Colin Kavanagh

Zhuowen Tu

COGS 181

3/23/24

Hyperparameter Horizons: Experimentation with CNN Hyperparameters

Abstract:

Convolutional neural networks are robust neural networks used for tasks like image processing. This paper explores some different hyperparameters in training a convolutional neural network. We also explore the impact of activation functions, pooling functions, and network architecture on the model's testing accuracy. We find that activation functions have the most negligible impact on model performance, while network architecture has the most impact.

Introduction:

Convolutional Neural Networks are becoming increasingly common in our everyday lives. They have become more common since the introduction of famous neural networks like

AlexNet, which proved their effectiveness compared to classical machine learning techniques and regular feed-forward neural networks. They have proved very effective for problems like image classification, which involves processing a large amount of data. Convolutional neural networks are perfect for this task because convolution and pooling reduce the dimensionality of the data inherently as it processes it through the network.

The first part of a convolutional neural network is the convolutional layer. The convolutional layer uses learned parameters for feature extraction to determine the most critical parts of the input. This feature extraction is usually done piece by piece through a "kernel," which acts as a scanner throughout the data. These kernels can have different parameters to them to modify how they function. These kernel parameters include size, padding, and stride. The size of the kernel determines how much input the kernel extracts for each part of the scan. For example, a 3x3 kernel will scan a 3x3 unit area of the data before moving on to the next part. The padding adds data units around the outside of the data input, allowing dimensions to stay consistent. Stride determines how far the kernel should "jump" when scanning through the data. The higher the stride, the less data the kernel outputs after doing a full scan. After scanning the input data, the layer returns a "feature map" of extracted data, usually with smaller dimensions than the input data. Mathematically, the convolutional layer computes a dot product between the input data and the kernel for each stride. The following equation represents the general convolution equation:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)g'(t)$$

Where $f(t)$ is the data and $g(t)$ is the kernel.

The second part of a general convolutional neural network is the pooling layer. Convolutional neural networks use pooling layers for quicker dimensionality reduction and feature extraction. The pooling layer uses a kernel like the convolutional layer. However, the pooling kernel uses no learned parameters to extract data. Instead, the pooling layer uses a predefined mathematical function. Some examples of pooling functions include Max Pooling and Average Pooling. Max pooling takes the maximum numerical value of the data captured by the kernel, while average pooling takes the average numerical value of the data captured by the kernel.

Finally, after we have extracted enough features using the convolutional and pooling layers, we provide the feature map into a simple "fully connected" network. Depending on the task, this layer usually outputs a prediction of the class of the provided data. These perceptrons in the fully connected layer rely on learned weights to give input to the other neurons in the next layer. The equation for a layer of these perceptrons is as follows:

$$y = f(Wx + b)$$

Where x is the input data, W is the weights matrix, b is the bias term, f is the activation function, and y is the output. The output can be either a prediction or an input to another layer of perceptrons. The activation function can vary depending on the purpose of the layer. These include activation functions like Softmax or ReLU. The Softmax function outputs a value between 0 and 1 for each neuron, which we interpret as the probability of a class for a classification layer. The math for the Softmax function is as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Where z is the vector of outputs from the layer.

The ReLU function provides a positive value from the previous output. Each neuron considers the input only if it is non-negative. If the value is non-negative, it passes the same input as its output. If it is negative, it passes a 0, so the neuron is not activated. The equation for the ReLU function is as follows:

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

Where x is the input to the perceptron.

As discussed, multiple parameters can be tweaked to optimize a convolutional neural network's performance. Modifying some of these parameters may benefit the neural network; however, depending on how we use them, some may be detrimental. This paper will explore various hyperparameters for a convolutional neural network for the CIFAR-10 dataset. The CIFAR-10 data contains thousands of 32x32 color images of 10 different classifications.

