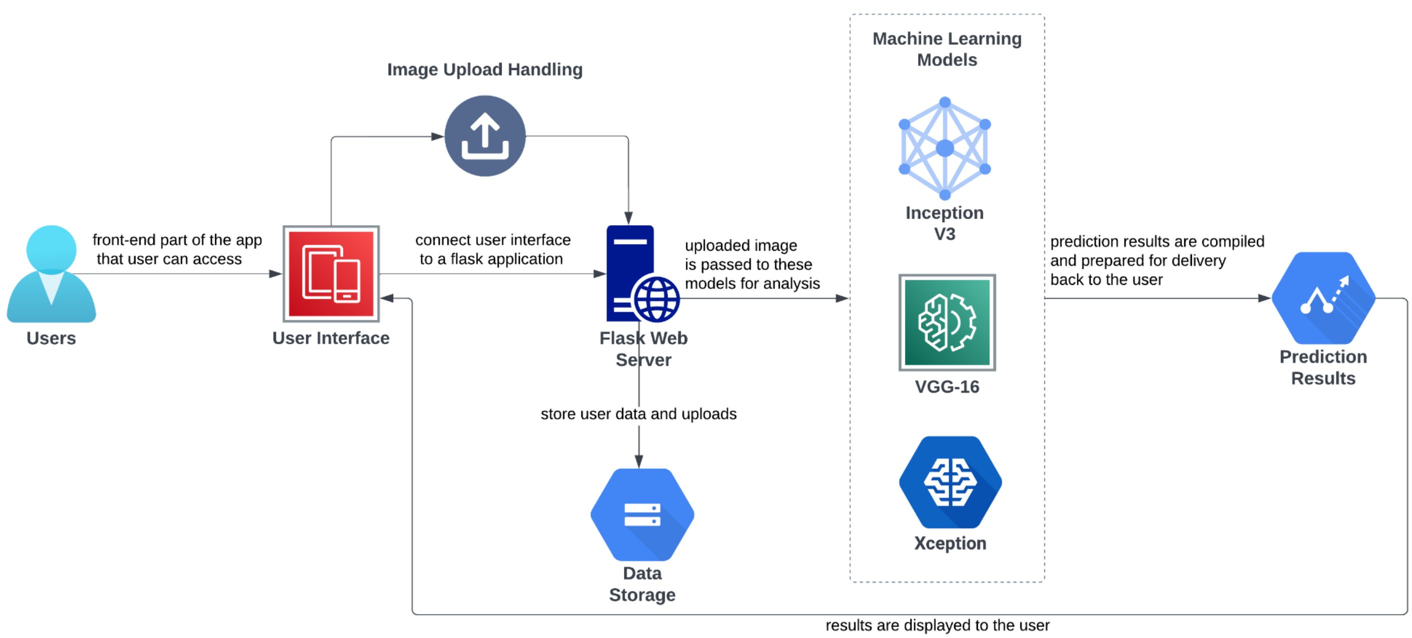
Deep Learning Model For Eye Disease Prediction

1) Project Idea:

Our project introduces an AI-driven tool that diagnoses eye diseases by analyzing retina images. Utilizing deep learning models like Inception V3, VGG16, and Xception, the system provides accurate assessments, distinguishing between ailments such as cataracts and diabetic retinopathy.

Hosted on a user-friendly website, patients upload images, and in moments, receive a diagnostic report. This innovative solution aims to enhance early detection and improve eye care accessibility.

2) Technical Architecture:



3) **Learning Outcomes:**

By working on this project we were able to learn following things:

- Developing a deep learning model to predict eye diseases by processing retina images with Python and leveraging libraries such as TensorFlow and Keras.

- Employing image processing techniques and model performance evaluation to enhance diagnostic accuracy.

- Implementing time series analysis to track disease progression over time, using advanced algorithms.

- Deploying the predictive model in a web environment using Flask, allowing for real-time user interaction and diagnosis delivery.

4) **Project Flow:**

First user interacts with the UI deployed using Flask in order to select the date.

Then the selected date is sent to the model in the backend which has already been trained on the historical data, the model then uses that historical data as reference to predict the eye disease of the image uploaded by the user.

Then in the final part the out produced by the model is carried to UI where it is showcased as result to the user.

To achieve this flow, we divide our project into **six major phases** which are as follows:

Phase one – Setting up the environment

In this phase we created a separate anaconda environment for the project, in-order to avoid any clashes in requirements, then we installed the Tensorflow, Flask, Pillow and other libraries using ‘pip install’ command.

