# ASSIGNMENT 4: LEARNING

Team Members: Surbhi Paithankar, Apurva Gupta and Hasika Mahtta(guptaapu-hmahtta-spaithan-a4)

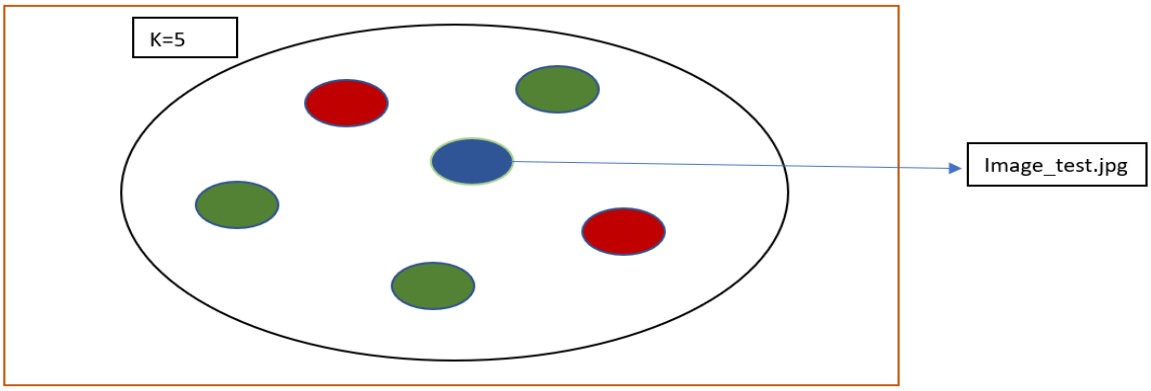
## K-NEAREST NEIGHBOURS ALGORITHM

INTRODUCTION

KNN is one of the simplest non-parametric classification algorithm i.e. it does not make any assumptions about the underlying data distribution. It does not create any generalized model using training data. Rather, it uses all the training data in testing phase. We calculate the distance of each test point from each training data point and take k nearest training data points. Then, we take the majority voting and based on that we assign label to that test data point.

**For Example:** Suppose we have a test image Image_test.jpg with feature values [1 0 3 5].

We take k = 5 closest training images to this test image. We can measure distance using various ways like Euclidian distance, Manhattan distance etc.




Suppose 5 closest images and their orientations are:-

Image_train1.txt [2 0 1 1] with class 0 degree
Image_train2.txt [1 0 0 1] with class 180 degree
Image_train3.txt [2 0 0 0] with class 90 degree
Image_train4.txt [1 2 2 1] with class 0 degree
Image_train5.txt [2 4 0 0] with class 0 degree

In these 5 images, we take majority voting.
0 degree has 3 votes, 90 degree has 1 vote, 180 degree has 1 vote.
Thus, we assign 0 degree label to Image_test.jpg.

OUR IMPLEMENTATION

Command line call for train: python orient.py train <training file> <model file> nearest
Command line call for test: python orient.py test <testing file> <model file> nearest

K NEAREST NEIGHBORS ALGORITHM:

**For Training**

1. Since KNN takes whole training file as model, we stored whole training file in model file.

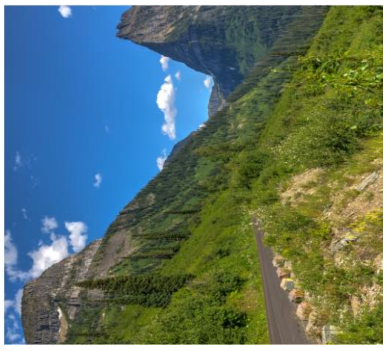
**For Testing**

1. Calculated Manhattan distance of each test point from each training image.
2. Sorted all the distances in increasing order
3. Took k smallest distance training points
4. Found majority class label amongst those k images
5. Assigned that label to test image
6. Repeat steps 1 to 5 for all test images

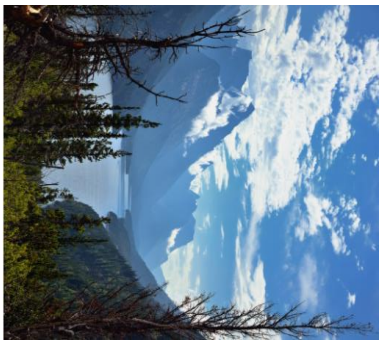
RESULTS

We recorded some of the images which were classified correctly and incorrectly to see that what went wrong.

