Adaptive wavelet pooling for convolutional neural networks

Moritz Wolter
Fraunhofer Center for Machine Learning
and Fraunhofer SCAI
Institute for Computer Science
University of Bonn
wolter@cs.uni-bonn.de

Jochen Garcke
Institute for Numerical Simulation
University of Bonn
Fraunhofer Center for Machine Learning
and Fraunhofer SCAI
garcke@ins.uni-bonn.de

Abstract

Convolutional neural networks (CNN)s have become the go-to choice for most image and video processing tasks. Most CNN architectures rely on pooling layers to reduce the resolution along spatial dimensions. The reduction allows subsequent deep convolution layers to operate with greater efficiency. This paper introduces adaptive wavelet pooling layers, which employ fast wavelet transforms (FWT) to reduce the feature resolution. The FWT decomposes the input features into multiple scales reducing the feature dimensions by removing the fine-scale subbands. Our approach adds extra flexibility through wavelet-basis function optimization and coefficient weighting at different scales. The adaptive wavelet layers integrate directly into well-known CNNs like the LeNet, Alexnet, or Densenet architectures. Using these networks, we validate our approach and find competitive performance on the MNIST, CIFAR-10, and SVHN (street view house numbers) data-sets.

1 Introduction

The machine learning community has largely turned to convolutional networks (CNNs) for image [He et al., 2016], audio [Nagrani et al., 2019] and video processing [Carreira and Zisserman, 2017] tasks. Within CNNs, pooling layers boost computational efficiency and introduce translation invariance.

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

Pooling operations replace the internal network representation with a summary statistic of the features at that point [Goodfellow et al., 2016]. Pooling operations are important yet imperfect because they often rely only on simple max or mean operations, or their mixture, in a relatively small neighborhood. Adaptive pooling attempts to improve classic pooling approaches by introducing learned parameters within the pooling layer. [Tsai et al., 2015] found adaptive pooling to be beneficial on image segmentation tasks, while [McFee et al., 2018] made similar observations for audio processing.

Recently more sophisticated pooling strategies have been introduced. These approaches utilize basis representations in the frequency domain [Rippel et al., 2015], or in time (spatial) and frequency [Williams and Li, 2018], a forward transform handles the conversion. Pooling is implemented by truncating those components a-priori deemed least important in that basis representation and then transferring the features back into the original space. Note that pooling by truncation in Fourier- or wavelet-basis representations can be considered a form of regularization by projection [Natterer, 1977, Engl et al., 1996]. [Zeiler and Fergus, 2013] presented stochastic pooling as an efficient regularizer.

Previous wavelet-based [Bruna and Mallat, 2013] layer and pooling architectures mostly utilized static hand-crafted wavelets. Optimizable wavelet basis representations have been designed previously for network compression [Wolter et al., 2020] and graph processing networks [Rustamov and Guibas, 2013]. From a pooling point of view, wavelets are an approach that can more accurately represent the feature contents with fewer artifacts than nearest-neighbor interpolation methods such as max- or mean- pooling [Williams and Li, 2018].

To the best of our knowledge, we are the first to propose wavelet (time and frequency domain) based adap-

tive pooling. In this paper, we make the following contributions:

- We introduce adaptive- and scaled-wavelet pooling as an alternative to spectral- [Rippel et al., 2015] and static-wavelet pooling [Williams and Li, 2018].
- We propose an improved cost function for wavelet optimization based on the alias cancellation and perfect reconstruction conditions.
- We show that adaptive and scaled wavelet pooling performs competitively in convolutional machine learning architectures on the MNIST, CIFAR-10, and SVHN data sets.

To aid with reproducing this work, source code for all models and our fast wavelet transformation implementation is available at `https://github.com/Fraunhofer-SCAI/wavelet_pooling` .

2 Related work

Alternating convolutional and pooling layers followed by one or multiple fully connected layers are the building blocks of most modern neural networks used for object recognition. It is therefore not surprising that the machine learning literature has long been studying pooling operations.
Early investigations explored max-pooling and non-linear subsampling [Scherer et al., 2010]. More recent work proposed subsampling by exclusively using strides in convolutions [Springenberg et al., 2015] and pooling for graph convolutional neural networks [Porrello et al., 2019]. Next, we highlight the regularizing, adaptive, and Fourier or Wavelet-based pooling approaches, which are of particular relevance for this paper.

2.1 Pooling and regularization

[Zeiler and Fergus, 2013] introduced stochastic pooling. The stochastic approach randomly picks the activation in each pooling neighborhood. Like dropout, these layers act as a regularizer. A similar idea has appeared in [Malinowski and Fritz, 2013] which explored random and learned choices of the pooling region.

2.2 Adaptive pooling

Learned or adaptive pooling layers seek to improve performance by adding extra flexibility. For semantic-segmentation adaptive region pooling [Tsai et al., 2015] has been proposed. Working on a weakly labeled sound event detection task [McFee et al., 2018] propose an adaptive pooling operator. Their approach interpolates between min-, max-, and average-pooling features. [Gulcehre et al., 2014] finds that learned norm-pooling can be seen as a generalization of average, root mean square, and max-pooling, [Liu et al., 2017] found this approach helpful for video processing tasks. [Gopinath et al., 2019] devised an adaptive pooling approach for graph convolutional neural networks and found improved performance on brain surface analysis tasks.

