A Project Report

on

DIABETIC RETINOPATHY DETECTION USING ARTIFICIAL INTELLIGENCE

carried out as part of the ***Artificial Intelligence Lab DS3230*** *Submitted*

by

Perumalla Thushara Meher Siva Mani
209302296

Vaibhav Nagar
209309072

Avval Kaur
209309001

in partial fulfillment for the award of the degree of

Bachelor of Technology

School of Computing and Information Technology
Department of Data Science and Engineering

MANIPAL UNIVERSITY JAIPUR
JAIPUR-303007
RAJASTHAN, INDIA

April 2023

CERTIFICATE

Date: 20/05/2022

This is to certify that the minor project titled **DIABETIC RETINOPATHY DETECTION USING ARTIFICIAL INTELLIGENCE** is a record of the bonafide work done by **Perumalla Thushara Meher Siva Mani**(Reg. No:209302296), **Vaibhav Nagar**(Reg. No:209309072) and **Avval Kaur**(Reg. No:209309001) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Data Science of Manipal University Jaipur, during the academic year 2022-23.

Dr. Avani Sharma

Project Guide, Department of Information Technology
Assistant Professor, Department of Information Technology
Manipal University Jaipur

Dr. Akhilesh Kumar Sharma

HoD, Department of Data Science and Engineering
Manipal University Jaipur

ABSTRACT

Diabetic retinopathy is a global medical concern, especially among elderly individuals, where uncontrolled glucose levels in the body can damage the retina, leading to permanent vision loss. To diagnose or monitor the progression of this condition, the fundus oculi technique is commonly used, which involves observing the eyeball.

In this study, we developed a Convolutional Neural Network (CNN) model to process fundus oculi images and accurately detect the presence of diabetic retinopathy. The dataset used for training and testing the model consisted of medical fundus oculi images, with labels indicating the severity of the pathology observed in the eyeball. The severity scale categorized the images into five classes, ranging from a healthy eyeball to the presence of proliferative diabetic retinopathy, which is indicative of blindness in the patient.

Our proposed CNN model achieved an impressive accuracy of 84%, demonstrating its capability to confidently predict the presence of diabetic retinopathy in fundus oculi images. This work contributes to the advancement of medical imaging and has the potential to assist healthcare professionals in accurately identifying and managing diabetic retinopathy, thereby improving patient care in this global health concern.

LIST OF FIGURES

TABLE OF CONTENTS

1. Introduction

1.1. Introduction

Diabetic retinopathy(DR), a complication of diabetes, is a leading cause of blindness worldwide. Early detection and timely intervention can greatly reduce the risk of vision loss. With the advent of Artificial Intelligence (AI) and Convolutional Neural Networks (CNNs), there has been significant progress in developing automated systems for diabetic retinopathy detection.

In this project, we aim to leverage the power of AI and CNNs to detect diabetic retinopathy from fundus oculi images. Fundus oculi images are images of the interior surface of the eye, specifically the retina, captured using a specialized camera. These images provide valuable information about the health of the retina and can be used for early diagnosis and monitoring of diabetic retinopathy.

Our project involves training a CNN model on a dataset of fundus oculi images with labeled severity grades of diabetic retinopathy. The CNN model will learn to automatically extract relevant features from the images and classify them into different severity levels of diabetic retinopathy(No_DR, Mild, Moderate, Severe and Proliferate_DR). The goal is to develop a highly accurate and automated system that can assist healthcare professionals in early detection and management of diabetic retinopathy, thus preventing vision loss in patients with diabetes.

The potential impact of our project is significant, as it has the potential to improve the efficiency and accuracy of diabetic retinopathy screening, particularly in resource-constrained settings. The successful development of an AI-based system for diabetic retinopathy detection could lead to early intervention, improved patient outcomes, and a reduction in the global burden of blindness due to diabetic retinopathy.

1.2. Problem Statement

Diabetic retinopathy is a significant global health concern that can result in permanent vision loss or blindness if not detected and managed early. Traditional methods of diabetic retinopathy screening involve manual examination of fundus oculi images by trained healthcare professionals, which can be time-consuming, labor-intensive, and subject to human error. Additionally, access to expert ophthalmologists and specialized equipment may be limited in certain remote regions, leading to delays in diagnosis and treatment.

The lack of automated and efficient diabetic retinopathy detection systems poses a challenge in timely identifying and managing this condition, especially in large populations with a high prevalence of diabetes. Therefore, the problem at hand is to develop an accurate and automated system using artificial intelligence techniques, specifically, Convolutional Neural Networks (CNNs), to detect diabetic retinopathy from fundus oculi images.

The proposed solution aims to address the limitations of traditional screening methods by leveraging the power of AI and CNNs to automatically analyze fundus oculi images and accurately classify them into different severity levels of diabetic retinopathy. The system should be capable of providing fast, reliable, and scalable diabetic retinopathy detection, aiding healthcare professionals in early diagnosis, monitoring, and management of this condition. The solution should also consider potential challenges such as data variability, class imbalance, and model interpretability, and strive for robustness and generalizability across different populations and settings.

The successful development of an AI-based diabetic retinopathy detection system would have a significant positive impact on public health by enabling early intervention, reducing the risk of vision loss, and improving patient outcomes.

1.3. Objectives

- Apply appropriate python libraries for import, pre-process and visualize images
- Perform Data Augmentation to improve model generalization capability
- Build Deep Learning model based on CNN and Residual Blocks using Keras with TensorFlow as Backend
- Classify the retina images into different severity levels of diabetic retinopathy(No_DR, Mild, Moderate, Severe and Proliferate_DR)

1.4. Scope of project

The project would involve collecting a large and diverse dataset of fundus oculi images from diabetic patients, along with corresponding labels indicating the severity levels of diabetic retinopathy. The collected data would need to be preprocessed, including image resizing, normalization, and augmentation, to prepare it for training and testing of the CNN model.

We would be developing a Convolutional Neural Network (CNN) model using deep learning techniques for diabetic retinopathy detection. The model would be trained on the preprocessed dataset to learn patterns and features from the images associated with different severity levels of diabetic retinopathy. Hyperparameter tuning and model optimization techniques may also be applied to enhance the model's performance.

The developed CNN model would be evaluated and validated using appropriate performance metrics such as accuracy, precision, recall, F1-score, etc. Cross-validation techniques may be employed to ensure the model's robustness and generalizability. Model interpretability and explainability techniques may also be applied to gain insights into the decision-making process of the CNN model.

Once the CNN model is trained and validated, it would be deployed as a functional system that can take fundus oculi images as input and provide diabetic retinopathy predictions as output. The system may be integrated into existing healthcare systems or standalone applications to enable easy access and utilization by healthcare professionals.

The deployed system would undergo thorough testing and validation to ensure its accuracy, reliability, and safety. Real-world fundus oculi images from diabetic patients would be used to validate the system's performance and assess its effectiveness in detecting diabetic retinopathy.

The project would consider ethical implications such as patient privacy, data security, and fairness in model predictions. Appropriate measures would be taken to protect sensitive patient information and ensure compliance with relevant regulations and ethical guidelines.

2. Background Detail

2.1. Literature Review

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

Deep learning is a subset of machine learning and artificial intelligence, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).

Convolutional Neural Networks (CNNs or ConvNets) are a class of deep neural networks specifically designed for image processing tasks, such as image classification, object detection, and image generation. CNNs have been widely used and have achieved state-of-the-art performance in various computer vision tasks.

The architecture of a typical CNN consists of several key components:

- **Convolutional Layers:** These layers apply convolution operations to input images, which involves applying filters or kernels to the input image to extract local patterns or features, such as edges, textures, and shapes. These convolutional layers are typically followed by activation functions, such as ReLU (Rectified Linear Unit), to introduce non-linearity into the model.
- **Pooling Layers:** These layers downsample the output of convolutional layers, reducing the spatial dimensions of the feature maps while retaining important features. Common pooling operations include max pooling, average pooling, and global pooling.
- **Fully Connected Layers:** These layers are traditional artificial neural network layers, where each neuron is connected to all neurons in the previous layer. Fully connected layers are typically used at the end of the CNN to combine the learned features into a final prediction. Activation functions are also applied to fully connected layers.
- **Dropout:** Dropout is a regularization technique used in CNNs to prevent overfitting. It randomly sets a fraction of the input units to 0 at each training iteration, which helps to prevent the network from relying too heavily on any single input neuron.
- **Loss Function:** The loss function quantifies the difference between the predicted output and the actual output. It is used during training to guide the network to minimize the error between predictions and actual values.
- **Optimizer:** The optimizer is used during training to update the weights of the network and minimize the loss function. Popular optimizers used in CNNs include Adam, SGD (Stochastic Gradient Descent), and RMSprop.

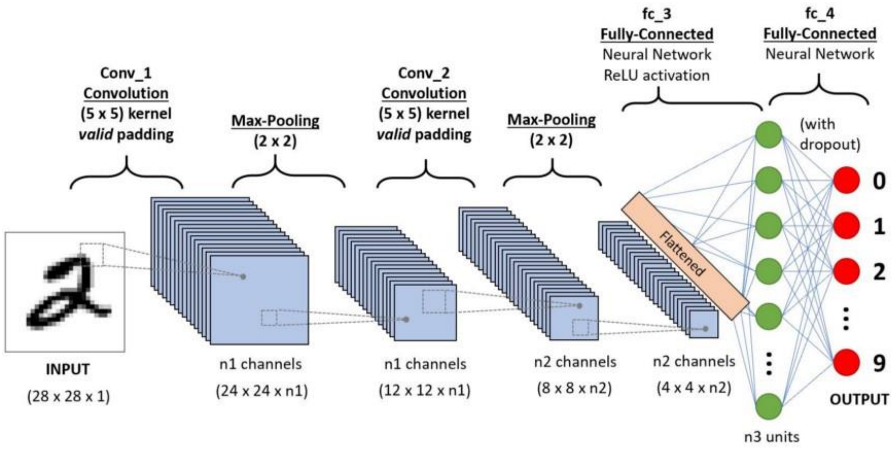


Fig 2.1 Architecture of Convolutional Neural Network

Residual neural network (ResNet) is a type of convolutional neural network (CNN) that was introduced in a 2015 paper by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.The main innovation of ResNet is the use of residual connections, which allow the network to learn residual mappings instead of trying to learn the desired output directly. This is achieved by adding skip connections that bypass one or more layers, allowing the input to be added directly to the output of a later layer. This helps to address the problem of vanishing gradients, where the gradients of the loss function become very small as they propagate backwards through many layers of the network during training.

ResNets have achieved state-of-the-art performance in many computer vision tasks, including image classification, object detection, and semantic segmentation. They have been widely used in various applications, including medical image analysis, self-driving cars, and facial recognition.

The **vanishing gradient problem** is a challenge that can arise in training deep neural networks, especially those with many layers. When a neural network is trained using backpropagation, the gradients of the loss function with respect to the weights of the network are computed and used to update the weights during training.

