Frequency-Domain Deconvolution in Deep Learning

Presented by:
Rashaad Meyer
MYRMOE002

Prepared for:
Fred Nicolls
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Master of Science degree in Electrical Engineering.

August 12, 2024

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....
Rashaad Meyer

Date: 10/02/2023

Acknowledgements

The completion of this project could not be possible without the support and guidance of many people. I sincerely thank the following people who have helped me along this amazing 2-year journey.

First off, I would like to thank my supervisor, Professor Fred Nicolls. His support and guidance throughout this journey have been invaluable. His flexibility allowed me to truly make this research my own, and I learnt so much along the way. This wouldn't have been possible without him.

I would like to thank my parents for their support throughout my life. I'm grateful that they have given me an opportunity to follow my passion.

Abstract

This dissertation presents an exhaustive exploration of a novel approach to deep learning in computer vision tasks: the frequency-domain deconvolution operation. Recognizing the unparalleled success of convolutional neural networks (CNNs) in the realm of computer vision, we critically evaluate the performance and computational demands of traditional convolution operations against the proposed deconvolution method. Using a systematic approach, we apply the deconvolution layer to two quintessential computer vision problems: image classification and single image super resolution (SISR). The results demonstrate the deconvolution layer's potential in certain scenarios, with marked improvements observed in image classification. For SISR tasks, though advantages were noticed under specific configurations, the traditional CNNs still demonstrated their robustness. Additionally, the dissertation touches upon the layer's computational demands, revealing an increased computational overhead for the deconvolution layer. Encouragingly, the layer demonstrated promising attributes like learning long-range filters and isolating objects from backgrounds effectively. Concluding with avenues for future research, this dissertation acts as a stepping stone in the uncharted territory of deconvolution operations, emphasising innovation alongside evaluation in the dynamic world of deep learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning has experienced a monumental increase in public interest in the last few years. This can be attributed to the widespread adoption of generative AI and the transformer architecture. While the popularity has surged particularly in the Natural Language Processing domain, computer vision has seen less of this transformative shift. Convolutional neural networks (CNNs) have been a standard approach for computer vision problems such as image classification, object detection, image super resolution, and image deblurring. Recently, many computer vision researchers have worked on fine-tuning and developing new complex CNN architectures. This dissertation seeks to deviate from the status quo and investigate an alternative operation, different from the convolution operation, as the backbone of a neural network for computer vision tasks. In this dissertation, we propose a frequency-domain deconvolution operation, which can be seen as a convolution with an inverse filter.

1.1 Motivation and Background

The field of computer vision has undergone significant transformations since its formation. At the heart of computer vision lies the CNN, a type of neural network architecture synonymous with digital image processing and computer vision. Over the last few decades CNNs have evolved, with each iteration aiming to push better-performing networks in both accuracy and speed. This push has led to groundbreaking discoveries that have shaped deep learning and computer vision today.

CNNs are used everywhere in computer vision and have enabled advancements in areas

ranging from image classification to object detection. The success that CNNs have brought to computer vision is undeniable. However, this has come at a cost. The pursuit of accuracy and better performance has often led to increasingly complex architectures that require more computational resources, often resulting in diminishing returns.

One major problem for CNNs is learning long-range spatial dependencies. For a CNN to learn such relationships in the data, it needs multiple convolutional layers to be cascaded or the size of the kernel to be increased. By adding these additional layers and increasing the filter size, it increases the number of trainable parameters of the network. Simply increasing the number of trainable parameters is an inefficient means of learning long range dependencies. We can relate this to the statistical principle of parsimony which suggests that a model should be as complex as necessary but as simple as possible. Taking this principle into account, we need to find an alternative that can learn long-range dependencies without increasing the trainable parameters. Furthermore, to learn long range correlations we will need filters with wide impulse responses and these filters should not come at the expense of an increase in the number of trainable parameters. If long range dependencies are important then filters with a wider impulse response should increase performance when added to standard CNN architectures.

The deconvolution layer aims to increase the width of the impulse response without increasing the trainable parameters. It uses a deconvolution process as its main operation as opposed to a convolution operation. Where a convolution operation can be viewed as applying an all zero filter to the input, a deconvolution operation inverts the filter and applies an all pole filter to the input which produces a wider impulse response. This wider impulse response allows it to capture a wider range of data compared to the standard convolutional layer. This allows it to learn long-range dependencies early on in the network whereas CNNs require multiple convolutional layers to be cascaded to get the same effect.

Deconvolution can be viewed as an inversion process used to reverse the effects of a convolution operation. This inversion is often not possible since the deconvolution operation requires all frequency components in the transfer function of the filter to be non-zero. A better way to view deconvolution is as a convolution operation with an inverse filter. This inversion results in a wider impulse response which allows a layer with the deconvolution operation to learn long range dependencies in the data. An added benefit is that the deconvolution layer's filter is not derived directly from its trainable parameters. This provides the deconvolution layer with greater freedom, as it is not restricted by the natural structure of a convolution filter.

On top of the deconvolution layer, we implemented different design strategies to potentially increase the performance of the layer. We test these strategies empirically to see whether they increase the performance of the layer. We present and test the following design strategies: four-factor deconvolution, first element trainable, filter bias, and padding. All of these design strategies aim to solve a problem with the deconvolution operation or convolution in the frequency domain. For instance, the four-factor deconvolution forces the deconvolution layer to apply a symmetric filter to the input which keeps spatial coherence between the input and output. The first element trainable ensures that the deconvolution layer starts in an optimal position on the loss curve.

To test our hypotheses of the deconvolution layer and the design strategies, we conducted experiments by using the layer in various networks trained on single image super resolution (SISR) and image classification. SISR was chosen due to its downscaling operation being similar to a blur operation. Since a blur operation is a convolution operation and the deconvolution operation can reverse the effects of convolution, it is reasonable to propose that the structure of the deconvolution operation and layer will be more suitable for upscaling process of SISR. Additionally, we choose image classification, because it is the most popular computer vision task and is often used to benchmark CNNs. The image classification problem will help us gauge whether the deconvolution layer's ability to learn long-range dependencies without an increase in trainable parameters is effective on more complex computer vision tasks. As networks that typically do well on image classification also do well in more complex computer vision tasks such as object detection and image segmentation. These experiments on SISR and image classification will help us evaluate the deconvolution layer's effectiveness and potential on the broader scope of computer vision tasks.

The idea of challenging conventions isn't new. Multiple scientific fields have witnessed transformations when traditional methods are replaced or combined with novel approaches. The success stories from these different fields serve as a testament to the potential rewards of venturing into uncharted territories and have been a motivating factor behind our research in this dissertation.

1.2 Problem Definition

Computer vision has seen rapid advancements in the last decade but has predominantly relied on CNNs as its primary tool to process images. These networks use convolution as

a foundational building block, enabling them to extract and learn features from images. To improve the accuracy of CNNs, researchers have both increased the complexity of CNNs and made deeper networks.

However, this increasing complexity presents several issues:

- **Computational Overhead:** As CNN architectures become more complex, they require more computational resources, leading to longer training times while facing challenges during deployment on resource-constrained devices.
- **Diminishing Returns:** The continuous improvement and layering of CNNs could reach a point where large increases in complexity and depth only yield a small performance gain.
- **Over-reliance on a Single Operation:** The convolution operation's success is undeniable but it might not be the most appropriate solution for every computer vision task. Relying on this operation could limit potential avenues of exploration in the field.

Given these challenges, this research has the following objectives:

- **Deconvolution Architecture:** Design and detail the architecture of the deconvolution layer while highlighting different design strategies and potential issues.
- **Performance on SISR:** Test and evaluate the deconvolution layer on an SISR task to see if it is a viable replacement for the convolutional layer in SISR.
- **Performance on Image Classification:** Test and evaluate the deconvolution layer on an image classification task to see if it is a viable replacement for the convolutional layer in image classification.
- **Layer Response:** Observe and discuss the response of the layer to both an impulse and image input.
- **Computational Demand:** Evaluate the deconvolution layer's computational resource requirements.

All these objectives tie together to answer the following question:

"Is it possible to develop an alternative operation to convolution, specifically a frequency domain deconvolution operation, that can be used as a backbone inside a neural network for computer vision problems? And if so, how does this new approach compare in terms of efficiency and accuracy to traditional CNNs?"

By answering this question, we aim to broaden the scope of the computer vision methodologies, potentially offering a new tool that can complement or even outperform existing CNN-based approaches in specific scenarios.

1.3 Scope and Limitations

The work presented in this dissertation is limited to developing a neural network layer that uses a deconvolution operation. This dissertation also investigates the performance of this layer on two different computer vision tasks: image classification and single image super resolution (SISR). There is a large amount of literature on both of these tasks; however, we limit the scope of the research to relatively small networks as the deconvolution layer's impact will be seen more prevalently in these networks.

The deconvolution layer was designed and developed in 2022, and the experiments on the layer were conducted in 2023 alongside the writing of this dissertation.

1.4 Contributions

This dissertation provides the following contributions:

Frequency-based Deconvolution Neural Network Layer: The design and implementation of an alternative neural network layer. This layer is a frequency-based deconvolution layer that utilises novel concepts and techniques.

Comparative Analysis with Traditional CNNs: We evaluate the proposed deconvolution layer by conducting experiments and comparing it to traditional CNNs. The experimental results provide insights into the efficiency and accuracy of the deconvolution operation compared to the convolution operation.

Application to Real-world Computer Vision Tasks: The deconvolution layer was

tested on two popular computer vision tasks: image classification and single image super resolution (SISR). The results from these experiments provide a practical understanding of the potential benefits and limitations of the proposed layer.

Open-source Implementation: We have implemented the deconvolution layer and the source code for the work in this dissertation is publicly available to promote transparency and encourage further research. This allows the research community to build upon or challenge the findings in this dissertation.

Potential Pathways for Future Research: Finally, we outline potential areas of exploration and improvement for the deconvolution-based approach based on the findings. These suggestions aim to guide future researchers in advancing the state-of-the-art in this novel domain.

1.5 Outline

This section outlines the structure of the rest of this dissertation.

Chapter 2 starts by providing a background around computer vision domains that are relevant to the work done in this project. It initially provides an overview of the convolution operation by stating the mathematical definition for the one-dimensional and two-dimensional inputs. The section also provides some intuition behind the operation while delving deeper into topics like the convolution theorem. Next, the chapter expands on the previous section by providing a context for deep learning and computer vision, known as convolutional neural networks. Next, the chapter briefly describes deconvolution and gives two different types of deconvolution. Subsequently, the chapter highlights some image scaling techniques. Lastly, the chapter gives an overview of PyTorch, which is the machine learning framework used to develop and test the deconvolution layer.

Chapter 3 provides an overview of the previous work that is related to the research in this dissertation. Initially, it explores the previous work done on SISR. It details research conducted prior to the advent of deep learning, then delves into work done on SISR using deep learning algorithms. Next, the chapter provides an overview of image classification. Furthermore, the chapter provides on overview on the work done prior to deep learning and then examines research for image classification using deep learning methods. Lastly, the chapter discusses image deconvolution where it presents work done both prior and

after the advent of deep learning. This section highlights that typical deep learning image deconvolution does not use deconvolution operations but rather convolution operations.

Chapter 4 gives design details of the deconvolution layer. It starts with a motivation for why the deconvolution layer will be beneficial over a convolutional layer. Next, it details the deconvolution operation implementation in the frequency domain. Subsequently, it provides the different design strategies that have been implemented on top of the deconvolution layer. After, the chapter moves onto a one-dimensional application of the deconvolution layer followed by a high-level overview of the implementation. Finally, the chapter ends with a section presenting an overview of the experiments conducted in this dissertation.

Chapter 5 begins with an introduction to single image super resolution (SISR). Next, it details the algorithms and networks tested in the experiments. Subsequently, it gives an outline of all the experiments. Next, it provides the details of the datasets that are used to train and validate the network. Finally, it presents the experiments' results while providing a discussion. The discussion includes a hypothesis and an analysis for each experiment.

Chapter 6 is similar to the previous chapter, except it deals with image classification. The section begins with an introduction then details the algorithms and networks tested in experiments. After, it provides the details of the dataset used to train and validate the network. Subsequently, it gives an outline of all the experiments. Finally, it presents the experiments' results while providing a discussion. The discussion includes a hypothesis and an analysis for each experiment.

Chapter 7 concludes by summarising the findings from this dissertation and Chapter 8 provides recommendations for future work in this domain.

Chapter 2

Background

This chapter provides a brief introduction into the convolution operation and convolutional neural networks, along with introducing the deconvolution operation and some image scaling techniques.

Section 2.1 provides an introduction to the convolution operation. It starts by providing the mathematical definition for both the 1D and 2D case of the convolution operation. Next, it presents intuitions behind the convolution operation for better understanding. Finally, it addresses convolution using Fast Fourier Transforms (FFT) while providing challenges and edge cases that arise when using this method. This section is especially important to this dissertation since the work done is largely based on a FFT convolution operation.

Section 2.2 introduces the concept of convolutional neural networks (CNNs). It explains the architecture of a CNN while highlighting the different components that appear in CNNs. Subsequently, it elaborates on how CNNs are trained while providing applications in both image recognition and image reconstruction. This section is relevant to the work in this dissertation because the Deconvolution Neural Networks (DeNN) are quite similar to each other while also having the same training process.

Section 2.3 presents a background on the deconvolution operation. It begins by introducing the concept of a deconvolution as the inverse of the convolution. Next, it provides two different types of deconvolution. Finally, it motivates the use of a deconvolution operation in deep learning. This section is vital since it introduces the operation that this dissertation revolves around.

Section 2.4 provides an introduction for image scaling techniques. It mainly touches on different interpolation techniques such as nearest-neighbour, bilinear, and bicubic. This is relevant to this paper since some work is done on super resolution which involves image scaling.

Section 2.5 introduces PyTorch, an open-source machine learning library. This is the framework that is used to build the deconvolution layer. This section explains the concept of dynamic computational graphs, tensors, and the `nn` module, all of which are vital components of PyTorch. It ends by providing some context on the libraries built around PyTorch and how it integrates with other popular tools.

2.1 Introduction to the Convolution Operation

Convolution is a fundamental operation in various scientific and engineering disciplines, especially in signal processing and image analysis. It offers a method to combine two functions, producing a third function that reflects how one shape is modified by another. This operation can be visualised as "sliding" one function over another and calculating the area of the product at each point.

The nature of this overlap and the way one function affects the other is central to convolution's effectiveness in various applications. For instance, in signal processing, convolution can be used to describe how an input signal transforms as it passes through a linear system.

Furthermore, convolution is closely related to cross-correlation, another important operation that measures the similarity between two functions. While both operations involve a sliding mechanism, they differ in the specifics of the function because the convolution reflects the shifted function.

In the sections to follow, we delve into the mathematical definitions of the convolution operation, exploring intuitions behind it and extending it to the frequency domain.

2.1.1 Mathematical Definition

Convolutions are primarily used in signal and image processing because they can transform one function using another. The operation can be visualised as sliding one function over the other and computing the area under the product for each shift value. Here, we will delve into the operation's mathematical representation for 1D and 2D cases.

Mathematical Representation of 1D Convolution

For two continuous time- or space-domain functions $f(t)$ and $g(t)$, the convolution is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} g(\tau) f(t - \tau)\, d\tau. \tag{2.1}$$

For discrete time- or space-domain functions $f[n]$ and $g[n]$ the formula becomes

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} g[m] f[n - m]. \tag{2.2}$$

In these equations, f is often referred to as the input while g is often referred to as the filter or kernel. The operation computes the sum of products for every possible overlap (defined by shifting g by t or n).

2.1.2 Mathematical Representation of 2D Convolution

The operation is easily extended to the 2D case, which applies to processing images. For two 2D functions $f(x, y)$ and $g(x, y)$ the continuous convolution is defined as

$$(f * g)(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(u, v) g(x - u, y - v) du dv, \tag{2.3}$$

and for two discrete 2D functions

$$(f * g)[x, y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m, n] g[x - m, y - n]. \tag{2.4}$$

2.1.3 Intuition behind the Mathematical Formula

A convolution can be viewed as a dot product between two functions at each shift value. Intuitively, this emphasises how the convolution captures the similarity between two functions at each shift value.

For 1D signals, the operation can be visualised as the reflected filter, $g(-t)$, sliding over the input signal, $f(t)$. The convolution is then computed by multiplying and summing the overlapping areas of both functions. The summation of these products gives the output at each point in the function.

For 2D inputs, such as images, the operation can be visualised as a small window, the filter, sliding over an image. At each position, the convolution is computed by multiplying the overlapping pixels and the filter values and summing these products. For image processing, using this operation can capture features, such as edges, from images.

2.1.4 The Fast Fourier Transform (FFT) and Convolution

Fourier analysis studies how general functions can be expressed as a sum of simpler trigonometric functions—the decomposition process of converting tasks into a sum of trigonometric functions at different frequencies [1]. The Discrete Fourier Transform (DFT) applies the Fourier Transform to a discrete sequence of values [2]. It is obtained by breaking down a function into its different frequency components [3]. Unfortunately, the DFT computed directly from the definition is too slow to be practically useful in many fields. However, a Fast Fourier Transform (FFT) can compute the DFT of a sequence by factorising the DFT matrix into a product of sparse factors [4]. The FFT manages to decrease the computation time of the DFT from $O(N^2)$ to $O(N \log N)$ where N is the size of the data being transformed. The following formula defines the DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j \cdot (2\pi k n / N)}. \tag{2.5}$$

The Convolution Theorem

The convolution theorem is one of the most significant breakthroughs in signal processing [5]. The theorem states that the convolution of two different functions in

the time domain equals the multiplication of the Fourier Transforms of these functions in the frequency domain. The theorem is defined by

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g). \tag{2.6}$$

$\mathcal{F}(f)$ denotes the Fourier Transform of the function f, and $f * g$ is the convolution of f and g.

2.1.5 Edge cases and Challenges

A common issue with the convolution is handling the borders of images and signals. When the filter moves across the input, it needs a strategy when it reaches the edges. Three different strategies that handle border effects are:

- **Valid:** No padding is introduced, and the convolution output is smaller than the input.
- **Same:** In this strategy, the borders are padded with zeroes so that the size of the input image is preserved.
- **Full:** Here, the padding is applied so that every element from the input is convolved with every filter element. This results in the output size being larger than the input size.

Padding the input not only helps control the output size but also helps reduce artefacts around the borders. Padding the input signal provides a buffer for the filter to pass over and results in a smoother edge around the border of the output image.

2.2 Convolutional Neural Networks

In deep learning, a convolutional neural Network (CNN) is a neural network that contains at least one convolutional layer [6, 7]. These networks are generally used for computer vision tasks [8]. Convolutional layers use mathematical operations called convolutions instead of matrix multiplication, which is used by fully connected or dense layers [6]. Convolutional layers preserve the structural information of images while transforming them into feature maps that capture hierarchically organised patterns. Convolutional

neural networks are used in various applications such as image and video recognition, recommender systems [9], image classification [10], object detection and image super resolution.

2.2.1 Architecture of a CNN

The input data of a convolutional layer is often comprised of feature maps. This could be both raw image data or outputs from preceding layers. The inputs and outputs for convolutional networks are four-dimensional tensors and have the following structure:

- The first dimension is the batch,
- the second dimension is for the image channels/feature maps, and
- the last two are for the image's height and width.

This is known as the NCHW format, which is the format that is used in PyTorch [11]. Alternatively, TensorFlow uses the NHWC, which means the last dimension is for the channel dimension [12]. TensorFlow does, however, have functionality that allows it to use the NCHW format.

Once the image data or feature maps are passed into the convolutional layers, the layer applies a two-dimensional convolution operation to the input. The convolutional layer slides a small filter over the input data and computes the dot product of each patch of the input data with the filter. For image data, the filter moves from left to right and then from top to bottom, computing the dot product at each position it covers. The convolutional layer does this for multiple independent filters. The result is referred to as the feature map. Figure 2.2.1 illustrates how this operation is performed for each filter.

Feature maps represent patterns that the filters have detected in the input data. These feature maps get more complex as we go deeper into the CNN. In the first layers, the CNN can detect edges, and in deeper layers it can detect more complex compositions such as shapes and objects.

In the literature, one often finds references to strides and padding. "Strides" refers to the number of pixels a filter moves over the input. With a stride of one, the filter will move one pixel at a time when computing the dot product, while a stride of two means that the filter jumps two pixels and computes the dot product. Padding can control

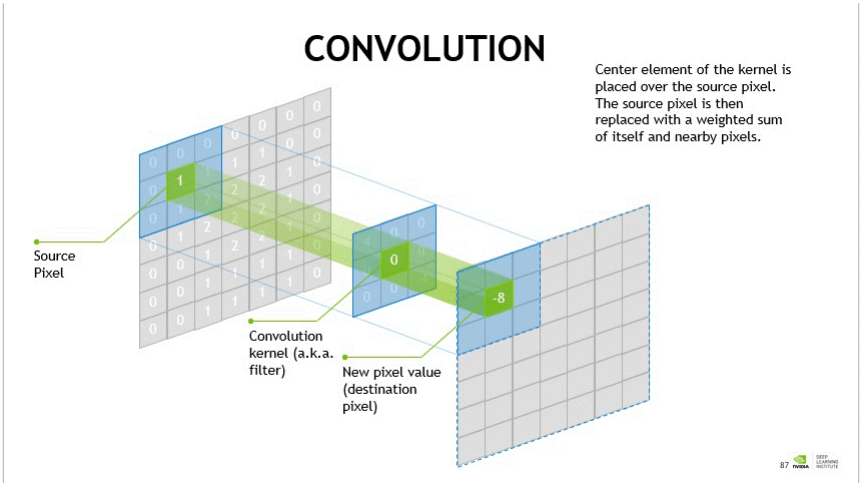

Figure 2.1: Illustration of how convolutional filter slides over an image data [13].

the spatial size of the output feature map. This is vital since applying a convolution operation reduces the size of the data that is passed in. This situation can be avoided by padding the edges of the input data with zeroes. The two most common padding methods are "valid" and "same". "Valid" padding means that no padding is used, and "same" padding means that the input image is padded so that the output has the exact spatial size as the input. Figure 2.2 shows a diagram of a typical CNN architecture.

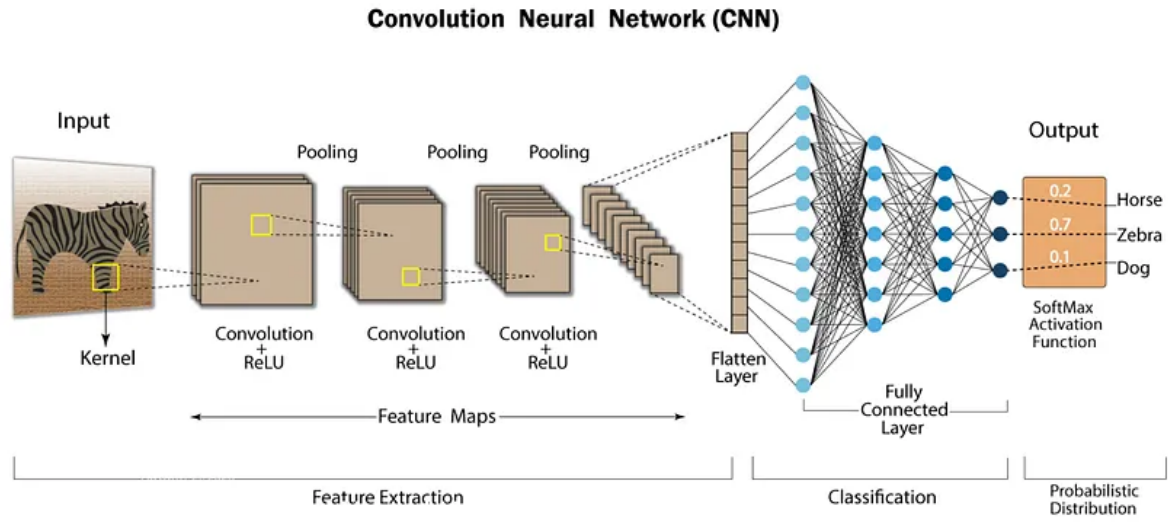

Figure 2.2: High-level diagram of a typical CNN architecture for image classification [14].