2.3 Fourier and wavelet domain pooling

[Rippel et al., 2015] proposes to learn convolution filter weights in the frequency domain and uses the Fast Fourier Transform for dimensionality reduction by low-pass filtering the frequency domain coefficients. Alternatively [Williams and Li, 2018] found the separable fast wavelet transform (FWT) useful for feature compression. The FWT obtains a multiscale analysis recursively. The approach computes a two-scale analysis wavelet decomposition and discards the first, fine-scale resolution level. The synthesis transform only uses the second, coarse-scale level coefficients to construct the reduced representation. [Williams and Li, 2018] proposes to use a fixed Haar wavelet basis; we consequently refer to this approach as wavelet pooling.

The papers closest to ours are [Williams and Li, 2018] and [Wolter et al., 2020]. [Wolter et al., 2020] proposes to use flexible wavelets for network compression and formulates a cost function to optimize the wavelets. We build on both approaches for feature pooling, add coefficient weights, and devise an improved broader loss function.

3 Methods

Our pooling approach relies on the multiscale representation we obtain from the fast wavelet transform (FWT). The recurrent evaluation of the FWT produces new scale coefficients each time it runs. We refer to the number of runs as levels.
This section discusses the FWT and the properties our wavelet filters must have, how to turn these into a cost-function, and the rescaling of wavelet coefficients.

3.1 The fast wavelet transform

The fast wavelet transform constitutes a change of representation and a form of multiscale analysis. It expresses the input data points in terms of a wavelet filter pair by computing [Strang and Nguyen, 1996]:

$$\mathbf{b} = \mathbf{A}\mathbf{x}, \tag{1}$$

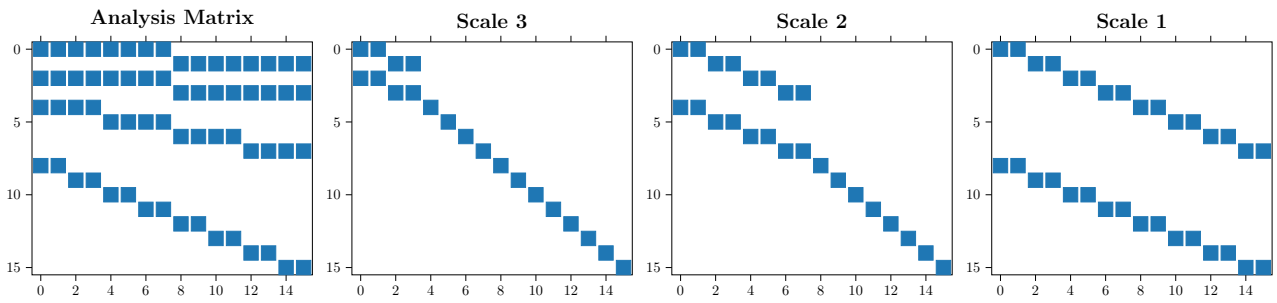

Figure 1: Structure of the Haar analysis fast wavelet transformation matrix on the left followed by the structures of the individual scale processing matrices. The full analysis matrix (left) is the product of multiple (here three) matrices. Each describes the FWT-operations at the current level. Two convolution matrices $\mathbf{H}_0$ and $\mathbf{H}_1$ are clearly visible in the three matrices on the left [Wolter, 2021].

the analysis matrix $\mathbf{A}$ is the product of multiple matrices, each decomposing the input signal $\mathbf{x}$ at an individual scale. Finally, multiplication of the total matrix $\mathbf{A}$ with the input-data yields the wavelet coefficients $\mathbf{b}$.

We show the structure of the individual matrices in figure 1. We observe a growing identity block matrix $\mathbf{I}$, as the FWT moves through the different scales. The filter matrix blocks move in steps of two. Each FWT step cuts the input length in half. The identity submatrices appear where the results from the previous steps have been stored. The reoccurring diagonals denote convolution operations with the analysis filter pair $\mathbf{h}_0$ and $\mathbf{h}_1$. Given the wavelet filter degree d, each filter has $N = 2d$ coefficients. The filters are arranged in vectors $\in \mathbb{R}^N$. The filter pair appears in the convolution matrices $\mathbf{H}_0$ and $\mathbf{H}_1$. Overall we observe the pattern [Strang and Nguyen, 1996],

$$\mathbf{A} = \ldots \left(\begin{array}{c|c} \begin{matrix}\mathbf{H}_0 \\ \mathbf{H}_1\end{matrix} & \\ \hline & \mathbf{I} \end{array}\right) \begin{pmatrix}\mathbf{H}_0 \\ \mathbf{H}_1\end{pmatrix}. \tag{2}$$

The first two FWT-matrices are shown. The dots $\ldots$ act as a placeholder for additional analysis matrices. All of which factor into the final analysis matrix.

