Configuration Manual

Intracranial Haemorrhage Detection using Machine Learning Models

MSc. Research Project

Bellana Tirupati Patro

Student ID: 18196551

School of Computing

National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	BELLANA TIRUPATI PATRO
Student ID:	X18196551
Programme:	MSc in Data Analytics
Year:	2019-2020
Module:	MSc Research Project
Lecturer:	Dr Rashmi Gupta
Submission Due Date:	17/08/2020
Project Title:	Intracranial Haemorrhage Detection using Deep Learning Models
Word Count:	1037
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: ………

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

Bellana Tirupati Patro
Student ID: 18196551

1 Hardware/Software Requirements

1.1 Hardware Requirements

Laptop	Lenovo Ideapad 330
RAM	8GB
Graphics Processing Unit(GPU)	Nvidia GTX 960 -2GB

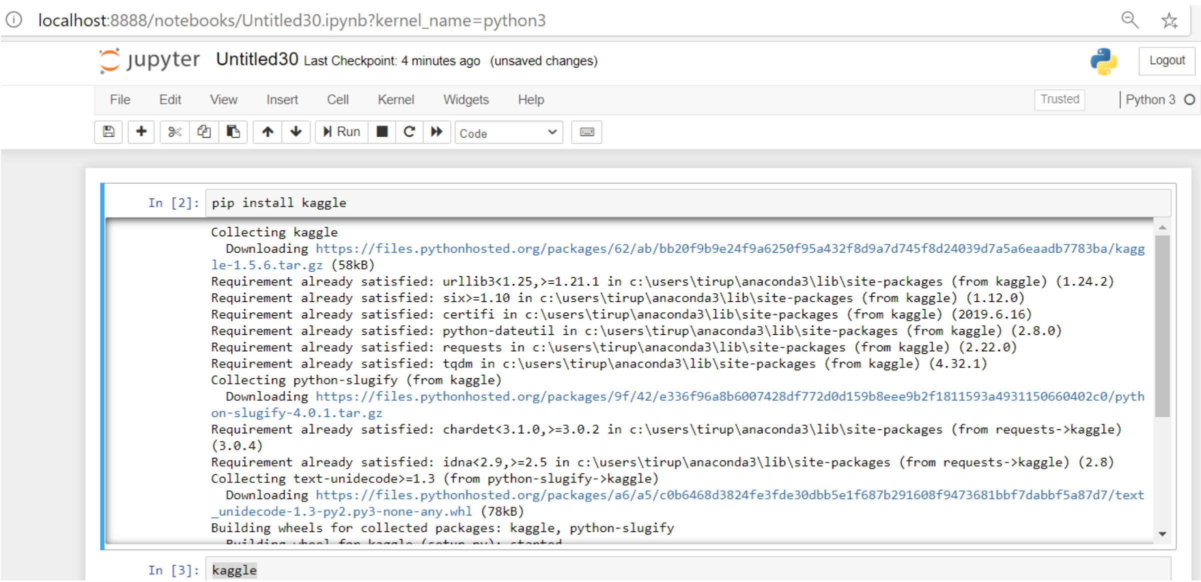
1.2 Software Requirements

Operating System	Windows 10
Programming Tools	Jupyter Notebook, Python Version 3, Anaconda
Graphics Processing Unit(GPU)	Nvidia GTX 960 -2GB

2 Data Gathering

Data Gathering is done for this research project using Kaggle API.

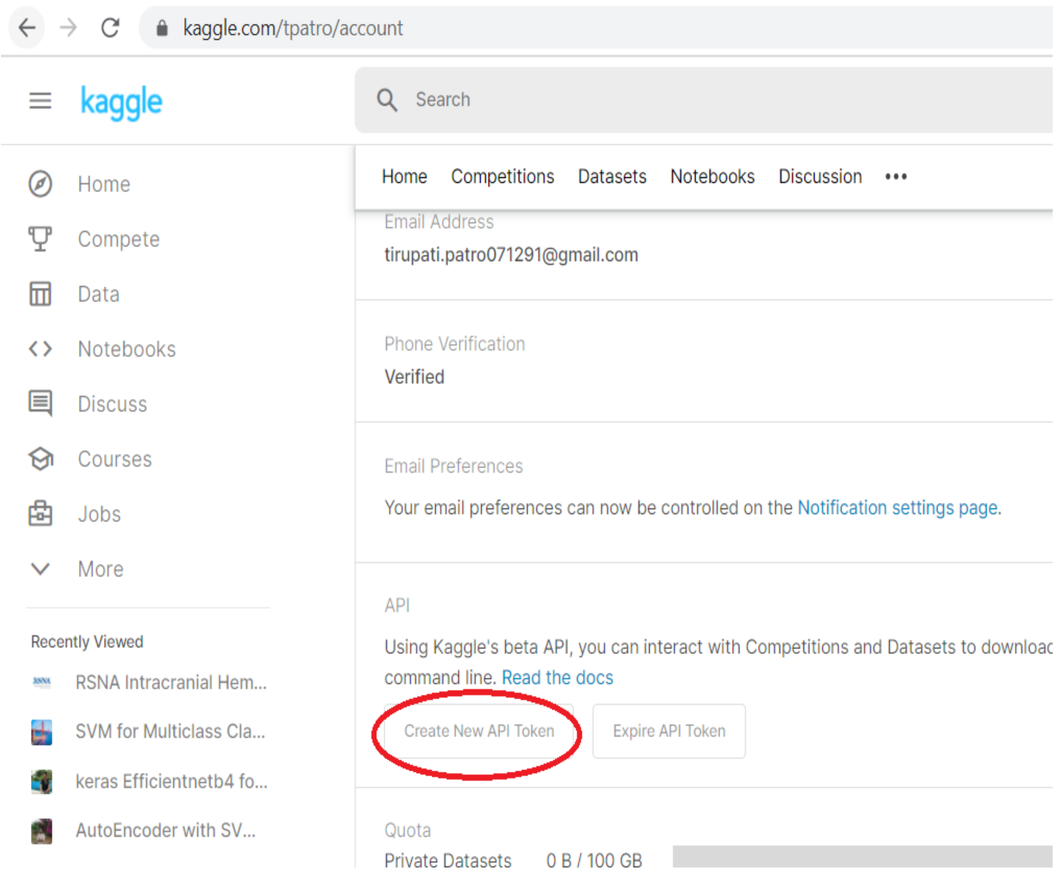
Step 1: Install the Kaggle in Jupyter Notebook.



