**EE258 PROJECT - I**

CLASSIFICATION OF CIFAR -10 DATASHEET

By

Parvathi Chandrasekhar (012474995)

I] DATASET:

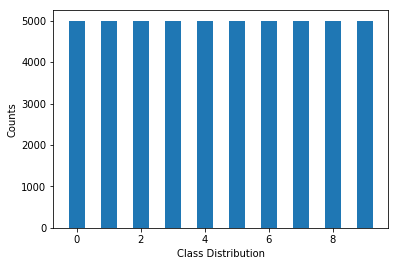
The data set used for this project is CIFAR-10. It is a subset of a larger set of 80 million tiny images. For the purpose of this project, we’re using a 10 classifier dataset, which is split into 50’000 training images and 10’000 test images. It is split into 5 training batches and 1 test batch containing 1000 images randomly selected from each class. The training images are also random within each batch but are of the same class. Each image is a 32x32 pixel colored image. The data is stored in arrays of dimension 10,000x3072 for each data batch. 3072 corresponds to 3 channels of Red, Blue and Green values, and 1024 values from 32x32. Each value is a 1 byte unsigned integer (uint8), and stores values from 0-255 (2^7) for the illumination level.

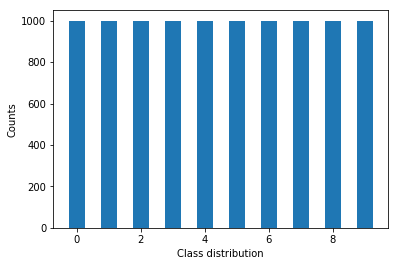
The batches are split into data and labels. Labels correspond to the classes that the images are classified into and range from 0-9. The dataset also has another file called “batches.meta” that contains the class names corresponding to the label numbers.

We used the “unpickle” Python module to convert the byte stream of data into an object hierarchy to be used in Jupyter. With this, we concatenated the 5 training batches into one “X\_train” batch containing 50’000 images, and assigned the test batch into a variable “X\_test”. The labels we loaded into “y\_train” and “y\_test”.

2] METHODOLOGY:

Then, we obtained the distribution of the training and test data, plotted them in graphs.

Figure 1. Training Distribution [Value/Counts vs Classes]

Figure 2. Test Distribution [Values/Counts vs Classes]

Here we observe that the images are indeed equally distributed between the classes. We started with reshaping the data from 50’000x32x3x3 to 50’000x3072.

The next step was to split the training data of 50’000 images into training data and validation data with the split percentage being 90% as training and 10% validation. This gives us 45’000 training images and 5’000 validation images.

There is no need to shuffle this data as the images are ordered randomly as it is and do not follow any ordering pattern.

The next step was to normalize the data, which is achieved by scaling it. We used the StandardScaler, and this is done to make the individual features to make them look more or less like standard normally distributed data.

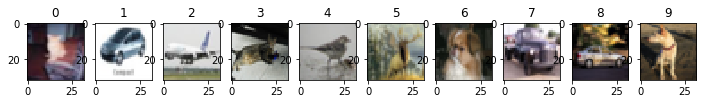


Figure 3. Unscaled Images



Figure 4. Scaled Images

The classifier used here is called DNNClassifier, or the Deep Neural Network Classifier which is a fully connected, feed forward multi layer perceptron model. This model allows the user to adjust the number of hidden layers, neurons per layer, batch size used to obtain weights, optimizer, the number of iterations or steps that the model runs for. These were our main hyperparameters in mind which we changed to receive better outputs. The number of output classes, feature columns are also passed as arguments which are fixed and are not changed.

