Codes Appendix

Code 1: pre-processing the data (termpaper_cll788_preprocessing.ipnyb)

```
import keras

from keras.preprocessing import image

from skimage.feature import canny

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tqdm import tqdm

import seaborn as sns

import sys

import skimage.io

from skimage.transform import resize

from imgaug import augmenters as iaa

import cv2

import keras.backend as K

from skimage import color

import tensorflow as tf

import warnings

warnings.filterwarnings("ignore")

# **Load the dataset**

train = pd.read_csv('../input/aptos2019-blindness-detection/train.csv') #Train data input

test = pd.read_csv('../input/aptos2019-blindness-detection/test.csv') #Test data input

train_y = train["diagnosis"]

# **Dataset Visualisation**

# Label wise distribution of data

sns.countplot(x="diagnosis", data=train)
```

```
#Let's see the size of an image in train and test dataset

image1 = cv2.imread(f"../input/aptos2019-blindness-
detection/train_images/{train['id_code'][1000]}.png")

image1.shape

# ##### Display 5 images from each class

fig = plt.figure(figsize=(25, 16))

for class_id in sorted(train_y.unique()):

   for i, (idx, row) in enumerate(train.loc[train['diagnosis'] == class_id].sample(5,
random_state=77).iterrows()):

      ax = fig.add_subplot(5, 5, class_id*5 + i + 1, xticks=[], yticks=[])

      path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"

      image = cv2.imread(path)

      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

      image = cv2.resize(image, (512, 512))

      plt.imshow(image)

      ax.set_title('Label: %d-%d-%s' % (class_id, idx, row['id_code']) )

# ##### Display Gray Scaled Gaussian Weighted 5 images from each class

fig = plt.figure(figsize=(25, 16))

for class_id in sorted(train_y.unique()):

   for i, (idx, row) in enumerate(train.loc[train['diagnosis'] == class_id].sample(5,
random_state=77).iterrows()):

      ax = fig.add_subplot(5, 5, class_id * 5 + i + 1, xticks=[], yticks=[])

      path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"

      image = cv2.imread(path)

      image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

      image = cv2.resize(image, (512, 512))

      image=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , 512/10) ,-4 ,128)

      plt.imshow(image, cmap='gray')

      ax.set_title('Label: %d-%d-%s' % (class_id, idx, row['id_code']) )

# ##### Function to crop grayscaled images
```

```
def crop_gray(img,tol=7):
    if img.ndim ==2:
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))]
    elif img.ndim==3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img>tol
        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape != 0):
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
            img = np.stack([img1,img2,img3],axis=-1)
        else:
            return img
        return img
# ##### Function to Crop colored images
def crop_image(img, sigmaX):
    img = cv2.imread(img)
    img = crop_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    height, width, depth = img.shape
    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y))
    new_img = np.zeros((height, width), np.uint8)
    cv2.circle(new_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=new_img)
    img = crop_gray(img)
```

```
    img=cv2.addWeighted ( img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img
# ##### Function to plot edges from images
def edge_Detect(img):
    img = color.rgb2gray(img)
    edges = canny(img)
    return edges
# ##### Trying crop_image on the sample image set
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(train.loc[train['diagnosis'] == class_id].sample(5,
random_state=77).iterrows()):
        ax = fig.add_subplot(5, 5, class_id * 5 + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = crop_image(path,30)
        plt.imshow(image)
        ax.set_title('%d-%d-%s' % (class_id, idx, row['id_code']) )
# ##### Display crop images only edges (Edge Detection)
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(train.loc[train['diagnosis'] == class_id].sample(5,
random_state=77).iterrows()):
        ax = fig.add_subplot(5, 5, class_id * 5 + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = crop_image(path,30)
        image = edge_Detect(image)
        plt.imshow(image)
        ax.set_title('%d-%d-%s' % (class_id, idx, row['id_code']) )
```

Code 2: Training the data (termpaper_cll788_models.ipnyb)

```
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, MaxPool2D , Flatten , Dropout, Dense
from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import seaborn as sns
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
import sys
import skimage.io
from skimage import color
from skimage.transform import resize
from skimage.feature import canny
import cv2
from keras.losses import categorical_crossentropy
from keras.applications.resnet50 import preprocess_input
import keras.backend as K
import tensorflow as tf
from keras.utils import Sequence
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
IMG_SIZE = 224
```

```
# ##### Load Dataset
train = pd.read_csv('../input/aptos2019-blindness-detection/train.csv')
test = pd.read_csv('../input/aptos2019-blindness-detection/test.csv')
# ##### Image Preprocessing functions as in previous code
# function to shorten gray images
def crop_gray(img,tol=7):
    if img.ndim ==2:
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))]
    elif img.ndim==3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img>tol

        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape != 0):
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
            img = np.stack([img1,img2,img3],axis=-1)
        else:
            return img
        return img
# function to crop images in a circle
def crop_image(img, sigmaX):
    img = cv2.imread(img)