Step 2: Login to Kaggle.com using your credentials.

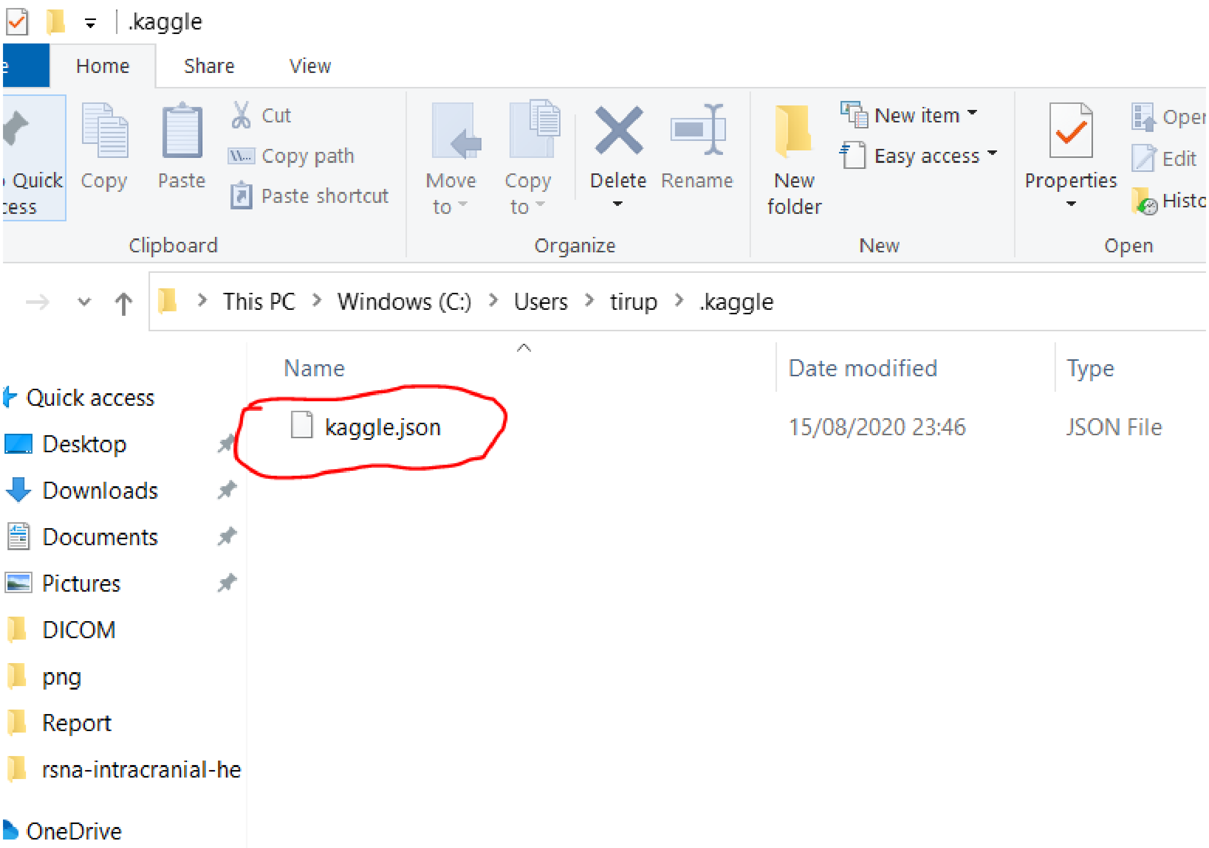
Step 3: Goto My Account page.

