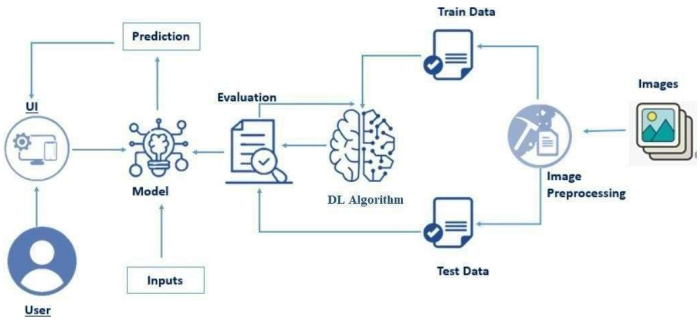
Eye Disease Detection Using Deep Learning

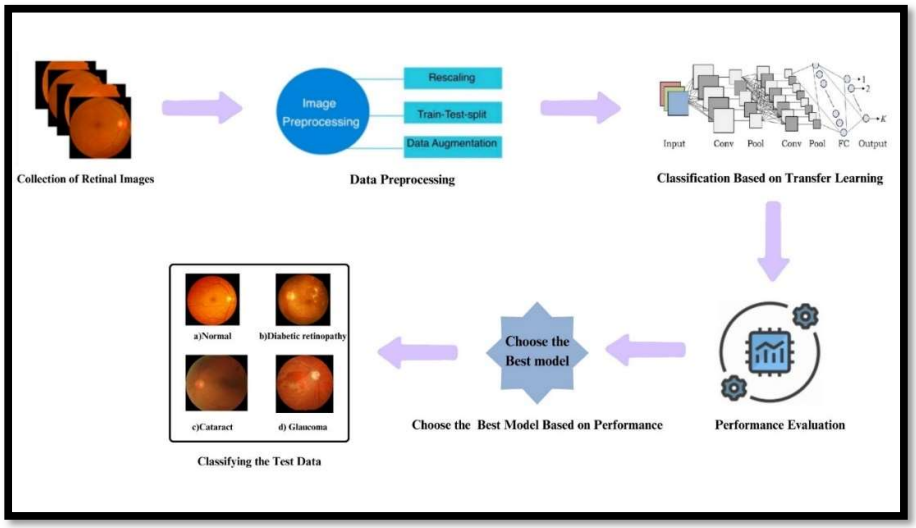
Project Description :

The "Deep Learning Model for Eye Disease Prediction" project is a significant advancement in healthcare and artificial intelligence. This project focuses on classifying eye diseases attributed to factors like age, diabetes, and other health issues into four categories: Normal, Cataract, Diabetic Retinopathy, and Glaucoma. Eye diseases can profoundly affect individuals' lives, potentially leading to vision impairment. Timely and accurate diagnosis is crucial, and this project employs deep learning and transfer learning to provide an efficient solution. Deep learning, a subset of artificial intelligence, excels in high-performance classification tasks, especially with image data. In this project, deep learning models automatically identify and classify eye diseases from medical images, benefitting both healthcare professionals and patients. A notable innovation is the use of transfer learning, known for its performance in various domains, particularly image analysis. The project utilizes models like Inception V3, VGG19, and ResNet50, with ResNet50 proving the most accurate in classifying eye diseases.

Technical Architecture :



Solution 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 RESNET50 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

- **Data Collection.**
  - Download the Dataset
  - Split the Dataset into Train and Test (for VGG19 & Inception V3)
  - Use image_dataset_from_directory to load and prepare the training and validation data. (for RESNET50)
- **Image Pre-processing (For VGG19 & InceptionV3)**
  - Import the necessary libraries.
  - Configure ImageDataGenerator for data augmentation
  - Apply ImageDataGenerator to the training and validation data.
- **Model Building**
  - Import the required libraries.
  - Define a model with custom output layers.
  - Compile the model, specifying loss, optimizer, and metrics.
  - Train the model using the training and validation data and save the best val_accuracy model using checkpoints
- **Testing the Model**
  - Load the model with best val_accuracy
  - Define a function to make predictions using the loaded model.
  - Load and preprocess test images.
  - Use the predict function to make predictions for each image.
- **Application Building**
  - Create an HTML file
  - Create a Flask application and integrate the HTML file

Prior Knowledge:

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

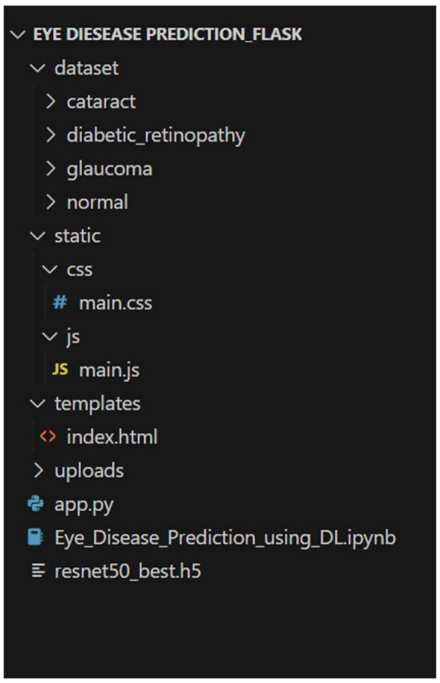
- *Deep Learning Concepts*
  - **CNN:** https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
  - ***VGG19***: VGG-19 convolutional neural network - MATLAB vgg19 - MathWorks India
  - **ResNet-50V2:**

    https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
  - **Inception-V3:** https://iq.opengenus.org/inception-v3-model-architecture/

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

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

Project Structure:



- The dataset directory comprises four folders, each dedicated to a specific eye disease, containing corresponding images
- To create a Flask application, you should have HTML pages stored in the "templates" folder for rendering, CSS files in the "static" folder to style the pages, JavaScript for client-side functionality, and a Python script named "app.py" for server-side scripting.
- The Eye_Disease_prediction_using_DL.ipynb file is the notebook in which the model is trained
- The resnet50_best.h5 is the saved model which has the best accuracy

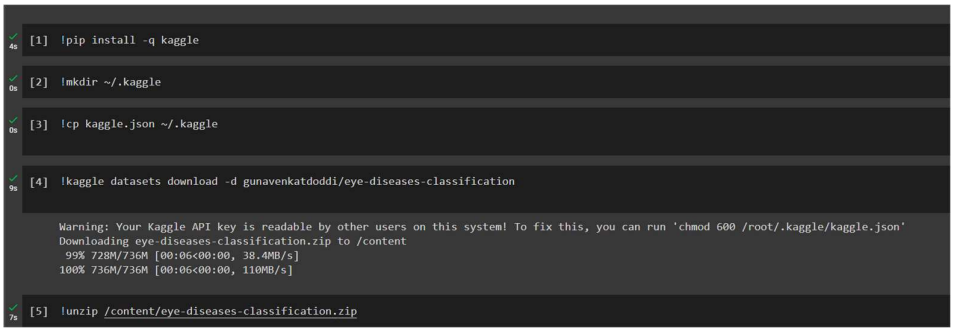
Milestone 1: Data Collection

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

We download the dataset from Kaggle by installing the Kaggle package, setting up our Kaggle API credentials, downloading the dataset, and unzipping it .



