

CORONAVIRUS ANALYSIS USING DEEP LEARNING TECHNIQUES

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

by

S.VICTOR ABILASH [Reg.No. 37110834]

VINO GEOFFREY [Reg.No. 37110840]



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

MARCH – 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **VICTOR ABILASH [Reg.No. 37110834]** who carried out the project entitled "**CORONAVIRUS ANALYSIS USING DEEP LEARNING TECHNIQUES**" under my supervision from November 2020 to April 2020.

Internal Guide

Dr. R. SATHYA BAMA KRISHNA, M.E., Ph.D.,

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **S. VICTOR ABILASH [Reg.No. 37110834]** hereby declare that the Project Report entitled “**Coronavirus Analysis using Deep Learning Techniques**” was done by us under the guidance of **Dr. R. Sathya Bama Krishna, ME., Ph.D.,** at **Sathyabama Institute of Science and Technology, Chennai** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **COMPUTER SCIENCE AND ENGINEERING.**

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr.SASIKALA.T.,M.E.,Ph.D., Dean, School of Computing** and **Dr.S.VIGNESHWARI.,M.E.,Ph.D., Head of the Department, Dept. of Computer Science and Engineering** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. R. SATHYABAMA., M.E., Ph.D.,** Assistant Professor for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of **COMPUTER SCIENCE AND ENGINEERING** who were helpful in many ways for the completion of the project.

ABSTRACT

The aim of this project is to build an efficient DL model for Coronavirus pandemic Analysis with Deep Learning Techniques using medical images and implementing it with the help of Python Programming Language, also to deploy a fully working website connected to the DL model to automatically predict and produce the results, whether the given input (X-ray image) of the person is COVID positive or Negative. Since December 2019, the coronavirus disease (COVID-19) outbreak has caused many death cases and affected all sectors of human life. With gradual progression of time, COVID-19 was declared by the world health organization (WHO) as an outbreak, which has imposed a heavy burden on almost all countries, especially ones with weaker healthcare systems and ones with slow responses due to lack of knowledge. In the field of healthcare, deep learning has been implemented in many applications, e.g., diabetic retinopathy detection, lung nodule classification, fetal localization, and thyroid diagnosis. Numerous sources of medical images (e.g., X-ray, CT, and MRI) make deep learning a great technique to combat the COVID-19 outbreak. Motivated by this fact, a large number of research works have been proposed and developed for the initial months of 2020. In this project we are going to implement the model by collecting reliable dataset that was verified by certified radiologists. The dataset will be of two types, training and testing dataset which further contains COVID Negative dataset and positive dataset, featuring various X-ray images. We are going to perform the necessary operations to build CNN model, ResNet 50. The model is declared and trained with a large variety of parameters and is passed into the 50 layers of the ResNet 50 architecture. When we arrive with the most efficient model. We are going to pass the testing dataset to predict COVID positive or negativity of the given input. To develop the Deep Learning Model we are going to be using Python for its extensive libraries to invoke various Deep Learning Techniques and implement the project in google collab as we require a spacious amount of GPU in the local machine to train the model.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	VI
	LIST OF FIGURES	X
	LIST OF TABLES	XI
	LIST OF ABBREVIATION	XII
1.	INTRODUCTION	11
	1.1 CORONAVIRUS PANDEMIC	11
	1.2 DEEP LEARNING	11
	1.3 NEURAL NETWORK	12
	1.3.1 RESNET 50	13
	1.4 PYTHON	13
	1.4.1 GOOGLE COLAB	14
	1.5 OBJECTIVE	14
2.	LITERATURE SURVEY	15
	2.1 EXISTING SYSTEM	15
	2.1.2 THE ROC CURVE, PRECISION- RECALL CURVE AND CONFUSION MATRIX	15
	2.2 PROPOSED SYSTEM	18
	2.3 REVIEWS OF THE LITERATURE SURVEY	18
3.	METHODOLOGY	21
	3.1 HARDWARE REQUIREMENTS	21
	3.2 SOFTWARE REQUIREMENTS	21
	3.3 MODULES	22
	3.3.1 DATA COLLECTION	24
	3.3.2 CLASSIFICATION	24
	3.3.3 LOCALIZATION	25
	3.3.4 DETECTION	25
	3.3.5 SEGMENTATION	26
	3.3.6 RESNET 50	27

	3.3.7 REGISTRATION	30
	3.3.8 DEPLOYING WEB APPLICATION OF THE MODEL	32
4.	RESULTS AND DISCUSSION	34
5.	CONCLUSION AND FUTURE WORK	36
	5.1 CONCLUSION	36
	5.2 FUTURE WORK	36
	REFERENCES	37
	APPENDIX	38
	A) SCREENSHOT	38
	B) SOURCE CODE	42

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
1.1	PRECISION-RECALL CRUVE FOR 4 TYPES OF CNN MODELS	16
1.2	ROC CURVE OF 4 TYPES OF CNN MODEL	17
3.1	COVID CHEST X-RAY IMAGES	20
3.2	COVID CHEST CT SCAN IMAGES	20
3.3	TRAINING DATASET WITH LABELS	21
3.4	LOCALIZATION	22
3.5	TENSOR FORM OF THE IMAGES	24
3.6	RELU	25
3.7	RESNET50 ARCHITECTURE	26
3.8	PERFORMANCE OF THE MODEL WITH EVERY EPOCHS	29
3.9	WEBSITE FRONTEND DESIGN	30
4.1	PERFORMANCE GRAPH	31
4.2	CONFUSION MATRIX	32
A.1	RESNET 50 MODEL	36
A.2	TESTING RESULTS	36
A.3	WEBSITE HOMEPAGE	37

A.4	WEBSITE ABOUT PAGE	37
A.5	WEBSITE FORM ENTRY PAGE	38
A.6	DATABASE OF THE WEBSITE	38

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
1.1	SUMMARY OF DL MODEL ON MEDICAL IMAGE PROCESSING ON COVID-19	16
3.1	DATASET DISTRIBUTION	20
3.2	DISTRIBUTION OF RESNET 18,34,50 ETC.,	26

LIST OF ABBREVIATIONS

ABBREVIATIONS	EXPANSION
DL	DEEP LEARNING
RESNET50	RESOLUTION NEURAL NETWORK
RELU	RECTIFIED LINEAR UNITS
ML	MACHINE LEARNING
COVID-19	CORONAVIRUS -19
CNN	CONVOLUTIONAL NEURAL NETWORK
ROC	RECEIVER OPERATING CHARACTERISTICS
AUC	AREA UNDER ROC CURVE
NN	NEURAL NETWORK
DNN	DEEP NEURAL NETWORK
AVG POOLING	AVERAGE POOLING
MAX POOLING	MAXIMUM POOLING

CHAPTER 1

INTRODUCTION

1.1 CORONA VIRUS PANDEMIC

The COVID-19 pandemic is causing a major outbreak in more than 150 countries around the world, having a severe impact on the health and life of many people globally. One of the crucial step in fighting COVID-19 is the ability to detect the infected patients early enough, and put them under special care. Detecting this disease from radiography and radiology images is perhaps one of the fastest ways to diagnose the patients. Some of the early studies showed specific abnormalities in the chest radiograms of patients infected with COVID-19.

Coronaviruses vary significantly in risk factor. Some can kill more than 30% of those infected, such as MERS-CoV, and some are relatively harmless, such as the common cold. Coronaviruses can cause colds with major symptoms, such as fever, and a sore throat from swollen adenoids. Coronaviruses can cause pneumonia (either direct viral pneumonia or secondary bacterial pneumonia) and bronchitis (either direct viral bronchitis or secondary bacterial bronchitis). The human coronavirus discovered in 2003, SARS-CoV, which causes severe acute respiratory syndrome (SARS), has a unique pathogenesis because it causes both upper and lower respiratory tract infections.

1.2 DEEP LEARNING

Deep learning also known as deep structured learning is a part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design where they have produced results comparable to and in some cases surpassing human expert performance.

It is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces. Neural networks are inspired by our understanding of the biology of our brains – all those interconnections between the neurons. But, unlike a biological brain where any neuron can connect to any other neuron within a certain physical distance, these artificial neural networks have discrete layers, connections, and directions of data propagation.

1.3 NEURAL NETWORK

When it comes to neural network there are two types of neural networks, Artificial neural network and Deep neural network(DNN). Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analysing example images that have been manually labelled as "cat" or "no cat" and using the analytic results to identify cats in other images.

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. For example, a DNN that is trained to recognize dog breeds will go over the given image and calculate the probability that the dog in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks.

DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or weights, to connections between them. The weights and inputs are multiplied and return an output between them 0 and 1. If the network did not accurately recognise a particular patten, an algorithm

would adjust the weights. Recurrent neural networks (RNNs), in which data can flow in any direction, are used for applications such as language modelling. Long short-term memory is particularly effective for this use. Convolutional deep neural networks (CNNs) are used in computer vision, which we are using to mainly implement our image caption generator.

1.3.1 ResNet 50

ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance. Taking advantage of its powerful representational ability, the performance of many computer vision applications other than image classification have been boosted, such as medical image processing object detection and face recognition.

However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitively small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly. Before ResNet, there had been several ways to deal the vanishing gradient issue, for instance, adds an auxiliary loss in a middle layer as extra supervision, but none seemed to really tackle the problem once and for all.

1.4 PYTHON

Python Programming language is an high level language with object oriented functions. Python is one of the high level programming language that uses an interpreter with dynamic semantics rather than a compiler. Python because of its efficiently built data structures and its interpreting capabilities combined dynamic binding and typing can be used as a scripting language, for rapid development etc., Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

The vast majority of the libraries used for data science or machine learning have Python interfaces, making the language the most popular high-level command interface to for machine learning libraries and other numerical algorithms. Hence we are going to implement Coronavirus Analysis with Deep Learning Technique using medical images with Python programming language.

1.4.1 GOOGLE COLAB

Google is quite aggressive in AI research. Over many years, Google developed AI framework called TensorFlow and a development tool called Colaboratory. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply Colab.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.

Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

1.5 OBJECTIVE

The objective of this project is to learn the concepts of a Convolutional neural network (CNN) and ResNet 50 model and building a working model for Coronavirus analysis with Deep learning techniques using medical images by implementing CNN and ResNet 50. We are going to implement the concept using Python programming language with Google Colab as the selected IDE because of the demanding nature of building an efficient DL Model .

This project can be applied in various real life application and some of the examples are as follows :

1. Image Recognition
2. Visual art processing
3. Medical Image processing
4. Fraudulent/burglary detection etc.,

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SYSTEM

Although millions of patients are getting infected by the disease, there still exists the absence of publicly available large datasets focusing especially on tests with missing infections. The accuracy of any DL model depends on the availability of data, and in the case of COVID-19, this is a major concern. The accuracy of the DL model becomes questionable due to the use of insufficient data. Also, accessibility to CT and X-ray images from wider demographics is a requirement that is difficult to collect from healthcare organizations. Also, the implementation of policy decisions based on DL based implementations requires access to behavioural attributes such as work, education, family background. All of this information is necessary to understand how people remain asymptomatic even after getting infected by the virus. These limitations stand predominant in achieving optimized scientific decisions. But as presented in the paper, various studies are being conducted relentlessly to eliminate such challenges and acquire more clarity in COVID-19 diagnosis.

DL has gained immense momentum, popularity and has generated impressive results with simple 2D images, there exist limitations in achieving a similar level of performance in medical image processing. Research work in this regard is still-in-progress and some of the lessons learned are mentioned below:

One of the most inhibiting factors is the unavailability of large datasets with high-quality images for training. In this case, synthesizing of the data is a possible solution so that the data collected from varied sources could be integrated together. The majority of the state-of-the-art DL models are trained for 2D images. However, CT and MRI are usually 3D and hence add an additional dimension to the existing problem. Since the conventional DL models are not adjusted to this, experience plays a major role when DL models are implemented on these images.

The non-standardized process of collecting image data is one of the major issues in medical image processing. It is important to understand that with the increase in data variety, the need of larger datasets arise to ensure the DL algorithm generates robust solutions. The best possible way to resolve this issue is the application of transfer learning, which makes pre-processing efficient and eliminates scanner and acquisition issues.

Table 1.1 Summary of DL models with medical image processing for COVID-19 w.r.t [14]

Summary of deep learning for medical image processing in COVID-19.

Ref.	Dataset	Methods used	Evaluation metrics	Research challenges
Ozturk et al. (2020)	Chest X-ray	CNN, DarkNet, DarkCovidNet	98.08% accuracy	Use of a limited number of COVID- 19 X-ray images
Chae et al. (2018)	OLS, ARIMA	DNN, LSTM	Average performance by 24% and 19%, respectively.	Prediction of infectious disease.
Zhou et al. (2020)	CT scan Images	logistic regression model	89.47% sensitivity, 67.42% specificity	The laboratory testing methods are not uniform among different hospitals.
Zhang, Yang, et al. (2020)	Electronic medical records	Univariate and multivariate Cox regression analysis	Hazard ratio and confidence interval was used to detect the adverse outcome	Predicting the adverse outcome at the early stage of COVID-19.
Chen, Wu, et al. (2020)	CT scan images	DL-based model	100% sensitivity, 93.55% specificity, 95.24% accuracy.	Achieving consistent results between the expert and model.
Li et al. (2020)	chest CT exams	COVNet	90% sensitivity, 96% specificity	Unable to categorize the disease into different severity levels.
Ai et al. (2020)	chest CT	DL	97% sensitivity	The clinical and laboratory data were limited when regional hospitals were overloaded.
Butt et al. (2020)	CT chest images	CNN models	0.996 AUC, 98.2% sensitivity, 92.2% specificity	Achieving fast and reliable detection of COVID-19 from chest CT datasets.
Gozes et al. (2020)	CT scan	2D Slice Analysis, 3D Volume Analysis	98.2% sensitivity, 0.996 AUC, 92.2% specificity	Challenging to achieve high accuracy in detection of Coronavirus as well as quantification and tracking of disease burden.
Wang and Wong (2020)	Chest X-ray	COVID-Net	93.3% test accuracy	AI systems leveraging the more readily available and accessible CXR imaging modality.

