Face Recognition using kNN classifier and Discrete Cosine Transform for Feature Extraction

Aakash Tripathi and Alexander Yu

Date: 12/11/2020

Course name: Signals and Systems

Section number:

Table of Contents

1 INTRODUCTION

Over the years of technological advancements, many new devices have implemented a feature called face recognition. On the newer iPhones, they got rid of the home button and implemented the new FaceID. The system uses the front camera to grab a picture of who is accessing the phone to then see if it is the owner of the phone. Although this may be the most recent and relevant implementation of face recognition, face recognition has been looked into for the past few decades. Face recognition is the idea of identifying or verifying one's identity solely off of their face. Now, it is used for many different purposes. Like mentioned before, you can access your personal device, and police can use it for investigations. Many people use face recognition without the understanding of how face recognition works. Face recognition is now taking on more applications through time and the goal of this report is to understand how it works and understand many other concepts that go along with face recognition.

Although there are many different ways that face recognition software's functions, generally, most software's follow similar steps. To begin with, pictures or videos are taken of a person's face. The facial features are then used as the main component in a signal that was taken from the aforementioned picture or video. The software then searches its database for similar signals to decipher whether or not the face matches with previously inputted information. This whole process is called linear projection and is the most common technique.

It is important to improve on this program for many reasons. First of all, the system is not always accurate. There are many factors such as lighting that prevent face recognition to work 100% of the time. There are also a lot of cases where people are wrongly put in jail for a misread on the face recognition system. And with the demand of the software being so high, more improvements to ensure accuracy is important. With that being said, engineers like ourselves should want to improve on the software for the safety of the public, and to prevent injustice from happening.

For this lab, the objectives are:

1. Understand face recognition
2. Learn the basics on how to create a facial recognition software
3. Learn about real-life applications of signal processing
4. Understand what a Discrete Cosine (DCT) is
5. Use a Discrete Cosine Transform to create a software implementation of face identification
6. Understanding k-nearest neighbor(kNN)
7. Use kNN to create a face identification software
8. Learn to use MATLAB
9. Learn to plot using MATLAB
10. Apply MATLAB to the software of face recognition

2.1 PART 1: TWO DIMENSIONAL (2-D) DCT AND INVERSE 2-D DCT

To explore face recognition algorithms, the AT&T face dataset (formerly known as the ORL dataset) was utilized. The dataset contains 400 grayscale Portable Gray Map (.pgm) images of size 92 x 112 pixels of 40 individuals, with each pixel value ranging between 0-256. The objective of this section was to become familiar with the AT&T database of images by using MATLAB functions to display and explore the images found in the database.

First a subject's image was chosen to visualize using MATLAB. Figure 1 shows the output of subject 1, image 1 from the dataset. This was achieved using the imread() function on MATLAB and then had the following image displayed using the imshow() function.

Figure 1: Subject 1, Image 1 from AT&T Database

The next step was to calculate the Two-Dimensional Discrete Cosine Transform (2D DCT) of Subject 1, Image 1.

The discrete cosine transform is a very popular algorithm used famously in JPEG compression and many other data compression applications. For the application of face detection, DCT can be very useful. As, DCT removes much of redundant and highly correlated high frequency information from images. DCT also helps reduce computation complexity by reducing memory usage, a factor that can help mitigate training time and complexity for machine learning models.

The 2D DCT was calculated using the dct2() function on MATLAB and displayed using the imshow() function, shown in figure 2.

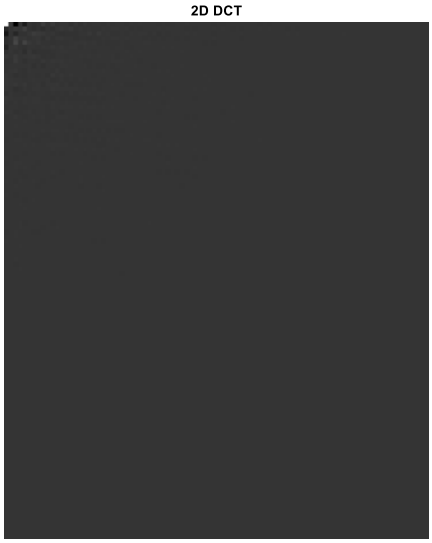

Figure 2: 2D DCT of the input image

The DCT output of the input subject 1 image reveals a mostly blank image with a bright spot on the top left corner of the image. This output is in line with what is expected from the DCT operation. Most of the low frequency data has be encoded at the top left of the image and the high frequency data has been compacted as per the Discrete Cosine Transform.

