1. Abstract

We aim to investigate the impacts of different neural network architectures on the classification accuracy and time complexity of training for a classification task on the CIFAR-10 dataset. We will use a convolutional neural network model from Homework 4 with slight modifications to serve as a baseline comparison model for other architectures. The effectiveness of parallel streams and optimization methods, such as model parallelism, stochastic gradient descent (SGD), adaptive moment estimation (Adam), and their variants, will be examined. The research aims to provide insights into the effectiveness of parallel streams and optimization methods on less deep neural network architectures for image classification tasks on the CIFAR-10 dataset - <but the hope is that findings can also generalize to deeper architectures>. The goal is to contribute to the development of more efficient and accurate neural network architectures, not only allowing machine learning to be more accessible to individuals and smaller companies.

2. Introduction

With the advancement of deep learning and its successful application in various fields, the interest in designing efficient deep neural network architectures has grown significantly. One of the most widely used datasets for image classification tasks is the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes. In recent years, numerous neural network architectures have been proposed to achieve higher classification accuracy on this dataset. However, with the growing complexity of deep neural networks, training time has become a significant bottleneck.

This research paper aims to investigate the impacts of different architectures on the classification accuracy and time complexity of training for a classification task on the CIFAR-10 dataset. In particular, we will use a convolutional neural network model from our 'homework 4' assignment, with slight modifications towards classification accuracy, to use as a baseline comparison model for other architectures. We hypothesize that by exploiting parallelism in more shallow neural networks, we can achieve significant reductions in training time without sacrificing classification accuracy.

To test our hypothesis, we will compare the performance of our baseline model with other neural network architectures on the CIFAR-10 dataset. We will use various parallelization methods, including model parallelism, and compare their training time and classification accuracy. We will also investigate the effectiveness of different optimization methods, such as stochastic gradient descent (SGD), adaptive moment estimation (Adam), and their variants, on the parallel streams.

Overall, this research will provide insights into the effectiveness of parallel streams and optimization methods on deep neural network architectures for image classification tasks on the CIFAR-10 dataset.

3. Architectures

- **Architecture 1:** Vanilla - This architecture is a slight improvement from the architecture that we had covered in Homework 4. We made slight changes to the architecture, extracting more activation layers from each convolutional layer, and also using max pooling instead of average pooling after each activation function. The intention behind this architecture is to encourage the network to learn more types of patterns per convolutional layer, and also to extract the most distinct features from a pixel space. By increasing the number of activation patterns extracted per convolutional layer, the model is able to recognize a larger variety of features per layer. And by using max instead of average pooling, the model is able to learn the most distinct features in the output's pixel space per convolution.
- **Architecture 2:** Vanilla V2 (5 layer) (NET 4) - The idea here was to create a deeper, more optimized convolutional network that was still trainable in a reasonable amount of time. We arrived at this architecture (named Net4 for being our 4th initial testing iteration) due to its relatively high accuracy within a short amount of time (being avg epoch time * number of epochs). This architecture contains 5 convolutional layers, with batch normalization at every layer and max pooling after layers 1, 3, and 5. The layers follow a kernel size progression of 5, 3, 5, 3, 5 and a filter size progression of 16, 32, 64, 128, 256. Finally, the flattened output of the 5th layer is fed through 1 dropout layer and 3 linear layers to reach the output size of 10 (for each of the 10 classes within CIFAR-10). The idea here was to maximize accuracy by using common optimization techniques while still keeping depth minimal.
- **Architecture 3:** Net4 Residual -
    - The architecture here is the exact same as Net4, except that it concatenates the flattened, linearized results from the 3rd and 5th layers before feeding the data through the last 3 linear layers. We actually ran into problems with overfitting here, so we ended up adding dropout layers after each linear output from layers 3 and 5 (3 total). The concatenation from layer 3 is effectively a residual layer (thus the name), and this network draws inspiration from ResNet [1]. The thought here is that by adding layers 3 and 5 together to find the output, we increase the range of mappable features while only increasing computational complexity slightly. This was supported by the fact that this network ran into an overfitting problem quite quickly while the standard Net4 did not (thus the added dropout layers in the architecture that we tested more thoroughly).
- **Architecture 4:** Parallel - We took inspiration from GoogleNet [2] here. Initially when doing research, we found the inception modules to be quite interesting in how they used information from previous layers. Excited, we tried to implement a