We have seen how to construct the analysis matrix $\mathbf{A}$ and will now invert this process. Overall the inverse FWT (IFWT) can again be thought of as a linear operation,

$$\mathbf{S}\mathbf{b} = \mathbf{x}. \tag{3}$$

The synthesis matrix $\mathbf{S}$ is constructed using the synthesis filter pair $\mathbf{f}_0, \mathbf{f}_1$, where structurally the synthesis matrices are transposed in comparison to their analysis counterparts. We show the multi-scale reconstruction matrices in figure 2. Given the transposed convolution matrices $\mathbf{F}_0$ and $\mathbf{F}_1$, $\mathbf{S}$ is constructed by [Strang and Nguyen, 1996]

$$\mathbf{S} = \begin{pmatrix}\mathbf{F}_0 & \mathbf{F}_1\end{pmatrix} \left(\begin{array}{cc|c} \multicolumn{2}{c|}{\multirow{2}{*}{$\mathbf{F}_0 \quad \mathbf{F}_1$}} & \\ & & \\ \hline & & \mathbf{I} \end{array}\right) \ldots. \tag{4}$$

In order to guarantee invertibility we must have $\mathbf{S}\mathbf{A} = \mathbf{I}$. To enforce it, conditions on the filter pairs $\mathbf{h}_0, \mathbf{h}_1$ as well as $\mathbf{f}_0, \mathbf{f}_1$, which make up the convolution and transposed convolution matrices, are required. In summary, computation of the fast wavelet transform relies on convolution pairs, which recursively build on each other.

3.2 The two-dimensional wavelet-transform

The two-dimensional wavelet transform is based on the same principles as the one-dimensional case. Separable and non-separable two-dimensional wavelet transforms exist [Jensen and la Cour-Harbo, 2001]. Separable two-dimensional transforms work with two one dimensional transforms. Non-separable transforms use a single two-dimensional FWT. Separable transforms have previously been found to problematically single out axis-aligned and diagonal structures [Jensen and la Cour-Harbo, 2001]. We, therefore, choose to work with two-dimensional wavelet transforms. We now have to handle the extra dimension and require two-dimensional filter quadruplets instead of pairs.

Given the one-dimensional filter pairs $\mathbf{f}_0, \mathbf{f}_1$ and $\mathbf{h}_0, \mathbf{h}_1$, their two-dimensional counterparts are computed using outer products [Vyas et al., 2018]. An outer product of two column vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as $\mathbf{a}\mathbf{b}^T$. The required four filters are computed using the outer products of all four combinations,

$$\mathbf{f}_{ll} = \mathbf{f}_0\mathbf{f}_0^T, \qquad \mathbf{f}_{lh} = \mathbf{f}_0\mathbf{f}_1^T, \tag{5}$$

$$\mathbf{f}_{hl} = \mathbf{f}_1\mathbf{f}_0^T, \qquad \mathbf{f}_{hh} = \mathbf{f}_1\mathbf{f}_1^T. \tag{6}$$

This filter set $\{\mathbf{f}_{ll}, \mathbf{f}_{lh}, \mathbf{f}_{hl}, \mathbf{f}_{hh}\} \in \mathfrak{f}_k$ is used in all forward convolutions at every level.

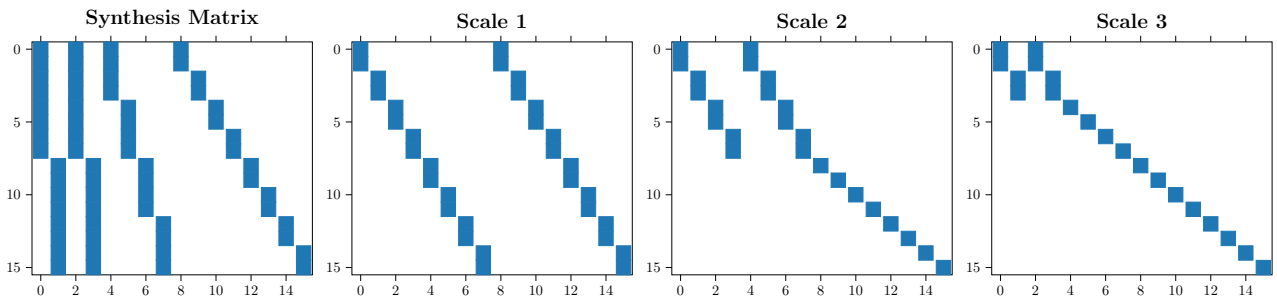

Figure 2: Structures of the Haar synthesis fast wavelet transformation matrices for three scales (right) as well as the complete inverse matrix (left). We observe that the synthesis matrix is the product of stacked transposed convolution matrices $\mathbf{F}_0$ and $\mathbf{F}_1$. In comparison to the forward transform matrices are reversed [Wolter, 2021].

In the non-seperated case, two-dimensional convolution $*$ is used to compute the wavelet coefficients. Denoting the filter with k and using i for the scale, convolution with each of the four filters

$$\mathbf{x}_i * \mathbf{f}_k = \mathbf{k}_i \tag{7}$$

yields the four subbands with $k \in [ll, lh, hl, hh]$. The last three coefficients are stored, and $\mathbf{ll}_i$ is used to compute the next scale. In other words, the input signal $\mathbf{x}_i$ must be $\mathbf{ll}_{i-1}$ after the first level where $i > 1$.

In the inverse or synthesis case, the two-dimensional filters are once more constructed using outer products. Given the four subbands at the higher level, $\mathbf{ll}_{i-1}$ is reconstructed using a transposed two-dimensional convolution. The reconstruction completes the three stored subband tensors on the level below, which we keep during the forward pass. While inversely traversing the list of sub-tensors stored during the analysis transform, transposed convolutions successively reconstruct the original input, combining the newly computed $\mathbf{ll}_{i-1}$ with the stored submatrices at every step.

