

DIAGNOSIS OF EYE FUNDUS DISEASE CLASSIFICATION AND SEGMENTATION USING ARTIFICIAL INTELLIGENCE TECHNIQUES

Submitted in partial fulfilment of the requirements for the award
of Master of Science in Bioinformatics and Data Science

BY

B. HARIDHRA (42739012)



**DEPARTMENT OF BIOINFORMATICS SCHOOL OF
BIO AND CHEMICAL ENGINEERING**

**SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY**

(DEEMED TO BE UNIVERSITY) Accredited with Grade “A++” by NAAC I 12B
Status by UGC I Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

APRIL – 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with “A++” grade by NAAC | 12B Status by UGC | Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai,

Chennai-600119 www.sathyabama.ac.in

DEPARTMENT OF BIOINFORMATICS

BONAFIDE CERTIFICATE

This is to certify that this Project Report is to bonafide work of **HARIDHRA B (42739012)** who carried out the final year project entitled “**Diagnosis of eye fundus disease classification and segmentation using AI techniques**” under our supervision from January 2024 to May 2024.

A handwritten signature in blue ink, appearing to read 'Swetha'.

Dr. SWETHA SUNKAR
Internal Guide

A handwritten signature in blue ink, appearing to read 'Jemmy Christy'.

Dr. H. JEMMY CHRISTY
Head of the Department

Submitted for Viva Voce Examination held on _____



Internal Examiner



External Examiner

DECLARATION

I, **B. HARIDHRA (42739012)**, hereby declare that the Project Report entitled “**Diagnosis of eye fundus disease classification and segmentation using AI Techniques**” done by me under the guidance of **Dr. SWETHA SUNKAR** is submitted in partial fulfilment of the requirements for the award of Master of Science in Bioinformatics and Data Science.

DATE: 20-04-2024

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management, SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY**, for their kind encouragement in doing this project for completing it successfully. I am grateful to them.

I extend my sincere gratitude to **Dr. REKHA CHAKRAVARTHY, Dean of the School of Science and Humanities**, and I am deeply grateful for her mentorship and leadership, which have been a constant source of inspiration. Her dedication to fostering a conducive academic environment has been instrumental in the successful completion of this thesis.

I convey my thanks to **DR. H. JEMMY CHRISTY, Head of the Department, Department of Bioinformatics**, for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. SWETHA SUNKAR, S** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Bioinformatics who were helpful in many ways for the completion of the project.

ABSTRACT

Eye fundus diseases are critical conditions that can lead to severe vision impairment and even permanent blindness if not diagnosed and treated promptly. Manual diagnosis of these diseases is time-consuming and heavily reliant on the expertise of ophthalmologists. This research aims to develop an efficient and accurate diagnostic system for eye fundus disease classification and segmentation using artificial intelligence techniques.

The study involves the compilation of a comprehensive dataset of eye fundus images, encompassing various types of diseases, including diabetic retinopathy, age-related macular degeneration, glaucoma, and others. Each image is accompanied by corresponding ground-truth annotations provided by expert ophthalmologists for segmentation.

The experimental results demonstrate the effectiveness and reliability of the proposed system in accurately classifying eye fundus diseases and segmenting affected regions within the images. The AI models achieve high accuracy and provide valuable insights into the presence and extent of various fundus diseases.

Keywords: Eye Fundus Disease, Artificial Intelligence, Deep Learning, Disease Classification, Segmentation, Convolutional Neural Networks, U-Net, Ophthalmology, Medical Imaging, Diagnostic System.

LIST OF CONTENT

CHAPTER	TITLE	PAGE NO
	ABSTRACT	ii
	LIST OF FIGURES	IV
01.	INTRODUCTION	1
1.1	ARTIFICIAL INTELLIGENCE	2
1.2	DEEP LEARNING	3
1.3	EYE FUNDUS	4
02.	LITERATURE SURVEY	5
03.	SYSTEM STUDY	
3.1	AIM	7
3.2	OBJECTIVES	7
3.3	SCOPE OF THE PROJECT	8
04.	MATERIALS AND METHODOLOGY	
4.1	METHODOLOGY	8
4.2	CNN MODEL	9
4.3	MAX POOLING	10
4.4	ARTIFICIAL NEURAL NETWORK	13
4.5	MODEL BUILDING ALGORITHM	18
4.6	DEPLOYMENT	24
4.7	DJANGO	24
4.8	HTML	25
4.9	CSS	25
05.	RESULT AND DISCUSSION	
5.1	WEB PAGE CREATION	26
5.2	IMPORTING DATA SET	30

5.3	DATA PREPARATION	31
5.4	PREDICTION OF DISEASE	36
5.5	OUTPUT	38
06.	CONCLUSION	
07.	REFERENCES	42
08.	APPENDIX	46

LIST OF FIGURES

S.NO	FIGURES	PAGE NO
1.1	Deep learning flow chart	3
4.1	Architecture of Alex Net	20
4.2	Architecture of LE Net	21
5.1	Web page for registration	30
5.2	Web page for eye disease prediction	31
5.3	Cataract data set 1	34
5.4	Cataract data set 2	34
5.5	Diabetic retinopathy dataset 1	35
5.6	Diabetic retinopathy dataset 2	35
5.7	Glaucoma dataset 1	36
5.8	Glaucoma dataset 2	36
5.9	Cataract disease result	39
5.10	Diabetic Retinopathy result	39
5.11	Glaucoma result	40
5.12	No disease predicted result	40

CHAPTER 1

INTRODUCTION

The diagnosis of eye fundus diseases plays a crucial role in the early detection and management of various ocular conditions, such as diabetic retinopathy, macular degeneration, and glaucoma. Timely and accurate identification of these diseases is essential to prevent vision loss and improve patient outcomes.

The integration of artificial intelligence (AI) techniques has revolutionized the field of ophthalmology, offering advanced tools for the classification and segmentation of eye fundus images. Artificial intelligence, particularly machine learning algorithms, has shown remarkable capabilities in analysing large datasets of eye fundus images.

These techniques enable automated identification and classification of subtle pathological changes that might be challenging for human observers to detect. The development of AI models for eye fundus disease diagnosis involves training algorithms on diverse datasets, encompassing a wide range of retinal pathologies and normal variations. This training allows the AI models to learn patterns and features indicative of specific diseases, paving the way for robust and accurate automated diagnostics.

1.1 ARTIFICIAL INTELLIGENCE:

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

AI research has tried and discarded many different approaches during its lifetime, including simulating the Eye fundus diseases, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. In the first decades of the 21st century, highly mathematical statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

To solve these problems, AI researchers use versions of search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

Learning processes. This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

Reasoning processes. This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

Self-correction processes. This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible. (Xue et al. 2023)

1.2. DEEP LEARNING

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human disease so deep learning is also a kind of mimic of human disease. It's on hype nowadays because earlier we did not have that much processing power and a lot of data.

A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In disease approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbours.

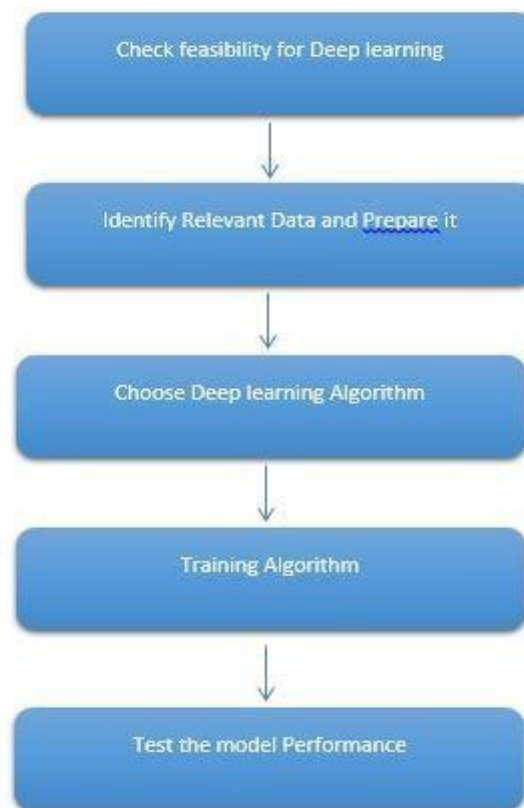


FIG:1.1 DEEPLARNING FLOW CHART

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological disease. Specifically, neural networks tend to be static and symbolic, while the biological disease of most living organisms is dynamic (plastic) and analogue. (Xue et al. 2023; Grzybowski 2021)

1.3. EYE FUNDUS DISEASE:

To develop an advanced diagnostic tool for eye fundus disease classification and segmentation using cutting-edge artificial intelligence techniques. Leveraging the power of deep learning algorithms, the system offers an efficient and accurate solution to assist ophthalmologists in diagnosing and treating various eye fundus diseases promptly. A comprehensive dataset of eye fundus images is collected, comprising diverse cases of different eye fundus diseases, including diabetic retinopathy, agerelated macular degeneration, glaucoma, and others. The dataset is annotated by expert ophthalmologists to provide ground-truth segmentation masks for each image. The proposed system employs Convolutional Neural Networks (CNNs) for eye fundus disease classification.

CHAPTER 2

LITERATURE SURVEY

- ❖ Ademola E. et al, (2023) explains that the frequency of eye-related diseases has risen significantly over the world, increasing the number of people suffering from acute conditions. According to the World Health Organization (WHO), around 2.2 billion people worldwide are visually impaired. Alarmingly, almost 1 billion instances could have been avoided if early detection had occurred. In the United States alone, an estimated 40 million people are plagued by varied degrees of serious eye-related disorders, with a major focus on retinal conditions such as glaucoma.
- ❖ S. Sengupta, et al, (2020), explains that more than 40 million people in the United States suffer from acute eye illnesses, which can result in total vision loss if not addressed. Many of these disorders affect the retina. The most prevalent retinal illnesses include glaucoma, diabetic retinopathy, and age-related macular degeneration. Glaucoma is one of the leading causes of blindness, with about 80 million people worldwide expected to suffer from it by 2020.
- ❖ Tamoor Aziz, Ilesanmi et al, (2023) explains that This work describes an algorithm for detecting and segmenting hemorrhages. Previous investigations by the research community indicate that two types of hemorrhages are difficult to distinguish. The first type of hemorrhage occurs near the retinal border, which is merged with the black background. The second type of hemorrhage is related to blood vessels. The suggested algorithm divides both categories satisfactorily. Benchmarking is carried out using cutting-edge methods, and the results are reported using quality evaluation standards. The SWAT algorithm (proposed approach) divides hemorrhages efficiently and precisely. Our results show that the suggested method outperforms other methods in terms
- ❖ Md Mohaimenul Islam, Hsuan-Chia Yang et al, (2021) (explains that, Deep learning can help detect referable DR with high sensitivity, specificity, and repeatability. The use of a DL-based automated approach could transform how

DR is diagnosed in the future. Automated tools may improve the quality of DR screening, increase access to healthcare, and lower the cost of screening. Early detection and appropriate treatment may prevent the beginning of the disease or help to stop its progression at an earlier stage.