Convolutions are the core component of convolutional neural networks; however, they do not make up the entirety of these networks. Here are the fundamental components of CNNs:

- **Activation function**: After the convolution operation, it is common to introduce non-linearities by adding subsequent activation functions. One of the most

common activation functions is the Rectified Linear Unit (ReLU). This is a simple function that clamps all negative values to zero. The sigmoid and softmax functions are generally used at the network's end for binary and multi-class classification, respectively. These functions are used because they transform logits into probabilities.

- **Pooling layer**: Pooling layers help reduce the input data's spatial dimensions (height and width). Reducing these dimensions increases the computational efficiency of CNNs and reduces overfitting. The two most used types of pooling are max pooling and average pooling. These pooling layers do a similar operation to a convolution operation in that the computation involves a sliding operation. Max pooling selects the maximum value from a group of values, and average pooling calculates the average of the group of values. Although pooling increases computational efficiency, there is some loss of information.

- **Fully connected layer**: Fully connected layers will often be found at the end of a CNN. These layers consist of neural nodes that are all connected. These layers will be found in most CNNs, and combining them with the CNN architecture has been shown to be highly effective in vision classification tasks and object detection.

2.2.2 Training CNNs

The training process of a CNN is the same as for any other neural network. Training data gets passed into the neural network, which computes the network's forward pass. The forward pass is calculated by applying operations for each layer in the network sequentially. Once the forward pass of the network has been computed on the data, the error, referred to as the network loss, is computed. This loss is backpropagated back through the network to calculate the gradients of the loss with respect to each of the parameters inside the network. These gradients measure how the loss changes with respect to changes in the parameters. The parameters are updated in the descent direction of the gradients. This is called an optimisation step, and the network loss decreases after every optimisation step.

2.2.3 Difference between CNNs and DeNNs

As mentioned previously, a deconvolution applies a convolution with an inverse filter as opposed to the convolution which just convolves the filter itself with the input. This gives

the deconvolution layer the ability to learn long range correlation which could increase the network it is placed in.

The deconvolution layer has been implemented in PyTorch, which is introduced in section 2.5. With the deconvolution layer's implementation, we try to keep its signature as close to the convolutional layer found in PyTorch [11] which allows it to be a drop-in replacement for a convolutional layer. This means the deconvolution layer applies a 2D over a signal composed of several channels. The layer consumes a 4D tensor as the input where the dimensions correspond to (N, C_{in}, H, W). The layer then produces an output with the following dimensions (N, C_{out}, H, W). Note that in this case N is a batch size, C_{in} denotes the number of channels in the input, C_{out} denotes the number of channels in the output, H is the height of input planes in pixels, and W is the width in pixels. The convolutional layer takes the input dimensions and produces the same output dimensions. Thus giving the deconvolution layer the same signature as the convolutional layer. With regards to the training process of a DeNN, it is trained using backpropagation the same as a CNN.

2.2.4 Image recognition Applications

One of the most common applications of CNNs is image recognition. This application involves different tasks related to understanding and interpreting image data. This includes tasks such as image classification, object detection, object tracking, and image segmentation. CNNs are extremely good at classifying images [15]. An example of their high performance can be seen in a dissertation from 2012 [16], where Ciresan *et al.* trained a CNN that can achieve an accuracy of 99.23% on the MNIST dataset [17].

One facet of image recognition is facial recognition. CNNs do not yield such impressive results compared to digit recognition for the MNIST dataset stated above [18]. In [19], Matusugu *et al.* trained a CNN on 5,600 still images of more than ten different people. The CNN achieved 97.6% accuracy on this data.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a benchmark in object detection and image classification. The dataset used in this challenge is the ImageNet dataset, which consists of over 14 million images [20] [21] which have been labelled, and at least one million of the images have bounding boxes provided for them. In ILSVRC 2014 [22], a large-scale image recognition challenge, most top teams used a CNN as their basic framework. The winner of this challenge was GoogleNet [23], which

achieved a mean average precision of 0.439 and a classification error of 0.0067%.

2.2.5 Image Reconstruction Applications

Another critical application of CNNs is image reconstruction. Image reconstruction refers to reconstructing high-quality images from low-resolution, corrupted, or incomplete images. This process is essential for domains where high-quality images help with analysis, but circumstances prevent high-quality images from being captured. This could be due to equipment limitations, noise, or other factors. CNNs have become popular for image reconstruction due to their ability to learn hierarchical features of the data given to them and their success in image recognition tasks.

Super Resolution

One of the most common use cases is super resolution (SR). SR aims to increase both the resolution and quality of low-resolution images. Early approaches use hand-crafted features and optimisation techniques. However, with the rise of deep learning, models such as SRCNN [24] achieved better performance when compared to the earlier methods. These networks are trained on low- and high-resolution image pairs, allowing them to learn mappings from the low- to high-resolution images. Further advancements in super resolution include the use of Generative Adversarial Networks (GANs) to produce higher quality upscaled images with even more detail, as seen in the super resolution GAN (SRGAN) [25].

Denoising and deblurring

Two other applications for CNNs are denoising images, where the aim is to remove noise from images, and deblurring images, which include sharpening images that have been blurred. Advancements in denoising saw researchers using convolutional autoencoders and developing models such as the DnCNN [26]. These models show significant improvements over previous methods for removing different types of noise, such as Gaussian, salt-and-pepper, and speckle noise. Similarly to super resolution, advancements in deblurring include the use of GANs. Networks like the DeblurGAN [27] use GANs to sharpen images.

Inpainting

Image inpainting refers to filling in missing or corrupted parts of an image so that it is consistent with the surrounding area and the rest of the picture. CNN-based methods have shown impressive results in image inpainting. They can generate plausible context-aware fillings for the missing or corrupted parts of the image. One notable model is DeepFill [28]. This model uses a feed-forward, fully convolutional neural network which can process images with multiple missing parts at arbitrary locations.

Medical Image Reconstruction

In medical imaging, high-quality images are essential for accurate diagnosis. CNNs have been applied in this area by using them for MRI reconstruction. CNNs are used to reduce the amount of time it takes to scan patients by generating high-resolution images from fewer measurements. Techniques such as AUTOMAP [29] transformed this space by learning end-to-end mapping from the raw sensor data.

In conclusion, the effectiveness of CNNs on image reconstruction is indisputable. These networks can learn complex mappings for image data and can adapt to a wide range of problems.

2.2.6 Other applications

The use of CNNs in Natural Language Processing (NLP) has been explored. CNNs have shown to have excellent performance in NLP tasks such as semantic parsing [30], search query retrieval [31], sentence modelling [32], classification [33], prediction [34], and other traditional NLP tasks [35]. Unlike recurrent neural networks (RNNs), CNNs can learn the diverse contextual interpretation of language without relying on the sequential order of the data. RNNs are more suited for sequence modelling tasks [36] [37] [38] [39].

CNNs are also used for anomaly detection. These CNNs are based on one-dimensional convolutions and use an unsupervised model to detect anomalies in the time domain [40].

2.3 The Deconvolution Operation

Deconvolution is the inverse operation of convolution. The deconvolution operation aims to reverse the convolution operation on a signal or image. This operation is used to recover the original signal that was inputted into the convolution operation and can be done with a certain level of accuracy [41]. The foundation for deconvolution was primarily established by Norbert Wiener, from the Massachusetts Institute of Technology, in his book Extrapolation, Interpolation, and Smoothing of Stationary Time Series [42].

2.3.1 Description

The deconvolution operation aims to recover the original signal, f, from a convolution equation of the form

$$f * g = h. \tag{2.7}$$

In most cases, h would be the recorded signal and g is the filter that convolves or distorts f. If the filter, g, is known or we know the form of it, we can perform a deterministic deconvolution. However, if g is unknown, then it needs to be estimated. This can be done by using statistical estimation or by building a model of the system that is used to transform f into h. Different methods of deconvolution can be chosen depending on the deconvolution parameters and measurement error. It is important to note that all frequency components of g need to be non-zero, otherwise a deconvolution is not possible.

Raw Deconvolution

Raw deconvolution is used when the measurement error is minimal. In this case, the deconvolution operation collapses into inverse filtering. Referring to equation 2.7, the original signal f can be recovered by Fourier transforming both the filter g and the recorded signal h into H and G, respectively. Using the convolution theorem, we find that

$$F = H(\omega)/G(\omega), \text{ if } G(\omega) \neq 0 \ \forall \ \omega, \tag{2.8}$$

where F is the Fourier transform of f. The inverse Fourier Transform of $\mathcal{F}$ is computed to obtain the estimated signal f, and ω is frequency.

Deconvolution with Noise

When dealing with physical methods, the situation is more accurately represented by the following equation

$$(f * g) + \epsilon = h. \tag{2.9}$$

where ϵ is the noise that was measured along with the signal. If the recorded signal is assumed to be noiseless the statistical estimate of g will be incorrect, and therefore the estimate of f will also be incorrect. The lower the signal-to-noise ratio, the more inaccurate the estimated deconvolved signal will be. This is why raw deconvolution is not a good solution for physical measurements. However, if the type of noise in the data is known, f can be estimated more accurately by using techniques such as Wiener deconvolution.

2.3.2 Our Approach: Deconvolution in Deep Learning

It is important to note that the convolution is generally non-invertible. For a convolution to be invertible, all frequency components of the convolution filter must be non-zero. Therefore, our aim with the deconvolution layer is not to invert a convolution operation but instead to have a convolution with a filter that provides a wider impulse response without an increase in the number of free parameters. With a wider impulse response, it gives the layer the ability to learn long-range relationships in the data which could be beneficial on some deep learning tasks.

A major issue with the deconvolution operation is how it relates to instability. Since the operation involves division in the frequency domain, any frequency component equal to zero will cause the output to be undefined and therefore cause the operation to be unstable. Even with frequency components close to zero will cause a large output. However, we circumvent this issue through deep learning optimisers. Since deep learning optimisers reduce the error in a network, optimisers will avoid these unstable regions because they will produce larger errors than stable instances.

2.4 Image Scaling

In computer graphics and digital imaging, image scaling refers to resizing a digital image to different resolutions. There are two different types of images found in computer

graphics, namely raster graphics images and vector graphics images. Raster graphics refers to images that are represented by a two-dimensional grid of square pixels. In contrast, vector graphics refers to images that are created from geometric shapes defined on a Cartesian plane. Figure 2.3 shows the difference between the two graphic types.

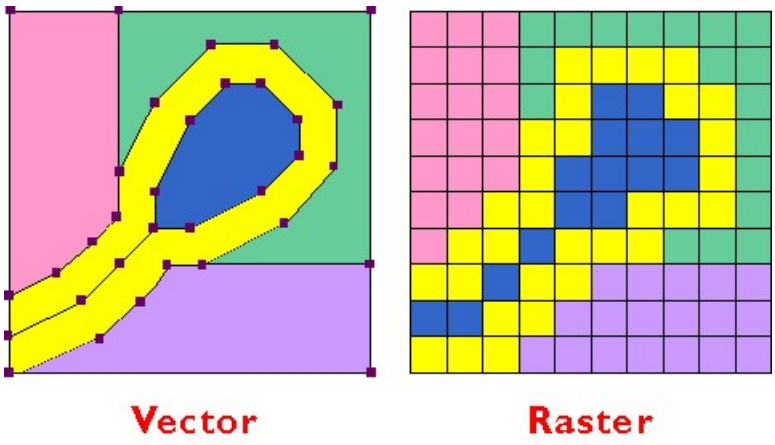

Figure 2.3: Illustration of the differences between vector and raster image [43].

Scaling vector graphic images is quite a simple task since the graphic shapes can be scaled by using geometric transformations. This can be done with no loss in image quality. On the other hand, scaling of raster graphics means that a new image with a different number of pixels must be generated. In the case where the resolution is lowered, this usually results in visible quality loss. In the context of digital signal processing, the scaling raster graphics can be seen as an example of two-dimensional sample rate conversion. In this dissertation, we will mainly focus on scaling raster images. Here are some algorithms for scaling images.

2.4.1 Nearest-neighbour Interpolation

Nearest-neighbour interpolation is a simple interpolation method. Interpolation is the process of approximating points of a sample function based on the points around the point of interest. The nearest-neighbour interpolation approximates points by copying the value of the point closest to it. This results in multiple pixels having the same colour when using this algorithm for upscaling. It can also preserve sharp details in pixel art but also introduce jaggedness in images that were previously smooth. A common implementation is to always round to zero instead of computing which pixel is mathematically the closest, which produces faster computations as well as few artefacts.

2.4.2 Bilinear Interpolation

Bilinear interpolation is an interpolation technique that utilises repeated linear interpolation. Linear interpolation estimates function values by constructing a straight line between two known data points and utilising this line to determine the value of the function at a specific intermediate location. This produces more natural-looking images compared to nearest-neighbour interpolation. Computationally, bilinear interpolation takes more time to compute when compared to nearest-neighbour; however, it is still faster than more advanced methods such as bicubic interpolation.

2.4.3 Bicubic Interpolation

Bicubic interpolation extends cubic spline interpolation to interpolating functions that can be represented on a two-dimensional grid. The interpolated surface, referring to the filter shape, is smoother than the interpolated shape of both bilinear interpolation and nearest-neighbour interpolation. Bicubic interpolation can be computed using cubic convolutions, cubic splines, or Lagrange polynomials. In image processing, bicubic interpolation is picked above bilinear and nearest-neighbour interpolation in image re-scaling when speed is not a problem. Compared to bilinear interpolation, which interpolates using 4 pixels, bicubic interpolation takes into account 16 pixels, a 4×4 grid surrounding the interpolation point.

2.5 Introduction to PyTorch

PyTorch is an open-source machine learning library. It is the framework that was used to build, train and test each neural network in this dissertation. PyTorch provides a flexible platform for deep learning research and applications. It was developed by Facebook's AI Research (FAIR) lab and it has emerged as one of the preferred tools for researchers and developers alike. This is due to its dynamic computational graph, intuitive design, and robust ecosystem.

2.5.1 Autograd

At the heart of PyTorch is the concept of a dynamic computational graph, which has been implemented using the `autograd` package. In contrast to static computational graphs, as found in other libraries like TensorFlow (prior to version 2.0), dynamic graphs are constructed during the computation of the forward pass of the neural network. This allows for greater flexibility, making it easier to modify network architectures during runtime and facilitate development of more complex and sophisticated models.

The `autograd` package showcases PyTorch's ability to automatically compute gradients, which is essential for backpropagation used in training neural networks. Every tensor object in PyTorch can have its gradient computed and stored, enabling seamless optimisation of model parameters.

The `autograd` package enables us to alter and test different designs of the deconvolution layer without having to derive and implement the gradient formulae for these operations.

2.5.2 Tensor Library

At its core, PyTorch revolves around the tensor, a multi-dimensional array similar to NumPy arrays but optimised for GPU acceleration. PyTorch tensors provide an extensive list of operations, ensuring compatibility with GPU computation without the need for complicated syntax. This integration between CPU and GPU computations allows developers and researchers to write code once and run it seamlessly on both platforms, facilitating faster experimentation and development.

2.5.3 The Neural Network Module

To help with the design and training of neural networks, PyTorch introduces the `nn` module. This module encompasses a collection of pre-defined layers, loss functions, and optimisation methods, encapsulating the intricate details and providing a high-level interface to users. Using the `nn.Module` class, users can define custom network architectures, benefiting from the in-built functionality for parameter management, GPU integration, and more. The deconvolution layer is built on this `nn.Module`.

2.5.4 Ecosystem and Community

Over the years, PyTorch's community has grown exponentially, contributing a wealth of tools, libraries, and extensions. Whether it's torchvision for computer vision tasks, torchaudio for audio processing, or torchtext for natural language processing (NLP), the PyTorch ecosystem is ever-expanding, catering to a diverse range of research areas and applications.

2.5.5 Integration with Other Tools

PyTorch seamlessly integrates with various other popular libraries and tools, such as Weights and Biases[1] for visualization, ONNX for model export, and many more. This ensures that while working within the PyTorch environment, one is not limited and can integrate the broader machine learning ecosystem into their PyTorch networks.

2.5.6 Conclusion

Essentially, PyTorch offers a balance between flexibility and ease of use, making it a great choice in the machine learning domain. Whether one is prototyping a new idea, conducting advanced research, or deploying a production-grade application, PyTorch provides the necessary tools and functionalities to make the process efficient and intuitive.

2.6 Conclusion

In this chapter, we explored the foundational concepts that are imperative to understanding the work done in this dissertation. We began with an in-depth examination of the convolution operation in section 2.1, presenting its mathematical formulae, providing insight to intuitions behind it, and its association with FFTs. Building on this foundation, section 2.2 delved into CNNs, detailing the architecture, components, training processes, and their applications in image domains. Subsequent to this, section 2.3 introduced the deconvolution operation. The section provides an overview of the deconvolution operation and motivates our deep learning approach. Image

[1]Learn more about Weights and Biases here: Weights & Biases Documentation

scaling techniques, vital for super-resolution tasks, were highlighted in section 2.4 with emphasis on varied interpolation methodologies. Lastly, section 2.5 acquainted readers with PyTorch, the instrumental machine learning library utilised in this dissertation, emphasising its dynamic computational graphs, tensor operations, and the `nn` module. Together, these sections equip the reader with the background to navigate the work in this dissertation.

Chapter 3

Related Work

In this chapter, we review the literature that is related to work done in this dissertation. First, we dive into previous work done on single image super resolution (SISR). We start with detailing methods used prior to deep learning. Next, we give an overview of all the work done on deep learning for SISR. We look into the different network architectures that have been proposed previously. Architectures such as SRCNN [24] and ESPCN [44] which are the two architectures we used in this dissertation. In section 3.2, we have a look at work done on image classification. Like SISR, we first give an overview of the algorithms used prior to deep learning. Next, we look into the work done using deep learning. We look into different network architecture but highlight the LeNet-5 [45] since this is the architecture we used to train a DeNN on image classification. Finally, we turn our attention to image deconvolution. This section starts with an overview of classical signal processing methods used for image deconvolution. Next, it provides more modern approaches like convolutional neural networks. It then relates to more recent work done using deconvolution operations in deep learning for non-blind deblurring. The section then ties this work to the work done in this dissertation.

3.1 Single Image Super Resolution

Super resolution (SR) is a popular area in the field of image processing and computer vision. SR aims to improve the quality of an image by increasing the resolution. This is done by passing a low-resolution (LR) image as input and reconstructing a high-resolution (HR) image as an output. This is especially valuable in domains such as satellite imaging, medical imaging, and surveillance, where obtaining high-quality images is vital. This

section touches on work done prior to the advent of deep learning and work done using deep learning.

3.1.1 Prior to deep learning

Before the emergence of deep learning, super resolution techniques were predominantly based on interpolation methods, such as bicubic interpolation [46], bilinear interpolation [47], and Lanczos resampling [48]. These methods were simple and computationally efficient but produced images with blurred edges and lacked finer details.

Later, more sophisticated methods were developed. These methods involved extracting patches from the low-resolution image and searching for similar patches in an external HR image database. We call these methods example-based super resolution, and they exploit the redundancy of image patches to reconstruct the HR image [49].

3.1.2 Deep Learning

The introduction of deep learning brought a transformation to the entire field of computer vision, including super resolution. Convolution Neural Networks (CNNs) emerged as a powerful tool for SR. This can be attributed to their ability to learn hierarchical features from LR images and use these features in the reconstruction of HR images.

One of the foundational networks in single image super resolution (SISR) is the Super Resolution Convolutional Neural Network (SRCNN) introduced by Dong *et al.* [24]. The SRCNN introduced an end-to-end SR technique where the network is trained to map the LR images directly to the HR reconstructions. This network consists of three convolution layers. The first layer was used as a patch extraction layer, the following layer made non-linear mappings of the extracted patches, and the final layer reconstructed the mappings to an HR image. The SRCNN achieved better performance over pre-deep learning methods and set a new benchmark for SR tasks.

Following the introduction of the SRCNN, many deep learning based SR methods were proposed. Among these networks, the Efficient Sub-Pixel Convolutional Neural Network (ESPCN) stands out due to its innovative approach [44]. Instead of increasing the resolution at the input layer, ESPCN performs upscaling in the feature space using a pixel shuffle layer. This reduces the computational resources required and, therefore,

allows standard hardware to run real-time SR.

Both the success of the SRCNN and ESPCN pushed further research into deeper and more sophisticated architectures, such as VDSR [50], EDSR [51], and SRGAN [25]. These networks, with advancements in training strategies and loss functions, have enabled SR to reconstruct high-resolution images that are often indistinguishable from the true HR image.

In conclusion, the field of SR has witnessed massive advancements over the years. It progressed from simple interpolation techniques to sophisticated deep learning architectures. Networks such as SRCNN and ESPCN have played an essential role in the evolution of super resolution as they set the stage for current and future innovations. We anticipate that as computational resources continue to grow, SR will achieve better performance.

3.2 Image Classification

Image classification is a fundamental task in computer vision. The goal of the task is to understand and categorise images from a number of different labels. Unlike object detection, which involves locating as well as categorising multiple different objects in a single image, image classification typically applies to images containing only one object. Deep convolution neural networks have led to breakthroughs in image classification [52]. CNNs learn low, high, and mid-level features [52], making an end-to-end multi-layer solution that can be enhanced by deepening the network.

3.2.1 Prior to deep learning

Computer vision can be dated back to the late 1960s and began at universities that were pioneering in Artificial Intelligence. The goal was to mimic the human visual system so it could help robots display intelligent behaviour [23]. In fact, it was believed that this could be accomplished through a summer project which involved attaching a camera to a computer and having it "describe what it saw" [53] [54].

At the time, what set computer vision apart from digital image processing was the desire to extract three-dimensional structures from images with the aim of understanding an

entire scene. Research from the 1970s built the foundations for many computer vision algorithms today.

In the next decade, research was based on mathematical analysis and quantitative aspects of deep learning. These studies led researchers to the realisation that many mathematical concepts could be treated within the same optimisation framework as regularisation and Markov random fields [55]. The 1990s saw researchers begin to use statistical techniques to recognise faces in images.

3.2.2 Deep Learning

With the resurgence of neural networks in the late 2000s and early 2010s, deep learning started gaining traction, especially in the field of computer vision. One of the earliest, and arguably the most significant deep neural network, is LeNet-5 [45]. It was proposed by Yann LeCun *et al.* in the late 1990s and was initially designed to recognise handwritten digits [17]. The network contained several convolutional layers, which were followed by pooling layers. This network was revolutionary due to its ability to understand raw images by learning spatial hierarchies of features so that the need for manually crafted feature engineering was not needed anymore.

LeNet-5 established the foundation for many modern CNN architectures. Its groundbreaking performance on the MNIST dataset paved the way for larger and more complex architectures that could solve larger and more diverse datasets.

