Face Recognition Dataset

Pedro Sobral - 98491 - (50%) - sobral@ua.pt, Eva Bartolomeu - 98513 - (50%) - evabartolomeu@ua.pt
Departamento de Eletrónica, Telecomunicações e Informática, University of Aveiro, Portugal
Fundamentos de Aprendizagem Automática - Course Instructor: Pétia Georgieva

***Abstract*—The main goal of this project is to apply suitable machine learning algorithms, notably deep learning techniques learned in class to solve a specific data science problem, in this case, models capable of recognize faces. All the data comes from a Kaggle Dataset [11]. In this report, we try to obtain the best result of the accuracy of the implemented models.**

***Index Terms*—Machine Learning, Classification Model, Logistic Regression, SVM, Neural Network, Normalization, Keras, DeepLearning**

I. Introduction

In the scope of the subject Fundamentals of Automatic Learning, a work was proposed on a problem (of our choice) of machine learning. The goal of this project is to apply suitable machine learning algorithms learned in class or self-learned to solve a specific data science problem (classification, regression, clustering). Represent the results in graphical/table formats and make analysis and conclusions. We decided to choose project proposal 3, Face Recognition from Olivetti Dataset. This data set can be used to recognize a face. This dataset is splited in two modules. Target data that have 10 different images of a determined face, and the data itself with other 10 images from the same face but different from the target dataset. Our principal objectives are determine a adequate number of principal components, using the PCA and train models to obtain the best results possible. Will be utilized the Convolutional neural network (CNN) deep learning model, this method was learned during the class. This method is a much more sophisticated approach, to do this implementation will be used the Keras library as in the classes.

II. State of Art

The article [1] presents a new face recognition method that combines the use of Principle Component Analysis (PCA) as a feature extractor and Support Vector Machines (SVMs) as a classifier. The authors claim that the performance of a face recognition system is determined by the accuracy of the feature extraction and classification. In this paper, PCA is used to extract features from the face images and SVMs are used to classify them. The authors evaluated the developed method, finding that recognition based on SVMs outperforms the standard eigenface approach, and that the results have better accuracy with the application of PCA.

The article [3] presents a new image classification method that uses a simple Convolutional Neural Network (CNN), and the respective evaluation with data set. The authors analyze the influence of different methods of learning rate set and different optimization algorithms on image classification. CNN proves to be highly effective in image classification tasks.

The article [2] presents a new face recognition method that uses Linear Discriminant Analysis (LDA), and the respective evaluation with data set. The LDA shows a high correct recognition rate for the face recognition problem.

The article [4] presents a new face recognition method that utilizes machine learning algorithms and PCA. The authors claim that traditional face recognition methods have difficulty in real-world situations, such as partial facial occlusion, illumination, and posture variations. The proposed method is tested using linear discriminant analysis, multilayer perceptron, Naive Bayes, and support vector machine, and achieves high accuracy results.

The article [5] presents a new face recognition system that uses machine learning classifier algorithms and PCA for feature extraction. The study evaluates seven machine learning algorithms, such as SVM, Decision Tree, K-Nearest Neighbour, Logistic Regression, Naive Bayes, Multi-Layer Perceptron, and CNN for their performance on the Olivetti faces database.

III. Data Description

Face recognition is a technique to identify people by the face characteristics. So our dataset [11] have data, in this case face images, taken between april 1992 and april 1994 from 40 different people. Each person have 10 different images with different facial expressions, variying ligthing, facial express and facial detail. There are 400 differente images in the dataset, each image have a size of 64x64 pixels and all images are in the gray scale. Names of 40 people were encoded to an integer from 0 to 39. The dataset were initially separate between data itself and target data, where target data have 10 images from the each person, although this images are different from the data that we will use to train our models.

IV. Data Visualization

So we start by visualize the content of the dataset, the figure 1 shows various sifferent faces, and different shots of each face with diferent angles and different facial expressions. As said before all the images are in gray scale.

Fig. 1. Example of data from the dataset

The image data is in matrix form, and before dividing the data into test and training, it is necessary to convert the respective data into a vector.

We decided that 20% of the data will be used for testing and 80% of the data will be used for training. For this we created two sets, the test, and the training, using the sklearn.model_selection.train_ test_ split library [18].

In order to have deep knowledge about the data-set it is important to have data visualization of the dataset. The dataset is balanced, this is important so that we have more chance that the model will be certain because the data is well distributed and non-bias. The next figure (Fig. 2) shows that the number of cases of each class is the same for Training Set.

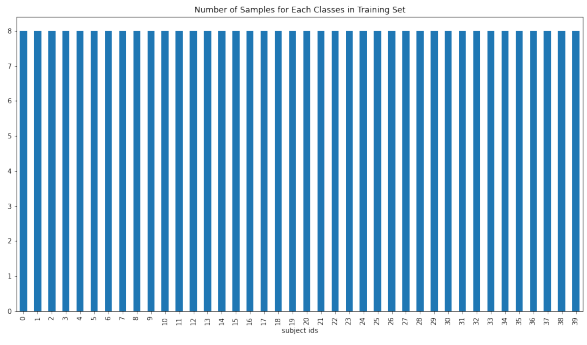

Fig. 2. Class Distribution in Training Set

The Figure 3 also shows that the number of cases of each class is the same for Test Set.

