PROJECT REPORT

COMPARISON BETWEEN DEEP NEURAL NETWORK AND PRINCIPAL COMPONENT ANALYSIS ALGORITHM IN FACE RECOGNITION

NADYA ANGELLA
18.K1.0008

Faculty of Computer Science
Soegijapranata Catholic University
2022

HALAMAN PENGESAHAN

Judul Tugas Akhir:	:	Comparison between Deep Neural Network and Principal Component Analysis Algorithm in Face Recognition
Diajukan oleh	:	Nadya Angella
NIM	:	18.K1.0008
Tanggal disetujui	:	06 Januari 2022
Telah setujui oleh		
Pembimbing	:	R. Setiawan Aji Nugroho S.T., MCompIT., Ph.D
Penguji 1	:	R. Setiawan Aji Nugroho S.T., MCompIT., Ph.D
Penguji 2	:	Hironimus Leong S.Kom., M.Kom.
Penguji 3	:	Yonathan Purbo Santosa S.Kom., M.Sc
Penguji 4	:	Rosita Herawati S.T., M.I.T.
Penguji 5	:	Yulianto Tejo Putranto S.T., M.T.
Penguji 6	:	Y.b. Dwi Setianto S.T., M.Cs.
Ketua Program Studi	:	Rosita Herawati S.T., M.I.T.
Dekan	:	Dr. Bernardinus Harnadi S.T., M.T.

Halaman ini merupakan halaman yang sah dan dapat diverifikasi melalui alamat di bawah ini.

sintak.unika.ac.id/skripsi/verifikasi/?id=18.K1.0008

DECLARATION OF AUTHORSHIP

I, the undersigned:

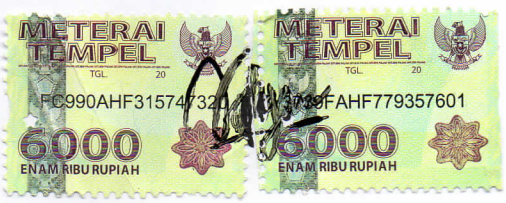
Name : NADYA ANGELLA

ID : 18.K1.0008

declare that this work, titled "COMPARISON BETWEEN DEEP NEURAL NETWORK AND PRINCIPAL COMPONENT ANALYSIS ALGORITHM IN FACE RECOGNITION", and the work presented in it is my own. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at Soegijapranata Catholic University
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
3. Where I have consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the work of others, the source is always given.
5. Except for such quotations, this work is entirely my own work.
6. I have acknowledged all main sources of help.
7. Where the work is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Semarang, January, 06, 2022



NADYA ANGELLA

18.K1.0008

STATEMENT PAGE OF SCIENTIFIC WORK PUBLICATION FOR ACADEMIC PURPOSE

I, the undersigned:

Name : NADYA ANGELLA

Study Program : Informatics Technology

Faculty : Computer Science

ID : 18.K1.0008

Type of Work : Thesis

approve to grant the Soegijapranata Semarang Catholic University, Non-exclusive Royalty-Free Rights for scientific work entitled "COMPARISON BETWEEN DEEP NEURAL NETWORK AND PRINCIPAL COMPONENT ANALYSIS ALGORITHM IN FACE RECOGNITION" along with existing tools (if needed). With these rights, Soegijapranata Catholic University has the right to save, transfer media/format, manage in database form, maintain, and publish this final project as long as it includes my name as the writer/creator and as the copyright owner.

This statement was made truthfully.

Semarang, January, 06, 2022

NADYA ANGELLA

18.K1.0008

ACKNOWLEDGMENT

I have received tons of support, encouragement, and assistance throughout this document writing. I want to thank my supervisor, Mr. Robertus Setiawan Aji Nugroho, Ph.D., for helping me formulate this topic and giving many knowledge and ideas. This project would not have been possible without his guidance and help. I would also like to thank all my friends for guiding me with advice to finish this document and for patiently answering my endless questions. Your unwavering help and consolation were priceless.

I would also like to thank my family for giving me everlasting love, support, and advice throughout my study. You gave me a great escape to rest my mind from my thesis.

Semarang, January, 06, 2022

NADYA ANGELLA

18.K1.0008

ABSTRACT

Face recognition is one technology that is commonly used now. Even our mobile phones use this technology as a security lock. Therefore, various algorithms continue to be developed to obtain maximum results with minimum costs. One of them is the Deep Neural Network or DNN algorithm. This DNN algorithm is part of the machine learning field. While DNN requires a large dataset to train the algorithm, another algorithm called the Principal Component Analysis (PCA) algorithm works good in a smaller dataset. These algorithms are compared to know which algorithm has the better result in given circumstances. Later the accuracy, speed, and optimality of the algorithms are analyzed. This project also examines the most preferable and optimum algorithms within the cases.

By comparing these algorithms, we could know which algorithm is preferable based on your needs. First, fetch the Olivetti faces dataset with the help of the Sklearn dataset library and split the dataset into two parts; training set and testing set. Then, the DNN algorithm is trained using the training set. After that, the model trained is tested with the testing set. The same step is also done for the PCA algorithm. After the result is obtained, we can conclude which algorithm is better within the given condition.

After the experiment is done, we can assume that the two algorithms have a s light difference in terms of accuracy. Also, the time used for running the PCA implementation code is slightly longer than DNN. However, that does not mean that the PCA algorithm is not great. If the dataset to be used is limited, PCA is going to be a good choice.

Keyword: Deep Neural Network, Principal Component Analysis, Face Recognition

TABLE OF CONTENTS

LIST OF FIGURE

Table of Figures

LIST OF TABLE

Index of Tables

CHAPTER 1
INTRODUCTION

1.1. Background

Recognizing faces sounds like a simple thing to humans. We can easily recognize someone in person or maybe through pictures or videos. Nevertheless, it is not that easy for computer vision to do it.

Back in the day, we could only see the use of face recognition technology on television or maybe in movies, but now facial recognition technology is commonly used in various fields. Our smartphone is one such example, and almost every phone has the face unlock feature nowadays. Because of that, so many algorithms are developed in order to find the most optimum in terms of time, speed, and costs. One of them is Neural Networks. There has been a surge of interest in neural networks, particularly deep and large networks. These networks have exhibited impressive results [1].

However, besides the significant advantages, beginner researchers have one problem: The approach is computationally expensive and requires a high degree of correlation between the test and training images [2]. What should we do to overcome this weakness?

There is another approach to recognizing faces by using a statistical approach or trying to search for patterns. One of the most popular algorithm was Principal Component Algorithm (PCA). This algorithm is also called Eigenfaces when implemented on the images. With more increase in the size of the training set, the algorithm shows increased accuracy [3]. This might be a solution for beginners who want to implement face recognition and do not have adequate resources.

That is why this project specifically compares those two algorithms. Will there be significant results in time, speed, accuracy, or even if these two algorithms compete with the results. Which one is more effective with the environment given? Should we still use a neural network with relatively large resources but high accuracy? Or is it better to use the PCA algorithm as an alternative, which has fewer resources?

1.2. Problem Formulation

1. How is the performance of the DNN and PCA algorithms in terms of accuracy and speed?
2. Which algorithm is more optimal with the given circumstances?
3. Could the PCA algorithm outperform the DNN algorithm in a particular condition?

1.3. Scope

Two algorithms, namely DNN and PCA, are used to perform facial recognition. The DNN algorithm is implemented with the help of the Keras library, while the PCA algorithm is done from scratch. As the first step, load the dataset. For the DNN algorithm coding, the dataset is loaded from Scikit Learn or often called Sklearn. Later the Sklearn is used as a loader for the Olivetti faces dataset (ORL). This dataset contains pictures of 40 people with some variances in lighting, expressions, and accessories. Then, after the dataset preprocessing is complete, the face recognition process begins. This process includes the training and test phase. The DNN and PCA algorithm are implemented separately, and a k-fold cross-validation technique is used to split the datasets for the training and test phase. Finally, we compare the result in terms of time, speed, and ease of implementation.

1.4. Objective

This project aims to test the DNN and PCA algorithm in face recognition and compare each algorithm's accuracy in given circumstances. Both of the algorithms are trained and tested with the same but randomized image datasets from ORL.

CHAPTER 2
LITERATURE STUDY

Face recognition is one of the most helpful experiences in our lives. There is much use of face recognition implementation, such as security, attendance tracking, digital entertainment, and more. Because of its benefits, several face recognition and modeling systems have been developed and deployed. However, accuracy and performance still be a challenge to researchers [4].

Sun et al. [5] declared that very deep neural networks recently achieved significant success on general object recognition because of their superb learning capacity, which motivates them to examine their effectiveness on face recognition. In this paper, the authors propose two deep neural network architectures, called DeepID3, which are significantly more profound than the previous state-of-the-art DeepID2+ architecture for face recognition. DeepID3 networks are rebuilt from basic elements of VGG net and GoogLeNet. During training, joint face identification-verification supervisory signals are added to the final feature extraction layer and a few intermediate layers of each network. The dataset is taken from the LFW face dataset and used for both training and testing. Being trained on the same dataset as DeepID2+, DeepID3 increases the face verification accuracy up to 0.06% and rank-1 face identification accuracy from 95.0% to 96.0% on LFW, compared with DeepID2+. There are three test face pairs labeled as the same person but are different on the LFW website. The DeepID3 algorithm classified two of them as the same person among these pairs while the other was classified as a different person. Therefore, when the label of these three pairs is corrected, the accuracy of DeepID3 has risen to 99.52%. In summary, the proposed DeepID3 networks achieve state-of-the-art performance for both LFW face verification and task identification. The effectiveness of very deep neural networks will be further investigated on larger scale training data in the future. While this paper uses DeepID3, the project implements DNN using the help of the Keras library. Also, this paper compares DeepID3 and DeepID2+ meanwhile the project compares DNN and PCA.

