AI-DRIVEN CLASSIFICATION OF OCULAR DISEASES

PROJECT WORK II PHASE I REPORT

Submitted by

HARISH C
21ADR016

MOHAMED RIZWAN M
21ADR028

MONISHA K
21ADR029

in partial fulfilment of the requirements

for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE
AND DATA SCIENCE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

KONGUENGINEERINGCOLLEGE
(Autonomous)

PERUNDURAI, ERODE - 638060

OCTOBER 2024

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

KONGU ENGINEERING COLLEGE
(Autonomous)

PERUNDURAI, ERODE – 638060

OCTOBER 2024

BONAFIDE CERTIFICATE

This is to certify that the Project Report titled **AI-DRIVEN CLASSIFICATION OF OCULAR DISEASES** is the bonafide record of project work done by **HARISH C(21ADR016), MOHAMED RIZWAN M(21ADR028), MONISHA K (21ADR029)** in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Artificial Intelligence of Anna University, Chennai during the year 2024-2025.

SUPERVISOR **HEAD OF THE DEPARTMENT**

(Signature with seal)

Date:

Submitted for the end semester viva voce examination held on______________.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

KONGU ENGINEERING COLLEGE
(Autonomous)

PERUNDURAI, ERODE – 638 060

OCTOBER 2024

DECLARATION

We affirm that the Project Report titled **AI-DRIVEN CLASSIFICATION OF OCULAR DISEASES** being submitted in partial fulfillment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Date:

HARISH C
(21ADR016)

MOHAMED RIZWAN M
(21ADR028)

MONISHA K
(21ADR029)

I certify that the declaration made by the above candidate is true to the best of my knowledge.

Date: Name & Signature of the Supervisor with seal

ABSTRACT

Early identification is essential for preventing serious problems from eye disorders such as diabetic retinopathy, cataracts, and glaucoma, which pose serious hazards to eyesight. Historically, ophthalmologists have relied on manual evaluation of retinal pictures, which is time-consuming, subjective, and error-prone. The goal of this research is to overcome these constraints by creating an automated disease diagnostic system that uses deep learning, increasing efficiency and accuracy. In this project, we employed the InceptionResNetV2 transfer learning architecture to categorize retinal images into illness groups. Notably, we compared the performance of our model with and without data augmentation, achieving an accuracy of 96.5% with augmentation and 92.8% without augmentation. This comparison demonstrates the influence of data augmentation on the diagnostic accuracy of our model. By leveraging deep learning and transfer learning, our system demonstrates a significant improvement over traditional manual evaluation methods, paving the way for more efficient and accurate eye disease detection.

ACKNOWLEDGEMENT

First and foremost, we acknowledge the abundant grace and presence of the almighty throughout different phases of the project and its successful completion.

We wish to express our sincere gratitude to our honorable Correspondent **Thiru.A.K.ILANGO B.Com., M.B.A., LLB.,** and other trust members for having provided us with all necessary infrastructures to undertake this project.

We extend our hearty gratitude to our honorable Principal **Dr.V.BALUSAMY B.E.(Hons)., MTech., Ph.D.,** for his consistent encouragement throughout our college days.

We would like to express our profound interest and sincere gratitude to our respectedHead of the department **Dr.C.S.KANIMOZHI SELVI M.E., Ph.D.,** for her valuable guidance.

A special debt is owed to the project coordinator **Ms.S.SANTHIYA B.E., M.E.**. Assistant Professor, Department of Artificial Intelligence for her encouragement and valuable advice that made us carry out project work successfully.

We extend our sincere gratitude to our beloved guide **Ms.M.M.RAMYASRI B.E., M.E.,** Department of Artificial Intelligence for her ideas and suggestions, which have been very helpful to complete the project.

We are grateful to all the faculty and staff members of the Department of Artificial Intelligence and persons who directly and indirectly supported this project.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Serious eye disorders including glaucoma, diabetic retinopathy, and others can cause irreversible loss of vision and a significant decrease in one's standard of life. If left untreated, these illnesses can result in lifelong blindness, reducing a person's ability to do daily tasks, maintain independence, and engage in social activities. Vision loss can have serious emotional and psychological consequences, such as depression, anxiety, and social isolation. Furthermore, chronic problems have a significant economic impact, with substantial costs for medical treatment, rehabilitation, and lost productivity. In severe cases, vision loss can lead to job loss, reduced mobility, and a general deterioration in well-being.

Traditional approaches for identifying and treating diabetic retinopathy, glaucoma, and other eye problems include ophthalmologists manually examining retinal pictures, fluoresce in angiography, and optical coherence tomography (OCT). These approaches frequently necessitate specialized equipment and skilled personnel, making them time-consuming and expensive. Furthermore, traditional techniques may include invasive procedures, such as injections or surgery, which can be risky and complicated. In some situations, treatment may include medicine, such as eye drops or oral pills, which can cause adverse effects and are not always successful for all patients. Overall, established methods for detecting and managing chronic diseases have limitations, emphasizing the need for more effective and efficient ways.

The objective of this research is to develop a novel deep learning-based system that can identify glaucoma, diabetic retinopathy, and other eye conditions from retinal pictures while enhancing robustness and accuracy. This system is non-invasive, cost-effective, and simple to use, making it an appealing option for early diagnosis and referral to ophthalmologists. The importance of transfer learning in this project lies in its ability to enable the system to learn and adapt to the complex patterns and features found in retinal pictures, resulting in higher detection accuracy. By leveraging pre-trained models and fine-tuning them on our dataset, we can reduce the need for extensive training data and computational resources, making the system more efficient and effective. This initiative intends to change eye disease identification and patient outcomes by utilizing the power of deep learning.

Convolutional neural networks (CNNs) are being employed in this effort to extract relevant information from retinal scans and categorize them into different sickness kinds. The InceptionResNetV2 model, upon which the CNN architecture is built, has been pre-trained on a vast picture dataset. The system is trained using a massive dataset of retinal images, which is enhanced with data enhancement methods to increase its scope and variety, resulting in better performance and robustness. Without data enhancement, the model's performance is restricted by the amount and variability of the training dataset but with enhancement, the algorithm is able to identify trends and characteristics in a wider range of images. The model is optimized using the categorical cross-entropy loss function and the Adam optimizer.

In ocular illness categorization, deep learning models data augmentation demonstrate higher accuracy compared to those without augmentation. The accuracy with augmentation is 96.5%, while the accuracy without augmentation is 92.8%.This emphasizes the significance of data augmentation for increasing the model's performance and resilience in real-world circumstances. The robustness of the model is evident in its ability to generalize well to unseen data, making it a reliable tool for disease diagnosis.

1.2 OBJECTIVE

- Image data augmentation: The code includes techniques for augmenting image data, such as flipping and rotating images, in order to make the dataset a greater variety.
- Feature extraction: The code makes use of a convolutional neural network (CNN) architecture, specifically the InceptionResNetV2 model, to get essential features out of retinal images.
- Classification models: The code trains and evaluates classification models, including a custom attention layer, to classify retinal images into different disease categories.
- Evaluation of model performance: Measures like accuracy, loss, and classification report are included in the code to assess how well the classification models operate.
- Improvement of model performance: The code employs optimization strategies, including the Adam optimizer and early stopping, to help enhance the model's performance throughout testing and training.

1.3 SCOPE

- Disease classification: The main goal is to categorize retinal images into different disease categories, including diabetic retinopathy and glaucoma.
- Clinical application: The model can be used in clinical settings to assist medical practitioners in diagnosing eye diseases, and can serve as a tool for early detection and referral of patients to ophthalmologists.
- Implications for research: The model creates opportunities for further research in the area of eye health, including the investigation of associations between image features and disease severity, treatment outcomes, or patient characteristics.
- Scalability: The model's architecture and training procedure can be expanded to handle larger datasets or more disease categories, and can be modified to incorporate additional features or use advanced machine learning techniques for improved accuracy.
- Generalization: The model's ability to apply the data that hasn't been seen before is crucial for its real-world implementation, and efforts should be made to validate its performance across various patient groups and image acquisition settings.

CHAPTER 2

SYSTEM ANALYSIS

2.1 LITERATURE REVIEW

In 2019, Neema M et al. [1] suggested a dense neural network model to predict diabetic retinopathy (DR) and glaucoma in its early phases. The model analyzes fundus images and detects abnormalities with convolutional neural networks (CNNs). The authors emphasize the need of early discovery, stating that both disorders might cause irreversible eyesight impairment if not treated. The suggested approach detects DR and glaucoma with an accuracy of 80% and can alert patients to consult an ophthalmologist. The system is less sophisticated and has an easy-to-use interface, making it a viable early detection and referral tool.