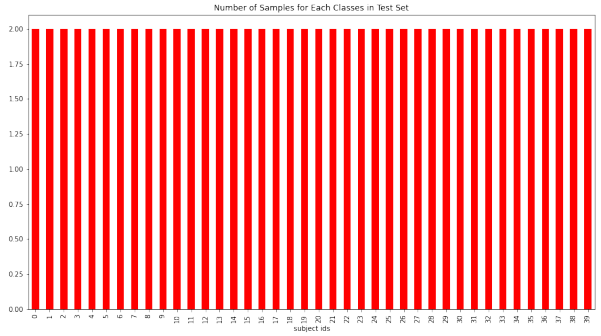

Fig. 3. Class Distribution in Test Set

V. Data Preprocessing

As it is a dataset of images, the concept of missing values or outliers is not applicable here. In order to apply a dimension reduction we implement the PCA [21], that identifies and discards features that are less useful for making a valid approximation on a dataset. At the same time, to maximise the orthogonality between the features in the transformed feature space.

In order to determine the appropriate number of components for the PCA, we developed a graph that illustrates the trade-off between the number of components used and the amount of information retained in the data after the PCA (Fig. 4).

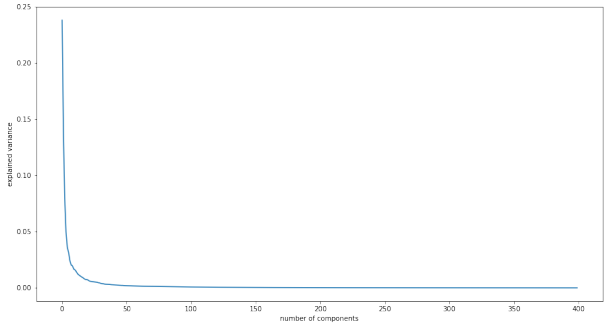

Fig. 4. Finding Optimum Number of Principle Component

Through this graph, we can conclude that the use of more than 50 components does not contribute significantly to the retention of information in the data.

Based on the previous conclusions, we decided to apply the PCA to the training and test sets, using 50 components and with whitening enabled (Fig. 5 and Fig. 6). Whitening is a technique that ensures that the components have zero correlation and unit variance.

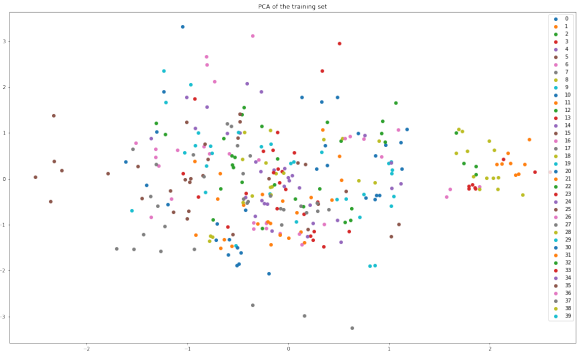

Fig. 5. PCA of the training set

Fig. 7. Average Face after PCA

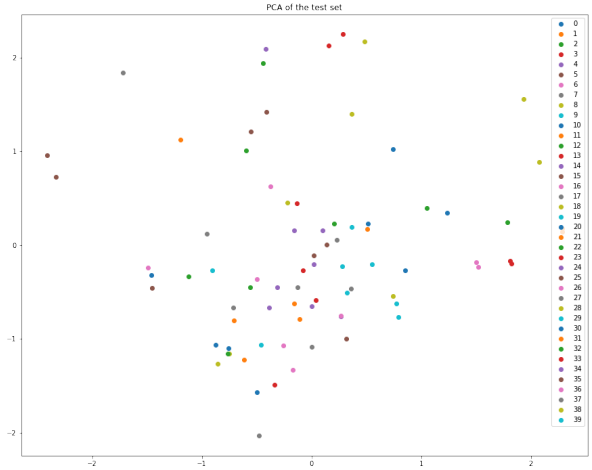

Fig. 6. PCA of the test set

The sum of all explained variance rates is (87%). Graphics of the PCA-transformed sets show that the different classes can be separated by a linear boundary and that the model learned from the training data can generalize well to unseen data.

After PCA, we develop a Average Face image (Fig. 7), which is obtained by taking the average of all images in the dataset, reshaping it to the shape of the original image, and displaying it in grayscale. The average face can be used to analyze the general characteristics of the dataset and as a reference to other faces in the dataset.

Figure 8 presents all the eigenfaces generated by the PCA. Eigenfaces are the main components of the dataset after the PCA and are used to represent the dataset with fewer dimensions. Furthermore, they are linear combinations of the original dataset and are ordered by the amount of variation they represent.

Fig. 8. Eigen Faces after PCA

VI. Machine Learning Models

In each implemented machine learning algorithm, we run the algorithm with the base model, with the hyper-parameter selection model, and with K-Fold Cross Validation on the hyper-parameter selection model. For the execution of the described models, we created a function that trains the model passed as an argument and shows graphs, tables, and data in the output, in order to facilitate the analysis of the model. We also implemented a function that, through the sklearn.model_selection. GridSearchCV library [13], performs an exhaustive search on a list of parameters, training a given model with all the parameters, in the end, we return the parameters that gave the best performance, according to the accuracy strategy.

We developed a function that splits the data set into k mixed sets, with training/test indices, using the sklearn.model_selection.KFold feature [14]. Still, within the function, we evaluate the metrics by cross-validation and record the fit/score times through sklearn.model_selection.cross_validate [12].

Finally, we create a function that calculates training and testing scores for an estimator with several different values on a specific parameter, these calculations are performed with the sklearn.model_selection.validation_curve [20]. We then graphed the average training scores, and the average test scores, on each specific parameter value.

A. Logistic Regression

The Logistic Regression [8] algorithm determines the probability of a binary event (which has only two different results). The Logistic Regression category is supervised learning.

In our problem, we use this algorithm, where the binary results are malignant or benign cancer. We performed the algorithm using the sklearn.linear_model.LogisticRegression library [16], and with the previously described functions.

