Learning Filters for the 2D Wavelet Transform

Daniel Recoskie and Richard Mann
Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
{*dprecosk, mannr*}*@uwaterloo.ca*

***Abstract*—We propose a new method for learning filters for the 2D discrete wavelet transform. We extend our previous work on the 1D wavelet transform in order to process images. We show that the 2D wavelet transform can be represented as a modified convolutional neural network (CNN). Doing so allows us to learn wavelet filters from data by gradient descent. Our learned wavelets are similar to traditional wavelets which are typically derived using Fourier methods. For filter comparison, we make use of a cosine measure under all filter rotations. The learned wavelets are able to capture the structure of the training data. Furthermore, we can generate images from our model in order to evaluate the filters. The main findings of this work is that wavelet functions can arise naturally from data, without the need for Fourier methods. Our model requires relatively few parameters compared to traditional CNNs, and is easily incorporated into neural network frameworks.**

***Keywords*-wavelets, convolution neural networks, filter banks**

I. Introduction

The purpose of this work is to extend our previous 1D wavelet transform model to 2D [1]. The wavelet transform is a linear time-frequency transform that makes use of a multiscale filter bank made up of self-similar wavelet filters. We focus on orthogonal filters so that we can perfectly reconstruct the input signal from its wavelet coefficients. Generally, wavelet filters are designed using Fourier methods. We propose a new method of learning wavelet filters directly from data. We accomplish this by framing the wavelet transform as a variant of a convolutional neural network (CNN). Our learning method makes use of an autoencoder framework [2]. We impose a sparsity constraint on the learned representation in order to learn wavelets that exploit structure in the training data.

A motivation of our previous work was to construct a model that was able to learn directly from raw 1D signals, such as audio. Typical models first perform a fixed feature transform (e.g. the Fourier transform) instead of dealing with the raw data directly [3]. More recently, there has been work showing impressive results on raw audio [4]. In [1] we proposed a novel model based on the wavelet transform that was able to learn useful filters from 1D data. The results are summarized in Figure 1.

We extend our 1D model in order to learn wavelet filters from 2D image data. This is in contrast to the more traditional method of using a fixed feature representation such as SIFT [5] or SURF [6]. One of the most notable methods of learning directly from image data is the CNN [7], which learns a set of 2D filters that are applied in a cascade. Our model instead learns 1D filters that are applied along each dimension of the image. Thus, the filters still have a 2D receptive field, but fewer parameters are required. Furthermore, we reuse the filters in each layer of the network, unlike in a traditional CNN where separate filters are used at each layer.

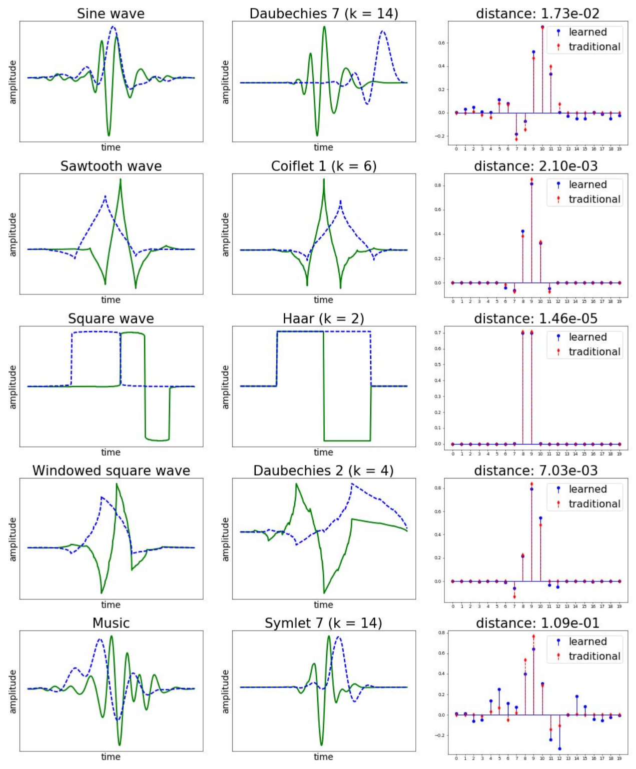


Figure 1. Summary of results from [1]. Left column: Learned wavelet (solid) and scaling (dashed) functions. Type of training data is shown above the plots. Middle column: Closest traditional wavelet (solid) and scaling (dashed) functions according to (17). Right column: Plots of the scaling filters from the first two columns with corresponding distance measure.

