Uses of Complex Wavelets in Deep Convolutional Neural Networks

Fergal Cotter

Supervisor: Prof. Nick Kingsbury
Prof. Joan Lasenby

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
PhD

Trinity College April 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Fergal Cotter
April 2019

Acknowledgements

I would like to thank my supervisor Nick Kingsbury who has dedicated so much of his time to help my research. He has not only been instructing and knowledgeable, but very kind and supportive. I would also like to thank my advisor, Joan Lasenby for supporting me in my first term when Nick was away, and for always being helpful. I must also acknowledge YiChen Yang and Ben Chaudhri who have done fantastic work helping me develop ideas and code for my research.

I sincerely thank Trinity College for both being my alma mater and for sponsoring me to do my research. Without their generosity I would not be here.

And finally, I would like to thank my girlfriend Cordelia, and my parents Bill and Mary-Rose for their ongoing support.

Abstract

Image understanding has long been a goal for computer vision. It has proved to be an exceptionally difficult task due to the large amounts of variability that are inherent to objects in scene. Recent advances in supervised learning methods, particularly convolutional neural networks (CNNs), have pushed the frontier of what we have been able to train computers to do.

Despite their successes, the mechanics of how these networks are able to recognize objects are little understood. Worse still is that we do not yet have methods or procedures that allow us to train these networks. The father of CNNs, Yann LeCun, summed it up as:

> There are certain recipes (for building CNNs) that work and certain recipes that don't, and we don't know why.

We believe that if we can build a well understood and well-defined network that mimicks CNN (i.e., it is able to extract the same features from an image, and able to combine these features to discriminate between classes of objects), then we will gain a huge amount of invaluable insight into what is required in these networks as well as what is learned.

In this paper we explore our attempts so far at trying to achieve this. In particular, we start by examining the previous work on Scatternets by Stephané Mallat. These are deep networks that involve successive convolutions with wavelets, a well understood and well-defined topic. We draw parallels between the wavelets that make up the Scatternet and the learned features of a CNN and clarify their differences. We then go on to build a two stage network that replaces the early layers of a CNN with a Scatternet and examine the progresses we have made with it.

Finally, we lay out our plan for improving this hybrid network, and how we believe we can take the Scatternet to deeper layers.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Gradients of Complex Activations

1.1.1 Cauchy-Riemann Equations

None of these will be solved. However, we can still find the partial derivatives w.r.t. the real and imaginary parts.

1.1.2 Complex Magntitude

Care must be taken when calculating gradients for the complex magnitude, as the gradient is undefined at the origin. We take the common approach of setting the gradient at the corner point to be 0.

Let us call the real and imaginary inputs to a magnitude block x and y, and we define the real output r as:

$$r = |x + jy| = \sqrt{x^2 + y^2}$$

Then the partial derivatives w.r.t. the real and imaginary inputs are:

$$\begin{aligned} \frac{\partial r}{\partial x} &= \frac{x}{\sqrt{x^2 + y^2}} = \frac{x}{r} \\ \frac{\partial r}{\partial y} &= \frac{y}{\sqrt{x^2 + y^2}} = \frac{y}{r} \end{aligned}$$

Except for the singularity at the origin, these partial derivatives are restricted to be in the range $[-1, 1]$. The complex magnitude is convex in x and y as:

$$\nabla^2 r(x,y) = \frac{1}{r^3}\begin{bmatrix} y^2 & -xy \\ -xy & x^2 \end{bmatrix} = \frac{1}{r^3}\begin{bmatrix} y \\ -x \end{bmatrix}\begin{bmatrix} y & -x \end{bmatrix} \geq 0$$

These partial derivatives are very variable around 0. **Show a plot of this**. We can smooth it out by adding a smoothing term:

$$r_s = \sqrt{x^2+y^2+b} - \sqrt{b}$$

This keeps the magnitude zero for small x, y but does slightly shrink larger values. The gain we get however is a new smoother gradient surface **Plot this**.

Chapter 2

Image Recognition Review

Image recognition tools have undergone a revolution in the past decade. Within a significant sector of the research community, previous state of the art models have been all but abandoned in place of newer deep learning methods. While there are too many to explore in detail, we give a brief history of some of the relevant methods in this chapter, outlining the benefits and drawbacks of each.

2.1 Notation

We define standard notation to help the reader better understand figures and equations. Many of the terms we define here relate to concepts that have not been introduced yet, so may be unclear until later.

- **Pixel coordinates**
  When referencing spatial coordinates in an image, the preferred index is $\mathbf{u}$ for a 2D vector of coordinates, or $[u_1, u_2]$ if we wish to specify the coordinates explicitly. u_1 indexes rows from top to bottom of an image, and u_2 indexes columns from left to right. We typically use $H \times W$ for the size of the image, (but this is less strict). I.e., $u_1 \in \{0, 1, \dots H-1\}$ and $u_2 \in 0, 1, \dots W-1$. An image can be either referenced by $x[\mathbf{u}]$ or $I[\mathbf{u}]$.

- **Convolutional networks**
  The input is $x[\mathbf{u}, d]$, with true label y. Intermediate feature maps are $z[\mathbf{u}, d]$ and the output vector is $\hat{y}[c]$, where d indexes the *depth* of the image/feature map (usually 1 or 3 for x, but arbitrary for z) and c indexes the number of classes. The convolutional kernels are $f[\mathbf{u}, d, n]$ and the fully connected weight matrices

are W_{ij}, where the new index, n indexes the number of kernels in a layer. The biases for both these layer types are contained within the vectors b.

To distinguish between features, filters, weights and biases of different levels in a deep network, we add a layer subscript, or l for the general case, i.e., $z_l[\mathbf{u}, d]$ indexes the feature map at the l-th layer of a deep network.

In cases where we need to index a particular sample in a dataset, we use the standard machine learning notation of adding a (i) superscript to the term, i.e., $x^{(i)}$ refers to the i-th input.

It is difficult to adhere strictly to this notation, particularly when including images from other works. Where the notation deviates in a non-obvious way, we will make it clear.

- **Fourier transforms and wavelets**
  When referring to the Fourier transform of a function, f, we typically adopt the overbar notation: i.e., $\mathcal{F}\{f\} = \bar{f}$.

  For wavelets, a slight variation on the standard notation is used. The reason for the modification will become clearer when we introduce the DT$\mathbb{C}$WT, which has two filter bank trees, so needs double the notation. The scaling function is still ϕ, the wavelet function is ψ but the filter banks are h_0, g_0 for low pass analysis, h_1, g_1 for high-pass analysis, $\tilde{h}_0, \tilde{g}_0$ for low pass synthesis and $\tilde{h}_1, \tilde{g}_1$ for high-pass synthesis.

  In a multiscale environment, j indexes scale from $\{1, 2, \ldots, J\}$. For 2D complex wavelets, θ indexes the orientation at which the wavelet has the largest response, i.e., $\psi_{\theta,j}$ refers to the wavelet at orientation θ at the j-th scale.

- **Other**
  A reconstructed signal is deonted with the hat notation: $\hat{f} \approx f$, and normalized signals are denoted with a tilde, e.g.

$$\tilde{z} = \frac{z - \mu}{\sigma}$$

2.2 Datasets

When considering an image recognition model, it is essential to have a set of images to build your model on, and to compare performance to other models. This naturally involves a choice at the outset of your design. Whatever choice is made, it is important

Figure 2.1: Sample images from ImageNet. These images have been cropped to make them square; the raw images vary in size and aspect ratio, but are all at least a few hundred pixels wide and tall.

to know the benefits of your dataset, its limitations, and most importantly, to remember that your model should still be valid outside of it.

In current image recognition and classification there are a handful of datasets which are used in the vast majority of literature. In particular, the two most commonly used are ImageNet [61] (part of the ongoing challenge — ImageNet Large Scale Visual Recognition Competition) and CIFAR-10[37]. These are at two ends of our decision spectrum.

ImageNet is a very large dataset, consisting of 1000 classes with a total of 1.4 million images. Each image is typically a few hundred pixels wide and tall, but this does vary image to image. Beyond the size of the dataset and the size of the images, it is also useful for having:

- Objects at varying scales from being only a few pixels in the image, to taking up the entire image.
- Objects with a varying number of instances in the images. I.e., it does not limit an object to only appear once in an image.
- Varying degrees of clutter in the images.
- Varying amounts of texture on the objects of interest.

Some examples are shown in Figure 2.1

On the other hand, CIFAR-10 is a dataset containing only 60000 images from 10 different classes. The images are tiny, all 32×32 pixels[1]. The key benefit of this dataset is its size. Both the images and the training set are small, which means training time is considerably reduced. This can allow us to test many different models and get feedback quickly. Some examples of these images are shown in Figure 2.2. Such small images are appealing to train models on, but when we as humans struggle to see the detail in them, perhaps too much information has been thrown away. On top of this criticism, most of the objects of interest are centred on the image, with little background clutter.

There are a few other datasets that also deserve a mention:

[1] The source images for CIFAR-10 were not necessarily square, but all have been scaled to be *before* including them in the dataset.

Figure 2.2: Sample images from the 10 classes of CIFAR-10.

- Pascal-VOC: The state of the art dataset before ImageNet, contains 21738 images in 20 classes. Like ImageNet, the images are typically medium resolution (a few hundred pixels in each direction).
- Caltech-101 and Caltech-256: Similar image size to ImageNet, but many fewer samples per class (15–30). Caltech is still often used if the goal of the model is training on small amounts of data.
- Microsoft COCO (Common Objects in Context) [43]: MS COCO has fewer object categories than ImageNet, and about one quarter of the number of images, but its 328000 images are fully segmented. It is the state of the art dataset currently being used for object detection and localization challenges.

2.3 Sparse Dictionary Learning

The work by Olshausen and Field in [55, 56] show that a system will learn the Gabor-like filters associated with the mammalian V1 cortex, if the constraints of sparsity and minimizing reconstruction error are used.

Introducing sparsity as a constraint may not seem obvious at first. The authors suggest though that it should follow from the intuition that natural images may be described in terms of a small number of structural primitives (e.g. edges, lines)[17], which is also supported by the high kurtosis (fourth order statistic) seen in natural images[16].

The problem is defined as trying to find ${\phi_i}^2$ that create a dictionary, so that the effective dimensionality of any image can be reduced:

$$\hat{I}(u_1, u_2) = \sum_i a_i \phi_i$$

PCA is a natural starting point to choose to solve this problem, as it attempts to find a set of mutually orthogonal basis functions that capture the directions of maximum variances of the data, and so can maximally reduce the dimensionality for a given reconstruction error, or alternatively, requires the fewest dictionary elements to best represent the data (due to the orthogonality constraint of PCA). The resulting

[2]The use of ϕ is deliberate here, to keep it in line with the tradition of it representing an *analysis* function in the wavelet community

Figure 2.3: Principal components calculated on 8×8 image patches from natural images, ordered in increasing variance. Taken from [55].

Figure 2.4: Sparse dictionary basis functions found from training on 16×16 patches of natural images. Taken from [55].

learned dictionary is shown in Figure 2.3. Unfortunately, the shortcomings of this approach are clear. These filters are not at all localized, and further, do not resemble any known cortical receptive fields.

Olshausen and Field instead define a cost function that promotes sparsity as well as the reconstruction error, given by:

$$E = -[\text{preserve information}] - \lambda[\text{sparseness of } a_i] \tag{2.3.1}$$

$$= \sum_{u_1,u_2} \left[I(u_1,u_2) - \sum_i a_i \phi_i \right]^2 + \lambda \sum_i S\frac{a_i}{\sigma} \tag{2.3.2}$$

where σ is a scaling constant, and $S(x)$ is a function that promotes sparsity, such as $\exp -x^2$, $\log(1+x^2)$, or $|x|$. Finding a_i and ϕ_i is then done per image patch by holding one constant, and using it to find/update the other. The resulting basis functions are shown in Figure 2.4.

The basis functions learned by this method both resemble the activations found in the V1 cortex, as well as those developed independently by engineers to efficiently code images. This suggests that the V1 cortex may be attempting to compress input into statistically independent and sparse representations.

2.4 SIFT Detectors

SIFT, or Scale Invariant Feature Transforms, were first introduced by Lowe [45] and described in depth in [46]. They are currently used extensively for image matching problems. They create a set of features in an image that are designed to be invariant to image scale and rotation, as well as illumination changes.

The algorithm has two main steps to it:

1. Keypoint detection by finding scale-space extrema — 'blobs' with Laplacian of Gaussian (LoG) filters

Figure 2.5: Keypoint descriptor for a 8×8 patch of pixels [46]. The left shows gradients being calculated per pixel, then weighted by a smooth gaussian filter. The right shows these gradients put into bins of 8 orientations, and then combined in 4 smaller 4×4 pixel areas. Unravelling this would give a $4 \times 8 = 32$ long feature vector

2. Generate feature vectors for each keypoint

'Scale' in the first step refers to the width of LoG filters (their σ parameter). They use four scales per octave, so the image is filtered successively by a LoG which has $\sigma_{i+1} = 2^{\frac{1}{3}}\sigma_i$. The increasing width of the LoG means we are successively searching coarser and coarser scales. Maxima are then found by comparing outputs to its neighbours in scale and space, and choosing the max. This point is then checked to see if it has sufficient curvature and rejected if not. If this value is above a set threshold, then that coordinate in scale and space (x, y, σ_i^2) is marked as a keypoint, and given to the feature generator.

To account for rotations across images, once a keypoint is found, its dominant orientation is first determined before generating any features. This is done by calculating the direction with the greatest gradient with a resolution of 10°. Further processing is done relative to this orientation. If there were multiple large orientations (this happens roughly 15% of the time), then multiple feature vectors are created.

