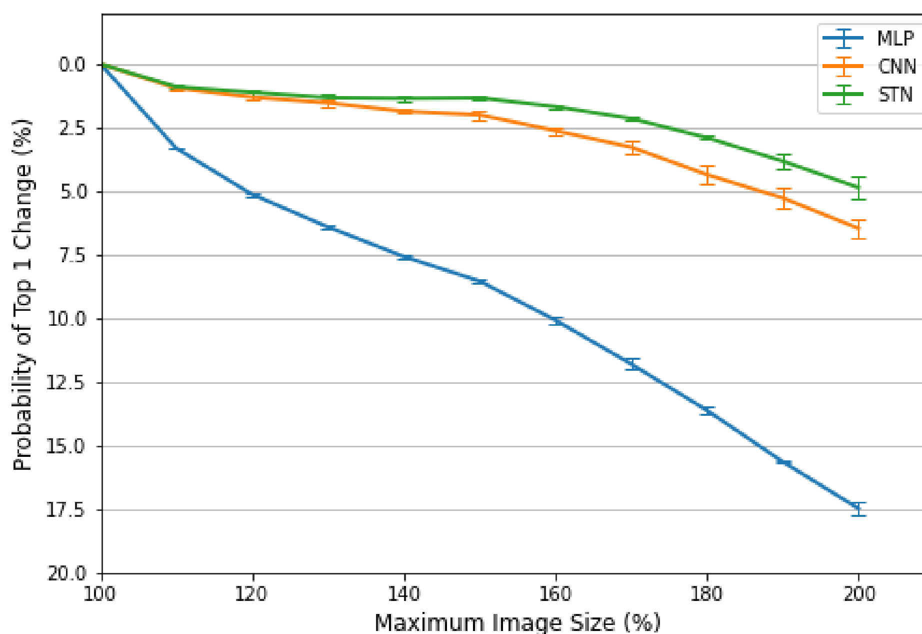


Figure 6.11: PTop1 change up-scale invariance comparison for CIFAR architectures with train set scaling (150% maximum up-scaling of the train set)

This results shows that these models are much more invariant than those illustrated in Figure 6.10, and furthermore the STN performs exceptionally well. It can be assumed that the spatial transformer module has learned to correct for up-scale transformations, which explains its improved performance in comparison to the normal CNN. Similarly, the CNN also performs very well, indicating that the convolutional filters have learned representations which are more scale invariant. The CNN only exhibits a slightly higher PTop1 change of 6.46%, compared to the 4.86% of the STN at a maximum scale of 200%. In this case, the MLP performs far worse than the other architectures, but still much

better than the models shown in Figure 6.10 (17.48% PTop1 change at maximum scale of 200%).

Before drawing any conclusions surrounding scale invariance, we also investigate down-scale invariance in the following section.

6.6.3 Down-scaling

We now turn our attention towards down-scale invariance. We use our previously specified baseline architectures, and measure their invariance to down-scaling. The PTop1 change for all three models is shown in Figure 6.12, down to a minimum of 50% the original scale.