Implementing Neural Networks is sometimes complex and time-consuming. Convolutional Neural Network (CovNets) is one of them. That is why Priya et al. [6] propose a new way of using neural networks. This approach does not provide raw pixel as input, but only the extracted facial features. The approach proposes using the frontal face in haar cascade (defined in OpenCV) to preprocess the input images and feed only the facial features

to the network for learning and classification. After the preprocessing, a multi-layer feed-forward deep neural network is applied to learn the simplified features from the previous step. Multiple sets of activation, dense (fully-connected), and dropout layers are added to make learning efficient. After each set, the number of features reduces, thus using only the essential features for classification in the last layer. The last layer of the network is softmax which is used to classify. The proposed method is trained and tested on the Yale face database, consisting of grayscale images of 15 samples. Each of these samples has 11 images in different expressions and conditions. From the experiments, the author concludes that the use of haar cascade to extract facial features and feeding them instead of raw pixel values helps decrease the complexity of the neural network-based recognition framework as the number of redundant input features has been decreased. Also, using DNN instead of CovNets makes the process lighter and faster. The accuracy is not compromised in the proposed method as the average accuracy obtained is 97.05%. Though one additional step of extracting facial features from each image is added, the process is still better for small datasets. The algorithm compared is the DNN and CovNets in this paper. Meanwhile, the project will compare DNN and PCA algorithms. The dataset used in the paper is the Yale face dataset, and the project uses the ORL dataset.

Wang et al. [7] stated that deep neural networks have significantly improved face recognition and facial attribute prediction performance. Yet still a big challenge for millions datasets, i.e., MegaFace. The authors advocate a multi-task deep neural network for jointly learning face recognition and facial attribute prediction tasks in this paper. The whole architecture proposed has two steps: facial attribute prediction and face recognition network step. The attribute classifier for facial attribute prediction is the Mixed Objective Optimization Network (MOON). In the face verification task, the output of the network is used as a representation to extract the features of each image and given a pair of images, calculate their cosine distance, and set the threshold to judge whether this pair is the same person or not. The network thus has four types of layers: convolutional layers, max-pooling layers, building blocks layers, and fully connected layers. The models are trained on CASIA-WebFace and CelebA datasets. Meanwhile, the MegaFace dataset is used to test the models. LFW dataset was also used to evaluate the key components of the network. From this paper, it can be concluded that experimental results clearly show the benefit of jointly learning structures. Such learning helps to capture both global functionality and local attribute information at the same time. The layer of the algorithm implemented in this paper consists of a convolutional

layer, max-pooling layer, building blocks layers, and fully connected layers. Meanwhile, the project uses ReLU (rectified linear units) and softmax as the DNN algorithm model.

Wu and Deng [8] have shown that pose and illumination are considered two main challenges facing recognition system encounters. This paper considers face recognition problems across pose and illumination variations, given a small number of training samples and a single sample per gallery (a.k.a., one-shot classification). Based on these problems, the authors attempt to combine the strength of 3D models in generating multiview and various illumination samples. Deep learning features that learn non-linear transformations is very suitable for pose and illumination normalization. The DNN architecture consists of two tasks, namely the normalization task and reconstruction task. In the normalization task, the features of each image are extracted from the pooling layer. Then the test image is compared to gallery images by square euclidean distance and gets recognition results. This experiment uses the MultiPIE database both for training and testing. The algorithm's overall performance in face recognition is 74.4%. Finally, this augmentation idea can be applied to train a deep neural network, but training samples are hard to acquire. Experiments on the MultiPIE database achieve competitive results with much less training data than other methods, verifying the effectiveness of the proposed method. This paper focuses on one algorithm while the project compares two algorithms to acknowledge each algorithm's advantages and disadvantages.

Face recognition is a crucial research topic in computer vision because of its many possible uses. That is why Zhang et al. [9] investigated a face recognition method based on a deep neural network. The face recognition method is based on the deep learning framework and the top classification algorithm. The framework structure of the algorithm consists of two parts. The first part is a deep learning network stacked by multi-layers of self-learning and can abstract characteristic data layer by layer. The second part is the classifier, which outputs the classification results by using the softmax algorithm. The softmax regression model generalizes logistic regression to classification problems, where the class label can take more than two possible values. In this paper, the sparse autoencoder of the neural network is used as the framework of deep learning, and softmax regression is used as the top classification algorithm to classify and identify data features. The results show that using deep learning can better extract the features of the human face, and overall fine-tuning of the depth of the network is better than the top of the fine-tuning. The method is evaluated on the ORL, Yale, Yale-B, and PERET face database to verify the recognition rate, respectively. The result of image pre-processing does not continuously improve the recognition rate. As for the

application of face recognition, the speed of face recognition based on deep learning is plodding, severely restricting the application of deep learning. At the same time, this paper focuses on one algorithm, the project comparing two different algorithms only with the ORL dataset.

Image classification includes image preprocessing, image segmentation, key feature extraction, and matching identification plays a vital role in computer vision. That is why Guo et al. [10] has an idea to build a simple convolutional neural network on image classification. In this project, the authors also analyzed different methods of learning rate sets and different optimization algorithms for solving the optimal parameters of the influence on image classification. This research aims to develop an algorithm to overcome the difficulties encountered in training deep neural networks. The Minist and Cifar-10 datasets are used in this study. Their research focuses on the design of convolutional and pooling layers. This research also studies the composition of the different Convolutional neural networks on the result of image classification. The CNN algorithm used in this experiment consists of a convolutional layer, pooling layer, and fully-connected layer. The authors use ReLU and dropout. In addition, unsupervised pre-training is needed. This simple convolutional neural network reduces a less computational costs. This research also verifies that shallow networks also have a relatively good recognition effect. While this paper used CNN, later the project implemented the DNN algorithm.

Siswanto et al. [11] are testing and developing face recognition as part of future multi-modal biometrics applications by taking the attendance System as its case study. Despite having a low accuracy compared to advanced biometrics, face recognition can be one of the most natural and "easy-to-collect" biometrics. This research compares PCA (Eigenface) and LDA (Fisherface) algorithms by analyzing the ROC (Receiver Operating Characteristic) curve. Those algorithms are chosen to overcome expensive computation and the need for large amounts of storage of older face recognition techniques such as correlation methods. The authors can conclude from the experiment that the PCA algorithm outclassed the LDA algorithm in the current training set. This paper analyzes the algorithm by its ROC curve, while the project examines the algorithm performance by the accuracy, speed, and implementation.

Li and Lin [12] propose a new face recognition method to solve the low accuracy of face recognition under non-restrictive conditions. This method is based on a gradient direction histogram (HOG) features extraction and fast principal component analysis (PCA). First, the

Haar feature classifier extracts the background interference data simultaneously in the original data preprocessing stage. Then PCA dimension reduction is implemented. Finally, the Support Vector Machines (SVM) work as a face recognizer. These algorithms are tested using the Labeled Faces in the Wild (LFW) face dataset and compared with the SVM algorithm, the PCA + SVM algorithm, and the HOG + SVM algorithm under the same experimental conditions. It turns out that the new method shows adequate results. Compared to the three previous algorithms, the new approach has higher accuracy with lower time. While this paper uses the LFW datasets, the project uses the ORL datasets.

The human face recognition system using the eigenface approach that is integrated with the Ry-UJI robot is proposed in this paper. The robot is recognized by a detected voice command looking for someone, and when a person's face has been found, face recognition is done. Ramadhani et al. [13] are using the eigenface method to recognize faces. Eigenface is one of the facial recognition methods based on the Principal Component Analysis (PCA) algorithm. The PCA included a mathematical process for deriving a set of features for face recognition. The face recognition stage begins with the face detection process using the cascade classifier method. After that, the face is preprocessed. The next step is to collect and train the face detected utilizing the author's private dataset, and finally, the face recognition. This research will focus on building a face recognition system integrated into the Ry-UJI robot and then measuring the accuracy. In this paper, the authors stated that PCA is one of the most popular multivariate statistical techniques and almost all scientific disciplines use it. Finally, the authors indicated that face recognition using the eigenface approach runs quickly and very well. They also conclude that using eigenvalues and eigenvectors in generating reconstructed faces produces excellent images and can be used to compare previously trained images. This paper concentrates on implementing face recognition with the PCA algorithm using the Ry-Uji robot as input data. In contrast, the project uses the ORL image dataset for training and testing.

Jadhav et al. [14] stated that face recognition had become one of the critical aspects of computer vision in modern times. That is why they propose an automated attendance management system. Simply, it is a computer application that automatically identifies people from a still images or video frames. This system will automatically recognizes when a student enters the classroom and mark the attendance by recognizing their faces. The algorithms used are Viola-Jones Algorithm face detection, which detects human face using cascade classifier and PCA algorithm for feature selection and SVM for classification. First, the camera is

mounted at a distance to capture the face images. Then, face detection is implemented using the Viola-Jones detection algorithm, which uses Integral Image and AdaBoost learning algorithm as the classifier. This algorithm gives better results in different lighting conditions and is combined with multiple haar classifiers to achieve better detection rates up to an angle of 30 degrees. After the face detection is done, the detected face is extracted and subjected to preprocessing. The preprocessing phase involves histogram equalization to the resized images. This step is necessary to improve the contrast of the image, as it expands the range of the intensity of the image by making it more precise. The last step is to insert the result into an excel sheet to record the attendance. The dataset used to train and test the system performance in this experiment is a private dataset. In this paper, the authors have proved that the proposed system is time-saving and secure. Also, the PCA algorithm outperforms other algorithms where there are unintentional changes in a person like accessories or beard. This paper proposes the PCA algorithm as a feature selection and SVM for the classification, different from the project that uses the PCA algorithm for feature selection and classification.

