Group – divrajen, garra, haputti (Divya Rajendran, Goutham Arra, Harika Putti)

KNN - (k-Nearest Neighbours)

In this problem of image orientation classification, we have used kNN algorithm which checks k-nearest neighbours of the image we want to test to obtain the image orientation of that test image. Given a set of training vectors, KNN algorithm identifies the k nearest neighbours of a vector for which we need to find the orientation.

The model for kNN is the entire training data and so we need to be careful with the consistency of the training data set. KNN makes predictions based on the training data, in our problem k nearest neighbours of a test image are checked and the match with the average of the orientation of its k-nearest neighbours or majority votes is assigned as the orientation of the image. Here in our problem, we have selected Euclidean distance as the distance measures to arrive at the k nearest neighbours.

We tried to train small sets of training data at once and we could find the program ran in considerably lesser time and with an acceptable accuracy. However, when we trained the entire data set, we got a greater accuracy even though it took some extra minutes to calculate the orientation of the image.

We have observed that with big k values or the nearest neighbours to check, the prediction is slower and is computationally expensive. It is a very simple algorithm to use and obtain results and the accuracy is high.

Adaboost:

Adaboost, is a boosting algorithm which uses several weak classifiers to create a stronger classifier linearly. First a model is created from the training data set, a second model is created to rectify any mistakes the first model has made, and the models are added accordingly until all the training data set is predicted appropriately. Every vector in our given training data is weighted.

A weak classifier is modelled first on the training data set using the weighted values, and each decision stump makes an output value between minus and plus 1. The initial weights were set to 1/n where n is the number of training data vectors we have. The error rates are calculated as (number of correct predictions – N)/N, N is the number of vectors in our training data set. We have modified it to contain the weighted sum as (weight of a vector * the predicted error)/ (sum of weights). Here the predicted error takes a value of 1 if wrongly classified and 0 if its correct.

Algorithm:

(1) **function** ADABOOST(examples,L,K) **returns** a weighted-majority hypothesis

**inputs**: examples, set of N labeled examples (x1, y1), . . . , (xN, yN)

L, a learning algorithm

K, the number of hypotheses in the ensemble

**local variables**: **w**, a vector of N example weights, initially 1/N

**h**, a vector of K hypotheses

**z**, a vector of K hypothesis weights

**for** k = 1 **to** K **do**

**h**[k]←L(examples,**w**)

error ←0

**for** j = 1 **to** N **do**

**if h**[k](xj ) _= yj **then** error ←error + **w**[j]

**for** j = 1 **to** N **do**

**if h**[k](xj) = yj **then w**[j]←**w**[j] ・ error/(1 − error )

**w**←NORMALIZE(**w**)

**z**[k]←log (1 − error )/error

**return** WEIGHTED-MAJORITY(**h**, **z**)

(1) Artificial Intelligence A Modern Approach by Russel and Norwig, page 751, figure 18.34

Neural nets:

Neural net is an algorithm which is based on a connection of randomized nodes which can transmit information from one to another. These nodes are called neurons, the neurons are arranged in a predefined matrix format and output of each neuron is calculated using a non-linear function. Each connection between the neurons are called synapses and have a weight associated with it. This weight determines the accuracy of our predictions.

Every set of neurons are placed in layers, the first layer being the input vector. In case of our image orientation classification problem, we have the vector of 192 units in the first layer. The second layer consists of forty neurons with different transformations and weights attached to them, which will transform the input data through a dot product between them and the input layer neurons to give an output to a layer of twenty neurons which further transform the data based on their weights into four neurons. The maximum of the four neurons value is attributed as the correct image orientation for our problem.