B. Gaussian Naive Bayes

Gaussian Naive Bayes [22]is a probabilistic machine learning algorithm that is used for classification tasks. It is based on the Bayes theorem and assumes that the data follows a normal (Gaussian) distribution. The algorithm uses the probability density function of the normal distribution to estimate the probability of a given sample belonging to each class. It then classifies the sample into the class with the highest probability.

C. Support Vector Machine

SVM [10] is a supervised learning algorithm, which analyzes data for classification and regression observation. Considering a set of training examples, where each example belongs to a category (there are two categories, two results), the SVM algorithm builds a model that classifies new data to a category. This classifier is a non-probabilistic linear binary. Examples are mapped to points in space, with a wide gap between categories. We use sklearn.svm.SVC [17] to develop this algorithm.

D. k-Nearest Neighbor

k-NN [7] is a non-parametric, supervised learning algorithm that uses the proximity of points to classify or predict, assuming that similar points are close. We implemented the algorithm using the sklearn.neighbors.KNeighborsClassifier [15] library.

E. Decision Tree

Decision Tree [9] is supervised learning, presents a tree structure, the internal nodes are the characteristics of a dataset, the branches are the decision conditions, and the leaf nodes are the results. Graphically represents all possible solutions to a problem. In the development of the algorithm, we used sklearn.tree_DecisionTreeClassifier [19].

F. Linear Discriminante Analysis

Linear Discriminant Analysis (LDA) [23] is a supervised machine learning algorithm that is used for classification problems. It is a simple and powerful technique that finds a linear combination of features that separates different classes most effectively. The basic idea behind LDA is to project the data onto a lower-dimensional space where the classes are well-separated. This is achieved by maximizing the ratio of the between-class variance to the within-class variance.

G. CNN

Convolutional Neural Networks (CNNs) [6] are a type of deep learning algorithm that are commonly used in image and video recognition tasks. They are designed to process data with a grid-like topology, such as an image. CNNs consist of several layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply a set of filters to the input data to learn local features, such as edges and textures. The pooling layers then reduce the spatial dimension of the data while retaining important features. The fully connected layers combine the features learned by the previous layers and make a final prediction. CNNs have been shown to be highly effective in tasks such as image classification, object detection, and facial recognition.

VII. Results

A. Logistic Regression

First, we run this algorithm with the base model, with the hyper-parameter selection model, and with K-Fold cross-validation on the hyper-parameter selection model. In Table I we can see the parameters chosen in the hyper-parameter selection model.

TABLE I
Best parameters in Logistic Regression

C	class_weight	max_iter	penalty	solver
0.1	balanced	100	l2	liblinear

The value selected for parameter C was 0.1. The higher the value of this parameter, the greater the weight on the training data and the lower the weight on the complexity penalty. The

smaller the value of this parameter, the smaller the weight in the training data and the greater the weight in the complexity penalty. This parameter is used to handle extreme data.

The parameter selected in the class_weight was “balanced”, this strategy implicitly replicates the class with the lowest weight, until there is the same number of samples as the class with the highest weight.

The chosen max_iter was 100, that is, the maximum number of iterations for the solvers to converge is 100.

The “l2” parameter was selected for the penalty. This parameter imposes the penalty of the square of the magnitude of the coefficients in the model. Thus reducing the coefficients with less contribution, that is, regularization is performed.

The parameter chosen in the solver was “liblinear”, this strategy is advised in small data sets, such as our set. This strategy behaves like a multiclass classified, through the co-ordinate descent algorithm, ie, separate binary classifiers are trained for all classes.

1) Base Model: The results of the Base model application can be observed in the Figures 9 and 10.

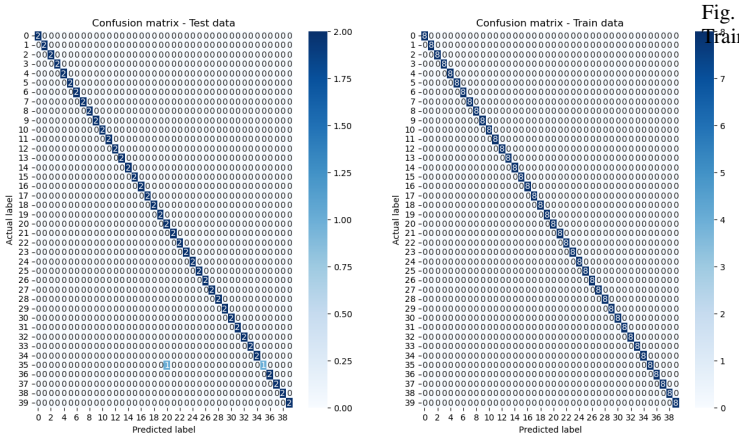

Fig. 9. Logistic Regression Base Model - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.988	0.992	0.988	0.987	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 10. Logistic Regression Base Model - Table with Test and Train Results for Logistic Regression

2) Hyper-Parameter Selection Model: The results of the Hyper-Parameter Selection Model can be observed in the Figures 11 and 12.

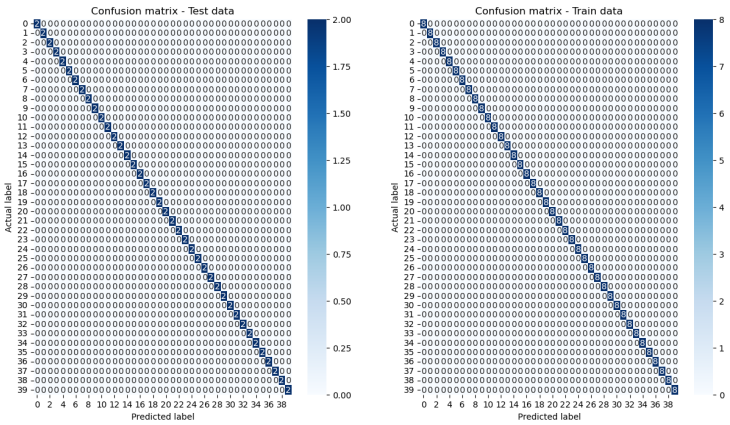

Fig. 11. Logistic Regression Hyper-Parameter SM - Confusion Matrix with Train and Test data for Logistic Regression

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 12. Logistic Regression Hyper-Parameter SM - Table with Test and Train Results

3) K-Fold Cross-Validation Model: The results of the K-Fold Cross-Validation Model can be observed in the Figures 13 and 14.