2.1.2 The ROC Curve, Precision Recall Curve, and Confusion Matrix

It is hard to compare different models only based on their sensitivity and specificity rates, since these rates change by varying the cut-off thresholds. To see the overall comparison between these models, we need to look at the comparison for all possible threshold values. One way to do this, is through the precision-recall curve, which provides the precision rate as a function of recall rate. Precision is defined as the true positive images divided by the total number of images flagged as positive by the model, and the recall is the same as sensitivity rate. The precision-recall curve of these four models is shown in Figure 1.1.

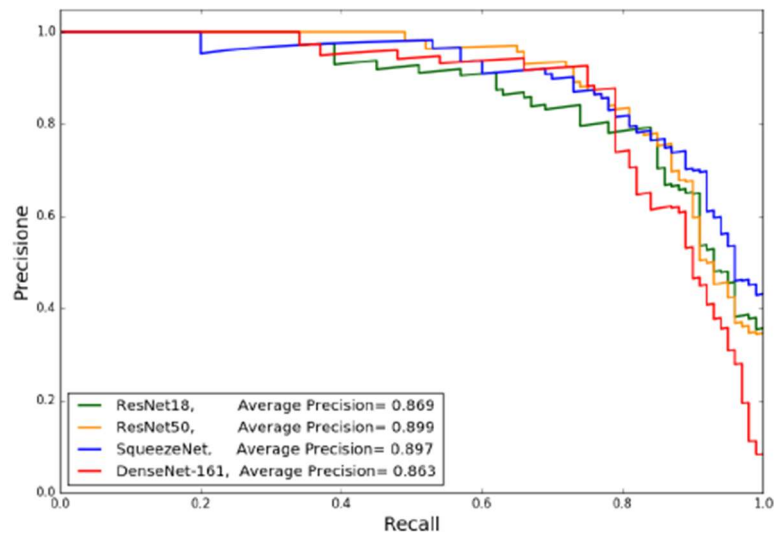


Fig. 1.1 The precision-recall curve of the 4 types of CNN Models w.r.t [1]

Another way to do this, is through the Receiver operating characteristic (ROC) curve, which provides the true positive rate as a function of false positive rate. The ROC curve of these four models is shown in Figure 1.2. All models have a similar performance according to the AUC with the SqueezeNet achieving a slightly higher AUC than the other models. It is worth mentioning that for highly imbalanced test sets, the AUC may not be a good indicative of model performance (as it can be very high), and looking at average -precision and precision-recall curve would be a better choice in that case. Here we provided both curves for the sake of completeness.

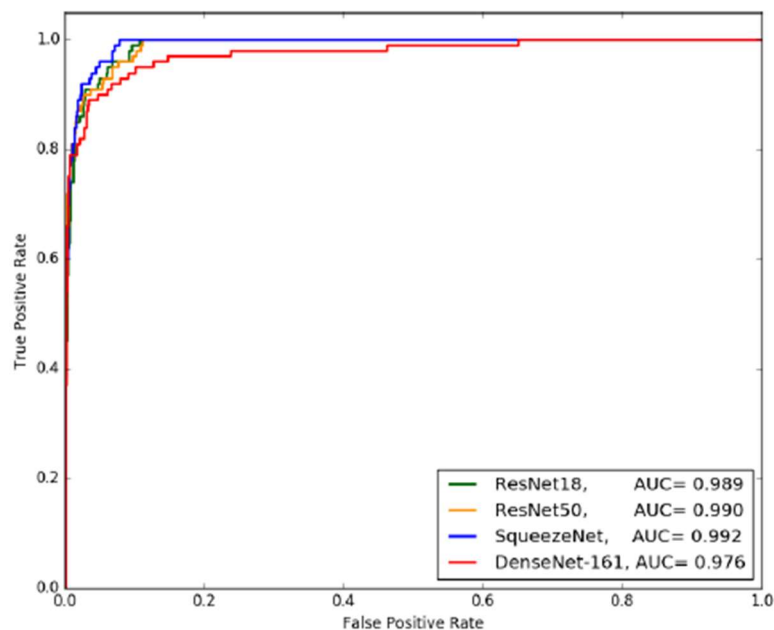


Fig. 1.2 ROC curve of 4 CNN Models on COVID-19 dataset w.r.t [1]

However, due to the limited number of COVID-19 images publicly available so far, further experiments are needed on a larger set of cleanly labelled COVID-19 images for a more reliable estimation of the accuracy of these models.

2.2 PROPOSED SYSTEM

In this project, after analysing the pros and cons of the existing system we have decided to choose ResNet 50, Deep learning model for the analysis of Coronavirus with medical images. We have chosen this particular type of the model because of the accuracy it provides in both the ROC and AUC curve, which has greater accuracy after SqueezeNet model. We are going to implement the project by collecting the testing and training dataset from Chestpert-X-ray and COVID-19 X-ray images dataset which is been verified by the certified radiologists for Positive COVID data. We are going to use a pre-trained ResNet 50 model which was trained with the weights and parameters of COVID-19 images. After training the model, we are going to test the model with the testing dataset and produce the results whether the given input (X-Ray Image) is COVID Positive or Negative. We have also deployed a web application with the model, so it will help the public to make it affordable and also time easing factor. The web application is built with the flask framework provided in Python for deploying DL models into an web application. We have used HTML, CSS AND JavaScript for the front-end and Python for the backend.

2.3 REVIEWS OF THE LITERATURE SURVEY

Title : Deep learning and medical image processing for coronavirus (COVID-19) pandemic: A survey

Author : Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Quoc-Viet Pham b, hippa Reddy Gadekallu, Siva Rama Krishnan S, Chiranji Lal Chowdhary, Mamoun Alazab, Md. Jalil Piran.

Year : 2019

Among solutions to combat the COVID-19 pandemic, DL has been considered a great technique to provide intelligent solutions. Motivated by many applications of

DL for medical image processing in the last decade, we have provided summarized recent efforts about the COVID- 19 outbreak for smart, healthy cities. DL has been employed to achieve various solutions for COVID-19 disruption, including outbreak prediction, a virus spread tracking, diagnosis and treatment, vaccine discovery and drug research. Despite promising results, the successful use of DL to process COVID-19 medical images still requires considerable time and effort as well as close operation between different parties from government, industry, and academia. We have also enlisted a number of challenges and issues associated with existing studies such as data privacy, variability of outbreak pattern, regulation and transparency, and distinction between COVID-19 and non-COVID-19 symptoms. Finally, we have discussed a number of future directions of DL applications for COVID-19 medical image processing. We believe that the COVID-19 outbreak will be ending soon with help from DL and image processing techniques as well as many other technologies such as biomedicine, data science, and mobile communications. We also hope that our work is a good source of reference and can drive many novel studies on DL and medical image processing in the battle against the COVID-19 outbreak.

Title : Pneumonia Detection Using Convolutional Neural Networks

Author : Sammy V. Militante, Brandon G. Sibbaluca

Year : 2020

This research explores the deep learning function in detecting pneumonia through computer vision using five convolutional neural network models. Every CNN models identified are assessed physically so that the framework in the feature extraction and fine-tuning are employed. The pneumonia infected diseases and normal chest x-ray image dataset are acquired from the Radiological Society of North America database. Our study enables to identify among the five models the best model to detect pneumonia. The AlexNet, LeNet, GoogleNet, and VGGNet are the best model based on the observations of the researchers having an accuracy rate of 95% to 97%. The ResNetmodel has the lowest accuracy rate of 74%. All the models have performed well on detecting pneumonia and normal chest x-rays. With this work, a distinguished process of diagnosing and detecting pneumonia can helpful in

providing medical services. For future studies, adaptation of other convolutional neural network architectures like Inception-v3, shuffle Net, and Mobile Net architectures for pneumonia detection must be implemented and the optimization of hyper-parameters should also be considered to improve the accuracy of the model. This study will help medical work force in their decision making for a real-time application of the use of accurate model in detecting pneumonia and discover the potential of diagnosing pneumonia using deep learning.

Title : Deep-COVID: Predicting COVID-19 From Chest X-Ray Images Using Deep Transfer Learning

Author : Shervin Minaee, Rahele Kafieh, Milan Sonka, Shakib Yazdani, Ghazaleh Jamalipour Soufi

Year : 2020

We reported a deep learning framework for COVID-19 detection from Chest X-ray images, by fine-tuning four pre-trained convolutional models (ResNet18, ResNet50, SqueezeNet, and DenseNet-121) on our training set. We prepared a dataset of around 5k images, called COVID-Xray-5k (using images from two datasets), with the help of a board-certified radiologist to confirm the COVID-19 labels. We make this dataset publicly available for the research community to use as a benchmark for training and evaluating future machine learning models for COVID-19 binary classification task. We performed a detail experimental analysis evaluating the performance of each of these 4 models on the test set of of COVID-Xray-5k Dataset, in terms of sensitivity, specificity, ROC, and AUC. For a sensitivity rate of 98%, these models achieved a specificity rate of around 90% on average. This is really encouraging, as it shows the promise of using X-ray images for COVID-19 diagnostics. This study is conducted on a set of publicly available images, which contains around 200 COVID-19 images, and 5,000 non-COVID images. The presented work is reflecting one of the earliest Covid-19 chest X-ray analysis and dataset preparation attempts, which brings time-sensitive relevance in combining these two aspects. However, due to the limited number of COVID-19 images publicly available so far, further experiments are needed on a larger set of cleanly labelled COVID-19 images for a more reliable estimation of the accuracy of these models.

CHAPTER 3

METHODOLOGY

3.1 HARDWARE REQUIREMENTS:

Processor	:	INTEL Core i5
Motherboard	:	Genuine Intel
RAM	:	Min. 4 GB
Hard Disc	:	80 GB
GPU	:	2 GB

3.2 SOFTWARE REQUIREMENTS:

Operating System	:	Windows 10
Technology Used	:	Deep Learning
IDE	:	Google Colab
Libraries Used	:	numpy, pandas, matplotlib, torch, sklearn, os etc.,
Emulators	:	Virtual GPU

PyTorch is python native, and integrates easily with other python packages, which makes this a simple choice for researchers. Many researchers use Pytorch because the API is intuitive and easier to learn, and get into experimentation quickly, rather than reading through documentation.

Here, I am using Residual Network (ResNet) via transfer learning, a popular deep learning neural network model, to classify COVID-19 chest X-ray scans. This dataset consists of around 3250 images in the training and testing set respectively. It is quite small by today's big era standard, and it presents some challenges such as data imbalance that I will attempt to mitigate. By leveraging the power of transfer learning, we are able to 'transfer' the weights of low-level features (e.g., lines, shapes, etc) that were detected in a pretrained model. Doing so saves us the need to train a model from scratch, which can be challenging for those who do not have a 'big' enough dataset or computational resources. Following are the modules I have used to build this project :

1. Data Collection

2. Classification
3. Localization (Pre- processing)
4. Detection
5. Segmentation
6. Registration (Final Model)
7. Deploying web application for the model.

3.3 MODULES

3.3.1 Data Collection

Two sources are used to create this dataset for this project and they are as follows:

- Covid-Chestxray-Dataset, for COVID-19 X-ray samples
- ChexPert Dataset, for Non-COVID samples

COVID-19 samples from Covid-Chestxray-Dataset are extracted from a several publications, and it is important to verify all their labels. With the help of a board certified radiologist, we went through X-ray images of COVID-19 samples, and only kept those which were selected to have a clear sign of COVID-19 by our radiologist. We also only kept the posterior-anterior images.

Out of all COVID-19 X-ray images in Covid-Chestxray-Dataset, a total of 3257 images are verified by our board-certified radiologist to have a clear sign of COVID-19, and are used in our dataset. Out of these images, 2357 are used for training and 1000 for test images (due to some consideration with respect to maximum confidence interval for sensitivity rate).

As the number of COVID-19 samples are much fewer than the number of Non-COVID samples, we used several data-augmentation techniques (as well as over-sampling) to increase the number of COVID-19 samples in training, to have a less imbalanced training set. Hopefully more cleanly labelled X-ray images from COVID-19 cases become available soon, so we do not have this imbalanced data issue.