Once this has been done, a feature is generated by finding gradients in a patch of pixels around the keypoint in the same scale. These gradients are then put into much coarser bins (usually around 45°), in sub-patches around the keypoint. These are then unravelled to make a low-dimensional vector to represent that keypoint. This is shown more clearly in Figure 2.5.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were initially introduced by LeCun et al. [39] in [39]. Due to the difficulty of training and initializing them, they failed to be popular for more than two decades. This changed in 2012, when advancements in pre-training with unsupervised networks [4], the use of an improved non-linearity — the Rectified Linear Unit, or ReLU, new regularization methods[26], and access to more powerful computers in graphics cards, or GPUs, allowed Krizhevsky, Sutskever and Hinton to develop AlexNet[38]. This network nearly halved the previous state of the art's error rate. Since then, interest in them has expanded very rapidly, and they have

Figure 2.6: Standard CNN architecture. Taken from [40]

been successfully applied to object detection [59] and human pose estimation [79]. It would take a considerable amount of effort to document the details of all the current enhancements and tricks many researches are using to squeeze extra accuracy, so for the purposes of this report we restrict ourselves to their generic design, with some time spent describing some of the more promising enhancements.

We would like to make note of some of the key architectures in the history of CNNs, which we, unfortunately, do not have space to describe:

- Yann LeCun's LeNet-5 [40], the state of the art design for postal digit recognition on the MNIST dataset.

- Google's GoogLeNet [75] achieved 6.67% top-5 error on ILSVRC2014, introducing the new 'inception' architecture, which uses combinations of 1×1, 3×3 and 5×5 convolutions.

- Oxford's VGG [70] — 6.8% and runner up in ILSVRC2014. The VGG design is very similar to AlexNet but was roughly twice as deep. More convolutional layers were used, but with smaller support — only 3×3. These were often stacked directly on top of each other without a non-linearity in between, to give the effective support of a 5×5 filter.

- Microsoft Research's ResNet [24] achieved 4.5% top-5 error and was the winner of ILSVRC2015. This network we will talk briefly about, as it introduced a very nice novel layer — the residual layer.

Despite the many variations of CNN architectures currently being used, most follow roughly the same recipe (shown in Figure 2.6):

2.5.1 Input Layer

The input is typically scaled, often removing the mean and dividing by the variance. This helps initialize the network in such a way as to operate within the region of activation of the non-linearities, which are necessary to turn linear convolutions into non-linear functions.

Figure 2.7: ?? Tight 2×2 pooling with stride 2, vs ?? overlapping 3×3 pooling with stride 2. Overlapping pooling has the possibility of having one large activation copied to two positions in the reduced size feature map, which places more emphasis on the odd columns.

2.5.2 Convolutional Layers

The image/layer of features is convolved by a set of filters. The filters are typically small, ranging from 3×3 in ResNet and VGG to 11×11 in AlexNet. We have quoted only spatial size here, as the filters in a CNN are always *fully connected in depth* — i.e., they will match the number of channels their input has.

For an input $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$, and filters $\mathbf{f} \in \mathbb{R}^{H' \times W' \times D \times D''}$ (D'' is the number of filters), our output $\mathbf{z} \in \mathbb{R}^{H'' \times W'' \times D''}$ will be given by:

$$z[u_1, u_2, d''] = b[d''] + \sum_{i=-\frac{H'}{2}}^{\frac{H'}{2}-1} \sum_{j=-\frac{W'}{2}}^{\frac{W'}{2}-1} \sum_{k=0}^{D-1} f[i, j, k, d'']x[u_1 - i, u_2 - j, k] \tag{2.5.1}$$

The stride is another important feature to note about the convolutional layer. Historically, unit stride was used [40, 38], but newer models started to use stride 2 [75]. Since then, designers vacillate between whether to keep unit stride or not, with some newer designs using both [74].

2.5.3 Pooling Layers

Typically following a convolutional layer (but not strictly), activations are subsampled with max pooling. Pooling adds some invariance to shifts smaller than the pooling size at the cost of information loss. For this reason, small pooling is typically done often 2×2 or 3×3, and the invariance to larger shifts comes after multiple pooling (and convolutional) layers.

While initial designs of max pooling would do it in non-overlapping regions, AlexNet used 3×3 pooling with stride 2 in their breakthrough design, quoting that it gave them an increase in accuracy of roughly 0.5% and helped prevent their network from 'overfitting'. More recent networks will typically employ either this or the original 2×2 pooling with stride 2, see Figure 2.7. A review of pooling methods in [52] found them both to perform equally well.

Figure 2.8: Differences in non-linearities. Green — the *sigmoid* function, Blue — the *tanh* function, and Red — the *ReLU*. The ReLU solves the problem of small gradients outside of the activation region (around $x = 0$) as well as promoting sparsity.

2.5.4 Activation Functions

Activation functions, neurons, or non-linearities, are the core of a neural networks expressibility. Historically, they were sigmoid or tanh functions, but these have been replaced recently by the Rectified Linear Unit (ReLU), which has equation $g(x) = \max(0, x)$. A ReLU non-linearity has two main advantages over its smoother predecessors [20, 53].

1. It is less sensitive to initial conditions as the gradients that backpropagate through it will be large even if x is large. A common observation of sigmoid and tanh non-linearities was that their learning would be slow for quite some time until the neurons came out of saturation, and then their accuracy would increase rapidly before levelling out again at a minimum [19]. The ReLU, on the other hand, has constant gradient.

2. It promotes sparsity in outputs, by setting them to a hard 0. Studies on brain energy expenditure suggest that neurons encode information in a sparse manner. Lennie [42] estimates the percentage of neurons active at the same time to be between 1 and 4%. Sigmoid and tanh functions will typically have *all* neurons firing, while the ReLU can allow neurons to fully turn off.

2.5.5 Fully Connected Layers

The convolutuion, pooling, and activation layers all conceptually form part of the *feature extraction* stage of a CNN. One or more fully connected layers are usually placed after these layers to form the *classifier*. One of the most elegant and indeed most powerful features of CNNs is this seamless connection between the *feature extraction* and *classifier* sub-networks, allowing the backpropagation of gradients through all layers of the entire network.

The fully connected layers in a CNN are the same as those in a classical Neural Network (NN), in that they compute a dot product between their input vector and a weight vector:

$$z_i = \sum_j W_{ij} x_j \tag{2.5.2}$$

The final output of the Fully Connected layer typically has the same number of outputs as the number of classes C in the classification problem.

2.5.6 Loss Function

For a given sample $q = (x, y)$, the loss function is used to measure the cost of predicting $\hat{y}$ when the true label was y. We define a loss function $l(y, \hat{y})$. A CNN is a deterministic function of its weights and inputs, $f(x, w)$ so this can be written as $l(y, f(x, w))$, or simply $l(y, x, w)$.

It is important to remember that we can choose to penalize errors however we please, although for CNNs, the vast majority of networks use the same loss function — the softmax loss. Some networks have experimented with using the hinge loss function (or 'SVM loss'), stating they could achieve improved results [22, 78].

1. *Softmax Loss:* The more common of the two, the softmax turns predictions into non-negative, unit summing values, giving the sense of outputting a probability distribution. The softmax function is applied to the C outputs of the network (one for each class):

$$p_j = \frac{e^{(f(x,w))(j)}}{\sum_{k=1}^{C} e^{(f(x,w))(k)}} \tag{2.5.3}$$

   where we have indexed the c-th element of the output vector $f(x, w)$ with $(f(x, w))(c)$. The softmax *loss* is then defined as:

$$l(y_i, x, w) = \sum_{j=1}^{C} \mathbb{1}\{y_i = j\} \log p_j \tag{2.5.4}$$

   Where $\mathbb{1}$ is the indicator function, and y_i is the true label for input i.

2. *Hinge Loss:* The same loss function from Support Vector Machines (SVMs) can be used to train a large margin classifier in a CNN:

$$l(y, x, w) = \sum_{l=1}^{C} \Big[\max(0, 1 - \delta(y_i, l) w^T x_i)\Big]^p \tag{2.5.5}$$

   Using a hinge loss like this introduces extra parameters, which would typically replace the final fully connected layer. The p parameter in Equation 2.5.5 can be used to choose ℓ^1 Hinge-Loss, or ℓ^2 Squared Hinge-Loss.

Regularization

Weight regularization, such as an ℓ^2 penalty is often given to the learned parameters of a system. This applies to the parameters of the fully connected, as well as the convolutional layers in a CNN. These are added to the loss function. Often the two loss components are differentiated between by their monikers - 'data loss' and 'regularization loss'. The above equation then becomes:

$$\mathcal{L} = \mathcal{L}_{data} + \underbrace{\frac{1}{2}\lambda_{fc}\sum_{i=1}^{L_{fc}}\sum_{j}\sum_{k}(w_i[j,k])^2}_{\text{fully connected loss}} + \underbrace{\frac{1}{2}\lambda_c\sum_{i=1}^{L_c}\sum_{u_1}\sum_{u_2}\sum_{d}\sum_{n}(f_i[u_1,u_2,d,n])^2}_{\text{convolutional loss}} \tag{2.5.6}$$

Where λ_{fc} and λ_c control the regularization parameters for the network. These are often also called 'weight decay' parameters.

The choice to split the λ's between fully connected and convolutional layers was relatively arbitrary. More advanced networks can make λ a function of the layer.

Empirical Risk vs Expected Risk

So far we have defined the loss function for a given data point $(x^{(i)}, y^{(i)})$. Typically, we want our network to be able to generalize to the true real world joint distribution $P(x, y)$, minimizing the expected risk (R_E) of loss:

$$R_E(f(x,w)) = \int l(y, f(x,w))dP(x,y) \tag{2.5.7}$$

Instead, we are limited to the training set, so we must settle for the empirical risk (R_{EMP}):

$$R_{EMP}(f(x,w)) = \frac{1}{N}\sum_{i=1}^{N} l(y^{(i)}, f(x^{(i)}, w)) \tag{2.5.8}$$

2.5.7 Gradient descent vs Stochastic Gradient Descent vs Mini-Batches

We can minimize Equation 2.5.8 with *gradient descent* [60]. Updates can be made on a generic network parameter w with:

$$w_{t+1} = w_t - \eta\frac{\partial E_n}{\partial w} \tag{2.5.9}$$

where η is caleld the learning rate. Calculating the gradient $\frac{\partial E_n}{\partial w}$ is done by averaging the individual gradients $\frac{\partial l}{\partial w}$ over the entire training dataset. This can be very slow, particularly for large training sets.

Instead, we can learn far more quickly by using a single estimate for the weight update equation, i.e.,

$$w_{t+1} = w_t - \eta \frac{\partial l}{\partial w} \tag{2.5.10}$$

This is called *stochastic gradient descent*. Each weight update now uses a noisy estimate of the true gradient $\frac{\partial E_n}{\partial w}$. Carefully choosing the learning rate η update scheme can ensure that the network converges to a local minimum, but the process may not be smooth (the empirical risk may fluctuate, which could be interpreted as the network diverging).

An often used trade off between these two schemes is called *mini-batch gradient descent*. Here, the variance of the estimate of $\frac{\partial E_n}{\partial w}$ is reduced by averaging out the point estimate $\frac{\partial L}{\partial w}$ over a mini-batch of samples, size N_B. Typically $1 << N_B << N$, with N_B usually being around 128. This number gives a clue to another benefit that has seen the use of mini-batches become standard — they can make use of parallel processing. Instead of having to wait until the gradient from the previous data point was calculated and the network weights are updated, a network can now process N_B samples in parallel, calculating gradients for each point with the same weights, average all these gradients in one quick step, update the weights, and continue on with the next N_B samples. The update equation is now:

$$w_{t+1} = w_t - \eta \sum_{n=1}^{N_B} \frac{\partial l}{\partial w} \tag{2.5.11}$$

2.5.8 Backpropagation and the Chain Rule

With a deep network like most CNNs, calculating $\frac{\partial L}{\partial w}$ may not seem particularly obvious if w is a weight in one of the lower layers. Say we have a deep network, with L layers. We need to define a rule for updating the weights in all L layers of the network, however, only the weights w_L are connected to the loss function, l. We assume for whatever function the last layer is that we can write down the derivative of the output with respect to the weights $\frac{\partial z_L}{\partial w_L}$. We can then write down the weight-update gradient, $\frac{\partial l}{\partial w_L}$ with application of the chain rule:

$$\frac{\partial l}{\partial w_L} = \frac{\partial l}{\partial z_L} \frac{\partial z_L}{\partial w_L} + \underbrace{\lambda w}_{\text{from the reg. loss}} \tag{2.5.12}$$

$\frac{\partial l}{\partial z_L}$ can be done simply from the equation of the loss function used. Typically this is parameterless.

Since all of the layers in a CNN are well-defined and differentiable[3] we assume that we can also write down what $\frac{\partial z_L}{\partial z_{L-1}}$ is. Repeating this process for the next layer down, we have:

$$\frac{\partial l}{\partial w_{L-1}} = \frac{\partial l}{\partial z_L}\frac{\partial z_L}{\partial z_{L-1}}\frac{\partial z_{L-1}}{\partial w_{L-1}} \tag{2.5.13}$$

We can generalize this easily like so:

$$\frac{\partial l}{\partial w_l} = \frac{\partial l}{\partial z_L} \underbrace{\prod_{i=L}^{l+1} \frac{\partial z_i}{\partial z_{i-1}}}_{\text{product to } l\text{'s output}} \frac{\partial z_l}{\partial w_l} \tag{2.5.14}$$

2.5.9 Normalization

A common practice amongst data scientists is to normalize input data. LeCun et al. [41] state that this is an important first step in training learning networks. The paradigm is typically to remove the mean of the data, (usually) divide by the standard deviation of the data and, if possible, decorrelate or whiten the data with Principle Components Analysis.

Normalization can be dangerous as signals, or dimensions of the signal with low signal to noise ratio, can cause lots of noise to be introduced into the system if normalized naively. However, normalizing the input is still useful if done appropriately.