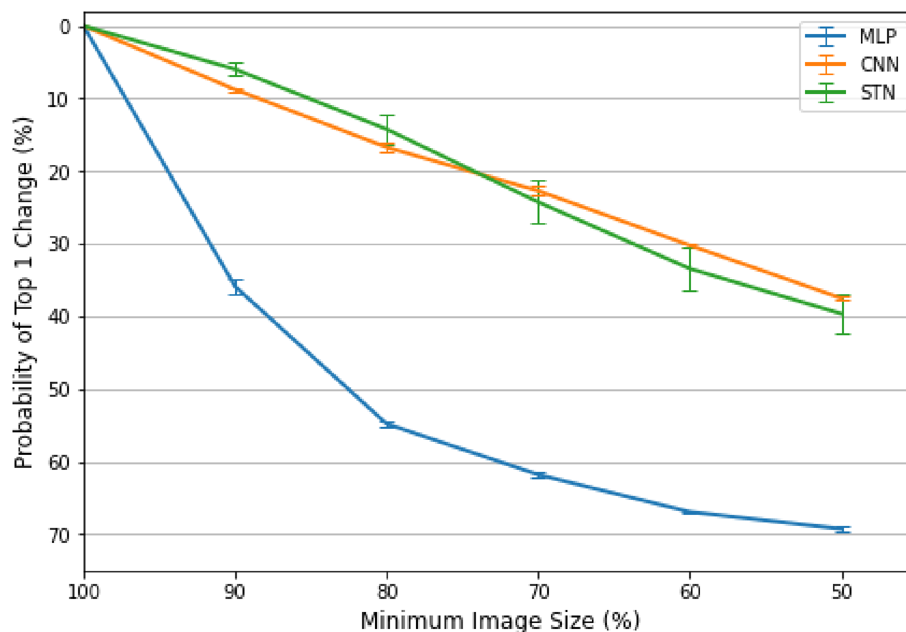


Figure 6.12: PTop1 change down-scale invariance comparison for CIFAR architectures without train set scaling

This result is vastly different from that observed when up-scaling the image samples (shown in Figure 6.10). The STN and CNN achieve similar performance, and additionally perform much better than the MLP. This would suggest that the STN is able to slightly correct for down-scaling of the image. It then follows that the CIFAR10 train set has some slight scale variability between samples (in terms of smaller scales than previously analysed).

We repeat this experiment on scaled data. Each sample of the train set is randomly scaled down to a minimum of 75% of its original scale, and the previous architectures are retrained. The down-scale invariance comparison for these models are shown in Figure 6.13. Note: the scale of the y-axis is smaller than that of Figure 6.12.

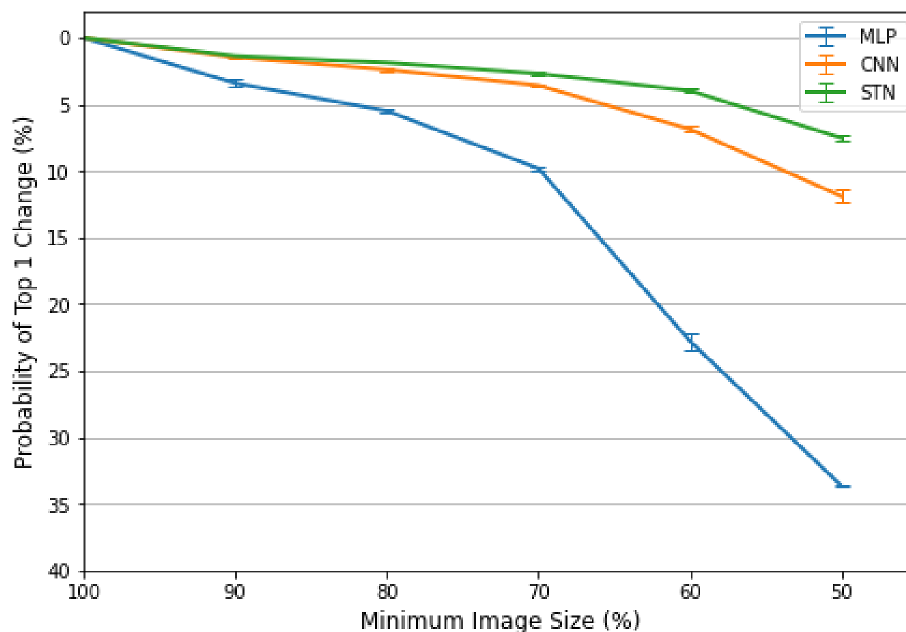


Figure 6.13: PTop1 change down scale invariance comparison for CIFAR architectures with train set scaling (75% minimum down-scaling of the train set)

This result is aligned with the previous results on transformed data. We observe that the STN performs best, albeit only a slight improvement with regards to the CNN. The CNN has a PTop1 change of 11.90% at a minimum down scaling of 50% of the original image scale, while the STN exhibits a slightly better 7.53%. As previously observed, the MLP fairs worst, with a 33.66% PTop1 change at the same range. In this case, the spatial transformer module has learned an adequate scale affine transformation, and outperforms the other architectures.