Methods:

In this paper, we will experiment with three different hyperparameter tunings. These tunings will be the activation functions, number of layers, and type of pooling layer. We will use the "skeleton code" from Homework 4 from COGS 181 Winter 2024 to streamline the model

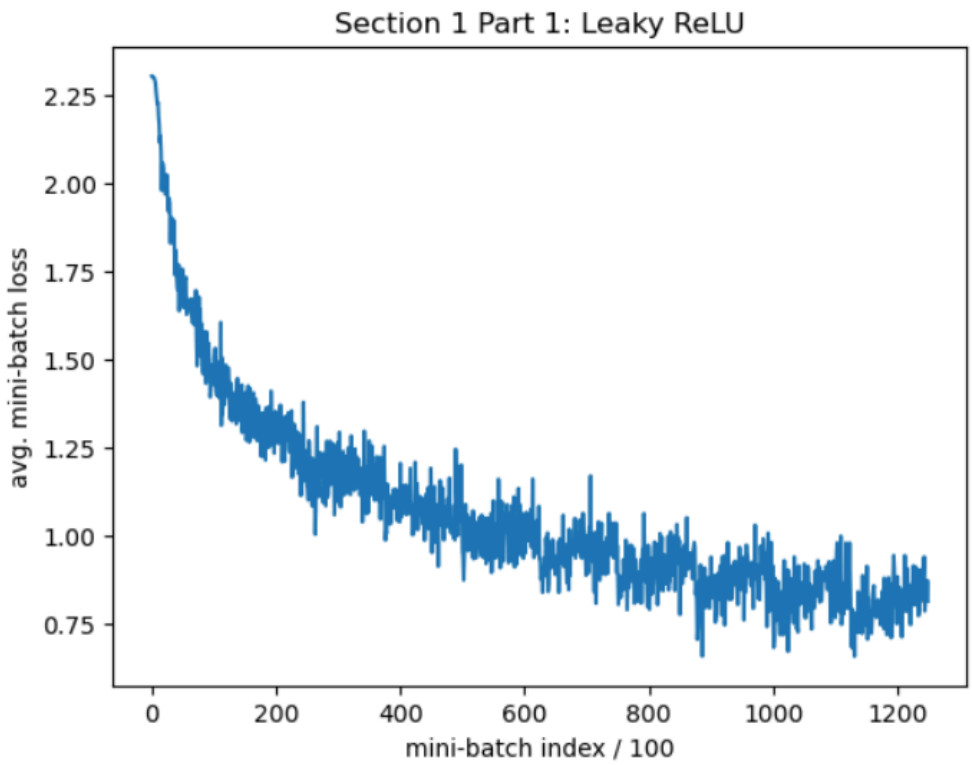
setup. As a baseline, we will use the tutorial model provided on the PyTorch website for the CIFAR-10 dataset as a "control" model. This model consists of two convolution and pooling layers and three fully connected linear layers. This baseline model accuracy is 61% on the CIFAR-10 dataset. We will use ten epochs of training for each model with mini-batch training. We will continue to use cross-entropy loss as the loss method. We will use stochastic gradient descent for the optimization method for all models. With these experiments, we will determine if modifying specific hyperparameters will increase or decrease the accuracy of the test set. We will also include a graph of the training curve for each model to indicate model viability beyond testing accuracy. We will modify these hyperparameters one at a time to better indicate their impact on the model.

Experiment:

Section 1: Activation Functions

Part 1: Leaky ReLU

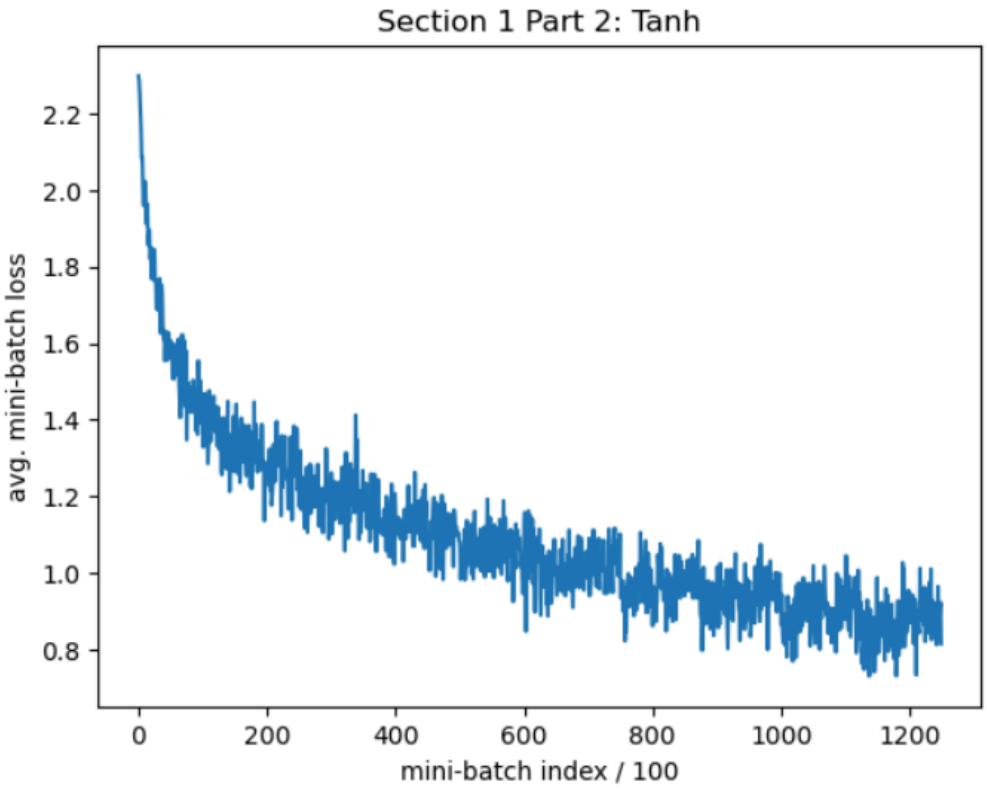
We will begin by experimenting with "leaky ReLU" for the activation functions. While ReLU provides the network with benefits such as speed and simplicity, it loses information by not allowing a negative output. Leaky ReLU can rectify this. For this model, we will change all the activation functions to leaky ReLU functions.



Changing these activation functions resulted in 63% test accuracy compared to 61% for the control model. Since leaky ReLU preserves more input than regular ReLU, more information is passed across layers, potentially leading to greater accuracy.

Part 2: Tanh functions

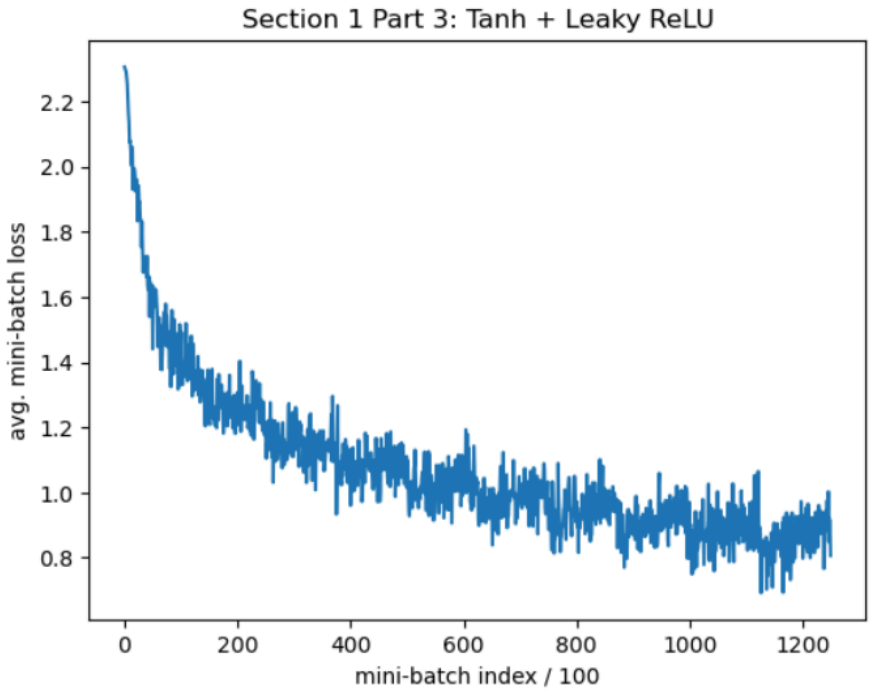
This section will change the network's activation functions to “tanh” functions. Tanh allows for either a positive or negative output. While leaky ReLU also allows for negative output, it heavily favors positive output.



