CATARACT DETECTION USING MACHINE LEARNING

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Aryan Agnihotri (191401)
Tanmay Agarwal (191416)

Under the supervision of

Dr. Ekta Gandotra

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan-173234, Himachal Pradesh

CANDIDATE'S DECLARATION

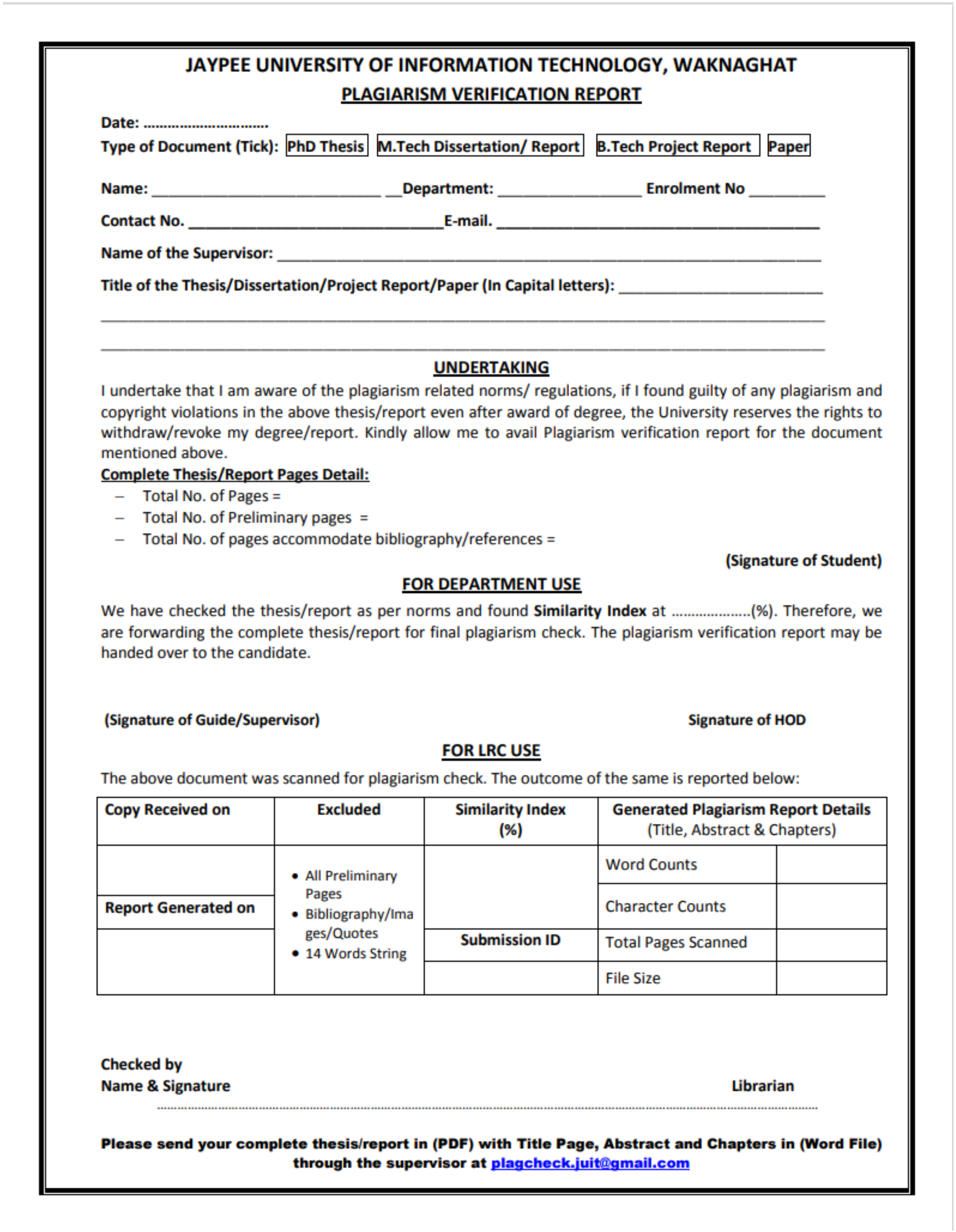
We hereby declare that the work presented in this report entitled **"Cataract Detection using Machine Learning"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of our own work carried out over a period from July 2022 to May 2023 under the supervision of **Dr. Ekta Gandotra, Associate Professor(SG), Dept. CSE&IT**.
We also authenticate that we have carried out the above-mentioned project work under the proficiency stream Machine Learning.
The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Tanmay Agarwal, 191416   Aryan Agnihotri, 191401

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Ekta Gandotra
Associate Professor (SG)
Department of CSE & IT
Dated:

PLAGIARISM CERTIFICATE



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): **PhD Thesis** **M.Tech Dissertation/ Report** **B.Tech Project Report** **Paper**

Name: ___________________________ __**Department:** _________________ **Enrolment No** _________

Contact No. ______________________________**E-mail.** ______________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ________________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

At the onset, we express our heartfelt thanks and gratefulness to God for his pure and divine blessing that makes it possible for us to complete the project work successfully within the right time. We are really humbled to do this endeavor project under our respected professor and we wish our profound indebtedness to Supervisor **Dr. Ekta Gandotra**, **Assistant Professor**, Department of CSE & IT, Jaypee University of Information Technology (JUIT), Waknaghat. She has deep Knowledge of this project related stuff & her keen interest in guiding us in the field of "**Machine Learning**" to carry out this project. Her endless patience, scholarly guidance, perennial encouragement, constant and energetic supervision, and valuable advice pertaining to many pre-published drafts have made it possible for me to complete this project.

We would like to express our deep gratitude from the bottom of our heart to **Dr. Ekta Gandotra**, Department of CSE, for her generous help to finish our project.

We would also acknowledge each one of those individuals who have helped us directly or indirectly in making this project a possibility. In this juncture, we would like to thank the staff fraternity, individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Project Group No.: 47

Tanmay Agarwal	**Aryan Agnihotri**
(191416)	(191401)

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

1. **ANN**	Artificial Neural Network
2. **CNN**	Convolutional Neural Network
3. **ConvNet**	Convolutional Network
4. **DCNN**	Deep Convolutional Neural Network
5. **ILSVRC**	ImageNet Large Scale Visual Recognition Challenge
6. **MLP**	Multi-Layer Perceptron
7. **ReLU**	Rectified Linear Unit
8. **UV**	Ultra-Violet

LIST OF FIGURES

ABSTRACT

Cataracts are the leading cause of reversible blindness and visual impairment. Cataract surgery is one of the most commonly performed surgeries in the world. The only treatment for cataract is surgery. It is also one of the oldest. By 2022, approximately 1 billion people worldwide are blind due to cataracts (95 million), glaucoma (7.7 million) and refractive errors (84.4 million), etc. In addition, from 1 million to 2 million people go blind every year. In our world, someone goes blind every 5 seconds, and a child goes blind every minute. In 75% of these cases, blindness is treatable or preventable. However, there are now deep learning convolutional neural networks (CNNs) used for pattern recognition that help automate image classification. This study was proposed to minimize data loss and increase the accuracy of the cataract identification process performing alternating epochs. Research results show that adding more epochs affects the accuracy and lost data of convolutional neural networks. According to this study, epoch value of 51 had the highest value of 98%.

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a part of machine learning and are the heart of deep learning algorithms. Their name and structure are inspired by the human brain and mimic the way biological neurons signal to each other. Artificial neural networks (ANNs) consist of layers of nodes that contain an input layer, one or more hidden layers, and an output layer. Each node or say, an artificial neuron connects to another, and has an associated weight and threshold. If the output of any individual node is above a specified threshold, that node is activated and sends data to the next layer of the network. Otherwise, no data will be passed to the next network layer

CHAPTER-1
INTRODUCTION

1.1 Introduction

When the lens inside the eye becomes cloudy, you have a cataract. This blurring appears gradually with age and can cause glare or blurred vision. Cataracts make it difficult to distinguish distant objects or read small print. Cataracts are a leading cause of blindness and blindness in an aging population and place a huge financial burden on both countries and individuals [1]. Most cataracts are caused by aging of the eye lens. The lens is unique in that it's one of the very few bodily structures that keeps growing as it ages because it continues to accumulate new lens fibers while not replacing any that have already been there. The problems caused due to cataracts increases with age. Nearly 50% of people have cataracts by the age of 75. It is associated with decreased vision, contrast sensitivity, and severe dysfunction caused by fading bright light [2][3]. There is little you can do to slow this process other than wearing sunglasses to reduce your exposure to UV rays. There have been studies of oral medications and eye drops that can slow the formation of cataracts. Ultraviolet (UV) light can cause lens opacification both in vivo and in vitro [2].

Factors involved in the formation of cataracts vary according to socioeconomic and geographic differences. Although these cataracts are usually the result of aging, they can also be caused by a variety of other causes, such as eye trauma, long-term eye disease, radiation exposure, long-term steroid use, and more. This opacification of the natural lens is known as cataract and this result in blurring of vision.

Numerous factors, including starvation, acute infections that cause dehydration in children, and exposure to excessive UV radiation, seem to be significant in the developing countries [1]. Our eye has a high-powered natural lens which helps to focus the light rays falling into the eyes onto the retina. Like any other body tissue, the lens is made up of many cells. As these cells age, they become opaque and blurred vision. It's like a frozen or foggy window, hard to see. Cataracts develop slowly in everyone, but they develop over time. As everyone gets older, we can't get away from this. Cataracts begin to affect your vision and interfere with your vision. Early on, glasses and stronger lighting can help manage this condition. However, if the cataract begins to interfere with your daily activities, cataract surgery may be your only option.

The area of the lens that is most damaged by cataracts is frequently mentioned. There are numerous subsections in a comprehensive taxonomy that can be used to describe the various severity levels of symptoms for various cataract subtypes. A number of risk factors, such as those that could result in a narrower pupil or a rupture of the posterior capsule during surgery, could make cataract surgery more difficult. [1][5]. Over the next 20 years, the population of the world is anticipated to rise by nearly one-third. The majority of this expansion will take place in underdeveloped nations. The number of people over 65 will more than double throughout that time. Both developing and developed nations will experience this "ageing" of the population. [8].

A neural network that makes use of convolution is known as CNN or a convolutional neural network (ConvNet). Through the use of convolution, two kinds of data can be combined mathematically. In CNN, the data is convolutional after the input is filtered, and the result is a feature map. The filter is also referred as kernel, or feature detector, and its dimensions can be, for example, 3x3. To perform convolution, the kernel goes through the input image and performs element-by-element matrix multiplication. The result of each receptive field is written into the object map. Scrolling and indenting can aid in more precise image processing. The convolution layer has a number of filters, each of which produces a filter map. As a result, the layer's output will be a collection of filter maps piled on top of one another. CNN seeks to reduce the size of images without sacrificing elements that are important for precise prediction. Three different types of layers make up the ConvNet architecture: a convolutional layer, a pooling layer, and a fully connected layer.

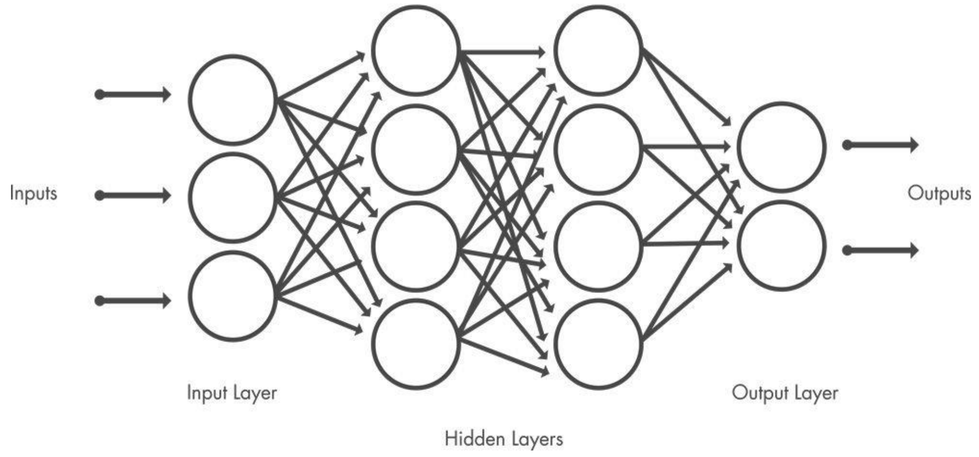


Fig. 1.1: Convolutional Neural Network

In terms of computing efficiency, CNNs outperform conventional neural networks. CNN uses parameter sharing and dimensionality reduction to facilitate quick and simple model deployment. They are adaptable to any technology, including cellphones. ConvNet is not flawless, though. It may appear to be a very sophisticated instrument, but it is still susceptible to attacks from the opposition. For image categorization, convolutional neural networks are frequently utilized. CNN can recognize various things in photos by identifying useful properties. They can be used in medicine, such as magnetic resonance imaging, thanks to their property. CNN is also useful in the farming industry.

1.2 Problem Statement

A cataract occurs when the lens, the small clear disc in the eye, forms a cloudy patch. Over time, these spots usually get bigger, blurring and clouding our vision, and eventually blindness. The proteins and fibers in the lens begin to break down, making our vision blurry or cloudy. Over time, cataracts can become worse and impair vision. Reading, working, engaging in hobbies, and participating in sports are just a few of the ways that visual impairment can impair critical skills and lower overall quality of life. If left untreated, cataracts can also lead to complete blindness. So, to avoid this problem there is a need of model like Cataract Detection which can help in detecting whether a person suffers from cataract or not, so that it can be detected & cured at an early stage.

1.3 Objectives

Objectives of the project are:

- To build the best classification model using Convolutional Neural Networks (CNN) such as VGG-16, VGG-19 and Inception V3 which gives the highest accuracy in predicting cataract in fundus images.
- To apply different Data Augmentation techniques to train a model to achieve an optimized model.
- To assess the importance of distinct features for cataract detection.

1.4 Methodology

The proposed methodology for identifying cataracts in the human eye is described in this section. In this study, a data collection made up of fundus photos of human eyes with and without cataracts was used. The data collection used to create these photographs was already available in the Kaggle repository. During CNN training, images were pre-processed to increase generalization. Additionally, data sets collected for categorization purposes are used to train CNNs. OpenCV framework is used to visualize the results. Finally, test results are analyzed and visualized.

1.4.1 Data Acquisition

Fundus images of the human eye were collected from images present in existing datasets in the Kaggle repository. The dataset also consisted of fundus images with retinal disease and glaucoma. Finally, the dataset consisted of 600 images, 300 of which were normal eye images and the rest were evenly distributed. Fig. 1.2 and Fig 1.3 show exemplary fundus images from normal eye and cataract eye datasets respectively.

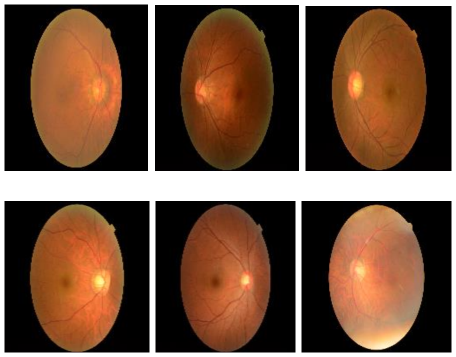


Fig. 1.2: Sample fundus images of dataset belonging to class – Normal Eye

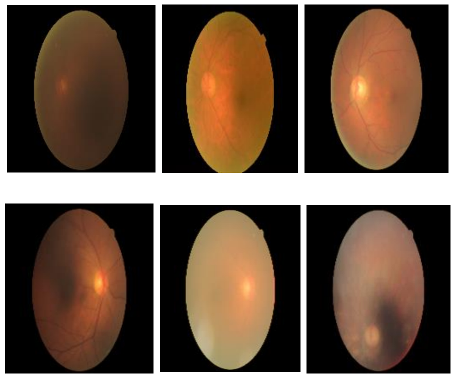


Fig. 1.3: Sample fundus images of dataset belong to class – Cataract Eye

1.4.2 Data Pre-processing

Data preparation is the process of processing unprocessed data for just the machine learning algorithms. The first and most important step in developing a machine learning technique is this one. Not often do we find clean, properly prepared data while developing a machine learning model. When working with data, it's important to clean it up and organize it properly. Therefore, we perform data preprocessing operations for this. Pre-processing implies the changes made to the data before the algorithm uses it. Data Preprocessing is a technique for turning a raw data set into a formatted one. Simply put, anytime data is gathered from multiple sources, it is received in raw format, which is not. "When it comes to designing a Machine Learning model, data preprocessing is the foremost step marking the commencement of the process. Usually, real-world data is" insufficient, conflicting, imperfect (contains errors or outliers), and most of the time lacks particular feature values. Here data preprocessing comes into the picture– it helps in data cleaning, data formatting, and data organizing of the raw data, as a result making it ready-to-use for Machine Learning models.

Since the data set consists of fundus images in .png format of various sizes (original sizes were close enough to 2000 x 1000 pixels), there was a need to convert the non-uniform sizes of these images to a consistent uniform one to train the model used. These images are converted to a uniform size of 299 × 299 pixels. This resolution is not small enough to make the features get lost and not large enough leading to exhaustion of resources. A 299 × 299 × 3 image (299 is representing height and width and 3 is representing the 3 channels in the image for each R-G-B) is provided as input for the proposed model after applying an RGB transform. The encoded labels and these pre-processed photos are added to separate lists. One for the class labels and one for the pre-processed photos. Additionally, to help with data generalization and enhance model performance, the images are subjected to data augmentation techniques such random rotations, translations, shears, and zooms. Data Augmentation resulted in a greater number of images resulting to 800 images in each section which increased our input size for each section, which later were used to train the model created during the project. Augmented images assured us that it is easier for the model to pick up the features in the images and that there is no overfitting in the model.

1.4.3 Logistic Regression

Machine learning has updated the categorization method of statistics known as logistic regression. A statistical technique referred as logistic regression is used to examine a data set when the outcome is influenced through one or even more independent factors. Finding the most accurate model to explain the relationship between the variables and relationship between the independent variable is the goal of utilizing logistic regression. A function referred as sigmoid function is used in logistic regression to convert predicted values into probabilities.

$$\mathbf{g(z) = 1/(1+e^{-z}) \text{ where } z = (Theta)^{T}x}$$

Sigmoid Function

This function converts any real value between 0 and 1 to another value. This function has precisely one inflection point and a non-negative derivative at each point.
The graph below can be used to depict this sigmoid curve.

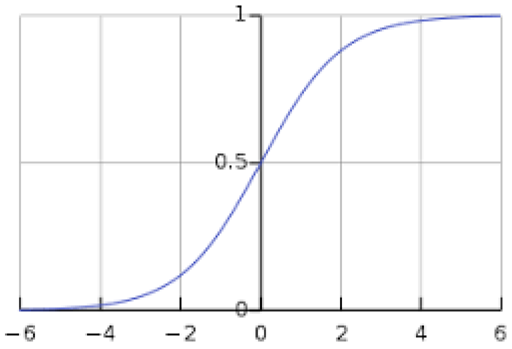


Fig. 1.4: Sigmoid Graph

We see that the y-axis values lie between 0 and 1 and intersect the v-axis at 0.5.

1.4.4 Support Vector Classifier

A support vector model (SVM) classifier is a type of machine learning technique that may be used to evaluate and classify data. Both regression and classification applications can benefit from the use of the supervised machine learning strategy known as a svm classifier. The support vector machine classifiers are used to locate the hyperplane that minimizes the difference between the two classes. For classification issues, a support vector classifier, a supervised learning technique, is widely used. By translating the pieces of data into a strong space and then determining the ideal hyperplane, SVC divides the information into two groups. Scikit-learn, a well-known machine learning package, provides an SVC implementation called Sklearn SVC.

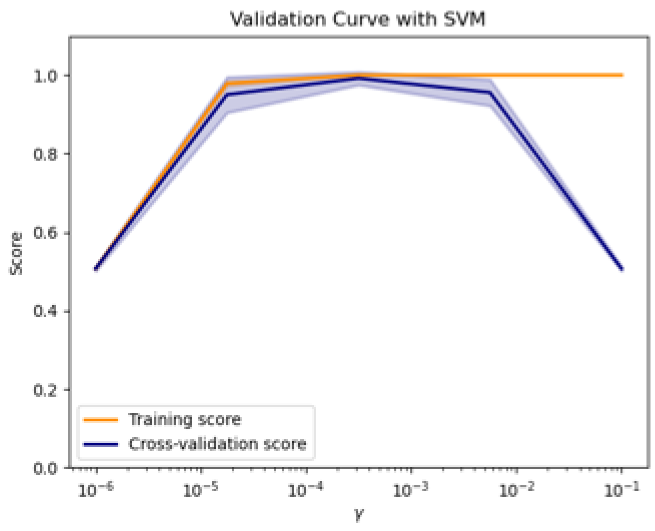


Fig. 1.5: SVC curve

1.4.5 Convolutional Neural Networks (CNN)

Deep learning is a novel method for pattern identification that uses the Convolutional Neural Network (CNN) (including images). Automating picture classification may be aided by this method. digital image of the retina. An input layer, an output layer, and a number of hidden layers make up a CNN. In the 1980s, CNNs were initially created and put to use. The best CNN could do at the time was decipher scribbled numbers. used to read pin codes, zip codes, and other data primarily in the postal industry.

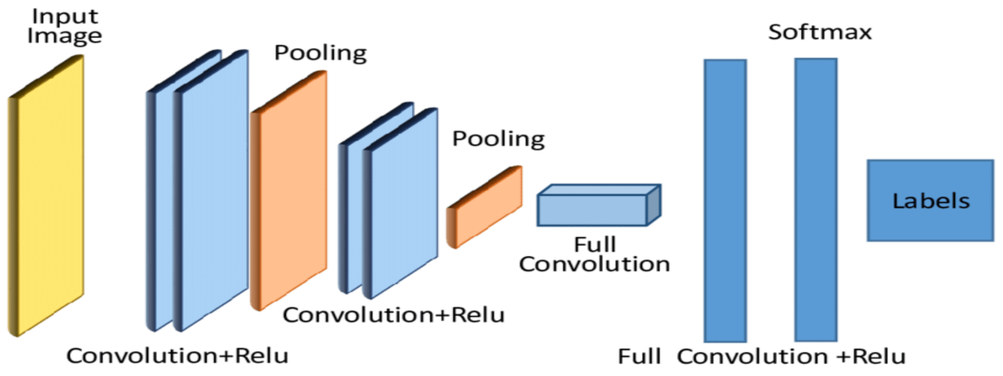


Fig. 1.6: CNN Architecture

Since it is effective at solving issues in computer vision domains, such as, among several others, systems for segmentation, monitoring devices, classification techniques, and other machine learning and video analysis, the convolutional neural network (CNN) is a somewhat well-known technique. The choice of several variables, including the size and number of kernels, the number of hidden layers, and the type of activation function, all have a direct impact on the CNN architecture [7]. In most cases, the first layer captures important properties like horizontal and diagonal edges. This result is forwarded to the next stage of object detection, which finds more intricate objects like corners or linked edges. More sophisticated features, such as objects, faces, and other attributes, can be recognized as you delve further into the network. Finding patterns in images for item, face, and scene identification using CNNs is very useful. It also does a great job at classifying non-image data, such as audio, time series, and signal data. CNNs have weights and biases in their neurons. With each new training example, the model constantly updates these values that it has gained throughout training. The CNN architecture then moves on to classification after learning the features at various levels.

CNN model is mainly composed of layers such as input layer, convolution layer, pooling layer and output layer.

1. Input Layer: The training data and input shape are provided to the model in the input layer, which is necessary for the model to function successfully in the next phases.
2. Convolution Layer: The foundation of a convolution neural network is the convolution layer. The model's convolution layer is crucial for adding nonlinearity. Convolution layer uses filters which brings more parameters to the computation which helps the model to train efficiently on the given dataset. A convolutional layer takes an image $A^{(m-1)}$ (with K^{m} channels) as input and computes a new image $A^{(m)}$ (consisting of Om channels) as output. The output at each channel is called the functional map and is calculated as:

$$A_o^{(m)} = g_m \left(\Sigma_k W_{ok}^{(m)} * A_k^{(m-1)} + b_o^{(m)} \right)$$

Where * denotes the (2D) convolution operation

$$W_{ok} * A_k[s,t] = \Sigma_{p,q} A_k[s+p,t+q] \; W_{ok}[P-1-p,Q-1-q]$$

where $W^{(m)}_{ok}$ is a matrix of the form $P_m \times Q_m$ and $b_o^{(m)}$. The matrix $W^{(m)}_{ok}$ parameterizes the spatial filters that the layer can use to detect or enhance features in the input image. The specific actions of this filter are automatically learned from the data in the network training process.

2.1 Filters: The function of filters is to operate as a single template or pattern that, when convolved across the input, identifies correspondences between the stored template and various locations/regions in the input image. The filter will carefully move across the image and extract its key details. There are different types of Filters like Gaussian Blur, Prewitt Filter and many more. Filters are generally matrix of smaller dimensions like 3x3 or 5x5, the smaller the dimensions the better the details in the input image is found.

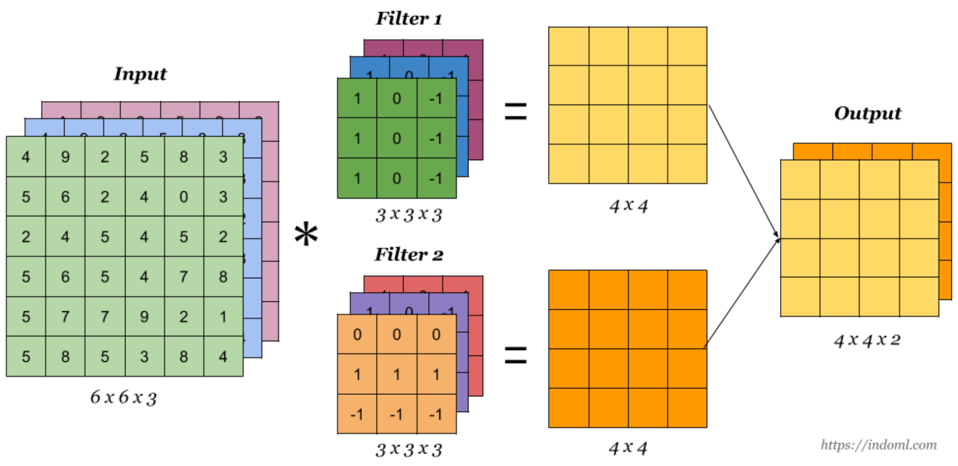


Fig. 1.7: Filters in CNN-2

Various filters can be used over the same input which can be fused to make output, different datasets react differently to different filters therefore it is necessary to try different types of filters to find out which one suite the given dataset (generally stack of images).

