National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shreyansh Gupta
Student ID: x21239347

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shreyansh gupta
Student ID:	x21239347
Programme:	MSc Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Abid Yaqoob
Submission Due Date:	14/12/23
Project Title:	Real Time Driver Drowsiness Detection Based on Deep Learning
Word Count:	1099
Page Count:	10

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:	Shreyansh Gupta
Date:	31st January 2024

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

Shreyansh gupta
x21239347

1 Introduction

This configuration manual aims to guide users in setting up the environment and running the real-time driver drowsiness detection system using Python, OpenCV, TensorFlow, and a pre-trained deep learning model.

2 System Configuration and Requirements

To set up the system for conducting research on eye state detection using machine learning models, consider the following configuration and requirements:

2.1 Hardware configuration

MSI GF63 Thin 9SCSR system has been used. The specifications are –

- Operating System - Microsoft Windows 10 Home Single Language.
- Processor – Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.60 GHz.
- RAM – 8GB, GPU – NVIDIA GeForce GTX 1650 Ti with Max-Q, SSD – 500GB.
- web cam

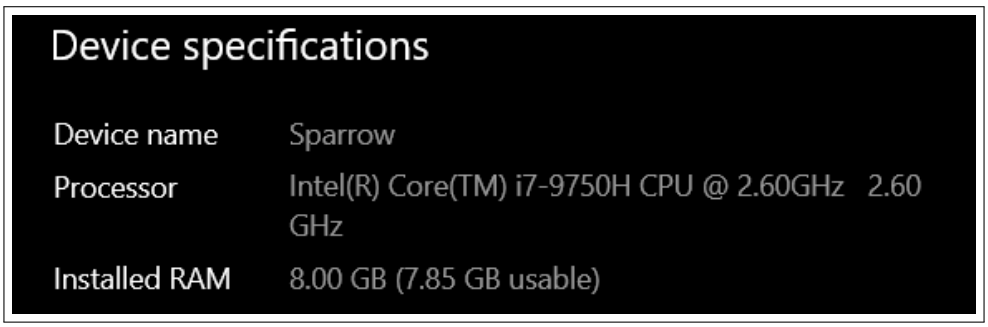

Figure 1: System Configuration

2.2 Software Configuration

- Jupyter Notebook The Jupyter Development Team (2022)serves as the primary graphical user interface (GUI) for developing models.
- Python 3.10 Python Software Foundation (2023)acts as the main programming language within this environment.

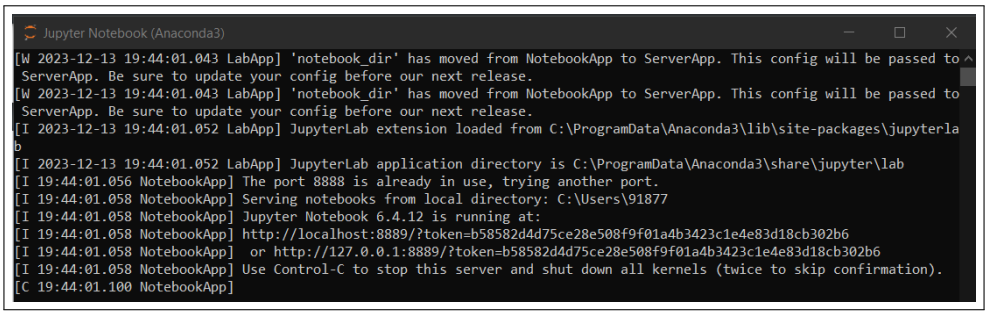

Figure 2: Jupyter Notebook

Additional Software:

- TensorFlow Google LLC (2023) and Keras . (2023a) frameworks for machine learning model development.
- Image processing libraries (OpenCV, Pillow) for image manipulation and preprocessing.
- Pygame for sound-based alerts in real-time testing.

