Project Development Phase
Project Manual

Date	10 November 2022
Team ID	609691
Project Name	Deep Learning Model For Eye Disease Prediction
Maximum Marks	10 Marks

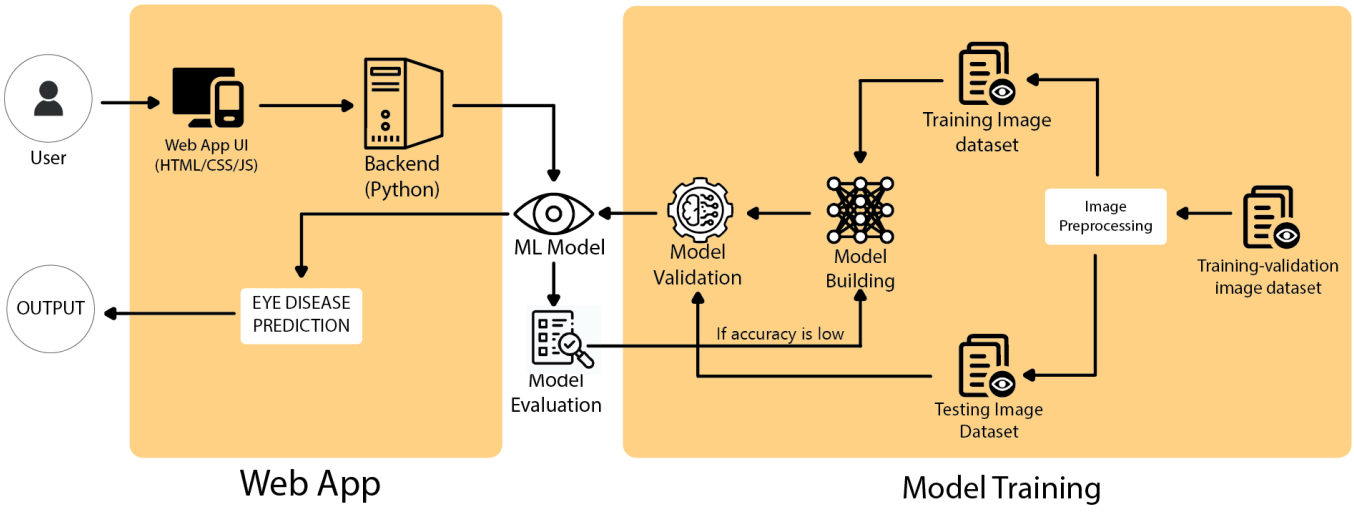
Deep Learning Model For Eye Disease Prediction

Introduction:

Doctors face difficulties in early identification of eye disorders through fundus images. Manual diagnosis of ocular illnesses is time-consuming, prone to errors, and intricate. Hence, an automated system aided by computer tools is crucial for detecting diverse eye disorders using fundus images. This necessity arises due to advancements in deep learning algorithms, enhancing image classification abilities. This study introduces a targeted ocular detection approach based on deep learning methodologies.

Using Transfer Learning based approach to design a reliable and adaptable model that would accept an image of a patient's retina scan and then detect if the patient belongs to one of the following categories : glaucoma', 'cataract', 'normal', 'diabetic_retinopathy'.

Technical Architecture:



Prerequisites:

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

Anaconda Navigator, Jupyter Notebook, Spyder, Visual Studio Code.

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 and Resnet50
- Gain a broad understanding of image data.
- Know how to pre-process/clean image data and use it to build a ML model.
- 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/CSS/JS based webpage
    - Build Python Script for handing backend.

1: Importing necessary libraries

Importing all the libraries that are necessary for building, training, testing and validating the machine learning model for our eye disease detection project.

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
```

2: Data Collection

Data regarding various eye diseases is needed to train the ML model. In our case the image dataset used is from kaggle and is classified into 4 categories : 'glaucoma', 'cataract', 'normal' and 'diabetic retinopathy'.

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

3: Creating the dataset:

After downloading the dataset, the next step is to prepare the dataframe on which the CNN model would be built.

