

Covid-19 Detection Using Chest X-Ray

A PROJECT REPORT

Submitted by

Bibek Rawat (18BCS6728)

*Submitted in partial fulfillment of the requirements for the award of
degree of*

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING



**CHANDIGARH
UNIVERSITY**
Discover. Learn. Empower.

Chandigarh University

March 2022



**CHANDIGARH
UNIVERSITY**
Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report “**Covid-19 Detection using chest X-ray**” is the bonafide work of “**Bibek Rawat**” who carried out the project work under my/our supervision.

SIGNATURE

Prof. Anup Lal Yadav

SUPERVISOR

BE-CSE

SIGNATURE

Dr. Reema Goyal

HEAD OF THE DEPARTMENT

BE-CSE

Submitted for the project viva-voce examination held on_____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

It is always a pleasure to remind the fine people of Chandigarh University for their sincere guidance that we received to uphold our practical skills in machine learning. It is opportunities like these makes us proud to be students of this University. This project is based on Deep Learning which had to be done in Python. Python had been taught to us as part of our 2nd year curriculum.

First of all, I would like to thank to my parents for giving encouragement, enthusiasm and invaluable assistance. Without all this, I might have not been able to complete this project properly and keep my morale up in times of failure.

Secondly, I would like to thank our Vice Chancellor for giving me the opportunity to present this project to the University.

Thirdly, I would like to thank my mentor Prof. Anup Lal Yadav as my project advisor for CSE - Department as he helped me a lot in dealing with project related problems. He has supported me by giving ideas to encourage critical thinking. He helped me all the time when I needed her and gave right direction for completion of project without which I would have been lost. I would like to express my special thanks of gratitude to my teacher as well as my HOD for giving me golden opportunity as well as approval of my project on the topic of deep learning. This has helped me in learning new things and gain experience and put my theoretical knowledge to practical use in the form of a real world project. This will no doubt help me a lot in my future for placements and higher studies. In the end I would like to thank all the people, such as my classmates who remain unnamed for the purpose of this Acknowledgment.

Table of Contents:

Topics	Pages
I. INTRODUCTION	11-14
II. PROJECT SCOPE	14
III. LITERATURE REVIEW 1. CNN 2. Transfer Learning 3. VGG-16 4. ResNet-50 5. Xception 6. InceptionV3 7. ResNet-101 8. MobileNet 9. InstaCovNet-19 10. DarkCovidNet 11. DenseNet	14-32
IV. FEASIBILITY STUDY 1. Technical Feasibility 2. Operational Feasibility 3. Economic Feasibility 4. Organizational Feasibility 5. Cultural Feasibility	32-34
V. METHODOLOGY/PLANNING OF WORK 1. Data Collection 2. Data Pre-processing 3. Image augmentation 4. Deep Learning Models	34-38

VI. PROJECT DESIGN FLOW / LIST OF PROPOSED MODELS 1. VGG-16 2. ResNet-50 3. CNN 4. CNN (local Maxima)	38-73
VII. CODE SCREENSHORT	73-94
VIII. OUTPUT	94-98
IX. OUTPUT VALIDATION AND COMPARISION	99
X. INNOVATION IN MODEL 1. LIMITATION	99-100
XI. FUTURE WORK	100
XII. CONCLUSION	100-101
XIII. BIBLIOGRAPHY	101-105

List of Figures

Figure no.	Name of the Figure	Page no.
1	Country wise cases distribution	12
2	Total Covid-19 death with time interval	13
3	CNN Architecture	15

4	Formula For Convolution Layer	16
5	Pooling Operations	16
6	Formula For Padding Layer	17
7	Sigmoid Activation Function	18
8	Tanh Activation Function	18
9	ReLU Activation Function	19
10	Architecture of Transfer Learning	20
11	VGG-16 Architecture	21
12	VGG-16 architecture map	21
13	ResNet-50 Architecture	23
14	ResNet-50 Detailed Architecture	23
15	Xception Architecture	25
16	Architecture of InceptionV3	26
17	Feature Extraction of ResNet-101	27
18	MobileNet Architecture	28
19	InstaCovNet-19 integrated stacked model architecture	29
20	DarkCovidNet Architecture	30
21	DenseNet Architecture	31
22	Waterfall Model Architecture	35

23	Activity Diagram for Covid-19 Detection	39
24	Use Case Diagram	40
25	Black Box View for All Models	41
26	VGG-16 General Architecture	42
27	Working Flow of VGG-16 Model	43
28	VGG-16 Model Training vs Test Accuracy	48
29	VGG-16 Model Training and Validation Loss Comparison	49
30	Working Flow of ResNet-50 Model	50
31	ResNet-16 Model Training and Validation Accuracy Comparison	62
32	ResNet-16 Model Training and Validation Loss Comparison	63
33	CNN Model Training and Validation Accuracy Comparison	65
34	CNN Model Training and	69

	Validation Loss Comparison	
35	Condition of Local Maxima	70
36	CNN (Local Maxima) Model Training and Validation Accuracy Comparison	72
37	CNN (Local Maxima) Model Training and Validation Loss Comparison	73

List of Tables

Table no.	Name of the Table	Page no.
1	List of Research Work on Covid-19 Detection	32
2	List of Dataset which are previously used	37
3	VGG-16 Model Summary	45
4	VGG-16 Model Training Status	48
5	ResNet-50 Model Summary	58
6	ResNet-50 Model Training Status	62

7	CNN Model Summary	66
8	CNN Model Training Status	68
9	CNN (Local Maxima) Model Summary	71
10	CNN (Local Maxima) Model Training Status	72
11	Comparison between Proposed Models	99

ABSTRACT

The medical sector is seeking, in this global health crisis, innovative solutions to track and contain the COVID-19 pandemic. Artificial intelligence is a technology that scientists can rely on, since it can quickly classify high-risk patients, monitor the progression of this virus, and effectively manage this outbreak in real time. A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/ objects in the image and be able to differentiate one from the other. Artificial intelligence has been a great contributor to the field of medical diagnosis and the design of new medications. Covid-19 pandemic has attracted the attention of big data analysts and artificial intelligence engineers. To solve such problem many A.I algorithm are being used, CNN is one of them for effective accuracy we have to adjust parameter accordingly.

In this project I am going to test different models along with their parameter and their tuning method for covid-19 detection using CNN or image processing. For any successful model accuracy (training and testing) is one of the biggest issue to solve that issue we have to consider the different parameter of model like activation function,

learning rate, number of trainable parameter, data pre-processing, epochs etc. Furthermore, we test CNN model with different architecture and see how it is going to perform with different architecture. With the help of our study we can improve any CNN model which is related to covid-19 detection as well as it also helps in real medical uses by reducing the cost and crowd of covid-19 detection i.e. RT PCR.

GLOSSARY

Covid-19

Coronavirus disease 2019 is a contagious disease caused by a virus.

Deep learning

A machine learning method which composes details together to obtain more abstract, higher level, features of the data through composition of mathematical functions. Powerful modern deep learning algorithms often involve a large number of these levels.

Machine learning

A set of rules that allows systems to learn directly from examples, data and experience.

Image Processing

Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it.

Activation Function

An activation function is a function used in artificial neural network which outputs a small value for small inputs, and a larger value if its inputs exceed a threshold.

Model

A mathematical description of a system.

Neural network

A computer model with a particular form that was originally inspired by early work on understanding the nervous system.

Test data

Data that is used to test the functioning of a machine learning system, or verify its outputs.

Training data

Data that can be used to train machine learning systems, having already been labelled or categorized into one or more groups.

List of Abbreviations

CXR	Chest X-ray
CNN	Convolutional Neural Network
DL	Deep Learning
Val. Acc	Validation Accuracy
Train. Acc	Training Accuracy
Val. Loss	Validation Loss
Train. Loss	Training Loss

KEYWORD

Covid-19, Convolutional Neural Network (CNN), Coronavirus, Deep learning, Image processing, Chest x-ray (CXR), VGG-16, ResNet-50, Covid-19 Models.

I. INTRODUCTION

SARS Cov-2 or Covid-19 is a contagious disease caused by a virus. Covid-19 is a fast spreading pandemic and early detection is crucial for stopping the spread of infection. The world health organization (WHO) declared the pandemic in March 2020, which is called the coronavirus pandemic or Covid-19 pandemic. The novel virus was first identified from an outbreak in Wuhan, China in Dec 2019, as of 23 April 2022, the pandemic had caused more than 508 million cases and 6.21 million death, making it one of the deadliest in history. Due to the limited resources and technology testing has

been limited in some countries to patients exhibiting symptoms and in many cases, multiple symptoms. Needless to point out, the large strain that the situation has imposed on the national health care systems and workers, even those in the most developed countries, feeds into the difficulties of identifying and tracking possible cases.

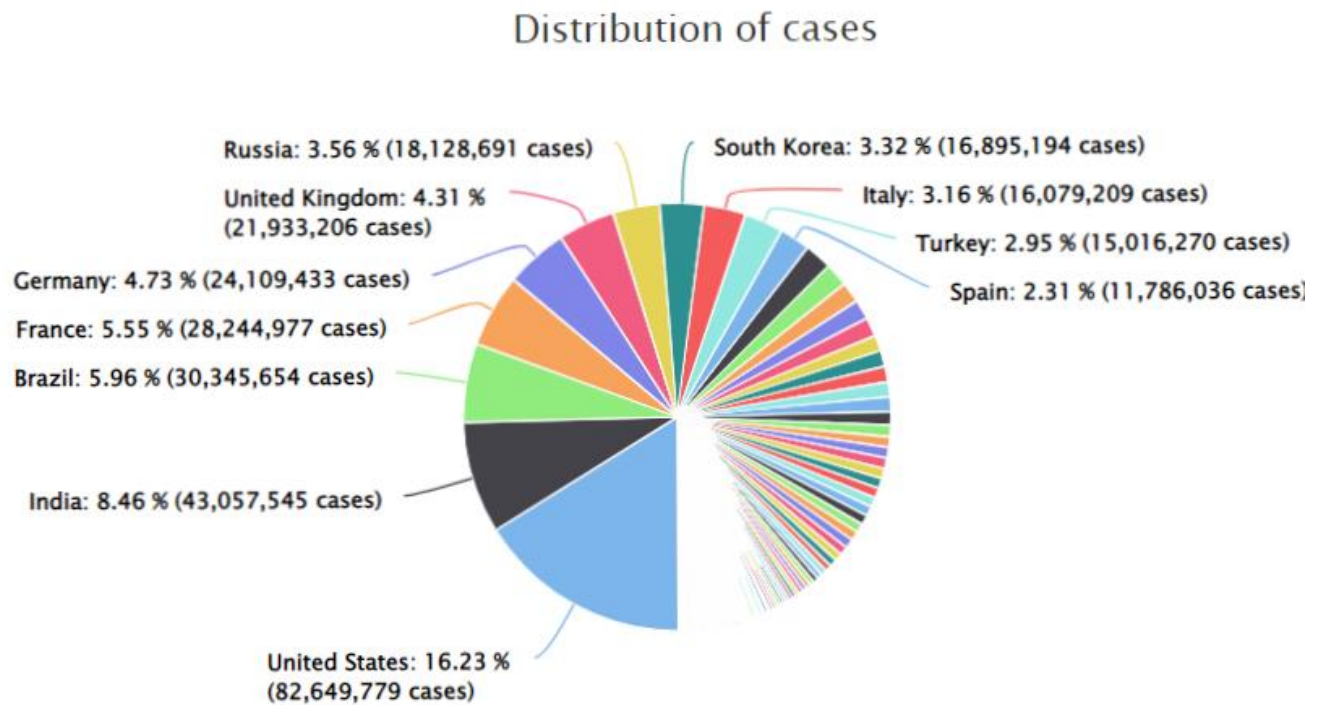


Fig 1:- Country wise cases distribution

The figure 1 shows that how deadliest the virus is. The most affected country by covid-19 is USA with 16.23 % of total cases follow by India and Brazil with 8.46 % and 5.96 %.

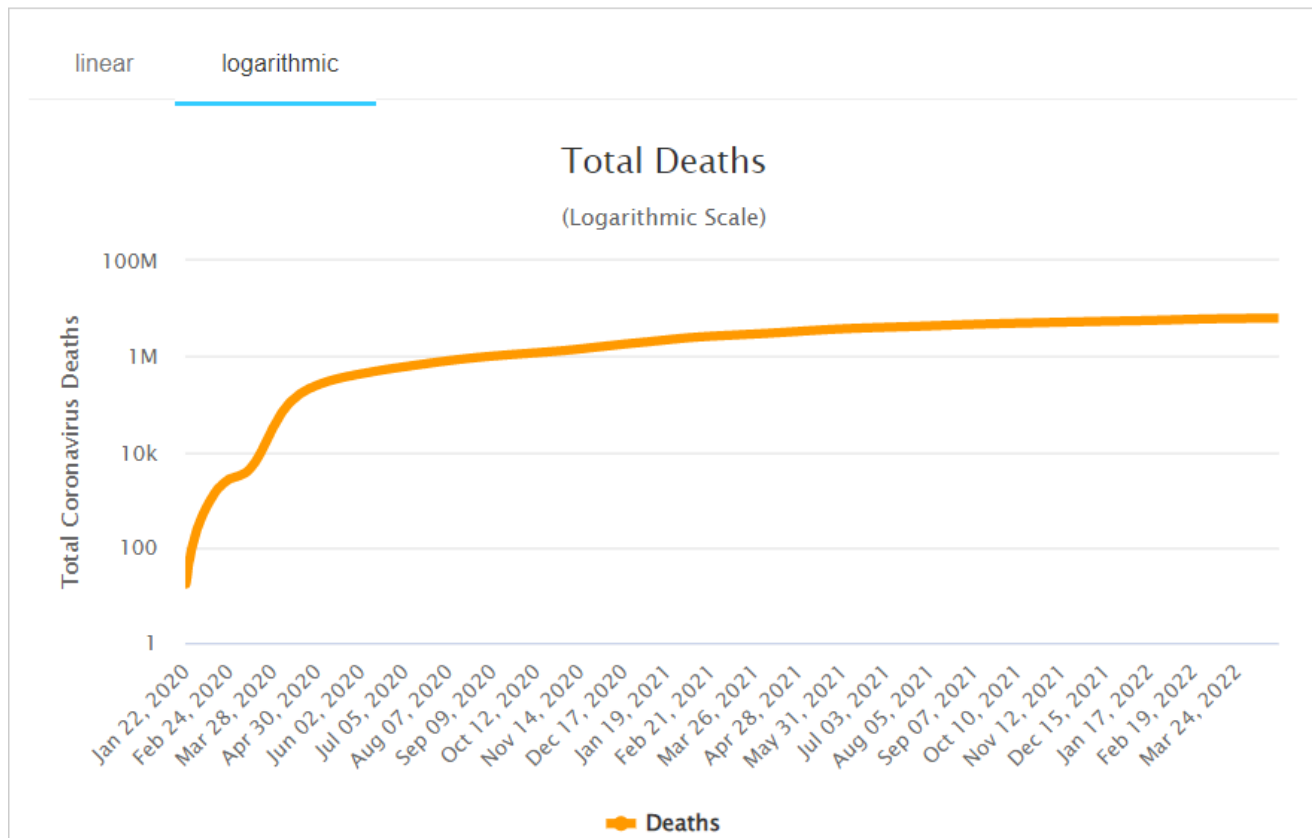


Fig 2:- Total Covid-19 death with time interval2

Till this date covid-19 deaths count become more than a million in world-wide which a huge number but from the figure 2 we can conclude that the deaths curve is now flatten which means death rate is very low in present context. The price of covid-19 diagnostic tests can vary based on the type of test performed, where it is processed the manufacturer and CDC affiliation. The price ranges between \$20 to \$1419 per diagnostic tests with a median of \$148, nearly half of charges (47%) were priced between \$100 to \$199 and one in five (20%) were priced between \$300 [1]. The price of diagnostic test is high and accuracy of test is low. Antigen tests have a sensitivity between 50% to 90 % which produce many false negative. For RT-PCR tests, rectal stools/swab, urine and plasma were less sensitive while sputum (97.2%) presented higher sensitivity for detecting the virus.

Problem Statement

First problem is cost of covid-19 RT-PCR is too much high and time consuming. This is because there is no proper man power and resource. Second main problem is trust, many researcher shows that RT-PCR produces many false positive record which produce doubt regarding covid-19 on our mind.

II. PROJECT SCOPE

This project mainly divided into three modules: 1. Covid-19 Data collection 2. Model training and 3. Model evaluation. At first data from different sources are collected and pre-processes then model is created. Evaluation of models will be start from that point. The proposed system should provide the accurate information about covid-19.

III. LITERATURE REVIEW

This section explain basic concepts of convolutional neural network, transfer learning, ResNet-18, VGG-16, MobileNet model for covid-19 detection. So, it is essential to have a knowledge of different types of deep learning models. These are the some of the best models that are used by previous researcher to identify covid-19 using image processing.

1. CNN (Convolutional neural network)

A convolutional neural network also known as CNN or Convnet is a class of neural network that specializes in processing data that has a grid-like topology such as images. In other words CNN are multilayer neural networks that are stacked on top of each other. A CNN typically has three layers: a convolutional layer, a pooling layer and a fully connected layer.

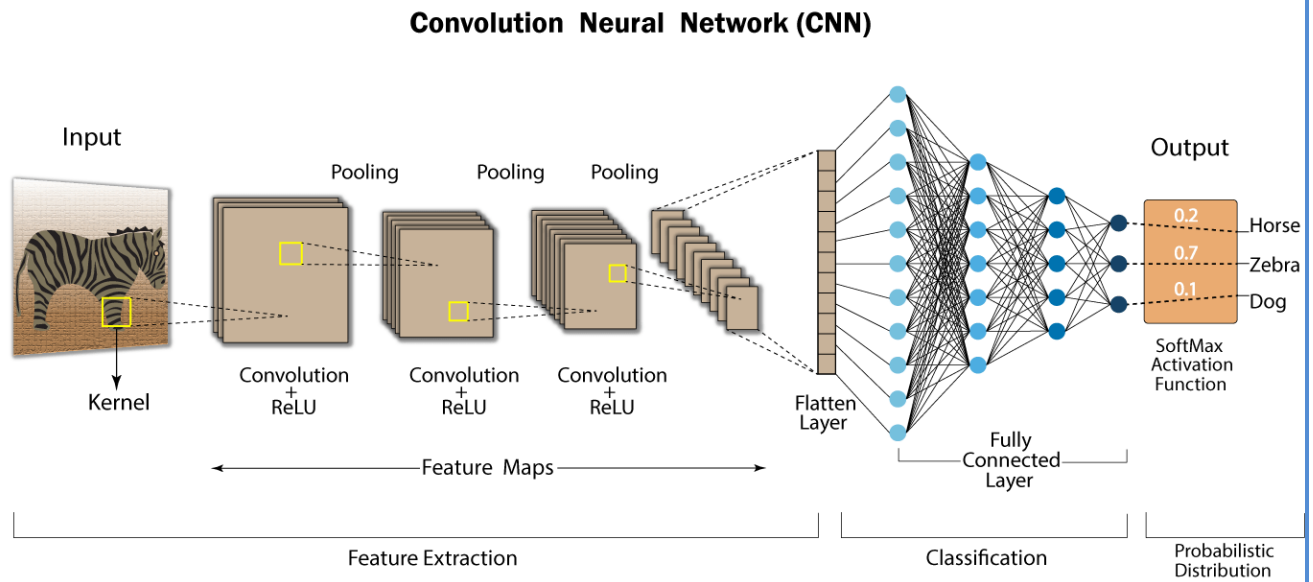


Fig 3:- CNN Architecture

Convolutional layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels. During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride. If we have an input of size $W \times W \times D$ and D_{out} number of kernels with a spatial size of F with stride S and amount of padding P , then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Fig 4:- Formula For Convolution Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually. There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood.

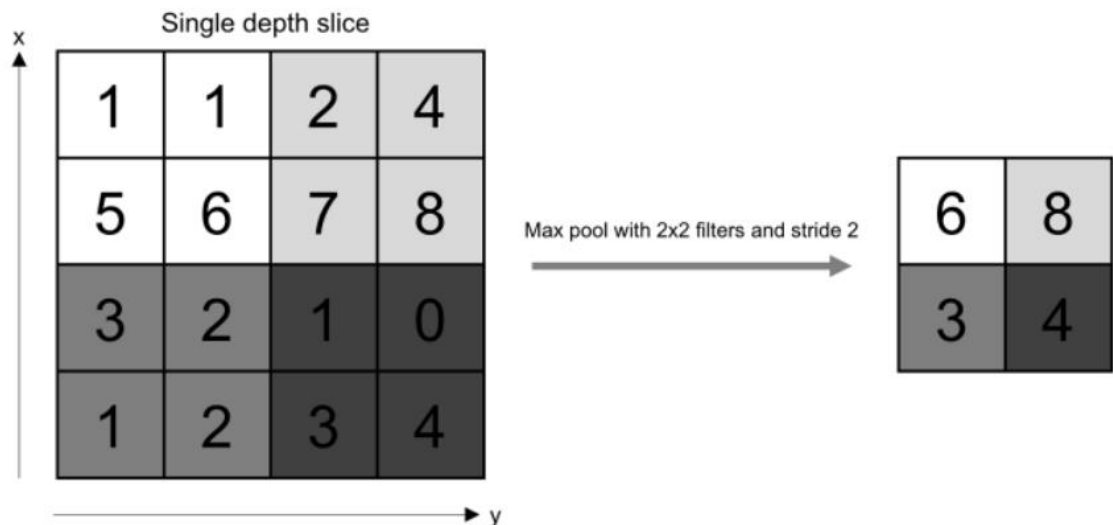


Fig 5:- Pooling Operations

If we have an activation map of size $W \times W \times D$, a pooling kernel of spatial

size F , and stride S , then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F}{S} + 1$$

Fig 6:- Formula For Padding Layer

Neurons in Fully connected layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

Sigmoid

The sigmoid non-linearity has the mathematical form $\sigma(\kappa) = 1/(1+e^{-\kappa})$. It takes a real-valued number and “squashes” it into a range between 0 and 1.

However, a very undesirable property of sigmoid is that when the activation is at either tail, the gradient becomes almost zero. If the local gradient becomes very small, then in back propagation it will effectively “kill” the gradient. Also, if the data coming into the neuron is always positive, then the output of sigmoid will be either all positives or all negatives, resulting in a zig-zag dynamic of gradient updates for weight.

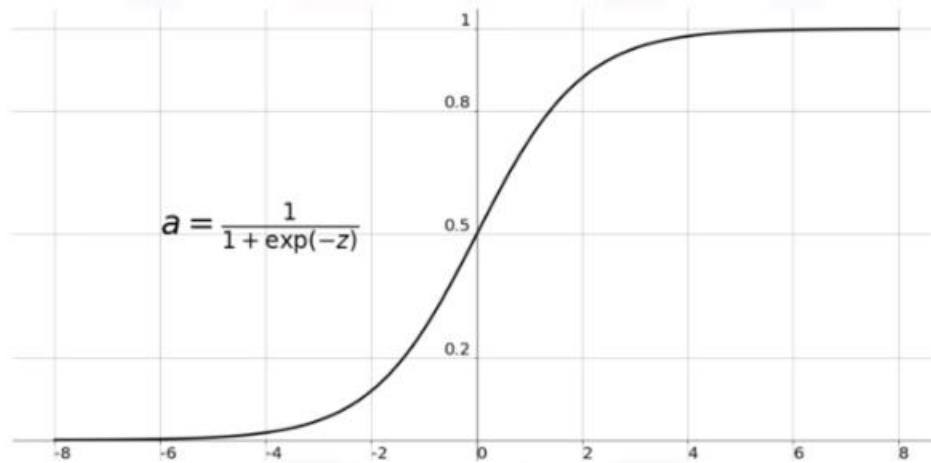


Fig 7:- Sigmoid Activation Function

Tanh

Tanh squashes a real-valued number to the range $[-1, 1]$. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

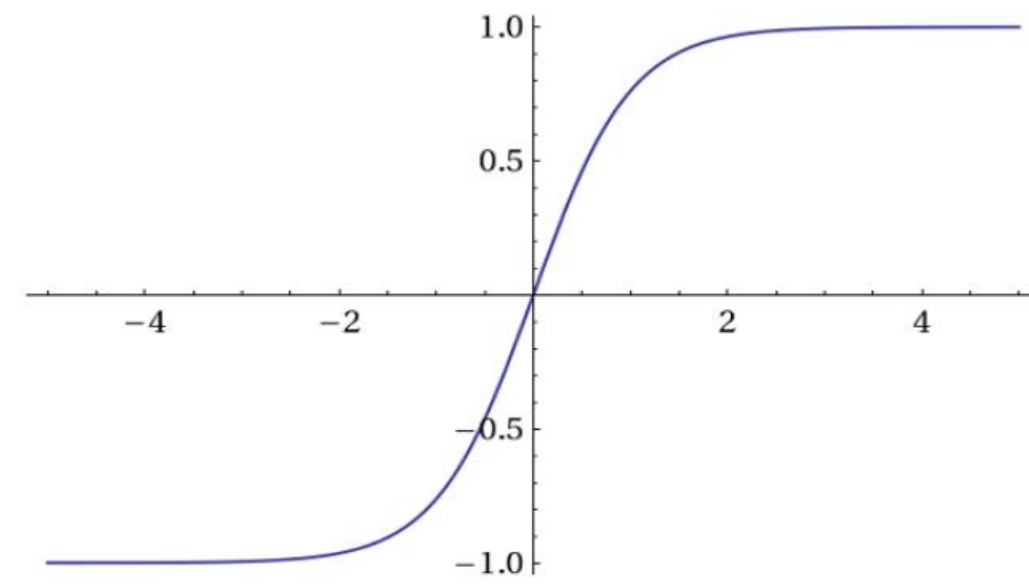


Fig 8:- Tanh Activation Function

ReLU

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(\kappa)=\max(0,\kappa)$. In other words, the activation is simply threshold at zero.

