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. With ScatterNets, complex wavelets are used to scatter the energy into different channels (corresponding to the different wavelet subbands), before the complex modulus demodulates the signal to low frequencies. These channels can then be mixed before scattering again (as we saw in the learnable scatternet), but the progressive stages all result in a steady demodulation of signal energy towards zero frequency.

In this chapter we introduce the *wavelet gain layer* which starts in a similar fashion to the ScatterNet – by taking the DT$\mathbb{C}$WT of a multi-channel input. Next, instead of taking a complex modulus, we learn a complex gain for each subband in each input channel. A single value here can amplify or attenuate all the energy in one part of the frequency plane. Then, while still in the wavelet domain, we mix the different input channels by subband (e.g. all the 15° wavelet coefficients are mixed together, but the 15° and 45° coefficients are not). We can then return to the pixel domain with the inverse wavelet transform.

We also briefly explore the possibility of doing nonlinearities in the wavelet domain. The goal being to ultimately connect multiple wavelet gain layers together with nonlinearities before returning to the pixel domain.

The proposed wavelet gain layer can then be used in conjunction with regular convolutional layers, with a network moving into the wavelet or pixel space and learning filters in one that would be difficult to learn in the other.

Our experiments so far have shown some promise. We are able to learn complex wavelet gains and a network with one or two gain layers shows small improvements. We have also found that the ReLU works well in the wavelet domain as a nonlinearity.

However, replacing a convolutional layer with a gain layer degrades performance by a small amount.

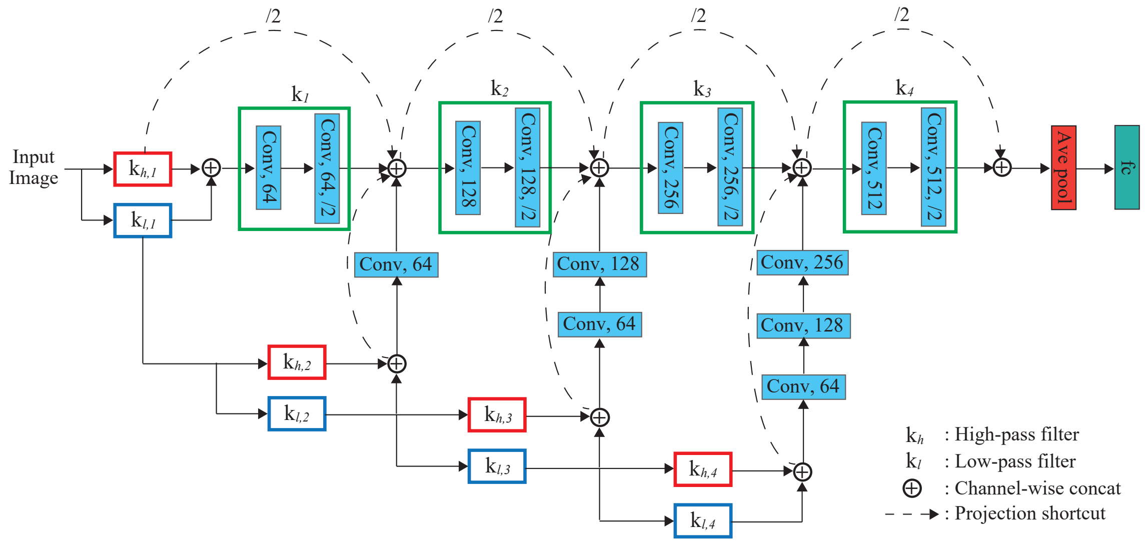


Figure 1.1: **Architecture using the DWT as a frontend to a CNN.** Figure 1 from [2]. Fujieda et. al. take a multiscale wavelet decomposition of the input before passing the input through a standard CNN. They learn convolutional layers independently on each subband and feed these back into the network at different depths, where the resolution of the subband and the network activations match.

1.1 Related Work

1.1.1 Wavelets as a Front End

Fujieda et. al. use a DWT in combination with a CNN to do texture classification and image annotation [1], [2]. 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 1.1, 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 [3] and for adding detail back into dense pixel-wise segmentation tasks [4]. These typically save wavelet coefficients and use them for the reconstruction phase.

1.2 Background and Notation

We make use of the 2-D Z-transform to simplify our analysis:

$$X(\mathbf{z}) = \sum_{n_1}\sum_{n_2} x[n_1,n_2]z_1^{-n_1}z_2^{-n_2} = \sum_{\mathbf{n}} x[c,\mathbf{n}]\mathbf{z}^{-\mathbf{n}} \tag{1.2.1}$$

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:

$$X(c,\mathbf{z}) = \sum_{n_1}\sum_{n_2} x[c,n_1,n_2]z_1^{-n_1}z_2^{-n_2} = \sum_{\mathbf{n}} x[c,\mathbf{n}]\mathbf{z}^{-\mathbf{n}} \tag{1.2.2}$$

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

$$y^{(l+1)}[f,\mathbf{n}] = \sum_{c=0}^{C_l-1} x^{(l)}[c,\mathbf{n}] * h_f^{(l)}[c,\mathbf{n}] \tag{1.2.3}$$

$$x^{(l+1)}[f,\mathbf{u}] = \sigma\left(y^{(l+1)}[f,\mathbf{u}]\right) \tag{1.2.4}$$

In shorthand, we can reduce the action of the convolutional layer in (1.2.3) to H, saying:

$$y^{(l+1)} = Hx^{(l)} \tag{1.2.5}$$

With the new Z-transform notation introduced in (1.2.2), we can rewrite (1.2.3) as:

$$Y^{(l+1)}(f,\mathbf{z}) = \sum_{c=0}^{C_l-1} X^{(l)}(c,\mathbf{z})\, H_f^{(l)}(c,\mathbf{z}) \tag{1.2.6}$$

Note that we cannot rewrite (1.2.4) 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.

1.2.1 DT$\mathbb{C}$WT Notation

For this chapter, we will work with lots of DT$\mathbb{C}$WT coefficients so we define some slightly new notation here.

A J scale DT$\mathbb{C}$WT gives $6J+1$ coefficients, 6 sets of complex bandpass coefficients for each scale (representing the oriented bands from 15 to 165 degrees) and 1 set of real lowpass coefficients.

$$\text{DT}\mathbb{C}\text{WT}_J(x) = \{u_{lp}, u_{j,k}\}_{1\leq j\leq J, 1\leq k\leq 6} \tag{1.2.7}$$

Each of these coefficients then has size:

$$u_{lp} \in \mathbb{R}^{N\times C\times \frac{H}{2^{J-1}}\times\frac{W}{2^{J-1}}} \tag{1.2.8}$$

$$u_{j,k} \in \mathbb{C}^{N\times C\times \frac{H}{2^J}\times\frac{W}{2^J}} \tag{1.2.9}$$

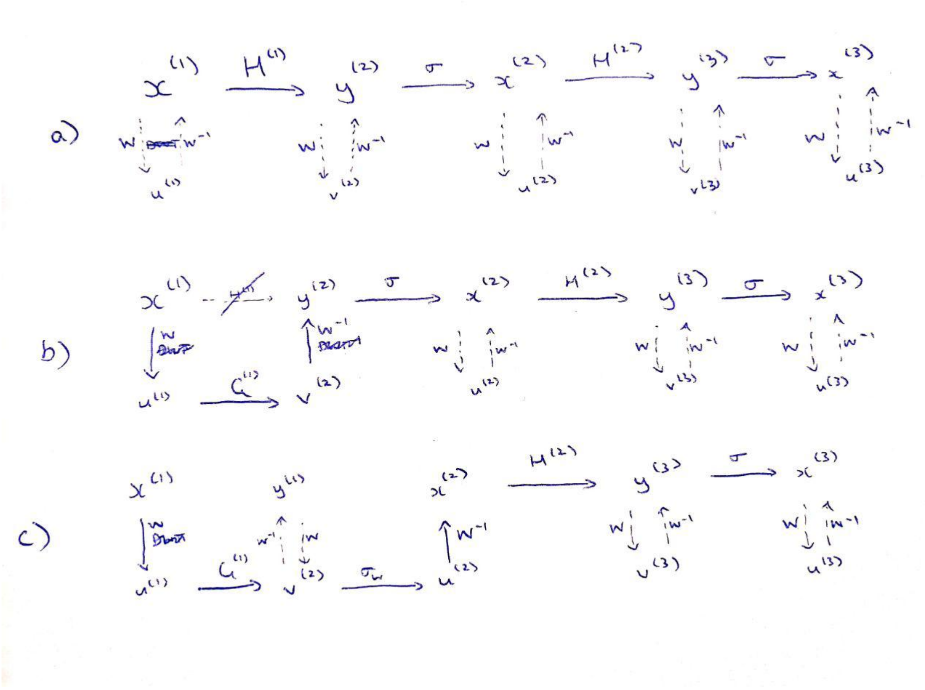


Figure 1.2: **Proposed new forward pass in the wavelet domain.** Two network layers with some possible options for processing. Solid lines denote the evaluation path and dashed lines indicate relationships. In (a) we see a regular convolutional neural network. We have included the dashed lines to make clear what we are denoting as u and v with respect to their equivalents x and y. In (b) we get to $y^{(2)}$ through a different path. First we take the wavelet transform of $x^{(1)}$ to give $u^{(1)}$, apply a wavelet gain layer $G^{(1)}$, and take the inverse wavelet transform to give $y^{(2)}$. The cross through $H^{(1)}$ indicates that this path is no longer present. Note that there may not be any possible $G^{(1)}$ to make $y^{(2)}$ from (b) equal $y^{(2)}$ from (a). In (c) we have stayed in the wavelet domain longer, and applied a wavelet nonlinearity σ_w to give $u^{(2)}$. We then return to the pixel domain to give $x^{(2)}$ and continue on from there in the pixel domain.

Note that the lowpass coefficients are twice as large as in a fully decimated transform, a feature of the redundancy of the DTℂWT.

If we ever want to refer to all the subbands at a given scale, we will drop the k subscript and call them u_j. Likewise, u refers to the whole set of DTℂWT coefficients.

1.3 Learning in Multiple Spaces

At the beginning of each stage of a neural network we have the activations $x^{(l)}$. Naturally, all of these activations have their equivalent wavelet coefficients $u^{(l)}$.

From (1.2.3), convolutional layers also have intermediate activations $y^{(l)}$. Let us differentiate these from the x coefficients and modify (1.2.7) to say the DTℂWT of $y^{(l)}$ gives $v^{(l)}$.

We now propose the *wavelet gain layer G*. The name 'gain layer' comes from the inspiration for this chapter's work, in that the first layer of CNN could theoretically be done in the wavelet domain by setting subband gains to 0 and 1.

The gain layer G can be used instead of a convolutional layer. It is designed to work on the wavelet coefficients of an activation, u to give outputs v.

This can be seen as breaking the convolutional path in Figure 1.2 and taking a new route to get to the next layer's coefficients. From here, we can return to the pixel domain by taking the corresponding inverse wavelet transform W^{-1}. Alternatively, we can stay in the wavelet domain and apply wavelet based nonlinearities for the lowpass σ_{lp} and highpass σ_{bp} coefficients to give $u^{(l+1)}$. Ultimately we would like to explore architecture design with arbitrary sections in the wavelet and pixel domain, but to do this we must first explore:

1. How effective is G at replacing H?
2. What are effective wavelet nonlinearities σ_{lp} and σ_{bp}?

1.3.1 The DTℂWT Gain Layer

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

$$g_{lp} \in \mathbb{R}^{C_{l+1} \times C_l \times k_{lp} \times k_{lp}} \tag{1.3.1}$$

$$g_{1,1} \in \mathbb{C}^{C_{l+1} \times C_l \times k_1 \times k_1} \tag{1.3.2}$$

$$g_{1,2} \in \mathbb{C}^{C_{l+1} \times C_l \times k_1 \times k_1} \tag{1.3.3}$$

$$\vdots$$

$$g_{J,6} \in \mathbb{C}^{C_{l+1} \times C_l \times k_J \times k_J} \tag{1.3.4}$$

where k is the size 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. Importantly, we can select the support size differently for each subband.

