1 Introduction

Using wavelet based methods with deep learning is nascent but not novel. Wavelets have been applied to texture classification [5, 18], super-resolution [6] and for adding detail back into dense pixel-wise segmentation tasks [13]. One exciting piece of work built on wavelets is the Scattering Transform [14], which has been used as a feature extractor for learning, firstly with simple classifiers [1, 19], and later as a front end to hybrid deep learning tasks[15, 20]. Despite their power and simplicity, scattering features are fixed and are visibly different to regular CNN features [3] - their nice invariance properties come at the cost of flexibility, as there is no ability to learn in between scattering layers.

For this reason, we have been investigating a slightly different approach, more similar to the Fourier based work in [16] in which Rippel et. al. investigate parameterization of filters in the Fourier domain. In the forward pass, they take the inverse DFT of their filter, and then apply normal pixel-wise convolution. We wish to extend this by not only parameterizing filters in the wavelet domain, but by performing the convolution there as well (i.e., also taking the activations into the wavelet domain). After processing is done, we can return to the pixel domain. Doing these forward and inverse transforms has two significant advantages:

1. The layers can easily replace standard convolutional layers if they accept and return the same format.

2. We can learn both in the wavelet and pixel space.

As neural network training involves presenting thousands of training samples, we want our layer to be fast. To achieve this we would ideally choose to use a critically sampled filter bank implementation. The fast 2-D Discrete Wavelet Transform (DWT) is a possible option, but it has two drawbacks: it has poor directional selectivity and any alteration of wavelet coefficients will cause the aliasing cancelling properties of the reconstructed signal to disappear. Instead we choose to use the Dual-Tree Complex Wavelet Transform (DT$\mathbb{C}$WT) [17] as at the expense of limited redundancy (4:1), it enables us to have better directional selectivity, and allows us to modify the wavelet coefficients and still have minimal aliasing terms when we reconstruct [9].

sec:method of the paper describes the implementation details of our design, and section 5 describes the experiments and results we have done so far.

2 Similar Work

2.1 Parameterizing filters in Fourier Domain

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

[1]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

Note that this is almost identical to a normal CNN, except it allows for slightly a reworked filter initialization and regularization, as well as affecting optimizers that keep track of past updates (such as SGD with momentum, Adam [8] or Adagrad). Figure 1 shows the fourier and spatial representation of some of the filters in [16], as well as histograms showing the sparsity of parameters and momenta for parameter updates. The lower mean in the distribution of parameter momenta indicates that fewer parameters are being updated in a constant direction.

2.1.1 Initialization

No specific mention is made of the initialization technique used but it may well be helpful to put a prior over the spectra.

2.1.2 Regularization

Due to Parseval's theorem, it is clear that applying an L2 regularization on the filter weights :

$$\sum_i \|w_i\|_2^2 = \sum_i \|\hat{w}_i\|_2^2 = \sum_i \|\mathrm{Re}\,(\hat{w}_i)\|_2^2 + \|\mathrm{Im}\,(\hat{w}_i)\|_2^2$$

However can apply L1 regularization on the complex magnitude of the DFT filter weights to impose spectral sparsity.

2.1.3 Optimization

If we define the DFT as the orthonormal version, i.e. for a square image, let:

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

then

$$Y = \mathrm{DFT}\{X\} = UXU \tag{2.1}$$
$$X = \mathrm{DFT}^{-1}\{Y\} = U^*YU^* \tag{2.2}$$
$$\Delta X = \frac{\partial L}{\partial X} = U^*\Delta YU^* = \mathrm{DFT}^{-1}\{\Delta Y\} \tag{2.3}$$
$$\Delta Y = \frac{\partial L}{\partial Y} = U\Delta XU = \mathrm{DFT}\{\Delta X\} \tag{2.4}$$

Parameterizing in the fourier domain does not affect linear optimizers like vanilla SGD and SGD with momentum (including with Nesterov momentum). For example consider a single filter parameterized in the DFT and spatial domains presented with the exact same data (and for the moment with no regularization). Let the initial values for $\hat{\mathbf{w}}$ and $\mathbf{w}$ be:

$$\hat{\mathbf{w}}_1 = \alpha \tag{2.5}$$
$$\mathbf{w}_1 = \beta = \mathrm{DFT}^{-1}\{\alpha\} \tag{2.6}$$

