Report
Wayne Chew Ming Chan (9071997606)
Sparsh Agarwal (9075905142)
CS 638
Lab-3

Introduction

The aim was to predict the categories of various images from a set of images.
We used back propagation in convolutional neural networks. We used both convolutional and pooling layers. Further for experimentation we implemented dropout. We implemented the backpropagation using multiple hidden layers to train our weights of multiple HUs.
The output is represented using confusion matrix which represents how our network starts learning the categories and eventually we get a accuracy of 85%.for the test set and 99.5% accuracy for the train set.
The accuracy varies mainly on following factors:
1. Learning rate
2. Number of hidden units
3. Number of filters used for the convolutional layer
4. Dropout rate

Team and Work Done:
Wayne: Work mostly on writing the code for the Convolutional Neural Network Classifier, One Hidden Layer Classifier and Single Perceptron Classifier. Focus on getting the best accuracy for the project.
Sparsh: Worked on putting all modules together and implementing dropout and experimenting with various scenarios, further worked on developing pooling and convolutional layers finally giving outputs to hidden layer for further computations. Also tried to experiment with different kinds of filters for images.

Data:

There are 554 training examples, 180 testing examples and 178 tuning examples. Each of which is a mix of images from all the 6 categories.
The 6 categories are-

1. Flowers
2. Star Fish
3. Watch
4. Giant Piano
5. Airplane
6. Butterfly

The images are in different orientations to help learn the neural network properly.

Layers:

We have used the following layers

Convolution-Pooling-Convolution-Pooling-hiddenlayer-output

Where neural network has one hidden layer. The entire network does forward propagation and backpropagation to learn the weights attached to each element.

Layer1: Convolutional Layer (10 Filters)

Weights: 5 x 5 x 4
Input Dimensions: 32 x 32 x 4
Output Dimensions: 28 x 28 x 10

Layer2: Pooling Layer (MAX)

Weights: 2 x 2 (Window size)
Input Dimensions: 28 x 28 x 10
Output Dimensions: 14 x 14 x 10

Layer3: Convolutional Layer (20 Filters)

Weights: 5 x 5 x 20
Input Dimensions: 14 x 14 x 10
Output Dimensions: 10 x 10 x 20

Layer4: Pooling Layer (MAX)

Weights: 2 x 2 (Window Size)
Input Dimensions: 10 x 10 x 20
Output Dimensions: 5 x 5 x 20

Layer5: Output Layer (With a hidden layer)

Input Dimensions: 5 x 5 x 20
Output Dimensions: 6 x 1 x 1

Below is the schema of our deep neural network layers.

Implementation of Classifiers

We implemented all three classifiers:

1) Single Perceptron Classifier
    a) This gives the worse results of around 50% - 60% accuracy. But we are surprised that it actually manages to do some prediction.
2) One Hidden Layer Classifier
    a) After adding one hidden layer of around 250 neurons. We are able to reach around 80% accuracy but it takes forever to train.
3) Convolutional Neural Network Classifier
    a) This is the best classifier and the training speed is a lot faster than the one hidden layer classifier.
    b) The design:
    Input -> CONV -> POOL -> CONV -> POOL-> HIDDEN -> OUTPUT
    c) The convolutional layer and the pooling layer significantly reduces the number of weights that is required for the neural network. And this speeds up the training compared to the one hidden layer classifier. We manage to get a stable accuracy of 85%.
    d) After some trial and error, we decided to use 10 filters for the first convolutional layer and 20 filters for the second convolutional layer. At first, we tested it with 50 and 50 each but it takes to long to train. Then, we slowly reduced it to get the best balance between training speed and accuracy. We found out that 10 and 20 each works the best for our neural network.
4) And of course, basic deep learning techniques such as early stopping are used for all classifiers.

Dropout

Using Dropout makes learning curve varying, the variance in accuracies increases for each epoch. The final highest accuracy might increase a bit but during the process of learning it can vary substantially from the required curve.

Finally for experimenting with various values, we have incorporated dropout in the code such that the required dropouts percentage can be passed to the neural network and the layers adapt accordingly. If dropout is passed as 0.5, means that 50% weights will only be considered on random basis.