The number of samples from each class (COVID-19, and Non-COVID) in our dataset is shown below. Here we share a subset of 580 non-Covid images for training set which we found to be sufficient for training the model:

Table 3.1 Dataset Distribution

Split	COVID-19	Non-COVID
Training Set	84 (357 after augmentation)	2000
Test Set	100	1000

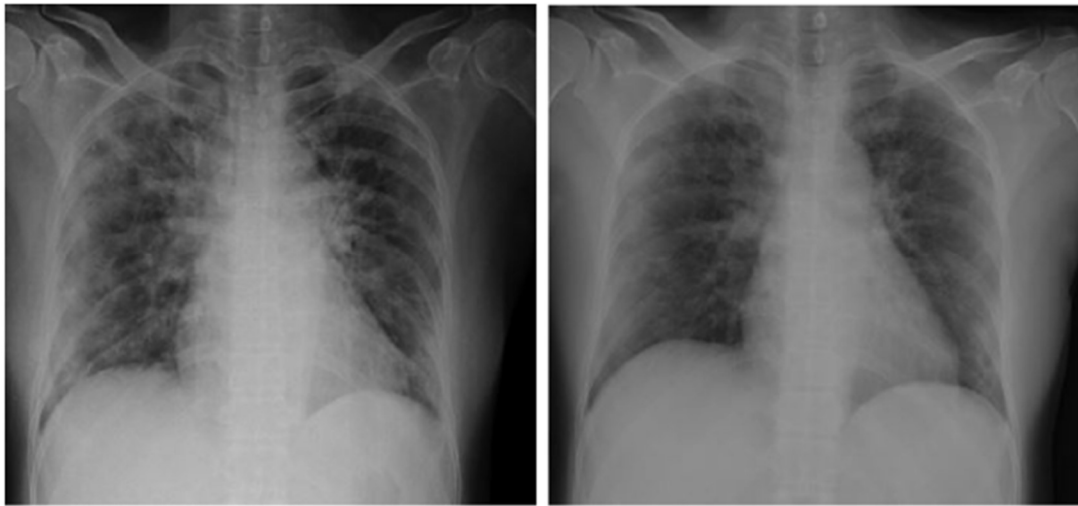


Fig. 3.1 COVID Chest X-ray Images w.r.t [14]

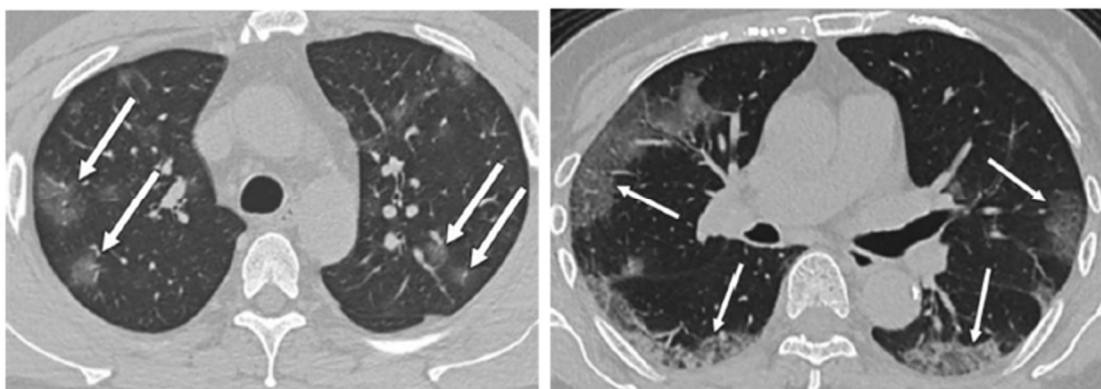


Fig. 3.2 COVID Chest CT Scan Images w.r.t [14]

Now we transform the images to work with Pytorch library provided by python. Pytorch requires us to transform the images into tensors before loading the images when we need it for training. Doing so means that images are not all loaded at once, consuming most of your memory.

3.3.2 Classification

Classification is often termed as Computer-Aided Diagnosis (CAD). Classification plays a significant role in medical image processing. During the classification processing phase, one or even more images are taken as input samples, and a single diagnosis factor is generated as an output which classifies the image. The detection procedure involves 84 chest X-ray images.

- During the classification phase, one or more images are taken as input samples and a single diagnosis factor is generated as an output which classifies the images (Dataset accordingly).
- The experimentation process comprises of hidden layers, which takes the X-ray & CT images as input and provide a response to the output layer of the lung nodule.

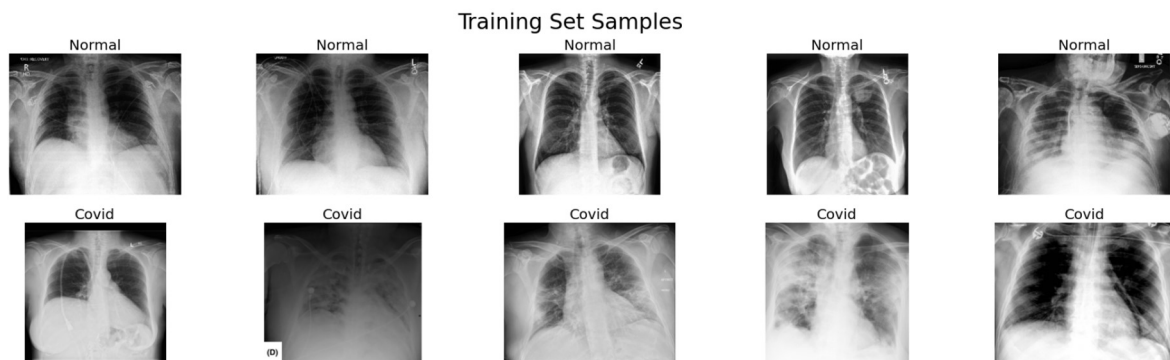


Fig. 3.3 Training dataset with labels

The classification of the image is going to be performed based on the gradient descend of the given dataset. Gradient Descend: The equation below describes what gradient descent does: b is the next position of our climber, while a represents his current position. The minus sign refers to the minimization part of gradient descent. The gamma in the middle is a waiting factor and the gradient term ($\Delta f(y)$) is simply the direction of the steepest descent.

$$x = y - \beta \Delta f(y)$$

Where $x \rightarrow$ current position, $F(y) \rightarrow$ is the steepest descent which should be more than zero for the model to learning curve to be high.

3.3.3 Localization

Localization aka pre-processing stage. In the classification, the images are fed to CNN, and the contents of the image are revealed. After the image classification is done, the next step in the detection of the disease is the image localization, which is responsible for placing the bounding box around the output position, which is called as classification with localization, the term localization here refers to figuring out the disease in the image. The localization of anatomy is a crucial pre-processing phase in a clinical diagnosis that enables the radiologist to recognize certain essential features. During recent years, several research works have been conducted using DL models to localize the disease. We used 3D CNN for landmarking in medical images. Spatial Configuration-Net (SCN) architecture was used to combine accurate response with landmark localization. Experimental evaluation of 3D image datasets using CNN provides higher accuracy.

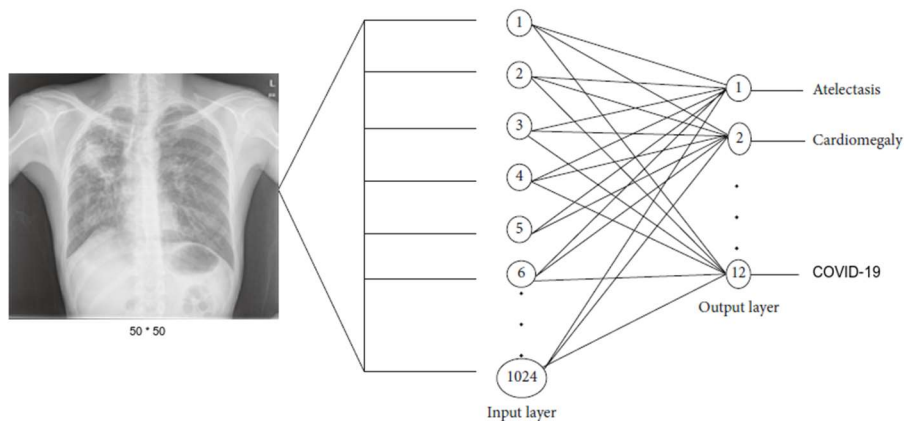


Fig. 3.4 Localization

3.3.4 Detection

Creating accurate ML models capable of classifying, localizing, and detecting multiple objects in a single image remained a core challenge in computer vision. With recent advancements in DL and computer vision models, medical image

detection applications are more comfortable to develop than ever before. Object detection allows for the recognition and localization of multiple objects within an image or video. Object detection is a computer vision technique that is used to identify instances of real-world objects. Object detection techniques train predictive models or use matching templates to locate and identify objects. Object detection algorithms use extracted features and learning algorithms to identify object type instances. Object detection is a key technology behind applications such as video surveillance, image retrieval system, and medical diagnostics.

We are going to train ResNet50 model for detecting objects as in the lumps, clusters or lung nodules which are abnormal from a normal lung.

3.3.5 Segmentation

Since we are working with medical dataset consisting of X-ray images, they might not be consistent throughout the dataset. Hence, we need to perform segmentation on the images to make it regular and uniform dataset. Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks. Note that it increase diversity by introducing random alterations of the original image, but it does not increase the number of images/samples of the dataset.

Also note that it is generally recommended to do **channel-wise data normalization** on the images. This means that for each channel (RGB), we subtract the mean and divide by the standard deviation of each channel of the entire dataset, from each pixel. This serves to prevent any color from having a wider influence/impact on the weights during training as one of the color can have a wider range in the dataset.

PyTorch datasets also allow us to specify one or more transformation functions which are applied to the images as they are loaded. torchvision.transforms contains many such predefined functions. Here we will apply the following:

1. Resize and crop to 224x224 as many images are of different sizes

2. Data Augmentation: random horizontal flip, random rotation and random grayscale
3. Convert images into PyTorch tensors

Once we are done with Data Segmentation the dataset is completely ready for us to go ahead and train the model with the X-ray Images.

```

↳ torch.Size([3, 224, 224]) 0
tensor([[[[0.0000, 0.0000, 0.0000, ..., 0.6353, 0.6353, 0.6314],
          [0.0000, 0.0000, 0.0000, ..., 0.5882, 0.5882, 0.5843],
          [0.0000, 0.0000, 0.0000, ..., 0.5490, 0.5451, 0.5451],
          ...,
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]]],

        [[0.0000, 0.0000, 0.0000, ..., 0.6353, 0.6353, 0.6314],
          [0.0000, 0.0000, 0.0000, ..., 0.5882, 0.5882, 0.5843],
          [0.0000, 0.0000, 0.0000, ..., 0.5490, 0.5451, 0.5451],
          ...,
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]]],

        [[0.0000, 0.0000, 0.0000, ..., 0.6353, 0.6353, 0.6314],
          [0.0000, 0.0000, 0.0000, ..., 0.5882, 0.5882, 0.5843],
          [0.0000, 0.0000, 0.0000, ..., 0.5490, 0.5451, 0.5451],
          ...,
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]]]])

```

Fig. 3.5 Tensor form of the images

3.3.6 ResNet50

A Residual Network is just like any other CNN network architecture, except that they use a residual block. This adds the original input back to the output feature map obtained by passing the input through one or more convolutional layers:

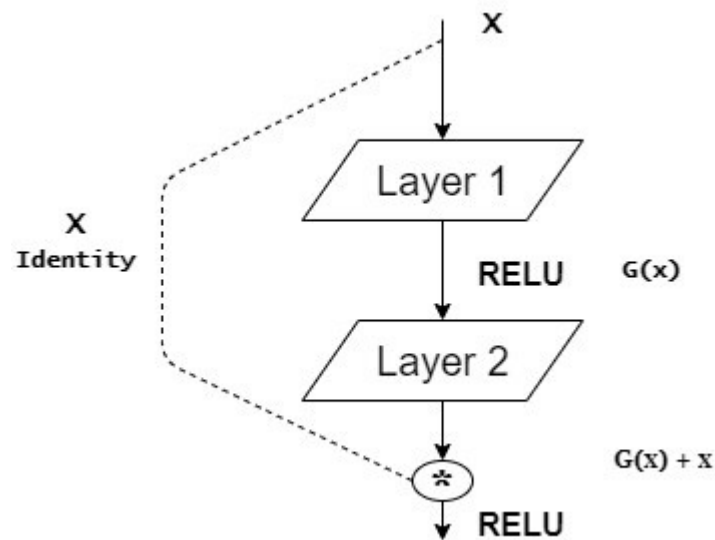


Fig. 3.6 ReLU in ResNet 50

This small change produces a drastic improvement in performance. It allows deeper networks to be trained without hurting performance as some layers can be skipped (residual blocks are also called skip-connections).

The main objective of ResNet is that the layers absorb and learn from the underlying mapping, this network can fit the residual mapping. Hence, here $H(x)$ is the initial mapping, let the network mapping fit be as follows,

$$G(x) := H(x) - x, \text{ which results in } H(x) := G(x) + x.$$

The advantage of skipping the connection is that, if the performance of architecture is damaged by any layer then it will be skipped by regularization. This helps us to train an efficient model without worrying about vanishing gradient.

It has also been observed that it is easier to learn residual of output and input, rather than only the input. As an added advantage, our network can now learn identity function by simply setting residual as zero. And if you truly understand backpropagation and how severe the problem of vanishing gradient becomes with increasing number of layers, then you can clearly see that because of these skip connections, we can propagate larger gradients to initial layers and these layers also could learn as fast as the final layers, giving us the ability to train deeper networks.