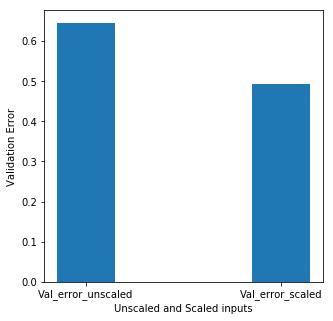
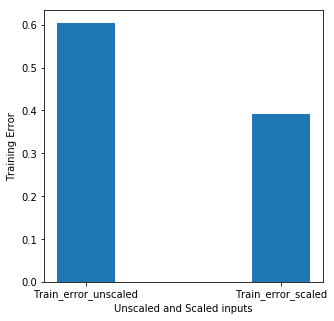
The unscaled inputs [X\_train and y\_train] and scaled [X\_train\_s and y\_train] are first run through the model to observe the performance analysis of scaled inputs. There is a definite performance increase for scaled inputs and therefore we continue using scaled inputs for our model.

Figure 5. Validation error difference in unscaled and scaled inputs

The training error differences are as follows:

 Figure 6. Training error between unscaled and scaled inputs

3] SIMULATIONS

Now, we change only the training set size [after we’ve split training = 45’000 images, validation = 5000 images] and epochs [1, 5 and 10 as multipliers of 1100 steps], just to observe the performance of having more data and more iterations to get better results in deep learning. We set the batch size to a default value of 50, and the neurons to 1 hidden layer, and 500 neurons in that layer. DNNClassifier by default uses RELU as its activation function.

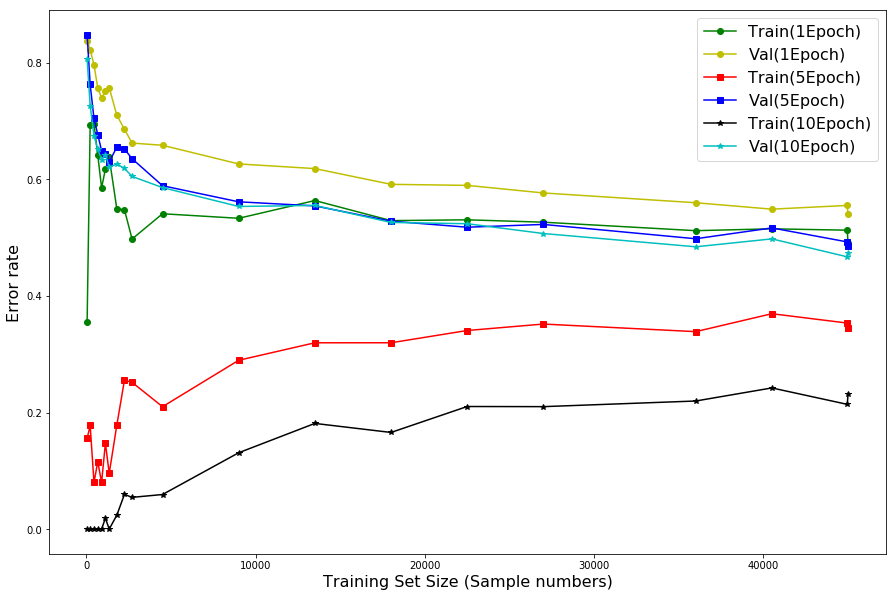


Figure 7. Observing validation and training error when changing Training size and epochs

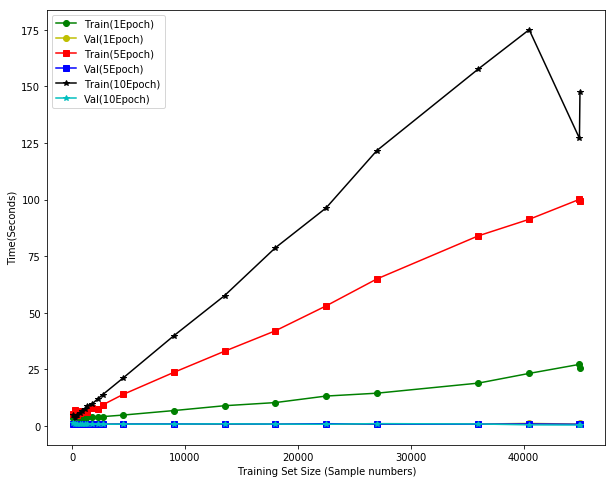


Figure 8. Time taken for the three epochs specified

3.1] CHANGING PARAMETERS:

3.1.1] ACTIVATION FUNCTION:

The first parameter to be changed was the activation function. The neuron set size [only one hidden layer and a default of 50 as batch size] is varied while checking for different activation functions.

