Damian Hupka
801091092
ECGR 4090 - Real-Time AI
https://github.com/dhupka/Real-Time-AI

Homework #4 https://github.com/dhupka/Real-Time-AI

Problem 1

The code provided from the "Deep Learning with Pytorch" Github repository was used as a baseline implementation and modified accordingly to suit this problem. A link to the repository is provided here: https://github.com/deep-learning-with-pytorch/dlwpt-code/blob/master/p1ch7/2_birds_airplanes.ipynb.

Problem 1.a.) "HW4/hw4p1ab.ipynb"

This problem was solved using some of the baseline code provided within "p1ch7" of the "Deep Learning with Pytorch" repository. However, instead of identifying between the two classes of birds and airplanes, it was modified to be distinguishing between four classes: frog, horse, ship, and truck. The solution to this problem and the following Problem 1.b.) is found within the "HW4/hw4p1ab.ipynb" Jupyter notebook of the Github repository for this course provided above. This network consisted of a fully-connected layer, followed by an activation layer being tanh, and a fully connected output layer for the four output classes. The model was trained for 200 epochs and the results of this training are below in Table 2.

One Hidden Layer Fully Connected Neural Network		
Training Time (s)	**Training Loss**	**Evaluation Accuracy**
329.63 s	0.07843	77.40%

Table 1: Fully Connected Neural Network Results (one hidden layer)

As in the table above, this network offered a reasonably low training time at approximately five minutes, the training loss was steadily approaching zero as expected. Furthermore, this model presented an adequate validation accuracy of 77.40%. This relatively low validation accuracy is caused by a few key issues. Primarily, fully-connected networks are typically less apt in image recognition due to each neuron layer being fully connected to the next, and the provided network is relatively shallow. Later in this assignment, a CNN model will be used and compared to the fully-connected network.

Problem 1.b.) "HW4/hw4p1ab.ipynb"

Similar to Problem 1.a.) a fully-connected neural network was utilized to classify between four classes of the CIFAR dataset, these four classes being the same as above: frog, horse, ship, and truck. However for this problem, the network depth was modified to include two additional hidden layers. These hidden layers were two more fully-connected layers each followed by an activation tanh layer respectively. The model was trained for 200 epochs and the results of this are below in table 2.

Damian Hupka
801091092
ECGR 4090 - Real-Time AI
https://github.com/dhupka/Real-Time-AI

Two Hidden Layers Fully Connected Neural Network (Four Classes)		
Training Time (s)	**Training Loss**	**Evaluation Accuracy**
855.91 s	0.000197	76.93%

Table 2: Fully Connected Neural Network Results (two hidden layers)

The table above demonstrates the model taking nearly 15 minutes to complete, this significantly increased training time is coming from the number of MACs that are necessary to complete fully-connected layers as compared to other options. Furthermore, compared to the 0.07843 training loss as in Problem 1.a.) above this network reached an outstanding training loss of 0.000197. Unfortunately, this improved training accuracy is only improving the accuracy of the model on the training set, but not the model's aptitude to predict new information. This is observed in the validation accuracy clocking in at 76.93%, nearly half a percent less than the original model with only one hidden layer. This is a prime example of overfitting in the model as the accuracy on new data did not improve, and in fact decreased. Thus, increasing the depth (complexity) of the model in a fully-connected network may not always improve the performance of the model and will similarly drastically increase the training time as more of these hidden layers are added. Nearly three times the training time compared to Problem 1.a.) is not at all profitable as the validation accuracy decreased, only improving on the training loss.

Problem 2

The code provided from the "Deep Learning with Pytorch" Github repository was used as a baseline implementation and modified accordingly to suit this Problem. A link to the repository is provided here: https://github.com/deep-learning-with-pytorch/dlwpt-code/blob/master/p1ch8/1_convolution.ipynb.

Problem 2.a.) "HW4/hw4p2a.ipynb"

This problem was solved using some of the baseline code provided in the "Deep Learning with Pytorch" Github repository referenced above, specifically within the "p1ch8" directory and the "1_convolution.ipynb" Jupyter notebook. However, the key distinction is that as in Problem 1 the number of output classes was modified from the bird and airplane, to be distinguishing between four classes: frog, horse, ship, and truck. This modification can be found in the Github repository used for this course within the "HW4" directory in the file "hw4p2a.ipynb" Jupyter notebook. This initial network consisted of two convolutional layer blocks (conv2d, activation function, pooling layer), followed by a linear layer to reshape the model into the expected format to be an input into the final output layer for the four output classes. The model was trained for 200 epochs and the results of model criteria are below in Table 3.

