Configuration Manual

MSc Research Project
Data Analytics

Rohit Kumar
Student ID: X15004902

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rohit Kumar
Student ID:	X15004902
Programme:	Data Analytics **Year:** 2021
Module:	MSc Research Project
Lecturer:	Jorge Basilio
Submission Due Date:	23/09/2021
Project Title:	Intracranial hemorrhage Detection Using Deep Learning and Transfer Learning
Word Count:	……………………… **Page Count:** …………………………………….……

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: Rohit Kumar

Date: 23/09/2021

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

Rohit Kumar
X15004902

1 Introduction

The objective of this paper is to outline the process used to code the project. The hardware and software combinations necessary to replicate the future research are described. This section details the programming and implementation processes necessary for efficient executable code, as well as the actions necessary to run the script.

2 System Configuration

2.1 Hardware Configuration

The Below figure 1 shows the hardware details used to execute the code.

Device specifications

HP Laptop 14s-dq1xxx

Device name	LAPTOP-FLPGDREF
Processor	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
Installed RAM	8.00 GB (7.70 GB usable)
Device ID	A7124827-360C-442C-84DF-429FD39641E7
Product ID	00325-81902-90277-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Fig. 1 Device specifications

2.2 Software Configuration

This section contains information about the software standards that were used.

2.2.1 Jupyter notebook in Anaconda:

Anaconda is a Python coding platform that is free, open source, and simple to use. The anaconda prompt screen in the base root environment is depicted in the following Figure 2.

The TensorFlow environment has been chosen for the purpose of executing the CNN model and Transfer learning model.

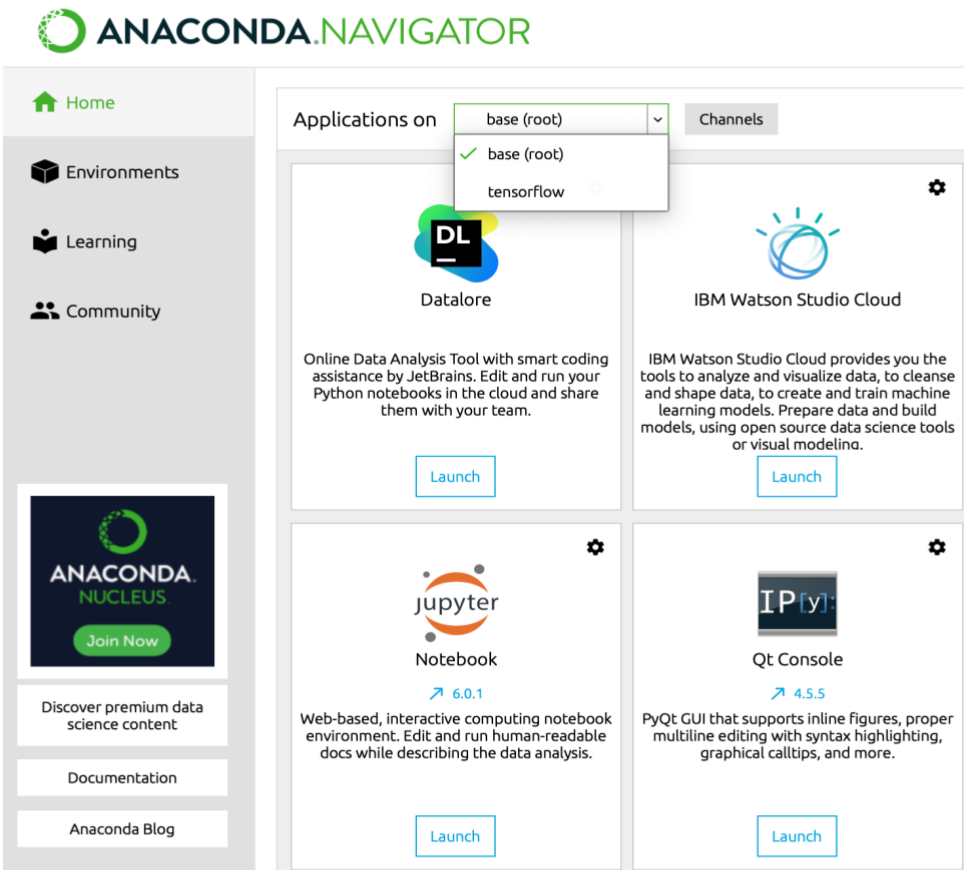


Figure 2: Anaconda prompt.

