Configuration Manual

MSc Research Project
MSc Data Analytics

Abhilash Chava
Student ID: X21178712

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Abhilash Chava
Student ID:	X21178712
Programme:	MSc Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	652
Page Count:	37

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:	*Abhilash Chava*
Date:	26th August 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

Abhilash Chava
X21178712

1 Introduction

Deep learning algorithms play a critical part in the approach used to diagnose diabetic retinopathy in retinal pictures. On retinal fundus pictures, several configurations, such as supervised and self-supervised models, as well as the unique Vision Transformer, were carefully evaluated. This study's system requirements centered on the capacity to handle and analyze high-resolution pictures, providing accurate and rapid model training. These requirements highlight the need of strong processing power, adequate storage, and specialized software libraries for properly deploying and evaluating the suggested models.

2 System Requirements

Component	**Specification**
RAM	16GB
GPU	4GB NVIDIA
Processor	Core i5, Windows
Platform	Jupyter Notebook and Google Colab

Table 1: System Requirements

3 Modeling and Evaluation

This study used sophisticated deep learning algorithms to diagnose diabetic retinopathy from retinal pictures, testing with a variety of configurations ranging from typical supervised models to revolutionary self-supervised ones. The Vision Transformer (ViT) was a major addition, indicating a trend away from traditional convolutional networks and towards designs capable of collecting complicated retinal picture patterns. The modeling process was expedited by the usage of platforms such as Jupyter Notebook and Google Colab. The comprehensive examination of models on retinal fundus pictures was critical to the study's reliability. While no specific metrics were mentioned in the extracted content, the mention of DenseNet's performance based on Kappa values suggests a thorough assessment approach, emphasizing not only accuracy but also predictability and consistency, ensuring the models' practical viability in medical contexts.

Google Colab Link

RETINOPATHY

```
!pip install tensorflow-addons==0.16.1
import tensorflow_addons as tfa
```

```
# Import the necessary packages
import os
import os.path
import os, shutil
from os import listdir

import numpy as np
import pandas as pd

import cv2

import tensorflow as tf
from tensorflow import keras

import glob
import PIL
from PIL import Image
from pathlib import Path

from pylab import rcParams

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.utils import shuffle
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score
```

```
from tensorflow.keras.applications import VGG19
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2

import tensorflow_addons as tfa
from IPython.display import display
from tensorflow.keras import backend as K
from tensorflow.keras.utils import plot_model, to_categorical
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.models import Model, load_model, Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import GlobalAveragePooling2D, Input, AveragePooling2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import BatchNormalization, ZeroPadding2D, Conv2D, Activation, MaxPooling2D, MaxPool2D, Add
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LearningRateScheduler
```

The above packages are used in the designing of the code.

```python
# Import Tensorflow utility device_lib to get more information about the GPU
# This will help verify if TF is using the intended GPU
from tensorflow.python.client import device_lib

# Helper function to get information about all available GPUs in a list
def get_available_gpus():
    local_device_protos = device_lib.list_local_devices()
    return [x for x in local_device_protos if x.device_type == 'GPU']
# Get information of all available GPUs
gpu_info = get_available_gpus()
# Display GPU information
for i, gpu in enumerate(gpu_info):
    print(f"################## GPU: {i} ##########################")
    print(f"Device Type: {gpu.device_type}")
    print(f"GPU ID: {gpu.name}")
    print(f"Physical Device Description:\n\t{gpu.physical_device_desc}")
    print("######################################################")
```

```python
from google.colab import files
files.upload() #upload your kaggle.json file

!mkdir ~/.kaggle #create a directory called .kaggle in the root folder
!cp kaggle.json ~/.kaggle/ #copy kaggle.json to this folder
!chmod 600 ~/.kaggle/kaggle.json #add full rights to this copied file
!rm kaggle.json #remove the original one

!kaggle datasets download -d tanlikesmath/diabetic-retinopathy-resized
!unzip diabetic-retinopathy-resized.zip  #unzip the zip file
!rm diabetic-retinopathy-resized.zip
```

```python
df = pd.read_csv("trainLabels_cropped.csv")
df
```

	Unnamed: 0	Unnamed: 0.1	image	level
0	0	0	10_left	0
1	1	1	10_right	0
2	2	2	13_left	0
3	3	3	13_right	0
4	4	4	15_left	1
...	...	...	...	...
35103	35104	35121	44347_right	0
35104	35105	35122	44348_left	0
35105	35106	35123	44348_right	0
35106	35107	35124	44349_left	0
35107	35108	35125	44349_right	1

35108 rows × 4 columns

The above screenshot is for the data acquisition in which the image information in the form of the file name and the level of the class to which it belongs.

```python
base_image_dir = "/content/resized_train_cropped"
df['path'] = df['image'].map(lambda x: os.path.join(base_image_dir,'resized_train_cropped','{}.jpeg'.format(x)))
df = df.drop(columns=['image'])
df = df.sample(frac=1).reset_index(drop=True) #shuffle dataframe
df['level'] = (df['level'] > 1).astype(int) # Disease or no disease
df.head(10)
```

	Unnamed: 0	Unnamed: 0.1	level	path
0	27887	27903	0	/content/resized_train_cropped/resized_train_c...
1	892	893	0	/content/resized_train_cropped/resized_train_c...
2	30340	30356	0	/content/resized_train_cropped/resized_train_c...
3	29506	29522	0	/content/resized_train_cropped/resized_train_c...
4	28119	28135	0	/content/resized_train_cropped/resized_train_c...
5	6878	6882	0	/content/resized_train_cropped/resized_train_c...
6	10874	10881	0	/content/resized_train_cropped/resized_train_c...
7	15511	15521	0	/content/resized_train_cropped/resized_train_c...
8	32370	32386	0	/content/resized_train_cropped/resized_train_c...
9	10507	10514	0	/content/resized_train_cropped/resized_train_c...

```python
df = df[:10000]
len(df)
```

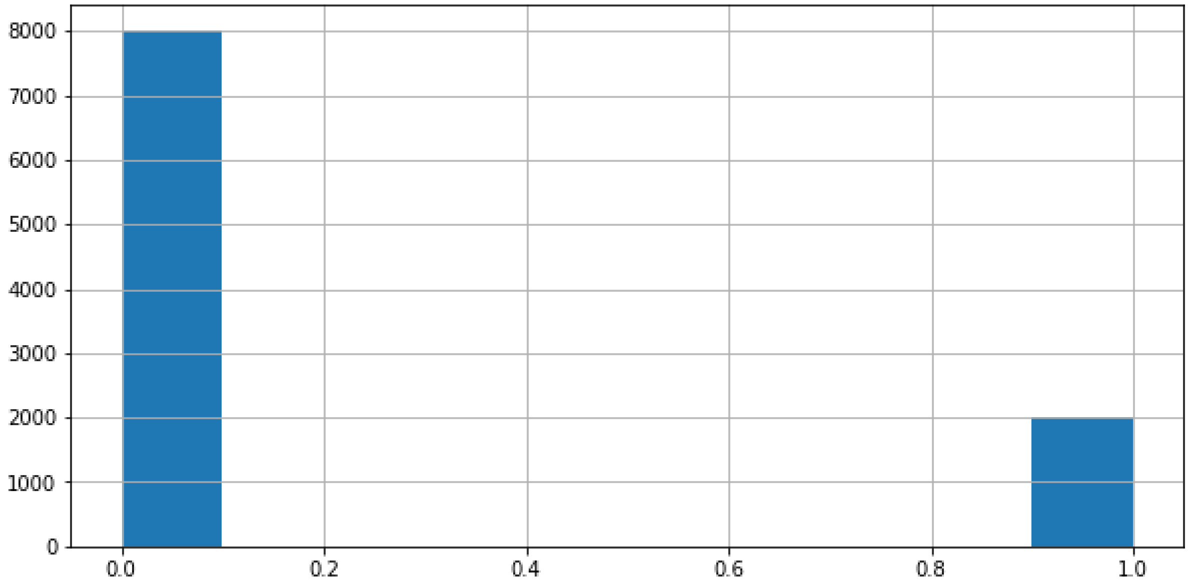
```python
# Separate majority and minority classes
df_majority = df[df.iloc[:,4608]==1]
df_minority = df[df.iloc[:,4608]==0]

# Downsample majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=1000)
#Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,
                                 n_samples=1000)
# Combine minority class with downsampled majority class
df_up_down_sampled = pd.concat([df_majority_downsampled, df_minority_upsampled])
```

```python
df['level'].unique()
```

```
array([0, 1])
```

For clarity in this analysis, cropped images were used.

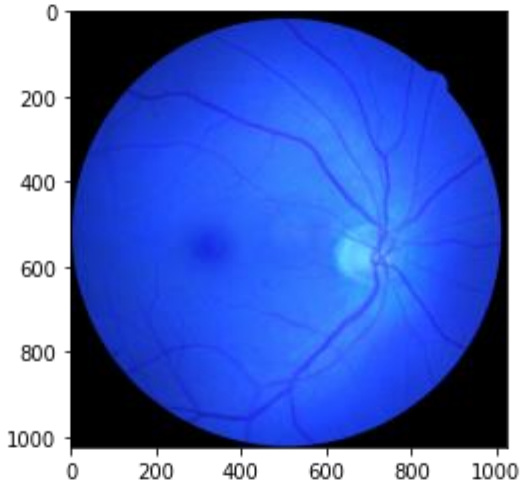
```python
df['level'].hist(figsize = (10, 5))
```




```python
df['path'][0]
```

```
'/content/resized_train_cropped/resized_train_cropped/35336_right.jpeg'
```

```python
img = cv2.imread(df['path'][0])
plt.imshow(img)
```



The image preprocessing which is mentioned in the thesis.

