National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Pooja Chopra
Student ID: x20145870

School of Computing
National College of Ireland

Supervisor: Dr. Bharathi Chakravarthi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Pooja Chopra
Student ID:	x20145870
Programme:	Programme Name
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Bharathi Chakravarthi
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	978
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:	
Date:	30th January 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

Pooja Chopra
x20145870

This document contains the hardware, software requirement to replicate this research project with title:"Crack Detection using Edge Detection and Transfer Learning Models".

1 Introduction

This document is the configuration manual for the project titled "Crack Detection using Edge Detection and Transfer Learning Models". This manual contains the details of hardware and software required to replicate the study. It has contain the information about the settings required for the replication. The information about about the different phases of the programming for the implementation of the research work is also given in sequential manner. There are 5 case studies done in this research. The last study is discussed in this manual. However, there other case studies can be done in similar manner.

2 Configration required

The system configuration required in this study are discussed in this section. Hardware and software requirement are also discussed in this section. These configurations helps are pre-requisite for this study.

2.1 Hardware Requirement

This study can be conducted in any system with internet facility as google colab will used for running code for this study.

2.2 Google Colaboratory

Google colab is a Google product.*Colaboratory – Google.* (n.d.) Colab or Colaboratory is a open source plateform. This allow users to excute data analytics, machine learning code through web browser.In this study CPU is used for the execution of the code.

2.2.1 Colab Specification

The specification of colab are give in table 1

Table 1: Colab Specifications

Number of Core	2
RAM	13.6 GB
Maximum lifetime of a VM	12 hours.
Idle VMs time out	90 mins
Runtime Types	CPUs, GPUs, and. TPUs
vendor_id	GenuineIntel
cpu family	6
model	79
model name	Intel(R) Xeon(R) CPU @ 2.20GHz
cpu MHz	2199.998
cache size	56320 KB
address sizes	46 bits physical, 48 bits virtual

2.2.2 Jupiter Notebook

Colab Jupiter notebook is used in this research. It is a open source web application that can used by anybody. Code is written in python langauge.

3 Project Development

This study was conducted using colab Jupiter notebook. We need to open colab web application. [1]. Login using user and password. Go to file and then click on new notebook

3.1 Data loading and mounting

Upload data in google drive.

3.2 How to use data in colab

Mount the Data using the command given in figure 1. Now the google drive path can be used for the analysis in this study.

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

Figure 1: Mounting data

3.3 Importing Libraries

Libraries can be imported using figure 2.

[1] www.https://colab.research.google.com/

```
import os
import os.path
from pathlib import Path
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as test_df
import tensorflow as tf
import keras
import tensorflow.keras as keras
import keras_metrics
import keras_metrics as km
import matplotlib.pyplot as plt
import cv2
import glob
from os import listdir, makedirs
from os.path import isfile, join
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import img_to_array
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Model
from keras.models import Sequential,Model
from keras.layers import Dense, Dropout, InputLayer
from keras.layers import Flatten
import matplotlib.pyplot as plt
from keras.layers import Input,BatchNormalization
```

Figure 2: Importing Libaries

3.4 Data Pre-processing in deck and pavement

Below are pre-processing steps for deck and pavement images. Figure 3 was used for pre-processing pavements cracked images and figure 4 was used for pre-processing pavements cracked images. The same steps can be used for pre-processing deck images.

1. Applying binary threshold function
2. Applying Canny image transformation to find edges
3. Finding contour on canny transformed images.
4. Increased contrast of the image and drawing contour on it
5. Saving pre-processed images in the back to google drive

```
path = r'/content/drive/MyDrive/data/Pavements/Cracked' # Source
dstpath = r'/content/drive/MyDrive/data/Pavements/transforming_crack/Cracked' # Destination
# All files ending with .txt
files= glob.glob("/content/drive/MyDrive/data/Pavements/Cracked/*.jpg")
try:
    makedirs(dstpath)
except:
    print ("Directory already exist, images will be written in BAW folder")
for img in files:
    try:
      image = cv2.imread(img)  # mandrill reference image from USC SIPI
      alpha = 1.5 # Contrast control (1.0-3.0)
      beta = 0 # Brightness control (0-100)
      adjusted = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
      figure,axis = plt.subplots(nrows=1,ncols=6,figsize=(14,14))
      Reading_Img = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
      _,Threshold_Img = cv2.threshold(Reading_Img,127,255,cv2.THRESH_BINARY)
      denoise = cv2.GaussianBlur(Threshold_Img,(5,5),0)
      Canny_Img = cv2.Canny(denoise,90,150)
      contours,_ = cv2.findContours(Canny_Img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
      Draw_Contours = cv2.drawContours(Reading_Img,contours,-1,(255,0,0),1)
      ## Writting transformed images into new folder
      dt=img.replace('Cracked','transforming_crack/Cracked')
      cv2.imwrite(dt,Draw_Contours)
      plt.clf()
    except:
      print ("{} is not converted".format(img))
```

