ECBM 4040 Final Project - Spectral Representations for Convolutional Neural Networks

Aarshay Jain *(aj2713)*, Jared Samet *(jss2272)*, Alex Wainger *(atw2131)*

Abstract—This project is an attempt to implement the ideas and replicate the results of [Rippel, Snoek and Adams 2015]. The paper introduces two ideas to improve training of CNNs: spectral pooling, a novel approach to dimensionality reduction, and spectral filter parameterization, which represents convolutional filters in the spectral domain to help train the network in fewer epochs. The main technical challenge was to **not screw up the Fourier transforms, need to write this better**. Although we successfully implemented the paper's novel ideas in TensorFlow, we were unable to replicate the improved training results that the original authors report.

1 Introduction

The impressive achievements of Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision in recent years. CNNs have won the annual ImageNet competition every year since the introduction of AlexNet [paper reference] in 2012 and have rapidly come to dominate the research landscape for computer vision. As their name suggests, CNNs are deep feedforward neural networks mostly composed of successive layers that first compute the convolution (actually the cross-correlation) of the input image with a set of learned filters, then apply a nonlinear activation function to the result of the convolution, and finally reduce the dimensionality of the resulting activations through an operation such as max-pooling before repeating the process.

The Fourier transform, in both its continuous and discrete forms, maps convolution in the spatial domain to multiplication in the spectral domain and is often used internally at the hardware or software level by libraries such as NVIDIA's cuDNN that provide APIs to efficiently compute convolutions. Rippel, Snoek and Adams claim that the Discrete Fourier Transform (DFT) can be used not only to improve the efficiency of computing the convolution itself, but also to improve the training of CNNs in three distinct ways. Their first proposal is to represent the learned CNN filters as complex numbers in the frequency domain instead of as real numbers in the spatial domain. The second proposal is to replace the max-pooling layer that is typically found in CNNs with a spectral pooling layer, which shrinks the dimensionality with significantly less information loss for the same degree of parameter reduction. The third proposal is to use a modified version of the spectral pooling operation as a regularization technique similar to dropout.

Our objectives in this project were, first, to successfully implement the proposals of original paper in TensorFlow and implement a CNN employing these proposals, and, second, to replicate the improved training results reported by the authors. There were a number of challenges we encountered while implementing these novel proposals. The first was conceptual: making sure we properly understood the intricacies involved in shifting back and forth between the spatial and frequency domains for both the filters (in the case of spectral filter parameterization) and the images (for spectral pooling and frequency dropout). The second involved actually implementing the spectral pooling and frequency dropout operations. Although these operations are conceptually straightforward, implementing them correctly required careful attention to detail. The third challenge we faced involved backpropagating the gradient to the the convolution filters when they were parameterized as complex numbers in the frequency domain. The final challenge was faithfully replicating the details of the authors' multiple network architectures, which are described tersely with many different parameters.

2 Original Paper

2.1 Methodology

The original paper proposes spectral filter parameterization as an alternate formulation for the learned weight matrices of CNNs. In this formulation, which has the equivalent expressive power and capacity as the traditional formulation, the parameters are represented as the complex coefficients of the DFT of the traditional convolution filters. The authors claim that this formulation is more suitable for optimization than the traditional formulation and compare the convergence time (measured in optimization epochs, not in wall-clock time) of CNNs parameterized with spatial to CNNs parameterized with spectral filters.

The authors also propose spectral pooling as an alternative to max-pooling for reducing the dimensionality of the image as it is transformed throughout the neural network. They argue that this pooling operation preserves information better than max-pooling and quantify this by comparing the approximation loss of the two pooling techniques. They propose a network architecture consisting of alternating convolutional and spectral-pooling layers in which the only form regularization is frequency dropout and weight decay and achieve impressive accuracy rates on the CIFAR-10 and CIFAR-100 datasets by optimizing the hyperparameters of their architecture.

