National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Aditya Pramod Shinde
Student ID: 20178883

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Aditya Pramod Shinde

Student ID: 20178883

Programme: Data Analytics **Year:** 2022

Module: MSc Research Project

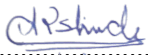
Lecturer: Prof. Vikas Sahni

Submission Due Date: 15/08/2022

Project Title: Configuration Manual

Word Count: 650 **Page Count:** 6

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: 5/08/2022

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

Aditya Pramod Shinde
Student ID: 20178883

1 Introduction

The aim of this configuration manual is to provide information about the hardware and software specifications used in this research alongside providing a detailed step-by-step implementation of the research project

2 System Configuration

2.1 Hardware Specifications

Table 1: Hardware Specification

Processor	Intel Core i7
RAM	8.00 GB
Dis Space	1 TB

2.2 Software Specifications

Table 2: Software Specification

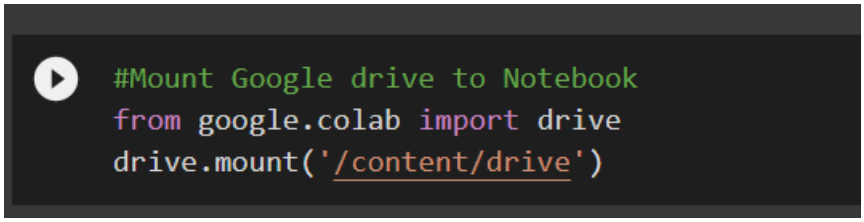
OS	Windows 11
Development Environment	Google Colab Pro
Storage	Google Drive
Libraries	TensorFlow, Keras, Matplotlib, Numpy,TF-Watcher, SKLearn
Scripting Language	Python

3 Project Implementation

I have changed the runtime setting of Google Colab to high RAM and Hardware accelerator to GPU to make the model training faster. The data folder is then moved to google drive and the implementation is divided into two parts which are the CNN model and the Pre-Trained Models. Both parts can be divided into three stages data loading and pre-processing, model building and training and the last stage is model evaluation. The first stage and last stage remain the same for CNN as well as pre-trained models so I will be describing them once to reduce repetitiveness

3.1 Data Loading and pre-processing

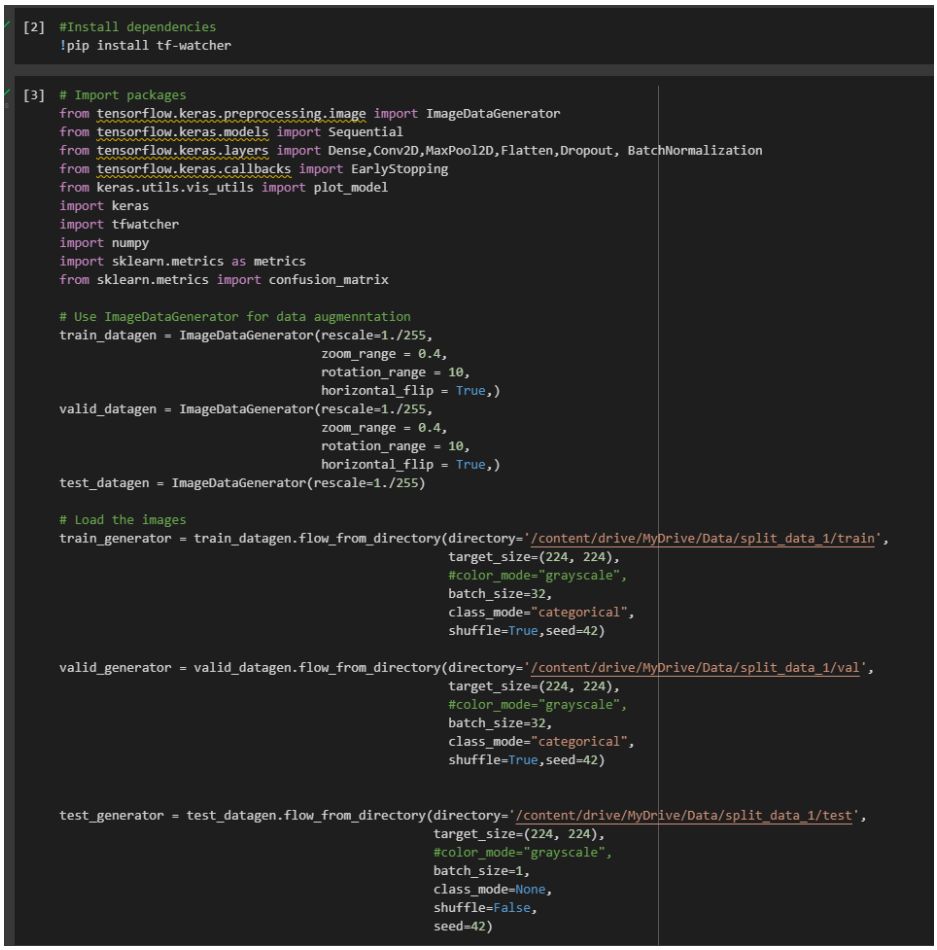
The data folder is first uploaded to google drive and the drive is mounted to the Google Colab notebook.



```
#Mount Google drive to Notebook
from google.colab import drive
drive.mount('/content/drive')
```