The wavelet transform has been used in neural networks in the past, such as in the wavelet network [8] and the

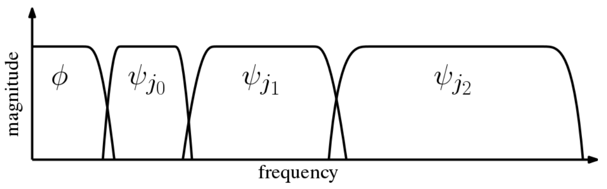


Figure 2. Frequency view of the wavelet functions.

scattering transform [9]. These models do not learn wavelet filters, but instead use fixed filters. The scattering transform computes coefficients similar to those of SIFT descriptors [10]. Learning wavelet filters using neural networks has been proposed before in [11]. However, the signals considered had domains over the vertices of graphs. Furthermore, second generation wavelets were considered [12]. In this work we consider only first generation wavelets.

II. Wavelet Transform

A. 1D Wavelet Transform

We will begin our discussion with the 1D wavelet transform. The wavelet transform is a linear time-frequency transform that makes use of a dictionary of wavelets. Wavelets are functions that are localized in time and frequency. In a traditional wavelet transform, each wavelet is a shifted and dilated version of a mother wavelet, ψ. We restrict ourselves to the discrete case, where our wavelets are of the form

$$\psi_j[n] = \frac{1}{2^j}\psi\left(\frac{n}{2^j}\right) \tag{1}$$

for $n, j \in \mathbb{Z}$. Let x be a 1D discrete signal of length N. The discrete wavelet transform is computed by convolving x with the wavelet functions:

$$Wx[n, 2^j] = \sum_{m=0}^{N-1} x[m]\psi_j[m-n]. \tag{2}$$

The wavelet functions are constrained to have zero mean and unit norm. For a fixed shift n, the wavelets form an overlapping bandpass filter bank illustrated in Figure 2. In order to cover the entire frequency axis, we require the notion of a scaling function, ϕ. The scaling function can be thought of as the sum of all wavelet functions above a fixed scale. Formally, it is defined such that its Fourier transform has the following property [13]:

$$|\hat{\phi}(\omega)|^2 = \int_1^{+\infty} \frac{|\hat{\psi}(s\omega)|^2}{s} ds \tag{3}$$

In order to compute the discrete wavelet transform efficiently, we will make use of an iterative algorithm. Let us first define two filters:

$$h[n] = \left\langle \frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle \tag{4}$$

$$g[n] = \left\langle \frac{1}{\sqrt{2}}\psi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle \tag{5}$$

We call h the scaling (lowpass) filter, and g the wavelet (highpass) filter. We will use k to denote the length of the filters. The discrete wavelet transform can then be computed with the following iterative steps:

$$a_{j+1}[p] = \sum_{n=-\infty}^{+\infty} h[n-2p]a_j[n] \tag{6}$$

$$d_{j+1}[p] = \sum_{n=-\infty}^{+\infty} g[n-2p]a_j[n] \tag{7}$$

with $a_0 = x$ (i.e. the input signal). We call a_j and d_j the approximation and detail coefficients respectively. Note that the detail coefficients are exactly the wavelet coefficients from (2). The discrete wavelet transform algorithm makes use of a cascade of convolutions, each followed by downsampling. At each iteration, the signal is split into high and low frequency components. The high frequency components correspond to the wavelet coefficients. The low frequency coefficients are then used in the next iteration.

The original signal can be recovered from the approximation and detail coefficients using:

$$\begin{aligned} a_j[p] = &\sum_{n=-\infty}^{+\infty} h[p-2n]a_{j+1}[n] \\ &+ \sum_{n=-\infty}^{+\infty} g[p-2n]d_{j+1}[n] \end{aligned} \tag{8}$$

Note that the coefficients are upsampled by a factor of two at each iteration by inserting zeros at even indices.

B. 2D Wavelet Transform

The discrete wavelet transform can be extended to two dimensions by computing the convolutions along each axis separately [14]. In the 1D case, we computed two components at each iteration of the algorithm (highpass and lowpass). In the 2D case, we will compute four components. Let LR and LC correspond to convolving the scaling (lowpass) filter along the rows and columns respectively. We can similarly define HR and HC for the wavelet (highpass) filter. At every iteration of the 2D wavelet transform algorithm, we compute the four components as illustrated in Figure 3:

$$\text{approximation: } LR(LC(x)) \tag{9}$$

$$\text{detail horizontal: } LR(HC(x)) \tag{10}$$

$$\text{detail vertical: } HR(LC(x)) \tag{11}$$

$$\text{detail diagonal: } HR(HC(x)) \tag{12}$$

where x is now a 2D discrete signal. The three components computed using at least one wavelet filter are kept as the detail coefficients. The single approximation component computed by convolving the scaling filter along both axes is passed to the next iteration of the algorithm. The output of

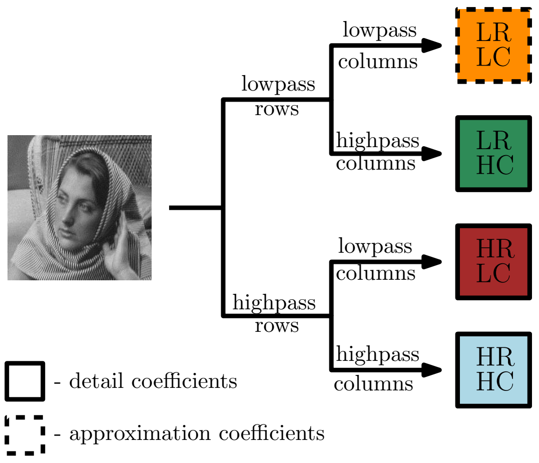


Figure 3. One iteration of the 2D discrete wavelet transform.

