Unit Code: CIS007-2
Assignment: 2165061
Unit Coordinator: Dr Vitaly Schetinin

Student name: Adedamola Olusunmade
ID: 2165061
Email: Adedamola.olusunmade@study.beds.ac.uk

1. Introduction:

We were tasked with the development of a Machine Learning (ML) solution to solve a biometric recognition task with the highest recognition accuracy. The facial images are taken from real subjects in slightly different conditions, so some images can be incorrectly recognised. This makes the ideal 100% accurate recognition difficult or even impossible. We are to design an ML solution providing minimal recognition error.

Artificial Neural Networks (ANNs) and Deep Learning (DL)are capable of learning patterns from data in the presence of noise and natural variations. In recent years the ANNs applied to face recognition problems have achieved an above-human accuracy. However, various edge case conditions thrown into the process drastically reduce the recognition performance. **(Selitskaya et al., 2021)**

To create a solution, I'm using traditional Multilayer Perceptron (MLP) models because the Dataset is not large enough to justify using Deep Learning and simpler MLP models are more practical for people with limited computational resources and time constraints.

2. Designing a Solution

I chose to use traditional Multilayer Perceptron (MLP) models because the data size is relatively small and I wanted the code to run on all systems, even the ones with limited computational resources, but as I add more data to this solution, the use of Deep Learning Models will be heavenly favoured. I also chose to use Google Collab because it's easy to use and it offer free GPU resources on less busy hours.

Accuracy may vary when ANN runs with the same parameters due to different reasons, it may be because of the initial weight of the neural network which can lead to different convergence paths during training. Data Shuffling can also make the accuracy vary. Difference in Hardware (e.g GPU, CPU, TPU) or software environment can influence the numerical precision of computations leading to variation of accuracy in the result. Different runs may converge to different local minima based on the initial conditions.

When comparing different models or parameters, it is very important to check the performance across multiple subsets of data. K-fold cross validation allows for better utilization of data and reduction of variance of the results, which is what I chose to use.
In k-fold cross validation, the dataset is divided in 'k' number of folds with are equally sized. The model is trained and evaluated 'k' times using different folds as the test set and the remaining data as the training set and the results are then averaged over the 'k' iterations. K-fold cross validation helps to reduce variance and ensure robustness.

Here I am going to use two scripts: ‘Process_yale_images script and Classify_yale Script.

With the Process_yale_images script being used to set up the images, target, and data, and the Classify_yale Script being used to train Data, transform data and Test Data.

Process_Yale_images script explanation.

This set of Codes mounts the Google Drive at the specified path to the Google Collab which is then accessed for its data. Then it lists all the files in its directory

This then unzips the Tro.zip file that contains the images I need and then makes a list of all the images in the directory I created.

This then displays the image using Matplotlib. This helps users to check if they unzipped the correct set of files.

This code is then used to load the images and store them in a matrix for further processing. It effectively loads data and labels into the NumPy array for further analysis and processing.

The data will be a matrix of n=1500 rows and h*w columns, where n is the number of images, h=77 and w=68 are the height and width of an image in pixels. The target will be a nx1 matrix.

This then saves the target and data on Google Drive as 'yaleExtB_target.npy' and 'yaleExtB_data.npy' which then be used for Machine Learning. After I do so, then I show the files in the root directory to check if the data has been successfully saved in the right file.

Classify_yale Script Explanation

Like I said before, this is used the mount the Google Drive to the Google Collab, it prompts the user to sign in and gives the notebook access to the Google Drive which will be needed for further execution of the Machine Learning

This code snipped is used to load the two NumPy Arrays previously created from the specified path in Google Drive. Then load the 'yaleExtB_target.npy' file into variable 'y' and 'yaleExtB_data.npy' into variable 'x.'

This imports the necessary Libraries and modules that will be needed for this task.

```
# Number of principal components for PCA
nof_prin_components = 200

nohn = 400

# PCA
pca = PCA(n_components=nof_prin_components, whiten=True)
X_pca = pca.fit_transform(X)

# Initialize MLP Classifier
mlp = MLPClassifier(
    hidden_layer_sizes=(nohn,),
    solver='sgd',
    activation='tanh',
    batch_size=250,
    max_iter=1000,
    learning_rate_init=0.05,
    momentum=0.9,
    early_stopping=True,
    validation_fraction=0.1,
    verbose=True
)

# Standardize the features after PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_pca)
```

This code performs Principal Component Analysis (PCA) on the training and testing data using Scikit-learn's PCA implementation. It involves the setting of the Number of Principal Components, Initializing and fitting PCA on training Data and transforming training and Testing Data.