Figure 3: Pre-processing steps in deck and pavement for cracked images

```
path = r'/content/drive/MyDrive/data/Pavements/Non_cracked' # Source
dstpath = r'/content/drive/MyDrive/data/Pavements/transforming_crack/Non_Cracked' # Destination
# All files ending with .txt
files= glob.glob("/content/drive/MyDrive/data/Pavements/Non_cracked/*.jpg")
try:
    makedirs(dstpath)
except:
    print ("Directory already exist, images will be written in BAW folder")

for img in files:
    try:
      image = cv2.imread(img)  # mandrill reference image from USC SIPI
      alpha = 1.5 # Contrast control (1.0-3.0)
      beta = 0 # Brightness control (0-100)
      adjusted = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
      Reading_Img = cv2.cvtColor(adjusted,cv2.COLOR_BGR2GRAY)
      _,Threshold_Img = cv2.threshold(Reading_Img,100,255,cv2.THRESH_BINARY_INV)
      denoise =cv2.blur(Threshold_Img,(10,10))
      Canny_Img = cv2.Canny(denoise,90,150)
      contours,_ = cv2.findContours(Canny_Img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
      Draw_Contours = cv2.drawContours(Reading_Img,contours,-1,(255,0,0),1)
      dt=img.replace('Non_cracked','transforming_crack/Non_Cracked')
      cv2.imwrite(dt,Draw_Contours)
      plt.clf()
    except:
      print ("{} is not converted".format(img))
```

Figure 4: Pre-processing steps in deck and pavement for cracked images

3.5 Data Pre-processing in walls

Below are pre-processing steps for walls images. Figure 5 was used for pre-processing cracked walls images. The same steps can be used for pre-processing non-cracked walls images.

1. De-noising Images using a median filter.
2. Applying adaptive threshold function to a denoised image.
3. Applying Canny image transformation to find edges.
4. Finding contour on canny transformed images.
5. Drawing contour on its original images
6. Saving pre-processed images in the back to google drive

```
path = r'/content/drive/MyDrive/data/Walls/Cracked' # Source
dstpath = r'/content/drive/MyDrive/data/Walls/transforming_crack/Cracked' # Destination
# All files ending with .txt
files= glob.glob("/content/drive/MyDrive/data/Walls/Cracked/*.jpg")
try:
    makedirs(dstpath)
except:
    print ("Directory already exist, images will be written in BAW folder")
for img in files:
    try:
      image = cv2.imread(img,0)# mandrill reference image from USC SIPI
      denoise = cv2.medianBlur(image,5)
      Threshold_Img = cv2.adaptiveThreshold(denoise,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,2)
      dt=img.replace('Cracked','transforming_crack/Cracked')
      cv2.imwrite(dt,Threshold_Img)
    except:
      print ("{} is not converted".format(img))
```