Activity 2 : Split the Dataset into Train and Test(for VGG19 & Inception V3)

To build VGG19 model with good accuracy we need to split training and testing data into two separate folders as the dataset folder does not contain separate training and testing .

```
!pip install split-folders
import splitfolders
splitfolders.ratio('/content/dataset',output='data',seed=1337,ratio=(0.8,0.2))
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
Copying files: 4217 files [00:02, 1648.10 files/s]
```

Activity 3: Use image_dataset_from_directory to load and prepare the training and validation data (For RESNET 50)

We used the image_dataset_from_directory function to load and Prepare the training and Validation Data, this code prepares the data that will be used to train and validate the RESNET50 model, ensuring it is correctly formatted and ready for training.

```
import tensorflow as tf
from tensorflow import keras


training_data = keras.preprocessing.image_dataset_from_directory(
    '/content/dataset',
    batch_size = 70,
    image_size =(224,224),

    shuffle = True,
    seed =123,
    subset ='training',
    validation_split=0.15,
    )
validation_data =keras.preprocessing.image_dataset_from_directory(
    '/content/dataset',
    batch_size = 70,
    image_size =(224,224),

    shuffle = True,
    seed =123,
    validation_split =0.15,
    subset ='validation',
    )
```

```
Found 4217 files belonging to 4 classes.
Using 3585 files for training.
Found 4217 files belonging to 4 classes.
Using 632 files for validation.
```

Milestone 2 : Image Pre-processing (For VGG19 & InceptionV3)

Activity 1: Configure ImageDataGenerator class

The ImageDataGenerator class is used to generate augmented images for training. It performs several data augmentation techniques to increase the diversity and robustness of the training data.

The specific augmentation techniques configured in ImageDataGenerator class are as follows:

1. Normalization: Rescaling pixel values to the range [0, 1] for numerical stability.

2. Shear transformations: Images can be sheared within a 20% range, introducing slant to the images.

3. Zooming: Random zooming in or out of the images within a 20% range.

4. Horizontal flipping: Images can be flipped horizontally to add variation.

5. Validation split: A portion (20%) of the data is reserved for validation

```
[9]  train_datagen = ImageDataGenerator(
         rescale=1. / 255,
         shear_range=0.2,
         zoom_range=0.2,
         horizontal_flip=True,
         validation_split=0.2
     )

     valid_datagen = ImageDataGenerator(rescale=1.0 / 255)
```

Activity 2: Apply ImageDataGenerator to the training and validation data.

The code utilizes the previously configured ImageDataGenerator instances to generate batches of training and validation data. These data generators are essential for feeding data to a deep learning model during training and validation, applying the previously defined data augmentation techniques.

```
✓ 0s [12] train_data = train_datagen.flow_from_directory(
          directory='/content/data/train',
          target_size=image_size,
          batch_size=batch_size,
          class_mode='categorical',
      )

      validation_data = valid_datagen.flow_from_directory(
          directory='/content/data/val',
          target_size=image_size,
          batch_size=batch_size,
          class_mode='categorical',
        )

     Found 3372 images belonging to 4 classes.
     Found 845 images belonging to 4 classes.
```

Milestone 3 : Model Building

Activity 1: Import the required libraries.

VGG19 :

```
[8]  from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.applications import VGG19
     from tensorflow.keras.models import Model
     from tensorflow.keras.layers import Dense, Flatten, BatchNormalization,Dropout
     from tensorflow.keras.callbacks import ModelCheckpoint
     from tensorflow.keras.optimizers import Adam
```

RESNET 50 :

```
[ ]  import tensorflow as tf
     import numpy as np
     from tensorflow import keras
     from tensorflow.keras.preprocessing import image
     from tensorflow.keras.applications.resnet50 import ResNet50
     from tensorflow.keras.layers import Dense, Flatten, BatchNormalization,Dropout
     from tensorflow.keras.models import Model,Sequential
     from tensorflow.keras.callbacks import ModelCheckpoint
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Inception V3 :

```
[ ]  from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.models import Model
     from tensorflow.keras.layers import Dense, Flatten, BatchNormalization,Dropout
     from tensorflow.keras.callbacks import ModelCheckpoint
     from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.applications.inception_v3 import InceptionV3
     import tensorflow as tf
```

Activity 2: Define a model with custom output layers.

Here we are creating three different models (VGG19, ResNet50, and Inception V3) with custom output layers, including the freezing of pre-trained layers and adding new layers for the specific classification task.

VGG19 :

```
[ ] vgg = VGG19(
        include_top=False,
        weights="imagenet",
        input_shape=(240,240, 3),
    )

    for layer in vgg.layers:
        layer.trainable = False

    # Add custom layers
    x = vgg.output
    x = Flatten()(x)

    x = BatchNormalization()(x)
    x= Dense(512, activation='relu')(x)
    predictions = Dense(4, activation='softmax')(x)  # 4 categories

    # Create new model
    model = Model(inputs=vgg.input, outputs=predictions)

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
    80134624/80134624 [==============================] - 4s 0us/step
```

RESNET 50 :

```
[ ]  resnet_model = Sequential()
     pretrained_model= ResNet50(include_top=False,
                        input_shape=(224,224,3),
                        pooling='avg',
                        weights='imagenet')
     for layer in pretrained_model.layers:
       layer.trainable=False
     resnet_model.add(pretrained_model)
     resnet_model.add(Flatten())
     resnet_model.add(BatchNormalization())
     resnet_model.add(Dense(512, activation='relu'))
     resnet_model.add(BatchNormalization())
     resnet_model.add(Dense(4, activation='softmax'))
```

Inception V3 :

```
inception_model = InceptionV3(input_shape=(224, 224, 3),
                      include_top=False,
                      weights='imagenet')

for layer in inception_model.layers:
    layer.trainable = False

# Add custom layers
inception_x= inception_model.output

x = Flatten()(inception_x)
x=BatchNormalization()(x)
x= Dense(4, activation='softmax')(x)  # 4 categories

# Create new model
inception_model = Model(inputs=inception_model.input, outputs=x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 3s 0us/step
```

Activity 3: Compile the model, specifying loss, optimizer, and metrics.

The code compiles various models with their corresponding loss function, utilizing the 'adam' optimizer for efficient parameter adjustment. The 'accuracy' metric assesses model performance during training and validation, facilitating their use in multi-class classification tasks.

VGG19 :

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

RESNET 50 :

