A Project Report on

Diabetic Retinopathy Detection using Deep Learning

Submitted in partial fulfilment of the requirements for the award of degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

JYOSHNA KANDI (160118733130)

LAHARI MADISHETTY (160118733131)

Department of Computer Science and Engineering,

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)

(Affiliated to Osmania University; Accredited by NBA(AICTE) and NAAC(UGC), ISO Certified 9001:2015)

GANDIPET, HYDERABAD – 500 075

Website: www.cbit.ac.in

[2021-2022]

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
Kokapet (Village), Gandipet, Hyderabad, Telangana-500075. www.cbit.ac.in

COMMITTED TO RESEARCH, INNOVATION AND EDUCATION 43 years

CERTIFICATE

This is to certify that the project titled **"Diabetic Retinopathy Detection using Deep Learning"** is the bonafide work carried out by **JYOSHNA KANDI (160118733130) and LAHARI MADISHETTY (160118733131)** ,the students of B.E.(CSE) of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2021-2022, submitted in partial fulfilment of the requirements for the award of the degree in **Bachelor of Engineering** (**Computer Science and Engineering**) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Supervisor	**Head,CSE Dept.**
Dr.B.Bhargavi	**Dr.Y.Ramadevi**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
CBIT,Hyderabad	CBIT,Hyderabad

Place : Hyderabad

Date :

DECLARATION

We hereby declare that the project entitled "**Diabetic Retinopathy Detection using Deep Learning**" submitted for the B.E (CSE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship, or any other similar titles.

Name(s) and Signature(s) of the Student

JYOSHNA KANDI
(160118733130)

LAHARI MADISHETTY
(160118733131)

Place: Hyderabad
Date:

ABSTRACT

Diabetic Retinopathy (DR) is a human eye disease that affects people who have diabetes and damages their retina, potentially leading to blindness. DR has been manually screened by an ophthalmologist until recently, which is a time-consuming technique. The goal is to use technology to scale their efforts and achieve the ability to automatically screen photographs for disease and provide information on the severity of the issue. This will be accomplished by developing a Convolutional neural network model that can automatically analyse a patient's ocular picture and determine the severity of blindness.This automation technique can save a lot of time, which can be used to screen the process of treating diabetic retinopathy on a wide scale. As a result, this research will focus on the analysis of various DR stages using Deep Learning (DL), which is a subset of Artificial Intelligence (AI).

We will train a neural network model using a large dataset of 3662 train photos to automatically recognise the DR stage, which will be categorised into high-resolution fundus images in this research. On Kaggle, you can find the Dataset [9] that we're utilising (APTOS). There are five different DR stages: 0, 1, 2, 3, and 4. The input parameters for this project are the patient's fundus eye photographs. The features of fundus images of the eye will be extracted by a trained model, and the activation function will provide the output.

Preprocessing, Data Augmentation, Network Architecture Building, and the training process, which comprises multiple parts such as pretraining, maintenance, posttraining, regularisation, and ensembling, are all part of the project's implementation. The CNN model is constructed using an ensemble of ResNet (residual neural network) and Xception architecture, and after training, the model has an accuracy of 0.72. The trained model is then put to the test on a variety of retinal pictures to determine the DR diagnosis level.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our Project Guide, **Dr. Bhargavi B** for her valuable guidance and constant support along with her capable instructions and persistent encouragement.

We also would like to express our gratitude towards our Project Coordinators, **Dr. S. China Ramu**, and **Smt. T. Suvarna Kumari** for standing with us whenever we needed their support.

We are grateful to our Head of the Department **Dr. Y. Ramadevi**, for her steady support and the provision of every resource required for the completion of the project.

We would like to take this opportunity to thank our Principal **Dr. P. Ravinder Reddy**, as well as the management of the institution for having designed an excellent learning atmosphere.

LIST OF FIGURES

LIST OF TABLES

Table of Contents

CHAPTER 1
INTRODUCTION

Deep learning is a type of machine learning that uses artificial neural networks exclusively, which are designed to imitate the human brain. As neural networks are designed to mimic the human brain, deep learning is likewise a kind of human brain mimic. We don't need to explicitly program anything in deep learning. In our project we use deep learning techniques to build our CNN model and to train the model to achieve accurate results.

Diabetic retinopathy, an eye condition induced by diabetes, was the leading cause of blindness in America, accounting for nearly 99 percent of cases in India, according to researchers. India and China now have over 90 million diabetic patients and are on the cusp of a diabetic population explosion. Unless diabetic retinopathy can be discovered early, this could result in an unprecedented number of people being blind. The automated diabetic retinopathy problem was a demanding computer vision problem whose purpose was to recognise retinopathy features in retinal color fundus images, such as hemorrhages and exudates.

1.1 Problem Definition

Diabetic retinopathy (DR) is one of the most serious complications of diabetes, causing damage to the retina and eventually blindness. The ability to automatically detect diabetic retinopathy in photographs and provide information on the severity of the condition. This will be accomplished by developing a convolutional neural network model that can automatically analyze a patient's ocular picture and determine the severity of blindness. This automation technique can save a lot of time. This can be used to monitor the progress of diabetic retinopathy treatment on a large scale. The importance of this project is that it assists doctors in initiating diabetic retinopathy treatment at the appropriate time and in diagnosing the disease in its early stages.

The main objective of this project is to create a Convolutional Neural Network model that will help us recognise and classify the stage of Diabetic Retinopathy that a patient is experiencing, as well as the severity of the condition.

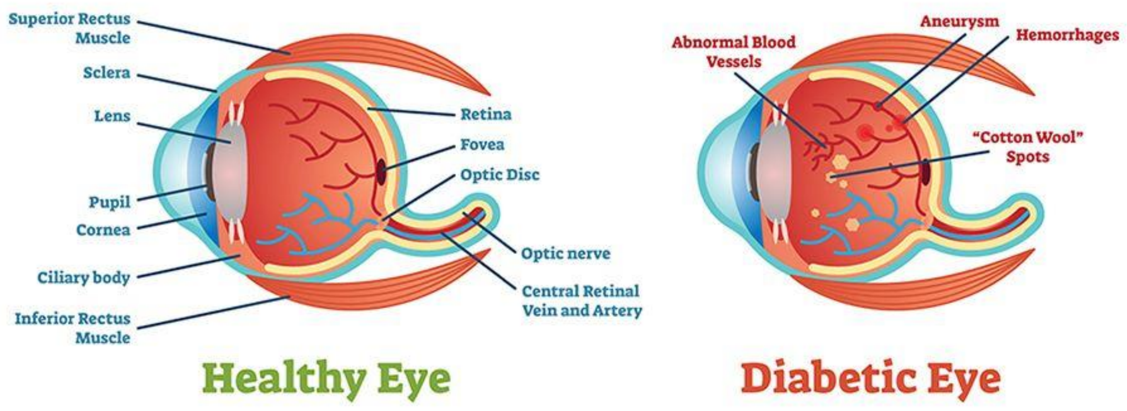


Figure 1.1: Comparison of Healthy Eye and Diabetic Eye

Figure 1.1 shows the difference between a healthy eye and diabetic eye. It also indicates the damaged area and the variables involved, such as the formation of aneurysms, hemorrhages, cotton wool patches, and abnormal blood vessel growth.

1.2 Methodologies

As we all know, a problem with DR detection is a leading cause of blindness. Early detection is the most important step in resolving this issue.

As a result, we're using the "ResNet Architecture," a deep learning architecture, for early detection.

The foundation for Diabetic Retinopathy deep learning includes:

- **Preprocessing:** For model training and validation, preprocessed copies of the source photographs will be used. The first steps in the preprocessing process are image cropping and scaling.
- **Data Augmentation:** A set of strategies for artificially increasing the amount of data by producing additional data points from existing data is known as data augmentation. Making modest adjustments to data or utilizing deep learning models to produce additional data points are examples of this.
- **Network architecture:** The goal of the network design is to appropriately characterize each fundus picture. Our neural networks

will be built using traditional deep CNN architecture, which includes a feature extractor and a smaller decoder for a specific task (head). To diagnose diabetic retinopathy, we will propose a multi-task learning technique. Three decoders are used. The classification head, regression head, and ordinal regression head are all trained to solve their tasks using features extracted with the CNN backbone.

- **Training process:** We employ a multi-stage training procedure that includes pre-training, main training, and post-training settings.
- **Testing process**: We would be testing the model with various retina images.

1.3 Outline of the Results

The severity level of diabetic retinopathy for the supplied input fundus image of the patient's eye is one of the outcomes. The final trained model has resulted in an accuracy of 0.72. The train and validation losses are decreasing and accuracies are increasing.

1.4 Scope of the project

Hospital doctors can use this model to assess the severity of Diabetic Retinopathy. This approach is more accurate than the traditional method of manually diagnosing the stage of the disease using fundus camera images.

1.5 Organization of the report

This project report is organized as follows:

Chapter 1 includes the introduction to the problem and the scope of the project. In chapter 2, we discuss the literature survey of the project which includes an insight into the core part of the project and this chapter also includes the technologies used across the project. In Chapter 3, which is methodology of the project we discuss the design of the proposed system. Chapter 4 includes the implementation of a proposed system for diabetic retinopathy detection using deep learning techniques. We discuss the results in Chapter 5 along with the series of screenshots obtained after implementing the model. In the final chapter, we talk about the conclusions, limitations and the future scope of the project.

CHAPTER 2
LITERATURE SURVEY

2.1 Introduction to the problem domain terminology

Using convolutional neural networks and deep learning techniques,this project creates a model that allows us to identify and classify Diabetic Retinopathy into five different classes.

Deep learning

Deep learning is a machine learning technique that allows computers to learn by example in the same way that humans do. Deep learning is a critical component of self-driving automobiles, allowing them to detect a stop sign or discriminate between a pedestrian and a lamppost. It enables voice control in consumer electronics such as phones, tablets, televisions, and hands-free speakers. Deep learning has gotten a lot of press recently, and with good cause. It's accomplishing accomplishments that were previously unattainable. A computer model learns to execute categorization tasks directly from images, text, or sound in deep learning. Deep learning models can achieve state-of-the-art accuracy, even surpassing human performance in some cases. Models are trained utilizing a huge quantity of labeled data and multilayer neural network topologies.[5].

Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning system that can take an input image and assign relevance (learnable weights and biases) to various aspects/objects in the image, as well as differentiate between them. The amount of pre-processing required by a ConvNet is much less than that required by other classification methods. CNN architectures are available in a wide range of shapes and sizes. (Refer **Table 2.1.1**).

Table 2.1: Types of CNN Architectures

Architecture Name	**No.of layers**	**Activation function**	**Dataset**	**Limitations**
LeNet(1989) [2]	7	Softmax classifier	MNIST	LeNet was not designed to work on large images.
AlexNet(2012) [4]	8	ReLU [3]	CIFAR-10	The depth of this model is very less and hence it struggles to learn features from image sets.
VGG-16(2014) [4]	16	ReLU	ImageNet [3]	Vanishing-Gradient Problem
GoogleNet(2014) [4]	22	ReLU	ImageNet	If a network is built with many deep layers it might face the problem of overfitting.
ResNet(2015) [6]	56	ReLU	ImageNet	A deeper network usually requires weeks for training

Description of the above types of CNN architecture table:

LeNet

Several banks used LeNet-5, a pioneering 7-level convolutional network that classifies digits developed by LeCun et al[2] in 1998, to recognise hand-written numbers on checks (cheques) scanned in 32x32 pixel grayscale input images. Because processing higher-resolution images necessitates larger and more convolutional layers, this technique is limited by the computational resources available. The network is called Lenet-5 since it contains five layers with learnable parameters. It uses a combination of average pooling and three sets of convolution layers. We have two completely connected layers after the convolution and average pooling layers. Finally, there's a Softmax classifier that divides the photos into categories.

AlexNet

AlexNet is made up of eight layers, each with its own set of learnable parameters[4]. The model comprises five layers, each of which uses Relu activation, with the exception of the output layer, which uses a combination of max pooling and three fully connected layers. They discovered that employing the relu as an activation function increased the training process' speed by nearly six times. They also used dropout layers to prevent overfitting in their model. In addition, the Imagenet dataset is used to train the model. The Imagenet collection contains about 14 million photos divided into a thousand categories.

VGG-16

The network's input is a two-dimensional image (224, 224, 3). The first two layers have the same padding[4] and have 64 channels of 3*3 filter size. Then, after a stride (2, 2) max pool layer, two layers of convolution layers with 256 filter size and filter size (3, 3). This is followed by a stride (2, 2) max-pooling layer, which is the same as the preceding layer. Then there are 256 filters and two convolution layers with filter sizes of 3 and 3. Then there are two sets of three convolution layers, as well as a max pool layer. Each filter has 512 filters of the same size (3, 3) and the same padding. This image is then fed into a two-layer convolution stack. We use 3*3 filters in these convolution and max-pooling layers, rather than 11*11 in AlexNet and 7*7 in ZF-Net. It also employs 1*1 pixels in some of the layers to adjust the amount of input channels. After each convolution layer, a 1-pixel padding (same padding) is applied to avoid the image's spatial information from being lost.

GoogleNet

In 2014, Google Net (or Inception V1) was proposed by Google researchers (with the help of other universities) in the research article "Going Deeper with Convolutions"[4]. The ILSVRC 2014 image classification competition was won by this architecture. In comparison to previous champions AlexNet (ILSVRC 2012) and ZF-Net (ILSVRC 2013), it has a much lower error rate, and it has a significantly lower error rate than VGG (2014 runner up). Techniques like 11 convolutions in the middle of the architecture and global average pooling are used in this architecture.

ResNet

Every succeeding winning architecture after the initial CNN-based architecture (AlexNet) that won the ImageNet 2012 competition employs more layers in a deep neural network to minimize the error rate[6]. This works for less layers, but as the number of layers grows, we run into a problem called Vanishing/Exploding gradient, which is a typical difficulty in deep learning. As a result, the gradient becomes 0 or too huge. As a result, as the number of layers increases, so does the training and test error rate.

Xception

The Google product Xception, which stands for Extreme version of Inception, is reviewed[14]. For both the ImageNet ILSVRC and JFT datasets, it outperforms Inception-v3 (also by Google, 1st Runner Up in ILSVRC 2015) with a modified depthwise separable convolution.

