Assignment 7

ECE 181 - Introduction to Computer vision

Kenny Hoang Nguyen
X299187
Section #3

March 2020

1 Problem 1

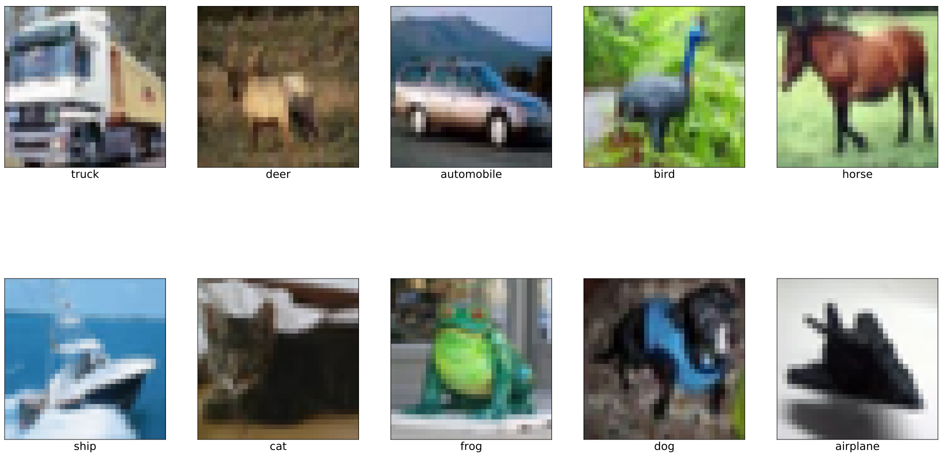
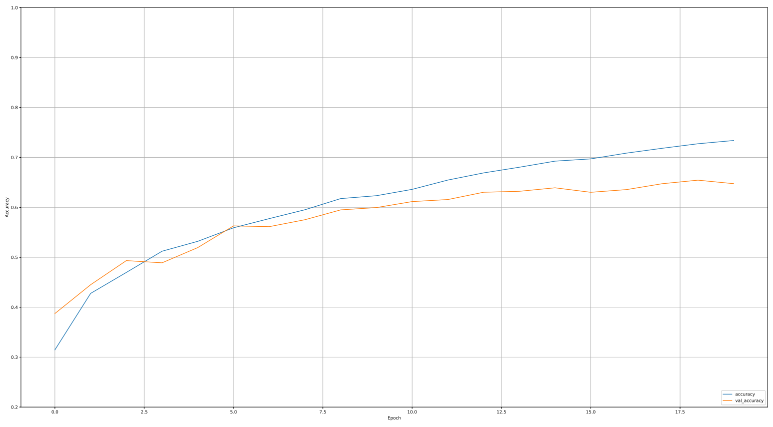