- ❖ Rishab Gargeya et al, (2023) explains that, Throughout the world, diabetic retinopathy, or DR, is one of the main causes of avoidable blindness. There is an unmet need to do retinal screening exams on all diabetes individuals, and many cases of DR go undetected and untreated. The goal of this research was to create reliable diagnostic tools for automated DR screening. Reducing the pace of vision loss would be made possible by referring eyes with DR to an ophthalmologist for additional assessment and treatment, allowing for prompt and precise diagnosis.
- ❖ Antal et al, (2012) explains that, A suggested approach to the detection of diabetic retinopathy (DR) is an ensemble-based one. This method is based on features that are taken from the output of many algorithms for processing retinal images. These features include anatomical (macula, optic disk), lesion specific (microaneurysms, exudates), and image-level (quality evaluation, pre-screening, AM/FM) components.
- ❖ Andras Hajdu et al, (2022), explains that, an overview of automated detection techniques currently in use for diabetic retinopathy screening. We additionally describe a two-step enhanced detection approach. In order to categorize the input digital fundus images according to the severity of anomalies, a pre-screening algorithm is first taken into consideration. Robust lesion detecting techniques will not be used to analyse an image further if it is determined to be substantially aberrant. We present a novel feature extraction method based on clinical observations as an additional enhancement.

- ❖ Niemeijer et al, (2009) explains that, the development of autonomous screening systems for diabetic retinopathy requires the reliable evaluation of the image quality of retinal screening images. A retinal screening image quality threshold system is given that can automatically assess if an image is good enough for automatic analysis. The foundation of the system is the idea that, in accordance with a specific pre-defined distribution, an image of adequate quality should have specific image structures. To provide a compact representation of the picture structures existing in an image, we cluster filter bank response vectors. This compact form is used to train a statistical classifier to discriminate between normal and low-quality photos, together with raw histograms of the R, G, and B

CHAPTER 3

AIM & SCOPE

3.1. AIM:

The aim in the diagnosis of eye fundus disease classification and segmentation using artificial intelligence (AI) techniques is to enhance the efficiency, accuracy, and speed of identifying and categorizing various pathological conditions affecting the retina. The eye fundus, or the back of the eye, contains critical information about the health of the retina and is instrumental in diagnosing conditions such as diabetic retinopathy, macular degeneration, and glaucoma. Leveraging AI techniques in this domain aims to revolutionize the traditional methods of diagnosis, providing a more sophisticated and automated approach to the analysis of fundus images.

3.2. Objectives:

1. To collect the four different types of eye fundus datasets from the Kaggle database.
2. To create a web app for deployment using HTML, CSS source code.
3. Deploying the model in Django Framework and predicting the output
4. To import dataset for training and testing the data Models
5. To prepare the dataset of four types of eye disease and for validation purposes
6. To load the trained model using tensorflow keras model.
7. To import the image in created web page to predict different types of eye disease

3.3 Scope:

The scope of utilizing artificial intelligence (AI) techniques in the diagnosis of eye fundus diseases encompasses a comprehensive approach to disease classification and segmentation. By leveraging advanced machine learning algorithms, deep learning models, and image processing techniques

CHAPTER 4

MATERIALS AND METHODOLOGY

4.1. METHODOLOGY

Preprocessing:

Preprocessing and training the model (CNN): The dataset is pre-processed such as Image reshaping, resizing and conversion to an array form. Similar processing is also done on the test image. A dataset consisting of about 4 different types Cataract, Diabetic retinopathy, Glaucoma, Normal out of which any image can be used as a test image for the software

Training dataset:

The train dataset is used to train the model (CNN) so that it can identify the test image and the disease it has CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify the different Cataract, Diabetic retinopathy, Glaucoma, Normal Classification image contained in the dataset. After successful training and preprocessing, comparison of the test image and trained model takes place to predict.

4.2. CNN Model steps:

Conv2d:

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer.

Whether or not an input feature falls within this “roughly same location”, gets determined directly by whether it’s in the area of the kernel that produced the output or not. This means the size of the kernel directly determines how many (or few) input features get combined in the production of a new output feature.

This is all in pretty stark contrast to a fully connected layer. In the above example, we have $5 \times 5 = 25$ input features, and $3 \times 3 = 9$ output features. If this were a standard fully connected layer, you’d have a weight matrix of $25 \times 9 = 225$ parameters, with every output feature being the weighted sum of every single input feature. Convolutions allow us to do this transformation with only 9 parameters, with each output feature, instead of “looking at” every input feature, only getting to “look” at input features coming from roughly the same location.

Do take note of this, as it’ll be critical to our later discussion.

4.3. MaxPooling2D layer:

Down samples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool size) for each channel of the input. The window is shifted by strides along each dimension.

The resulting output, when using the "valid" padding option, has a spatial shape (number of rows or columns) of: $\text{output shape} = \text{math.floor}((\text{input shape} - \text{pool size}) / \text{strides}) + 1$ (when input shape $\geq$ pool size)

The resulting output shape when using the "same" padding option is: $\text{output shape} = \text{math.floor}((\text{input shape} - 1) / \text{strides}) + 1$

Arguments

- `pool_size`: integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
- `strides`: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to `pool_size`.
- `padding`: One of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of

the input such that output has the same height/width dimension as the input.

- data format: A string, one of channels last (default) or channels first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels first correspond to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Input shape

- If data_format='channels_last': 4D tensor with shape (batch_size, rows, cols, channels).
- If data_format='channels_first': 4D tensor with shape (batch_size, channels, rows, cols).

Output shape

- If data_format='channels_last': 4D tensor with shape (batch_size, pooled_rows, pooled_cols, channels).
- If data_format='channels_first': 4D tensor with shape (batch_size, channels, pooled_rows, pooled_cols).

Flatten layer

It is used to flatten the dimensions of the image obtained after convolving it. Dense: It is used to make this a fully connected model and is the hidden layer. Dropout: It is used to avoid over fitting on the dataset and dense is the output layer contains only one neuron which decide to which category image belongs.

Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4)

Flatten has one argument as follows keras.

layers. Flatten (data_format = None)

data_format is an optional argument and it is used to preserve weight ordering when switching from one data format to another data format. It accepts either

channels_last or channels_first as value. channels_last is the default one and it identifies the input shape as (batch_size, channels) whereas channels_first identifies the input shape as (batch_size, channels,)

Dense layer

Dense implements the operation: $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$ where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). These are all attributes of Dense.

Arguments

- units: Positive integer, dimensionality of the output space.
- activation: Activation function to use. If you don't specify anything, no activation is applied (i.e. "linear" activation: $a(x) = x$).
- use_bias: Boolean, whether the layer uses a bias vector.
- kernel_initializer: Initializer for the kernel weights matrix.
- bias_initializer: Initializer for the bias vector.
- kernel_regularizer: Regularize function applied to the kernel weights matrix.
- bias_regularizer: Regularize function applied to the bias vector.
- activity_regularizer: Regularize function applied to the output of the layer (its "activation").
- kernel_constraint: Constraint function applied to the kernel weights matrix.

Input shape

N-D tensor with shape: (batch size, input dim). The most common situation would be a 2D input with shape (batch size, input dim).

Output shape

N-D tensor with shape: (batch size, ..., units). For instance, for a 2D input with shape (batch size, input dim), the output would have shape (batch size, units).

Dropout_layer

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by $1 / (1 - \text{rate})$ such that the sum over all inputs is unchanged.

Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using `model.Fit`, training will be appropriately set to True automatically, and in other contexts, you can set the explicitly to True when calling the layer.

(This is in contrast to setting `trainable=False` for a Dropout layer. `trainable` does not affect the layer's behavior, as Dropout does not have any variables/weights that can be frozen during training.)

Arguments

- **rate**: Float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch size, 1, features)`.
- **seed**: A Python integer to use as random seed.

Image Data Generator:

It is that rescales the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This Image Data Generator includes all possible orientation of the image.

Validation process:

Validation data is used to feed the validation/test data into the model. Validation steps denotes the number of validation/test samples.

4.4. Artificial Neural Networks:

Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.

ARCHITECTURE OF CNN

CONVOLUTIONAL NEURAL NETWORK:

A Convolutional neural network (CNN) is one type of Artificial Neural Network. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.

Models API:

There are three ways to create Keras models:

- The Sequential model, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).
- The Functional API, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.
- Model subclassing, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

Types of Keras Models

Models in keras are available in two types:

- Keras Sequential Model
- Keras Functional API

1. Sequential Model in Keras

It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs.

It is best for simple stack of layers which have 1 input tensor and 1 output tensor.

This model is not suited when any of the layer in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

2. Functional API in Keras

It provides more flexibility to define a model and add layers in keras. Functional API allows us to create models that have multiple input or output.

It also allows us to share these layers. In other words. we can make graphs of layers using Keras functional API.

As functional API is a data structure, it is easy to save it as a single file that helps in recreating the exact model without having the original code. Also its easy to model the graph here and access its nodes as well.

Model Subclassing in Keras

Sequential model does not allow you much flexibility to create your models. Functional API also only has a little of customization available for you. But you may create your own fully-customizable models in Keras. This is done by subclassing the Model class and implementing a call method. Input () is used to instantiate a Keras tensor.

A Keras tensor is a symbolic tensor-like object, which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.

For instance, if a, b and c are Keras tensors, it becomes possible to do: model = Model (input= [a, b], output=c)

kernels:

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations; each convolution is a type of affine function.

In computer vision the input is often a 3 channel RGB image. For simplicity, if we take a greyscale image that has one channel (a two-dimensional matrix) and a 3x3 convolutional kernel (a two-dimensional matrix). The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the images pixel values. Then the kernel strides down vertically to subsequent rows. Note, the filter may stride over one or several pixels at a time, this is detailed further below.

In other non-vision applications, a one-dimensional convolution may slide vertically over an input matrix.