Khrisnan [15] proposed a system that uses a PCA algorithm to automatically detect student attendance, store the faces in the database, and retrieve absentee list to overcome the shortcomings of manual attendance. The system uses an eigenface approach to perform face recognition. This method analyzes and calculates eigenfaces, which are faces composed of eigenvectors. The method also compares the eigenfaces to identify the presence of a person and its identity. As a first step, the system needs to be initialized with a set of training faces. Then, when the face is detected, the eigenface is calculated. The system then compares the eigenvectors of the current face and the stored face image and determines whether the face is identified or not. Students must be in front of the camera at a distance of at least 60 cm. The system will detect the student's image, convert it into grayscale, and store it in an XML file. When the student reappears before the camera, faces are recognized by comparing the eigenfaces of current and stored images. Then the names of the detected faces are stored in Microsoft Access Database. The authors collected the dataset for testing and training by themselves. In conclusion, the proposed system could reduce the effort and manage the time effectively. The experiment results show improved performance compared to the traditional pen and paper type attendance system. This paper's goals are to successfully recognize a face and record it on the database, while the project goal is to know the performance of each algorithm.

Summing up from all of the study above, many neural networks algorithms are implemented for face recognition. Almost all of them achieved great accuracy. But still, the computational resource is a problem for most neural network algorithms. Also, the dataset size plays a significant role in the recognition accuracy of the algorithms. Meanwhile, the PCA algorithm can reduce the dataset dimensionality. But in the same condition, could the PCA algorithm reach higher accuracy?. This project is trying to acknowledge the performance of those two algorithms in terms of accuracy, speed, and implementation.

CHAPTER 3
DATASET AND ALGORITHM

3.1. Dataset

The dataset used in this project is the Olivetti faces dataset acquired by AT&T Laboratories Cambridge between April 1992 and April 1994. This dataset contains 40 classes and have different facial details such as the lighting, facial expressions, and accessories like glasses or no glasses. All the images were using a dark and homogeneous background, with the subject in an upright front position and able to withstand for some lateral movements. The image is quantized to 256 grey levels and stored as an unsigned 8-bit integer. The target of this database is an integer from 0 to 39, indicating the person's identity pictured.

The `sklearn.datasets` package helps the machine learning researchers to fetch large datasets that they use to benchmark algorithms. This module also includes some utilities such as load datasets, fetch reference datasets, and generates artificial data. There are three main functions of `sklearn`: general dataset API, dataset loaders, and dataset fetchers. This project uses `sklearn.datasets` as datasets that download and load larger datasets from the real world. Further detailed information about the dataset is explained later. The `sklearn.datasets` functions return an object called Bunch. The Bunch object is a dictionary that has key as an attribute.

Figure 3.1: SKLearn

As stated above, the dataset is managed as the training and testing data. Because of that, the dataset must be split into two parts. To do this process, `sklearn.model_selection.train_test_split` is used. The train-test split is a technique

for evaluating the performance of a machine learning algorithm. There are some parameters in this quick utility, such as `test_size`, `train_size`, `random_state`, `shuffle`, and `stratify`. The `test_size` parameter represents the proportion of the dataset to be set as test subsets, likewise, with the `train_size` parameter. These parameters should have some variation in the experiment to measure the algorithm's accuracy, and there is no absolute value. It all depends on the dataset and the project's objective found from experimental trials. The `random_state` parameter controls the shuffling applied to the data before splitting. This parameter intends to control the random number generation used so that consistency is ensured. Integer random seeds 42 is used to fill this parameter. The `shuffle` parameter is to determine whether or not to shuffle the data before splitting. The last parameter is `stratify`, which does a split. Our dataset has a various number of data for each class. It is advisable to splits the dataset into the train sets and test sets in the same proportions of data in each class. As shown in the Figure 3.2 below, the splitting proportion is equal for each class. In this way, the train and test sets contain all of the classes on the dataset.

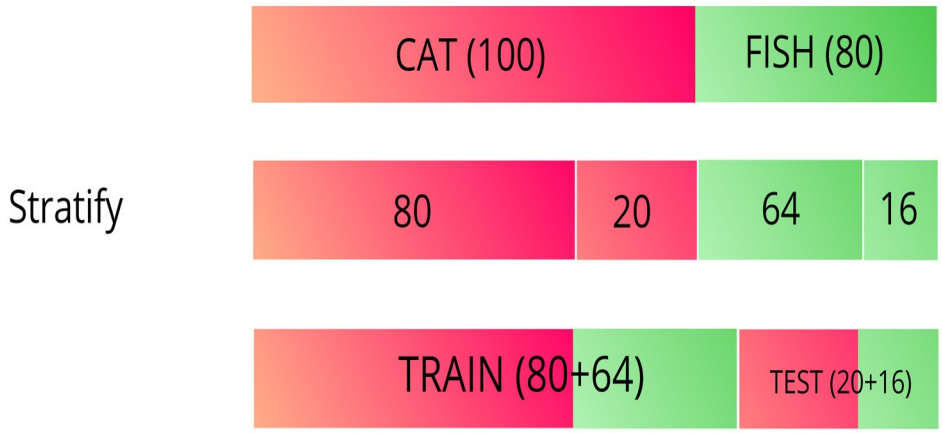


Figure 3.2: Stratify Illustration

Meanwhile, the ORL dataset used for the PCA algorithm is processed from the image data type. This data was obtained from the kaggle by Marlon Tavarez with 400 images that will be manually split into two parts: the training and the test sets. The images data are already labeled with the image id and the person id, separated by the underscore symbol (_).

For example, label 10_1 means the photo with id ten (10) belongs to person number 1. Later, these images are switched into the ndarray shape.

Figure 3.3: ORL Image 10_1.jpg

3.2. Deep Neural Network

Artificial Intelligence (AI) is a technology developed to 'think' like the works of the human brain. Inside AI, there are other subfields: machine learning, expert systems, and natural language processing, to name a few**.** However, machine learning is the most popular field at this cultural moment [16], and Deep Neural Networks (DNNs) are currently the state-of-the-art ML algorithms [17]. The DNN is a machine learning member with multiple layers between the input and output layers. Each layer of DNN uses the output from the previous layer as input which is very similar to how the human brain transmits information from one neuron to another. Figure 3.4 [18] shows the typical architecture of DNN consists of an input layer, some hidden layers, and the output layer.

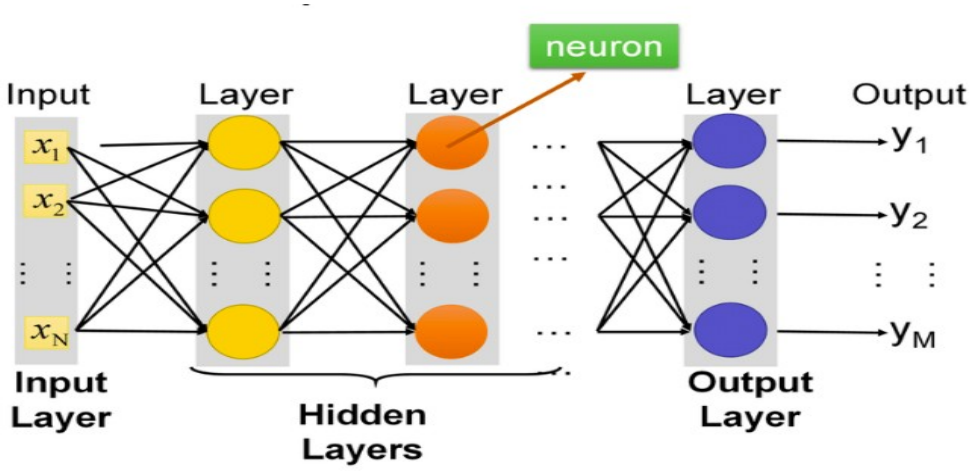


Figure 3.4: Typical Architecture of DNN

The model used in this project is a sequential model. The sequential model is suitable for a plain stack of layers with only one input tensor and one output tensor. This model allows us to build a model layer by layer. Each layer has weights that correspond to the layer that follows it. Those multiple layers are non-linear processing units, often called activation functions, used for feature extraction and transformation. There are some activation functions such as Rectified Linear Activation (ReLU), Logistic (Sigmoid), and Hyperbolic Tangent (Tanh). Meanwhile, this project uses the ReLU and softmax activation function. More details are explained below.

ReLU is an activation function that has a biological and mathematical base. It works by thresholding values at 0, i.e., as shown in Figure 3.5 ReLU activation function [19], when the output value is less than 0, convert the value to 0. Conversely, it outputs a linear function when the output value is more than 0, unlike the tanh and sigmoid activation functions, which

approximate a zero output, e.g., a value very close to zero but not an actual zero value. The ReLU also did not require exponential calculation, so the computations are cheaper [20]. Emphasized by Krizhevsky et al. as cited in Wu and Deng, 2016, the ReLU function's advantages include faster training speed, decreased saturation problems, a smaller number of epochs, and usually fewer samples.

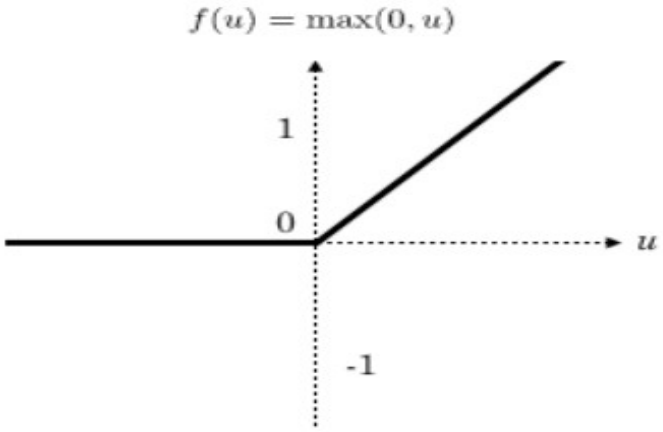


Figure 3.5: ReLU Activation Function

The softmax function is used to calculates the probability of each target class from all probable target classes. It is used as the activation function of the output layer of a neural network model that predicts a polynomial probability distribution. The probability is used to determine the target class for the specified inputs. These probability values are ranged from 0 to 1, and if we sum all of the probabilities, it will equal to 1. Later, the calculated probabilities will be used for determining the target class of the inputs. If we use the softmax function for the multi-classification model, it will returns the probabilities of each class. Here in Figure 3.6 [21] show a graphical representation of the softmax activation function.

