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

Background

This thesis combines work in several fields. We provide a background for the most important and relevant fields in this chapter. We first introduce the definition and properties of the Wavelet Transforms, focussing on the DT$\mathbb{C}$WT, then we describe the fundamentals of Convolutional Neural Networks and how they learn, and finally we introduce the Scattering Transform, the original inspiration for this thesis.

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 $u_2 \in 0, 1, \dots W-1$.

- **Convolutional networks**
  Image convolutional neural networks often work with 4-dimensional arrays. In particular, mini-batches of images with multiple channels. When we need to, we index over the minibatch with the variable n and over the channel dimension with c. For example, we can index an activation $\mathbf{x}$ by $\mathbf{x}[n, c, u_1, u_2]$.

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

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

- **Wavelet Filter Banks**
  Using standard notation, we define the scaling function as ϕ and the wavelet function as ψ. For a filter bank implementation of a wavelet transform, we use h for analysis and g for synthesis filters.

  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.

2.2 The Fourier and Wavelet Transforms

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 simple experiments to demonstrate this is shown in Figure 2.1.

2.2.1 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 section 3.1.

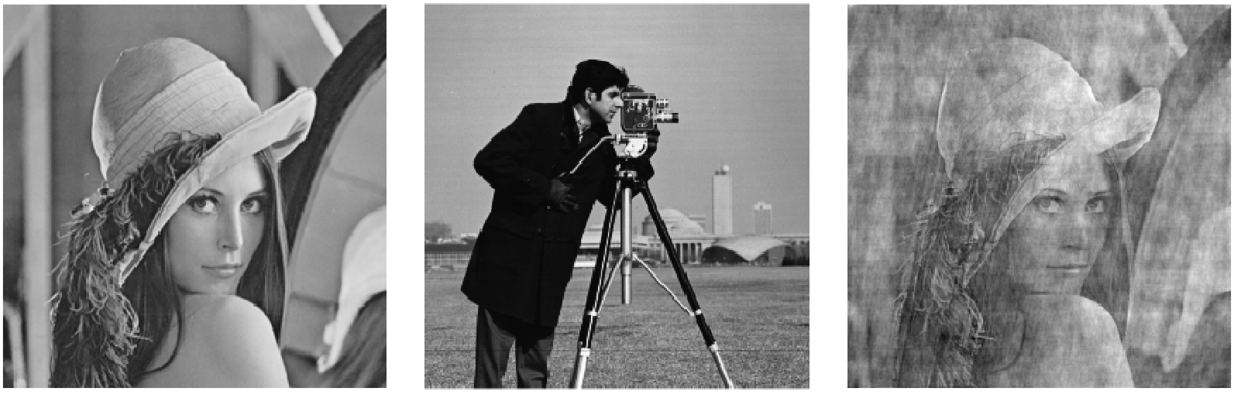


Figure 2.1: **Importance of phase over magnitude for images.** The phase of the Fourier transform of the first image is combined with the magnitude of the Fourier transform of the second image and reconstructed. Note that the first image has entirely won out and nothing is left visible of the cameraman.

2.2.2 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.

2.2.3 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 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 2.2.

- 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 2.3.

- They are not shift invariant. In particular, small shifts greatly perturb the wavelet coefficients. Figure 2.2 shows this for the centre-left and centre-right images. Figure 2.11 (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 [1, 2], 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.

2.2.4 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:

$$e^{j\omega t} = \cos(\omega t) + j\sin(\omega t) \tag{2.2.1}$$

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.

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

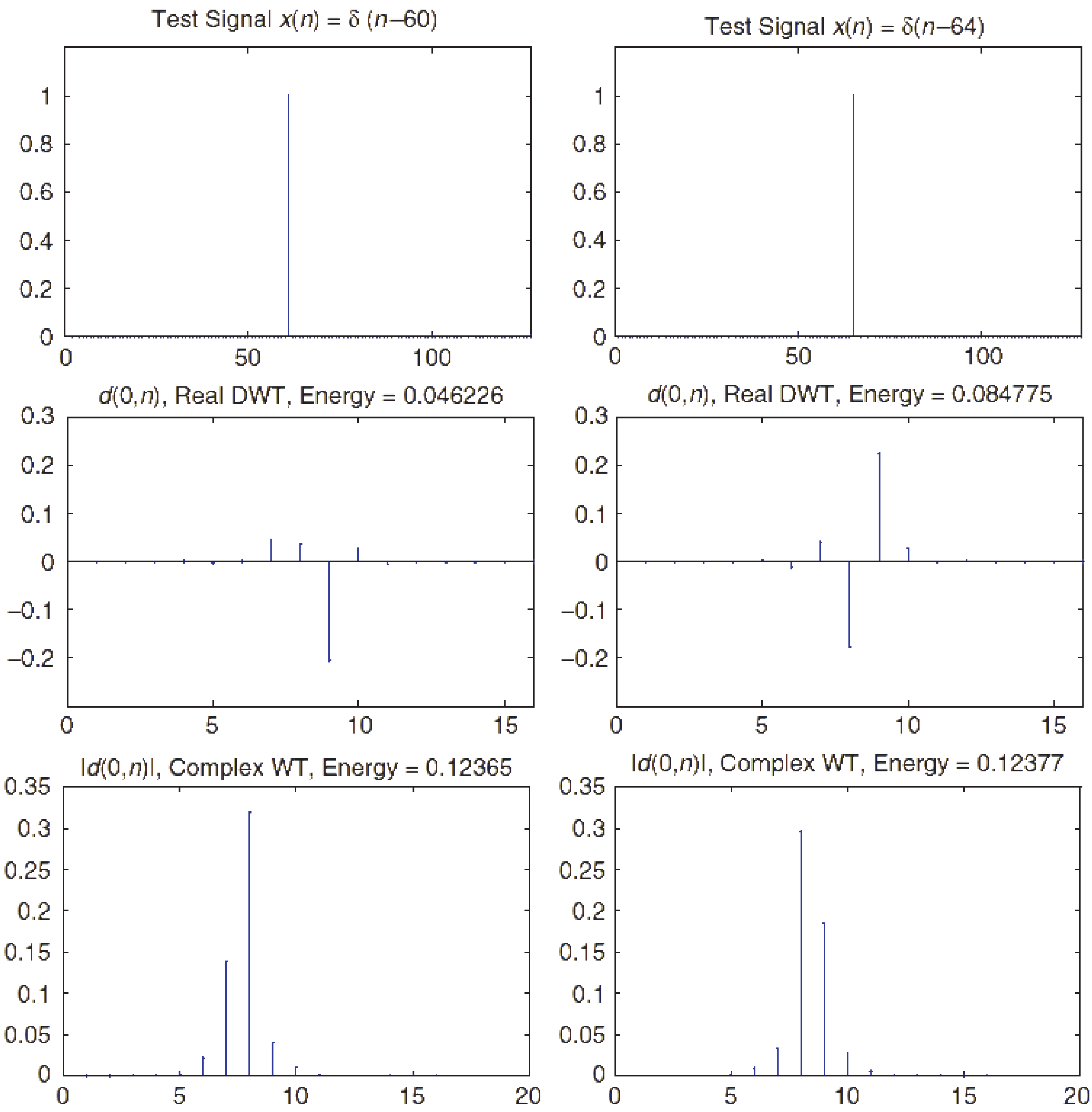


Figure 2.2: **Sensitivity of DWT coefficients to zero crossings and small shifts.** Two impulse signals $\delta(n-60)$ and $\delta(n-64)$ are shown (top), as well as the wavelet coefficients for scale $j=1$ for the DWT (middle) and for the DTℂWT (bottom). In the middle row, not only are the coefficients 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 DTℂWT coefficients has remained far more constant, as has the shape of the envelope of the output. Image taken from [3].

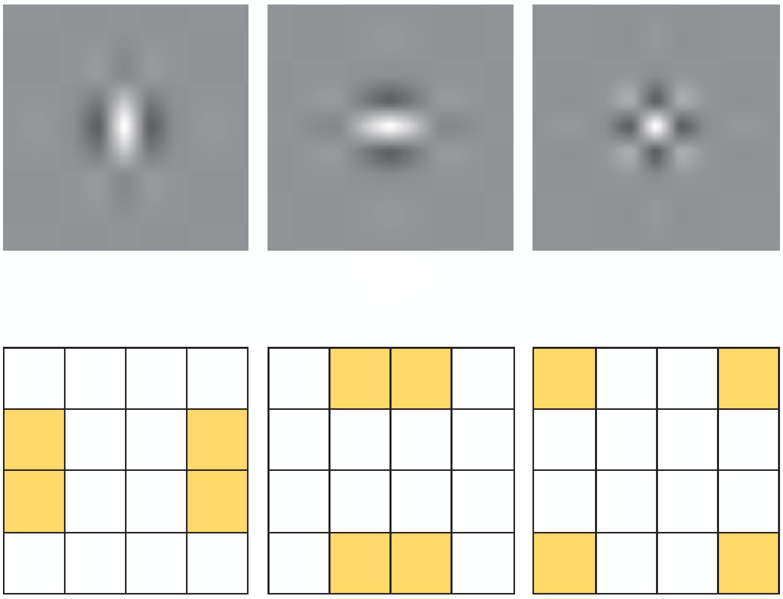


Figure 2.3: **Typical wavelets from the 2D separable DWT.** Top: Wavelet point spread functions for the low-high, high-low, and high-high wavelets. High-high wavelets are in a checkerboard pattern, with no favoured orientation. Bottom: Idealized support of the spectra of each of the wavelets. Image taken from [3].

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:

$$\psi_c(t) = \psi_r(t) + j\psi_c(t) \tag{2.2.2}$$

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 [4–12], and the separable, filter bank based DTℂWT developed by Kingsbury [13–19, 3].

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

$$\psi(\mathbf{u}) = \frac{1}{2\pi\sigma^2}(e^{i\mathbf{u}\xi} - \beta)e^{-\frac{|\mathbf{u}|^2}{2\sigma^2}} \tag{2.2.3}$$

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:

$$\psi(\mathbf{u}) = \frac{\gamma}{2\pi\sigma^2}(e^{i\mathbf{u}\xi} - \beta)e^{-\mathbf{u}^\mathbf{t}\Sigma^{-1}\mathbf{u}} \tag{2.2.4}$$

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 2.4. 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 2.2.4:

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

combine these two variables into a single coordinate

$$\lambda = (\theta, j/Q) \tag{2.2.5}$$

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:

$$\psi_\lambda(\mathbf{u}) = 2^{-j/Q}\psi(2^{-j/Q}R_\theta^{-1}\mathbf{u}) \tag{2.2.6}$$

Implementation

The Fourier Implementation of this Morlet decomposition is shown in Figure 2.6. It is based on the fact that

$$\mathcal{F}(x * \psi)(\omega) = \mathcal{F}x(\omega)\mathcal{F}\psi(\omega) \tag{2.2.7}$$

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:

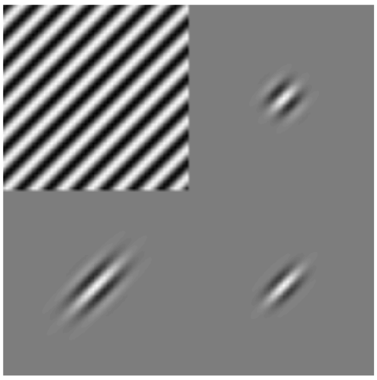


Figure 2.4: **Single Morlet filter with varying slants and window sizes.** Top left — 45° plane wave (real part only). Top right — plane wave with $\sigma = 3, \gamma = 1$. Bottom left — plane wave with $\sigma = 3, \gamma = 0.5$. Bottom right — plane wave with $\sigma = 2, \gamma = 0.5$.

