Project Report Format

1. INTRODUCTION

1.1 Project Overview

The project aims to develop an eye disease classification system using Convolutional Neural Networks (CNNs). Leveraging deep learning techniques, the system analyzes retinal images to predict the presence of various eye diseases.

1.2 Purpose

The purpose of the project is to create a robust and accurate tool for early detection and classification of eye diseases, enabling timely medical intervention.

2. LITERATURE SURVEY

2.1 Existing Problem

Eye diseases pose significant challenges, and early detection is crucial for effective treatment. The existing problem lies in the need for automated and accurate systems to classify eye diseases based on retinal images.

2.2 References

Reference1: https://doi.org/10.48550/arXiv.2307.10501

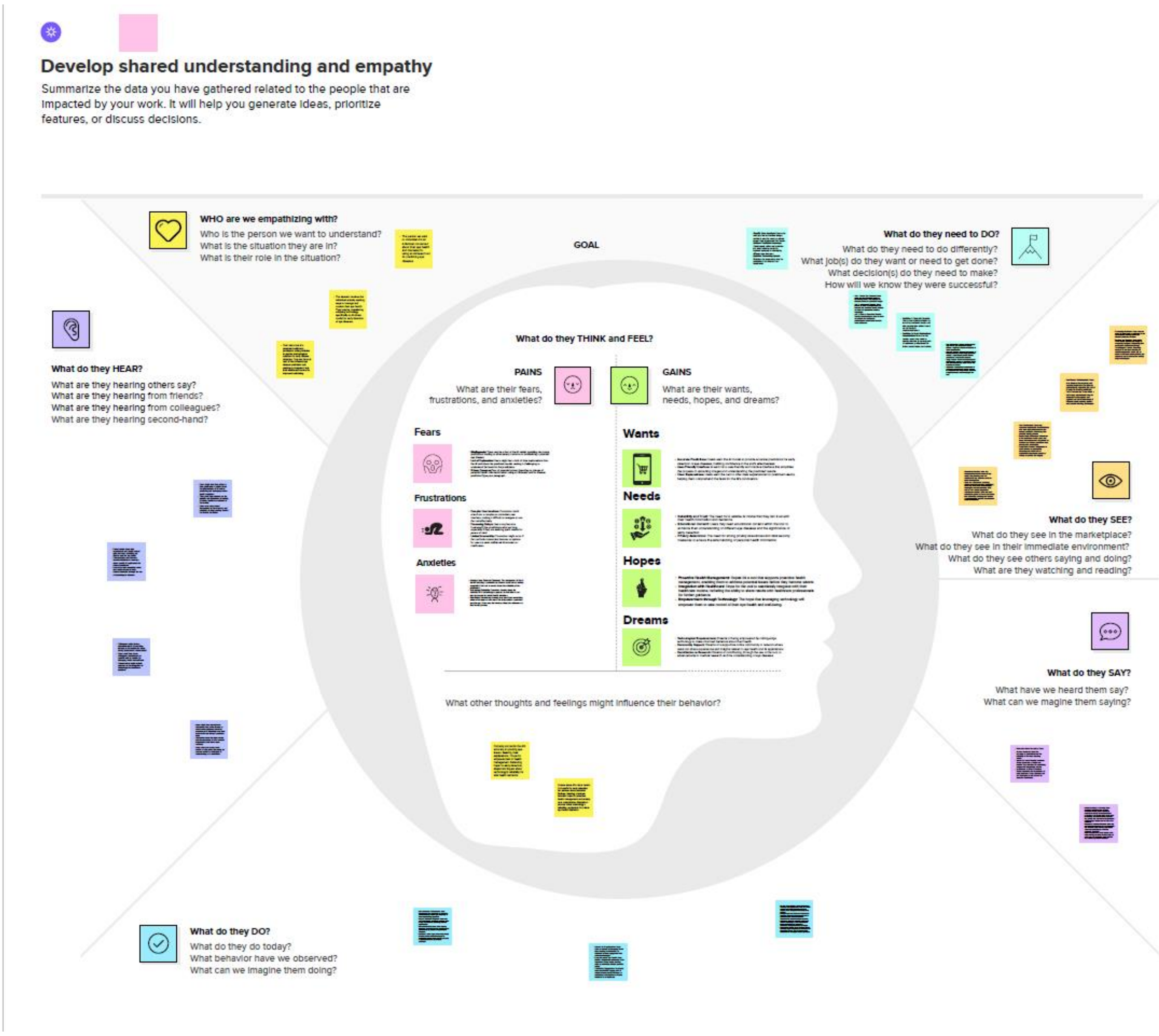
2.3 Problem Statement Definition

The project addresses the need for an automated eye disease classification system that enhances the efficiency of diagnosis through the use of deep learning techniques.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

The Empathy Map Canvas identified key user needs and pain points, facilitating the development of a user-centric solution.