3 Data Gathering

The Dataset available on Kaggle is 427.45GB and can is downloaded from Kaggle using the Kaggle API.

1. The first step is to set the directory and download the Kaggle in the Jupyter file as shown in the below image.

```
In [1]: import os
        Dirpath ="C:/Users/Rohit/Thesis"
        os.chdir(Dirpath)
```

```
In [2]: pip install kaggle
```

2. Next step is to login on your Kaggle account and go to my account page and click on create new API token.

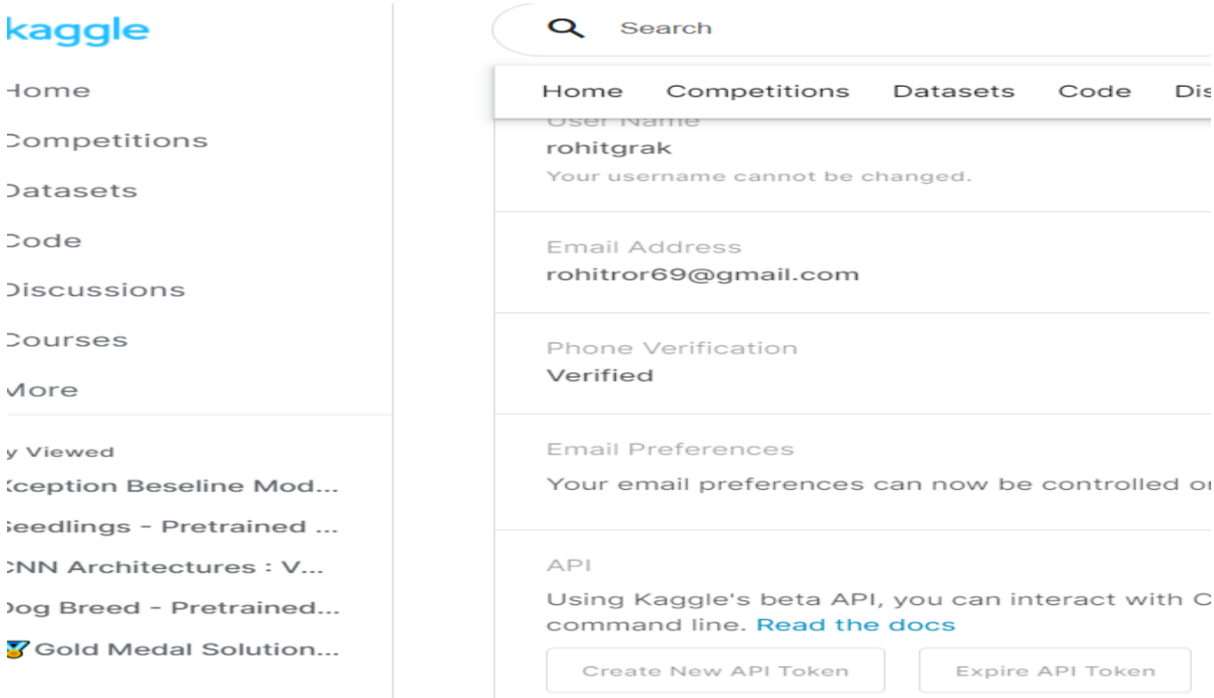


Fig. 3 Kaggle my Account page

3. Once we click on Create New API Token the Kaggle.json file will be download on the machine.
4. To download the dataset, store the Kaggle.json file in the Kaggle folder on the C drive. When we install Kaggle in Jupyter, it displays the downloaded file path, therefore we must save the Kaggle.json file in that location.
5. Next step is to visit on the link https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection/data and copy the dataset API. The below figure 4 shows the dataset page with the API.