Some of the correctly classified images are:



Predicted Output: 270°
Actual Output: 270°



Predicted Output:90°
Actual Output:90°

Predicted Output:0°
Actual Output:0°

Some of the incorrectly classified images are:

Predicted Output:$180^0$
Actual Output:$0^0$

Predicted Output:$180^0$
Actual Output: $270^0$

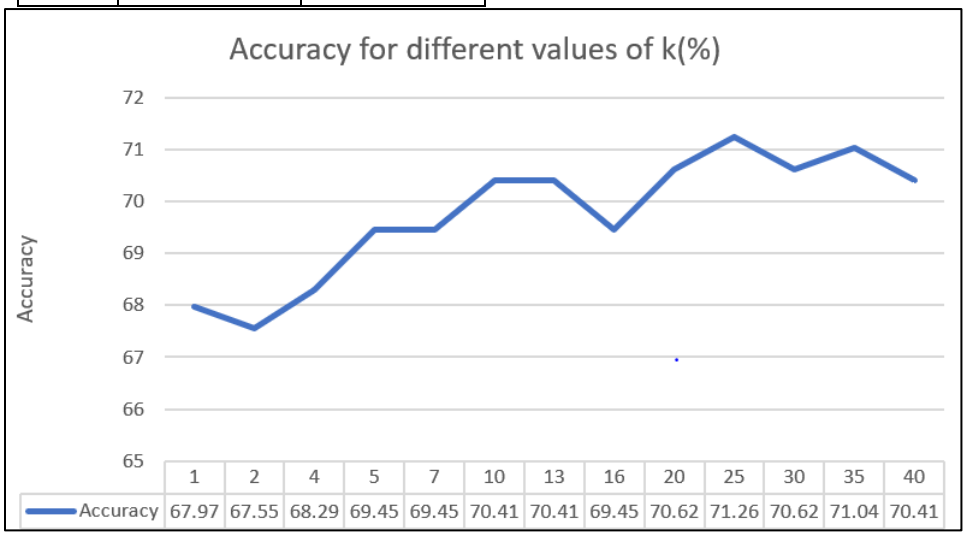
Predicted Output:180°
Actual Output:270°

In the above example, we can observe that images which have sky above and green grass or land below are classified correctly. Whereas, the images which seems to be symmetrical get classified incorrectly.

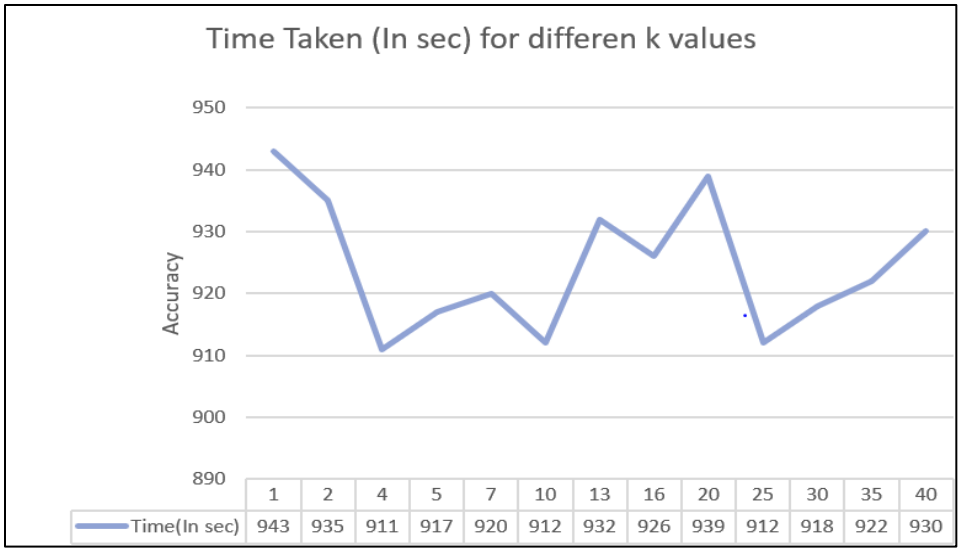
PARAMETER OPTIMIZATION

We tried different values of k, to find the k value which gives best results. Here, are our results:

| K | performance | Time(In sec) |
|---|---|---|
| 1 | 67.97 | 943 |
| 2 | 67.55 | 935 |
| 4 | 68.29 | 911 |
| 5 | 69.45 | 917 |
| 7 | 69.45 | 920 |
| 10 | 70.41 | 912 |
| 13 | 70.41 | 932 |
| 16 | 69.45 | 926 |
| 20 | 70.62 | 939 |
| 25 | 71.26 | 912 |
| 30 | 70.62 | 918 |
| 35 | 71.04 | 922 |
| 40 | 70.41 | 930 |




We can observe that at K=25 we get the highest accuracy. Performance keeps on fluctuating with different values of K.