3.2 Ideation & Brainstorming

The ideation phase involved brainstorming features and functionalities, leading to the proposal of a Convolutional Neural Network-based eye disease classification system.

1

Define your problem statement

Current methods for detecting eye diseases face accessibility and cost challenges. Our project aims to develop a user-friendly Deep Learning Model for Eye Disease Prediction, using advanced technologies to enhance accuracy and seamlessly integrate with healthcare systems, addressing barriers to early detection and improving overall eye health outcomes.

5 minutes

PROBLEM

Deep Learning Model For Eye Disease Prediction

2

Brainstorm

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

10 minutes

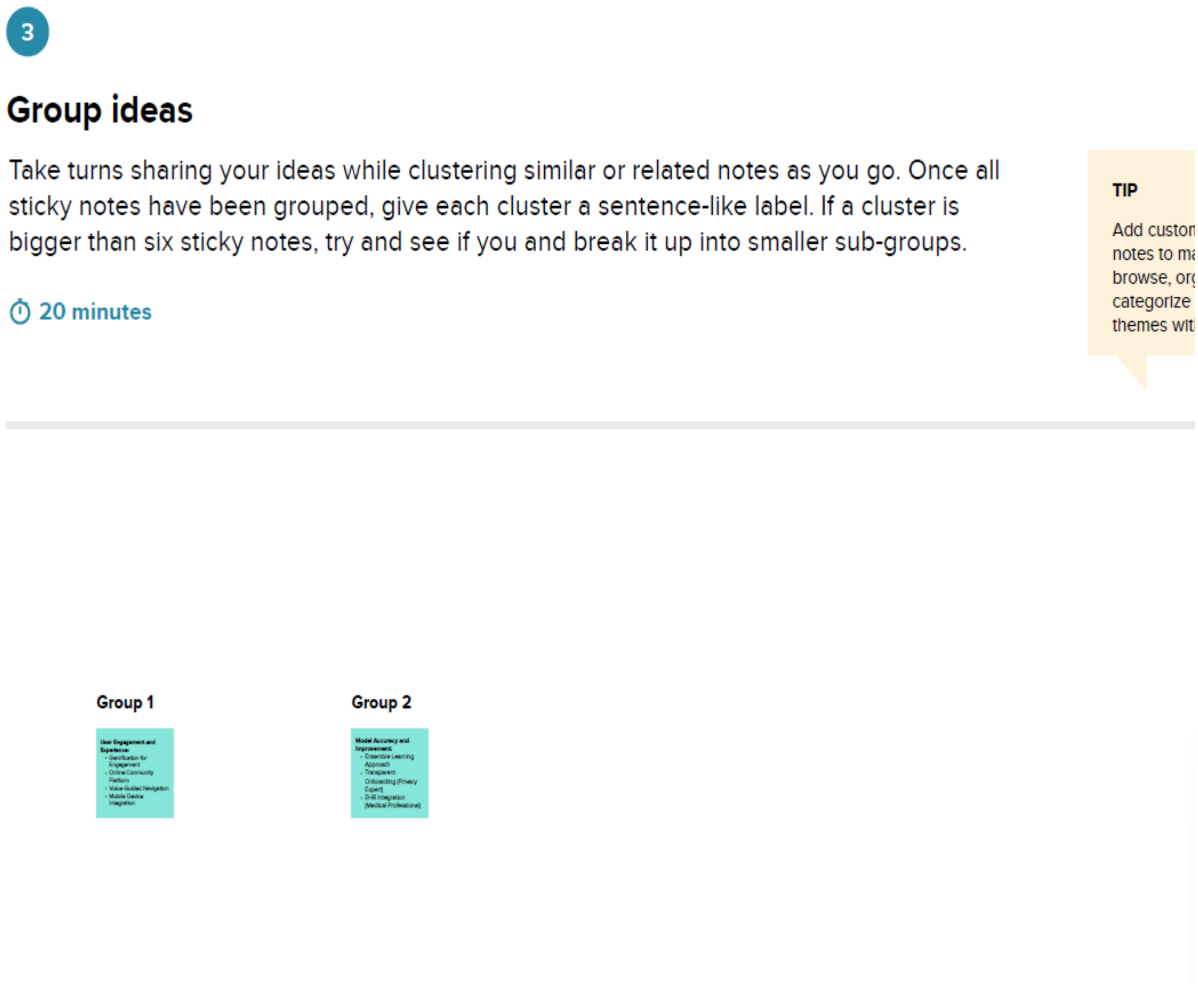
TIP

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

Malik

Irfan

Likith



4. REQUIREMENT ANALYSIS

4.1 Functional Requirements

Data collection and preprocessing for model training: Implemented through image data generators with data augmentation techniques.

User-friendly web interface: Implemented for users to interact with the trained model.