Step 4: Click on "Create New API Token"



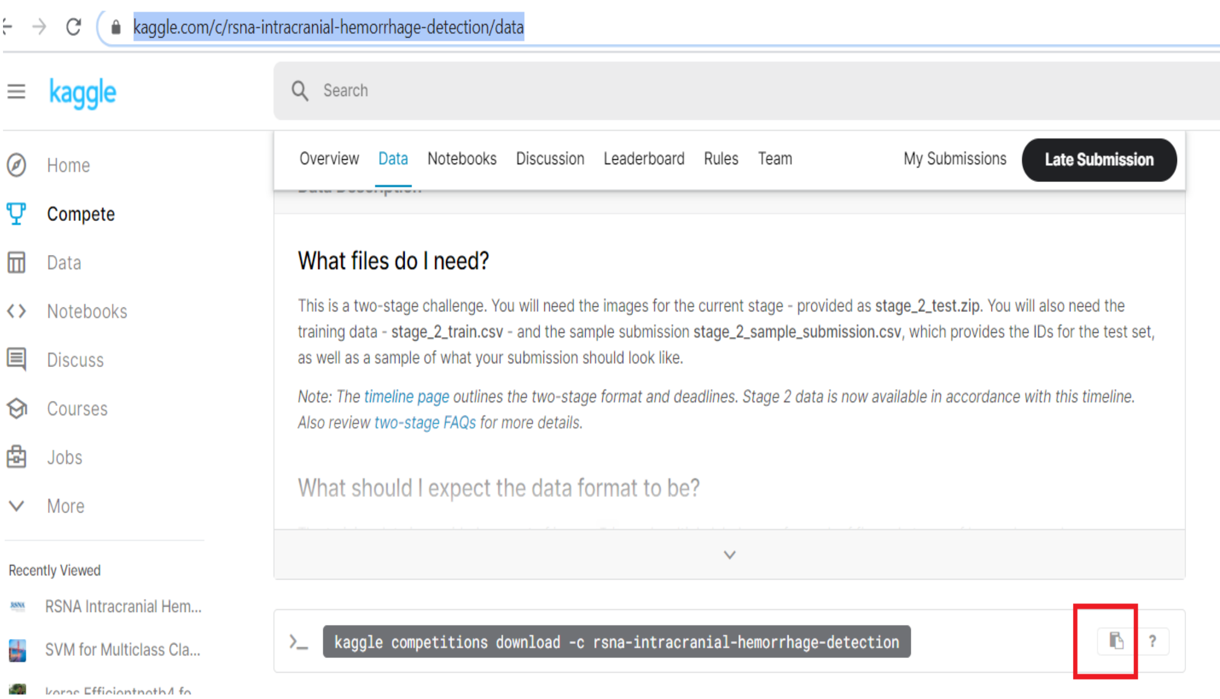
Step 5: The kaggle.json file is downloaded in Downloads folder of local machine.

Step 6: Copy this kaggle.json file from Dowloads folder to .kaggle folder present in Users folder inside C: drive of your local machine.

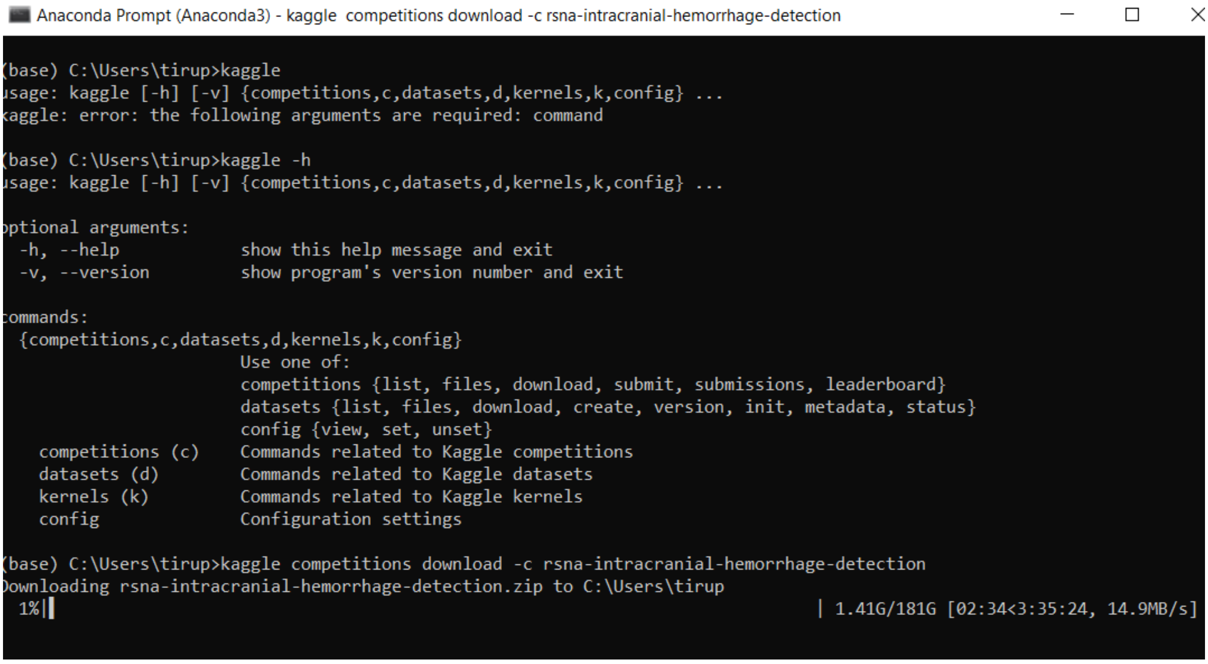


Step 7: Goto dataset page by clicking on this link https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection/data