Figure 5: Pre-processing steps in walls

3.6 Loading transformed images in dataframes

Code in figure 6 was used to loading pavement images into the dataframe. Same process can be used for loading deck and walls images. Images were downsampled for balancing data before making dataframes. Dataframes for decks, pavement and walls were joined using figure 7.

```
Surface_Data = Path("/content/drive/MyDrive/data/Pavements/transforming_crack")
Surface_JPG_Path = list(Surface_Data.glob(r"*/*.jpg"))
Surface_Labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],Surface_JPG_Path))
Surface_JPG_Path_Series = pd.Series(Surface_JPG_Path,name="JPG").astype(str)
Surface_Labels_Series = pd.Series(Surface_Labels,name="CATEGORY")
Main_Surface_Data = pd.concat([Surface_JPG_Path_Series,Surface_Labels_Series],axis=1)
Main_Surface_Data = Main_Surface_Data.sample(frac=1).reset_index(drop=True)
Positive_Surface = Main_Surface_Data[Main_Surface_Data["CATEGORY"] == "Cracked"]
Negative_Surface = Main_Surface_Data[Main_Surface_Data["CATEGORY"] == "Non_Cracked"]
Negative_Surface = Negative_Surface.sample(n = 2608) ###Downsampling the images as data is imbalanced
frames = [Negative_Surface, Positive_Surface]
df1 = pd.concat(frames)
```

Figure 6: Loading transformed images in dataframes

```
frame = [df1, df2, df3]
df = pd.concat(frame)
df = df.sample(frac=1).reset_index(drop=True)
```

Figure 7: Joining dataframes

3.7 Splitting test, train and validation dataset

Train and test dataset was splitted into 80% and 20% ratio. Train dataset was again splitted into train and validation in 80% and 20% ratio. The code can be seen in figure 8

```
train_df,test_df=train_test_split(df,test_size=0.2,shuffle=True,random_state=101)
train_gen=tf.keras.preprocessing.image.ImageDataGenerator(
                                    rescale=1./255,validation_split=0.2)
test_gen=tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
Train_Set=train_gen.flow_from_dataframe(train_df,
                                        x_col="JPG",
                                        y_col="CATEGORY",target_size=(128,128),
                                        color_mode="rgb",class_mode="categorical",batch_size=32,
                                        shuffle=True,seed=42,subset="training")
Validation_Set=train_gen.flow_from_dataframe(train_df,x_col="JPG",
                                        y_col="CATEGORY",target_size=(128,128),
                                        color_mode="rgb",class_mode="categorical",batch_size=32,
                                        shuffle=True,seed=42,subset="validation")
Test_Set=test_gen.flow_from_dataframe(test_df,
                                      x_col="JPG",
                                        y_col="CATEGORY",target_size=(128,128),
                                        color_mode="rgb",class_mode="categorical",batch_size=32,
                                      shuffle=False,seed=42)
```

Figure 8: Splitting test, train and validation dataset

3.8 Building Models

3 models were applied in case study 6. Xception, VGG19 and Resnet50 models were applied to the dataset.

3.8.1 Xception Model

Figure 9 contains xception model building code.

```
xception = Xception(include_top=False, input_shape=(256, 256, 3))

x = xception.output
x = layers.GlobalMaxPooling2D()(x)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense(512, activation='relu')(x)
output = layers.Dense(2, activation='softmax')(x)

model = Model(xception.input, output)

# Freezing all the Imported Layers
for layers in xception.layers:
    layers.trainable = False
```

Figure 9: Xception model building

3.8.2 Resnet50 Model

Figure 10 contains Resnet50 model building code.

```
model_resnet = Sequential()
model_resnet.add(model_res)
model_resnet.add(Flatten())
model_resnet.add(BatchNormalization())
model_resnet.add(Dense(256, activation='relu'))
model_resnet.add(Dropout(0.5))
model_resnet.add(BatchNormalization())
model_resnet.add(Dense(128, activation='relu'))
model_resnet.add(Dropout(0.5))
model_resnet.add(BatchNormalization())
model_resnet.add(Dense(2, activation='softmax')) # [0.9,0.1]
model_resnet.layers[0].trainable = False
```

Figure 10: Resnet50 Model building

3.8.3 VGG19 Model

Figure 11 contains VGG19 model building code.

```
model_vgg = Sequential()
model_vgg.add(model_19)
model_vgg.add(Flatten())
model_vgg.add(BatchNormalization())
model_vgg.add(Dense(256, activation='relu'))
model_vgg.add(Dropout(0.5))
model_vgg.add(BatchNormalization())
model_vgg.add(Dense(128, activation='relu'))
model_vgg.add(Dropout(0.5))
model_vgg.add(BatchNormalization())
model_vgg.add(Dense(2, activation='softmax'))

model_vgg.layers[0].trainable = False
```