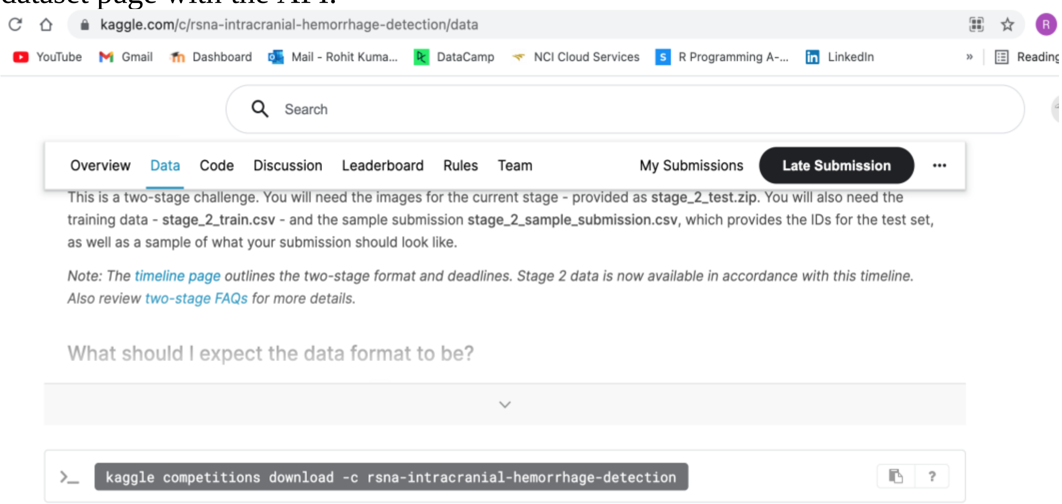


Fig. 4 RSNA Dataset Page.

6. The dataset can be downloaded using 2, option 1 is open anaconda command prompt and paste the API key and click enter as shown in the figure 5 and another method is to use Jupyter Notebook and add the Kaggle username and Password and then Run the API as shown in the Figure 6.

```
(base) C:\Users\Rohit>cd Thesis

(base) C:\Users\Rohit\Thesis>kaggle competitions download -c rsna-intracranial-hemorrhage-detection
Downloading rsna-intracranial-hemorrhage-detection.zip to C:\Users\Rohit\Thesis
100%|                                        | 181G/181G [5:30:50<00:00, 9.77MB/s]
100%|                                        | 181G/181G [5:30:50<00:00, 9.81MB/s]

(base) C:\Users\Rohit\Thesis>
```

Fig5 Anaconda Command Prompt

```
from keras import backend as K
import tensorflow as tf
```

```
Using TensorFlow backend.
```

```
In [7]: # Set environment variables for using the Kaggle API.
os.environ["KAGGLE_USERNAME"] = "Account_username"
os.environ["KAGGLE_KEY"] = "Account_Key"
```

```
In [8]: !kaggle competitions download -c rsna-intracranial-hemorrhage-detection

Downloading rsna-intracranial-hemorrhage-detection.zip to {raw_data_dir}
  0%|                                        | 524M/181G [01:50<7:16:35, 7.41MB/s]^C
  0%|                                        | 525M/181G [01:51<10:52:45, 4.96MB/s]
```

Fig. 6 Jupyter Notebook

4 Data Conversion

First step: Unzip the downloaded dataset in the hard disk. Because the data is so big and I do not have enough space in my device so I have used an external hard disk.

Second step: open the Intracranial_Hemorrhage_Detection.ipynb file and load the dataset files as shown in the below code

```
In [1]: ######### importing Library to read the files

import pandas as pd
import numpy as np
```

```
In [2]: Data_PATH = 'D:/rsna-intracranial-hemorrhage-detection/'

TRAIN_Image = 'stage_2_train/'
TEST_Image = 'stage_2_test/'
train_CSV = pd.read_csv(Data_PATH + 'stage_2_train.csv')
subm_CSV = pd.read_csv(Data_PATH + 'stage_2_sample_submission.csv')
```

