Configuration Manual

MSc Research Project
Data Analytics

Ananya Pratap Singh Chandel
Student ID: 19237529

School of Computing
National College of Ireland

Supervisor: Prof. Noel Cosgrave

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ananya Pratap Singh Chandel …. …………………………………………………………………………………………
Student ID:	x19237529……..………………………………………………………………………..……
Programme:	Data Analytics……………………………………… **Year:** …2020-2021…..
Module:	MSc Research project ….……………………………………………………….………
Lecturer:	Noel Cosgrave …………………………………………………………………….………
Submission Due Date:	16/08/2021……………………………………………………………………………….………
Project Title:	Detecting Diabetic Retinopathy from Retinal fundus images using DC-CNN ……………………………………………………………………………….………
Word Count:	903……………………………… **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: ……………………………………………………………………………………………………

Date: ……………………………………………………………………………………………………

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

Ananya Pratap Singh Chandel
Student ID: 19237529

1 Introduction

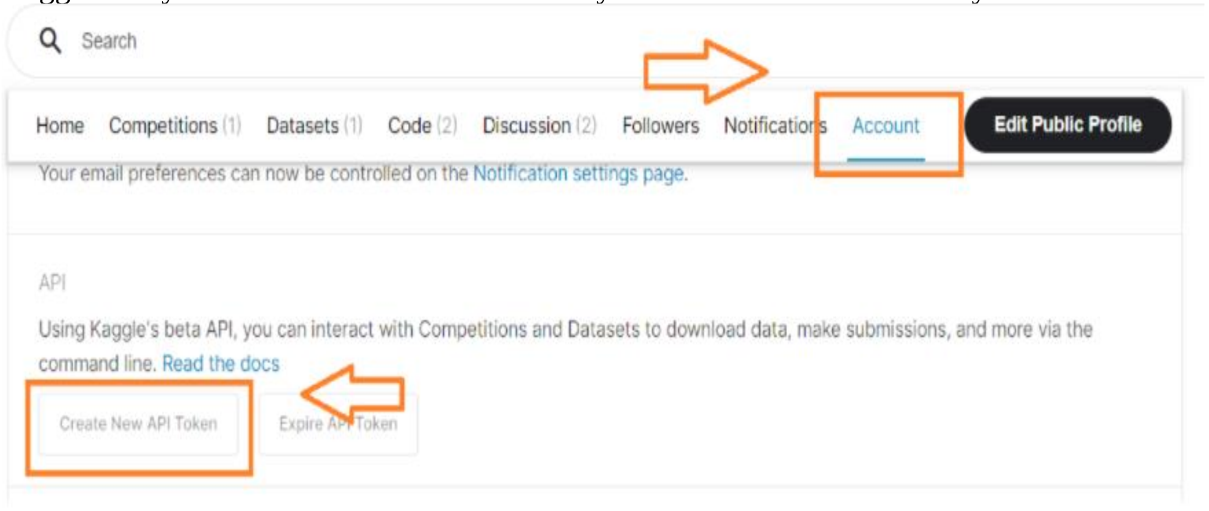
This research project detects diabetic retinopathy using retinal fundus images by designing a novel dual-channel convolutional neural network. All the steps that may be required for the replication are mentioned under this configuration manual. Project design flow from data collection to model evaluation is explained. Code snippets from different sections are also added as per requirement.

2 System Configuration

To carry out the implementation of this project, Google Colaboratory has been used. Google provides free-to-use cloud-based machines that are used to run python codes but with certain limitations. Google Collab Pro can be used with added GPU usage and ram utilization. System configuration of google Collaboratory used for this project is GPU(Tesla K80) having 2496 cuda cores and 12GB DDR5 VRAM, CPU hyper-threaded Xeon Processors @2.3Ghz, disk 100 GB available, and ram 12.6 GB available.

