

**A Project Report
on
Detection of Age - related Macular Degeneration Using
Deep Learning**

**Submitted in partial fulfillment of the requirements for the award of the degree
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING**

by

16WH1A0597

Ms. P. SANDHYA RANI

16WH1A05B4

Ms. V. S. N DEEPIKA

17WH5A0504

Ms. K. SARIKA

17WH5A0505

Ms. U. SHIVARANI

under the esteemed guidance of

**Ms. B. Nagaveni
Assistant Professor**



**Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090**

May, 2020

DECLARATION

We hereby declare that the work presented in this project entitled “**Detection of Age-related Macular Degeneration Using Deep Learning**” submitted towards completion of Project Work in IV year of B.Tech., CSE at ‘BVRIT HYDERABAD College of Engineering For Women’, Hyderabad is an authentic record of our original work carried out under the guidance of **Ms. B. Nagaveni**, Assistant Professor, Department of CSE.

Sign. with date:

Ms. P. SANDHYA RANI

(16WH1A0597)

Sign. with date:

Ms. V. S. N. DEEPIKA

(16WH1A05B4)

Sign. with date:

Ms. K. SARIKA

(17WH5A0504)

Sign. with date:

Ms. U. SHIVARANI

(17WH5A0505)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering



Certificate

This is to certify that the Project Work report on “**Detection of Age- related Macular Degeneration Using Deep Learning**” is a bonafide work carried out by **Ms. P. Sandhya Rani (16WH1A0597), Ms. V.S.N Deepika(16WH1A05B4), Ms.K.Sarika (17WH5A0504 Ms. U. Shivarani (17WH5A0505)** in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. Ch. Srinivasulu
Professor
Department of CSE

Guide
Ms.B. Nagaveni
Assistant Professor
Department of CSE

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K.V.N Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. Ch. Srinivasulu, HOD & Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms. B. Nagaveni Assistant Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of the **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Sign. with date:

Ms. P. Sandhya Rani
(16WH1A0597)

Sign. with date:

Ms. V. S. N. Deepika
(16WH1A05B4)

Sign. with date:

Ms. K. Sarika
(17WH5A0504)

Sign. with date:

Ms. U. Shivarani
(17WH5A0505)

Contents

S.No.	Topic	Page No.
	Abstract	i
	List of Figures	ii
1.	Introduction	1
	1.1 Objective	3
	1.2 Motivation	3
	1.3 Methodology	3
	1.3.1 Dataset	3
	1.3.2 Deep Learning and Neural Networks	4
2.	Theoretical Analysis of the proposed project	15
	2.1 Software Requirements	15
	2.2 Technologies Description	15
	2.2.1 Python	15
	2.2.2 TensorFlow	17
	2.2.3 Keras	17
	2.2.4 Colab	18
	2.2.5 Python Packages Used	18
3.	Design	20
	3.1 Introduction	20
	3.2 Concepts and Model Used	21
	3.2.1 Transfer Learning	
	3.3 Architecture	22
4.	Implementation	24
	4.1 Accessing the Data	24
	4.2 Preprocessing	26
	4.3 Balancing the Dataset	26
	4.4 Converting the images to generators	28

5.	Results	34
	5.1 Parameters influencing performance	34
	5.1.1 Loss functions	34
	5.1.2 Optimizers	34
	5.1.3 Dropout	35
	5.2 Dataset imbalance and Hyperparameters tuning	36
	5.3 Visualizing Outputs in TensorBoard	38
6.	Conclusion	41
7.	References	42

ABSTRACT

Age – related macular degeneration is a disease that leads to loss of vision in the central field of the eye. It is known to affect nearly a million people in India alone every year. This condition occurs in 2 stages, although not always guaranteed to be consecutive: dry AMD and wet AMD. The presence of dry AMD generally isn't very noticeable by the patient as their vision is not really affected until the wet AMD stage. However upon reaching the wet AMD stage, the loss of vision is irreversible.

The usage of AI in fields like medicine is already proving to be revolutionary. Even though in the current state of things, the suggestions given by AI cannot totally replace a doctor's, such systems nonetheless could reduce the overall work that has to be done by the doctor. Especially in rural areas, where people may not have access to ophthalmologists, an early stage screening could prove to potentially prevent blindness. An OCT scan gives a cross – sectional view of the various layers in the eye and their thicknesses. Such a scan could prove to be useful for the identification of the stage of disease in the eye.

This project aims to produce a deep learning model to act as a screening test. The model would take an OCT scan as the input and predict the stage of the disease.

LIST OF FIGURES

Fig No	Figure Description	Page. No
1	The anatomy of a human eye	1
2	Vision of a person with AMD	2
3	Sample OCT scans from dataset	4
4	Relation amongst AI, ML and Deep Learning	5
5	A Neural Network	6
6	What happens at one neuron	12
7	A typical CNN Architecture	14
8	Architecture	22
9	Class distribution of original dataset	26
10	Class distribution of the undersampled dataset	27
11	Undersampling and oversampling	27
12	Directory Structure of Dataset	29
13	Results	36
14	TensorBoard Interface	39
15	Epoch Accuracy Plot in TensorBoard	39
16	Epoch Loss Plot in TensorBoard	40

1. INTRODUCTION

The macula is the central part of the retina. It is responsible for the central, high-resolution, color vision that is possible in good light. As the body ages and possibly due to genetics, it could wear down. Age – related macular degeneration is a disease that leads to loss of vision in the central field of the eye. It's a common disease with more than a million cases per year in India alone.

It occurs in 2 stages: dry AMD, wet AMD. These are however, not guaranteed to be consecutive which makes the disease all the more difficult to detect and diagnose.

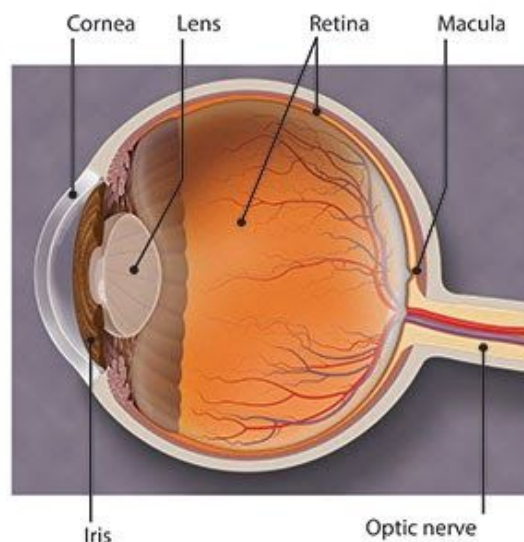


Fig 1: The anatomy of a human eye

The presence of yellowish fatty deposits on the macula, technically known as drusen, could lead to dry AMD. A few small drusen, however, do not lead to any significant changes in vision. As the condition gets worse, the light-sensitive cells in the macula get thinner and eventually die.

The wet AMD stage, which is much more serious, is indicated by the presence of abnormal leaky blood cells under the retina. These blood vessels leak blood and fluid into the retina causing the scarring of the macula. Vision becomes distorted such that straight lines look wavy.



Fig 2: Vision of a person with AMD

1.1 Objective

The progression of AMD to the wet stage makes the damage permanent. So, the earlier AMD is detected the less chances of permanent damage to the retina. Early detection could further potentially prevent blindness. The objective, therefore, of this project is to act as a screening test for the detection of AMD.

1.2 Motivation

In rural areas, where people may not have access to ophthalmologists, an early stage screening could prove to be vision saving. People of a certain age could have their retinal scans input to an algorithm. If a positive result is output they could then understand that there is some sort of risk associated with the condition of their eyes and seek further medical help.

1.3 Methodology

OCT scans (Optical Coherence Tomography) gives a cross – sectional view of the various layers in the eye and their thicknesses. Such a scan could prove to be useful for the identification of the stage of disease in the eye.

1.3.1 Dataset

The dataset was obtained from the ‘Retinal OCT Images’ dataset on Kaggle. It consisted of around 72,000 OCT scans for training and around 700 scans for testing. The scans were categorised into four types: Drusen, CNV, DME and Normal. The ‘DME’ category of images were removed in concern with the irrelevance of the images in this particular use case. Sample scans from the 3 categories are shown below:

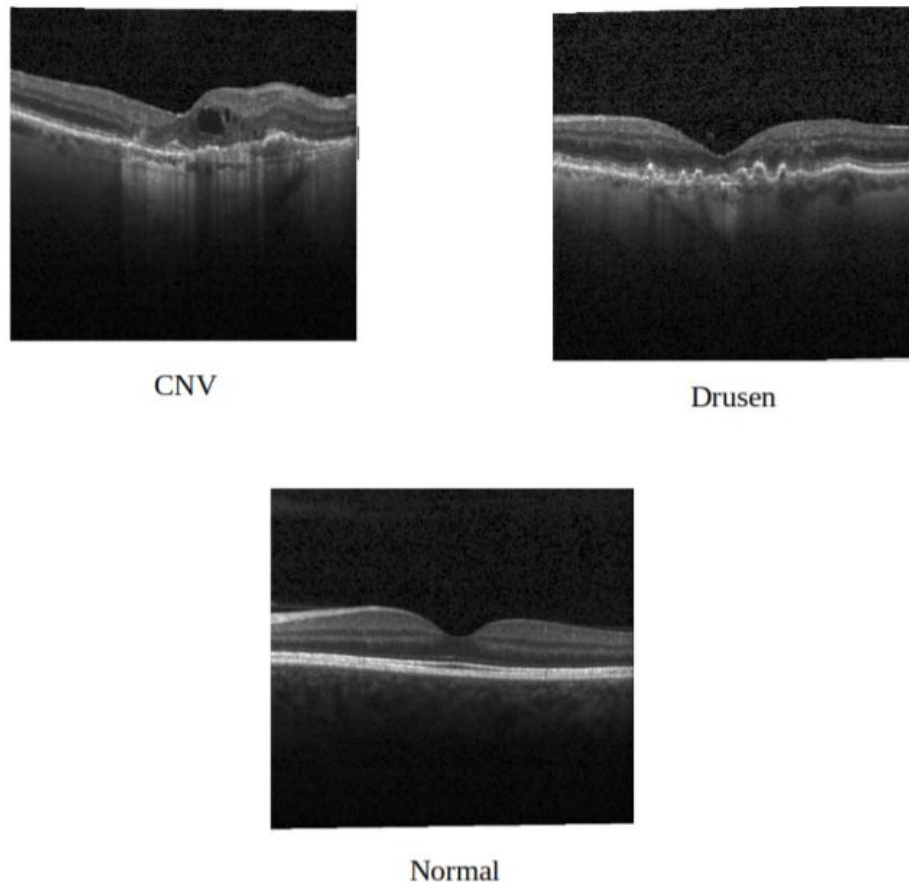


Fig 3: Sample OCT Scans from Dataset

1.3.2 Deep Learning and Neural Networks

Deep Learning:

Artificial intelligence is the simulation of human intelligence in machines. Several sources define the study of ‘intelligent agents’ that perceive their environment and take actions that maximize their chance of successfully achieving their goals. Machine learning is a subset of AI which gives systems the ability to automatically learn and improve from experience.

Deep learning is a subset of machine learning in artificial intelligence. It uses a hierarchical level of artificial neural networks to carry out the process of machine learning. The relationship between the technologies is shown below:

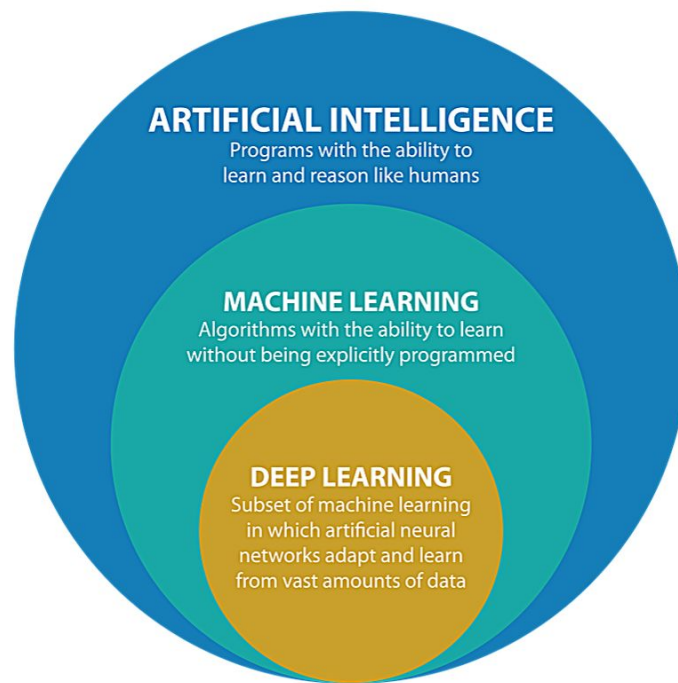


Fig 4: Relation amongst AI, ML and Deep Learning

As the amount of data increases, deep learning algorithms can prove to perform exponentially better than traditional machine learning algorithms.

Neural Networks:

Neural networks are modeled loosely on the human brain. They consist of thousands of simple processing nodes that are densely interconnected. They are organised into layers and typically have 2 phases: forward propagation and backward propagation. Each of the incoming connections to a node will have a weight and bias associated with it.

Deep neural networks have many layers in between the input and output layers, hence the name 'deep'. These are the networks that are used for deep learning. A typical neural networks looks like the image given below:

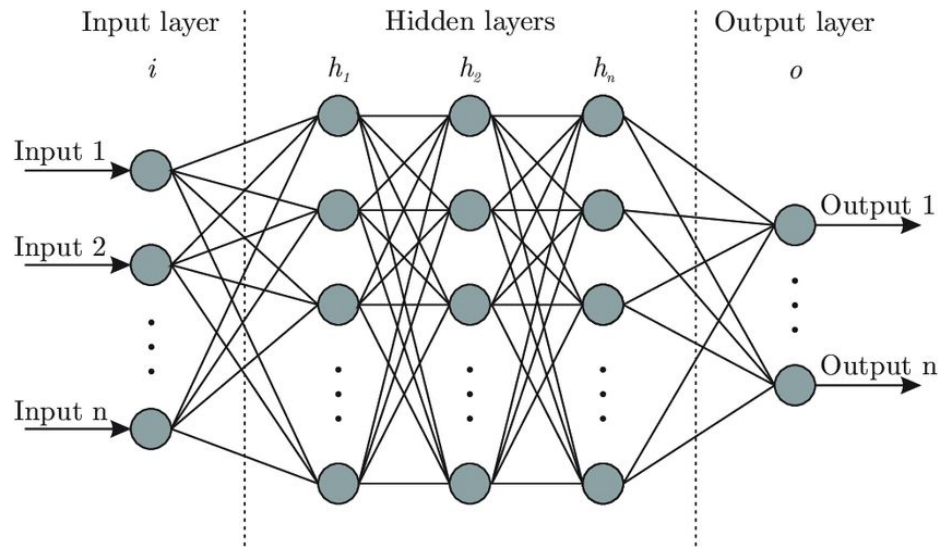




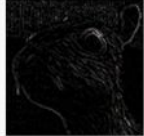


Fig 5: A Neural Network

Convolutional Neural Networks

Using neural networks directly for image data often leads to too many parameters even for very small images. For this reason, convolutional neural networks are used for computer vision tasks. Firstly, the layers are organised in 3 dimensions: width, height and depth. Convnets also use a sequence of operations or filters on the images. The architecture of a convnet was inspired by the human visual cortex. The following are the major building blocks of convnets:

1. **Convolution operation:** The purpose of a convolutional layer is to extract features from the input image. It learns the image features by using small squares of input data. The output image would change depending on the values of the filter matrix. An example of the operation followed by the outputs obtained by the application of different kinds of filters to the same image are shown below:

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

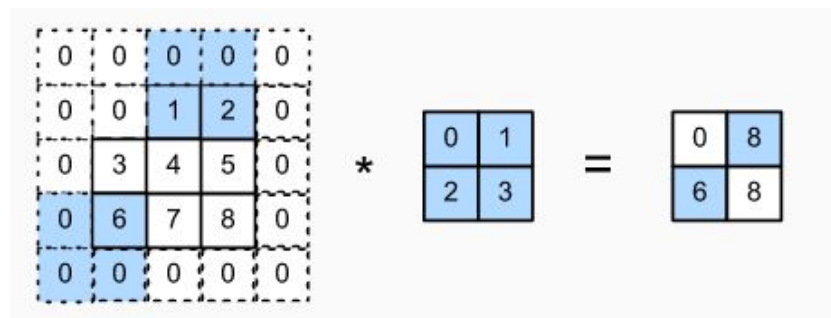
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

- 2. Padding:** Using an $(f \times f)$ filter on an $(n \times n)$ image results in an $(n-f+1 \times n-f+1)$ output. Now, applying such an operation would keep on shrinking the image, making it too small after a few applications of the operation. A layer of 0's is added to the image so that the output is of the same dimensions as the input. Essentially, adding padding to an image processed by

a convnet allows for more accurate analysis of images. A padded image is shown below.

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

- 3. Striding:** When applying the convolution operation, the filter is usually stepped over by one step. However, this does not have to be the case every time. Filters can be stepped over by an arbitrary amount. A stride of two is shown below.

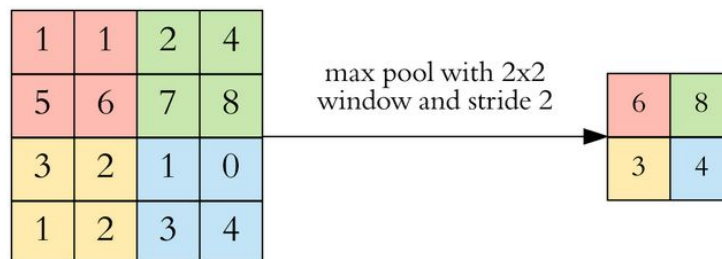


So, taking an image of dimensions $(n \times n)$, a filter of $(f \times f)$ and applying a padding 'p' with stride 's' results in an output of dimensions:

$$((n+2p-f)/s + 1 \times (n+2p-f)/s + 1)$$

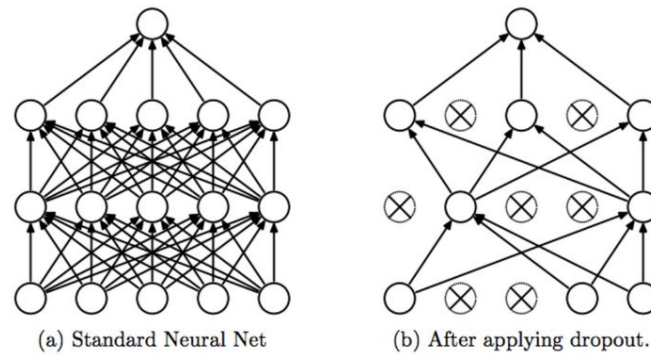
- 4. Pooling:** Pooling layers are often used in Convnets to reduce the size of the representation. They also help make the features detected more robust. They perform what is technically known as downsampling. A pooling layer generally consists of 2 hyperparameters: filter size and stride. There are three types of pooling: max, sum and average, although max pooling is more commonly used.