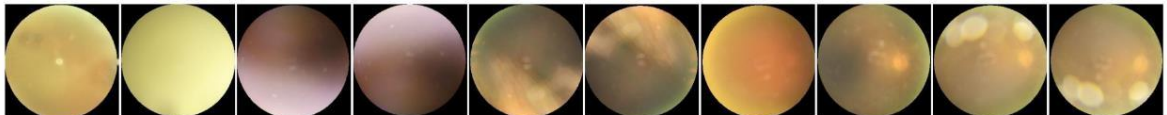
Normalisation:

Normalisation is the process of subtracting the mean and dividing by the standard deviation. It transforms the range of the data to be between -1 and 1 making the data use the same scale, sometimes called Min-Max scaling.

It is common to normalize the input features, standardising the data by removing the mean and scaling to unit variance. It is often important the input features are centred around zero and have variance in the same order. With some data, such as images the data is scaled so that it's range is between 0 and 1, most simply dividing the pixel values by 255.

TRAINING DATA FOR CATARACT:

```
===== Images in: Dataset/cataract
Images_count : 301
Min_width : 256
Max_width : 256
Min_height : 256
Max_height : 256
```



TRAINING DATA FOR DIABETIC RETINOPATHY:

```
===== Images in: Dataset/diabetic_retinopathy
Images_count : 300
Min_width : 512
Max_width : 512
Min_height : 512
Max_height : 512
```



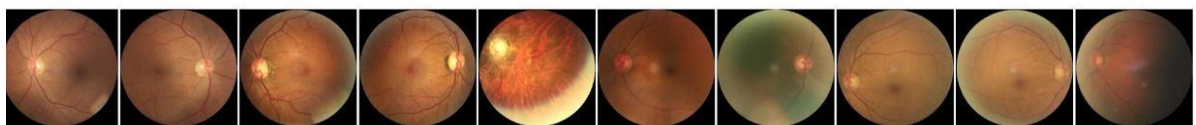
TRAINING DATA FOR GLAUCOMA:

```
===== Images in: Dataset/glaucoma
Images_count : 300
Min_width : 256
Max_width : 256
Min_height : 256
Max_height : 256
```



TRAINING DATA FOR SEBORRHEIC NORMAL:

```
===== Images in: Dataset/normal
Images_count : 300
Min_width : 512
Max_width : 512
Min_height : 512
Max_height : 512
```



Batch normalisation:

Batch normalisation has the benefits of helping to make a network output more stable predictions, reduce overfitting through regularisation and speeds up training by an order of magnitude.

Batch normalisation is the process of carrying normalisation within the scope activation layer of the current batch, subtracting the mean of the batch's activations and dividing by the standard deviation of the batch's activations.

This is necessary as even after normalizing the input as some activations can be higher, which can cause the subsequent layers to act abnormally and makes the network less stable. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. Batch normalization works differently during training and during inference.

During inference (i.e. when using `evaluate ()` or `predict ()` or when calling the layer/model with the argument `training=False` (which is the default), the layer normalizes its output using a moving average of the mean and standard deviation

of the batches it has seen during training. That is to say, it returns $\gamma * (\text{batch-self. moving_mean}) / \sqrt{(\text{self. moving_var} + \epsilon)} + \beta$.

`self. moving_mean` and `self. moving_var` are non-trainable variables that are updated each time the layer is called in training mode, as such:

- `moving_mean = moving_mean * momentum + mean(batch) * (1 - momentum)`
- `moving_var = moving_var * momentum + var(batch) * (1 - momentum)`

As such, the layer will only normalize its inputs during inference *after having been trained on data that has similar statistics as the inference data*.

4.5. MODEL BUILDING ALGORITHM:

- ALEXNET
- LENET

Alex NET:

Alex Net architecture consists of 5 convolutional layers, 3 maxpooling layers, 2 normalization layers, 2 fully connected layers, and 1 SoftMax layer. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. The pooling layers are used to perform max pooling. Alex Net contained eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid.

Architecture of Alex Net:

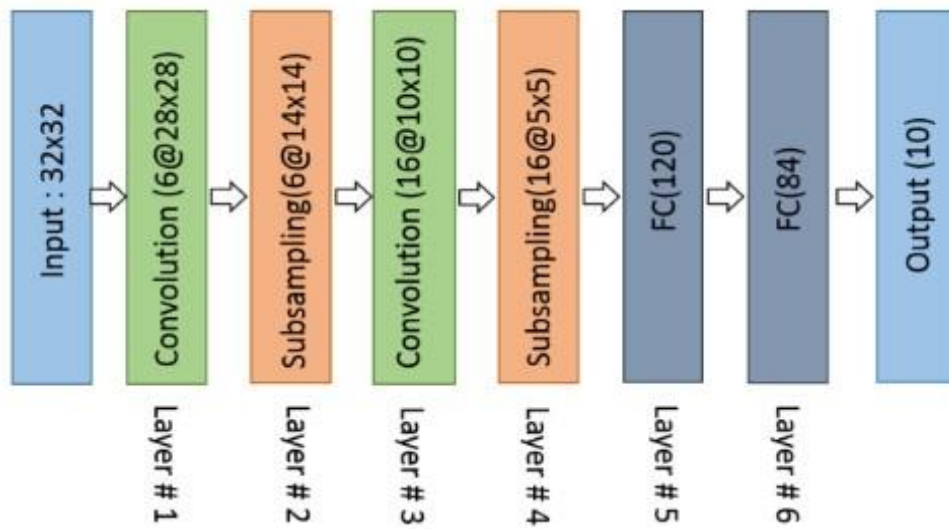


FIG :4.1 Architecture of Alex Net **Pooling layers:**

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

LENET

As a representative of the early convolutional neural network, LeNet possesses the basic units of convolutional neural network, such as convolutional layer, pooling layer and full connection layer, laying a foundation for the future development of convolutional neural network. As shown in the figure (input image data with 32*32 pixels) LeNet-5 consists of seven layers. In addition to input, every other layer can train parameters.

Architecture of LeNet:

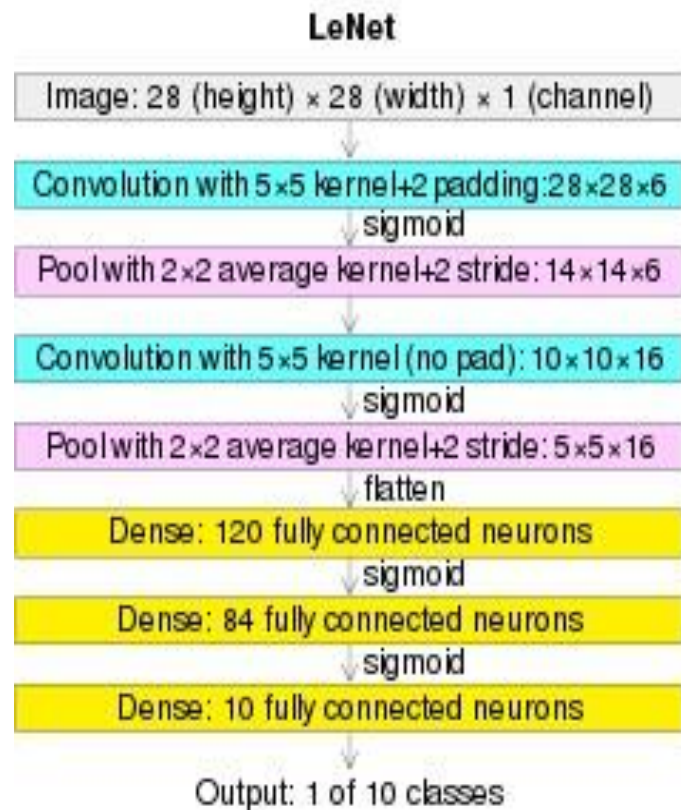


FIG:4.2. ARCHITECTURE OF LENET

Pooling layers:

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

Dense or fully connected layers:

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

LIST OF MODULES

1. Data Analysis
2. Manual Architecture
3. LeNet Architecture
4. U-Net Architecture
5. Deployment

MODULE DESCRIPTION

IMPORT THE GIVEN IMAGE FROM DATASET:

We have to import our data set using keras preprocessing image data generator function also we create size, rescale, range, zoom range, horizontal flip. Then we import our image dataset from folder through the data generator function. Here we set train, test, and validation also we set target size, batch size and class-mode from this function we have to train using our own created network by adding layers of CNN.

Manual Architecture:

Creating a manual architecture for image segmentation typically involves designing a neural network or algorithm that can process an input image and produce pixel-level segmentation masks or regions of interest. Here's a simplified manual architecture for image segmentation

1. Input Image: The architecture takes an input image as its primary input. This can be a grayscale or colour image depending on your application.
2. Preprocessing: Preprocess the input image to enhance features and reduce noise. Common preprocessing steps include resizing, normalization, and data augmentation.
3. Convolutional Neural Network (CNN): Use a convolutional neural network as the backbone of your segmentation architecture. CNNs are highly effective at capturing local patterns and spatial features in images.

- You can use a pre-trained CNN architecture like VGG, Resnet, or U-Net as a starting point, or design a custom architecture.

4. Encoder-Decoder Architecture: Many segmentation architectures follow an encoder-decoder structure:

- Encoder: The encoder part of the network extracts features from the input image through a series of convolutional layers. These layers reduce spatial dimensions while increasing the number of feature maps.

- Decoder: The decoder part of the network up samples the feature maps to the original image size while reducing the number of channels. This process helps generate the segmentation mask.

5. Skip Connections: To improve segmentation accuracy, consider adding skip connections that concatenate feature maps from the encoder to the corresponding layers in the decoder. This allows the network to capture both high-level and low-level features.

6. Convolutional Transpose (Deconvolution) Layers: Use convolutional transpose layers (sometimes called deconvolution layers) to up sample feature maps in the decoder. These layers expand the spatial resolution of the feature maps.

7. Activation Function: Apply an activation function, typically a SoftMax or sigmoid, to the final layer of the decoder to produce the segmentation mask.

For binary segmentation, sigmoid is often used; for multi-class segmentation, SoftMax is common.

8. Loss Function: Define an appropriate loss function to measure the difference between the predicted segmentation mask and the ground truth mask. Common loss functions for segmentation include binary cross-entropy and categorical cross-entropy.

9. Optimization Algorithm: Use an optimization algorithm like stochastic gradient descent (SGD), Adam, or RMSprop to update the network's weights and minimize the loss function.

10. Training Data: Train the network using a dataset of annotated images. The dataset should include input images and corresponding segmentation masks.

11. Post-processing: Apply post-processing techniques if needed, such as morphological operations (erosion, dilation) to refine the segmentation mask.