W can observe that it takes almost same time for KNN to run on different values of k. This is because the difference comes only while sorting the elements which does not contribute much to the running time.

We would recommend setting k value to 25, for best performance of K neighbors algorithm.

## ADABOOST ALGORITHM

INTRODUCTION

Adaboost is also known as Adaptive boosting. The basic idea behind it is constructing learning algorithms ('weak learners') that are combined into a weighted sum that represents the final output of the boosted classifier. The weak classifiers are also referred as decision stumps.

A typical Adaboost classifier performs binary classification i.e. it predicts whether a test example belongs to class1 or class2. We can extend this feature for multiclass classification using One vs Many method or One vs One method.

In One vs many, we consider one class to be say 0 degree and the other class to be anything that is not 0 degree. So basically, we use n number of Adaboost classifiers for n-class problem. In our case, we would need to use 4 Adaboost classifiers in conjunction for classifying an image into either of the four classes (0,90,180,270).

In one vs one, we make Nc2 number of binary classifiers. In this we make classifiers with all possible class combinations, like 0 vs 90, 90 vs 180…etc. Later we assign the class that get majority number of votes that are output by each binary classifier. In case of tie, we random pick up one of the class amongst the contenders.

We have implemented One vs One algorithm.

## WEAK CLASSIFIER

We follow a simple approach to build a weak classifier based on pixel values. Ideally we can compare each pixel in our 8*8*3 image with every other pixel. This would require 192^2 comparisions.

As this would significantly increase the running time of our algorithm, we random pick 500 pair of pixels and compare them with one other.

Let us say for decision Stump 1, we have (a1, b1) pair.
If Pixel_Value[a1] > Pixel_Value[b1] in training example say Ti:
Orientation of Ti is the orientation that maximum examples in training file have whenever pixel a1 greater than pixel b1.
Else, Orientation of Ti is the orientation that maximum examples in training file have whenever pixel a1 lesser than pixel b1.

For example: say we have a1 = 14, b2 = 100, then we compare if for a training example Ti, value of pixel at $14^{th}$ position is greater or less than value at $100^{th}$ position.
Let us assume we observe Pixel_Val[14]>Pixel_Val[100].
Suppose there are 1000 training images having Pixel_Val[14]>Pixel_Val[100]. Out of these we see that 600 images have orientation as 0 degrees. Then we assign the training example Ti as 0 degrees.
Note: We use subset of training examples with only the orientation of either class1 or class 2 for a binary adaboost classifier (class1 vs class2)

## WEIGHTS

Initially we are going to assign a weight of 1/no. of training examples to each training image. Later whenever a pixel classified with the correct orientation, we reduce its weight. This increases the importance for the wrongly classified images.

## PREDICTION OF ORIENTATION FOR TEST IMAGES

The training phase of the algorithm returns a hypothesis containing k pairs of decision stump and their corresponding weights.
For a given test image say Ti, we get predicted orientation using K pairs. Later we compute the total weights. The orientation with maximum total weight is our answer.
For example: Out of 10 decision stumps for image test1.png, 6 predicts it to be 0 degrees and 4 predict it to be 90 degrees. So we add the corresponding weights of the 6 decision stumps(say 4.2) versus weights of the 6 decision stumps(say 7.5). As the total weight of 90 degrees is more, it gains the weighed majority.
Therefore we assign Test1.png as 90 degrees.

## MULTI-CLASS CLASSIFICATION

As we are using one vs one classifier, we will have 6 classifiers as follows:

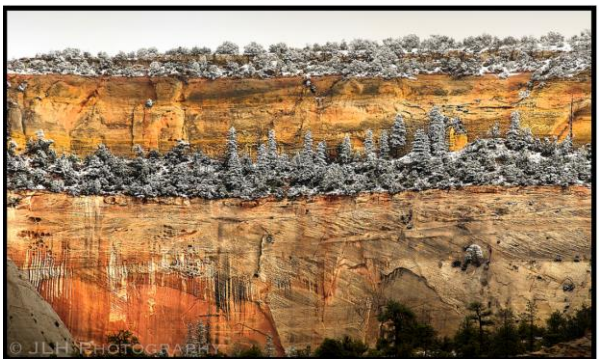
Adaboost(0,90)
Adaboost(0,180)
Adaboost(0,270)
Adaboost(90,180)
Adaboost(90,270)
Adaboost(180,270)
Each classifier predicts one of the two classes. We classify the class that has maximum vote amongst all binary classifiers.
For instance: Say we have a test image Test1.png with following results:

Adaboost(0,90) predicts 0
Adaboost(0,180) predicts 180
Adaboost(0,270) predicts 270
Adaboost(90,180) predicts 180
Adaboost(90,270) predicts 270
Adaboost(180,270) predicts 180

We observe that 3/6 classifiers voted for 180 degrees. Hence, we classify the image Test1.png as 180 degrees.

## RESULTS

We observe surprisingly good performance of adaboost, even after using just 10 weak classifiers. When weak classifiers are used in conjunction with each other, they together prove out to build a strong classifier.

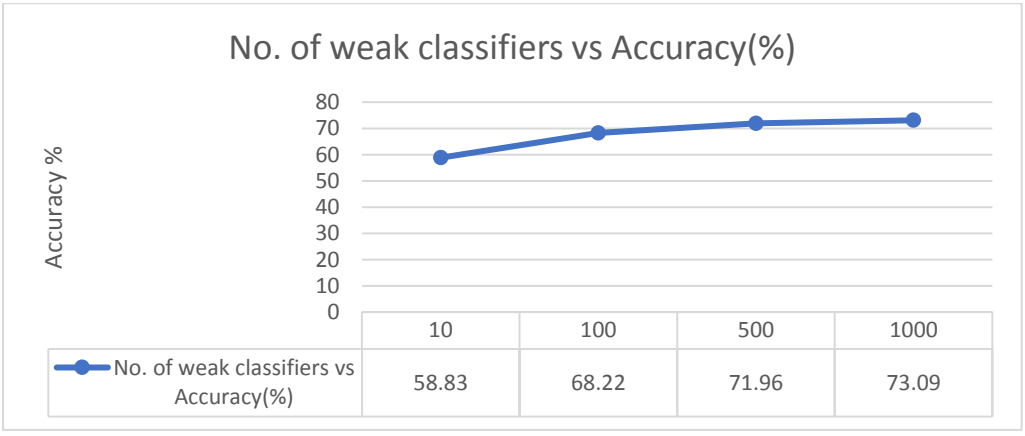


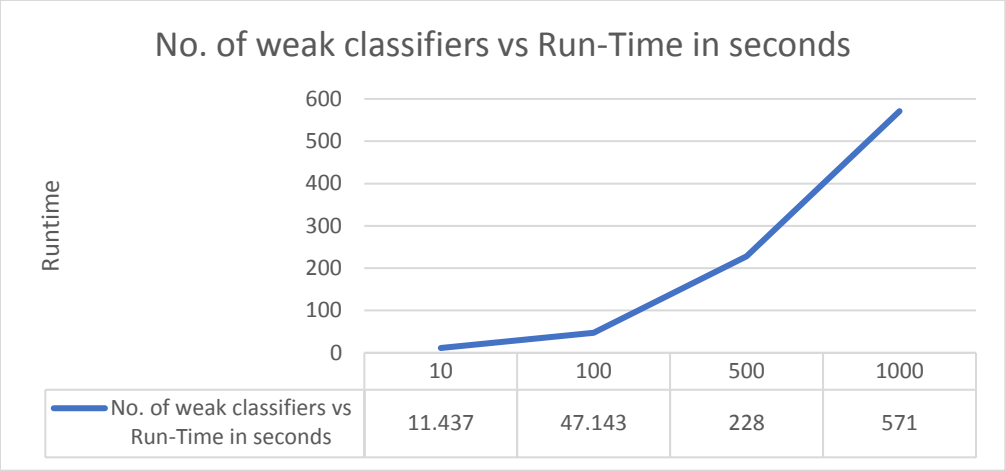
Predicted output: 0 degrees
Actual Output: 0 degrees
Explanation: Any landscape images are normally seen to have either sky blue or cloud white colors at the top whereas dark colors at the bottom(may be due to soil,trees etc). The algorithm learns this behavior by pixel comparison during the training phase. Hence it correctly classifies the test image as 0 degrees.