6.6.4 Conclusion

In this section we have studied the scale invariance of three different architectures (MLP, CNN, and STN) and made comparisons between their performance. We have handled the cases of up-scaling and down-scaling separately, and reached the following conclusions:

- In the case of up-scaling, the CIFAR10 train set does not contain any adequate variability between samples to allow a STN to learn a corrective transformation. Furthermore, the CNN and MLP models achieve very similar performance. The STN performs significantly worse than all other architectures (from Figure 6.10).
- Artificially up-scaling the CIFAR10 train set allows the STN model to learn a corrective transformation, and it slightly outperforms a CNN, where the STN has $\sim 2\%$ lower PTop1, at a maximum scale of 200%. Furthermore, the MLP performs the worst in comparison to the other architectures (from Figure 6.11).
- When considering down-scaling, the CIFAR10 train does contain some slight variability, and the STN achieves comparable performance to that of the normal CNN. Once again, the MLP performs far worse that the other architectures (from Figure 6.12).
- Artificially down-scaling the CIFAR10 train set further improves the performance of the STN, and that of the other architectures. The STN performs the best, with a 4.37% lower PTop1 than the CNN at a minimum scale of 50%. The MLP model once again performs significantly worse than the other architectures (from Figure 6.13).

In the following section we study the generalization abilities of STNs by comparing their test set accuracy to that of the other architectures.

6.7 Generalization of spatial transformer networks

In this section we contrast the generalization abilities of spatial transformer networks with CNNs and MLPs. The accuracy on the standard, unaltered test set for each CIFAR10 MLP, CNN, and STN-CNN analysed in the previous sections is shown in Table 6.3. The first row shows the accuracy of the baseline models, while the others show the accuracy for models trained on transformed data.

Table 6.3: Standard test set accuracy for CIFAR10 models trained on different train sets, averaged over three initialization seeds (SE < 0.29)

Training Data	**Architecture**		
	MLP	CNN	STN-CNN
Standard/Unaltered	57.66	79.04	77.27
Translated (5 pixel max)	59.47	80.99	79.19
Rotated (45 degree max)	57.34	77.59	78.04
Up-scaled (150% max)	57.45	78.32	78.62
Down-scaled (75% minimum)	57.78	78.77	78.25

Comparing the 'CNN' and 'STN-CNN' columns, we observe that generally these two architectures perform similarly, but the CNN appears to perform best overall, especially for the standard and translated train sets. However, in the case of the STN, we observe that transforming the train set in any way improves its generalization in comparison to the unaltered train set. This pattern is less clear with the CNN and MLP, where only translation of the train set seems to provide any significant improvement in generalization. This finding is concurrent with our earlier test accuracy results for different CNN models trained on translated CIFAR10 data, shown in Table 5.7 of Chapter 5. It would appear that some variation in spatial location is critical for better performance on the test set. This would be a logical assumption, as even in the case of biased data sets where objects tend to be centered, the test set can contain some slight variation in the location of its features. For example, in the case of the "angled" plane in Figure 6.8 (b), the wings of the aeroplane are higher up, and more to the left relative to the "level" plane in Figure 6.8 (a). It would then be logical to assume that such variation would also occur in the test set samples.

The STNs improvement in generalization when trained on transformed data supports our

findings surrounding invariance, where we observed that STNs often decrease invariance (compared to a CNN) if trained on the standard train set, and could possibly be incorrectly transforming the input samples. By exposing a STN to transformed data, it learns specific corrective transformations, and ultimately generalizes better to unseen samples, compared to its counterpart trained on standard data. The results of Table 6.3 also raise the question of what is meant by "generalization". Viewed in the strict sense, the term merely points to test set accuracy, and "generalization error" to the difference in accuracy between the train and test set [63]. More broadly speaking, the term points to classification performance on samples that were not seen during training, including those sampled from a different distribution, such as those sampled from "the greater visual world" (as described in Section 2.5). To better understand the performance of STNs in the latter sense of the term, we once again measure the test accuracy of the models shown in Table 6.3, but also apply the specific transformation in each case to the test set. The result of this analysis is shown in Table 6.4, which shows the accuracy on a transformed test set for each model. We apply the same transformation to the test set as that applied to the train set. For example, for the models trained on translated data, the test set samples are also translated up to a maximum of 5 pixels.