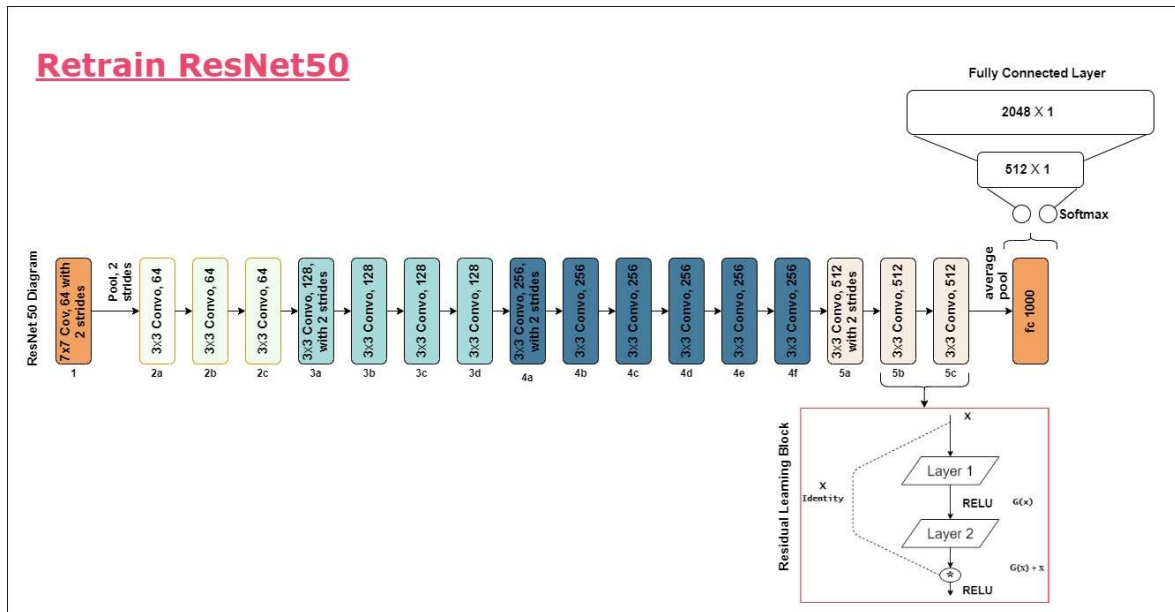


Fig. 3.7 ResNet 50 Architecture

The above is the ResNet50 model, which is being loaded and ready for an inspection. There are 48 Conv layers together with 1 Avg Pool layer and 1 Max Pool layer. Note that this network has an advantage of AdaptiveAvgPool2d layer, which allows us to input(images) of any size into the model.

Table 3.2 Distribution of ResNet 18, 34, 50

Name of the layers	O/p Size	Layers = 18	Layers = 34	Layers = 50
ConvolutionalL1	112 x 112	7 x 7, stride = 2, 64		
ConvL2	56 x 56	3 x 3 max pool layer with 2 strides		
		[3 x 3, 64] x2 [3 x 3, 64] x2	[3 x 3, 64] x3 [3 x 3, 64] x3	[1 x 1, 64] x3 [3 x 3, 64] x3 [1 x 1, 256] x3
ConvL3	28 x 28	[3 x 3, 128] x2 [3 x 3, 128] x2	[3 x 3, 128] x4 [3 x 3, 128] x4	[1 x 1, 128] x4 [3 x 3, 128] x4 [1 x 1, 512] x4
ConvL4	14 x 14	[3 x 3, 256] x2 [3 x 3, 256] x2	[3 x 3, 256] x6 [3 x 3, 256] x6	[1 x 1, 256] x6 [3 x 3, 256] x6 [1 x 1, 1024] x6
ConvL5	7 x 7	[3 x 3, 512] x2 [3 x 3, 512] x2	[3 x 3, 512] x3 [3 x 3, 512] x3	[1 x 1, 512] x3 [3 x 3, 512] x3 [1 x 1, 2048] x3
	1 x 1	avg pool, 1000 fc, softmax		
Operations required		1.8×10^9	3.6×10^9	3.8×10^9

So as shown in the table 3.3.1 the ResNet 50 architecture comprises of the following layers:

- The convolution layer consists of a kernel with size 7×7 and 64 other kernels with a stride value of 2 concluding in 1 layer. A Max Pooling layer is also considered with the stride size = 2.
- Next Conv layer kernel consists of 1×1 , 64 following this is a 64 kernel, 3×3 . And in the end we have 256 kernel, 1×1 . In total all this layers are repeated for 3 three times, concluding with 9 layers in this Conv.
- Then, we see a kernel 128 of 1×1 , after that another kernel 128 of 3×3 and in the end, a kernel 512 of 1×1 . This process is repeated for 4 times, concluding with 12 layers in this Conv.
- Following it is repeated 6 times with kernel of 1×1 , 256 and twice more kernels of 3×3 , 256 and 1×1 , 1024 which gives us a total of 18 layers.
- Then 1×1 , 512 kernel with twice of 3×3 , 512 and a 1×1 , 2048 and this was run for 3 times repeatedly resulting in 9 layers in this step.
- Average the pooling layers, and finish it with a connected layer containing fully connected nodes (1000) and in the final step one more layer of SoftMax function which concludes the network with the final 1 layer.

Here Avg, Max pooling layers and the activation function is not taken into count, so totally we end up with $1 + 9 + 12 + 18 + 9 + 1 = 50$ layers Deep Convolutional network.

3.3.7 Registration

Image registration is a method for converting datasets to a single coordinate model. Image registration plays a vital role in the area of medical imaging, biological imaging. Registration is necessary to analyse or integrate data from several medical sources. Usually, a medical technician is supposed to display several images in different directions to reduce the visual contrast between images (Haskins, Kruger, & Yan, 2020). The medical technician is often expected to manually classify points in the image that have significant signal variations as part of a sizeable anatomical structure. Medical image registration saves a lot of time for doctors and physicists. To address the shortcoming of the manual registration process, DL implementations in the image registration process have improved the productivity of the image registration process.

Next we create the class to setup the model. Inside the init constructor method, we load the ResNet50 model, setting the pretrained parameter to true to load the pretrained weights. We then freeze the starting layers of the network (except the fully connected layer) and replaced the last layer (fc) with our own as our output has only 2 classes, whereas the ResNet50 was trained to output 1000 classes (refer to the model above).

The forward method is invoked when we pass a batch of inputs to the model.

Now we define the helper functions to fit and evaluate the model.

- `evaluate`: this simply calls the validation functions we defined in the base model class above and returns the output. This function is called inside the `fit` function.
- `get_lr`: for getting the learning rate of each epoch, so that one can store them if a learning rate scheduler is used, to keep track of the changes.
- `fit`: this is the main function to fit the model. We pass in our model and dataloaders and other hyperparameters. There are some codes that were commented out. These are some other methods that I tried but they did not seem to help with performance, such as weight decay, gradient clipping and one-cycle learning rate scheduler.
- `model.eval()` set specific layers like dropout and batchnorm to evaluation mode (dropout won't drop activations, batchnorm will use running estimates instead of batch statistics) while `model.train()` will enable them.
- `torch.no_grad()` impacts the autograd engine and deactivate it. It will reduce memory usage and speed up computations but we won't be able to backprop (which we don't want in an evaluation phase).

Before we train the model, we need to ensure that the data and the model's parameters (weights and biases) are on the same device (CPU or GPU). We can reuse the `to_device` function to move the model's parameters to the right device. This is important and is a step that is usually forgotten and will cause error.

Finally, it's time to train and evaluate our model on the entire train and validation sets. Configurations like batch size, learning rate etc. need to be picked in advance

while training machine learning models, and are called hyperparameters. Picking the right hyperparameters is critical for training an accurate model within a reasonable amount of time, and is an active area of research and experimentation. Feel free to try different learning rates and see how it affects the training process.

```
Epoch [1], train_loss: 0.1568, train_acc: 0.9637, val_loss: 6.6135, val_acc: 0.2647
Epoch [2], train_loss: 0.0163, train_acc: 0.9866, val_loss: 0.0654, val_acc: 0.9806
Epoch [3], train_loss: 0.0067, train_acc: 0.9951, val_loss: 0.0643, val_acc: 0.9806
Epoch [4], train_loss: 0.0027, train_acc: 0.9963, val_loss: 0.0674, val_acc: 0.9729
Epoch [5], train_loss: 0.0037, train_acc: 0.9958, val_loss: 0.0418, val_acc: 0.9840
Epoch [6], train_loss: 0.0017, train_acc: 0.9982, val_loss: 0.0177, val_acc: 0.9970
Epoch [7], train_loss: 0.0014, train_acc: 0.9987, val_loss: 0.0201, val_acc: 0.9944
Epoch [8], train_loss: 0.0006, train_acc: 1.0000, val_loss: 0.0147, val_acc: 0.9970
Epoch [9], train_loss: 0.0053, train_acc: 0.9952, val_loss: 0.5823, val_acc: 0.8354
Epoch [10], train_loss: 0.0025, train_acc: 0.9976, val_loss: 0.0427, val_acc: 0.9901
Epoch [11], train_loss: 0.0008, train_acc: 1.0000, val_loss: 0.0185, val_acc: 0.9957
Epoch [12], train_loss: 0.0006, train_acc: 1.0000, val_loss: 0.0158, val_acc: 0.9974
Epoch [13], train_loss: 0.0013, train_acc: 0.9988, val_loss: 0.0152, val_acc: 0.9970
Epoch [14], train_loss: 0.0003, train_acc: 1.0000, val_loss: 0.0194, val_acc: 0.9970
Epoch [15], train_loss: 0.0003, train_acc: 1.0000, val_loss: 0.0144, val_acc: 0.9970
Epoch [16], train_loss: 0.0003, train_acc: 1.0000, val_loss: 0.0128, val_acc: 0.9987
Epoch [17], train_loss: 0.0028, train_acc: 0.9976, val_loss: 0.4504, val_acc: 0.8656
Epoch [18], train_loss: 0.0015, train_acc: 0.9994, val_loss: 0.0049, val_acc: 0.9987
Epoch [19], train_loss: 0.0009, train_acc: 0.9987, val_loss: 0.0124, val_acc: 0.9987
Epoch [20], train_loss: 0.0007, train_acc: 0.9994, val_loss: 0.0230, val_acc: 0.9957
```

Fig. 3.8 Training of the model

3.3.8 Deploying Web Application of the model

Now that our machine learning pipeline and model are ready we will start building a web application that can connect to them and generate predictions on new data in real-time. There are two parts of this application:

- Front-end (designed using HTML and JavaScript)
- Back-end (developed using Flask in Python)

The front-end of the web applications is built using HTML, CSS and JavaScript. Html to define the elements, CSS to design and add some splash of colors to the elements. Finally we finish it with a pinch of JS for making the page even more interactive. We have used a HTML template and a CSS style sheet to design an input form.

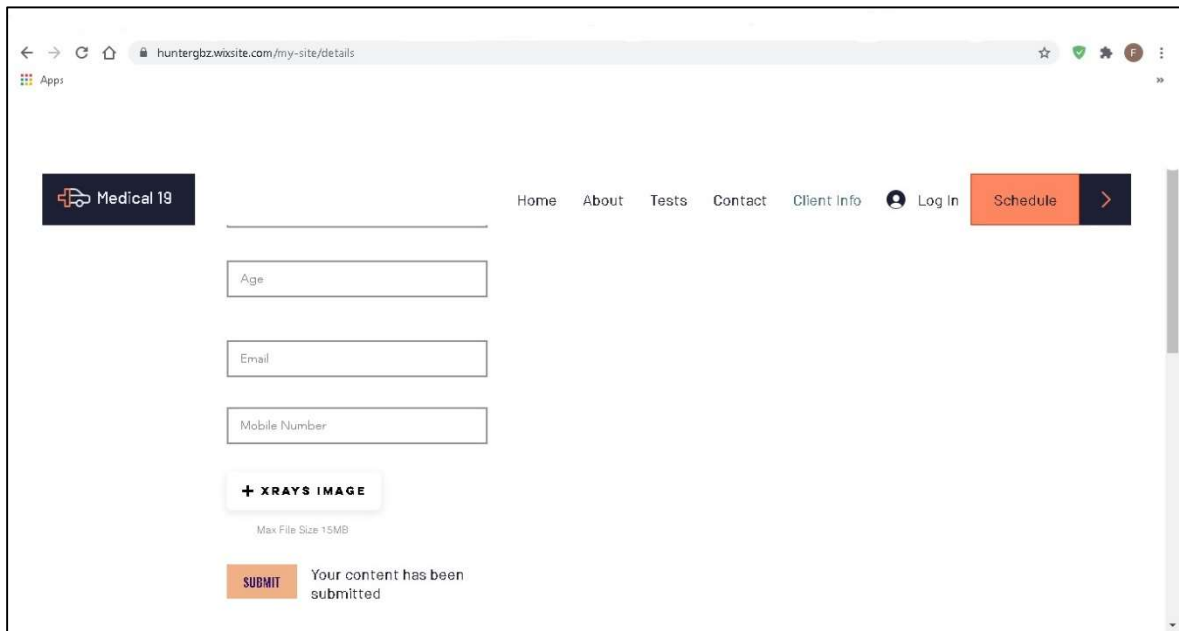


Fig. 3.9 Website Frontend Design

The back-end of a web application is developed using a Flask framework. For beginner's it is intuitive to consider Flask as a library that you can import just like any other library in Python. Now that the model is trained, the machine learning pipeline is ready, and the application is tested on our local machine, we are ready to start our deployment on Heroku. There are couple of ways to upload your application source code onto Heroku. The simplest way is to link a GitHub repository to your Heroku account. That sums up the project and have fulfilled the desired objective as intended.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 RESULTS

To understand the created DL model better I have plotted some graphs and have reached to some discussions and they are as follows:

We made plots of the accuracy and loss for the training and validation data. This gives us an idea of how our model is performing (e.g., underfitting, overfitting).