In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times.

Unfortunately, a con is that ReLU can be fragile during training. A large gradient flowing through it can update it in such a way that the neuron will never get further updated. However, we can work with this by setting a proper learning rate.

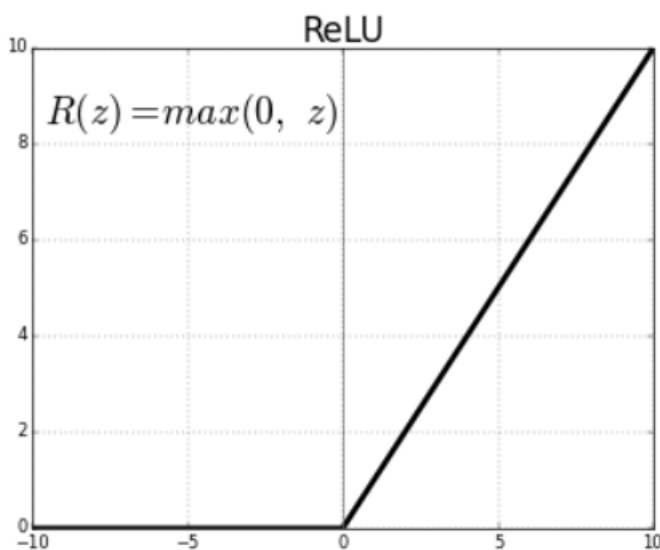


Fig 9:- ReLU Activation Function

2. Transfer Learning

Transfer learning (TL) is a research problems in machine learning (ML) that focuses on storing knowledge gained while solving one problems and applying it to a different but related problems. For example, knowledge gained while learning to recognize a cars could apply when trying to recognize trucks. With basically try to exploit what has been learned in one task to improve generalization in another. We transfer the weights that a network has learned

from model 'A' to 'B'.

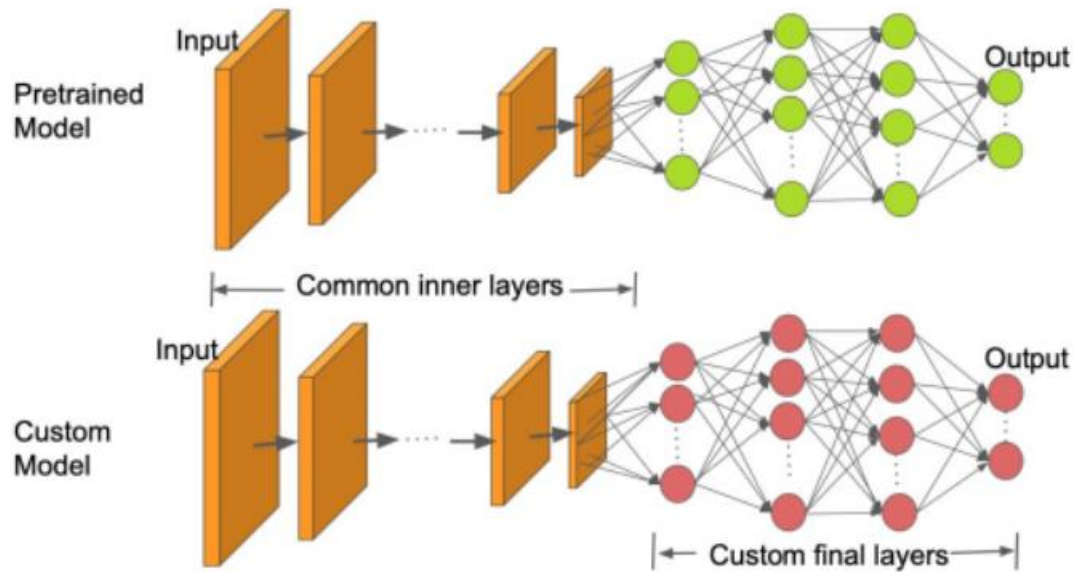


Fig 10:- Architecture of Transfer Learning

3. VGG-16

VGG-16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper “very deep convolutional networks for large scale image recognition”. The VGG architecture was originally proposed for image recognition application. In vgg 16 and 19 wt. layers are used with a smaller convolutional filter size of 3*3. The network won first and second places in the ILSVR(ImageNet) competition in 2014. This models achieves 92.7% top-5 test accuracy on ImageNet dataset which contain 14 million images belonging to 1000 unique classes.

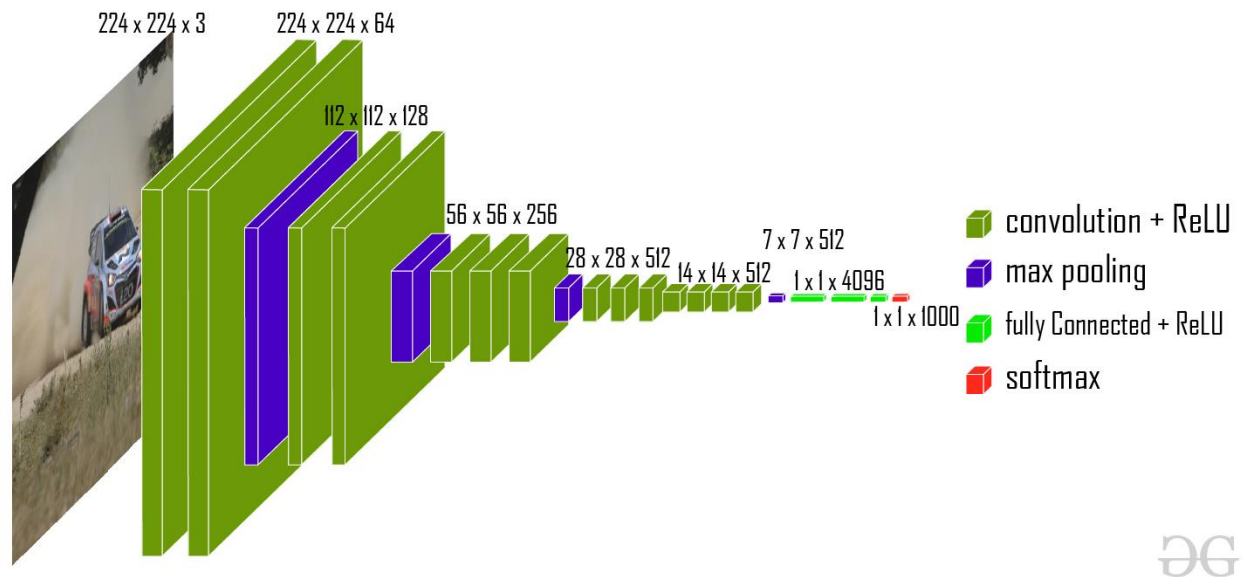


Fig 11:- VGG-16 Architecture

The input to the network is image of dimensions (224, 224, 3). The first two layers have 64 channels of 3×3 filter size and same padding. Then after a max pool layer of stride (2, 2), two layers which have convolution layers of 256 filter size and filter size (3, 3). This followed by a max pooling layer of stride (2, 2) which is same as previous layer. Then there are 2 convolution layers of filter size (3, 3) and 256 filter. After that there are 2 sets of 3 convolution layer and a max pool layer. Each have 512 filters of (3, 3) size with same padding. This image is then passed to the stack of two convolution layers. In these convolution and max pooling layers, the filters we use is of the size 3×3 instead of 11×11 in AlexNet and 7×7 in ZF-Net. In some of the layers, it also uses 1×1 pixel which is used to manipulate the number of input channels. There is a padding of 1-pixel (same padding) done after each convolution layer to prevent the spatial feature of the image.

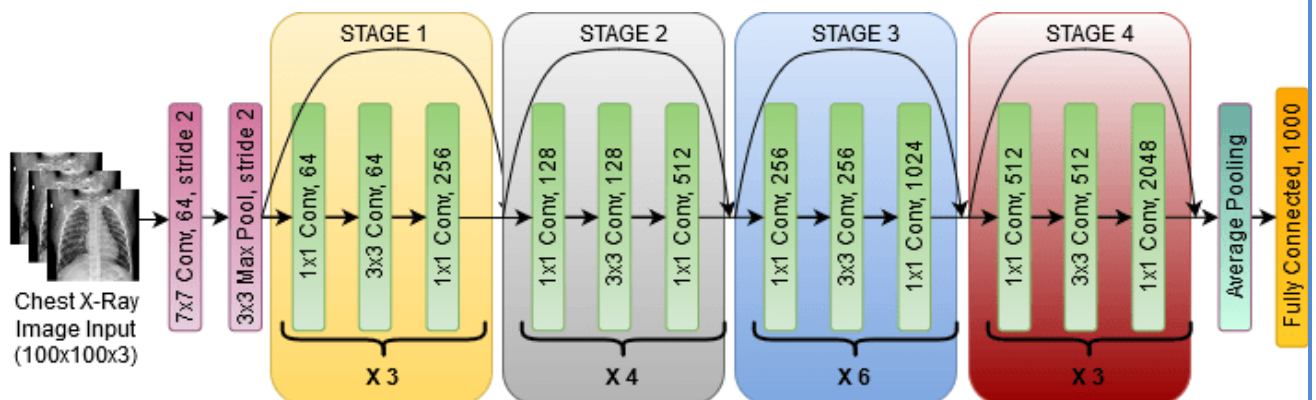


Fig 12:- VGG-16 architecture map

After the stack of convolution and max-pooling layer, we got a (7, 7, 512) feature map. We flatten this output to make it a (1, 25088) feature vector. After this there are 3 fully connected layer, the first layer takes input from the last feature vector and outputs a (1, 4096) vector, second layer also outputs a vector of size (1, 4096) but the third layer output a 1000 channels for 1000 classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to softmax layer in order to normalize the classification vector. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problem.

4. ResNet-50

Deep Convolutional neural networks are really great at identifying low, mid and high level features from the images and stacking more layers generally gives us better accuracy so a question arises that is getting better model performance as easy as stacking more layers?

With this questions arises the problem of vanishing/exploding gradients those problems were largely handled by many ways and enabled networks with tens of layers to converge but when deep neural networks start to converge we see another problem of the accuracy getting saturated and then degrading rapidly and this was not caused by overfitting as one may guess and adding more layers to a suitable deep model just increased the training error. This problem was further rectified by taking a shallower model and a deep model that was constructed with the layers from the shallow model and adding identity layers to it and accordingly the deeper model shouldn't have produced any higher training error than its counterpart as the added layers were just the identity layers. The authors addressed this problem by introducing deep residual learning framework so for this they introduce shortcut connections that simply perform identity mappings.

Retrain ResNet50

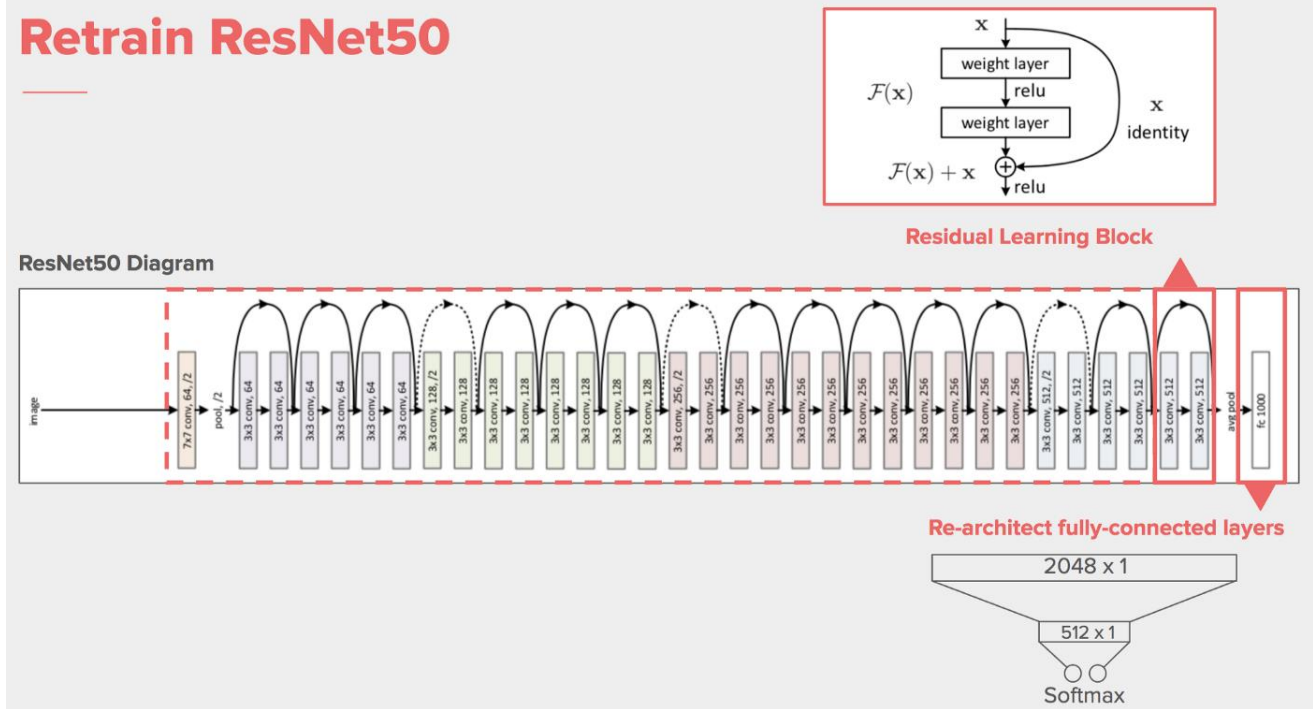


Fig 13:- ResNet-50 Architecture

They explicitly let the layers fit a residual mapping and denoted that as $H(x)$ and they let the non linear layers fit another mapping $F(x) := H(x) - x$ so the original mapping becomes $H(x) := F(x) + x$ as can be seen in Fig 13.

And the benefit of these shortcut identity mapping were that there was no additional parameters added to the model and also the computational time was kept in check.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig 14:- ResNet-50 Detailed Architecture

ResNet-50 architecture contains the following element:

- ✓ A convolution with a kernel size of $7 * 7$ and 64 different kernels all with a stride of size 2 giving us **1 layer**.
- ✓ max pooling with also a stride size of 2.
- ✓ In the next convolution there is a $1 * 1, 64$ kernel following this a $3 * 3, 64$ kernel and at last a $1 * 1, 256$ kernel, These three layers are repeated in total 3 time so giving us **9 layers** in this step.
- ✓ kernel of $1 * 1, 128$ after that a kernel of $3 * 3, 128$ and at last a kernel of $1 * 1, 512$ this step was repeated 4 time so giving us **12 layers** in this step.
- ✓ After that there is a kernel of $1 * 1, 256$ and two more kernels with $3 * 3, 256$ and $1 * 1, 1024$ and this is repeated 6 time giving us a total of **18 layers**.
- ✓ And then again a $1 * 1, 512$ kernel with two more of $3 * 3, 512$ and $1 * 1, 2048$ and this was repeated 3 times giving us a total of **9 layers**.
- ✓ After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function so this gives us **1 layer**.

5. Xception

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. This network was introduced Francois Chollet who works at Google, Inc.

Xception stands for “extreme inception”, it takes the principles of Inception to an extreme. In Inception, 1×1 convolutions were used to compress the original input, and from each of those input spaces we used different type of filters on each of the depth space. Xception just reverses this step. Instead, it first applies the filters on each of the depth map and then finally compresses the input space using 1×1 convolution by applying it across the depth. This method is almost identical to a depthwise separable convolution, an operation that has been used in neural network design as early as 2014. There is one more difference between Inception and Xception. The presence or absence of a non-linearity after the first operation. In Inception model, both operations are followed by a ReLU non-linearity, however Xception doesn't introduce any non-linearity.

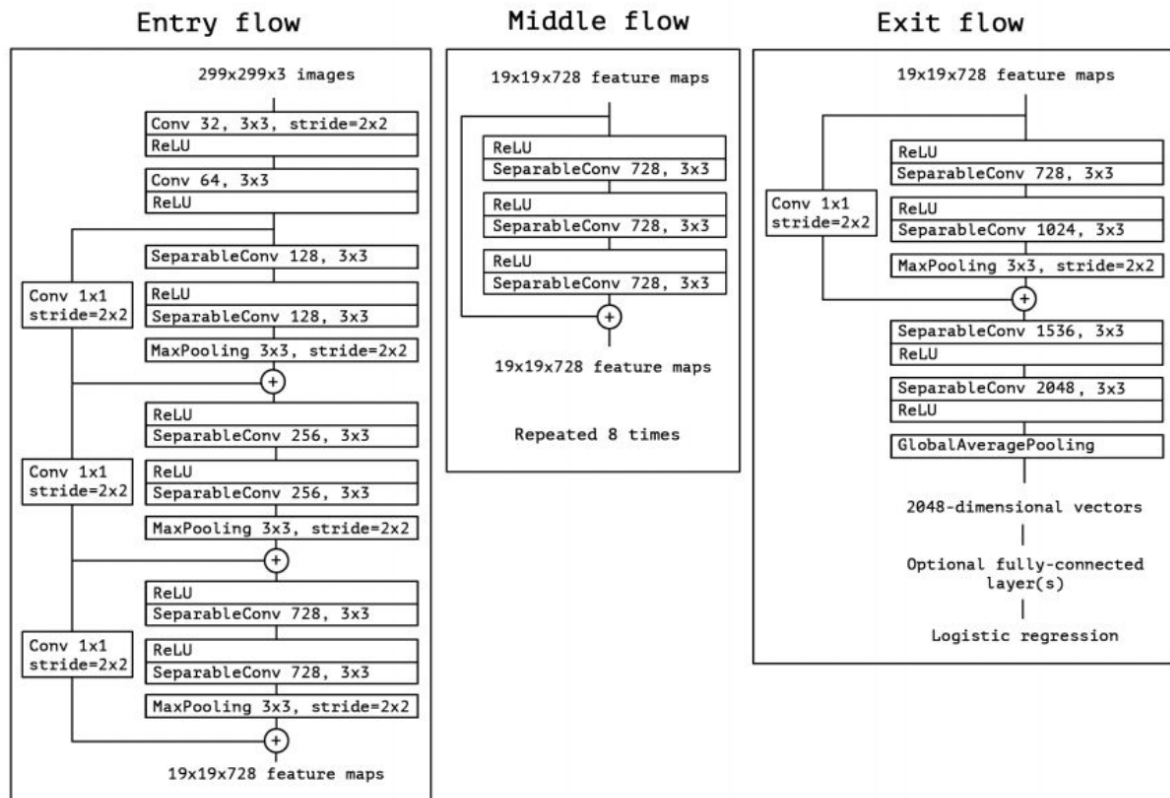


Fig 15:- Xception Architecture

The data first goes through the entry flow, then after than it. Goes through the middle flow (repeating itself 8 times in this middle flow), and finally through the exit flow.

6. InceptionV3

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification. The inception V3 is a superior version of the basic model Inception V1 which was introduced as GoogLeNet in 2014. As the name suggests it was developed by a team at Google.

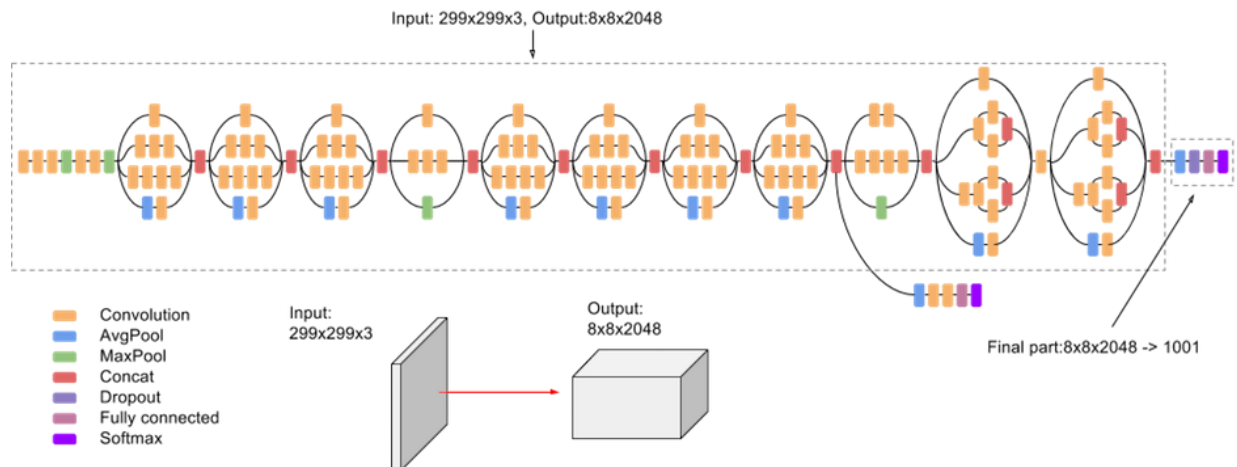


Fig 16:- Architecture of InceptionV3

In total, the inception V3 model is made up of 42 layers which is a bit higher than the previous inception V1 and V2 models. But the efficiency of this model is really impressive. That's why many researcher used this model for covid-19 detection.

7. ResNet-101

ResNet-101 is a convolutional neural network that is 101 layers deep. It is updated version of ResNet-50. we can load a pre-trained version of the network trained on more than a million images from the ImageNet database. The pre-trained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

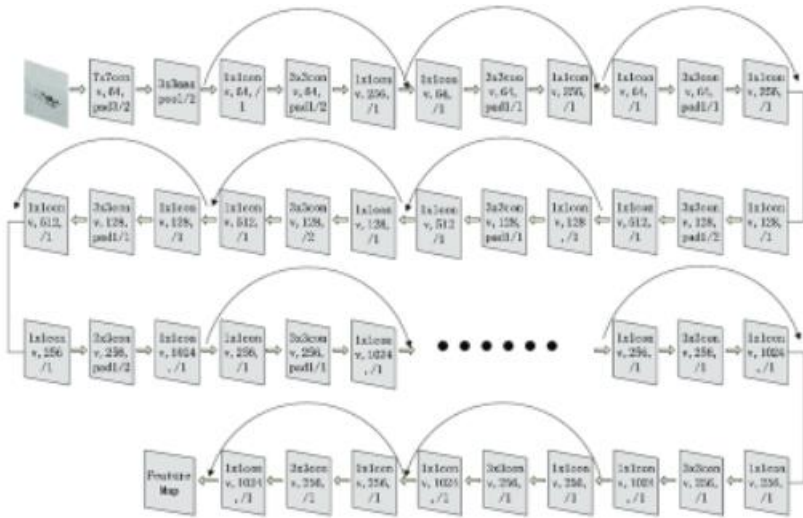


Fig 17:- Feature Extraction of ResNet-101

8. MobileNet

The MobileNet model is designed to be used in mobile application and it's TensorFlow first mobile computer vision model. MobileNet uses depth wise separable convolutions. It significantly reduces the number of parameters when compared to the network with regular convolutions with the same depth in the nets. This architecture was proposed by Google. Depth wise separable convolutions are used in MobileNet architecture to drastically reduces the number of trainable parameters in comparison to regular CNNs separable convolution deals with the both spatial dimensions along with depth dimension. MobileNet uses 3 X 3 depth-wise separable convolutions that save the computation cost up to 8 to 9 times while the reduction in accuracy is minor. A depthwise separable convolution layer with 3X3 depth-wise convolution followed by batch normalization and ReLU activation layer and a pointwise convolution followed by batch normalization and ReLU.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
	Conv dw / s2	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 1024$
	Conv dw / s2	$3 \times 3 \times 1024$ dw
	Conv / s1	$1 \times 1 \times 1024 \times 1024$
	Avg Pool / s1	Pool 7×7
	FC / s1	1024×1000
	Softmax / s1	Classifier

Fig 18:- MobileNet Architecture

9. InstaCovNet-19

InstaCovNet-19 is a deep convolutional architecture (DCNN) used for the detection of patients with COVID-19 using chest X-ray images. As there is a shortage of data consisting of COVID-19 X-rays, training models from scratch using randomly initialized weights are not very efficient and may lead to lousy variance versus bias trade-off. Therefore, to avoid these problems, Transfer Learning and multiple pre-trained DCNNs were used in this study. For the fine-tuning process, we used Inception v3, MobileNetV2, ResNet101, NASNet and Xception. The above-mentioned pre-trained models were chosen after rigorous experimentation, the results of which concluded that each of the above-mentioned pre-trained models contributes towards the Improvement in classification performance because of the unique feature extraction techniques employed by each of these models, explained in detail in later subsections. These models were first imported with their pre-trained weights matrix (on ImageNet). Then these models were fine-tuned for our dataset. The fine-tuned models were then combined using the Integrated Stacking technique, making the stacked model a larger and more robust model.