3. Pooling Layer: In order to pool the features inside the filter's coverage region, a two-dimensional filter must be applied to each channel of the feature map. To make feature maps less dimensional, layers are pooled. As a result, the network needs to execute fewer computations and learn fewer parameters. A pooling layer condenses the features that are present in the region of the feature map produced by the convolutional layer. As a result, subsequent operations are performed on the merged features rather than the carefully positioned features produced by the convolutional layer. The model is therefore better able to cope with changes in the positioning of features in the input image.

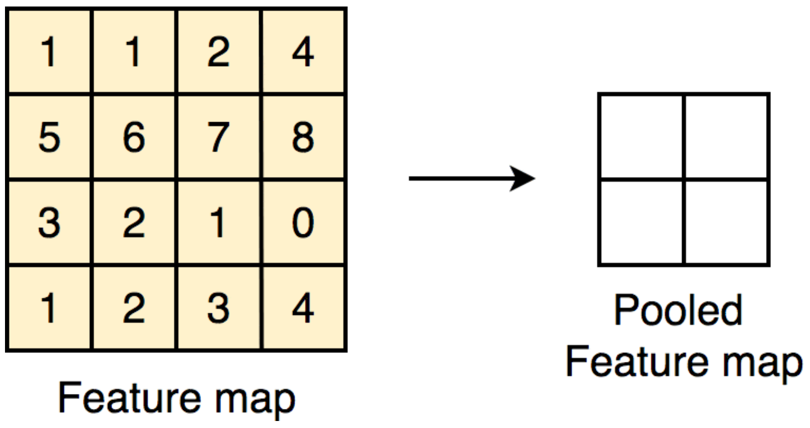


Fig. 1.8: Pooling Layer

Pooling Layer Types:

Max Pooling: This feature map area is covered by the filter, and a pooling procedure is used to choose the largest elements. So, a feature map comprising the standout features from the prior feature map will be the output following the max pooling layer.

Average Pooling: It assesses the mean of the components found in the feature map area that the filter has selected. This means that whereas average pooling delivers the average of the features present in the patches, max pooling gives the most prominent features in a specific patch of the feature map.

Global Pooling: Global pooling reduces each channel in the feature map to one value. Therefore, a n_h x n_w x n_c feature map reduces to a 1 x 1 x n_c feature map. This corresponds to using a filter of dimension n_h x n_w, i.e. dimensions of feature maps. Additionally, it can be either global max pooling or global average pooling.

4. Dense Layer in CNN: A dense layer in any neural network is one that has a strong connection to the layer below it. In other words, every neuron in the layer above is coupled to every neuron in the layer below. The most often utilized layer in artificial neural networks is this one. The model's dense layer neurons get the output from every neuron in the layer below. There, thick layer neurons multiply vectors on matrices. The row vector of the previous layer's output is equivalent to the column vector of the dense layer in the matrix-vector multiplication technique. Individual neurons in the dense layer receive the results from each neuron in the preceding layer. In this case, we say that the output has passed through the dense layer if the preceding layer outputs a (M x N) matrix after combining the results from each neuron. There should be N neurons in the dense layer. It can be used in Keras. The number of neurons or units specified in the dense layer has an impact on the output format. Dense layer functions in Keras implement the following operations –

Output(o/p) = Activation (Points (Input, Kernel) + Bias)

The function Activation is used in the equation above to perform per-element activation, where the kernel is the weight matrix and bias vector created by the layer. The Keras dense layer in the output layer performs the inner product of the input tensor and the weight kernel matrix. When a bias vector is introduced, the output values are elementwise activated.

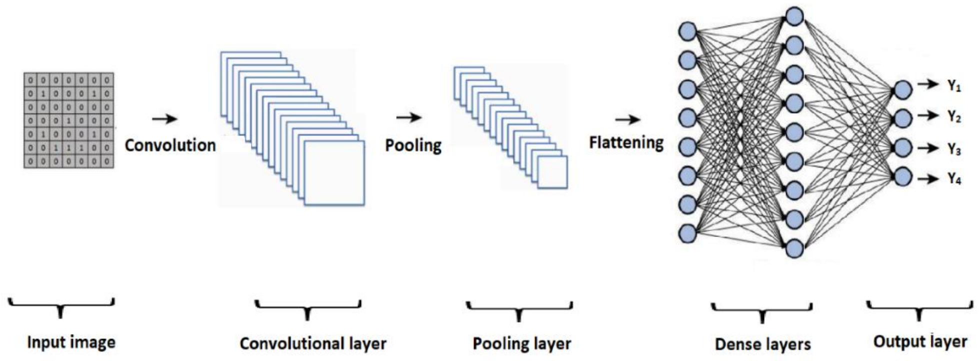


Fig. 1.9: Dense Layer in CNN

With transfer learning, we leverage an existing model to address several, connected issues. In essence, our goal is to use the knowledge gained from one activity to help in generalization in a different one. To address our issue, we employ the model's pre-trained weights or model architecture. On the flower dataset, we have used the pre-trained weights of the VGG16, VGG19, and Inception V3 models, changed the output layer, and solved a classification problem.

1.4.6 VGG-16

The 2014 ILSVR competition was won using the convolution neural network (CNN) architecture VGG16. It is said to be among the greatest vision model designs ever developed. The most notable aspect of VGG16 is that it consistently followed the same padding & max pooling layer of 2x2 filters with a stride 2 and emphasized maintaining convolution layers of 3x3 filtration systems with a stride 1 instead of switching between different padding and max pool layers. Convolution and max pooling layers are distributed uniformly over the whole architecture. Two completely connected layers (FC) as well as a softmax are used as its outputs in the end. The object categorization and recognition algorithm VGG16 has a 92.7% overall accuracy when categorizing 1000 images into thousand different groups. It is a popular method for categorizing photographs and is easy to use with transfer learning. VGG16's 16th digit denotes its sixteen weighted layers. VGG16 has 21 layers in total where 13 convolutional layers, 5 Max Pooling layers and 3 Dense layers, but only 16 of those are weighted layers, also referred as trainable parameters layers.

VGG16 was demonstrated to be the model with the top performance among all the settings on the ImageNet dataset. Any network setup is imagined to have a fixed input of 224×224 pixels with R, G, and B channels. The only pre-processing done is to normalize the RGB values of each pixel. To do this, the average score is removed from each pixel. The picture is then sent through the initial stacking of two convolution layers, which have a very tiny receptive size of 3×3, after ReLU activations. Each of these two layers has 64 filters. Following that, the activations pass through a secondary stack that is identical to the first but includes 128 filters as compared to the first stack's 64 filters. The size is therefore 56 x 56 x 128 after the second layer. A max pooling layer and 3 convolutional layers make up the final stack, which is subsequently appended. The convolution layers stacks are followed by three fully linked layers and a flattening layer. VGG16 can also be used as pre-trained model in the Keras Applications package. The ImageNet weights are present in the pre-trained model. To leverage the pre-trained model and train on your personalized photographs, we may apply transfer learning concepts.

1.4.7 VGG-19

Model used in the project is VGG-19 which unlike other models is 19 layers deep, where first layer is used for taking input and the last layer giving us the predicted values. VGG-19 was proved to be very useful for this model where we have used activation functions like 'ReLU', and the output layer is using 'sigmoid' function for giving us the final value ('softmax' is also a potent function which can be used). The Visual Geometry Group at the University of Oxford School of Engineering has developed a pre-trained convolutional neural network called VGG-19. From the ImageNet database, you can load a pretrained version of the network that was built using more than 1 million images. Images of objects like keyboards, pencils, and numerous animals can be categorized into his 1000 object categories by a pretrained network. The network subsequently learned detailed representations of a variety of images. The input network image measures 224 by 224 pixels.

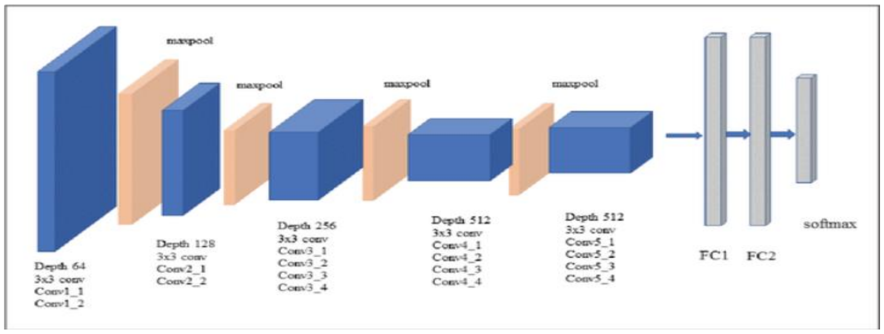


Fig. 1.10: VGG19 Architecture

The primary purpose for which the VGG mesh was developed was to obtain ILSVRC, but it has been used in many other ways-

- The authors have shared the model so that it can be used as-is or adjusted for other similar jobs. It has been utilized as an outstanding classification architecture for many other datasets.
- Transfer learning: Applicable to activities requiring facial recognition. You can easily modify the weights in other frames like Keras because they are widely available.
- Loss of content and style when using VGG-19 network.

1.4.8 InceptionV3

Network with convolutions Object recognition and image analysis are made easier with the use of Inception V3, a GoogleNet plugin that was created. The ImageNet Recognition Challenge first saw the Google Inception Convolutional Neural Network, that is now in its third generation. Inceptionv3 features less than 25 million parameters as opposed to AlexNet's 60 million, which was designed with the intention of allowing deeper networks without letting the number of variables to become unmanageably huge. Inception assists in the categorization of things in the area of computer vision, much the same as ImageNet may be thought of as a database of categorized visual objects. The Inceptionv3 architecture has been utilized in a large number of applications, typically in conjunction with ImageNet's pre-trained models. Inception-v3 is a 42-layer deep convolutional neural network. A pre - trained deep variant of the system which has been trained on even more than a million images is included in the ImageNet database. A variety of animals, a keyboard, a mouse, as well as a pencil are just a few of the 1000 distinct item categories that the pretrained network can classify images into. As a result, the network now includes rich visual features for a range of images. The input image for the network is 299 × 299 pixels in size. Total layer count for the Inception V3 model is 42, somewhat more than for the Inception V1 and V2 models. However, the efficiency of this technique is truly astounding.

1.4.9 ResNet50

Two key design tenets are followed by the ResNet architecture. First, each layer contains the same number of filters regardless of the output feature map's size. Second, it has twice as many filters to maintain the temporal complexity of each layer even if the size of the feature map is halved.

The components of the 50-layer ResNet50 architecture are listed in the table below:

• A convolution of a 7x7 kernel with 64 additional kernels and a 2-sized step.

• A two-sized stride max pooling layer.

• 9 more layers: 3x3,64 kernel convolution, 1x1,64 kernel, and 1x1,256 kernel versions. Three times are repeated for these three levels.

• 12 more layers with iterations of 4, 1x1, 128 kernels, 3x3, 128 kernels and 1x1, 512 kernels.

• Six iterations of 18 more layers each with 1x1, 256 cores and 2 cores, 3x3, 256 cores and 1x1, 1024 cores.

• Nine more layers that were iterated three times and had 1x1,512 cores, 3x3, 512 cores and 1x1, 2048 cores.

1.4.10 DenseNet201

A CNN of 201 layers deep is called DenseNet-201. Every layer is connected to the others by the Dense Convolutional Network (DenseNet). They greatly reduce the number of parameters, improve feature propagation, boost feature reuse, and resolve the vanishing-gradient issue.

DenseNet is founded on the idea that convolutional networks may be taught to be much deeper, more accurate, and more efficient if the connections between the layers near the input and the layers near the output are shorter.

1.4.11 MobileNet

Applications for embedded and mobile vision employ CNNs, such as MobileNet. These lightweight deep neural networks with low latency for embedded and mobile devices are constructed using depth-wise separable convolutions.

A collection of depth-separable convolutional layers forms the core of MobileNet. Each depth-wise separable convolution layer consists of a pointwise convolution and a depth-wise convolution. If depth-wise and pointwise convolutions are counted individually, a MobileNet has 28 layers. The 4.2 million parameters that make up a typical MobileNet can be further minimized by appropriately modifying the width multiplier hyperparameter.

CHAPTER-2
LITERATURE SURVEY

Here, we have studied the recent studies on the cataract and how cataract can act as an interface between clinicians and treatment.

D. Allen et al. [1] described the exciting developments in surgical fashion. Most cataracts were caused by the crystalline lens ageing. Young adults with cataracts are a prevalent occurrence in many poor nations, and they are frequently linked to diabetes, atopic illnesses, and the medications used to treat them. This evaluation only addressed cataract caused by ageing. The primary area of the lens damaged by cataracts was frequently mentioned in descriptions of the condition. There were several services included in a comprehensive bracket, and these can be useful in tying various symptom scenarios to various cataract subtypes.
Still, the straightforward trio of nuclear, cortical, and subcapsular cataract suffices for generalists. The problem of cataract blindness is far less prevalent in the developing world because most people waited until a cataract progressed or an eye has experienced excruciating vision loss from lens-contracted glaucoma before seeking help. This is due, in part, to a lack of awareness regarding cataract treatment, coitus prejudice, poor socioeconomic situations, and a lack of a government-sponsored old age conservation plan.

The essay also discussed cataract surgeries. For the birth of cataracts, there are two broad terms: extracapsular and intracapsular. Intracapsular birth entails taking out the entire lens together with its entire capsule. In the developed world, this style was no longer popular because the visual outcomes were typically worse and the operational and postoperative problems were less severe than with volition. The lens was extracted from its capsule, which was kept inside the eye and served as a barrier between the anterior and posterior, in order to give birth extracapsularly. parts as well as forming the most common place for the insertion of relief lenses. The nexus of the lens was removed enbloc during home extracapsular birth, necessitating a sizable cut.

Due to significant improvements in uncorrected visual acuity and lower operative and postoperative difficulties, complications from ultramodern cataract surgery were much better than they were 20 years ago. In 85–90% of instances undergoing cataract surgery, the patient will have 6/12 (20/40 or 0.5) stylish corrected vision; in cases without any optical comorbidities, such as macular degeneration, diabetic retinopathy, or glaucoma, this percentage jumps to about 95%. Choosing the right optic power for the relief lens was crucial because the surgery included negotiating the case's natural lens with an artificial implant. Most patients want to retain decent unaided distant vision, while some prefer to have some diplopia so that their fashionable unaided vision was at a close range. Colorful improvements in surgical technique, each quite slight, have added upto much greater efficacy. Phacoemulsification equipment has improved, partially due to a greater understanding of how it functions and partially due to improved microelectronic control. As a result, less physical energy is required to break the nexus apart than it was 10 times ago, and surgeons may now effectively remove the pieces using sophisticated vacuums and aspiration rates.

L. Robman et al. [30] looked at risk variables for cataract prevalence over the past 25 years. Only a few risk factors fit epidemiological requirements for causality, according to a recent assessment that included a thorough appraisal of study results. Smoking, excessive sun exposure, and other illnesses or associated treatments that affect lens metabolism had been proposed as the main causal external risk factors to date. employment, socioeconomic variables, and education. The causes indicated

by other criteria, like geography and even iris color, were yet unknown, but they were most likely indications of pathology.
It was the major cataract risk factor. Increasing age serves as a proxy for a number of potential external risk factors, the impacts of which are cumulative, even though the genetic component is self-evident. Finding the risk factors that were directly linked to the onset of cataracts can help prevent them from occurring. Risk factors hardly ever satisfy the requirements for causality. Smoking increases the risk of nuclear cataract while excessive UVB exposure and diabetes increase the risk of cortical cataract. Steroid therapy, diabetes, and ionizing radiation all increase the risk of stroke.
The development of posterior subcapsular opacities. Further research on the effects of medications on cataract development is necessary in order to distinguish between the impacts of disability and therapy. While "stop smoking" and "UVB protection" campaigns have gained popularity as preventative strategies, efforts to use antioxidants to actively prevent cataracts have proved ineffective. Research on cataracts has been made easier by recent advancements in measuring accuracy and standardization. However, this issue cannot be resolved by measurement accuracy. Larger studies routinely examine exposure to standard health risks, but risk factors that they were unaware of and so cannot be measured were crucial components of the equation. New ideas and theories are required.

According to **G. Brian et al. [5]**, cataracts were a serious global issue at the turn of the 20th century, but they were not generally acknowledged as such. Age-related increased in cataract prevalence are linked to ageing. Despite occurring more frequently and sooner in life, cataract incidence raised with age in developing nations. For instance, visually important cataracts appeared in the Indian study 14 years earlier than they did in a study of a similar nature conducted in the United States. These demographic trends will double the number of cataracts, increase visual morbidity, and increase the demand for cataract surgery, even if nothing else changes. By 2020, there were 40 million more individuals with cataracts than that time, or a significant reduction in vision of 3/60 or worse. The goal was to postpone the onset of cataracts and make cataract surgery easily accessible to everyone who requires it in order to stop this from happening.

Their knowledge of the origin of age-related cataract was still limited, despite the fact that several cross-sectional investigations of cataract risk factors have already been carried out and the outcomes of certain longitudinal research were available. Secondary cataracts were significantly less frequent and could develop as a result of trauma, inflammation inside the eye, radiation exposure, and other things.

Surgery was used to cure cataracts. But not everyone could afford this operation, and even when it was, the outcomes were not always the same. A choice must be made about who should have the surgery, how it should be performed and delivered, and how it should be paid for based on the current situation and the size of the cataract needed to justify the procedure. Criteria must be created to determine whether cataract surgery was genuinely necessary given the reduced threshold for the treatment. The choice had effects on the scheduling, provision, financing, and accessibility of cataract surgical services. An important and growing concern on a global scale was cataract. The goal was to stop or slow the development of cataracts and treat those that already exist. It was necessary to identify, improve, and execute preventive interventions through research, alterations in governmental regulations and laws, and alterations in collective and individual behavior.

I. Weni et al. [7] suggested choosing the best CNN for cataract identification with a predetermined number of epochs. Many earlier scholars have classified or identified cataracts in their studies. To demonstrate the accuracy and dependability of the developed system, Deep CNN, GoogleNet Transfer Learning, and MATLAB were mostly used to identify cataract eye images and their characteristics. 87.5% accuracy was achieved using transfer learning and TensorFlow with data from an initial V3 architecture trained on a deep learning image network split into mature and immature cataracts. Zhang automated the detection and scoring of cataracts using a Deep Convolutional Neural Network (DCNN), and he also displayed numerous feature maps on the pool5 layer with high-order empirical semantic meaning that describes the feature representation obtained by the DCNN. The classification accuracy of DCNN improved and the range of fluctuation inaccuracy became more steadier as the number of samples that were accessible grow. Based on a multi-layer perceptron (MLP) that was intended to process two-dimensional data in the form of images, a convolutional neural network (CNN) was created. Due to the large network depth of CNN, a deep neural network type that was frequently employed for complicated image data. In theory, the image classification process could only use MLP; however, one of this MLP method's shortcomings was that it was unsuitable because it cannot store spatial information from the mind data and treats each pixel as an independent or large element, making it possible to obtain unfavorable results. In the CNN architecture, the convolutional layer is a component of the stage. The output of the preceding layer was subjected to a convolution operation in this stage. The architecture of the CNN network was primarily underpinned by this layer of operations. In this process, the output feature was applied to the input image as a feature map. The weights (w) in machine learning applications were multidimensional arrays of parameters that can be learned.

The output signal was subjected to mathematical processes known as activation functions. Based on a weighted total of the input, an activation function was utilized to assess whether a neuron is active or not. Convolutional neural networks frequently employ the activation functions tanh (), Rectified Linear Unit (ReLu), sigmoid, and softmax. The activation functions of ReLu and SoftMax were used in this study

1. ReLU -

If the input value is negative, the ReLU function's output value for the neuron can be represented as 0. If the input value is positive, the activation input value itself is the neuron's output. The following equation illustrates this functional equation:

$$\mathbf{f(x) = max\ (0, x)}$$

2. Softmax-

The last layer on the neural network uses softmax activation. Compared to ReLU, sigmoid, and tanh, Softmax is more often utilised (). The neural network's output may be changed to better reflect the underlying probability distribution by using Softmax. This is how the softmax equation is shown:

$$f(x) = (e^{xi})/(\Sigma^{k}_{j=1} e^{xj}) \text{ for } i = 1,2,3,\ldots,k$$

The size of the matrix was reduced via max-pooling or average pooling once the activation function had been calculated. A matrix with fewer dimensions than the original image was the output of the pooling procedure. The desired feature map was obtained by a convolution and pooling method and supplied into the fully meshed layer. One of the parameters required to build the model was the optimizer. In order to improve the model's accuracy, the optimizer was crucial. In the study, the optimizer parameter was determined using Adam's approach.

One of the hyperparameters that had a big impact on how well the CNN model performs was learning rate. To determine the appropriate learning rate, a technique known as cycling was used. In this method, the training was repeated several times with the learning rate starting at a low value, and with each iteration, the learning level increased. The gained losses govern each repetition, and if the gained loss gets increased, the search process ends abruptly. After running the CNN modelling process and getting the best accuracy result, the image testing method was used by adding a new image. It was able to recognize the cataract eye images that were submitted into the system during image testing. Eight separate photos' test data was to be used for testing. The photos were separated into two categories: eyes with cataracts and eyes with normal vision. Manual testing was used to assess the effectiveness and correctness of the test before it was carried out by the user.

A CNN model that was to be used to test the image could be created based on the research that was done. It could be disregarded that the greater the age values employed, the higher the value of the model by comparing different epoch values. In this study, the greatest value was obtained with a value of 95% utilizing an epoch value of 50.
In addition to receive a high score for accuracy, the CNN model used by the examiner 50 times in total produces accurate findings. It was successful to obtain images that fit the requested class using the model that was created. 10 photos underwent extensive examination, and their average accuracy was 88%.

D. Lam et al. [8] focused only on cataracts. The article examined many cataract kinds, including morphological classification, age-related cataract, infantile cataract, and secondary cataract. According to research on surgery rates, industrialized regions, such as North America, Western Europe, and Japan, execute 4,000–6,000 cataract procedures annually, compared to 500 operations per 1 million persons in underdeveloped regions. India, which enjoyed a high rate of operations, was the exception to those patterns.
Additionally, it described the lens protein. The told that the lens was a special organ with very little cellular metabolism and neither arterial nor venous circulation. Organelles like nuclei, mitochondria, and ribosomes were absent from the elongated lens fiber cells, which were instead stuffed full of the lens crystallin, a distinctive protein. The maintenance of the lens's transparency over a person's

lifetime depended on the crystalline retention of the native structure and solubility of the lens proteins, which were neither replaced nor degraded.

It has frequently been assumed that crystalline aggregates were not the result of recurrent protein interactions because the shape of the aggregates was still unclear. However, the interactions—such as changing domains or introducing loops and leaves—were probably particular and recurring. Some eyes showed severe circumstances that enhanced surgical difficulties and danger of complications, as with any eye surgery. The most frequent causes of surgical problems were small pupils, advanced cataracts, and weak zonules. Thankfully, advancements are being made. The prognosis of cataract surgery conducted on problematic eyes had improved thanks to advancements in phacoemulsification technology, the creation of ancillary devices, and enhanced surgical training and experience.

Between 1990 and 2010, the age-standardized prevalence of cataracts considerably dropped globally. Between 1990 and 2010, respectively, the numbers of people with blindness and moderate to severe vision impairment brought on by cataracts fell from 12.3 million to 10.8 million and from 44.0 million to 35.2 million. For those under 50 years old, the prevalence of cataract as a cause of blindness and moderate to severe vision impairment was 0.7% and 2.2%, respectively, in 2010. This was a decline from 4.4% for moderate to severe vision impairment and 1.3% for blindness in 1990, respectively. East Asia, tropical Latin America, and Western Europe saw the greatest reduction in cataracts as a cause of blindness and visual impairment.

R. Hossain et. al. [12] research addressed a variety of statistical techniques created for cataract diagnosis, but because to the constraints of feature extraction and preprocessing, those techniques do not yield satisfying results. Based on a 2D Gaussian filter and a decision tree algorithm trained on 1355 retinal fundus pictures, they created a cataract diagnosis system that had a 92.8% accuracy rate. However, large photos could not be used with the decision tree. It had reported on a DCNN with a random forest-based cataract classification model. They used 3460 fundus images to train the cataract detection system and tested 1948 images with 97.04% accuracy. An active shape model with SVM was recently employed in a work that trained on more than 5000 photos and had a 95.00% accuracy rate. The high-dimensional feature map cannot be trained using the SVM kernel. The wavelet approach and principle component analysis (PCA) were used to classify and identify cataracts. The statistical based learning algorithms (SVM, decision tree, random forest, etc.) are required prior training knowledge and cannot train high-dimensional features, according to the majority of past research. The overfitting and underfitting issues caused by layer design could not be solved by DCNN or other contemporary methods.

This study suggested an automated cataract detection system, which was made up of two essential modules: training and testing, which illustrated the suggested system's design. Using the DCNNs classifier technique, the training module was trained using the labelled datasets. Unlabeled data are tested by the test module. A supervised DCNN algorithm was utilized to implement the cataract detection train module. This module was a cataract and non-cataract retinal fundus image identification model using Res-Net50-like architecture. Training labelled datasets ($D = (x_1, y_1), (x_2, y_2), (x_3, y_3)\ldots...(x_n, y_n)$), are represented by the pair (x_i, y_i), which identified the input picture with labelled data, and n, which represented the total amount of marked data in the training system. where C was the class number, and y_i was the indicator. For purposes of detection, $C = 2$. The test module contained unlabeled picture data. The DCNN model was then used by the module to determine whether or not the photos contained cataracts. The cataract detection value served as the initialization for the DCNN's trainable parameters. Only the feature was extracted from the learnt parameters via fully linked layers and convolution layer parameters.