PCA is usually used for dimensionality reduction which helps to deal with high dimensional data.

It also includes a parameter which sets a neural network with a specified number of hidden neurons, using the Adam solver and includes other parameters.

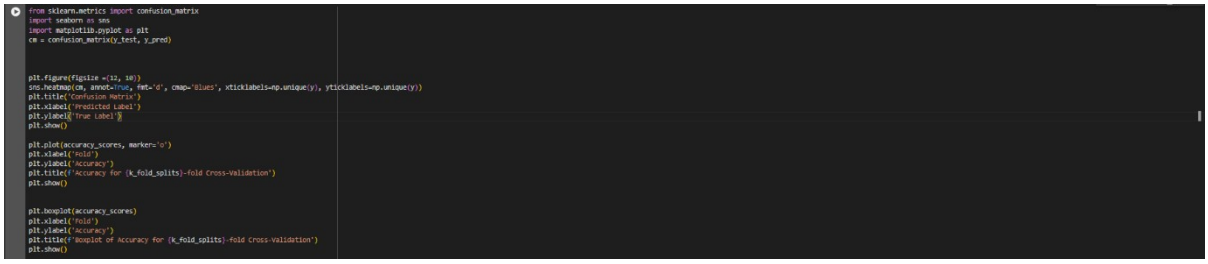
- hidden_layer_sizes: This specifies the number of neurons in the hidden layers.
- solver: This is the optimization algorithm used to update weights.
- activation: This activation function is used in the hidden layers.
- batch_size: The number of samples used in each iteration for updating the weights.
- max_iter: The maximum number of iterations that the neural network uses to be trained.
- learning_rate_init: The initial learning rate for the weights updates.
- momentum: Momentum for gradient descent update. It helps accelerate training and dampen oscillations.
- early_stopping: Whether to use early stopping to terminate training when the validation score is not improving. To prevent overfitting
- validation_fraction: The proportion of training data to set aside as validation set for early stopping.
- verbose: Controls the verbosity of the output during training.

```
[197]
    # Number of splits for k-fold cross-validation
    k_fold_splits = 5

    # Initialize KFold with shuffling
    kf = KFold(n_splits=k_fold_splits, shuffle=True, random_state=42)

    # Lists to store accuracy scores for each fold
    accuracy_scores = []

    # Perform k-fold cross-validation
    for train_index, test_index in kf.split(X_scaled):
        X_train, X_test = X_scaled[train_index], X_scaled[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Train the MLP model
        mlp.fit(X_train, y_train)

        # Predict on the test set
        y_pred = mlp.predict(X_test)

        # Calculate accuracy and store it
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_scores.append(accuracy)

    # Calculate the average accuracy across all folds
    average_accuracy = np.mean(accuracy_scores)

    print(f'Average Accuracy across {k_fold_splits}-fold cross-validation: {average_accuracy}')
```

It utilizes the GPU acceleration to create a scikit-learn pipleline, where data passes through PCA and then through the MLP Classifier. This code also initializes a RandomizedSearchCV object and performs randomized searches over the hyperparameter space with the number of iterations you want, 5-fold cross-validation, using accuracy as the scoring metric. It also runs in parallel (n_jobs=-1) for faster execution.

In k-fold cross validation, the dataset is divided in ‘k’ number of folds with are equally sized. The model is trained and evaluated ‘k’ times using different folds as the test set and the remaining data as the training set and the results are then averaged over the ‘k’ iterations.

It then fits the object to the training data, searching for the most optimal hyperparameters.



```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_pred)



plt.figure(figsize =(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

plt.plot(accuracy_scores, marker='o')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title(f'Accuracy for {k_fold_splits}-fold Cross-Validation')
plt.show()


plt.boxplot(accuracy_scores)
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title(f'Boxplot of Accuracy for {k_fold_splits}-fold Cross-Validation')
plt.show()
```

This code snippet uses ‘seaborn’ and ‘metplotlib’ to create a heatmap visualization of the confusion matrix. Each cell in the matrix represents the number of instances where a true class (y-axis) was predicted as a particular predicted class (x-axis). The colour intensity indicates the frequency.

As for the plotting accuracy scores and the boxplot for the accuracy score.