```
# Path definition for different eye disease categories
glaucoma = Path("C:/Users/harsh/Desktop/kaggle/dataset/glaucoma")
cataract = Path("C:/Users/harsh/Desktop/kaggle/dataset/cataract")
normal = Path("C:/Users/harsh/Desktop/kaggle/dataset/normal")
diabetic_retinopathy = Path("C:/Users/harsh/Desktop/kaggle/dataset/diabetic_retinopathy")
```

```
# Create a DataFrame 'df' with file paths and labels for images
disease_type = [glaucoma, cataract,normal,diabetic_retinopathy]
df = pd.DataFrame()
from tqdm import tqdm
for types in disease_type:
    for imagepath in tqdm(list(types.iterdir()), desc= str(types)):
        df = pd.concat([df, pd.DataFrame({'image': [str(imagepath)],'disease_type': [disease_type.index(types)]})], ignore_index
```

```
C:\Users\harsh\Desktop\kaggle\dataset\glaucoma: 100%|██████████| 1007/1007 [00:00<00:00, 2924.67it/s]
C:\Users\harsh\Desktop\kaggle\dataset\cataract: 100%|██████████| 1038/1038 [00:00<00:00, 2791.09it/s]
C:\Users\harsh\Desktop\kaggle\dataset\normal: 100%|██████████| 1074/1074 [00:00<00:00, 2757.63it/s]
C:\Users\harsh\Desktop\kaggle\dataset\diabetic_retinopathy: 100%|██████████| 1098/1098 [00:00<00:00, 3012.81it/s]
```

Plotting the images with labels for image data visualization.

```python
# Function to plot sample images for each disease type
def plot_image(n, num_samples=3):
    disease_labels = ['glaucoma', 'cataract', 'normal', 'diabetic_retinopathy']
    images = df[df['disease_type'] == n].sample(num_samples)['image']

    plt.figure(figsize=(12, 12))

    for i, path in enumerate(images, 1):
        img = (plt.imread(path) - plt.imread(path).min()) / plt.imread(path).max()
        plt.subplot(3, 3, i)
        plt.imshow(img)
        plt.axis('off')
        plt.title(disease_labels[n])

    plt.show()
```

```python
# Plotting sample images for each disease type
plot_image(0)
```

5: Importing necessary libraries for building the resnet model.

```python
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
```

6: Image Preprocessing

The next step is to improve the image data by suppressing the unwilling distortions and enhancing some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

a) 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.

```
# Defining ImageDataGenerator for data preprocessing
datagen = ImageDataGenerator(preprocessing_function=preprocess_input,validation_split=0.2)
```

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

b) 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 'glaucoma', 'cataract', 'normal' and 'diabetic retinopathy'.

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).

```
# Generating train and validation data using flow_from_dataframe
train_data = datagen.flow_from_dataframe(dataframe=df1, x_col ='image',y_col = 'disease_type',target_size=(224,224),class_mode =
```

```
Found 3374 validated image filenames belonging to 4 classes.
```

```
valid_data = datagen.flow_from_dataframe(dataframe=df1, x_col ='image', y_col = 'disease_type', target_size=(224,224), class_mode
```

```
Found 843 validated image filenames belonging to 4 classes.
```

We notice that 2527 images belong to 6 classes for training and 782 images belong to 6 classes for testing purposes.

7) 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.

a) Initializing the model and adding CNN layers

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.

- As the input image contains three channels, we are specifying the input shape as (224,224,3).

- 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.

b) Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used 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.

```
labels=[key for key in train_data.class_indices]
num_classes = len(disease_type)

# Defining ResNet50 model architecture
model = keras.Sequential([
    layers.Rescaling(1./255, input_shape=(224,224, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes,activation='softmax')
])
```

c) 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

```
# Compiling the model and displaying summary
model.compile(optimizer='adam', loss=tf.keras.losses.categorical_crossentropy, metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 224, 224, 16)      448

 max_pooling2d (MaxPooling2  (None, 112, 112, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 128)               6422656

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 6446756 (24.59 MB)
Trainable params: 6446756 (24.59 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

d) Train The model

Now, let us train our model with our image dataset. The model is trained for 30 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 30 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.

