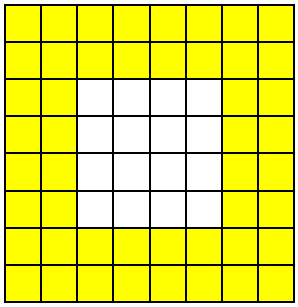
K Nearest Neighbor Algorithm:

The K Nearest Neighbors algorithm was implemented to predict the correct orientation of images. To arrive at the best configuration for KNN that works best for this data, many experiments were conducted. All these experiments are detailed in the upcoming sections.

EXPERIMENTS:

The algorithm was trained and tested using two different methods:

1. The data was trained using the train data provided which contained 192 pixels as the features.
2. The data was trained and tested by considering only the 2 pixels in the border. If a grid represents the image where each box contains the RGB values of that pixel, only the pixels belonging to the shaded portion were taken as features. The intuition behind this idea is that if the orientation of an image is changed it is highly likely that the position of pixels in the border change. The pixel positions in the center may or may not change. This method seemed to give a higher accuracy when experimented with Euclidean and Manhattan distance. However, this is not presented in the report as this method may lead to overfitting and was not used in the Final model.



In addition to using two different datasets, different distances used for finding the nearest neighbors. Also the value of 'k' was varied from 10 to 100 in order to pick the best value of K. The results of various experiments are shown below.

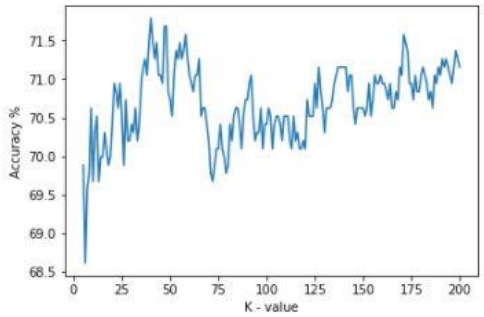
All the experiments were performed using all the 192 pixels as features

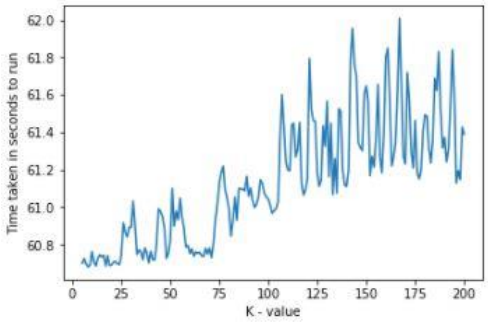
VARIAITON OF ACCURACY AND TIME WITH K

The variation of Accuracy and time is studied for each of the distances.

Euclidean Distance

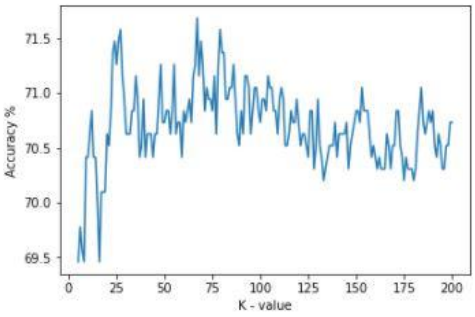
Euclidean distance is one of the most common distance metrics used. The formula for Euclidean distance is $\sum_{i=1}^{192}\sqrt{\left(x_i^2 - y_i^2\right)}$. The highest accuracy of 71.79% was observed for k=40. However, using an even number may result in ties hence the next best value k=47 was taken as the best value of K. The accuracy for k= 47 was 71.68%. The time taken to run the algorithm increases as the value of K also increases.

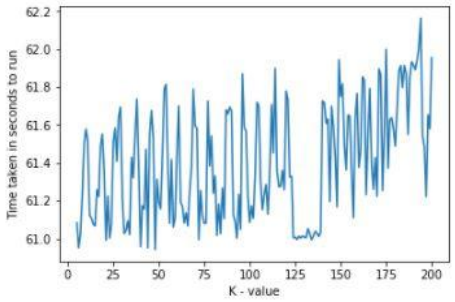




Manhattan Distance

Manhattan distance is another common distance metric. The formula for Manhattan distance is $\sum_{i=1}^{192}\sqrt{(x_i^2 - y_i^2)}$. The plot depicting the variation of Accuracy with the value of K is shown below.



