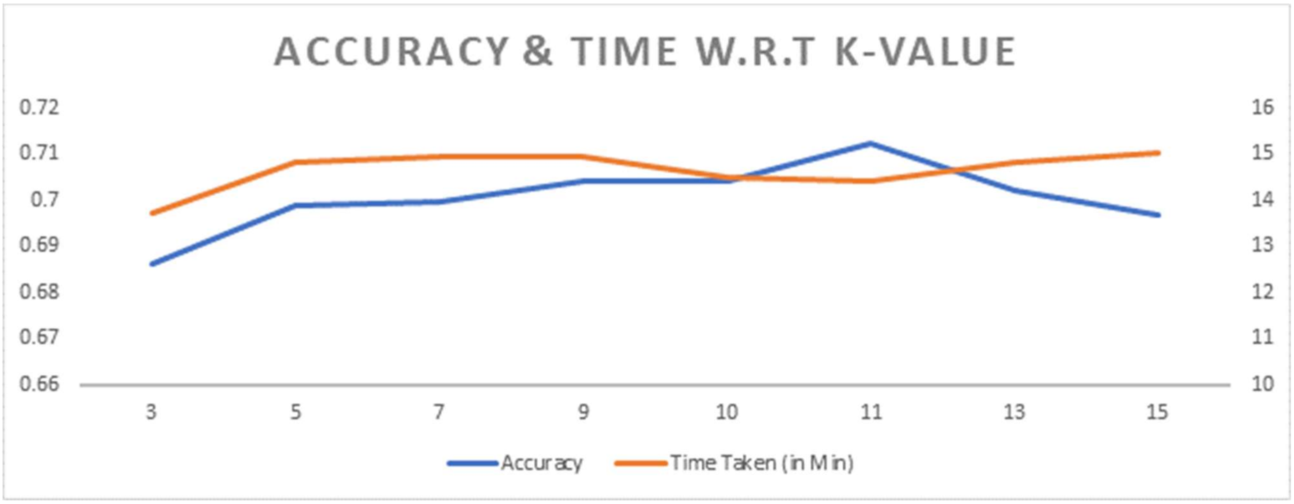
Assignment 4: Machine learning

Part 1: Nearest:

Steps followed in this algorithm is as follows:

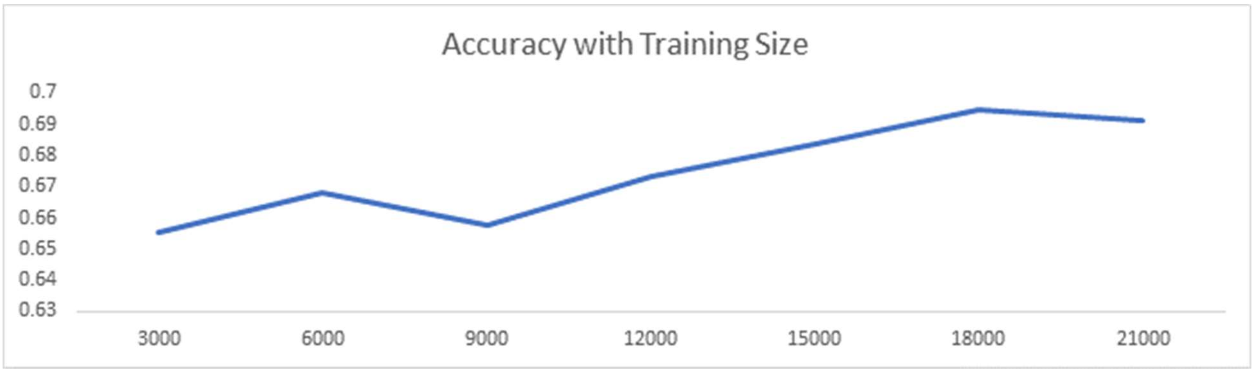
1) Calculating the Euclidean distance from a given test data to all the test data(~36K).
2) find the K nearest neighbors and then based on the max. votes of these K neighbors to predict the orientation of given picture. In our case K=11 yields maximum accuracy (~72%).

