National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Arundev Vamadevan
Student ID: x22144421

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arundev Vamadevan
Student ID:	x22144421
Programme:	MSc Data Analytics **Year:** 2022-23
Module:	MSc Research Project
Lecturer:	Dr. Christian Horn
Submission Due Date:	14 December 2023
Project Title:	Person Identification Using Landmarks and Deep Learning Techniques
Word Count:	..... **Page Count:** 9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: .....

Date: .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Arundev Vamadevan
Student ID: x22144421

1 Introduction

This configuration manual contains details about hardware and software requirements and specifications used to develop various models and the person identification system. The below sections will walk through the steps to follow to setup minimal running environment along with different applications to be downloaded and packages to be installed.

2 System Specification

For the implementation of the system, Python 3.9.13 is used. Other libraries and frameworks are,

IDE	: Jupyter Notebook
Computation	: CPU
Modules	: Numpy, Pandas, Matplotlib, Scikit-learn, CV2
Frameworks	: Tensorflow, Keras
Softwares	: MS Excel

Hardware Specifications: Minimum RAM 8GB RAM is required to provide required results. But 16 GB or more is recommended. Figure 1 depicts the hardware and system configuration used for the project.

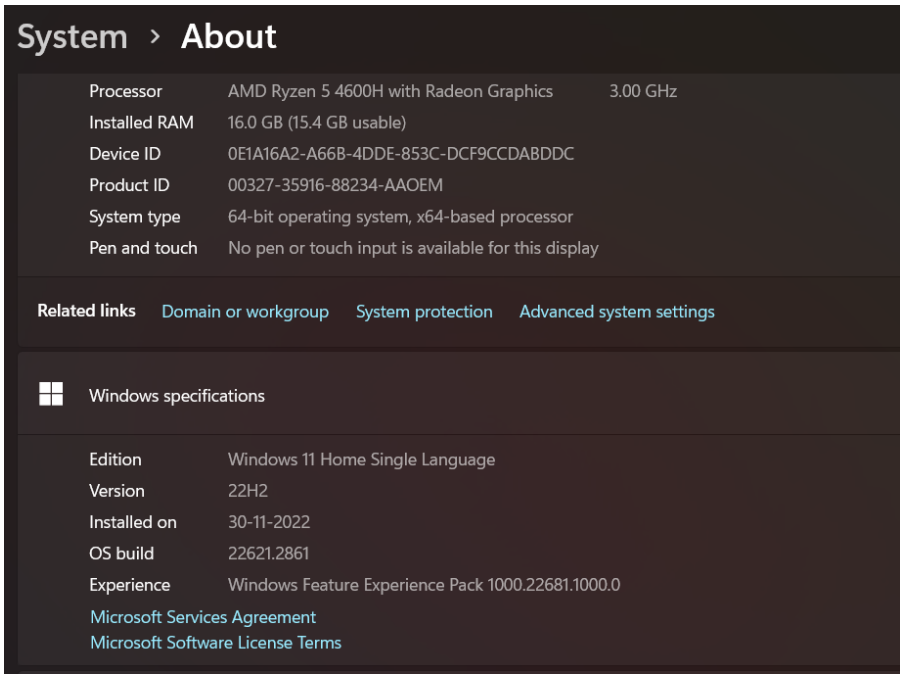

Fig 1 : System Configuration

3 Dataset Description

This research uses the dataset, VGG FACE-2. It was introduced by (Cao, 2018). The dataset consists of around 3.31 million of images of 9131 persons with an approximate average of 362 samples for each person. The dataset is in compressed format and is about 40 GB in size. It can be downloaded from the website https://academictorrents.com./

Dataset also contains facial landmark coordinates in a csv file.

