PROJECT DOCUMENTATION

Deep Learning Model For Eye Disease Prediction

1. Introduction

1.1 Project Overview

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

The Eye Disease Detection Model aims to develop an AI-based system for accurate and early detection of various eye diseases, such as glaucoma, cataract, diabetic retinopathy, and others. The project focuses on leveraging deep learning techniques to analyze eye images and classify them into specific disease categories.

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

1.2 Purpose

The primary objective of this project is to create an efficient and reliable system that assists medical practitioners in diagnosing eye diseases at an early stage. By automating the detection process, it aims to improve patient care, reduce human error, and facilitate timely interventions. For this purpose, using Convolution Neural Network and Transfer Learning based approach is used to design a reliable and adaptable model that would accept an image of a patient's retina scan and then detect if the patient belongs to one of the following categories : glaucoma', 'cataract', 'normal', 'diabetic_retinopathy’.

2. Literature Survey

2.1 Existing Problem

A myriad of eye diseases, stemming from factors like genetics, aging, environment, or underlying health issues can compromise vision over a prolonged period of time. The need in the current medical field is to develop a reliable and adaptable system to help patients by allowing healthcare professionals to diagnose and manage eye diseases, emphasizing accuracy, efficiency and early detection. Although many solutions exist already, but by having a newer, faster and less resource intensive method can make sure that the disease can be detected at its earliest and more people could be diagnosed easily, making the process of treatment more beneficial for the patient.

2.2 Problem Statement Definition

Defining the need for an AI-driven eye disease detection model that overcomes the limitations of manual diagnosis, aiming for high accuracy, speed, and scalability.

3. Ideation & Proposed Solution

3.1 Empathy Map Canvas

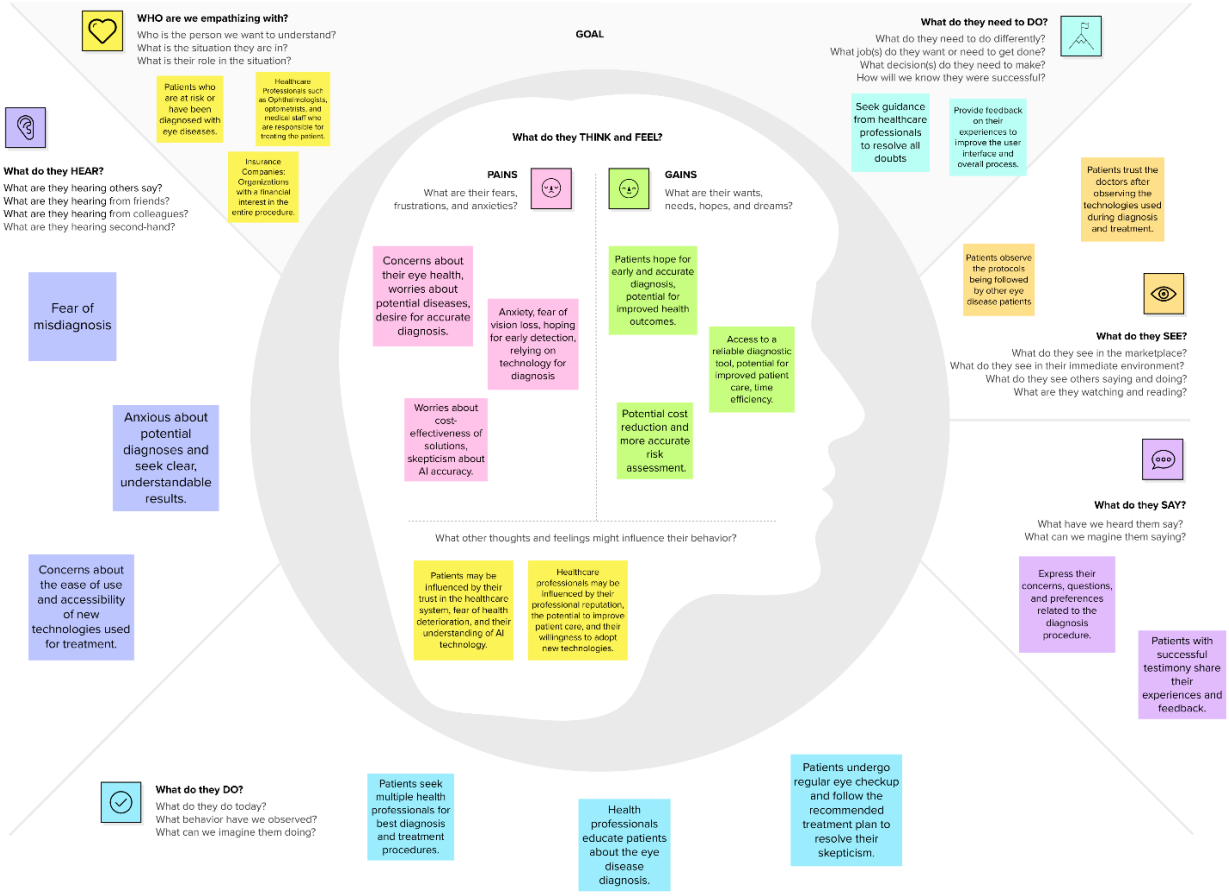
Understanding the needs and perspectives of stakeholders: patients, medical professionals, and AI practitioners in the context of eye disease diagnosis.

Empathy Map Canvas:

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviors and attitudes.

It is a useful tool to help teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges. URL for our Empathy Map:

https://app.mural.co/t/harshsworkspace5817/m/harshsworkspace5817/1699424110745/106d4e5ca83b0f90387aa6114196d932206cd2d5?sender=u1a765872020f680dc3b76204

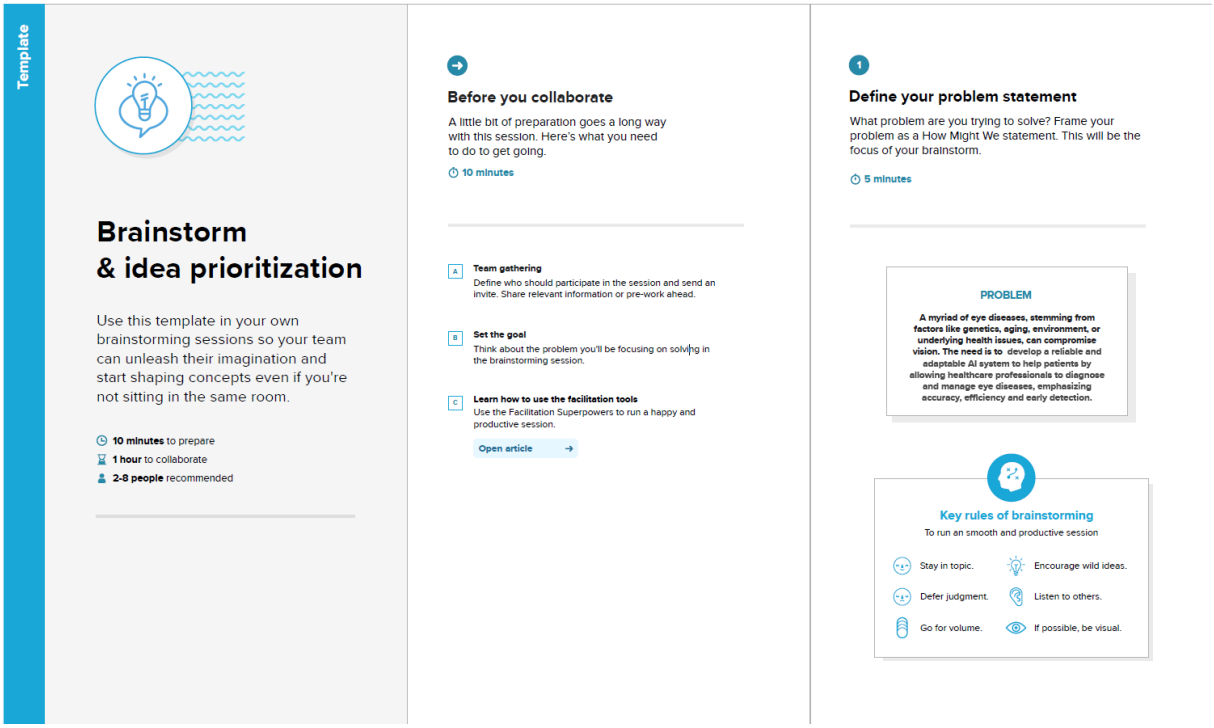


3.2 Ideation & Brainstorming

Exploration of various AI models (e.g., CNNs, transfer learning) and techniques suitable for image analysis. Brainstorming sessions to outline the architecture and components of the proposed eye disease detection model.
Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

https://app.mural.co/t/harshsworkspace5817/m/harshsworkspace5817/1699610529891/51de21bb7053a3df522472c7555107c16d9abbc1?sender=u1a765872020f680dc3b76204

Step-1: Team Gathering, Collaboration and Select the Problem Statement



Finalized Problem Statement:

A myriad of eye diseases, stemming from factors like genetics, aging, environment, or underlying health issues, can compromise vision. The need is to develop a reliable and adaptable AI system to help patients by allowing healthcare professionals to diagnose and manage eye diseases, emphasizing accuracy, efficiency and early detection.