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:

$$\begin{aligned}
\mathbf{w}_2 &= \mathbf{w}_1 - \eta \left.\frac{\partial L}{\partial \mathbf{w}}\right|_{\mathbf{w}=\beta} \\
&= \beta - \eta \delta_{\mathbf{w}} \\
\hat{\mathbf{w}}_2 &= \hat{\mathbf{w}}_1 - \eta \left.\frac{\partial L}{\partial \hat{\mathbf{w}}}\right|_{\hat{\mathbf{w}}=\alpha} \\
&= \alpha - \eta \mathrm{DFT}\{\delta_{\mathbf{w}}\}
\end{aligned}$$

We can then compare the effect the new parameters would have on the next minibatch by calculating $\mathrm{DFT}^{-1}\{\hat{\mathbf{w}}_2\}$:

$$\begin{aligned}
\mathrm{DFT}^{-1}\{\hat{\mathbf{w}}_2\} &= \mathrm{DFT}^{-1}\{\alpha - \eta \mathrm{DFT}\{\delta_{\mathbf{w}}\}\} \\
&= \mathrm{DFT}^{-1}\{\alpha\} - \eta \mathrm{DFT}^{-1}\{\mathrm{DFT}\{\delta_{\mathbf{w}}\}\} \\
&= \beta - \eta \delta_{\mathbf{w}} \\
&= \mathbf{w}_2
\end{aligned}$$

In fact it can easily be seen that any optimizer that uses linear combinations of gradients makes this new parameterization identical to parameterization in the spatial domain. Optimizers like Adam and Adagrad however have update rules that are not simply linear combinations of past gradients and so this affects the learning trajectory. For Adam, there is a step where the gradients are squared to estimate the variance of the parameter updates.

This shows that any work involving reparametrizing or rethinking filter convolution must be careful in what is done.

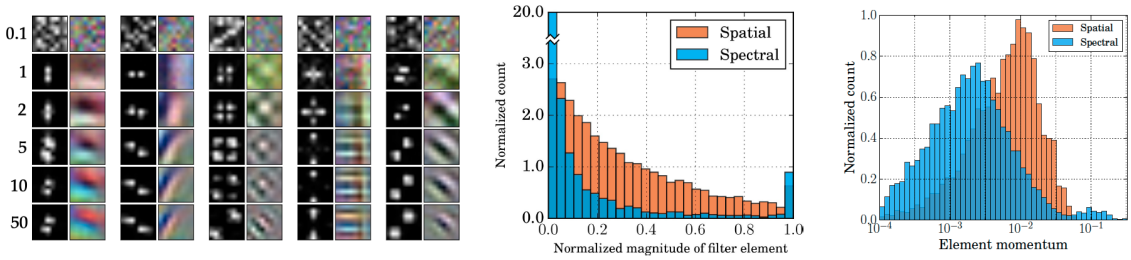
Further to this, while the inspiration for the work comes from wanting to remain in the frequency domain, it is unsatisfying to simply parameterize filters there and then take the inverse DFT and perform normal convolution. Ideally we would like to fully take advantage of the benefits of a frequency representation.

2.2 Spectral Pooling

The Spectral Pooling method introduced in [16] is described in the paper as:

> We assume we are given an input $\mathbf{x} \in \mathbb{R}^{M \times N}$, and some desired output map dimensionality $H \times W$. First we compute the discrete Fourier transform of the input into the frequency domain as $y = \mathcal{F}(\mathbf{x}) \in \mathbb{C}^{M \times N}$, and assume that the DC componenet has been shifted to the center of the domain as is standard practice. We then crop the frequency representation by maintaining only the central $H \times W$ submatrix of frequencies, which we denote as $\hat{\mathbf{y}} \in \mathbb{C}^{H \times W}$. Finally, we map this approximation back into the spatial domain by taking its inverse DFT as $\hat{\mathbf{x}} = \mathcal{F}^{-1}(\hat{\mathbf{y}}) \in \mathbb{R}^{H \times W}$.

While they have chosen to do this in the Fourier domain, this could also be achieved by convolving with a separable, resampled 2D sinc function of the appropriate frequencies for the horizontal and vertical directions:



(a) Filters over time. (b) Sparsity patterns. (c) Momenta distributions.

