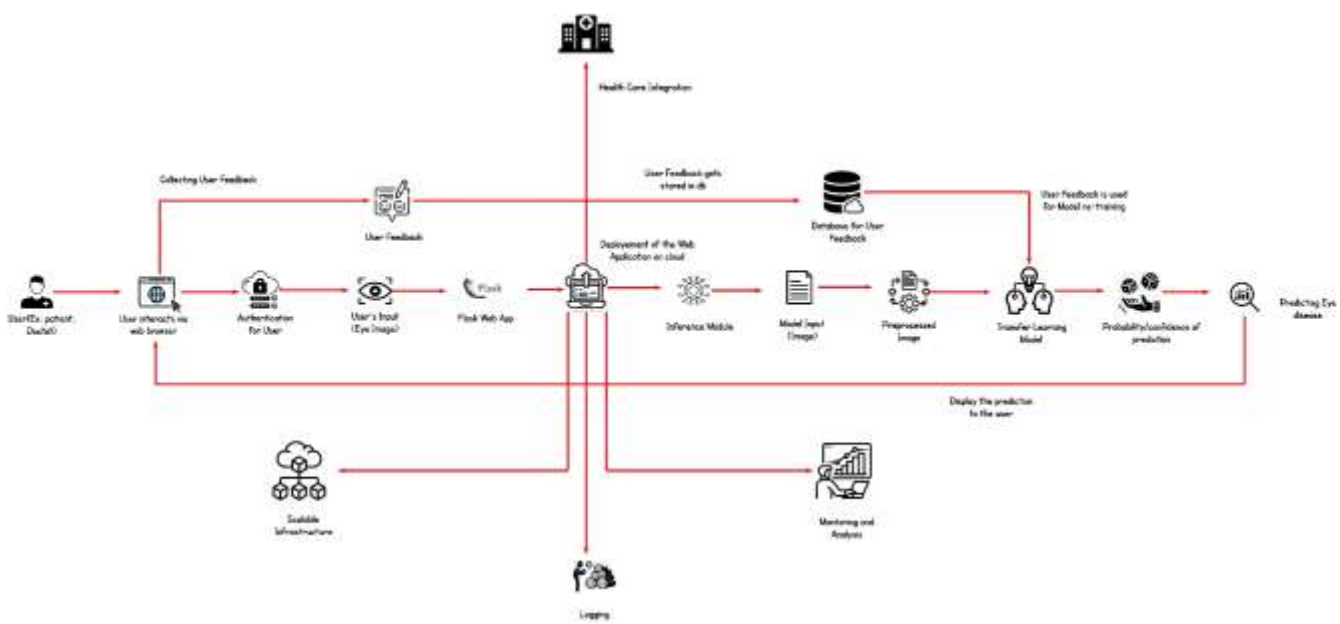
Eye Disease Detection Using Deep Learning

Project Description:

In this project we are classifying various types of Eye Diseases that people get due to various reasons like age, diabetes, etc. These diseases are majorly classified into 4 categories namely Normal, cataract, Diabetic Retinopathy & Glaucoma. Deep-learning (DL) methods in artificial intelligence (AI) play a dominant role as high-performance classifiers in the detection of the Eye Diseases using images.

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in image analysis and classification. We used Transfer Learning techniques like Inception V3, VGG19, Xception V3 that are more widely used as a transfer learning method in image analysis and they are highly effective.

Technical Architecture:



Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- The VGG19 Model analyzes the image, then the prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

o Data Collection.
  o Create a Train, validation and Test path.

o Image Pre-processing.
  o Import the required library
  o Configure ImageDataGenerator class
  o Apply ImageDataGenerator functionality to Trainset and Testset.

o Model Building
  o Pre-trained CNN model as a Feature Extractor
  o Adding Dense Layer
  o Configure the Learning Process
  o Train the model
  o Save the Model
  o Test the model

o Application Building
  o Create an HTML file

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- Deep Learning Concepts
  o CNN: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add

o VGG19: VGG-19 convolutional neural network - MATLAB vgg19 - MathWorks India

o ResNet-50V2: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

o Inception-V3: https://iq.opengenus.org/inception-v3-model-architecture/

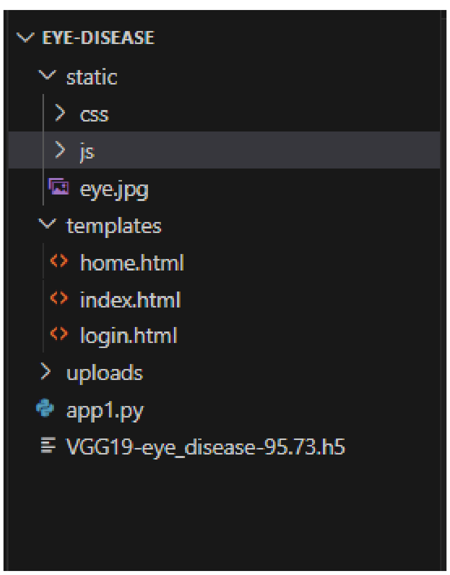
o Xception: https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

- Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

  Link: https://www.youtube.com/watch?v=lj4I_CvBnt0

Build Python Code:

Project Structure:



For building a Flask Application we needs HTML pages stored in the templates folder,CSS for styling the pages stored in the static folder and a python script app1.py for server side scripting

Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Activity 1: Download the dataset

Collect images of Eye Diseases then organize into subdirectories based on their respective names as shown in the project structure. Create folders of types of Eye Diseases that need to be recognized.

In this project, we have collected images of 4 types of Eye Diseases images like Normal,cataract, Diabetic Retinopathy & Glaucoma and they are saved in the respective sub directories with their respective names.

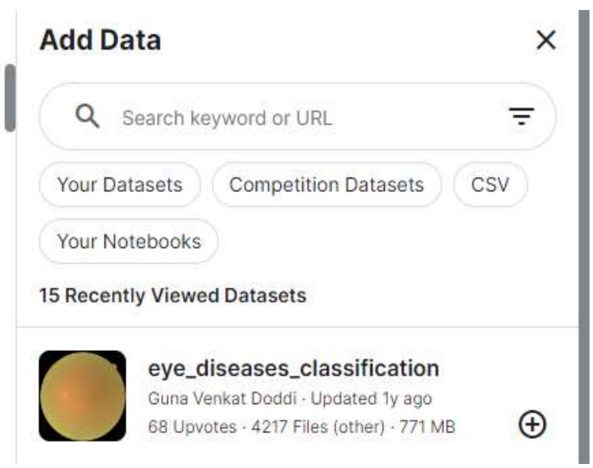
You can download the dataset used in this project using the below link

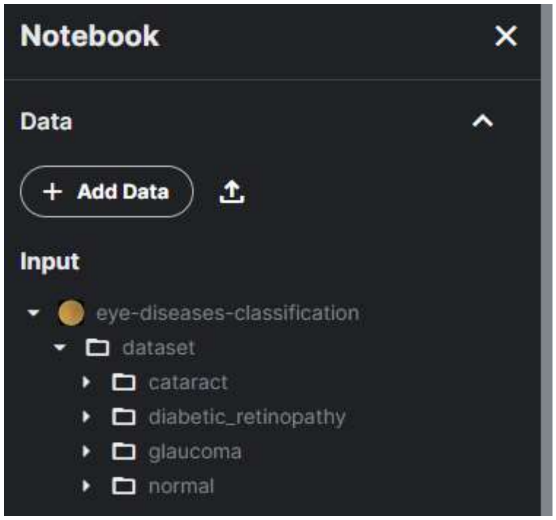
Dataset:- https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification

Note: For better accuracy train on more images

We are going to build our training model on Kaggle notebook.