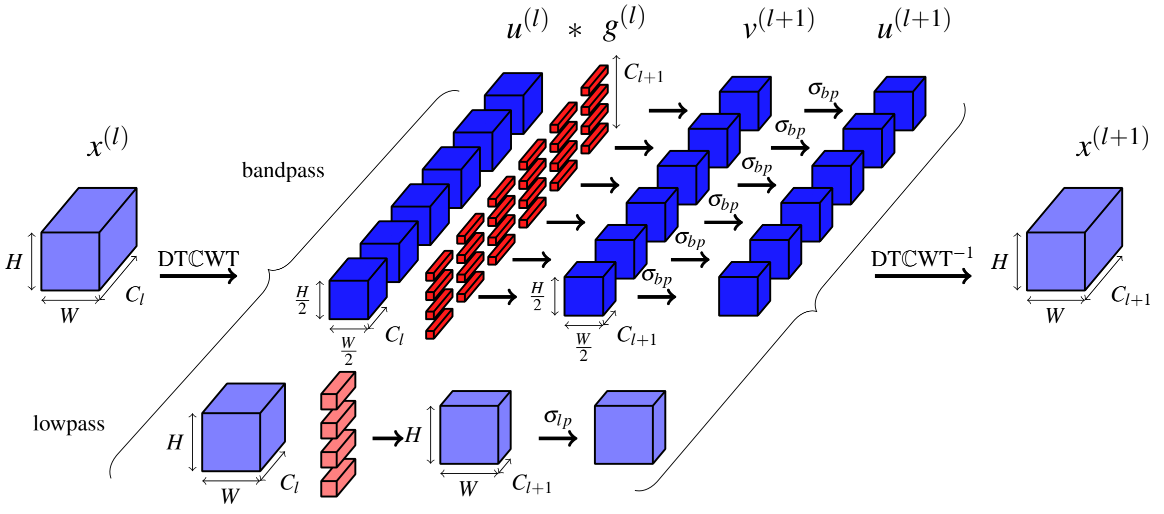


Figure 1.3: **Diagram of proposed method to learn in the wavelet domain.** Activations are shaded blue and learned parameters red. Deeper shades of blue and red indicate complex valued activations/weights, and lighter values indicate real valued activations/weights. The input $x^{(l)} \in \mathbb{R}^{C_l \times H \times W}$ is taken into the wavelet domain (here $J = 1$) and each subband is mixed independently with C_{l+1} sets of convolutional filters. After mixing, a possible wavelet nonlinearity σ_w is applied to the subbands, before returning to the pixel domain with an inverse wavelet transform.

With these gains we define the action of the gain layer $v = Gu$ to be:

$$v_{lp}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{lp}[c, \mathbf{n}] * g_{lp}[f, c, \mathbf{n}] \tag{1.3.5}$$

$$v_{1,1}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{1,1}[c, \mathbf{n}] * g_{1,1}[f, c, \mathbf{n}] \tag{1.3.6}$$

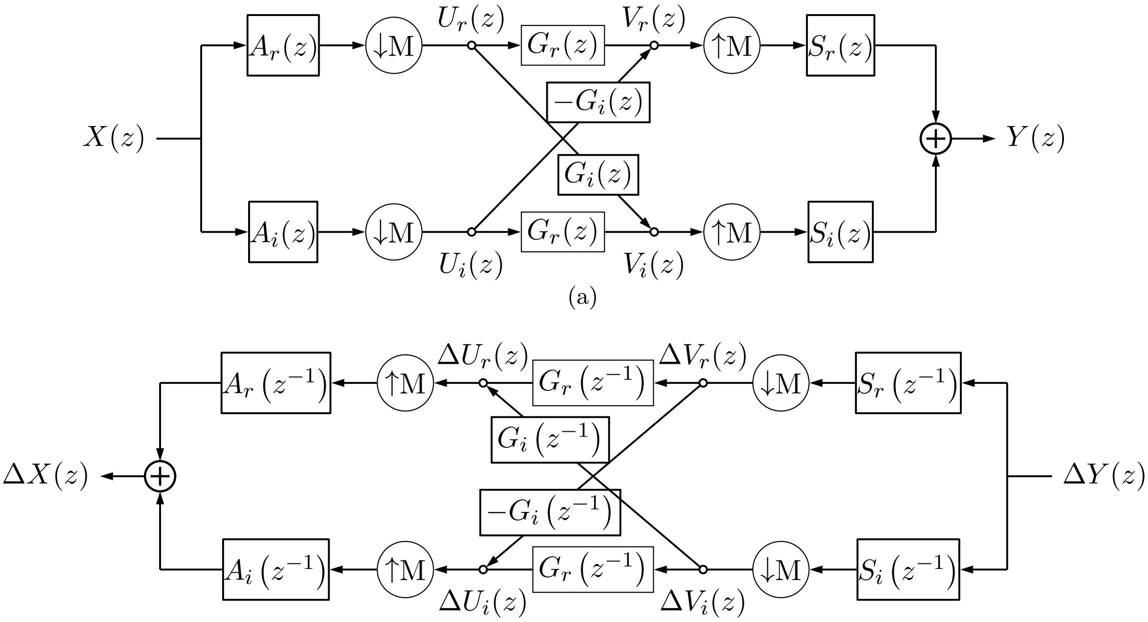
$$v_{1,2}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{1,2}[c, \mathbf{n}] * g_{1,2}[f, c, \mathbf{n}] \tag{1.3.7}$$

$$\vdots$$

$$v_{J,6}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{J,6}[c, \mathbf{n}] * g_{J,6}[f, c, \mathbf{n}] \tag{1.3.8}$$

Note that for complex signals a, b the convolution $a * b$ is defined as $(a_r * b_r - a_i * b_i) + j(a_r * b_i + a_i * b_r)$ (see **??**).

The action of the gain layer with $J = 1$ is is shown in Figure 1.3.



(a)

(b)

Figure 1.4: **Forward and backward block diagrams for** DTℂWT **gain layer.** Based on Figure 4 in [5]. Ignoring the G gains, the top and bottom paths (through A_r, S_r and A_i, S_i respectively) make up the the real and imaginary parts for *one subband* of the dual tree system. Combined, $A_r + jA_i$ and $S_r - jS_i$ make the complex filters necessary to have support on one side of the Fourier domain (see Figure 1.5). 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).

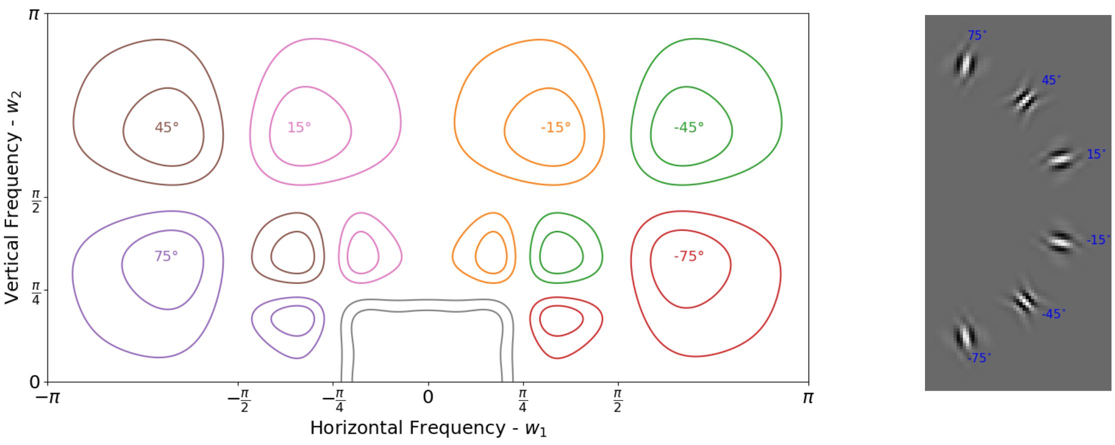
1.3.1.1 The Output

Due to the shift invariant properties of the DTℂWT, the gain layer can achieve aliasing cancelling and therefore has a transfer function. The proof of this is done in Appendix B.

Figure 1.4a shows a single subband DTℂWT based gain layer[1] Let us call the analysis filters $A = A_r + jA_i$ and the synthesis filters $S = S_r + jS_i$ (these are normally called H and G, but we keep those letters reserved for the CNN and gain layer filters). The gain for a specific subband previously was called $g_{j,k}$ but we here refer to it simply as $G = G_r + jG_i$. The output of this layer is:

$$\begin{aligned} Y(z) = \frac{2}{M} X(z) \big[& G_r(z^M)\left(A_r(z)S_r(z) + A_i(z)S_i(z)\right) \\ & + G_i(z^M)\left(A_r(z)S_i(z) - A_i(z)S_r(z)\right)\big] \end{aligned} \tag{1.3.9}$$

[1] Note that despite the resemblance to many block diagrams for fully decimated DWTs, Figure 1.4a is different. The top rung corresponds to the real part of a subband and the bottom specifies the imaginary part.



(a) (b)

Figure 1.5: DTℂWT **subbands.** (a) -1dB and -3dB contour plots 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 of the single side band filters $P(z)$ and $Q(z)$ from Theorem B.1. (b) The pixel domain impulse responses for the second scale wavelets. The Hilbert pair for each wavelet is the underlying sinusoid phase shifted by 90 degrees.

See Appendix B for the derivation. The G_r term modifies the subband gain $A_rS_r + A_iS_i$ and the G_i term modifies its Hilbert Pair $A_rS_i - A_iS_r$. Figure 1.5 show the contour plots for the frequency support of each of these subbands. The complex gain g can be used to reshape the frequency response for each subband independently.

1.3.1.2 Backpropagation

We start with the 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 (proved in **??**). More formally, if $Y(z) = H(z)X(z)$:

$$\Delta X(z) = H(z^{-1})\Delta Y(z) \tag{1.3.10}$$

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 1.3.10 would be $\bar{H}(1/\bar{z})$, but as each individual block in the DTℂWT is purely real, we can use the simpler form $H(z^{-1})$.

Assume we already have access to the quantity $\Delta Y(z)$ (this is the input to the backwards pass). Figure 1.4b illustrates the backpropagation procedure.