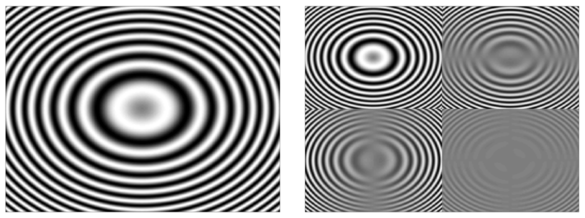
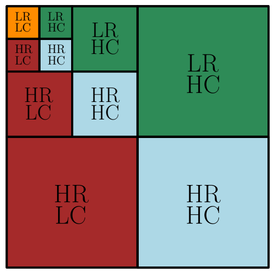
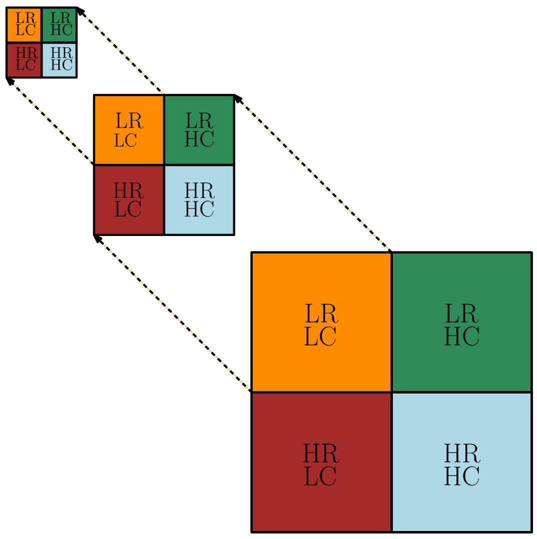


Figure 4. One iteration of the 2D discrete wavelet transform applied to a circular image. Not that the different coefficient components pick up different edge orientations.



(a)



(b)

Figure 5. (a) Wavelet coefficient matrix after three iterations of the wavelet transform algorithm. (b) The wavelet coefficients are computed from the scaling coefficients of the previous iteration.

the transform is structured as in Figure 5a. Note that after each convolution, the image is downsampled by a factor of two along the direction of the convolution. Computing the 2D wavelet transform in this fashion is used in the JPEG2000 standard [15].

The three components containing the detail coefficients each correspond to a different orientation: horizontal, vertical, and diagonal. Each component responds to changes along its corresponding direction. Figure 4 shows one level of the wavelet transform applied to an image. Note that the three detail coefficient components highlight different edge orientations.

As in the 1D case, we subsample by a factor of two after each convolution. Thus, the total number of coefficients is equal to the number of pixels in our original image. Figure 5 shows how the coefficients are typically represented graphically. Note that we can discard the approximation coefficients after each iteration as they can be reconstructed by the coefficients at the next level.

To get an idea of how the 1D wavelet filters behave in 2D, we can compute the impulse responses associated with each orientation. The impulse responses are computed by an inverse wavelet transform on wavelet coefficients that have a single nonzero value in each of the three detail coefficient components. The impulse responses are thus the images that each filter orientation maximally responds to. Figure 6 shows the impulse responses for a typical wavelet filter. The first two impulse responses correspond to the horizontal and vertical components. The third impulse response has a checkerboard appearance since it is effectively the product of the first two. The 2D wavelet transform thus suffers from poor directional selectivity, which can be addressed by using a dual-tree version of the transform [16]. We leave this for future work.

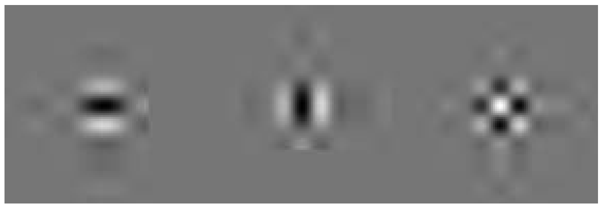


Figure 6. Impulse responses of a typical wavelet filter.

