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. However, with three or more such layers (in an 6 layer network) the network performance degrades.

Additionally, we have not been successful in finding a nonlinearity that works well in the wavelet domain.

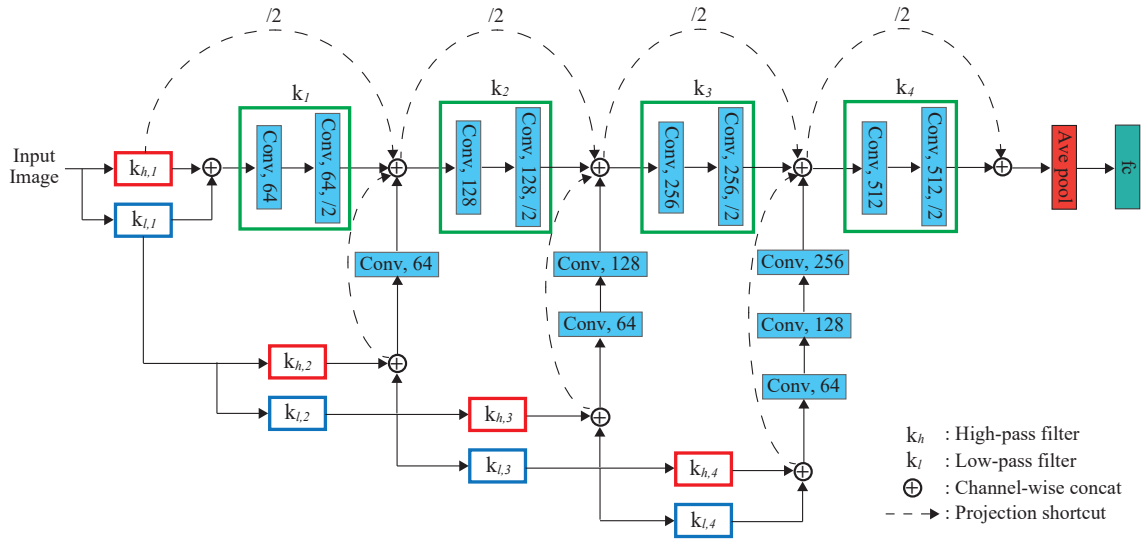

Figure 1.1: **Architecture using the DWT as a frontend to a CNN.** Figure 1 from [2]. Fujieda et. al. take a multiscale wavelet decomposition of the input before passing the input through a standard CNN. They learn convolutional layers independently on each subband and feed these back into the network 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, so are a little less applicable than the first work.

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.

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

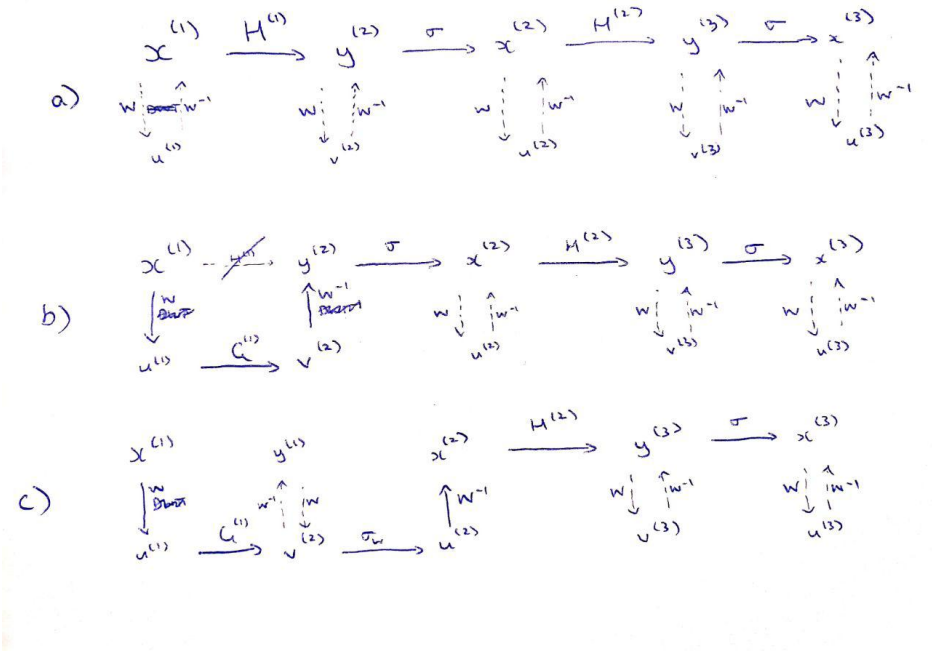

Figure 1.2: **Proposed new forward pass in the wavelet domain.** Two network layers with some possible options for processing. Solid lines denote the evaluation path and dashed lines indicate relationships. In (a) we see a regular convolutional neural network. We have included the dashed lines to make 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.

Each of these coefficients then has size:

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

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

Note that the lowpass coefficients are twice as large as in a fully decimated transform, a feature of the redundancy of the DT$\mathbb{C}$WT.

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 DT$\mathbb{C}$WT 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 DT$\mathbb{C}$WT 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 a wavelet based nonlinearity σ_w 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 G is at replacing H.

2. How effective σ_w is at replcaing σ.

1.3.1 The DT$\mathbb{C}$WT 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.