At a high level, the process that takes place is this. The filter gets placed at each $(f \times f)$ position in the $(n \times n)$ matrix. In each submatrix the $\max(\text{average/ sum})$ is computed and replaced. This process continues for each sub matrix.

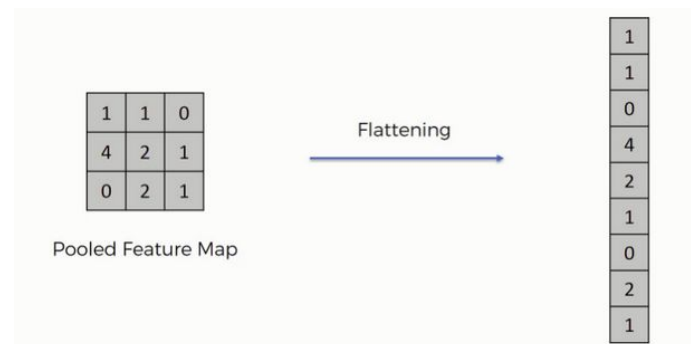


The main purpose of a pooling layer in the neural network is to progressively reduce the spatial size of the representation. This is done in order to reduce the number of parameters and the amount of computation in the network.

- 5. Dropout:** Dropout is a regularization technique. The primary purpose of a dropout is to prevent the model from overfitting. In other words, they help increase the generalizability of the model. It works by randomly ignoring or setting the outgoing units of several hidden layer neurons to zero. Dropout operations can be performed on only some or all of the hidden layers in the neural network. The application of the operation often depends on parameters like the nature of the model and the data itself.



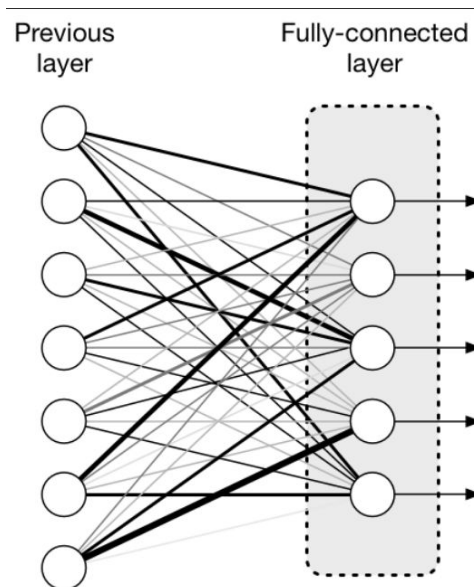
- 6. Flatten:** In flattening, as the name implies, the current matrix is made into a 1 - dimensional column matrix. In other words, a single feature long vector is created. The purpose of this step is to enable the matrix to be input to a fully - connected layer.



- 7. Fully Connected Layer:** These layers, also known as dense layers, often come at the end of a sequence of layers in a Convnet. In a dense layer, all the neurons in the previous layer are connected to their next layer. These fully connected layers heading to the next layer for softmax, in a multi-class case, or sigmoid, in a binary-class case. Finally, the last layers take the results of all the previous layers and use them to classify the image into one of the possible labels.

Now, in a neural network, as the layers go deeper and deeper, the

features that the model deals with become more and more complex. And there have been architectures developed to deal with the problems that arise in such cases.



Activation Functions:

Activation functions are mathematical equations that determine the output of each neuron. At a high level, the activation function determines whether a particular neuron gets fired or not. Activation functions also help normalize the output of each neuron to a range in between 1 and 0 or in between -1 and 1. Another aspect of activation functions is that they must be computationally efficient. This is because they are computed over thousands or millions of neurons for each data sample.

Activation functions are mathematical “gates” in between the input feeding the current neuron and its output going to the next layer. They can be as simple as a step function or it can be a transformation from the input to the output. Now these functions can be either linear or nonlinear. Nonlinear activation functions are

generally preferred to help the network learn more complex data.

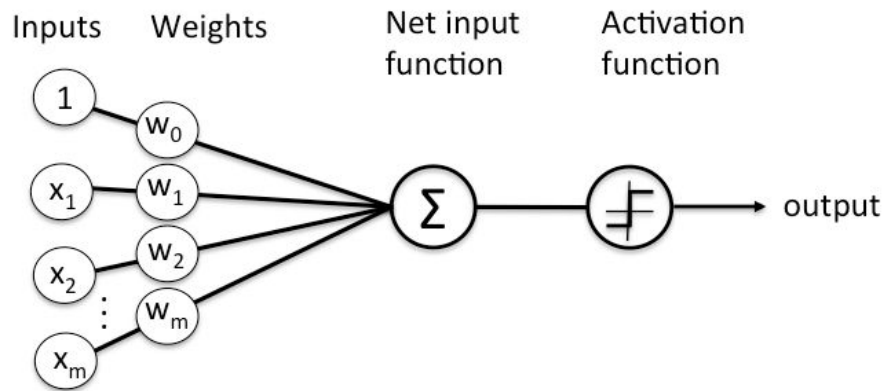
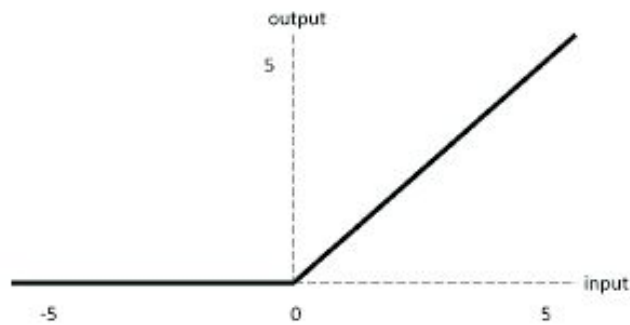


Fig 6: What happens at one neuron

Linear activation functions cannot be used for back propagation. The derivative of a linear function is a constant and hence possesses no relation to the input. Hence nonlinear activation functions are used in modern day neural networks.

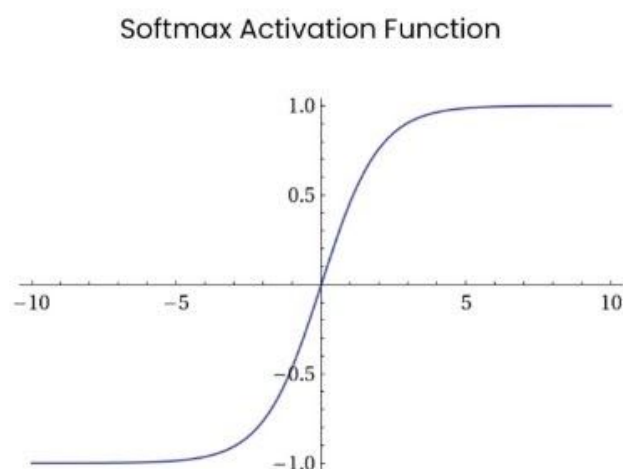
There are many types of nonlinear activation functions like: sigmoid, tanh, softmax, etc. In this project, the ReLU activation function along with softmax in the last layer were used.

1. **ReLU Activation Function:** ReLU stands for Rectified Linear Unit. It is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. There are several variations to this activation function like leaky ReLU and parametric ReLU. The basic ReLU function when plotted looks like:



- 2. Softmax Activation Function:** The softmax function performs a sort of logistic regression on the input. It essentially turns the numbers into probabilities that sum to one. The output values obviously range from $[0, 1]$ since we are dealing with probabilities. Mathematically, it is the exponent of the individual input divided by a sum of the exponents of all the inputs.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



Typical CNN Architecture:

Based on the building blocks of convolutional neural networks stated above, a typical convnet has an architecture as shown below. The convolution filters are applied most regularly. This is because the convolution operator is the one that performs the actual feature extraction. It is the core of the very functionality of a convnet. Typically, pooling (often maxpool) layers are introduced after a couple of convolution operations. This, as stated above is to prevent overfitting. After a sufficient number of these (convolution + pooling) layers are applied, a flatten layer is followed. This again reduces the three dimensional network so far into two dimensions so that it can be handled by a regular neural network from here onwards.

