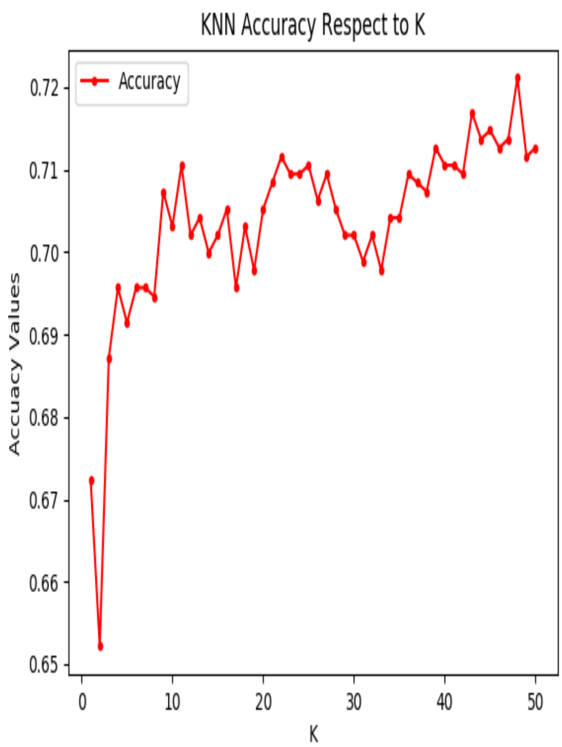
K nearest Neighbor Report:

(1) a brief description of how you formulated the problem

As how KNN algorithm goes, I took every test examples and calculated the distance with all training examples. Then, the distance was ordered in least to ascending order. I used this ordered list and extracted K closest training points with the test set. These training points will have their votes for classifying the test class label, which is their own class label. We estimate the test data's class as the class with the most votes. Let's say if K=5, then we get 5 closest points from the training. 5 training points will have their class and these will be votes for the test data. In ['90','90','180', '270', '0'], it will estimate the test example's class as '90'.

Since K is the user-defined parameter and ultimate K which works for every data doesn't exist, I tried multiple Ks for the best accuracy. Testing from K=1 to 10, I saw an increase of the accuracy, so I tried increasing K. Till K=30, I got the highest accuracy at K=22 but wasn't sure if it was the local maximum or global maximum, so I tried till 50. Below is the plot I earned from K=1 to 50 and output of the accuracy of the results. At K=48, I got about 0.7211, which was the highest among the 3 algorithms we implemented. If I had more time, I hoped to run till K=100 to check how the accuracy goes.




```
Accuracy for K 1 is 0.672322375397667
Accuracy for K 2 is 0.6521739130434783
Accuracy for K 3 is 0.6871686108165429
Accuracy for K 4 is 0.6956521739130435
Accuracy for K 5 is 0.6914103923647932
Accuracy for K 6 is 0.6956521739130435
Accuracy for K 7 is 0.6956521739130435
Accuracy for K 8 is 0.694591728525981
Accuracy for K 9 is 0.7073170731707317
Accuracy for K 10 is 0.7030752916224814
Accuracy for K 11 is 0.7104984093319194
Accuracy for K 12 is 0.7020148462354189
Accuracy for K 13 is 0.704135737009544
Accuracy for K 14 is 0.6998939554612937
Accuracy for K 15 is 0.7020148462354189
Accuracy for K 16 is 0.7051961823966065
Accuracy for K 17 is 0.6956521739130435
Accuracy for K 18 is 0.7030752916224814
Accuracy for K 19 is 0.6977730646871686
Accuracy for K 20 is 0.7051961823966065
Accuracy for K 21 is 0.7083775185577943
Accuracy for K 22 is 0.711558854718982
Accuracy for K 23 is 0.7094379639448568
Accuracy for K 24 is 0.7094379639448568
Accuracy for K 25 is 0.7104984093319194
Accuracy for K 26 is 0.7062566277836692
Accuracy for K 27 is 0.7094379639448568
Accuracy for K 28 is 0.7051961823966065
Accuracy for K 29 is 0.7020148462354189
Accuracy for K 30 is 0.7020148462354189
Accuracy for K 31 is 0.6988335100742312
Accuracy for K 32 is 0.7020148462354189
Accuracy for K 33 is 0.6977730646871686
Accuracy for K 34 is 0.704135737009544
Accuracy for K 35 is 0.704135737009544
Accuracy for K 36 is 0.7094379639448568
Accuracy for K 37 is 0.7083775185577943
Accuracy for K 38 is 0.7073170731707317
Accuracy for K 39 is 0.7126193001060446
Accuracy for K 40 is 0.7104984093319194
Accuracy for K 41 is 0.7104984093319194
Accuracy for K 42 is 0.7094379639448568
Accuracy for K 43 is 0.7168610816542949
Accuracy for K 44 is 0.7136797454931071
Accuracy for K 45 is 0.7147401908801697
Accuracy for K 46 is 0.7126193001060446
Accuracy for K 47 is 0.7136797454931071
Accuracy for K 48 is 0.721102863202545
Accuracy for K 49 is 0.711558854718982
Accuracy for K 50 is 0.7126193001060446
```