3 Data Collection

The dataset used for this research has been taken from Kaggle. Kaggle allows downloading data by using API token that can be obtained from the profile section of the Kaggle account as can be seen in figure 1 below. After that Create a Google Colab notebook and connect it to the cloud (basically start the notebook interface). After that, upload the "kaggle.json" file that Kaggle sent you. This allows data to be directly loaded into the Colaboratory environment.


Your email preferences can now be controlled on the Notification settings page.

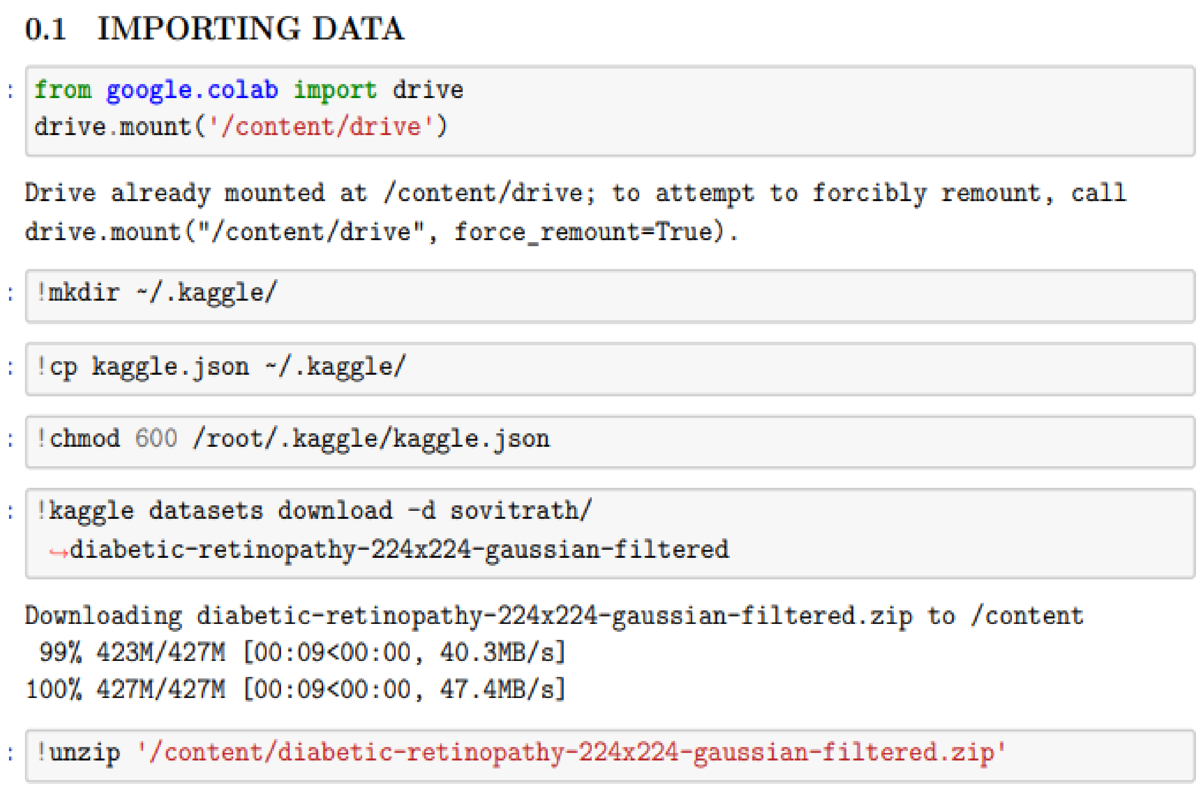
API

Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. Read the docs

Figure 1: Downloading kaggle API

4 Setting up Google Colaboratory

Alternatively, a dataset can also be downloaded and stored in Google Drive and can be accessed by Google colaboratory anytime. Before reading the data, Google drive must be mounted. Figure 2 shows the mounting of google drive and also reading data using Kaggle.json file. You may be asked to authenticate your account to access drive through google collab on mounting google drive. For faster performance runtime environment can be changed to GPU. The coding is done using python version 3.9.3, and google colab provides ready to use jupyter notebook environment for coding.