12. Inference: During inference, feed an unseen image through the trained network to obtain the segmentation mask.

13. Evaluation: Evaluate the segmentation accuracy using metrics like Intersection over Union (IoU), Dice coefficient, or pixel accuracy.

Encoder-Decoder Structure:

U-Net follows an encoder-decoder structure. It consists of two main parts: the encoder and the decoder.

The encoder captures features from the input image at multiple scales by using convolutional and pooling layers. It gradually reduces the spatial dimensions while increasing the number of feature maps.

The decoder then takes these features and up samples them to the original image size while reducing the number of feature maps. This helps generate a detailed segmentation mask.

Contracting and Expansive Paths:

The encoder path is often referred to as the contracting path because it reduces spatial dimensions. The decoder path is called the expansive path because it increases the spatial dimensions. Skip connections connect the contracting and expansive paths, facilitating the flow of information between them.

Final Layer:

The final layer of the U-Net architecture typically consists of a convolutional layer with a SoftMax activation function for multi-class segmentation or a sigmoid activation function for binary segmentation. The output of this layer is the segmentation mask, where each pixel is classified into the desired classes (e.g., tumour or background).

Training Data:

To train a U-Net model for Eye fundus diseases cancer segmentation, you need a dataset of annotated Eye fundus diseases images. The dataset should

include input mammograms or other relevant images and corresponding pixel-level segmentation masks indicating the regions of interest (e.g., tumours).

Inference:

During inference, you feed an unseen Eye fundus diseases image through the trained U-Net model to obtain the segmentation mask, which highlights the areas of interest, such as tumours or lesions.

Post-processing:

Post-processing steps, such as morphological operations (e.g., erosion, dilation) or connected component analysis, can be applied to refine the segmentation mask and remove any artifacts.

U-Net has been widely adopted in medical image segmentation due to its ability to capture fine details and its effectiveness in handling limited training data. Researchers and practitioners often customize U-Net architectures by adjusting the number of layers, filter sizes, and skip connections to suit the specific requirements of their Eye fundus diseases cancer segmentation tasks. Additionally, data augmentation techniques are commonly used to increase the diversity of training data and improve model generalization.

4.6 DEPLOYMENT:

Deploying the model in Django Framework and predicting output

In this module the trained deep learning model is converted into hierarchical data format file (.h5 file) which is then deployed in our Django framework for providing better user interface and predicting the output.

4.7 Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a

thriving and active community, great documentation, and many options for free and paid-for support

4.8 HTML

HTML stands for Hypertext Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

4.9 CSS

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colors, layout, and fonts, thus making our web pages presentable to the users.

CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. Now let's try to break the acronym:

- Style: Adding designs/Styling our HTML tags
- Sheets: Writing our style in different documents

CSS Syntax

```
Selector {  
  Property 1: value;  
  Property 2: value;  
  Property 3: value;
```

```
}
```

For example:

```
h1
```

```
{
```

```
Color: red;
```

```
Text-align: centre;
```

```
}
```

```
#unique
```

```
{
```

```
color: green;
```

CHAPTER 5

RESULTS & DISCUSSION

5.1 WEBPAGE CREATION:

HTML CODE:

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DEPLOY</title>
<style>
  body{
    background:url("/static/images/D5.jpg");
background-repeat: no-repeat;
background-attachment: fixed;
background-size: 100% 100%;
  }
  .container{
width: 600px;
```



```
<img src= {{image_uri}} height="200" width="200">
</center>
{% endif %}
</body>
</html>
```

It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages.

Meta Tags: These provide metadata about the HTML document. For example, specifying the character set, viewport settings, etc.

Title: Sets the title of the HTML page, which appears in the browser tab.

Style: Contains CSS styles for styling the HTML elements. In this case, it sets the background image, container styles, form styles, etc.

Body: This section contains the content of the HTML page.

Header: Contains the title of the web application and a link to the home page.

Container: A div that holds the form for uploading images.

Form: Allows users to upload images. It has a CSRF token for security, and it sends data to the specified URL using the POST method.

Conditional Display: Checks if an image URI is available. If so, it displays the result section, including the classification output and the original and segmented images.

WEB PAGE:



FIG 5.1 WEB PAGE FOR REGISTRATION

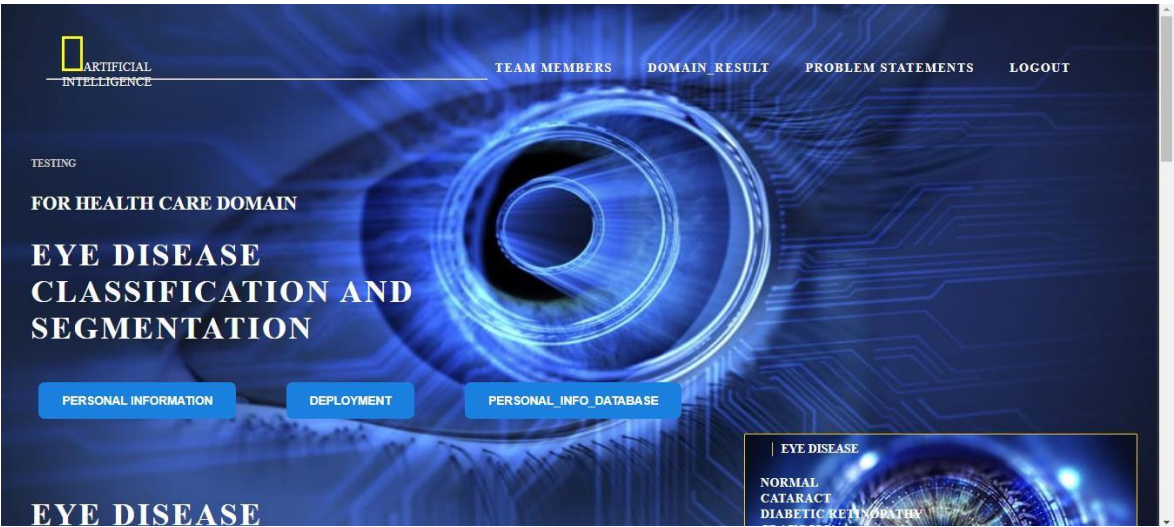


FIG 5.2 WEB PAGE OF EYE DISEASE PREDICTION

5.2 IMPORTING DATA SET:

PREPROCESSING THE DATASET:

Preprocessing and training the model (CNN): The dataset is pre-processed such as Image reshaping, resizing and conversion to an array form. Similar processing is also done on the test image. A dataset consisting of about 4 different types Cataract, Diabetic retinopathy, Glaucoma, Normal out of which any image can be used as a test image for the software.

TRAINING AND TESTING MODEL:

TRAINING: The train dataset is used to train the model (CNN) so that it can identify the test image and the disease it has CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify the different types of eye disease like Cataract, Diabetic retinopathy, Glaucoma, Normal Classification image contained in the dataset. After successful training and preprocessing, comparison of the test image and trained model takes place to predict.

TESTING: Data Preparation: Preprocess the testing dataset in a similar way to how you pre-processed the training dataset. This may involve resizing images, normalizing pixel values, and any other preprocessing steps necessary for your model.

Model Loading: Load the trained CNN model that want to evaluate using the testing dataset.

Testing Dataset Loading: Load the testing dataset, which includes input images

Evaluation: Use the testing dataset to evaluate the performance of your model.

This typically involves feeding the input images through the model and comparing the predicted outputs with the ground truth labels

5.3 DATA PREPARATION:

Resize images to the input size required by your CNN model.

Normalize pixel values (typically between 0 and 1).

Convert images to the appropriate data format (e.g., NumPy arrays).

MODEL LOADING:

```
from tensorflow.keras.models import load_model
```

```
# Load the trained model
```

```
model = load_model('path_to_your_trained_model.h5')
```

Loading a trained model using TensorFlow and Keras is straightforward with the function (LOAD MODEL). After loading the model, can provide additional information or an explanation of what the loaded model will be used for or how it will be further utilized in application.

TESTING DATASET LOADING:

```
import numpy as np
```

```
# Load testing data
```

```
X_test = ... # Load input images y_test = ... # Load
```

corresponding labels (if available)

EVALUATION:

```
# Evaluate the model on the testing dataset
```

```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print(f'Test Loss: {loss}')
```

