# Configuration Manual

MSc Research Project
Data Analytics

Senthilnathan Balamurugan
Student ID: X21193371

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

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

| | |
|---|---|
| **Student Name:** | Senthilnathan Balamurugan |
| **Student ID:** | X21193371 |
| **Programme:** | Data Analytics |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Dr Giovani Estrada |
| **Submission Due Date:** | 14/08/2023 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 544 |
| **Page Count:** | 8 |

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:** | Senthilnathan Balamurugan |
| **Date:** | 17th September 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. | ☐ |

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

Senthilnathan Balamurugan
X21193371

## 1 Introduction

This configuration manual can be utilized to achieve the same objectives as the original work. It includes the system configuration used to carry out the project, the methods used for data pre preocessing, the model architecture, and the model evaluations.

## 2 System Requirements

This section describes the system requirements , hardware configurations and software packages that are necessary to reproduce the research are given in table 1

Table 1: System Requirements

| **Environment** | **Jupyter Notebook** |
|---|---|
| Operating System | Windows 11 64-bit OS |
| RAM(Random Access Memory) | 16GB |
| Processor | AMD Ryzen 9 5900HX |
| Graphical processing unit | NVIDIA GeForce RTX 3060 |
| Storage(Harddisk) | 477GB |

### 2.1 Software Requuirements

I have used the following softwares for this research -

- Anaconda Navigator
- Jupyter Notebook
- Python 3.11

The research project was performed in the Jupyter Notebook environment powered by Anaconda Navigator. Python was the programming language used for the research.

## 3 Dataset

I have used an open source dataset from kaggle for this research.
Link - https://www.kaggle.com/datasets/serenaraju/yawn-eye-dataset-new

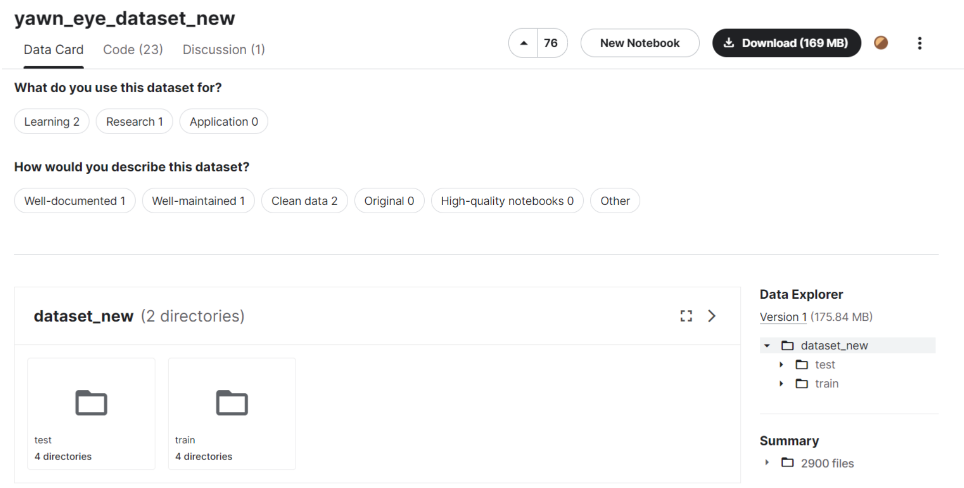


Figure 1: Dataset

# 4 Workflow

Figure 2 shows the workflow of the research.

