Eye Disease Detection Using Deep Learning

Introduction

Eye diseases are a significant global health concern, with conditions such as **cataracts**, **diabetic retinopathy (DR)**, and **glaucoma** leading to vision impairment or blindness if not diagnosed and treated early. Traditional diagnostic methods rely on manual examination by ophthalmologists, which can be time-consuming, expensive, and prone to human error.

Deep Learning (DL), a subset of Artificial Intelligence (AI), has emerged as a transformative tool for automating the detection of eye diseases using medical images. By leveraging **Convolutional Neural Networks (CNNs)** and **Transfer Learning**, we can build highly accurate models to classify eye diseases into four categories: **Normal**, **Cataract**, **Diabetic Retinopathy**, and **Glaucoma**.

This project focuses on developing a deep learning-based system for eye disease detection and integrating it into a user-friendly web application using the **Flask** framework. The system aims to provide a cost-effective, scalable, and efficient solution for early diagnosis of eye diseases.

Objectives

By the end of this project, you will:

- Understand the process of preprocessing medical images for deep learning.
- Apply **Transfer Learning** techniques using pre-trained models like **VGG19**, **ResNet50**, **InceptionV3**, and **Xception**.
- Build and train a deep learning model to classify eye diseases into four categories.
- Evaluate the model's performance using metrics such as accuracy, loss, and confusion matrices.
- Develop a web application using **Flask** to deploy the model for real-time predictions.
- Gain insights into the challenges and future scope of deep learning in medical image analysis.

Project Flow

The project follows a structured workflow to ensure systematic development and deployment:

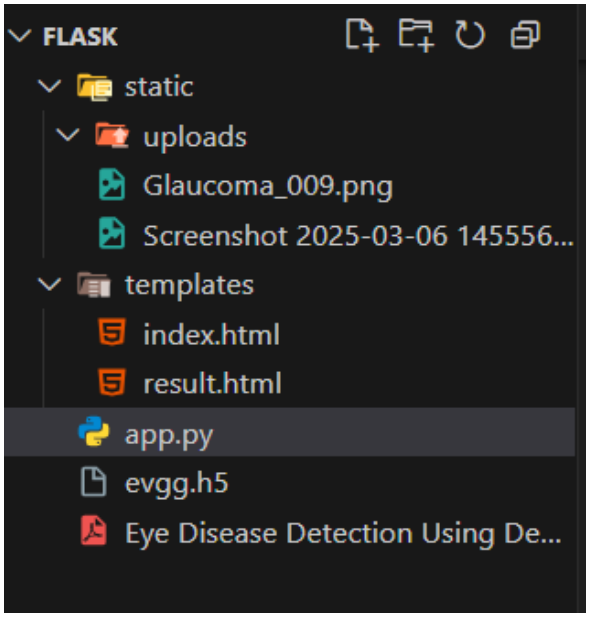
1. **Data Collection**: Gather and organize eye disease images into categories.
2. **Data Preprocessing**: Augment and normalize images to prepare them for model training.
3. **Model Building**: Use transfer learning to train a deep learning model on the preprocessed dataset.
4. **Model Evaluation**: Test the model on unseen data to measure its accuracy and generalization ability.
5. **Application Building**: Integrate the trained model into a Flask web application for real-time predictions.
6. **Deployment**: Run the application and allow users to upload images for disease classification.

Project Structure

The project is organized into the following folders and files:

- **Dataset**: Contains training and testing images categorized into four classes: Normal, Cataract, Diabetic Retinopathy, and Glaucoma.
- **Training**: Includes the model training notebook and the saved model file.
- **templates**: Contains the HTML file for the web interface.
- **static**: Includes CSS files for styling the web interface.
- **app.py**: The Flask application script for handling user requests and predictions.

