Table of Content

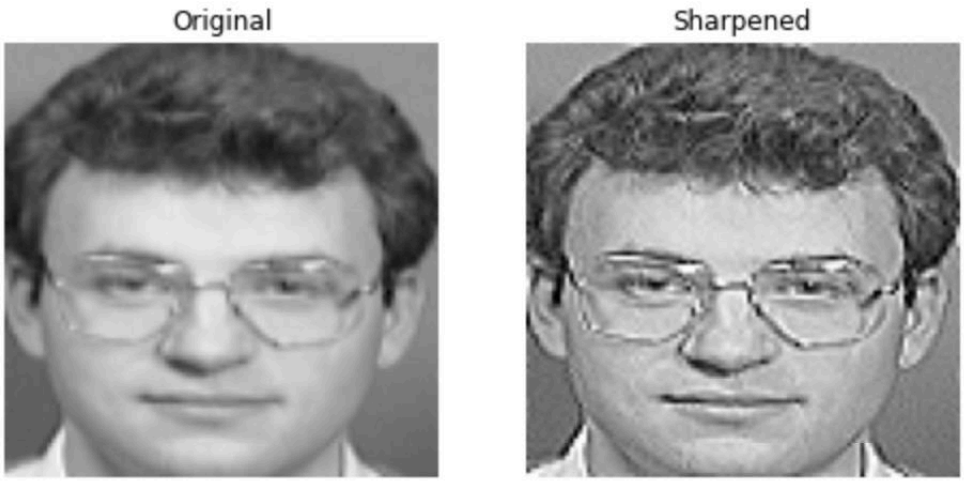
1. Introduction

Nowadays, face recognition is able to achieve high state of the art accuracy through Deep Learning algorithms such as Convolutional Neural Networks or Vision Transformers (Dosovitskiy et al., 2021). However, Deep Learning methods usually requires a large amount of dataset, numbering in the thousands to millions of datasets in order to prevent overfitting (Seguin, 2017). Since we do not have access to this large amount of dataset, a better way is to use a simpler Machine Learning classification model such as Nearest Centre Classifier. Since the models are simpler and not suitable to handle complex dataset such as face images, we can use the Eigenface approach (Turk & Pentland, 1991). We first apply Principal Component Analysis (PCA) to lower the dimensions of the faces. Once the lower dimension eigenfaces are generated from the PCA method, we use these as training data for the Nearest Centre Classifier. This will require less computational power and can be done in a shorter time (Lee, 2022). The trade-off of this method is reduction in accuracy due to the information loss when compressing. Hence, the purpose of this report is to design and train three different types of face recognition systems and compare their performance. All of the necessary files needed can be found in G0191 Files.

2. Face Image Dataset

Before training our classifiers, we should first collect and pre-process the face images. Our face images are obtained from individual group members and also from the internet. There are 46 people in our training data, where 6 are from the pictures collected by the members while 40 more are found from the internet. Each person has 5 greyscale (.pgm) face images with different orientation totalling up to 230 images. The number of test images are 15, which mainly consists of the images obtained from the members. The face images are separated into training dataset and testing dataset. The face images contained in each folder are added into our array dataset, with the name of the folder acting as its label and consequently added into the array. The face images contained in test is added into the array, with the name of each image added into the array acting as its label. Pre-processing is also carried on each of the images through image sharpening. All of images are reshaped into size of (100, 100). Then, the images are sharpened by a filter with a kernel. The filtering is carried out by convolution

of a 2D matrix with our image. In this case, the matrix is 3x3 sharpening kernel. The reason for this is because the filter is most useful for highlighting edges and lines in an image. As shown in the sample image before and after applying the sharpening process, it can be seen that the curves and edges of the face has been highlighted:



3. Eigenface + Nearest Centre Classifier Approach

Once all the training images are pre-processed, we can convert the images into eigenfaces and use a Nearest Centre Classifier for the face recognition. In order to convert the face images into the eigenfaces, we first must first carry out Principal Component Analysis (PCA) on each of the faces. The outline of the Eigenface algorithm is as follows:

1. Mean normalize the given data X into X_C that the new data X_C has a mean of 0.
2. Compute the covariance matrix of the mean normalized data

$$\Sigma = \frac{1}{n} {X^T}_c X_c$$

3. Compute the eigenvectors and the eigenvalues of the covariance matrix by carrying out Eigenvalue decomposition:

$$\Sigma = VUV^T$$

where represents V the matrix whose i^{th} column is the eigenvector of the corresponding eigenvalue in U , and U represents a matrix with the eigenvalues along its diagonal.

4. Project the given data points onto the frst p Eigenvectors by multiplying the first p Eigenvectors (also known as projection vectors) by the mean normalized data:

$$X_r = X_c \cdot V[:n\_components]$$

5. Obtain the eigenfaces by dotting the centered data with the mean normalized projection matrix

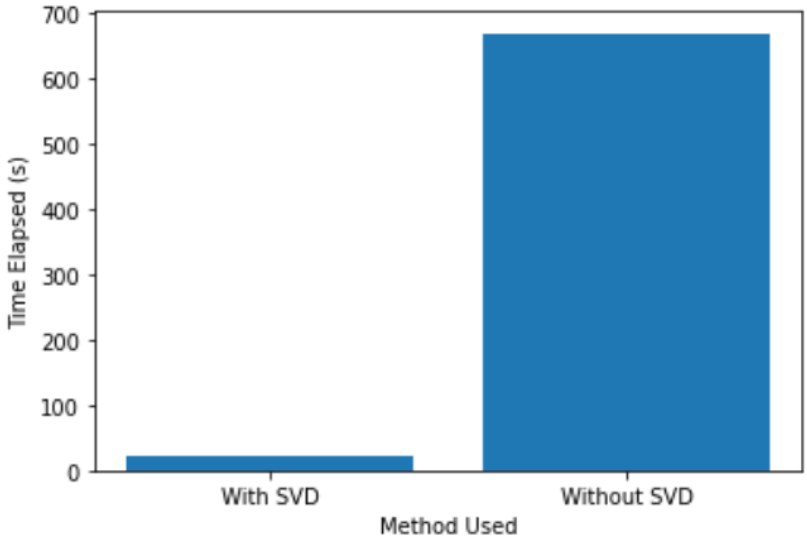
$$X_{eigenfaces} = {X^T}_c \cdot X_r$$

However, the computation of eigenvalues and eigenvectors from the Covariance matrix at step 3 requires a long time to run. This is due to the large dimensions of the Covariance matrix (10000 x 10000). When using a 12GB RAM, the time taken to compute the eigenvalues and eigenvectors could take up to 15 to 20 minutes. Another way to obtain the eigenvectors and eigenvalues is by performing Singular Value Decomposition (Wall et al., 2003).

