Empirical study of convolutional neural network architectures

Jaime Velazquez, A12865121, j5velazq@ucsd.edu

***Abstract*— Convolutional neural networks is a class of deep neural networks that is commonly applied to analyze visual data. I will be studying the performance of different architectures of convolutional neural networks (CNNs) to gauge what tweaks and tuning brings about the best result for our models. I will be constructing a custom designed CNN to compare to the Alexnet and GoogLeNet architectures. I will be training all models on the CIFAR-10 dataset.**

I. INTRODUCTION

A convolutional neural network is a deep learning algorithm that takes in an input image, assigns regions of importance through the use of filters that scan the image through the image. These regions of importance are then passed into a fully connected neural network that identifies and distinguishes the regions. Along the way, weights are kept and updated throughout the training process. In this paper, I will design my own architecture for a CNN and test the performance against established models such as Alexnet and GoogLeNet. I will also be modifying the models by means of their activation functions, initialization methods, optimizers, pooling layers, and number of layers.
In the end, I hope to establish some understanding of how much of a difference model modifications can be.

II. METHODOLOGY

A. CNN Architectures

This section will summarize the hyperparameters used to train each model. The input images are of size 32x32. All architectures are downloaded from the Pytorch website [1].

1. *Custom CNN*

   For the custom CNN, I made a very simple CNN with one convolutional layer that I will be referring to as the basic CNN. The layer has a 3x3 kernel size with no padding and stride of 1. The output maps are then passed into a pooling layer with a kernel of size 2x2, which then feeds into a fully-connected linear layer that outputs 10 softmax values.

   For this study, I will be testing several modifications to this architecture. For the pooling layer, I will be testing the network with a max-pool layer and with an average-pool layer. For the activation functions, I'll be using ReLU, tanh, and sigmoid. Lastly, I will be using two different optimizers, Standard Gradient Descent (SDG) and the Adam optimizer.

2. *Alexnet*

   I used Alexnet net to compare what a world renowned model would perform like against a simpler custom CNN. Alexnet contains a total of 8 layers, the first 5 of which were convolutional layers. 3 out of 5 of those layers contain max-pooling layers. The last 3 layers are fully-connected layers. The architecture also uses ReLU as the activation function. I modified the last fully-connected layer to output 10 features for each class in the CIFAR-10 dataset instead of the pre-trained 1000 Alexnet uses. For this network, I also trained the model using the Adam optimizer to see how it would perform compared to SGD.

3. *GoogLeNet*

   In addition to Alexnet, I also used GoogLeNet to compare the networks. GoogLeNet is a much deeper network than Alexnet with 22 total layers. This architecture utilizes convolutional layers, max-pooling, avg-pooling and inception layers to create a very advanced network. As with Alexnet, I will be training the model with the Adam and SGD optimizers and compare.

Performance Metrics

The only performance metric I use in this study is accuracy to get a rough estimate on which model performs best.

B. Dataset

I compared the models on the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [2]. The 10 classes are: airplane,

automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

III. Experiment - Performance by Problem

Each of the models were trained for 10 epochs on a predetermined training subset of the dataset by Pytorch. I will be focusing on the three main models that performed the best, one for each architecture. In addition, I will be showing the performance results for the basic CNN architecture as a reference point. The performance results for the others will be in the Appendix.

Table 1: Basic Standard

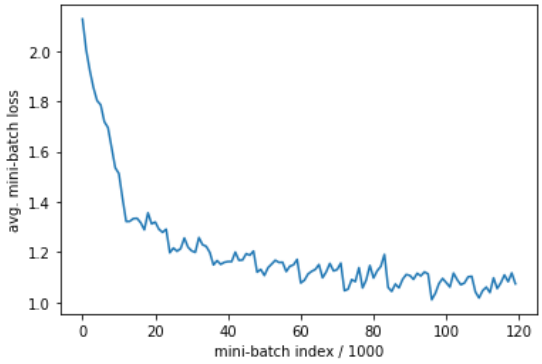


Table 1 shows the average mini-batch loss over the course of the training for the basic CNN using its standard setup of SGD for its optimizer, ReLU for its activation function, and max-pooling. The network trains well as the loss is decreasing steadily.

Table 2: Basic Standard Heatmap

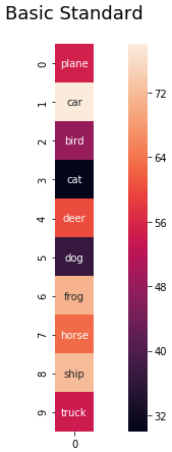


Table 2 contains a heatmap of the accuracies for each individual class in the dataset. Darker colors means the network performed worse when classifying images with these labels, and lighter colors means it performed better. Overall, this classifier performed pretty average, with a total overall accuracy of 57% on all 10,000 images.

