

# 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  $\ell_1$  regularization to impose sparsity. This is presented in ??.
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 ??

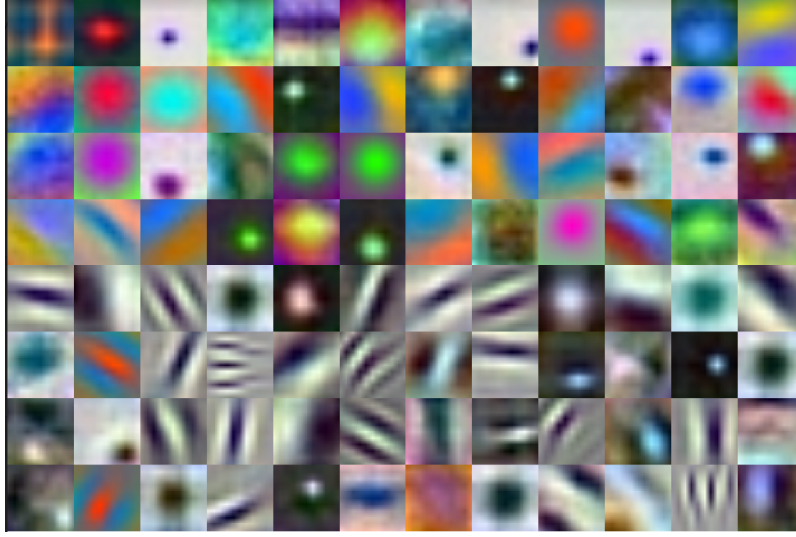


Figure 1.1: **First layer filters of the AlexNet architecture.** The first layer filters of the seminal AlexNet [1] are an inspiration for considering learning filters in the wavelet domain. Each of these  $11 \times 11$  filters would only require a handful of non zero coefficients in the wavelet domain. 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 activations 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 competitive 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

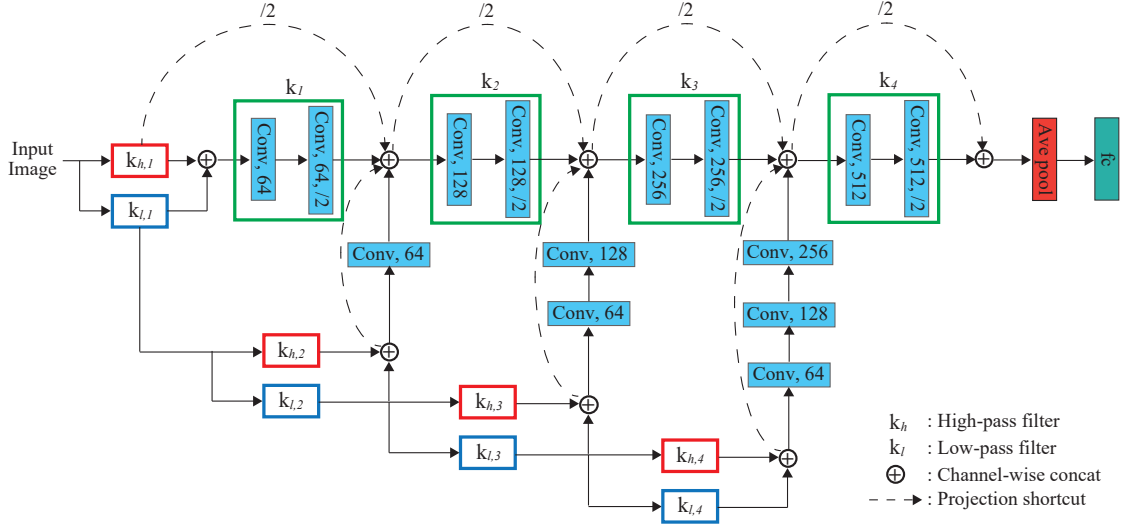


Figure 1.2: **Architecture using the DWT as a frontend to a CNN.** Figure 1 from [4]. Fujieda et. al. take a multiscale wavelet decomposition of the input before passing the input through a standard CNN. They learn convolutional layers independently on each subband and feed these back into the network 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.<sup>1</sup>

### 1.3 Introduction

We would like to explore the possibility of using the wavelet domain as a space to learn. Unlike the work of [7] which only parameterized filters in the wavelet domain and transformed the filters back to the pixel domain to do convolution, this chapter 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 DTCWT [8]. This comes with a

<sup>1</sup>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

memory overhead which we discuss more in [subsection 1.6.1](#), but it enables us to have better directional selectivity and allows for the possibility of returning to the pixel domain with minimal aliasing [\[9\]](#).

We will look at the merits and drawbacks of both options, discuss how to implement them, and finally run some experiments exploring how well these layers work.

### 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. This is proved in [A](#).

## 1.4 Background and Notation

As we now want to consider the DWT and the DT-CWT 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:

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

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

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

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:

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

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

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

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

In shorthand, we can reduce the action of the convolutional layer in [\(1.4.4\)](#) to  $\mathcal{H}$ , saying:

$$y^{(l+1)} = \mathcal{H}x^{(l)} \quad (1.4.6)$$

With the new  $Z$ -transform notation introduced in [\(1.4.3\)](#), we can rewrite [\(1.4.4\)](#) as:

$$Y^{(l+1)}(f, \mathbf{z}) = \sum_{c=0}^{C_l-1} X^{(l)}(c, \mathbf{z}) H_f^{(l)}(c, \mathbf{z}) \quad (1.4.7)$$

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  $W^k$ .

### 1.4.1 DWT Notation

In 2-D, a  $J$  scale DWT gives  $3J+1$  coefficients, 3 sets of bandpass coefficients for each scale, representing the horizontal, vertical and diagonal regions of the frequency plane, and 1 set of lowpass coefficients at the final scale. Let us refer to the DWT wavelet coefficients with the typewriter font  $\mathbf{u}$  and write the  $J$  scale DWT as:

$$\text{DWT}_J(x) = \mathbf{u}_{lp}, \{\mathbf{u}_{j,k}\}_{1 \leq j \leq J, 1 \leq k \leq 3} \quad (1.4.8)$$

If  $x$  is a batch of 2-D images with multiple channels, the DWT is done independently on each channel in each minibatch sample. I.e. if the input is a minibatch of  $N$  samples with  $C$  channels of images of spatial size  $H \times W$ , then  $x \in \mathbb{R}^{N \times C \times H \times W}$  and:

$$\mathbf{u}_{lp} \in \mathbb{R}^{N \times C \times \frac{H}{2^J} \times \frac{W}{2^J}} \quad (1.4.9)$$

$$\mathbf{u}_{j,k} \in \mathbb{R}^{N \times C \times \frac{H}{2^J} \times \frac{W}{2^J}} \quad (1.4.10)$$

Sometimes we may want to refer to all of the subband coefficients at a single scale with one variable, in which case we will simply drop the  $k$  subscript and call them  $\mathbf{u}_j$ . Additionally if we want to refer to all the coefficients (lowpass and bandpass) we will call them  $\mathbf{u}$ .