Phase two – Data collection

Once the environment was ready, our first task was to collect the historical data

Phase three – Data preprocessing and data visualization

In this phase we visualized the time series data and cleaned the data and made it ready for the algorithm

Phase four – Model building

Once the data was ready it was time to feed the data to our VGG16 model in order to train it, this is what was done in this phase

Phase five – Deployment

After the model was ready we finally deployed the model using HTML,CSS and Flask

<u>Let's now look at these phases one by one in detail</u>

5) Setting up the environment:

This phase was simple and straightforward, we started by downloading anaconda navigator, then installed the Jupyter notebook.

Once this was done we began installing required libraries- Tensorflow, Flask, Pillow and other libraries using 'pip install' command.

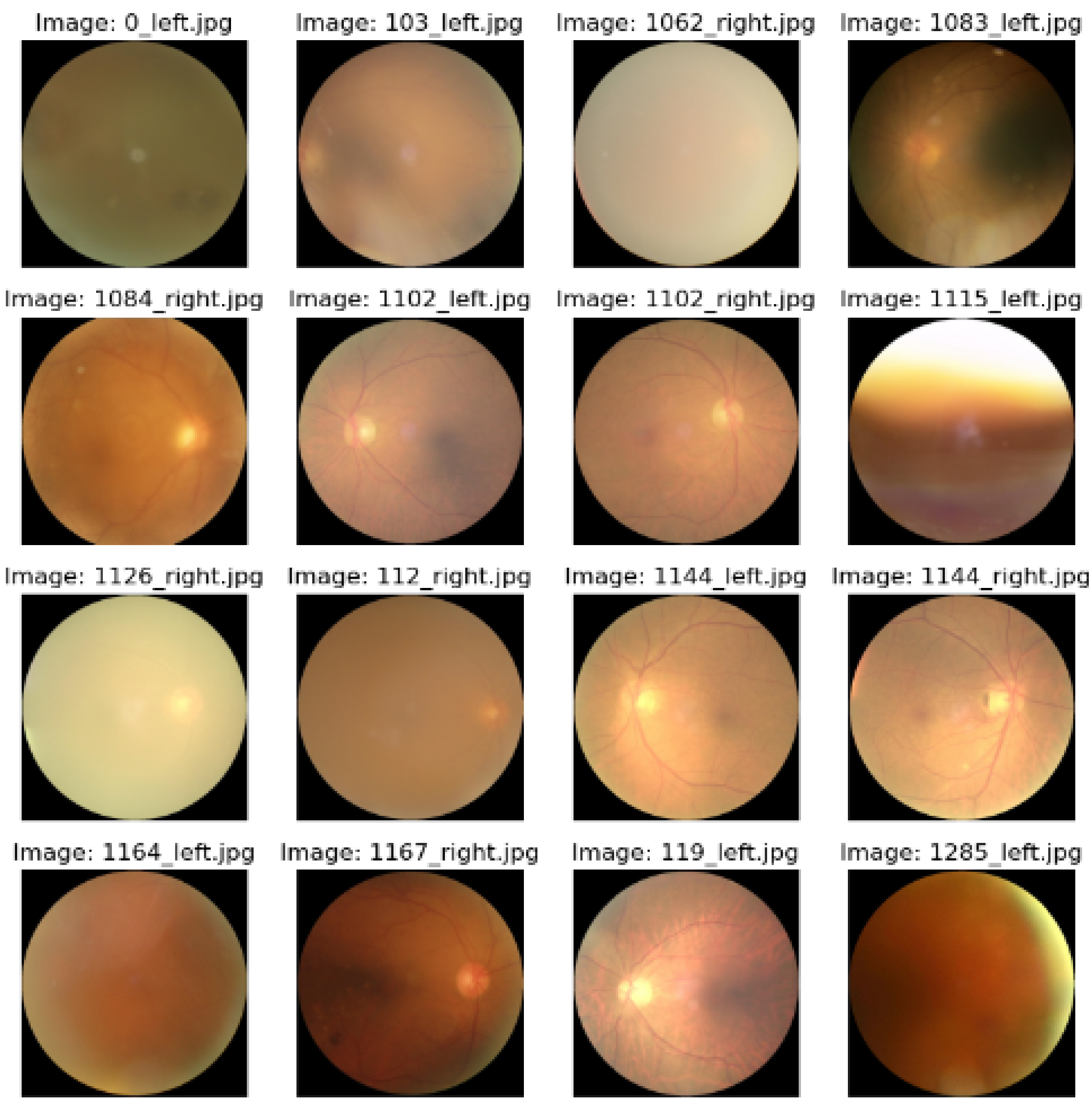
6) Data collection:

The data collected in this project comes directly from Kaggle link provided to us-
https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification

The dataset consists of Normal, Diabetic Retinopathy, Cataract and Glaucoma retinal images where each class have approximately 1000 images. These images are collected from various sources like IDRiD, Oculur recognition, HRF etc.