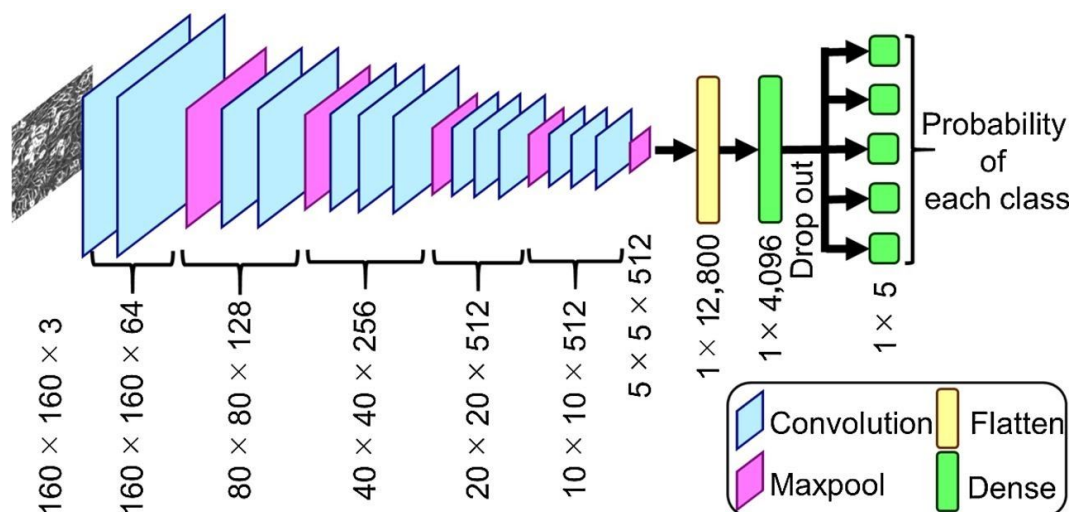


Fig 7: A typical CNN Architecture

Finally, the flatten layer is connected to one or more fully connected layers. It is here that the actual classification decision is taken. The probabilities for each of the possible classes are output. The class with the maximum probability is given as the final output.

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Software Requirements

Programming Language : Python

Dataset : Retinal OCT Images Dataset from Kaggle

Packages : TensorFlow, NumPy, Keras

Visualization : Matplotlib, Tensorboard

Environment : Google Colab

2.2 Technologies Description

2.2.1 Python:

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python code is understandable by humans, which makes it preferred to build models for machine learning. It's an intuitive language and it's many frameworks, libraries, and extensions that simplify the implementation of many different functionalities. It's rich technology stack offers packages like keras, tensorflow, scikit-learn, numpy, pandas and many more.

Python is the most popular language for machine learning and AI. A few reasons why are given below:

1. A great library ecosystem: A rich set of libraries is one of the main reasons Python is the most popular programming language used for AI. Machine learning requires continuous data processing, and Python's libraries let you access, handle and transform data.
2. Easy to pick up: Python programming language resembles the everyday English language, and that makes the process of learning easier. The low entry barrier allows more data scientists to quickly pick up Python and start using it for AI development without wasting too much effort into learning the language.
3. Flexibility: The flexible style of Python allows developers to choose the programming styles which they are fully comfortable with or even combine these styles to solve different types of problems in the most efficient way. Some programming styles are: imperative style, functional style, object-oriented style and the procedural style. The flexibility factor decreases the possibility of errors as well.
4. Platform independence: Python is very versatile. Python for machine learning development can run on any platform including Windows, MacOS, Linux, Unix and several others.
5. Readability: Python is very easy to read. There is no confusion, errors or conflicting paradigms. This helps for the efficient exchange of algorithms, ideas, and tools between AI and ML professionals.
6. Rich set of tools for Visualization: In artificial intelligence, deep learning, and machine learning, it is vital for developers to be able to represent data in a human-readable format. Python offers good options for this.

2.2.2 TensorFlow:

TensorFlow was developed by the Google Brain team initially for internal Google use. A tensor is mathematically an n - dimensional vector. Tensorflow is an open source artificial intelligence library, using data flow graphs to build models. It allows developers to create large-scale neural networks with many layers. TensorFlow offers multiple levels of abstraction. It also offers several AI accelerators like GPUs and TPUs (tensor processing units) which are application specific integrated circuits designed specifically for running machine learning models.

Tensorboard:

Tensorboard is tensorflow's visualization toolkit. The ability to visualize what is happening in the code is very convenient. Tensorboard offers a suite of web applications that help in inspecting and understanding the TensorFlow runs and graphs.

2.2.3 Keras:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It was developed with a focus on enabling fast experimentation. It allows for easy and fast prototyping. The core data structure of Keras is a model. Models are ways to organize the neural network layers. The model can be built by stacking layers together using the `add()` function.

After the model is built, it is compiled and finally fit. The fitting step can be performed in batches as well. Layers and models can be imported as packages in the code.

```
from tensorflow.keras.models import Sequential, Model  
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D, GlobalAveragePooling2D
```

2.2.4 Colab:

Colaboratory, or "Colab" for short, was chosen as the environment to write the code in. Its main advantages include: zero configuration requirement, free access to hardware accelerators like GPUs and TPUs and easy sharing. Colab notebooks get stored in the user's Google Drive account. Hence can be easily accessed from any system that has access to the internet. In addition to this, Colab notebooks allow the combination of executable code and rich text in a single document, along with images, HTML, LaTeX and more.

Apart from the advantages stated above, the Google Colab environment was preferred over a Jupyter notebook for the main reason of the sheer size of the dataset being used. A jupyter notebook would eventually get the dataset locally. Using the Colab environment would enable the usage of Google Cloud for storage instead.

Colab also offers great support for Tensorflow like rendering the Tensorboard visualization in a code block of the current notebook itself rather than a new window.

2.2.5 Python Packages Used:

1. **Matplotlib:** Matplotlib is a visualization library in Python for 2D plots of arrays. One of the main benefits of visualization is that it allows us to perceive information about the dataset. On plotting the class distribution of the dataset, it came to be known that there was a class imbalance and appropriate steps

were taken thereon. The matplotlib library can be imported with the following line of code.

```
import matplotlib.pyplot as plt
```

2. **OS:** Python's OS module provides functions for interacting with the operating system. For the purposes of this project, it was used to access the directory structure of the downloaded dataset to manipulate the scans. There are several predefined functions in the library like os.name() and os.getcwd(). The os.listdir() function was used to get an object of the directory structure.

```
import os  
files = os.listdir('/content/octScans/OCT2017 /train/CNV')
```

3. **NumPy:** NumPy stands for Numerical Python. It is the python library which adds support for large, multi-dimensional arrays and matrices. It also provides at our disposal a collection of high-level mathematical functions to operate on these large arrays. They are preferred over lists as they follow the locality of reference principle. In other words, they are stored at one continuous place in memory unlike lists. This allows processes to access and manipulate them very efficiently. It is imported as follows:

```
import numpy as np
```

3. DESIGN

3.1 Introduction

Design is one of the most critical stages of an engineering process. The first step, however, is to gain an empathic understanding of the problem you are trying to solve and understand the users well. For this we have to understand the real world problems from users. Only then can something meaningful be created. At a high level, the designer's goal is to produce a model or representation of an entity that will later be built.

Ideation would be the stage of the design process in which it is aimed to generate radical design alternatives. An effort is made to come up with as many solutions to the problem that was thought of in the previous step. Then a suitable dataset is searched for or synthesized based on the problem that is being solved.

Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. It is here that a model is chosen for the specific problem. It is the only way in which the customer's views can be accurately translated into a finished software product or system.

Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage. During design, progressive refinement of data structure, program structure, and procedural details are developed, reviewed and documented.

3.2 Concepts and Model Used:

3.2.1 Transfer Learning:

Transfer learning refers to storing knowledge gained while solving one problem and applying it to a different problem. A simple analogy would be that the knowledge gained in learning to recognize cars can be used in recognizing trucks. Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error.

The Vanishing Gradients Problem:

In deep neural networks as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. Certain activation functions, like the sigmoid function, scale down a large input space into a small input space between 0 and 1. So a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.

For a shallow network, this is not a significant problem. But when more layers are used, it can cause the gradient to be too small for training to work effectively.

Resnet50:

Resnet (residual network) is a classic neural network. This model was the winner of the ImageNet challenge in 2015. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients.

Residual networks first introduced the concept of skip connections. As the name implies they provide residual connections straight to earlier layers. More specifically, the activation unit from one layer could be fed directly to a deeper layer

of the network. This is a skip connection. These skip connections enable the building of very deep neural networks. The ‘50’ indicates the presence of 50 layers. The resnet50 model consists of a series of convolutional layers + skip connections, then average pooling and then an output fully connected (dense) layer.