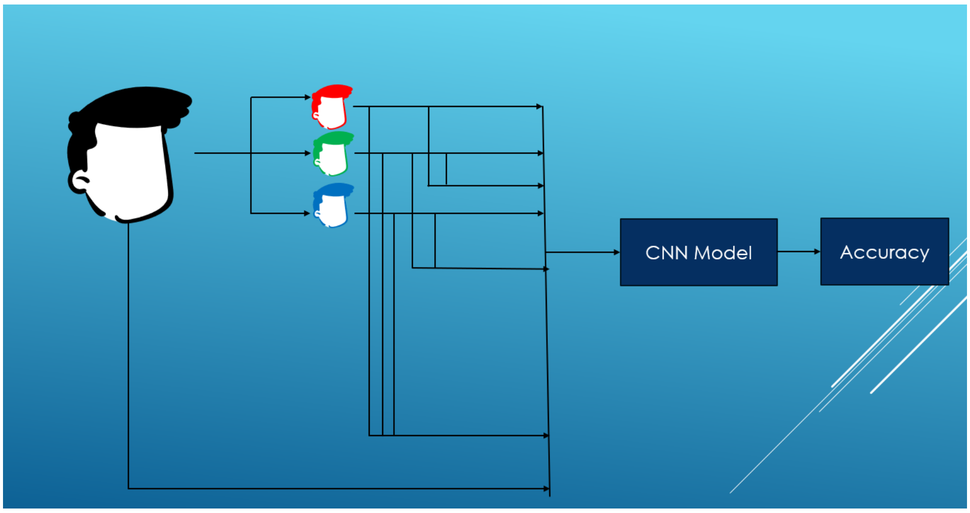


Figure 2: Workflow

# 5 Packages and Libraries

In this section, I will describe about the packages and libraries used in python for performing the research. The packages and libraries used are shown in the table 2

# 6 Dataset

Figure 3 and 4 shows the code for loading the dataset and visualizing the dataset. We can see that the dataset is loaded into as 4 classses such as eyes open, eyes closed, yawn

```
In [1]: import keras
        from keras.models import Sequential
        from keras.callbacks import ModelCheckpoint
        from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
        from keras.preprocessing.image import ImageDataGenerator

        import tensorflow as tf

In [2]: import matplotlib.pyplot as plt
        plt.style.use('dark_background')

In [3]: import os
        def plot_imgs(directory, top=10):
            all_item_dirs = os.listdir(directory)
            item_files = [os.path.join(directory, file) for file in all_item_dirs][:5]

            plt.figure(figsize=(20, 20))

            for i, img_path in enumerate(item_files):
                plt.subplot(10, 10, i+1)

                img = plt.imread(img_path)
                plt.tight_layout()
                plt.imshow(img, cmap='gray')

In [4]: batch_size = 40
        img_height = 256
        img_width = 256
```

Figure 3: Dataset Load 1

```
In [5]: data_path = 'dataset_new/train'

        directories = ['/Closed', '/Open', '/no_yawn', '/yawn']

        for j in directories:
            plot_imgs(data_path+j)
```

Figure 4: Dataset Load 2

Table 2: Packages and Libraries

| **Package Name** | **Jupyter Notebook** |
|---|---|
| OS | Used for loading the dataset and managing the file directories |
| matplotlib.pyplot | Used for plotting visuals |
| keras | Used for creating and training machine learning models |
| tensorflow | Used for creating and training machine learning models |
| keras.preprocessing.image | Used for image augmentation |
| keras.layers | Used for defining the layers |
| keras.models | Used for defining models |
| keras.callbacks | Used for creating model checkpoints |

```
batch_size = 128
train_datagen = ImageDataGenerator(horizontal_flip = True,
                                   rescale = 1./255,
                                   zoom_range = 0.2,
                                   validation_split = 0.1)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
train_data_path = 'dataset_new/train'
test_data_path = 'dataset_new/test'

train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              color_mode = 'rgb',
                                              class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              color_mode = 'rgb',
                                              class_mode = 'categorical')
Found 2467 images belonging to 4 classes.
Found 433 images belonging to 4 classes.
```

Figure 5: RGB

and no yawn. The batch size is set to 40, height and width are set as 256. The images are then plotted using plot_img function

# 7 Color channel split

Figure 5 shows the code for taking the input as RGB Channel. The *ImageDataGenerator* library is used to help with the data preprocessing. The color mode is set as RGB and class mode is set as categorical.

Figure 6 shows the code for taking the input as Grayscale Channel. The color mode is set as Grayscale and the class mode is set as categorical.

Figure 7 shows the code for taking the input as Red Channel. Preprocessing functions are used to extract only the red colour channel. The color mode is not defined as only red channel is taken as input and class mode is set as categorical.

Figure 8 shows the code for taking the input as Green Channel.

Figure 9 shows the code for taking the input as Blue Channel.

Figure 10 shows the code for taking the input as Red and Green Channel.