    img = crop_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    height, width, depth = img.shape
    x = int(width/2)
```

```
    y = int(height/2)
    r = np.amin((x,y))
    new_img = np.zeros((height, width), np.uint8)
    cv2.circle(new_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=new_img)
    img = crop_gray(img)
    img=cv2.addWeighted ( img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img
# function to detect and display edges in the image
def edge_Detect(img):
    img = color.rgb2gray(img)
    edges = canny(img)
    return edges
# ##### Preprocessing the whole train dataset
def preProcessTrain(channel,train):
    train_image = []
    X =np.array([])
    if channel==1:
        for i in tqdm(range(train.shape[0])):
            path=f"../input/aptos2019-blindness-detection/train_images/{train['id_code'][i]}.png"
            img = cv2.imread(path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (224, 224))
#             img = crop_image(path,30)
#             img = edge_Detect(img)
            img=cv2.addWeighted ( img,4, cv2.GaussianBlur( img , (0,0) , 30) ,-4 ,128)
            img_array = img/255
            train_image.append(img_array)
        X = np.array(train_image)
```

```
        X = X.reshape(list(X.shape) + [1])
    else:
        for i in tqdm(range(train.shape[0])):
            img = image.load_img('../input/aptos2019-blindness-
detection/train_images/'+train['id_code'][i]+'.png', target_size=(IMG_SIZE,IMG_SIZE,3), grayscale=False)
        #     img = circle_crop (img,30)
#             img = edge_Detect(img)
            img_array = image.img_to_array(img)
            img_array =img_array/255
            train_image.append(img_array)
        X = np.array(train_image)
    return X
# ##### data augmenter to increase the train data size
dataAugmenter = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range = 30,
        zoom_range = 0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip = True,
        vertical_flip=False)
# ##### Function to plot accuracy and loss for training and testing set
def plot_results(result,epoch_range):
    acc = result.history['accuracy']
    val_acc = result.history['val_accuracy']
```

```
    loss = result.history['loss']
    val_loss = result.history['val_loss']
    plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epoch_range, acc, label='Training Accuracy')
    plt.plot(epoch_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')
    plt.subplot(2, 2, 2)
    plt.plot(epoch_range, loss, label='Training Loss')
    plt.plot(epoch_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
# ### Model 1 (Simple CNN Model single channel)
# ##### preprocessing the training dataset for model 1 (as 1 channel is required)
X = preProcessTrain(1,train)
Y=train['diagnosis'].values
Y = to_categorical(Y)
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=42, test_size=0.2,stratify=Y)
dataAugmenter.fit(X_train)
#model 1
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(224,224,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(224, activation='relu'))
```

```
model.add(Dropout(0.5))

model.add(Dense(5, activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])

#computing model1 results

result1=model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

#plotting model1 accuracy

plot_results(result1,range(10))

# ##### preprocessing data for model 2 and 3 (both uses 3 channels)

X = preProcessTrain(3,train)

Y=train['diagnosis'].values

Y = to_categorical(Y)

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=42, test_size=0.2,stratify=Y)

#fitting dataset in data augmenter

dataAugmenter.fit(X_train)

# ### Model 2 (3 layer CNN model Keras Image Classification) first set of CONV => RELU => MAX POOL layers

# model 2

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(224,224,3)))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(224, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(5, activation='softmax'))
adam = Adam(lr=1e-3, decay=1e-3/ 15)
model.compile(loss="categorical_crossentropy", optimizer=adam, metrics=["accuracy"])
#computing model 2 results
result2=model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
#plotting model 2 accuracy
plot_results(result2,range(10))
# ### Model 3 (Transfer Learning Model)
pretrained_model = tf.keras.applications.MobileNetV2(input_shape = (224, 224, 3),
                                include_top = False,
                                weights = "imagenet")
pretrained_model.trainable = False
model = tf.keras.Sequential([pretrained_model,
                         tf.keras.layers.GlobalAveragePooling2D(),
                         tf.keras.layers.Dropout(0.25),
                         tf.keras.layers.Dense(5, activation="softmax")
                        ])
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),
          loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
          metrics=['accuracy'])
#computing model 3 results
result3 = model.fit(X_train,y_train,epochs = 100 , validation_data = (X_test, y_test))
#plotting model 3 results
plot_results(result3,range(100))
# ##### Preprocessing and predicting labels for the test image
```

```
test_image =[]
for i in tqdm(range(test.shape[0])):
    img = image.load_img('../input/aptos2019-blindness-detection/test_images/'+test['id_code'][i]+'.png', target_size=(IMG_SIZE,IMG_SIZE,3), grayscale=False)
#     img = crop_image (img,30)
#     img = edge_Detect(img)
    img_array = image.img_to_array(img)
    img_array =img_array/255
    test_image.append(img_array)
# ##### Test Labels
X_Test = np.array(test_image)
test_predict = model.predict_classes(X_Test) #stores the test labels
test_predict
sns.countplot(x=test_predict)
plt.title('countplot for test labels predicted using our model')
# ##### Adding diagnosis labels column on test dataset
test['diagnosis'] = test_predict
test.head()
```