3.3 Wavelet pooling

The fast wavelet transform turns into a pooling method, when the first level, fine-scale coefficients are discarded [Williams and Li, 2018]. The full analysis two-dimensional transform is computed for two or more scales. The synthesis transform, however, is computed without using the subbands at the first scale, which cuts the resolution in half [Williams and Li, 2018]. We are comparing pooling based on non-separable and separable transforms [Williams and Li, 2018]. Separable transforms treat the rows first and afterwards, independently, the columns. Two-dimensional wavelet transforms us a single 2D-convolution, see equations (5)-(7).

Without re-weighting, the inverse transform recombines higher level sub-tensors into the original input. In Section 3.4 we introduce sub-tensor weights, which make multiscale wavelet pooling meaningful.

3.4 Scaled wavelet pooling

To further benefit from multiscale analysis transforms, we introduce scaling weights for the pooling operation. Our weights rescale the wavelet coefficient subbands at higher levels. The idea is to allow our scaled wavelet pooling layers to in- or decrease the contribution of a particular scale to the pooled result.

The forward two-dimensional wavelet transform produces the coefficient quadruples $\mathbf{ll}_i, \mathbf{lh}_i, \mathbf{hl}_i, \mathbf{hh}_i$ at each scale i. It suffices to store the $\mathbf{lh}, \mathbf{hl}, \mathbf{hh}$ submatrices at each level to fully reconstruct the signal, since the low-low $\mathbf{ll}_i$ coefficients serve as input to compute the next quadruple at $i + 1$.

We introduce three scale coefficient parameters $w_{lh,i}, w_{hl,i}, w_{hh,i} \in \mathbb{R}^1 > 0$, at all but the last scale, where we additionally use a single $w_{ll} > 0$. The learned parameters are multiplied with the submatrices. For example, in the final low-low case, the inverse transform would work with $w_{ll} \cdot \mathbf{ll}$. With I indicating the total number of scales and dots denoting intermediate subbands, overall the weighted list,

$$[[w_{lh,0} \cdot \mathbf{lh}_0, \cdot w_{hl,0}\mathbf{hl}_0, w_{hh,0} \cdot \mathbf{hh}_0], \dots \tag{8}$$

$$[w_{ll,I} \cdot \mathbf{ll}_I, w_{lh,I} \cdot \mathbf{lh}_I, \cdot w_{hl,I}\mathbf{hl}_I, w_{hh,I} \cdot \mathbf{hh}_I]] \tag{9}$$

is fed into the inverse FWT to compute the pooled features. Our approach gives the pooling operation the opportunity to rescale features at various scales according to their importance.

3.5 Wavelet properties

The chosen wavelet basis representation impacts the pooling. We now aim to optimize the wavelet basis. We observe that for invertibility, the synthesis-transform must undo the analysis-transform. This requirement does not allow us to work with any filter.

The so-called alias cancellation and perfect reconstruction conditions must hold [Strang and Nguyen, 1996]. Filters that satisfy these conditions turn into wavelets. The analysis filter pair is called $\mathbf{h}_0$, $\mathbf{h}_1$, while one refers to the synthesis pair as $\mathbf{f}_0$, $\mathbf{f}_1$. Both conditions work on their z-transformed counterparts. For a complex number $z \in \mathbb{C}$, the transformed filters $H_p(z)$ and $F_p(z)$ are defined as, $p \in \{0, 1\}$

$$H_p(z) = \sum_{n\in\mathbb{Z}} \mathbf{h}_p(n) z^{-n},\ F_p(z) = \sum_{n\in\mathbb{Z}} \mathbf{f}_p(n) z^{-n}, \quad (10)$$

for both pairs. In the z-domain the alias cancellation condition is given by [Strang and Nguyen, 1996]

$$H_0(-z)F_0(z) + H_1(-z)F_1(z) = 0. \quad (11)$$

In filter design the alias cancellation condition is often solved by the alternating signs solution $F_0(z) = H_1(-z)$ and $F_1(z) = -H_0(z)$ [Strang and Nguyen, 1996]. For the perfect reconstruction condition to hold we must have [Strang and Nguyen, 1996],

$$H_0(z)F_0(z) + H_1(z)F_1(z) = 2z^c. \quad (12)$$

The equation requires the filter product-sum at the center z^c to be equal to two, and zero everywhere else. The power at the center is denoted as c.

3.6 The adaptive wavelet loss

It turns out that we do not have to compute the z-transform in order to evaluate both conditions. The time domain representations can be used, by exploiting the equivalence of polynomial multiplication and convolution [Wolter et al., 2020]. Instead of measuring the deviation from the alternating signs solution, we choose to work with convolutions for both our alias cancellation and perfect reconstruction losses. Based on (11) we now define for alias cancellation

$$\mathcal{L}_{ac}(\theta_w) = \sum_{k=0}^{N-1} \Big[([\mathbf{h}_0 \cdot (-1)^k] * \mathbf{f}_0)_k + ([\mathbf{h}_1 \cdot (-1)^k] * \mathbf{f}_1)_k \Big]^2. \quad (13)$$

For the perfect reconstruction loss we follow [Wolter et al., 2020] and use, based on (12)

$$\mathcal{L}_{pr}(\theta_w) = \sum_{k=0}^{N-1} \Big[(\mathbf{h}_0 * \mathbf{f}_0)_k + (\mathbf{h}_1 * \mathbf{f}_1)_k - 2 \cdot e_{\lfloor N/2 \rfloor} \Big]^2, \quad (14)$$

where $e_{\lfloor N/2 \rfloor}$ is the $\lfloor N/2 \rfloor$-th unit vector. And k the filter position as counted from left to right. Summing up both expression leads us to the overall wavelet loss

$$\mathcal{L}(\theta_w) = \mathcal{L}_{ac}(\theta_w) + \mathcal{L}_{pr}(\theta_w). \quad (15)$$

To allow joint optimization of the network and wavelet weights, this pooling loss function must be added to the task-dependent loss function that we want the optimizer to improve.