### 1.4.2 DTCWT Notation

Unlike the DWT, a  $J$  scale DTCWT 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.

Although we will only ever use either the DWT or the DTCWT, to avoid confusion we use a regular font  $u$  to refer to the DTCWT coefficients of  $x$ :

$$\text{DTCWT}_J(x) = u_{lp}, \{u_{j,k}\}_{1 \leq j \leq J, 1 \leq k \leq 6} \quad (1.4.11)$$

Each of these coefficients then has size:

$$u_{lp} \in \mathbb{R}^{N \times C \times \frac{H}{2^{J-1}} \times \frac{W}{2^{J-1}}} \quad (1.4.12)$$

$$u_{j,k} \in \mathbb{C}^{N \times C \times \frac{H}{2^J} \times \frac{W}{2^J}} \quad (1.4.13)$$

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

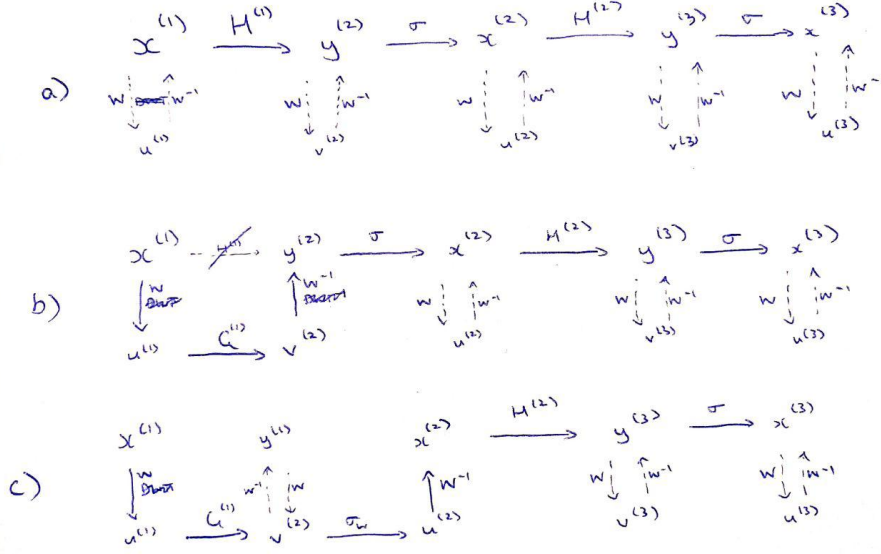


Figure 1.3: **Proposed new forward pass in the wavelet domain.** Two network layers with some possible options for processing. Solid lines denote the evaluation path and dashed lines indicate relationships. In (a) we see a regular convolutional neural network. We have included the dashed lines to make 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  $\mathcal{G}^{(1)}$ , and take the inverse wavelet transform to give  $y^{(2)}$ . The cross through  $\mathcal{H}^{(1)}$  indicates that this path is no longer present. Note that there may not be any possible  $\mathcal{G}^{(1)}$  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  $\sigma_w$  to give  $u^{(2)}$ . We then return to the pixel domain to give  $x^{(2)}$  and continue on from there in the pixel domain.

Again 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.

When we want to be agnostic of the chosen transform type, we use  $\mathcal{W}$  and  $\mathcal{W}^{-1}$  to denote the forward and inverse transform.

## 1.5 The Wavelet Gain Layer

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

From (1.4.4), convolutional layers also have intermediate activations  $y^{(l)}$ . Let us differentiate these from the  $x$  coefficients and modify (1.4.8) and (1.4.11) to say the DWT of  $y^{(l)}$  gives  $v$  and the DTCWT of  $y^{(l)}$  gives  $v$ .

We now propose the wavelet gain layers  $\mathbf{G}$  and  $G$  for the DWT and DTCWT respectively, or  $\mathcal{G}$  for an implementation agnostic layer. The name ‘gain layer’ comes from the inspiration for this chapter’s work, in that the first layer of CNN could be nearly done in the wavelet domain by setting subband gains to 0 and 1.

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

This can be seen as breaking the convolutional path in [Figure 1.3](#) 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  $\mathcal{W}^{-1}$ . Alternatively, we can stay in the wavelet domain and apply a wavelet based nonlinearity  $\sigma_w$  to give the next layer’s  $u$  coefficients. 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:

- How effective  $\mathcal{G}$  is at replacing  $\mathcal{H}$ .
- How effective  $\sigma_w$  is at replcaing  $\sigma$ .

### 1.5.1 The DWT Gain Layer

As mentioned previously, modifying the wavelet coefficients of a critically sampled DWT will necessarily result in a loss of the aliasing cancelling properties. However, in a deep neural network, this is not as obviously a bad thing as it is for denoising or deconvolution. For this reason, we note that there may be a problem in using a DWT, but proceed nonetheless.

For each subband in our  $J$  scale system, we introduce a gain term  $\mathbf{g}$ . Let us specify our input  $\mathbf{u}$  has  $C_l$  channels, and we would like our output  $\mathbf{v}$  to have  $C_{l+1}$  channels, then  $\mathbf{g}$  is made up of:

$$\mathbf{g}_{lp} \in \mathbb{R}^{C_{l+1} \times C_l \times k_{lp} \times k_{lp}} \quad (1.5.1)$$

$$\mathbf{g}_{1,1} \in \mathbb{R}^{C_{l+1} \times C_l \times k_1 \times k_1} \quad (1.5.2)$$

$$\mathbf{g}_{1,2} \in \mathbb{R}^{C_{l+1} \times C_l \times k_1 \times k_1} \quad (1.5.3)$$

$$\vdots \quad (1.5.4)$$

$$\mathbf{g}_{J,3} \in \mathbb{R}^{C_{l+1} \times C_l \times k_J \times k_J} \quad (1.5.5)$$

Note that we have allowed for different kernel spatial sizes for the lowpass and for each bandpass scale  $\mathbf{g}_j$ . This is to allow for the flexibility of putting more emphasis on certain frequency areas if desired.