Dataset

In [12]:
```python
folder_path = "dataset/cataract"
image_filenames = os.listdir(folder_path)
image_filenames = image_filenames[:16]
fig, axes = plt.subplots(4, 4, figsize=(8, 8))

for i, filename in enumerate(image_filenames):
    image_path = os.path.join(folder_path, filename)
    img = Image.open(image_path)
    axes[i // 4, i % 4].imshow(img)
    axes[i // 4, i % 4].set_title(f"Image: {filename}")
    axes[i // 4, i % 4].axis("off")  # Turn off axis labels

plt.tight_layout()
plt.show()
```



7) Data Augmentation:

Now the preprocessed data is augmented to enrich the dataset and improve model robustness.

Data augumentation

```
In [34]: train_datagen = ImageDataGenerator(
             rescale=1./255,
             shear_range=0.2,
             zoom_range=0.2,
             horizontal_flip=True,
             vertical_flip=True,
             validation_split=0.2
         )
```

```
In [35]: train_set = train_datagen.flow_from_directory(
             data_path,
             target_size=IMG_SIZE[:2],
             batch_size=BATCH_SIZE,
             class_mode='categorical',
             subset='training',
             shuffle=True,
         )
```

```
Found 3376 images belonging to 4 classes.
```

```
In [36]: test_set = train_datagen.flow_from_directory(
             data_path,
             target_size=IMG_SIZE[:2],
             batch_size=BATCH_SIZE,
             class_mode='categorical',
             subset='validation',
             shuffle=False,
         )
```

```
Found 841 images belonging to 4 classes.
```

8) Using Transfer Learning:

Now we use pre-trained models such as Inception V3, VGG-16, and Xception for feature extraction and model enhancement.

Transfer Learning in VGG16

In [37]:
```
vgg16 = VGG16(input_shape=IMG_SIZE, weights='imagenet', include_top=False)
```

In [38]:
```
for layer in vgg16.layers:
    layer.trainable = False
```

In [39]:
```
folders = glob('dataset/*')
```

In [40]:
```
x = Flatten()(vgg16.output)
```

In [41]:
```
prediction = Dense(len(folders), activation='softmax')(x)

model = Model(inputs=vgg16.input, outputs=prediction)
```

In [42]:
```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

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

 flatten_1 (Flatten)         (None, 25088)             0

 dense_1 (Dense)             (None, 4)                 100356

=================================================================
Total params: 14815044 (56.51 MB)
Trainable params: 100356 (392.02 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

9) Implementing Callbacks:

Now we are integrating callbacks to handle real-time events within the application, such as triggering the analysis of retina images once uploaded and subsequently displaying diagnostic results to the user interface automatically..

Callbacks

In [43]:
```
BUR_callback = BackupAndRestore(backup_dir="./temp1/backup", save_freq='epoch',
                                delete_checkpoint=True,)
```

In [44]:
```
early_callback = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True,
                               start_from_epoch=2)
```

In [45]:
```
csvlog_callback = CSVLogger(
    'logger.csv', separator=',', append=False
)
```

In [46]:
```
import tensorflow as tf

decay_steps = 3
decay_rate = tf.math.exp(-0.1)

def lr_scheduler(epoch, lr):
    if epoch % decay_steps == 0 and epoch > 0:
        return lr * decay_rate
    return lr

lr_callback = LearningRateScheduler(lr_scheduler, verbose=1)
```

In [47]:
```
checkpoint_filepath = '/temp1/checkpoint'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

10) Model Compilation And Fitting:

Now we are compiling and fitting the model, which involves setting up the loss function, optimizer, and metrics for training, and then training the model on the preprocessed data to learn patterns relevant for eye disease prediction.

Compilation and Fitting

In [50]:
```
model.compile(
  loss='categorical_crossentropy',
  optimizer=Adam(learning_rate=0.0005),
  metrics=['accuracy'],
)
```

In [51]:
```
r = model.fit(
  train_set,
  validation_data=test_set,
  epochs=30,
  steps_per_epoch=len(train_set),
  validation_steps=len(test_set),
  callbacks=[BUR_callback, csvlog_callback, lr_callback]
)
```