However, this data is not very clear, at least visually. To fix this, the scale of the DCT output needs to be changed to highlight the bright pixels at the top leftmost of the image. To do this the log magnitude of the DCT was taken using MATLAB, and the output for which is show in figure #. Doing this operation provided us with an image where the gradient of bright pixels in the top left to the dark pixels in the bottom right is more apparent.

Figure 3: Log Magnitude of 2-D DCT

To ensure that there was no data loss during the energy compaction process of DCT of the image shown in figure 1, the inverse DCT can be taken to recover an image that should be close to the original.

Using the inverse DCT function on MATLAB on the 2D DCT of the input image show in figure 2, gives the following output shown in figure 4. Revealing no 'perceived' loss in data from the compression caused by the 2D-DCT.

Figure 4: Inverse DCT image output

2.2 PART 2: FEATURE EXTRACTION

The objective for this section is to understand the provided findfeature() MATLAB function. The goal of the program is two folds. One, to convert the 2D DCT plot into a one-dimensional vector array. Two, to take the converted one-dimensional vector and plot the values to visualize the features of the 2D – DCT images.

To do this the program first takes the image as an input and converts it to a matrix. This is done in MATLAB using the imread() function. Next the code uses the Hoffman algorithm to convert this matrix into a 1-dimensional vector array.

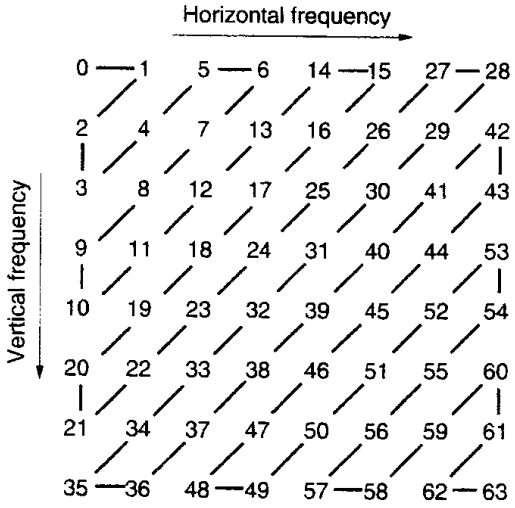

Figure 5: Huffman coding algorithm.

Pictured above in figure 5, is a visual illustration of the Huffman coding algorithm which follows a ‘zigzag’ pattern. The algorithm works in the following manner:

1. Go to the first pixel on the 1st row first column. Read the value and store in a new one-dimensional vector with index 0.
2. Move one pixel to the right and read the value of the said pixel.
3. Move diagonally in negative 45-degree angle one step at a time until the edge of the matrix is met and at each step record the pixel value.
4. Once the edge is met move one-pixel length downwards and record the pixel value
5. Next move 45 degrees upwards one pixel at a time until the topmost edge of the matrix is met. Make sure to record the pixel value at each pixel step made.
6. Do this until there are no more moves left to be made.

Applying the Huffman coding algorithm transforms the 2D DCT 112 x 92 matrix into a one-dimensional vector array of size 10,304. These 10,304 points correspond to the pixel grayscale values of the 2D DCT encoded by the Huffman coding algorithm. Figure 6 represents a chart of the probability of each pixel value being non-zero for a small subset of an entire sample one-dimensional vector array.