However, as the gradients propagate backwards through the network, they can become very small or even vanish completely. This can occur because the gradients are multiplied by the weights of each layer during backpropagation, and if the weights are small, the gradients can become exponentially smaller as they propagate backwards through the layers.

When the gradients become very small, it becomes difficult to update the weights of the network effectively, and the network may stop learning or learn very slowly. This can make it difficult to train deep neural networks with many layers, especially when using certain activation functions, such as the sigmoid function, which can saturate and produce small gradients.

The vanishing gradient problem can be mitigated through various techniques, such as using different activation functions (such as the ReLU function), using weight initialization methods that can help prevent the gradients from becoming too small, or using regularization techniques that can help prevent overfitting and improve the generalization of the network. Another solution is to use skip connections, such as in residual neural networks, which allow the gradients to bypass one or more layers and propagate more directly to earlier layers, reducing the likelihood of the gradients becoming very small.

ResNet was introduced to address this problem by using a special type of skip connection that allows the gradients to flow more easily through the network. By using these skip connections, ResNet enables very deep neural networks to be trained successfully, with improved accuracy and generalization performance.

There are several variations of the ResNet architecture, including:

- ResNet-18: This is a relatively shallow ResNet architecture with 18 layers.
- ResNet-34: This architecture has 34 layers and is slightly deeper than ResNet-18.
- ResNet-50: This architecture has 50 layers and is more complex than the previous two versions. It was the winning model of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015.
- ResNet-101: This architecture has 101 layers and is even more complex than ResNet-50.

- ResNet-152: This is the deepest ResNet architecture with 152 layers and achieved state-of-the-art performance on several benchmark datasets.

We are using ResNet-18 as it is enough for our problem statement and due to hardware limitations

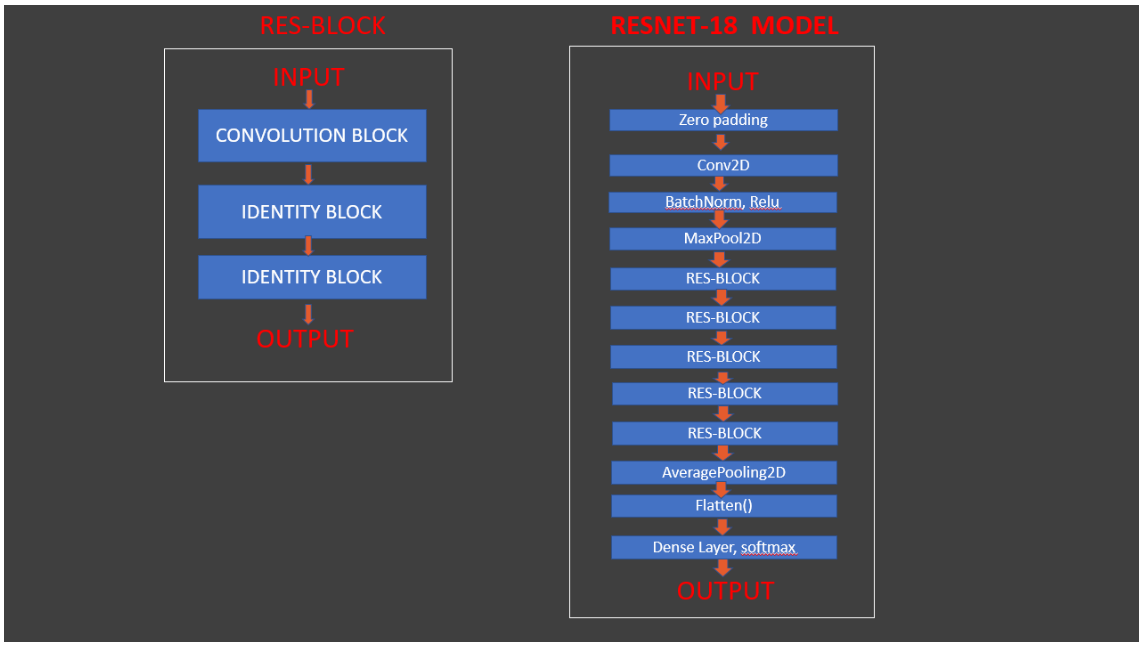


Fig 2.2 Architecture of RES-Block and RESNET-18 Model

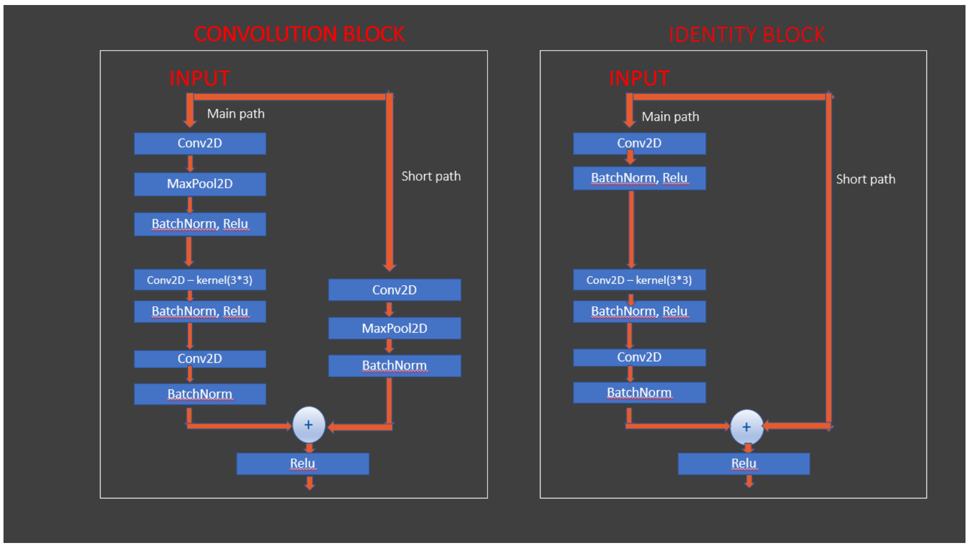


Fig 2.3 Convolution Block and Idenity block of RESNET-18 Model

Data augmentation is a technique used in machine learning and deep learning to artificially increase the size and diversity of a dataset by applying various transformations to the original data. These transformations create new data samples that are similar to the original data but have slight modifications, such as rotation, scaling, flipping, zooming, or changing brightness/contrast.

Data augmentation is commonly used in computer vision tasks, such as image classification, object detection, and image segmentation, to improve the performance and robustness of deep learning models. By augmenting the original dataset, data augmentation can help the model to learn more generalized and invariant features, reduce overfitting, and improve the model's ability to handle real-world variations and challenges, such as changes in lighting conditions, viewpoints, and occlusions.

Data augmentation can be easily implemented using data preprocessing techniques in popular deep learning frameworks, such as TensorFlow and Keras. During training, the augmented data is generated on-the-fly, and the original dataset is not modified. Common data augmentation techniques include rotation, flipping, zooming, shear, and changing brightness/contrast. The choice of data augmentation techniques depends on the specific task, dataset, and domain knowledge, and it is often used in combination with other regularization techniques, such as dropout and weight decay, to further improve model performance.

Confusion Matrix: It is very informative performance measures for classification tasks. Ci,j an element of matrix tells how many of items with label i are classified as label j. Ideally we are looking for diagonal Confusion matrix where no item is miss-classified. The matrix in Figure 1 is a good representation for our binary classification. Positive (P) represents toxic label and n (negative) represents non-toxic label.

Accuracy: This metric measures how many of the comments are labelled correctly. However, in our data set, where most of comments are not toxic, regardless of performance of model, a high accuracy was achieved.

Accuracy = (TP + TN)/(TP + TN + FP + FN)

Precision and Recall: Precision and recall in were designed to measure the model performance in its ability to correctly classify the toxic comments. Precision explains what fraction of toxic classified comments are truly toxic, and Recall measures what fraction of toxic comments are labelled correctly

Precision = TP/(TP + FN),   Recall = TP/(TP + TN)

F1 Score: Both Precision and Recall are important for checking the performance of the model. However, implementing a more advanced metric that combines both Precision and Recall together is quite informative and applicable. In this equation, setting β = 1 leads equation to return harmonic mean of Precision and Recall.

F1-Score = (Percision * Recall)/(Precision + Recall)