With these gains, we define  $\mathbf{v} = \mathbf{G}\mathbf{u}$  to be:

$$\mathbf{v}_{lp}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} \mathbf{u}_{lp}[c, \mathbf{n}] * \mathbf{g}_{lp}[f, c, \mathbf{n}] \quad (1.5.6)$$

$$\mathbf{v}_{1,1}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} \mathbf{u}_{1,1}[c, \mathbf{n}] * \mathbf{g}_{1,1}[f, c, \mathbf{n}] \quad (1.5.7)$$

$$\mathbf{v}_{1,2}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} \mathbf{u}_{1,2}[c, \mathbf{n}] * \mathbf{g}_{1,2}[f, c, \mathbf{n}] \quad (1.5.8)$$

$$\vdots \quad (1.5.9)$$

$$\mathbf{v}_{J,3}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} \mathbf{u}_{J,3}[c, \mathbf{n}] * \mathbf{g}_{J,3}[f, c, \mathbf{n}] \quad (1.5.10)$$

I.e., we do independent mixing at each of the different subbands. For  $1 \times 1$  kernels, this is simply a matrix multiply of the wavelet coefficients. This is shown visually in [1.8a](#).

#### 1.5.1.1 The Output

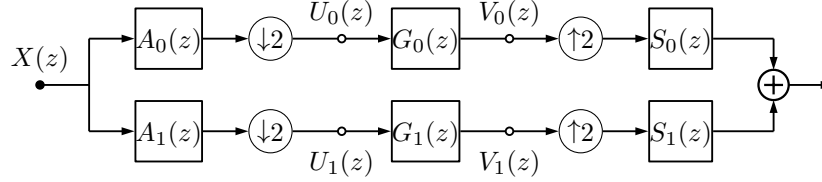


Figure 1.4: **Block Diagram of 1-D DWT Gain Layer.** Here we show the low and highpass for a single scale 1-D DWT, Gain Layer and inverse DWT. The gain layer has gains  $\mathbf{g}_0$  and  $\mathbf{g}_1$ .

To explore the action of the gain layer  $\mathbf{G}$  on DWT coefficients, let us examine a 1-D, single scale, and single channel system. We want to find the action of the layer without any nonlinearities, i.e.,

$$\mathbf{y} = \mathcal{W}^{-1} \mathbf{G} \mathcal{W} \mathbf{x} \quad (1.5.11)$$

or the path taken in [Figure 1.3b](#).

Let us call the low and highpass analysis filters of the DWT  $A_0$  and  $A_1$ , and the synthesis filters  $S_0$  and  $S_1$  (these are normally called  $H$  and  $G$ , but we keep those letters reserved for the CNN and gain layer filters). From the perfect reconstruction property of the DWT, we know

$$A_0(z)S_0(z) + A_1(z)S_1(z) = 2 \quad (1.5.12)$$

and from the aliasing cancellation property of the DWT, we know:

$$A_0(-z)S_0(z) + A_1(-z)S_1(z) = 0 \quad (1.5.13)$$



If we add in gains  $G_0$  and  $G_1$  to the low and highpass coefficients, as shown in [Figure 1.4](#), then the output is:

$$Y(z) = \frac{1}{2}X(z) [A_0(z)S_0(z)G_0(z^2) + A_1(z)S_1(z)B(z^2)] + \frac{1}{2}X(-z) [A_0(-z)S_0(z)G_0(z^2) + A_1(-z)S_1(z)G_1(z^2)] \quad (1.5.14)$$

If we let  $D(z) = G_1(z^2) - G_0(z^2)$  then the above becomes:

$$Y(z) = \frac{1}{2}X(z) [A_0(z)S_0(z)G_0(z^2) + A_1(z)S_1(z)G_1(z^2)] + \frac{1}{2}X(-z)D(z)A_1(-z)S_1(z) \quad (1.5.15)$$

### 1.5.1.2 Backpropagation

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

$$\Delta X(z) = H(z^{-1})\Delta Y(z) \quad (1.5.16)$$

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.

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 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. The weight update gradients are then calculated by finding  $\Delta V_i(z) = S_i(z^{-1})Y(z)$  for  $i = 0, 1$ , and then convolving with the time reverse of the saved wavelet coefficients from the forward pass -  $U_i(z^{-1})$ .

$$\Delta G_i(z) = \Delta V_i(z)U_i(z^{-1}) \quad (1.5.17)$$

Unsurprisingly, the passthrough gradients have similar form to [\(1.5.15\)](#)

$$\Delta X(z) = \frac{1}{2}\Delta Y(z) [A_0(z)S_0(z)G_0(z^{-2}) + A_1(z)S_1(z)G_1(z^{-2})] + \frac{1}{2}\Delta Y(-z)D(z)A_1(-z)S_1(z) \quad (1.5.18)$$

Note that we only need to evaluate [\(1.5.17\)](#), [\(1.5.18\)](#) over the support of  $G(z)$  i.e., if it is a single number we only need to calculate  $\Delta G(z)|_{z=0}$ .

## 1.5.2 The DTCWT Gain Layer

The DTCWT gain layer is the same in principle to the action of the DWT gain layer, but due to the different properties of the two transforms, the implementation is slightly different. 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 DTCWT. We also reintroduce the channel dimension.

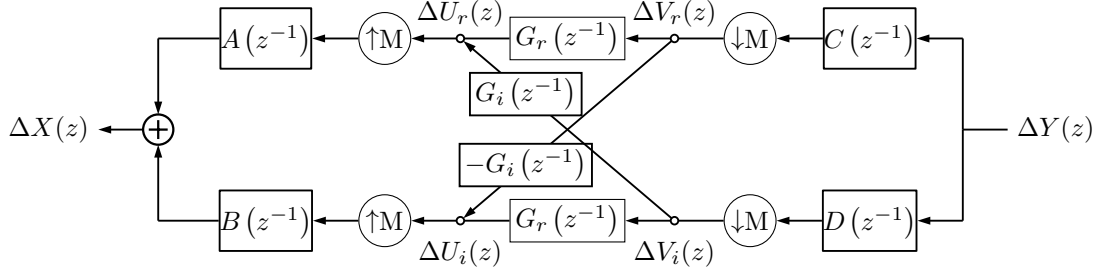


Figure 1.5: **Forward and backward block diagrams for DTCWT gain layer.** Based on Figure 4 in [9]. Ignoring the  $G$  gains, the top and bottom paths (through  $A, C$  and  $B, D$  respectively) make up the the real and imaginary parts for *one subband* of the dual tree system. Combined,  $A + jB$  and  $C - jD$  make the complex filters necessary to have support on one side of the Fourier domain (see Figure 1.7). 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).