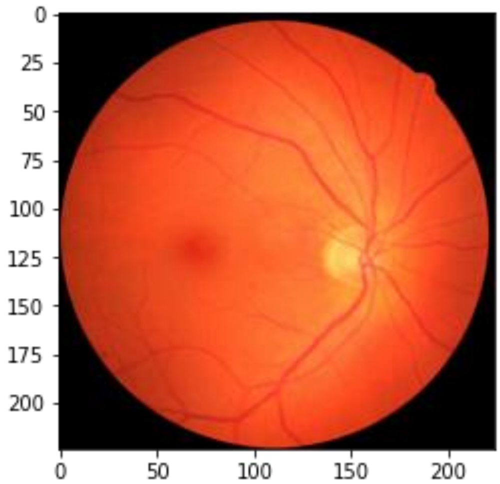
```
dim = (224, 224)
X_image_train = []
Y_image = []
print(len(df))
for i in range(0, len(df)):
    img = Image.open(df['path'][i]).convert('RGB')
    im_resized = img.resize(dim)
    X_image_train.append(im_resized)
    Y_image.append(df['level'][i])
print(len(X_image_train))
```

```
10000
10000
```

```
X_image_array=[]
for x in range(len(X_image_train)):
    X_image=np.array(X_image_train[x],dtype='uint8')
    X_image_array.append(X_image)
features = np.array(X_image_array)
print(features.shape)

labels= []
for x in Y_image:
    a = []
    a.append(x)
    labels.append(a)
labels = np.array(labels,dtype='uint8')
print(labels.shape)
```

```
(10000, 224, 224, 3)
(10000, 1)
```

```
plt.imshow(X_image_train[0])
```



Model Sampling data using Stratified KFold to keep the ration of classes intact.

```
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state = 5)
training = features
target = labels
for trn_idx, val_idx in folds.split(training, target):
    print("TRAIN:", trn_idx, "TEST:", val_idx)
    trainX, x_test = training[trn_idx], training[val_idx]
    trainY, y_test = target[trn_idx], target[val_idx]
    break

TRAIN: [   0    1    4 ... 9997 9998 9999] TEST: [   2    3   11 ... 9969 9987 9988]
```

Model Building.

```
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")
baseModel = ResNet50(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
```

```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
#headModel = Flatten(name="flatten")(headModel)
headModel = Dense(1024, activation="relu")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = GlobalAveragePooling2D()(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

```
res_model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
    layer.trainable = True
```

```
res_model.compile(optimizer = 'adam', loss = 'SparseCategoricalCrossentropy', metrics= ['accuracy'])
```

```
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
checkpointer = ModelCheckpoint(filepath="RES_weights.hdf5", verbose=1, save_best_only=True)
```

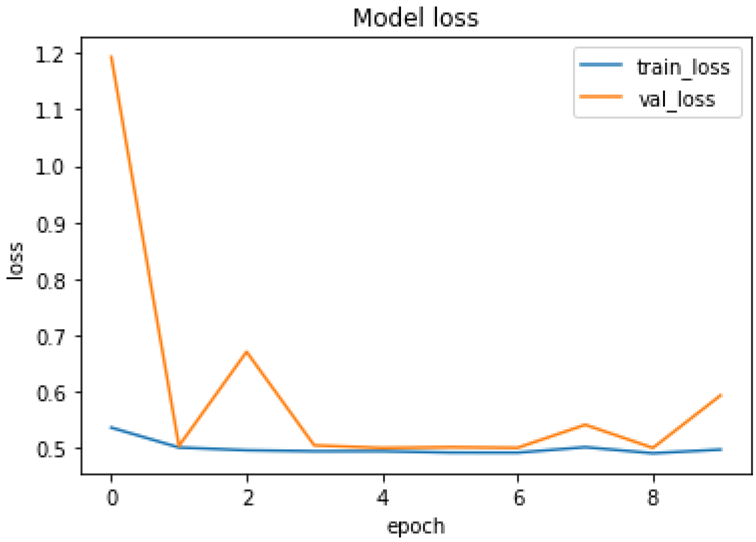
```
res_history = res_model.fit(x=trainX,
                            y=trainY,
                            batch_size=8,
                            epochs=50,
                            validation_split=0.1,
                            callbacks=[checkpointer, earlystopping],)
```

```
Epoch 1/50
900/900 [==============================] - ETA: 0s - loss: 0.5365 - accuracy: 0.8061
Epoch 1: val_loss improved from inf to 1.19285, saving model to RES_weights.hdf5
900/900 [==============================] - 79s 68ms/step - loss: 0.5365 - accuracy: 0.8061 - val_loss: 1.1929 - val_accuracy: 0.8000
Epoch 2/50
900/900 [==============================] - ETA: 0s - loss: 0.5013 - accuracy: 0.8075
Epoch 2: val_loss improved from 1.19285 to 0.50406, saving model to RES_weights.hdf5
900/900 [==============================] - 59s 66ms/step - loss: 0.5013 - accuracy: 0.8075 - val_loss: 0.5041 - val_accuracy: 0.8000
Epoch 3/50
900/900 [==============================] - ETA: 0s - loss: 0.4964 - accuracy: 0.8075
Epoch 3: val_loss did not improve from 0.50406
900/900 [==============================] - 58s 64ms/step - loss: 0.4964 - accuracy: 0.8075 - val_loss: 0.6710 - val_accuracy: 0.8000
Epoch 4/50
900/900 [==============================] - ETA: 0s - loss: 0.4946 - accuracy: 0.8075
Epoch 4: val_loss did not improve from 0.50406
900/900 [==============================] - 58s 65ms/step - loss: 0.4946 - accuracy: 0.8075 - val_loss: 0.5051 - val_accuracy: 0.8000
Epoch 5/50
900/900 [==============================] - ETA: 0s - loss: 0.4947 - accuracy: 0.8074
Epoch 5: val_loss improved from 0.50406 to 0.50045, saving model to RES_weights.hdf5
900/900 [==============================] - 59s 66ms/step - loss: 0.4947 - accuracy: 0.8074 - val_loss: 0.5005 - val_accuracy: 0.8000
Epoch 6/50
900/900 [==============================] - ETA: 0s - loss: 0.4923 - accuracy: 0.8075
Epoch 6: val_loss did not improve from 0.50045
900/900 [==============================] - 58s 65ms/step - loss: 0.4923 - accuracy: 0.8075 - val_loss: 0.5019 - val_accuracy: 0.8000
Epoch 7/50
...
900/900 [==============================] - ETA: 0s - loss: 0.4976 - accuracy: 0.8074
Epoch 10: val_loss did not improve from 0.50045
900/900 [==============================] - 58s 65ms/step - loss: 0.4976 - accuracy: 0.8074 - val_loss: 0.5937 - val_accuracy: 0.8000
Epoch 10: early stopping
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Performance Curves

```
plt.plot(res_history.history['loss'])
plt.plot(res_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()
```




```
res_model.load_weights("RES_weights.hdf5")

evaluate = res_model.evaluate(x_test, y_test)
print('Accuracy Test : {}'.format(evaluate[1]))
```

```
# import cv2
prediction = []
original = y_test
image = []
count = 0

for item in x_test:
    img= item
    img = img.reshape(-1,224,224,3)
    predict = res_model.predict(img)
    predict = np.argmax(predict)
    prediction.append(predict)

# Getting the test accuracy
score = accuracy_score(original, prediction)
print("Test Accuracy : {}".format(score))
```

```python
class_names = ['No DR','DR']
import random
plt.figure(figsize = (20,20))
for i in range(9):
    random_int_index = random.choice(range(len(x_test)))
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[random_int_index])
    if prediction[random_int_index] == original[random_int_index]:
        color = "g"
    else:
        color = "r"
    plt.title("True Label: " + class_names[original[random_int_index][0]] + " || " + "Predicted Label: " +
              class_names[prediction[random_int_index]])
    plt.axis(False);
```

```python
# accuracy, sensitivity, and specificity
cm = confusion_matrix(np.asarray(original), np.asarray(prediction))
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```python
# Print out the classification report
print(classification_report(np.asarray(original), np.asarray(prediction)))
```

```python
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")
baseModel = VGG19(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
baseModel.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
...
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
```

```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
#headModel = Flatten(name="flatten")(headModel)
headModel = Dense(1024, activation="relu")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = GlobalAveragePooling2D()(headModel)
headModel = Dense(2, activation="softmax")(headModel)

vgg_model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
    layer.trainable = False
```

```
vgg_model.compile(optimizer = 'adam', loss =
                   'SparseCategoricalCrossentropy', metrics= ['accuracy'])
```

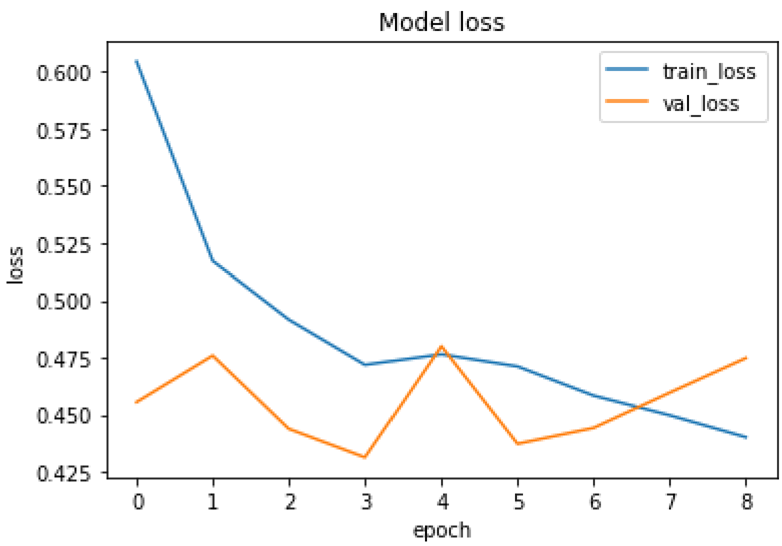
```
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
checkpointer = ModelCheckpoint(filepath="VGG_weights.hdf5", verbose=1, save_best_only=True)
```

```python
print("[INFO] training head...")
vgg_history = vgg_model.fit(x=trainX,
                            y=trainY,
                            batch_size=8,
                            epochs=50,
                            validation_split=0.1,
                            callbacks=[checkpointer, earlystopping],)
```

```
Epoch 1/50
449/450 [============================>.] - ETA: 0s - loss: 0.6042 - accuracy: 0.7901
Epoch 1: val_loss improved from inf to 0.45553, saving model to VGG_weights.hdf5
450/450 [==============================] - 18s 37ms/step - loss: 0.6044 - accuracy: 0.7900 - val_loss: 0.4555 - val_accuracy: 0.8175
Epoch 2/50
449/450 [============================>.] - ETA: 0s - loss: 0.5172 - accuracy: 0.8032
Epoch 2: val_loss did not improve from 0.45553
450/450 [==============================] - 16s 36ms/step - loss: 0.5172 - accuracy: 0.8031 - val_loss: 0.4759 - val_accuracy: 0.8175
Epoch 3/50
449/450 [============================>.] - ETA: 0s - loss: 0.4923 - accuracy: 0.8026
Epoch 3: val_loss improved from 0.45553 to 0.44385, saving model to VGG_weights.hdf5
450/450 [==============================] - 16s 36ms/step - loss: 0.4916 - accuracy: 0.8031 - val_loss: 0.4438 - val_accuracy: 0.8175
Epoch 4/50
449/450 [============================>.] - ETA: 0s - loss: 0.4719 - accuracy: 0.8040
Epoch 4: val_loss improved from 0.44385 to 0.43135, saving model to VGG_weights.hdf5
450/450 [==============================] - 16s 36ms/step - loss: 0.4718 - accuracy: 0.8039 - val_loss: 0.4314 - val_accuracy: 0.8175
Epoch 5/50
449/450 [============================>.] - ETA: 0s - loss: 0.4766 - accuracy: 0.8032
Epoch 5: val_loss did not improve from 0.43135
450/450 [==============================] - 16s 36ms/step - loss: 0.4764 - accuracy: 0.8033 - val_loss: 0.4800 - val_accuracy: 0.8175
Epoch 6/50
449/450 [============================>.] - ETA: 0s - loss: 0.4704 - accuracy: 0.8040
Epoch 6: val_loss did not improve from 0.43135
450/450 [==============================] - 16s 35ms/step - loss: 0.4711 - accuracy: 0.8042 - val_loss: 0.4373 - val_accuracy: 0.8250
...
449/450 [============================>.] - ETA: 0s - loss: 0.4405 - accuracy: 0.8082
Epoch 9: val_loss did not improve from 0.43135
450/450 [==============================] - 16s 35ms/step - loss: 0.4402 - accuracy: 0.8086 - val_loss: 0.4748 - val_accuracy: 0.8200
Epoch 9: early stopping
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
plt.plot(vgg_history.history['loss'])
plt.plot(vgg_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()
```