```
print(f'Test Accuracy: {accuracy}')
```

evaluate the model's performance by making predictions and comparing them with the actual labels or by other means such as visual inspection or domain-specific metrics.

DATASET:

On Kaggle, datasets containing eye fundus images may include:

Diabetic Retinopathy Datasets: These datasets consist of fundus images of patients with diabetic retinopathy, a common complication of diabetes that can lead to vision loss if left untreated. Researchers and machine learning practitioners use these datasets to develop algorithms for automatic detection and grading of diabetic retinopathy from fundus images.

Glaucoma Datasets: Glaucoma is a group of eye conditions that damage the optic nerve, leading to vision loss and blindness if untreated. Fundus images may be used to study optic disc morphology, assess retinal nerve fiber layer thickness, and develop algorithms for early detection of glaucoma

1)CATARACT TYPE DATA SET:

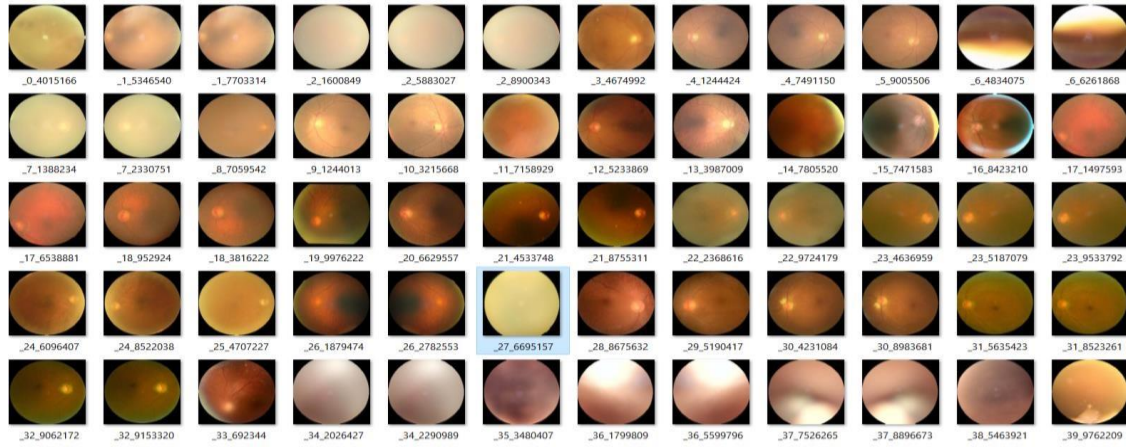


FIG 5.3 CATARACT DATA SET 1

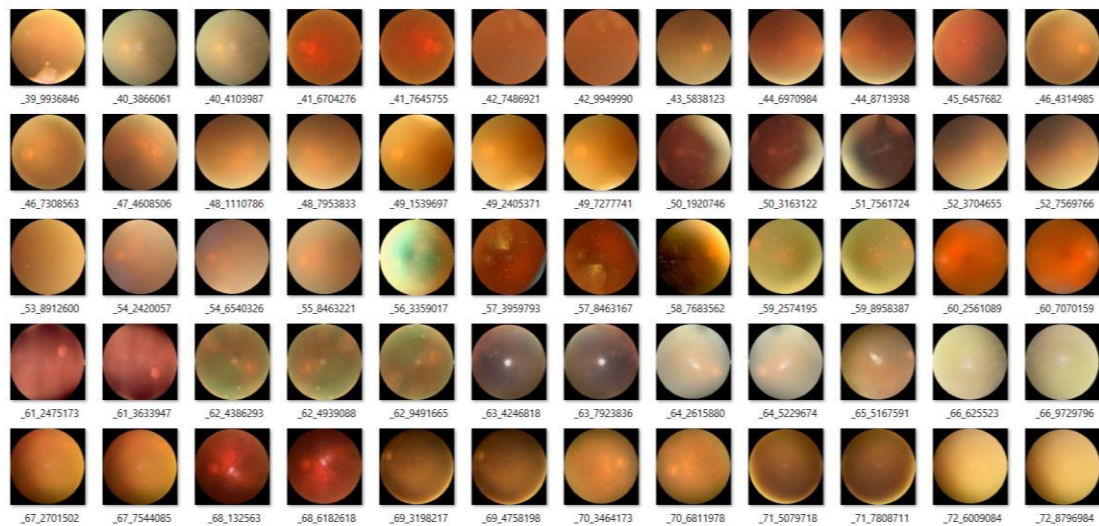


FIG 5.4 CATARACT DATASET 2

SYMPTOMS OF CATARACT EYE DISEASE:

In addition to blurred vision, cataracts can cause other symptoms such as sensitivity to light, difficulty seeing at night, seeing halos around lights, double vision in one eye, and fading or yellowing of colours.

2)DIABETIC RETINOPATHY DATA SET:

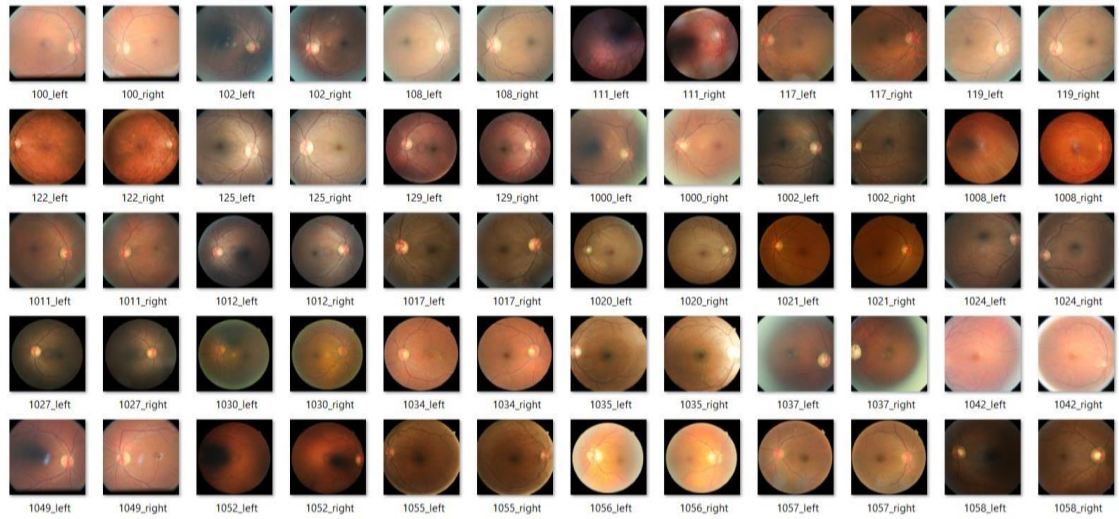


FIG 5.5 DIABETIC RETINOPATHY DATASET 1

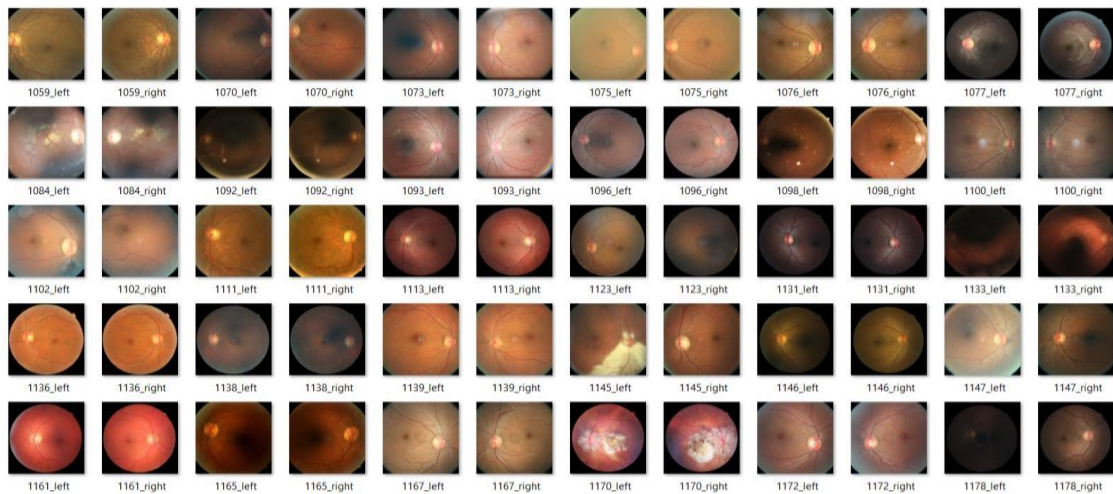


FIG 5.6 DIABETIC RETINOPATHY DATASET 2

SYMPTOMS OF DIABETIC RETINOPATHY:

In the early stages, diabetic retinopathy may not cause any noticeable symptoms. as the condition progresses, symptoms may include blurry or fluctuating vision, floaters (spots or dark strings floating in your vision), impaired colour vision, dark or empty areas in your vision, and difficulty seeing at night.

3) GLAUCOMA DATASET:

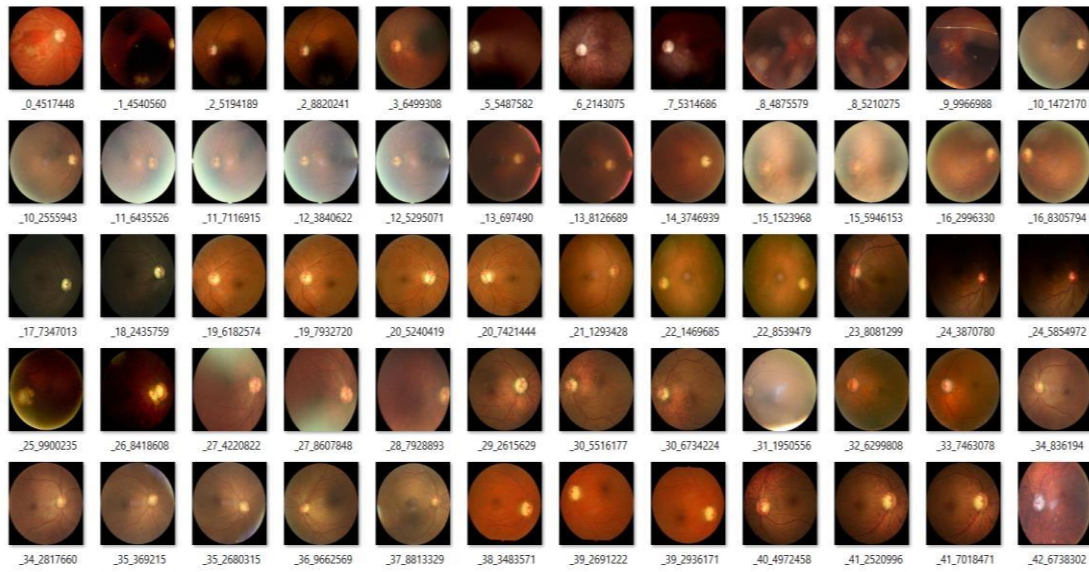


FIG 5.7 GLAUCOMA DATASET 1

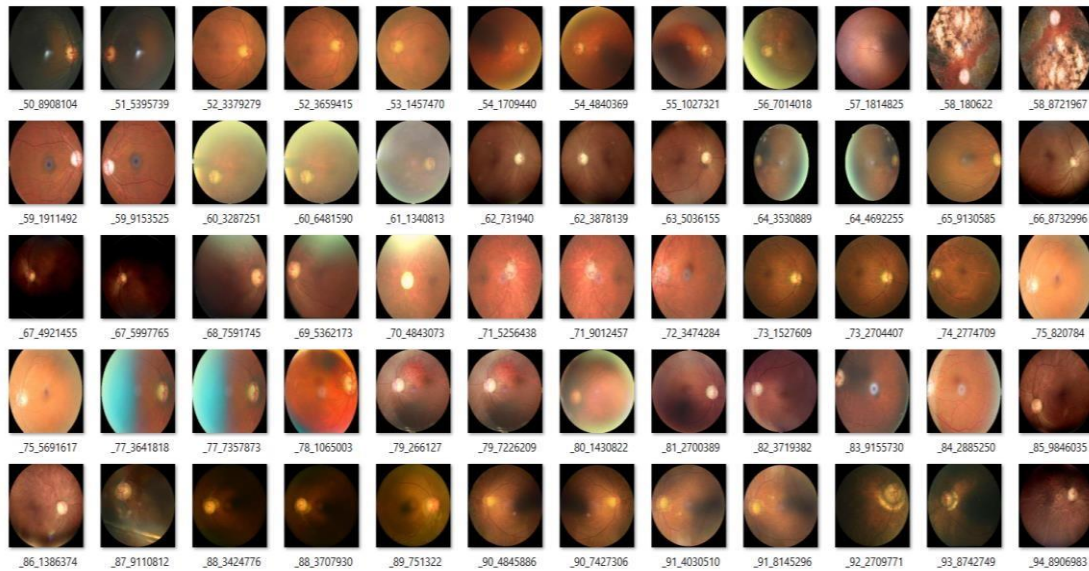


FIG 5.8 GLAUCOMA DATASET 2

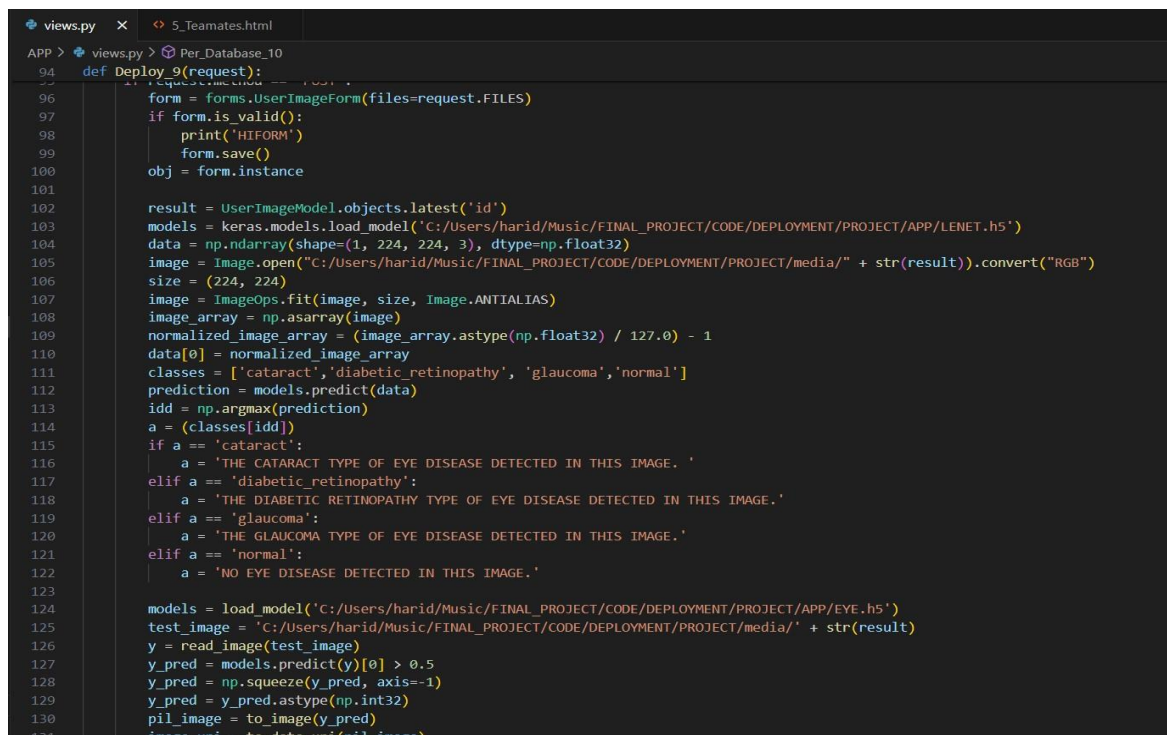
SYMPTOMS OF GLAUCOMA :

In the early stages, glaucoma may not cause noticeable symptoms. However, as the condition progresses, symptoms may include gradual loss of peripheral vision, tunnel vision, blurred vision, halos around lights, severe eye pain, headache, nausea, and vomiting (in acute angle-closure glaucoma).

5.4 PREDICTION OF DISEASE:

USING CNN MODEL:

- 1)Preprocess the Input Images: Prepare the input images in the same way they were prepared during training. This typically involves resizing, normalization, and any other necessary preprocessing steps.
- 2)Load the Trained CNN Model: Load the pre-trained CNN model that was trained on eye disease data.
- 3)Make Predictions: Use the loaded model to make predictions on the input images.
- 4)Interpret Predictions: Interpret the model's predictions to understand the predicted eye disease for each input image.