2.2 Key Results

The potential explanatory variables we included in our models were: The first key result of the original paper is that spectral pooling preserves information better than max pooling, which they demonstrate both visually, by showing a series of images whose dimensionality has been reduced by the two methods, and quantitatively, by computing the approximation loss, measured as the L2 norm of the difference of the pooled and unpooled images. The second key result is the error rates of 8.6% and 31.6%, respectively, that they achieve on the CIFAR-10 and CIFAR-100 datasets with their proposed CNN architecture. The third key result is the improved convergence time they achieve using the Adam optimizer. They measure this time by comparing the number of training epochs for the traditionally parameterized CNN and the spectrally parameterized CNN to converge to the same error rate.

3 METHODOLOGY

3.1 Implementation

We first implemented the paper's three proposals in TensorFlow. Since the concepts proposed were not immediately obvious upon reading the original paper, we have included a description here of precisely how these were implemented; we trust that these proposals accurately reflect the authors' intentions.

3.1.1 Spectral Filter Parameterization

In a traditionally parameterized CNN, a single convolutional layer transforms a three-dimensional tensor of shape $(C_{input}, H_{input}, W_{input})$ to a three-dimensional tensor of shape $(C_{output}, H_{output}, W_{output})$ by computing the cross-correlation of the three-dimensional volume at every x- and y-dimension of the input tensor with C_{output} separate filters. Here, C denotes the number of channels of the tensor, H denotes the height of the tensor and W denotes the width of the tensor. For each of the C_{output} filters, the learned parameters involved in this transformation consist of a set of real numbers that constitute the weights of the filter and a bias term. In a spectrally parameterized CNN, the convolution operation itself is identical, but the learned parameters are the complex coefficients of the DFT of the filter as opposed to the filter weights themselves. Given an input tensor and the complex numbers that constitute the spectral parameters for the filter, the output tensor is computed by first computing the inverse DFT of the filter and then convolving the resulting filter with the input tensor as usual.

We implemented this proposal in TensorFlow by writing a custom convolutional layer class where the Variables (TensorFlow's learned parameters) correspond to the complex DFT coefficients instead of the filter weights.

3.1.2 Spectral Pooling

In most CNN architectures, the operation of the network gradually transforms an input image with 3 channels (R, G, and B values) and a large height and weight dimension (e.g. 3 x 32 x 32 or 3 x 224 x 224) to an output "image" consisting of a large (e.g., 512) number of channels and a small (e.g. 5x5, 3x3, or even 1x1) number of height and weight "pixels". Different CNN architectures achieve this in different ways, but one common approach is to use max-pooling to downsample the image periodically; for example, after every convolutional layer, or after every other convolutional layer, max-pooling may be used to cut the height and weight of the image roughly in half. With spectral pooling, the dimensionality reduction is instead performed by:

- Computing the two-dimensional DFT of the input tensor for each input channel
- Truncating the resulting frequency matrix to the desired output dimensionality
- Computing the inverse DFT of the truncated frequency matrix to obtain the output tensor for the given channel

The truncation operation requires careful attention to detail in order to produce a real-valued output tensor. The DFT of a real-valued signal in the spatial domain obeys certain complex symmetries in the frequency domain; specifically, F[s, t] is the complex conjugate of F[-s, -t]; and, for a finite frequency matrix, if (s,t) and (-s,t) coincide modulo the dimension of the matrix, then F[s,t] must be real-valued; for example, the DC component, F[0,0] must be real valued, and for a 16 x 16 frequency matrix, F[8,8], F[0,8] and F[8,0] must also be real-valued. As a result, when an even-numbered output dimension is desired, the input frequency matrix cannot simply be truncated; the complex symmetries governing the original frequency matrix are not the same as the ones governing the truncated matrix. The paper refers to the need to handle these corner cases carefully but does not include the algorithm itself.

We implemented this proposal in TensorFlow by writing a custom spectral pooling function and layer class that always produces a truncated frequency matrix obeying the required complex symmetries. As with max-pooling, this layer has no learned parameters.

3.1.3 Frequency Dropout

This proposal is related to spectral pooling and is implemented in the same class, but serves a different purpose. The frequency matrix is computed for each input channel as before, but instead of being truncated to a smaller dimensionality, a brick-wall low-pass filter is instead applied: the higher frequencies are simply set to zero. The cutoff frequency for the filter is chosen randomly for each minibatch as a method of regularization. In the paper, this regularization is applied as part of the spectral pooling operation.