Table 6.4: Transformed test set accuracy for CIFAR10 models trained on transformed train sets, averaged over three initialization seeds (SE < 0.32)

Train and test data	**Architecture**		
	MLP	CNN	STN-CNN
Translated (5 pixel max)	58.04	80.21	79.14
Rotated (45 degree max)	56.53	75.07	76.94
Up-scaled (150% max)	57.72	79.12	78.56
Down-scaled (75% minimum)	58.35	77.89	76.80

Surprisingly, the CNN and STN-CNN once again perform very similarly. We had previously concluded that STNs usually provide a small improvement in terms of transformation invariance, but this difference seems to disappear when only considering accuracy. As a further assessment of a STNs performance, we train our baseline CNN and STN-CNN (as specified in Section 6.3) on a training set with mixed transformations. Each train set sample has the following transformations applied to it:

- Translation up to a maximum of 5 pixels

- Rotation up to a maximum of 45 degrees
- Scale transformation between 75% and 150% of the original scale

We measure the test accuracy of these models on the standard test set, and then also on a test set with the same transformations applied. To optimize each model, we choose the epoch with the highest validation accuracy, where the validation set is also subject to the same transformations as the train set. This is shown in Table 6.5.

Table 6.5: Standard and transformed test set accuracy for CIFAR10 networks trained on data with mixed transformations, averaged over three initialization seeds

Test data	**Architecture**	
	CNN	STN-CNN
Unaltered	79.82	78.88
Transformed	76.21	77.82

We observe that once again the CNN performs better than the STN on the standard, unaltered test set. However, for the transformed test set, the STN performs slightly better (1.61% higher classification accuracy). We would expect that given the STNs better transformation invariance, it would provide a significant benefit with regards to generalization on transformed test samples. However, this does not seem to be the case, as the difference between these models in both Tables 6.4 and 6.5 appears to be minute. This then raises the question of why the STN does not seem to provide any significant benefits in terms of generalization. We offer the following hypothesis:

- Over-fitting. We believe that the spatial transformer module learns corrective transformations which are overly biased towards transformations of the train set. Given that we do not make use of any explicit regularization, it is possible that the STN's performance can be greatly improved using some form of regularization. While this is merely a hypothesis, we do point out the following anecdotal evidence: As we make use of early stopping, the epoch exhibiting the highest validation accuracy during training is chosen to represent the model. In the case of the CNN, the average train accuracy is 88.25%, while the STN-CNN has an average of 99.01%. This large discrepancy (∼12%) could be indicative of over-fitting on the part of the STN.

- Limited data. Given that we do not increase the size of the train set, the spatial transformer module is limited in the number of transformed samples it is exposed to. We believe that a method such as data augmentation, where transformed samples are added to the standard train set, can significantly improve its performance.

In summary, STN-CNNs do not provide any significant advantages in generalization, in comparison to a CNN, if trained on the standard or transformed CIFAR10 train set. However, we believe that the performance of the STN can be significantly improved if trained on augmented data, and if explicit regularization is used.

6.8 Conclusion

In this chapter we explored the transformation invariance of spatial transformer networks, convolutional neural networks, and multi-layer perceptrons. We analyzed and compared these architectures for translation, rotation, and scale invariance. Finally we studied the generalization capabilities of these models.

Our main findings are summarised below:

- STNs require exposure to diverse samples in order to learn corrective transformations. In the case of CIFAR10, the standard train set does not contain enough variability for spatial transformers to learn corrective translation or scale transformations (Figures 6.2, 6.10, and 6.12). However, both the standard MNIST and CIFAR10 train set is sufficient for a spatial transformer to learn a slight corrective rotation transformation (Figures 6.4 and 6.7).
- Transforming the train set prior to training can greatly improve the invariance of all the architectures considered (MLP, CNN, and STN). Furthermore, STNs perform better in terms of invariance than other architectures when the training set is transformed (Figures 6.3, 6.6, 6.11, and 6.13). We attribute this to the spatial transformer module learning a corrective transformation after being exposed to di-

verse samples. However, in the case of rotation invariance on CIFAR10 (Figure 6.9), the STN performs poorly when measured on greater rotations than the train set was subject to. We believe this can be rectified with a rotation protocol which more accurately simulates a real-world rotation.

- Spatial transformer modules can negatively affect transformation invariance if not exposed to diverse samples (Figures 6.2, 6.10, and 6.12). We believe that this is due to the module being improperly trained (as it had not been exposed to samples that are diverse enough), and subsequently applying transformations which are counter-productive.