2.2. Background

The biggest problem that DR patients encounter is that they are unaware of the condition until the alterations in the retina have progressed to the point that treatment is less effective. Automated screening systems for detection have a significant impact on cost, time, and labor savings. Diabetic individuals who are screened for the development of diabetic retinopathy have a 50% lower chance of blindness. Because the number of ophthalmologists is insufficient to treat all patients, particularly in rural areas or when the workload of local ophthalmologists is considerable, automated approaches are becoming increasingly crucial as the number of people affected by the condition grows. As a consequence, automated early diagnosis could reduce the severity of the condition and let ophthalmologists investigate and treat it more effectively. Exudates, hemorrhage, and microaneurysm are the most prevalent abnormal lesions associated with diabetic retinopathy.

A few studies employing diverse approaches for Diabetic Retinopathy have been published in the literature. The use of automation methods based on fundus imaging for DR has recently gotten a lot of interest from researchers. Here's a quick rundown of some recent research findings.

According to previous research, providing a variety of automated algorithms for detecting anomalies in fundus images is crucial [1]. These studies intended to develop their automated bleeding detection system to help diagnose diabetic retinopathy. They discovered a new preprocessing and false-positive removal approach in this investigation. The brightness of the fundus image was altered by a nonlinear curve with brightness values from the hue saturation value (HSV) space. To emphasize brown parts, gamma correction was applied to each red, green, and blue-bit picture. As a result, the histograms of each red, blue, and blue-bit image were stretched. After that, the bleeding candidates were determined. The brown patches indicated hemorrhages and blood vessels, which were identified using density analysis. The major candidates, such as blood vessels, were eliminated. Finally, a 45-feature analysis was used to eliminate erroneous positives. They looked at 125 fundus images, 35 of which had hemorrhages and 90 of which were normal, to see how well the novel method for detecting hemorrhages worked. The detection of aberrant instances had a sensitivity and specificity of 80 percent and 88 percent, respectively. These findings suggest that the novel method could significantly improve the performance of their bleeding computer-aided diagnosis system.

2.3. Existing Solutions and Related Works

Existing diagnostic approaches are inefficient because they take a long time, causing treatment to go in the incorrect direction. To diagnose retinopathy, doctors use a fundus camera, which takes photos of veins and nerves behind the retina. Because there are no symptoms of DR in the early stages of the disease, it may be difficult to detect it.

The issue of early detection of diabetic retinopathy has prompted a lot of research. First and foremost, the problem was solved using traditional computer vision and machine learning methods. They retrieved features from raw image processing operations and fed them to the SVM for binary classification on a testing set of 250 pictures, reaching a sensitivity of 98 percent, specificity of 96 percent, and accuracy of 97.6%.

Additional researchers used a dataset of 151 images with various resolutions to train other models for multiclass classification, such as applying PCA to images and fitting decision trees, naive Bayes, or k-NN with best results of 73.4 percent accuracy and 68.4 percent for F-measure.

Other studies have looked into a number of ways that use CNNs to overcome this problem. They developed a network with CNN architecture and data augmentation that can detect minor aspects in the classification task, such as micro-aneurysms, exudates, and hemorrhages in the retina, and provide an automatic diagnosis without human intervention. They achieved a sensitivity of 95 percent and an accuracy of 75 percent on 5,000 validation photographs. [7].

In an existing system which performed analysis in different ways using CNN architectures for diabetic retinopathy detection. The first way is implementation of VGG16 without using ImageNet and QWK. The second way is implementation of DenseNet with ImageNet and QWK. So, without ImageNet VGG16 gave less accuracy which is 0.73 and with ImageNet DenseNet gave 0.96 accuracy value which is better than VGG16 [1].

2.4 Tools/Technologies used

2.4.1 Hardware Requirements:

Laptop/Desktop with at least 4GB RAM

2.4.2 Software Requirements:

- Basic libraries like numpy, pandas, matplotlib - To handle the dataset and to perform operations required.
- Tensorflow/Keras - To write the neural network models.
- Google Colab - Interface to run the code
- Operating system - Windows 10; 64-bit
- Python shell - To implement the GUI

Python:

Python is an interpreted high-level computer language for general-purpose programming. Guido van Rossum designed Python, which was first released in 1991.

Its design philosophy prioritizes code readability and incorporates a large amount of whitespace. At both small and big scales, it provides structures that allow for clear programming. Van Rossum stepped down in July 2018 after 30 years as the language community's leader.

Keras library:

Keras is a deep learning API written in Python that works on top of the TensorFlow machine learning framework. It was made with the intention of facilitating speedy experimentation. When conducting research, it's critical to be able to move rapidly from concept to conclusion.

Keras is:

- **Simple** -- It's simple, but it's not simplistic. Keras relieves developer cognitive load, allowing you to focus on the most important aspects of the problem.
- **Flexible** --Simple processes should be quick and easy, whereas arbitrarily advanced workflows should be feasible via a clear path that builds on what you've already learned, according to the Keras principle of progressive disclosure of complexity.
- **Powerful** -- Keras provides industry-leading performance and scalability, and it is utilized by NASA, YouTube, and Waymo, among others.

Tensorflow:

TensorFlow is an open-source machine learning platform that automates the entire process. It's a kind of foundation layer for differentiable programming. It combines four critical abilities:

- Using the CPU, GPU, or TPU to efficiently perform low-level tensor operations.
- The gradient of arbitrary differentiable expressions is computed.
- Scaling computing to a large number of devices, such as hundreds of GPUs in a cluster.
- External runtimes, such as servers, browsers, mobile devices, and embedded devices, are used to export programmes (graphs).

Keras is a user-friendly, high-productivity interface for tackling machine learning difficulties in TensorFlow's high-level API, with a focus on current deep learning. It provides fundamental abstractions and building elements for designing and releasing high-iteration-rate machine learning systems.

Engineers and researchers can utilize Keras to fully use TensorFlow's scalability and cross-platform capabilities: Keras can run on TPU or huge GPU clusters, and Keras models can be exported to run on the web or on a mobile device.

Numpy:
NumPy is a library that consists of multidimensional array objects and a set of algorithms for manipulating them. NumPy is a Python library that allows you to perform mathematical and logical operations on arrays. Python scripting language NumPy. Its full name is 'Numerical Python.'

Pandas:
Pandas is a popular open-source Python program for data science, data analysis, and machine learning activities.
It is based on Numpy, a multi-dimensional arrays-supporting library.

Matplotlib:

For 2D array charts, Matplotlib is an excellent Python visualization library. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the entire SciPy stack. One of the most significant advantages of visualization is that it provides us with visual access to massive volumes of data in simply understandable graphics. Matplotlib includes a variety of plots, including line, bar, scatter, histogram, and more.

CHAPTER 3
DESIGN OF THE PROPOSED SYSTEM

3.1 Block Diagram

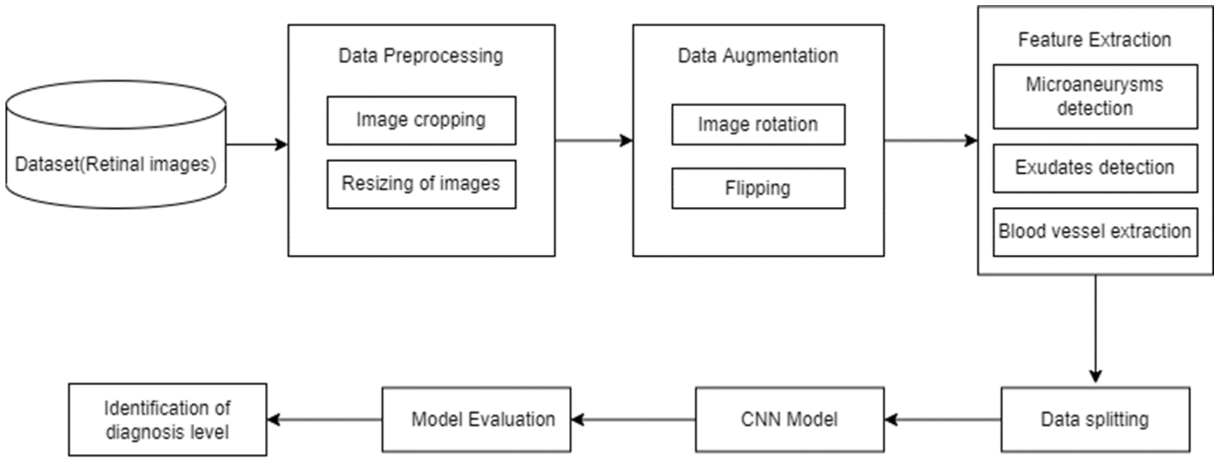


Figure 3.1 : Block diagram of the proposed system

Figure 3.1 depicts the proposed system's block diagram, and paragraph 3.2 provides a thorough description of the modules shown in the diagram.

3.2 Module Description

The implementation of the proposed system consists of several steps (Refer **Figure 3.1**) which are described as follows.

3.2.1 Dataset (Fundus/Retinal images) :

We define fundus imaging as the process of employing reflected light to create a two-dimensional (2D) representation of the three-dimensional (3D) semi-transparent retinal tissues projected onto the imaging plane.

3.2.2 Data Preprocessing :

Data preprocessing is the process of transforming raw data into a format that can be understood.

The following are the steps involved in Data Preprocessing:

1. Take an image as an input.
2. Use preprocessing techniques to bring out the most important details.
3. Cropping and resizing of the image.
4. Proper data cleaning and removing black images.

3.2.3 Data Augmentation :

Data augmentation is a term used in data analysis to describe methods for enhancing the quantity of data available by adding slightly modified copies of existing data or developing new synthetic data from existing data. It acts as a regularizer and helps to avoid overfitting when training a machine learning model.

The following are the steps involved in Data Augmentation:

1. Take the preprocessed images.
2. Rotation, Flipping and mirroring of images is to be done to balance the dataset[9] if the dataset is imbalanced.
3. Now use the images for training or testing.

3.2.4 Feature Extraction :

3.2.4.1 Microaneurysms Detection :

Microaneurysms are saccular outpouchings of capillary walls that can leak fluid, causing intraretinal edema and hemorrhages. Because microaneurysms are the first clinically noticeable indicator of diabetic nonproliferative eye illness, recognising them can be the first step in secondary prevention of diabetic retinopathy progressing to the proliferative stage and resulting in severe vision loss.

3.2.4.2 Exudates Detection :

The development of exudates on the retina is the most prevalent sign of diabetic retinopathy. Exudates are one of the first clinical indications of DR, thus detecting them would be a useful addition to the mass screening assignment, as well as a necessary step toward automatic disease grading and monitoring.

3.2.4.3 Blood vessel Extraction:

Diabetic retinopathy[1], glaucoma, arteriosclerosis, and hypertension are all diseases with retinal blood vessels that have a role in diagnosis and treatment. As a result, extracting the retinal vasculature is essential for supporting experts in the diagnosis and treatment of systemic illnesses.

3.2.5 Data Splitting :

In machine learning, data splitting is widely used to divide data into train, test, and validation sets.
In this case, we divided the train images into 20% validation images and 80% training images.

3.2.6 CNN Model :

Here we build the CNN model that is the ResNet model for our project and then we train the model. The architecture of the ResNet model consists of various modules (Refer **Figure 3.2**) and layers which include pooling layers, dense layers and dropout layers.

3.2.7 Model Evaluation :

We evaluate our CNN model and calculate the performance metrics like accuracy, loss (Refer Fig 5.1).

3.2.8 Identification of Diagnosis Level :

We test our trained CNN model using the test images and get the diagnosis level of DR as output results.

3.3 Architecture

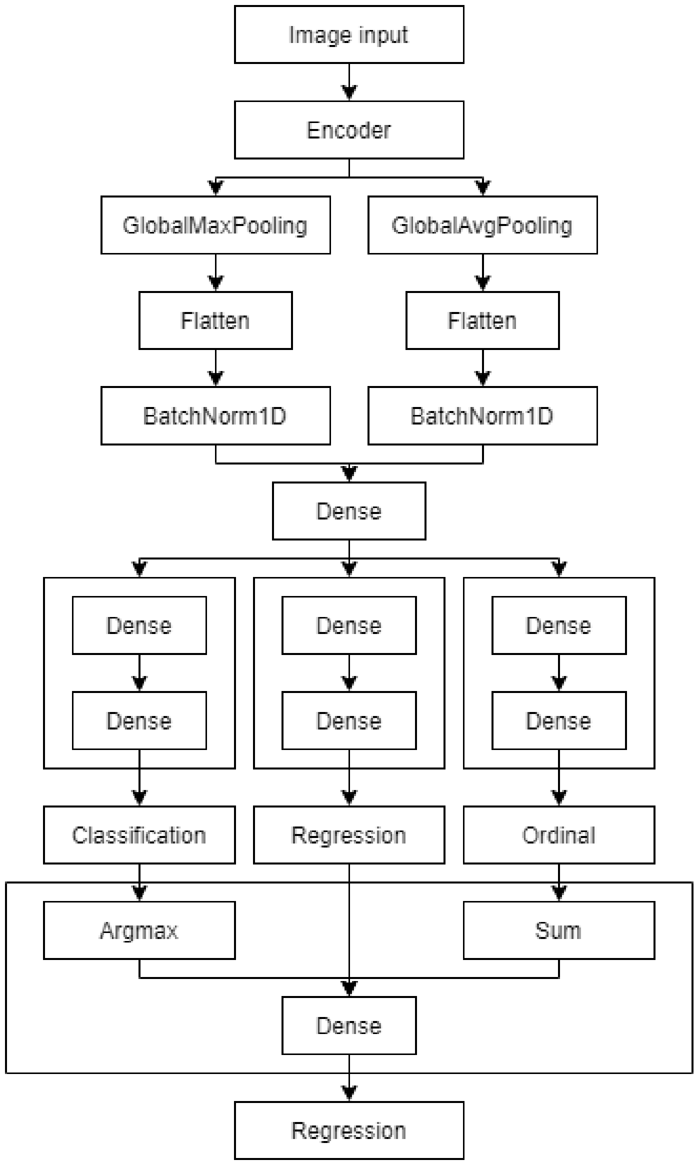


Figure 3.2: Three-head CNN Architecture[16]

Figure 3.2 represents the three head CNN architecture in which we focus on the classification part of the architecture. It consists of various modules which are described as follows

GlobalAvgPooling:

Global Average Pooling is a pooling mechanism used to replace completely connected layers in classical CNNs. Rather than establishing fully linked layers on top of the feature maps, we take the average of each feature map and feed the resulting vector directly into the softmax layer [10].

GlobalMaxPooling:

Max pooling for 2D spatial data is implemented via the MaxPooling2D class. By getting the greatest value for each channel of the input over an input window, downsamples the input along its spatial dimensions (height and breadth) (of size determined by pool size). Strides [10] are used to alter the window's dimensions.

BatchNorm1D:

Batch normalization [7] is a method for training very deep neural networks in which each mini-input batch is normalized to a layer. This stabilizes the learning process and significantly decreases the number of training epochs needed to build deep networks [11].