CNN (Four Classes)			
Training Time (s)	**Training Loss**	**Training Accuracy**	**Evaluation Accuracy**
149.27 s	0.0323	99.0%	90.0%

Table 3: CNN Results

As seen in the table above, compared to the higher performing of the two fully-connected networks, every benchmark in the CNN model for identifying between four classes of the CIFAR dataset are substantially improved. Let's further analyze these characteristics: The training time is nearly halved compared to the fully-connected network. This is caused by the number of parameters involved in computing a fully-connected layer for each neuron being connected to each and every neuron in the previous layer, which is explicitly not the functionality of a CNN. In the CNN, a layer is only connected to a select number of neurons leading to decreased parameters to be learned **a**nd allowing for network growth on fewer parameters**.** Likewise, the training loss, training accuracy, and validation accuracy is significantly improved with the CNN model providing 90.0% validation accuracy compared to the 77.40% of the fully-connected network. However, this improved accuracy logically follows as CNNs will perform significantly better in regards to image classification as the main benefit is in feature extraction and in maintaining data integrity.

Problem 2.b.) "HW4/hw4p2bc.ipynb"

The solution to this problem is similar to Problem 2.a) with some modifications, the necessary code to complete can be found in the provided course Github repository under "HW4/hw4p2bc.ipynb" Jupyter notebook file. The model that was utilized in Problem 2.a) was similar in this problem. However, the classes that were to be recognized were modified. Instead of distinguishing between the four classes frog, horse, ship, and truck of the CIFAR dataset, this problem identified between all 10 of the classes of the CIFAR10 dataset. Thus, the model was trained for 200 epochs and the results for the model are shown below in Table 4.

CNN (All 10 Classes)			
Training Time (s)	**Training Loss**	**Training Accuracy**	**Evaluation Accuracy**
317.51 s	0.576	80.0%	64.0%

Table 4: CNN Results

The results above are certainly expected. The training time using a CNN model with only the four classes was approximately 2.5 minutes; wherein, the training time when distinguishing between all 10 classes is nearly doubled at three minutes. In terms of model accuracy, both the training loss, training accuracy, and

validation accuracy are all performing markedly worse than with just the four classes. Furthermore, as a relatively simple model in Problem 2.a.) with four classes the performance was adequate, but when increased to a more complex problem of identifying between 10 classes it is likely that the depth of our somewhat shallow model may need to be improved to increase performance. This will be examined further in Problem 2.c.) below.

Problem 2.c.) "HW4/hw4p2bcipynb"

The model that was used above in Problem 2.b.) was similarly implemented for this solution, however an additional convolution block was added (Conv2d, activation (tanh), and pooling). The fully connected layer was also modified to match the added convolution layer's intermediate feature dimensions. The code to complete this problem is also found in "HW4/hw4p2bc.ipynb" Jupyter notebook file. The model was trained for 200 epochs and the output characteristics are below in Table 5.

CNN (All 10 Classes, with additional layers)			
Training Time (s)	**Training Loss**	**Training Accuracy**	**Evaluation Accuracy**
459.05 s	0.4733	82.0%	68%

Table 5: CNN Results (additional Convolution, activation, and pooling layer)

As in the table, above each of the analyzed characteristics improved compared to the slightly more shallow network in Problem 2.b.). The training time showed a minor increase which conceptually follows as the addition of another convolution block will increase the model complexity, and further increase the number of MACs that are occurring causing increased processing time. The model loss decreased marginally to 0.4733 and showed a similar minor increase from 80.0% to 82.0%. Lastly, the validation accuracy improved from 64.0% to 68.0%. Thus, with the only slight increase in training accuracy leading to a rather substantial increase in validation accuracy it is unlikely that the model experienced much overfitting as in such a case the model would experience a larger increase in training accuracy with the validation accuracy decreasing. Furthermore, the addition of an additional convolution block proved highly beneficial towards this model to more adequately predict new information and the increased model complexity was necessary to reach higher accuracy when distinguishing between all 10 of the CIFAR dataset classes. Continued testing could be competed with the potential benefits of further increasing the model depth, but this would likely lead to overfitting as at that point the increased complexity would be detrimental. However, utilizing regularization methods such as dropout or batch normalization may lead to more prosperous improvements on the model performance in this application without drastically increasing model size, complexity, and number of parameters.