The final testing accuracy using the Tanh functions was 60%. While this is slightly worse than the control, Tanh works better when combined with different hyperparameters. For example, multiple max-pooling layers inherently disincentivize negative values, making the possibility of negativity somewhat irrelevant. Thus, Tanh works better with the proper hyperparameters.

Part 3: Leaky Relu and Tanh

For this part, we will combine Tanh and Leaky ReLU functions. Hopefully, this combination of functions will provide better accuracy than exclusively using one type of function.



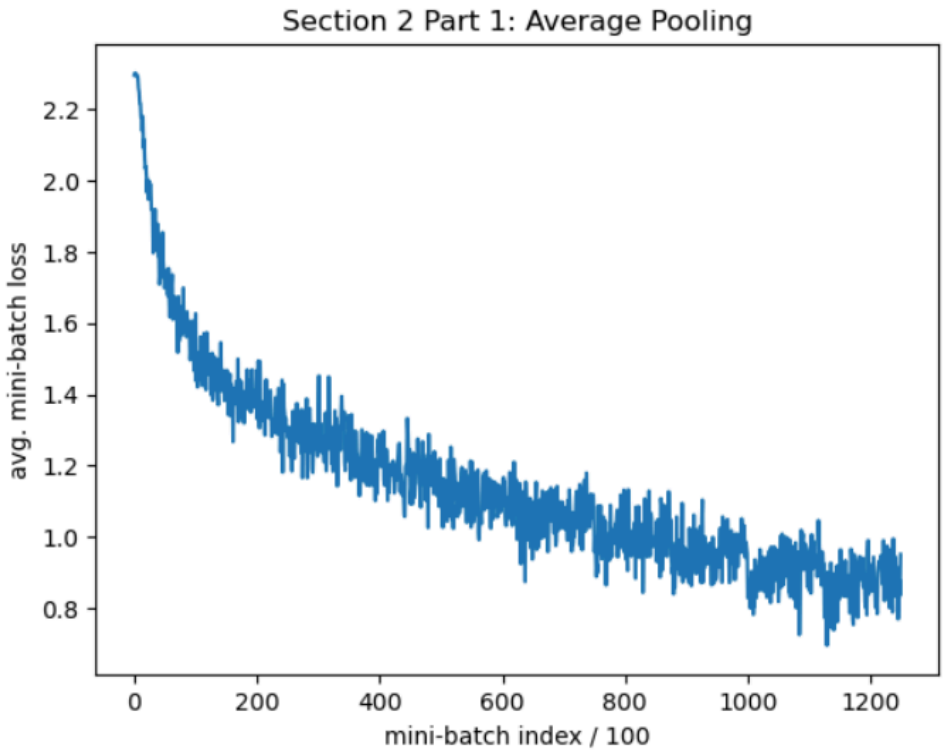
Combining the Tanh function with the Leaky ReLU function provided a slightly better performance on the testing set with an accuracy of 62%. This increased performance could be due to both functions allowing negative input. The tanh function balances positive and negative input equally, while the Leaky ReLU function provides more bias to positive inputs. Combining the two functions could have allowed for an equilibrium between positive and negative inputs, ensuring the network passes the most relevant features through the activation functions. Therefore, the most relevant perceptrons are activated.

Section 2 Pooling Layers:

Part 1: Average Pooling

To begin with, we will change the max pooling layers to average pooling layers. The average pooling layers take the average value from the kernel's values instead of the maximum