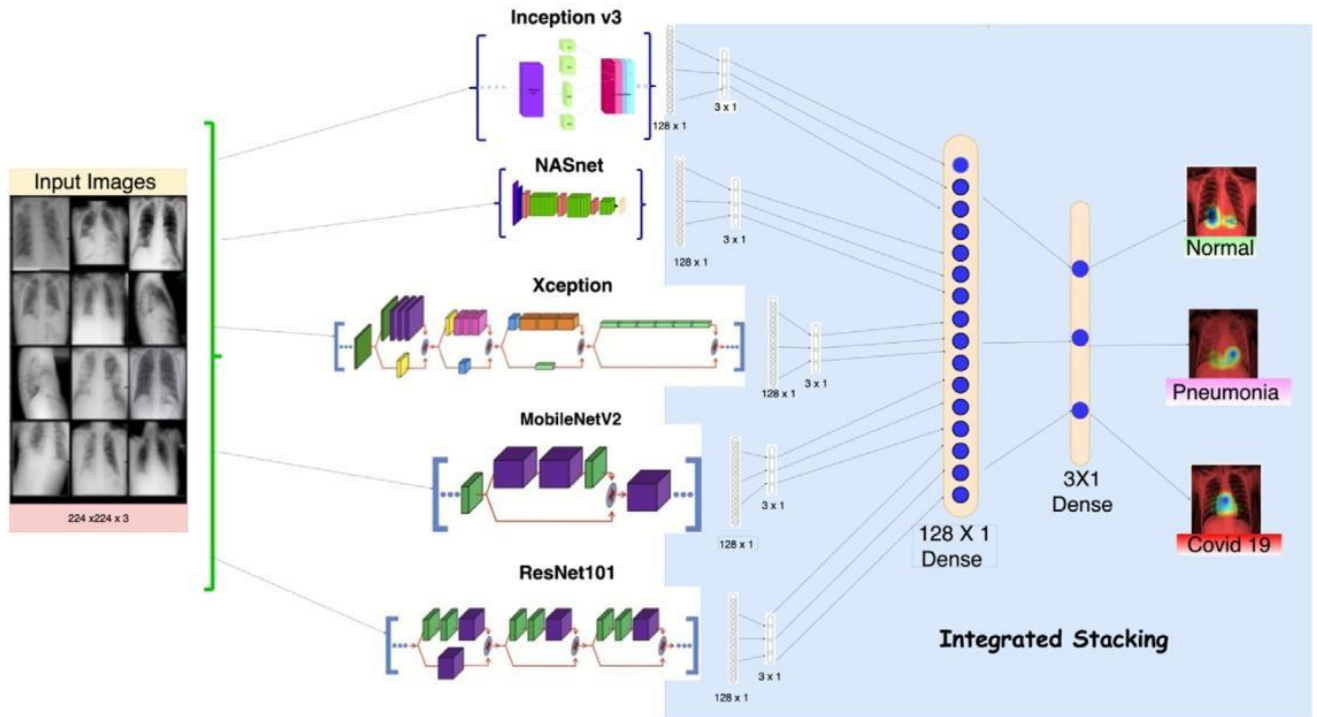


Fig 19:- InstaCovNet-19 integrated stacked model architecture

10. DarkCovidNet

Darknet-19 is the classifier model that forms the basis of a real-time object detection system named YOLO (You only look once). This system has the state-

of-the-art architecture designed for object detection. The DarkNet classifier is used on the basis of this successful architecture.

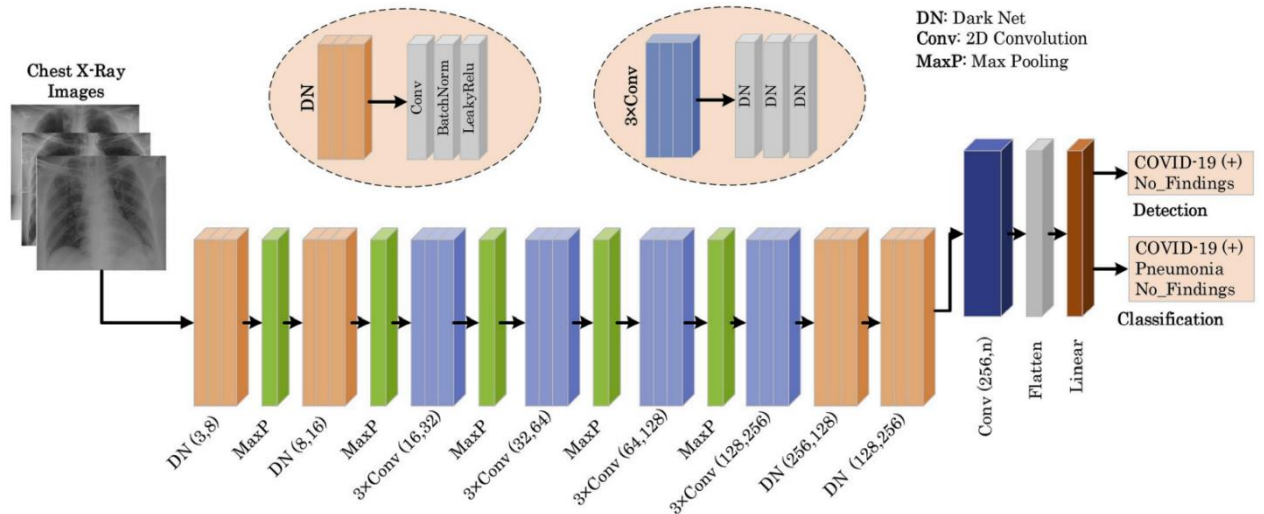


Fig 20:- DarkCovidNet Architecture

The proposed model has 17 convolution layers. Each DN (DarkNet) layer has one convolutional layer followed by BatchNorm, and LeakyReLU operations, while each 3 Conv layer has the same setup three times in successive form. The batch normalization operation is used to standardize the inputs, and this operation has other benefits, such as reducing training time and increasing stability of the model. LeakyReLU is a variation of the ReLU operation used to prevent dying neurons. Unlike ReLU or sigmoid activation functions, which have zero value in the negative part of their derivatives, LeakyReLU has a small epsilon value to overcome the dying neuron problem. Similar to the Darknet-19 model, the Maxpool method is used in all the pooling operations. Maxpool downsizes an input by taking the maximum of a region determined by its filter. When working with two classes, this models performs the COVID-19 detection task. If three different classes of images are used in the input, the same model performs the classification task to determine the labels of the input chest X-ray images as COVID-19, Pneumonia, or No-Findings.

11. DenseNet

The **DenseNet** architecture is all about modifying this standard CNN architecture. In a **DenseNet** architecture, each layer is connected to every other layer, hence the name **Densely Connected Convolutional Network**. For L layers, there are $L(L+1)/2$ direct connections. For each layer, the feature maps

of all the preceding layers are used as inputs, and its own feature maps are used as input for each subsequent layers.

This is really it, as simple as this may sound, DenseNets essentially connect every layer to every other layer. This is the main idea that is extremely powerful. The input of a layer inside **DenseNet** is the concatenation of feature maps from previous layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problems, strength feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

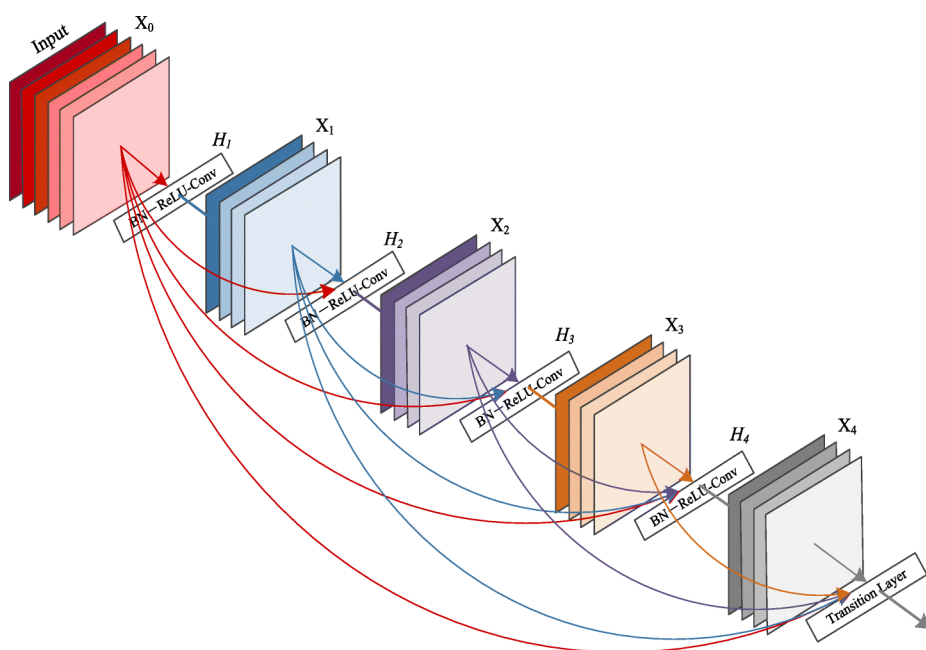


Fig 21:- DenseNet Architecture

Table 1 presents list of researcher who have worked on hand covid-19 detection Using deep learning using different algorithm and their accuracy respectively.

Citation	Model	Accuracy	Precision	F1-Score	Sensitivity
[2]	InstaCovNet-19	99.08%	99%	99%	
[3]	MobileNet and SVM	98.62%		98.451%	
[4]	CNN	100%	100%	100%	
[5]	DLM	96.43 %			93.68%
[6]	CoviAID	90.5%			100%

[7]	Xception	97%	99%	95%	93%
[7]	Inception V3	96%	96%	96%	95%
[7]	ResNext	93%	94%	95%	97%
[8]	Covid-XNet	94.43%	93.76%	93.14%	92.53%
[9]	Inception V3	95.4%	73.4%	81.1%	90.6%
[9]	ResNet 50	96.1%	76.5%	83.5%	91.8%
[9]	ResNet 101	96.1%	84.2%	81.2%	
[9]	Inception V2	94.2%	67.7%	74.8%	83.5%
[9]	ResNet-152	93.9%	74.8%	69.8%	78.3%
[10]	ResNet-50	99%	99%	99.8%	99.8%
[11]	DarkCovidNet	98.08%	98.03%	96.51%	95.13%
[12]	Stacked Ensemble	84.73%	79.13%	85.45%	92%
[13]	ResNet-50	96.5%	96.04%	96.5%	97%
[13]	CoroNet	93.5%	93.63%	91.77%	90%
[13]	DenseNet	96.4%	96%	96%	96%
[13]	CNN	91.21%	90.47%	90.5%	90.52%
[13]	VGG	95%	95.5%	92.7%	90%

Table 1:- List of Research Work on Covid-19 Detection

IV. FEASIBILITY STUDY

Feasibility study can help you determine whether you should proceed with your project. It is essential to evaluate cost and benefit. It is essential to evaluate cost and benefit of the proposed system. Five types of feasibility study are taken into consideration.

1. Technical Feasibility

It includes finding out technologies for the project, both hardware and software. For covid-19 detection, user must have a pc to see a results. Pcs are very cheap now a days and everyone generally possess them. Besides, system needs chest x-ray images. It is also not an issue in this era where almost every hospitals or medical store has X-ray machine.

2. Operational feasibility

It is the ease and simplicity of operation of proposed system. System does not require any special skill set for users to operate it. In fact, it is designed to be used by almost everyone. Kids who still don't know to write can read out problems for system and get answers. But if user want more details he/she must a domain knowledge.

3. Economic feasibility

Here, we find the total cost and benefit of the proposed system over current system. For this project, the main cost is X-ray cost. User have to pay around more than 2.5\$ which is quite feasible than cost of RT-PCR which is more than 8.5\$. Again, using deep learning models is economical and fast. As far as maintenance is concerned, deep learning models re-train time to time.

4. Organizational feasibility

This shows the management and organizational structure of the project. This project is not built by a team. The management tasks are all to be carried out by a single person. That won't create any management issues and will increase the feasibility of the project.

5. Cultural feasibility

It deals with compatibility of the project with cultural environment. Covid-19 detection is built in accordance with the general culture. The project is named to represent Indian culture without undermining local beliefs.

This project is technically feasible with no external hardware requirements. Also, it is simple in operation and does not cost training or repairs. Overall feasibility study of the project reveals that the goals of the proposed system are achievable. Decision is taken to proceed with the project.

Software Requirements:

- ✓ Python and python IDE

- ✓ Package version same as requirements.txt file

Hardware Requirements:

- ✓ Ram: 8 GB
- ✓ SSD : 256 GB
- ✓ Processor : i7 9th generations
- ✓ GPU : min. 4 GHz

V. METHODOLOGY/PLANNING OF WORK

This section explains details of methodology being used in software development. The project methodology is important because it helps to organize investigation in a scientific way to overcome problems, structure, plan, and control the process of developing an information system. This chapter will explain about rule-based system. The system will use Iterative Model development as a framework methodology. After system has been completely developed, it should be tested to make sure it achieves the objectives of the project.

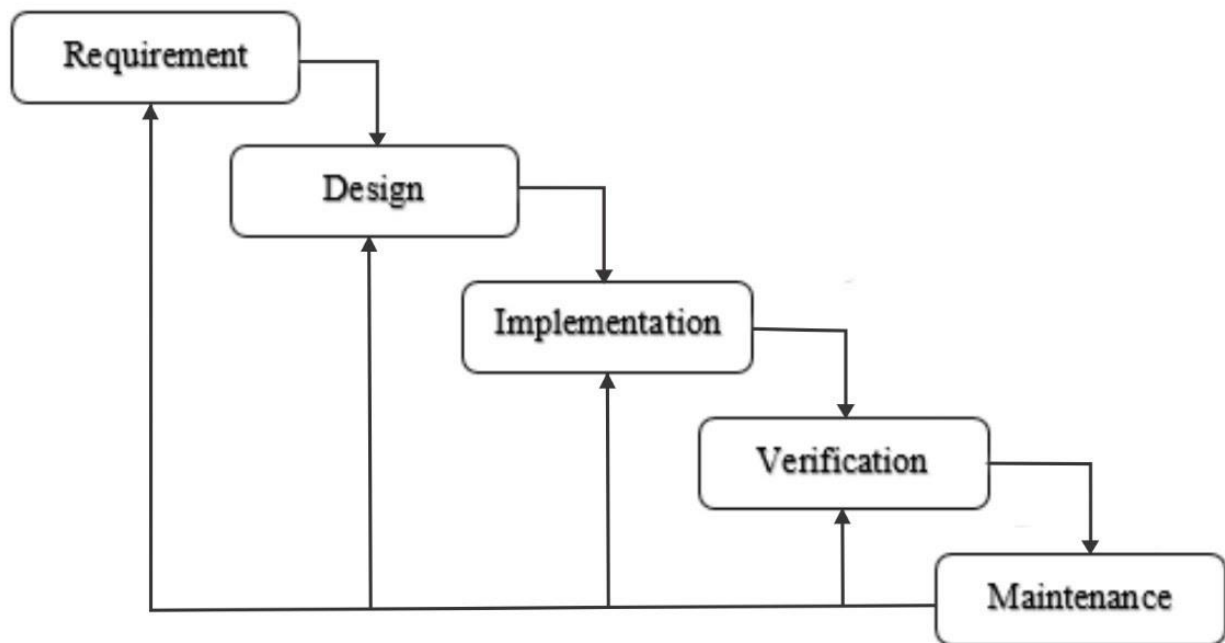


Fig 22:- Waterfall Model Architecture

Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document. For the covid-19 detection I collected the more than 100 research paper and filter out almost from them. In this phase I found out why this project is necessary for society and human.

System Design –The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture. For covid-19 detection using deep learning there are lot of models available like CNN, ResNet, DenseNet, VGG and choosing few of them according to our needs is very much tough. In this phase I selected three models, CNN, VGG-16 and ResNet-50.

Implementation –With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing. It is one of the hard task to complete. During these phase I implemented four models two from scratch and rest are pre trained models.

Verification and Testing – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures. In this phase I compared my result with actual result and models accuracy metric.

Maintenance – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment. If client want dynamic models based on the latest images, we have to re-train our models.

For building a deep learning models I have to follow some steps which is preliminary steps for such kinds of models.

1. Data Collection

It is one of the first and important task to do. For deep learning models I have to find the data of chest X-ray images which contains normal and covid-19 patient. I took the data from kaggle(<https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia>) of size around 2 GB and filter out according to my requirements. The size of the data is the most important factor influencing the performance of any deep learning models. If dataset is too small it may leads to model in overfitting. Since covid-19 is always changeable, only limited number of

datasets are publically available. Dataset that are used by previous researcher are:-

CITATION	NO. OF IMAGE	DATASET TYPE	NO. OF CLASSES
[2]	3047	CXR	3
[3]	388	CXR	2
[4]	1000	CXR	2
[5]	10040	CXR	3
[6]		CXR	3
[7]	6432	CXR	2
[9]	3141	CT	2
[10]	5155	CXR	3
[13]	1500	CT	4

Table 2:- List of Dataset which are previously used

2. Data Pre-processing

Since data often comes from multiple sources, a method to guide the complexity and accuracy is required. Image preprocessing ensures complexity reductions and better accuracy fielded from certain data. The main steps for image preprocessing are:-

Image standardization

Neural network that deals with images need unified aspect ratio images. That's why it is necessary to resize image. The standard size of image is (224, 224, 3).

Normalization

It is a scaling technique in deep learning applied during data preparation to change the value of numeric columns in the datasets to use a common scale. Normalization is the action of subtracting the mean of the distribution from each pixel and dividing by standard deviation.

3. Image augmentation

Data augmentation is a techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizers and helps reduce over fitting when training a machine learning model. Augmentation technique are employed

efficiently when data availability and quantity require it. Operations in data augmentations are,

Rotation

Image is rotated within a interval of certain degree. For covid-19 data set degree of rotation between -10 to 20 helps us to increase accuracy of models.

Zoom

Scaling the image by zooming in or out would also increase the accuracy of models.

Shear

Shearing used to transform the orientation of the image.

Gaussian Blur

Using a Gaussian blur filter, high frequency factors can be eliminated, causing a blurred version of an image.

4. Deep Learning Models

It is a machine learning method which composes details together to obtain more abstract, higher level, features of the data through composition of mathematical functions. Powerful modern deep learning algorithms often involve a large number of these levels. For covid-19 detection I created a four different deep learning architecture and compares their results and accuracy.

VI. PROJECT DESIGN FLOW

I used four different deep learning architecture to solve a covid-19 detection problem, two of them are customized and rest are pre-trained architecture.

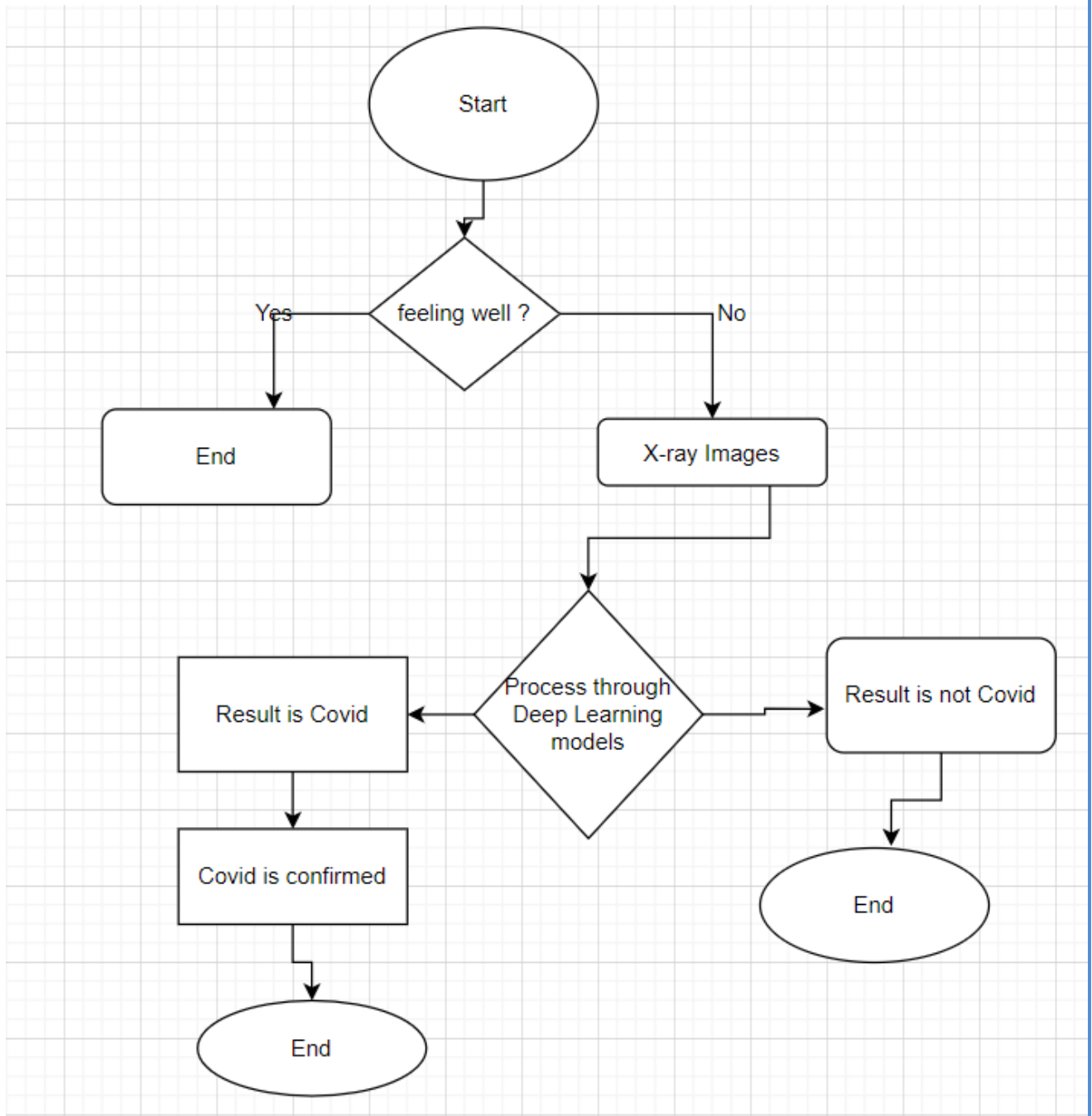


Fig 23:- Activity Diagram for Covid-19 Detection

For the covid-19 detection, problems always starts with am I feeling well, if he/she is not feeling well and symptoms is similar to covid-19 then only he/she is going for chest x-ray after that x-ray image will be processed through deep

learning models then he/she will get their results, if it is a negative then no need to worry but if it is a positive he/she should government general guideline for covid-19 prevention.

For the x-ray image processing all the deep learning models follow same style which is taking input as x-ray image and process it through deep leaning models.