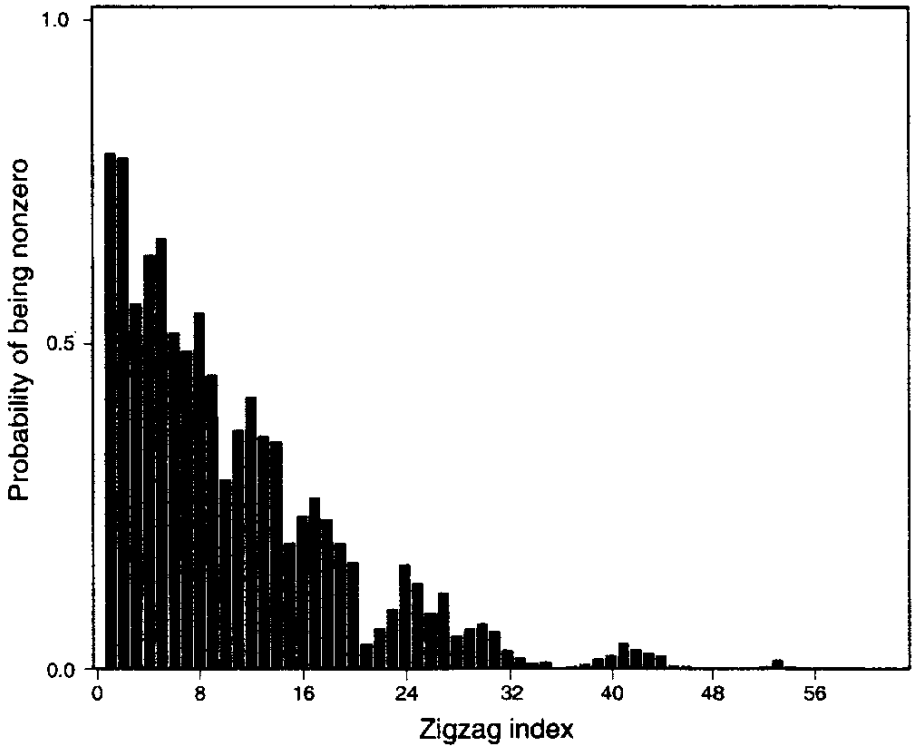

Figure 6: Probability of being nonzero for index of every vector

Figure 6, also highlights a few interesting features of the data acquired from the DCT operation:

1. The distribution of probabilities having the highest magnitude in the front of the vector.
2. There are spikes in the probability distributions at certain ‘zigzag index’
    a. These spikes can be defined as a correlation to a certain feature found in the original .pgm image.

3. The leftmost of index contains the lowest frequency information of the 2D DCT.
4. The probability distribution approaches zero, further down the 1D vector.

Since the value approaches zero as the index values is increased, the overall data required to represent the original image is reduced significantly.

Performing this for two images from the AT&T images, subject 3 face 1 and subject 5 face 1 shown in figures 7 and 8, gives the graphical output shown in figures 9, and 10.