They concluded that the DCNN architecture was suggested in this study as the basis for an autonomous cataract detection system. On the test sets, the suggested method had an accuracy of 95.77%. The suggested DCNN approach circumvents the conventional feature extraction constraints and does not call for image pre-processing. The graphical and statistical findings demonstrate that the suggested method also resolved issues with overfitting and inadequate models. Ophthalmologists could benefit greatly from the suggested approach by integrating it into Internet of Things (IoT) devices. Rural and low-income persons may readily use the suggested system since it is less expensive and offers superior cataract treatment options. On retinal fundus pictures, the suggested approach was unable to identify mild or partial cataract. Future research and DCNN system development would focus on the detection of mild stage or partial cataract.

Kumar et al. [13] described that Healthcare is only one of the many sectors where artificial intelligence (AI) has recently experienced great growth. Traditional diagnostic and therapeutic approaches have been significantly impacted in recent years by significant developments in AI, particularly in machine learning (ML) and deep learning (DL). Even though deep learning (DL) has long been used in voice recognition, image identification, and natural language processing, it is just now starting to have an impact on healthcare. Optical coherence tomography (OCT) and fundus digital photography are currently the main imaging methods for the early diagnosis and treatment of eye disorders. To train and evaluate AI-based intelligence learning algorithms, a variety of photos of eye illnesses including diabetic macular edoema (DME) and choroidal neovascularization (CNV), DRUSEN, GLAUCOMA, NORMAL, and CATARACTS have been employed. For the purpose of predicting eye illnesses, this study offered different transfer learning models. Basic CNN, Deep CNN, AlexNet 2, Xception, Inception V3, ResNet 50, and DenseNet121 are the deep transfer learning methodologies employed in the study. The simulation results show that the ResNet50 performed better than all other methods, with a validation accuracy of 98.9%. Achieving 98.4% accuracy, the Xception model performed well. The Xception model successfully achieved training and validation losses of 0.15 and 0.05, respectively. The best root mean squared error that the Xception model could obtain was 0.22. In order to improve clinical decision-making, which has a number of applications for ophthalmologists, research is being done.

CHAPTER-3
SYSTEM DEVELOPMENT

3.1 DESIGN

- IMPORT IMAGE DATASET
- DATA AUGMENTATION
- LOGISTIC REGRESSION & SVC MODEL
- IMPLEMENTING CONVOLUTIONAL NEURAL NETWORKS
- TRAIN NETWORK
- EVALUATING BEST EPOCH VALUE
- PLOTTING GRAPH
- APPLYING STATISTICAL MEASURES
- EVALUATING ACCURACY

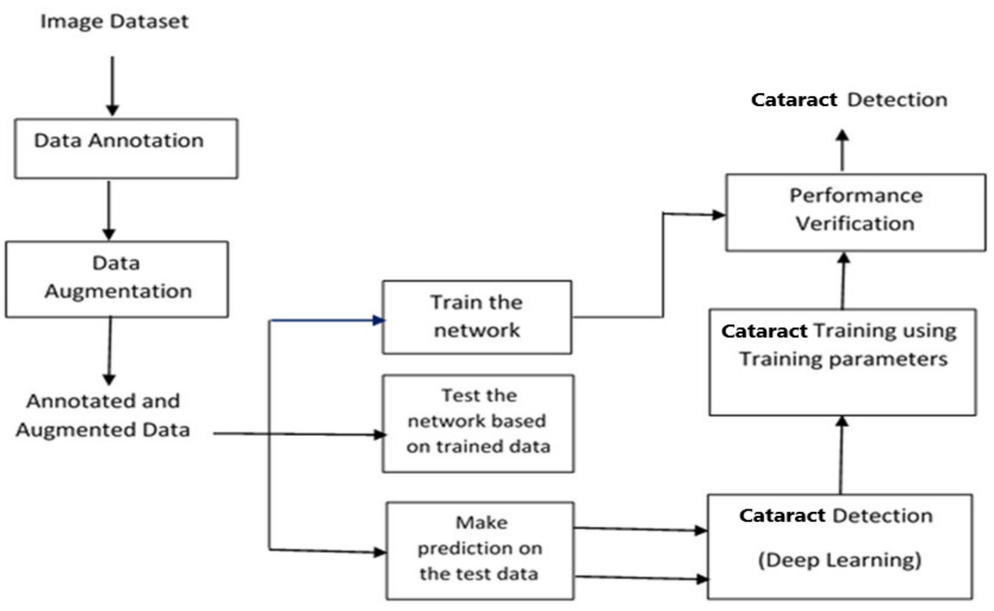
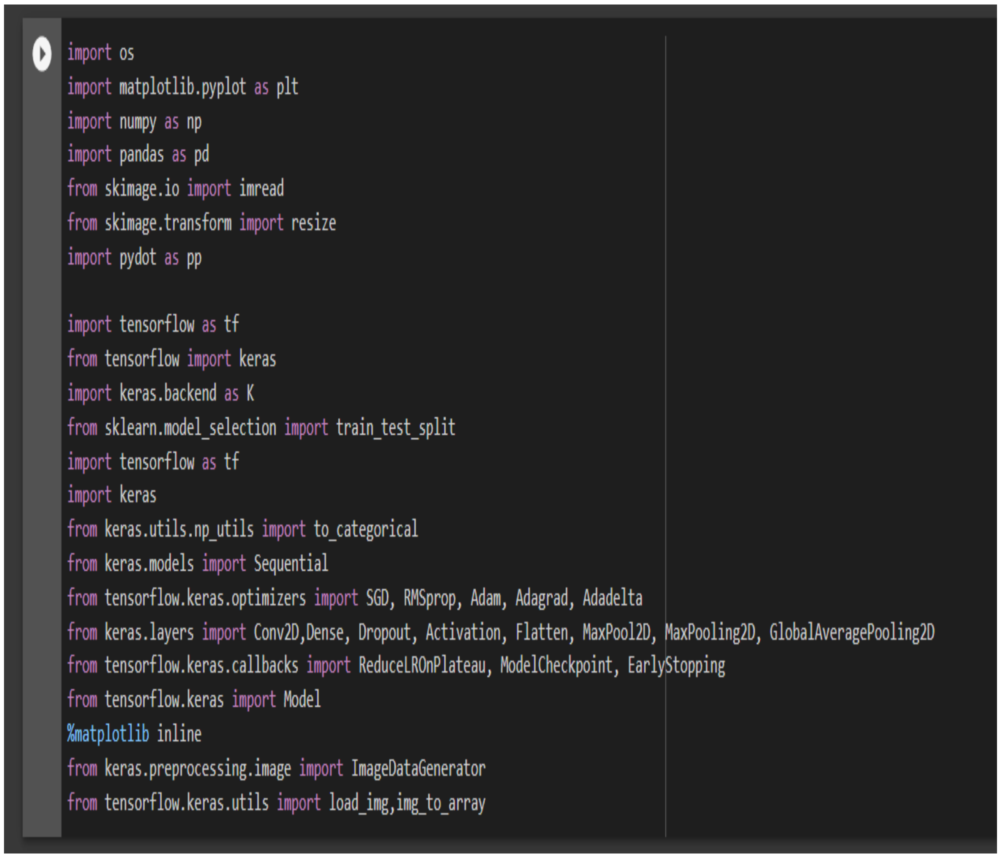


Image Dataset
Data Annotation
Data Augmentation
Annotated and Augmented Data
Train the network
Test the network based on trained data
Make prediction on the test data
Cataract Detection
Performance Verification
Cataract Training using Training parameters
Cataract Detection (Deep Learning)

Fig. 3.1: Data Flow Diagram

➢ IMPORTING LIBRARIES:

The following libraries have been used for the implementation of the model and for training the dataset:

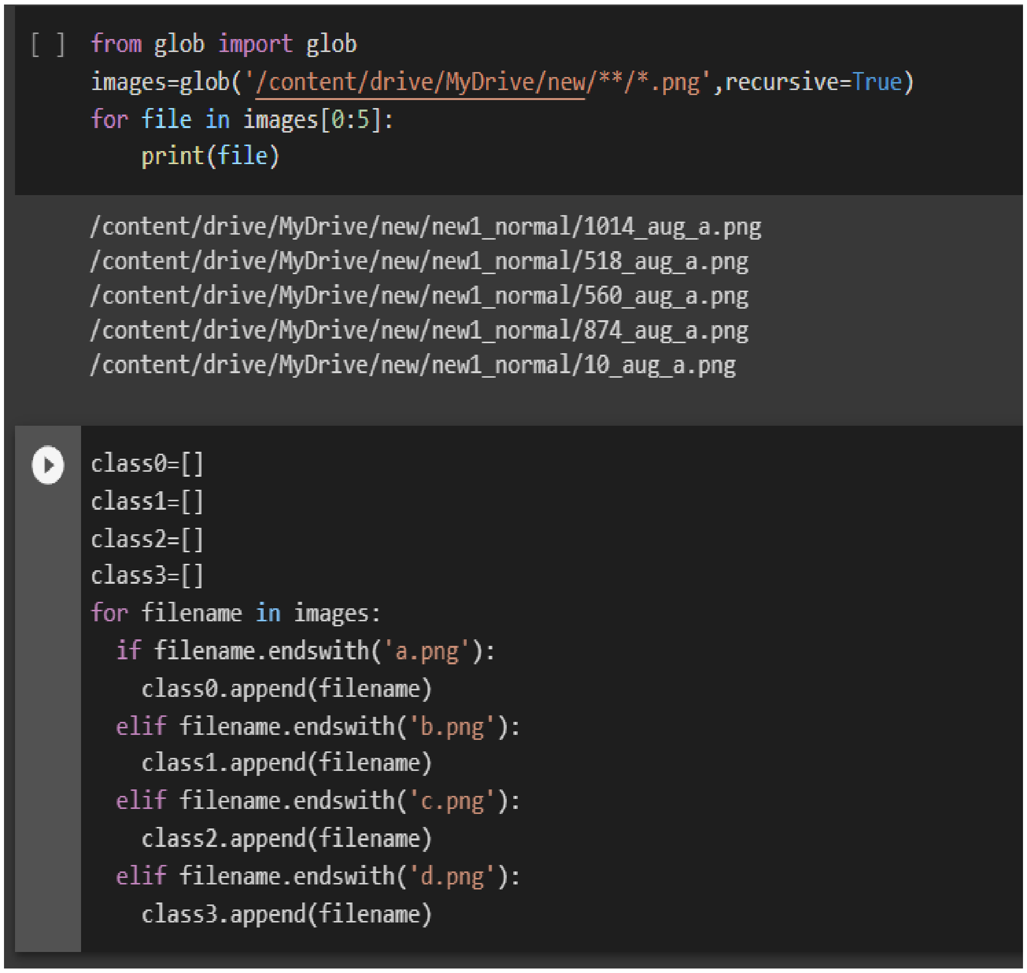


```
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from skimage.io import imread
from skimage.transform import resize
import pydot as pp

import tensorflow as tf
from tensorflow import keras
import keras.backend as K
from sklearn.model_selection import train_test_split
import tensorflow as tf
import keras
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from tensorflow.keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta
from keras.layers import Conv2D,Dense, Dropout, Activation, Flatten, MaxPool2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from tensorflow.keras import Model
%matplotlib inline
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import load_img,img_to_array
```

Fig. 3.2: Importing Libraries

In Python, a file is regarded as a module. The module must be imported using the import keyword before being used. By importing the module, the function or variables existing in the file can be utilised in another file. Instead of importing the entire module, we can merely import the necessary functions and variable names from the module. The "from" keyword can be used to import only certain items when we only want those items imported. By using import and the module name, i.e., the filename or the library to be used, we can import the entire module.

➢ IMPORTING DATASET:



```
from glob import glob
images=glob('/content/drive/MyDrive/new/**/*.png',recursive=True)
for file in images[0:5]:
    print(file)
```

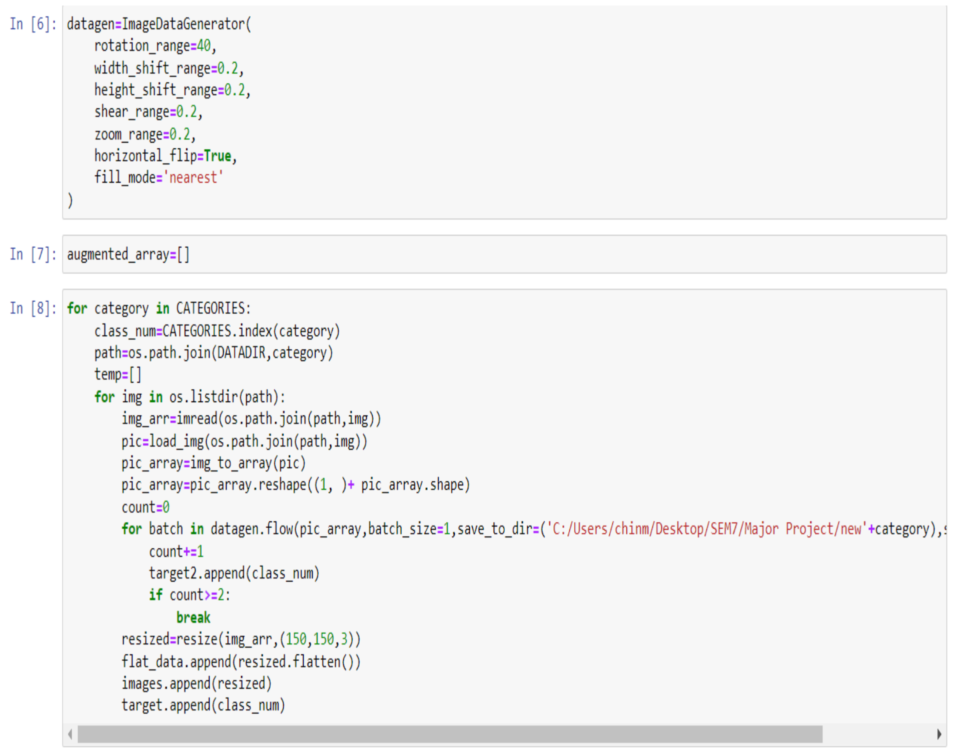
```
/content/drive/MyDrive/new/new1_normal/1014_aug_a.png
/content/drive/MyDrive/new/new1_normal/518_aug_a.png
/content/drive/MyDrive/new/new1_normal/560_aug_a.png
/content/drive/MyDrive/new/new1_normal/874_aug_a.png
/content/drive/MyDrive/new/new1_normal/10_aug_a.png
```

```
class0=[]
class1=[]
class2=[]
class3=[]
for filename in images:
  if filename.endswith('a.png'):
    class0.append(filename)
  elif filename.endswith('b.png'):
    class1.append(filename)
  elif filename.endswith('c.png'):
    class2.append(filename)
  elif filename.endswith('d.png'):
    class3.append(filename)
```

Fig. 3.3: Importing of Augmented Dataset

Uploading of Cataract dataset after Augmentation of fundus images of Cataract which were taken from the Kaggle repository. The pre-existing dataset consist of 600 images which were distributed among four classes – Normal, Cataract, Glaucoma and Retina Disease.

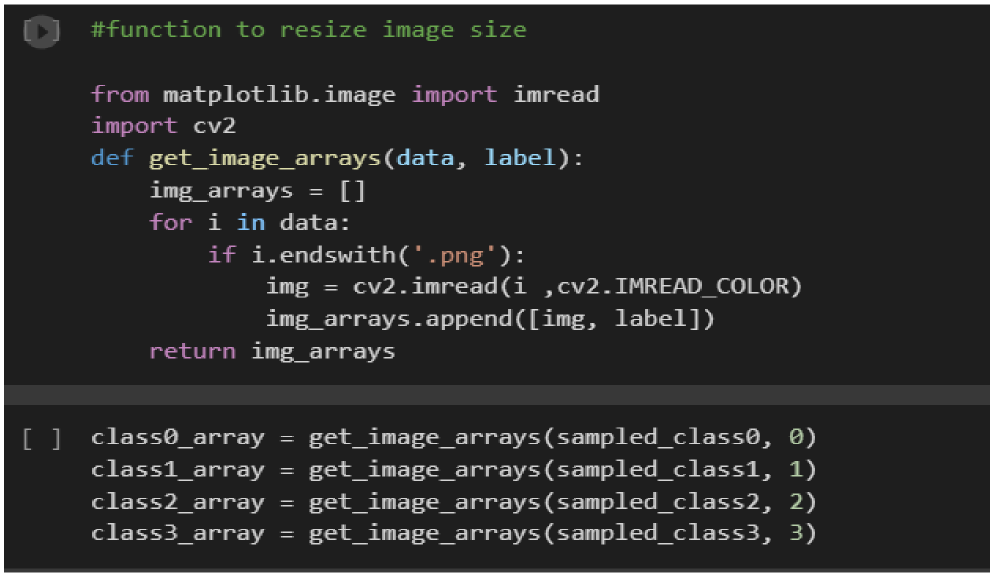
➢ DATA AUGMENTATION:



```
In [6]: datagen=ImageDataGenerator(
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest'
        )

In [7]: augmented_array=[]

In [8]: for category in CATEGORIES:
            class_num=CATEGORIES.index(category)
            path=os.path.join(DATADIR,category)
            temp=[]
            for img in os.listdir(path):
                img_arr=imread(os.path.join(path,img))
                pic=load_img(os.path.join(path,img))
                pic_array=img_to_array(pic)
                pic_array=pic_array.reshape((1, )+ pic_array.shape)
                count=0
                for batch in datagen.flow(pic_array,batch_size=1,save_to_dir=('C:/Users/chinm/Desktop/SEM7/Major Project/new'+category),
                    count+=1
                    target2.append(class_num)
                    if count>=2:
                        break
                resized=resize(img_arr,(150,150,3))
                flat_data.append(resized.flatten())
                images.append(resized)
                target.append(class_num)
```

Fig. 3.4: Data Augmentation

Algorithms can utilize machine learning to classify and differentiate between different objects for image recognition. This emerging method employs data augmentation to build models with higher performance. Machine learning models need to be capable of identifying objects in all lighting conditions, including rotation, close-up, and fuzzy pictures. A synthetic approach of combining training data with precise changes was sought by researchers. To artificially enhance the amount of data by creating extra data points from the present data, a variety of techniques referred to as data augmentation may be applied. Deep learning models are used in this to either add new data points or make a few minor adjustments to the current data. Businesses may be able to reduce their dependency on the preparation and gathering of training data by adopting data augmentation to develop machine learning models more rapidly and precisely. In order to increase the performance and output of machine learning models, training datasets are supplemented with more distinctive instances. The effectiveness and precision of machine learning models are enhanced with large and sufficient datasets. To collect and organize data for machine learning models, a lot of effort and money may be required. Companies that use data augmentation techniques. Data augmentation offers a variety of options for modifying the original image and may be helpful in providing enough data for bigger models. Making straightforward modifications to visual data is common in data augmentation. The following are typical image processing tasks for data augmentation: padding, random rotation, rescaling, translation, cropping, zooming, vertical and horizontal flipping, etc.



```
#function to resize image size

from matplotlib.image import imread
import cv2
def get_image_arrays(data, label):
    img_arrays = []
    for i in data:
        if i.endswith('.png'):
            img = cv2.imread(i ,cv2.IMREAD_COLOR)
            img_arrays.append([img, label])
    return img_arrays
```

```
[ ] class0_array = get_image_arrays(sampled_class0, 0)
    class1_array = get_image_arrays(sampled_class1, 1)
    class2_array = get_image_arrays(sampled_class2, 2)
    class3_array = get_image_arrays(sampled_class3, 3)
```

Fig. 3.5: Resizing of Image

A machine learning model receives a one-dimensional feature vector as its input. Convolutionary neural networks, two- and three-dimensional feature tensors, and other more recent learning models may also be used. Throughout training, the machine adjusts its internal parameters to direct each feature tensor in the desired direction. Following training, the computer can predict the target for feature tensors that were previously unknown. The same number of features must be present in each sample, in other words. The processing of words and pictures is hampered by the fact that they frequently have a wide range of sizes and, as a result, a variety of properties. A three-dimensional tensor, where each channel's value corresponds to a feature, serves as the input for a convolutional neural network (CNN) used to classify an image. The three-dimensional feature tensor has to have the same dimensions for every picture. The size of images and the feature tensors that accompany them, however, is often different. Photographs may be resized to the same size, but it might be challenging to do so without destroying any potential patterns.

- IMAGES AFTER DATA AUGMENTATION:

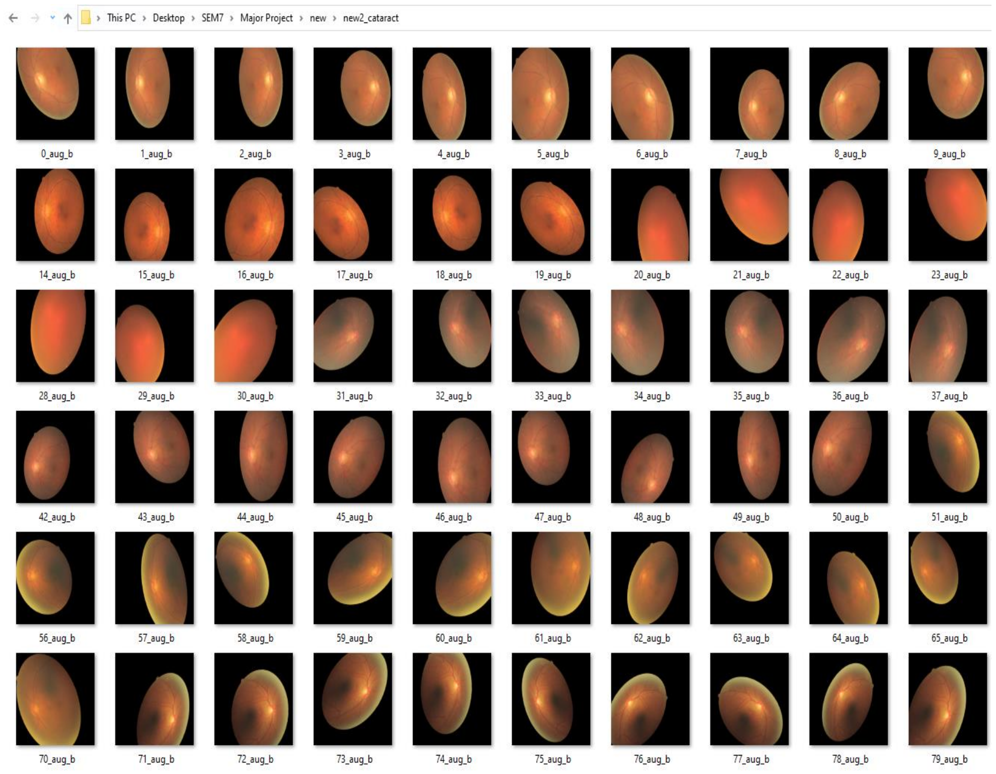
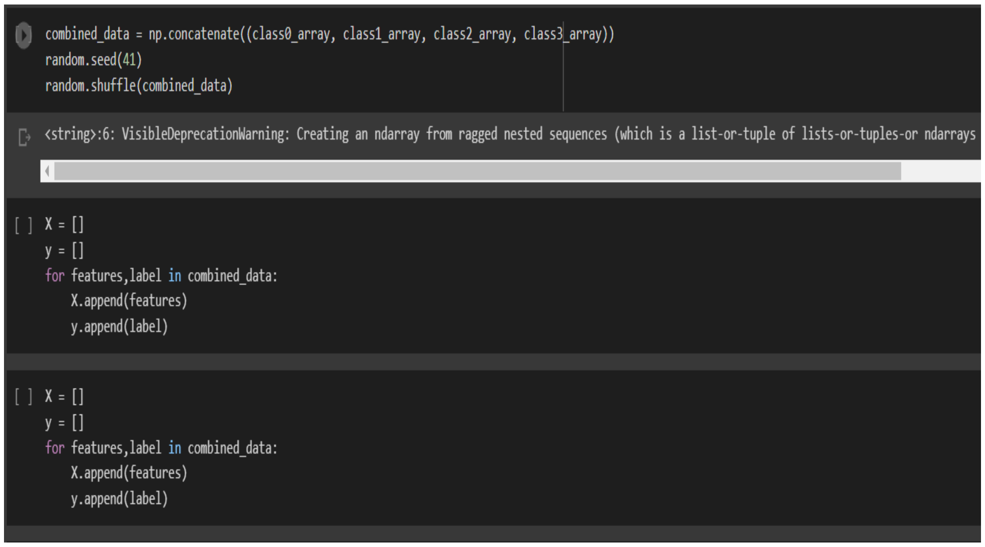


Fig. 3.6: Cataract Images after Data Augmentation

➢ COMBINING DATASET:



```
combined_data = np.concatenate((class0_array, class1_array, class2_array, class3_array))
random.seed(41)
random.shuffle(combined_data)
```

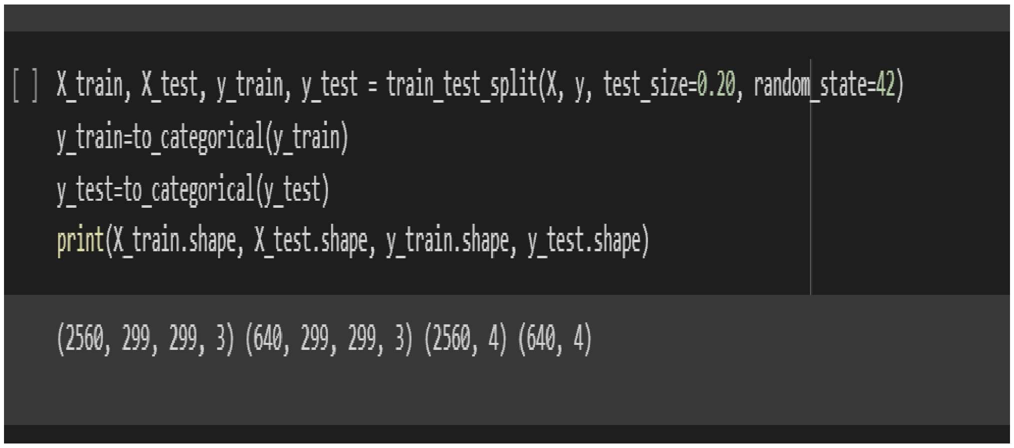
```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays
```

```
[ ] X = []
    y = []
    for features,label in combined_data:
        X.append(features)
        y.append(label)
```

```
[ ] X = []
    y = []
    for features,label in combined_data:
        X.append(features)
        y.append(label)
```