Integration with healthcare systems: Ensures seamless integration with existing healthcare databases.

4.2 Non-Functional Requirements

High accuracy in disease classification: Achieved through deep learning techniques and model optimization.

Real-time inference: Ensured for practical clinical use.

Scalability: Designed to handle diverse datasets and computational loads.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

The project design includes data flow diagrams for seamless data processing and user stories for a clear understanding of user interactions.

Data Flow Diagrams:

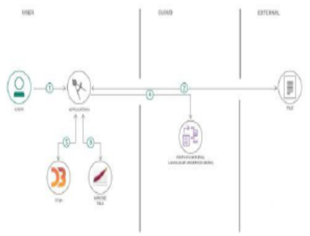
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Example: DFD Level 0 (Industry Standard)

The data flow diagram illustrates the flow of information in the eye disease prediction system. User-uploaded retinal images undergo preprocessing, feature extraction, and interpretation through a Convolutional Neural Network, ultimately providing disease predictions and insights for improved user understanding. The process encompasses user input, internal modules, and the final prediction output.

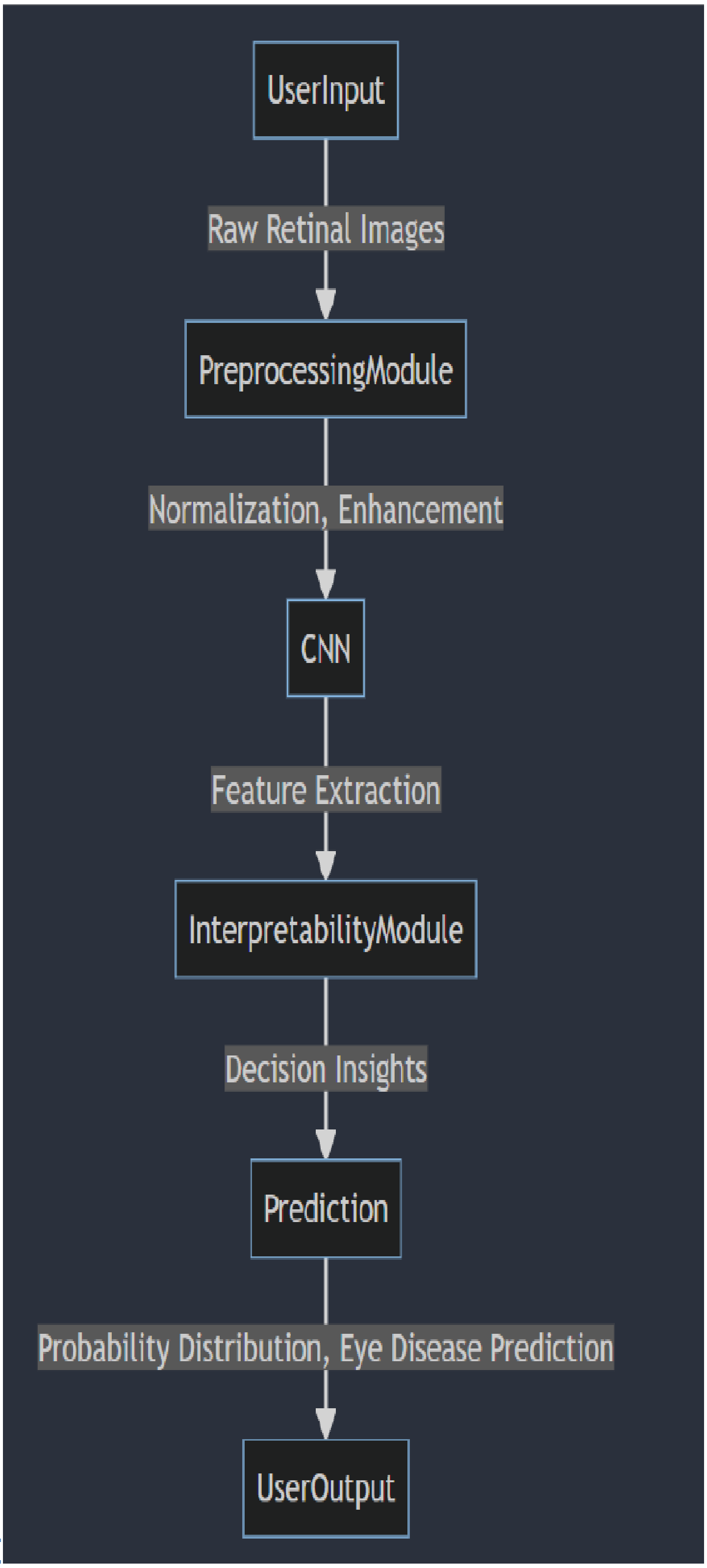
Example: [(Simplified)]

