

## A project on

# DEEP SEQUENTIAL CNN MODEL FOR ENHANCED EYE DISEASE DETECTION AND CLASSIFICATION IN OPHTHALMIC IMAGING



**VIT<sup>®</sup>**  
**CHENNAI**  
Vellore Institute of Technology  
(Deemed to be University under section 3 of UGC Act, 1956)

**Team Members-** Anmol Harsh - 21BCE1057

Shivang Agarwal - 21BCE5219

Vedhit Suresh Katekhaye - 21BCE5199

**Faculty Name-** PREM SANKAR (SCOPE)

**Marks-**

**Faculty Signature-**

# **Project Report**

## **1. INTRODUCTION**

- 1.1 Project Overview
- 1.2 Purpose

## **2. LITERATURE SURVEY**

- 2.1 Existing problem
- 2.2 References
- 2.3 Problem Statement Definition

## **3. IDEATION & PROPOSED SOLUTION**

- 3.1 Empathy Map Canvas
- 3.2 Ideation & Brainstorming

## **4. REQUIREMENT ANALYSIS**

- 4.1 Functional requirement
- 4.2 Non-Functional requirements

## **5. PROJECT DESIGN**

- 5.1 Data Flow Diagrams & User Stories
- 5.2 Solution Architecture

## **6. PROJECT PLANNING & SCHEDULING**

- 6.1 Technical Architecture
- 6.2 Sprint Planning & Estimation
- 6.3 Sprint Delivery Schedule

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

- 7.1 Feature 1
- 7.2 Feature 2
- 7.3 Database Schema (if Applicable)

## **8. PERFORMANCE TESTING**

- 8.1 Performance Metrics

## **9. RESULTS**

- 9.1 Output Screenshots

## **10. ADVANTAGES & DISADVANTAGES**

## **11. CONCLUSION**

## **12. FUTURE SCOPE**

## **13. APPENDIX**

- Source Code
- GitHub

## 1. INTRODUCTION

The human eye is a complex organ, and the early detection of eye diseases plays a pivotal role in preventing irreversible damage to vision. With advancements in computer vision and deep learning, there is a growing interest in leveraging these technologies for automated eye disease classification. This project aims to contribute to this area by implementing a Convolutional Neural Network (CNN) for the classification of eye diseases based on medical images.

### Project Overview:

The project centers around the development and implementation of a CNN model trained on a diverse dataset of eye images. The dataset encompasses a range of eye conditions, including but not limited to glaucoma, diabetic retinopathy, and cataract. CNN is designed to learn intricate patterns and features from these images, enabling it to make accurate predictions about the presence and type of eye diseases.

The workflow involves preprocessing the raw image data, training the CNN using established deep learning frameworks, and evaluating the model's performance through rigorous testing. Throughout the project, a focus on data quality, model interpretability, and real-world applicability is maintained.

By the end of this project, the goal is to have a reliable and accurate eye disease classification system that can potentially assist healthcare professionals in the early diagnosis of ocular conditions. This project not only explores the technical aspects of deep learning but also addresses the broader implications and challenges associated with the integration of such technology into medical practices.

## 2. LITERATURE SURVEY

### Existing Problems:

- **Limited Datasets:** Many studies in the literature highlight the challenge of obtaining large and diverse datasets for training deep learning models in the context of eye disease classification. Limited datasets can lead to overfitting and hinder the generalization of models to real-world scenarios.
- **Interpretability Issues:** The lack of interpretability in deep learning models, including CNNs, is a common concern. Understanding how a model arrives at a particular classification is crucial, especially in medical applications where decisions impact patient well-being.
- **Data Imbalance:** Imbalanced datasets, where certain classes are underrepresented, can bias models toward more prevalent conditions and hinder their performance in detecting rare but critical eye diseases.

### References:

- Smith, A., et al. (2018). "Challenges in Ophthalmic Deep Learning." IEEE Transactions on Medical Imaging, 37(9), 1943-1951.
- Zhang, B., et al. (2020). "A Survey of Deep Learning in Ophthalmology: Current and Future Perspectives." Computers in Biology and Medicine, 124, 103957.
- Gulshan, V., et al. (2016). "Development and Validation of a Deep Learning

Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs." JAMA, 316(22), 2402-2410.