Figure 11 shows the code for taking the input as Red and Blue Channel.

Figure 12 shows the code for taking the input as Green and Blue Channel.

# 8 Model Creation and Summary

This section provides the code snippet for the creation and summary of the CNN model.

```python
batch_size = 128
train_datagen = ImageDataGenerator(horizontal_flip = True,
                                   rescale = 1./255,
                                   zoom_range = 0.2,
                                   validation_split = 0.1)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

```python
train_data_path = 'dataset_new/train'
test_data_path = 'dataset_new/test'

train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              color_mode = 'grayscale',
                                              class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                            batch_size = batch_size,
                                            color_mode = 'grayscale',
                                            class_mode = 'categorical')
```

```
Found 2467 images belonging to 4 classes.
Found 433 images belonging to 4 classes.
```

Figure 6: Grayscale

```python
batch_size = 128

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: img[:, :, 0:1],
    zoom_range = 0.2,
    validation_split = 0.1)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: img[:, :, 0:1])
```

```python
train_data_path = 'dataset_new/train'
test_data_path = 'dataset_new/test'

train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                            batch_size = batch_size,
                                            class_mode = 'categorical')
```

```
Found 2467 images belonging to 4 classes.
Found 433 images belonging to 4 classes.
```

Figure 7: Red Channel

Figure 13 shows the code for the creation of the CNN model.
Figure 14 shows the code for the summary of the created CNN model.

# 9 Model Evaluation

Figure 15 shows the code for the training and evaluation of the model. Categorical cross entropy is used as loss function. Adam is the optimizer and accuracy is the evaluation metric. Checkpoint is created to save the best model with highest val_accuracy. 30 epochs were set to train the model. The model is then evaluated with the test set

```
batch_size = 128

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: img[:, :, 1:2],
    zoom_range = 0.2,
    validation_split = 0.1)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: img[:, :, 1:2])
```

```
train_data_path = 'dataset_new/train'
test_data_path = 'dataset_new/test'

train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')
```

```
Found 2467 images belonging to 4 classes.
Found 433 images belonging to 4 classes.
```

Figure 8: Green Channel

```
batch_size = 128

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: img[:, :, 2:],
    zoom_range = 0.2,
    validation_split = 0.1)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: img[:, :, 2:])
```

```
train_data_path = 'dataset_new/train'
test_data_path = 'dataset_new/test'

train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')
```

```
Found 2467 images belonging to 4 classes.
Found 433 images belonging to 4 classes.
```

Figure 9: Blue Channel

```
batch_size = 128

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: tf.stack([img[:, :, 0], img[:, :, 1], tf.zeros_like(img[:, :, 0])], axis=-1),zoom_range =
    validation_split = 0.1
)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda img: tf.stack([img[:, :, 0], img[:, :, 1], tf.zeros_like(img[:, :, 0])], axis=-1)
)
```

```
train_data_path = 'dataset_new/train'
test_data_path = 'dataset_new/test'

train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                              batch_size = batch_size,
                                              class_mode = 'categorical')
```

```
Found 2467 images belonging to 4 classes.
Found 433 images belonging to 4 classes.
```

Figure 10: RedGreen Channel

```
In [7]: batch_size = 128

        train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
            rescale=1./255,
            preprocessing_function=lambda img: tf.stack([img[:, :, 0], tf.zeros_like(img[:, :, 0]), img[:, :, 2]], axis=-1),
            zoom_range = 0.2,
            validation_split = 0.1
        )

        test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
            rescale=1./255,
            preprocessing_function=lambda img: tf.stack([img[:, :, 0], tf.zeros_like(img[:, :, 0]), img[:, :, 2]], axis=-1)
        )

In [8]: train_data_path = 'dataset_new/train'
        test_data_path = 'dataset_new/test'

        train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                                      batch_size = batch_size,
                                                      class_mode = 'categorical')

        test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                                      batch_size = batch_size,
                                                      class_mode = 'categorical')

        Found 2467 images belonging to 4 classes.
        Found 433 images belonging to 4 classes.
```

Figure 11: RedBlue Channel

