

Assignment 3

Deep Convolutional Neural Networks ELE494-09

Nasir Khalid
b00065082

I. IMPORTING CIFAR-10 DATASET

This was done using the `keras.datasets` object. They were imported as training and testing images. Along with training and testing labels. Shown below is an output from the code that displays their shape and randomly displays a single image from the dataset.

```
Shape of training images: (50000, 32, 32, 3)
Shape of training labels: (50000, 1)
Shape of testing images: (10000, 32, 32, 3)
Shape of testing labels: (10000, 1)
Image 49348: Label #[1] (image of Automobile)
0
5
10
15
20
25
30
0 5 10 15 20 25 30
```

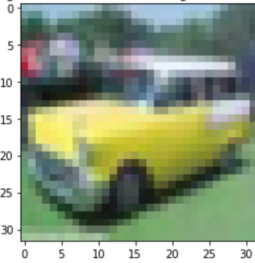


Figure 1: Data set shapes and random image from dataset

II. DESIGN YOUR OWN ARCHITECTURE

For this step I created a custom architecture that consists of 4 convolution layers and 2 max pooling layers. The final few layers consist of 2 dense layers and the final softmax layer for the output. Batch size is 32. A summary is shown below

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 32, 32, 32)	896
activation_15 (Activation)	(None, 32, 32, 32)	0
conv2d_10 (Conv2D)	(None, 30, 30, 32)	9248
activation_16 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_5 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_11 (Conv2D)	(None, 15, 15, 64)	18496
activation_17 (Activation)	(None, 15, 15, 64)	0
conv2d_12 (Conv2D)	(None, 13, 13, 64)	36928
activation_18 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_6 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_3 (Flatten)	(None, 2304)	0
dense_7 (Dense)	(None, 512)	1180160
activation_19 (Activation)	(None, 512)	0
dense_8 (Dense)	(None, 512)	262656
activation_20 (Activation)	(None, 512)	0
dense_9 (Dense)	(None, 10)	5130
activation_21 (Activation)	(None, 10)	0
Total params: 1,513,514		
Trainable params: 1,513,514		
Non-trainable params: 0		

Figure 2: Summary of initial architecture

There are a total of 1513514 parameters. When this network was trained it performed poorly with a final loss value of 14.4988 and a validation loss of 14.5740 and an accuracy of 10% with a validation accuracy of 9.58%. On the test data the accuracy was 10%. Shown below are the accuracy and loss curves for this network.

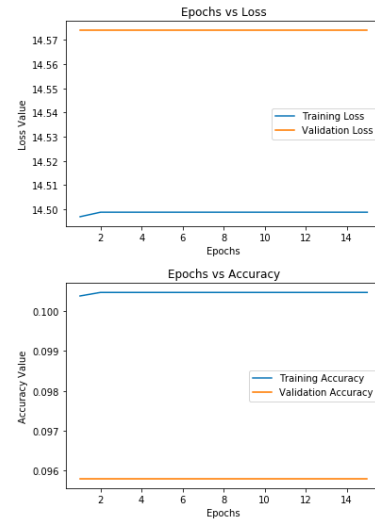


Figure 3: Loss and Accuracy curves for initial architecture

III. BATCH NORMALIZATION

Batch normalization is used to increase the speed and performance of neural networks. It is used to normalize the inputs of each layer and was originally introduced to solve the internal covariate shift problem where the distribution of the input layer continuously changes and therefore the next layer has to readjust it self to account for the change in distribution. Therefore we use batch normalization to readjust the distribution before we continue forward through the network.

