# Configuration Manual

MSc Research Project
MSc. Data Analytics

Mayur Said
Student ID: x21118515

School of Computing
National College of Ireland

Supervisor: Qurrat Ul Ain, PhD

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Mayur Said |
| **Student ID:** | x21118515 |
| **Programme:** | MSc. Data Analytics |
| **Year:** | 2018 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Qurrat Ul Ain, PhD |
| **Submission Due Date:** | 20/12/2018 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 506 |
| **Page Count:** | 6 |

| | |
|---|---|
| **Signature:** | |
| **Date:** | 27th January 2023 |

**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. | ☐ |

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

# Configuration Manual

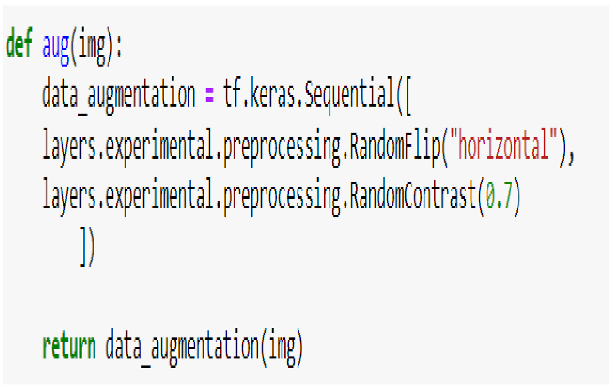
Mayur Said
x21118515

This configuration manual provides an overview of the software and hardware that are necessary to run the code and implement the project. Python codes for data preparation, implementing data preprocessing techniques, visualization, and training deep learning models are written using Jupyter Notebook. Jupyter Notebook can be used by installing Anaconda in the system or with the help of Google Colab.

**Python**: Version 3.7-3.9
**keras_vggface library** [1]: Latest Version
**MTCNN Library**[2]: Latest Version
**OpenCV**: Latest Version
**scikit-learn**: Latest Version
**Tensorflow Library**: Version 2.7.0
**Nvidia Drivers**: Nvidia Drivers for NVIDIA GeForce GTX 1650 Ti
**Nvidia CUDA Toolkit**: Version 11.2
**Nvidia cuDNN**: Version 8.1

## 1 Data Augmentation

To implement data augmentation the code in figure 1 is written in python. Run the code on a particular dataset to apply data augmentation techniques.



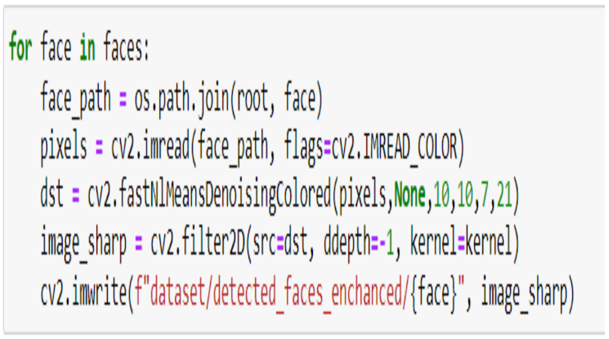
```
def aug(img):
    data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal"),
    layers.experimental.preprocessing.RandomContrast(0.7)
       ])

    return data_augmentation(img)
```

Figure 1: Data Augmentation Code

[1]keras_vggface library `https://github.com/rcmalli/keras-vggface`
[2]MTCNN Library: `https://github.com/ipazc/mtcnn`

## 2 Image Enhancement

To implement image enhancement techniques, a code is written in figure 2. Run the code on a particular image to apply image enhancement techniques.

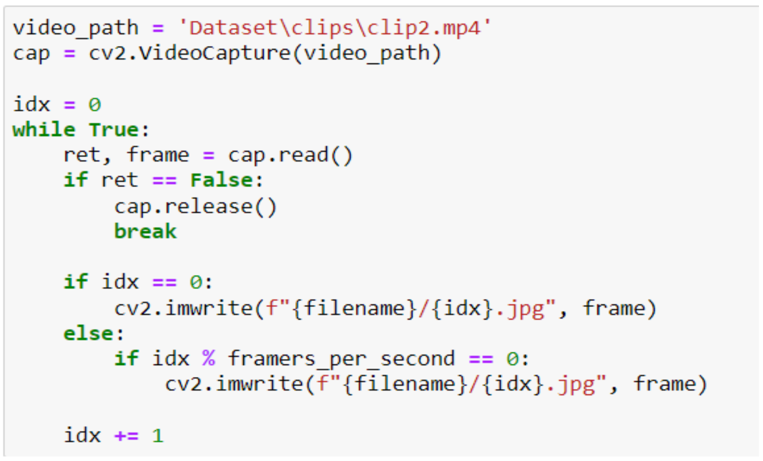


```
for face in faces:
    face_path = os.path.join(root, face)
    pixels = cv2.imread(face_path, flags=cv2.IMREAD_COLOR)
    dst = cv2.fastNlMeansDenoisingColored(pixels,None,10,10,7,21)
    image_sharp = cv2.filter2D(src=dst, ddepth=-1, kernel=kernel)
    cv2.imwrite(f"dataset/detected_faces_enchanced/{face}", image_sharp)
```

