Chapter 1

Learning in the Wavelet Domain

In this chapter we move away from the ScatterNet ideas from the previous chapters and instead look at using the wavelet domain as a new space in which to learn. In particular the ScatterNet, and even the learnable ScatterNet proposed in the previous chapter, are built around taking complex magnitudes of the highpass wavelets. This inherently builds invariance to shifts but at the cost of making things smoother. In many ways this was beneficial, as it allowed us to subsample the output and we saw that the scattering layers worked well just before downsampling stages of a CNN. However, we would now like to explore if it is possible and at all beneficial to learn with wavelets without taking the complex magnitude. This means that the frequency support of our activations will remain in the same space in the Fourier domain.

The inspiration to this chapter is the hope that learning in the frequency/wavelet domain may afford simpler filters than learning in the pixel domain. A classic example of this is the first layer filters in AlexNet shown in Figure 1.1. These could be parameterized with only a few nonzero wavelet coefficients, or alternatively, we could take a decomposition of each input channel and keep individual subbands (or equivalently, attenuate other bands), then take the inverse wavelet transform.

Our experiments show that . . . **FINISH ME**

1.1 A Summary of Choices

As mentioned in the inspiration for this chapter, many filters that have complex support in the pixel domain would have simple support in the wavelet domain, but as the previous section showed, naively reparameterizing things in a different domain may not afford us any benefit in the optimization procedure.

There are two possible ways we can try to leverage the wavelet domain for learning:

1. We can reparameterize filters in the wavelet domain if we use nonlinear optimizers like ADAM, or ℓ_1 regularization to impose sparsity. This is presented in section 1.3.

2. We can take wavelet transforms of the inputs and learn filters on the wavelet coefficients. We can also apply nonlinearities to the wavelet coefficients such as wavelet shrinkage. On the output of this, we have the choice of either staying in the wavelet domain or returning to the pixel domain with an inverse wavelet transform. This is presented in section 1.4

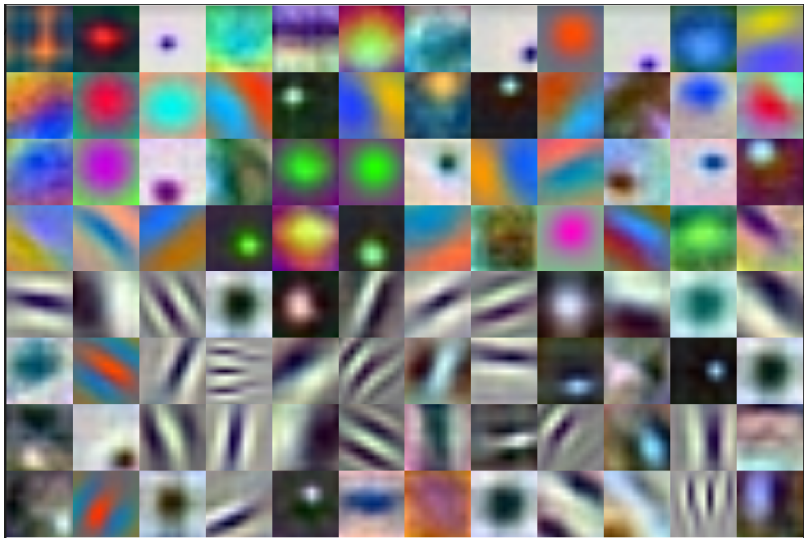


Figure 1.1: **First layer filters of the AlexNet architecture.** The first layer filters of the seminal AlexNet [1] are an inspiration for considering learning filters in the wavelet domain. Each of these 11×11 filters would only require a handful of non zero coefficients in the wavelet domain. The weights shown here were taken from a pretrained network from torchvision [2].

This chapter explores both possible methods and the merits and drawbacks of each.

1.2 Related Work

1.2.1 Wavelets as a Front End

Fujieda et. al. use a DWT in combination with a CNN to do texture classification and image annotation [3], [4]. In particular, they take a multiscale wavelet transform of the input image, combine the actviations at each scale independently with learned weights, and feed these back into the network where the activation resolution size matches the subband resolution. The architecture block diagram is shown in Figure 1.2, taken from the original paper. This work found that their dubbed 'Wavelet-CNN' could outperform competetive non wavelet based CNNs on both texture classification and image annotation.

Several works also use wavelets in deep neural networks for super-resolution [5] and for adding detail back into dense pixel-wise segmentation tasks [6]. These typically save wavelet coefficients and use them for the reconstruction phase, so are a little less applicable than the first work.

1.2.2 Parameterizing filters in Fourier Domain

In "Spectral Representations for Convolutional Neural Networks" [7], Rippel et. al. explore parameterization of filters in the DFT domain. Note that they do not necessarily do the convolution in the Frequency domain, they simply parameterize a filter $\mathbf{w} \in \mathbb{R}^{F \times C \times K \times K}$ as a set of fourier coefficients $\hat{\mathbf{w}} \in \mathbb{C}^{F \times C \times K \times \lceil K/2 \rceil}$ (the reduced spatial size is a result of enforcing that the inverse DFT of their

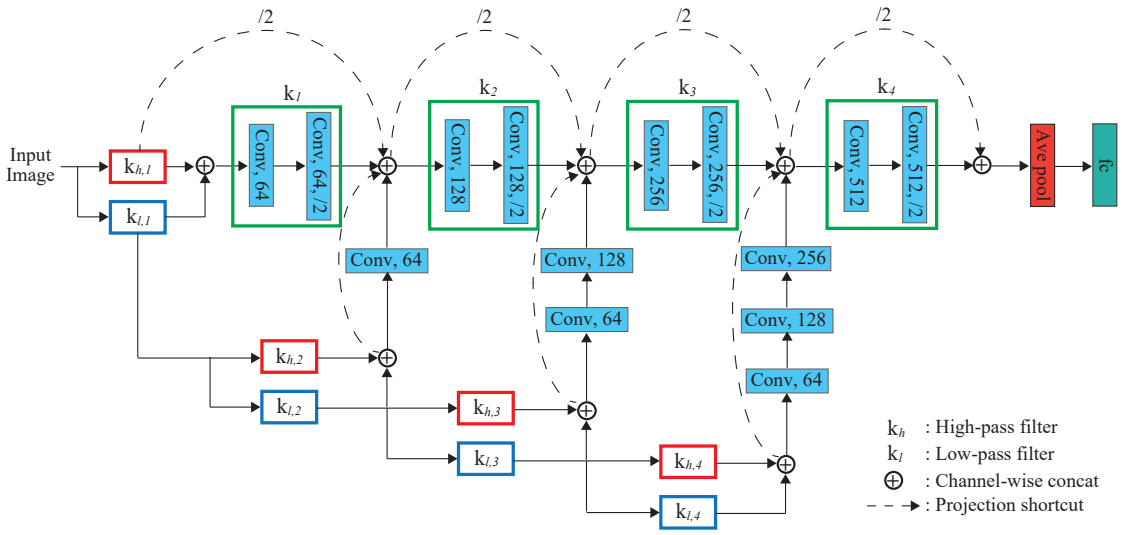


Figure 1.2: **Architecture using the DWT as a frontend to a CNN.** Figure 1 from [4]. Fujieda et. al. take a multiscale wavelet decomposition of the input before passing the input through a standard CNN. They learn convolutional layers independently on each subband and feed these back into the network at different depths, where the resolution of the subband and the network activations match.