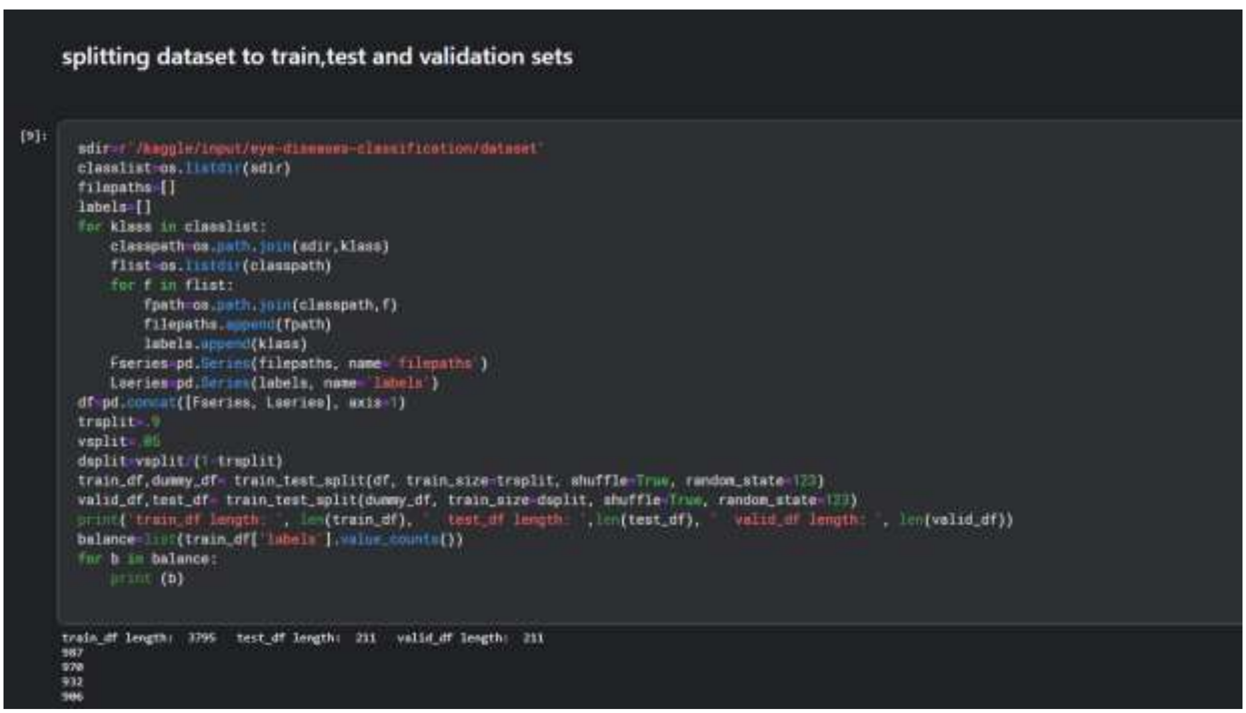
We can simply add the Eye disease classification dataset to the kaggle notebook.





Activity 2: Create training,validating and testing dataset

To build a DL model we have to split training and testing data into two separate folders. But in this project dataset folder training,validating and testing folders are not present. So, in this case we have to separate the data into train,validate & test folders.



splitting dataset to train,test and validation sets

```
[9]:
sdir=r'/kaggle/input/eye-diseases-classification/dataset'
classlist=os.listdir(sdir)
filepaths=[]
labels=[]
for klass in classlist:
    classpath=os.path.join(sdir,klass)
    flist=os.listdir(classpath)
    for f in flist:
        fpath=os.path.join(classpath,f)
        filepaths.append(fpath)
        labels.append(klass)
    Fseries=pd.Series(filepaths, name='filepaths')
    Lseries=pd.Series(labels, name='labels')
df=pd.concat([Fseries, Lseries], axis=1)
trsplit=.9
vsplit=.05
dsplit=vsplit/(1-trsplit)
train_df,dummy_df= train_test_split(df, train_size=trsplit, shuffle=True, random_state=123)
valid_df,test_df= train_test_split(dummy_df, train_size=dsplit, shuffle=True, random_state=123)
print('train_df length: ', len(train_df), '  test_df length: ',len(test_df), '  valid_df length: ', len(valid_df))
balance=list(train_df['labels'].value_counts())
for b in balance:
    print (b)
```

```
train_df length:  3795   test_df length:  211   valid_df length:  211
987
970
932
906
```

```python
sample_list=[]
max_size= 906
min_size = 0
groups=train_df.groupby('labels')
for label in train_df['labels'].unique():
    group=groups.get_group(label)
    sample_count=len(group)
    if sample_count> max_size :
        samples=group.sample(max_size, replace=False, weights=None, random_state=123, axis=0).reset_index(drop=True)
        sample_list.append(samples)
    elif sample_count>= min_size:
        sample_list.append(group)
train_df=pd.concat(sample_list, axis=0).reset_index(drop=True)
balance=list(train_df['labels'].value_counts())
print (balance)
```

```
[906, 906, 906, 906]
```

Transfer learning model used in our project is VGG-19.The image input size of VGG-19 model is (224, 224).

```python
height=224
width=224
channels=3
batch_size=40
img_shape=(height, width, channels)
```

Milestone 2: Image Preprocessing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Importing the libraries

Import the necessary libraries as shown in the image

```
[1]:
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPooling2D,BatchNormalization, Flatten
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, load_model, Sequential
import numpy as np
import pandas as pd
import shutil
import time
import cv2 as cv2
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import os
import seaborn as sns
# sns.set_style('darkgrid')
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
from IPython.core.display import display, HTML
# stop annoying tensorflow warning messages
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
```

Activity 2: Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- The image flips via the horizontal_flip and vertical_flip arguments.
- Image rotations via the rotation_range argument
- Image brightness via the brightness_range argument.
- Image zoom via the zoom_range argument.

```
height=224
width=224
channels=3
batch_size=40
img_shape=(height, width, channels)
img_size=(height, width)
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps)
def scalar(img):
    return img  # EfficientNet expects pixelsin range 0 to 255 so no scaling is required
trgen=ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=True)
tvgen=ImageDataGenerator(preprocessing_function=scalar)
```

An instance of the ImageDataGenerator class can be constructed for train and test.

Activity 3: Apply ImageDataGenerator functionality to Train set and Test set

Let us apply ImageDataGenerator functionality to the Train set and Test set by using the following code. For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories Arguments:

- directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data which is 64.
- target_size: Size to resize images after they are read from disk.
- Class_mode:
  - ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).
  - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
  - ’binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
  - None (no labels).

```
trgen=ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=True)
tvgen=ImageDataGenerator(preprocessing_function=scalar)
train_gen=trgen.flow_from_dataframe( train_df, x_col='filepaths', y_col='labels', target_size=img_size, class_mode='categorical',
                                    color_mode='rgb', shuffle=True, batch_size=batch_size)
test_gen=tvgen.flow_from_dataframe( test_df, x_col='filepaths', y_col='labels', target_size=img_size, class_mode='categorical',
                                    color_mode='rgb', shuffle=False, batch_size=test_batch_size)
valid_gen=tvgen.flow_from_dataframe( valid_df, x_col='filepaths', y_col='labels', target_size=img_size, class_mode='categorical',
                                    color_mode='rgb', shuffle=True, batch_size=batch_size)
classes=list(train_gen.class_indices.keys())
class_count=len(classes)
train_steps=np.ceil(len(train_gen.labels)/batch_size)
```

```
test batch size:  1   test steps:  211
Found 3624 validated image filenames belonging to 4 classes.
Found 211 validated image filenames belonging to 4 classes.
Found 211 validated image filenames belonging to 4 classes.
```

Total the dataset is having 3624 train images,211 validation images and 211test images divided under 4 classes.

Milestone 3: Model Building

Now it's time to build our model. Let’s use the pre-trained model which is VGG19, one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification.