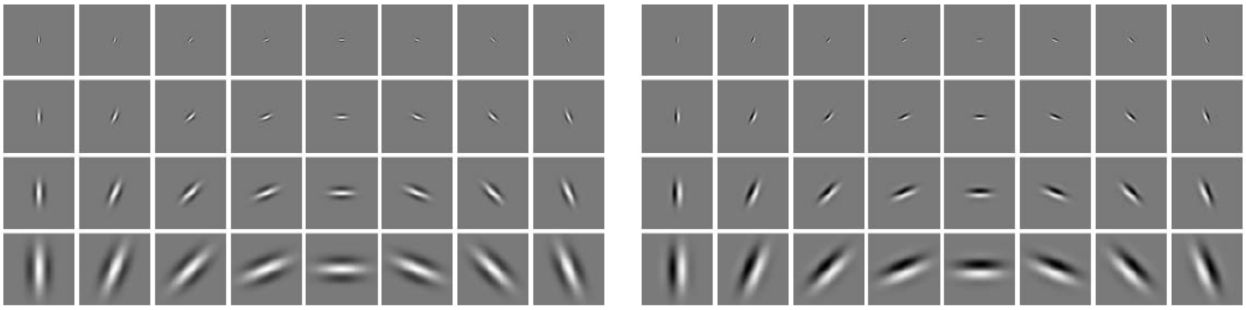


Figure 2.5: **The full dictionary of Morlet wavelets used by Mallat.** The real filters are on the left and the imaginary on the right. The first row correspond to scale $j = 1$, increasing up to $j = 4$. The first column corresponding to $\theta = 0$, rotating through $\pi/8$ up to the eighth column of $7\pi/8$, $\gamma = 1/2$.

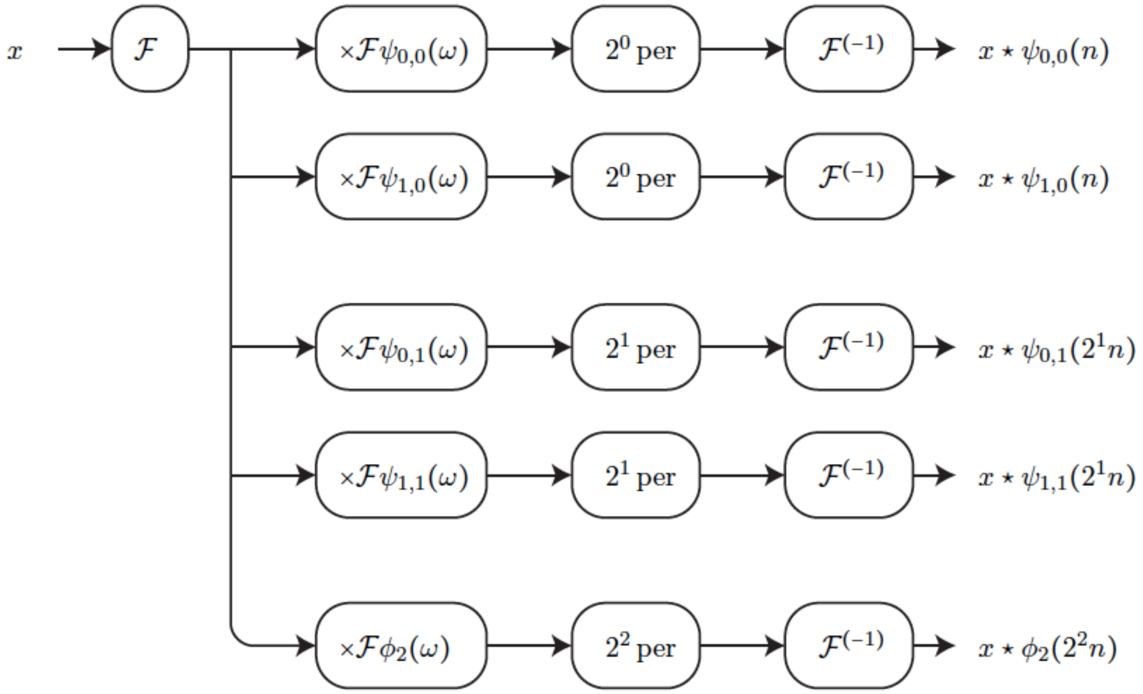

x
$\mathcal{F}$
$\times\mathcal{F}\psi_{0,0}(\omega)$
2^0 per
$\mathcal{F}^{(-1)}$
$x \star \psi_{0,0}(n)$
$\times\mathcal{F}\psi_{1,0}(\omega)$
2^0 per
$\mathcal{F}^{(-1)}$
$x \star \psi_{1,0}(n)$
$\times\mathcal{F}\psi_{0,1}(\omega)$
2^1 per
$\mathcal{F}^{(-1)}$
$x \star \psi_{0,1}(2^1 n)$
$\times\mathcal{F}\psi_{1,1}(\omega)$
2^1 per
$\mathcal{F}^{(-1)}$
$x \star \psi_{1,1}(2^1 n)$
$\times\mathcal{F}\phi_2(\omega)$
2^2 per
$\mathcal{F}^{(-1)}$
$x \star \phi_2(2^2 n)$

Figure 2.6: 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 [11].

- $O(N^2 \log N)$ for the forward FFT of x.

- $O(JLN^2)$ for the multiplication in the frequency domain. We can see this from Figure 2.6, 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:

$$T(N) = O(N^2 \log N) + O(JLN^2) + O(L\sum_j N^2 log2^{-j}N) \tag{2.2.8}$$

Invertibility and Energy Conservation

We can write the wavelet transform of an input x as

$$\mathcal{W}x = \{x * \phi_J, x * \psi_\lambda\}_\lambda \tag{2.2.9}$$

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

$$\|\mathcal{W}x\|^2 = \|x * \phi_J\|^2 + \sum_\lambda \|x * \psi_\lambda\|^2 \tag{2.2.10}$$

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

$$\|\mathcal{W}x\| = \|x\| \tag{2.2.11}$$

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:

$$A(\omega) = |\mathcal{F}\phi_J(\omega)|^2 + \sum_\lambda |\mathcal{F}\psi_\lambda(\omega)|^2 \tag{2.2.12}$$

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

$$1 - \alpha \leq A(\omega) \leq 1 \tag{2.2.13}$$

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 2.7 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 inverse filters are defined by:

$$\begin{aligned} \mathcal{F}\phi_J^{-1}(\omega) &= A(\omega)^{-1}\mathcal{F}\phi_J(\omega) && (2.2.14) \\ \mathcal{F}\psi_\lambda^{-1}(\omega) &= A(\omega)^{-1}\mathcal{F}\psi_\lambda(\omega) && (2.2.15) \end{aligned}$$

[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 2.7: 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 [11].

2.2.6 The DTℂWT

The DTℂWT was first proposed by **(author?)** in [14, 15] 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 DTℂWT is done in [16, 3]. Building on these properties, it been used successfully for denoising and inverse problems [20–23], texture classification [24, 25], image registration [26, 27] and SIFT-style keypoint generation matching [28–32] amongst many other applications.

Compared to Gabor (or Morlet) image analysis, the authors of [3] 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 subsection 2.2.5. The DTℂWT is a filter bank (FB) based wavelet transform. It is faster to implement than the Morlet analysis, as well as being more readily invertible.

Deisgn Criteria for the DTℂWT

It was stated in subsection 2.2.4 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 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 subsection 2.2.4, we want to have a complex mother wavelet ψ_c that satisfies Equation 2.2.2, 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:

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

Figure 2.8: Analysis FB for the DT$\mathbb{C}$WT . 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 [3].

- 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:

$$\phi_h(t) = \sqrt{2}\sum_n h_0(n)\phi_h(2t-n) \tag{2.2.16}$$

$$\psi_h(t) = \sqrt{2}\sum_n h_1(n)\phi_h(2t-n) \tag{2.2.17}$$

$$\phi_g(t) = \sqrt{2}\sum_n g_0(n)\phi_g(2t-n) \tag{2.2.18}$$

$$\psi_g(t) = \sqrt{2}\sum_n g_1(n)\phi_g(2t-n) \tag{2.2.19}$$

This implementation is shown in Figure 2.8.

Designing a filter bank implementation that results in Hilbert symmetric wavelets does not appear to be an easy task. However, it was shown by **(author?)** [16] (and later proved by **(author?)** [33]) that the necessary conditions are conceptually very simple. One low-pass filter must be a *half-sample shift* of the other. I.e.,

$$g_0(n) \approx h_0(n-0.5) \rightarrow \psi_g(t) \approx \mathcal{H}\{\psi_h(t)\} \tag{2.2.20}$$

As the DT$\mathbb{C}$WT is designed as an invertible filter bank implementation, this is only one of the constraints. Naturally, there are also:

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

(a)

(b)

Figure 2.9: (a) The high-high DWT wavelet having a passband in all 4 corners of the frequency plane vs (b) the high-high DTℂWT wavelet frequency support only existing in one quadrant. Taken from [3]

to consider when building the h's and g's. The derivation of the filters that meet these conditions is covered in detail in [19, 34], and in general in [3]. 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.

The Resulting Wavelets and their Properties

While analytic wavelets in 1D are useful for their shift invariance, the real beauty of the DTℂWT is in its ability to make a separable 2D wavelet transform with oriented wavelets.

2.9a shows the spectrum of the wavelet when the separable product uses purely real wavelets, as is the case with the DWT. 2.9b 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:

$$\begin{aligned}\psi(\omega_1,\omega_2) &= \psi_c(\omega_1)\overline{\psi_c(\omega_2)} && (2.2.21)\\ &= (\psi_h(\omega_1)+j\psi_g(\omega_1)\overline{(\psi_h(\omega_2)+\psi_g(\omega_2))} \\ &= \psi_h(\omega_1)\psi_h(\omega_2)+\psi_g(\omega_1)\psi_g(\omega_2) \\ &\quad +j\left(\psi_g(\omega_1)\psi_h(\omega_2)-\psi_h(\omega_1)\psi_g(\omega_2)\right) && (2.2.22)\end{aligned}$$

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 2.2.21 to get the right hand side of the frequency plane.

Figure 2.10 shows the resulting wavelets both in the spatial domain and their idealized support in the frequency domain.

Figure 2.11 shows how the DTℂWT compares with the DWT with a shifting input.

[4] note that 2.9b shows the 135° wavelet

Figure 2.10: Wavelets from the 2d DTℂWT . **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 [3].

Figure 2.11: The shift invariance of the DTℂWT (left) vs. the real DWT (right). The DTℂWT linearizes shifts in the phase change of the complex wavelet. Image taken from [14].

Implementation and Efficiency

Figure 2.8 showed the layout for the DTℂWT for 1D signals. We saw from Equation 2.2.22 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 2.12 shows how to efficiently implement this with FBs.

As we did for section 2.2.5, we calculate and compare the complexity of the DTℂWT . 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 $= N \times N$):
    - Column filtering requires $5N^2 + 7N^2$ multiply-adds
    - Row filtering to make the LoLo term requires $5N^2$ more multiply-adds

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

Figure 2.12: The filter bank implementation of the DTℂWT . Image taken from [16]

  - 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 2.12).
  - Row filtering to make the 45° and 135° requires $7N^2 + 4N^2$ multiply-adds
  - Row filtering to make the $75° and 105°$ 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 2.12).
  - 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:

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

$$T(N) = 48N^2 + \sum_{j=1}^{J} 68 \times 2^{-2j}N^2 \approx 100N^2 \tag{2.2.23}$$

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 section 2.2.5, 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 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.

Figure 2.13: Four scales of the DTℂWT (left) and its associated frequency coverage, or $A(\omega)$ (right). Note the reduced scale compared to Figure 2.7.

Figure 2.14: 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 [35].

For a simple comparison experiment, we performed the two transforms on a four core Intel i7 processor (a moderately high-end personal computer). The DTℂWT 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 DTℂWT implementation took 0.35s vs. 3.5s (times were averaged over several runs).

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 DTℂWT ?

It is important to note that in the case of the DTℂWT the wavelet transform is also approximately unitary, i.e.,

$$\|x\|^2 \approx \|\mathcal{W}x\|^2 \tag{2.2.24}$$

and the implementation is perfectly invertible as the Littlewood-Paley function is unity (or very near unity) $\forall \omega$. See Figure 2.13. 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 [35]. 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 2.14). Having a unitary transform limits the amount the features can change, which will make the entire network more stable to distortion and noise.

(a) DTℂWT wavelets (left to right) — 15°, 45° and 75°

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

Figure 2.15: Normalized Energy spectra of the DTℂWT 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.

2.2.7 Summary of Methods

One final comparison to make between the DTℂWT and the Morlet wavelets is their frequency coverage. The Morlet wavelets can be made to be tighter than the DTℂWT, which gives better angular resolution — see Figure 2.15. 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 2.1 compares the advantages and disadvantages of the wavelet methods discussed in this chapter.