Predicted output: 0 degree
Actual output: 270 degrees
Explanation: In this image, the algorithm predicts 0 degrees. We know that in general, any landscape image has sky blue on the top. So, the algorithm recognizes the blue color in the top part of the image as blue sky instead of water. Hence it misclassifies the image as 0 degree.

## PARAMETER OPTIMIZATION

We observe an increase in accuracy with the increase in the number of weak classifiers. This is because with more pairs, we would have a better visual information for any image. This would result in better construction of hypothesis for classification.






We observe a significant increase in runtime as well as accuracy with increase in the number of decision stumps.

We would recommend using 1000 decision stumps to any potential client, as we get a good amount of accuracy in a reasonable amount of training time.

## ARTIFICIAL NEURAL NETWORKS

INTRODUCTION

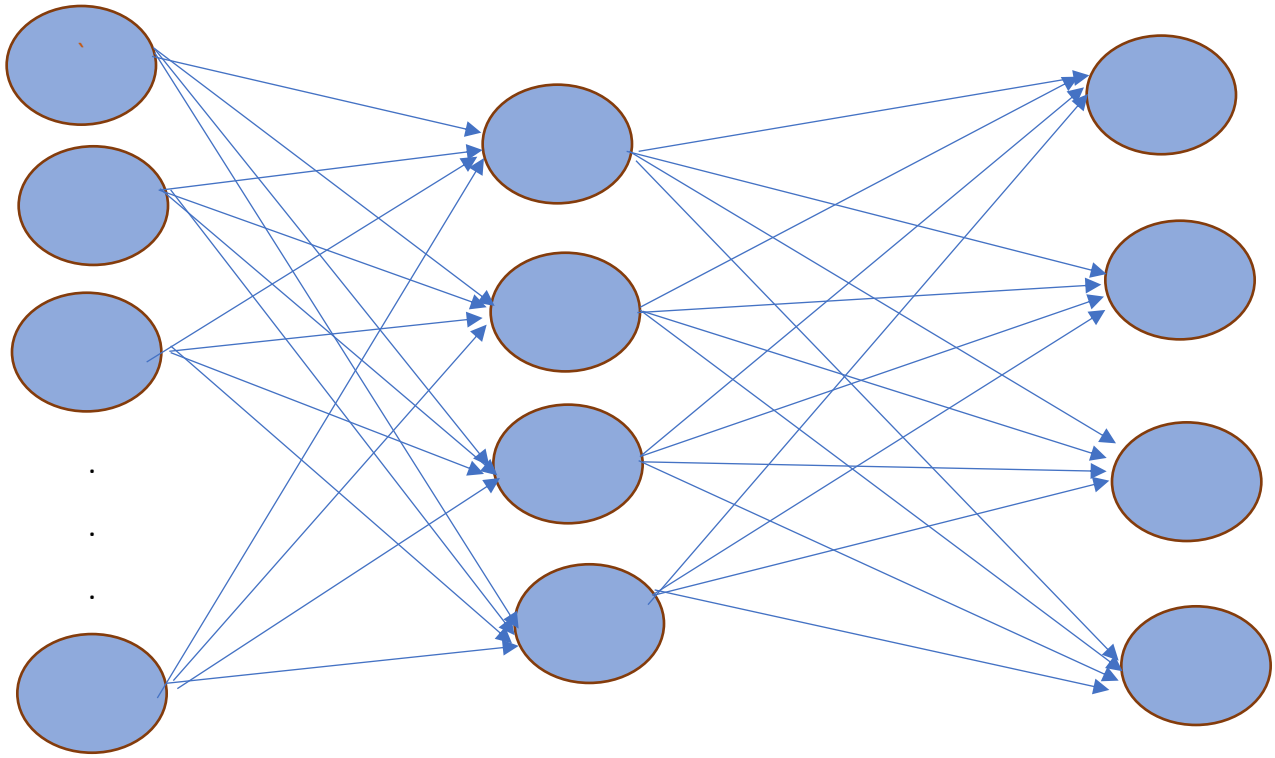
Artificial neural networks are the statistical learning models. They are represented by a system of interconnected 'neurons' which send messages to each other. A neural network has 3 components: input layer, set of one or more hidden layers and output layer.

Neural networks are widely used in computer vision and speech recognition applications.

There are two steps in creating model for neural net.

1. Forward Propagation
   We apply a set of weights in all the layers of neural network and calculate the predicted output
2. Backward Propagation
   We go from output layer to input layer and calculate the margin of error of output and adjust weights accordingly to decrease the error.