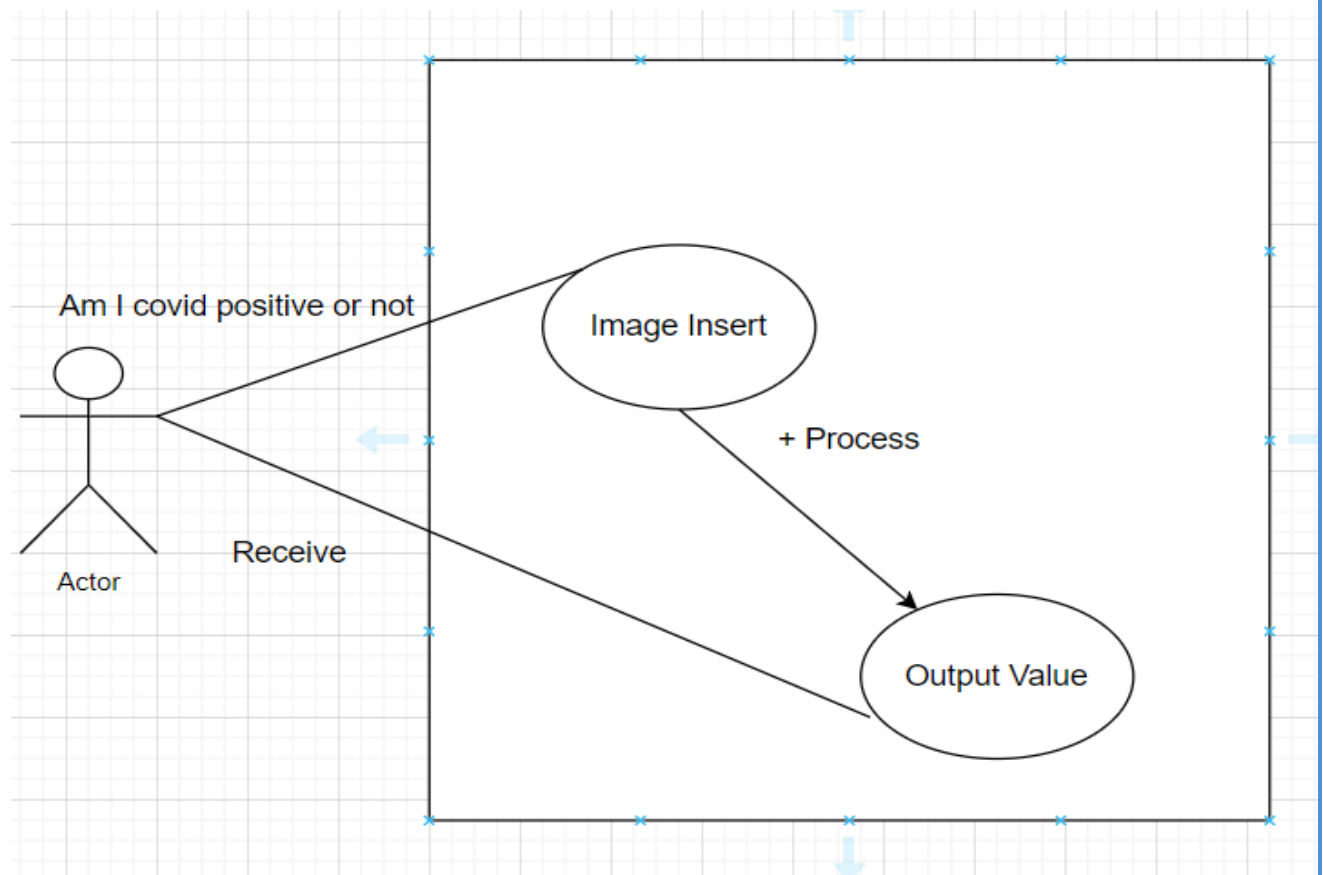


Fig 24:- Use Case Diagram

In this project there is only one user. The user queries command to the system. System then interprets it and fetches results. Then response is sent back to the user.

I proposed four different architecture to solve this problem but black box view of all models are same. And list of proposed models are as follow.

1. VGG-16
2. ResNet-50
3. CNN
4. CNN (Local Maxima)

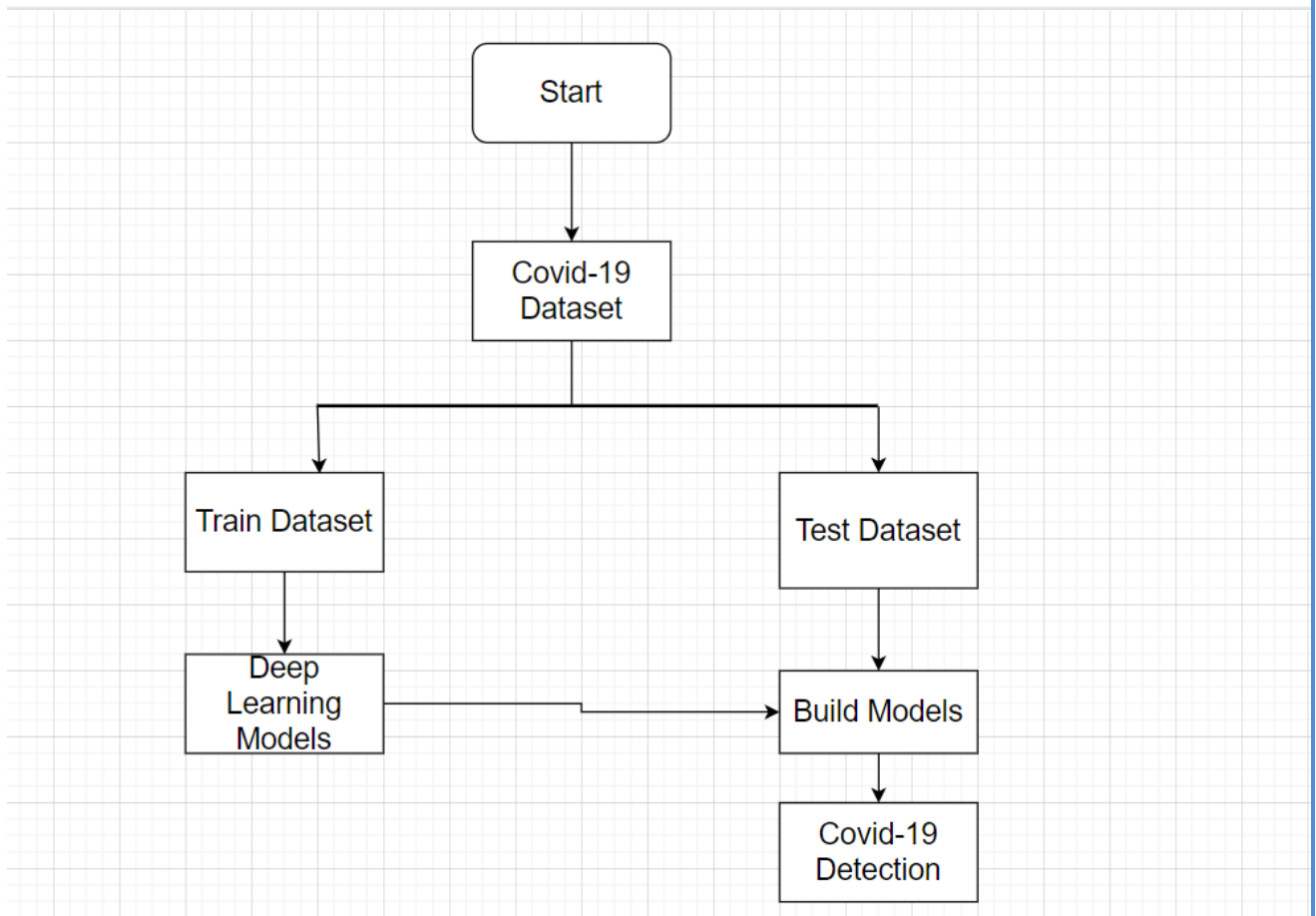


Fig 25:- Black Box View for All Models

I. VGG-16

VGG-16 was proposed by Karen Simonyan and Andrew Zisserman of the

Visual Geometry Group Lab of Oxford University in 2014 in the paper “very deep convolutional networks for large scale image recognition”. The VGG architecture was originally proposed for image recognition application. In vgg 16 and 19 wt. layers are used with a smaller convolutional filter size of 3*3. The network won first and second places in the ILSVR(ImageNet) competition in 2014. This models achieves 92.7% top-5 test accuracy on ImageNet dataset which contain 14 million images belonging to 1000 unique classes.

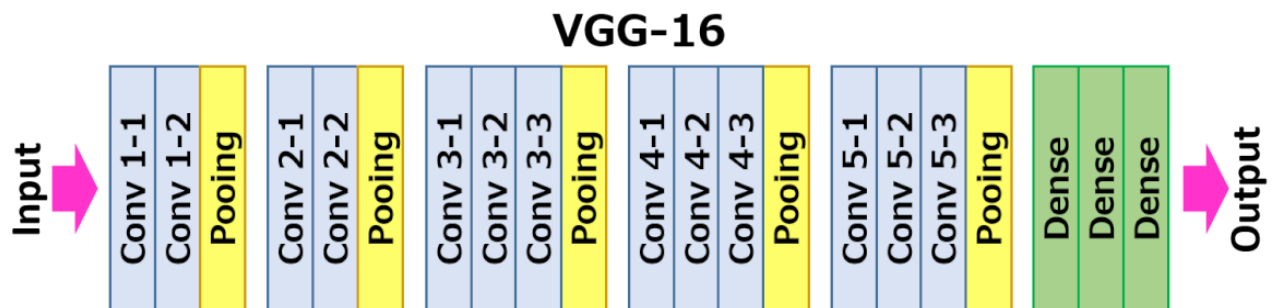


Fig 26:- VGG-16 General Architecture

VGGNet consists of 138 million parameters, which can be a bit challenging to handle. VGG can be achieved through transfer learning. In which the model is pre trained on a dataset and the parameters are updated for better accuracy and you can use the parameters values. We can see that there are 16 layers in VGG-16 models they are:-

16 layers of VGG16

1. Convolution using 64 filters
2. Convolution using 64 filters + Max pooling
3. Convolution using 128 filters
4. Convolution using 128 filters + Max pooling
5. Convolution using 256 filters

6. Convolution using 256 filters
7. Convolution using 256 filters + Max pooling
8. Convolution using 512 filters
9. Convolution using 512 filters
10. Convolution using 512 filters+Max pooling
11. Convolution using 512 filters
12. Convolution using 512 filters
13. Convolution using 512 filters+Max pooling
14. Fully connected with 4096 nodes
15. Fully connected with 4096 nodes
16. Output layer with Softmax activation with 1000 nodes.

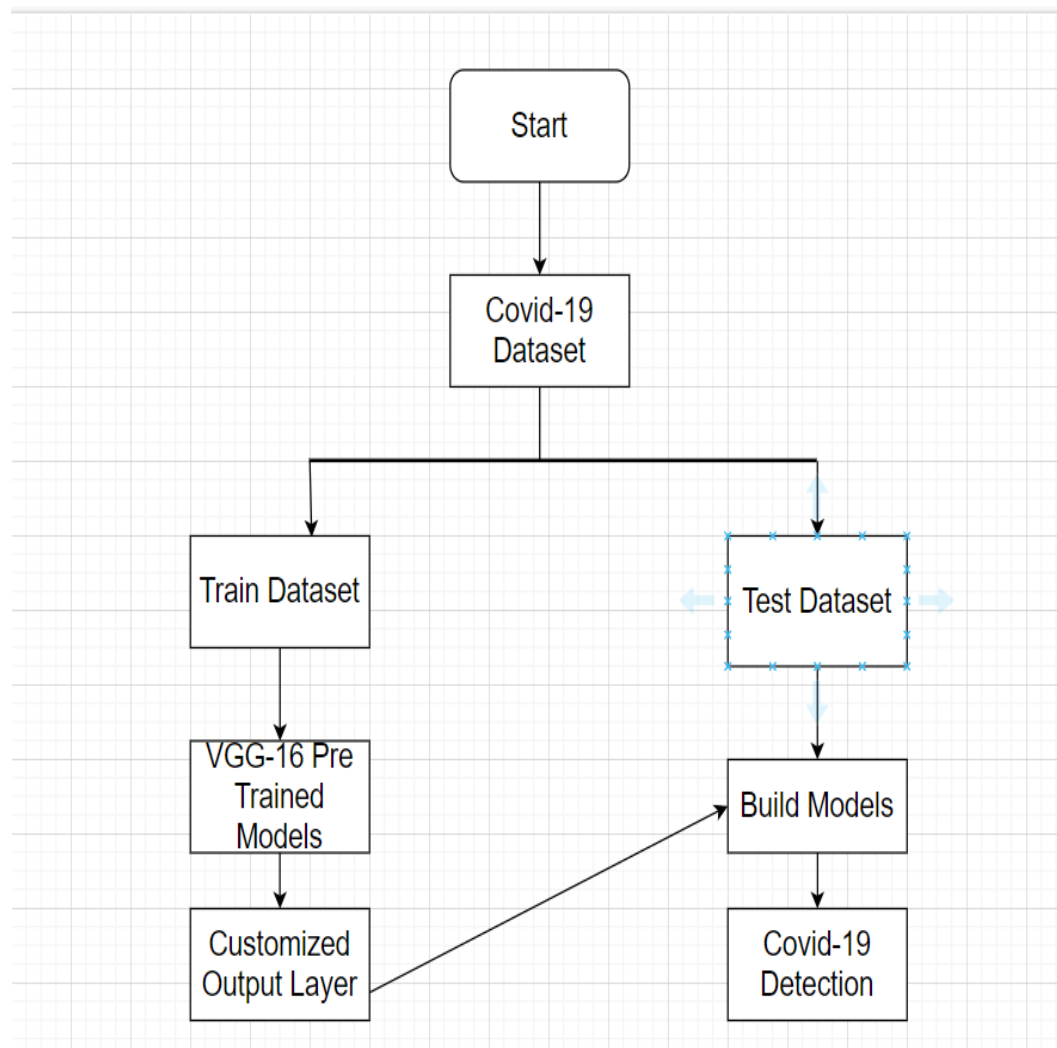


Fig 27:- Working Flow of VGG-16 Model

Model: "model"

Layer (type)	Output Shape	Param #
=====		
===		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0

block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
predictions (Dense)	(None, 2)	50178

=====

===

Total params: 14,764,866

Trainable params: 50,178

Non-trainable params: 14,714,688

Table 3:- VGG-16 Model Summary

In the proposed models there are initially 14,764,866 parameters but only 50,178 parameters need to be train which save our resources. VGG-16 is derived from CNN models. After dividing data set and pre-processing of image dataset. I started to train my models with the epochs of 30.

Epoch 1/30

2022-05-07 23:54:37.719804: I tensorflow/stream_executor/cuda/cuda_dnn.cc:368]

Loaded cuDNN version 8202

2022-05-07 23:54:39.313285: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 3.46GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:39.313314: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 3.46GiB with freed_by_count=0. The caller indicates

that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:39.313324: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 3.04GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:39.313332: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 3.04GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:39.508085: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.74GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:39.508116: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.74GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:40.346214: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.74GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:40.346243: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1.74GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:40.346253: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.55GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 23:54:40.346261: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.55GiB with freed_by_count=0. The caller indicates

that this is not a failure, but may mean that there could be performance gains if more memory were available.

10/10 [=====] - ETA: 0s - loss: 0.7088 - accuracy: 0.7531

Epoch 1: val_accuracy improved from -inf to 0.92148, saving model to bestmodel.h5

10/10 [=====] - 27s 2s/step - loss: 0.7088 - accuracy: 0.7531 - val_loss: 0.1453 - val_accuracy: 0.9215

Epoch 2/30

10/10 [=====] - ETA: 0s - loss: 0.1659 - accuracy: 0.9344

Epoch 2: val_accuracy improved from 0.92148 to 0.93995, saving model to bestmodel.h5

10/10 [=====] - 18s 2s/step - loss: 0.1659 - accuracy: 0.9344 - val_loss: 0.1211 - val_accuracy: 0.9400

Epoch 3/30

10/10 [=====] - ETA: 0s - loss: 0.0496 - accuracy: 0.9811

Epoch 3: val_accuracy improved from 0.93995 to 0.96305, saving model to bestmodel.h5

10/10 [=====] - 21s 2s/step - loss: 0.0496 - accuracy: 0.9811 - val_loss: 0.0803 - val_accuracy: 0.9630

Epoch 4/30

10/10 [=====] - ETA: 0s - loss: 0.0380 - accuracy: 0.9844

Epoch 4: val_accuracy improved from 0.96305 to 0.98614, saving model to bestmodel.h5

10/10 [=====] - 18s 2s/step - loss: 0.0380 - accuracy: 0.9844 - val_loss: 0.0309 - val_accuracy: 0.9861

Epoch 5/30

10/10 [=====] - ETA: 0s - loss: 0.0376 - accuracy: 0.9812

Epoch 5: val_accuracy did not improve from 0.98614

10/10 [=====] - 18s 2s/step - loss: 0.0376 - accuracy: 0.9812 - val_loss: 0.0343 - val_accuracy: 0.9838

Epoch 6/30

10/10 [=====] - ETA: 0s - loss: 0.0240 - accuracy: 0.9937

Epoch 6: val_accuracy improved from 0.98614 to 0.99538, saving model to bestmodel.h5

10/10 [=====] - 20s 2s/step - loss: 0.0240 - accuracy: 0.9937 - val_loss: 0.0293 - val_accuracy: 0.9954
 Epoch 7/30
 10/10 [=====] - ETA: 0s - loss: 0.0328 - accuracy: 0.9875
 Epoch 7: val_accuracy did not improve from 0.99538
 10/10 [=====] - 18s 2s/step - loss: 0.0328 - accuracy: 0.9875 - val_loss: 0.0453 - val_accuracy: 0.9792
 Epoch 7: early stopping

Table 4:- VGG-16 Model Training Status

For VGG-16 Pre-trained models I used the concept of Early Stopping which help me to save computer resource like CPU, GPU, RAM that's why my model is stopped at 7 epoch which help me to reduce over fitting .

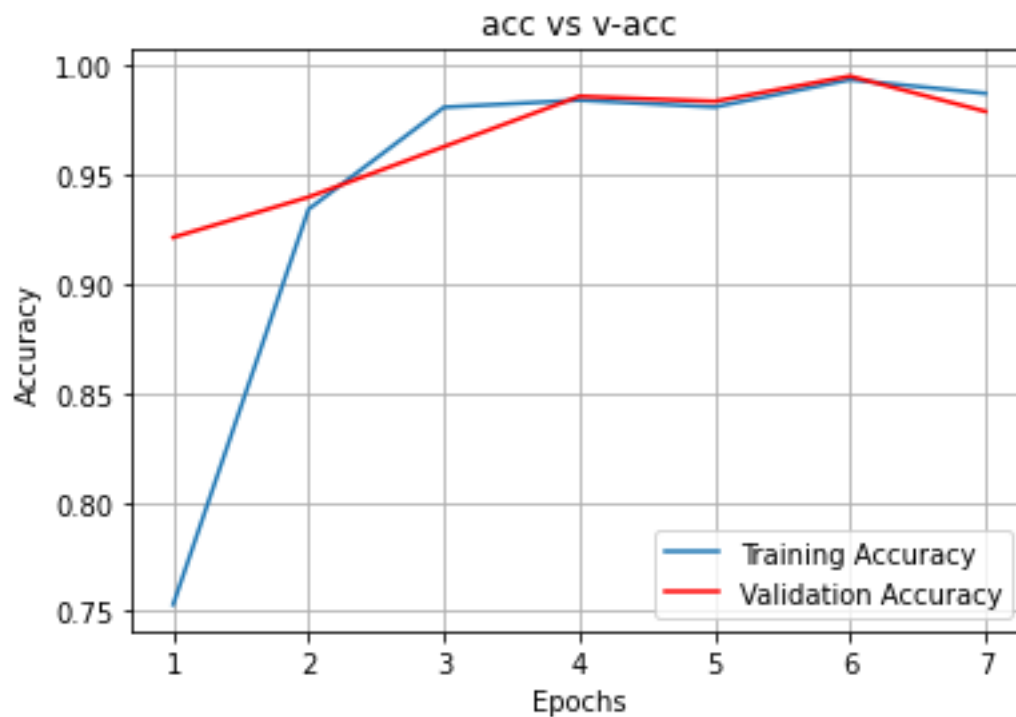


Fig 28:- VGG-16 Model Training vs Test Accuracy

For this models I only need train my models up to just 7 epochs with that I am able to get validation accuracy 97.92 % and training accuracy is 98.75 %.

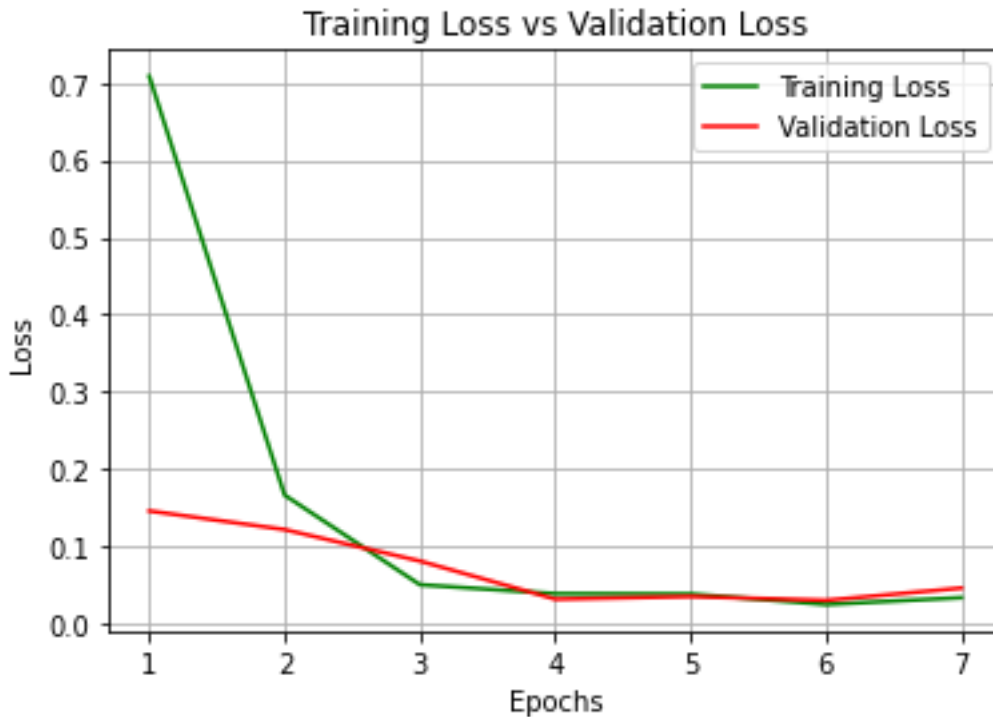


Fig 29:- VGG-16 Model Training and Validation Loss Comparison

From figure 29, it shows that training loss and validation for this model is very less like 3-4 % which great. From figure 28 and 29 we can observe that model perform excellent with training set as well as validation set. Accuracy of VGG-16 proposed model is 99.53%.

II. ResNet-50

ResNet-50 model is a convolutional neural network (CNN) that is 50 layers deep. A Residual Neural Network (ResNet) is an Artificial Neural Network (ANN) of a kind that stacks residual blocks on top of each other to form a

network.