```
[ ]  resnet_model.compile(
          loss='sparse_categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy']
     )
```

Inception V3 :

```
[ ]  # Compile the model
     inception_model.compile(
         optimizer=Adam(learning_rate=0.0001),
         metrics=['accuracy'],
         loss='categorical_crossentropy'
     )
```

Activity 4: Train the model using the training and validation data and save the best val_accuracy model using checkpoints

Now, let us train our model with our image dataset. The model is trained for certain number of epochs and after every epoch, the current model state is saved if the model has the least val_accuracy encountered till that time using callbacks. Here we used callback in RESNET 50 and Commented the callbacks method **.fit** function is used to train a deep learning neural network

Arguments:

1. **Training Epochs:** The training process is organized into epochs, where each epoch corresponds to one complete pass through the entire training dataset. In the case of VGG19, it's trained for 20 epochs, ResNet50 for 50 epochs, and Inception V3 for 10 epochs. This allows the models to learn from the data over multiple iterations.
2. **Batch Size:** During each epoch, data is divided into batches, and each batch contains a fixed number of data samples. The batch size is a critical hyperparameter that influences training speed and memory usage. In the provided code, VGG19 uses a batch size of 100, ResNet50 uses a batch size of 70, and Inception V3 uses a batch size of 100.
3. **Steps per Epoch:** The parameter **steps_per_epoch** determines how many batches of data are processed within one epoch. It is calculated based on the total number of training samples divided by the batch size. This controls the number of weight updates that occur within an epoch.
4. **Validation Steps:** In the case of generator-based validation data (for ResNet50 and Inception V3), the parameter **validation_steps** are used to specify how many validation data batches are processed for evaluation during each epoch.
5. **Callbacks:** callbacks are a crucial component of the training process. Callbacks are functions that are executed at specific points during training. They can be used to save model checkpoints, monitor training progress, and make early stopping decisions.

VGG 19 :

```
[ ] model.fit(train_data,
                    validation_data=validation_data,
                    epochs=20,
                    batch_size=100,
                    steps_per_epoch=len(train_data),
                    validation_steps=len(validation_data),
                    #callbacks=[model_checkpoint]
                    )
```

```
Epoch 1/20
53/53 [==============================] - 85s 1s/step - loss: 4.5195 - accuracy: 0.6862 - val_loss: 2.9883 - val_accuracy: 0.4982
Epoch 2/20
53/53 [==============================] - 67s 1s/step - loss: 2.3846 - accuracy: 0.7714 - val_loss: 0.7659 - val_accuracy: 0.7101
Epoch 3/20
53/53 [==============================] - 67s 1s/step - loss: 0.7452 - accuracy: 0.7977 - val_loss: 0.9504 - val_accuracy: 0.5586
Epoch 4/20
53/53 [==============================] - 67s 1s/step - loss: 0.4765 - accuracy: 0.8277 - val_loss: 0.7224 - val_accuracy: 0.6757
Epoch 5/20
53/53 [==============================] - 67s 1s/step - loss: 0.4481 - accuracy: 0.8482 - val_loss: 0.5925 - val_accuracy: 0.7479
Epoch 6/20
53/53 [==============================] - 69s 1s/step - loss: 0.4002 - accuracy: 0.8482 - val_loss: 0.6196 - val_accuracy: 0.7444
Epoch 7/20
53/53 [==============================] - 66s 1s/step - loss: 0.3878 - accuracy: 0.8553 - val_loss: 0.4678 - val_accuracy: 0.8071
Epoch 8/20
53/53 [==============================] - 68s 1s/step - loss: 0.3571 - accuracy: 0.8577 - val_loss: 0.6088 - val_accuracy: 0.7456
Epoch 9/20
53/53 [==============================] - 67s 1s/step - loss: 0.3410 - accuracy: 0.8603 - val_loss: 0.4908 - val_accuracy: 0.8118
Epoch 10/20
53/53 [==============================] - 68s 1s/step - loss: 0.3323 - accuracy: 0.8639 - val_loss: 0.4619 - val_accuracy: 0.8414
Epoch 11/20
53/53 [==============================] - 66s 1s/step - loss: 0.3281 - accuracy: 0.8740 - val_loss: 0.5981 - val_accuracy: 0.7811
Epoch 12/20
53/53 [==============================] - 68s 1s/step - loss: 0.3294 - accuracy: 0.8760 - val_loss: 0.4671 - val_accuracy: 0.8083
Epoch 13/20
53/53 [==============================] - 67s 1s/step - loss: 0.2899 - accuracy: 0.8855 - val_loss: 0.4269 - val_accuracy: 0.8497
Epoch 14/20
53/53 [==============================] - 67s 1s/step - loss: 0.2915 - accuracy: 0.8909 - val_loss: 0.5027 - val_accuracy: 0.8237
Epoch 15/20
53/53 [==============================] - 68s 1s/step - loss: 0.3141 - accuracy: 0.8731 - val_loss: 0.4746 - val_accuracy: 0.8225
Epoch 16/20
53/53 [==============================] - 67s 1s/step - loss: 0.2755 - accuracy: 0.8923 - val_loss: 0.5161 - val_accuracy: 0.8414
Epoch 17/20
53/53 [==============================] - 67s 1s/step - loss: 0.2818 - accuracy: 0.8977 - val_loss: 0.4015 - val_accuracy: 0.8604
Epoch 18/20
53/53 [==============================] - 67s 1s/step - loss: 0.2576 - accuracy: 0.8968 - val_loss: 0.4498 - val_accuracy: 0.8615
Epoch 19/20
53/53 [==============================] - 68s 1s/step - loss: 0.2498 - accuracy: 0.8992 - val_loss: 0.4242 - val_accuracy: 0.8544
```

RESNET 50 :

```
# Train the model
resnet_model.fit(training_data,
                 validation_data=validation_data,
                 epochs=50,
                 batch_size=70,
                 steps_per_epoch=len(training_data),
                 validation_steps=len(validation_data),
                 callbacks=[model_checkpoint]
)
```