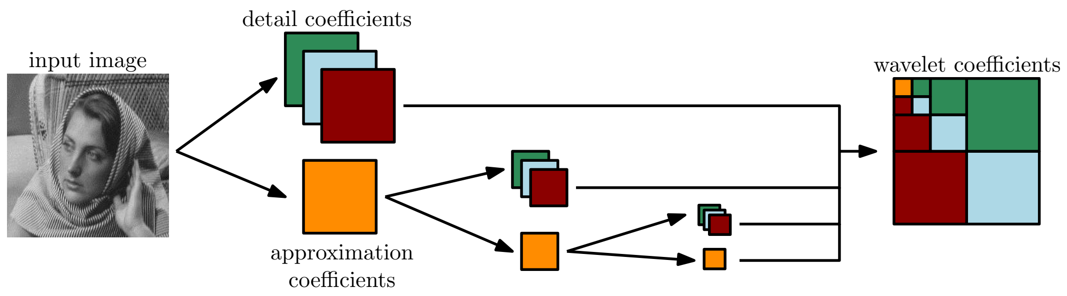


Figure 7. The wavelet transform as a neural network. Each layer of the network computes the three detail components and single approximation component. The approximation coefficients are passed to the next layer, while the detail coefficients are passed to the final layer. The number of layers corresponds to the number of iterations of the wavelet transform algorithm.

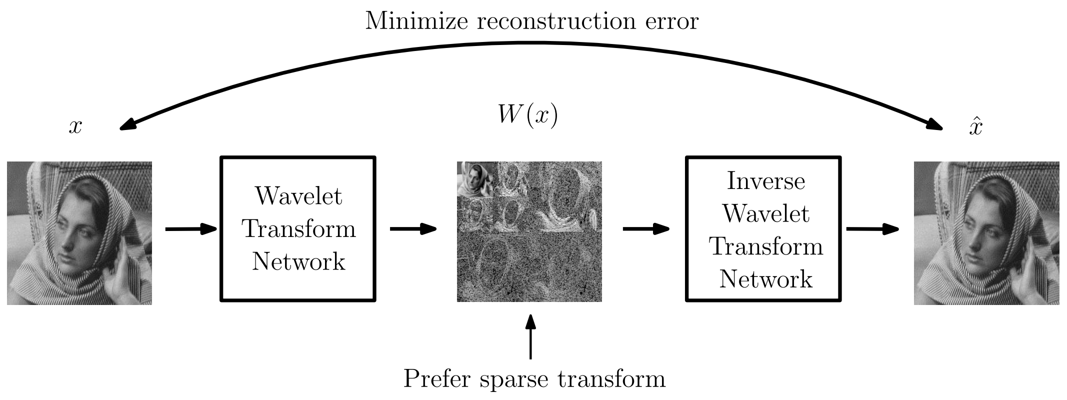

Minimize reconstruction error
x
$W(x)$
$\hat{x}$
Wavelet Transform Network
Inverse Wavelet Transform Network
Prefer sparse transform

Figure 8. The autoencoder framework used in our experiments. An L_1 penalty is put on the wavelet coefficients so that the learned wavelets must exploit the structure in the data.

III. The Wavelet Transform as a Neural Network

The wavelet transform is computed by a cascade of convolutions. This computation is similar in structure to that of CNNs. As such, we propose a modified CNN that directly computes the wavelet transform. By doing so, we are able to leverage the mathematical properties of the wavelet transform into the successful deep learning architectures. In simplest terms, our model is an unrolling of the discrete wavelet transform algorithm. Figure 7 shows an overview of our wavelet transform network. Each layer of the network computes one iteration of the discrete wavelet transform. The detail coefficients are passed directly to the final output layer. The approximation coefficients are passed as input to the next layer. The parameters of the network are the wavelet and scaling filters. These filters are reused at each layer of the network, and hence the model is only required to learn a single filter pair. This property is similar to that of recurrent neural networks, where weights are reused at each time step [17]. In our network, however, weights are reused at each scale.

In this work we consider quadrature mirror filters. Hence, we restrict the filters such that

$$g[n] = (-1)^n h[-n]. \tag{13}$$

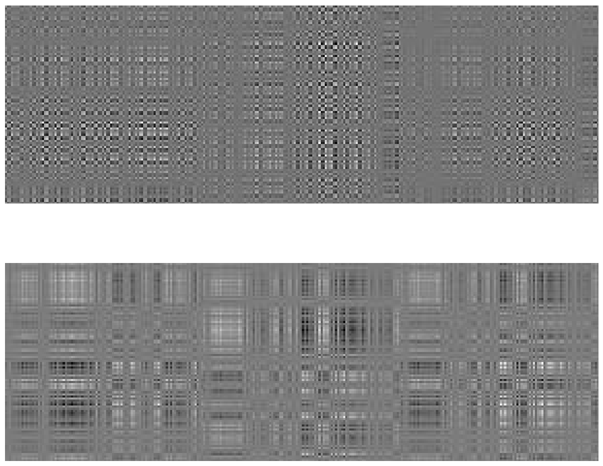


Figure 9. Impulse responses of two random filters satisfying (14). Each filter has a length of ten.

