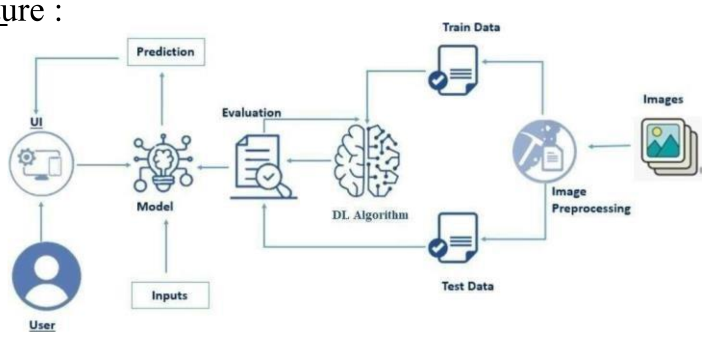
Eye Disease Detection Using Deep Learning

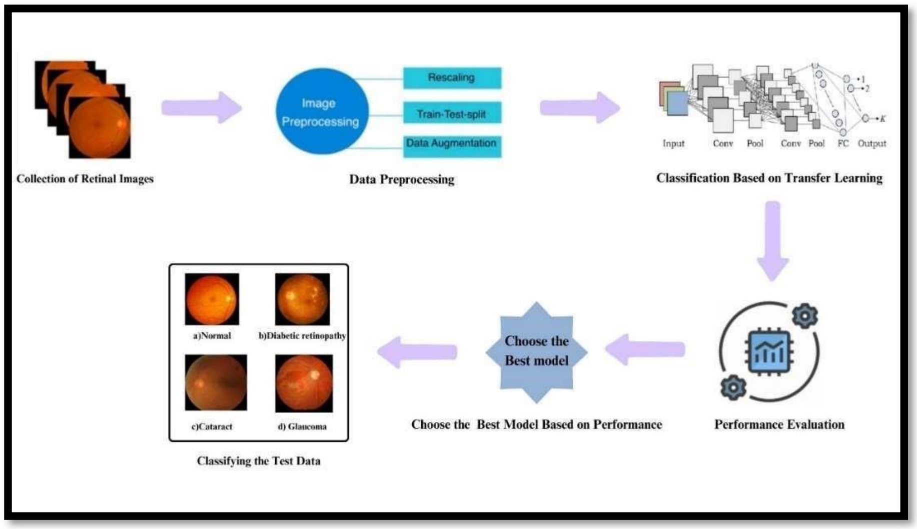
Project Description :

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 o 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)

o Image Pre-processing (For VGG19 & InceptionV3) Import the necessary libraries.

- Configure ImageDataGenerator for data augmentation
- Apply ImageDataGenerator to the training and validation data.

o 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

o 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.

o Application Building

- Create an HTML file
- Create a Flask application and integrate the HTML file

Prior Knowledge:

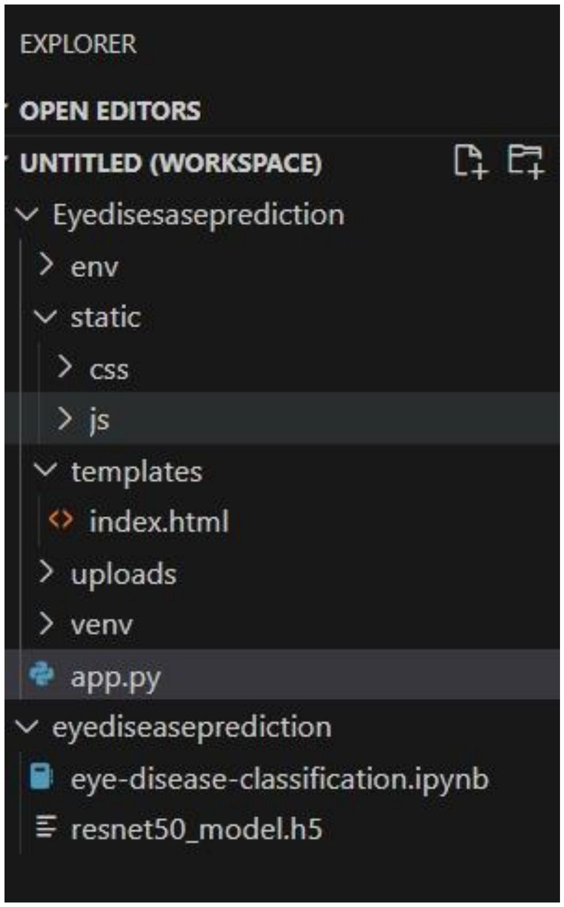
You must have prior knowledge of following topics to complete this project.

● 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 library used for developing web applications.

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

Project Structure:



EXPLORER
OPEN EDITORS
UNTITLED (WORKSPACE)
```
Eyedisesaseprediction
  env
  static
    css
    js
  templates
    index.html
  uploads
  venv
  app.py
eyediseaseprediction
  eye-disease-classification.ipynb
  resnet50_model.h5
```

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

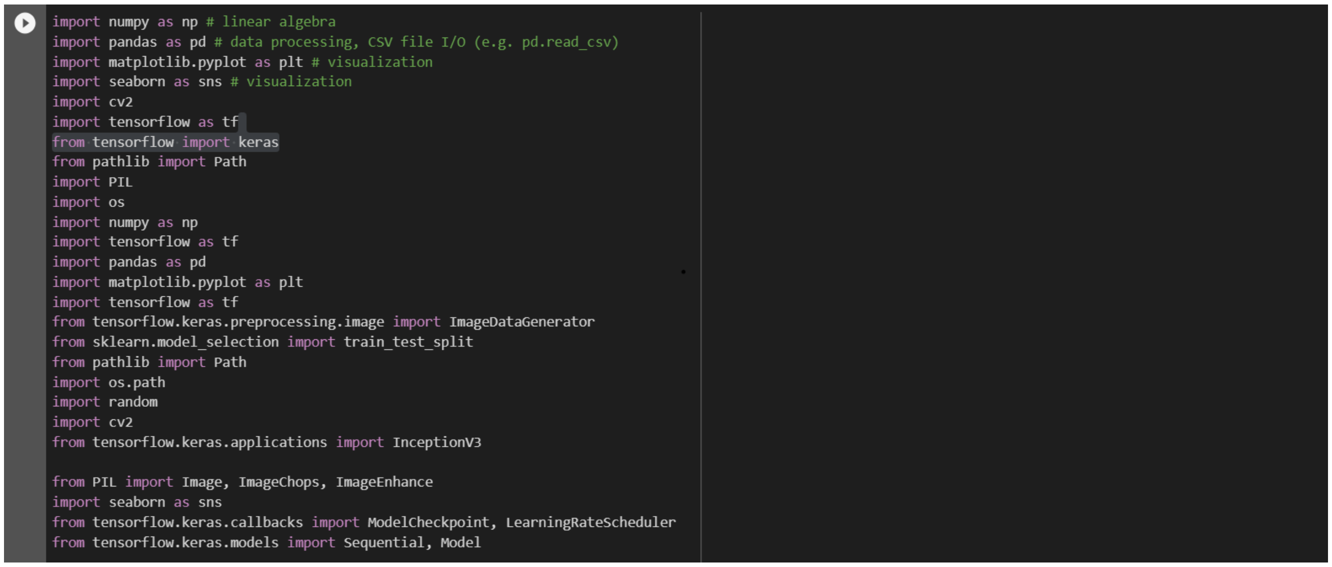
Milestone 1: Data Collection

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.