The following years saw numerous advancements in CNNs. In 2012, the AlexNet architecture [10] dominated the ImageNet competition, which set a precedent for deep CNNs that are trained on large image classification datasets. This spawned a multitude of different sophisticated architectures such as VGG, ResNet, Inception, and others. All of them pushed the boundaries of performance.

Deep learning networks, including those based on CNN architectures, have become the standard for tackling various computer vision problems, such as image classification, segmentation, object detection, and image generation.

Initial research and applications were focused on understanding natural images but were soon adapted to various other domains, such as medical imaging, and satellite imagery. The robustness and adaptability of networks made them versatile tools for extracting meaningful information from large amounts of unstructured data.

In conclusion, the rise of deep learning and CNNs can be attributed to the foundation that the LeNet-5 provided. It revolutionised the field of computer vision. The CNNs have managed to outperform traditional machine learning and digital image processing techniques in multiple applications. Researchers are able to adapt networks for tasks of varying complexities by scaling the network's depth or width while balancing the trade-off between performance and the computational resources required. Furthermore, as advancements in GPU technologies continue to unfold, it has enabled deep learning practitioners to train deeper and more sophisticated models. Today, problems such as object detection, image generation, and even video understanding can be computed in real time. The legacy of LeNet-5 stands as a testament to the power of deep learning and CNNs. LeNet-5 acted as a catalyst by setting the stage for innovations in deep learning and computer vision.

3.3 Image Deconvolution

Image deconvolution is an essential process in the field of image processing and computer vision, primarily used for removing blur from images. The goal of image deconvolution is to reconstruct a high-quality image from a degraded image that has been blurred by a known or unknown filter. This technique is widely used in various domains such as astronomical imaging, medical imaging, and photography.

3.3.1 Prior to Deep Learning

Before the advent of deep learning in image processing, deconvolution techniques relied heavily on classical signal processing methods. These methods typically involved a deconvolution operation, where the degraded image is filtered with the inverse of the blurring function. However, deconvolution was highly sensitive to noise and often produced unsatisfactory results in practical applications.

Another popular method was the Wiener filter, which provided a statistical approach to deconvolution. This method aimed to minimise the overall mean square error between the estimated and true image. While the Wiener filter performed better than inverse filtering, especially in the presence of noise, it still struggled with recovering high-frequency details in the image.

Lucy-Richardson deconvolution, a method based on maximum likelihood estimation, also saw significant use. This iterative technique was particularly useful for astronomical and microscopic images but was computationally expensive and often slow for large images [56].

3.3.2 Deep Learning

The introduction of deep learning in image deconvolution marked a significant advancement in the field. Convolutional neural networks (CNNs) were employed for learning the deconvolution process end-to-end. These models could handle complex and non-linear blur patterns, which were challenging for traditional methods.

One of the early and significant contributions was the use of CNNs for blind image deconvolution, where the blur kernel is unknown. Networks like DeblurGAN [27], introduced by Kupyn *et al.*, leveraged Generative Adversarial Networks (GANs) for deblurring tasks, producing sharp images from blurred inputs without explicitly estimating the blur kernel.

Another advancement was the incorporation of deep learning in non-blind deconvolution, where the blur kernel is known or estimated. Here, deep learning methods showed superior performance in restoring fine details and textures, compared to classical algorithms.

Recent trends in image deconvolution involve using deep learning for specific applications, such as removing motion blur from photographs taken with handheld devices, or deblurring medical images for improved diagnosis. These specialised networks are trained on large datasets and can handle various types of blur effectively.

Deep learning-based deconvolution methods continue to evolve, with researchers exploring various architectures and training strategies. The focus has been on improving the speed of deconvolution, reducing artefacts, and enhancing the capability to handle more complex and realistic blur scenarios.

The paper [57] introduces an innovative approach to non-blind image deblurring that leverages the strengths of both classical Wiener deconvolution techniques and modern deep learning methods. Traditional Wiener deconvolution, while effective in certain scenarios, often falls short in handling the complexities of real-world image blur, especially when noise and other artefacts are present. By integrating deep learning, the proposed method overcomes these limitations through a two-step process.

Firstly, the method utilizes a feature-based Wiener deconvolution module that operates in a deep feature space rather than the conventional image space. This involves embedding the Wiener deconvolution step within a deep neural network framework, allowing the deconvolution process to be guided by learned features extracted from the blurry input. These features, obtained through convolutional layers and residual blocks, capture more nuanced information about the image, facilitating more effective artefact removal and detail restoration.

Secondly, the deblurred features are refined using a multi-scale refinement module. This module processes the features at multiple scales, progressively recovering fine details and small-scale structures that are often lost in traditional deconvolution methods. The multi-scale approach ensures that both global and local image details are preserved, resulting in a clearer and more accurate final image.

One of the key innovations of this method is its adaptive noise estimation capability. Unlike many existing approaches that require a predefined noise level, the proposed network can dynamically estimate the noise level from the input features. This adaptability enhances the network's robustness, enabling it to handle varying levels of noise effectively.

The effectiveness of this approach is demonstrated through extensive experiments on both simulated and real-world blurred images. The results show significant improvements in artefact suppression, detail recovery, and overall image quality compared to state-of-the-art non-blind deblurring methods. Quantitative evaluations using metrics such as PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index) further validate the superiority of the proposed method.

In summary, the paper presents a compelling case for combining classical deconvolution techniques with deep learning to address the challenges of non-blind image deblurring. By operating in a deep feature space and employing a multi-scale refinement strategy, the method achieves remarkable improvements in deblurring performance, particularly in terms of handling noise and preserving image details.

The paper [58] presents a novel approach to address the complex challenges of non-blind image deblurring in low-light environments. This method, named INFWIDE (Image and Feature Space Wiener Deconvolution Network), ingeniously combines the strengths of image space and feature space Wiener deconvolution to effectively manage the prevalent noise and saturation issues found in low-light images. INFWIDE introduces a sophisticated dual-branch architecture that simultaneously processes information in

both the image and feature domains. The image space branch is dedicated to explicitly removing noise and reconstructing saturated regions, while the feature space branch focuses on suppressing ringing artefacts that often degrade image quality.

One of the core innovations of INFWIDE is its multi-scale fusion network, which adeptly integrates the outputs of both branches. This integration ensures that the complementary strengths of each branch are leveraged, resulting in significantly enhanced deblurring performance. The multi-scale fusion network not only improves the overall visual quality of the deblurred images but also ensures that fine details are preserved, which is critical for high-quality night photography.

In terms of technical advancements, INFWIDE employs a set of loss functions that incorporate both a forward imaging model and backward reconstruction. This close-loop regularization strategy helps secure good convergence of the deep neural network, ensuring robust and reliable performance during training and inference. Moreover, to enhance the applicability of INFWIDE in real-world low-light conditions, the authors developed a physical-process-based low-light noise model. This model synthesizes realistic noisy night photographs, providing a more accurate and challenging dataset for training the network.

The extensive experiments conducted on both synthetic and real data demonstrate that INFWIDE outperforms traditional deblurring algorithms. It particularly excels in scenarios involving severe noise and saturation, where existing methods often fail. By effectively addressing these issues, INFWIDE sets a new benchmark for deblurring quality in low-light environments, showcasing superior performance in preserving image details and reducing artefacts.

Both [57] and [58] use Wiener deconvolution for non-blind image deblurring. The former applies a deconvolution in the feature space while the latter deconvolves in both the feature and image space. In this dissertation, we apply a deconvolution in the image space and extend two other computer vision tasks namely single image super resolution (SISR) and image classification.

In conclusion, the field of image deconvolution has undergone a significant transformation from classical signal processing techniques to sophisticated deep learning models. Most research, investigates networks that use the convolution as a fundamental operation. However, there is some research done on image deconvolution using the deconvolution operation. These networks have shown impressive performance on image deblurring. In this dissertation we aim to extend this investigation to more computer vision tasks such

as SISR and image classification.

Chapter 4

Deconvolution Layer Architecture

Deep learning and artificial intelligence are rapidly evolving fields. Most research in deep learning is on multi-purpose layers such as fully connected layers and convolutional layers. However, in the diverse landscape of deep learning, one-size-fits-all solutions are uncommon, just as in other fields of science and engineering. In light of this, it's important to explore alternative approaches that may offer different advantages. The deconvolution layer represents one such alternative that allows a neural network to learn long-range correlations in the data earlier in the network. Experiments conducted in chapter 5 and 6 determine the effectiveness and applicability of the deconvolution layer.

In this chapter, we delve into the architecture and rationale behind the deconvolution layer in deep learning. Section 4.1 offers a motivation for the deconvolution layer, discussing its ability to capture long-range spatial dependencies which often limits CNNs. Section 4.2 delves into the FFT (Fast Fourier Transform) implementation of the deconvolution layer, highlighting how this approach enhances computational efficiency. The subsequent sections address various critical aspects of the deconvolution layer's design and implementation. In section 4.3, we explore key design strategies, such as the significance of the first element's trainability and filter initialisation techniques. Section 4.4 presents a practical example of applying a 1D deconvolution layer in audio classification, providing empirical insights into its effectiveness. The chapter progresses to section 4.5, where we outline a high-level implementation of the deconvolution layer, breaking down its operational steps. Finally, section 4.6 details the experiments designed to test the deconvolution layer in various scenarios, including single image super resolution (SISR) and image classification, setting the stage for a thorough empirical evaluation of this layer in the next two chapters.

4.1 Motivation

As discussed in [59], the ability of CNNs to learn long range spatial dependencies is limited by the localised receptive fields of each layer. Essentially, each point in a channel of a convolution output involves aggregating input information over a domain determined by the kernel size, often 3×3 or 5×5. The overall receptive field of the network can be increased by cascading multiple layers, but simply increasing the depth of a feed-forward network is an inefficient means of learning long range dependencies.

Viewing a single channel in a CNN layer as a shift-invariant filter, the kernel defines the impulse response. A localised receptive field is equivalent to a narrow impulse response, which corresponds to a filter that aggregates information over a limited spatial domain and is unable to capture pixel dependencies over wide pixel separations. The receptive field of a convolutional layer can be increased by using a larger kernel size. However, this increases the complexity of the layer and the number of parameters to be estimated.

The statistical principle of parsimony suggests that a model should be as complex as necessary but as simple as possible. If long range correlations are important then filters with wide impulse responses are necessary, but these should not come with a substantial increase in the number of free parameters that have to be estimated from limited training data. Filters with wide impulse responses but which are specified by a small number of parameters might be desirable as layers in standard CNN architectures.

In broad terms, Figure 4.1 defines our proposed deconvolution layer.

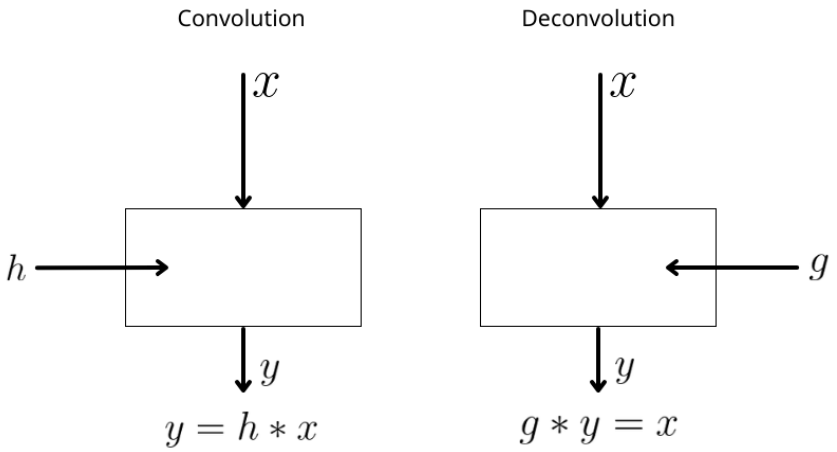

Figure 4.1: A diagram illustrating the operation of both the deconvolution and convolutional layers.

The standard convolutional layer is on the left. The trainable parameters are the

convolution kernel weights h, and the output y is the convolution of the kernel h with the input x. The deconvolution layer is on the right. Here the trainable parameters are the kernel weights g, but the relationship governing the input and output is $g * y = x$. In other words, given an input x and a kernel g, the output is that quantity y which, when convolved with g, yields the input x. This is a deconvolution operation.

If convolution with g is invertible and is inverted by a filter with impulse response g^{-1}, then the output can be calculated directly from the input as $y = g^{-1} * x$.

4.1.1 Signal Processing Motivation

The domain of 1-D signals is analytically tractable with much theory that relates to the factors of the underlying polynomials. Specifically, the location of poles and zeros of the transfer function determines the behaviour of a filter. A conventional convolutional layer corresponds to an all-zero or moving average filter. Assuming causal filters for now, with $x[n]$ the input signal and $y[n]$ the output, the following relation defines a first-order moving average filter:

$$y[n] = x[n] - ax[n-1].$$

This corresponds to the input-output relation $y[n] = h[n] * x[n]$ with $h[n] = \delta[n] - a\delta[n-1]$ the impulse response. This impulse response has a very small support, and consequently the layer has a small receptive field.

A stable and causal all-pole filter has all poles inside the unit circle and uses temporal feedback to effectively increase the impulse response and hence the receptive field. Consider now the auto-regressive recursion

$$y[n] = ay[n-1] + x[n].$$

In this case the relationship between the input and the output can be written as $h[n] * y[n] = x[n]$ with $h[n]$ as before. However, it is a simple signal processing exercise to show that this filter can equivalently be written as $y[n] = g[n] * x[n]$, with $g[n] = a^n u[n]$ and $u[n]$ the unit step function. This filter, which is specified by a single parameter, has an infinite impulse response.

The moving average filter above has the system function $H(z) = 1 - az^{-1} = \frac{z-a}{z}$. The all pole filter system function is $G(z) = \frac{1}{1-az^{-1}} = \frac{z}{z-a}$. The two are inverses of one another. The deconvolution layer proposed in this work is parameterised by the impulse response

of the inverse filter rather than of the forward filter.

4.2 FFT Implementation

Implementing a deconvolution layer in neural networks, particularly large-scale data such as images, can be computationally expensive if done in the spatial domain. The FFT provides a means to expedite the process by transforming the deconvolution operation in the spatial domain to an element-wise multiplication in the frequency domain, which is computationally more efficient.

Given the relation $g * y = x$, where g is the kernel y is the output, and x is the input, we can apply the Fourier transform to both sides of the equation. This gives us

$$\mathcal{F}(g * y) = G(\omega) \cdot Y(\omega) = X(\omega),$$

where $G(\omega)$, $Y(\omega)$, and $X(\omega)$ are the Fourier transforms of g, y, and x. To solve for $Y(\omega)$ and ultimately y, we divide both sides by $G(\omega)$. This gives us

$$Y(\omega) = \frac{X(\omega)}{G(\omega)}.$$

Finally, we take the inverse Fourier transform of both sides to get the final output

$$y = \mathcal{F}^{-1}(\frac{X(\omega)}{G(\omega)}).$$

To implement this in a deep learning framework, each of these steps would be encapsulated as operations in a computational graph, allowing for automatic differentiation and learning of the kernel g parameters through backpropagation.

The FFT-based approach offers a significant reduction in computational complexity. The standard convolution operation has a complexity of $O(n^2)$ for each output element, whereas the FFT-based approach has a complexity of $O(n \cdot log(n))$, where n is the number of elements in the input signal or image. This makes the FFT implementation particularly attractive for layers like deconvolution that require a large receptive field and would otherwise have a high computational cost.

4.2.1 Invertibility

In general the convolution operation is not invertible. Specifically, if there are any locations in the frequency response of the filter that have a zero value, then the input components at these frequencies cannot be recovered from the output (which has been multiplied by zero).

Consider a simple image restoration problem, attempting to recover high-resolution data y from an image x blurred via convolution with a non-invertible filter h_{blur} according to the process $x = h_{blur} * y$. Using a single deconvolution layer for this restoration would involve optimising over filter parameters h in the input-output relation $x = h * y$. For this problem, we will use a pixel-wise loss that the optimiser will reduce. Due to the deconvolution layer having an all-pole filter there will be unstable regions. However, the optimiser will naturally move the filter away from these regions since they will have larger errors compared to stable regions.

4.2.2 Issues with Convolving with an FFT

When convolving or deconvolving any image or signal, some problems can arise. This is especially apparent when compared to direct spatial or time-based convolution. The following are some issues that one can run into when using an FFT for convolution or deconvolution:

- **Circular Convolution:** One of the predominant issues when convolving using an FFT is that the FFT operates on a finite portion of a signal and effectively assumes periodicity. This results in circular convolution that can introduce artefacts around the boundaries of the image.
- **Computational Complexity and Memory:** While the FFT is computationally efficient for large convolution operations, it still requires large amounts of memory to store complex numbers for each pixel and data point. This is especially apparent in large images.

4.3 Design Strategies

This section highlights critical design strategies and explains the rationale behind these strategies.

4.3.1 First Element Trainable

In this section, we present a key design strategy which is comprised of two parts, the first being that we constrain the first element of each filter to 1. Note that we refer to the element at the top left of the filter as the first element. The second design choice is setting the first element as trainable or non-trainable. The next subsection motivates why it's necessary to set the first element equal to 1 using linear algebra.

Linear Algebraic Motivation

Generally, a matrix-vector formulation does not represent a time-invariant transformation well, but it is useful for exposing the effect of boundary conditions on invertibility. Consider the causal convolution $y = h * x$, with $h[n] = h_0\delta[n] + h_1\delta[n-1] + h_2\delta[n-2]$. If the boundary conditions applied simply truncate the input and output signal vectors, then the following linear transform representation is obtained:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 & 0 \\ & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & h_1 & h_0 & 0 \\ & & & h_2 & h_1 & h_0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{pmatrix}.$$

This assumes $x_n = 0$ for $n < 0$ and truncates the output elements y_n for $n \geq N$. Essentially, $\mathbf{y} = \mathbf{H}\mathbf{x}$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ and $\mathbf{H} \in \mathbb{R}^{N \times N}$.

Since $\mathbf{H}$ is triangular its determinant is the product of the diagonal elements. Thus the relationship $\mathbf{y} = \mathbf{H}\mathbf{x}$ is invertible as long as $h_0 \neq 0$, and this is the case even if the original relation $y = h * x$ is not invertible. By setting $h_0 = 1$, we can avoid the issue of h_0 getting close to 0 since the optimiser will avoid regions of instability.

First Element Trainability

In this section, we highlight the key difference between setting the first element as trainable or non-trainable. It is also important to note the first element can be set as a trainable parameter or as a non-trainable parameter:

- **First Element Trainable:** In this approach, the first element of each filter is set to trainable. This adds an extra learnable parameter to the network, allowing it to potentially learn more nuanced representations.
- **First Element Non-trainable:** In contrast to the first strategy, in this method the first element is set as a non-trainable parameter. This helps ensure that the deconvolution operation is kept stable and invertible throughout training.

The experiments conducted in this work investigate whether holding $h_0 = 1$ while optimising over the other kernel parameters is beneficial to the operation of the layer. Admittedly, if the remaining (off diagonal) elements become large then $\mathbf{H}$ could become ill conditioned, but at least formally the operation is guaranteed to always be invertible.

4.3.2 Filter Initialisation

Filter or kernel initialisation plays a vital role in deep learning. It can significantly influence the training, and ultimately the performance of the neural network. The filter refers to a set of learnable parameters in a convolutional layer, but in this case it refers to a set of learnable parameters in the deconvolution layer. A good filter initialisation is essential for the following reasons:

- **Convergence:** the filter initialisation can impact the speed at which the network converges during training. A poor initialisation could lead to the network converging slowly or even result in the network not converging at all [60].
- **Vanishing and Exploding Gradients:** Good initialisation can minimise the problems of vanishing and exploding gradients. If the initialisation places the network in an unstable region, it can cause the deconvolution operation to be unstable which makes the activation of neurons large. This can lead to neurons pushing activations into areas that have low gradients and cause a vanishing gradient problem.

- **Model Performance:** On top of convergence, a good initialisation can put the model in a favourable position on the loss curve. This can help the model find a better local minimum to converge to, which could lead to better model performance on both seen and unseen data.

We experimented with different filter initialisations on single image super resolution (SISR). We have provided a list of these initialisations as well as the findings from our experiments. Note that all these experiments set the first element as 1 and non-trainable.

- **Normal Distribution:** In this case, we initialised the rest of the filter with random numbers from a normal distribution with a mean of 0 and a variance of $\frac{1}{no.\,of\,trainable\,parameters}$. We found that using an initialisation with a normal distribution placed the network in an unstable region on the loss curve and thus caused the deconvolution to be unstable.
- **Zero Distribution:** Here we initialised all the elements (except for the first element) to 0. This initialised the filter as an "identity" filter and ensures that the deconvolution will be stable. Through our experiments, we found that this initialisation places the deconvolution layer into a sub-optimal local minimum. In this situation, the model's accuracy will not improve because the optimiser will push the network towards the local minimum (where the network already is).
- **Uniform Distribution:** Here we initialise the filter weights with random float numbers from a uniform distribution on the interval $[0, \frac{1}{no.\,of\,trainable\,parameters})$. We found that this initialisation produced a stable deconvolution filter each time while also placing the network in a position where the optimiser can find an optimal local minimum. This initialisation was chosen to be used in the deconvolution layer and in the experiments in both chapter 5 and 6.

To summarise, filter initialisation is a vital part of the training process in both neural network layers. The consequences of the choice at initialisation are propagated throughout the entire training process, and could lead to poor network performance. For filter initialisation, we have chosen to set the first element to 1 and letting the rest of the filter be random floats from a uniform distribution. We found that this type of initialisation provides stability and good placement on the loss landscape.

4.3.3 Padding

In digital signal processing, and by extension convolutional layers in deep learning, padding is the process of adding zero (or other specific values) to the edge of the signals or images. The primary motivation behind padding stems from the following reasons:

- **Avoiding Circular Convolution:** When you perform convolution in the time domain using an FFT (or its 2D equivalent for images), the data is assumed to be periodic, which leads to circular convolution. If the signal is not periodic then the circularity often causes edge artefacts. However, this can be avoided by padding the input before deconvolving it.
- **Richer Frequency Representation:** By padding the image, one increases the number of pixels that the FFT gets computed on. This increases the size of the Fourier representation of the image. This richer Fourier representation of the Fourier transform means that higher frequency components can be accurately represented.
- **Output Dimension Preservation:** This issue is only relevant to spatial or time-based convolutions, which is the type of convolution that is found inside traditional convolutional layers. In FFT-based convolution, padding inputs, and therefore increasing the size of the input image, results in a larger output. If keeping the size of the image is desired, the centre of the output of the convolution can be extracted to the original image size.

The deconvolution uses padding to mitigate the problems mentioned above. It adjusts the amount of padding to the size of the input. To ensure that the output size is consistent with the input, the layer also extracts the centre of the output at the end of its operation.

In conclusion, padding has the potential to reduce the effects of circular convolution and provide a richer frequency domain representation. It is a tool that originates from the principles of Digital Signal Processing. The question is whether the padding enhances the deconvolution layer's ability to learn patterns and features for the data that it is trained on. The experiments in the next chapter look to answer these questions. We hypothesise that zero padding will help the deconvolution neural networks (DeNNs) learn better feature maps because the deconvolution layer will have a richer frequency representation.