Figure 7: Subject 3, Image 1

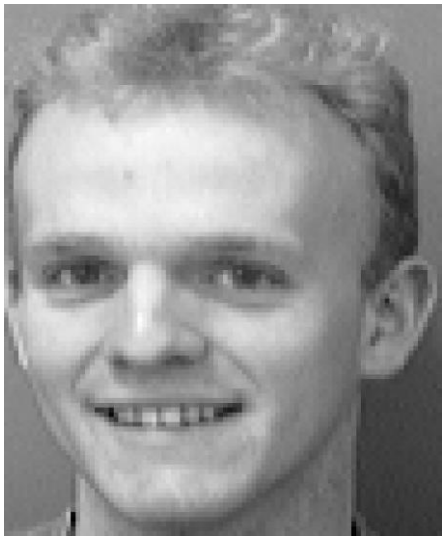

Figure 8: Subject 5, Image 1

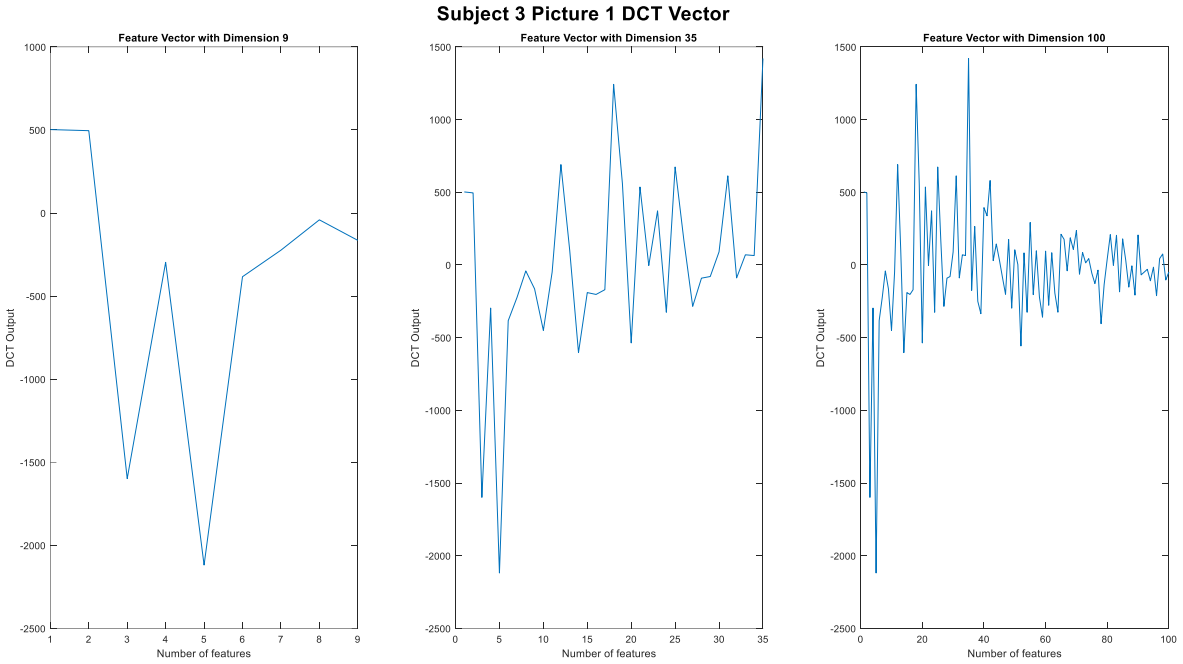

Figure 9: DCT output vs. number of feature vectors with a dimension of 9, 35, 100 for subject 3 image 1

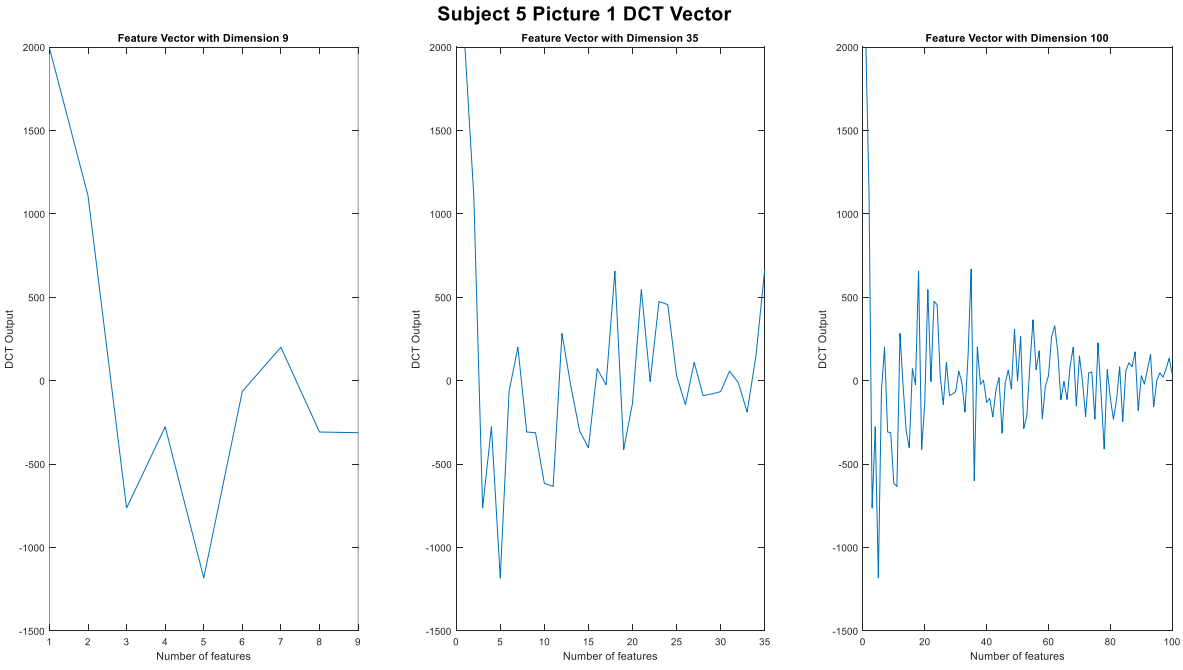

Figure 10: DCT output vs. number of feature vectors with a dimension of 9, 35, 100 for subject 5 image 1

Analyzing the plot outputs for each subjects DCT vectors with varying feature size, it is evident that the features discussed earlier are also present in the DCT output plots of the two subject images.