Table 3: Advanced Tanh

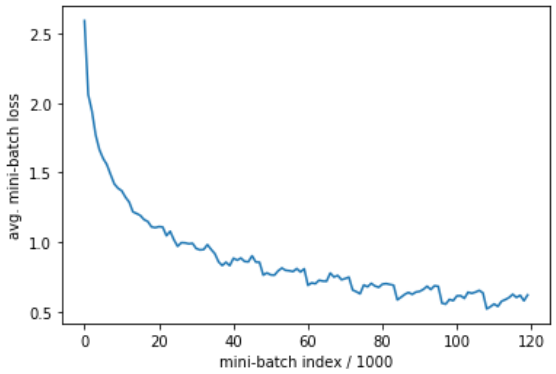


Table 3 shows the average mini-batch loss over the course of the training for the advanced CNN architecture using tanh as the activation function instead of ReLU used in the standard setup. It also uses SGD as its optimizer and utilizes max-pooling layers. The graph shows a much lower loss compared to the basic standard CNN.

Table 4: Advanced Tanh Heatmap

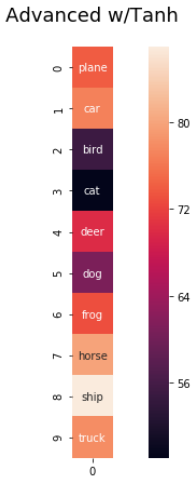


Table 4 contains the heatmap of accuracies for each class in the dataset for the advanced CNN. Compared to the basic CNN, the overall accuracy of the network is greatly improved, with bird, cat, and dog, still proving to be difficult to classify. The advanced CNN performed better than the basic CNN with a total accuracy of 70% over all 10,000 images.

Table 5: AlexNet Adam

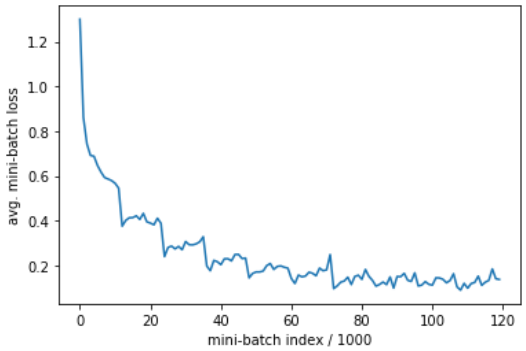


Table 5 shows the average mini-batch loss of the AlexNet network trained over 10 epochs. This version of AlexNet uses Adam as the optimizer instead of SGD. This network achieves a loss lower than the advanced CNN network.

Table 6: AlexNet Adam

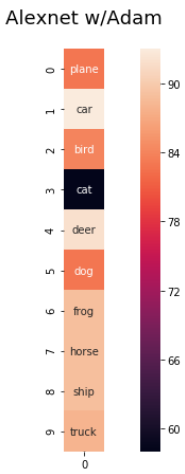


Table 6 contains the heatmap for the AlexNet network. Nearly every class performs with an accuracy in the high 80s or better, except for the cat class. This is significantly better than the advanced CNN architecture which was already an improvement over the basic CNN network. The network performs with an overall accuracy of 85% on all 10,000 images.

Table 7: GoogLeNet Adam

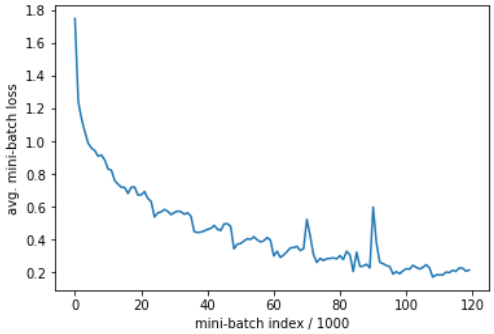


Table 7 shows the average mini-batch loss over 10 epochs of training of the GoogLeNet network. The loss spikes at around the 90th index in the graph and drops down again for an overall decreasing loss. The loss is slightly more than the AlexNet network, but outperforms the basic and advanced CNNs.

Table 8: GoogLeNet Adam Heatmap

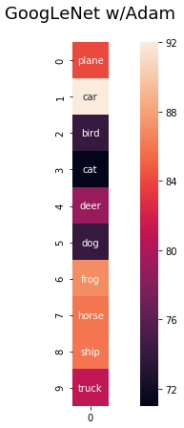


Table 8 shows the heatmap for the accuracies of each class in the dataset. The network performed worse than the AlexNet network, with half of the class' accuracies in the 70s. The other class' accuracies achieve better than 80% which is better than the advanced CNN, per class. The network performs with an accuracy of 81% overall 10,000 images.