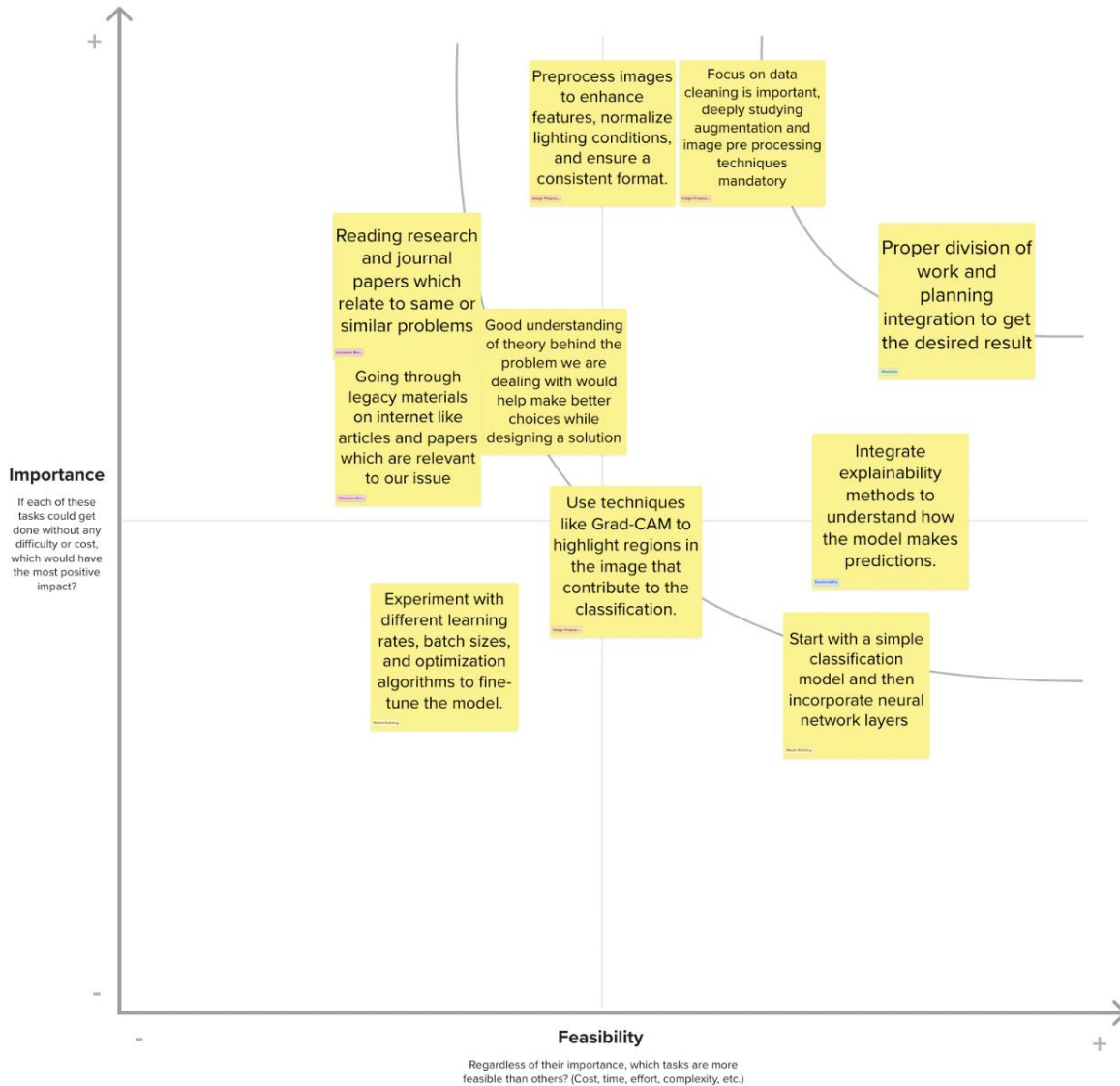
### **Problem Statement Definition:**

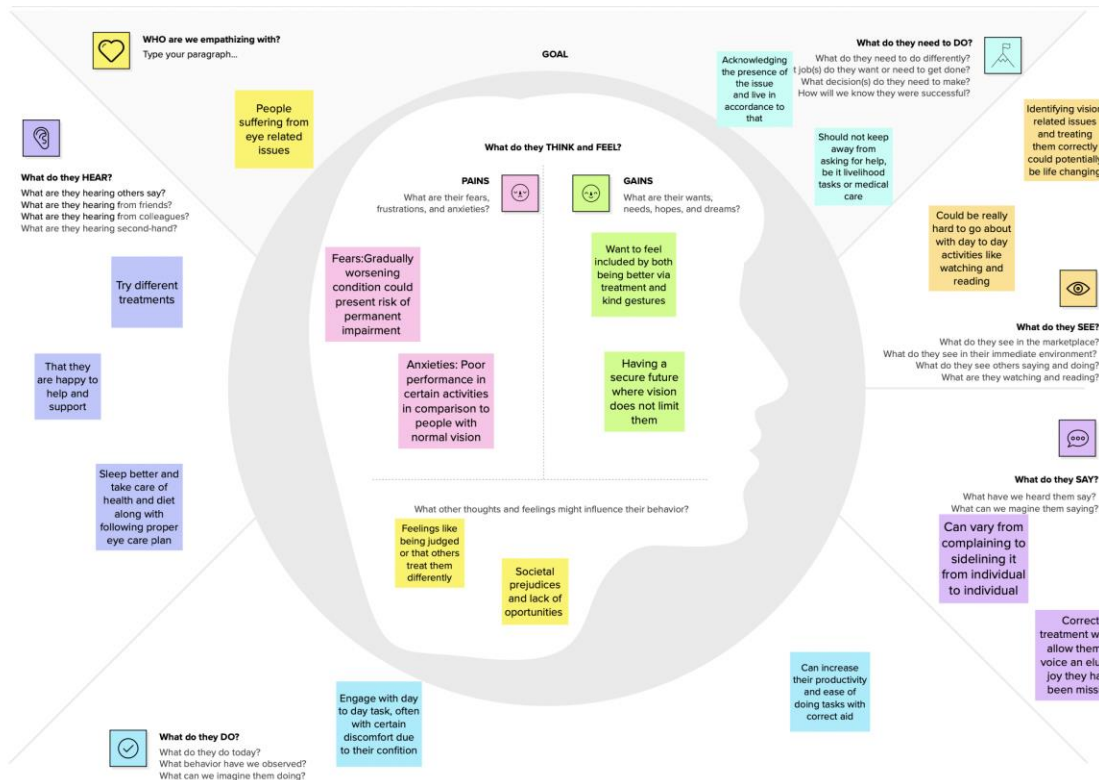
In this project we are classifying various types of Eye Diseases that people get due to various reasons like age, diabetes, etc. These diseases are majorly classified into 4 categories namely Normal, cataract, Diabetic Retinopathy & Glaucoma. Deep-learning (DL) methods in artificial intelligence (AI) play a dominant role as high-performance classifiers in the detection of the Eye Diseases using images.