The graphs show the effect of DCT output convergence to zero, as the number of features size increases. With the plot of DCT output with only 9 features having the least amount of data to represent the original image, and arguably the plot with 100 features containing too many features whose values are non-crucial for keeping the perceived data saved. The best option for the feature vector shape, seems to be in the range of 35-80 as it still provides clear low frequency changes in the original image. The narrower range also provides a good balance between enough data for a machine learning algorithm to train on and keep the data size low enough to not cause any unnecessary computational costs.

2.3 PART 3: TRAINING THE FACE IDENTIFICATION SYSTEM

The objective of this section was to train a face identification system using the provided k-nearest neighbor function called 'face_recog_knn_train.m'.

The KNN is perhaps the simplest machine learning supervised model. This algorithm is best explained using a visual aid. Shown below in figure 11, is an example of a KNN in action. The figure contains 2 classes, Class A and Class B. Class A is highlighted in yellow and Class B is highlighted in purple. Let us say you were to add another data point to the plot, highlighted in red, which class would it belong to? The KNN can solve this problem using a simple algorithm. The K in KNN, is the number of training data points to consider in order to make a prediction. So, for this example, if the K is selected as three, then the algorithm encounters two data points in Class B and one data point in Class A, hence the algorithm picks the red data point to be part of Class B. Now let us say that we increase the K from 3 to 6. This changes the total data count in favor of Class A, four for Class A and two for Class B. Hence the algorithm decides to label the red point as part of Class B. One thing to note for KNNs Is that the K value must always be odd, in order to ensure majority vote.

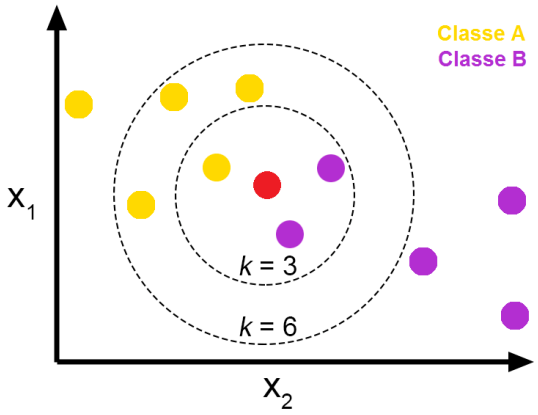

Figure 11: KNN example

For the purposes of this face identification system described in this report, the kNN was trained using the supplied function and the first five images of each of the 40 subjects in the AT&T database.

The function takes two parameters as inputs: 'subject_range' and 'dct_coef'. Where the subject_range is the user specified range of faces to be used for training the kNN; with the

possible range of 1-40. The ‘dct_coef’ input is the user specified length of the feature DCT vector that will also be used for training.

The following function input arguments were given for training the kNN for this report:

‘subject_range’ = [1 40]
‘dct_coef’ = 70

After parsing the input arguments, the ‘findfeature.m’ function converts the image inputs into a one-dimensional feature vector for each of the five images per subject. These feature vectors have a length of 70 as per the user input requirements and are added to one of the two outputs of the ‘face_recog_knn_train.m’ function labeled, ‘trdata_raw’ with a size 200 x 70 matrix. The other output from the function is the ‘trclass’ array. This array encodes the label corresponding to the subject number in the AT&T database.

At the end of the function execution, a raw_data.mat file is saved containing the following variable:

1. dct_coef
2. trdata_raw
3. trclass
4. f_range
5. nsubjects

dct_coef
Reading the ‘dct_coef’ gives a data of value 70, as specified by the input parameter, representing the length of the feature vectors.

trdata_raw
Reading the variable ‘trdata_raw’ returns a matrix that has the shape the 200 x 70. The 200 rows correspond to the 5 images of 40 subjects, and the 70 are the dct_coef values.

trclass
trclass is the labelled information for each of the subject.

f_range
‘f_range’ is the range of subjects being trained. Reading the variable gives a matrix of size 1 x 40 matrix.

nsubjects
‘nsubjects’ is the number of subjects being trained and as expected when the data is read it returns the value of 40, for the 40 subjects.

2.4 PART 4: PERFORMANCE EVALUATION OF THE FACE IDENTIFICATION SYSTEM – CLEAN DATA

Finally, for this section the objective was to evaluate the performance of the face identification system utilizing the remaining 5 images of the 40 subjects in the AT&T database. The

performance was measured by evaluating the accuracy metrics of the algorithm. Where the goal of the algorithm was to correctly label the corresponding subject images, and to plot a graph of the metric to perform hyperparameter tuning.