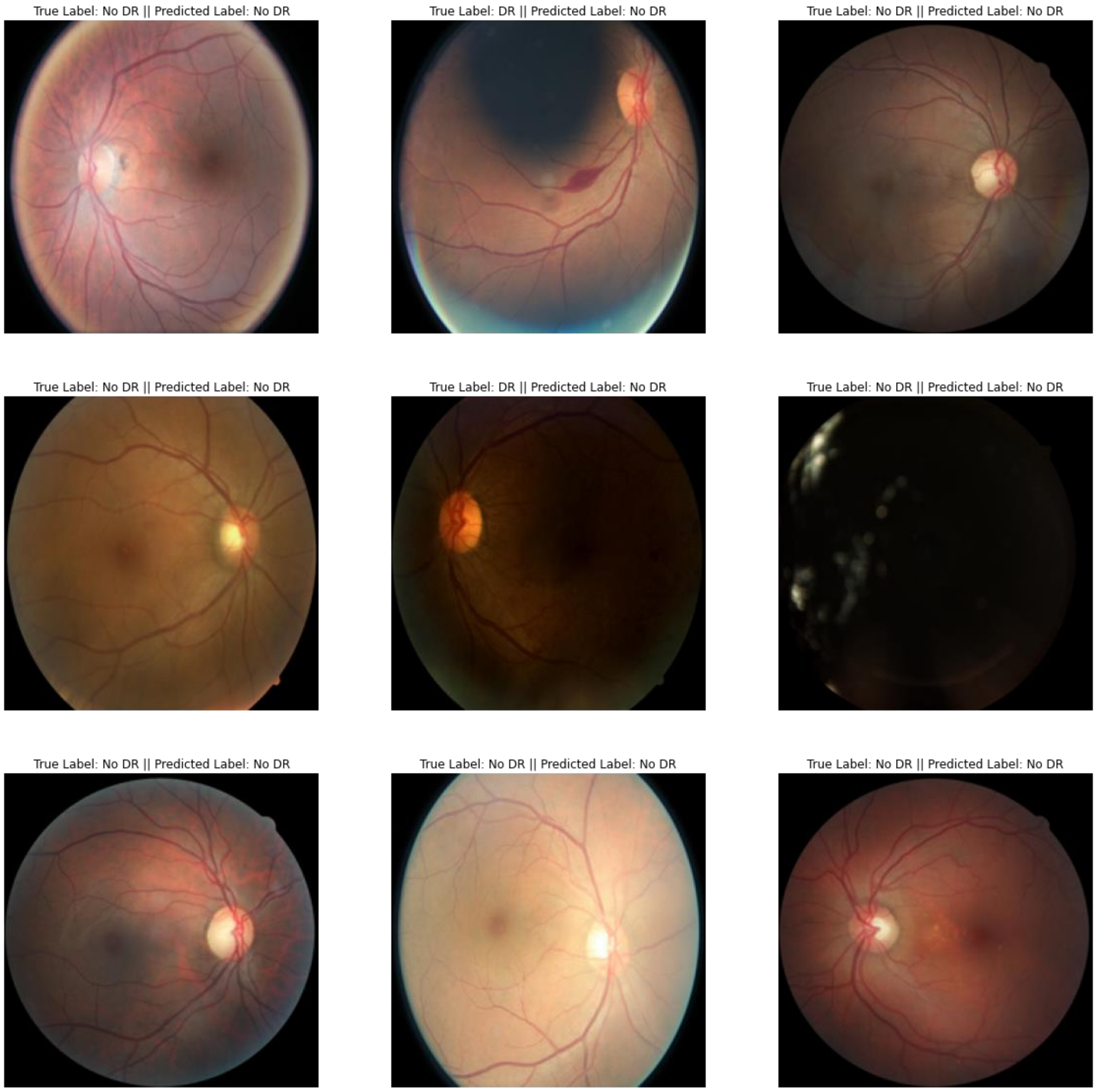



```
vgg_model.load_weights("VGG_weights.hdf5")

evaluate = vgg_model.evaluate(x_test, y_test)
print('Accuracy Test : {}'.format(evaluate[1]))
```

```
# import cv2
prediction = []
original = y_test
image = []
count = 0

for item in x_test:
    img= item
    img = img.reshape(-1,224,224,3)
    predict = vgg_model.predict(img)
    predict = np.argmax(predict)
    prediction.append(predict)

# Getting the test accuracy
score = accuracy_score(original, prediction)
print("Test Accuracy : {}".format(score))
```

```
class_names = ['No DR','DR']
import random
plt.figure(figsize = (20,20))
for i in range(9):
    random_int_index = random.choice(range(len(x_test)))
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[random_int_index])
    if prediction[random_int_index] == original[random_int_index]:
        color = "g"
    else:
        color = "r"
    plt.title("True Label: " + class_names[original[random_int_index][0]] + " || " + "Predicted Label: " +
              class_names[prediction[random_int_index]])
    plt.axis(False);
```

Final Prediction




```python
# accuracy, sensitivity, and specificity
cm = confusion_matrix(np.asarray(original), np.asarray(prediction))
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[806   0]
 [194   0]]
acc: 0.8060
sensitivity: 1.0000
specificity: 0.0000
```

```
# Print out the classification report
print(classification_report(np.asarray(original), np.asarray(prediction)))
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89       806
           1       0.00      0.00      0.00       194

    accuracy                           0.81      1000
   macro avg       0.40      0.50      0.45      1000
weighted avg       0.65      0.81      0.72      1000
```

```
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")
baseModel = DenseNet121(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
baseModel.summary()
```

```
Model: "densenet121"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 input_5 (InputLayer)           [(None, 224, 224, 3  0           []                               
                                )]                                                                
                                                                                                  
 zero_padding2d_2 (ZeroPadding2  (None, 230, 230, 3)  0          ['input_5[0][0]']                
 D)                                                                                               
                                                                                                  
 conv1/conv (Conv2D)            (None, 112, 112, 64  9408        ['zero_padding2d_2[0][0]']       
                                )                                                                 
                                                                                                  
 conv1/bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1/conv[0][0]']             
                                )                                                                 
                                                                                                  
 conv1/relu (Activation)        (None, 112, 112, 64  0           ['conv1/bn[0][0]']               
                                )                                                                 
                                                                                                  
 zero_padding2d_3 (ZeroPadding2  (None, 114, 114, 64  0          ['conv1/relu[0][0]']             
 D)                             )                                                                 
                                                                                                  
 pool1 (MaxPooling2D)           (None, 56, 56, 64)   0           ['zero_padding2d_3[0][0]']       
                                                                                                  
 conv2_block1_0_bn (BatchNormal  (None, 56, 56, 64)  256         ['pool1[0][0]']                  
...
Total params: 7,037,504
Trainable params: 6,953,856
Non-trainable params: 83,648
```

```python
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
#headModel = Flatten(name="flatten")(headModel)
headModel = Dense(1024, activation="relu")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = GlobalAveragePooling2D()(headModel)
headModel = Dense(2, activation="softmax")(headModel)

densenet_model = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:
    layer.trainable = False
```

```python
densenet_model.compile(optimizer = 'adam',
                       loss = 'SparseCategoricalCrossentropy', metrics= ['accuracy'])
```

```python
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=15)
checkpointer = ModelCheckpoint(filepath="DENSENET121_weights.hdf5", verbose=1, save_best_only=True)
```

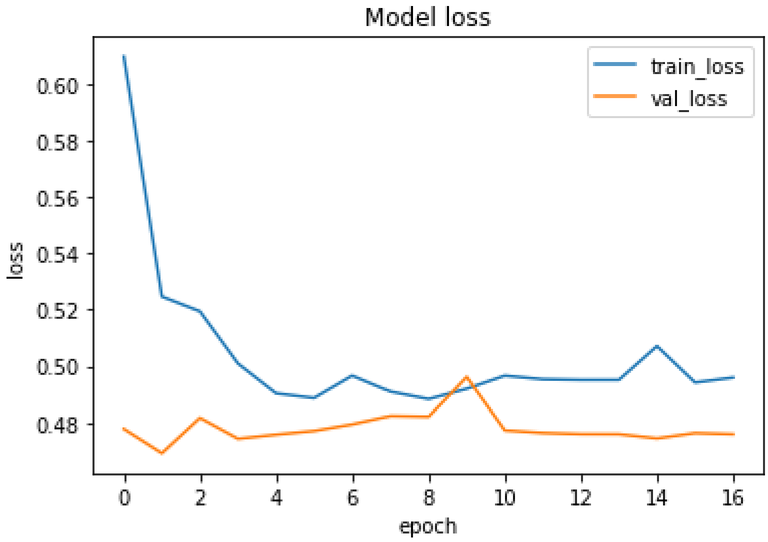
```python
print("[INFO] training head...")
densenet_history = densenet_model.fit(x=trainX,
                                      y=trainY,
                                      batch_size=8,
                                      epochs=50,
                                      validation_split=0.1,
                                      callbacks=[checkpointer, earlystopping],)
```

```
[INFO] training head...
Epoch 1/50
449/450 [===========================>.] - ETA: 0s - loss: 0.6107 - accuracy: 0.7912
Epoch 1: val_loss improved from inf to 0.47776, saving model to DENSENET121_weights.hdf5
450/450 [==============================] - 24s 34ms/step - loss: 0.6097 - accuracy: 0.7917 - val_loss: 0.4778 - val_accuracy: 0.8175
Epoch 2/50
448/450 [===========================>.] - ETA: 0s - loss: 0.5233 - accuracy: 0.8041
Epoch 2: val_loss improved from 0.47776 to 0.46917, saving model to DENSENET121_weights.hdf5
450/450 [==============================] - 13s 29ms/step - loss: 0.5246 - accuracy: 0.8033 - val_loss: 0.4692 - val_accuracy: 0.8175
Epoch 3/50
449/450 [===========================>.] - ETA: 0s - loss: 0.5200 - accuracy: 0.8009
Epoch 3: val_loss did not improve from 0.46917
450/450 [==============================] - 12s 27ms/step - loss: 0.5194 - accuracy: 0.8014 - val_loss: 0.4817 - val_accuracy: 0.8175
Epoch 4/50
448/450 [===========================>.] - ETA: 0s - loss: 0.5014 - accuracy: 0.8036
Epoch 4: val_loss did not improve from 0.46917
450/450 [==============================] - 12s 27ms/step - loss: 0.5009 - accuracy: 0.8039 - val_loss: 0.4743 - val_accuracy: 0.8175
Epoch 5/50
449/450 [===========================>.] - ETA: 0s - loss: 0.4905 - accuracy: 0.8037
Epoch 5: val_loss did not improve from 0.46917
450/450 [==============================] - 12s 27ms/step - loss: 0.4903 - accuracy: 0.8039 - val_loss: 0.4757 - val_accuracy: 0.8175
Epoch 6/50
450/450 [==============================] - ETA: 0s - loss: 0.4887 - accuracy: 0.8039
Epoch 6: val_loss did not improve from 0.46917
450/450 [==============================] - 12s 27ms/step - loss: 0.4887 - accuracy: 0.8039 - val_loss: 0.4770 - val_accuracy: 0.8175
...
450/450 [==============================] - ETA: 0s - loss: 0.4959 - accuracy: 0.8039
Epoch 17: val_loss did not improve from 0.46917
450/450 [==============================] - 12s 27ms/step - loss: 0.4959 - accuracy: 0.8039 - val_loss: 0.4759 - val_accuracy: 0.8175
Epoch 17: early stopping
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

DenseNet