Accuracy & Time W.R.T K-value:



From the above graph, time taken to run for different K value is almost equal and close to ~12-13mins. But the Accuracy is maximum at K=11.

Performance with respective to the training size:



We can observe that with the increase in the training size, the accuracy improves, so as the training data is about 36k, so at the full training data the accuracy is about ~71%.

When the training size is small, the following table shows the images of correctly predicted orientation and incorrectly predicted orientation.

Correctly predicted	Incorrectly predicted
test/11610344784.jpg	test/10196604813.jpg
test/11673951276.jpg	test/10351347465.jpg
test/11722525924.jpg	test/10352491496.jpg
test/11897908725.jpg	test/10484444553.jpg
test/1194122691.jpg	test/10577249185.jpg
test/12063320613.jpg	test/108172840.jpg
test/12178947064.jpg	test/112713406.jpg
test/12449930553.jpg	test/11418669873.jpg

Based on the above classification, some of the data points are correctly predicted because in smaller training set, we have found the reliable neighbors where as in incorrectly predicted points we couldn't find the reliable neighbors.

Part 2: Ada Boost:

Steps followed for Ada Boost is as follows:

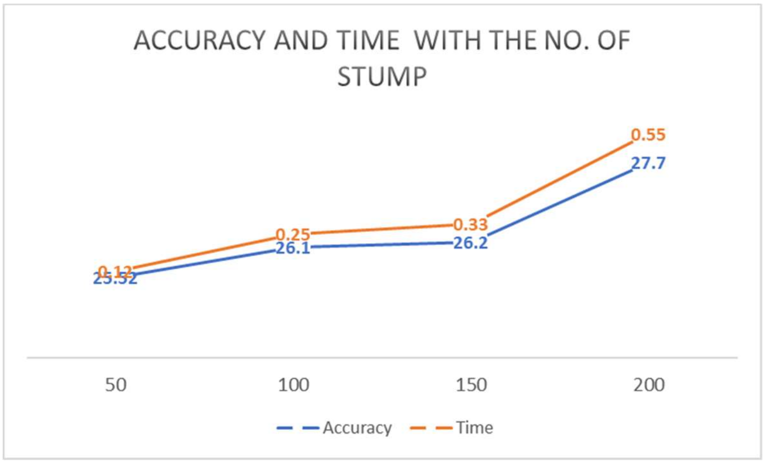
Step1: Equal weights are assigned to each data point in the training set.

Step2: A root node is created by picking up a random pixel and pixel which is four units away from it and partition value is such that difference between two pixels is 40. At this point error and hypothesis weights (alpha) is calculated. weight for this hypothesis is calculated by log(1-error/error)

Step3: And the weights of each point are recalculated by decreasing the weights of corrected predicted data points with error/(1-error). And then all the points are normalized.

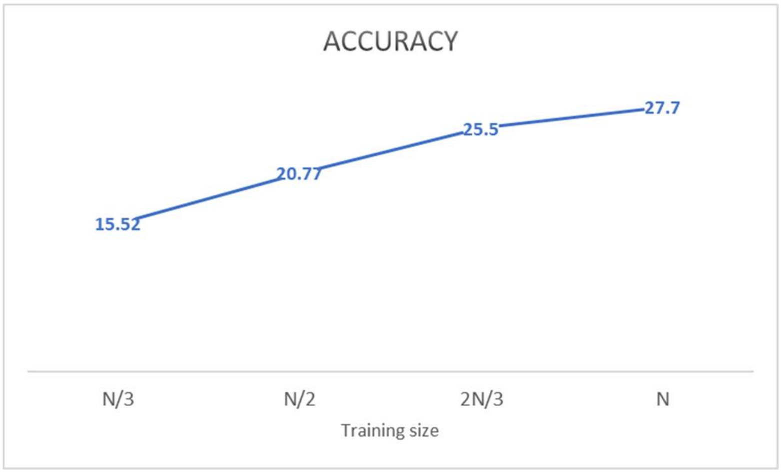
Step4: Step 2 & Step3 are carried out for creating multiple stumps.