To accomplish this a new MATLAB script was written containing two functions. The goals of the functions were to evaluate the performance and to plot a 3D plot with the data collected from function one, respectively.

Function one, called recogEval() utilized the findfeatures() method that was explored in part 2 of this report, to create the feature vectors of the images 6-10 of the subject for evaluation. The algorithm was tested for a variety of K values, ranging from 1-7 at intervals of 2, and dct coefficients from 25-100 at intervals of 15.

The feature vector was then evaluated against the pretrained 200 feature vectors in the training data. Next, the Euclidian distance was calculated between both vectors and had the smallest values stored in a matrix using the second function in the script called ‘score’. The number of total values of distance calculated is dependent on the k value input to the script. For each of the index values in the array, a prediction of a subject was outputted into another 1d vector array. This array was then used to compare to the real labels of the tested images, and a accuracy vector was generated that kept the score of number of correct prediction against total predictions.

The highest accuracy rate was 92.5% for k = 1, when the dimension of feature vector was 55 and the lowest result at 78.5% for k = 7 at dimension 25. Figures 13 and 14, highlight a correlation between the changing K values, dimension of feature vectors, and their corresponding accuracy. The relation of model accuracy to changing K is very evident. As, the K value is increasing, the overall accuracy decreases. This is in line with how kNNs operate, as the number of neighbors increase (k values) the lower the accuracy of the model gets.

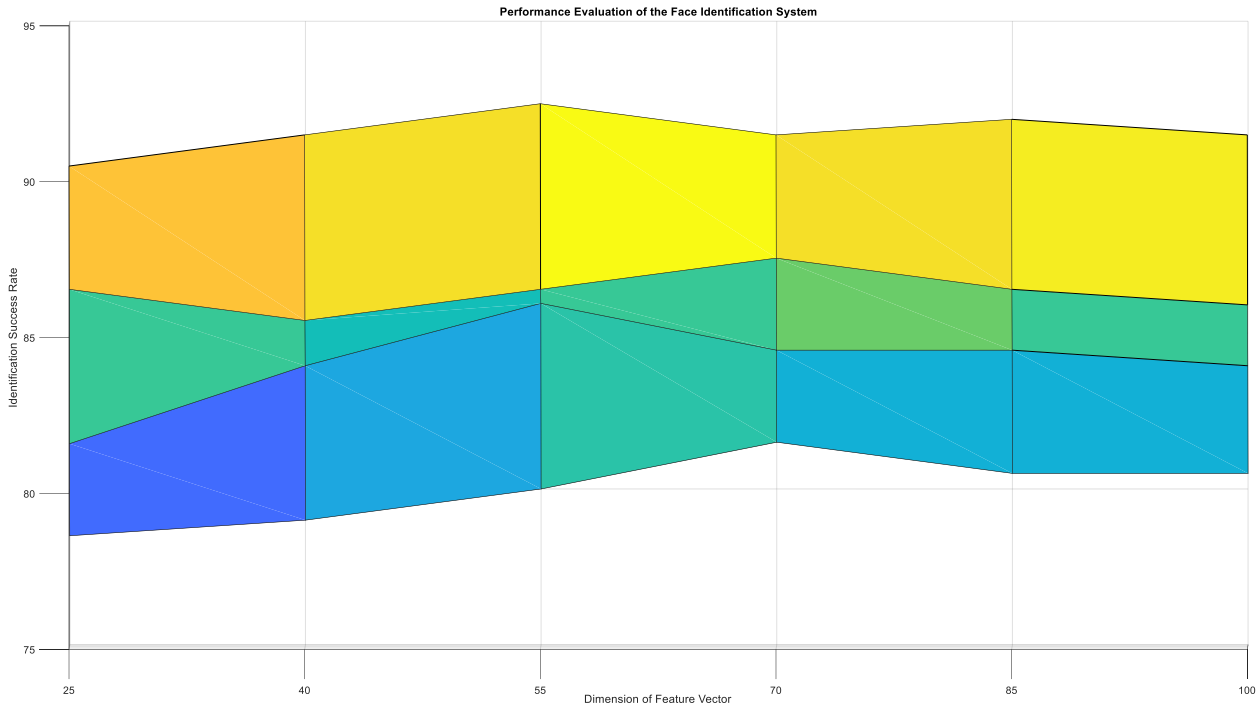

Figure 12: Impact on accuracy by dimension of feature vector

Another key relation that is also apparent in the plot, show in figure 12, is the relation between the dimensions of feature vector and its impact on model accuracy. The accuracy did not see any improvement past a dimension feature vector of 55, showing that the model has most likely begun to either over or under fit.