0.1 IMPORTING DATA

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

```
!mkdir ~/.kaggle/
```

```
!cp kaggle.json ~/.kaggle/
```

```
!chmod 600 /root/.kaggle/kaggle.json
```

```
!kaggle datasets download -d sovitrath/
 ↪diabetic-retinopathy-224x224-gaussian-filtered
```

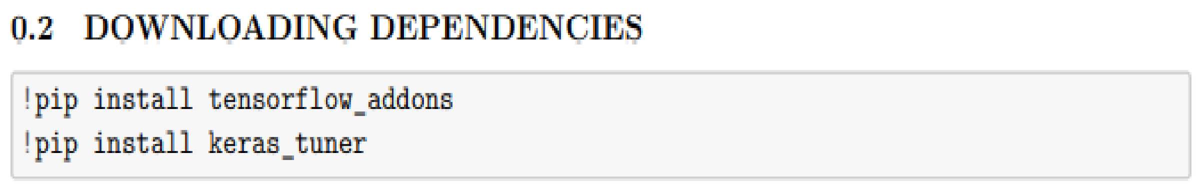
```
Downloading diabetic-retinopathy-224x224-gaussian-filtered.zip to /content
 99% 423M/427M [00:09<00:00, 40.3MB/s]
100% 427M/427M [00:09<00:00, 47.4MB/s]
```

```
!unzip '/content/diabetic-retinopathy-224x224-gaussian-filtered.zip'
```

Figure 2: Shows mounting of google drive and reading data through kaggle API

5 Installing Python Libraries

All the necessary packages can be installed using !pip install commands in google colab, as can be seen in figure 3 below.



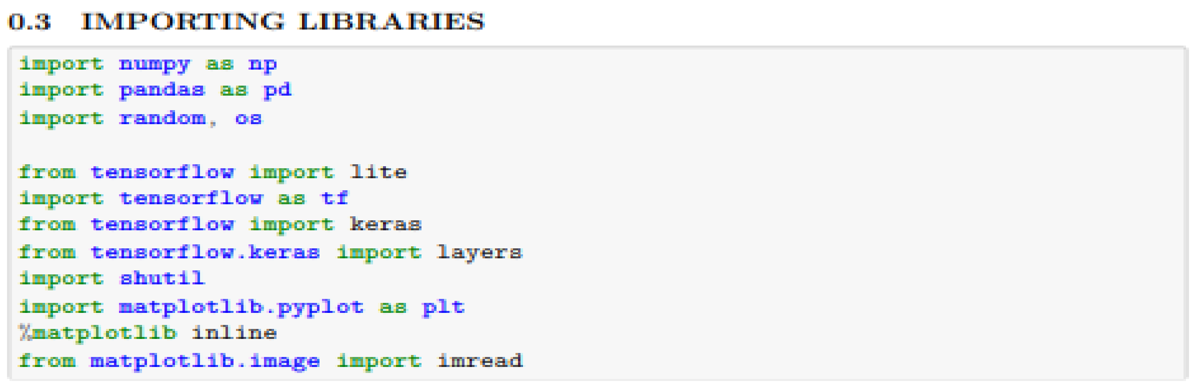
0.2 DOWNLOADING DEPENDENCIES

```
!pip install tensorflow_addons
!pip install keras_tuner
```