3 Implementation

3.1 Dataset Acquisition:

Obtain the eye dataset from the MRL eye dataset website Vietnam National University (2018) Refer Figure 3

3.2 Feature Extraction

Feature engineering involves preparing and organizing data to enhance the machine learning model's performance.

- **Import libraries** Open a Jupyter Notebook and import necessary libraries to perform data manipulation and file operations for the eye state detection task. Begin by importing essential libraries such as os, shutil, glob, and split_folders to manage file paths, file operations, and dataset splitting. Refer Figure 4
- **Setup Directories:**

  Define the Raw_DIR variable with the directory path where the original images are stored.

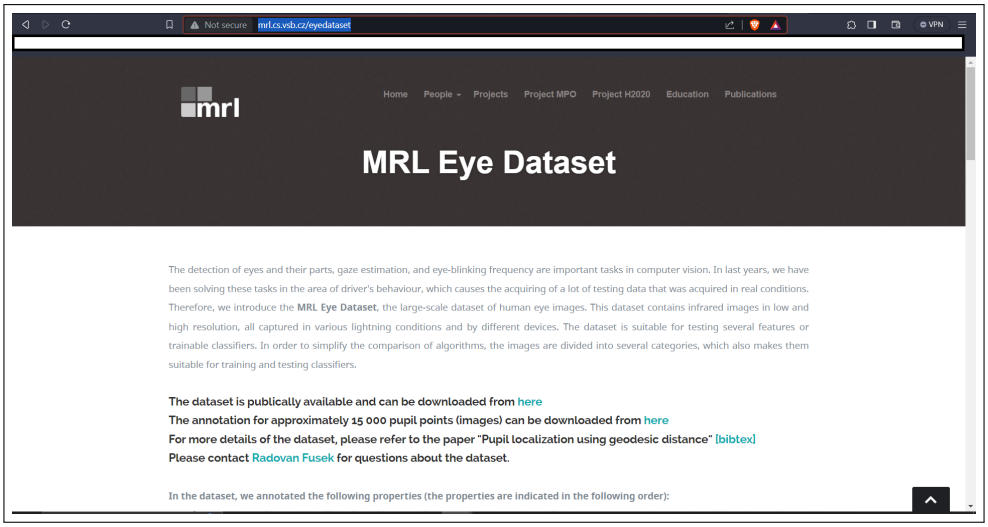

Figure 3: MRL Eye Dataset

```
import os
import shutil
import glob
from tqdm import tqdm
import random
from PIL import Image
```

Figure 4: Importing Libraries

Iterates through the Raw_DIR directory, segregating images based on whether the eye status is 'Closed' or 'Open'. It creates separate directories named 'Closed_Eyes' and 'Open_Eyes' within the Prepared_Data directory to store the categorized images. 5

```
Raw_DIR= r'C:\Users\91877\Downloads\mrlEyes_2018_01\mrlEyes_2018_01'
for dirpath, dirname, filenames in os.walk(Raw_DIR):
    for i in tqdm([f for f in filenames if f.endswith('.png')]):
        if i.split('_')[4]=='0':
            shutil.copy(src=dirpath+'/'+i, dst=r'C:\Users\91877\Downloads\mrlEyes_2018_01\Prepared_Data\Closed_Eyes')

        elif i.split('_')[4]=='1':
            shutil.copy(src=dirpath+'/'+i, dst=r'C:\Users\91877\Downloads\mrlEyes_2018_01\Prepared_Data\Open_Eyes')
```

Figure 5: Segregating Images

- **Display Random Images:**

  Define the paths to the 'Closed eyes' and 'Open eyes' directories (closed_eyes_dir and open_eyes_dir). Set the variable num_images_to_display to specify the number of random images you want to display for each category. Refer Figure 6