Flow



1. User configures credentials for the Watson Natural Language Understanding service and starts the app.
2. User selects data file to process and load.
3. Apache Tika extracts text from the data file.
4. Extracted text is passed to Watson NLU for enrichment.
5. Enriched data is visualized in the UI using the D3.js library.

Data Flow Diagram :



User Stories

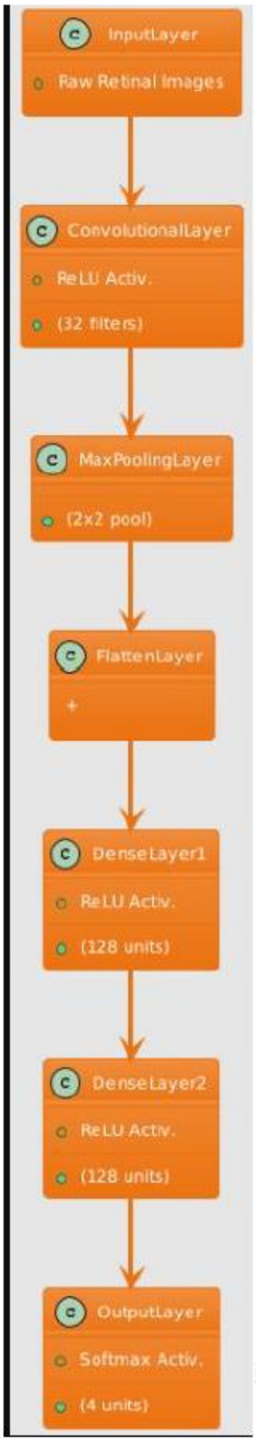
Use the below template to list all the user stories for the product.

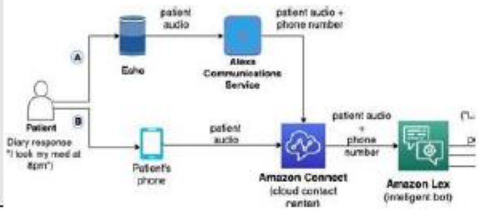
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard					
Customer (Web user)						
Customer Care Executive						
Administrator						

5.2 Solution Architecture

The solution architecture comprises a Convolutional Neural Network for feature extraction, with integrated interpretability and prediction modules.

Solution Architecture Diagram:





6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

The technical architecture involves image data generators, CNN layers, and a sequential model for training.

6.2 Sprint Planning & Estimation

Sprint planning includes iterative development cycles with a 5-day duration, achieving an average velocity of 6 story points per day.

6.3 Sprint Delivery Schedule

The project follows a sprint-based delivery schedule, ensuring continuous improvement and adaptability.