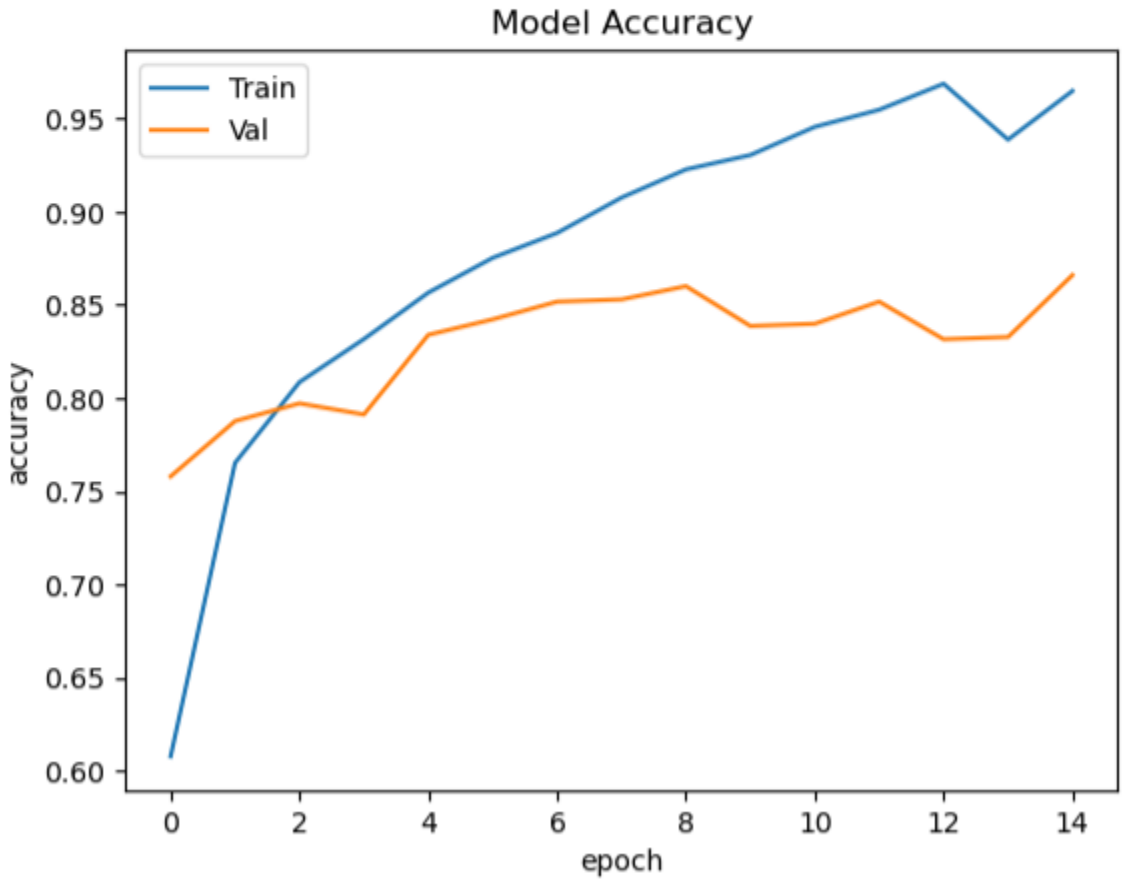
```
# Fitting the model on training and validation data
his = model.fit(train_data, validation_data=valid_data, epochs=15)
```

```
Epoch 1/15
106/106 [==============================] - 110s 1s/step - loss: 0.9138 - accuracy: 0.6079 - val_loss: 0.6640 - val_accuracy: 0.
7580
Epoch 2/15
106/106 [==============================] - 77s 726ms/step - loss: 0.5845 - accuracy: 0.7653 - val_loss: 0.5673 - val_accuracy:
0.7877
Epoch 3/15
106/106 [==============================] - 76s 717ms/step - loss: 0.4858 - accuracy: 0.8085 - val_loss: 0.5356 - val_accuracy:
0.7972
Epoch 4/15
106/106 [==============================] - 74s 697ms/step - loss: 0.4202 - accuracy: 0.8317 - val_loss: 0.5024 - val_accuracy:
0.7912
Epoch 5/15
106/106 [==============================] - 75s 704ms/step - loss: 0.3676 - accuracy: 0.8566 - val_loss: 0.4381 - val_accuracy:
0.8339
Epoch 6/15
106/106 [==============================] - 73s 686ms/step - loss: 0.3090 - accuracy: 0.8752 - val_loss: 0.4101 - val_accuracy:
0.8422
Epoch 7/15
106/106 [==============================] - 72s 676ms/step - loss: 0.2777 - accuracy: 0.8886 - val_loss: 0.4151 - val_accuracy:
0.8517
Epoch 8/15
106/106 [==============================] - 72s 673ms/step - loss: 0.2212 - accuracy: 0.9075 - val_loss: 0.4141 - val_accuracy:
0.8529
Epoch 9/15
106/106 [==============================] - 71s 672ms/step - loss: 0.1969 - accuracy: 0.9226 - val_loss: 0.4268 - val_accuracy:
0.8600
Epoch 10/15
106/106 [==============================] - 72s 674ms/step - loss: 0.1770 - accuracy: 0.9303 - val_loss: 0.4734 - val_accuracy:
0.8387
Epoch 11/15
106/106 [==============================] - 72s 677ms/step - loss: 0.1404 - accuracy: 0.9455 - val_loss: 0.5226 - val_accuracy:
0.8399
Epoch 12/15
106/106 [==============================] - 72s 677ms/step - loss: 0.1168 - accuracy: 0.9547 - val_loss: 0.5635 - val_accuracy:
0.8517
Epoch 13/15
106/106 [==============================] - 74s 692ms/step - loss: 0.0962 - accuracy: 0.9686 - val_loss: 0.7077 - val_accuracy:
0.8316
Epoch 14/15
106/106 [==============================] - 72s 677ms/step - loss: 0.1678 - accuracy: 0.9386 - val_loss: 0.5145 - val_accuracy:
0.8327
Epoch 15/15
106/106 [==============================] - 73s 687ms/step - loss: 0.0924 - accuracy: 0.9647 - val_loss: 0.5919 - val_accuracy:
0.8660
```