  The Output of the Random images is displayed in Figure 7

```python
def display_random_images(dataset_dir, num_images_to_display, category):
    # List all image files in the dataset directory
    image_files = [os.path.join(dataset_dir, f) for f in os.listdir(dataset_dir) if f.endswith('.png')]

    # Get random image paths
    random_image_paths = random.sample(image_files, num_images_to_display)

    # Display the random images
    plt.figure(figsize=(12, 6))
    for i, image_path in enumerate(random_image_paths, 1):
        image = Image.open(image_path)
        plt.subplot(1, num_images_to_display, i)
        plt.imshow(image)
        plt.axis('off')  # Remove axis labels
        plt.text(0.5, -0.15, f'{category} image {i}', ha='center', transform=plt.gca().transAxes, fontsize=10)

    plt.tight_layout()
    plt.show()

# Paths to directories containing closed and open eye images
closed_eyes_dir = r"C:\Users\91877\Downloads\mrlEyes_2018_01\Train\Closed eyes"
open_eyes_dir = r"C:\Users\91877\Downloads\mrlEyes_2018_01\Train\Open eyes"

# Set the number of random images you want to display for each category
num_images_to_display = 5

# Display random images from 'Closed eyes' directory with category label
display_random_images(closed_eyes_dir, num_images_to_display, 'Closed')

# Display random images from 'Open eyes' directory with category label
display_random_images(open_eyes_dir, num_images_to_display, 'Open')
```

Figure 6: Display Random Images

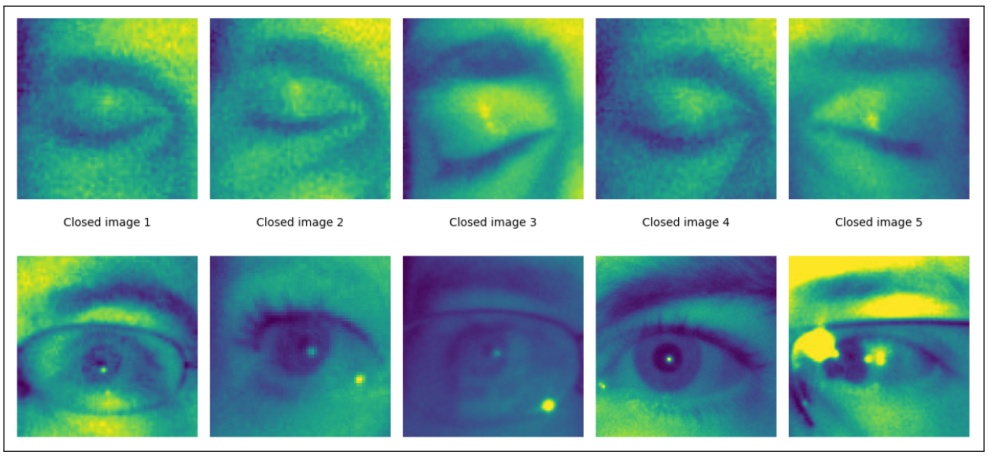

Figure 7: Random Images from the dataset

- **Manually transfer images**

  After the open and closed eyes are separated, manually create train and test folders and transfer 80% of open and closed eye images to train subset folder and 20% to test subset folder.

3.3 Data Preprocessing

- **ImageDataGenerator:**

  It preprocesses and augments the image data for training, validation, and testing sets. The augmentation parameters are: rotation_range, shear_range, zoom_range, width_shift_range, height_shift_range:

- **validation_split:** Splits the training data for validation.

- **Flow from Directory:** Reads and generates batches of preprocessed images from

directories.