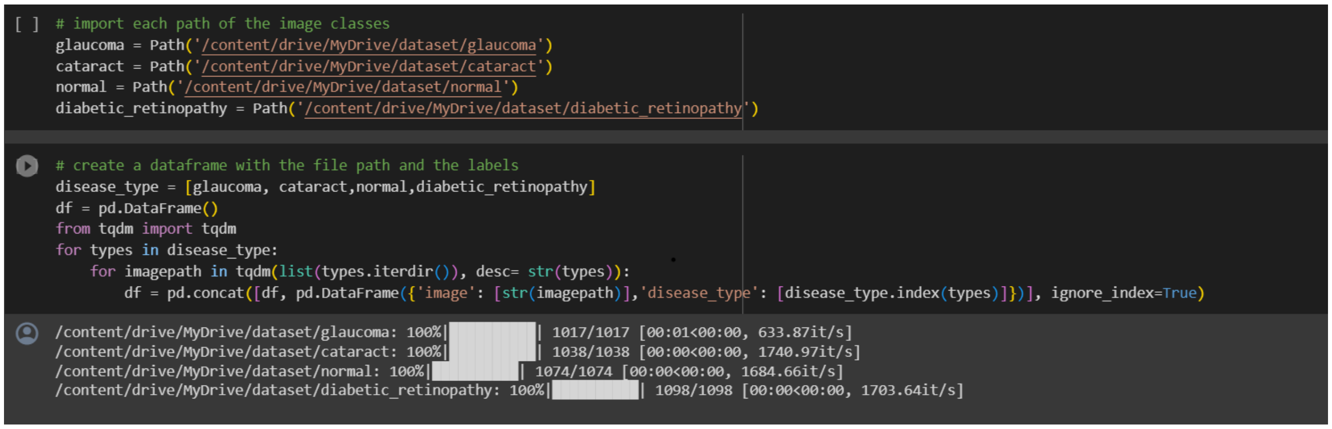
You can download the dataset used in this project using the below link
Dataset: https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification

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



```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # visualization
import seaborn as sns # visualization
import cv2
import tensorflow as tf
from tensorflow import keras
from pathlib import Path
import PIL
import os
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from pathlib import Path
import os.path
import random
import cv2
from tensorflow.keras.applications import InceptionV3

from PIL import Image, ImageChops, ImageEnhance
import seaborn as sns
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
from tensorflow.keras.models import Sequential, Model
```

Activity 2 : Import each path of the image and create a DataFrame with the File path and Labels



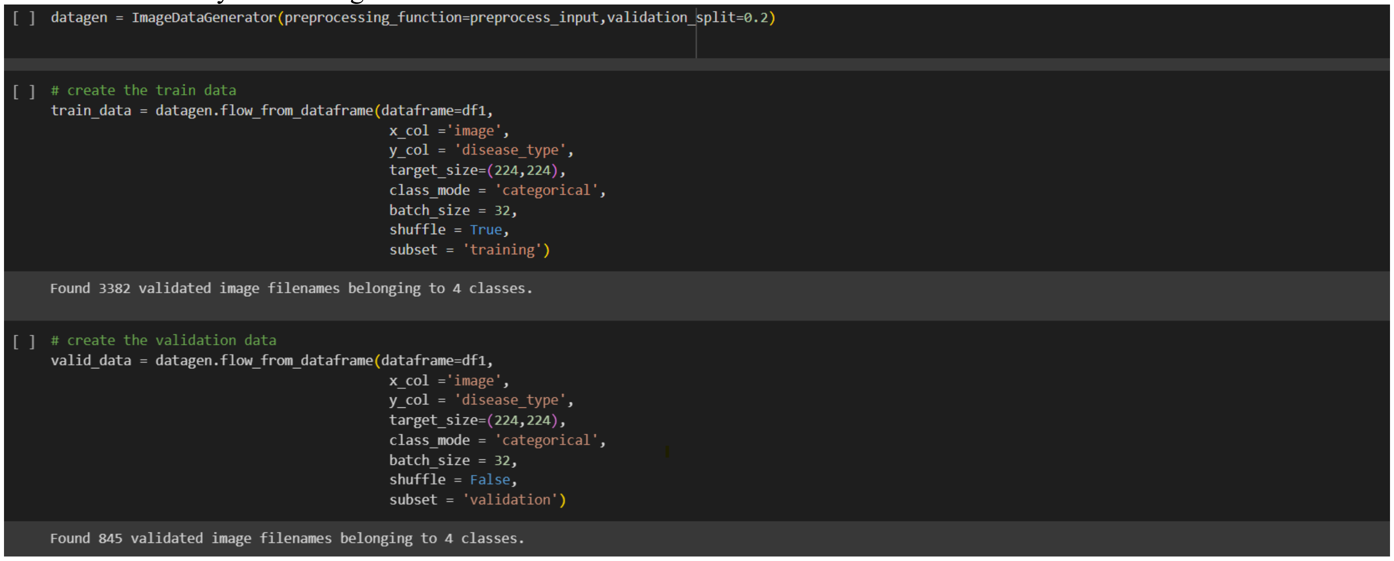
```
[ ] # import each path of the image classes
    glaucoma = Path('/content/drive/MyDrive/dataset/glaucoma')
    cataract = Path('/content/drive/MyDrive/dataset/cataract')
    normal = Path('/content/drive/MyDrive/dataset/normal')
    diabetic_retinopathy = Path('/content/drive/MyDrive/dataset/diabetic_retinopathy')
```

```
    # create a dataframe with the file path and the labels
    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=True)
```

```
/content/drive/MyDrive/dataset/glaucoma: 100%|██████████| 1017/1017 [00:01<00:00, 633.87it/s]
/content/drive/MyDrive/dataset/cataract: 100%|██████████| 1038/1038 [00:00<00:00, 1740.97it/s]
/content/drive/MyDrive/dataset/normal: 100%|██████████| 1074/1074 [00:00<00:00, 1684.66it/s]
/content/drive/MyDrive/dataset/diabetic_retinopathy: 100%|██████████| 1098/1098 [00:00<00:00, 1703.64it/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.