- STNs do not provide any significant improvements in generalization compared to CNNs (Tables 6.3, 6.4, and 6.5). We believe that this can be rectified through the use of explicit regularization and data augmentation. Finally, we observe that some transformation of the train set always results in increased accuracy on the standard test set in the case of STNs.

Chapter 7

Conclusion

"I have made this longer than usual because I have not had time to make it shorter." - Blaise Pascal, in a letter, translated from French [64]

7.1 Introduction

In this chapter we first summarise our main findings, in Section 7.2. We then discuss how each research question posed earlier is addressed (Section 7.3), before stating the practical implications of our research (Section 7.4). Finally we list some avenues for future work, in Section 7.5.

7.2 Main findings

Throughout this study we have investigated many architectures, and drawn several conclusions surrounding transformation invariance and generalization. We now summarise our main findings:

- Subsampling and translation invariance:
  - Subsampling improves a CNNs invariance to translation, given that it is combined with sufficient local homogeneity.
  - Filtering, such as anti-aliasing or max pooling, improves local homogeneity. Given sufficient filtering, subsampling can greatly improve a model's invariance to translation.
  - The amount of filtering required is dependent on the inherent homogeneity of a given data set.
- Translation invariance and generalization:
  - Filter kernels that are too large (5x5 max pooling), or too much subsampling leads to a reduction in generalization ability.
  - As greater filtering and subsampling leads to better translation invariance, there is a slight trade-off between translation invariance and generalization.
  - Some mild subsampling and some mild filtering improves generalization.
- Learned translation invariance:
  - Translating the train set prior to training is very effective in increasing a model's translation invariance.
- Spatial transformer networks and translation invariance:
  - The standard CIFAR10 train set does not contain enough variability in spatial location for a spatial transformer module to learn a corrective transformation.
  - Translating the train set prior to training allows a spatial transformer module to learn a corrective transformation.
  - STNs outperform CNNs in terms of translation invariance if trained on translated data.
  - CNNs and STNs outperform MLPs in terms of translation invariance, when trained on a standard or translated train set.

- Spatial transformer networks and rotation invariance:
  - The standard MNIST and CIFAR10 train set does contain enough variability in the angle of its samples for a spatial transformer module to learn a corrective rotation. STNs outperform CNNs in terms of rotation invariance.
  - Train on rotated data, STNs perform better than other architectures, with the exception of the CIFAR10 data set. We attribute this to our rotation protocol, which isn't able to accurately simulate a realistic rotation.
  - STNs and CNNs outperform MLPs in terms of rotation invariance, and STNs generally outperform CNNs.
- Spatial transformer networks and scale invariance:
  - For up-scale invariance, the standard CIFAR10 train set does not contain sufficient variability to learn any corrective transformation, and STNs perform worse than MLPs and CNNs.
  - For down-scale invariance, the standard CIFAR10 train set contains some variability, and STNs perform similar to CNNs, and better than MLPs.
  - Scaling the train set prior to training allows STNs to perform better than CNNs and MLPs, in terms of both up-scale and down-scale invariance.
- Spatial transformer networks and generalization:
  - STNs and CNNs outperform MLPs in terms of generalization, on both a standard and transformed test set.
  - STNs do not provide any significant benefits in terms of generalization compared to a CNN. However, we believe that this can be rectified through the use of data augmentation and explicit regularization.

7.3 Addressing research questions

Given the key findings expressed in the previous section, we can now provide answers surrounding our initial research questions:

- Which architectural elements are responsible for a given CNN's translation invariance (or lack thereof)?
  - We find that kernel stride, and the size of pooling kernels in a CNN architecture have the greatest impact on the translation invariance of a model.
  - Larger kernels and stride improve a network's invariance to translation.
- How do methods proposed by other authors fair in ensuring translation invariance?
  - For Zhang's anti-aliasing methods, we found that combined with subsampling, anti-aliasing filters can significantly improve translation invariance. Furthermore we find that these filters have no significant effects on generalization ability.
  - We found that a network with no subsampling, and global average pooling, as proposed by Azulay and Weiss, is completely translation invariant. However, by using no subsampling the computational burden is much greater, and additionally we believe that the GAP operation can potentially hamper generalization ability.
- How does the transformation invariance of different deep learning architectures compare to one another?
  - If trained on a standard, untransformed train set, CNNs tend to perform better than STNs in terms of transformation invariance.
  - When trained on a transformed data set, STNs are generally more invariant to transformation than CNNs.
  - Both STNs and CNNs greatly outperform MLPs in terms of transformation invariance.