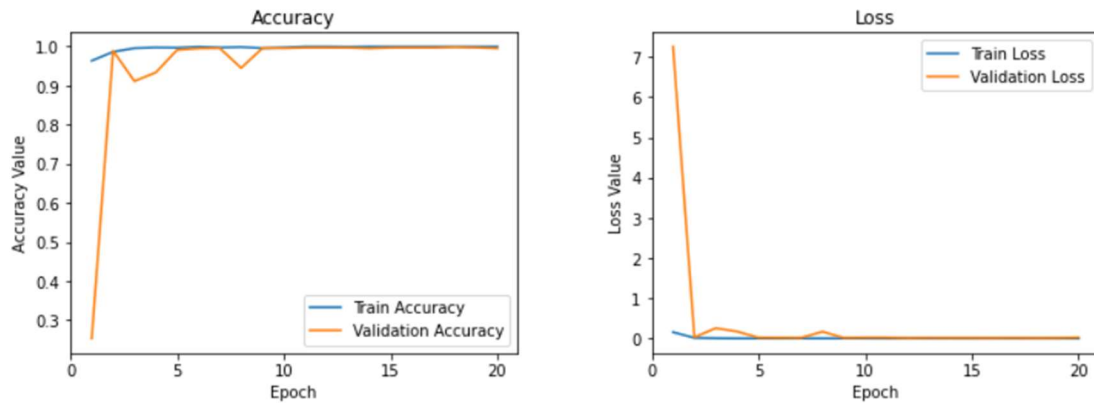


Fig. 4.1 Performance graph

It's time to test our model on the test set and see how well it performs on data that it has not seen before. This is important, as we want a model that can generalize to other datasets other than what it is trained on. A model that only does well on the training and validation dataset but not on a testing dataset is not a useful one.

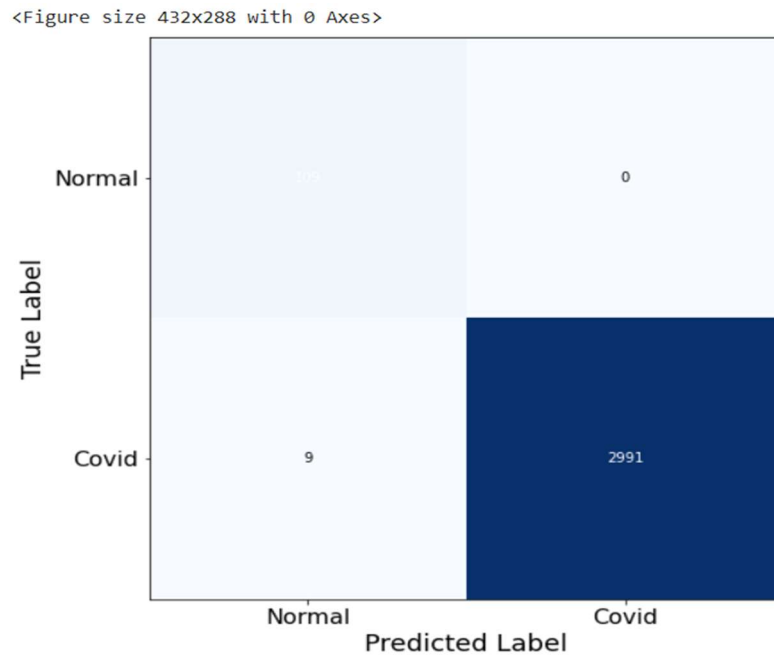


Fig. 4.2 Confusion Matrix

The following conclusion have been drawn from the confusion matrix between True label and Predicted Label. Our model is performing quite well with a F1-score of about 70-80%. Precision is lower than recall, but this is expected due to the data class imbalance. The weightings can perhaps we tuned to improve the precision. However, recall and precision is balance and one has to be sacrificed for the other.

```
Accuracy of the model is 1.00
Recall of the model is 1.00
Precision of the model is 1.00
F1 Score of the model is 1.00
```

We can conclude with this results that we have built an efficient model to analyse whether the person is COVID positive or Negative from the given X-ray image.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In conclusion, our model performed reasonably well, as it manages to predict most of the COVID affected images. This is important as in healthcare diagnosis, accurate prediction of diseases saves lives. However, false positives can also increase the cost of healthcare, as more people with diseases are diagnosed with diseases. It can also lead to panic and affect people's physical and mental well-being. Better performance can be achieved with better tuning of the model hyperparameters. This is an iterative process, with lots of trial and error. Below are some of the difficulties that I faced for this dataset:

- **Data class imbalance:** This is quite prevalent in the real world and as we should find ways to get around this problem. Usually, one way is to simply collect more data for the under sampled class. However, this is not possible for this dataset and especially for healthcare data, which is very difficult to collect and share. In this kernel, I used weighted loss to give more weights to the loss of the normal images. This weighting can be tuned as well to achieve the optimal performance. Stratified sampling is also important in this case, but Pytorch does not yet have any built-in functions.
- **Overfitting:** This dataset is prone to overfitting as seen by the train and validation plots. Therefore, I only selected the model weights that achieved the best/lowest validation loss. However, sometimes the best validation loss are achieved in the early epochs, and it performs very badly on the test set, probably due to class imbalance when batching or sampling. For that reason, I have chosen to select the best validation loss only after certain epochs.

5.2 Future Work

The accuracy of the DL model becomes questionable due to the use of insufficient data. Also, accessibility to CT and X-ray images from wider demographics is a requirement that is difficult to collect from healthcare organizations. More efficient and robust models can be used to develop DL model like SqueezeNet etc.,

REFERENCES

1. Minaee, Shervin and Kafieh, Rahele and Sonka, Milan and Yazdani, Shakib and JamalipourSoufi, Ghazaleh, "Deep-COVID: Predicting COVID-19 From Chest X-Ray Images Using Deep Transfer Learning" arXiv preprint arXiv:2004.09363, 2020
2. Ai, T., Yang, Z., Hou, H., Zhan, C., Chen, C., Lv, W., et al. (2020). Correlation of chest CT and RT-PCR testing in coronavirus disease 2019 (COVID-19) in China: A report of 1014 cases. *Radiology*, 200642.
3. Cheng, Y. (2019). Semi-supervised learning for neural machine translation. *Joint training for neural machine translation*, 25–40.
4. Alom, M. Z., Yakopcic, C., Nasrin, M. S., Taha, T. M., &Asari, V. K. (2019). Breast cancer classification from histopathological images with inception recurrent residual convolutional neural network. *Journal of Digital Imaging*, 32(4), 605–617.
5. A Krizhevsky, I Sutskever, GE Hinton, "Imagenet classification with deep convolutional neural networks", Advances in neural information processing systems, 2012.
6. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
7. Bai, H. X., Hsieh, B., Xiong, Z., Halsey, K., Choi, J. W., Tran, T. M. L., et al. (2020). Performance of radiologists in differentiating COVID-19 from viral pneumonia on chest CT. *Radiology*, 200823.
8. Apostolopoulos, I. D., &Mpesian, T. A. (2020). COVID-19: Automatic detection from X-ray images utilizing transfer learning with convolutional neural networks. *Physical and Engineering Sciences in Medicine*, 1–6.
9. J. Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size." arXiv preprint arXiv:1602.07360 (2016).
10. Qasem Abu Al-Haija; Adeola Adebajo, "Breast Cancer Diagnosis in Histopathological Images Using ResNet-50 Convolutional Neural Network" IEEE 2020.

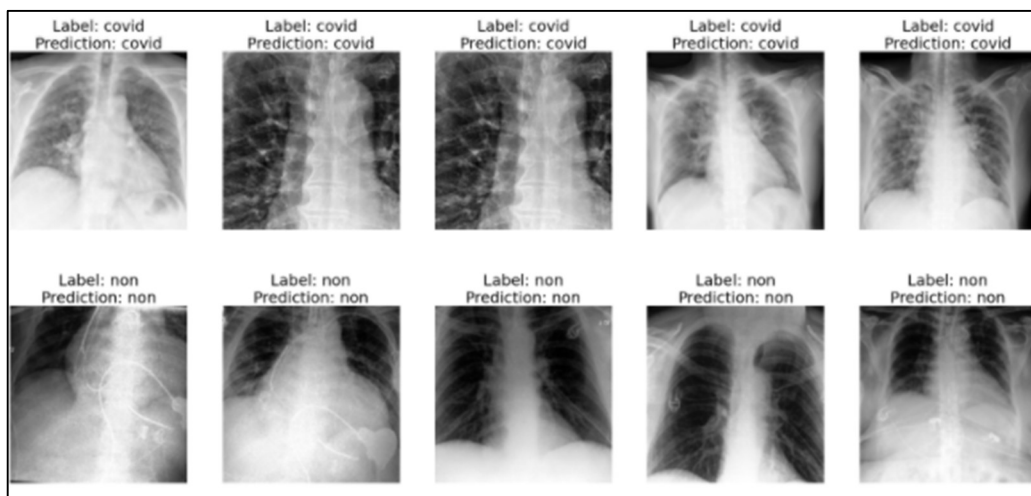
11. Joseph Paul Cohen and Paul Morrison and Lan Dao "COVID-19 image data collection" arXiv 2003.11597, 2020.
12. S. Saha, "A Comprehensive Guide to Convolutional Neural Networks-ELI5 way", *Towards Data Science A Medium publication sharing concepts ideas and codes*, 2018.
13. C. Shorten, "Introduction to ResNets", *Towards Data Science A Medium publication sharing concepts ideas and codes*, Jan 2019.
14. Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Quoc-Viet Pham b, hipa Reddy Gadekallu, Siva Rama Krishnan S, Chiranjil Lal Chowdhary, Mamoun Alazab, Md. Jalil Piran, "Deep learning and medical image processing for coronavirus (COVID-19) pandemic: A survey", Elsevier, 2020.
15. Rahib H. Abiyev and Mohammad Khaleel Sallam Ma'aitah, "Deep Convolutional Neural Networks for Chest Diseases Detection", Hindawi, Journal of Healthcare Engineering, Volume 2018, Article ID 4168538, 2020.

APPENDIX

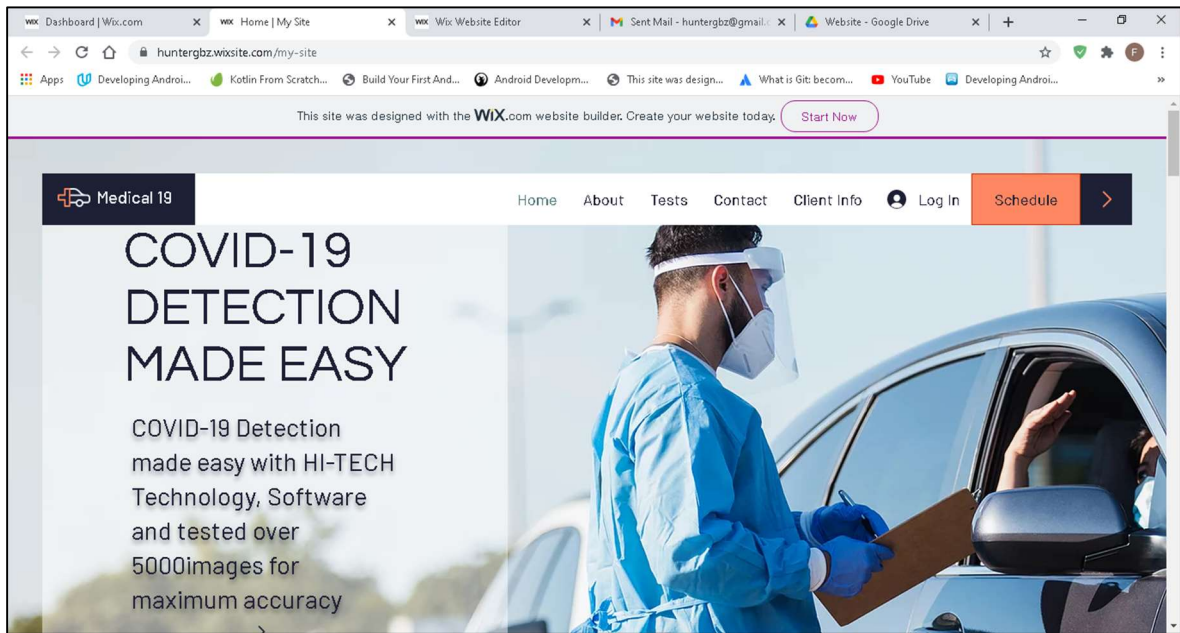
A) SCREENSHOTS :

```
ResNet(  
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)  
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
  (relu): ReLU(inplace=True)  
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)  
  (layer1): Sequential(  
    (0): Bottleneck(  
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)  
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (downsample): Sequential(  
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      )  
    )  
    (1): Bottleneck(  
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)  
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
    )  
    (2): Bottleneck(  
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)  
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
    )  
  )  
)
```