Dense:

In a Dense layer, all of the preceding layer's outputs are given to all of the neurons, with each neuron giving one output to the next layer. This is the most basic layer in neural networks.

Dropout:

The Dropout layer, which helps to reduce overfitting, randomly sets input units to 0 at a rate frequency at each step during the training period. The dropout layer uses a simple but effective approach to improve the generalization ability of the neural network. During training, neurons are randomly deleted and restored with a probability determined by the dropout rate, which is specified by the hyperparameter.

Classification head:

The classification head generates a one-hot encoded vector, with the existence of each stage denoted by 1.

Regression head:

The regression head produces a real value between 0 and 4.5, which is then rounded to an integer that represents the disease stage.

Ordinal Regression head:

If a data point belongs to category k, it immediately belongs to all of the categories from 0 to k 1. As a result, the goal of this head is to anticipate all categories up to and including the target. Fitting a linear regression model to the outputs of three heads yields the final forecast.

CHAPTER 4
IMPLEMENTATION OF THE PROPOSED SYSTEM

Proposed System :

In this project we will build a CNN model using ResNet architecture and then we will train our model using an inbuilt keras method called train_generator and finally we will calculate the performance metrics. To improve the accuracy of the CNN model we will ensemble the previous model results with VGG architecture and then fine-tune our resulting model using an inbuilt keras method called fit_generator and calculate the metrics again. We will ensemble the ResNet model with Xception architecture and then fine-tune the model again to ensure that the train and validation losses do not increase. We'll calculate the final performance measures, draw the graphs, display the confusion matrix, and test our model with a variety of input photos in the last phase.

4.1 Design Diagrams

4.1.1 Data Flow Diagram

A data-flow diagram is a visual representation of data flowing through a system or a process (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow — there are no decision rules and no loops. The detailed data flow diagram (Refer **Figure 4.3**) of this project includes a sequence of processes like preprocessing, resizing the images, cropping the images, extracting the features, training the model, evaluation of CNN model, testing the model and a data storage of retinal images.

The following diagram depicts the project's detailed flow:

The preprocessing stage receives the user's retinal image as input, now the preprocessed data is stored in a data storage and this data along with the input parameters are fed to the model for training, then the model evaluation will be done which will give performance metrics as the results. In the last step, the trained model will be tested which will give the diagnosis level of DR as the final output.

(Level - 0)

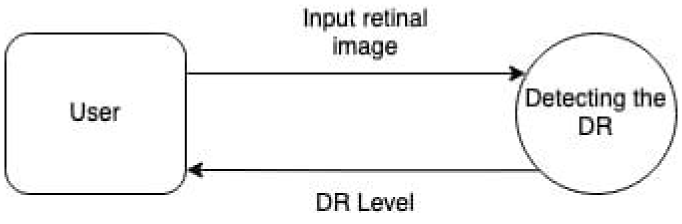


Figure 4.1: Data Flow diagram of the proposed system(Level-0)

Figure 4.1 represents the Level-0 Data Flow Diagram of Diabetic Retinopathy Detection.

(Level - 1)

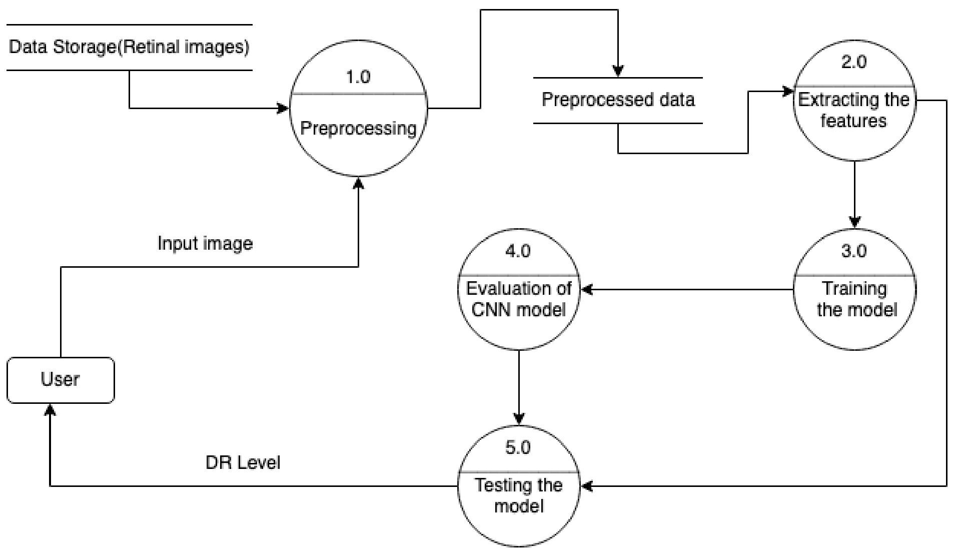


Figure 4.2: Data Flow diagram of the proposed system(Level-1)

Figure 4.2 represents the Level-1 Data Flow Diagram of Diabetic Retinopathy Detection.

(Level - 2)

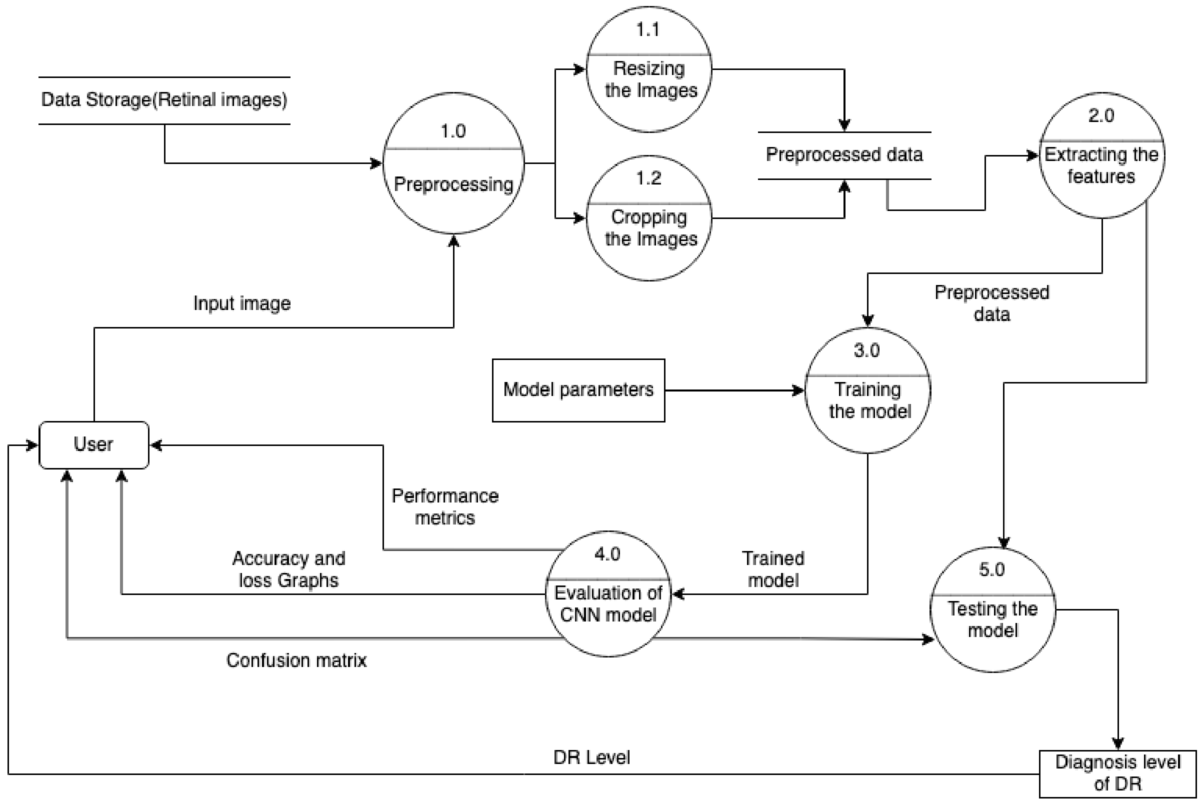


Figure 4.3: Data Flow diagram of the proposed system(Level-2)

Figure 4.3 represents the Level-2 Data Flow Diagram of Diabetic Retinopathy Detection.

4.1.2 Use Case Diagram :

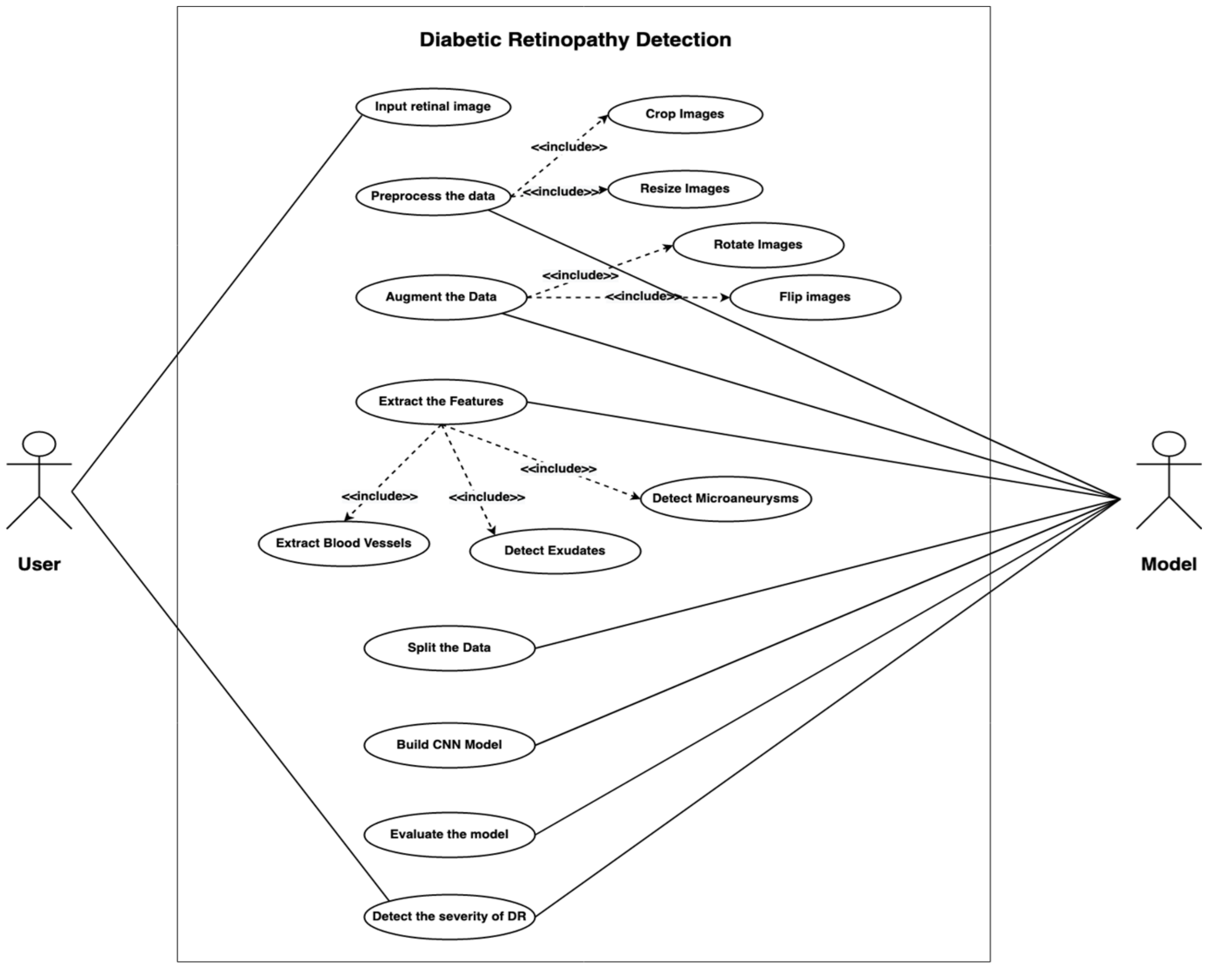


Figure 4.4 : Use Case diagram of the proposed system

A use case diagram is a visual representation of how a user might interact with a technology. A use case diagram depicts the system's numerous use cases and different sorts of users, and is frequently supplemented by other diagrams. Circles or ellipses are used to depict the use cases. The actors are frequently depicted as stick figures. The Diabetic Retinopathy Detection Use Case Diagram is shown in **Figure 4.4**.