```
plt.plot(densenet_history.history['loss'])
plt.plot(densenet_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()
```




```
densenet_model.load_weights("DENSENET121_weights.hdf5")

evaluate = densenet_model.evaluate(x_test, y_test)
print('Accuracy Test : {}'.format(evaluate[1]))
```

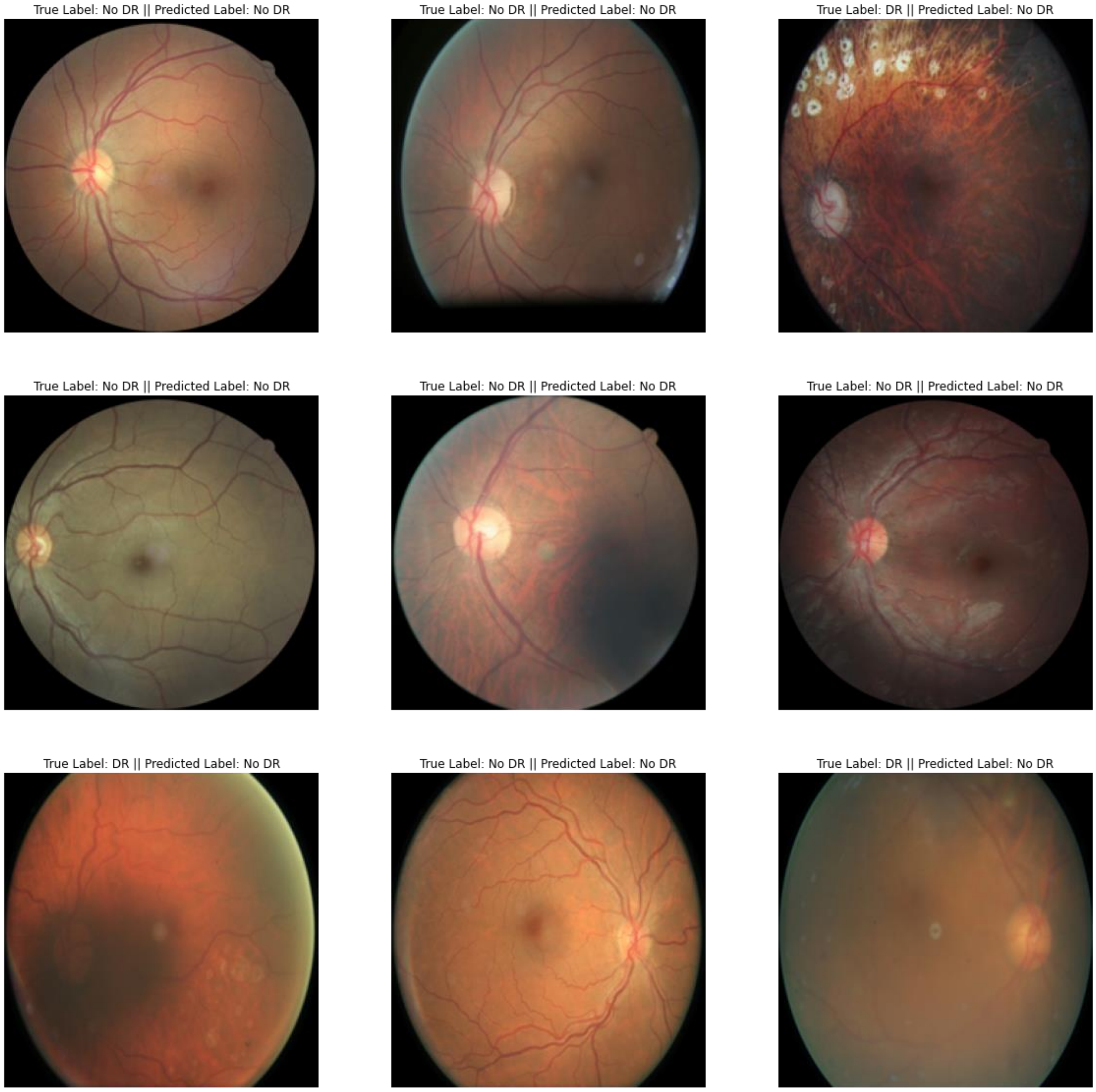
```
32/32 [==============================] - 4s 58ms/step - loss: 0.4908 - accuracy: 0.8060
Accuracy Test : 0.8059999942779541
```

```python
# import cv2
prediction = []
original = y_test
image = []
count = 0

for item in x_test:
    img= item
    img = img.reshape(-1,224,224,3)
    predict = densenet_model.predict(img)
    predict = np.argmax(predict)
    prediction.append(predict)

# Getting the test accuracy
score = accuracy_score(original, prediction)
print("Test Accuracy : {}".format(score))
```

```
Test Accuracy : 0.806
```

```python
class_names = ['No DR','DR']
import random
plt.figure(figsize = (20,20))
for i in range(9):
    random_int_index = random.choice(range(len(x_test)))
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[random_int_index])
    if prediction[random_int_index] == original[random_int_index]:
        color = "g"
    else:
        color = "r"
    plt.title("True Label: " + class_names[original[random_int_index][0]] + " || " + "Predicted Label: " +
              class_names[prediction[random_int_index]])
    plt.axis(False);
```

DenseNet Prediction




```
# accuracy, sensitivity, and specificity
cm = confusion_matrix(np.asarray(original), np.asarray(prediction))
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[806   0]
 [194   0]]
acc: 0.8060
sensitivity: 1.0000
specificity: 0.0000
```

```python
# Print out the classification report
print(classification_report(np.asarray(original), np.asarray(prediction)))
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89       806
           1       0.00      0.00      0.00       194

    accuracy                           0.81      1000
   macro avg       0.40      0.50      0.45      1000
weighted avg       0.65      0.81      0.72      1000
```

Mobilenet

```python
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")
baseModel = MobileNet(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))

baseModel.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_6 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 conv1 (Conv2D)              (None, 112, 112, 32)      864       
                                                                 
 conv1_bn (BatchNormalizatio  (None, 112, 112, 32)     128       
 n)                                                              
                                                                 
 conv1_relu (ReLU)           (None, 112, 112, 32)      0         
                                                                 
 conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)     288       
                                                                 
 conv_dw_1_bn (BatchNormaliz  (None, 112, 112, 32)     128       
 ation)                                                          
                                                                 
 conv_dw_1_relu (ReLU)       (None, 112, 112, 32)      0         
                                                                 
 conv_pw_1 (Conv2D)          (None, 112, 112, 64)      2048      
                                                                 
...
Total params: 3,228,864
Trainable params: 3,206,976
Non-trainable params: 21,888
```

```python
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
#headModel = Flatten(name="flatten")(headModel)
headModel = Dense(1024, activation="relu")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = GlobalAveragePooling2D()(headModel)
headModel = Dense(2, activation="softmax")(headModel)

mobilenet_model = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:
    layer.trainable = False
```

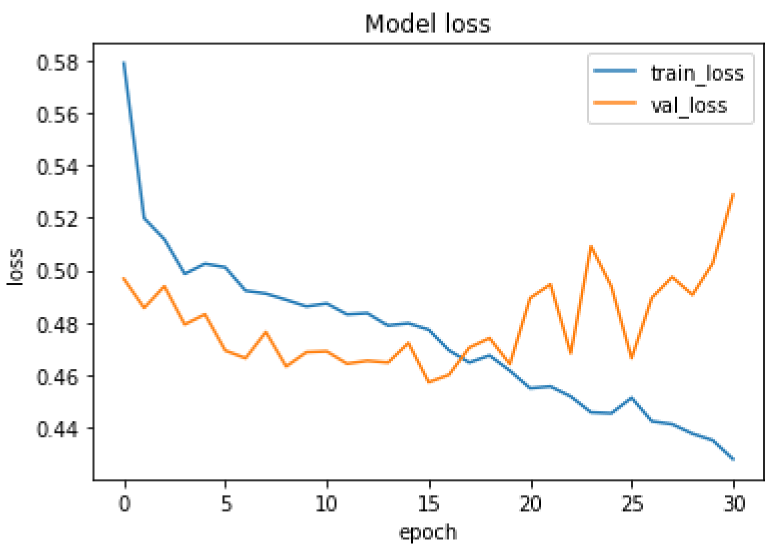
```python
mobilenet_model.compile(optimizer = 'adam', loss = 'SparseCategoricalCrossentropy', metrics= ['accuracy'])
```

```python
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
checkpointer = ModelCheckpoint(filepath="MOBILENET_weights.hdf5", verbose=1, save_best_only=True)
```

```python
print("[INFO] training head...")
mobilenet_history = mobilenet_model.fit(x=trainX,
                                        y=trainY,
                                        batch_size=8,
                                        epochs=50,
                                        validation_split=0.1,
                                        callbacks=[checkpointer, earlystopping],)
```

