Traing the model for Diabetic Retinopathy Detection

Mounting the google drive

In [1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [8]:
```
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Kaggle"
```

In [9]:
```
#changing the working directory
%cd /content/gdrive/My Drive/Kaggle
```

```
/content/gdrive/My Drive/Kaggle
```

In [10]:
```
pwd
```

Out[10]:
```
'/content/gdrive/My Drive/Kaggle'
```

Importing the necessary libraries

In [26]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras import optimizers
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import random
import pickle
```

Initialising the TPU cores provided by Colab

In [ ]:

```
%tensorflow_version 2.x
print("Tensorflow version " + tf.__version__)

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()  # TPU detection
    print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])
except ValueError:
    raise BaseException('ERROR: Not connected to a TPU runtime; please see the previous cell in this notebook for ins

tf.config.experimental_connect_to_cluster(tpu)
tf.tpu.experimental.initialize_tpu_system(tpu)
tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
```

Defining the path of image files to be trained and tested.

In [11]:

```
DATADIR = "/content/gdrive/My Drive/Kaggle/gaussian_filtered_images/gaussian_filtered_images"
CATEGORIES = ['No_DR','Mild','Moderate','Severe','Proliferate_DR'] # five categories of Diabetic Retinopathy we are d
```

Converting the image files to array and assigning them categories

In [12]:

```
with tpu_strategy.scope():
    dataset = []
    for category in CATEGORIES: #looping through the five folders in the path and joing the folder name in the path
        path = os.path.join(DATADIR,category)
        class_category = CATEGORIES.index(category)
        for im in os.listdir(path): # looping through all the images in the above joined path.
            try:
                img_array = cv2.imread(os.path.join(path,im)) #converting image to array
                img_res = cv2.resize(img_array,(224,224)) # resizing the array to 224x224
                dataset.append([img_res, class_category]) # appending the array and category to the list
            except Exception as e:
                pass
```

In [13]:

```
X=[]
y=[]
```

In [14]:
```
# Separating the array and the categories as X and y respectively
for features, label in dataset:
    X.append(features)
    y.append(label)
```

In [15]:
```
# Splitting the data into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
```

In [16]:
```
# converting the dataset into numpy array
X_train = np.array(X_train).reshape(-1,224,224,3)
X_test = np.array(X_test).reshape(-1,224,224,3)
y_train=np.array(y_train).reshape(-1,1)
```

Defining the architecture of the CNN model

In [60]:

```
def create_model():
    model = Sequential()
# five layers of Conv2D and MaxPooling2D with different number of channels in each layer
    model.add(Conv2D(16, (3,3), strides=(1,1), activation="relu", input_shape = X_train.shape[1:]))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(32, (3,3), strides=(1,1), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(64, (3,3), strides=(1,1), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(64, (3,3), strides=(1,1), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(64, (3,3), strides=(1,1), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2)))

# Two dense layers and softmax activation to get probabilities value for our five different categories for each input
    model.add(Flatten())
    model.add(Dense(128, activation = 'relu'))
    model.add(Dense(5, activation = 'softmax'))
# Using adam optimiser and chasing the accuract metric in the compile phase of the model.
# Because we have five categories hence using categorical_crossentropy as loss function.
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

In [61]:

```
with tpu_strategy.scope():
    model = create_model()
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_80 (Conv2D)           (None, 222, 222, 16)      448
_________________________________________________________________
max_pooling2d_80 (MaxPooling (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_81 (Conv2D)           (None, 109, 109, 32)      4640
_________________________________________________________________
max_pooling2d_81 (MaxPooling (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_82 (Conv2D)           (None, 52, 52, 64)        18496
_________________________________________________________________
max_pooling2d_82 (MaxPooling (None, 26, 26, 64)        0
_________________________________________________________________
conv2d_83 (Conv2D)           (None, 24, 24, 64)        36928
_________________________________________________________________
max_pooling2d_83 (MaxPooling (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_84 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_84 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_16 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_32 (Dense)             (None, 128)               204928
_________________________________________________________________
dense_33 (Dense)             (None, 5)                 645
=================================================================
Total params: 303,013
Trainable params: 303,013
Non-trainable params: 0
_________________________________________________________________
```

Fitting the model.