Fig. 3.7: Combining images of all the classes

➢ TRAINING DATASET:



```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
    y_train=to_categorical(y_train)
    y_test=to_categorical(y_test)
    print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

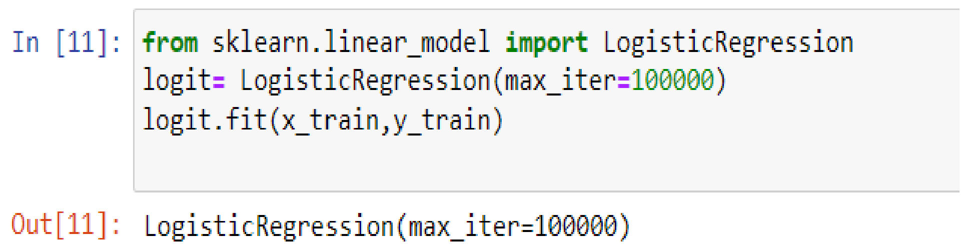
```
(2560, 299, 299, 3) (640, 299, 299, 3) (2560, 4) (640, 4)
```

Fig. 3.8: Splitting dataset into training & testing

Your model might overfit the training set, which is a risk throughout the training process. In other words, the model can develop a function that is highly specialized and effective on your training data but ineffective on photos it has never seen before. A method for assessing a machine learning system's performance is the train-test split. Problems requiring classification, regression, or any other supervised learning technique may be resolved using it. The dataset has to be split into two groups first. The training dataset serves as the first subset for parameter estimation. The model is trained using the dataset's input component rather than the second subset, and its estimates are then generated and contrasted to the predicted value. The test dataset is the second sample in question. When machine learning algorithms are used to generate predictions based on data that was not used to train the system, its performance is evaluated using the train-test split technique. To evaluate prediction performance objectively, you must divide your dataset.

▪ IMPLEMENTATION OF LOGISTIC REGRESSION

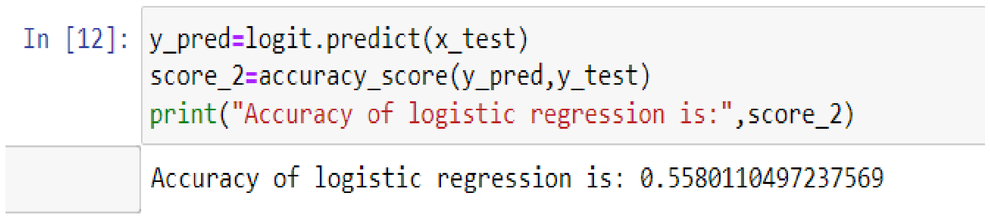
➢ IMPORTING OF LOGISTIC REGRESSION



```
In [11]: from sklearn.linear_model import LogisticRegression
         logit= LogisticRegression(max_iter=100000)
         logit.fit(x_train,y_train)

Out[11]: LogisticRegression(max_iter=100000)
```

Fig. 3.9: Importing and Implementing Logistic Regression

➢ RESULT OF LOGISTIC REGRESSION MODEL

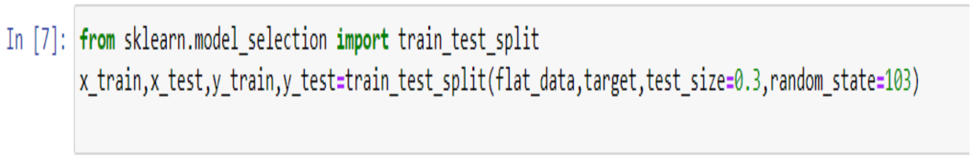


```
In [12]: y_pred=logit.predict(x_test)
         score_2=accuracy_score(y_pred,y_test)
         print("Accuracy of logistic regression is:",score_2)

         Accuracy of logistic regression is: 0.5580110497237569
```

Fig. 3.10: Accuracy of Logistic Regression

▪ IMPLEMENTATION OF SUPPORT VECTOR CLASSIFIER

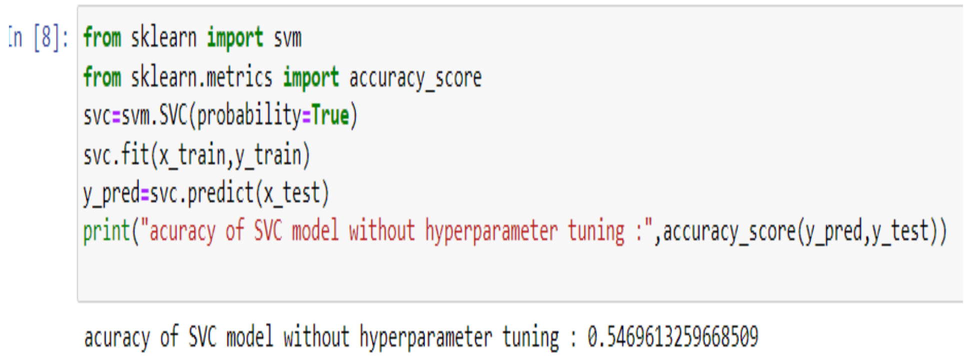
➢ TRAIN TEST SPLIT OF DATA



```
In [7]: from sklearn.model_selection import train_test_split
        x_train,x_test,y_train,y_test=train_test_split(flat_data,target,test_size=0.3,random_state=103)
```

Fig. 3.11: Importing and Training of SVC

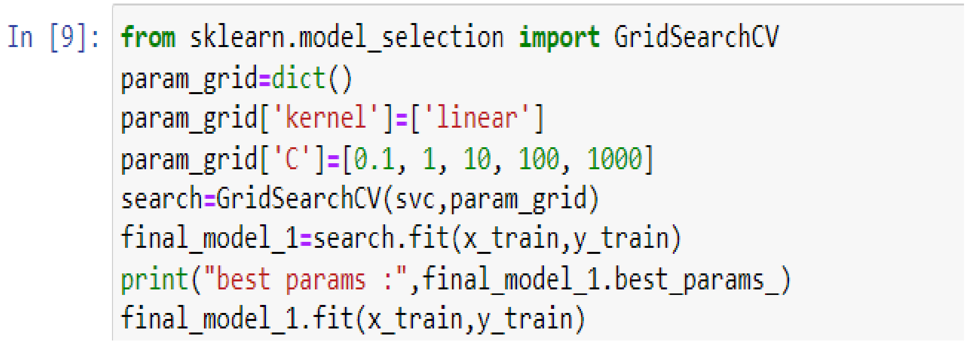
➢ ACCURACY OF LOGISTIC REGRESSION MODEL



```
In [8]: from sklearn import svm
        from sklearn.metrics import accuracy_score
        svc=svm.SVC(probability=True)
        svc.fit(x_train,y_train)
        y_pred=svc.predict(x_test)
        print("acuracy of SVC model without hyperparameter tuning :",accuracy_score(y_pred,y_test))

        acuracy of SVC model without hyperparameter tuning : 0.5469613259668509
```

Fig. 3.12: Accuracy of SVC before Hyper-parameter tuning

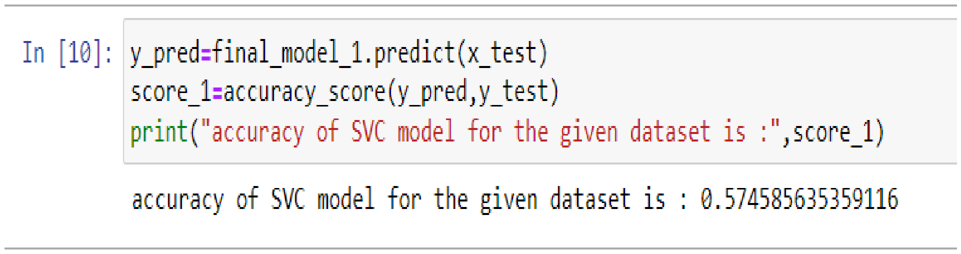
➢ HYPER-PARAMETER TUNING



```
In [9]: from sklearn.model_selection import GridSearchCV
        param_grid=dict()
        param_grid['kernel']=['linear']
        param_grid['C']=[0.1, 1, 10, 100, 1000]
        search=GridSearchCV(svc,param_grid)
        final_model_1=search.fit(x_train,y_train)
        print("best params :",final_model_1.best_params_)
        final_model_1.fit(x_train,y_train)
```

Fig. 3.13: Hyper-parameter tuning of SVC

➢ ACCURACY OF LOGISTIC AFTER HYPER-PARAMETER TUNING

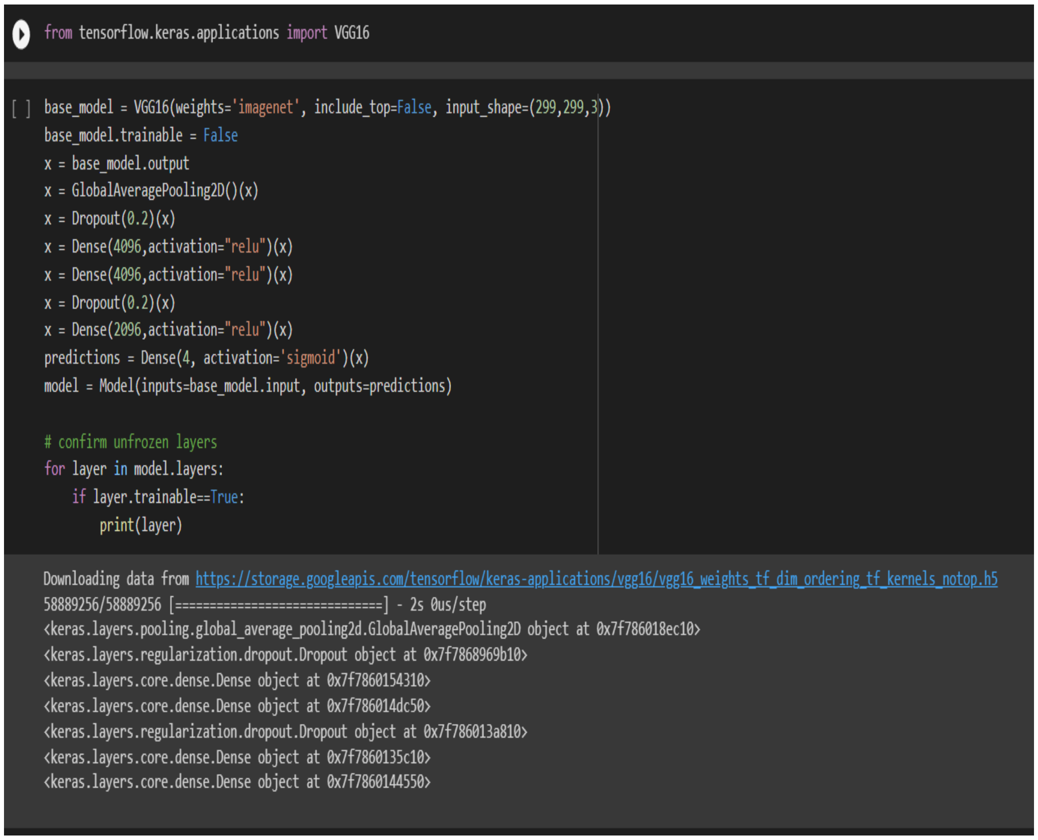


```
In [10]: y_pred=final_model_1.predict(x_test)
         score_1=accuracy_score(y_pred,y_test)
         print("accuracy of SVC model for the given dataset is :",score_1)

         accuracy of SVC model for the given dataset is : 0.574585635359116
```

Fig. 3.14: Accuracy of SVC after Hyper-parameter tuning

▪ **IMPLEMENTATION OF VGG-16 MODEL**

➢ IMPORTING AND IMPLEMENTING VGG-16

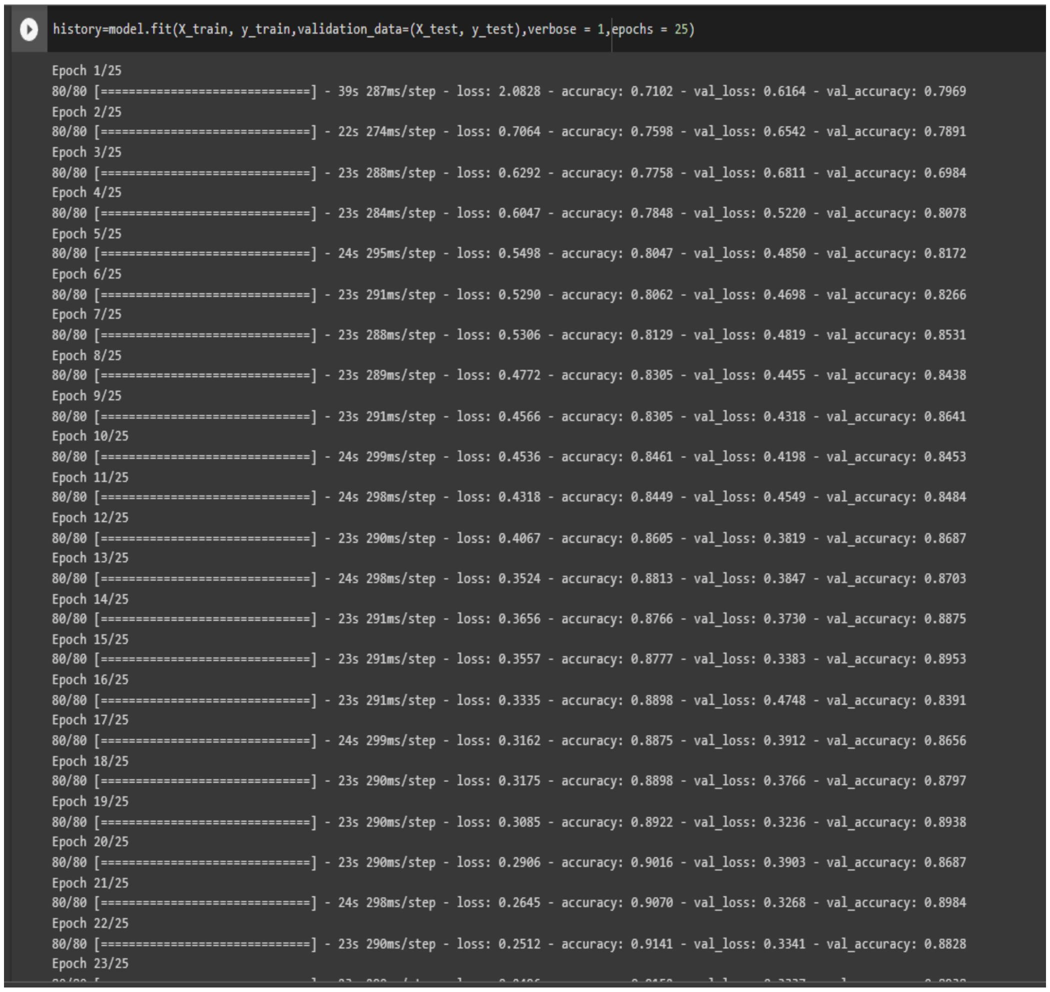


```
from tensorflow.keras.applications import VGG16

[ ] base_model = VGG16(weights='imagenet', include_top=False, input_shape=(299,299,3))
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(4096,activation="relu")(x)
    x = Dense(4096,activation="relu")(x)
    x = Dropout(0.2)(x)
    x = Dense(2096,activation="relu")(x)
    predictions = Dense(4, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # confirm unfrozen layers
    for layer in model.layers:
        if layer.trainable==True:
            print(layer)

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
    58889256/58889256 [==============================] - 2s 0us/step
    <keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object at 0x7f786018ec10>
    <keras.layers.regularization.dropout.Dropout object at 0x7f7868969b10>
    <keras.layers.core.dense.Dense object at 0x7f7860154310>
    <keras.layers.core.dense.Dense object at 0x7f786014dc50>
    <keras.layers.regularization.dropout.Dropout object at 0x7f786013a810>
    <keras.layers.core.dense.Dense object at 0x7f7860135c10>
    <keras.layers.core.dense.Dense object at 0x7f7860144550>
```

Fig. 3.15: Importing and Implementing VGG-16

Importing VGG16 from Keras library and implementing model using ReLU Activation function with an input size of 299 x 299 x 3.

➢ FINDING EPOCH VALUE



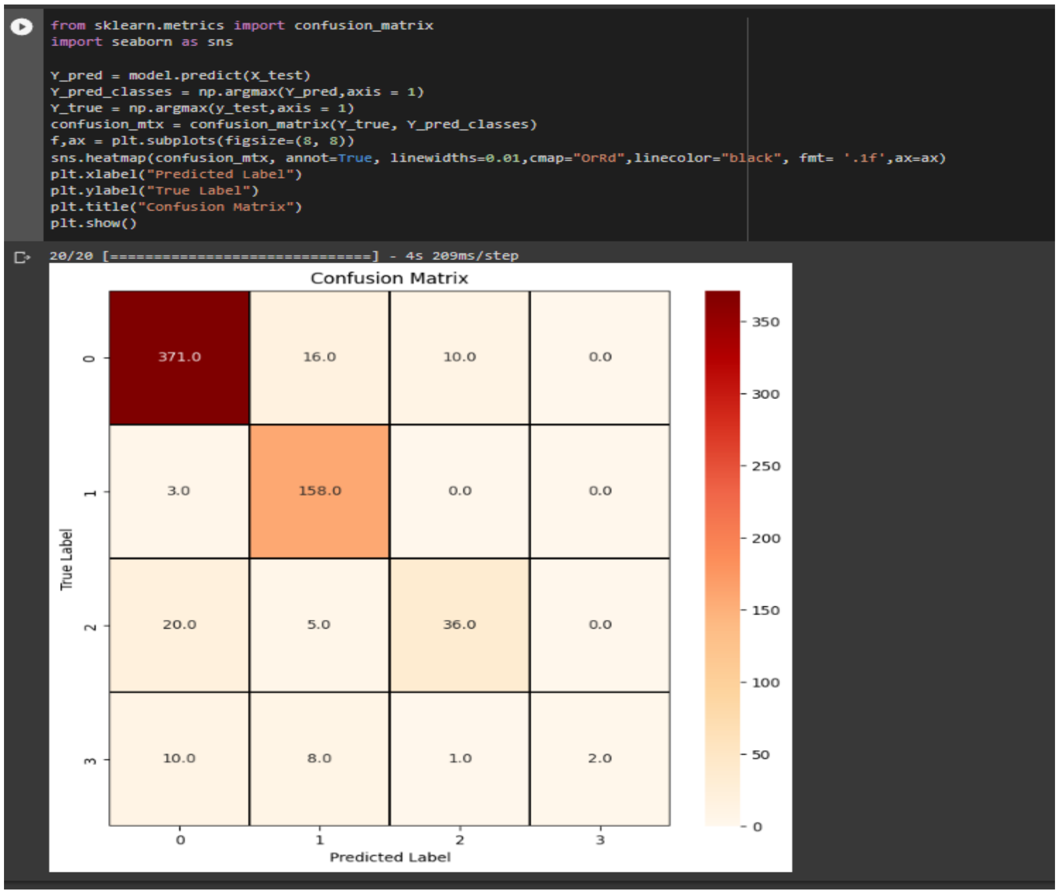
```
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 25)

Epoch 1/25
80/80 [==============================] - 39s 287ms/step - loss: 2.0828 - accuracy: 0.7102 - val_loss: 0.6164 - val_accuracy: 0.7969
Epoch 2/25
80/80 [==============================] - 22s 274ms/step - loss: 0.7064 - accuracy: 0.7598 - val_loss: 0.6542 - val_accuracy: 0.7891
Epoch 3/25
80/80 [==============================] - 23s 288ms/step - loss: 0.6292 - accuracy: 0.7758 - val_loss: 0.6811 - val_accuracy: 0.6984
Epoch 4/25
80/80 [==============================] - 23s 284ms/step - loss: 0.6047 - accuracy: 0.7848 - val_loss: 0.5220 - val_accuracy: 0.8078
Epoch 5/25
80/80 [==============================] - 24s 295ms/step - loss: 0.5498 - accuracy: 0.8047 - val_loss: 0.4850 - val_accuracy: 0.8172
Epoch 6/25
80/80 [==============================] - 23s 291ms/step - loss: 0.5290 - accuracy: 0.8062 - val_loss: 0.4698 - val_accuracy: 0.8266
Epoch 7/25
80/80 [==============================] - 23s 288ms/step - loss: 0.5306 - accuracy: 0.8129 - val_loss: 0.4819 - val_accuracy: 0.8531
Epoch 8/25
80/80 [==============================] - 23s 289ms/step - loss: 0.4772 - accuracy: 0.8305 - val_loss: 0.4455 - val_accuracy: 0.8438
Epoch 9/25
80/80 [==============================] - 23s 291ms/step - loss: 0.4566 - accuracy: 0.8305 - val_loss: 0.4318 - val_accuracy: 0.8641
Epoch 10/25
80/80 [==============================] - 24s 299ms/step - loss: 0.4536 - accuracy: 0.8461 - val_loss: 0.4198 - val_accuracy: 0.8453
Epoch 11/25
80/80 [==============================] - 24s 298ms/step - loss: 0.4318 - accuracy: 0.8449 - val_loss: 0.4549 - val_accuracy: 0.8484
Epoch 12/25
80/80 [==============================] - 23s 290ms/step - loss: 0.4067 - accuracy: 0.8605 - val_loss: 0.3819 - val_accuracy: 0.8687
Epoch 13/25
80/80 [==============================] - 24s 298ms/step - loss: 0.3524 - accuracy: 0.8813 - val_loss: 0.3847 - val_accuracy: 0.8703
Epoch 14/25
80/80 [==============================] - 23s 291ms/step - loss: 0.3656 - accuracy: 0.8766 - val_loss: 0.3730 - val_accuracy: 0.8875
Epoch 15/25
80/80 [==============================] - 23s 291ms/step - loss: 0.3557 - accuracy: 0.8777 - val_loss: 0.3383 - val_accuracy: 0.8953
Epoch 16/25
80/80 [==============================] - 23s 291ms/step - loss: 0.3335 - accuracy: 0.8898 - val_loss: 0.4748 - val_accuracy: 0.8391
Epoch 17/25
80/80 [==============================] - 24s 299ms/step - loss: 0.3162 - accuracy: 0.8875 - val_loss: 0.3912 - val_accuracy: 0.8656
Epoch 18/25
80/80 [==============================] - 23s 290ms/step - loss: 0.3175 - accuracy: 0.8898 - val_loss: 0.3766 - val_accuracy: 0.8797
Epoch 19/25
80/80 [==============================] - 23s 290ms/step - loss: 0.3085 - accuracy: 0.8922 - val_loss: 0.3236 - val_accuracy: 0.8938
Epoch 20/25
80/80 [==============================] - 23s 290ms/step - loss: 0.2906 - accuracy: 0.9016 - val_loss: 0.3903 - val_accuracy: 0.8687
Epoch 21/25
80/80 [==============================] - 24s 298ms/step - loss: 0.2645 - accuracy: 0.9070 - val_loss: 0.3268 - val_accuracy: 0.8984
Epoch 22/25
80/80 [==============================] - 23s 290ms/step - loss: 0.2512 - accuracy: 0.9141 - val_loss: 0.3341 - val_accuracy: 0.8828
Epoch 23/25
```

Fig. 3.16: Finding Epoch Value

Overfitting is a major issue when neural networks are trained using sample data. A neural network model will learn very precise patterns in the sample data if more training epochs are employed than are necessary. As a result, the model won't fit the new data set properly. On the training data set (sample data), this model achieves great accuracy; however, on the test data set, it does not. In other words, by overfitting the training set, the model loses its capacity to generalize.

The highest epoch value achieved in VGG-16 model is 0.8953 at epoch value = 15.

➢ CONFUSION MATRIX OF VGG-16 MODEL

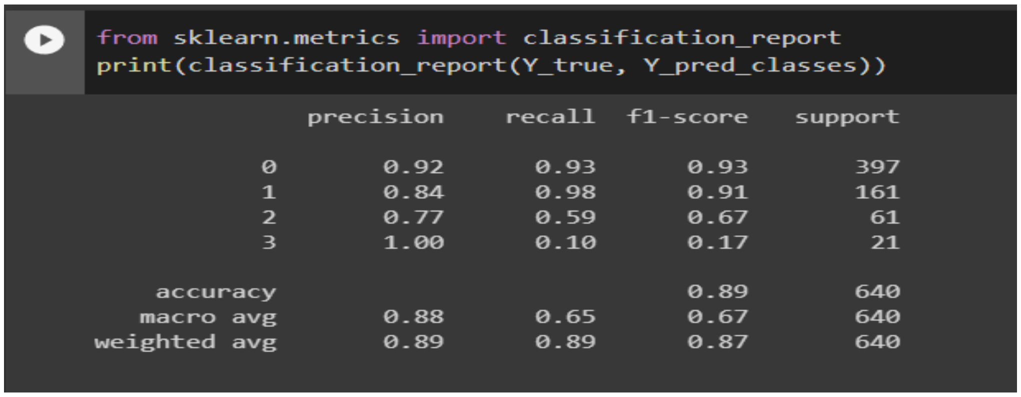


```
from sklearn.metrics import confusion_matrix
import seaborn as sns

Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(y_test,axis = 1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="OrRd",linecolor="black", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
20/20 [==============================] - 4s 209ms/step
```

Fig. 3.17: Confusion Matrix for VGG-16

When the result might potentially comprise two or more classes, it is a technique to evaluate how effectively a machine learning categorization system works. In the table, there are four different potential combinations of predicted and actual values. This program may be used very well to assess AUC-ROC curves, recall, precision, specificity, accuracy, and—most importantly—all of the above.



