Eye Disease Prediction using Deep learning

Introduction:

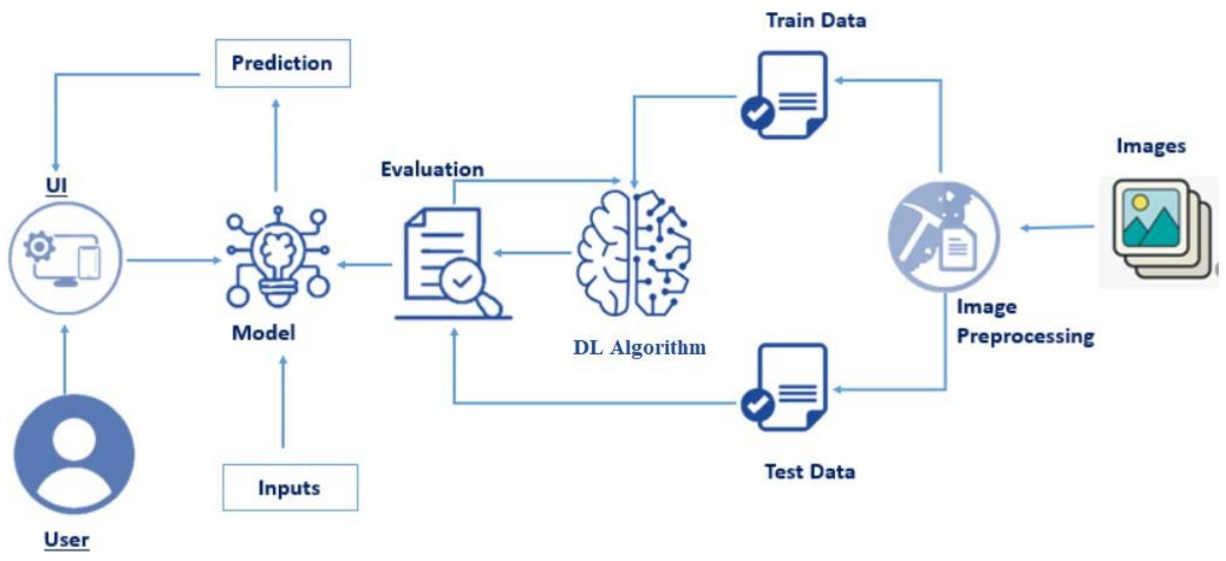
The field of medical diagnostics has witnessed remarkable advancements in recent years, thanks to the integration of cutting-edge technologies like deep learning. One area where this innovation has shown significant promise is in the early detection and prediction of eye diseases. Eye diseases, if left undetected and untreated, can lead to severe visual impairment and even irreversible damage. Therefore, the development of accurate and efficient diagnostic tools is crucial for timely intervention and improved patient outcomes.

This project aims to leverage the power of deep learning algorithms to predict and diagnose eye diseases. Deep learning, a subset of artificial intelligence, has demonstrated exceptional capabilities in image recognition and pattern analysis. By training a deep neural network on a diverse dataset of ocular images, the system can learn intricate patterns and features associated with various eye conditions. Once trained, the model can then analyze new, unseen images and provide predictions regarding the presence of specific eye diseases.

The significance of this project lies in its potential to revolutionize the traditional methods of eye disease diagnosis. Conventional diagnostic approaches often rely on manual examination by healthcare professionals, which can be time-consuming and may not always catch subtle early signs of diseases. By automating the diagnostic process through deep learning, we can enhance the speed and accuracy of detection, allowing for prompt medical intervention and personalized treatment plans.

Throughout this project, we will explore the key components of deep learning, including the architecture of neural networks, the process of training on labeled datasets, and the fine-tuning of models for optimal performance. Additionally, we will delve into the ethical considerations and challenges associated with implementing deep learning in medical applications.

Technical Architecture:



Prerequisites:

To complete this project, you must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

1. **To build Machine learning models you must require the following packages**

- **Numpy**:
  - It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- **Scikit-learn:**
  - It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- **Flask:**
  Web framework used for building Web applications
- **Python packages:**
  - open anaconda prompt as administrator
  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type “pip install tensorflow==2.3.2” and click enter.
  - Type “pip install keras==2.3.1” and click enter.

    - Type “pip install Flask” and click enter.

- **Deep Learning Concepts**
    - **CNN:** a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.
      CNN Basic
    - **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

      Flask Basics

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using the Flask framework.

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.
- CNN Models analyze the image, then prediction is showcased on the Flask UI.

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

- Data Collection.
    - Create Train and Test Folders.
- Data Preprocessing.
    - Import the ImageDataGenerator library
    - Configure ImageDataGenerator class
    - ApplyImageDataGenerator functionality to Trainset and Testset
- Model Building
    - Import the model building Libraries
    - Initializing the model
    - Adding Input Layer
    - Adding Hidden Layer

        - Adding Output Layer
        - Configure the Learning Process
        - Training and testing the model
        - Save the Model
    - Application Building
        - Create an HTML file
        - Build Python Code

Project Structure:

Create a Project folder

- The Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the **templates** folder and a python script **app.py** for server side scripting
- we need the model which is saved and the saved model in this content is a Eye.h5
- templates folder contains base.html,index.html pages

Milestone 1: Data Collection

Collect the images of the various eye disease into subdirectories based on respective names .Create folders of types of Eye disease that need to be recognized .In this project ,we have collected images of 6 types of eye diseases like Catract,bulgingeyes,crossedeyes,uveitis,glaucoma and they are saved in the respective sub directories with their respective names.

Download the Dataset

- https://www.kaggle.com/code/shefaligoyal/eye-disease-classification-through-deep-learning

Milestone 2: Image Preprocessing

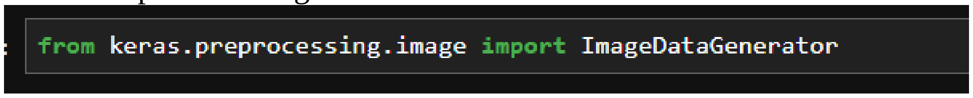
In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1:Import the ImageDataGenerator library

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from tensorflow Keras



```
from keras.preprocessing.image import ImageDataGenerator
```

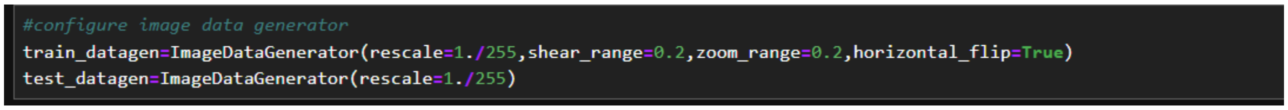
Activity 2: **Configure ImageDataGenerator class**

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.