IV. Conclusion:

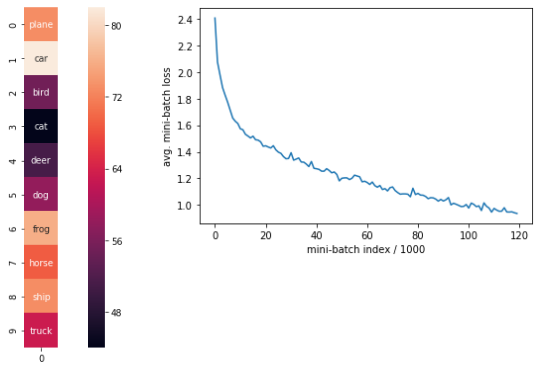
In conclusion, the modifications made to the architecture of a network play a significant role in how well the network will perform. The type of optimizer used, in this case between SGD and Adam, made the biggest difference in terms of performance. However, this performance boost was dependent on which network the optimizer was used on. On the advanced CNN, the SGD optimizer worked best, while on AlexNet and GoogLeNet, the Adam optimizer improved them significantly. The second biggest factor was the type of activation function used. Overall, ReLU performed best, with tanh closely behind. Sigmoid, however, produced abysmal results compared to the other two. Lastly, the pooling layer did cause slight shifts in performance, but nothing noteworthy in this experiment. Overall, testing different settings and modifications in creating an architecture is immensely important and should not be overlooked.

References

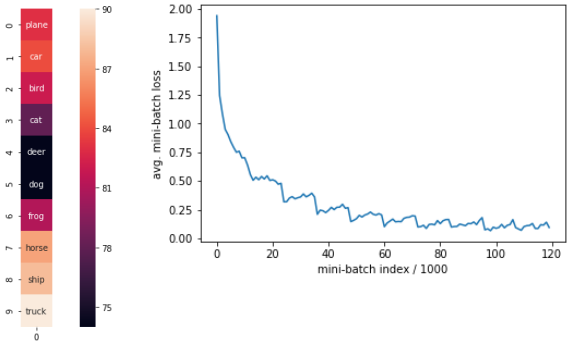
[1] https://pytorch.org/
[2] Krizhevsky, Alex. (2012). Learning Multiple Layers of Features from Tiny Images. University of Toronto.

APPENDIX

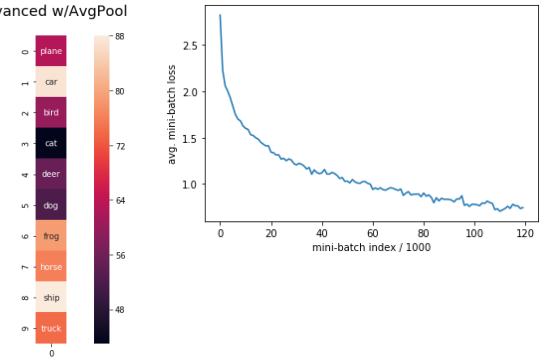
Advanced w/Adam



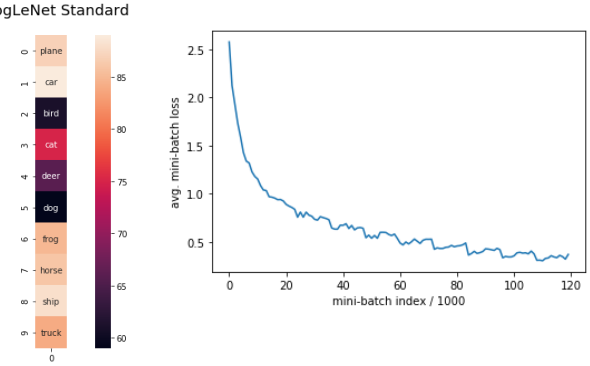
Alexnet Standard



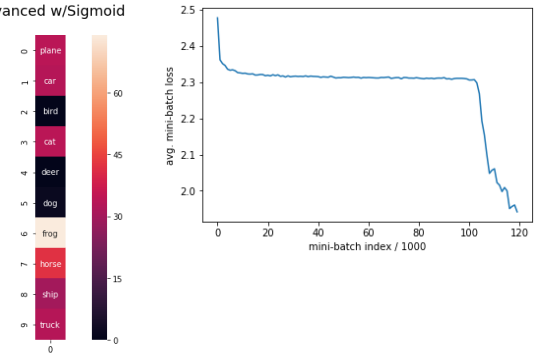
Advanced w/AvgPool



GoogLeNet Standard



Advanced w/Sigmoid



Advanced Standard