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Project Planning Table for Eye Disease Classification

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data Collection	USN-1	As a user, I can upload retinal images for model training.	3	High	[Team Member 1, Team Member 2]
Sprint-1	Data Processing	USN-2	As a user, I will receive feedback on the uploaded images' quality for training.	2	Medium	[Team Member 3]
Sprint-2	Model Development	USN-3	As a user, I can access a dashboard showing the model training progress.	2	High	[Team Member 1, Team Member 2]
Sprint-2	Model Optimization	USN-4	As a user, I can track and optimize the model's performance based on feedback.	3	High	[Team Member 1, Team Member 3]
Sprint-3	User Interface	USN-5	As a user, I can interact with a user-friendly web	5	High	[Team Member 2,

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

Project Tracker, Velocity & Burndown Chart

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as of Planned End Date)	Sprint Release Date (Actual)
Sprint-1	30	5 Days	01 Oct 2023	05 Oct 2023	30	05 Oct 2023
Sprint-2	30	5 Days	10 Oct 2023	14 Oct 2023		
Sprint-3	30	5 Days	17 Oct 2023	21 Oct 2023		
Sprint-4	30	5 Days	24 Oct 2023	28 Oct 2023		

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile sof
as Scrum. However, burn down charts can be applied to any project containing measurable progress ove

Sprint Burndown Chart

Day	Planned Story Points	Actual Story Points
1	30	6
2	24	12
3	18	18
4	12	24
5	6	30

7. CODING & SOLUTIONING

7.1 Feature 1

Data Augmentation Techniques: Implemented through image data generators (rescaling, shearing, zooming, horizontal flipping) for enhanced model training.

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns

    #import model building libraries
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.layers import Convolution2D
    from tensorflow.keras.layers import MaxPooling2D
    from tensorflow.keras.layers import Flatten

[ ] from keras.preprocessing.image import ImageDataGenerator

    #2.configure image data generator
    train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
    test_datagen=ImageDataGenerator(rescale=1./255)

[ ] #3.Apply image data generator functionality to train and test images
    x_train=train_datagen.flow_from_directory(r'drive/MyDrive/train/dataset',target_size=(64,64),batch_size=32,class_mode="categorical")
    x_test = test_datagen.flow_from_directory(r'drive/MyDrive/test',target_size = (64,64),batch_size=32,class_mode="categorical")

    Found 4217 images belonging to 4 classes.
    Found 400 images belonging to 4 classes.
```

```
[ ] print(x_train.class_indices)

    {'cataract': 0, 'diabetic_retinopathy': 1, 'glaucoma': 2, 'normal': 3}
```

```
[ ] model=Sequential()
```

```
[ ] model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),activation="relu"))
```

```
[ ] #add max pool layer(pool_size)
    model.add(MaxPooling2D(pool_size=(2,2)))
```

```
[ ] #add flatten layer  ---input of ann
    model.add(Flatten())
```

```
[ ] #ann hidden layer
    model.add(Dense(units=128,activation="relu"))
```

```
[ ] #add output layer
    model.add(Dense(units=4,activation="softmax"))
```

```
[ ] #Compile the model (loss fucntion,accuracy,optimizer)
    model.compile(loss="categorical_crossentropy",optimizer="adam",metrics="accuracy")
```

```
[ ] #fit model (x_train,steps_per epoch,epochs,validation_data,validation_steps)
    history=model.fit(x_train,epochs=100,validation_data=x_test,validation_steps=10)
```