```
[ ] datagen = ImageDataGenerator(preprocessing_function=preprocess_input,validation_split=0.2)
```

```
[ ] # create the train data
    train_data = datagen.flow_from_dataframe(dataframe=df1,
                                              x_col ='image',
                                              y_col = 'disease_type',
                                              target_size=(224,224),
                                              class_mode = 'categorical',
                                              batch_size = 32,
                                              shuffle = True,
                                              subset = 'training')

    Found 3382 validated image filenames belonging to 4 classes.
```

```
[ ] # create the validation data
    valid_data = datagen.flow_from_dataframe(dataframe=df1,
                                              x_col ='image',
                                              y_col = 'disease_type',
                                              target_size=(224,224),
                                              class_mode = 'categorical',
                                              batch_size = 32,
                                              shuffle = False,
                                              subset = 'validation')

    Found 845 validated image filenames belonging to 4 classes.
```

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.

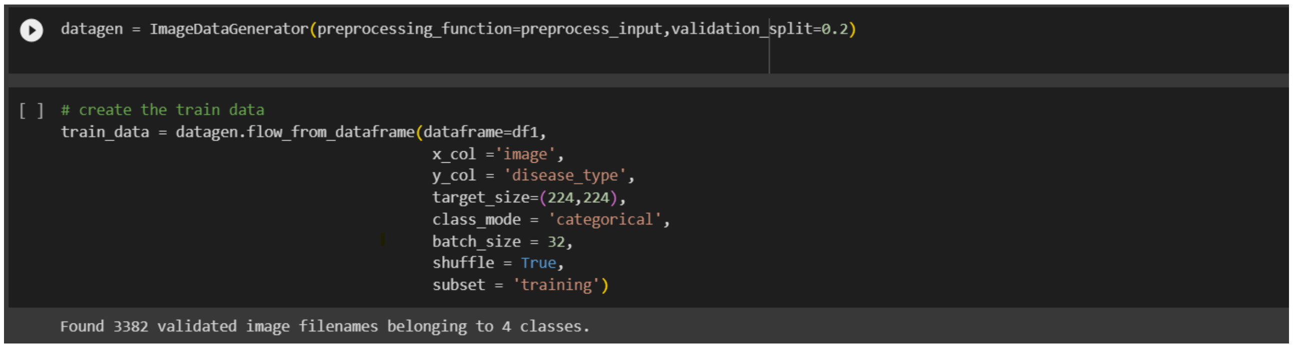
3. Zooming: Random zooming in or out of the images within a 20% range.

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

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

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.



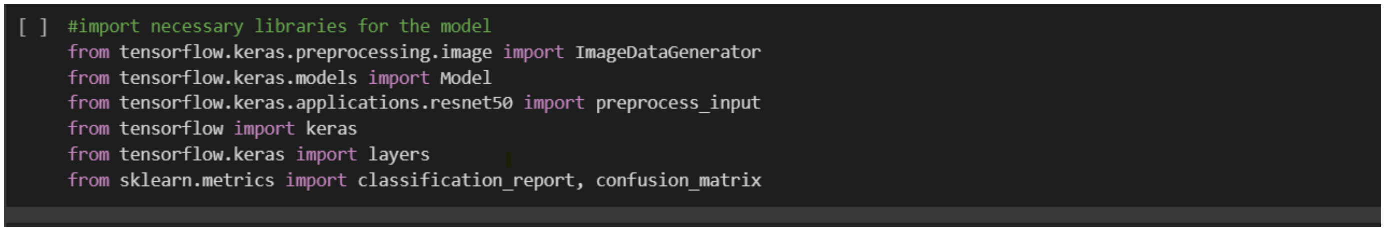
```
datagen = ImageDataGenerator(preprocessing_function=preprocess_input,validation_split=0.2)
```

```
[ ] # create the train data
    train_data = datagen.flow_from_dataframe(dataframe=df1,
                                             x_col ='image',
                                             y_col = 'disease_type',
                                             target_size=(224,224),
                                             class_mode = 'categorical',
                                             batch_size = 32,
                                             shuffle = True,
                                             subset = 'training')
```

```
Found 3382 validated image filenames belonging to 4 classes.
```

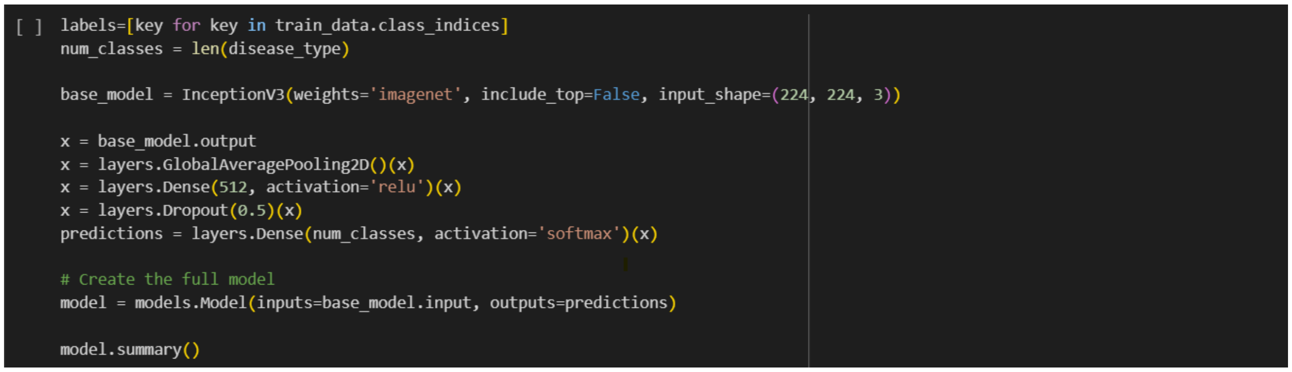
Milestone 3 : Model Building

Activity 1: Import the required libraries.

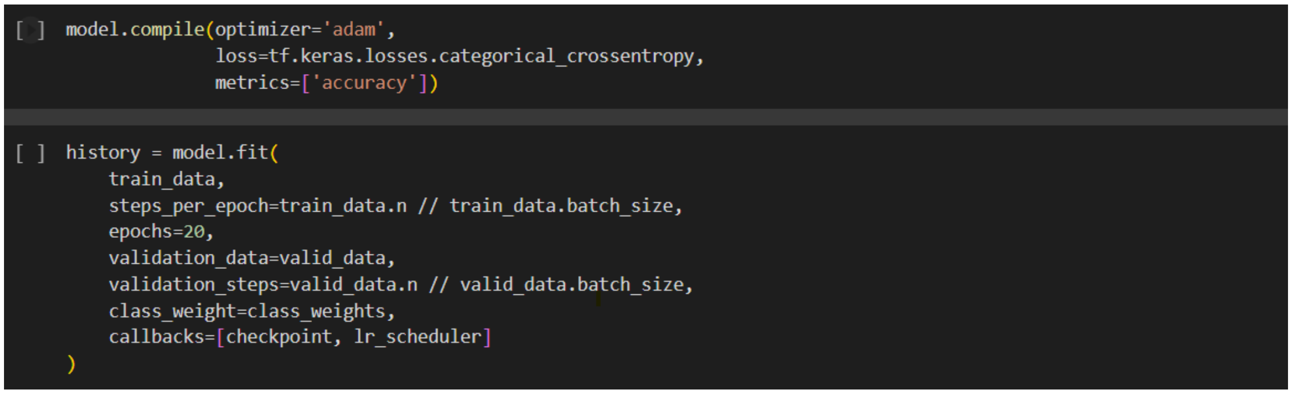