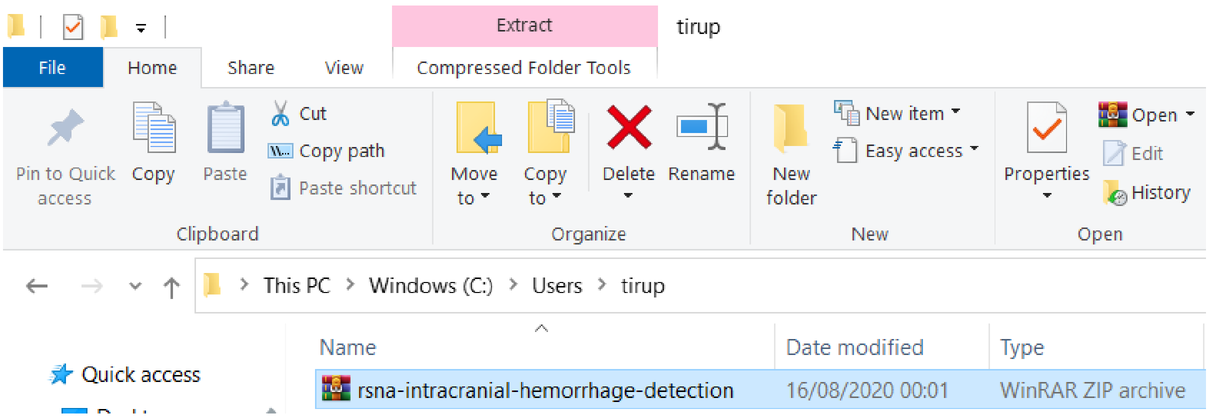
Step 8: Scroll down a bit and click on the link as per screenshot below.



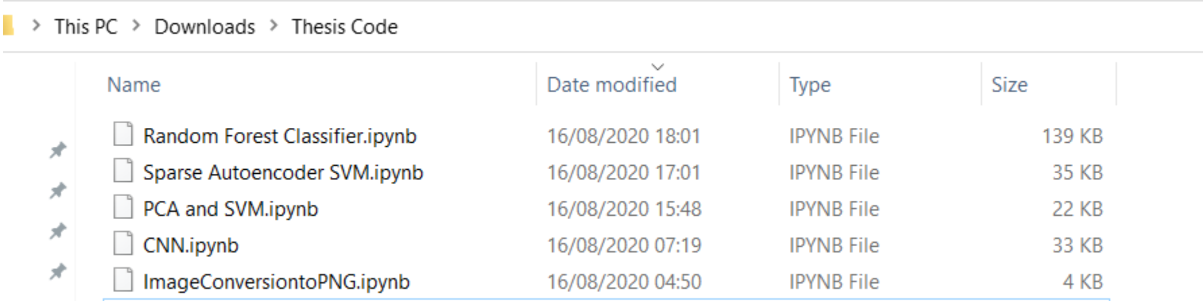
Step 9: Goto Anaconda Prompt and paste the command: **kaggle competitions download -c rsna-intracranial-hemorrhage-detection and click enter .**(refer the screenshot below).



Step 10: The dataset starts downloading in Zip folder inside Windows folder in C drive. (refer the screenshot below).



3 Code Structure

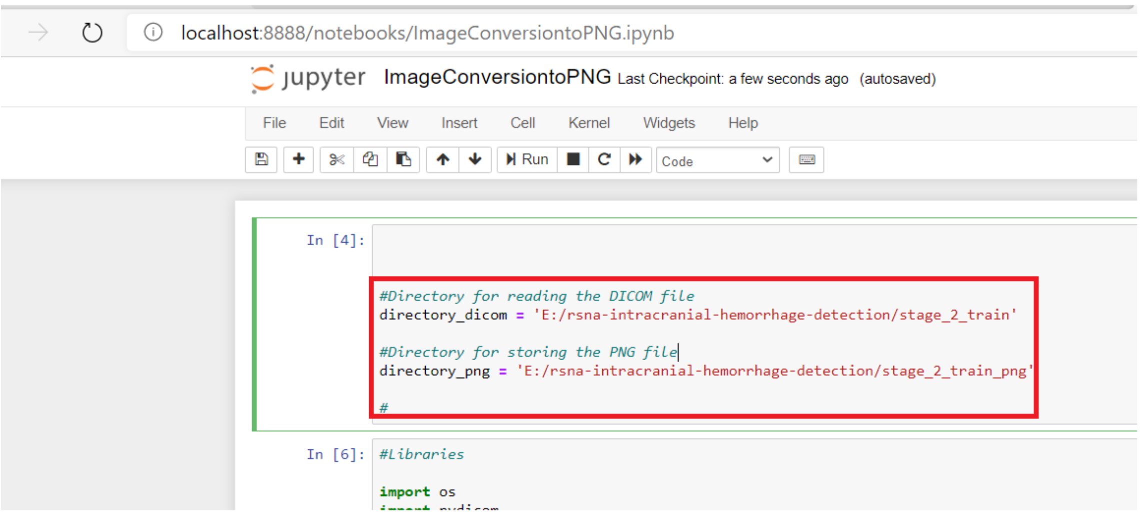


4 Data Conversion

Step 1: Unzip the data set downloaded to any specific folder you need.