It was implemented in the architecture by adding it in between the activation and output layers for each layer. The new summary can be seen below

Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization_19 (Batch Normalization)	(None, 32, 32, 32)	128
activation_50 (Activation)	(None, 32, 32, 32)	0
conv2d_30 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_20 (Batch Normalization)	(None, 32, 32, 32)	128
activation_51 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_15 (MaxPooling)	(None, 15, 15, 32)	0
conv2d_31 (Conv2D)	(None, 15, 15, 64)	18496
batch_normalization_21 (Batch Normalization)	(None, 15, 15, 64)	256
activation_52 (Activation)	(None, 15, 15, 64)	0
conv2d_32 (Conv2D)	(None, 13, 13, 64)	36928
batch_normalization_22 (Batch Normalization)	(None, 13, 13, 64)	256
activation_53 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_16 (MaxPooling)	(None, 6, 6, 64)	0
flatten_8 (Flatten)	(None, 2304)	0
dense_22 (Dense)	(None, 512)	1180160
batch_normalization_23 (Batch Normalization)	(None, 512)	2048
activation_54 (Activation)	(None, 512)	0
dense_23 (Dense)	(None, 512)	262656
batch_normalization_24 (Batch Normalization)	(None, 512)	2048
activation_55 (Activation)	(None, 512)	0
dense_24 (Dense)	(None, 10)	5130
activation_56 (Activation)	(None, 10)	0
Total params: 1,518,378		
Trainable params: 1,515,946		
Non-trainable params: 2,432		

Figure 4: Summary of architecture with batch normalization

Adding batch normalization greatly improved performance as the network as the training loss had a final value of 0.1017 with a accuracy of 96.67%. The validation loss ended at 1.3341 with a final accuracy of 78.26%. The test set reported an accuracy of 77%. The loss and accuracy graphs can be seen below:

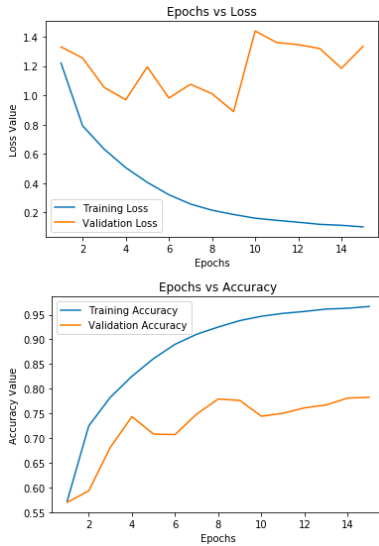


Figure 5: Loss and Accuracy curves for batch normalized network

Although the network is learning much faster it has overfit the data because it performs great on the training dataset but fails at the validation and test set.

IV. MEAN SUBTRACTION AND NORMALIZATION

Some code was written to take the mean of each channel (RGB) of each of the total images. The mean for each channel

was then subtracted from the image. After this we computer the standard deviation for each channel (RGB) of each image and normalized the image by it. For example in the first image the first pixel of the red channel was 59 and after mean subtraction it became -82.20508 and after standard deviation normalization it became -2.0215852. The result on one image is shown below:

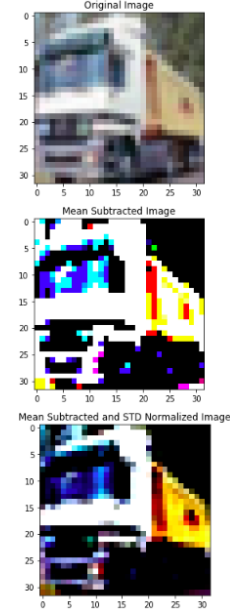


Figure 6: Result of mean subtraction and STD normalization

After doing this the network was retrained and this time the training loss had a final value of 0.0606 with a accuracy of 98.13%. The validation loss ended at 1.2952 with a final accuracy of 79.08%. The test set reported an accuracy of 77.34%. It slightly improved network performance. The image below shows the accuracy and loss curves:

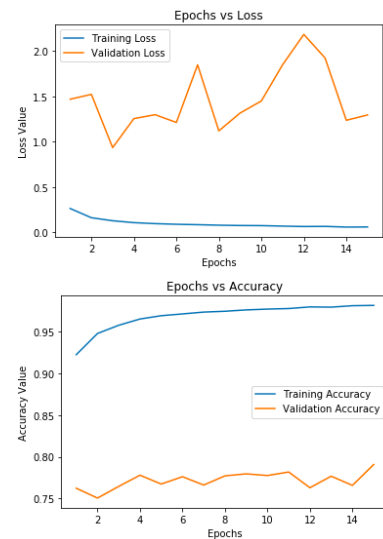


Figure 7: Loss and Accuracy curves after doing Mean subtraction and normalization