```
Epoch 12/50
52/52 [==============================] - ETA: 0s - loss: 0.0138 - accuracy: 0.9972
Epoch 12: val_accuracy did not improve from 0.92880
52/52 [==============================] - 26s 458ms/step - loss: 0.0138 - accuracy: 0.9972 - val_loss: 0.2355 - val_accuracy: 0.9288
Epoch 13/50
52/52 [==============================] - ETA: 0s - loss: 0.0079 - accuracy: 0.9989
Epoch 13: val_accuracy did not improve from 0.92880
52/52 [==============================] - 27s 471ms/step - loss: 0.0079 - accuracy: 0.9989 - val_loss: 0.2492 - val_accuracy: 0.9225
Epoch 14/50
52/52 [==============================] - ETA: 0s - loss: 0.0129 - accuracy: 0.9972
Epoch 14: val_accuracy did not improve from 0.92880
52/52 [==============================] - 27s 466ms/step - loss: 0.0129 - accuracy: 0.9972 - val_loss: 0.2646 - val_accuracy: 0.9209
Epoch 15/50
52/52 [==============================] - ETA: 0s - loss: 0.0204 - accuracy: 0.9955
Epoch 15: val_accuracy did not improve from 0.92880
52/52 [==============================] - 26s 462ms/step - loss: 0.0204 - accuracy: 0.9955 - val_loss: 0.2757 - val_accuracy: 0.9146
Epoch 16/50
52/52 [==============================] - ETA: 0s - loss: 0.0144 - accuracy: 0.9955
Epoch 16: val_accuracy did not improve from 0.92880
52/52 [==============================] - 27s 455ms/step - loss: 0.0144 - accuracy: 0.9955 - val_loss: 0.2813 - val_accuracy: 0.9272
Epoch 17/50
52/52 [==============================] - ETA: 0s - loss: 0.0301 - accuracy: 0.9908
Epoch 17: val_accuracy did not improve from 0.92880
52/52 [==============================] - 28s 467ms/step - loss: 0.0301 - accuracy: 0.9908 - val_loss: 0.3538 - val_accuracy: 0.9177
Epoch 18/50
52/52 [==============================] - ETA: 0s - loss: 0.0690 - accuracy: 0.9788
Epoch 18: val_accuracy did not improve from 0.92880
52/52 [==============================] - 27s 474ms/step - loss: 0.0690 - accuracy: 0.9788 - val_loss: 0.2835 - val_accuracy: 0.9130
Epoch 19/50
52/52 [==============================] - ETA: 0s - loss: 0.0313 - accuracy: 0.9914
Epoch 19: val_accuracy improved from 0.92880 to 0.93038, saving model to resnet50best_model.h5
52/52 [==============================] - 27s 469ms/step - loss: 0.0313 - accuracy: 0.9914 - val_loss: 0.2706 - val_accuracy: 0.9304
Epoch 20/50
52/52 [==============================] - ETA: 0s - loss: 0.0145 - accuracy: 0.9958
```

Inception V3:

```
[ ] inception_model.fit(train_data,
              validation_data=validation_data,
              epochs=10,
              batch_size=100,
              steps_per_epoch=len(train_data),
              validation_steps=len(validation_data),
              #callbacks=[model_checkpoint]
             )
```

```
Epoch 1/10
53/53 [==============================] - 77s 1s/step - loss: 0.8368 - accuracy: 0.6948 - val_loss: 0.6310 - val_accuracy: 0.7515
Epoch 2/10
53/53 [==============================] - 64s 1s/step - loss: 0.5668 - accuracy: 0.7859 - val_loss: 0.5430 - val_accuracy: 0.8024
Epoch 3/10
53/53 [==============================] - 65s 1s/step - loss: 0.4724 - accuracy: 0.8197 - val_loss: 0.5037 - val_accuracy: 0.8071
Epoch 4/10
53/53 [==============================] - 66s 1s/step - loss: 0.4170 - accuracy: 0.8455 - val_loss: 0.4709 - val_accuracy: 0.8166
Epoch 5/10
53/53 [==============================] - 71s 1s/step - loss: 0.3699 - accuracy: 0.8541 - val_loss: 0.4600 - val_accuracy: 0.8178
Epoch 6/10
53/53 [==============================] - 66s 1s/step - loss: 0.3453 - accuracy: 0.8713 - val_loss: 0.4737 - val_accuracy: 0.8367
Epoch 7/10
53/53 [==============================] - 65s 1s/step - loss: 0.3120 - accuracy: 0.8894 - val_loss: 0.4782 - val_accuracy: 0.8331
Epoch 8/10
53/53 [==============================] - 65s 1s/step - loss: 0.2880 - accuracy: 0.8915 - val_loss: 0.4771 - val_accuracy: 0.8272
Epoch 9/10
53/53 [==============================] - 71s 1s/step - loss: 0.2816 - accuracy: 0.8953 - val_loss: 0.4283 - val_accuracy: 0.8414
Epoch 10/10
53/53 [==============================] - 66s 1s/step - loss: 0.2668 - accuracy: 0.8980 - val_loss: 0.4303 - val_accuracy: 0.8379
<keras.src.callbacks.History at 0x7c2a77483b80>
```

Summary of the models:

VGG19:

```
model.summary()

Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 240, 240, 3)]     0

block1_conv1 (Conv2D)        (None, 240, 240, 64)      1792

block1_conv2 (Conv2D)        (None, 240, 240, 64)      36928

block1_pool (MaxPooling2D)   (None, 120, 120, 64)      0

block2_conv1 (Conv2D)        (None, 120, 120, 128)     73856

block2_conv2 (Conv2D)        (None, 120, 120, 128)     147584

block2_pool (MaxPooling2D)   (None, 60, 60, 128)       0

block3_conv1 (Conv2D)        (None, 60, 60, 256)       295168

block3_conv2 (Conv2D)        (None, 60, 60, 256)       590080

block3_conv3 (Conv2D)        (None, 60, 60, 256)       590080

block3_conv4 (Conv2D)        (None, 60, 60, 256)       590080

block3_pool (MaxPooling2D)   (None, 30, 30, 256)       0

block4_conv1 (Conv2D)        (None, 30, 30, 512)       1180160

block4_conv2 (Conv2D)        (None, 30, 30, 512)       2359808

block4_conv3 (Conv2D)        (None, 30, 30, 512)       2359808

block4_conv4 (Conv2D)        (None, 30, 30, 512)       2359808

block4_pool (MaxPooling2D)   (None, 15, 15, 512)       0

block5_conv1 (Conv2D)        (None, 15, 15, 512)       2359808

block5_conv2 (Conv2D)        (None, 15, 15, 512)       2359808

block5_conv3 (Conv2D)        (None, 15, 15, 512)       2359808

block5_conv4 (Conv2D)        (None, 15, 15, 512)       2359808

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

flatten_2 (Flatten)          (None, 25088)             0

batch_normalization_4 (Bat   (None, 25088)             100352
chNormalization)

dense_5 (Dense)              (None, 512)               12845568

dense_6 (Dense)              (None, 4)                 2052

=================================================================
```

RESNET50 :

```
[ ]  resnet_model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten_3 (Flatten)         (None, 2048)              0

 batch_normalization_98 (Ba  (None, 2048)              8192
 tchNormalization)

 dense_5 (Dense)             (None, 512)               1049088

 batch_normalization_99 (Ba  (None, 512)               2048
 tchNormalization)

 dense_6 (Dense)             (None, 4)                 2052

=================================================================
Total params: 24649092 (94.03 MB)
Trainable params: 1056260 (4.03 MB)
Non-trainable params: 23592832 (90.00 MB)
_________________________________________________________________
```

Inception V3 :