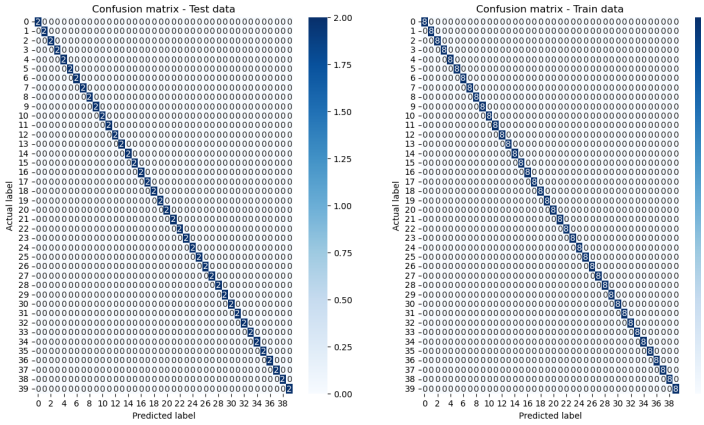

Fig. 13. Logistic Regression K-Fold CV - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 14. Logistic Regression K-Fold CV - Table with Test and Train Results

By the figures 10, 12 and 14 we can verify that the Accuracy and F1 score of the 3 methods are very identical, with almost no difference in choosing the best method.

B. Gaussian Naive Bayes

We run this model in three different ways, the base model, the hyper-parameter selection model, and the K-Fold cross-validation with the hyper-parameter selected. The following Table II shows the parameters chosen by the algorithm.

TABLE II
BEST PARAMETERS IN GAUSSIAN NAIVE BAYES

activation	alpha	hidden_layer_sizes	learning_rate
relu	1	(12, 12)	constant

var$_s moothing$	var$_s moothing$
0.533669923120631	0.533669923120631

1) Base Model: The results of the Base model application can be observed in the Figures 15 and 16.

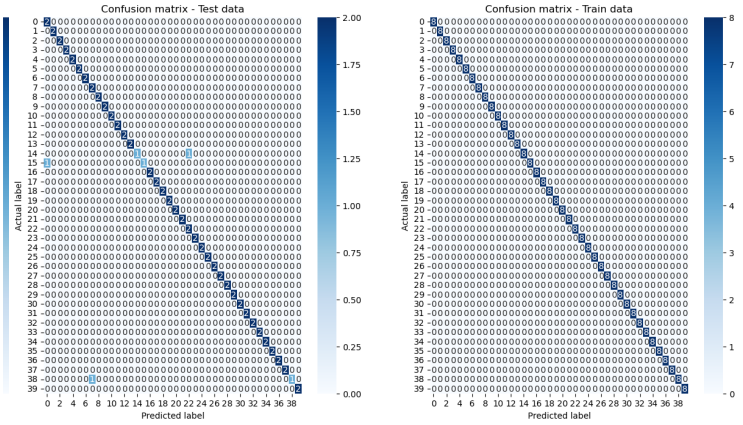

Fig. 15. GNB Base Model - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.963	0.975	0.963	0.960	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 16. GNB Base Model - Table with Test and Train Results

2) Hyper-Parameter Selection Model: The results of the Hyper-Parameter Selection Model can be observed in the Figures 17 and 18.

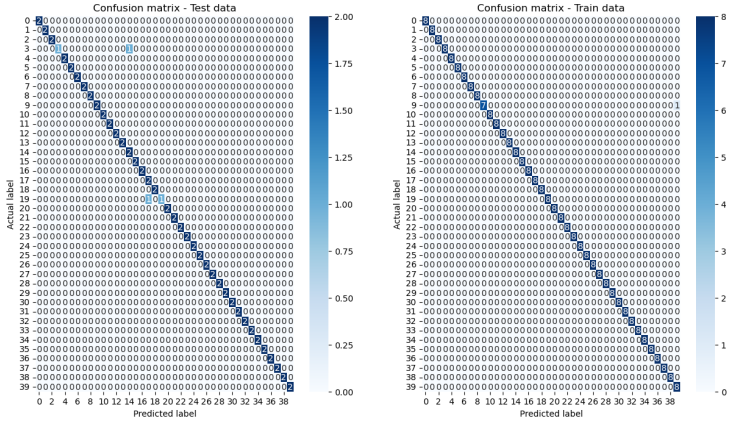

Fig. 17. GNB Hyper-Parameter Selection Model - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.975	0.983	0.975	0.973	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
0.997	0.997	0.997	0.997	320.000

Fig. 18. GNB Hyper-Parameter Selection Model - Table with Test and Train Results

3) K-Fold Cross-Validation Model: The results of the K-Fold Cross-Validation Model can be observed in the Figures 19 and 20.