In other words, the wavelet filter is derived from the scaling filter by reversing it and negating the odd indices. Thus, we must only learn the scaling filter in order to define the network parameters.

In order for our learned filters to compute a valid wavelet transform, we introduce the following constraints on the filters [1]:

$$L_w(h,g) = (||h||_2 - 1)^2 + (\mu_h - \sqrt{2}/k)^2 + \mu_g^2 \qquad (14)$$

where μ_h and μ_g are the filter means. The first two terms prefer a scaling function with unit L_2 norm and finite L_1 norm respectively [18], [13]. The last term prefers a wavelet filter with zero mean. These constraints define a family of filters. See Figure 9 for a selection of random filters that minimize (14). We can see that the constraints by themselves are not sufficient for learning meaningful filters. The space of filters that minimize (14) may share similarities to the space of parameterized wavelet families [19].

We make use of an autoencoder framework in order to demonstrate that our model is able to learn useful wavelet functions [2]. Figure 8 shows an overview of the framework. The goal is to reconstruct an input image by first computing the forward wavelet transform, imposing sparsity constraints on the coefficients, and then performing an inverse wavelet transform. We choose to use an L_1 sparsity constraint in order to prefer mostly zero coefficients.

We argue that a wavelet that gives a sparse representation of an image must exploit inherent structure present in the data. Thus, the model must learn something useful about the images used for training. In experiments we make use of images of faces [20] and synthetic images containing harmonic waves of different shapes. We extend the synthetic generation process for 1D data from [1]:

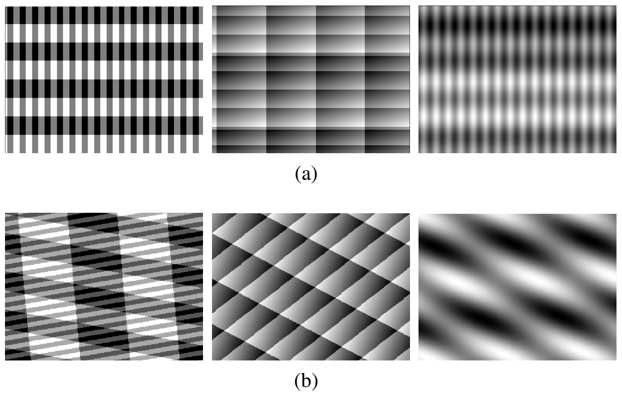
$$x(t) = \sum_{k=0}^{K-1} a_k \cdot s(2^k t + \phi_k) \qquad (15)$$

where ϕ_k is a phase offset chosen uniformly at random from $[0, 2\pi]$, and a_k is the k^{th} harmonic indicator which takes the value of 1 with probability p and zero otherwise. In our experiments we set $p = 1/2$. In order to generate images, we first choose a random orientation angle for each harmonic wave. In the case of axis-aligned waves, the angle is 0 or $\pi/2$. The waves are then added to the image along the chosen orientation and extended orthogonally to fill the image. Three base waves are considered: square waves, sawtooth waves, and sine waves. See Figure 10 for samples of synthetic images

In order to learn the parameters of the network, we must first define a loss function over a dataset of images $X = \{x_1, x_2, \ldots x_M\}$:

$$L(X; g, h) = \frac{1}{M}\sum_{i=1}^{M} ||x_i - \hat{x}_i||_2^2 + \lambda_1 \frac{1}{M}\sum_{i=1}^{M} ||W(x_i)||_1 + \lambda_2 L_w(h,g) \qquad (16)$$

We use mean squared error for our reconstruction loss and the L_1 norm for the sparsity penalty. The parameters λ_1 and λ_2 control the trade-off between the three loss terms. In our experiments we set $\lambda_1 = \lambda_2 = 1/2$ and used a filter length of ten. Our model was implemented using Google's Tensorflow library and makes use of automatic differentiation [21]. We trained using the Adam gradient descent algorithm with a batch size of four [22].


(a)

(b)

Figure 10. Samples of our synthetic images with harmonics that are (a) axis-aligned and (b) randomly oriented. Base waves from left to right: square, sawtooth, and sine

A sample of the learned wavelet functions can be found in Figure 11. These functions were learned from axis-aligned data where applicable. The wavelets learned form the randomly oriented data had similar structure. The wavelet and scaling functions are computed from the filter coefficients using the cascade algorithm [23]. Note that the learned wavelet functions are able to capture the structure of the different base waves present in the data. We compared the learned filters to traditional wavelets from the following families: Haar, Daubechies, Symlets, and Coiflets. The most similar traditional wavelets are included in Figure 11. The distance measure used was the cosine distance under all rotations of the filters. It is defined below:

$$dist(h_1, h_2) = \min_{0 \le i < k} 1 - \frac{\langle h_1, \text{shift}(h_2, i)\rangle}{||h_1||_2 \cdot ||h_2||_2} \qquad (17)$$

where shift(h, i) is the circular shift of h by i samples. If the two filters are of different lengths, the shorter filter is zero-padded.