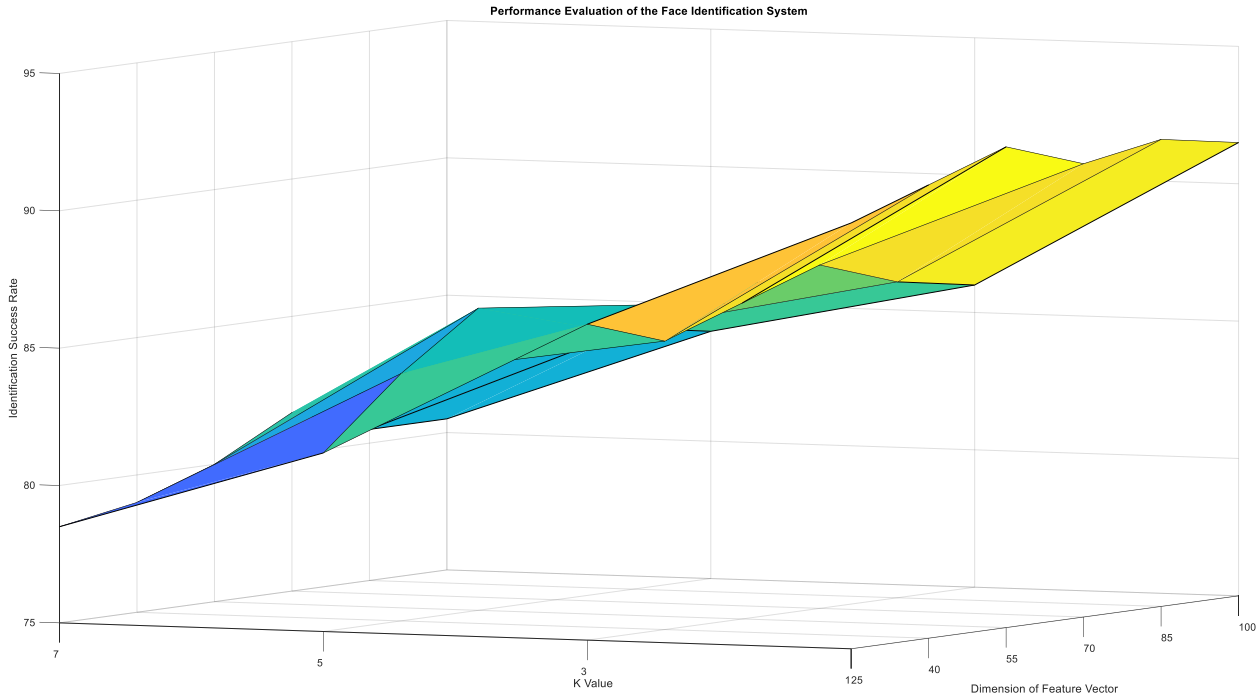

Figure 13: Performance plot of facial recognition kNN algorithm

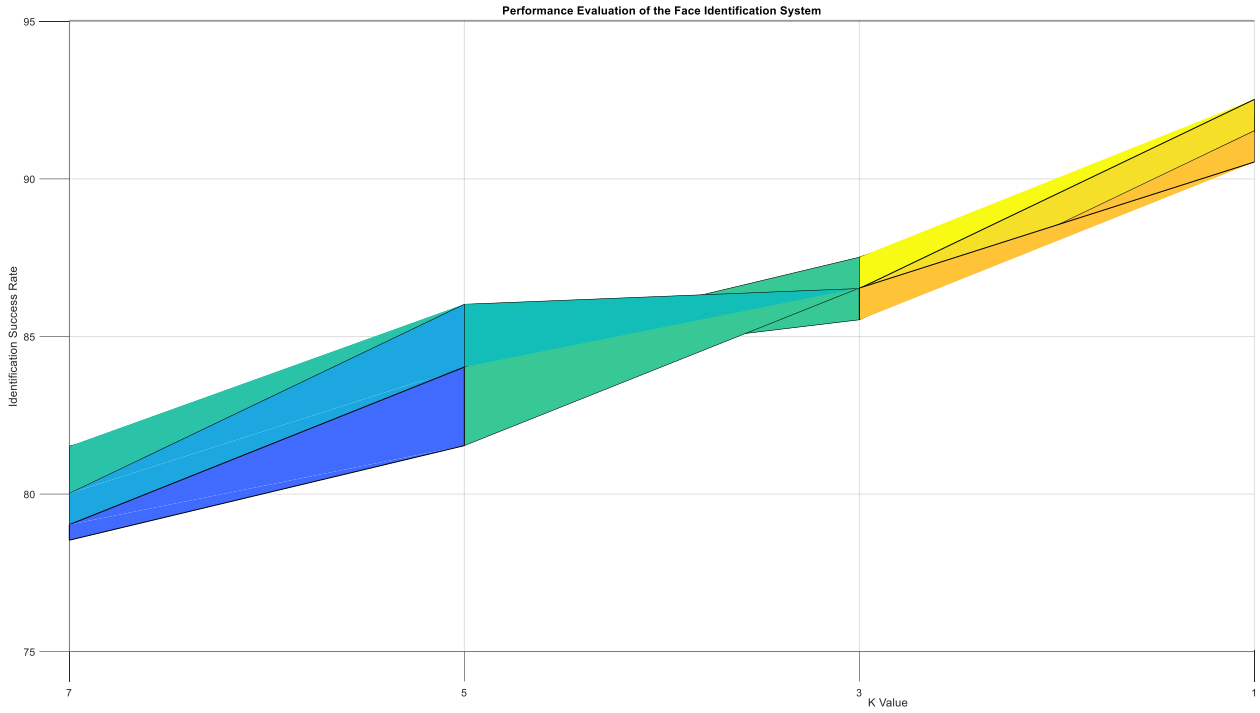

Figure 14: Performance plot in relation to k value of facial recognition kNN algorithm

3 SUMMARY AND CONCLUSIONS

In conclusion, the overall lab asked for the group to design an algorithm that would act as a reliable facial recognition system. This system was to match face to a face database run by AT&T, with 40 subjects and 10 images of each subject. The setup of the lab allowed the group to work in a fashion that is known to be the standard.

Overall, the objectives that were set for the lab were all met. Part 1 required us to take an image, and find its discrete cosine transform of the image. We then took the logarithmic form of the DCT to give a clear image of the frequency. The next part took the DCT and turned it into the feature vector. This was done over different dimensions that were given to us. We also compared two images using these feature vectors. For part 3, we used the given file to train the system, by getting the feature vectors of the first five images of each subject. We also analyzed the data from the previous function of each feature vector. In part 4, MATLAB code was written to analyze the performance of the facial recognition software, with our software performing at 92% success rate. It uses the output file of part 3 and scans the last 5 images of the 40 subjects. The process then