- How does transformation invariance and generalization relate to one another?
  - In terms of translation invariance, too much subsampling and filtering leads to a reduction in generalization, but also greater translation invariance. Similarly, too little subsampling and filtering negatively affects both generalization and translation invariance. Thus, for larger subsampling and kernel sizes, there is a trade-off between translation invariance and generalization.
  - For the specific spatial transformer networks studied, we find that they provide improved transformation invariance, but no significant benefits in terms of generalization.
  - STN-CNNs and normal CNNs have similar generalization, and both perform much better than MLPs.

7.4 Practical implications

In terms of practical implications, we highlight the advantages and disadvantages of different methods for transformation invariance. We hope that these findings can guide others in designing better models for a specific task, depending on the scope of the problem and the resources available, for example:

- If translation invariance is of great importance to the specific task, our research can guide the selection of pooling kernel size and kernel stride, depending on the inherent homogeneity of the data set. Additionally, we have verified the efficacy of Zhang's anti-aliasing filters [43], which could also be a suitable solution.
- If complete translation invariance is required, global average pooling and no subsampling (as proposed by Azulay and Weiss [9]) can be an adequate solution, given there is enough computational resources available.
- For better transformation invariance in general, we have shown that spatial transformer networks can be a suitable solution. However, we also show that these

networks require considerable transformation of the train set and could also require data augmentation and explicit regularization.

7.5 Future research

While we have conducted an in-depth study of transformation invariance, there are still several research avenues that require exploration. We first discuss additional research that is required to further verify our results, and then mention other areas that also require examination.

In terms of additional verification, we believe the following can further help our understanding of transformation invariance:

- To ensure that our results are not subject to any secondary effects, we had not made use of any explicit regularization throughout this study. We believe that repeating some of our key experiments with added regularization such as dropout or batch normalization can improve our understanding of transformation invariance.
- When comparing the transformation invariance of several models, we only considered the test set samples that were correctly classified prior to transformation by all models within the analysis. For further verification it would be beneficial to repeat the analysis on the remaining samples for the CIFAR10 models. (For MNIST the fraction is too small to have any effect.)
- We have made use of only two data sets, namely CIFAR10 and MNIST. To further verify our results, other, large image data sets must also be considered, such as ImageNet [39].
- While spatial transformer modules can be inserted as intermediary transformers in a network, we had only used them as initial input image transformers. We believe that measuring the effects of these modules when transforming intermediary feature maps can also be beneficial to our understanding of their efficacy.

In addition to further verification, there are also other avenues surrounding transformation invariance that require exploration:

- The intricacies of rotation and scale invariance in CNNs. We have thoroughly studied the translation invariance of CNNs, but have not done an in-depth analysis of which architectural elements contribute to rotation and scale invariance. Similarly, we have not measured the efficacy of other methods surrounding these transformations such as those mentioned in Section 2.6.
- Capsule Networks introduced by Hinton et al. [65]. These are unique architectures which are (supposedly) able to encode better representations of the spatial relationship between features in an image than a normal CNN, and can potentially be very beneficial for transformation invariance.
- Fully convolutional neural networks (FCNNs) - Fully convolutional networks do not make use of any fully connected layers. This implies that they could handle transformations of the input image differently than a normal CNN, and require exploration.

7.6 Generalized conclusion

Transformation invariance is a difficult problem in deep learning image classification systems. We have studied some solutions to this problem, and demonstrated the respective strengths and weaknesses of each method.

All of our results suggest that MLPs fair far worse in terms of transformation invariance and generalization than CNNs and STN-CNNs. For spatial transformer networks, we have shown that they require a significant amount of prior transformation to be effective. We have also shown that without explicit regularization or data augmentation they do not provide any significant benefits in terms of generalization.