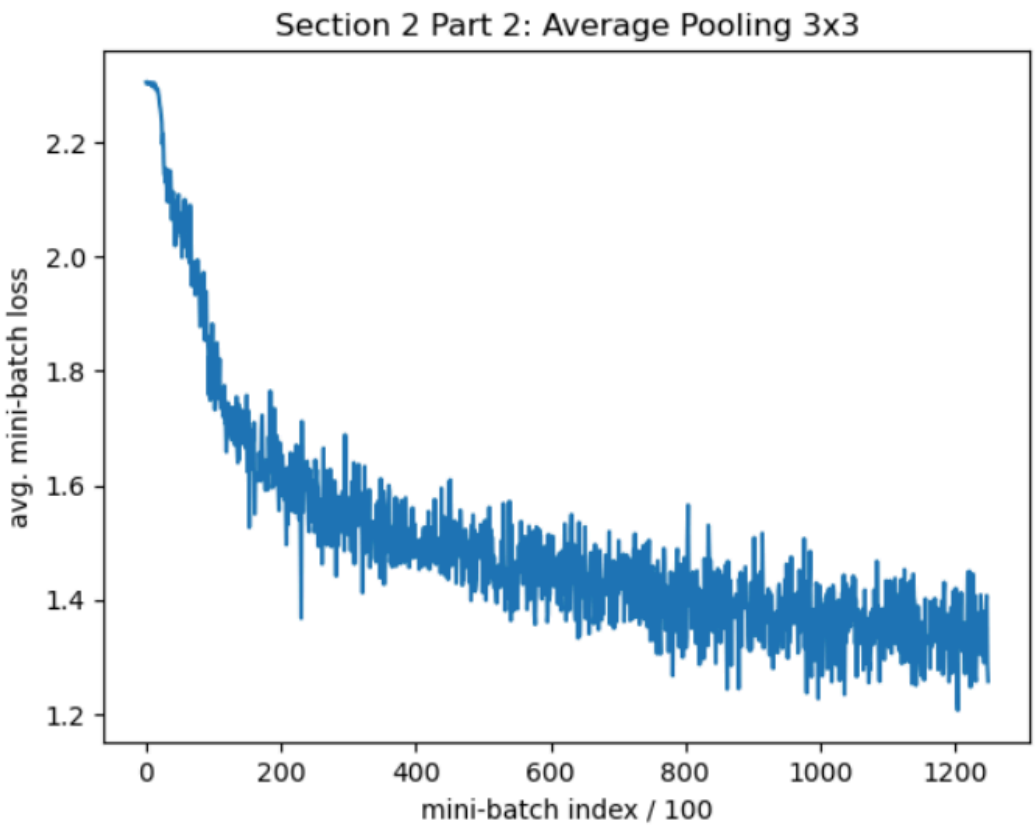
value. While maximum pooling tends to be more common in CNNs, we will experiment with average pooling to see if our results change.



The average pooling network had an accuracy of 61%. These results might indicate that, for the size of our network, there is no substantial difference between average pooling and max pooling. However, this might change if we increase the kernel size.

Part 2: Average Pooling 3x3:

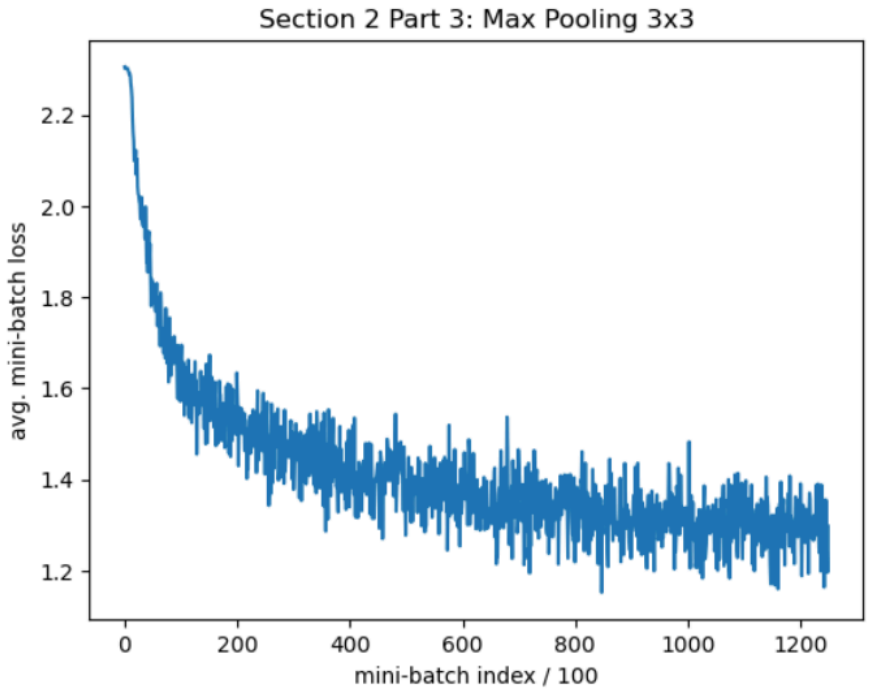
To further streamline the process, we will create a network with an average pooling kernel size of 3x3 instead of 2x2. This bigger kernel size will, hopefully, allow for more significant data compression and improve the network's overall performance.