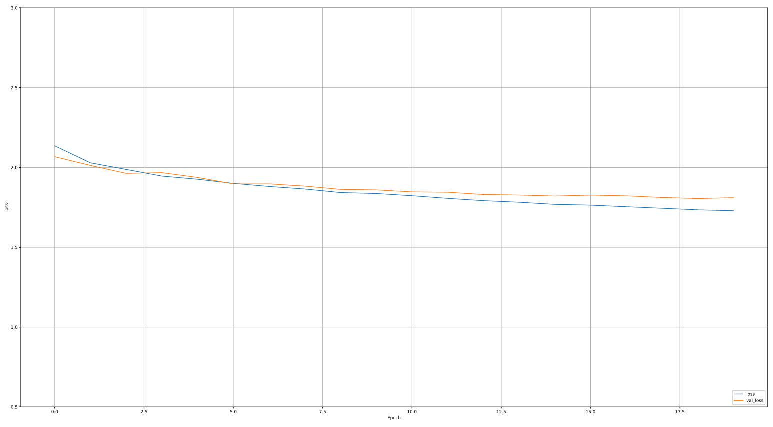
Figure 1: Sample image for each class

2 Problem 2

In this problem I chose to implement the layers as given as the classical MNIST model. Since it was not specified how our output layer is going to be implemented, other than 10 neurons, we used softmax as activation function since this is a multi-class classification problem. In the neural network we used rectified linear unit since it seemed the easiest to use between the layers.
For the compilation of the model I tuned down the learning rate from the default value and used 0.005 as learning rate. I also added that the model splits up the training data so that it trains on 80% of the training data and use the rest as validation data.
Running the model it seems like the results vary for each run if we generate a new model every time. This seems to be because the random initialized value is random and sometime it generates a good initializing value. I ran the neural networks twice with a new initialization value each time, and an epoch of 20. The first run ended up with a training accuracy of 73.4%, validation accuracy of 64.7% with sparse categorical crossentropy loss of 1.7289 and 1.8108 for training and validation respectively. Using the test data to find out the loss and accuracy of that data we got 1.8193 and 63.8%. The result of how the accuracy and loss changed for each epoch is shown in 2a and 2b.
For the second run we ended up with accuracy of 72.0% and 64.3% for training and validation respec-



(a) Accuracy for 1st model

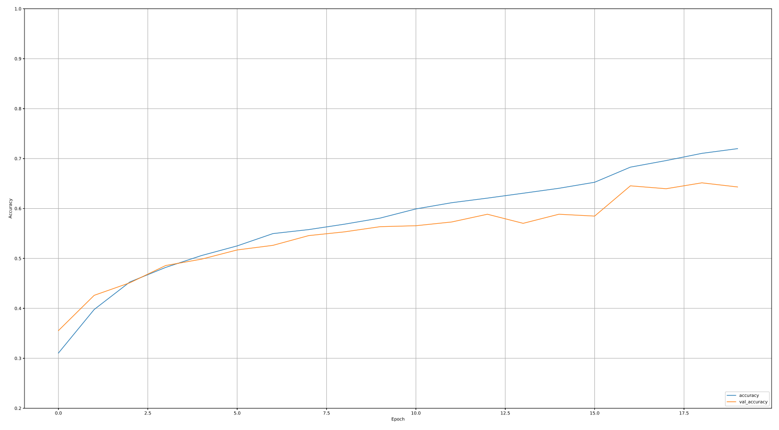


(b) Loss for 1st model

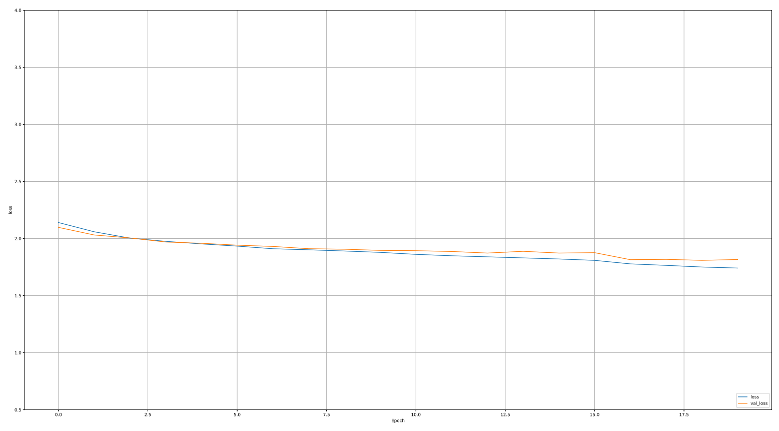
Figure 2: Accuracy and loss for 1st model

tively, and losses at 1.7415 and 1.8161 for training and validation. The testing data had an accuracy at 64.2% and loss at 1.8176. The accuracy and loss for training and validation at each epoch is shown in 3a and 3b.

We notice that for both models the accuracy increased for each epoch and the loss decreased, which is



(a) Accuracy for 2nd model



(b) Loss for 2nd model

Figure 3: Accuracy and loss for 2nd model

what is expected if we got reasonable tuning parameters.