```
views.py x 5_Teamates.html
APP > views.py > Per_Database_10
94 def Deploy_9(request):
95     form = forms.UserImageForm(files=request.FILES)
96     if form.is_valid():
97         print("HIFORM")
98         form.save()
99     obj = form.instance
100
101     result = UserImageModel.objects.latest('id')
102     models = keras.models.load_model('C:/Users/harid/Music/FINAL_PROJECT/CODE/DEPLOYMENT/PROJECT/APP/LENET.h5')
103     data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
104     image = Image.open("C:/Users/harid/Music/FINAL_PROJECT/CODE/DEPLOYMENT/PROJECT/media/" + str(result)).convert("RGB")
105     size = (224, 224)
106     image = ImageOps.fit(image, size, Image.ANTIALIAS)
107     image_array = np.asarray(image)
108     normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1
109     data[0] = normalized_image_array
110     classes = ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
111     prediction = models.predict(data)
112     idd = np.argmax(prediction)
113     a = (classes[idd])
114     if a == 'cataract':
115         a = 'THE CATARACT TYPE OF EYE DISEASE DETECTED IN THIS IMAGE.'
116     elif a == 'diabetic_retinopathy':
117         a = 'THE DIABETIC RETINOPATHY TYPE OF EYE DISEASE DETECTED IN THIS IMAGE.'
118     elif a == 'glaucoma':
119         a = 'THE GLAUCOMA TYPE OF EYE DISEASE DETECTED IN THIS IMAGE.'
120     elif a == 'normal':
121         a = 'NO EYE DISEASE DETECTED IN THIS IMAGE.'
122
123     models = load_model('C:/Users/harid/Music/FINAL_PROJECT/CODE/DEPLOYMENT/PROJECT/APP/EYE.h5')
124     test_image = 'C:/Users/harid/Music/FINAL_PROJECT/CODE/DEPLOYMENT/PROJECT/media/' + str(result)
125     y = read_image(test_image)
126     y_pred = models.predict(y)[0] > 0.5
127     y_pred = np.squeeze(y_pred, axis=-1)
128     y_pred = y_pred.astype(np.int32)
129     pil_image = to_image(y_pred)
130     image_url = to_data_url(pil_image)
```

Form Handling: expect an image file to be uploaded via a form (UserImageForm). If the form is valid, save the uploaded image.

Model Loading and Prediction: load trained CNN models (LENET.h5 and EYE.h5) using Keras. Then, preprocess the uploaded image, make predictions using the

loaded models, and interpret the predictions to determine the type of eye disease detected in the image.

Segmentation: performing image segmentation using another model (EYE.h5). load this model, read the uploaded image, make predictions for segmentation, and convert the segmentation result into a data URI for displaying the segmented image in the HTML template.

Rendering: render result along with the form and segmented image URI in the HTML template (9_Deploy.html).

```
(c) Microsoft Corporation. All rights reserved.

(base) C:\Users\harid\Music\FINAL_PROJECT\CODE\DEPLOYMENT\PROJECT>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

2024-04-18 12:25:41.861430: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
C:\Users\harid\anaconda3\lib\site-packages\scipy\_init_.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")
WARNING:tensorflow:From C:\Users\harid\anaconda3\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