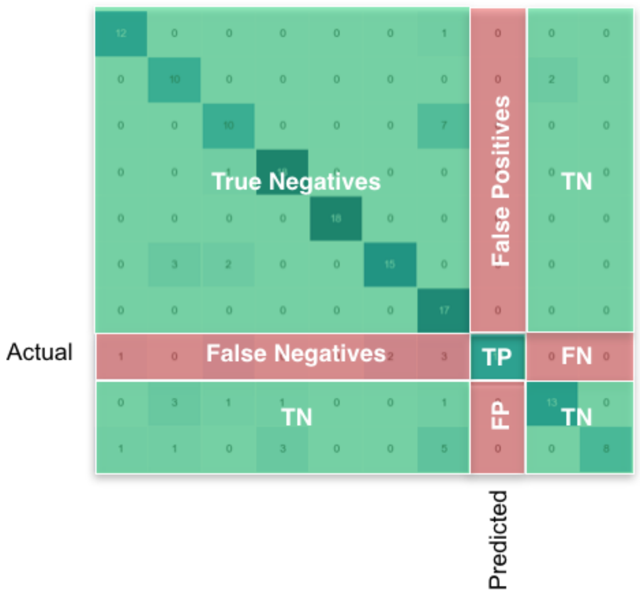


Fig 2.4 Confusion Matrix for Multi-class Classification problem

3. System Design and Methodology

3.1. System Architecture (Flow Chart)

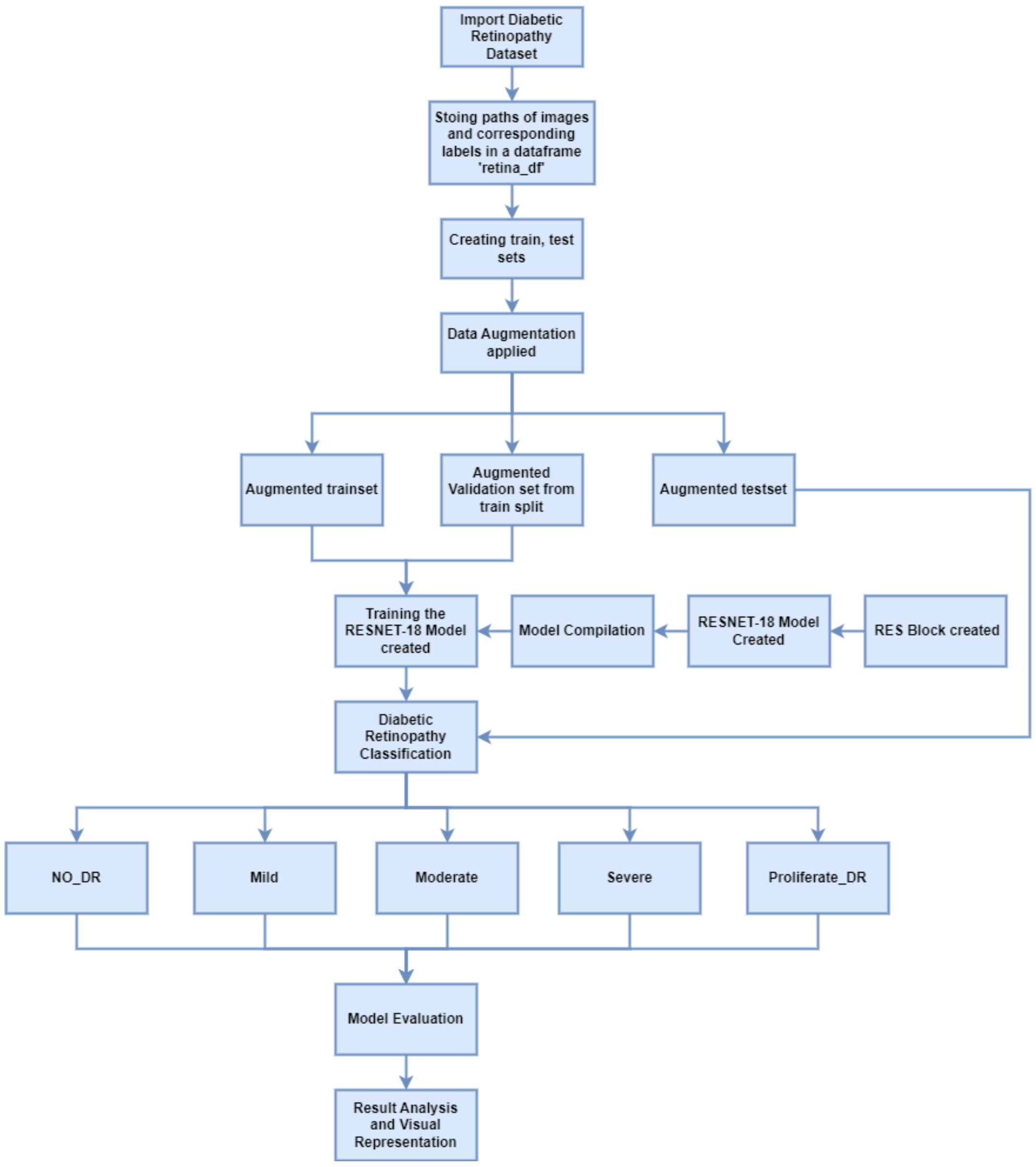

Import Diabetic Retinopathy Dataset
Stoing paths of images and corresponding labels in a dataframe 'retina_df'
Creating train, test sets
Data Augmentation applied
Augmented trainset
Augmented Validation set from train split
Augmented testset
Training the RESNET-18 Model created
Model Compilation
RESNET-18 Model Created
RES Block created
Diabetic Retinopathy Classification
NO_DR
Mild
Moderate
Severe
Proliferate_DR
Model Evaluation
Result Analysis and Visual Representation

Fig 3.1 Flow Chart

3.2. Development Environment

Hardware Requirement

- I5 processor or higher
- 8GB RAM or higher
- 200GB ROM or higher

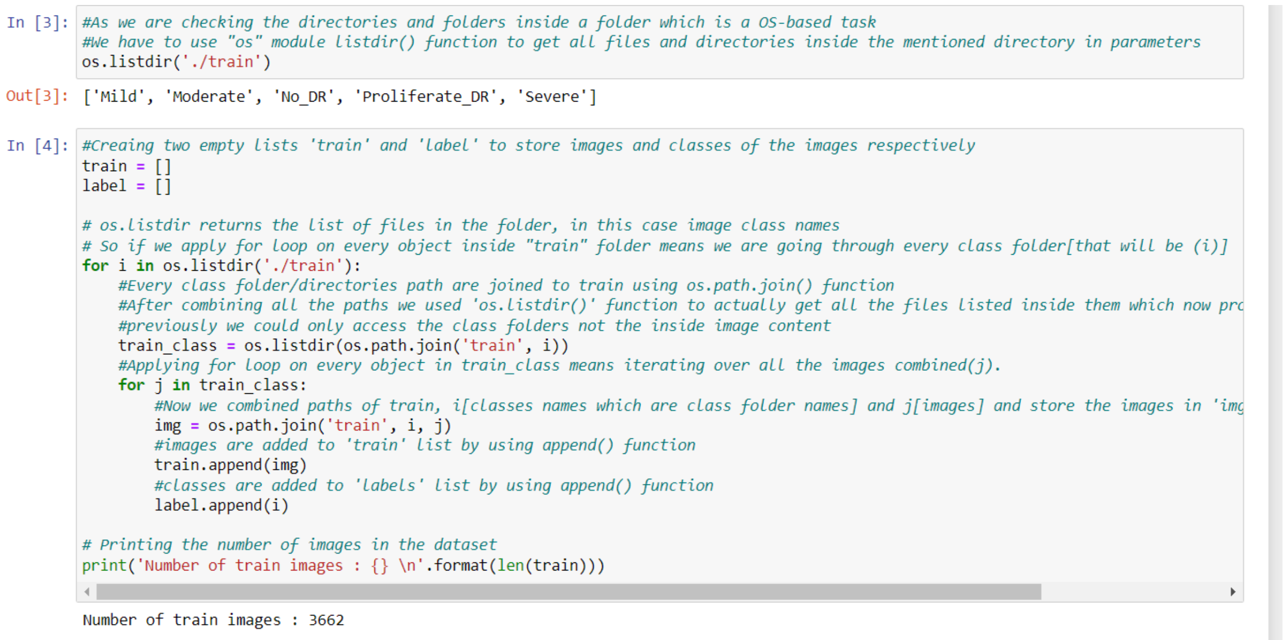
Software Requirement

- Windows 7 or higher
- Python 3.10 or higher
- Any python supported IDE-Jupyter Notebook, Google colab,etc.,

3.3. Methodology

Diabetic retinopathy(DR), a complication of diabetes, is a leading cause of blindness worldwide. Early detection and timely intervention can greatly reduce the risk of vision loss. With the advent of Artificial Intelligence (AI) and Convolutional Neural Networks (CNNs), there has been significant progress in developing automated systems for diabetic retinopathy detection.

Dataset:-We selected the Diabetic Retinopathy dataset from Kaggle's DR challenge dataset for multi-class classification as the dataset for this project. We took a subset of data (3662 images) from the original dataset which is a medium-sized dataset due to hardware constraints. The dataset contains fundus occuli images of five different severity levels such as No_DR, Mild, Moderate, Severe and Proliferate_DR.



```
In [3]: #As we are checking the directories and folders inside a folder which is a OS-based task
        #We have to use "os" module listdir() function to get all files and directories inside the mentioned directory in parameters
        os.listdir('./train')

Out[3]: ['Mild', 'Moderate', 'No_DR', 'Proliferate_DR', 'Severe']

In [4]: #Creaing two empty lists 'train' and 'label' to store images and classes of the images respectively
        train = []
        label = []

        # os.listdir returns the list of files in the folder, in this case image class names
        # So if we apply for loop on every object inside "train" folder means we are going through every class folder[that will be (i)]
        for i in os.listdir('./train'):
            #Every class folder/directories path are joined to train using os.path.join() function
            #After combining all the paths we used 'os.listdir()' function to actually get all the files listed inside them which now prc
            #previously we could only access the class folders not the inside image content
            train_class = os.listdir(os.path.join('train', i))
            #Applying for loop on every object in train_class means iterating over all the images combined(j).
            for j in train_class:
                #Now we combined paths of train, i[classes names which are class folder names] and j[images] and store the images in 'img
                img = os.path.join('train', i, j)
                #images are added to 'train' list by using append() function
                train.append(img)
                #classes are added to 'labels' list by using append() function
                label.append(i)

        # Printing the number of images in the dataset
        print('Number of train images : {} \n'.format(len(train)))

        Number of train images : 3662
```

Fig 3.2 Overview of Dataset

pd.DataFrame():- is a function in the pandas library of Python programming language that creates a 2-dimensional labeled data structure called a DataFrame. It can be thought of as a table in a relational database or a spreadsheet.

- The DataFrame consists of rows and columns, where each column can have a different data type such as numeric, string, boolean, datetime, etc. It is a flexible and powerful data structure that can handle large datasets efficiently and provide various operations such as filtering, sorting, merging, joining, grouping, pivoting, etc.
- The pd.DataFrame() function can take various types of input data such as lists, arrays, dictionaries, tuples, or other dataframes, and convert them into a new dataframe.[Here in the above example, train and label assigned to columns are lists]
- We stored all the paths of images and their labels in 'retina_df' dataframe for further tasks such as data preprocessing and data augmentation.

Data Augmentation:-Data augmentation is a technique used in machine learning and computer vision to artificially increase the size and diversity of a dataset by creating new examples through various transformations of the existing data. This technique can be especially useful when the original dataset is small or when the model needs to be trained on a wider range of variations of the input data to improve its accuracy and generalization capabilities.Data Augmentation can be done on images,audio,text,...

- **Some common data augmentation techniques for images include:**(As this project is based on images of retina):-

    - Geometric transformations: randomly flip, crop, rotate, stretch, and zoom images. You need to be careful about applying multiple transformations on the same images, as this can reduce model performance.
    - Color space transformations: randomly change RGB color channels, contrast, and brightness.
    - Kernel filters: randomly change the sharpness or blurring of the image.
    - Random erasing: delete some part of the initial image.
    - Mixing images: blending and mixing multiple images.

Data Generator:-Data generators are Python functions that can generate batches of augmented data on the fly during training, rather than storing all the augmented data in memory.

- This is particularly useful when dealing with large datasets that can't fit into memory, or when dealing with real-time data streams.
- The data augmentation techniques are used to generate new examples, while the data generator function is used to load and preprocess the augmented data on the fly during training.
- **ImageDataGenerator()** is a class from the tensorflow.keras.preprocessing.image module that generates augmented image data in batches.Example:- It releases 32 images if batch-size is 32 per iteration while training the model.
- **Parameters and methods used in creation of datagenerator:-**
    - **flow_from_dataframe() method:-** takes the Pandas DataFrame and the path to a directory and generates batches of augmented/normalized data.
    - **directory='./':-** The directory parameter specifies the location of the image files, which is set to ./ (i.e., the current directory).
    - **x_col and y_col:-** The x_col parameter specifies the name of the column in the DataFrame that contains the file names of the images(i.e., "Image" coloumn), and y_col specifies the column that contains the labels(i.e., "Labels" coloumn).
    - **target_size:-** The target_size parameter specifies the size of the input images that the model will expect.We lready know that all the retina images are of 256*256(width*height in pixels)
    - **color_mode:-** The color_mode parameter specifies the color mode of the images, which is set to RGB.
    - **class_mode:-** The class_mode parameter is set to "categorical" because the labels are one-hot encoded.
    - **batch_size:-** The batch_size parameter specifies the size of the batches that will be generated by the generator.We already know that data generator generates batches of augmented data.So the batch size(no.of images per iteration) is specified by this parameter

Data Normalization:-Normalization is a common data preprocessing technique used in machine learning and deep learning to scale the input data to a common range.