```
[ ] #import necessary libraries for the model
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.models import Model
    from tensorflow.keras.applications.resnet50 import preprocess_input
    from tensorflow import keras
    from tensorflow.keras import layers
    from sklearn.metrics import classification_report, confusion_matrix
```

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.



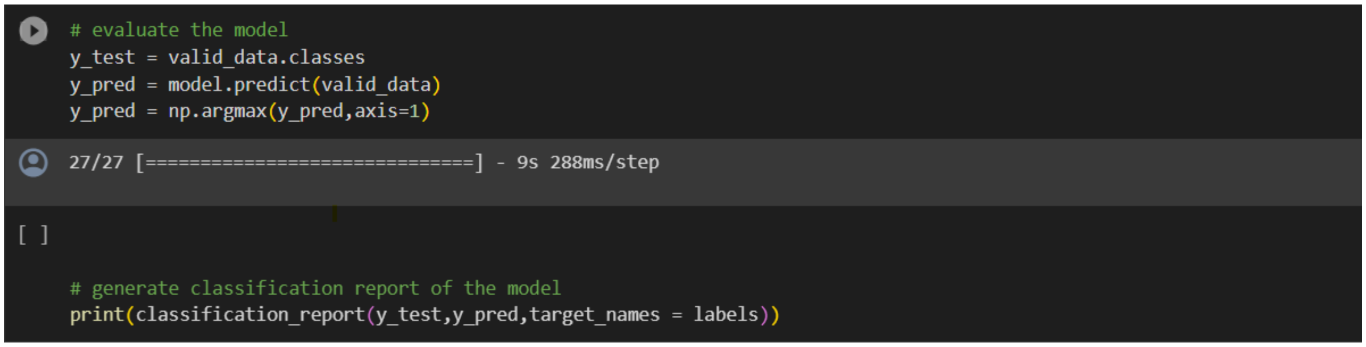
```
[ ] labels=[key for key in train_data.class_indices]
    num_classes = len(disease_type)

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

    x = base_model.output
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(512, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    predictions = layers.Dense(num_classes, activation='softmax')(x)

    # Create the full model
    model = models.Model(inputs=base_model.input, outputs=predictions)

    model.summary()
```



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

[ ] history = model.fit(
        train_data,
        steps_per_epoch=train_data.n // train_data.batch_size,
        epochs=20,
        validation_data=valid_data,
        validation_steps=valid_data.n // valid_data.batch_size,
        class_weight=class_weights,
        callbacks=[checkpoint, lr_scheduler]
    )
```

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

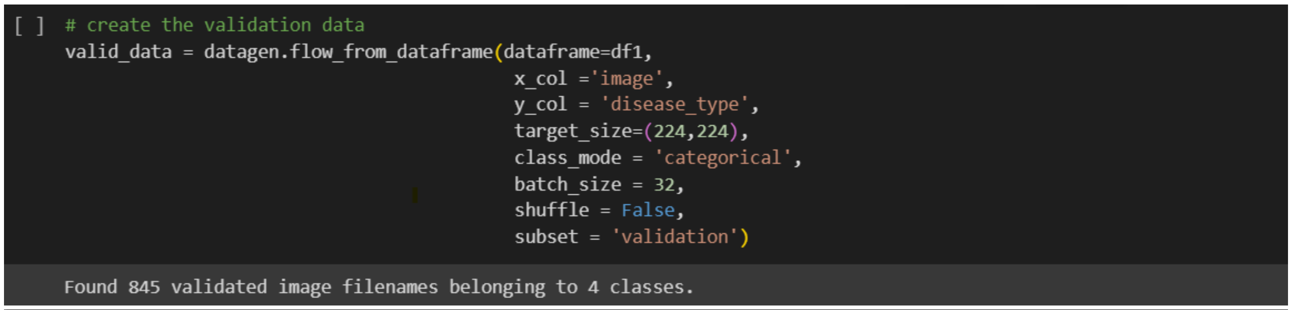
The code compiles various models with their corresponding loss function, utilizing the 'adam' optimizer for efficient parameter adjustment. The 'accuracy' metric assesses model performance during training and validation, facilitating their use in multi-class classification tasks.



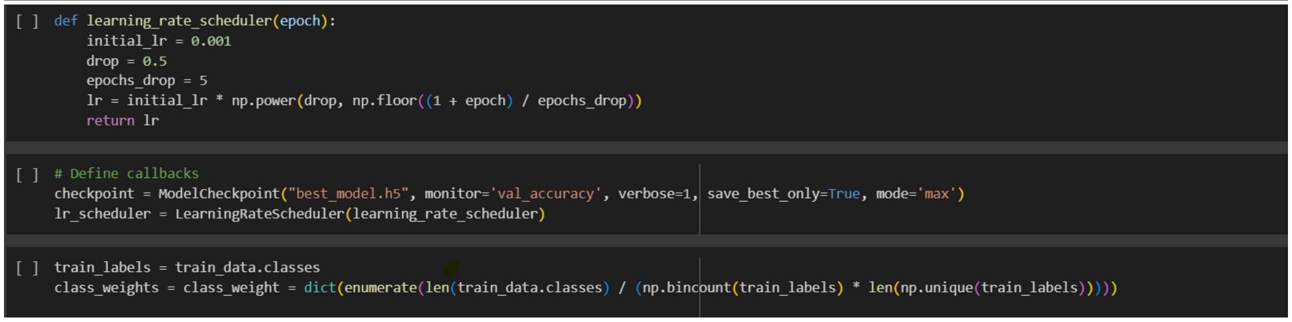
```
# evaluate the model
y_test = valid_data.classes
y_pred = model.predict(valid_data)
y_pred = np.argmax(y_pred,axis=1)

27/27 [==============================] - 9s 288ms/step

[ ]

# generate classification report of the model
print(classification_report(y_test,y_pred,target_names = labels))
```

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



```
[ ]  # create the validation data
     valid_data = datagen.flow_from_dataframe(dataframe=df1,
                                              x_col ='image',
                                              y_col = 'disease_type',
                                              target_size=(224,224),
                                              class_mode = 'categorical',
                                              batch_size = 32,
                                              shuffle = False,
                                              subset = 'validation')

     Found 845 validated image filenames belonging to 4 classes.
```