The Average Pooling 3x3 model gives us noticeably worse performance at 50% accuracy. This model's performance is the steepest drop in accuracy we have seen through this paper so far, dropping 11% from the control model's performance of 61%. This change in performance, however, might not be due to the change in the pooling function itself but instead to the change of the input to the first fully connected layer. To make the more extensive pooling layer work with the rest of the network, we needed to change the input size of the first fully connected layer from 16*5*5 (400) to 16. Now, instead of downscaling the input from 400 to 120, the fully connected layer has to *upscale* the features from 16 features to 120 features. This functional change in network structure could be worse overall than the change in the kernel size for the average pooling layer.

Part 3: Max Pooling 3x3:

To see if the bigger kernel substantially changes the pooling type, we will experiment using 3x3 kernels for max pooling layers. While both the 2x2 kernels for both types of pooling resulted in equal accuracy, we will explore whether the max pooling layer benefits from a larger kernel size.

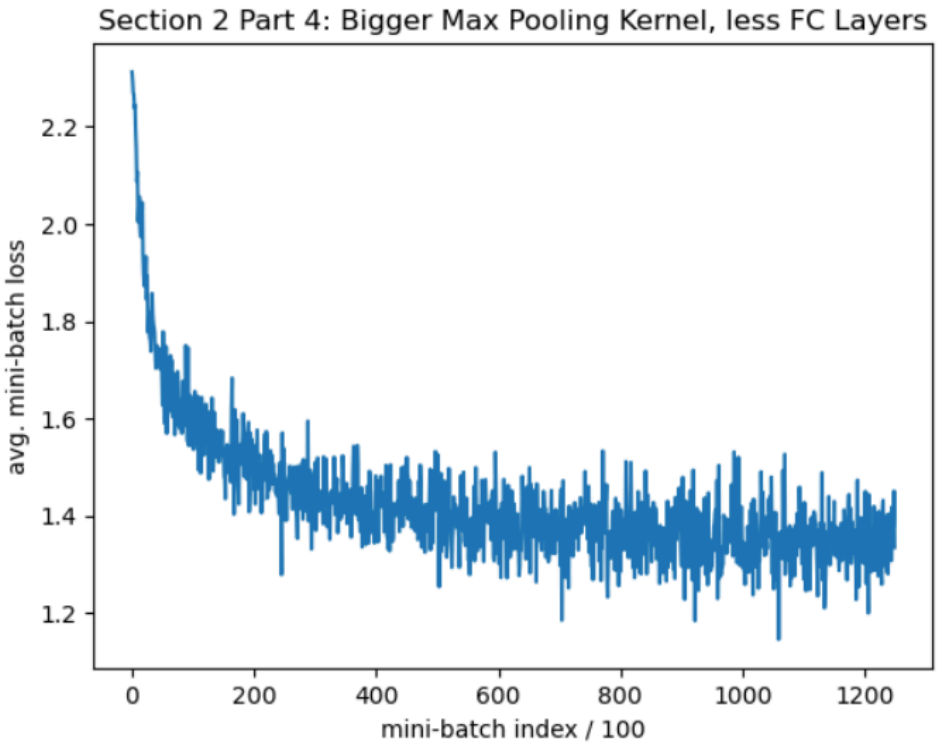


The test accuracy on the max pooling 3x3 kernel was 52%, only slightly better than the accuracy of the average pooling 3x3 kernel. This change in accuracy could also be due to the change in network structure necessary to implement a bigger kernel.