4.1.3 Class Diagram :

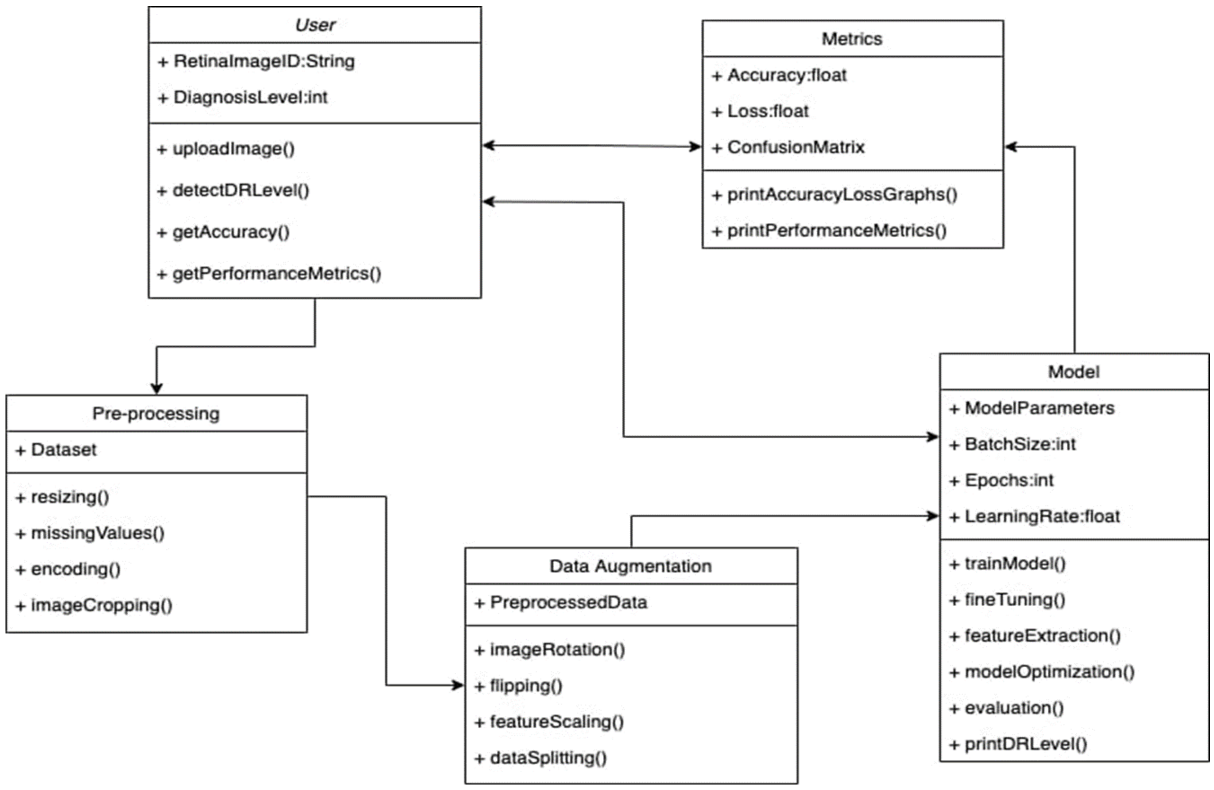
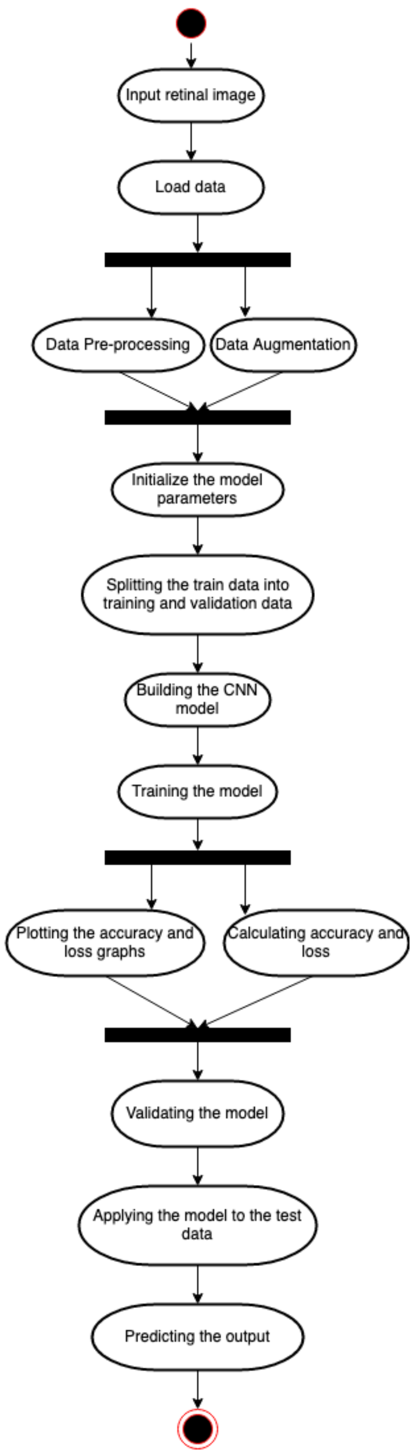


Figure 4.5 : Class diagram of the proposed system

A static diagram is a class diagram. It depicts an application's static view. A class diagram is used not only for visualizing, describing, and documenting many parts of a system, but also for creating executable code for a software programme. A class diagram depicts a class's attributes and operations, as well as the system's limitations. Because class diagrams are the only UML diagrams that can be directly mapped with object-oriented languages, they are frequently used in the modeling of object-oriented systems. The Diabetic Retinopathy Detection Class Diagram is shown in **Figure 4.5**.

4.1.4 Activity Diagram :



Input retinal image
Load data
Data Pre-processing
Data Augmentation
Initialize the model parameters
Splitting the train data into training and validation data
Building the CNN model
Training the model
Plotting the accuracy and loss graphs
Calculating accuracy and loss
Validating the model
Applying the model to the test data
Predicting the output

Figure 4.6 : Activity diagram of the proposed system

The activity diagram is used in UML to show the system's flow of control rather than the implementation. It simulates simultaneous and sequential activity. The activity diagram aids in visualizing the flow of work from one action to the next. It emphasizes the state of flow and the sequence in which it occurs. The flow can be sequential, branching, or concurrent, and the activity diagram provides forks, joins, and other features to deal with these types of flows. The Diabetic Retinopathy Detection Activity Diagram is shown in **Figure 4.6**.

4.2 Dataset Description

We used a dataset that included a large number of high-resolution retina images taken in a variety of imaging conditions. A fundus camera was used to take the shots in the collection, which produces colour fundus images of DR. A fundus camera is a low-power microscope with a camera attached that is used to photograph the inner surface of the eye. Because the fundus image offered a distinct picture that could be detected, it was used to document the DR status. Tensorflow[13] and Keras [12] were used in order to create this project.

Fundus image:
The fundus, or back of the eye, is photographed in fundus photography. Fundus photography is done with fundus cameras, which combine an intricate microscope with a flash-enabled camera.The central and peripheral retina, optic disc, and macula are the primary components visible on a fundus shot. Colored filters or specific dyes like fluorescein and indocyanine green can be used to photograph the fundus.

The clinicians are divided these DR into five classes [8] which show the stages of DR: No DR (class 0), Mild DR (class 1), Moderate DR (class 2), Severe DR (class 3), PDR (Proliferative DR) (class 4)(Refer Fig 4.7). We will be using the dataset which is publicly available on Kaggle i.e., APTOS 2019 Blindness Detection (APTOS2019) dataset. The full dataset consists of fundus photographs, which includes 3662 training images and it is of size 454MB.
Diabetic Retinopathy is divided into stages that allow us determine the severity of the disease and, as a result, develop prevention strategies. Diabetic retinopathy has several stages.

1. **No DR**: The disease has not yet progressed to the point where it will affect the eye. To put it another way, the eye is in excellent condition.
2. **Mild DR**:In the first stage of mild nonproliferative DR, there will be balloon-like swelling in small parts of the blood vessels in the retina.

3. **Moderate DR**: In the second stage, termed as mild nonproliferative retinopathy, some of the blood vessels in the retina become occluded.
4. **Severe DR**: The third stage of DR, severe nonproliferative retinopathy, causes more blood vessels to clog, resulting in inadequate blood flow to portions of the retina. Due to a lack of appropriate blood flow, the retina is unable to produce new blood vessels to replace those that have been destroyed.
5. **Proliferative DR**: Diabetic retinopathy progresses to the fourth and final stage, proliferative retinopathy. This is how far the illness has progressed. New blood vessels will begin to grow in the retina, but they will be fragile and abnormal. They may leak blood as a result, causing vision loss and possibly blindness.

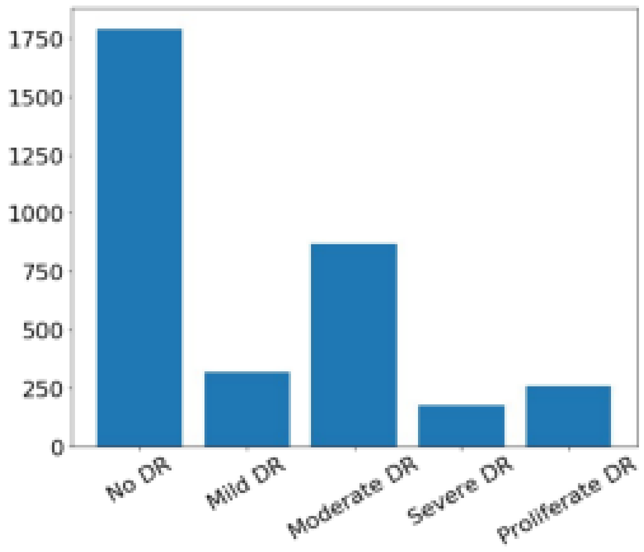


Figure 4.7 : No.of train images in the dataset

Figure 4.7 shows the bar plot representing the distribution of train images in the dataset belonging to each stage of DR.

4.3 Detailed Description of implementation

The detailed source code of the implementation of diabetic retinopathy detection using deep learning can be found in Appendix - 1.

4.3.1 Importing necessary libraries

Keras is a deep learning API written in Python that works on top of the TensorFlow machine learning framework. It was made with the intention of facilitating speedy experimentation. When conducting research, it's critical to be able to move rapidly from concept to conclusion.

- It's straightforward, yet not simplistic. Keras lessens the cognitive load of developers, letting them focus on the most relevant components of the problem.
- Simple workflows should be quick and simple, but more complicated workflows should be possible if you follow a clear route that builds on what you've already learned.
- NASA, YouTube, and Waymo, among others, use Keras, a sophisticated platform with industry-leading performance and scalability.

TensorFlow is an open-source machine learning platform that automates the entire process. It's a kind of foundation layer for differentiable programming. It combines four critical abilities:

- Using the CPU, GPU, or TPU to efficiently perform low-level tensor operations.
- The gradient of arbitrary differentiable expressions is computed.
- Scaling computing to a large number of devices, such as hundreds of GPUs in a cluster.
- External runtimes, such as servers, browsers, mobile devices, and embedded devices, are used to export programmes (graphs).

Keras is the high-level API for TensorFlow 2: a user-friendly, high-productivity interface for dealing with machine learning problems, with a focus on contemporary deep learning. It provides fundamental abstractions and building elements for designing and releasing high-iteration-rate machine learning systems.

Engineers and researchers can utilise Keras to fully use TensorFlow 2's scalability and cross-platform capabilities: Keras can be run on TPU or enormous clusters of GPUs, and Keras models can be exported to run in the web or on mobile devices.

4.3.2 Data preprocessing

Preprocessing data is a frequent first step in the deep learning workflow to prepare raw data in a way that the network can accept. You can, for example, change picture input to match the size of an image input layer. You may also employ preprocessing to improve desired characteristics or remove artifacts that could lead to a biased network. For example, you can normalize or remove noise from input data.

4.3.3 Initializing the Model parameters

Model Parameters are features of training data that will be learned throughout the learning process, such as weight and bias in the case of deep learning. The term parameter is frequently used to describe how effectively a model performs.

```
BATCH_SIZE = 8
EPOCHS = 20
WARMUP_EPOCHS = 2
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
HEIGHT = 728
WIDTH = 728
CANAL = 3
N_CLASSES = train['diagnosis'].nunique()
ES_PATIENCE = 5
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
```

4.3.4 Data Augmentation

To artificially increase the amount of data available, data augmentation is a set of ways for creating more data points from present data. Small changes to data or the use of deep learning models to generate more data points are examples of this. Data augmentation can assist improve the performance and output of machine learning models by producing new and varied cases to train datasets. When the dataset is large and diverse, a machine learning model performs better and more accurately. For machine learning models, data gathering and labeling can be time-consuming and costly. Data augmentation techniques can be used to reduce operational expenses by modifying datasets.

4.3.5 Creating the CNN model (ResNet)

The ResNet CNN model will be created using the inbuilt keras libraries and the necessary layers will also be defined.

4.3.6 Summary of the model

The model summary which includes all the parameters of the model will be displayed in detail.The model will be compiled using model.compile() and to get the summary model.summary() function call should be executed.

4.3.7 Training the model

The model will be trained using the inbuilt keras method train_generator and it will be fine-tuned using fit_generator method, for the necessary number of epochs.

4.3.8 Fine-tuning the model

In general, fine-tuning refers to making minor tweaks to a process in order to get the intended output or performance. The weights of a prior deep learning algorithm are used to design another comparable deep learning process for fine-tuning deep learning.

4.3.9 Calculation of metrics and plotting graphs

These are the most basic indicators since they identify a deep learning application's vital effectiveness. The number of right predictions divided by the total number of forecasts made is a simple way to calculate accuracy.

4.3.10 Validation of the model

The model will be saved in the .h5 format using model.save() function call. The trained model will be validated using the validation set of input images.

4.3.11 Defining the VGG model

The VGG architecture is a multilayer deep Convolutional Neural Network (CNN) architecture that stands for Visual Geometry Group. VGG-16 and VGG-19 feature 16 and 19 convolutional layers, respectively, and the term "deep" refers to the number of layers.

The VGG architecture serves as the foundation for cutting-edge object recognition models. The VGGNet, which was created as a deep neural network, outperforms baselines on a variety of tasks and datasets in addition to ImageNet.It is also one of the most commonly utilized image recognition architectures today.

4.3.12 Defining the Xception model

The term "extreme inception" refers to pushing Inception's concepts to their logical conclusion. To compress the original input in Inception, 1x1 convolutions were employed, and different types of filters were used to each of the depth spaces dependent on the input spaces. This is exactly what Xception does. Instead, the filters are applied to each depth map separately before the input space is compressed using 1X1 convolution across the depth. A depthwise separable convolution, a neural network design process that has been used since 2014, is virtually equivalent to this approach. There is one more distinction between Inception and Xception. The presence or absence of a non-linearity after the original operation. In the Inception model, a ReLU non-linearity follows both processes. Xception, on the other hand, does not introduce any non-linearity.

4.3.13 Display of Confusion Matrix

A confusion matrix is a method of summarizing a classification algorithm's performance. Calculating a confusion matrix can help you figure out what your classification model is getting right and where it's going wrong. The number of correct and bad predictions is totaled and broken down by class using count values. The confusion matrix's key is this.

4.3.14 Testing the model

The trained model will be tested using the inbuilt keras method called test_generator. Here the model is fed with different types of input images and the diagnosis level of DR corresponding to that input image will be displayed as output.

4.3.15 Demonstration of the results

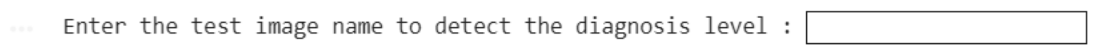
The image id is given as a dynamic user input and the model will process the image and give the diagnosis level of DR as output to the user.

4.3.16 Implementation of Graphical user interface

A graphical user interface (GUI) is a computer programme that uses symbols, visual metaphors, and pointing devices to allow a person to communicate with a computer.
Here there are two buttons - upload image button to give image as an input and classify image button to display the diagnosis level of DR as output.