Algorithm:

(2)**function** BACK-PROP-LEARNING(examples, network ) **returns** a neural network

**inputs**: examples, a set of examples, each with input vector **x** and output vector **y**

network , a multilayer network with L layers, weights wi,j , activation function g

**local variables**: Δ , a vector of errors, indexed by network node

**repeat**

**for each** weight wi,j in network **do**

wi,j ←a small random number

**for each** example (**x**, **y**) **in** examples **do**

/* *Propagate the inputs forward to compute the outputs* */

**for each** node i in the input layer **do**

ai←xi

**for** _ = 2 **to** L **do**

**for each** node j in layer _ **do**

inj ←_i wi,j ai

aj ←g(inj)

*/* Propagate deltas backward from output layer to input layer */*

**for each** node j in the output layer **do**

Δ [j]←g_(inj) × (yj − aj)

**for** _ = L − 1 **to** 1 **do**

**for each** node i in layer _ **do**

Δ [i]←g_(ini)_j wi,j Δ [j]

*/* Update every weight in network using deltas */*

**for each** weight wi,j in network **do**

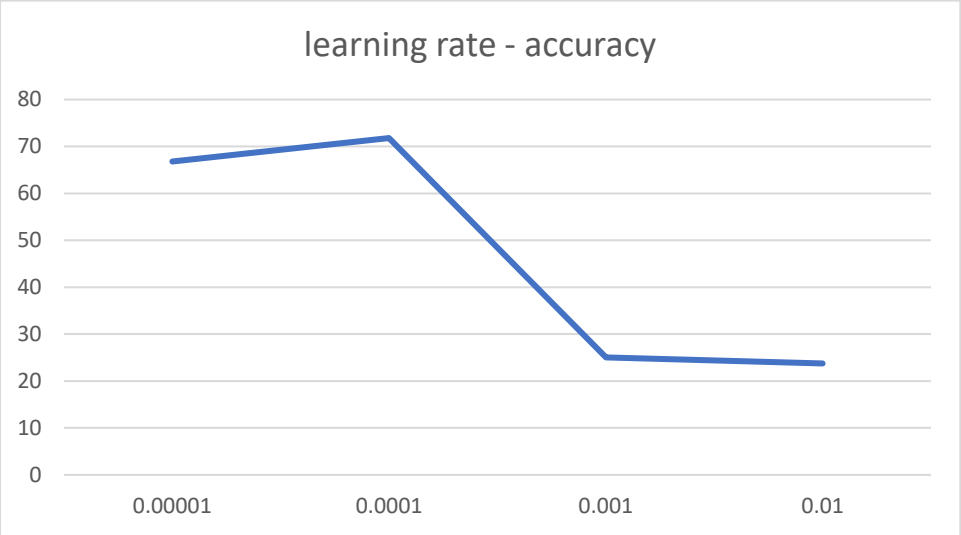
wi,j ←wi,j + $\alpha$ × ai × Δ [j]

**until** some stopping criterion is satisfied

**return** network

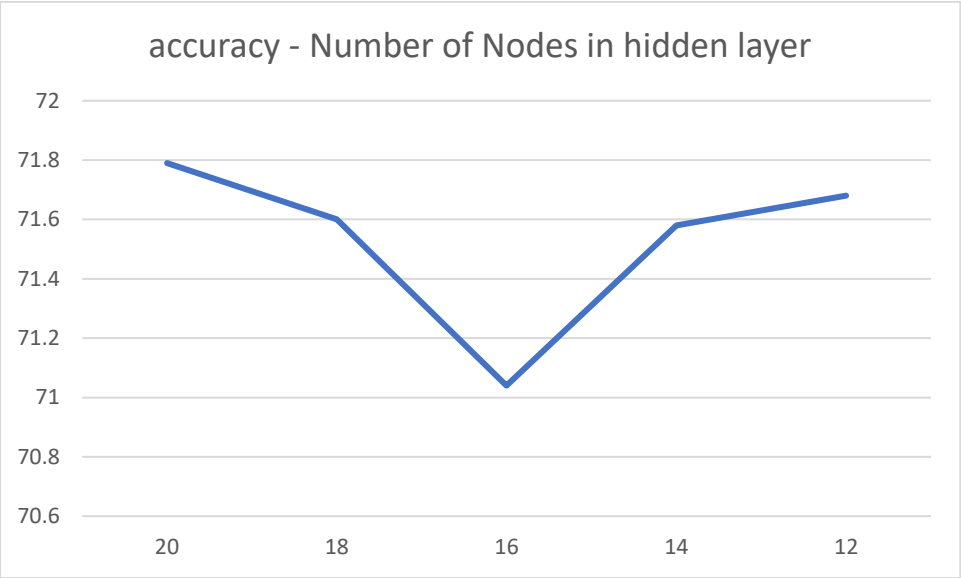
(2) **Artificial Intelligence A Modern Approach by Russel and Norwig, Page 734 Fig18.24**

We have noted the learning rates and accuracies given the number of iterations of the data and the weights we gave, our analysis of the data for the algorithm is as below.