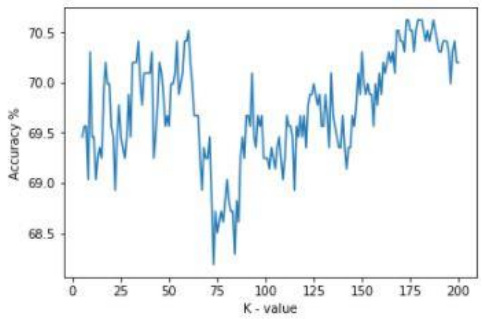


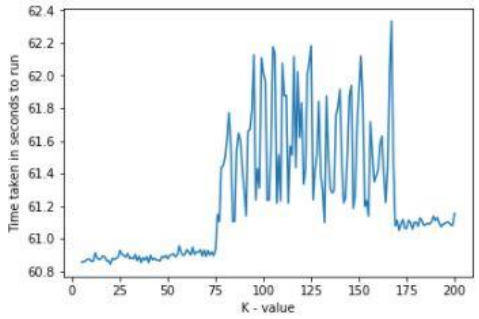
K= 67 gave the best accuracy of 71.68 % for Manhattan distance. This is about the same as Euclidean distance. Overall, the time seems to increase as the value of K increases. However, there are visible peaks that can be observed.

Mahalanobis Distance:

Mahalanobis distance is a modification of Euclidean distance where the inverse of variance is used to weight the features. The formula is $\sum_{i=1}^{192}\sqrt{\frac{(x_i^2 - y_i^2)}{\sigma_i^2}}$. Where σ_i^2 is the variance of the ith feature. The plot shows the variation of Accuracy and Time with K value for this distance.

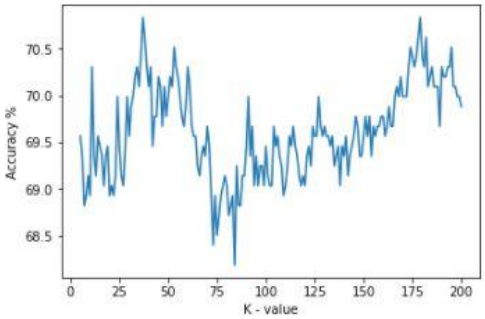
The best accuracy is obtained for k= 179 which was 70.62% . Interestingly , this seems to work well for large values of K. The time taken to run almost remains same from 5 to 75. When K increases beyond 75, the time to run also increases drastically and the run time is on the higher side till k=175.

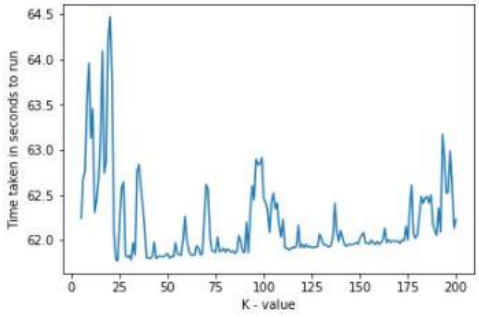




Distance weighted KNN

Instead of giving equal weights to all the votes of the neighbors, we tried to weight the votes using the inverse of the distance. This way points that are closest to the test data point will be given more importance than the ones that are far away. However, from the plot below it can be observed that it did not result in a significant improvement in the result and the best accuracy was around 70.84% for k= 37.





EFFECT OF TRAIN DATA SIZE ON RUN TIME AND ACCURACY

The number of data points used for training (distance computation in the vector space) was varied from 5000 to 36976 where k= 47 and the distance metric as Euclidean. For smaller number of data points it was observed that the run time is a evidently low. As the number of train data points increases the run time also increases. One interesting observation is that, even when $1/6^{th}$ of the data was used as train, there was not a considerable drop in the accuracy. This indicates that a well randomized sample of the original data is taken, KNN would work reasonably well.

	Number of records used to Train						
	5000	**10000**	**15000**	**20000**	**25000**	**30000**	**36976**
Accuracy	68.611	68.823	70.201	69.989	68.929	69.671	71.680
Time	6.859	13.122	19.760	25.414	32.359	45.436	60.201

SUGGESTED CONFIGURATION OF k NEAREST NEIGHBOR ALGORITHM:

We observed that Maximum accuracy is observed for Euclidean distance when k=47 is used. A similar accuracy is observed when Manhattan distance is used for k=67. Since both the distance metrics gave similar results, Euclidean distance was finally selected as it had slightly lesser runtime than when Manhattan distance was used at K=67 (We did observe the runtime increases as k is increased).