3. (alternative) Use Singular Decomposition on the centred data to get the root eigenvalue and eigenvectors:

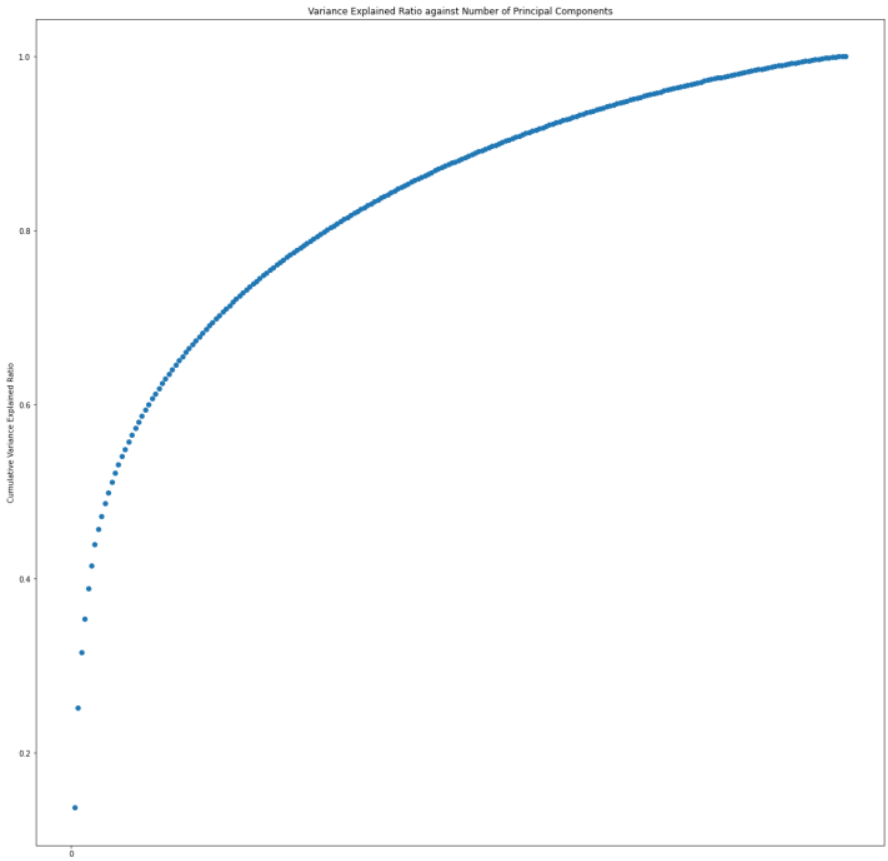
$$X_c = U\Sigma V^T$$

Computing the eigenvectors and eigenvalues by applying the Singular Value Decomposition on the centred matrix is much faster than computing it from the Covariance Matrix, with a speed up of at least 20-35 times depending on the system memory. When using a system of 12GB RAM, the time taken to compute the eigenvalues and eigenvectors from the Covariance Matrix is at least 15-20 minutes. However, computing the eigenvalues and eigenvectors using SVD takes only 20-30 seconds. A bar chart is shown below to highlight the difference:

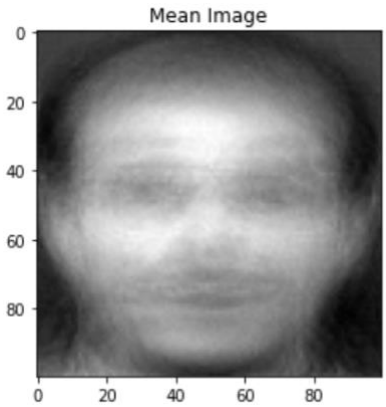


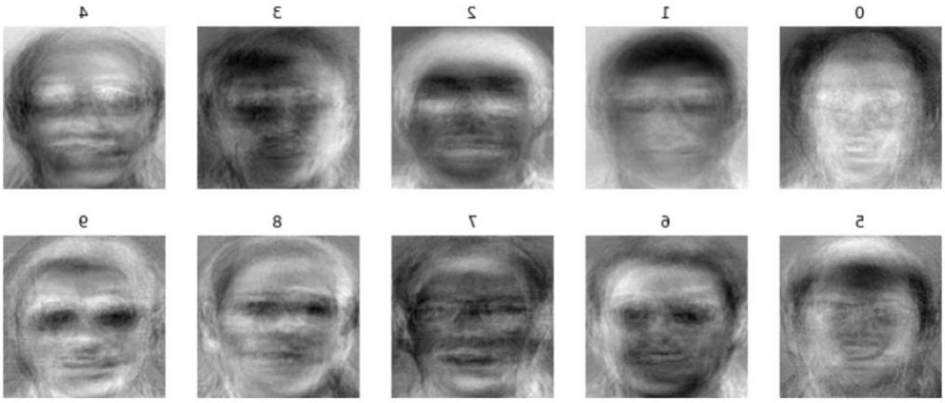
Using the Eigenface algorithm outlined, the eigenfaces can be computed from our training images. The Variance Explained Ratio against the Number of Principal Components are also determined. The Variance Explained Ratio is the ratio between the variance of the

principal component and the total variance (Cheplyaka, 2017). For example, from the graph below shown, we can see that the number of principal components needed to achieve a 95% explained variance is at the range of 165 ~ 170.



The mean image and the top 10 eigenvectors obtained from our train images when computing the eigenfaces are also shown:

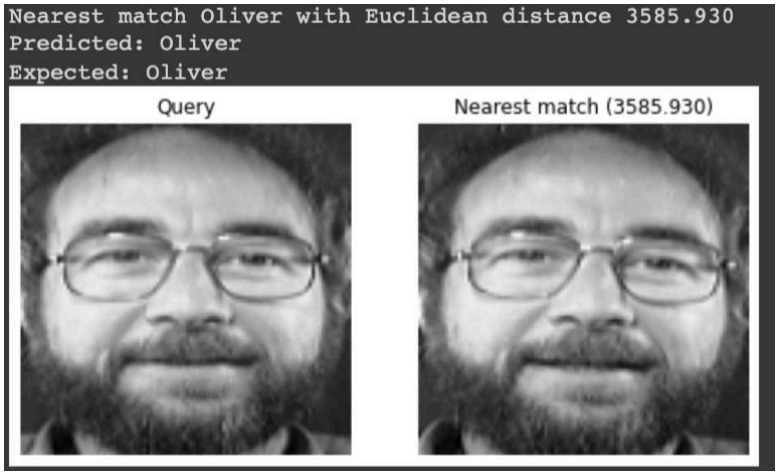


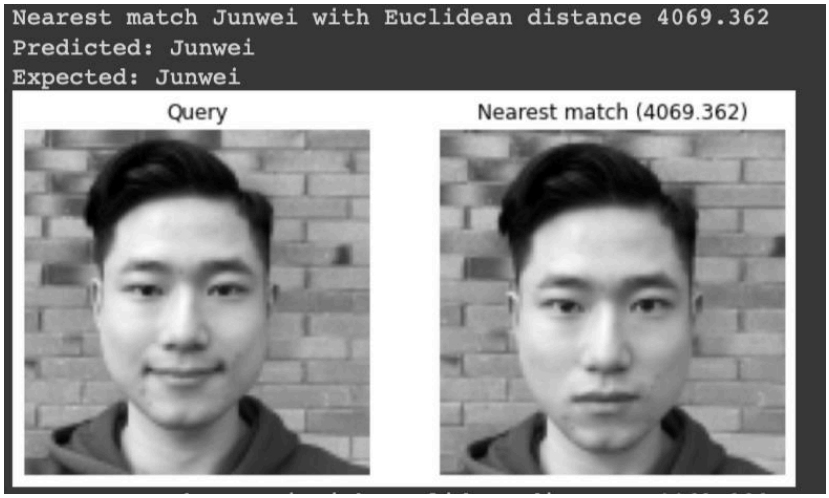


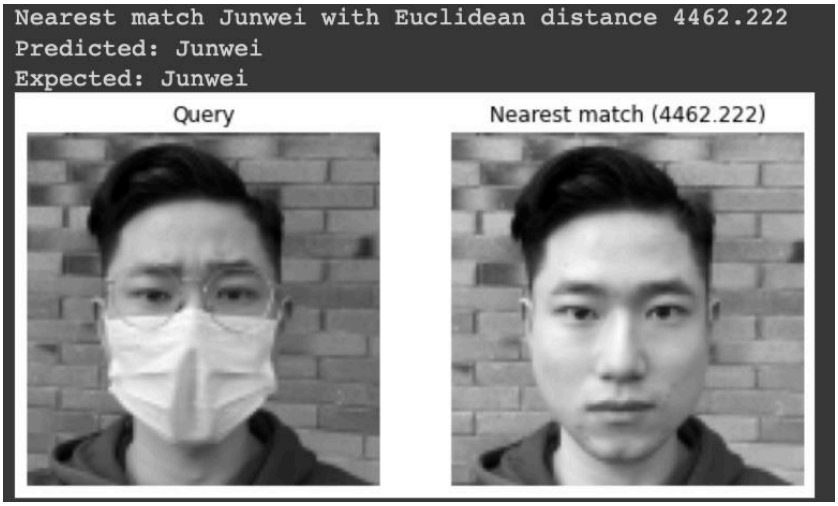
Once the eigenfaces have been obtained, new test images can be classified using Nearest Centre Classifier / Minimum Distance Classifier. The outline of the classifier is as follows (Wang, 2022):

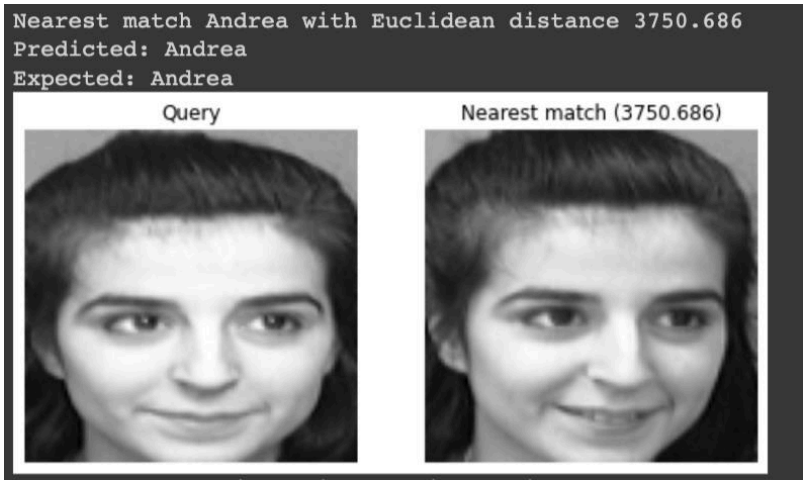
1. For each object class w_k, we find the centroid m_k of the set of samples ${x_1}^k, {x_2}^k, .., {x_N}^k$ belonging to class w_k.
2. For any object feature vector, x, find the distances of x to each of the class centroids (or prototypes).
3. Then we classify an unknown object, x, as belonging to the class whose centroid (m_k) is closest (i.e. having minimum distance) to the unknown object's feature vector.

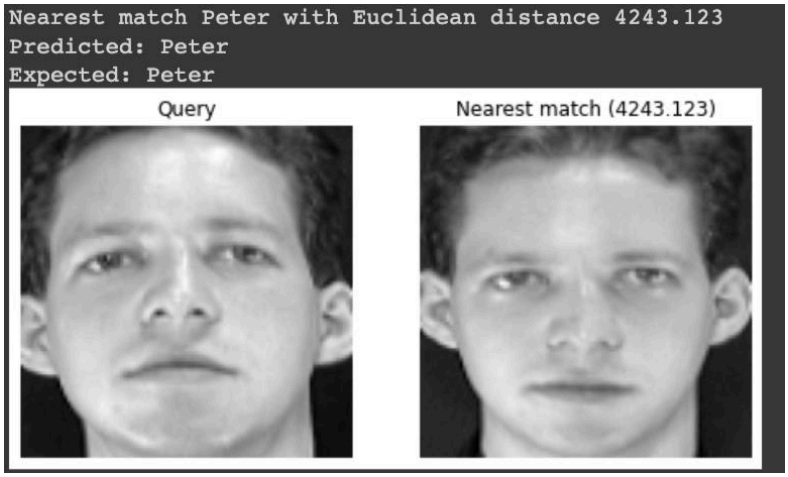
In our implementation, no clustering of centroid is carried out due to to the low number of images in each class. Before classifier, each of the images are transformed into eigenfaces using the weights from our training data. Then, the NCC classifier measures the Euclidean Distance of the transformed test images with the weights of the trained eigenfaces, and classifies according to the lowest Euclidean Distance. If the Euclidean Distance is larger than 6000, then the following image is labelled as an 'unknown' image. Below shows the performance of our classifier on our test set:


Nearest match Oliver with Euclidean distance 3585.930
Predicted: Oliver
Expected: Oliver
Query
Nearest match (3585.930)

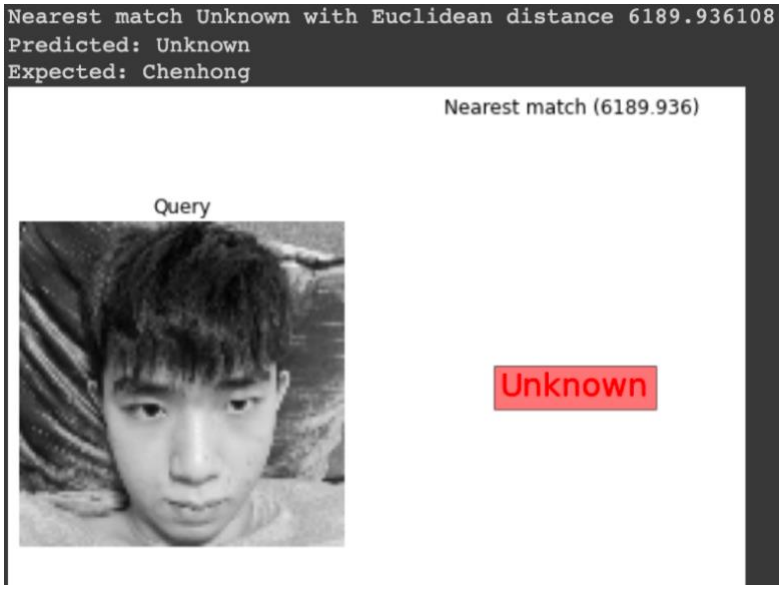

Nearest match Junwei with Euclidean distance 4069.362
Predicted: Junwei
Expected: Junwei
Query
Nearest match (4069.362)