Figure 2: Image Enhancement Code

## 3 Capture Frames from video clip

To capture frames from the test video clip, the code in figure 3 is executed.

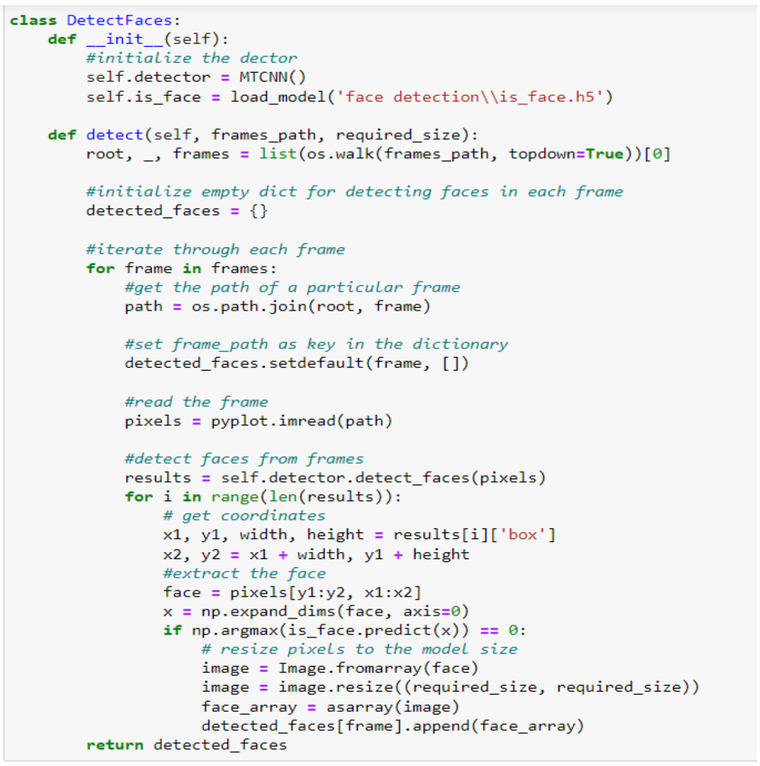


```
video_path = 'Dataset\clips\clip2.mp4'
cap = cv2.VideoCapture(video_path)

idx = 0
while True:
    ret, frame = cap.read()
    if ret == False:
        cap.release()
        break

    if idx == 0:
        cv2.imwrite(f"{filename}/{idx}.jpg", frame)
    else:
        if idx % framers_per_second == 0:
            cv2.imwrite(f"{filename}/{idx}.jpg", frame)

    idx += 1
```

Figure 3: Code to Capture Frames from video clip

## 4 Detect Faces from Capture Video Frame

To detect faces from the captured frames, a python class DetectFaces is written. The code for the same is shown in figure 4. This class is the implementation of a face detection block in the proposed solution. It takes the captured frame as the input and outputs all the faces in that frame. To detect faces, first, instantiate the class DetectFaces and use detect method to detect faces from the input image.



```python
class DetectFaces:
    def __init__(self):
        #initialize the dector
        self.detector = MTCNN()
        self.is_face = load_model('face detection\\is_face.h5')

    def detect(self, frames_path, required_size):
        root, _, frames = list(os.walk(frames_path, topdown=True))[0]

        #initialize empty dict for detecting faces in each frame
        detected_faces = {}

        #iterate through each frame
        for frame in frames:
            #get the path of a particular frame
            path = os.path.join(root, frame)

            #set frame_path as key in the dictionary
            detected_faces.setdefault(frame, [])

            #read the frame
            pixels = pyplot.imread(path)

            #detect faces from frames
            results = self.detector.detect_faces(pixels)
            for i in range(len(results)):
                # get coordinates
                x1, y1, width, height = results[i]['box']
                x2, y2 = x1 + width, y1 + height
                #extract the face
                face = pixels[y1:y2, x1:x2]
                x = np.expand_dims(face, axis=0)
                if np.argmax(is_face.predict(x)) == 0:
                    # resize pixels to the model size
                    image = Image.fromarray(face)
                    image = image.resize((required_size, required_size))
                    face_array = asarray(image)
                    detected_faces[frame].append(face_array)
        return detected_faces
```