- In image processing, normalization typically involves scaling the pixel values of an image to a range of [0, 1].

CNN Model:- Convolutional Neural Networks (CNNs or ConvNets) are a class of deep neural networks specifically designed for image processing tasks, such as image classification, object detection, and image generation. CNNs have been widely used and have achieved state-of-the-art performance in various computer vision tasks.

- CNN is used in this project because it is better suited to for our project which involves image classification of fundus occuli(retina images).

RESNET-18 Model:- ResNet-18, short for Residual Network-18, is a convolutional neural network (CNN) architecture that was proposed by researchers at Microsoft Research in 2016. It is part of the ResNet family of CNN architectures, which are known for their deep network depths and residual connections that help address the problem of vanishing gradients during training. ResNet-18 specifically consists of 18 layers, including convolutional layers, pooling layers, fully connected layers, and skip connections (residual connections). The skip connections allow for the gradient to bypass certain layers, facilitating the training of very deep networks. ResNet-18 has been widely used for various computer vision tasks, such as image classification, object detection, and image segmentation, and has shown state-of-the-art performance in many benchmark datasets.

- As we are trying to build a deeper CNN network, it may cause us Vanishing Gradient problem which leads the model to overfit. So, we used RESNET-18 model which uses skip connections to solve vanishing gradient problem.
- The RESNET-18 model is trained and then evaluated using metrics.

4. Implementation & Result

4.1. Modules/Classes of Implemented Project

The Modules used in the following Project are as followed:

1. Pandas for cleaning the data and empty cells as well as the wrong format. It is used for pure Data Analysis Portion.
2. Numpy as it offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more
3. TensorFlow as it allows you to create dataflow graphs that describe how data moves through a graph. The graph consists of nodes that represent a mathematical operation. A connection or edge between nodes is a multidimensional data array
4. OS as this module provides a portable way of using operating system dependent functionality.
5. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.
6. The Python Imaging Library adds image processing capabilities to your Python interpreter.

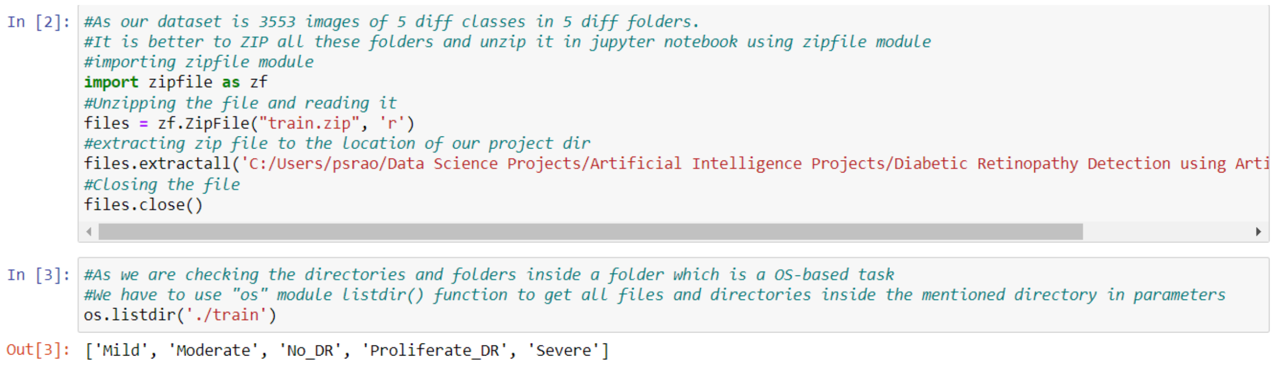
   This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

These are the MAIN Modules used , with some others too. Many packages are used from the inbuilt functions of these modules.

4.2. Implementation Detail

4.2.1. Loading Dataset

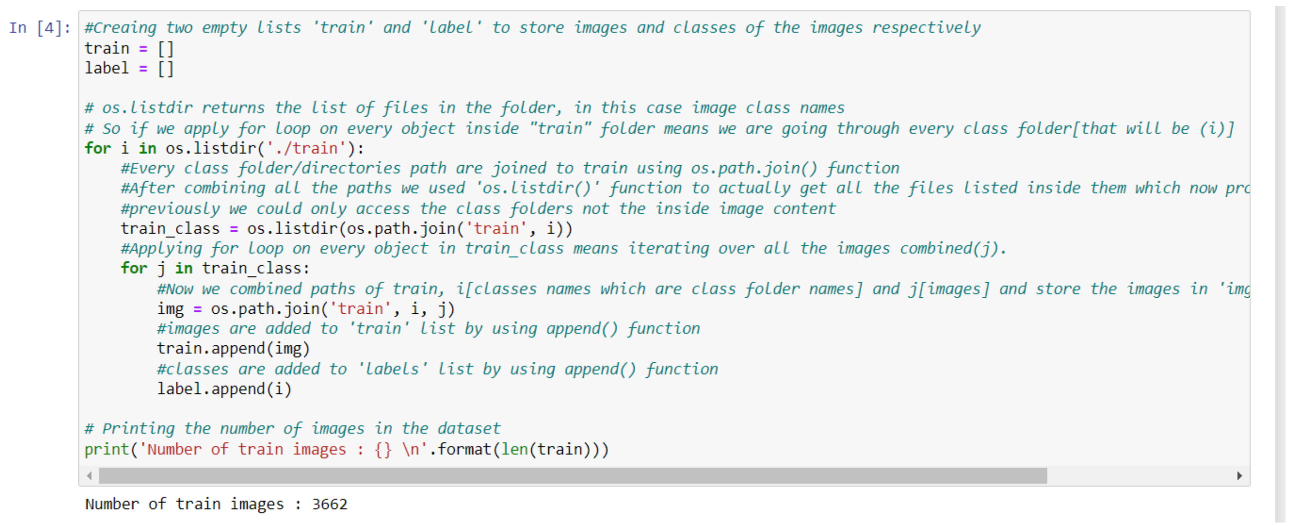
- We first extract out dataset from the ZIP File and use a os module function to get all files and directories inside the mentioned directory in parameters.



```
In [2]: #As our dataset is 3553 images of 5 diff classes in 5 diff folders.
        #It is better to ZIP all these folders and unzip it in jupyter notebook using zipfile module
        #importing zipfile module
        import zipfile as zf
        #Unzipping the file and reading it
        files = zf.ZipFile("train.zip", 'r')
        #extracting zip file to the location of our project dir
        files.extractall('C:/Users/psrao/Data Science Projects/Artificial Intelligence Projects/Diabetic Retinopathy Detection using Arti
        #Closing the file
        files.close()

In [3]: #As we are checking the directories and folders inside a folder which is a OS-based task
        #We have to use "os" module listdir() function to get all files and directories inside the mentioned directory in parameters
        os.listdir('./train')

Out[3]: ['Mild', 'Moderate', 'No_DR', 'Proliferate_DR', 'Severe']
```

Fig 4.1 Loading Dataset

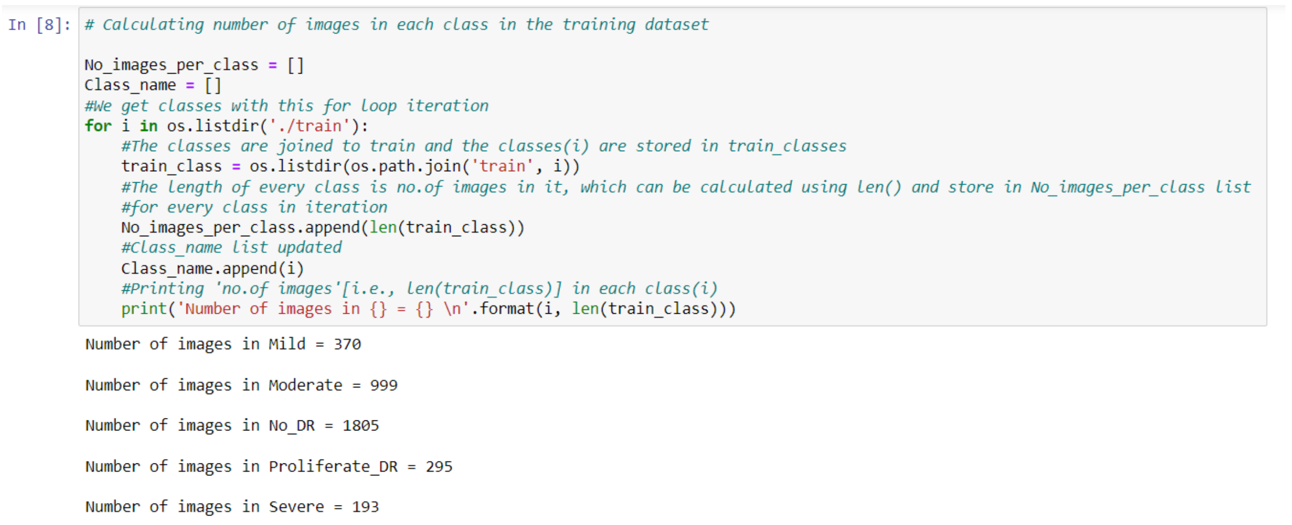
- Creaing two empty lists 'train' and 'label' to store images and classes of the images respectively and then apply a loop on every object of the "train" folder. We print the total number of datasets as shown in the given picture.



```
In [4]: #Creaing two empty lists 'train' and 'label' to store images and classes of the images respectively
        train = []
        label = []

        # os.listdir returns the list of files in the folder, in this case image class names
        # So if we apply for loop on every object inside "train" folder means we are going through every class folder[that will be (i)]
        for i in os.listdir('./train'):
            #Every class folder/directories path are joined to train using os.path.join() function
            #After combining all the paths we used 'os.listdir()' function to actually get all the files listed inside them which now prc
            #previously we could only access the class folders not the inside image content
            train_class = os.listdir(os.path.join('train', i))
            #Applying for loop on every object in train_class means iterating over all the images combined(j).
            for j in train_class:
                #Now we combined paths of train, i[classes names which are class folder names] and j[images] and store the images in 'img
                img = os.path.join('train', i, j)
                #images are added to 'train' list by using append() function
                train.append(img)
                #classes are added to 'labels' list by using append() function
                label.append(i)

        # Printing the number of images in the dataset
        print('Number of train images : {} \n'.format(len(train)))

        Number of train images : 3662
```

4.2.2. Data Exploration And Visualization

- We visualise 5 images in each class of the dataset. We are creating 5*5 grid. 5 images for each of the five classes. Height and width in figsize is set at 20 . Then we Initialize a count variable with 0. So that after incrementing we can iterate over to other classes. In the end we calcualte the total number of images in each class of our training Dataset.



```
In [8]: # Calculating number of images in each class in the training dataset

        No_images_per_class = []
        Class_name = []
        #We get classes with this for loop iteration
        for i in os.listdir('./train'):
            #The classes are joined to train and the classes(i) are stored in train_classes
            train_class = os.listdir(os.path.join('train', i))
            #The length of every class is no.of images in it, which can be calculated using len() and store in No_images_per_class list
            #for every class in iteration
            No_images_per_class.append(len(train_class))
            #Class_name list updated
            Class_name.append(i)
            #Printing 'no.of images'[i.e., len(train_class)] in each class(i)
            print('Number of images in {} = {} \n'.format(i, len(train_class)))

        Number of images in Mild = 370

        Number of images in Moderate = 999

        Number of images in No_DR = 1805

        Number of images in Proliferate_DR = 295

        Number of images in Severe = 193
```