In terms of translation invariance, subsampling is often “villainized” as it breaks the

equivariance property. However, we have introduced a novel perspective which shows that subsampling can be greatly beneficial towards translation invariance, given it is combined with sufficient local homogeneity.

We hope that our findings can guide others in designing better, more effective image classification systems, depending on the requirements and resources available.

Bibliography

[1] J. R. Ross and N. Chomsky, "Constraints on variables in syntax," PhD thesis, Massachusetts Institute of Technology, 1967, p. 4.

[2] D. H. Ballard and C. M. Brown, *Computer vision*. Prentice-hall, 1982.

[3] A. Lee, "Comparing deep neural networks and traditional vision algorithms in mobile robotics," 2016.

[4] Y. LeCun, C. Cortes, and C. Burges, *MNIST database of handwritten digits*, `http://yann.lecun.com/exdb/mnist/`.

[5] A. Krizhevsky, V. Nair, and G. Hinton, *CIFAR10 dataset*, `https://www.cs.toronto.edu/~kriz/cifar.html`.

[6] Y. Lecun, "Generalization and network design strategies," English (US), in *Connectionism in perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Elsevier, 1989.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 200, `http://www.deeplearningbook.org`.

[8] M. Alghali, A. Abdalazeem Ahmed, and T. Khalid, "Benchmark analysis of popular ImageNet classification deep CNN architectures," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, Aug. 2017, pp. 902–907. DOI: `10.1109/SmartTechCon.2017.8358502`.

[9] A. Azulay and Y. Weiss, "Why do deep convolutional networks generalize so poorly to small image transformations?" *CoRR*, vol. abs/1805.12177, 2018. arXiv: `1805.12177`.

[10] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks,” vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 2017–2025, 2015.

[11] 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, pp. 267–281, 2020.

[12] 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, pp. 282–295, 2020.

[13] G. B. Hinkley, *Lord, increase our faith*, Jun. 1983.

[14] J. Dean, “The deep learning revolution and its implications for computer architecture and chip design,” 2019. arXiv: `1911.05289 [cs.LG]`.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 164, `http://www.deeplearningbook.org`.

[16] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014, p. 229, ISBN: 1107057132.

[17] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, p. 227, ISBN: 0387310738.

[18] A. F. Agarap, “Deep learning using rectified linear units (relu),” *arXiv e-prints*, arXiv:1803.08375, arXiv:1803.08375, Mar. 2018. arXiv: `1803.08375 [cs.NE]`.

[19] Hecht-Nielsen, “Theory of the backpropagation neural network,” in *International 1989 Joint Conference on Neural Networks*, 1989, 593–605 vol.1. DOI: `10.1109/IJCNN.1989.118638`.

[20] *Introduction to Probability, Statistics and Random Processes*, 2020. [Online]. Available: `https://www.probabilitycourse.com/chapter9/9_1_5_mean_squared_error_MSE.php` (visited on 11/06/2020).

[21] Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels,” in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018, pp. 8778–8788.

[22] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, p. 233, ISBN: 0387310738.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 326, http://www.deeplearningbook.org.

[24] ——, *Deep Learning*. MIT Press, 2016, p. 327, http://www.deeplearningbook.org.

[25] J. Gonzalez-Barajas and D. Montenegro, “Digital image processing,” in. Apr. 2016, p. 136, ISBN: 978-1-63485-210-4.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 139, 145, 167–170, 192–194, http://www.deeplearningbook.org.

[27] ——, *Deep Learning*. MIT Press, 2016, p. 330, http://www.deeplearningbook.org.

[28] ——, *Deep Learning*. MIT Press, 2016, p. 333, http://www.deeplearningbook.org.

[29] ——, *Deep Learning*. MIT Press, 2016, p. 334, http://www.deeplearningbook.org.

[30] S. Sumit, *A comprehensive guide to convolutional neural networks*, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 11/06/2020).

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 342, http://www.deeplearningbook.org.

[32] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: An overview and application in radiology,” *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, Aug. 2018, ISSN: 1869-4101. DOI: 10.1007/s13244-018-0639-9.

[33] O. Semih Kayhan and J. C. van Gemert, "On Translation Invariance in CNNs: Convolutional Layers Can Exploit Absolute Spatial Location," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 262–14 273. DOI: 10.1109/CVPR42600.2020.01428.