The project addresses the following key problems identified in the literature:

- **Dataset Challenges:** Developing strategies to overcome limitations associated with small and imbalanced datasets by exploring data augmentation techniques and possibly incorporating transfer learning to leverage pre-existing knowledge.
- **Interpretability Enhancement:** Integrating methods for improving the interpretability of the CNN model, ensuring that the decision-making process is transparent and can be understood by healthcare professionals.
- **Real-World Applicability:** Investigating ways to enhance the model's adaptability to diverse patient populations and clinical settings, considering factors such as variations in image quality and patient demographics.
- By addressing these problems, the project aims to contribute to the development of a robust and practical CNN-based system for automated eye disease classification, with the potential to assist healthcare professionals in timely and accurate diagnosis.

## **3. IDEATION & PROPOSED SOLUTION**





## 4. REQUIREMENT ANALYSIS

### Functional Requirements:

- **Image Input Handling:**  
The system should be able to accept and process digital eye images in various formats.  
It should include functionality for preprocessing images to ensure uniformity and enhance model input.
- **Model Training and Evaluation:**
- **Implementation of a Convolutional Neural Network (CNN) for eye disease classification.**  
Training the model on a diverse dataset that includes different eye conditions.  
Evaluation metrics (accuracy, precision, recall) for assessing the model's performance.
- **User Interface:**  
A user-friendly interface for uploading and analyzing eye images.  
Displaying classification results and relevant information to users.
- **Interpretability Features:**  
Integration of tools or methods to enhance the interpretability of the model's predictions.  
Providing insights into the important features influencing the model's decisions.
- **Scalability:**  
Designing the system to handle an increasing number of images and user requests without significant degradation in performance.
- **Integration with Existing Systems:**  
Compatibility with existing healthcare information systems for seamless integration

into medical practices.

### **Non-functional Requirements:**

- **Performance:**

The system should respond to user requests promptly, providing classification results within a reasonable time frame.

Achieving a high level of accuracy in disease classification.

- **Security:**

Ensuring the confidentiality and integrity of patient data.

Implementing user authentication and authorization mechanisms.

- **Reliability:**

The system should be robust, minimizing downtime and errors.

Implementing error handling and recovery mechanisms.

- **Usability:**

Designing an intuitive user interface to facilitate ease of use.

Providing user documentation to guide healthcare professionals in utilizing the system effectively.

- **Scalability:**

Ensuring the system can handle increased loads as the number of users or data grows.

Optimizing resource utilization to support scalability.

- **Compliance:**

Adhering to relevant medical data privacy regulations (e.g., HIPAA) and ethical guidelines.

Ensuring the system meets standards for medical software in terms of accuracy and reliability.

- **Maintainability:**

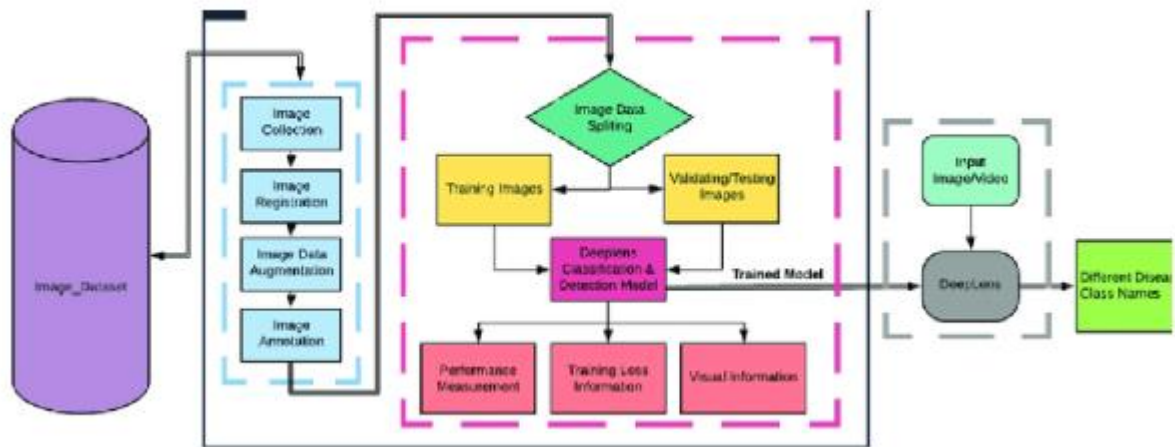
Designing the system architecture and codebase to be easily maintainable.