Figure 11: VGG19 Model Building

3.9 Model training

3 different models were trained using VGG19, Resnet50 and Xception Model. Figure 12 contain code for training the model.

```
# distribution
model_resnet.compile(optimizer='adam', loss="categorical_crossentropy",
                     metrics=["accuracy",km.precision(), km.recall(),km.f1_score()])
# train the network
Early_Stopper = tf.keras.callbacks.EarlyStopping(monitor="val_recall",patience=4,mode="auto")
Checkpoint_Model = tf.keras.callbacks.ModelCheckpoint(monitor="val_recall",
                                                      save_best_only=True,
                                                      save_weights_only=True,
                     filepath="/content/drive/MyDrive/data/log/resnet/modelcheck")
hist_res = model_resnet.fit(Train_Set,
                 validation_data=Validation_Set, callbacks=[Checkpoint_Model],
                 steps_per_epoch=25, validation_steps=25, epochs=10)
```

Figure 12: Model training

3.10 Choosing epoch for training

Epoch were chosen based on the loss value of the validation dataset. If loss in the validation set continuously decreased, then epochs were stopped. Figure 13 can be used for checking metrics in training and validation datasets.

```
def plot(hist,name,EPOCHS):
    num_epoch = [i for i in range(EPOCHS)]
    plt.figure(figsize=(20,15))
    plt.title("{} ".format(name))
    plt.subplot(231)
    plt.title("Accuracy")
    plt.plot(num_epoch,hist.history["accuracy"],label="train")
    plt.plot(num_epoch,hist.history["val_accuracy"],label="validation")
    plt.legend(loc="best")
    plt.subplot(232)
    plt.title("Loss")
    plt.plot(num_epoch,hist.history["loss"],label="train")
    plt.plot(num_epoch,hist.history["val_loss"],label="validation")
    plt.legend(loc="best")
    plt.subplot(233)
    plt.title("Recall")
    plt.plot(num_epoch,hist.history["recall"],label="train")
    plt.plot(num_epoch,hist.history["val_recall"],label="validation")
    plt.legend(loc="best")
    plt.subplot(234)
    plt.title("Precision")
    plt.plot(num_epoch,hist.history["precision"],label="train")
    plt.plot(num_epoch,hist.history["val_precision"],label="validation")
    plt.legend(loc="best")
    plt.subplot(235)
    plt.title("F1-Score")
    plt.plot(num_epoch,hist.history["f1_score"],label="train")
    plt.plot(num_epoch,hist.history["val_f1_score"],label="validation")
    plt.legend(loc="best")
    plt.show()
```

Figure 13: Checking epoch and validation parameters

3.11 Prediction

Code in figure 14 was used for test the efficiency of the training model.

```
print('Classification Report')
print(classification_report(Test_Set.classes, Model_Test_Prediction_vgg))
```

Figure 14: Prediction

3.12 Evaluation

The models were evaluated using precision, recall, f1 score and accuracy. The overall accuracy of the model can be checked using code in Figure 15. The figure 16 contain code for checking the precision, recall, f1 score and accuracy of test dataset. Figure 17 contains the information about the True Positive value i.e. images that are actually true and predicted true the model.

```
Model_Results = model.evaluate(Test_Set)
print("LOSS:  " + "%.4f" % Model_Results[0])
print("ACCURACY:  " + "%.2f" % Model_Results[1])
```

Figure 15: Checking overall accuracy and loss

```
print('Classification Report')
print(classification_report(Test_Set.classes, Model_Test_Prediction_vgg))
```

Figure 16: Classification report

```
cm=confusion_matrix(Test_Set.classes, Model_Test_Prediction_vgg)
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
plt.xticks(ticks=np.arange(2) + 0.5, labels=["NEGATIVE", "POSITIVE"])
plt.yticks(ticks=np.arange(2) + 0.5, labels=["NEGATIVE", "POSITIVE"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Figure 17: Confusion Matrix

Same steps can be followed to replicate the rest of the case study.

References

Colaboratory – Google. (n.d.). `http:https://research.google.com/colaboratory/faq.html`. Accessed: 2020-12-15.