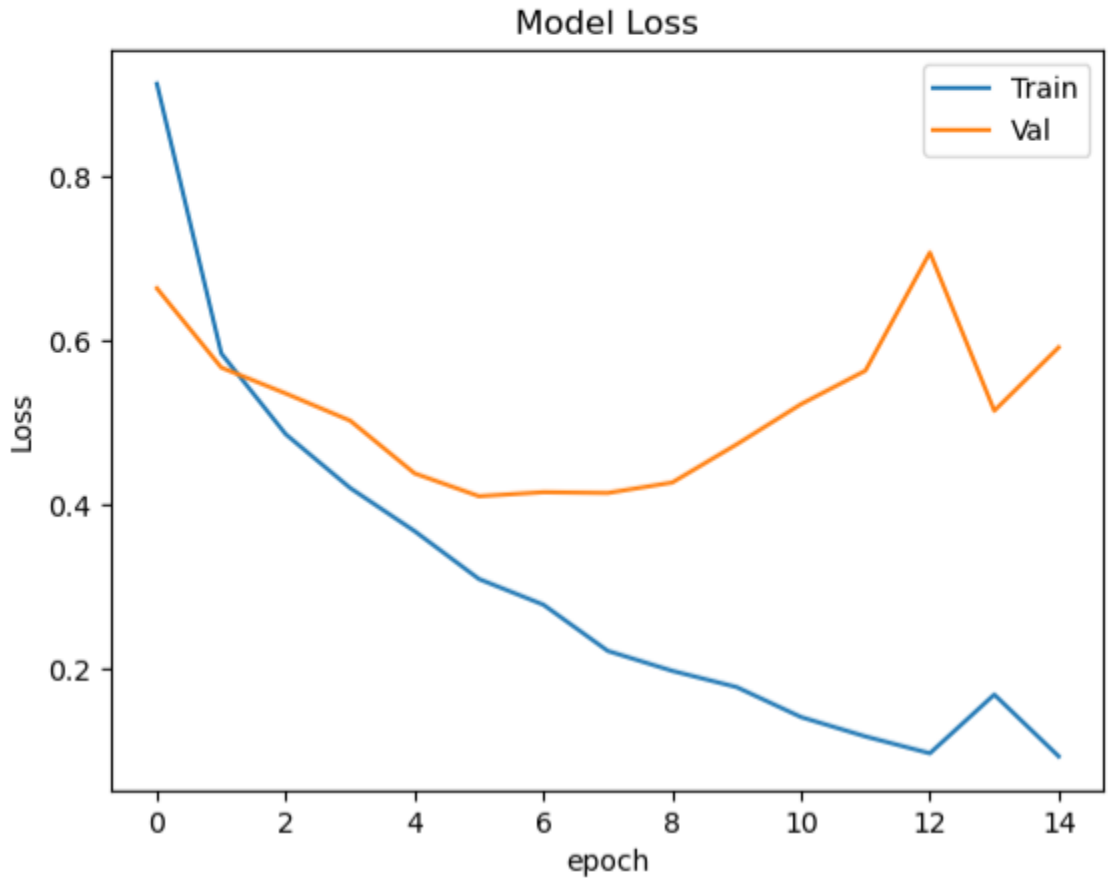
e) Evaluating The model accuracy

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

```python
# Plotting model accuracy and loss over epochs
plt.plot(his.history['accuracy'])
plt.plot(his.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Val'])
plt.show()
```



```
plt.plot(his.history['loss'])
plt.plot(his.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Val'])
plt.show()
```