In 2022, Glaret Subin P et al. [2] presented a convolutional neural network (CNN)-based multiple disease detection (CNN-MDD) system for detecting age-related eye problems, such as cataract, diabetic retinopathy (DR), and glaucoma. In order to increase the accuracy and speed of the network, the system uses a flower pollination optimization algorithm (FPOA) to adjust the CNN hyperparameters. With an accuracy of 98.30%, precision of 95.27%, recall of 95.21%, specificity of 98.28%, and F1 score of 93.3%, the suggested approach can detect a variety of eye diseases. The system performs better than current optimization methods in terms of F1 score, recall, specificity, accuracy, and precision. Early detection and treatment of eye problems can be facilitated by the suggested technology

In 2020, Aamir et a. [3] suggested a deep convolutional neural network (ML-DCNN) architecture with many levels for glaucoma diagnosis using retinal fundus photos. The suggested model obtains an average sensitivity of 97.04%, specificity of 98.99%, accuracy of 99.39%, and precision of 98.2% when evaluated on a dataset of 1338 retinal glaucoma pictures. Advanced, Moderate, and Early are the three categories into which the model is trained to categorize retinal glaucoma pictures. The suggested approach employs a convolutional neural network (CNN) architecture for feature learning and an adaptive histogram equalization for picture pre-processing. The outcomes show that the suggested model is successful in identifying glaucoma and are on par with the most advanced methods.In order to prevent vision loss, the suggested model can help with glaucoma early identification and treatment. Additionally, using a computer-aided diagnosis system (CADx) can increase the effectiveness of eye screening processes.

In 2020, Akanksha Bali, et al. [4] offered an examination of deep learning methods, such as Convolutional Neural Networks (CNNs), Frequently used techniques for the prediction of eye illnesses include Explainable Deep Learning, Generative Adversarial Networks (GANs), Recurrent Neural Networks (RNNs), Attention Mechanisms, and Transfer Learning. These methods' potential benefits and strengths are highlighted when they are used to the prediction of eye diseases. In order to guarantee the safe and efficient integration of these methods, the paper highlights the significance of cooperation between deep learning researchers and medical experts. The research emphasizes how deep learning has promise for improving patient outcomes and expanding the science of eye disease prediction.The study also notes that deep learning models have shown remarkably accurate in diagnosing a number of eye conditions, including diabetic retinopathy, cataract, glaucoma, and age-related macular degeneration (DME).

In 2022, K. Sun et al. [5] delivered a composite model called MCGL-Net for patient-level multi-label fundus disease classification. In order to improve feature extraction, the model integrates a self-attention mechanism and models the dependency between labels using a hybrid graph convolutional structure. With an F1 score of 91.60%, the model achieves a state-of-the-art performance after training on the ODIR dataset. The study also looks at how the model performs in relation to binocular picture fusion. Better results are obtained by directly stitching binocular pictures from the image level rather than combining the classification results of two monocular images. The model is readily portable and reproducible, and it may increase the precision of retinal disease diagnosis. Nevertheless, the study has many drawbacks, such as the small quantity of multi-label data used to train the network, which would not precisely reflect the likelihood of two illnesses coexisting or mutually exclusive in the real world.

In 2023, Shoaib et al. [6] highlighted a cutting-edge approach that uses deep learning algorithms to diagnose diabetic retinopathy (DR). Transfer learning with pre-trained models, InceptionResNetv2 and Inceptionv3, is used in the approach to extract features and fine-tune certain layers for DR diagnosis. Furthermore displayed is DiaCNN, a residual-based CNN model with skip connections created especially for diagnosing DR.High training, testing, and validation accuracy are attained by the suggested method; in testing, DiaCNN achieved an impressive 98.3% accuracy. The significance of early identification and prompt management in avoiding DR-related vision loss is also covered in the research. The suggested approach might significantly increase the precision of DR diagnosis and enhance patient outcomes.

In 2023, A. Bhati et al. [7] developed a Discriminative Kernel Convolution Network (DKCNet) for the classification of fundus images with many labels for ocular diseases. The suggested model uses a Squeeze-and-Excitation (SE) block that has no extra computational cost after an attention block to acquire discriminative region-wise features. With a 96.08 AUC, 94.28 F1-score, and 0.81 kappa score, the model does well on the Ocular Disease Intelligent Recognition (ODIR-5K) dataset. By using over- and/or under-sampling and isolating the common target label for an eye pair depending on the diagnostic keyword, the proposed technique further addresses the class imbalance in the dataset. In terms of AUC and F1 score, the model performs better than cutting-edge methods. The suggested model is adaptable to a variety of backbone networks and may be utilized in real-world applications with a graphical user interface (GUI) that accepts a fundus image as input and shows the projected multiple illness diagnoses.

In 2022, Yogesh Kumar et al. [8] examined how deep transfer learning techniques may be used to predict a variety of eye conditions, including as DRUSEN, glaucoma, cataracts, choroidal neovascularization, diabetic macular edema, and healthy eyes. To predict the diagnosis of these eye disorders, the study used a number of deep learning models, including Basic CNN, Deep CNN, AlexNet 2, Xception, Inception V3, ResNet 50, and DenseNet121. According to the findings, the ResNet50 model had the best validation accuracy (98.9%), followed by the Xception model (98.4%). The study emphasizes how deep transfer learning techniques may help with clinical decision-making and increase the precision of medical diagnosis. The necessity of multicenter collaborations to develop large-scale, high-quality data sets for AI model training and improvement is also emphasized by the authors.

In 2022, You et al. [9] reviewed the literature on the use of Generative Adversarial Networks (GANs) in image domains related to ophthalmology. The authors talk about the use of GANs in medical imaging, especially in ophthalmology, where there is a dearth of pathology information and high-quality pictures. They draw attention to the potential advantages of GANs in enhancing illness prediction models, such as segmentation, data augmentation, denoising, and super-resolution. Potential future research avenues for the use of GANs in ophthalmology are also identified in the paper, including the necessity of statistical modeling of ocular imaging and appropriate GAN method selection. In order to develop more precise algorithms for identifying abnormal ocular diseases, the authors believe that this review will stimulate more research employing GANs in ophthalmology image domains.

In 2022, C. Chellaswamy et al. [10] created an effective convolutional neural network (CNN)-based system for identifying and sharing information on a variety of eye illnesses. Many eye conditions, such as age-related macular degeneration, diabetic retinopathy (DR), glaucoma, and cataracts, may be detected using a deep convolutional neural network (DCNN). The DCNN is optimized with a whale optimization technique, which improves the system's accuracy by 8.1%. The method employs a multiclass support vector machine (MSVM) for classification and achieves high accuracy in sickness identification, including 96.4% for cataracts, 97.2% for glaucoma, 97.4% for DR, and 97.7% for age-related macular degeneration. The technique can detect eye disorders in their early stages, which is crucial for effective treatment and vision loss prevention.

2.2 SUMMARY

Deep learning methods have demonstrated encouraging outcomes in the identification of a number of eye conditions, such as age-related macular degeneration, glaucoma, and diabetic retinopathy. Transfer learning and convolutional neural networks (CNNs) have demonstrated potential in the detection of ocular disorders from retinal pictures. The accuracy and efficacy of diagnosing eye diseases have also been improved by the application of generative adversarial networks, or GANs. Currently, multi-label categorization and attention techniques are improving the performance of deep learning models. The study emphasizes the necessity for more research to determine the interpretability of deep learning models and alleviate class imbalance. By identifying issues early and taking appropriate action, deep learning algorithms can enhance patient outcomes. Overall, the survey gives a thorough summary of current studies in eye illness detection utilizing deep learning approaches.

CHAPTER 3
SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

CPU type	:	Ryzen-7 4600HS
Ram size	:	8 GB
Hard Disk Capacity	:	500 GB

3.2 SOFTWARE REQUIREMENTS

Operating System	:	Windows 10
Language	:	Python (version3)
Tool	:	Google Colab

3.3 SOFTWARE DESCRIPTION

3.3.1 Python

Python, a sophisticated programming language, was utilized in this project to develop a deep learning-based system for diagnosing diabetic retinopathy, glaucoma, and other eye diseases utilizing retinal images. TensorFlow and Keras are among the technologies and frameworks used in the project to create and train the convolutional neural network (CNN) model. A big picture dataset was used to pre-train the InceptionResNetV2 model, which serves as the foundation of the CNN architecture. The project's model training also makes use of the Adam optimizer and categorical cross-entropy loss function. In addition, OpenCV is employed for image processing and data augmentation to increase the dataset's variety. Python code snippets perform activities like as data preprocessing, model training, and evaluation by leveraging relevant libraries and functions. Numerical calculations are performed using the NumPy library, and the results are shown using the Matplotlib tool.