In multilayer networks such as a CNN, the interface between layers is often thought of as being an input to a smaller network. And so, the data should be normalized once again. Aside from simply subtracting the mean and dividing by the standard deviation, two other methods have become popular recently. The first — Local Response Normalization — was used in AlexNet, and is quite unlike standard normalization. The second — Batch Normalization [27] — was introduced more recently, and is a modification of standard normalization to include some learnable parameters.

Standard Normalization

Data in a convolutional network are passed layer to layer in arrays of 'slices'. If the input is RGB, then the input slice would be of size $H \times W \times 3$. In deeper layers, the

[3]The ReLU is not differentiable at its corner, but backpropagation can still be done easily by simply looking at the sign of the input.

number of slices typically grows to much larger numbers (it will be equal to the number of filters in the layer below).

For a generic array of feature slices, $z(u_1, u_2, d)$, standard normalization is done across the u_1 and u_2 coordinates. I.e., the mean is:

$$\mu_z(d) = \frac{1}{HW} \sum_{u_1=0}^{H-1} \sum_{u_2=0}^{W-1} z(u_1, u_2, d) \tag{2.5.15}$$

and the variance is:

$$\sigma_z(d)^2 = \frac{1}{HW} \sum_{u_1=0}^{H-1} \sum_{u_2=0}^{W-1} (z(u_1, u_2, d) - \mu_z(d))^2 \tag{2.5.16}$$

Each of which are vectors of size D. These are only point estimates on the distribution of statistics of the feature slice, so an estimate of the true $E[z]$ and $Var[z]$ is taken by averaging the means and variances over the entire *training* dataset. The normalized feature vectors are then:

$$\tilde{z}(u_1, u_2, d) = \frac{z - E[z]}{\sqrt{Var[z]}} \tag{2.5.17}$$

Local Response Normalization

This normalization scheme used in AlexNet was first introduced in [28]. It is a form of brightness normalization, as it does not involve subtracting the mean. Further, instead of dividing by the variance of a single feature map[4], they divide by a scaled version of the energy of the same pixel (**u** coordinate) in neighbouring feature maps.

I.e., consider the activation $z(u'_1, u'_2, d')$. We compute the energy of the 'local' feature maps for this pixel:

$$\gamma = \sum_{j=d'-n/2}^{d'+n/2} z^2(u'_1, u'_2, j) \tag{2.5.18}$$

Where n is on the order of 10 slices. Then, the normalized pixel value is obtained by:

$$\tilde{z}(u'_1, u'_2, d') = \frac{z(u'_1, u'_2, d')}{(k + \alpha\gamma)^\beta} \tag{2.5.19}$$

[4]or the energy of the feature map, as we have already stated the mean is not subtracted

Where $k = 2$, $\alpha = 10^{-4}$, and $\beta = 0.75$ in the case of AlexNet. While this formula may seem odd, it is just a normalization scheme that promotes 'competition' or high frequencies across feature maps. An example makes this clear.

Example 2.5.1. *Suppose the activation values are all quite large and uniform across feature maps, say they are all 100.*

In this case, the γ term will be quite large — if $n = 10$, then $\gamma = 10^5$. Then the $\alpha\gamma$ term in the denominator of Equation 2.5.19 will dominate the k term, and the output will be reduced significantly (down to 16.5).

If however, the neighbouring activations were all 0 around a central 100, then the γ term will be smaller, and the output is not reduced as much (down to 43.9).

This technique supposedly draws inspiration from the human visual system, where neuron outputs are reduced if there is a lot of activity in a neighbourhood around them.

Batch Normalization

Batch normalization proposed only very recently in [27] is a conceptually simpler technique. Despite that, it has become quite popular and has been found to be very useful. At its core, it is doing what standard normalization is doing, but also introduces two learnable parameters — scale (γ) and offset (β). Equation 2.5.17 becomes:

$$\tilde{z}(u_1, u_2, d) = \gamma \frac{z - E[z]}{\sqrt{Var[z]}} + \beta \tag{2.5.20}$$

These added parameters make it a *renormalization* scheme, as instead of centring the data around zero with unit variance, it can be centred around an arbitrary value with arbitrary variance. S etting $\gamma = \sqrt{Var[z]}$ and $\beta = E[z]$, we would get the identity transform. Alternatively, setting $\gamma = 1$ and $\beta = 0$ (the initial conditions for these learnable parameters), we get standard normalization.

The parameters γ and β are learned through backpropagation. As data are usually processed in batches, the gradients for γ and β are calculated per sample, and then averaged over the whole batch.

From Equation 2.5.20, let us briefly use the hat notation to represent the standard normalized input: $\hat{z} = (z - E[z])/\sqrt{Var[z]}$, then:

$$\begin{aligned} \tilde{z}^{(i)} &= \gamma \hat{z}^{(i)} + \beta \\ \frac{\partial \mathcal{L}}{\partial \gamma} &= \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \tilde{z}^{(i)}} \cdot \hat{z}^{(i)} \qquad (2.5.21) \\ \frac{\partial \mathcal{L}}{\partial \beta} &= \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \tilde{z}^{(i)}} \qquad (2.5.22) \end{aligned}$$

Batch normalization layers are typically placed *between* convolutional layers and non-linearities. I.e., if $Wu+b$ is the output of a convolutional layer, and $z = g(Wu+b)$ is the output of the non-linearity, then with the batch normalization step, we have:

$$\begin{aligned} z &= g(BN(Wu+b)) \\ &= g(BN(Wu)) \qquad (2.5.23) \end{aligned}$$

Where the bias term was ignored in the convolutional layer, as it can be fully merged with the 'offset' parameter β.

This has particular benefit of removing the sensitivity of our network to our initial weight scale, as for scalar a,

$$BN(Wu) = BN((aW)u) \qquad (2.5.24)$$

It is also particularly useful for backpropagation, as an increase in weights leads to *smaller* gradients [27], making the network far more resilient to the problems of vanishing and exploding gradients:

$$\begin{aligned} \frac{\partial BN((aW)u)}{\partial u} &= \frac{\partial BN(Wu)}{\partial u} \\ \frac{\partial BN((aW)u)}{\partial (aW)} &= \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W} \qquad (2.5.25) \end{aligned}$$

2.5.10 Other Layers and Trends

There are far too many paradigm shifts in the field of CNNs to cover all of them exhaustively, but we describe some other layers/network features that could influence our future work are.

Figure 2.9: Dropout Neural Net Model. Left — a standard neural net with 2 hidden layers. Right — an example of a thinned net produced by applying dropout to the left network. Crossed units have been dropped. Taken from [73]

Batch Normalization instead of Dropout

Dropout is a particularly harsh regularization scheme that randomly turns off neurons in a neural network or deep belief network, and all of its ingoing and outgoing connections [73, 26]. Each node is retained with a fixed probability p (typically around 0.5 for hidden units, and closer to 1 for input units), and the 'thinned' netowrk is sampled, giving a sample output — see Figure 2.9.

An estimate of the output distribution can be taken by repetitively sampling thinned networks, but a simpler method is typically used at test time — run the entire feedforward network, but multiply the output of each node by its dropout probability p.

This was used with great success in the AlexNet design, but recent state of the art models forgo it in favour of Batch Normalization [24, 74]. The paper that introduced Batch Normalization[27] states that:

> Removing Dropout from Modified BN-Inception (their model) speeds up training, without increasing overfitting.

No More Max Pooling

Max pooling has been falling out of favour recently, in favour of decimation by a factor of two in each direction [72] (this can be achieved efficiently by only performing the convolution on every second pixel, so is often called 'stride 2 convolution'). This is quite interesting, as is something that happens in Scatternets.

Improved Initialization

Initialization techniques are steadily being improved. Typically, the weights for a convolutional layer are initialized from random Gaussian noise, with a small variance, usually around 0.01. This arbitrary size becomes an issue with deeper designs, as the ability for a network to train is *very* dependent on its initial conditions. Weights that are too small can often lead to little or slow learning, weights that are too big can lead to divergence.

Glorot and Bengio [19] defined the set scale weights should be defined at as a function of the fan-in n_{in} (number of neurons that feed the current neuron) and fan-out

Figure 2.10: A residual unit. The identity mapping is always present, and the network learns the difference from the identity mapping, $\mathcal{F}(x)$. Taken from [24].

n_{out} (number of neurons the current neuron is connected to) of the layers:

$$\text{var}(w) = \frac{2}{n_{in} + n_{out}} \tag{2.5.26}$$

Saxe et al. [62] and Mishkin and Matas [51] independently showed that forcing orthonormality on weights works much better than this Guassian initialization. I.e., if an output $z = Wx$, then we want $W^T W = I$. Ensuring this condition can allow the network to train in a similar way to the greedy, layer-wise pre-training methods described in [4].

Residual Layers

The current state of the art design introduced a clever novel feature called a residual unit[24, 25]. The inspiration for their design came from the difficulties experienced in training deeper networks. Often, adding an extra layer would *decrease* network performance. This is counter-intuitive as the deeper layers could simply learn the identity mapping, and achieve the same performance.

To promote the chance of learning the identity mapping, they define a residual unit, shown in Figure 2.10. If a desired mapping is denoted $\mathcal{H}(x)$, instead of trying to learn this, they instead learn $\mathcal{F}(x) = \mathcal{H}(x) - x$.

2.6 Visualization Schemes

Since CNNs have become so good at a variety of tasks, it is more important now than ever to gain more insight into *how* and *what* they are learning. As described in the motivation for the project (**??**), back projecting from the result to the input space can help improve a network's interpretability, and can even help improve its ability to learn.

Zeiler et al. first attempted to use 'deconvolution' to improve their learning [81], then later for purely visualization purposes [82]. Their method involves mapping activations at different layers of the network back to the pixel space

Figure 2.11 shows the block diagram for how deconvolution is done. Inverting a convolutional layer is done by taking the 2D transpose of each slice of the filter.

Figure 2.11: A block diagram view of the model. Note the switches that are saved before the pooled features, and the filters used for deconvolution are the transpose of the filters used for a forward pass. Taken from [82].

Figure 2.12: Unpooling operation in a deconvnet. On the forward pass, the locations of the maximum values are stored (the centre map with grey squares). This is then used on the backwards pass to put values at the correct location. Figure taken from [82].

Inverting a ReLU is done by simply applying a ReLU again (ensuring only positive values can flow back through the network). Inverting a max pooling step is a little trickier, as max pooling is quite a lossy operation. Zeiler et al. get around this by saving extra information on the forward pass of the model — switches that store the location of the input that caused the maximum value. This way, on the backwards pass, it is trivial to store activations to the right position in the larger feature map. Note that the positions that did not contribute to the max pooling operation remain as zero on the backwards pass. This is shown in Figure 2.12.

Figure 2.13 gives a more detailed view of how the deconvolution works for convolutional layers.

Mahendran and Vedaldi take a slightly different route on deconvolution networks [47]. They do not store this extra information but instead define a cost function to maximize to. This results in visualization images that look very surreal, and can be quite different from the input.

Figure 2.13: Deconvolution by slices. Visualization of 2 layers of the model showing how to invert a convolutional layer. At layer 2 of the model, there are L feature maps $z_{l,2}$ (top green). Each of these feature maps was made from a different filter. The L different filters are shown below the activations in red — $f^c_{l,2}$. The c superscript on these filters indexes the channel. E.g. a convolutional filter could be $5 \times 5 \times 64$, where the first two indices the spatial support of the filter, and the third index — the 64 — is the fully connected depth aspect of the filter, the c in this case. Each filter is laid out slice by slice. For simplicity, only two slices are shown in the figure. The 2D transpose of this filter is taken and convolved with the feature map $z_{l,2}$. The result of this is $L \times C$ images. For each $c \in \{0 \dots C-1\}$, the c'th output from all L feature maps are summed together to make a pooled map $p_{c,1}$. These C pooled maps are then expanded to make the C feature maps at the layer below (indexed by k in the figure) — $z_{k,2}$. This process then repeats until we return to the input space. Not shown on this diagram are non-linear layers, but these are simple, point-wise operations. It would be trivial to insert them conceptually, by putting one more step, going from an intermediate feature map $z'_{k,1}$ to $z_{k,1}$. This figure was taken from [81].

Figure 2.14: Visualization of some deconvolved features. 9 coordinates are chosen in the feature map for layer 1 z^1, 16 for the second layer feature map z^2 and 16 for the third layer feature map. The entire dataset is run, and 9 images that made the largest activation at this point are noted. Deconvolution is then done on the feature maps from these 9 images, and the results are shown next to the actual images. Deeper layers are picking up more complex structures. Taken from [82].

Chapter 3

Frequency Analysis of Images

Computer vision is an extremely difficult task. Greyscale intensities in an image are not very helpful in understanding what is in that image. Indeed, these values are sensitive to lighting conditions and camera configurations. It would be easy to take two photos of the same scene and get two vectors x_1 and x_2 that have a very large Euclidean distance, but to a human, would represent the same objects. What is most important in an image are the local variations of image intensity. In particular, the location or phase of the waves that make up the image. A few simple experiments to prove this are shown in Figure 3.1. This likely shows that frequency based analysis is important for image understanding.

3.1 Aside — why Invertibility?

In many sections of this chapter, transforms are checked and analysed for their ability to be invertible. This is not strictly necessary for Image understanding tasks but we emphasize it because we believe that visualizing what is being learnt inside a network is the key to improving it, as do **?**] and many others. This section covers non-learned

Figure 3.1: Importance of phase for images, not greyscale intensities. **Top:** The phase of the Fourier transform of the first image is combined with the magnitude of the Fourier transform of the second image and reconstructed. Note that the first image has entirely won out and nothing is left visible of the cameraman. **Bottom:** Several pointwise operators have been applied to the Lena image (pointwise operations retain the image phase). Left: displaying the log of pixel values, centre: displaying the square of pixel values, right: displaying $x^4 - 10x^3 + 5$.

architectures, but we look at using them as front ends to learned architectures in **??**. This means that it is important to have invertible, or at least approximately invertible transforms, to be able to do visualizations.