filter to be real, so the parameterization is symmetric). On the forward pass of the neural network, they take the inverse DFT of $\hat{\mathbf{w}}$ to obtain $\mathbf{w}$ and then convolve this with the input $\mathbf{x}$ as a normal CNN would do.[1].

1.3 Parameterizing Filters in the Wavelet Domain

This is a simple extension of the work done by Rippel et. al. Their work also added another layer called 'Spectral Pooling' which effectively is a lowpass filter. As with the Fourier parameterization, a Wavelet parameterization introduces some challenges. Most notably that any filters parameterized with a decimated wavelet transform will naturally want to have a spatial support of a power of 2 whereas most convolutional filters in CNNs have an odd spatial support, typically of size 3 (and maybe more for earlier layers). This is done for a very good reason too, in that we do not want our filters to shift the activations.

[1]The convolution may be done by taking both the image and filter back into the fourier space but this is typically decided by the framework, which selects the optimal convolution strategy for the filter and input size. Note that there is not necessarily a saving to be gained by enforcing it to do convolution by product of FFTs, as the FFT size needed for the filter will likely be larger than $K \times K$, which would require resampling the coefficients

1.3.1 Invertible Transforms and Optimization

Note that an important point should be laboured about reparameterizing filters in either the wavelet or Fourier domains. That is that any invertible linear transform of the parameter space will not change the updates if a linear optimization scheme (like standard gradient descent, or SGD with momentum) is used.

To see this, let us consider the work from [7] where filters are parameterized in the Fourier domain.

If we define the DFT as the orthonormal version, i.e. let:

$$U_{ab} = \frac{1}{\sqrt{N}} \exp\{\frac{-2j\pi ab}{N}\}$$

then call $X = \text{DFT}\{x\}$. In matrix form the 2-D DFT is then:

$$\begin{aligned} X &= \text{DFT}\{x\} = UxU && (1.3.1)\\ x &= \text{DFT}^{-1}\{X\} = U^*YU^* && (1.3.2) \end{aligned}$$

When it comes to gradients, these become:

$$\begin{aligned} \frac{\partial L}{\partial X} &= U\frac{\partial L}{\partial x}U = \text{DFT}\left\{\frac{\partial L}{\partial x}\right\} && (1.3.3)\\ \frac{\partial L}{\partial x} &= U^*\frac{\partial L}{\partial X}U^* = \text{DFT}^{-1}\left\{\frac{\partial L}{\partial X}\right\} && (1.3.4) \end{aligned}$$

Now consider a single filter parameterized in the DFT and spatial domains presented with the exact same data and with the same ℓ_2 regularization ϵ and learning rate η. Let the spatial filter at time t be $\mathbf{w}_t$, the Fourier-parameterized filter be $\hat{\mathbf{w}}_t$, and let

$$\hat{\mathbf{w}}_1 = \text{DFT}\{\mathbf{w}_1\} \tag{1.3.5}$$

After presenting both systems with the same minibatch of samples $\mathcal{D}$ and calculating the gradient $\frac{\partial L}{\partial \mathbf{w}}$ we update both parameters:

$$\begin{aligned} \mathbf{w}_2 &= \mathbf{w}_1 - \eta\left(\frac{\partial L}{\partial \mathbf{w}} + \epsilon\mathbf{w}_1\right) && (1.3.6)\\ &= (1-\eta\epsilon)\mathbf{w}_1 - \eta\frac{\partial L}{\partial \mathbf{w}} && (1.3.7)\\ \hat{\mathbf{w}}_2 &= \hat{\mathbf{w}}_1 - \eta\left(\frac{\partial L}{\partial \hat{\mathbf{w}}} + \epsilon\hat{\mathbf{w}}_1\right) && (1.3.8)\\ &= (1-\eta\epsilon)\hat{\mathbf{w}}_1 - \eta\frac{\partial L}{\partial \hat{\mathbf{w}}} && (1.3.9)\\ & && (1.3.10) \end{aligned}$$

Where we have shortened the gradient of the loss evaluated at the current parameter values to $\delta_{\mathbf{w}}$ and $\delta_{\hat{\mathbf{w}}}$. We can then compare the effect the new parameters would have on the next minibatch by calculating $\text{DFT}^{-1}\{\hat{\mathbf{w}}_2\}$. Using equations 1.3.3 and 1.3.5 we then get:

$$\begin{aligned}
\text{DFT}^{-1}\{\hat{\mathbf{w}}_2\} &= \text{DFT}^{-1}\left\{(1-\eta\epsilon)\hat{\mathbf{w}}_1 - \eta\,\frac{\partial L}{\partial \hat{\mathbf{w}}}\right\} && (1.3.11)\\
&= (1-\eta\epsilon)\mathbf{w}_1 - \eta\,\text{DFT}^{-1}\left\{\frac{\partial L}{\partial \hat{\mathbf{w}}}\right\} && (1.3.12)\\
&= (1-\eta\epsilon)\mathbf{w}_1 - \eta\frac{\partial L}{\partial \mathbf{w}} && (1.3.13)\\
&= \mathbf{w}_2 && (1.3.14)
\end{aligned}$$

1.3.2 Regularization

If we use ℓ_1 then the above doesn't hold.

1.3.3 Optimization

If we us adam things r different.

This does not hold for the Adam [8] or Adagrad optimizers, which automatically rescale the learning rates for each parameter based on estimates of the parameter's variance. Rippel et. al. use this fact in their paper [7].

1.4 Taking Wavelet Transforms of Inputs

In contrast to the previous section where we only parameterized filters in the wavelet domain and transformed the filters back to the pixel domain to do convolution, this section explores learning wholly in the wavelet domain. I.e., we want to take a wavelet decomposition of the input and learn gains to apply to these coefficients, and optionally return to the pixel domain.