Let us calculate $\Delta V_r(z)$ and $\Delta V_i(z)$ by backpropagating $\Delta Y(z)$ through the inverse DTℂWT. This is the same as doing the forward DTℂWT on $\Delta Y(z)$ with the synthesis and analysis filters swapped and time reversed[2]. Then the weight update equations are:

$$\Delta G_r(z) = \Delta V_r(z) U_r(z^{-1}) + \Delta V_i(z) U_i(z^{-1}) \tag{1.3.11}$$

$$\Delta G_i(z) = -\Delta V_r(z) U_i(z^{-1}) + \Delta V_i(z) U_r(z^{-1}) \tag{1.3.12}$$

The passthrough equations have similar form to (1.3.9):

$$\Delta X(z) = \frac{2\Delta Y(z)}{M} \left[G_r(z^{-M}) \left(A_r(z) S_r(z) + A_i(z) S_i(z) \right) + j G_i(z^{-M}) \left(A_r(z) S_i(z) - A_i(z) S_r(z) \right) \right] \tag{1.3.13}$$

1.3.2 Examples

Figure 1.6 show example impulse responses of the DT$\mathbb{C}$WT gain layer. For comparison, we also show similar ‘impulse responses’ for a gain layer done in the DWT domain[3]. The DWT outputs come from three random variables: a 1×1 convolutional weight applied to each of the low-high, high-low and high-high subbands. The DT$\mathbb{C}$WT outputs come from twelve random variables, again a 1×1 convolutional weight, but now applied to six complex subbands. Our experiments have shown that the distribution of the normalized cross-correlation between 512 of such randomly generated shapes for the DWT matches the distribution for random vectors with roughly 2.8 degrees of freedom (c.f. 3 random variables in the layer). Similarly for the DT$\mathbb{C}$WT, the distribution of the normalized cross-correlation matches the distribution for random vectors with roughtly 11.5 degrees of freedom (c.f. 12 random variables in the layer). This is particularly reassuring for the DT$\mathbb{C}$WT as it is showing that there is still representatitve power despite the redundancy of the transform.

1.3.3 Implementation Details

Before analyzing its performance, we compare the implementation properties of our proposed layer to a standard convolutional layer.

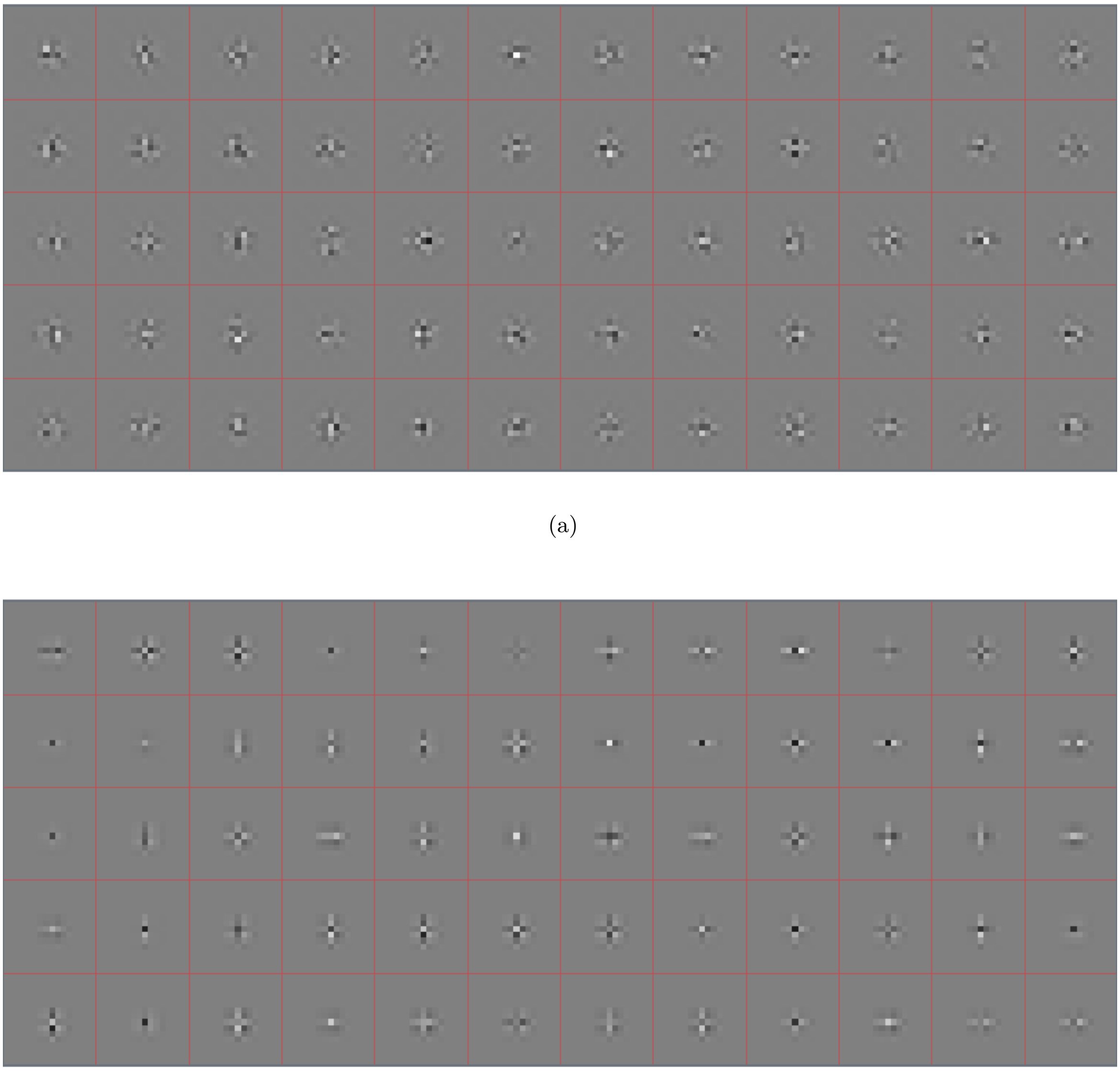
1.3.3.1 Parameter Memory Cost

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

$$\#\text{conv params} = k^2 C_l C_{l+1} \tag{1.3.14}$$

[2] An interesting result is that for orthogonal wavelet transforms, $S(z^{-1}) = A(z)$, so the backwards pass of an inverse wavelet transform is equivalent to doing a forward wavelet transform. Similarly, the backwards pass of the forward transform is equivalent to doing the inverse transform.

[3] Modifying DWT coefficients causes a loss of the alias cancellation properties so these are not true impulse response.



(a)

(b)

Figure 1.6: **Example outputs from an impulse input for the proposed gain layers.** Example outputs $y = W^{-1}GWx$ for an impulse x for the DT$\mathbb{C}$WT gain layer and for a similarly designed DWT gain layer. (a) shows the output y for a DT$\mathbb{C}$WT based system. $g_{lp} = 0$ and g_1 has spatial size 1×1. The 12 values in g_1 are independently sampled from a random normal of variance 1. The 60 samples come from 60 different random draws of the weights. (b) shows the outputs y when x is an impulse and W is the DWT with a 'db2' wavelet family. The strong horizontal and vertical properties of the DWT can clearly be seen in comparison to the much freer DT$\mathbb{C}$WT.

We must choose the spatial sizes of both the lowpass and bandpass mixing kernels. In our work, we are somewhat limited in how large we would like to set the bandpass spatial size, as every extra pixel of support requires $2 \times 6 = 12$ extra parameters. For this reason, we almost always set it to have support 1×1. The lowpass gains are less costly, and we are free to set them to size $k_{lp} \times k_{lp}$ (with $k_{lp} = 1, 3, 5$ in many of our experiments). Further, due to the size of the datasets we test on, we typically limit ourselves initially to only considering a single scale. If we wish, we can decompose the input into more scales, resulting in a larger net area of effect. In particular, it may be useful to do a two scale transform and discard the first scale coefficients. This does not increase the number of gains to learn, but changes the position of the bands in the frequency space.

The number of parameters for the gain layer with $k_{lp} = 1$ is then:

$$\#\text{params} = (2 \times 6 + 1) C_l C_{l+1} = 13 C_l C_{l+1} \tag{1.3.15}$$

This is slightly larger than the $9C_lC_{l+1}$ parameters used in a standard 3×3 convolution, but as Figure 1.6 shows, the spatial support of the full filter is larger than an equivalent one parameterized in the filter domain. If $k_{lp} = 3$ then we would have $21C_lC_{l+1}$ parameters, slightly fewer than a 5×5 convolution.

1.3.3.2 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

$$\#\text{conv floats} = HWC_l \tag{1.3.16}$$

Our layers require us to save the wavelet coefficients u_{lp} and $u_{j,k}$ for updating the g terms as in (1.3.11) and (1.3.12). For the $4:1$ redundant DT$\mathbb{C}$WT, this requires:

$$\#\text{DT}\mathbb{C}\text{WT floats} = 4HWC_l \tag{1.3.17}$$

to be saved for the backwards pass. You can see this difference from the difference in the block diagrams in Figure 1.3.

Note that a single scale DT$\mathbb{C}$WT gain 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. Note that a two scale DT$\mathbb{C}$WT gain layer would still only require $4HWC_l$ floats.

1.3.3.3 Computational Cost

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

For the DT$\mathbb{C}$WT, 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 $k_{lp} = 1$ then the total cost for the gain layer is:

$$\#\text{mults/pixel} = \underbrace{\frac{6 \times 4}{4} C_{l+1}}_{\text{bandpass}} + \underbrace{C_{l+1}}_{\text{lowpass}} + \underbrace{36}_{\text{DT}\mathbb{C}\text{WT}} + \underbrace{36}_{\text{DT}\mathbb{C}\text{WT}^{-1}} = 7C_{l+1} + 72 \tag{1.3.18}$$

which is marginally smaller than a 3×3 convolutional layer.

1.3.3.4 Parameter Initialization

For both layer types we use the Glorot Initialization scheme [6] with $a = 1$:

$$g_{ij} \sim U\left[-\sqrt{\frac{6}{(C_l + C_{l+1})k^2}}, \sqrt{\frac{6}{(C_l + C_{l+1})k^2}}\right] \tag{1.3.19}$$

where k is the kernel size.

1.4 Gain Layer Experiments

Before we explore the possibilities and performance of using a nonlinearity in the wavelet domain, let us present some experiments and results for the wavelet gain layer. This is the first objective in section 1.3, comparing G to H.

1.4.1 CNN activation regression

One of the early inspriations for using wavelets in CNNs was the visualizations of the first layer filters learned in AlexNet. These 11×11 colour filters (see **??**) look very much like a 2-D oriented wavelet transform.

So how well can the gain layer emulate the action of this layer? How would it compare to trying to use a reduced size convolutional kernel to learn the action of the layer?

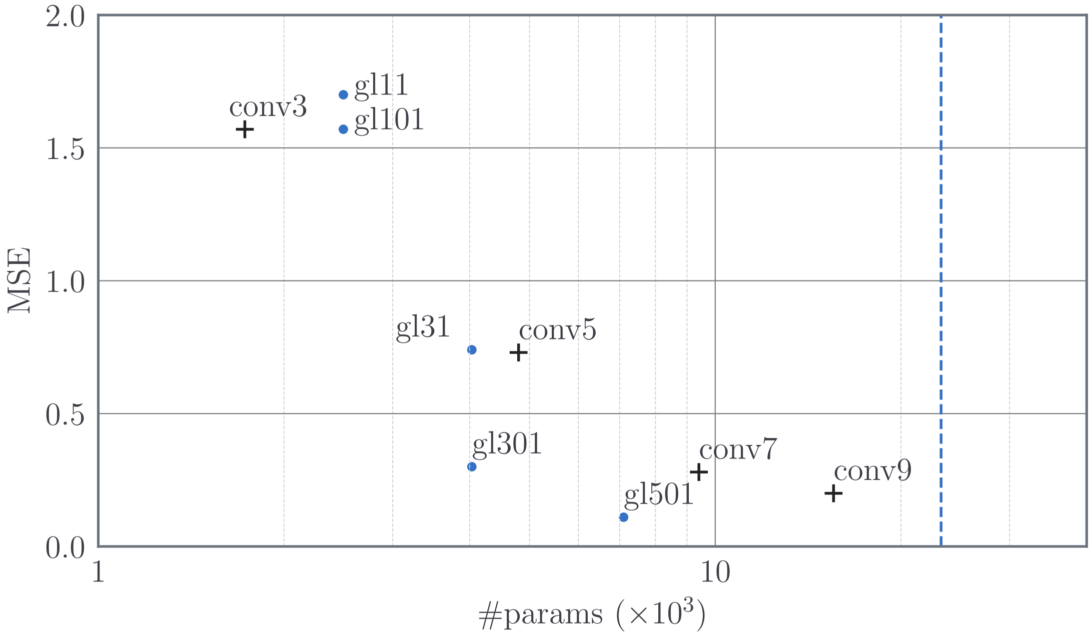


Figure 1.7: **Mean Squared Error for Conv and Wavelet Gain Layer Regression with AlexNet first layer filters.** After minimization of (1.4.1) and (1.4.2), this plot shows the final MSE score compared to the number of learnable parameters. The original conv layer has spatial support 11×11, and the equivalent number of parameters is shown as a blue dotted line. The four points labelled 'conv$n$' correspond to filters with $n \times n$ spatial support. The four points labelled 'glabc' correspond to two scale gain layers with $a \times a$ support in the lowpass, $b \times b$ spatial support in the first scale, and $c \times c$ spatial support in the second scale. The gain layer can regress to the AlexNet filters quite capably. In this example, it is important to have at least 3×3 lowpass support for the gain layer, and the second scale coefficients are more important than the first scale.