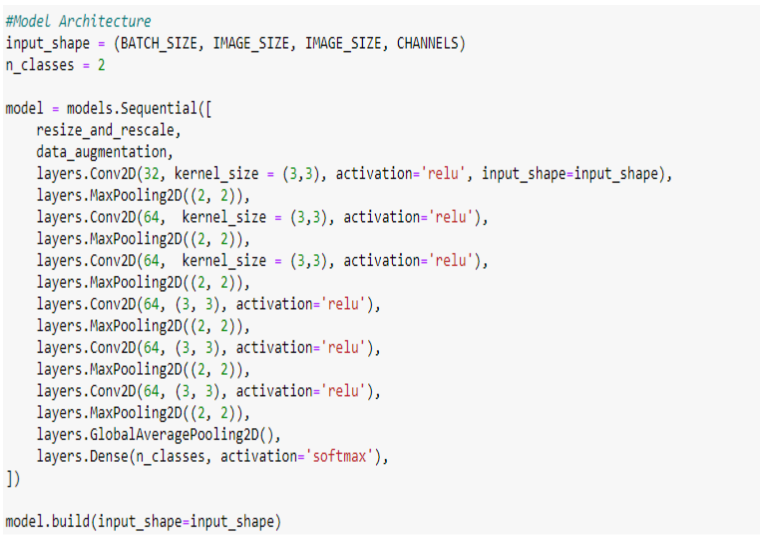
Figure 4: Code to Detect Faces

## 5 Implementation of Custom Convolutional Neural Network Model

To create the custom convolutional model, run the code shown in figure 5. To compile and fit the created model to the dataset run the code in figure 6.

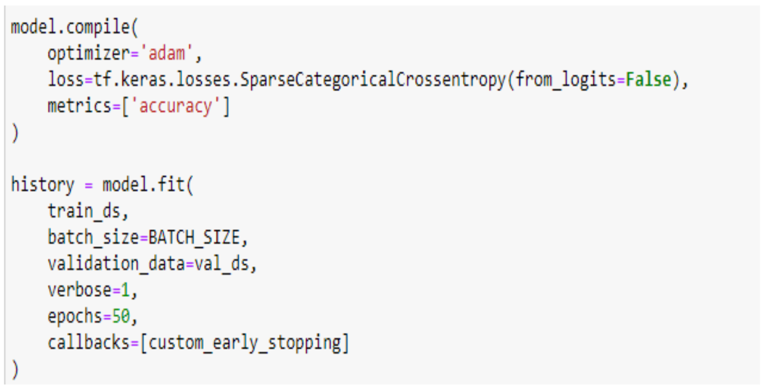
## 6 Fine-tuning ResNet-50

First, freeze the first four stages, get the output layer of the ResNet-50 and then add a global average pooling layer and a dense layer of five neurons on top of it. Then the model is compiled and fitted on the authorised face dataset by running the code in figure 7.



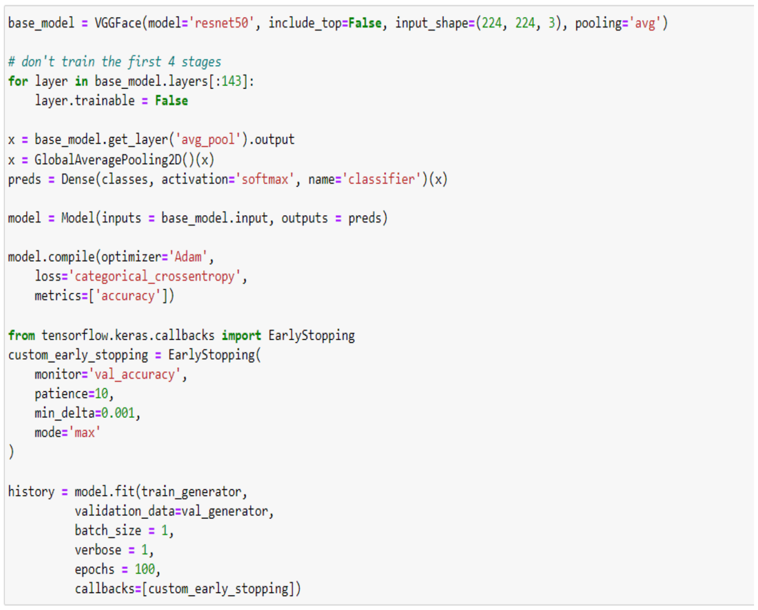
```python
#Model Architecture
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 2

model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

Figure 5: Code to Build CNN Model