As neural network training involves presenting thousands of training samples on memory limited GPUs, we want our layer to be fast and as memory efficient as possible. 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. Another option is to use the DT$\mathbb{C}$WT [9]. This comes with a memory overhead which we discuss more in subsection 1.7.1, but it enables us to have have better directional selectivity and allows for the possibility of returning to the pixel domain with minimal aliasing [10].

In the next section we describe in more detail how the proposed layer works, agnostic of the wavelet transform used, before describing the differences between using the DWT and the DT$\mathbb{C}$WT.

1.4.1 Background

As we now want to consider the DWT and the DT$\mathbb{C}$WT which are both implemented as filter bank systems, we deviate slightly from the notation in the previous chapter (which was inspired by sampling a continuous wavelet transform).

Firstly, instead of talking about the continuous spatial variable $\mathbf{u}$, we now consider the discrete spatial variable $\mathbf{n} = [n_1, n_2]$. We switch to square brackets to make this clearer. With the new discrete notation, the output of a CNN at layer l is:

$$x^{(l)}[c, \mathbf{n}], \quad c \in \{0, \dots C_l - 1\}, \mathbf{n} \in \mathbb{Z}^2 \tag{1.4.1}$$

where c indexes the channel dimension. We also make use of the 2-D Z-transform to simplify our analysis:

$$X(\mathbf{z}) = \sum_{n_1} \sum_{n_2} x[n_1, n_2] z_1^{-n_1} z_2^{-n_2} = \sum_{\mathbf{n}} x[c, \mathbf{n}] \mathbf{z}^{-\mathbf{n}} \tag{1.4.2}$$

As we are working with three dimensional arrays (two spatial and one channel) but are only doing convolution in two, we introduce a slightly modified 2-D Z-transform which includes the channel index:

$$X(c, \mathbf{z}) = \sum_{n_1} \sum_{n_2} x[c, n_1, n_2] z_1^{-n_1} z_2^{-n_2} = \sum_{\mathbf{n}} x[c, \mathbf{n}] \mathbf{z}^{-\mathbf{n}} \tag{1.4.3}$$

Recall that a typical convolutional layer in a standard CNN gets the next layer's output in a two-step process:

$$y^{(l+1)}[f, \mathbf{n}] = \sum_{c=0}^{C_l - 1} x^{(l)}[c, \mathbf{n}] * h_f^{(l)}[c, \mathbf{n}] \tag{1.4.4}$$

$$x^{(l+1)}[f, \mathbf{u}] = \sigma\left(y^{(l+1)}[f, \mathbf{u}]\right) \tag{1.4.5}$$

With the new Z-transform notation introduced in (1.4.3), we can rewrite (1.4.4) as:

$$Y^{(l+1)}(f, \mathbf{z}) = \sum_{c=0}^{C_l - 1} X^{(l)}(c, \mathbf{z}) H_f^{(l)}(c, \mathbf{z}) \tag{1.4.6}$$

Note that we cannot rewrite (1.4.5) with Z-transforms as it is a nonlinear operation.

Also recall that with multirate systems, upsampling by M takes $X(z)$ to $X(z^M)$ and downsampling by M takes $X(z)$ to $\frac{1}{M} \sum_{k=0}^{M-1} X(W_M^k z^{1/k})$ where $W_M^k = e^{\frac{j2\pi k}{M}}$. We will drop the M subscript below unless it is unclear of the sample rate change, simply using W^k.

1.5 DT$\mathbb{C}$WT Single Subband Gain

Let us consider one subband of the DT$\mathbb{C}$WT. This includes the coefficients from both tree A and tree B. For simplicity in this analysis we will consider the 1-D DT$\mathbb{C}$WT without the channel parameter c. If we only keep coefficients from a given subband and set all the others to zero, then we have a reduced tree as shown in Figure 1.3. The end to end transfer function is:

$$\frac{Y(z)}{X(z)} = \frac{1}{M} \sum_{k=0}^{M-1} \Big[A(W^k z) C(z) + B(W^k z) D(z) \Big] \tag{1.5.1}$$

where the aliasing terms are formed from the addition of the rotated z transforms, i.e. when $k \neq 0$.

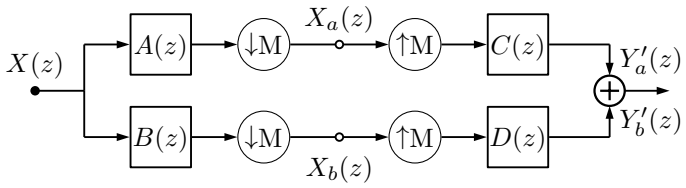


Figure 1.3: **Block Diagram of 1-D** DT$\mathbb{C}$WT**.** Note the top and bottom paths are through the wavelet or scaling functions from just level m ($M = 2^m$). Figure based on Figure 4 in [10].

Theorem 1.1. *Suppose we have complex filters $P(z)$ and $Q(z)$ with support only in the positive half of the frequency space. If $A(z) = 2\mathrm{Re}\,(P(z))$, $B(z) = 2\mathrm{Im}\,(P(z))$, $C(z) = 2\mathrm{Re}\,(Q(z))$ and $D(z) = -2\mathrm{Im}\,(Q(z))$, then the aliasing terms in (1.5.1) are nearly zero and the system is nearly shift invariant.*

Proof. See section 4 of [10] for the full proof of this, and section 7 for the bounds on what 'nearly' shift invariant means. In short, from the definition of A, B, C and D it follows that:

$$\begin{aligned} A(z) &= P(z) + P^*(z) \\ B(z) &= -j(P(z) - P^*(z)) \\ C(z) &= Q(z) + Q^*(z) \\ D(z) &= j(Q(z) - Q^*(z)) \end{aligned}$$

where $H^*(z) = \sum_n h^*[n] z^{-n}$ is the Z-transform of the complex conjugate of the complex filter h. This reflects the purely positive frequency support of $P(z)$ to a purely negative one. Substituting these into (1.5.1) gives:

$$A(W^k z)C(z) + B(W^k z)D(z) = 2P(W^k z)Q(z) + 2P^*(W^k z)Q^*(z) \tag{1.5.2}$$

Using (1.5.2), Kingsbury shows that it is easier to design single side band filters so $P(W^k z)$ does not overlap with $Q(z)$ and $P^*(W^k z)$ does not overlap with $Q^*(z)$ for $k \neq 0$. □

Using Theorem 1.1 (1.5.1) reduces to:

$$\frac{Y(z)}{X(z)} = \frac{1}{M}\left[A(z)C(z) + B(z)D(z)\right] \tag{1.5.3}$$

Let us extend this idea to allow for any linear gain applied to the passbands (not just zeros and ones). Ultimately, we may want to allow for nonlinear operations applied to the wavelet coefficients, but we initially restrict ourselves to linear gains so that we can build from a sensible base. In particular, if we want to have gains applied to the wavelet coefficients, it would be nice to maintain the shift invariant properties of the DT$\mathbb{C}$WT.