V. INITIALIZATION METHODS

A. Random Uniform and Zero Bias

Here I added random uniform initialization to all weights of the network and the biases as zero. With this initializer the network started off worse than before and in the same number of epochs it did not do as well as the network without initializer. The training loss had a final value of 0.1020 with a accuracy of 96.64%. The validation loss ended at 1.2535 with a final accuracy of 76.86%. The test set reported an accuracy of 77.00%. The loss and accuracy graphs are shown below.

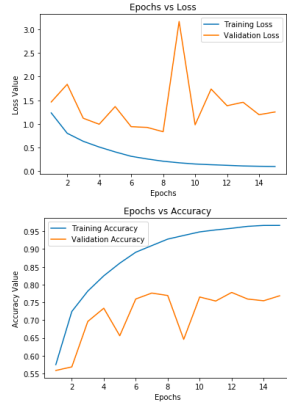


Figure 8: Loss and Accuracy curves with random uniform initialization with zero bias

B. Glorot Normal and Zero Bias

Here I added Glorot to all weights of the network and the biases as zero. With this initializer the network started off worse than without initializers and in the same number of epochs it did not do as well as the network without initializer. However it ended up performing better than the 'Random Uniform' initializer on the training data but worse on test and validation. The training loss had a final value of 0.0971 with a accuracy of 96.96%. The validation loss ended at 1.1313 with a final accuracy of 76.76%. The test set reported an accuracy of 76.23%. The loss and accuracy graphs are shown below.

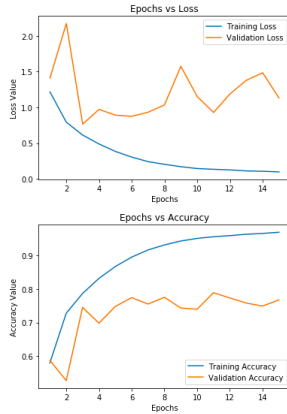


Figure 9: Loss and Accuracy curves with Glorot Normal initialization with zero bias

C. Ones and Zero Bias

Here I added ones initialization to all weights of the network and the biases as zero. With this initializer the network started off worse than before and in the same number of epochs it did not do as well as the network without initializer. It was also the worst initializer. The training loss had a final value of 2.0804 with a accuracy of 19.71%. The validation loss ended at 2.0614 with a final accuracy of 20.38%. The test set reported an accuracy of 20.88%. The loss and accuracy graphs are shown below.

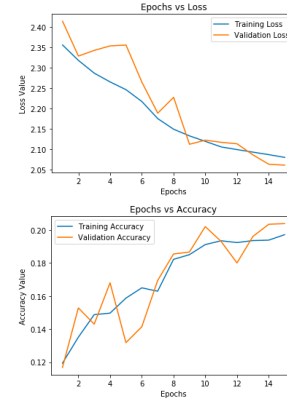


Figure 10: Loss and Accuracy curves with Ones initialization with zero bias

D. LeCun Normal and Zero Bias

Here I added the LeCun Normal initialization to all weights of the network and the biases as zero. With this initializer the network started off as the best one and in the same number of epochs it did better than the other initializers but not as good as network without an initializer. It was also the worst initializer. The training loss had a final value of 0.1008 with a accuracy of 96.78%. The validation loss ended at 1.1751 with a final accuracy of 76.88%. The test set reported an accuracy of 76.48%. The loss and accuracy graphs are shown below.