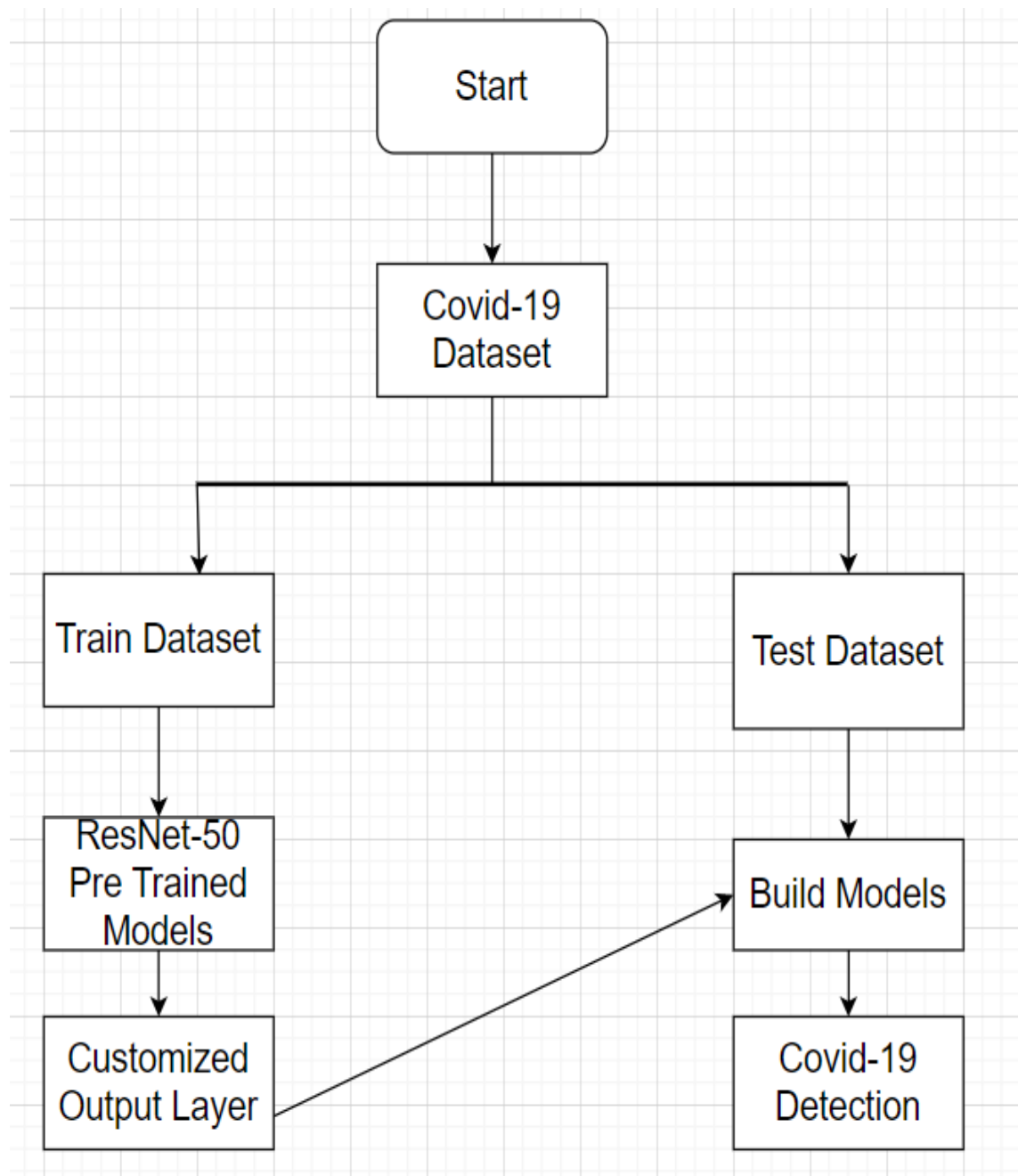


Fig 30:- Working Flow of ResNet-50 Model

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
50			

```

=====
==
input_1 (InputLayer)      [(None, 224, 224, 3) 0] []

conv1_pad (ZeroPadding2D) (None, 230, 230, 3) 0 ['input_1[0][0]']

conv1_conv (Conv2D)        (None, 112, 112, 64) 9472 ['conv1_pad[0][0]']

conv1_bn (BatchNormalization) (None, 112, 112, 64) 256 ['conv1_conv[0][0]']

conv1_relu (Activation)    (None, 112, 112, 64) 0 ['conv1_bn[0][0]']

pool1_pad (ZeroPadding2D) (None, 114, 114, 64) 0 ['conv1_relu[0][0]']

pool1_pool (MaxPooling2D) (None, 56, 56, 64) 0 ['pool1_pad[0][0]']

conv2_block1_1_conv (Conv2D) (None, 56, 56, 64) 4160 ['pool1_pool[0][0]']

conv2_block1_1_bn (BatchNormalization) (None, 56, 56, 64) 256 ['conv2_block1_1_conv[0][0]']

conv2_block1_1_relu (Activation) (None, 56, 56, 64) 0 ['conv2_block1_1_bn[0][0]']

conv2_block1_2_conv (Conv2D) (None, 56, 56, 64) 36928 ['conv2_block1_1_relu[0][0]']

conv2_block1_2_bn (BatchNormalization) (None, 56, 56, 64) 256 ['conv2_block1_2_conv[0][0]']

conv2_block1_2_relu (Activation) (None, 56, 56, 64) 0 ['conv2_block1_2_bn[0][0]']

conv2_block1_0_conv (Conv2D) (None, 56, 56, 256) 16640 ['pool1_pool[0][0]']

conv2_block1_3_conv (Conv2D) (None, 56, 56, 256) 16640 ['conv2_block1_2_relu[0][0]']

conv2_block1_0_bn (BatchNormalization) (None, 56, 56, 256) 1024 ['conv2_block1_0_conv[0][0]']

conv2_block1_3_bn (BatchNormalization) (None, 56, 56, 256) 1024 ['conv2_block1_3_conv[0][0]']

conv2_block1_add (Add)      (None, 56, 56, 256) 0 ['conv2_block1_0_bn[0][0]',
                                     'conv2_block1_3_bn[0][0]']

conv2_block1_out (Activation) (None, 56, 56, 256) 0 ['conv2_block1_add[0][0]']

conv2_block2_1_conv (Conv2D) (None, 56, 56, 64) 16448 ['conv2_block1_out[0][0]']

conv2_block2_1_bn (BatchNormalization) (None, 56, 56, 64) 256 ['conv2_block2_1_conv[0][0]']

conv2_block2_1_relu (Activation) (None, 56, 56, 64) 0 ['conv2_block2_1_bn[0][0]']

conv2_block2_2_conv (Conv2D) (None, 56, 56, 64) 36928 ['conv2_block2_1_relu[0][0]']

conv2_block2_2_bn (BatchNormalization) (None, 56, 56, 64) 256 ['conv2_block2_2_conv[0][0]']

```

ization)

conv2_block2_2_relu (Activation) (None, 56, 56, 64) 0 ['conv2_block2_2_bn[0][0]']

conv2_block2_3_conv (Conv2D) (None, 56, 56, 256) 16640 ['conv2_block2_2_relu[0][0]']

conv2_block2_3_bn (BatchNormal (None, 56, 56, 256) 1024 ['conv2_block2_3_conv[0][0]']
ization)

conv2_block2_add (Add) (None, 56, 56, 256) 0 ['conv2_block1_out[0][0]',
'conv2_block2_3_bn[0][0]']

conv2_block2_out (Activation) (None, 56, 56, 256) 0 ['conv2_block2_add[0][0]']

conv2_block3_1_conv (Conv2D) (None, 56, 56, 64) 16448 ['conv2_block2_out[0][0]']

conv2_block3_1_bn (BatchNormal (None, 56, 56, 64) 256 ['conv2_block3_1_conv[0][0]']
ization)

conv2_block3_1_relu (Activation) (None, 56, 56, 64) 0 ['conv2_block3_1_bn[0][0]']

conv2_block3_2_conv (Conv2D) (None, 56, 56, 64) 36928 ['conv2_block3_1_relu[0][0]']

conv2_block3_2_bn (BatchNormal (None, 56, 56, 64) 256 ['conv2_block3_2_conv[0][0]']
ization)

conv2_block3_2_relu (Activation) (None, 56, 56, 64) 0 ['conv2_block3_2_bn[0][0]']

conv2_block3_3_conv (Conv2D) (None, 56, 56, 256) 16640 ['conv2_block3_2_relu[0][0]']

conv2_block3_3_bn (BatchNormal (None, 56, 56, 256) 1024 ['conv2_block3_3_conv[0][0]']
ization)

conv2_block3_add (Add) (None, 56, 56, 256) 0 ['conv2_block2_out[0][0]',
'conv2_block3_3_bn[0][0]']

conv2_block3_out (Activation) (None, 56, 56, 256) 0 ['conv2_block3_add[0][0]']

conv3_block1_1_conv (Conv2D) (None, 28, 28, 128) 32896 ['conv2_block3_out[0][0]']

conv3_block1_1_bn (BatchNormal (None, 28, 28, 128) 512 ['conv3_block1_1_conv[0][0]']
ization)

conv3_block1_1_relu (Activation) (None, 28, 28, 128) 0 ['conv3_block1_1_bn[0][0]']

conv3_block1_2_conv (Conv2D) (None, 28, 28, 128) 147584 ['conv3_block1_1_relu[0][0]']

conv3_block1_2_bn (BatchNormal (None, 28, 28, 128) 512 ['conv3_block1_2_conv[0][0]']
ization)

conv3_block1_2_relu (Activation) (None, 28, 28, 128) 0 ['conv3_block1_2_bn[0][0]']

conv3_block1_0_conv (Conv2D) (None, 28, 28, 512) 131584 ['conv2_block3_out[0][0]']

conv3_block1_3_conv (Conv2D) (None, 28, 28, 512) 66048 ['conv3_block1_2_relu[0][0]']

conv3_block1_0_bn (BatchNormal (None, 28, 28, 512) 2048 ['conv3_block1_0_conv[0][0]']
ization)

conv3_block1_3_bn (BatchNormal (None, 28, 28, 512) 2048 ['conv3_block1_3_conv[0][0]']
ization)

conv3_block1_add (Add) (None, 28, 28, 512) 0 ['conv3_block1_0_bn[0][0]',
'conv3_block1_3_bn[0][0]']

conv3_block1_out (Activation) (None, 28, 28, 512) 0 ['conv3_block1_add[0][0]']

conv3_block2_1_conv (Conv2D) (None, 28, 28, 128) 65664 ['conv3_block1_out[0][0]']

conv3_block2_1_bn (BatchNormal (None, 28, 28, 128) 512 ['conv3_block2_1_conv[0][0]']
ization)

conv3_block2_1_relu (Activatio (None, 28, 28, 128) 0 ['conv3_block2_1_bn[0][0]']
n)

conv3_block2_2_conv (Conv2D) (None, 28, 28, 128) 147584 ['conv3_block2_1_relu[0][0]']

conv3_block2_2_bn (BatchNormal (None, 28, 28, 128) 512 ['conv3_block2_2_conv[0][0]']
ization)

conv3_block2_2_relu (Activatio (None, 28, 28, 128) 0 ['conv3_block2_2_bn[0][0]']
n)

conv3_block2_3_conv (Conv2D) (None, 28, 28, 512) 66048 ['conv3_block2_2_relu[0][0]']

conv3_block2_3_bn (BatchNormal (None, 28, 28, 512) 2048 ['conv3_block2_3_conv[0][0]']
ization)

conv3_block2_add (Add) (None, 28, 28, 512) 0 ['conv3_block1_out[0][0]',
'conv3_block2_3_bn[0][0]']

conv3_block2_out (Activation) (None, 28, 28, 512) 0 ['conv3_block2_add[0][0]']

conv3_block3_1_conv (Conv2D) (None, 28, 28, 128) 65664 ['conv3_block2_out[0][0]']

conv3_block3_1_bn (BatchNormal (None, 28, 28, 128) 512 ['conv3_block3_1_conv[0][0]']
ization)

conv3_block3_1_relu (Activatio (None, 28, 28, 128) 0 ['conv3_block3_1_bn[0][0]']
n)

conv3_block3_2_conv (Conv2D) (None, 28, 28, 128) 147584 ['conv3_block3_1_relu[0][0]']

conv3_block3_2_bn (BatchNormal (None, 28, 28, 128) 512 ['conv3_block3_2_conv[0][0]']
ization)

conv3_block3_2_relu (Activatio (None, 28, 28, 128) 0 ['conv3_block3_2_bn[0][0]']
n)

conv3_block3_3_conv (Conv2D) (None, 28, 28, 512) 66048 ['conv3_block3_2_relu[0][0]']

conv3_block3_3_bn (BatchNormal (None, 28, 28, 512) 2048 ['conv3_block3_3_conv[0][0]']
ization)

conv3_block3_add (Add) (None, 28, 28, 512) 0 ['conv3_block2_out[0][0]',
'conv3_block3_3_bn[0][0]']

conv3_block3_out (Activation) (None, 28, 28, 512) 0 ['conv3_block3_add[0][0]']

conv3_block4_1_conv (Conv2D) (None, 28, 28, 128) 65664 ['conv3_block3_out[0][0]']

```

conv3_block4_1_bn (BatchNormal (None, 28, 28, 128) 512      ['conv3_block4_1_conv[0][0]']
ization)

conv3_block4_1_relu (Activatio (None, 28, 28, 128) 0        ['conv3_block4_1_bn[0][0]']
n)

conv3_block4_2_conv (Conv2D) (None, 28, 28, 128) 147584    ['conv3_block4_1_relu[0][0]']

conv3_block4_2_bn (BatchNormal (None, 28, 28, 128) 512      ['conv3_block4_2_conv[0][0]']
ization)

conv3_block4_2_relu (Activatio (None, 28, 28, 128) 0        ['conv3_block4_2_bn[0][0]']
n)

conv3_block4_3_conv (Conv2D) (None, 28, 28, 512) 66048      ['conv3_block4_2_relu[0][0]']

conv3_block4_3_bn (BatchNormal (None, 28, 28, 512) 2048     ['conv3_block4_3_conv[0][0]']
ization)

conv3_block4_add (Add) (None, 28, 28, 512) 0                ['conv3_block3_out[0][0]',
'conv3_block4_3_bn[0][0]']

conv3_block4_out (Activation) (None, 28, 28, 512) 0          ['conv3_block4_add[0][0]']

conv4_block1_1_conv (Conv2D) (None, 14, 14, 256) 131328    ['conv3_block4_out[0][0]']

conv4_block1_1_bn (BatchNormal (None, 14, 14, 256) 1024     ['conv4_block1_1_conv[0][0]']
ization)

conv4_block1_1_relu (Activatio (None, 14, 14, 256) 0        ['conv4_block1_1_bn[0][0]']
n)

conv4_block1_2_conv (Conv2D) (None, 14, 14, 256) 590080    ['conv4_block1_1_relu[0][0]']

conv4_block1_2_bn (BatchNormal (None, 14, 14, 256) 1024     ['conv4_block1_2_conv[0][0]']
ization)

conv4_block1_2_relu (Activatio (None, 14, 14, 256) 0        ['conv4_block1_2_bn[0][0]']
n)

conv4_block1_0_conv (Conv2D) (None, 14, 14, 1024 525312    ['conv3_block4_out[0][0]']
)

conv4_block1_3_conv (Conv2D) (None, 14, 14, 1024 263168    ['conv4_block1_2_relu[0][0]']
)

conv4_block1_0_bn (BatchNormal (None, 14, 14, 1024 4096     ['conv4_block1_0_conv[0][0]']
ization)

conv4_block1_3_bn (BatchNormal (None, 14, 14, 1024 4096     ['conv4_block1_3_conv[0][0]']
ization)

conv4_block1_add (Add) (None, 14, 14, 1024 0                ['conv4_block1_0_bn[0][0]',
'conv4_block1_3_bn[0][0]']
)

conv4_block1_out (Activation) (None, 14, 14, 1024 0          ['conv4_block1_add[0][0]']
)

conv4_block2_1_conv (Conv2D) (None, 14, 14, 256) 262400    ['conv4_block1_out[0][0]']

conv4_block2_1_bn (BatchNormal (None, 14, 14, 256) 1024     ['conv4_block2_1_conv[0][0]']
ization)

```

```

conv4_block2_1_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block2_1_bn[0][0]']
n)

conv4_block2_2_conv (Conv2D) (None, 14, 14, 256) 590080      ['conv4_block2_1_relu[0][0]']

conv4_block2_2_bn (BatchNormal (None, 14, 14, 256) 1024      ['conv4_block2_2_conv[0][0]']
ization)

conv4_block2_2_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block2_2_bn[0][0]']
n)

conv4_block2_3_conv (Conv2D) (None, 14, 14, 1024 263168      ['conv4_block2_2_relu[0][0]']
)

conv4_block2_3_bn (BatchNormal (None, 14, 14, 1024 4096      ['conv4_block2_3_conv[0][0]']
ization)

conv4_block2_add (Add) (None, 14, 14, 1024 0      ['conv4_block1_out[0][0]',
)                                     'conv4_block2_3_bn[0][0]']

conv4_block2_out (Activation) (None, 14, 14, 1024 0      ['conv4_block2_add[0][0]']
)

conv4_block3_1_conv (Conv2D) (None, 14, 14, 256) 262400      ['conv4_block2_out[0][0]']

conv4_block3_1_bn (BatchNormal (None, 14, 14, 256) 1024      ['conv4_block3_1_conv[0][0]']
ization)

conv4_block3_1_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block3_1_bn[0][0]']
n)

conv4_block3_2_conv (Conv2D) (None, 14, 14, 256) 590080      ['conv4_block3_1_relu[0][0]']

conv4_block3_2_bn (BatchNormal (None, 14, 14, 256) 1024      ['conv4_block3_2_conv[0][0]']
ization)

conv4_block3_2_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block3_2_bn[0][0]']
n)

conv4_block3_3_conv (Conv2D) (None, 14, 14, 1024 263168      ['conv4_block3_2_relu[0][0]']
)

conv4_block3_3_bn (BatchNormal (None, 14, 14, 1024 4096      ['conv4_block3_3_conv[0][0]']
ization)

conv4_block3_add (Add) (None, 14, 14, 1024 0      ['conv4_block2_out[0][0]',
)                                     'conv4_block3_3_bn[0][0]']

conv4_block3_out (Activation) (None, 14, 14, 1024 0      ['conv4_block3_add[0][0]']
)

conv4_block4_1_conv (Conv2D) (None, 14, 14, 256) 262400      ['conv4_block3_out[0][0]']

conv4_block4_1_bn (BatchNormal (None, 14, 14, 256) 1024      ['conv4_block4_1_conv[0][0]']
ization)

conv4_block4_1_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block4_1_bn[0][0]']
n)

conv4_block4_2_conv (Conv2D) (None, 14, 14, 256) 590080      ['conv4_block4_1_relu[0][0]']

conv4_block4_2_bn (BatchNormal (None, 14, 14, 256) 1024      ['conv4_block4_2_conv[0][0]']
ization)

```

```

conv4_block4_2_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block4_2_bn[0][0]']
n)

conv4_block4_3_conv (Conv2D) (None, 14, 14, 1024) 263168    ['conv4_block4_2_relu[0][0]']
)

conv4_block4_3_bn (BatchNormal (None, 14, 14, 1024) 4096    ['conv4_block4_3_conv[0][0]']
ization)
)

conv4_block4_add (Add) (None, 14, 14, 1024) 0      ['conv4_block3_out[0][0]',
)      'conv4_block4_3_bn[0][0]']

conv4_block4_out (Activation) (None, 14, 14, 1024) 0      ['conv4_block4_add[0][0]']
)

conv4_block5_1_conv (Conv2D) (None, 14, 14, 256) 262400    ['conv4_block4_out[0][0]']

conv4_block5_1_bn (BatchNormal (None, 14, 14, 256) 1024    ['conv4_block5_1_conv[0][0]']
ization)

conv4_block5_1_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block5_1_bn[0][0]']
n)

conv4_block5_2_conv (Conv2D) (None, 14, 14, 256) 590080    ['conv4_block5_1_relu[0][0]']

conv4_block5_2_bn (BatchNormal (None, 14, 14, 256) 1024    ['conv4_block5_2_conv[0][0]']
ization)

conv4_block5_2_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block5_2_bn[0][0]']
n)

conv4_block5_3_conv (Conv2D) (None, 14, 14, 1024) 263168    ['conv4_block5_2_relu[0][0]']
)

conv4_block5_3_bn (BatchNormal (None, 14, 14, 1024) 4096    ['conv4_block5_3_conv[0][0]']
ization)
)

conv4_block5_add (Add) (None, 14, 14, 1024) 0      ['conv4_block4_out[0][0]',
)      'conv4_block5_3_bn[0][0]']

conv4_block5_out (Activation) (None, 14, 14, 1024) 0      ['conv4_block5_add[0][0]']
)

conv4_block6_1_conv (Conv2D) (None, 14, 14, 256) 262400    ['conv4_block5_out[0][0]']

conv4_block6_1_bn (BatchNormal (None, 14, 14, 256) 1024    ['conv4_block6_1_conv[0][0]']
ization)

conv4_block6_1_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block6_1_bn[0][0]']
n)

conv4_block6_2_conv (Conv2D) (None, 14, 14, 256) 590080    ['conv4_block6_1_relu[0][0]']

conv4_block6_2_bn (BatchNormal (None, 14, 14, 256) 1024    ['conv4_block6_2_conv[0][0]']
ization)

conv4_block6_2_relu (Activation) (None, 14, 14, 256) 0      ['conv4_block6_2_bn[0][0]']
n)

conv4_block6_3_conv (Conv2D) (None, 14, 14, 1024) 263168    ['conv4_block6_2_relu[0][0]']
)

```



```

conv4_block6_3_bn (BatchNormal (None, 14, 14, 1024 4096    ['conv4_block6_3_conv[0][0]']
ization)
)

conv4_block6_add (Add)      (None, 14, 14, 1024 0    ['conv4_block5_out[0][0]',
)                               'conv4_block6_3_bn[0][0]']

conv4_block6_out (Activation) (None, 14, 14, 1024 0    ['conv4_block6_add[0][0]']
)

conv5_block1_1_conv (Conv2D) (None, 7, 7, 512) 524800    ['conv4_block6_out[0][0]']

conv5_block1_1_bn (BatchNormal (None, 7, 7, 512) 2048    ['conv5_block1_1_conv[0][0]']
ization)

conv5_block1_1_relu (Activatio (None, 7, 7, 512) 0    ['conv5_block1_1_bn[0][0]']
n)

conv5_block1_2_conv (Conv2D) (None, 7, 7, 512) 2359808    ['conv5_block1_1_relu[0][0]']

conv5_block1_2_bn (BatchNormal (None, 7, 7, 512) 2048    ['conv5_block1_2_conv[0][0]']
ization)

conv5_block1_2_relu (Activatio (None, 7, 7, 512) 0    ['conv5_block1_2_bn[0][0]']
n)

conv5_block1_0_conv (Conv2D) (None, 7, 7, 2048) 2099200    ['conv4_block6_out[0][0]']

conv5_block1_3_conv (Conv2D) (None, 7, 7, 2048) 1050624    ['conv5_block1_2_relu[0][0]']

conv5_block1_0_bn (BatchNormal (None, 7, 7, 2048) 8192    ['conv5_block1_0_conv[0][0]']
ization)

conv5_block1_3_bn (BatchNormal (None, 7, 7, 2048) 8192    ['conv5_block1_3_conv[0][0]']
ization)

conv5_block1_add (Add)      (None, 7, 7, 2048) 0    ['conv5_block1_0_bn[0][0]',
)                               'conv5_block1_3_bn[0][0]']

conv5_block1_out (Activation) (None, 7, 7, 2048) 0    ['conv5_block1_add[0][0]']

conv5_block2_1_conv (Conv2D) (None, 7, 7, 512) 1049088    ['conv5_block1_out[0][0]']

conv5_block2_1_bn (BatchNormal (None, 7, 7, 512) 2048    ['conv5_block2_1_conv[0][0]']
ization)

conv5_block2_1_relu (Activatio (None, 7, 7, 512) 0    ['conv5_block2_1_bn[0][0]']
n)

conv5_block2_2_conv (Conv2D) (None, 7, 7, 512) 2359808    ['conv5_block2_1_relu[0][0]']

conv5_block2_2_bn (BatchNormal (None, 7, 7, 512) 2048    ['conv5_block2_2_conv[0][0]']
ization)

conv5_block2_2_relu (Activatio (None, 7, 7, 512) 0    ['conv5_block2_2_bn[0][0]']
n)

conv5_block2_3_conv (Conv2D) (None, 7, 7, 2048) 1050624    ['conv5_block2_2_relu[0][0]']

conv5_block2_3_bn (BatchNormal (None, 7, 7, 2048) 8192    ['conv5_block2_3_conv[0][0]']
ization)

conv5_block2_add (Add)      (None, 7, 7, 2048) 0    ['conv5_block1_out[0][0]',
)                               'conv5_block2_3_bn[0][0]']

```

```

conv5_block2_out (Activation) (None, 7, 7, 2048) 0      ['conv5_block2_add[0][0]']

conv5_block3_1_conv (Conv2D) (None, 7, 7, 512) 1049088  ['conv5_block2_out[0][0]']

conv5_block3_1_bn (BatchNormal (None, 7, 7, 512) 2048  ['conv5_block3_1_conv[0][0]']
ization)

conv5_block3_1_relu (Activatio (None, 7, 7, 512) 0      ['conv5_block3_1_bn[0][0]']
n)

conv5_block3_2_conv (Conv2D) (None, 7, 7, 512) 2359808  ['conv5_block3_1_relu[0][0]']

conv5_block3_2_bn (BatchNormal (None, 7, 7, 512) 2048  ['conv5_block3_2_conv[0][0]']
ization)

conv5_block3_2_relu (Activatio (None, 7, 7, 512) 0      ['conv5_block3_2_bn[0][0]']
n)

conv5_block3_3_conv (Conv2D) (None, 7, 7, 2048) 1050624  ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal (None, 7, 7, 2048) 8192  ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add) (None, 7, 7, 2048) 0      ['conv5_block2_out[0][0]',
                                     'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation) (None, 7, 7, 2048) 0      ['conv5_block3_add[0][0]']

flatten (Flatten) (None, 100352) 0      ['conv5_block3_out[0][0]']

predictions (Dense) (None, 2) 200706  ['flatten[0][0]']

```

```

=====
==
Total params: 23,788,418
Trainable params: 200,706
Non-trainable params: 23,587,712

```

Table 5:- ResNet-50 Model Summary

ResNet-50 architecture contains the following element:

- ✓ A convoultion with a kernel size of $7 * 7$ and 64 different kernels all with a stride of size 2 giving us **1 layer**.
- ✓ max pooling with also a stride size of 2.
- ✓ In the next convolution there is a $1 * 1,64$ kernel following this a $3 * 3,64$ kernel and at last a $1 * 1,256$ kernel, These three layers are repeated in total 3 time so giving us **9 layers** in this step.
- ✓ kernel of $1 * 1,128$ after that a kernel of $3 * 3,128$ and at last a kernel of 1

- * 1,512 this step was repeated 4 time so giving us **12 layers** in this step.
- ✓ After that there is a kernal of $1 * 1,256$ and two more kernels with $3 * 3,256$ and $1 * 1,1024$ and this is repeated 6 time giving us a total of **18 layers**.
- ✓ And then again a $1 * 1,512$ kernel with two more of $3 * 3,512$ and $1 * 1,2048$ and this was repeated 3 times giving us a total of **9 layers**.
- ✓ After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function so this gives us **1 layer**.