Deep understanding on the VGG19 model – Link is referred to in the prior knowledge section. Kindly refer to it before starting the model building part.

Activity 1: Pre-trained CNN model as a Feature Extractor

For one of the models, we will use it as a simple feature extractor by freezing all the five convolution blocks to make sure their weights don't get updated after each epoch as we train our own model. Here, we have considered images of dimension (224,224,3).
Also, we have assigned include_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our image classification (since it is not the part of Imagenet dataset)

Flatten layer flattens the input. Does not affect the batch size.

```
from keras.applications import VGG19

model_name='VGG19'
base_model=tf.keras.applications.VGG16(include_top=False, weights="imagenet",input_shape=img_shape, pooling='max')
x=base_model.output
x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
```

Activity 2: Adding Dense Layers

```
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.45, seed=123)(x)
output=Dense(class_count, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(optimizer=Adamax(learning_rate=.0001), loss='categorical_crossentropy', metrics=['accuracy'])

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. Let us create a model object named model with inputs as VGG19.input and output as dense layer.

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities. Understanding the model is a very important phase to properly use it for training and prediction purposes.

Keras provides a simple method, summary to get the full information about the model and its layers.

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (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

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 global_max_pooling2d (Glob  (None, 512)               0
 alMaxPooling2D)
```

```
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 global_max_pooling2d (Glob  (None, 512)               0
 alMaxPooling2D)

 batch_normalization (Batch  (None, 512)               2048
 Normalization)

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 4)                 1028

=================================================================
Total params: 14849092 (56.64 MB)
Trainable params: 14848068 (56.64 MB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
```

+ Code + Markdown

Activity 3: Configure the Learning Process

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process. Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

```
model.compile(optimizer=Adamax(learning_rate=.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Activity 4: Train the model

Now, let us train our model with our image dataset. The model is trained for 50 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch and probably there is further scope to improve the model.

.fit functions used to train a deep learning neural network

Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

- Callbacks:This custom callback, named LRA (short for Learning Rate Adjuster), is designed to be used during the training process of a Keras model. It dynamically adjusts the learning rate based on the training and validation performance of the model.

  To use this callback in the fit function of your Keras model, you would instantiate an object of this class and pass it as a callback to the cals parameter in the fit method.

Custom defined callback for Learning Rate Adjuster.

```python
class LRA(keras.callbacks.Callback):
    def __init__(self,model, base_model, patience,stop_patience, threshold, factor, dwell, batches, initial_epoch,epochs, ask_epoch):
        super(LRA, self).__init__()
        self.model=model
        self.base_model=base_model
        self.patience=patience # specifies how many epochs without improvement before learning rate is adjusted
        self.stop_patience=stop_patience # specifies how many times to adjust lr without improvement to stop training
        self.threshold=threshold # specifies training accuracy threshold when lr will be adjusted based on validation loss
        self.factor=factor # factor by which to reduce the learning rate
        self.dwell=dwell
        self.batches=batches # number of training batch to runn per epoch
        self.initial_epoch=initial_epoch
        self.epochs=epochs
        self.ask_epoch=ask_epoch
        self.ask_epoch_initial=ask_epoch # save this value to restore if restarting training
        # callback variables
        self.count=0 # how many times lr has been reduced without improvement
        self.stop_count=0
        self.best_epoch=1   # epoch with the lowest loss
        self.initial_lr=float(tf.keras.backend.get_value(model.optimizer.lr)) # get the initiallearning rate and save it
        self.highest_tracc=0.0 # set highest training accuracy to # initially
        self.lowest_vloss=np.inf # set lowest validation loss to infinity initially
        self.best_weights=self.model.get_weights() # set best weights to model's initial weights
        self.initial_weights=self.model.get_weights()   # save initial weights if they have to get restored
```

```python
    def on_train_begin(self, logs=None):
        if self.base_model != None:
            status=base_model.trainable
            if status:
                msg=' initializing callback starting train with base_model trainable'
            else:
                msg='initializing callback starting training with base_model not trainable'
        else:
            msg='initialing callback and starting training'
        print_in_color (msg, (244, 252, 3), (55,65,80))
        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:^8s}'.format('Epoch', 'Loss', 'Accuracy',
                                                                                        'V_loss','V_acc', 'LR', 'Next LR', 'Monitor', 'Duration')
        print_in_color(msg, (244,252,3), (55,65,80))
        self.start_time= time.time()

    def on_train_end(self, logs=None):
        stop_time=time.time()
        tr_duration= stop_time- self.start_time
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))

        self.model.set_weights(self.best_weights) # set the weights of the model to the best weights
        msg=f'Training is completed - model is set with weights from epoch {self.best_epoch} '
        print_in_color(msg, (0,255,0), (55,65,80))
        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f} minutes, {seconds:4.2f} seconds)'
        print_in_color(msg, (0,255,0), (55,65,80))
```

```python
def on_train_batch_end(self, batch, logs=None):
    acc=logs.get('accuracy')* 100  # get training accuracy
    loss=logs.get('loss')
    msg='{0:20s}processing batch {1:4s} of {2:5s} accuracy= {3:8.3f}  loss: {4:8.5f}'.format(' ', str(batch), str(self.batches), acc, loss)
    print(msg, '\r', end='') # prints over on the same line to show running batch count

def on_epoch_begin(self,epoch, logs=None):
    self.now= time.time()

def on_epoch_end(self, epoch, logs=None):  # method runs on the end of each epoch
    later=time.time()
    duration=later-self.now
    lr=float(tf.keras.backend.get_value(self.model.optimizer.lr)) # get the current learning rate
    current_lr=lr
    v_loss=logs.get('val_loss')  # get the validation loss for this epoch
    acc=logs.get('accuracy')  # get training accuracy
    v_acc=logs.get('val_accuracy')
    loss=logs.get('loss')
    if acc < self.threshold: # if training accuracy is below threshold adjust lr based on training accuracy
        monitor='accuracy'
        if acc>self.highest_tracc: # training accuracy improved in the epoch
            self.highest_tracc=acc # set new highest training accuracy
            self.best_weights=self.model.get_weights() # traing accuracy improved so save the weights
            self.count=0 # set count to 0 since training accuracy improved
            self.stop_count=0 # set stop counter to 0
            if v_loss<self.lowest_vloss:
                self.lowest_vloss=v_loss
            color= (0,255,0)
```



```python
                self.best_epoch=epoch + 1  # set the value of best epoch for this epoch
            else:
                # training accuracy did not improve check if this has happened for patience number of epochs
                # if so adjust learning rate
                if self.count>=self.patience -1: # lr should be adjusted
                    color=(245, 170, 66)
                    lr= lr* self.factor # adjust the learning by factor
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate in the optimizer
                    self.count=0 # reset the count to 0
                    self.stop_count=self.stop_count + 1 # count the number of consecutive lr adjustments
                    self.count=0 # reset counter
                    if self.dwell:
                        self.model.set_weights(self.best_weights) # return to better point in N space
                    else:
                        if v_loss<self.lowest_vloss:
                            self.lowest_vloss=v_loss
                else:
                    self.count=self.count +1 # increment patience counter
        else: # training accuracy is above threshold so adjust learning rate based on validation loss
            monitor='val_loss'
            if v_loss< self.lowest_vloss: # check if the validation loss improved
                self.lowest_vloss=v_loss # replace lowest validation loss with new validation loss
                self.best_weights=self.model.get_weights() # validation loss improved so save the weights
                self.count=0 # reset count since validation loss improved
                self.stop_count=0
                color=(0,255,0)
                self.best_epoch=epoch + 1 # set the value of the best epoch to this epoch
            else: # validation loss did not improve
```