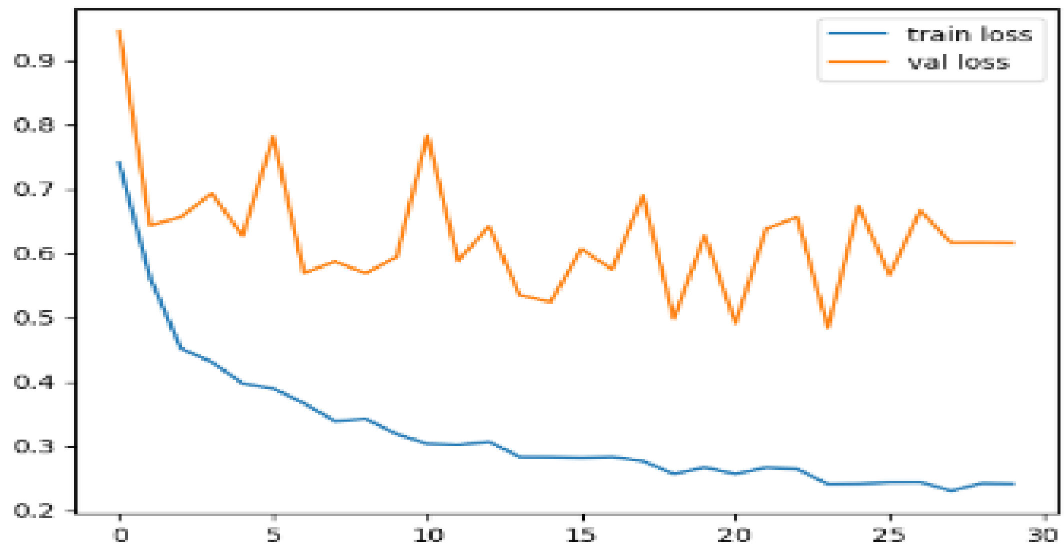
```
Epoch 1: LearningRateScheduler setting learning rate to 0.0005000000237487257.
Epoch 1/30
106/106 [==============================] - 278s 3s/step - loss: 0.7398 - accuracy: 0.6943 - val_loss: 0.9448 - val_accuracy: 0.
6409 - lr: 5.0000e-04

Epoch 2: LearningRateScheduler setting learning rate to 0.0005000000237487257.
Epoch 2/30
106/106 [==============================] - 262s 2s/step - loss: 0.5617 - accuracy: 0.7793 - val_loss: 0.6428 - val_accuracy: 0.
7503 - lr: 5.0000e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.0005000000237487257.
Epoch 3/30
106/106 [==============================] - 259s 2s/step - loss: 0.4514 - accuracy: 0.8303 - val_loss: 0.6556 - val_accuracy: 0.
7372 - lr: 5.0000e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.00045241872430779040.
Epoch 4/30
106/106 [==============================] - 267s 3s/step - loss: 0.4303 - accuracy: 0.8377 - val_loss: 0.6925 - val_accuracy: 0.
7265 - lr: 4.5242e-04

Epoch 5: LearningRateScheduler setting learning rate to 0.00045241872430779040.
Epoch 5/30
106/106 [==============================] - 266s 3s/step - loss: 0.3970 - accuracy: 0.8546 - val_loss: 0.6268 - val_accuracy: 0.
7455 - lr: 4.5242e-04

Epoch 6: LearningRateScheduler setting learning rate to 0.00045241872430779040.
Epoch 6/30
106/106 [==============================] - 271s 3s/step - loss: 0.3890 - accuracy: 0.8498 - val_loss: 0.7806 - val_accuracy: 0.
7134 - lr: 4.5242e-04

Epoch 7: LearningRateScheduler setting learning rate to 0.0004093653988093138.
```

11) Data Visualization:

Now we are plotting the loss and accuracy for training and validation metrics, visualizing the model's performance over epochs to assess and fine-tune its predictive accuracy and generalization ability.

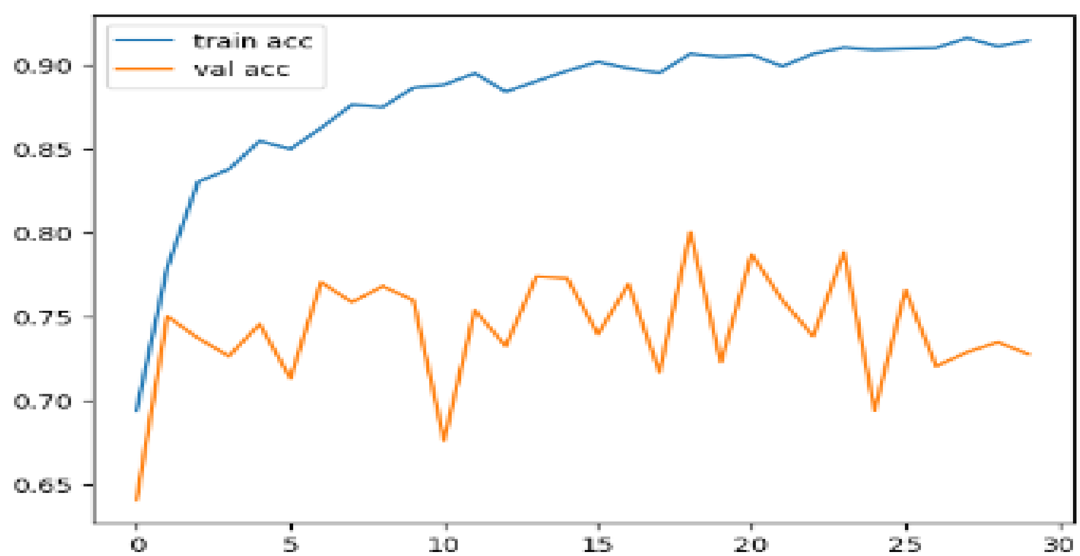
Data Visualization

In [52]:
```
# plotting the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```



<Figure size 640x480 with 0 Axes>

In [53]:
```
# plotting the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```