Changing K values didn't change the process time. This is because most of the operation time is calculating Euclidean distance not extracting K training points.

```
LOADING DATA...
DATA LOADED...
Computing Euclidean Distance for KNN...
Computing Euclidean Distance is NOW COMPLETED
Estimating CLASS Based on Distance...
Accuracy for K = 50 is 0.7126193001060446

real    7m31.168s
user    7m26.848s
```

```
LOADING DATA...
DATA LOADED...
Computing Euclidean Distance for KNN...
Computing Euclidean Distance is NOW COMPLETED...
Estimating CLASS Based on Distance...
Accuracy for K = 1 is 0.672322375397667

real    7m46.978s
user    7m42.354s
sys     0m6.692s
```

(2) Brief description of how your program works

My program is composed of 3 functions : *readfile*(file), *k_nearest neighbors*(K, train_px, train_ot), and *get_accuracy*(test_label, est_label).

readfile reads in the data and returns the list of photo_id, the list of orientation of the photos (which is the class we want to estimate) and the list of pixel values. *k_nearest_neighbors* takes K, training pixel values and train orientation and uses the algorithm explained above and returns estimated class for the test sets. *get_accuracy* will take the estimated class the previous function returned and compare it with the test label to calculate the performance of the algorithm.

After the execution of the above functions, it will create the document called *output.txt,* which writes the test id and the estimated class of the test sets.

My implementation can take different values for K. As a default, it takes K that had the highest accuracy, which is K=48. However, user can give the list of values for K, for instance; K=[i for i in range(1,11)] and it will test for K's 1 to 10. This may be useful, if you want to get the best K for estimating the test's orientation.

If you also want a plot for multiple Ks, you can also uncomment the plot part in the bottom, and you will get the accuracy graph as given in **(1)**.

(3) Discussion of any problems, assumptions or simplifications

Making Euclidean distance calculation to be simple and cheap as possible was the main theme for solving the problem efficiently. If we assume that the number of test examples to be N, training examples as M and number of dimensions as D, it will be O(N*M*D), which calculating distance alone, is very expensive.

At my first implementation, I tried using numpy.linalg.norm(test_array-train_array) but it was not converging for 10 minutes. So, I changed the algorithm to be cheaper by taking step by step approach: take difference-> np.square->np.sum. This made the algorithm converge in 8 minute.

Also my previous program used to calculate Euclidean distance each time for each K, and extract K training points(which is O(NMDK + NK) and output the estimate class, which was very inefficient. I changed my program to calculate Euclidean distance once and extract different K in one huge distance list (O(NMD +NK)). (NK is for choosing K closest training point for N test data)

Things that I can try to make my program better are as follows. I would have tried taking Manhatten distance other than the Euclidean since it may imply different meaning and classification accuracy.

Also, I could have tried dimension reduction since I have found from R coding that some variables are more important than the other. (PCA etc..)