Activation functions used were :

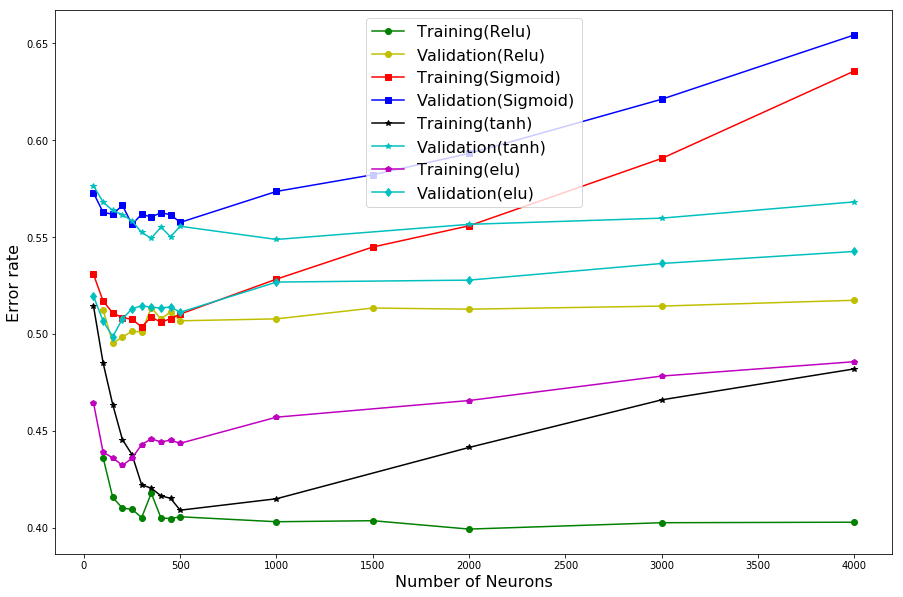
1)RELU

2)SIGMOID

3)TANH

4)ELU

The below figure gives the comparison of performance [training and validation error] for each activation function while varying the neuron set size.

Figure 9. Comparison of activation functions while changing neurons

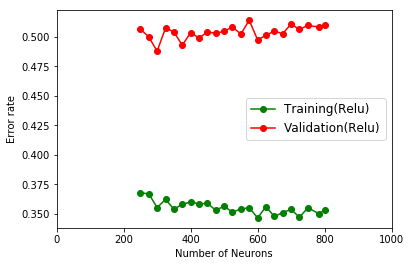
From the above graph, it is evident that RELU is the most appropriate for our 10 class classifier dataset as it gives the lowest validation and training error.

Figure 10. Number of neurons vs Error for RELU

After determining RELU works best, we vary number of neurons once again for RELU to check the validation error and training error.

3.1.2] BATCH SIZE

Now that RELU is determined as the best possible activation function, the batch size is determined by varying the number of neurons [only one hidden layer], and checking for the validation accuracy [1 – validation error].

Batch Size

Neurons

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 500 | 1000 | 1500 | 2000 | 3000 | 4000 |
| 50 | 0.4178 | 0.4124 | 0.4046 | 0.3942 | 0.3902 | 0.3842 |
| 100 | 0.4876 | 0.4796 | 0.4752 | 0.4654 | 0.4646 | 0.4554 |
| 200 | 0.508 | 0.5094 | 0.5096 | 0.5084 | 0.4972 | 0.4950 |