- We plot a bar graph to get to know the number of images under each class.
- We also create a Pie-Chart for better Visualization.(To plot a pie chart we need the input values to be in data structures with sequential numerical values and labels corresponding to them in the same length. Some ex of data structures of this type are dictionaries, lists, and tuples,...So we took 'No_images_per_class'(which contains the no.of images of each list as shown above) list as 'x' to plot the piechart And we took 'Class_name'(which contains the labels corresponding to No_images_per_class) to name/label the pies of chart plotted we set the autopct parameter to '%1.1f%%', which formats the percentage labels to one decimal place)

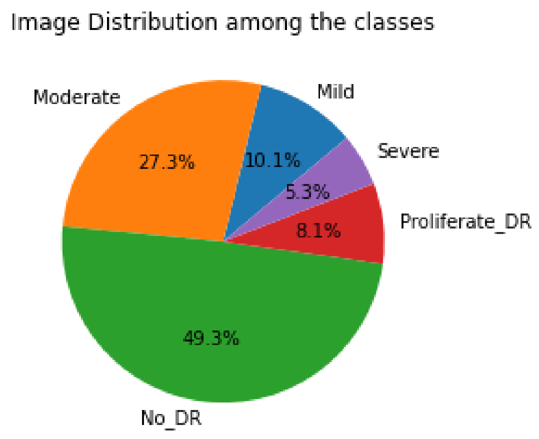


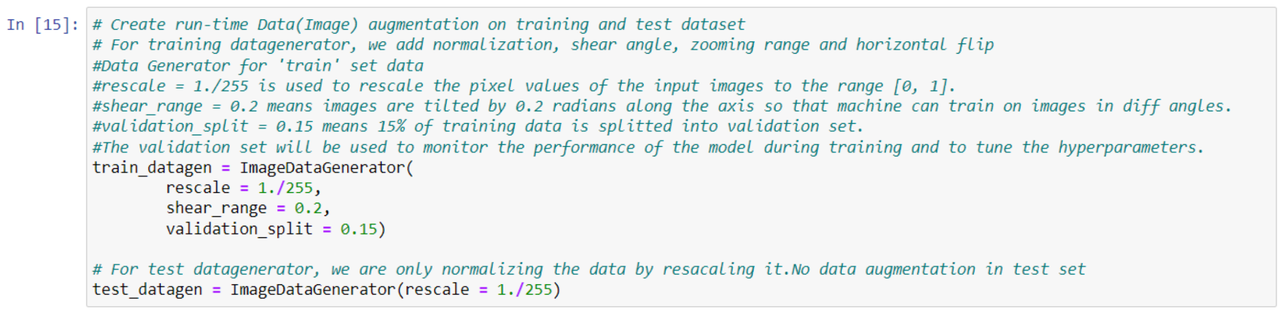
Fig 4.2 Pie chart on no.of images in dataset

4.2.3. Data Augmentation And Data Generator

Data Augmentation:-Data augmentation is a technique used in machine learning and computer vision to artificially increase the size and diversity of a dataset by creating new examples through various transformations of the existing data. This technique can be especially useful when the original dataset is small or when the model needs to be trained on a wider range of variations of the input data to improve its accuracy and generalization capabilities.

- We Shuffle the data in dataframe 'retina_df' using shuffle function so that we don't get the same class images in test data. Shuffling the data makes us get to train and test sets with a mix of all class images
- We split into a train and test sets which is about 80-20 %

Data Generator:-Data generators are Python functions that can generate batches of augmented data on the fly during training, rather than storing all the augmented data in memory.
We Create a train_datagen and test_datagen objects using Image DATA Generator() class for generating test,train and validation sets in the next step:



```
In [15]: # Create run-time Data(Image) augmentation on training and test dataset
         # For training datagenerator, we add normalization, shear angle, zooming range and horizontal flip
         #Data Generator for 'train' set data
         #rescale = 1./255 is used to rescale the pixel values of the input images to the range [0, 1].
         #shear_range = 0.2 means images are tilted by 0.2 radians along the axis so that machine can train on images in diff angles.
         #validation_split = 0.15 means 15% of training data is splitted into validation set.
         #The validation set will be used to monitor the performance of the model during training and to tune the hyperparameters.
         train_datagen = ImageDataGenerator(
                 rescale = 1./255,
                 shear_range = 0.2,
                 validation_split = 0.15)

         # For test datagenerator, we are only normalizing the data by resacaling it.No data augmentation in test set
         test_datagen = ImageDataGenerator(rescale = 1./255)
```

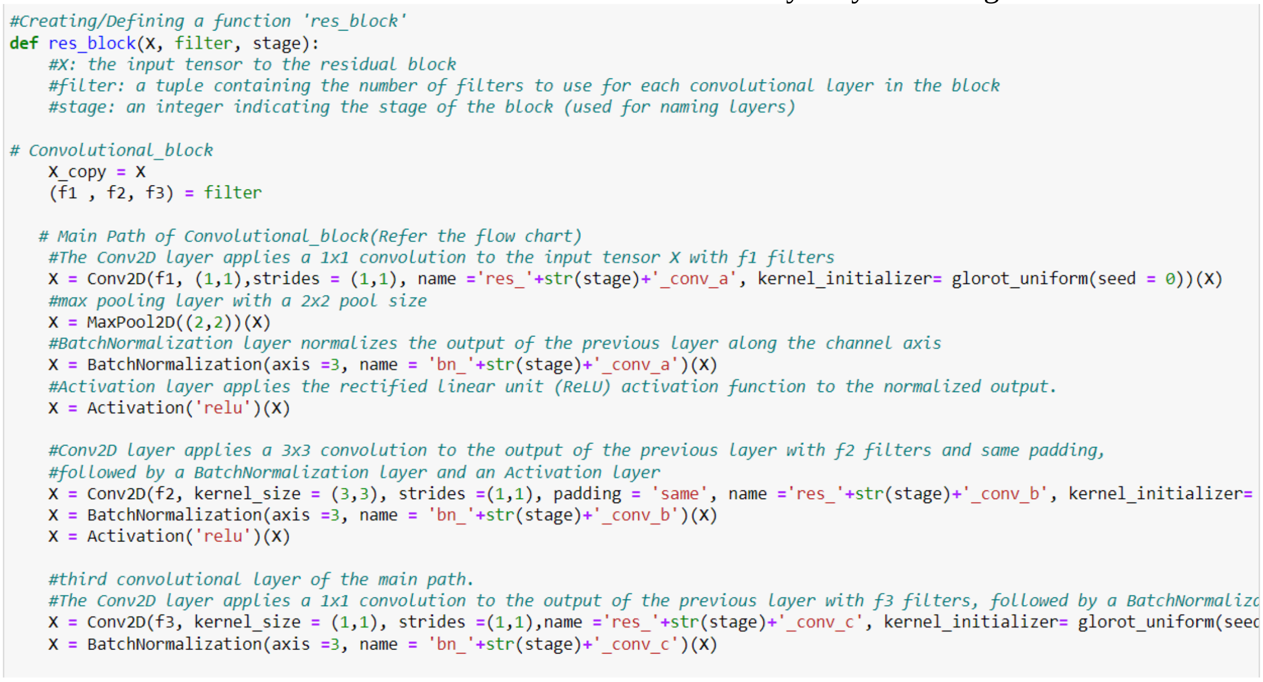
Fig 4.3 Code Snippet on data generator for adata augmentation

4.2.4. Res-Block Based Dl Model

- In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks. In this network, we use a technique called skip connections. The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. Resnets are made by stacking these residual blocks together.

- We First create the RES-Blocks for the RESNET and create a function res-block in the process which will use

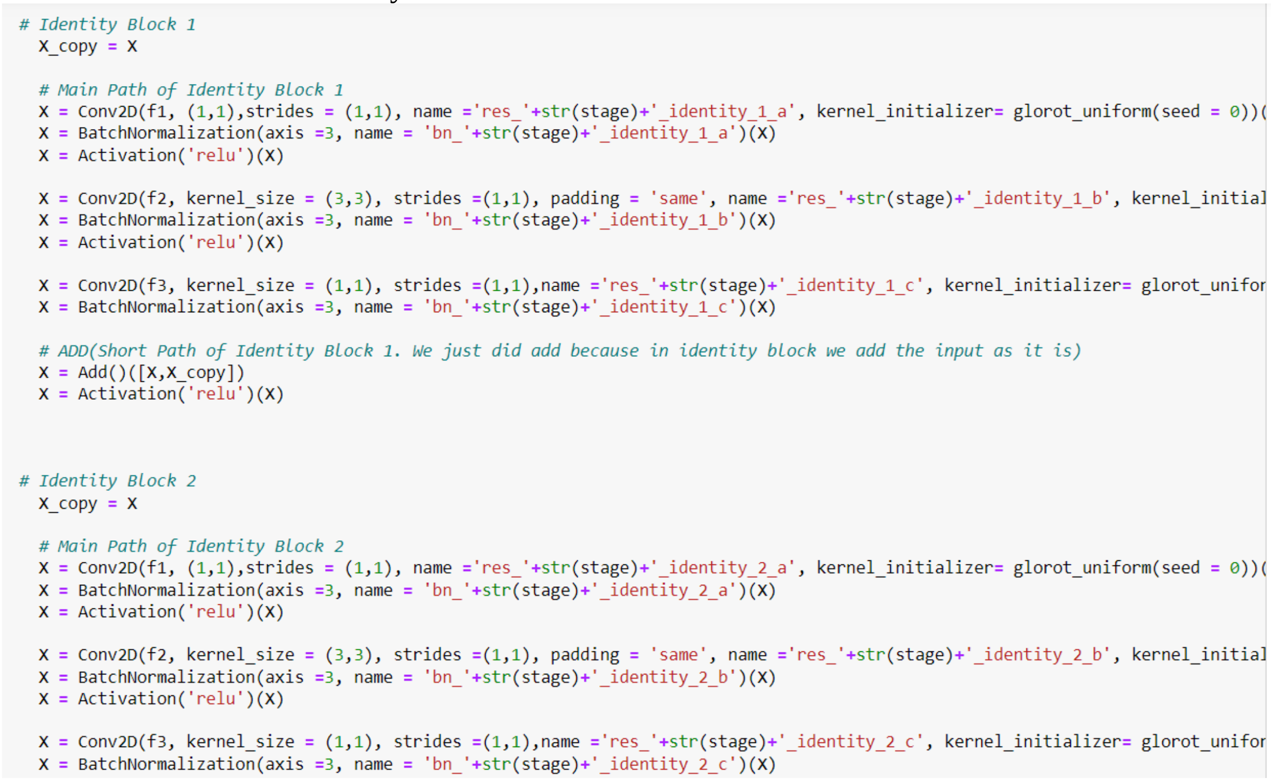
  - X: the input tensor to the residual block
  - filter: a tuple containing the number of filters to use for each convolutional layer in the block
- We can find the Main Path of the Convolutional layer by following the flow chart.



```
#Creating/Defining a function 'res_block'
def res_block(X, filter, stage):
    #X: the input tensor to the residual block
    #filter: a tuple containing the number of filters to use for each convolutional layer in the block
    #stage: an integer indicating the stage of the block (used for naming layers)

# Convolutional_block
    X_copy = X
    (f1 , f2, f3) = filter

   # Main Path of Convolutional_block(Refer the flow chart)
    #The Conv2D layer applies a 1x1 convolution to the input tensor X with f1 filters
    X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_conv_a', kernel_initializer= glorot_uniform(seed = 0))(X)
    #max pooling layer with a 2x2 pool size
    X = MaxPool2D((2,2))(X)
    #BatchNormalization layer normalizes the output of the previous layer along the channel axis
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_a')(X)
    #Activation layer applies the rectified linear unit (ReLU) activation function to the normalized output.
    X = Activation('relu')(X)

    #Conv2D layer applies a 3x3 convolution to the output of the previous layer with f2 filters and same padding,
    #followed by a BatchNormalization layer and an Activation layer
    X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+str(stage)+'_conv_b', kernel_initializer=
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_b')(X)
    X = Activation('relu')(X)

    #third convolutional layer of the main path.
    #The Conv2D layer applies a 1x1 convolution to the output of the previous layer with f3 filters, followed by a BatchNormaliz
    X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_conv_c', kernel_initializer= glorot_uniform(see
    X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_c')(X)
```