Step 2: Upload the thesis code folder to jupyter notebook and open ImageConversiontoPNG.ipynb to jupyter notebook.

Step 3: Change the path for reading the DICOM file and change the path for storing the PNG file in ImageConversiontoPNG.ipynb.(Refer the screenshot below)



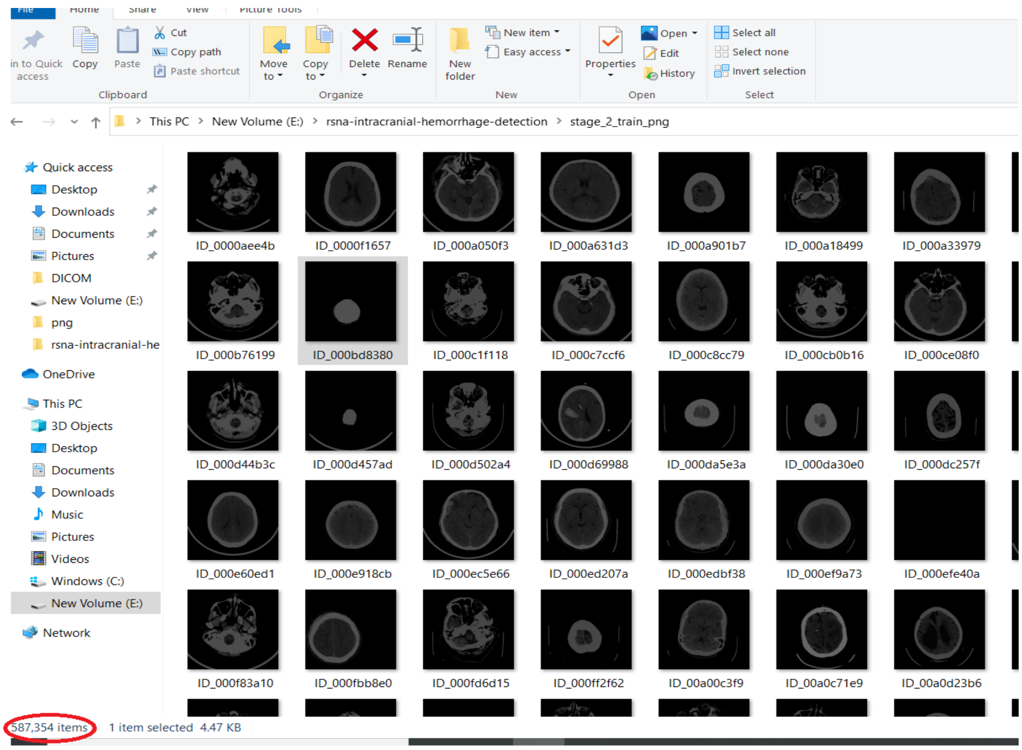
Step 4: After replacing the directory paths, import the necessary libraries.

```
In [8]: #Libraries

import os
import pydicom
import cv2
import glob
import multiprocessing as mp
import numpy as np
```

Step 5: Run the remaining part of the code.

```
In [ ]: # Functions

def window_image(img, window_center, window_width, intercept, slope):
    img = (img * slope + intercept)
    img_min = window_center - window_width // 2
    img_max = window_center + window_width // 2
    img[img < img_min] = img_min
    img[img > img_max] = img_max

    return img


def get_first_of_dicom_field_as_int(x):
    if type(x) == pydicom.multival.MultiValue:
        return int(x[0])
    else:
        return int(x)


def get_windowing(data):
    dicom_fields = [data[('0028', '1050')].value,  # window center
                    data[('0028', '1051')].value,  # window width
                    data[('0028', '1052')].value,  # intercept
                    data[('0028', '1053')].value]  # slope
    return [get_first_of_dicom_field_as_int(x) for x in dicom_fields]

def convert_to_png(dcm_in):

    dicomimage = pydicom.dcmread(dcm_in)
    window_center, window_width, intercept, slope = get_windowing(dicomimage)
    img = pydicom.read_file(dcm_in).pixel_array
    img = window_image(img, window_center, window_width, intercept, slope)
    cv2.imwrite(os.path.join(directory_png, os.path.basename(dcm_in)[:-3] + 'png'), img)

# Extract images in parallel
dicom = glob.glob(os.path.join(directory_dicom, '*.dcm'))
type(dicom)
print(dicom)
len(dicom)
import dask as dd
all_images = [dd.delayed(convert_to_png)(dicom[x]) for x in range(len(dicom))]
dd.compute(all_images)
```

Step 5: The image starts getting converted to PNG in the corresponding folder. (Refer Screenshot below)



5 Executing Sparse autoencoder and SVM Model

Step 1: Open the sparseautoencoder and SVM.ipynb from Thesis code folder.