<Figure size 640x480 with 0 Axes>

12) Creating h5 File:

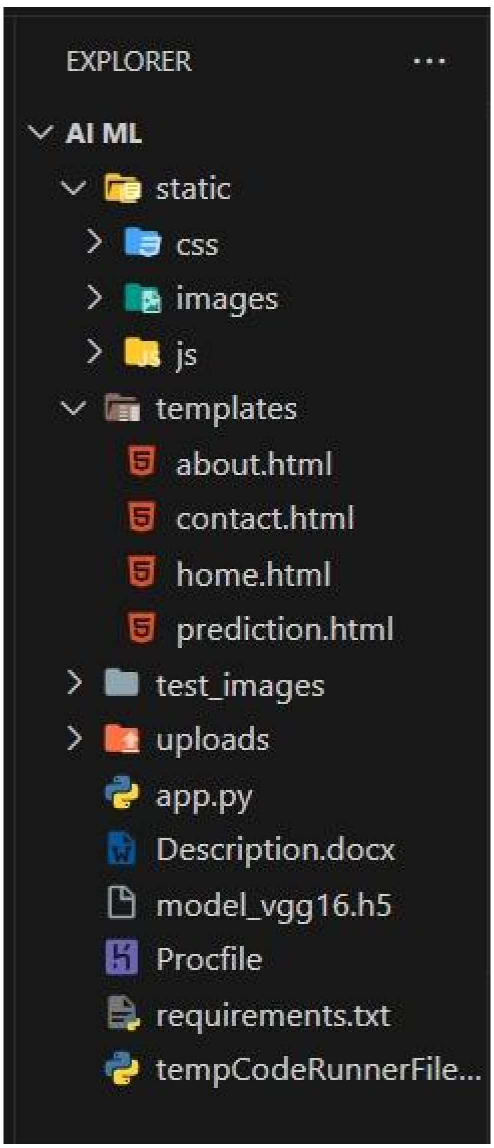
Now we are creating an .h5 file, which serves to save the trained model, encapsulating its architecture, weights, and training configuration, ensuring it can be easily reloaded and deployed for future predictions.

Creating h5 file

In [54]:
```
from tensorflow.keras.models import load_model
model.save('model_vgg16.h5')
```

13) Model Deployment:

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.