The research used 6402 images of 20 personalities. The folders used are as below

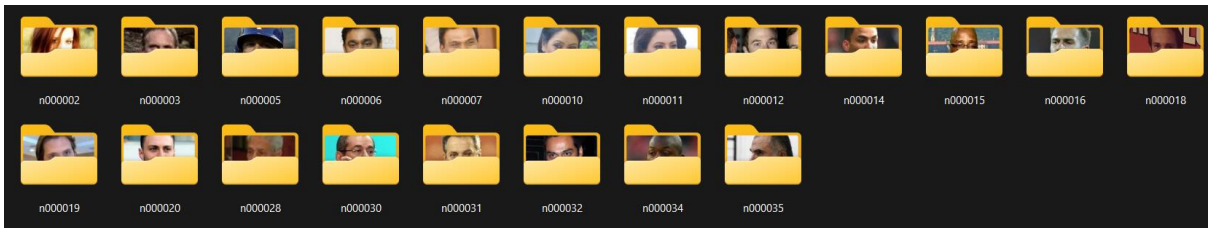

Fig 2: 20 folders used in research
The 'Data' Folder in the ICT code artefact zipped folder contains all the 20 folders, which the jupyter notebook is accessing for processing and corresponding csv files.

```
]: #LOAD THE IMAGES
image_path1=os.listdir("Data/faces_20/")
image1=[]
for i in image_path1:
    image1.append(os.path.join("Data/faces_20/",i))
image1
```

```
]: ['Data/faces_20/n000002',
 'Data/faces_20/n000003',
 'Data/faces_20/n000005',
 'Data/faces_20/n000006',
 'Data/faces_20/n000007',
 'Data/faces_20/n000010',
 'Data/faces_20/n000011',
 'Data/faces_20/n000012',
 'Data/faces_20/n000014',
 'Data/faces_20/n000015',
 'Data/faces_20/n000016',
 'Data/faces_20/n000018',
 'Data/faces_20/n000019',
 'Data/faces_20/n000020',
 'Data/faces_20/n000028',
 'Data/faces_20/n000030',
 'Data/faces_20/n000031',
 'Data/faces_20/n000032',
 'Data/faces_20/n000034',
 'Data/faces_20/n000035']
```

```
]: #LOAD LANDMARK DATASET
landmarks=pd.read_csv("Data/Landmarks_20.csv")
landmarks
```

Fig 3 : Accessing folders and landmarks

4 Project Development and Implementation

After installing all the required software and downloading dataset, open a new notebook in Jupyter Notebook. Then load and open the code file and execute as per the need. Execution can be done by clicking Run All to run all cells simultaneously or individually cell by cell.

4.1 Importing Library

The packages needed to run the file should be pre-installed using the command “pip install ‘pakcage-name’”

```
#IMPORT NESSESSORY LIBRARIES
import os
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from glob import glob
from skimage import io
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from tensorflow.keras.preprocessing.image import load_img
```

```
import os
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

```
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix

import joblib
from mtcnn.mtcnn import MTCNN
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import load_model
from sklearn.preprocessing import MinMaxScaler
```

Fig 4: Various python packages used in the project

4.2 Pre-processing

```
#PREPROCESSING
import cv2
import numpy as np
data=landmarks

df=pd.DataFrame(data,columns=['NAME_ID', 'P1X', 'P1Y', 'P2X', 'P2Y', 'P3X', 'P3Y', 'P4X', 'P4Y','P5X', 'P5Y'])
# Define dimensions for resize
target_height, target_width = 128, 128

# Function to preprocess an image and annotation
def preprocess_image_and_annotation(row):
    image_path = row["NAME_ID"]
    print(image_path)
    P1X,P1Y,P2X, P2Y,P3X,P3Y,P4X,P4Y,P5X,P5Y=row["P1X"], row["P1Y"], row["P2X"],row["P2Y"], row["P3X"], row["P3Y"], row["P4X"], r

    # Load the image using OpenCV
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    original_height, original_width = image.shape[0], image.shape[1]
    # Resize the images to the target dimensions
    image = cv2.resize(image, (target_width, target_height))