```python
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
    callbacks=[custom_early_stopping]
)
```

Figure 6: Code to Compile and Fit CNN Model

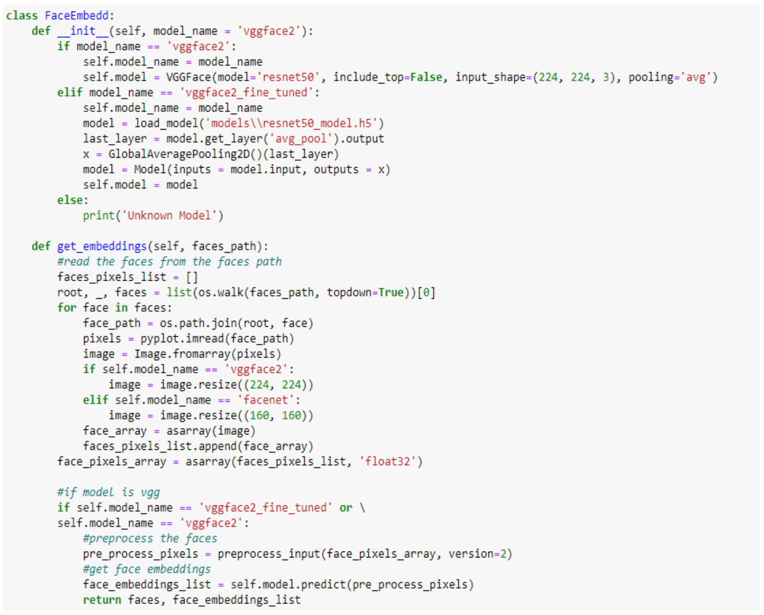


```python
base_model = VGGFace(model='resnet50', include_top=False, input_shape=(224, 224, 3), pooling='avg')

# don't train the first 4 stages
for layer in base_model.layers[:143]:
    layer.trainable = False

x = base_model.get_layer('avg_pool').output
x = GlobalAveragePooling2D()(x)
preds = Dense(classes, activation='softmax', name='classifier')(x)

model = Model(inputs = base_model.input, outputs = preds)

model.compile(optimizer='Adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping
custom_early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=10,
    min_delta=0.001,
    mode='max'
)

history = model.fit(train_generator,
          validation_data=val_generator,
          batch_size = 1,
          verbose = 1,
          epochs = 100,
          callbacks=[custom_early_stopping])
```

Figure 7: Code to Fine-tune ResNet-50 Model

## 7 Predict Face Embedding of the detected faces

To predict face embeddings of a particular face, a python class FaceEmbedd is written. This class FaceEmbedd can predict faces using the ResNet-50 model pre-trained on the VGGFace2 dataset without any fine-tuning as well the fine-tuned version. First, instantiate the class FaceEmbedd with either of the two models and then use the get_embeddings method to predict face embeddings. Execute the code in Figure 8 to implement the same.

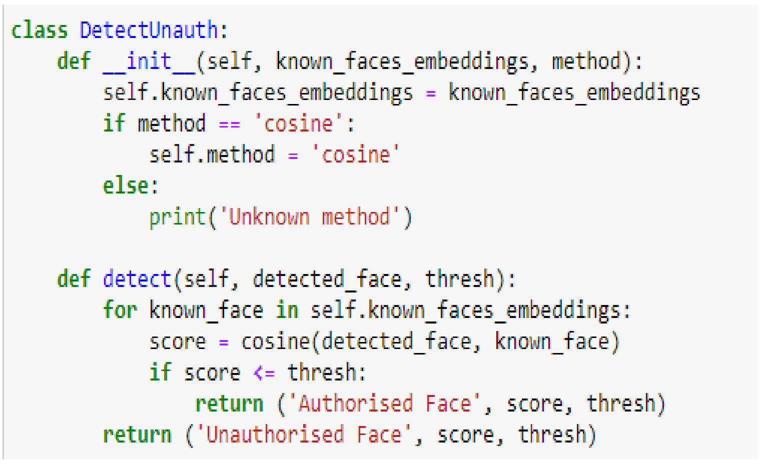