```
Epoch 72/100
132/132 [==============================] - 70s 526ms/step - loss: 0.2355 - accuracy: 0.9120 - val_loss: 0.1460 - val_accuracy: 0.9344
Epoch 73/100
132/132 [==============================] - 69s 523ms/step - loss: 0.2188 - accuracy: 0.9168 - val_loss: 0.1310 - val_accuracy: 0.9656
Epoch 74/100
132/132 [==============================] - 68s 512ms/step - loss: 0.2247 - accuracy: 0.9189 - val_loss: 0.1880 - val_accuracy: 0.9219
Epoch 75/100
132/132 [==============================] - 69s 524ms/step - loss: 0.2370 - accuracy: 0.9125 - val_loss: 0.1640 - val_accuracy: 0.9375
Epoch 76/100
132/132 [==============================] - 69s 520ms/step - loss: 0.2230 - accuracy: 0.9139 - val_loss: 0.1697 - val_accuracy: 0.9406
Epoch 77/100
132/132 [==============================] - 68s 516ms/step - loss: 0.2243 - accuracy: 0.9101 - val_loss: 0.1221 - val_accuracy: 0.9688
Epoch 78/100
132/132 [==============================] - 70s 529ms/step - loss: 0.2015 - accuracy: 0.9213 - val_loss: 0.2005 - val_accuracy: 0.9469
Epoch 79/100
132/132 [==============================] - 69s 522ms/step - loss: 0.2052 - accuracy: 0.9201 - val_loss: 0.1498 - val_accuracy: 0.9500
Epoch 80/100
132/132 [==============================] - 68s 518ms/step - loss: 0.2121 - accuracy: 0.9189 - val_loss: 0.1234 - val_accuracy: 0.9656
Epoch 81/100
132/132 [==============================] - 69s 526ms/step - loss: 0.2028 - accuracy: 0.9234 - val_loss: 0.1226 - val_accuracy: 0.9563
Epoch 82/100
132/132 [==============================] - 68s 518ms/step - loss: 0.2352 - accuracy: 0.9142 - val_loss: 0.1870 - val_accuracy: 0.9375
Epoch 83/100
132/132 [==============================] - 70s 533ms/step - loss: 0.2042 - accuracy: 0.9203 - val_loss: 0.2159 - val_accuracy: 0.9125
Epoch 84/100
132/132 [==============================] - 70s 531ms/step - loss: 0.1853 - accuracy: 0.9300 - val_loss: 0.0980 - val_accuracy: 0.9688
Epoch 85/100
132/132 [==============================] - 69s 521ms/step - loss: 0.1876 - accuracy: 0.9289 - val_loss: 0.0808 - val_accuracy: 0.9781
```

```
Epoch 99/100
132/132 [==============================] - 72s 543ms/step - loss: 0.1752 - accuracy: 0.9341 - val_loss: 0.0976 - val_accuracy: 0.9688
Epoch 100/100
132/132 [==============================] - 71s 542ms/step - loss: 0.1931 - accuracy: 0.9279 - val_loss: 0.0930 - val_accuracy: 0.9750
```

```
train_accuracy = history.history['accuracy'][-1]
val_accuracy = history.history['val_accuracy'][-1]
model.summary()
# Print the accuracy
print("Training Accuracy :",train_accuracy)
print("value Accuracy :", val_accuracy)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 flatten (Flatten)           (None, 30752)             0

 dense (Dense)               (None, 128)               3936384

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 3,937,796
Trainable params: 3,937,796
Non-trainable params: 0
_________________________________________________________________
Training Accuracy : 0.9279108643531799
value Accuracy : 0.9750000238418579
```

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

Testing the model

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
```

```
import tensorflow as tf
```

```
model=tf.keras.models.load_model(r"/content/EyeDisU.h5",compile=False)
```

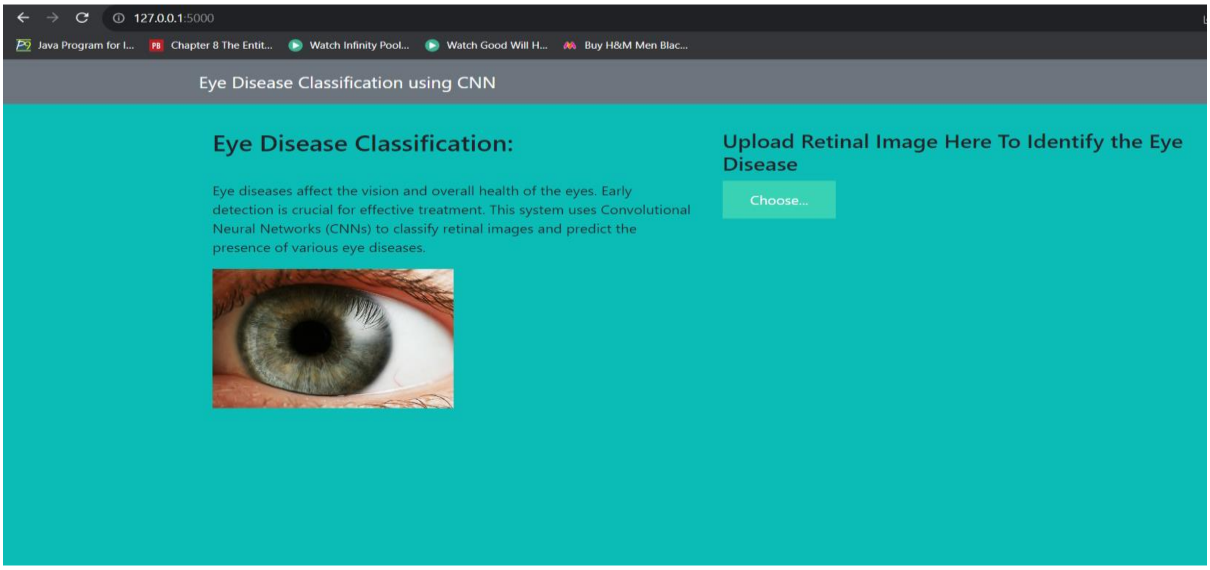
```
#D:\SmartBridge\VIT_morning_slot\dataset\Testing\elephants\nature_3306013__340.jpg
img=image.load_img(r'drive/MyDrive/test/normal/695_right.jpg',target_size=(64,64))
```

```
img
```