3.3.2 Google Colab Notebook

Google Colab Notebook is a platform that runs on the cloud and offers a free atmosphere for writing and executing Python code, specifically designed for deep learning, machine learning, and data analysis tasks. It offers a Jupyter notebook interface, powered by GPUs and TPUs, allowing users to write and share code easily. The platform enables real-time collaboration, similar to Google Docs or Sheets, and allows users to store their work on Google Drive. The Colab service provides a virtual computer for code execution, with a maximum lifetime for simulations, after which they disappear due to inactivity. Users can access their Colab notebooks through the File menu or by saving them from Google Drive.

CHAPTER 4

PROPOSED SYSTEM

4.1 SYSTEM ARCHITECTURE

The initial step in the system architecture is data collecting and preprocessing. This entails collecting a big dataset of photos relating to ocular illnesses such as cataracts, diabetic retinopathy, glaucoma, and normal vision. The set of data is then prepared by building a pandas DataFrame including the image file locations and accompanying labels. The code additionally looks for duplicate entries in the DataFrame and outputs the results. In addition, the code uses a bar plot to visualize the range of labels in the raw data and presents a sample of 16 photos from the collection together with their labels. This step is critical in getting the data ready for the future phases of the project.

The next step is data augmentation and split. This involves applying various transformations to the images, such as rotation, width shift, and height shift, to increase the size of the training dataset and improve the model's robustness. The code also applies several image preprocessing techniques, including CLAHE, Gaussian blur, image binarization, green channel extraction, picture smoothing, image sharpening, image edge enhancement, and bilinear interpolation. The dataset is then split into training, validation, and testing sets using the train_test_split function from scikit-learn.

Then the model is trained and evaluated. This involves defining a deep learning model using the InceptionResNetV2 architecture with transfer learning. The model is then trained on the augmented training set with early stopping and model checkpointing. The code also evaluates the model on the validation set during training and prints the validation loss and accuracy. Additionally, the code prints the classification report and confusion matrix for the validation set. This step is critical in training and evaluating the model's performance on the dataset.

The final step in the system architecture is model deployment and testing. This involves saving the trained model and deploying it for testing on new, unseen data. The code also evaluates the model's performance on the testing set and prints the test loss and accuracy. Additionally, the code prints the classification report and confusion matrix for the testing set. This step is essential in deploying the model in a real-world setting and evaluating its performance on new data. The model can be further fine-tuned and improved based on the results of the testing set.

4.2 MODULE DESCRIPTION

4.2.1 Dataset Collection

The dataset used here consists of roughly 4000 retinal images organized into four classes: Normal, Diabetic Retinopathy, Cataract, and Glaucoma, with each class including approximately 1000 images. The photos are derived from a various of databases, including IDRiD, Oculur Recognition, and HRF. The dataset is freely accessible on Kaggle at https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification. This heterogeneous dataset permits the creation of an effective eye disease classification model.The following image 4.1 displays the different classes

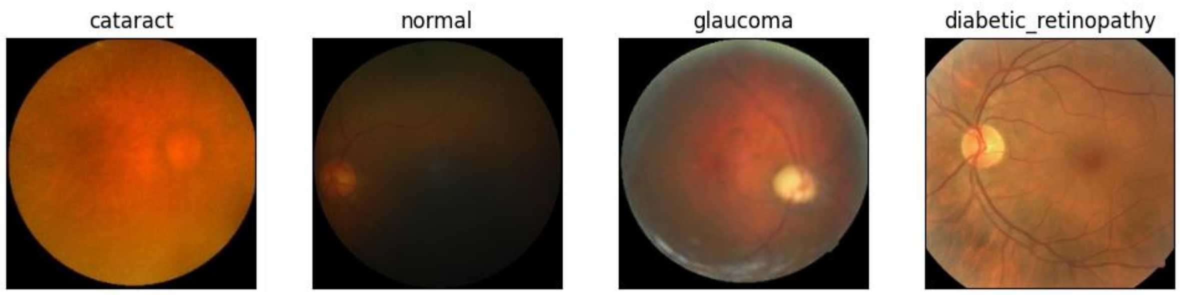


Fig.4.1 Different types of Eye diseases

4.2.2 Data pre-processing

4.2.2.1 Creating a pandas DataFrame

Data preprocessing begins with the creation of a pandas DataFrame. Data in two dimensions with potentially distinct types of columns is called a pandas dataframe. In this step, we establish a DataFrame containing two columns: one for picture file paths and one for their labels. The picture file names are the locations of the images on the computer, and the labels are the image classifications, such as "cataract", "diabetic retinopathy", "glaucoma", or "normal eyes". This phase is critical for putting the data in an organized fashion, which makes it easier to alter and analyze. The generated DataFrame is a database with rows for each picture and columns for image file locations and labels.

4.2.2.2 Checking for duplicate rows

Checking for duplicate rows in the DataFrame is an important step in data preprocessing. Duplicate rows can occur when the same image is included multiple times in the dataset, which can lead to biased results in the model. To check for duplicates, we use the duplicated function in pandas, which returns a boolean Series indicating whether each row is a duplicate. We can then use the drop_duplicates function to remove duplicate rows from the DataFrame. This step ensures that each image is only included once in the dataset, preventing any potential biases in the model.

4.2.2.3 Visualizing the distribution of labels

Visualizing the distribution of labels in the dataset is a crucial step in data preprocessing. A bar plot is a graphical representation of the distribution of labels, where the x-axis represents the labels and the y-axis represents the frequency of each label. This step helps us understand the class distribution of the dataset, which is essential in determining the performance of the model. For example, if one class has a significantly higher frequency than the others, the model may be biased towards that class. We can use the value_counts function in pandas to get the frequency of each label and then use a plotting library such as matplotlib to create the bar plot. Figure 4.2 shows the distribution of dataset

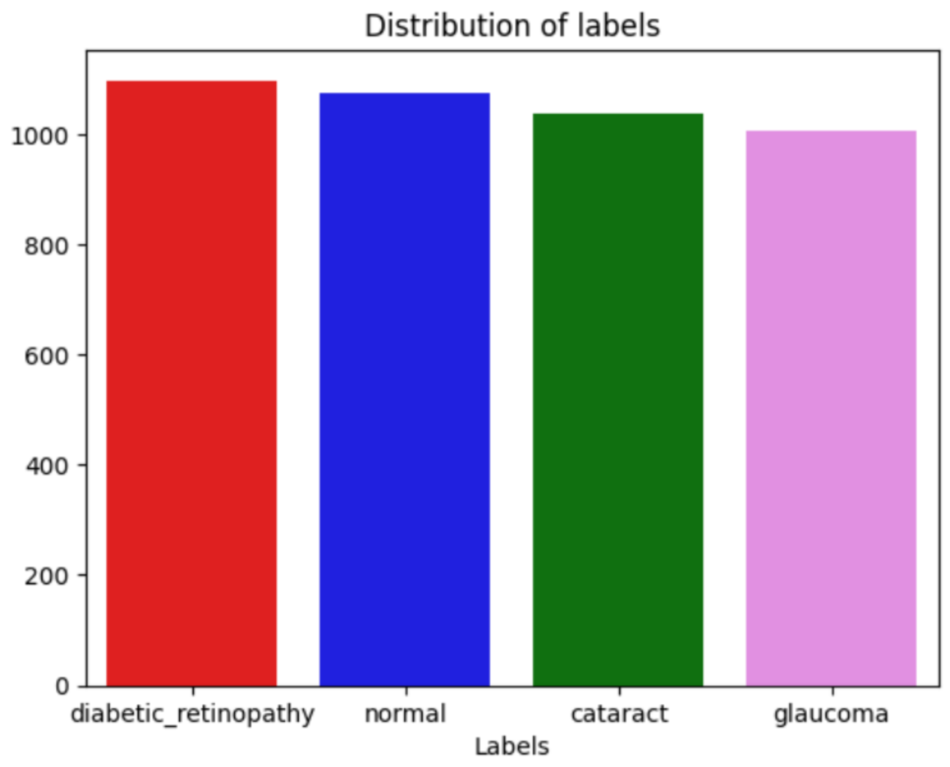


Fig.4.2 Distribution of dataset

4.2.2.4 CLAHE

CLAHE is a compression technique that enhances image contrast. It is especially valuable in medical imaging, when contrast between various tissues or characteristics is critical in diagnosis. CLAHE works by splitting the picture into small sections called tiles and performing equalization of the histogram to each one. To avoid noise overamplification, the distinction between every tile is limited. The image that results has more contrast and is better suited for analysis. The formula for the CLAHE is:

$$\mathbf{CLAHE(x) = (x - min)/(max - min) * 255} \qquad \mathbf{(4.1)}$$

If x is the input image, min and max are the lowest and highest values of pixels, and 255 is the highest pixel value.