3.2 The Fourier Transform

While the Fourier transform is the core of frequency analysis, it is a poor feature descriptor due to the infinite support of its basis functions. If a single pixel changes in the input, this can theoretically change all of the Fourier coefficients. As natural images are generally non-stationary, we need to be able to isolate frequency components in local regions of an image, and not have this property of global dependence.

The Fourier transform does have one nice property, however, in that the magnitude of Fourier coefficients are invariant to global translations, a nuisance variability. We explore this theme more in our review of the Scattering Transform by in Chapter 4.

3.3 The Wavelet Transform

The Wavelet Transform, like the Fourier Transform, can be used to decompose a signal into its frequency components. Unlike the Fourier transform though, these frequency components can be localized in space. The localization being inversely proportional to the frequency of interest which you want to measure. This means that changes in one part of the image will not affect the wavelet coefficients in another part of the image, so long as the distance between the two parts is much larger than the wavelength of the wavelets you are examining.

The corollary of this property is that wavelets can also be localised in frequency.

The Continuous Wavelet Transform (CWT) gives full flexibility on what spatial/frequency resolution is wanted, but is very redundant and computationally expensive to compute, instead the common yardstick for wavelet analysis is the Discrete Wavelet Transform (DWT) which we introduce.

3.4 Discrete Wavelet Transform and its Shortcomings

A particularly efficient implementation of the DWT is Mallat's maximally decimated dyadic filter tree, commonly used with an orthonormal basis set. Orthonormal basis

Figure 3.2: **Sensitivity of DWT coefficients to zero crossings and small shifts.** Two impulse signals $\delta(n-60)$ and $\delta(n-64)$ are shown (top), as well as the wavelet coefficients for scale $j=1$ for the DWT (middle) and for the DT$\mathbb{C}$WT (bottom). In the middle row, not only are the coefficients very different from a shifted input, but the energy has almost doubled. As the DWT is an orthonormal transform, this means that this extra energy has come from other scales. In comparison, the energy of the magnitude of the DT$\mathbb{C}$WT coefficients has remained far more constant, as has the shape of the envelope of the output. Image taken from [64].

Figure 3.3: **Typical wavelets from the 2D separable DWT.** Top: Wavelet point spread functions for the low-high, high-low, and high-high wavelets. High-high wavelets are in a checkerboard pattern, with no favoured orientation. Bottom: Idealized support of the spectra of each of the wavelets. Image taken from [64].

sets are non-redundant, which makes them computationally and memory efficient, but they also have several drawbacks. In particular:

- The DWT is sensitive to the zero crossings of its wavelets. We would like singularities in the input to yield large wavelet coefficients, but if they fall at a zero crossing of a wavelet, the output can be small. See Figure 3.2.
- They have poor directional selectivity. As the wavelets are purely real, they have passbands in all four quadrants of the frequency plane. While they can pick out edges aligned with the frequency axis, they do not have admissibility for other orientations. See Figure 3.3.
- They are not shift invariant. In particular, small shifts greatly perturb the wavelet coefficients. Figure 3.2 shows this for the centre-left and centre-right images. Figure 3.11 (right) also shows this.

The lack of shift invariance and the possibility of low outputs at singularities is a price to pay for the critically sampled property of the transform. Indeed, this shortcoming can be overcome with the undecimated DWT [48, 11], but it pays a heavy price for this both computationally, and in the number of wavelet coefficients it produces, particularly if many scales J are used.

3.5 Complex Wavelets

So the DWT has overcome the problem of space-frequency localisation, but it has introduced new problems. Fortunately, we can improve on the DWT with complex

wavelets, as they can solve these new shortcomings while maintaining the desired localization properties.

The Fourier transform does not suffer from a lack of directional selectivity and shift invariance because its basis functions are based on the complex sinusoid:

$$e^{j\omega t} = \cos(\omega t) + j\sin(\omega t) \tag{3.5.1}$$

whereas the DWT's basis functions are based on only the real sinusoid $\cos(\omega t)$[1]. As t moves along the real line, the phase of the Fourier coefficients change linearly, while their magnitude remains constant. In contrast, as t moves along the real line, the sign of the real coefficient oscillates between -1 and 1, and its magnitude changes in a non-linear way.

These nice properties come from the fact that the cosine and sine functions of the Fourier transform form a Hilbert Pair, and so together they constitute an analytic signal.

We can achieve these nice properties if the mother wavelet for our wavelet transform is analytic:

$$\psi_c(t) = \psi_r(t) + j\psi_c(t) \tag{3.5.2}$$

where $\psi_r(t)$ and $\psi_c(t)$ form a Hilbert Pair (i.e., they are 90° out of phase with each other).

There are a number of possible ways to do a wavelet transform with complex wavelets. We examine two in particular, the Fourier-based method used by Mallat et. al. in their scattering transform [7, 8, 6, 58, 57, 68, 69, 65, 66], and the separable, filter bank based DT$\mathbb{C}$WT developed by Kingsbury [36, 30, 29, 31–34, 64].

3.6 Fourier Based Wavelet Transform

The Fourier Based method used by Mallat et. al. is an efficient implementation of the Gabor Transform. Mallat tends to prefer to use a close relative of the Gabor wavelet — the Morlet wavelet — as the mother wavelet for his transform.

While the Gabor wavelets have the best theoretical trade-off between spatial and frequency localization, they have a non-zero mean. This makes the wavelet coefficients non-sparse, as they will all have a DC component to them, and makes them inadmissible as wavelets. Instead, the Morlet wavelet has the same shape, but with an extra degree

[1] we have temporarily switched to 1D notation here as it is clearer and easier to use, but the results still hold for 2D

of freedom chosen to set $\int \psi(\mathbf{u}) d\mathbf{u} = 0$. This wavelet has equation (in 2D):

$$\psi(\mathbf{u}) = \frac{1}{2\pi\sigma^2}(e^{i\mathbf{u}\xi} - \beta)e^{-\frac{|\mathbf{u}|^2}{2\sigma^2}} \tag{3.6.1}$$

where β is usually $<< 1$ and is this extra degree of freedom, σ is the size of the gaussian window, and ξ is the location of the peak frequency response — i.e., for an octave based transform, $\xi = 3\pi/4$.

add an extra degree of freedom to this by allowing for a non-circular gaussian window over the complex sinusoid, which gives control over the angular resolution of the final wavelet, so this now becomes:

$$\psi(\mathbf{u}) = \frac{\gamma}{2\pi\sigma^2}(e^{i\mathbf{u}\xi} - \beta)e^{-\mathbf{u}^\mathbf{t}\Sigma^{-1}\mathbf{u}} \tag{3.6.2}$$

Where

$$\Sigma^{-1} = \begin{bmatrix} \frac{1}{2\sigma^2} & 0 \\ 0 & \frac{\gamma^2}{2\sigma^2} \end{bmatrix}$$

The effects of modifying the eccentricity parameter γ and the window size σ are shown in Figure 3.4. To have a full two dimensional wavelet transform, we need to rotate this mother wavelet by angle θ and scale it by j/Q, where Q is the number of scales per octave (usually 1 in image processing). This can be done by doing the following substitutions in Equation 3.6.2:

$$\begin{aligned} R_\theta &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \\ \mathbf{u}_\theta &= R_{-\theta}\mathbf{u} \\ \sigma_j &= 2^{\frac{j-1}{Q}}\sigma \\ \xi_j &= \frac{\xi}{2^{\frac{j-1}{Q}}} \end{aligned}$$

combine these two variables into a single coordinate

$$\lambda = (\theta, j/Q) \tag{3.6.3}$$

Returning to the higher level notation, we can write the Morlet wavelet as the sum of real and imaginary parts: And scaled and rotated wavelets as:

$$\psi_\lambda(\mathbf{u}) = 2^{-j/Q}\psi(2^{-j/Q}R_\theta^{-1}\mathbf{u}) \tag{3.6.4}$$

Figure 3.4: Single Morlet filter with varying slants and window sizes. Top left — 45° plane wave (real part only). Top right — plane wave with $\sigma = 3, \gamma = 1$. Bottom left — plane wave with $\sigma = 3, \gamma = 0.5$. Bottom right — plane wave with $\sigma = 2, \gamma = 0.5$.

Figure 3.5: The full dictionary of Morlet wavelets used by . The real filters are on the left and the imaginary on the right. The first row correspond to scale $j = 1$, increasing up to $j = 4$. The first column corresponding to $\theta = 0$, rotating through $\pi/8$ up to the eighth column of $7\pi/8$, $\gamma = 1/2$.

3.6.1 Implementation

The Fourier Implementation of this Morlet decomposition is shown in Figure 3.6. It is based on the fact that

$$\mathcal{F}(x * \psi)(\omega) = \mathcal{F}x(\omega)\mathcal{F}\psi(\omega) \tag{3.6.5}$$

so to compute the family of outputs of $x * \psi_\lambda$, we can precompute the Fourier transform of all of the wavelets, then at run time, take the Fourier transform of the image, x, multiply with the Fourier transform of the wavelets, and then take the inverse Fourier transform of the product. The output scale can be chosen by periodizing the product of the Fourier transforms, and then compute the inverse Fourier transform at the reduced resolution.

The resulting complexity of the entire operation for an image with $N \times N$ pixels is:

- $O(N^2 \log N)$ for the forward FFT of x.
- $O(JLN^2)$ for the multiplication in the frequency domain. We can see this from Figure 3.6, there are J scales to do multiplication at, and each scale has L orientations, except for the low-low.
- $O(L\sum_j (2^{-2j}N^2)\log 2^{-2j}N^2)$ for the inverse FFTs. The term inside the sum is just the $O(N^2 \log N)$ term of an inverse FFT that has been downsampled by 2^j in each direction.

And altogether:

$$T(N) = O(N^2 \log N) + O(JLN^2) + O(L\sum_j N^2 log 2^{-j} N) \tag{3.6.6}$$

Figure 3.6: Fourier Implementation of the Morlet decomposition of an input image, with $J = 2$ scales and $L = 2$ orientations. The Fourier transform of x is calculated and multiplied with the (precomputed) Fourier transforms of a bank of Morlet filters. The results are periodized according to the target resolution, and then the inverse Fourier transform is applied. Image taken from [65].

3.6.2 Invertibility and Energy Conservation

We can write the wavelet transform of an input x as

$$\mathcal{W}x = \{x * \phi_J, x * \psi_\lambda\}_\lambda \tag{3.6.7}$$

The ℓ^2 norm of the wavelet transform is then defined by

$$\|\mathcal{W}x\|^2 = \|x * \phi_J\|^2 + \sum_\lambda \|x * \psi_\lambda\|^2 \tag{3.6.8}$$

An energy preserving transform will satisfy Plancherel's equality, so that

$$\|\mathcal{W}x\| = \|x\| \tag{3.6.9}$$

which is a nice property to have for invertibility, as well as for analysing how different signals get transformed (e.g. white noise versus standard images).

For a transform to be invertible, we examine the measure of how tightly its basis functions tile the Fourier plane with its Littlewood-Paley function:

$$A(\omega) = |\mathcal{F}\phi_J(\omega)|^2 + \sum_\lambda |\mathcal{F}\psi_\lambda(\omega)|^2 \tag{3.6.10}$$

If the tiliing is α-tight, then $\forall \omega \in \mathbb{R}^2$:

$$1 - \alpha \leq A(\omega) \leq 1 \tag{3.6.11}$$

and the wavelet operator, $\mathcal{W}$ is an α frame. If $A(\omega)$ is ever close to 0, then there is not a good coverage of the frequency plane at that location. If it ever exceeds 1, then there is overlap between bases. Both of these conditions make invertibility difficult[2]. Figure 3.7 show the invertibility of a few families of wavelets used by . Invertibility is possible, but not guaranteed for all configurations. The Fourier transform of the

[2]In practise, if $A(\omega)$ is only slightly greater 1 for only a few small areas of ω, approximate inversion can be achieved

Figure 3.7: Three Morlet Wavelet Families and their tiling of the frequency plane. For each set of parameters, the point spread functions of the wavelet bases are shown, next to their Littlewood-Paley sum $A(\omega)$. None of the configurations cover the corners of the frequency plane, and they often exceed 1. Increasing J, L (Sifre uses C in these diagrams) or Q gives better frequency localization but at the cost of spatial localization and added complexity. Image taken from [65].

inverse filters are defined by:

$$\mathcal{F}\phi_J^{-1}(\omega) = A(\omega)^{-1}\mathcal{F}\phi_J(\omega) \tag{3.6.12}$$

$$\mathcal{F}\psi_\lambda^{-1}(\omega) = A(\omega)^{-1}\mathcal{F}\psi_\lambda(\omega) \tag{3.6.13}$$

3.7 The DT$\mathbb{C}$WT

The DT$\mathbb{C}$WT was first proposed by Kingsbury in [30, 29] as a way to combat many of the shortcomings of the DWT, in particular, its poor directional selectivity, and its poor shift invariance. A thorough analsysis of the properties and benefits of the DT$\mathbb{C}$WT is done in [31, 64]. Building on these properties, it been used successfully for denoising and inverse problems [13, 84, 83, 50], texture classification [23, 12], image registration [44, 9] and SIFT-style keypoint generation matching [15, 1–3, 54] amongst many other applications.

Compared to Gabor (or Morlet) image analysis, the authors of [64] sum up the dangers as:

> A typical Gabor image analysis is either expensive to compute, is noninvertible, or both.

This nicely summarises the difference between this method and the Fourier based method outlined in § 3.6. The DT$\mathbb{C}$WT is a filter bank (FB) based wavelet transform. It is faster to implement than the Morlet analysis, as well as being more readily invertible.