```
Epoch 1/50
445/450 [============================>.] - ETA: 0s - loss: 0.5798 - accuracy: 0.7919
Epoch 1: val_loss improved from inf to 0.49666, saving model to MOBILENET_weights.hdf5
450/450 [==============================] - 7s 11ms/step - loss: 0.5789 - accuracy: 0.7919 - val_loss: 0.4967 - val_accuracy: 0.8175
Epoch 2/50
448/450 [============================>.] - ETA: 0s - loss: 0.5200 - accuracy: 0.8036
Epoch 2: val_loss improved from 0.49666 to 0.48555, saving model to MOBILENET_weights.hdf5
450/450 [==============================] - 5s 10ms/step - loss: 0.5198 - accuracy: 0.8039 - val_loss: 0.4856 - val_accuracy: 0.8175
Epoch 3/50
445/450 [============================>.] - ETA: 0s - loss: 0.5121 - accuracy: 0.8014
Epoch 3: val_loss did not improve from 0.48555
450/450 [==============================] - 4s 9ms/step - loss: 0.5117 - accuracy: 0.8019 - val_loss: 0.4938 - val_accuracy: 0.8175
Epoch 4/50
447/450 [============================>.] - ETA: 0s - loss: 0.4988 - accuracy: 0.8037
Epoch 4: val_loss improved from 0.48555 to 0.47923, saving model to MOBILENET_weights.hdf5
450/450 [==============================] - 5s 10ms/step - loss: 0.4985 - accuracy: 0.8039 - val_loss: 0.4792 - val_accuracy: 0.8175
Epoch 5/50
445/450 [============================>.] - ETA: 0s - loss: 0.5030 - accuracy: 0.8034
Epoch 5: val_loss did not improve from 0.47923
450/450 [==============================] - 4s 10ms/step - loss: 0.5024 - accuracy: 0.8039 - val_loss: 0.4831 - val_accuracy: 0.8175
Epoch 6/50
446/450 [============================>.] - ETA: 0s - loss: 0.5012 - accuracy: 0.8038
Epoch 6: val_loss improved from 0.47923 to 0.46930, saving model to MOBILENET_weights.hdf5
450/450 [==============================] - 5s 10ms/step - loss: 0.5011 - accuracy: 0.8039 - val_loss: 0.4693 - val_accuracy: 0.8175
...
445/450 [============================>.] - ETA: 0s - loss: 0.4291 - accuracy: 0.8039
Epoch 31: val_loss did not improve from 0.45722
450/450 [==============================] - 4s 10ms/step - loss: 0.4279 - accuracy: 0.8050 - val_loss: 0.5287 - val_accuracy: 0.7825
Epoch 31: early stopping
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
plt.plot(mobilenet_history.history['loss'])
plt.plot(mobilenet_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()
```

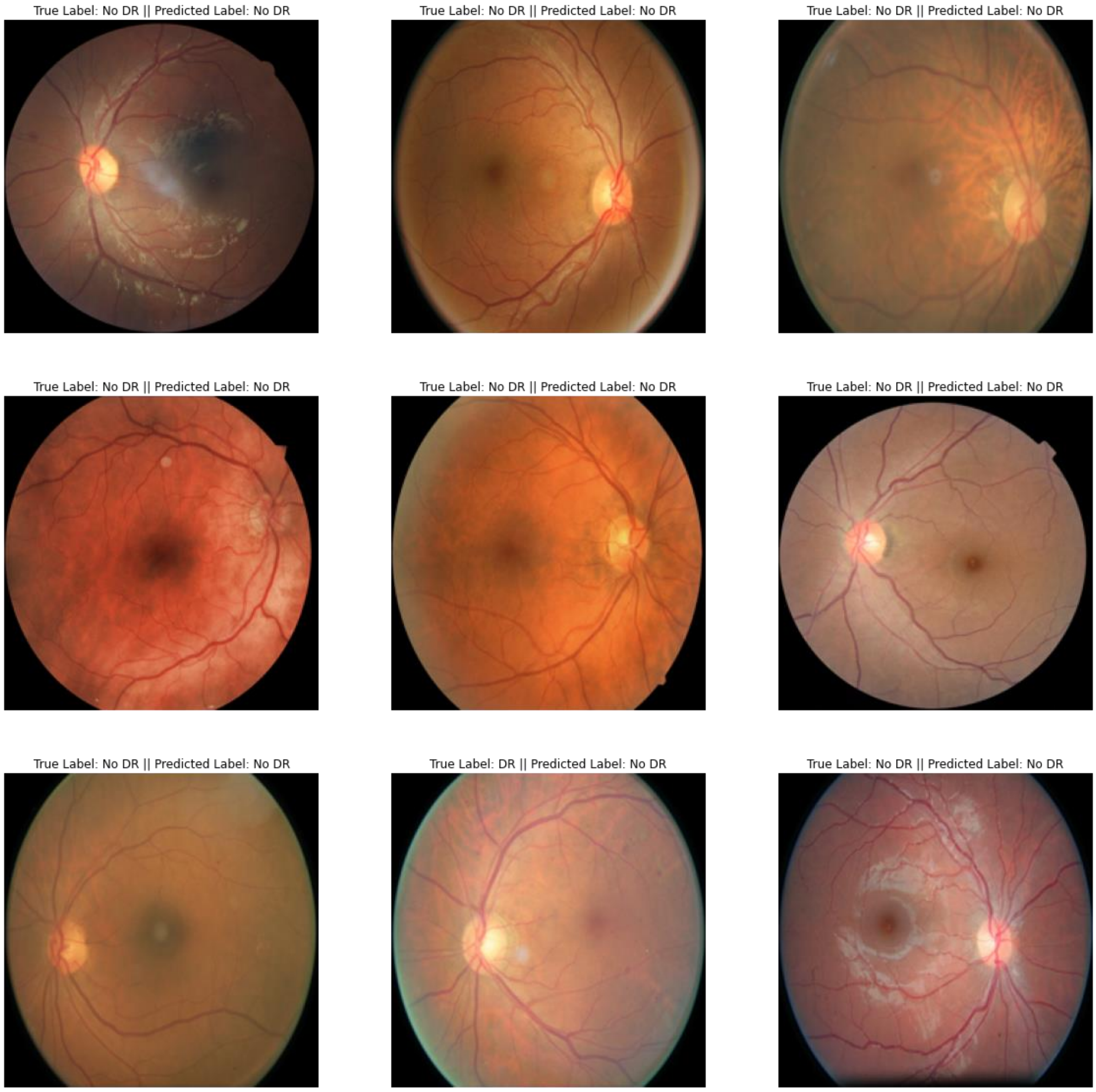


```
mobilenet_model.load_weights("MOBILENET_weights.hdf5")

evaluate = mobilenet_model.evaluate(x_test, y_test)
print('Accuracy Test : {}'.format(evaluate[1]))
```

```
prediction = []
original = y_test
image = []
count = 0

for item in x_test:
    img= item
    img = img.reshape(-1,224,224,3)
    predict = mobilenet_model.predict(img)
    predict = np.argmax(predict)
    prediction.append(predict)

# Getting the test accuracy
score = accuracy_score(original, prediction)
print("Test Accuracy : {}".format(score))
```

```
Test Accuracy : 0.806
```

```
class_names = ['No DR','DR']
import random
plt.figure(figsize = (20,20))
for i in range(9):
    random_int_index = random.choice(range(len(x_test)))
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[random_int_index])
    if prediction[random_int_index] == original[random_int_index]:
        color = "g"
    else:
        color = "r"
    plt.title("True Label: " + class_names[original[random_int_index][0]] + " || " + "Predicted Label: " +
              class_names[prediction[random_int_index]])
    plt.axis(False);
```


True Label: No DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR
True Label: DR || Predicted Label: No DR
True Label: No DR || Predicted Label: No DR

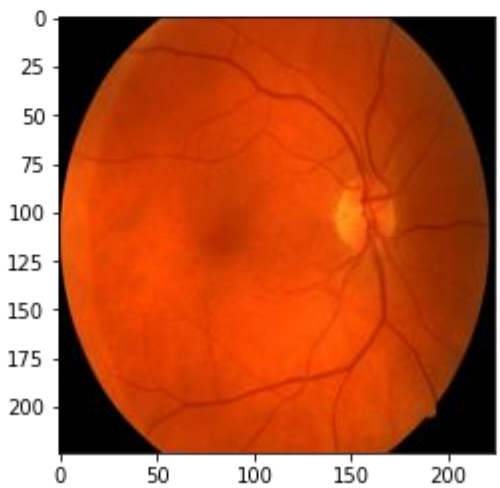
```python
# accuracy, sensitivity, and specificity
cm = confusion_matrix(np.asarray(original), np.asarray(prediction))
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[806   0]
 [194   0]]
acc: 0.8060
sensitivity: 1.0000
specificity: 0.0000
```

```python
# Print out the classification report
print(classification_report(np.asarray(original), np.asarray(prediction)))
```

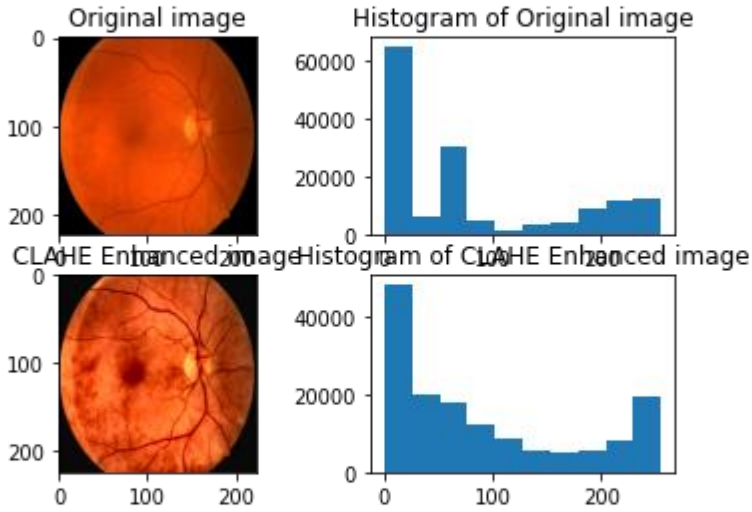
```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89       806
           1       0.00      0.00      0.00       194

    accuracy                           0.81      1000
   macro avg       0.40      0.50      0.45      1000
weighted avg       0.65      0.81      0.72      1000
```

```python
plt.imshow(features[0], cmap='bone')
```



```python
clahe = cv2.createCLAHE(clipLimit=5.0, tileGridSize=(8,8))
def clahe_enhancer(img, demo=False):
    img_lab = cv2.cvtColor(img, cv2.COLOR_RGB2Lab)