Fig 4.4 Convolution block of RES-Block

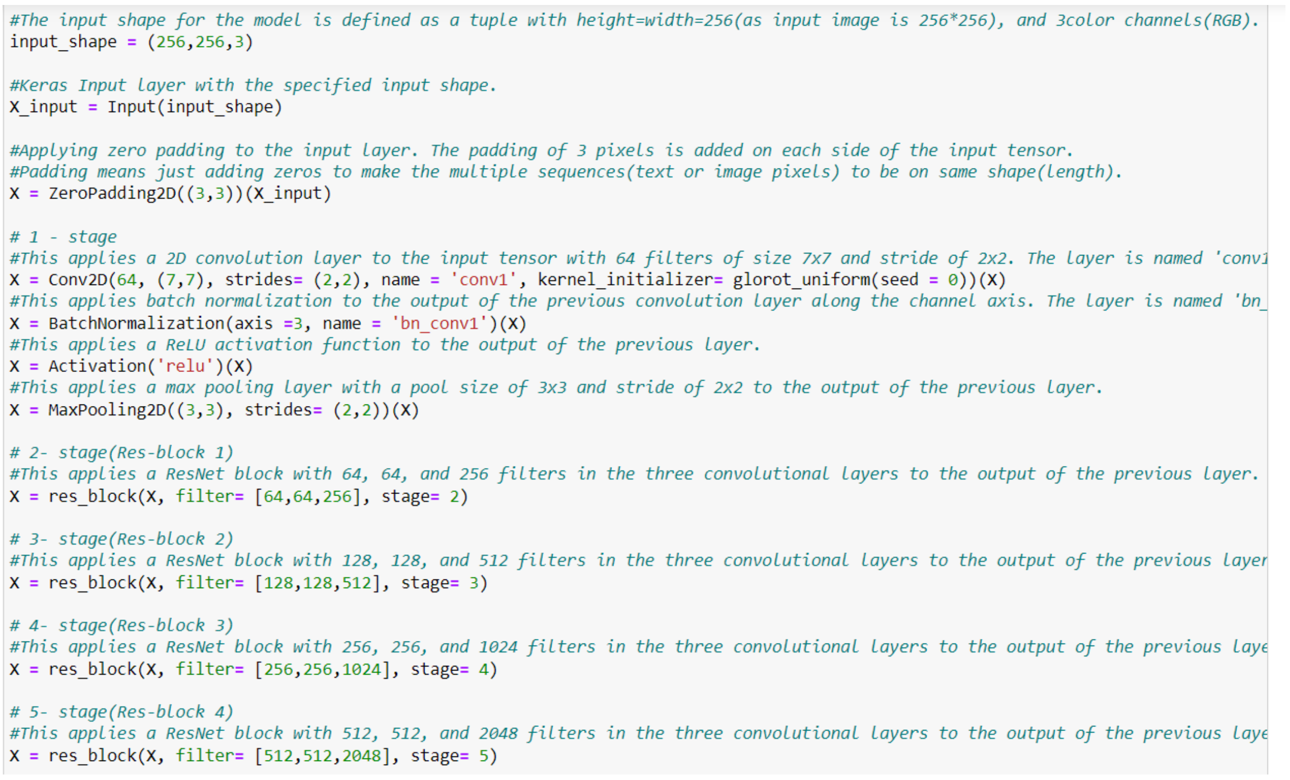
- We create 2 Identity Blocks after this function.



```
# Identity Block 1
  X_copy = X

  # Main Path of Identity Block 1
  X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_identity_1_a', kernel_initializer= glorot_uniform(seed = 0))(
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_a')(X)
  X = Activation('relu')(X)

  X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+str(stage)+'_identity_1_b', kernel_initial
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_b')(X)
  X = Activation('relu')(X)

  X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_identity_1_c', kernel_initializer= glorot_unifor
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_c')(X)

  # ADD(Short Path of Identity Block 1. We just did add because in identity block we add the input as it is)
  X = Add()([X,X_copy])
  X = Activation('relu')(X)



# Identity Block 2
  X_copy = X

  # Main Path of Identity Block 2
  X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_identity_2_a', kernel_initializer= glorot_uniform(seed = 0))(
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_a')(X)
  X = Activation('relu')(X)

  X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+str(stage)+'_identity_2_b', kernel_initial
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_b')(X)
  X = Activation('relu')(X)

  X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_identity_2_c', kernel_initializer= glorot_unifor
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_c')(X)
```

Fig 4.5 Identity blocks of RES-Block

- We create the Model for our RESNET afterwards.



```
#The input shape for the model is defined as a tuple with height=width=256(as input image is 256*256), and 3color channels(RGB).
input_shape = (256,256,3)

#Keras Input layer with the specified input shape.
X_input = Input(input_shape)

#Applying zero padding to the input layer. The padding of 3 pixels is added on each side of the input tensor.
#Padding means just adding zeros to make the multiple sequences(text or image pixels) to be on same shape(length).
X = ZeroPadding2D((3,3))(X_input)

# 1 - stage
#This applies a 2D convolution layer to the input tensor with 64 filters of size 7x7 and stride of 2x2. The layer is named 'conv1
X = Conv2D(64, (7,7), strides= (2,2), name = 'conv1', kernel_initializer= glorot_uniform(seed = 0))(X)
#This applies batch normalization to the output of the previous convolution layer along the channel axis. The layer is named 'bn_
X = BatchNormalization(axis =3, name = 'bn_conv1')(X)
#This applies a ReLU activation function to the output of the previous layer.
X = Activation('relu')(X)
#This applies a max pooling layer with a pool size of 3x3 and stride of 2x2 to the output of the previous layer.
X = MaxPooling2D((3,3), strides= (2,2))(X)

# 2- stage(Res-block 1)
#This applies a ResNet block with 64, 64, and 256 filters in the three convolutional layers to the output of the previous layer.
X = res_block(X, filter= [64,64,256], stage= 2)

# 3- stage(Res-block 2)
#This applies a ResNet block with 128, 128, and 512 filters in the three convolutional layers to the output of the previous layer
X = res_block(X, filter= [128,128,512], stage= 3)

# 4- stage(Res-block 3)
#This applies a ResNet block with 256, 256, and 1024 filters in the three convolutional layers to the output of the previous laye
X = res_block(X, filter= [256,256,1024], stage= 4)

# 5- stage(Res-block 4)
#This applies a ResNet block with 512, 512, and 2048 filters in the three convolutional layers to the output of the previous laye
X = res_block(X, filter= [512,512,2048], stage= 5)
```

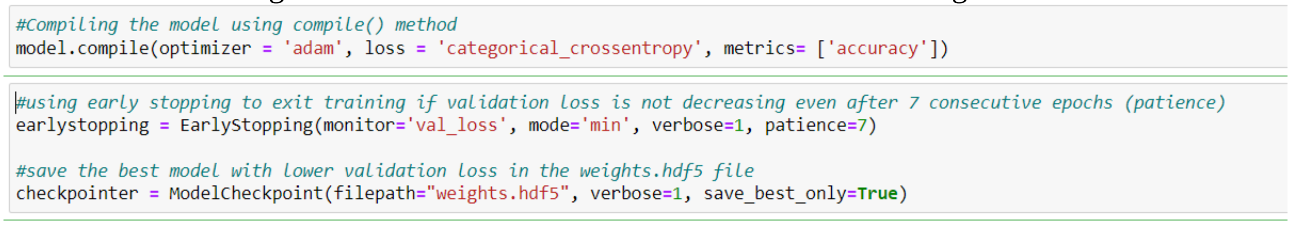
Fig 4.6 RESNET-18 Model Creation

4.2.5. Summary of our Model:-

- Total paramaters(Trainable+Non-Trainable): 19,950,213
- Trainable parameters: 19,907,845
- Non-trainable parameters: 42,368
- Parameters:-parameters generally refer to the weights and biases of the neurons.
- Trainable parameters:-Type of Parameters(weights of neurons) that can be updated in training.During the training,Optimizer algorithm updates these weights to minimize the loss function and improve the accuracy of the model.
- Non-trainable parameters:-Type of Parameters(weights of neurons) that cannot be updated in training.These typically include the parameters of BatchNormalization layers and other regularization layers.

4.2.6. Model Compilation

- Compiling the model using compile() method #using early stopping to exit training if validation loss is not decreasing even after 7 consecutive epochs (patience)
- Saving the best model with lower validation loss in the weights.hdf5 file



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

```
#using early stopping to exit training if validation loss is not decreasing even after 7 consecutive epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=7)

#save the best model with lower validation loss in the weights.hdf5 file
checkpointer = ModelCheckpoint(filepath="weights.hdf5", verbose=1, save_best_only=True)
```

Fig 4.7 Model Compilation

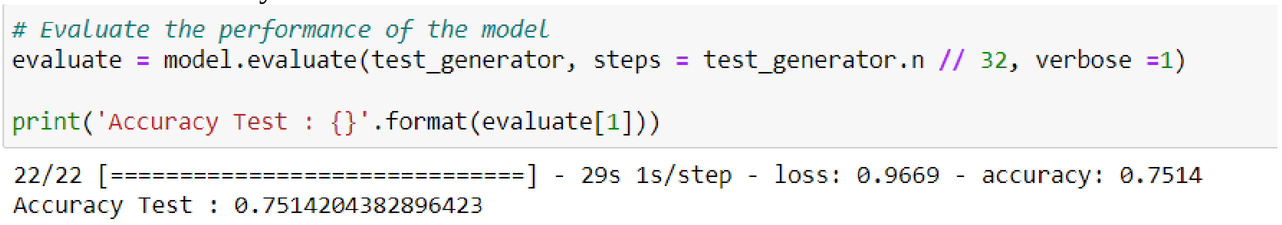
4.2.7. Training The Model

We fit the dataset into our Model and then use 24 epochs in the initial training of the model. We use validation_generator and validation_steps.