```python
            if self.count>=self.patience-1: # need to adjust lr
                color=(245, 170, 66)
                lr=lr * self.factor # adjust the learning rate
                self.stop_count=self.stop_count + 1 # increment stop counter because lr was adjusted
                self.count=0 # reset counter
                tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate in the optimizer
                if self.dwell:
                    self.model.set_weights(self.best_weights) # return to better point in N space
            else:
                self.count =self.count +1 # increment the patience counter
            if acc>self.highest_tracc:
                self.highest_tracc= acc
        msg=f'{str(epoch+1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc*100:^9.3f}{v_loss:^9.5f}{v_acc*100:^9.3f}{current_lr:^9.5f}{lr:^9.5f}{monitor:^11s}{d
        print_in_color (msg,color, (55,65,80))
        if self.stop_count> self.stop_patience - 1: # check if learning rate has been adjusted stop_count times with no improvement
            msg=f' training has been halted at epoch {epoch + 1} after {self.stop_patience} adjustments of learning rate with no improvement'
            print_in_color(msg, (0,255,255), (55,65,80))
            self.model.stop_training = True # stop training
        else:
            if self.ask_epoch !=None:
                if epoch + 1 >= self.ask_epoch:
                    msg='enter H to halt ,F to fine tune model, or an integer for number of epochs to run then ask again'
                    print_in_color(msg, (0,255,255), (55,65,80))
                    ans=input('')
                    if ans=='H' or ans=='h':
                        msg=f'training has been halted at epoch {epoch + 1} due to user input'
                        print_in_color(msg, (0,255,255), (55,65,80))
```



```python
                    if ans=='H' or ans=='h':
                        msg=f'training has been halted at epoch {epoch + 1} due to user input'
                        print_in_color(msg, (0,255,255), (55,65,88))
                        self.model.stop_training = True # stop training
                    elif ans == 'F' or ans=='f':
                        msg='setting base_model as trainable for fine tuning of model'
                        self.base_model.trainable=True
                        print_in_color(msg, (0, 255,255), (55,65,88))
                        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:^8s}'.format('Epoch', 'Loss', 'Accuracy',
                                                                                              'V_loss','V_acc', 'LR', 'Next LR', 'Monitor', 'Duration')
                        print_in_color(msg, (244,252,3), (55,65,88))
                        self.count=0
                        self.stop_count=0
                        self.ask_epoch = epoch + 1 + self.ask_epoch_initial

                    else:
                        ans=int(ans)
                        self.ask_epoch +=ans
                        msg=f' training will continue until epoch ' + str(self.ask_epoch)
                        print_in_color(msg, (0, 255,255), (55,65,88))
                        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:^8s}'.format('Epoch', 'Loss', 'Accuracy',
                                                                                              'V_loss','V_acc', 'LR', 'Next LR', 'Monitor', 'Duration')
                        print_in_color(msg, (244,252,3), (55,65,88))
```

[14]:

```python
epochs = 30
patience= 1 # number of epochs to wait to adjust lr if monitored value does not improve
stop_patience =3 # number of epochs to wait before stopping training if monitored value does not improve
threshold=.9 # if train accuracy is < threshhold adjust monitor accuracy, else monitor validation loss
factor=.5 # factor to reduce lr by
dwell=True # experimental, if True and monitored metric does not improve on current epoch set  modelweights back to weights of previous epoch
freeze=False # if true free weights of  the base model
ask_epoch=20 # number of epochs to run before asking if you want to halt training
batches=train_steps


callbacks=[LRA(model=model,base_model= base_model,patience=patience,stop_patience=stop_patience, threshold=threshold,
               factor=factor,dwell=dwell, batches=batches,initial_epoch=0,epochs=epochs, ask_epoch=ask_epoch )]

history=model.fit(x=train_gen,  epochs=epochs, verbose=0, callbacks=callbacks,  validation_data=valid_gen,
            validation_steps=None,  shuffle=False,  initial_epoch=0)
```

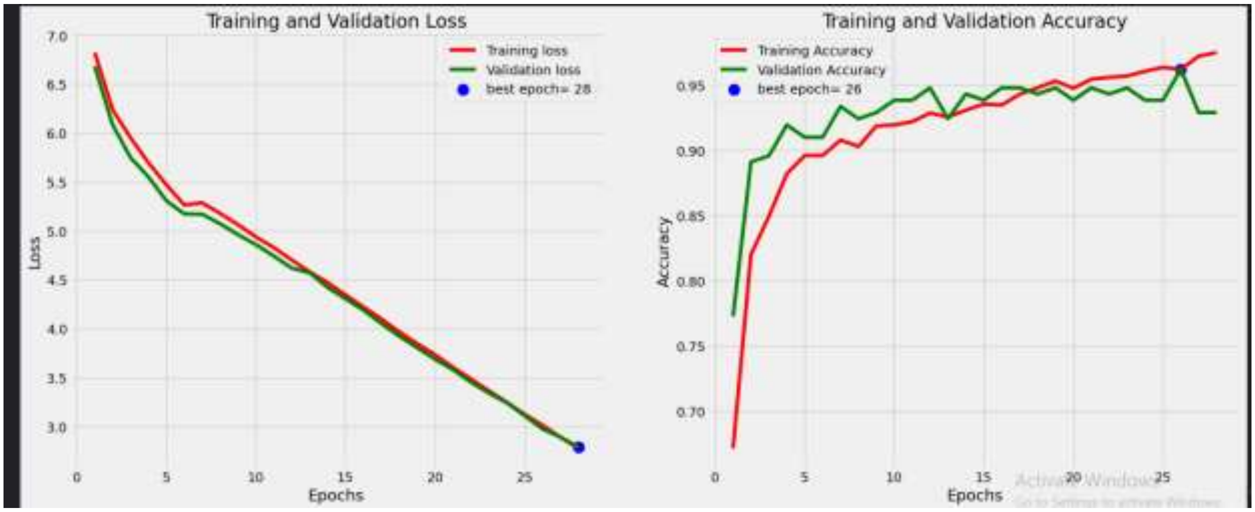
```
initializing callback starting train with base_model trainable
```

Epoch	Loss	Accuracy	V_loss	V_acc	LR	Next LR	Monitor	Duration
1 /30	6.823	67.136	6.68300	77.251	0.00010	0.00010	accuracy	104.44
2 /30	6.239	82.009	6.08374	89.100	0.00010	0.00010	accuracy	50.09
3 /30	5.952	84.934	5.74931	89.573	0.00010	0.00010	accuracy	50.88
4 /30	5.696	88.190	5.55143	91.943	0.00010	0.00010	accuracy	50.46
5 /30	5.471	89.597	5.30847	90.995	0.00010	0.00010	accuracy	50.48
6 /30	5.266	89.597	5.17403	90.995	0.00010	0.00005	accuracy	50.60
7 /30	5.287	90.784	5.17048	93.365	0.00005	0.00005	val_loss	50.60
8 /30	5.179	90.287	5.07375	92.417	0.00005	0.00005	val_loss	50.54
9 /30	5.063	91.860	4.96154	92.891	0.00005	0.00005	val_loss	50.57
10 /30	4.939	91.943	4.85973	93.839	0.00005	0.00005	val_loss	50.87
11 /30	4.829	92.191	4.74251	93.839	0.00005	0.00005	val_loss	50.74
12 /30	4.703	92.853	4.61943	94.787	0.00005	0.00005	val_loss	50.65
13 /30	4.582	92.550	4.57152	92.417	0.00005	0.00005	val_loss	50.38
14 /30	4.471	93.074	4.42204	94.313	0.00005	0.00005	val_loss	50.65
15 /30	4.346	93.543	4.30663	93.839	0.00005	0.00005	val_loss	50.74
16 /30	4.223	93.460	4.19027	94.787	0.00005	0.00005	val_loss	50.41
17 /30	4.101	94.288	4.05368	94.787	0.00005	0.00005	val_loss	50.39
18 /30	3.968	94.785	3.92377	94.313	0.00005	0.00005	val_loss	50.46
19 /30	3.847	95.309	3.80657	94.787	0.00005	0.00005	val_loss	50.53
20 /30	3.735	94.757	3.68732	93.839	0.00005	0.00005	val_loss	50.62
21 /30	3.607	95.447	3.58148	94.787	0.00005	0.00005	val_loss	50.65
22 /30	3.487	95.585	3.45381	94.313	0.00005	0.00005	val_loss	50.65