Step-2: Brainstorm, Idea Listing and Grouping :

All the team members came up with possible solutions based on their understanding and interpretation of the newly defined problem statement.

2

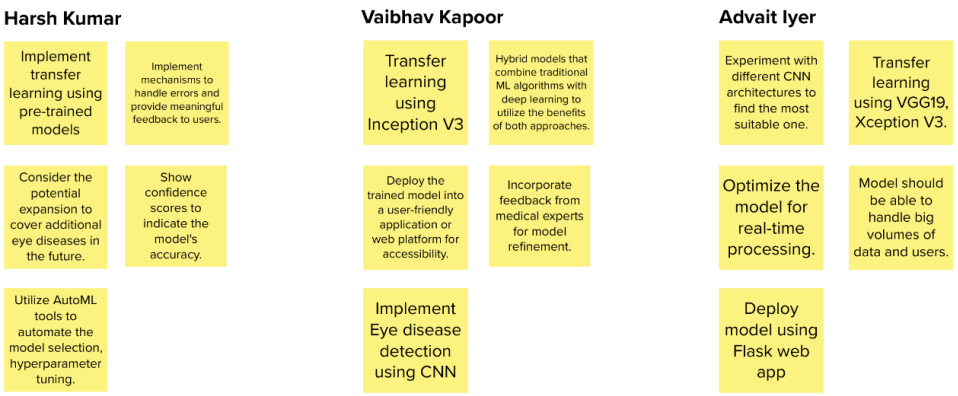
Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!



These 15 ideas are then clustered into following 6 clusters as shown in following figure:

3

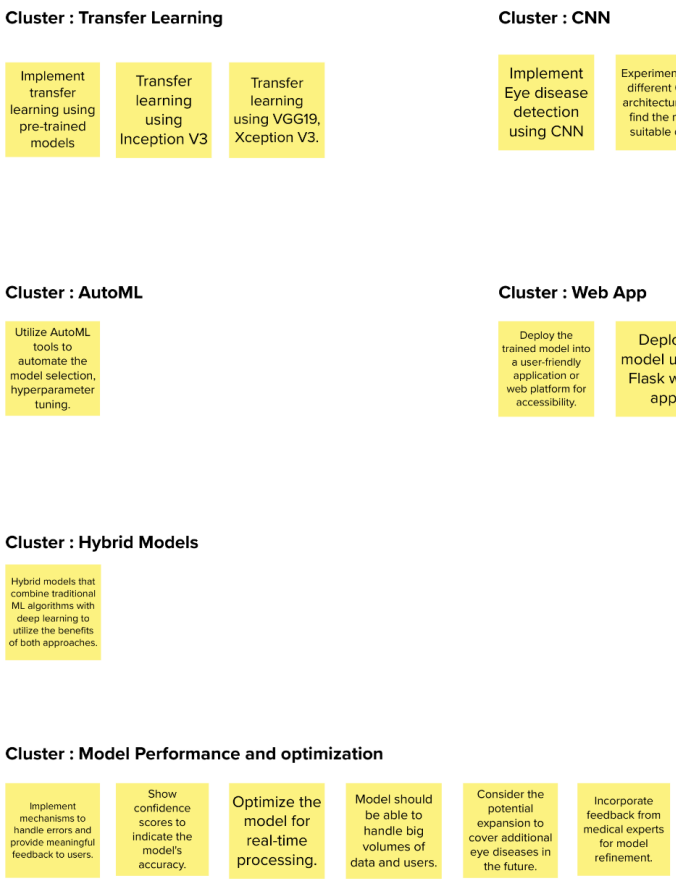
Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.



RANKING THE IDEAS :

The 13 ideas are ranked based on 4 factors : Impact, Feasibility, Cost and Alignment with goal.

SCORES BY HARSH:

CLUSTER	IDEA	SCORES BY HARSH

		Impact	Feasible	Cost	Alignment with goal	Total
Transfer Learning	· Implement transfer learning using pre-trained models · Transfer learning using Inception V3 · Transfer learning using VGG19, Xception V3.	5	4.5	2	5	16.5
CNN	· Implement Eye disease detection using CNN · Experiment with different CNN architectures to find the most suitable one.	4.5	4	3	5	16.5
AutoML	Utilize AutoML tools to automate the model selection, hyperparameter tuning.	3.5	3.5	2	5	14

WebApp	· Deploy the trained model into a user-friendly application or web platform for accessibility. · Deploy model using Flask web app	4.5	4	2.5	4	15
Hybrid Models	Hybrid models that combine traditional ML algorithms with deep learning to utilize the benefits of both approaches.	3	2.5	3	5	13.5
Model Performance and Optimization	Implement mechanisms to handle errors and provide meaningful feedback to users.	3	2	4	3	12
Model Performance and Optimization	Show confidence scores to indicate the model's accuracy.	3	5	1	3.5	12.5
Model Performance and Optimization	Optimize the model for real-time processing.	2	2	3.5	3	10.5

Model Performance and Optimization	Model should be able to handle big volumes of data and users.	3.5	3	3.5	4.5	14.5
Model Performance and Optimization	Consider the potential expansion to cover additional eye diseases in the future.	2	3	4	4	13
Model Performance and Optimization	Incorporate feedback from medical experts for model refinement.	4	2.5	3.5	4	14

SCORES BY ADVAIT:

CLUSTER	IDEA	SCORES BY ADVAIT				
		Impact	**Feasible**	**Cost**	**Alignment with goal**	**Total**
Transfer Learning	· Implement transfer learning using pre-trained models · Transfer learning using Inception V3 · Transfer learning using VGG19, Xception V3.	5	5	3	5	18

CNN	· Implement Eye disease detection using CNN · Experiment with different CNN architectures to find the most suitable one.	4	4	3	4.5	15.5
AutoML	Utilize AutoML tools to automate the model selection, hyperparameter tuning.	3	3.5	2	5	13.5
WebApp	· Deploy the trained model into a user-friendly application or web platform for accessibility. · Deploy model using Flask web app	5	4	2.5	4	15.5

Hybrid Models	Hybrid models that combine traditional ML algorithms with deep learning to utilize the benefits of both approaches.	2.5	3	3	5	13.5
Model Performance and Optimization	Implement mechanisms to handle errors and provide meaningful feedback to users.	3.5	1	4	3	11.5
Model Performance and Optimization	Show confidence scores to indicate the model's accuracy.	2	4.5	1	3.5	11
Model Performance and Optimization	Optimize the model for real-time processing.	4	3	3.5	3	13.5
Model Performance and Optimization	Model should be able to handle big volumes of data and users.	5	3.5	4	4.5	17
Model Performance and Optimization	Consider the potential expansion to cover additional eye diseases in the future.	4.5	3	4	4	15.5

Model Performance and Optimization	Incorporate feedback from medical experts for model refinement.	3	2	3.5	4	12.5

SCORES BY VAIBHAV:

CLUSTER	IDEA	SCORES BY VAIBHAV				
		Impact	**Feasible**	**Cost**	**Alignment with goal**	**Total**
Transfer Learning	· Implement transfer learning using pre-trained models · Transfer learning using Inception V3 · Transfer learning using VGG19, Xception V3.	5	3.5	2	5	15.5
CNN	· Implement Eye disease detection using CNN · Experiment with different CNN architectures to find the most suitable one.	4.5	4	3	4.5	16

AutoML	Utilize AutoML tools to automate the model selection, hyperparameter tuning.	3.5	3.5	2	4	13
WebApp	· Deploy the trained model into a user-friendly application or web platform for accessibility. · Deploy model using Flask web app	4.5	3.5	2.5	5	15.5
Hybrid Models	Hybrid models that combine traditional ML algorithms with deep learning to utilize the benefits of both approaches.	3	2.5	3	5	13.5
Model Performance and Optimization	Implement mechanisms to handle errors and provide meaningful feedback to users.	3	2	4	2.5	11.5
Model Performance and Optimization	Show confidence scores to indicate the model's accuracy.	3	5	1	3.5	12.5