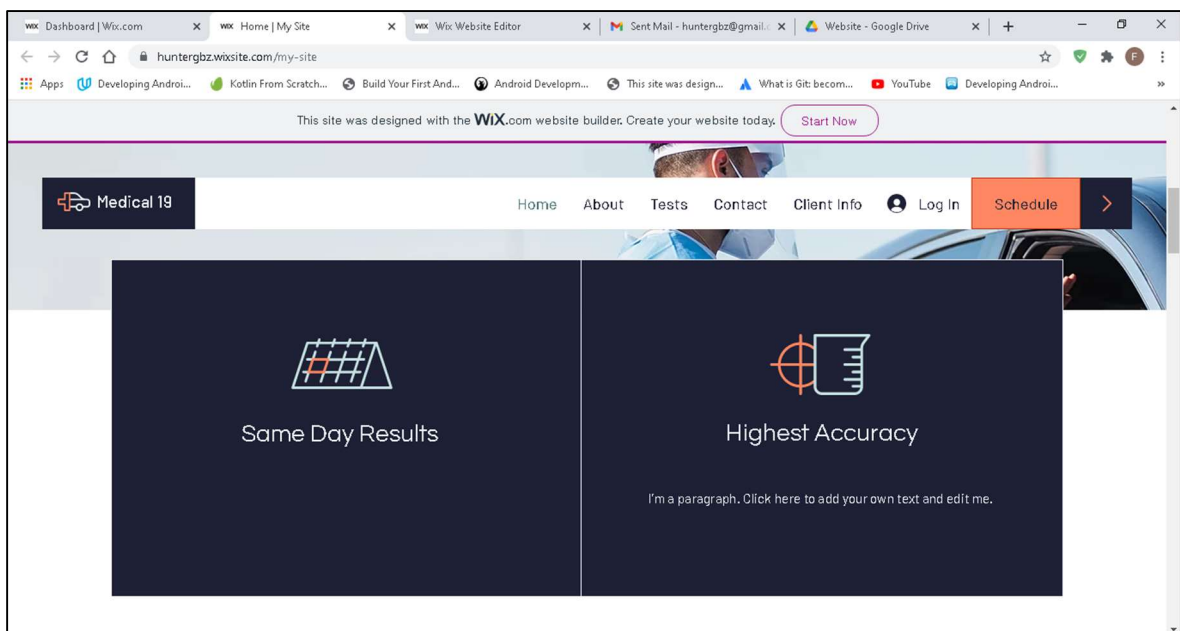
1. ResNet 50 Model



2. Testing Results



3. Website Homepage



4. Website About Page

This site was designed with the Wix.com website builder. Create your website today. [Start Now](#)

Mr.

+ X-RAYS IMAGE

SUBMIT

5. Website Form entry page

Dashboard

My Site
Role: Owner
Set Up Your Site
3 steps left

Dashboard
Booking Services
Booking Calendar
Analytics & Reports
Upgrade

Content Manager > Patient Details

< **Patient Details** LIVE SYNC LIVE TO SANDBOX + New Item

Default View 9 Items **FlinT** 9 Items + New View

4 Hidden Fields Sort Filter

	T Title	T Name	# Mobile Number	Email	X-RAYS
1	Item 1	Vino Geoffrey	919176068498	! huntergbz@gmail.com	
2	Item 1	Vino Geoffrey	919176068498	! huntergbz@gmail.com	
3	Item 1	Vino Geoffrey	919176068498	! huntergbz@gmail.com	
4	Item 1	FlinT	9875641233	! vinovio@hotmail.com	
5	Item 1	Vic	7894250	! vic@123gamil.com	
6					
7	Item 1	Abdullah	1121212	! abdullah1232222	

6. Database of the website

B) SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.image import imread
import seaborn as sns
import random
import cv2
import copy

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tt
import torchvision.models as models
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
from torch.utils.data import random_split, DataLoader

from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

import os
data_dir = 'gdrive/MyDrive/Dataset'

print(os.listdir(data_dir))
classes = os.listdir(data_dir + "/train")
print(classes)

covid_files = os.listdir(data_dir + "/train/covid")
print('No. of training examples for Covid:', len(covid_files))
print(covid_files[:5])

normal_files = os.listdir(data_dir + "/train/non")
print('No. of training examples for Normal:', len(normal_files))
print(normal_files[:5])

dataset = ImageFolder(data_dir+'train', transform=tt.Compose([tt.Resize(255),tt.CenterCrop(224),
                                                             tt.RandomHorizontalFlip(),
                                                             tt.RandomRotation(10),tt.RandomGrayscale(),
```

```

tt.RandomAffine(translate=(0.05,0.05), degrees=0),tt.ToTensor()))
img, label = dataset[0]
print(img.shape, label)
img

print(dataset.classes)

# number of images for each class

train_samplesize = pd.DataFrame.from_dict(
    {'Normal': [len([os.path.join(data_dir+'/train/non', filename)
                        for filename in os.listdir(data_dir+'/train/non')])],
      'Covid': [len([os.path.join(data_dir+'/train/covid', filename)
                        for filename in os.listdir(data_dir+'/train/covid')])])}]

sns.barplot(data=train_samplesize).set_title('Training Set Data Inbalance', fontsize
=20)
plt.show()

import matplotlib.pyplot as plt
%matplotlib inline

def show_example(img, label):
    print('Label: ', dataset.classes[label], "("+str(label)+")")
    plt.imshow(img.permute(1, 2, 0))

show_example(*dataset[4])

# Function for plotting samples
def plot_samples(samples):
    fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(30,8))
    for i in range(len(samples)):
        image = cv2.cvtColor(imread(samples[i]), cv2.COLOR_BGR2RGB)
        ax[i//5][i%5].imshow(image)
        if i<5:
            ax[i//5][i%5].set_title("Normal", fontsize=20)
        else:
            ax[i//5][i%5].set_title("Covid", fontsize=20)
        ax[i//5][i%5].axis('off')
## Plot training samples
rand_samples = random.sample([os.path.join(data_dir+'/train/non', filename)
                                for filename in os.listdir(data_dir+'/train/non')], 5) + \
    random.sample([os.path.join(data_dir+'/train/covid', filename)

```

```

        for filename in os.listdir(data_dir+'train/covid'), 5)

plot_samples(rand_samples)
plt.suptitle('Training Set Samples', fontsize=30)
plt.show()

# set random seed so we get the same sampling every time for reproducibility

random_seed = 2020
torch.manual_seed(random_seed);

train_size = round(len(dataset)*0.7) # 70%
val_size = len(dataset) - train_size # 30%

train_ds, val_ds = random_split(dataset, [train_size, val_size])
len(train_ds), len(val_ds)

batch_size=128

train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)
def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 12))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images[:60], nrow=10).permute(1, 2, 0))
        break

show_batch(train_dl)

def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

device = get_default_device()
device

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]

```

```

return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device) # yield will stop here, perform other steps, and then resumes to the next loop/batch

    def __len__(self):
        """Number of batches"""
        return len(self.dl)

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds)), preds

def F1_score(outputs, labels):
    _, preds = torch.max(outputs, dim=1)

    # precision, recall, and F1
    cm = confusion_matrix(labels, preds)
    tn, fp, fn, tp = cm.ravel()
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1 = 2*((precision*recall)/(precision+recall))

    return precision, recall, f1, preds

class CovidModelBase(nn.Module):

    # this is for loading the batch of train image and outputting its loss, accuracy
    # & predictions
    def training_step(self, batch, weight):
        images, labels = batch
        out = self(images) # generate predictions
        loss = F.cross_entropy(out, labels, weight=weight) # weighted compute loss

```

```

    acc,preds = accuracy(out, labels)                # calculate accuracy

    return {'train_loss': loss, 'train_acc':acc}

# this is for computing the train average loss and acc for each epoch
def train_epoch_end(self, outputs):
    batch_losses = [x['train_loss'] for x in outputs]    # get all the batches loss
    epoch_loss = torch.stack(batch_losses).mean()        # combine losses
    batch_accs = [x['train_acc'] for x in outputs]       # get all the batches acc
    epoch_acc = torch.stack(batch_accs).mean()           # combine accuracies

    return {'train_loss': epoch_loss.item(), 'train_acc': epoch_acc.item()}

# this is for loading the batch of val/test image and outputting its loss, accuracy,
# predictions & labels
def validation_step(self, batch):
    images,labels = batch
    out = self(images)                                # generate predictions
    loss = F.cross_entropy(out, labels)                 # compute loss
    acc,preds = accuracy(out, labels)                  # calculate acc & get preds

    return {'val_loss': loss.detach(), 'val_acc':acc.detach(),
            'preds':preds.detach(), 'labels':labels.detach()}
# detach extracts only the needed number, or other numbers will crowd memory

# this is for computing the validation average loss and acc for each epoch
def validation_epoch_end(self, outputs):
    batch_losses = [x['val_loss'] for x in outputs]     # get all the batches loss
    epoch_loss = torch.stack(batch_losses).mean()        # combine losses
    batch_accs = [x['val_acc'] for x in outputs]         # get all the batches acc
    epoch_acc = torch.stack(batch_accs).mean()           # combine accuracies

    return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

# this is for printing out the results after each epoch
def epoch_end(self, epoch, train_result, val_result):
    print('Epoch [{}], train_loss: {:.4f}, train_acc: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}'.
          format(epoch+1, train_result['train_loss'], train_result['train_acc'],
                 val_result['val_loss'], val_result['val_acc']))

# this is for using on the test set, it outputs the average loss and acc,
# and outputs the predictions
def test_prediction(self, outputs):

```

```

    batch_losses = [x['val_loss'] for x in outputs]
    epoch_loss = torch.stack(batch_losses).mean()          # combine losses
    batch_accs = [x['val_acc'] for x in outputs]
    epoch_acc = torch.stack(batch_accs).mean()            # combine accuracies
    # combine predictions
    batch_preds = [pred for x in outputs for pred in x['preds']].tolist()
    # combine labels
    batch_labels = [lab for x in outputs for lab in x['labels']].tolist()

    return {'test_loss': epoch_loss.item(), 'test_acc': epoch_acc.item(),
            'test_preds': batch_preds, 'test_labels': batch_labels}

resnet50 = models.resnet50(pretrained=True)
#resnet50

class CovidResnet(CovidModelBase):
    def __init__(self):
        super().__init__()
        # Use a pretrained model
        self.network = models.resnet50(pretrained=True)
        # Freeze training for all layers before classifier
        for param in self.network.fc.parameters():
            param.requires_grad = False
        num_features = self.network.fc.in_features # get number of in features of last
layer
        self.network.fc = nn.Linear(num_features, 2) # replace model classifier

    def forward(self, xb):
        return self.network(xb)

@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def fit(epochs, lr, model, train_loader, val_loader, weight,
        weight_decay=0, grad_clip=None, opt_func=torch.optim.SGD):

```

```

torch.cuda.empty_cache() # release all the GPU memory cache
history = {}

optimizer = opt_func(model.parameters(), lr)

best_loss = 1 # initialize best loss, which will be replaced with lower better loss
for epoch in range(epochs):

    # Training Phase
    model.train()
    train_outputs = []
    lrs = []

    for batch in train_loader:
        outputs = model.training_step(batch, weight)
        loss = outputs['train_loss'] # get the loss
        train_outputs.append(outputs)
        # get the train average loss and acc for each epoch
        train_results = model.train_epoch_end(train_outputs)
        loss.backward()

        # Gradient clipping
        if grad_clip:
            nn.utils.clip_grad_value_(model.parameters(), grad_clip)

        optimizer.step() # update weights
        optimizer.zero_grad() # reset gradients

    # Record & update learning rate
    # lrs.append(get_lr(optimizer))
    # sched.step()
    # Validation phase
    val_results = evaluate(model, val_loader)

    # Save best loss
    if val_results['val_loss'] < best_loss and epoch + 1 > 15:
        best_loss = min(best_loss, val_results['val_loss'])
        best_model_wts = copy.deepcopy(model.state_dict())
        #torch.save(model.state_dict(), 'best_model.pt')

    # print results
    model.epoch_end(epoch, train_results, val_results)

```

```

# save results to dictionary
to_add = {'train_loss': train_results['train_loss'],
          'train_acc': train_results['train_acc'],
          'val_loss': val_results['val_loss'],
          'val_acc': val_results['val_acc'], 'lrs':lrs}
# update performance dictionary
for key,val in to_add.items():
    if key in history:
        history[key].append(val)
    else:
        history[key] = [val]

model.load_state_dict(best_model_wts)                # load best model

return history, optimizer, best_loss

train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)

model = to_device(CovidResnet(), device)

epochs = 20
lr = 0.0001
grad_clip = None
weight_decay = 1e-4
opt_func = torch.optim.Adam
# weighted loss for data class imbalance
weight = torch.FloatTensor([357/(2000+357), 2000/(2000+357)]).to(device)

history, optimizer, best_loss = fit(epochs, lr, model, train_dl, val_dl, weight,
                                   grad_clip=grad_clip,
                                   weight_decay=weight_decay,
                                   opt_func=opt_func)
# this is for loading the model from a previously saved one

def load_checkpoint(filepath):
    checkpoint = torch.load(filepath)
    model = checkpoint['model']
    model.load_state_dict(checkpoint['state_dict'])
    for parameter in model.parameters():
        parameter.requires_grad = False

model.eval()

```

```

return model

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
t = f.suptitle('Performance', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

epoch_list = list(range(1, epochs+1))
ax1.plot(epoch_list, history['train_acc'], label='Train Accuracy')
ax1.plot(epoch_list, history['val_acc'], label='Validation Accuracy')
ax1.set_xticks(np.arange(0, epochs+1, 5))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history['train_loss'], label='Train Loss')
ax2.plot(epoch_list, history['val_loss'], label='Validation Loss')
ax2.set_xticks(np.arange(0, epochs+1, 5))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")