2.3 Neural Networks

2.3.1 The Neuron and Single Layer Neural Networks

The neuron, shown in Figure **??** is the core building block of Neural Networks. It takes the dot product between an input vector $\mathbf{x} \in \mathbb{R}^N$ and a weight vector $\mathbf{b}$, before applying a chosen nonlinearity, f. I.e.

$$y = f(\langle \mathbf{x}, \mathbf{w} \rangle) = f\left(\sum_{i=0}^{N} x_i w_i \right)$$

where we have used the shorthand $b = w_0$ and $x_0 = 1$. Note that if $\langle \mathbf{w}, \mathbf{w} \rangle = 1$ then $\langle \mathbf{x}, \mathbf{w} \rangle$ is the distance from the point $\mathbf{x}$ to the hyperplane with normal $\mathbf{w}$. With general $\mathbf{w}$ this can be thought of as a scaled distance.

Table 2.1: 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

Typical nonlinear functions f are the sigmoid function: (**include figure**)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

or ReLU:

$$\text{ReLU}(x) = \max(x, 0)$$

The weight vector $\mathbf{w}$ defines a hyperplane in $\mathbb{R}^N$ which splits the space into two. The choice of nonlinearity then affects how points on each side of the plane are treated. For a sigmoid, points far below the plane get mapped to 0 and points far above the plane get mapped to 1 (with points near the plane having a value of 0.5). For tanh nonlinearities, these points get mapped to -1 and 1. For ReLU nonlinearities, every point below the plane ($\langle \mathbf{x}, \mathbf{w} \rangle < 0$) gets mapped to zero and every point above the plane keeps its inner product value.

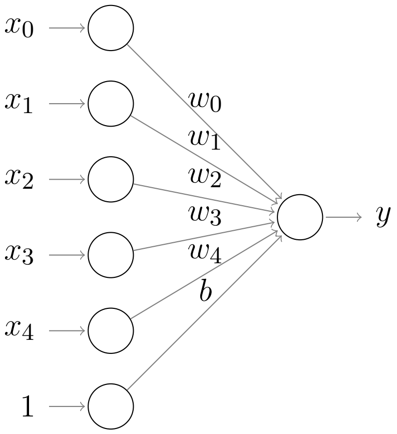


Figure 2.16: **A single neuron.** The neuron is composed of inputs x_i, weights w_i (and a bias term), as well as an activation function. Typical activation functions include the sigmoid function, tanh function and the ReLU

Consider a single neuron acting as logistic regressor. I.e. we have $\mathcal{D} = \{\mathbf{x}_i, y_i\}$ where $y_i \in \{0, 1\}$. We can easily find the maximum likelihood estimates for $\mathbf{w}$ by minimizing the loss

2.3.2 Loss or Error Term

2.3.3 Backpropagation

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:

$$\frac{\partial l}{\partial w_L} = \frac{\partial l}{\partial z_L}\frac{\partial z_L}{\partial w_L} + \underbrace{\lambda w}_{\text{from the reg. loss}} \tag{2.3.1}$$

$\frac{\partial l}{\partial z_L}$ 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[6] 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:

$$\frac{\partial l}{\partial w_{L-1}} = \frac{\partial l}{\partial z_L}\frac{\partial z_L}{\partial z_{L-1}}\frac{\partial z_{L-1}}{\partial w_{L-1}} \tag{2.3.2}$$

We can generalize this easily like so:

$$\frac{\partial l}{\partial w_l} = \frac{\partial l}{\partial z_L} \underbrace{\prod_{i=L}^{l+1} \frac{\partial z_i}{\partial z_{i-1}}}_{\text{product to } l\text{'s output}} \frac{\partial z_l}{\partial w_l} \tag{2.3.3}$$

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:

$$R_E(f(x,w)) = \int l(y, f(x,w))dP(x,y) \tag{2.3.4}$$

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

$$R_{EMP}(f(x,w)) = \frac{1}{N}\sum_{i=1}^{N} l(y^{(i)}, f(x^{(i)}, w)) \tag{2.3.5}$$

2.3.4 Gradient descent vs Stochastic Gradient Descent vs Mini-Batches

We can minimize Equation 2.3.5 with *gradient descent* [36]. Updates can be made on a generic network parameter w with:

$$w_{t+1} = w_t - \eta \frac{\partial E_n}{\partial w} \tag{2.3.6}$$

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.,

$$w_{t+1} = w_t - \eta \frac{\partial l}{\partial w} \tag{2.3.7}$$

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

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:

$$w_{t+1} = w_t - \eta \sum_{n=1}^{N_B} \frac{\partial l}{\partial w} \tag{2.3.8}$$

Loss Functions

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 [37, 38].

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):

$$p_j = \frac{e^{(f(x,w))(j)}}{\sum\limits_{k=1}^{C} e^{(f(x,w))(k)}} \tag{2.3.9}$$

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:

$$l(y_i,x,w) = \sum_{j=1}^{C} \mathbb{1}\{y_i = j\} \log p_j \tag{2.3.10}$$

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:

$$l(y,x,w) = \sum_{l=1}^{C} \left[\max(0, 1-\delta(y_i,l)w^T x_i)\right]^p \tag{2.3.11}$$

Using a hinge loss like this introduces extra parameters, which would typically replace the final fully connected layer. The p parameter in Equation 2.3.11 can be used to choose ℓ^1 Hinge-Loss, or ℓ^2 Squared Hinge-Loss.

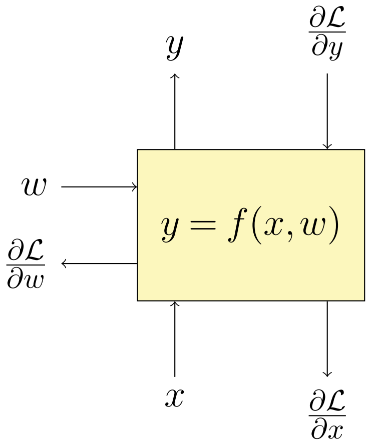
Regularization

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:

$$\mathcal{L} = \mathcal{L}_{data} + \underbrace{\frac{1}{2}\lambda_{fc}\sum_{i=1}^{L_{fc}}\sum_{j}\sum_{k}(w_i[j,k])^2}_{\text{fully connected loss}} + \underbrace{\frac{1}{2}\lambda_{c}\sum_{i=1}^{L_{c}}\sum_{u_1}\sum_{u_2}\sum_{d}\sum_{n}(f_i[u_1,u_2,d,n])^2}_{\text{convolutional loss}} \tag{2.3.12}$$

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.


y
$\frac{\partial \mathcal{L}}{\partial y}$
w
$y = f(x,w)$
$\frac{\partial \mathcal{L}}{\partial w}$
x
$\frac{\partial \mathcal{L}}{\partial x}$

2.3.5 Learning

2.3.6 Optimization

2.3.7 Extending to multiple layers and different block types

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:

$$z[u_1, u_2, d''] = b[d''] + \sum_{i=-\frac{H'}{2}}^{\frac{H'}{2}-1} \sum_{j=-\frac{W'}{2}}^{\frac{W'}{2}-1} \sum_{k=0}^{D-1} f[i,j,k,d'']x[u_1 - i, u_2 - j, k] \tag{2.3.13}$$

ReLUs

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 [39, 40].

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

Figure 2.17: 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.

Figure 2.18: **??** 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.

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 [41]. 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. **(author?)** [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.

Pooling

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 2.18. A review of pooling methods in [43] found them both to perform equally well.

Batch Normalization

Batch normalization proposed only very recently in [44] 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). **??** becomes:

$$\tilde{z}(u_1,u_2,d) = \gamma \frac{z - E[z]}{\sqrt{Var[z]}} + \beta \tag{2.3.14}$$

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

$$\begin{aligned}
\tilde{z}^{(i)} &= \gamma \hat{z}^{(i)} + \beta \\
\frac{\partial \mathcal{L}}{\partial \gamma} &= \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \tilde{z}^{(i)}} \cdot \hat{z}^{(i)} && (2.3.15) \\
\frac{\partial \mathcal{L}}{\partial \beta} &= \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \tilde{z}^{(i)}} && (2.3.16)
\end{aligned}$$

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:

$$\begin{aligned}
z &= g(BN(Wu + b)) \\
&= g(BN(Wu)) && (2.3.17)
\end{aligned}$$

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,

$$BN(Wu) = BN((aW)u) \tag{2.3.18}$$

Figure 2.19: A residual unit. The identity mapping is always present, and the network learns the difference from the identity mapping, $\mathcal{F}(x)$. Taken from [45].

It is also particularly useful for backpropagation, as an increase in weights leads to *smaller* gradients [44], making the network far more resilient to the problems of vanishing and exploding gradients:

$$\begin{aligned} \frac{\partial BN((aW)u)}{\partial u} &= \frac{\partial BN(Wu)}{\partial u} \\ \frac{\partial BN((aW)u)}{\partial (aW)} &= \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W} \end{aligned} \tag{2.3.19}$$

2.4 Relevant Architectures

2.4.1 LeNet

2.4.2 AlexNet

2.4.3 VGG

2.4.4 Residual Networks

The current state of the art design introduced a clever novel feature called a residual unit[45, 46]. 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 2.19. If a desired mapping is denoted $\mathcal{H}(x)$, instead of trying to learn this, they instead learn $\mathcal{F}(x) = \mathcal{H}(x) - x$.

2.4.5 old

Convolutional Neural Networks (CNNs) were initially introduced by **(author?)** [47] in [47]. 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 [48], the use of an improved non-linearity — the Rectified Linear Unit, or ReLU, new regularization methods[49], and access to more powerful computers in graphics cards, or GPUs, allowed Krizhevsky, Sutskever and Hinton

Figure 2.20: Standard CNN architecture. Taken from [53]

to develop AlexNet[50]. 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 been successfully applied to object detection [51] and human pose estimation [52]. 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 [53], the state of the art design for postal digit recognition on the MNIST dataset.
- Google's GoogLeNet [54] 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 [55] — 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 [45] 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 2.20):

2.4.6 Fully Connected Layers

The convolution, 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:

$$z_i = \sum_j W_{ij} x_j \tag{2.4.1}$$

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.

Chapter 3

ScatterNet Visualization

3.1 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 **(author?)** in their work [4], and then were rigorously defined by Mallat in [56]. 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 section 2.3, 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 **??**.

3.2 Translation Invariant Scatternets

The translation invariant Scatternets were mostly covered in [5]. This section summarises the method of this paper.

3.2.1 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 3.1: 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

$$\Phi x_c = \Phi x \tag{3.2.1}$$

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.

$$\|\Phi x' - \Phi x\| \leq C\|x' - x\| \tag{3.2.2}$$

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.

$$\|\Phi x_\tau - \Phi x\| \leq C\|x\| \sup_{\mathbf{u}} |\nabla\tau(\mathbf{u})| \tag{3.2.3}$$

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

3.2.2 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 [56], however, the wavelet transform is not invariant to shifts.

We saw in **??** 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 3.2). Interestingly, the modulus operator, in this case, does not lose any information [57] (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 **(author?)** 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 ($\|x * \psi_\lambda\|$). **(author?)** 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 3.3.

3.3 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 [9], but it was limited to only averaging over scale once, and they were no longer using it in [8], 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 [58, 9–11], and the second is image classification with Oyallon in [7, 8]. In this section, we outline the properties and structure of this extended Scatternet.

3.3.1 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.,

$$\Phi_2(\Phi_1(g_1 g_2 x)) = \Phi_2(g_2 \Phi_1(x)) = \Phi_2(\Phi_1(x)) \tag{3.3.1}$$

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 `images/scatternet_diagram.png`

Figure 3.3: 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 [5].

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 [59–61].

3.3.2 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:

$$g\mathbf{u} = v + R_\theta \mathbf{u} \tag{3.3.2}$$

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

$$gh = (v + R_\theta v', \theta + \theta') \tag{3.3.3}$$

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.,

$$U[p](gx) = gU[p]x \tag{3.3.4}$$

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}

$$S[p]x(g) = \sum_{h \in G_{rt}} hU[p]x\Phi_J(h^{-1}g) \tag{3.3.5}$$

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

3.3.3 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.,

$$\tilde{W}_1 x = \Big(x * \phi_J, \{|x * \psi_{\theta,j}|\}\Big) = (S_0 x, U_1 x) \tag{3.3.6}$$

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:

$$\Phi_J(g) = \bar{\phi}(\theta) * \phi_J(u) \tag{3.3.7}$$

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:

$$U_1 x = \Big\{|x * \psi_{j,\theta}| \Big| j = \{0,1\}, \theta = k\pi/4, k = \{0,1,2,3\}, \Big\} \tag{3.3.8}$$

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

images/scatternet_roto_scale_3dwavelet.png

Figure 3.4: 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 [9].