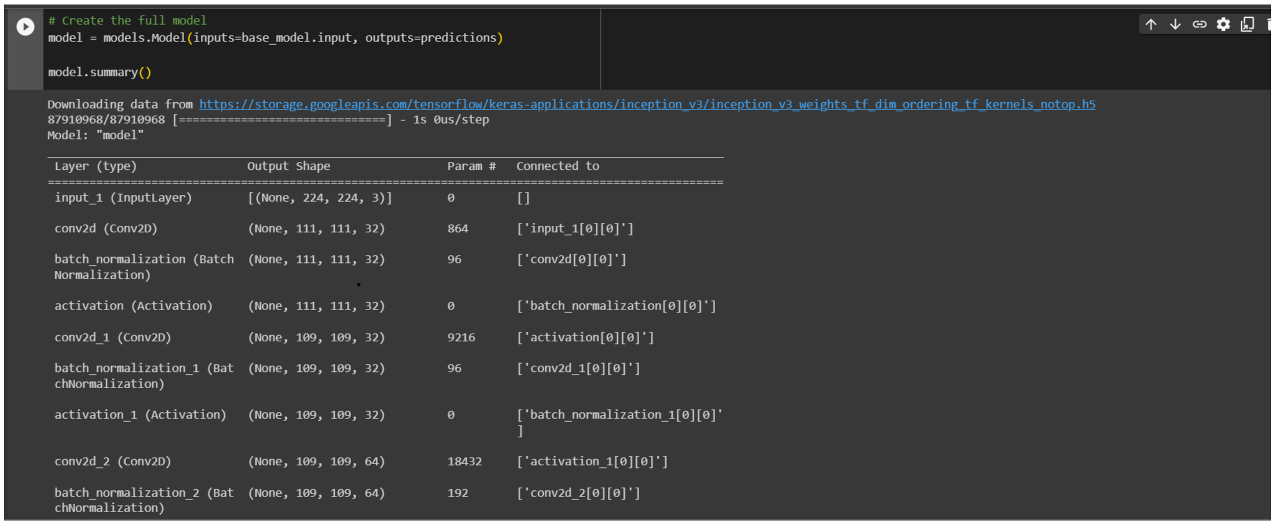
```
[ ] def learning_rate_scheduler(epoch):
        initial_lr = 0.001
        drop = 0.5
        epochs_drop = 5
        lr = initial_lr * np.power(drop, np.floor((1 + epoch) / epochs_drop))
        return lr
```

```
[ ] # Define callbacks
    checkpoint = ModelCheckpoint("best_model.h5", monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
    lr_scheduler = LearningRateScheduler(learning_rate_scheduler)
```

```
[ ] train_labels = train_data.classes
    class_weights = class_weight = dict(enumerate(len(train_data.classes) / (np.bincount(train_labels) * len(np.unique(train_labels)))))
```

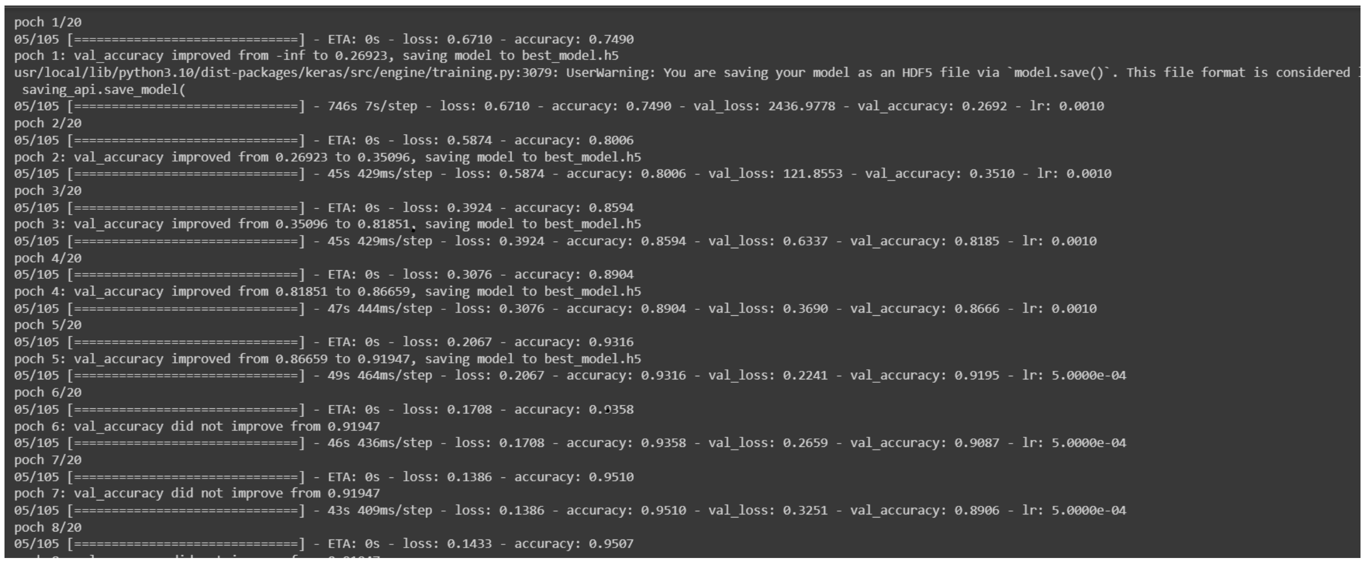
Arguments:

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.



```
# Create the full model
model = models.Model(inputs=base_model.input, outputs=predictions)

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 1s 0us/step
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3)]  0         []

 conv2d (Conv2D)                (None, 111, 111, 32)  864        ['input_1[0][0]']

 batch_normalization (Batch     (None, 111, 111, 32)  96         ['conv2d[0][0]']
 Normalization)

 activation (Activation)        (None, 111, 111, 32)  0          ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)              (None, 109, 109, 32)  9216       ['activation[0][0]']

 batch_normalization_1 (Bat     (None, 109, 109, 32)  96         ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)      (None, 109, 109, 32)  0          ['batch_normalization_1[0][0]'
                                                                  ]

 conv2d_2 (Conv2D)              (None, 109, 109, 64)  18432      ['activation_1[0][0]']

 batch_normalization_2 (Bat     (None, 109, 109, 64)  192        ['conv2d_2[0][0]']
 chNormalization)
```