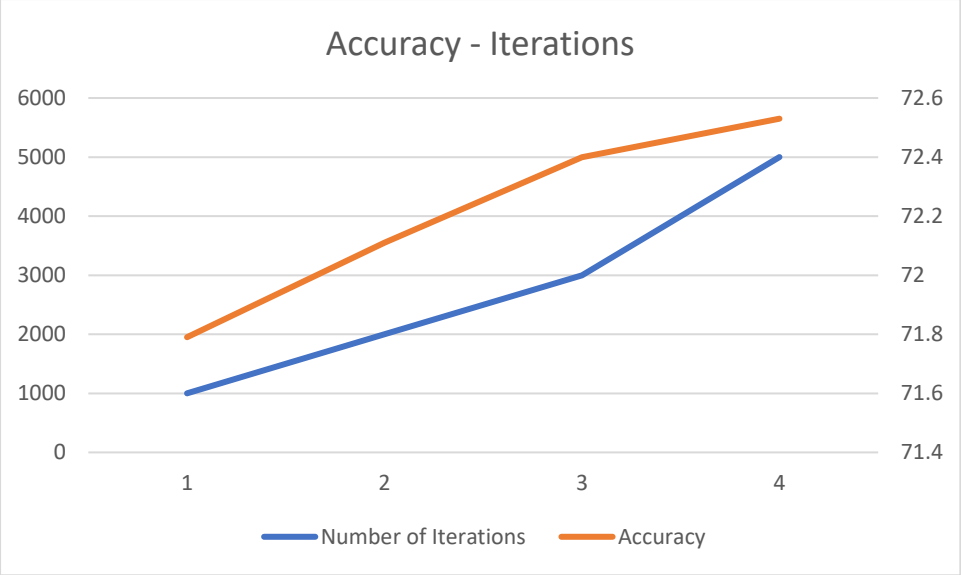
| learning rate | accuracy | iterations | range of initial weights |
|---|---|---|---|
| 0.00001 | 66.8 | 1000 | -1 to 1 |
| 0.0001 | 71.79 | 1000 | -1 to 1 |
| 0.001 | 25.02 | 1000 | -1 to 1 |
| 0.01 | 23.75 | 1000 | -1 to 1 |

We see that the accuracy of the predictions increases as the learning rate is reduced in the above graph




| number of nodes in hidden layer | accuracy | iterations | learning rate | range of initial weights |
|---|---|---|---|---|
| 20 | 71.79 | 1000 | 0.0001 | -1 to 1 |
| 18 | 71.6 | 1000 | 0.0001 | -1 to 1 |
| 16 | 71.04 | 1000 | 0.0001 | -1 to 1 |
| 14 | 71.58 | 1000 | 0.0001 | -1 to 1 |
| 12 | 71.68 | 1000 | 0.0001 | -1 to 1 |

We can see from the above graph that the accuracy reduces as we reduce the number of nodes and increases once we reduce it further to 12 and below.




| iterations | accuracy | learning rate |
|---|---|---|
| 1000 | 71.79 | 0.0001 |
| 2000 | 72.11 | 0.0001 |
| 3000 | 72.4 | 0.0001 |
| 5000 | 72.53 | 0.0001 |

Here the accuracy increases as the number of iterations increase and it has a small depression at 3000 iterations and rises back once more.

Finally, we choose the below configuration for our nueral nets model.

| iterations | accuracy | learning rate | range of initial weights | number of hidden layers |
|---|---|---|---|---|
| 5000 | 71.79 | 0.0001 | -1 to 1 | 20 |

When we check the images below, we had some hard time in getting the classification proper as the red colour dominates most of the picture and similarly the blues dominate the upper most part of the pictures making it harder for the distance functions to calculate proper values to get the orientation right.

Similarly the picture below has the same issue occurred with the immense concentration of rgb colors at same locations.