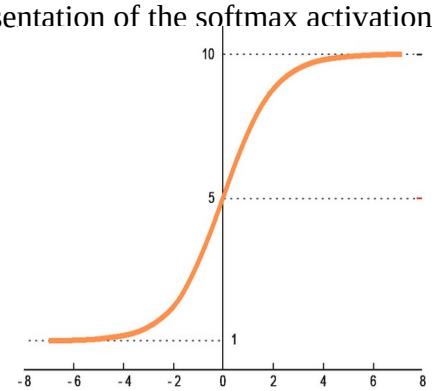


Figure 3.6: Graphic Representation of Softmax Function

Figure 3.7 below shows the softmax function formula that calculates the ratio of the exponential of the input value and the sum of the exponential values.

$$\sigma(\vec{z})i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 3.7: Softmax Formula

For example, we have an array with three values $\begin{bmatrix}2\\9\\7\end{bmatrix}$. These values could be the output of the machine learning model, but with the softmax function, we convert those values into a probability distribution.

$e^{z1} = e^{2} = 7.39$

$e^{z2} = e^{9} = 8130.08$

$e^{z3} = e^{7} = 1096.63$

Then, sum all three exponentials to obtain the normalization term.

$\sum_{j=1}^{K} e^{zj} = e^{z1} + e^{z2} + e^{z3} = 7.39 + 8130.08 + 1096.63 = 9234.1$

From the value, we can see that z2 dominates the normalization term. The next step is to divide each of the array values to z2.

$\sigma(\vec{z})_1 = \frac{7.39}{9234.1} = 0.0008$

$\sigma(\vec{z})_2 = \frac{8130.08}{9234.1} = 0.8804$

$\sigma(\vec{z})_1 = \frac{1096.63}{9234.1} = 0.1188$

Three output values lie between 0 and 1, and also they sum to 1.

3.3. PCA

In 1991, *Turk and Pentland s*uggested an approach to face recognition that uses dimensionality reduction and linear algebra concepts. This approach is commonly used as it is computationally less expensive and easy to implement. Principal component analysis (PCA) is a technique used to reduce the dimensions of a dataset. This method works by transforming a large set of variables into smaller ones containing most of the information while minimizing information loss. Smaller datasets make data analyzing easier and faster for machine learning. PCA is one of the oldest and most widely used algorithms. The PCA can be divided into five main steps.

1. Normalize the dataset

   Subtract the mean of each variable from the dataset to normalizing them. This step means that we are removing the "common features" we got from the mean of the dataset.

2. Compute the covariance matrix

   To know if there is a correlation in the input dataset's variables, compute the covariance matrix of normalized data. It's the value of covariance that matters. If the value is positive, then the two variables are correlated. If negative, then it is uncorrelated

3. Compute the eigenvectors and eigenvalues from the covariance matrix

   From the covariance matrix, compute the eigenvectors and eigenvalues. Eigenvector is a nonzero vector that does not change direction when a transformation is applied. Meanwhile, the eigenvalue is a scalar associated with eigenvectors. Let us assume A is an "n × n" matrix, λ is an eigenvalue of matrix A, and x (a nonzero vector) is called an eigenvector if it satisfies $Ax = \lambda x$ expression.

4. Sort the eigenvalues in descending order and select a subset from the rearranged Eigenvalues matrix

Each column in the eigenvector matrix corresponds to a principal component. The principal components are new variables constructed as a linear combination that uncorrelated, and most of the information within the initial variables is squeezed or compressed. Arranging the eigenvalues in descending order of their eigenvalue will also set the principal component by their variability. Then, select a subset or from the arranged eigenvalue that captures the highest variability.

5. Transform the data

Compute a dot product between the transpose of the eigenvector subset and the transpose of the normalized data. The outcome of this step is data that is reduced to lower dimensions.

CHAPTER 4
ANALYSIS AND DESIGN

4.1. Analysis

One of this project's goals is to find the performance of the DNN and PCA algorithms in terms of accuracy and speed. The analysis process involves the following steps :

1. Fetch the ORL faces dataset using sklearn and split it into training and test sets
2. Create the DNN model; this model uses ReLU and softmax as its activation functions
3. Train the dataset using the model that has been compiled
4. Calculate and analyze the accuracy of the DNN algorithm with the given circumstance
5. The dataset used for the PCA algorithm is in the image format, so we need to convert them into the ndarray data type
6. Calculate the mean face of the dataset. The mean face is the average feature of the datasets
7. After that, find the normalized faces by subtracting each face in the dataset by the mean face. This normalized face is an extracted or unique feature of each face
8. Calculate the eigenvector using the covariance matrix of the normalized faces matrix
9. Select best eigenvector as much as K, where K is less than the total training images and can represent the whole training set
10. Convert lower-dimensional K eigenvectors to the original face dimensionality
11. Calculate weight vector for each face
12. Calculate the distance between the weight of each input vector and all the weight vectors of the training set
13. Calculate and analyze the accuracy of the PCA algorithm under the given circumstance
14. Compare the accuracy between the DNN and the PCA algorithms

4.2. Design

Python is used as the primary computing language for this project. The reasons for choosing Python are because of its one of the most accessible programming languages available. It has simplified syntax, which gives more emphasis on natural language. Due to its ease and various tools, Python codes can be easily written and executed faster than the other programming languages.

The code of this project is run in the Google Colaboratory or Google Colab. Google Colab is a hosted Jupyter notebook that does not requires setup. The reason of using Google Colab is because it is an excellent tool that provides free access to Google computing resources such as Tensor Processing Unit or TPUs and Graphical Processing Unit or abbreviated as GPUs.

The dataset used for the PCA algorithm is stored in Google Drive, while the dataset used for the DNN algorithm is directly fetched using the sklearn library.

As the first step, we're going to implement the DNN algorithm using the Keras library. Keras is one example of a deep learning API written in Python, running on TensorFlow's machine learning platform. It was developed with a focus on enabling fast experimentation. The core data structures of Keras are layers and models. The simplest type of the model is a linear stack of layers called the Sequential Model.

Before starting the learning process, we need to fetch the dataset using sklearn datasets and split it into a train and test set using the sklearn model selection. Preprocess the train and test set with the help of sklearn preprocessing, which standardizes a dataset along any axis. The next step of the process is to create the model. The model used is a sequential model consisting of 9 layers with softmax and ReLU as the activation functions. Compile the model with the RMSprop optimizer. RMSprop is a gradient-based optimization technique used in neural networks training. Gradients in a very complicated functions, such as neural networks tend to explode or may disappear as the data is propagating through a function. RMSprop addresses this issue by normalizing the gradient using a moving average of the quadratic gradient. After the model is compiled, start the training and evaluate the loss and accuracy of the model using the test data.

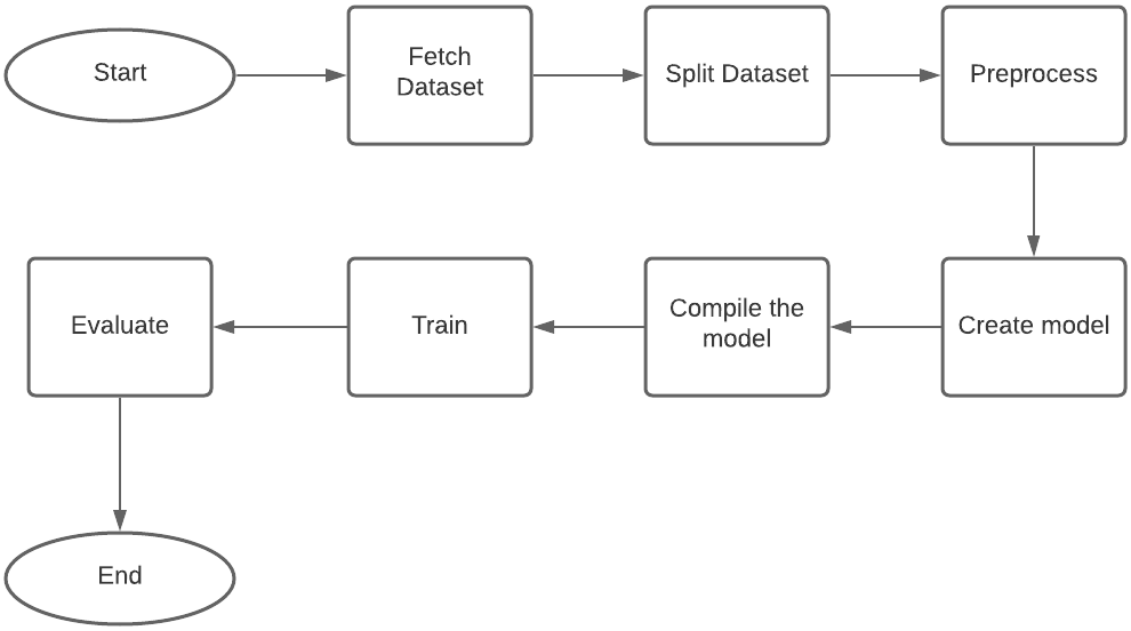


Figure 4.1: DNN Implementation Flowchart

Next, implement the PCA algorithm. The image dataset is stored on Google Drive, so we need to integrate the Google Colab by mounting the Google Drive. Convert all of the image datasets into ndarray using the NumPy library. NumPy is a Python library used to manipulate arrays. We need to find the mean face of the dataset to remove all the dataset's common features. By subtracting each face image from the mean face, we get each image's unique feature, called normalized images. PCA is done by decompose the covariance matrix, so compute the covariance of the normalized images and find the eigenfaces. Then, find a K number of significant eigenface that can represent the whole training set. They must not leave out any important information about the data that we have. Then, calculate the weight of each eigenface. These weights are the proportions of each eigenface to make up each person's face in the dataset. Then we analyze the accuracy by giving the algorithm a test set.