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

$$g_{lp} \in \mathbb{R}^{C_{l+1} \times C_l \times k_{lp} \times k_{lp}} \quad (1.5.19)$$

$$g_{1,1} \in \mathbb{C}^{C_{l+1} \times C_l \times k_1 \times k_1} \quad (1.5.20)$$

$$g_{1,2} \in \mathbb{C}^{C_{l+1} \times C_l \times k_1 \times k_1} \quad (1.5.21)$$

$$\vdots \quad (1.5.22)$$

$$g_{J,6} \in \mathbb{C}^{C_{l+1} \times C_l \times k_J \times k_J} \quad (1.5.23)$$

where  $k, k$  are the sizes of the mixing kernels. These could be  $1 \times 1$  for simple gain control, or could be larger, say  $3 \times 3$ , to do more complex filtering on the subbands.

With these gains we define  $v = Gu$  to be:

$$v_{lp}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{lp}[c, \mathbf{n}] * g_{lp}[f, c, \mathbf{n}] \quad (1.5.24)$$

$$v_{1,1}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{1,1}[c, \mathbf{n}] * g_{1,1}[f, c, \mathbf{n}] \quad (1.5.25)$$

$$v_{1,2}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{1,2}[c, \mathbf{n}] * g_{1,2}[f, c, \mathbf{n}] \quad (1.5.26)$$

$$\vdots \quad (1.5.27)$$

$$v_{J,6}[f, \mathbf{n}] = \sum_{c=0}^{C_l-1} u_{J,6}[c, \mathbf{n}] * g_{J,6}[f, c, \mathbf{n}] \quad (1.5.28)$$

Note that for complex signals  $a, b$  the convolution  $a * b$  is defined as  $(a_r * b_r - a_i * b_i) + j(a_r * b_i + a_i * b_r)$ . This is shown in 1.8b.

### 1.5.2.1 The Output

Unlike the DWT gain layer, the DTCWT gain layer can achieve aliasing cancelling and therefore has a transfer function. The proof of this is done in B.

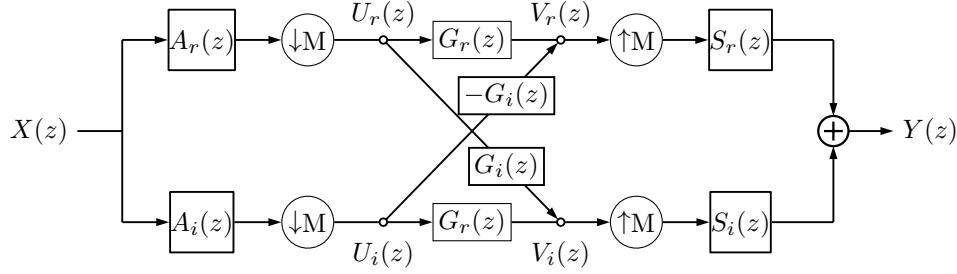


Figure 1.6: **Block Diagram of 1-D DTCWT Gain Layer.** Here we the real and imaginary trees for a single subband. Note that while it may look similar to Figure 1.4, this diagram represents the two trees for one subband rather than a single tree with a pair of subbands. The gain layer does a complex multiply, using both the real and imaginary parts of the decomposed signal. This preserves the shift invariance of the DTCWT for the reconstructed signal  $Y$ .

For a single subband in the DTCWT, the gain layer uses a complex learned weight  $g$ , but otherwise produces the output  $v$  in much the same way as the DWT gain layer. Figure 1.6 shows a single subband DTCWT based gain layer. The output of this layer is:

$$Y(z) = \frac{2}{M} X(z) \left[ G_r(z^M) (A_r(z) S_r(z) + A_i(z) S_i(z)) + G_i(z^M) (A_r(z) S_i(z) - A_i(z) S_r(z)) \right] \quad (1.5.29)$$

See B for the derivation. The  $G_r$  term modifies the subband gain  $A_r S_r + A_i S_i$  and the  $G_i$  term modifies its Hilbert Pair  $A_r S_i - A_i S_r$ . Figure 1.7 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.

### 1.5.2.2 Backpropagation

If  $H$  were complex, the first term in Equation 1.5.16 would be  $\bar{H}(1/\bar{z})$ , but as each individual block in the DTCWT is purely real, we can use the simpler form  $H(z^{-1})$ .

Again, let us calculate  $\Delta V_r(z)$  and  $\Delta V_i(z)$  by backpropagating  $\Delta Y(z)$  through the inverse DTCWT. Again this is the same as doing the forward DTCWT on  $\Delta Y(z)$ . Then the weight update equations are:

$$\Delta G_r(z) = \Delta V_r(z) U_r(z^{-1}) + \Delta V_i(z) U_i(z^{-1}) \quad (1.5.30)$$

$$\Delta G_i(z) = -\Delta V_r(z) U_i(z^{-1}) + \Delta V_i(z) U_r(z^{-1}) \quad (1.5.31)$$

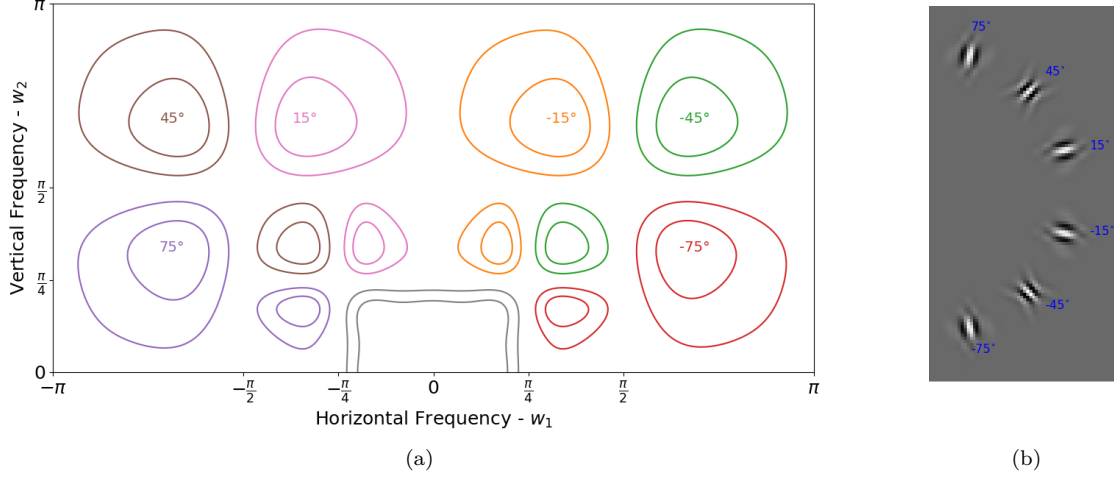


Figure 1.7: DTCWT **subbands**. (a) -1dB and -3dB contour plots showing the support in the Fourier domain of the 6 subbands of the DTCWT at scales 1 and 2, and the scale 2 lowpass. These are the product of the single side band filters  $P(z)$  and  $Q(z)$  from [Theorem B.1](#). (b) The pixel domain impulse responses for the second scale wavelets. The Hilbert pair for each wavelet is the underlying sinusoid phase shifted by 90 degrees.