In the proposed models there are initially 23,788,418 parameters but only 200,706 parameters need to be train which save our resources. ResNet-50 is derived from CNN models. After dividing data set and pre-processing of image dataset. I started to train my models with the epochs of 30.

Epoch 1/30

2022-05-07 12:34:08.573922: I tensorflow/stream_executor/cuda/cuda_dnn.cc:368]

Loaded cuDNN version 8202

1/10 [==>.....] - ETA: 47s - loss: 1.4231 - accuracy: 0.4375

2022-05-07 12:34:10.838053: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.30GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:10.838083: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.30GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:10.895444: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.41GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:10.895469: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.41GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

10/10 [=====] - ETA: 0s - loss: 6.0184 - accuracy: 0.8313

Epoch 1: val_accuracy improved from -inf to 0.97921, saving model to bestmodelresnet50.h5

2022-05-07 12:34:25.214707: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.29GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:25.214735: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.29GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:25.253885: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.34GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:25.253909: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.34GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

10/10 [=====] - 20s 2s/step - loss: 6.0184 - accuracy: 0.8313 - val_loss: 0.5003 - val_accuracy: 0.9792

Epoch 2/30

2/10 [=====>.....] - ETA: 13s - loss: 1.2894 - accuracy: 0.9516

2022-05-07 12:34:28.410933: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.30GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

2022-05-07 12:34:28.410960: W

tensorflow/core/common_runtime/bfc_allocator.cc:275] Allocator (GPU_0_bfc) ran

out of memory trying to allocate 2.30GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available.

10/10 [=====] - ETA: 0s - loss: 1.2308 - accuracy: 0.9717

Epoch 2: val_accuracy did not improve from 0.97921

10/10 [=====] - 14s 1s/step - loss: 1.2308 - accuracy: 0.9717 - val_loss: 1.9962 - val_accuracy: 0.9376

Epoch 3/30

10/10 [=====] - ETA: 0s - loss: 0.6879 - accuracy: 0.9844

Epoch 3: val_accuracy did not improve from 0.97921

10/10 [=====] - 14s 1s/step - loss: 0.6879 - accuracy: 0.9844 - val_loss: 0.5825 - val_accuracy: 0.9792

Epoch 4/30

10/10 [=====] - ETA: 0s - loss: 0.1119 - accuracy: 0.9969

Epoch 4: val_accuracy improved from 0.97921 to 0.99307, saving model to bestmodelresnet50.h5

10/10 [=====] - 15s 2s/step - loss: 0.1119 - accuracy: 0.9969 - val_loss: 0.1176 - val_accuracy: 0.9931

Epoch 5/30

10/10 [=====] - ETA: 0s - loss: 0.1007 - accuracy: 0.9875

Epoch 5: val_accuracy did not improve from 0.99307

10/10 [=====] - 15s 2s/step - loss: 0.1007 - accuracy: 0.9875 - val_loss: 0.2832 - val_accuracy: 0.9815

Epoch 6/30

10/10 [=====] - ETA: 0s - loss: 0.3263 - accuracy: 0.9937

Epoch 6: val_accuracy did not improve from 0.99307

10/10 [=====] - 14s 1s/step - loss: 0.3263 - accuracy: 0.9937 - val_loss: 0.6959 - val_accuracy: 0.9769

Epoch 7/30

10/10 [=====] - ETA: 0s - loss: 0.0370 - accuracy: 0.9969

Epoch 7: val_accuracy improved from 0.99307 to 0.99538, saving model to bestmodelresnet50.h5

10/10 [=====] - 17s 2s/step - loss: 0.0370 - accuracy: 0.9969 - val_loss: 0.1256 - val_accuracy: 0.9954

Epoch 7: early stopping

Table 6:- ResNet-50 Model Training Status

For ResNet-50 Pre-trained models I used the concept of Early Stopping which help me to save computer resource like CPU, GPU, RAM that's why my model is stopped at 7 epoch which help me to reduce over fitting .

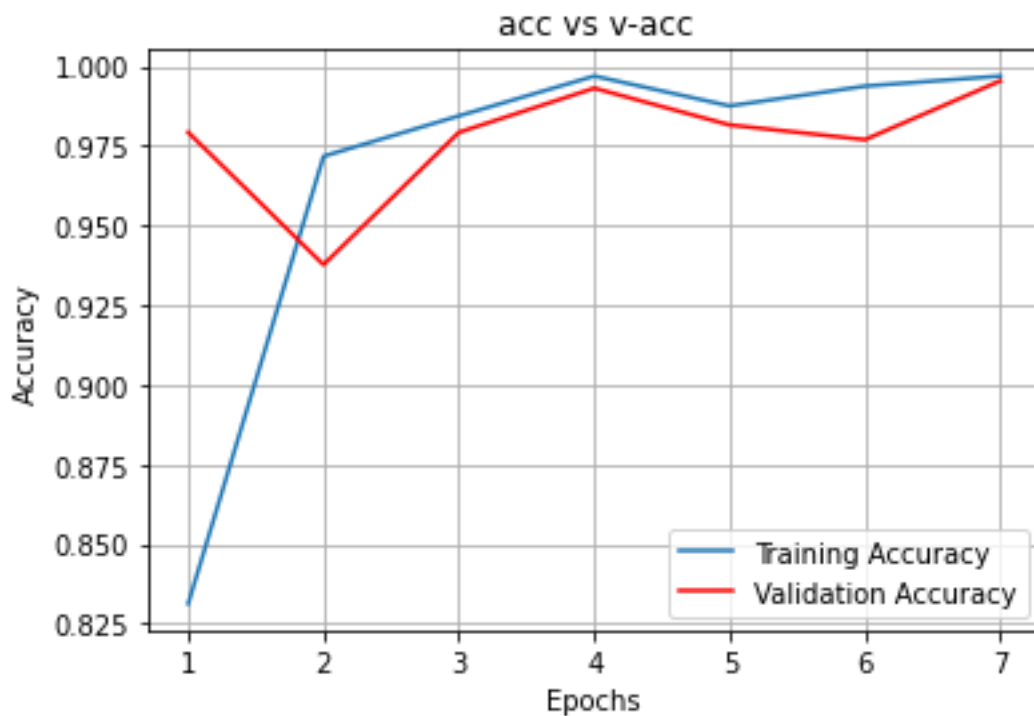


Fig 31:- ResNet-16 Model Training and Validation Accuracy Comparison

For this models I only need train my models up to just 7 epochs with that I am able to get validation accuracy 99.54 % and training accuracy is 99.69 %.

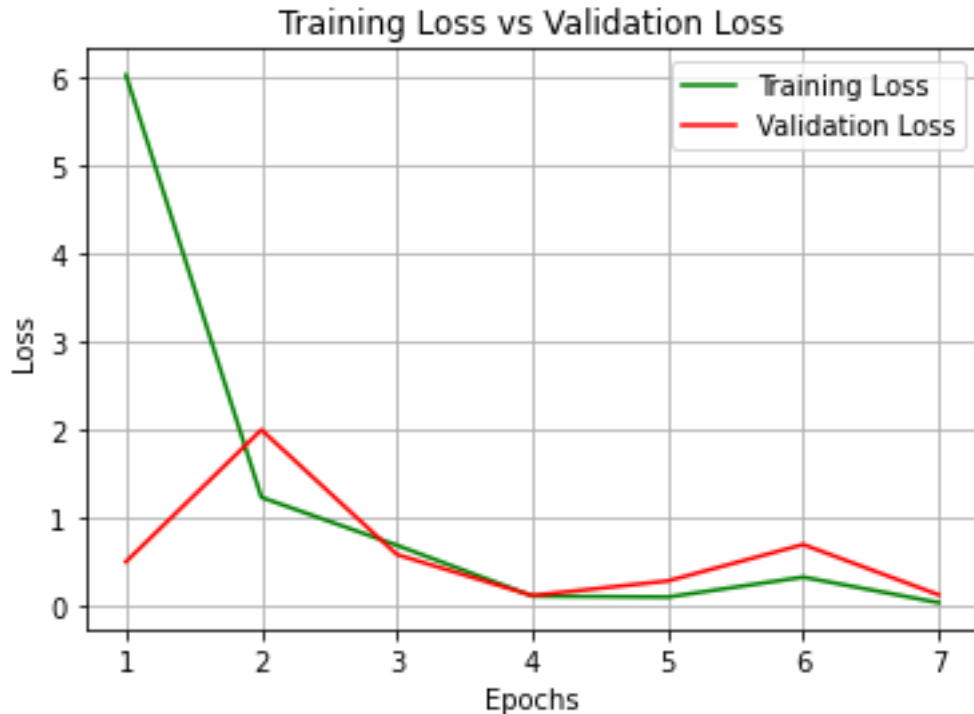


Fig 32:- ResNet-16 Model Training and Validation Loss Comparison

From figure 32, it shows that training loss and validation for this model is very less like 3-12 % which great. From figure 31 and 32 we can observe that model perform excellent with training set as well as validation set. Accuracy of ResNet-50 proposed model is 99.53%.

III. CNN

A convolutional neural network also known as CNN or Convnet is a class of neural network that specializes in processing data that has a grid-like topology such as images. In other words CNN are multilayer neural networks that are stacked on top of each other. A CNN typically has three layers: a convolutional layer, a pooling layer and a fully connected layer.

Convolutional layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the

other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

Neurons in Fully connected layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

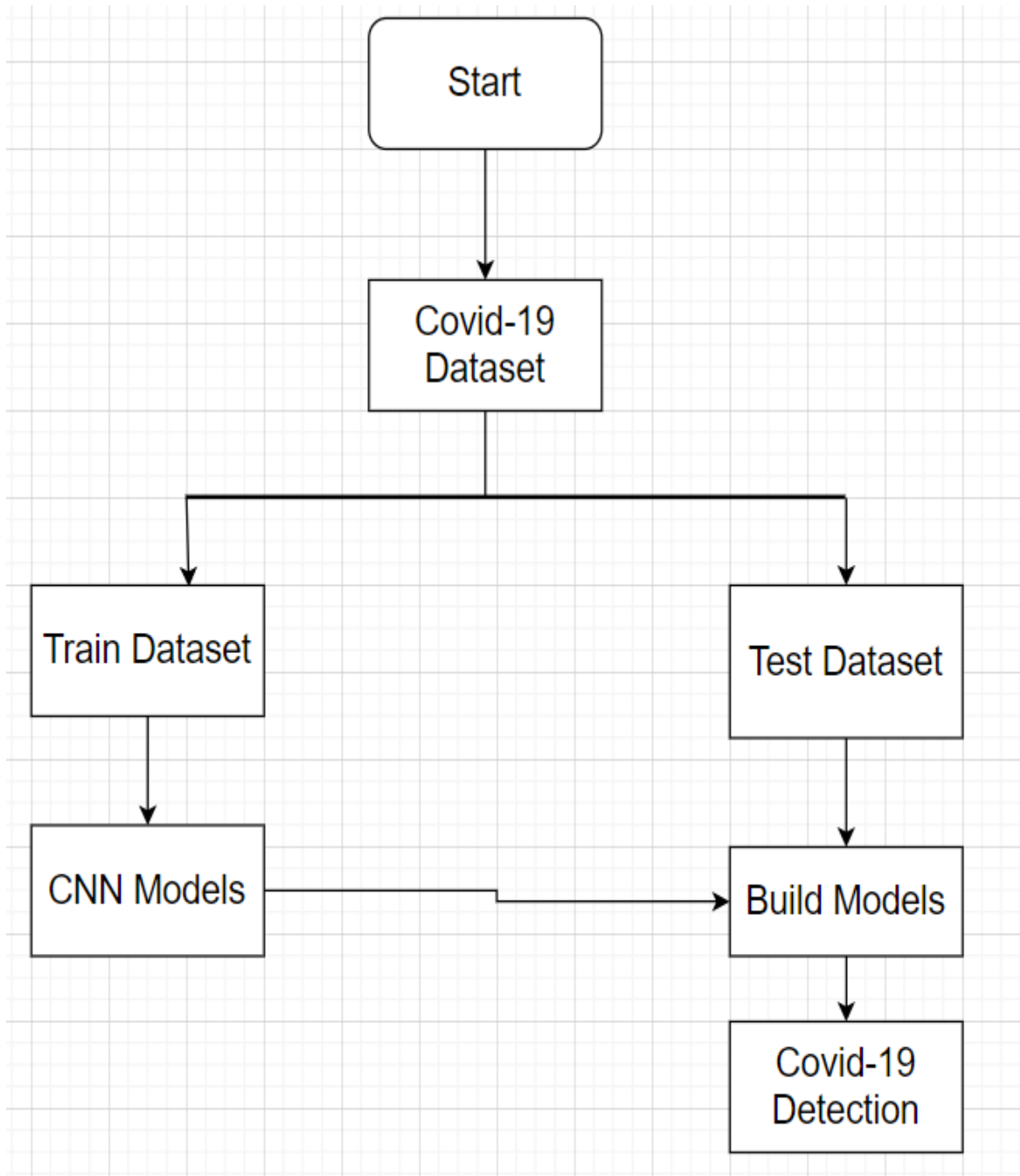


Fig 33:- Working Flow of CNN Model

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 128)	3584

max_pooling2d (MaxPooling2D (None, 111, 111, 128))		0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	73792
max_pooling2d_1 (MaxPooling 2D)	(None, 54, 54, 64)	0
dropout (Dropout)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 64)	36928
max_pooling2d_2 (MaxPooling 2D)	(None, 26, 26, 64)	0
dropout_1 (Dropout)	(None, 26, 26, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
max_pooling2d_3 (MaxPooling 2D)	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 2048)	37750784
dense_1 (Dense)	(None, 1024)	2098176
dense_2 (Dense)	(None, 256)	262400
dense_3 (Dense)	(None, 64)	16448
dense_4 (Dense)	(None, 16)	1040
dense_5 (Dense)	(None, 1)	17

=====

Total params: 40,317,025
Trainable params: 40,317,025
Non-trainable params: 0

Table 7:- CNN Model Summary

In the proposed models there are initially 40,317,025 parameters and all parameters are used for train which consume our lot of resources. After dividing data set and pre-processing of image dataset. I started to train my models with the epochs of 100.

Epoch 1/100

108/108 [=====] - ETA: 0s - loss: 0.4393 - accuracy: 0.8007.

108/108 [=====] - 53s 443ms/step - loss: 0.4393 - accuracy: 0.8007 - val_loss: 0.4831
- val_accuracy: 0.8499

Epoch 2/100

108/108 [=====] - 46s 426ms/step - loss: 0.3262 - accuracy: 0.8714 - val_loss: 0.3820

- val_accuracy: 0.9145

Epoch 3/100

108/108 [=====] - 49s 450ms/step - loss: 0.2827 - accuracy: 0.8917 - val_loss: 0.2836

- val_accuracy: 0.9538

Epoch 4/100

108/108 [=====] - 47s 437ms/step - loss: 0.2623 - accuracy: 0.9050 - val_loss: 0.3246

- val_accuracy: 0.9538

Epoch 5/100

108/108 [=====] - 51s 469ms/step - loss: 0.2694 - accuracy: 0.9027 - val_loss: 0.2649

- val_accuracy: 0.9376

Epoch 6/100

108/108 [=====] - 50s 460ms/step - loss: 0.2537 - accuracy: 0.9073 - val_loss: 0.2776

- val_accuracy: 0.9630

Epoch 7/100

108/108 [=====] - 52s 485ms/step - loss: 0.2578 - accuracy: 0.9119 - val_loss: 0.1967

- val_accuracy: 0.9677

Epoch 8/100

108/108 [=====] - 51s 471ms/step - loss: 0.2420 - accuracy: 0.9143 - val_loss: 0.2080

- val_accuracy: 0.9607

Epoch 9/100

108/108 [=====] - 52s 485ms/step - loss: 0.2478 - accuracy: 0.9079 - val_loss: 0.1892

- val_accuracy: 0.9492

Epoch 10/100

108/108 [=====] - 52s 483ms/step - loss: 0.2127 - accuracy: 0.9253 - val_loss: 0.1837

- val_accuracy: 0.9423

Epoch 11/100

108/108 [=====] - 53s 491ms/step - loss: 0.2204 - accuracy: 0.9166 - val_loss: 0.1383

- val_accuracy: 0.9469

Epoch 12/100

108/108 [=====] - 53s 491ms/step - loss: 0.2010 - accuracy: 0.9247 - val_loss: 0.1223

- val_accuracy: 0.9654

Epoch 13/100

108/108 [=====] - 52s 482ms/step - loss: 0.1805 - accuracy: 0.9316 - val_loss: 0.1304

- val_accuracy: 0.9607

Epoch 14/100

108/108 [=====] - 53s 488ms/step - loss: 0.1660 - accuracy: 0.9363 - val_loss: 0.1454

- val_accuracy: 0.9515

Epoch 15/100

108/108 [=====] - 56s 513ms/step - loss: 0.1476 - accuracy: 0.9490 - val_loss: 0.1004

- val_accuracy: 0.9746

Epoch 16/100

108/108 [=====] - 54s 503ms/step - loss: 0.1226 - accuracy: 0.9490 - val_loss: 0.0714

- val_accuracy: 0.9792

Epoch 17/100
 108/108 [=====] - 53s 492ms/step - loss: 0.1388 - accuracy: 0.9473 - val_loss: 0.1932
 - val_accuracy: 0.9376
 Epoch 18/100
 108/108 [=====] - 53s 492ms/step - loss: 0.1217 - accuracy: 0.9554 - val_loss: 0.1263
 - val_accuracy: 0.9584
 Epoch 19/100
 108/108 [=====] - 52s 480ms/step - loss: 0.1261 - accuracy: 0.9537 - val_loss: 0.1090
 - val_accuracy: 0.9723

Table 8:- CNN Model Training Status

For CNN model I used the concept of Early Stopping which help me to save computer resource like CPU, GPU, RAM that's why my model is stopped at 19 epoch which help me to reduce over fitting .

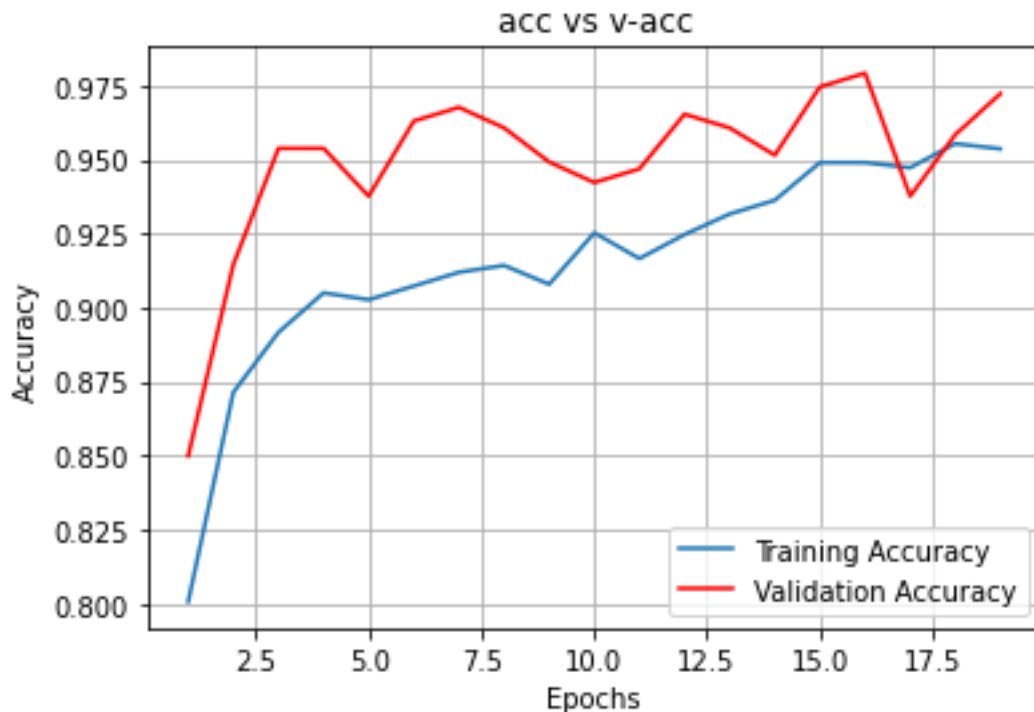


Fig 33:- CNN Model Training and Validation Accuracy Comparison

For this models I only need train my models up to just 19 epochs with that I am able to get validation accuracy 97.27 % and training accuracy is 95.37 %.

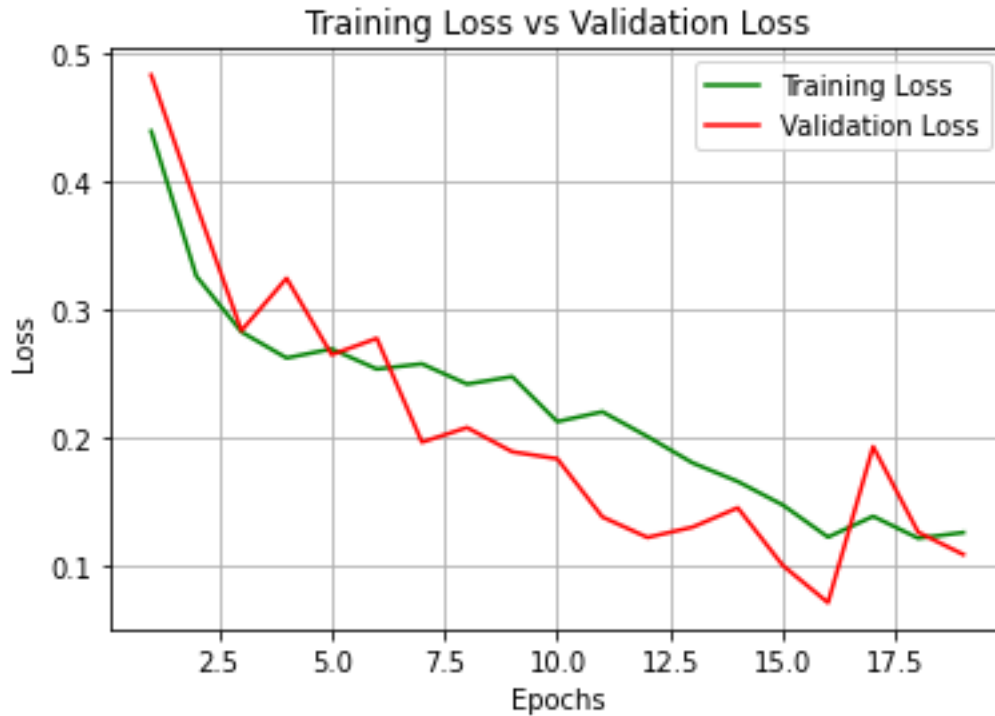


Fig 34:- CNN Model Training and Validation Loss Comparison

From figure 34, it shows that training loss and validation for this model is very less like 10-12 % which is good. From figure 33 and 34 we can observe that the model performs excellently with both the training set as well as the validation set. The accuracy of the CNN proposed model is 95.36%.

IV. CNN(Local Maxima)

In this model, I am demonstrating how the model can get stuck on a global minimum, for this model also I am using CNN architecture. The local maxima is the input value for which the function gives the maximum output values. The function equation or the graph of the function is not sufficient to find the local maximum. The derivative of the function is very helpful in finding the local maximum of the function. The below graph shows the local maximum within the defined interval of the domain. Further, the function has another maximum range value across the entire domain, which is called the global maximum.