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

$$\Psi_{0,J,k_2}(u,\theta) = \phi_J(u) * \bar{\psi}_{k_2}(\theta) \tag{3.3.9}$$

$$\Psi_{\theta_2,j_2,}(u,\theta) = \psi_{\theta_2,j_2}(u) * \bar{\phi}(\theta) \tag{3.3.10}$$

$$\Psi_{\theta_2,j_2,k_2}(u,\theta) = \psi_{\theta_2,j_2}(u) * \bar{\psi}_{k_2}(\theta) \tag{3.3.11}$$

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 3.4 for what this looks like.

The wavelet-modulus operator then is:

$$\tilde{W}_m Y = \left(Y * \Phi_J(g), |Y * \Psi_{\theta_m,j_m,k_m}(g)|\right) \tag{3.3.12}$$

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

$$Sx = (x * \phi_J(\mathbf{u}), U_1 x * \Phi_J(p_1), U_2 x * \Phi_J(p_2)) \tag{3.3.13}$$

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

Figure 3.5: 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 [63].

Figure 3.6: 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 [63].

3.4 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.

(author?) first attempted to use 'deconvolution' to improve their learning [62], then later for purely visualization purposes [63]. Their method involves mapping activations at different layers of the network back to the pixel space

Figure 3.5 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. 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. **(author?)** 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 3.6.

Figure 3.7 gives a more detailed view of how the deconvolution works for convolutional layers.

(author?) take a slightly different route on deconvolution networks [64]. 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 3.7: 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 [62].

Figure 3.8: 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 [63].

3.5 Introduction

Scattering transforms, or ScatterNets, have recently gained much attention and use due to their ability to extract generic and descriptive features in well defined way. They can be used as unsupervised feature extractors for image classification [5, 8, 65, 66] and texture classification [9], or in combination with supervised methods such as Convolutional Neural Networks (CNNs) to make the latter learn quicker, and in a more stable way [67].

ScatterNets have been shown to perform very well as image classifiers. In particular, they can outperform CNNs for classification tasks with reduced training set sizes, e.g. in CIFAR-10 and CIFAR-100 (Table 6 from [67] and Table 4 from [65]). They are also near state-of-the-art for Texture Discrimination tasks (Tables 1–3 from [9]). Despite this, there still exists a considerable gap between them and CNNs on challenges like CIFAR-10 with the full training set (83% vs. 93%). Even considering the benefits of ScatterNets, this gap must be addressed.

We first revise the operations that form a ScatterNet in section 3.6. We then introduce our DeScatterNet (section 3.7), and show how we can use it to examine the layers of ScatterNets (using a similar technique to the CNN visualization in [63]). We use this analysis tool to highlight what patterns a ScatterNet is sensitive to (section 3.8), showing that they are very different from what their CNN counterparts are sensitive to, and possibly less useful for discriminative tasks.

We use these observations to propose an architectural change to ScatterNets, which have not changed much since their inception in [56]. Two changes of note however are the work of Sifre and Mallat in [9], and the work of Singh and Kingsbury in [65]. Sifre and Mallat introduced Rotationally Invariant ScatterNets which took ScatterNets in a new direction, as the architecture now included filtering across the wavelet orientations (albeit with heavy restrictions on the fitlers used). Singh and Kingsbury achieved improvements in performance in a Scattering system using the spatially implementable DTℂWT [19] wavelets instead of the Fourier Transform (FFT) based Morlet previously used.

We build on these two systems, showing that with carefully designed complex filters applied across the complex spatial coefficients of a 2-D DTℂWT, we can build filters that are sensitive to more recognizable shapes like those commonly seen in CNNs, such as corners and curves (section 3.9).

3.6 The Scattering Transform

The Scattering Transform, or ScatterNet, is a cascade of complex wavelet transforms and modulus non-linearities (throwing away the phase of the complex wavelet coefficients). At a chosen scale, averaging filters provide invariance to nuisance variations such as shift and deformation (and potentially rotations). Due to the non-expansive nature of the wavelet transform and the modulus operation, this transform is stable to deformations.

Typical implementations of the ScatterNet are limited to two 'orders' (equivalent to layers in a CNN) [8, 65, 67]. In addition to scattering *order*, we also have the *scale* of invariance, J. This is the number of band-pass coefficients output from a wavelet filter bank (FB), and defines the cut-off frequency for the final low-pass output: $2^{-J}\frac{f_s}{2}$ (f_s is the sampling frequency of the signal). Finally, we call the number of oriented wavelet coefficients used L. These are the three main hyper-parameters of the scattering transform and must be set ahead of time. We describe a system with scale parameter $J = 4$, order $m = 2$ and with $L = 6$ orientations (L is fixed to 6 for the DT$\mathbb{C}$WT but is flexible for the FFT based Morlet wavelets).

Consider an input signal $x(\boldsymbol{u}), \boldsymbol{u} \in \mathbb{R}^2$. The zeroth order **scatter** coefficient is the lowpass output of a J level FB:

$$S_0 x(\boldsymbol{u}) \triangleq (x * \phi_J)(\boldsymbol{u}) \tag{3.6.1}$$

This is invariant to translations of up to 2^J pixels[3]. In exchange for gaining invariance, the S_0 coefficients have lost a lot of information (contained in the rest of the frequency space). The remaining energy of x is contained within the first order **wavelet** coefficients:

$$W_1 x(\boldsymbol{u}, j_1, \theta_1) \triangleq x * \psi_{j_1,\theta_1} \tag{3.6.2}$$

for $j_1 \in \{1, 2, \ldots, J\}, \theta_1 \in \{1, 2, \ldots, L\}$. We will want to retain this information in these coefficients to build a useful classifier.

Let us call the set of available scales and orientations Λ_1 and use λ_1 to index it. For both Morlet and DT$\mathbb{C}$WT implementations, ψ is complex-valued, i.e., $\psi = \psi^r + j\psi^i$ with ψ_r and ψ_i forming a Hilbert Pair, resulting in an analytic ψ. This analyticity provides a source of invariance — small input shifts in x result in a phase rotation (but little magnitude change) of the complex wavelet coefficients[4].

[3] From here on, we drop the $\boldsymbol{u}$ notation when indexing x, for clarity.

[4] In comparison to a system with purely real filters such as a CNN, which would have rapidly varying coefficients for small input shifts [19].

Taking the magnitude of W_1 gives us the first order **propagated** signals:

$$U_1x(\boldsymbol{u},\lambda_1) \triangleq |x*\psi_\lambda| = \sqrt{(x*\psi^r_{\lambda_1})^2+(x*\psi^i_{\lambda_1})^2} \tag{3.6.3}$$

The first order scattering coefficient makes U_1 invariant up to our scale J by averaging it:

$$S_1x(\boldsymbol{u},\lambda_1) \triangleq |x*\psi_\lambda|*\phi_J \tag{3.6.4}$$

If we define $U_0 \triangleq x$, then we can iteratively define:

$$\begin{aligned} W_m &= U_{m-1}*\psi_{\lambda_m} & (3.6.5)\\ U_m &= |W_m| & (3.6.6)\\ S_m &= U_m*\phi_J & (3.6.7) \end{aligned}$$

We repeat this for higher orders, although previous work shows that, for natural images, we get diminishing returns after $m=2$. The output of our ScatterNet is then:

$$Sx = \{S_0x, S_1x, S_2x\} \tag{3.6.8}$$

3.6.1 Scattering Color Images

A wavelet transform like the DT$\mathbb{C}$WT accepts single channel input, while we often work on RGB images. This leaves us with a choice. We can either:

1. Apply the wavelet transform (and the subsequent scattering operations) on each channel independently. This would triple the output size to $3C$.

2. Define a frequency threshold below which we keep color information, and above which, we combine the three channels into a single luminance channel.

The second option uses the well known fact that the human eye is far less sensitive to higher spatial frequencies in color channels than in luminance channels. This also fits in with the first layer filters seen in the well known Convolutional Neural Network, AlexNet. Roughly one half of the filters were low frequency color ‘blobs’, while the other half were higher frequency, grayscale, oriented wavelets.

For this reason, we choose the second option for the architecture described in this paper. We keep the 3 color channels in our S_0 coefficients, but work only on grayscale for high orders (the S_0 coefficients are the lowpass bands of a J-scale wavelet transform, so we have effectively chosen a color cut-off frequency of $2^{-J}\frac{f_s}{2}$).

For example, consider an RGB input image x of size $64 \times 64 \times 3$. The scattering transform we have described with parameters $J = 4$ and $m = 2$ would then have the following coefficients:

$$\begin{aligned}
S_0 &: (64 \times 2^{-J}) \times (64 \times 2^{-J}) \times 3 = 4 \times 4 \times 3 \\
S_1 &: 4 \times 4 \times (LJ) = 4 \times 4 \times 24 \\
S_2 &: 4 \times 4 \times \left(\frac{1}{2}L^2 J(J-1)\right) = 4 \times 4 \times 216 \\
S &: 4 \times 4 \times (216 + 24 + 3) = 4 \times 4 \times 243
\end{aligned}$$

3.7 The Inverse Network

We now introduce our inverse scattering network. This allows us to back project scattering coefficients to the image plane; it is inspired by the **DeconvNet** used by Zeiler and Fergus in [63] to look into the deeper layers of CNNs.

We emphasize that instead of thinking about perfectly reconstructing x from $S \in \mathbb{R}^{H' \times W' \times C}$, we want to see what signal/pattern in the input image caused a large activation in each channel. This gives us a good idea of what each output channel is sensitive to, or what it extracts from the input. Note that we do not use any of the log normalization layers described in [8, 65].

3.7.1 Inverting the Low-Pass Filtering

Going from the U coefficients to the S coefficients involved convolving by a low pass filter, ϕ_J followed by decimation to make the output $(H \times 2^{-J}) \times (W \times 2^{-J})$. ϕ_J is a purely real filter, and we can 'invert' this operation by interpolating S to the same spatial size as U and convolving with the mirror image of ϕ_J, $\widetilde{\phi}_J$ (this is equivalent to the transpose convolution described in [63]).

$$\hat{S}_m = S_m * \widetilde{\phi}_J \tag{3.7.1}$$

This will not recover U as it was on the forward pass, but will recover all the information in U that caused a strong response in S.

3.7.2 Inverting the Magnitude Operation

In the same vein as [63], we face a difficult task in inverting the non-linearity in our system. We lend inspiration from the *switches* introduced in the DeconvNet; the switches in a DeconvNet save the location of maximal activations so that on the backwards pass activation layers could be unpooled trivially. We do an equivalent operation by saving the phase of the complex activations. On the backwards pass we reinsert the phase to give our recovered W.

$$\hat{W}_m = \hat{U}_m e^{j\theta_m} \tag{3.7.2}$$

3.7.3 Inverting the Wavelet Decomposition

Using the DT$\mathbb{C}$WT makes inverting the wavelet transform simple, as we can simply feed the coefficients through the synthesis filter banks to regenerate the signal. For complex ψ, this is convolving with the conjugate transpose $\widetilde{\psi}$:

$$\begin{aligned} \hat{U}_{m-1} &= \hat{S}_{m-1} + \hat{W}_m \\ &= S_{m-1} * \widetilde{\phi}_J + \sum_{j,\theta} W_m(\boldsymbol{u}, j, \theta) * \widetilde{\psi}_{j,\theta} \end{aligned} \tag{3.7.3}$$

3.8 Visualization with Inverse Scattering

To examine our ScatterNet, we scatter all of the images from ImageNet's validation set and record the top 9 images which most highly activate each of the C channels in the ScatterNet. This is the *identification* phase (in which no inverse scattering is performed).

Then, in the *reconstruction* phase, we load in the $9 \times C$ images, and scatter them one by one. We take the resulting $4 \times 4 \times 243$ output vector and mask all but a single value in the channel we are currently examining.

This 1-sparse tensor is then presented to the inverse scattering network from Figure 3.9 and projected back to the image space. Some results of this are shown in Figure 3.10. This figure shows reconstructed features from the layers of a ScatterNet. For a given output channel, we show the top 9 activations projected independently to pixel space. For the first and second order coefficients, we also show the patch of pixels in the input image which cause this large output. We display activations from various scales (increasing from first row to last row), and random orientations in these scales.

The order 1 scattering (labelled with ‘Order 1’ in Figure 3.10) coefficients look quite similar to the first layer filters from the well known AlexNet CNN [50]. This is not too surprising, as the first order scattering coefficients are simply a wavelet transform followed by average pooling. They are responding to images with strong edges aligned with the wavelet orientation.