4.3.4 Four-factor Deconvolution

In this section, we introduce a design strategy called the four-factor deconvolution. This design strategy allows the deconvolution to apply a symmetric filter to the input. We predict that having a symmetric filter will increase the accuracy of the deconvolution layer and the network it is a part of. This could be due to the fact that convolutions with asymmetric filters can lead to shifts in the output. This is particularly problematic in tasks where a misalignment in the network output image and the ground truth image could negatively affect the network's training.

Motivation

A four-factor convolution filter was initially introduced in [61]. It leads to a symmetric impulse response or equivalently a zero phase frequency response. This ensures spatial coherence between the input and output of the layer's frequency-components. Additionally, Lim shows how applying a nonlinear-phase filter to 2D images generated artefacts that are likely detrimental to image analysis and interpretation [62]. For example, when filtered the different frequency components of edges can disperse and become spatially incoherent, making them weak and generating ringing artefacts. In the context of deep learning, four-factor deconvolution could help networks learn end-to-end mappings more easily since there will be spatial coherence inside the network.

Formulation

In 1-D signal processing, forward-backward filters are often used to eliminate this dispersion. Two systems serially cascaded, one with a real-valued impulse response $h[n]$ and the other with the time-reversed impulse response $h_r[n] = h[-n]$, produce an overall system with an even symmetric impulse response $h_{\text{eff}}[n] = h_r[n] * h[n] = h[-n] * h[n]$. The corresponding filter will be zero phase.

The above forward-backward filter can be considered a two-factor filter, where one factor has the impulse response h and the other the time-reversed impulse response h_r. In 2-D, given a factor $h[x, y]$, we could define a four-factor filter with a combined impulse response

$$h_{\text{eff}}[x, y] = h[x, y] * h[-x, y] * h[x, -y] * h[-x, -y].$$

This impulse response will be completely parameterised $h(x, y)$ but will be symmetric in both the x and the y directions. Hence it will be a zero-phase filter.

Implementation

The four factor deconvolution is essentially four filters which are reflections of the original filter. The first filter is just the unchanged deconvolution filter. The other filters are obtained by reflecting the filter across the first dimension, the second dimension, and reflecting across both dimensions, respectively. The product of these four filters is a symmetric filter that eliminates any spatial shifts that the deconvolution layer could produce. The four-factor deconvolution can be computed using the following steps:

1. Compute the two-dimensional FFT of the deconvolution kernel.
2. Generate the three additional filters by flipping and aligning the original filter's frequency representation.
3. Multiply all four frequency representations element-wise, creating the four-factor deconvolution filter.
4. Divide image input by the four-factor filter.
5. Compute the inverse FFT on the result of the last step.

Conclusion

The four-factor deconvolution ensures coherence between the input and output spatial features by convolving the input with a symmetrical filter. Additionally, the four-factor deconvolution increases the width of the impulse response which will help the layer easily learn long-range relationships in the data. The experiments in the next chapter seek to answer whether a four-factor deconvolution filter is beneficial for training deconvolution neural networks (DeNNs) on both SISR and image classification.

4.3.5 Bias

Here, we introduce the filter bias design strategy. This design strategy is relatively simple. It involves adding a trainable bias term to each deconvolution filter in a deconvolution

layer which allows the optimiser to learn mappings with an offset. This design strategy aims to enhance the flexibility and performance of the deconvolution layer in various deep learning applications.

Motivation

The primary motivation for adding a filter bias is that it allows the deconvolution layer to adjust the level of activation in its output. This can be particularly beneficial in scenarios where the data processed by the layer has inherent biases or where shifting the activation range could lead to more effective learning. The filter bias can serve as a mechanism for adapting the layer's responses to suit the specific characteristics of the input data, thereby improving the overall performance of the network.

4.4 1D Example

In this section, we use a deconvolution layer for an audio classification task using the UrbanSound8k dataset, which consists of urban sounds from 10 different classes such as car horns, children playing, and dogs barking. The network used consists of a 1D deconvolution layer followed by a fully connected layer to classify the sounds into the various categories. As a baseline, we also train a convolutional layer with the same structure as the deconvolution layer so that we can evaluate the effectiveness of the deconvolution operation. Same structure refers to the convolutional layer also using an FFT convolution, two factor convolution, and setting the first element as not trainable. We refer to the network with the deconvolution layer as the deconvolution neural network (DeNN) and the network with the convolutional layer as the convolutional neural network (CNN).

4.4.1 Network Architecture

1. **1D Deconvolution Layer:** This layer applies a deconvolution operation to the input of the network. The deconvolution layer only uses one filter that consists of 3 elements. The first element is set to 1.0 and is not trainable and the other two parameters are trainable, which is all the trainable parameters in this layer. It also applies a two-factor deconvolution which means the input is deconvolved with both

the filter and the reverse filter. Finally, a ReLU activation is applied to the output of this layer.

2. **Fully Connected Layer:** This layer classifies the output of the deconvolution by multiplying it by an 90000×10 matrix and applying a softmax activation to the output.

4.4.2 Training

Both networks are trained using the Adam optimiser [63] with a learning rate of 1×10^{-3}. The networks were trained on the UrbanSound8k dataset for 10 epochs each using a batch size of 32.

4.4.3 Results

After training the network on the UrbanSound8k dataset, we found that the DeNN achieves an accuracy of 89% whereas the network with the CNN achieves an accuracy of 87%. This shows that the deconvolution layer is slightly better than the convolutional layer in this audio classification task.

Another key aspect to look at is the impulse response. This will help showcase the difference between both the convolutional and deconvolution layer. In figure 4.2, we present the impulse response for the deconvolution layer and the convolutional layer.

Figure 4.2 shows the impulse response of the deconvolution layer after it has been trained for audio classification. If we look at the plot, we see that the deconvolution layer has a wider impulse response and therefore is able to capture longer range correlations for the input data which, translates to better accuracy of the network.

4.4.4 Summary

This example illustrates that a deconvolution layer can be effectively used for tasks such as audio classification, as demonstrated on the UrbanSound8K dataset. The experiment suggests that deconvolution layers might have an edge over traditional convolutional layers in certain deep learning tasks. In this example, we also investigated

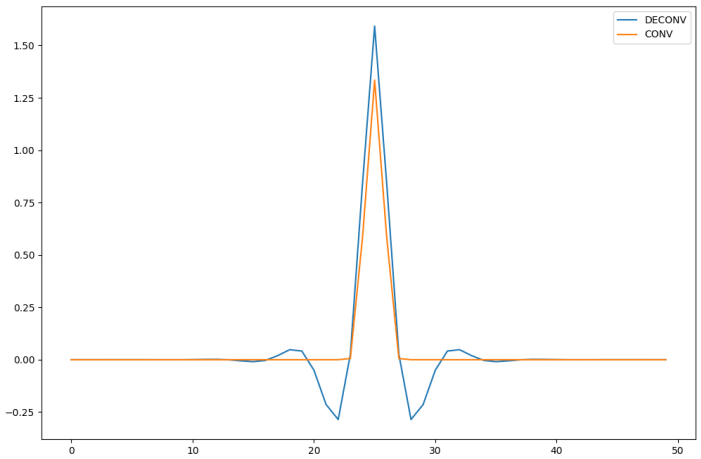

Figure 4.2: The deconvolution layer's response to impulse input.

the impulse response of both the deconvolution and convolutional layer. We found that the deconvolution layer has a wider impulse response, which indicates that it can capture longer range correlations compared to the convolutional layer while still having the same number of trainable parameters.

4.5 High Level Implementation

This section presents the implementation of the deconvolution layer at a high level. The implementation of this layer involves several steps, including padding, Fourier transformations, filter multiplication, and inverse transformations.

4.5.1 Overview

The deconvolution layer's operation can be divided into the following steps:

1. **Padding the Input:** The input image or feature map is padded to ensure that the output size remains consistent after the deconvolution operation.
2. **Fourier Transformation:** The padded input is transformed into the frequency domain using the Fast Fourier Transform (FFT).

3. **Filter Multiplication:** The transformed input is multiplied with the deconvolution filter in the frequency domain. This step involves the four-factor deconvolution to ensure spatial consistency and stability.

4. **Inverse Fourier Transformation:** The product from the previous step is transformed back into the spatial domain using the Inverse Fast Fourier Transform (IFFT).

5. **Truncation and Bias Addition:** The output is truncated to the desired size, and a bias term is added to finalise the deconvolution operation.

4.5.2 Pseudo-code

The pseudo-code in Figure 4.3 provides a step-by-step breakdown of the deconvolution layer's operation:

```
function DeconvolutionLayer(x):
# Step 1: Padding the Input
padded_x = pad(x)

# Step 2: Fourier Transformation
X = FFT(padded_x)

# Step 3: Filter Multiplication
# Create the four-factor filters
G1 = 1 / FFT(filter)
G2 = flip_and_shift(G1, vertical=True)
G3 = flip_and_shift(G1, horizontal=True)
G4 = flip_and_shift(G1, vertical=True, horizontal=True)

# Multiply the filters to get the final filter
G = G1 * G2 * G3 * G4

# Multiply with the transformed input
Y = X * G

# Step 4: Inverse Fourier Transformation
y = IFFT(Y)

# Step 5: Truncation and Bias Addition
y = truncate(y)
output = y + bias
```

Figure 4.3: Pseudo-code illustrating the step-by-step process of the deconvolution layer's forward operation.

4.6 Experiment Motivation

The first set of experiments that were conducted in this work were on SISR. SISR was chosen because the downscaling process (bicubic interpolation) shares similar properties to a convolution with a blur filter. We hypothesise that a DeNN will perform better than a CNN because its operation is better suited for an SISR task. Its property to learn long-range relationships in the data will allow it to consider a larger area of pixels when applying a convolution (with the inverse filter) and will therefore help it pick more suitable values for the image reconstruction.

We tested the deconvolution layer in two similar networks, namely, SRCNN and ESPCN [24]. In the SRCNN experiments, we investigated the impact that the different design strategies have on network performance on SISR and whether our hypotheses are correct. In the ESPCN experiments, we explored the impact that hyperparameters such as filter size, number of filters and loss functions have on network performance in SISR. These experiments were designed to test the deconvolution layer's performance on SISR while also elucidating the scenarios that the deconvolution layer shows its effectiveness in.

The second set of experiments were conducted on image classification. Image classification was chosen because it is often used to benchmark CNNs and other vision-based neural networks. Additionally, we chose image classification because it will showcase the deconvolution layer's ability to learn long-range correlations. We predict that this ability of the deconvolution layer will allow it to classify better than the convolutional layer.

For experiments on image classification, we tested the deconvolution layer by adding it to the LeNet-5 architecture [45]. The first experiments were conducted by progressively replacing more convolutional layers with deconvolution layers. By replacing convolutional layers systematically, we will find the best configuration which can be used in the later experiments.

These experiments will answer our questions on whether the deconvolution layer is able to learn long-range dependencies and if it is more suitable for computer vision tasks that require an inversion of the convolution operation. Additionally, we investigate the best configuration for the deconvolution layer by testing how each hyperparameter affects its performance.

4.7 Conclusion

This chapter detailed the design of the deconvolution layer by first providing a motivation for why it will be beneficial in a deep learning context. Next, we dived into the FFT implementation while also highlighting the issue with invertibility and convolving in the frequency domain. Subsequently, we presented the different design strategies that are implemented in the deconvolution layer. These design strategies include setting the first element as trainable, four-factor deconvolution, padding, and filter bias. Next, we moved onto a 1D example, where a DeNN and a CNN were trained on audio classification. This helps to motivate the deconvolution layer's potential in deep learning tasks. Lastly, we provided a high-level implementation of the deconvolution layer.

Section 4.6 gave an overview of the different experiments that will be conducted on the deconvolution layer. The section motivates why particular computer vision tasks were chosen. The section also elaborates on the experiments conducted to test the performance of the different design strategies. The results and details for the experiments follow in chapters 5 and 6.

Chapter 5

Single Image Super Resolution

This chapter presents the work on super resolution using a deconvolution neural network. Single image super resolution is a computer vision task that increases the resolution and quality of images. In other words, it reconstructs a high-resolution image from a lower-resolution image. This can have various applications, such as medical imaging, security, and surveillance. The reconstruction of the high-resolution image depends on how information is extracted and used in the low-resolution image. SISR is an ill-posed task due to the same low-resolution (LR) image having multiple potential high-resolution (HR) counterparts, making SISR inherently a challenging problem. In recent research, convolutional neural networks (CNNs) have been popular in tackling SISR [24] [50] [64] [51] [65] [66] [67]. This is due to their ability to learn complex non-linear mappings between their inputs and outputs. Even with the outstanding performance that CNNs have achieved, they still face two limitations. One is that most methods blindly increase the depth or width of the network to boost the network's ability to extract features. Still, they forget to utilise all the contextual features present in the low-resolution image. As network depth increases, the network gradually loses more of the contextual information of the LR image. Additionally, during the high-resolution image reconstruction, most networks only use the spatial information from the low-resolution image and ignore useful features in the high-resolution space that can aid in reconstructing visually appealing high-resolution images.

Section 5.1 presents the different networks tested on SISR. This section also presents the baselines that these networks will be tested against. Section 5.1.1 presents the super resolution deconvolution neural network (SRDeNN), which is an adaptation of the SRCNN. The SRCNN is also the baseline for the experiments using the SRDeNN. Section 5.1.2 uses a similar network architecture to the SRCNN, called the ESPCN.

Section 5.2 presents the different datasets that are used to train and evaluate the networks on SISR. We introduce the 91-image dataset and the Set5 dataset which are used for training and evaluation respectively.

Section 5.3 elaborates on the details of the experiments that were conducted on the networks. These experiments outline the questions we aim to address and the section explains how conducting the experiments will contribute to addressing this study.

Section 5.4 showcases and discusses the results from the experiments outlined in section 5.3. We found that integrating deconvolution into both the SRCNN and ESPCN does not yield better performance than the original architecture. Furthermore, we tested different design strategy combinations and hyperparameter configurations, but the DeNN failed to outperform both CNN architectures. However, there are promising results from the DeNN when a frequency-based optimiser is chosen.

Finally, we conclude the work done on single image super resolution by summarising the results found in the previous section.

5.1 Algorithms and Networks

This section presents the proposed SISR networks along with their respective baselines.

Section 5.1.1 presents a super resolution deconvolution neural network (SRDeNN) and its baseline, the super resolution convolutional neural network (SRCNN). Both of these networks are based on [24].

Section 5.1.2 presents the ESPCN network. It is similar to the SRCNN, except low-resolution images are not upscaled before being passed into the network. In contrast, in the previous network they are upscaled using bicubic interpolation before being passed into the network.

5.1.1 SRCNN

The SRCNN was initially presented in [24], where they proposed a deep learning method over traditional example-based methods. The network architecture is relatively straightforward since it consists of only three convolutional layers of varying numbers

of filters and filter sizes. For SISR, we proposed an adapted version of the SRCNN, where we replaced the first convolutional layer with a deconvolution layer. This will be referred to as the SRDeNN. Here is an outline of the SRDeNN architecture. Layer 1 is a deconvolution layer with 64 filters ($n_1 = 64$), and the size of the filter is set to 9 by 9 ($f_1 = 9$). Layer 2 is a convolutional layer with 32 filters ($n_2 = 64$), and the size of the filter is set to 5 by 5 ($f_2 = 5$). Finally, the third and final layer has a filter size of 5 by 5 ($f_3 = 5$), and the number of filters is equal to the number of output channels ($n_3 = n_{channels}$). The baseline, the SRCNN, has the same number of filters and filter sizes as the SRDeNN, thus keeping the network architectures as similar as possible while keeping the learnable parameters the same. Figure 5.1 shows a diagram of the SRCNN architecture.

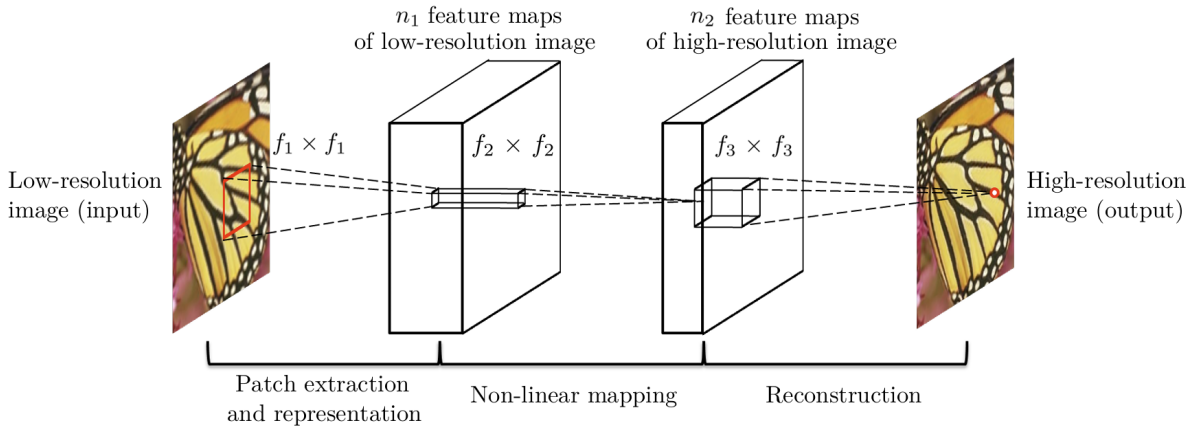

Figure 5.1: High-level diagram of the SRCNN architecture [24].

5.1.2 ESPCN

ESPCN uses a similar network architecture as the SRCNN, but there are a few key differences between ESPCN and SRCNN:

1. An additional layer called the pixel shuffle has been added to the end of the network. The pixel shuffle layer reshuffles elements in a tensor with shape $(B, C \times r^2, H, W)$ to a tensor of shape $(B, C, H \times r, W \times r)$, where r is the upscaling factor.

2. The final convolutional layer's channels are increased from $n_3 = n_{channels}$ to $n_3 = n_{channels} \times r^2$, where r is the upscaling factor.

3. This network architecture uses the tanh activation function as opposed to a ReLU activation function.

4. In the previous network, the low-resolution image is upscaled using bicubic interpolation before being passed into the network. However, this is not the case here since pixel shuffle has been added to the end of the network.

5. The filter sizes have been adjusted slightly to $f_1 = 5$, $f_2 = 3$, and $f_3 = 3$.

Like the SRCNN, we test the deconvolution layer by replacing the first convolutional layer with a deconvolution layer. We refer to the network with this replacement as an ESPDeN.

5.2 Datasets

In this section, we present the two datasets that are used to train and evaluate the SISR networks above. Firstly, we look at the 91-image dataset which is used to train the networks on SISR. Next, we look at the Set5 dataset which is used to evaluate the accuracy of the network on unseen data.

5.2.1 The 91-image dataset

The 91-image dataset is a popular dataset used for training networks for single image super resolution (SISR) and other image processing tasks such as denoising and restoration. This dataset was one of the earlier collections that were used for training and evaluating super resolution models before larger datasets, like Div2K [68], were introduced.

This dataset is not split into subsets. In fact, the whole dataset is used during training, while another smaller dataset, such as Set5, is used as a test set. To create the low-resolution images, down-sampling techniques are applied to the high-resolution images. During training and evaluation, the low-resolution images are passed into the network, and the network attempts to reconstruct a high-resolution image.

The 91-image dataset has been a vital resource for research in image super resolution. It has contributed to the development and evaluation of numerous super resolution algorithms. Despite the introduction of larger, more comprehensive datasets, the 91-image dataset is still popular and is frequently used as a benchmark for super image resolution.

Figure 5.2: Examples of images found in the 91-image dataset.

5.2.2 Set5 dataset

The Set5 dataset is one of the most popular datasets for evaluating super resolution algorithms. It was initially introduced by Bevilacqua *et al.* in their 2012 paper, "Low-complexity Single-image Super-resolution based on Nonnegative Neighbour Embedding" [69], and is widely used as a standard for comparing the performance of different super resolution algorithms.

The dataset is relatively small, containing only five images. The images in the dataset provide a diverse set of contexts, which includes both human subjects and animals.

The images have differing dimensions, so algorithms are tested on their ability to handle different image sizes and aspect ratios. The dataset contains images of the following:

- A portrait of a baby
- A butterfly
- A bird
- The head of a woman
- A plant or flower.

The low-resolution versions of this dataset are usually generated by applying a down-sampling technique to the original high-resolution images.

This dataset was chosen due to its relatively small size while containing a diverse set of images. In [24], they found that using the 91-image dataset, compared to a larger dataset, does not affect performance significantly for smaller networks such as the ones used in this dissertation.

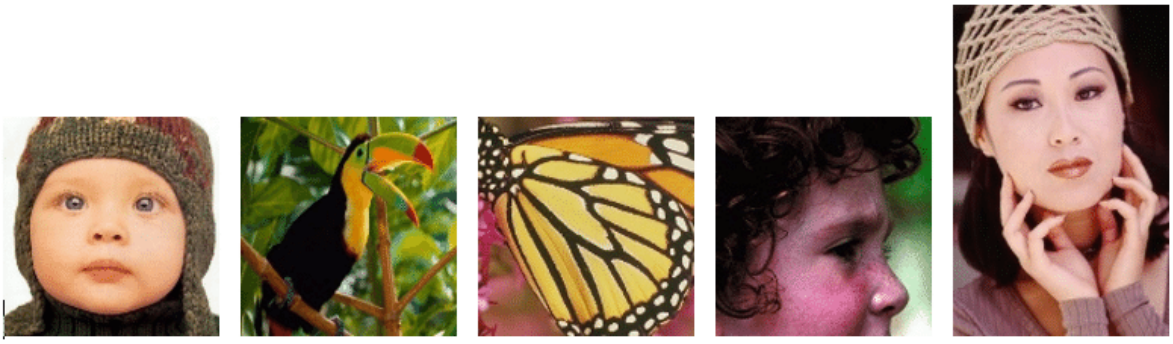

Figure 5.3: The five images from the Set5 dataset [70].

The Set5 dataset is publicly available and can be accessed from a GitHub repository [71] or other platforms that host super-resolution benchmark datasets like HuggingFace.

5.3 Experimental Design

The experiments were designed to evaluate the impact the deconvolution layer has on the performance of the two network architectures presented above. We also aim to understand the impact of different design strategies (for the deconvolution layer) and loss functions on the network's performance.

5.3.1 Experimental Setup

Network: The two network architectures (SRDeNN and ESPDeN) were trained and evaluated against their baselines.

Optimiser: The AdamW optimiser was used with a learning rate of 10^{-3} for the first two convolutional/deconvolution layers and 10^{-4} for the last layer. We chose this learning rate since it is used to train the SRCNN [24] and the ESPCN [44]. AdamW was chosen due to its broad adoption in state-of-the-art training processes [72].

Batch Size: The batch size for these experiments was set to 16 because it was the same size used in both the original SRCNN and ESPCN papers.

Loss Functions: The networks were trained on three different loss functions, namely Mean Absolute Error loss, Mean Square Error loss, and Discrete Cosine Transform loss.

Epochs: Each experiment ran a total of 400 epochs over the entire dataset.

Datasets: Patches extracted from the 91-image dataset were used to train the networks. The patches were of size 32 by 32 and were extracted using a stride of 14, thus giving a total of 24800 different training samples. The Set5 dataset was chosen as the validation dataset that was used to evaluate the model after every epoch.

Number of runs: Each experiment was run three times to ensure the reliability and validity of the results generated by the experiments. Results are then reported as a mean.

Implementation: The experiments were implemented using PyTorch [73].