@torch.no_grad()
def test_predict(model, test_loader):
    model.eval()
    # perform testing for each batch
    outputs = [model.validation_step(batch) for batch in test_loader]
    results = model.test_prediction(outputs)
    print('test_loss: {:.4f}, test_acc: {:.4f}'
          .format(results['test_loss'], results['test_acc']))

    return results['test_preds'], results['test_labels']

# Plot confusion matrix
cm = confusion_matrix(labels, preds)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), cmap=plt.cm.Blues)
plt.xticks(range(2), ['Normal', 'Covid'], fontsize=16)
plt.yticks(range(2), ['Normal', 'Covid'], fontsize=16)
plt.xlabel('Predicted Label', fontsize=18)
plt.ylabel('True Label', fontsize=18)
plt.show()

```

```

# Compute Performance Metrics
tn, fp, fn, tp = cm.ravel()

accuracy = (np.array(preds) == np.array(labels)).sum() / len(preds)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = 2*((precision*recall)/(precision+recall))

print("Accuracy of the model is {:.2f}".format(accuracy))
print("Recall of the model is {:.2f}".format(recall))
print("Precision of the model is {:.2f}".format(precision))
print("F1 Score of the model is {:.2f}".format(f1))

# select 5 normal and 5 COVID images indices
idxs = torch.tensor(np.append(np.arange(start=0, stop=5, step=1),
                              np.arange(start=500, stop=505, step=1)))

fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(30,14))