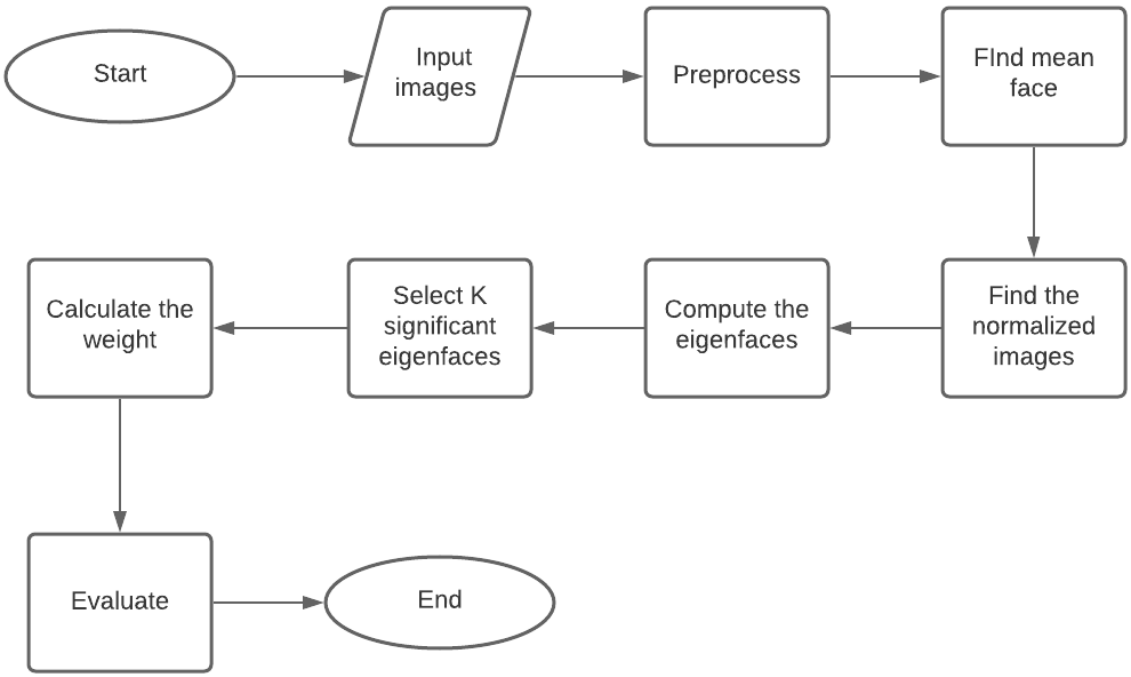


Figure 4.2: PCA Implementation Flowchart

Now, after we implement the DNN and PCA algorithm, compare and analyze these two algorithms in terms of time and accuracy in given circumstances.

CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

For the DNN algorithm, we utilize the sklearn.datasets library to fetch the ORL dataset. Line number 1 is used to import the Olivetti faces from the sklearn.dataset library. The parameter of return_X_y is set to true, which returns (data, target) instead of a Bunch object. The following line stores the data fetched to the variables X and y.

```
1  from sklearn.datasets import fetch_olivetti_faces
2  X,y = fetch_olivetti_faces(return_X_y=True)
```

The next step is to split the dataset into training and test sets. Here we give the parameter X and y, which contains the data and target. The test_size is going to be varied. The first proportion is 0.1, then 0.2, 0.3, and lastly, 0.4. Set the stratify and random state to improve the precision of the sample and control the shuffling before applying the split.

```
3  train_X,  test_X,  train_y,  test_y  =  train_test_split(X,  y,
   test_size=0.20, stratify=y, random_state=42)
```

Scale the data simply by passing the train_X and test_X variable to preprocessing.scale command.

```
4  train_X = preprocessing.scale(train_X)
5  test_X = preprocessing.scale(test_X)
```

Then, create the model architecture. The model used is the sequential model with softmax and ReLU activation functions. Here we add the layers one by one. First, we add the dense layer. The dense layer is the regular deeply connected neural network layer most commonly used for neural networks. The Dense layer supplies all the outputs from the previous layer to all its neurons, and each neuron provides the output to the next layer.

The unit parameter of the dense means the dimensionality of the output space is 200. The input dimension of the first dense layer is 4096, which is the total number of pixels from the dataset, a face image with a size of 64 x 64 pixels. Kernel regularizer is a function applied to the kernel of the weights matrix. Input parameter l2 means that we use the L2 regularization penalty. The L2 regularization penalty is computed as: loss = l2 * reduce_sum(square(x)).

The next layer is the dropout layer. The Dropout layer is randomly sets input units to 0 at each step during training time. After the model is done, it is compiled with the rmsprop

optimizer. For the loss function, this project uses sparse categorical crossentropy. It is the default loss function for multi-class classification problems where each class is assigned a unique integer value from 0 to (num_classes – *1*). The last parameter of this compile command is metrics. These metrics contain a list of metrics to be evaluated by the model during training and testing.

```
6  model = Sequential([
7  Dense(units=200,     input_dim=4096,     kernel_regularizer=l2(0.0001),
   activation='relu'),
8  Dropout(0.2),
9  Dense(units=200,     input_dim=200,     kernel_regularizer=l2(0.0001),
   activation='relu'),
10 Dropout(0.2),
11 Dense(units=200,     input_dim=200,     kernel_regularizer=l2(0.0001),
   activation='relu'),
12 Dropout(0.1),
13 Dense(units=200,     input_dim=200,     kernel_regularizer=l2(0.0001),
   activation='relu'),
14 Dropout(0.1),
15 Dense(units=40, input_dim=200, activation='softmax'),])
16 model.compile(loss='sparse_categorical_crossentropy',
17 optimizer='rmsprop',
18 metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 200)               819400

 dropout (Dropout)           (None, 200)               0

 dense_1 (Dense)             (None, 200)               40200

 dropout_1 (Dropout)         (None, 200)               0

 dense_2 (Dense)             (None, 200)               40200

 dropout_2 (Dropout)         (None, 200)               0

 dense_3 (Dense)             (None, 200)               40200

 dropout_3 (Dropout)         (None, 200)               0

 dense_4 (Dense)             (None, 40)                8040

=================================================================
Total params: 948,040
Trainable params: 948,040
Non-trainable params: 0
```

Figure 5.1: DNN Model Summary

After that, train the model and count the time consumed to run this code. This training step will be repeated by changing the dataset split ratio parameters. First, apply the 90:10 ratio by passing 0.1 to the validation split parameter. In the same way, use the 80:20, 70:30, and 60:40 ratios. Evaluate the model for each training.

```
19 h = model.fit(train_X, train_y, batch_size=50, epochs=num_epochs,
   validation_split = 0.2, verbose=0)
20 finish = time.perf_counter()
21 print(f"Training finished in {finish - start:0.4f} seconds\n")
22 print("Training history: ")
23 for i in range(num_epochs):
24 los = h.history['loss'][i]
25 acc = h.history['accuracy'][i] * 100
26 print("epoch: %5d loss = %0.4f acc = %0.2f%%" \
27 % (i, los, acc))
28
29 eval = model.evaluate(test_X, test_y, verbose=0)
30 print("\nEvaluation on test data: \nloss = %0.4f \
31 accuracy = %0.2f%%" % (eval[0], eval[1]*100) )
```

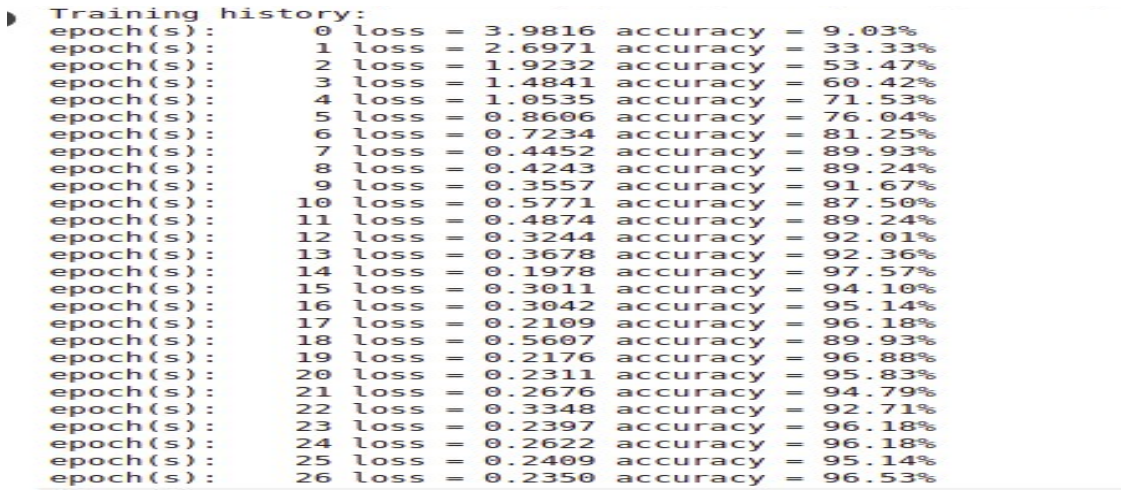


Figure 5.2: Training History

Now after the DNN algorithm is implemented, it is time to implement the PCA algorithm. The first thing to do is to split the dataset into the training and testing set. After that, find the mean face of the training set.

```
1  mean_face = np.zeros((1,height*width))
2  print(mean_face)
3
4  for i in training:
5  mean_face = np.add(mean_face,i)
6
7  mean_face = np.divide(mean_face,len(train_images)).flatten()
```

