DEEP LEARNING APPROACH FOR PREDICTION OF DIABETIC RETINOPATHY

project report submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Submitted by

ALOKAM GNANESWARA SAI	319106410005
BALARAM MAHANTHI DEEPAK PATNAIK	319106410006
BETHA MADHURI	319106410007

Under the esteemed guidance of

Prof. M. SHASHI

Professor

DEPARTMENT OFCOMPUTER SCIENCE AND SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING

ANDHRA UNIVERSITY

VISAKHAPATNAM-530003

2023

DEPARTMENT OF COMPUTER SCIENCE AND SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING

ANDHRA UNIVERSITY

CERTIFICATE

This is to certify that the project report entitled "**DEEP LEARNING APPROACH FOR PREDICTION OF DIABETIC RETINOPATHY**" is a bonafide record of work carried out **Alokam Gnaneswara Sai (319106410005), Balaram Mahanthi Deepak Patnaik (319106410006), Betha Madhuri (319106410007)** students submitted in partial fulfilment of the requirement for the award of the degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide
Prof. M. SHASHI

Head of Department
Prof. K. VENKATA RAO

DEPARTMENT OF COMPUTER SCIENCE AND SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING

ANDHRA UNIVERSITY

DECLARATION

I/We, hereby declare that the project report entitled "**DEEP LEARNING APPROACH FOR PREDICTION OF DIABETIC RETINOPATHY"** has been prepared by us during the period December 2023 – April 2023 submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering.

Registration No(s).	**Name(s)**
319106410005	Alokam Gnaneswara Sai
319106410006	Balaram Mahanthi Deepak Patnaik
319106410007	Betha Madhuri

ACKNOWLEDGEMENT

We have immense pleasure in expressing our earnest gratitude to our Project Guide **Prof. M. Shashi**, Andhra University for her inspiring and scholarly guidance. Despite her pre-occupation with several assignments, she has been kind enough to spare her valuable time and gave us the necessary counsel and guidance at every stage of planning and constitution of this work. We express sincere gratitude for having accorded us permission to take up this project work and for helping us graciously throughout the execution of this work.

We express sincere thanks to **Prof. K. Venkata Rao**, Head of the Department Computer Science and Systems Engineering, Andhra University College of Engineering for his keen interest and providing necessary facilities for this project study.

We express sincere thanks to **Prof. G. Sasibhusana Rao**, Principal, Department of Electronics and Communication Engineering, Andhra University College of Engineering for his keen interest and for providing necessary facilities for this project study.

We express sincere thanks to **Prof. PVGD Prasad Reddy**, Vice Chancellor, Andhra University for his keen interest and for providing necessary facilities for this project study.

We extend our sincere thanks to our academic teaching staff and nonteaching staff for their help throughout our study.

ABSTRACT

Diabetic retinopathy is one of the most threatening complications of diabetes that leads to permanent blindness if left untreated. Diabetic retinopathy is an eye condition that can cause vision loss and blindness in people who have diabetes. It affects blood vessels in the retina (the light-sensitive layer of tissue in the back of your eye). If you have diabetes, it's important to get a comprehensive dilated eye exam at least once a year. Diabetic retinopathy may not have any symptoms at first — but finding it early can help you take steps to protect your vision. One of the essential challenges is early detection, which is very important for treatment success. Unfortunately, the exact identification of the diabetic retinopathy stage is notoriously tricky and requires expert human interpretation of fundus images. Simplification of the detection step is crucial and can help millions of people.

However, the high cost of big labeled datasets, as well as inconsistency between different doctors, impede the performance of these methods. Convolutional neural networks (CNN) have been successfully applied in many adjacent subjects, and for diagnosis of diabetic retinopathy itself. In this thesis, we propose a deep-learning-based method for automatic stage detection of diabetic retinopathy by single photography of the human fundus. Additionally, we propose the multistage approach to transfer learning, which makes use of similar datasets with different labeling. The presented method can be used as a screening method for early detection of diabetic retinopathy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

DR	Diabetic Retinopathy
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
ResNet	Residual Neural Network

CHAPTER 1: INTRODUCTION

1.1 Introduction

Diabetic retinopathy (DR) is one of the most threatening complications of diabetes in which damage occurs to the retina and causes blindness. It damages the blood vessels within the retinal tissue, causing them to leak fluid and distort vision. Along with diseases leading to blindness, such as cataracts and glaucoma, Diabetic Retinopathy is one of the most frequent ailments, according to the US, UK, and Singapore statistics.

DR progresses with four stages:

1.Mild non-proliferative retinopathy: the earliest stage, where only microaneurysms can occur.

2.Moderate non-proliferative retinopathy: a stage which can be described by losing the blood vessels' ability of blood transportation due to their distortion and swelling with the progress of the Disease.

3.Severe non-proliferative retinopathy: results in deprived blood supply to the retina due to the increased blockage of more blood vessels, hence signaling the retina for the growing of fresh blood vessels;

4. Proliferative diabetic retinopathy: It is the advanced stage, where the growth features secreted by the retina activate proliferation of the new blood vessels, growing along inside covering of retina in some vitreous gel, filling the eye.

Each stage has its characteristics and particular properties, so doctors possibly could not take some of them into account, and thus make an incorrect diagnosis. So this leads to the idea of creation of an automatic solution for DR detection. According to recent survey, at least 56% of new cases of this disease could be reduced with

proper and timely treatment and monitoring of the eyes. Existing ways of diagnosing are quite inefficient due to their duration time, and the number of ophthalmologists included in patient's problem solution.In our project, we propose the transfer learning approach and an automatic method for detection of the stage of diabetic retinopathy by single photography of the human fundus.

1.2 PROGRAMMING LANGUAGE

Python is a widely used language in deep learning and is the primary language used in this project. It has several powerful libraries such as TensorFlow, Keras, and which make it easier to build and train neural networks. These libraries provide an interface to build deep learning models and train them on large datasets efficiently. Python has a large and active developer community that contributes to a vast library of open-source packages and tools.

1.3 MOTIVATION

The motivation behind this project is to develop an automated system for the detection of diabetic retinopathy, which can save time and reduce the cost of screening. The system can also be used in areas where there is a shortage of ophthalmologists.

1.4 PROBLEM STATEMENT

This is **a 5 Class Image Classification** Task based on a Kaggle dataset from Eye Images (Aravind Eye hospital) - APTOS 2019 Challenge. The goal is to predict the Blindness Stage (0-4) class from the Eye retina Image using Deep Learning Models (CNN).

The full dataset consists of 18590 fundus photographs, which are divided into 3662 training, 1928 validation, and 13000 testing images by organizers of Kaggle competition, each photography is of different resolution.

The smallest native size among all of the datasets is 640x480.

Here, we consider 5 stages of diabetic retinopathy**:**

1.No diabetic retinopathy (label 0)

2. Mild diabetic retinopathy (label 1)

3. Moderate diabetic retinopathy (label 2)

4.Severe diabetic retinopathy (label 3)

5.Proliferative diabetic retinopathy (label 4)

1.5 OBJECTIVES

The main objective of the project is to develop a deep learning model that can accurately detect diabetic retinopathy from eye retina images.

1. To collect a large dataset of eye retina images.
2. To pre-process the images to remove noise and enhance the contrast.
3. To train a deep learning model using a convolutional neural network.
4. To evaluate the performance of the model using metrics such as accuracy, precision, and recall.
5. To deploy the model to a web application or a mobile app, where users can upload their retina images and receive a prediction of whether they have diabetic retinopathy.

CHAPTER 2
LITERATURE SURVEY

Lam et al. have developed an automated system for DR detection using 243 retinal images from Kaggle's EyePACS dataset, and have generated 1324 image patches to detect HEs, MAs, EXs, NV from normal structures. The CNN model is trained on 1050 image patches and tested on 274 image patches. The model has used a 128 × 128 × 3 patch-trained GoogleLeNet-v1 CNN sliding window to scan the image patches and generate a probability score for each of the five classes of DR through detection of MAs and HEs. The model has compared the performance of AlexNet, VGG16, GoogLeNet, ResNet, and Inception-v3 on the 274 test patches and has achieved five-class binary classification accuracy of 74% and 79%, 86% and 90%, 95% and 98%, 92% and 95%, and 96% and 98%, respectively.

Pratt et al. have proposed a CNN architecture to extract DR features using the Kaggle dataset which comprises of 80,000 images, for the detection of DR. The model has used color normalization for preprocessing, real-time data augmentation, L2-regularization for updating weights and biases and cross-entropy loss function for optimization. The proposed CNN is trained using Stochastic Gradient Descent (SGD) and has achieved a specificity of 95% and an accuracy of 75% and sensitivity of 30%.

Xu et al. have proposed a model which uses label preserving transformation for data augmentation and Deep Convolutional Neural Network (DCNN) based image classification, for the detection of DR, using Kaggle's dataset. The proposed methodology has used two classifiers in which one combines each of the extracted features with the Gradient Boosting Machines (GBM) - eXtreme Gradient Boosting method (XGBoost) and the other classifier uses CNN-based features, with and without data augmentation. The entire network is

optimized using backpropagation and Stochastic Gradient Descent (SGD). The proposed methodology has detected hard EXs, red lesions, MAs and RBVs. The proposed methodology has obtained an accuracy of 89.4% for hard EXs and GBM, 88.7% for red lesions and GBM, 86.2% for MAs and GBM, 79.1% for RBVs and GBM, 91.5% for CNN without data augmentation and 94.5% for CNN with data augmentation. It is observed that DL models perform better with CNN-extracted features in comparison to conventional feature extraction methods with data augmentation.

Khojasteh et al. have proposed a ten-layered CNN and employs patch-based and image-based analysis upon the fundus images, for the detection of DR. The model has used a total of 284 retinal images from DIARETDB1 and e-ophtha datasets, of which 75 images from DIARETDBI dataset are used for training, for patch-based analysis and the remaining 209 images, both from DIARETDB1 and e-ophtha, are used for testing, for image-based analysis. The model has performed contrast enhancement to extract EXs, HEs and MAs, and then segmented patches of size 50X50 to obtain rule-based probability maps. During the patch-based analysis, the model has detected EXs with sensitivity, specificity and accuracy of 0.96, 0.98 and 0.98, HEs with sensitivity, specificity and accuracy of 0.84, 0.92 and 0.90, and MAs with sensitivity, specificity and accuracy of 0.85, 0.96 and 0.94, respectively. In the image level evaluation, the proposed method has achieved an accuracy of 0.96, 0.98 and 0.97 with error rate of 3.9%, 2.1% and 2.04% for segmentation of EXs, HEs and MAs, respectively on DIARETDB1 test set, and an accuracy of 0.88, and 3.0, and error rate of 4.2% and 3.1%, for EXs and MAs, respectively, on e-Ophtha dataset. It is observed that simultaneous detection of features can reduce potential error more accurately than individual detection, without any redundancy. It is also observed that post-processing has reduced the error rate and image patching has improved the quality of the images through consideration of the neighborhood and background of candidate lesions.

Soniya et al. have proposed two CNN models 1 and 2, which consists of a single CNN and heterogeneous CNN modules, trained using gradient

descent and backpropagation respectively, and are compared to evaluate the effectiveness of detecting DR, using DIARETDB0 dataset. The CNN model 1 has identified MAs, HEs, hard EXs and soft EXs and the CNN model 2 has identified NV. The proposed model has used a multilayer perceptron network classifier with 1620-10-5-6 architecture whose output corresponds to six classes as class 1, class2, class 3, class 4, class 5, and class 6 for normal images, MAs, HEs, hard EXs, soft EXs and NV, respectively. The model has used 130 color images from DIARETDB0 dataset and has performed four experiments of which three experiments have used single CNN with different filter size and receptive field and the fourth experiment has employed heterogenous CNN modules. On using single CNN for the first two experiments, with three convolutional layers each, having 10–30-30 filters for the former, and 30–30-10 filters for the latter, the model has achieved accuracies of 95%, 65%, 42.5%, 67.5% and 92.5% for the former, and accuracies of 95%, 75%, 62.5%, 65% and 95% for the latter, for detection of MAs, HEs, hard EXs, soft EXs and NV respectively. On using single CNN in the third experiment with 50–70–80-100-200 filters, the model has achieved accuracies of 75%, 77.5%, 70%, 52.5%, and 95% for MAs, HEs, hard EXs, soft EXs and NV respectively. The heterogenous CNN modules introduced have achieved 100% accuracy for extraction and detection of class specific features in comparison to the single CNN which has continuously shown low sensitivity and specificity values. It is observed that slight modifications in the filters of CNN, have enhanced the performance of detection of DR lesions. It is also observed that heterogenous CNN has performed better than single CNN.