```python
class FaceEmbedd:
    def __init__(self, model_name = 'vggface2'):
        if model_name == 'vggface2':
            self.model_name = model_name
            self.model = VGGFace(model='resnet50', include_top=False, input_shape=(224, 224, 3), pooling='avg')
        elif model_name == 'vggface2_fine_tuned':
            self.model_name = model_name
            model = load_model('models\\resnet50_model.h5')
            last_layer = model.get_layer('avg_pool').output
            x = GlobalAveragePooling2D()(last_layer)
            model = Model(inputs = model.input, outputs = x)
            self.model = model
        else:
            print('Unknown Model')

    def get_embeddings(self, faces_path):
        #read the faces from the faces path
        faces_pixels_list = []
        root, _, faces = list(os.walk(faces_path, topdown=True))[0]
        for face in faces:
            face_path = os.path.join(root, face)
            pixels = pyplot.imread(face_path)
            image = Image.fromarray(pixels)
            if self.model_name == 'vggface2':
                image = image.resize((224, 224))
            elif self.model_name == 'facenet':
                image = image.resize((160, 160))
            face_array = asarray(image)
            faces_pixels_list.append(face_array)
        face_pixels_array = asarray(faces_pixels_list, 'float32')

        #if model is vgg
        if self.model_name == 'vggface2_fine_tuned' or \
        self.model_name == 'vggface2':
            #preprocess the faces
            pre_process_pixels = preprocess_input(face_pixels_array, version=2)
            #get face embeddings
            face_embeddings_list = self.model.predict(pre_process_pixels)
            return faces, face_embeddings_list
```

Figure 8: Code to Predict Face Embeddings

## 8 Predict detected face as ‘Authorised Face’ or ‘Unauthorised Face’

To predict the detected face as an ‘Authorised Face’ or ‘Unauthorised Face’, a python class called DetectUnauth is written. First Instantiate this class using the authorised face embeddings and then use detect method in it to detect face embeddings of a particular face as ‘Authorised Face’ or ‘Unauthorised Face’. The code for the same is shown in Figure 9

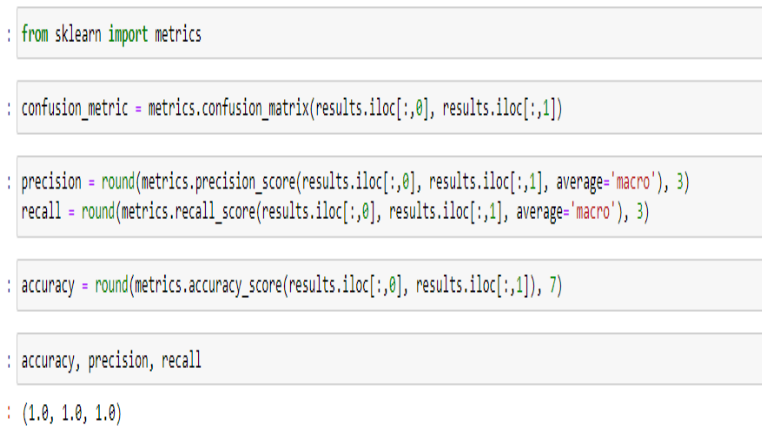


```python
class DetectUnauth:
    def __init__(self, known_faces_embeddings, method):
        self.known_faces_embeddings = known_faces_embeddings
        if method == 'cosine':
            self.method = 'cosine'
        else:
            print('Unknown method')

    def detect(self, detected_face, thresh):
        for known_face in self.known_faces_embeddings:
            score = cosine(detected_face, known_face)
            if score <= thresh:
                return ('Authorised Face', score, thresh)
        return ('Unauthorised Face', score, thresh)
```

Figure 9: Code to Detect a Given Face as ’Authorised Face’ or ’Unauthorised Face’

## 9 Calculate precision, recall, and f1-score

Precision, recall, and f1-score are calculated using the scikit-learn library. Run the code in figure 10 to calculate the same.



```python
from sklearn import metrics

confusion_metric = metrics.confusion_matrix(results.iloc[:,0], results.iloc[:,1])

precision = round(metrics.precision_score(results.iloc[:,0], results.iloc[:,1], average='macro'), 3)
recall = round(metrics.recall_score(results.iloc[:,0], results.iloc[:,1], average='macro'), 3)

accuracy = round(metrics.accuracy_score(results.iloc[:,0], results.iloc[:,1]), 7)

accuracy, precision, recall
```

```
(1.0, 1.0, 1.0)
```

Figure 10: Code to calculate precision, recall, and f1-score