Configuration Manual

MSc Research Project
Data Analytics

Sangeetha Pillay Anil Kumar
Student ID: 20232195

School of Computing
National College of Ireland

Supervisor: Prof. Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sangeetha Pillay Anil Kumar
Student ID:	20232195
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Prof. Qurrat Ul Ain
Submission Due Date:	19/08/2022
Project Title:	Configuration Manual
Word Count:	631
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:	18th September 2022

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

Sangeetha Pillay Anil Kumar
20232195

1 Introduction

This study uses lesion features extracted from the images to classify the stages of diabetic retinopathy (DR), using the retinal fundus image. There are code snippets from several parts that are added as necessary in this document to provide all of the instructions required to reproduce this study.

2 Hardware Configuration

The machine utilized for the implementation of this study has Windows 10 Pro, a 64-bit operating system, an x64-based processor, an 7th Gen Intel(R) Core(TM) i5-7200U CPU, and 20GB of RAM. The Hardware configuration of the system is depicted in Figure 1

Figure 1: Hardware Configuration

3 System Configuration

Jupyter Notebook (Anaconda Navigator). was used to carry out the project's implementation. All of the coding in this article is done with Python 3. The version of Anaconda is 2.0.3 which shown in the Figure 2. The notebook server is at versions 6.3.0 and 3.8.8. (default, Apr 13 2021, 15:08:03) v.1916 of MSC (AMD64) is the Python version running on the server depicted in the Figure 3.

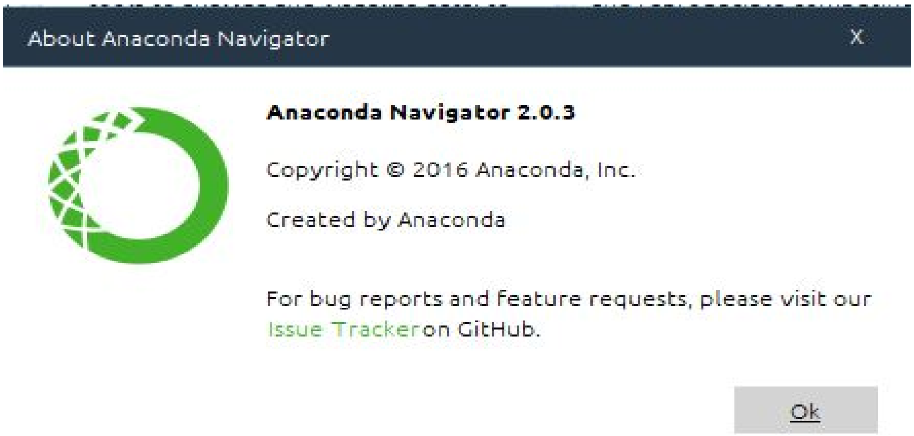

Figure 2: AnacondaNavigator Configuration

.