Alghamdi et al. have proposed an end-to-end supervised model for OD abnormality detection, which constitutes two successive DL architectures with integrated cascade CNN classifiers and abnormality assessment through feature learning, respectively, for the detection of DR. The model has used AdaBoost ensemble algorithm for feature selection and training of the classifier. The proposed approach has used a total of 5781 images from datasets such as DRIVE, DIARETDB1, MESSIDOR, STARE, KENYA, HAPIEE, PAMDI and KFSH. The model has used the annotated OD images in PAMDI and HAPIEE, to

train and evaluate the abnormality detector. The model has achieved an accuracy of 100%, 98.88%, 99.20%, 86.71%, 99.53%, 98.36%, 98.13% and 92%, on DRIVE, DIARETDB1, MESSIDOR, STARE, KENYA, HAPIEE, PAMDI and KFSH respectively, for OD localization. The proposed OD abnormality detector has achieved a sensitivity of 96.42%, a specificity of 86% and an accuracy of 86.52% on HAPIEE dataset, and a sensitivity of 94.54%, a specificity of 98.59% and an accuracy of 97.76% on PAMDI dataset. It is observed that the use of *cascade classifiers* which is an ensemble of weak classifiers can work well with good quality images only and cannot withstand variations and hence requires learning of discriminative features using CNN.

Amongst all these, SVM, Neural Networks (NN), hybrid ML-DL models and ensemble algorithms have produced effective results upon an effective dataset. ML techniques exhibit high generalization error and presents only sub-optimal solutions, and are incompetent against any real-time, complex, and high-dimensional data such as medical images (here fundus images). Moreover, they lack domain awareness and representation, which makes them computationally intensive and inflexible to extract patterns and relationships using handcrafting rules and algorithms upon high-dimensional image data.

Using deep learning CNN Deep Learning (DL) is a new advent of ML and inherits the appropriate and advantageous attributes of ML such as perform complex tasks, smart and automated, better generalization, domain knowledge, decision making etc. and efficiently applies them upon image data, thereby outperforming shallow ML algorithms. DL permits data processing of diverse types in amalgamation, known as cross-modal learning (multiple forms of representations), and can generate well-defined features through automated feature learning, unlike ML algorithms which are dependent on various feature extraction algorithms and procedures. DL models have the capability to learn and to generate new features from extracted and existing features such as points, lines, edges, gradients, vessel structure, corners, boundaries etc. using representation learning.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

Functional requirements indicate what a software system must do and how it must function, they are product features that focus on user needs.

- Image acquisition: The system should be able to acquire high-quality eye retina images from a camera or other imaging device.
- Image preprocessing: The system should be able to preprocess the images to enhance their quality and remove any noise or artifacts.
- Deep learning model: The system should implement a deep learning model, such as a convolutional neural network (CNN), to classify the images as either diabetic retinopathy or non-diabetic retinopathy.
- Training dataset: The system should have access to a large and diverse dataset of labeled retina images for training the deep learning model.
- Validation dataset: The system should have access to a separate dataset of labeled retina images for validation and testing purposes.
- Performance metrics: The system should use appropriate performance metrics, such as accuracy, precision, recall, and F1-score, to evaluate the performance of the deep learning model.
- User interface: The system should have a user-friendly interface that allows users to input retina images and receive a diagnosis for diabetic retinopathy.

3.2 NON-FUNCTIONAL REQUIREMENTS

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of asystem,rather than specific behaviors.

- Performance: The system should be able to process retina images quickly and provide a diagnosis in a timely manner.

- Accuracy: The system should have a high level of accuracy in detecting diabetic retinopathy to ensure reliable diagnoses.
- Scalability: The system should be designed to handle large volumes of data and be scalable to meet increasing demand.
- Reliability: The system should be reliable and available at all times to avoid any potential harm to patients.
- Usability: The system should be user-friendly and easy to use for healthcare professionals, regardless of their technical skills.
- Compatibility: The system should be compatible with various types of imaging devices and platforms to allow for widespread adoption.
- Security: The system should ensure the security of patient data and comply with relevant regulations and standards.
- Privacy: The system should ensure patient privacy and confidentiality in accordance with regulations and standards.
- Maintainability: The system should be easy to maintain, update, and troubleshoot to minimize downtime and ensure optimal performance.
- Performance efficiency: The system should be designed to use computing resources efficiently and optimize system performance.

3.2.1 Hardware Requirements

The most common set of requirements defined by an operating system or software application is the physical computer resources, also known as hardware, a hardware requirements list is often accompanied by a hardware compatibility list, especially in case of operating systems.

Listed below are some hardware requirements:

1. Processor: Intel corei3 10^{th} gen
2. RAM: Minimum 4GB
3. Hard disk: Minimum 20GB

4. Input devices: Keyboard and Mouse
5. Output devices: Monitor

3.2.2 Software Requirements

The Software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's points of view.

Listed below are some software requirements:

1. Operating system: Windows 7 or above
2. Programming Language: Python
3. Version: 3.9 or higher
4. IDE: Jupyter Notebook
5. GUI: Anaconda Navigator

3.3 TECHNOLOGY DESCRIPTION

3.3.1 PYTHON 3.9

Python 3.9 is Python Version. It is a high level programming language. Most of the research use it to do their research. It is highly recommended programming language for AI based work and it is very popular among new generation's programmer because it is very easy to learn and understand.

3.3.2 ANACONDA

Anaconda is a distribution of the python and R programming languages for scientific computing, that aims to simply package management and deployment. The distribution includes data science packages suitable for windows, linux and macOs.

3.3.3 TENSORFLOW

TensorFlow is an open-source software library for machine learning and artificial intelligence developed by Google Brain Team.It provides a range of tools and resources for building and training machine learning models, including neural

networks, deep learning models, and other types of models. TensorFlow is highly scalable and can be used on a wide range of platforms, including desktop computers, mobile devices, and cloud-based systems. It supports multiple programming languages, including Python, C++, and Java, making it accessible to a wide range of developers and researchers. TensorFlow has been used in a wide range of applications, including natural language processing, image and video recognition, and predictive analytics.

3.3.4 KERAS

Keras is a high-level, open-source neural network library written in Python. It is built on top of other popular libraries like TensorFlow and Theano and is designed to simplify the process of building, training, and deploying deep learning models. In this essay, we will discuss the features, benefits, and various layers of Keras.

Features of Keras:

Keras provides a simple and intuitive API for building deep learning models. It offers a range of pre-built layers, such as convolutional, pooling, and recurrent layers, that can be easily combined to create complex models. Keras also provides a range of loss functions, activation functions, and optimizers that can be used to train and fine-tune deep learning models.

- Simple and intuitive API
- Pre-built layers for easy model building
- Support for a range of loss functions, activation functions, and optimizers
- Portability and compatibility with TensorFlow

Benefits of Keras:

Keras is highly powerful and dynamic framework and comes up with the following advantages-

- Ease of use and minimal coding required
- Excellent documentation and community support

- Fast and efficient model building and training
- Versatility and applicability to a wide range of deep learning applications
- Ability to handle large amounts of data and complex models
- Highly scalable and can be used on a range of platforms and devices

Various Layers of Keras:

1. **Input Layer:**

   The input layer is the first layer of a neural network and is used to specify the input shape of the data.

2. **Dense Layer:**

   The dense layer is a fully connected layer that performs a linear operation on the input data. It is commonly used in deep learning models for classification and regression tasks.

3. **Convolutional Layer:**

   The convolutional layer is used for processing images and other multidimensional data. It performs a convolution operation on the input data, which helps to extract features from the data.

4. **Pooling Layer:**

   The pooling layer is used to down sample the output of the convolutional layer by reducing the spatial dimensions of the data. This helps to reduce the number of parameters in the model and prevents overfitting**.**

5. **Recurrent Layer:**

   The recurrent layer is used for processing sequential data, such as time-series data or natural language data. It maintains an internal state that is updated for each time step, allowing the model to learn long-term dependencies in the data.

3.3.5 NUMPY

NumPy is a powerful Python library used for scientific computing and data analysis. It provides a multidimensional array object and a range of functions for working with arrays. With NumPy, it is easy to perform complex mathematical operations on arrays, including linear algebra, Fourier transforms, and random number generation. NumPy is also highly efficient, making it ideal for handling large datasets and complex computations.

3.3.6 MATPLOTLIB

Matplotlib is a popular data visualization library in Python that provides a range of tools for creating high-quality plots, charts, and graphs. It can be used to create a wide range of visualizations, including line plots, scatter plots, histograms, bar charts, and more. Matplotlib is highly customizable and can be used to create publication-quality figures for scientific publications or presentations. It also provides seamless integration with NumPy and Pandas, making it an essential tool in the data science and machine learning ecosystem.

3.3.7 PANDAS

Pandas is a powerful Python library used for data manipulation and analysis. It provides a range of data structures for working with tabular and time-series data, including the DataFrame and Series objects. With Pandas, it is easy to read and write data to and from a variety of sources, including CSV files, Excel spreadsheets, SQL databases, and more. Pandas also provides a range of functions for data cleaning, filtering, and transformation, making it a valuable tool for data scientists and analysts**.**

3.3.8 SEABORN

Seaborn is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn offers a range of built-in themes and color palettes that can be used to customize the visual appearance of plots. It provides a range of visualization types, including heatmaps, histograms, scatter plots, and more. Seaborn is also capable of handling complex data structures, such as wide-form data and time-series data.

With Seaborn, it is easy to create beautiful and informative visualizations for data exploration and presentation.

3.3.9 JUPYTER NOTEBOOK

Jupyter Notebook is a web-based interactive computing environment that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It is a popular tool among data scientists, researchers, and educators, as it provides an easy way to write and execute code in various programming languages, including Python, R, and Julia. Jupyter Notebook provides a range of features, including syntax highlighting, inline plotting, interactive widgets, and more. It can also be used for collaborative work and can be shared with others in various formats, including HTML, PDF, and Markdown.

3.3.10 DEEP LEARNING

Deep Learning is a subfield of machine learning ,that is inspired by the structure and function of human brain. It involves training artificial neural networks that are composed of many layers of interconnected nodes or neurons, to preform many such tasks such as image recognition. Deep learning models learn by analyzing large amounts of data, and extract features from them.

3.3.11 TRANSFER LEARNING

It is a machine learning technique, that involves using a pre-trained model on a large dataset for a new,similar task.It involves taking a neural network ,that has already been trained on a large dataset ,such as IMAGENET,and reusing its learned features and weights to solve a different problem.The idea behind transfer learning is that learned features and weights of pre-Trained model can be useful for a new task ,as they have already been trained on a large dataset.Rather than training a new model from scratch,transfer learning enables us to leverage the power of a pre-trained model and fine tune it for a specific task,to reduce the amount of training data and time required for the new model to achieve high accuracy.

3.3.12 RESNET-50

ResNet-50 is a deep convolutional neural network architecture,that was introduced in 2015 by researchers at Microsoft Research.It was designed to address the problem of vanishing gradients that occur in very deep neural networks.It consists of 50 layers and uses residual connections ,to enable the gradient to flow more easily through the network during training.

CHAPTER 4

4.1 SYSTEM ARCHITECTURE

This project's system architecture begins by collecting a dataset of retinal images from diabetic patients, followed by preprocessing to normalize the images' size and color. Gaussian blur is applied to remove any artifacts or noise. Data augmentation techniques such as rotations, translations, scaling, and flipping are applied to increase the dataset's diversity. The dataset is split into training, validation, and testing sets. A ResNet50 model is used as a feature extractor, and a CNN model is built to classify the images based on the extracted features. The CNN model is trained using the extracted features as inputs and the ground truth labels as outputs. Hyperparameter tuning is conducted to fine-tune the model's performance, and the model is deployed in a production environment.