Resize images to 80x80 pixels as Smaller size is faster to train while retaining features. Pixel values are scaled to [0,1] range which is known as Normalization for uniformity. Image augmentation on train set Synthetically expands dataset More robust to variations Rotation (up to 20 degrees) Handles head movement Shear, zoom, horizontal/vertical shifts (up to 20%) and Mimics eyes opening/closing Validation split of 0.2 on train set. Refer Figure 8

```
train_datagen= ImageDataGenerator(rescale=1./255, rotation_range=0.2,shear_range=0.2,
    zoom_range=0.2,width_shift_range=0.2,
    height_shift_range=0.2, validation_split=0.2)
```

Figure 8: Data Augmentation

- **Visualize Preprocessed Images**:

Displays a few preprocessed images from each dataset using Matplotlib. Refer Figure 9

```
import matplotlib.pyplot as plt

def visualize_preprocessed_images(generator, num_images, dataset_type):
    plt.figure(figsize=(12, 6))
    for i in range(num_images):
        batch = generator.next()
        image = batch[0][0]
        plt.subplot(1, num_images, i+1)
        plt.imshow(image)
        plt.xlabel('Width')
        plt.ylabel('Height')
        plt.axis('on')  # Show axis lines and labels
        plt.title(f"{dataset_type} Image {i+1}")
    plt.suptitle(f"Preprocessed {dataset_type} Images")
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# Visualizing preprocessed images
num_images_to_visualize = 6
visualize_preprocessed_images(train_data, num_images_to_visualize, 'Training')
visualize_preprocessed_images(validation_data, num_images_to_visualize, 'Validation')
visualize_preprocessed_images(test_data, num_images_to_visualize, 'Test')
```

Figure 9: Data visualisation

3.4 Model Creation and Training:

- **Model 1:** Model 1 Uses InceptionV3 (2023b) as the base model with additional Dense layers. Freezes the layers of the base model and compiles it for training. Trains the model and evaluates its performance on the test dataset.

  Defining the desired model architecture by adjusting the layers, units, and dropout rates. Customizing the callbacks (ModelCheckpoint, EarlyStopping, ReduceLROnPlateau) based on requirements. ModelCheckpoint used to later load for evaluation or predictions. Patience parameter under EarlyStopping kept at 7 and the number of epochs for training is 10. Refer Figure 10 and Figure 11

- Summarize the model 12

Model 1

```
bmodel = InceptionV3(include_top=False, weights='imagenet', input_tensor=Input(shape=(80,80,3)))
hmodel = bmodel.output
hmodel = Flatten()(hmodel)
hmodel = Dense(64, activation='relu')(hmodel)
hmodel = Dropout(0.5)(hmodel)
hmodel = Dense(2,activation= 'softmax')(hmodel)

model = Model(inputs=bmodel.input, outputs= hmodel)
for layer in bmodel.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 8s 0us/step
```

Figure 10: Model 1

```
checkpoint = ModelCheckpoint(r"C:\Users\91877\Downloads\mrlEyes_2018_01\Models\model.h5",
                             monitor='val_loss',save_best_only=True,verbose=3)

earlystop = EarlyStopping(monitor = 'val_loss', patience=7, verbose= 3, restore_best_weights=True)

learning_rate = ReduceLROnPlateau(monitor= 'val_loss', patience=3, verbose= 3, )

callbacks=[checkpoint,earlystop,learning_rate]
```

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

model.fit_generator(train_data,steps_per_epoch=train_data.samples//batchsize,
                    validation_data=validation_data,
                    validation_steps=validation_data.samples//batchsize,
                    callbacks=callbacks,
                     epochs=10)
```

Figure 11: Model 1

```
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 80, 80, 3)]       0         []

 conv2d (Conv2D)             (None, 39, 39, 32)        864       ['input_1[0][0]']

 batch_normalization (Batch  (None, 39, 39, 32)        96        ['conv2d[0][0]']
 Normalization)

 activation (Activation)     (None, 39, 39, 32)        0         ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)           (None, 37, 37, 32)        9216      ['activation[0][0]']