We can also consider the impulse responses of the filters. Figure 14 shows the impulse responses of the learned filters from Figure 11. Surface plots of the impulse responses are also included. Note that the first two impulse responses are axis-aligned. The shape of each impulse response is similar in structure to its corresponding training data. Square data yields rectangular filters, sawtooth data yields triangular filters, and sine data yields filters that appear Gabor-like.

One way to determine how well the learned wavelets capture the structure of the data is to generate images from the model. Image generation has a long history in computer science [24]. We make use of two different generative

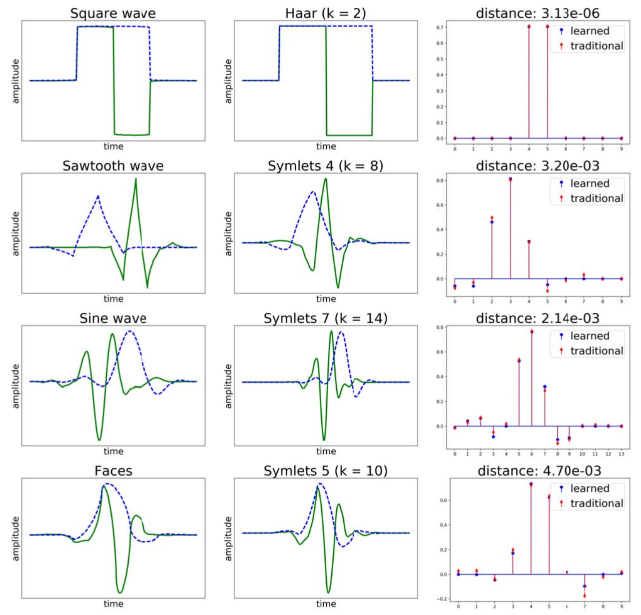


Figure 11. Summary of learned filters. Left column: Learned wavelet (solid) and scaling (dashed) functions. Type of training data is shown above the plots. Middle column: Closest traditional wavelet (solid) and scaling (dashed) functions according to (17). Right column: Plots of the scaling filters from the first two columns with corresponding distance measure.

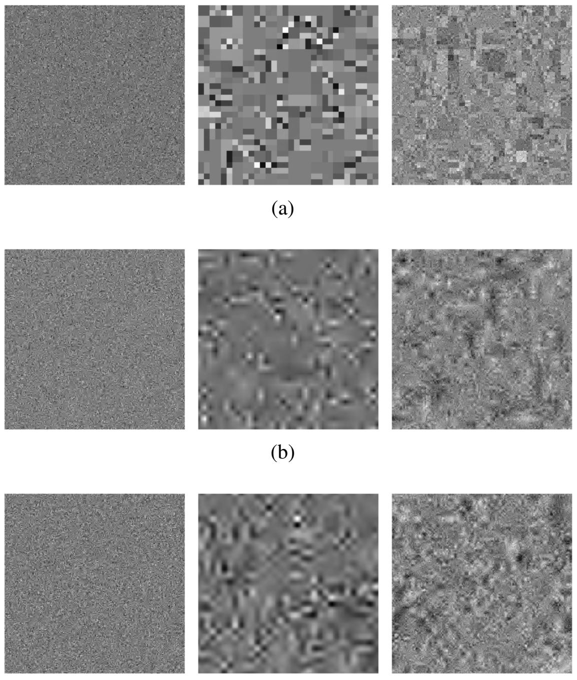


Figure 12. Generated image data using process one with (a) square wavelet, (b) sawtooth wavelet, and (c) sine wavelet. Left column: equal power at each scale. Middle column: equal power at each scale with three highest scales removed. Right column: coefficients scaled so that lower frequencies are more pronounced.