Also training on all data points seems to work the best for KNN though not a great decrease in the accuracy is seen if lesser number of data points is used.

The suggested configuration for KNN

K: 47

Distance Metric: Euclidean distance

Train Data Size: The entire data

Images that were correctly classified by KNN

Images that were incorrectly classified by KNN

Confusion matrix

Actual \Predicted	0	90	180	270
0	71%	7%	14%	9%
90	8%	75%	5%	13%
180	15%	9%	70%	6%
270	8%	16%	5%	71%

From the confusion matrix it can be observed that the classifier gets 16% of the pictures with orientation 270 as 90 which is the highest misclassification %. It also tends to predict 0 as 180 and 180 as 0. The % of such classifications is 14% and 15 % respectively.

Adaboost

ALGORITHM

The algorithm given in Freund & Schapire was used for implementing Adaboost.

Step 1: 6 Classifiers were built for identifying: Orientation 0 vs Orientation 90, Orientation 0 vs Orientation 180, Orientation 0 vs Orientation 270, Orientation 90 vs Orientation 180, Orientation 90 vs Orientation 270, Orientation 180 vs Orientation 270
Step 2: For each classifier, 20 stumps were built. Each stump compares the value of two pixels. 192 x 191 stumps were created to compare if pixel_i ≥ pixel_j
Step 3: To choose the stumps, the weighted error ($\varepsilon = \sum w(h(x) - y)$) was calculated for each of the stumps and the stump with least weighted error was chosen. The weights of the images are initialized based on positive and negative sample distribution
Step 4: After choosing the first stump, prediction is done on the images and correctly classified images are down-weighted. The weights are then normalized, and the subsequent stumps are created in a similar manner
Step 5: The pixels compared by the stumps and the alpha value corresponding to the stumps are stored
Step 6: Other 5 classifiers are built in a similar manner, taking the same number of images as input and the same number of stumps
Step 7: The data label is predicted by each stump of the classifier. This predicted vector is multiplied by the alpha value of the corresponding stump. The vectors produced by the 20 stumps are added to generate a final vector for the classifier
Step 8: Each of the 6 classifiers predictions are swapped to generate predictions for the opposite classes
Step 9: The final prediction is done by comparing the 12 vectors produced in Step 7. The class corresponding to maximum value is chosen as the predicted class

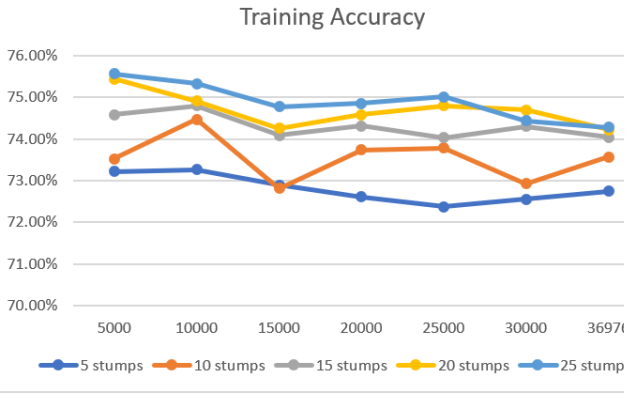
Failed Methods

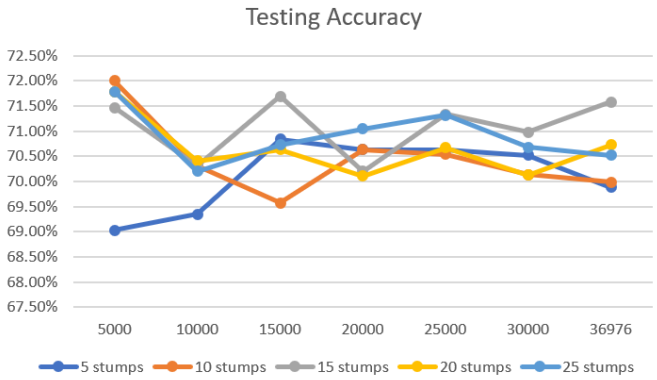
Three other algorithms were implemented in an attempt to better predict the orientations.