Figure 1.4 shows a block diagram of the extension of the above to general gains. This is a two port network with four individual transfer functions. Let the transfer fucntion from U_i to V_j be

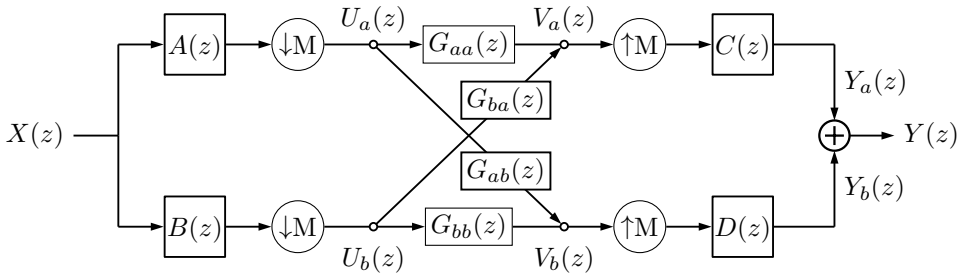


Figure 1.4: **Block Diagram of 1-D** DT$\mathbb{C}$WT**.** Note the top and bottom paths are through the wavelet or scaling functions from just level m ($M = 2^m$). Figure based on Figure 4 in [10].

G_{ij} for $i, j \in \{a, b\}$. Then V_a and V_b are:

$$\begin{aligned} V_a(z) &= U_a(z)G_{aa}(z) + U_b(z)G_{ba}(z) && (1.5.4)\\ &= \frac{1}{M}\sum_k X(W^k z^{1/k})\left[A(W^k z^{1/k})G_{aa}(z) + B(W^k z^{1/k})G_{ba}(z)\right] && (1.5.5)\\ V_b(z) &= U_a(z)G_{ab}(z) + U_b(z)G_{bb}(z) && (1.5.6)\\ &= \frac{1}{M}\sum_k X(W^k z^{1/k})\left[A(W^k z^{1/k})G_{ab}(z) + B(W^k z^{1/k})G_{bb}(z)\right] && (1.5.7) \end{aligned}$$

Further, Y_a and Y_b are:

$$\begin{aligned} Y_a(z) &= C(z)V_a(z^M) && (1.5.8)\\ Y_b(z) &= D(z)V_b(z^M) && (1.5.9) \end{aligned}$$

Then the end to end transfer function is:

$$\begin{aligned} Y(z) = Y_a(z) + Y_b(z) = \frac{1}{M}\sum_{k=0}^{M-1} X(W^k z)\Big[&A(W^k z)C(z)G_{aa}(z^k) + B(W^k z)D(z)G_{bb}(z) + \\ &B(W^k z)C(z)G_{ba}(z^k) + A(W^k z)D(z)G_{ba}(z)\Big] \end{aligned} \quad (1.5.10)$$

Theorem 1.2. *If we let $G_{aa}(z^k) = G_{bb}(z^k) = G_r(z^k)$ and $G_{ab}(z^k) = -G_{ba}(z^k) = G_i(z^k)$ then the end to end transfer function is shift invariant.*

Proof. Using the above substitutions, the terms in the square brackets of (1.5.10) become:

$$G_r(z^k)\left[A(W^k z)C(z) + B(W^k z)D(z)\right] + G_i(z^k)\left[A(W^k z)D(z) - B(W^k z)C(z)\right] \quad (1.5.11)$$

Theorem 1.1 already showed that the G_r terms are shift invariant and reduce to $A(z)C(z) + B(z)D(z)$. To prove the same for the G_i terms, we follow the same procedure. Using our definitions of A, B, C, D from Theorem 1.1 we note that:

$$\begin{aligned} A(W^k z)D(z) - B(W^k z)C(z) &= j\left[P(W^k z) + P^*(W^k z)\right][Q(z) - Q^*(z)] + && (1.5.12)\\ &\quad j\left[P(W^k z) - P^*(W^k z)\right][Q(z) + Q^*(z)] && (1.5.13)\\ &= 2j\left[P(W^k z)Q(z) - P^*(W^k z)Q^*(z)\right] && (1.5.14) \end{aligned}$$

We note that the difference between the G_r and G_i terms is just in the sign of the negative frequency parts, $AD-BC$ is the Hilbert pair of $AC+BD$. To prove shift invariance for the G_r terms in Theorem 1.1, we ensured that $P(W^k z)Q(z) \approx 0$ and $P^*(W^k z)Q^*(z) \approx 0$ for $k \neq 0$. We can use this again here to prove the shift invariance of the G_i terms in (1.5.11). This completes our proof. □

Using Theorem 1.2, the end to end transfer function with the gains is now

$$\frac{Y(z)}{X(z)} = \frac{2}{M}\left[G_r(z^M)\left(A(z)C(z)+B(z)D(z)\right)+G_i(z^M)\left(A(z)D(z)-B(z)C(z)\right)\right] \quad (1.5.15)$$

$$= \frac{2}{M}\left[G_r(z^M)\left(PQ+P^*Q^*\right)+jG_i(z^M)\left(PQ-P^*Q^*\right)\right] \quad (1.5.16)$$

Now we know can assume that our DT$\mathbb{C}$WT is well designed and extracts frequency bands at local areas, then our complex filter $G(z) = G_r(z) + jG_i(z)$ allows us to modify these passbands (e.g. by simply scaling if $G(z) = C$, or by more complex functions.

1.5.1 Backpropagation Analysis

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 $Y(z) = H(z)X(z)$:

$$\Delta X(z) = H(z^{-1})\Delta Y(z) \quad (1.5.17)$$

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 1.5.17 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.[2] Similarly, the backwards pass of the forward transform is equivalent to doing the inverse transform. The weight update gradients are then calculated by finding **FIX this - make it not use the** DT$\mathbb{C}$WT **to get delta v**. $\Delta V(z) = \text{DT}\mathbb{C}\text{WT}\{\Delta Y(z)\}$ and then convolving with the time reverse of the saved wavelet coefficients from the forward pass - $U(z)$.

$$\Delta G_r(z) = \Delta V_r(z)U_r(z^{-1}) + \Delta V_i(z)U_i(z^{-1}) \quad (1.5.18)$$

$$\Delta G_i(z) = -\Delta V_r(z)U_i(z^{-1}) + \Delta V_i(z)U_r(z^{-1}) \quad (1.5.19)$$

Unsurprisingly, the passthrough gradients have similar form to Equation 1.5.16:

$$\Delta X(z) = \frac{2\Delta Y(z)}{M}\left[G_r(z^{-M})\left(PQ+P^*Q^*\right) + jG_i(z^{-M})\left(PQ-P^*Q^*\right)\right] \quad (1.5.20)$$

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 1.5.18,1.5.19,1.5.20 over the support of $G(z)$ i.e., if it is a single number we only need to calculate $\Delta G(z)|_{z=0}$.