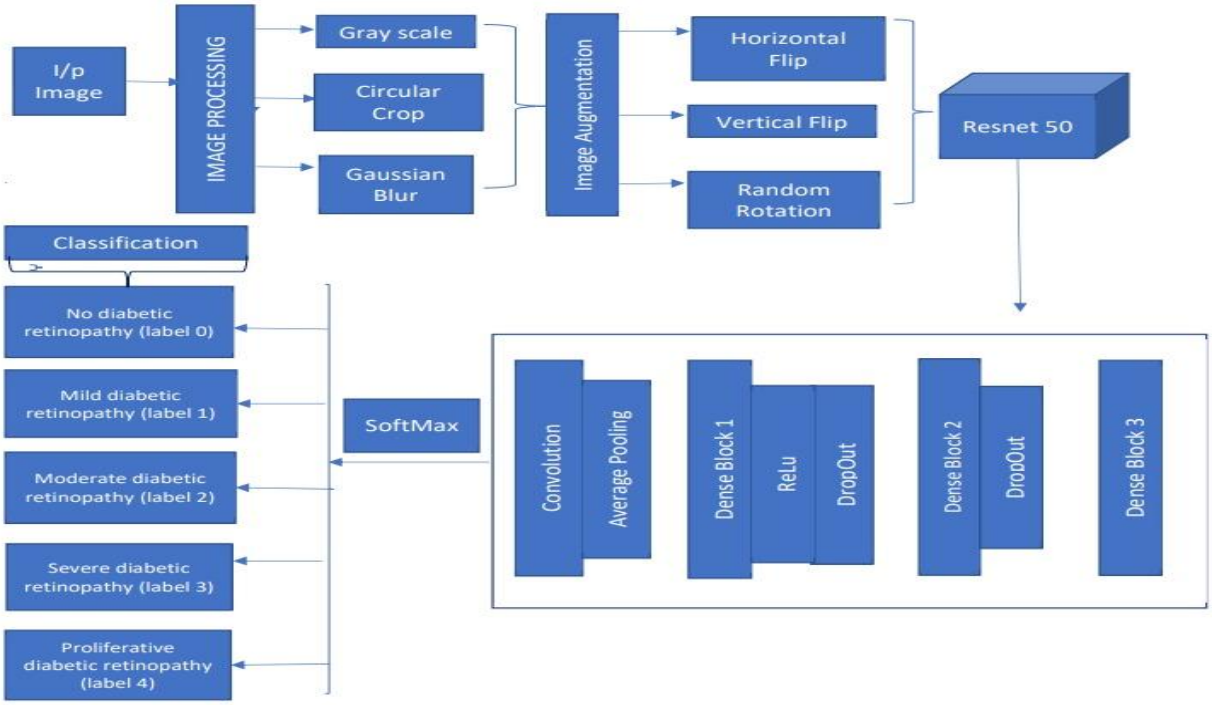


Fig 1 -system architecture

4.2 METHODOLOGY

4.2.1 EXISTING SYSTEM:

Existing system for diabetic retinopathy involves a lot of tedious and human work. To diagnose DR at an early stage, manual methods such as bio-microscopy, retinal imaging of the fundus, Retinal Thickness Analyzer (RTA), Scanning Laser Ophthalmoscopy (SLO), Adaptive Optics, Retinal Oximetry, Optical Coherence Tomography (OCT), OCT Angiography, Doppler OCT, and many more can be adopted. However, such conventional methods for manually analyzing the disease makes it cumbersome, time consuming and highly prone to error. Besides, it demands a sophisticated task force which is sometimes not feasible w.r.t (with respect to) prevailing circumstances. Thus, it is not feasible to perform manual diagnosis for early detection of DR at any time and at any place. The present ratio of Ophthalmologists to patient especially in India is 1:10000,and in such a situation, the need of an automated intelligent detection system for primary analysis of early signs of DR is realized. Thus, a faster and a revolutionary method, proposing an intelligent system is necessitated.

4.2.2 PROPOSED SYSTEM:

We propose the deep learning approach to detect diabetic retinopathy. Deep Learning (DL) is a new advent of ML which can perform automated and complex tasks, discover unseen insights, highly scalable, better domain knowledge, reliable decision making etc and efficiently applies them upon high-dimensional data, thereby outperforming shallow ML models. DL methods such as Convolutional Neural Network (CNN) , Deep Convolutional Neural Network (DCNN), and Deep Neural Network (DNN/DLNN) architectures such as AlexNet, Residual Network (ResNet) and its variants etc. are proposed for deep feature extraction and image classification. DR detection is highly dependent on assessment and analysis of fundus images. The presence of various DR lesions can be identified using high resolution fundus images. Based on the presence and

absence of DR retinal lesions and the corresponding severity level of the disease, this system classifies DR into five categories as mentioned below.

1.No diabetic retinopathy (label 0)

2. Mild diabetic retinopathy (label 1)

3. Moderate diabetic retinopathy (label 2)

4.Severe diabetic retinopathy (label 3)

5.Proliferative diabetic retinopathy (label 4)

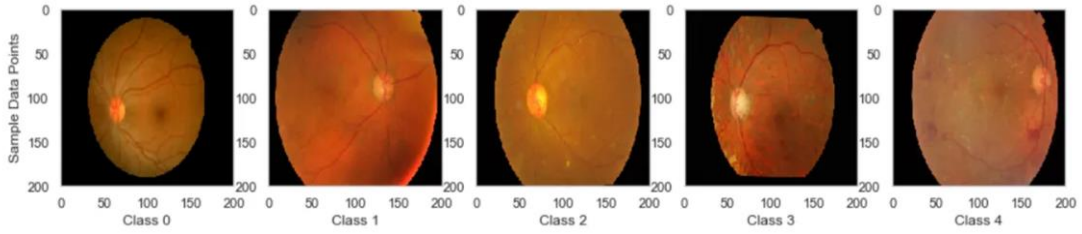


Fig 2 – Sample image for each class

This system is includes following steps.

1. Data collection: Collect a large dataset of retinal images from diabetic patients, including both normal and diseased images.
2. Data preprocessing: Preprocess the images to normalize their size and color, and remove any artifacts or noise. Apply Gaussian blur to smooth out any minor variations in the images.
3. Data augmentation: Augment the images by applying various transformations such as rotations, translations, scaling, and flipping. This helps increase the diversity of the dataset and improves the robustness of the model.
4. Split the data: Split the dataset into training, validation, and testing sets.
5. Transfer learning: Use a pre-trained ResNet50 model as a feature extractor, and freeze its layers. Extract the features from the augmented dataset using the ResNet50 model.
6. CNN model: Build a CNN model to classify the retinal images based on the extracted features. The CNN model can consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

7. Training: Train the CNN model using the extracted features as inputs, and the ground truth labels as outputs. This involves initializing the model, defining the loss function, optimizing the model parameters, and evaluating the model's performance on the validation set.
8. Hyperparameter tuning: Experiment with different hyperparameters such as learning rate, batch size, and number of epochs to fine-tune the model's performance.
9. Testing: Evaluate the trained model on the testing set to measure its accuracy, precision, recall, and F1 score.
10. Deployment: Deploy the model in a production environment, such as a mobile app or web service, where it can be used to classify retinal images as normal or diseased.

CHAPTER 5

IMPLEMENTATION

5.1 IMPLEMENTATION

5.1.1 SAMPLE CODE

Importing the dependencies:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import multiprocessing
from multiprocessing.pool import ThreadPool
from sklearn.metrics import confusion_matrix,
cohen_kappa_score,accuracy_score
from PIL import Image
import cv2
import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers,Model,Sequential
from keras.layers import
Input,GlobalAveragePooling2D,Dropout,Dense,Activation
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
from keras.applications.resnet import ResNet50
from tensorflow.keras.layers import Input, Dense, Dropout,
GlobalAveragePooling2D,Conv2D,MaxPooling2D
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.models import Model
```

Data Collection and PreProcessing:

We used csv files provided along with the eye retina images folder,which has the data about the image ID and corresponding diagnosis column in train.csv and image ID of the retina,which are to be predicted.To make preprocessing easier,we imported these csv files as DataFrames,and added few other column ,which makes retrieval of image information easier.

Later,we imported this data in 2 dataframes df_train and df_test,and we have checked the integrity of the dataframes using top 6 columns of df_train.

```
def load_data():
    train = pd.read_csv('train.csv')
    test = pd.read_csv('test.csv')

    train_dir = os.path.join('./','train_images/') #why separate joining
    test_dir = os.path.join('./','test_images/')

    train['file_path'] = train['id_code'].map(lambda x: os.path.join(train_dir,'{}.png'.format(x))) # maps the columns
    test['file_path'] = test['id_code'].map(lambda x: os.path.join(test_dir,'{}.png'.format(x)))

    train['file_name'] = train["id_code"].apply(lambda x: x + ".png") #updating file_name column
    test['file_name'] = test["id_code"].apply(lambda x: x + ".png")

    train['diagnosis'] = train['diagnosis'].astype(str) #type casting to string

    return train,test
```

```
df_train,df_test = load_data() # calling load_data() function
print(df_train.shape,df_test.shape,'\n') #no. of rows
df_train.head(6) #get top 6 rows to verify the changes
```

Output:

```
def plot_classes(df):
    df_group = pd.DataFrame(df.groupby('diagnosis').agg('size').reset_index())
    df_group.columns = ['diagnosis','count']
    sns.set(rc={'figure.figsize':(10,5)}, style = 'whitegrid')
    sns.barplot(x = 'diagnosis',y='count',data = df_group,palette = "Blues_d")
    plt.title('Output Class Distribution')
    plt.show()
plot_classes(df_train)
```

```
(3662, 4) (1928, 3)
```

	id_code	diagnosis	file_path	file_name
0	000c1434d8d7	2	./train_images/000c1434d8d7.png	000c1434d8d7.png
1	001639a390f0	4	./train_images/001639a390f0.png	001639a390f0.png
2	0024cdab0c1e	1	./train_images/0024cdab0c1e.png	0024cdab0c1e.png
3	002c21358ce6	0	./train_images/002c21358ce6.png	002c21358ce6.png
4	005b95c28852	0	./train_images/005b95c28852.png	005b95c28852.png
5	0083ee8054ee	4	./train_images/0083ee8054ee.png	0083ee8054ee.png

Class Distribution:

We tried to plot the class distribution of the labels and seen that the images are more of type 0 DR,and to compensate this imbalance ,we used **data augmentation**

Output:

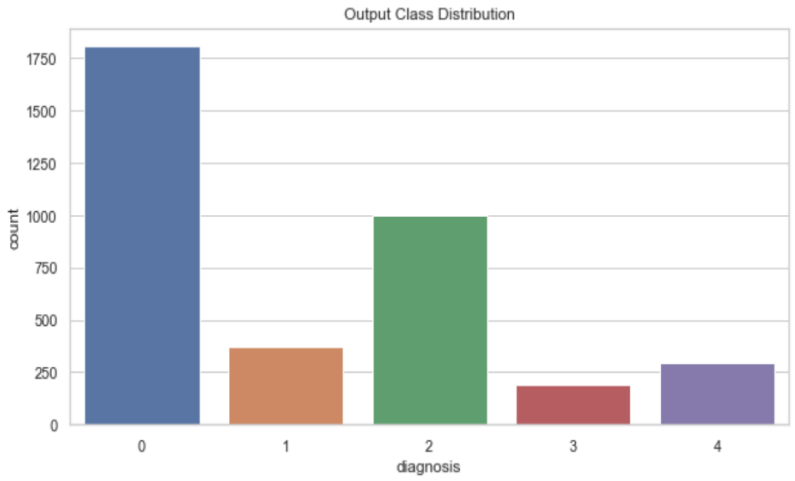


Fig 3- Class Distribution

Visualization of Images:

Here,we tried to visualize 3 rows 5 retina images per class in original color and gray scale

```
IMG_SIZE = 3000
```

```
def conv_gray(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    return img
```

```
def visualize_imgs(df,pts_per_class,color_scale):
    df = df.groupby('diagnosis',group_keys = False).apply(lambda df: df.sample(pts_per_class))
    df = df.reset_index(drop = True)
    plt.rcParams["axes.grid"] = False
    for pt in range(pts_per_class):
        f, axarr = plt.subplots(1,5,figsize = (15,15))
        axarr[0].set_ylabel("Sample Data Points")
        df_temp = df[df.index.isin([pt + (pts_per_class*0),pt + (pts_per_class*1), pt + (pts_per_class*2),pt +
                                    (pts_per_class*3),pt + (pts_per_class*4)])]
        for i in range(5):
            if color_scale == 'gray':
                img = conv_gray(cv2.imread(df_temp.file_path.iloc[i]))
                axarr[i].imshow(img,cmap = color_scale)
            else:
                axarr[i].imshow(Image.open(df_temp.file_path.iloc[i]).resize((IMG_SIZE,IMG_SIZE)))
            axarr[i].set_xlabel('Class '+str(df_temp.diagnosis.iloc[i]))
```