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

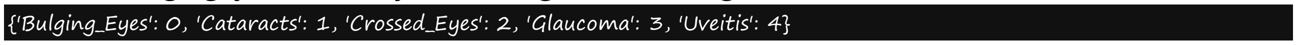


```
#configure image data generator
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
```

Activity 3:Apply ImageDataGenerator functionality to Trainset and Testset

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

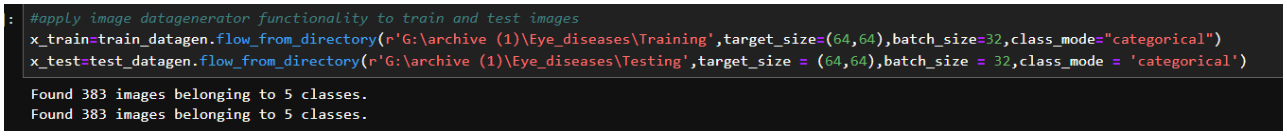
This function will return batches of images from the subdirectories Catract,bulgingeyes,crossedeyes,uveitis,glaucoma, 'together with labels 0 to 5



```
{'Bulging_Eyes': 0, 'Cataracts': 1, 'Crossed_Eyes': 2, 'Glaucoma': 3, 'Uveitis': 4}
```

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 32.
- 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).



```
#apply image datagenerator functionality to train and test images
x_train=train_datagen.flow_from_directory(r'G:\archive (1)\Eye_diseases\Training',target_size=(64,64),batch_size=32,class_mode="categorical")
x_test=test_datagen.flow_from_directory(r'G:\archive (1)\Eye_diseases\Testing',target_size = (64,64),batch_size = 32,class_mode = 'categorical')

Found 383 images belonging to 5 classes.
Found 383 images belonging to 5 classes.
```

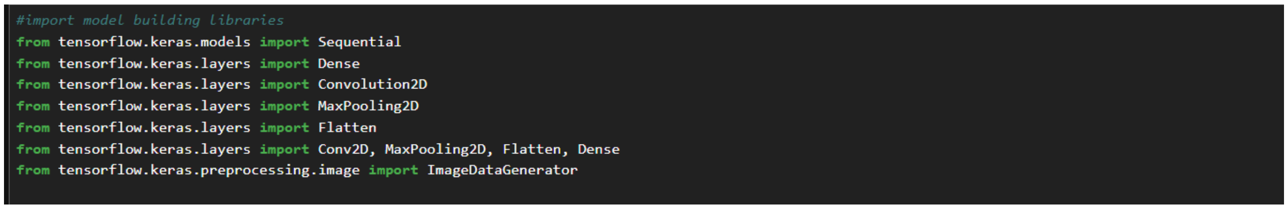
We notice that 383 images belong to 5 classes for training and 383 images belong to 5 classes for testing purposes.

Milestone 3: Model Building

Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries



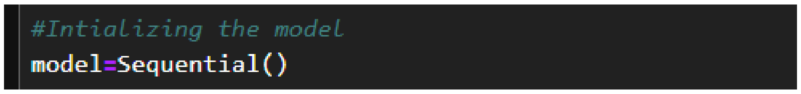
```
#import model building libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Activity 2: Initializing the model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.



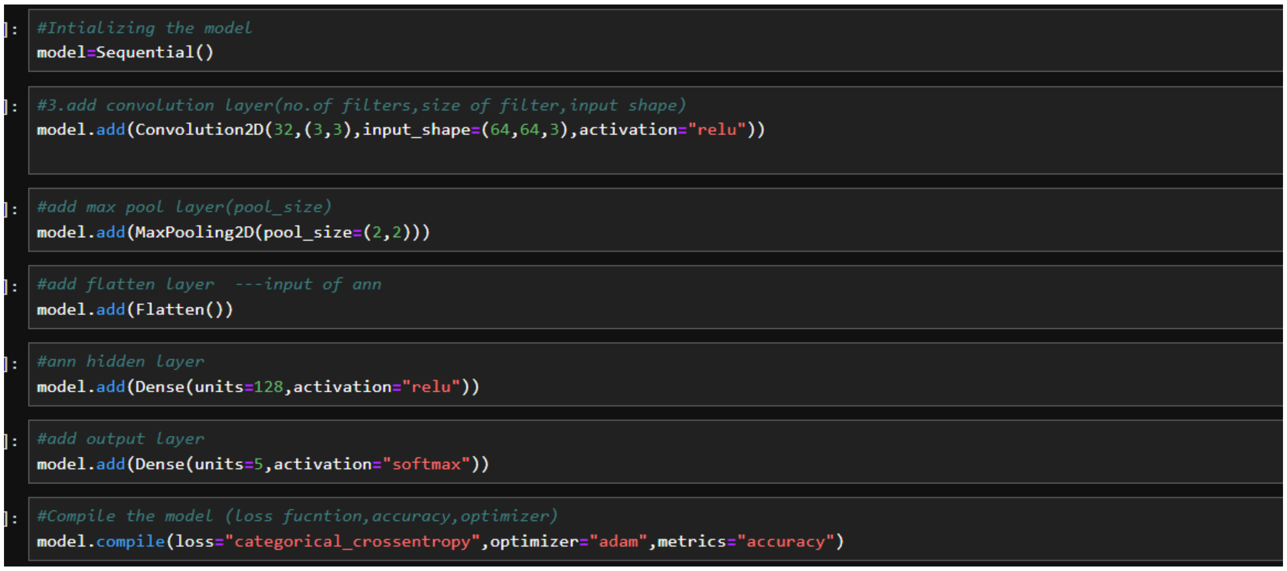
```
#Intializing the model
model=Sequential()
```

Activity 3: Adding CNN Layers

- As the input image contains three channels, we are specifying the input shape as (128,128,3).

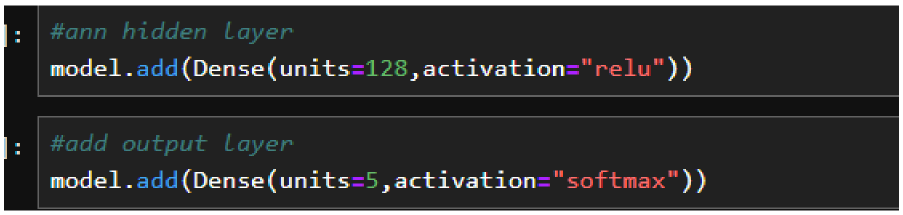
- We are adding a convolution layer with activation function as "relu" and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.
- Max pool layer is used to downsample the input.(Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter)

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