Model Performance and Optimization	Optimize the model for real-time processing.	2	2	4	1	9
Model Performance and Optimization	Model should be able to handle big volumes of data and users.	3.5	3	3	5	14.5
Model Performance and Optimization	Consider the potential expansion to cover additional eye diseases in the future.	2	3	3	3	11
Model Performance and Optimization	Incorporate feedback from medical experts for model refinement.	4	2.5	2	5	13.5

TOTAL SCORES:

Cumulative scores for each idea by summing up scores from individual team members:

CLUSTER	**IDEA**	**SCORES**			
		Harsh	**Advait**	**Vaibhav**	**Total**

Transfer Learning	· Implement transfer learning using pre-trained models · Transfer learning using Inception V3 · Transfer learning using VGG19, Xception V3.	16.5	18	15.5	50
CNN	· Implement Eye disease detection using CNN · Experiment with different CNN architectures to find the most suitable one.	16.5	15.5	16	48
AutoML	Utilize AutoML tools to automate the model selection, hyperparameter tuning.	14	13.5	13	40.5
WebApp	· Deploy the trained model into a user-friendly application or web platform for accessibility. · Deploy model using Flask web app	15	15.5	15.5	46

Hybrid Models	Hybrid models that combine traditional ML algorithms with deep learning to utilize the benefits of both approaches.	13.5	13.5	13.5	40.5
Model Performance and Optimization	Implement mechanisms to handle errors and provide meaningful feedback to users.	12	11.5	11.5	35
Model Performance and Optimization	Show confidence scores to indicate the model's accuracy.	12.5	11	12.5	36
Model Performance and Optimization	Optimize the model for real-time processing.	10.5	13.5	9	33
Model Performance and Optimization	Model should be able to handle big volumes of data and users.	14.5	17	14.5	46
Model Performance and Optimization	Consider the potential expansion to cover additional eye diseases in the future.	13	15.5	11	39.5
Model Performance and Optimization	Incorporate feedback from medical experts for model refinement.	14	12.5	13.5	40

RANKING ORDER:

Ranking the ideas based on their scores.

CLUSTER	IDEA	Total
Transfer Learning	· Implement transfer learning using pre-trained models · Transfer learning using Inception V3 · Transfer learning using VGG19, Xception V3.	50
CNN	· Implement Eye disease detection using CNN · Experiment with different CNN architectures to find the most suitable one.	48
WebApp	· Deploy the trained model into a user-friendly application or web platform for accessibility. · Deploy model using Flask web app	46
Model Performance and Optimization	Model should be able to handle big volumes of data and users.	46
AutoML	Utilize AutoML tools to automate the model selection, hyperparameter tuning.	40.5

Hybrid Models	Hybrid models that combine traditional ML algorithms with deep learning to utilize the benefits of both approaches.	40.5
Model Performance and Optimization	Incorporate feedback from medical experts for model refinement.	40
Model Performance and Optimization	Consider the potential expansion to cover additional eye diseases in the future.	39.5
Model Performance and Optimization	Show confidence scores to indicate the model's accuracy.	36
Model Performance and Optimization	Implement mechanisms to handle errors and provide meaningful feedback to users.	35
Model Performance and Optimization	Optimize the model for real-time processing.	33

Step-3: Idea Prioritization

The top 5 ideas in ranking from the previous table and the last idea are mapped on a prioritization graph.

4

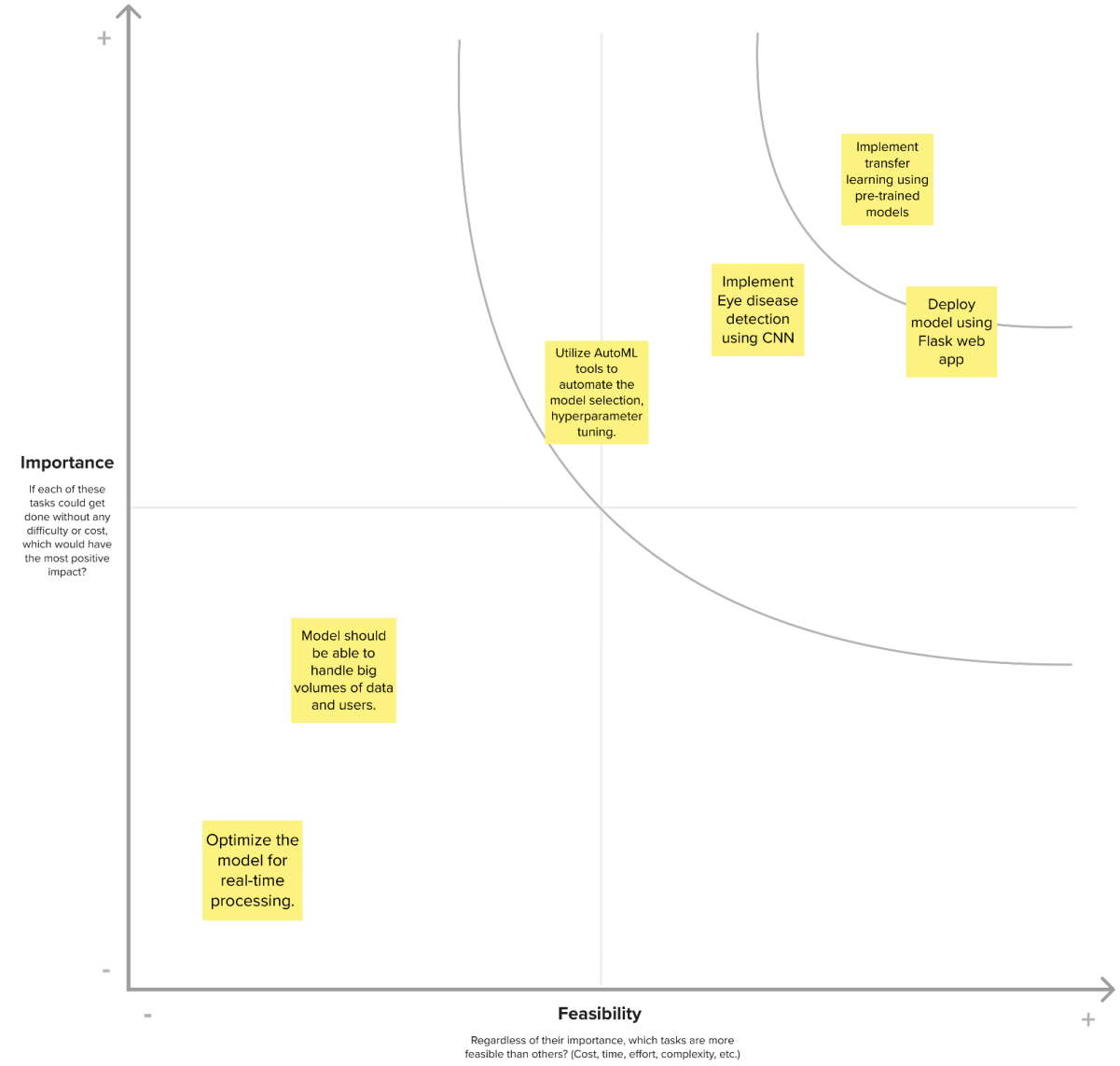
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.



4. Requirement Analysis

4.1 Functional Requirements

Specification of functionalities such as image preprocessing, model training, classification, and result visualization. Requirement for user-friendly interfaces for both medical practitioners and patients.

List of functional requirements for this project:

Table-1 : Components & Technologies:

S. No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI	HTML, CSS, JavaScript.
2.	Application Logic-1	Logic for a process in the application	Python Flask web app
3.	Database	Collect the Dataset Based on the Problem Statement	File Manager, etc.
4.	File Storage/ Data	File storage requirements for Storing the dataset	Local System, Google Drive Etc
5.	Frame Work	Used to Create a web Application, Integrating Frontend and Back End	Python Flask
6.	Deep Learning Model	Purpose of Model	CNN and Transfer Learning
7.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration :	Localhost port: 5000

4.2 Non-Functional Requirements

Identification of non-functional aspects like system scalability, accuracy threshold, computational efficiency, and ethical considerations regarding patient data privacy.