Third Step: In this step reading and train label file and converting into data frame.

```
train_CSV['filename'] = train_CSV['ID'].apply(lambda st: "ID_" + st.split('_')[1] + ".png")
train_CSV['type'] = train_CSV['ID'].apply(lambda st: st.split('_')[2])
subm_CSV['filename'] = subm_CSV['ID'].apply(lambda st: "ID_" + st.split('_')[1] + ".png")
subm_CSV['type'] = subm_CSV['ID'].apply(lambda st: st.split('_')[2])

print(train_CSV.shape)
train_CSV.head()
```

Fourth Step: Due to the limited hardware available for implementation, I chose to convert 50,000 DICOM images from the train folder and 5,000 DICOM images from the test folder.

Fifth Step: Duplicate photos are deleted from the training data, and 15 percent of the training data is separated into validation data, which is then used to further validate the model after it has been trained.

Sixth Step: importing the import libraries for windowing and image conversion.

```
In [19]: # Importing Library for windowing and conversion

         import json
         import cv2
         import pydicom
         from tqdm import tqdm
```

Seventh Step: once the libraries imported get the pixel of the image and save if in a function.

```
In [20]: def get_pixels_hu(scan):
             image = np.stack([scan.pixel_array])
             image = image.astype(np.int16)

             image[image == -2000] = 0

             intercept = scan.RescaleIntercept
             slope = scan.RescaleSlope

             if slope != 1:
                 image = slope * image.astype(np.float64)
                 image = image.astype(np.int16)

             image += np.int16(intercept)

             return np.array(image, dtype=np.int16)
```

Eighth Step: 3-channel windowing of Brain, Subdural and bone is done in this step

```python
def apply_window(image, center, width):
    image = image.copy()
    min_value = center - width // 2
    max_value = center + width // 2
    image[image < min_value] = min_value
    image[image > max_value] = max_value
    return image


def apply_window_policy(image):

    image1 = apply_window(image, 40, 80) # brain
    image2 = apply_window(image, 80, 200) # subdural
    image3 = apply_window(image, 40, 380) # bone
    image1 = (image1 - 0) / 80
    image2 = (image2 - (-20)) / 200
    image3 = (image3 - (-150)) / 380
    image = np.array([
        image1 - image1.mean(),
        image2 - image2.mean(),
        image3 - image3.mean(),
    ]).transpose(1,2,0)

    return image
```

<u>Ninth Step:</u> Setting the directory to save the resized 128x128 PNG dimensional file.

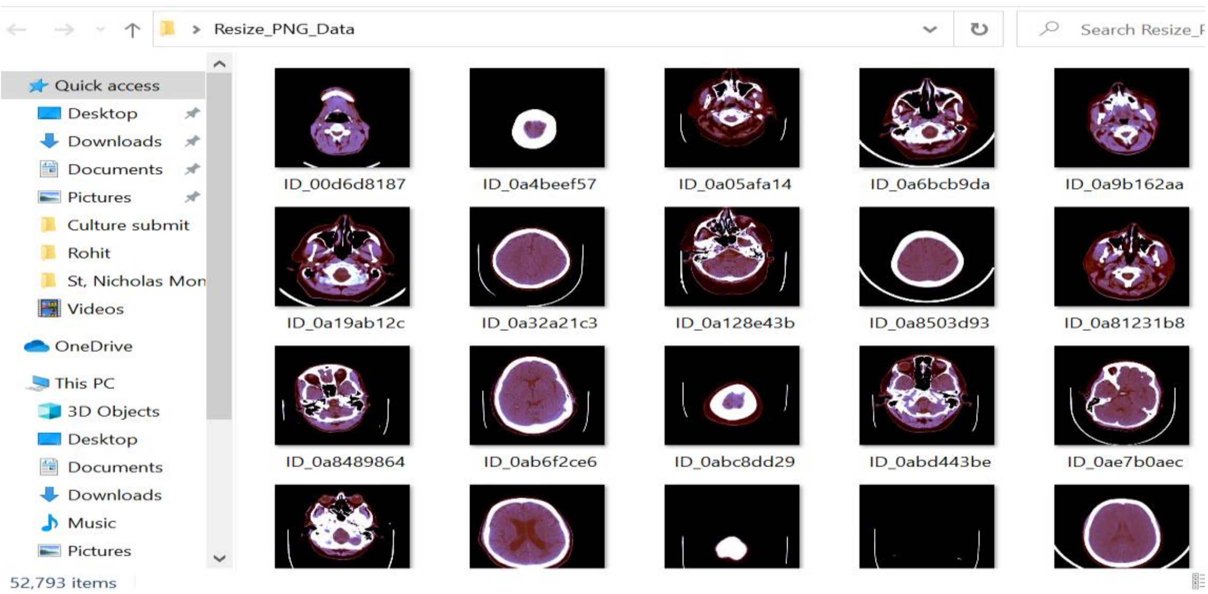
```python
def resize_save(filenames, load_dir):
    save_dir = 'C:/Users/yufen/Desktop/Resize_PNG_Data/'

    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    for filename in tqdm(filenames):
        try:
            path = load_dir + filename
            new_path = save_dir + filename.replace('.dcm', '.png')
            dcm = pydicom.dcmread(path)
            image = get_pixels_hu(dcm)
            image = apply_window_policy(image[0])
            image -= image.min((0,1))
            image = (255*image).astype(np.uint8)  # Normalize the Image
            image = cv2.resize(image, (128, 128)) # Resize image pixel
            res = cv2.imwrite(new_path, image)

        except ValueError:
            continue # black image,
```