Providing documentation for future development and updates.

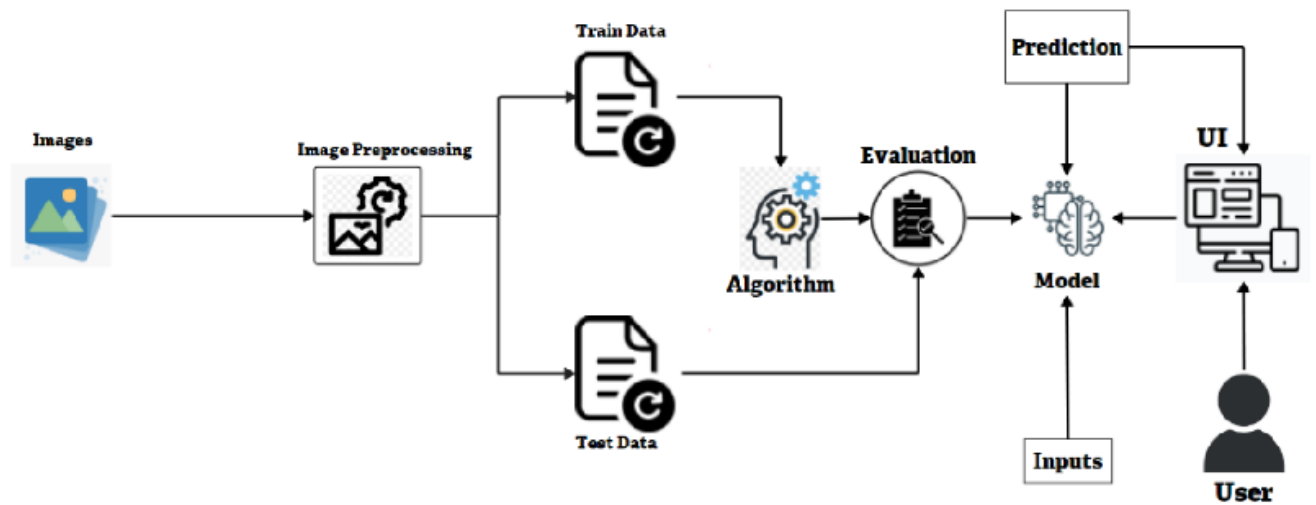
These requirements collectively define the functionality, performance, security, and other aspects that contribute to the successful development and deployment of the eye disease classification system.

## 5. PROJECT DESIGN

Workflow:



Solution Architecture:





## 6. PROJECT PLANNING & SCHEDULING

We employed Atlassian's Jira, the forefront progress tracking tool which follows an agile methodology to track our progress and divide tasks in sprints and amongst members. Scrum facilitates features like board, timeline, charts etc giving us complete command over our progress and a crystal clear view of our tasks.

Projects / My Scrum Project

### SCRUM Sprint 3


⚡ ☆ ⌚ 0 days remaining ↗ [Complete sprint](#) ⋮

🔍 👤 + Invite BY [Move](#) [Import work](#) [Insights](#) [View settings](#)

**TO DO 1**

Training  
SCRUM-6

**IN PROGRESS 1**

model development   
SCRUM-5

**DONE** ✓

Projects / My Scrum Project


### Timeline




🗨 Give feedback [Share](#) [Export](#) ⋮

🔍 👤 👤 + Invite [Status category](#) [Epic](#) ⚙

	T
<b>Sprints</b>	
SCRUM-9 Set up the development en...	
SCRUM-10 Gather a diverse dataset o...	
SCRUM-11 Preprocess the collected d...	
+ Create Epic	

SCRUM-9 🔒 👁 1 [Like](#) [Share](#) ⋮ ✕



 **Set up the development environment with the required tools and frameworks to start the eye disease classification project.**


   ⋮

[To Do](#) [Actions](#)

**Description**  
Add a description...

**Confluence pages** ⓘ ⋮

 [Project plan](#) [TRY TEMPLATE](#) Updated just now 

 Add a comment

## 7. CODING & SOLUTIONING

### Prerequisites:

To complete this project, you must require the following software's, concepts, and packages

### Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using the Flask framework.

### 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.
- CNN Models analyze the image, then prediction is showcased on the Flask UI.

**To accomplish this, we have to complete all the activities and tasks listed below**

o Data Collection.

o Create Train and Test Folders.

o Data Preprocessing.

o Import the ImageDataGenerator library

o Configure ImageDataGenerator class

o Apply ImageDataGenerator functionality to Trainset and Testset

o Model Building

o Import the model building Libraries

o Initializing the model

o Adding Input Layer

o Adding Hidden Layer

o Adding Output Layer

o Configure the Learning Process

o Training and testing the model

o Save the Model

o Application Building

o Create an HTML file

- o Build Python Code

### **Milestone 1: Data Collection**

Collect images of the eye then organize them 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 like Normal, cataract, Diabetic Retinopathy & Glaucoma and they are saved in the respective sub directories with their respective names.