3. Experiments

Changing Hidden Neurons Table

Using:

mlp = MLPClassifier(hidden_layer_sizes=(nohn,), solver='sgd', activation='tanh', batch_size=256, max_iter=1000, learning_rate_init=0.05, momentum=0.9, early_stopping=True, validation_fraction=0.1, verbose=True) and PCA= 200

Neurons Trials	1	50	75	100	500	1000	1500	1501
1	0.96	0.96	0.96	0.95	0.96	0.96	0.95	0.96
2	0.93	0.93	0.94	0.94	0.94	0.93	0.93	0.92
3	0.96	0.95	0.97	0.94	0.95	0.97	0.96	0.95
4	0.95	0.96	0.95	0.95	0.97	0.95	0.96	0.95
5	0.96	0.95	0.96	0.94	0.94	0.96	0.96	0.95
Average	**0.95**	**0.95**	**0.96**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**

Changing the component: PCA Table

Using:

mlp = MLPClassifier(hidden_layer_sizes=(nohn,), solver='sgd', activation='tanh', batch_size=256, max_iter=1000, learning_rate_init=0.05, momentum=0.9, early_stopping=True, validation_fraction=0.1, verbose=True) and nohn= 200

PCA Trials	1	50	100	500	1000
1	0.03	0.94	0.96	0.80	0.33
2	0.06	0.95	0.95	0.81	0.35
3	0.04	0.96	0.96	0.77	0.37
4	0.02	0.95	0.96	0.83	0.38
5	0.04	0.95	0.98	0.78	0.32
Average	**0.04**	**0.95**	**0.96**	**0.80**	**0.33**

I noticed that changing the parameters manually is inefficient as there can be over 10,000 different combinations of different parameters that have to be run to get the most optimal set of parameters. The tables above are a nice starting point to get the most optimal set of parameters, but I had a better idea. And that better idea was to use Random search which runs a set number of combinations of assigned parameters at random to get the most optimal set.

4. Random Search

Random Search Optimization 1

```
[CV] END mlp__activation=tanh, mlp__batch_size=64, mlp__hidden_layer_sizes=(50,), mlp__solver=sgd, pca__n_components=None; total time=   7.5s
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
[CV] END mlp__activation=tanh, mlp__batch_size=64, mlp__hidden_layer_sizes=(50,), mlp__solver=adam, pca__n_components=50; total time=   2.6s
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
[CV] END mlp__activation=tanh, mlp__batch_size=64, mlp__hidden_layer_sizes=(50,), mlp__solver=adam, pca__n_components=50; total time=   2.2s
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

The code worked but it was not optimized as some of the Hyper parameters used could not yield the desired results due to the maximum number of iterations being too small. The iterations were forced to end before they could converge.

To solve this problem, I added the "mlp_max_iter" parameter which was used to increase the maximum number of iterations possible which helps to be optimized.

From this:

```
# Define the parameter grid for grid search
param_dist = {
    'pca__n_components': [50, 100, 200, 400, 600, 800, None],
    'mlp__hidden_layer_sizes': [(100,), (200,), (400,), (600,), (800,), (1000,)],
    'mlp__activation': ['tanh', 'relu', 'logistic'],
    'mlp__solver': ['sgd', 'adam'],
    'mlp__batch_size': [256],
    'mlp__learning_rate_init': [0.01, 0.05],
    'mlp__momentum': [0.9, 0.5],
    'mlp__early_stopping': [True],
    'mlp__validation_fraction': [0.1, 0.01],
}
```

To this:

```
# Define the parameter grid for grid search
param_dist = {
    'pca__n_components': [50, 100, 200, 400, 600, 800, None],
    'mlp__hidden_layer_sizes': [(100,), (200,), (400,), (600,), (800,), (1000,)],
    'mlp__activation': ['tanh', 'relu', 'logistic'],
    'mlp__solver': ['sgd', 'adam'],
    'mlp__batch_size': [256],
    'mlp__max_iter': [1000],
    'mlp__learning_rate_init': [0.01, 0.05],
    'mlp__momentum': [0.9, 0.5],
    'mlp__early_stopping': [True],
    'mlp__validation_fraction': [0.1, 0.01],
}
```

Random Search Optimization 2

In this experiment, I was suddenly curious about the runtime of the whole code, so I added a code that displays the runtime after the entire code is done.

```
# Start measuring time
start_time = time.time()
```

```
# End measuring time
end_time = time.time()
runtime = end_time - start_time
print(f"Total Runtime: {runtime} seconds")
```

Which Displays this:

```
#End Measuring Time
end_time = time.time()
runtime = end_time - start_time
print(f"  Total Runtime: {runtime} seconds")
```

```
Total Runtime: 3785.2574212551117 seconds
```

Random Search Optimization 3

The code took too long to run, so instead of the CPU, I used the GPU, which is 110 times faster than the CPU.

I added this code, which checks for GPU availability and sets up a GPU strategy if running on GPU. Otherwise, it informs you that it is not running on a GPU and continues with CPU operations, depending on your environment.