The second order coefficients (labelled with ‘Order 2’ in Figure 3.10) appear very similar to the order 1 coefficients at first glance. They too are sensitive to edge-like features, and some of them (e.g. third row, third column and fourth row, second column) are mostly just that. These are features that have the same oriented wavelet applied at both the first and second order. Others, such as the 9 in the first row, first column, and first row, fourth column are more sensitive to checker-board like patterns. Indeed, these are activations where the orientation of the wavelet for the first and second order scattering were far from each other (15° and 105° for the first row, first column and 105° and 45° for the first row, fourth column).

For comparison, we include reconstructions from the second layer of the well-known VGG CNN (labelled with ‘VGG conv2_2’, in Figure 3.10). These were made with a DeconvNet, following the same method as [63]. Note that while some of the features are edge-like, we also see higher order shapes like corners, crosses and curves.

3.9 Corners, Crosses and Curves

These reconstructions show that the features extracted from ScatterNets vary significantly from those learned in CNNs after the first order. In many respects, the features extracted from a CNN like VGGNet look preferable for use as part of a classification system.

[9] and [8] introduced the idea of a ‘Roto-Translation’ ScatterNet. Invariance to rotation could be made by applying averaging (and bandpass) filters across the L orientations from the wavelet transform *before* applying the complex modulus. Momentarily ignoring the form of the filters they apply, referring to them as $F_k \in \mathbb{C}^L$, we can think of this stage as stacking the L outputs of a complex wavelet transform on top of each other, and convolving these filters F_k over all spatial locations of the

wavelet coefficients $W_m x$ (this is equivalent to how filters in a CNN are fully connected in depth):

$$V_m x(\boldsymbol{u}, j, k) = W_m x * F_k = \sum_\theta W_m x(\boldsymbol{u}, j, \theta) F_k(\theta) \tag{3.9.1}$$

We then take the modulus of these complex outputs to make a second propagated signal:

$$U'_m x \triangleq |V_m x| = |W_m x * F_k| = |U_{m-1} x * \psi_{\lambda_m} * F_k| \tag{3.9.2}$$

We present a variation on this idea, by filtering with a more general $F \in \mathbb{C}^{H\times W\times 12}$. We use F of length 12 rather than 6, as we use the $L = 6$ orientations and their complex conjugates; each wavelet is a 30° rotation of the previous, so with 12 rotations, we can cover the full 360°.

Figure 3.11 shows some reconstructions from these V coefficients. Each of the four quadrants show reconstructions from a different class of ScatterNet layer. All shapes are shown in real and imaginary Hilbert-like pairs; the top images in each quadrant are reconstructed from a purely real V, while the bottom inputs are reconstructed from a purely imaginary V. This shows one level of invariance of these filters, as after taking the complex magnitude, both the top and the bottom shape will activate the filter with the same strength. In comparison, for the purely real filters of a CNN, the top shape would cause a large output, and the bottom shape would cause near 0 activity (they are nearly orthogonal to each other).

In the top left, we display the 6 wavelet filters for reference (these were reconstructed from U_1, not V_1). In the top right of the figure we see some of the shapes made by using the F's from the Roto-Translation ScatterNet [9, 8]. The bottom left is where we present some of our novel kernels. These are simple corner-like shapes made by filtering with $F \in \mathbb{C}^{1\times 1\times 12}$

$$F = [1, j, j, 1, 0, 0, 0, 0, 0, 0, 0, 0] \tag{3.9.3}$$

The six orientations are made by rolling the coefficients in F along one sample (i.e. $[0, 1, j, j, 1, 0, \ldots]$, $[0, 0, 1, j, j, 1, 0, \ldots]$, $[0, 0, 0, 1, j, j, 1, 0, \ldots]$ $\ldots$). Coefficients roll back around (like circular convolution) when they reach the end.

Finally, in the bottom right we see shapes made by $F \in \mathbb{C}^{3\times 3\times 12}$. Note that with the exception of the ring-like shape which has 12 non-zero coefficients, all of these shapes were reconstructed with F's that have 4 to 8 non-zero coefficients of a possible

64. These shapes are now beginning to more closely resemble the more complex shapes seen in the middle stages of CNNs.

3.10 Discussion

This paper presents a way to investigate what the higher orders of a ScatterNet are responding to - the DeScatterNet described in section 3.7. Using this, we have shown that the second 'layer' of a ScatterNet responds strongly to patterns that are very dissimilar to those that highly activate the second layer of a CNN. As well as being dissimilar to CNNs, visual inspection of the ScatterNet's patterns reveal that they may be less useful for discriminative tasks, and we believe this may be causing the current gaps in state-of-the-art performance between the two.

We have presented an architectural change to ScatterNets that can make it sensitive to more recognizable shapes. We believe that using this new layer is how we can start to close the gap, making more generic and descriptive ScatterNets while keeping control of their desirable properties.

A future paper will include classifier results for these new filters.

figure1.png

Figure 3.9: A DeScattering layer (left) attached to a Scattering layer (right). We are using the same convention as [63] Figure 1 - the input signal starts in the bottom right hand corner, passes forwards through the ScatterNet (up the right half), and then is reconstructed in the DeScatterNet (downwards on the left half). The DeScattering layer will reconstruct an approximate version of the previous order's propagated signal. The 2×2 grids shown around the image are either Argand diagrams representing the magnitude and phase of small regions of *complex* (De)ScatterNet coefficients, or bar charts showing the magnitude of the *real* (De)ScatterNet coefficients (after applying the modulus non-linearity). For reconstruction, we need to save the discarded phase information and reintroduce it by multiplying it with the reconstructed magnitudes.

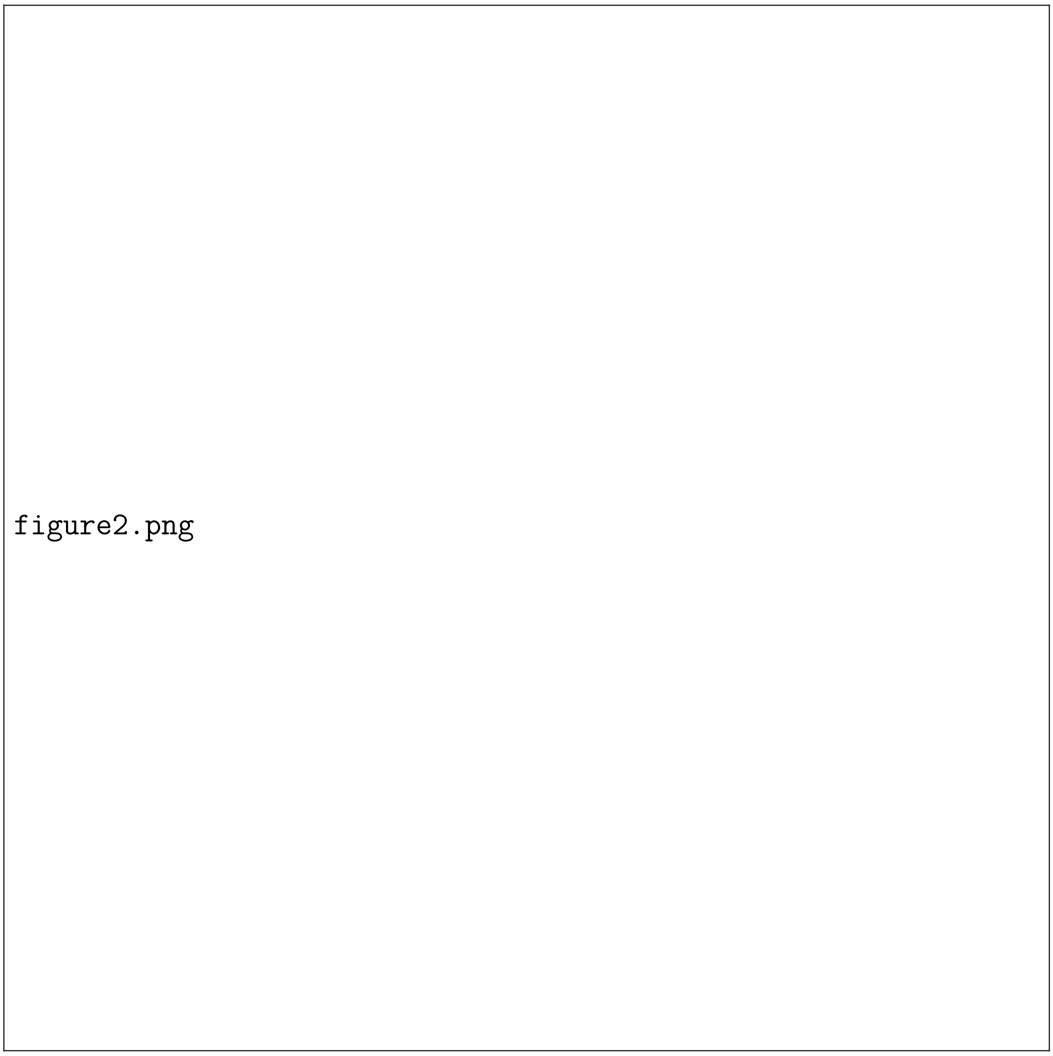


Figure 3.10: Visualization of a random subset of features from S_0 (all 3), S_1 (6 from the 24) and S_2 (16 from the 240) scattering outputs. We record the top 9 activations for the chosen features and project them back to the pixel space. We show them alongside the input image patches which caused the large activations. We also include reconstructions from layer conv2_2 of VGG Net [55](a popular CNN, often used for feature extraction) for reference — here we display 16 of the 128 channels. The VGG reconstructions were made with a CNN DeconvNet based on [63]. Image best viewed digitally.

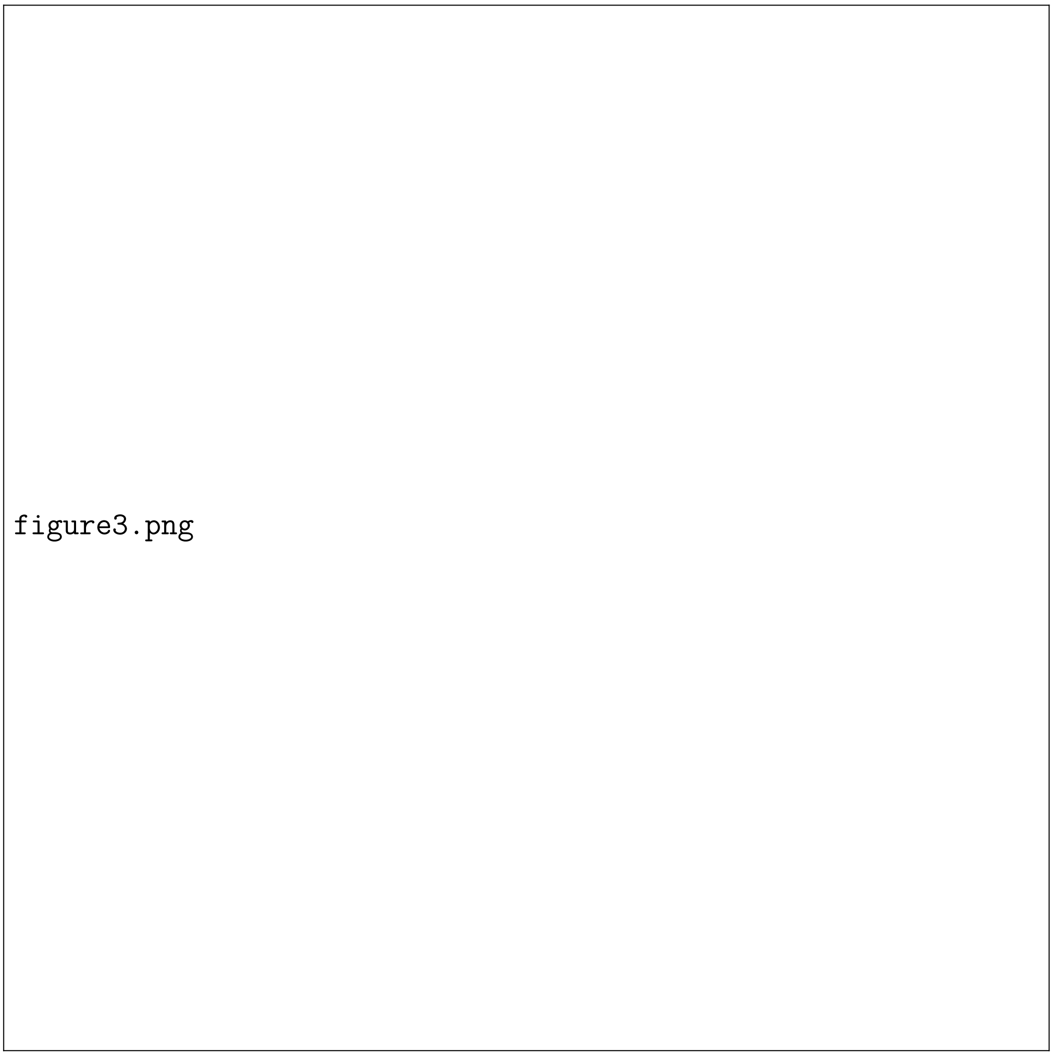


Figure 3.11: Shapes possible by filtering across the wavelet orientations with complex coefficients. All shapes are shown in pairs: the top image is reconstructed from a purely real output, and the bottom image from a purely imaginary output. These 'real' and 'imaginary' shapes are nearly orthogonal in the pixel space (normalized dot product < 0.01 for all but the doughnut shape in the bottom right, which has 0.15) but produce the same U', something that would not be possible without the complex filters of a ScatterNet. Top left - reconstructions from U_1 (i.e. no cross-orientation filtering). Top right- reconstructions from U_1' using a $1 \times 1 \times 12$ Morlet Wavelet, similar to what was done in the 'Roto-Translation' ScatterNet described in [9, 8]. Bottom left - reconstructions from U_1' made with a more general $1 \times 1 \times 12$ filter, described in Equation 3.9.3. Bottom right - some reconstructions possible by filtering a general $3 \times 3 \times 12$ filter.