repeats with different kNN values and number of features. And the matrix is then plotted on a 3D graph. And for the objectives where we were to understand or learn certain topics, extensive research on the topic was done to get a deeper understanding of what is truly happening through each step of the process.

As stated before, this lab is crucial for ECE's to hop on. Facial recognition is something that is more prominent and is now part of people's everyday lives. With facial recognition being used for security of personal belongings, and ways of catching suspects of crime, we need to make sure that software of this kind is 100% accurate. iPhones store passwords and allow people to access their credit card using facial recognition. Any mistakes from the software, that person's personal information is in limbo. And who knows how far we advance with facial recognition. Soon enough more technology may use it. The sky's the limit for facial recognition.

4 REFERENCES

2-D discrete cosine transform - MATLAB dct2. (n.d.). MATLAB. https://www.mathworks.com/help/images/ref/dct2.html

Newton, J. Z. (2015). https://digitalcommons.ciis.edu/cgi/viewcontent.cgi?article=1079&context=ijts-transpersonalstudies. *International Journal of Transpersonal Studies, 34*(1–2), 172–186. https://doi.org/10.24972/ijts.2015.34.1-2.172

Peterson, L. E. (2009, February 21). *K-nearest neighbor - Scholarpedia*. Scholarpedia. http://www.scholarpedia.org/article/K-nearest_neighbor