 batch_normalization_1 (Bat  (None, 37, 37, 32)        96        ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, 37, 37, 32)        0         ['batch_normalization_1[0][0]'
```

Figure 12: Summary

- **Model 2:**

  Similar to Model 1 but with an added architecture of two additional Dense layers and Dropout for regularization.

  Freezes the layers of the base model, patience parameter increased to 14 and the model was run for 20 epochs then compiles it for training and evaluates its performance on the test dataset. Refer to Figure 13

- **Model 3:**

  Similar to Model 2 but with further added Dense layers and Dropouts. Freezes the layers of the base model and compiles it for training. Trains the model and

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input
from tensorflow.keras.applications import InceptionV3

# Define the base model
base_model2 = InceptionV3(include_top=False, weights='imagenet', input_tensor=Input(shape=(80, 80, 3)))

# Get the output from the base model
head_model2 = base_model2.output
head_model2 = Flatten()(head_model2)
head_model2 = Dense(128, activation='relu')(head_model2)  # Additional dense layer
head_model2 = Dropout(0.5)(head_model2)
head_model2 = Dense(64, activation='relu')(head_model2)   # Another additional dense layer
head_model2 = Dropout(0.5)(head_model2)
head_model2 = Dense(2, activation='softmax')(head_model2)

# Create the model by specifying the inputs and outputs
model2 = Model(inputs=base_model2.input, outputs=head_model2)

# Freeze the layers of the base model
for layer in base_model2.layers:
    layer.trainable = False
```

Figure 13: Model 2 Architecture

evaluates its performance on the test dataset. Refer to Figure 14

Model 3

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

# Define the base model
base_model3 = InceptionV3(include_top=False, weights='imagenet', input_tensor=Input(shape=(80, 80, 3)))

# Get the output from the base model
head_model3 = base_model3.output
head_model3 = Flatten()(head_model3)
head_model3 = Dense(128, activation='relu')(head_model3)  # Additional dense layer
head_model3 = Dropout(0.5)(head_model3)
head_model3 = Dense(64, activation='relu')(head_model3)   # Another additional dense layer
head_model3 = Dropout(0.5)(head_model3)
head_model3 = Dense(32, activation='relu')(head_model3)   # 3rd dense layer
head_model3 = Dropout(0.5)(head_model3)
head_model3 = Dense(16, activation='relu')(head_model3)   # 4th dense layer
head_model3 = Dropout(0.5)(head_model3)
head_model3 = Dense(8, activation='relu')(head_model3)    # 5th dense layer
head_model3 = Dropout(0.5)(head_model3)
head_model3 = Dense(2, activation='softmax')(head_model3)

# Create the model by specifying the inputs and outputs
model3 = Model(inputs=base_model3.input, outputs=head_model3)

# Freeze the layers of the base model
for layer in base_model3.layers:
    layer.trainable = False

model3.summary()
```

Figure 14: Model 3 Architecture

3.5 Model Evaluation:

After training, evaluate all the three models on the test data. Use the trained models to make predictions on test data and compute performance metrics (accuracy, confusion matrix, classification report). Refer Figure 15

4 Real-Time Implementation

After the model is trained and saved, Its time to load it and run predictions by feeding live images via the web cam.

```
# Assuming your model has been trained and your test_data is prepared

# Load the trained model
from keras.models import load_model
trained_model_path = r"C:\Users\91877\Downloads\mrlEyes_2018_01\Models\model.h5"
model = load_model(trained_model_path)

# Evaluate the model on test data
test_results = model.evaluate(test_data)

# Extract the test accuracy
test_accuracy = test_results[1]  # The accuracy metric is typically at index 1

# Print the test accuracy
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
572/572 [==============================] - 29s 49ms/step - loss: 0.4201 - accuracy: 0.8604
Test Accuracy: 86.04%
```

Figure 15: Evaluations

- **Load Libraries**

  OpenCV: Used for video capture, face, and eye detection.

  TensorFlow and Keras: For loading a pre-trained deep learning model.

  Pygame: For playing sound alerts.

- **Face and Eye Detection:**

  Cascade Classifiers: Utilizes pre-trained Haar Cascade classifiers to detect faces and eyes in the video frames. Refer Figure 16

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
model = load_model(r'C:\Users\91877\Downloads\mrlEyes_2018_01\Models\model3.h5')
```

Figure 16: Classifier

- **Eye State Prediction and Alert System:**

  Preprocesses the detected eye region, resizes it, normalizes pixel values, and predicts the eye state using the loaded model. Alert System:

  If closed eyes are detected continuously (based on the prediction probability thresholds), it increments a score and triggers an alarm sound. Conversely, for open eyes, it decrements the score.