```
]: #Intializing the model
   model=Sequential()

]: #3.add convolution layer(no.of filters,size of filter,input shape)
   model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),activation="relu"))

]: #add max pool layer(pool_size)
   model.add(MaxPooling2D(pool_size=(2,2)))

]: #add flatten layer  ---input of ann
   model.add(Flatten())

]: #ann hidden layer
   model.add(Dense(units=128,activation="relu"))

]: #add output layer
   model.add(Dense(units=5,activation="softmax"))

]: #Compile the model (loss fucntion,accuracy,optimizer)
   model.compile(loss="categorical_crossentropy",optimizer="adam",metrics="accuracy")
```

Adding fully connected layer

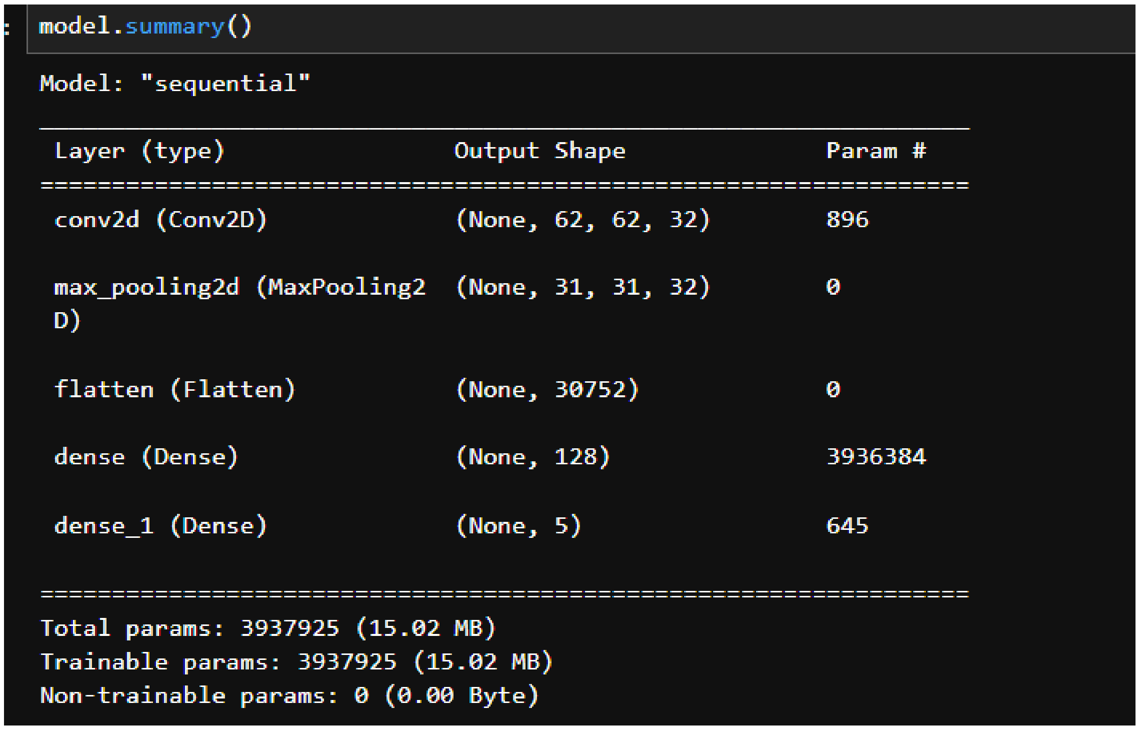


```
]: #ann hidden layer
   model.add(Dense(units=128,activation="relu"))

]: #add output layer
   model.add(Dense(units=5,activation="softmax"))
```

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

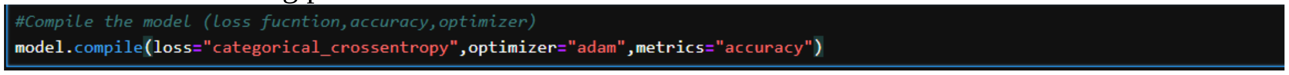


```
model.summary()

Model: "sequential"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 flatten (Flatten)           (None, 30752)             0

 dense (Dense)               (None, 128)               3936384

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3937925 (15.02 MB)
Trainable params: 3937925 (15.02 MB)
Non-trainable params: 0 (0.00 Byte)
```

Activity 6: 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



```
#Compile the model (loss fucntion,accuracy,optimizer)
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics="accuracy")
```

Activity 7: Train The model

Now, let us train our model with our image dataset. The model is trained for 10 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 till 10 epochs and probably there is further scope to improve the model.

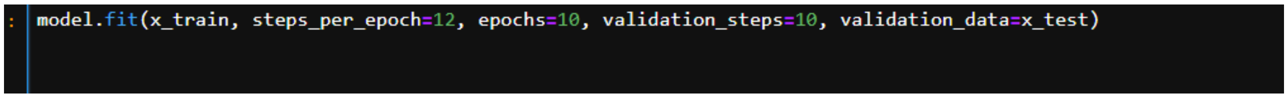
fit_generator functions used to train a deep learning neural network

Arguments:

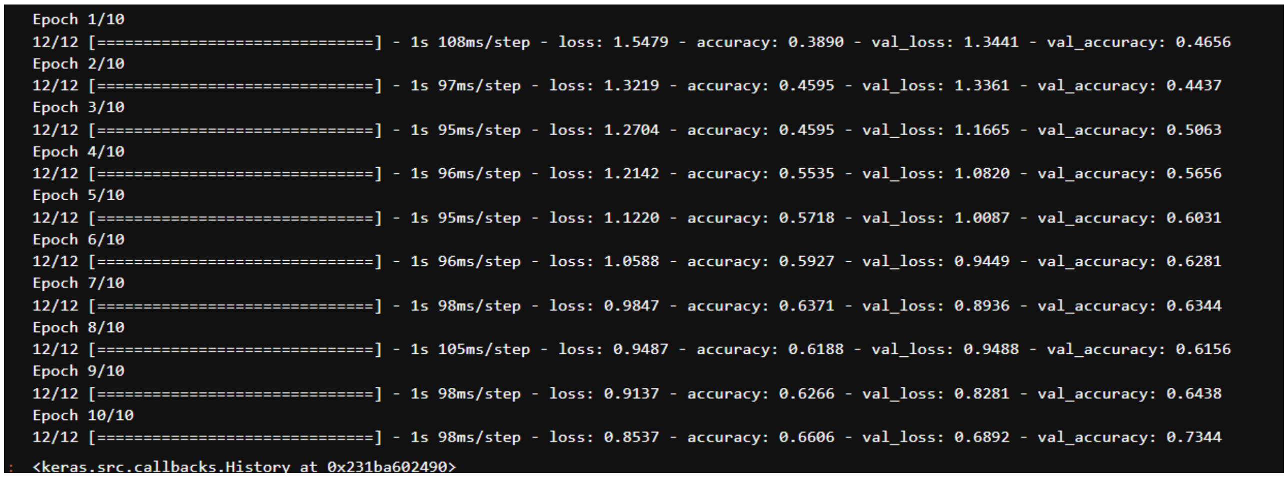
- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
- Epochs: an integer and number of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

Fit the model



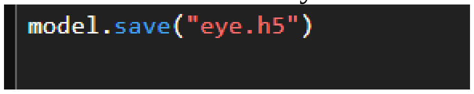
```
model.fit(x_train, steps_per_epoch=12, epochs=10, validation_steps=10, validation_data=x_test)
```