In [62]:
```
with tpu_strategy.scope():
    model.fit(X_train, y_train, batch_size=16, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
156/156 [==============================] - 16s 62ms/step - loss: 2.5709 - accuracy: 0.5788 - val_loss: 0.7860 - val_accu
racy: 0.7143
Epoch 2/10
156/156 [==============================] - 5s 30ms/step - loss: 0.7710 - accuracy: 0.7277 - val_loss: 0.7510 - val_accur
acy: 0.7159
Epoch 3/10
156/156 [==============================] - 5s 31ms/step - loss: 0.7379 - accuracy: 0.7409 - val_loss: 0.7671 - val_accur
acy: 0.7207
Epoch 4/10
156/156 [==============================] - 5s 30ms/step - loss: 0.7172 - accuracy: 0.7632 - val_loss: 0.7335 - val_accur
acy: 0.7368
Epoch 5/10
156/156 [==============================] - 5s 31ms/step - loss: 0.6605 - accuracy: 0.7522 - val_loss: 0.7230 - val_accur
acy: 0.7319
Epoch 6/10
156/156 [==============================] - 5s 30ms/step - loss: 0.6120 - accuracy: 0.7793 - val_loss: 0.6943 - val_accur
acy: 0.7448
Epoch 7/10
156/156 [==============================] - 5s 30ms/step - loss: 0.5819 - accuracy: 0.7792 - val_loss: 0.7887 - val_accur
acy: 0.7319
Epoch 8/10
156/156 [==============================] - 5s 34ms/step - loss: 0.5490 - accuracy: 0.8031 - val_loss: 0.8023 - val_accur
acy: 0.7287
Epoch 9/10
156/156 [==============================] - 5s 31ms/step - loss: 0.5000 - accuracy: 0.8261 - val_loss: 0.8646 - val_accur
acy: 0.7287
Epoch 10/10
156/156 [==============================] - 5s 31ms/step - loss: 0.4291 - accuracy: 0.8365 - val_loss: 0.9192 - val_accur
acy: 0.7352
```

Extracting the predicted category result on the test dataset

In [63]:
```
Y_predict=[]
prediction = model.predict(X_test) # predicting on the test data
for pr in prediction:
    Y_predict.append(np.argmax(pr)) # getting the index value of the max probability value of the output array
```

In [65]:
```
Y_predict[:10]
```

Out[65]:
```
[2, 0, 0, 0, 0, 0, 4, 1, 0, 2]
```

In [66]:
```
conf_matrix = confusion_matrix(y_test, Y_predict) # creating confusion matrix of predicted values and actual test val
```

In [67]:
```
np.trace(conf_matrix) # trace of a matrix gives the sum of diagonal elements ehcih are the correctly predicted values
                      # model on the test dataset
```

Out[67]:
```
420
```

In [68]:
```
conf_matrix
```

Out[68]:
```
array([[273,   4,   1,   0,   0],
       [  1,  19,  26,   0,   3],
       [ 13,  14, 117,   3,  10],
       [  1,   0,  15,   5,   2],
       [  5,   2,  27,   3,   6]])
```

In [69]:
```
np.sum(conf_matrix)
```

Out[69]:
```
550
```

In [70]:
```
420/550 #out of 550 test data, the model predicted 420 correctly.
```

Out[70]:
```
0.7636363636363637
```

Dumping the dataset as a pickle file for later use

In [71]:
```
dataset_in = open('dataset.pickle', 'wb')
pickle.dump(dataset, dataset_in)
dataset_in.close()
```

In [ ]:
```
input_X = open('dataset.pickle','rb')
dataset = pickle.load(input_X)
```

In [72]:
```
model.save('diab_retin.h5') # saving the model to use it in the frontend application
```

In [ ]:
```

```