```
[17] inception_model.summary()
```

```
tchNormalization)

batch_normalization_91 (Ba  (None, 5, 5, 384)          1152      ['conv2d_91[0][0]']
tchNormalization)

batch_normalization_92 (Ba  (None, 5, 5, 384)          1152      ['conv2d_92[0][0]']
tchNormalization)

conv2d_93 (Conv2D)          (None, 5, 5, 192)          393216    ['average_pooling2d_8[0][0]']

batch_normalization_85 (Ba  (None, 5, 5, 320)          960       ['conv2d_85[0][0]']
tchNormalization)

activation_87 (Activation)  (None, 5, 5, 384)          0         ['batch_normalization_87[0][0]
                                                                 ']

activation_88 (Activation)  (None, 5, 5, 384)          0         ['batch_normalization_88[0][0]
                                                                 ']

activation_91 (Activation)  (None, 5, 5, 384)          0         ['batch_normalization_91[0][0]
                                                                 ']

activation_92 (Activation)  (None, 5, 5, 384)          0         ['batch_normalization_92[0][0]
                                                                 ']

batch_normalization_93 (Ba  (None, 5, 5, 192)          576       ['conv2d_93[0][0]']
tchNormalization)

activation_85 (Activation)  (None, 5, 5, 320)          0         ['batch_normalization_85[0][0]
                                                                 ']

mixed9_1 (Concatenate)      (None, 5, 5, 768)          0         ['activation_87[0][0]',
                                                                  'activation_88[0][0]']

concatenate_1 (Concatenate  (None, 5, 5, 768)          0         ['activation_91[0][0]',
)                                                                 'activation_92[0][0]']

activation_93 (Activation)  (None, 5, 5, 192)          0         ['batch_normalization_93[0][0]
                                                                 ']

mixed10 (Concatenate)       (None, 5, 5, 2048)         0         ['activation_85[0][0]',
                                                                  'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

flatten (Flatten)           (None, 51200)              0         ['mixed10[0][0]']

batch_normalization_94 (Ba  (None, 51200)              204800    ['flatten[0][0]']
tchNormalization)

dense (Dense)               (None, 4)                  204804    ['batch_normalization_94[0][0]
                                                                 ']

==================================================================================================
Total params: 22212388 (84.73 MB)
Trainable params: 307204 (1.17 MB)
Non-trainable params: 21905184 (83.56 MB)
__________________________________________________________________________________________________
```

Milestone 4 : Testing the Model

Activity 1 : Load the model with best val_accuracy

Out of all the models we tried (VGG19, Resnet50 , Inception V3) RESNET50 gave us the best Accuracy of 93% so we load the RESNET50 model saved through callbacks function

```
[ ]  from tensorflow.keras.models import load_model
     resnet_loadedmodel= load_model('resnet50best_model.h5')
```

Activity 2 : Define a function to make predictions using the loaded model.

The code features a predict function using a pre-trained ResNet model to classify input images into 'cataract,' 'diabetic_retinopathy,' 'glaucoma,' or 'normal.' It simplifies image classification by determining the most likely class and returning a human-readable label.

```
[ ]  def predict(img):
       y=image.img_to_array(img)
       y = np.expand_dims(y,axis = 0)
       pred =np.argmax(incepion_loaded_model.predict(y))
       op =['cataract','diabetic_retinopathy','glaucoma','normal']
       return op[pred]
```

Activity 3 : Testing and Prediction with Preprocessed Images

```
[ ] img1 = image.load_img('/content/dataset/glaucoma/1211_right.jpg',target_size =(300,300))
    predict(img1)

    1/1 [==============================] - 2s 2s/step
    'glaucoma'

[ ] img2 = image.load_img('/content/dataset/normal/2599_left.jpg',target_size =(300,300))
    predict(img2)

    1/1 [==============================] - 0s 25ms/step
    'normal'

    img3 = image.load_img('/content/dataset/cataract/2127_left.jpg',target_size =(300,300))
    predict(img3)

    1/1 [==============================] - 0s 37ms/step
    'cataract'

[ ] img4 = image.load_img('/content/dataset/diabetic_retinopathy/1000_left.jpeg',target_size =(300,300))
    predict(img4)

    1/1 [==============================] - 0s 39ms/step
    'diabetic_retinopathy'

[ ] img5 = image.load_img('/content/dataset/normal/2780_left.jpg',target_size =(300,300))
    predict(img5)

    1/1 [==============================] - 0s 25ms/step
    'normal'
```

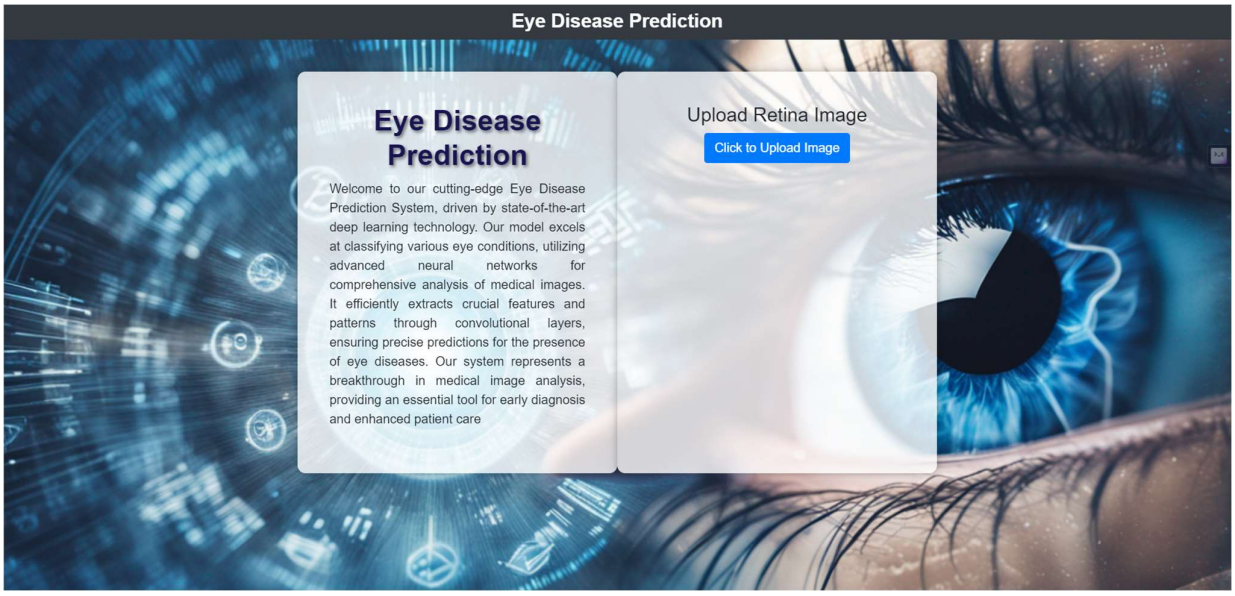
Here we used the RESNET 50 model and all the predictions are predicted correctly.

Milestone 5 : 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 upload the images for predictions. The images are given to the saved model and prediction is showcased on the UI.

Activity1: Building HTML Pages

For this project create one HTML file namely index.html
Let's see how our index.html page looks like:



Activity 2 : Build a python application

1. Import the necessary libraries

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, render_template, request
import os
import numpy as np
```