Evaluation Metrics: To evaluate the networks we use Peak Signal-to-Noise Ratio. This is a common evaluation metric used for assessing image quality. It is defined as the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. A higher PSNR value indicates better image quality. The formula used to compute the PSNR is presented below. Note that in this formula MAX_I^2 refers to the maximum value a pixel can be.

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}} \right). \tag{5.1}$$

5.3.2 Testing Deconvolution Design Strategies

On top of the network settings above, the different design choices that were previously stated will be tested exhaustively. The impact of the following strategies will be tested:

- **Four-factor:** The impact of a four-factor deconvolution will be tested. This experiment will answer whether having a layer that ensures spatial coherence is beneficial when learning feature maps for the SISR.
- **Padding:** In this experiments, we investigate the impact that padding and cropping the image, before and after the deconvolution, has on model performance. We predict that padding increases the FFT resolution and can lead to more accurate reconstruction as well as reduce the effects of circular convolution.
- **First Element Trainability:** We investigate the impact of setting the first element of each filter to trainable. Setting the element to trainable could allow the network to learn more intricate mappings and improve reconstruction. However, it

could potentially lead to instability.

- **Multiple Runs:** Finally, each experiment will be run three times to assess the consistency of the results.

5.3.3 Experimental Scenarios

To summarise the previous two sections, the following experiments will be conducted:

- **Two Network Architectures:** The two different network architectures, namely SRDeNN and the ESPCN, are tested against their baselines.
- **Deconvolution Design Strategies:** On top of testing the two architectures against their baselines, the deconvolution design strategies will be tested systematically in section 5.3.2.
- **Hyperparameter Sensitivity:** Here, we test the deconvolution layer's sensitivity to hyperparameters such as the number of filters and filter size. Additionally, the impact of different loss functions will be tested on the best-performing networks from the experiments above.

5.4 Results and Discussion

In this section, we present results obtained from experiments conducted on single image super resolution (SISR). We present the experimental results while linking them to our hypothesis and also drawing conclusions from our findings.

Section 5.4.1 presents the results of the experiment ran while integrating the deconvolution layer into the SRCNN architecture. In this section, we train all networks using patches extracted from the 91-image dataset and evaluate these networks on the Set5 dataset with the PSNR metric. This section investigates explicitly the impact of the deconvolution layer and its various deconvolution design strategies on the network performance in an SISR task. We hypothesise that the deconvolution layer and each of its design strategies will increase the SISR network performance. This is reasonable to assume because the deconvolution layer uses the deconvolution operation which is more suitable for an inversion process such as SISR. Furthermore, the deconvolution layer's

ability to learn long-range correlations will help it take into account more pixels when reconstructing the high-resolution images. Additionally, the design strategies are there to increase performance or ensure stability.

Section 5.4.2 presents the findings of experiments on the ESPDeN while comparing it to the ESPCN at every step. Similar to the previous section, the networks in this section are trained on the 91-image dataset and evaluated on the Set5 dataset. We investigate the impact that the deconvolution layer has when integrated into the ESPCN architecture while testing its sensitivity to hyperparameters, such as the number of filters, filter size, and choice of loss function. The aim of these experiments is to gain a better understanding of how different hyperparameters affect the deconvolution layer in SISR.

5.4.1 SRCNN

This section presents experiments conducted on SISR with the SRCNN architecture first introduced in the paper [24]. We alter this network by replacing the first convolutional layer with a deconvolution layer. We will refer to the altered network as the SRDeNN. The original network, SRCNN, will be used as our baseline while we test the SRDeNN using different deconvolution design strategies.

First Element Trainable and Four-Factor Deconvolution

Our first set of experiments was conducted to investigate the impact of two different design strategies, namely, the four-factor deconvolution and setting the first element to trainable. We systematically turn these two strategies on and off to not only test the impact that they have on the network but also to test the impact they have on each other.

By setting the first element as trainable, we allow the optimisers to alter the weight of the first element for each deconvolution filter. This was disabled initially due to stability concerns and additionally acts as a form of regularisation, which might help the network generalise better to unseen data.

With regard to the four-factor deconvolution, we expect that adding a four-factor deconvolution will implicitly increase the effective size of the filter. This allows the deconvolution layer to capture more pixels while computing the feature maps.

Furthermore, the four-factor deconvolution allows the spatial coherence between the input and output to be retained since the effective impulse response is symmetric.

Lastly, we compare the SRDeNN's performance against a CNN by training the SRCNN using the same parameters. We predict that the SRDeNN will perform better than the SRCNN due to the SRDeNN's ability to learn longer-range filters.

Note that FET refers to the First Element Trainable design strategy and FF refers to the Four-Factor design strategy.

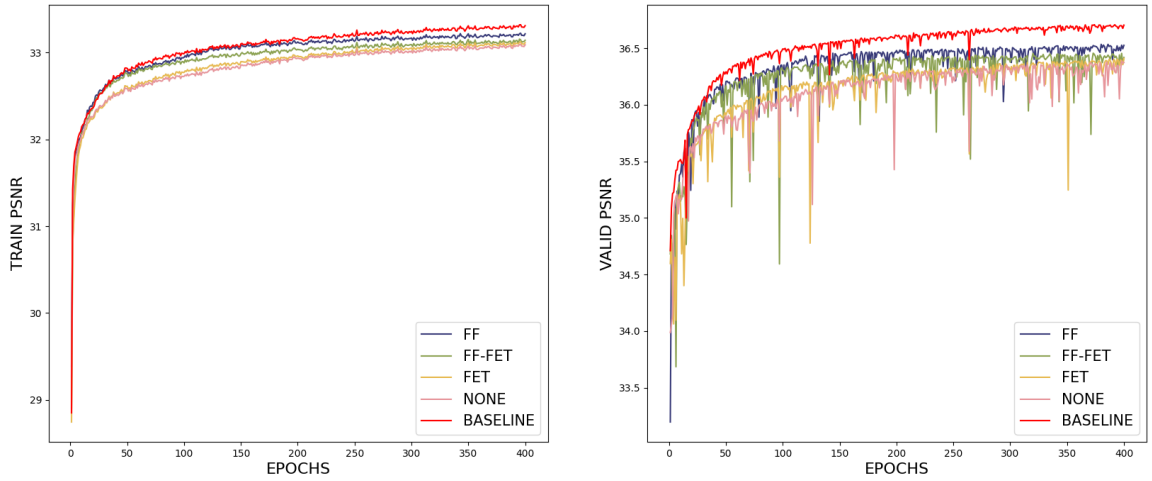

Figure 5.4: Training and validation PSNR of SRDeNN using different combinations of the two design strategies.

Table 5.1: The best training and validation PSNR for the ESPDeNs trained using different design strategies.

Network Name	**Best Training PSNR**	**Best Validation PSNR**
FF	33.22 dB	36.54 dB
FF-FET	33.16 dB	36.46 dB
FET	33.12 dB	36.41 dB
NONE	33.09 dB	36.39 dB
BASELINE	33.32 dB	36.70 dB

Figure 5.4 and table 5.1 show how the network's performance is impacted while the design strategies are enabled or disabled. The SRCNN, which is the baseline for this experiment, is also included in this plot so that the deconvolution layer's performance can be compared to a similar CNN architecture.

Based on the findings in figure 5.4 and table 5.1, we see that setting the first element as trainable only slightly increases network performance when there is no four-factor deconvolution. However, it is essential to note that this is not the case when it's included

with a four-factor deconvolution. This can be explained as follows: setting the first element trainable allows for a smoother loss landscape (due to the operation being more stable) and thus makes the learning process easier for the optimiser.

In relation to the four-factor deconvolution, figure 5.4 and table 5.1 show that this design strategy also slightly increases the reconstruction accuracy of the network. This can be attributed to four-factor deconvolution, expanding the effective filter size while ensuring spatial coherence between input and output.

Moreover, figure 5.4 and table 5.1 show that by replacing a convolutional layer with a deconvolution layer, the network suffers and yields a lower PSNR on both the training and validation set. This could be attributed to the CNN's ability to learn hierarchical feature maps, which is more suited to learning mappings for SISR.

Overall, it seems that the deconvolution design strategies does not impact layer performance. Additionally, the difference in PSNRs could also be attributed to the variation in training from randomising the dataset and parameters. Lastly, one important thing to note is that the PSNRs on the validation set is higher than the PSNRs on the test set. This can be attributed to the size of the training dataset (24800 samples) and the validation set which consists of 5 samples. The network is quite small compared to the dataset and therefore cannot overfit to the training data which causes the training PSNR to be lower than the validation PSNR.

Padding

The last deconvolution design strategy that we investigate is padding. This strategy will help reduce the effects of circular convolution mentioned in chapter 4 and will increase the size of the Fourier representation of the image, which results in the frequency components in the image being captured with better accuracy. This also adds a buffer around the edges of the image, which will help alleviate the circular convolution problem. We expect to see padding improve the network's ability to reconstruct high-resolution images. However, the network will require more computational resources to train and reconstruct high-resolution images.

Figure 5.5 shows the PSNR of each network when the image is padded during the deconvolution operation. Results for three different networks are plotted, namely, the SRCNN (baseline), an SRDeNN with padding (padding=50%), and an SRDeNN without padding. Setting padding equal to 50% refers to padding zeroes equal to 50% of the

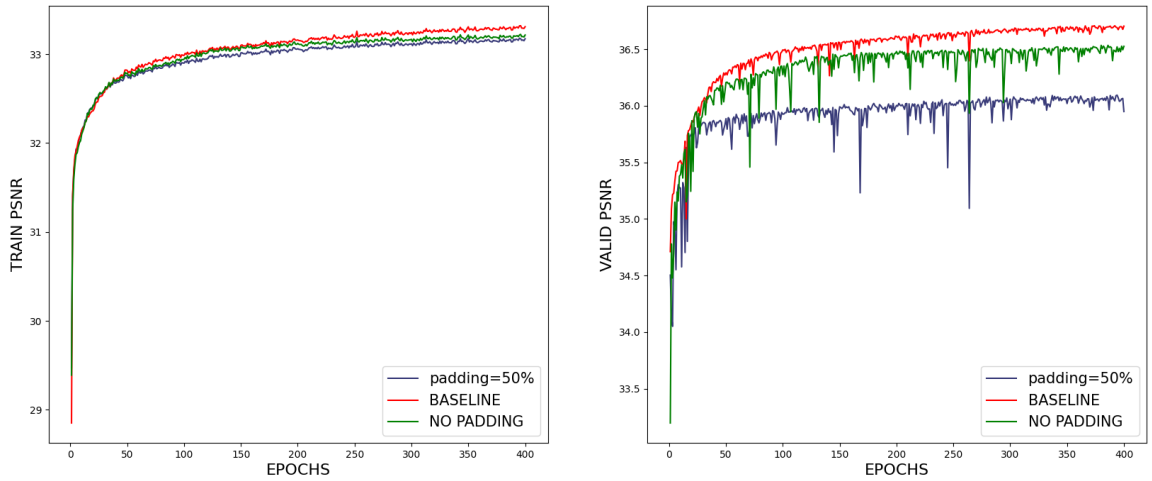

Figure 5.5: Training and validation PSNR of SRDeNN when padding is applied before and after the deconvolution.

image's dimensions on each side. Note that for the SRDeNN, we have enabled the four-factor deconvolution as the only design strategy. The plot clearly shows that padding degrades the performance of the network on both the training and validation sets. This disproves our hypothesis that padding will result in a richer Fourier representation and, therefore, more accurate feature maps. This result proves that padding does not positively impact the network's ability to learn low to high-resolution mappings.

5.4.2 ESPCN

In this section, we explore the impact of the deconvolution layer on SISR using the ESPCN network architecture, which was first introduced in [44]. This network has a similar architecture to the SRCNN except it uses a Tanh activation instead of ReLU. Another difference is that the low-resolution image is not upscaled before it is passed into the network. The computation resources and time are reduced since the network is processing a smaller image. Similar to the previous network, we alter this network by replacing the first convolutional layer with a deconvolution layer. However, we do not explore the impact of different layer configurations but focus on the impact of the deconvolution layer and its sensitivity to hyperparameter changes. We also aim to investigate if this architectural change from SRCNN to ESPCN affects the findings when comparing the deconvolution and convolutional layers. Lastly, we use the ESPCN as a baseline for this section.

Number of filters

In this section we explore the impact of varying the number of filters in the deconvolution layer. We hypothesise that the network performance will increase as the number of filters inside the deconvolution layer increases. We also predict that we will see diminishing returns as the filter count is increased, which can be attributed to the limitations of the architecture and bottlenecking in subsequent layers.

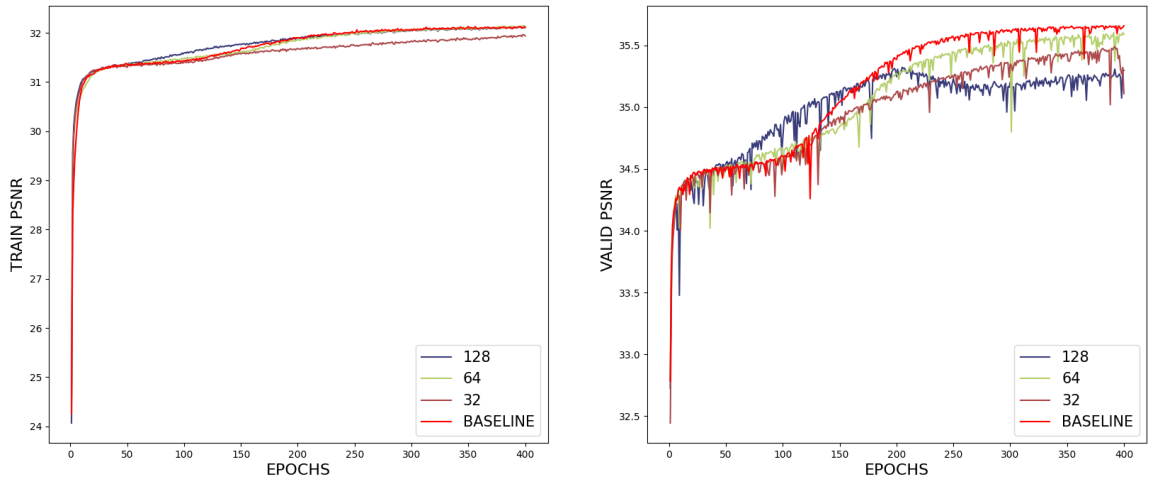

Figure 5.6: Training and validation PSNR of ESPDeN when the deconvolution layer's number of filters is varied.

Figure 5.6 shows the PSNR of the ESPDeN on both the training and the validation sets when the number of filters is varied. The baseline is also plotted to compare the DeNN against a CNN. Based on the PSNR for both the training and the validation set, we see that using the ESPCN achieves better performance than all the ESPDeN configurations. This shows that the convolutional layer is still the better option for a SISR task.

When we compare the different ESPDeN configurations, we see that increasing the number of filters past 64 does not yield better results, which indicates that our prediction is incorrect. If we look more closely at the 128 filter network, we see that the validation loss degrades as the network trains past 200 epochs. This indicates that the network is overfitting on the training data.

Filter size

In this section, we explore the impact that different filter sizes have on the network's ability to reconstruct high-resolution images for SISR. We predict that as the filter size

increases, so will the network's performance. We know that the number of learnable parameters increases as the filter size increases and therefore we can assume that this will enable the network to learn more intricate mappings from low to high-resolution images.

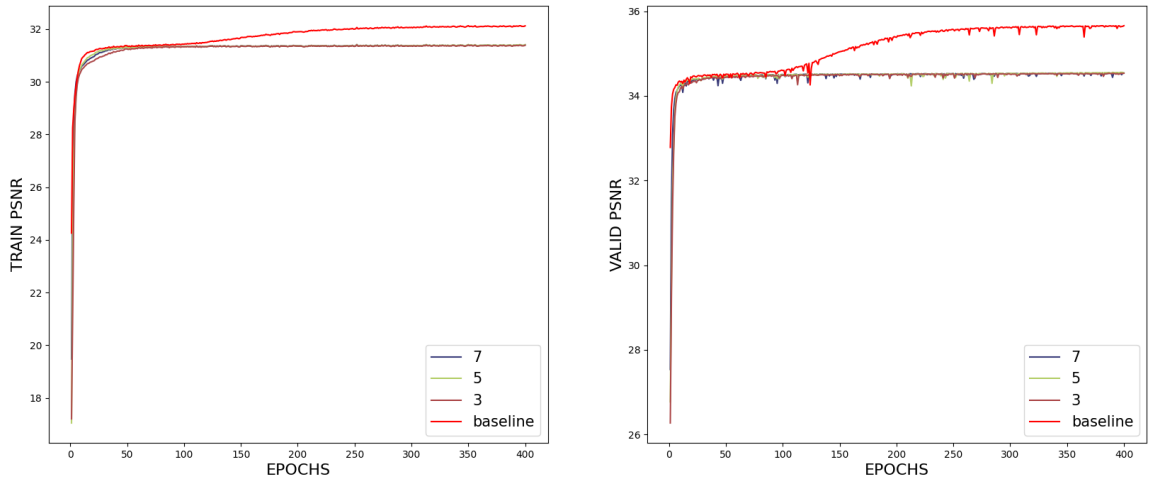

Figure 5.7: Training and validation PSNR of ESPDeN when the deconvolution layer's filter size is varied.

Figure 5.7 shows the PSNR of the ESPDeN as the filter size of the deconvolution layer is varied. The baseline is plotted as well. Based on the results in the figure, we can conclude that our prediction is incorrect as the filter size does not impact the network's PSNR and, therefore, the network's accuracy. This could be attributed to the fact that a 3×3 filter has sufficient parameters to learn and capture the features of the low-resolution images.

Loss functions

Next, we explore how different loss functions affect a DeNN's performance. We test three different loss functions, namely, Mean Square Error (MSE), Mean Absolute Error (referred to as L1 Loss), and Discrete Cosine Transform (DCT) Loss. We expect the network trained on the DCT loss to achieve the best performance because it is a frequency-based loss function like the operation in the deconvolution layer. Additionally, we postulate that the L1 loss will achieve a higher PSNR than the MSE loss despite MSE loss being the core components of the PSNR calculation.

Figure 5.8 shows the plots of the PSNR of the ESPDeN for different loss functions. Table 5.2 shows the best training and validation PSNR when the ESPDeN is trained using different loss functions. We can see that the DCT and MSE loss achieves the best

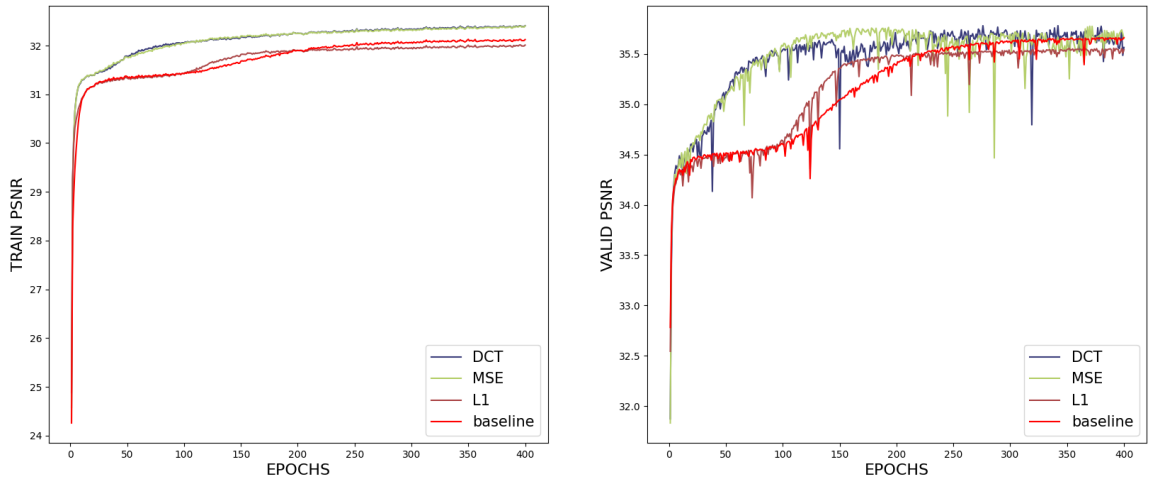

Figure 5.8: Training and validation PSNR of ESPDeN for different loss functions.

Table 5.2: The best training and validation PSNR for the ESPDeNs trained using different loss functions.

Network Name	**Best Training PSNR**	**Best Validation PSNR**
DCT	32.41 dB	35.78 dB
MSE	32.41 dB	35.77 dB
L1	32.01 dB	35.56 dB
BASELINE	32.13 dB	35.66 dB

PSNR on the training set. In the validation PSNR plot, we see the baseline ESPCN trained using the L1 loss achieves similar results to the ESPDeN trained on MSE and DCT in the last 100 epochs. This showcases the ESPCN's ability to generalise well to unseen data. However, it also shows the ESPDeN's susceptibility to overfit training data. Nevertheless, our assumption is incorrect since the frequency-based loss function does not improve the accuracy of an ESPDeN. The network trained on the frequency-based loss function achieves the same PSNR as a pixel-based loss function (MSE). It is important to note that the DCT achieves its best PSNR after 200 epochs, which is half the amount that the pixel-based loss functions to reach their best PSNR. This shows that a frequency-based loss function provides an easier loss landscape for the optimiser to navigate. This allows the optimiser to get to an optimal position faster. Additionally, the network trained using MSE loss achieves better performance than the one trained using L1 loss. This shows that the MSE loss is a more suitable loss function for training DeNNs on SISR. Lastly, it is important to note that the change in performance is quite minuscule and this change could be due to variations in the training set up which are caused by the randomisation of the dataset and the network's random initialisation.

5.5 Conclusion

In this chapter, we investigated the performance of the deconvolution layer on single image super resolution (SISR). We tested the performance of the layer on two similar network architectures, namely the effective sub-pixel CNN (ESPCN) and the super resolution CNN (SRCNN). We tested each network with different configurations of both the network and training setup. We trained the networks using patches extracted from the 91-image dataset. We compared the SISR networks using the Peak-Signal-to-Noise Ratio (PSNR) metric.

First, we looked at the deconvolution performance inside the SRCNN. We did this by replacing the first convolutional layer with a deconvolution layer. We refer to this network as the SRDeNN. Our hypothesis stated that the deconvolution layer and each of its design strategies will increase the network's performance in SISR. Based on the results from these experiments we can conclude that the deconvolution layer does not increase the network's performance on SISR. This can be attributed to the CNN's robustness and ability to learn hierarchical features of the low-resolution images. Furthermore, design strategies such as four-factor deconvolution and first element trainability supports our hypothesis because DeNN performance increases when they're added. However, padding does not support our hypothesis since it decreases the DeNN's performance.

Next, we investigated the deconvolution layer's performance inside the ESPCN. The architecture was altered in the same way as the previous section, and we refer to this architecture as the ESPDeN. In these experiments, we aimed to see how different hyperparameters affected the deconvolution layer's performance. We saw that increasing the number of filters positively impacted the DeNN's accuracy on the training data. However, the additional filters caused the DeNN to overfit the training data. We found that filter size did not impact the DeNN's performance but loss function choice did. We found that the ESPDeNs trained using MSE or DCT loss achieved the best result. Additionally, we found that the network trained on DCT loss reached its peak accuracy halfway through training whereas the network trained on MSE loss, a pixel-based loss, took twice as long to reach the same accuracy. This shows the potential of using the frequency-based loss function for the deconvolution layer in SISR.