Then, find the normalized matrix by subtracting each face of the training set from the mean face. This normalized image contains only the unique features of the faces.

```
8  for i in range(len(train_images)):
9  normalised_training[i] = np.subtract(training[i],mean_face)
```

The next step is to find the eigenvalues and eigenvectors. To find these values, we need to count the covariance of the normalized image first. The covariance matrix is a square matrix denoting the covariance of the elements with each other.

```
cov_matrix = np.cov(normalised_training)
eigenvalues, eigenvectors, = np.linalg.eig(cov_matrix)
```

Sort the Eigenvalues in the descending order and the corresponding Eigenvector to arrange the principal component in descending order of their variability. Select K number of Eigenfaces to reduce the data to K number variables.

```
eigen_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]
eigen_pairs.sort(reverse=True)
sort_eigenvalues = [eigen_pairs[index][0] for index in range(len(eigenvalues))]
sort_eigvectors = [eigen_pairs[index][1] for index in range(len(eigenvalues))]
reduced_data = np.array(sort_eigvectors[:7]).transpose()
```

Then, transform the data by dot the transposed reduced data and training data. Then transpose the result. By transposing the outcome of the dot product, the data is reduced to lower dimensions from higher dimensions.

```
proj_data = np.dot(training.transpose(),reduced_data)
proj_data = proj_data.transpose()
```

Find the weight for each normalized data. This weight tells us how important that particular Eigenface is in contributing to the mean face.

```
w = np.array([np.dot(proj_data,i) for i in normalised_training])
```

Finally, recognize all the test images and analyze the accuracy of the recognition results.

```
def recogniser(img, train_images,proj_data,w):
global count,highest_min,num_images,correct_pred
unknown_face = plt.imread('drive/MyDrive/orl/'+img)
num_images += 1
unknown_face_vector = np.array(unknown_face, dtype='float64').flatten()
normalised_uface_vector = np.subtract(unknown_face_vector,mean_face)
w_unknown = np.dot(proj_data, normalised_uface_vector)
diff = w - w_unknown
norms = np.linalg.norm(diff, axis=1)
index = np.argmin(norms)
min(norms)
t1 = 99999999999
t0 = 99999999999
```

```
if norms[index] < t1:
plt.subplot(80,10,1+count)
if norms[index] < t0: # Face is found
if img.split('_')[1] == train_images[index].split('_')[1]:
plt.title('Matched:'+'.'.join(train_images[index].split('.')[:2]),
color='g')
plt.imshow(imread('drive/MyDrive/orl/'+train_images[index]),
cmap='gray')
correct_pred += 1
else:
plt.title('Mismatched:'+'.'.join(train_images[index].split('.')[:2]),
color='b')
plt.imshow(imread('drive/MyDrive/orl/'+train_images[index]),
cmap='gray')
plt.subplots_adjust(right=1.2, top=2.5)
count+=1
fig = plt.figure(figsize=(15, 15))
for i in range(len(test_images)):
recogniser(test_images[i], train_images,proj_data,w)
```

5.2. Results

The experiments are done several times with different dataset split ratio parameters for both algorithms. All of the results can be seen from the tables below.

Table 5.1: DNN Algorithm Results

	Iteration	Loss	Accuracy
Training : 90% Testing : 10%	I	0.2729	97.50%
	II	0.1495	97.50%
	III	0.7175	90.00%
	IV	0.6295	92.50%
	V	0.5469	95.00%
	Accuracy average :		94.50%
Training : 80% Testing : 20%	I	0.5062	92.50%
	II	0.2632	97.50%
	III	0.6995	92.50%
	IV	0.2753	95.00%
	V	0.5852	90.00%
	Accuracy average :		93.50%
Training : 70% Testing : 30%	I	0.4724	90.00%
	II	0.1589	97.50%
	III	0.5163	92.50%
	IV	0.9885	92.50%
	V	2.4577	85.00%
	Accuracy average :		91.50%
Training : 60% Testing : 40%	I	0.9663	92.50%
	II	0.2881	95.00%
	III	0.7447	90.00%
	IV	0.9142	87.50%
	V	1.2623	87.50%
	Accuracy average :		90.50%
Training : 50% Testing : 50%	I	0.8710	91.50%
	II	1.2444	87.50%
	III	1.1506	87.00%
	IV	1.1535	88.00%
	V	1.4483	88.00%
	Accuracy average :		88.40%

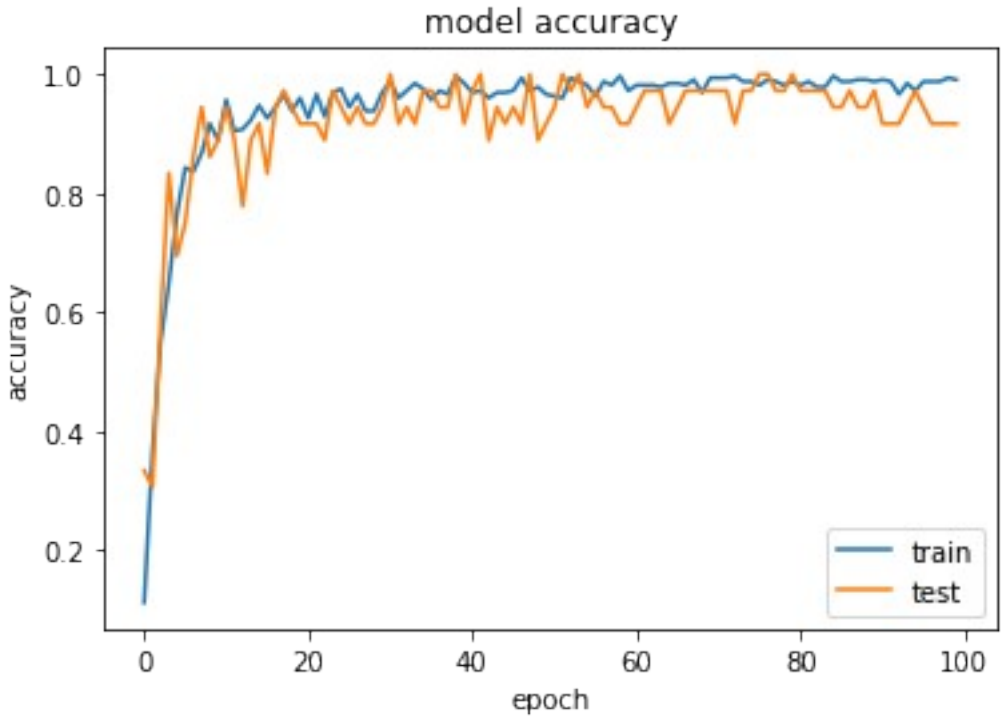


Figure 5.3: DNN Accuracy Model

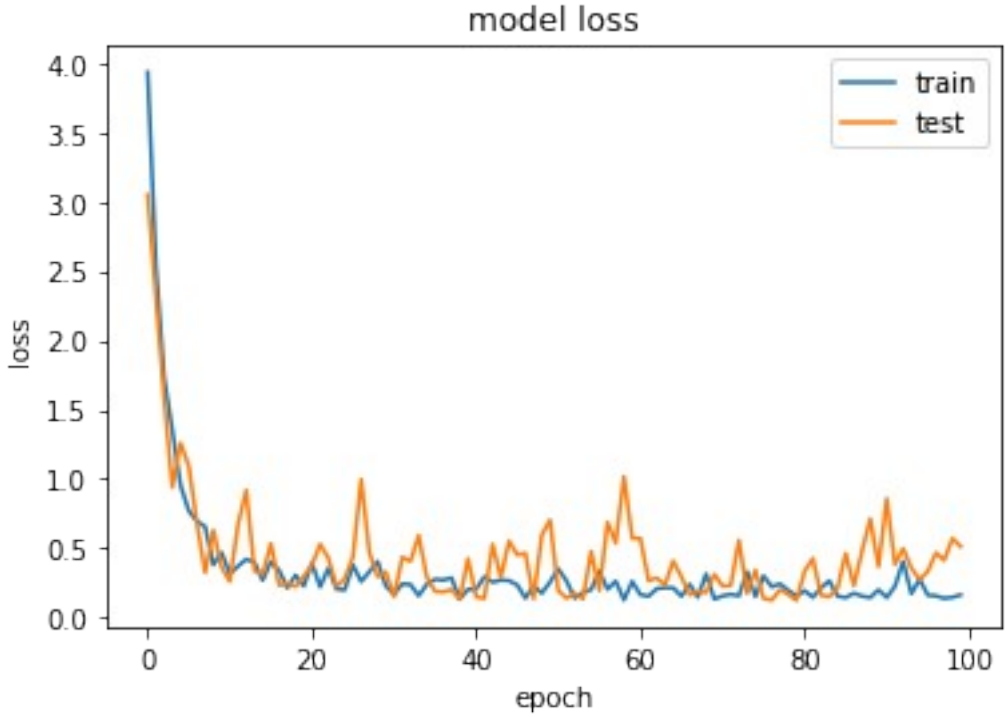


Figure 5.4: DNN Loss Model

The results above show that the larger the training data, the more accurate the DNN algorithm will be, so we can say that the DNN would be more optimum when the training data is more extensive. The highest accuracy is obtained from experiments with training ratios

of 90% and testing of 10%. The accuracy values of these experiments varied from 90% to 97.50%, and the accuracy average after five trials is 94.50%. The accuracy and loss model shows that our model is not underfitting nor overfitting because the train and test are correlated.