Figure 1: **Learning Dynamics of CNNs with DFT parameterization.** (a) Progression over several epochs of filters parametrized in the frequency domain. The left column shows the parameterized filter and the right column its inverse DFT. (b) Sparsity patterns for different parameterizations - spectral representations tend to be sparser. (c) Momenta distributions for parameters of CNNs trained with and without spectral parameterization. In the spectral parameterization fewer parameters are updated. Image taken from [16].

$$\hat{\mathbf{x}} = \frac{HW}{MN}\mathbf{x} * \text{sinc}(\frac{Hn_y}{M}) * \text{sinc}(\frac{Wn_x}{N})$$

where $n_x \in \{??\}, n_y \in \{??\}$. Of course, sincs have infinite support in the time domain, so to achieve a similar result to spectral pooling you would have to convolve with windowed sincs, or do some other form of similar low pass filtering with resampling. In [21] they do exactly this and call it 'wavelet pooling', which is just keeping the low-low output of a separable 2D DWT, or simply convolving with a separable lowpass filter. They experimentally showed that this was equivalent to spectral and average pooling.

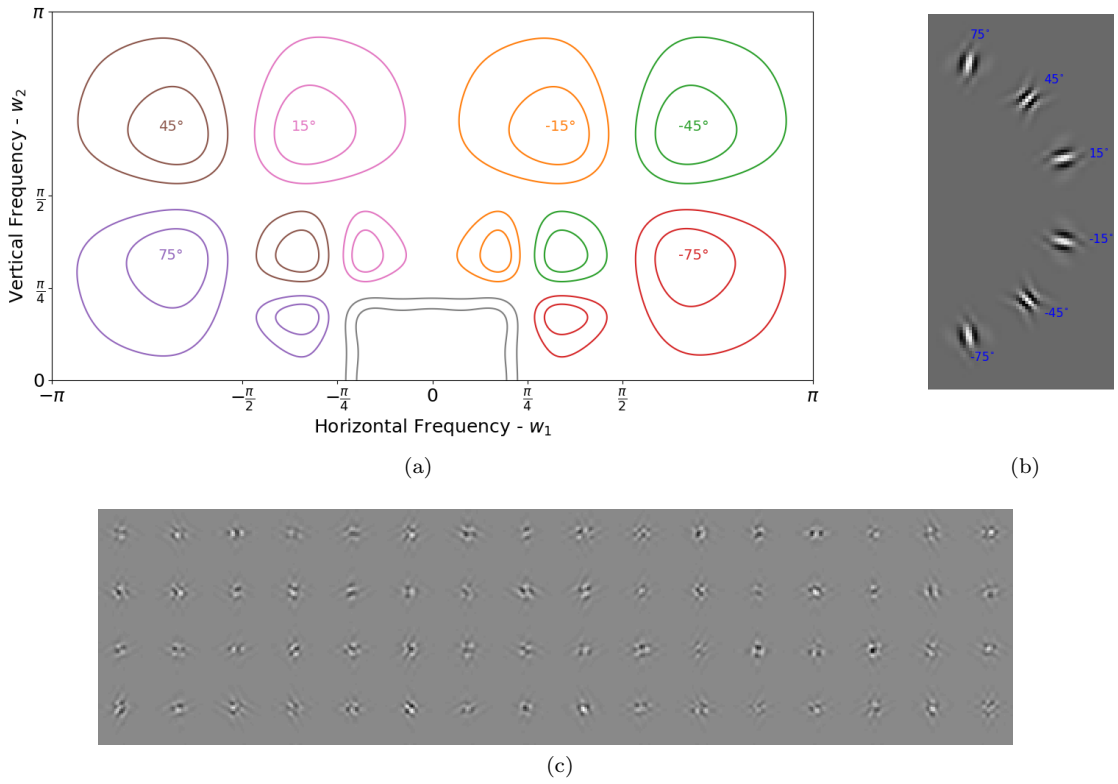
Note that speed ups could be achieved here if the convolution is done in the Fourier domain, as it would involve only computing the reduced spectrum size before taking inverse DFTs.