```
23 /30     3.366    95.695    3.34244   94.787    0.00005   0.00005   val_loss    50.88

24 /30     3.249    96.054    3.24359   93.839    0.00005   0.00005   val_loss    50.63

25 /30     3.126    96.358    3.11213   93.839    0.00005   0.00005   val_loss    50.80

26 /30     3.013    96.192    2.97481   96.209    0.00005   0.00005   val_loss    50.78

27 /30     2.890    97.213    2.89270   92.891    0.00005   0.00005   val_loss    50.66

28 /30     2.776    97.489    2.79111   92.891    0.00005   0.00005   val_loss    50.41

enter H to halt ,F to fine tune model, or an integer for number of epochs to run then ask again

 H
training has been halted at epoch 28 due to user input

Training is completed - model is set with weights from epoch 28

training elapsed time was 0.0 hours, 25.0 minutes, 51.22 seconds)
```

User defined function for train plot

```python
def tr_plot(tr_data, start_epoch):
    #Plot the training and validation data
    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)#  this is the epoch with the lowest validation loss
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(20,8))
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', label=sc_label)
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs')
    axes[0].set_ylabel('Loss')
    axes[0].legend()
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', label=vc_label)
```



From the above run time, we can observe that at 26 th epoch the model is giving the best accuracy.

Activity 5: Save the Model

A custom defined function has written for saving the model in the .h5 format.

Definition of the custom defined saver function.

```
def saver(save_path, model, model_name, subject, accuracy,img_size, scalar, generator):
    # first save the model
    save_id=str (model_name +  '-' + subject +'-'+ str(acc)[:str(acc).rfind('.')+3] + '.h5')
    model_save_loc=os.path.join(save_path, save_id)
    model.save(model_save_loc)
    print_in_color ('model was saved as ' + model_save_loc, (0,255,0),(55,65,80))
    # now create the class_df and convert to csv file
    class_dict=generator.class_indices
    height=[]
    width=[]
    scale=[]
    for i in range(len(class_dict)):
        height.append(img_size[0])
        width.append(img_size[1])
        scale.append(scalar)
    Index_series=pd.Series(list(class_dict.values()), name='class_index')
    Class_series=pd.Series(list(class_dict.keys()), name='class')
    Height_series=pd.Series(height, name='height')
    Width_series=pd.Series(width, name='width')
    Scale_series=pd.Series(scale, name='scale by')
    class_df=pd.concat([Index_series, Class_series, Height_series, Width_series, Scale_series], axis=1)
    csv_name='class_dict.csv'
    csv_save_loc=os.path.join(save_path, csv_name)
    class_df.to_csv(csv_save_loc, index=False)
    print_in_color ('class csv file was saved as ' + csv_save_loc, (0,255,0),(55,65,80))
    return model_save_loc, csv_save_loc
```

```
save_dir='/kaggle/working/'
subject='eye_disease'
acc=model.evaluate( test_gen, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
msg=f'accuracy on the test set is {acc:5.2f} %'
print_in_color(msg, (0,255,0),(55,65,80))
save_id=str (model_name +  '-' + subject +'-'+ str(acc)[:str(acc).rfind('.')+3] + '.h5')
save_loc=os.path.join(save_dir, save_id)
model.save(save_loc)
generator=train_gen
scale = 1
result=saver(save_dir, model, model_name, subject, acc, img_size, scale,  generator)
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legac
y. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
model was saved as /kaggle/working/VGG19-eye_disease-93.83.h5

class csv file was saved as /kaggle/working/class_dict.csv
```

The model is saved with .h5 extension as follows

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Testing the model:

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model using load_model.

Custom defined print_info function, for visualising the test results and testing the model.

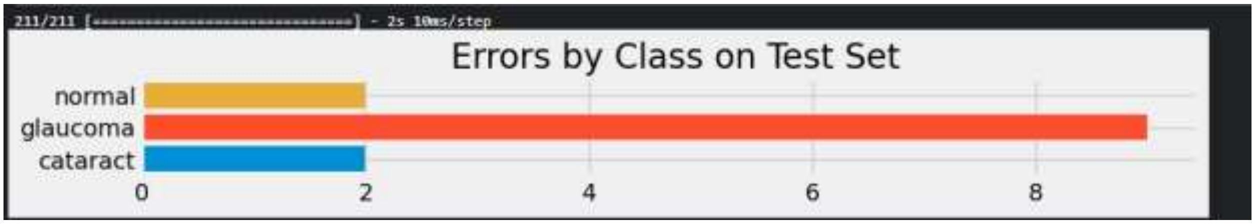
```
def print_info( test_gen, preds, print_code, save_dir, subject ):
    class_dict=test_gen.class_indices
    labels= test_gen.labels
    file_names= test_gen.filenames
    error_list=[]
    true_class=[]
    pred_class=[]
    prob_list=[]
    new_dict={}
    error_indices=[]
    y_pred=[]
    for key,value in class_dict.items():
        new_dict[value]=key             # dictionary {integer of class number: string of class name}
    # store new_dict as a text fine in the save_dir
    classes=list(new_dict.values())     # list of string of class names
    errors=0
    for i, p in enumerate(preds):
        pred_index=np.argmax(p)
        true_index=labels[i]  # labels are integer values
        if pred_index != true_index: # a misclassification has occurred
            error_list.append(file_names[i])
            true_class.append(new_dict[true_index])
            pred_class.append(new_dict[pred_index])
            prob_list.append(p[pred_index])
            error_indices.append(true_index)
            errors=errors + 1
        y_pred.append(pred_index)
```