```
visualize_imgs(df_train,3,color_scale = None)
```

Output:

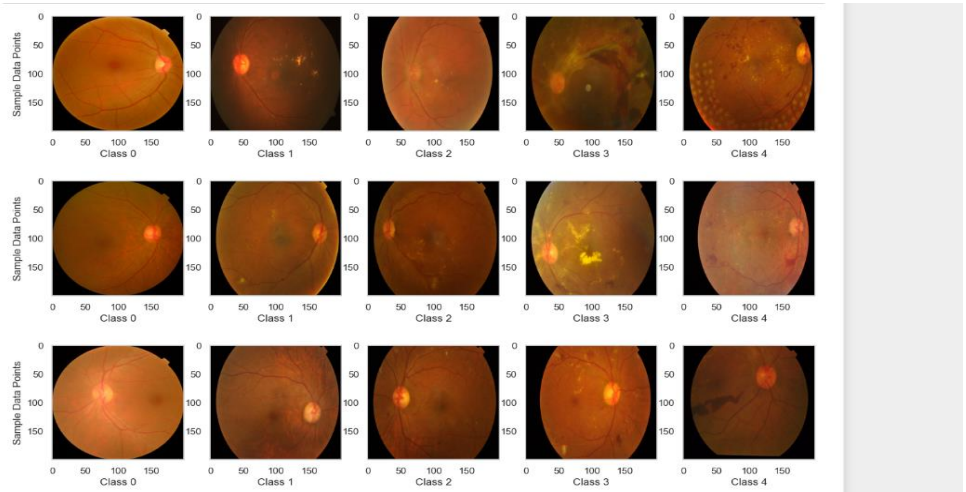


Fig 4-Images before preprocessing

```
visualize_imgs(df_train,2,color_scale = 'gray')
```

Output:

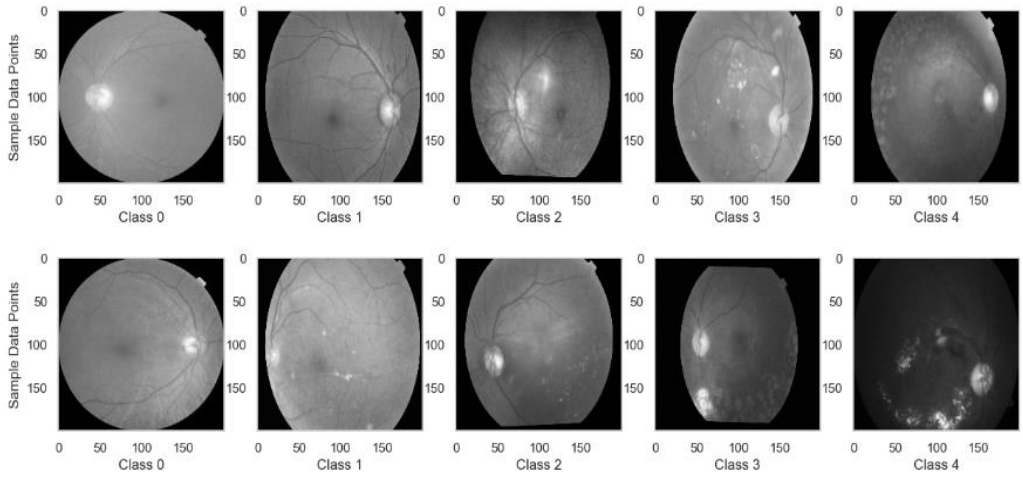


Fig 5-Images after grayscale

Image Processing:

To extract the features from the retina images,we preprocessed given images using:

1.Gaussian Blur:It is a method of blurring an image by convolving the image with a gaussian filter.The blur effect is achieved by averaging the colors of the pixels in the image.

2.Circular cropping:We found that the size of an image can further reduced by focusing only on the circular retina part from the image.So we tried to crop the image circularly ,which reduced the size of image to some extent.

```
def crop_image_from_gray(img,tol=7):#tol ->tolerance
    if img.ndim ==2: #since image is an array of values.so given image is of 2D(Black&white)
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))] #Using ix_ one can quickly construct index arr
            #vectorization                                  #a[np.ix_([1,3],[2,5])] returns th
            # look for rows and columns that have at least one pixel along rows and columns that i
            # you can use these boolean arrays to index into the image data for extraction of vali
    elif img.ndim==3:#given image is of 3D.now ,we need to convert it into gray
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) #RBG->Gray
        mask = gray_img>tol

        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0] #?
        if (check_shape == 0): # image is too dark so that we crop out everything,
            return img # return original image
        else:
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
    #         print(img1.shape,img2.shape,img3.shape)
            img = np.stack([img1,img2,img3],axis=-1)
    #         print(img.shape)
        return img
```

```
lef circle_crop(img, sigmaX=30):
    """
    Create circular crop around image centre
    """
    img = crop_image_from_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, depth = img.shape    #setting height, width, depth based on shape of image

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y)) #returns min of array(radius)

    circle_img = np.zeros((height, width), np.uint8) #?
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=circle_img)
    img = crop_image_from_gray(img)
    img=cv2.addWeighted(img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img
```

```
'''Perform Image Processing on a sample image'''

rn = np.random.randint(low = 0,high = len(df_train) - 1) #selecting random image

img = cv2.imread(df_train.file_path.iloc[rn]) #reading image
img_t = circle_crop(img,sigmaX = 30) #circular crop of image

f, axarr = plt.subplots(1,2,figsize = (11,11)) #subplot each of dimension 11*11 and 1 ro
axarr[0].imshow(cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2RGB),(IMG_SIZE,IMG_SIZE)))
axarr[1].imshow(img_t)
plt.title('After applying Circular Crop and Gaussian Blur')
plt.show()
```

Output:

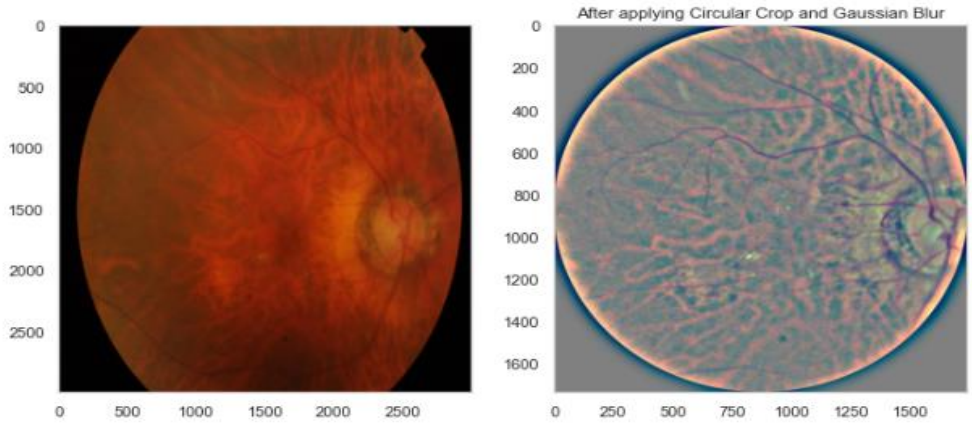


Fig 6 -Image after processiong

Data Augmentation:

We displayed augmented images from a given random image

```
from keras.preprocessing.image import ImageDataGenerator

'''This Function generates 'lim' number of Image Augmentations from a random Image in the directory'''
 #allows your model to receive new variations of the images at each epoch
# only returns the transformed images and does not add it to the set of images that you have.
def generate_augmentations(lim):
    datagen = ImageDataGenerator(featurewise_center=True,            #Set input mean to 0 over the dat
                                 featurewise_std_normalization=True, #Divide inputs by std dev of the
                                 rotation_range=20,                   #randomly rotates the image up t
                                 width_shift_range=0.2,               #20 times vertical shift
                                 height_shift_range=0.2,              #20 times horizontal shift
                                 horizontal_flip=True)                #horizontal flip of selected ima
    img = cv2.imread(df_train.file_path.iloc[np.random.randint(low = 0,high = len(df_train) - 1)])#sel
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.title('ORIGINAL IMAGE')
    plt.show()

    img_arr = img.reshape((1,) + img.shape)
    i = 0
    for img_iterator in datagen.flow(x = img_arr,batch_size = 1):#img_Arr?
        i = i + 1
        if i > lim:
            break
        plt.imshow((img_iterator.reshape(img_arr[0].shape)).astype(np.uint8)) #uint8:unsigned int 8-bi
        plt.title('IMAGE AUGMENTATION ' + str(i))
        plt.show()
```

Output

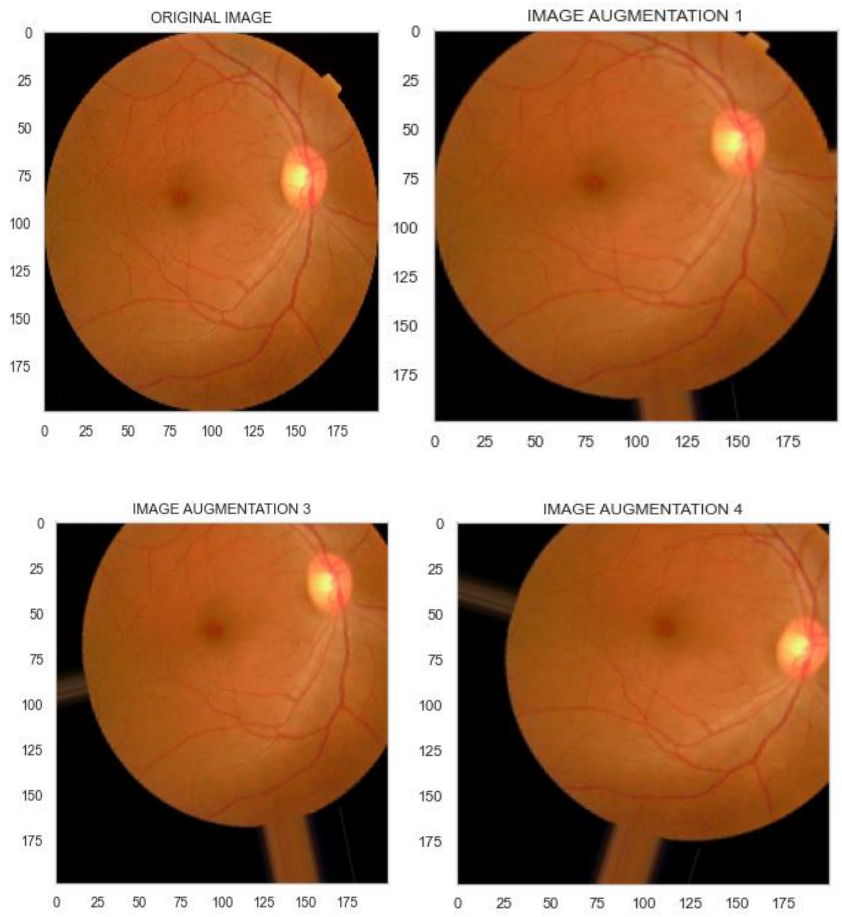


Fig 7-Data Augumentation Images

Image Resizing:

We have splitted df_train dataframe into df_train_train and df_train_valid,and preserved them for model building,by converting them into excels.

Here,we resized the images and separated them into train_images_resized and valid_images_resized.

```
from sklearn.model_selection import train_test_split
df_train_train,df_train_valid = train_test_split(df_train,test_size = 0.2)
print(df_train_train.shape,df_train_valid.shape)
```

```
(2929, 4) (733, 4)
```

```
df_train_train.to_excel('df1.xlsx')
df_train_valid.to_excel('df2.xlsx')
```

```
def image_resize_save(file):
    input_filepath = os.path.join('./','train_images','{}.png'.format(file))
    output_filepath = os.path.join('./','valid_images_resized','{}.png'.format(file))
    img = cv2.imread(input_filepath)
    cv2.imwrite(output_filepath, cv2.resize(img, (IMG_SIZE,IMG_SIZE)))
```

```
import multiprocessing
from multiprocessing.pool import ThreadPool
'''This Function uses Multi processing for faster saving of images into folder'''

def multiprocess_image_downloader(process:int, imgs:list):
    """
    Inputs:
        process: (int) number of process to run
        imgs:(list) list of images
    """
    print(f'MESSAGE: Running {process} process')
    results = ThreadPool(process).map(image_resize_save, imgs)
    return results
```

```
multiprocess_image_downloader(6, list(df_train_valid.id_code.values))
```

output:

```
MESSAGE: Running 6 process

Out[25]: [None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
          None,
```

In the same way we have to resize test images, train images to test_images_resized, train_images_resized folders.

Image Cropping and Circular Cropping:

We now preprocessed the resized images using gaussian blur and circular cropping and saved them into train_images_resized_preprocessed and valid_images_resized_preprocessed.