Figure 3: Installing dependencies

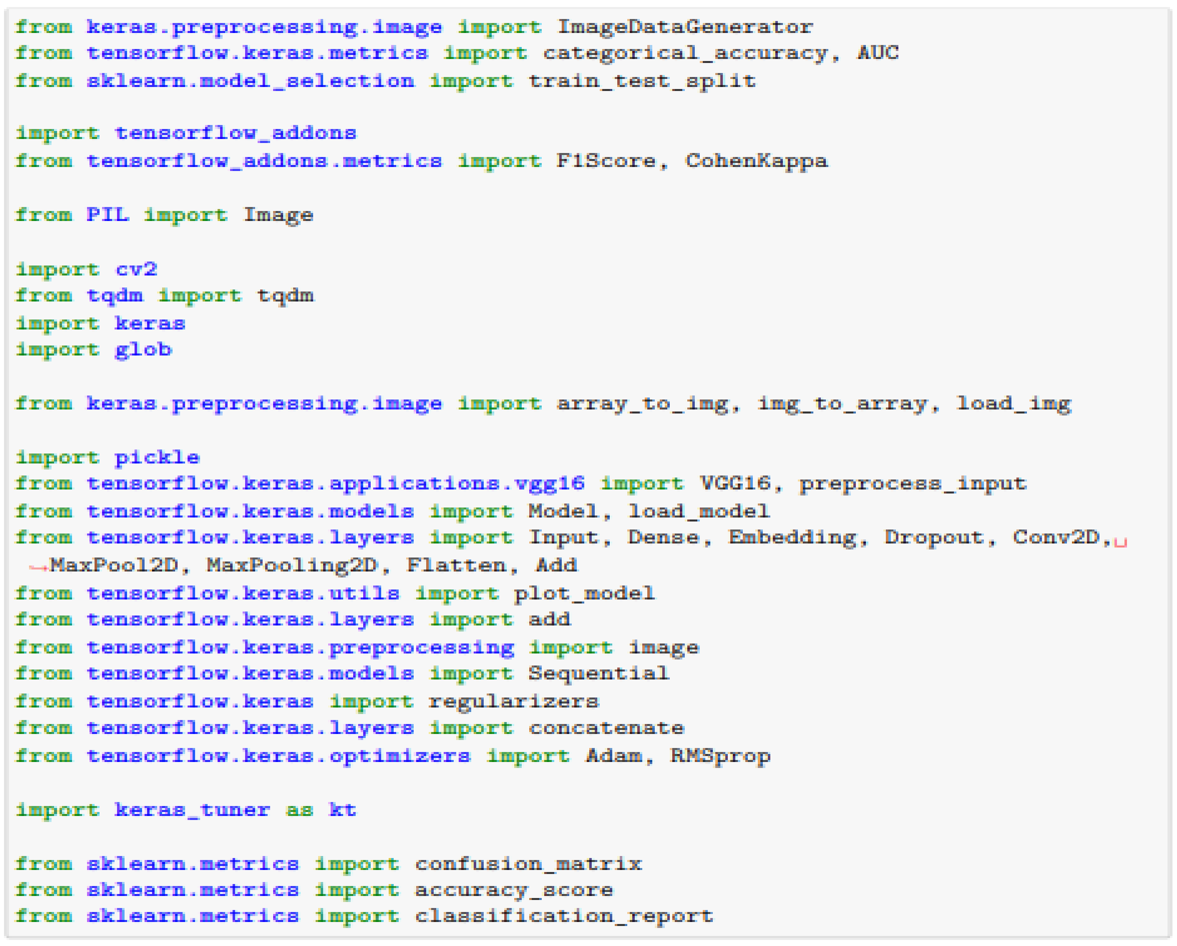
The Figure 4 below represents all the necessary libraries that need to be imported for carrying out the execution of the project. The list of packages that need to be installed prior to the execution of code is as follows:

Numpy Version: 1.19.5
Pandas Version: 1.1.5
Tensorflow Version: 2.5.0
Matplotlib Version: 3.2.2
Sklearn Version: 0.22.2.post1
Keras Version: 2.5.0
OpenCV Version: 4.1.2
Keras Tuner Version: 1.0.3