3 Aliasing in the DWT

Consider

4 Method

In a standard convolutional layer, an input with C channels, H rows and W columns is $X \in \mathbb{R}^{C\times H\times W}$, which is then convolved with F filters of spatial size K - $w \in \mathbb{R}^{F\times C\times K\times K}$, giving $Y \in \mathbb{R}^{F\times H\times W}$. In many systems like [11, 7], the first layer is typically a selection of bandpass filters, selecting edges with different orientations and center frequencies. In the wavelet space this would be trivial - take a decomposition of each input channel and keep individual subbands (or equivalently, attenuate other bands), then take the inverse wavelet transform. Figure 2 shows the frequency space for the DT$\mathbb{C}$WT and makes it clearer as to how this could be done practically for a two scale transform. To attenuate all but say the 15° band at the first scale for the first input



(a) (b)

(c)

Figure 2: **Building blocks of the** DT$\mathbb{C}$WT **gain layer.** (a) Contour plots at -1dB and -3dB showing the support in the Fourier domain of the 6 subbands of the DT$\mathbb{C}$WT at scales 1 and 2 and the scale 2 lowpass. These are the product $P(z)Q(z)$ from Equation 4.1.impulse responses for the second scale wavelets. (c) Example impulses of our layer when g_1, and g_{lp} are 0 and $g_2 \in \mathbb{C}^{6\times1\times1}$, with each real and imaginary element drawn from $\mathcal{N}(0,1)$. I.e., only information in the 6 subbands with $\frac{\pi}{4} < |w_1|, |w_2| < \frac{\pi}{2}$ from (a) is passed through.

channel, we would need to have $13C$ gains for the 13 subbands and C input channels, $13C-1$ of which would be zero and the remaining coefficient one.

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

4.1 Memory Cost

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

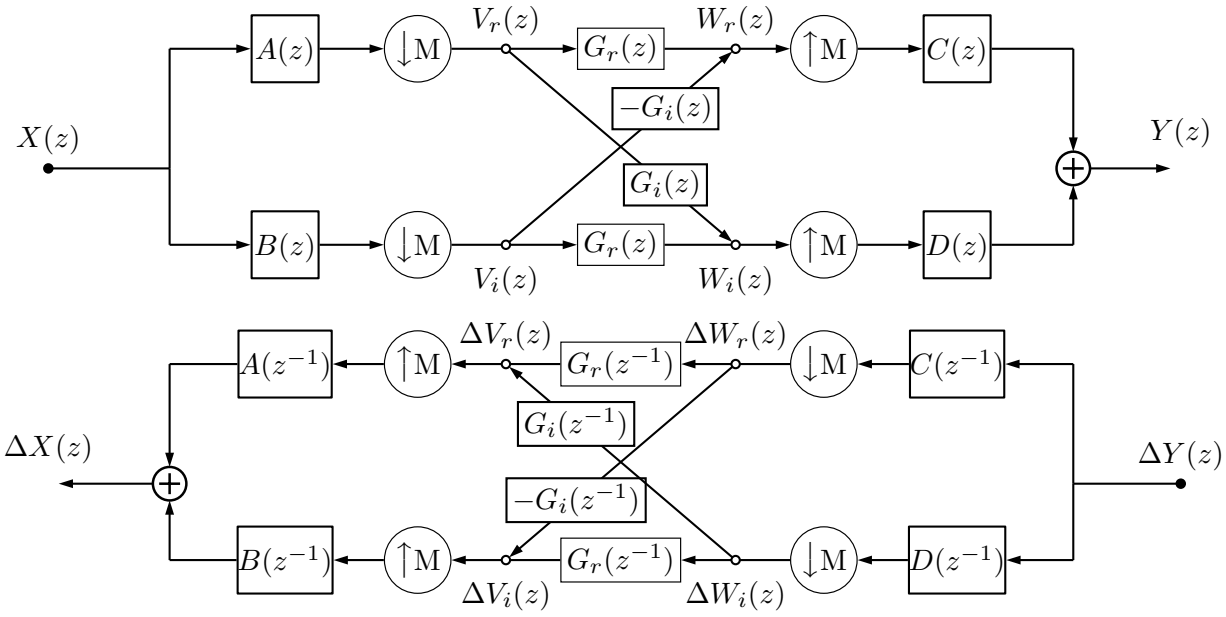


Figure 3: **Block Diagram of 1-D** DT$\mathbb{C}$WT **gain layer.** (Top) Forward and (bottom) backward pass of our system, based on Figure 4 in [9]. Ignoring the G gains, the top and bottom paths (through A, C and B, D respectively) make up the the real and imaginary parts for *one subband* of the dual 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 2). 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).

$$
\begin{aligned}
g_1 \in \quad & \mathbb{C}^{F \times C \times 6 \times 1 \times 1} \\
g_2 \in \quad & \mathbb{C}^{F \times C \times 6 \times 1 \times 1} \\
g_{lp} \in \quad & \mathbb{R}^{F \times C \times 1 \times 1}
\end{aligned}
$$

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

4.2 Computational Cost

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

4.3 Examples

2c show example impulse responses of our layer. These impulses were generated by randomly initializing both the real and imaginary parts of $g_2 \in \mathbb{C}^{6\times1\times1}$ from $\mathcal{N}(0,1)$ and g_1, g_{lp} are set to 0. I.e. each shape has 12 random variables. It is good to see that there is still a large degree of variability between shapes. Our experiments have shown that the distribution of the normalized cross-correlation between 512 of such randomly generated shapes matches the distribution for random vectors with roughly 11.5 degrees of freedom.