The passthrough equations have similar form to [\(1.5.29\)](#):

$$\Delta X(z) = \frac{2\Delta Y(z)}{M} \left[ G_r(z^{-M})(A_r(z)S_r(z) + A_i(z)S_i(z)) + jG_i(z^{-M})(A_r(z)S_i(z) - A_i(z)S_r(z)) \right] \quad (1.5.32)$$

### 1.5.3 Examples

[Figure 1.9](#) show example impulse responses of our layer for both the DWT and DTCWT systems. The DWT outputs come from three random variables: a  $1 \times 1$  convolutional weight applied to each of the low-high, high-low and high-high subbands. The DTCWT outputs come from twelve random variables. Again a  $1 \times 1$  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 DTCWT, the distribution of the normalized cross-correlation matches the distribution for random vectors with roughly 11.5 degrees of freedom (c.f. 12 random variables in the layer). This is particularly reassuring for the DTCWT as it is showing that there is still representative power despite the redundancy of the transform.

## 1.6 Implementation Details

Before analyzing its performance, we compare the implementation properties of our two new proposed layers (for the DWT and DTCWT) to a standard convolutional layer.

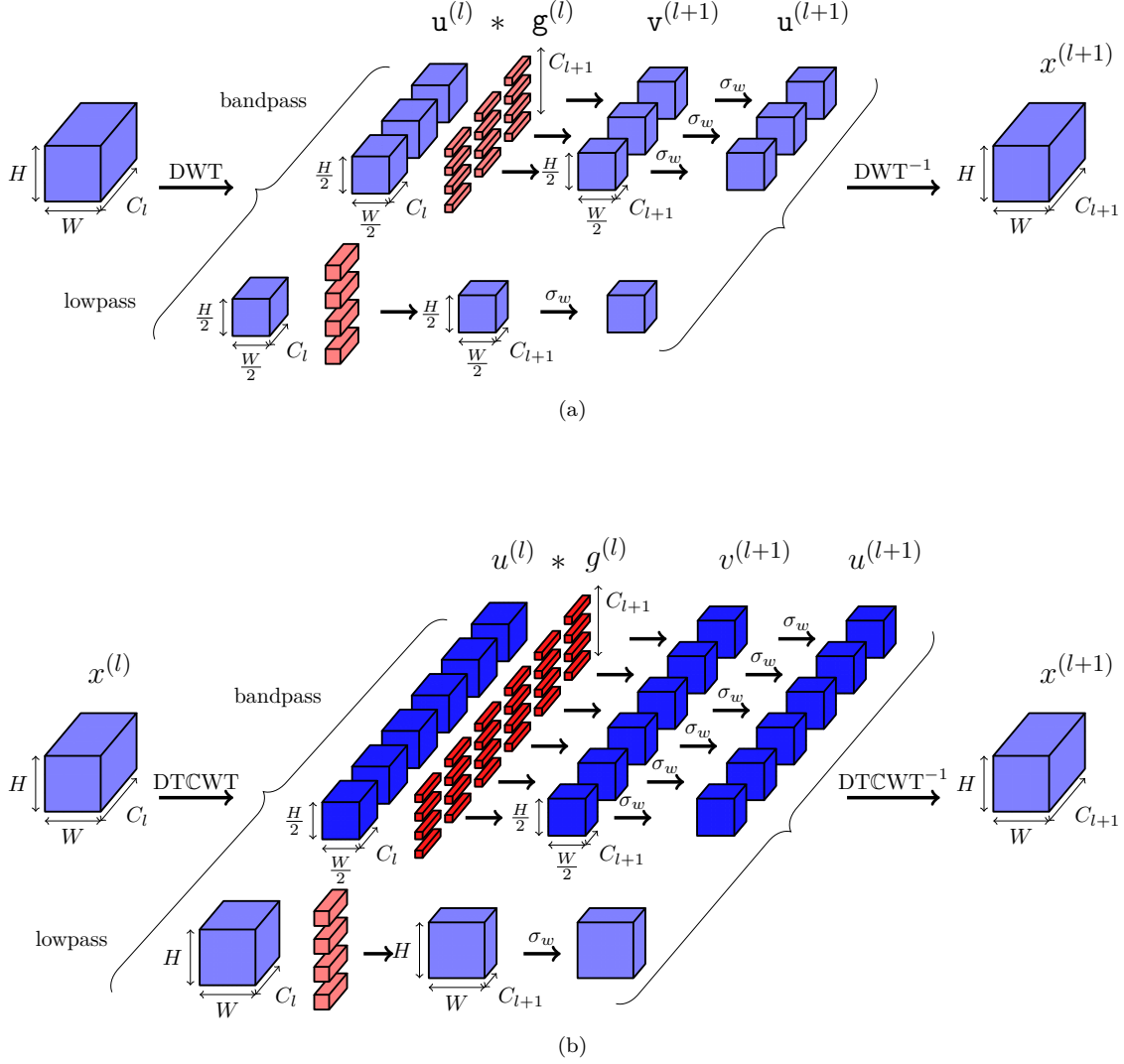
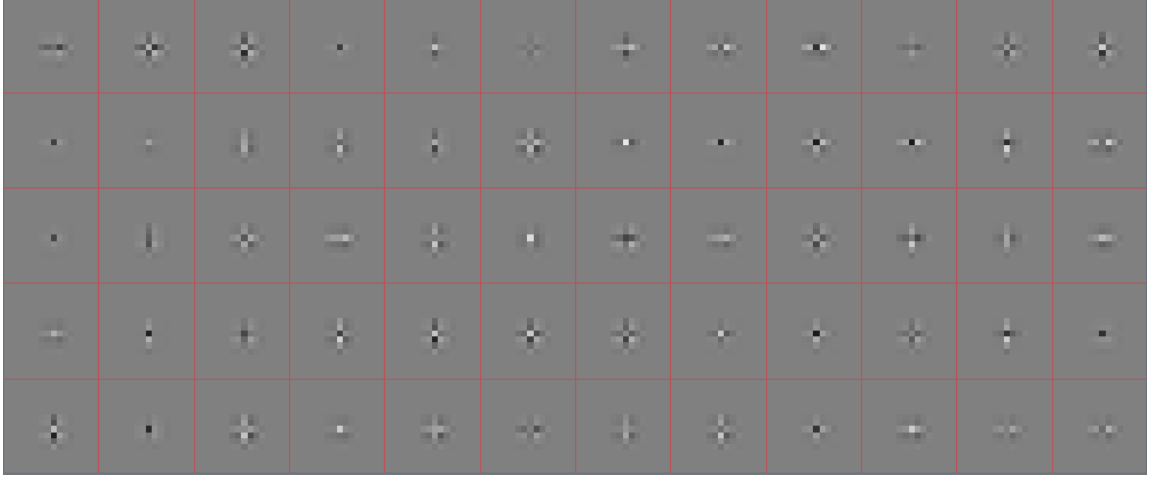
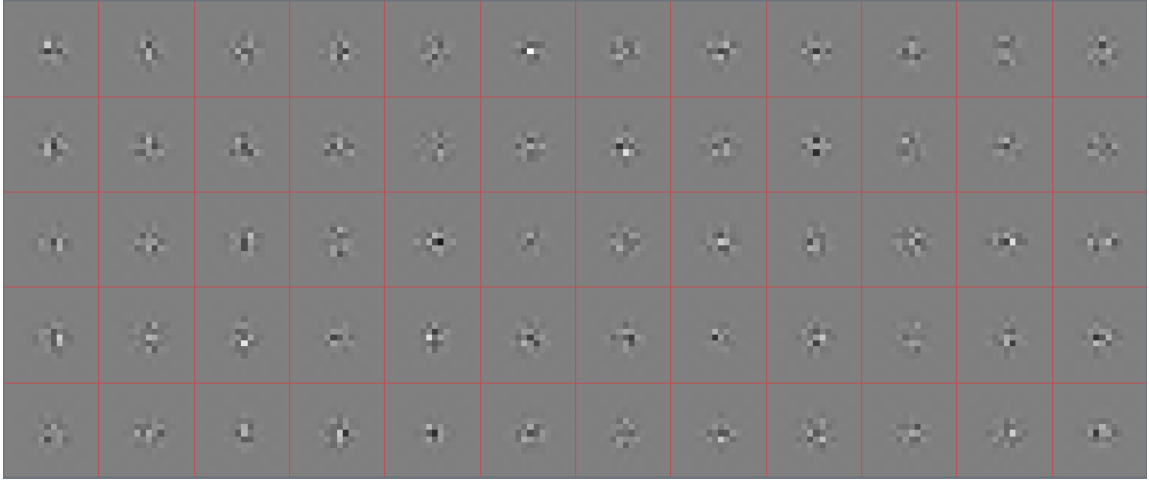