[2] 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 [11] have this property.

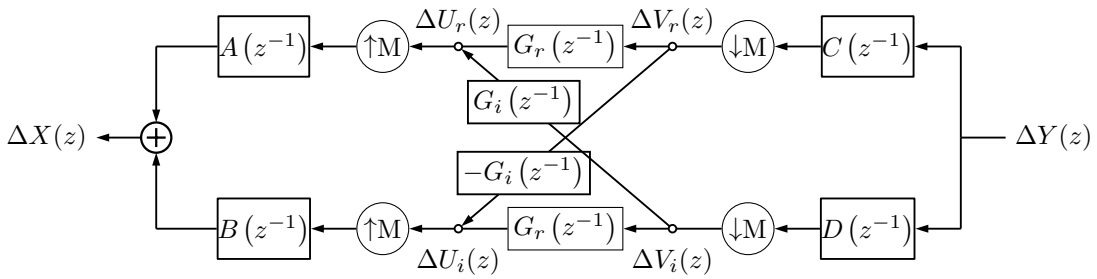


Figure 1.5: **Forward and backward block diagrams for** DT$\mathbb{C}$WT **gain layer.** Based on Figure 4 in [10]. Ignoring the G gains, the top and bottom paths (through A, C and B, D respectively) make up the the real and imaginary parts for *one subband* of the dual tree system. Combined, $A+jB$ and $C-jD$ make the complex filters necessary to have support on one side of the Fourier domain (see Figure 1.6). 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).

1.6 DT$\mathbb{C}$WT Multiple Subband Gains

Now that we have the framework for applying a complex gain at one subband, we can extend this to all of the subbands in the DT$\mathbb{C}$WT. We also reintroduce the channel dimension.

In 2-D, a J scale DT$\mathbb{C}$WT gives $6J+1$ coefficients, 6 sets of complex coefficients for each scale (representing the oriented bands from 15 to 165 degrees) and 1 set of real lowpass coefficients. Let us write this as:

$$\text{DT}\mathbb{C}\text{WT}\{x\} = \{u_{lp}, u_{j,k}\}_{1\le j\le J, 1\le k\le 6} \tag{1.6.1}$$

Sometimes we may want to refer to all of the subband coefficients at a single scale with one variable, in which case we will call them $\mathbf{u}_j$.

To do the mixing across the C_l channels at each subband, giving C_{l+1} output channels, we introduce the learnable filters:

$$\begin{aligned}
g_{lp} &\in \mathbb{R}^{C_{l+1}\times C_l\times k_h\times k_w} && (1.6.2)\\
g_{1,1} &\in \mathbb{C}^{C_{l+1}\times C_l\times k_h\times k_w} && (1.6.3)\\
g_{1,2} &\in \mathbb{C}^{C_{l+1}\times C_l\times k_h\times k_w} && (1.6.4)\\
&\vdots && (1.6.5)\\
g_{J,6} &\in \mathbb{C}^{C_{l+1}\times C_l\times k_h\times k_w} && (1.6.6)
\end{aligned}$$

where k_h, k_w are the sizes of the mixing kernels. These could be 1×1 for simple gain control, or could be larger, say 3×3, to do more complex filtering on the subbands. Let us index the lowpass filters g_{lp} and the bandpass filters $g_{j,k}$ as $g_{lp}[f, c, \mathbf{n}]$ and $g_{j,k}[f, c, \mathbf{n}]$.

With these gains we create new coefficients:

$$v_{lp}[f,\mathbf{n}] = \sum_{c=0}^{C_l-1} u_{lp}[c,\mathbf{n}] * g_{lp}[f,c,\mathbf{n}] \tag{1.6.7}$$

$$v_{1,1}[f,\mathbf{n}] = \sum_{c=0}^{C_l-1} u_{1,1}[c,\mathbf{n}] * g_{1,1}[f,c,\mathbf{n}] \tag{1.6.8}$$

$$v_{1,2}[f,\mathbf{n}] = \sum_{c=0}^{C_l-1} u_{1,2}[c,\mathbf{n}] * g_{1,2}[f,c,\mathbf{n}] \tag{1.6.9}$$

$$\vdots \tag{1.6.10}$$

$$v_{J,6}[f,\mathbf{n}] = \sum_{c=0}^{C_l-1} u_{J,6}[c,\mathbf{n}] * g_{J,6}[f,c,\mathbf{n}] \tag{1.6.11}$$

I.e., we do independent mixing at each of the different subbands. For 1×1 kernels, this is simply a matrix multiply of the wavelet coefficients. Note that for complex signals a, b the convolution $a * b$ is defined as $(a_r * b_r - a_i * b_i) + j(a_r * b_i + a_i * b_r)$.

1.6.1 Examples

Figure 1.6 show example impulse responses of our layer. These impulses were generated by randomly initializing both the real and imaginary parts of $g_{2,k} \in \mathbb{C}^{1\times 1}$ from $\mathcal{N}(0,1)$ independently for the six orientations k. $g_{1,k}, g_{lp}$ 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. Figure 1.6 shows the frequency support of the $6J+1$ subbands for a two scale DT$\mathbb{C}$WT as well as some of the equivalent impulse responses for a randomly initialized set of g filters.

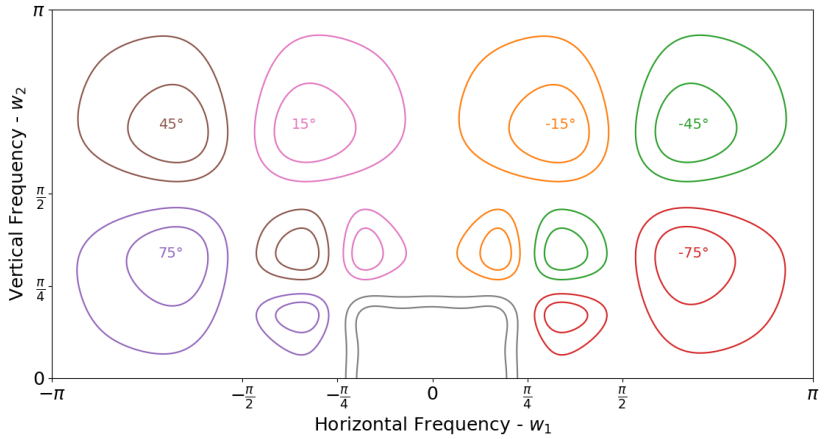
1.7 Implementation Details

We use our PyTorch Wavelets package [12] to perform the DT$\mathbb{C}$WT. As mentioned in section 1.6, our layer requires a set of learnable filters:

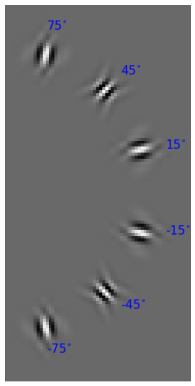
$$\{g_{lp}, g_{j,k}\}_{1\leq j\leq J, 1\leq k\leq 6} \tag{1.7.1}$$

where each of these filters is a four dimensional array as is standard for a normal convolutional layer. The lowpass coefficients of a DT$\mathbb{C}$WT are real, and therefore, g_{lp} is also real. In contrast, the bandpass coefficients are complex, and so to get the full use of the phase information of the DT$\mathbb{C}$WT, we also learn complex bandpass filters. This comes with an overhead which we will fully quantify in a later section.

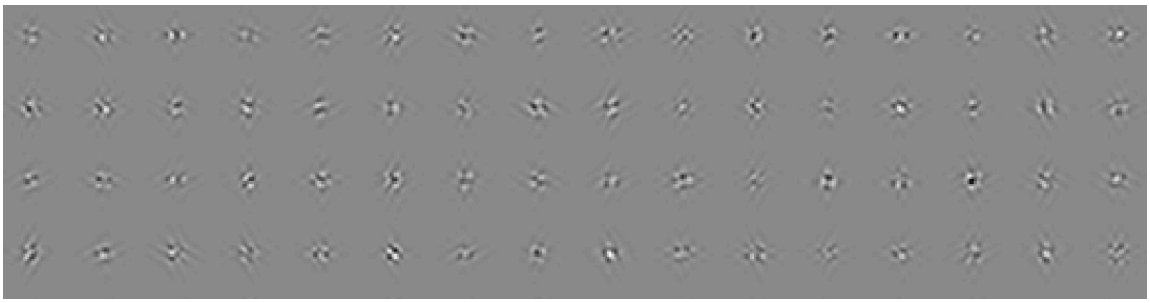
We must choose the spatial sizes of both the lowpass and bandpass mixing kernels. In our work, we set the spatial support of the complex bandpass filters $g_{j,k}$ to be 1×1 and the lowpass filters g_{lp} to be 3×3. Further, we limit ourselves initially to only considering a single scale transform. If we wish, we can learn larger spatial sizes to have more complex attenuation/magnification of the subbands. We also can use more than one wavelet scale.



(a)



(b)



(c)

Figure 1.6: **Frequency and Spatial support proposed gain layer.** (a) Contour plots at -1dB and -3dB showing the support in the Fourier domain of the 6 subbands of the DT$\mathbb{C}$WT at scales 1 and 2, and the scale 2 lowpass. These are the product $P(z)Q(z)$ from **??**.(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.

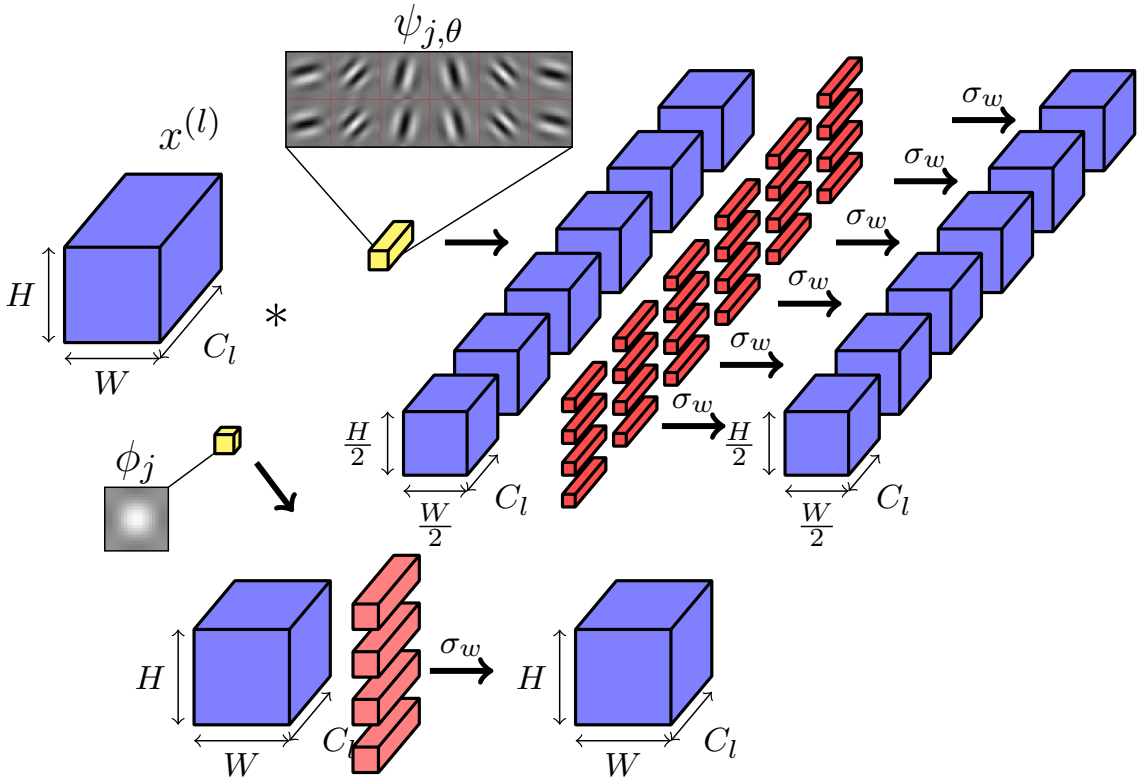


Figure 1.7: **Block Diagram of proposed** DT$\mathbb{C}$WT **gain layer for** $J = 1$**.** Activations are shaded blue, fixed parameters yellow and learned parameters red. Darker shades of blue and red indicate complex valued activations and weights, whereas the lighter shades indicate purely real values. The input $x^{(l)} \in \mathbb{R}^{C_l \times H \times W}$ is taken into the wavelet domain and each subband is mixed independently with C_{l+1} sets of convolutional filters. After mixing, a possible wavelet based nonlinearity σ_w is applied to the subbands, before returning to the pixel domain with an inverse DT$\mathbb{C}$WT.

.