    # Scale annotation coordinates
    P1X = int((P1X/original_width) * target_width)
    P1Y = int((P1Y/original_height) * target_height)
    P2X = int((P2X/original_width) * target_width)
    P2Y = int((P2Y/original_height) * target_height)
    P3X = int((P3X/original_width) * target_width)
    P3Y = int((P3Y/original_height) * target_height)
    P4X = int((P4X/original_width) * target_width)
    P4Y = int((P4Y/original_height) * target_height)
    P5X = int((P5X/original_width) * target_width)
    P5Y = int((P5Y/original_height) * target_height)


    # Normalize the image to values between 0 and 1
    image = image / 255.0

    return image, (P1X,P1Y,P2X, P2Y,P3X,P3Y,P4X,P4Y,P5X,P5Y)
# Preprocess the images and annotations
preprocessed_data = df.apply(preprocess_image_and_annotation, axis=1)

# Create a NumPy array of images with the same shape
preprocessed_images = np.array([item[0] for item in preprocessed_data], dtype=np.float32)

# Now, preprocessed_images is a NumPy array containing all the images with the same shape.
```

Fig 5 : Preprocessing images and landmarks

Preprocessing of images includes, resizing, normalizing.

```
# IMAGE AUGMENTATION IN PREPROCESSED IMAGES
# Function for image flipping and landmark flipping
def apply_augmentation(image, landmarks_rec):
    # Image Flipping
    img = image
    flipped_img = np.fliplr(img)

    # Landmarks Flipping

    original_image_width = 128
    landmarks_new = landmarks_rec.reshape((-1,1,2))
    for i in landmarks_new:
        i[0][0]= original_image_width - i[0][0]

    return flipped_img, landmarks_new
```

```
# Apply data augmentation to each image and its corresponding landmarks
def augment_data(images, landmarks):
    augmented_images = []
    augmented_landmarks = []

    for img, landmark in zip(images, landmarks):
        aug_img, aug_landmark = apply_augmentation(img, landmark)
        augmented_images.append(aug_img)
        augmented_landmarks.append(aug_landmark)

    return np.array(augmented_images), np.array(augmented_landmarks)
```

Fig 6: Image Augmentation

4.3 Modelling

4.3.1 VGG-16 Model

```
# Split the data into training and testing set using sklearn.
x_train,x_test,y_train,y_test = train_test_split(x_train_combined,y_train_combined,train_size=0.8,random_state=42)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

```
# Load the VGG16 model pre-trained on ImageNet data
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
```

```
#Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Create a Sequential model
model = Sequential()

# Add the VGG16 base model to the Sequential model
model.add(base_model)

# Flatten the output of the VGG16 base model
model.add(Flatten())

# Add a dense layer for regression
model.add(Dense(512, activation='relu'))
model.add(Dense(10)) # Assuming you have one continuous output
```

```
# Compile the model with Mean Squared Error loss for regression
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['mean_squared_error'])
```

```
# Train the model
batch_size = 64
epochs = 40
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test))
```

Fig 7 : Model building with VGG-16

4.3.2 Model Building using SVM for person identification with landmarks

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
print(len(X_train),len(y_train))
```

Hyper Parameter Tuning with SVC Model

```
param_grid = {
    'C':[0.1,1,10,100],
    'gamma': [0.001,0.01,0.1,1],
    'kernel' : ['rbf']
}
```

```
# Create an SVC model and train it
model = SVC()
```

```
grid_search = GridSearchCV(model, param_grid, cv=2, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=2, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1],
                         'kernel': ['rbf']})
```

```
best_params = grid_search.best_params_
print("Best Hyperparameters:",best_params)
```

```
Best Hyperparameters: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
best_model = grid_search.best_estimator_
# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Fig 8 : Model building using SVM with landmarks

4.3.3 Model building using SVM for person identification using Images

Training with SVM Model

```
# Create an SVC model and train it
model = SVC(C = 10, gamma = 0.001, kernel= 'rbf')
```

```
model.fit(X_train, y_train)
```

```
SVC(C=10, gamma=0.001)
```

```
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 70.60%
```

Fig 9 : Model building using SVM with images

References

Cao, Q., Shen, L., Xie, W., Parkhi, O.M. and Zisserman, A., 2018, May. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)* (pp. 67-74). IEEE.