for c,i in enumerate(idxs):
    img_tensor, label = test_dataset[i]
    ax[c//5][c%5].imshow(img_tensor[0,:,:], cmap='gray')
    ax[c//5][c%5].set_title('Label: {}\nPrediction: {}'.format(test_dataset.classes[label],
                                                                test_dataset.classes[preds[i]]),
                           fontsize=25)
    ax[c//5][c%5].axis('off')

```

C) PAPER TO PUBLISH

A STUDY ON CORONAVIRUS PANDEMIC ANALYSIS USING DEEP LEARNING TECHNIQUES

Victor Abilash¹, Vino Jeffrey², Sathya Bama Krishna. R³

^{1,2}Student, ³Assistant Professor, Dept of CSE, Sathyabama Institute of Science and Technology, Jeppiaar Nagar, Sholinganallur, Chennai, Tamil Nadu 600119, India
vicshal65@gmail.com, sathyabamar.cse@sathyabama.ac.in, huntergbz@gmail.com

Abstract:

The Corona Virus pandemic has caused a major outburst in many countries around the world, impacting on health and the lifestyle of millions. The major task fighting against Corona Virus pandemic is the ability to identify the infected cases early enough, and to put them in a separate care and essential treatment. The fastest way to detect this disease is from radiology images and radiography. Studies shows specific deformity in the chest radiology images of the infected patients by Corona Virus pandemic. In this project we use the application of deep learning models to expose whether the person is Corona Virus Positive or Negative from their chest radiograph images. This study paper is used to build an efficient model based on various deep learning techniques like convolutional neural network (CNN) to give an accurate prediction whether the person is COVID positive or negative.

Keywords: CNN, Deep Learning, COVID-19, X-Ray, ResNet.

1. Introduction

The Corona Virus pandemic and the attempt to contain the virus has resulted in creating a health hazard dilemma worldwide affecting all the sides of human life. In the starting stage of the pandemic, the number of people affected was minimal and it did not cause any menace of such an extensive capacity as the majority cases were sorted out instantaneously. With time Coronavirus positive cases were increasing in an alarming rate and later was announced as an global pandemic by World Health Organization (WHO) alerting that it could affect millions of life and especially the one with weaker healthcare system. The Coronavirus is considered to be fatal basically due to two reasons- it has no vaccines and is distinctive. Secondly, it is a contagious disease.

The statistical graph of Corona Virus pandemic with 2.42 million lives giving away the battle as of January 29th, 2021, shows a great many reasons for a lot of concern with 109 million affected globally and counting, by the pandemic. USA being one of the dominant country in healthcare development stand with the highest number of

Corona Virus victims followed by India, Brazil, United Kingdom, Russia and the affected countries list grows around the globe.

In a desperate attempt to fight and find a solution for the Corona Virus pandemic, researches and technicians across the world have initiated scientific studies in every possible directions. Deep Learning incorporated with medical image processing approach is also used to conclude with an unravelling solution. Currently countries like India and Russia have found the vaccines to partially control the disease on experimental basis. But research and development cannot be put on hold as we need a solution for this disease. In [1][7][11] the paper proposes methods to collect the dataset for Coronavirus and how to clearly extract the main attributes necessary to build a model such as ResNet 18 etc., Information fusion plays an significant role in the development of vaccine. The knowledge available should be studied at all ways to find the ultimate goal which is the vaccine. The technical definition fusion, mentions that “the process of combining and associating information from one or multiple sources to provide useful information for the detection, identification, and characterization of a particular entity” which is essential to keep battling this horrendous disease. In DL and ML employment, the accessibility of larger and high quality DL dataset has a key role in the precision of the outcome. Information fusion integrates multiple datasets and assist us to use them in the DL (Deep Learning) trained models to accomplish conclusions to different types of problems and to learn more about to get enhanced accuracy in predictions.

2. Overview of the Methods

Data Language and Artificial Neural Networks (ANN) have acquired enormous momentum in the contemporary scientific research agenda. They got the ability to help us learn from the surrounding and aid us to understand the virus and its effect in the human system in a better way. Deep Learning and Neural Network methods have been broadly used in copious fields such as image processing, image recognition, pattern recognition, IOT driven appliances, object recognition, smart homes etc., because of their ability to learn and adapt to various types of data across numerous domain.

2.1 Deep Learning

DL techniques are classified as supervised, unsupervised, and semi-supervised learning. In advanced development, the training model is edified with a known put in and out dataset as we know where the virus attack can be found i.e. Lungs, the model has been trained to look at a particular place where various studies have proved to be the right place to look at to find if the virus has attacked an individual or not. All the known value comprises of a put in and the required value called as the supervisory signal. This method predicts the labels of the desired output with the help of existing labels. Classification methods widely uses variety of supervised learning to recognize a fraud in the signature, to the scenarios of identifying face of a person, to training to describing an image in its own words etc.,

ANN are used for feature extraction and processing in most of the DL techniques. A summarized representation is formed in Feedback technique for learning the mechanism where it updates each layer with the input data. In Deep Learning the term ‘deep’ refers to the no. of layers that it requires to learn the underlying pattern/trend. Credit Assignment Path is vastly used in this networks for transformation process. In NN, the measure of Credit Assignment Path is calculated by the no. of hidden DL layers, and the no. of output DL layers.

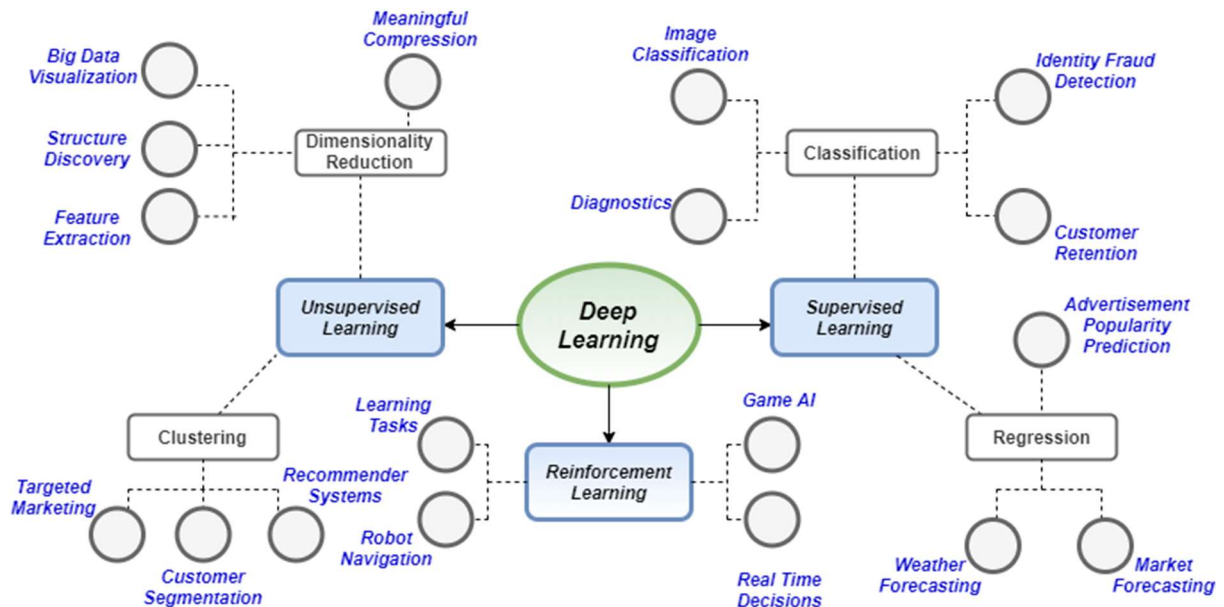


Fig.2.1.1 Several Techniques and Applications of DL.

There are various other DL Techniques follows such as Convolutional Neural Network(CNN), Generative Adversarial Network(GAN) and Recurrent Neural Network(RNN) etc., which have proven to be reliable and efficient irrespective of the field/data we use them in. In [1] the author have discussed various ways of analysis of Coronavirus including number of deep learning techniques such as SqueezeNet, ResNet 18, DenseNet and many more DL models with varying accuracy but all proving to be better than the other in one way or another. Among which SqueezeNet have been proven to have a better precision curve of 0.897.

2.2 Supervised Learning

Supervised learning, is also called as supervised ML, it comes under machine learning and artificial intelligence. Labelled datasets are used to train the models/algorithms to predict or classify the output precisely. Firstly, in supervised ML, the model is fed with an input dataset consisting of numerous data enough to fit the model. The weights of the model is automatically adjusted using reinforcement learning, hence the model can be fitted suitably. The supervised ML helps us to solve a variety of real-world problems easily, such as

- Classifying spam emails from the regular emails.

- The model is used to identify or describe images, for instance its widely used in AIs and driver plate detection etc,. The model created using training dataset continuous to learn from the new dataset fed to it and updates itself automatically.
- The is model is also trained with molecules and information to identify drugs etc,.

There are two types of Supervised Learning:

- I. Classification
- II. Regression

2.2.1 Classification

An algorithm accurately assigns the testing dataset into a specific category of the classification. It automatically identifies the trend/pattern within the given dataset and tries to conclude some results on how the entities should be labelled. Some of the classification models/algorithms are as follows:

- Support Vector Machines (SVM)
- Random forest
- K-nearest neighbour and
- Decision Tree.

2.2.2 Regression

Regression is applicable in places to know the relationship between independent and dependent dataset. It is mainly used to make estimation, such as for sales revenue, operation efficiency etc,. Some of the common regression algorithms used are as follows:

- Linear Regression
- Polynomial Regression and
- Logistic Regression.

2.3 Unsupervised Learning

Unsupervised learning is frequently learnt with supervised ML together. In this learning technique it uses unlabelled dataset. It explores patterns that help us solve for association problems or clustering of the data. It comes very handy when the experts are not sure about the common properties or trends present in the subject of matter within the dataset. Common Un-supervised learning models/algorithms are hierarchical, K-means and Gaussian mixture. Some of the real-life application of Unsupervised learning includes

- Data Mining (K- means clustering)

- Customer service personalization
- Principle Component Analysis.

There are two types of Unsupervised Learning:

- I. Clustering
- II. Dimensionality Reduction

2.3.1 Clustering

The process of collecting similar dataset together is called “Clustering”. Clustering technique is used to find the similarities in the data and group all the similar data together from a defined dataset. Collecting similar entities together help profile the properties/trends of different data. That is, in other words, clustering will help us to identify the pattern underlying a particular phenomenon. There are numerous applications of grouping unlabelled data, for instance, we can identify different groups of customers and the type of revenue each product gather around different types of customers, which in turn can be used to maximize the revenue. Another example includes grouping of similar documents together which belongs to the same group/segment etc,. This method is also used to reduce the dimensionality of the data in the dataset, when we are dealing with an abundant no. of variables.

Some of the common clustering algorithms are as follows:

- K-mean Clustering
- Hierarchical Clustering
- Hidden Markov Model (NLP, Pattern Recognition)

2.3.2 Dimensionality Reduction

Dimensionality reduction is the process of filtering the pertinent information from the bulk or getting the unnecessary information out of the context. For instance, this can be explained with a simple problem that we face day to day, that is to find whether the received email is a spam or not. Identifying the spam email involves extracting number of features such as whether the email as a particular template or not, or whether the email as a generic title etc,. However, several of these features may overlap with each other. In another case, such as a classification problem of both rainfall and humidity can be put in as one feature as both are closely correlated to one another in a high degree. This way, we can reduce the number of features for the analysis in this type of problems.

Few of the popular algorithms used in dimension reduction are as follows:

- Principle Component Analysis (PCA)
- Linear and Manifold learning methods
- T-distributed Stochastic Neighbour Embedding.

2.4 Reinforcement ML

The training of ML model to make a sequence of conclusion/decision is called Reinforcement machine learning. The model achieve the goal in an undetermined way, likely in a complex surrounding. The AI features a game-like situation in this type of ML. The model uses the trial and error method to arise with a solution to the given problem. The model aka AI gets rewards or penalties for the measures it takes for the given problem, hence the machine basically does what the programmer wants it to do. The main objective of the AI is to get more rewards than penalties which ultimately results in an efficient model. Although the programmer sets a bunch of rules for the rewards/policy system, it is important that the given instruction doesn't give away any clue or hint to the model to how to solve the problem. The AI model has to choose its way up, as how to maximize the rewards starting with a number of trails and ending the task with a superior skills and sophisticated techniques with superhuman knowledge set. Some of the real-life applications of Reinforcement learning are as follows

- Application self-driving cars
- Reinforcement learning in trading and finance
- Natural Language Processing (NLP)

The commonly used reinforcement algorithm includes:

- Associative reinforcement learning
- Safe reinforcement learning
- Deep reinforcement learning
- Inverse reinforcement learning

3. DL Networks

DL networks are basically mathematical models that are used to copy the actions of the human brains as it is meant to solve the problems using unstructured data, these models are created in form of neural network that consists of neurons. The neural network is divided into three major layers that are input layer(first layer of neural network), hidden layer (all the middle layer of neural network) and the output layer(last layer of the neural network). In [9] clearly discusses about the accuracy that we can achieve in different type of types of DL Networks.

3.1 Convolutional Neural Network

A **Convolutional Neural Network (CNN)** is a DL algorithm which takes input images and assign some parameters (learnable biases and weights) of different features of the image and should be able to differentiate from one image to another. The primary advantage of choosing a CNN network is that, it requires less pre-processing in contrast to other typical classification models/algorithms. The method filters are engineered enough and CNN has the ability to absorb all about this from the filters/characteristics from the given data set.

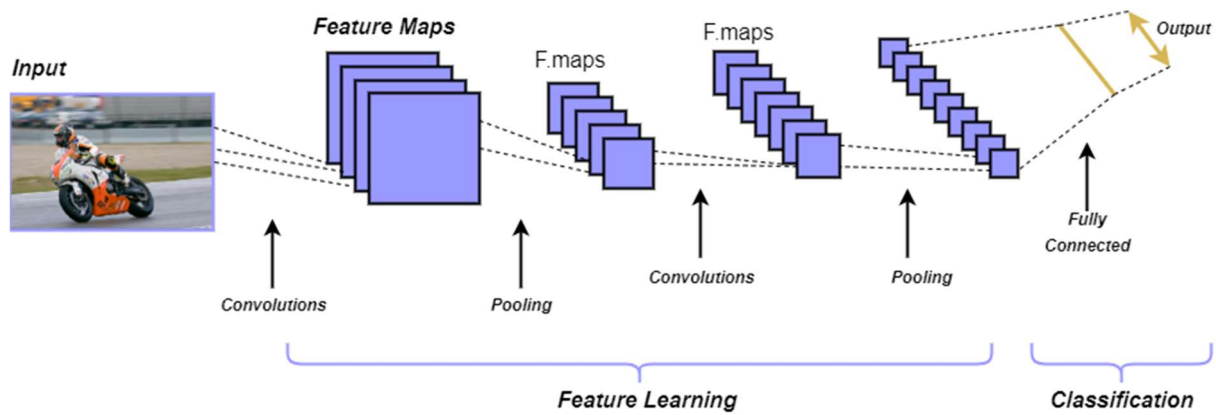


Fig.3.1.1. Architecture of CNN

CNN has a various types of architecture, which plays a vital role in building models/algorithms which gives AI the power in the upcoming events.[12] clearly explains about various types of Neural networks and how it can be used in daily lifestyle to provide a more efficient solution. Some of the famous CNN models are listed below:

1. GoogleLeNet
2. ResNet
3. VGGNet
4. LeNet
5. AlexNet

3.2 Residual Network (ResNet)

The earliest CNN-based architecture (AlexNet), won the ImageNet competition(2012), later based on the award winning architecture, the subsequent award winning model started to add more DL neural networks in their model to reduce the error rate of the architecture. But when we expand the no. of layers, this results in an unsophisticated problem of vanishing gradient. This in point of fact makes the vanishing gradient to become 0 or too big to contain. Therefore, when we expand the no. of layers, the training of the model also increases, causing to increase the test error rate.

In 2015, Microsoft research team introduced a whole new architecture called ResNet. It was introduced in the hope of solving the concept of vanishing gradient also known as exploding gradient. Using a dynamic approach called skip connections in the network. This approach basically skips some connections with a few layers and uninterruptedly is connected to the output layer. The main objective of ResNet is that the layers absorb and learn from the underlying mapping, this network can fit the residual mapping. Hence, here $H(x)$ is the initial mapping, let the network mapping fit be as follows,

$$G(x) := H(x) - x$$

which results in $H(x) := G(x) + x$.

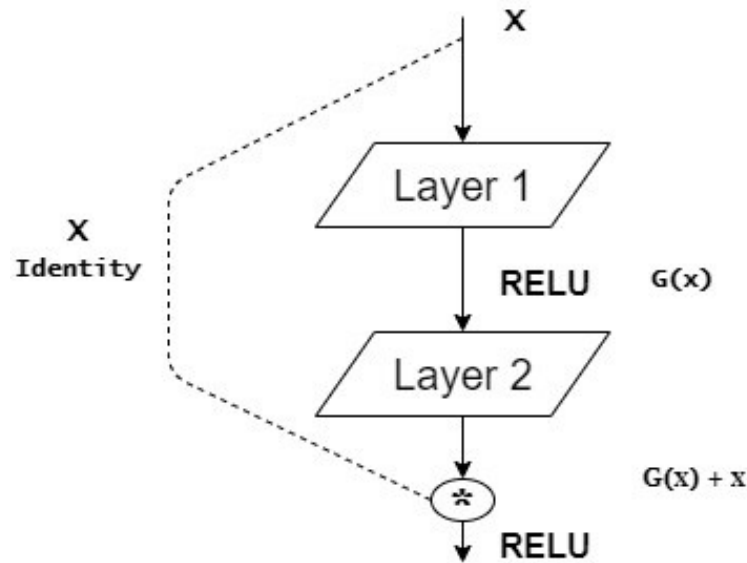


Fig.3.2.1 Residual Network Structure

The advantage of skipping the connection is that, if the performance of architecture is damaged by any layer then it will be skipped by regularization. This helps us to train an efficient model without worrying about vanishing gradient.

3.3. ResNet50

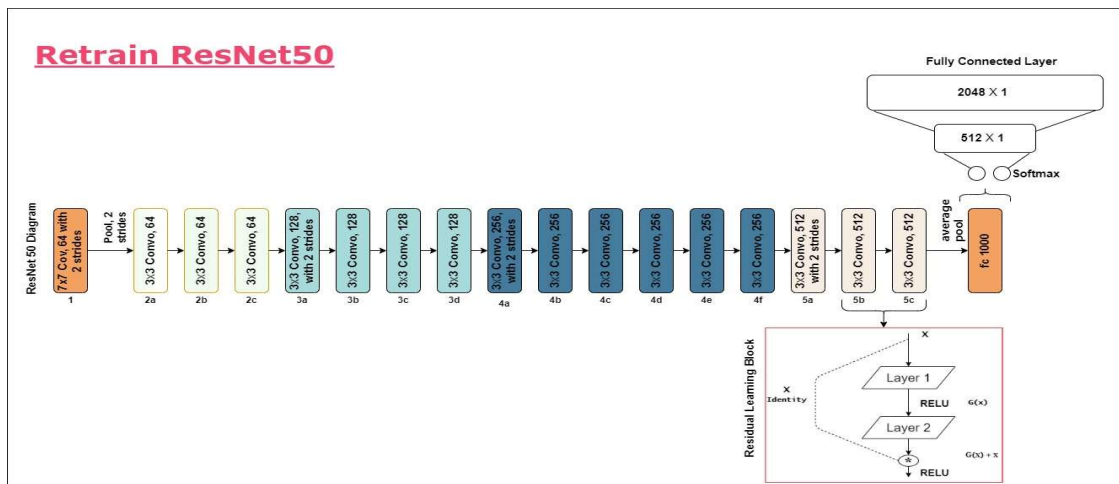


Fig.3.3.1. Architecture of ResNet50

The above is the ResNet50 model, which is being loaded and ready for an inspection. There are 48 Conv layers together with 1 Avg Pool layer and 1 Max Pool layer. Note that this network has an advantage of AdaptiveAvgPool2d layer, which allows us to input(images) of any size into the model. In [10] the author performs how to analyse Breast Cancer which contributes heavily in serious situations and help us find the disease much sooner than later.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
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

Table.3.3.1. Architecture of ResNet 18, 34, 50.

So as shown in the table 3.3.1 the ResNet 50 architecture comprises of the following layers:

- The convolution layer consists of a kernel with size 7 * 7 and 64 other kernels with a stride value of 2 concluding in **1 layer**. A Max Pooling layer is also considered with the stride size = 2.
- Next Conv layer kernel consists of 1 * 1, 64 following this is a 64 kernel, 3 * 3. And in the end we have 256 kernel, 1 * 1. In total all this layers are repeated for 3 three times, concluding with **9 layers** in this Conv.
- Then, we see a kernel 128 of 1*1, after that another kernel 128 of 3 * 3 and in the end, a kernel 512 of 1 * 1. This process is repeated for 4 times, concluding with **12 layers** in this Conv.
- Following it is repeated 6 times with kernel of 1 * 1, 256 and twice more kernels of 3 * 3, 256 and 1 * 1, 1024 which gives us a total of **18 layers**.
- Then 1 * 1, 512 kernel with twice of 3 * 3, 512 and a 1 * 1, 2048 and this was run for 3 times repeatedly resulting in **9 layers** in this step.
- Average the pooling layers, and finish it with a connected layer containing fully connected nodes (1000) and in the final step one more layer of SoftMax function which concludes the network with the final **1 layer**.

Here Avg, Max pooling layers and the activation function is not taken into count, so totally we end up with $1 + 9 + 12 + 18 + 9 + 1 = 50$ **layers** Deep Convolutional network.

4. Discussion and Conclusion

The manual detection of cases of Corona Virus attacked or non-Corona virus pneumonia affected cases is a commanding and time-consuming process as no other possible way to detect the disease, as these cases are aggressively rising. The implementation of DL plays a crucial role in prediction of various diseases with the bulk datasets that can be collected from private as well as public organizations like WHO etc, is essential. DL implementation appeal focuses on medical imaging, which seems to be a hopeful solution. Deep Learning implementation is used to process and analyse medical image data to help doctors and radiologists increase the precision of the diagnosis. Deep Learning from medical imaging has the inherent capability to even identify possible targets for an suitable Corona Virus vaccines, which will help the weaker people get vaccinated faster. Various studies on Corona virus pandemic have been conducted to accentuate the importance of automated Corona virus analysis using DL models using medical image datasets. The use of neural networks for the detection of coronavirus pandemic can be improved in the future by expanding the dataset with more Coronavirus positive images. More accurate models can be build to increase the accuracy of the predictions like SqueezeNet.

References

16. Minaee, Shervin and Kafieh, Rahele and Sonka, Milan and Yazdani, Shakib and JamalipourSoufi, Ghazaleh: "Deep-COVID: Predicting COVID-19 From Chest X-Ray Images Using Deep Transfer Learning"; arXiv preprint arXiv, (2020): 2004.09363.
17. Ai, T., Yang, Z., Hou, H., Zhan, C., Chen, C., Lv, W., et al; "Correlation of chest CT and RT-PCR testing in coronavirus disease 2019(COVID-19) in China: A report of 1014 cases"; *Radiology*, (2019): 200642 .
18. Cheng, Y; "Semi-supervised learning for neural machine translation. Joint training for neural machine translation"; (2019): 25–40.
19. Alom, M. Z., Yakopcic, C., Nasrin, M. S., Taha, T. M., &Asari, V. K; "Breast cancer classification from histopathological images with inception recurrent residual convolutional neural network"; *Journal of Digital Imaging*, 32(4), (2019): 605–617.
20. A Krizhevsky, I Sutskever, GE Hinton; "Imagenet classification with deep convolutional neural networks"; *Advances in neural information processing systems*, (2012).
21. He, Kaiming, et al; "Deep residual learning for image recognition."; *Proceedings of the IEEE conference on computer vision and pattern recognition*. (2016).
22. Bai, H. X., Hsieh, B., Xiong, Z., Halsey, K., Choi, J. W., Tran, T. M. L., et al; "Performance of radiologists in differentiating COVID-19 from viral pneumonia on chest CT"; *Radiology*, (2019): 200823.
23. Apostolopoulos, I. D., and Mpesiana, T. A.; "COVID-19: Automatic detection from X-ray images utilizing transfer learning with convolutional neural networks."; *Physical and Engineering Sciences in Medicine*, (2020): 1–6.

24. Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer; "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size."; arXiv preprint arXiv (2016): 1602.07360.
25. Qasem Abu Al-Haija; Adeola Adebanjo; "Breast Cancer Diagnosis in Histopathological Images Using ResNet-50 Convolutional Neural Network"; IEEE (2020).
26. Joseph Paul Cohen and Paul Morrison and Lan Dao; "COVID-19 image data collection"; arXiv (2020):2003.11597.
27. S. Saha; "A Comprehensive Guide to Convolutional Neural Networks-ELI5 way"; *Towards Data Science A Medium publication sharing concepts ideas and codes*, (2018).
28. C. Shorten; "Introduction to ResNets"; *Towards Data Science A Medium publication sharing concepts ideas and codes*, (2019).