Figure 1: Mounting Google Drive.

Install pre-requisites and import required Libraries.
Create ImageDataGeneraor and load the data from Google Drive



```
[2] #Install dependencies
    !pip install tf-watcher

[3] # Import packages
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten,Dropout, BatchNormalization
    from tensorflow.keras.callbacks import EarlyStopping
    from keras.utils.vis_utils import plot_model
    import keras
    import tfwatcher
    import numpy
    import sklearn.metrics as metrics
    from sklearn.metrics import confusion_matrix

    # Use ImageDataGenerator for data augmenntation
    train_datagen = ImageDataGenerator(rescale=1./255,
                                       zoom_range = 0.4,
                                       rotation_range = 10,
                                       horizontal_flip = True,)
    valid_datagen = ImageDataGenerator(rescale=1./255,
                                       zoom_range = 0.4,
                                       rotation_range = 10,
                                       horizontal_flip = True,)
    test_datagen = ImageDataGenerator(rescale=1./255)

    # Load the images
    train_generator = train_datagen.flow_from_directory(directory='/content/drive/MyDrive/Data/split_data_1/train',
                                                        target_size=(224, 224),
                                                        #color_mode="grayscale",
                                                        batch_size=32,
                                                        class_mode="categorical",
                                                        shuffle=True,seed=42)

    valid_generator = valid_datagen.flow_from_directory(directory='/content/drive/MyDrive/Data/split_data_1/val',
                                                        target_size=(224, 224),
                                                        #color_mode="grayscale",
                                                        batch_size=32,
                                                        class_mode="categorical",
                                                        shuffle=True,seed=42)


    test_generator = test_datagen.flow_from_directory(directory='/content/drive/MyDrive/Data/split_data_1/test',
                                                      target_size=(224, 224),
                                                      #color_mode="grayscale",
                                                      batch_size=1,
                                                      class_mode=None,
                                                      shuffle=False,
                                                      seed=42)
```

Figure 2: Data Pre-Processing.

3.2 Model building and training.

3.2.1 CNN Model.

Creating Model Architecture.

```
#Create the Model architecture
model= Sequential()

model.add(Conv2D(filters=128,strides= 1, kernel_size = (5,5), activation='relu', input_shape=(224,224,3,)))
model.add(MaxPool2D(3,3))
model.add(Conv2D(filters=64,kernel_size = (5,5),activation='relu'))
model.add(MaxPool2D(3,3))

model.add(Conv2D(filters=32,kernel_size = (3,3),activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Conv2D(filters=32,kernel_size = (3,3),activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dropout(.1))
model.add(Dense(128,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dropout(.1))
model.add(Dense(8,activation='relu'))
model.add(Dense(4,activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer ='adam', metrics= ['accuracy'])
model.summary()
```

Figure 3: Creating Model.

Defining training parameters and Model Training.

```
[ ]  ###### Hyperparameters
     STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
     STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
     # Stop the training when there is no improvement after 3 epochs trainings.
     early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)

     # Train the model
     history = model.fit(train_generator,steps_per_epoch=STEP_SIZE_TRAIN,
               validation_data=valid_generator,verbose= 1,
               validation_steps=STEP_SIZE_VALID,
               epochs=100, callbacks=early_stop)
```

Figure 4: Model Training.