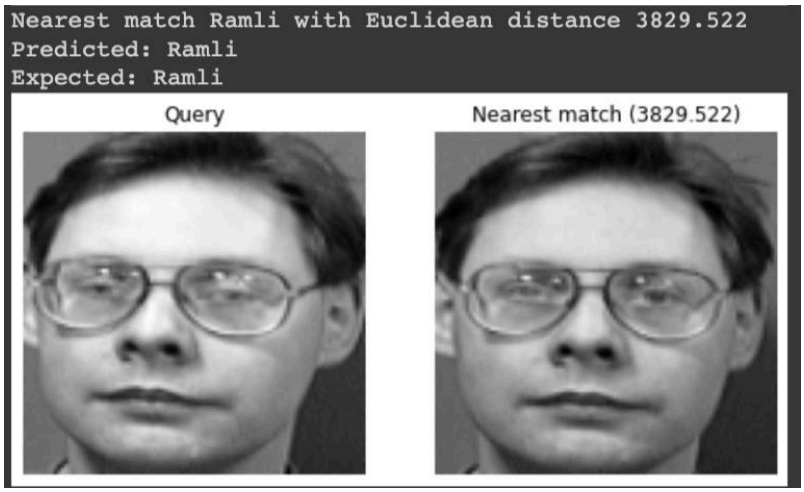

Nearest match Junwei with Euclidean distance 4462.222
Predicted: Junwei
Expected: Junwei
Query
Nearest match (4462.222)


Nearest match Andrea with Euclidean distance 3750.686
Predicted: Andrea
Expected: Andrea
Query
Nearest match (3750.686)

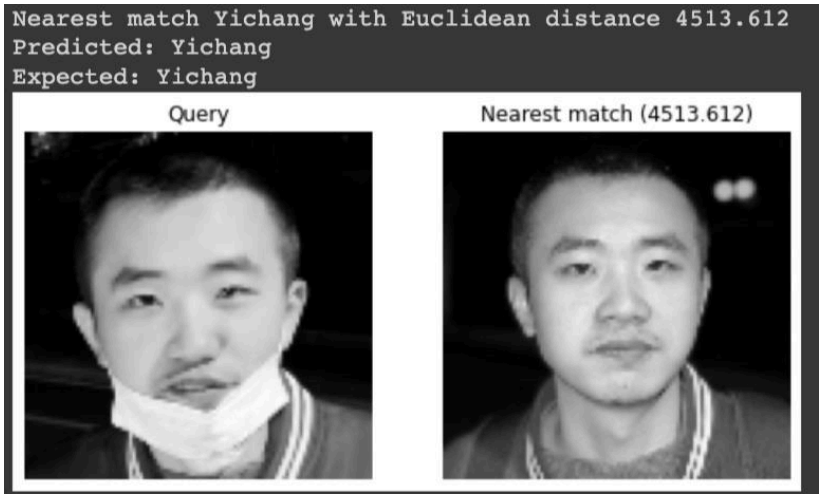

Nearest match Peter with Euclidean distance 4243.123
Predicted: Peter
Expected: Peter
Query
Nearest match (4243.123)

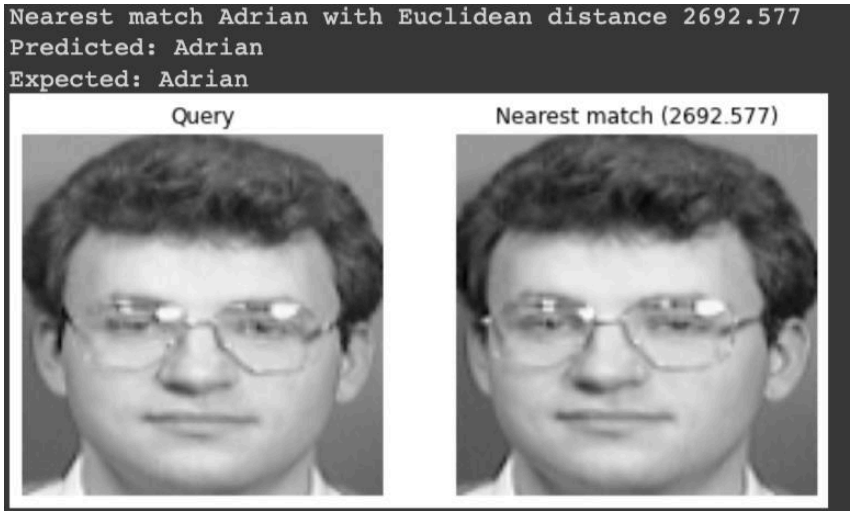

Nearest match Kechen with Euclidean distance 4537.706
Predicted: Kechen
Expected: Kechen
Query
Nearest match (4537.706)


Nearest match Unknown with Euclidean distance 6189.936108
Predicted: Unknown
Expected: Chenhong
Nearest match (6189.936)
Query
Unknown


Nearest match Ramli with Euclidean distance 3829.522
Predicted: Ramli
Expected: Ramli
Query
Nearest match (3829.522)

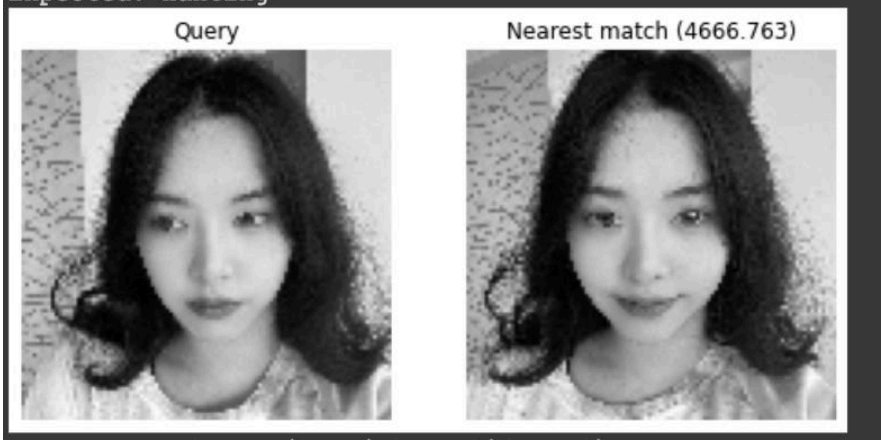

Nearest match Unknown with Euclidean distance 7011.643432
Predicted: Unknown
Expected: Chenhong
Nearest match (7011.643)
Query
Unknown