4 Experiments

All experiments use Pytorch [Paszke et al., 2017] and run on Nvidia Titan Xp cards, with 12GB memory.

Starting with a proof of concept experiment, we compute a forward 2d-FWT, set the first subband coefficients to zero, and compute the synthesis FWT. The resulting image appears in figure 3 on the left. Using the cost function defined in equation 15, we optimize zero-padded Haar wavelet filters to minimize the information loss resulting from not having the first level coefficients. The second image from the left in figure 3 shows the resulting reconstruction. Right next to it, figure 3 depicts the mean channel difference of both reconstructions. We observe that the new reconstruction is improved, especially at the edges, and shows fewer artifacts. Finally, the mean squared error versus the optimization steps appears on the very right of figure 3. We conclude that the filter optimization approach could help to improve wavelet pooling layers in principle.

Haar-wavelet initialization tends to get stuck at the Haar-Wavelet. In subsequent experiments, we choose to work with randomly initialized filter coefficients. The randomly initialized wavelet weights are pretrained to obtain sufficiently wavelet-like coefficients. These, in turn, ensure stable FWT and iFWT computations.

We repeat the proof of concept experiment with randomly initialized filters and show the converged filter in figure 4. The solution displays a pattern that solves the anti-aliasing condition. Some possible solutions require $F_0(z) = H_1(-z)$ and $F_1(z) = -H_0(-z)$. Substituting $(-z)$ produces a minus sign at odd powers in the coefficient polynomial. Multiplication with (-1) shifts the pattern to even powers. Whenever F_0 and H_1 share the same sign F_1 and H_0 do not and the other way around. Other patterns appear too. We picked an alternating sign example because it is well known, and other solutions to the convolutions in equations (13) and (14) are harder to recognize.

In the next sections, we will evaluate the different wavelet pooling approaches compared to max- and mean-pooling on several data sets. Here, working with standard CNN architectures, we compare the methods by replacing the local pooling layers. We work with first-degree wavelets. The wavelets in our CNN experiments, therefore, use two coefficients in all four filter

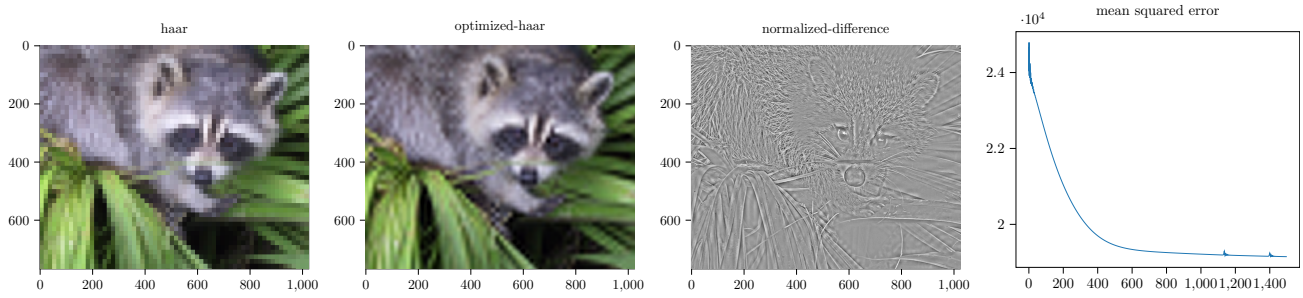

Figure 3: Wavelet optimization proof of concept. The first level coefficients are set to zero. Using the remaining coefficients the input image is reconstructed. We show the original Haar-wavelet and optimized Haar-wavelet reconstruction. To the right, the difference between the two reconstructions divided by the maximum of its largest absolute value for better visibility. On the very right, the mean squared error is plotted during optimization. We observe that our approach reduces the reconstruction error.

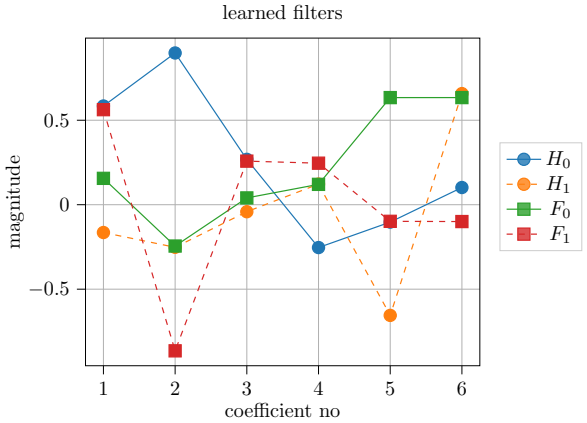

Figure 4: Converged wavelet using the sum of our loss formulation and mean squared error on the proof of concept experiment shown in figure 3 using a random initialization. H_0, H_1 denote the analysis, F_0, F_1 denote the synthesis filter pairs.

banks. Each wavelet-pooling layer has two scales and its own weights.