4.4 Testing Process

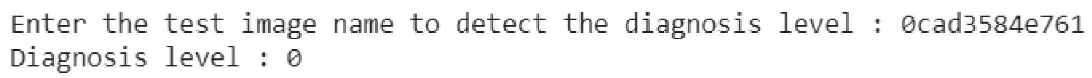
While Testing the Model we have to give several input testing images and then the trained CNN model will perform necessary steps and display the appropriate diagnosis level of DR as output.
The accuracy of our model will determine the outcome.
To test the model in our project we will give the image ids from the testing folder as input to the model and we will get the diagnosis level as output.
The dynamic input of the image will be as follows:



```
Enter the test image name to detect the diagnosis level :
```

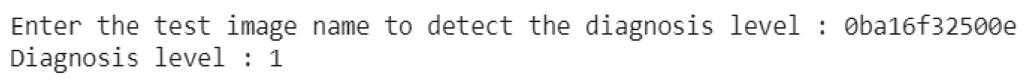
If we give the image id values in the prompted box and click enter we will get the results as shown below in different cases :

Case-0 :



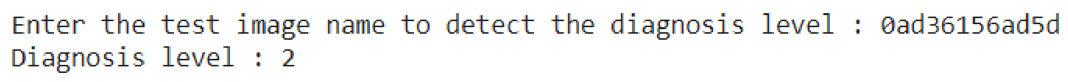
```
Enter the test image name to detect the diagnosis level : 0cad3584e761
Diagnosis level : 0
```

Case-1 :



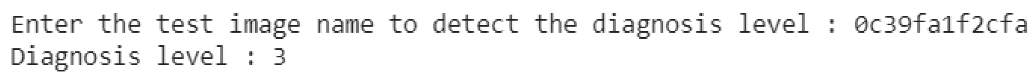
```
Enter the test image name to detect the diagnosis level : 0ba16f32500e
Diagnosis level : 1
```

Case-2 :



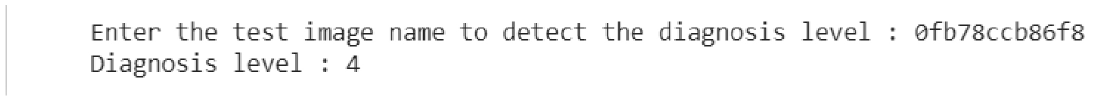
```
Enter the test image name to detect the diagnosis level : 0ad36156ad5d
Diagnosis level : 2
```

Case-3 :



```
Enter the test image name to detect the diagnosis level : 0c39fa1f2cfa
Diagnosis level : 3
```

Case-4 :



```
Enter the test image name to detect the diagnosis level : 0fb78ccb86f8
Diagnosis level : 4
```

CHAPTER-5

RESULTS/OUTPUTS AND DISCUSSION

The code takes the retinal fundus image as input and then preprocesses the input image and sends it through the model then finally we will get the diagnosis level of Diabetic Retinopathy as output.

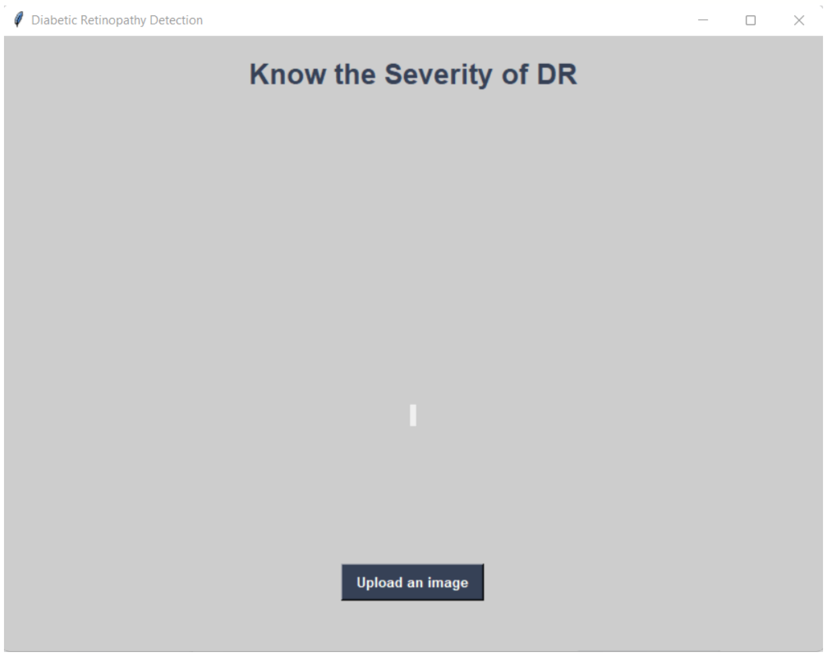


Figure 5.1 : GUI window of diabetic retinopathy detection

After running the code a GUI window will be prompted as shown in **figure 5.1**

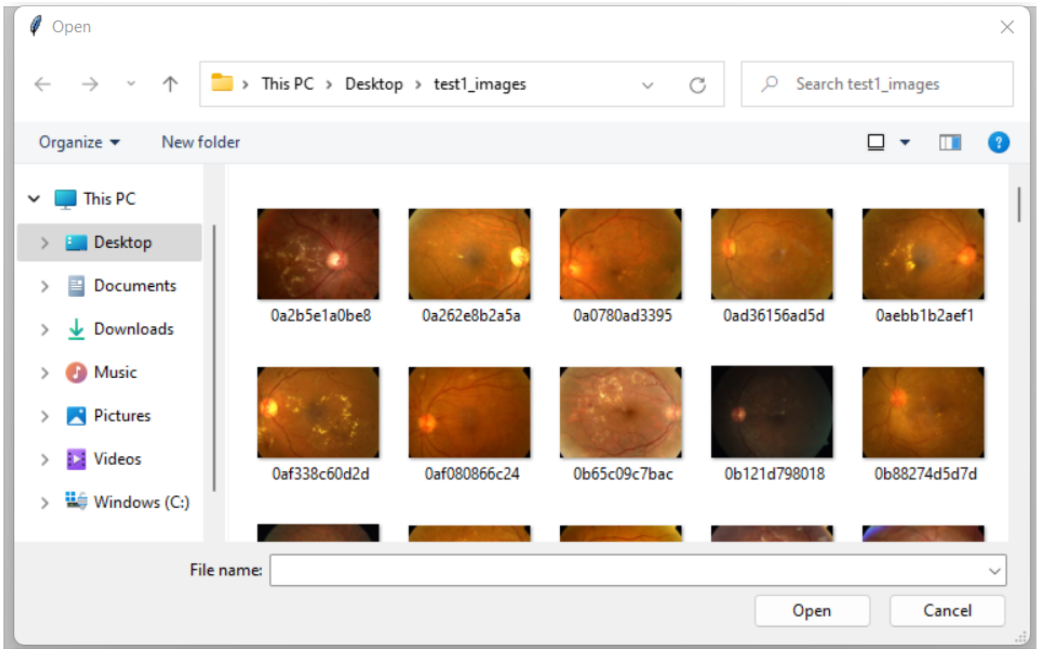


Figure 5.2 : Test images of the dataset

On clicking the upload an image button we will be redirected to a test images folder as shown in **figure 5.2**

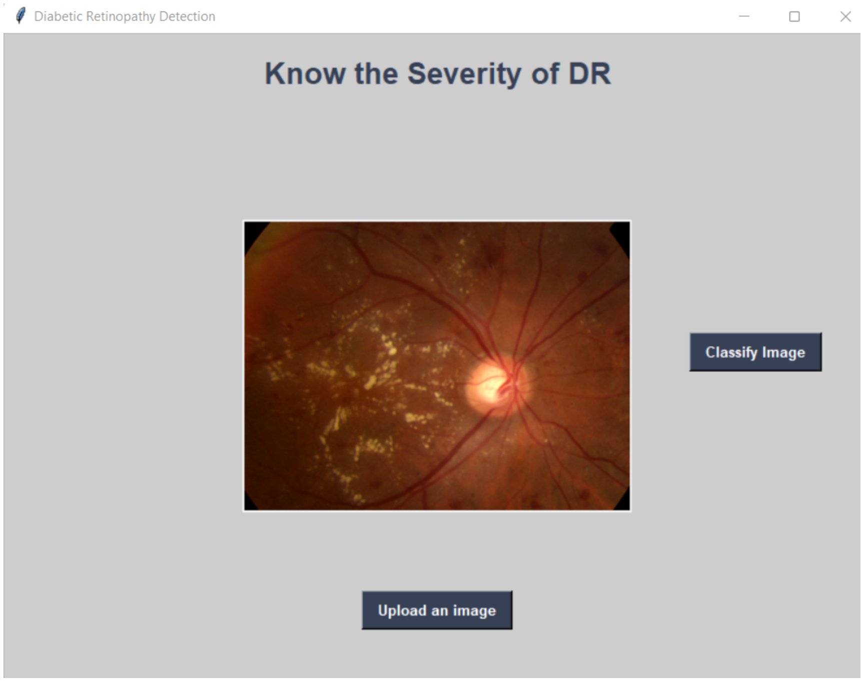
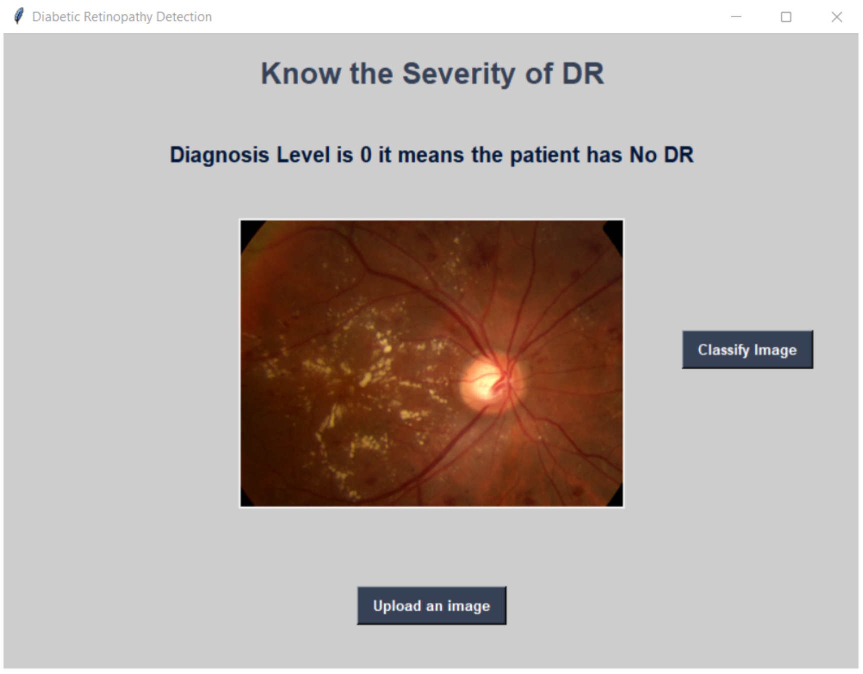


Figure 5.3 : The uploaded image

We will obtain a window containing an uploaded image after picking an image and clicking open, as illustrated in **figure 5.3**.

The diagnosis level of DR will be displayed when you click the classify image button, as illustrated in the following figures.

Case-0 :


Diabetic Retinopathy Detection

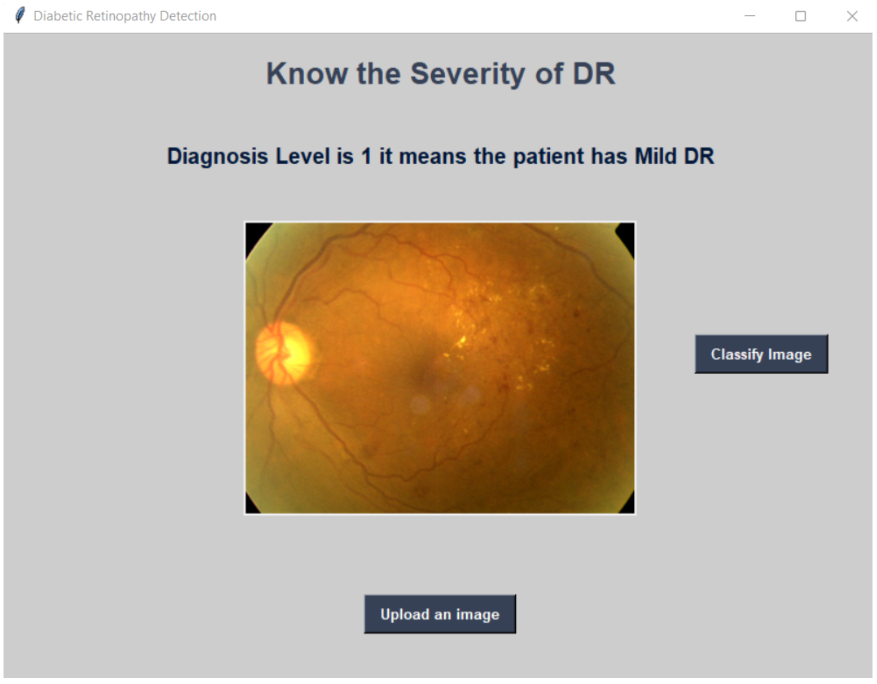
Know the Severity of DR

Diagnosis Level is 0 it means the patient has No DR

Figure 5.4 : Test image with diagnosis level - 0

The window displaying diagnosis level of a test image that belongs to level-0 is shown in **figure 5.4**

Case-1 :


Diabetic Retinopathy Detection

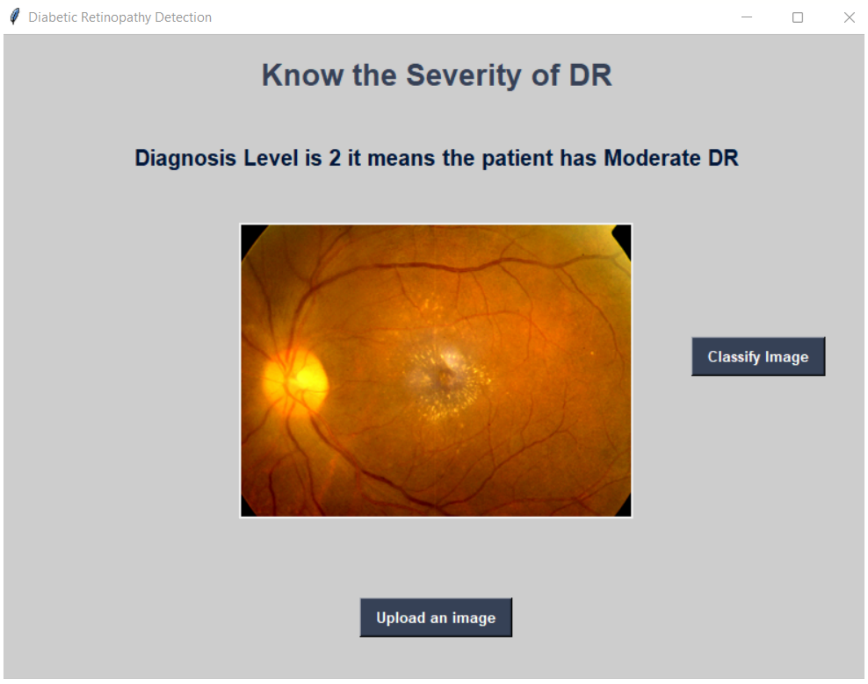
Know the Severity of DR

Diagnosis Level is 1 it means the patient has Mild DR

Figure 5.5 : Test image with diagnosis level - 1

The window displaying diagnosis level of a test image that belongs to level-1 is shown in **figure 5.5**