Nearest match Yichang with Euclidean distance 4513.612
Predicted: Yichang
Expected: Yichang
Query
Nearest match (4513.612)

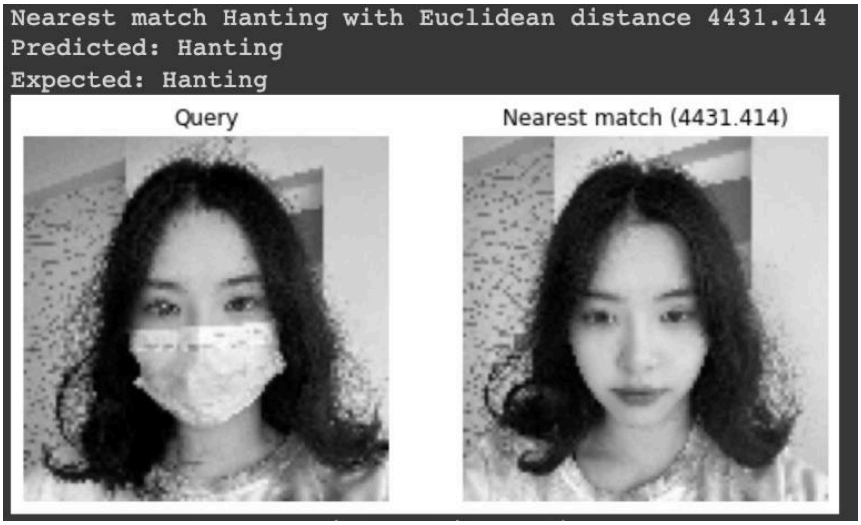

Nearest match Adrian with Euclidean distance 2692.577
Predicted: Adrian
Expected: Adrian
Query
Nearest match (2692.577)


Nearest match Yichang with Euclidean distance 4392.797
Predicted: Yichang
Expected: Yichang
Query
Nearest match (4392.797)

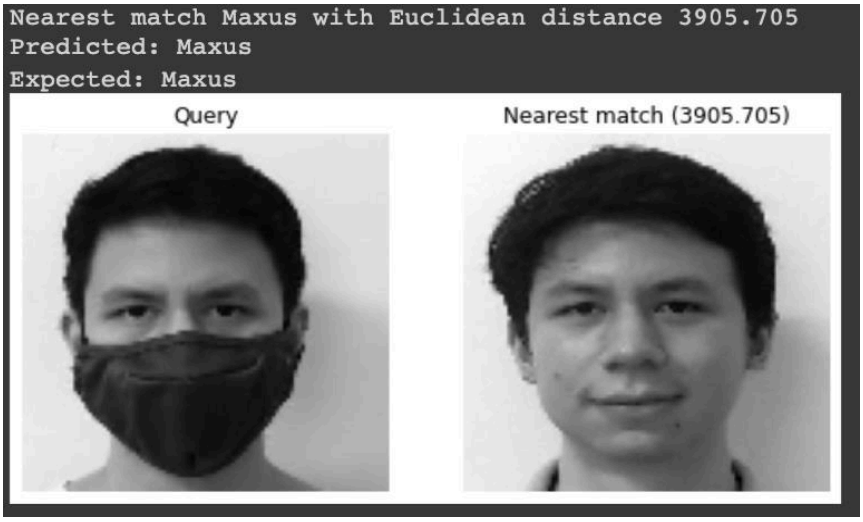
```
Nearest match Hanting with Euclidean distance 4666.763
Predicted: Hanting
Expected: Hanting
```




```
Nearest match Hanting with Euclidean distance 4431.414
Predicted: Hanting
Expected: Hanting
```




```
Nearest match Maxus with Euclidean distance 3905.705
Predicted: Maxus
Expected: Maxus
```



From the results shown above, it can be seen that our PCA + NCC classifier predicts correct for 13 out of 15 test images with accuracy of 86.67%. It is able to cope with in-plane rotation (Kechen), and also able to deal with images involving face masks (Maxus, Hating & Junwei). An image of partially covered face (Yicheng) can also be recognized by our classifier. However, it is unable to identify Chenhong's face images due to the large difference in lighting and in-plane rotations. Hence, the images are labelled "unknown" by our classifier.

4. Deep Learning (Convolutional Neural Network) Approach

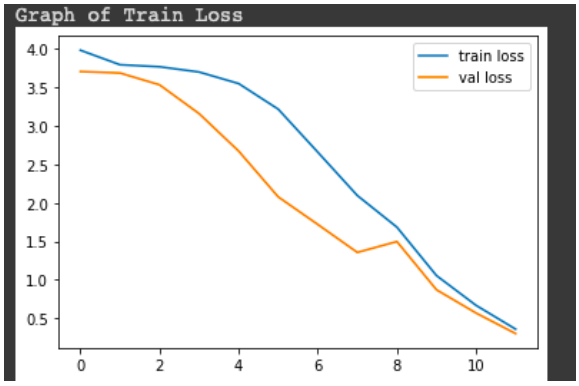
Now that we have the seen the performance of our simple PCA + NCC approach, we will look at Deep Learning on the face datasets as a comparison. The Deep Learning algorithm used will be a Convolutional Neural Network, a neural network widely used for image recognition and image classification problems (Valueva et al., 2020). The summary of the model used is:

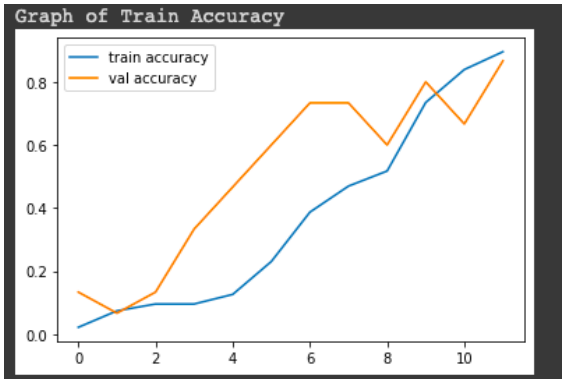
```
Model: "sequential_50"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_278 (Conv2D)         (None, 156, 156, 32)      2432

 conv2d_279 (Conv2D)         (None, 156, 156, 32)      9248

 max_pooling2d_138 (MaxPooli  (None, 78, 78, 32)       0
 ng2D)

 conv2d_280 (Conv2D)         (None, 78, 78, 32)        9248

 conv2d_281 (Conv2D)         (None, 78, 78, 32)        9248

 max_pooling2d_139 (MaxPooli  (None, 39, 39, 32)       0
 ng2D)

 conv2d_282 (Conv2D)         (None, 39, 39, 64)        18496

 conv2d_283 (Conv2D)         (None, 39, 39, 64)        36928

 max_pooling2d_140 (MaxPooli  (None, 19, 19, 64)       0
 ng2D)

 flatten_46 (Flatten)        (None, 23104)             0

 dense_88 (Dense)            (None, 160)               3696800

 dense_89 (Dense)            (None, 46)                7406

=================================================================
Total params: 3,789,806
Trainable params: 3,789,806
Non-trainable params: 0
_________________________________________________________________
```