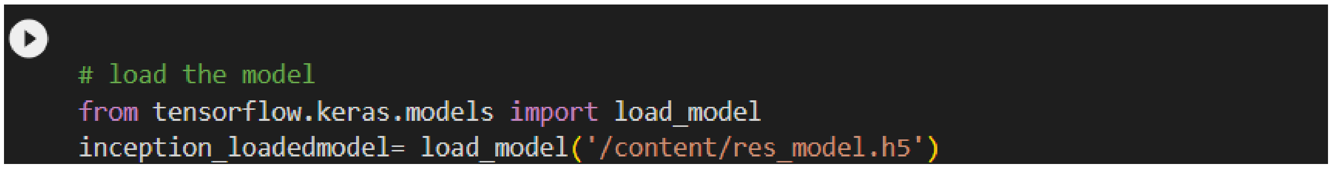


```
poch 1/20
05/105 [==============================] - ETA: 0s - loss: 0.6710 - accuracy: 0.7490
poch 1: val_accuracy improved from -inf to 0.26923, saving model to best_model.h5
usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered
 saving_api.save_model(
05/105 [==============================] - 746s 7s/step - loss: 0.6710 - accuracy: 0.7490 - val_loss: 2436.9778 - val_accuracy: 0.2692 - lr: 0.0010
poch 2/20
05/105 [==============================] - ETA: 0s - loss: 0.5874 - accuracy: 0.8006
poch 2: val_accuracy improved from 0.26923 to 0.35096, saving model to best_model.h5
05/105 [==============================] - 45s 429ms/step - loss: 0.5874 - accuracy: 0.8006 - val_loss: 121.8553 - val_accuracy: 0.3510 - lr: 0.0010
poch 3/20
05/105 [==============================] - ETA: 0s - loss: 0.3924 - accuracy: 0.8594
poch 3: val_accuracy improved from 0.35096 to 0.81851, saving model to best_model.h5
05/105 [==============================] - 45s 429ms/step - loss: 0.3924 - accuracy: 0.8594 - val_loss: 0.6337 - val_accuracy: 0.8185 - lr: 0.0010
poch 4/20
05/105 [==============================] - ETA: 0s - loss: 0.3076 - accuracy: 0.8904
poch 4: val_accuracy improved from 0.81851 to 0.86659, saving model to best_model.h5
05/105 [==============================] - 47s 444ms/step - loss: 0.3076 - accuracy: 0.8904 - val_loss: 0.3690 - val_accuracy: 0.8666 - lr: 0.0010
poch 5/20
05/105 [==============================] - ETA: 0s - loss: 0.2067 - accuracy: 0.9316
poch 5: val_accuracy improved from 0.86659 to 0.91947, saving model to best_model.h5
05/105 [==============================] - 49s 464ms/step - loss: 0.2067 - accuracy: 0.9316 - val_loss: 0.2241 - val_accuracy: 0.9195 - lr: 5.0000e-04
poch 6/20
05/105 [==============================] - ETA: 0s - loss: 0.1708 - accuracy: 0.9358
poch 6: val_accuracy did not improve from 0.91947
05/105 [==============================] - 46s 436ms/step - loss: 0.1708 - accuracy: 0.9358 - val_loss: 0.2659 - val_accuracy: 0.9087 - lr: 5.0000e-04
poch 7/20
05/105 [==============================] - ETA: 0s - loss: 0.1386 - accuracy: 0.9510
poch 7: val_accuracy did not improve from 0.91947
05/105 [==============================] - 43s 409ms/step - loss: 0.1386 - accuracy: 0.9510 - val_loss: 0.3251 - val_accuracy: 0.8906 - lr: 5.0000e-04
poch 8/20
05/105 [==============================] - ETA: 0s - loss: 0.1433 - accuracy: 0.9507
```

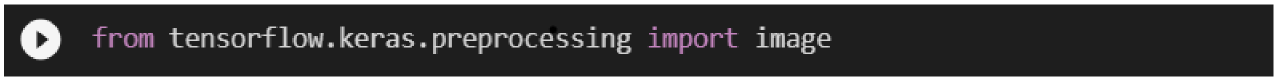
Milestone 4 : Testing the Model

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



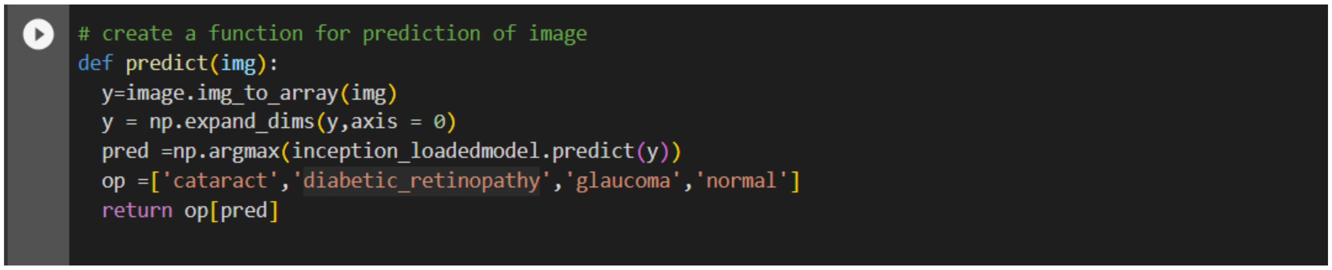
```python
# load the model
from tensorflow.keras.models import load_model
inception_loadedmodel= load_model('/content/res_model.h5')
```



```python
from tensorflow.keras.preprocessing import image
```

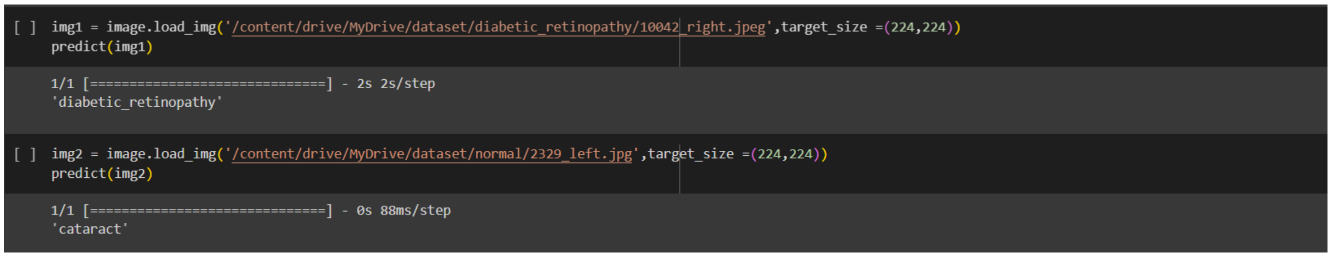
Activity 2 : Define a function to make predictions using the loaded model.

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.



```python
# create a function for prediction of image
def predict(img):
  y=image.img_to_array(img)
  y = np.expand_dims(y,axis = 0)
  pred =np.argmax(inception_loadedmodel.predict(y))
  op =['cataract','diabetic_retinopathy','glaucoma','normal']
  return op[pred]
```