```
def crop_image_from_gray(img,tol=7):#tol ->tolerance
    if img.ndim ==2: #since image is an array of values.so given image is of 2D(Black&white)
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))] #Using ix_ one can quickly construct index arr
            #vectorization                                  #a[np.ix_([1,3],[2,5])] returns th
            # look for rows and columns that have at least one pixel along rows and columns that i
            # you can use these boolean arrays to index into the image data for extraction of vali
    elif img.ndim==3:#given image is of 3D.now ,we need to convert it into gray
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) #RBG->Gray
        mask = gray_img>tol

        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0] #?
        if (check_shape == 0): # image is too dark so that we crop out everything,
            return img # return original image
        else:
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
    #         print(img1.shape,img2.shape,img3.shape)
            img = np.stack([img1,img2,img3],axis=-1)
    #         print(img.shape)
        return img
```

```
def circle_crop(img, sigmaX=30):
    """
    Create circular crop around image centre
    """
    img = crop_image_from_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, depth = img.shape    #setting height, width, depth based on shape of image

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y)) #returns min of array(radius)

    circle_img = np.zeros((height, width), np.uint8) #?
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=circle_img)
    img = crop_image_from_gray(img)
    img=cv2.addWeighted(img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img
```

```
def preprocess_image(file):
    input_filepath = os.path.join('./','valid_images_resized','{}.png'.format(file))
    output_filepath = os.path.join('./','valid_images_resized_preprocessed','{}.png'.format(file))

    img = cv2.imread(input_filepath)
    img = circle_crop(img)
    if img is None:
        print(input_filepath)
    cv2.imwrite(output_filepath, cv2.resize(img, (IMG_SIZE,IMG_SIZE)))
```

```
def multiprocess_image_processor(process:int, imgs:list):
    """
    Inputs:
        process: (int) number of process to run
        imgs:(list) list of images
    """
    print(f'MESSAGE: Running {process} process')
    results = ThreadPool(process).map(preprocess_image, imgs)
    return results
```

```
multiprocess_image_processor(6, list(df_train_valid.id_code.values))
```

In the same way, apply image cropping and circular cropping on resized train images, test images.

5.1.2.1 MODEL BUILDING (MODEL-1,USING PRE-TRAINED RESNET LAYERS ONLY)

Model parameters:

BATCH_SIZE = 8

EPOCHS = 40

WARMUP_EPOCHS = 2

LEARNING_RATE = 1e-4

WARMUP_LEARNING_RATE = 1e-3

HEIGHT = 320

WIDTH = 320

CHANNEL = 3

N_CLASSES = df1['diagnosis'].nunique()

ES_PATIENCE = 5

RLROP_PATIENCE = 3

DECAY_DROP = 0.5

Steps in model creation:

1.Load data

2.Define model

3.Model compilation

4.Model training

5.Making Predictions

6.Model Evaluation

1.Load Data

Using ImageDataGenerator,we augmented the images to compensate the imbalance present in the given dataset.

```
import keras
from keras.preprocessing.image import ImageDataGenerator#The ImageDataGenerator class allows your model to rece
                                                        #new variations of the images at each epoch.(Data Augme
                                                    #(only returns transformed images,but not adds to existing set
HEIGHT = 320
WIDTH = 320
BATCH_SIZE = 8
N_CLASSES = df1['diagnosis'].nunique()
def img_generator(train,test):
    train_datagen=ImageDataGenerator(rescale=1./255, validation_split=0.2,horizontal_flip=True)
            #rescale- used to rescale pixel values from the range of 0-255 to the range 0-1 preferred for neur

    train_generator=train_datagen.flow_from_dataframe(dataframe=train,  # loads the image dataset in memory and
                                                      directory="./train_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      y_col="diagnosis",
                                                      batch_size=BATCH_SIZE,#produce BATCH_SIZE no.of images in
                                                      class_mode="categorical",#since,more than 2 classes to pr
                                                      target_size=(HEIGHT, WIDTH),
                                                      subset='training',#validate_filenames=False
                                                     ) # set as training data
    #no valid dataset,so create one
    valid_generator=train_datagen.flow_from_dataframe(dataframe=train,   #loads the image dataset in memory and
                                                      directory="./train_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      y_col="diagnosis",
                                                      batch_size=BATCH_SIZE,#produce BATCH_SIZE no.of images in
                                                      class_mode="categorical",#since,more than 2 classes to pr
                                                      target_size=(HEIGHT, WIDTH),
                                                      subset='validation',
                                                     )# set as validation data
```

```
    test_datagen = ImageDataGenerator(rescale=1./255)
    test_generator = test_datagen.flow_from_dataframe(dataframe=test,  # loads the image dataset in memory and gene
                                                      directory = "./valid_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      target_size=(HEIGHT, WIDTH),
                                                      batch_size=1,
                                                      shuffle=False,
                                                      class_mode=None,
                                                     )

    return train_generator,valid_generator,test_generator
#Remember to import excel/csv data that is preserved from the previous file,because they are used in extracting res
```

```
df1['diagnosis'] = df1['diagnosis'].astype(str)#to avoid error
df1['diagnosis'] = df1['diagnosis'].astype(str)
df2['diagnosis'] = df2['diagnosis'].astype(str)
```

```
train_generator,valid_generator,test_generator = img_generator(df1,df2)
```

```
Found 2344 validated image filenames belonging to 5 classes.
Found 585 validated image filenames belonging to 5 classes.
Found 733 validated image filenames.
```

2.Defining the Model

```
from tensorflow.keras.layers import Input, Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.models import Model

def create_model(input_shape, n_out):
    input_tensor = Input(shape=input_shape)
    base_model = 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
#Note that we've changed the pooling layer to GlobalAveragePooling2D, and set include_top=False when
```

```
model = create_model(input_shape=(HEIGHT, WIDTH, CANAL), n_out=N_CLASSES)

for layer in model.layers:
    layer.trainable = False

for i in range(-5, 0):
    model.layers[i].trainable = True
model.summary()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet
_kernels_notop.h5
94765736/94765736 [==============================] - 30s 0us/step
Model: "model"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==========================================================================================
 input_1 (InputLayer)        [(None, 320, 320, 3       0           []
                             )]

 conv1_pad (ZeroPadding2D)   (None, 326, 326, 3)       0           ['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 160, 160, 64       9472        ['conv1_pad[0][0]']
                             )

 conv1_bn (BatchNormalization)  (None, 160, 160, 64    256         ['conv1_conv[0][0]']
                             )

 conv1_relu (Activation)     (None, 160, 160, 64       0           ['conv1_bn[0][0]']
                             )
```

3.Model Compilation

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

```
293 73
```

```
from keras import optimizers
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
WARMUP_EPOCHS = 2

# train the model on the training dataset(just for warm_up)

model.compile(optimizer = optimizers.Adam(lr=WARMUP_LEARNING_RATE),loss = 'categorical_crossentropy',metrics = ['accuracy'])

history_warmup = model.fit_generator(generator=train_generator,
                                     steps_per_epoch=STEP_SIZE_TRAIN,
                                     validation_data=valid_generator,validation_steps=STEP_SIZE_VALID,
                                     epochs=WARMUP_EPOCHS,
                                     verbose=1).history
```

```
C:\Users\DELL\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:117: UserWarning: The `lr` argument is depr
ecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2488982508.py:10: UserWarning: `Model.fit_generator` is deprecated and will
be removed in a future version. Please use `Model.fit`, which supports generators.
  history_warmup = model.fit_generator(generator=train_generator,
```

```
Epoch 1/2
293/293 [==============================] - 1117s 4s/step - loss: 1.6632 - accuracy: 0.4441 - val_loss: 1.3377 - val_accurac
y: 0.4469
Epoch 2/2
293/293 [==============================] - 1114s 4s/step - loss: 1.3097 - accuracy: 0.4855 - val_loss: 1.3328 - val_accurac
y: 0.4486
```

```
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=

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'])
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 320, 320, 3       0         []
                             )]

 conv1_pad (ZeroPadding2D)   (None, 326, 326, 3)       0         ['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 160, 160, 64       9472      ['conv1_pad[0][0]']
                             )

 conv1_bn (BatchNormalizati  (None, 160, 160, 64       256       ['conv1_conv[0][0]']
 on)                         )

 conv1_relu (Activation)     (None, 160, 160, 64       0         ['conv1_bn[0][0]']
                             )

 pool1_pad (ZeroPadding2D)   (None, 162, 162, 64       0         ['conv1_relu[0][0]']
```

4.Model Training

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,''
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=EPOCHS,
                                          callbacks=callback_list,
                                          verbose=1).history
```

```
C:\Users\sai\AppData\Local\Temp\ipykernel_34532\1317685637.py:1: UserWarning: `Model.fit_generator` is deprecated and will be r
emoved in a future version. Please use `Model.fit`, which supports generators.
  history_finetunning = model.fit_generator(generator=train_generator,
```

```
Epoch 1/40
293/293 [==============================] - ETA: 0s - loss: 0.3208 - accuracy: 0.6924WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1613s 5s/step - loss: 0.3208 - accuracy: 0.6924 - lr: 1.0000e-04
Epoch 2/40
293/293 [==============================] - ETA: 0s - loss: 0.2298 - accuracy: 0.7479WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1612s 6s/step - loss: 0.2298 - accuracy: 0.7479 - lr: 1.0000e-04
Epoch 3/40
293/293 [==============================] - ETA: 0s - loss: 0.1924 - accuracy: 0.7956WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1607s 5s/step - loss: 0.1924 - accuracy: 0.7956 - lr: 1.0000e-04

Epoch 4/40
293/293 [==============================] - ETA: 0s - loss: 0.1773 - accuracy: 0.8131WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1605s 5s/step - loss: 0.1773 - accuracy: 0.8131 - lr: 1.0000e-04
Epoch 5/40
293/293 [==============================] - ETA: 0s - loss: 0.1589 - accuracy: 0.8340WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1658s 6s/step - loss: 0.1589 - accuracy: 0.8340 - lr: 1.0000e-04
Epoch 6/40
293/293 [==============================] - ETA: 0s - loss: 0.1365 - accuracy: 0.8545WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1638s 6s/step - loss: 0.1365 - accuracy: 0.8545 - lr: 1.0000e-04
Epoch 7/40
293/293 [==============================] - ETA: 0s - loss: 0.1281 - accuracy: 0.8686WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1622s 6s/step - loss: 0.1281 - accuracy: 0.8686 - lr: 1.0000e-04
Epoch 8/40
293/293 [==============================] - ETA: 0s - loss: 0.1175 - accuracy: 0.8869WARNING:tensorflow:Early stopping condition
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are: l
oss,accuracy,lr
293/293 [==============================] - 1598s 5s/step - loss: 0.1175 - accuracy: 0.8869 - lr: 1.0000e-04
Epoch 9/40
```

```
293/293 [==============================] - 1598s 5s/step - loss: 0.1175 - accuracy: 0.8869 - lr: 1.0000e-04
Epoch 9/40
293/293 [==============================] - ETA: 0s - loss: 0.0938 - accuracy: 0.9023WARNING:tensorflow:Early stopping conditio
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are:
oss,accuracy,lr
293/293 [==============================] - 1593s 5s/step - loss: 0.0938 - accuracy: 0.9023 - lr: 1.0000e-04
Epoch 10/40
293/293 [==============================] - ETA: 0s - loss: 0.0720 - accuracy: 0.9394WARNING:tensorflow:Early stopping conditio
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are:
oss,accuracy,lr
293/293 [==============================] - 1637s 6s/step - loss: 0.0720 - accuracy: 0.9394 - lr: 1.0000e-04
Epoch 11/40
293/293 [==============================] - ETA: 0s - loss: 0.0671 - accuracy: 0.9403WARNING:tensorflow:Early stopping conditio
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are:
oss,accuracy,lr
293/293 [==============================] - 1852s 6s/step - loss: 0.0671 - accuracy: 0.9403 - lr: 1.0000e-04
Epoch 12/40
293/293 [==============================] - ETA: 0s - loss: 0.0639 - accuracy: 0.9480WARNING:tensorflow:Early stopping conditio
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are:
oss,accuracy,lr
293/293 [==============================] - 1755s 6s/step - loss: 0.0639 - accuracy: 0.9480 - lr: 1.0000e-04
Epoch 13/40
293/293 [==============================] - ETA: 0s - loss: 0.0554 - accuracy: 0.9561WARNING:tensorflow:Early stopping conditio
ed on metric `val_loss` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_loss` which is not available. Available metrics are:
oss,accuracy,lr
293/293 [==============================] - 1667s 6s/step - loss: 0.0554 - accuracy: 0.9561 - lr: 1.0000e-04
```

```python
plt.figure(figsize=(8,5))

plt.plot(history_finetunning['accuracy'])
#plt.plot(history_finetunning['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.gca().ticklabel_format(axis='both', style='plain', useOffset=False)
plt.show()
```