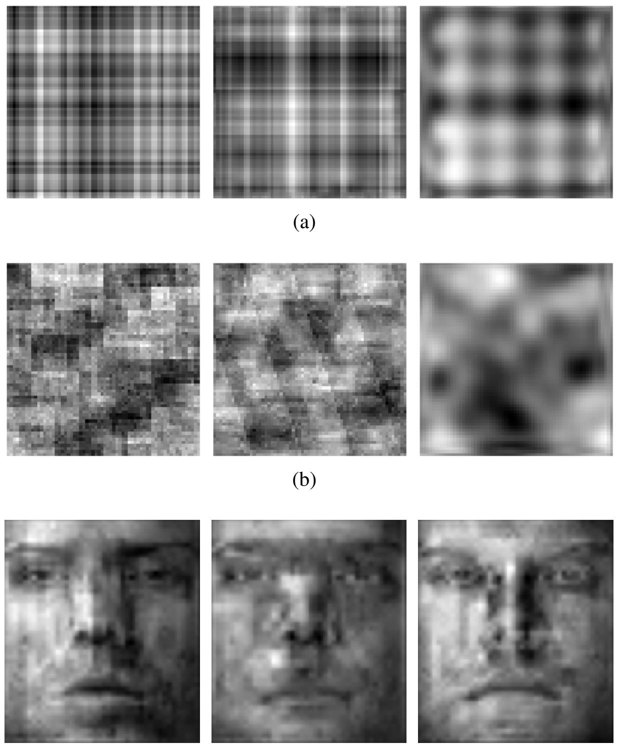
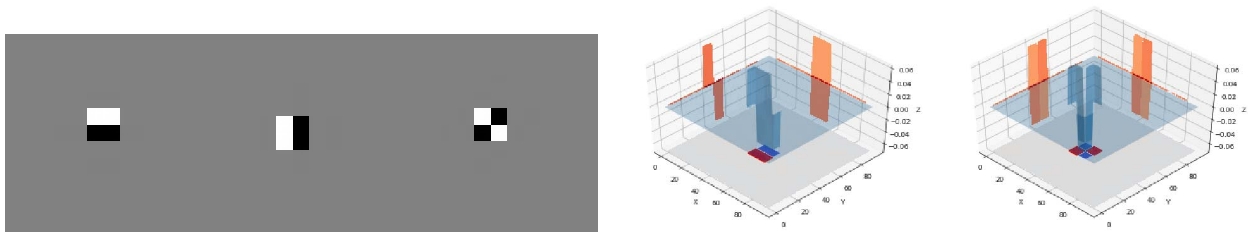
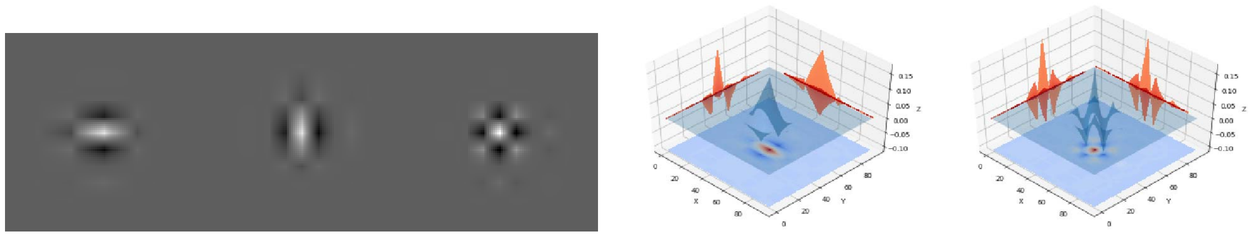


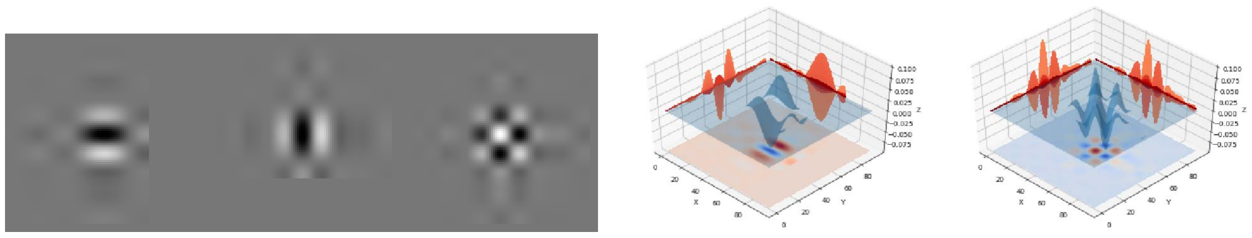
Figure 13. Generated image data using process two with (a) axis-aligned data, (b) random harmonic orientations, and (c) faces. (a-b) Wavelets from left to right: square wavelet, sawtooth wavelet, and sine wavelet.

processes. Our first generative process consists of sparsely populating random wavelet coefficients, and then computing an inverse transform using one of the learned wavelets. The density of coefficients is equal across all wavelet scales. Figure 12 shows examples of generated images using various learned wavelets. The first column uses coefficients of equal magnitude and includes coefficients at each scale. It is difficult to see any structure in this data as it appears similar to white noise. The second column is similar, but excludes the three highest frequency scales. We can see that different wavelets generate different structured images. The third column scales the magnitudes of the coefficients so that low frequency scales are more pronounced (similar to pink noise). We again see the different structure of the wavelets.