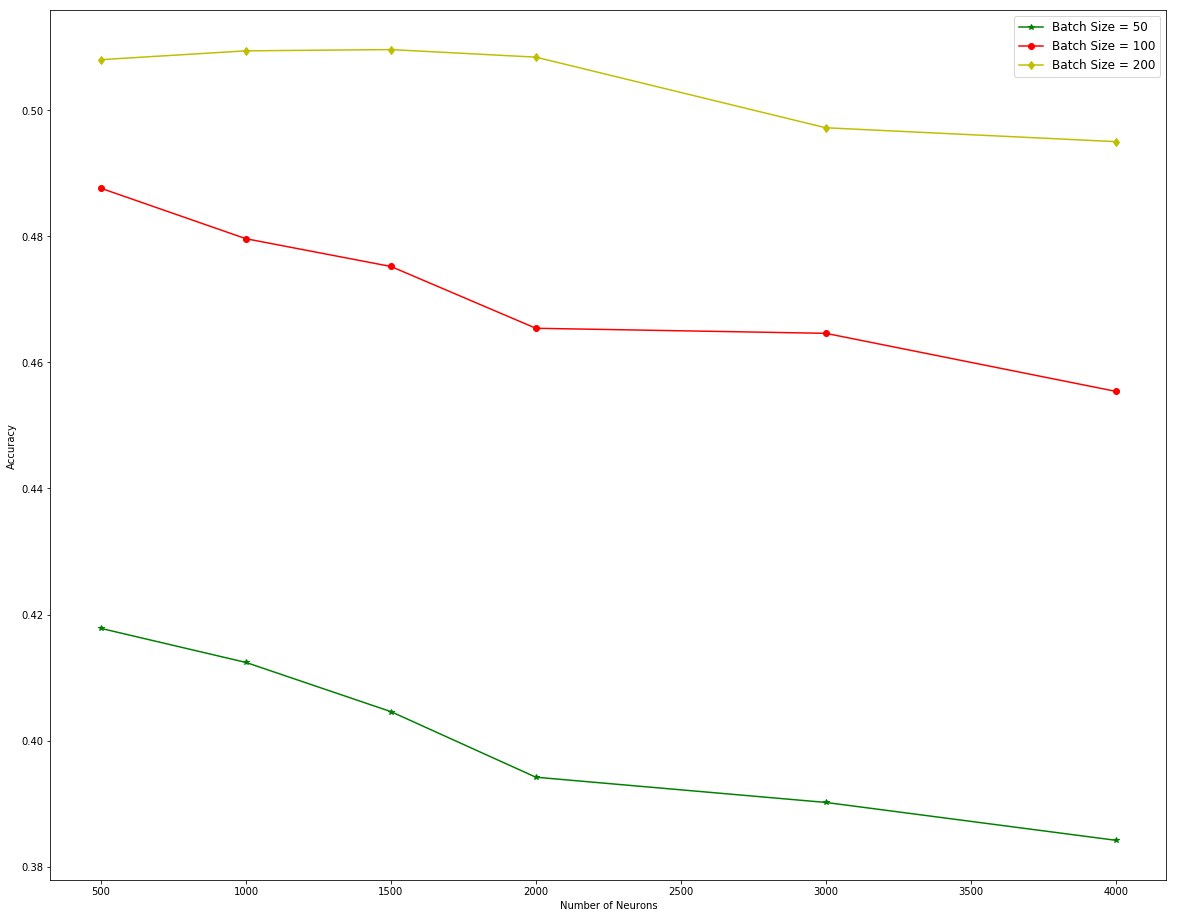


Figure 11. Validation Accuracy vs Number of Neurons

From this, it is seen that with a batch size of 200, validation accuracy is higher than 100 or 200. Using 200 as the batch size, other parameters are changed.

3.1.3] HIDDEN LAYERS AND NEURONS

After fixing the activation function as RELU and batch size as 200, the number of neurons and hidden layer were changed. Setting hidden layers = 2 and changing neurons:

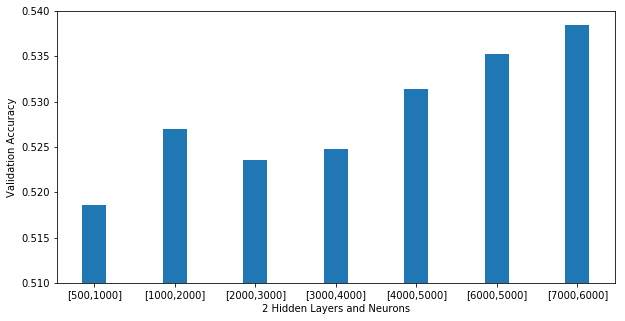


Figure 12. Two Hidden Layers and Neurons vs Validation Accuracy

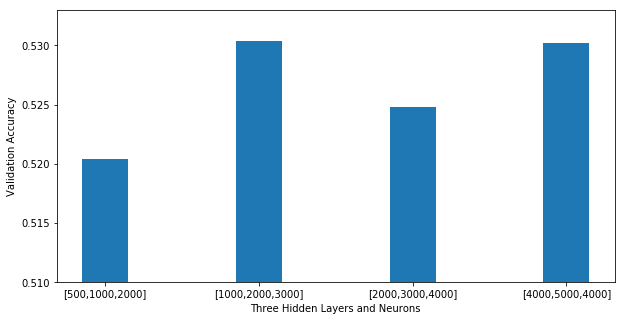
Setting hidden layers as 3 and varying neurons to check validation accuracy:

Figure 13. Three Hidden Layers and Neurons vs Validation Accuracy

From the graph it can be seen that even though the number of hidden layers are increased, the accuracy doesn’t increase as much.

So, the best output so far is setting the number of hidden layers as 2 and the number of neurons as [7000,6000].

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Neurons  Val Acc | [500,  1000] | [1000,  2000] | [2000,  3000] | [3000,  4000] | [4000,  5000] | [6000,  5000] | [7000,  6000] |
|  | 0.5186 | 0.527 | 0.5236 | 0.5248 | 0.5314 | 0.5352 | 0.5384 |

Table 2. Val Acc vs Neurons [2 Hidden Layers]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Neurons  Val Acc | [500,  1000,  2000] | [1000,  2000,  3000] | [2000,  3000,  4000] | [4000,  5000,  4000] |
|  | 0.5204 | 0.5304 | 0.5248 | 0.5302 |

Table 3. Val Acc vs Neurons [3 Hidden Layers]

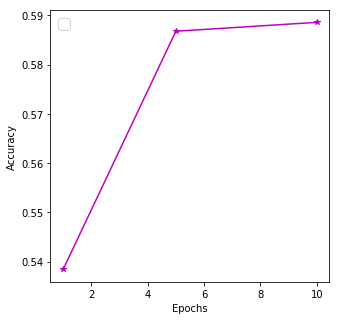
Here it is evident, using 2 hidden layers and [7000, 6000] neurons give the best validation accuracy of 0.5384.

3.1.4] EPOCHS

Setting batch size = 200, Hidden Layers = 2, Neuron Sizes = [7000,6000] and Activation Function = RELU, we’re changing the epoch size to get better accuracy in terms of the validation data.

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs | 1 EPOCH | 5 EPOCHS | 10 EPOCHS |
| Val Acc | 0.5384 | 0.5868 | 0.5886 |

Table 4. Number of Epochs vs Validation Accuracy

Figure 14. Validation Accuracy vs No. of Epochs

From the table and graph, it’s seen that the validation accuracy increases as the number of epochs increase, upto 0.5886 for 10 epochs. We also obtain the train accuracy as 1.0.

From this, we set our parameters accordingly for the test data and fit it to the model.

4] RESULTS

Batch Size = 200

Activation Function = RELU

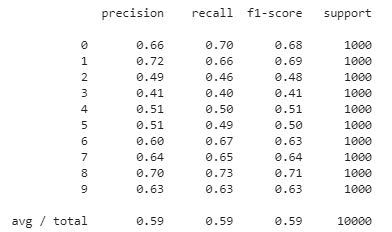
Hidden Layers = 2

Neuron Size = [7000,6000]

Epochs = 10

We then fit the DNNClassifier model with these parameters before predicting the accuracy of the test data. We then obtain the test accuracy as 0.5896.

Classification Report:



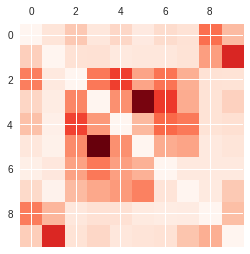
Confusion Matrix:

Figure 16. Confusion Matrix