As shown in the model summary, the Convolutional Neural Network used is a simple one with small number of parameters. This is to prevent overfitting due to the low number of images in our training dataset. Overfitting is when the large capacity of the neural network 'copies' the training data which results in it doing badly on unseen data and not generalize well. When fitting the model, the model is first trained on our training data and is evaluated on the test data set for validation. Below shows the number of epochs along with the accuracies:

```
Epoch 1/12
2/2 [==============================] - 26s 11s/step - loss: 3.9830 - accuracy: 0.0217 - val_loss: 3.7063 - val_accuracy: 0.1333
Epoch 2/12
2/2 [==============================] - 23s 11s/step - loss: 3.7944 - accuracy: 0.0739 - val_loss: 3.6872 - val_accuracy: 0.0667
Epoch 3/12
2/2 [==============================] - 24s 11s/step - loss: 3.7682 - accuracy: 0.0957 - val_loss: 3.5338 - val_accuracy: 0.1333
Epoch 4/12
2/2 [==============================] - 24s 11s/step - loss: 3.7000 - accuracy: 0.0957 - val_loss: 3.1597 - val_accuracy: 0.3333
Epoch 5/12
2/2 [==============================] - 24s 11s/step - loss: 3.5498 - accuracy: 0.1261 - val_loss: 2.6727 - val_accuracy: 0.4667
Epoch 6/12
2/2 [==============================] - 24s 11s/step - loss: 3.2174 - accuracy: 0.2304 - val_loss: 2.0781 - val_accuracy: 0.6000
Epoch 7/12
2/2 [==============================] - 24s 11s/step - loss: 2.6589 - accuracy: 0.3870 - val_loss: 1.7201 - val_accuracy: 0.7333
Epoch 8/12
2/2 [==============================] - 23s 11s/step - loss: 2.0955 - accuracy: 0.4696 - val_loss: 1.3569 - val_accuracy: 0.7333
Epoch 9/12
2/2 [==============================] - 24s 11s/step - loss: 1.6848 - accuracy: 0.5174 - val_loss: 1.4979 - val_accuracy: 0.6000
Epoch 10/12
2/2 [==============================] - 24s 11s/step - loss: 1.0558 - accuracy: 0.7348 - val_loss: 0.8715 - val_accuracy: 0.8000
Epoch 11/12
2/2 [==============================] - 24s 11s/step - loss: 0.6692 - accuracy: 0.8391 - val_loss: 0.5694 - val_accuracy: 0.6667
Epoch 12/12
2/2 [==============================] - 24s 11s/step - loss: 0.3602 - accuracy: 0.8957 - val_loss: 0.3027 - val_accuracy: 0.8667
Time taken for training: 323.61
```

The performance of the simple Convolutional Neural Network works quite well, with a 89.57% training accuracy and 86.67% validation accuracy which is the test accuracy. The test accuracy obtained is the same as the PCA + NCC approach, with 12 out of 15 images classified correctly. Throughout the entire training, the validation accuracy is mostly higher than the training accuracy. This usually happens when the testing dataset is small. A graph of training loss against validation loss and training accuracy against validation accuracy is shown below:





Our Convolutional Neural Network is able to cope with in-plane rotations and images involving masks. However, it is unable to recognize low resolution images. The performance of the neural network on our test data is shown below:

Predicted: Andrea
Expected: Andrea

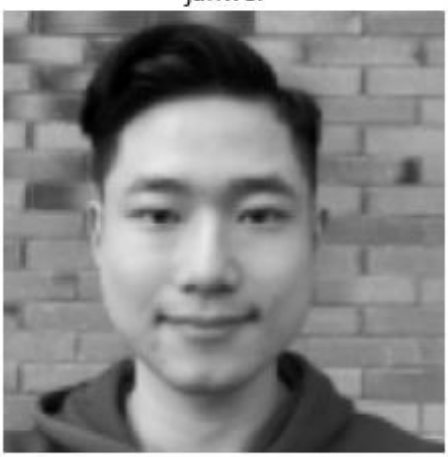
Andrea

Predicted: Oliver
Expected: Oliver

Oliver

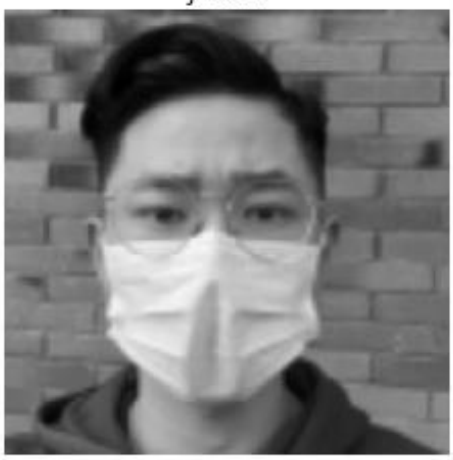
Predicted: Junwei
Expected: Junwei

Junwei



Predicted: Junwei
Expected: Junwei

Junwei



Predicted: Peter
Expected: Peter

Peter

Predicted: Kechen
Expected: Chenhong

Kechen

Predicted: Chris
Expected: Ramli

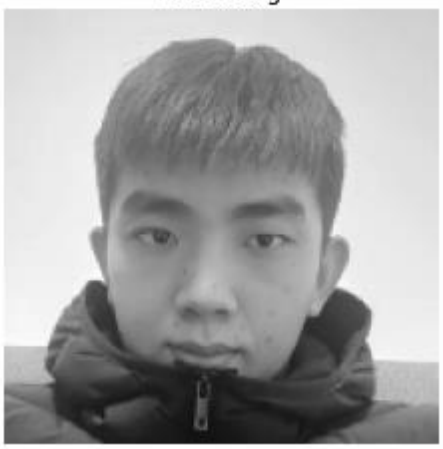
Chris

Predicted: Kechen
Expected: Kechen

Kechen

Predicted: Chenhong
Expected: Chenhong

Chenhong



Predicted: Yichang
Expected: Yichang

Yichang

Predicted: Adrian
Expected: Adrian

Adrian

Predicted: Yichang
Expected: Yichang

Yichang

5. FaceNet + K-Nearest Neighbour Classifier Approach

As shown in the results obtained above, the performance of our Convolutional Neural Network model is the same as our Eigenface + NCC approach. However, the training accuracy gets stuck at 89.57%. This is due to the low amount of training data which makes it difficult for Deep Learning algorithms to work well (Najafabadi et al., 2015). Instead, a better approach for deep learning on small image datasets is to use a pretrained model. In our situation, a pretrained model is a model that was previously trained on a large face image dataset for

image-classification purposes. If this original model is trained on a general and large enough dataset, it can maintain the spatial hierarchy of features learned and act as a generic model for recognizing all types of faces. Hence, its features can prove useful for many different computer vision problems that involves faces, even though these new problems may involve completely different faces than those of the original task. In this case, we will be using FaceNet, a ResNet-Inception Deep Learning model trained on 1 million celebrity face images. The pretrained FaceNet model is provided by Hiroki Tanai at FaceNet in the **model** directory or our files in the **sample_model** directory. Even though FaceNet is initially trained on colored images, it should work well with our grayscale images due to the facial features already learned in the model.