Let us call the action of our target layer H_0, our convolutional layer H and our gain layer G. Let $\|H\|_2$, $\|G\|_2$ be the ℓ_2 norm of the weights for each layer. We would like assume that we do not have direct access to H_0 but only the convolved outputs $Y = H_0 X$. Then, we would like to solve:

$$\underset{H}{\arg\min}\ (Y - HX)^2 + \frac{\lambda}{2}\|H\|_2^2, \quad \text{s.t. } h[c,\mathbf{n}] = 0,\ \forall \mathbf{n} \notin \mathcal{R} \tag{1.4.1}$$

$$\underset{G}{\arg\min}\ (Y - W^{-1}GWX)^2 + \frac{\lambda}{2}\|G\|_2^2, \quad \text{s.t. } g_{j,k}[c,\mathbf{n}] = 0,\ \forall \mathbf{n} \notin \mathcal{R}' \tag{1.4.2}$$

for some support regions $\mathcal{R}, \mathcal{R}'$. E.g. $\mathcal{R}$ could be a 3×3 or 5×5 block, and similarly $\mathcal{R}'$ could define a desired support for each gain in each subband.

(1.4.1) and (1.4.2) are convex regression problems, with many possible ways to solve. We are not worried with the optimization procedure chosen here, but of the final distances $\|Y - HX\|$ and $\left\|Y - W^{-1}GWx\right\|$ (or equivalently, their squares). We choose to find H and

G by gradient descent, using the validation set for ImageNet as the data input-output pair (X, Y). After 3–5 epochs, both H and G typically settle into their global minimum. Because of the large size of the input filters, we allow for both a $J = 1$ and $J = 2$ scale gain layer, but only learn weights at the lowest frequency bandpass (i.e. for a 2 scale gain layer, we discard the first scale highpass outputs and only learn g_2).

The resulting MSE are shown in Figure 1.7. A label 'glab' indicates a single scale gain layer with $a \times a$ support in the lowpass and $b \times b$ support in the highpass; a label 'glabc' indicates a two scale gain layer with $a \times a$ support in the lowpass, $b \times b$ support in the scale 1 highpass and $c \times c$ support in the scale 2 bandpass gains.

This figure shows several interesting things yet unsurprising things. Firstly, bigger lowpass support is very helpful – see the difference between gl101, gl301, and gl501, 3 instances that only vary in the size of the support of their lowpass filter g_{lp}. Additionally, the second scale coefficients appear more useful than the first scale – see the difference between gl310 and gl301, two instances that have the same number of parameters, but gl310 has g_1 with non-zero support, and gl301 has g_2 with non-zero support.

1.4.2 Ablation Studies

Figure 1.7 is a useful guide on how the gainlayer might be placed in a deep CNN. gl110 (a gain layer with a 1×1 lowpass kernel and a 1×1 bandpass kernel at the first scale), gl101 (same as gl110 but no gain at first scale and 1×1 at second scale), and conv3 all achieve similar MSEs. Additionally gl310, gl301, and conv5 all achieve similar MSEs.

Most modern CNNs are built with 3×3 kernels, which may not well be the best use for the gain layer. For this reason, we deviate from the ablation study done in the previous chapter, and build a shallower network with larger kernel sizes [4].

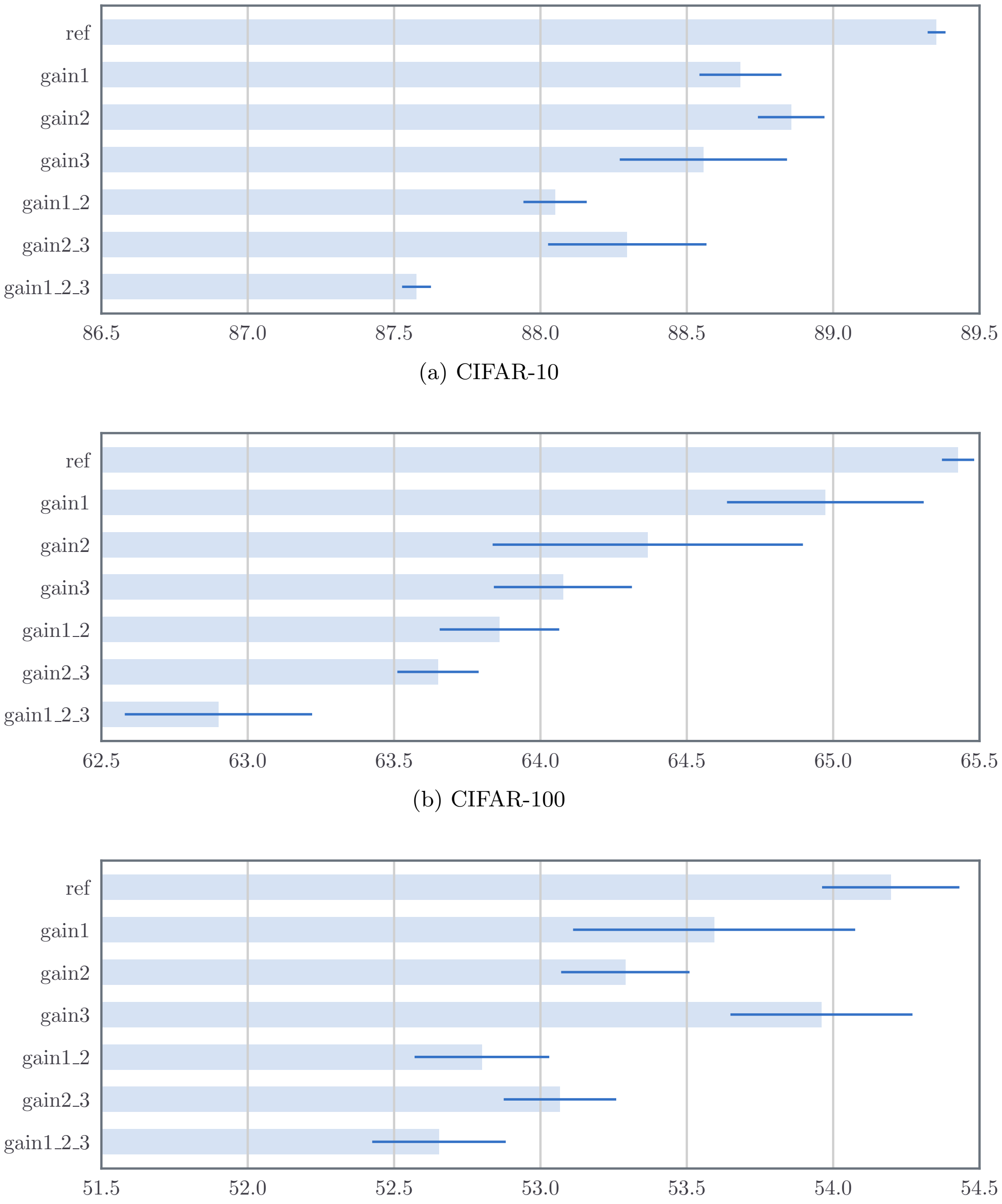
1.4.2.1 Large Kernel Ablation

In this experiment, we build a 3 layer CNN with 5×5 convolutional kernels, described in Table 1.1. To help differentiate with the small kernel network introudeced in the ablation study of the previous chapter, we have labelled the convolutions here 'conv1', 'conv2' and 'conv3' (as opposed to 'convA', 'convB', 'convC', ...).

We then test the difference in accuracy achieved by replacing each of the three convolution layers with gl310[5]. On the two CIFAR datasets, we train for 120 epochs, decaying learning rate by a factor of 0.2 at 60, 80 and 100 epochs, and for the Tiny ImageNet dataset, we train for 45 epochs, decaying learning rate at 18, 30 and 40 epochs. The experiment code is available at.

[4] We also include the experiment results for a deeper network with smaller kernels in Appendix D.

[5] Although the gain layers with no gain in the first scale and gain in the second scale performed better than those with gain gl310 and gl510 in subsection 1.4.1, we saw them perform consinstently worse in the following ablation studies. For ease of presentation, we have shown only the results from the single scale gain layer.



(a) CIFAR-10

(b) CIFAR-100

(c) Tiny ImageNet

Figure 1.8: **Large kernel ablation results CIFAR and Tiny ImageNet.** Results showing the accuracies obtained by swapping combinations of the three conv layers in the reference architecture from Table 1.1 with gain layers. Results shown are averages of 3 runs with the standard deviations shown as dark blue lines. These results show that changing a convolutional layer for a gain layer is possible, but comes with a small accuracy cost which compounds as more layers are swapped.

Table 1.1: **Ablation Base Architecture.** Reference architecture used for experiments on CIFAR-10, CIFAR-100 and Tiny ImageNet. The activation size rows are offset from the layer description rows to convey the input and output shapes. Unlike **??**, this architecture is shallower and uses 5×5 convolutional kernels as a base. C is a hyperparameter that controls the network width, we use $C = 64$ for our tests.

Activation Size	Reference Arch.		Alternate Arch.
$3 \times 32 \times 32$			
	conv1, $w \in \mathbb{R}^{C\times 3\times 5\times 5}$	*or*	gain1, $g_{lp} \in \mathbb{R}^{C\times 3\times 3\times 3}$, $g_1 \in \mathbb{C}^{C\times 6\times 3\times 1\times 1}$
$C \times 32 \times 32$			
	batchnorm + relu		
$C \times 32 \times 32$			
	pool1, max pool 2×2		
$C \times 16 \times 16$			
	conv2, $w \in \mathbb{R}^{2C\times C\times 5\times 5}$	*or*	gain2, $g_{lp} \in \mathbb{R}^{2C\times C\times 5\times 5}$, $g_1 \in \mathbb{C}^{2C\times 6\times C\times 1\times 1}$
$2C \times 16 \times 16$			
	batchnorm + relu		
$2C \times 16 \times 16$			
	pool2, max pool 2×2		
$2C \times 8 \times 8$			
	conv3, $w \in \mathbb{R}^{4C\times 2C\times 5\times 5}$	*or*	gain3, $g_{lp} \in \mathbb{R}^{4C\times 2C\times 5\times 5}$, $g_1 \in \mathbb{C}^{4C\times 6\times 2C\times 1\times 1}$
$4C \times 8 \times 8$			
	batchnorm + relu		
$4C \times 8 \times 8$			
	avg, 8×8 average pool		
$4C \times 1 \times 1$			
	fc1, fully connected		
10, 100			

The results of various swaps for our three datasets are shown in Figure D.2. Note that as before, swapping ‘conv1’ with a gain layer is marked by ‘gain1’, and swapping the first two conv layers with two gain layers is marked by ‘gain1_2’ and so forth.