The

```
print(classification_report(y_test,y_pred,target_names = labels))
```

	precision	recall	f1-score	support
cataract	0.91	0.83	0.87	221
diabetic_retinopathy	0.97	1.00	0.98	215
glaucoma	0.85	0.72	0.78	194
normal	0.75	0.90	0.82	213
accuracy			0.87	843
macro avg	0.87	0.86	0.86	843
weighted avg	0.87	0.87	0.87	843

f) Save the Model

The model is saved with .h5 extension as follows:

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

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

8) 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.

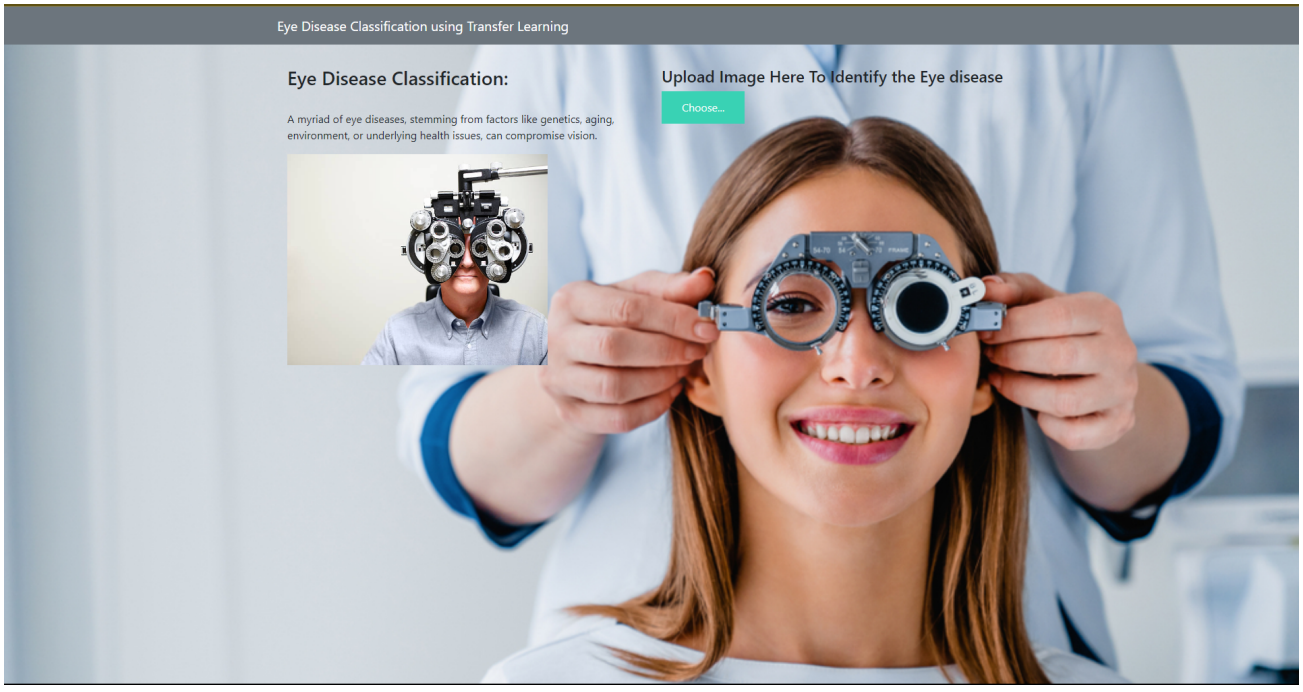
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the "Image" button, the next page is opened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

a) Create HTML - CSS -JS Pages

- We use HTML to create the front end part of the web page.
- Intro.html displays an introduction about the project
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

○ **Link :**CSS , JS

index.html looks like this:



b) Build python code

Task 1: Importing Libraries

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

```
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
#from werkzeug.utils import secure_filename
#from gevent.pywsgi import WSGIServer
```

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

```
app = Flask(__name__, template_folder='template', static_folder='static')
model = load_model(r'C:/Users/harsh/Desktop/Team609691_Eye_Disease_Prediction/Flask_App/eye_disease_detection_model.h5',compile=False)
```


```
app = Flask(__name__, template_folder='template', static_folder='static')
model = load_model(r'C:/Users/harsh/Desktop/Team609691_Eye_Disease_Prediction/Fl
```