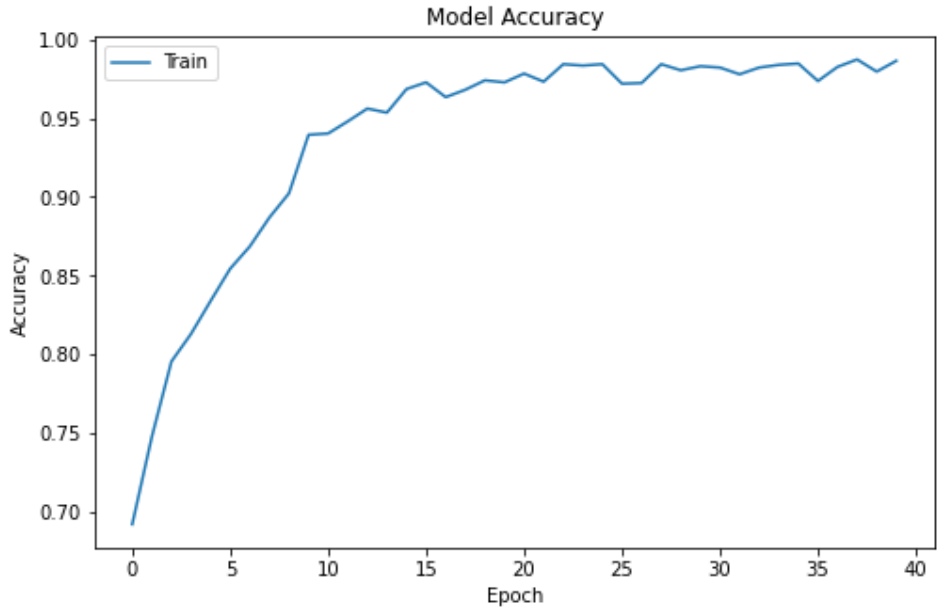


Fig 8 -Model 1 accuracy

5.Model Prediction:

```
complete_datagen = ImageDataGenerator(rescale=1./255)
complete_generator = complete_datagen.flow_from_dataframe(dataframe=df1,
                                                          directory = "./train_images_resized_preprocessed/",
                                                          x_col="file_name",
                                                          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,verbose = 1)
train_preds = [np.argmax(pred) for pred in train_preds]
```

```
Found 2929 validated image filenames.
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2271427693.py:11: UserWarning: `Model.predict_generator` is de
be removed in a future version. Please use `Model.predict`, which supports generators.
  train_preds = model.predict_generator(complete_generator, steps=STEP_SIZE_COMPLETE,verbose = 1)
```

```
2929/2929 [==============================] - 1348s 459ms/step
```

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

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2753895509.py:3: UserWarning: `Model.predict_generator` is dep
be removed in a future version. Please use `Model.predict`, which supports generators.
  test_preds = model.predict_generator(test_generator, steps=STEP_SIZE_TEST,verbose = 1)
```

```
733/733 [==============================] - 324s 442ms/step
```

6.Model Evaluation

```
print("Train Accuracy score : %.3f" % accuracy_score(df1['diagnosis'].astype('int'),train_preds))
```

```
Train Cohen Kappa score: 0.946
Train Accuracy score : 0.938
```

```
print("Test Accuracy score : %.3f" % accuracy_score(df2['diagnosis'].astype('int'),test_labels))
```

```
Test Cohen Kappa score: 0.856
Test Accuracy score : 0.783
```

5.1.2.2 MODEL BUILDING (MODEL-2,USING PRE-TRAINED RESNET LAYERS +ADDITIONAL LAYERS)

Model parameters:

BATCH_SIZE = 8

EPOCHS = 40

WARMUP_EPOCHS = 2

LEARNING_RATE = 1e-4

WARMUP_LEARNING_RATE = 1e-3

HEIGHT = 320

WIDTH = 320

CHANNEL = 3

N_CLASSES = df1['diagnosis'].nunique()

ES_PATIENCE = 5

RLROP_PATIENCE = 3

DECAY_DROP = 0.5

Steps in model creation:

1.Load data

2.Define model

3.Model compilation

4.Model training

5.Making Predictions

6.Model Evaluation

1.Load Data

Using ImageDataGenerator,we augmented the images to compensate the imbalance present in the given dataset.

```
import keras
from keras.preprocessing.image import ImageDataGenerator#The ImageDataGenerator class allows your model to rece
                                                        #new variations of the images at each epoch.(Data Augme
                                                    #(only returns transformed images,but not adds to existing set
HEIGHT = 320
WIDTH = 320
BATCH_SIZE = 8
N_CLASSES = df1['diagnosis'].nunique()
def img_generator(train,test):
    train_datagen=ImageDataGenerator(rescale=1./255, validation_split=0.2,horizontal_flip=True)
             #rescale- used to rescale pixel values from the range of 0-255 to the range 0-1 preferred for neur

    train_generator=train_datagen.flow_from_dataframe(dataframe=train,  # loads the image dataset in memory and
                                                      directory="./train_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      y_col="diagnosis",
                                                      batch_size=BATCH_SIZE,#produce BATCH_SIZE no.of images in
                                                      class_mode="categorical",#since,more than 2 classes to pr
                                                      target_size=(HEIGHT, WIDTH),
                                                      subset='training',#validate_filenames=False
                                                     ) # set as training data
    #no valid dataset,so create one
    valid_generator=train_datagen.flow_from_dataframe(dataframe=train,   #loads the image dataset in memory and
                                                      directory="./train_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      y_col="diagnosis",
                                                      batch_size=BATCH_SIZE,#produce BATCH_SIZE no.of images in
                                                      class_mode="categorical",#since,more than 2 classes to pr
                                                      target_size=(HEIGHT, WIDTH),
                                                      subset='validation',
                                                     )# set as validation data
```

```
    test_datagen = ImageDataGenerator(rescale=1./255)
    test_generator = test_datagen.flow_from_dataframe(dataframe=test,  # loads the image dataset in memory and gene
                                                      directory = "./valid_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      target_size=(HEIGHT, WIDTH),
                                                      batch_size=1,
                                                      shuffle=False,
                                                      class_mode=None,
                                                     )

    return train_generator,valid_generator,test_generator
#Remember to import excel/csv data that is preserved from the previous file,because they are used in extracting res
```

```
df1['diagnosis'] = df1['diagnosis'].astype(str)#to avoid error
df1['diagnosis'] = df1['diagnosis'].astype(str)
df2['diagnosis'] = df2['diagnosis'].astype(str)
```

```
train_generator,valid_generator,test_generator = img_generator(df1,df2)
```

```
Found 2344 validated image filenames belonging to 5 classes.
Found 585 validated image filenames belonging to 5 classes.
Found 733 validated image filenames.
```

2.Defining the Model

```
from tensorflow.keras.layers import Input, Dense, Dropout, GlobalAveragePooling2D,Conv2D,MaxPooling2D
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.models import Model
def create_model(input_shape, n_out):
    input_tensor = Input(shape=input_shape)#Input() is used to represent a Keras tensor
    base_model = ResNet50(weights='imagenet', include_top=False, input_tensor=input_tensor)#transfer learning-
    #ImageNet is an image database organized according to the WordNet hierarchy,designed for use in visual obj
    #More than 14 million images have been hand-annotated by the project to indicate what objects are pictured


    #Setting include_top to False means it will allow adding input and output layers custom to a problem
    #ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one

    #base_model = applications.ResNet50(weights=None, include_top=False,input_tensor=input_tensor)
    #base_model.load_weights('resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5')
    #https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_
    #h5-An H5 is one of the Hierarchical Data Formats (HDF) used to store large amount of data  in the form of
    #The format is primarily used to store scientific data that is well-organized for quick retrieval and anal

    x=Conv2D(64,kernel_size=3,activation='relu')(base_model.output)#64-number of filters of size 3*3
    x=MaxPooling2D(pool_size=(2,2))(x)

    #add a fully connected output layer to the model where the learning can happen:
    x = GlobalAveragePooling2D()(base_model.output)

    x = Dropout(0.5)(x)#Dropout layers have been the go-to method to reduce the overfitting of neural networks

    x = Dense(2048, activation='relu')(x)#fully connected layer

    final_output = Dense(n_out, activation='softmax', name='final_output')(x)
    model = Model(input_tensor, final_output)
    return model
```

```
from keras import applications
HEIGHT = 320
WIDTH = 320
CHANNEL = 3 #3 channel,since RGB
model = create_model(input_shape=(HEIGHT, WIDTH, CHANNEL), n_out=N_CLASSES)

#The for loop on the model's layers ensures it doesn't learn its weights again a
for layer in model.layers:
    layer.trainable = False

#Only some weights in the model are freezed or locked
for i in range(-5, 0):
    model.layers[i].trainable = True
model.summary()
```

Output:

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 320, 320, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 326, 326, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 160, 160, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 160, 160, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 160, 160, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 162, 162, 64  0           ['conv1_relu[0][0]']
```

3.Model Compilation

```
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)
```

```
293 73
```

```
from keras import optimizers
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
WARMUP_EPOCHS = 2

# train the model on the training dataset(just for warm_up)

model.compile(optimizer = optimizers.Adam(lr=WARMUP_LEARNING_RATE),loss = 'categorical_crossentropy',metrics = ['accuracy'])

history_warmup = model.fit_generator(generator=train_generator,
                                     steps_per_epoch=STEP_SIZE_TRAIN,
                                     validation_data=valid_generator,validation_steps=STEP_SIZE_VALID,
                                     epochs=WARMUP_EPOCHS,
                                     verbose=1).history
```

```
C:\Users\DELL\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2488982508.py:10: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history_warmup = model.fit_generator(generator=train_generator,
```

```
Epoch 1/2
293/293 [==============================] - 1117s 4s/step - loss: 1.6632 - accuracy: 0.4441 - val_loss: 1.3377 - val_accuracy: 0.4469
Epoch 2/2
293/293 [==============================] - 1114s 4s/step - loss: 1.3097 - accuracy: 0.4855 - val_loss: 1.3328 - val_accuracy: 0.4486
```

```
from keras.callbacks import EarlyStopping,ReduceLROnPlateau

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 = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
model.summary()
```

Output:

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 320, 320, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 326, 326, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 160, 160, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 160, 160, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 160, 160, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 162, 162, 64  0           ['conv1_relu[0][0]']
```

4.Model Training

```
EPOCHS=15
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=EPOCHS,
                                          callbacks=callback_list,
                                          verbose=1).history
```

Output:

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2112883953.py:2: UserWarning: `Model.fit_generator` is deprecated and will
be removed in a future version. Please use `Model.fit`, which supports generators.
  history_finetunning = model.fit_generator(generator=train_generator,
Epoch 1/15
293/293 [==============================] - 3611s 12s/step - loss: 1.0211 - accuracy: 0.7009 - val_loss: 2.2248 - val_accura
cy: 0.4469 - lr: 1.0000e-04
Epoch 2/15
293/293 [==============================] - 2952s 10s/step - loss: 0.6485 - accuracy: 0.7624 - val_loss: 0.9869 - val_accura
cy: 0.6387 - lr: 1.0000e-04
Epoch 3/15
293/293 [==============================] - 3224s 11s/step - loss: 0.5384 - accuracy: 0.7969 - val_loss: 0.7643 - val_accura
cy: 0.7551 - lr: 1.0000e-04
Epoch 4/15
293/293 [==============================] - 3704s 13s/step - loss: 0.4707 - accuracy: 0.8251 - val_loss: 0.6536 - val_accura
cy: 0.7620 - lr: 1.0000e-04
Epoch 5/15
293/293 [==============================] - 3390s 12s/step - loss: 0.4224 - accuracy: 0.8336 - val_loss: 0.7400 - val_accura
cy: 0.7312 - lr: 1.0000e-04
Epoch 6/15
293/293 [==============================] - 2944s 10s/step - loss: 0.3668 - accuracy: 0.8601 - val_loss: 0.5959 - val_accura
cy: 0.8151 - lr: 1.0000e-04
Epoch 7/15
293/293 [==============================] - 3300s 11s/step - loss: 0.3219 - accuracy: 0.8771 - val_loss: 0.7666 - val_accura
cy: 0.7979 - lr: 1.0000e-04
Epoch 8/15
293/293 [==============================] - 3706s 13s/step - loss: 0.2896 - accuracy: 0.8989 - val_loss: 0.9253 - val_accura
cy: 0.6866 - lr: 1.0000e-04
Epoch 9/15
293/293 [==============================] - ETA: 0s - loss: 0.2371 - accuracy: 0.9147
Epoch 9: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
293/293 [==============================] - 3202s 11s/step - loss: 0.2371 - accuracy: 0.9147 - val_loss: 0.8669 - val_accura
cy: 0.7414 - lr: 1.0000e-04
Epoch 10/15
293/293 [==============================] - 3082s 11s/step - loss: 0.1441 - accuracy: 0.9539 - val_loss: 0.7832 - val_accura
cy: 0.7979 - lr: 5.0000e-05
Epoch 11/15
293/293 [==============================] - ETA: 0s - loss: 0.0923 - accuracy: 0.9706 Restoring model weights from the end o
f the best epoch: 6.
293/293 [==============================] - 3704s 13s/step - loss: 0.0923 - accuracy: 0.9706 - val_loss: 0.9195 - val_accura
cy: 0.8134 - lr: 5.0000e-05
Epoch 11: early stopping
```

```
import matplotlib.pyplot as plt
# evaluate the ResNet-50 model after training the model
plt.figure(figsize=(8,5))

plt.plot(history_finetunning['accuracy'])
plt.plot(history_finetunning['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.gca().ticklabel_format(axis='both', style='plain', useOffset=False)
plt.show()
```