In summary, our investigation into the integration of the deconvolution layer within SISR architectures, specifically the SRCNN and ESPCN, has presented some insightful findings. While the introduction of various deconvolution design strategies, such as the four-factor deconvolution and padding adjustments, showed potential in enhancing the

performance of the altered network architectures, they were unable to surpass the original architectures in terms of PSNR. This showcases the robustness of the original SRCNN and ESPCN designs for SISR tasks. Furthermore, our experiments highlight the importance of hyperparameter tuning and the choice of loss function when integrating new layers into existing architectures. We found that the ESPDeN network achieves better performance than the ESPCN when using the MSE or DCT loss functions. This shows that the DeNN can beat a CNN in SISR, given that it has optimal training settings.

Chapter 6

Image Classification

This chapter presents the work on image classification using a deconvolution neural network(DeNN). Image classification is a fundamental task in computer vision and is one of the most popular tasks used to benchmark CNNs. It has advanced and optimised various applications like medical imaging, autonomous vehicles [74], and social media content filtering [75]. In image classification, network performance is dependent on the network's ability to learn features from the image. Despite its popularity, it remains a challenging problem to solve due to changes in perspective, lighting, and occlusions. In the last decade, CNNs have become a popular method for solving image classification problems [10] [76] [77]. This is due to their ability to learn hierarchical representations of the image data. However, recent research has shown that increasing the network depth or width could lead to worse performance [78]. This phenomenon could be due to reasons such as overfitting when a network consists of hundreds of layers without regularisation. Another issue that excessively Deep CNNs face is the exploding or vanishing gradient problem [79] [80].

Section 6.1 presents the different networks and their baselines that were tested on image classification. This section details the LeNet-5 architecture (used as the baseline) and how the deconvolution layer is integrated into this architecture.

Section 6.2 details the dataset that is used to train and evaluate the image classification networks. The dataset is called CIFAR-10 and contains 32×32 RGB images of 10 different classses.

Section 6.3 elaborates on the details of the experiments that were conducted on the image classification networks. First, it dives into the experiment setup and network

initialisation while giving details of the hyperparameters. Next, the section discusses experimental scenarios and the different design strategies of the deconvolution layer that were tested.

Section 6.4 presents and discusses the results from the experiment outlined in the earlier sections. We found that the deconvolution layer does improve the LeNet-5 architecture's performance on an image classification task, although this does come at the cost of a slight increase in computational demand. We also investigated the layer's sensitivity to various combinations of design strategies and hyperparameter changes.

Finally, we conclude the work done on image classification by summarising the results found in the previous section.

6.1 Algorithms and Networks

This section presents the proposed CNN architecture along with its baselines.

The proposed architecture is an adaptation of the LeNet-5 architecture, which was originally presented in [45]. The LeNet-5 has a straightforward architecture and consists of 3 convolution blocks followed by two fully connected layers. The convolution blocks consist of a convolutional layer with a Tanh activation function to introduce non-linearity, followed by an average pooling layer. The average pooling layer is for regularisation so that the network can generalise better to unseen data while increasing computational efficiency.

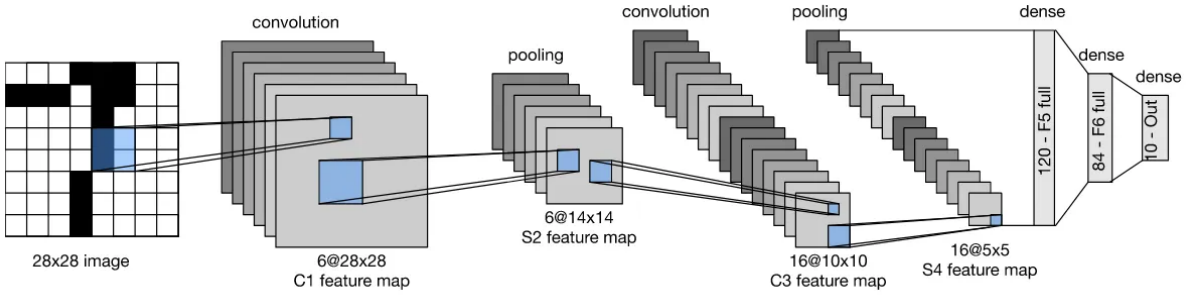

Figure 6.1: High-level diagram of the LeNet-5 architecture [81].

The details of the layers inside this architecture are as follows.

- **Convolution Block 1:** The convolutional layer in the first convolution block uses a

5 by 5 ($f_1 = 5$) kernel with 6 different filters ($n_1 = 6$) followed by a Tanh activation. The average pooling layer in this block has a filter size of 2 and stride of 2.

- **Convolution Block 2:** The convolutional layer in the second convolution block uses a 5 by 5 ($f_2 = 5$) kernel with 16 different filters ($n_2 = 16$) followed by a Tanh activation. The average pooling layer in this block has a filter size of 2 and stride of 2.
- **Convolution Block 3:** The convolutional layer in the final convolution block uses a 5 by 5 ($f_3 = 5$) kernel with 120 different filters ($n_3 = 120$) followed by a Tanh activation.
- **Fully Connected Layer 1:** The first fully connected layer has 84 different neurons followed by a Tanh activation function.
- **Fully Connected Layer 2:** The second fully connected layer, and the last layer of the network, has 10 different neurons and is followed by a softmax activation function that transforms the logits received by the last fully connected layer to probabilities.

6.2 Dataset

In this section, we present the dataset that is used to train and evaluate the CNNs and DeNNs on image classification. The dataset that was chosen for this deep learning task is the CIFAR-10 dataset. The dataset is split into two subsets. One is used for training while the other is used to evaluate the network's accuracy after every epoch.

6.2.1 CIFAR-10 Dataset

The CIFAR-10 dataset is a popular dataset used for training and evaluating algorithms used for image classification. This dataset was introduced by Krizhensky in 2009 [82] and has become a widely used dataset for the evaluation and training of deep learning image classification networks.

CIFAR-10 consist of 60,000 RGB images of size 32×32. The dataset is split into two different sets, namely the training set, which contains 50,000 images, and the test set, which contains the other 10,000 images. The images are split into 10 different classes, each

class having 6,000 images. The classes for this dataset include aeroplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

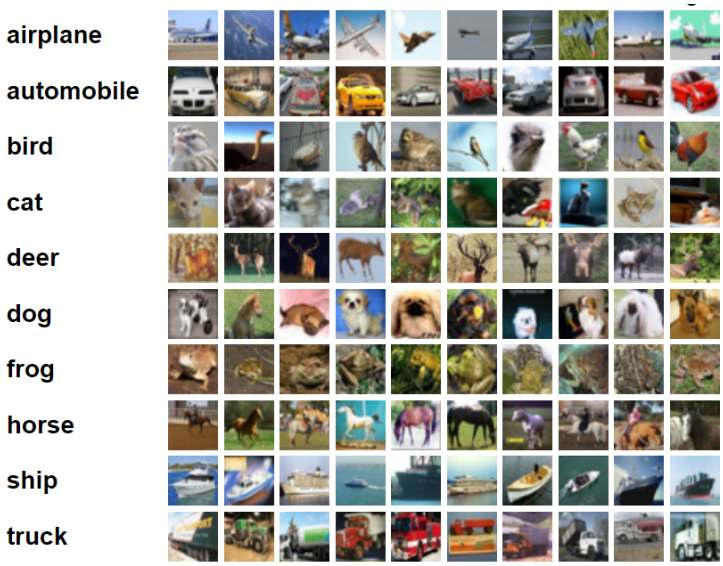

Figure 6.2: Examples of the different images and classes that are found in the CIFAR-10 dataset [83].

The CIFAR-10 dataset is predominantly used for the following:

- **Image Classification**: Evaluation and comparison of image classification algorithms, mainly deep neural networks.
- **Model Development**: The size and complexity of the CIFAR-10 dataset makes it ideal for model development. The relatively small dataset size and image size allow for models to be trained faster so that developers can employ rapid prototyping to find the best model. The dataset is still complex due to it containing a wide range of different objects, which makes the classification problem non-trivial.
- **Transfer Learning**: Models that have been pre-trained on a CIFAR-10 dataset can be used as a starting point for more complex tasks or datasets.

The CIFAR-10 dataset has been vital in the development of deep-learning models for image classification. Its popularity stems from its balanced complexity. The datasets are challenging enough to stimulate innovation while still being simple enough for quick experiments and validation.

The dataset is publicly accessible through the official CIFAR website [83], but it can also be accessed natively in PyTorch and TensorFlow.

6.3 Experimental Design

This section pertains to the details of the experiments run for the image classification problem. The experiments were designed to evaluate the impact the deconvolution layer has on the performance of the adapted LeNet-5 architecture proposed above. We also aim to estimate the different design strategies (for the deconvolution layer) as well as replace multiple convolutional layers with deconvolution layers.

6.3.1 Experiment Setup

Network: The network to test the deconvolution layer's performance on image classification tasks is the adapted LeNet-5 architecture mentioned above. For the first network, we replace only the first convolutional layer with a deconvolution layer. In the second network, we replace the first and second convolutional layers with deconvolution layers.

Optimiser: The Adam optimiser was used with a learning rate of 10^{-3} for all the layers [63]. Different optimisers and learning rates are investigated in section 6.4.3.

Batch Size: The batch size for these experiments was set to 32. Different batch sizes are investigated in section 6.4.3.

Loss Functions: The networks were trained using the Cross-Entropy Loss since this is the standard loss for multi-class classification.

Epochs: Each experiment ran a total of 10 epochs over the entire dataset.

Datasets: The CIFAR-10 dataset was used to train and evaluate network performance on image classification. The dataset consists of 60,000 images of 10 different classes. Specifically, 50,000 images were used to train the network, and the other 10,000 images were used to evaluate the network after every epoch. We chose CIFAR-10 for its balanced complexity: it is challenging while still simple enough for quick experimentation and validation.

Number of runs: Each experiment was run three times to ensure the reliability and validity of the results generated by the experiments. Results are then reported as a mean (and standard deviation where suitable).

Implementation: The experiments were implemented using PyTorch [73].

Evaluation Metrics: The metric that was used to evaluate the performance of the network is the prediction accuracy. This is calculated by dividing the number of correct predictions by the number of total predictions. It is computed for both the training and validation set separately after every epoch.

6.3.2 Testing Deconvolution Design Choices

On top of the network settings above, the different design choices that were previously stated are tested exhaustively. The following strategies are covered:

- **Four-factor:** The impact of a four-factor deconvolution will be tested. This experiment investigates whether having a layer that ensures spatial consistency is beneficial when learning feature maps for image classification. It might also answer whether having a filter kernel that covers more data inside the image would be beneficial to an image classification problem.
- **First Element Trainability:** We investigate the impact of setting the first element of each filter to trainable. Setting the element to trainable could allow the network to learn more intricate mappings of inputs to feature maps. However, it could lead to instability.
- **Filter Bias:** We investigate the effects that the bias term has on the DeNN's performance.
- **Multiple Runs:** Finally, each experiment will be run 3 times to assess the consistency of the results. The mean and standard deviation, where applicable, of the performance metrics will be presented.

6.3.3 Experimental Scenarios

To summarise the previous two sections, namely sections 6.3.1 and 6.3.2, the following experiments are conducted:

- **Convolution Layer Replacement:** In this scenario, we take the LeNet-5 architecture and systematically replace layers one by one to see how the

deconvolution layer impacts performance.

- **Deconvolution Design Strategies:** In this scenario, the networks are also trained using every possible permutation of the design strategies presented in section 6.3.2.
- **Hyperparameter Sensitivity:** In section 6.4.3, we explore the DeNN's sensitivity to changes in the hyperparameters.
- **Layer Response:** Next, we look at the deconvolution layer's response to an impulse and image input.
- **Computational Resources:** Lastly, we test the DeNN's computational demand by looking at GPU memory usage and training time. We compare these findings to a CNN's computational resource usage.

6.4 Results and Discussion

In this section, we present the results from the experiments on image classification in a cohesive manner that shows links between experiments and the conclusions drawn from the results. Note that all networks in this section were trained and evaluated using the CIFAR-10 dataset. We use cross-entropy as the loss function and evaluate performance based on the network's accuracy on both the training and validation data subsets.

Section 6.4.1 investigates the deconvolution layer's impact on image classification. This is done by systematically replacing each convolutional layer with a deconvolution layer. We evaluate every architecture variation to find the best configuration.

Section 6.4.2 presents the results of the experiments where the deconvolution design strategies were varied. In this section, we evaluate every possible configuration of design strategies to find which configuration achieves the best result in an image classification task.

Similar to the experiments on SISR, we hypothesise that adding deconvolution layers to a CNN architecture will increase the accuracy of the network on image classification. Furthermore, our second hypothesis states that the design strategies will also increase performance but may lead to overfitting of the training data.

Section 6.4.3 examines the deconvolution layer's sensitivity to changes in hyperparameters. In this section, we vary hyperparameters such as the number

of filters, filter size, batch size, learning rate, and optimiser choice so that we can understand which hyperparameters affect the deconvolution layer's performance in a significant way. In these experiments, the aim is to understand how different hyperparameters affect the deconvolution layer.

Section 6.4.4 presents the deconvolution layer's response to an impulse and image input. We pass both an image and an impulse into the network as the input and present the response of each filter. Additionally, we provide a rationale as to why the layer has responded in this way.

Finally, in section 6.4.5 we compare the DeNN and CNN computational demand using metrics such as training time and GPU memory usage (VRAM usage). On top of this comparison, we also evaluate the computational resources that various design strategies use.

6.4.1 Convolutional Layer Replacement

In this section, we present the results found in the experiments where the LeNet-5 architecture is altered by replacing convolutional layers with deconvolution layers. Note that when we refer to network configurations such as D-C-C, the D indicates that the first layer is a deconvolution layer. Subsequently, the two Cs indicate convolutional layers for both the second and third layers.

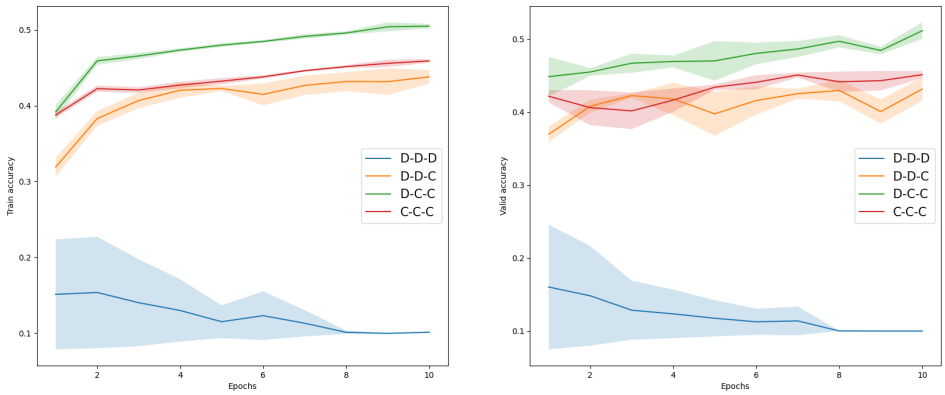

Figure 6.3: Training and validation accuracy for image classification experiments where convolutional layers are replaced.

Figure 6.3 illustrates the accuracy of networks with different layer compositions on both the training and validation datasets during the training process. We can see that the

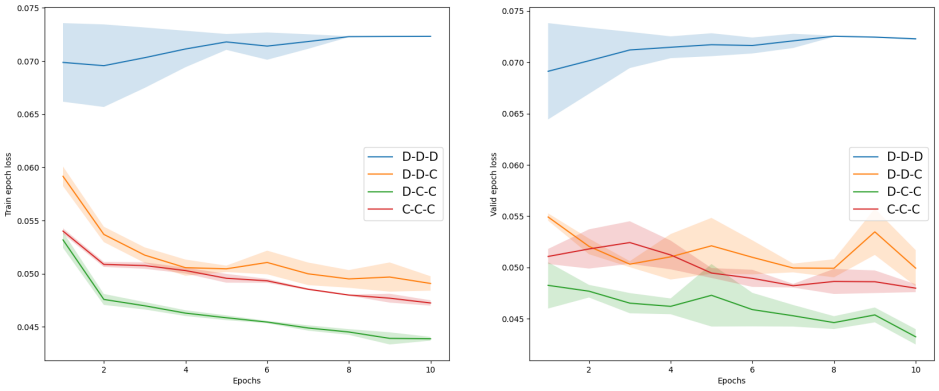

Figure 6.4: Training and validation loss for image classification experiments where convolutional layers are replaced.

D-C-C composition achieves the best accuracy for both the training and validation set. It beats the original LeNet-5 network by 5%, which confirms that the deconvolution layer is quite powerful for image classification. Moreover, the deconvolution neural networks (DeNNs) manage to achieve better accuracy on the validation set when compared to the training set. This indicates that the deconvolution layer is able to generalise well to unseen data and does not over-fit the training data.

Similar to figure 6.3, figure 6.4 shows that the deconvolution layer is able to optimise the loss to a lower value. This indicates that it is easier to optimise when incorporating a combination of both convolutional and deconvolution layers.

An important observation to make is how the accuracy plot and the loss curve vary. This can be attributed to factors such as random weight initialisation and random sampling of the training dataset.

6.4.2 Design Strategy

In this section, we present the results found in the experiments for testing the impact of different design strategies on image classification. Initially, experiments are run to investigate the impact of the following strategies: four-factor deconvolution, first element trainable, and adding a bias term to the network. We expect to see all three of these strategies improving network performance. However, setting the first element to trainable might result in the deconvolution operation being unstable, which could cause the output of the layer to explode. The network accuracy for each of the design strategies is plotted

below. Note that in this figure, B refers to Bias, FET refers to First Element Trainable, and FF refers to Four-Factor Deconvolution.

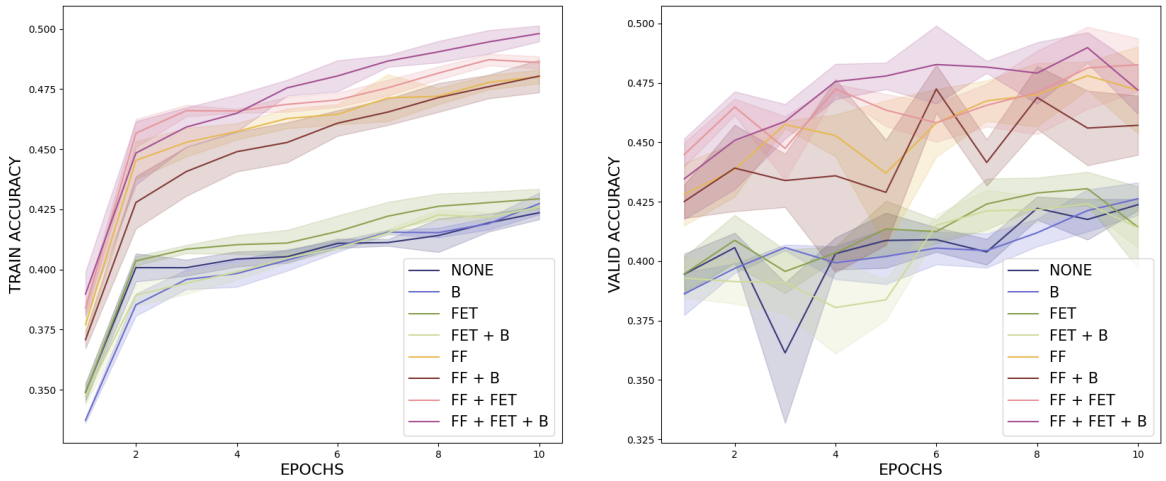

Figure 6.5: Training and validation accuracy for image classification experiments investigating the impact of design strategies.

Figure 6.5 shows the accuracy of all the design strategies during training. This result indicates that our expectation and hypothesis are correct since the best training accuracy is achieved by the network where all three of the design strategies are enabled. It is quite interesting to note that there is a clear distinction between the networks that have four-factor deconvolution enabled and those that do not. This strategy impacts the performance the most, which indicates it is quite important to the success of the deconvolution layer.

It is important to note that there are discrepancies between the training and validation accuracy. For training accuracy, the network with all three strategies enabled performs the best. However, in the validation accuracy plot, the four-factor and first-element trainable combination overtakes this configuration in the final iteration of the dataset. This shows that the former is prone to overfitting the data. Let's examine the validation accuracy plot between these two networks. We can see that the network with all strategies enabled achieves a peak validation accuracy at the 6th epoch while the training accuracy continues to improve after the epoch. This confirms that this network is overfitting the training data. To reduce overfitting, one can employ regularisation and data augmentation; however, this is out of the scope of this dissertation.

Figure 6.6 depicts a similar story to the one just presented above, where the network with all strategies enabled achieves the lowest loss in training but suffers on the validation data. As mentioned previously, there is some variation in the experiment. This can

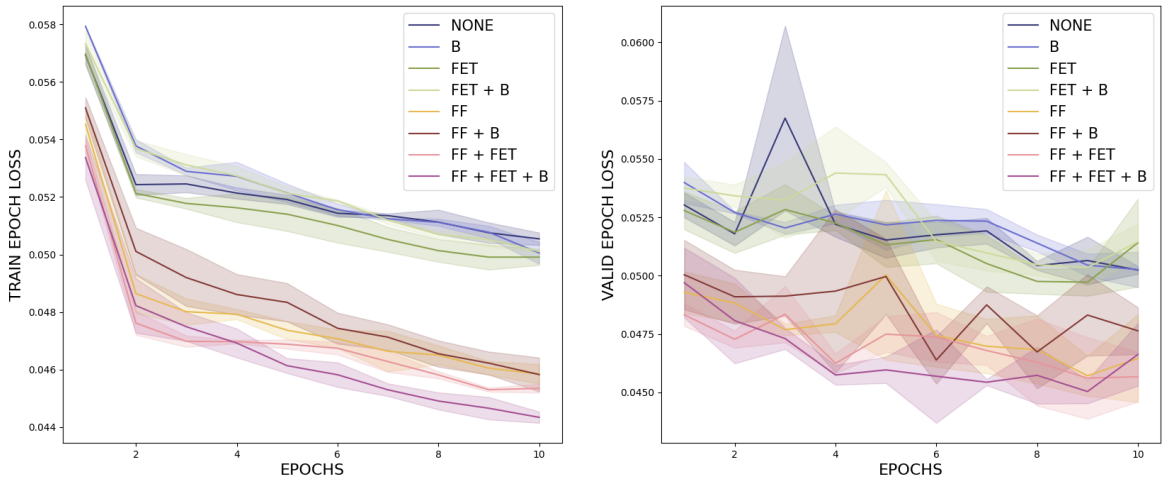

Figure 6.6: Training and validation loss for image classification experiments investigating the impact of design strategies.

be largely attributed to the random weight initialisation and random sampling of the training dataset.

6.4.3 Hyperparameter Sensitivity

Number of filters

In this section, we explore the deconvolution layer's sensitivity to the number of filters. We use a network configuration of D-C-C with four-factor deconvolution and bias enabled while the first element is set to trainable. We assume that adding more filters will improve the performance of the network. We tested the following numbers of filters: 3, 6, 12, 24. The figure below plots the accuracy of all the networks with varying filter sizes on both the training and validation sets.