List of non-functional requirements for this project:

Table-2: Application Characteristics:

S. No	Characteristics	Description	Technology

1.	Open-Source Frameworks	List the open-source frameworks used	Python's Flask library
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	-
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Cloud Services
4.	Availability	Justify the availability of applications (e.g. use of load balancers, distributed servers etc.)	Web hosting (Github Pages)
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	-

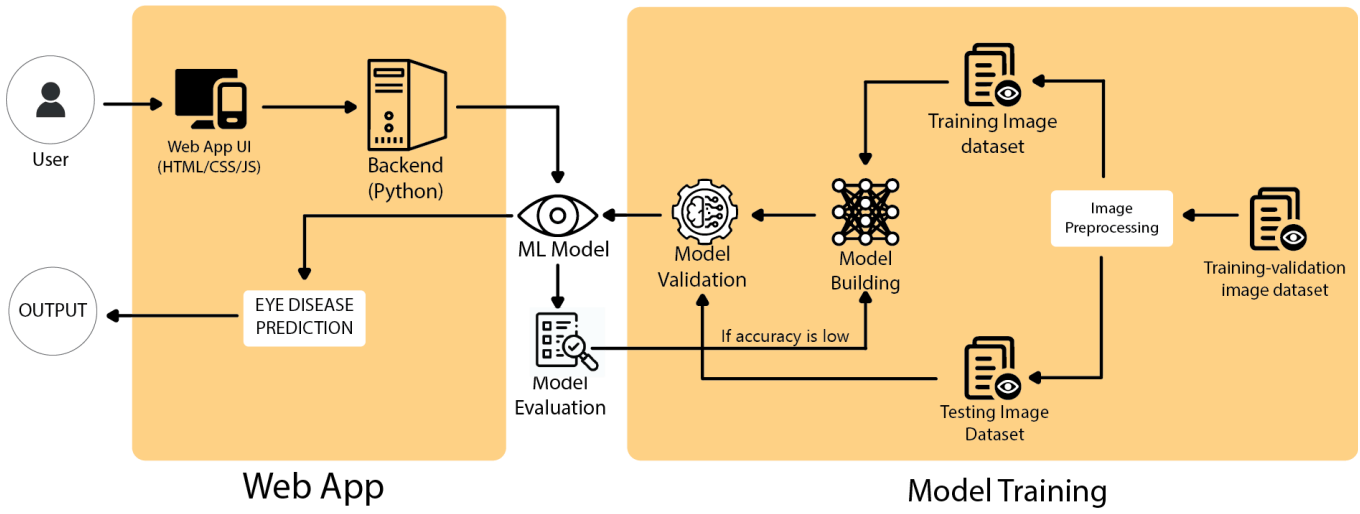
5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

DFD Level 0 (Industry Standard)



FLOW

1) User enters the test image of the retina.
2) The image info is converted into array format in the backend.

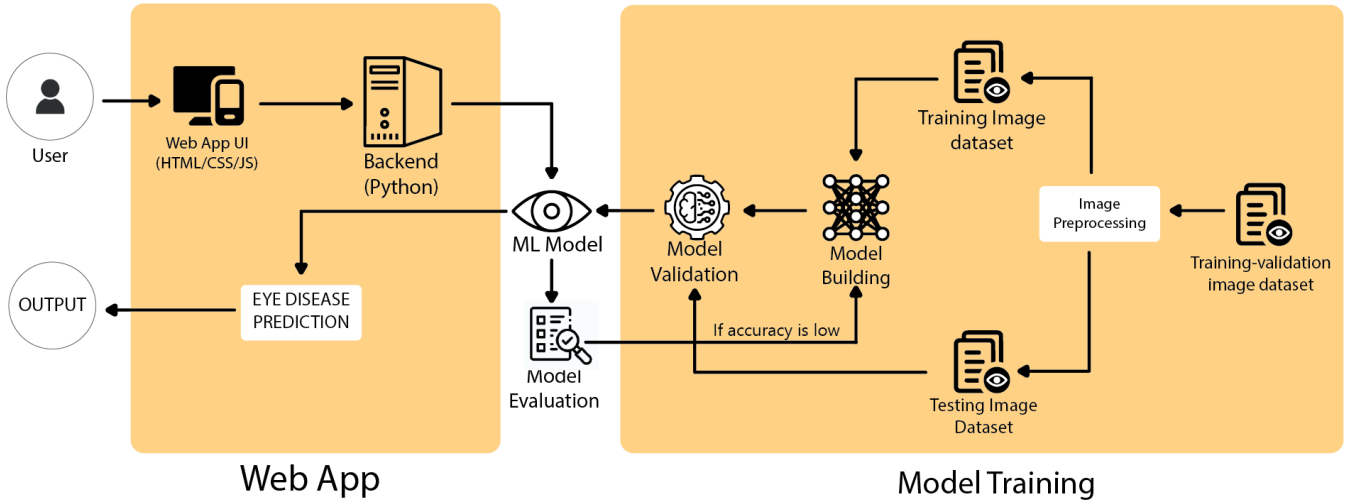
3) The array data is passed through the ml model.
4) The ML model returns the probability of detecting the following from the input image : glaucoma', 'cataract', 'normal', 'diabetic_retinopathy'.
5) The category with highest probability is printed as predicted output.

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Desktop)	Web hosting	USN-1	As a user, I am able to access the web app from the desktop. It is working.	I can access the portal.	High	Sprint 1
Customer (Desktop/Mobile)	POST/GET request	USN-2	As a user, I am able to select an image of my choice	The image of my choice is being displayed on web portal	High	Sprint 2
Customer (Desktop/Mobile)	Python Script	USN-3	As a user, I am able to get a prediction result after uploading the retina image	The system is functional end to end.	Moderate	Sprint 3
Customer (Mobile)	Web hosting	USN-4	As a user, I am able to access the web app from my smartphone.	I can access the portal.	High	Sprint 1
Customer (Desktop/Mobile)	Trained Machine Learning model	USN-5	As a user, I am able to get varied predicted values as per my input.	The prediction is able to detect the disease with high accuracy.	High	Sprint 4
Admin	ML Model	USN-6	As an admin, I am able to edit changes in the ML model.	Trainable model in .ipynb file	Low	Sprint 5
Admin	UI/UX (HTML/CSS/JS)	USN-7	As an admin, I am able to modify UI/UX as per user convenience.	Editable HTML, CSS, JS	High	Sprint 6