Results

We calculated accuracies by training for various % of training sets and with different dropout rates.

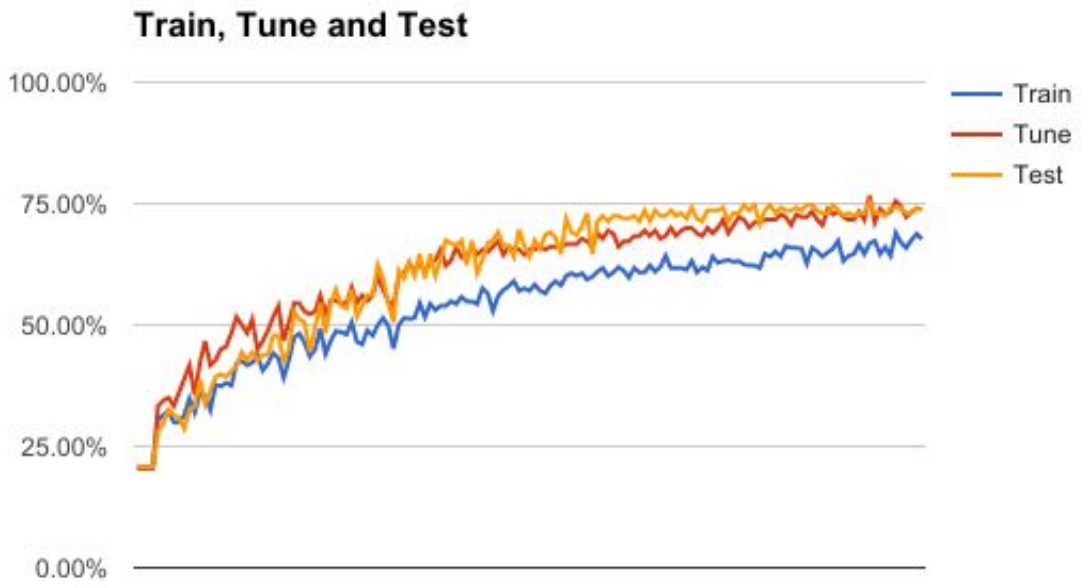
The neural network can be trained depending on our requirements, we can consider different **fractions of training data** to be considered for training the neural network. On the other hand, **dropout can be varied** to experiment with various conditions and their outcomes

Below are some test results based on our experiments:

Here we can see that as the epochs proceed, the accuracy of training, tuning and test set increase. The interesting observation is that the improvement in accuracy is more in the beginning and as we reach higher accuracies, the rate of improvement decreases. Further we can see that since early stopping with patience is used to determine optimal weights, the neural network keeps on learning until we keep on improving accuracy. When accuracy starts to decrease and the difference goes beyond patience, the neural network stops training and a final set of optimal weights is obtained.

1. **dropout 10% and 100% training set**

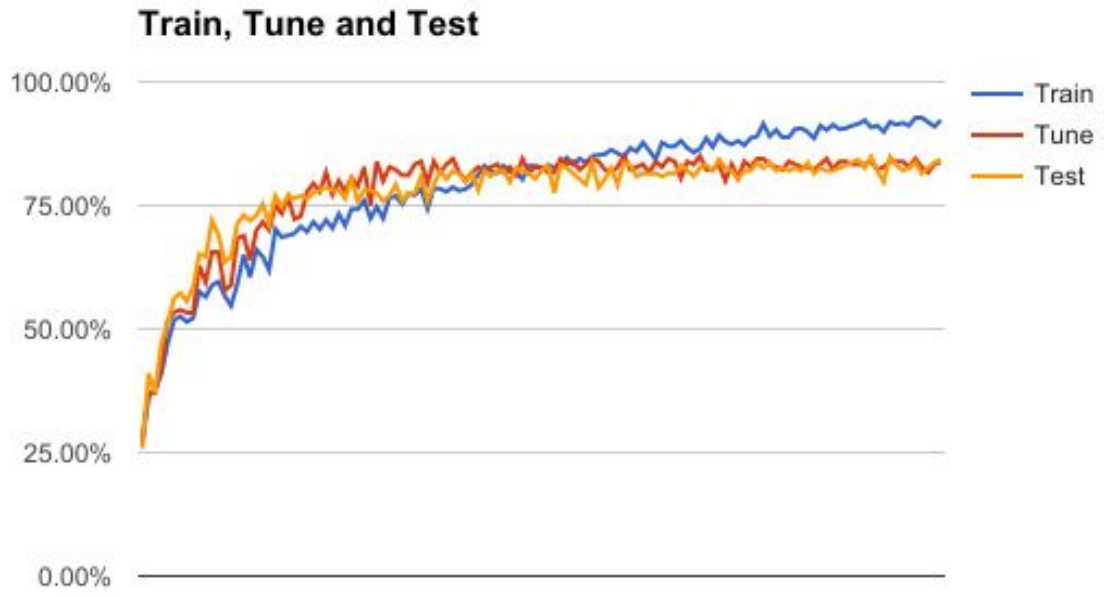
Train	Tune	Test
67.69%	**73.89%**	**73.60%**