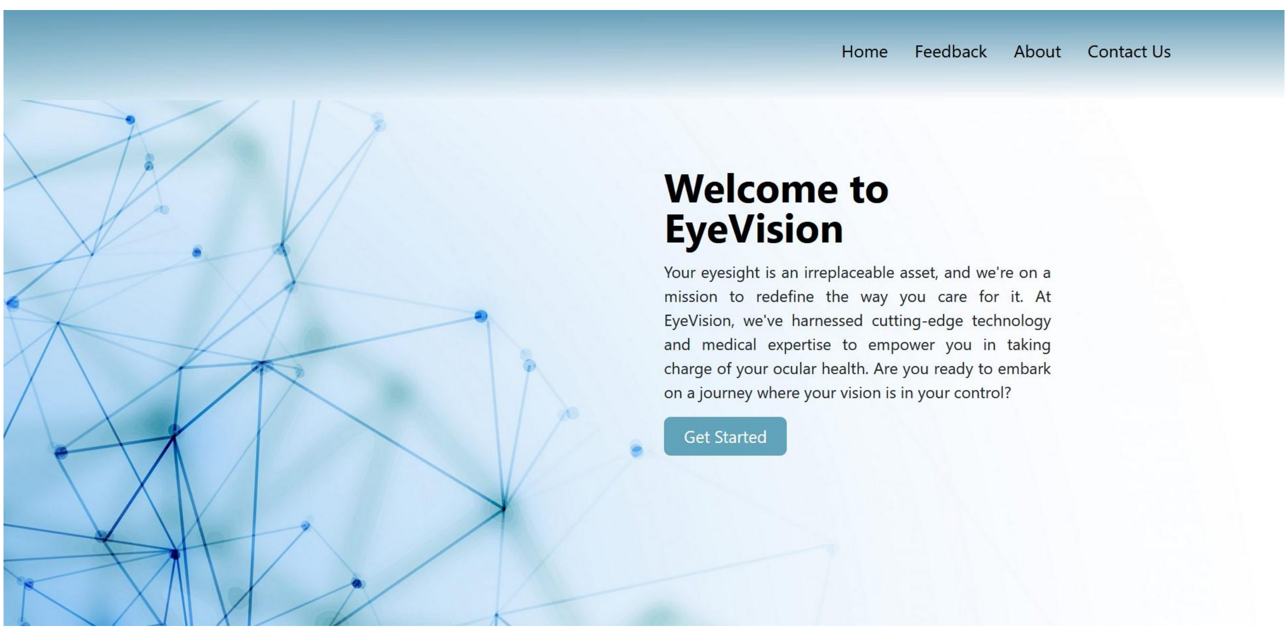


In the **flask application**, the input parameters are taken from the HTML page These factors are then given to the model to predict the type of eye disease and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Get Started Upload Image” button, the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

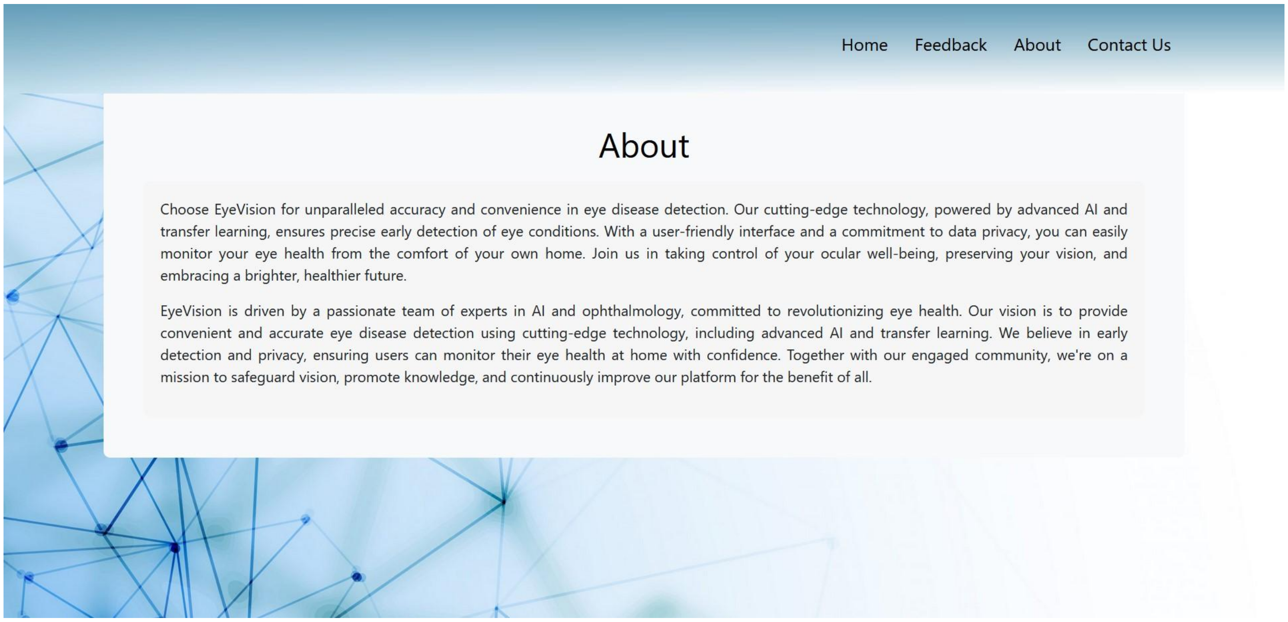
I] Create HTML Pages

- o We use HTML to create the front end part of the web page.
- o Here, we have created 4 HTML pages- home.html, about.html, contact.html, and prediction.html
- o home.html displays the home page
- o about.html displays an introduction about the project
- o prediction.html gives the user the interface to check for eye diseases for the images uploaded by the user
- o contact.html gives the user a platform to ask us their queries and share their valuable feedback.
- o We also use JavaScript and CSS to enhance our functionality and view of HTML page