```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

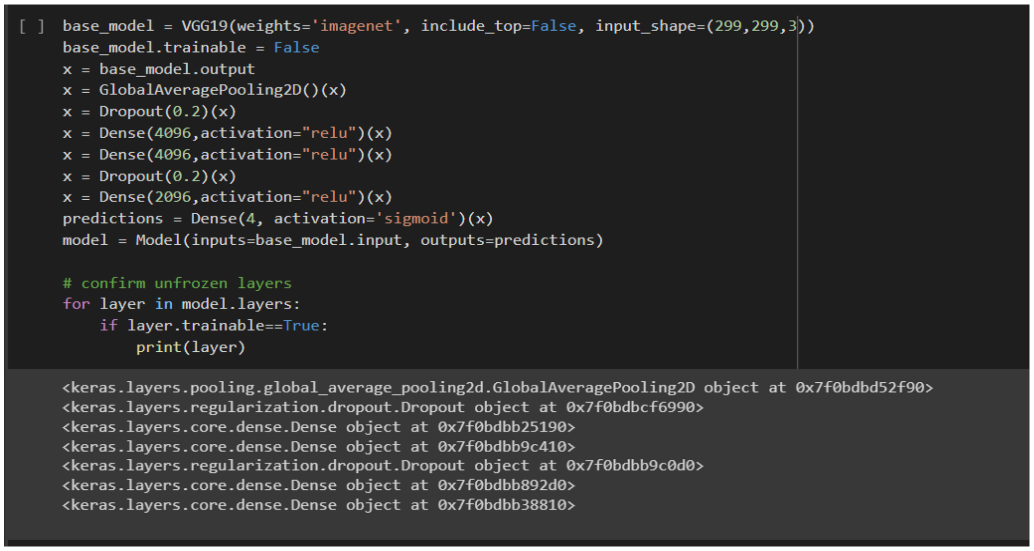
```
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       397
           1       0.84      0.98      0.91       161
           2       0.77      0.59      0.67        61
           3       1.00      0.10      0.17        21

    accuracy                           0.89       640
   macro avg       0.88      0.65      0.67       640
weighted avg       0.89      0.89      0.87       640
```

Fig. 3.18: Classification Report for VGG-16

The Accuracy achieved by the VGG-16 model is 89%.

▪ IMPLEMENTATION OF VGG-19 MODEL

➢ APPLYING RELU ACTIVATION FUNCTION:



```
[ ] base_model = VGG19(weights='imagenet', include_top=False, input_shape=(299,299,3))
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(4096,activation="relu")(x)
    x = Dense(4096,activation="relu")(x)
    x = Dropout(0.2)(x)
    x = Dense(2096,activation="relu")(x)
    predictions = Dense(4, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # confirm unfrozen layers
    for layer in model.layers:
        if layer.trainable==True:
            print(layer)
```

```
<keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object at 0x7f0bdbd52f90>
<keras.layers.regularization.dropout.Dropout object at 0x7f0bdbcf6990>
<keras.layers.core.dense.Dense object at 0x7f0bdbb25190>
<keras.layers.core.dense.Dense object at 0x7f0bdbb9c410>
<keras.layers.regularization.dropout.Dropout object at 0x7f0bdbb9c0d0>
<keras.layers.core.dense.Dense object at 0x7f0bdbb892d0>
<keras.layers.core.dense.Dense object at 0x7f0bdbb38810>
```

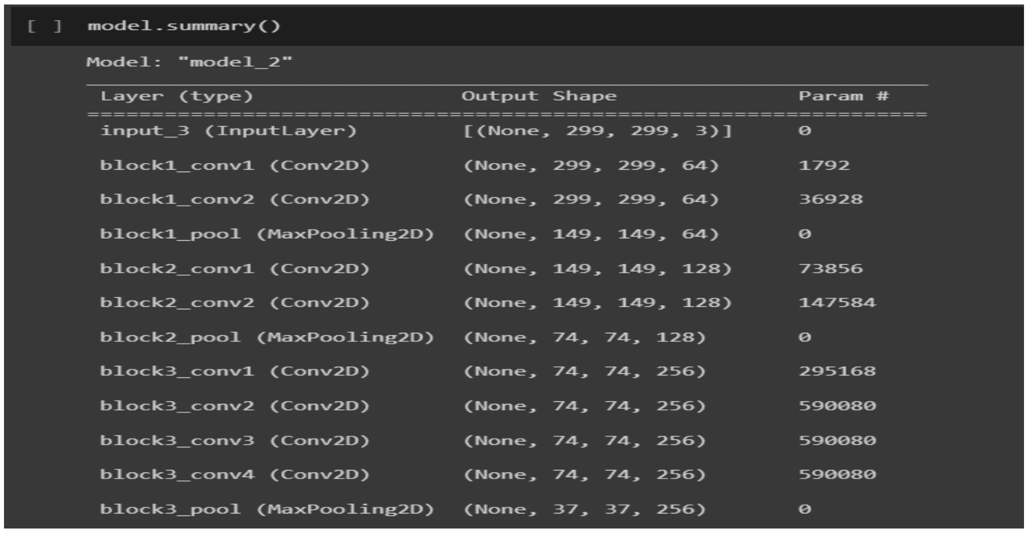
Fig. 3.19: Importing and Implementation of VGG-19

In a neural network, the activation function is responsible for activating the node or output for each input by transforming the node's cumulative weighted input. The rectified linear function, often known as ReLU, will output its input in its entirety if indeed the input is positive; otherwise, it will produce zero. It has developed into the common activation function for several various types of neural networks since a model that employs it is easier to train and commonly outperforms others. In required to practice deep neural networks using gradient descent with training algorithm of errors, an activation function that looks and acts like a linear model but is actually a nonlinear function is needed. The function must also be more adaptive to the input of the activation sum and prevent simple exhaustion. The adoption of ReLU, such as the techniques that today make it feasible to generate very deep neural networks on even a regular basis, may be considered as one of the few turning moments in the deep learning revolution. When creating most types of neural networks, the rectified linear activation function has quickly taken over as the standard activation function.

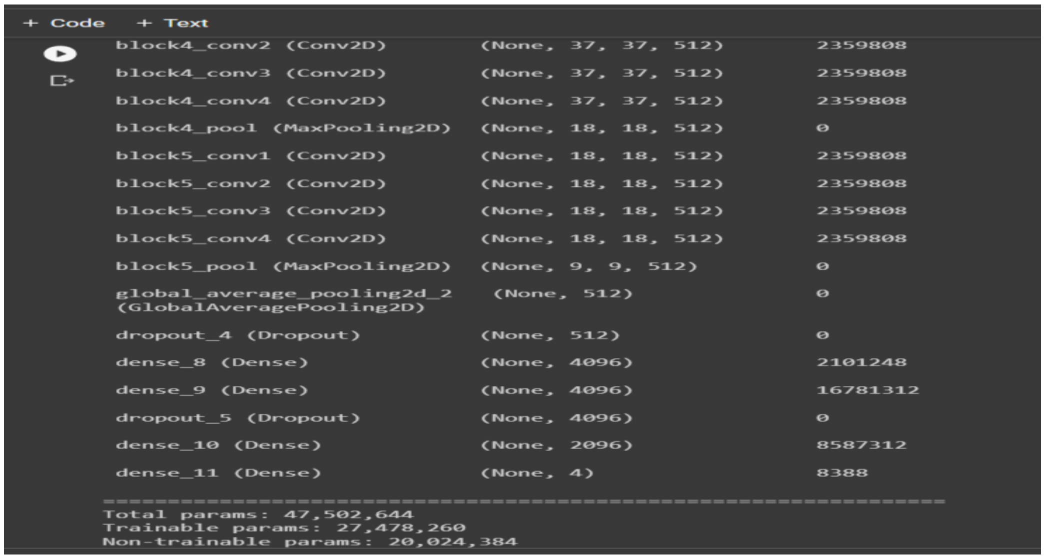
Advantages of ReLU are –

- Computational Simplicity
- Representational Sparsity
- Linear Behavior
- Train Deep Networks

➢ MODEL SUMMARY:



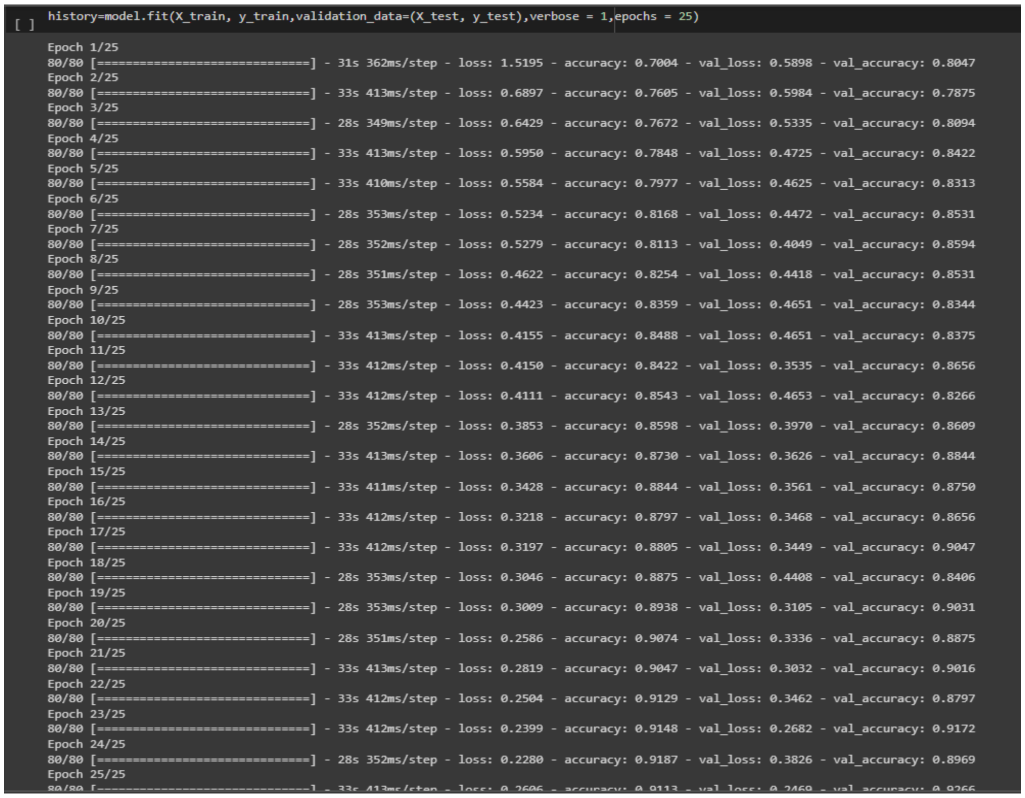
```
[ ]  model.summary()

     Model: "model_2"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      input_3 (InputLayer)        [(None, 299, 299, 3)]     0

      block1_conv1 (Conv2D)       (None, 299, 299, 64)      1792

      block1_conv2 (Conv2D)       (None, 299, 299, 64)      36928

      block1_pool (MaxPooling2D)  (None, 149, 149, 64)      0

      block2_conv1 (Conv2D)       (None, 149, 149, 128)     73856

      block2_conv2 (Conv2D)       (None, 149, 149, 128)     147584

      block2_pool (MaxPooling2D)  (None, 74, 74, 128)       0

      block3_conv1 (Conv2D)       (None, 74, 74, 256)       295168

      block3_conv2 (Conv2D)       (None, 74, 74, 256)       590080

      block3_conv3 (Conv2D)       (None, 74, 74, 256)       590080

      block3_conv4 (Conv2D)       (None, 74, 74, 256)       590080

      block3_pool (MaxPooling2D)  (None, 37, 37, 256)       0
```



```
+ Code    + Text
 block4_conv2 (Conv2D)        (None, 37, 37, 512)       2359808
 block4_conv3 (Conv2D)        (None, 37, 37, 512)       2359808
 block4_conv4 (Conv2D)        (None, 37, 37, 512)       2359808
 block4_pool (MaxPooling2D)   (None, 18, 18, 512)       0
 block5_conv1 (Conv2D)        (None, 18, 18, 512)       2359808
 block5_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
 block5_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
 block5_conv4 (Conv2D)        (None, 18, 18, 512)       2359808
 block5_pool (MaxPooling2D)   (None, 9, 9, 512)         0
 global_average_pooling2d_2    (None, 512)              0
 (GlobalAveragePooling2D)
 dropout_4 (Dropout)          (None, 512)               0
 dense_8 (Dense)              (None, 4096)              2101248
 dense_9 (Dense)              (None, 4096)              16781312
 dropout_5 (Dropout)          (None, 4096)              0
 dense_10 (Dense)             (None, 2096)              8587312
 dense_11 (Dense)             (None, 4)                 8388

=================================================================
Total params: 47,502,644
Trainable params: 27,478,260
Non-trainable params: 20,024,384
```

Fig. 3.20: Model Summary

➢ FINDING BEST EPOCH VALUE:



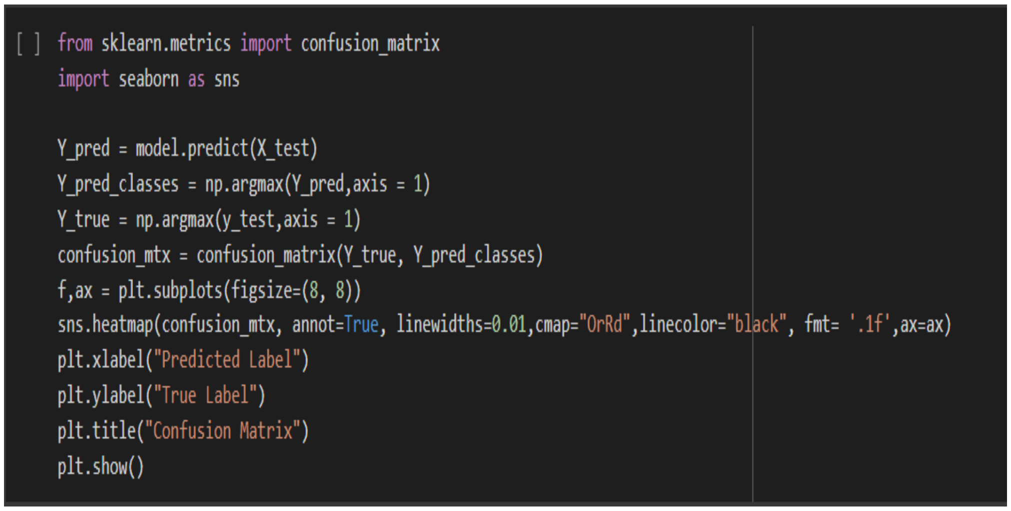
```
[ ]  history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 25)

Epoch 1/25
80/80 [==============================] - 31s 362ms/step - loss: 1.5195 - accuracy: 0.7004 - val_loss: 0.5898 - val_accuracy: 0.8047
Epoch 2/25
80/80 [==============================] - 33s 413ms/step - loss: 0.6897 - accuracy: 0.7605 - val_loss: 0.5984 - val_accuracy: 0.7875
Epoch 3/25
80/80 [==============================] - 28s 349ms/step - loss: 0.6429 - accuracy: 0.7672 - val_loss: 0.5335 - val_accuracy: 0.8094
Epoch 4/25
80/80 [==============================] - 33s 413ms/step - loss: 0.5950 - accuracy: 0.7848 - val_loss: 0.4725 - val_accuracy: 0.8422
Epoch 5/25
80/80 [==============================] - 33s 410ms/step - loss: 0.5584 - accuracy: 0.7977 - val_loss: 0.4625 - val_accuracy: 0.8313
Epoch 6/25
80/80 [==============================] - 28s 353ms/step - loss: 0.5234 - accuracy: 0.8168 - val_loss: 0.4472 - val_accuracy: 0.8531
Epoch 7/25
80/80 [==============================] - 28s 352ms/step - loss: 0.5279 - accuracy: 0.8113 - val_loss: 0.4049 - val_accuracy: 0.8594
Epoch 8/25
80/80 [==============================] - 28s 351ms/step - loss: 0.4622 - accuracy: 0.8254 - val_loss: 0.4418 - val_accuracy: 0.8531
Epoch 9/25
80/80 [==============================] - 28s 353ms/step - loss: 0.4423 - accuracy: 0.8359 - val_loss: 0.4651 - val_accuracy: 0.8344
Epoch 10/25
80/80 [==============================] - 33s 413ms/step - loss: 0.4155 - accuracy: 0.8488 - val_loss: 0.4651 - val_accuracy: 0.8375
Epoch 11/25
80/80 [==============================] - 33s 412ms/step - loss: 0.4150 - accuracy: 0.8422 - val_loss: 0.3535 - val_accuracy: 0.8656
Epoch 12/25
80/80 [==============================] - 33s 412ms/step - loss: 0.4111 - accuracy: 0.8543 - val_loss: 0.4653 - val_accuracy: 0.8266
Epoch 13/25
80/80 [==============================] - 28s 352ms/step - loss: 0.3853 - accuracy: 0.8598 - val_loss: 0.3970 - val_accuracy: 0.8609
Epoch 14/25
80/80 [==============================] - 33s 413ms/step - loss: 0.3606 - accuracy: 0.8730 - val_loss: 0.3626 - val_accuracy: 0.8844
Epoch 15/25
80/80 [==============================] - 33s 411ms/step - loss: 0.3428 - accuracy: 0.8844 - val_loss: 0.3561 - val_accuracy: 0.8750
Epoch 16/25
80/80 [==============================] - 33s 412ms/step - loss: 0.3218 - accuracy: 0.8797 - val_loss: 0.3468 - val_accuracy: 0.8656
Epoch 17/25
80/80 [==============================] - 33s 412ms/step - loss: 0.3197 - accuracy: 0.8805 - val_loss: 0.3449 - val_accuracy: 0.9047
Epoch 18/25
80/80 [==============================] - 28s 353ms/step - loss: 0.3046 - accuracy: 0.8875 - val_loss: 0.4408 - val_accuracy: 0.8406
Epoch 19/25
80/80 [==============================] - 28s 353ms/step - loss: 0.3009 - accuracy: 0.8938 - val_loss: 0.3105 - val_accuracy: 0.9031
Epoch 20/25
80/80 [==============================] - 28s 351ms/step - loss: 0.2586 - accuracy: 0.9074 - val_loss: 0.3336 - val_accuracy: 0.8875
Epoch 21/25
80/80 [==============================] - 33s 413ms/step - loss: 0.2819 - accuracy: 0.9047 - val_loss: 0.3032 - val_accuracy: 0.9016
Epoch 22/25
80/80 [==============================] - 33s 412ms/step - loss: 0.2504 - accuracy: 0.9129 - val_loss: 0.3462 - val_accuracy: 0.8797
Epoch 23/25
80/80 [==============================] - 33s 412ms/step - loss: 0.2399 - accuracy: 0.9148 - val_loss: 0.2682 - val_accuracy: 0.9172
Epoch 24/25
80/80 [==============================] - 28s 352ms/step - loss: 0.2280 - accuracy: 0.9187 - val_loss: 0.3826 - val_accuracy: 0.8969
Epoch 25/25
80/80 [==============================] - 33s 413ms/step - loss: 0.2606 - accuracy: 0.9113 - val_loss: 0.2469 - val_accuracy: 0.9266
```

Fig. 3.21: Accuracy of VGG-19 for total 25 epochs

To reduce overfitting and improve the generalization ability of neural networks, models should be trained with an optimal number of epochs. Validate the model using a portion of the training data and check the performance of the model after each training epoch. The training and validation sets are monitored for loss and accuracy, and the number of epochs before the model starts to overfit is checked.

The highest epoch value achieved in VGG-19 model is 0.9172 at epoch value = 23.

➢ CONFUSION MATRIX:



```
[ ] from sklearn.metrics import confusion_matrix
    import seaborn as sns

    Y_pred = model.predict(X_test)
    Y_pred_classes = np.argmax(Y_pred,axis = 1)
    Y_true = np.argmax(y_test,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    f,ax = plt.subplots(figsize=(8, 8))
    sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="OrRd",linecolor="black", fmt= '.1f',ax=ax)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()
```

Fig. 3.22: Confusion Matrix Code Snippet for VGG-19

It might be difficult to compare two models with great recall but low accuracy. To compare them, we thus use F-Score. The F-score assists in concurrently assessing recall and accuracy. It uses the harmonic mean instead of the arithmetic mean by punishing the outliers more severely.

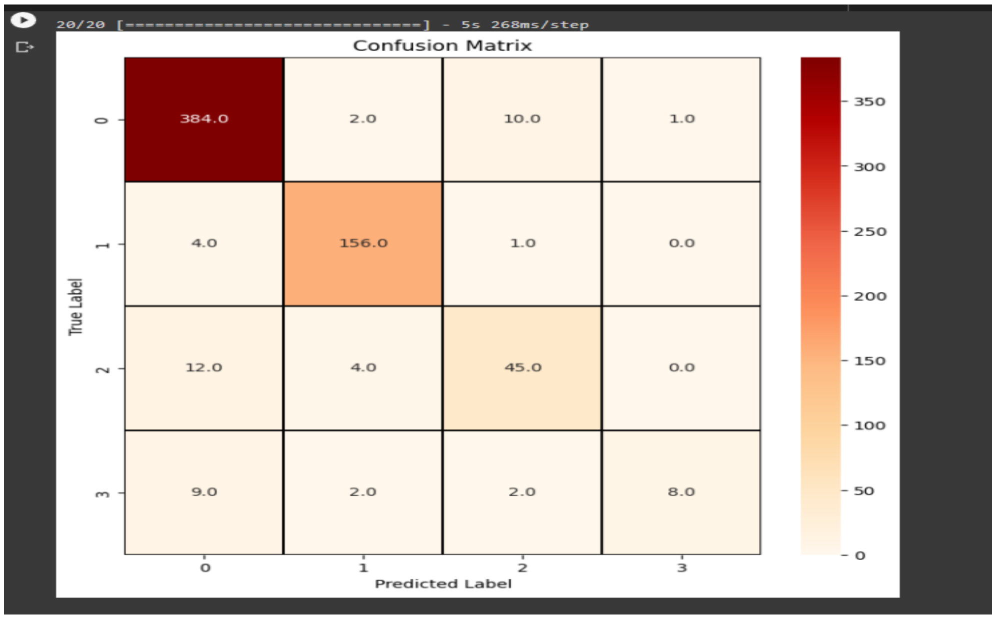
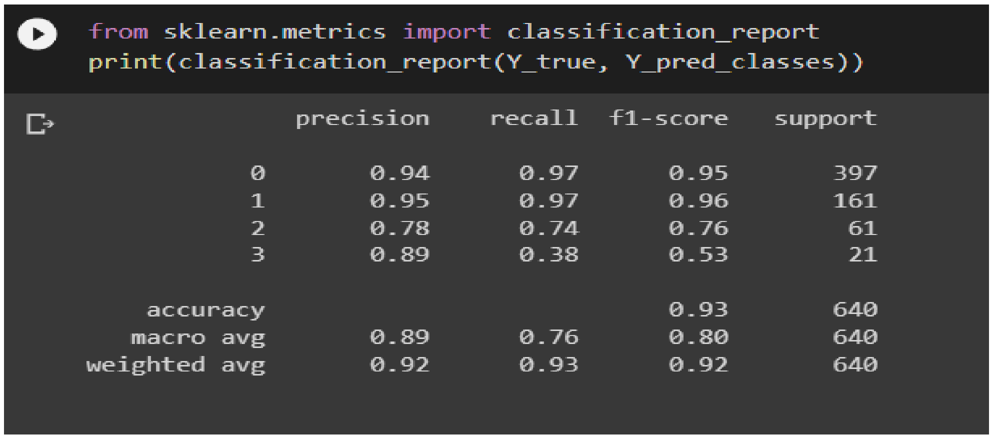


Fig. 3.23: Confusion Matrix for VGG-19


```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       397
           1       0.95      0.97      0.96       161
           2       0.78      0.74      0.76        61
           3       0.89      0.38      0.53        21

    accuracy                           0.93       640
   macro avg       0.89      0.76      0.80       640
weighted avg       0.92      0.93      0.92       640
```
Fig. 3.24: Classification Report for VGG-19

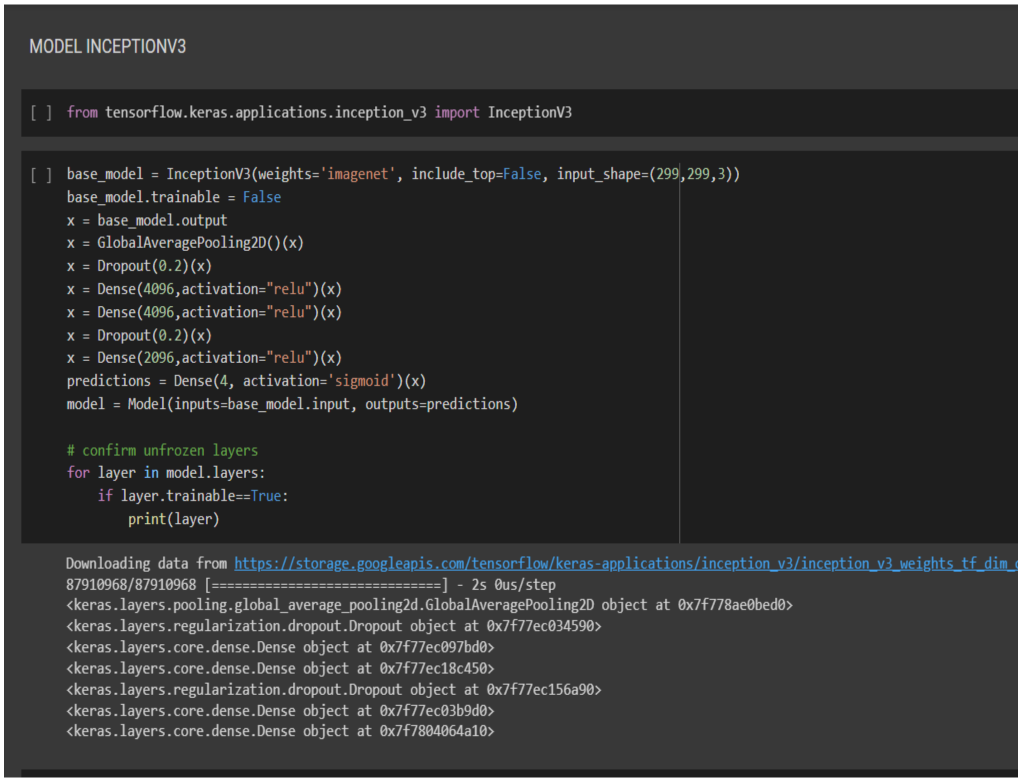
For convolutional neural networks (CNNs), the confusion matrix (Figure 3.13) shows where the model gets confused: which classes it predicts correctly and which classes it predicts incorrectly.

A method for summarizing the classification algorithm's performance is the confusion matrix. When more than two classes are present in the data set or when there are not an equal number of observations in each class, classification report accuracy alone can be deceiving. We gain a better grasp of the categorization model's strengths and weaknesses by computing the confusion matrix.

The Accuracy achieved by the VGG-19 model is 93%.