3 Problem 3

For the input layer, where you take in the image that is of size $32x32x3$, you do not have any weights. For the first convolutional layer (conv2d + maxpooling) you have a $5x5$-filter with 3 in depth since the image has 3 channels and have therefore 3 in depth. You will have to tune $5x5x3$ parameters plus one bias term for one filter. For this layer we have 32 different filters which means that in the first layer we have a total of

$$5 \cdot 5 \cdot 3 \cdot 32 + 32 = \underline{2432} \text{ parameters} \tag{1}$$

parameters. We repeat this procedure for both our convolutional layers.
The maxpooling "layers" do not have any parameters/weights because that layer looks at the maximum value in an area and pools that area together with that maximum value.
From the first layer we get 32 different resulting images because we had 32 different filters in that layer. Essentialy that means that the resulting image here is of dimensions $28x28x32$. Because the dimension we get after the convolution for width and height of the picture is $32 - 5 + 1 = 28$ Since we used a stride of 2 the image the input into the next layer is of size $14x14x32$. This means that in the second layer we need to have a convolutional filter that is 32 units deep. Also the dimension of that filter is $5x5x32$ in addition to bias term for each filter. In this layer we have 64 different filters meaning

$$5 \cdot 5 \cdot 32 \cdot 64 + 64 = \underline{51264} \text{ parameters} \tag{2}$$

The resulting image is of size $(14 - 5 + 1)x(14 - 5 + 1)x64 = 10x10x64$. In the maxpooling here we also use a stride of 2 meaning the image into the next layer is of size $5x5x64$.
Since we here have dense layers and fully connected network from here on we have to flatten the image to use it from here. Meaning the input into the first dense layer is

$$5 \cdot 5 \cdot 64 = 1600 \text{ long} \tag{3}$$

All the values in this input vector is fully connected into the next layer with weights, which in this layer has 1024 neurons in addition to bias terms for each neuron meaning the number of parameters here are

$$1600 \cdot 1024 + 1024 = \underline{1639424} \text{ parameters} \tag{4}$$

The dropout layer does not have any parameters. But to connect to the next dense layer which is the output layer with 10 neurons we need to fully connect the former layer of 1024 neurons, which results in

$$1024 \cdot 10 + 10 = \underline{10250} \text{ parameters} \tag{5}$$

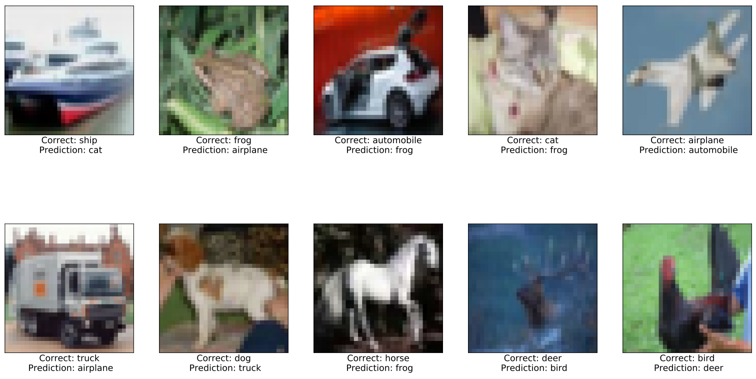
In total for the whole neural network there are

$$2432 + 51264 + 1639424 + 10250 = \underline{\underline{1,703,370}} \text{ parameters} \tag{6}$$

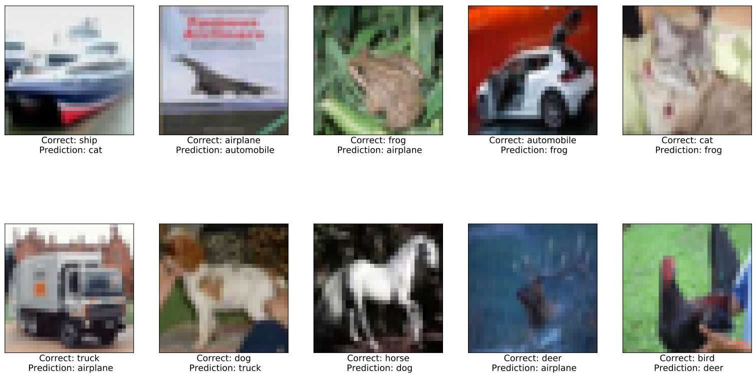
that needs to be tuned.

4 Problem 4

Here we used model.predict to get a vector of the class prediction for each image we send into the neural network. Then I iterated through each test image and found 10 unique images that were wrongly predicted. This resulted in the figure 4a for the 1st model and 4b for the 2nd model.



(a) Wrong predictions from 1st model



(b) Wrong predictions from 2nd model

Figure 4: Wrong predictions from the two models