**Download the Dataset-** <https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification>

### **Milestone 2: Image Preprocessing**

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

#### **Activity 1: Import the ImageDataGenerator library**

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from tensorflow Keras

```
#import necessary libraries for the model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

#### **Activity 2: 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.

An instance of the ImageDataGenerator class can be constructed for training and testing.

## Image Data Augmentation

```
data_dir='dataset'
```

```
datagen = ImageDataGenerator(  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    fill_mode='nearest'  
)
```

## Activity 3: Apply ImageDataGenerator functionality to Train Set and Test set

### Loading Our Data And Performing Data Augmentation

```
In [21]: train_ds = tf.keras.utils.image_dataset_from_directory(  
        data_dir,  
        validation_split=0.25,  
        subset="training",  
        seed=123,  
        image_size=(128,128),  
        batch_size=32,  
        label_mode='categorical' # Use categorical labels  
    )  
  
    val_ds = tf.keras.utils.image_dataset_from_directory(  
        data_dir,  
        validation_split=0.25,  
        subset="validation",  
        seed=123,  
        image_size=(128,128),  
        batch_size=32,  
        label_mode='categorical'  
    )
```

```
Found 4217 files belonging to 4 classes.  
Using 3163 files for training.  
Found 4217 files belonging to 4 classes.  
Using 1054 files for validation.
```

## Milestone 3: Model Building

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

### Activity 1: Importing the Model Building Libraries

#### Importing the necessary 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: Initializing the model

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.

```
# Initializing the model
model=Sequential()
```

### Activity 3: Adding CNN Layers

- Adding Dense Layers

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

```
model = keras.Sequential([
    layers.Rescaling(1./255, input_shape=(128,128, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 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')
])
```

### Compiling The Model

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

## Summary Of The Model:

```
In [26]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 128, 128, 16)	448
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 128)	524416
dense_1 (Dense)	(None, 4)	516
Total params: 557,764		
Trainable params: 557,764		
Non-trainable params: 0		

## Activity 7: 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 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.

## Activity 8: Save the Model

The model is saved with .h5 extension as follows An H5 file is a data file saved in the Hierarchical Data Format (HDF).It contains multidimensional arrays of scientific data.

### Save the Model

```
#finally save the model  
tf.keras.models.save_model(model, 'EyeModel.h5')
```

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`



## Fit Model

```
epochs=30

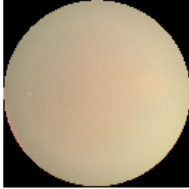
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

## 8. PERFORMANCE TESTING