3.7.1 Deisgn Criteria for the DT$\mathbb{C}$WT

It was stated in § 3.5 that if the mother (and daughter) wavelets were complex, with their real and imaginary parts forming a Hilbert pair, then the wavelet transform of a

Figure 3.8: Analysis FB for the DTℂWT . Top 'tree' forms the real component of the complex wavelet ψ_r, and the bottom tree forms the imaginary (Hilbert pair) component ψ_i. Image taken from [64].

signal with these $\{\psi_{j,n}\}_{j,n}$ would give a representation that had nice shift properties[3], was insensitive to zero crossings of the wavelet, and had good directional selectivity.

As in § 3.5, we want to have a complex mother wavelet ψ_c that satisfies Equation 3.5.2, but now achieved with filter banks. A slight deviation from standard filter bank notation, where h_0, h_1 are the analysis and g_0, g_1 are the synthesis filters. We define:

- h_0, h_1 the low and high-pass analysis filters for ψ_r (henceforth called ψ_h)
- g_0, g_1 the low and high-pass analysis fitlers for ψ_i (henceforth called ψ_g)
- $\tilde{h}_0, \tilde{h}_1$ the low and high-pass synthesis filters for $\tilde{\psi}_h$.
- $\tilde{g}_0, \tilde{g}_1$ the low and hig pass synthesis filters for $\tilde{\psi}_g$.

The dilation and wavelet equations for a 1D filter bank implementation are:

$$\phi_h(t) = \sqrt{2}\sum_n h_0(n)\phi_h(2t-n) \tag{3.7.1}$$

$$\psi_h(t) = \sqrt{2}\sum_n h_1(n)\phi_h(2t-n) \tag{3.7.2}$$

$$\phi_g(t) = \sqrt{2}\sum_n g_0(n)\phi_g(2t-n) \tag{3.7.3}$$

$$\psi_g(t) = \sqrt{2}\sum_n g_1(n)\phi_g(2t-n) \tag{3.7.4}$$

This implementation is shown in Figure 3.8.

Designing a filter bank implementation that results in Hilbert symmetric wavelets does not appear to be an easy task. However, it was shown by Kingsbury [31] (and later proved by Selesnick [63]) that the necessary conditions are conceptually very simple. One low-pass filter must be a *half-sample shift* of the other. I.e.,

$$g_0(n) \approx h_0(n-0.5) \rightarrow \psi_g(t) \approx \mathcal{H}\{\psi_h(t)\} \tag{3.7.5}$$

As the DTℂWT is designed as an invertible filter bank implementation, this is only one of the constraints. Naturally, there are also:

[3] in particular, that a shift in input gives the same shift in magnitude of the wavelet coefficients, and a linear phase shift

- Perfect reconstruction
- Finite support
- Linear phase
- Many vanishing moments at $z = -1$ for good stopband properties

to consider when building the h's and g's. The derivation of the filters that meet these conditions is covered in detail in [34, 35], and in general in [64]. The result is the option of three families of filters: biorthogonal filters ($h_0[n] = h_0[N-1-n]$ and $g_0[n] = g_0[N-n]$), q-shift filters ($g_0[n] = h_0[N-1-n]$), and common-factor filters.

3.7.2 The Resulting Wavelets and their Properties

While analytic wavelets in 1D are useful for their shift invariance, the real beauty of the DT$\mathbb{C}$WT is in its ability to make a separable 2D wavelet transform with oriented wavelets.

Figure 3.9a shows the spectrum of the wavelet when the separable product uses purely real wavelets, as is the case with the DWT. Figure 3.9b however, shows the separable product of two complex, analytic wavelets resulting in a localized and oriented 2D wavelet.

I.e., for the $+45°$ wavelet[4] (which is high in both ω_1 and ω_2), the separable product is:

$$\begin{aligned} \psi(\omega_1,\omega_2) &= \psi_c(\omega_1)\overline{\psi_c(\omega_2)} && (3.7.6)\\ &= (\psi_h(\omega_1) + j\psi_g(\omega_1))\overline{(\psi_h(\omega_2) + \psi_g(\omega_2))} \\ &= \psi_h(\omega_1)\psi_h(\omega_2) + \psi_g(\omega_1)\psi_g(\omega_2) \\ &\quad + j\left(\psi_g(\omega_1)\psi_h(\omega_2) - \psi_h(\omega_1)\psi_g(\omega_2)\right) && (3.7.7) \end{aligned}$$

Similar equations can be obtained for the other five wavelets and the scaling function, by replacing ψ with ϕ for both directions, and not taking the complex conjugate in Equation 3.7.6 to get the right hand side of the frequency plane.

Figure 3.10 shows the resulting wavelets both in the spatial domain and their idealized support in the frequency domain.

Figure 3.11 shows how the DT$\mathbb{C}$WT compares with the DWT with a shifting input.

[4] note that Figure 3.9b shows the $135°$ wavelet

(a)

(b)

Figure 3.9: (a) The high-high DWT wavelet having a passband in all 4 corners of the frequency plane vs (b) the high-high DT$\mathbb{C}$WT wavelet frequency support only existing in one quadrant. Taken from [64]

Figure 3.10: Wavelets from the 2d DT$\mathbb{C}$WT . **Top:** The six oriented filters in the space domain (only the real wavelets are shown). **Bottom:** Idealized support of the Fourier spectrum of each wavelet in the 2D frequency plane. Spectra of the the real wavelets are shown — the spectra of the complex wavelets ($\psi_h + j\psi_g$) only has support in the top half of the plane. Image taken from [64].

3.7.3 Implementation and Efficiency

Figure 3.8 showed the layout for the DT$\mathbb{C}$WT for 1D signals. We saw from Equation 3.7.7 that the 2D separable product of wavelets involved the product of ψ_g, ψ_h, ϕ_g, and ϕ_h terms, with some summing and differencing operations. Figure 3.12 shows how to efficiently implement this with FBs.

As we did for § 3.6.1, we calculate and compare the complexity of the DT$\mathbb{C}$WT . To do this, we must know the length of our h and g filters. It is also important to know that we must use different filters for the first scale to the deeper scales, as this achieves better analyticity. A typical configuration will use biorthogonal filters for the first scale, then qshift for subsequent scales. These filters have the following number of taps[5]:

	h_0	h_1	g_0	g_1
biorthogonal	5	7	7	5
qshift	10	10	10	10

The resulting complexity of the entire forward wavelet transform for an image with $N \times N$ pixels is:

- First layer (Image size $= N \times N$):

[5]This implementation uses the shorter near_sym_a biorthogonal filters and qshift_a filters. Smoother wavelets can have slightly more taps

Figure 3.11: The shift invariance of the DT$\mathbb{C}$WT (left) vs. the real DWT (right). The DT$\mathbb{C}$WT linearizes shifts in the phase change of the complex wavelet. Image taken from [30].

Figure 3.12: The filter bank implementation of the DT$\mathbb{C}$WT . Image taken from [31]

- Column filtering requires $5N^2 + 7N^2$ multiply-adds
- Row filtering to make the LoLo term requires $5N^2$ more multiply-adds
- Row filtering to make 15° and 165° requires $5N^2 + 4N^2$ multiply-adds (the 4 here comes from the Σ/Δ function block in Figure 3.12).
- Row filtering to make the 45° and 135° requires $7N^2 + 4N^2$ multiply-adds
- Row filtering to make the $75°and105°$ requires $7N^2 + 4N^2$ multiply-adds

The total being

$$T(N) = (5+7+5+5+4+7+4+7+4)N^2 = 48N^2$$

- Second and deeper layers (Image size $= 2^{-j}N \times 2^{-j}N = M \times M$):
  - Column filtering requires $10M^2 + 10M^2$ multiply-adds
  - Row filtering to make the LoLo term requires $10M^2$ more multiply-adds
  - Row filtering to make 15° and 165° requires $10M^2 + 4M^2$ multiply-adds (the 4 here comes from the Σ/Δ function block in Figure 3.12).
  - Row filtering to make the 45° and 135° requires $10M^2 + 4M^2$ multiply-adds
  - Row filtering to make the 75° and 105° requires $10M^2 + 4M^2$ multiply-adds

The total being:

$$T(M) = (10+10+10+10+4+10+4+10+4)M^2 = 68M = 68 \times 2^{-2j}N^2$$

$$T(N) = 48N^2 + \sum_{j=1}^{J} 68 \times 2^{-2j}N^2 \approx 100N^2 \tag{3.7.8}$$

As the term $\sum_{j=1}^{J} 68 \times 2^{-j}$ is a geometric series that sums to 1/3 as $j \to \infty$. We have also rounded up the sum to be conservative.

It is difficult to compare this with the complexity of the Fourier-based implementation of the Morlet wavelet transform we have derived in § 3.6.1, as we cannot readily estimate the big-O order constants for the 2D FFT method. However, the central term, JLN^2, would cost $24N^2$ multiplies for four scales and six orientations. These are complex multiplies, as they are in the Fourier domain, which requires four real

Figure 3.13: Four scales of the DT$\mathbb{C}$WT (left) and its associated frequency coverage, or $A(\omega)$ (right). Note the reduced scale compared to Figure 3.7.

multiplies. On top of this, to account for the periodic repetition of the inverse FFT implementation, Mallat et. al. symmetrically extend the image by $N/2$ in each direction. This means that the central term is already on the order of $\sim 200N^2$ multiplies, without even considering the more expensive forward and inverse FFTs.

For a simple comparison experiment, we performed the two transforms on a four core Intel i7 processor (a moderately high-end personal computer). The DT$\mathbb{C}$WT transform took roughly 0.15s on a black and white 32×32 image, vs. 0.5s for the Fourier-based method. For a larger, 512×512 image, the DT$\mathbb{C}$WT implementation took 0.35s vs. 3.5s (times were averaged over several runs).

3.7.4 Invertibility and Energy Conservation

We analysed the Littlewood-Paley function for the Morlet-Fourier implementation, and saw what areas of the spectrum were better covered than others. How about for the DT$\mathbb{C}$WT ?

It is important to note that in the case of the DT$\mathbb{C}$WT the wavelet transform is also approximately unitary, i.e.,

$$\|x\|^2 \approx \|\mathcal{W}x\|^2 \tag{3.7.9}$$

and the implementation is perfectly invertible as the Littlewood-Paley function is unity (or very near unity) $\forall\omega$. See Figure 3.13. This is not a surprise, as it is a design constraint in choosing the filters, but nonetheless is important to note.

A beneficial property of energy conservation is that the noise in the input will equal the noise in the wavelet coefficients. When we introduce Scatternets, we can show that we can keep the unitary property in the scattering coefficients. This is an important property, particularly in light of the recent investigations in [77]. This paper saw that it is easy to find cases in CNNs where a small amount of input perturbation results in a completely different class label (see Figure 3.14). Having a unitary transform limits the amount the features can change, which will make the entire network more stable to distortion and noise.

Figure 3.14: Two adversarial examples generated for AlexNet. The left column shows a correctly predicted sample, the right column an incorrectly predicted example, and the centre column the difference between the two images, magnified 10 times. Image taken from [77].

(a) DT$\mathbb{C}$WT wavelets (left to right) — 15°, 45° and 75°

(c) Morlet wavelets (left to right) — 0°, 22.5°, 45°, 67.5°, and 90°

Figure 3.15: Normalized Energy spectra of the DT$\mathbb{C}$WT wavelets versus the preferred 8 orientation Morlet wavelets by Mallat for the second quadrant. Orientations listed refer to the edge orientation in the spatial domain that gives the highest response. All wavelets have been normalized to be between zero and one. The Morlet wavelets have finer angular resolution, which can give better discrimination, at the cost of decreasing stability to deformations, and requiring larger spatial support.

3.8 Summary of Methods

One final comparison to make between the DT$\mathbb{C}$WT and the Morlet wavelets is their frequency coverage. The Morlet wavelets can be made to be tighter than the DT$\mathbb{C}$WT, which gives better angular resolution — see Figure 3.15. However it is not always better to keep getting finer and finer resolutions, indeed the Fourier transform gives the ultimate in angular resolution, but as mentioned, this makes it less stable to shifts and deformations.

Table 3.1 compares the advantages and disadvantages of the wavelet methods discussed in this chapter.

Table 3.1: A comparison of wavelet transform methods

DWT	DT$\mathbb{C}$WT	FFT-based
Efficient filter bank implementation	Efficient filter bank implementation	Convolutions done in the frequency domain
$O(N)$	$O(N)$	$O(N \log N)$
Perfect reconstruction inherent to the design	Perfect reconstruction inherent to the design	Perfect reconstruction possible, but not inherent
Energy preserving	Energy preserving	Energy preserving
Critically sampled	Tight frame, with $4:1$ redundancy	Tight frame, with variable redundancy (often around $4:1$ in implenetations by)
Purely real wavelets	Complex, analytic wavelets	Complex, analytic wavelets
Poorly Oriented	Well oriented	Well oriented
Fixed number of bandpass wavelets	Fixed number of bandpass wavelets	Flexible number of bandpass wavelets
Oversampling costly (at odds with the algorithm)	Oversampling costly (at odds with the design)	Oversampling only linearly more expensive
Can use symmetric extension for the edges of images	Can use symmetric extension for the edges of images	Assumes periodic extension of signals

Chapter 4

Scatternets

Scatternets get their own chapter as they have been a very large influence on our work, as well as being quite distinct from the previous discussions on learned methods. They were first introduced by Bruna and Mallat in their work [7], and then were rigorously defined by Mallat in [49]. Several updates and newer models have since been released by Mallat's group, which we will review in this chapter.

It is helpful to introduce this chapter with one further note. Unlike the CNNs introduced in § 2.5, which were set up to minimize some cost function which had certain constraints to promote certain properties, the scattering operator may be thought of as an operator Φ which has some desirable properties for image understanding. These properties may ultimately help us minimize some cost function and improve our image understanding system, which we explore more in **??**.

4.1 Translation Invariant Scatternets

The translation invariant Scatternets were mostly covered in [8]. This section summarises the method of this paper.