Task 3: Routing to the html Page

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

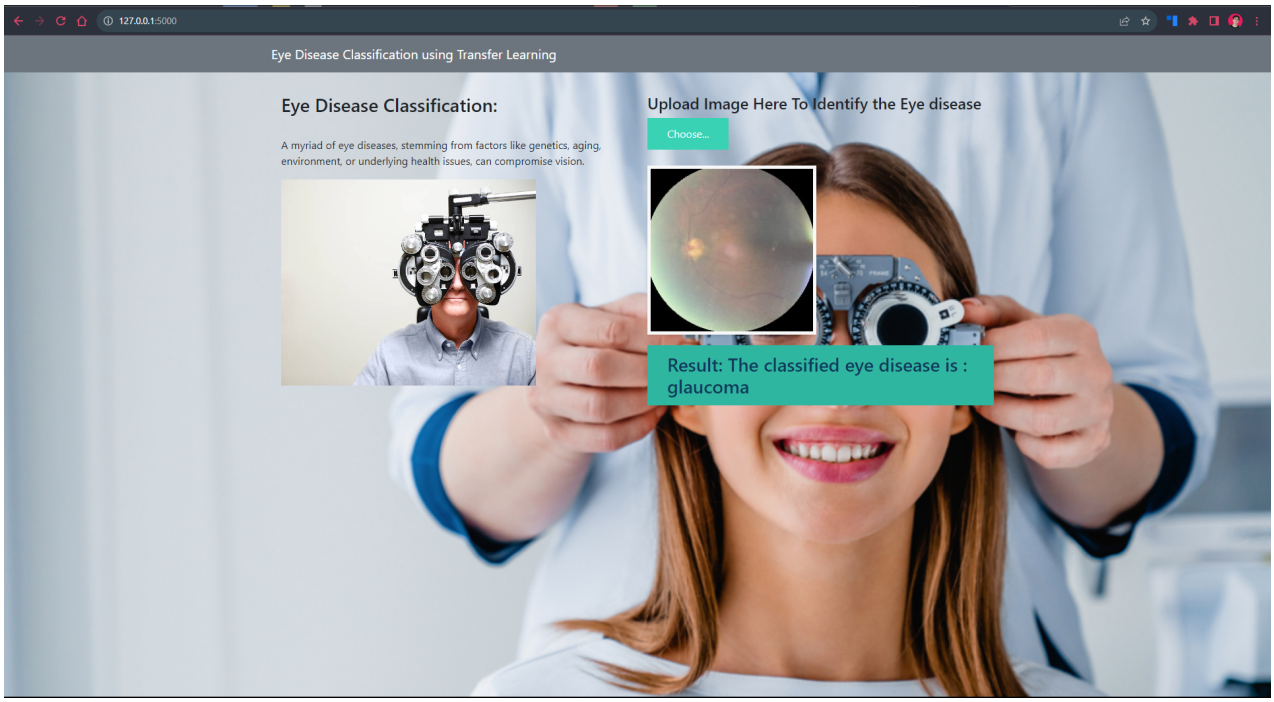
```
@app.route('/')
def index():
    return render_template("/index.html")
    #return "Hello Harshman!"

@app.route('/predict',methods = ['GET','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 = (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)
        #preds = model.predict_classes(x)
        print("prediction",preds)
        index = ['glaucoma', 'cataract', 'normal', 'diabetic_retinopathy']
        text = "The classified eye disease is : " + str(index[preds[0]])
    return text
if __name__ == '__main__':
    app.run(debug = False, threaded = False)
```

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.

Showcasing prediction on UI:



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 directory using 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 numerical value of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable preds. This numerical value 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.

Predicting the results

We then proceed to detect all type of Garbage in the input image using model.predict function and the result is stored in the result variable.