The results are not too promising. Across all three datasets, changing a convolutional layer for a gain layer of similar number of parameters results in a small decrease in accuracy at all depths, and the more layers swapped out the more this degradation compounds.

1.4.3 Network Analysis

It is nonetheless interesting to see that a network with only gain layers (‘gain1_2_3’) can get accuracies within a couple of percentage points of a purely convolutional architecture.

In this section, we look at some of the properties of the ‘gain1_2_3’ for CIFAR-100 and compare them to the reference architecture.

1.4.3.1 Bandpass Coefficients

When analyzing the ‘gain1_2_3’ architecture, the most noticeable thing is the distribution of the bandpass gain magnitudes. Figure 1.9a shows these for the second gain layer, gain2. Of the $64 \times 128 = 8192$ complex coefficients most have very small magnitude, in particular the diagonal wavelet gains. This raises an interesting question – how many of these coefficients are important for classification? What if we were to apply a hard thresholding scheme to the weights, would setting some of these values to 0 impact the entire network accuracy?

We measure the dropoff in accuracy when a hard threshold t is applied to the bandpass gains g_1 for the three gain layers of 'gain1_2_3'. The resulting sparsity of each layer and the network performance is shown in Figure 1.9b. This figure shows that despite the high cost of the bandpass gains – $12C_lC_{l+1}$ for a 1×1 gain, very few of these need to be nonzero.

1.4.3.2 DeConvolution and Filter Sensitivity

1.5 Wavelet Based Nonliearities

Returning to the goals from section 1.3, the experiments from the previous section have shown that while it is possible to use a wavelet gain layer (G) in place of a convolutional layer (H), this may come with a small performance penalty. Ignoring this effect for the moment, in this section, we continue with our investigations into learning in the wavelet domain. In particular, is it possible to replace a pixel domain nonlinearity σ with a wavelet based one σ_w?

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. 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 [7], [8].

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

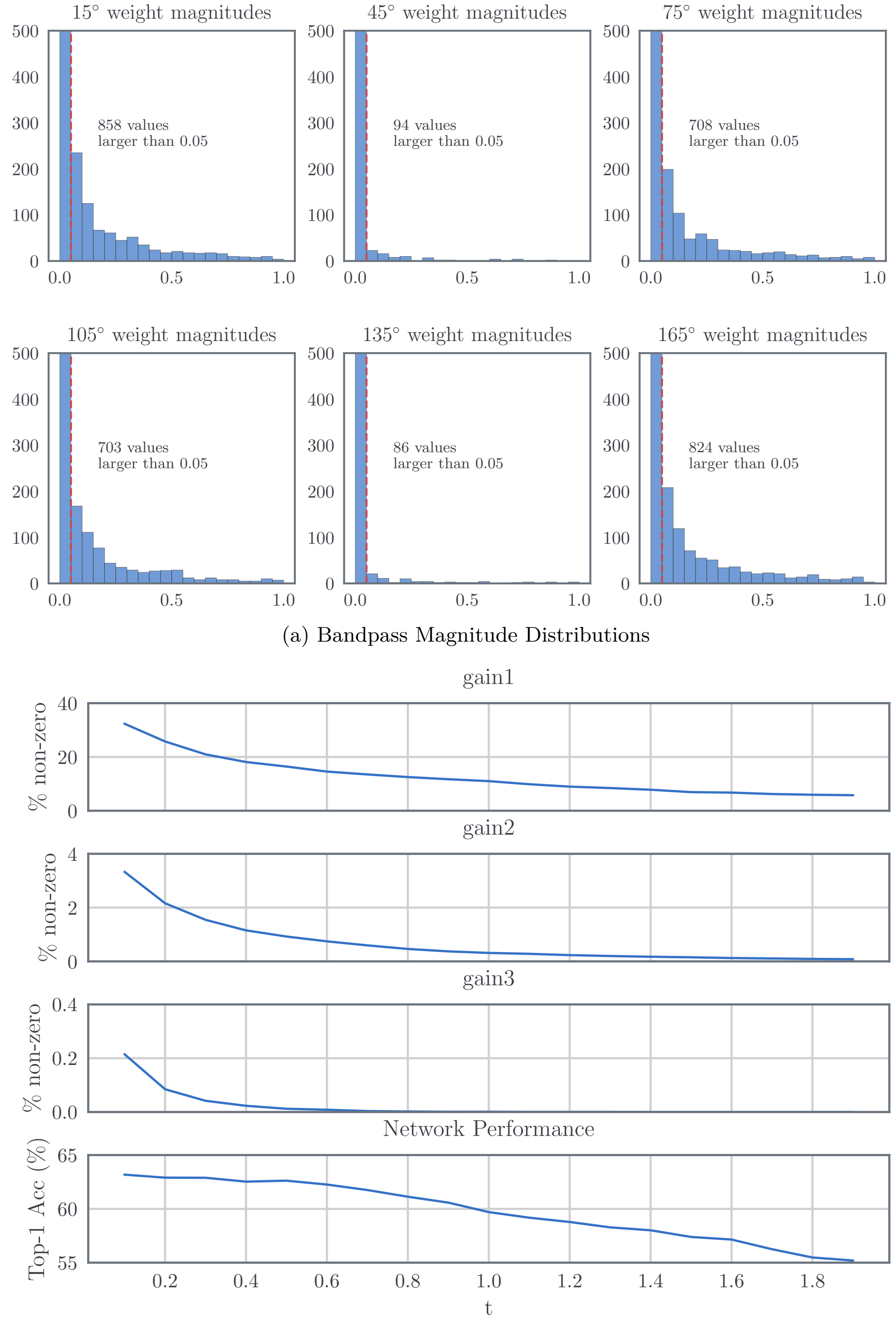
1.5.1 ReLUs in the Wavelet Domain

Applying the ReLU to the real lowpass coefficients is not difficult, but it does not generalize so easily to complex coefficients. The simplest option is to apply it independently to the real and imaginary coefficients, effectively only selecting one quadrant of the complex plane:

$$u_{lp} = \max(0,\ v_{lp}) \tag{1.5.1}$$

$$u_j = \max(0,\ \mathrm{Re}(v_j)) + j\max(0,\ \mathrm{Im}(v_j)) \tag{1.5.2}$$

Another option is to apply it to the magnitude of the bandpass coefficients. Of course these are all strictly positive so the ReLU on its own would not do anything. However,



(a) Bandpass Magnitude Distributions

(b) Accuracy Dropoff from Thresholding

Figure 1.9: **Bandpass Gain Properties.** (a) shows the distribution of the magnitudes for bandpass coefficients for the second layer (gain2). Each orientation has $128 \times 64 = 8192$ complex weights, most of which are close to or near 0. The 45° and 135° weights have many fewer large coefficients. (b) shows the increase in sparsity and dropoff in classification accuracy when the weights are hard-thresholded with value t (same threshold applied to all 3 layers). For a threshold value of $t = 0.4$, 80% of the weights in gain1 are 0, 99% of the weights in gain2 are 0, 99.98% of the weights in gain3 are 0 and classification accuracy is 0.5% lower than the non-thresholded accuracy.

they can be arbitrarily scaled and shifted by using a batch normalization layer. Then the magnitude could shift to (invalid) negative values, which can then be rectified by the ReLU.

$$\theta_j = \arctan\left(\frac{\mathrm{Im}\,(v_j)}{\mathrm{Re}\,(v_j)}\right) \tag{1.5.3}$$

$$r_j = \sqrt{\mathrm{Re}\,(v_j)^2 + \mathrm{Im}\,(v_j)^2} \tag{1.5.4}$$

$$r'_j = \max(0,\ BN(r_j)) = \max(0,\ \gamma\frac{r_j - \mu_r}{\sigma_r} + \beta) \tag{1.5.5}$$

$$u_j = r'_j e^{j\theta_j} \tag{1.5.6}$$

This also works nicely on the lowpass coefficients:

$$u_{lp} = \max(0,\ BN(v_{lp})) \tag{1.5.7}$$

1.5.2 Thresholding

The soft and hard thresholding operators for a threshold $t > 0$ are defined as:

$$\mathcal{H}(x,t) = \mathbb{I}(|x| > t)x \tag{1.5.8}$$

$$\mathcal{S}(x,t) = \mathrm{sgn}(x)\max(0,\ |x| - t) \tag{1.5.9}$$

$$= \frac{x}{|x|}\max(0,\ |x| - t) \tag{1.5.10}$$

Note that (1.5.10) is very similar to (1.5.5) as $r_j \geq 0$. We can rewrite (1.5.5) by taking the strictly positive terms γ, σ outside of the max operator:

$$r'_j = \max(0, \gamma\frac{r_j - \mu}{\sigma} + \beta) \tag{1.5.11}$$

$$= \frac{\gamma}{\sigma_r}\max\left(0, r_j - \mu_r + \frac{\sigma_r\beta}{\gamma}\right) \tag{1.5.12}$$

then if $t' = \mu_r - \frac{\sigma_r\beta}{\gamma} > 0$, doing batch normalization followed by a ReLU on the magnitude of the coefficients is the same as soft shrinkage with threshold t', scaled by a factor $\frac{\gamma}{\sigma_r}$. The same analogy does not apply to the lowpass coefficients, as v_{lp} is not strictly positive.

To get strict soft or hard thresholding for the low and bandpass coefficients, we have the option of learning the threshold t or setting it to achieve a desired sparsity level. In early experiments we found that any learnt threshold t quickly goes to 0, turning off the thresholding. We can force some sparsity level however by keeping a track of the moments of the activations with an exponential moving average, and finding quantiles by matching the distribution of the activations. For the bandpass coefficients, the activations very nearly follow an exponential distribution. The lowpass coefficients are a bit more varied, following a

Algorithm 1.1 The *wave layer* pseudocode

```
procedure WaveLayer($x$)
    $u_{lp},\ u_1 \leftarrow \mathrm{DT\mathbb{C}WT}(x, \mathrm{nlevels} = 1)$
    $v_{lp},\ v_1 \leftarrow G(u_{lp},\ u_1)$                ▷ the normal wave gain layer
    $w_{lp} \leftarrow \sigma_{lp}(v_{lp})$                  ▷ lowpass nonlinearity
    $w_1 \leftarrow \sigma_{bp}(v_1)$                        ▷ bandpass nonlinearity
    $y \leftarrow \mathrm{DT\mathbb{C}WT}^{-1}(w_{lp},\ w_1)$
    $x \leftarrow \sigma_{pixel}(y)$                         ▷ pixel nonlinearity
    return $x$
end procedure
```

generalized Gaussian distribution (GGD) with shape parameter β ranging from 0.6 for the earlier layers to 1.5 for deepr layers.

1.5.3 Non-Linearity Experiments

Taking the same 'gain1_2_3' architecture used for CIFAR-100, we expand the *wave gain layer* into one bigger layer dubbed the *wave layer*, described in Algorithm 1.1. In the wave layer, there are we have 3 different nonliearities: the pixel, the lowpass and the bandpass nonlinearity.