▪ IMPLEMENTATION OF INCEPTION V3 MODEL

➢ IMPORTING AND IMPLEMENTING INCEPTION V3



MODEL INCEPTIONV3

```
[ ] from tensorflow.keras.applications.inception_v3 import InceptionV3
```

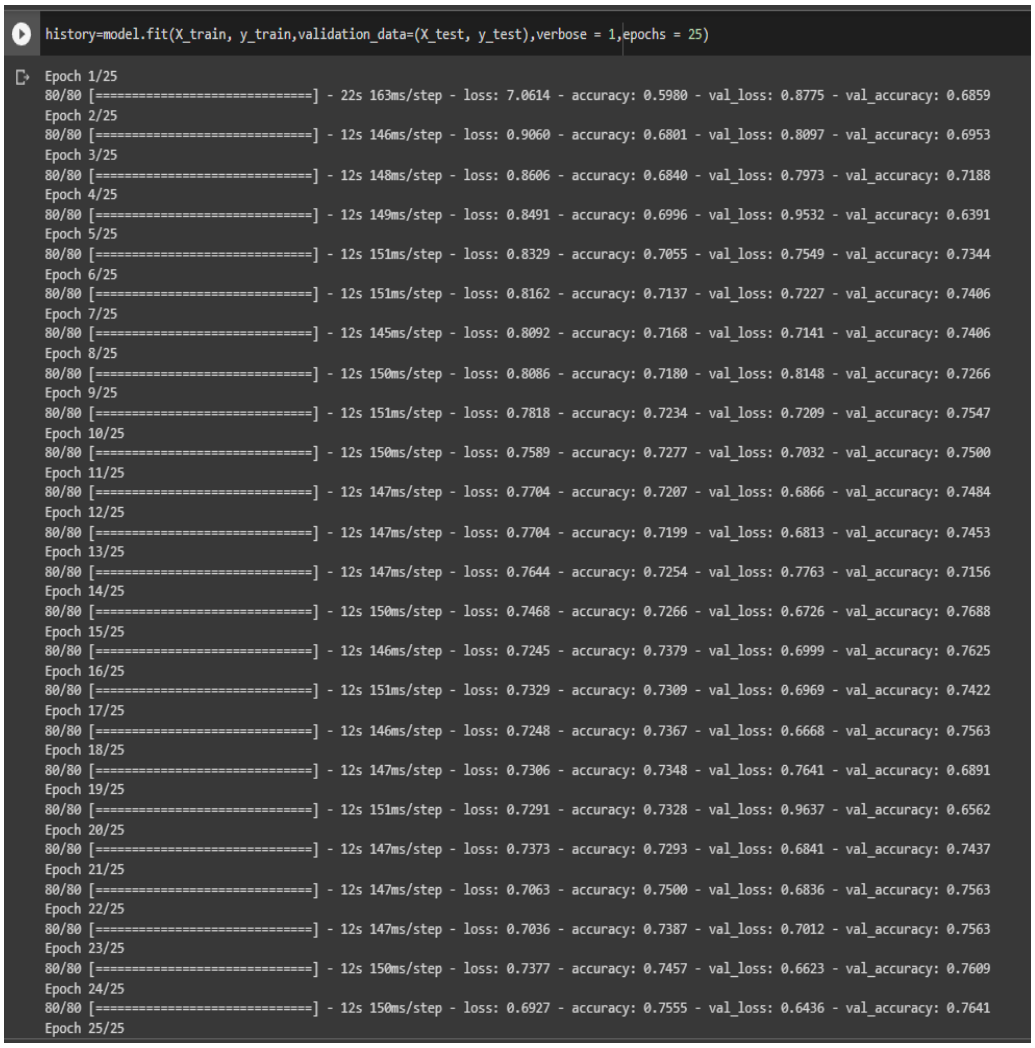
```
[ ] base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299,299,3))
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(4096,activation="relu")(x)
    x = Dense(4096,activation="relu")(x)
    x = Dropout(0.2)(x)
    x = Dense(2096,activation="relu")(x)
    predictions = Dense(4, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # confirm unfrozen layers
    for layer in model.layers:
        if layer.trainable==True:
            print(layer)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_
87910968/87910968 [==============================] - 2s 0us/step
<keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object at 0x7f778ae0bed0>
<keras.layers.regularization.dropout.Dropout object at 0x7f77ec034590>
<keras.layers.core.dense.Dense object at 0x7f77ec097bd0>
<keras.layers.core.dense.Dense object at 0x7f77ec18c450>
<keras.layers.regularization.dropout.Dropout object at 0x7f77ec156a90>
<keras.layers.core.dense.Dense object at 0x7f77ec03b9d0>
<keras.layers.core.dense.Dense object at 0x7f7804064a10>
```

Fig. 3.25: Importing and Implementation of Inception V3

Importing Inception V3 from Keras library and implementing model using ReLU Activation function with an input size of 299 x 299 x 3.

➢ FINDING EPOCH VALUE



```
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 25)

Epoch 1/25
80/80 [==============================] - 22s 163ms/step - loss: 7.0614 - accuracy: 0.5980 - val_loss: 0.8775 - val_accuracy: 0.6859
Epoch 2/25
80/80 [==============================] - 12s 146ms/step - loss: 0.9060 - accuracy: 0.6801 - val_loss: 0.8097 - val_accuracy: 0.6953
Epoch 3/25
80/80 [==============================] - 12s 148ms/step - loss: 0.8606 - accuracy: 0.6840 - val_loss: 0.7973 - val_accuracy: 0.7188
Epoch 4/25
80/80 [==============================] - 12s 149ms/step - loss: 0.8491 - accuracy: 0.6996 - val_loss: 0.9532 - val_accuracy: 0.6391
Epoch 5/25
80/80 [==============================] - 12s 151ms/step - loss: 0.8329 - accuracy: 0.7055 - val_loss: 0.7549 - val_accuracy: 0.7344
Epoch 6/25
80/80 [==============================] - 12s 151ms/step - loss: 0.8162 - accuracy: 0.7137 - val_loss: 0.7227 - val_accuracy: 0.7406
Epoch 7/25
80/80 [==============================] - 12s 145ms/step - loss: 0.8092 - accuracy: 0.7168 - val_loss: 0.7141 - val_accuracy: 0.7406
Epoch 8/25
80/80 [==============================] - 12s 150ms/step - loss: 0.8086 - accuracy: 0.7180 - val_loss: 0.8148 - val_accuracy: 0.7266
Epoch 9/25
80/80 [==============================] - 12s 151ms/step - loss: 0.7818 - accuracy: 0.7234 - val_loss: 0.7209 - val_accuracy: 0.7547
Epoch 10/25
80/80 [==============================] - 12s 150ms/step - loss: 0.7589 - accuracy: 0.7277 - val_loss: 0.7032 - val_accuracy: 0.7500
Epoch 11/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7704 - accuracy: 0.7207 - val_loss: 0.6866 - val_accuracy: 0.7484
Epoch 12/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7704 - accuracy: 0.7199 - val_loss: 0.6813 - val_accuracy: 0.7453
Epoch 13/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7644 - accuracy: 0.7254 - val_loss: 0.7763 - val_accuracy: 0.7156
Epoch 14/25
80/80 [==============================] - 12s 150ms/step - loss: 0.7468 - accuracy: 0.7266 - val_loss: 0.6726 - val_accuracy: 0.7688
Epoch 15/25
80/80 [==============================] - 12s 146ms/step - loss: 0.7245 - accuracy: 0.7379 - val_loss: 0.6999 - val_accuracy: 0.7625
Epoch 16/25
80/80 [==============================] - 12s 151ms/step - loss: 0.7329 - accuracy: 0.7309 - val_loss: 0.6969 - val_accuracy: 0.7422
Epoch 17/25
80/80 [==============================] - 12s 146ms/step - loss: 0.7248 - accuracy: 0.7367 - val_loss: 0.6668 - val_accuracy: 0.7563
Epoch 18/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7306 - accuracy: 0.7348 - val_loss: 0.7641 - val_accuracy: 0.6891
Epoch 19/25
80/80 [==============================] - 12s 151ms/step - loss: 0.7291 - accuracy: 0.7328 - val_loss: 0.9637 - val_accuracy: 0.6562
Epoch 20/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7373 - accuracy: 0.7293 - val_loss: 0.6841 - val_accuracy: 0.7437
Epoch 21/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7063 - accuracy: 0.7500 - val_loss: 0.6836 - val_accuracy: 0.7563
Epoch 22/25
80/80 [==============================] - 12s 147ms/step - loss: 0.7036 - accuracy: 0.7387 - val_loss: 0.7012 - val_accuracy: 0.7563
Epoch 23/25
80/80 [==============================] - 12s 150ms/step - loss: 0.7377 - accuracy: 0.7457 - val_loss: 0.6623 - val_accuracy: 0.7609
Epoch 24/25
80/80 [==============================] - 12s 150ms/step - loss: 0.6927 - accuracy: 0.7555 - val_loss: 0.6436 - val_accuracy: 0.7641
Epoch 25/25
```

Fig. 3.26: Finding Epoch Value for Inception V3

The highest epoch value achieved in Inception V3 model is 0.7703 at epoch value = 25.

➢ CONFUSION MATRIX OF INCEPTION V3 MODEL

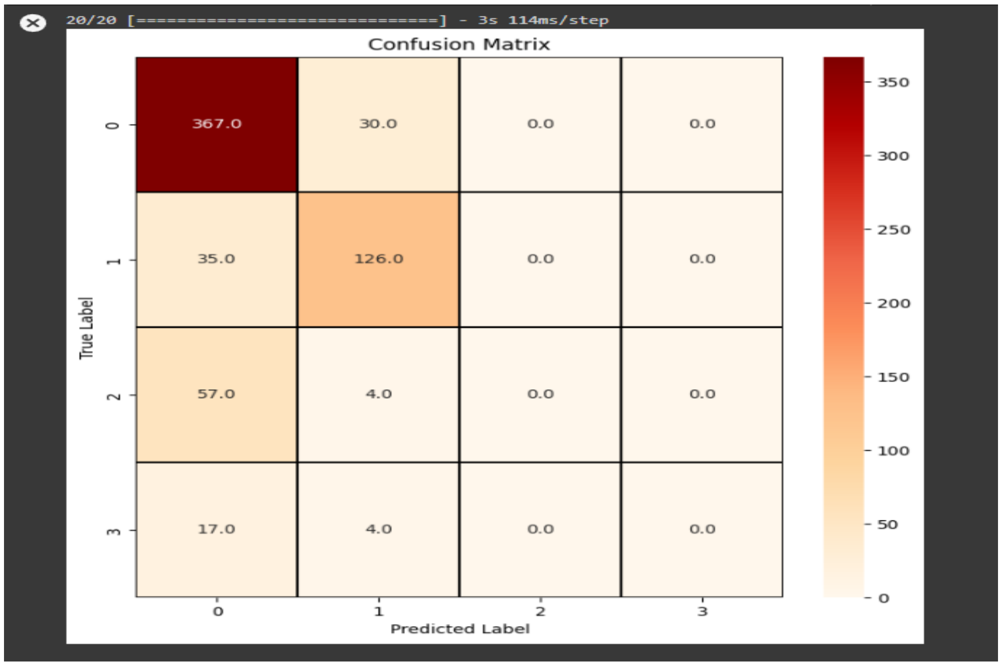
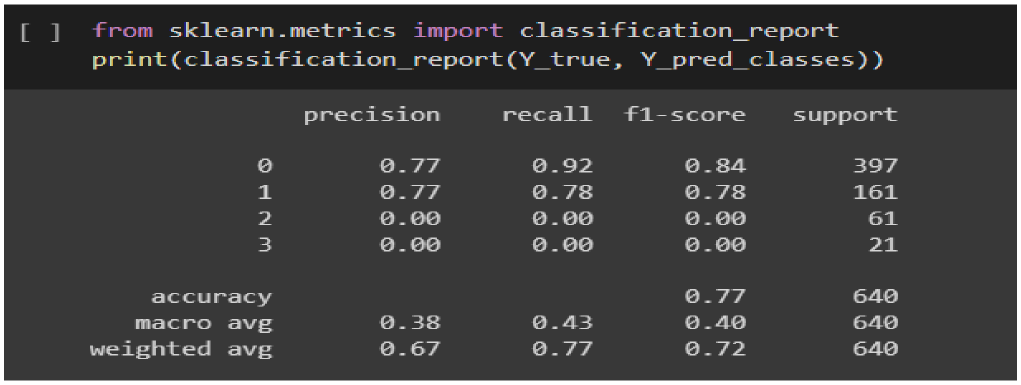


Fig. 3.27: Confusion Matrix of Inception V3



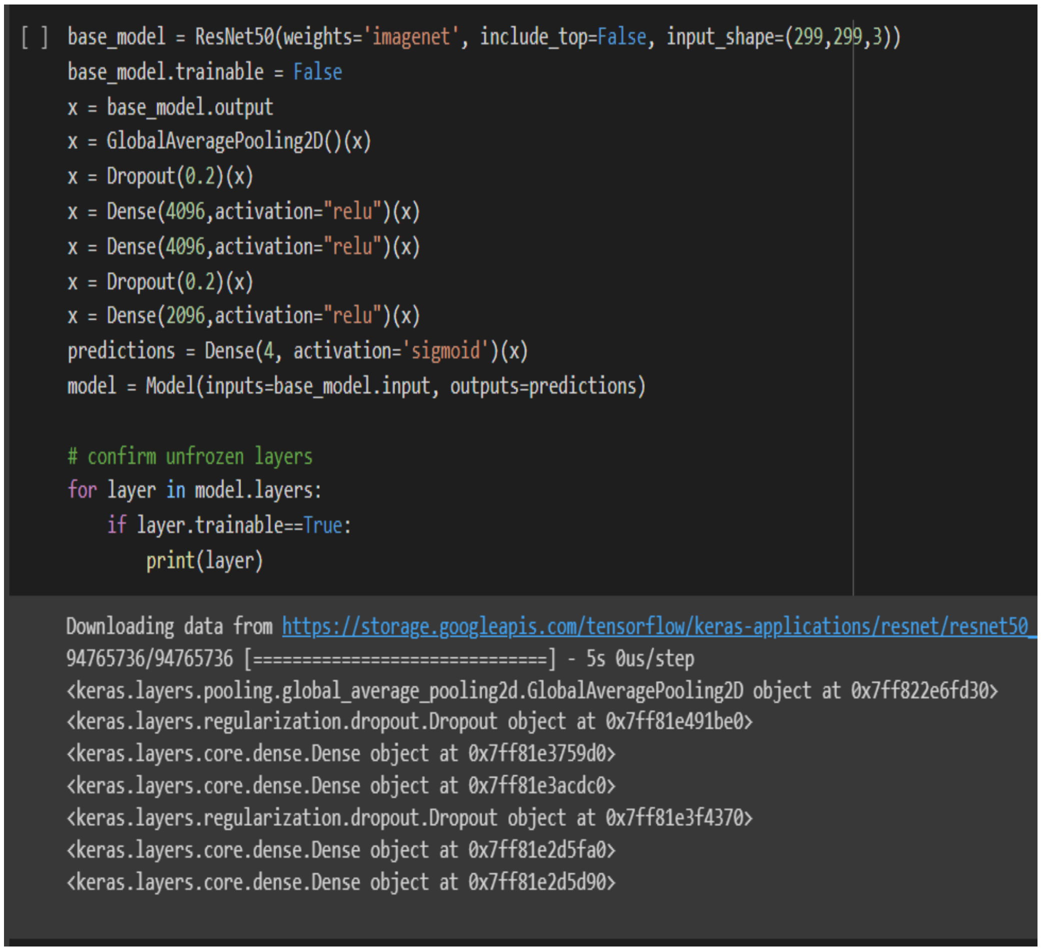
```
[ ]  from sklearn.metrics import classification_report
     print(classification_report(Y_true, Y_pred_classes))

              precision    recall  f1-score   support

           0       0.77      0.92      0.84       397
           1       0.77      0.78      0.78       161
           2       0.00      0.00      0.00        61
           3       0.00      0.00      0.00        21

    accuracy                           0.77       640
   macro avg       0.38      0.43      0.40       640
weighted avg       0.67      0.77      0.72       640
```

Fig. 3.28: Classification report of Inception V3

The Accuracy achieved by the Inception V3 model is 77%

▪ IMPLEMENTATION OF RESNET50 MODEL

➢ IMPORTING AND IMPLEMENTING OF RESNET50

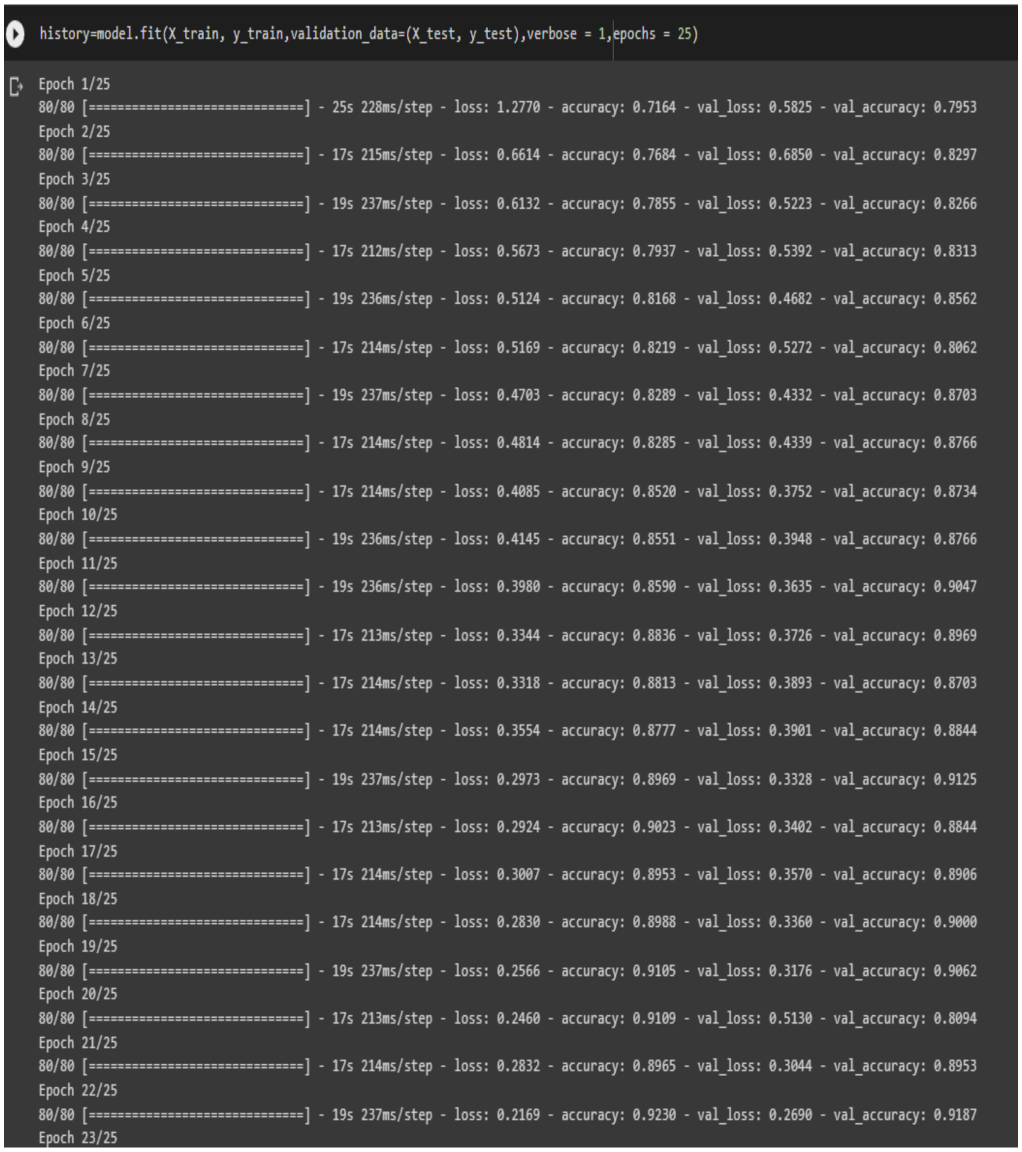


```
[ ] base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(299,299,3))
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(4096,activation="relu")(x)
    x = Dense(4096,activation="relu")(x)
    x = Dropout(0.2)(x)
    x = Dense(2096,activation="relu")(x)
    predictions = Dense(4, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # confirm unfrozen layers
    for layer in model.layers:
        if layer.trainable==True:
            print(layer)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_
94765736/94765736 [==============================] - 5s 0us/step
<keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object at 0x7ff822e6fd30>
<keras.layers.regularization.dropout.Dropout object at 0x7ff81e491be0>
<keras.layers.core.dense.Dense object at 0x7ff81e3759d0>
<keras.layers.core.dense.Dense object at 0x7ff81e3acdc0>
<keras.layers.regularization.dropout.Dropout object at 0x7ff81e3f4370>
<keras.layers.core.dense.Dense object at 0x7ff81e2d5fa0>
<keras.layers.core.dense.Dense object at 0x7ff81e2d5d90>
```

Fig. 3.29: Importing and Implementing of ResNet50

➢ FINDING EPOCH VALUE



```
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 25)
```

```
Epoch 1/25
80/80 [==============================] - 25s 228ms/step - loss: 1.2770 - accuracy: 0.7164 - val_loss: 0.5825 - val_accuracy: 0.7953
Epoch 2/25
80/80 [==============================] - 17s 215ms/step - loss: 0.6614 - accuracy: 0.7684 - val_loss: 0.6850 - val_accuracy: 0.8297
Epoch 3/25
80/80 [==============================] - 19s 237ms/step - loss: 0.6132 - accuracy: 0.7855 - val_loss: 0.5223 - val_accuracy: 0.8266
Epoch 4/25
80/80 [==============================] - 17s 212ms/step - loss: 0.5673 - accuracy: 0.7937 - val_loss: 0.5392 - val_accuracy: 0.8313
Epoch 5/25
80/80 [==============================] - 19s 236ms/step - loss: 0.5124 - accuracy: 0.8168 - val_loss: 0.4682 - val_accuracy: 0.8562
Epoch 6/25
80/80 [==============================] - 17s 214ms/step - loss: 0.5169 - accuracy: 0.8219 - val_loss: 0.5272 - val_accuracy: 0.8062
Epoch 7/25
80/80 [==============================] - 19s 237ms/step - loss: 0.4703 - accuracy: 0.8289 - val_loss: 0.4332 - val_accuracy: 0.8703
Epoch 8/25
80/80 [==============================] - 17s 214ms/step - loss: 0.4814 - accuracy: 0.8285 - val_loss: 0.4339 - val_accuracy: 0.8766
Epoch 9/25
80/80 [==============================] - 17s 214ms/step - loss: 0.4085 - accuracy: 0.8520 - val_loss: 0.3752 - val_accuracy: 0.8734
Epoch 10/25
80/80 [==============================] - 19s 236ms/step - loss: 0.4145 - accuracy: 0.8551 - val_loss: 0.3948 - val_accuracy: 0.8766
Epoch 11/25
80/80 [==============================] - 19s 236ms/step - loss: 0.3980 - accuracy: 0.8590 - val_loss: 0.3635 - val_accuracy: 0.9047
Epoch 12/25
80/80 [==============================] - 17s 213ms/step - loss: 0.3344 - accuracy: 0.8836 - val_loss: 0.3726 - val_accuracy: 0.8969
Epoch 13/25
80/80 [==============================] - 17s 214ms/step - loss: 0.3318 - accuracy: 0.8813 - val_loss: 0.3893 - val_accuracy: 0.8703
Epoch 14/25
80/80 [==============================] - 17s 214ms/step - loss: 0.3554 - accuracy: 0.8777 - val_loss: 0.3901 - val_accuracy: 0.8844
Epoch 15/25
80/80 [==============================] - 19s 237ms/step - loss: 0.2973 - accuracy: 0.8969 - val_loss: 0.3328 - val_accuracy: 0.9125
Epoch 16/25
80/80 [==============================] - 17s 213ms/step - loss: 0.2924 - accuracy: 0.9023 - val_loss: 0.3402 - val_accuracy: 0.8844
Epoch 17/25
80/80 [==============================] - 17s 214ms/step - loss: 0.3007 - accuracy: 0.8953 - val_loss: 0.3570 - val_accuracy: 0.8906
Epoch 18/25
80/80 [==============================] - 17s 214ms/step - loss: 0.2830 - accuracy: 0.8988 - val_loss: 0.3360 - val_accuracy: 0.9000
Epoch 19/25
80/80 [==============================] - 19s 237ms/step - loss: 0.2566 - accuracy: 0.9105 - val_loss: 0.3176 - val_accuracy: 0.9062
Epoch 20/25
80/80 [==============================] - 17s 213ms/step - loss: 0.2460 - accuracy: 0.9109 - val_loss: 0.5130 - val_accuracy: 0.8094
Epoch 21/25
80/80 [==============================] - 17s 214ms/step - loss: 0.2832 - accuracy: 0.8965 - val_loss: 0.3044 - val_accuracy: 0.8953
Epoch 22/25
80/80 [==============================] - 19s 237ms/step - loss: 0.2169 - accuracy: 0.9230 - val_loss: 0.2690 - val_accuracy: 0.9187
Epoch 23/25
```

Fig. 3.30: Evaluating accuracy for 25 epochs.

The highest accuracy achieved is 0.9203 at epoch value 25.