- 4 One vs All classifiers were built following the same algorithm mentioned above. But the accuracies were slightly lower than that produced by the 6-classifier case. This might be because of the class imbalance between positive and negative samples in each of the classifiers. This method had an accuracy of 7259% on training data and 70.13% on the test data. However, its performance was slightly less than that of the chosen model
- RGB classifier which compares pixel values within the respective planes did not produce good results. This is because, the original classifier produced stumps that are a superset of those considered in this classifier. So, the original classifier gave better results as compared to this. This method had an accuracy of 68.3% on the training data and 66.31% on the test data
- Stumps were built by picking a value between 0 – 255 that gave minimum error. Instead of comparing pixels, thresholds were set for each pixel. This method also did not produce the best results. This method gives an accuracy of 73.33% on the training data and 69.67% even when stumps were created for each of the features (i.e. 192 stumps per classifier)

Accuracies & Run-times:

The best model chosen for prediction uses 15 decision stumps constructed on the entire training dataset containing 36969 images. This model has an accuracy of 74.04% on the training data 71.58% on the test data. From the plot of accuracies for varied number of stumps in train and test data, it was observed that the models with 15 stumps performed best in terms of accuracy and not overfitting. For stumps more than 20, the model was overfitting the data and performing poorly on the test dataset. For models with stumps less than 10% the accuracy itself was relatively low.





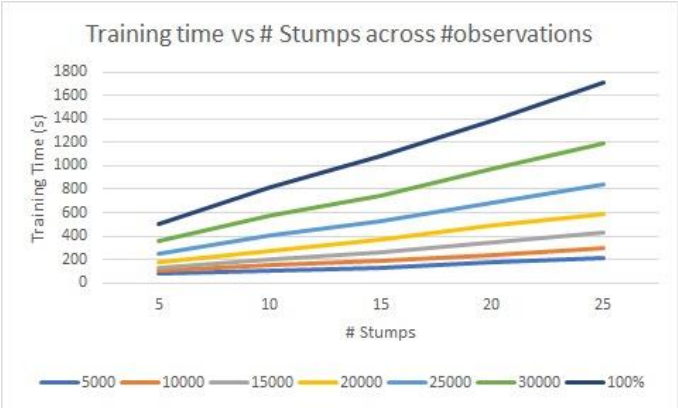
The table given below gives the confusion matrix for the predictions made by the adaboost model. It can be observed that most images have been wrongly predicted to be belonging to class with orientation 0.

Actual Orientation	Predicted Orientation				
	0	90	180	270	Total
0	170	17	32	20	239
90	18	157	22	27	224
180	30	18	165	23	236
270	26	21	14	183	244
Total	244	213	233	253	943

Given below are a sub sample of some of the images that have been correctly predicted by the model:

Images given below are the ones that were not correctly classified by the model. From these images it can be seen that some of them are symmetric in nature and even humans find it confusing to determine their orientation. Some other images like the first and last one are relatively simple but have been missed by the model.

The time taken to implement the adaboost algorithm for varying stump size and sample size. The implementation took a long time because 192x191 comparisons and the corresponding error had to be computed and the one with the lowest error was chosen as the stump. This had to be done everytime a stump was created.



Neural Network

A fully connected Feed forward neural network with hidden layers varying from 1 to 5 was implemented to predict the correct orientation of images. A momemtum update was also performed while updating the weights and biases after the gradient calculation. Weights of the network are randomly initialized between -1 to 1. Any other initialization resulted in slower convergence. Also , the nn was trained for 25 epochs and in each epoch the training dataset is shuffled before use.

DATA PROCESSING

Input data was normalized to have zero mean and one standard deviation for each of the 192 features. This ensured proper convergence of weights and biases of the neural network.

PARAMETERS CONSIDERED

Hidden layers: 1, 2….5

Learning Rate : 0.01, 0.05, 0.1

Activation Function: sigmoid

Neurons in the hidden layer: 256, 192, 128, 64, 32

256 neurons were used if the neural network consisted of one hidden layer. In case of a netwoek with 2 hidden layers 256 and 192 neurons were used In the first and second hidden layer respectively. Each of this configurations were run for a maximum of 25 epochs.

RESULTS

Figure 1 shows the train and test data accuracy for each incremental number of hidden layers. Best Accuracy of 74.76% was achieved on the test dataset when a neural network with 5 hidden layers was used to predict the correct orientation. The number of neurons in each of the 5 hidden layers are 256, 192, 128, 64, 32 respectively. The best performance was achieved when a learning rate of 0.1 was used.