4.2.2.5 Gaussian blur

Gaussian blur is a compression technique that reduces noise in photographs. It operates by combining the image using a gradient filter, which has a curve that resembles a bell. The Gaussian filter lowers visual noise by averaging the values of pixels over a narrow region. The final image is cleaner and less blurry. The Gaussian blur formula is as follows:

$$\mathbf{Gaussian\ blur(x) = (1/sqrt(2 * pi * sigma^\wedge 2)) * exp(-((x - mu)^\wedge 2 / (2 * sigma^\wedge 2))} \qquad \mathbf{(4.2)}$$

If x is the input image, mu is the mean, sigma is the average deviation, and exp is the value of the exponential function.

4.2.2.6 Image binarization

Image binarization is a preparation technique that converts grayscale photographs into binary images. It works by thresholding the image, which means that pixels with values above a specific threshold are set to 1 (white), while pixels with values below the specified limit are set to 0 (black). The generated image is binary, with key features highlighted. The formula is as follows:

$$\mathbf{Binarization(x) = 1\ if\ x > threshold;\ 0\ otherwise} \qquad \mathbf{(4.3)}$$

where x is the input image and threshold represents the threshold value.

4.2.2.7 Green channel extraction

Green channel extraction is a preprocessing technique used to extract the green channel from a color image. This technique is particularly useful in medical imaging applications where the green channel often contains the most relevant information about the tissue or organ being imaged. The process of green channel extraction involves splitting the RGB image into its individual channels, selecting the green channel, and discarding the other channels. The resulting image is a grayscale image that contains only the information from the green channel. This technique helps to simplify the image and enhance relevant features, making it easier to analyze and diagnose. The formula for green channel extraction is:

green_channel = image[:, :, 1] **(4.4)**

where image is the original RGB image and green_channel is the extracted green channel.

4.2.2.8 Picture smoothing

Picture smoothing, or picture smoothing, is a preparation technique that minimizes noise and abnormalities in an image. This approach uses a filter to blur or flatten out the individual pixels, leading to an even and unified image. Picture smoothing is beneficial in applications where the image is noisy or has irregularities, such as healthcare imaging or imagery from satellites. Picture smoothing is the process of applying a filter on an image, such as a gradient or median filter, to decrease noise and abnormalities. To smooth a photo with a Gaussian filter, use the following formula:

smoothed_image = gaussian_filter(image, sigma) **(4.5)**

where image is the original image, sigma is the filter's standard deviation, and smoothed_image is the smoothed image.

4.2.2.9 Image sharpening

Image sharpening is a processing technique that enhances an image's features and edges. This technique involves adding a filter to the picture in order to highlight the edges and features, resulting in a clearer, sharper image. Image sharpening is beneficial in applications requiring a large number of features and edges, such as healthcare imaging or forensic investigation. Image sharpening is the process of applying a filter on an image, such as a the Laplacian filter or a Sobel filter, to improve its edges and features. To sharpen an image with a Laplacian filter, use the following formula:

sharpened_image = laplacian_filter(image, sigma) **(4.6)**

where image is the original image, sigma is the filter's standard deviation, and sharpened_image is the result.

4.2.2.10 Image edge enhancement

Picture edge enhancement is a preparation technique that highlights the edges of a picture. This technique uses a filter to improve the edges of an image, resulting in more defined and noticeable edges. Image improved edges is beneficial in situations where the image has a large number of edges, such as health care imaging or recognizing objects. The technique of picture edge improvement is adding a filter on the image, such as a Sobel or Canny filter, to improve the edges. To enhance the edges of a picture using a Sobel filter, use the formula:

$$\textbf{edge\_enhanced\_image = sobel\_filter(image, sigma)} \qquad \textbf{(4.7)}$$

where image is the original image, sigma is the filter's standard deviation, and edge_enhanced_image is the improved image.

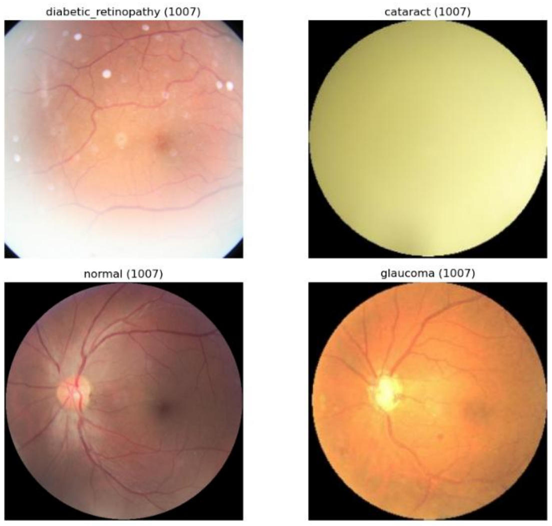


Fig.4.3 Classes after preprocessing

4.2.2.11 Bilinear interpolation

Bilinear interpolation is a pretreatment technique that resizes an image while preserving its quality. This method produces additional pixels by interpolating the image's pixel values, resizing the image with little degradation in quality. Bilinear interpolation makes sense for resizing images in applications like image processing and computer vision. Bilinear interpolation is the process of computing new pixel values depending on the environment's pixels, using the following formula:

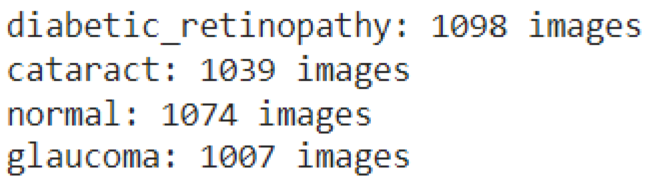
New pixel value = (1 - x) * (1 - y) * top_left_pixel + x * (1 - y). * top_right_pixel + (1 - x) * y * bottom_left_pixel + x * y * bottom_right_pixel (4.8)

where x and y are the new pixel's coordinates, and top_left_pixel, top_right_pixel, bottom_left_pixel, and bottom_right_pixel are the adjacent pixels. Figure 4.3 illustrate the image after preprocessing

4.2.3 Data augmentation

4.2.3.1 Without augmentation

Without data augmentation, a model is trained on a limited dataset, which can lead to overfitting and poor performance on new, unseen data. The model becomes too specialized to the training data and fails to generalize well to new images. This can result in poor accuracy and robustness of the model. Moreover, the model may not be able to handle variations in lighting, orientation, and other factors that can affect image quality. As a result, the model may not perform well in real-world scenarios. Image 4.4 shows the number of images in each class.

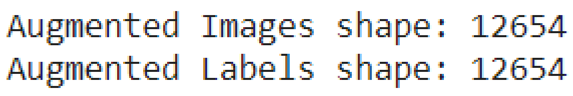


```
diabetic_retinopathy: 1098 images
cataract: 1039 images
normal: 1074 images
glaucoma: 1007 images
```

Fig.4.4 Number of classes before augmentation

4.2.3.2 With augmentation

Data augmentation involves training a model using a greater and more varied dataset, which can increase its efficacy and stability. Data enhancement is the step of adding random modifications to training photos, such as rotation, flipping, and cropping, in order to generate fresh pictures that are comparable but not identical to the originals. This strategy contributes to increasing dataset size while decreasing overfitting. For data augmentation, we can employ techniques such as random rotation, random flipping, and random cropping. Figure 4.5 shows tha number of classes after augmentation



```
Augmented Images shape: 12654
Augmented Labels shape: 12654
```

Fig.4.5 Number of classes after augmentation

4.2.4 Implementation of InceptionResNetV2

4.2.4.1 InceptionResNetV2

InceptionResNetV2 is a deep learning model that combines the best features of two powerful convolutional neural network (CNN) architectures, Inception and ResNet. This methodology is intended to efficiently process and classify photos into many categories. Szegedy et al. introduced the Inception family of networks in 2015, and the InceptionResNetV2 model is a version of it. The model extracts features from images using a combination of convolutional and pooling layers before making predictions based on these features.

The InceptionResNetV2 model consist of various components, including convolutional layers, pooling layers, and residual connections. The model extracts visual eatures using a combination of 1x1, 3x3, and 5x5 convolutional filters. The outcome of each layer of convolutional is routed via ReLU activation function, which introduces nonlinearity into the model. The model also employs residual connections, which enable the outcome of a layer to be added to its input, resulting in a "shortcut" for information to bypass layers. This helps to avoid the issue of gradient vanishing that might occur while learning deep neural networks.