In the folder, the selected images begin to download. As shown in the figure below,



5 Data Generator

Data generator function is used on the resized PNG images of Train, Test and Validation Data.

```
######## Image Data Generator is applied for the augmentation of the converted png Image,

def Datagen_creater():
    return ImageDataGenerator()


def training_gen(datagen):
    return datagen.flow_from_dataframe(
        training_dataf,
        directory="C:/Users/yufen/Desktop/Resize_PNG_Data/",
        x_col='filename',
        y_col=['any', 'epidural', 'intraparenchymal',
               'intraventricular', 'subarachnoid', 'subdural'],
        class_mode='raw',
        target_size=(128, 128),
        batch_size=32,
    )

def testing_gen():
    return ImageDataGenerator().flow_from_dataframe(
        sample_test,
        directory= "C:/Users/yufen/Desktop/Resize_PNG_Data/",
        x_col='filename',
        class_mode=None,
        target_size=(128, 128),
        batch_size=32,
        shuffle=False
    )

def validating_gen(datagen):
    return datagen.flow_from_dataframe(
        validate_dataf,
        directory="C:/Users/yufen/Desktop/Resize_PNG_Data/",
        x_col='filename',
        y_col=['any', 'epidural', 'intraparenchymal',
               'intraventricular', 'subarachnoid', 'subdural'],
        class_mode='raw',
        target_size=(128, 128),
        batch_size=32,
        shuffle=False,
    )

# Using original generator
data_generator = Datagen_creater()
training_gen = training_gen(data_generator)
validating_gen = validating_gen(data_generator)
testing_gen = testing_gen()
```

6 Executing CNN

Below function is used for the model checkpoint of the and learning rate reduction of the model

```python
### below is the function used to Reduce learning rate when a metric has stopped improving.
### early stop
from keras.callbacks import ReduceLROnPlateau

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,

checkpoint = ModelCheckpoint(
    'Full_model.h5',            #full model checkoint is set because during training the mode
    monitor='val_loss',
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode='auto')
```

Below is the code of CNN Model building

```python
#initializing CNN

cnn_model = models.Sequential()

#model architecture defining
cnn_model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape = (128, 128, 3)))
cnn_model.add(MaxPooling2D((2, 2)))

cnn_model.add(Conv2D(32, (3, 3), activation = 'relu'))
cnn_model.add(MaxPooling2D((2, 2)))

#fully connected layer
cnn_model.add(Flatten())
cnn_model.add(Dense(64, activation = 'relu'))
cnn_model.add(Dense(128, activation = 'relu'))

#one layer activated by sigmoid
cnn_model.add(Dense(6, activation = 'sigmoid'))
```