Case-2 :


Diabetic Retinopathy Detection

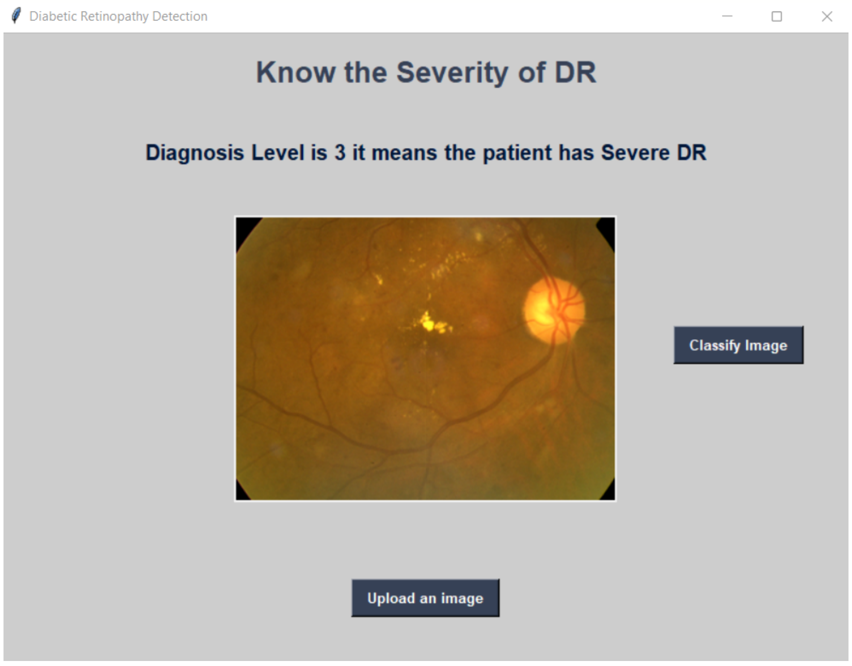
Know the Severity of DR

Diagnosis Level is 2 it means the patient has Moderate DR

Figure 5.6 : Test image with diagnosis level - 2

The window displaying diagnosis level of a test image that belongs to level-2 is shown in **figure 5.6**

Case-3 :


Diabetic Retinopathy Detection

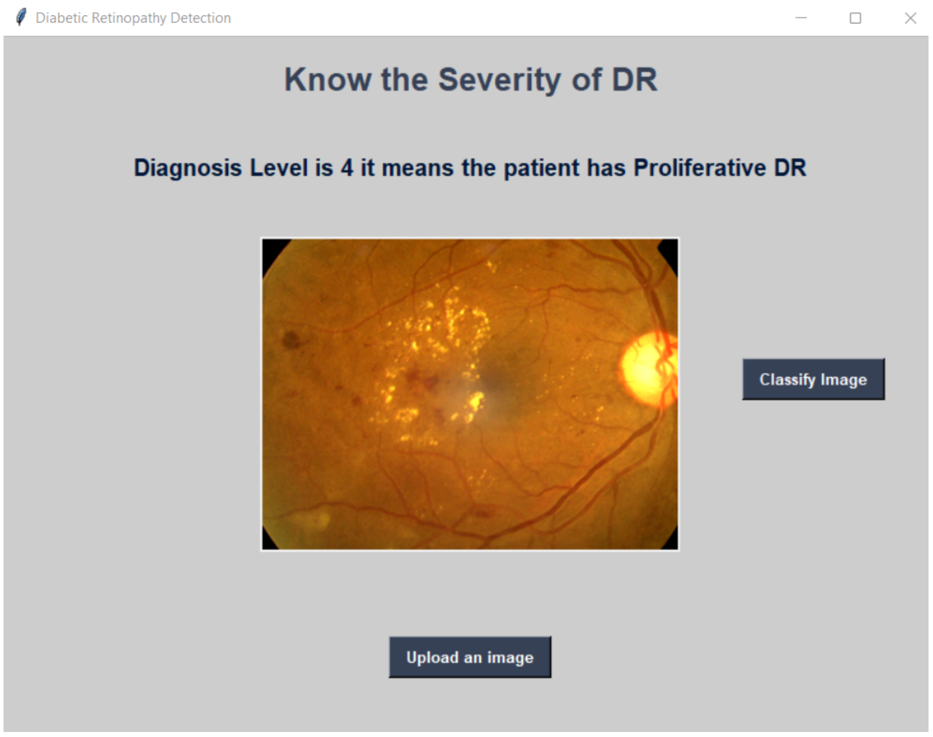
Know the Severity of DR

Diagnosis Level is 3 it means the patient has Severe DR

Figure 5.7 : Test image with diagnosis level - 3

The window displaying diagnosis level of a test image that belongs to level-3 is shown in **figure 5.7**

Case-4 :


Diabetic Retinopathy Detection

Know the Severity of DR

Diagnosis Level is 4 it means the patient has Proliferative DR

Figure 5.8 : Test image with diagnosis level - 4

Figure 5.8 shows the window that displays the diagnosis level of a test image that belongs to level-4.

The accuracy and loss graphs

Using ResNet :

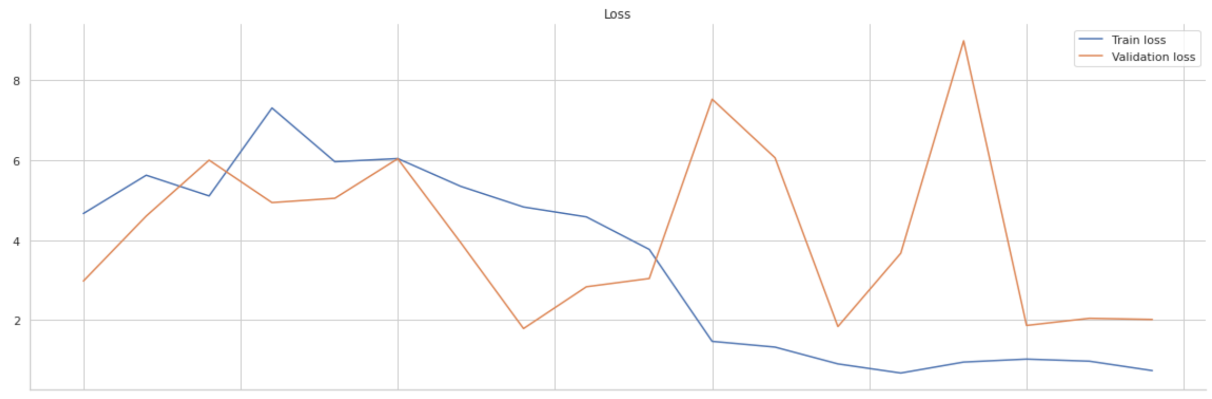


Figure 5.9 : Graphs of Train and Validation loss (using ResNet)

Figure 5.9 represents the graphs of validation loss and train loss. It is the plot between loss values and epochs. The loss numbers in this case are from the ResNet model.

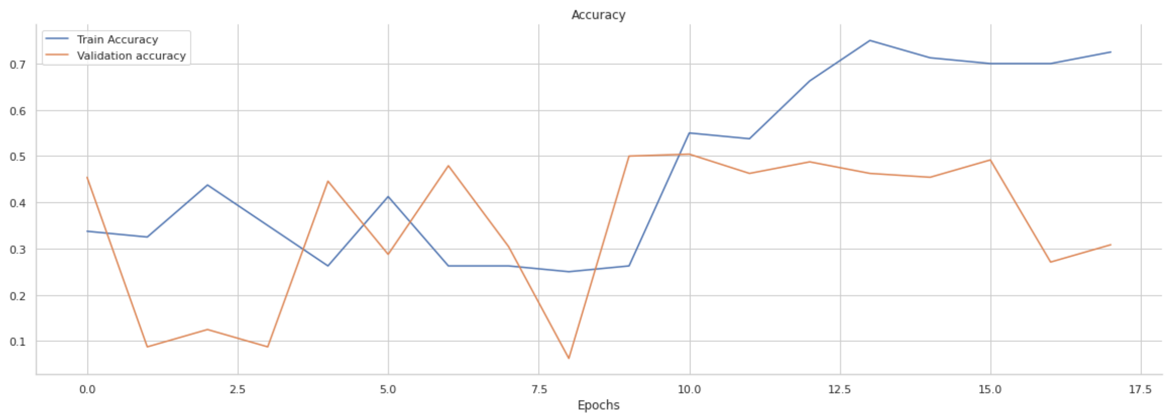
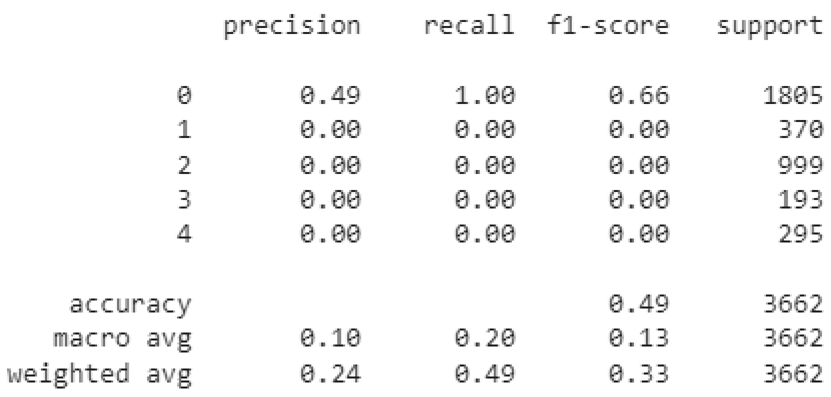


Figure 5.10 : Graphs of Train and Validation Accuracy (using ResNet)

Figure 5.10 represents the graphs of validation accuracy and train accuracy. It is the plot between accuracy values and epochs. Here the accuracy values belong to the ResNet model. The validation accuracy is observed to be 0.49.



```
              precision    recall  f1-score   support

           0       0.49      1.00      0.66      1805
           1       0.00      0.00      0.00       370
           2       0.00      0.00      0.00       999
           3       0.00      0.00      0.00       193
           4       0.00      0.00      0.00       295

    accuracy                           0.49      3662
   macro avg       0.10      0.20      0.13      3662
weighted avg       0.24      0.49      0.33      3662
```

Figure 5.11 : Performance metrics report obtained for Resnet model

Figure 5.11 gives the information regarding the performance metrics of the Resnet model.

Ensemble of ResNet and VGG :

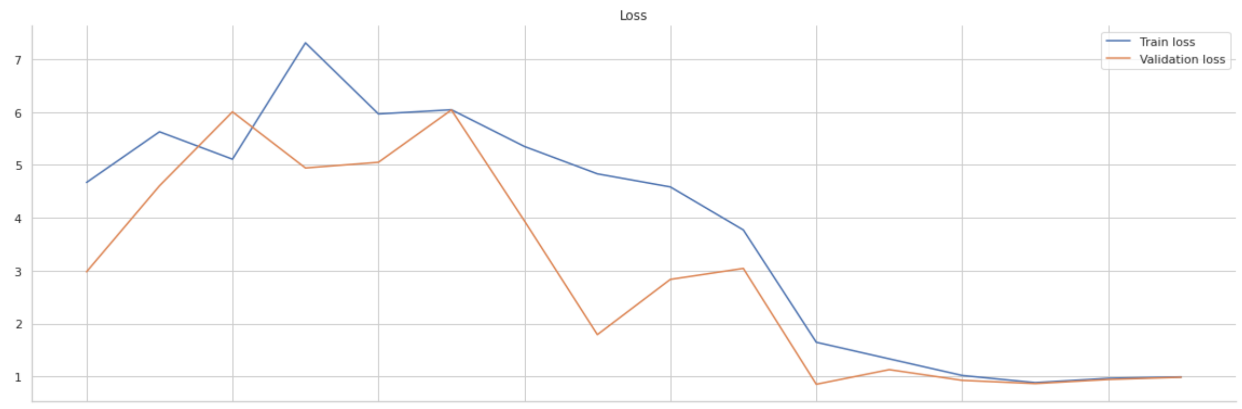


Figure 5.12 : Graphs of Train and Validation loss (Ensemble of ResNet and VGG)

Figure 5.12 represents the graphs of validation loss and train loss. It is the plot between loss values and epochs. Here the loss values belong to Ensemble of ResNet and VGG models.

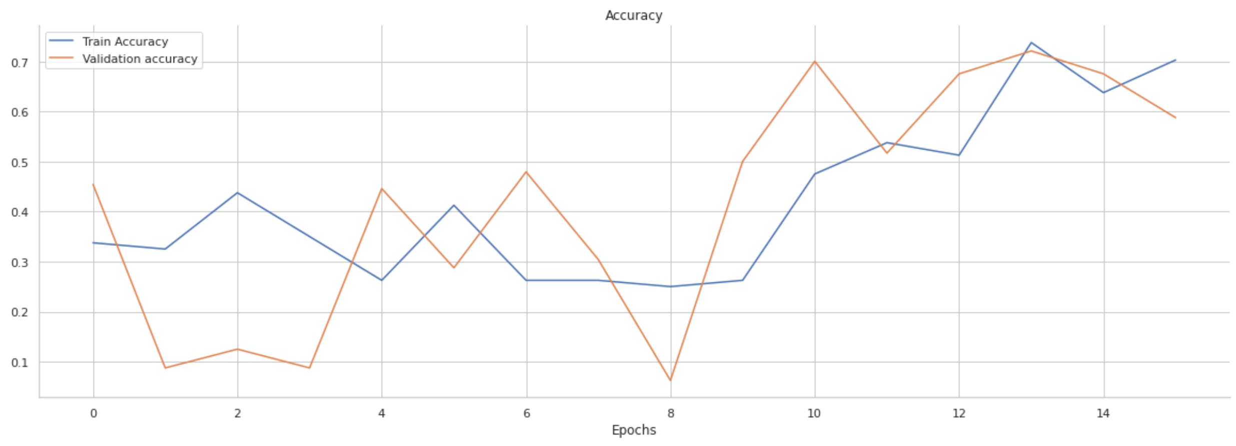
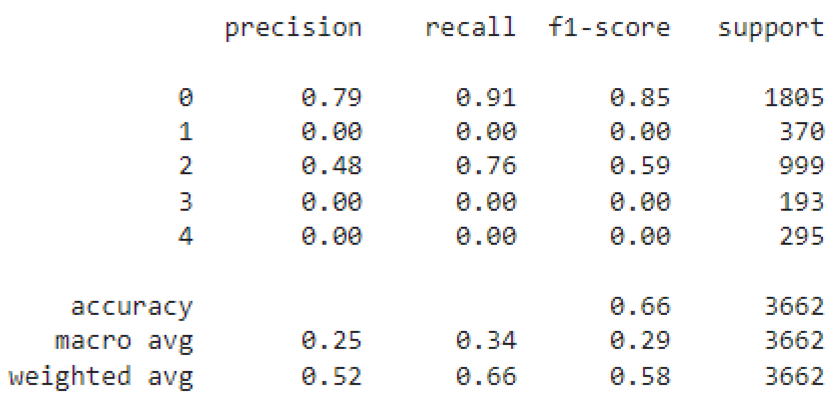


Figure 5.13 : Graphs of Train and Validation Accuracy (Ensemble of ResNet and VGG)

Figure 5.13 represents the graphs of validation accuracy and train accuracy. It is the plot between accuracy values and epochs. Here the accuracy values belong to the Ensemble of ResNet and VGG models. The validation accuracy is observed to be 0.66.



```
              precision    recall  f1-score   support

           0       0.79      0.91      0.85      1805
           1       0.00      0.00      0.00       370
           2       0.48      0.76      0.59       999
           3       0.00      0.00      0.00       193
           4       0.00      0.00      0.00       295

    accuracy                           0.66      3662
   macro avg       0.25      0.34      0.29      3662
weighted avg       0.52      0.66      0.58      3662
```

Figure 5.14 : Performance metrics report obtained for Ensemble of Resnet and VGG

Figure 5.14 gives the information regarding the performance metrics of the model which is an Ensemble of Resnet and VGG.