Our Implementation



192 nodes 5 hidden nodes 4 output nodes

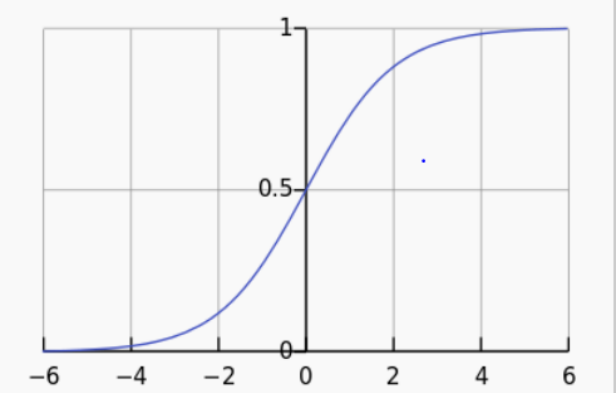
## DESIGN DECISIONS

We used one hidden layer. We tried different nodes in hidden layer to check our performance.

**Activation Function**

We used sigmoid function from input layer to hidden layer and hidden layer to output layer.

$$S(t) = \frac{1}{1 + e^{-t}}.$$



The logistic curve

The value of sigmoid function varies between 0 and 1.
There can be other activation functions like SoftMax, Relu which we can use to check our accuracy.

**Value of Epochs**

We tried different values of epochs from 0 to 1000 to compare our results.

**Learning Rate**

Learning rate is one of the most important parameter for tuning neural networks. It remains constant. We must set learning rate so that it minimizes our loss function. We tried different value of learning rate to compare our performance.

Suppose we have 5 nodes in hidden layer.

**Input Matrix (X):**
We have 192 feature values for each image. So, size of input matrix is 1*192.

**W1:**
Size of W1 should be 192* no of hidden layer nodes i.e. 192*5.

**Z1:**
We do matrix multiplication of X and W1 to obtain Z1.
Therefore, size of Z1 is 1*5(no of hidden layer nodes).

**A1:**
We pass Z1 through an activation function (say sigmoid). The result is stored in a matrix. The size of A1 is same as Z1 i.e. 1*5(no of hidden layer nodes).

**W2:**
Size of W2 should be no of hidden layer nodes* no of output layer nodes.
In our assignment, we have four output nodes for each type of orientation (0,180,90,270). Therefore, size of w2 is 5*4.

**Z2:**
We take a dot product of A1 and W2 to obtain Z2. Therefore, size of Z2 is 1*4.

**Y HAT:**
This is our predicted output. Therefore, size of this matrix is 1*4. We pass Z2 through our activation function to find predicted output.

NEURAL NETWORK ALGORITHM:

**FOR TRAINING**

1. Randomly assign weights to W1 and W2 matrices between -1 and 1.
2. For each training image, do:
   Forward Propagation
   a) Z1 = X.W1 where X is Training image matrix
   b) A1= sigmoid(Z1)
   c) Z2 = A1.W2
   d) YHAT = sigmoid(Z2)

   Backward propagation
   e) Error = actual output – yhat
   f) Derivate_Yhat = derivative of sigmoid function value for YHAT.
   g) Delta = Multiplication of error and Derivative_Yhat
   h) Djdw = dot product of A1 transpose and Delta
   i) Derivate_A1 = derivative of sigmoid function value for A1.
   j) Delta1 = dot product of Delta and Transpose of W2.
   k) Delta 2 = Delta1 * Derivative_A1
   l) Djdw1 = dot product of x and Delta2
   m) Set Learning Parameter = constant value

n) Adjust weights according to djdw and djdw1.
   I. W1 = W1 + Learning Parameter*Djdw1
   II. W2 = W2 + Learning Parameter * Djdw
3. Repeat Step 2 for N times.

This gives us W1 AND W2 matrices after training. We store these in our training file.

**FOR TESTING**

1. For each test image, do :
   a. X = feature values of test image
   b. Z2 = dot product of X and W1
   c. A2 = sigmoid(Z2)
   d. Z3 = dot product of A2 and W2
   e. YHAT = sigmoid(Z3)
   f. We find the maximum value in YHAT and assign label corresponding to the max value

RESULTS

We found some of the correctly and incorrectly classified images.
We observed some pattern in the images that were classified incorrectly.