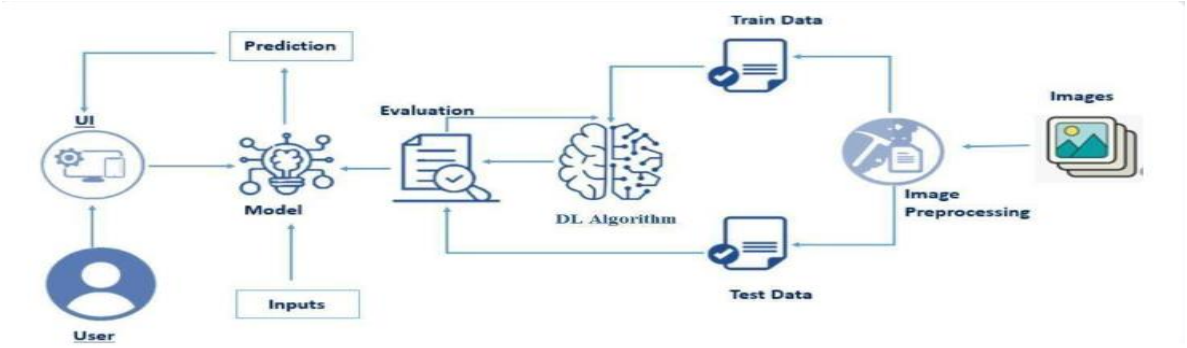
Create a Project folder which contains files as shown below



5. Technical Architecture

The technical architecture of the project consists of the following components:

1. **Frontend**: HTML and CSS for the user interface, allowing users to upload images and view predictions.
2. **Backend**: Flask framework for handling user requests, processing images, and serving predictions.
3. **Deep Learning Model**: Pre-trained CNN models (e.g., VGG19, ResNet50) for feature extraction and classification.
4. **Database**: Local storage for training and testing datasets.



6. Data Collection

- **Dataset Source**: The dataset is collected from publicly available sources like **Kaggle** and organized into four categories: **Normal**, **Cataract**, **Diabetic Retinopathy**, and **Glaucoma**.
- **Dataset Link**: [Eye Diseases Classification Dataset]
- **Dataset Structure**:
    - **Training Data**: 3,372 images.
    - **Testing Data**: 845 images.

Dataset folder contains two folders train and test ,which are used to train and test the model.

Both train and test contains eye diseases of four categories Normal, Cataract, Diabetic Retinopathy, and Glaucoma.

Data Preprocessing

- **Image Augmentation**: Techniques like rotation, scaling, flipping, and brightness adjustment are applied to increase dataset diversity and improve model generalization.
- **Normalization**: Pixel values are scaled to the range [0, 1] by dividing by 255 to ensure consistent input for the model.

- **Resizing**: Images are resized to 224x224 pixels to match the input size of pre-trained models like VGG19.

Model Building

Transfer Learning

- **Pre-trained Models**: VGG19, ResNet50, InceptionV3, and Xception are used as feature extractors. These models are pre-trained on the ImageNet dataset and fine-tuned for the eye disease classification task.
- **Model Architecture**:
    - The base model (e.g., VGG19) is used with frozen layers to extract features from the input images.
    - Additional dense layers are added for classification, followed by a softmax activation layer to output probabilities for the four classes.

Training

- The model is trained for 50 epochs using the training dataset.
- Model checkpoints are saved to retain the best-performing model based on validation accuracy.
- The model is evaluated on the testing dataset to measure its performance.
- Metrics such as accuracy, loss, and confusion matrices are used to assess the model's effectiveness.

When You Run train_model.py Loads the dataset from Dataset/train/ and Dataset/test/. After completion a trained model evgg.h5 is stored in the project directory.

train_model

```
import splitfolders

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from PIL import ImageFile

ImageFile.LOAD_TRUNCATED_IMAGES = True

from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input

from tensorflow.keras.layers import Flatten, Dense

from tensorflow.keras.models import Model

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing import image

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

%matplotlib inline
```

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

train_datagen = ImageDataGenerator (rescale = 1./255,shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
training_set =train_datagen.flow_from_directory('/content/output/train',target_size=(224, 224),batch_size = 64, class_mode='categorical')

test_set = test_datagen.flow_from_directory('/content/output/val', target_size=(224, 224), batch_size=64, class_mode='categorical')
```

```
Found 1372 images belonging to 4 classes.
Found 845 images belonging to 4 classes.
```

```
IMAGE_SIZE = (224, 224)  # Standard input size for VGG19
```

```
from tensorflow.keras.applications import VGG19

IMAGE_SIZE = (224, 224)  # Define image size
VGG19_model = VGG19(input_shape=IMAGE_SIZE + (3,), weights='imagenet', include_top=False)
# Import necessary libraries
from tensorflow.keras.applications import VGG19
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Model