Chapter 4

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.

4.1 Fast GPU Implementation

4.2 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.

4.3 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.

4.4 Activation Statistics

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

4.5 Wavelet CNNs

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

4.6 Properties of a Scatternet

Adversarial attacks are of the form of additive noise. One of the nice properties of Scattering transforms is that

$$||\Phi(x) - \Phi(x+\epsilon)|| \leq \epsilon$$

Other transformations such as camera warping come under the scope of diffeomorphisms:

$$L_\tau x(u) = x(u - \tau(u))$$

We can define the largest displacement of this field as

$$||\tau||_\infty = \sup_{u \in \mathbb{R}^2} |\tau(u)|$$

Deformations not only change u, but they change x as well. A Taylor series expansion around u shows this:

$$u + v - \tau(u+v) \approx u + v - \nabla\tau(u)v = u - \tau(u) + (1 - \nabla\tau(u))v$$

This can be summarised by noting that in the neighbourhood of u, τ introduces a translation by $\tau(u)$ and a warping that differes from 1 by $\nabla\tau(u)$. This warping can be quantified as

$$||\nabla\tau||_\infty = \sup_{u\in\mathbb{R}^2} ||\nabla\tau(u)||$$

A representation is stable to deformations if we can define to small constants, C_1, C_2 such that for all $x \in L^2(\mathbb{R}^2$:

$$||\Phi L_\tau x - \Phi x|| \leq (C_1||\tau||_\infty + C_2||\nabla\tau||_\infty)||x||$$

Note that if $C_1 = 0$ then we have full translation invariance (translation is when $\tau(u) = C$, $\nabla\tau(u) = 0$). Full translation invariance does not imply stability to transformations. E.g. the Fourier modulus has full translation invariance. If we introduce a warping:

$$\tau(u) = \epsilon u, \epsilon > 0$$

Then a sine wave with frequency ω will get shifted to $\frac{w}{1-\epsilon}$ and $||\Phi L_\tau x - \Phi x|| = 2$ even when $||\nabla\tau||_\infty = \epsilon$ is made arbitrarily small.

Is it possible to maintain these properties if I modify the Scattering Transform? So the translation invariance property isn't really invariance. Well it is is invariant to sub-pixel and to an extent pixel shifts.

Chapter 5

Learning in the Wavelet Domain

I wonder how my layers that learn in the frequency domain would compare to another state of the art network on texture classification.

5.1 Gradients in critically sampled wavelet systems

If wavelet transforms are to have any place in a deep learning architecture, it is important that we are able to calculate the derivatives of wavelet activations with respect to inputs, and use that to define efficient methods to propagate gradients back from a loss function to any point.

Let us start with the 1-D discrete wavelet transform for this. The 1-D DWT is composed of the following components:

1. Decimation
2. Interpolation
3. Convolution
4. Padding (often used to handle the borders of images before convolution)

Once we define the derivative of the output w.r.t. the input for each of these blocks, we can use the chain rule to arbitrarily find the derivatives through any path in the system.

The gradient of decimation is interpolation and the gradient of interpolation is decimation

Proof. Easy to see, not sure how to prove. □

The gradient of convolution is correlation.

Proof. This is a well-known property but we can prove it here for the discrete 1-D case. Let

$$y[n] = (x * h)[n] = \sum_{m=-\inf}^{\inf} x[m]h[n-m] = \sum_{m=-\inf}^{\inf} x[n-m]h[m]$$

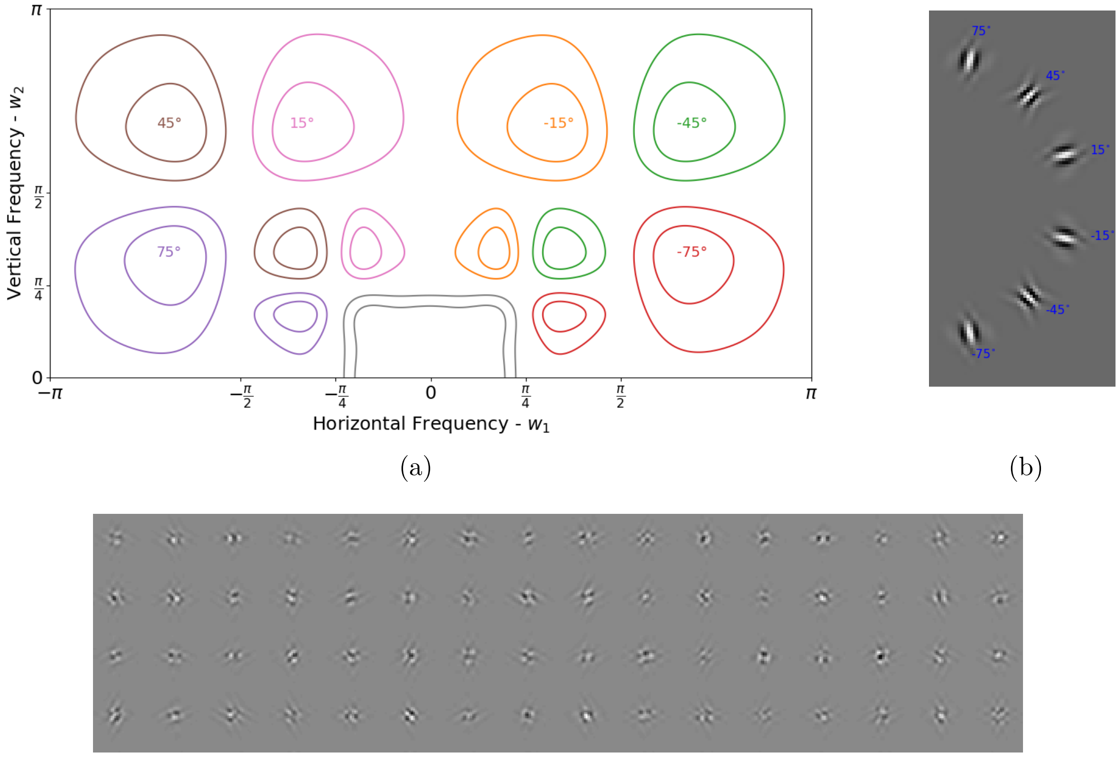
Then □

5.2 Introduction

Using wavelet based methods with deep learning is nascent but not novel. Wavelets have been applied to texture classification [68, 58], super-resolution [**?**] and for adding detail back into dense pixel-wise segmentation tasks [**?**]. One exciting piece of work built on wavelets is the Scattering Transform [56], which has been used as a feature extractor for learning, firstly with simple classifiers [5**?**], and later as a front end to hybrid deep learning tasks[67**?**]. Despite their power and simplicity, scattering features are fixed and are visibly different to regular CNN features [69] - their nice invariance properties come at the cost of flexibility, as there is no ability to learn in between scattering layers.

For this reason, we have been investigating a slightly different approach, more similar to the Fourier based work in [70] in which Rippel et. al. investigate parameterization of filters in the Fourier domain. In the forward pass, they take the inverse DFT of their filter, and then apply normal pixel-wise convolution. We wish to extend this by not only parameterizing filters in the wavelet domain, but by performing the convolution there as well (i.e., also taking the activations into the wavelet domain). After processing is done, we can return to the pixel domain. Doing these forward and inverse transforms has two significant advantages: i) the layers can easily replace standard convolutional layers if they accept and return the same format; ii) we can learn both in the wavelet and pixel space.

As neural network training involves presenting thousands of training samples, we want our layer to be fast. To achieve this we would ideally choose to use a critically sampled filter bank implementation. The fast 2-D Discrete Wavelet Transform (DWT) is a possible option, but it has two drawbacks: it has poor directional selectivity and any alteration of wavelet coefficients will cause the aliasing cancelling properties of the reconstructed signal to disappear. Instead we choose to use the Dual-Tree Complex Wavelet Transform (DT$\mathbb{C}$WT) [3] as at the expense of limited redundancy (4:1), it



(a)

(b)

(c)

Figure 5.1: (a) Contour plots at -1dB and -3dB showing the support in the Fourier domain of the 6 subbands of the DTℂWT at scales 1 and 2 and the scale 2 lowpass. These are the product $P(z)Q(z)$ from Equation 5.3.1.(b) The pixel domain impulse responses for the second scale wavelets. (c) Example impulses of our layer when g_1, and g_{lp} are 0 and $g_2 \in \mathbb{C}^{6\times1\times1}$, with each real and imaginary element drawn from $\mathcal{N}(0,1)$. I.e., only information in the 6 subbands with $\frac{\pi}{4} < |w_1|, |w_2| < \frac{\pi}{2}$ from (a) is passed through.

enables us to have better directional selectivity, and allows us to modify the wavelet coefficients and still have minimal aliasing terms when we reconstruct [19].

section 5.3 of the paper describes the implementation details of our design, and section 5.4 describes the experiments and results we have done so far.

5.3 Method

In a standard convolutional layer, an input with C channels, H rows and W columns is $X \in \mathbb{R}^{C\times H\times W}$, which is then convolved with F filters of spatial size K - $w \in \mathbb{R}^{F\times C\times K\times K}$, giving $Y \in \mathbb{R}^{F\times H\times W}$. In many systems like [50, 45], the first layer is typically a selection of bandpass filters, selecting edges with different orientations and center frequencies.

In the wavelet space this would be trivial - take a decomposition of each input channel and keep individual subbands (or equivalently, attenuate other bands), then take the inverse wavelet transform. Figure 5.1 shows the frequency space for the DT$\mathbb{C}$WT and makes it clearer as to how this could be done practically for a two scale transform. To attenuate all but say the 15° band at the first scale for the first input channel, we would need to have $13C$ gains for the 13 subbands and C input channels, $13C-1$ of which would be zero and the remaining coefficient one.

Instead of explicitly setting the gains, we can randomly initialize them and use backpropagation to learn what they should be. This gives us the power to learn more complex shapes rather than simple edges, as we can mix the regions of the frequency space per input channel in an arbitrary way.

5.3.1 Memory Cost

Again considering a two scale transform — instead of learning $w \in \mathbb{R}^{F\times C\times K\times K}$ we learn complex gains at the two scales, and a real gain for the real lowpass:

$$\left\{g_1 \in \mathbb{C}^{F\times C\times 6\times 1\times 1}, g_2 \in \mathbb{C}^{F\times C\times 6\times 1\times 1}, g_{lp} \in \mathbb{R}^{F\times C\times 1\times 1}\right\}$$

We have set the spatial dimension to be 1×1 to show that this gain is identical to a 1×1 convolution over the complex wavelet coefficients. If we wish, we can learn larger spatial sizes to have more complex attenuation/magnification of the subbands. We also can use more/fewer than 2 wavelet scales. At first glance, we have increased our parameterization by a factor of 25 (13 subbands, of which all but the lowpass are complex), but each one of these gains affects a large spatial size. For the first scale, the effective size is about 5×5 pixels, for the second scale it is about 15×15.