[34] D. Scherer, A. C. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *ICANN*, 2010.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 337, http://www.deeplearningbook.org.

[36] O. Byer, F. Lazebnik, and D. Smeltzer, *Methods for Euclidean Geometry*, ser. Classroom Resource Materials. Mathematical Association of America, 2010, p. 260, ISBN: 9780883857632.

[37] K. Zakka, *Deep learning paper implementations: Spatial transformer networks - part ii*, 2020. [Online]. Available: https://kevinzakka.github.io/2017/01/18/stn-part2/ (visited on 11/12/2020).

[38] G. R. Arce, J. Bacca, and J. L. Paredes, "Chapter 12 - nonlinear filtering for image analysis and enhancement," in *The Essential Guide to Image Processing*, A. Bovik, Ed., Boston: Academic Press, 2009, pp. 263–291, ISBN: 978-0-12-374457-9. DOI: https://doi.org/10.1016/B978-0-12-374457-9.00012-3.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[40] A. Khosla, T. Zhou, T. Malisiewicz, A. A. Efros, and A. Torralba, "Undoing the damage of dataset bias," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 158–171, ISBN: 978-3-642-33718-5.

[41] N. Noord and E. Postma, "Learning scale-variant and scale-invariant features for deep image classification," *Pattern Recognition*, vol. 61, Feb. 2016. DOI: 10.1016/j.patcog.2016.06.005.

[42] P. Follmann and T. Bottger, "A rotationally-invariant convolution module by feature map back-rotation," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 784–792. DOI: 10.1109/WACV.2018.00091.

[43] R. Zhang, "Making convolutional networks shift-invariant again," *CoRR*, vol. abs/1904.11486, 2019. arXiv: 1904.11486.

[44] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," *International Journal of Computer Vision*, vol. abs/1411.5908, pp. 456–476, 2014. arXiv: 1411.5908.

[45] Y. Xu, T. Xiao, J. Zhang, K. Yang, and Z. Zhang, "Scale-invariant convolutional neural networks," *CVPR2015*, vol. abs/1411.6369, 2014. arXiv: 1411.6369.

[46] E. Kauderer-Abrams, "Quantifying translation-invariance in convolutional neural networks," *CoRR*, vol. abs/1801.01450, 2018. arXiv: 1801.01450.

[47] R. P. Feynman, *The scientific method*, 1964.

[48] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 818–833, ISBN: 978-3-319-10590-1.

[49] W. Heymans, *Generalisation of single hidden-layer feedforward neural networks*, 2020.

[50] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[51] Heraclitus and Plato, *Doctrine of Flux*. 500BCE.

[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 328, http://www.deeplearningbook.org.

[53] K. He and J. Sun, "Convolutional neural networks at constrained time cost," Jun. 2015, pp. 5353–5360. DOI: 10.1109/CVPR.2015.7299173.

[54] C. Hitchens, *Oxford Essential Quotations*, 4th ed. Oxford University Press, 2016, ISBN: 9780191826719.

[55] S. K. Mitra, *Digital Signal Processing*. McGraw-Hill Science/Engineering/Math, 2005, p. 335, ISBN: 0073048372.

[56] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 2nd. USA: Prentice Hall Press, 1999, p. 185, ISBN: 0137549202.

[57] J. Gonzalez-Barajas and D. Montenegro, "Average filtering: Theory, design and implementation," in. Apr. 2016, ISBN: 978-1-63485-210-4.

[58] M. Aubury and W. Luk, "Binomial filters," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 12, no. 1, pp. 35–50, 1996. DOI: 10.1007/bf00936945.

[59] M. Lin, Q. Chen, and S. Yan, *Network in network*, 2014. arXiv: 1312.4400 [cs.NE].

[60] 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, Dec. 2010, ISSN: 0899-7667, 1530-888X. DOI: 10.1162/NECO_a_00052.

[61] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, pp. 1–48, 2019.

[62] *Roll out*. [Online]. Available: https://tfwiki.net/wiki/Roll_out (visited on 12/03/2020).

[63] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," 2016, Published in ICLR 2017.

[64] B. Pascal, *Lettres Provinciales*, 1657.

[65] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3859–3869, ISBN: 9781510860964.