Configuration Manual

MSc Research Project
MSCDAD JAN22 A

Babita Chaini
Student ID: x21139211

School of Computing
National College of Ireland

Supervisor: Mohammed Hasanuzzaman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Babita Chaini
Student ID:	x21139211
Programme:	MSCDAD JAN22 A
Year:	2022
Module:	MSc Research Project
Supervisor:	Mohammed Hasanuzzaman
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	453
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:	1st February 2023

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

Babita Chaini
x21139211

1 Introduction

A Hybrid Machine Learning Model For Brain Tumor Classification is described in depth in this configuration manual, along with the system configuration, software requirements, and operational procedures. Section 2 of this document provides details on hardware and software specs. The setting up of the environment, gathering and preparation of the data, importation of libraries, and mounting of Google Drive are covered in Section 3. The many procedures involved in image processing are described in Section 4. The models' design and implementation are discussed in Section 5.

2 System Configuration

The system configuration that was used to carry out the project is described in this part of the configuration manual.

2.1 Hardware Requirements

Table 1: Hardware Configuration

Operating System	Windows 10
RAM	27.3 GB (Google Colab Pro)
Disk Space	210 GB (Google Colab Pro)
Runtime Model Name	Intel(R) Xeon(R) CPU @ 2.20GHz

2.2 Software Requirements

Table 2: Software Configuration

Programming Language	Python 3.8.16
IDE	Google Colab Pro
Database Management	Google Drive
Web Browser	Google Chrome
Email Account	Gmail account for Google Drive and Colab
Other Softwares	Microsoft Office and Overleaf

3 Project Development

Information about environment setup, data collection, and data consumption is provided in this part of the configuration manual.

3.1 Google Colaboratory Environment Setup

The research implementation environment is Google Colaboratory [1]. It offers Google Cloud hosting, Python version 3.6.9, and strong RAM and GPU support. To utilize Google Colab, you must have a Gmail account because of login requirements.

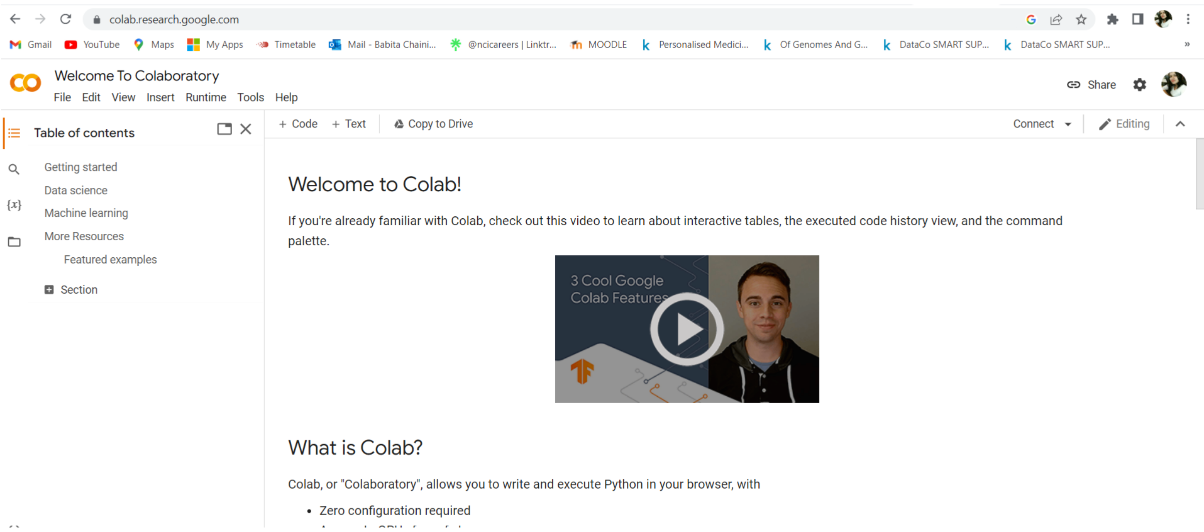

Figure 1: Google Colab

3.2 Data Acquisition

Both the data is collected from Kaggle [2] Dataset 1 [3] and Dataset 2 [4] .It is an ethical data source that is openly accessible.Both the dataset are brain tumor MRI dataset.