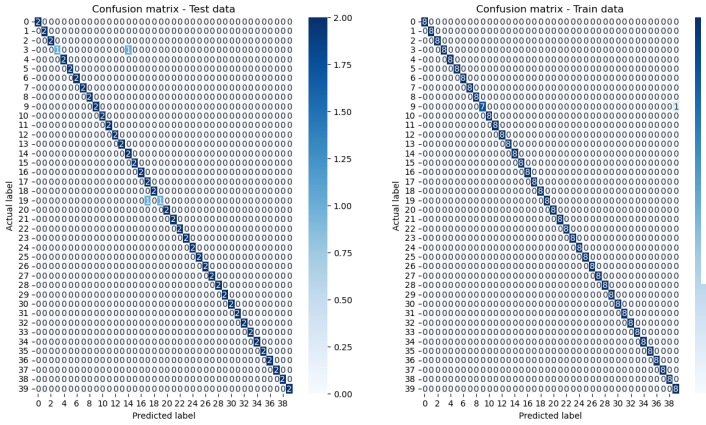

Fig. 19. GNB K-Fold CV - Confusion Matrix with Test and Train data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.975	0.983	0.975	0.973	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
0.997	0.997	0.997	0.997	320.000

Fig. 20. GNB K-Fold CV - Table with Test and Train Results

C. Support Vector Machine

We run this model in three different ways, the base model, the hyper-parameter selection model, and the K-Fold cross-validation with the hyper-parameter selected. The following Table III shows the parameters chosen by the algorithm.

TABLE III
BEST PARAMETERS IN SVM

C	class_weight	gamma	kernel
10	balanced	0.01	rbf

1) Base Model: The results of the Base model application can be observed in the Figures 21 and 22.

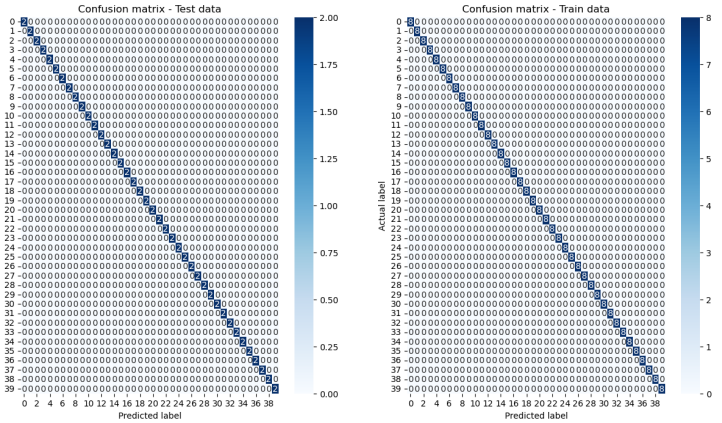

Fig. 21. SVM Base Model - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 22. SVM Base Model - Table with Test and Train Results

2) Hyper-Parameter Selection Model: The results of the Hyper-Parameter Selection Model can be observed in the Figures 23 and 24.

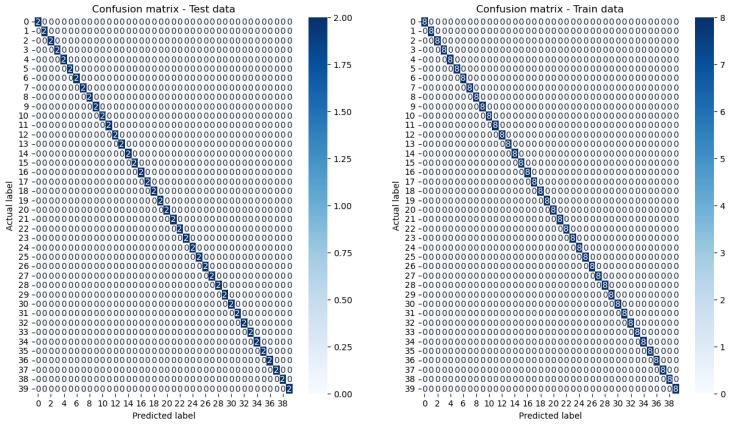

Fig. 23. SVM Hyper-Parameter SM - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 24. SVM Hyper-Parameter SM - Table with Test and Train Results

3) K-Fold Cross-Validation Model: The results of the K-Fold Cross-Validation Model can be observed in the Figures 25 and 26.

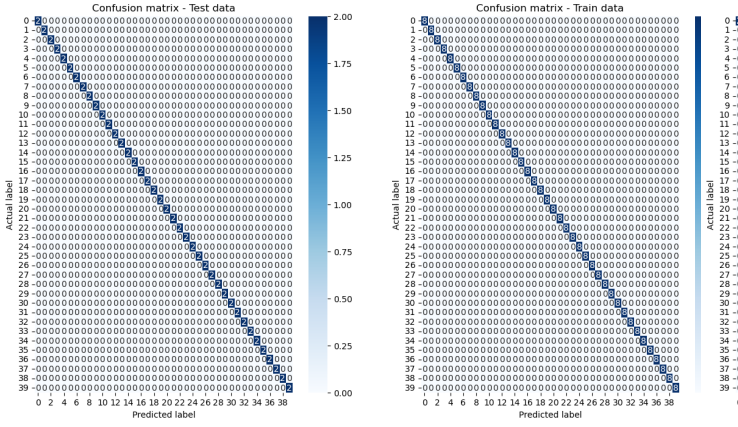

Fig. 25. SVM K-Fold CV - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 26. SVM K-Fold CV - Table with Test and Train Results

D. k-Nearest Neighbor

We run this model in three different ways, the base model, the hyper-parameter selection model, and the K-Fold Cross Validation with the hyper-parameter selected. The following Table IV shows the parameters chosen by the algorithm.

algorithm	$n_neighbors$	weights
auto	3	distance

TABLE IV
BEST PARAMETERS IN K-NEAREST NEIGHBOR

1) Base Model: The results of the Base model application can be observed in the Figures 27 and 28.