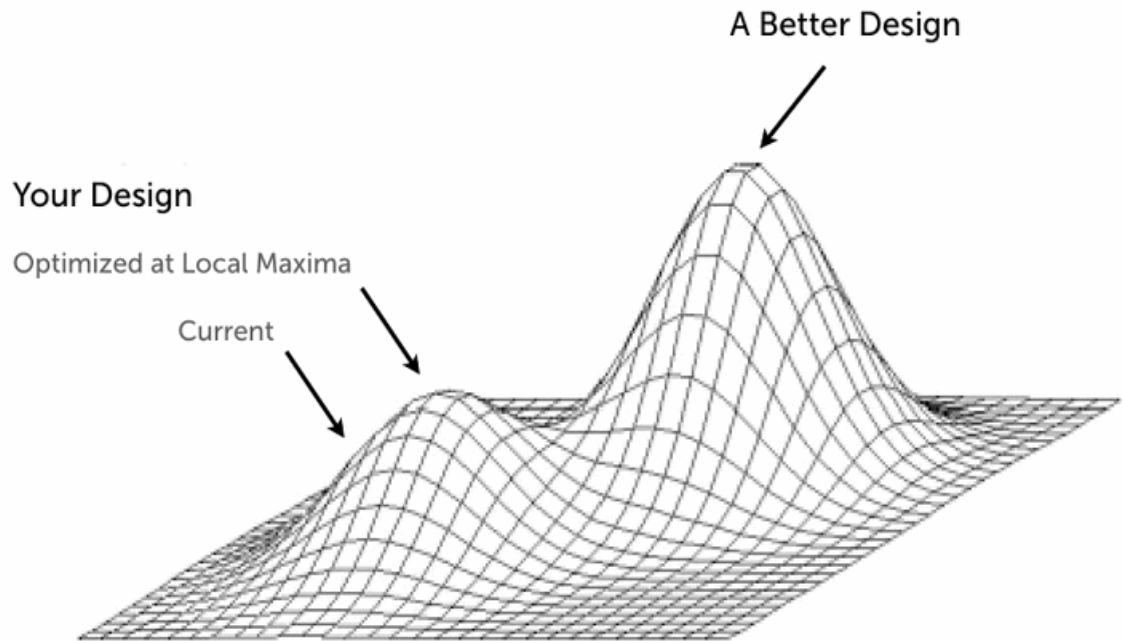


Fig 35:- Condition of Local Maxima

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 128)	3584
max_pooling2d (MaxPooling2D)	(None, 31, 31, 128)	0
conv2d_1 (Conv2D)	(None, 29, 29, 64)	73792
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0

dropout_1 (Dropout)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 256)	1179904
dense_1 (Dense)	(None, 64)	16448
dense_2 (Dense)	(None, 16)	1040
dense_3 (Dense)	(None, 1)	17

```
=====
Total params: 1,348,641
Trainable params: 1,348,641
Non-trainable params: 0
```

Table 9:- CNN (Local Maxima) Model Summary

In the proposed models there are initially 1,348,641 parameters and all parameters are used for train which consume our lot of resources. After dividing data set and pre-processing of image dataset. I started to train my models with the epochs of 10.

```
Epoch 1/10
2022-03-19 13:28:33.870642: I tensorflow/stream_executor/cuda/cuda_dnn.cc:368] Loaded cuDNN
version 8202
8/8 [=====] - 15s 1s/step - loss: 0.6554 - accuracy: 0.7598 -
val_loss: 0.6941 - val_accuracy: 0.5000
Epoch 2/10
8/8 [=====] - 10s 1s/step - loss: 0.6313 - accuracy: 0.7283 -
val_loss: 0.6991 - val_accuracy: 0.5000
Epoch 3/10
8/8 [=====] - 10s 1s/step - loss: 0.5901 - accuracy: 0.7539 -
val_loss: 0.7103 - val_accuracy: 0.5000
Epoch 4/10
8/8 [=====] - 10s 1s/step - loss: 0.5707 - accuracy: 0.7422 -
val_loss: 0.7262 - val_accuracy: 0.5000
Epoch 5/10
```

8/8 [=====] - 10s 1s/step - loss: 0.5277 - accuracy: 0.7734 -
val_loss: 0.7492 - val_accuracy: 0.5000
Epoch 6/10
8/8 [=====] - 10s 1s/step - loss: 0.5377 - accuracy: 0.7578 -
val_loss: 0.7608 - val_accuracy: 0.5000
Epoch 7/10
8/8 [=====] - 10s 1s/step - loss: 0.5563 - accuracy: 0.7422 -
val_loss: 0.7567 - val_accuracy: 0.5000
Epoch 8/10
8/8 [=====] - 11s 1s/step - loss: 0.6249 - accuracy: 0.6797 -
val_loss: 0.7328 - val_accuracy: 0.5000
Epoch 9/10
8/8 [=====] - 10s 1s/step - loss: 0.5765 - accuracy: 0.7070 -
val_loss: 0.7226 - val_accuracy: 0.5000
Epoch 10/10
8/8 [=====] - 10s 1s/step - loss: 0.5425 - accuracy: 0.7500 -
val_loss: 0.7244 - val_accuracy: 0.5000

Table 10:- CNN (Local Maxima) Model Training Status

If we see the table 10 we can notice that we got constant validation accuracy i.e. 75 % at every epochs that is the condition for global minima.

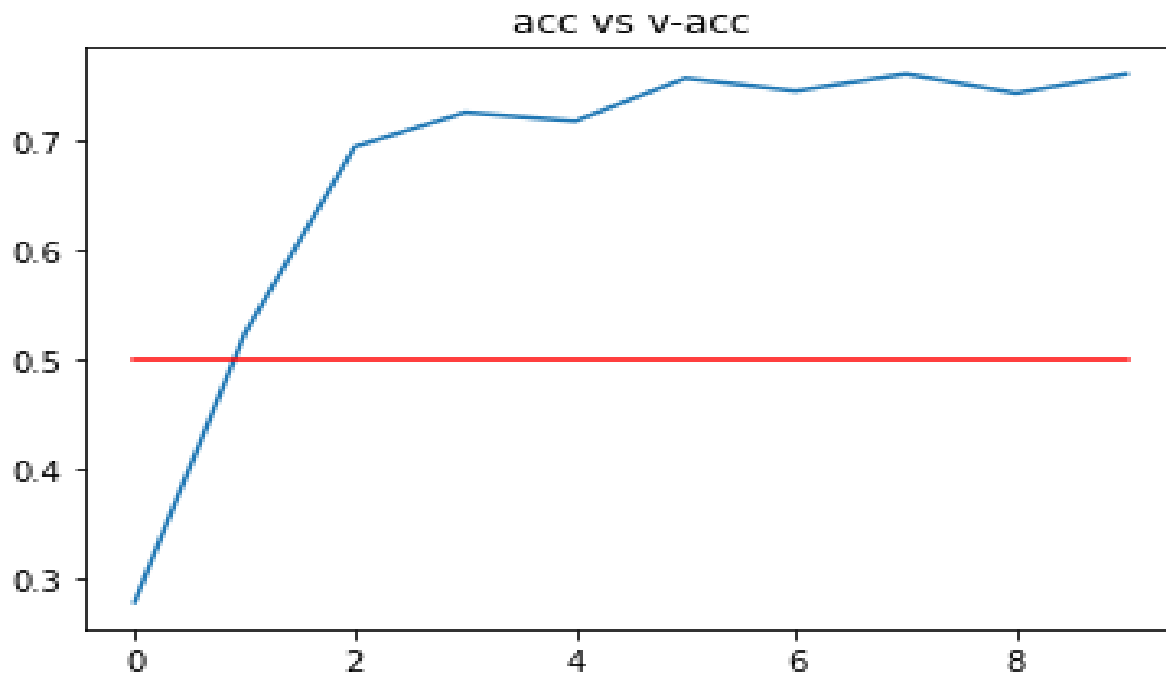


Fig 36:- CNN (Local Maxima) Model Training and Validation Accuracy Comparison

Form the fig 36 we can visually observe the condition for global minima. It is happen due to learning rate.

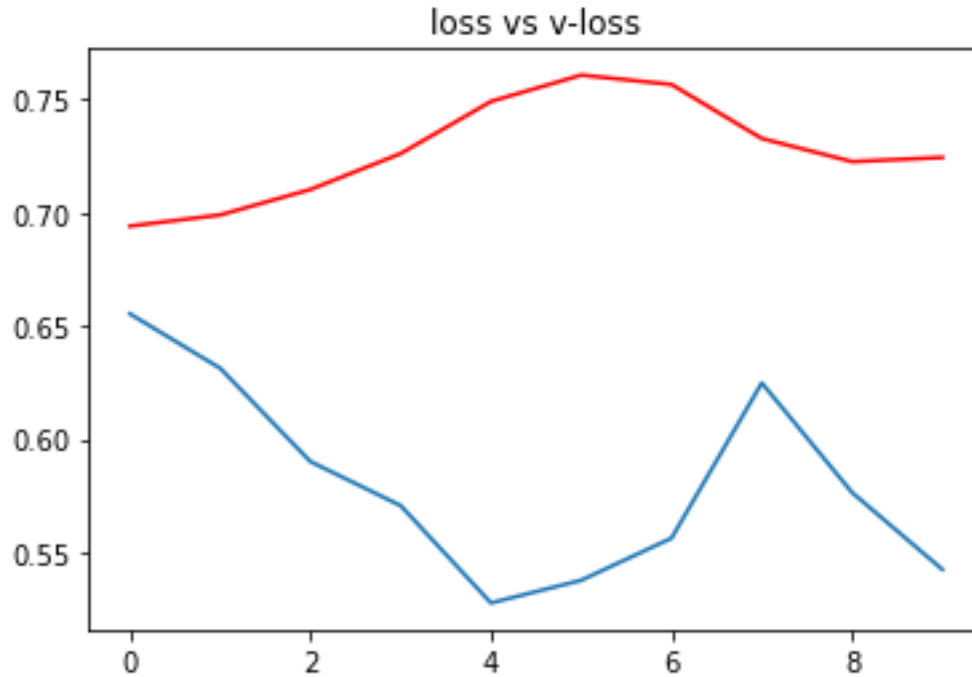


Fig 37:- CNN (Local Maxima) Model Training and Validation Loss Comparison

In a condition of local maxima loss are too much and can't control by developer. From figure 37 we can notice that loss is too much high i.e. more than 60 % and curve in a figure 37 is zig - zag that means models is useless.

VII. CODE SCREENSHORT

For VGG-16

```
VGG-16.ipynb
File Edit View Insert Runtime Tools Help Last saved at 13:29

+ Code + Text
Connect Editing

[ ] import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

[ ] train_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/"
test_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/test/"

import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense,MaxPool2D,Conv2D,Dropout,Flatten
from keras.models import Sequential
from tensorflow.keras.applications import vgg16

[ ] train_data_gen = ImageDataGenerator(preprocessing_function= vgg16.preprocess_input ,
                                      zoom_range= 0.2, horizontal_flip= True,
                                      shear_range= 0.2 ,
                                      rescale= 1./255)

train = train_data_gen.flow_from_directory(directory= train_path , target_size=(224,224))

Found 1726 images belonging to 2 classes.

[ ] test_data_gen = ImageDataGenerator(preprocessing_function= vgg16.preprocess_input,
                                      rescale= 1./255 )
test = train_data_gen.flow_from_directory(directory= test_path ,
                                      target_size=(224,224), shuffle= False)
```

```
(6) Eminem - Stan (Long Ver... VGG-16.ipynb - Colaboratory
colab.research.google.com/drive/1r2oxMhG9lpTEH5x-el7p-vj8ShuQIno#scrollTo=1ef7cd22

VGG-16.ipynb
File Edit View Insert Runtime Tools Help Last saved at 13:29

+ Code + Text
Connect Editing

Found 1726 images belonging to 2 classes.

[ ] test_data_gen = ImageDataGenerator(preprocessing_function= vgg16.preprocess_input,
                                      rescale= 1./255 )
test = train_data_gen.flow_from_directory(directory= test_path ,
                                      target_size=(224,224), shuffle= False)

Found 433 images belonging to 2 classes.

from tensorflow.keras.applications.vgg16 import VGG16
from keras.layers import Flatten, Dense, Dropout, MaxPool2D
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
from keras.models import Model

[ ] vgg = VGG16( input_shape=(224,224,3), include_top= False)
# include_top will consider the new weights

2022-05-07 23:54:34.941144: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:34.967710: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:34.968515: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:34.970174: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the followi
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-05-07 23:54:34.971185: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:34.971796: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:34.972399: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:35.386525: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:35.386796: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:35.387016: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:936] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUM
2022-05-07 23:54:35.387212: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2333 MB memory: -> device: 0, nam

Activate Windows
Go to Settings to activate Windows.
```

```
[ ] for layer in vgg.layers:           # Dont Train the parameters again
    layer.trainable = False
```

```
[ ] x = Flatten()(vgg.output)
    x = Dense(units=2, activation='sigmoid', name = 'predictions')(x)

    model = Model(vgg.input, x)
```

```
[ ] model.compile(optimizer='adam', loss = 'categorical_crossentropy', metrics=['accuracy'])
```

```
[ ] # implementing early stopping and model check point
```

```
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

es = EarlyStopping(monitor= "val_accuracy" , min_delta= 0.01, patience= 3, verbose=1)
mc = ModelCheckpoint(filepath="bestmodel.h5", monitor="val_accuracy",
                     verbose=1, save_best_only= True)
```

```
[ ] hist = model.fit_generator(train, steps_per_epoch= 10, epochs= 30,
                             validation_data= test ,
                             callbacks=[es,mc])
```

```
plt.plot(epochs,h['loss'],c='green',label="Training Loss")
plt.plot(epochs,h['val_loss'],c='red',label="Validation Loss")
plt.title("Training Loss vs Validation Loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()
plt.show()
```

```
[ ] ## load only the best model
    from keras.models import load_model
    model = load_model("bestmodel.h5")
```

```
[ ] h = hist.history
    h.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[ ] epochs=range(1,len(h['accuracy'])+1)
    epochs
```

```
range(1, 8)
```

```
[ ] epochs=range(1,len(h['accuracy'])+1)
    plt.plot(epochs,h['accuracy'],label="Training Accuracy")
    plt.plot(epochs,h['val_accuracy'], c = "red",label="Validation Accuracy")
    plt.title("acc vs v-acc")
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
[ ] # checking out the accuracy of our model

acc = model.evaluate_generator(generator= test)[1]

print(f"The accuracy of your model is = {acc} %")
```

The accuracy of your model is = 0.9953810572624207 %

```
[ ] from keras.preprocessing import image

def get_img_array(img_path):
    """
    Input : Takes in image path as input
    Output : Gives out Pre-Processed image
    """
    path = img_path
    img = image.load_img(path, target_size=(224,224,3))
    img = image.img_to_array(img)/255
    img = np.expand_dims(img , axis= 0 )

    return img
```

```
path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/NORMAL/NORMAL(100).jpg"

#predictions: path:- provide any image from google or provide image from all image folder
img = get_img_array(path)

print(f"The chances of image being Covid is : {model.predict(img)[0][0]*100} percent")
print()
print(f"The chances of image being Normal is : {model.predict(img)[0][1]*100} percent")

# to display the image
plt.imshow(img[0], cmap = "gray")
plt.title("input image")
plt.show()
```

```
[ ] import tensorflow as tf
```

```
[ ] # this function is used to generate the heat map of an image
```

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
```

```

# We multiply each channel in the feature map array
# by "how important this channel is" with regard to the top predicted class
# then sum all the channels to obtain the heatmap class activation
last_conv_layer_output = last_conv_layer_output[0]
heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
heatmap = tf.squeeze(heatmap)

# For visualization purpose, we will also normalize the heatmap between 0 & 1
heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
return heatmap.numpy()

```

```

[ ] import matplotlib.cm as cm
    from IPython.display import Image, display

```

```

[ ] # put the heatmap to our image to understand the area of interest

def save_and_display_gradcam(img_path , heatmap, cam_path="cam.jpg", alpha=0.4):
    """
    img input should not be expanded
    """

    # Load the original image
    img = keras.preprocessing.image.load_img(img_path)
    img = keras.preprocessing.image.img_to_array(img)

```

```

# Rescale heatmap to a range 0-255
heatmap = np.uint8(255 * heatmap)

# Use jet colormap to colorize heatmap
jet = cm.get_cmap("jet")

# Use RGB values of the colormap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

# Create an image with RGB colorized heatmap
jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

# Superimpose the heatmap on original image
superimposed_img = jet_heatmap * alpha + img
superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

# Save the superimposed image
superimposed_img.save(cam_path)

# Display Grad CAM
display(Image(cam_path))

```



```

# function that is used to predict the image type and the ares that are affected by covid

def image_prediction_and_visualization(path,last_conv_layer_name = "block5_conv3", model = model):
|

    img_array = get_img_array(path)

    heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)

    img = get_img_array(path)

    print(f"The chances of image being Covid is : {model.predict(img)[0][0]*100} %")
    print(f"The chances of image being Normal is : {model.predict(img)[0][1]*100} %")

    print()
    print("image with heatmap representing the covid spot")

    # function call
    save_and_display_gradcam(path, heatmap)

    print()
    print("the original input image")
    print()

    a = plt.imread(path)
    plt.imshow(a, cmap = "gray")
    plt.title("Original image")
    plt.show()

```

```

#predictions
# provide the path of any image from google or any other scource
# the path is already defigned above , but you can also provide the path here to avoid scrolling up

# for covid image :
path="/home/bibek/Downloads/Small_Size_Data/dataset/covid/1-s2.0-S0929664620300449-gr2_lrg-c.jpg"

image_prediction_and_visualization(path)

```

For ResNet-50

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
[ ] train_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/"
    val_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/test/"
```

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
    from keras.models import Model
    from keras.layers import Dense, MaxPool2D, Conv2D, Flatten
    import keras
```

```
[ ] train_data_gen = ImageDataGenerator(preprocessing_function= preprocess_input,
                                       zoom_range= 0.2,
                                       horizontal_flip= True,
                                       shear_range= 0.2,
                                       )

    train = train_data_gen.flow_from_directory(directory= train_path,
                                              target_size=(224,224))
```

Found 1726 images belonging to 2 classes.

```
[ ] validation_data_gen = ImageDataGenerator(preprocessing_function= preprocess_input )
```

```
[ ] validation_data_gen = ImageDataGenerator(preprocessing_function= preprocess_input )  
  
valid = validation_data_gen.flow_from_directory(directory= val_path,  
                                                target_size=(224,224))
```

Found 433 images belonging to 2 classes.

```
[ ] class_type = {0:'Covid', 1 : 'Normal'}
```

```
[ ] t_img , label = train.next()
```

```
[ ] def plotImages(img_arr, label):  
    """  
    input :- images array  
    output :- plots the images  
    """  
  
    for im, l in zip(img_arr,label) :  
        plt.figure(figsize= (5,5))  
        plt.imshow(im, cmap = 'gray')  
        plt.title(im.shape)  
        plt.axis = False  
        plt.show()
```

```
[ ] plotImages(t_img, label)
```

```
[ ] res = ResNet50( input_shape=(224,224,3), include_top= False)  
    # include_top will consider the new weights
```

```
[ ] for layer in res.layers:                # Dont Train the parameters again  
    layer.trainable = False
```

```
[ ] x = Flatten()(res.output)  
    x = Dense(units=2 , activation='sigmoid', name = 'predictions' )(x)  
  
    # creating our model.  
    model = Model(res.input, x)
```

```
[ ] model.compile( optimizer= 'adam' , loss = 'categorical_crossentropy', metrics=['accuracy'])

[ ] # implementing early stopping and model check point

from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

es = EarlyStopping(monitor= "val_accuracy" , min_delta= 0.01, patience= 3, verbose=1)
mc = ModelCheckpoint(filepath="bestmodelresnet50.h5", monitor="val_accuracy",
                     verbose=1, save_best_only= True)

[ ] hist = model.fit_generator(train, steps_per_epoch= 10, epochs= 30,
                             validation_data= valid ,
                             callbacks=[es,mc])
```

```
## load only the best model
from keras.models import load_model
model = load_model("bestmodelresnet50.h5")
```

```
[ ] h = hist.history
    h.keys()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[ ] epochs=range(1,len(h['accuracy'])+1)
    epochs

range(1, 8)
```

```
[ ] epochs=range(1,len(h['accuracy'])+1)
    plt.plot(epochs,h['accuracy'],label="Training Accuracy")
    plt.plot(epochs,h['val_accuracy'], c = "red",label="Validation Accuracy")
    plt.title("acc vs v-acc")
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
[ ] # checking out the accurscy of our model

acc = model.evaluate_generator(generator= valid)[1]

print(f"The accuracy of your model is = {acc} %")

The accuracy of your model is = 0.9953810572624207 %
```

```
[ ] from keras.preprocessing import image

def get_img_array(img_path):
    """
    Input : Takes in image path as input
    Output : Gives out Pre-Processed image
    """
    path = img_path
    img = image.load_img(path, target_size=(224,224,3))
    img = image.img_to_array(img)
    img = np.expand_dims(img , axis= 0 )

    return img
```

```
path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/NORMAL/NORMAL(100).jpg"

#predictions: path:- provide any image from google or provide image from all image folder
img = get_img_array(path)

res = class_type[np.argmax(model.predict(img))]
print(f"The given X-Ray image is of type = {res}")
print()
print(f"The chances of image being Covid is : {model.predict(img)[0][0]*100} percent")
print()
print(f"The chances of image being Normal is : {model.predict(img)[0][1]*100} percent")

# to display the image
plt.imshow(img[0]/255, cmap = "gray")
plt.title("input image")
plt.show()
```

For CNN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
[ ] train_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/"
    val_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/test/"
```

```
[ ] import tensorflow as tf
    from tensorflow import keras
    from keras.preprocessing.image import ImageDataGenerator
    from keras.layers import Dense,MaxPool2D,Conv2D,Dropout,Flatten
    from keras.models import Sequential
```

```
[ ] config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth=True
    sess = tf.compat.v1.Session(config=config)
```

```
[ ] train_data_gen=ImageDataGenerator(rescale=1./255,
                                     horizontal_flip=True,
                                     zoom_range=0.2,
                                     shear_range=0.3,
                                     width_shift_range=0.3,
                                     height_shift_range=0.3,
                                     rotation_range=15)
    training_set=train_data_gen.flow_from_directory(directory=train_path,
                                                    target_size=(224,224),
                                                    class_mode='binary',
                                                    batch_size=16,
                                                    shuffle=True)
```

```
[ ] training_set.class_indices
```

```
{'COVID19': 0, 'NORMAL': 1}
```

```
[ ] test_data_gen=ImageDataGenerator(rescale=1./255)
    testing_set=test_data_gen.flow_from_directory(directory=val_path,
                                                    target_size=(224,224),
                                                    shuffle=False,
                                                    batch_size=16,
                                                    class_mode='binary')
```

Found 433 images belonging to 2 classes.

```
[ ] testing_set.class_indices
```

```
{'COVID19': 0, 'NORMAL': 1}
```

```
[ ] import glob
    import matplotlib.image as img
    path_covid=list(glob.glob(val_path+"COVID19/"+"/*"))
    path_normal=list(glob.glob(val_path+"NORMAL/"+"/*"))
```

```
[ ] plt.title("Covid 19 Image")
    covid_image_sample=img.imread(path_covid[0])
    plt.imshow(covid_image_sample)
```

```
model=Sequential()
model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu',input_shape=(224,224,3)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Dropout(0.15))
```

```
[ ] model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu'))
    model.add(MaxPool2D())
    model.add(Dropout(0.3))
```

```
[ ] model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu'))
    model.add(MaxPool2D())
    model.add(Dropout(0.15))
```

```
[ ] model.add(Flatten())
    # model.add(Dropout(0.1))
    model.add(Dense(2048,activation='relu'))
    model.add(Dense(1024,activation='relu'))
    model.add(Dense(256,activation='relu'))
    model.add(Dense(64,activation='relu'))
    model.add(Dense(16,activation='relu'))
    model.add(Dense(1,activation='sigmoid'))
```

```
▶ from tensorflow.keras.optimizers import SGD,RMSprop
  from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau

  checkpoint=ModelCheckpoint('final_model.h5',
                             monitor='val_loss',
                             mode='min',
                             save_best_only=True)

  earlystopping=EarlyStopping(monitor='val_loss',
                              min_delta=0.001,
                              patience=3,
                              restore_best_weights=True)

  |

  callback=[checkpoint,earlystopping]
```

```
[ ] from tensorflow.keras.optimizers import Adam
    # opt = SGD(lr=0.001)
    # opt = Adam(lr=0.003, beta_1=0.9, beta_2=0.999,
    #           epsilon=0.1, decay=0.0)
    opt=Adam(lr=0.0001)
    model.compile(loss = "binary_crossentropy", optimizer = opt,metrics=['accuracy'])
    # model.compile(loss = "binary_crossentropy",
    #               optimizer = RMSprop(learning_rate=0.0001),
    #               metrics=['accuracy'])
```

```
[ ] model.summary()
```



```
[ ] hist=model.fit(training_set,
                    epochs= 100,
                    validation_data= testing_set,
                    callbacks=callback)
```

```
[ ] h = hist.history
    h.keys()

    dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[ ] h = hist.history
    # h
```

```
[ ] epochs=range(1,len(h['accuracy'])+1)
    epochs

    range(1, 20)
```

```
[ ] epochs=range(1,len(h['accuracy'])+1)
    plt.plot(epochs,h['accuracy'],label="Training Accuracy")
    plt.plot(epochs,h['val_accuracy'], c = "red",label="Validation Accuracy")
    plt.title("acc vs v-acc")
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
[ ] plt.plot(epochs,h['loss'],c='green',label="Training Loss")
    plt.plot(epochs,h['val_loss'],c='red',label="Validation Loss")
    plt.title("Training Loss vs Validation Loss")
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
[ ] from sklearn.metrics import confusion_matrix, classification_report
    from keras.models import load_model
    from keras.preprocessing import image
    import numpy as np
    import os
    model=load_model('final_model.h5')

[ ] # Input : Takes in image path as input
    # Output : Gives out Pre-Processed image
    def get_img_array(img_path):
        path = img_path
        img = image.load_img(path, target_size=(224,224,3))
        img = image.img_to_array(img)/255
        img = np.expand_dims(img , axis= 0 )
        return img

[ ] def Final_Result(array_value):
    if array_value < 0.50:
        return "Covid"
    else:
        return "Normal"

[ ] class_type = {0:'Covid', 1 : 'Normal'}

[ ] path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/NORMAL/NORMAL(100).jpg"
```

```
[ ] res = class_type[np.argmax(model.predict(img))]
    print(f"The given X-Ray image is of type = {res}")
    print(f"The chances of image being Covid is : {model.predict(img)[0][0]*100} percent")
    # print(f"The chances of image being Normal is : {model.predict(img)[0][1]*100} percent")
```

For CNN (Global Minima)

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

[ ] train_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/"
    test_path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/test/"

[ ] import tensorflow as tf
    from tensorflow import keras
    from keras.preprocessing.image import ImageDataGenerator
    from keras.layers import Dense,MaxPool2D,Conv2D,Dropout,Flatten
    from keras.models import Sequential
    from keras.applications import vgg16

[ ] config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth=True
    sess = tf.compat.v1.Session(config=config)

```

```
[ ] train_data_gen=ImageDataGenerator(rescale=1./255,
                                     horizontal_flip=True,
                                     zoom_range=0.2,
                                     shear_range=0.3)
training_set=train_data_gen.flow_from_directory(directory=train_path,
                                                target_size=(64,64),
                                                class_mode='binary')
```

Found 1726 images belonging to 2 classes.