➤ CONFUSION MATRIX OF RESNET50 MODEL

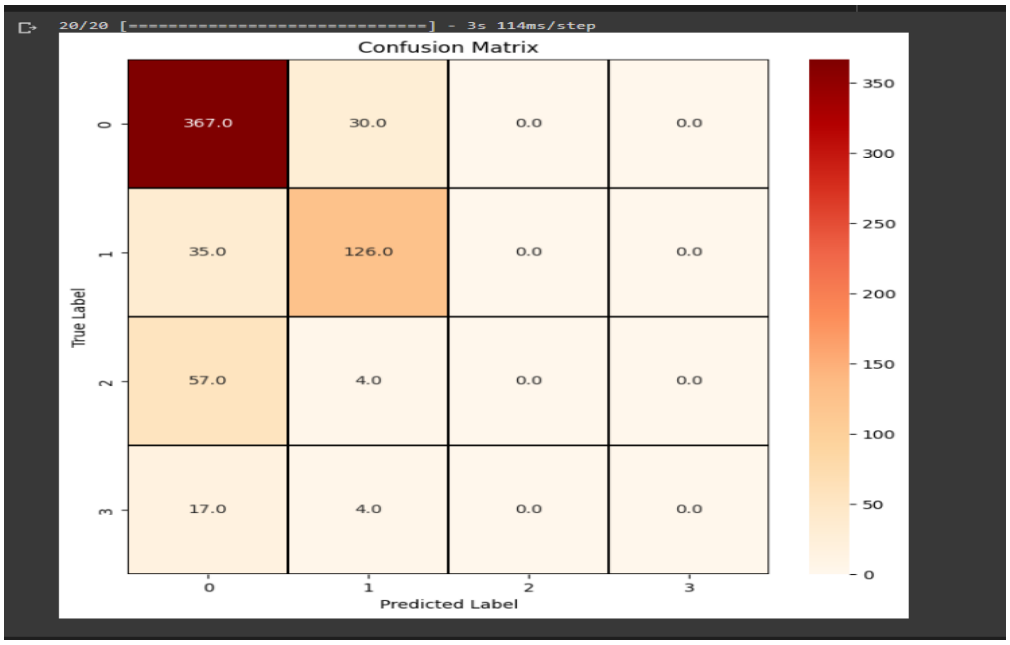
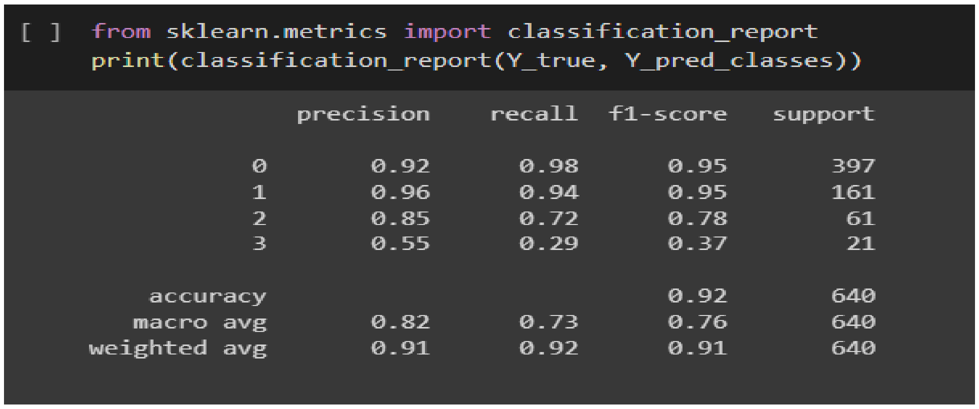


Fig. 3.31: Confusion Matrix for ResNet50

➤ RESULTS



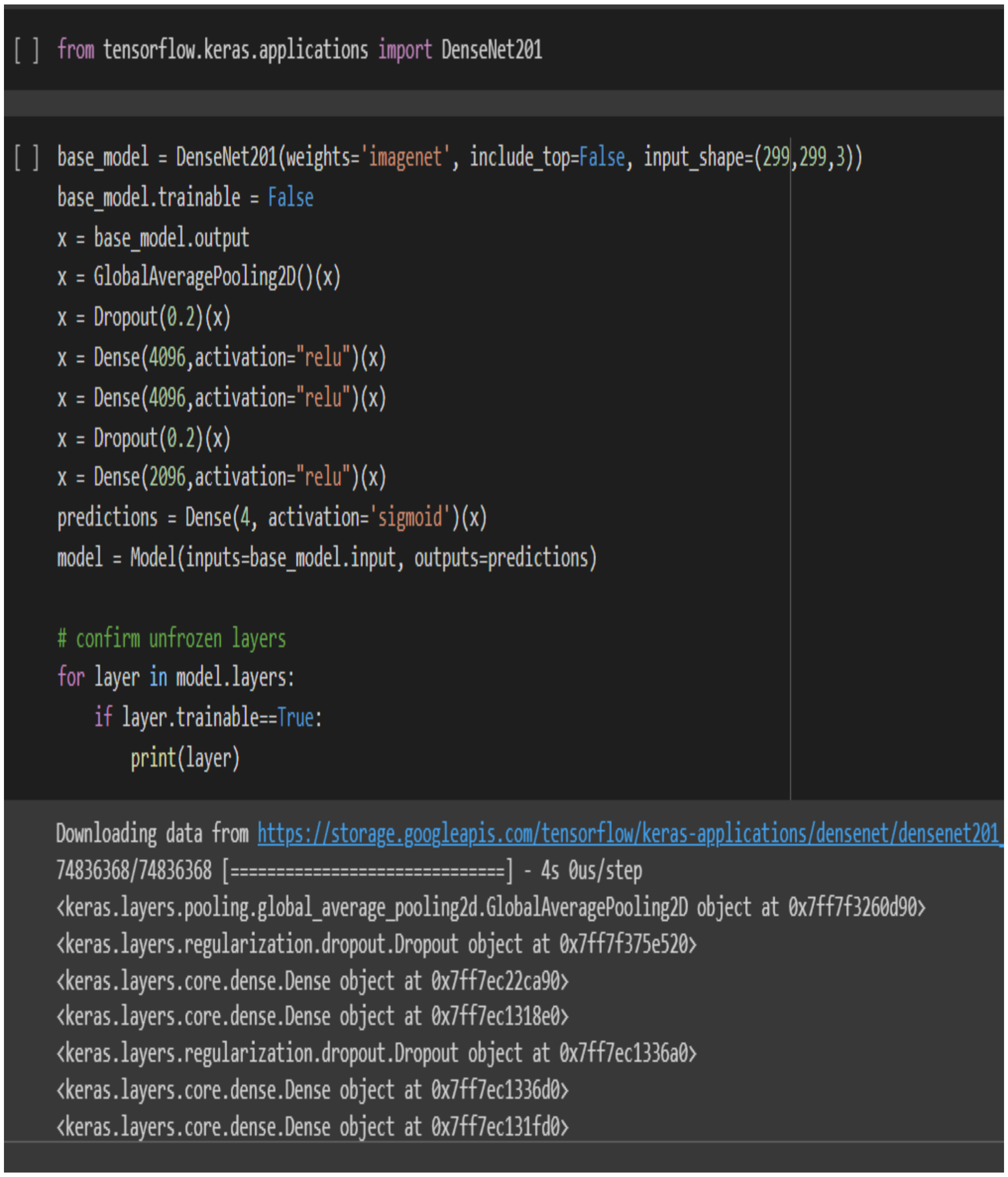
```
[ ]  from sklearn.metrics import classification_report
     print(classification_report(Y_true, Y_pred_classes))

              precision    recall  f1-score   support

           0       0.92      0.98      0.95       397
           1       0.96      0.94      0.95       161
           2       0.85      0.72      0.78        61
           3       0.55      0.29      0.37        21

    accuracy                           0.92       640
   macro avg       0.82      0.73      0.76       640
weighted avg       0.91      0.92      0.91       640
```

Fig. 3.32: Classification report of ResNet50

The accuracy achieved by the ResNet50 model is 92% which is the highest achieved accuracy after VGG-19 model.

- **IMPLEMENTATION OF DENSENET 201 MODEL**

➢ IMPORTING AND IMPLEMENTING OF DENSENET 201



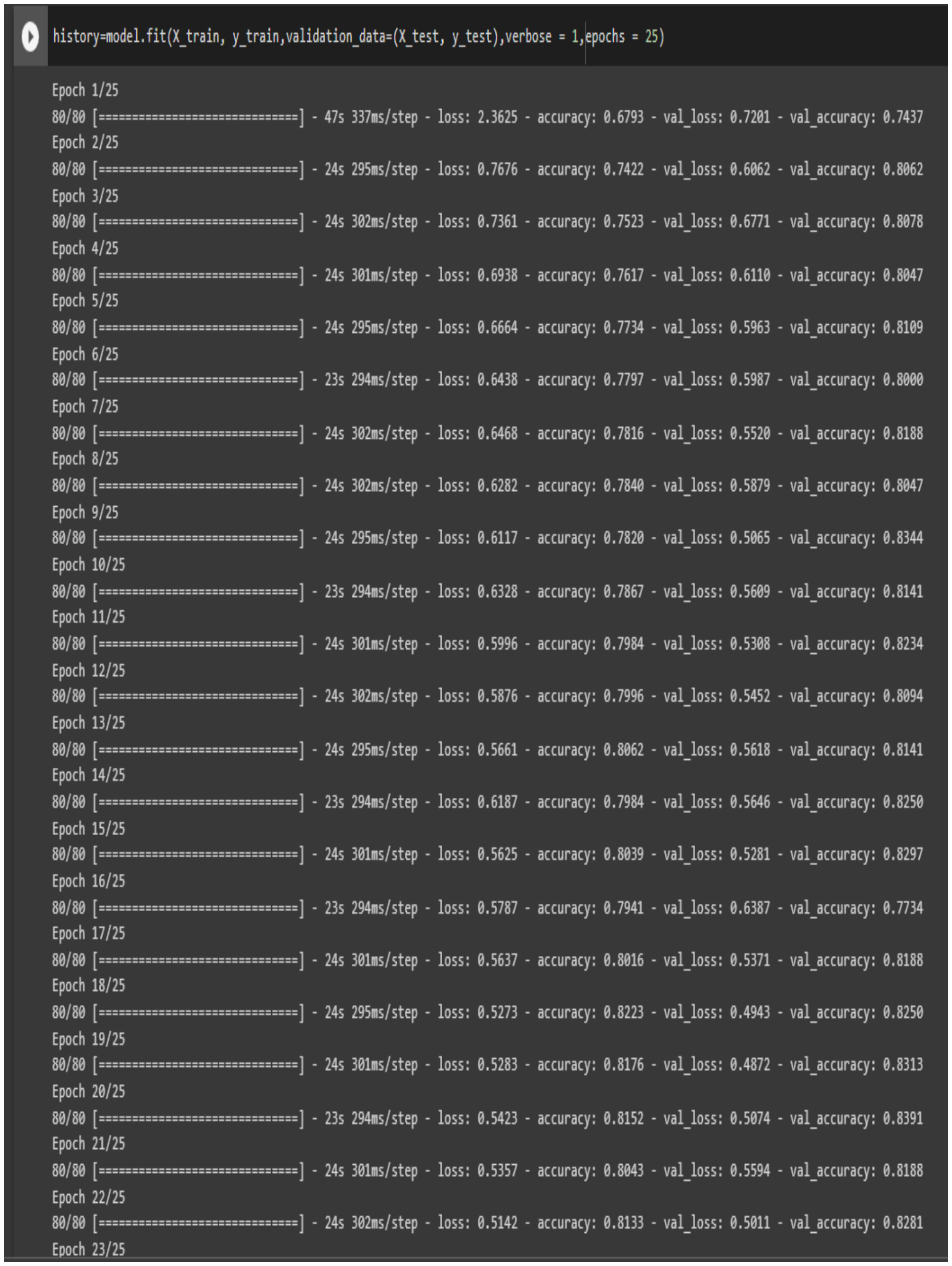
```
[ ] from tensorflow.keras.applications import DenseNet201
```

```
[ ] base_model = DenseNet201(weights='imagenet', include_top=False, input_shape=(299,299,3))
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(4096,activation="relu")(x)
    x = Dense(4096,activation="relu")(x)
    x = Dropout(0.2)(x)
    x = Dense(2096,activation="relu")(x)
    predictions = Dense(4, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # confirm unfrozen layers
    for layer in model.layers:
        if layer.trainable==True:
            print(layer)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201
74836368/74836368 [==============================] - 4s 0us/step
<keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object at 0x7ff7f3260d90>
<keras.layers.regularization.dropout.Dropout object at 0x7ff7f375e520>
<keras.layers.core.dense.Dense object at 0x7ff7ec22ca90>
<keras.layers.core.dense.Dense object at 0x7ff7ec1318e0>
<keras.layers.regularization.dropout.Dropout object at 0x7ff7ec1336a0>
<keras.layers.core.dense.Dense object at 0x7ff7ec1336d0>
<keras.layers.core.dense.Dense object at 0x7ff7ec131fd0>
```

Fig. 3.33: Importing and Implementing DenseNet201 model

➢ FINDING EPOCH VALUE



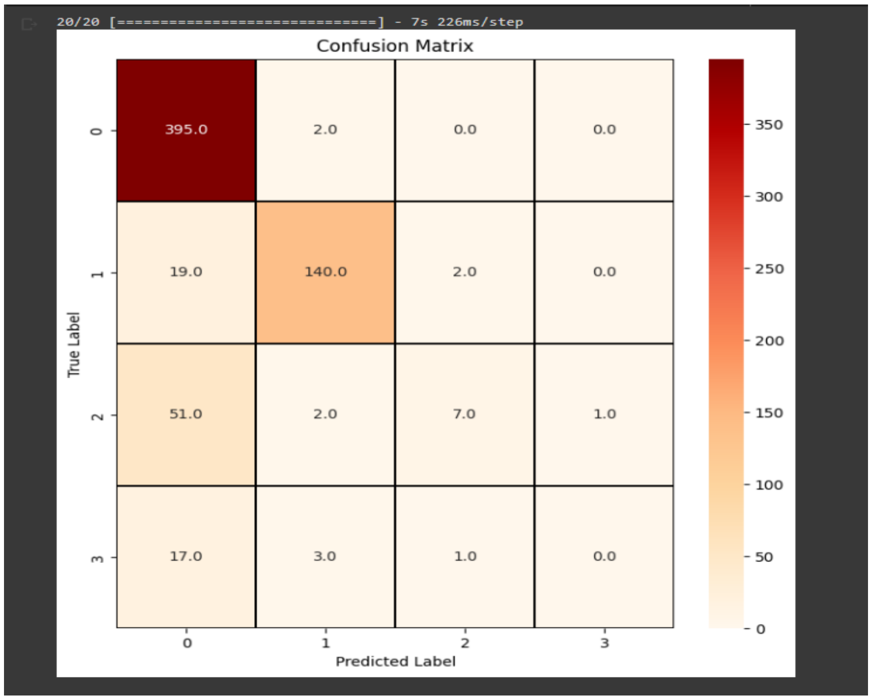
```
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 25)
```

```
Epoch 1/25
80/80 [==============================] - 47s 337ms/step - loss: 2.3625 - accuracy: 0.6793 - val_loss: 0.7201 - val_accuracy: 0.7437
Epoch 2/25
80/80 [==============================] - 24s 295ms/step - loss: 0.7676 - accuracy: 0.7422 - val_loss: 0.6062 - val_accuracy: 0.8062
Epoch 3/25
80/80 [==============================] - 24s 302ms/step - loss: 0.7361 - accuracy: 0.7523 - val_loss: 0.6771 - val_accuracy: 0.8078
Epoch 4/25
80/80 [==============================] - 24s 301ms/step - loss: 0.6938 - accuracy: 0.7617 - val_loss: 0.6110 - val_accuracy: 0.8047
Epoch 5/25
80/80 [==============================] - 24s 295ms/step - loss: 0.6664 - accuracy: 0.7734 - val_loss: 0.5963 - val_accuracy: 0.8109
Epoch 6/25
80/80 [==============================] - 23s 294ms/step - loss: 0.6438 - accuracy: 0.7797 - val_loss: 0.5987 - val_accuracy: 0.8000
Epoch 7/25
80/80 [==============================] - 24s 302ms/step - loss: 0.6468 - accuracy: 0.7816 - val_loss: 0.5520 - val_accuracy: 0.8188
Epoch 8/25
80/80 [==============================] - 24s 302ms/step - loss: 0.6282 - accuracy: 0.7840 - val_loss: 0.5879 - val_accuracy: 0.8047
Epoch 9/25
80/80 [==============================] - 24s 295ms/step - loss: 0.6117 - accuracy: 0.7820 - val_loss: 0.5065 - val_accuracy: 0.8344
Epoch 10/25
80/80 [==============================] - 23s 294ms/step - loss: 0.6328 - accuracy: 0.7867 - val_loss: 0.5609 - val_accuracy: 0.8141
Epoch 11/25
80/80 [==============================] - 24s 301ms/step - loss: 0.5996 - accuracy: 0.7984 - val_loss: 0.5308 - val_accuracy: 0.8234
Epoch 12/25
80/80 [==============================] - 24s 302ms/step - loss: 0.5876 - accuracy: 0.7996 - val_loss: 0.5452 - val_accuracy: 0.8094
Epoch 13/25
80/80 [==============================] - 24s 295ms/step - loss: 0.5661 - accuracy: 0.8062 - val_loss: 0.5618 - val_accuracy: 0.8141
Epoch 14/25
80/80 [==============================] - 23s 294ms/step - loss: 0.6187 - accuracy: 0.7984 - val_loss: 0.5646 - val_accuracy: 0.8250
Epoch 15/25
80/80 [==============================] - 24s 301ms/step - loss: 0.5625 - accuracy: 0.8039 - val_loss: 0.5281 - val_accuracy: 0.8297
Epoch 16/25
80/80 [==============================] - 23s 294ms/step - loss: 0.5787 - accuracy: 0.7941 - val_loss: 0.6387 - val_accuracy: 0.7734
Epoch 17/25
80/80 [==============================] - 24s 301ms/step - loss: 0.5637 - accuracy: 0.8016 - val_loss: 0.5371 - val_accuracy: 0.8188
Epoch 18/25
80/80 [==============================] - 24s 295ms/step - loss: 0.5273 - accuracy: 0.8223 - val_loss: 0.4943 - val_accuracy: 0.8250
Epoch 19/25
80/80 [==============================] - 24s 301ms/step - loss: 0.5283 - accuracy: 0.8176 - val_loss: 0.4872 - val_accuracy: 0.8313
Epoch 20/25
80/80 [==============================] - 23s 294ms/step - loss: 0.5423 - accuracy: 0.8152 - val_loss: 0.5074 - val_accuracy: 0.8391
Epoch 21/25
80/80 [==============================] - 24s 301ms/step - loss: 0.5357 - accuracy: 0.8043 - val_loss: 0.5594 - val_accuracy: 0.8188
Epoch 22/25
80/80 [==============================] - 24s 302ms/step - loss: 0.5142 - accuracy: 0.8133 - val_loss: 0.5011 - val_accuracy: 0.8281
Epoch 23/25
```

Fig. 3.34: Evaluating accuracy for total 25 epochs

The highest accuracy achieved is 0.8469 at epoch value 25.

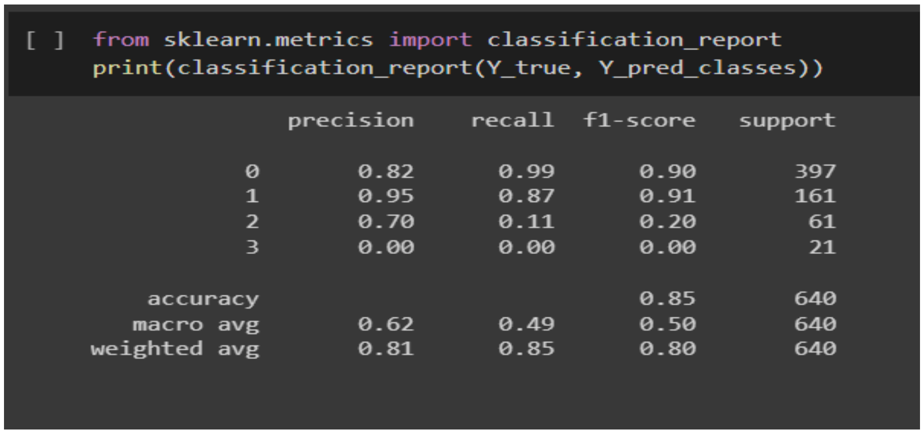
➢ CONFUSION MATRIX OF DENSENET 201 MODEL



```
20/20 [==============================] - 7s 226ms/step
```

	0	1	2	3
0	395.0	2.0	0.0	0.0
1	19.0	140.0	2.0	0.0
2	51.0	2.0	7.0	1.0
3	17.0	3.0	1.0	0.0

Fig.3.35: Confusion Matrix for DenseNet201



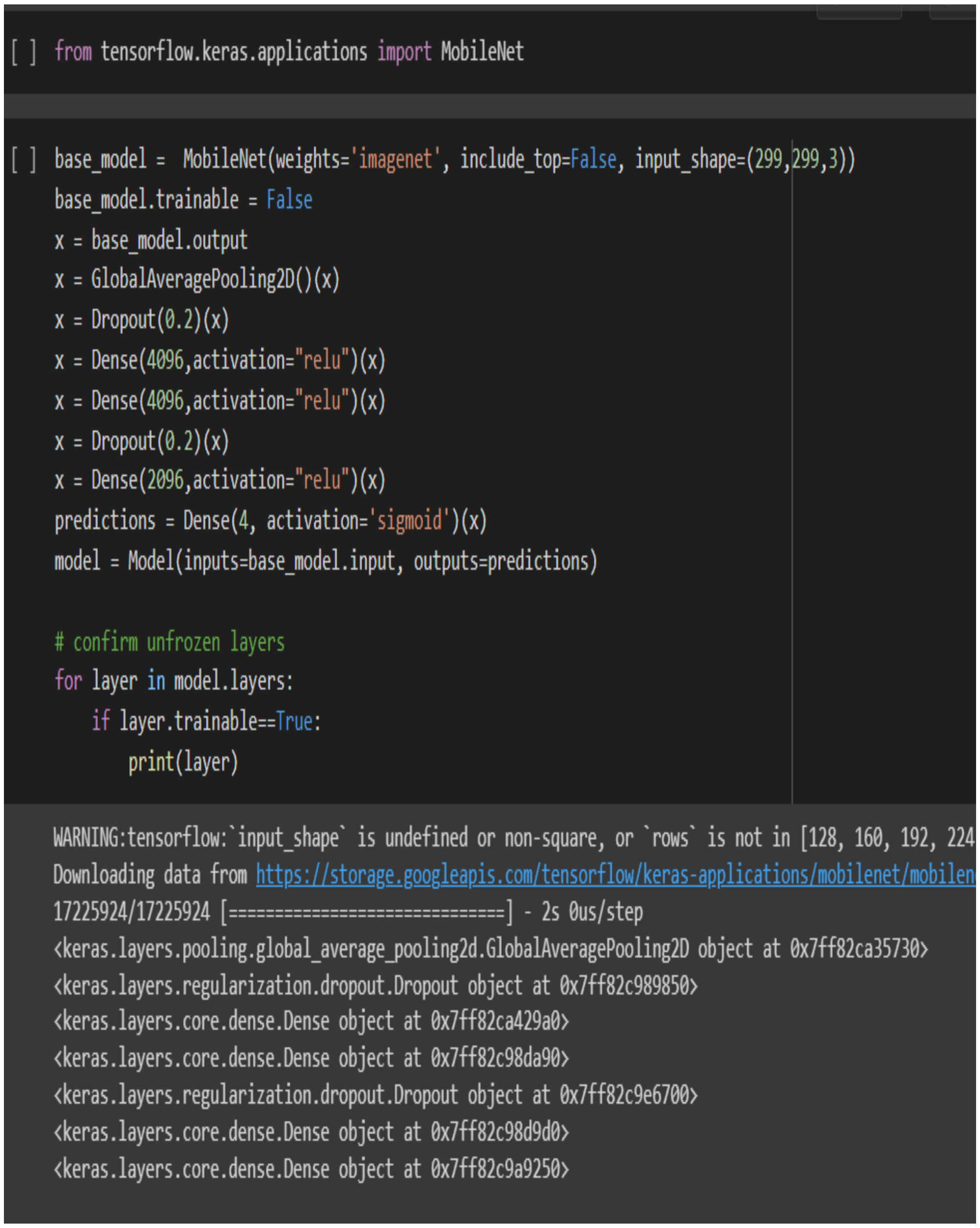
```
[ ] from sklearn.metrics import classification_report
    print(classification_report(Y_true, Y_pred_classes))

              precision    recall  f1-score   support

           0       0.82      0.99      0.90       397
           1       0.95      0.87      0.91       161
           2       0.70      0.11      0.20        61
           3       0.00      0.00      0.00        21

    accuracy                           0.85       640
   macro avg       0.62      0.49      0.50       640
weighted avg       0.81      0.85      0.80       640
```

Fig. 3.36: Classification report of DenseNet201

The accuracy achieved by the DenseNet201 model is 85%.