Figure 1.8: **Block diagrams of proposed method to learn in the wavelet domain.** Activations are shaded blue and learned parameters red. Darker shades of blue and red indicate complex valued activations and weights, whereas the lighter shades indicate purely real values. The input  $x^{(l)} \in \mathbb{R}^{C_l \times H \times W}$  is taken into the wavelet domain and each subband is mixed independently with  $C_{l+1}$  sets of convolutional filters (this is what we call the ‘wavelet gain layer’. After mixing, a possible wavelet nonlinearity  $\sigma_w$  is applied to the subbands, before returning to the pixel domain with an inverse wavelet transform. (a) shows the system in the DWT domain. The DWT has fewer coefficients, and the gain layer requires fewer parameters; this is the simpler layer. (b) shows the DTCWT based system. With 6 complex subbands and a larger lowpass, this is the more complex layer.



(a)



(b)

Figure 1.9: **Example outputs from an impulse input for the proposed gain layers.** Example outputs  $y = \mathcal{W}^{-1}\mathcal{G}\mathcal{W}x$  for an impulse  $x$  for both the DWT and DTCWT based systems. (a) shows the outputs  $y$  when  $x$  is an impulse,  $\mathcal{W}$  is the DWT with a ‘db2’ wavelet family and  $\mathcal{G} = \mathbf{G}$  is the DWT gain layer. The gain layer weights  $\mathbf{g}_{lp}$  coefficients are set to 0 and  $\mathbf{g}_1$  coefficients have spatial size  $1 \times 1$  and are sampled independently from a random normal of variance 1. The 60 samples come from 60 different random initializations. (b) shows the output  $y$  for a DTCWT based system. Again,  $g_{lp} = 0$  and  $g_1$  has spatial size  $1 \times 1$ . The 12 values in  $g_1$  are independently sampled from a random normal of variance 1.

### 1.6.1 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.

$$\# \text{conv params} = 9C_l C_{l+1} \quad (1.6.1)$$

We must choose the spatial sizes of both the lowpass and bandpass mixing kernels. In our work, we set:

- The spatial support of the lowpass filters for the DWT gain layer to be  $3 \times 3$  and the support of the bandpass layers to be  $1 \times 1$ .
- The spatial support of the lowpass filters for the DTCWT gain layer to be  $1 \times 1$  and the support of the complex bandpass filters to be  $1 \times 1$ .

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.

This means that for the DWT layer, the number of parameters is:

$$\# \text{DWT params} = (3 + 3^2)C_l C_{l+1} = 12C_l C_{l+1} \quad (1.6.2)$$

And for the DTCWT layer, the number of parameters is:

$$\# \text{DTCWT params} = (2 \times 6 + 1)C_l C_{l+1} = 13C_l C_{l+1} \quad (1.6.3)$$

These are both slightly larger than the  $9C_l C_{l+1}$  parameters used in a standard  $3 \times 3$  convolution, but as [Figure 1.9](#) shows, 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  $1 \times 1$  filters would increase [\(1.6.2\)](#) to  $15C_l C_{l+1}$  and [\(1.6.3\)](#) to  $25C_l C_{l+1}$ .