Wavelet and scaled wavelet layers use Haar wavelets. The scales are initially set to one. Adaptive wavelet layers work with random filter initializations. Initial values for the adaptive filters are drawn from the uniform distribution $\mathcal{U}[-1, 1]$. To ensure working FWT and IFWT transforms, we pre-train the adaptive filters for 250 steps to ensure that our forward- and backward-fast wavelet transform produce useful values from the start.

4.1 MNIST

The MNIST data set [LeCun et al., 1998] consists of sixty thousand train- and ten thousand-test images. All images have 28x28 pixels and belong to ten classes, one for each digit from zero to nine.

Table 1: Pooling method comparison on the MNIST data set. Our evaluation uses a LeNet5 like network. [Williams and Li, 2018] used a network based on Zeilers network and what we refer to as wavelet pooling. On MNIST adaptive wavelet pooling compares favorably to classic pooling methods. While scaled wavelet pooling performs competitively. For our experiments mean, and standard deviation over ten runs are shown.

Experiment	Accuracy [%]
adaptive wavelet	99.173 ± 0.037
scaled wavelet	99.075 ± 0.112
wavelet	99.073 ± 0.103
max	99.055 ± 0.140
adaptive max	98.809 ± 0.065
mean	99.059 ± 0.122
adaptive mean	99.064 ± 0.038
separable wavelet	99.071 ± 0.0967
wavelet [Williams and Li, 2018]	99.01

We solve this classification problem using a LeNet-like network with two convolution, two pooling, and three fully connected layers. We choose to optimize using stochastic gradient descent with a learning rate of 0.12, a momentum value of 0.6, and decay the learning rate by multiplication with 0.95 after every weight update. Training stops after 25 epochs.

Table 1 shows experimental results on MNIST. We observe the highest mean accuracy with a small standard deviation, for the adaptive wavelet approach. Followed by scaled wavelet, mean, and max pooling. In these experiments using adaptive mean oder max pooling did not help. To evaluate the impact of choosing a

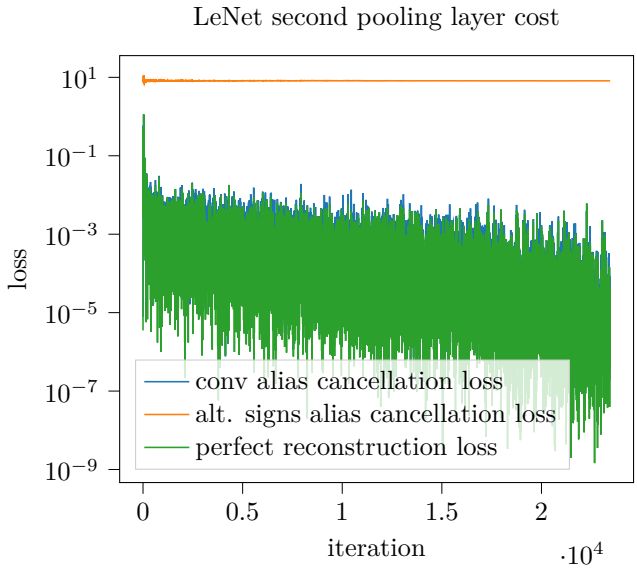

Figure 5: Logarithmically scaled wavelet loss during training in the second MNIST pooling layer. The green line shows the perfect reconstruction loss as defined in equation 14. The blue line the alias cancellation loss from equation 13. Both plots overlap most of the time. The yellow plot depicts the alias cancellation loss function proposed in [Wolter et al., 2020]. We observe that our wavelet has converged to a working solution that the previous formulation does not allow.

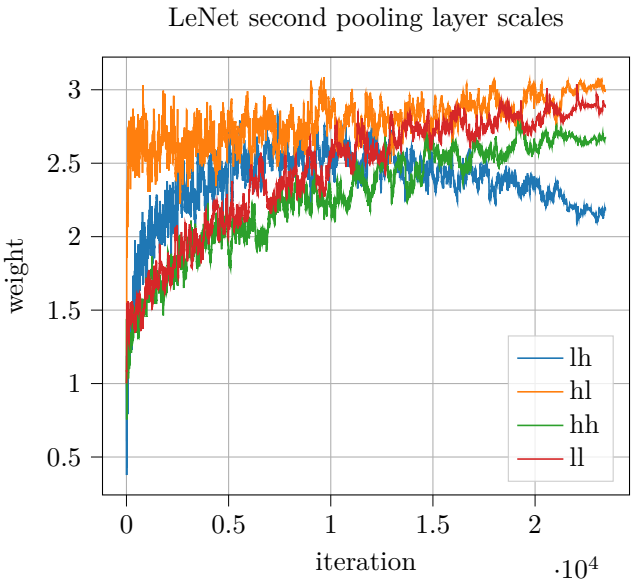

Figure 6: Wavelet scales weight during training in the second MNIST pooling layer. Originally initialized to have a weight of one, over time the optimizer adjusts the relative weight of the subbands with respect to each other.

seperable or non-separable wavelet transform, we implemented and explored the effect of a separable transform in the last row of Table 1, using a LeNet5 like architecture as we do in all other MNIST experiments. [Jensen and la Cour-Harbo, 2001] discourage this approach, but it does not reduce accuracy on MNIST.