1.7.1 Parameter Memory Cost

A standard convolutional layer with C_l input channels, C_{l+1} output channels and kernel size $L \times L$ has $L^2 C_l C_{l+1}$ parameters, with $L = 3$ or $L = 5$ common choices for the spatial size.

Returning to (1.6.2) – (1.6.6), let us consider a single scale ($J = 1$) wavelet gain layer. We will keep the spatial size of all the bandpass kernels the same, and allow for a differently sized lowpass gain. The filters are then:

$$\begin{aligned} g_{lp} &\in \mathbb{R}^{F \times C \times k_{lp} \times k_{lp}} \\ g_{1,1} &\in \mathbb{C}^{F \times C \times k_{bp} \times k_{bp}} \\ g_{1,2} &\in \mathbb{C}^{F \times C \times k_{bp} \times k_{bp}} \\ g_{1,3} &\in \mathbb{C}^{F \times C \times k_{bp} \times k_{bp}} \\ g_{1,4} &\in \mathbb{C}^{F \times C \times k_{bp} \times k_{bp}} \\ g_{1,5} &\in \mathbb{C}^{F \times C \times k_{bp} \times k_{bp}} \\ g_{1,6} &\in \mathbb{C}^{F \times C \times k_{bp} \times k_{bp}} \end{aligned}$$

or more compactly:

$$\begin{aligned} g_{lp} &\in \mathbb{R}^{F \times C \times k_{lp} \times k_{lp}} \\ \mathbf{g}_1 &\in \mathbb{C}^{6 \times F \times C \times k_{bp} \times k_{bp}} \end{aligned}$$

We typically set $k_{bp} = 1$ to avoid large parameter costs, but allow k_{lp} to be 1 or 3. For $k_{lp} = 3$, the total parameter cost is then (requiring 2 floats for complex values):

$$\#\text{params} = (2 \times 6 + 3^2) C_l C_{l+1} = 21 C_l C_{l+1} \tag{1.7.2}$$

At first glance, we have increased our parameterization by over twofold on a 3×3 convolution, but as with the invariant layer, the spatial support of the full filter is larger than an equivalent one parameterized in the filter domain. Note that using a second scale ($J = 2$) with 1×1 filters would increase (1.7.2) to $33 C_l C_{l+1}$.

Compared to a single scale invariant layer from the previous chapter, this requires $21 C_l C_{l+1} - 7 C_l C_{l+1} = 14 C_l C_{l+1}$ more parameters, or a threefold increase. $6 C_l C_{l+1}$ of these come from learning a complex gain rather than a purely real gain for the bandpass, and $8 C_l C_{l+1}$ come from using a larger kernel on the lowpass.

If we were to set $k_{lp} = 1$, we would minimize the relative parameter cost increase to be slightly less than $2:1$ on the invariant layer.

1.7.2 Activation Memory Cost

A standard convolutional layer needs to save the activation $x^{(l)}$ to convolve with the backpropagated gradient $\frac{\partial L}{\partial y^{(l+1)}}$ on the backwards pass (to give $\frac{\partial L}{\partial w^{(l)}}$). For an input with C_l channels of spatial size $H \times W$, this means HWC_l floats must be saved.

Our layer requires us to save the wavelet coefficients from (1.6.1), $\{u_{lp}, u_{j,k}\}_{1 \le j \le J,1}$ for updating the g terms as in (1.5.18), (1.5.19). These are $4:1$ redundant, so require $4HWC_l$ floats to be saved.

Note that a single scale layer requires $16/7$ times as many floats to be saved as compared to the invariant layer of the previous chapter. The extra cost of this comes from two things. Firstly, we keep the real and imaginary components for the bandpass (as opposed to only the magnitude), meaning we need $3HWC_l$ floats, rather than $\frac{3}{2}HWC_l$. Additionally, the lowpass was downsampled in the previous chapter, requiring only $\frac{1}{4}HWC_l$, whereas we keep the full sample rate costing HWC_l.

If memory is an issue and the computation of the DT$\mathbb{C}$WT is very fast, then we only need to save the $x^(l)$ coefficients and can calculate the u's on the fly during the backwards pass.

1.7.3 Computational Cost

A standard convolutional layer with kernel size $L \times L$ needs $L^2 C_{l+1}$ multiplies per input pixel (of which there are $C_l \times H \times W$).

The overhead calculations are the same as in **??**, so we will omit their derivation here. The mixing is however different, requiring complex convolution for the bandpass coefficients, and convolution over a higher resolution lowpass. The bandpass has one quarter spatial resolution at the first scale, but this is offset by the $4:1$ cost of complex multiplies compared to real multiplies. Again assuming we have set $J = 1$ and $k_{lp} = 3$ then the total cost for the gain layer is:

$$\underbrace{\frac{6 \times 4}{4} C_{l+1}}_{\text{bandpass}} + \underbrace{3^2 C_{l+1}}_{\text{lowpass}} + \underbrace{36}_{\text{DT}\mathbb{C}\text{WT}} + \underbrace{36}_{\text{DT}\mathbb{C}\text{WT}^{-1}} = 15C_{l+1} + 72 \quad \text{mults/pixel} \tag{1.7.3}$$

This is larger than both a standard 3×3 convolutional layer, using $9C_{l+1}$ multiplies per input pixel, and the invariant layer of the previous chapter which uses $\frac{7}{4}C_{l+1} + 36$ multiplies per input pixel.

1.7.4 Parameter Initialization

1.7.5 Forward and Backward Algorithm

Algorithm 1.

1.8 Experiments

To explore the effectiveness of our gain layer, we do a similar ablation study to **??** on CIFAR-10, CIFAR-100 and Tiny ImageNet. We use the same reference network so that we can also compare the wavelet gain layer to the invariant layer (see **??**). We also use the same naming technique, calling a network 'gainX' means that the 'convX' layer was replaced with a wavelet gain layer, but otherwise keeping the rest of the architecture the same.

We train all our networks for with stochastic gradient descent with momentum. The initial learning rate is 0.5, momentum is 0.85, batch size $N = 128$ and weight decay is 10^{-4}. For CIFAR-10/CIFAR-100 we scale the learning rate by a factor of 0.2 after 60, 80 and 100 epochs, training

Table 1.1: Results for testing VGG like architecture with convolutional and wavelet gain layers on several datasets. An architecture with 'gainX' means the equivalent convolutional layer 'convX' from **??** was swapped for our proposed layer. The top row is the reference architecture using all convolutional layers. The 'A' architecture is the originally proposed gain layer, the 'B' architecture uses the gainlayer with random shifting of the activations, and the 'C' architecture changes the mixing to be a learned 3×3 kernel acting on the invariant coefficients. Numbers are averages over 5 runs. **TODO: Rerun C models with different hypes**.