```
import tensorflow as tf
print("Tensorflow version " + tf.__version__)

# Check if GPU is available
if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("No GPU available. Please ensure you have configured GPU acceleration in your Colab runtime.")
```

```
Tensorflow version 2.15.0
Default GPU Device: /device:GPU:0
```

i also added a code to upload the data to Collab and use the data from there since it makes the code runway faster than using the code directly from Google Drive

```
[3] from google.colab import drive
    import numpy as np

    # Mount Google Drive
    drive.mount('/content/drive')

    # Define the path to the files on Google Drive
    drive_path = '/content/drive/My Drive/'
    y_filename = 'yaleExtB_target.npy'
    X_filename = 'yaleExtB_data.npy'

    # Copy files from Google Drive to Colab
    !cp "{drive_path}{y_filename}" .
    !cp "{drive_path}{X_filename}" .

    # Load data
    y = np.load(y_filename)
    X = np.load(X_filename)

    # Display confirmation message
    print(f'Data loaded successfully.\nTarget shape: {y.shape}, Data shape: {X.shape}')
```

And I encompassed the part of the code that takes the longest with an indicator that forces the code to use GPU for that part of the code, using “with tf.device('/device:GPU:0'):”

```
def gpu():
  with tf.device('/device:GPU:0'):
    # Create the PCA object
    pca = PCA(whiten=True)

    # Create the MLP classifier
    mlp = MLPClassifier()

    # Create the pipeline
    pipeline = Pipeline([
        ('pca', pca),
        ('mlp', mlp)
    ])

    # Create the random search object
    random_search = RandomizedSearchCV(pipeline, param_distributions=param_dist, n_iter=10, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)

    # Fit the random search to the data
    random_search.fit(X_train, y_train)

    # Print the best hyperparameters
    print("Best Hyperparameters:", random_search.best_params_)

    # Save the best model to disk
    best_model = random_search.best_estimator_
    joblib.dump(best_model, 'best_model_with_pca_random_search.joblib')

    # Evaluate the model with the best hyperparameters on the test set
    y_pred = best_model.predict(X_test)

    # Print classification report
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
gpu()
```

And lastly I added the ‘n_jobs’ settings and set it to -1 so that it can use all available CPU cores, leading to faster execution

```
# Create the random search object
random_search = RandomizedSearchCV(pipeline, param_distributions=param_dist, n_iter=100, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)
```

Random Search Code explanation.

```
[10] from google.colab import drive
     drive.mount('/content/drive')
```

This part of the code gives this notebook access to the Google Drive that you choose

```
from numpy import load
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
import seaborn as sns
import matplotlib.pyplot as plt
import joblib
import time
from tqdm import tqdm
from sklearn.utils import parallel_backend
```

This code imports the necessary Libraries and Modules needed for this machine Learning Project which includes NumPy for data processing, Random search for Model selection and tuning, Confusion Matrix for metrics evaluation, PCA for dimensionality reduction, MLP Classifier for the neural network, Joblib for model serialization, Pipeline for pipeline creation and seaborn/matplotlib.pyplot for visualization.

```
[12] from google.colab import drive
     import numpy as np

     # Mount Google Drive
     drive.mount('/content/drive')

     # Define the path to the files on Google Drive
     drive_path = '/content/drive/My Drive/'
     y_filename = 'yaleExtB_target.npy'
     X_filename = 'yaleExtB_data.npy'

     # Copy files from Google Drive to Colab
     !cp "{drive_path}{y_filename}" .
     !cp "{drive_path}{X_filename}" .

     # Load data
     y = np.load(y_filename)
     X = np.load(X_filename)

     # Display confirmation message
     print(f'Data loaded successfully.\nTarget shape: {y.shape}, Data shape: {X.shape}')
```

This code mounts the relevant Google Drive to your Google Collab, uploads the files 'yaleExtB_target.npy' and 'yaleExtB_data.npy' from Google Drive to Google Collab and then loads the uploaded data directly to the variables y and X.

```
[ ] import tensorflow as tf
    print("Tensorflow version " + tf.__version__)
    #check if GPU is available
    if tf.test.gpu_device_name():
      print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
    else:
      print("No GPU available. Please ensure you have configured GPU acceleration in your Colab runtime.")