Performance in terms of accuracy and time with respective to number of stumps.



Better accuracy is at the stump value 200

Performance in terms of accuracy with varying training size:



We can observe that, that accuracy is increasing with the training size.

Part 3: Random Forest:

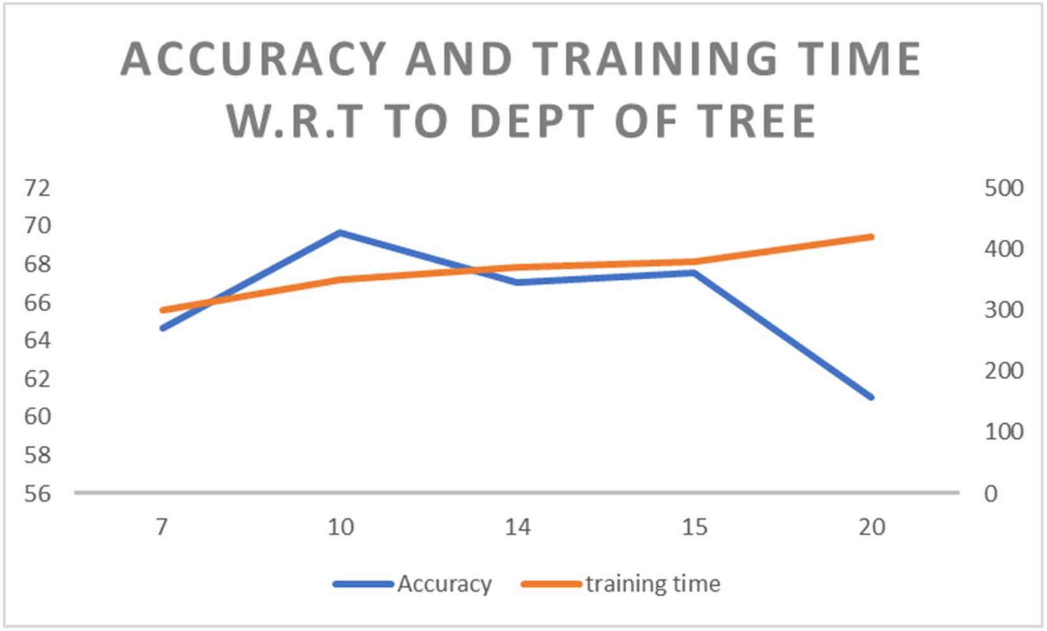
Steps followed for random forest is as follows:

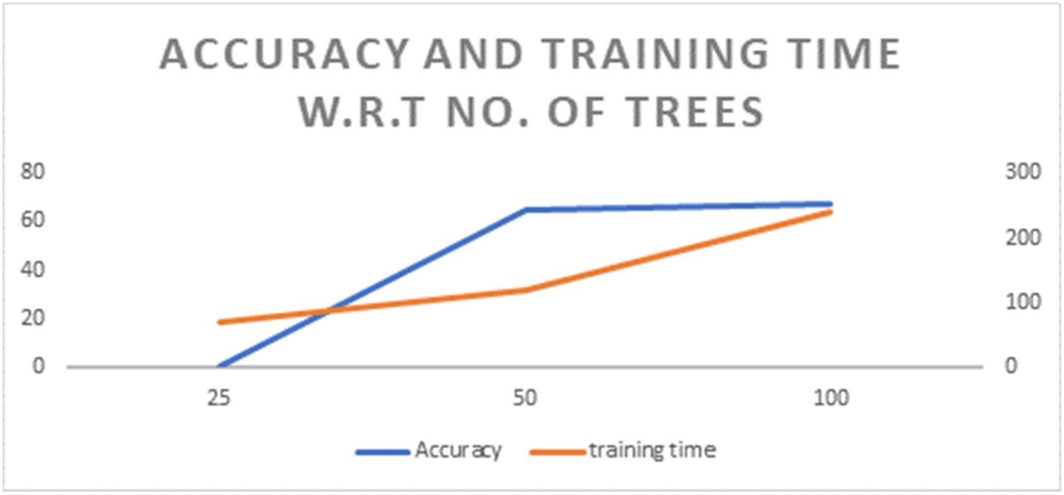
1. for each tree
   On 2/3rd of training data selected randomly
   {a root node and partition value are found by calculating the entropy at each partition value and each 192 pixels.}
   Partition threshold considered are:
   [10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250]
   Entropy is calculating by using the below formula:

$$\underset{\text{All the partitions}}{\sum} (\text{fraction of sample in branch I })\underset{\text{Sum of class}}{\sum} \text{pi}*(\log \text{pi})$$

   So, the pixel with minimum entropy is selected as the root node, then after the split each branch is further split using the above method. And this process is continued until certain depth, upon trails, it is found depth of 15 has yielded better accuracy results.
2. And the above process is carried to generate number of trees.
3. And while testing, for each testing point, the point traverse through all these trees. And then the value of orientation is calculated based on maximum number of votes.

Parameter in random forest are the depth of tree and the number of trees:

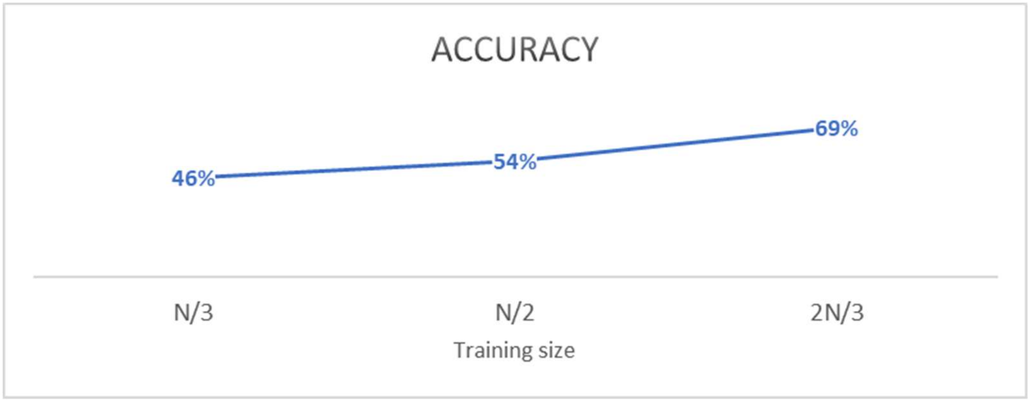




Observations from the above graph:

1. **Parameter dept of tree (No. trees fixed at 100)**: we can see that accuracy increase till the point dept=10 and then decreases. Whereas training time increases with increase in the depth of the tree.
2. **Parameter number of trees**: With increase in number of trees both the accuracy and training time increases. As there is trade of between computational time and accuracy, we can stick with numbers of trees to be 100.
   Note that testing time is nearly same. So we can go for the parameters number of trees is 100 and depth is 10.

Performance with respective to the training size:



We can observe that, the accuracy increases with increase in the training size.

Correct: -	Incorrect
'test/10008707066.jpg',	test/10161556064.jpg',
'test/10099910984.jpg',	'test/10196604813.jpg',
'test/10107730656.jpg',	'test/10351347465.jpg',
'test/10164298814.jpg',	'test/10352491496.jpg',
'test/102461489.jpg',	'test/10484444553.jpg',
'test/10304005245.jpg',	'test/10577249185.jpg',
'test/10313218445.jpg',	'test/10684428096.jpg',
'test/10353444674.jpg',	'test/112713406.jpg',
'test/1074374765.jpg',	'test/11418669873.jpg',
'test/10795283303.jpg'	'test/1160014608.jpg'

Some pictures are incorrectly labelled because the actual orientation is incorrect. Out of which some of them are pictures, where a person would also find difficulty in classifying the orientation.

By decreasing the training size, we are missing out certain things which is resulting in the misclassification.

Part 4: Best Model:

Among the above model, looking through both the accuracy and computational time. KNN performs best among the 3.