4.1.1 Defining the Properties

The first release of Scatternets aimed at building a translation invariant operator, which was also stable to additive noise and deformations. Translation is often defined as being uninformative for classification — an object appearing in the centre of the image should be treated the same way as an the same object appearing in the corner of

Figure 4.1: A Lipschitz continuous function is shown. There is a cone for this function (shown in white) such that the graph always remains entirely outside the cone as it's shifted across. The minimum gradient needed for this to hold is called the 'best Lipschitz constant'.

an image, i.e., Φx is invariant to translations $x_c(\mathbf{u}) = x(\mathbf{u} - \mathbf{c})$[1] by $\mathbf{c} = (c_1, c_2) \in \mathbb{R}^2$ if

$$\Phi x_c = \Phi x \tag{4.1.1}$$

Stability to additive noise is another good choice to include in this operator, as it is a common feature in measured signals. Stability is defined in terms of Lipschitz continuity, which is a strong form of uniform continuity for functions, which we briefly introduce here.

Formally, a Lipschitz continuous function is limited in how fast it can change; there exists an upper bound on the gradient the function can take, although it doesn't necessarily need to be differentiable everywhere. The modulus operator $|x|$ is a good example of a function that has a bounded derivative and so is Lipschitz continuous, but isn't differentiable everywhere.

Returning again to stability to additive noise, state that for a new signal $x'(\mathbf{u}) = x(\mathbf{u}) + \epsilon(\mathbf{u})$, there must exist a bounded $C > 0$ s.t.

$$\|\Phi x' - \Phi x\| \leq C\|x' - x\| \tag{4.1.2}$$

The final requirement is to be stable to small deformations. Enough so that we can ignore intra-class variations, but not so invariant that an object can morph into another (in the case of MNIST for example, we do not want to be so stable to deformations that 7s can map to 1s). Formally, for a new signal $x_\tau(\mathbf{u}) = x(\mathbf{u} - \tau(\mathbf{u}))$, where $\tau(\mathbf{u})$ is a non constant displacement field (i.e., not just a translation) that deforms the image, we require a $C > 0$ s.t.

$$\|\Phi x_\tau - \Phi x\| \leq C\|x\| \sup_{\mathbf{u}} |\nabla \tau(\mathbf{u})| \tag{4.1.3}$$

The term on the right $|\nabla \tau(\mathbf{u})|$ measures the deformation amplitude, so the supremum of it is a limit on the global defomation amplitude.

[1] here we adopt a slight variation on 's notation, by using boldface letters to represent vectors, as is the custom in Signal Processing

Figure 4.2: Real, Imaginary and Modulus of complex wavelet convolved with an impulse.

4.1.2 Finding the Right Operator

A Fourier modulus satisfies the first two of these requirements, in that it is both translation invariant and stable to additive noise, but it is unstable to deformations due to the infinite support of the sinusoid basis functions it uses. It also loses too much information — very different signals can all have the same Fourier modulus, e.g. a chirp, white noise and the Dirac delta function all have flat spectra.

Unlike the Fourier modulus, a wavelet transform is stable to deformations due to the grouping together frequencies into dyadic packets [49], however, the wavelet transform is not invariant to shifts.

We saw in Chapter 3 that the modulus of complex, analytic wavelets commuted with shifts. The real and imaginary parts are also commutative with shifts, but these vary much quicker than the modulus (Figure 4.2). Interestingly, the modulus operator, in this case, does not lose any information [80] (due to the redundancies of the wavelet transform), which is why it may be nice to think of it as a *demodulator*.

The modulus can be made fully invariant by integrating, i.e.,:

$$\int Fx(\mathbf{u})d\mathbf{u} = \int |x * \psi_\lambda(\mathbf{u})|d\mathbf{u}$$

is translation invariant. Total invariance to shifts means integrating over the entire function, which may not be ideal as it loses a significant amount of information in doing this. Instead Bruna and Mallat define scales 2^J, over which their operator is invariant to shifts. Now instead of integrating, the output $\|x * \psi_\lambda\|$ is convolved with an averaging window, or conveniently, the scaling function for the chosen wavelet:

$$\phi_{2^J}(\mathbf{u}) = 2^{-2J}\phi(2^{-J}\mathbf{u})$$

Even still, this averaging means that a lot of information is lost from the first layer outputs ($\|x * \psi_\lambda\|$). Bruna and Mallat combat this by also convolving the output with wavelets that cover the rest of the frequency space, giving

$$U[p]x = U[\lambda_2]U[\lambda_1]x = \||x * \psi_{\lambda_1}| * \psi_{\lambda_2}\|$$

The choice of wavelet functions λ_1 and λ_2 is combined into a path variable, $p = (\lambda_1, \lambda_2, \ldots \lambda m)$.

Local invariants can be again computed by convolving this with another scaling function ϕ. The result is now a multiscale scattering transform, with coefficients:

$$S[p]x = U[p]x * \phi_{2^J}(\mathbf{u})$$

A graphical representation of this is shown in Figure 4.3.

4.2 Rotation and Translation Invariant Scatternets

Mallat's group refined their Scatternet architecture by expanding their list of invariants to also include rotation. They also experimented with adding scale invariance in [68], but it was limited to only averaging over scale once, and they were no longer using it in [57], so for brevity we omit it.

This work was done by two authors, each tackling different challenges. The first is texture analysis with Sifre in [67–69, 65], and the second is image classification with Oyallon in [58, 57]. In this section, we outline the properties and structure of this extended Scatternet.

4.2.1 An Important note on Joint vs. Separable Invariants

When building multiple invariants, some thought must be given as to how to combine them — separably or jointly? Let us call the group of operations we want to be invariant to G, with $g \in G$ a single realization from this group — in this case, G is the group of affine transformations. We want our operator Φ to be invariant to all $g \in G$, i.e., $\Phi(gx) = \Phi(x)$. Building separable invariants would mean representing the group as $G = G_2 G_1$ (an assumption of the group, not of our model), and building $\Phi = \Phi_2 \Phi_1$, where Φ_1 is invariant to members of G_1 and covariant to members of G_2, and Φ_2 is invariant to members of G_2. I.e.,

$$\Phi_2(\Phi_1(g_1 g_2 x)) = \Phi_2(g_2 \Phi_1(x)) = \Phi_2(\Phi_1(x)) \tag{4.2.1}$$

An example of this would be in the group G of 2D translations, building horizontal invariance first, then building vertical invariance second. warn about this approach, however, as it cannot capture the action of G_2 relative to G_1. In the case of veritcal and horizontal translations, for example, it would not be able to distinguish if the

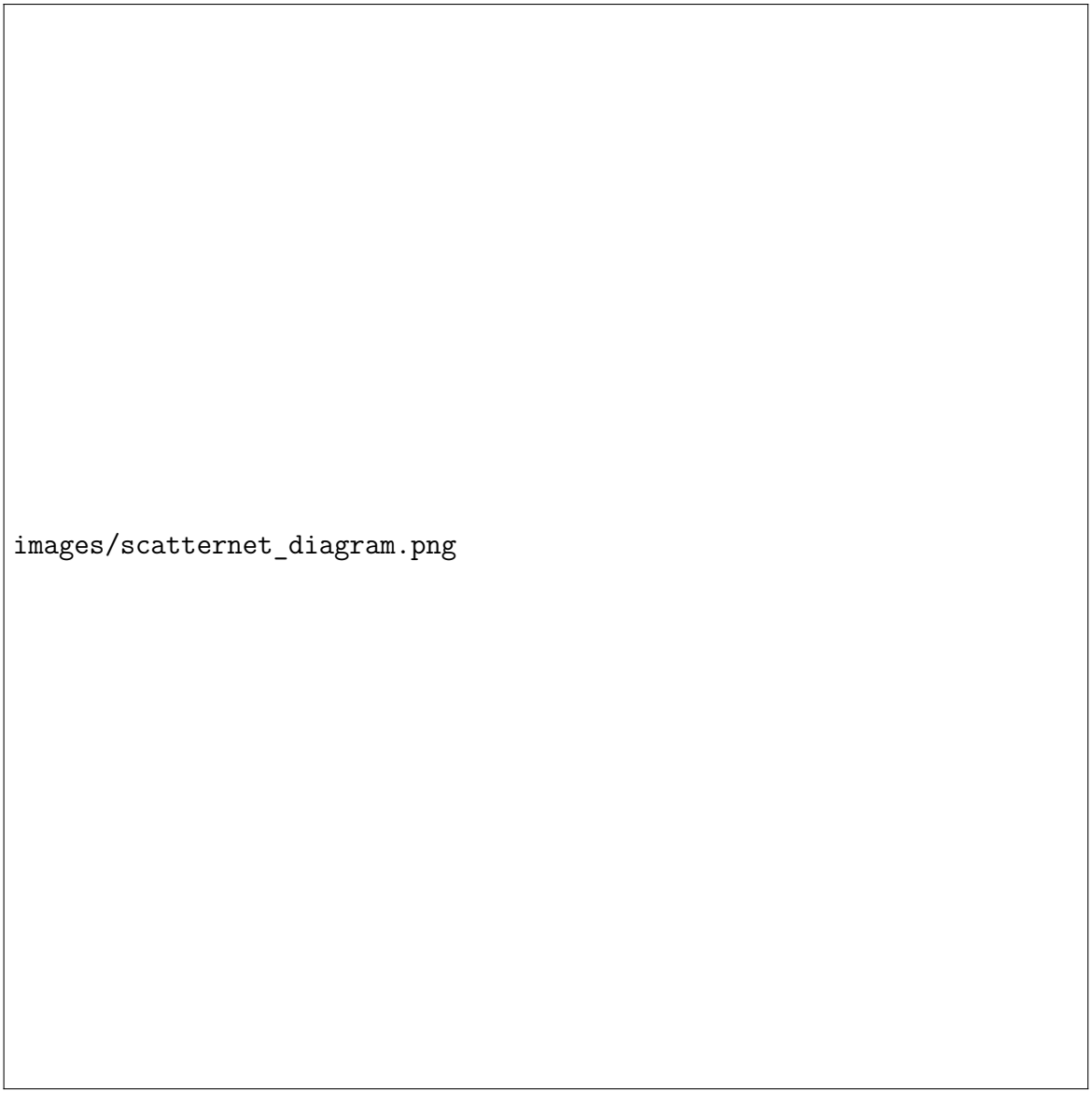

Figure 4.3: The translation invariant Scattering Transform. Scattering outputs are the leftward pointing arrows $S[p]x$, and the intermediate coefficients $U[p]x$ are the centre nodes of the tree. Taken from [8].

patterns had moved apart as well as being shifted, whereas a joint horizontal and vertical translation invariant would be able to distinguish these two cases.

In this vein, suggest that in the case of rotation and translation invariance, a joint invariant should be used, building on the work in [10, 5, 14].

4.2.2 Defining the Properties

A translation $g = (v, \theta)$ of the roto-translation group G_{rt} acting on $\mathbf{u} \in \mathbb{R}^2$ combines translation by v and rotation by R_θ as:

$$g\mathbf{u} = v + R_\theta \mathbf{u} \tag{4.2.2}$$

The product of two successive roto-translations $h = (v', \theta')$ and $g = (v, \theta)$is:

$$gh = (v + R_\theta v', \theta + \theta') \tag{4.2.3}$$

In much the similar approach to the simple translation invariant Scatternet defined above, calculate successive layers of signal coefficients $U[p]x$ that are covariant to the actions of all $g \in G_{rt}$ — i.e.,

$$U[p](gx) = gU[p]x \tag{4.2.4}$$

Creating invariants of order $m = \text{length}(p) = \text{length}([\lambda_1, \lambda_2, \ldots, \lambda_m])$ is then done by averaging $hU[p]x$ for all h in G_{rt}

$$S[p]x(g) = \sum_{h \in G_{rt}} hU[p]x\Phi_J(h^{-1}g) \tag{4.2.5}$$

This convolution averages $hU[p]x$ over all rotation angles in a spatial neighbourhood of $\mathbf{u}$ of size proportional to 2^J.

4.2.3 The Operator

Roto-Translation Invariance

Although we want to have a joint invariant for rotations and translations, this can be done with a cascade of wavelet transforms — so long as the final averaging operation is done over both rotation and translation. do just this, building a 3 layer scattering transform, the first layer of which is exactly identical to the previous translation

scattering transform, i.e.,

$$\tilde{W}_1 x = \Big(x * \phi_J, \{|x * \psi_{\theta,j}|\}\Big) = (S_0 x, U_1 x) \tag{4.2.6}$$

The second and third layers are, however, new. The invariant part of U_1 is computed with an averaging over spatial and angle variables. *This averaging is implemented at fixed scales j* (see our note earlier about choosing separable scale invariance). For an action $g = (v, \theta)$, the averaging kernel is defined as:

$$\Phi_J(g) = \bar{\phi}(\theta) * \phi_J(u) \tag{4.2.7}$$

Where $\phi_J(u)$ is a kernel that averages each $U_1 x$ over scale 2^J, and $\bar{\phi}(\theta = (2\pi)^{-1})$ averages the result of that average over all angles.

To clarify, we look at an example architecture with $J = 2$ scales and $L = 4$ orientations. The output of the first layer $U_1 x$ would be a set of coefficients:

$$U_1 x = \Big\{|x * \psi_{j,\theta}| \,\Big|\, j = \{0,1\}, \theta = k\pi/4, k = \{0,1,2,3\},\Big\} \tag{4.2.8}$$

i.e., there would be 4 high frequency coefficients, which were created with wavelets centred at $|\boldsymbol{\omega}| = 3\pi/4$, and 4 medium frequency components created with wavelets centred at $|\boldsymbol{\omega}| = 3\pi/8$. Each of these 8 will be averaged across the entire image, then each pair of 4 will be averaged across all 4 rotations, leaving 2 invariants.