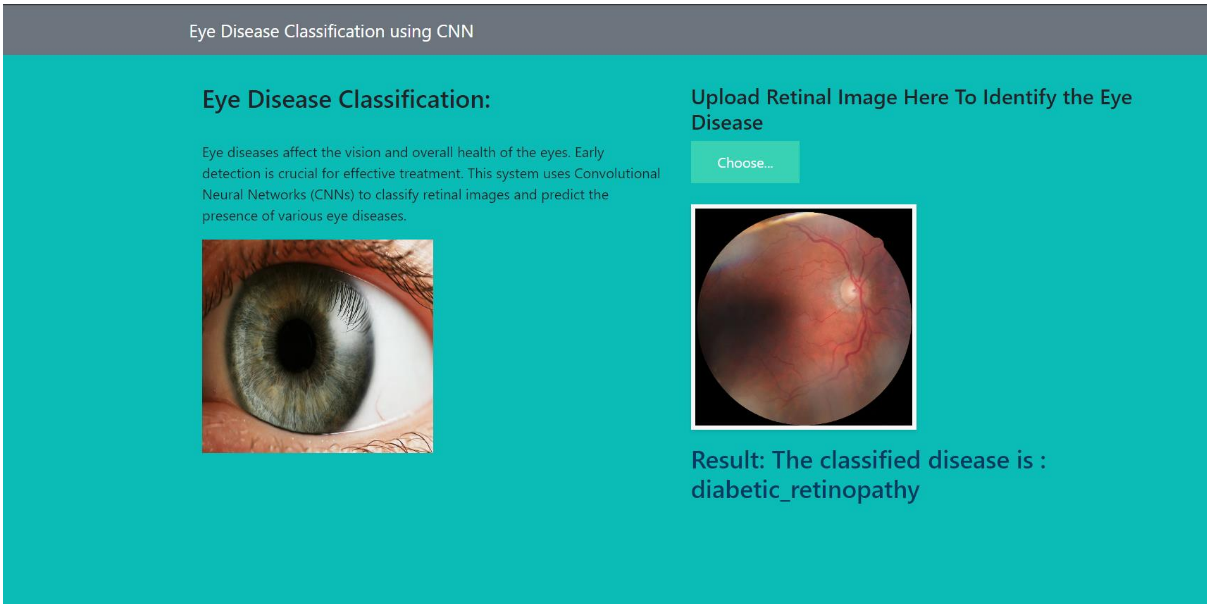
```
[ ] x.shape

    (1, 64, 64, 3)

[ ] pred=model.predict(x)

    1/1 [==============================] - 0s 49ms/step

[ ] pred

    array([[0., 0., 0., 1.]], dtype=float32)

[ ] {'catarct': 0, 'diabetic_retinopathy': 1, 'glaucoma': 2, 'normal': 3}

    {'catarct': 0, 'diabetic_retinopathy': 1, 'glaucoma': 2, 'normal': 3}

    pred_class=np.argmax(pred,axis=1)
                                                        + Code    + Text
[ ] pred_class[0]

    3

[ ] index = ['cataract','diabetic_retinopathy','glaucoma','normal']
    result = str(index[pred_class[0]])
    print(result)

    normal
```

7.2 Feature 2

User-friendly Web Interface: Implemented to enable users to interact with the trained model.