    Tensorflow version 2.15.0
    Default GPU Device: /device:GPU:0
```

This code snippet is used to check for available GPUs and inform the user of the GPU type and Tensorflow version is the GPU is available.

This code splits the data into training and testing sets with will be used for the machine learning.

This defines a parameter ‘dist’ which will be used for the Random search. It helps to search through different hyperparameter combinations for the most optimal set, used to fine-tune the model.

It uses all combinations of the parameters using cross-validation.

This is used to track the execution time of the code.

Using the function ‘gpu,’ I use TensorFlow with GPU and sci-kit-learn that is used to perform a Random search with the PCA and MLP classifier.

Which will then perform the Random search and print out the most optimal Parameters which I will use for further analysis.

This is used to stop the time tracking and output the time spent from the start to the end of the Random Search

Random search Results:
Running the Random search I got these parameters as the most optimal for this ANN

Nohn = 500

PCA = 200

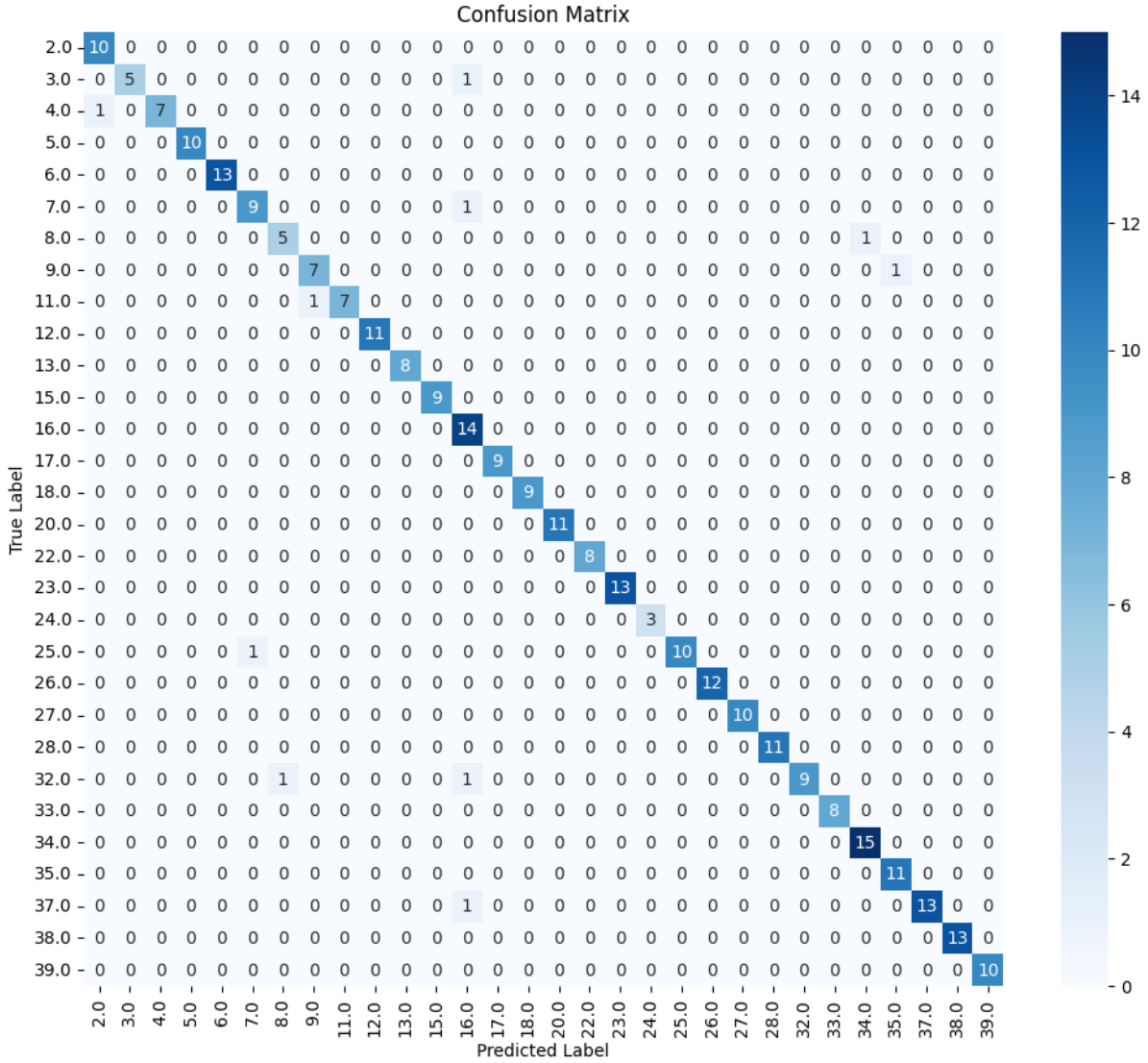
```
{
    hidden_layer_sizes=(nohn,),
    solver='sgd',
    activation='tanh',
    batch_size=256,
    max_iter=1000,
    learning_rate_init=0.05,
    momentum=0.9,
    early_stopping=True,
    validation_fraction=0.1,
    verbose=True
}
```

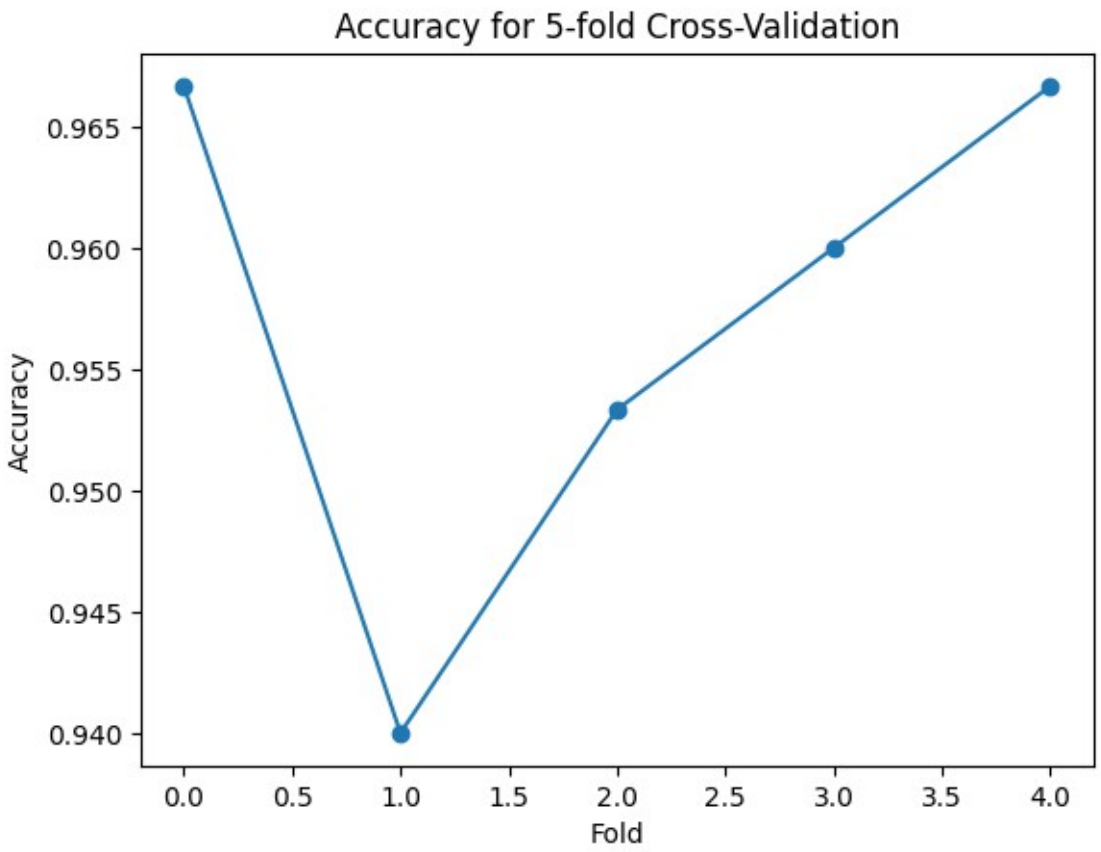
And got an average of 0.96 (96% Accuracy)

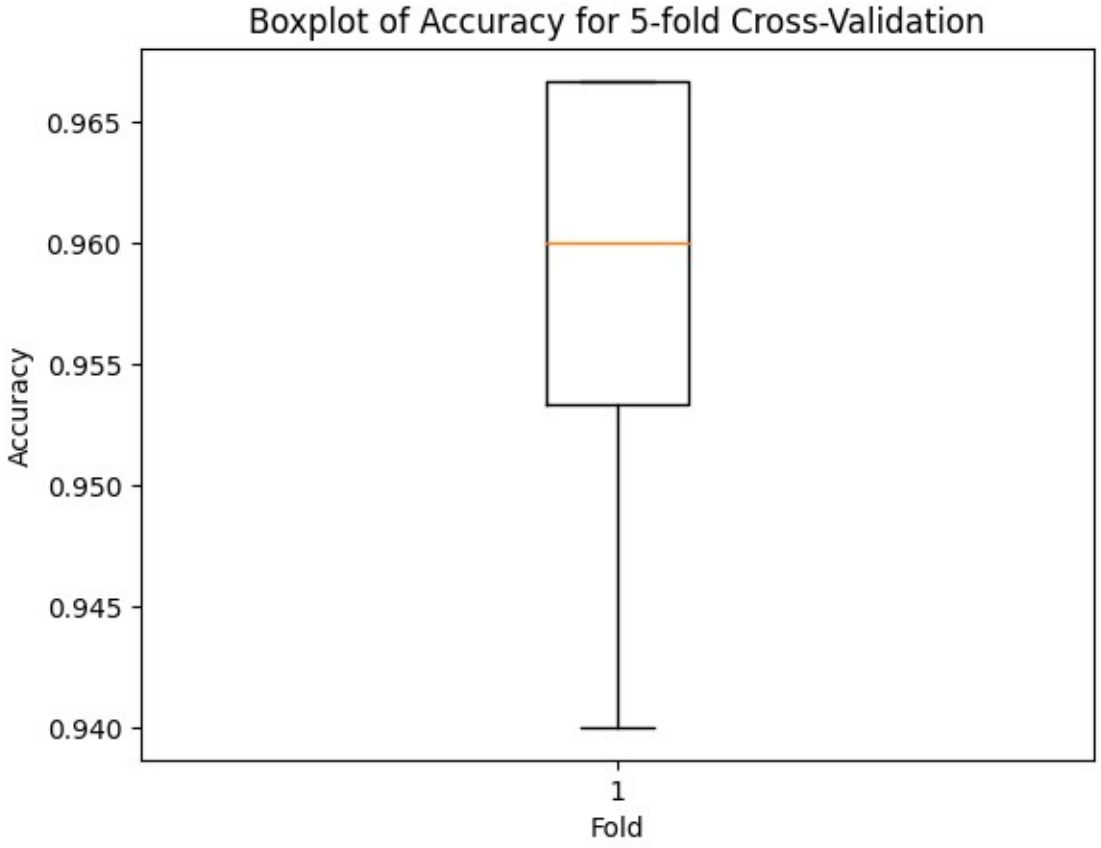
Classification Report:

	precision	recall	f1-score	support
2.0	0.91	1.00	0.95	10
3.0	1.00	0.83	0.91	6
4.0	1.00	0.88	0.93	8
5.0	1.00	1.00	1.00	10
6.0	1.00	1.00	1.00	13
7.0	0.90	0.90	0.90	10
8.0	0.83	0.83	0.83	6
9.0	0.88	0.88	0.88	8
11.0	1.00	0.88	0.93	8
12.0	1.00	1.00	1.00	11
13.0	1.00	1.00	1.00	8
15.0	1.00	1.00	1.00	9
16.0	0.78	1.00	0.88	14
17.0	1.00	1.00	1.00	9
18.0	1.00	1.00	1.00	9
20.0	1.00	1.00	1.00	11
22.0	1.00	1.00	1.00	8
23.0	1.00	1.00	1.00	13
24.0	1.00	1.00	1.00	3
25.0	1.00	0.91	0.95	11
26.0	1.00	1.00	1.00	12
27.0	1.00	1.00	1.00	10
28.0	1.00	1.00	1.00	11
32.0	1.00	0.82	0.90	11
33.0	1.00	1.00	1.00	8
34.0	0.94	1.00	0.97	15
35.0	0.92	1.00	0.96	11
37.0	1.00	0.93	0.96	14
38.0	1.00	1.00	1.00	13
39.0	1.00	1.00	1.00	10
accuracy			0.97	300
macro avg	0.97	0.96	0.97	300
weighted avg	0.97	0.97	0.97	300

Average Accuracy across 5-fold cross-validation: 0.9573333333333334


Confusion Matrix
True Label
Predicted Label