4.4 Forward propagation

Figure 3 shows the block diagram using Z-transforms for a single band of our system (it is based on Figure 4 in [9]). To keep things simple for the rest of section 4 the figure shown is for a 1-D system; it is relatively straightforward to extend this to 2-D[17]. The complex analysis filter (taking us into the wavelet domain) is

$$P(z) = \frac{1}{2}\left(A(z) + jB(z)\right)$$

and the complex synthesis filter (returning us to the pixel domain) is

$$Q(z) = \frac{1}{2}\left(C(z) - jD(z)\right)$$

where A, B, C, D are real. If $G(z) = G_r(z) + jG_i(z) = 1$ then the end-to-end transfer function is (from section 4 of [9]):

$$\frac{Y(z)}{X(z)} = \frac{2}{M}\left(P(z)Q(z) + P^*(z)Q^*(z)\right) \tag{4.1}$$

where P, Q have support only in the top half of the Fourier plane and P^*, Q^* are P and Q reflected in the horizontal frequency axis. Examples of $P(z)Q(z)$ for different subbands of a 2-D DT$\mathbb{C}$WT have spectra shown in 2a, $P^*(z)Q^*(z)$ make up the missing half of the frequency space. Modifying this from the standard wavelet equations by adding the subband gains $G_r(z)$ and $G_i(z)$, the transfer function becomes:

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

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

$$\frac{Y(z)}{X(z)} = \frac{2}{M}\left[G_r(z^M)\left(P(z)Q(z) + P^*(z)Q^*(z)\right) + jG_i(z^M)\left(P(z)Q(z) - P^*(z)Q^*(z)\right)\right] \tag{4.2}$$

4.5 Backpropagation

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

$$\Delta X(z) = H(z^{-1})\Delta Y(z) \tag{4.3}$$

where $H(z^{-1})$ is the Z-transform of the time/space reverse of $H(z)$, $\Delta Y(z) \triangleq \frac{\partial L}{\partial Y}(z)$ is the gradient of the loss with respect to the output, and $\Delta X(z) \triangleq \frac{\partial L}{\partial X}(z)$ is the gradient of the loss with respect to the input. If H were complex, the first term in Equation 4.3 would be $\bar{H}(1/\bar{z})$, but as each individual block in the DT$\mathbb{C}$WT is purely real, we can use the simpler form.

Assume we already have access to the quantity $\Delta Y(z)$ (this is the input to the backwards pass). **??** illustrates the backpropagation procedure. An interesting result is that the backwards pass of an inverse wavelet transform is equivalent to doing a forward wavelet transform.[2] Similarly, the backwards pass of the forward transform is equivalent to doing the inverse transform. The weight update gradients are then calculated by finding $\Delta W(z) = \text{DT}\mathbb{C}\text{WT}\{\Delta Y(z)\}$ and then convolving with the time reverse of the saved wavelet coefficients from the forward pass - $V(z)$.

$$\Delta G_r(z) = \Delta W_r(z)V_r(z^{-1}) + \Delta W_i(z)V_i(z^{-1}) \tag{4.4}$$

$$\Delta G_i(z) = -\Delta W_r(z)V_i(z^{-1}) + \Delta W_i(z)V_r(z^{-1}) \tag{4.5}$$

Unsurprisingly, the passthrough gradients have similar form to Equation 4.2:

[2] As shown in **??**, the analysis and synthesis filters have to be swapped and time reversed. For orthogonal wavelet transforms, the synthesis filters are already the time reverse of the analysis filters, so no change has to be done. The q-shift filters of the DT$\mathbb{C}$WT [10] have this property.

$$\Delta X(z) = \frac{2\Delta Y(z)}{M}\Big[G_r(z^{-M})\,(PQ + P^*Q^*) + jG_i(z^{-M})\,(PQ - P^*Q^*)\Big] \tag{4.6}$$

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

Note that we only need to evaluate equations 4.4, 4.5,4.6 over the support of $G(z)$ i.e., if it is a single number we only need to calculate $\Delta G(z)|_{z=0}$.