Some of the correctly classified images are:

Predicted Label:270
Actual Label:270

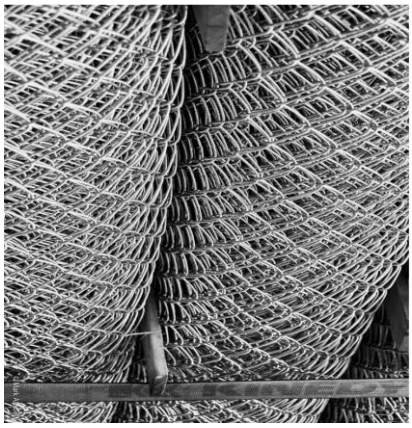

Predicted Label:90
Actual Label: 90

Predicted Label:0
Actual Label:0

Some of the incorrectly classified images are:

Predicted Label: 90 degrees
Actual Label: 270 Degrees

Predicted Label: 0 degrees
Actual Label: 270 degrees



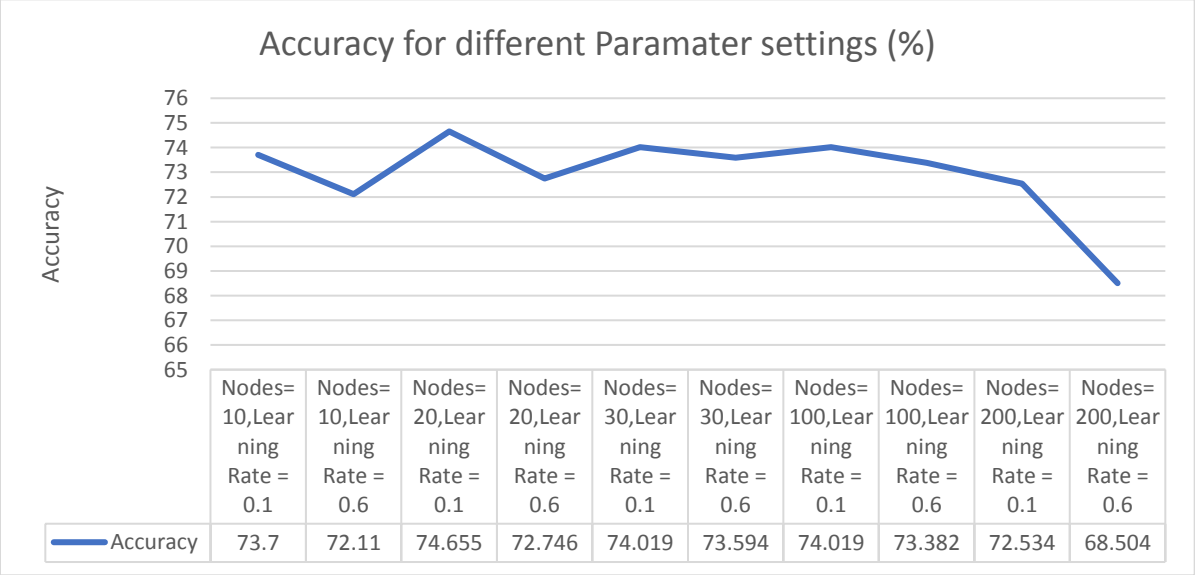
Predicted Label: 270 degrees
Actual Label: 90 degrees

We can observe that symmetrical images which do not have any pattern, or which have same color in whole image does not get classified correctly. This makes sense because we do not have required feature information which distinguishes between different patterns in different parts of image.