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 11.0 12.0 13.0 15.0 16.0 17.0 18.0 20.0 22.0 23.0 24.0 25.0 26.0 27.0 28.0 32.0 33.0 34.0 35.0 37.0 38.0 39.0
0 2 4 6 8 10 12 14


Accuracy for 5-fold Cross-Validation
Accuracy
0.965
0.960
0.955
0.950
0.945
0.940
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Fold


Boxplot of Accuracy for 5-fold Cross-Validation
Accuracy
0.965
0.960
0.955
0.950
0.945
0.940
1
Fold

Conclusion

In conclusion, we were told to develop a Machine Learning (ML) solution to solve a biometric recognition task with the highest recognition accuracy. The only part of the assignment that was unclear to me was the experiment and the table. None of the weeks and lectures were difficult to follow and none of the slides were unclear. This assignment helped me to learn a lot more about Maching Learning and it made me appreciate AI and its capabilities.

As for the experiments that were conducted, using traditional Multilayer Perceptron (MLP) models for biometric recognition yielded valuable insights and noteworthy outcomes.

Experiment 1 - Choice of MLP Models:

When choosing MLP Models, I had to decide between using a Keras Framework Model or a traditional Multilayer Perceptron Model. So, I ran both and then noticed that the Keras Framework Model took a toll on the GPU that Google Collab provided for me, and the total time used to run it was way more than the total time used to run the traditional Multilayer Perception model which influenced my decision to choose the traditional Multilayer Perception model. Although on the second assignment I aim to improve and optimize my script drastically.

Experiment 2 - K-fold Cross Validation:

K-fold cross validation was a great help to prevent overfitting, I noticed I was getting the same high accuracy despite me changing the parameters drastically, so I decided to use the K-fold Cross Validation with the value of ‘k’ being 5. And although the use of the K-fold Cross Validation made my script run slower, it was necessary as it help to prevent the overfitting problem I had earlier. This was a great start to help prevent the overfitting of my script.

Experiment 3 - Scripts Implementation:

As you can see in the report, I initially created two scipts. ‘Process_yale_images’ to set up the images, training data and target data, and ‘classify_yale’ to train and test the data to get the highest accuracy for the Machine solution. But as I tried to improve the code further, I noticed that the manual input of different parameters was ineffiecient for the finding of the most optimal parameter. So, at first I used Gridsearch, but it really took a toll on the GPU and CPU finding the most optimal parameters by running all possible combinations, and also it brought back different results each time ran which shows that there can be multiple combinations of parameters that shows the same result. So, I decided to use Random Search instead, which was easier to optimize and faster to run.

By delving into the specifics of each experiment and providing detailed observations, this extended conclusion aims to offer a more comprehensive understanding of the undertaken work. The insights gained from these experiments contribute significantly to the overall success of the machine learning solution.

Plan on how my work can be extended to research paper.

A report, on being made an extended research paper, normally follows an organized format, which is also pursued in common respective academic papers/research reports. Here is a recommended outline of some of the ideas that can be incorporated into the extended version of my report.

Title: I should be able to create a clear and concise title that reflects the focus of this report without it being too complicated and complex.

Abstract: I should summarize the major objectives, methods, and key results of your study as they relate to the task given to me. It should be clear on why this report was made and what we aim to achieve with the creation of this report. (Machine Learning).

Introduction: I should be able to introduce the overall problem of biometric recognition in the real world, hence show a question, and objectives and tasks of the research, and demonstrate the significance and relevance to the ANN. As well as explain how I can use ANN to achieve the aim and objective.

Literature Review: Further elaboration to literature review should be made to include more relevant studies. The study should look for different kinds of studies on the approaches to biometric recognition and be able to show the strengths, give the weaknesses, give a broader review, and most necessarily review the current research on ANNs, DL, and challenges of face recognition.

Methodology: I should explain the way an ML solution is developed. They should touch more about the details concerning traditional MLP models, data preprocessing steps, and reasoning in the selection of parameters. Briefly, it is necessary to describe the access to Google Collab and the usage of GPU resources.