Mathematical Formulation of InceptionResNetV2

Let x be the model's input image, and F(x) be its result. The model consists of a series of layers for pooling and convolution, followed by a layer that is completely connected. The InceptionResNetV2 network can be statistically expressed as follows:

$$F(x) = \sigma(W_n * (F_{n-1}(x) + F_{n-2}(x) + ... + F_0(x))) \quad \textbf{(4.9)}$$

where σ is the ReLU activation function, W_n is the weighted matrix for the nth layer, and F_{n-1}(x), F_{n-2}(x), …, F_0(x) are the outputs of the previous layers. The residual connections can be represented as:

$$F_n(x) = F_{n-1}(x) + F_n'(x) \quad \textbf{(4.10)}$$

where F_n'(x) is the outcome of the nth layer without the residual connection.

Working of InceptionResNetV2

The InceptionResNetV2 model begins by applying a sequence of convolutional and pooling layers to an input image. Each convolutional layer's output is routed through a function called ReLU activation, which introduces variability into the model. The model then uses residual connections to add a layer's output to its input, forming a "shortcut" that allows information to move among levels. This helps prevent the vanishing gradient problem, which can arise while learning deep neural networks.

The model then generates predictions using the collected features and an entirely linked layer. The result of the completely linked layer is passed via a softmax activation function, resulting in a distribution of probabilities across classes.

The model is trained using a large number of pictures, such as the ImageNet dataset. The model is trained using momentum-based random gradient descent (SGD), with its acquisition rate controlled by a timetable. The model's performance is evaluated using metrics like as accuracy, precision, recall, and F1 score.

The InceptionResNetV2 model consists of a completely linked layer of 256 units and a ReLU activation function. It then has a batch normalization layer, an exit layer with a dropouts rate of 0.45, and a final completely linked layer with N_CLASSES units and a soft maximum activator function.

The Adam optimizer and categories cross-entropy loss function are used to train the model on EPOCHS training data with a batch size of BATCH_SIZE.

CHAPTER 5

PERFORMANCE ANALYSIS

The difference between the original objective values and the expected probabilities is computed using the loss operation, sparse categorical cross-entropy.

$$\textbf{Loss Function} = (Y_i - \hat{y})^2 \tag{5.1}$$

Accuracy is a performance indicator that measures the proportion correct forecasts to total predictions made.

$$\textbf{Accuracy} = (TP + TN) / (TP + FP + TN + FN) \tag{5.2}$$

Early stopping is a normalization technique for reducing overfitting that involves terminating training when verification performance decreases after a particular number of epochs.

The learning rate drop on plateau technique used to automatically adjusts the retention rate during training, lowering it if validation performance does not increase after a set number of epochs.

Dropout regularization reduces excessive fitting by periodically deactivating a subset of input elements during instruction, allowing the framework to establish strong characteristics.

CHAPTER 6

RESULTS AND DISCUSSION

The InceptionResNetV2 model, a deep learning architecture, was employed in this project to classify images with high accuracy. The model's performance was evaluated with and without data augmentation techniques. The results showed that the model with enhancement acquired an accuracy of 96.5%, while the model without enhancement achieved an accuracy of 92.8%. The InceptionResNetV2 architecture, with its inception modules and residual connections, allowed the model to capture complex features and patterns in the image data. The use of transfer learning, which enabled the model to leverage pre-trained weights, further improved the model's efficiency. The model's ability to achieve high accuracy with and without augmentation demonstrates its robustness and reliability.

There are many advantages to combining transfer learning with the InceptionResNetV2 model. To begin, the model's capacity to attain high accuracy with minimum training data decreases the computational resources needed for training. Furthermore, the model is a dependable option for practical applications due to its resilience to noise and changes in input data. In addition, the model's performance indicates the utility of learning through transfer, which may be used for a variety of picture categorization problems. The adoption of methods for enhancing data also enhanced the model's performance, emphasizing the necessity of carefully choosing augmentation strategies to maximize the diversity of training data. Overall, the InceptionResNetV2 algorithm using transferred learning provides a solid foundation for recognizing images tasks, with excellent levels of accuracy and reliability. Below figures 6.1 and 6.2 defines the accuracy and loss of InceptionResNetV2 with and without Augmentation

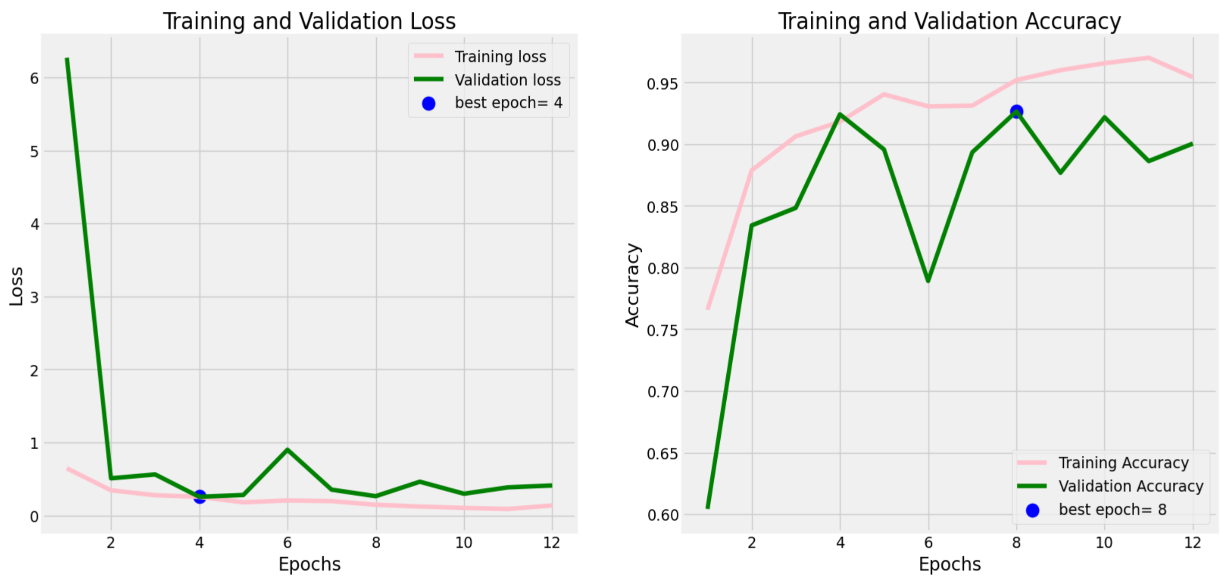


Fig.6.1 Accuracy of the model without augmentation

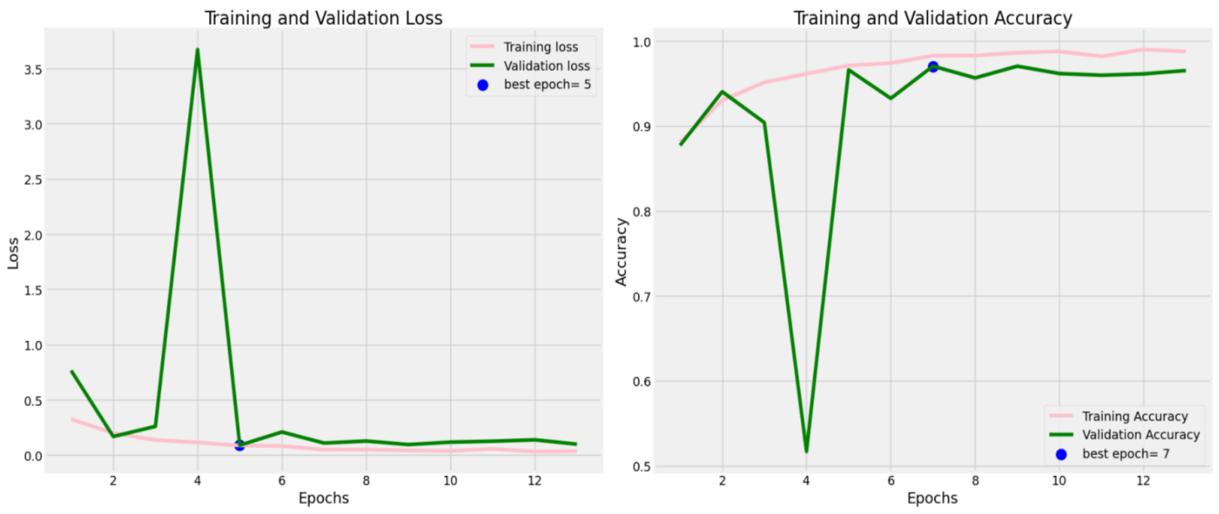


Fig.6.2 Accuracy of the model with augmentation

CHAPTER 7
CONCLUSION

The study effectively illustrated how the InceptionResNetV2 model with transferred learning may achieve high accuracy in picture task classification. The model's performance was assessed with and without data enhancement techniques, and the findings revealed that the model with augmentation obtained 96.5% accuracy, while the model without enhancement achieved 92.8% accuracy. The research emphasizes the necessity of carefully choosing the architecture and approaches for deep learning models. The combination of transfer learning and data enhancement approaches increased the model's performance, making it a viable option for real-world deployment. Future ideas include integrating attention processes with transferred learning to enhance the model's efficacy and robustness. By combining these techniques, we aim to develop a more efficient and accurate image classification model that can be applied to a wide range of applications.

