An exploration of wavelet convolutional neural networks

Zeppelin Vanbarriger

***Abstract*—This paper investigates the efficacy of wavelets and multi-resolution analysis with regards to convolutional neural networks. We found that Wavelet based convolutional neural networks performed similarly as traditional convolutional neural networks.**

I. Introduction

Waveletes and their extension multi-resolution analysis are an important mathematical tool for the exploration of spectral data, particularly in images. By repeatedly applying high and low pass filters, spectral data at different "resolutions" can be extracted. Additional analysis can then be done on the extracted features.

Modern convolutional neural networks (CNNs) are a powerful tool that operates on images, and is popular for classification, labeling, and predictive tasks. These CNNs use convolutions on image data for feature extraction. The filters of these convolutions are learned through back-propagation on the network as in traditional neural networks.

This paper investigates the efficacy of a network which combines these two. The spectral feature extraction wavelet transforms provide could add additional learning power over standard CNNs. Two models- a wavelet CNN (WCNN) and a traditional CNN- are constructed and tested on a multi-class classification problem. Datasets are taken from [FFP], consisting of 10 classes of images, each containing more than 300 samples.

A. Literature Review

The inspiration for this project was found in [FTH18], where the formulation of CNNs as a limited type of multi-resolution analysis was proposed. The authors suggested that an integration of the wavelet transform into the network could allow the it to utilize spectral information that traditional CNNs do not learn. The efficacy of a WCNN was explored in comparison to standard high-performance neural networks such as AlexNet and texture CNNs (T-CNNs). The authors found that their networks performed slightly better or comparable to the baseline, while requiring fewer parameters.

II. Wavelet Convolutional Neural Networks

We propose a synthesis of CNNs and wavelet transforms by reformulating the convolutions of a CNN as a type of filter as in the case of wavelet transforms. Our specification will focus on 2D data as in images. We will first introduce convolutions and the building blocks of the typical CNN, then the wavelet transform/multi-resolution analysis, and finally their synthesis.

A. *Convolutions*

A convolution in the context of CNNs is a mathematical operation where parts of the input data is convolved with a filer or kernel. That is, given an input $X^{l-1} \in \mathbb{R}^{n\times n}$ and a kernel $K \in \mathbb{R}^{k\times k}$, the output $X^l \in \mathbb{R}^{m\times m}$ is described as

$$x_{i,j} = \sum_{p,q\in M_{ij}} x^l_{p,q} k_{p,q}$$

,where l is the output layer, $l-1$ is the input layer, and M_{ij} are the neighbors of the input $x^{l-1}_{i,j}$. These neighbors are determined by the size of the kernel as well as the stride s of the convolution. Common kernel sizes are 2×2 and 3×3. Common strides are 1 and 2. Additionally, a "padding" layer of 0s is typically added to the input data. This padding is a set of row vectors added before and after to the first and last rows of the matrix respectively. This is done similarly to the first and last columns as well. The output data size $m\times m$ is given as $m = \frac{n-k+2p}{s}+1$. As an aside, a "channel" is commonly used in CNNs to add an additional dimension for feature extraction. For an example, a single channel input $x \in \mathbb{R}^{n\times n\times l}$ will often be convolved to some output $y \in\in \mathbb{R}^{n\times n\times m}$, where m is the out channel amount. Thus in multi-channel convolutions, the number of kernels is really $l\times m$.

B. *Pooling*

It is common after convolution layers to apply a max or average pooling operation. These pooling layers let the network learn spatial invariance. In this network, average pooling was used. Given an input $X^{l-1} \in \mathbb{R}^{n\times n}$, the output $X^l \in \mathbb{R}^{m\times m}$ is described as

$$x^l_{ij} = \frac{1}{r}\sum_{p,q=0}^{r} x^{l-1}_{p,q}$$

.

In this network, global average pooling was used, that is

$$x^l = \frac{1}{n^2}\sum_{p=1}^{n}\sum_{q=1}^{n} x^{l-1}_{p,1}$$

.