- Training accuracy is 92%.

4.2.8. Performance

Test Accuracy of our model comes to near 75%.



```
# Evaluate the performance of the model
evaluate = model.evaluate(test_generator, steps = test_generator.n // 32, verbose =1)

print('Accuracy Test : {}'.format(evaluate[1]))
```

```
22/22 [==============================] - 29s 1s/step - loss: 0.9669 - accuracy: 0.7514
Accuracy Test : 0.7514204382896423
```

Fig 4.8 Performance of RESNET-18 Model

- We Load up the Images and their predictions and then again look at the model accuracy on our test Images. We get a Accuracy of 75% on the following.

4.3. Results and Discussion

4.3.1 Observations from No.of Images in each class:

- We can now see that it is a unbalanced dataset with 'No_DR' and 'Moderate' classes as major classes and 'Mild' , 'Proliferative_DR' and 'Severe' as minor classes

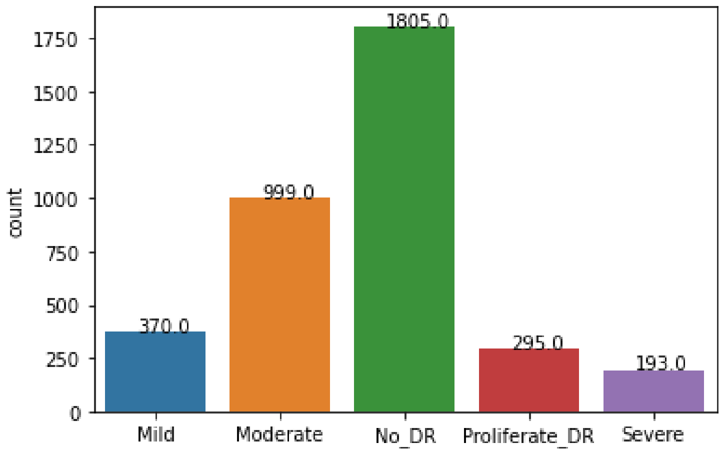
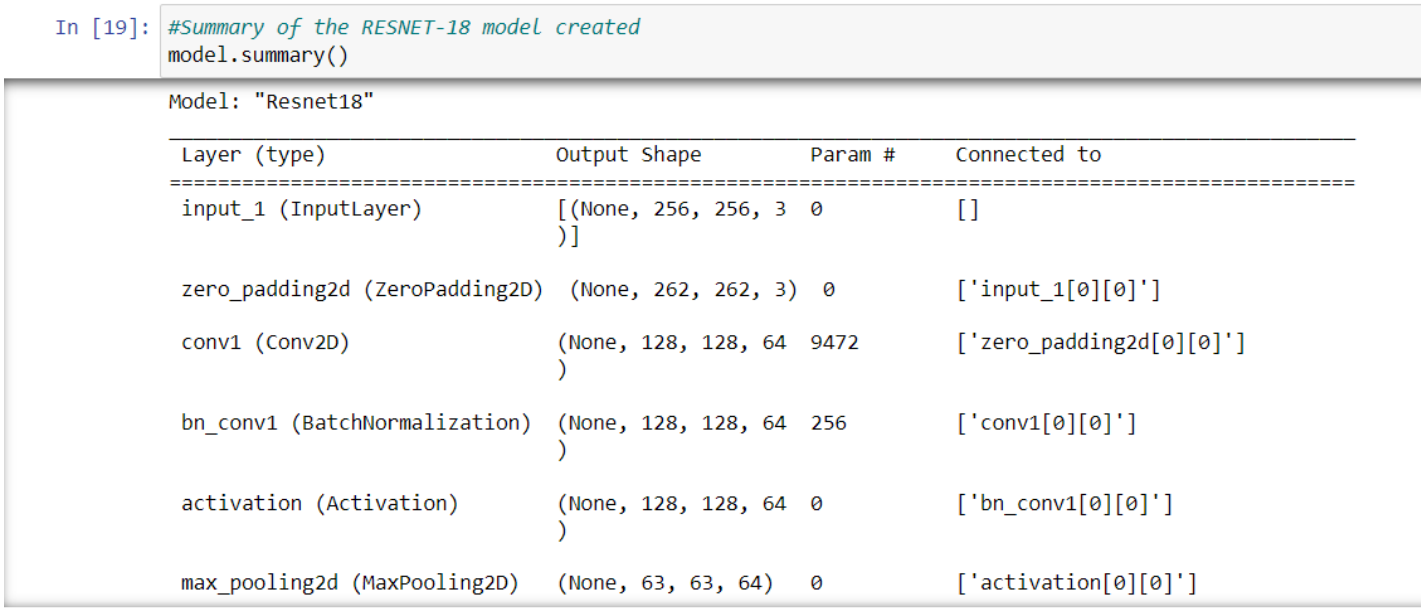


Fig 4.9 Barchart on no.of images in each class

4.3.2 Summary of the created model



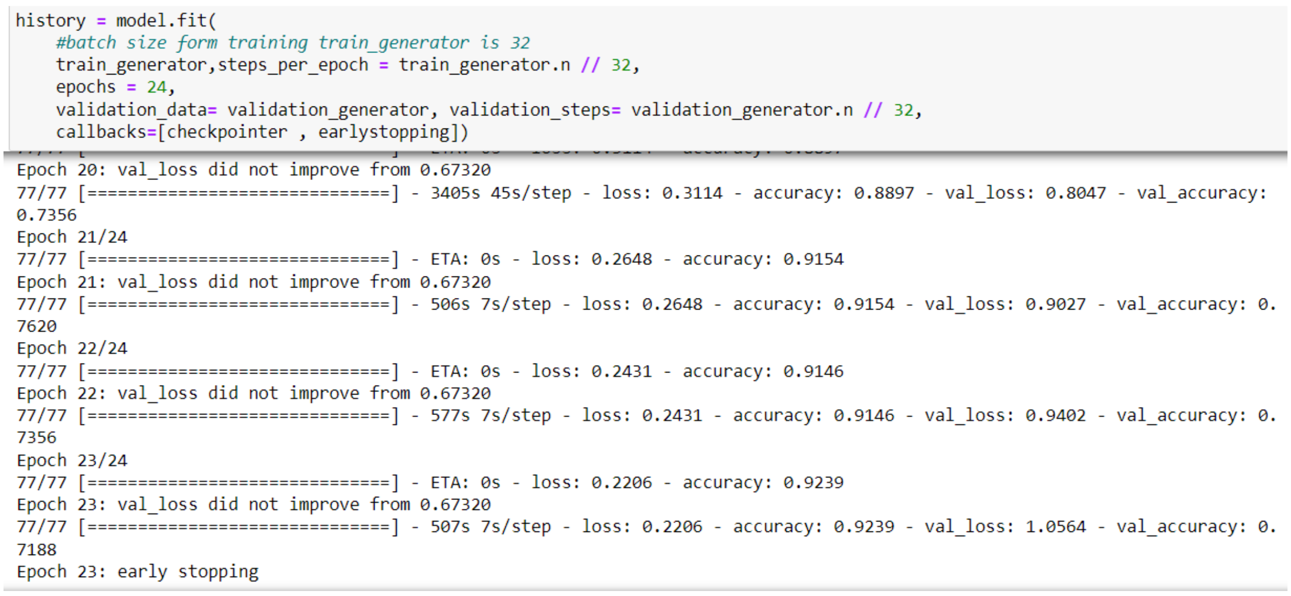
```
In [19]: #Summary of the RESNET-18 model created
         model.summary()
```

```
Model: "Resnet18"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 zero_padding2d (ZeroPadding2D)  (None, 262, 262, 3)  0           ['input_1[0][0]']

 conv1 (Conv2D)                 (None, 128, 128, 64  9472        ['zero_padding2d[0][0]']
                                )

 bn_conv1 (BatchNormalization)  (None, 128, 128, 64  256         ['conv1[0][0]']
                                )

 activation (Activation)        (None, 128, 128, 64  0           ['bn_conv1[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 63, 63, 64)   0           ['activation[0][0]']
```

Fig 4.10 Summary of RESNET-18 model created

4.3.3 Observations from the training are:-

- Accuracy of train set:92%
- Validation accuracy or accuracy on validation set:71%
- The val_accuracy is not improving for continuous seven epochs(patience=7). So, training stopped at 23rd epoch.



```
history = model.fit(
    #batch size form training train_generator is 32
    train_generator,steps_per_epoch = train_generator.n // 32,
    epochs = 24,
    validation_data= validation_generator, validation_steps= validation_generator.n // 32,
    callbacks=[checkpointer , earlystopping])
```

```
Epoch 20: val_loss did not improve from 0.67320
77/77 [==============================] - 3405s 45s/step - loss: 0.3114 - accuracy: 0.8897 - val_loss: 0.8047 - val_accuracy:
0.7356
Epoch 21/24
77/77 [==============================] - ETA: 0s - loss: 0.2648 - accuracy: 0.9154
Epoch 21: val_loss did not improve from 0.67320
77/77 [==============================] - 506s 7s/step - loss: 0.2648 - accuracy: 0.9154 - val_loss: 0.9027 - val_accuracy: 0.
7620
Epoch 22/24
77/77 [==============================] - ETA: 0s - loss: 0.2431 - accuracy: 0.9146
Epoch 22: val_loss did not improve from 0.67320
77/77 [==============================] - 577s 7s/step - loss: 0.2431 - accuracy: 0.9146 - val_loss: 0.9402 - val_accuracy: 0.
7356
Epoch 23/24
77/77 [==============================] - ETA: 0s - loss: 0.2206 - accuracy: 0.9239
Epoch 23: val_loss did not improve from 0.67320
77/77 [==============================] - 507s 7s/step - loss: 0.2206 - accuracy: 0.9239 - val_loss: 1.0564 - val_accuracy: 0.
7188
Epoch 23: early stopping
```