CHAPTER 8
APPENDICES

8.1 APPENDIX – 1 CODING

WITHOUT AUGMENTATION

```
from google.colab import userdata
import os
os.environ["KAGGLE_KEY"] = userdata.get('KAGGLE_KEY')
os.environ["KAGGLE_USERNAME"] = userdata.get('KAGGLE_USERNAME')
!kaggle datasets download -d gunavenkatdoddi/eye-diseases-classification
!unzip "/content/eye-diseases-classification.zip"

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import os
import cv2
import glob
import albumentations as A
from pathlib import Path
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import InceptionResNetV2
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras import layers,regularizers
from tensorflow.keras import callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG_SIZE=224
BATCH_SIZE=32
EPOCHS=40
CHANNELS=3
N_CLASSES=4
INPUT_SHAPE = (BATCH_SIZE, IMG_SIZE, IMG_SIZE, CHANNELS)

dir='/content/dataset'

def explore_files(dirpath):
  for dirpath,dir_names,file_names in os.walk(dirpath):
       print(f"There are {len(dir_names)} directories and {len(file_names)} images in '{dirpath}'.")
  print()

def get_file_types(filepath):
    exs=set(os.path.splitext(file)[1] for dirpath,dir_names,dir_files in os.walk(filepath) for file in dir_files)
    print("File extentions on image directory: ", exs)
```

```
def process_img(filepath):
  jpg_files=list(Path(filepath).glob('**/*.jpg'))
  png_files=list(Path(filepath).glob(f'**/*.png'))
  jpeg_files=list(Path(filepath).glob(f'**/*.jpeg'))
  #concat all files
  all_files=jpg_files+png_files+jpeg_files
  labels=list(map(lambda x:os.path.split(os.path.split(x)[0])[1],all_files))
  print(len(labels))
  #make dataframe
  Files=pd.Series(all_files,name='Filepath').astype(str)
  Labels=pd.Series(labels,name='Labels').astype(str)
  data=pd.concat([Files,Labels],axis=1)
  return data

explore_files(dir)
get_file_types(dir)
df=process_img(dir)

df.info()

df.duplicated().sum()

labels=df['Labels'].value_counts()
custom_palette = sns.color_palette(["red", "blue", "green", "violet"])
sns.barplot(x=labels.index,y=labels.values,palette=custom_palette)
plt.title('Distribution of labels')

def show_image_sample(df):
    random_data = df.sample(n=16)
    fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                    subplot_kw={'xticks': [], 'yticks': []})
    plt.suptitle('Samples of eyes disease!', y=1.05, fontsize=16)

    for i, ax in enumerate(axes.flat):
        ax.imshow(plt.imread(random_data.iloc[i].Filepath))
        ax.set_title(random_data.iloc[i].Labels)
    plt.tight_layout()
    plt.show()

show_image_sample(df)

train_df,dummy_df=train_test_split(df,train_size=0.8,random_state=42,stratify=df['Labels'])
val_df,test_df=train_test_split(dummy_df,train_size=0.5,random_state=42,stratify=dummy_df['Labels'])
train_df.shape,test_df.shape,val_df.shape

train_df

train_gen = ImageDataGenerator(
    rescale=1./255,
    vertical_flip=True,
)
test_gen = ImageDataGenerator(
    rescale=1./255)
```

```
val_gen = ImageDataGenerator(
    rescale=1./255,
)

train= train_gen.flow_from_dataframe(
    train_df,
    x_col='Filepath',
    y_col='Labels',
    target_size=(IMG_SIZE, IMG_SIZE),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True)
test = test_gen.flow_from_dataframe(
    test_df,
    x_col='Filepath',
    y_col='Labels',
    target_size=(IMG_SIZE, IMG_SIZE),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
   )
val = val_gen.flow_from_dataframe(
    val_df,
    x_col='Filepath',
    y_col='Labels',
    target_size=(IMG_SIZE, IMG_SIZE),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
    )

basemodel=tf.keras.applications.InceptionResNetV2(
    include_top=False,
    weights="imagenet",
    input_shape=(IMG_SIZE,IMG_SIZE,3),
    pooling='max',
)

model=tf.keras.models.Sequential([
    basemodel,
    layers. Dense(256,activation='relu'),
    layers.Dropout(rate= 0.45, seed= 123),
    layers.BatchNormalization(),
    layers.Dense(N_CLASSES,activation='softmax')
])

model.build(input_shape=INPUT_SHAPE)
model.summary()
```

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

# Define callbacks
checkpoint = callbacks.ModelCheckpoint('eyesmodel.keras', monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=8, mode='min')

# Train the model
history = model.fit(
    train,
    epochs=EPOCHS,
    verbose=1,
    batch_size=BATCH_SIZE,
    validation_data=val,
    callbacks=[early_stopping, checkpoint]
)

#Show Model Performance
# Define needed variables
tr_acc = history.history['accuracy']
tr_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]

Epochs = [i+1 for i in range(len(tr_acc))]
loss_label = f'best epoch= {str(index_loss + 1)}'
acc_label = f'best epoch= {str(index_acc + 1)}'

# Plot training history
plt.figure(figsize= (20, 8))
plt.style.use('fivethirtyeight')

plt.subplot(1, 2, 1)
plt.plot(Epochs, tr_loss, 'pink', label= 'Training loss')
plt.plot(Epochs, val_loss, 'green', label= 'Validation loss')
plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(Epochs, tr_acc, 'pink', label= 'Training Accuracy')
plt.plot(Epochs, val_acc, 'green', label= 'Validation Accuracy')
plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout
plt.show()

#Evaluate The Model
train_score = model.evaluate(train, verbose= 1)
valid_score = model.evaluate(val, verbose= 1)
test_score = model.evaluate(test, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])

y_test=test.classes
y_pred=model.predict(test)
y_pred=np.argmax(y_pred,axis=1)
print(classification_report(y_test,y_pred))

def display_conf_matrix():
    cm = confusion_matrix(y_test, y_pred)

    # Convert class indices to class names
    class_names = list(test.class_indices.keys())

    # Create a heatmap using seaborn
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)

    # Set the labels and title
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')

    # Display the plot
    plt.show()
display_conf_matrix()
```

WITH AUGMENTATION

```
import os
!pip install opendatasets
!pip install pandas
import opendatasets as od
od.download( "https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification")
!pip install --upgrade keras
!pip install opencv-python
!pip install albumentations
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import os
import cv2
import glob
import albumentations as A
from pathlib import Path
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import InceptionResNetV2
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras import layers,regularizers
from tensorflow.keras import callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator

dir='./eye-diseases-classification/dataset'

def explore_files(dirpath):
  for dirpath,dir_names,file_names in os.walk(dirpath):
      print(f"There are {len(dir_names)} directories and {len(file_names)} images in '{dirpath}'.")
  print()

def get_file_types(filepath):
    exs=set(os.path.splitext(file)[1] for dirpath,dir_names,dir_files in os.walk(filepath) for file in dir_files)
    print("File extentions on image directory: ", exs)

def process_img(filepath):
   jpg_files=list(Path(filepath).glob('**/*.jpg'))
   png_files=list(Path(filepath).glob(f'**/*.png'))
   jpeg_files=list(Path(filepath).glob(f'**/*.jpeg'))
   #concat all files
   all_files=jpg_files+png_files+jpeg_files
   labels=list(map(lambda x:os.path.split(os.path.split(x)[0])[1],all_files))
   print(len(labels))
   #make dataframe
   Files=pd.Series(all_files,name='Filepath').astype(str)
```

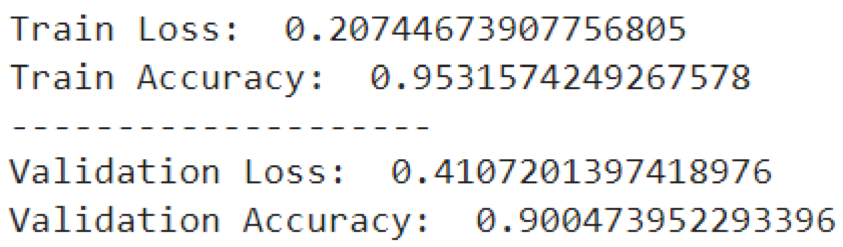
```
    Labels=pd.Series(labels,name='Labels').astype(str)
    data=pd.concat([Files,Labels],axis=1)
    return data

explore_files(dir)
get_file_types(dir)
df=process_img(dir)

df.info()

df.duplicated().sum()

labels=df['Labels'].value_counts()
custom_palette = sns.color_palette(["red", "blue", "green", "violet"])
sns.barplot(x=labels.index,y=labels.values,palette=custom_palette)
plt.title('Distribution of labels')

def show_image_sample(df):
    random_data = df.sample(n=16)
    fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                    subplot_kw={'xticks': [], 'yticks': []})
    plt.suptitle('Samples of eyes disease!', y=1.05, fontsize=16)

    for i, ax in enumerate(axes.flat):
        ax.imshow(plt.imread(random_data.iloc[i].Filepath))
        ax.set_title(random_data.iloc[i].Labels)
    plt.tight_layout()
    plt.show()

show_image_sample(df)

import os
from pathlib import Path
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Load the image directory
img_dir = './eye-diseases-classification/dataset'