Activity 3 : Testing and Prediction with Preprocessed Images



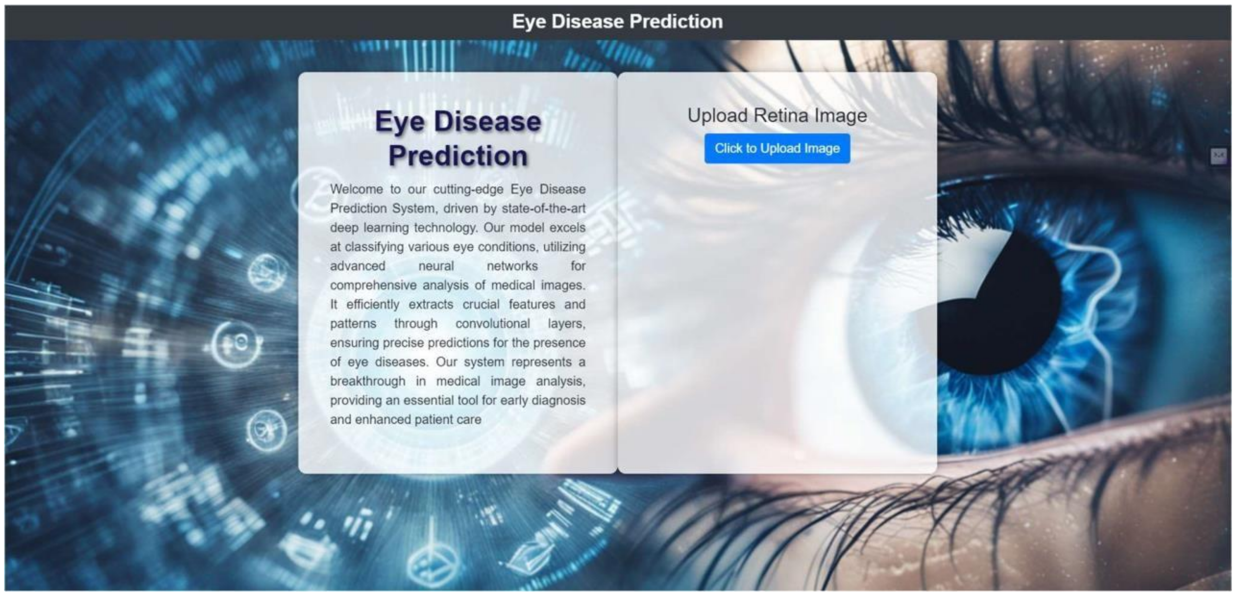
```
[ ] img1 = image.load_img('/content/drive/MyDrive/dataset/diabetic_retinopathy/10042_right.jpeg',target_size =(224,224))
    predict(img1)

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

[ ] img2 = image.load_img('/content/drive/MyDrive/dataset/normal/2329_left.jpg',target_size =(224,224))
    predict(img2)

    1/1 [==============================] - 0s 88ms/step
    'cataract'
```

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:



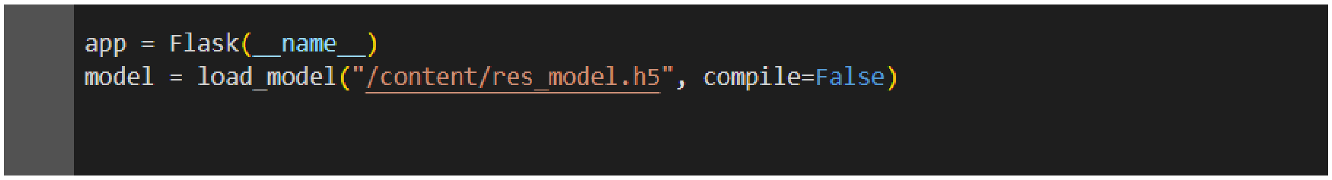
Welcome to our cutting-edge Eye Disease Prediction System, driven by state-of-the-art deep learning technology. Our model excels at classifying various eye conditions, utilizing advanced neural networks for comprehensive analysis of medical images. It efficiently extracts crucial features and patterns through convolutional layers, ensuring precise predictions for the presence of eye diseases. Our system represents a breakthrough in medical image analysis, providing an essential tool for early diagnosis and enhanced patient care

Activity 2 : Build a python application

1. Import the necessary libraries

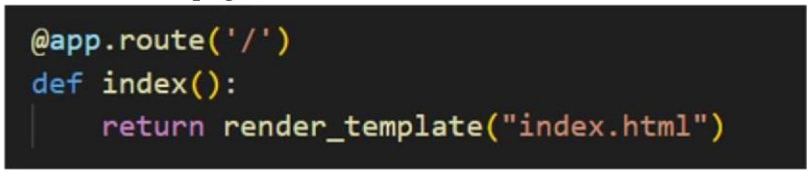
```python
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, render_template, request
import os
import numpy as np
```

2. Initialize the Flask app, load the model



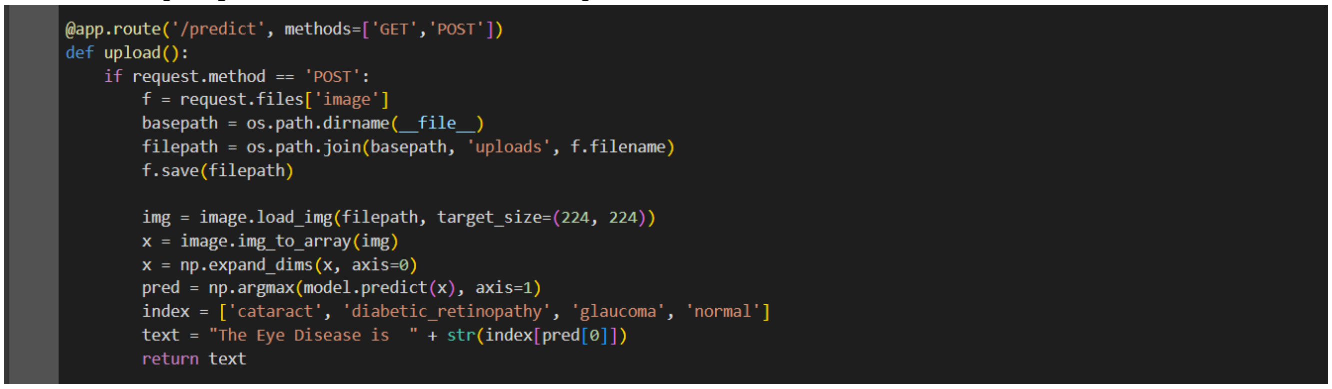
```
app = Flask(__name__)
model = load_model("/content/res_model.h5", compile=False)
```

3. Create the Homepage Route



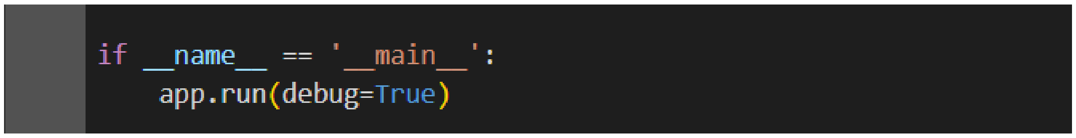
```
@app.route('/')
def index():
    return render_template("index.html")
```