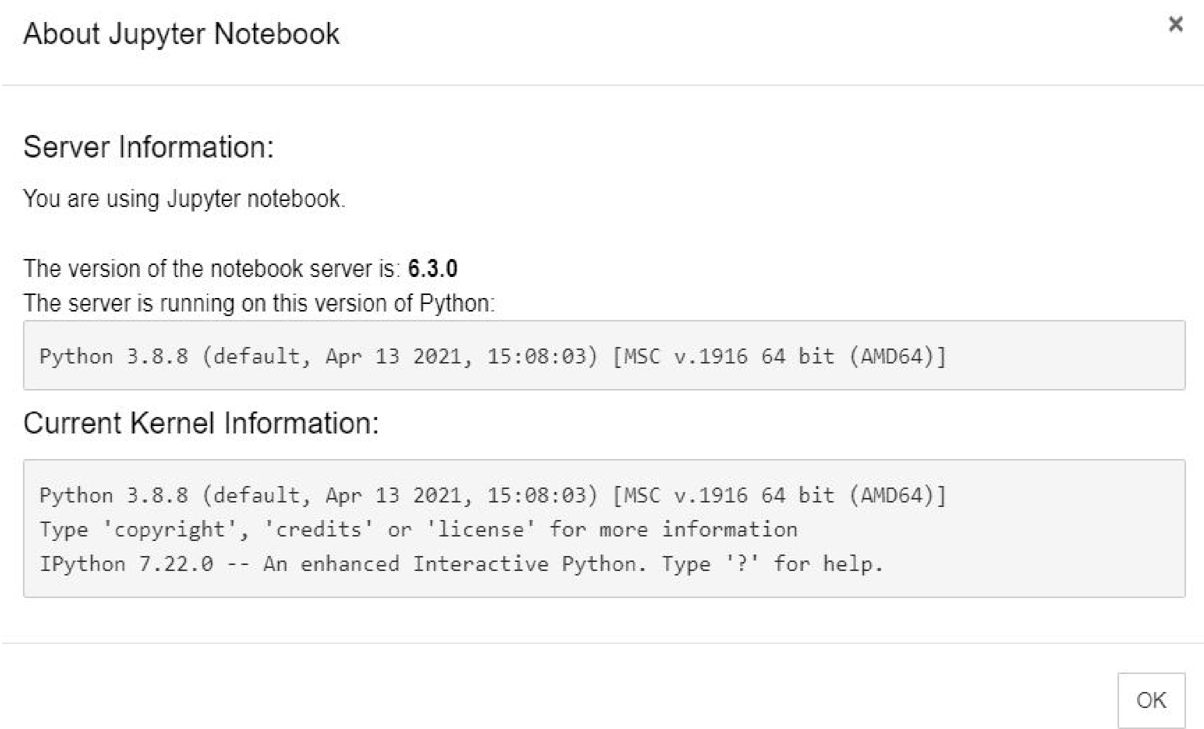

Figure 3: JupyterNotebook Configuration

.

4 Library Requirement

The Figure ?? is a list of the Python packages that must be installed in the environment for reproducing the study. Each package is installed using the "pip" command. The "import" method was used to import the remaining internal libraries which is depicted in Figure ??.

Table 1: Required Libraries

Packages	**Version**
PIL	8.2.0
cv2	4.0.1
keras	2.8.0
matplotlib	3.3.4
pandas	1.2.4
seaborn	0.11.1
session_info	1.0.0
sklearn	1.1.2
tensorflow	2.8.0
theano	1.0.5

5 Data Collection

The Standard Diabetic Retinopathy Database and Kaggle provided the datasets for this study. This datasets were allowed to downloaded from the websites in the system as directories . The Kaggle dataset was loaded to the python. The Standard Diabetic Retinopathy dataset was loaded into python as a csv file which consists of the images along with the presence and absence of the lesions.

6 Data Preprocessing

The preprocessing of the datasets was done separately for the two datasets.

6.1 Standard Diabetic Retinopathy Database

The lesion feature was learned using this dataset. This image dataset was initially converted to an array in order to extract the lesion feature from the images. For that, the code shown in the Figure 4 was applied. For all four different types of lesions, this was used,by changing the $Image.open()$ argument. Then, the data was divided into train and test sets as depicted in Figure 5.

6.2 Kaggle Dataset

This image dataset was loaded into the python and the obtained importance feature from the lesion dataset was used to extract the lesion feature from this dataset. And the data was splited into train and test data as shown in Figure 6. This same Method was carried out in the data training for the classification stage of DR.

```
img_rows, img_cols = 224, 224
immatrix=[]
imlabel=[]
for indx,item in df.iterrows():
    imlabel.append(item[0])
    im = Image.open(item[4])
    img = im.resize((img_rows,img_cols))
    gray = img.convert('L')
    immatrix.append(np.array(img).flatten())
immatrix = np.asarray(immatrix)
imlabel = np.asarray(imlabel)
```

Figure 4: Code for converting image to array

```
X_train, X_test, y_train, y_test = train_test_split(train_data[0], train_data[1], test_size = 0.2, random_state = 5)
```

Figure 5: Train Test Split

```
len(imlabel)
for index in range(0,len(final_matrix)):
    cv2.imwrite(f'F:\\NCI_Documents\\Final Thesis\\feature\\{index}.png',final_matrix[index])
    final_matrix[index] = f'F:\\NCI_Documents\\Final Thesis\\feature\\{index}.png'
```

```
X_train, X_test, y_train, y_test = train_test_split(final_matrix, imlabel, test_size = 0.2, random_state = 100)
```

```
data_tuples = list(zip(X_train,y_train))
```

```
data_tup = pd.DataFrame(data_tuples,columns=['image','label'])
```

Figure 6: Code for dataframe of DR No_DR image dataset

7 Data Augmentation

In order to accomplish real-time augmentation, image data generators were created. The keras image data generators are implemented in the code depicted in Figure 7. This same method was carried out with the DR stage classification image.

8 Model Building

Figure 8 depicts the model-building code for detecting the presence of DR or No DR, and Figure 9 depicts the code for classifying the stage of DR.

9 Model Fitting

Pretrained VGG-16 and VGG-19 models from CNN were employed in this study. Figure 10 and Figure 11 shows the code used for fitting this pretrained models.

```python
def create_gen():
    # Load the Images with a generator and Data Augmentation
    train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
        validation_split=0.1
    )

    test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input
    )

    train_images = train_generator.flow_from_dataframe(
        dataframe=train_df,
        x_col='image',
        y_col='label',
        target_size=(32,32),
        color_mode='rgb',
        class_mode='categorical',
        batch_size=32,
        shuffle=True,
        seed=0,
        subset='training',
        rotation_range=30, # Uncomment to use data augmentation.value_counts()
        zoom_range=0.15,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.15,
        horizontal_flip=True,
        fill_mode="nearest"
    )

    val_images = train_generator.flow_from_dataframe(
        dataframe=train_df,
        x_col='image',
        y_col='label',
        target_size=(32,32),
        color_mode='rgb',
        class_mode='categorical',
        batch_size=32,
        shuffle=True,
        seed=0,
        subset='validation',
        rotation_range=30, # Uncomment to use data augmentation
        zoom_range=0.15,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.15,
        horizontal_flip=True,
        fill_mode="nearest"
    )

    test_images = test_generator.flow_from_dataframe(
        dataframe=test_df,
        x_col='image',
        y_col='label',
        target_size=(32, 32),
        color_mode='rgb',
        class_mode='categorical',
        batch_size=32,
        shuffle=False
    )

    return train_generator,test_generator,train_images,val_images,test_images
```