5.2 Solution Architecture

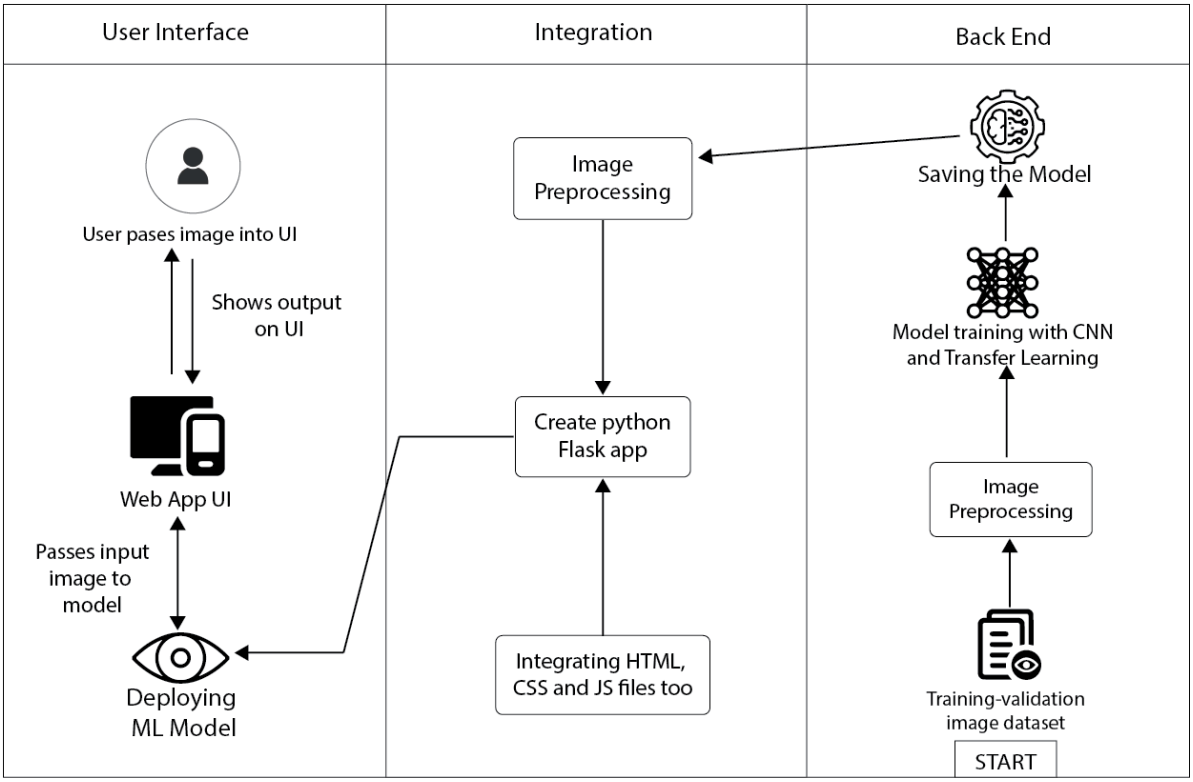


S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	A myriad of eye diseases, stemming from factors like genetics, aging, environment, or underlying health issues, can compromise vision.The need is to develop a reliable and adaptable system to help patients by allowing healthcare professionals to diagnose and manage eye diseases, emphasizing accuracy, efficiency and early detection.
2.	Idea / Solution description	Using Transfer Learning based approach to design a reliable and adaptable model that would accept an image of a patient's retina scan and then detect if the patient belongs to one of the following categories : glaucoma', 'cataract', 'normal', 'diabetic_retinopathy'.
3.	Novelty / Uniqueness	Use of ResNet50 convolutional base for training the ML model using the image dataset.
4.	Social Impact / Customer Satisfaction	The system enables quicker and more accurate detection of eye diseases, contributing to early intervention and improved treatment outcomes.It contributes to public health by offering accessible and efficient diagnostic tools.

		Patients benefit from timely and accurate diagnoses, reducing anxiety and enabling prompt medical intervention.
5.	Business Model (Revenue Model)	Subscription Model: Charging healthcare institutions or individual users for access to the detection system on a subscription basis. Licensing Model: Licensing the technology to medical facilities, clinics, or AI developers for integration into their systems. Partnerships: Collaborating with pharmaceutical companies, insurance providers, or research institutions for mutually beneficial ventures.
6.	Scalability of the Solution	In future, as the number of users increases, utilizing cloud services for scalability would be the best solution. Shifting all the workload from local servers to cloud servers, enabling the system to handle increasing data volume and user demands efficiently.

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Web hosting	USN-1	As a user, I am able to access the web app from the desktop. It is working.	2	High	Advait
Sprint 2	POST/GET request	USN-2	As a user, I am able to select an image of my choice	2	High	Advait
Sprint 3	Python Script	USN-3	As a user, I am able to get a prediction result after uploading the retina image	2	Moderate	Vaibhav
Sprint 1	Web hosting	USN-4	As a user, I am able to access the web app from my smartphone.	2	High	Vaibhav
Sprint 4	Trained Machine Learning	USN-5	As a user, I am able to get varied predicted values as per my input.	2	High	Advait

	model					
Sprint 5	ML Model	USN-6	As an admin, I am able to edit changes in the ML model.	2	Low	Harsh
Sprint 6	UI/UX (HTML/CSS/JS)	USN-7	As an admin, I am able to modify UI/UX as per user convenience.	2	High	Harsh

Project Tracker, Velocity & Burndown Chart: (4 Marks)

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	2	4 Days	14 Oct 2023	18 Nov 2023	4	14 Oct 2022
Sprint-2	2	8 Days	18 Nov 2023	25 Nov 2023	2	14 Oct 2022
Sprint-3	2	7 Days	26 Nov 2023	02 Nov 2023	2	14 Oct 2022
Sprint-4	2	8 Days	02 Nov 2023	09 Nov 2023	2	14 Oct 2022
Sprint-5	2	8 Days	09 Nov 2023	16 Nov 2023	2	14 Oct 2022
Sprint-6	2	5 Days	16 Nov 2023	20 Nov 2023	2	14 Oct 2022

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

1: Importing necessary libraries

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

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

2: Data Collection

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

Dataset-

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

3: Creating the dataset:

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

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

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

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

Plotting the images with labels for image data visualization.