However, the FaceNet model layers is enormous compared to our Convolutional Neural Network. The large size of the model makes it difficult to retrain our face images due to the large magnitude computational power and time needed. Hence, a more suitable method is to transform the face images into face embedding vectors using FaceNet (Forsythe, 2021).
Face embedding vectors are features extracted from face images with all the important information necessary to recognize a face. This makes FaceNet an excellent choice based on the reasons explained. A face embedding is similar to an eigenface, with the difference that it is much better at extracting essential facial features due to the large amount of training data it has been trained on. These face embeddings can then be fitted into classifiers as training data (Le, 2019). The classifier that will be used is the K Nearest Neighbour classifier from our course. To summarise, the FaceNet model will only be used to transform the training and test data into face embeddings, which will then be fitted and tested by the KNN classifier. No retraining of the neural network is needed

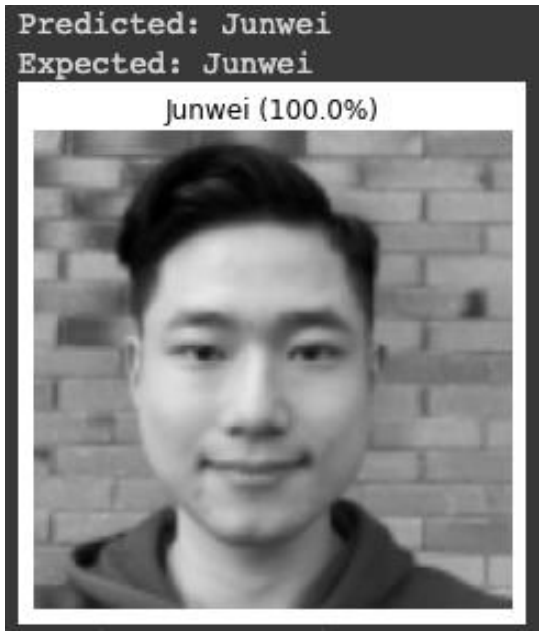
The outline of our approach is as follows (Le, 2019):

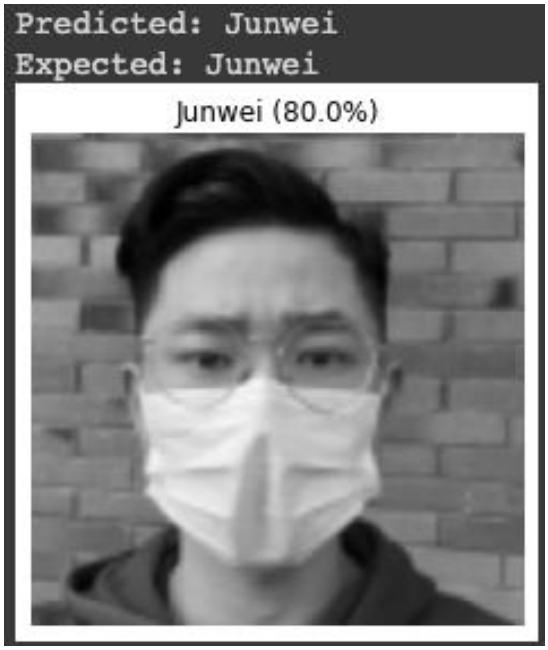
1. Import the pretrained Inception model FaceNet, which helps to transform the face images into embedding vectors.
2. Extract the embeddings vectors of the training face matrix using the pre-trained FaceNet. Encode the train and test labels into numerical values and fit the embedding vectors along with the encoded names into a K Nearest Neighbor classifier.
3. Transform the test data into embedding vectors, and use the trained K Nearest Neighbor to find the nearest embedding vectors from the test data.

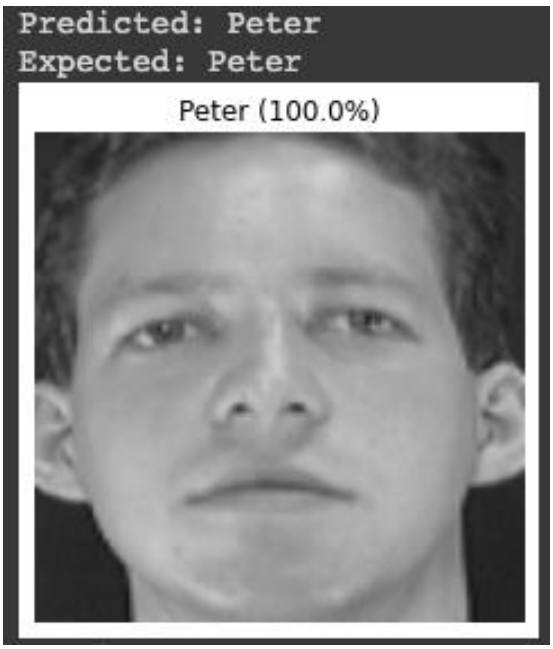
The K Nearest Neighbor algorithm is outlined as follows (Wang, H., 2022):

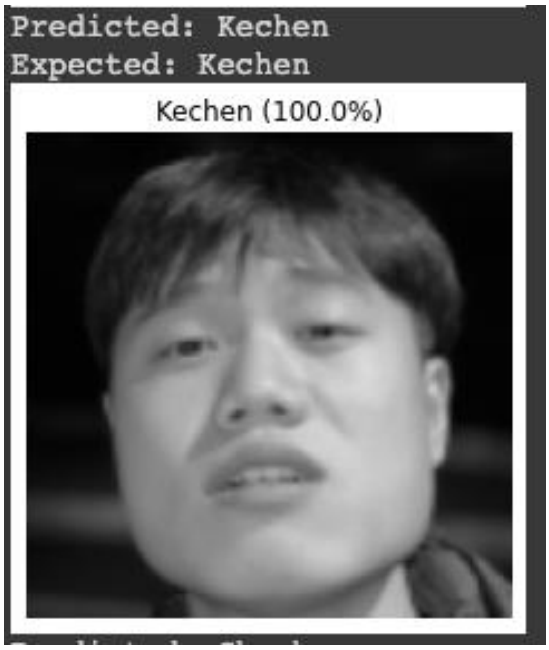
1. Given a measurement x , find the k nearest samples to x.
2. From these k samples, check how many of them belong to each class.
3. The measurement p will be assigned to the class with the most samples in the k nearest samples.