System check identified no issues (0 silenced).
April 18, 2024 - 12:25:51
Django version 4.2.11, using settings 'PROJECT.settings'
Starting development server at http://127.0.0.1:8000/
```

System Information: The output starts with system information, indicating that using Microsoft Windows 10.0.22621.3447.

Python Environment: running your Django project in a Python environment. The (base) prefix in command prompt suggests that might be using Anaconda or a similar environment manager.

Django Server Start: The command python manage.py run server is used to start the Django development server. It listens for file changes and reloads the server accordingly using the Statehooder.

5.5 OUTPUT:

CATARACT EYE DISEASE:



FIG 5.9 CATARACT DISEASE

DIABETIC RETINOPATHY:

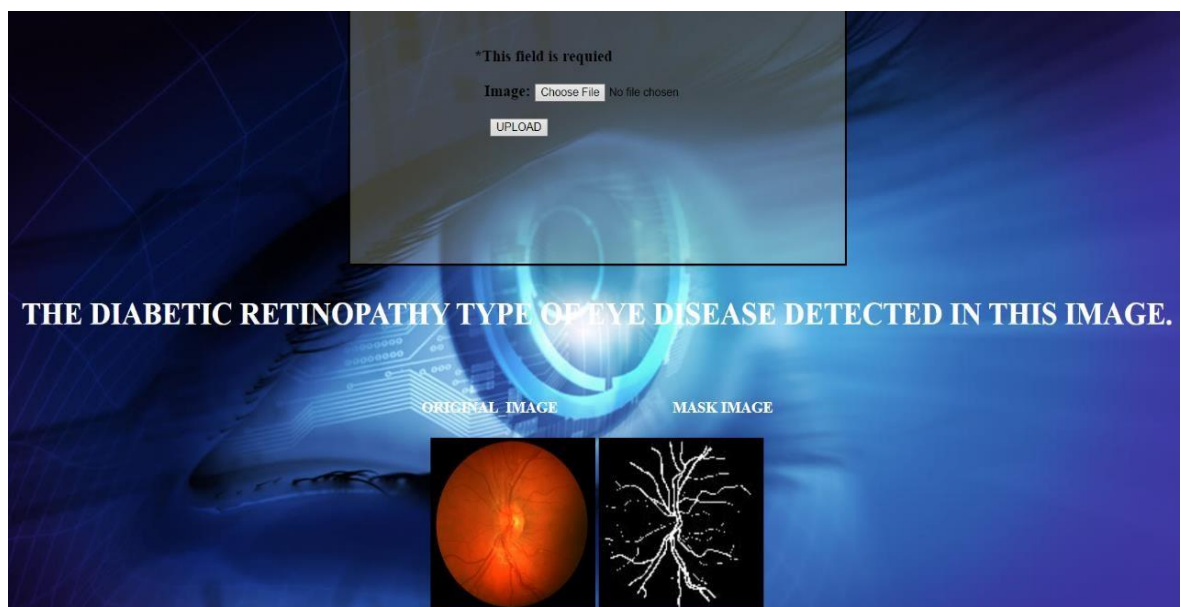


FIG 5.10 DIABETIC RETINOPATHY

GLAUCOMA:

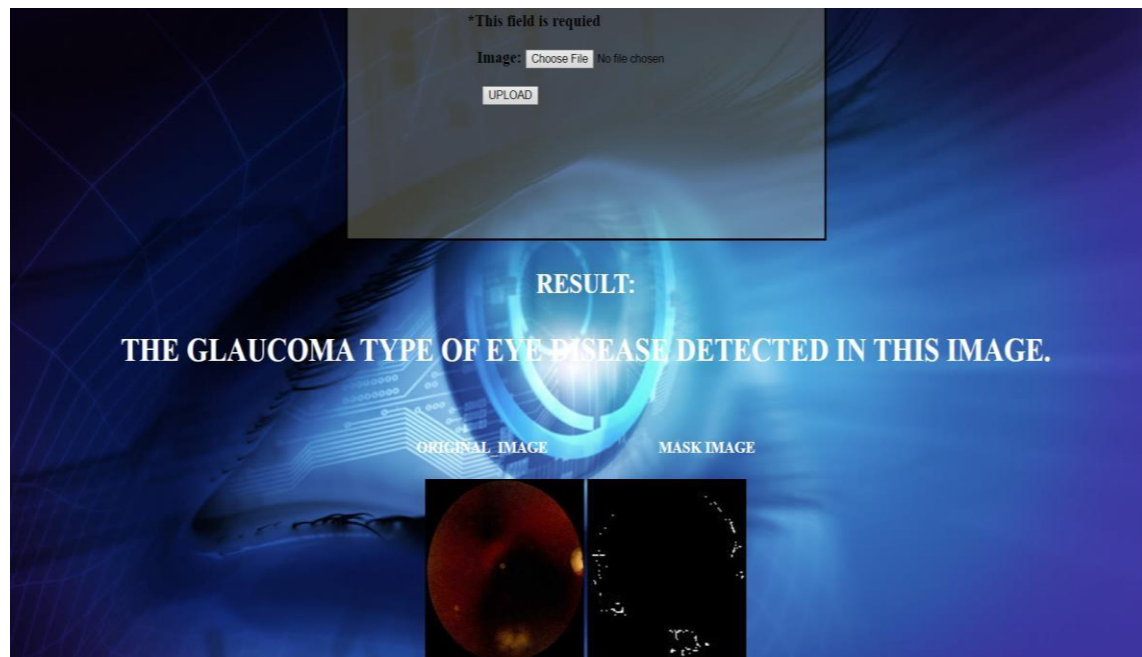


FIG 5.11 GLAUCOMA DISEASE

NORMAL:

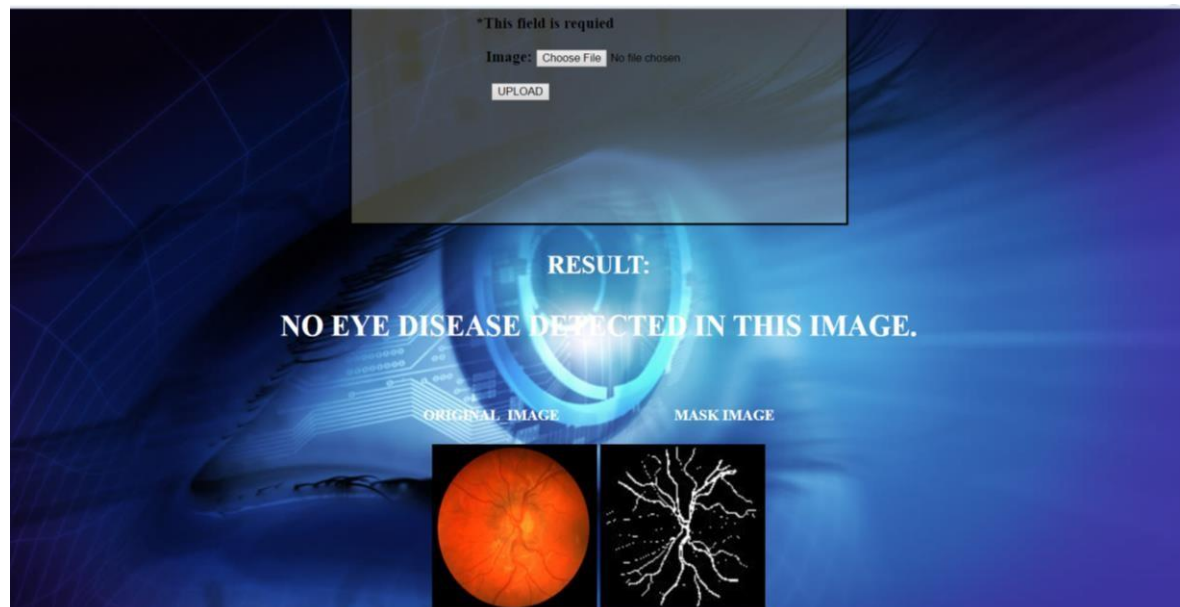


FIG 5.13 NO DISEASE IMG

CHAPTER 6

CONCLUSION

The application of artificial intelligence techniques in the diagnosis of eye fundus diseases through classification and segmentation has shown promising results in enhancing the efficiency and accuracy of medical assessments. The use of advanced algorithms and machine learning models has facilitated the automatic identification and categorization of various eye fundus pathologies, providing valuable support to healthcare professionals. These AI-driven approaches have the potential to streamline the diagnostic process, enabling early detection and intervention, thereby improving patient outcomes. However, ongoing research and validation studies are crucial to ensuring the robustness and reliability of these techniques in real-world clinical settings, and collaborative efforts between the medical and AI communities are essential for the continued development and implementation of effective tools for eye fundus disease diagnosis.

REFERENCES

- 1) Abdel-Hamid L. (2022) TWEEC: Computer-aided glaucoma diagnosis from retinal images using deep learning techniques. *Int. J. Imaging Syst. Technol.*; (32), 387-401. Doi: [10.1002/ima.22621](https://doi.org/10.1002/ima.22621)
- 2) Ahmed I.A., Senan E.M., Rassem T.H., Ali M.A., Shatnawi H.S.A., Alwazer S.M., Alshahrani M (2022). Eye Tracking-Based Diagnosis and Early Detection of Autism Spectrum Disorder Using Machine Learning and Deep Learning Techniques. *Electronics*; (11), 525-530. Doi: [10.3390/electronics11040530](https://doi.org/10.3390/electronics11040530)
- 3) Ahlam Shamsan, Ebrahim Mohammed Senan, and Hamzeh Salameh Ahmad Shatnawi (2023), Automatic Classification of Colour Fundus Images for Prediction Eye Disease Types Based on Hybrid Features, *13*(10), 1706 -1715, Doi: [10.3390/diagnostics13101706](https://doi.org/10.3390/diagnostics13101706)
- 4) Atteia G., Abdel Samee N., El-Kenway E.-S.M., Ibrahim (2022). CNN Hyperparameter Optimization for Diabetic Maculopathy Diagnosis in Optical Coherence Tomography and Fundus Retinography. *Mathematics*. (10), 3274-3295 Doi: [10.3390/math10183274](https://doi.org/10.3390/math10183274)
- 5) Bhardwaj C., Jain S., Sood M. Hierarchical (2021) severity grade classification of non-proliferative diabetic retinopathy. *J. Ambient. Intell. Humaniz. Comput.*; (12), 2649–2670. Doi: [10.1007/s12652-020-02426-9](https://doi.org/10.1007/s12652-020-02426-9).
- 6) Chang J., Lee J., Ha A., Han Y.S., Bak E., Choi S., Park S.M (2021). Explaining the rationale of deep learning glaucoma decisions with adversarial examples. *Ophthalmology*. (128), 78–88. Doi: [10.1016/j.ophtha.2020.06.036](https://doi.org/10.1016/j.ophtha.2020.06.036).
- 7) Elloumi Y (2021). *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* Volume 12721. Springer; Cham, Switzerland. Mobile Aided System of Deep-

8)Farooq M.S., Arooj A., Alroobaea R., Baqasah A.M., Jabarulla M.Y., Singh D., Sardar R (2022). Untangling computer-aided diagnostic system for screening diabetic retinopathy based on deep learning techniques. *Sensors*. (22)1803.

Doi: [10.3390/s22051803](https://doi.org/10.3390/s22051803)

9) Gayathri S., Gopi V.P., Palanisamy P (2020). A lightweight CNN for Diabetic Retinopathy classification from fundus images. *Biomed. Signal Process. Control.*; (62) 102-115, Doi: [10.1016/J.BSPC.2020.102115](https://doi.org/10.1016/J.BSPC.2020.102115).

10)Hu K., Zhang Z., Niu X., Zhang Y., Cao C., Xiao F., Gao X (2018). Retinal vessel segmentation of color fundus images using multiscale convolutional neural network with an improved cross-entropy loss function. *Neurocomputing*; (309) 179–191. Doi: [10.1016/j.neucom.2018.05.011](https://doi.org/10.1016/j.neucom.2018.05.011).

11)I. Rubasinghe, D. Meedeniya(2019). Ultrasound nerve segmentation using deep probabilistic programming. *Journal of ICT Research and Applications*, vol. 13, no. 3, 241–256, DOI: <https://doi.org/10.5614/itbj.ict.res.appl.2019.13.3.5>.

12)Junayed M.S., Islam M.B., Sadeghzadeh A., Rahman S (2021). Cataract Net: An automated cataract detection system using deep learning for fundus images. *IEEE.*; 9: 128799–128808. Doi: [10.1109/ACCESS.2021.3112938](https://doi.org/10.1109/ACCESS.2021.3112938).

13) J. Kim, L. Tran, E. Y. Chew, S. Antani (2019). Optic disc and cup segmentation for glaucoma characterization using deep learning., *IEEE, Córdoba, Spain*, 489–494, DOI: <https://doi.org/10.1109/cbms.2019.00100>.

14)Keenan T.D., Chen Q., Agrón E., Tham Y.C., Goh J.H.L., Lei X., Zeleny A. (2022;) DeepLensNet: Deep learning automated diagnosis and quantitative classification of cataract type and severity. *Ophthalmology*.571–584.

Doi:[10.1016/j.ophtha.2021.12.017](https://doi.org/10.1016/j.ophtha.2021.12.017)

- 15)Kumar K.S., Singh N.P. (2023) Retinal disease prediction through blood vessel segmentation and classification using ensemble-based deep learning approaches. *Neural Comput.*1–17. Doi: [10.1007/s00521-023-08402-6](https://doi.org/10.1007/s00521-023-08402-6)
- 16) Li T., Bo W., Hu C., Kang H., Liu H., Wang K., Fu H(2021). Applications of deep learning in fundus images: A review. *Med. Image Anal.*10-15. Doi: [10.1016/j.media.2021.101971](https://doi.org/10.1016/j.media.2021.101971).
- 17)Li J.Q., Welchowski T., Schmid M., Letow J., Wolpers C., Pascual-Camps I., Finger R.P (2019). Prevalence, incidence and future projection of diabetic eye disease in Europe: A systematic review and meta-analysis. *Eur. J. Epidemiol.*; 35:11–23, Doi: [10.1007/s10654-019-00560-z](https://doi.org/10.1007/s10654-019-00560-z).
- 18) Mahmoud M.H., Alamery S., Fouad H., Altinawi A., Youssef A.E(2021). An automatic detection system of diabetic retinopathy using a hybrid inductive machine learning algorithm. *Pers. Ubiquitous Comput.*1–15. Doi: [10.1007/s00779-020-01519-8](https://doi.org/10.1007/s00779-020-01519-8)
- 19)Nawaz M., Nazir T., Javed A., Tariq U., Yong H.S., Khan M.A., Cha J. An (2022) efficient deep learning approach to automatic glaucoma detection using optic disc and optic cup localization. *Sensors.* (22)434. Doi: [10.3390/s22020434](https://doi.org/10.3390/s22020434).
- 20)Ouda O., AbdelMaksoud E., Abd El-Aziz A.A., Elmogy M(2022). Multiple Ocular Disease Diagnosis Using Fundus Images Based on Multi-Label Deep LearningClassification. *Electronics.*;11:1966. Doi: [10.3390/electronics11131966](https://doi.org/10.3390/electronics11131966)
- 21)P. Elangovan, D. Vijayalakshmi, M. K. Nath (2021). Detection of glaucoma from fundus image using pre-trained Densenet201 model. *Indian Journal of Radio & Space Physics*, vol. 50, no. 1,33–39.
- 22) ´sPlayout C., Duval R., Cheriet F. A Novel(2019) Weakly Supervised Multitask Architecture for Retinal Lesions Segmentation on Fundus Images. *IEEE Trans. Med. Imaging.*2434–2444. Doi: [10.1109/TMI.2019.2906319](https://doi.org/10.1109/TMI.2019.2906319)

23)R. Mahum, S. Ur Rehman, O. D. Okon, A. Alabrah, T. Meraj, H. T. Rauf. A novel (2022) hybrid approach based on deep CNN to detect glaucoma using fundus imaging. Electronics, vol. 11, no. 1, Article number 26, DOI: <https://doi.org/10.3390/electronics11010026>.

24)Ryu G., Lee K., Park D., Park S.H., Sagong M (2021). A deep learning model for identifying diabetic retinopathy using optical coherence tomography angiography. Sci. Rep.11:23024. Doi: [10.1038/s41598-021-02479-6](https://doi.org/10.1038/s41598-021-02479-6).

25)Saranya P., Pranati R., Patro S.S(2023). Detection and classification of red lesions from retinal images for diabetic retinopathy detection using deep learning models. Multimed. Tools Appl.1–21. Doi: [10.1007/s11042-023-15045-1](https://doi.org/10.1007/s11042-023-15045-1)

26)Thanki R(2023) . A deep neural network and machine learning approach for retinal fundus image classification. Health. Anal; 105-115.
Doi: [10.1016/j.health.2023.100140](https://doi.org/10.1016/j.health.2023.100140).

27) Y. Prajna, M. K. Nath (2022). Efficient blood vessel segmentation from color fundus image using deep neural network. Journal of Intelligent & Fuzzy Systems, vol. 42, no. 4, pp. 3477–3489, DOI: <https://doi.org/10.3233/jifs-211479>.

APPENDIX

Module 1:

Classification:

MANUAL NET ARCHITECTURE