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

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

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

    plt.show()
```

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

5: Importing necessary libraries for building the resnet model.

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

6: Image Preprocessing

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

a) Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- The image flips via the horizontal_flip and vertical_flip arguments.

- Image rotations via the rotation_range argument
- Image brightness via the brightness_range argument.
- Image zoom via the zoom_range argument.

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

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

b) Apply ImageDataGenerator functionality to Trainset and Testset

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code.For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories 'glaucoma', 'cataract', 'normal' and 'diabetic retinopathy'.
Arguments:

- directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data which is 64.
- target_size: Size to resize images after they are read from disk.
- class_mode:
  - 'int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).
  - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
  - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
  - None (no labels).

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

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

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

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

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

7) Model Building

Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

a) Initializing the model and adding CNN layers

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

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

- We are adding a convolution layer with activation function as “relu” and with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.
- Max pool layer is used to downsample the input.(Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter) ● Flatten layer flattens the input. Does not affect the batch size.

b) Adding Dense Layers

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

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

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

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

c) Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.

- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

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

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

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

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

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

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

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

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

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 128)               6422656

 dense_1 (Dense)             (None, 4)                 516

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

d) Train The model

Now, let us train our model with our image dataset. The model is trained for 30 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

fit_generator functions used to train a deep learning neural network
Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

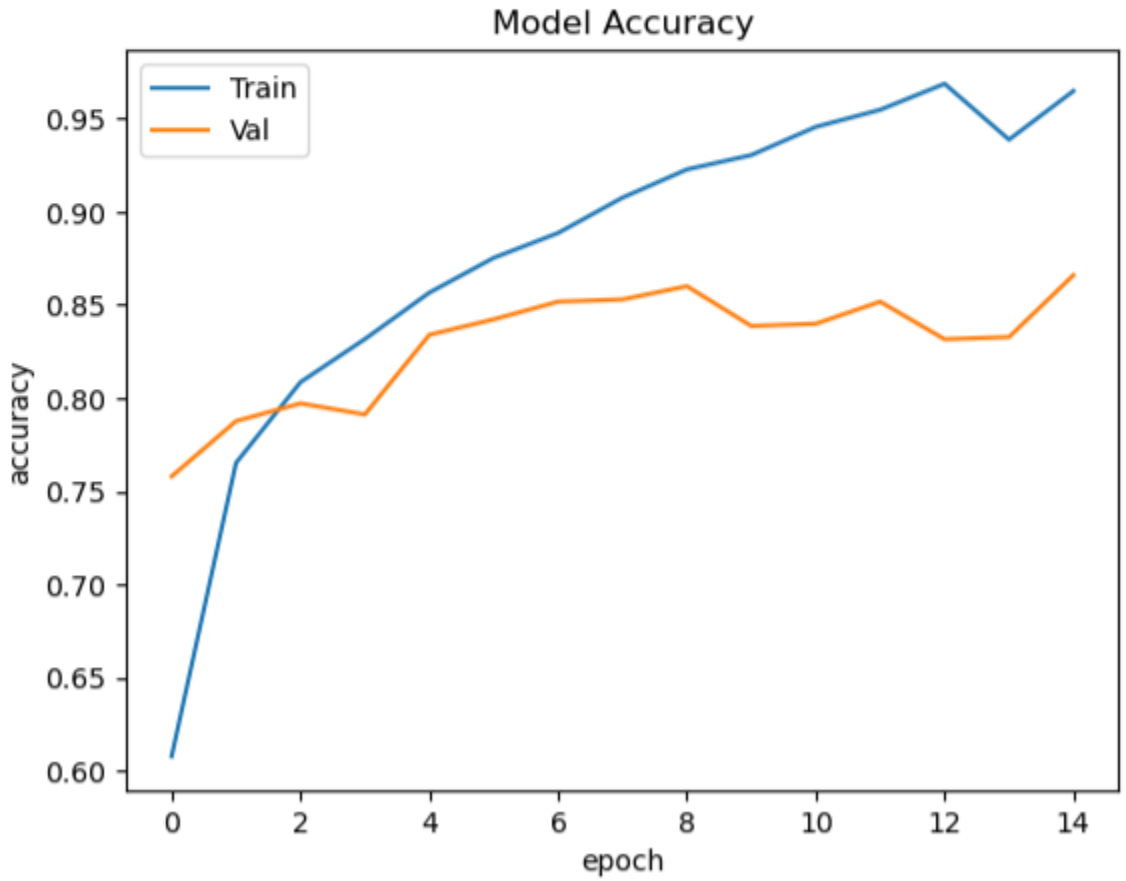
```
# Fitting the model on training and validation data
his = model.fit(train_data, validation_data=valid_data, epochs=15)
```

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

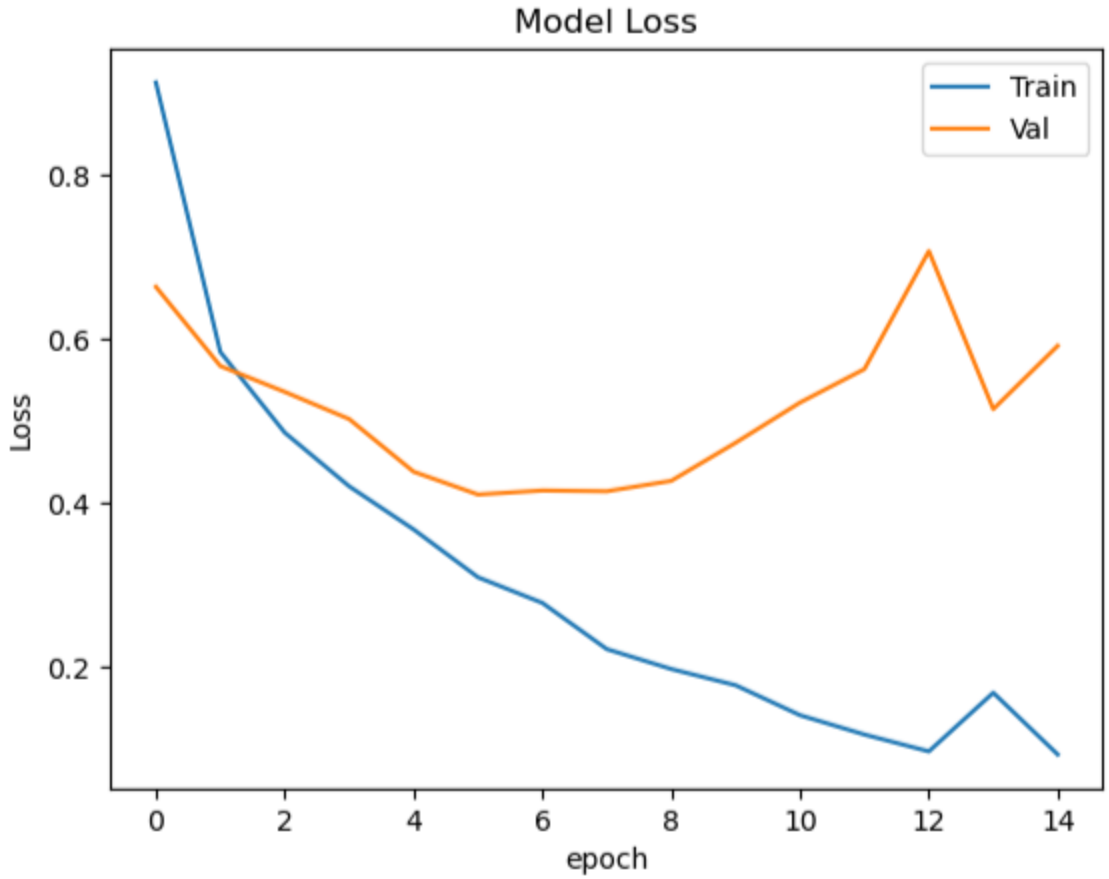
e) Evaluating The model accuracy

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.
Load the saved model using load_model

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



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



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

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

f) Save the Model

The model is saved with .h5 extension as follows:

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

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

8) Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

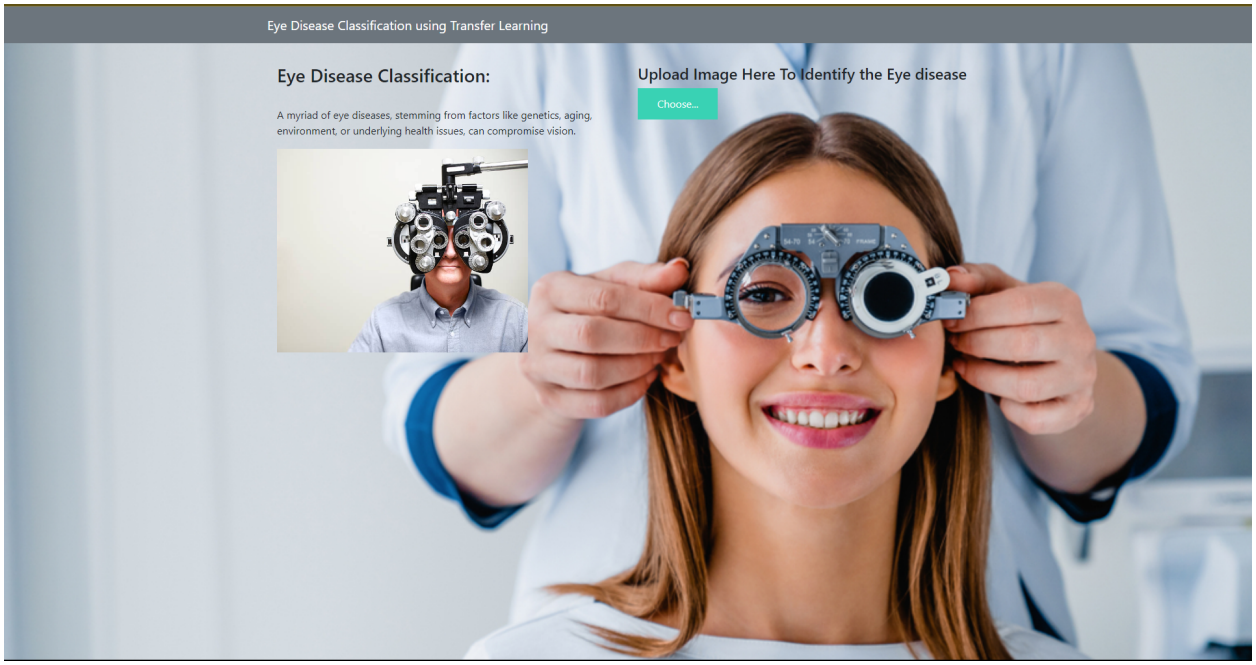
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the "Image" button, the next page is opened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

a) Create HTML - CSS -JS Pages

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

○ **Link :**CSS , JS

index.html looks like this:



b) Build python code

Task 1: Importing Libraries

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

```
1
2   import numpy as np
3   import os
4   from tensorflow.keras.models import load_model
5   from tensorflow.keras.preprocessing import image
6   from flask import Flask , request, render_template
7   #from werkzeug.utils import secure_filename
8   #from gevent.pywsgi import WSGIServer
```

Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

Task 2: Creating our flask application and loading our model by using load_model method

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

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

Task 3: Routing to the html Page

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

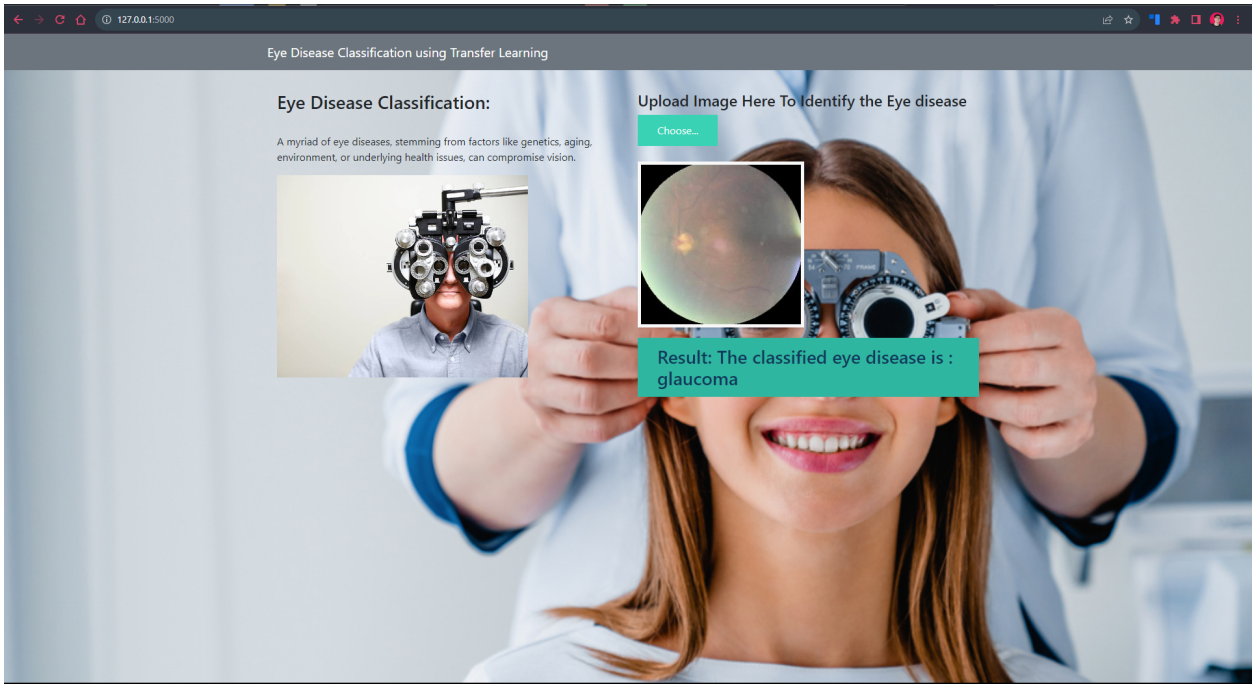
```
@app.route('/')
def index():
    return render_template("/index.html")
    #return "Hello Harshman!"