5.3.2 Computational Cost

A standard convolutional layer needs K^2F multiplies per input pixel (of which there are $C\times H\times W$). In comparison, the wavelet gain method does a set number of operations per pixel for the forward and inverse transforms, and then applies gains on subsampled activations. For a 2 level DT$\mathbb{C}$WT the transform overhead is about 60 multiplies for both the forward and inverse transform. It is important to note that unlike the filtering operation, this does not scale with F. The learned gains in each subband do scale with the number of output channels, but can have smaller spatial size (as they have larger effective sizes) as well as having fewer pixels to operate on (because of the decimation).

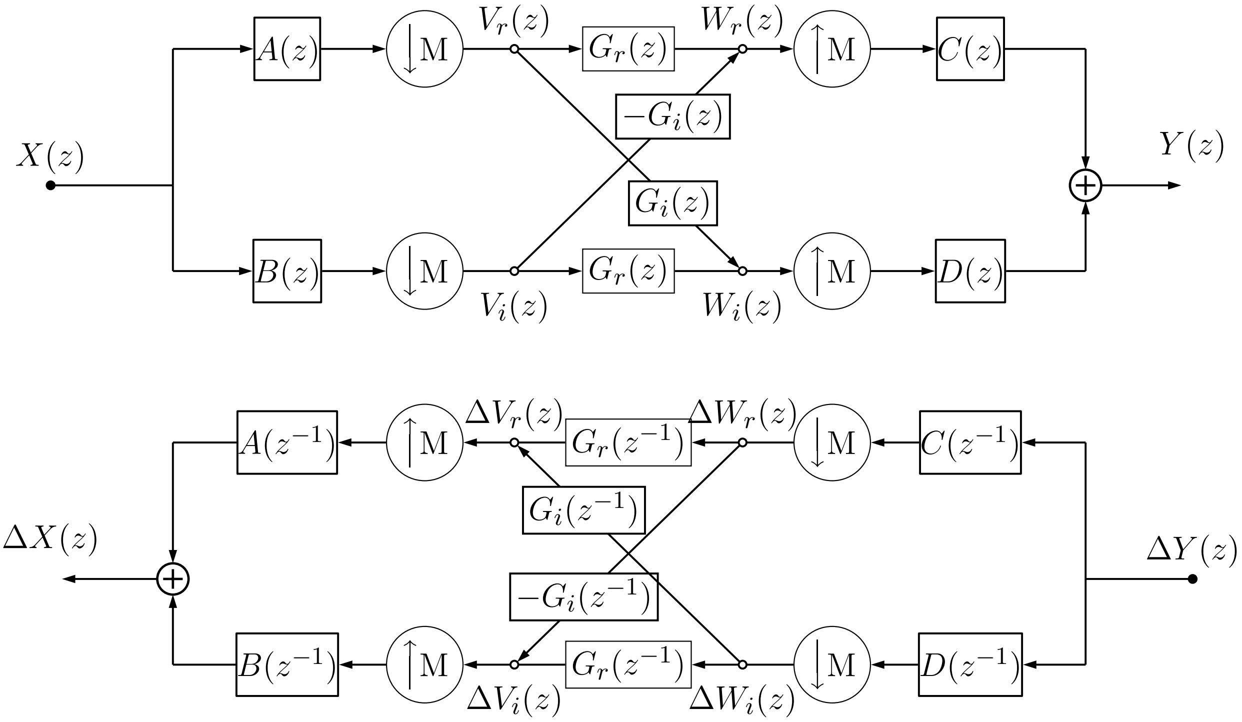


Figure 5.2: **Forward and backward block diagrams for dtcwt gain layer.** Based on Figure 4 in [19]. Ignoring the G gains, the top and bottom paths (through A, C and B, D respectively) make up the the real and imaginary parts for *one subband* of the dual tree system. Combined, $A + jB$ and $C - jD$ make the complex filters necessary to have support on one side of the Fourier domain (see Figure 5.1). Adding in the complex gain $G_r + jG_i$, we can now attenuate/shape the impulse response in each of the subbands. To allow for learning, we need backpropagation. The bottom diagram indicates how to pass gradients $\Delta Y(z)$ through the layer. Note that upsampling has become downsampling, and convolution has become convolution with the time reverse of the filter (represented by z^{-1} terms).

The end result is that as F and C grow, the overhead of the C forward and F inverse transforms is outweighed by cost of FC mixing processes, which should in turn be significantly less than the cost of FC $K \times K$ standard convolutions for equivalent spatial sizes.

5.3.3 Examples

5.1c show example impulse responses of our layer. These impulses were generated by randomly initializing both the real and imaginary parts of $g_2 \in \mathbb{C}^{6\times 1\times 1}$ from $\mathcal{N}(0,1)$ and g_1, g_{lp} are set to 0. I.e. each shape has 12 random variables. It is good to see that there is still a large degree of variability between shapes. Our experiments have shown that the distribution of the normalized cross-correlation between 512 of such randomly generated shapes matches the distribution for random vectors with roughly 11.5 degrees of freedom.

5.3.4 Forward propagation

Figure 5.2 shows the block diagram using Z-transforms for a single band of our system (it is based on Figure 4 in [19]). To keep things simple for the rest of section 5.3 the figure shown is for a 1-D system; it is relatively straightforward to extend this to 2-D[3]. The complex analysis filter (taking us into the wavelet domain) is $P(z) = \frac{1}{2}(A(z) + jB(z))$ and the complex synthesis filter (returning us to the pixel domain) is $Q(z) = \frac{1}{2}(C(z) - jD(z))$ where A, B, C, D are real. If $G(z) = G_r(z) + jG_i(z) = 1$ then the end-to-end transfer function is (from section 4 of [19]):

$$\frac{Y(z)}{X(z)} = \frac{2}{M}\left(P(z)Q(z) + P^*(z)Q^*(z)\right) \tag{5.3.1}$$

where P, Q have support only in the top half of the Fourier plane and P^*, Q^* are P and Q reflected in the horizontal frequency axis. Examples of $P(z)Q(z)$ for different subbands of a 2-D DT$\mathbb{C}$WT have spectra shown in 5.1a, $P^*(z)Q^*(z)$ make up the missing half of the frequency space.

Modifying this from the standard wavelet equations by adding the subband gains $G_r(z)$ and $G_i(z)$, the transfer function becomes:

$$\begin{aligned}\frac{Y(z)}{X(z)} = \frac{2}{M}\Big[&G_r(z^M)\left(P(z)Q(z) + P^*(z)Q^*(z)\right) + \\ &jG_i(z^M)\left(P(z)Q(z) - P^*(z)Q^*(z)\right)\Big]\end{aligned} \tag{5.3.2}$$

5.3.5 Backpropagation

We start with the commonly known property that for a convolutional block, the gradient with respect to the input is the gradient with respect to the output convolved with the time reverse of the filter. More formally, if $Y(z) = H(z)X(z)$:

$$\Delta X(z) = H(z^{-1})\Delta Y(z) \tag{5.3.3}$$

where $H(z^{-1})$ is the Z-transform of the time/space reverse of $H(z)$, $\Delta Y(z) \triangleq \frac{\partial L}{\partial Y}(z)$ is the gradient of the loss with respect to the output, and $\Delta X(z) \triangleq \frac{\partial L}{\partial X}(z)$ is the gradient of the loss with respect to the input. If H were complex, the first term in Equation 5.3.3

would be $\bar{H}(1/\bar{z})$, but as each individual block in the DT$\mathbb{C}$WT is purely real, we can use the simpler form.

Assume we already have access to the quantity $\Delta Y(z)$ (this is the input to the backwards pass). **??** illustrates the backpropagation procedure. An interesting result is that the backwards pass of an inverse wavelet transform is equivalent to doing a forward wavelet transform.[1] Similarly, the backwards pass of the forward transform is equivalent to doing the inverse transform. The weight update gradients are then calculated by finding $\Delta W(z) = \mathrm{DT}\mathbb{C}\mathrm{WT}\{\Delta Y(z)\}$ and then convolving with the time reverse of the saved wavelet coefficients from the forward pass - $V(z)$.

$$\Delta G_r(z) = \Delta W_r(z) V_r(z^{-1}) + \Delta W_i(z) V_i(z^{-1}) \tag{5.3.4}$$

$$\Delta G_i(z) = -\Delta W_r(z) V_i(z^{-1}) + \Delta W_i(z) V_r(z^{-1}) \tag{5.3.5}$$

Unsurprisingly, the passthrough gradients have similar form to Equation 5.3.2:

$$\begin{aligned}\Delta X(z) = \frac{2\Delta Y(z)}{M}\Big[&G_r(z^{-M})\left(PQ + P^*Q^*\right) + \\ &jG_i(z^{-M})\left(PQ - P^*Q^*\right)\Big]\end{aligned} \tag{5.3.6}$$

where we have dropped the z terms on $P(z), Q(z), P^*(z), Q^*(z)$ for brevity.

Note that we only need to evaluate equations 5.3.4,5.3.5,5.3.6 over the support of $G(z)$ i.e., if it is a single number we only need to calculate $\Delta G(z)|_{z=0}$.

5.4 Experiments and Preliminary Results

To examine the effectiveness of our convolutional layer, we do a simple experiment on CIFAR-10 and CIFAR-100. For simplicity, we compare the performance using a simple yet relatively effective convolutional architecture - LeNet [53]. LeNet has 2 convolutional layers of spatial size 5×5 followed by 2 fully connected layers and a softmax final layer. We swap both these convolutional layers out for two of our proposed wavelet gain layers (keeping the ReLU between them). As CIFAR has very small spatial size, we only take a single scale DT$\mathbb{C}$WT . Therefore each gain layer has 6 complex gains for the 6 subbands, and a 3×3 real gain for the lowpass (a total of $21C$ parameters vs $25C$ for the original system). We train both networks for 200 epochs

[1] As shown in **??**, the analysis and synthesis filters have to be swapped and time reversed. For orthogonal wavelet transforms, the synthesis filters are already the time reverse of the analysis filters, so no change has to be done. The q-shift filters of the DT$\mathbb{C}$WT [34] have this property.

Table 5.1: Comparison of LeNet with standard convolution to our proposed method which learns in the wavelet space (WaveLenet) on CIFAR-10 and CIFAR-100. Values reported are the average top-1 accuracy (%) rates for different train set sizes over 5 runs.

	Train set size	1000	2000	5000	10000	20000	50000
CIFAR-10	LeNet	48.5	52.4	59.5	65.0	69.5	73.3
	WaveLeNet	47.3	52.1	58.7	63.8	68.0	72.4
CIFAR-100	LeNet	11.1	15.8	23.1	29.5	34.4	41.1
	WaveLeNet	11.1	15.4	23.2	28.4	33.9	39.6

with Adam [71] optimizer with a constant learning rate of 10^{-3} and a weight decay of 10^{-5}. The code is available at [?]. Table 5.1 shows the mean of the validation set accuracies for 5 runs. The different columns represent undersampled training set sizes (with 50000 being the full training set). When undersampling, we keep the samples per class constant. We see our system perform only very slightly worse than the standard convolutional layer.

5.5 Conclusion and Future Work

In this work we have presented the novel idea of learning filters by taking activations into the wavelet domain, learning mixing coefficients and then returning to the pixel space. This work is done as a preliminary step; we ultimately hope that learning in both the wavelet and pixel space will have many advantages, but as yet it has not been explored. We have considered the possible challenges this proposes and described how a multirate system can learn through backpropagation.

Our experiments so far have been promising. We have shown that our layer can learn in an end-to-end system, achieving very near similar accuracies on CIFAR-10 and CIFAR-100 to the same system with convolutional layers instead. This is a good start and shows the plausibility of such an idea, but we need to search for how to improve these layers if they are to be useful. It will be interesting to see how well we can learn on datasets with larger images - our proposed method naturally learns large kernels, so should scale well with the image size.

In our experiments so far, we only briefly go into the wavelet domain before coming back to the pixel domain to do ReLU nonlinearities, however we plan to explore using nonlinearities in the wavelet domain, such as soft-shrinkage to denoise/sparsify the coefficients [?]. We feel there are strong links between ReLU non-linearities and denoising/sparsity ideas, and that there may well be useful performance gains from mixing real pixel-domain non-linearities with complex wavelet-domain shrinkage functions. Thus we present these ideas here as a starting point for a novel and exciting avenue of deep network research.

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 — **(author?)** [66] 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 [8] and [66]).