Measurements of three wavelet loss formulations during training are shown in figure 5. The perfect reconstruction loss formulation defined in equation 14 is shown in green. Two different alias cancellation loss formulations are compared. The blue line shows the alias cancellation loss from equation 13. The yellow plot depicts a measure of the mean squared deviation from $F_0(z) = H_1(-z)$ and $F_1(z) = -H_0(-z)$,

$$L_{ac}(\theta_w) = \sum_{k=0}^{N} \left(f_{0,k} - (-1)^k h_{1,k}\right)^2 + \left(f_{1,k} + (-1)^k h_{0,k}\right)^2 \tag{16}$$

as proposed by [Wolter et al., 2020]. While this loss formulation leads to alias cancelling wavelet filters if optimized, we find it to be unnecessarily restrictive. Figure 5 illustrates this point. The loss shown by the blue line converges to a minimum which the formulation from equation 16 would not have allowed.

Figure 6 depicts the evolution of the wavelet coefficient submatrix weights during training. Since all weights were initialized to one we see that the optimizer moves these factors significantly as it adjusts the contribution of the individual subbands. Table 1 showed us that the scaling weights can lead to performance gains over fixed wavelet pooling, while basis optimization appears to be more efficient in this case.

4.2 CIFAR-10

The CIFAR-10 data set [Krizhevsky et al., 2009] consists of 60k 32 by 32 color images of these 50k are used for training and 10k for testing. The images show airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks. Networks trained on this data set have to recognize these correctly.

We start our comparison of pooling methods using an Alexnet [Krizhevsky, 2014] like network, which we modify to work on CIFAR-10. While moving from experiment to experiment, we always swap all pooling layers.

We train using stochastic gradient descent with a learning rate 0.06 and 0.6 momentum. The learning rate was divided by 10 after 150 and again after 250 epochs. The training process was stopped after 300 epochs.

Results are tabulated in Table 2, again our adaptive

Table 2: Pooling method comparison on the CIFAR-10 data set using an Alexnet backbone. The last row again shows the wavelet-pooling result from a Zeiler-like network. We find our adaptive pooling approach to perform well, while the scaled approach does slighly worse yets remains competitive in this case.

Experiment	Accuracy
adaptive wavelet	89.79%
scaled wavelet	88.94%
wavelet	89.01%
max	89.64%
mean	89.24%
separable wavelet	88.72%
wavelet [Williams and Li, 2018]	80.28%

Table 3: Pooling method comparison on the CIFAR-10 data set using a densenet backbone. We find that adaptive wavelet and max pooling work best, while scaled wavelet and fixed wavelet pooling outperform mean pooling.

Experiment	Accuracy
adaptive wavelet	94.41%
scaled wavelet	94.30%
wavelet	94.22%
max	94.37%
mean	94.17%

wavelet pooling performed competitively. Overall we find that the Alexnet backbone does much better than the Zeiler-like approach used jointly with dropout in [Williams and Li, 2018].

We repeat our experiments replacing the Alexnet with a densenet [Huang et al., 2017] architecture. For all experiments, we replace the pooling layers within all transition blocks and leave the final global pooling layer untouched. Optimization again used stochastic gradient descent with momentum. The learning rate was set to 0.05, again dividing it by ten after 150 and 250 epochs. Training stopped after 300 epochs. Weight decay helped in this case and was set to 10^{-4}.

Table 3 shows the best test set performance observed during training. We find that adaptive and max-pooling work best, while scaled and wavelet pooling do better than mean pooling.

Table 4: Pooling method comparison on the SVHN data set. Experiments again are based on a modified Alexnet architecture. We find our adaptive pooling method to perform on par with max pooling in this case.

Experiment	Accuracy
adaptive wavelet	94.87%
scaled wavelet	94.66%
wavelet	94.29%
max	94.87%
mean	94.42%
wavelet [Williams and Li, 2018]	91.10%

4.3 SVHN

The Street View House Numbers (SVHN) Dataset [Netzer et al., 2011] has 73k color training 26k testing images. We choose to use the CIFAR-10-like setting with 32 by 32 color images.

We again use an Alexnet [Krizhevsky, 2014] structure. Wavelet initialization and pretraining were left unchanged. For optimization, we choose stochastic gradient descent with a learning rate of 0.08 and no momentum.

The results shown in Table 4 confirm our previous observation that adaptive-wavelet and max-pooling perform best. The wavelet approach did not outperform mean pooling. This observation is in line with [Williams and Li, 2018], for their experiments with and without dropout. We note that scaled wavelet pooling outperformed mean pooling in this case and conclude that some form of learned flexibility is required within wavelet pooling layers.

5 Conclusion

We have introduced the usage of non-separable two-dimensional fast wavelet transforms for pooling layers in CNNs. Additionally, we proposed scaled and adaptive extensions. Here, an improved convolutional alias cancellation wavelet loss formulation permits wavelets, which an earlier employed restrictive alternating sign formulation did not allow. In our experiments, we have found that adaptive and scaled wavelet pooling delivers competitive results and adds extra flexibility to wavelet pooling.

Experimentally we found that the extra flexibility, added trough scaling or wavelet optimization leads to improved results in comparison to static wavelet pooling. In particular, we saw that fixed wavelet pool-

ing often failed to outperform mean pooling. With the exception of our CIFAR-10 experiments using the Alexnet backbone, we saw that adding the scaling of the coefficients generally leads to better performance in comparison to mean pooling.