C. Deep (Feed-forward) Networks

A "typical" feed-forward network consists of an input layer I, hidden layers H^j, $j = 1, ...n-1$, and an output layer O. The CNN described in this paper takes in outputs from the convolution layers as inputs to a feed-forward network of size $n = 2$ (one hidden layer H^1 and an output layer O). Each layer L^l consists of a number of "neurons". Each neuron x_j^l receives input from the neurons of the previous layer x^{l-1} multiplied by some weight value which is the passed in to some nonlinear activation function. This is written as

$$x_j^l = f(u^{l-1})$$
$$u^{l-1} = \sum_i x_i^{l-1} w_i$$

,where f is some nonlinear function such as $tanh$, $logistic$, or $ReLU$. $ReLU$ is used in this network and is given as

$$f(x) = max(x, 0)$$

When $l = n$, $L^l = O$ and so the output activation function is chosen based on the type of problem and the associated loss function. In this paper, the network uses the softmax activation function which is written as

$$\sigma : \mathbb{R}^n \to \mathbb{R}^n$$
$$\sigma(u_i) = \frac{e^{u_i}}{\sum_{j=1}^{k=1} e^{u_j}}$$

for $j = 1, ..., n$ and $u = (u_1, ..., u_n)^T \in \mathbb{R}^n$. This activation function takes in n real numbers and gives a probability distribution over n probabilities. This is suitable for a multi-class classification problem where each output probability gives the likelihood that that input belongs to a certain class.

The associated loss function is a way of measuring the error rate of the network's predictions. For softmax activation, that error rate is given by Cross-Entropy Loss:

$$L = -\sum_i^n \sum_k^K t_k^n log(y_k^n)$$

Here, t_k^n is the k^{th} class of the n^{th} sample's target label and y_k^n is the value of the k^{th} output layer unit.

D. Backpropagation

Neural networks learn by adjusting the weight values between neurons. The weights are adjusted through a process called gradient descent of the error, or backpropagation. Essentially, the goal is to change the weights in the direction of the maximum decrease in the error function. Consider $E = -\sum_i^n \sum_k^K t_k^n log(y_k^n)$. To compute the gradient of the error with respect to the weights, we first need to compute the gradient of the error with respect to the input to the softmax. We have $\frac{\partial E}{\partial u} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial u}$. Since

$$\frac{\partial E}{\partial y} = -\sum_i^n \sum_k^K t_k^n \frac{1}{(y_k^n)} \frac{\partial y_k^n}{\partial u_k}$$

and

$$\frac{\partial y_i}{\partial u_j} = \sigma_i(\delta_{ij} - \sigma_j)$$

, for $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$, we can conclude that

$$\frac{\partial E}{\partial u_j} = \sigma_i^n - t_i^n$$

.

Let us define $\delta^n = \frac{\partial E}{\partial y}$. Then, $\frac{\partial E}{\partial W^n} = x^{n-1}\delta^n$. In general, we can define a recurrence relation $\delta^l = (W^{l+1})^T \delta^{l+1} * f'(u^l)$, where $*$ is element-wise multiplication. So, our update rule for W^l is: $W'^l = W^l - \mu \frac{\partial E}{\partial W^l}$.

E. Wavelets and Multi-resolution Analysis

Wavelet transforms and multi-resolution analysis are mathematical transforms which extract spectral information from input data. More formally, wavelet transforms provide a means of decomposing a function into a series of basis functions where the coefficients for each basis function represents some information about the original function with respect to that part of the basis [AH13]. In this treatment, we will only discuss the Haar wavelets. The Haar scaling function is defined as:

$$\phi(x) = \begin{cases} 1, & 0 \leq x < 1 \\ 0, & x < 0 \text{ or } x \geq 1 \end{cases}$$

We can translate $\phi(x)$ by an integer $k \in \mathbb{Z}$, as in the form $\phi(x-k)$. A function $f \in \{\phi(x-k) | k \in \mathbb{Z}\}$ can be represented as $f(x) = \sum_k^{n \neq \infty} a_k \phi(x-k)$. We can also scale these functions such as $\phi(2^j x - k), k \in \mathbb{Z}$ to form a level j subspace V_j of $L^2(\mathbb{R})$. We can thus write $f \in V_j$ as $f(x) = \sum_k^{n \neq \infty} a_k \phi(2^j x - k)$.

We can also define a function $\psi(x) = \phi(2x) - \phi(2x-1)$, called the Haar wavelet function. This function defines a subspace W_j consisting of all finite linear combinations of $\psi(2^j x - k), k \in \mathbb{Z}$. The subspace W_j is such that $V_{j+1} = V_j \oplus W_j$. That is, W_j is the orthogonal compliment of V_j and together they construct a new space V_{j+1}. We can thus decompose the space $V_j = V_{j-1} \oplus W_{j-1} = V_{j-2} \oplus W_{j-2} \oplus W_{j-1} = ... = V_i \oplus W_i \oplus ... \oplus W_{j-1}$. This leads to the decomposition $f_J = f_0 + \sum_{j=0}^{J-1} w_j(x)$, for $w_j \in W_j, 0 \leq j \leq J-1$.