```
[ ] training_set.class_indices

{'COVID19': 0, 'NORMAL': 1}
```

```
[ ] test_data_gen=ImageDataGenerator(rescale=1./255)
testing_set=test_data_gen.flow_from_directory(directory=test_path,
                                              target_size=(64,64))
```

Found 433 images belonging to 2 classes.

```
[ ] testing_set.class_indices

{'COVID19': 0, 'NORMAL': 1}
```

```
[ ] class_type={0:'Covid',1:'Normal'}
```

```
[ ] import glob
import matplotlib.image as img
path_covid=list(glob.glob(test_path+"COVID19/"+"/"))
path_normal=list(glob.glob(test_path+"NORMAL/"+"/"))
```

```
[ ] plt.title("Covid 19 Image")
covid_image_sample=img.imread(path_covid[0])
plt.imshow(covid_image_sample)
```

```
[ ] model=Sequential()  
    model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu',input_shape=(64,64,3)))  
    model.add(MaxPool2D(pool_size=(2,2)))  
    # model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu'))  
    # model.add(MaxPool2D())  
    # model.add(Dropout(0.15))
```

```
[ ] model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu'))  
    model.add(MaxPool2D())  
    model.add(Dropout(0.5))
```

```
[ ] model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu'))  
    model.add(MaxPool2D())  
    model.add(Dropout(0.15))
```

```
[ ] model.add(Flatten())  
    model.add(Dense(256,activation='relu'))  
    # model.add(Dropout(0.1))  
    model.add(Dense(64,activation='relu'))  
    model.add(Dense(16,activation='relu'))  
    model.add(Dense(1,activation='sigmoid'))
```

```
[ ] from tensorflow.keras.optimizers import Adam  
    # opt = SGD(lr=0.001)  
    opt = Adam(lr=0.003, beta_1=0.9, beta_2=0.999,  
               epsilon=0.1, decay=0.0)  
    model.compile(loss = "binary_crossentropy", optimizer = opt,metrics=['accuracy'])
```

```
[▶] hist=model.fit(training_set,  
                   steps_per_epoch= 8,  
                   epochs= 10,  
                   validation_data= testing_set)
```

Epoch 1/10

```
h = hist.history  
h.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[ ] plt.plot(h['accuracy'])  
plt.plot(h['val_accuracy'] , c = "red")  
plt.title("acc vs v-acc")  
plt.show()
```

```
[ ] plt.plot(h['loss'])  
plt.plot(h['val_loss'] , c = "red")  
plt.title("loss vs v-loss")  
plt.show()
```

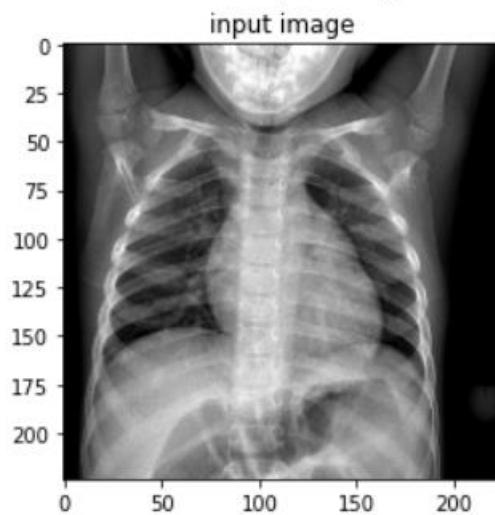
VIII. OUTPUT

Output for VGG-16

```
] path="/home/bibek/Desktop/Project_Sem_8/Projects/Data/train/NORMAL/NORMAL(100).jpg"
```

The chances of image being Covid is : 0.20836330950260162 percent

The chances of image being Normal is : 98.59039187431335 percent



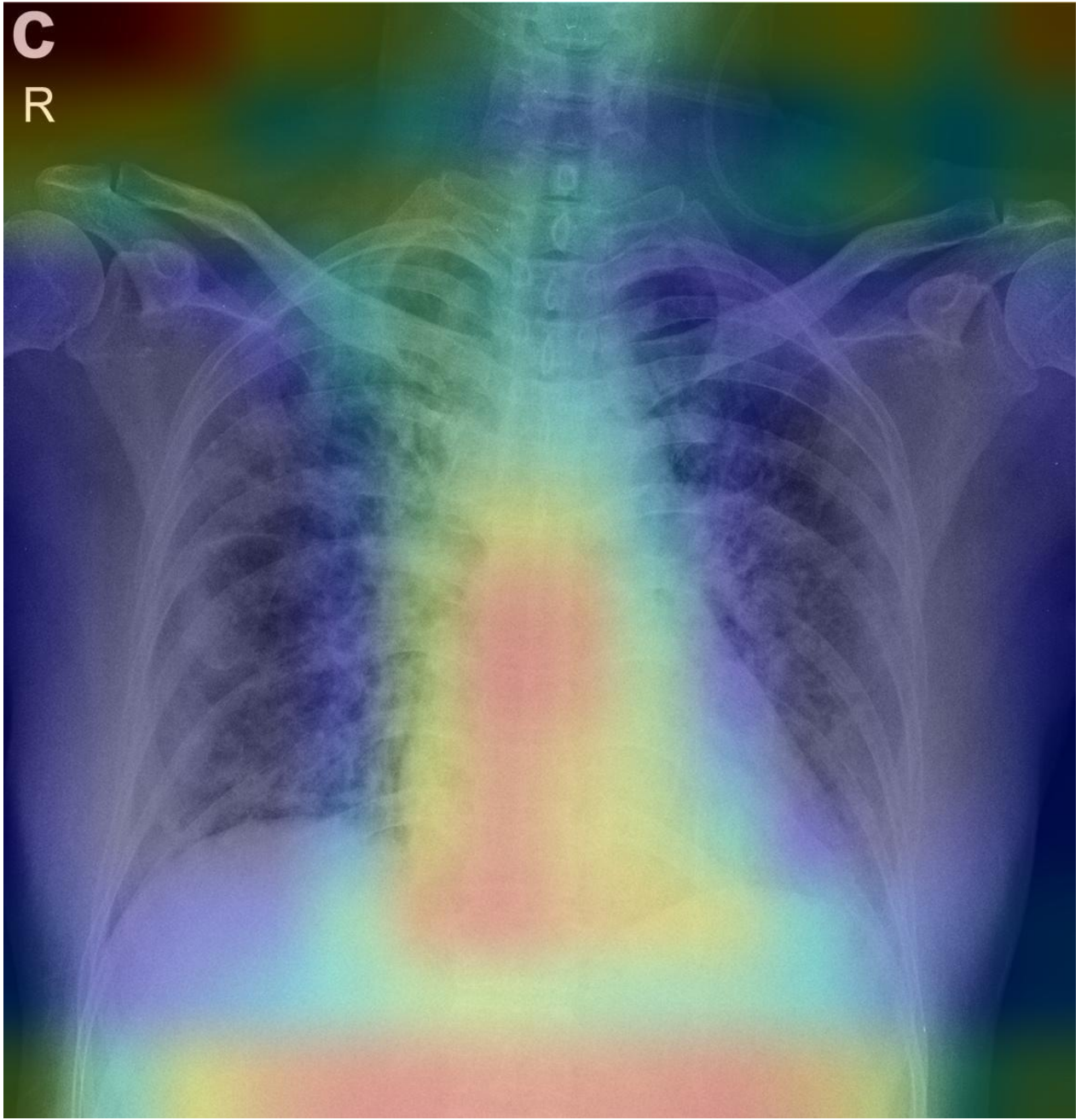
```
#predictions
# provide the path of any image from google or any other source
# the path is already defigned above , but you can also provide the path here to avoid scrolling up

# for covid image : |
path="/home/bibek/Downloads/Small_Size_Data/dataset/covid/1-s2.0-S0929664620300449-gr2_lrg-c.jpg"

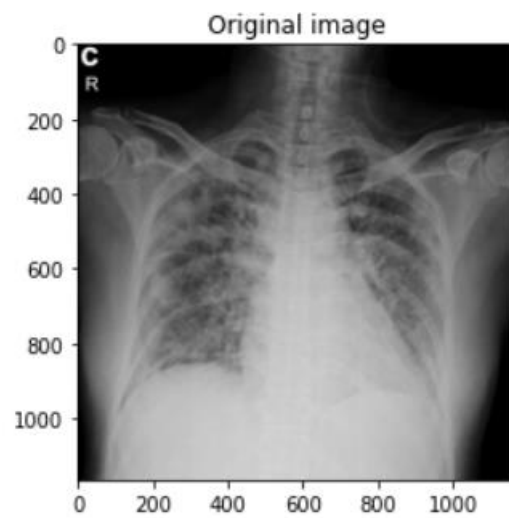
image_prediction_and_visualization(path)
```

The chances of image being Covid is : 94.24750804901123 %
The chances of image being Normal is : 2.01861634850502 %

Image with Heatmap Representing the Covid-19 Spot



the original input image

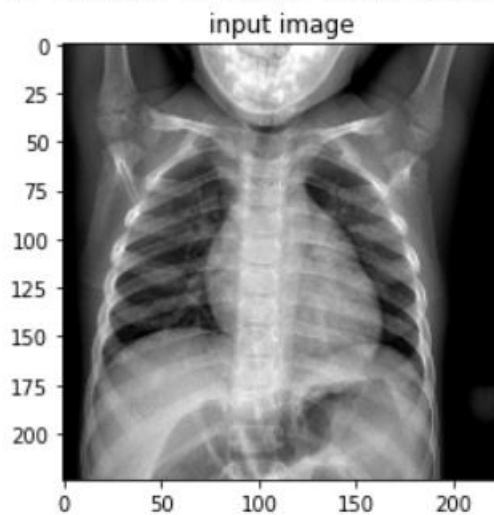



Output for ResNet-50

 The given X-Ray image is of type = Normal

The chances of image being Covid is : 0.0 percent

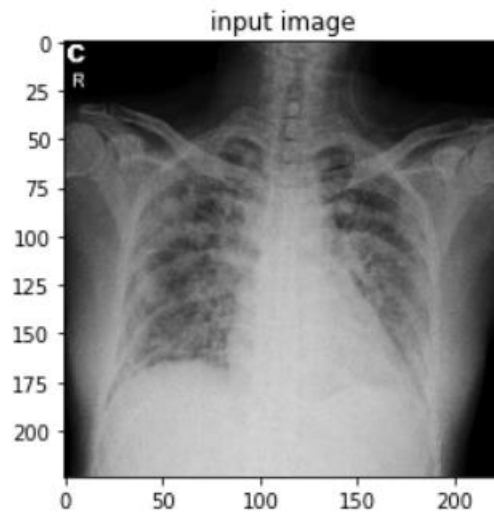
The chances of image being Normal is : 100.0 percent



 The given X-Ray image is of type = Covid

The chances of image being Covid is : 100.0 percent

The chances of image being Normal is : 6.964541654497225e-06 percent



Output for CNN

```
[ ] res = class_type[np.argmax(model.predict(img))]  
    print(f"The given X-Ray image is of type = {res}")  
    print(f"The chances of image being Covid is : {model.predict(img)[0][0]*100} percent")  
    # print(f"The chances of image being Normal is : {model.predict(img)[0][1]*100} percent")
```

The given X-Ray image is of type = Covid
The chances of image being Covid is : 99.96517896652222 percent

```
[ ] path="/home/bibek/Downloads/Small_Size_Data/dataset/normal/NORMAL2-IM-1090-0001.jpeg"  
    img = get_img_array(path)  
    x=model.predict(img)  
    print(x)  
    print(Final_Result(x))  
    print(model.predict(img)[0][0]*100)
```

```
[[0.99996877]]  
Normal  
99.99687671661377
```

IX. OUTPUT VALIDATION AND COMPARISION

Model	Val. Acc	Train. Acc	Val. Loss	Train. loss	Model Size
VGG-16	97.92%	98.75%	4.53%	3.28%	59.5 MB
ResNet-50	99.54%	99.69%	12%	3.70%	97.3 MB
CNN	97.23%	95.37%	10.9%	12.61%	483.9 MB
CNN(Global Minima)	50%	75%	72.44%	54.25%	—————

Table 11:- Comparison between Proposed Models

From the table 11, we can see that ResNet-50 model have the highest validation and training accuracy. But in terms of validation loss and training loss VGG-16 models perform better. I tested more than two test image for every models, all the models classify the image truly that means models is able to identify the covid-19 through the x-ray images. At the end in terms of model size produces by deep learning models, VGG-16 produce least model size. So, in terms of overall performance VGG-16 is the winner.

X. INNOVATION IN MODEL

- ✓ It is easy to use, just for end user they just have to provide the image path. For developer / organizer they just have to create virtual environment and install requirements.txt files.
- ✓ It is open source, which means everyone can use and contribute.
- ✓ Best Accuracy, this project provides best accuracy which is up to around 99%.
- ✓ Since project is open source, so everyone can use it on free of cost.

- ✓ It reduces the actual cost of covid-19 detection (RT-PCR) from 8\$ to 2\$.

1. LIMITATION

- Mobile Application
- Application GUI Can be Improve
- Model size is Heavy
- Feature is Limited

XI. FUTURE WORK

There is always a room for improvements in any software package, however good and efficient it may be done. But the most important thing should be flexible to accept further modification. Right now I am just dealing with model accuracy. In future this software may be extended to include features such as:

- ✓ After reducing model size (below 30 MB) then I am able to create a mobile application.
- ✓ For the newly found virus I will create a dynamic models where even with a small size of data set with the help of stacking and data augmentation technique.
- ✓ I will add some user friendly graphics and tools for models.

XII. CONCLUSION

As few previous studies review different types of deep learning models with their pros and cons, which play a crucial role in the detection of covid-19 with minimal cost. So in this project, I provided a fundamentals of different methods of covid-19 detection using deep learning models with their measurement factors. In conclusion, there are number of issue which should be considered like image dataset, model overfitting problems. Covid-19 datasets play major role to detect covid-19. Along with this image processing, data augmentation and pre trained models help us to improve model

accuracy. With the help of proposed models we can reduce the cost of covid-19 test from 8\$ to 2\$ in minimal time. For the current covid-19 situations VGG-16 performs exceptionally.

XIII. BIBLIOGRAPHY

- [1] B. N. Kurani, K. Pollitz, D. Cotliar, N. Shanosky, and C. C. Kff, “COVID-19 Test Prices and Payment Policy,” pp. 1–8, 2021, [Online]. Available: <https://www.healthsystemtracker.org/brief/covid-19-test-prices-and-payment-policy/>
- [2] A. Gupta, Anjum, S. Gupta, and R. Katarya, “InstaCovNet-19: A deep learning classification model for the detection of COVID-19 patients using Chest X-ray,” *Appl. Soft Comput.*, vol. 99, no. xxxx, p. 106859, 2021, doi: 10.1016/j.asoc.2020.106859.
- [3] E. F. Ohata *et al.*, “Automatic detection of COVID-19 infection using chest X-ray images through transfer learning,” *IEEE/CAA J. Autom. Sin.*, vol. 8, no. 1, pp. 239–248, 2021, doi: 10.1109/JAS.2020.1003393.
- [4] N. C. Pratiwi, N. Ibrahim, Y. N. Fu’adah, and K. Masykuroh, “Computer-Aided Detection (CAD) for COVID-19 based on Chest X-Ray Images using Convolutional Neural Network,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 982, no. 1, 2020, doi: 10.1088/1757-899X/982/1/012004.
- [5] S. Chakraborty, B. Murali, and A. K. Mitra, “An Efficient Deep Learning Model to Detect COVID-19 Using Chest X-ray Images,” *Int. J. Environ. Res. Public Health*, vol. 19, no. 4, 2022, doi: 10.3390/ijerph19042013.
- [6] A. Mangal *et al.*, “CovidAID: COVID-19 Detection Using Chest X-Ray,” pp. 1–10, 2020, [Online]. Available: <http://arxiv.org/abs/2004.09803>
- [7] S. Malik, S. Singh, N. M. Singh, and N. Panwar, “International Journal of Informatics , Information System and Computer Engineering Diagnosis of

- COVID-19 Using Chest X-ray,” *Ojs.Unikom.Ac.Id*, vol. 2, no. March 2020, pp. 55–64, 2021, [Online]. Available: <https://ojs.unikom.ac.id/index.php/injiiscom/article/download/4024/2138>
- [8] L. Duran-Lopez, J. P. Dominguez-Morales, J. Corral-Jaime, S. Vicente-Diaz, and A. Linares-Barranco, “COVID-XNet: A custom deep learning system to diagnose and locate COVID-19 in chest x-ray images,” *Appl. Sci.*, vol. 10, no. 16, pp. 1–12, 2020, doi: 10.3390/app10165683.
 - [9] A. Narin, C. Kaya, and Z. Pamuk, “Department of Biomedical Engineering, Zonguldak Bulent Ecevit University, 67100, Zonguldak, Turkey.,” *arXiv Prepr. arXiv2003.10849.*, 2020, [Online]. Available: <https://arxiv.org/abs/2003.10849>
 - [10] E. B. G. Kana, M. G. Z. Kana, A. F. D. Kana, and R. H. A. Kenfack, “A web-based diagnostic tool for COVID-19 using machine learning on chest radiographs (CXR),” *medRxiv*, 2020, doi: 10.1101/2020.04.21.20063263.
 - [11] T. Ozturk, M. Talo, E. Azra, U. Baran, and O. Yildirim, “Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID- 19 . The COVID-19 resource centre is hosted on Elsevier Connect , the company ’ s public news and information ,” no. January, 2020.
 - [12] A. Irsyad and H. Tjandrasa, “Detection of Covid-19 from Chest CT Images Using Deep Transfer Learning,” *Proc. 2021 13th Int. Conf. Inf. Commun. Technol. Syst. ICTS 2021*, pp. 167–172, 2021, doi: 10.1109/ICTS52701.2021.9608160.
 - [13] H. Kaheel, A. Hussein, and A. Chehab, “AI-Based Image Processing for COVID-19 Detection in Chest CT Scan Images,” *Front. Commun. Networks*, vol. 2, no. August, pp. 1–12, 2021, doi: 10.3389/frcmn.2021.645040.
 - [14] D. P. Tian, “A review on image feature extraction and representation techniques,” *Int. J. Multimed. Ubiquitous Eng.*, vol. 8, no. 4, pp. 385–395, 2013.

- [15] H. C. Shin *et al.*, “Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning,” *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1285–1298, 2016, doi: 10.1109/TMI.2016.2528162.
- [16] H. S. Alghamdi, G. Amoudi, S. Elhag, K. Saeedi, and J. Nasser, “Deep Learning Approaches for Detecting COVID-19 from Chest X-Ray Images: A Survey,” *IEEE Access*, vol. 9, pp. 20235–20254, 2021, doi: 10.1109/ACCESS.2021.3054484.
- [17] J. He, L. Luo, Z. Luo, J. Lyu, M. Ng, and X. Shen, “Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID- 19 . The COVID-19 resource centre is hosted on Elsevier Connect , the company ’ s public news and information ,” no. January, 2020.
- [18] A. Shoeibi *et al.*, “Automated Detection and Forecasting of COVID-19 using Deep Learning Techniques: A Review,” 2020, [Online]. Available: <http://arxiv.org/abs/2007.10785>
- [19] M. Ghaderzadeh and F. Asadi, “Deep Learning in the Detection and Diagnosis of COVID-19 Using Radiology Modalities: A Systematic Review,” *J. Healthc. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/6677314.
- [20] N. Subramanian, O. Elharrouss, S. Al-Maadeed, and M. Chowdhury, “A review of deep learning-based detection methods for COVID-19,” *Comput. Biol. Med.*, vol. 143, p. 105233, 2022, doi: 10.1016/j.combiomed.2022.105233.
- [21] M. M. Fachi, R. O. Vilhena, and A. F. Cobre, “Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID- 19 . The COVID-19 resource centre is hosted on Elsevier Connect , the company ’ s public news and information ,” no. January, 2020.

- [22] J. Wu, “Introduction to Convolutional Neural Networks,” *Introd. to Convolutional Neural Networks*, pp. 1–31, 2017, [Online]. Available: https://web.archive.org/web/20180928011532/https://cs.nju.edu.cn/wujx/teaching/15_CNN.pdf
- [23] S. Albawi, O. Bayat, S. Al-Azawi, and O. N. Ucan, “Social touch gesture recognition using convolutional neural network,” *Comput. Intell. Neurosci.*, vol. 2018, 2018, doi: 10.1155/2018/6973103.
- [24] WHO, “WHO-convened Global Study of Origins of SARS-CoV-2 : China Part (14 January-10 February 2021)LI, Q. et al. Early Transmission Dynamics in Wuhan, China, of Novel Coronavirus–Infected Pneumonia. *New England Journal of Medicine*, v. 382, n. 13, p. 1199–1207,” *Jt. WHO-China Study Team Rep.*, no. February, p. 120, 2021.
- [25] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0197-0.
- [26] World Health Organization, “Reducing public health risks associated with the sale of live wild animals of mammalian species in traditional food markets,” *Interim Guid.*, no. April, pp. 1–8, 2021.
- [27] A. Gupta, S. Gupta, and R. Katarya, “Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID- 19 . The COVID-19 resource centre is hosted on Elsevier Connect , the company ’ s public news and information ,” no. January, 2020.
- [28] “What is COVID-19,” no. December, 2019.
- [29] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 2818–2826, 2016, doi:

10.1109/CVPR.2016.308.

- [30] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.