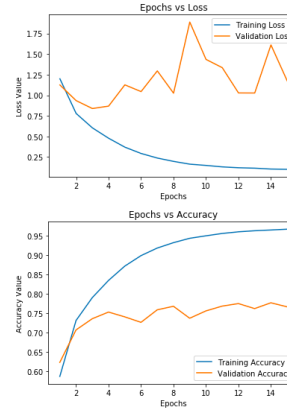


Figure 11: Loss and Accuracy curves with Ones initialization with zero bias

VI. DROPOUT

Here we added 3 dropout layers to the network after we removed the initializer code. The summary is shown below:

Layer (type)	Output Shape	Param #
conv2d_70 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization_77 (Batch Normalization)	(None, 32, 32, 32)	128
activation_117 (Activation)	(None, 32, 32, 32)	0
conv2d_71 (Conv2D)	(None, 30, 30, 32)	9248
batch_normalization_78 (Batch Normalization)	(None, 30, 30, 32)	128
activation_118 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_35 (MaxPooling)	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_72 (Conv2D)	(None, 15, 15, 64)	18496
batch_normalization_79 (Batch Normalization)	(None, 15, 15, 64)	256
activation_119 (Activation)	(None, 15, 15, 64)	0
conv2d_73 (Conv2D)	(None, 13, 13, 64)	36928
batch_normalization_80 (Batch Normalization)	(None, 13, 13, 64)	256
activation_120 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_36 (MaxPooling)	(None, 6, 6, 64)	0
dropout_2 (Dropout)	(None, 6, 6, 64)	0
flatten_17 (Flatten)	(None, 2304)	0
dense_49 (Dense)	(None, 512)	1180160
batch_normalization_81 (Batch Normalization)	(None, 512)	2048
activation_121 (Activation)	(None, 512)	0
dense_50 (Dense)	(None, 512)	262656
batch_normalization_82 (Batch Normalization)	(None, 512)	2048
activation_122 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_51 (Dense)	(None, 10)	5130
activation_123 (Activation)	(None, 10)	0
Total params: 1,518,378		
Trainable params: 1,515,946		
Non-trainable params: 2,432		

Figure 12: Network with Dropout

Adding dropout greatly improved the network and helped stop the issue of overfitting. In the end the network had a training loss of 0.7583 and a training accuracy of 75.31%. The validation loss was 0.6121 and validation accuracy was 79.28%. The final test set had an accuracy of 79.17%. The loss and accuracy curves are shown below:

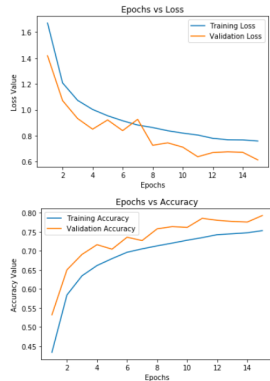


Figure 13: Network with Dropout

VII. LEARNING RATE AND OPTIMIZER

So far we have been using the 'RMSprop' optimizer but now we adjust there parameters.

A. RMSprop with higher learning rate

Here the learning rate was increased to 0.002. In the end the network had a training loss of 0.8161 and a training accuracy of 74.41%. The validation loss was 0.6391 and validation accuracy was 79.38%. The final test set had an accuracy of 78.78%. The network did not perform better with a faster learning rate. Loss and accuracy curve is shown below:

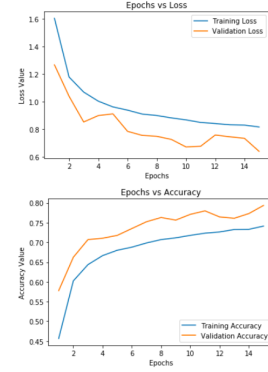


Figure 14: Loss and Accuracy curves with increased learning rate of RMSprop

B. SGD with default parameters

Here the optimizer was just switched to SGD. In the end the network had a training loss of 0.9607 and a training accuracy of 66.17%. The validation loss was 0.8582 and validation accuracy was 69.96%. The final test set had an accuracy of 69%. The network did worse that RMSprop. Loss and accuracy curve is shown below:

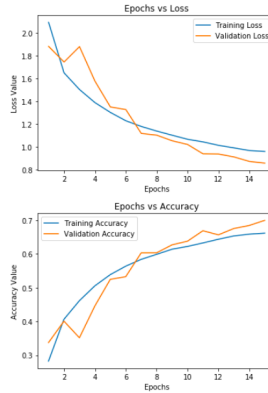


Figure 15: Loss and Accuracy curves with SGD

C. Adam

Here the optimizer was just switched to Adam. In the end the network had a training loss of 0.5902 and a training accuracy of 79.12%. The validation loss was 0.5763 and validation accuracy was 80.70%. The final test set had an

accuracy of 79.93%. The network with Adam is the best performing so far. Loss and accuracy curve is shown below:

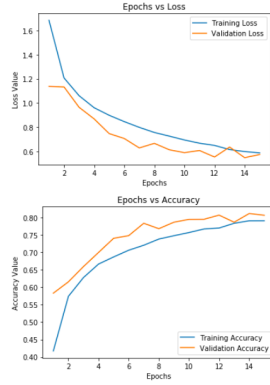


Figure 16: Loss and Accuracy curves with Adam

D. Nadam

Here the optimizer was just switched to Nestrov Adam. In the end the network had a training loss of 0.5835 and a training accuracy of 79.33%. The validation loss was 0.5379 and validation accuracy was 81.80%. The final test set had an accuracy of 80.38%. The network with Nadam is the best performing so far. Loss and accuracy curve is shown below:

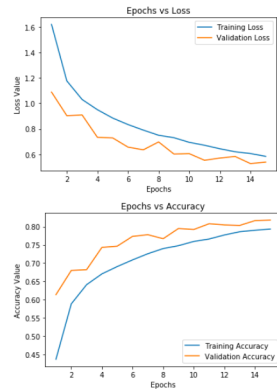


Figure 17: Loss and Accuracy curves with Nadam

E. Nadam with changed parameters

Here the Nestrov Adam learning rate was set to 0.008 and the decay is set to 0.01. In the end the network had a training loss of 0.6739 and a training accuracy of 76.78%. The validation loss was 0.6381 and validation accuracy was 78.12%. The final test set had an accuracy of 77.75%. The network with changed parameters did not perform better than the default one. Loss and accuracy curve is shown below:

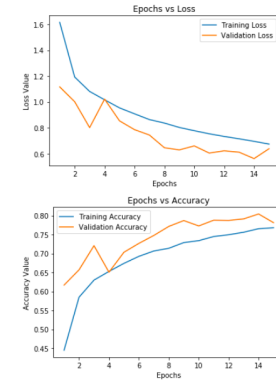


Figure 18: Loss and Accuracy curves with Nadam with changed parameters

VIII. BATCH SIZES

So far the batch size being used has been 32.

A. Batch size of 128

I used the network with Nadam and changed the batch size to 128. In the end the network had a training loss of 0.5078 and a training accuracy of 82.20%. The validation loss was 0.5273 and validation accuracy was 82.06%. The final test set had an accuracy of 81.39%. This network is the best performing so far. Loss and accuracy curve is shown below:

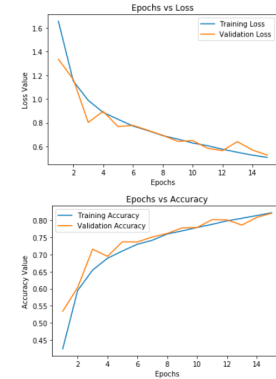


Figure 19: Loss and Accuracy curves with batch size of 128

B. Batch size of 512

Now the batch size is changed to 512. In the end the network had a training loss of 0.5369 and a training accuracy of 81.15%. The validation loss was 0.6154 and validation accuracy was 79.64%. The final test set had an accuracy of 78.93%. This network did not perform better. Loss and accuracy curve is shown below:

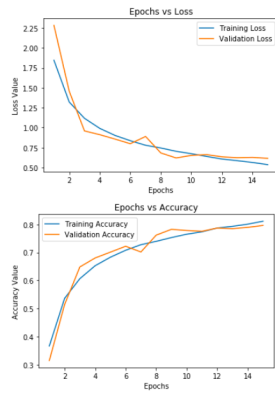


Figure 20: Loss and Accuracy curves with batch size of 512

IX. RESULTS

In the end the best network used the 'Nestrov Adam' optimizer and had a batch size of 128. Its summary is shown in figure 12 and results details are in VIII.A . Throughout the homework only 20 epochs were used. The final test accuracy was 81.39%