For these experiments, we test over a grid of possible options for these three functions:

Nonlinearity	Values				
Pixel	None	BN+ReLU			
Lowpass	None	ReLU	BN+ReLU	$\mathcal{S}$	$\mathcal{H}$
Bandpass	None	ReLU	BN+MagReLU	$\mathcal{S}$	$\mathcal{H}$

Where:

- 'None' means no nonlinearity – $\sigma(x) = x$.
- 'BN+ReLU' is batch normalization and ReLU (applies only to real valued activations) e.g. (1.5.7).
- 'ReLU' is a ReLU without batch normalization. Can be applied to the real and imaginary parts of a complex activation independently i.e. (1.5.2).
- 'BN+MagReLU' applies batch normalization to the magnitude of complex coefficients and then makes them strictly positive with a ReLU. See (1.5.5).
- $\mathcal{S}$ and $\mathcal{H}$ are the soft and hard thresholding operators applied to the magnitudes of coefficients. We choose a conservative sparsity level of 0.2 (20% of coefficients set to 0) for these thresholds. A full grid search over sparsity levels would be beneficial, but setting it low initially allows us to test its plausibility as a nonlinearity.

As the pixel nonliearity has only two options, the results are best displayed as a pair of tables, firstly for no nonlinearity and secondly for the the standard batch normalization and ReLU. See Table 1.2 for these two tables.

Digesting this information gives us some useful insights:

1. It is possible to improve on the gainlayer from the previous experiments (top left entry of the second table) with the right nonlinearities.

2. Looking at the fourth and fifth rows/columns of both tables, soft and hard thresholding for both the lowpass and bandpass coefficients does not perform as well as ReLUs (and sometimes worse than no nonlinearity) even with a low sparsity level. We saw this performance worsen as the sparsity level increased.

3. Doing a ReLU on the real and imaginary parts of the bandpass coefficients independently (the second row of both tables) almost always performs worse than having no nonlinearity (first row of both tables).

4. The best combination is to have batch normalization and a ReLU applied to the magnitudes of the bandpass coefficients and batch norm and a ReLU applied to either the lowpass or pixel coefficients.

The best accuracy score of 65.5% is now 0.4% lower than the fully convolutional architecture, however this network has slightly fewer parameters.

1.6 Conclusion

In this chapter we have presented the novel idea of learning filters by taking activations into the wavelet domain. In the wavelet domain then we can apply the proposed gain layer G instead of a pixel wise convolution, and can apply wavelet based nonlinearities σ_w. We have considered the possible challenges this proposes and described how a multirate system can learn through backpropagation. Our experiments have been promising but not convincing. We have shown that our layer can learn in an end-to-end system, achieving similar accuracies on CIFAR-10, CIFAR-100 and Tiny ImageNet to the same system with convolutional layers instead. This is a good start and shows the plausibility of the wavelet gain layer, but we have not yet seen a benefit to learning in the wavelet space.

We have searched for good candidates for wavelet nonlinearities, and saw that using Batch Normalization followed by a ReLU on the lowpass coefficients, and Batch Normalization and a ReLU on the magnitudes of the bandpass coefficients improved the performance of the gain layer considerably.

Table 1.2: **Different Nonlinearities in the Gain Layer.** Top-1 Accuracies for 'gain1_2_3' network trained on CIFAR-100 using different wavelet and pixel nonlinearities. The rows of the table correspond to different bandpass nonlinearities and the columns correspond to different lowpass nonlinearities. $\sigma_{pixel} = \sigma_{lp} = \sigma_{bp} = \text{None}$ is a linear system (with max pooling). $\sigma_{pixel} = \text{ReLU}$, $\sigma_{lp} = \sigma_{bp} = \text{None}$ is the system used in earlier experiments which is linear in the wavelet domain and has a nonlinearity in the pixel domain. Results are averages over 3 runs.

(a) $\sigma_{pixel} = \text{None}$

σ_{lp} σ_{bp}	None	ReLU	BN+ReLU	$\mathcal{S}$	$\mathcal{H}$
None	45.0	63.1	64.4	54.1	42.5
ReLU	42.6	62.4	63.9	54.4	41.8
BN+MagReLU	48.5	65.0	65.1	X	X
$\mathcal{S}$	X	X	X	X	X
$\mathcal{H}$	41.3	60.7	X	52.0	40.6

(b) $\sigma_{pixel} = \text{BN} + \text{ReLU}$

σ_{lp} σ_{bp}	None	ReLU	BN+ReLU	$\mathcal{S}$	$\mathcal{H}$
None	62.8	62.0	65.1	62.8	62.9
ReLU	62.8	61.4	64.6	61.6	61.8
BN+MagReLU	64.5	63.1	64.5		
$\mathcal{S}$					
$\mathcal{H}$	59.7	58.7		59.6	59.7

References

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

[2] ——, “Wavelet Convolutional Neural Networks”, *arXiv:1805.08620 [cs]*, May 2018. arXiv: 1805.08620 [cs].

[3] 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.

[4] 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].

[5] 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.

[6] X. Glorot and Y. 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.

[7] 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.

[8] 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].

[9] 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.

[10] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: 1412.6980 `[cs]`.

Appendix A

Invertible Transforms and Optimization

To see this, let us consider the work from [9] 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 $X = \mathrm{DFT}\{x\}$. In matrix form the 2-D DFT is then:

$$\begin{aligned} X &= \mathrm{DFT}\{x\} = UxU && \text{(A.0.1)} \\ x &= \mathrm{DFT}^{-1}\{X\} = U^{*}YU^{*} && \text{(A.0.2)} \end{aligned}$$

When it comes to gradients, these become:

$$\begin{aligned} \frac{\partial L}{\partial X} &= U\frac{\partial L}{\partial x}U = \mathrm{DFT}\left\{\frac{\partial L}{\partial x}\right\} && \text{(A.0.3)} \\ \frac{\partial L}{\partial x} &= U^{*}\frac{\partial L}{\partial X}U^{*} = \mathrm{DFT}^{-1}\left\{\frac{\partial L}{\partial X}\right\} && \text{(A.0.4)} \end{aligned}$$

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

$$\hat{\mathbf{w}}_1 = \mathrm{DFT}\{\mathbf{w}_1\} \tag{A.0.5}$$

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:

$$\begin{aligned}
\mathbf{w}_2 &= \mathbf{w}_1 - \eta\left(\frac{\partial L}{\partial \mathbf{w}} + \epsilon \mathbf{w}_1\right) && \text{(A.0.6)}\\
&= (1-\eta\epsilon)\mathbf{w}_1 - \eta\frac{\partial L}{\partial \mathbf{w}} && \text{(A.0.7)}\\
\hat{\mathbf{w}}_2 &= \hat{\mathbf{w}}_1 - \eta\left(\frac{\partial L}{\partial \hat{\mathbf{w}}} + \epsilon \hat{\mathbf{w}}_1\right) && \text{(A.0.8)}\\
&= (1-\eta\epsilon)\hat{\mathbf{w}}_1 - \eta\frac{\partial L}{\partial \hat{\mathbf{w}}} && \text{(A.0.9)}\\
& && \text{(A.0.10)}
\end{aligned}$$

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 $\mathrm{DFT}^{-1}\{\hat{\mathbf{w}}_2\}$. Using equations A.0.3 and A.0.5 we then get:

$$\begin{aligned}
\mathrm{DFT}^{-1}\{\hat{\mathbf{w}}_2\} &= \mathrm{DFT}^{-1}\left\{(1-\eta\epsilon)\hat{\mathbf{w}}_1 - \eta\,\frac{\partial L}{\partial \hat{\mathbf{w}}}\right\} && \text{(A.0.11)}\\
&= (1-\eta\epsilon)\mathbf{w}_1 - \eta\,\mathrm{DFT}^{-1}\left\{\frac{\partial L}{\partial \hat{\mathbf{w}}}\right\} && \text{(A.0.12)}\\
&= (1-\eta\epsilon)\mathbf{w}_1 - \eta\frac{\partial L}{\partial \mathbf{w}} && \text{(A.0.13)}\\
&= \mathbf{w}_2 && \text{(A.0.14)}
\end{aligned}$$

This does not hold for the Adam [10] 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 [9].

Appendix B

DTℂWT **Single Subband Gains**

Let us consider one subband of the DTℂWT. This includes the coefficients from both tree A and tree B. For simplicity in this analysis we will consider the 1-D DTℂWT 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 B.1. The end to end transfer function is:

$$\frac{Y(z)}{X(z)} = \frac{1}{M} \sum_{k=0}^{M-1} \Big[A(W^k z)C(z) + B(W^k z)D(z) \Big] \tag{B.0.1}$$

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

Theorem B.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* (B.0.1) *are nearly zero and the system is nearly shift invariant.*

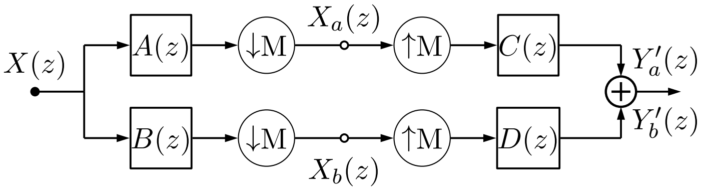


Figure B.1: **Block Diagram of 1-D** DTℂWT**.** Note the top and bottom paths are through the wavelet or scaling functions from just level m ($M = 2^m$). Figure based on Figure 4 in [5].

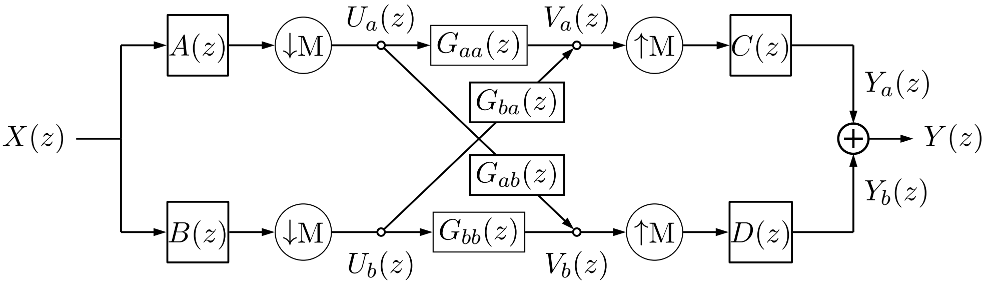


Figure B.2: **Block Diagram of 1-D** DTℂWT**.** Note the top and bottom paths are through the wavelet or scaling functions from just level m ($M = 2^m$). Figure based on Figure 4 in [5].

Proof. See section 4 of [5] 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 (B.0.1) gives:

$$A(W^kz)C(z) + B(W^kz)D(z) = 2P(W^kz)Q(z) + 2P^*(W^kz)Q^*(z) \tag{B.0.2}$$

Using (B.0.2), Kingsbury shows that it is easier to design single side band filters so $P(W^kz)$ does not overlap with $Q(z)$ and $P^*(W^kz)$ does not overlap with $Q^*(z)$ for $k \neq 0$. □

Using Theorem B.1 (B.0.1) reduces to:

$$\frac{Y(z)}{X(z)} = \frac{1}{M}\left[P(z)Q(z) + P^*(z)Q^*(z)\right] \tag{B.0.3}$$

$$= \frac{1}{M}\left[A(z)C(z) + B(z)D(z)\right] \tag{B.0.4}$$

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 DTℂWT.