Step 2: Set the path in the code from where data labels and PNG images are read.

```
In [2]:
#Set the path where the csv file is located
csv_path = ("E:/rsna-intracranial-hemorrhage-detection/stage_1_train.csv")

# Set the directory where converted PNG images are stored
imgpath = ("C:/Users/tirup/Downloads/trainimagespng/stage_1_train_png_224x/")
```

Step 3: Reading the train label file and creating a dataframe.

```
n [4]: import pandas as pd
import numpy as np
train_df = pd.read_csv(csv_path)
train_df['filename'] = train_df['ID'].apply(lambda st: "ID_" + st.split('_')[1] + ".png")
train_df['type'] = train_df['ID'].apply(lambda st: st.split('_')[2])
train_df.head()
```

Step 4: Creating a dataframe by checking the filename from labels dataframe and checking if the file exists for 50,000 images in the Image folder

```
In [39]: # Creating a dataframe by checking  the filename from labels dataframe and checking if the file exists in the Image folder

import os
Total_images=50000
sample_files = np.random.choice(os.listdir(imgpath), Total_images)
print(len(sample_files))
sample_df = train_df[train_df.filename.apply(lambda x: x.replace('.png', '.png')).isin(sample_files)]
sample_df = sample_df[['Label', 'filename', 'type']].drop_duplicates().pivot(
    index='filename', columns='type', values='Label').reset_index()
#sample_df = train_df[train_df.filename.apply(lambda x: x.replace('.png', '.png')).isin(sample_files)]
#sample_df = sample_df[['Label', 'filename', 'type']].drop_duplicates()
print(sample_df.shape)
sample_df.head()

sample_df = sample_df[sample_df["filename"].isin(os.listdir(imgpath))]
print(sample_df.shape)
sample_df.head()
```

Step 5: Splitting the label data frame created into test and train labels data frame.

```
In [93]: from sklearn.model_selection import train_test_split

train_label, test_label = train_test_split(sample_df, test_size=0.2)
print(train_label.shape)
print(test_label.shape)
arraylist1=list(train_label.index.values)
print(len(arraylist1))
arraylist2=list(test_label.index.values)
print(len(arraylist2))
```

Step 6: Import libraries for loading training image and convert to numpy array

```
: from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
import PIL
```

Step 6: After splitting the label dataframe, read filename and from numpy array for training images.

```
In [95]: from keras.preprocessing import image
all_images=[]
for i in arraylist1:
    file = train_label.loc[i]["filename"]
    if(os.path.isfile(imgpath+file)):
      img = image.load_img(imgpath+file, target_size=(128,128))
      img = image.img_to_array(img)
      all_images.append(img)
print(len(all_images))
print(len(train_label))
```

```
Trainimagearray=np.array(all_images)
```

Step 7: Loading image for testing data and converting to numpy array

```
5]: test_images=[]
    for j in arraylist2:
      file =test_label.loc[j]["filename"]
      if(os.path.isfile(imgpath+ file)):
          img = image.load_img(imgpath+file, target_size=(128,128))
          img = image.img_to_array(img)
          test_images.append(img)
    print(len(test_images))
    print(len(test_label))
```

```
In [99]: Testimagearray=np.array(test_images)
```

Step 8: One hot encoding for train and test labels

```
#converting encoded data to categories, one hot encoded data provided in the datas

def label_img(label_arr):
    labels=[]
    for i in range(len(label_arr)):
        if (label_arr[i][0]==0):
            labels.append("Normal")
        elif (label_arr[i][1]==1):
            labels.append("Epidural")
        elif (label_arr[i][2]==1):
            labels.append("Intraparenchymal")

        elif (label_arr[i][3]==1):
            labels.append("Intraventricular")

        elif (label_arr[i][4]==1):
            labels.append("Subarachnoid")
        else:
            labels.append("Subdural")
    return labels
```

```
train_label=np.array(train_label)
train_label= label_img(train_label)
```

```
test_label=np.array(test_label)
test_label= label_img(test_label)
test_label
```

Step 9: Normalize the pixel values of training and testing image numpy arrays

```
In [30]: # Normalize the pixels
Trainimagearray = Trainimagearray/255.
Testimagearray = Testimagearray/255.
```

Step 10: Building the encoder part of the auto encoder

```
|: # Modelling the autoencoder with keras library
from keras import regularizers
from keras.layers import Dense, Activation, Embedding, concatenate, Flatten, Input
from keras.models import Model

# the encoder part
encoder_input = Input(shape=(2352,))
encoded = Dense(2000, activation='relu')(encoder_input)
encoded = Dense(500, activation='relu',
                activity_regularizer=regularizers.l1(10e-10))(encoded)
encoded = Dense(500, activation='relu',
                activity_regularizer=regularizers.l1(10e-10))(encoded)
encoded = Dense(300, activation='sigmoid',
                activity_regularizer=regularizers.l1(10e-10))(encoded)
```

Step 11: Building the decoder part of autoencoder

```
In [33]:
# Decoder Part of the autoencoder
decoded = Dense(500, activation='relu')(encoded)
decoded = Dense(500, activation='relu')(decoded)
decoded = Dense(2000, activation='relu')(decoded)
decoded = Dense(2352,)(decoded)

# this model maps an input to its reconstruction
autoencoder = Model(encoder_input, decoded)
```