0.3 IMPORTING LIBRARIES

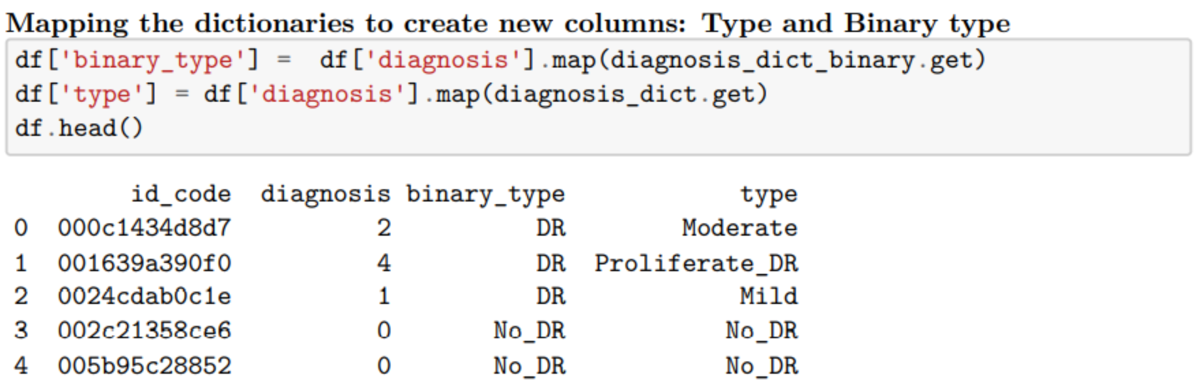
```
import numpy as np
import pandas as pd
import random, os

from tensorflow import lite
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import shutil
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.image import imread
```



```
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.metrics import categorical_accuracy, AUC
from sklearn.model_selection import train_test_split

import tensorflow_addons
from tensorflow_addons.metrics import F1Score, CohenKappa

from PIL import Image

import cv2
from tqdm import tqdm
import keras
import glob

from keras.preprocessing.image import array_to_img, img_to_array, load_img

import pickle
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense, Embedding, Dropout, Conv2D,␣
 ↪MaxPool2D, MaxPooling2D, Flatten, Add
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import add
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras import regularizers
from tensorflow.keras.layers import concatenate
from tensorflow.keras.optimizers import Adam, RMSprop

import keras_tuner as kt

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

6 Data Preprocessing

The data preprocessing code can be seen from the code snippet below, which uses pandas data frame .



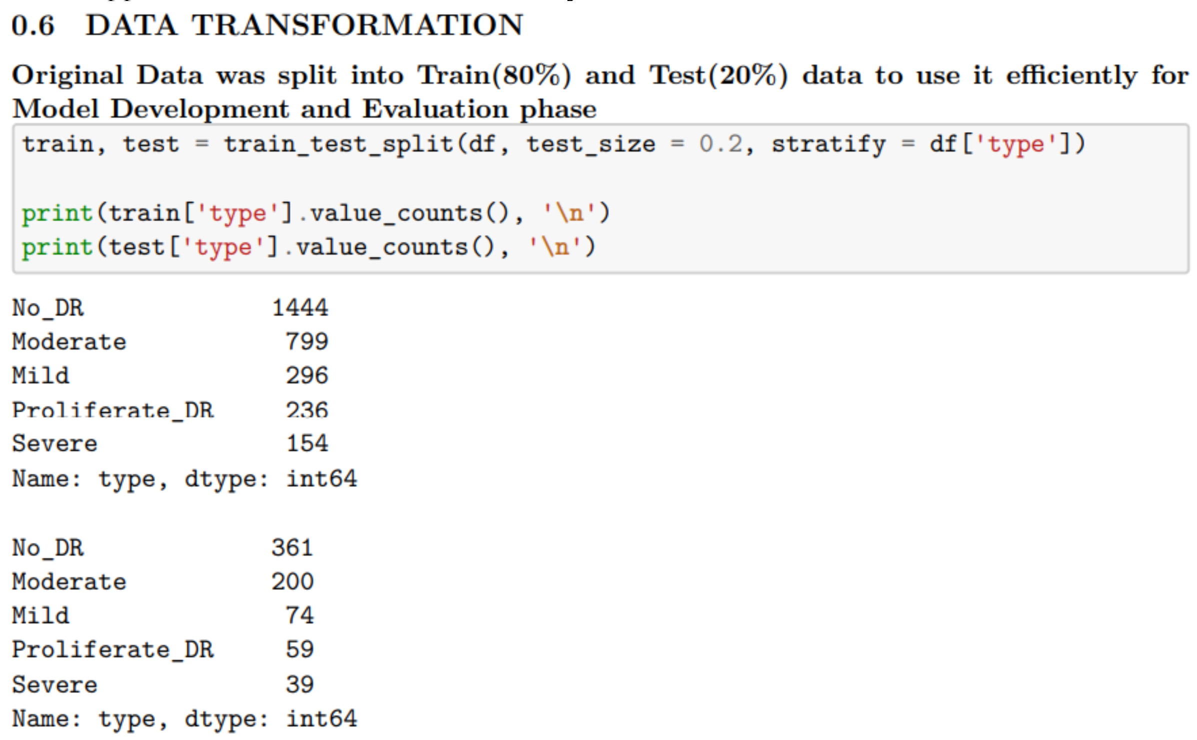
Mapping the dictionaries to create new columns: Type and Binary type