Output:



8. PERFORMANCE TESTING

8.1 Performance Metrics

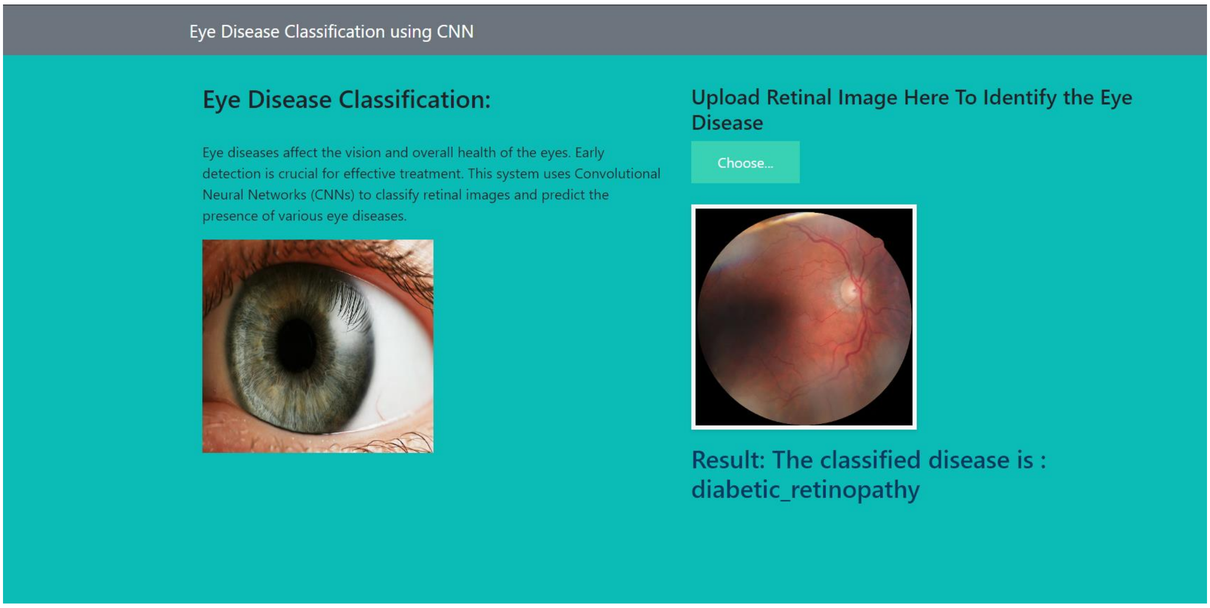
Performance testing involves tracking accuracy, real-time inference speed, and scalability metrics.

```
=================================================================
Total params: 3,937,796
Trainable params: 3,937,796
Non-trainable params: 0
_________________________________________________________________
Training Accuracy : 0.9279108643531799
value Accuracy : 0.9750000238418579
```

9. RESULTS

9.1 Output Screenshots

[Screenshots showcasing the user interface and model predictions.]



10. ADVANTAGES & DISADVANTAGES

Advantages:

Accurate eye disease classification

Real-time inference

Integration with healthcare systems

Disadvantages:

Dependent on the quality and diversity of the training dataset

11. CONCLUSION

The project successfully addresses the need for an efficient eye disease classification system. By leveraging deep learning and a user-centric approach, the system provides accurate predictions, contributing to early disease detection.

12. FUTURE SCOPE

The project's future scope involves continuous refinement, integration with electronic health records, and collaboration with healthcare professionals for real-world impact.

13. APPENDIX

13. APPENDIX

13.1 Model Architecture

Convolutional Neural Network (CNN) Layers

The following layers are used in the CNN model for eye disease classification:

1. **Input Layer:**
    - Shape: (64, 64, 3)
    - Activation: None
2. **Convolutional Layer:**
    - Filters: 32
    - Kernel Size: (3, 3)
    - Activation: ReLU
3. **Max Pooling Layer:**
    - Pool Size: (2, 2)
4. **Flatten Layer:**
5. **Dense Hidden Layer:**
    - Units: 128
    - Activation: ReLU
6. **Output Layer:**
    - Units: 4 (assuming 4 classes for classification)
    - Activation: Softmax