▪ IMPLEMENTATION OF MOBILENET MODEL

➢ IMPORTING AND IMPLEMENTING OF MOBILENET



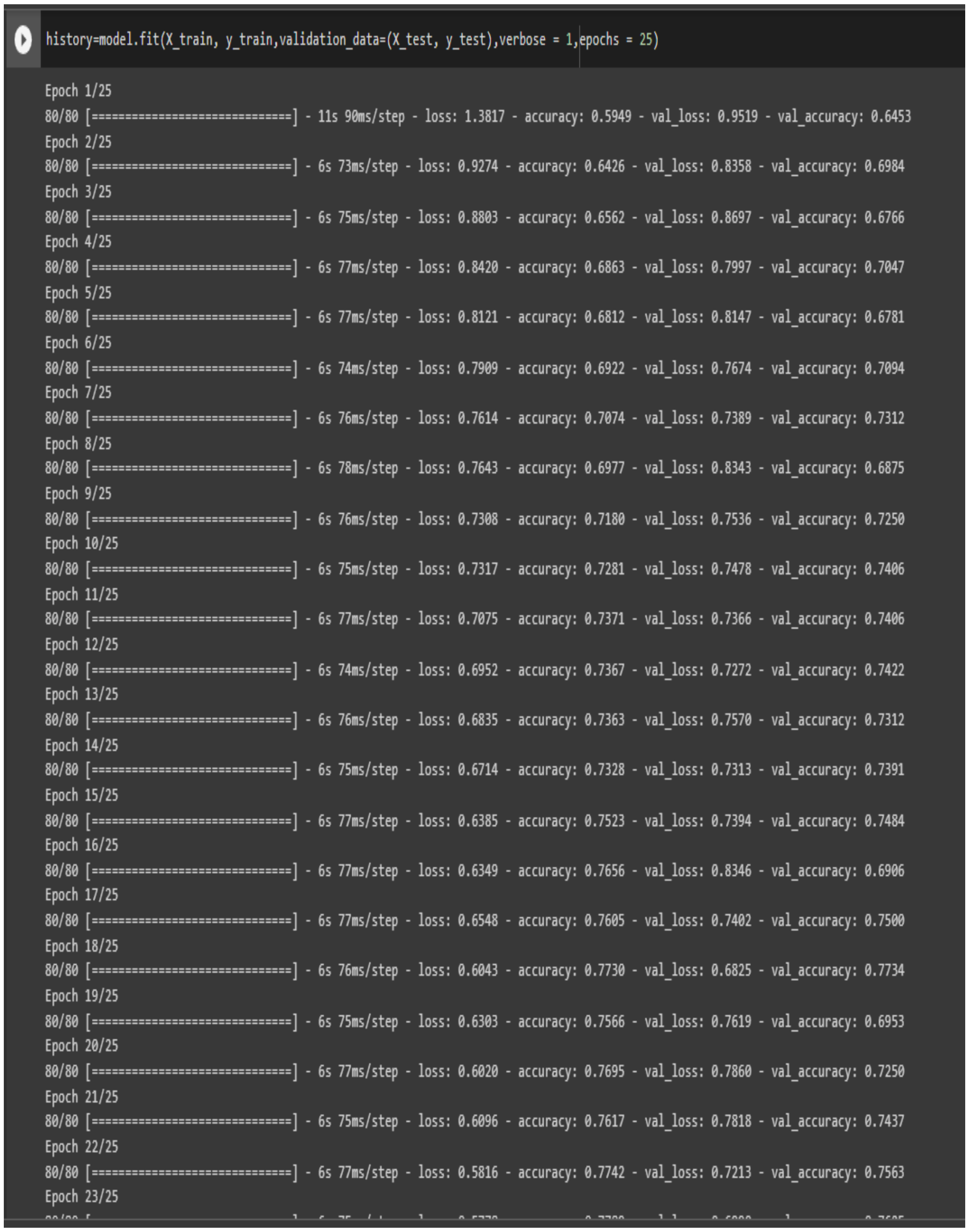
```
[ ] from tensorflow.keras.applications import MobileNet
```

```
[ ] base_model =  MobileNet(weights='imagenet', include_top=False, input_shape=(299,299,3))
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    x = Dense(4096,activation="relu")(x)
    x = Dense(4096,activation="relu")(x)
    x = Dropout(0.2)(x)
    x = Dense(2096,activation="relu")(x)
    predictions = Dense(4, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # confirm unfrozen layers
    for layer in model.layers:
        if layer.trainable==True:
            print(layer)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilen
17225924/17225924 [==============================] - 2s 0us/step
<keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object at 0x7ff82ca35730>
<keras.layers.regularization.dropout.Dropout object at 0x7ff82c989850>
<keras.layers.core.dense.Dense object at 0x7ff82ca429a0>
<keras.layers.core.dense.Dense object at 0x7ff82c98da90>
<keras.layers.regularization.dropout.Dropout object at 0x7ff82c9e6700>
<keras.layers.core.dense.Dense object at 0x7ff82c98d9d0>
<keras.layers.core.dense.Dense object at 0x7ff82c9a9250>
```

Fig. 3.37: Importing and Implementing MobileNet model.

➢ FINDING EPOCH VALUE



```
history=model.fit(X_train, y_train,validation_data=(X_test, y_test),verbose = 1,epochs = 25)

Epoch 1/25
80/80 [==============================] - 11s 90ms/step - loss: 1.3817 - accuracy: 0.5949 - val_loss: 0.9519 - val_accuracy: 0.6453
Epoch 2/25
80/80 [==============================] - 6s 73ms/step - loss: 0.9274 - accuracy: 0.6426 - val_loss: 0.8358 - val_accuracy: 0.6984
Epoch 3/25
80/80 [==============================] - 6s 75ms/step - loss: 0.8803 - accuracy: 0.6562 - val_loss: 0.8697 - val_accuracy: 0.6766
Epoch 4/25
80/80 [==============================] - 6s 77ms/step - loss: 0.8420 - accuracy: 0.6863 - val_loss: 0.7997 - val_accuracy: 0.7047
Epoch 5/25
80/80 [==============================] - 6s 77ms/step - loss: 0.8121 - accuracy: 0.6812 - val_loss: 0.8147 - val_accuracy: 0.6781
Epoch 6/25
80/80 [==============================] - 6s 74ms/step - loss: 0.7909 - accuracy: 0.6922 - val_loss: 0.7674 - val_accuracy: 0.7094
Epoch 7/25
80/80 [==============================] - 6s 76ms/step - loss: 0.7614 - accuracy: 0.7074 - val_loss: 0.7389 - val_accuracy: 0.7312
Epoch 8/25
80/80 [==============================] - 6s 78ms/step - loss: 0.7643 - accuracy: 0.6977 - val_loss: 0.8343 - val_accuracy: 0.6875
Epoch 9/25
80/80 [==============================] - 6s 76ms/step - loss: 0.7308 - accuracy: 0.7180 - val_loss: 0.7536 - val_accuracy: 0.7250
Epoch 10/25
80/80 [==============================] - 6s 75ms/step - loss: 0.7317 - accuracy: 0.7281 - val_loss: 0.7478 - val_accuracy: 0.7406
Epoch 11/25
80/80 [==============================] - 6s 77ms/step - loss: 0.7075 - accuracy: 0.7371 - val_loss: 0.7366 - val_accuracy: 0.7406
Epoch 12/25
80/80 [==============================] - 6s 74ms/step - loss: 0.6952 - accuracy: 0.7367 - val_loss: 0.7272 - val_accuracy: 0.7422
Epoch 13/25
80/80 [==============================] - 6s 76ms/step - loss: 0.6835 - accuracy: 0.7363 - val_loss: 0.7570 - val_accuracy: 0.7312
Epoch 14/25
80/80 [==============================] - 6s 75ms/step - loss: 0.6714 - accuracy: 0.7328 - val_loss: 0.7313 - val_accuracy: 0.7391
Epoch 15/25
80/80 [==============================] - 6s 77ms/step - loss: 0.6385 - accuracy: 0.7523 - val_loss: 0.7394 - val_accuracy: 0.7484
Epoch 16/25
80/80 [==============================] - 6s 77ms/step - loss: 0.6349 - accuracy: 0.7656 - val_loss: 0.8346 - val_accuracy: 0.6906
Epoch 17/25
80/80 [==============================] - 6s 77ms/step - loss: 0.6548 - accuracy: 0.7605 - val_loss: 0.7402 - val_accuracy: 0.7500
Epoch 18/25
80/80 [==============================] - 6s 76ms/step - loss: 0.6043 - accuracy: 0.7730 - val_loss: 0.6825 - val_accuracy: 0.7734
Epoch 19/25
80/80 [==============================] - 6s 75ms/step - loss: 0.6303 - accuracy: 0.7566 - val_loss: 0.7619 - val_accuracy: 0.6953
Epoch 20/25
80/80 [==============================] - 6s 77ms/step - loss: 0.6020 - accuracy: 0.7695 - val_loss: 0.7860 - val_accuracy: 0.7250
Epoch 21/25
80/80 [==============================] - 6s 75ms/step - loss: 0.6096 - accuracy: 0.7617 - val_loss: 0.7818 - val_accuracy: 0.7437
Epoch 22/25
80/80 [==============================] - 6s 77ms/step - loss: 0.5816 - accuracy: 0.7742 - val_loss: 0.7213 - val_accuracy: 0.7563
Epoch 23/25
```

Fig. 3.38: Evaluating accuracy for total 25 epochs in MobileNet

The highest accuracy achieved is 0.7734 at epoch value 18 and epoch value 25.

➢ CONFUSION MATRIX OF MOBILENET MODEL

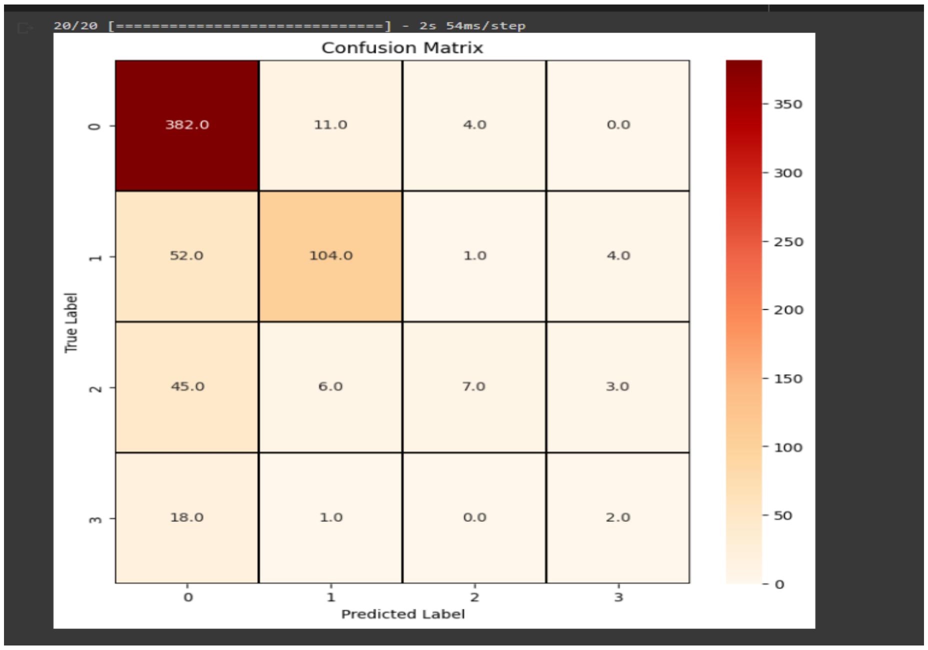
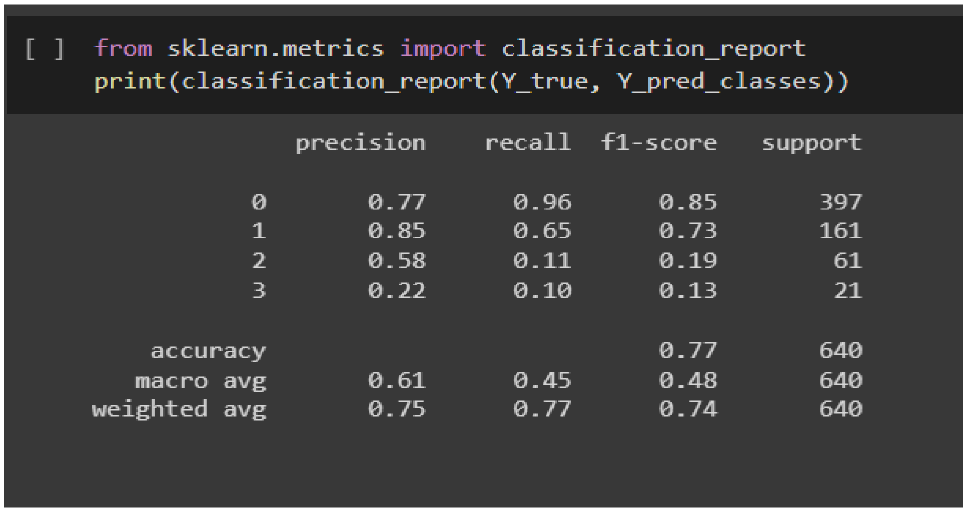

```
20/20 [==============================] - 2s 54ms/step
```

Fig. 3.39: Confusion Matrix for MobileNet


```
from sklearn.metrics import classification_report
print(classification_report(Y_true, Y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.77      0.96      0.85       397
           1       0.85      0.65      0.73       161
           2       0.58      0.11      0.19        61
           3       0.22      0.10      0.13        21

    accuracy                           0.77       640
   macro avg       0.61      0.45      0.48       640
weighted avg       0.75      0.77      0.74       640
```

Fig. 3.40: Classification Report of MobileNet

The accuracy achieved by the MobileNet model is 77%.

CHAPTER-4
RESULT ANALYSIS

The proposed CNN-based cataract detection system is evaluated in two phases. Loss values for the training and validation phases are calculated. Accuracy is calculated using these ratios of actual and predicted signs. Actual predicted values are computed from the base truth and predicted values. Evaluations during the testing phase are based on a number of statistical measures. Ground truth cataracts correctly identified as cataracts are called true positives (T_p), and cataracts incorrectly identified as positives are called false positives (F_p). Ground truth negative labelled cataracts recognized as negatives are called true negatives (T_n) and false negatives are called false negatives (F_n). where the sensitivity (S_t), specificity (S_f), true positive rate (TPR), false positive rate (FPR), error rate (ERR) and accuracy (A) are calculated according to equation (1-6) [12].

STASTICAL ANALYSIS –

1) $Sf = \frac{Tn}{Tn+Fp}$
2) $FPR = \frac{Fp}{Fp+Tn}$
3) $Accuracy\ (A) = \frac{Tp+Tn}{Tp+Fn+Fp+Fn}$
4) $Precision\ (P) = \frac{Tp}{Tp+Fp}$
5) $Recall\ (R)\ or\ (TPR)\ or\ (St) = \frac{Tp}{Tp+Fn}$
6) $ERR = \frac{Fp+Fn}{Tp+Fn+Fn+Tn}$

Table 4.1: Performance Analysis Table

CNN Models	**Different Performance Measure**			
	Precision	**Recall**	**F1-score**	**Accuracy (in %)**
VGG-16	0.92	0.93	0.93	89
VGG-19	0.94	0.97	0.95	93
Inception V3	0.77	0.92	0.84	77
ResNet50	0.92	0.98	0.95	92
DenseNet201	0.82	0.99	0.90	85
MobileNet	0.77	0.96	0.85	77

Above shown, Table 4.1 displays the metrics achieved for different CNN models i.e. for VGG-16, VGG-19, Inception V3, ResNet50, DenseNet201 and MobileNet. The results displayed are in percentage in which VGG-19 model achieved the highest accuracy followed by ResNet50 comparing other models.

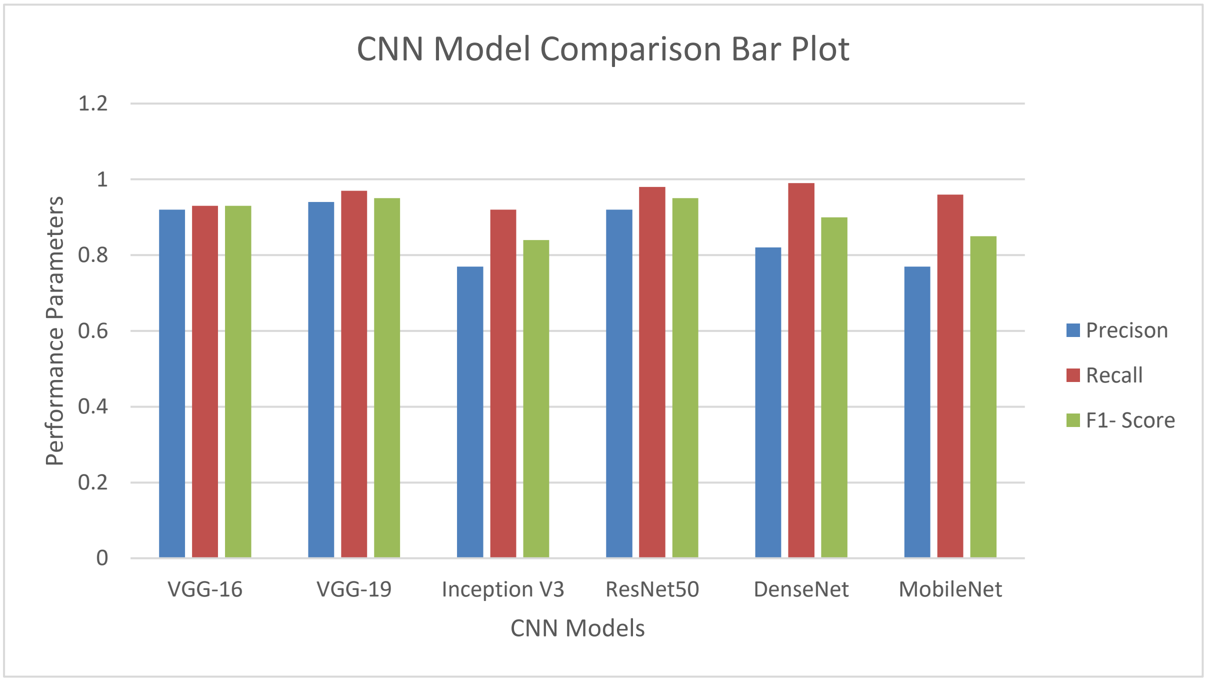


Fig. 4.1: CNN Model Comparison Graph

Above Fig. 4.1 represents a bar plot according to the result analysis shown in Table 4.1. This demonstrates the visual representation of data as vertical rectangular bars, with the length of the bars corresponding to the measure of the data. The bar plot shows the comparison between different Performance parameters such as Precision, Recall and F1-Score.

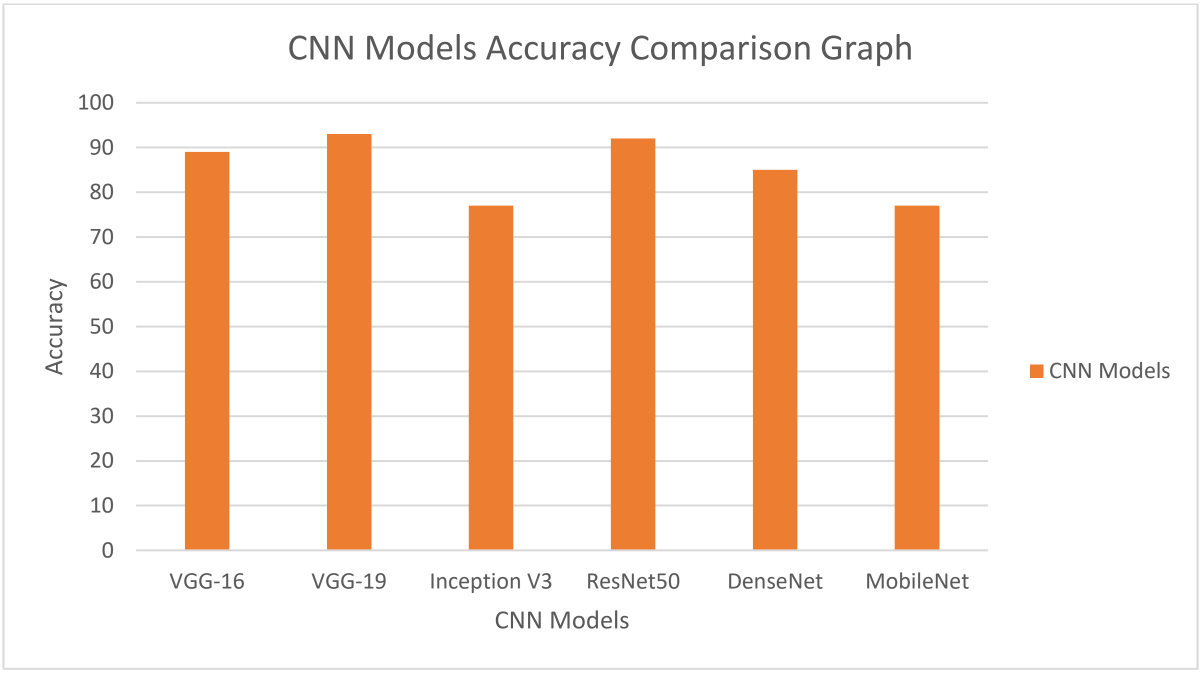


Fig. 4.2: Accuracy Comparison Graph

Above Fig. 4.1 represents a bar plot according to the result analysis shown in Table 4.2. This demonstrates the visual representation of data as vertical rectangular bars, with the length of the bars corresponding to the measure of the data. The bar plot shows the comparison between Performance parameters accuracy achieved of different CNN models.

CHAPTER-5
CONCLUSIONS

5.1 Conclusions

According to the aforementioned study, convolutional neural networks are a fantastic tool for computer vision and related fields since they can identify features in unprocessed data. They are capable of identifying the relationships between various pixels in the training set and utilizing this knowledge to create their own features, ranging from low-level (edges, circles) to high-level (faces, hands, eyes). The issue is that certain features may be difficult for humans to understand. A crazy pixel in a photograph might occasionally produce unexpected new outcomes, too.
This study was proposed to minimize data loss and increase the accuracy of the cataract identification process performing alternating epochs. Research results show that adding more epochs affects the accuracy and lost data of convolutional neural networks. According to this study, VGG-19 model achieved the best accuracy followed by ResNet50 among all the models (VGG-16, VGG-19, Inception V3, ResNet50, DenseNet201 and MobileNet) with an accuracy of 93% and 92% respectively. Apart, Inception V3 has 42 deep layers, the accuracy achieved totally depends upon the dataset.

5.2 Future Scope

Cataract surgeries will increase by about 10% each year due to population demographics, but also due to the increased safety and accuracy of cataract surgery. There is a huge percentage of patients who have astigmatism, and if not corrected, they are basically left with glasses after cataract surgery. Some patients are fine with it. But the ability to correct astigmatism during cataract surgery, whether it's mono-focal, multifocal, extended depth of field, or some other type of lens is such a big technological advance. And that's such a basic thing to correct during cataract surgery. For the future, one thing we can absolutely guarantee is that surgery will be more accurate. It will become safer. Reimbursement will most likely decrease, but I think the surgery will be more adjustable moving forward. Deep learning shows

some promising directions for VGG19. The transfer learning, generative models, and reinforcement machine learning research directions discussed in the previous section have significant methodological advancement potential, but they also highlight the need for additional study. Computer vision tasks have seen the most use of generative models. Unresolved research concerns include evaluating the generated samples' reliability in light of the coherence of physical systems. Another unresolved issue is how to manage the creation of pertinent data samples in terms of physical processes.

Recently, several methods to produce faulty samples or features using generative neural networks have been put forth. However, the majority of research focuses on signals like pictures and preprocessed vibrational data. Examining the applicability of such methods to more complicated datasets and time series data is an intriguing research area. Additional analysis of the created samples' physical plausibility and the effects of different generated mistakes on algorithm performance are also necessary.

REFERNCES

[1] D. Allen and A. Vasavada, “Cataract and surgery for cataract,” *BMJ*, vol. 333, no. 7559, pp. 128–132, Jul. 2006, doi: 10.1136/bmj.333.7559.128.

[2] F. Hollows and D. Moran, “CATARACT-THE ULTRAVIOLET RISK FACTOR,” *The Lancet*, vol. 318, no. 8258, pp. 1249–1250, Dec. 1981, doi: 10.1016/s0140-6736(81)91490-2.

[3] L. Robman and H. Taylor, “External factors in the development of cataract,” *Eye*, vol. 19, no. 10, pp. 1074–1082, Oct. 2005, doi: 10.1038/sj.eye.6701964.

[4] A. Foster, C. Gilbert and J. Rahi, “Epidemiology of cataract in childhood: A global perspective,” *Journal of Cataract & Refractive Surgery*, vol. 23, pp. 601–604, Jan. 1997, doi: 10.1016/S0886-3350(97)80040-5.

[5] G. Brian and H. Taylor, "Cataract blindness: challenges for the 21st century."https://www.scielosp.org/article/ssm/content/raw/?resource_ssm_path=/media/assets/bwho/v79n3/v79n3a15.pdf

[6] T. Toh, J. Morton, J. Coxon and M. J. Elder, “Medical treatment of cataract,” *Clinical & Experimental Ophthalmology*, vol. 35, no. 7, pp. 664–671, Sep. 2007, doi: 10.1111/j.1442-9071.2007.01559.x.

[7] I. Weni, P. E. P. Utomo, B. F. Hutabarat, and M. Alfalah, "Detection of Cataract Based on Image Features Using Convolutional Neural Networks," *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, vol. 15, no. 1, p. 75, Jan. 2021, doi: 10.22146/ijccs.61882.

[8] D. Lam, S.K. Rao, V. Ratra, Y. Liu, P. Mitchell, J. King, M.J. Tassignon, J. Jonas, C.P. Pang and D.F. Chang, "Cataract," *Nature Reviews Disease Primers*, vol. 1, no. 1, Jun. 2015, doi: 10.1038/nrdp.2015.14.

[9] Bansal, M. Kumar, M. Sachdeva and A. Mittal, "Transfer learning for image classification using VGG19: Caltech-101 image data set," *Journal of Ambient Intelligence and Humanized Computing*, Sep. 2021, doi: 10.1007/s12652-021-03488-z.

[10] L. Gutierrez, J.S. Lim, L.L. Foo, W.Y. Ng, M. Yip, G.Y.S. Lim, M.H.Y. Wong, A. Fong, M. Rosman, J.S. Mehta, H. Lin, D.S.J. Ting and D.S.W. Ting, "Application of artificial intelligence in cataract management: current and future directions," *Eye and Vision*, vol. 9, no. 1, Jan. 2022, doi: https://doi.org/10.1186/s40662-021-00273-z.

[11] Md. S. M. Khan, M. Ahmed, R. Z. Rasel and M. M. Khan, "Cataract Detection Using Convolutional Neural Network with VGG-19 Model," *IEEE Xplore*, May 01, 2021. https://ieeexplore.ieee.org/abstract/document/9454244 (accessed May 31, 2022).

[12] R. Hossain, S. Afroze, N. Siddique and M. M. Hoque, "Automatic Detection of Eye Cataract using Deep Convolution Neural Networks (DCNNs)," *IEEE Xplore*, Jun. 01, 2020. https://ieeexplore.ieee.org/abstract/document/9231045/.

[13] Y. Kumar and S. Gupta, "Deep Transfer Learning Approaches to Predict Glaucoma, Cataract, Choroidal Neovascularization, Diabetic Macular Edema, DRUSEN and Healthy Eyes: An Experimental Review," *Archives of Computational Methods in Engineering*, Sep. 2022, doi: https://doi.org/10.1007/s11831-022-09807-7.

[14] S. Jayachitra, K. N. Kanna, G. Pavithra and T. Ranjeetha, "A Novel Eye Cataract Diagnosis and Classification Using Deep Neural Network," *Semantic Scholar*, 2021, doi: https://doi.org/10.1088/1742-6596/1937/1/012053.

[15] A. K. Bitto and I. Mahmud, "Multi categorical of common eye disease detect using convolutional neural network: a transfer learning approach," *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 4, pp. 2378–2387, Aug. 2022, doi: https://doi.org/10.11591/eei.v11i4.3834.