```python
    if print_code !=0:
        if errors>0:
            if print_code>errors:
                r=errors
            else:
                r=print_code
            msg='{0:^28s}{1:^28s}{2:^28s}{3:^16s}'.format('Filename', 'Predicted Class' , 'True Class', 'Probability')
            print_in_color(msg, (0,255,0),(55,65,80))
            for i in range(r):
                split1=os.path.split(error_list[i])
                split2=os.path.split(split1[0])
                fname=split2[1] + '/' + split1[1]
                msg='{0:^28s}{1:^28s}{2:^28s}{3:4s}{4:^6.4f}'.format(fname, pred_class[i],true_class[i], ' ', prob_list[i])
                print_in_color(msg, (255,255,255), (55,65,60))
                #print(error_list[i] , pred_class[i], true_class[i], prob_list[i])
        else:
            msg='With accuracy of 100 % there are no errors to print'
            print_in_color(msg, (0,255,0),(55,65,80))
    if errors>0:
        plot_bar=[]
        plot_class=[]
        for  key, value in new_dict.items():
            count=error_indices.count(key)
            if count!=0:
                plot_bar.append(count) # list containg how many times a class c had an error
                plot_class.append(value)   # stores the class
        fig=plt.figure()
        fig.set_figheight(len(plot_class)/3)
        fig.set_figwidth(10)
        plt.style.use('fivethirtyeight')
        for i in range(0, len(plot_class)):
            c=plot_class[i]
            x=plot_bar[i]
            plt.barh(c, x, )
            plt.title( ' Errors by Class on Test Set')
```

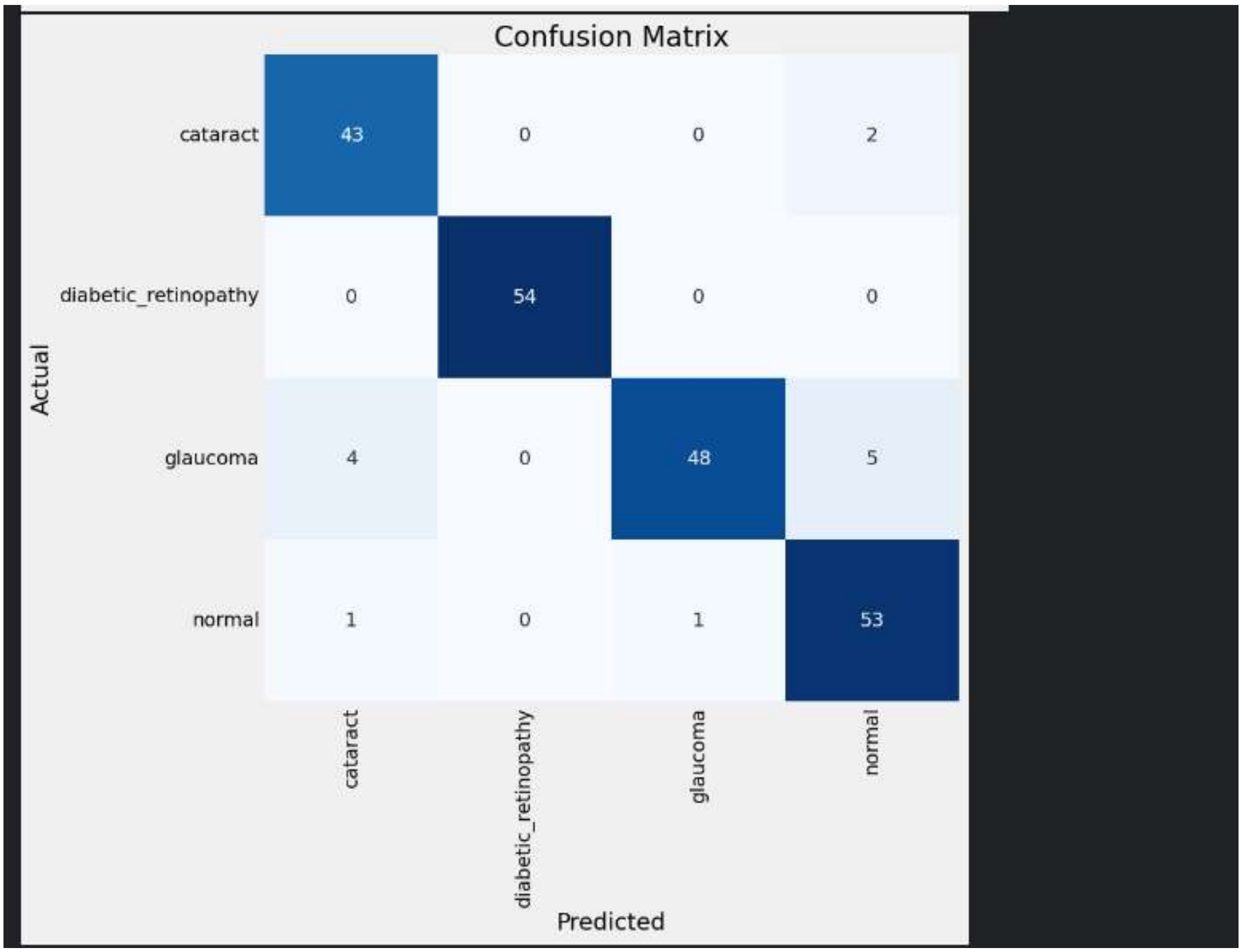
```python
    y_true= np.array(labels)
    y_pred=np.array(y_pred)
    if len(classes)<= 30:
        # create a confusion matrix
        cm = confusion_matrix(y_true, y_pred )
        length=len(classes)
        if length<8:
            fig_width=8
            fig_height=8
        else:
            fig_width= int(length * .5)
            fig_height= int(length * .5)
        plt.figure(figsize=(fig_width, fig_height))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
        plt.xticks(np.arange(length)+.5, classes, rotation= 90)
        plt.yticks(np.arange(length)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes)
    print("Classification Report:\n----------------------\n", clr)
```

```python
print_code=0
preds=model.predict(test_gen)
print_info( test_gen, preds, print_code, save_dir, subject )
```

Output:



Confusion matrix for the testing data



Classification report of model on test dataset

```
Classification Report:
----------------------
                      precision    recall  f1-score   support

            cataract       0.90      0.96      0.92        45
diabetic_retinopathy       1.00      1.00      1.00        54
            glaucoma       0.98      0.84      0.91        57
              normal       0.88      0.96      0.92        55

            accuracy                           0.94       211
           macro avg       0.94      0.94      0.94       211
        weighted avg       0.94      0.94      0.94       211
```

Probabilities of predicted eye disease of test dataset Images:

```
[[9.47886780e-02 6.79917783e-02 3.62593949e-01 4.74625617e-01]
 [3.20689008e-02 1.51970917e-02 3.32672931e-02 9.19466674e-01]
 [2.88582873e-02 1.15023805e-02 9.36146140e-01 2.34932024e-02]
 [6.16099685e-03 3.80260032e-03 9.78645742e-01 1.13906628e-02]
 [6.98447553e-03 6.17412664e-03 9.20284353e-03 9.77638543e-01]
 [4.74282587e-03 3.09469341e-03 7.59920664e-03 9.84563291e-01]
 [1.03082717e-03 9.96003926e-01 3.79189529e-04 2.58605136e-03]
 [4.36505332e-04 3.46504530e-04 9.98899579e-01 3.17389320e-04]
 [2.18856931e-02 2.51646712e-02 5.23394831e-02 9.00610149e-01]
 [8.88059475e-03 9.73006010e-01 4.16361261e-03 1.39497416e-02]
 [9.95534897e-01 1.37462304e-03 1.87768054e-03 1.21274311e-03]
 [3.60637121e-02 6.50051283e-03 9.51133668e-01 6.30206661e-03]
 [9.75324631e-01 6.92449464e-03 3.93394195e-03 1.38170449e-02]
 [4.13046684e-03 9.87467110e-01 3.05534760e-03 5.34705911e-03]
 [9.85990405e-01 6.40183361e-03 3.30417859e-03 4.30353079e-03]
 [7.37557362e-04 9.96156514e-01 1.01816398e-03 2.08776770e-03]
 [3.41510661e-02 2.74470542e-02 9.20717657e-01 1.76842101e-02]
 [8.29122495e-03 2.24637371e-02 1.42798079e-02 9.54965174e-01]
 [4.72807884e-03 9.76809680e-01 1.64089736e-03 1.68213397e-02]
 [8.03965330e-01 3.52401584e-02 6.25611469e-02 9.82333571e-02]
 [1.92953777e-02 6.02640696e-02 3.64474244e-02 8.83993149e-01]
 [9.92089987e-01 2.76019797e-03 2.69994140e-03 2.44986964e-03]
 [9.88374576e-02 4.08385694e-02 7.45089412e-01 1.15234524e-01]
```

Milestone 4: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages

● Building python code
● Run the programme

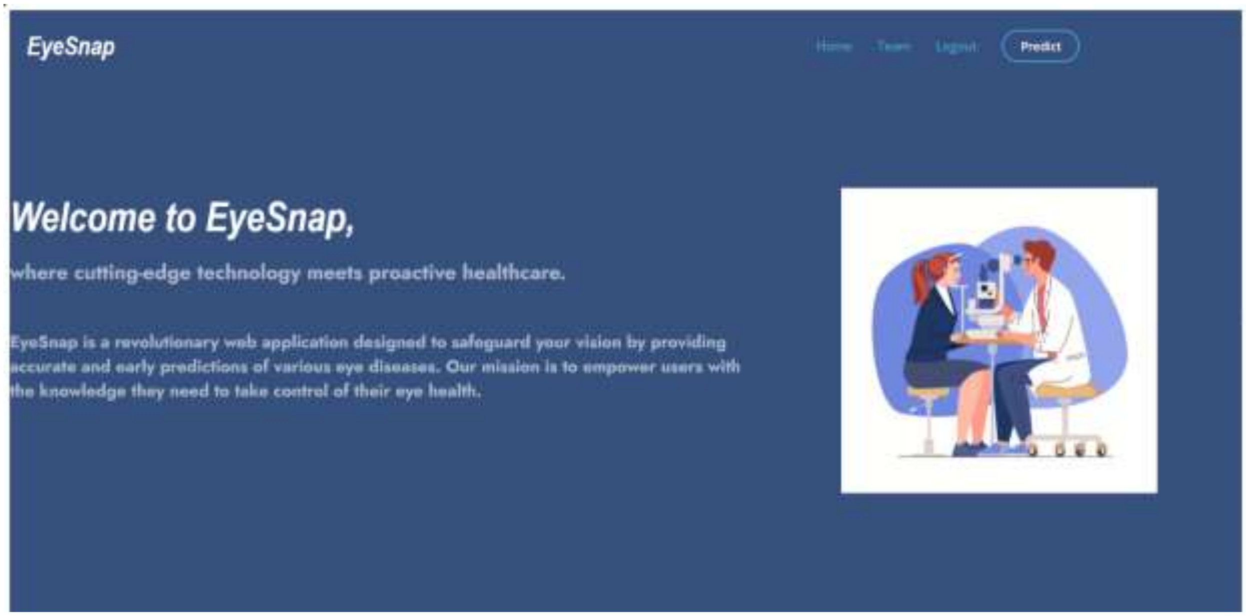
Activity1: Building HTML Pages:

For this project create one HTML file namely

● index.html

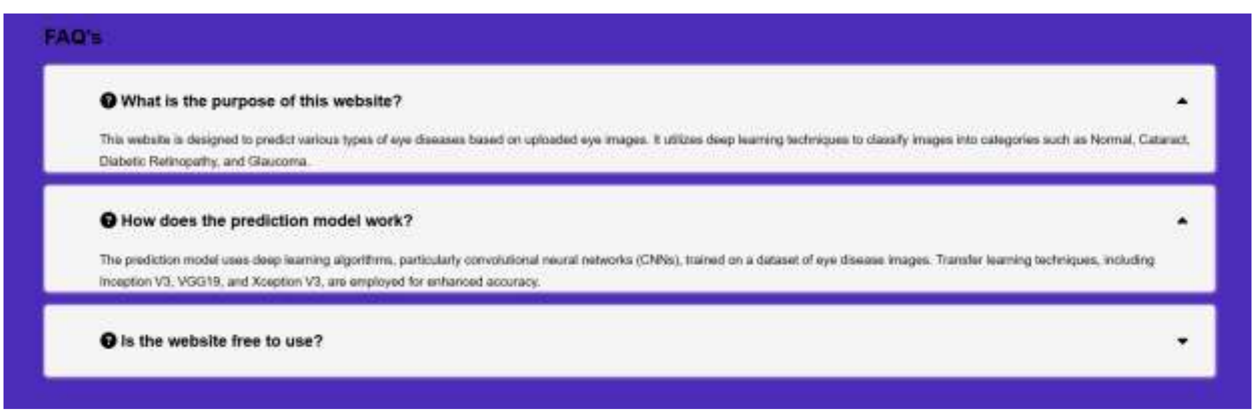
Let’s see how our index.html page looks like:

predict section




TEAM
RAHUL CHEREDDY
Vellore Institute of Technology
KESAVA KUMAR MATTUPALLI
Vellore Institute of Technology
TEJASWINI RAYA
Vellore Institute of Technology
SREERAM VENKATA SAI SUCHITHA
Vellore Institute of Technology


Eye Disease Prediction and Awareness
Learn about various eye diseases, their symptoms, prevention, and treatment options.
Raise awareness about eye health and take the necessary steps to protect your vision.
Upload Image Here To Identify the Eye Condition
Choose Image
Selected Image
Predict!


FAQ's
What is the purpose of this website?
This website is designed to predict various types of eye diseases based on uploaded eye images. It utilizes deep learning techniques to classify images into categories such as Normal, Cataract, Diabetic Retinopathy, and Glaucoma.
How does the prediction model work?
The prediction model uses deep learning algorithms, particularly convolutional neural networks (CNNs), trained on a dataset of eye disease images. Transfer learning techniques, including Inception V3, VGG19, and Xception V3, are employed for enhanced accuracy.
Is the website free to use?

Activity 2: Build Python code:

Import the libraries

```
app1.py > logout
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, request, render_template, jsonify,session,redirect,g,url_for
import os
```

Loading the saved model and initializing the flask app

```
app = Flask(__name__, static_folder='static')

model = load_model("VGG19-eye_disease-95.73.h5", compile=False)
app.secret_key = os.urandom(24)
```

Render HTML pages:

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if 'username' in session:
        return redirect(url_for('protected'))

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        if username in users and users[username] == password:
            session['username'] = username
            return redirect(url_for('protected'))

    return render_template('login.html')
```

```python
@app.route('/protected')
def protected():
    if 'username' in session:
        return render_template('home.html', user=session['username'])
    return redirect(url_for('login'))
@app.route('/login.html')
def login():
        return render_template('login.html')

@app.route('/index.html')
def indexing():
        return render_template('index.html')

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('login'))
```

Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with index.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

```python
@app.route('/predict', methods=['POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        print("upload folder is ", filepath)
        f.save(filepath)

        # img = image.load_img(filepath, target_size=(64, 64))
        img = image.load_img(filepath, target_size=(224, 224))

        x = image.img_to_array(img)
        print(x)
        x = np.expand_dims(x, axis=0)
        print(x)
        y = model.predict(x)
        preds = np.argmax(y, axis=1)

        print("prediction", preds)

        index = ['Cataract', 'diabetic_retinopathy', 'Glaucoma','Normal']
        prediction_result = index[preds[0]]

```

Here we are routing our app to predict function. This function retrieves all the values from the HTML
page using Post request. That is stored in an array. This array is passed to the model.predict() function.
This function returns the prediction. And this prediction value will rendered to the text that we have
mentioned in the index.html page earlier.

Main Function:

```python

if __name__ == '__main__':
    app.run(debug=False, threaded=False)