We implemented this proposal in TensorFlow by writing a function to perform the required filtering and incorporating the function into the spectral pooling layer.

3.2 Result Replication

We attempted to reproduce the three key results of the original paper. The impressive classification rates in the original paper required an intense hyperparameter optimization over six separate hyperparameter dimensions that would have required more computational resources than we had available for this project, but the authors report most of the results of their optimization. For the parameters where they reported optimal results, we used those and attempted to search over the others. For the other two key results, we followed their methodology as closely as possible.

3.2.1 *Information Preservation*

3.2.2 *Spectral Pooling and Frequency Dropout for Traditionally Parameterized CNNs*

We implemented a CNN class implementing the architecture specified in section 5.1 of the original paper. This architecture begins with M pairs of alternating layers of convolutional and spectral-pooling/frequency dropout layers, followed by a 1x1 convolutional layer with multiple filters, a 1x1 convolutional layer outputting the number of target classes, finally followed by a global averaging layer over which the softmax was computed for classification. The optimal hyperparameters which were not specified in the original paper were the number of layers and the weight decay rate. We performed hyperparameter search over a subsample of CIFAR-10 dataset to identify the optimal parameters. As we used the Adam optimizer instead of SGD with momentum, we added the initial learning rate and the dimensionality decay rate gamma to the set of hyperparameters over which we searched. We anneal the learning rate by 50% after every ten epochs. We randomly chose parameters on a log scale for the L2 norm and the learning rate and on a uniform scale for gamma and the number of layers M. We ran this search in parallel across multiple GPU instances and trained the network on the full datasets once the optimal hyperparameters had been identified. In addition to the random hyperparameter search, we also attempted to identify good hyperparameters by manually tuning.

3.2.3 *Optimization Convergence Speed*

To test the convergence speed of spectral parameterization, we created two new CNN architectures ? deep and generic. The deep architecture is defined in the paper as two 96-filter convolutional layers, a 3x3 stride 2 max pool, three 192-filter convolutional layers, a 3x3 stride 2 max pool, a 192-filter 1x1 convolutional layer, a 10-filter 1x1 convolutional layer, and a global averaging layer. The generic architecture is a more standard CNN, with a 96-filter convolutional layer, 3x3 stride 2 max pool, 192-filter convolutional layer, 3x3 stride 2 max pool, then fully connected layers with sizes of 1024 and 512. Both architectures end with a softmax to compute the probabilities of each sample belonging to each of the 10 classes. We also tested with the spectral pooling architecture described in section 3.2.2. For each of these three architectures, we trained 4 different configurations ? 3x3 kernels with spectrally parameterized conv weights, 3x3 kernels with standard conv weights, and the same thing with 5x5 kernels.

With all of our architectures defined, we used a standard training procedure, feeding mini-batches of 128 images through the network before computing and applying the gradient updates in an effort to minimize the loss (which was the sum of the cross entropy and l2 losses). We added vertical translations and horizontal flips after each epoch to increase the difficulty of training, as the paper indicates. For the models with spectral parameterization, we used the spectral convolution layer that was described in section 3.1.1. The paper mentioned that they used a Bayesian optimizer to find the best hyperparameter values, but they did not actually state what those values were for the generic and deep architectures, so we hand-tuned the networks on small samples of the training data before training on the larger dataset. Due to the number of models involved with this test and the computational intensity that spectral pooling and spectral parameterization add, we opted to train them on just one of the five batches of CIFAR10 data (i.e. 10,000 of the 50,000 samples in the training set), and even that took 12 hours on a Google Cloud GPU. The results of this test are discussed in section 5.1.3.

4 Implementation

4.1 Deep Learning Network

4.2 Software Design

5 Results

5.1 Project Results and Comparison

5.2 Insights Gained

6 Conclusion

7 Acknowledgements

References

[1] REPLACE THIS OF COURSE Almanie, Tahani, Rsha Mirza, and Elizabeth Lor. "Crime Prediction Based On Crime Types And Using Spatial And Temporal Criminal Hotspots." arXiv preprint arXiv:1508.02050 (2015).

8 Appendix

8.1 Individual Student Contributions