Figure 7: Image Generators

```python
def get_model(model):
# Load the pretained model
    kwargs =    {'input_shape':(32,32,3),
                'include_top':False,
                'weights':'imagenet',
                'pooling':'avg'}

    pretrained_model = model(**kwargs)
    pretrained_model.trainable = False

    inputs = pretrained_model.input

    x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
    x = tf.keras.layers.Dense(128, activation='relu')(x)

    outputs = tf.keras.layers.Dense(2, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

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

    return model
```

Figure 8: CNN model 1

```python
def get_model(model):
# Load the pretained model
    kwargs =    {'input_shape':(32,32,3),
                'include_top':False,
                'weights':'imagenet',
                'pooling':'avg'}

    pretrained_model = model(**kwargs)
    pretrained_model.trainable = False

    inputs = pretrained_model.input

    x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
    x = tf.keras.layers.Dense(128, activation='relu')(x)

    outputs = tf.keras.layers.Dense(4, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

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

Figure 9: CNN model 2

10 Evaluation

The model's train accuracy, validation accuracy, training time, specificity, and sensitivity were all calculated for the model evaluation. And compared the models' levels of accuracy. Figure 12 and shows the code used for evaluating the models.

```python
# Dictionary with the models
from time import perf_counter
seed_value = 0
np.random.seed(seed_value)
models = {
    "VGG16": {"model":tf.keras.applications.VGG16, "perf":0},
    "VGG19": {"model":tf.keras.applications.VGG19, "perf":0},
}
# Create the generators
train_generator,test_generator,train_images,val_images,test_images=create_gen()
print('\n')
trained_models = []
# Fit the models
for name, model in models.items():

    # Get the model
    m = get_model(model['model'])
    models[name]['model'] = m

    start = perf_counter()

    # Fit the model
    history = m.fit(train_images,validation_data=val_images,epochs=200,verbose=1)
    trained_models.append(history)
    # Sav the duration, the train_accuracy and the val_accuracy
    duration = perf_counter() - start
    duration = round(duration,2)
    models[name]['perf'] = duration
    print(f"{name:20} trained in {duration} sec")

    val_acc = history.history['val_accuracy']
    models[name]['val_acc'] = [round(v,4) for v in val_acc]

    train_acc = history.history['accuracy']
    models[name]['train_accuracy'] = [round(v,4) for v in train_acc]
```

Figure 10: Model Fitting of CNN 1

```python
# Dictionary with the models
from time import perf_counter
seed_value = 200
np.random.seed(seed_value)
models = {
    "VGG16": {"model":tf.keras.applications.VGG16, "perf":0},
    "VGG19": {"model":tf.keras.applications.VGG19, "perf":0},
}
# Create the generators
train_generator,test_generator,train_images,val_images,test_images=create_gen()
print('\n')

# Fit the models
for name, model in models.items():

    # Get the model
    m = get_model(model['model'])
    models[name]['model'] = m

    start = perf_counter()

    # Fit the model
    history = m.fit(train_images,validation_data=val_images,epochs=250,verbose=1)

    # Sav the duration, the train_accuracy and the val_accuracy
    duration = perf_counter() - start
    duration = round(duration,2)
    models[name]['perf'] = duration
    print(f"{name:20} trained in {duration} sec")

    val_acc = history.history['val_accuracy']
    models[name]['val_acc'] = [round(v,4) for v in val_acc]

    train_acc = history.history['accuracy']
    models[name]['train_accuracy'] = [round(v,4) for v in train_acc]
    models[name]['predictions']=m.predict(test_images)
```

Figure 11: Model Fitting of CNN 2

```
# Create a DataFrame with the results
models_result = []

for name, v in models.items():
    models_result.append([ name,
                          models[name]['train_accuracy'][-1],
                          models[name]['val_acc'][-1],
                          models[name]['perf']])

df_results = pd.DataFrame(models_result,
                          columns = ['model','train_accuracy','val_accuracy','Training time (sec)'])
df_results.sort_values(by='val_accuracy', ascending=False, inplace=True)
df_results.reset_index(inplace=True,drop=True)
df_results
```

Figure 12: Model Accuracyl

```
pd.DataFrame(history.history)[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.show()

pd.DataFrame(history.history)[['loss','val_loss']].plot()
plt.title("Loss")
plt.show()


from sklearn.metrics import classification_report
y_test = list(test_df.label)
```

Figure 13: Learning curve