[1] Google Colab:https://colab.research.google.com/
[2] Kaggle : `https://www.kaggle.com/`
[3] Dataset 1: `https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection`
[4] Dataset 2:https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

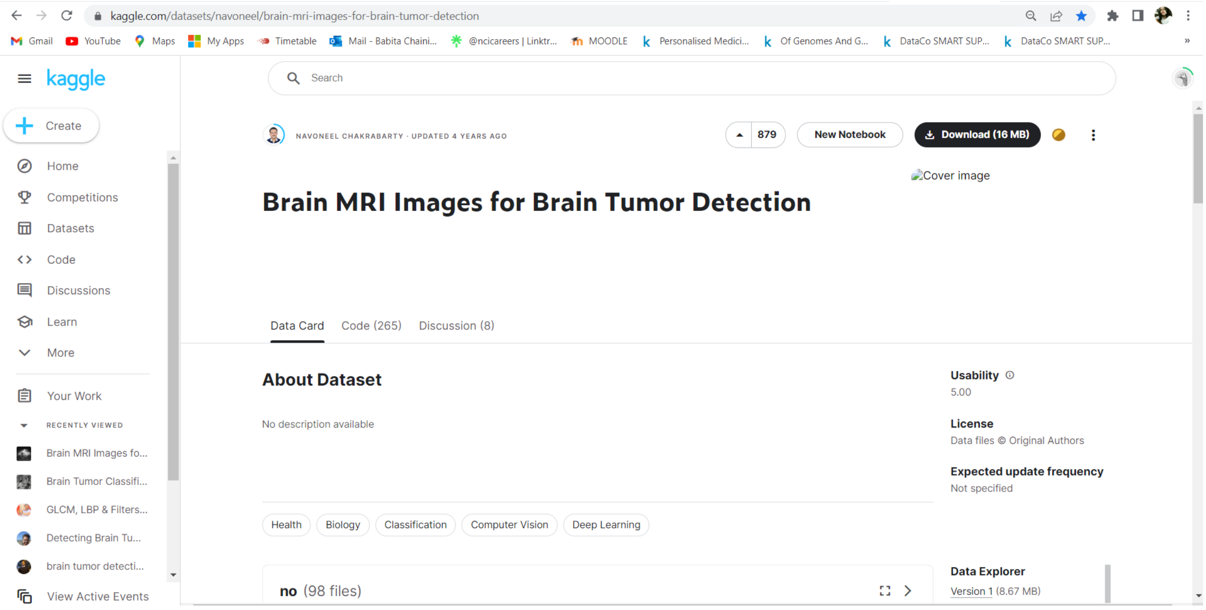

Figure 2: Dataset 1 Source

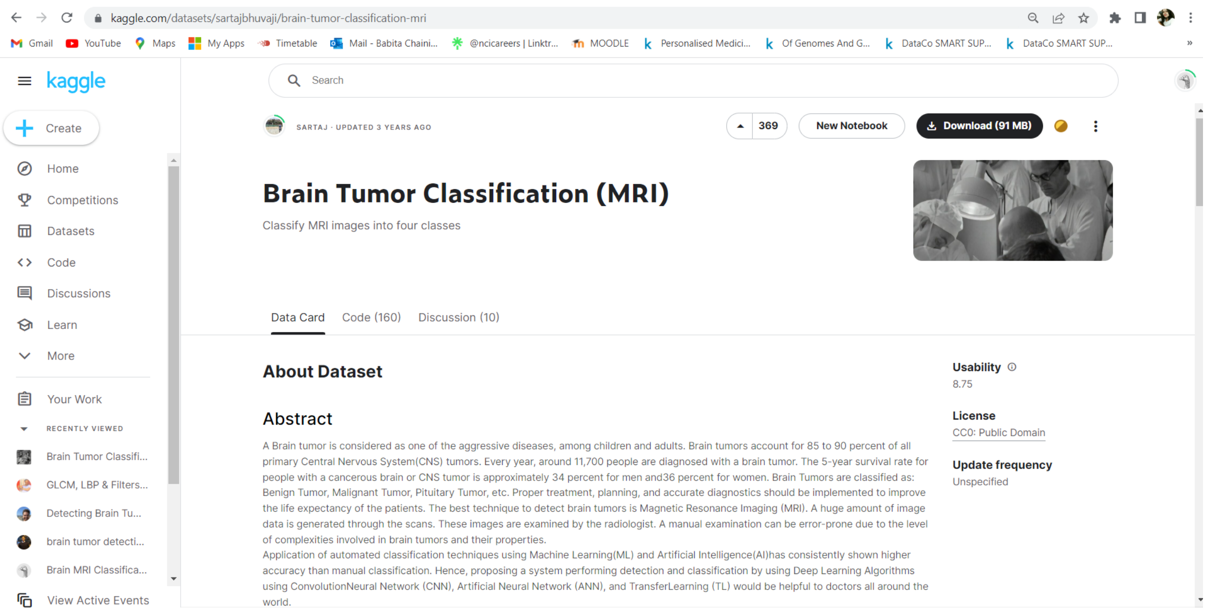

Figure 3: Dataset 2 Source

3.3 Data Preparation

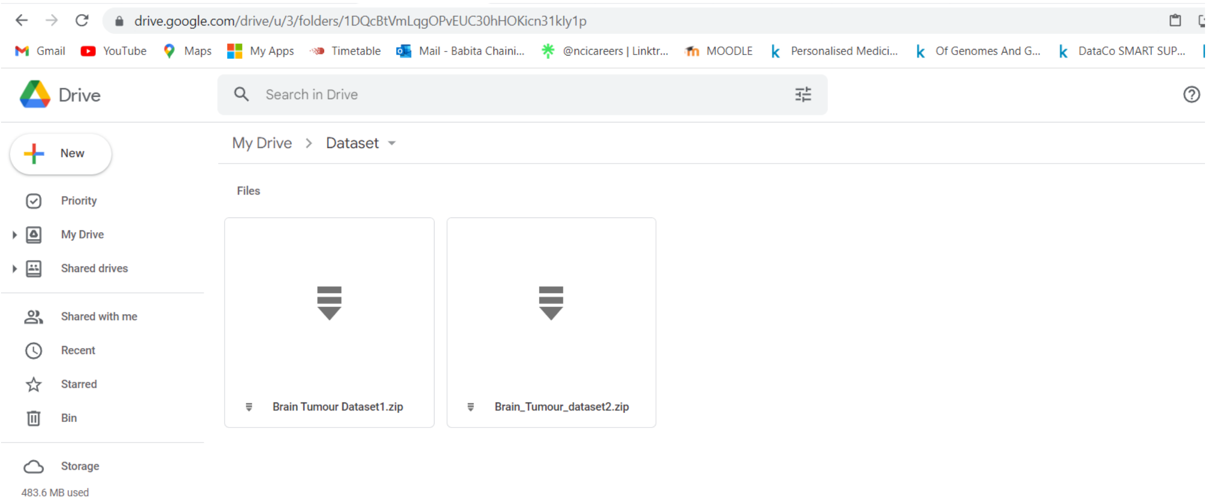

Figure 4: Google Drive Data

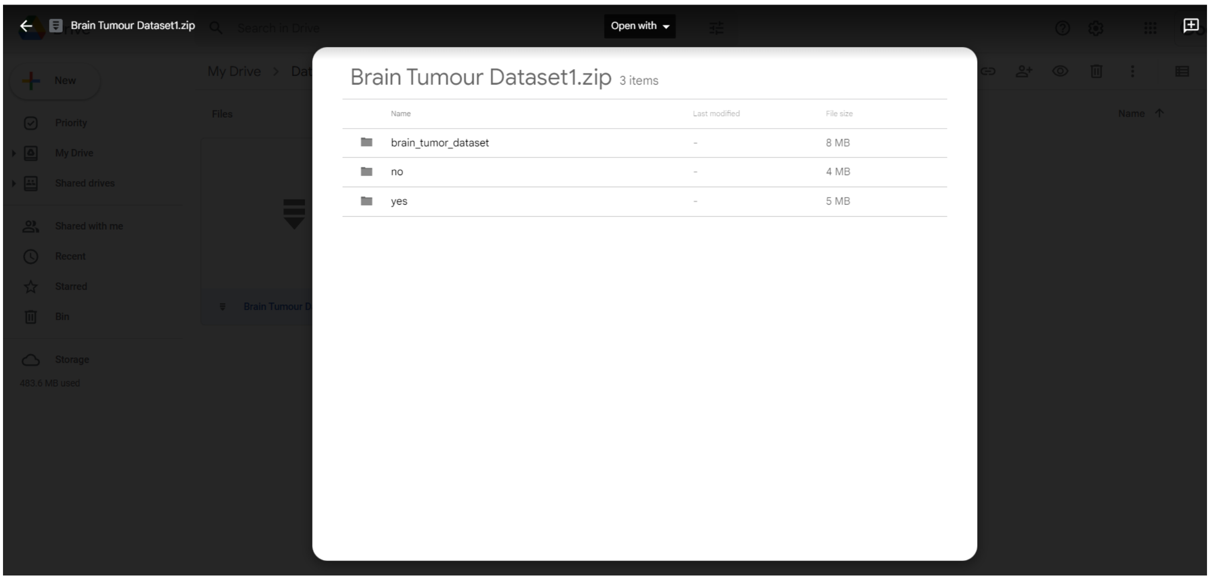

Figure 5: Dataset 1 Folders

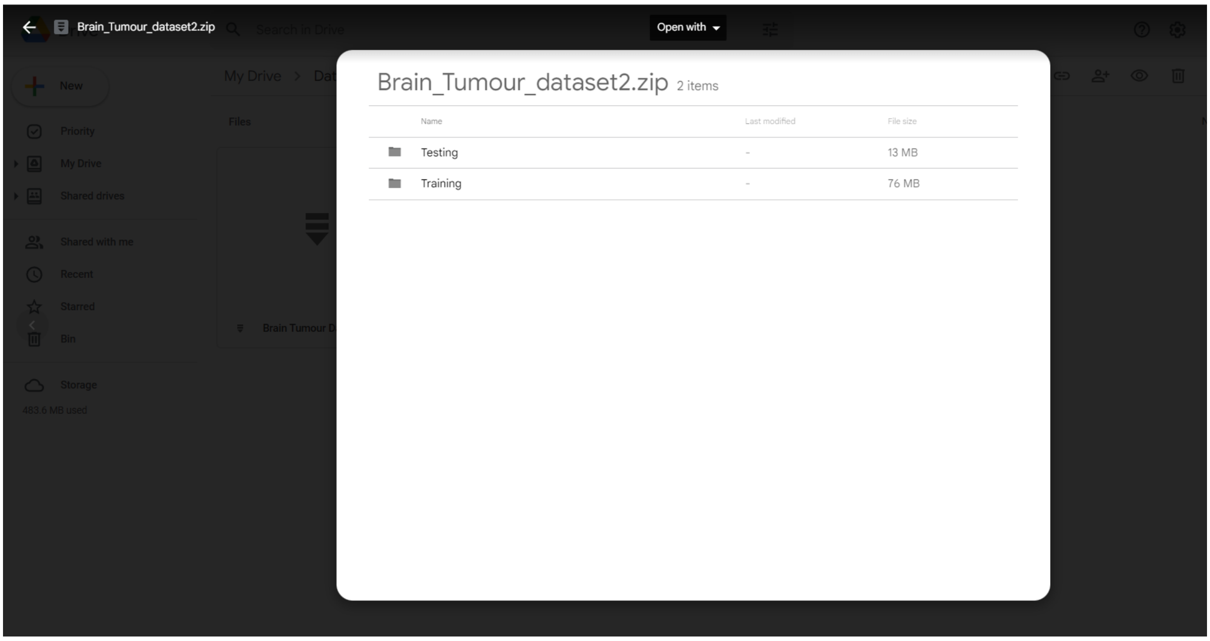

Figure 6: Dataset 2 Folders

3.4 Importing libraries

The libraries listed in this section are necessary for carrying out the research project. The libraries are imported using pip. sklearn [5], Tensorflow [6] and Keras [7] are the primarily used libraries in the research.

[5]sklearn: https://scikit-learn.org/
[6]Tensorflow: https://www.tensorflow.org/
[7]Keras: https://keras.io/

```
# Import All libraries
import os
import zipfile
import seaborn as sns
import cv2
from tqdm import tqdm
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow import keras
from keras import optimizers
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from tqdm import tqdm
from sklearn import svm
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import keras
```

Figure 7: Libraries Imported

3.5 Mounting Google Drive

To utilize the data, Google Colab must have Google Drive mounted. It has to be authenticated using the Colab Gmail account.

```
[ ] # Mounting Google drive
    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).
```

Figure 8: Mounting Google Drive on Colab Notebook

4 Image Processing

Information on the various phases of image processing is provided in this section.