Wavelet optimization had a bigger impact than rescaling the subbands. We speculate that this might be a result of the small input images we chose to use for computational reasons. These small images limit the number of meaningful scales the FWT can produce. Using more scales in future work might give the scaled wavelet pooling approach an opportunity to have more of an impact.

Our framework supports high-degree wavelets in principle, but these require additional padding to be invertible. Eventually, the extra padding impacts results. Adopting boundary filters [Strang and Nguyen, 1996] and wavelets instead of a padding-based approach may help here. We hope to investigate this in future work.

Future work could also train orthogonal wavelets. The orthogonal FWT matrices do not change the norm or the length of feature input vectors. Such orthogonality is of particular importance for the stability of recurrent neural networks, and could also help stabilize CNNs. Having orthogonal wavelet filters should make the FWT analysis matrix orthogonal $\mathbf{A}\mathbf{A}^T = \mathbf{I}$ and $\mathbf{A}^T\mathbf{A} = \mathbf{I}$ [Strang and Nguyen, 1996]. Translating these conditions back into the filters leads to $\mathbf{f}_0[k] = \mathbf{h}_0[-k]$ and $\mathbf{f}_1[k] = \mathbf{h}_1[-k]$ [Jensen and la Cour-Harbo, 2001].

This condition suggests an orthogonal loss function. Such a loss could, for example, measure the mean squared deviation from orthogonality, which might be interesting in future investigations.

6 Acknowledgments

We thank Helmut Harbrecht for insightful discussions and our anonymous reviewers for their feedback. Moritz would like to thank the University of Bonn for access to the Auersberg and Teufelskapelle clusters. This work was supported by the Fraunhofer Cluster of Excellence Cognitive Internet Technologies (CCIT).

References

[Bruna and Mallat, 2013] Bruna, J. and Mallat, S. (2013). Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886.

[Carreira and Zisserman, 2017] Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308.

[Engl et al., 1996] Engl, H., Hanke, M., and Neubauer, A. (1996). *Regularization of Inverse Problems*. Kluwer.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Gopinath et al., 2019] Gopinath, K., Desrosiers, C., and Lombaert, H. (2019). Adaptive graph convolution pooling for brain surface analysis. In *IPMI*.

[Gulcehre et al., 2014] Gulcehre, C., Cho, K., Pascanu, R., and Bengio, Y. (2014). Learned-norm pooling for deep feedforward and recurrent neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 530–546. Springer.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[Jensen and la Cour-Harbo, 2001] Jensen, A. and la Cour-Harbo, A. (2001). *Ripples in mathematics: the discrete wavelet transform*. Springer Science & Business Media.

[Krizhevsky, 2014] Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Liu et al., 2017] Liu, D., Zhou, Y., Sun, X., Zha, Z., and Zeng, W. (2017). Adaptive pooling in multi-instance learning for web video annotation. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 318–327.

[Malinowski and Fritz, 2013] Malinowski, M. and Fritz, M. (2013). Learnable pooling regions for image classification. *ICLR (workshop track)*.

[McFee et al., 2018] McFee, B., Salamon, J., and Bello, J. (2018). Adaptive pooling operators for weakly labeled sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26:2180–2193.

[Nagrani et al., 2019] Nagrani, A., Chung, J. S., Xie, W., and Zisserman, A. (2019). Voxceleb: Large-scale speaker verification in the wild. *Computer Science and Language*.

[Natterer, 1977] Natterer, F. (1977). Regularisierung schlecht gestellter Probleme durch Projektionsverfahren. *Numer. Math.*, 28:329–341.

[Netzer et al., 2011] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS Workshop Autodiff*.

[Porrello et al., 2019] Porrello, A., Abati, D., Calderara, S., and Cucchiara, R. (2019). Classifying signals on irregular domains via convolutional cluster pooling. pages 1388–1397. PMLR.

[Rippel et al., 2015] Rippel, O., Snoek, J., and Adams, R. P. (2015). Spectral representations for convolutional neural networks. In *NIPS*.

[Rustamov and Guibas, 2013] Rustamov, R. and Guibas, L. J. (2013). Wavelets on graphs via deep learning. In *Advances in neural information processing systems*, pages 998–1006.

[Scherer et al., 2010] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks – ICANN 2010*, pages 92–101. Springer Berlin Heidelberg.

[Springenberg et al., 2015] Springenberg, J., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*.

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

[Tsai et al., 2015] Tsai, Y.-H., Hamsici, O. C., and Yang, M.-H. (2015). Adaptive region pooling for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 731–739.

[Vyas et al., 2018] Vyas, A., Yu, S., and Paik, J. (2018). *Multiscale transforms with application to image processing*. Springer.

[Williams and Li, 2018] Williams, T. and Li, R. (2018). Wavelet pooling for convolutional neural networks. In *ICLR*.

[Wolter, 2021] Wolter, M. (2021). *Frequency Domain Methods in Recurrent Neural Networks for Sequential Data Processing*. PhD thesis, Institut für Informatik, Universität Bonn. submitted.

[Wolter et al., 2020] Wolter, M., Lin, S., and Yao, A. (2020). Neural network compression via learnable wavelet transforms. In *29th International Conference on Artificial Neural Networks*.

[Zeiler and Fergus, 2013] Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *ICLR 2013*. arXiv preprint arXiv:1301.3557.