# Define image size
IMAGE_SIZE = (224, 224)

# Load the VGG19 model without the top layers
VGG19_model = VGG19(input_shape=IMAGE_SIZE + (3,), weights='imagenet', include_top=False)

# Freeze all layers in VGG19
for layer in VGG19_model.layers:
    layer.trainable = False

# Add custom layers
x = Flatten()(VGG19_model.output)
prediction = Dense(4, activation='softmax')(x)  # Assuming 4 classes

# Define the final model
model = Model(inputs=VGG19_model.input, outputs=prediction)

# Print model summary to verify
model.summary()
```

Application Building

Flask Web Application:

Backend

Flask framework for handling image uploads, processing them using the trained model, and displaying the results.

The app.py file is the core backend of the application, responsible for running the **Flask web server**. It loads the **trained deep learning model (evgg.h5)**, processes incoming requests, and serves HTML templates to users. When a user uploads an eye image, app.py preprocesses it and passes it through the deep learning model to predict the disease. The output is then displayed on the web page. This file also manages routing for different pages such as home, image upload, and the about section.

app.py

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

app = Flask(__name__)

UPLOAD_FOLDER = "static"
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

# Load the trained model
model = load_model("evgg.h5")

# Define disease categories
index = ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']

# Ensure upload folder exists
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

@app.route("/", methods=["GET", "POST"])
def upload_file():
    if request.method == "POST":
        # Get uploaded file
        file = request.files["file"]
        if file:
            file_path = os.path.join(app.config["UPLOAD_FOLDER"], file.filename)
            file.save(file_path)

            # Process the image
            img = image.load_img(file_path, target_size=(224, 224))
            x = image.img_to_array(img)
            x = np.expand_dims(x, axis=0)

            # Make prediction
            preds = model.predict(x)
            pred_class = np.argmax(preds, axis=1)[0]
            result = index[pred_class]

            return render_template("result.html", filename=file.filename, result=result)

    return render_template("index.html")

@app.route("/display/<filename>")
def display_image(filename):
    return f'<img src="{os.path.join(app.config["UPLOAD_FOLDER"], filename)}" width="300">'

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

Frontend

A simple and intuitive user interface built using HTML and CSS, allowing users to upload images and view predictions.

The web interface includes a file upload feature and a display area to show the uploaded image and the predicted disease class.

The /templates/ folder contains all the HTML files that structure the web interface of the application. These HTML files define the layout and user interaction:

index.html serves as the homepage. It introduces the project, provides navigation links, and allows users to proceed to the image upload page.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload Image</title>
    <style>
        a{
            display: inline-block;
            width: 80px;
            text-decoration: none;
            font-size: 18px;
            color: rgb(41, 90, 154)
        }
    </style>
</head>

<body style="margin: 0px;padding: 0px;font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;">
    <div style="background-color: lightblue;">
        <div
            style="background-color: lightblue;display: flex;align-items: center;justify-content: space-around;color: rgb(41, 90, 154);height: 80px;">
            <div>
                <h1>LIVEDOC</h1>
            </div>
            <div>
                <a href="">About</a>
                <a style="">Help</a>
                <a href="">Contact</a>
                <button style="background-color:lightblue ;border-color: darkblue;border-width: 3px;border-radius: 10px;cursor: pointer;"><a
                        style="font-size: 18px;display: inline-block;
                        width: 80px;
                        text-decoration: none;
                        font-weight: bold;
                        color: rgb(41, 90, 154)">Predict</a></button>
            </div>
        </div>
        <h1 style="text-align: center;padding:20px;padding-top: -30px;color: darkblue;font-weight: bolder;">Predict
            Diesease</h1>
    </div>
    <div style="display: flex;align-items: center;justify-content: center;">
        <div style="margin-left: 70px;">
            <img style="width: 70%;" src="C:\Users\NEERAJ KUMAR\OneDrive\Desktop\Flask\static\Screenshot 2025-03-06 145556.png" alt="">
        </div>
        <div style="margin-right: 100px;">
            <h2 style="text-align: center;padding:20px;color: darkblue;font-weight: bolder;">Please Upload The Image</h2>
            <form action="/" method="post" enctype="multipart/form-data">
                <input type="file" name="file" required>
                <button style="background-color: darkblue;color: white; padding: 10px 30px;border-radius: 10px;" type="submit">Predict</button>
            </form>
        </div>
    </div>
</body>

</html>
```

output.html displays the prediction results from the model after analyzing the uploaded image, showing whether the eye is normal or affected by cataract, diabetic retinopathy, or glaucoma.