Thus, given a function $f_j(x) = \sum_k a_k^j (2^j x - k)$, we can decompose it as $f_j(x) = f_{j-1}(x) + w_{j-1}(x)$, where

$$f_{j-1}(x) = \sum_k a_k^{j-1} \phi(2^{j-1} x - k) \in V_{j-1}$$

$$w_{j-1}(x) = \sum_k b_k^{j-1} \psi(2^{j-1} x - k) \in W_{j-1}$$

,with coefficients

$$a_k^{j-1} = \frac{1}{2}(a_{2k}^j + a_{2k+1}^j)$$

and

$$b_k^{j-1} = \frac{1}{2}(a_{2k}^j - a_{2k+1}^j)$$

Informally, this describes a sequence of subspaces, each representing a different level of "detail" of the function. These subspaces build iteratively, using a set of scaling functions and a set of orthogonal basis functions (the wavelets) to define the subspaces and also decompose the original function. This is called the *multi-resolution* approach, and is what makes wavelets an attractive mathematical tool for spectral analysis.

F. The Wavelet CNN

With the above knowledge, we can construct a mathematical model for the WCNN. The abstraction lies in the observation that a single wavelet transform $w(x) = y$, where $x \in \mathbb{R}^{n \times n}$ (as in the case of an image) and $y \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2} \times 4}$ is equivalent to a convolution with a single input channel and 4 output channels, followed by a downsampling by 2. We can thus replace a normal convolution with the wavelet transform and then apply convolutions to that output, while *also successively decomposing the low-pass low-pass output of the previous transform*. This is the key insight into the WCNN and is what motivates this experiment.

III. Methodology

This paper implemented two CNNs, a WCNN and a Baseline CNN, to compare performance in image classification. Two Python libraries were used: Pytorch [Pas+17] and Pywavelets [Lee+19]

A. Architecture and Design

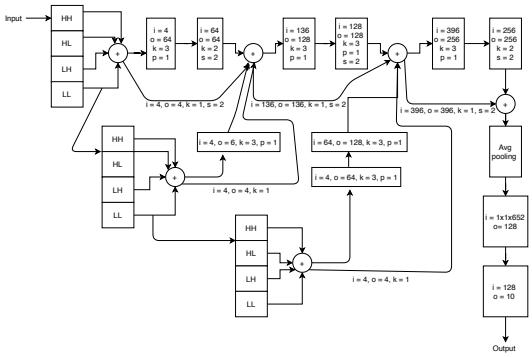

Fig. 1. The WCNN Architecture

We shall first discuss the implementation of the WCNN and then compare that to the Baseline.

Both networks were constructed to take full advantage of the multi-resolution approach to convolutions. The network begins with a single wavelet transform on the input data, which we shall denote as $w_1(x) = y$, where $x \in \mathbb{R}^{n \times n}$ and $y \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2} \times 4}$. The output y is then passed in as input to the main layers of convolutions $c_i(\cdot)$ $i = 1, .., 6$. The convolution layers alternate such that when i is odd, $c_i(\cdot) : \mathbb{R}^{n \times n \times l} \rightarrow \mathbb{R}^{n \times n \times m}$ and when i is even, $c_i(\cdot) : \mathbb{R}^{n \times n \times m} \rightarrow \mathbb{R}^{\frac{n}{2} \times \frac{n}{2} \times m}$. That is, the even convolution layers down-sample the input data whereas the odd convolution layers do not down-sample. The down-sampling convolution layers imitate the down-sampling by 2 of the wavelet transform.

Additionally, both networks contain a series of branching convolutions which are applied after additional wavelet transforms. Specifically, the down-sampled output $y_1 \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2} \times 1}$ (that is, only the low-pass low-pass filtered output) from $w_1(\cdot)$ is passed in as input to the second wavelet transform $w_2(y_1) = y'$, where $y' \in \mathbb{R}^{\frac{n}{4} \times \frac{n}{4} \times 4}$. A convolution filter $d(\cdot) : \mathbb{R}^{\frac{n}{4} \times \frac{n}{4} \times 4} \rightarrow \mathbb{R}^{\frac{n}{4} \times \frac{n}{4} \times m}$ is applied to y'. This output is then concatenated channel-wise (concatenated in the 3^{rd} dimension) with the output from $c_2(\cdot)$ and then passed into $c_3(\cdot)$.