### 1.6.2 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

$$\# \text{conv floats} = HWC_l \quad (1.6.4)$$

Our layers require us to save the wavelet coefficients  $u_{lp}$  and  $u_{j,k}$  for updating the  $g$  terms as in [\(1.5.17\)](#) and [\(1.5.30\)](#), [\(1.5.31\)](#). For the critically sampled DWT, this requires:

$$\# \text{DWT floats} = HWC_l \quad (1.6.5)$$

to be saved for the backwards pass. For the 4 : 1 redundant DTCWT, this requires:

$$\# \text{DTCWT floats} = 4HWC_l \quad (1.6.6)$$

to be saved for the backwards pass. You can see this difference from the difference in the block diagrams in [Figure 1.8](#).

Note that a single scale DTCWT 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 DTCWT 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 DTCWT gain layer would still only require  $4HWC_l$  floats.

### 1.6.3 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  $C_l \times H \times W$ ).

For the DWT with Daubechies 2 filters, the forward and inverse transform only require about 6 multiplies per input pixel. The mixing is then done at a reduced spatial resolution. For our proposed kernel sizes of  $3 \times 3$  for the lowpass and  $1 \times 1$  for the bandpass, the DWT gain layer requires:

$$\#DWT \text{ mults/pixel} = \underbrace{\frac{3}{4}C_{l+1}}_{\text{bandpass}} + \underbrace{\frac{3^2}{4}C_{l+1}}_{\text{lowpass}} + \underbrace{6}_{DWT} + \underbrace{6}_{DWT^{-1}} = 4C_{l+1} + 12 \quad (1.6.7)$$

This is smaller than a standard  $3 \times 3$  convolutional layer using  $9C_{l+1}$  multiplies per pixel.

For the DTCWT, 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:

$$\#DTCWT \text{ mults/pixel} = \underbrace{\frac{6 \times 4}{4}C_{l+1}}_{\text{bandpass}} + \underbrace{C_{l+1}}_{\text{lowpass}} + \underbrace{36}_{DTCWT} + \underbrace{36}_{DTCWT^{-1}} = 7C_{l+1} + 72 \quad (1.6.8)$$

This is marginally smaller than a  $3 \times 3$  convolutional layer.

### 1.6.4 Parameter Initialization

For both layer types we use the Glorot Initialization scheme [10] with  $a = 1$ :

$$g_{ij} \sim U \left[ -\sqrt{\frac{6}{(C_l + C_{l+1})k^2}}, \sqrt{\frac{6}{(C_l + C_{l+1})k^2}} \right] \quad (1.6.9)$$

where  $k$  is the kernel size.



## 1.7 Gain Layer 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.

In this section, we are only exploring the wavelet gain layer and not any wavelet nonlinearities. This equates to the path shown in Figure 1.3b. We 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 ??, 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).

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.

## 1.8 Wavelet Based Nonlinearities and Multiple Gain Layers

So far we have only considered doing the mixing operations in the wavelet domain. This is a good starting point, and it is nice to see that we can do apply the gain layer that preserve some of the nice properties of the DTCWT. Let us call the layer considered so far a **first order gain layer**. Recall this is where a DTCWT is done, followed by a single linear convolution and then straight away returning to the pixel domain with the inverse DTCWT. While it is good to see that it is possible to do and even achieves some benefit over a convolutional layer, the proposed layer (B.0.17) 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 possible 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 [11], [12].

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.

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 DWT architecture takes a single scale DWT of the activations at the given layer, and mixes the coefficients to get new wavelet coefficients. The lowpass mixing coefficients for the DWT architecture have spatial size  $3 \times 3$  and the bandpass coefficients have spatial size  $1 \times 1$ . The DTCWT architecture takes a single scale DTCWT and mixes the lowpass and complex bandpass coefficients. Both the lowpass and bandpass coefficients have spatial size  $1 \times 1$ . Numbers are averages over 3 runs.

	CIFAR-10		CIFAR-100		Tiny ImgNet	
	DWT	DTCWT	DWT	DTCWT	DWT	DTCWT
reference	92.6		72.0		59.3	
gainA	93.0	93.0	72.3	72.8	61.3	60.8
gainB	93.0	92.9	72.5	73.4	61.3	61.3
gainC	92.8	93.2	72.1	72.8	60.8	60.7
gainD	92.9	93.3	72.9	73.1	60.3	60.3
gainE	93.3	93.0	72.8	72.7	60.8	60.6
gainF	93.6	93.1	73.6	72.7	61.3	61.3
gainA, gainB	91.9	92.2	70.6	72.3	60.6	60.9
gainB, gainC	92.1	92.6	71.7	73.1	60.3	60.7
gainC, gainD	91.9	92.9	71.0	72.4	60.1	59.9
gainD, gainE	93.1	92.8	72.4	71.9	60.6	60.0
gainA, gainC	92.4	92.8	71.8	73.6	60.7	60.0
gainB, gainD	92.5	92.8	71.7	72.9	60.6	61.2
gainC, gainE	92.4	92.7	71.7	72.2	59.7	60.1
gainA-F						

### 1.8.1 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

## 1.9 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 [13]. 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.

# Appendices

## Appendix A

# Invertible Transforms and Optimization

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:

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

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

$$X = \text{DFT}\{x\} = UxU \tag{A.0.1}$$

$$x = \text{DFT}^{-1}\{X\} = U^*YU^* \tag{A.0.2}$$

When it comes to gradients, these become:

$$\frac{\partial L}{\partial X} = U \frac{\partial L}{\partial x} U = \text{DFT}\left\{\frac{\partial L}{\partial x}\right\} \tag{A.0.3}$$

$$\frac{\partial L}{\partial x} = U^* \frac{\partial L}{\partial X} U^* = \text{DFT}^{-1}\left\{\frac{\partial L}{\partial X}\right\} \tag{A.0.4}$$

Now consider a single filter parameterized in the DFT and spatial domains presented with the exact same data and with the same  $\ell_2$  regularization  $\epsilon$  and learning rate  $\eta$ . Let the spatial filter at time  $t$  be  $\mathbf{w}_t$ , the Fourier-parameterized filter be  $\hat{\mathbf{w}}_t$ , and let