3.3 Architecture

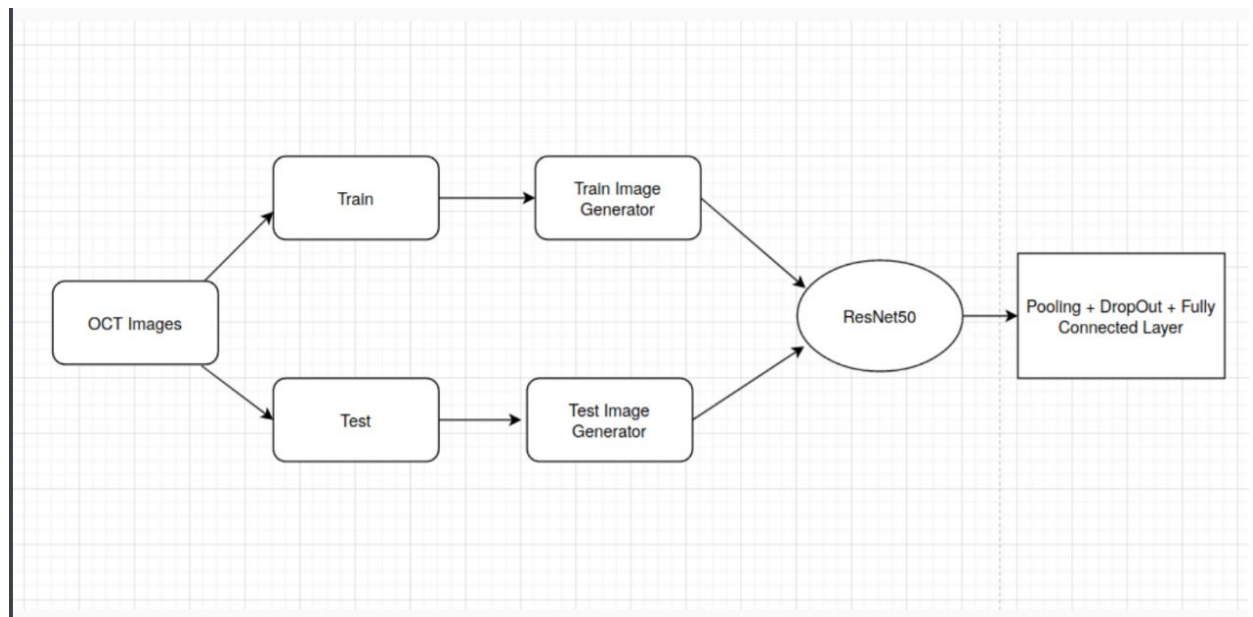


Fig 8: Architecture

In the above flowchart, the OCT scans are first preprocessed. Preprocessing involves several steps like rescaling and resizing. Both the train and test datasets are then converted to image generators. Image generators can be used both for the purposes of dataset augmentation and for training the model in mini batches rather than on the whole dataset at once. Here, they are used for the purpose of mini batches.

A resnet50 model was chosen as the base model. To an instance of this model, the last fully connected layer is removed. Additional pooling, dropout and dense layers are added. In the last fully connected layer, 3 neurons are present since there are 3 categories in this specific classification problem.

4. IMPLEMENTATION

4.1 Accessing the Data:

With the dataset being quite large, downloading such a dataset locally was out of the question. Hence, Kaggle's API was used to download a compressed version of the scans directly into our coding environment, Google Colaboratory. The following steps were followed:

Installation:

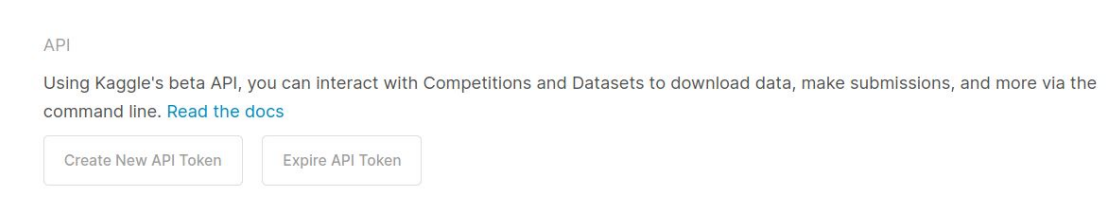
Kaggle's public API was used in Colab via the command-line tool implemented in Python. This was installed using the package manager pip by running the following command:



```
pip install kaggle
```

Authentication:

In order to use Kaggle's API, authentication must first be performed using an API token. To create a new token, the "Create New API Token" button is clicked from a Kaggle account.



Then suitable credentials are given:


```
os.environ['KAGGLE_USERNAME'] = "iamdeepika" # username from the json file
os.environ['KAGGLE_KEY'] = "c549ee63c4b891280109f6f52ad88c72" # key from the json file
```

Next the compressed dataset is downloaded. A directory was created into which the whole dataset gets unzipped.

```
!kaggle datasets download -d paultimothymooney/kermany2018
```

```
Downloading kermany2018.zip to /content
100% 10.8G/10.8G [04:23<00:00, 97.1MB/s]
100% 10.8G/10.8G [04:23<00:00, 44.2MB/s]
```

```
[7] !mkdir octScans
```

```
!unzip /content/kermany2018.zip -d octScans
```

```
Streaming output truncated to the last 5000 lines.
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8055145-1.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8055145-2.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8055145-3.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8055590-1.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8055590-2.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8055590-3.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-1.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-2.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-3.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-4.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-5.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-6.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-7.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8056259-8.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8057481-1.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8057481-2.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8057481-3.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8057481-4.jpeg
inflating: octScans/oct2017/___MACOSX/OCT2017 /train/NORMAL/._NORMAL-8058202-1.jpeg
```

Then the paths for the training and test data were set:

```
train_data_path = '/content/octScans/OCT2017 /train'
test_data_path = '/content/octScans/OCT2017 /test'
```

4.2 Preprocessing:

1. **Rescaling:** For the purpose of preprocessing, the scans were rescaled. Rescaling is a normalization technique. Simply stated, normalization is the process of transforming a dataset so that its minimum is 0 and its maximum 1. In order to do this the following operation is performed:

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# rescale converts range from integers in [1-255] to floats in the [0,1]
image_generator = ImageDataGenerator(rescale = 1./255)
```

2. **Resizing:** Since the dataset could possibly consist of images that are of varying sizes, a 'target_size' parameter is specified in the ImageDataGenerator. Here, the target_size specified is (192, 192).

4.3 Balancing the Dataset:

The initial class distribution of the dataset is given below.

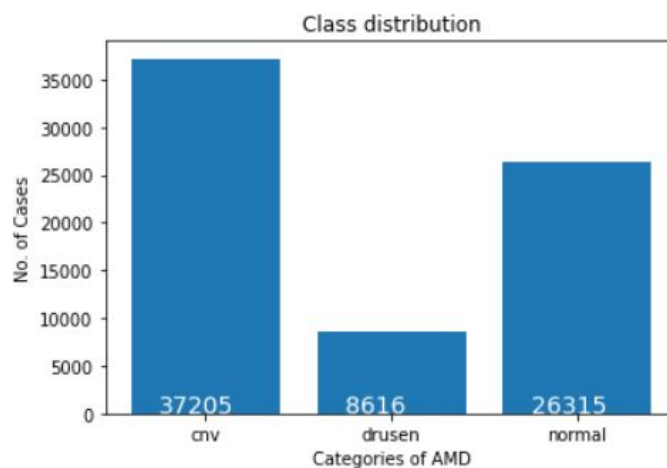


Fig 9: Class distribution of original dataset

To this skewed dataset, the following approaches were applied in an attempt to balance the distribution.

1. Undersampling:

A random sample of about 8500 images were removed from the CNV category. The class distribution that this resulted in is shown below. Since the dataset was already pretty large, it's negative impact on the effectiveness of the model was relatively less.

```
from random import sample

files = os.listdir('/content/octScans/OCT2017 /train/CNV')
for file in sample(files,8500):
    os.remove('/content/octScans/OCT2017 /train/CNV/' + file)
```

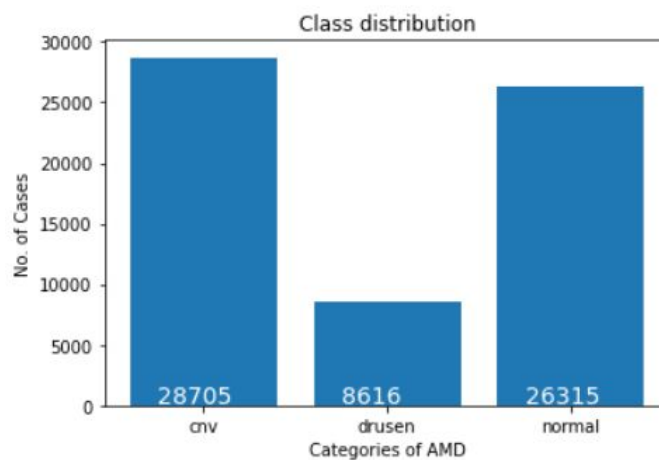


Fig 10: Class distribution of the undersampled dataset

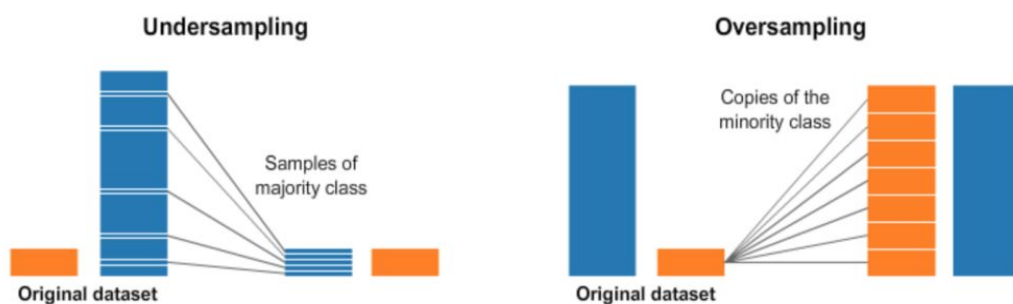


Fig 11: Undersampling and oversampling

2. Adjusting the weights of the classes:

Although reducing a few images from the dominating class was somewhat effective, it was at the cost of the loss of some valuable data.

‘class_weight’ is a parameter which is basically a dictionary mapping of the class indices(integers) to weights(floats) used during the training phase. This parameter looks at the distribution of labels and produces weights to equally penalize under or over-represented classes in the training set. An array is outputted which has to be converted to a dictionary.