```
Epoch 1/10
12/12 [==============================] - 1s 108ms/step - loss: 1.5479 - accuracy: 0.3890 - val_loss: 1.3441 - val_accuracy: 0.4656
Epoch 2/10
12/12 [==============================] - 1s 97ms/step - loss: 1.3219 - accuracy: 0.4595 - val_loss: 1.3361 - val_accuracy: 0.4437
Epoch 3/10
12/12 [==============================] - 1s 95ms/step - loss: 1.2704 - accuracy: 0.4595 - val_loss: 1.1665 - val_accuracy: 0.5063
Epoch 4/10
12/12 [==============================] - 1s 96ms/step - loss: 1.2142 - accuracy: 0.5535 - val_loss: 1.0820 - val_accuracy: 0.5656
Epoch 5/10
12/12 [==============================] - 1s 95ms/step - loss: 1.1220 - accuracy: 0.5718 - val_loss: 1.0087 - val_accuracy: 0.6031
Epoch 6/10
12/12 [==============================] - 1s 96ms/step - loss: 1.0588 - accuracy: 0.5927 - val_loss: 0.9449 - val_accuracy: 0.6281
Epoch 7/10
12/12 [==============================] - 1s 98ms/step - loss: 0.9847 - accuracy: 0.6371 - val_loss: 0.8936 - val_accuracy: 0.6344
Epoch 8/10
12/12 [==============================] - 1s 105ms/step - loss: 0.9487 - accuracy: 0.6188 - val_loss: 0.9488 - val_accuracy: 0.6156
Epoch 9/10
12/12 [==============================] - 1s 98ms/step - loss: 0.9137 - accuracy: 0.6266 - val_loss: 0.8281 - val_accuracy: 0.6438
Epoch 10/10
12/12 [==============================] - 1s 98ms/step - loss: 0.8537 - accuracy: 0.6606 - val_loss: 0.6892 - val_accuracy: 0.7344
<keras.src.callbacks.History at 0x231ba602490>
```

Activity 8: Save the Model

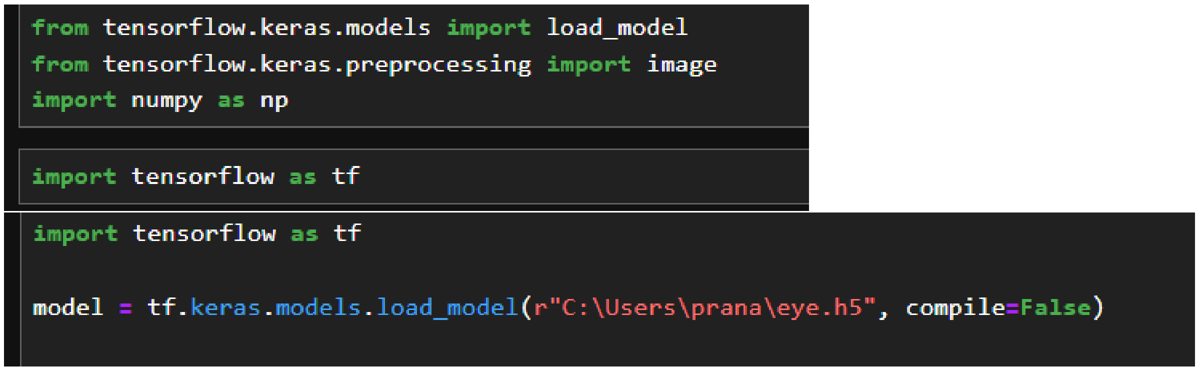
The model is saved with .h5 extension as follows
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.



```
model.save("eye.h5")
```

Activity 9: Test 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

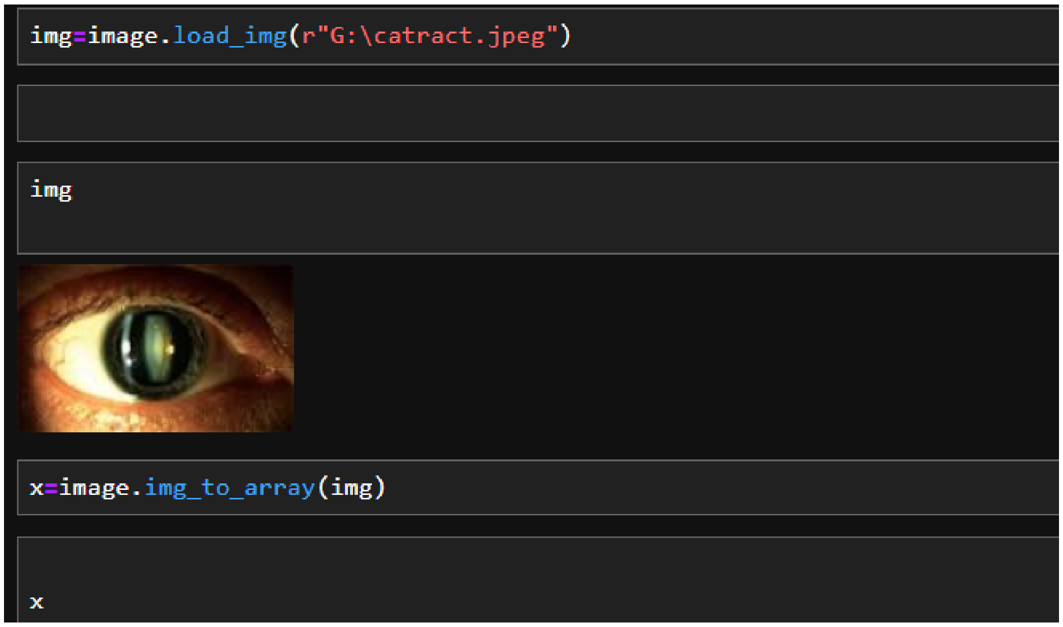