2. Initialize the Flask app, load the model

```
app = Flask(__name__)
model = load_model("resnet50_best.h5", compile=False)
```

3. Create the Homepage Route

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

4. Image Upload and Prediction Handling

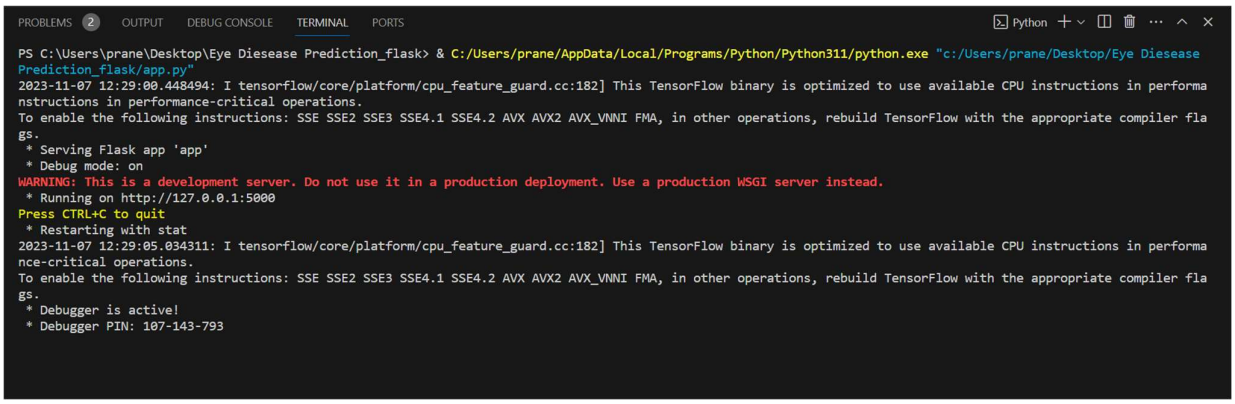
```
@app.route('/predict', methods=['GET','POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        basepath = os.path.dirname(__file__)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        f.save(filepath)

        img = image.load_img(filepath, target_size=(224, 224))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        pred = np.argmax(model.predict(x), axis=1)
        index = ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
        text = "The Eye Disease is  " + str(index[pred[0]])
        return text
```

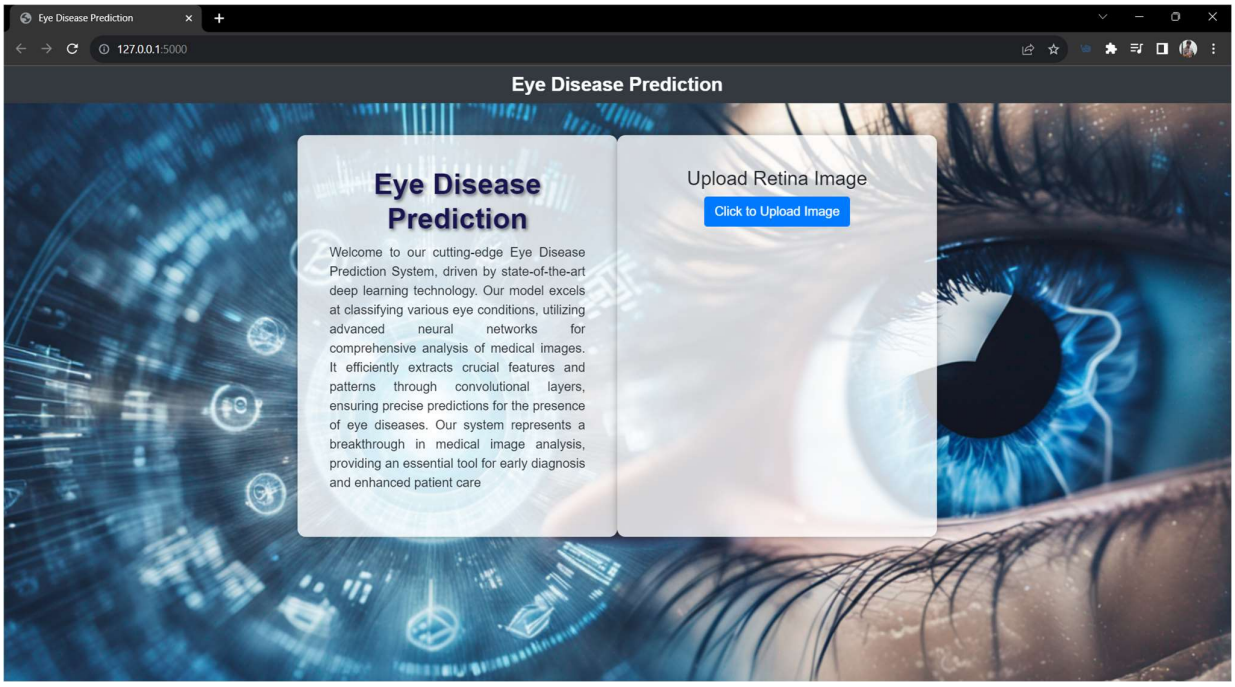
5. Main function

```
if __name__ == '__main__':
    app.run(debug=True)
```