Figure 6.7 confirms our assumption that increasing the number of filters improves the accuracy of the network. This can be attributed to the fact that increasing the number of filters also increases the number of trainable parameters, which enables the network to learn more intricate and complex patterns in the data. If we look more closely at the plots, we find that there is only a small increase in performance when going from 3 filters to 6 filters. However, when we look at the performance increase from 6 filters to 12 filters, we see a massive leap in performance. This could be attributed to the fact that more filters are being added when jumping from 6 to 12 than from 3 to 6. Furthermore, the network containing the 24-filter deconvolution layer does not see a large performance

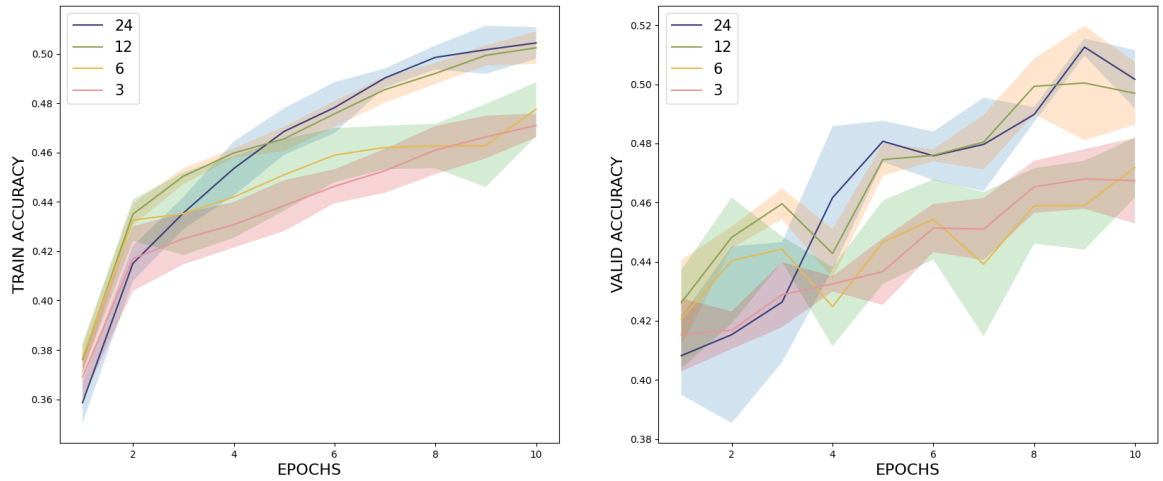

Figure 6.7: Training and validation accuracy for image classification experiments varying number of filters.

jump over the 6-filter network. This could be attributed to the following reasons: the ceiling of the current architecture has been reached, the subsequent layers in the network are bottlenecking, or the network needs to be trained further as larger networks require more steps in optimisation to converge.

Filter size

In this section, we investigate the network's sensitivity to varying filter sizes. In these experiments we investigate different filter sizes, namely, 3×3, 5×5, 7×7, and 9×9. Note that the filter size stays consistent throughout the network, so the filter sizes are equal in all three layers. We predict that a filter size of 3 will provide the best performance since it has been used in several state-of-the-art architectures such as ResNet [77] and VGG [76]. Furthermore, we expect the network's performance to increase as the filter size decreases.

Figure 6.8 plots the accuracy of the networks with varying filter sizes. Our prediction is correct since we see that a filter size of 3×3 yields the best performance. Furthermore, we see that as the filter size is decreased the network performs better.

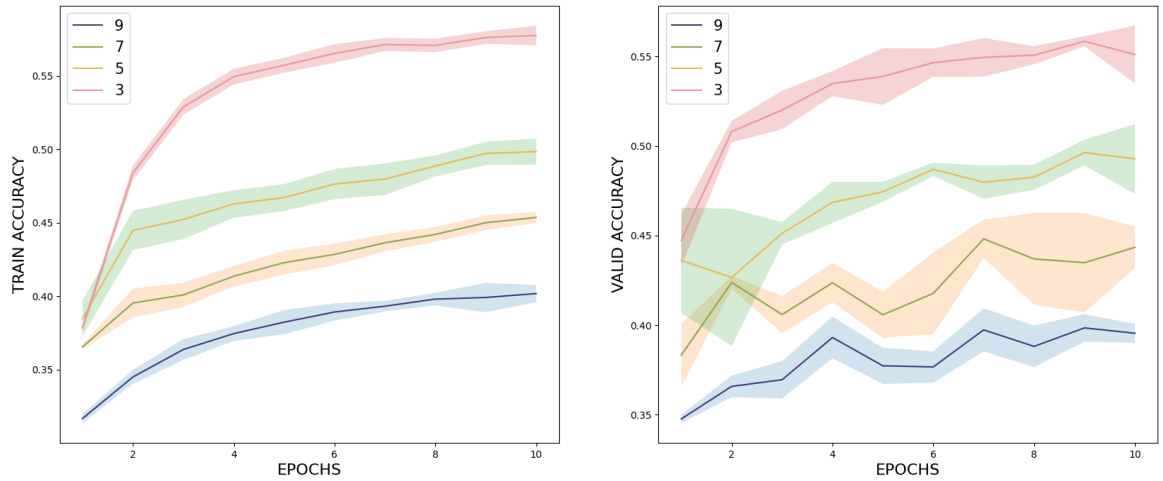

Figure 6.8: Training and validation accuracy for image classification experiments varying filter sizes.

Batch size

This section presents the results and discussion of the experiments where the batching size is varied. These experiments are conducted to find the network's sensitivity to variations in batch sizes. In this experiment, we expect to see the network performance increase as the batch size increases.

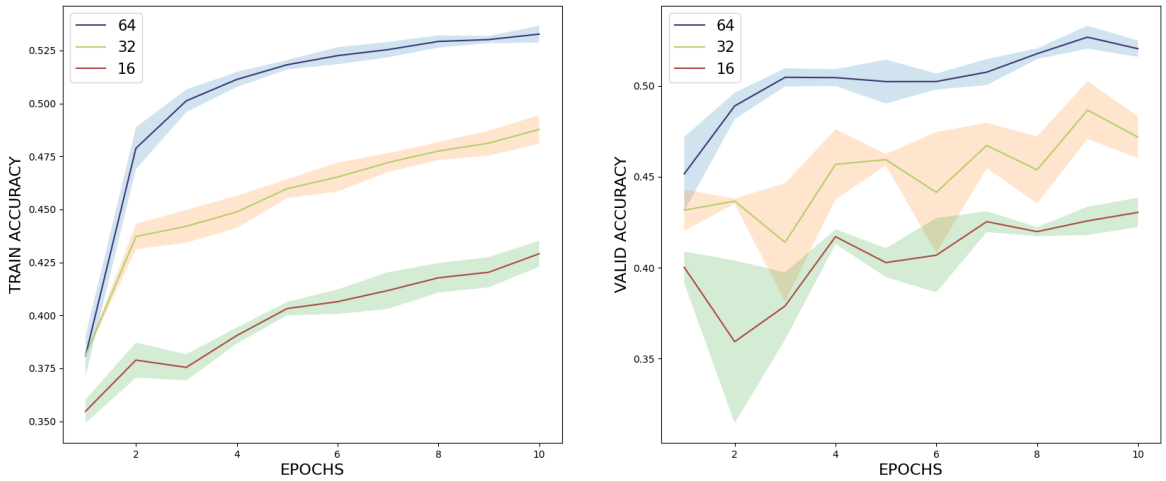

Figure 6.9: Training and validation accuracy for image classification experiments varying batch sizes.

Figure 6.9 illustrates the network's accuracy as the batch size is varied. The plot indicates that our prediction is correct since the network accuracy increases as the batch size increases. This is congruent with previous research [84] that found that increasing batch size generally leads to increased network performance, but does vary on a case-by-case

basis.

Learning Rate

In this section we investigate the effect that the learning rate has on the network's training. Originally we were using a learning rate of 1×10^{-3}. Now we will investigate the network's accuracy after training with a learning rate of 1×10^{-2} and 1×10^{-4}. We expect the larger learning rate, 1×10^{-2} to yield a faster training optimisation and, therefore, will get a higher accuracy after 10 iterations through the dataset. We also expect that using a learning rate of 1×10^{-4} will cause the network to converge a lot slower than the two larger learning rates.

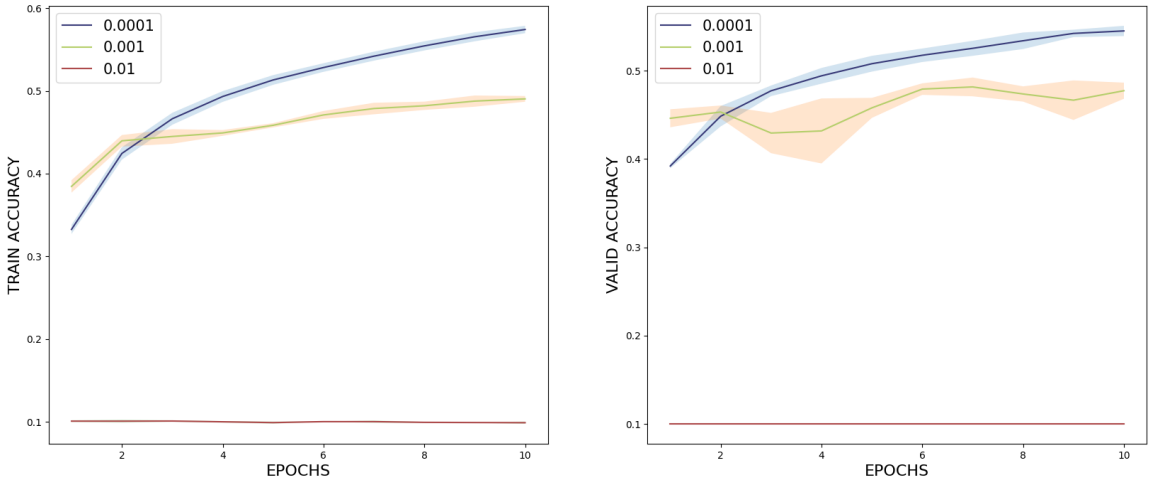

Figure 6.10: Training and validation accuracy for image classification experiments while varying the learning rate.

Figure 6.10 shows that the smallest learning rate, 1×10^{-4}, manages to achieve the highest accuracy at almost 60%. A learning rate of 1×10^{-2} results in accuracy comparable to random guessing. This indicates that the network struggles to train effectively at this high learning rate, as the optimiser's steps are excessively large, preventing loss optimisation. The learning rate of 1×10^{-4} yields the highest accuracy among the tested values.

The learning rate determines the size of the steps the optimiser takes while adjusting the network's weights during training. If the learning rate is too high, the optimiser might overshoot the optimal weights, leading to poor convergence or even divergence. Conversely, if it's too low, the training process can become exceedingly slow, and the network might get stuck in local minima. A learning rate of 1×10^{-4} appears to strike the right balance for this particular network and dataset. It's neither too large to cause

instability nor too small to slow down the training. This allows the optimiser to efficiently navigate the loss landscape and find a solution that offers the highest accuracy.

Furthermore, this result indicates that the loss landscape the deconvolution provides to the network is quite sensitive to weight changes. This means that if it had to learn values, even minor adjustments could lead to significant shifts in the network's performance, emphasising the importance of fine-tuning and careful weight initialisation for optimal results.

Optimiser

This section presents the experiments conducted by varying the optimisation algorithm. We use two different optimisation algorithms, namely Stochastic Gradient Descent (SGD) and Adaptive moment estimation (Adam). Stochastic Gradient Descent is simple as it just multiplies the learning rate by the gradients of the weights. Adam is more sophisticated as it utilises both the first and second moments of the gradients. With these characteristics, Adam will find an optimal solution. However, it is quite interesting to note that a well-tuned learning rate for SGD can often beat Adam. In these experiments we assume that Adam will optimise the network better than SGD since, in this case, we have not fine-tuned the learning rate.

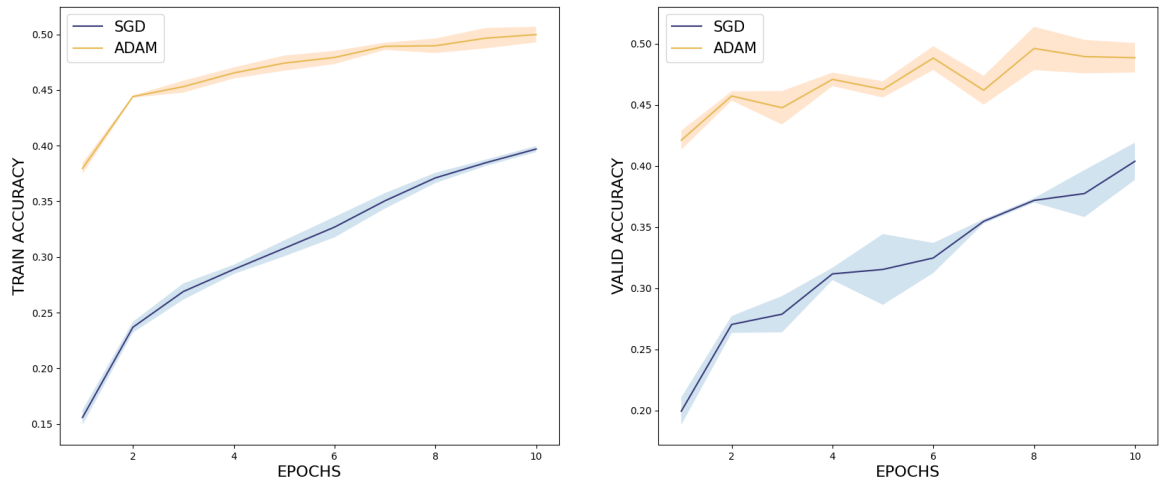

Figure 6.11: Training and validation accuracy for image classification experiments while varying the optimisation algorithm.

Figure 6.11 clearly demonstrates that the Adam optimiser outperforms Stochastic Gradient Descent (SGD) in terms of both training and validation accuracy. This result is consistent with our assumption, which predicted that Adam would be more effective

in optimising the network, especially when the learning rate is not fine-tuned.

Upon closer examination of the graph, it's evident that the network trained with Adam converges faster and achieves a higher accuracy compared to SGD. This can be attributed to Adam's momentum-based optimisation, which adjusts the learning rate for each parameter based on a weighted sum of the previous gradients. This technique helps Adam to navigate the loss landscape more efficiently and avoid potential pitfalls like saddle points or local minima that might trap simpler optimisers like SGD.

Furthermore, Adam utilises both the first and second moments of the gradients, providing it with a more stable optimisation. This enables Adam to make better updates to the weights, leading to faster convergence. On the other hand, while SGD is a more straightforward optimisation technique, its performance can be significantly enhanced with a fine-tuned learning rate.

In conclusion, the results from figure 6.11 show the importance of choosing the right optimisation algorithm for a given task. While Adam's adaptability and sophisticated optimisation make it a strong choice for many deep learning tasks, it's essential to remember that the effectiveness of an optimiser can be highly dependent on the specific problem, network architecture, and dataset.

6.4.4 Deconvolution Layer Response

In this section, we investigate the deconvolution layer's response to an impulse and an image input (from the CIFAR-10 dataset). We use a network that we trained in a previous experiment. Specifically, we use a network that consists of the D-C-C configuration with four-factor deconvolution enabled, the first element set as non-trainable, and no bias term.

Impulse Response

We begin this analysis by observing the layer output when an impulse is passed in as the input. We expect the deconvolution layer's response to have a different distribution than the convolutional layer's. Depending on the filter, the response could also have a distribution that includes a wider range of values or a more concentrated distribution around a point, showing the deconvolution layer's versatility to learn unique features for

each filter.

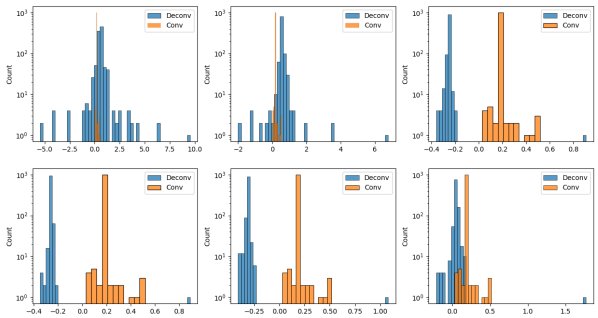

Figure 6.12: Distribution of the impulse responses for both the convolutional and deconvolution layer when an impulse is given as the input.

Figure 6.12 shows the distribution of the impulse responses for both the convolutional and deconvolution layers. We can see that the convolutional layer's response has a narrower distribution compared to most of the deconvolution layer's responses. We also see more variety in the deconvolution response distribution.

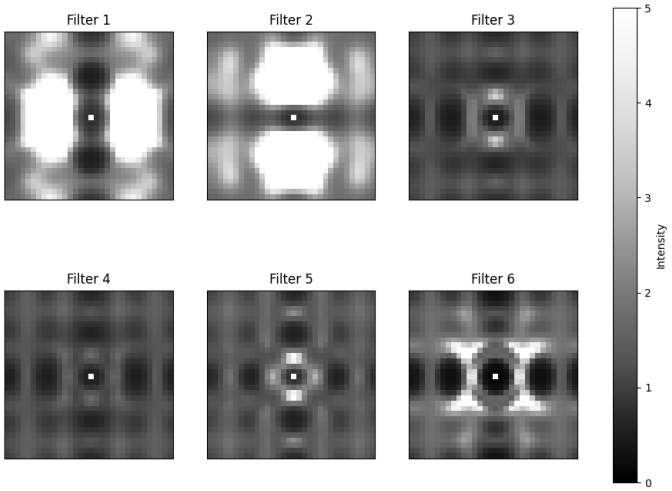

Figure 6.13: Magnitude of the impulse response of the deconvolution layer in the frequency domain.

Figure 6.13 illustrates the frequency response of the deconvolution layer. We can see that the first two filters amplify some frequencies significantly, whereas the other filters amplify the frequencies quite evenly. It is important to note the symmetry found in all the filter outputs. This can be attributed to the four-factor deconvolution that ensures a zero-phase response.

Lastly, figure 6.14 shows the distribution of the phase of the deconvolution and convolutional layers' impulse responses. We can see that the deconvolution layer applies

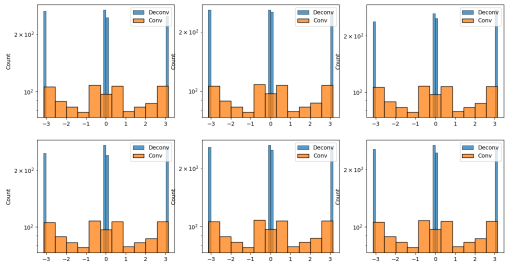

Figure 6.14: The distribution of the phase of the deconvolution and convolutional layer's impulse responses.

a zero-phase operation to the input since all the phases are found at either 0 or $\pm\pi$ indicating that the deconvolution does not add any phase to the response. It is also important to note the contrast in the distribution between the convolutional and deconvolution layer phases. We can see that the convolution has an even distribution between $-\pi$ and π whereas the deconvolution layer only has values at 0 or $\pm\pi$. This shows that the deconvolution layer does not introduce any phase shift to its output.

Image Response

We now examine the deconvolution layer's response to an input image from the CIFAR-10 dataset. From this dataset we chose an image from the aeroplane class as these images have a distinct shape, which makes it easier to see the object's shape in the deconvolution layer output. For the network in this observation, we use the network from the previous section.

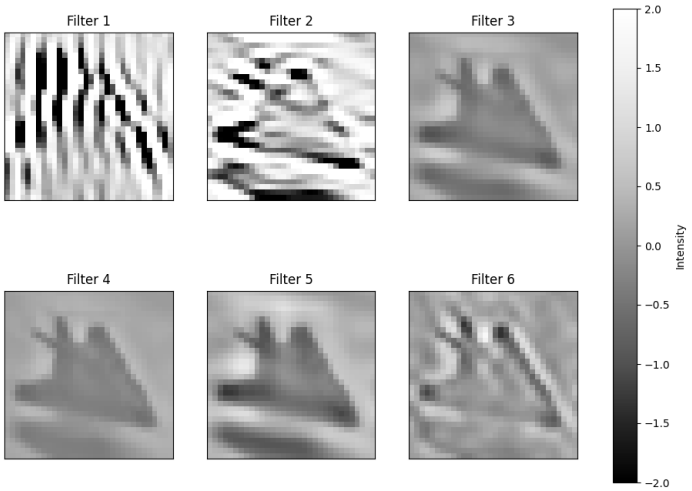

Figure 6.15: The deconvolution layer's response to an aeroplane image.

Figure 6.15 shows the deconvolution layer's response to the aeroplane input image. From filters 3, 4, 5, and 6, we can see that the deconvolution layer learns weights that accurately extract the plane's shape while suppressing elements from the background. Filter 1 shows the result of learning sub-optimal filter weights. The filter amplifies frequencies, which results in an unrecognisable image.

6.4.5 Computational Resources

In this section, we dive into the computational resource utilisation for the experiments carried out in the previous sections. We assess computational utilisation on two different metrics: training time and memory (VRAM) consumption. We also investigate the impact of both network architecture and deconvolution design strategies on computational demand. All networks were trained using the NVIDIA Tesla T4 GPU. This GPU has 16GB of GDDR6 memory with a bus width of 256 bits that is clocked at 1250Mhz. It uses the TU104 graphics processor with 2560 CUDA cores [85].

Table 6.1: Computational Demand based on network architecture changes.

Architecture	**VRAM (%)**	**VRAM (GB)**	**Total time**	**Time per epoch**
C-C-C	**9.11%**	**1.45 GB**	**3m 23s**	**20.3s**
D-C-C	9.15%	1.46 GB	3m 34s	21.4s
D-D-C	9.17%	1.47 GB	3m 57s	23.7s
D-D-D	9.81%	1.57 GB	4m 40s	28.0s

Table 6.1 shows how the VRAM and training time change when the architecture is altered. Total time and time per epoch refer to the training time in total and per epoch. VRAM refers to the amount of GPU memory used during the training of the networks. We assume the deconvolution layer consumes more VRAM and takes more time to train due to it being more computationally complex (compared to the convolutional layer). The table confirms this as we can see both the VRAM and training time increase as more deconvolution layers are added. This increase in computational demand can be attributed to the deconvolution operation having a higher complexity while also utilising an FFT. The FFT is efficient; however, it requires a lot of memory to compute. PyTorch has to store every tensor and its history. These added complexities lead to an increase in VRAM usage and training time. Lastly, these results confirm that the deconvolution, although more powerful on this classification task, requires more computational resources.

Table 6.2 shows how the VRAM and training time changes for different configurations of design strategies. In this table, the abbreviations FF, FET, and B refer to Four-Factor, First Element Trainable, and Bias, respectively. Similar to the previous experiment

Table 6.2: Computational Demand based on Changes in Deconvolution Strategies.

FF	FET	B	VRAM (%)	VRAM (GB)	Total time	Time per epoch
			9.15%	1.46 GB	3m 34s	21.4s
		✓	9.15%	1.46 GB	3m 35s	21.5s
	✓		9.15%	1.46 GB	3m 34s	21.4s
	✓	✓	9.15%	1.46 GB	3m 38s	21.8s
✓			9.15%	1.46 GB	**3m 43s**	**22.3s**
✓		✓	9.15%	1.46 GB	3m 42s	22.2s
✓	✓		9.15%	1.46 GB	3m 38s	21.8s
✓	✓	✓	9.15%	1.46 GB	3m 47s	22.7s

we expect that as more design strategies are introduced, the VRAM consumption and computation time will increase. Based on the results from table 6.2, we can see that the training time increases as design strategies are included. Still, the VRAM consumption remains the same throughout the experiments. The increase in training time shows that the added complexities of the design strategies increase computation time and, therefore, require more computational resources. It is especially interesting to note the jump in training time when the four-factor deconvolution is added. Table 6.2 shows that the four-factor deconvolution takes more time to compute. These results can be attributed to the added operations, such as three more element-wise multiplications, reflections, and shifts. This confirms that the deconvolution design strategies require more computational resources while also providing better performance.