Figure B.2 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 G_{ij} for $i,j \in \{a,b\}$. Then V_a and V_b are:

$$\begin{aligned}
V_a(z) &= U_a(z)G_{aa}(z) + U_b(z)G_{ba}(z) && \text{(B.0.5)}\\
&= \frac{1}{M}\sum_k X(W^k z^{1/k})\left[A(W^k z^{1/k})G_{aa}(z) + B(W^k z^{1/k})G_{ba}(z)\right] && \text{(B.0.6)}\\
V_b(z) &= U_a(z)G_{ab}(z) + U_b(z)G_{bb}(z) && \text{(B.0.7)}\\
&= \frac{1}{M}\sum_k X(W^k z^{1/k})\left[A(W^k z^{1/k})G_{ab}(z) + B(W^k z^{1/k})G_{bb}(z)\right] && \text{(B.0.8)}
\end{aligned}$$

Further, Y_a and Y_b are:

$$\begin{aligned}
Y_a(z) &= C(z)V_a(z^M) && \text{(B.0.9)}\\
Y_b(z) &= D(z)V_b(z^M) && \text{(B.0.10)}
\end{aligned}$$

Then the end to end transfer function is:

$$\begin{aligned}
Y(z) = Y_a(z) + Y_b(z) = \frac{1}{M}\sum_{k=0}^{M-1} X(W^k z)\Big[&A(W^k z)C(z)G_{aa}(z^k) + B(W^k z)D(z)G_{bb}(z) + \\
&B(W^k z)C(z)G_{ba}(z^k) + A(W^k z)D(z)G_{ba}(z)\Big]
\end{aligned} \quad \text{(B.0.11)}$$

Theorem B.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 (B.0.11) become:

$$G_r(z^k)\left[A(W^k z)C(z) + B(W^k z)D(z)\right] + G_i(z^k)\left[A(W^k z)D(z) - B(W^k z)C(z)\right] \quad \text{(B.0.12)}$$

Theorem B.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 B.1 we note that:

$$\begin{aligned}
A(W^k z)D(z) - B(W^k z)C(z) &= j\left[P(W^k z) + P^*(W^k z)\right][Q(z) - Q^*(z)] + && \text{(B.0.13)}\\
& j\left[P(W^k z) - P^*(W^k z)\right][Q(z) + Q^*(z)] && \text{(B.0.14)}\\
&= 2j\left[P(W^k z)Q(z) - P^*(W^k z)Q^*(z)\right] && \text{(B.0.15)}
\end{aligned}$$

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 B.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 (B.0.12). This completes our proof. □

Using Theorem B.2, the end to end transfer function with the gains is now

$$\begin{aligned} \frac{Y(z)}{X(z)} &= \frac{2}{M}\left[G_r(z^M)\left(A(z)C(z)+B(z)D(z)\right)+G_i(z^M)\left(A(z)D(z)-B(z)C(z)\right)\right] && \text{(B.0.16)} \\ &= \frac{2}{M}\left[G_r(z^M)\left(PQ+P^*Q^*\right)+jG_i(z^M)\left(PQ-P^*Q^*\right)\right] && \text{(B.0.17)} \end{aligned}$$

Now we know can assume that our DTℂWT 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.

Appendix C

Complex Convolution and Gradients

Consider a complex number $z = x + iy$, and the complex mapping

$$w = f(z) = u(x,y) + iv(x,y) \tag{C.0.1}$$

where u and v are called ‘conjugate functions’. Let us examine the properties of $f(z)$ and its gradient.

The definition of gradient for complex numbers is:

$$\lim_{\Delta z \to 0} \frac{f(z+\Delta z) - f(z)}{\Delta z} \tag{C.0.2}$$

A necessary condition for $f(z,\bar{z})$ to be an analytic function is $\frac{\partial f}{\partial \bar{z}} = 0$. I.e. f must be purely a function of z, and not $\bar{z}$.

A geometric interpretation of complex gradient is shown in Figure C.1. As Δz shrinks to 0, what does Δw converge to? E.g. consider the gradient of approach $m = \frac{dy}{dx} = \tan\theta$, then the derivative is

$$\gamma = \alpha + i\beta = D(x,y) + P(x,y)e^{-2i\theta} \tag{C.0.3}$$

where

$$\begin{aligned} D(x,y) &= \frac{1}{2}(u_x + v_y + i(v_x - u_y)) && \text{(C.0.4)} \\ P(x,y) &= \frac{1}{2}(u_x - v_y + i(v_x + u_y)) && \text{(C.0.5)} \end{aligned}$$

$P(x,y) = \frac{dw}{d\bar{z}}$ needs to be 0 for the function to be analytic. This is where we get the Cauchy-Riemann equations:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{C.0.6}$$

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \tag{C.0.7}$$

The function $f(z)$ is analytic (or regular or holomorphic) if the derivative $f'(z)$ exists at all points z in a region R. If R is the entire z-plane, then f is entire.

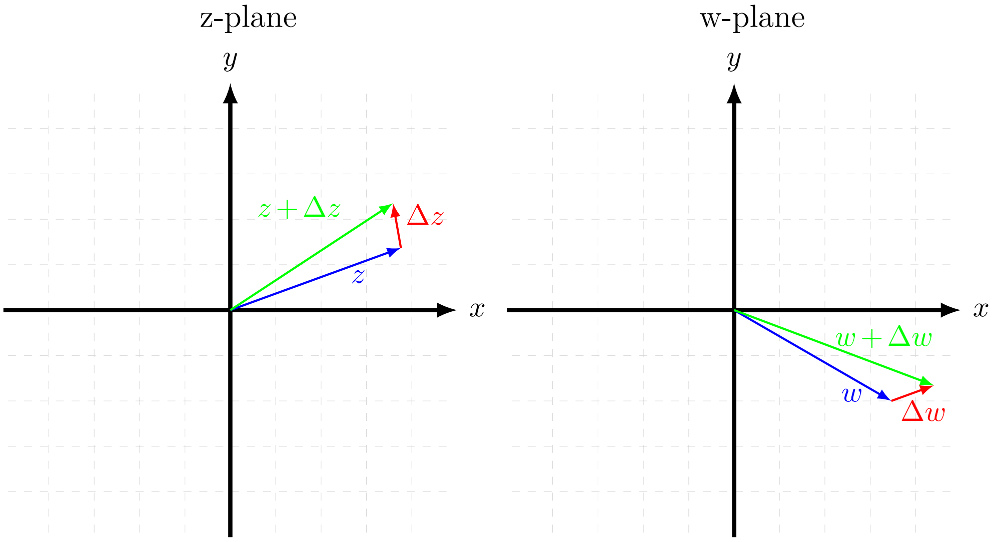


Figure C.1: **Geometric interpretation of complex gradient.** The gradient is defined as $f'(z) = \lim_{\Delta z \to 0} \frac{\Delta w}{\Delta z}$. It must approach the same value independent of the direction Δz approaches zero. This turns out to be a very strong and somewhat restrictive property.

C.1 Grad Operator

Recall, the gradient is a multi-variable generalization of the derivative. The gradient is a vector valued function. In the case of complex numbers, it can be represented as a complex number too. E.g. consider $W(z) = F(x,y)$ (note that in general it may be simple to find F given G, but they are different functions).

I.e.

$$\nabla F = \frac{\partial F}{\partial x} + i\frac{\partial F}{\partial y}$$

Consider the case when F is purely real, then $F(x,y) = F(\frac{z+\bar{z}}{2}, \frac{z-\bar{z}}{2i}) = G(z,\bar{z})$ Then

$$\nabla F = \frac{\partial F}{\partial x} + i\frac{\partial F}{\partial y} = 2\frac{\partial G}{\partial \bar{z}}$$

If F is complex, let $F(x,y) = P(x,y) + iQ(x,y) = G(z,\bar{z})$, then

$$\nabla F = \left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}\right)(P+iQ) = \left(\frac{\partial P}{\partial x} - \frac{\partial Q}{\partial x}\right) + i\left(\frac{\partial P}{\partial y} + \frac{\partial Q}{\partial x}\right) = 2\frac{\partial G}{\partial \bar{z}}$$

It is clear to see how the purely real case is a subset of this (set Q=0 and all its partials will be 0 too).

If G is an analytic function, then $\frac{\partial G}{\partial \bar{z}} = 0$ and so the gradient is 0, and the Cauchy-Riemann equations hold $\frac{\partial P}{\partial x} = \frac{\partial Q}{\partial y}$ and $\frac{\partial P}{\partial y} = -\frac{\partial Q}{\partial x}$

C.2 Forward pass

C.2.1 Convolution

In the above example **f** has a spatial support of only 1×1. We still were able to get a somewhat complex shape by shifting the relative phases of the complex coefficients, but we are inherently limited (as we can only rotate the coefficient by at most 2π). So in general, we want to be able to consider the family of filters $\mathbf{f} \in \mathbb{C}^{m_1 \times m_2 \times C}$. For ease, let us consider only square filters of spatial support m, so $\mathbf{f} \in \mathbb{C}^{m \times m \times C}$. Note that we have restricted the third dimension of our filter to be $C = 12$ in this case. This means that convolution is only in the spatial domain, rather than across channels. Ultimately we would like to be able to handle the more general case of allowing the filter to rotate through channels, but we will tackle the simpler problem first[1]

Let us represent the complex input with z, which is of shape $\mathbb{C}^{n_1 \times n_2 \times C}$. We call w the result we get from convolving **z** with **f**, so $\mathbf{w} \in \mathbb{C}^{n_1+m-1, n_2+m-1, 1}$. With appropriate zero or symmetric padding, we can make w have the same spatial shape as **z**. Now, consider the full complex convolution to get w:

$$w[l_1,l_2] = \sum_{c=0}^{C-1} \sum_{k_1,k_2} f[k_1,k_2,c] z[l_1-k_1, l_2-k_2, c] \tag{C.2.1}$$

Let us define

$$z = z_R + jz_I \tag{C.2.2}$$

$$w = w_R + jw_I \tag{C.2.3}$$

$$f = f_R + jf_I \tag{C.2.4}$$

[1] Recall from **??**, the benefit of allowing a filter to rotate through the channel dimension was we could easily obtain 30° shifts of the sensitive shape.

where all of these belong to the real space of the same dimension as their parent. Then

$$\begin{aligned}
w[l_1,l_2] =& w_R + jw_I \\
=& \sum_{c=0}^{C-1}\sum_{k_1,k_2} f[k_1,k_2,c]z[l_1-k_1,l_2-k_2,c] \\
=& \sum_{c=0}^{C-1}\sum_{k_1,k_2} (f_R[k_1,k_2,c]+jf_I[k_1,k_2,c])(z_R[l_1-k_1,l_2-k_2,c]+jz_I[l_1-k_1,l_2-k_2,c]) \\
=& \sum_{c=0}^{C-1}\sum_{k_1,k_2} (z_R[l_1-k_1,l_2-k_2,c]f_R[k_1,k_2,c]-z_I[l_1-k_1,l_2-k_2,c]f_I[k_1,k_2,c]) \\
&+j\sum_{c=0}^{C-1}\sum_{k_1,k_2} (z_R[l_1-k_1,l_2-k_2,c]f_I[k_1,k_2,c]+z_I[l_1-k_1,l_2-k_2,c]f_R[k_1,k_2,c]) \\
=& \left((z_R * f_R)-(z_I * f_I)\right)[l_1,l_2] + \left((z_R * f_I)+(z_I * f_R)\right)[l_1,l_2] \qquad \text{(C.2.5)}
\end{aligned}$$

Unsurprisingly, complex convolution is then the sum and difference of 4 real convolutions.