Data Collection and Processing: Explain Source, Size, and Characteristics of Pertinent Data Set - An elaborate, accurate explanation of the data set, including the source, size, and characteristics, should follow. And be able to elaborate on the script "Process_Yale_images" and traces in the data.

Model Implementation: My report is expected to cover much more explanation on the Classify_Yale script, including the choice of MLP models and what impact might come from varying parameters on the accuracy that might be gotten. It is meant to explain what PCA is and how it influences dimension reduction.

Experiments and Results: That is, the section will be extended to more detailed information about the experiments, all coupled with insights on the rationale for the parameter choices and their impact on this class of model performance.

Random Search Optimization: This report also fully elaborates on optimization with all steps involved, giving discussions of relationships of challenges, improvements effected, and a more detailed description of Random Search in its place in the scour for optimal hyperparameters.

Discussion: I am also proposing to discuss the implications the results have on biometric recognition and the limitations encountered in carrying out the research and giving suggestions on potential areas of future research from my findings.

Conclusion: This section provides an overall summary of the main findings and contributions from the research, including the importance of the ML solution within the scope of biometric recognition.

References: I should expand the reference section, including more relevant studies and literature.

Acknowledgments: If applicable, acknowledge any individuals or organizations that supported your research.

Appendices: Include any supplementary material, such as additional charts, graphs, or detailed code snippets.

Through elaboration of the sections and more information on the same and being tested further, my report can then be turned to a research paper.

5. References

- Selitskaya, N., Selitskiy, S., & Christou, N. (2021). *Challenges in Real-Life Face Recognition with Heavy Makeup and Occlusions Using Deep Learning Algorithms. In Machine Learning, Optimization, and Data Science. LOD 2020.* Lecture Notes in Computer Science (pp. 600-611). Springer Nature. **DOI: 10.1007/978-3-030-64580-9_49.**