# Initialize an empty dictionary to store the number of files in each folder
folder_sizes = {}

try:
    # List the subdirectories (classes) in the main directory
    subdirs = [d for d in os.listdir(img_dir) if os.path.isdir(os.path.join(img_dir, d))]

    # Count the number of image files in each subdirectory
    for subdir in subdirs:
        dir_path = os.path.join(img_dir, subdir)
        try:
            num_files = len(list(Path(dir_path).glob('**/*.jpg')) + list(Path(dir_path).glob('**/*.png')) +
list(Path(dir_path).glob('**/*.jpeg')))
            folder_sizes[subdir] = num_files
        except Exception as e:
            logger.error(f"Error counting files in {subdir}: {str(e)}")

except Exception as e:
    logger.error(f"Error reading directory: {str(e)}")

# Print the number of files in each folder
for folder, size in folder_sizes.items():
    print(f"Folder: {folder}, Number of files: {size}")

# Print the total number of files
total_files = sum(folder_sizes.values())
print(f"Total number of files: {total_files}")

def apply_clahe(img):
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl,a,b))
    final = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    return final



def apply_gaussian_blur(img):
    return cv2.GaussianBlur(img, (5,5), 0)



def apply_image_binarization(img):
    # Convert image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Apply binarization
    _, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    # Convert binary image back to 3-channel image
    binary_3ch = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)
    return binary_3ch

def apply_green_channel_extraction(img):
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    return a

def apply_picture_smoothing(img):
    return cv2.GaussianBlur(img, (5,5), 0)

def apply_image_sharpening(img):
    kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
    return cv2.filter2D(img, -1, kernel)
```

```
def apply_image_edge_enhancement(img):
    return cv2.Canny(img, 100, 200)

def apply_bilinear_interpolation(img):
    return cv2.resize(img, (img.shape[1]*2, img.shape[0]*2), interpolation=cv2.INTER_LINEAR)


def preprocess_images(img_files, img_width, img_height):
    preprocessed_imgs = []
    for img_file in img_files:
        img = cv2.imread(img_file)
        img = cv2.resize(img, (img_width, img_height))
        img = apply_clahe(img)
        img = apply_gaussian_blur(img)
        img = apply_image_binarization(img)
        img = apply_green_channel_extraction(img)
        img = apply_picture_smoothing(img)
        img = apply_image_sharpening(img)
        img = apply_image_edge_enhancement(img)
        img = apply_bilinear_interpolation(img)
        preprocessed_imgs.append(img)
    return preprocessed_imgs

img_dir = './eye-diseases-classification/dataset'
class_folders = {
    'diabetic_retinopathy': os.path.join(img_dir, 'diabetic_retinopathy'),
    'cataract': os.path.join(img_dir, 'cataract'),
    'normal': os.path.join(img_dir, 'normal'),
    'glaucoma': os.path.join(img_dir, 'glaucoma'),
}
print(class_folders)
# Gather image file paths
img_files = []
img_labels = []
image_counts = defaultdict(int)

for label, folder in class_folders.items():
    files = list(Path(folder).glob('**/*.jpg')) + list(Path(folder).glob('**/*.png')) +
list(Path(folder).glob('**/*.jpeg'))
    img_files.extend(files)
    img_labels.extend([label] * len(files))
    image_counts[label] = len(files)

# Convert to string format
img_files = [str(file) for file in img_files]

# Step 4: Print counts and show one image from each class
print("Image counts in each class:")
for label, count in image_counts.items():
    print(f"{label}: {count} images"
# Display one image from each class
plt.figure(figsize=(10, 8))
for i, (label, folder) in enumerate(class_folders.items()):
    files = list(Path(folder).glob('**/*.jpg')) + list(Path(folder).glob('**/*.png')) +
list(Path(folder).glob('**/*.jpeg'))
    if files:  # Check if there are any files in the folder
        img_path = str(files[0])  # Get one image path
        img = plt.imread(img_path)
        plt.subplot(2, 2, i + 1)  # Create a subplot
        plt.imshow(img)
        plt.title(f"{label} ({count})")  # Show the class name and count
        plt.axis('off')

plt.tight_layout()
plt.show()

# Step 5: Preprocess images
IMG_WIDTH, IMG_HEIGHT = 299, 299  # Size for InceptionResNetV2
preprocessed_imgs = preprocess_images(img_files, IMG_WIDTH, IMG_HEIGHT)

# Data augmentation code...
datagen = ImageDataGenerator(rotation_range=5, width_shift_range=0.05, height_shift_range=0.05)

augmented_images = []
augmented_labels = []

for img_file, label in zip(preprocessed_imgs, img_labels):
    img = cv2.imread(img_file)
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB
    img = np.expand_dims(img, 0)  # Add a batch dimension

    for _ in range(3):  # 3 augmentations per image
        for aug_img in datagen.flow(img, batch_size=1):
            augmented_images.append(aug_img[0].astype('uint8'))  # Convert back to uint8
            augmented_labels.append(label)
            break

# Convert to NumPy arrays for verification
augmented_images_array = np.array(augmented_images)
augmented_labels_array = np.array(augmented_labels)

# Shape checking
print("Augmented Images shape:", augmented_images_array.shape)
print("Augmented Labels shape:", augmented_labels_array.shape)

# Match count check
if augmented_images_array.shape[0] != augmented_labels_array.shape[0]:
    print("Mismatch in number of images and labels!")
else:
    print("Number of images and labels match:", augmented_images_array.shape[0])
```

```
X_train, X_val, y_train, y_val = train_test_split(augmented_images, augmented_labels, test_size=0.2,
random_state=42, stratify=augmented_labels)

# Step 8: Encode labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)

# Step 9: Define and compile model
base_model = InceptionResNetV2(weights='imagenet', include_top=False, input_shape=(IMG_WIDTH,
IMG_HEIGHT, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(len(class_folders), activation='softmax')(x)  # Output layer for classification (4
classes)
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
print("Augmented Images shape:", augmented_images.shape if hasattr(augmented_images, 'shape') else
len(augmented_images))
print("Augmented Labels shape:", augmented_labels.shape if hasattr(augmented_labels, 'shape') else
len(augmented_labels))

checkpoint = callbacks.ModelCheckpoint('eyesmodel.keras', monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=8, mode='min')

# Step 11: Train the model
history = model.fit(
    np.array(X_train),
    y_train_encoded,
    validation_data=(np.array(X_val), y_val_encoded),
    epochs=20,
    batch_size=32,
    callbacks=[early_stopping, checkpoint]
)

# After training, you can save the model and any other necessary data for future use.
model.save('final_model.keras')

# Show Model Performance
tr_acc = history.history['accuracy']
tr_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
```

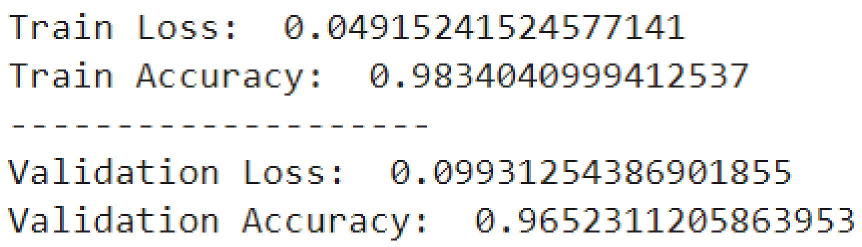
```
Epochs = [i + 1 for i in range(len(tr_acc))]
loss_label = f'best epoch= {str(index_loss + 1)}'
acc_label = f'best epoch= {str(index_acc + 1)}'