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

        img = image.load_img(filepath,target_size = (224,224))
        x = image.img_to_array(img)
        print(x)
        x = np.expand_dims(x,axis =0)
        print(x)
        y=model.predict(x)
        preds=np.argmax(y, axis=1)
        #preds = model.predict_classes(x)
        print("prediction",preds)
        index = ['glaucoma', 'cataract', 'normal', 'diabetic_retinopathy']
        text = "The classified eye disease is : " + str(index[preds[0]])
    return text
if __name__ == '__main__':
    app.run(debug = False, threaded = False)
```

In the above example, '/' URL is bound with index.html function. Hence, when the home page of a web server is opened in the browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accessed through POST or GET Method.

Showcasing prediction on UI:



Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using the OS library. Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in the html page.

Predicting the results

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

Finally, Run the application

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

c) Run the application

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

web page. Then it will run on localhost:5000

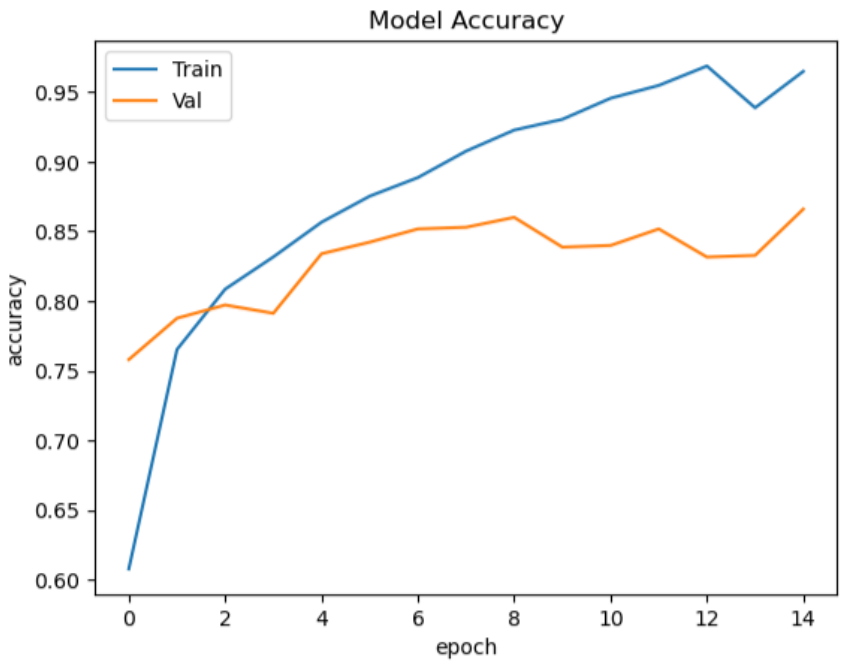
Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.

8. PERFORMANCE TESTING

8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot

1.	Model Summary	-	`# Compiling the model and displaying summary`
			`model.compile(optimizer='adam', loss=tf.keras.losses.categorical_cros`
			`model.summary()`
			
			`Model: "sequential"`
			`_____`
			`Layer (type) Output Shape Param #`
			`=====`
			`rescaling (Rescaling) (None, 224, 224, 3) 0`
			`conv2d (Conv2D) (None, 224, 224, 16) 448`
			`max_pooling2d (MaxPooling2 (None, 112, 112, 16) 0`
			`D)`
			`conv2d_1 (Conv2D) (None, 112, 112, 32) 4640`
			`max_pooling2d_1 (MaxPoolin (None, 56, 56, 32) 0`
			`g2D)`
			`conv2d_2 (Conv2D) (None, 56, 56, 64) 18496`
			`max_pooling2d_2 (MaxPoolin (None, 28, 28, 64) 0`
			`g2D)`
			`flatten (Flatten) (None, 50176) 0`
			`dense (Dense) (None, 128) 6422656`
			`dense_1 (Dense) (None, 4) 516`
			`=====`
			`Total params: 6446756 (24.59 MB)`
			`Trainable params: 6446756 (24.59 MB)`
			`Non-trainable params: 0 (0.00 Byte)`

2.	Accuracy	Training Accuracy - 96.47%
		
		Validation Accuracy - 86.60%
			`# fitting the model`
			`his = model.fit(train_data, validation_data=valid_data, epochs=15)`
			
			Epoch 1/15
			106/106 [==========] - 110s 1s/step - loss: 0.9138 - accuracy: 0.6079 - val_loss: 0.6640 - val_accuracy: 0.7580
			Epoch 2/15
			106/106 [==========] - 77s 726ms/step - loss: 0.5845 - accuracy: 0.7653 - val_loss: 0.5673 - val_accuracy: 0.7877
			Epoch 3/15
			106/106 [==========] - 76s 717ms/step - loss: 0.4858 - accuracy: 0.8085 - val_loss: 0.5356 - val_accuracy: 0.7972
			Epoch 4/15
			106/106 [==========] - 74s 697ms/step - loss: 0.4202 - accuracy: 0.8317 - val_loss: 0.5024 - val_accuracy: 0.7912
			Epoch 5/15
			106/106 [==========] - 75s 704ms/step - loss: 0.3676 - accuracy: 0.8566 - val_loss: 0.4381 - val_accuracy: 0.8339
			Epoch 6/15
			106/106 [==========] - 73s 686ms/step - loss: 0.3090 - accuracy: 0.8752 - val_loss: 0.4101 - val_accuracy: 0.8422
			Epoch 7/15
			106/106 [==========] - 72s 676ms/step - loss: 0.2777 - accuracy: 0.8886 - val_loss: 0.4151 - val_accuracy: 0.8517
			Epoch 8/15
			106/106 [==========] - 72s 673ms/step - loss: 0.2212 - accuracy: 0.9075 - val_loss: 0.4141 - val_accuracy: 0.8529
			Epoch 9/15
			106/106 [==========] - 71s 672ms/step - loss: 0.1969 - accuracy: 0.9226 - val_loss: 0.4268 - val_accuracy: 0.8600
			Epoch 10/15
			106/106 [==========] - 72s 674ms/step - loss: 0.1770 - accuracy: 0.9303 - val_loss: 0.4734 - val_accuracy: 0.8387
			Epoch 11/15
			106/106 [==========] - 72s 677ms/step - loss: 0.1404 - accuracy: 0.9455 - val_loss: 0.5226 - val_accuracy: 0.8399
			Epoch 12/15
			106/106 [==========] - 72s 677ms/step - loss: 0.1168 - accuracy: 0.9547 - val_loss: 0.5635 - val_accuracy: 0.8517
			Epoch 13/15
			106/106 [==========] - 74s 692ms/step - loss: 0.0962 - accuracy: 0.9686 - val_loss: 0.7077 - val_accuracy: 0.8316
			Epoch 14/15