```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
```

```
import tensorflow as tf
```

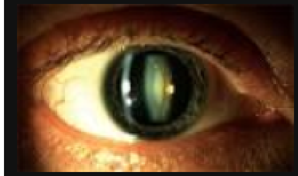
```
import tensorflow as tf

model = tf.keras.models.load_model(r"C:\Users\prana\eye.h5", compile=False)
```

Loading the image and converting it into an array



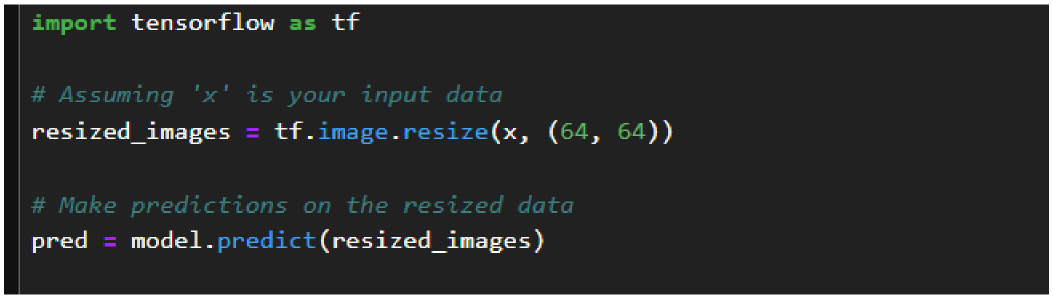
```
img=image.load_img(r"G:\catract.jpeg")
```

```
img
```

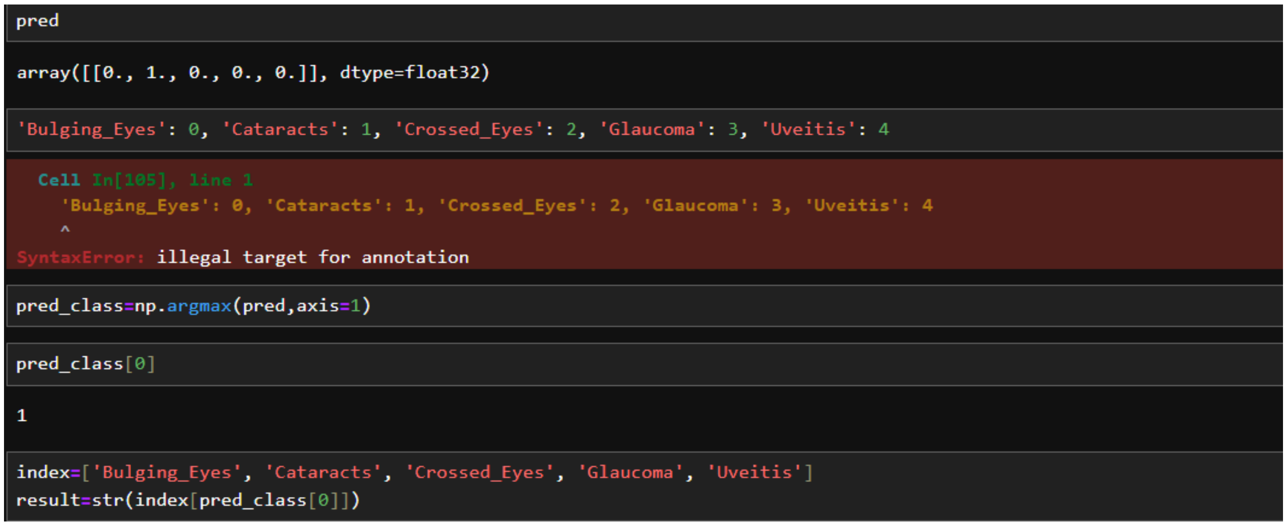


```
x=image.img_to_array(img)
```

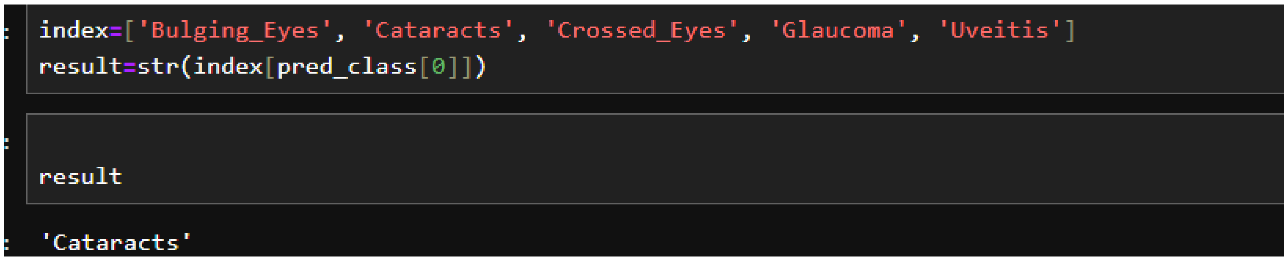
```
x
```



```
import tensorflow as tf

# Assuming 'x' is your input data
resized_images = tf.image.resize(x, (64, 64))

# Make predictions on the resized data
pred = model.predict(resized_images)
```



```
pred

array([[0., 1., 0., 0., 0.]], dtype=float32)

'Bulging_Eyes': 0, 'Cataracts': 1, 'Crossed_Eyes': 2, 'Glaucoma': 3, 'Uveitis': 4