To recover the information lost from averaging, also convolve $U_1 x$ with corresponding rotation and scale wavelets to pass on the high frequency information. These roto-translation wavelets, while joint, can also be computed with the cascade of separable wavelets. It may be helpful to consider the spatial variable $\mathbf{u}$ as single dimensional, and consider the rotation variable θ as a second dimension. The above equation calculated the low-low frequency component of these two variables, the remaining components are the low-high, high-low, and high-high.

We define the low frequency spatial scaling functions $\phi_J(u)$[2], the spatial wavelets $\psi_{\theta,j}(u)$, the rotation scaling function $\bar{\phi}(\theta)$ (which is just the constant $(2\pi)^{-1}$, but we write out in generic form nonetheless), and the rotation wavelet $\bar{\psi}_k(\theta)$, which is a 2π periodic wavelet.

[2] we temporarily drop the boldface from the spatial parameter u to make it clearer it can be considered as single dimensional

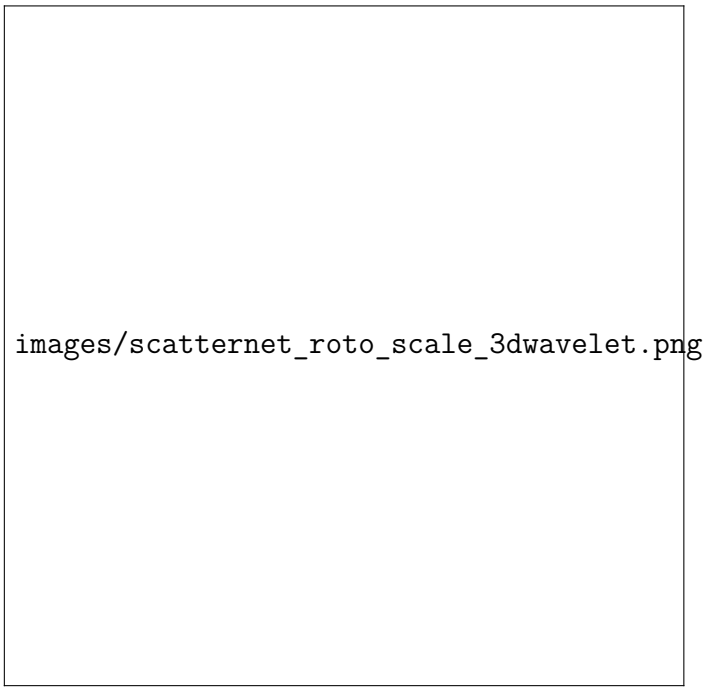

Figure 4.4: Three dimensional convolution with $\Psi_{\theta_m,j_m,k_m}(u_1,u_2,\theta)$ factorised into a two dimensional convolution with $\psi_{\theta_m,j_m}(u_1,u_2)$ and a one dimensional convolution with $\psi_{k_m}(\theta)$. Colours represent the amplitude of the 3D wavelet. Image taken from [68].

Then, the remaining low-high, high-low, and high-high information is:

$$\Psi_{0,J,k_2}(u,\theta) = \phi_J(u) * \bar{\psi}_{k_2}(\theta) \tag{4.2.9}$$

$$\Psi_{\theta_2,j_2,}(u,\theta) = \psi_{\theta_2,j_2}(u) * \bar{\phi}(\theta) \tag{4.2.10}$$

$$\Psi_{\theta_2,j_2,k_2}(u,\theta) = \psi_{\theta_2,j_2}(u) * \bar{\psi}_{k_2}(\theta) \tag{4.2.11}$$

The k parameter is newly introduced here, and it represents the number of scales the rotation wavelet has (a typical value used by was $K = 3$). We call this combined operator Ψ_{θ_m,j_m,k_m}. See Figure 4.4 for what this looks like.

The wavelet-modulus operator then is:

$$\tilde{W}_m Y = \Big(Y * \Phi_J(g), |Y * \Psi_{\theta_m,j_m,k_m}(g)|\Big) \tag{4.2.12}$$

for $m \geq 2$ and the final third order roto-translation Scatternet is:

$$Sx = (x * \phi_J(\mathbf{u}), U_1 x * \Phi_J(p_1), U_2 x * \Phi_J(p_2)) \tag{4.2.13}$$

with $p_1 = (\mathbf{u}, \theta_1, j_1)$ and $p_2 = (\mathbf{u}, \theta_1, j_1, \theta_2, j_2, k_2)$.

Chapter 5

Front Ends

This chapter is about having a wavelet transform/scatternet front end to a deep learning system.

Previous work including Oyallon and Singh have explored ways in which ScatterNets can be improved on for image analysis tasks. This chapter explores some alternate methods we have.

5.1 Fast GPU Implementation

5.2 Something

Where to modify the scatternet? We could look at the difference in training cost when we drop the second order coefficients.

Perhaps just doing the wavelet transform and complex magnitude is enough?

Nick wanted me to move the magnitude before mixing the channels. The argument being that taking the magnitude means the output becomes shift invariant. I.e., we only need to see that there is energy in a wavelet band, and then we can accommodate shifts.

The question is though, how much do we need the phase? Also can we impose some priors on the distribution of subbands for CNN activations? We certainly can for the image statistics.

5.3 Relu Properties

What is the effect of the ReLU on the activations? Show that it sparsifies things. This wasn't really the case for resnet, and the sparsity stayed at around 50% the whole way through.

5.4 Activation Statistics

I want to take the DTCWT of activations throughout a neural network and look at the pdf over the subband energy.

5.5 Wavelet CNNs

I wonder if it's worth looking at the work of [18] and doing something similar to that

Chapter 6

Conclusion

This chapter aims to logically tie together the results from the previous chapter, outlining what has been promising and what has not been, offering explanations as to why we think that is the case.

6.1 Discussion of Attempts so Far

6.1.1 Multiscale Scatternets + SVM

Much of the beginning of the project was spent understanding and analysing the Scatternet design proposed by Mallat and the proposed DT$\mathbb{C}$WT based multiscale Scatternet. It was found that the DT$\mathbb{C}$WT based Scatternets are much faster than the Mallat based Scatternets (3–4 times faster for tiny images, and 10–15 times faster for medium resolution images). Work by a colleague — Singh and Kingsbury [71] showed that this came at the cost of no appreciable loss of performance.

Attempts to use the four layer multiscale Scatternet with linear SVMs on CIFAR-10 data showed that little performance was gained from lots of extra coefficients from the fourth layer. For example, the H4 block could easily be removed with no appreciable difference in performance, but saving one third of the coefficients. Nonetheless, it was difficult to improve this accuracy past 70% without making use of further feature extraction methods (orthogonal least squares was used in both [57] and [71]).

This shows the multiscale Scatternet alone is not achieving the same quality of feature extraction as a few convolutional layers in a CNN, as simply connecting a classifier on the output achieved a lower test accuracy.

We believe that this was because the initial Scatternet design lacks the fully connectedness in the depth dimension that comes with the CNN. This would allow a

Figure 6.1: Comparison of test accuracy for our network vs a two layer CNN. The network used for the Scatternet+CNN design was the reduced Scatternet with data augmentation (see **??**). The standard CNN reference architecture was used (see **??**). Our network trains much faster but plateaus earlier than the CNN network. We must find out how to bring the asymptotes of these two curves closer together.

network to learn how to add linear combinations of different output activations from the layer below. At the simplest case, at layer two, this would allow the CNN to have a filter that could combine two oriented wavelets, perhaps at a slight offset from each other, allowing it to detect a corner.

6.1.2 Reduced Multiscale Scatternets and CNNs

Focus switched in the latter half of the year to bringing back in one layer of CNNs with a scaled down multiscale Scatternet, followed by a neural network classifier. Our initial results were quite poor compared to the pure CNN solution — a 10% reduction in accuracy from 87% to 77%. This was still an improvement on the $\sim 69\%$ we were getting with the pure SVM method which, while comforting, does little to meet our project goals. However, subsequent refinement of our methods narrowed this gap significantly and started to show some promising results.

As a side note, poor initial results do not necessarily indicate failings of the Scatternet. There are a huge variety of hyperparameters to choose when designing the convolutional side of our hybrid network, and on top of that, a selection of different extra layers/schemes that can be used to 'normalize' the statistics of the data, all to promote learning. Replacing one layer of an optimized CNN with a Scatternet and getting a reduction in accuracy could be due to the later layers of the CNN now no longer being optimized. In fact, it was very surprising we had such a large reduction in accuracy. The ability for us to eventually narrow the gap only reinforces the necessity of the project, to better understand how we should train CNNs.

Quicker Training Time

Our Scatternet and shallow CNN was able to train much faster (in fewer epochs) than a pure CNN method. We regard this as an early sign that we are on the right track. Unfortunately, our network does plateau earlier than a CNN, so we end up performing worse in the long run — see Figure 6.1.

Data Augmentation

The other interesting thing we have found so far is our model is far less dependent on data augmentation methods. In particular, taking out the random cropping and shifting from the pure CNN method resulted in a drastic reduction of performance, while it only marginally affected the Scatternet plus CNN performance.

6.1.3 Improved Analysis Methods

One benefit of having set filters in the Scatternet design versus purely learned ones can be seen in the difference between the axes labels of **??** and **??**. We were able to label the slices in the Scatternet design, see that some of the slices were not being used much in the learned CNN, and design experiments that removed these slices, which improved training time and accuracy. This kind of targetted design would be hugely beneficial to the field of CNNs.

Unfortunately, we have not yet fine tuned our visualizations. The figures in **??** are only toy examples at this stage. They show that it is possible to get images like those from the work of Zeiler and Fergus [82] from a subset of Scatternet coefficients.

6.2 Future Work

6.2.1 Closing the Gap

The first layer of a CNN *can* be replaced with a first order Scatternet, with currently a small loss of performance. Getting this result is promising, but it needs further work. If we were to reduce this gap, we would have developed a network that could train *faster* than current methods. While this is not our primary research goal, it will nonetheless be a useful addition to the field.

We need to spend some time researching the cause for the current performance gap. We must revisit the first layers of the CNN and draw inspiration from here. What is missing?

The basis functions for the first order Scatternet are considerably fewer in number than used by the CNN. We are currently using 24 compared to 64 in the pure CNN network. While relying on fewer operations is beneficial, this may be an easy way to increase our final accuracy. Two possibilities immediately come to mind — we could add in more colour channels as currently, the assumption is that only low frequency is necessary for colour, but there certainly are some mid-low level frequency colour

filters in both the AlexNet first layer **??**, and the CIFAR-10 first layer **??**. Also, we could increase the number of scales we have per octave from one to two, to get a better coverage of the frequency space.

Unfortunately, both of these additions would mean we would lose perfect reconstruction, but that does not mean we cannot still invert the representations, it just becomes slightly more difficult.

6.2.2 Moving to a new Dataset

We believe that we are not playing wavelets to their strengths by using such small images as the ones found in CIFAR-10. Further, they are very poor representations of the images found in the real world, and it would be lackluster to restrict ourselves to them. Fortunately, there are no shortages in choosing an alternative dataset, with PASCAL-VOC a good candidate due to its similar number of training samples.

This is best done sooner rather than later. It will take time to get used to working on a new dataset, and then some more to fine tune our design. As a first step, we will look at a design like AlexNet, replacing the first layer with a Scatternet, as we have done for CIFAR-10.

6.2.3 Improved Visualizations

We believe that visualizations are key to unlocking the secrets of CNNs. Our attempts so far at creating visualizations have taught us a great deal, but further work needs to be done. In particular, a recent paper by Grun et al. [21] has given a detailed comparison and analysis of how visualizations have developed since the work of Zeiler and Fergus [82]. We must spend time to study these works and use them to improve our visualizations method with Scatternets.

6.2.4 Going Deeper

To really indicate a gained insight, we need to be able to replace more than one layer of a CNN.

This does not mean that Scatternets are a poor feature extractor, they have already been proven to be state of the art in texture datasets [68, 69], but they do not help us gain insight into CNNs.

As we now have a framework that successfully mimics one layer of a CNN. We believe that the visualization tools that we are currently working on will be the key to

Figure 6.2: Possible future work with residual mappings. **??** shows the residual layer as used by [24]. These networks started with the assumption that the the identity mapping $f(\cdot) = I$ was a good reference function, and the network should learn deviations from that. **??** shows the proposed architecture. Assuming the first order Scatternet (wavelet transform + modulus + averaging) is a good base mapping, improve on it by allowing some deviation from it as a residual.

unlocking more information about the deeper layers of CNNs, which will in turn allow us to design enhancements to the Scatternets.

6.2.5 Resiudal Scattering Layers

This somewhat continues on from § 6.2.1, but also could lead to a complete rethink of the Scatternet design. We want to investigate to achieve this is inspired from the recent state of the art ResNets — mentioned in § 2.5.10. Residual networks work on the basis that if the ideal mapping $\mathcal{H}(x)$ is close to the identity mapping, then it will be easier to learn $\mathcal{F}(x) := \mathcal{H}(x) - x$ than to learn $\mathcal{H}(x)$ directly. To do this, they construct a residual layer, shown again in **??**.

Assuming we have developed a near-optimal mapping with a first order Scatternet, S_J^1, then a sensible thing to do is to improve performance by adding in extra flexibility and the ability to better fit the nuances of the training set. We propose to do that with the residual layer shown in **??**.

6.2.6 Exploring the Scope of Scatternets

We believe that applications a designed architecture like the Scatternet is well suited for are unsupervised and low data tasks. More specifically, any method that is difficult to propagate back gradients (due to the lack of labelled data, or to only a few training points). We would like to spend some time analysing what progress can be made in these fields with the knowledge we have.

6.2.7 Analysing Newer CNN Layers

For the moment we have satisfied ourselves with examining architectures like Cuda Convnet and AlexNet, which while high end, are not state of the art. There have been many new layers and designs added in since then, which have taken modern CNNs even further. We have already covered one example, the Residual Unit. Another example

is the 'Inception Unit' [75, 76], which combines 5×5, 3×3, and 1×1 convolutions of different.