```
df['binary_type'] =  df['diagnosis'].map(diagnosis_dict_binary.get)
df['type'] = df['diagnosis'].map(diagnosis_dict.get)
df.head()
```

```
        id_code  diagnosis binary_type            type
0  000c1434d8d7          2          DR        Moderate
1  001639a390f0          4          DR  Proliferate_DR
2  0024cdab0c1e          1          DR            Mild
3  002c21358ce6          0       No_DR           No_DR
4  005b95c28852          0       No_DR           No_DR
```

7 Data Transformation

The code for data transformation and splitting the data into train test split can be seen in the code snippet below with utilizes clean library.



0.6 DATA TRANSFORMATION

Original Data was split into Train(80%) and Test(20%) data to use it efficiently for Model Development and Evaluation phase

```
train, test = train_test_split(df, test_size = 0.2, stratify = df['type'])

print(train['type'].value_counts(), '\n')
print(test['type'].value_counts(), '\n')
```

```
No_DR             1444
Moderate           799
Mild               296
Proliferate_DR     236
Severe             154
Name: type, dtype: int64

No_DR             361
Moderate          200
Mild               74
Proliferate_DR     59
Severe             39
Name: type, dtype: int64
```

8 Data Visualization

The below code shows a snippet for visualizing the retinal fundus Images.

0.6.1 Visualization of DR and Non-DR Images

```
f = plt.figure(figsize=(12,6))
# creating a object
im1 = Image.open(r"/content/New_DR/train/DR/001639a390f0.png")
im2 = Image.open(r"/content/New_DR/train/No_DR/002c21358ce6.png")


f.add_subplot(1,2, 1)
plt.title('DR')
plt.imshow(im1)
```

9 Data Augmentation

The below code shows the implementation of the Keras image data generator for performing data augmentation.