	CIFAR-10			CIFAR-100			Tiny ImgNet		
	A	B	C	A	B	C	A	B	C
reference	92.6			72.0			59.3		
gainA									
gainB									
gainC				73.0					
gainD				73.5					
gainE									
gainF									
gainA, gainB									
gainB, gainC									
gainC, gainD									
gainD, gainE									
gainA, gainC									
gainB, gainD									
gainC, gainE									

Algorithm 1 Locally Invariant Convolutional Layer forward and backward passes

procedure GAINLAYERFWD($x,\ g_{lp},\ g_1$)
  $u_{lp},\ u_1 \leftarrow \mathrm{DT\mathbb{C}WT}(x^l, \mathrm{nlevels}=1)$ ▷ u_1 has 6 orientations and is complex
  save $u_{lp},\ u_1$ ▷ For the backwards pass
  $v_{lp} \leftarrow u_{lp} * g_{lp}$
  for $1 \le k \le 6$ **do**
    $v_{1,k} \leftarrow \left(\mathrm{Re}\left(u_{1,k}\right) * \mathrm{Re}\left(g_{1,k}\right) - \mathrm{Im}\left(u_{1,k}\right) * \mathrm{Im}\left(g_{1,k}\right)\right) +$
      $j\left(\mathrm{Re}\left(u_{1,k}\right) * \mathrm{Im}\left(g_{1,k}\right) + \mathrm{Im}\left(u_{1,k}\right) * \mathrm{Re}\left(g_{1,k}\right)\right);$
  end for
  $y \leftarrow \mathrm{DT\mathbb{C}WT}^{-1}(v_{lp}, v_1)$
  return y
end procedure

procedure GAINLAYERBWD($\frac{\partial L}{\partial Y},\ g_{lp},\ g_1$)
  load $u_{lp},\ u_1$
  return $\frac{\partial L}{\partial x},\ \frac{\partial L}{\partial A}$
end procedure

for 120 epochs in total. For Tiny ImageNet, the rate change is at 18, 30 and 40 epochs (training for 45 in total).

Table 1.1 lists the results from these experiments. These show a promising start to this work. We can get an improvement in accuracy by over a percent when we use the gain layer **Finish me when results are done**.

Unlike the scatternet inspired invariant layer from the previous chapter, the gain layer does naturally downsample the output, so we are able to stack more of them? Maybe.

1.9 Wavelet Based Nonlinearities and Multiple Gain Layers

So far we have only considered linear operations in the wavelet domain. This is a good starting point, and it is nice to see that we can do operations that preserve some of the nice properties of the DT$\mathbb{C}$WT. Let us call the layer considered so far a **first order gain layer**. Recall this is where a DT$\mathbb{C}$WT is done, followed by a single linear convolution and then straight away returning to the pixel domain with the inverse DT$\mathbb{C}$WT. While it is good to see that it is possible to do and even achieves some benefit over a convolutional layer, the proposed layer (1.5.16) can be implemented in the pixel domain as a single convolution.

It would be particularly interesting if we could find a sensible nonlinearity to do in the wavelet domain. This would mean it would be no longer possibile to do the gain layer in the pixel domain. Further, we could then do multiple mixing stages in the wavelet domain before returning to the pixel domain.

But what sensible nonlinearity to use? Two particular options are good initial candidates:

1. The ReLU: this is a mainstay of most modern neural networks and has proved invaluable in the pixel domain. Perhaps its sparsifying properties will work well on wavelet coefficients too.

2. Soft thresholding: a technique commonly applied to wavelet coefficients for denoising and

compression. Many proponents of compressed sensing and dictionary learning even like to compare soft thresholding to a two sided ReLU [13], [14].

In this section we will look at each, see if they add to the first order gain layer, and see if they open the possibility of having multiple layers in the wavelet domain.

1.9.1 ReLUs in the Wavelet Domain

A ReLU could be applied to the real lowpass coefficients with ease, but it does not generalize so easily to complex coefficients. One option could be to apply it independently to the real and imaginary coefficients, effectively only selecting one quadrant of the complex plane.

One potential problem with this is that applying a ReLU independently to the real and imaginary components

1.10 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 [15]. 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.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in *NIPS*, Curran Associates, Inc., 2012, pp. 1097–1105.

[2] S. Marcel and Y. Rodriguez, "Torchvision the Machine-vision Package of Torch", in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10, New York, NY, USA: ACM, 2010, pp. 1485–1488.

[3] S. Fujieda, K. Takayama, and T. Hachisuka, "Wavelet Convolutional Neural Networks for Texture Classification", *arXiv:1707.07394 [cs]*, Jul. 2017. arXiv: 1707.07394 [cs].

[4] ——, "Wavelet Convolutional Neural Networks", *arXiv:1805.08620 [cs]*, May 2018. arXiv: 1805.08620 [cs].

[5] T. Guo, H. S. Mousavi, T. H. Vu, and V. Monga, "Deep Wavelet Prediction for Image Super-Resolution", in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Honolulu, HI, USA: IEEE, Jul. 2017, pp. 1100–1109.

[6] L. Ma, J. Stückler, T. Wu, and D. Cremers, "Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform", *arXiv:1808.01834 [cs]*, Aug. 2018. arXiv: 1808.01834 [cs].

[7] O. Rippel, J. Snoek, and R. 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., Curran Associates, Inc., 2015, pp. 2440–2448.

[8] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: 1412.6980 [cs].

[9] I. W. Selesnick, R. G. Baraniuk, and N. G. Kingsbury, "The dual-tree complex wavelet transform", *Signal Processing Magazine, IEEE*, vol. 22, no. 6, pp. 123–151, 2005.

[10] 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.

[11] ——, "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*, vol. 1, Sep. 2003, I-1013-16 vol.1.

[12] F. Cotter, *Pytorch Wavelets*, GitHub fbcotter/pytorch_wavelets, 2018.

[13] V. Papyan, Y. Romano, J. Sulam, and M. Elad, "Theoretical Foundations of Deep Learning via Sparse Representations: A Multilayer Sparse Model and Its Connection to Convolutional Neural Networks", *IEEE Signal Processing Magazine*, vol. 35, no. 4, pp. 72–89, Jul. 2018.

[14] V. Papyan, Y. Romano, and M. Elad, "Convolutional Neural Networks Analyzed via Convolutional Sparse Coding", *arXiv:1607.08194 [cs, stat]*, Jul. 2016. arXiv: 1607.08194 [cs, stat].

[15] D. L. Donoho and J. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage", en, *Biometrika*, vol. 81, no. 3, pp. 425–455, Sep. 1994.