```
In [7]: batch_size = 128

        train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
            rescale=1./255,
            preprocessing_function=lambda img: tf.stack([tf.zeros_like(img[:, :, 0]), img[:, :, 1], img[:, :, 2]], axis=-1),
            zoom_range = 0.2,
            validation_split = 0.1
        )

        test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
            rescale=1./255,
            preprocessing_function=lambda img: tf.stack([tf.zeros_like(img[:, :, 0]), img[:, :, 1], img[:, :, 2]], axis=-1)  # Extract gr
        )

In [8]: train_data_path = 'dataset_new/train'
        test_data_path = 'dataset_new/test'

        train_set = train_datagen.flow_from_directory(train_data_path, target_size = (256,256),
                                                      batch_size = batch_size,
                                                      class_mode = 'categorical')

        test_set = test_datagen.flow_from_directory(test_data_path, target_size = (256,256),
                                                      batch_size = batch_size,
                                                      class_mode = 'categorical')

        Found 2467 images belonging to 4 classes.
        Found 433 images belonging to 4 classes.
```

Figure 12: GreenBlue Channel

```
classes = 4

model = Sequential()
model.add(Conv2D(32, (3,3), padding = 'same', input_shape = (256,256,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(64, (3,3), padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(128,(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Flatten())

model.add(Dense(32, activation = 'relu'))

model.add(Dense(classes, activation = 'softmax'))

print(model.summary())
```

Figure 13: Model Creation

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 128)      0
 2D)

 flatten (Flatten)           (None, 131072)            0

 dense (Dense)               (None, 32)                4194336

 dense_1 (Dense)             (None, 4)                 132

=================================================================
Total params: 4,287,716
Trainable params: 4,287,716
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 14: Model Summary

```
In [10]: model.compile(loss = 'categorical_crossentropy',optimizer = 'adam' , metrics = ['accuracy'])

         model_path="yawn_detection1.h5"

         checkpoint = ModelCheckpoint(model_path, monitor='val_accuracy', verbose=1,
                                      save_best_only=True, mode='max')

         callbacks_list = [checkpoint]

         num_epochs = 30
         training_steps=train_set.n//train_set.batch_size
         validation_steps =test_set.n//test_set.batch_size

In [11]: history = model.fit(train_set, epochs=num_epochs, steps_per_epoch=training_steps,validation_data=test_set,
                             validation_steps=validation_steps, callbacks = callbacks_list)
```

```
19/19 [==============================] - ETA: 0s - loss: 0.0821 - accuracy: 0.9679
Epoch 27: val_accuracy improved from 0.95833 to 0.96094, saving model to yawn_detection1.h5
19/19 [==============================] - 73s 4s/step - loss: 0.0821 - accuracy: 0.9679 - val_loss: 0.1089 - val_accuracy: 0.
9609
Epoch 28/30
19/19 [==============================] - ETA: 0s - loss: 0.0625 - accuracy: 0.9799
Epoch 28: val_accuracy improved from 0.96094 to 0.96354, saving model to yawn_detection1.h5
19/19 [==============================] - 76s 4s/step - loss: 0.0625 - accuracy: 0.9799 - val_loss: 0.1041 - val_accuracy: 0.
9635

Epoch 29/30
19/19 [==============================] - ETA: 0s - loss: 0.0710 - accuracy: 0.9743
Epoch 29: val_accuracy did not improve from 0.96354
19/19 [==============================] - 79s 4s/step - loss: 0.0710 - accuracy: 0.9743 - val_loss: 0.0941 - val_accuracy: 0.
9583
Epoch 30/30
19/19 [==============================] - ETA: 0s - loss: 0.0771 - accuracy: 0.9714
Epoch 30: val_accuracy did not improve from 0.96354
19/19 [==============================] - 75s 4s/step - loss: 0.0771 - accuracy: 0.9714 - val_loss: 0.1706 - val_accuracy: 0.
9375
```

```
In [12]: score = model.evaluate(test_set)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])

         4/4 [==============================] - 4s 756ms/step - loss: 0.1561 - accuracy: 0.9446
         Test loss: 0.1560841202735901
         Test accuracy: 0.9445727467536926
```

Figure 15: Model Evaluation