Output:

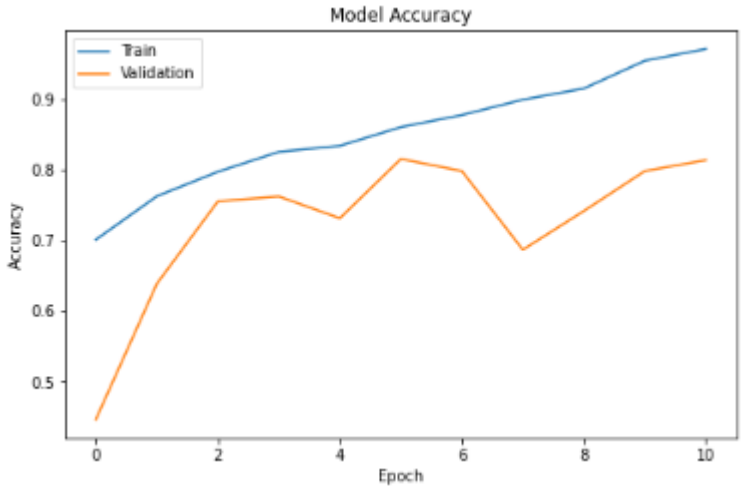


Fig 9-Model 2 accuracy

5.Model Prediction:

```
complete_datagen = ImageDataGenerator(rescale=1./255)
complete_generator = complete_datagen.flow_from_dataframe(dataframe=df1,
                                                          directory = "./train_images_resized_preprocessed/",
                                                          x_col="file_name",
                                                          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,verbose = 1)
train_preds = [np.argmax(pred) for pred in train_preds]
```

```
Found 2929 validated image filenames.
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2271427693.py:11: UserWarning: `Model.predict_generator` is de
be removed in a future version. Please use `Model.predict`, which supports generators.
  train_preds = model.predict_generator(complete_generator, steps=STEP_SIZE_COMPLETE,verbose = 1)
```

```
2929/2929 [==============================] - 1348s 459ms/step
```

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

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_3784\2753895509.py:3: UserWarning: `Model.predict_generator` is dep
be removed in a future version. Please use `Model.predict`, which supports generators.
  test_preds = model.predict_generator(test_generator, steps=STEP_SIZE_TEST,verbose = 1)
```

```
733/733 [==============================] - 324s 442ms/step
```

6.Model Evaluation

```
#print("Train Cohen Kappa score: %.3f" % cohen_kappa_score(train_preds, df_train['diagnosis'].astype('int'
print("Train Accuracy score : %.3f" % accuracy_score(df1['diagnosis'].astype('int'),train_preds))
```

```
Train Accuracy score : 0.896
```

```
#print("Test Cohen Kappa score: %.3f" % cohen_kappa_score(test_labels, df2['diagnosis'].astype('int'), wei
print("Test Accuracy score : %.3f" % accuracy_score(df2['diagnosis'].astype('int'),test_labels))
```

```
Test Accuracy score : 0.797
```

7.Model Saving

```
from keras.models import load_model

# Assuming that your model is named `model`
model.save('my_model.h5')
```

```
model_loaded= load_model('my_model.h5') # loads the model from the saved file
```

```
model_loaded # just checking the model loaded
```

```
<keras.engine.functional.Functional at 0x2042bf67370>
```

```
model_loaded.summary() # just checking the loaded model weights
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 320, 320, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 326, 326, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 160, 160, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 160, 160, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 160, 160, 64  0           ['conv1_bn[0][0]']
                                )
```

5.1.3 USER INTERFACE

```
import streamlit as st
import tensorflow as tf
from PIL import Image
import gradio as gr
import cv2
import pandas as pd
import numpy as np
#streamlit run app.py
# Load the model
model = tf.keras.models.load_model('my_model.h5')

# Define the class labels
classes = ['No DR', 'Mild', 'Moderate', 'Severe', 'Proliferative DR']
# st.write("hello")

st.set_page_config(page_title='Diabetic Retinopathy Detection', page_icon=':eyes:')
st.sidebar.title('Diabetic Retinopathy Detection')
st.sidebar.info('Upload an image and the app will predict the damage level of diabetic retinopathy.')
```

```python
    # Define the Streamlit app
def app():
    # Set the title and sidebar

    # Upload the image
    uploaded_file = st.file_uploader('Choose an image', type=['jpg', 'jpeg', 'png'])
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded Image', use_column_width=True)

        # Preprocess the image
        preprocessed_image = preprocess_image(np.array(image))

        st.image(preprocessed_image, caption='preprocessing above Image', use_column_width=True)
        img_array = tf.keras.preprocessing.image.img_to_array(preprocessed_image)
        img_array = tf.image.resize(img_array, [320, 320])
        img_array = tf.expand_dims(img_array, 0)
        img_array = img_array / 255.0

        # Make a prediction
        prediction = model.predict(img_array)
        predicted_class = classes[prediction.argmax()]
        st.success(f'The damage level of diabetic retinopathy is {predicted_class}.')

app()
```

Output:

We pass a retina image to this Interface and it returns preprocessed image and predicts its label as output.

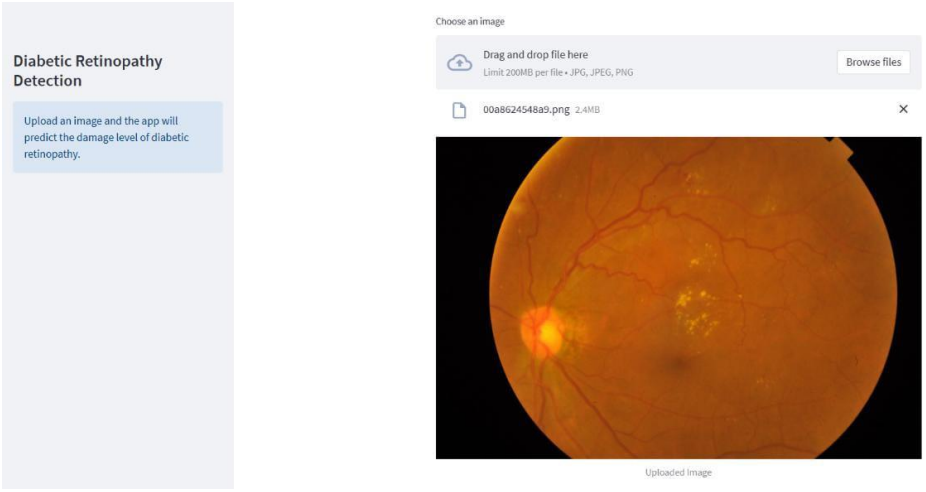


Fig 10 User Interface preview-I

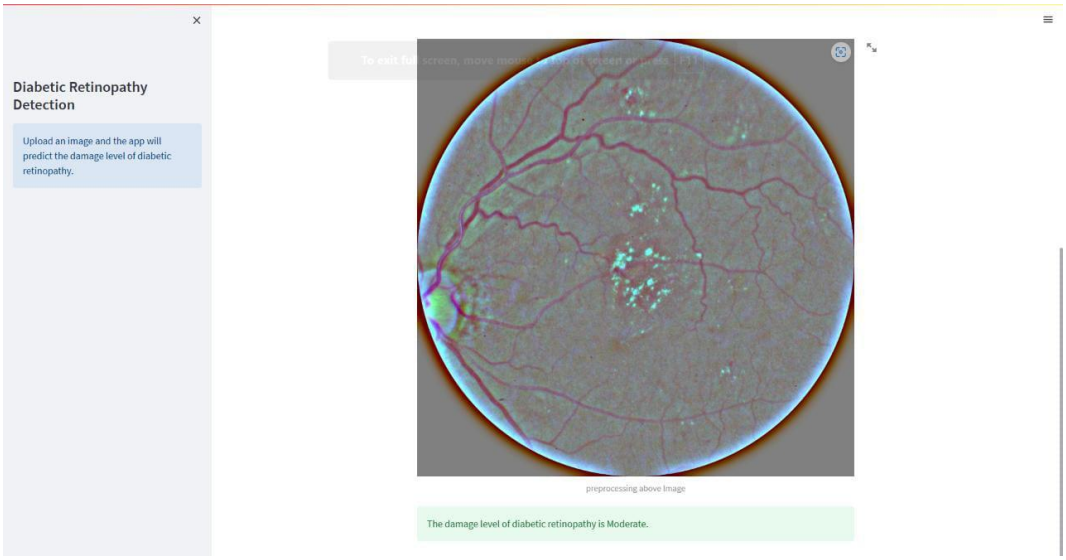


Fig 11- User Interface Preview-II

5.2 TESTING

5.2.1 INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

5.2.2 TESTING CONCEPTS:

- A component is a part of the system that can be isolated for testing. A component can be an object, a group of objects, or one or more subsystems.
- A fault, also called bug or defect, is a design or coding mistake that may cause abnormal component behaviour.
- An error is a manifestation of a fault during the execution of the system.
- A failure is a deviation between the specification of a component and its behaviour. A failure is triggered by one or more errors.
- A test case is a set of inputs and expected results that exercise a component with the purpose of causing failures and detecting faults.

5.2.3 TESTING ACTIVITIES:

- Inspecting a component, which finds faults in an individual component through the manual inspection of its source code.
- Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the components using a test case.
- Integration testing, which finds faults by integrating several components together.
- System testing, which focuses on the complete system, its functional and Non-functional requirements and its target environment.

Testing is the phase where the errors remaining from all the previous phases must be detected. Hence, testing performs a very critical role for quality assurance and for ensuring the liability of software. Testing of designed software consists of providing the software with a set of test outputs and observing if the software behaves as expected if the software fails to behaves as expected, then the conditions under which a failure occurs are needed for debugging and correction.

The following terms are some commonly used terms associated with testing.

ERROR

The term error is used in two different ways. It refers to the discrepancy between a computed, observed, or measured value and true, specified, or theoretically correct value. Error is also used to refer to human action that results in software containing a defect or fault. This definition is quite general and encompasses all the phases.

FAULT

Fault is a condition that causes a system to fail in performing its required function. In other word a fault is an incorrect intermediate state that may have been entered during program execution.

FAILURE

Failure is the inability of the system or component to perform a required function according to its specifications. In other word a failure is a manifestation of error. But the mere presence of an error may not cause a failure.

5.2.4 TESTING METHODS

1. White box testing.
2. Black box testing.

5.2.4.1 White box testing:

1. In the white box testing the designer can derive test cases that guarantee that all independent paths within a module have been exercised at least once.
2. Exercise all logical decision on their true and false sides.
3. Exercise all loops at their boundaries and within their operational bounds.
4. Exercise internal data structures to ensure their validity.

There are several methods used in the white box testing in the software design:

Statement coverage

In this, each statement in a program is executed at least once. This is done by checking the program in debug mode and by verifying each statement.

Branch coverage

In this, each and every branch is tested whether the different branch conditions are true or false. It is a stronger test criterion over statement coverage testing.

Condition coverage

In this, each component of a condition of a composite conditional expression is given true and false values. It is a stronger test criterion over branch coverage testing.

Path coverage

In this testing, all the linearly independent paths in the program are executed once. A linearly independent path is defined in terms of the control flow graph of a program. The control flow graph describes how the flow of control passes through the program.

Data-flow based testing

In this method, selects the test paths of a program according to the locations of the definitions and use of different variables in program. Data-flow based testing strategies are useful for selecting tested paths of a program containing nested if and loop statements.

Mutation testing

In this method, the software is first tested by the above methods after the initial testing. In the mutation testing few small changes to program at a time such as, changing a conditional operator or changing the type of variable each time the program is changed, is called a mutated program and the change effected is called mutant. Then it is tested if it gives dead, if it is working properly then the test data is an error, then the mutant is said to enhanced to kill the mutant.

5.2.4.2 Black box testing

The black box testing focuses on the functional requirement of the software. That is for a swinger the black box testing will give the desired results for what it is meant for, it means that he has to exercise all functional requirements for a program. The black box testing is done after the software is developed.

Black box testing attempts to find errors in the following:

1. Incorrect or missing functions.
2. Interface errors.
3. Errors in data structures or external database access.
4. Performance errors.

5. Initialization and termination errors.

Testing object oriented programs are different from testing functional oriented programs. As the test cases in black box testing are selected independently of the internal structure of the code, it does not matter whether the software is object oriented or function oriented however, if the testing is white box, then the test cases are determined based on the structure of the program. Testing classes is fundamentally different from testing functions as a function has a clearly defined input-output behaviour while a class does not have an input-output behavior specification. Further a class cannot be tested directly; only an instance of class can be tested.

This means that, we test a class indirectly by testing its instance. Further there is no sequential control flow with in a class like in functions. In a function, arguments, passed to the function with global data determine the path of execution within the procedure, but in an object, the state associated with object also influences the path of execution, and methods of a class can communicate each other among themselves through this state, because this state is persistent across invocation of methods. Hence, state of an object plays a major role in testing objects.

System testing

System tests are designed to validate a fully developed system with a view to assuring that it meets requirements. There are three kinds of system testing.

Alpha testing

Alpha testing refers to the system testing that it's carried out by the customer within the organization along with the developer. The alpha tests are conducted in a controlled manner.

Beta testing

Beta testing is the system testing performed by a select group customer’s. The developer is not present at the site and the user will inform the problems that are encountered. As a result of problems reported during the Beta test, the software

developer makes the modifications and then prepares for release of the software to the customer.

Acceptance Testing

Acceptance Testing is the system testing performed by the customer to whether or not of accept the delivery of the system.

5.2.5 UNIT TESTING

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality. Test cases and results are shown in the Tables.

Unit Testing Benefits

- Unit testing increases confidences in changing/ managing code.
- Codes are more reusable.
- Development is faster.
- The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.
- Codes are more reliable.
- Debugging is easy.

Unit testing:

S1 # Test Case: -	UTC-1
Name of Test: -	Uploading image
Items being tested: -	Tested for uploading different images
Sample Input: -	Upload sample images
Expected Output: -	Image should upload properly
Actual Output: -	Upload successful
Remarks: -	Pass

TABLE 1: Unit Testing

5.2.6 INTEGRATION TESTING

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing. Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. It occurs after unit testing and before validation testing. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

5.2.6.1 BOTTOM-UP INTEGRATION

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

5.2.6.2 TOP-DOWN INTEGRATION

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter. In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

Integration testing:

S1 # Test Case: -	ITC-1
Name of Test: -	Working of chose file option
Items being tested: -	User convenience to access images stored
Sample Input: -	Click and select image
Expected Output: -	Should open selected image
Actual Output: -	Selected image should load

Remarks: -	Pass

TABLE 2: Integration Testing

5.2.7 VALIDATION TESTING

An engineering validation test (EVT) is performed on first engineering prototypes, to ensure that the basic unit performs to design goals and specifications. It is important in identifying design problems, and solving them as early in the design cycle as possible, is the key to keeping projects on time and within budget. Too often, product design and performance problems are not detected until late in the product development cycle when the product is ready to be shipped. The old adage holds true: It costs a penny to make a change in engineering, a dime in production and a dollar after a product is in the field.

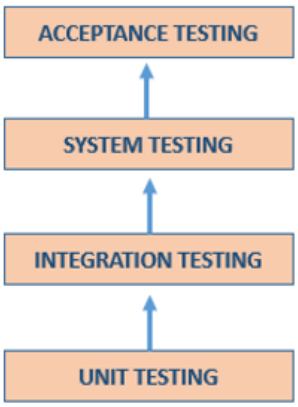


Fig 7- Test Process

CHAPTER 6

RESULTS AND DISCUSSION

6.1 RESULTS

In a Diabetic Retinopathy project, the test accuracy score reflects how well the model can classify retinal images as normal or diseased. It is calculated by comparing the model's predicted labels with the ground truth labels of the testing set, and computing the percentage of correctly classified images.

A high test accuracy score indicates that the model is able to accurately classify retinal images and has learned the relevant features and patterns in the data. However, it is important to note that a high accuracy score alone does not necessarily mean that the model is perfect or optimal. Other metrics such as precision, recall, and F1 score should also be considered to evaluate the model's performance on different aspects such as false positives, false negatives, and overall balance.

Model-1 result:

```
print("Train Accuracy score : %.3f" % accuracy_score(df1['diagnosis'].astype('int'),train_preds))
```

```
Train Cohen Kappa score: 0.946
Train Accuracy score : 0.938
```

```
print("Test Accuracy score : %.3f" % accuracy_score(df2['diagnosis'].astype('int'),test_labels))
```

```
Test Cohen Kappa score: 0.856
Test Accuracy score : 0.783
```

Model-2 result:

```
print("Train Accuracy score : %.3f" % accuracy_score(df1['diagnosis'].astype('int'),train_preds))
```

```
Train Accuracy score : 0.896
```

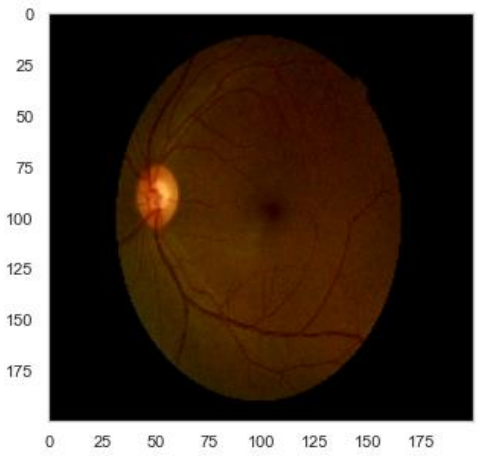
```
print("Test Accuracy score : %.3f" % accuracy_score(df2['diagnosis'].astype('int'),test_labels))
```

```
Test Accuracy score : 0.797
```

6.2 COMPARISION

BEFORE IMAGE PROCESSING:

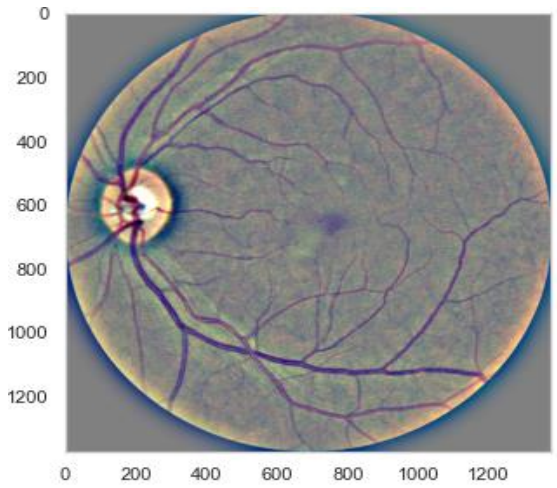
Before image processing, the raw images may contain various types of noise, artifacts, and distortions that can affect the quality and clarity of the images. For example, there may be variations in brightness, contrast, and color that make it difficult to distinguish between normal and diseased retinal images. There may also be blurring or sharpness issues that affect the image's details and edges. In addition, the images may contain unwanted objects or regions that need to be removed or isolated.



AFTER IMAGE PROCESSING:

Preprocessing the images before applying any analysis or modeling techniques is essential to ensure that the images are of good quality and contain the necessary information to make accurate predictions. This can involve various techniques such as filtering, noise reduction, normalization, and feature extraction. For example, Gaussian blur can be applied to smooth out minor variations and improve the

image's overall quality. Additionally, normalization techniques can be used to adjust the brightness and contrast levels of the images to make them more consistent and comparable. By performing preprocessing on the images, we can improve their quality and make them more suitable for further analysis and modeling.



6.3 DISCUSSION

We researched for different approaches to the problem of DR.Few were:

1.Using classical methods of computer vision and machine learning to provide a suitable solution to this problem, a computer-vision-based approach for the detection of diabetic retinopathy stages using color fundus images,and extract features from the raw image using image processing techniques, and fed them to the SVM for binary classification and achieved an accuracy of 97.6% on a testing set of 250 images.But if gave 65% accuracy on large dataset.

2.Other method is to fit other models for multiclass classification, e.g., applying PCA to images and fitting decision trees, naive Bayes, or k-NN with best results 73.4% of accuracy, while using a dataset of 151 images with different resolutions.

3.Another method is to develop a network with CNN architecture and data augmentation, which can identify the intricate features involved in the classification task such as micro-aneurysms, exudate, and hemorrhages in the retina and

consequently provide a diagnosis automatically and without user input. They achieved an accuracy of 75% on 5,000 validation images.

4.But,we tried to use transfer learning(ResNet50,which is pretrained on ImageNet Database), with CNN and we achieved train accuracy of 89.6% and test accuracy of 79.7%. with Resnet-50 and few customized layers.Because using only Resnet 50 gave train accuracy of about 93.6% ,but test accuracy of 78.3%.

MODEL	**TEST ACCURACY ACHIEVED**
SVM for binary classification	65%
Multiclass Classification	73.4%
CNN+Data Augmentation	75%
Transfer Learning	79.7%

TABLE 3- Comparision Table

CHAPTER 7

CONCLUSION AND FUTURE EHNACEMENT

We proposed the multistage transfer learning approach and an automatic method for detection of the stage of diabetic retinopathy by single photography of the human fundus. We have used an ensemble of 3 CNN architectures (EfficientNet-B4, EfficientNet-B5, SE- ResNeXt50) and made transfer learning for our final solution. The experimental results show that the proposed method achieves high and stable results even with unstable metric. The main advantage of this method is that it increases generalization and reduces variance by using an ensemble of the networks, pretrained on a large dataset, and finetuned on the target dataset.

The future work can extend this method with the calculation of SHAP for the whole ensemble, not only for a particular network, and with more accurate hyperparameter optimization. Besides, we can do experiments using pretrained encoders on other connected to eye ailments tasks. Also, it is possible to investigate meta-learning (Nichol et al., 2018) with these models, but realized that it requires the separate in-depth research.

CHAPTER 8
REFERENCES

1. APTOS (2019). APTOS 2019 blindness detection. Accessed: 2019-10-20.

2. Asiri, N., Hussain, M., and Aboalsamh, H. A. (2018). Deep learning based computer-aided diagnosis systems for diabetic retinopathy: A survey. CoRR, abs/1811.01238.

3. Carson Lam, Darvin Yi, M. G. and Lindsey, T. (2018). Automated detection of diabetic retinopathy using deep learning.

4. Cheng, J. (2007). A neural network approach to ordinal regression. CoRR, abs/0704.1028.

5. Christopher E.Hann, J. Geoffrey Chase, J. A. R. D. H. G. M. S. (2009). Diabetic retinopathy screening using computer vision.

6. Conde, P., de la Calleja, J., Medina, M., and Benitez Ruiz, A. B. (2012). Application of machine learning to classify diabetic retinopathy.

7. Decencire, E., Zhang, X., Cazuguel, G., Lay, B., Cochener, B., Trone, C., Gain, P., Ordonez, R., Massin, P., Erginay, A., Charton, B., and Klein, J.-C. (2014). Feedback on a publicly distributed database: the messidor database. Image Analysis & Stereology, 33(3):231–234.

8. Devries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. CoRR, abs/1708.04552.
9. A. Buslaev, A. Parinov, E. K. V. I. I. and Kalinin, A. A. (2018). Albumentations: fast and flexible image augmentations. ArXiv e-prints.

10. Ananth, C. V. and Kleinbaum, D. G. (1997). Regression models for ordinal responses: a review of methods and applications. International Journal of Epidemiology, 26(6):1323–1333.