Also, there is a new trend of using stride-2 convolution with downsampling is something we would like to investigate more, as it could lead us to clues about how to use subsampling and scale in a CNN, ideas that fit very naturally in a wavelet based network.

6.2.8 Revisiting Greedy Layer-Wise Training

The work on using unsupervised Deep Belief Networks to progressively train the hidden units of a deep neural network by Bengio et al. [4] has been largely abandoned recently. This is sad to see, as it looked like a promising, well-defined way to train deep networks. We would like to revisit this in the context of a Scatternet, and attempt to use this paradigm.

6.3 Timeline

We set our timeline for the next two years in Table 6.1. The future work we have discussed so far roughly falls into two categories: the first four (§§ 6.2.1–6.2.4) fall on what we can consider the main line of research into designing a learned CNN style network. The second four (§§ 6.2.5–6.2.8) is more speculative research, that we hope will allow us to explore and gain inspiration from similar problems.

Ordering these tasks chronologically would not be very well thought out. For example, successfully achieving § 6.2.4 is the main goal of the PhD after all, so it would be naive to say we believe we can achieve it by the end of the second year. Instead, we believe that the main line of research will take us a long time, and during that time, we must explore the more speculative research. Also, a few of the tasks will need to be revisited at certain points — just as importantly as not being too ambitious, we should not be too lazy. I.e., we should not leave our first attempt at building deeper layers of a CNN until our third year. The same goes for improving our visualizations.

Table 6.1: Our plan for the remainder of the PhD. The first column lists the months we want to do the work in, and the second column describes the task, and the questions we want to answer in this time.

Second Year		
Date	Task	Second Task
Oct–Nov	Closing the Gap § 6.2.1	
Dec–Jan	Developing ResNet Layers § 6.2.5	
Feb–Mar	Improved Visualizations Research § 6.2.3	
Apr–May	Moving to a new dataset (preparation) § 6.2.2	
Jun–Jul	Moving to a new dataset (testing) § 6.2.2	Going Deeper § 6.2.4
Aug–Sep	Going Deeper § 6.2.4	Recap on Second Year
Third Year		
Oct–Nov	Exploring the Scope of Scattternets § 6.2.6	
Dec–Jan	Revisiting Greedy Layer-Wise Training § 6.2.8	Analysing Newer CNN Layers § 6.2.7
Feb–Mar	Revisiting Greedy Layer-Wise Training§ 6.2.8	Analysing Newer CNN Layers § 6.2.7
Apr–May	Improved Visualizations Research § 6.2.3	
Jun–Jul	Going Deeper § 6.2.4	
Aug–Sep	Going Deeper § 6.2.4	Recap on Third Year
Fourth Year		
Oct-Dec	Writing up	

Bibliography

[1] Anderson, R., Kingsbury, N., and Fauqueur, J. (2005). Determining Multiscale Image Feature Angles from Complex Wavelet Phases. In Kamel, M. and Campilho, A., editors, *Image Analysis and Recognition*, number 3656 in Lecture Notes in Computer Science, pages 490–498. Springer Berlin Heidelberg. 00026.

[2] Anderson, R., Kingsbury, N., and Fauqueur, J. (2006). Rotation-invariant object recognition using edge profile clusters. In *Signal Processing Conference, 2006 14th European*, pages 1–5. IEEE. 00010.

[3] Bendale, P., Triggs, W., and Kingsbury, N. (2010). Multiscale keypoint analysis based on complex wavelets. In *BMVC 2010-British Machine Vision Conference*, pages 49–1. BMVA Press. 00009.

[4] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy Layer-Wise Training of Deep Networks. pages 153–160. 01545.

[5] Boscain, U., Duplaix, J., Gauthier, J.-P., and Rossi, F. (2010). Anthropomorphic image reconstruction via hypoelliptic diffusion. *arXiv:1006.3735 [math]*. 00032.

[6] Bruna, J. (2013). *Scattering Representations for Recognition*. Theses, Ecole Polytechnique X. 00010 Déposée Novembre 2012.

[7] Bruna, J. and Mallat, S. (2011). Classification with scattering operators. In *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1561–1566. 00058.

[8] Bruna, J. and Mallat, S. (2013). Invariant Scattering Convolution Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886. 00150.

[9] Chen, H. and Kingsbury, N. (2012). Efficient Registration of Nonrigid 3-D Bodies. *IEEE Transactions on Image Processing*, 21(1):262–272. 00011.

[10] Citti, G. and Sarti, A. (2006). A Cortical Based Model of Perceptual Completion in the Roto-Translation Space. *Journal of Mathematical Imaging and Vision*, 24(3):307–326. 00224.

[11] Coifman, R. R. and Donoho, D. L. (1995). Translation-Invariant De-Noising. In Antoniadis, A. and Oppenheim, G., editors, *Wavelets and Statistics*, number 103 in Lecture Notes in Statistics, pages 125–150. Springer New York. 02529.

[12] de Rivaz, P. and Kingsbury, N. (1999). Complex wavelet features for fast texture image retrieval. In *1999 International Conference on Image Processing, 1999. ICIP 99. Proceedings*, volume 1, pages 109–113 vol.1. 00086.

[13] de Rivaz, P. and Kingsbury, N. (2001). Bayesian image deconvolution and denoising using complex wavelets. In *2001 International Conference on Image Processing, 2001. Proceedings*, volume 2, pages 273–276 vol.2. 00044.

[14] Duits, R. and Burgeth, B. (2007). Scale Spaces on Lie Groups. In Sgallari, F., Murli, A., and Paragios, N., editors, *Scale Space and Variational Methods in Computer Vision*, volume 4485, pages 300–312. Springer Berlin Heidelberg, Berlin, Heidelberg. 00000.

[15] Fauqueur, J., Kingsbury, N., and Anderson, R. (2006). Multiscale keypoint detection using the dual-tree complex wavelet transform. In *Image Processing, 2006 IEEE International Conference On*, pages 1625–1628. IEEE. 00060.

[16] Field, D. J. (1993). Scale-invariance and self-similar 'wavelet' transforms: An analysis of natural scenes and mammalian visual systems. *ResearchGate*. 00216.

[17] Field, D. J. (1994). What Is the Goal of Sensory Coding? *Neural Computation*, 6(4):559–601. 01262.

[18] Fujieda, S., Takayama, K., and Hachisuka, T. (2018). Wavelet Convolutional Neural Networks. *arXiv:1805.08620 [cs]*. 00007.

[19] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. 00649.

[20] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323. 00526.

[21] Grun, F., Rupprecht, C., Navab, N., and Tombari, F. (2016). A Taxonomy and Library for Visualizing Learned Features in Convolutional Neural Networks. *arXiv:1606.07757 [cs]*. 00000.

[22] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., and Wang, G. (2015). Recent Advances in Convolutional Neural Networks. *arXiv:1512.07108 [cs]*. 00002.

[23] Hatipoglu, S., Mitra, S. K., and Kingsbury, N. (1999). Texture classification using dual-tree complex wavelet transform. In *Seventh International Conference on Image Processing and Its Applications*, pages 344–347. 00064.

[24] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*. 00013.

[25] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity Mappings in Deep Residual Networks. *arXiv:1603.05027 [cs]*. 00021.

[26] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*. 01289.

[27] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*. 00137.

[28] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153. 00640.

[29] Kingsbury, N. (1998a). The dual-tree complex wavelet transform: A new efficient tool for image restoration and enhancement. In *Signal Processing Conference (EUSIPCO 1998), 9th European*, pages 1–4. 00345.

[30] Kingsbury, N. (1998b). The Dual-Tree Complex Wavelet Transform: A New Technique For Shift Invariance And Directional Filters. In *1998 8th International Conference on Digital Signal Processing (DSP)*, pages 319–322, Utah. 00586.

[31] Kingsbury, N. (1999a). Image processing with complex wavelets. *Philosophical Transactions of the Royal Society a-Mathematical Physical and Engineering Sciences*, 357(1760):2543–2560. 00870.

[32] Kingsbury, N. (1999b). Shift invariant properties of the dual-tree complex wavelet transform. In *Icassp '99: 1999 Ieee International Conference on Acoustics, Speech, and Signal Processing, Proceedings Vols I-Vi*, pages 1221–1224. 00234.

[33] Kingsbury, N. (2000). A dual-tree complex wavelet transform with improved orthogonality and symmetry properties. 00355.

[34] Kingsbury, N. (2001). Complex wavelets for shift invariant analysis and filtering of signals. *Applied and Computational Harmonic Analysis*, 10(3):234–253. 01524.

[35] Kingsbury, N. (2003). Design of Q-shift complex wavelets for image processing using frequency domain energy minimization. In *2003 International Conference on Image Processing, 2003. ICIP 2003. Proceedings*, volume 1, pages I–1013–16 vol.1. 00115.

[36] Kingsbury, N. and Magarey, J. (1997). Wavelet transforms in image processing. 00076.

[37] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Technical report. 00757.

[38] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, pages 1097–1105. Curran Associates, Inc. 03570.

[39] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551. 01508.

[40] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 05502.

[41] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient BackProp. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, number 7700 in Lecture Notes in Computer Science, pages 9–48. Springer Berlin Heidelberg. 01085.

[42] Lennie, P. (2003). The cost of cortical computation. *Current biology: CB*, 13(6):493–497. 00627.

[43] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014). Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*. 00500.

[44] Loo, P. and Kingsbury, N. G. (2001). Motion-estimation-based registration of geometrically distorted images for watermark recovery. pages 606–617. 00035.

[45] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999*, volume 2, pages 1150–1157 vol.2. 11668.

[46] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110. 32503.

[47] Mahendran, A. and Vedaldi, A. (2015). Understanding Deep Image Representations by Inverting Them. *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 00042.

[48] Mallat, S. (1998). *A Wavelet Tour of Signal Processing*. Academic Press. 17547.

[49] Mallat, S. (2012). Group Invariant Scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398. 00135.

[50] Miller, M. and Kingsbury, N. (2008). Image denoising using derotated complex wavelet coefficients. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, 17(9):1500–1511. 00069.

[51] Mishkin, D. and Matas, J. (2015). All you need is a good init. *arXiv:1511.06422 [cs]*. 00012.

[52] Mishkin, D., Sergievskiy, N., and Matas, J. (2016). Systematic evaluation of CNN advances on the ImageNet. *arXiv:1606.02228 [cs]*. 00000.

[53] Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. 00861.

[54] Ng, E. S. and Kingsbury, N. G. (2012). Robust pairwise matching of interest points with complex wavelets. *Image Processing, IEEE Transactions on*, 21(8):3429–3442. 00006.

[55] Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609. 04099.

[56] Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325. 02433.

[57] Oyallon, E. and Mallat, S. (2015). Deep Roto-Translation Scattering for Object Classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2865–2873. 00011.

[58] Oyallon, E., Mallat, S., and Sifre, L. (2013). Generic Deep Networks with Wavelet Scattering. *arXiv:1312.5940 [cs]*. 00006.

[59] Ren, S., He, K., Girshick, R., Zhang, X., and Sun, J. (2015). Object Detection Networks on Convolutional Feature Maps. *arXiv:1504.06066 [cs]*. 00008.

[60] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. pages 318–362. MIT Press, Cambridge, MA, USA. 00087.

[61] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252. 01188.

[62] Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120 [cond-mat, q-bio, stat]*. 00087.

[63] Selesnick, I. (2001). Hilbert transform pairs of wavelet bases. *IEEE Signal Processing Letters*, 8(6):170–173. 00351.

[64] Selesnick, I. W., Baraniuk, R. G., and Kingsbury, N. G. (2005). The dual-tree complex wavelet transform. *Signal Processing Magazine, IEEE*, 22(6):123–151. 01602.

[65] Sifre, L. (2014). *Rigid-Motion Scattering for Image Classification*. PhD Thesis, Ecole Polytechnique. 00000.

[66] Sifre, L. and Anden, J. (2013). ScatNet. 00004.

[67] Sifre, L. and Mallat, S. (2012). Combined scattering for rotation invariant texture analysis. In *European Symposium on Artificial Neural Networks (ESANN) 2012*. 00032.

[68] Sifre, L. and Mallat, S. (2013). Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination. In *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1233–1240. 00053.

[69] Sifre, L. and Mallat, S. (2014). Rigid-Motion Scattering for Texture Classification. *arXiv:1403.1687 [cs]*. 00003.

[70] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*. 01664.

[71] Singh, A. and Kingsbury, N. (2017). Multi-Resolution Dual-Tree Wavelet Scattering Network for Signal Classification. *arXiv:1702.03345 [cs]*.

[72] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*. 00084.

[73] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958. 00417.

[74] Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv:1602.07261 [cs]*. 00000.

[75] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015a). Going Deeper With Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9. 00569.

[76] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015b). Rethinking the Inception Architecture for Computer Vision. *arXiv:1512.00567 [cs]*. 00047.

[77] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*. 00223.

[78] Tang, Y. (2013). Deep Learning using Linear Support Vector Machines. *arXiv:1306.0239 [cs, stat]*. 00065.

[79] Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2015). Efficient Object Localization Using Convolutional Networks. *CVPR 2015*. 00009.

[80] Waldspurger, I., d'Aspremont, A., and Mallat, S. (2012). Phase Recovery, MaxCut and Complex Semidefinite Programming. *arXiv:1206.0102 [math]*. 00125.

[81] Zeiler, M., Taylor, G., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *2011 IEEE International Conference on Computer Vision (ICCV)*, pages 2018–2025. 00138.

[82] Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, pages 818–833. 00000.

[83] Zhang, G. and Kingsbury, N. (2015). Variational Bayesian image restoration with group-sparse modeling of wavelet coefficients. *Digital Signal Processing*, 47:157–168. 00001.

[84] Zhang, Y. and Kingsbury, N. (2008). A Bayesian wavelet-based multidimensional deconvolution with sub-band emphasis. In *Engineering in Medicine and Biology Society*, pages 3024–3027. 00007.