```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight(
    'balanced',
    np.unique(train_data_gen.classes),
    train_data_gen.classes)

class_weights_mapping = dict(enumerate(class_weights))
```

This parameter is set during the model fitting stage.

4.4 Converting the images to generators:

The ImageDataGenerator class from Keras is used to create generators. Rather than performing the operations on the entire image dataset in memory, this API is designed to be iterated by the deep learning model fitting process.

This class can also be used for dataset augmentation. It uses what is known as ‘in-place’ or ‘on-the-fly’ data augmentation, which essentially means that the process is done at the training phase, creating ‘just-in-time’ data. An input batch of training images is provided to the ImageDataGenerator. It transforms each image in the batch

by a series of random translations, rotations, etc. Then the randomly transformed batch is then returned to the calling function. The end goal of data augmentation is to increase the generalizability of the model. To accomplish this goal the training data is replaced with this randomly transformed, augmented data.

The ‘flow_from_directory’ method is used as the images are neatly sorted into categories, each having its own folder. The directories of the training and test scans are given as parameters. The directory structure is shown below:

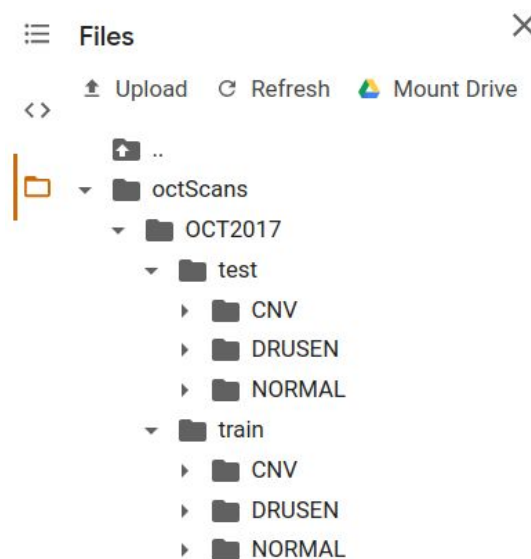


Fig 12: Directory Structure of Dataset

```
train_data_gen = image_generator.flow_from_directory(directory = train_data_path,
                                                    batch_size = 32,
                                                    shuffle = True,
                                                    target_size = (192, 192),
                                                    classes = list(CLASS_NAMES))
```

Found 72136 images belonging to 3 classes.

```
test_data_gen = image_generator.flow_from_directory(directory = test_data_path,
                                                    shuffle = True,
                                                    target_size = (192, 192),
                                                    classes = list(CLASS_NAMES))
```

Found 726 images belonging to 3 classes.

Now, first an instance of resnet50 from Keras is created.

```
model = ResNet50(weights= None, include_top=False, input_shape= (192, 192, 3))
```

resnet50 is a pre-trained model, trained on the ImageNet database on a million images of 1000 categories. Setting the 'weights' parameter to None indicates a random weights initialization. Since transfer learning is being used, the top most fully connected layer from the original architecture is not included.

Now, to this instance, 3 additional layers are added as already indicated in the architecture section. A GlobalAveragePooling2D layer, Dropout layer and a fully connected layer. The dense layer would have 3 neurons as there are 3 categories in the dataset.

```
x = model.output  
x = GlobalAveragePooling2D()(x)  
x = Dropout(0.7)(x)  
predictions = Dense(3, activation= 'softmax')(x)  
model = Model(inputs = model.input, outputs = predictions)
```

After the model is built, it is compiled. This step specifies the loss function, the optimizer and the metrics. Here 'categorical_crossentropy' is used as the loss function. The loss function estimates the loss of the model so that the weights can be updated to reduce the loss on the next evaluation. Cross-entropy is the default loss function to use for multi-class classification problems. It calculates a score which indicates the average difference between the actual and predicted probability

distributions for all classes in the problem. So, the ideal cross-entropy score is 0.

Optimizers update the model in response to the output of the loss function. Gradient descent is one of the oldest optimization algorithms. It is used to minimize some function by iteratively moving in the direction of steepest descent. Then it is used to update the parameters. However, one problem with gradient descent is that there is a lot of overhead with respect to the computations. Hence it is not very effective with large datasets. Stochastic gradient descent builds upon this. While selecting data points at each step to calculate the derivatives, SGD randomly picks one data point from the whole data set at each iteration. This helps to reduce the number of computations enormously.

Adam is an adaptive learning rate optimization algorithm. It tries to leverage the power of adaptive learning rates methods to find individual learning rates for each parameter.

```

▶ from tensorflow.keras.optimizers import SGD, Adam
  #sgd = SGD(lr = 0.0001)
  adam = Adam(lr = 0.0001)
  model.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics=['accuracy'])

```


After the compiling step, the data generators are fit to the model. This is performed using the fit function to which the train and test data generators are passed.

```

▶ model.fit(
    train_data_gen,
    steps_per_epoch = len(train_data_gen),
    epochs = 10,
    validation_data = test_data_gen,
    validation_steps = 25,
    class_weight = class_weights_mapping,
    callbacks=[tensorboard_callback])

```


The ‘epoch’ parameter indicates the number of times the learning algorithm passes through the entire dataset. The ‘steps_per_epoch’ parameter indicates the number of batch iterations before an epoch is considered finished. It is typically calculated as `train_data_length // batch_size`. The train data generator’s length indirectly indicates this value.

 `model.summary()`

conv2_block1_3_bn (BatchNormali	(None, 48, 48, 256)	1024	conv2_block1_3_conv[0][0]
conv2_block1_add (Add)	(None, 48, 48, 256)	0	conv2_block1_0_bn[0][0] conv2_block1_3_bn[0][0]
conv2_block1_out (Activation)	(None, 48, 48, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 48, 48, 64)	16448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormali	(None, 48, 48, 64)	256	conv2_block2_1_conv[0][0]
conv2_block2_1_relu (Activation	(None, 48, 48, 64)	0	conv2_block2_1_bn[0][0]
conv2_block2_2_conv (Conv2D)	(None, 48, 48, 64)	36928	conv2_block2_1_relu[0][0]
conv2_block2_2_bn (BatchNormali	(None, 48, 48, 64)	256	conv2_block2_2_conv[0][0]
conv2_block2_2_relu (Activation	(None, 48, 48, 64)	0	conv2_block2_2_bn[0][0]
conv2_block2_3_conv (Conv2D)	(None, 48, 48, 256)	16640	conv2_block2_2_relu[0][0]
conv2_block2_3_bn (BatchNormali	(None, 48, 48, 256)	1024	conv2_block2_3_conv[0][0]
conv2_block2_add (Add)	(None, 48, 48, 256)	0	conv2_block1_out[0][0] conv2_block2_3_bn[0][0]
conv2_block2_out (Activation)	(None, 48, 48, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 48, 48, 64)	16448	conv2_block2_out[0][0]
conv2_block3_1_bn (BatchNormali	(None, 48, 48, 64)	256	conv2_block3_1_conv[0][0]
conv2_block3_1_relu (Activation	(None, 48, 48, 64)	0	conv2_block3_1_bn[0][0]
. . .			

After the 50 layers present in the resnet50’s architecture, the additional layers can also be seen in the model summary as shown in the below image.

conv5_block3_1_bn (BatchNormali	(None, 6, 6, 512)	2048	conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation	(None, 6, 6, 512)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 6, 6, 512)	2359808	conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormali	(None, 6, 6, 512)	2048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation	(None, 6, 6, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 6, 6, 2048)	1050624	conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali	(None, 6, 6, 2048)	8192	conv5_block3_3_conv[0][0]
conv5_block3_add (Add)	(None, 6, 6, 2048)	0	conv5_block2_out[0][0] conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)	(None, 6, 6, 2048)	0	conv5_block3_add[0][0]
global_average_pooling2d (Globa	(None, 2048)	0	conv5_block3_out[0][0]
dropout (Dropout)	(None, 2048)	0	global_average_pooling2d[0][0]
dense (Dense)	(None, 3)	6147	dropout[0][0]
=====			
Total params: 23,593,859			
Trainable params: 23,540,739			
Non-trainable params: 53,120			

5. RESULTS

5.1 Parameters influencing performance

The parameters that were taken into account for potentially influencing the performance of the model are: dataset imbalance, choice of optimizer and loss function and the dropout hyperparameter.

5.1.1 Loss functions:

The loss/error function is used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation.

`'categorical_crossentropy'` is the default loss function used for multi-class classification problems. Mathematically, it uses the concept of maximum likelihood. It calculates a score which is the average difference between the actual and predicted probability distributions for all classes in the problem. The perfect cross-entropy score is 0.

5.1.2 Optimizers:

Optimizers update the weight parameters in order to minimize the loss function. In other words, loss functions act as guides, directing the optimizers. Optimizers also specify what is known as learning rate. It is a hyperparameter which specifies the amount that the weights are updated during training. It's also referred to as the step size.

The learning rate controls how quickly the model is adapted to the problem. A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck. Clearly it is one of the most important hyperparameters of the model.

Stochastic Gradient Descent:

A gradient mathematically means a slope. In gradient descent, we move in the direction of steepest descent. It's a minimization algorithm used to update the parameters of the model. In gradient descent the cost is computed from the entire data set. So using such an algorithm on a very large dataset can prove to be very costly.

In stochastic gradient descent, the weights are computed after each training sample. Since this is an approximation of the true weights, the name 'stochastic'.

Adam Optimizer:

Adam is an adaptive learning rate optimization algorithm. As this suggests, the algorithm uses adaptive learning rates methods to compute individual learning rates for each parameter. Adam is generally seen as a combination of RMSprop and Stochastic Gradient Descent with momentum.

5.1.3 Dropout:

Deep neural networks tend to overfit. So the main purpose of a dropout layer is to combat this overfitting. Dropout has one hyperparameter which specifies the probability at which outputs of the layer are dropped out, or inversely the probability at which the outputs are retained. This detail is generally left to the specific libraries implementation.

For example, a value of 0.6 indicates that there is a 60% chance for a node to get dropped out.

5.2 Dataset imbalance and Hyperparameters tuning

Dataset	Optimizer	Dropout	Epochs	Accuracy
Unbalanced	Adam	0.7	5	83.78
Unbalanced	Adam	0.7	10	70.38
Unbalanced	Adam	0.6	10	84.30
Unbalanced	SGD	0.7	10	36.6
Unbalanced	SGD	0.6	10	46.33
Reduced CNV	Adam	0.7	10	79.62
Reduced CNV	Adam	0.6	10	70.54
Reduced CNV	SGD	0.7	10	43.54
Reduced CNV	SGD	0.6	10	44.68
Weighted classes	Adam	0.7	10	86.55
Weighted classes	Adam	0.6	10	82.41
Weighted classes	SGD	0.7	10	46.33
Weighted classes	SGD	0.6	10	47.47
Pre trained weights	Adam	0.5	10	Overfitting

Fig 13: Results

The accuracies were computed and compared for different cases as summarized in the above table. Hyperparameter tuning was performed for 3 variations of the dataset: the original dataset, undersampled dataset and the weighted classes dataset.

For each variation of the dataset, accuracies were computed for 2 kinds of optimizers namely: SGD (stochastic gradient descent) and Adam, and for 2 kinds of dropout rates namely: 0.6, 0.7.

The epochs along with the output metrics like train, test accuracy and train, test loss values for two sets of the parameters stated above are shown in the pictures given below:

1. Undersampled dataset

Optimizer - SGD

Dropout - 0.7