  Cell In[105], line 1
    'Bulging_Eyes': 0, 'Cataracts': 1, 'Crossed_Eyes': 2, 'Glaucoma': 3, 'Uveitis': 4
    ^
SyntaxError: illegal target for annotation

pred_class=np.argmax(pred,axis=1)

pred_class[0]

1

index=['Bulging_Eyes', 'Cataracts', 'Crossed_Eyes', 'Glaucoma', 'Uveitis']
result=str(index[pred_class[0]])
```



```
index=['Bulging_Eyes', 'Cataracts', 'Crossed_Eyes', 'Glaucoma', 'Uveitis']
result=str(index[pred_class[0]])

result

'Cataracts'
```

Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1 : Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- home.html displays the home page.

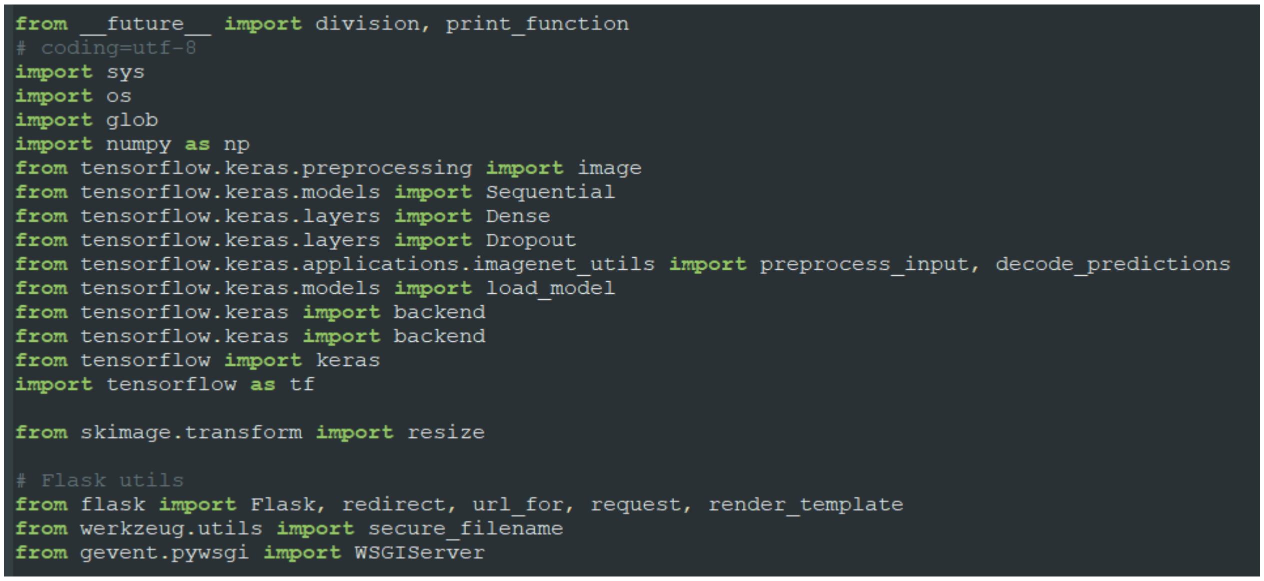
Intro.html displays an introduction about the project

Task 1: Importing Libraries

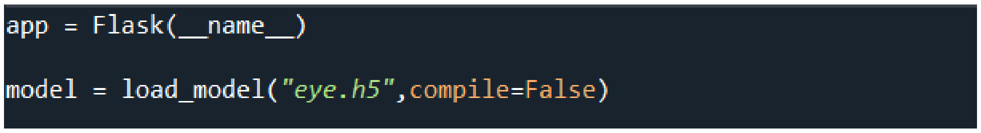
The first step is usually importing the libraries that will be needed in the program.

Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

Task 2: Creating our flask application and loading our model by using load_model method



```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
```

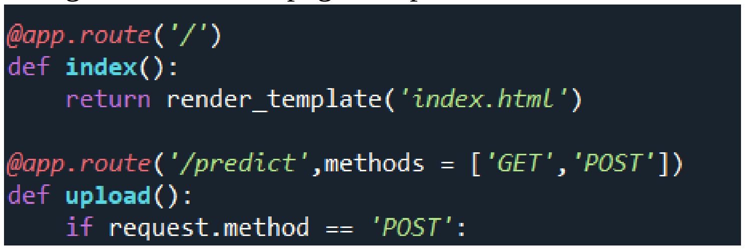


```
app = Flask(__name__)

model = load_model("eye.h5",compile=False)
```

Task 3: Routing to the html Page

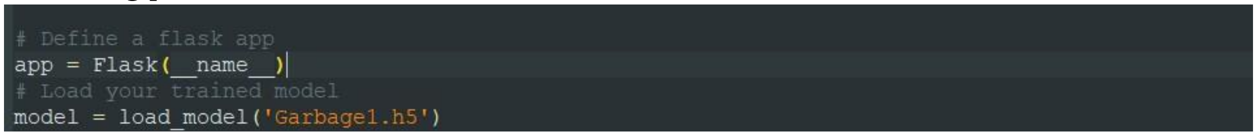
Here, the declared constructor is used to route to the HTML page created earlier.

In the above example, '/' URL is bound with index.html function. Hence, when the home page of a web server is opened in the browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accessed through POST or GET Method.

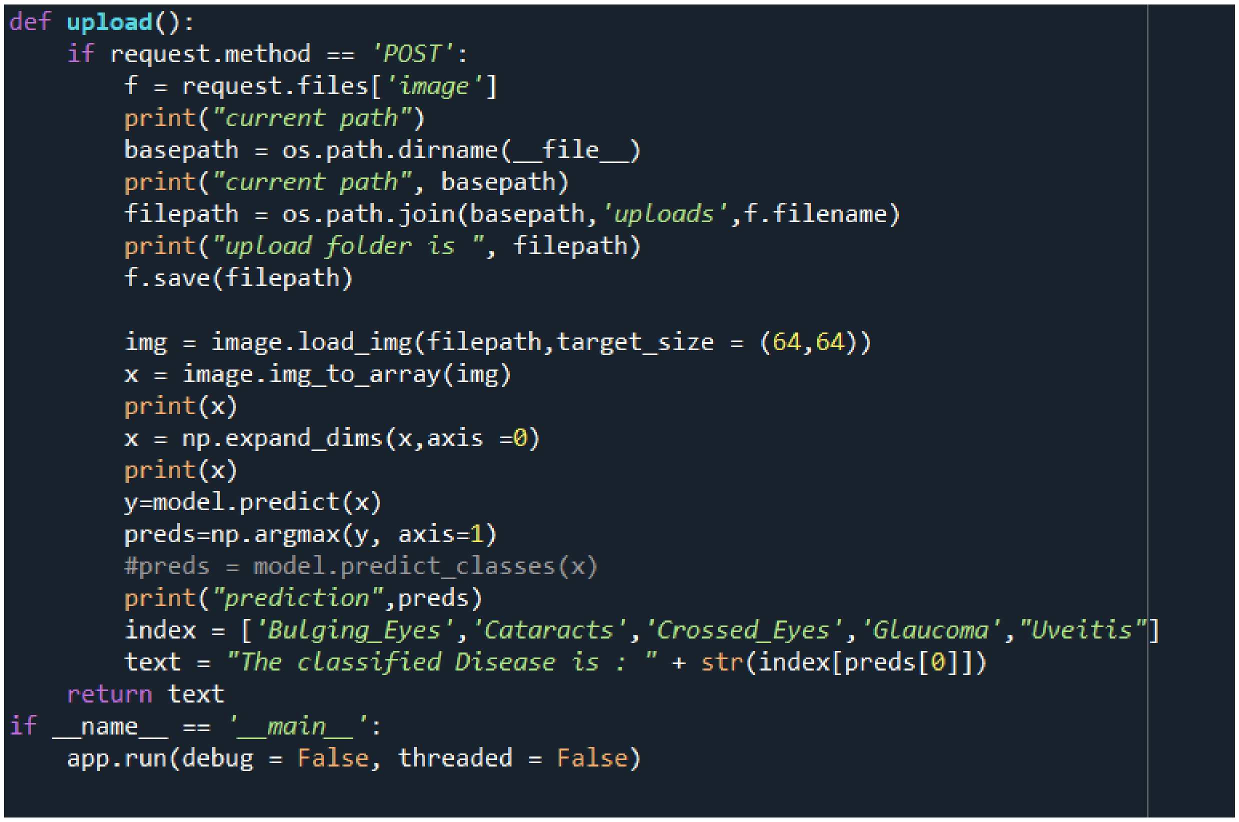