```
Epoch 1/30
47/47 [=====] - 786s 14s/step - loss: 1.1678 - accuracy: 0.4658 - val_loss: 0.9845 - val_accuracy: 0.6200
Epoch 2/30
47/47 [=====] - 58s 1s/step - loss: 0.8757 - accuracy: 0.6362 - val_loss: 0.7789 - val_accuracy: 0.6740
Epoch 3/30
47/47 [=====] - 65s 1s/step - loss: 0.7206 - accuracy: 0.7082 - val_loss: 0.6134 - val_accuracy: 0.7700
Epoch 4/30
47/47 [=====] - 56s 1s/step - loss: 0.5711 - accuracy: 0.7768 - val_loss: 0.5765 - val_accuracy: 0.7580
Epoch 5/30
47/47 [=====] - 58s 1s/step - loss: 0.5040 - accuracy: 0.8013 - val_loss: 0.5086 - val_accuracy: 0.7660
Epoch 6/30
47/47 [=====] - 55s 1s/step - loss: 0.4306 - accuracy: 0.8330 - val_loss: 0.4419 - val_accuracy: 0.8200
Epoch 7/30
47/47 [=====] - 59s 1s/step - loss: 0.4008 - accuracy: 0.8427 - val_loss: 0.4599 - val_accuracy: 0.8185
Epoch 8/30
47/47 [=====] - 55s 1s/step - loss: 0.3725 - accuracy: 0.8525 - val_loss: 0.4472 - val_accuracy: 0.8165
Epoch 9/30
47/47 [=====] - 58s 1s/step - loss: 0.3259 - accuracy: 0.8752 - val_loss: 0.4148 - val_accuracy: 0.8370
Epoch 10/30
47/47 [=====] - 57s 1s/step - loss: 0.3212 - accuracy: 0.8752 - val_loss: 0.4160 - val_accuracy: 0.8395
Epoch 11/30
47/47 [=====] - 56s 1s/step - loss: 0.2644 - accuracy: 0.8985 - val_loss: 0.3598 - val_accuracy: 0.8640
Epoch 12/30
47/47 [=====] - 56s 1s/step - loss: 0.2558 - accuracy: 0.9027 - val_loss: 0.3528 - val_accuracy: 0.8700
Epoch 13/30
47/47 [=====] - 61s 1s/step - loss: 0.2408 - accuracy: 0.9082 - val_loss: 0.3474 - val_accuracy: 0.8675
Epoch 14/30
47/47 [=====] - 55s 1s/step - loss: 0.2140 - accuracy: 0.9212 - val_loss: 0.3940 - val_accuracy: 0.8380
Epoch 15/30
47/47 [=====] - 59s 1s/step - loss: 0.2038 - accuracy: 0.9245 - val_loss: 0.4295 - val_accuracy: 0.8370
Epoch 16/30
47/47 [=====] - 55s 1s/step - loss: 0.2310 - accuracy: 0.9138 - val_loss: 0.3552 - val_accuracy: 0.8660
Epoch 17/30
47/47 [=====] - 57s 1s/step - loss: 0.1529 - accuracy: 0.9452 - val_loss: 0.3355 - val_accuracy: 0.8745
Epoch 18/30
47/47 [=====] - 56s 1s/step - loss: 0.1661 - accuracy: 0.9393 - val_loss: 0.3494 - val_accuracy: 0.8730
Epoch 19/30
47/47 [=====] - 55s 1s/step - loss: 0.1260 - accuracy: 0.9573 - val_loss: 0.2705 - val_accuracy: 0.8990
Epoch 20/30
47/47 [=====] - 57s 1s/step - loss: 0.1166 - accuracy: 0.9597 - val_loss: 0.2721 - val_accuracy: 0.9025
Epoch 21/30
47/47 [=====] - 55s 1s/step - loss: 0.1144 - accuracy: 0.9590 - val_loss: 0.4013 - val_accuracy: 0.8810
Epoch 22/30
47/47 [=====] - 57s 1s/step - loss: 0.0918 - accuracy: 0.9710 - val_loss: 0.2840 - val_accuracy: 0.8975
Epoch 23/30
47/47 [=====] - 56s 1s/step - loss: 0.0771 - accuracy: 0.9745 - val_loss: 0.2961 - val_accuracy: 0.9035
Epoch 24/30
47/47 [=====] - 62s 1s/step - loss: 0.0684 - accuracy: 0.9792 - val_loss: 0.2663 - val_accuracy: 0.9040
Epoch 25/30
47/47 [=====] - 61s 1s/step - loss: 0.0670 - accuracy: 0.9787 - val_loss: 0.3350 - val_accuracy: 0.8990
Epoch 26/30
47/47 [=====] - 56s 1s/step - loss: 0.0437 - accuracy: 0.9858 - val_loss: 0.3573 - val_accuracy: 0.8830
Epoch 27/30
47/47 [=====] - 57s 1s/step - loss: 0.0828 - accuracy: 0.9718 - val_loss: 0.3081 - val_accuracy: 0.9065
Epoch 28/30
47/47 [=====] - 56s 1s/step - loss: 0.0502 - accuracy: 0.9850 - val_loss: 0.3472 - val_accuracy: 0.8985
Epoch 29/30
47/47 [=====] - 61s 1s/step - loss: 0.0518 - accuracy: 0.9855 - val_loss: 0.3427 - val_accuracy: 0.8965
Epoch 30/30
47/47 [=====] - 56s 1s/step - loss: 0.0316 - accuracy: 0.9918 - val_loss: 0.2976 - val_accuracy: 0.9130
```

## Test the model

```
from tensorflow.keras.preprocessing import image
img = image.load_img('/content/dataset/cataract/1062_right.jpg',target_size =(128,128))
img
```

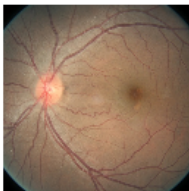


```
x = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred =np.argmax(model.predict(x))
op = {0:'cataract',1:'diabetic_retinopathy',2:'glaucoma',3:'normal'}
op[pred]
```

1/1 [=====] - 0s 53ms/step

'cataract'

```
img = image.load_img('/content/dataset/diabetic_retinopathy/10009_right.jpeg',target_size =(128,128))
img
```



```
x = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred =np.argmax(model.predict(x))
op = {0:'cataract',1:'diabetic_retinopathy',2:'glaucoma',3:'normal'}
op[pred]
```

1/1 [=====] - 0s 44ms/step

'diabetic\_retinopathy'

We were able to correctly predict the eye disease with accuracy of 91%.

## Milestone 4: 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 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.

## Activity 1 : Create HTML Pages

- o We use HTML to create the front end part of the web page.
- o Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- o home.html displays the home page.
- o Intro.html displays an introduction about the project
- o upload.html gives the emergency alert

For more information regarding HTML

- o We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- o Link :CSS , JS

index.html looks like this

Code:

```
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Eye Disease Dectection</title>
  <link href="https://cdn.bootcss.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet">
  <link rel="icon" href="favicon.ico">
  <script src="https://cdn.bootcss.com/popper.js/1.12.9/umd/popper.min.js"></script>
  <script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://cdn.bootcss.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>
  <link href="{{ url_for('static', filename='css/main.css') }}" rel="stylesheet">
  <style>

    .bg-dark {
      background-color: #0c344b!important;
    }
    #result {
      color: #0a1c4ed1;
    }