Step 12. Fitting the autoencoder

```
]: # Fitting the model
autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.fit(X_train, X_train, epochs=50, batch_size=28, validation_data=(X_test, X_test))
```

Step 13: Retrieving the encoded image from autoencoder.

```
39]: # retrieving the encoded training image from auto encoder
Encodeimagetrain=encoder.predict(X_train)
```

```
2]: # retrieving encoded image for testing images
Encodeimagetest=encoder.predict(X_test)
```

Step 14: Training the SVM classifier and predicting for training set of images.

```
4]: # Training the SVM classifier with encoder training images
from sklearn.svm import SVC
svm = SVC(kernel='rbf', probability=True, random_state=0, gamma=.01, C=1)

# fit model
SVMmodel=svm.fit(Encodeimagetrain, train_label)
```

```
5]: # Testing the SVM classifier with encoded image of test image array
SVMPredict=SVMmodel.predict(Encodeimagetest)
```

Step 15. Accuracy of the Model

```
]: # Accuracy of the model
from sklearn.metrics import accuracy_score
# calculate accuracy
accuracy = accuracy_score(test_label, SVMPredict)
print('Model accuracy is: ', accuracy)
```

Step 16: Generating the confusion matrix.

```
: from sklearn.metrics import confusion_matrix,classification_report
matrix=confusion_matrix(test_label,SVMPredict, labels=['Any','Normal','Epidural','Subdural','Intraparenchyma
print(matrix)
```

Step 17. Classification report for precision, recall and f1 score

```
[48]: report = classification_report(test_label,SVMPredict,labels=['Any','Normal','Epidural','Subdural','Intraparenchymal','Intraventr
print('Classification report : \n',report)
```

6 Executing the PCA and SVM Model

Step 1: Open the PCA and SVM.ipynb from Thesis code folder.

Step 2: Set the path in the code from where data labels and PNG images are read.

```
In [2]:
#Set the path where the csv file is located
csv_path = ("E:/rsna-intracranial-hemorrhage-detection/stage_1_train.csv")

# Set the directory where converted PNG images are stored
imgpath = ("C:/Users/tirup/Downloads/trainimagespng/stage_1_train_png_224x/")
```

Step 3: Reading the train label file and creating a dataframe.

```
n [4]: import pandas as pd
import numpy as np
train_df = pd.read_csv(csv_path)
train_df['filename'] = train_df['ID'].apply(lambda st: "ID_" + st.split('_')[1] + ".png")
train_df['type'] = train_df['ID'].apply(lambda st: st.split('_')[2])
train_df.head()
```

Step 4: Creating a dataframe by checking the filename from labels dataframe and checking if the file exists for 50,000 images in the Image folder

```
In [39]: # Creating a dataframe by checking  the filename from labels dataframe and checking if the file exists in the Image folder

import os
Total_images=50000
sample_files = np.random.choice(os.listdir(imgpath), Total_images)
print(len(sample_files))
sample_df = train_df[train_df.filename.apply(lambda x: x.replace('.png', '.png')).isin(sample_files)]
sample_df = sample_df[['Label', 'filename', 'type']].drop_duplicates().pivot(
    index='filename', columns='type', values='Label').reset_index()
#sample_df = train_df[train_df.filename.apply(lambda x: x.replace('.png', '.png')).isin(sample_files)]
#sample_df = sample_df[['Label', 'filename', 'type']].drop_duplicates()
print(sample_df.shape)
sample_df.head()

sample_df = sample_df[sample_df["filename"].isin(os.listdir(imgpath))]
print(sample_df.shape)
sample_df.head()
```

Step 5: Splitting the label data frame created into test and train labels data frame.

```
In [93]: from sklearn.model_selection import train_test_split

train_label, test_label = train_test_split(sample_df, test_size=0.2)
print(train_label.shape)
print(test_label.shape)
arraylist1=list(train_label.index.values)
print(len(arraylist1))
arraylist2=list(test_label.index.values)
print(len(arraylist2))
```

Step 6: Import libraries for loading training image and convert to numpy array

```
: from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
import PIL
```

Step 6: After splitting the label dataframe, read filename and from numpy array for training images.

```
In [95]: from keras.preprocessing import image
all_images=[]
for i in arraylist1:
    file = train_label.loc[i]["filename"]
    if(os.path.isfile(imgpath+file)):
      img = image.load_img(imgpath+file, target_size=(128,128))
      img = image.img_to_array(img)
      all_images.append(img)
print(len(all_images))
print(len(train_label))
```

```
Trainimagearray=np.array(all_images)
```

Step 7: Loading image for testing data and converting to numpy array

```
5]: test_images=[]
for j in arraylist2:
  file =test_label.loc[j]["filename"]
  if(os.path.isfile(imgpath+ file)):
      img = image.load_img(imgpath+file, target_size=(128,128))
      img = image.img_to_array(img)
      test_images.append(img)
print(len(test_images))
print(len(test_label))
```

```
In [99]: Testimagearray=np.array(test_images)
```