    #0 to 'L' channel, 1 to 'a' channel, and 2 to 'b' channel
    img_lab[:,:,0] = clahe.apply(img_lab[:,:,0])
    clahe_img = cv2.cvtColor(img_lab, cv2.COLOR_Lab2RGB)

    if demo:
        img_flattened = img.flatten()
        clahe_img_flattened = clahe_img.flatten()
        fig = plt.figure()
        rcParams['figure.figsize'] = 10,10

        plt.subplot(2, 2, 1)
        plt.imshow(img, cmap='bone')
        plt.title("Original image")

        plt.subplot(2, 2, 2)
        plt.hist(img_flattened)
        plt.title("Histogram of Original image")

        plt.subplot(2, 2, 3)
        plt.imshow(clahe_img, cmap='bone')
        plt.title("CLAHE Enhanced image")

        plt.subplot(2, 2, 4)
        plt.hist(clahe_img_flattened)
        plt.title("Histogram of CLAHE Enhanced image")

    return clahe_img
```

```python
clahe_img = clahe_enhancer(trainX[0], True)
print(clahe_img.shape)
```




```python
clahe_features = []
for img in trainX:
    clahe_img = clahe_enhancer(img, False)
    clahe_features.append(clahe_img)
clahe_features = np.array(clahe_features)
clahe_features.shape
```

```
(4000, 224, 224, 3)
```

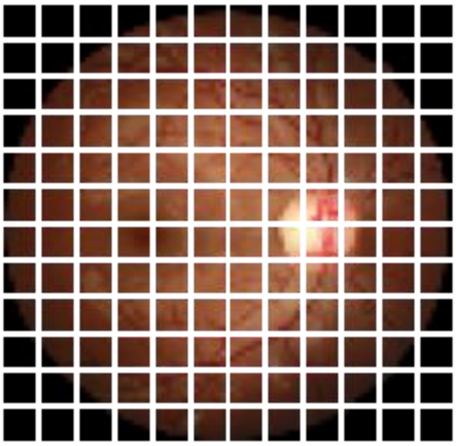
```python
num_classes = 2
input_shape = (224, 224, 3)
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 20
num_epochs = 100
image_size = 72                             # We'll resize input images to this size
patch_size = 6                              # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [projection_dim * 2,
                     projection_dim,]       # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]               # Size of the dense layers of the final classifier
```

```python
data_augmentation = Sequential([tf.keras.layers.experimental.preprocessing.Normalization(),
                                tf.keras.layers.experimental.preprocessing.Resizing(image_size, image_size),],
                               name="data_augmentation",)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(trainX)
```

```python
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

```python
from tensorflow.keras import layers
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(images=images,
                                           sizes=[1, self.patch_size, self.patch_size, 1],
                                           strides=[1, self.patch_size, self.patch_size, 1],
                                           rates=[1, 1, 1, 1],
                                           padding="VALID",)
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches
```

```python
plt.figure(figsize=(4, 4))
image = trainX[np.random.choice(range(trainX.shape[0]))]
plt.imshow(image.astype("uint8"))
plt.axis("off")

resized_image = tf.image.resize(tf.convert_to_tensor([image]), size=(image_size, image_size))
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")
```



```python
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(input_dim=num_patches, output_dim=projection_dim)

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

Vit

```python
def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    augmented = data_augmentation(inputs)
    patches = Patches(patch_size)(augmented)
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        attention_output = layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim, dropout=0.1)(x1, x1)
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)

    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    logits = layers.Dense(num_classes)(features)
    model = tf.keras.Model(inputs=inputs, outputs=logits)
    return model
```

```python
vit_classifier = create_vit_classifier()
```

```python
INIT_LR = 1e-3
EPOCHS = 25
BS = 8

optimizer = tfa.optimizers.AdamW(learning_rate=learning_rate, weight_decay=weight_decay)
# optimizer = tf.keras.optimizers.Adam(lr=learning_rate, decay=weight_decay)
# opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)

vit_classifier.compile(optimizer=optimizer,
                       loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                       metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")])
```

```
es = EarlyStopping(monitor='val_loss',
                   mode='min',
                   patience=3,
                   restore_best_weights=True,
                   verbose=1)

reduce_lr = ReduceLROnPlateau(monitor = 'val_loss',
                              factor = 0.2,
                              patience = 3,
                              min_lr = 1e-5,
                              mode = 'min',
                              verbose=1)

metric = 'val_loss'
checkpoint_callback = ModelCheckpoint("VIT_weights.hdf5",
                                      monitor=metric,
                                      save_best_only=True,
                                      save_weights_only=True,
                                      verbose = 2,
                                      mode='max')
```

```
vit_history = vit_classifier.fit(x=trainX,
                                 y=trainY,
                                 batch_size=batch_size,
                                 epochs=50,
                                 validation_split=0.1,
                                 callbacks=[checkpoint_callback])
```

```
Epoch 1/50
179/180 [============================>.] - ETA: 0s - loss: 1.8844 - accuracy: 0.7128
Epoch 1: val_loss improved from -inf to 0.48210, saving model to VIT_weights.hdf5
180/180 [==============================] - 18s 47ms/step - loss: 1.8759 - accuracy: 0.7136 - val_loss: 0.4821 - val_accuracy: 0.8175
Epoch 2/50
179/180 [============================>.] - ETA: 0s - loss: 0.5534 - accuracy: 0.7860
Epoch 2: val_loss improved from 0.48210 to 0.49276, saving model to VIT_weights.hdf5
180/180 [==============================] - 7s 41ms/step - loss: 0.5531 - accuracy: 0.7861 - val_loss: 0.4928 - val_accuracy: 0.8175
Epoch 3/50
179/180 [============================>.] - ETA: 0s - loss: 0.5375 - accuracy: 0.8039
Epoch 3: val_loss did not improve from 0.49276
180/180 [==============================] - 7s 39ms/step - loss: 0.5381 - accuracy: 0.8036 - val_loss: 0.4784 - val_accuracy: 0.8175
Epoch 4/50
179/180 [============================>.] - ETA: 0s - loss: 0.5268 - accuracy: 0.8000
Epoch 4: val_loss improved from 0.49276 to 0.51943, saving model to VIT_weights.hdf5
180/180 [==============================] - 7s 41ms/step - loss: 0.5269 - accuracy: 0.8000 - val_loss: 0.5194 - val_accuracy: 0.8175
Epoch 5/50
179/180 [============================>.] - ETA: 0s - loss: 0.5218 - accuracy: 0.8008
Epoch 5: val_loss did not improve from 0.51943
180/180 [==============================] - 7s 39ms/step - loss: 0.5222 - accuracy: 0.8008 - val_loss: 0.4829 - val_accuracy: 0.8175
Epoch 6/50
179/180 [============================>.] - ETA: 0s - loss: 0.5254 - accuracy: 0.8014
Epoch 6: val_loss did not improve from 0.51943
180/180 [==============================] - 7s 39ms/step - loss: 0.5255 - accuracy: 0.8011 - val_loss: 0.4867 - val_accuracy: 0.8175
Epoch 7/50
...
Epoch 50/50
179/180 [============================>.] - ETA: 0s - loss: 0.4883 - accuracy: 0.8028
Epoch 50: val_loss did not improve from 0.51943
180/180 [==============================] - 7s 39ms/step - loss: 0.4884 - accuracy: 0.8031 - val_loss: 0.4591 - val_accuracy: 0.8175
```

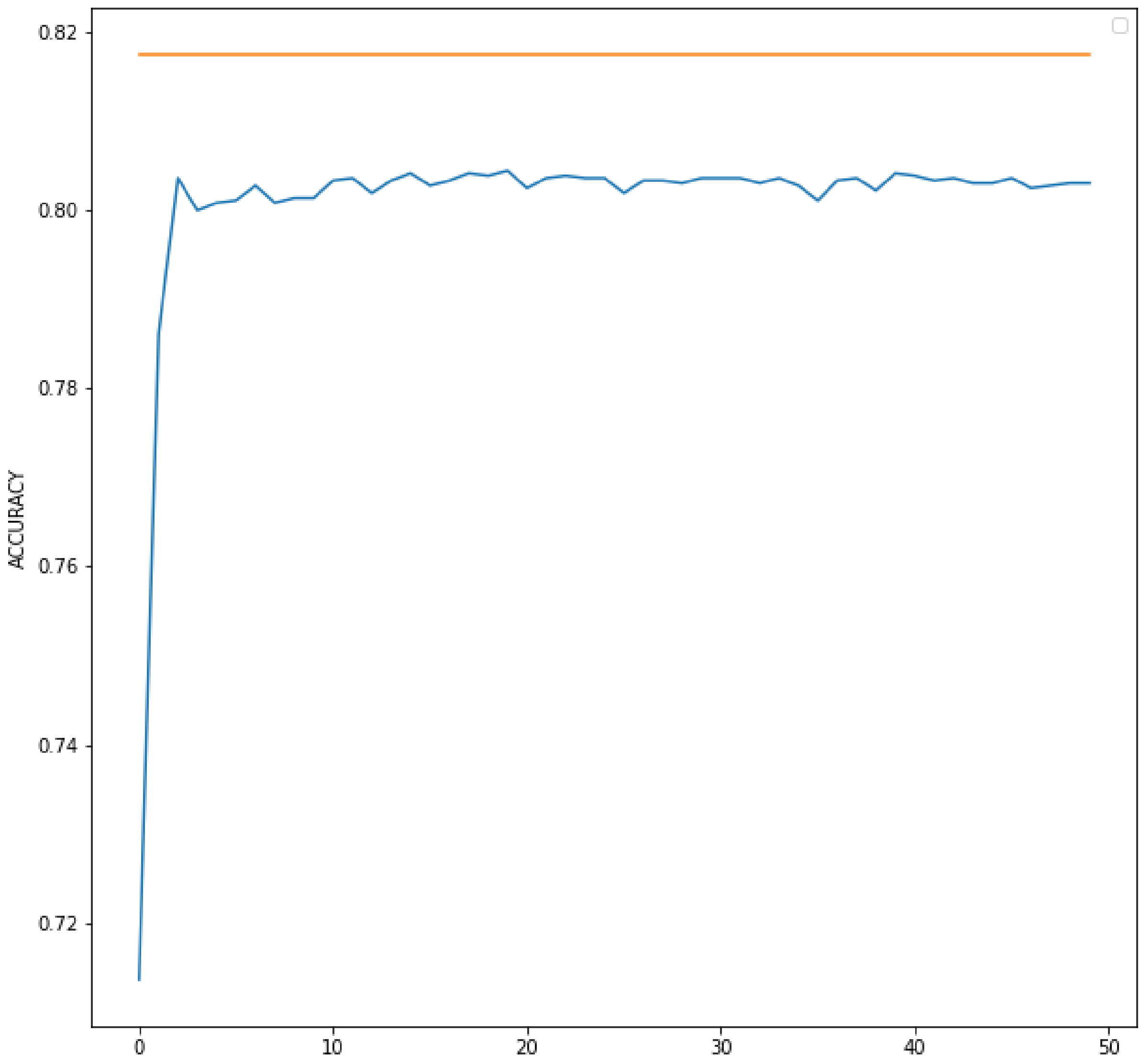
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
vit_classifier.load_weights("VIT_weights.hdf5")
_, accuracy = vit_classifier.evaluate(x_test, y_test)
print(f"Test accuracy: {round(accuracy * 100, 2)}%")
```