4.1 Image Pre-processing

The study employed a variety of image processing approaches, as illustrated below.

```
#method for cropping image i.e. for getting brain area
def crop_brain_contour(image, plot=False):
    # Convert the image to grayscale, and blur it slightly
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)

    # Threshold the image, then perform a series of erosions +
    # dilations to remove any small regions of noise
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    # Find contours in thresholded image, then grab the largest one
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)


    # Find the extreme points
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])

    # crop new image out of the original image using the four extreme points (left, right, top, bottom)
    new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]

    if plot:
        plt.figure()

        plt.subplot(1, 2, 1)
        plt.imshow(image)
```

Figure 9: Image Pre-processing

4.2 Image Augmentation

The dataset is augmented with additional images.There are many methods employed, including crop, blur the image, find contours and extreme edges in the image, etc

Gaussian Filter

Basically Blurs the image

```
[ ] fig = plt.figure(figsize = (12,12))
    plt.gray()  # show the filtered result in grayscale
    ax1 = fig.add_subplot(121) # On the Left
    ax2 = fig.add_subplot(122) # Right Side
    ax1.set_title("Gaussian Filter")
    ax2.set_title("Without Filter")
    gaussian_img = nd.gaussian_filter(og,sigma = 1)
    ax1.imshow(gaussian_img)
    ax2.imshow(og)
    plt.show()
```

```
[ ] real,gabor_img = gabor(og,frequency = 0.9)

    fig = plt.figure(figsize = (12,12))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)
    ax1.set_title("Gabor Filter")
    ax2.set_title("Without Filter")
    ax1.imshow(gabor_img)
    ax2.imshow(og)
```

Sobel Filter

Helps to enhance the edges

```
[ ] sobel_img = sobel(og)

    fig = plt.figure(figsize = (12,12))
     # show the filtered result in grayscale
    ax1 = fig.add_subplot(121)# left side
    ax2 = fig.add_subplot(122)
    ax1.set_title("Sobel Filter")
    ax2.set_title("Without Filter")
    ax1.imshow(sobel_img)
    ax2.imshow(og)
    plt.show()
```

```
[ ] hessian_img = hessian(og,sigmas = range(1,100,1))
    fig = plt.figure(figsize = (12,12))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)
    ax1.set_title("hessian Filter")
    ax2.set_title("Without Filter")
    ax1.imshow(hessian_img)
    ax2.imshow(og)
```

Figure 10: Image Transformation

5 Modelling

5.1 Squeeze-Net

```
sq1x1 = "squeeze1x1"
exp1x1 = "expand1x1"
exp3x3 = "expand3x3"
relu = "relu_"

WEIGHTS_PATH = "https://github.com/rcmalli/keras-squeezenet/releases/download/v1.0/squeezenet_weights_tf_dim_ordering_tf_kernels.h5"
WEIGHTS_PATH_NO_TOP = "https://github.com/rcmalli/keras-squeezenet/releases/download/v1.0/squeezenet_weights_tf_dim_ordering_tf_kernels_notop.h5

# Modular function for Fire Node

def fire_module(x, fire_id, squeeze=16, expand=64):
    s_id = 'fire' + str(fire_id) + '/'

    if K.image_data_format() == 'channels_first':
        channel_axis = 1
    else:
        channel_axis = 3

    x = Convolution2D(squeeze, (1, 1), padding='valid', name=s_id + sq1x1)(x)
    x = Activation('relu', name=s_id + relu + sq1x1)(x)

    left = Convolution2D(expand, (1, 1), padding='valid', name=s_id + exp1x1)(x)
    left = Activation('relu', name=s_id + relu + exp1x1)(left)

    right = Convolution2D(expand, (3, 3), padding='same', name=s_id + exp3x3)(x)
    right = Activation('relu', name=s_id + relu + exp3x3)(right)

    x = concatenate([left, right], axis=channel_axis, name=s_id + 'concat')
    return x
```

```
# Original SqueezeNet from paper.

def SqueezeNet(include_top=True, weights='imagenet',
               input_tensor=None, input_shape=None,
               pooling=None,
               classes=1000):
    """Instantiates the SqueezeNet architecture.
    """

    if weights not in {'imagenet', None}:
        raise ValueError('The `weights` argument should be either '
                         '`None` (random initialization) or `imagenet` '
                         '(pre-training on ImageNet).')

    if weights == 'imagenet' and classes != 1000:
        raise ValueError('If using `weights` as imagenet with `include_top`'
                         ' as true, `classes` should be 1000')


    input_shape = _obtain_input_shape(input_shape,
                                      default_size=227,
                                      min_size=48,
                                      data_format=K.image_data_format(),
                                      require_flatten=include_top)

    if input_tensor is None:
        img_input = Input(shape=input_shape)
    else:
        if not K.is_keras_tensor(input_tensor):
```