# Plot training history
plt.figure(figsize=(20, 8))
plt.style.use('fivethirtyeight')

plt.subplot(1, 2, 1)
plt.plot(Epochs, tr_loss, 'pink', label='Training loss')
plt.plot(Epochs, val_loss, 'green', label='Validation loss')
plt.scatter(index_loss + 1, val_lowest, s=150, c='blue', label=loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(Epochs, tr_acc, 'pink', label='Training Accuracy')
plt.plot(Epochs, val_acc, 'green', label='Validation Accuracy')
plt.scatter(index_acc + 1, acc_highest, s=150, c='blue', label=acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

# Evaluate The Model
train_score = model.evaluate(np.array(X_train), y_train_encoded, verbose=1)
valid_score = model.evaluate(np.array(X_val), y_val_encoded, verbose=1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)

# Make predictions on validation set
y_pred = model.predict(np.array(X_val))
y_pred_classes = np.argmax(y_pred, axis=1)

print(classification_report(y_val_encoded, y_pred_classes, target_names=label_encoder.classes_))

def display_conf_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    # Create a heatmap using seaborn
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)

    # Set the labels and title
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')

    # Display the plot
    plt.show()

display_conf_matrix(y_val_encoded, y_pred_classes)
```

8.2 APPENDIX –2 SCREENSHOTS



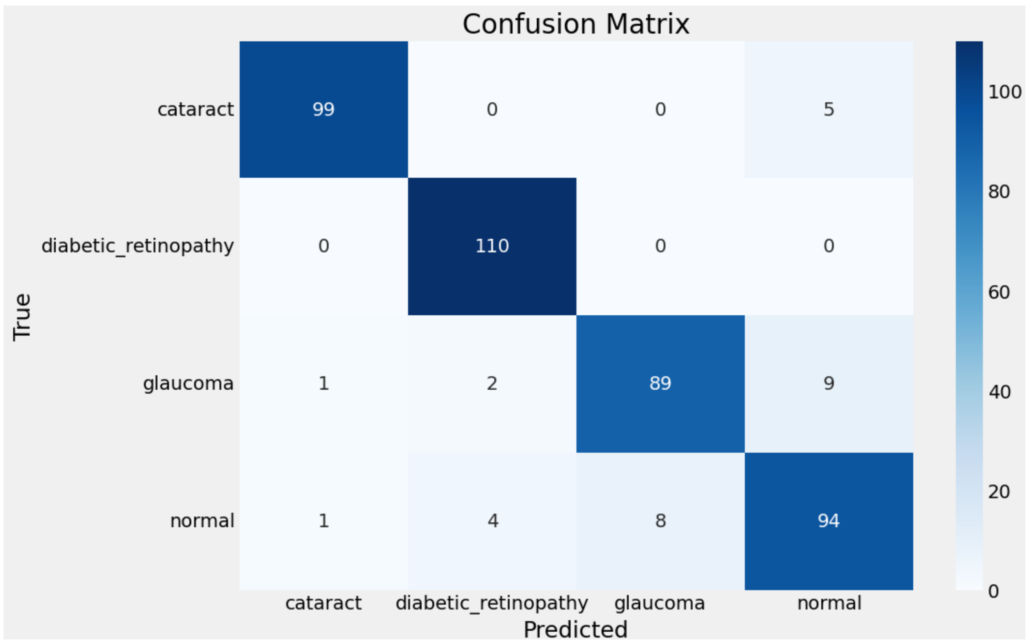
```
Train Loss:  0.20744673907756805
Train Accuracy:  0.9531574249267578
--------------------
Validation Loss:  0.4107201397418976
Validation Accuracy:  0.900473952293396
```

FigA2.1: Accuracy and loss of
InceptionResNetV2 without augmentation

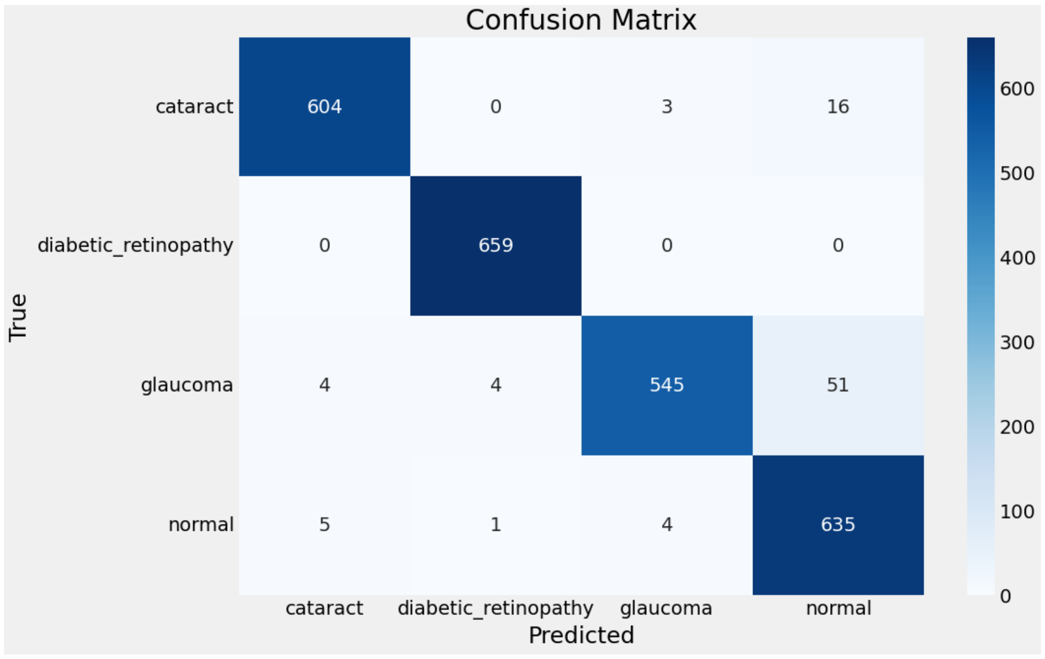


```
Train Loss:  0.04915241524577141
Train Accuracy:  0.9834040999412537
--------------------
Validation Loss:  0.09931254386901855
Validation Accuracy:  0.9652311205863953
```

FigA2.2: Accuracy and loss of
InceptionResNetV2 with augmentation



FigA2.3: Confusion matrix of
InceptionResNetV2 without augmentation



FigA2.4: Confusion matrix of
InceptionResNetV2 with augmentation

REFERENCE

[1] Prasad, K., Sajith, P. S., Neema, M., Madhu, L., & Priya, P. N. (2019, October). Multiple eye disease detection using Deep Neural Network. In TENCON 2019-2019 IEEE Region 10 Conference (TENCON) (pp. 2148-2153). IEEE.

[2] Muthukannan, P. (2022). Optimized convolution neural network based multiple eye disease detection. Computers in Biology and Medicine, 146, 105648.

[3] Aamir, M., Irfan, M., Ali, T., Ali, G., Shaf, A., Al-Beshri, A., ... & Mahnashi, M. H. (2020). An adoptive threshold-based multi-level deep convolutional neural network for glaucoma eye disease detection and classification. Diagnostics, 10(8), 602.

[4] Bali, A., & Mansotra, V. (2024). Analysis of deep learning techniques for prediction of eye diseases: A systematic review. Archives of Computational Methods in Engineering, 31(1), 487-520.

[5] Sun, K., He, M., Xu, Y., Wu, Q., He, Z., Li, W., ... & Pi, X. (2022). Multi-label classification of fundus images with graph convolutional network and LightGBM. Computers in Biology and Medicine, 149, 105909.

[6] Shoaib, M. R., Emara, H. M., Zhao, J., El-Shafai, W., Soliman, N. F., Mubarak, A. S., ... & Esmaiel, H. (2024). Deep learning innovations in diagnosing diabetic retinopathy: The potential of transfer learning and the DiaCNN model. Computers in Biology and Medicine, 169, 107834.

[7] Bhati, A., Gour, N., Khanna, P., & Ojha, A. (2023). Discriminative kernel convolution network for multi-label ophthalmic disease detection on imbalanced fundus image dataset. Computers in Biology and Medicine, 153, 106519.

[8] Kumar, Y., & Gupta, S. (2023). Deep transfer learning approaches to predict glaucoma, cataract, choroidal neovascularization, diabetic macular edema, drusen and healthy eyes: an experimental review. Archives of Computational Methods in Engineering, 30(1), 521-541.

[9] You, A., Kim, J. K., Ryu, I. H., & Yoo, T. K. (2022). Application of generative adversarial networks (GAN) for ophthalmology image domains: a survey. Eye and Vision, 9(1), 6.

[10] Chellaswamy, C., Geetha, T. S., Ramasubramanian, B., Abirami, R., Archana, B., & Bharathi, A. D. (2022, May). Optimized convolutional neural network based multiple eye disease detection and information sharing system. In 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 1105-1113). IEEE.