Fig. 27. kNN Base Model - Confusion Matrix with Train and Test data

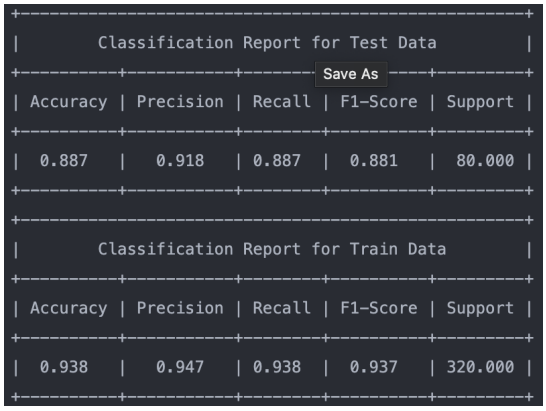

Fig. 28. kNN Base Model - Table with Test and Train Results

2) Hyper-Parameter Selection Model: The results of the Hyper-Parameter Selection Model can be observed in the Figures 29 and 30.

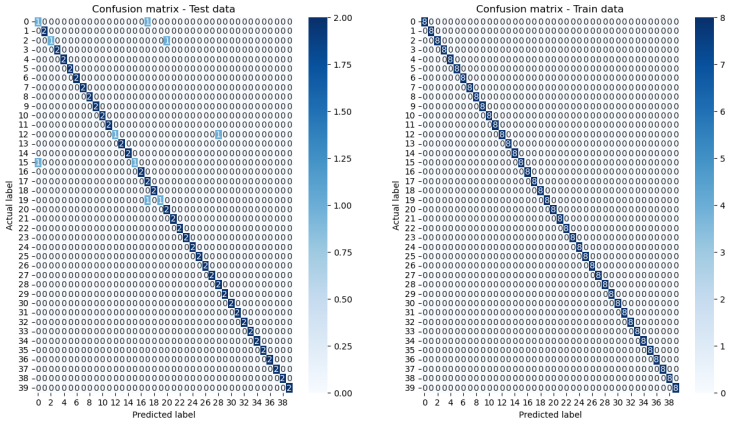

Fig. 29. kNN Hyper-Parameter SM - Confusion Matrix with Train and Test data

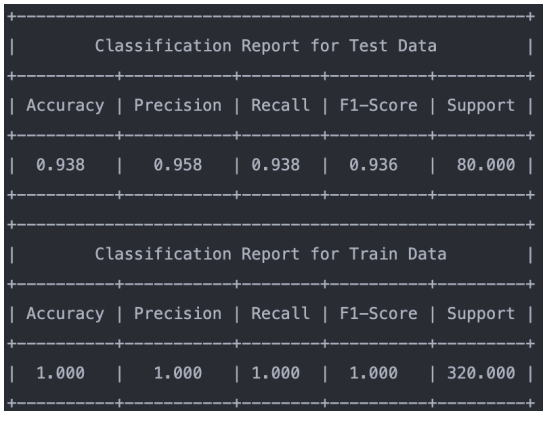

Fig. 30. kNN Hyper-Parameter SM - Table with Test and Train Results

3) K-Fold Cross-Validation Model: The results of the K-Fold Cross-Validation Model can be observed in the Figures 31 and 32.

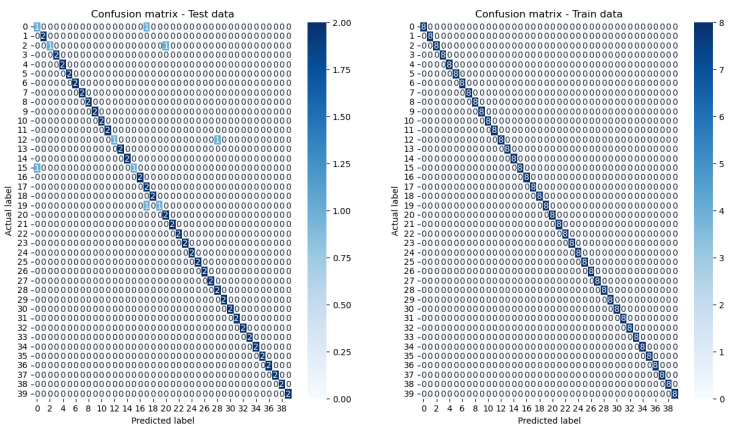

Fig. 31. kNN K-Fold CV - Confusion Matrix with Train and Test data

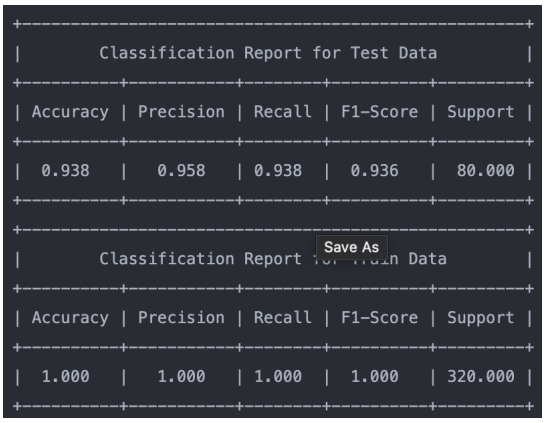

Fig. 32. kNN K-Fold CV - Table with Test and Train Results

E. Decision Tree

We run this model in three different ways, the base model, the hyper-parameter selection model, and the K-Fold cross-validation with the hyper-parameter selected. The following Table V shows the parameters chosen by the algorithm.

criterion	max_depth	min_samples_leaf	min_samples_split	splitter
entropy	8	1	7	best

TABLE V
BEST PARAMETERS IN DECISION TREE

1) Base Model: The results of the Base model application can be observed in the Figures 33 and 34.