Finally, Run the application

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

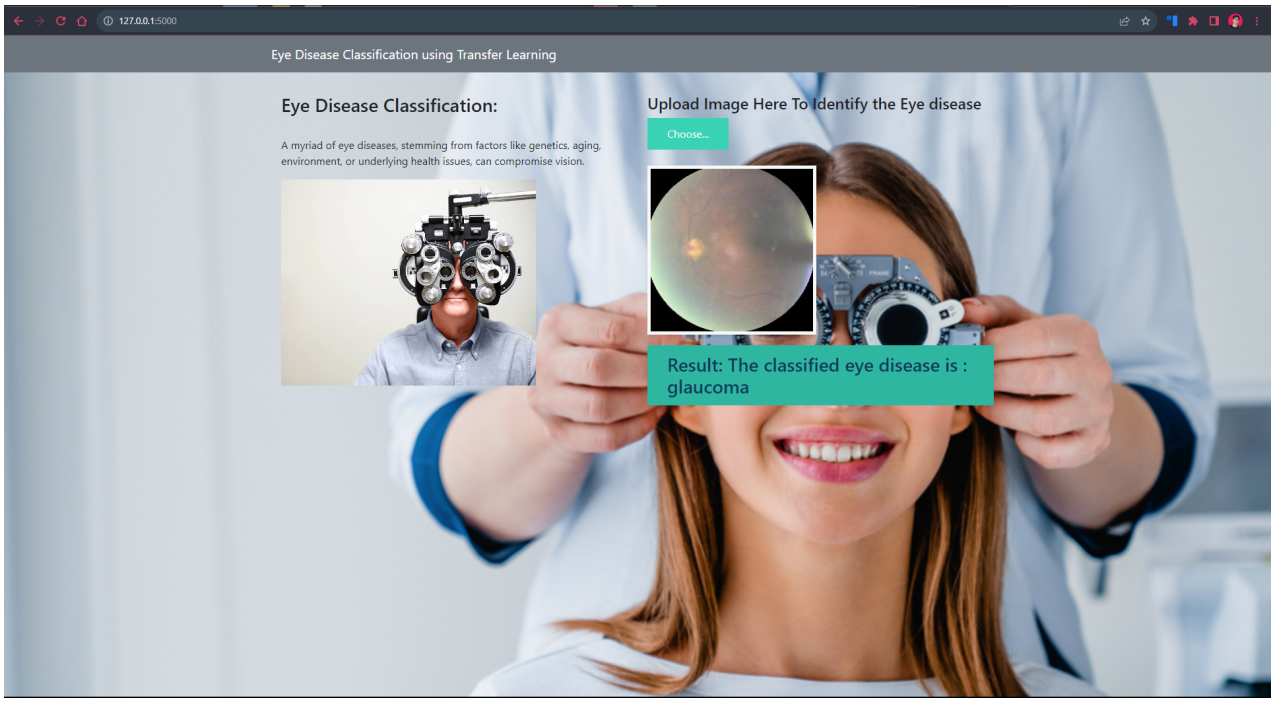
c) Run the application

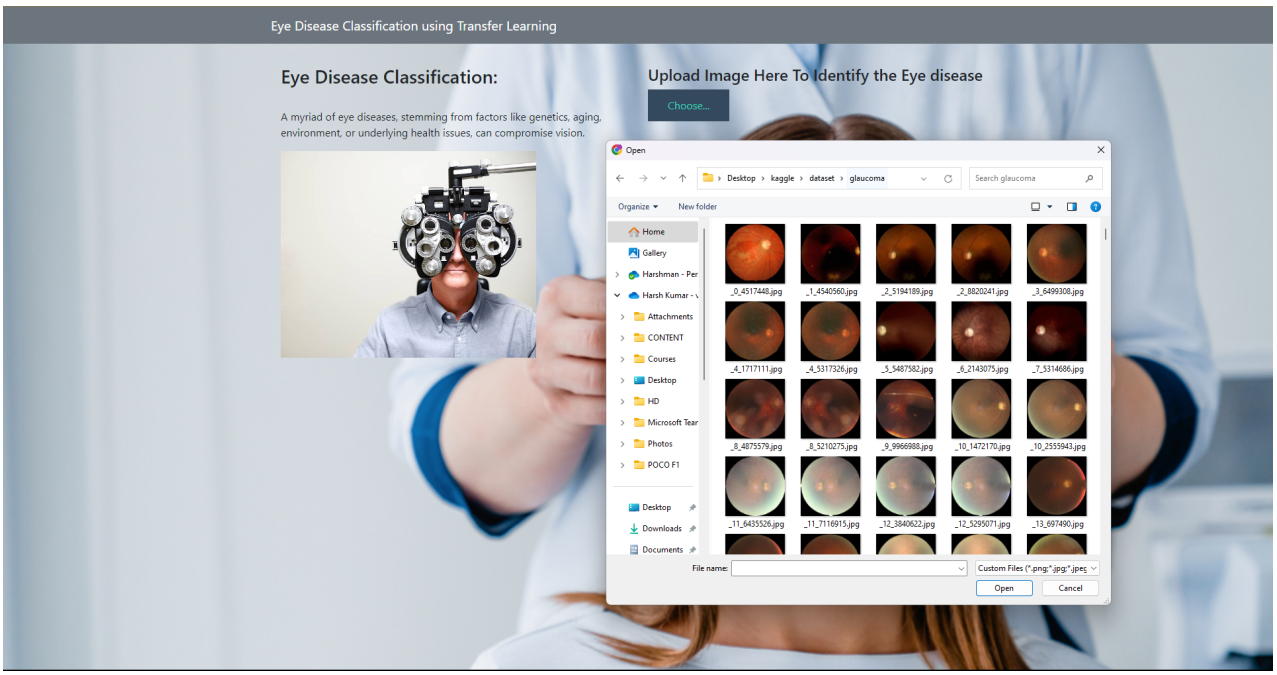
- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type "python app.py" command.
- It will show the local host where your app is running on **http://127.0.0.1.5000/** ● Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