Similarly, the down-sampled output $y'_1 \in \mathbb{R}^{\frac{n}{4} \times \frac{n}{4} \times 1}$ from $w_2(\cdot)$ is passed in as input to the third wavelet transform $w_3(y'_1) = y''$, where $y'' \in \mathbb{R}^{\frac{n}{8} \times \frac{n}{8} \times 4}$. Two convolution filters $f_1(\cdot) : \mathbb{R}^{\frac{n}{8} \times \frac{n}{8} \times 4} \rightarrow \mathbb{R}^{\frac{n}{8} \times \frac{n}{8} \times l}$ and $f_2(\cdot) : \mathbb{R}^{\frac{n}{8} \times \frac{n}{8} \times l} \rightarrow \mathbb{R}^{\frac{n}{8} \times \frac{n}{8} \times m}$ are applied in succession to y''. The output from that convolution is also concatenated channel-wise to the output of $c_4(\cdot)$ and passed in as input to $c_5(\cdot)$.

Finally, "skip" connections are used, wherein output from one layer is concatenated to the output of convolutions one or two layers ahead. This is done to assure that the gradient of the error does not vanish as it propagates deeper into the network.

The output of $c_6(\cdot)$ is then passed into a global max pooling operation described above in section **B. Pooling** above. The pooled data is then fed into a two layer feed-forward neural network and finally passed into a softmax activation function to give a probability distribution over the possible classes of the input.

The WCNN was designed to be compatible with the Haar and Daubechies-1 wavelets. These wavelets are gauranteed to map inputs of size $n \rightarrow \frac{n}{2}$ (or $n \rightarrow \frac{n+1}{2}$ when n is odd). Other wavelets, such as the Mexican Hat wavelet or other Daubechies-1 wavelets do not generate a similar mapping and so would require specific changes to the architecture to match their mappings. As such, they were left out of this experiment.

The Baseline CNN was designed to mirror the WCNN as closely as possible, but without the wavelet transforms. The network architecture is thus exactly the same but with convolutions $w'_i(\cdot) : \mathbb{R}^{\frac{n}{2^{i-1}} \times \frac{n}{2^{i-1}} \times 1} \rightarrow \mathbb{R}^{\frac{n}{2^i} \times \frac{n}{2^i} \times 1}$ for $i = 1, 2, 3$ replacing the wavelet transforms.

B. Datasets

The dataset chosen for this experiment was the CalTech101[2] image dataset consisting of over $9,000$ images belonging to 101 categories. Each category is of an everyday image. The images are roughly 200×300 and so were upsampled to 300×300. Additionally, only the 10 largest classes by image count were chosen for training and testing, yielding a total dataset size of $3,739$ images. This was done to increase training time and to reduce variability in classification, due to the difficulty in training on small numbers of training examples.

C. *Training Procedure*

During training, the dataset was randomly partitioned into test and train (with the same test and train being used throughout the entire training and testing process). The test data was ignored during the training process and the the training data was split using k-fold cross validation for hyperparameter selection. In k-fold cross validation the training data is split into k groups and each group is chosen once to be the validation data (essentially a testing dataset that isn't the test dataset) and the other groups are used as the training data. This reduces over fitting during training since each section of the entire training dataset can be trained on and the resutls compared. For this experiment, a split of 2 was used.

Hereafter, "training data" refers to the train dataset without the validation data. Each group is chosen once as the validation data and the remaining data is trained on for a number of "epochs". A single epoch denotes one run through of the training data and one run through of the validation data. During a run through of the training data, the network learns from its errors, while it does not learn during a run through of the validation data. In our experimentation, a single epoch was found to be insufficient to get accurate results. As a result, 20 epochs was chosen a it was found to be sufficient for the error to hit a minimum and due to resource constraints.

Additionally, it was found that a learning rate of 0.00001 was too slow to reach convergence to a minimum of the loss function and that a learning rate of 0.001 could cause the error to get stuck in a local minimum and thus halt the learning process. Therefore, a learning rate of 0.0001 was used.