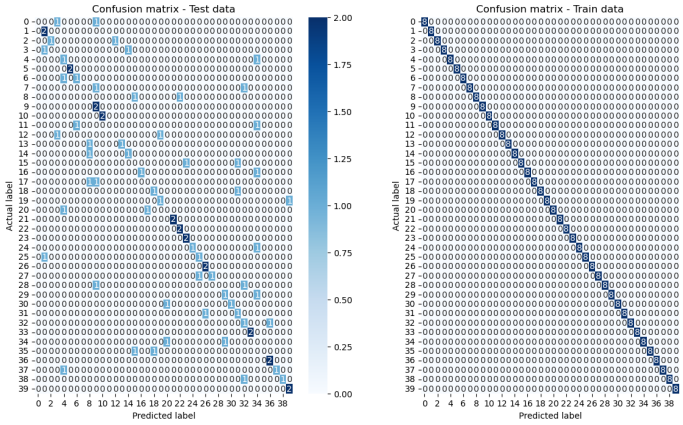

Fig. 33. Decision Tree Base Model - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.487	0.500	0.487	0.464	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
1.000	1.000	1.000	1.000	320.000

Fig. 34. Decision Tree Base Model - Table with Test and Train Results

2) Hyper-Parameter Selection Model: The results of the Hyper-Parameter Selection Model can be observed in the Figures 35 and 36.

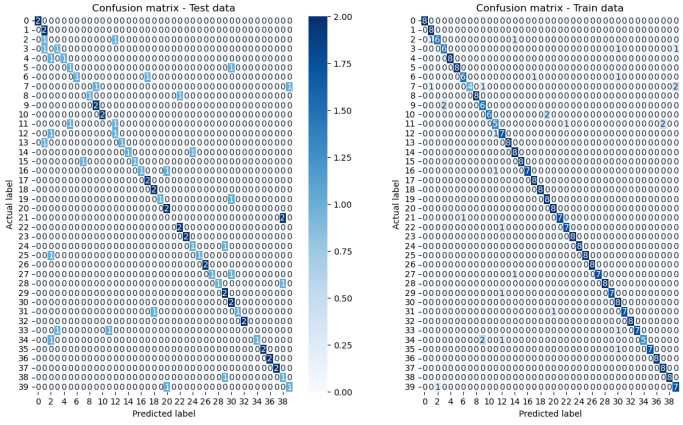

Fig. 35. Decision Tree Hyper-Parameter SM - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.637	0.701	0.637	0.627	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
0.906	0.919	0.906	0.905	320.000

Fig. 36. Decision Tree Hyper-Parameter SM - Table with Test and Train Results

3) K-Fold Cross-Validation Model: The results of the K-Fold Cross-Validation Model can be observed in the Figures 37 and 38.

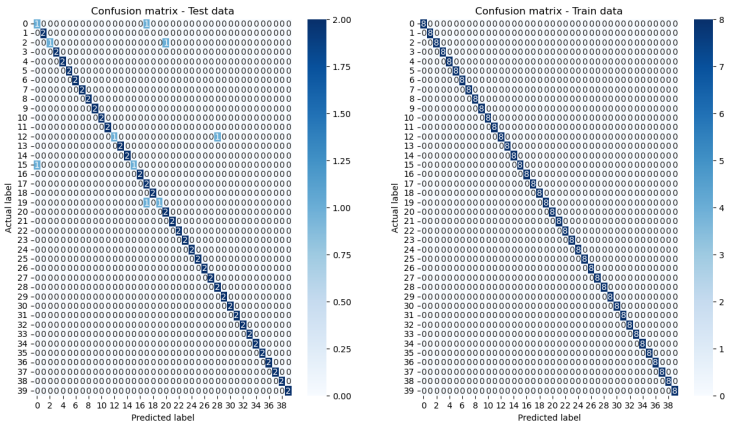

Fig. 37. Decision Tree K-Fold CV - Confusion Matrix with Train and Test data

Classification Report for Test Data				
Accuracy	Precision	Recall	F1-Score	Support
0.650	0.639	0.650	0.613	80.000

Classification Report for Train Data				
Accuracy	Precision	Recall	F1-Score	Support
0.906	0.920	0.906	0.906	320.000

Fig. 38. Decision Tree K-Fold CV - Table with Test and Train Results

F. CNN

The results of the CNN can be observed in the Figure 39.

```
10/10 [==============================] - 1s 53ms/step - loss: 0.0212 - accuracy: 1.0000
Test accuracy, on train data: 1.0
Test loss, on train data: 0.021180707961320877
3/3 [==============================] - 0s 38ms/step - loss: 0.0630 - accuracy: 0.9875
Test accuracy, on test data: 0.987500011920929
Test loss, on test data: 0.06295185536146164
```

Fig. 39. Results

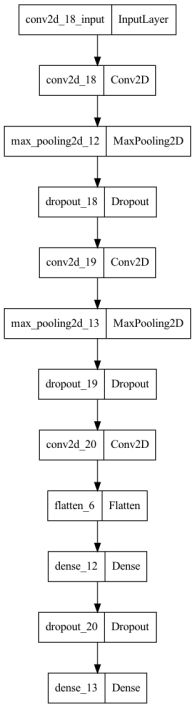

Fig. 40. Model

This model was built with the parameters shown in Figure 40.

In the Figure 41 we can see the evolution of the accuracy through the number of epochs.