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 **(author?)** [63] 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 **(author?)** [72] has given a detailed comparison and analysis of how visualizations have developed since the work of **(author?)** [63]. 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 [9, 10], 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 [45]. 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 subsection 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 **??**. 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’ [54, 73], 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 **(author?)** [48] 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 subsection 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 subsection 6.2.1	
Dec–Jan	Developing ResNet Layers subsection 6.2.5	
Feb–Mar	Improved Visualizations Research subsection 6.2.3	
Apr–May	Moving to a new dataset (preparation) subsection 6.2.2	
Jun–Jul	Moving to a new dataset (testing) subsection 6.2.2	Going Deeper subsection 6.2.4
Aug–Sep	Going Deeper subsection 6.2.4	Recap on Second Year
Third Year		
Oct–Nov	Exploring the Scope of Scatternets subsection 6.2.6	
Dec–Jan	Revisiting Greedy Layer-Wise Training subsection 6.2.8	Analysing Newer CNN Layers subsection 6.2.7
Feb–Mar	Revisiting Greedy Layer-Wise Trainingsubsection 6.2.8	Analysing Newer CNN Layers subsection 6.2.7
Apr–May	Improved Visualizations Research subsection 6.2.3	
Jun–Jul	Going Deeper subsection 6.2.4	
Aug–Sep	Going Deeper subsection 6.2.4	Recap on Third Year
Fourth Year		
Oct-Dec	Writing up	

Bibliography

[1] Matthew D. Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks,” in *Computer Vision – ECCV 2014*, Sept. 2014, pp. 818–833.

[2] Karen Simonyan and Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Sept. 2014.

[3] L. Sifre and S. Mallat, “Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination,” in *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 1233–1240.

[4] Edouard Oyallon and Stephane Mallat, “Deep Roto-Translation Scattering for Object Classification,” 2015, pp. 2865–2873.

[5] Stephane Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 1998.

[6] R. R. Coifman and D. L. Donoho, “Translation-Invariant De-Noising,” in *Wavelets and Statistics*, Anestis Antoniadis and Georges Oppenheim, Eds., number 103 in Lecture Notes in Statistics, pp. 125–150. Springer New York, 1995.

[7] Ivan W. Selesnick, Richard G. Baraniuk, and Nick G. Kingsbury, “The dual-tree complex wavelet transform,” *Signal Processing Magazine, IEEE*, vol. 22, no. 6, pp. 123–151, 2005.

[8] J. Bruna and S. Mallat, “Classification with scattering operators,” in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2011, pp. 1561–1566.

[9] J. Bruna and S. Mallat, “Invariant Scattering Convolution Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, Aug. 2013.

[10] Joan Bruna, *Scattering Representations for Recognition*, Theses, Ecole Polytechnique X, Feb. 2013.

[11] Edouard Oyallon, Stéphane Mallat, and Laurent Sifre, “Generic Deep Networks with Wavelet Scattering,” *arXiv:1312.5940 [cs]*, Dec. 2013.

[12] Laurent Sifre and Stéphane Mallat, “Rigid-Motion Scattering for Texture Classification,” *arXiv:1403.1687 [cs]*, Mar. 2014.

[13] Laurent Sifre, *Rigid-Motion Scattering for Image Classification*, Ph.D. thesis, Ecole Polytechnique, Oct. 2014.

[14] L. Sifre and J. Anden, “ScatNet,” Nov. 2013.

[15] N. Kingsbury and J. Magarey, “Wavelet transforms in image processing,” 1997.

[16] Nick Kingsbury, “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)*, Utah, Aug. 1998, pp. 319–322.

[17] Nick Kingsbury, “The dual-tree complex wavelet transform: A new efficient tool for image restoration and enhancement,” in *Signal Processing Conference (EUSIPCO 1998), 9th European*, Sept. 1998, pp. 1–4.

[18] N. Kingsbury, “Image processing with complex wavelets,” *Philosophical Transactions of the Royal Society a-Mathematical Physical and Engineering Sciences*, vol. 357, no. 1760, pp. 2543–2560, Sept. 1999.

[19] N. Kingsbury, “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*, pp. 1221–1224. 1999.

[20] N. Kingsbury, “A dual-tree complex wavelet transform with improved orthogonality and symmetry properties,” 2000.

[21] N. Kingsbury, “Complex wavelets for shift invariant analysis and filtering of signals,” *Applied and Computational Harmonic Analysis*, vol. 10, no. 3, pp. 234–253, May 2001.

[22] P. de Rivaz and N. Kingsbury, “Bayesian image deconvolution and denoising using complex wavelets,” in *2001 International Conference on Image Processing, 2001. Proceedings*, Oct. 2001, vol. 2, pp. 273–276 vol.2.

[23] Yingsong Zhang and Nick Kingsbury, “A Bayesian wavelet-based multidimensional deconvolution with sub-band emphasis,” in *Engineering in Medicine and Biology Society*, 2008, pp. 3024–3027.

[24] Ganchi Zhang and Nick Kingsbury, “Variational Bayesian image restoration with group-sparse modeling of wavelet coefficients,” *Digital Signal Processing*, vol. 47, pp. 157–168, Dec. 2015.

[25] Mark Miller and Nick Kingsbury, “Image denoising using derotated complex wavelet coefficients,” *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, vol. 17, no. 9, pp. 1500–1511, Sept. 2008.

[26] S. Hatipoglu, S. K. Mitra, and N. Kingsbury, “Texture classification using dual-tree complex wavelet transform,” in *Seventh International Conference on Image Processing and Its Applications*, pp. 344–347. 1999.

[27] P. de Rivaz and N. Kingsbury, “Complex wavelet features for fast texture image retrieval,” in *1999 International Conference on Image Processing, 1999. ICIP 99. Proceedings*, 1999, vol. 1, pp. 109–113 vol.1.

[28] Patrick Loo and Nick G. Kingsbury, "Motion-estimation-based registration of geometrically distorted images for watermark recovery," Aug. 2001, pp. 606–617.

[29] H. Chen and N. Kingsbury, "Efficient Registration of Nonrigid 3-D Bodies," *IEEE Transactions on Image Processing*, vol. 21, no. 1, pp. 262–272, Jan. 2012.

[30] Julien Fauqueur, Nick Kingsbury, and Richard Anderson, "Multiscale keypoint detection using the dual-tree complex wavelet transform," in *Image Processing, 2006 IEEE International Conference On*. 2006, pp. 1625–1628, IEEE.

[31] Ryan Anderson, Nick Kingsbury, and Julien Fauqueur, "Determining Multiscale Image Feature Angles from Complex Wavelet Phases," in *Image Analysis and Recognition*, Mohamed Kamel and Aurélio Campilho, Eds., number 3656 in Lecture Notes in Computer Science, pp. 490–498. Springer Berlin Heidelberg, Sept. 2005.

[32] Ryan Anderson, Nick Kingsbury, and Julien Fauqueur, "Rotation-invariant object recognition using edge profile clusters," in *Signal Processing Conference, 2006 14th European*. 2006, pp. 1–5, IEEE.

[33] Pashmina Bendale, William Triggs, and Nick Kingsbury, "Multiscale keypoint analysis based on complex wavelets," in *BMVC 2010-British Machine Vision Conference*. 2010, pp. 49–1, BMVA Press.

[34] Ee Sin Ng and Nick G. Kingsbury, "Robust pairwise matching of interest points with complex wavelets," *Image Processing, IEEE Transactions on*, vol. 21, no. 8, pp. 3429–3442, 2012.

[35] I.W. Selesnick, "Hilbert transform pairs of wavelet bases," *IEEE Signal Processing Letters*, vol. 8, no. 6, pp. 170–173, June 2001.

[36] N. Kingsbury, "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*, Sept. 2003, vol. 1, pp. I–1013–16 vol.1.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," *arXiv:1312.6199 [cs]*, Dec. 2013.

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

[39] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang, "Recent Advances in Convolutional Neural Networks," *arXiv:1512.07108 [cs]*, Dec. 2015.

[40] Yichuan Tang, "Deep Learning using Linear Support Vector Machines," *arXiv:1306.0239 [cs, stat]*, June 2013.

[41] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, "Deep Sparse Rectifier Neural Networks," 2011, pp. 315–323.

[42] Vinod Nair and Geoffrey E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” 2010, pp. 807–814.

[43] Xavier Glorot and Yoshua Bengio, “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*, 2010.

[44] Peter Lennie, “The cost of cortical computation,” *Current biology: CB*, vol. 13, no. 6, pp. 493–497, Mar. 2003.

[45] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas, “Systematic evaluation of CNN advances on the ImageNet,” *arXiv:1606.02228 [cs]*, June 2016.

[46] Sergey Ioffe and Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *arXiv:1502.03167 [cs]*, Feb. 2015.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Identity Mappings in Deep Residual Networks,” *arXiv:1603.05027 [cs]*, Mar. 2016.

[49] Y LeCun, B Boser, J Denker, D Henderson, R Howard, W Hubbard, and L Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[50] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle, “Greedy Layer-Wise Training of Deep Networks,” 2007, pp. 153–160.

[51] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv:1207.0580 [cs]*, July 2012.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *NIPS*. 2012, pp. 1097–1105, Curran Associates, Inc.

[53] Shaoqing Ren, Kaiming He, Ross Girshick, Xiangyu Zhang, and Jian Sun, “Object Detection Networks on Convolutional Feature Maps,” *arXiv:1504.06066 [cs]*, Apr. 2015.

[54] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler, “Efficient Object Localization Using Convolutional Networks,” *CVPR 2015*, 2015.

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

[56] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going Deeper With Convolutions," 2015, pp. 1–9.

[57] Stéphane Mallat, "Group Invariant Scattering," *Communications on Pure and Applied Mathematics*, vol. 65, no. 10, pp. 1331–1398, Oct. 2012.

[58] Irène Waldspurger, Alexandre d'Aspremont, and Stéphane Mallat, "Phase Recovery, MaxCut and Complex Semidefinite Programming," *arXiv:1206.0102 [math]*, June 2012.

[59] Laurent Sifre and Stéphane Mallat, "Combined scattering for rotation invariant texture analysis," in *European Symposium on Artificial Neural Networks (ESANN) 2012*, 2012.

[60] G. Citti and A. Sarti, "A Cortical Based Model of Perceptual Completion in the Roto-Translation Space," *Journal of Mathematical Imaging and Vision*, vol. 24, no. 3, pp. 307–326, Feb. 2006.

[61] Ugo Boscain, Jean Duplaix, Jean-Paul Gauthier, and Francesco Rossi, "Anthropomorphic image reconstruction via hypoelliptic diffusion," *arXiv:1006.3735 [math]*, June 2010.

[62] Remco Duits and Bernhard Burgeth, "Scale Spaces on Lie Groups," in *Scale Space and Variational Methods in Computer Vision*, Fiorella Sgallari, Almerico Murli, and Nikos Paragios, Eds., vol. 4485, pp. 300–312. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[63] M.D. Zeiler, G.W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *2011 IEEE International Conference on Computer Vision (ICCV)*, Nov. 2011, pp. 2018–2025.

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

[65] Amarjot Singh and Nick Kingsbury, "Dual-Tree Wavelet Scattering Network with Parametric Log Transformation for Object Classification," *arXiv:1702.03267 [cs]*, Feb. 2017.

[66] Amarjot Singh and Nick Kingsbury, "Multi-Resolution Dual-Tree Wavelet Scattering Network for Signal Classification," *arXiv:1702.03345 [cs]*, Feb. 2017.

[67] Edouard Oyallon, Eugene Belilovsky, and Sergey Zagoruyko, "Scaling the Scattering Transform: Deep Hybrid Networks," *arXiv:1703.08961 [cs]*, Mar. 2017.

[68] Shin Fujieda, Kohei Takayama, and Toshiya Hachisuka, "Wavelet Convolutional Neural Networks for Texture Classification," *arXiv:1707.07394 [cs]*, July 2017.

[69] Fergal Cotter and Nick Kingsbury, "Visualizing and Improving Scattering Networks," *arXiv:1709.01355 [cs]*, Sept. 2017.

[70] Oren Rippel, Jasper Snoek, and Ryan P Adams, “Spectral Representations for Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 2440–2448. Curran Associates, Inc., 2015.

[71] Diederik Kingma and Jimmy Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Dec. 2014.

[72] Felix Grun, Christian Rupprecht, Nassir Navab, and Federico Tombari, “A Taxonomy and Library for Visualizing Learned Features in Convolutional Neural Networks,” *arXiv:1606.07757 [cs]*, June 2016.

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna, “Rethinking the Inception Architecture for Computer Vision,” *arXiv:1512.00567 [cs]*, Dec. 2015.