4. Image Upload and Prediction Handling



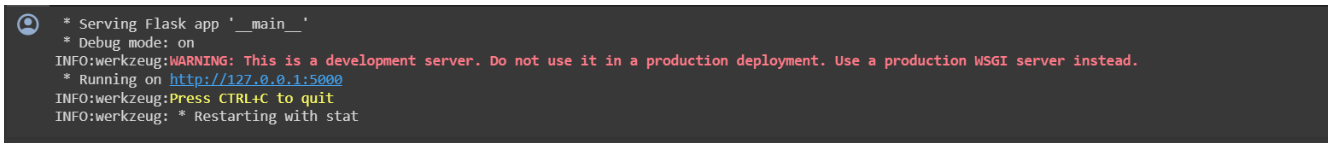
```
@app.route('/predict', methods=['GET','POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        basepath = os.path.dirname(__file__)
        filepath = os.path.join(basepath, 'uploads', f.filename)
        f.save(filepath)

        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



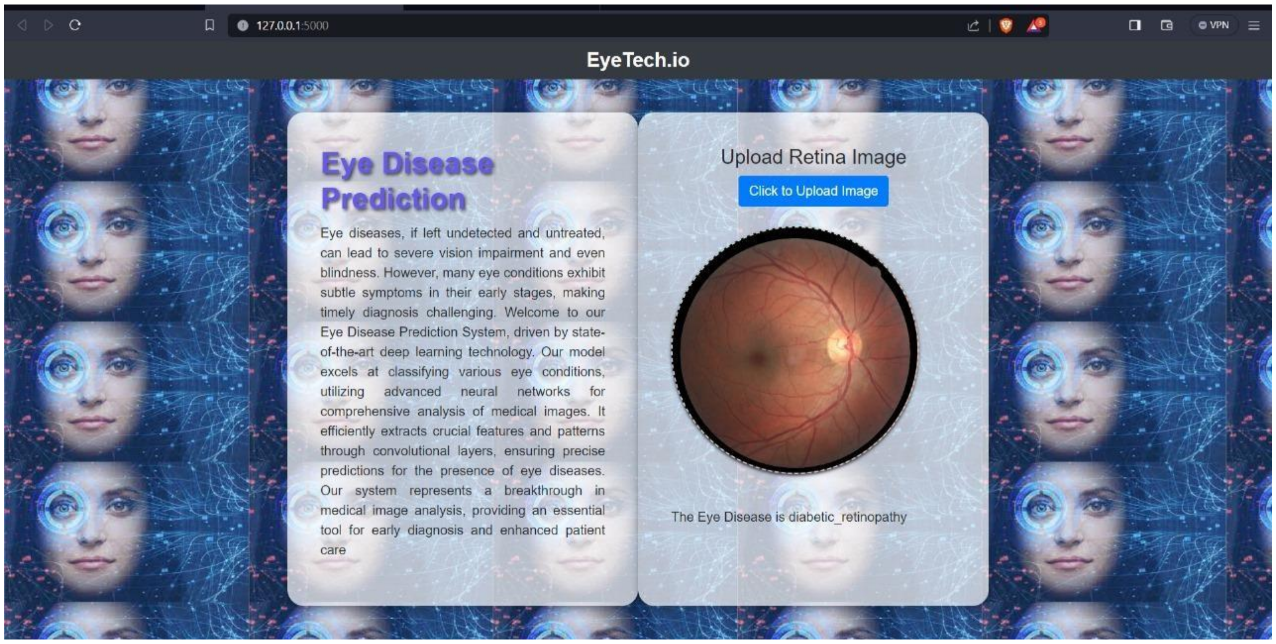
```
if __name__ == '__main__':
    app.run(debug=True)
```

Activity 3 : Run the Python application on in VS Code



```
 * Serving Flask app '__main__'
 * Debug mode: on
INFO:werkzeug: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
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
```

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



EyeTech.io

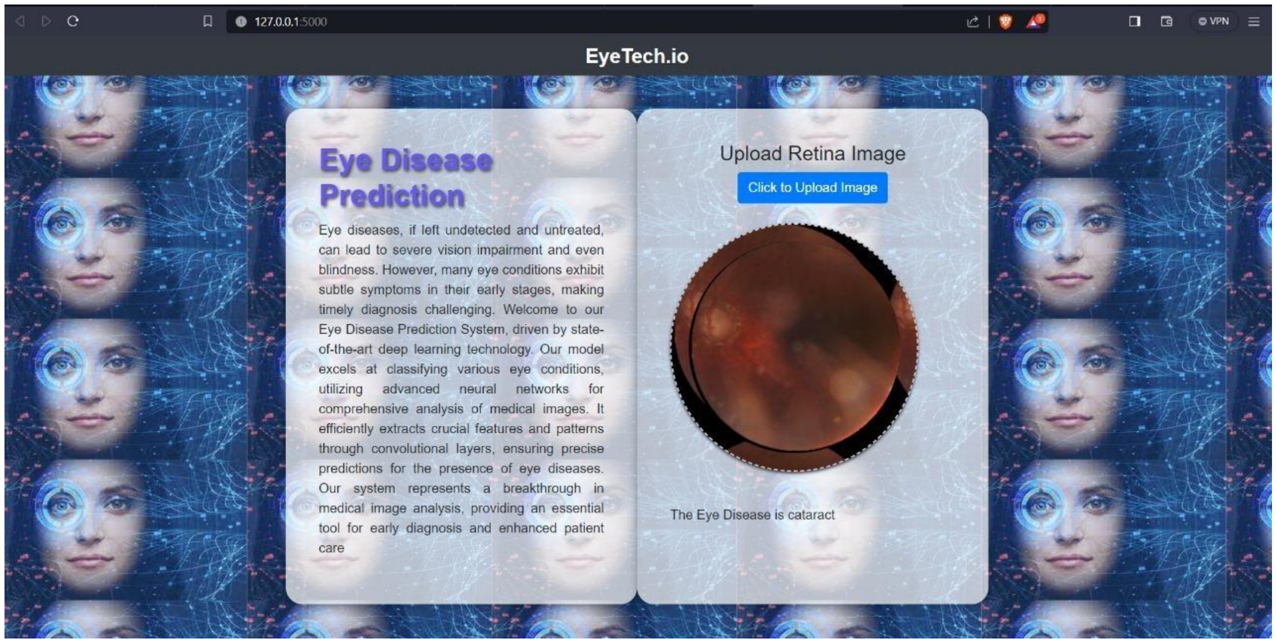
Eye Disease Prediction

Eye diseases, if left undetected and untreated, can lead to severe vision impairment and even blindness. However, many eye conditions exhibit subtle symptoms in their early stages, making timely diagnosis challenging. Welcome to our Eye Disease Prediction System, driven by state-of-the-art deep learning technology. Our model excels at classifying various eye conditions, utilizing advanced neural networks for comprehensive analysis of medical images. It efficiently extracts crucial features and patterns through convolutional layers, ensuring precise predictions for the presence of eye diseases. Our system represents a breakthrough in medical image analysis, providing an essential tool for early diagnosis and enhanced patient care

Upload Retina Image

The Eye Disease is diabetic_retinopathy

Click on the upload image button on to upload the retina image of the Patient <u>Output 2:</u>



EyeTech.io

Eye Disease Prediction

Eye diseases, if left undetected and untreated, can lead to severe vision impairment and even blindness. However, many eye conditions exhibit subtle symptoms in their early stages, making timely diagnosis challenging. Welcome to our Eye Disease Prediction System, driven by state-of-the-art deep learning technology. Our model excels at classifying various eye conditions, utilizing advanced neural networks for comprehensive analysis of medical images. It efficiently extracts crucial features and patterns through convolutional layers, ensuring precise predictions for the presence of eye diseases. Our system represents a breakthrough in medical image analysis, providing an essential tool for early diagnosis and enhanced patient care

Upload Retina Image

The Eye Disease is cataract