Ensemble of ResNet and Xception :

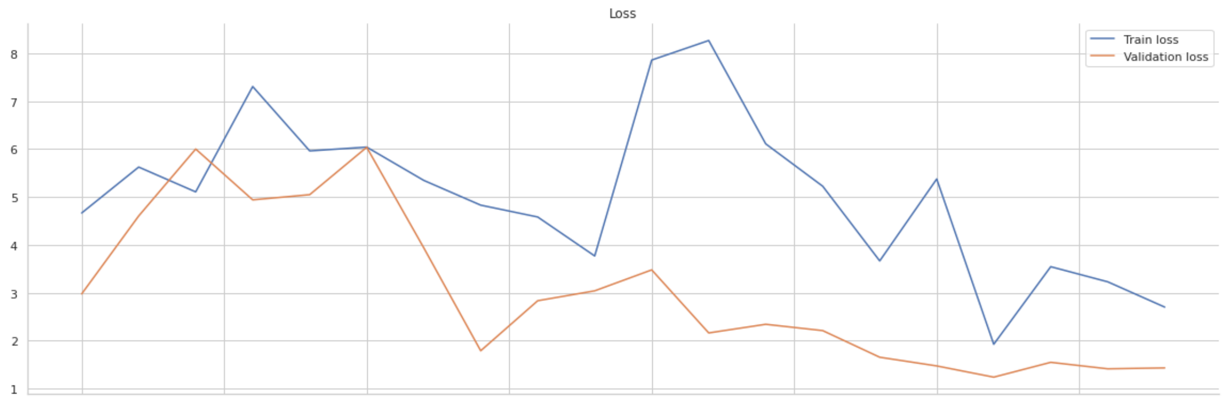


Figure 5.15 : Graphs of Train and Validation loss (Ensemble of ResNet and Xception)

Figure 5.15 represents the graphs of validation loss and train loss. It is the plot between loss values and epochs. Here the loss values belong to Ensemble of ResNet and Xception models.

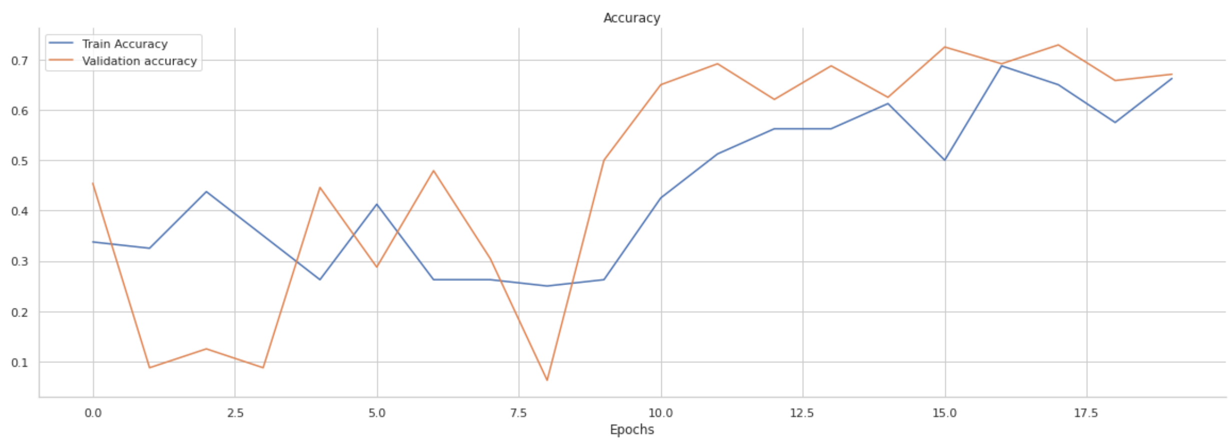
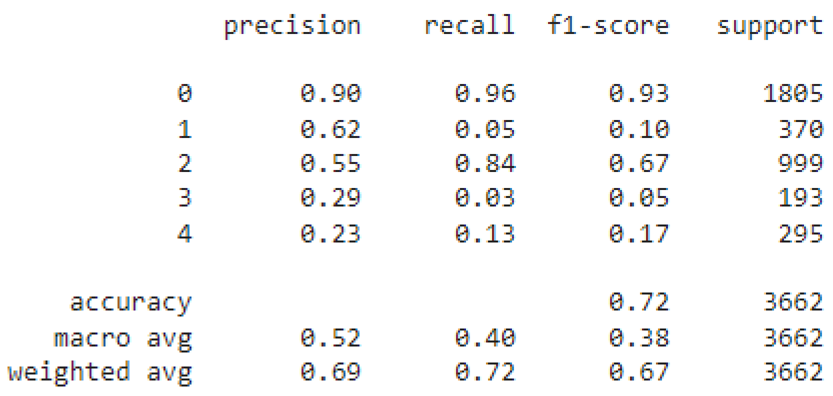


Figure 5.16 : Graphs of Train and Validation Accuracy (Ensemble of ResNet and Xception)

Figure 5.16 represents the graphs of validation accuracy and train accuracy. It is the plot between accuracy values and epochs. Here the accuracy values belong to the Ensemble of ResNet and Xception models. The validation accuracy is observed to be 0.72.



```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      1805
           1       0.62      0.05      0.10       370
           2       0.55      0.84      0.67       999
           3       0.29      0.03      0.05       193
           4       0.23      0.13      0.17       295

    accuracy                           0.72      3662
   macro avg       0.52      0.40      0.38      3662
weighted avg       0.69      0.72      0.67      3662
```

Figure 5.17 : Performance metrics report obtained for Ensemble of Resnet and Xception

Figure 5.17 gives the information regarding the performance metrics of the CNN model.

The fraction of relevant occurrences among the recovered examples is known as precision (also known as positive predictive value). The precision values for DR levels 0,1,2,3,4 are 0.90,0.62,0.55,0.29,0.23 and 0.90,0.62,0.55,0.29,0.23 correspondingly. The fraction of relevant instances that were retrieved is known as recall (also known as sensitivity). The recall values for DR levels 0,1,2,3,4 are

0.96,0.05,0.84,0.03,0.13 and 0.96,0.05,0.84,0.03,0.13 correspondingly. The F1-score takes the harmonic mean of a classifier's precision and recall to create a single statistic. The F1-score values for DR levels 0,1,2,3,4 are 0.97,0.10,0.67,0.05,0.17, and 0.97,0.10,0.67,0.05,0.17, respectively. The number of samples of the genuine response that fall into each class of goal values is known as support. The support values related to DR levels 0,1,2,3,4 are1805,370,999,193,295 respectively which sums up to 3662 which are the total no.of train images present in the dataset. The accuracy of the CNN model is 0.72.

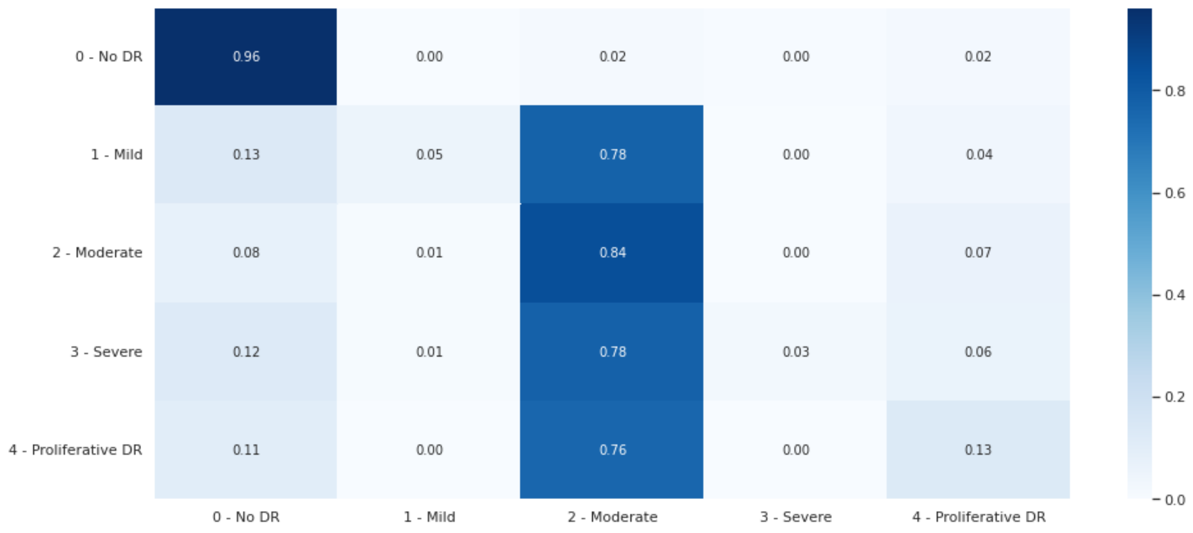


Figure 5.18 : Confusion matrix

Figure 5.18 represents the confusion matrix of CNN model. A confusion matrix represents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes. It is a matrix used to determine the performance of the classification models for a given set of test data. Only if the true values for test data are known can it be determined.

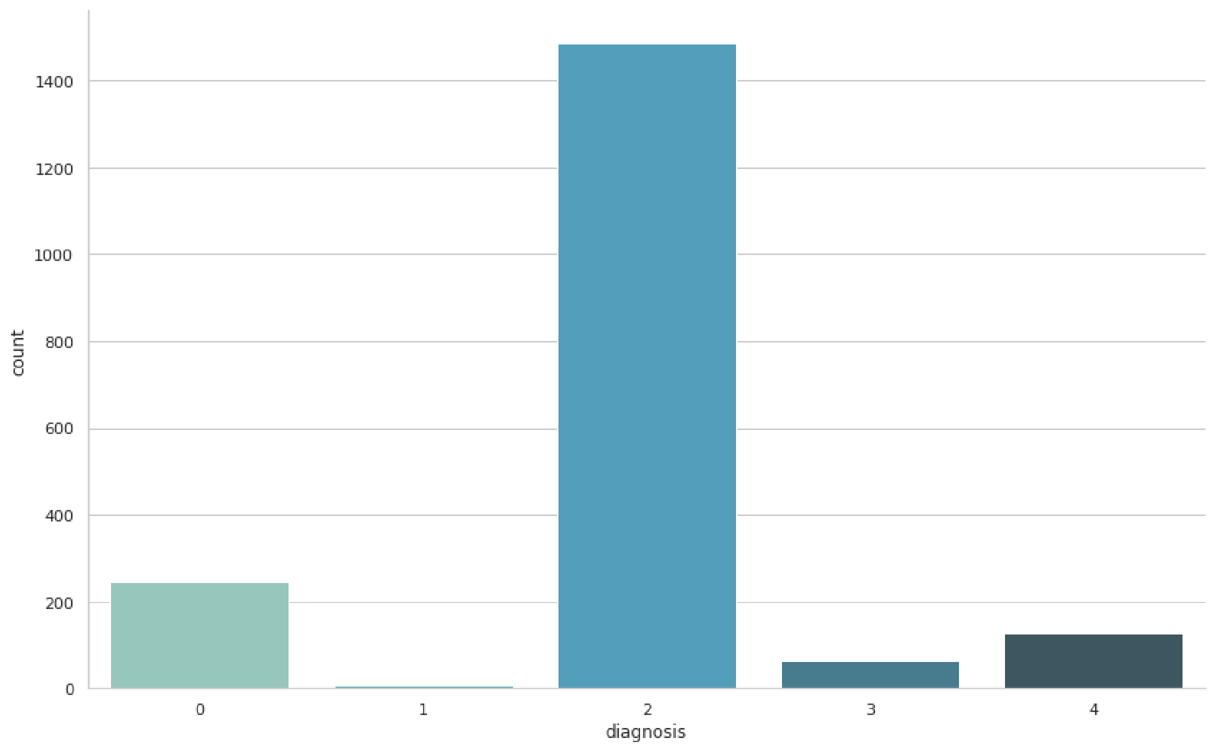


Figure 5.19 : No.of Test images in each class of DR

Figure 5.19 shows the bar plot representing the number of test images that belongs to the respective diagnosis level of DR.

We can see, from the above three models i.e., Resnet, ensemble of Resnet and VGG and the other ensemble of Resnet and Xception the accuracies gradually got increasing and the losses got decreased. For the first model which is Resnet (Refer **Figure 5.9** and **Figure 5.10**) the validation loss is greater than the train loss and the validation accuracy is less than the train accuracy. For the second model which is an ensemble of Resnet and VGG (Refer **Figure 5.12** and **Figure 5.13**) we observe that the train and validation loss are gradually decreasing and they have the similar value of losses, when coming to accuracies the train accuracy is increasing gradually and the validation accuracy is getting decreased at a point of time where the train accuracy is more than the validation accuracy. The train and validation losses are gradually decreasing, and the validation loss is less than the train loss, in the third model, which is an ensemble of Resnet and Xception (Refer **Figure 5.15** and **Figure 5.16**) we observe that the train and validation accuracies are increasing, and the validation accuracy is greater than the train accuracy. We were able to achieve a 0.72 total accuracy (Refer **Figure 5.17**).