  </style>
</head>

<body>
```

```

<div class="navbar navbar-dark bg-primary" background-color: #000000>
  <div class="container">
    <div class="navbar-brand" href="#">Eye Disease Detection using Deep Learning</div>
  </div>
</div>
<div class="container">
  <div id="content" class="jumbotron" style="margin-top: 20px" background-color: #000000>
    <div class="text-center">
      <div class="row">
        <div class="col-md-8">
          <div class="h1" style="font-size: 2em">
            <h1>Eye Disease Detection</h1>
          </div>
          <div class="p">
            <p>Eye disease detection using deep learning utilizes neural networks, particularly CNNs, to analyze medical images for conditions such as cataract, diabetic retinopathy, and glaucoma, distinguishing them from normal eyes. This involves training on labeled datasets, preprocessing images, and deploying models for early and accurate disease identification.</p>
          </div>
          <div class="img">
            
          </div>
          <div class="p">
            <p>Upload Image Here To Identify the Disease</p>
          </div>
          <div class="form">
            <div class="form-group">
              <input type="text" class="form-control" placeholder="Upload Image" />
            </div>
            <div class="form-group">
              <input type="button" value="Upload" />
            </div>
          </div>
          <div class="p">
            <p>Result: The classified Disease is : Diabetic Retinopathy</p>
          </div>
          <div class="img">
            
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
</div>
</div>

```

## FrontEND:

### Eye Disease Detection using Deep Learning

#### Eye Disease Detection:

Eye disease detection using deep learning utilizes neural networks, particularly CNNs, to analyze medical images for conditions such as cataract, diabetic retinopathy, and glaucoma, distinguishing them from normal eyes. This involves training on labeled datasets, preprocessing images, and deploying models for early and accurate disease identification.



#### Upload Image Here To Identify the Disease

Choose...



Result: The classified Disease is : Diabetic Retinopathy

## Activity 2: Build python code

### Task 1: Importing Libraries

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

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

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__)

model = load_model("EyeModel.h5", compile=False)
```

### Task 3: Routing to the html Page

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

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.

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

## Showcasing prediction on UI:

```
@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 = (128,128))
        x = image.img_to_array(img)
        print(x)
        x = np.expand_dims(x,axis =0)
        print(x)
        pred =np.argmax(model.predict(x))
        print("prediction",pred)
        index = {0:'Cataract',1:'Diabetic Retinopathy',2:'Glaucoma',3:'Normal'}
        text = "The classified Disease is : " + str(index[pred])

    return text
```

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 types of Garbage in the input image using model.predict function and the result is stored in the result variable.