form of GoogleNet to base our project off of, but found it to be extremely difficult to implement. Not only did 10 epochs take upwards of 3 hours to train, but it didn't even increase to 20% accuracy afterwards! Nonetheless, the idea of multiple layers working in the same "time" dimension was quite intriguing as a way to increase capabilities of a network, so we opted to design a smaller network with similar parallel convolving-like behaviors.

In order to do this, we designed the network to have multiple layers, A, B, and Pass in this case, that all feed directly, via concatenation, into 3 linear layers that condense into an output of 10 (referred to as the classifier). We know concatenation to be a viable method for this thanks to [1]. Beforehand, each of these layers processes at a different depth. First off, there are two convolutional layers, 1 and 2, that 2x2 max pool their results. *A* takes its input from layer 1, convolves it to 64 channels with a kernel size of 5, then max pool's its result. The flattened linearized output of 512 feeds directly into the classifier. *B* takes its input from layer 2, feeds it through 2 convolutions with 2x2 max pooling of kernel size 3, then feeds the flattened linearized output of 512 directly into the classifier. *Pass* convolves the *direct input* into an 8x8 channel (using filter size 8, stride 4, and padding 2) in effort to capture larger features in the image. It then 1x1 convolves this output to 64 channels, 2x2 max pools, and feeds a flattened linearized output of 128 nodes into the classifier. We also use batch normalization with each layer. The idea behind the kernel choices was to capture features at different levels of abstraction. *Pass* grabs much larger, more direct features, while *B* grabs more abstracted ones, and *A* stands somewhere in-between. Notably, this algorithm too proved to be quite powerful in terms of overfitting, so we opted to include dropout layers at each parallel block as well as after the first of the 3 linear layers in the classifier.

4. Experiments and Results

4.1 Adam vs SGD

After training Net4 for 15 epochs using PyTorch with a learning rate of 0.0005 and either the SGD or Adam optimizer, it was observed that the model had better performance when using SGD as the optimizer. The classification accuracy of the model using SGD as the optimizer was consistently higher, fluctuating between 76-79% over 3 testing trials, in comparison to the accuracy of Net4 using Adam optimization which demonstrated 74-75% classification accuracy over 3 testing trials.

We then experimented with the NetParallel architecture's optimization method to search for any improvements in classification accuracy. In the first 3 training trials, the Adam optimizer was used with a learning rate of 0.001 for 11 epochs and NetParallel demonstrated a 61% classification accuracy. In the second 3 training trials, the SGD optimizer with momentum and weight decay was used with a learning rate of 0.001 for

11 epochs. In this case, the accuracy of the network on the test images was 74%, indicating that the optimizer was able to converge to a better solution.

One possible explanation for the performance difference is that the SGD optimizer may be more effective at navigating the model's parameter space and finding a better solution than Adam. While Adam has shown to be effective in a variety of tasks and has become a popular optimizer choice, it may not be the optimal choice for every model and dataset. Additionally, since Adam includes an adaptive learning rate, it may converge slower than SGD, which could have a negative effect on Net4's training performance.

Furthermore, the learning rate used in this experiment may have also played a role in the performance difference. A learning rate of 0.0005 may be unideal for Adam given the dataset, which could cause the optimizer to overshoot the optimal solution and lead to slower convergence. In contrast, the same learning rate may have been more appropriate for SGD, leading to faster convergence and better performance.

Overall, these results suggest that different optimizers may perform differently on a given model and dataset, and that it's important to experiment with different optimization techniques to find the best option for a particular task.

4.2 Epoch Impact

A major part of our research in CNN architecture optimization is focused around reducing training time, in practice this usually implies training using minimal epochs and inherently reducing computational complexity. Thus we decided to research the number of epochs that maximized classification accuracy with respect to time among our different architectures. As a standardization method for comparing epochs across different architectures, we need a starting point when training other models for observing their performances using different epoch values.