```
Epoch 1/10
100/100 [=====] - 52s 520ms/step - loss: 2.0829 - accuracy: 0.3572 - val_loss: 1.2278 - val_accuracy: 0.3544
Epoch 2/10
100/100 [=====] - 51s 506ms/step - loss: 1.6117 - accuracy: 0.4416 - val_loss: 4.0881 - val_accuracy: 0.3595
Epoch 3/10
100/100 [=====] - 51s 506ms/step - loss: 1.3674 - accuracy: 0.5731 - val_loss: 13.0867 - val_accuracy: 0.3342
Epoch 4/10
100/100 [=====] - 51s 505ms/step - loss: 1.0722 - accuracy: 0.6419 - val_loss: 3.3611 - val_accuracy: 0.3215
Epoch 5/10
100/100 [=====] - 51s 505ms/step - loss: 0.9308 - accuracy: 0.6956 - val_loss: 3.7287 - val_accuracy: 0.3410
Epoch 6/10
100/100 [=====] - 51s 506ms/step - loss: 0.8449 - accuracy: 0.7194 - val_loss: 0.9087 - val_accuracy: 0.5658
Epoch 7/10
100/100 [=====] - 51s 506ms/step - loss: 0.8899 - accuracy: 0.7063 - val_loss: 0.4414 - val_accuracy: 0.8532
Epoch 8/10
100/100 [=====] - 50s 505ms/step - loss: 0.8215 - accuracy: 0.7272 - val_loss: 0.7566 - val_accuracy: 0.8646
Epoch 9/10
100/100 [=====] - 51s 506ms/step - loss: 0.7566 - accuracy: 0.7503 - val_loss: 0.9099 - val_accuracy: 0.6063
Epoch 10/10
100/100 [=====] - 51s 506ms/step - loss: 0.6771 - accuracy: 0.7803 - val_loss: 2.2114 - val_accuracy: 0.4152
<tensorflow.python.keras.callbacks.History at 0x7fdf1dbd8828>
```

2. Weighted class neural network

Optimizer - Adam

Dropout - 0.7

```
Epoch 1/10
2255/2255 [=====] - 398s 176ms/step - loss: 1.7076 - accuracy: 0.3483 - val_loss: 1.0507 - val_accuracy: 0.4752
Epoch 2/10
2255/2255 [=====] - 375s 166ms/step - loss: 1.5330 - accuracy: 0.3757 - val_loss: 0.9836 - val_accuracy: 0.5744
Epoch 3/10
2255/2255 [=====] - 376s 167ms/step - loss: 1.3923 - accuracy: 0.4251 - val_loss: 0.8728 - val_accuracy: 0.6625
Epoch 4/10
2255/2255 [=====] - 375s 166ms/step - loss: 1.2442 - accuracy: 0.5078 - val_loss: 0.6982 - val_accuracy: 0.7149
Epoch 5/10
2255/2255 [=====] - 373s 165ms/step - loss: 1.0957 - accuracy: 0.5971 - val_loss: 0.5324 - val_accuracy: 0.7837
Epoch 6/10
2255/2255 [=====] - 371s 165ms/step - loss: 0.9370 - accuracy: 0.6738 - val_loss: 0.4961 - val_accuracy: 0.7796
Epoch 7/10
2255/2255 [=====] - 371s 165ms/step - loss: 0.8480 - accuracy: 0.7099 - val_loss: 0.3619 - val_accuracy: 0.8636
Epoch 8/10
2255/2255 [=====] - 371s 165ms/step - loss: 0.7803 - accuracy: 0.7299 - val_loss: 0.3751 - val_accuracy: 0.8430
Epoch 9/10
2255/2255 [=====] - 372s 165ms/step - loss: 0.7278 - accuracy: 0.7476 - val_loss: 0.2889 - val_accuracy: 0.9008
Epoch 10/10
2255/2255 [=====] - 367s 163ms/step - loss: 0.6868 - accuracy: 0.7607 - val_loss: 0.2666 - val_accuracy: 0.9091
<tensorflow.python.keras.callbacks.History at 0x7f292d6f7eb8>
```

5.3 Visualising the outputs in Tensorboard

Tensorboard was used for output and model visualization purposes. It can be observed in the following code snippet that a call back is made. Callback APIs are objects that can perform certain actions at various stages of the model. These callbacks can be used to write TensorBoard logs after every batch of training. This is done to be able to monitor the metrics obtained.

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

The tensorboard interface can be loaded from a code block from within the Colab environment itself. However the extension must first be loaded. This is done by the following command.

```
%load_ext tensorboard
```

Next the logs from previous models are cleared (if any) and then Tensorboard is loaded.