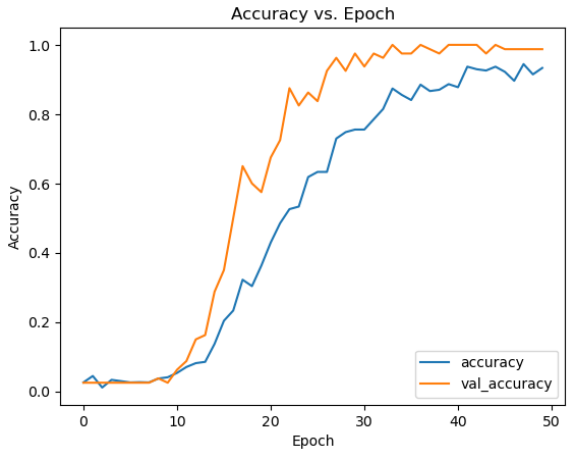

Fig. 41. Accuracy vs. Epochs

G. Comparison between the models

The comparison here VI will be done with the best result from all models done in the previous section.

TABLE VI
CLASSIFICATION - ALL MODEL

——	Accuracy(Test)	Accuracy(Train)
Logistic Regression (HP/K-Fold)	1.0	1.0
Gaussian Naive Bayes (HP/ K-Fold)	0.975	0.997
SVM (Base Model/HP/ K-Fold)	1.0	1.0
K-NN (HP/K-Fold)	0.938	1.0
Decision Tree (K-Fold)	0.650	0.906
LDA (Base Model/HP/ K-Fold)	1.0	1.0
CNN	0.987	1.0

As we can see all models give us a great solution, apart Decision Tree, but we can consider the Logistic Regression and the SVM models the best, comparing the results with the others.

VIII. NOVELTY AND CONTRIBUTIONS

We found two published articles that study the same problem as us, Performance Evaluation of Machine Learning Classifiers for Face Recognition [5] and Face Recognition System Using Machine Learning Algorithm [4]. The results obtained in the articles can be seen in the following tables VII VIII.

TABLE VII
ARTICLE PERFORMANCE EVALUATION RESULTS

Machine Learning Model	Accuracy
PCA + Logistic Regression	0.975
PCA + Naïve Bayes	0.950
PCA + SVM	0.988
PCA + KNN	0.888
PCA + Decision Tree	0.575
CNN	0.988

TABLE VIII
ARTICLE FACE RECOGNITION SYSTEM RESULTS

Machine Learning Model	Accuracy
PCA+Linear Discriminant Analysis	1
PCA + Naïve Bayes	0.95
PCA + SVM	1

All our models give better, higher accuracy values than the paper Performance Evaluation of Machine Learning Classifiers for Face Recognition [5] results. With paper Face Recognition System Using Machine Learning Algorithm [4], they give the same results, with the exception of Naïve Bayes which gives a slightly better result with our implemented solution.

Based on this analysis, we can say that we propose a better solution, with better performance of Machine Learning models.

IX. Conclusion

Doing this project was very relevant since we not only successfully applied the machine learning algorithms learned in class, but we also self-learned new technics of machine learning using the internet. As we were developing our work, there were some periods where we found some difficulties, especially when we were dealing with data preprocessing and when we were implemented the CNN model. If we were to develop this work even further we would like to improve our data preprocessing. In conclusion, we can say that our best models are:Logistic Regression, LDA, and SVM with 100% accuracy. In the end, we are very content with the work we presented, since it made us increase our knowledge.

References

[1] Md. Omar Faruqe and Md. Al Mehedi Hasan. "Face recognition using PCA and SVM". In: *2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication*. 2009, pp. 97–101. DOI: 10.1109/ICASID.2009.5276938.

[2] Suman Kumar Bhattacharyya and Kumar Rahul. "Face recognition by linear discriminant analysis". In: *International Journal of Communication Network Security* 2.2 (2013), pp. 31–35.

[3] Tianmei Guo et al. "Simple convolutional neural network on image classification". In: *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. 2017, pp. 721–724. DOI: 10.1109/ICBDA.2017.8078730.

[4] Sudha Sharma, Mayank Bhatt, and Pratyush Sharma. "Face Recognition System Using Machine Learning Algorithm". In: *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. 2020, pp. 1162–1168. DOI: 10.1109/ICCES48766.2020.9137850.

[5] Dodi Sudiana, Mia Rizkinia, and Fahri Alamsyah. "Performance Evaluation of Machine Learning Classifiers for Face Recognition". In: *2021 17th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*. 2021, pp. 71–75. DOI: 10.1109/QIR54354.2021.9716171.

[6] *Convolutional Neural Network*. URL: https://www.happiestminds.com/insights/convolutional-neural-networks-cnns/.

[7] IBM. *knn-definition*. URL: https://www.ibm.com/topics/knn.

[8] javatpoint. *logistic-regression-definition*. URL: https://www.javatpoint.com/logistic-regression-in-machine-learning.

[9] javatpoint. *tree-definition*. URL: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm.

[10] MonkeyLearn. *svm-definition*. URL: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/.

[11] Olivetti. *Face Recognition from Olivetti Data Set*. URL: https://www.kaggle.com/code/serkanpeldek/face-recognition-on-olivetti-dataset/notebook.

[12] scikit-learn. *cross-validate-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html.

[13] scikit-learn. *GridSearchCV-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[14] scikit-learn. *KFold-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.

[15] scikit-learn. *knn-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[16] scikit-learn. *logistic-regression-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[17] scikit-learn. *svm-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[18] scikit-learn. *train-test-split-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[19] scikit-learn. *tree-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[20] scikit-learn. *validation-curve-documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.validation_curve.html.

[21] sklearn. *PCA: Principal Component Analysis using Python (Scikit-learn)*. URL: https://www.jcchouinard.com/pca-with-python/.

[22] upGrade. *Gaussian Naive Bayes*. URL: https://www.upgrad.com/blog/gaussian-naive-bayes/.

[23] Wikipidia. *Linear Discriminante Analysis*. URL: https://en.wikipedia.org/wiki/Linear_discriminant_analysis.