The second generative process treats each band of wavelet coefficients at each scale as a multivariate normal distribution. We estimate the means and covariances of the distributions by computing the wavelet coefficients of a sample of 32 images from our synthetic training data. We then sample wavelet coefficients from the multivariate distributions and perform an inverse wavelet transform to obtain our final image. Figures 13a and 13b show some sample images. Note the similarity to the training samples in Figure 10. We repeated this process for the face dataset by sampling


(a)


(b)


(c)

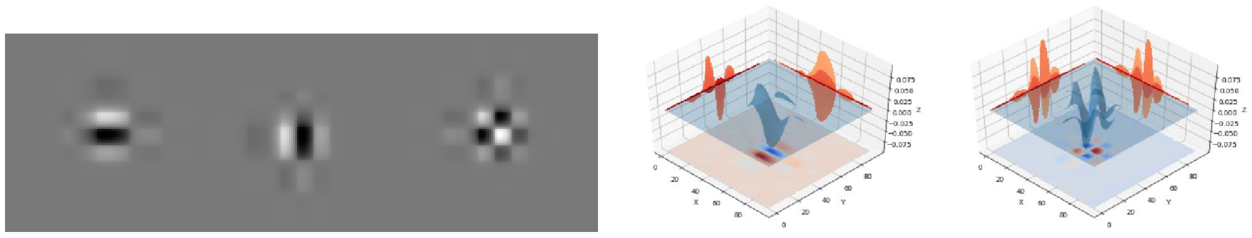

(d)

Figure 14. Impulse responses of the learned filters from Figure 11. The training data was (a) square waves, (b) sawtooth waves, (c) sine waves, and (d) faces. Surface plots of the horizontal and diagonal orientations are also shown shown.

32 images of a single subject. For efficiency, we set the coefficients in the highest two wavelet scales to zero since much of this high frequency data is noise. Sample generated face images are shown in Figure 13c.

IV. Conclusion

We have proposed a new model for learning wavelet filters directly from image data. We have shown that useful wavelets can be learned using an autoencoder framework with sparsity constraints. The autoencoder is comprised of a 2D discrete wavelet transform followed by an inverse transform. Preferring a sparse representation forces the model to learn wavelets that exploit structure in the training data. A reconstruction constraint means that the learned filters are (nearly) orthogonal, and so can be used for both analysis and synthesis. We frame our model as a modified CNN, making it easy to incorporate into existing neural network architectures. A benefit of this model over traditional CNNs is that we require very few parameters. This is due to two properties: the filters used are one-dimensional, and the filters are reused at each layer of the network.

Acknowledgment

Daniel Recoskie would like to thank NSERC for funding.

References

[1] D. Recoskie and R. Mann, “Learning sparse wavelet representations,” *arXiv preprint arXiv:1802.02961*, 2018.

[2] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[4] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” *arXiv preprint arXiv:1609.03499*, 2016.

[5] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. IEEE, 1999, pp. 1150–1157.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (surf),” *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[7] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in neural information processing systems*, 1990, pp. 396–404.

[8] Q. Zhang and A. Benveniste, “Wavelet networks,” *IEEE transactions on Neural Networks*, vol. 3, no. 6, pp. 889–898, 1992.

[9] S. Mallat, “Group invariant scattering,” *Communications on Pure and Applied Mathematics*, vol. 65, no. 10, pp. 1331–1398, 2012.

[10] J. Bruna and S. Mallat, “Classification with scattering operators,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1561–1566.

[11] R. Rustamov and L. J. Guibas, “Wavelets on graphs via deep learning,” in *Advances in Neural Information Processing Systems*, 2013, pp. 998–1006.

[12] W. Sweldens, “The lifting scheme: A construction of second generation wavelets,” *SIAM journal on mathematical analysis*, vol. 29, no. 2, pp. 511–546, 1998.

[13] S. Mallat, *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*, 3rd ed. Academic Press, 2008.

[14] S. G. Mallat, “A theory for multiresolution signal decomposition: the wavelet representation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[15] C. Christopoulos, A. Skodras, and T. Ebrahimi, “The jpeg2000 still image coding system: an overview,” *IEEE transactions on consumer electronics*, vol. 46, no. 4, pp. 1103–1127, 2000.

[16] I. W. Selesnick, R. G. Baraniuk, and N. C. Kingsbury, “The dual-tree complex wavelet transform,” *IEEE signal processing magazine*, vol. 22, no. 6, pp. 123–151, 2005.

[17] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[18] S. G. Mallat, “Multiresolution approximations and wavelet orthonormal bases of $L^2(R)$,” *Transactions of the American mathematical society*, vol. 315, no. 1, pp. 69–87, 1989.

[19] C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to wavelets and wavelet transforms: a primer*. Prentice-Hall, Inc., 1997.

[20] K.-C. Lee, J. Ho, and D. J. Kriegman, “Acquiring linear subspaces for face recognition under variable lighting,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 5, pp. 684–698, 2005.

[21] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org.

[22] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[23] G. Strang and T. Nguyen, *Wavelets and filter banks*. SIAM, 1996.

[24] D. Mumford and A. Desolneux, *Pattern theory: the stochastic analysis of real-world signals*. CRC Press, 2010.