Activity 3 : Run the Python application in VS Code

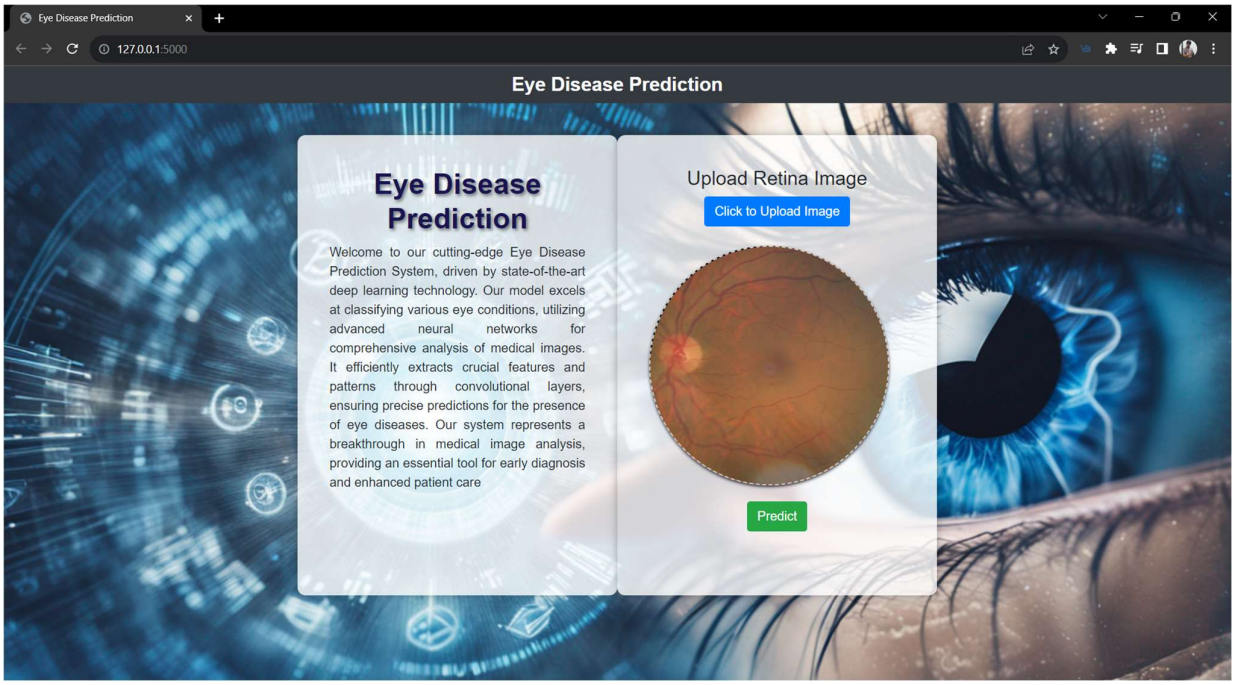


Here open the given URL (http://127.0.0.1:5000) which takes to the User Interface of our app

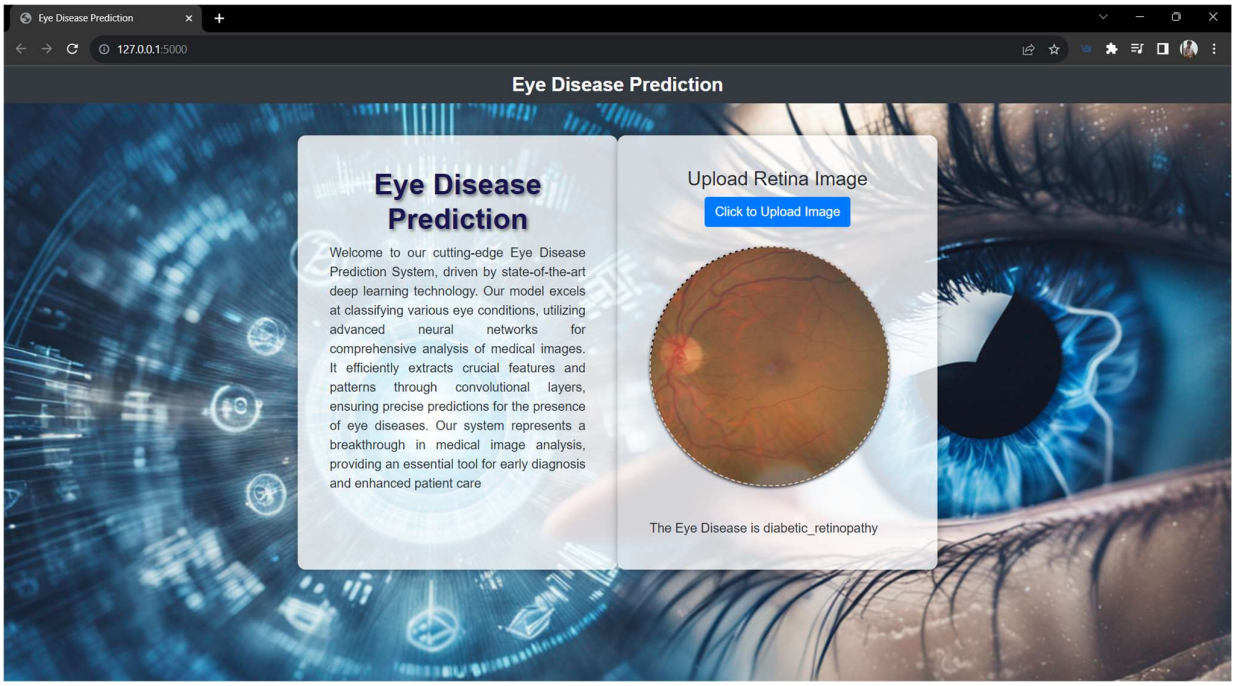


Click on the upload image button to upload the retina image of the Patient

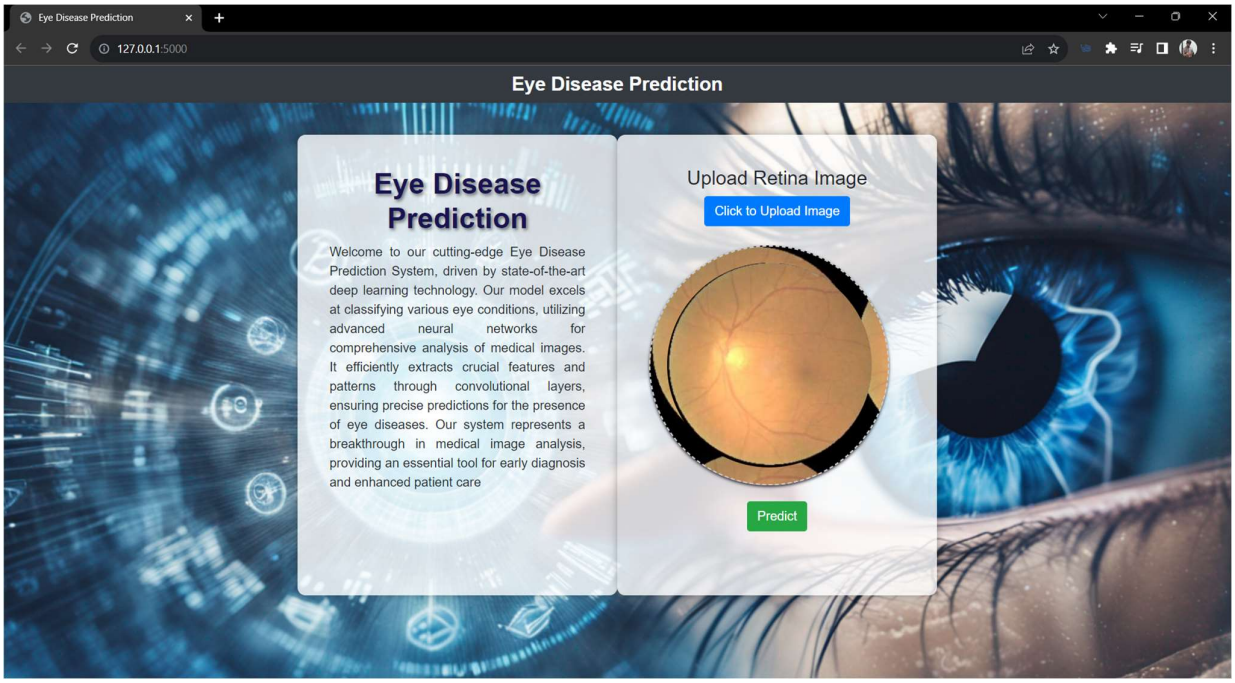
Input 1:



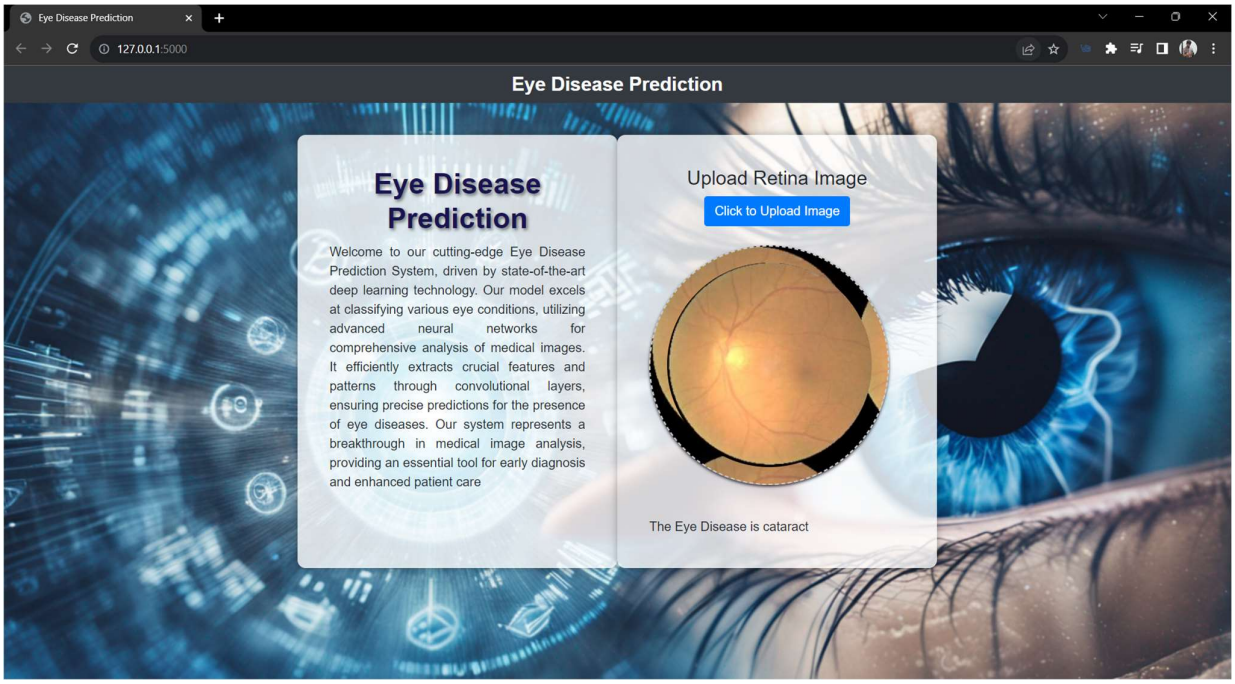
Output 1 :



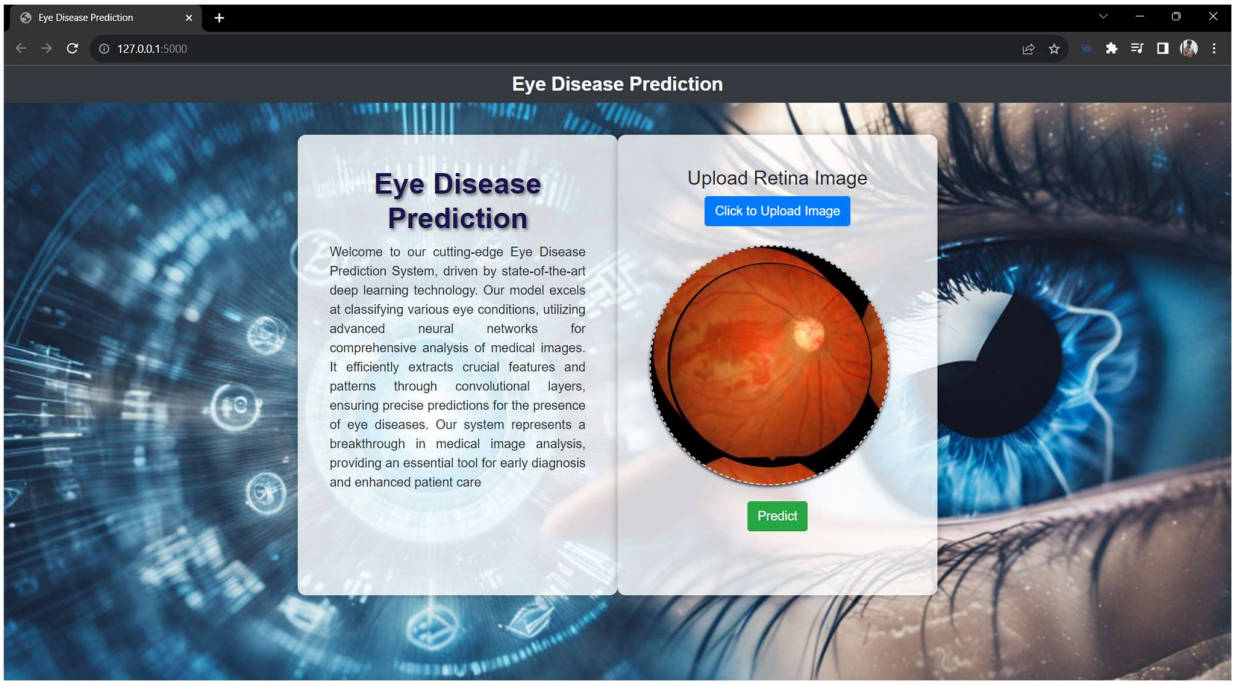
Input 2 :



Output 2 :



Input 3 :



Output 3 :