After the experimentation, we got the results in which we showed the accuracy of our project. For the same dataset, we employed three architectures and compared their accuracies (Refer **Table 5.1**).

Table 5.1: Results of the architectures

Architecture	**Dataset**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Resnet	Kaggle	0.49	0.24	0.49	0.33
Ensemble(Resnet, VGG)	Kaggle	0.66	0.52	0.66	0.58
Ensemble(Resnet, Xception)	Kaggle	0.72	0.69	0.72	0.67

CHAPTER - 6
CONCLUSIONS AND FUTURE SCOPE

6.1 Conclusions

Our project's purpose is to use Fundus Photographs to estimate the severity of Diabetic Retinopathy using a CNN model. It is a truth that the more precise the treatment plan is, the better and more accurate the diagnosis is. As a result, for an effective treatment regimen, diagnostic measures should aim for precision. We have proposed an approach of ensembling ResNet and Xception architectures for diabetic retinopathy detection. In comparison to other ways, our proposed method performed well which has resulted in an increase in train and validation accuracies and a decrease in train and validation losses. We were able to get a high level of accuracy in the diagnosis outcomes in our project.

6.2 Limitations

- The proposed model is not compatible with mobile-like devices. For making the model more compatible with all other devices we could use lightweight models[15] like MobileNet.

6.3 Future Scope

- One can train the model using a large number of images for training if there is an availability of high computational devices like GPU.
- There is also a chance to explore various large-size datasets of Diabetic Retinopathy and increase the accuracy of the model.
- One can also make the model more accurate by increasing the number of epochs while training the model if there is an availability of GPU.

REFERENCES

[1]. Supriya Mishra, Seema Hanchate, Zia Saquib; “Diabetic Retinopathy Detection using Deep Learning”; International Conference on Smart Technologies in Computing, Electrical, and Electronics (ICSTCEE 2020); December 19, 2020.

[2]. Yann LeCun, Leon Bottou, Yoshua, and Patrick Haffner; “Gradient-Based Learning Applied to Document Recognition”; PROC. OF THE IEEE, November 1998.

[3]. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”; Microsoft Research, 2015.

[4]. CNN Architectures, “https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5”, 2017.

[5]. Deep Learning, “https://www.geeksforgeeks.org/introduction-deep-learning/”, 2019.

[6]. Residual Networks(ResNet), “https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/”, 2020.

[7]. Anuj Jain, Arnav Jalui, Jahanvi Jasani, Yash Lahoti, Ruhina Karani. "Deep Learning for Detection and Severity Classification of Diabetic Retinopathy", 2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT), 2019

[8]. Emma Beede, Elizabeth Baylor, Fred Hersch, Lauren Wilcox, “A Human-Centered Evaluation of a Deep Learning System Deployed in Clinics for the

Detection of Diabetic Retinopathy", CHI 2020, April 25–30, 2020, Honolulu, HI, USA, 2020.

[9]. Dataset, "https://www.kaggle.com/sovitrath/diabetic-retinopathy-224x224-gaussian-filtered"

[10]. Introduction to Pooling layer,"https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/",2021

[11]. Batch Normalization,"https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/",2021

[12]. Introduction to Keras,"https://keras.io/"

[13].Introduction to Tensorflow, "https://www.tensorflow.org/tutorials"

[14]. Explanation of Xception, "https://openaccess.thecvf.com/content_cvpr_2017/html/Chollet_Xception_Deep_Learning_CVPR_2017_paper.html".

[15]. Kolla, M., Venugopal, T. (2022). Diabetic Retinopathy Classification Using Lightweight CNN Model. In: Kumar, A., Mozar, S. (eds) ICCCE 2021. Lecture Notes in Electrical Engineering, vol 828. Springer, Singapore. https://doi.org/10.1007/978-981-16-7985-8_131

[16]. Borys Tymchenko , Philip Marchenko and Dmitry Spodarets, "Deep Learning Approach to Diabetic Retinopathy Detection"; March 3,2020. https://arxiv.org/pdf/2003.02261.pdf

APPENDIX - 1

The Appendix - 1 gives the information about the source code implementation of the project.

Data preprocessing

```
train["id_code"] = train["id_code"].apply(lambda x: x + ".png")
test["id_code"] = test["id_code"].apply(lambda x: x + ".png")
train['diagnosis'] = train['diagnosis'].astype('str')
display(train.head())
f, ax = plt.subplots(figsize=(14, 8.7))
ax = sns.countplot(x="diagnosis", data=train, palette="GnBu_d")
sns.despine()
plt.show()
sns.set_style("white")
count = 1
plt.figure(figsize=[20, 20])
for img_name in train["id_code"][:15]:
    img = cv2.imread("/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/train1_images/%s" %
img_name)[...,[2, 1, 0]]

    plt.subplot(5, 5, count)
    plt.imshow(img)
    plt.title("Image %s" % count)
    count += 1
plt.show()
```

Data Augmentation

```
train_datagen=ImageDataGenerator(rescale=1./255,
                                 validation_split=0.2,
                                 horizontal_flip=True)

train_generator=train_datagen.flow_from_dataframe(
    dataframe=train,
    directory=path+"train1_images",
    x_col="id_code",
    y_col="diagnosis",
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    target_size=(HEIGHT, WIDTH),
    subset='training')

valid_generator=train_datagen.flow_from_dataframe(
    dataframe=train,
    directory=path+"train1_images",
    x_col="id_code",
    y_col="diagnosis",
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    target_size=(HEIGHT, WIDTH),
    subset='validation')

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(
        dataframe=test,
        directory = path+"test1_images",
        x_col="id_code",
        target_size=(HEIGHT, WIDTH),
        batch_size=1,
        shuffle=False,
        class_mode=None)
```

Creating the CNN model (ResNet)

```
def create_model(input_shape, n_out):
        input_tensor = Input(shape=input_shape)
        base_model = tensorflow.keras.applications.ResNet50(weights='imagenet',
                                include_top=False,
                                input_tensor=input_tensor)
        x = GlobalAveragePooling2D()(base_model.output)
        x = Dropout(0.5)(x)
        x = Dense(2048, activation='relu')(x)
        x = Dropout(0.5)(x)
        final_output = Dense(n_out, activation='softmax', name='final_output')(x)
        model = Model(input_tensor, final_output)
        return model
```

Training the model

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size

history_warmup = model.fit_generator(generator=train_generator,
                              steps_per_epoch=10,
                              validation_data=valid_generator,
                              validation_steps=30,
                              epochs=10,
                              verbose=1).history

for layer in model.layers:
    layer.trainable = True
```

```
es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE,
restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min',
patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)
callback_list = [es, rlrop]

optimizer = tensorflow.keras.optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",
metrics=metric_list)
model.summary()
```

Fine-tuning the model

```
history_finetunning = model.fit_generator(generator=train_generator,
                              steps_per_epoch=10,
                              validation_data=valid_generator,
                              validation_steps=30,
                              epochs=10,
                              callbacks=callback_list,
                              verbose=1).history
```

Validation of the model

```
model.save("/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/my_model.h5")
complete_datagen = ImageDataGenerator(rescale=1./255)
complete_generator = complete_datagen.flow_from_dataframe(
    dataframe=train,
    directory = "/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/train1_images/",
    x_col="id_code",
    target_size=(HEIGHT, WIDTH),
    batch_size=1,
        shuffle=False,
        class_mode=None)
STEP_SIZE_COMPLETE = complete_generator.n//complete_generator.batch_size
train_preds = model.predict_generator(complete_generator,
steps=STEP_SIZE_COMPLETE)
train_preds = [np.argmax(pred) for pred in train_preds]
labels = ['0 - No DR', '1 - Mild', '2 - Moderate', '3 - Severe', '4 - Proliferative DR']
cnf_matrix = confusion_matrix(train['diagnosis'].astype('int'), train_preds)
cnf_matrix_norm = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
df_cm = pd.DataFrame(cnf_matrix_norm, index=labels, columns=labels)

plt.figure(figsize=(16, 7))
sns.heatmap(df_cm, annot=True, fmt='.2f', cmap="Blues")
plt.show()
```

Defining the VGG model

```
from keras.applications.vgg16 import VGG16
from keras.models import Sequential
from tifffile.tifffile import sequence
from keras.layers import Dense, Dropout, Flatten
VGG_MODEL = VGG16(weights = 'imagenet', include_top = False ,input_shape
=(728,728,3))
for layer in VGG_MODEL.layers[:-4]:
  layer.trainable = False

model = Sequential()
model.add(VGG_MODEL)
model.add(Flatten())
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(126, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(5, activation = 'softmax'))
es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE,
restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min',
patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = tensorflow.keras.optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",
metrics=metric_list)
model.summary()

history_finetunning = model.fit_generator(generator=train_generator,
                              steps_per_epoch=10,
                              validation_data=valid_generator,
                              validation_steps=30,
                              epochs=10,

                              callbacks=callback_list,
                              verbose=1).history
history = {'loss': history_warmup['loss'] + history_finetunning['loss'],
        'val_loss': history_warmup['val_loss'] + history_finetunning['val_loss'],
        'accuracy': history_warmup['accuracy'] + history_finetunning['accuracy'],
        'val_accuracy': history_warmup['val_accuracy'] +
history_finetunning['val_accuracy']}

sns.set_style("whitegrid")
fig, (ax1, ax2) = plt.subplots(2, 1, sharex='col', figsize=(20, 14))

ax1.plot(history['loss'], label='Train loss')
ax1.plot(history['val_loss'], label='Validation loss')
ax1.legend(loc='best')
ax1.set_title('Loss')
```

```
ax2.plot(history['accuracy'], label='Train Accuracy')
ax2.plot(history['val_accuracy'], label='Validation accuracy')
ax2.legend(loc='best')
ax2.set_title('Accuracy')

plt.xlabel('Epochs')
sns.despine()
plt.show()

complete_datagen = ImageDataGenerator(rescale=1./255)
complete_generator = complete_datagen.flow_from_dataframe(
        dataframe=train,
        directory = "/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/train1_images/",
        x_col="id_code",
        target_size=(HEIGHT, WIDTH),
        batch_size=1,
        shuffle=False,
        class_mode=None)

STEP_SIZE_COMPLETE = complete_generator.n//complete_generator.batch_size
train_preds = model.predict_generator(complete_generator,
steps=STEP_SIZE_COMPLETE)
train_preds = [np.argmax(pred) for pred in train_preds]
labels = ['0 - No DR', '1 - Mild', '2 - Moderate', '3 - Severe', '4 - Proliferative DR']
cnf_matrix = confusion_matrix(train['diagnosis'].astype('int'), train_preds)
cnf_matrix_norm = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
df_cm = pd.DataFrame(cnf_matrix_norm, index=labels, columns=labels)
plt.figure(figsize=(16, 7))
sns.heatmap(df_cm, annot=True, fmt='.2f', cmap="Blues")
plt.show()
model.save("/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/my_model1.h5")
```

Defining the Xception model

```
from keras.applications.xception import Xception
xception = Xception(include_top=False, input_shape = (728,728,3))
for layer in xception.layers[:-4]:
  layer.trainable = False

model = Sequential()
model.add(xception)
model.add(Flatten())
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(126, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(5, activation = 'softmax'))

import tensorflow as tf
es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE,
restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min',
patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)
callback_list = [es, rlrop]
optimizer = tf.optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",
metrics=metric_list)
model.summary()
history_finetunning = model.fit_generator(generator=train_generator,
                              steps_per_epoch=10,
                              validation_data=valid_generator,
                              validation_steps=30,
                              epochs=10,
                              callbacks=callback_list,
                              verbose=1).history
```

```
history = {'loss': history_warmup['loss'] + history_finetunning['loss'],
          'val_loss': history_warmup['val_loss'] + history_finetunning['val_loss'],
          'accuracy': history_warmup['accuracy'] + history_finetunning['accuracy'],
          'val_accuracy': history_warmup['val_accuracy'] +
history_finetunning['val_accuracy']}

sns.set_style("whitegrid")
fig, (ax1, ax2) = plt.subplots(2, 1, sharex='col', figsize=(20, 14))

ax1.plot(history['loss'], label='Train loss')
ax1.plot(history['val_loss'], label='Validation loss')
ax1.legend(loc='best')
ax1.set_title('Loss')

ax2.plot(history['accuracy'], label='Train Accuracy')
ax2.plot(history['val_accuracy'], label='Validation accuracy')
ax2.legend(loc='best')
ax2.set_title('Accuracy')
plt.xlabel('Epochs')
sns.despine()
plt.show()
model.save("/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/my_model2.h5")

complete_datagen = ImageDataGenerator(rescale=1./255)
complete_generator = complete_datagen.flow_from_dataframe(
        dataframe=train,
        directory = "/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/train1_images/",
        x_col="id_code",
        target_size=(HEIGHT, WIDTH),
        batch_size=1,
        shuffle=False,
        class_mode=None)
```

```
STEP_SIZE_COMPLETE = complete_generator.n//complete_generator.batch_size
train_preds = model.predict_generator(complete_generator,
steps=STEP_SIZE_COMPLETE)
train_preds = [np.argmax(pred) for pred in train_preds]
```

Testing the model

```
test_generator.reset()
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size
preds = model.predict_generator(test_generator, steps=STEP_SIZE_TEST,verbose
=1)
predictions = [np.argmax(pred) for pred in preds]

filenames = test_generator.filenames
results = pd.DataFrame({'id_code':filenames, 'diagnosis':predictions})
results['id_code'] = results['id_code'].map(lambda x: str(x)[:-4])

results.to_csv('/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/submission1.csv',index=False)
from google.colab import files
files.download('/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/submission1.csv')

f, ax = plt.subplots(figsize=(14, 8.7))
ax = sns.countplot(x="diagnosis", data=results, palette="GnBu_d")
sns.despine()
plt.show()

from keras import models
my_model = models.load_model("/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/my_model2.h5")
import sklearn
print(sklearn.metrics.classification_report(train['diagnosis'].astype('int'), train_preds))
```

```
import pandas as pd
pd.read_csv("/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/submission1.csv")
```

Demonstration of the results

```
del input
import pandas as pd
input=input("Enter the test image name to detect the diagnosis level : ")
result=pd.read_csv('/content/drive/My Drive/Colab
Notebooks/aptos2019-blindness-detection/Dataset/submission1.csv')
id_codes=result['id_code'].tolist()
diagnosiss=result['diagnosis'].tolist()
for i in range(len(id_codes)):
  if(str(id_codes[i]) == input):
    print("Diagnosis level : "+str(diagnosiss[i]))
```