			106/106 [==========] - 72s 677ms/step - loss: 0.1678 - accuracy: 0.9386 - val_loss: 0.5145 - val_accuracy: 0.8327
			Epoch 15/15
			106/106 [==========] - 73s 687ms/step - loss: 0.0924 - accuracy: 0.9647 - val_loss: 0.5919 - val_accuracy: 0.8660
			
			 

3.	Confidence Score (Only Yolo Projects)	NOT APPLICABLE	

S.No.	Parameter	Values	Screenshot

1.	Metrics	**Classification Model:** Accuray Score- & Classification Report -	`print(classification_report(y_test,y_pred,target_names = labels))`
			precision recall f1-score support
			cataract 0.91 0.83 0.87 221
			diabetic_retinopathy 0.97 1.00 0.98 215
			glaucoma 0.85 0.72 0.78 194
			normal 0.75 0.90 0.82 213
			accuracy 0.87 843
			macro avg 0.87 0.86 0.86 843
			weighted avg 0.87 0.87 0.87 843
			`# fitting the model`
			`his = model.fit(train_data, validation_data=valid_data, epochs=15)`
			Epoch 1/15
			106/106 [==========] - 110s 1s/step - loss: 0.9138 - accuracy: 0.6079 - val_loss: 0.6640 - val_accuracy: 0.7580
			Epoch 2/15
			106/106 [==========] - 77s 726ms/step - loss: 0.5845 - accuracy: 0.7653 - val_loss: 0.5673 - val_accuracy: 0.7877
			Epoch 3/15
			106/106 [==========] - 76s 717ms/step - loss: 0.4858 - accuracy: 0.8085 - val_loss: 0.5356 - val_accuracy: 0.7972
			Epoch 4/15
			106/106 [==========] - 74s 697ms/step - loss: 0.4202 - accuracy: 0.8317 - val_loss: 0.5024 - val_accuracy: 0.7912
			Epoch 5/15
			106/106 [==========] - 75s 704ms/step - loss: 0.3676 - accuracy: 0.8566 - val_loss: 0.4381 - val_accuracy: 0.8339
			Epoch 6/15
			106/106 [==========] - 73s 686ms/step - loss: 0.3090 - accuracy: 0.8752 - val_loss: 0.4101 - val_accuracy: 0.8422
			Epoch 7/15
			106/106 [==========] - 72s 676ms/step - loss: 0.2777 - accuracy: 0.8886 - val_loss: 0.4151 - val_accuracy: 0.8517
			Epoch 8/15
			106/106 [==========] - 72s 673ms/step - loss: 0.2212 - accuracy: 0.9075 - val_loss: 0.4141 - val_accuracy: 0.8529
			Epoch 9/15
			106/106 [==========] - 71s 672ms/step - loss: 0.1969 - accuracy: 0.9226 - val_loss: 0.4268 - val_accuracy: 0.8600
			Epoch 10/15
			106/106 [==========] - 72s 674ms/step - loss: 0.1770 - accuracy: 0.9303 - val_loss: 0.4734 - val_accuracy: 0.8387
			Epoch 11/15
			106/106 [==========] - 72s 677ms/step - loss: 0.1404 - accuracy: 0.9455 - val_loss: 0.5226 - val_accuracy: 0.8399
			Epoch 12/15
			106/106 [==========] - 72s 677ms/step - loss: 0.1168 - accuracy: 0.9547 - val_loss: 0.5635 - val_accuracy: 0.8517
			Epoch 13/15
			106/106 [==========] - 74s 692ms/step - loss: 0.0962 - accuracy: 0.9686 - val_loss: 0.7077 - val_accuracy: 0.8316
			Epoch 14/15
			106/106 [==========] - 72s 677ms/step - loss: 0.1678 - accuracy: 0.9386 - val_loss: 0.5145 - val_accuracy: 0.8327
			Epoch 15/15
			106/106 [==========] - 73s 687ms/step - loss: 0.0924 - accuracy: 0.9647 - val_loss: 0.5919 - val_accuracy: 0.8660

9. RESULTS

9.1 Output Screenshots

Eye Disease Classification using Transfer Learning

Eye Disease Classification:

A myriad of eye diseases, stemming from factors like genetics, aging, environment, or underlying health issues, can compromise vision.

Upload Image Here To Identify the Eye disease

Choose...

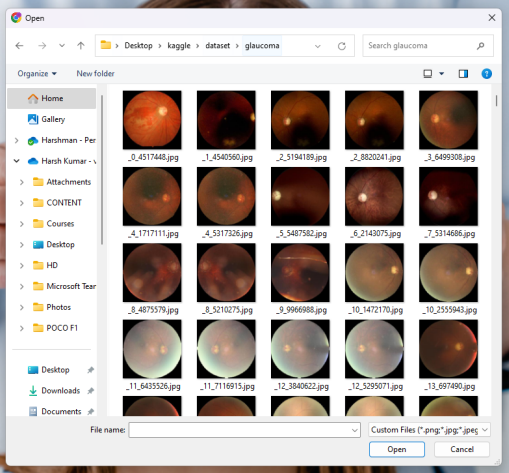
Eye Disease Classification using Transfer Learning

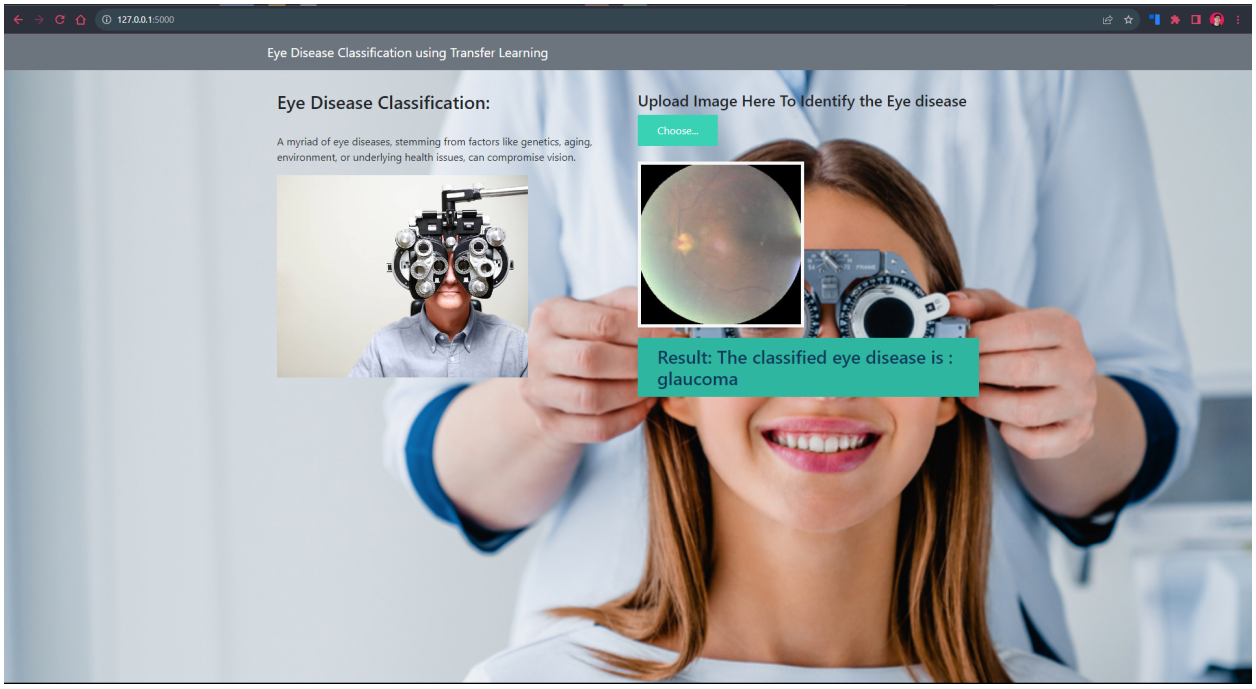
Eye Disease Classification:

A myriad of eye diseases, stemming from factors like genetics, aging, environment, or underlying health issues, can compromise vision.

Upload Image Here To Identify the Eye disease

Choose...





10. Advantages & Disadvantages

Advantages:

a) improved diagnostic accuracy
b) faster detection
c) Trainable model can get better and better accuracy over time by continuously updating it with new datasets.
d) Scalability can be increased to allow more and more users access the website and take the benefit of this service.

Disadvantages:

a) dependency on data quality and interpretability of results.
b) 100% accuracy unachievable. Chances of minute errors always exist.

11. Conclusion

Therefore, the need for an efficient and reliable system that assists medical practitioners in diagnosing eye diseases at an early stage has been satisfied with the help of diligent application of Machine Learning concepts such as . By automating the detection process, it aims to improve patient care, reduce human error, and facilitate timely interventions