```
def train_data_gen(path):
  datagen = ImageDataGenerator(rescale = 1./255,shear_range = 0.1,zoom_range =␣
 ↪0.1,horizontal_flip = True)
  train_dr_path = glob.glob(path)
  train_dr_label = train_dr_path[0].split("/")[-1]
  for image_path in glob.glob(os.path.join(train_dr_path[0], "*.png")):
      img = load_img(image_path)
      x = img_to_array(img)
      x = x.reshape((1,) + x.shape)
      i = 0
      for batch in datagen.flow(x, batch_size=1,save_to_dir=(os.path.join("/
 ↪content/New_DR/train/",train_dr_label))+'/', save_prefix=train_dr_label,␣
 ↪save_format='png'):
          i += 1
          if i > 5:
              break
```

10 Keras tuner

Hyperparameter optimization is performed using the Keras tuner library. A random search function from the Keras library is used to tune one of the channels in the network.

The Random Search from Keras Tuner was implemented to find the best model by considering Validation Loss as the objective function to compare the different models.

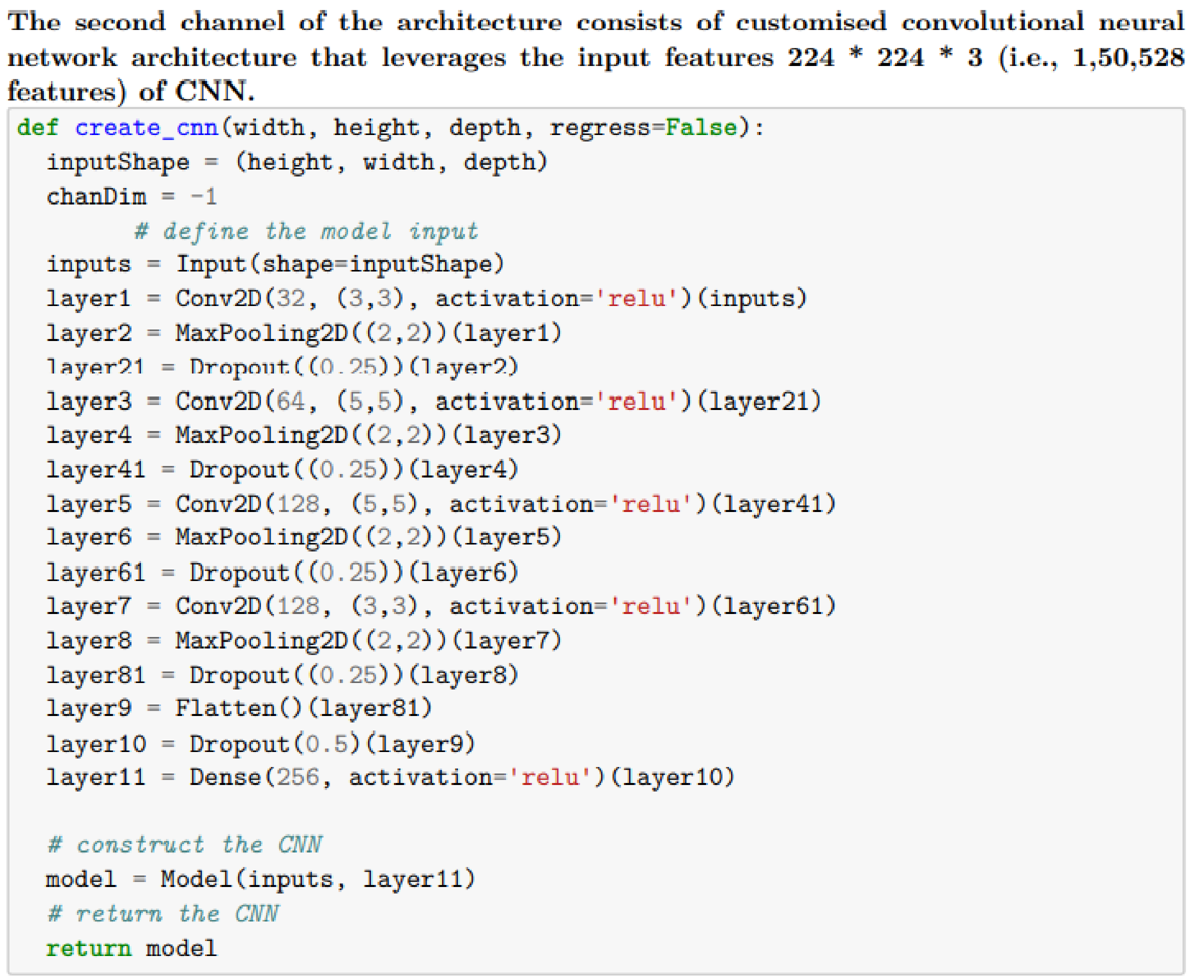
```
tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=5,
    overwrite=True)
```

The Keras Tuner was fed with the original Train and Test images based on which it provided us with the best model with least validation loss.