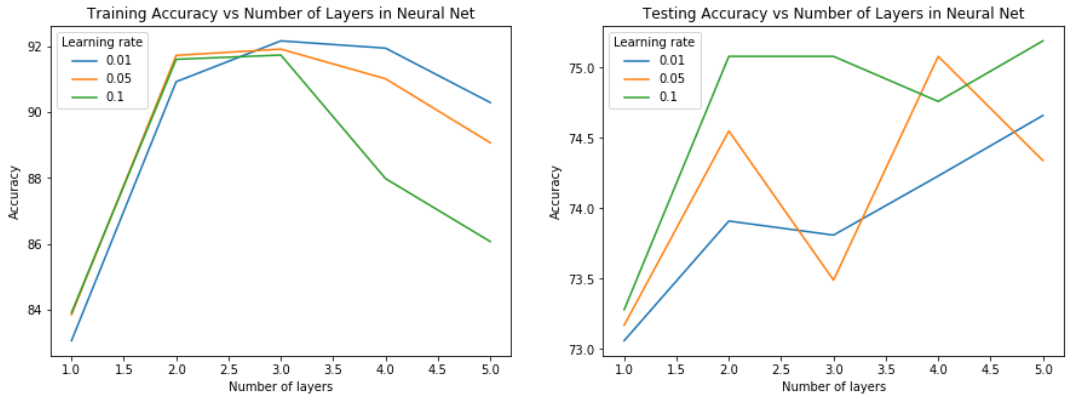


Figure 1: Training & Test data accuracy for hidden layers from 1 to 5

For the best model mentioned above, Figure 2 shows the variation of accuracy with the number of epochs. It can be noted that there was only a marginal difference in the best testing accuracy (75%) compared to the first epoch's test accuracy (72.5%).

Figure 2: Training & Test data accuracy for 5 hidden layers NN across 25 epochs

Figure 3 shows the variation of average time taken (in seconds) for one epoch with the number of hidden layers. A single hidden layer with 256 neurons took approximately 8 seconds to run compared to the 5 hidden layer network which took about 25 seconds respectively.

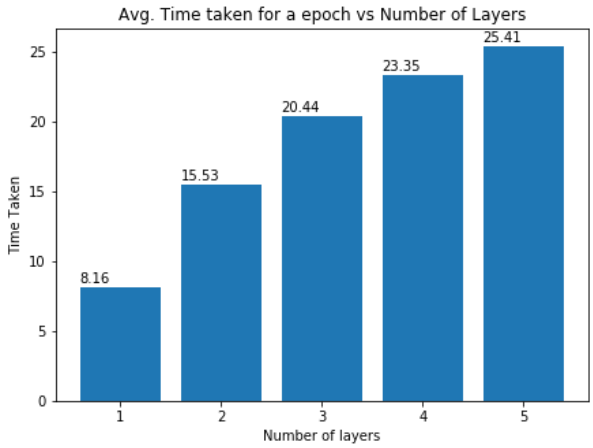


Figure 3: Number of hidden layers vs Avg. Time taken for an epoch

Figure 4 explains what happens to the train and test accuracy for every 5% increase in the size of train data. It is evident to note that for just 5% of the training dataset the neural network achieves a 66% and 64% accuracy on the train and test respectively. The remaining 95% of the training dataset accounts only for ~11% improvement in the testing accuracy.

Figure 4: Training and Testing accuracy vs 5% incremental addition of training dataset

Below is a confusion matrix which indicates the performance of model's predictions. It can be observed from the Table 1 that the model predicts the "270" orientation the most (84.43 %) and "180" the least (65.68 %).

Actual /Predicted	0	90	180	270
0	74.90	4.91	8.47	11.89
90	3.77	75.00	3.39	15.98
180	16.32	4.02	65.68	13.52
270	5.02	9.38	2.12	84.43

Table 1: Confusion matrix of the best model with 5 hidden layers

Best Algorithm

The best algorithm for predicting the correct orientation of the image is Neural Network with 5 hidden layers with 256, 192, 128, 64, 32 neurons in each layer and with learning rate =0.1 and momentum rate=0.05. The model file for this algorithm is nnet_model_file.npy

Correctly Classified Images


18079654


4994447095


5831306106

9297964944


14330314647

Incorrectly Classified Images


9476970


167350424


2689816071


2861139985


4132279737