```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict',methods = ['GET','POST'])
def upload():
    if request.method == 'POST':
```

Showcasing prediction on UI:



```
# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

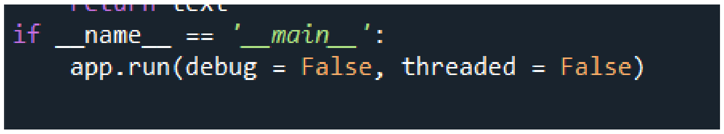


```
def upload():
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath,'uploads',f.filename)
        print("upload folder is ", filepath)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

        img = image.load_img(filepath,target_size = (64,64))
        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)
        #preds = model.predict_classes(x)
        print("prediction",preds)
        index = ['Bulging_Eyes','Cataracts','Crossed_Eyes','Glaucoma',"Uveitis"]
        text = "The classified Disease is : " + str(index[preds[0]])
    return text
if __name__ == '__main__':
    app.run(debug = False, threaded = False)
```

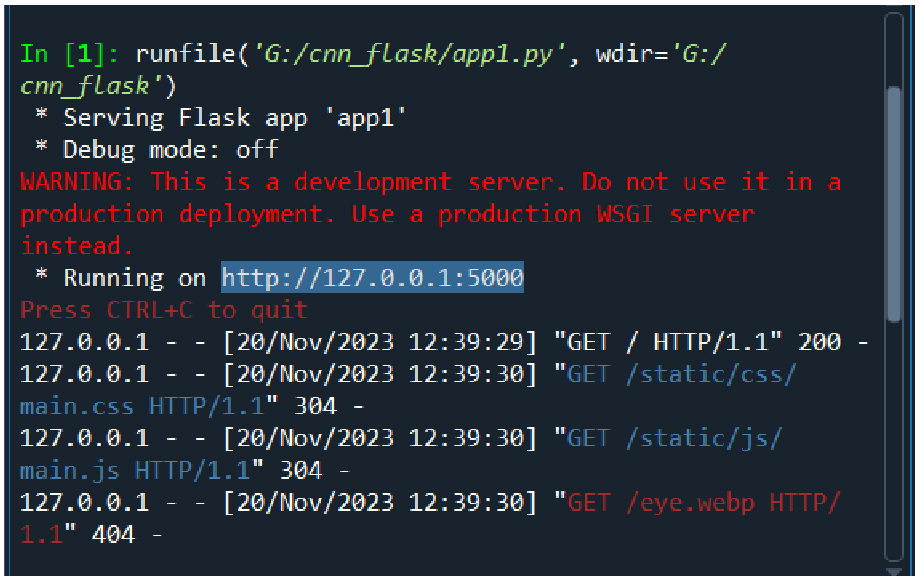
Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directoryusing OS library. Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numericalvalue of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable preds. This numericalvalue is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in the html page

Finally, Run the application

This is used to run the application in a local host.