With Net4 being the most simple architecture, we used Net4 to find a baseline for the number of epochs to train our other models. For Net4, increasing the number of epochs from 10 to 15 with a learning rate of 0.0005 and momentum of 0.9 led to a slight improvement in accuracy from 74% to 75% on the test set. This suggests that the network was still learning and benefiting from more training epochs at a slower learning rate. However, classification accuracy seemed to plateau between 15-20 epochs for Net4.

Using SGD as the optimization method for NetParallel, increasing the number of epochs from 11 to 16 with a learning rate of 0.001, momentum of 0.9, and weight decay of 1e-5 did not lead to any improvement in accuracy on the test set, plateauing at 74%, indicating that the network had already converged and additional training epochs did not improve its performance. However, when we used Adam as the optimization method for NetParallel and compared performance after training with 11, 16, and 21 epochs the

model demonstrated 61%, 64%, and plateaued at 66% classification accuracy respectively.

4.3 Comparing Architectures using SGD

To compare the performance of each architecture, we compared accuracy using SGD, batch size of 8, and cross entropy as our loss function. The progression of training parameters was as follows:

- First 11 epochs: lr=0.001, momentum=0.9, weight_decay=1e-5
- Next 5 epochs: lr=0.0005, momentum=0.9, weight_decay=1e-5
- Final 5 epochs: lr=0.0005, momentum=0.8, weight_decay=1e-5

The results for 4.3 are as follows:

Net4Residual:

- 0.406 -> 0.227 -> 0.088 (avg mini batch loss)
- 2.7 min (avg epoch time on James' computer)
- 76% -> 77% -> 79% (testing accuracy)
- 2.56 -> 1.78 -> 1.39 (cumulative accuracy per minute spent training)

NetParallel:

- 0.556 -> 0.316 -> 0.18
- 2.3 min (avg epoch time on James' computer)
- 70% -> 75% -> 76% (testing accuracy)
- 2.77 -> 2.04 -> 1.57 (cumulative accuracy per minute spent training)

Net4:

- 0.437 -> 0.208 -> 0.104
- 2.6 min (avg epoch time on James' computer)
- 75% -> 76% -> 77% (testing accuracy)
- 2.62 -> 1.83 -> 1.41 (cumulative accuracy per minute spent training)

NetVanilla:

- 0.167 -> 0.111 -> 0.098
- 1.0 min (avg epoch time on James' computer)
- 76% -> 75% -> 74% (testing accuracy)
- 6.91 -> 4.69 -> 3.52 (cumulative accuracy per minute spent training)

**Notably, NetParallel seemed to still be decreasing steadily in avg mini-batch loss even after 20 epochs. But we decided to stop at 20 in order to maintain consistency between network testing.

5. Discussion

We experimented with different optimization techniques demonstrating varying performances of the classification task on the Cifar-10 dataset, using numerous models in our experiment. We demonstrated the importance of experimenting with different optimization

techniques to find the best option for a particular task. For example, in the case of the Net4 model, we showcased that the SGD optimizer was more effective than Adam, and that the learning rate of optimizers has a significant impact on performance.

In pursuit of our goal of minimizing time complexity of convolutional architectures, we found that the number of epochs that maximized classification accuracy with respect to time varied among the different architectures. For instance, for Net4, increasing the number of epochs from 10 to 15 led to a slight improvement in accuracy, but classification accuracy seemed to plateau between 15-20 epochs, demonstrating an architecture that had a higher classification accuracy using lower epochs.

Overall, this study provides valuable insights into the impacts of different neural network architectures, parallel streams, and optimization methods on classification accuracy and training time complexity for a classification task on the CIFAR-10 dataset. The findings suggest that different optimization techniques may perform differently on a given model and dataset and that it is important to experiment with different optimization techniques to find the best option for a particular task. Additionally, the results suggest that the number of epochs that maximizes classification accuracy with respect to time varies among different architectures.

6. References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385.

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. arXiv preprint arXiv:1409.4842.