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Detection Result</title>
    <style>
        div a{
            display: inline-block;
            width: 80px;
            text-decoration: none;
            font-size: 18px;
            color: rgb(41, 90, 154)
        }
    </style>
</head>
<body style="margin: 0px;padding: 0px;font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;">
    <div style="background-color: lightblue;">
        <div
            style="background-color: lightblue;display: flex;align-items: center;justify-content: space-around;color: rgb(41, 90, 154);height: 80px;">
            <div>
                <h1>LIVEDOC</h1>
            </div>
            <div>
                <a href="">About</a>
                <a style="">Help</a>
                <a href="">Contact</a>
                <button style="background-color: lightblue ;border-color: darkblue;border-width: 3px;border-radius: 10px;cursor: pointer;"><a
                        style="font-size: 18px;display: inline-block;
                        width: 80px;
                        text-decoration: none;
                        font-weight: bold;
                        color: rgb(41, 90, 154)">Predict</a></button>
            </div>
        </div>
        <h1 style="text-align: center;padding:20px;padding-top: -30px;color: darkblue;font-weight: bolder;">Predicted
            Output</h1>
    </div>
    <p style="text-align: center;color: darkblue;font-size: 30px;font-weight: bold;"><strong>Predicted Disease:</strong> {{ result }}</p>
    <p style="text-align: center;"><a style="display: inline-block; text-align: center;color: darkblue;font-size: 20px;font-weight: bold;text-decoration: none;" href="/">Upload Another Image</a></p>
</body>
</html>
```

Static/ - Styling, Images, and Uploaded Files

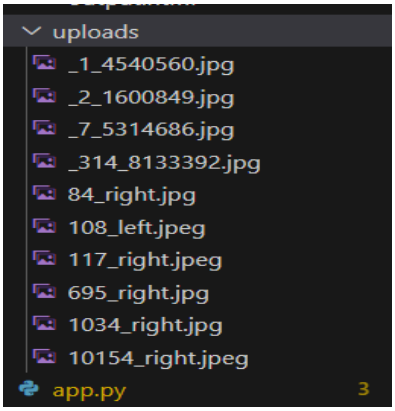
The /static/ folder contains all **static files** used in the application, such as CSS styles, images, and uploaded files.

- styles.css is responsible for designing and improving the appearance of the website. It defines the layout, color schemes, button styles, and overall user interface aesthetics.

Uploads/

/uploads/ is a subfolder where user-uploaded images are temporarily stored before being processed by the deep learning model.

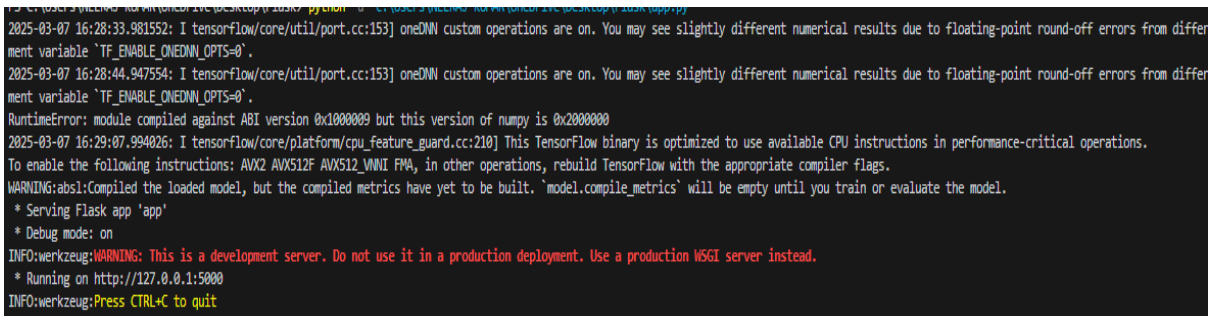
- /images/ contains preloaded images used for the website's design, such as background images, icons, or example eye disease images. These files enhance the UI, making the web application visually appealing and easy to use.