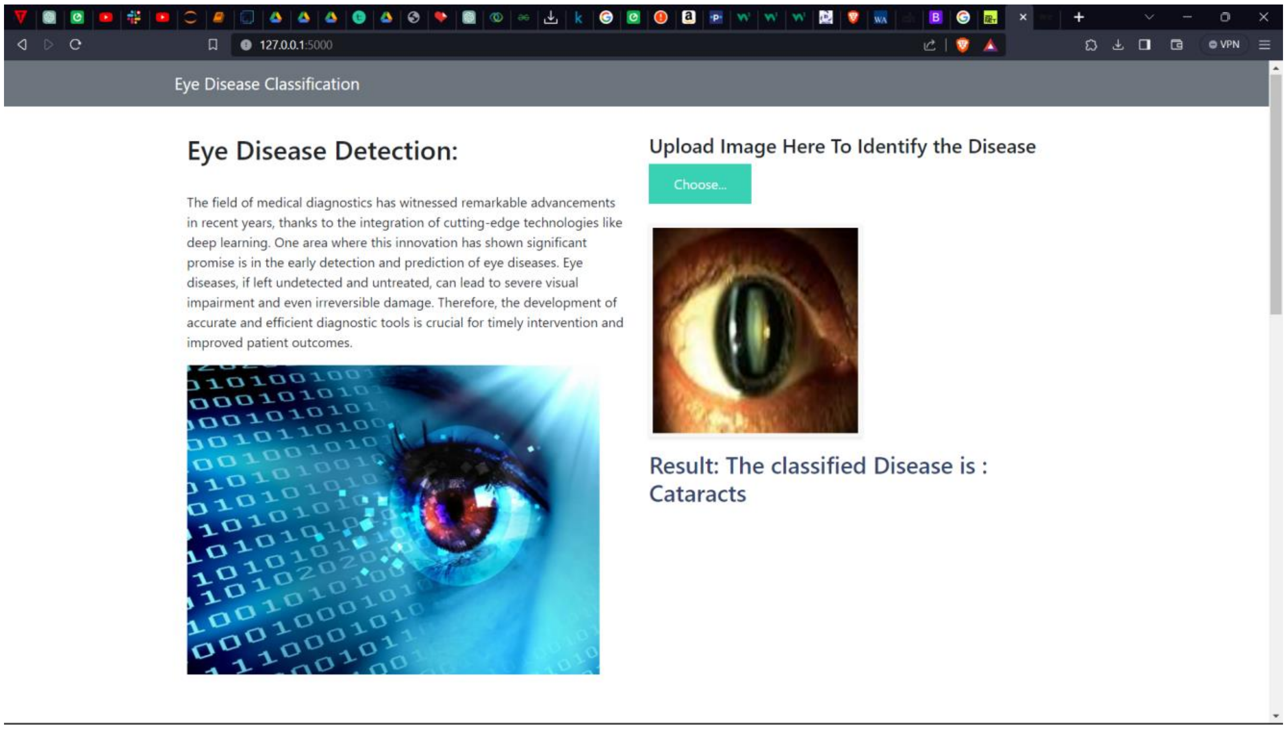


```
if __name__ == '__main__':
    app.run(debug = False, threaded = False)
```



```
In [1]: runfile('G:/cnn_flask/app1.py', wdir='G:/
cnn_flask')
 * 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 - - [20/Nov/2023 12:39:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Nov/2023 12:39:30] "GET /static/css/
main.css HTTP/1.1" 304 -
127.0.0.1 - - [20/Nov/2023 12:39:30] "GET /static/js/
main.js HTTP/1.1" 304 -
127.0.0.1 - - [20/Nov/2023 12:39:30] "GET /eye.webp HTTP/
1.1" 404 -
```

FINAL OUTPUT:

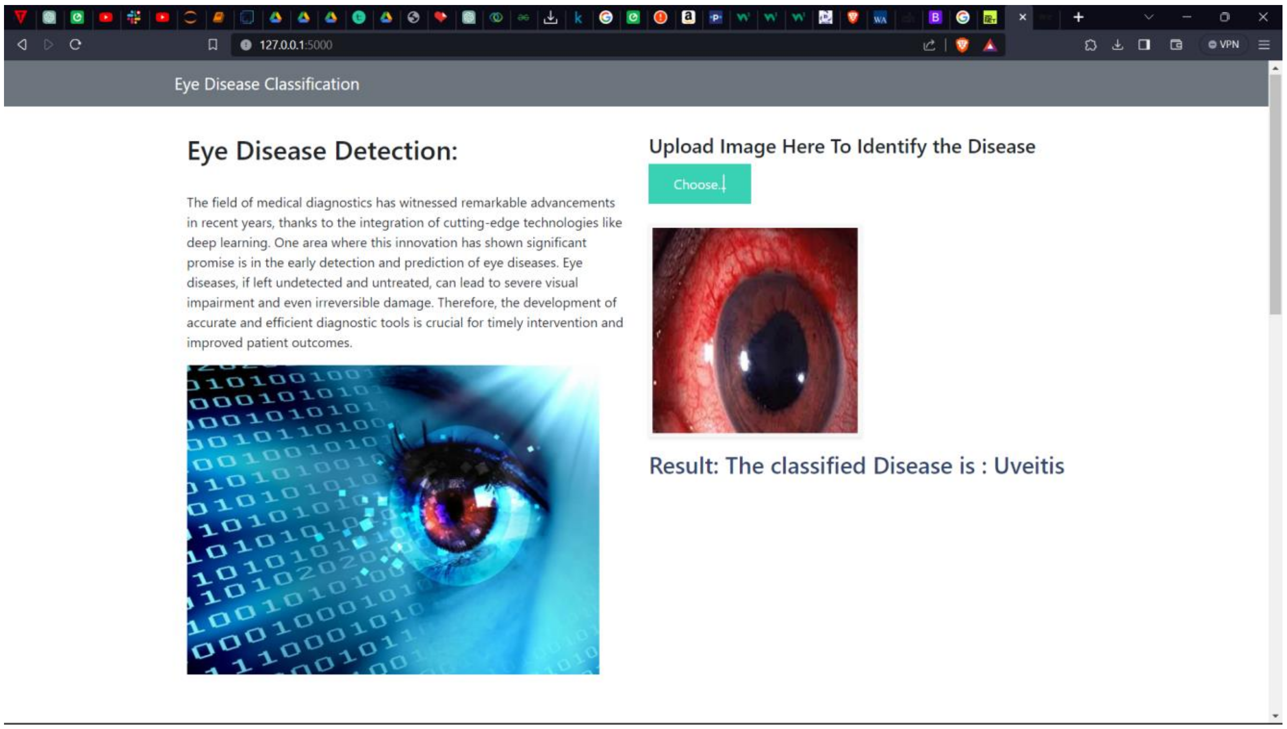


Eye Disease Detection:

The field of medical diagnostics has witnessed remarkable advancements in recent years, thanks to the integration of cutting-edge technologies like deep learning. One area where this innovation has shown significant promise is in the early detection and prediction of eye diseases. Eye diseases, if left undetected and untreated, can lead to severe visual impairment and even irreversible damage. Therefore, the development of accurate and efficient diagnostic tools is crucial for timely intervention and improved patient outcomes.

Upload Image Here To Identify the Disease

Result: The classified Disease is : Cataracts



Eye Disease Detection:

The field of medical diagnostics has witnessed remarkable advancements in recent years, thanks to the integration of cutting-edge technologies like deep learning. One area where this innovation has shown significant promise is in the early detection and prediction of eye diseases. Eye diseases, if left undetected and untreated, can lead to severe visual impairment and even irreversible damage. Therefore, the development of accurate and efficient diagnostic tools is crucial for timely intervention and improved patient outcomes.

Upload Image Here To Identify the Disease

Result: The classified Disease is : Uveitis