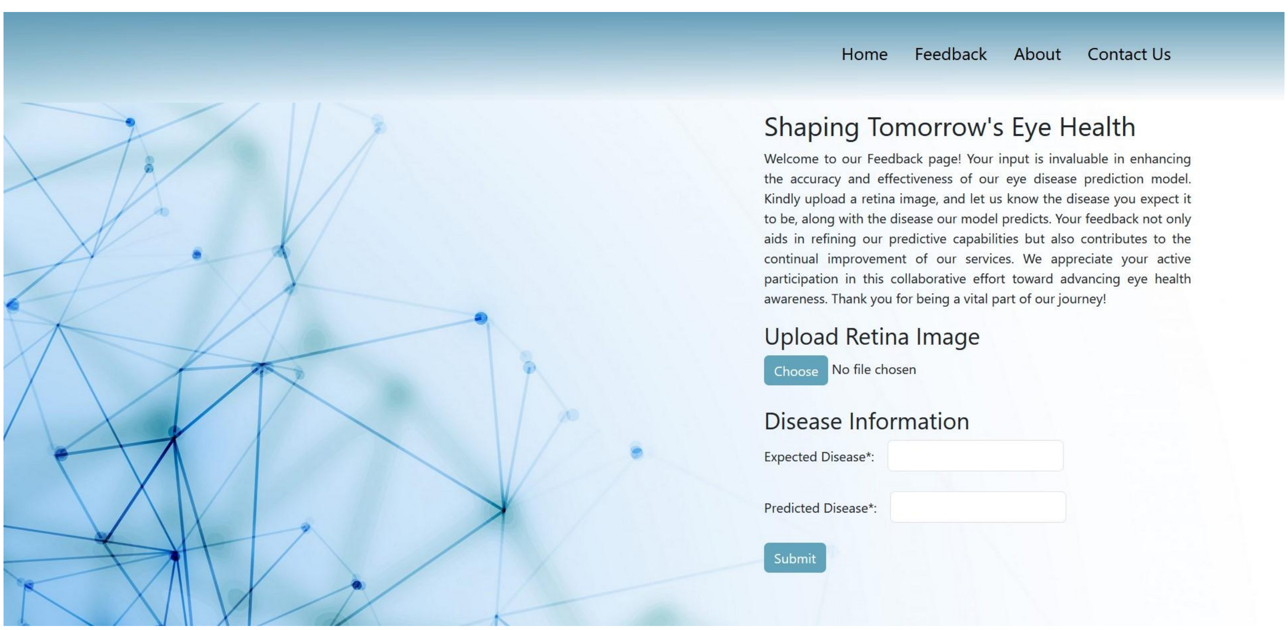
A} Home:-



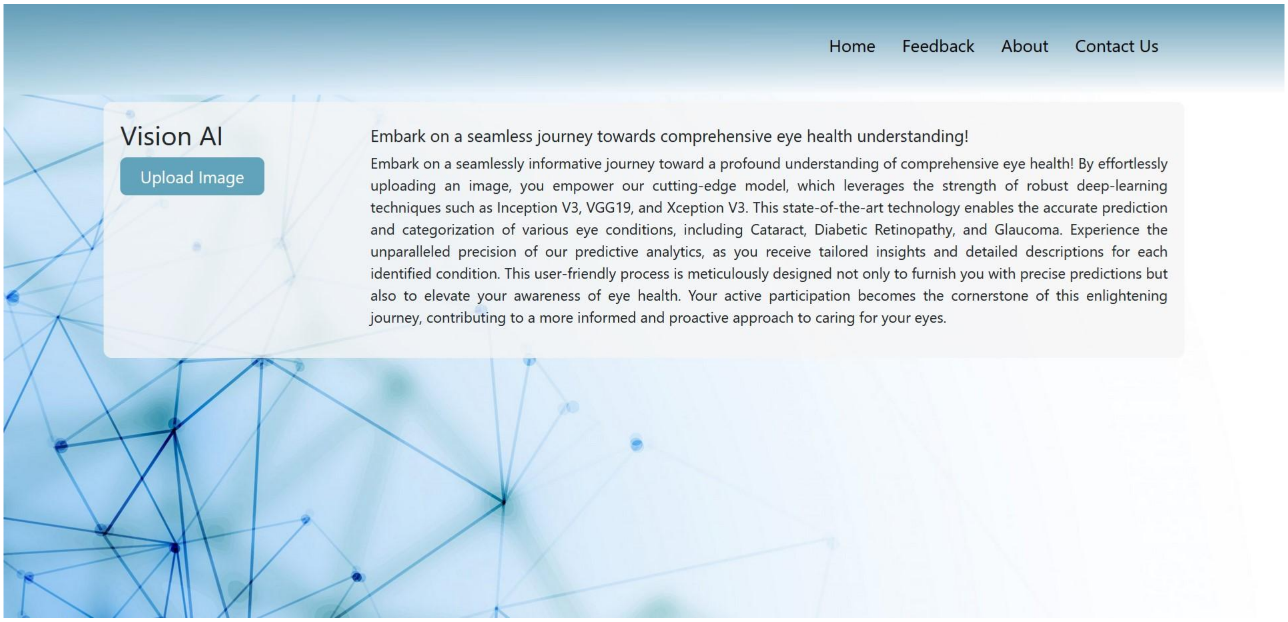
B} About Us:-



C} Contact Us:-



D} Prediction:-



II] Build python code

A} Importing Libraries :-

The first step involves importing the necessary libraries for the program.

```
from __future__ import division, print_function
import sys
import os
import glob
import re
import numpy as np
import tensorflow as tf
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
```