WCNNs with both wavelets, Daubechies-1 and Haar, were subject to this training process, as was the Baseline CNN. It was found that Daubechies-1 performed better and was chosen to run on the test dataset. The Baseline CNN was run on the same hyperparameters for the test set as the WCNN.

IV. Results

In this section, we report on the results of the training and testing procedure. Training and validation losses are reported for each WCNN model (Haar and Daubechies-1) as well as the Baseline CNN. The final testing accuracies are reported at the end of the section.

A. *WCNN- Haar Wavelet*

Both of these figures show that the loss roughly bottomed out for both training and the validation sets around epoch 17 to 20. This suggests that the minimum value of the loss was attained.

B. *WCNN - Daubechies-1 Wavelet*

The training loss for the Daubechies-1 WCNN is fairly consistent between splits and bottoms out at around epoch

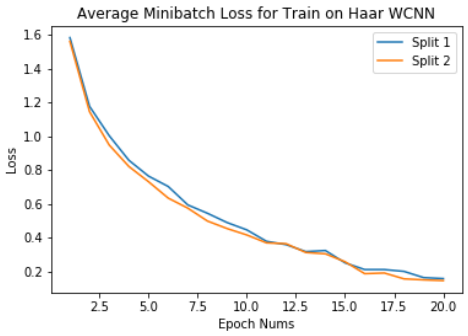

Fig. 2. Training Loss for Haar Wavelet

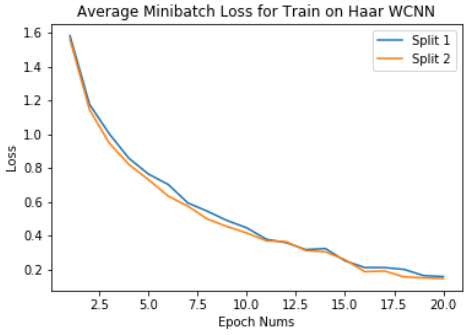

Fig. 3. Training Loss for Haar Wavelet

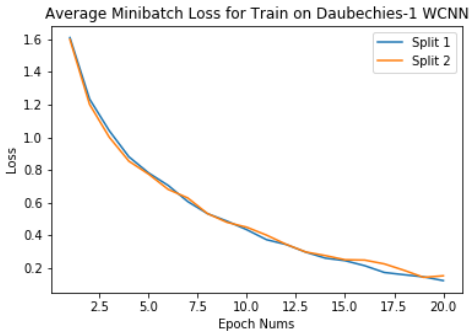

Fig. 4. Training Loss for Daubechies-1 Wavelet

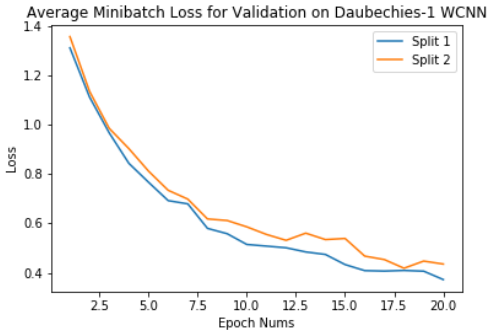

Fig. 5. Validation Loss for Daubechies-1 Wavelet

19. However, the loss for the 2^{nd} validation split is much higher and does not reach the same level as the loss for the 1^{st} split. Additionally, the validation loss is much higher for this WCNN is much higher- about 0.2 higher- than in the case of the Haar WCNN.

C. *Baseline CNN*

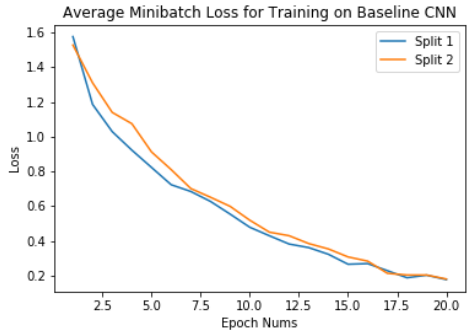

Fig. 6. Training Loss for Baseline CNN

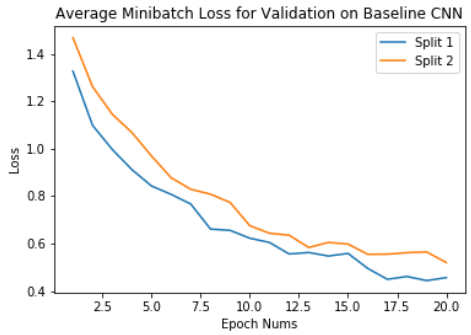

Fig. 7. Validation Loss for Baseline CNN

The training loss for the Baseline CNN is roughly consistent with the previous networks. We see a slight increase in the training loss in split 2 during the initial epochs, but this converges to the same point as in split 1. The validation is similar to that of Daubechies-1- split 2 had a much higher loss than split 1 and neither got below 0.4.