Step 8: One hot encoding for train and test labels

```
#converting encoded data to categories, one hot encoded data provided in the datas

def label_img(label_arr):
    labels=[]
    for i in range(len(label_arr)):
        if (label_arr[i][0]==0):
            labels.append("Normal")
        elif (label_arr[i][1]==1):
            labels.append("Epidural")
        elif (label_arr[i][2]==1):
            labels.append("Intraparenchymal")

        elif (label_arr[i][3]==1):
            labels.append("Intraventricular")

        elif (label_arr[i][4]==1):
            labels.append("Subarachnoid")
        else:
            labels.append("Subdural")
    return labels
```

```
train_label=np.array(train_label)
train_label= label_img(train_label)
```

Step 9: Reshaping to one dimension vector for 128*128*3= 49152

```
Trainimagearray=Trainimagearray.reshape(len(Trainimagearray),49152)
Testimagearray=Testimagearray.reshape(len(Testimagearray),49152)
Trainimagearray.shape
```

Step 10:

Applying the model:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=300)
pca.fit(Trainimagearray)
```

```
data_pca = pca.transform(Trainimagearray)
test_pca=pca.transform(Testimagearray)
```

```
test_label=np.array(test_label)
train_label=np.array(train_label)
```

```
train_label
from sklearn.svm import SVC
svm = SVC(kernel='linear', probability=True, random_state=42)

# fit model
SVMmodel=svm.fit(data_pca, train_label)
```

```
SVMPredict=SVMmodel.predict(test_pca)
```

Step 11: Accuracy,confusion matrix and classification report:

```
SVMPredict=SVMmodel.predict(test_pca)
```

```
from sklearn.metrics import accuracy_score
# calculate accuracy
accuracy = accuracy_score(test_label, SVMPredict)
print('Model accuracy is: ', accuracy)
```

```
from sklearn.metrics import confusion_matrix,classification_report
matrix=confusion_matrix(test_label,SVMPredict, labels=['Normal','Epidural','Subdural','Intraparenchymal','Intraventricular','Sub
print(matrix)
```

```
report = classification_report(test_label,SVMPredict,labels=['Normal','Epidural','Subdural','Intraparenchymal','Intraventricular
print('Classification report : \n',report)
```

7 Executing CNN Model

Step 1: Repeat step 1 to step 7 from previous section 5.

Step 2: Perform one hot encoding for labels.

```
In [11]: #converting encoded data to categories, one hot encoded data provided in the dataset
         import pdb
         def label_img(label_arr):
             labels=[]
             for i in range(len(label_arr)):
                 if (label_arr[i][0]==0):
                     labels.append(0)
                 elif (label_arr[i][1]==1):
                     labels.append(1)
                 elif (label_arr[i][2]==1):
                     labels.append(2)

                 elif (label_arr[i][3]==1):
                     labels.append(3)

                 elif (label_arr[i][4]==1):
                     labels.append(4)
                 else:
                     labels.append(5)
             return labels
```

```
In [12]: import sys
         test_label= label_img(np.array(test_label))
         train_label= label_img(np.array(train_label))
         train_label=np.array(train_label)
         test_label=np.array(test_label)
```

Step 3: Normalize the pixels in an image.

```
]: # Normalize the pixels in an image
   Trainimagearray = Trainimagearray/255.
   Testimagearray = Testimagearray/255.
```

Step 4: import libraries for CNN model

```
# import libraries for CNN model
from keras.layers import MaxPool2D
import keras
from keras.layers import MaxPooling2D
from keras.models import Sequential
from tensorflow import keras
from keras.layers import Conv2D
```

Step 5: Building the CNN Model.

```
#Building the model
model = Sequential([
    Conv2D(16, (3, 3), activation='sigmoid', input_shape=(224,224, 3)),
    MaxPooling2D(2, 2),

    Conv2D(32, (3, 3), activation='sigmoid'),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='sigmoid'),
    Conv2D(64, (3, 3), activation='sigmoid'),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='sigmoid'),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.50),

    Flatten(),
    Dense(512, activation='relu'),
    Dense(6, activation='softmax')
])

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics =['accuracy'])
```

Step 6: Fitting the model.

```
# fitting the model
Model=model.fit(Trainimagearray, train_label, epochs=50,verbose=1, batch_size=28, validation_data=(Testimagearray, test_label))
```

8 Executing the Random Forest Classifier.

Step 1: Follow the steps from 1 to 9 used in Section 5.
Step 2: Generate the random forest classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=10)
rf.fit(Trainimagearray,train_label)
# Predictions on training and validation
y_pred_train = rf.predict(Testimagearray)

# training metrics
from sklearn.metrics import accuracy_score
# calculate accuracy
accuracy = accuracy_score(test_label, y_pred_train)
print('Model accuracy is: ', accuracy)
```

Step 4. Generate the confusion matrix using sk learn library

```
from sklearn.metrics import confusion_matrix,classification_report
matrix=confusion_matrix(test_label,y_pred, labels=['Normal','Epidural','Subdural','Intraparenchymal','Intraventricular','Subarac
print(matrix)
```

Step 4. Create the classification report for evaluation metrics like precision, recall, f1score.

```
from sklearn.metrics import classification_report
report = classification_report(test_label,y_pred_train,labels=['Any','Normal','Epidural','Subdural','Intraparenchymal','Intraven
print('Classification report : \n',report)
```