Table 5.2: PCA Algorithm Result

	K	Correct Prediction	Accuracy (correct prediction / total test image x 100%)
Training : 90%	5	31	31 / 40 = 77.50%
	7	33	33 / 40 = 82.50%
	10	34	34 / 40 = 85.00%
	12	36	36 / 40 = 90.00%
	16	38	38 / 40 = 95.00%
Training : 80%	5	32	32 / 40 = 80.00%
	7	33	33 / 40 = 82.50%
	10	34	34 / 40 = 85.00%
	12	36	36 / 40 = 90.00%
	16	38	38 / 40 = 95.00%
Training : 70%	5	28	28 / 40 = 70.00 %
	7	30	30 / 40 = 75.00 %
	10	34	34 / 40 = 85.00%
	12	36	36 / 40 = 90.00%
	16	38	38 / 40 = 95.00%
Training : 60%	5	27	27 / 40 = 67.50%
	7	31	31 / 40 = 77.50%
	10	35	35 / 40 = 87.50%
	12	36	36 / 40 = 90.00%
	16	38	38 / 40 = 95.00%
Training : 50%	5	30	30 / 40 = 75.00%
	7	32	32 / 40 = 80.00%
	10	36	36 / 40 = 90.00%
	12	36	36 / 40 = 90.00%
	16	37	37 / 40 = 92.50%

From the results above, we can see that the PCA algorithm's highest accuracy is 95%. It is also noticed that the value of the K variable influences the accuracy results. However, there is no certain value of K; it depends on the dataset used. The value of this variable must go through trial and error. It is also seen that number of data used for training does not significantly affect the accuracy, so we can say that the PCA algorithm is good to use when the dataset is small.

Figure 5.5: Face Recognition with PCA Results

Figure 5.4 is a test with taking 80% of the dataset as training data. We can see that the PCA algorithm achieved an accuracy of 95% with the input of 40 testing images. There are two misrecognized images: a person with id ten who is mistakenly recognized as a person with id eight and a woman with id 35 who is recognized as a man with id 40 instead.

5.3. Comparison

The accuracy of both algorithms is relatively reliable. The minimum accuracy of the DNN algorithm is 87.00%, using 50% training data from the total dataset. Meanwhile, the PCA algorithm works better in certain K values. With 50% training data, the highest accuracy achieved by the DNN algorithms is 91.50%. With the same ratio of training data, the PCA algorithm accuracy can reach 92.5%.

As we can see, the accuracy of the two algorithms is only slightly different. The highest accuracy of the DNN algorithm is 97.50%, and the maximum accuracy of the PCA algorithm is 95%. But there are other factors we need to consider in choosing an algorithm, such as time of implementation. The project's code is compiled with Google Colab, which has 12GB of RAM.

Table 5.3: Highest Accuracy of DNN Algorithm

	Iteration	Loss	Accuracy
Training : 90% Testing : 10%	I	0.2729	97.50%
	II	0.1495	97.50%
	III	0.7175	90.00%
	IV	0.6295	92.50%
	V	0.5469	95.00%
	Accuracy average :		94.50%

Table 5.4: Highest Accuracy of PCA Algorithm

	K	Correct Prediction	Accuracy (correct prediction / total test image x 100%)
Training : 90%	16	38	38 / 40 = 95.00%
Training : 80%	16	38	38 / 40 = 95.00%
Training : 70%	16	38	38 / 40 = 95.00%
Training : 60%	16	38	38 / 40 = 95.00%

The DNN algorithm takes about half a minute to train data with 4096 dimensionalities, while the PCA algorithm needs three-quarters of a minute. There is no significant difference in running time because the dataset used is relatively small. But still, the DNN takes less training and testing time than the PCA algorithm. Both algorithms did not have any difficulties in terms of computation due to the small and neat dataset.

CHAPTER 6
CONCLUSION

From the experiment result, we can say that these two algorithms are almost equivalent. There is only a slight difference between the two algorithms in terms of accuracy and time. However, considering the convenience and running time, in this case, the DNN algorithm is preferred rather than the PCA algorithm. However, there is a chance you might choose the PCA algorithm, especially when your dataset is limited. We must thoughtfully consider the selection of an algorithm according to the needs and abilities of the users themselves. Different conditions can also affect the passage of the algorithm that is suitable for you.

Further research is needed because within this project, the DNN algorithm is implemented using Keras assistance. This might be a great advantage to the DNN result that makes the comparison unbalanced. In the future experiment, we could try to implement the PCA algorithm using the available libraries in Python. We can also try to use the enhanced PCA algorithm to overcome various conditions, especially datasets diversity. Above that, bigger datasets are needed in order to emphasize the final results and conclusion.

More extensive research also intended to conduct the same experiment using a different face database and compare the results with our current experiment to ensure the validity of current implementation over different types and sizes of databases.

REFERENCES

[1] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, Jun. 2014, pp. 1701–1708. doi: 10.1109/CVPR.2014.220.

[2] L. Masupha, T. Zuva, S. Ngwira, and O. Esan, “Face recognition techniques, their advantages, disadvantages and performance evaluation,” in *2015 International Conference on Computing, Communication and Security (ICCCS)*, Pointe aux Piments, Mauritius, Dec. 2015, pp. 1–5. doi: 10.1109/CCCS.2015.7374154.

[3] N. Delbiaggio, “A comparison of facial recognition’s algorithms,” p. 45.

[4] S. Z. Li and A. K. Jain, Eds., *Handbook of Face Recognition*. London: Springer London, 2011. doi: 10.1007/978-0-85729-932-1.

[5] Y. Sun, D. Liang, X. Wang, and X. Tang, “DeepID3: Face Recognition with Very Deep Neural Networks,” *arXiv:1502.00873 [cs]*, Feb. 2015, Accessed: Sep. 26, 2021. [Online]. Available: http://arxiv.org/abs/1502.00873

[6] Maharaja Agrasen College, University of Delhi, Vasundhara Enclave, Delhi - 110096, India, P. Gupta, N. Saxena, M. Sharma, and J. Tripathi, “Deep Neural Network for Human Face Recognition,” *IJEM*, vol. 8, no. 1, pp. 63–71, Jan. 2018, doi: 10.5815/ijem.2018.01.06.

[7] Z. Wang, K. He, Y. Fu, R. Feng, Y.-G. Jiang, and X. Xue, “Multi-task Deep Neural Network for Joint Face Recognition and Facial Attribute Prediction,” in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, Bucharest Romania, Jun. 2017, pp. 365–374. doi: 10.1145/3078971.3078973.

[8] Z. Wu and W. Deng, “One-shot deep neural network for pose and illumination normalization face recognition,” in *2016 IEEE International Conference on Multimedia and Expo (ICME)*, Seattle, WA, USA, Jul. 2016, pp. 1–6. doi: 10.1109/ICME.2016.7552902.

[9] Z. Zhang, J. Li, and R. Zhu, “Deep neural network for face recognition based on sparse autoencoder,” in *2015 8th International Congress on Image and Signal Processing (CISP)*, Shenyang, China, Oct. 2015, pp. 594–598. doi: 10.1109/CISP.2015.7407948.

[10] T. Guo, J. Dong, H. Li, and Y. Gao, “Simple convolutional neural network on image classification,” in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(*, Beijing, China, Mar. 2017, pp. 721–724. doi: 10.1109/ICBDA.2017.8078730.

[11] A. R. S. Siswanto, A. S. Nugroho, and M. Galinium, “Implementation of face recognition algorithm for biometrics based time attendance system,” in *2014 International Conference on ICT For Smart Society (ICISS)*, Bandung, Indonesia, Sep. 2014, pp. 149–154. doi: 10.1109/ICTSS.2014.7013165.

[12] X.-Y. Li and Z.-X. Lin, “Face Recognition Based on HOG and Fast PCA Algorithm,” in *Proceedings of the Fourth Euro-China Conference on Intelligent Data Analysis and Applications*, vol. 682, P. Krömer, E. Alba, J.-S. Pan, and V. Snášel, Eds. Cham: Springer International Publishing, 2018, pp. 10–21. doi: 10.1007/978-3-319-68527-4_2.

[13] A. L. Ramadhani, P. Musa, and E. P. Wibowo, “Human face recognition application using pca and eigenface approach,” in *2017 Second International Conference on Informatics and Computing (ICIC)*, Jayapura, Nov. 2017, pp. 1–5. doi: 10.1109/IAC.2017.8280652.

[14] A. Jadhav, A. Jadhav, T. Ladhe, and K. Yeolekar, “AUTOMATED ATTENDANCE SYSTEM USING FACE RECOGNITION,” vol. 04, no. 01, p. 6.

[15] M. G. Krishnan, “Implementation of Automated Attendance System using Face Recognition.,” vol. 6, no. 3, p. 4, 2015.
[16] M. Broussard, N. Diakopoulos, A. L. Guzman, R. Abebe, M. Dupagne, and C.-H. Chuan, “Artificial Intelligence and Journalism,” *Journalism & Mass Communication Quarterly*, vol. 96, no. 3, pp. 673–695, Sep. 2019, doi: 10.1177/1077699019859901.
[17] H. Hosseini, B. Xiao, and R. Poovendran, “Google’s Cloud Vision API Is Not Robust To Noise,” *arXiv:1704.05051 [cs]*, Jul. 2017, Accessed: Oct. 23, 2021. [Online]. Available: http://arxiv.org/abs/1704.05051
[18] J. Feng, X. He, Q. Teng, C. Ren, H. Chen, and Y. Li, “Reconstruction of porous media from extremely limited information using conditional generative adversarial networks,” *Phys. Rev. E*, vol. 100, no. 3, p. 033308, Sep. 2019, doi: 10.1103/PhysRevE.100.033308.
[19] L. Pauly, H. Peel, S. Luo, D. Hogg, and R. Fuentes, “Deeper Networks for Pavement Crack Detection,” presented at the 34th International Symposium on Automation and Robotics in Construction, Taipei, Taiwan, Jul. 2017. doi: 10.22260/ISARC2017/0066.
[20] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” p. 9.
[21] F. Es-Sabery, A. Hair, J. Qadir, B. Sainz-De-Abajo, B. Garcia-Zapirain, and I. Torre-Diez, “Sentence-Level Classification Using Parallel Fuzzy Deep Learning Classifier,” *IEEE Access*, vol. 9, pp. 17943–17985, 2021, doi: 10.1109/ACCESS.2021.3053917.