B} Creating our Flask Application and Loading our Model :-

Initializing the Flask app and loading the pre-trained model using the load_model method.

```
# Define a flask app
app = Flask(__name__)

# Model saved with Keras model.save()
MODEL_PATH ='model_vgg16.h5'

# Load your trained model
model = load_model(MODEL_PATH)
```

C} Routing to the HTML Page :-

Setting up routes to different HTML pages using Flask.

```
@app.route('/', methods=['GET'])
def home():
    return render_template('home.html')

@app.route('/about', methods=['GET'])
def about():
    return render_template('about.html')

@app.route('/contact', methods=['GET'])
def contact():
    return render_template('contact.html')

@app.route('/prediction', methods=['GET'])
```

```
def prediction():
    prediction_results = [
        {
            'disease_name': 'Glaucoma',
            'disease_description': 'Glaucoma is a group of eye conditions
that damage the optic nerve...'
        },
        {
            'disease_name': 'Cataract',
            'disease_description': 'Cataracts are a clouding of the lens in
the eye which leads to a decrease in vision...'
        },
    ]
    return
render_template('prediction.html',prediction_results=prediction_results)

@app.route('/get_started', methods=['POST'])
def get_started():
    return redirect(url_for('prediction'))
```

D} Showcasing Prediction on UI :-

Processing the uploaded image and displaying the prediction result.

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

```
        preds = model_predict(file_path, model)
        result=preds
        return result
    return None
```

E} Predicting the Results :-

The image file uploaded via the UI is processed and analyzed using the deep learning model.

F} Finally, Run the Application :-

Running the Flask application on a local server.

```
if __name__ == '__main__':
    app.run(port=5001,debug=True)
```

III] Run the application

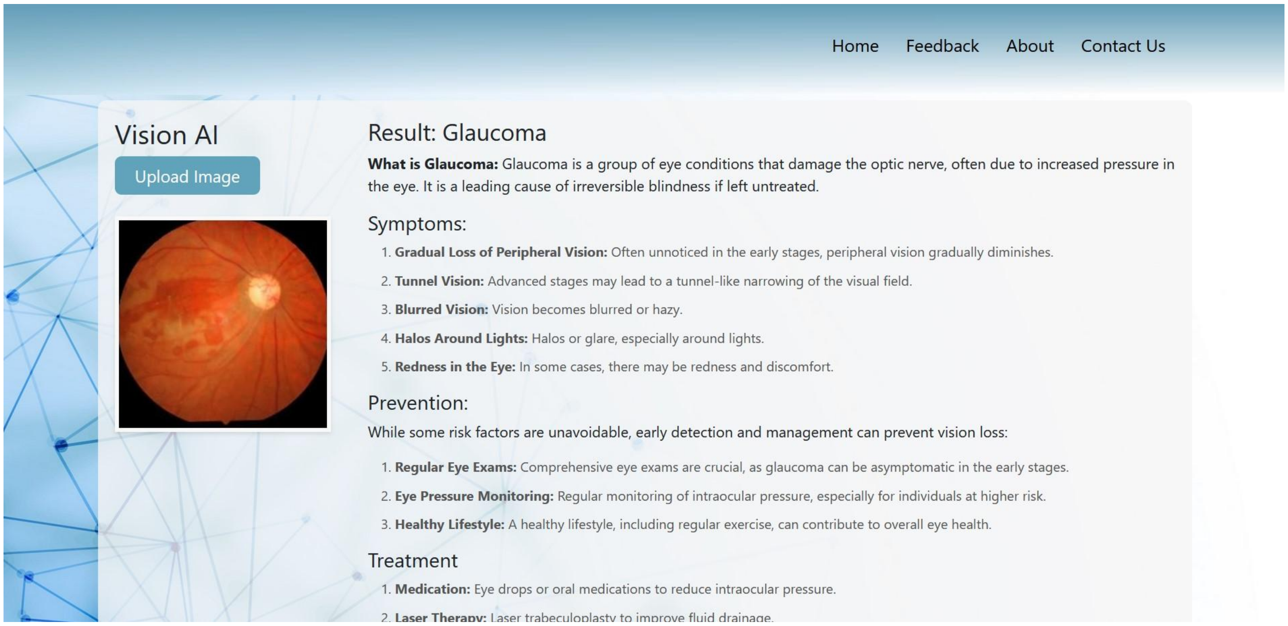
- Navigate to the folder where your app.py resides.
- Now type "python app.py" command.
- It will show the local host where your app is running on http://127.0.0.1.5001/
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.

```
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5001
Press CTRL+C to quit
 * Restarting with stat
2023-11-16 22:24:20.625263: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performan
ce-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Debugger is active!
 * Debugger PIN: 193-430-037
```

Navigate to the localhost (http://127.0.0.1:5001/)where you can view your web page.

13) Results

I] Test Case 1



II] Test Case 2