```
import TensorFlow import
tensorflow as tf print (tf.
__version__)
```

```
import os import glob from
PIL import Image import
matplotlib. pyplot as plt
```

```
from tensorflow. keras. Models import Sequential
```

```
from tensorflow. keras. Layers import Convolution2D from
tensorflow. keras. Layers import MaxPooling2D from tensorflow.
keras. Layers import Flatten from tensorflow. keras. Layers
import Dense from tensorflow. keras. Layers import Dropout from
tensorflow. keras. Layers import Activation cataract =
'Dataset/cataract' diabetic_retinopathy =
'Dataset/diabetic_retinopathy' glaucoma = 'Dataset/glaucoma'
normal = 'Dataset/normal' def plot_images (item_dir, n=6):
all_item_dir = os. listdir(item_dir) item_files = [os. path. join
(item_dir, file) for file in all_item_dir][:n]
```

```
plt. Figure(fig size=(80, 40))
for idx, img_path in enumerate (item_files):
plt. subplot (3, n, idx+1) img =
plt. imread(img_path) plt.
imshow (img, cmap='gray') plt.
Axis('off')
```

```
plt. tight_layout ()
```

```

def image_details_print (data, path):
print ('===== Images in: ', path) for
key, values in data. Items ():
print (key,': \t', values)

```

```

def images_details(path):
files= [f for f in glob. Glob (path + "**/*. **", recursive=True)]
data= {} data['Images_count'] =len(files) data['Min_width']
=10**100 data['Max_width'] =0 data['Min_height'] =10**100
data['Max_height'] =0

```

```

for f in files:
img=Image. Open(f) width, height=img. Size
data['Min_width'] =min (width, data['Min_width'])
data['Max_width'] =max (width, data['Max_width'])
data['Min_height'] =min (height, data['Min_height'])
data['Max_height'] =max (height, data['Max_height'])

```

```

image_details_print(data, path)
print (""")
print ("TRAINING DATA FOR CATARACT:")
print (""") images_details(cataract) print
(""") plot_images (cataract, 10) print (""")
print ("TRAINING DATA FOR DIABETIC RETINOPATHY:")
print (""") images_details(diabetic_retinopathy) print (""")
plot_images (diabetic_retinopathy, 10) print (""")
print ("TRAINING DATA FOR GLAUCOMA:") print
(""")
images_details(glaucoma)
print (""") plot_images
(glaucoma, 10) print (""")

```

```

print ("TRAINING DATA FOR SEBORRHEIC NORMAL:")
print ("" ) images_details(normal) print ("" ) plot_images
(normal, 10)
# Define the path to your dataset directory dataset
Dir = 'Dataset'

# Set the batch size and number of classes
batch_size = 64 num_classes = 4
# Create an ImageDataGenerator for data augmentation and preprocessing

from tensorflow. keras.      preprocessing.      Image      import
ImageDataGenerator

datagen = ImageDataGenerator (
rescale=1. /255, shear_range=0.2,
zoom_range=0.2,
rotation_range=15,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
validation_split=0.2)

# Load and preprocess the dataset using the
ImageDataGenerator      train_generator = datagen.
flow_from_directory ( dataset_dir, target_size= (224, 224),
batch_size=batch_size, class_mode='categorical', subset='training')
validation_generator = datagen.flow_from_directory ( dataset_dir,
target_size= (224, 224), batch_size=batch_size,
class_mode='categorical', subset='validation')

MODEL=Sequential ()
MODEL.add(Convolution2D(32,(3,3),input_shape=(224,224,3),activation='relu'))
MODEL.add(MaxPooling2D(pool_size=(2,2)))

```

```

MODEL.add(Flatten())
MODEL.add(Dense(4, activation='softmax'))
MODEL.compile(optimizer='rmsprop',loss='categorical_crossentropy'
,metrics=['accuracy']) epochs = 10
history = MODEL.fit( train_generator,
validation_data=validation_generator,
epochs=epochs # Add the callbacks here
)
import matplotlib.pyplot as plt

```

```

def graph():
#Plot training & validation accuracy
values plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy') plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

```

```

graph() import
matplotlib.pyplot as plt

```

```

def graph():
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss') plt.ylabel('Loss')
plt.xlabel('Epoch') plt.legend(['Train',
'Test'], loc='upper left') plt.show() graph()

```

Module 2:

```

# LENET ARCHITECTURE
import tensorflow import

```

```

tensorflow as tf
print(tf.__version__)

from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten from
tensorflow.keras.layers import Dense from
tensorflow.keras.layers import Dropout from
tensorflow.keras.layers import Activation
# Create an ImageDataGenerator for data augmentation and preprocessing

from tensorflow.keras.preprocessing.image import
ImageDataGenerator

datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    validation_split=0.2)
# Define the path to your dataset directory dataset_dir
= 'Dataset'

# Set the batch size and number of classes
batch_size = 64 num_classes
= 9

# Load and preprocess the dataset using the ImageDataGenerator

train_generator = datagen.flow_from_directory(
    dataset_dir, target_size=(224, 224),
    batch_size=batch_size, class_mode='categorical',

```

```

subset='training') validation_generator =
datagen.flow_from_directory( dataset_dir,
target_size=(224, 224), batch_size=batch_size,
class_mode='categorical', subset='validation')

from tensorflow.keras.models import Sequential

MODEL=Sequential()
MODEL.add(Convolution2D(filters=32,      kernel_size=(3,3), strides=(3,3),
      input_shape=(224,224,3),      padding=('valid'), activation='relu'))
MODEL.add(MaxPooling2D(pool_size=(2,2),      strides=(2,2),
padding='valid'))
MODEL.add(Convolution2D(filters=128,      kernel_size=(3,3), strides=(3,3),
padding=('valid'), activation='relu'))
MODEL.add(MaxPooling2D(pool_size=(2,2),      strides=(2,2),
padding='valid'))
MODEL.add(Flatten())
MODEL.add(Dense(256, activation='relu'))
MODEL.add(Dense(4, activation='softmax'))

OPT  = tensorflow.keras.optimizers.Adam(0.001)

MODEL.compile(optimizer=OPT,loss='categorical_crossentropy',metr
ics=["accuracy", tensorflow.keras.metrics.Precision()])
MODEL.summary() model_path = "LENET.h5"

from keras.callbacks import ModelCheckpoint
M = ModelCheckpoint(model_path, monitor='accuracy', verbose=1,
save_best_only=True) epochs = 50 history = MODEL.fit(
train_generator,      validation_data=validation_generator,
epochs=epochs, callbacks=[M] # Add the callbacks here
)

import matplotlib.pyplot as plt import
numpy as np

```

```

plt.figure(figsize=(20, 8))
plt.plot(history.history['accuracy'])

for i in range(34):
    if i%5 == 0:
        plt.annotate(np.round(history.history['accuracy'][i]*100,2),xy=(i,history.history['accuracy'][i]))

plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()

plt.figure(figsize=(20, 8))
plt.plot(history.history['loss'])

for i in range(34):
    if i%5 == 0:
        plt.annotate(np.round(history.history['loss'][i]*100,2),xy=(i,history.history['loss'][i]))

plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()

```