APPENDIX

DNN Implementation Code

```
1  from sklearn.datasets import fetch_olivetti_faces
2  from sklearn.model_selection import train_test_split
3  import tensorflow as tf
4  import time
5  import matplotlib.pyplot as plt
6  import numpy as np
7  from tensorflow import keras as k
8  from tensorflow.keras.models import Sequential
9  from  tensorflow.keras.layers  import  Activation,  Dense,  Conv2D,
   MaxPooling2D, Flatten, Dropout
10 from sklearn import preprocessing
11 from keras.regularizers import l2
12 from sklearn import metrics
13 from sklearn.metrics import f1_score
14
15 def main():
16 X,y = fetch_olivetti_faces(return_X_y=True)
17 print(X.shape, y.shape)
18 train_X,   test_X,   train_y,   test_y   =   train_test_split(X,   y,
   test_size=0.10, stratify=y, random_state=42)
19 train_X = preprocessing.scale(train_X)
20 test_X = preprocessing.scale(test_X)
21 print(train_X.shape, train_y.shape)
22 print(test_X.shape, test_y.shape)
23
24 model = Sequential([
25 Dense(units=200,    input_dim=4096,    kernel_regularizer=l2(0.0001),
   activation='relu'),
26 Dropout(0.2),
27 Dense(units=200,    input_dim=200,    kernel_regularizer=l2(0.0001),
   activation='relu'),
28 Dropout(0.2),
29 Dense(units=200,    input_dim=200,    kernel_regularizer=l2(0.0001),
   activation='relu'),
30 Dropout(0.1),
31 Dense(units=200,    input_dim=200,    kernel_regularizer=l2(0.0001),
   activation='relu'),
32 Dropout(0.1),
33 Dense(units=40, input_dim=200, activation='softmax'),])
34 model.summary()
35 model.compile(loss='sparse_categorical_crossentropy',
36 optimizer='rmsprop',
37 metrics=['accuracy'])
38
39 print("Starting training ")
40 start = time.perf_counter()
41 num_epochs = 100
42 h  =  model.fit(train_X,  train_y,  batch_size=32,  epochs=num_epochs,
   validation_split = 0.1, verbose=1)
43 finish = time.perf_counter()
44 print(f"Training finished in {finish - start:0.4f} seconds\n")
45
```

```
print(h.history.keys()) # dict_keys(['accuracy', 'loss'])

print("Training history: ")
for i in range(num_epochs):
loss = h.history['loss'][i]
accuracy = h.history['accuracy'][i] * 100
print("epoch(s): %5d loss = %0.4f accuracy = %0.2f%%" \
% (i, loss, accuracy))

eval = model.evaluate(test_X, test_y, verbose=0)
print("\nEvaluation on test data: \nloss = %0.4f \
accuracy = %0.2f%%" % (eval[0], eval[1]*100) )
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()

plt.plot(h.history['loss'])
plt.plot(h.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()

test_class = np.argmax(model.predict(test_X), axis=-1)
print("Confussion matrix:\n%s" %
metrics.confusion_matrix(test_y, test_class))
print("Classification report:\n%s" %
metrics.classification_report(test_y, test_class))
print("Classification accuracy: %f" %
metrics.accuracy_score(test_y, test_class))
model.save('dnn_model.h5')
if __name__=="__main__":
main()
```

PCA Implementation Code

```
from matplotlib import pyplot as plt
from matplotlib.image import imread
import numpy as np
import os
import time
from google.colab import drive
from google.colab import files
mount= drive.mount('/content/drive')
dataset_path = 'drive/MyDrive/orl/'
dataset_dir = os.listdir(dataset_path)
print(dataset_dir)
train_images = []
test_images = []
width = 80
height = 70

i=1
for j in range(len(dataset_dir)):
img_name = (str(j+1)+'_'+str(i)+'.jpg')
if (j+1)%10==0:
test_images.append(img_name)
i+=1
else:
train_images.append(img_name)
# print(train_images)
# print(test_images)

training    =    np.ndarray(shape=(len(train_images),    height*width),
dtype=np.float64)
for i in range(len(train_images)):
img = plt.imread(dataset_path + train_images[i])
training[i,:] = np.array(img, dtype='float64').flatten()
if i<12:
plt.subplot(3,4,1+i)
plt.subplots_adjust(right=1.5, top=1.5)
plt.imshow(img, cmap='gray')
plt.show()

testing    =    np.ndarray(shape=(len(test_images),    height*width),
dtype=np.float64)
# print(len(test_images))
for j in range(len(test_images)):
img = plt.imread(dataset_path + test_images[j])
testing[j,:] = np.array(img, dtype='float64').flatten()
plt.subplot(9,5,1+j)
plt.subplots_adjust(right=1.2, top=1.2)
plt.imshow(img, cmap='gray')
plt.show()
print(training)

#meanface
mean_face = np.zeros((1,height*width))
print(mean_face)

for i in training:
mean_face = np.add(mean_face,i)
```

```
55
56 mean_face = np.divide(mean_face,len(train_images)).flatten()
57
58 plt.imshow(mean_face.reshape(height, width), cmap='gray')
59 plt.show()
60
61 #normalized
62 normalised_training       =       np.ndarray(shape=(len(train_images),
   height*width))
63
64 for i in range(len(train_images)):
65 normalised_training[i] = np.subtract(training[i],mean_face)
66
67 for i in range(len(train_images)):
68 img = normalised_training[i].reshape(height,width)
69 if i<12:
70 plt.subplot(3,4,1+i)
71 plt.imshow(img, cmap='gray')
72 plt.show()
73
74 #covariance
75 cov_matrix = np.cov(normalised_training)
76
77 #eigen
78 eigenvalues, eigenvectors, = np.linalg.eig(cov_matrix)
79
80 print(eigenvalues)
81
82 #sort and pairing
83 eigen_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index
   in range(len(eigenvalues))]
84 eigen_pairs.sort(reverse=True)
85 sort_eigvalues     =     [eigen_pairs[index][0]     for     index     in
   range(len(eigenvalues))]
86 sort_eigvectors     =     [eigen_pairs[index][1]     for     index     in
   range(len(eigenvalues))]
87
88 #select k eigenfaces
89 reduced_data = np.array(sort_eigvectors[:7]).transpose()
90
91 proj_data = np.dot(training.transpose(),reduced_data)
92 proj_data = proj_data.transpose()
93
94 for i in range(proj_data.shape[0]):
95 img = proj_data[i].reshape(height,width)
96 plt.subplot(2,4,1+i)
97 plt.imshow(img, cmap='gray')
98 plt.show()
99
100      #find weight
101      w     =     np.array([np.dot(proj_data,i)     for     i     in
   normalised_training])
102      w
103
104      #recognize test images
105      count = 0
106      num_images = 0
107      correct_pred = 0
108      def recogniser(img, train_images,proj_data,w):
```

```
109       global count,highest_min,num_images,correct_pred
110       unknown_face = plt.imread('drive/MyDrive/orl/'+img)
111       num_images += 1
112       unknown_face_vector           =           np.array(unknown_face,
    dtype='float64').flatten()
113       normalised_uface_vector                                          =
    np.subtract(unknown_face_vector,mean_face)
114       plt.subplot(80,10,1+count)
115       plt.imshow(unknown_face, cmap='gray')
116       plt.title('Input:'+'.'.join(img.split('.')[:2]))
117       plt.tick_params(labelleft='off',              labelbottom='off',
    bottom='off',top='off',right='off',left='off', which='both')
118       count+=1
119       w_unknown = np.dot(proj_data, normalised_uface_vector)
120       diff = w - w_unknown
121       norms = np.linalg.norm(diff, axis=1)
122       index = np.argmin(norms)
123       min(norms)
124       t1 = 99999999999
125       t0 = 99999999999
126       if norms[index] < t1:
127       plt.subplot(80,10,1+count)
128       if norms[index] < t0: # Face is found
129       if img.split('_')[1] == train_images[index].split('_')[1]:
130       plt.title('Matched:'+'.'.join(train_images[index].split('.')
    [:2]), color='g')
131       plt.imshow(imread('drive/MyDrive/orl/'+train_images[index]),
    cmap='gray')
132       correct_pred += 1
133       else:
134       plt.title('Mismatched:'+'.'.join(train_images[index].split('.')
    [:2]), color='b')
135       plt.imshow(imread('drive/MyDrive/orl/'+train_images[index]),
    cmap='gray')
136       plt.subplots_adjust(right=1.2, top=2.5)
137       count+=1
138       fig = plt.figure(figsize=(15, 15))
139       for i in range(len(test_images)):
140       recogniser(test_images[i], train_images,proj_data,w)
141
142       plt.show()
143
144       print('Correct  predictions:  {}/{}  =  {}%'.format(correct_pred,
    num_images, correct_pred/num_images*100.00))
```

Report #14310419

1. CHAPTER 1 INTRODUCTION 1.1. Background Recognizing faces sounds like a simple thing to humans. We can easily recognize someone in person or maybe through pictures or videos. Nevertheless, it is not that easy for computer vision to do it. Back in the day, we could only see the use of face recognition technology on television or maybe in movies, but now facial recognition technology is commonly used in various fields. Our smartphone is one such example, and almost every phone has the face unlock feature nowadays. Because of that, so many algorithms are developed in order to find the most optimum in terms of time, speed, and costs. One of them is Neural Networks. There has been a surge of interest in neural networks, particularly deep and large networks. These networks have exhibited impressive results [1]. However, besides the significant advantages, beginner researchers have one problem: The approach is computationally expensive and requires a high degree of correlation between the test and training