This example takes into consideration only the 90% nodes of hidden layer as dropout is 10% and we use entire training set in this experiment.
We get a final test accuracy of 77%

2. **dropout 0% and 100% training set**

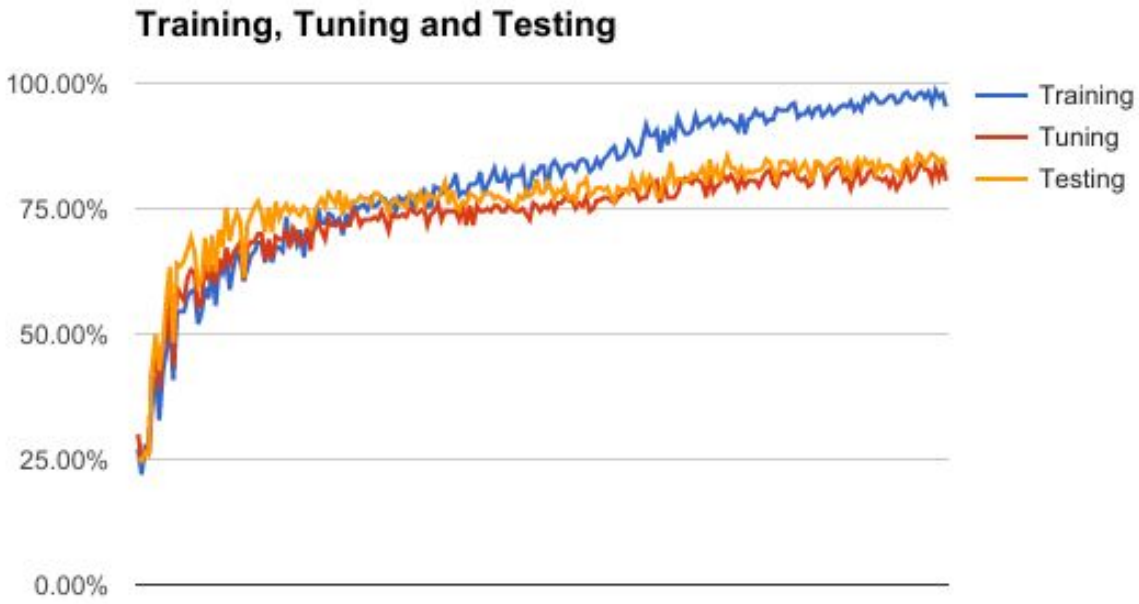
Train	Tune	Test
92.24%	**83.89%**	**84.27%**



This example takes into consideration all nodes of hidden layer as dropout is 0% and we use entire training set in this experiment.
We get a final test accuracy of 84%. The increase in accuracy is because now all the nodes of hidden layer which should contribute to prediction are being considered.

3. dropout 0, 50% training set

Training	Tuning	Testing
95.31%	**80.56%**	**83.71%**



This example takes into consideration all nodes of hidden layer as dropout is 0% but we use only 50% training examples.
We get a final test accuracy of 84%. Since the accuracy is same as achieved when 10% training set is used, it depicts that our neural network is capable of learning and predicting correctly even with fewer example.