C.2.2 Regularization

We must be careful with regularizing complex weights. We want to set some of the weights to 0, and let the remaining ones evolve to whatever phase they please. To do this, either use the L-2 norm on the real and imaginary parts independently, or be careful about using the L-1 norm. This is because we really want to be penalising the magnitude of the complex weights, r and:

$$\|r\|_2^2 = \left\|\sqrt{x^2+y^2}\right\|_2^2 = \sum x^2+y^2 = \sum x^2 + \sum y^2 = \|x\|_2^2 + \|y\|_2^2 \qquad \text{(C.2.6)}$$

But for ℓ_1 regularization:

$$\|r\|_2 = \left\|\sqrt{x^2+y^2}\right\|_1 = \sum \sqrt{x^2+y^2} \neq \|x\|_1 + \|y\|_1 \qquad \text{(C.2.7)}$$

C.3 Soft Shrinkage

Let $z = x+jy$ and $w = u+jv = \mathcal{S}(z,t)$ where we define the soft shrinkage on a complex number $z = re^{j\theta}$ by a real threshold t as:

$$\mathcal{S}(z,t) = \begin{cases} 0 & r < t \\ (r-t)e^{j\theta} & r \geq t \end{cases} \qquad \text{(C.3.1)}$$

This can alternatively be written as:

$$\mathcal{S}(z,t) = \frac{\max(r-t,0)}{r} z \tag{C.3.2}$$

$$= gz \tag{C.3.3}$$

To find the pass through gradients $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}$ we need to find all the real and imaginary partial derivatives.

$$\frac{\partial g}{\partial x} = \begin{cases} 0 & r < t \\ \frac{r\frac{\partial r}{\partial x} - (r-t)\frac{\partial r}{\partial x}}{r^2} & r \geq t \end{cases} \tag{C.3.4}$$

$$= \frac{xt\mathbb{I}(g>0)}{r^3} \tag{C.3.5}$$

$$\frac{\partial g}{\partial y} = \frac{yt\mathbb{I}(g>0)}{r^3} \tag{C.3.6}$$

$$\frac{\partial g}{\partial t} = \frac{-\mathbb{I}(g>0)}{r} \tag{C.3.7}$$

Then from the definition of $w = u + jv$ we have $u = gx$ and $v = gy$, giving us:

$$\frac{\partial u}{\partial x} = g + \frac{x^2t\mathbb{I}(g>0)}{r^3} \tag{C.3.8}$$

$$\frac{\partial u}{\partial y} = \frac{xyt\mathbb{I}(g>0)}{r^3} \tag{C.3.9}$$

$$\frac{\partial u}{\partial t} = \frac{-x\mathbb{I}(g>0)}{r} \tag{C.3.10}$$

$$\frac{\partial v}{\partial x} = \frac{xyt\mathbb{I}(g>0)}{r^3} \tag{C.3.11}$$

$$\frac{\partial v}{\partial y} = g + \frac{y^2t\mathbb{I}(g>0)}{r^3} \tag{C.3.12}$$

$$\frac{\partial v}{\partial t} = \frac{-y\mathbb{I}(g>0)}{r} \tag{C.3.13}$$

And putting it all together we have:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial L}{\partial v}\frac{\partial v}{\partial x} \tag{C.3.14}$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial L}{\partial v}\frac{\partial v}{\partial y} \tag{C.3.15}$$

$$\frac{\partial L}{\partial t} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial L}{\partial v}\frac{\partial v}{\partial t} \tag{C.3.16}$$

These equations are for point-wise application of soft-thresholding. When applied to an entire image, then we sum $\frac{\partial L}{\partial t}$ over all locations.

Appendix D

GainLayer Additional Results

D.0.0.1 Small Kernel Ablation

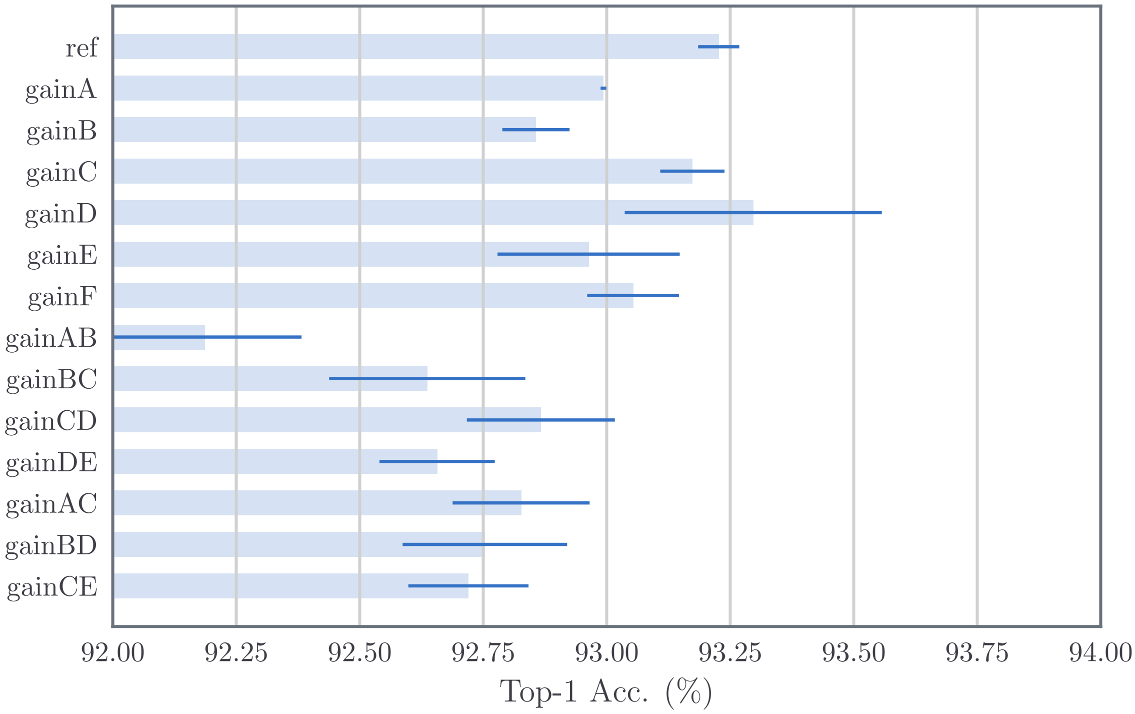
For consistency, we use the same reference network as the one introduced in the previous chapter in **??**. Again, we run this on CIFAR-10, CIFAR-100 and Tiny ImageNet.

We use the same naming technique as in the previous chapter, 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.

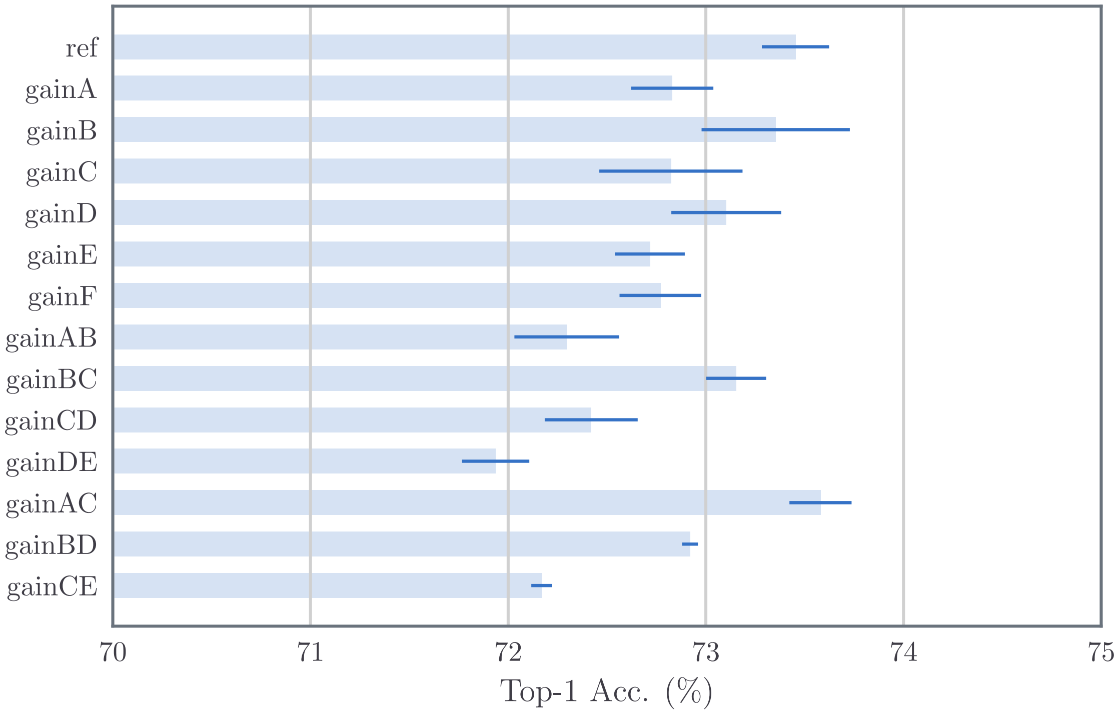
As we are only exploring the wavelet gain layer, and not any wavelet nonlinearities, we come back into the pixel domain to apply the ReLU after learning. This equates to the path shown in Figure 1.2b. I.eẇe are taking wavelet transforms of inputs, applying the gain layer and taking inverse wavelet transforms to do a ReLU in the pixel space. In cases like ‘gainA, gainB’ from Figure D.2, we go in and out of the wavelet layer twice.

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

?? lists the results from these experiments. These show a promising start to this work. 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.

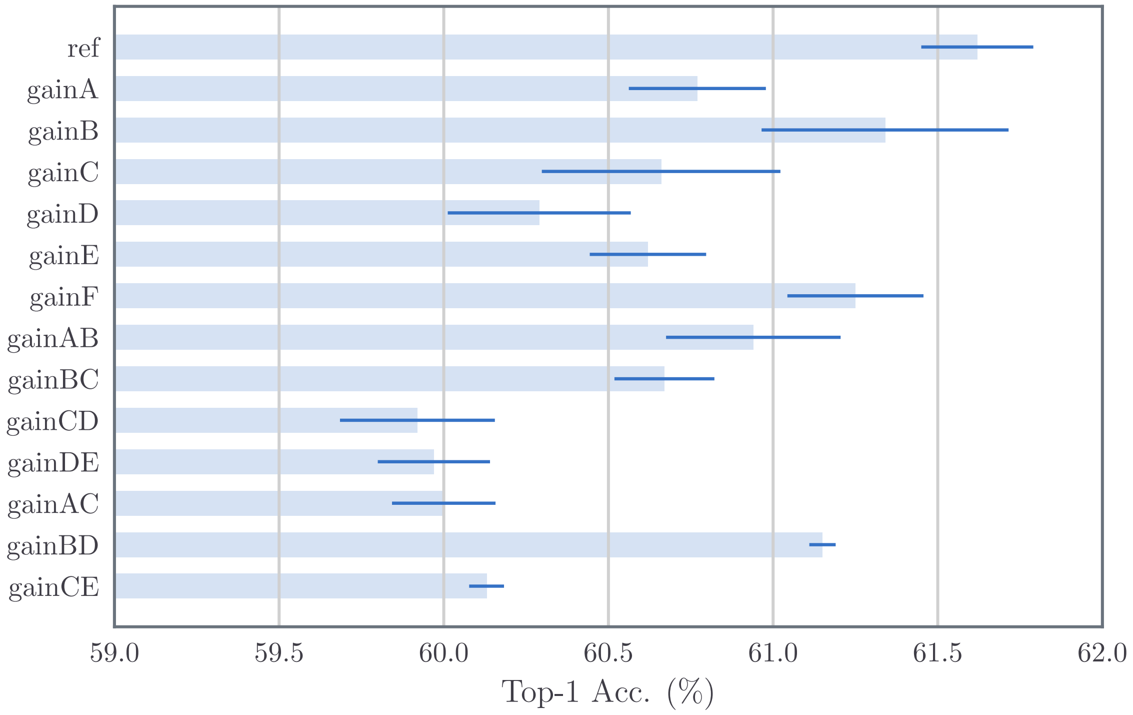


(a) CIFAR-10



(b) CIFAR-100

Figure D.1: **Small kernel ablation results CIFAR.**



(a) Tiny ImageNet

Figure D.2: **Small kernel ablation results Tiny ImageNet.**