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

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:

$$\mathbf{w}_2 = \mathbf{w}_1 - \eta \left( \frac{\partial L}{\partial \mathbf{w}} + \epsilon \mathbf{w}_1 \right) \quad (\text{A.0.6})$$

$$= (1 - \eta\epsilon) \mathbf{w}_1 - \eta \frac{\partial L}{\partial \mathbf{w}} \quad (\text{A.0.7})$$

$$\hat{\mathbf{w}}_2 = \hat{\mathbf{w}}_1 - \eta \left( \frac{\partial L}{\partial \hat{\mathbf{w}}} + \epsilon \hat{\mathbf{w}}_1 \right) \quad (\text{A.0.8})$$

$$= (1 - \eta\epsilon) \hat{\mathbf{w}}_1 - \eta \frac{\partial L}{\partial \hat{\mathbf{w}}} \quad (\text{A.0.9})$$

$$(\text{A.0.10})$$

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 [A.0.3](#) and [A.0.5](#) we then get:

$$\text{DFT}^{-1}\{\hat{\mathbf{w}}_2\} = \text{DFT}^{-1}\left\{(1 - \eta\epsilon)\hat{\mathbf{w}}_1 - \eta \frac{\partial L}{\partial \hat{\mathbf{w}}}\right\} \quad (\text{A.0.11})$$

$$= (1 - \eta\epsilon)\mathbf{w}_1 - \eta \text{DFT}^{-1}\left\{\frac{\partial L}{\partial \hat{\mathbf{w}}}\right\} \quad (\text{A.0.12})$$

$$= (1 - \eta\epsilon)\mathbf{w}_1 - \eta \frac{\partial L}{\partial \mathbf{w}} \quad (\text{A.0.13})$$

$$= \mathbf{w}_2 \quad (\text{A.0.14})$$

### A.0.1 Regularization

If we use  $\ell_1$  then the above doesn't hold.

### A.0.2 Optimization

If we use adam things are different.

This does not hold for the Adam [\[14\]](#) 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 [\[7\]](#).

## 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:

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

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  $A(z) = 2\text{Re}(P(z))$ ,  $B(z) = 2\text{Im}(P(z))$ ,  $C(z) = 2\text{Re}(Q(z))$  and  $D(z) = -2\text{Im}(Q(z))$ , then the aliasing terms in [\(B.0.1\)](#) are nearly zero and the system is nearly shift invariant.*

*Proof.* See section 4 of [\[9\]](#) 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:

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

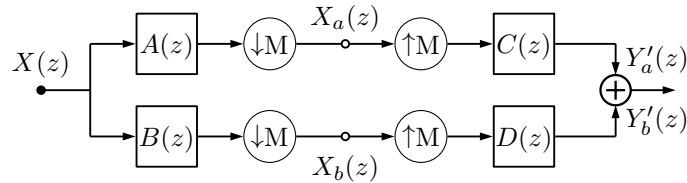


Figure B.1: **Block Diagram of 1-D DTCWT.** Note the top and bottom paths are through the wavelet or scaling functions from just level  $m$  ( $M = 2^m$ ). Figure based on Figure 4 in [\[9\]](#).

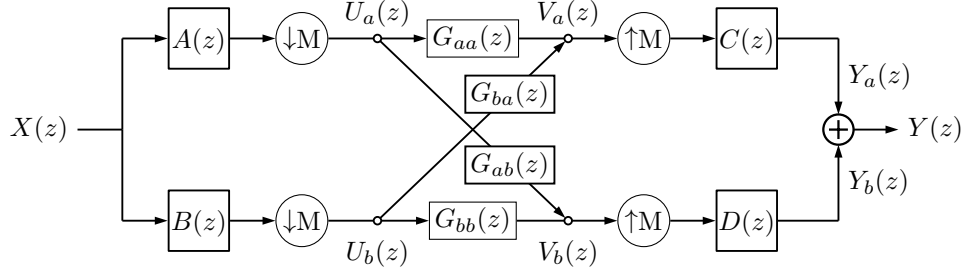


Figure B.2: **Block Diagram of 1-D DTCWT**. Note the top and bottom paths are through the wavelet or scaling functions from just level  $m$  ( $M = 2^m$ ). Figure based on Figure 4 in [9].

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:

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

Using (B.0.2), 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  $k \neq 0$ .  $\square$

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

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

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

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 function from  $U_i$  to  $V_j$  be  $G_{ij}$  for  $i, j \in \{a, b\}$ . Then  $V_a$  and  $V_b$  are:

$$V_a(z) = U_a(z)G_{aa}(z) + U_b(z)G_{ba}(z) \quad (\text{B.0.5})$$

$$= \frac{1}{M} \sum_k X(W^k z^{1/k}) [A(W^k z^{1/k})G_{aa}(z) + B(W^k z^{1/k})G_{ba}(z)] \quad (\text{B.0.6})$$

$$V_b(z) = U_a(z)G_{ab}(z) + U_b(z)G_{bb}(z) \quad (\text{B.0.7})$$

$$= \frac{1}{M} \sum_k X(W^k z^{1/k}) [A(W^k z^{1/k})G_{ab}(z) + B(W^k z^{1/k})G_{bb}(z)] \quad (\text{B.0.8})$$

Further,  $Y_a$  and  $Y_b$  are:

$$Y_a(z) = C(z)V_a(z^M) \quad (\text{B.0.9})$$

$$Y_b(z) = D(z)V_b(z^M) \quad (\text{B.0.10})$$



Then the end to end transfer function is:

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

**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:

$$A(W^k z)D(z) - B(W^k z)C(z) = j \left[ P(W^k z) + P^*(W^k z) \right] [Q(z) - Q^*(z)] + \quad (\text{B.0.13})$$

$$j \left[ P(W^k z) - P^*(W^k z) \right] [Q(z) + Q^*(z)] \quad (\text{B.0.14})$$

$$= 2j \left[ P(W^k z)Q(z) - P^*(W^k z)Q^*(z) \right] \quad (\text{B.0.15})$$

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.  $\square$

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

$$\frac{Y(z)}{X(z)} = \frac{2}{M} \left[ G_r(z^M) (A(z)C(z) + B(z)D(z)) + G_i(z^M) (A(z)D(z) - B(z)C(z)) \right] \quad (\text{B.0.16})$$

$$= \frac{2}{M} \left[ G_r(z^M) (PQ + P^*Q^*) + jG_i(z^M) (PQ - P^*Q^*) \right] \quad (\text{B.0.17})$$

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.

## 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] 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.
- [9] 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.
- [10] 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.
- [11] V. Pappayan, 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.
- [12] V. Pappayan, Y. Romano, and M. Elad, “Convolutional Neural Networks Analyzed via Convolutional Sparse Coding”, *arXiv:1607.08194 [cs, stat]*, Jul. 2016. arXiv: [1607.08194 \[cs, stat\]](#).
- [13] D. L. Donoho and J. M. Johnstone, “Ideal spatial adaptation by wavelet shrinkage”, in *Biometrika*, vol. 81, no. 3, pp. 425–455, Sep. 1994.
- [14] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: [1412.6980 \[cs\]](#).