D. *Test Accuracy*

The Daubechies-1 WCNN was chosen to run on the test data and the Baseline CNN was run to compare. We see that both networks reach roughly the same accuracy of 90%. The average accuracy for the WCNN was slightly more jagged, fluctuating a but more during the middle epochs before settling around 90%. The Baseline CNN featured a slightly smoother climb to its highest accuracy. Additionally, the Baseline CNN appears as if the maximum accuracy was achieved at epoch 17, featuring a trend toward lower accuracies. This suggests that the Baseline CNN takes less time to train.

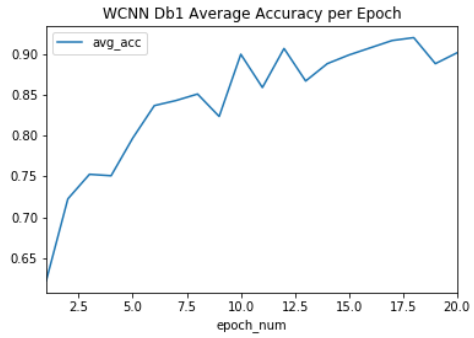

Fig. 8. Test Accuracy for Daubechies-1 WCNN

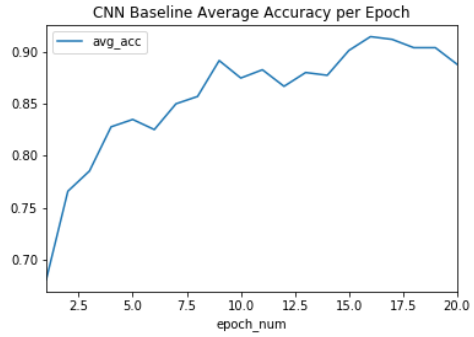

Fig. 9. Test Accuracy for Baseline CNN

V. Discussion

As reported in the Results section, both networks achieved roughly the same accuracies 90%, but the Baseline CNN reached the peak accuracy earlier. This is promising for the use of WCNNs. While the Baseline CNN performed slightly better, there could be room for improvements that were not possible in this investigation.

There are a number of reasons that the Baseline CNN achieved peak accuracy slightly earlier. It could be possible that the wavelets used in the WCNN were not suited for classification or that this dataset is not suitable for wavelet analysis. There is also the possibility that the Baseline CNN is just a better network for this task.

The performance of the WCNN is very promising and it is certainly possible that this architecture could be improved upon. The most obvious solution is to try different wavelet transforms. Other flavors of Daubechies, Mexican Hat wavelet, and the many other wavelet transforms in existence could prove very fruitful in this regard. The current WCNN model proposed here could also be improved by adding additional layers, training for more epochs, or adding additional wavelet transforms.

Thus our investigation of this problem concludes that Wavelet Convolutional Neural Networks are a promising model for deep learning, and have at least comparable perfor-

mance to traditional CNNs. More experimentation is certainly required to find an optimal architecture.

REFERENCES

[AH13] Kendall E. Atkinson and Weimin Han. *Theoretical numerical analysis: a functional analysis framework*. World Publishing Corporation, 2013.

[Pas+17] Adam Paszke et al. "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*. 2017.

[FTH18] Shin Fujieda, Kohei Takayama, and Toshiya Hachisuka. "Wavelet Convolutional Neural Networks". In: *CoRR* abs/1805.08620 (2018). arXiv: 1805.08620. URL: http://arxiv.org/abs/1805.08620.

[Lee+19] Gregory Lee et al. "PyWavelets: A Python package for wavelet analysis". In: *Journal of Open Source Software* 4.36 (2019), p. 1237. DOI: 10.21105/joss.01237.

[FFP] Li Fei-Fei, R. Fergus, and P. Perona. "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories". In: *2004 Conference on Computer Vision and Pattern Recognition Workshop* (). DOI: 10.1109/cvpr.2004.383.