```
!rm -rf ./logs/

[ ] %tensorboard --logdir logs/fit
```

After tensorboard is loaded, an interface like the one shown below is loaded. Tools like histograms, epoch accuracy and loss plots, neural network visualization and many more are offered. These can be accessed using the drop down shown in the picture below:

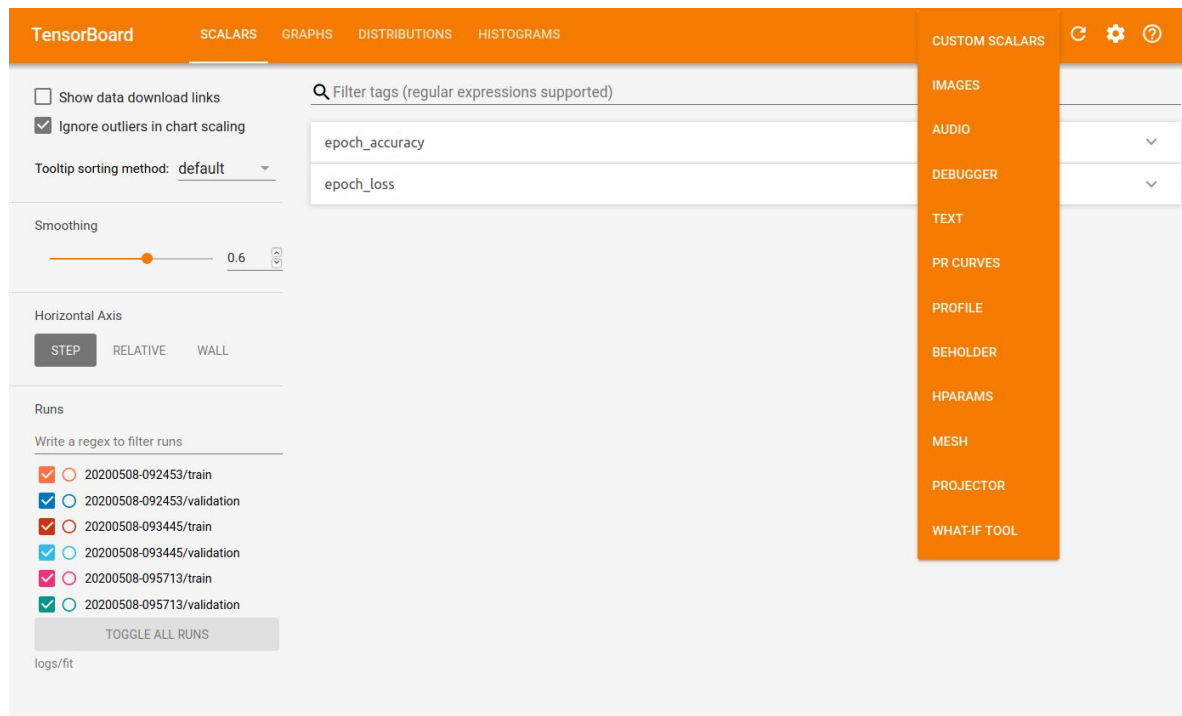


Fig 14: TensorBoard Interface

The epoch accuracy and loss plots for the results tabulated in the previous section are shown below. Each epoch has a unique id associated with it. From the accuracy plot it can be seen that the results are mostly categorised into two regions. Accuracies of

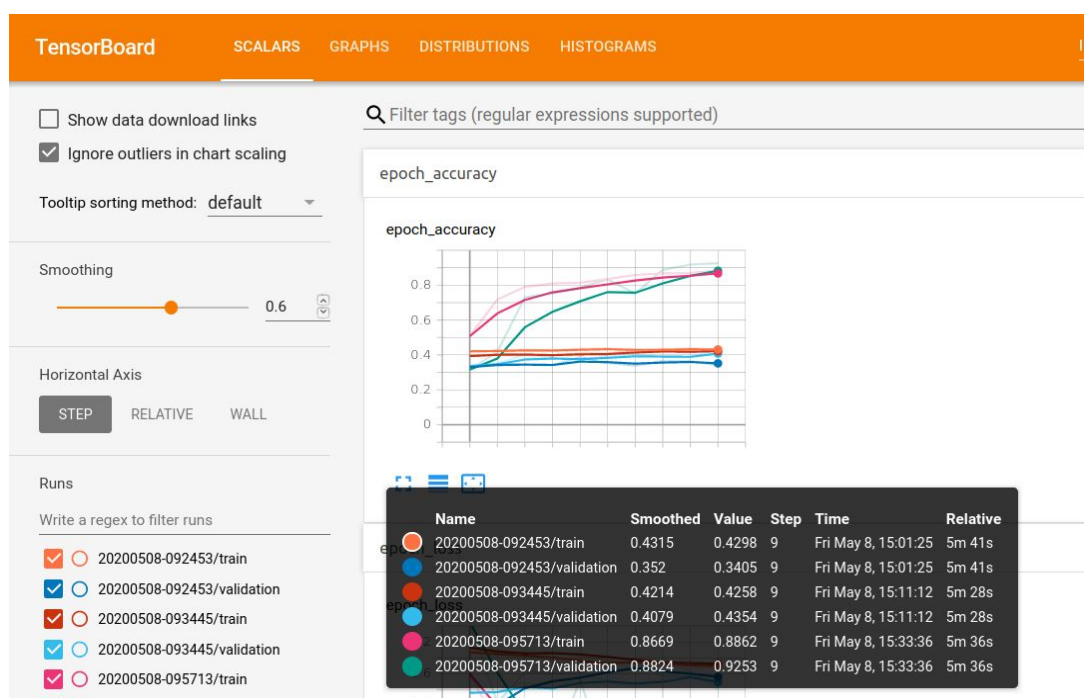


Fig :15 Epoch Accuracy Plot in TensorBoard

around 80% and accuracies of around 30-40%. These are primarily because of the optimizer used. Similarly in the epoch loss plot shown below, the loss values are primarily in the following groups: loss values in between 1.5 - 2 and loss values in between 0 - 0.4. As was the case with the accuracies, the loss values are also significantly influenced by the optimization function used.

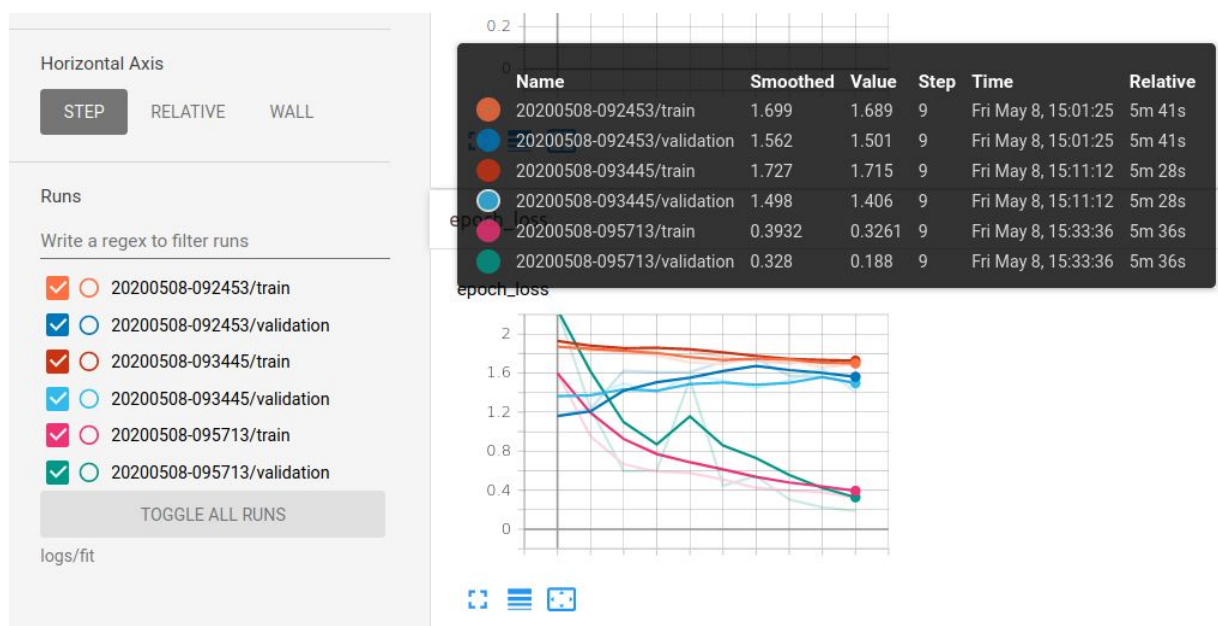


Fig 16: Epoch Loss Plot in TensorBoard

6. CONCLUSION

From the above tabulated results, the following conclusions can be drawn. On the whole, the test cases which used the SGD optimizers offered lower accuracies as compared to the test cases that used the Adam optimizer. Secondly the accuracies got comparatively better for the undersampled dataset when compared to the original dataset. These results go well with the hypothesis of the negative impact that dataset imbalance could have.

The undersampled case clearly loses valuable data. The weighted class neural network on the other hand doesn't lose any of the data but simply adjusts priorities or rather weights of the nodes. This is done depending on their relative population ratio in the original dataset. From the results it can be seen that most cases perform better in the weighted class neural network case when compared to the original and undersampled cases.

Finally, by comparing the results tabulated in the above section, it can be observed that a maximum accuracy of around 86.55% is obtained. This is in the case where a dropout layer with a hyperparameter of 0.7 and the Adam optimizer are used on the weighted class neural network.

7. REFERENCES

- [1] Hwang DK, Hsu CC, Chang KJ, Chao D, Sun CH, Jheng YC, Yarmishyn AA, Wu JC, Tsai CY, Wang ML, Peng CH, Chien KH, Kao CL, Lin TC, Woung LC, Chen SJ, Chiou SH. Artificial intelligence-based decision-making for age-related macular degeneration. *Theranostics* 2019
- [2] Soichiro Kuwayama, Yuji Ayatsuka, Daisuke Yanagisano, Takaki Uta, Hideaki Usui, Aki Kato, Noriaki Takase, Yuichiro Ogura, Tsutomu Yasukawa. Automated Detection of Macular Diseases by Optical Coherence Tomography and Artificial Intelligence Machine Learning of Optical Coherence Tomography Images, *Thno.org*, 2019.
- [3] Jason Brownlee, Image Augmentation for Deep Learning With Keras in Deep Learning, Machine Learning Mastery, September 13, 2019.
- [4] Susan Ruyu Qi: Machine Learning and OCT Images — the Future of Ophthalmology, *Medium*, Dec 8, 2017
- [5] Porto Seguro, Resampling strategies for imbalanced datasets, *Kaggle*, 2018.
- [6] Roy Swartz, Anat Loewenstein, Early detection of age related macular degeneration, *International Journal of Retina and Vitreous*, Article Number 20, December 01 2015.
- [7] Jason Brownlee, Random Oversampling and Undersampling for Imbalanced Classification, *Machine Learning Mastery*, January 15, 2020.
- [8] Jason Brownlee, Transfer Learning in Keras with Computer Vision Models in Deep Learning for Computer Vision, *Machine Learning Mastery*, May 15, 2019.
- [9] Jason Brownlee, How to Choose Loss Functions When Training Deep Learning Neural Networks in Deep Learning Performance, *Machine Learning Mastery*, January 30 2019.
- [10] Thanh Vân Phan, Lama Seoud, Hadi Chakor, Farida Cheriet, Automatic Screening and Grading of Age-Related Macular Degeneration from Texture Analysis of Fundus Images, *Journal of Ophthalmology*, Volume 2016, Article ID 5893601, March 21 2016.