```

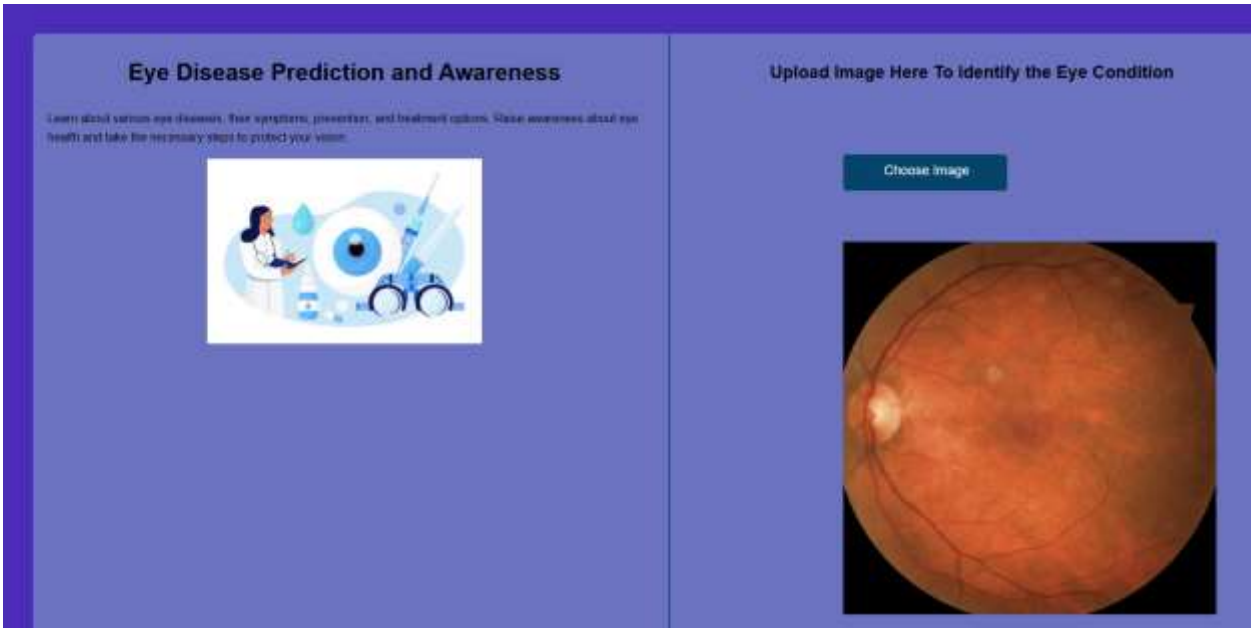
Activity 3: Run the application

● Open VSCODE
● Navigate to the folder where your Python script is.
● Now click on the green play button above.
● Click on the predict button from the top right corner, enter the inputs, click on the Classify button, and see the result/prediction on the web.

```
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app1'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [19/Nov/2023 12:39:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Nov/2023 12:39:38] "GET /static/js/script.js HTTP/1.1" 304 -
127.0.0.1 - - [19/Nov/2023 12:39:38] "GET /static/css/login.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Nov/2023 12:39:38] "GET /favicon.ico HTTP/1.1" 404 -
```

The home page looks like this. When you click on the Predict button, you’ll be redirected to the predict section

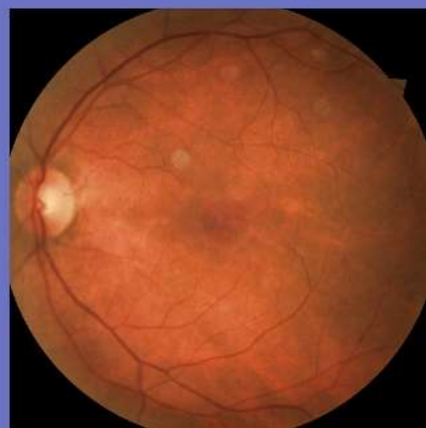
Input 1:



Output1:

Upload Image Here To Identify the Eye Condition

Choose Image



Result: You may have *diabetic retinopathy*. It is recommended to consult with an eye specialist.

Diabetic retinopathy is a serious eye condition that affects people with diabetes, primarily those who have had the disease for a long time or have poorly managed their blood sugar levels. It can lead to vision impairment and even blindness if not properly diagnosed and treated. Here is some information to raise awareness about diabetic retinopathy:

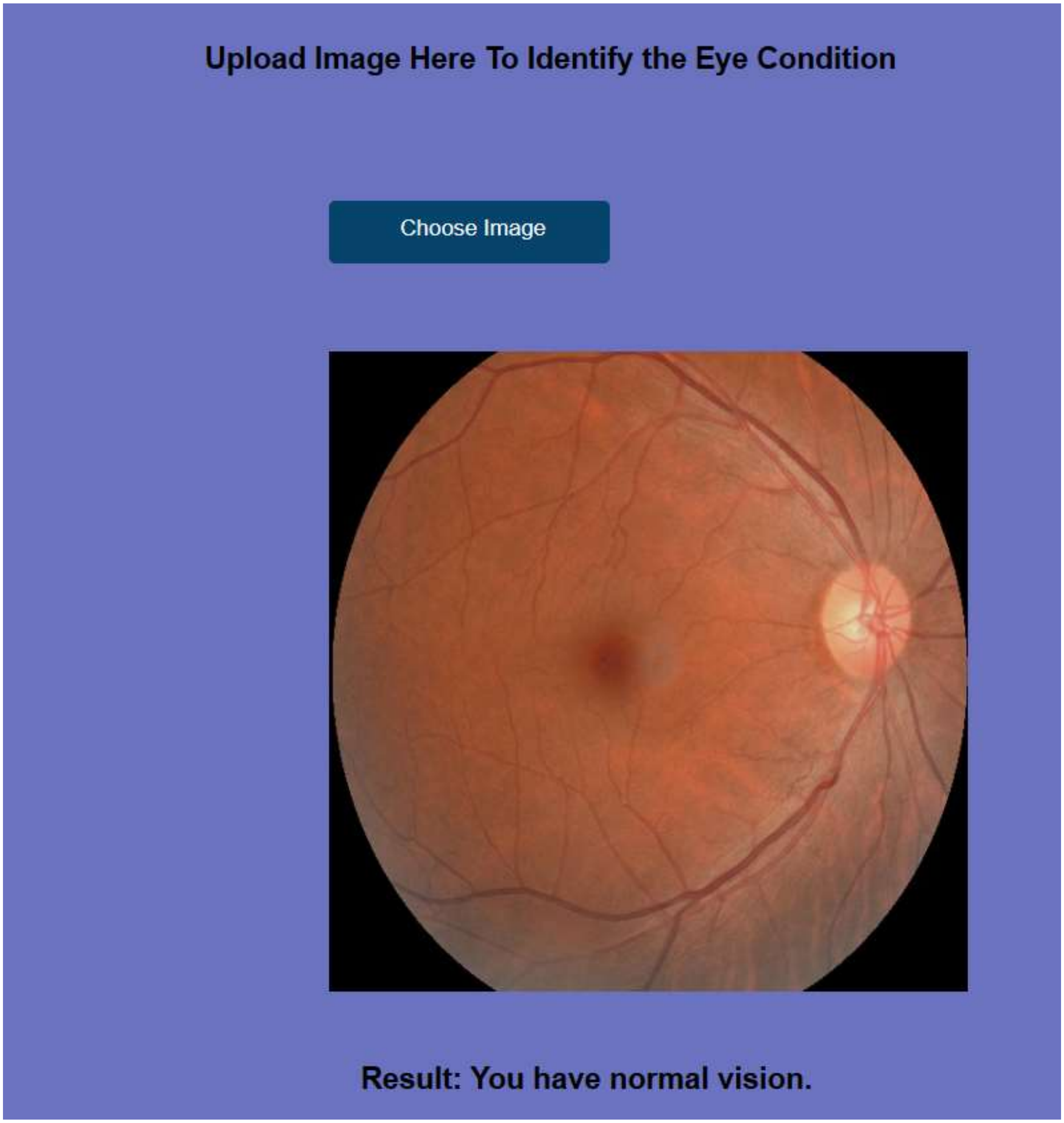
<u>Treatment:</u> Treatment options for diabetic retinopathy depend on the stage and severity of the disease. They can include laser therapy to seal leaking blood vessels, medications injected into the eye, and surgery to remove blood from the vitreous gel. Effective management of diabetes through blood sugar control and other health measures is also essential to slow or prevent the progression of diabetic retinopathy.

<u>Prevention and awareness:</u> Raising awareness about diabetic retinopathy is crucial, as early detection and management are key to preserving vision. People with diabetes should be educated about the importance of regular eye exams and good diabetes management to reduce the risk of diabetic retinopathy.

Input2:


Eye Disease Prediction and Awareness
Upload Image Here To Identify the Eye Condition
Choose Image

Output-2:


Upload Image Here To Identify the Eye Condition
Choose Image
Result: You have normal vision.