CNN model applied using the code Below:

```python
# setting the time to check the time taken by model
BATCH_SIZE = 32
import datetime
start = datetime.datetime.now()

### Setting steps per epochs for final model
total_steps = files.shape[0] // BATCH_SIZE
total_steps = total_steps // 4

#fitting the model
cnn_history = cnn_model.fit(training_gen,
                            steps_per_epoch = total_steps,
                            epochs = 10,
                            validation_data = validating_gen,
                            validation_steps=total_steps * 0.15,
                            callbacks = [learning_rate_reduction, checkpoint])
```

Evaluation of the model:

calculating the model prediction values

```python
validating_preds = cnn_model.predict(validating_gen, verbose = 1)
```

Confusion matrix is imported using kears application

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_true, y_prediction))
cm = confusion_matrix(y_true, y_prediction)
```

Confusion matrix function created :

```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize = (6,6))
    plt.imshow(cm, interpolation='nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    cm = np.round(cm,2)
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

Below code is to calculate for classification report accuracy

```
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score, accuracy_score
```

```
print('Model: CNN', '\n', classification_report(y_true, y_prediction, target_names = ['No Hemorrhage', 'Has Hemmorrhage
Model: CNN
```

7 Executing the Dense Net

```
######## Second Transfer Learning Model ------DenseNet121---------------------


from keras.applications.densenet import DenseNet121
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model
from keras import backend as K

base_model = DenseNet121(input_shape=(128, 128, 3), include_top=False, weights='imagenet', pooling='avg')

base_model.summary()
```

Calculating the dense Layer

```
layers = base_model.layers
print(f"The model has {len(layers)} layers")
```

```
#model = Sequential()
base_model = DenseNet121(include_top=False, weights='imagenet')
x = base_model.output

x = GlobalAveragePooling2D()(x)

predictions = Dense(6, activation="sigmoid")(x)

model = Model(inputs=base_model.input, outputs=predictions)

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

Evaluation steps for this model is same as we have used earlier for CNN Model

8 Executing the Xception Model

```
######## -------------------Xception MOdel ---------------------
from keras.applications import Xception
```

```
def create_model():
    base_model = Xception(weights = 'imagenet', include_top = False, input_shape = (128,128,3))
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.15)(x)
    y_pred = Dense(6, activation = 'sigmoid')(x)

    return Model(inputs = base_model.input, outputs = y_pred)
```

```
LR = 0.00005
model = create_model()
```

```
model.compile(optimizer = Adam(learning_rate = LR),
              loss = 'binary_crossentropy',
              metrics = [tf.keras.metrics.AUC()])
```

```
#### To Check the MOdel Layers

layers = model.layers
print(f"The model has {len(layers)} layers")
```

```
The model has 135 layers
```

```
#train_length = len(train_df)
total_steps = files.shape[0] // BATCH_SIZE
total_steps = total_steps // 4

history = model.fit(
    training_gen,
    steps_per_epoch = total_steps,
    validation_data=validating_gen,
    validation_steps=total_steps * 0.15,
    callbacks=[learning_rate_reduction, checkpoint],
    epochs=10
)
```

Evaluation steps for this model is same as we have used earlier for CNN Model, Because the model gave high accuracy, so I have evaluated the model using ROC Curve.

The below code is for ROC Curve

```
from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)

from sklearn.metrics import roc_curve
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_true, xception_y_preds)
```

```
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='ROC curve (area = {:.3f})'.format(auc_keras))

plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

The prediction value of the Xception model is then used in conjunction with the test picture file to identify the different subtypes of intracranial hemorrhage.

```
from PIL import Image
for i in range(20):

    for j in range(1,7):
        if test_frame.iloc[i,j] > 0.01:
            path = "C:/Users/yufen/Desktop/Resize_PNG_Data/" + str(test_frame.iloc[i,0])
            img = Image.open(path)
            plt.imshow(img)
            print(str(test_frame.iloc[i,0]) + " has a probability: "  + str(test_frame.iloc[i,j]) + " for a '" + str(te
            plt.show()
```