Figure 11: Squeeze-Net Model Implementation

```
#loading SqueezeNet pretrained model

SIZE=256
# SqueezeNet_model=SqueezeNet(input_shape=(SIZE,SIZE,3),nb_classes = y_train.shape[1],
#                     use_bypass = False,dropout_rate = None,compression=1.0)

# def SqueezeNet(include_top=True, weights='imagenet',
#                input_tensor=None, input_shape=None,
#                pooling=None,
#                classes=1000):
SqueezeNet_model=SqueezeNet(input_shape=(SIZE,SIZE,3),include_top=False,weights='imagenet')
```

```
[ ] #we are not not using SqueezeNet model for training...so we made all layers as non trainable
    for layer in SqueezeNet_model.layers:
        layer.trainable=False
```

```
[ ] SqueezeNet_model.summary()

Model: "squeezenet"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv1 (Conv2D)                 (None, 127, 127, 64  1792        ['input_2[0][0]']
                                )

 relu_conv1 (Activation)        (None, 127, 127, 64  0           ['conv1[0][0]']
                                )

 pool1 (MaxPooling2D)           (None, 63, 63, 64)   0           ['relu_conv1[0][0]']
```

Figure 12: Squeeze-Net Model Implementation Cont.

5.2 Feature Extraction using Squeeze-Net

Features are extracted using pre-trained Squeeze-Net model and the features are passed to SVM classifier for classification.

```
[ ] #saving vgg model into file for further use
    SqueezeNet_model.save("/content/drive/My Drive/SqueezeNet_model.h5")

    WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compil

    #extract features for training data
    feature_ex=SqueezeNet_model.predict(X_train)

    82/82 [==============================] - 41s 495ms/step

[ ] feature_ex.shape

    (2611, 15, 15, 512)

[ ] features=feature_ex.reshape(feature_ex.shape[0],-1)

[ ] features.shape

    (2611, 115200)

[ ] #extract features for testing data
    test_feature_ex=SqueezeNet_model.predict(X_test)
    test_features=test_feature_ex.reshape(test_feature_ex.shape[0],-1)

    21/21 [==============================] - 8s 378ms/step
```

Figure 13: Feature Extraction Using Squeeze-Net Model

5.3 Support Vector Machine (SVM)

Below is an illustration of how the SVM classifier is implemented using Grid Search Cross Validation.

```
parameters = {'kernel':('linear',"sigmoid"),
              'C': [1000,100,300]}
svm = SVC()
svm_cv = GridSearchCV(svm,parameters,cv=10)
tqdm(svm_cv.fit(features, y_train))
#Predict the response for test features
y_pred = svm_cv.predict(test_features)
```

```
[ ] svm_cv.score(test_features,y_test)
```

```
0.8947368421052632
```

```
[ ] #testing accuracy
    metrics.accuracy_score(y_test,y_pred)
```

```
0.8947368421052632
```

```
[ ] # accuracy: (tp + tn) / (p + n)
    accuracy_svm = accuracy_score(y_test, y_pred)
    print('Accuracy: %f' % accuracy_svm)
    # precision tp / (tp + fp)
    precision_svm = precision_score(y_test, y_pred)
    print('Precision: %f' % precision_svm)
    # recall: tp / (tp + fn)
    recall_svm = recall_score(y_test, y_pred)
    print('Recall: %f' % recall_svm)
    # f1: 2 tp / (2 tp + fp + fn)
    f1_svm = f1_score(y_test, y_pred)
    print('F1 score: %f' % f1_svm)
```

```
Accuracy: 0.894737
Precision: 0.875000
```

Figure 14: SVM classifier