Part 4: Bigger Kernel, less FC layers

The lower accuracy could be due to the change in layer structure required to take the input of the bigger kernel size for the fully connected layers. In this part, we will attempt to keep

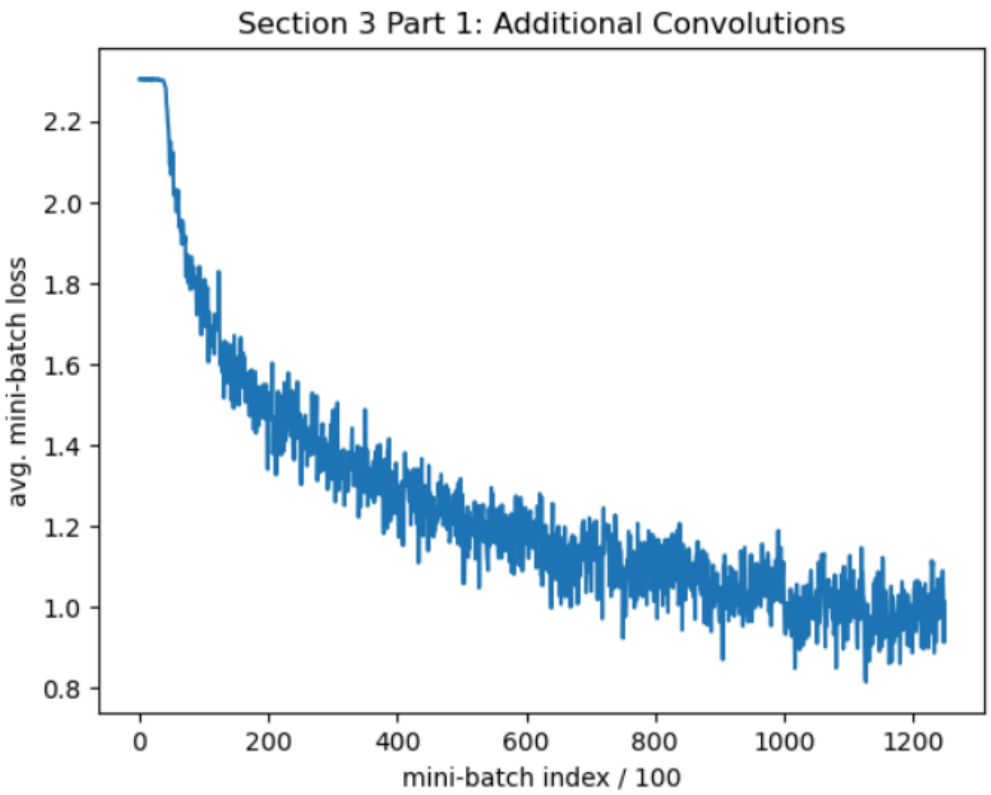
a max pooling kernel of 3x3. However, we will eliminate the last two layers of the fully connected part of the network and change the output size of the first layer from 120 to 10. This layer's functionality now changes to class predictions.



The change in pooling kernel and fully connected layers results in an accuracy of 51%. The average mini-batch training loss also levels out and becomes much "flatter" compared to the training curves of the other models we have explored. This odd training curve could indicate that changing the structure of the layers will have the most substantial impact on the network's accuracy.

Section 3: Network Structure

We will now modify CNN's network structure to improve the model's accuracy on the testing set. For this first iteration, we will attempt to add more convolutional and pooling layers to the network. We will remove the last pooling layer and add three additional convolutional layers. These additional convolutional layers will hopefully allow for greater feature extraction and accuracy.