Hence, the images are converted transformed into face embedding vectors and then fitted into our K Nearest Neighbor classifier. From the scores obtained using the FaceNet + KNN method, we see that this method has better performance than with just the Deep Learning approach. This method achieves train accuracy of 99.53% with test accuracy 100.00%. So far, this is the best accuracy obtained for both training and testing data out of all the three methods. In each image shown below, the probability confidence from the KNN displayed alongside it.




Predicted: Junwei
Expected: Junwei
Junwei (80.0%)


Predicted: Peter
Expected: Peter
Peter (100.0%)

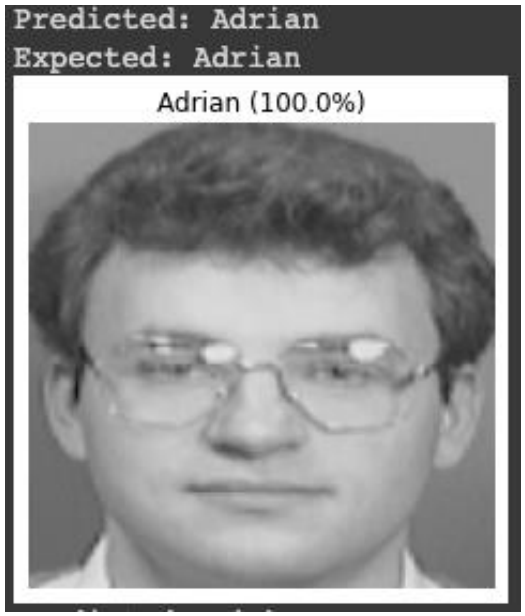

Predicted: Kechen
Expected: Kechen
Kechen (100.0%)


Predicted: Chenhong
Expected: Chenhong
Chenhong (60.0%)

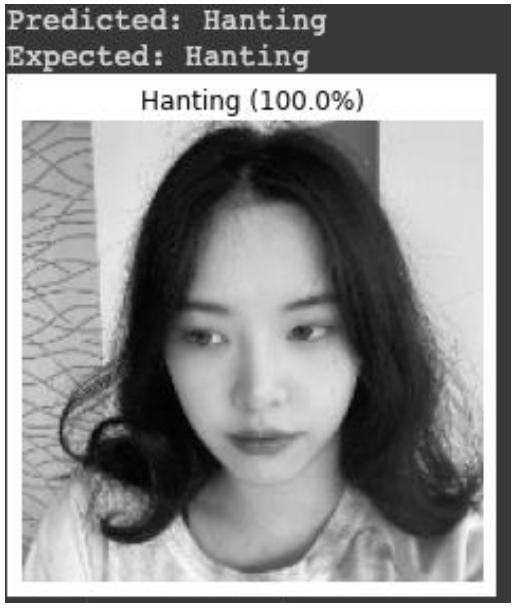

Predicted: Ramli
Expected: Ramli
Ramli (100.0%)


Predicted: Chenhong
Expected: Chenhong
Chenhong (40.0%)


Predicted: Yichang
Expected: Yichang
Yichang (100.0%)


Predicted: Adrian
Expected: Adrian
Adrian (100.0%)


Predicted: Yichang
Expected: Yichang
Yichang (100.0%)


Predicted: Hanting
Expected: Hanting
Hanting (100.0%)


Predicted: Hanting
Expected: Hanting
Hanting (80.0%)


Predicted: Andrea
Expected: Andrea
Andrea (100.0%)


Predicted: Maxus
Expected: Maxus
Maxus (100.0%)

6. Conclusion

In this report, our group has carried out a comparison between three different types of methods for face recognition which are Eigenface + NCC, Convolutional Neural Network and FaceNet + KNN. Based on the three approaches carried out, we see that the Eigenface + NCC is the most effective when we have low computational power and training data because requires most of the training images have been compressed into lower dimensions. However, the Eigenface + NCC approach loses accuracy, especially when the images have different lighting. For the Convolutional Neural Network, we see that it is quite effective for most cases but requires a larger sample size. The most effective method is the FaceNet + K-Nearest Neighbour classifier method with a test accuracy of 100%. Besides, no training is needed since the FaceNet is just used to transform the face images into face embedding vectors. However, one downside the large amount of storage space needed by the FaceNet (92.3Mb).

References

Cheplyaka, R. (2017, December 11). *Explained variance in PCA*. Retrieved March 6, 2022,

from https://ro-che.info/articles/2017-12-11-pca-explained-variance

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

Forsythe, M. M. (2021, September 21). Generate image embeddings using a pre-trained CNN and store them in Hub. Activeloop. Retrieved February 28, 2022, from https://www.activeloop.ai/resources/1E3oKP83o4KXIFChCesv7r/generate-image-embeddings-using-a-pre-trained-cnn-and-store-them-in-hub/

Image filtering using convolution in opencv. LearnOpenCV. (2021, July 15). Retrieved February 28, 2022, from https://learnopencv.com/image-filtering-using-convolution-in-opencv/

Le, K. D. (2019). A Study of Face Embedding in Face Recognition.

Lee, W. M. (2022, January 31). Using principal component analysis (PCA) for Machine Learning. Medium. Retrieved February 28, 2022, from https://towardsdatascience.com/using-principal-component-analysis-pca-for-machine-learning-b6e803f5bf1e

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of big data, 2(1), 1-21.

Seguin, J. (2017, October 30). Overfitting and regularization. IRIC's Bioinformatics Platform. Retrieved February 28, 2022, from https://bioinfo.iric.ca/overfitting-and-regularization/

Shanthamallu, U. S., & Spanias, A. (2021). Machine and Deep Learning Algorithms and Applications. Synthesis Lectures on Signal Processing, 12(3), 1-123.

Turk, M. A., & Pentland, A. P. (1991, January). Face recognition using eigenfaces. In Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition (pp. 586-587). IEEE Computer Society.

Valueva, M. V., Nagornov, N. N., Lyakhov, P. A., Valuev, G. V., & Chervyakov, N. I. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. Mathematics and Computers in Simulation, 177, 232-243.

Wang, H. (2022, March 4). Face Recognition (G0191 Chapter 1) [PowerPoint Slides]. Xiamen University Malaysia.

Wang, H. (2022, March 4). Real-Time Face Detection (G0191 Chapter 2) [PowerPoint Slides]. Xiamen University Malaysia.

Wang, H. (2022, March 4). Classifiers (G0191 Chapter 3 & 4) [PowerPoint Slides]. Xiamen University Malaysia.

Wall, M. E., Rechtsteiner, A., & Rocha, L. M. (2003). Singular value decomposition and principal component analysis. In A practical approach to microarray data analysis (pp. 91-109). Springer, Boston, MA.