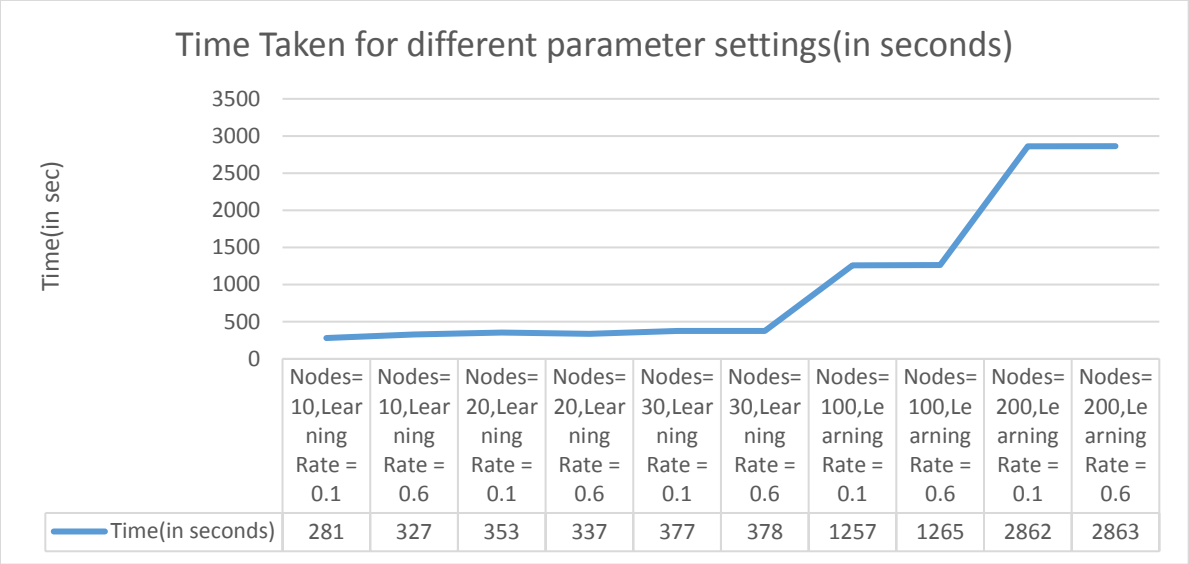
PARAMETER OPTIMIZATION

We tried different parameter setting and recorded the accuracy. Results are displayed in table below:

| Hidden Nodes | Learning rate | Epochs | Accuracy | Time (in seconds) |
|---|---|---|---|---|
| 10 | 0.1 | 100 | 73.7 | 281 |
| | 0.6 | 100 | 72.11 | 327 |
| 20 | 0.1 | 100 | 74.655 | 353 |
| | 0.6 | 100 | 72.746 | 337 |
| 30 | 0.1 | 100 | 74.019 | 377 |
| | 0.6 | 100 | 73.594 | 378 |
| 100 | 0.1 | 100 | 74.019 | 1257 |
| | 0.6 | 100 | 73.382 | 1265 |
| 200 | 0.1 | 100 | 72.534 | 2862 |
| | 0.6 | 100 | 68.504 | 2863 |



W observe that we get different accuracy for different settings for Parameters. It is very important to try different set of parameters to find the best performing parameters. We observe that at 20 nodes and 0.1 learning rate with epochs =100, gives us best performance (74.66%).




| | Nodes=10,Learning Rate = 0.1 | Nodes=10,Learning Rate = 0.6 | Nodes=20,Learning Rate = 0.1 | Nodes=20,Learning Rate = 0.6 | Nodes=30,Learning Rate = 0.1 | Nodes=30,Learning Rate = 0.6 | Nodes=100,Learning Rate = 0.1 | Nodes=100,Learning Rate = 0.6 | Nodes=200,Learning Rate = 0.1 | Nodes=200,Learning Rate = 0.6 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time(in seconds) | 281 | 327 | 353 | 337 | 377 | 378 | 1257 | 1265 | 2862 | 2863 |

Running time increases as we run for more number of nodes. It also increases if we run for different values of epochs.

We would recommend using 20 hidden nodes, 1 layer,0.1 learning rate and Number of epochs=100 for an accuracy of 74-76%. This parameter setting gives the best performance in reasonable amount of time.

**BEST ALGORITHM**

We have set our best algorithm as neural nets with one hidden layer,0.1 learning rate and 25 hidden nodes.

**PERFORMANCE COMPARISON FOR ALL ALGORITHMS**

We observe the accuracy as below when the algorithms are run for the 1/4$^{th}$ of the given training set(approx. 7500 images).

| ALGORITHM | ACCURACY | RUNTIME |
|---|---|---|
| **KNN** | 69.56% | 3m 52sec |
| **Adaboost** | 65.95% | 52.59sec |
| **NNET** | 72.32% | 1m 29sec |
| **Best** | 72.32% | 1m 27sec |

We observe the accuracy as below when the algorithms are run for the entire training set.

| ALGORITHM | ACCURACY | RUNTIME |
|---|---|---|
| **KNN** | 71.26% | 15m54.054s |
| **Adaboost** | 69.03% | 4m11.255s |
| **NNET** | 74.34% | 5m37.158s |
| **Best** | 74.44% | 5m36.430s |

We see a drop in performance when the partial training set is used. This because, in partial training set there are less training examples for the algorithm to learn. However, we can see a faster computation time due to less number of training examples.

Hence we can conclude that with more training data the accuracy of the machine learning algorithm increases.