4. Training accuracy vs epochs

Epoch	Training Set Accuracy	Training Set Mean Squared Error
185	99.46%	1.27%

This graph shows the training set accuracy and the training set mean squared error. The x-axis is the epoch. After experimenting using different settings, we find that the training set accuracy increases greatly at the first few epochs, then the learning rate slows down. After around 100 epochs with batch size of 5, the learning rate becomes flat with an accuracy of 95% and above. It stops learning when the accuracy is 99.5% with 3 wrong prediction. The mean squared error of the training set also stops at 2% and won't go below that. We might need more convolutional layers or more hidden units to get it up to 100% but that will take too long to train.

5. Results for training using GrayScale Images

Convolutional Neural Network Deep Training
Data Info: **useRGB = false**, imageSize = 32, fractionOfTrainingToUse = 1.00
Using: ETA = 0.010000, Dropout = 0.00, Patience = 50, BatchSize = 5

Best Tuning Set Accuracy: 78.8889% at Epoch: 151
********** FINAL RESULTS **********
Test Set result for convolutional neural network:

---------- Confusion Matrix ----------

39	0	1	0	0	1
1	5	6	0	1	5
0	1	32	1	2	1
1	0	5	13	0	0
0	0	6	2	5	4
1	2	2	0	0	41

Accuracy: 75.8427%
Mean Squared Error: 12.6733%

As expected, training using grayscale images give worse results (75%) compared to training using RGB (85%). However, training using grayscale images is a lot faster compared to training using RGB.

Sidenote: We tried training using 8x8 and 16x16 images but the results are not good. We tried using an images size of 64 x 64 but it takes too long to train and we think that the slow speed is not worth it.

6. Results for dropout = 0.75
Convolutional Neural Network Deep Training
Data Info: useRGB = true, imageSize = 32, fractionOfTrainingToUse = 1.00
Using: ETA = 0.010000, **Dropout = 0.75**, Patience = 50, BatchSize = 5

Best Tuning Set Accuracy: 86.1111% at Epoch: 56
********** FINAL RESULTS **********
Test Set result for convolutional neural network:
---------- Confusion Matrix ----------

| 41 | 1 | 0 | 0 | 0 | 0 |

| 0 | 7 | 1 | 0 | 3 | 0 |

| 0 | 5 | 27 | 0 | 2 | 1 |

| 0 | 0 | 2 | 16 | 0 | 0 |

| 0 | 3 | 5 | 0 | 12 | 2 |

| 0 | 2 | 2 | 3 | 0 | 43 |

Accuracy: 82.0225%
Mean Squared Error: 8.2461%

Average of 22 seconds per Epoch.
Took 37 minutes and 26 seconds to train.

Interestingly, a dropout of 75% at the hidden layer gives an accuracy of 82%. Although the training speed is slightly faster than training with 0% dropout, the accuracy is 4% lower compare to 0% dropout.

Best Results (Tune for Speed too, took around 19 seconds per Epoch, 36 minutes overall)
Convolutional Neural Network Deep Training
Data Info: useRGB = true, imageSize = 32, fractionOfTrainingToUse = 1.00
Using: ETA = 0.010000, Dropout = 0.00, Patience = 40, BatchSize = 5

Best Tuning Set Accuracy: 88.8889% at Epoch: 69
********** FINAL RESULTS **********
Test Set result for convolutional neural network:
---------- Confusion Matrix ----------

| 41 | 2 | 0 | 0 | 0 | 0 |

| 0 | 10 | 3 | 0 | 0 | 1 |

| 0 | 3 | 29 | 1 | 4 | 0 |

| 0 | 0 | 0 | 17 | 0 | 0 |

| 0 | 1 | 4 | 0 | 10 | 1 |

| 0 | 2 | 1 | 1 | 3 | 44 |

Accuracy: 84.8315%
Mean Squared Error: 7.2866%

Average of 19 seconds per Epoch.
Took 35 minutes and 58 seconds to train.

After optimizing the code and removing extra implementations. The final version of our Convolutional Neural Network gives an accuracy of around 85% for the test set. It uses RGB and 32 x 32 image size with a 0.01 learning rate and 0% dropout. It takes 19 seconds to run for each epoch of batch size 5.