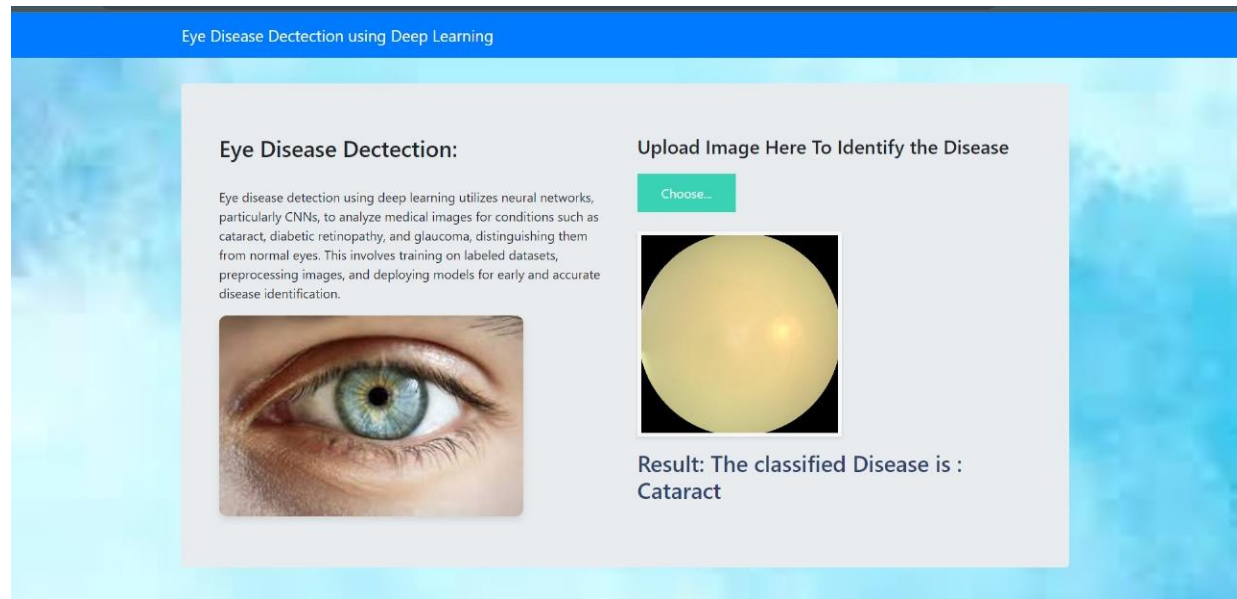
Finally, Run the application

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

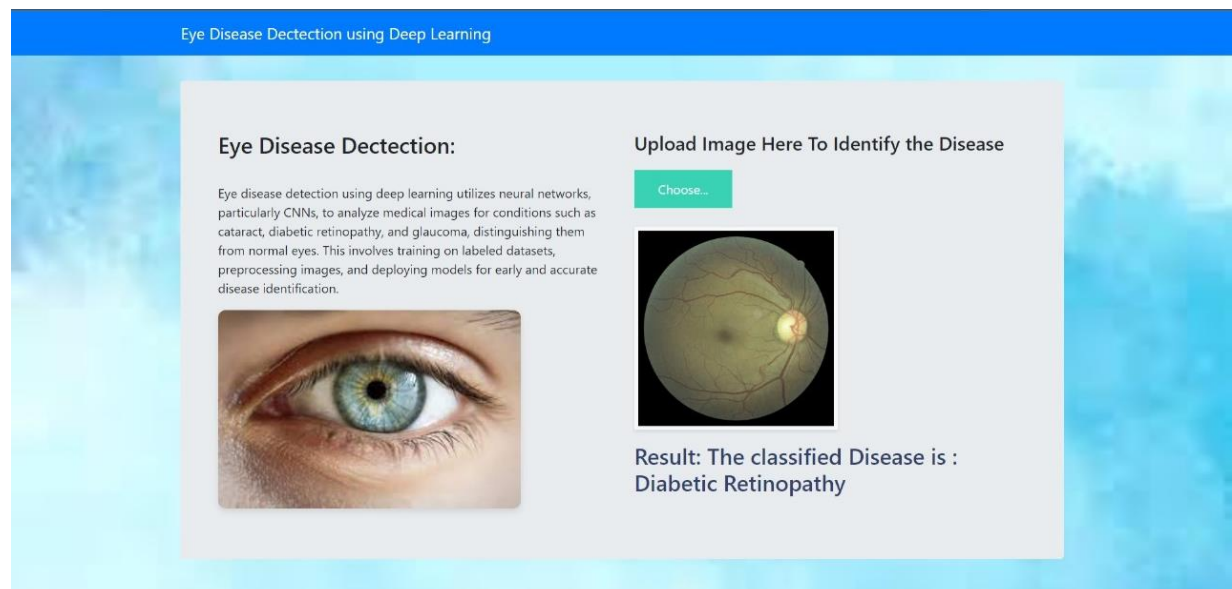
```
if __name__ == '__main__':  
    app.run(debug = False, threaded = False)
```

## 9. FINAL OUTPUT:

### Output 1:



### Output 2:



## 10. ADVANTAGES & DISADVANTAGES

### Advantages:

- High Accuracy: CNNs are known for their ability to achieve high accuracy in image classification tasks, making them suitable for medical diagnosis.
- Feature Learning: Convolutional Neural Networks excel at learning hierarchical features from raw data, enabling automatic extraction of relevant features from eye images.
- Generalization: Once trained on a diverse dataset, CNNs can generalize well to new, unseen data, enhancing the model's adaptability to different cases.
- Automation: The automated nature of CNN-based classification systems can significantly reduce the time and effort required for eye disease diagnosis compared to manual methods.

### Disadvantages:

- Data Dependency: CNNs heavily rely on large and diverse datasets for effective training. Limited or biased datasets may lead to suboptimal performance.
- Computational Intensity: Training deep CNNs can be computationally intensive and may require substantial computing resources, making it challenging for some users or environments.
- Interpretability: CNNs often lack transparency in decision-making, making it challenging to interpret why a particular classification was made. This can be a concern in critical applications like healthcare.
- Overfitting: CNNs may be prone to overfitting, especially with small datasets. Regularization techniques need to be implemented to mitigate this risk.

## 11. CONCLUSION

To conclude, similarly to any viable AI driven application, the Eye Disease Detection Model using CNN holds tremendous value for the public if this base prototype is carefully integrated into real world scenarios after careful scaleup.

The key findings for us whilst doing this project were how different techniques such as ResNet, Classical implementation and Neural Networks fared against each other for this particular project.

Being able to work on this was also a learning experience like nothing has been and the current small success of it definitely instills us with the confidence to work on this even further and hopefully have a market ready Eye Disease Classification Application at our disposal.

## 12. FUTURE SCOPE

AI powered diagnostic services definitely commands attention due to their objective diagnosis and room for customization. With time the technology would only advance and win the faith of more and more potential users.

Talking particularly about our Eye Disease Detection Model, it could be levied to :

- Clinics
- Hospitals
- Government for camps and health drive purposes
- Medical Institutions



For quick and accurate diagnosis of diseases so that every practitioner could focus on treating the problem itself and allowing to connect with more people in need.

In terms of development, we could hone the performance even further with more resources being allocated to research and development. More diseases could be included to support more conditions and the availability of the software can be widened.

### **13. APPENDIX**

Source Code Link:

[EyeDiseaseDetectionModel.ipynb](#)

Github Link:

<https://github.com/Vedhitsk/ImagedetectionAIml>