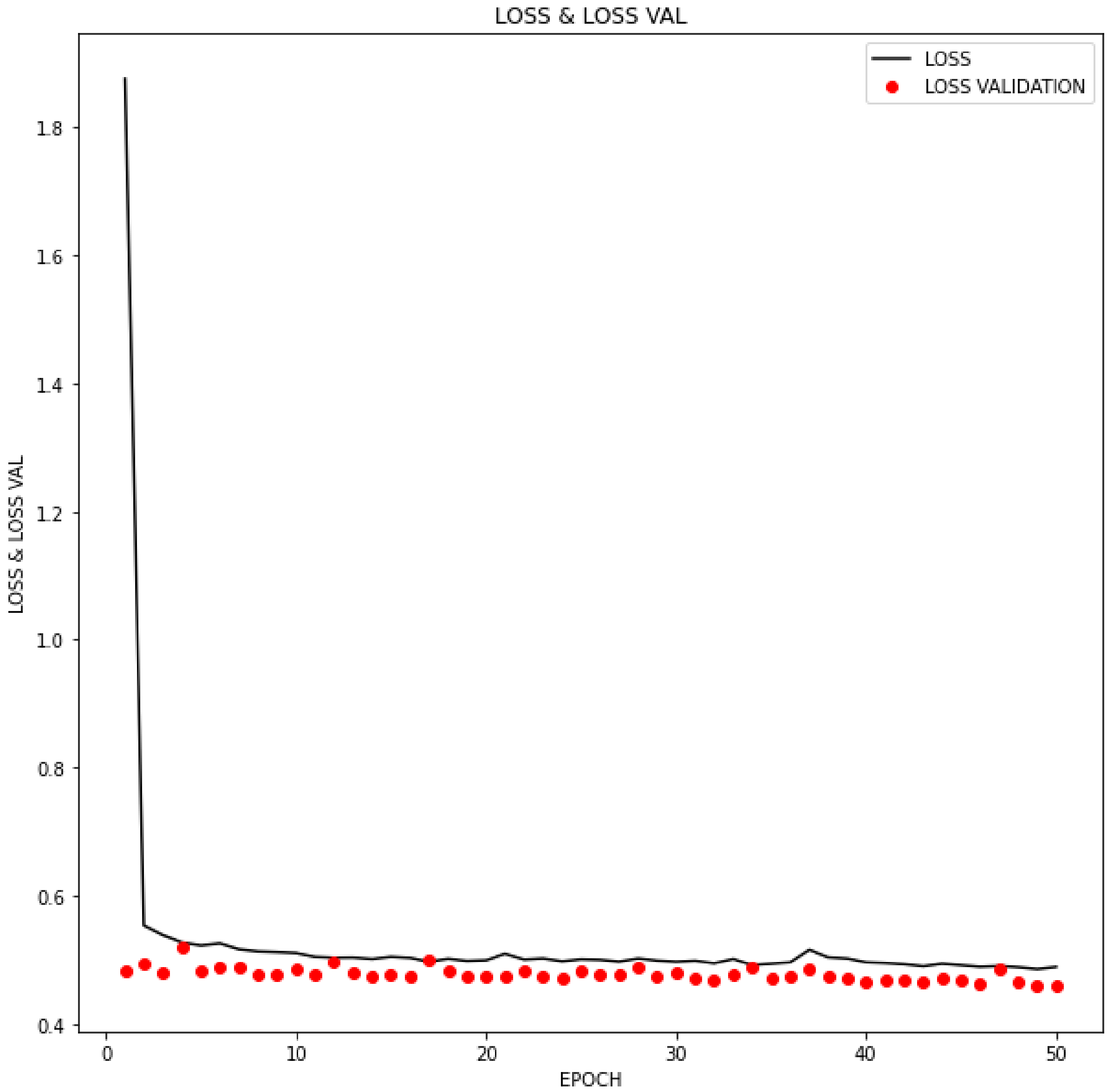
```
32/32 [==============================] - 2s 26ms/step - loss: 0.5264 - accuracy: 0.8060
Test accuracy: 80.6%
```

```
HistoryDict = vit_history.history

val_losses = HistoryDict["val_loss"]
val_acc = HistoryDict["val_accuracy"]
acc = HistoryDict["accuracy"]
losses = HistoryDict["loss"]
epochs = range(1,len(val_losses)+1)
```

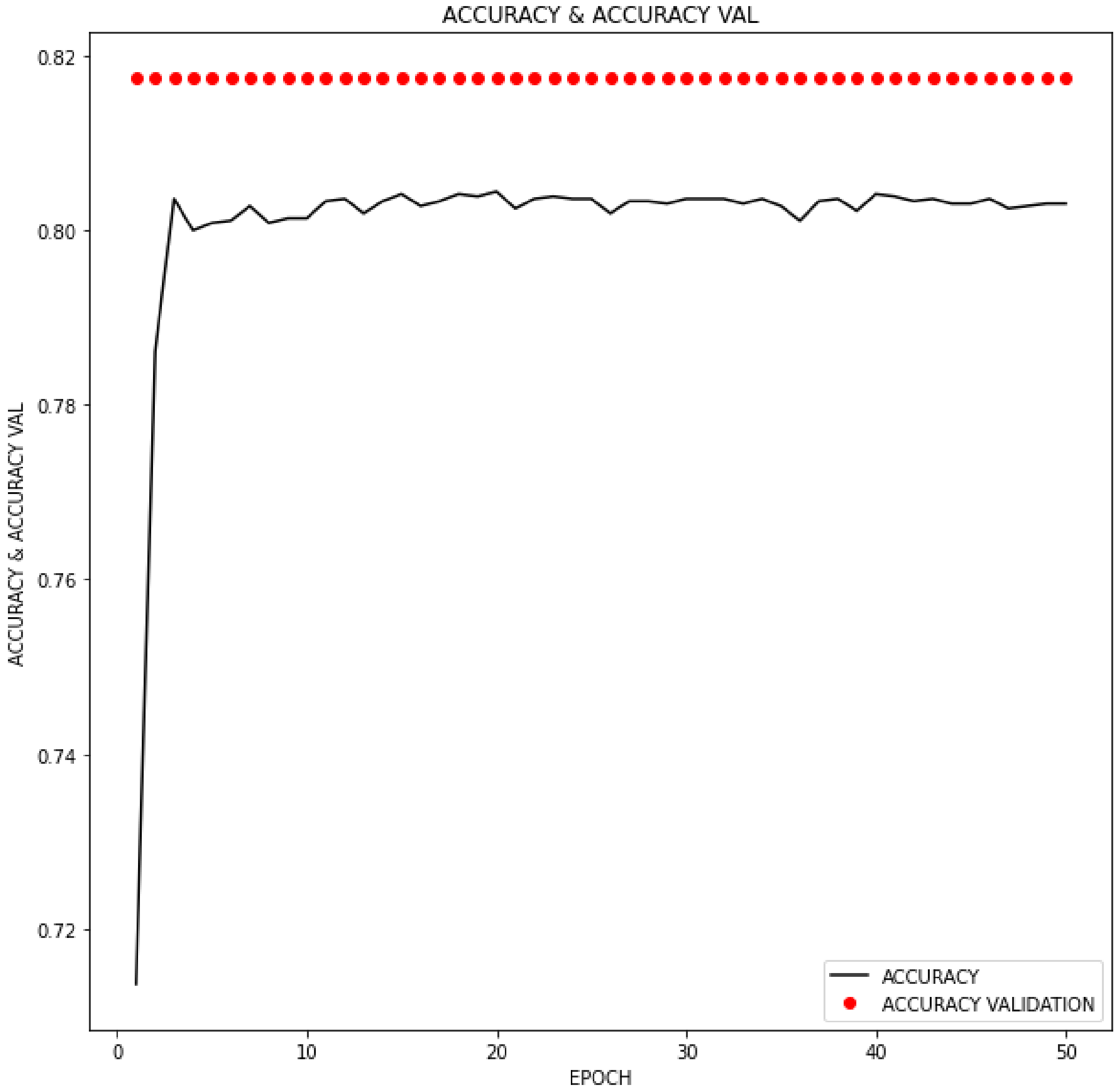
```
plt.plot(vit_history.history["accuracy"])
plt.plot(vit_history.history["val_accuracy"])
plt.ylabel("ACCURACY")
plt.legend()
plt.show()
```