3.2.2 Pre-Trained Model.

3.2.2.1 VGG16 Model.

Loading the pre-trained base Model

```
#Import base VGG16 model trained on imagenet
IMAGE_SIZE = [224, 224]
vgg16 = VGG16(input_shape= IMAGE_SIZE + [3], weights = 'imagenet', include_top = False)

for layer in vgg16.layers:
    layer.trainable = False

vgg16.summary()
```

Figure 5: Loading pre-trained base Model.

Adding Extra Layers to the model

```
#Add extra layers to VGG16
no_of_categories = 4
x =  AveragePooling2D(pool_size = (4, 4))(vgg16.output)
x = Flatten()(x)
x = Dense(units = 32, activation = 'relu')(x)
x = Dropout(0.2)(x)
x = Dense(units = 512, activation = 'relu')(x)
predictions = Dense(units = no_of_categories, activation = tf.nn.softmax)(x)
model = Model(inputs = vgg16.input, outputs = predictions)
model.summary()
```

Figure 6: Adding layers to the base model.

Defining training parameters and Model Training.

```
[ ] #Create Callback methods for model
    EarlyStoppingCallBack = EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=20)
    CheckPointCallback = ModelCheckpoint("chest-xray.h5", monitor='val_loss',save_best_only=True, mode='min',verbose=1)

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

[ ] #Train the Model
    history_vgg = model.fit(train_generator,
                            epochs = 100,
                            validation_data= valid_generator,
                            callbacks = [EarlyStoppingCallBack, CheckPointCallback],
                            verbose = 1)
```

Figure 7: Model Training.

3.2.2.2 DenseNet121 Model.

Loading the pre-trained base Model

```
##Import base DenseNet121 model trained on imagenet
IMAGE_SIZE = [224, 224]
densenet121 = DenseNet121(input_shape= IMAGE_SIZE + [3], weights = 'imagenet', include_top = False)

for layer in densenet121.layers:
    layer.trainable = False

densenet121.summary()
```

Figure 8: Loading pre-trained base Model.

Adding Extra Layers to the model

```
#Add extra layers to the pre trained model
no_of_categories = 4
x =  AveragePooling2D(pool_size = (4, 4))(densenet121.output)
x = Flatten()(x)
x = Dense(units = 16, activation = 'relu')(x)
x = Dropout(0.2)(x)
x = Dense(units = 512, activation = 'relu')(x)
predictions = Dense(units = no_of_categories, activation = tf.nn.softmax)(x)
model = Model(inputs = densenet121.input, outputs = predictions)
model.summary()
```

Figure 9: Adding layers to the base model.

Defining training parameters and Model Training.

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

```
[ ] #Train the model
densenet_model = model.fit(train_generator,
                           epochs = 100,
                           validation_data= valid_generator,
                           callbacks = [EarlyStoppingCallBack, CheckPointCallback],
                           verbose = 1)
```

Figure 10: Model Training.

3.3 Model Evaluation.

Creating a Function for the calculation of metrics and calling that function.

```
[ ]  # Function to evaluate the model
     def eval_model(model, test_generator):
         test_steps_per_epoch = numpy.math.ceil(test_generator.samples / test_generator.batch_size)
         predictions = model.predict(test_generator, steps=test_steps_per_epoch)
         # Get most likely class
         predicted_classes = numpy.argmax(predictions, axis=1)
         true_classes = test_generator.classes
         class_labels = list(test_generator.class_indices.keys())
         report = metrics.classification_report(true_classes, predicted_classes, target_names=class_labels)
         print(report)
         print(confusion_matrix(true_classes, predicted_classes))

[ ]  #Call Evaluate Function
     eval_model(model, test_generator)
```

Figure 11: Model Evaluation Metrics.

Creating a function for the confusion matrix and calling that function

```
     #Function for confusion Matrix

     def plot_model(model):
         test_steps_per_epoch = numpy.math.ceil(test_generator.samples / test_generator.batch_size)
         predictions = model.predict(test_generator, steps=test_steps_per_epoch)
         # Get most likely class
         predicted_classes = numpy.argmax(predictions, axis=1)
         true_classes = test_generator.classes
         class_labels = list(test_generator.class_indices.keys())
         y_pred = predicted_classes
         y_test = true_classes
         labels = class_labels

         cm = confusion_matrix(y_test, y_pred)

         disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)

         disp.plot(cmap=plt.cm.Blues)
         plt.show()

[ ]  #Creating the confusion Matrix
     plot_model(densenet_model)
     plot_model(vgg16_model)
     plot_model(cnn_model)
```

Figure 12: Plotting Confusion Matrix.