6.5 Conclusion

In this chapter we explored the deconvolution layer performance on an image classification task. Our first set of experiments aimed to answer the question of which layer composition produces the best performance for an image classification task. We used the LeNet-5 architecture as a baseline and found that having a deconvolution layer preceded by two convolutional layers achieved the best accuracy. Furthermore, this result does not fully support our hypothesis that adding more deconvolution layer's will produce better performance. This is because we only see a performance increase when one deconvolution layer is added. However, when more of the convolution layers are replace with deconvolution layer's it decreases network performance.

Next, we looked at the effect that certain deconvolution design strategies have on the network's performance on image classification. We found that having four-factor enabled, first element set to trainable and bias enabled produced the best result on the training

set which supports our hypothesis that these design strategies will enhance the DeNN's performance. However, having all three of these design options enabled makes the network susceptible to overfitting the training data. We found that turning off the bias term alleviates this problem and achieves the best accuracy on the validation data. This indicates that this configuration allows the network to generalise better to unseen data.

Subsequently, we tested the network's sensitivity to changes in hyperparameters. We changed the following hyperparameters: number of filters, filter size, batch size, learning rate, and optimisation algorithm. We found that increasing the number of filters improves their accuracy; however, the improvements start to diminish at around 24 filters. Increasing the number of filters makes the network more prone to over-training. When testing filter sizes, we found that a filter size of 3×3 achieves the best performance by a margin of 10%. We also found that the network performance degrades as the filter sizes are increased. With regard to batch size and learning rate, we found that a batch size of 64 and a learning rate of 1×10^{-4} achieved the best result. Finally, the Adam optimisation algorithm produced a better result than SGD. However, we believe that with more training time, SGD will be able to outperform Adam.

The next section explored the deconvolution layer's response to an impulse and to a specific image input. By looking at the layer's impulse response, we found that the deconvolution layer can learn a filter with both a small number of parameters, similar to a convolution filter, and a large effective spatial extent that a convolutional layer is unable to learn. We also found that the deconvolution layer provides a symmetrical frequency response across both axes, with zero phase. The deconvolution image response shows how the layer is able to learn filters that extract the object from the background. However, the filters are susceptible to learning weights that amplify frequencies excessively, which generates sub-optimal features.

Finally, we compare the computational cost of the deconvolution layer against the convolutional layer. We found that the deconvolution layer demands more computational resources than the convolutional layer. We also investigated the impact that the different deconvolution design strategies have on the computational resource requirements. We found that as more strategies are enabled the computational cost increases, with the four-factor strategy increasing the computational cost by the most.

In summary, the deconvolution layer has been shown to improve image classification performance when integrated into a CNN. The layer offers multiple design strategies, each able to enhance network performance. While promising in terms of accuracy, the deconvolution layer demands slightly more computational resources when compared

to traditional convolutional layers, thus making it important to balance performance gains with these resource implications. This research sheds light on the advantages and challenges posed by deconvolution layers, paving the way for future studies to delve deeper into optimising its usage in varied applications. It also underlines the importance of understanding layer responses and carefully selecting design strategies to harness the layer's full potential while mitigating issues like overfitting.

Chapter 7

Conclusion

The objectives of this dissertation were to design and detail the deconvolution layer architecture, test it on computer vision tasks such as SISR and image classification, investigate the layer's response to different inputs, and investigate the layer's computational resource demand. All these objectives tie together to answer the question posed in the introduction:

"Is it possible to develop an alternative operation to convolution, specifically a frequency domain deconvolution operation, that can be used as a backbone inside a neural network for computer vision problems? And if so, how does this new approach compare in terms of efficiency and accuracy to traditional CNNs?"

The dissertation begins with a background in deep learning and the deconvolution operation. Chapter 2 starts with an overview of the convolution operation and then dives into the CNNs, explaining their architecture, how they're trained, as well as their different applications. Next, it explains the deconvolution operation and its applications. The next section of the chapter gives an overview of interpolation techniques. The final section provides a background on concepts in PyTorch like autograd, tensors, and the neural network module.

Chapter 3 provides an overview of previous work that is related to the research in this dissertation. Initially, it details research conducted prior to the advent of deep learning and then dives into work done on SISR using deep learning algorithms. Next, the chapter provides an overview of image classification. The chapter speaks about work done prior to deep learning then discusses research on classification using deep learning methods. Lastly, the chapter investigates work done on image deconvolution where it presents work

done both prior and after the advent of deep learning. This section highlights that typical deep learning image deconvolution do not use a deconvolution and rather a convolution operation.

In chapter 4 we complete the first objective by detailing the deconvolution layer architecture that we have designed. This is done by motivating why using a deconvolution operation is beneficial in deep learning. Furthermore, we present the FFT implementation of the deconvolution layer which is followed by the details of each design strategy implemented inside of it. Next, the chapter presents a one-dimensional application of the deconvolution layer. This example helps solidify understanding while also showing the deconvolution layer's potential on a deep learning task. Finally, the chapter ends with an overview of the experiments conducted in this dissertation.

In chapter 5, we test the deconvolution layer's performance using two popular SISR architectures: the SRCNN and the ESPCN. The experiments were designed to evaluate the potential advantages of integrating the deconvolution layer into these architectures. These experiments focused on employing different design strategies and hyperparameter configurations. We found that while certain configurations, such as four-factor deconvolution and padding, showed potential to improve the performance of the modified network, they consistently fell short of the original SRCNN architecture. This showcases the robustness of the SRCNN architecture for SISR tasks. We tested the deconvolution layer's hyperparameter sensitivity by integrating it into the ESPCN architecture and varied hyperparameters such as the number of filters, filter size, and loss function choice. We found that using the DCT or MSE loss function helped the ESPDeN achieve a better result than the original ESPCN. Furthermore, we found that varying the filter size does not affect network performance and increasing the number of filters leads to overfitting to the training data. These insights shed light on the potential and ability that the deconvolution layer has in SISR.

In chapter 6, we investigated the impact of the deconvolution layer on an image classification task. This was done by running experiments testing different layer configurations, different combinations of design strategies, and the deconvolution layer's sensitivity to change in hyperparameters. We found that if we use a combination of a deconvolution layer followed by two convolutional layers, it achieves the highest accuracy. This also shows that integrating a deconvolution layer into a CNN improves the CNN's performance and also outperforms the original CNN architecture. When testing different design strategies, we found that activating four-factor deconvolution, including a bias term, and setting the first element as trainable yields the best result on the training data. However, this makes the network prone to overfitting. We discovered that if we

remove the bias term, it reduces the network's ability to overfit the training data and increases the accuracy of the validation set. When exploring hyperparameter sensitivity, we found that all hyperparameters affect the network's performance. By tuning these hyperparameters, one can improve the network's accuracy substantially.

In the same chapter, we investigated the deconvolution layer's response to an impulse and image input. We found that the deconvolution layer is able to learn long-range filters while still having the ability to learn filters similar to the convolutional layer. From the image response, we found that the deconvolution layer manages to separate the object from the background successfully.

In the final section of chapter 6, we investigate the deconvolution layer's computational demand. We found that the deconvolution layer requires slightly more computational resources compared to the convolution. We explain this can be attributed to the increase in complexity, compared to the convolutional layer, and the use of an FFT, which would require more memory to compute.

To answer the main question of this dissertation: Firstly, we have managed to develop an alternative layer to the convolutional layer that is based on frequency domain deconvolution. By integrating deconvolution into CNNs, we found significant improvements in the network's accuracy on image classification. With regards to SISR, we found specific network settings that allow a DeNN to outperform a CNN. However, there is not a lot of evidence that supports the hypothesis that DeNNs are inherently better than CNNs on an SISR task. In summary, the deconvolution layer does provide a performance gain in image classification and, in some cases, for SISR. Still, it does come at a slightly higher computational cost compared to traditional CNNs.

In conclusion, this dissertation aimed to comprehensively explore the potential of the deconvolution layer as an alternative to the convolutional layer in computer vision tasks. This was done through rigorous experimentation on image classification and SISR, where we found the strengths and weaknesses of this novel deep learning layer. While the deconvolution layer presented promising attributes, such as its ability to learn long-range filters and separate objects from the rest of the image, it also highlights the efficiency and resilience of established networks. As the field of deep learning evolves, it is imperative to not only push the limits of existing architectures but also to innovate and experiment with new approaches. The research and findings presented in this paper serve as the foundation for new methodologies, emphasising the meticulous design, comprehensive experimentation and continuous innovation in the rapidly evolving landscape that is deep learning.

Chapter 8

Recommendations for Future Work

This dissertation, while comprehensive, only scratches the surface of what is possible with the deconvolution layer. Here are our recommendations for future work on this topic:

- **Explore Other Computer Vision Tasks:** We recommend exploring applications beyond image classification and SISR. We recommend exploring applications commonly tackled by CNNs. This can be object detection, image segmentation and more.
- **Generative Adversarial Networks:** We suggest exploring how the deconvolution layer impacts the performance of GANS, especially in SISR.
- **Sensitivity to Regularisation:** We also recommend exploring regularisation techniques such as data augmentation and L1 regularisation to see how they affect the network's performance on seen and unseen data.
- **Deconvolution-based Architectures:** The architectures used in this dissertation were adapted from CNN architectures. We suggest designing new architectures that utilise the strengths of the deconvolution network.
- **Optimised Implementation:** We recommend refining the implementation for better computational efficiency. This can propel its adoption in industry settings.
- **Loss Functions for SISR:** We recommend investigating different loss functions for the DeNNs on SISR. This dissertation briefly touched on this. There is some potential with frequency-based loss functions; however, we recommend looking into an additional loss function called perceptual loss, which was introduced in [86].

Bibliography

[1] E. M. Stein and R. Shakarchi, *Fourier analysis: an introduction*. Princeton University Press, 2003.

[2] A. V. Oppenheim and R. W. Schafer, *Digital signal processing*. Prentice-Hall, 1975.

[3] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the fast Fourier transform," *IEEE ASSP Magazine*, vol. 1, no. 4, pp. 14–21, 1984, archived (PDF) from the original on 2013-03-19. [Online]. Available: https://ieeexplore.ieee.org/document/1162257

[4] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992.

[5] R. Bracewell, *The Fourier transform & its applications*, 3rd ed. Boston: McGraw-Hill, 1999.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, 2020, s2CID 218955622.

[9] A. van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2643–2651.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] (2023) torch.nn.Conv2d. PyTorch. PyTorch Documentation, accessed 2023-08-05. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[12] (2023) tf.keras.layers.Conv2D. TensorFlow. TensorFlow Documentation, accessed 2023-08-05. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

[13] (2021) Convolutional Neural Network. NVIDIA. NVIDIA Glossary, accessed 2023-08-05. [Online]. Available: https://www.nvidia.com/en-us/glossary/data-science/convolutional-neural-network/

[14] N. Shahriar, "What is Convolutional Neural Network-CNN (deep learning)," Feb 2023. [Online]. Available: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5

[15] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, High Performance Convolutional Neural Networks for Image Classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence-Volume Volume Two*, vol. 2, 2011, pp. 1237–1242, retrieved 17 November 2013. [Online]. Available: https://www.ijcai.org/Proceedings/11/Papers/210.pdf

[16] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. New York, NY: Institute of Electrical and Electronics Engineers (IEEE), June 2012, pp. 3642–3649.

[17] Y. LeCun, C. Cortes, and C. J. Burges, "THE MNIST DATABASE of handwritten digits," NYU: New York, NY and Google Labs: New York and Microsoft Research: Redmond, 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[18] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face Recognition: A Convolutional Neural Network Approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[19] M. Matusugu, K. Mori, Y. Mitari, and Y. Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," *Neural Networks*, vol. 16, no. 5, pp. 555–559, 2003, retrieved 17 November 2013.

[20] "New computer vision challenge wants to teach robots to see in 3D," *New Scientist*, retrieved 3 February 2018.

[21] J. Markoff, "For Web Images, Creating New Technology to Seek and Find," *The New York Times*, retrieved 3 February 2018.

[22] "ImageNet Large Scale Visual Recognition Competition 2014 (ILSVRC2014)," 2014, accessed 2023-08-05. [Online]. Available: http://www.image-net.org/challenges/LSVRC/2014/

[23] Szegedy, Christian and Liu, Wei and Jia, Yangqing and Sermanet, Pierre and Reed, Scott E. and Anguelov, Dragomir and Erhan, Dumitru and Vanhoucke, Vincent and Rabinovich, Andrew, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*. Boston, MA, USA: IEEE Computer Society, June 7–12 2015, pp. 1–9.

[24] C. Dong, C. C. Loy, K. He, and X. Tang, "Image Super-Resolution Using Deep Convolutional Networks," 2015.

[25] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 105–114. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.19

[26] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, jul 2017. [Online]. Available: https://doi.org/10.1109%2Ftip.2017.2662206

[27] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, "DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks," 2018.

[28] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative Image Inpainting with Contextual Attention," 2018.

[29] B. Zhu, J. Liu, S. Cauley *et al.*, "Image reconstruction by domain-transform manifold learning," *Nature*, vol. 555, pp. 487–492, 2018. [Online]. Available: https://doi.org/10.1038/nature25988

[30] E. Grefenstette, P. Blunsom, N. de Freitas, and K. M. Hermann, "A Deep Architecture for Semantic Parsing," *arXiv:1404.7296 [cs.CL]*, April 29 2014.

[31] G. Mesnil, L. Deng, J. Gao, X. He, and Y. Shen, "Learning Semantic Representations Using Convolutional Neural Networks for Web Search – Microsoft Research," Microsoft Research, April 2014, retrieved 2015-12-17.

[32] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," *arXiv:1404.2188 [cs.CL]*, April 8 2014.

[33] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *arXiv:1408.5882 [cs.CL]*, August 25 2014.

[34] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008.

[35] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (almost) from Scratch," *arXiv:1103.0398 [cs.LG]*, March 2 2011.

[36] S. Bai, J. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv:1803.01271 [cs.LG]*, 2018.

[37] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of CNN and RNN for natural language processing," *arXiv:1702.01923 [cs.LG]*, March 2 2017.

[38] N. Gruber, "Detecting dynamics of action in text with a recurrent neural network," *Neural Computing and Applications*, vol. 33, no. 12, pp. 15 709–15 718, 2021.

[39] J. Haotian, L. Zhong, and L. Qianxiao, "Approximation Theory of Convolutional Architectures for Time Series Modelling," in *International Conference on Machine Learning*, 2021.

[40] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-Series Anomaly Detection Service at Microsoft," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, jul 2019. [Online]. Available: https://doi.org/10.1145%2F3292500.3330680

[41] T. O'Haver, "Intro to Signal Processing - Deconvolution," Retrieved 2007-08-15, 2007, university of Maryland at College Park.

[42] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. Cambridge, Mass: MIT Press, 1964.

[43] G. Cha, D. Lee, and S. Park, "Visualization of Structural Shape Information based on Octree using Terrestrial Laser Scanning," *Journal of the Korea Academia-Industrial cooperation Society*, vol. 17, pp. 8–16, 08 2016.

[44] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," 2016.

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

[46] T. Lehmann, C. Gonner, and K. Spitzer, "Survey: interpolation methods in medical image processing," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1049–1075, 1999.

[47] J. A. Parker, R. V. Kenyon, and D. E. Troxel, "Comparison of Interpolating Methods for Image Resampling," *IEEE Transactions on Medical Imaging*, vol. 2, no. 1, pp. 31–39, 1983.

[48] C. E. Duchon", ""lanczos filtering in one and two dimensions"," *"Journal of Applied Meteorology and Climatology"*, vol. "18", no. "8", pp. "1016 – 1022", "1979". [Online]. Available: "https://journals.ametsoc.org/view/journals/apme/18/8/1520-0450_1979_018_1016_lfioat_2_0_co_2.xml"

[49] W. Freeman, T. Jones, and E. Pasztor, "Example-based super-resolution," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, 2002.

[50] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[51] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017.

[52] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional neural networks," in *European Conference on Computer Vision (ECCV)*, 2014.

[53] S. Papert, "The Summer Vision Project," *MIT AI Memos (1959 - 2004)*, 7 1966. [Online]. Available: hdl:1721.1/6125

[54] M. A. Boden, *Mind as Machine: A History of Cognitive Science.* Clarendon Press, 2006.

[55] T. Kanade, *Three-Dimensional Machine Vision.* Springer Science Business Media, 12 2012.

[56] W. H. Richardson, "Bayesian-Based Iterative Method of Image Restoration," *Journal of the Optical Society of America*, vol. 62, pp. 55–59, 1972. [Online]. Available: https://api.semanticscholar.org/CorpusID:14492817

[57] J. Dong, S. Roth, and B. Schiele, "Deep Wiener Deconvolution: Wiener Meets Deep Learning for Image Deblurring," *CoRR*, vol. abs/2103.09962, 2021. [Online]. Available: https://arxiv.org/abs/2103.09962

[58] Z. Zhang, Y. Cheng, J. Suo, L. Bian, and Q. Dai, "Infwide: Image and feature space wiener deconvolution network for non-blind image deblurring in low-light conditions," *IEEE Transactions on Image Processing*, vol. 32, pp. 1390–1402, 2023.

[59] D. Linsley, J. Kim, V. Veerabadran, and T. Serre, "Learning long-range spatial dependencies with horizontal gated-recurrent units," 2019.

[60] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[61] A. Tekalp, H. Kaufman, and J. Woods, "Identification of image and blur parameters for the restoration of noncausal blurs," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 963–972, 1986.

[62] J. Lim, *Two-dimensional Signal and Image Processing*, ser. Prentice-Hall international editions. Prentice Hall, 1990. [Online]. Available: https://books.google.co.za/books?id=6vdSAAAAMAAJ

[63] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, 12 2014.

[64] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-Recursive Convolutional Network for Image Super-Resolution," 2016.

[65] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, "Residual dense network for image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[66] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *European Conference on Computer Vision (ECCV)*, 2018.

[67] J. Cai, H. Zeng, H. Yong, Z. Cao, and L. Zhang, "Toward real-world single image super-resolution: A new benchmark and a new model," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[68] Computer Vision Lab, ETH Zurich, "DIV2K dataset: High Quality 2K Resolution Images," Computer Vision Lab, ETH Zurich, 2017. [Online]. Available: https://data.vision.ee.ethz.ch/cvl/DIV2K/

[69] M. Bevilacqua, A. Roumy, C. Guillemot, and M.-L. Alberi-Morel, "Low-Complexity Single Image Super-Resolution Based on Nonnegative Neighbor Embedding," 09 2012.

[70] D. Occorsio, G. Ramella, and W. Themistoclakis, "Image scaling by de la Vallée-Poussin filtered interpolation," 09 2021.

[71] C. Wang *et al.*, "Awesome Super Resolution: Dataset," GitHub repository, 2023, accessed: 20-08-2023. [Online]. Available: https://github.com/ChaofWang/Awesome-Super-Resolution/blob/master/dataset.md

[72] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," 2019.

[73] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. Devito, Z. Lin, A. Desmaison, L. Antiga, O. Srl, and A. Lerer, "Automatic differentiation in PyTorch," in *Advances in Neural Information Processing Systems*, vol. 32. Facebook AI Research, 2019.

[74] I. Sonata, Y. Heryadi, L. Lukas, and A. Wibowo, "Autonomous car using CNN deep learning algorithm," *Journal of Physics: Conference Series*, vol. 1869, no. 1, p. 012071, apr 2021. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1869/1/012071

[75] J. A. Diaz-Garcia, M. D. Ruiz, and M. J. Martin-Bautista, "NOFACE: A new framework for irrelevant content filtering in social media according to credibility and expertise," *Expert Systems with Applications*, vol. 208, p. 118063, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422012684

[76] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[78] F. Chen and J. Y. Tsou, "Assessing the effects of convolutional neural network architectural factors on model performance for remote sensing image classification: An in-depth investigation," *International Journal of Applied Earth Observation and Geoinformation*, vol. 112, p. 102865, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156984322200067X

[79] M. Liu, L. Chen, X. Du, L. Jin, and M. Shang, "Activated gradients for deep neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 2156–2168, 2023.

[80] D. Gong, Y. Lu, Y. Zhao, X. Chen, and X. Guo, "A Novel U-Net Based Deep Learning Method for 3D Cardiovascular MRI Segmentation," *Computational Intelligence and Neuroscience*, vol. 2022, p. 4103524, 2022. [Online]. Available: https://doi.org/10.1155/2022/4103524

[81] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023, https://D2L.ai.

[82] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *University of Toronto*, 05 2012.

[83] A. Krizhevsky *et al.*, "The CIFAR-10 and CIFAR-100 Datasets," University of Toronto, 2009, accessed 2023-08-21. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[84] Y.-T. Chen, Y. Chen, Y.-Y. Chen, Y.-T. Huang, H. Wong, J.-L. Yan, and J.-J. Wang, "Deep Learning–Based Brain Computed Tomography Image Classification with Hyperparameter Optimization through Transfer Learning for Stroke," *Diagnostics*, vol. 12, no. 4, p. 807, 2022. [Online]. Available: https://www.mdpi.com/2075-4418/12/4/807/pdf?version=1648209686

[85] "TechPowerUp — techpowerup.com," https://www.techpowerup.com/gpu-specs/tesla-t4.c3316, accessed 19-09-2023.

[86] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 694–711.

Appendix A

Additional Diagrams

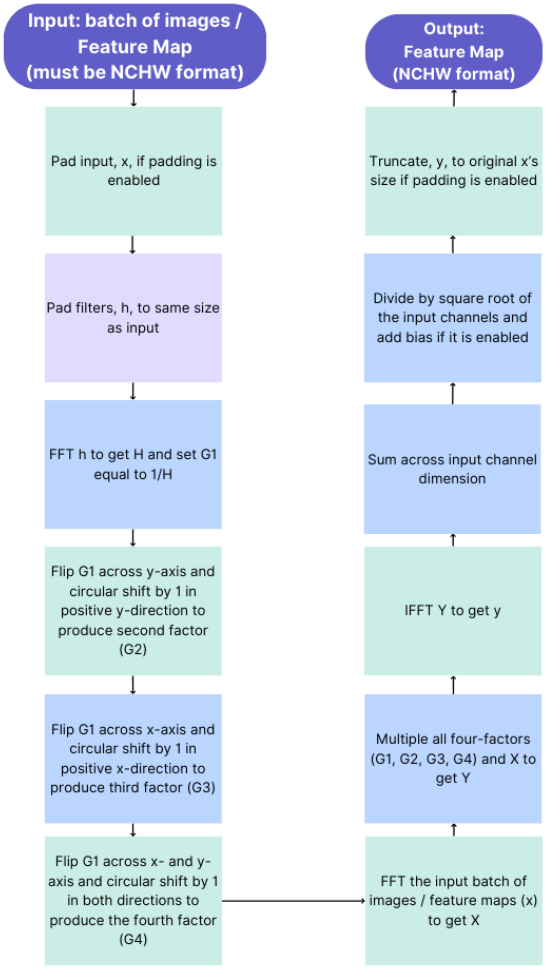

Figure A.1: Flowchart of Deconvolution Layer.

Appendix B

Source code

The source code for this project can be found at `https://github.com/rashaad-meyer/ml_masters_uct`.