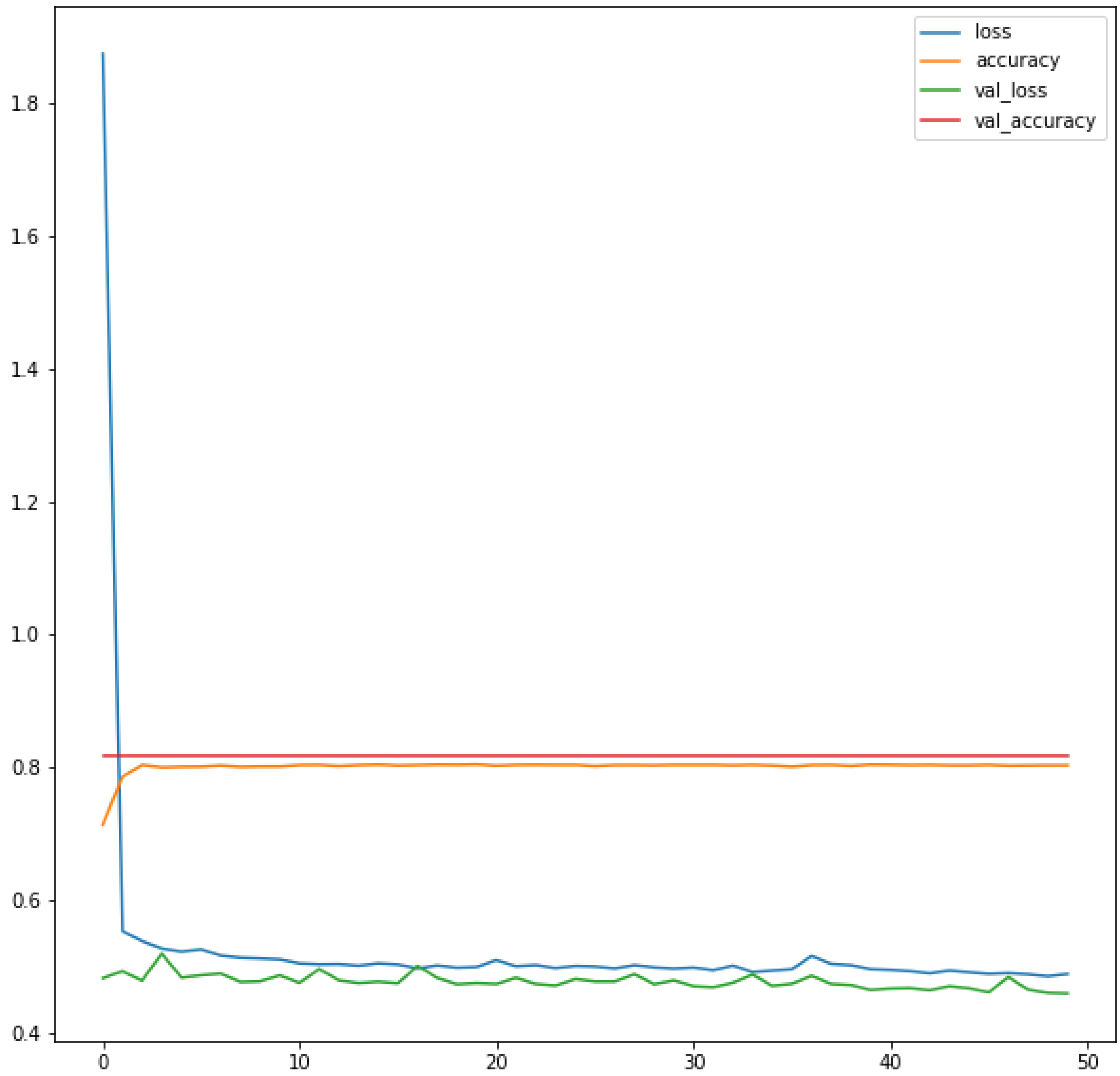
```python
plt.plot(epochs,losses,"k-",label="LOSS")
plt.plot(epochs,val_losses,"ro",label="LOSS VALIDATION")
plt.title("LOSS & LOSS VAL")
plt.xlabel("EPOCH")
plt.ylabel("LOSS & LOSS VAL")
plt.legend()
plt.show()
```



```
plt.plot(epochs,acc,"k-",label="ACCURACY")
plt.plot(epochs,val_acc,"ro",label="ACCURACY VALIDATION")
plt.title("ACCURACY & ACCURACY VAL")
plt.xlabel("EPOCH")
plt.ylabel("ACCURACY & ACCURACY VAL")
plt.legend()
plt.show()
```




```
vit_Summary = pd.DataFrame(vit_history.history)
vit_Summary.plot()
```



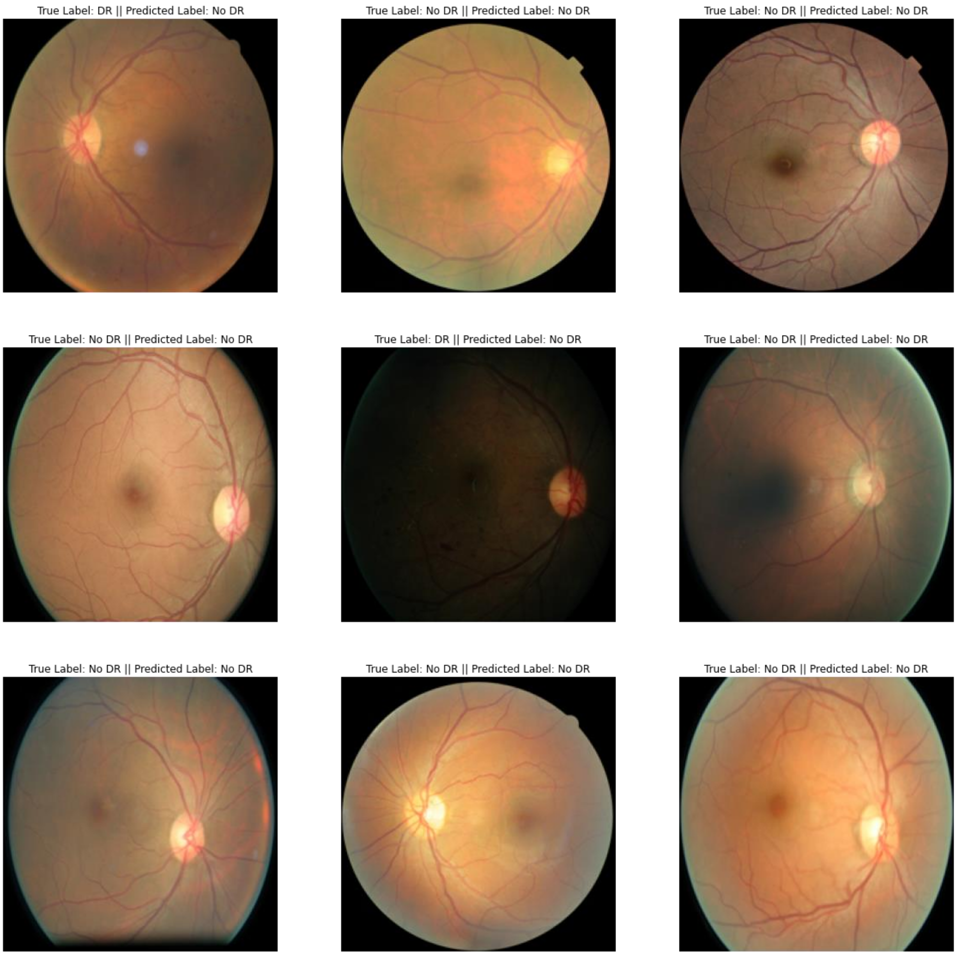

```python
vit_classifier.load_weights("VIT_weights.hdf5")

evaluate = vit_classifier.evaluate(x_test, y_test)
print('Accuracy Test : {}'.format(evaluate[1]))
```

```
32/32 [==============================] - 1s 26ms/step - loss: 0.5264 - accuracy: 0.8060
Accuracy Test : 0.8059999942779541
```

```python
# import cv2
prediction = []
original = y_test
image = []
count = 0

for item in x_test:
    img= item
    img = img.reshape(-1,224,224,3)
    predict = vit_classifier.predict(img)
    predict = np.argmax(predict)
    prediction.append(predict)

# Getting the test accuracy
score = accuracy_score(original, prediction)
print("Test Accuracy : {}".format(score))
```

```
Test Accuracy : 0.806
```

```python
class_names = ['No DR','DR']
import random
plt.figure(figsize = (20,20))
for i in range(9):
    random_int_index = random.choice(range(len(x_test)))
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[random_int_index])
    if prediction[random_int_index] == original[random_int_index]:
        color = "g"
    else:
        color = "r"
    plt.title("True Label: " + class_names[original[random_int_index][0]] + " || " + "Predicted Label: " +
              class_names[prediction[random_int_index]])
    plt.axis(False);
```




```
# accuracy, sensitivity, and specificity
cm = confusion_matrix(np.asarray(original), np.asarray(prediction))
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[806   0]
 [194   0]]
acc: 0.8060
sensitivity: 1.0000
specificity: 0.0000
```

```
# Print out the classification report
print(classification_report(np.asarray(original), np.asarray(prediction)))
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89       806
           1       0.00      0.00      0.00       194

    accuracy                           0.81      1000
   macro avg       0.40      0.50      0.45      1000
weighted avg       0.65      0.81      0.72      1000
```

References

Abreu, A., Ferreira, D., Neto, C., Abelha, A. and Machado, J., 2021, February. Diagnosis of Diabetic Retinopathy Using Data Mining Classification Techniques. In International Conference on Advances in Digital Science (pp. 198-209). Springer, Cham.

Alabdulwahhab, K.M., Sami, W., Mehmood, T., Meo, S.A., Alasbali, T.A. and Alwadani, F.A., 2021. Automated detection of diabetic retinopathy using machine learning classifiers. European Review for Medical and Pharmacological Sciences, 25(2), pp.583-590.

Atwany, M.Z., Sahyoun, A.H. and Yaqub, M., 2022. Deep Learning Techniques For Diabetic Retinopathy Classification: A Survey. IEEE Access.

Bora, A., Balasubramanian, S., Babenko, B., Virmani, S., Venugopalan, S., Mitani, A., de Oliveira Marinho, G., Cuadros, J., Ruamviboonsuk, P., Corrado, G.S. and Peng, L., 2021. Predicting the risk of developing diabetic retinopathy using deep learning. The Lancet Digital Health, 3(1), pp.e10-e19.

Costa, P., Araújo, T., Aresta, G., Galdran, A., Mendonça, A.M., Smailagic, A. and Campilho, A., 2019, May. EyeWes: weakly supervised pre-trained convolutional neural networks for diabetic retinopathy detection. In 2019 16th international conference on machine vision applications (MVA) (pp. 1-6). IEEE.

Farooq, M.S., Arooj, A., Alroobaea, R., Baqasah, A.M., Jabarulla, M.Y., Singh, D. and Sardar, R., 2022. Untangling computer-aided diagnostic system for screening diabetic retinopathy based on deep learning techniques. Sensors, 22(5), p.1803.

Filos, A., Farquhar, S., Gomez, A.N., Rudner, T.G., Kenton, Z., Smith, L., Alizadeh, M., De Kroon, A. and Gal, Y., 2019. A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks. arXiv preprint arXiv:1912.10481.

Imran, M., Ullah, A., Arif, M. and Noor, R., 2022. A unified technique for entropy enhancement based diabetic retinopathy detection using hybrid neural network. Computers in Biology and Medicine, p.105424.

Kassani, S.H., Kassani, P.H., Khazaeinezhad, R., Wesolowski, M.J., Schneider, K.A. and Deters, R., 2019, December. Diabetic retinopathy classification using a modified xception architecture. In 2019 IEEE international symposium on signal processing and information technology (ISSPIT) (pp. 1-6). IEEE.

Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S. and Shah, M., 2021. Transformers in vision: A survey. ACM Computing Surveys (CSUR).

Masud, M., Alhamid, M.F. and Zhang, Y., 2022. A Convolutional Neural Network Model Using Weighted Loss Function to Detect Diabetic Retinopathy. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 18(1s), pp.1-16.

Mohanty, K.K., Barik, P.K., Barik, R.C. and Bhuyan, K.C., 2019, December. An efficient prediction of diabetic from retinopathy using machine learning and signal processing approach. In 2019 International Conference on Information Technology (ICIT) (pp. 103-108). IEEE.