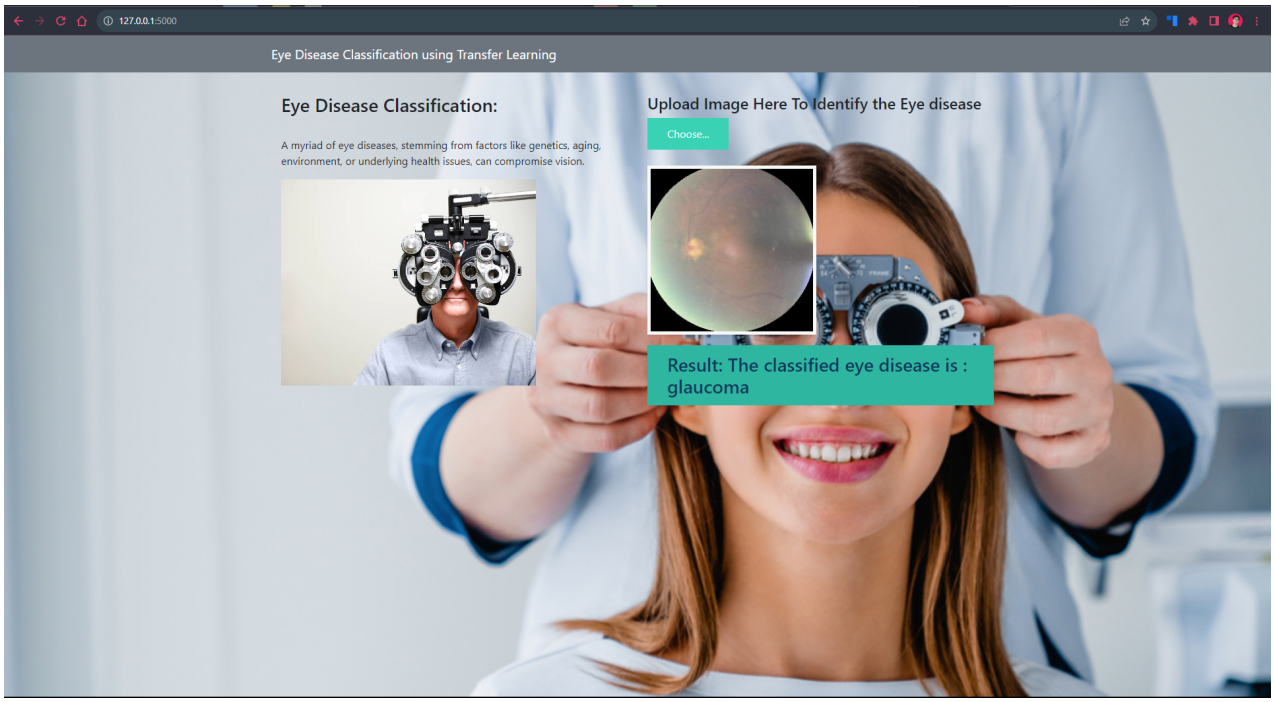
Then it will run on localhost:5000

Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.

FINAL OUTPUT:






127.0.0.1:5000
Eye Disease Classification using Transfer Learning
Eye Disease Classification:
A myriad of eye diseases, stemming from factors like genetics, aging, environment, or underlying health issues, can compromise vision.
Upload Image Here To Identify the Eye disease
Choose...
Result: The classified eye disease is : glaucoma