Fig 4.11 Epochs of training

4.3.1 Displaying an example prediction

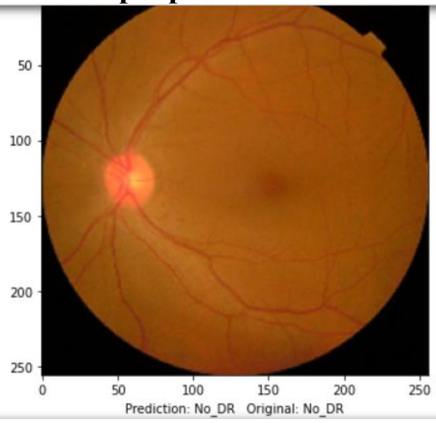


Fig 4.12 Visualizing an example prediction

4.3.2 Loss of training and validation sets

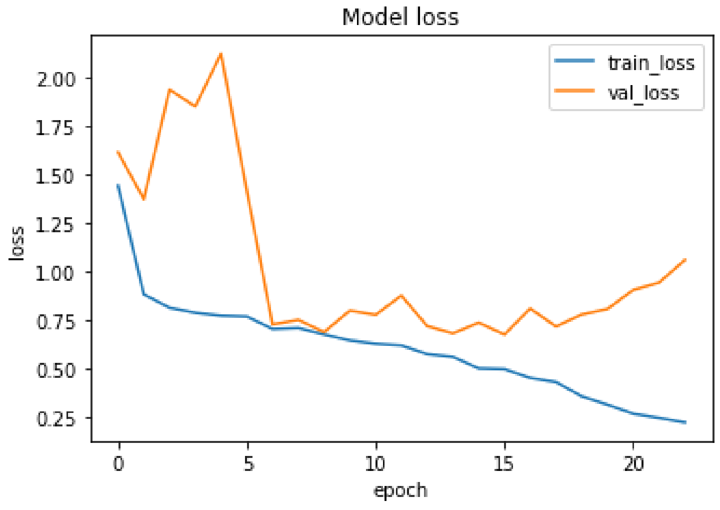


Fig 4.13 Loss of training and validation sets

4.3.1 Confusion Matrix

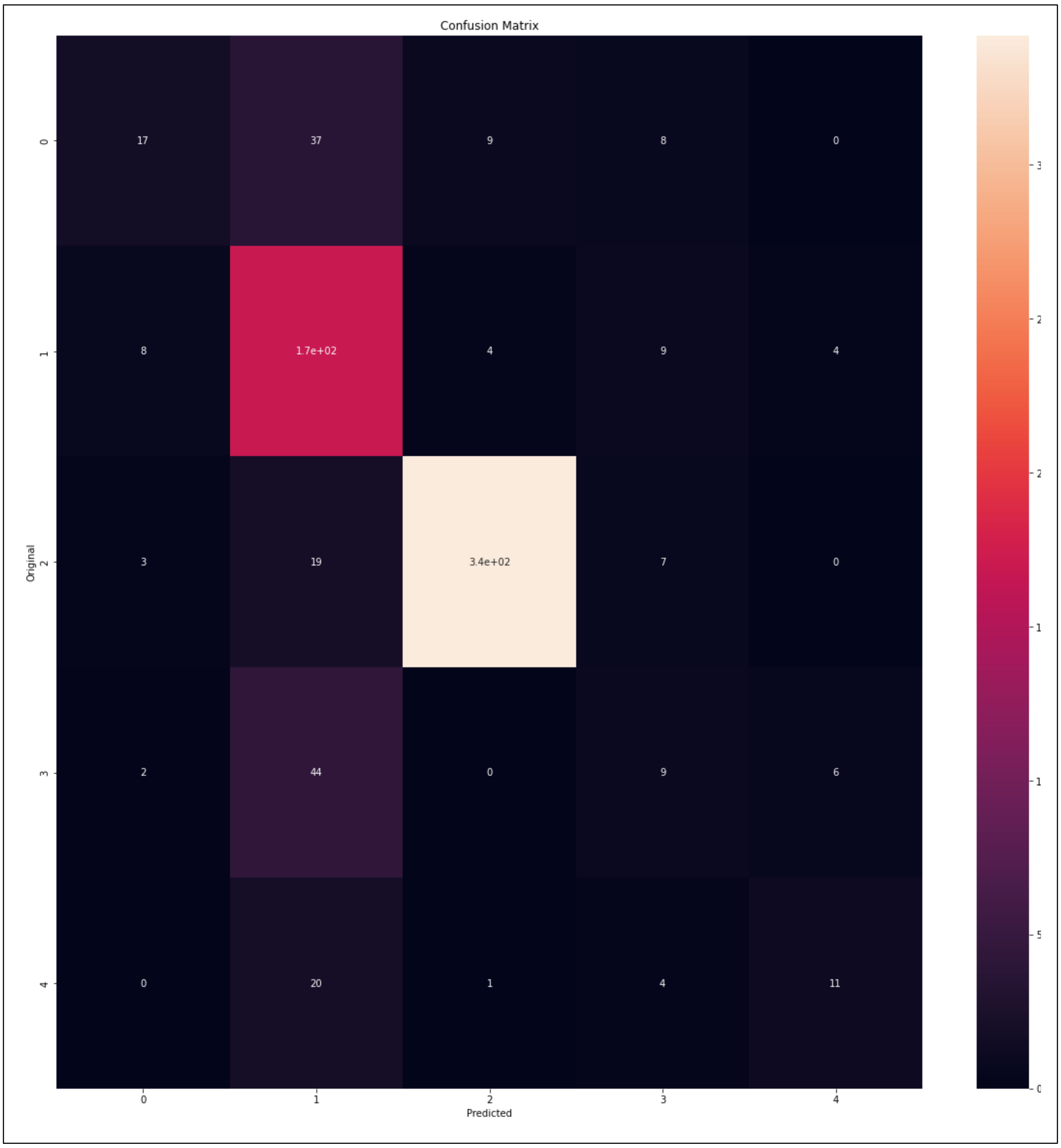
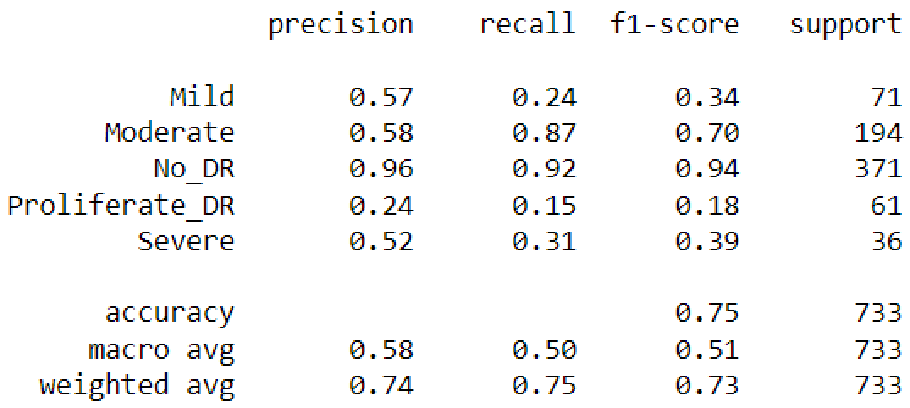


Fig 4.14 Confusion matrix

4.3.2 Explanation of Classification report

- We have six classes which originally represent anger, surprise, sadness, fear, joy, and love respectively.
- Overall Accuracy is 75%



```
                precision    recall  f1-score   support

          Mild       0.57      0.24      0.34        71
      Moderate       0.58      0.87      0.70       194
         No_DR       0.96      0.92      0.94       371
Proliferate_DR       0.24      0.15      0.18        61
        Severe       0.52      0.31      0.39        36

      accuracy                           0.75       733
     macro avg       0.58      0.50      0.51       733
  weighted avg       0.74      0.75      0.73       733
```

Fig 4.15 Classification report of all classes/labels

4.4. Month wise plan of work

- 4th February 2023- Deciding on topics of projects
- 11th February 2022- First initial draft of project
- 11th to 20th February 2023- Data Exploration
- 1st to 7th March 2023- Data preprocessing
- 8th to 20th March 2023- CNN and RESNET-18 Model
- 21st to 26th March 2023- RESNET-18 Model Evaluation
- 25th to 31st March 2023- Results and Analysis
- 12th to 16th April 2023- Preparation of project report and presentation

5. Conclusion and Future Plan

We implemented a RESNET-18 model on the Diabetic Retinopathy dataset to classify the fundus occuli(Retina images) into five main classes based on their severity such as No_DR, Mild, Moderate, Severe and Proliferate_DR. This technique proves to be efficient for the multiclass classification problem. We have applied data augmentation on the dataset to improve the diversity in the preprocessed images. RESNET-18 Model(type of CNN) applied on the emotion dataset stands out with overall training accuracy 92% and test accuracy 75%.

We used python as a language in this project to do all the necessary steps like loading dataset, data exploration, pre-processing data, data augmentation, splitting of dataset into train, test and validation sets, building RESNET-18 model and evaluation of models, etc.,

There are still some limitations of this project as we used 3662 only which is considered to be a medium-sized dataset. The accuracy may somewhat drop due to this as neural network models are data-hungry. We can solve this problem to some extent using methods like transfer learning which is useful for smaller datasets.

We plan to implement the same model on much larger datasets compared to the present dataset. We also plan to implement transfer learning on the same model to improve test accuracy. This will help the model to work efficiently on unseen data. We will implement the feature extraction step using a pre-trained model and apply it to various algorithms, such as support vector machines, to enhance the performance of the model. By incorporating measures such as specificity and sensitivity, we aim to increase the trustworthiness of the healthcare model for real-time usage. Additionally, we will explore different image pre-processing techniques on the dataset and compare their performances. Furthermore, we will evaluate and compare various transfer learning techniques and apply pre-trained models to address complex image classification challenges in real-world scenarios.

6. References

Web

[1] Diabetic Retinopathy Dataset, https://www.kaggle.com/c/diabetic-retinopathy-detection
[2] Data Augmentation, https://www.datacamp.com/tutorial/complete-guide-data-augmentation
[3] How to Configure Image Data Augmentation in Keras, https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
[4] Convolutional Neural Networks, Explained, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
[5]RESNET-18 Model, https://www.kaggle.com/datasets/pytorch/resnet18
[6] RESNET 18 Pytorch documentation, https://pytorch.org/vision/master/models/generated/torchvision.models.resnet18.html
[7] Keras Model creation and compilation https://www.tutorialspoint.com/keras/keras_model_compilation.htm
[8] Keras Model Evaluation and Prediction https://www.tutorialspoint.com/keras/keras_model_evaluation_and_prediction.htm

7. Acknowledgement

We, Perumalla Thushara Meher Siva Mani, Vaibhav Nagar and Avval Kaur, the students of BTech in Data Science and Engineering - 2020 batch (third year second semester) would like to thank all of our teachers and Manipal University Jaipur for providing us this platform. We would like to thank our mentor Dr.Avani Sharma who helped us to clear any doubt what so ever we had to encounter while making this project.

This is a wonderful opportunity for us students to show our creativity and to get ourself involve in the work that we love and desire to do. This has enable us to learn and have fun at same time. We will try our best to perform as per the expectation put upon us by our parents, family members, teachers, Manipal University Jaipur and the whole nation.

Lastly, I would like to thank my Family for the tremendous amount of support they offered me while making this Project. I also had the invaluable support of my friends and colleagues which helped me in completing this Project.