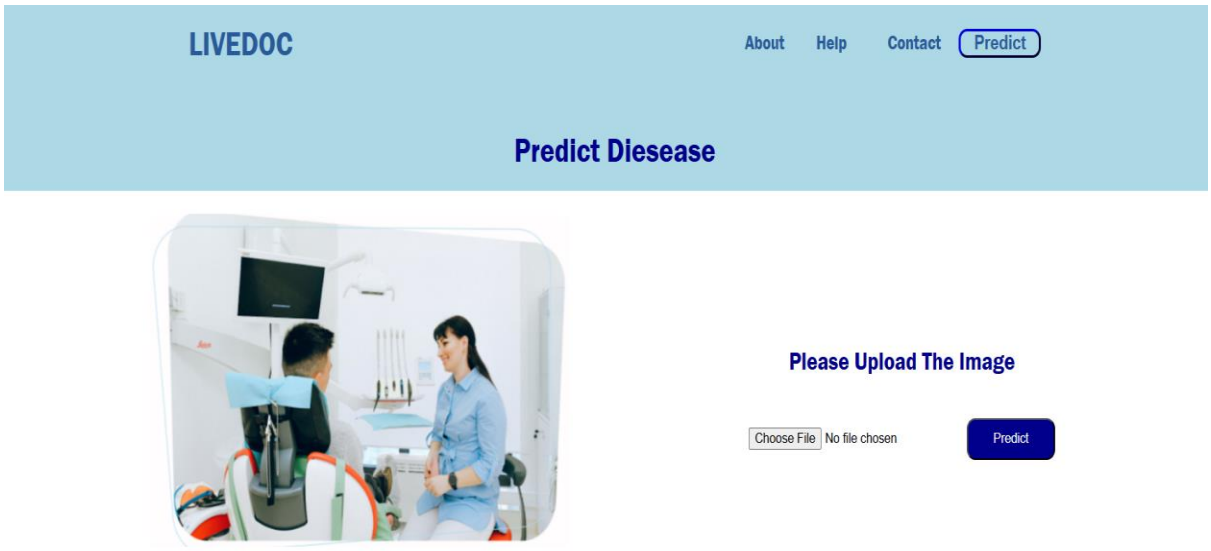
Running the Code

The execution of the project begins with training the deep learning model and then deploying it via a Flask web application. The process starts by running the train_model.py script, which loads the dataset, preprocesses the images, and trains a convolutional neural network using transfer learning techniques such as VGG19. Once training is completed, the model is saved as evgg.h5, which serves as the core of the prediction system.

After the model is trained and saved, app.py is executed to launch the Flask web server. The Flask application initializes by loading evgg.h5, setting up routes, and rendering HTML templates. When a user accesses the web application through http://127.0.0.1:5000/, they are directed to the homepage, index.html.



From there, users can navigate to the image upload page, index.html, where they can select an eye image for disease detection. Once an image is uploaded, it is temporarily stored in the /static/uploads/ folder and then passed to the deep learning model for analysis.The model processes the image, predicts the category (Normal, Cataract, Diabetic Retinopathy, or Glaucoma), and sends the result to output.html, where the prediction is displayed. Users can then choose to upload another image or navigate to the about.html page to learn more about the project. The entire process ensures seamless interaction between the backend and frontend components, making it an efficient and user-friendly system for detecting eye diseases.



Conclusion

This project demonstrates the effectiveness of deep learning in detecting eye diseases using medical images. By leveraging transfer learning and Flask, we have built a scalable and user-friendly system for automated eye disease classification. The system can assist healthcare professionals in diagnosing eye diseases more efficiently and accurately.

Future work can focus on:

- Improving model accuracy by using larger and more diverse datasets.
- Expanding the system to detect additional eye diseases.
- Deploying the application on cloud platforms for wider accessibility.
- Incorporating explainable AI techniques to provide insights into the model's predictions.

This project highlights the potential of deep learning in revolutionizing healthcare and improving patient outcomes.