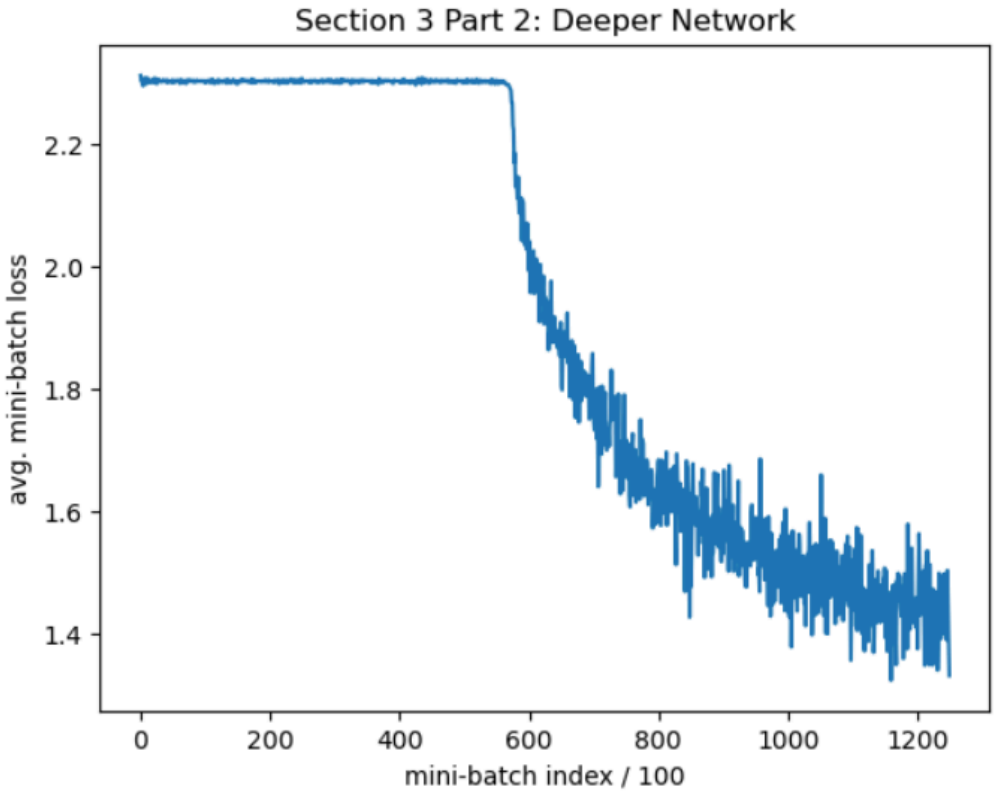


This network structure has a marginal difference in accuracy for the testing set, resulting in 61% test accuracy. This lack of change in accuracy could be because the network is overfitting

the data due to the increased convolutional layers or because there are not enough pooling layers to aggregate the data.

Part 2: Deeper Network

This model will explore how adding even more convolutional layers impacts performance. This model will use six convolutional layers, with a max pooling layer every two layers. However, we will also change some of the hyperparameters of the layers. The convolutional layers will have an increasingly smaller kernel size to extract more niche features. The layers will also have smaller outputs to force the network to extract only the most essential features. Therefore, the fully connected layers will receive an input of 180 from the convolutional layers compared to the 400 in the control model.



This “deeper” network performs the worst out of all the models, with a test accuracy of 48%. While this is the worst performance we have seen so far, it gives us some interesting insights into how changing the architecture of the network impacts performance. First, we see a unique training graph, with nearly no progress until around index 600. This lack of training progression could be because of how we are optimizing the network. Modifying things like the learning rate or optimization function could improve mini-batch accuracy earlier in the training process. A more significant number of epochs could also give the model more time and opportunity to train, overtaking the other models.

Conclusion:

This paper explored some hyperparameters involved in training a convolutional neural network. We have explored activation functions, pooling methods, and network architecture and their impact on model performance compared to a simple control model. Modifying activation functions can provide slightly better or worse performance, indicating that they impact model performance. We found that max pooling and average pooling can provide similar results depending on the context. Finally, we have found that modifying the layer structure of the network has the most impact on model performance. Even if we could not use this to increase the model's accuracy, we have evidence of the substantial impact of network structure on model performance.

Works Cited

Craig, Lev. “What is a convolutional neural network (CNN)?” *TechTarget*, January 2024, https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network. Accessed 22 March 2024.

IBM. “What are Convolutional Neural Networks?” *IBM*, https://www.ibm.com/topics/convolutional-neural-networks. Accessed 22 March 2024.

Kain, Nitin Kumar. “Understanding of Multilayer perceptron (MLP) | by Nitin Kumar Kain.” *Medium*, 21 November 2018, https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f. Accessed 22 March 2024.

Priya C, Bala. “Softmax Activation Function: Everything You Need to Know.” *Pinecone*, 30 June 2023, https://www.pinecone.io/learn/softmax-activation/. Accessed 22 March 2024.

Shiksha Online. “RELU and SIGMOID Activation Functions in a Neural Network.” *Shiksha*, 12 April 2023, https://www.shiksha.com/online-courses/articles/relu-and-sigmoid-activation-function/. Accessed 22 March 2024.

Unzueta, Diego. “Fully Connected Layer vs Convolutional Layer: Explained.” *Built In*, 18 October 2022, https://builtin.com/machine-learning/fully-connected-layer. Accessed 22 March 2024.