5 Experiments and Preliminary Results

To examine the effectiveness of our convolutional layer, we do a simple experiment on CIFAR-10 and CIFAR-100. For simplicity, we compare the performance using a simple yet relatively effective convolutional architecture - LeNet [12]. LeNet has 2 convolutional layers of spatial size 5×5 followed by 2 fully connected layers and a softmax final layer. We swap both these convolutional layers out for two of our proposed wavelet gain layers (keeping the ReLU between them). As CIFAR has very small spatial size, we only take a single scale DT$\mathbb{C}$WT . Therefore each gain layer has 6 complex gains for the 6 subbands, and a 3×3 real gain for the lowpass (a total of $21C$ parameters vs $25C$ for the original system). We train both networks for 200 epochs with Adam [8] optimizer with a constant learning rate of 10^{-3} and a weight decay of 10^{-5}. The code is available at [2]. Table 1 shows the mean of the validation set accuracies for 5 runs. The different columns represent undersampled training set sizes (with 50000 being the full training set). When undersampling, we keep the samples per class constant. We see our system perform only very slightly worse than the standard convolutional layer.

6 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 [4]. We feel there are strong links between ReLU non-linearities and denoising/sparsity ideas, and that there may well be useful performance gains from mixing real pixel-domain non-linearities with complex wavelet-domain shrinkage functions. Thus we present these ideas here as a starting point for a novel and exciting avenue of deep network research.

References

[1] J. Bruna and S. Mallat. "Invariant Scattering Convolution Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1872–1886.

[2] Fergal Cotter. *DTCWT Gainlayer*. Nov. 2018.

[3] Fergal Cotter and Nick Kingsbury. "Visualizing and Improving Scattering Networks". In: *arXiv:1709.01355 [cs]* (Sept. 2017). arXiv: 1709.01355 [cs].

[4] David L. Donoho and Jain M. Johnstone. "Ideal Spatial Adaptation by Wavelet Shrinkage". en. In: *Biometrika* 81.3 (Sept. 1994), pp. 425–455.

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

[6] Tiantong Guo et al. "Deep Wavelet Prediction for Image Super-Resolution". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Honolulu, HI, USA: IEEE, July 2017, pp. 1100–1109.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385 [cs].

[8] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980 [cs].

[9] N. Kingsbury. "Complex Wavelets for Shift Invariant Analysis and Filtering of Signals". In: *Applied and Computational Harmonic Analysis* 10.3 (May 2001), pp. 234–253.

[10] N. Kingsbury. "Design of Q-Shift Complex Wavelets for Image Processing Using Frequency Domain Energy Minimization". In: *2003 International Conference on Image Processing, 2003. ICIP 2003. Proceedings*. Vol. 1. Sept. 2003, I-1013-16 vol.1.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*. Curran Associates, Inc., 2012, pp. 1097–1105.

[12] Y. Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324.

[13] Lingni Ma et al. "Detailed Dense Inference with Convolutional Neural Networks via Discrete Wavelet Transform". In: *arXiv:1808.01834 [cs]* (Aug. 2018). arXiv: 1808.01834 [cs].

[14] Stéphane Mallat. "Group Invariant Scattering". en. In: *Communications on Pure and Applied Mathematics* 65.10 (Oct. 2012), pp. 1331–1398.

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

[16] Oren Rippel, Jasper Snoek, and Ryan P Adams. "Spectral Representations for Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2440–2448.

[17] Ivan W. Selesnick, Richard G. Baraniuk, and Nick G. Kingsbury. "The Dual-Tree Complex Wavelet Transform". In: *Signal Processing Magazine, IEEE* 22.6 (2005), pp. 123–151.

[18] Laurent Sifre and Stéphane Mallat. "Combined Scattering for Rotation Invariant Texture Analysis". In: *European Symposium on Artificial Neural Networks (ESANN) 2012*. 2012.

[19] A. Singh and N. Kingsbury. "Scatternet Hybrid Deep Learning (SHDL) Network for Object Classification". In: *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. Sept. 2017, pp. 1–6.

[20] Amarjot Singh. "ScatterNet Hybrid Frameworks for Deep Learning". PhD thesis. University of Cambridge, May 2018.

[21] Travis Williams and Robert Li. "Wavelet Pooling for Convolutional Neural Networks". In: (Feb. 2018).