```
tuner.search(np.array(X_train_cnn), np.array(y_train), epochs=20,␣
 ↪validation_data=(X_test_cnn, y_test))
best_model_1 = tuner.get_best_models()[0]
```

11 Defining CNN

The below code snippet gives an idea about defining a function to create a CNN.



The second channel of the architecture consists of customised convolutional neural network architecture that leverages the input features 224 * 224 * 3 (i.e., 1,50,528 features) of CNN.

```
def create_cnn(width, height, depth, regress=False):
  inputShape = (height, width, depth)
  chanDim = -1
        # define the model input
  inputs = Input(shape=inputShape)
  layer1 = Conv2D(32, (3,3), activation='relu')(inputs)
  layer2 = MaxPooling2D((2,2))(layer1)
  layer21 = Dropout((0.25))(layer2)
  layer3 = Conv2D(64, (5,5), activation='relu')(layer21)
  layer4 = MaxPooling2D((2,2))(layer3)
  layer41 = Dropout((0.25))(layer4)
  layer5 = Conv2D(128, (5,5), activation='relu')(layer41)
  layer6 = MaxPooling2D((2,2))(layer5)
  layer61 = Dropout((0.25))(layer6)
  layer7 = Conv2D(128, (3,3), activation='relu')(layer61)
  layer8 = MaxPooling2D((2,2))(layer7)
  layer81 = Dropout((0.25))(layer8)
  layer9 = Flatten()(layer81)
  layer10 = Dropout(0.5)(layer9)
  layer11 = Dense(256, activation='relu')(layer10)

  # construct the CNN
  model = Model(inputs, layer11)
  # return the CNN
  return model
```

12 Visualising performance

After model has been trained the performance of the model is plotted using learning curves same can be seen below.

Accuracy and Loss plot of DC-CNN performed on Training and Testing Data without Augmentation

```
# summarize history for accuracy
plt.plot(history.history['accuracy'][1:])
plt.plot(history.history['val_accuracy'][1:])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'][1:])
plt.plot(history.history['val_loss'][1:])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

13 Performing Predictions

The code snippet below defined a function to perform predictions on the trained model using the test data.

0.14 MODEL PREDICTIONS

Creating a function to generate the predictions of our DC-CNN model

```
def generate_predictions(path):
  X_data = []
  files = glob.glob (path)
  for myFile in files:
    # print(myFile)
    image = cv2.imread (myFile)
    X_data.append (image)
  X_test_data_vgg = extract_features1(X_data)
  X_test_data_cnn = np.array(X_data)

  preds = model.predict([X_test_data_vgg, X_test_data_cnn])

  return preds
```

14 Evaluating results

After the results are predicted, the below code snippet is used to visualize the results.

Further analysis is also performed to identify few extra parameters for evaluation.

```
conf_matrix = [[350,24],[11,348]]


# Creating a function to report confusion metrics
def confusion_metrics (conf_matrix):
# save confusion matrix and slice into four pieces
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    print('True Positives:', TP)
    print('True Negatives:', TN)
    print('False Positives:', FP)
    print('False Negatives:', FN)

    # calculate accuracy
    conf_accuracy = (float (TP+TN) / float(TP + TN + FP + FN))

    # calculate mis-classification
    conf_misclassification = 1- conf_accuracy

    # calculate the sensitivity
    conf_sensitivity = (TP / float(TP + FN))
    # calculate the specificity
    conf_specificity = (TN / float(TN + FP))

    # calculate precision
    conf_precision = (TN / float(TN + FP))
    # calculate f_1 score
    conf_f1 = 2 * ((conf_precision * conf_sensitivity) / (conf_precision +␣
↪conf_sensitivity))
    print('-'*50)
    print(f'Accuracy: {round(conf_accuracy,4)}')
    print(f'Mis-Classification: {round(conf_misclassification,4)}')
    print(f'Sensitivity: {round(conf_sensitivity,4)}')
```