  Draws rectangles around detected faces and eyes. Displays the current eye state ('closed' or 'open') and the score.

  Quit Command:

  Press 'q' to exit the live video feed. Refer figure 17

```
mixer.init()
sound= mixer.Sound(r'C:\Users\91877\Downloads\mrlEyes_2018_01\alarm.wav')
cap = cv2.VideoCapture(0)
Score = 0
while True:
    ret, frame = cap.read()
    height,width = frame.shape[0:2]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces= face_cascade.detectMultiScale(gray, scaleFactor= 1.2, minNeighbors=3)
    eyes= eye_cascade.detectMultiScale(gray, scaleFactor= 1.1, minNeighbors=1)

    cv2.rectangle(frame, (0,height-50),(200,height),(0,0,0),thickness=cv2.FILLED)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,pt1=(x,y),pt2=(x+w,y+h), color= (255,0,0), thickness=3 )

    for (ex,ey,ew,eh) in eyes:
        #cv2.rectangle(frame,pt1=(ex,ey),pt2=(ex+ew,ey+eh), color= (255,0,0), thickness=3 )

        # preprocessing steps
        eye= frame[ey:ey+eh,ex:ex+ew]
        eye= cv2.resize(eye,(80,80))
        eye= eye/255
        eye= eye.reshape(80,80,3)
        eye= np.expand_dims(eye,axis=0)
        # preprocessing is done now model prediction
        prediction = model.predict(eye)
        #print(prediction)

        # if eyes are closed
        if prediction[0][0]>0.30:
            cv2.putText(frame,'closed',(10,height-20),fontFace=cv2.FONT_HERSHEY_COMPLEX_SMALL,fontScale=1,color=(255,255,255),
                        thickness=1,lineType=cv2.LINE_AA)
            cv2.putText(frame,'Score'+str(Score),(100,height-20),fontFace=cv2.FONT_HERSHEY_COMPLEX_SMALL,fontScale=1,color=(255,
                        thickness=1,lineType=cv2.LINE_AA)
            Score=Score+1
            if(Score>10):
                try:
                    sound.play()
                except:
                    pass

        # if eyes are open
        elif prediction[0][1]>0.90:
            cv2.putText(frame,'open',(10,height-20),fontFace=cv2.FONT_HERSHEY_COMPLEX_SMALL,fontScale=1,color=(255,255,255),
                        thickness=1,lineType=cv2.LINE_AA)
            cv2.putText(frame,'Score'+str(Score),(100,height-20),fontFace=cv2.FONT_HERSHEY_COMPLEX_SMALL,fontScale=1,color=(255,
                        thickness=1,lineType=cv2.LINE_AA)
            Score = Score-1
            if (Score<0):
                Score=0




        cv2.imshow('frame',frame)
        if cv2.waitKey(33) & 0xFF==ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

Figure 17: Real time detection

References

. (2023a). Keras Documentation, `https://keras.io/`. Online; Accessed December 2023.

(2023b). Keras InceptionV3 Documentation, `https://keras.io/api/applications/`

inceptionv3/. Online; Accessed December 2023.

Google LLC (2023). TensorFlow, https://www.tensorflow.org/. Online; Accessed December 2023.

Python Software Foundation (2023). Python Programming Language. Online; Accessed December 2023.
URL: *https://www.python.org/*

The Jupyter Development Team (2022). Jupyter Project. Online; Accessed January 2022.
URL: *https://jupyter.org/*

Vietnam National University, H. C. M. C. (2018). MRL Eyes Dataset, http://mrl.cs.vsb.cz/eyedataset. Online.