

**PNEUMONIA DETECTOR: Convolutional Neural Network Architecture designed for
the Detection of Pneumonia using Chest-X Rays**

Submitted By:

Anjishnu Roy
(18-300-4-02-0553)

Rashmiman Nandi
(18-300-4-02-0540)

Shuvam Aich
(18-300-4-02-588)

**B.SC. COMPUTER SCIENCE
SEMESTER VI**

**Supervised By:
PROF. SONALI SEN**

**St. Xavier's College (Autonomous), Kolkata
Department of Computer Science
30, Mother Teresa Sarani, Kolkata-700016**

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled **PNEUMONIA DETECTOR: Convolutional Neural Network Architecture designed for the Detection of Pneumonia using Chest-X Rays** submitted to the Department of Computer Science, St. Xavier's College(Autonomous), Kolkata, in partial fulfillment of the requirement for the award of the degree of Bachelor of Science (B.Sc.) is an entirely original work carried out by **Mr Shuvam Aich** (Roll Number: 18-300-4-02-0588 and Registration Number: A01-1112-0882-18), **Mr Rashmiman Nandi** (Roll Number: 18-300-4-02-0540 and Registration Number: A01-1112-0855-18) and **Mr Anjishnu Roy** (Roll Number: 18-300-4-02-0553 and Registration Number: A01-1112-0863-18) under my guidance. The matter embodied in this project is authentic and is genuine work done by the aforementioned students and has not been submitted whether to this college or to any other institute for the fulfilment of the requirement of any course of study.

Shuvam Aich

Shuvam Aich,
Student
Date: 27/04/2021

Rashmiman Nandi

Rashmiman Nandi,
Student
Date: 27/04/2021

Anjishnu Roy

Anjishnu Roy,
Student
Date: 27/04/2021

Prof. Sonali Sen,
Project Guide
Date: 27/04/2021

We hereby recommend that the project report prepared by Shuvam Aich, Rashmiman Nandi and Anjishnu Roy with the title "**PNEUMONIA DETECTOR: Convolutional Neural Network Architecture designed for the Detection of Pneumonia using Chest-X Rays**" be accepted in partial fulfilment of the requirements for the degree of B.Sc. in Computer Science at St. Xavier's College (Autonomous), Kolkata.

Prof. Romit S Beed
Head of the Department
Department of Computer Science
Date:

External Examiner
Date:

Abstract

Pneumonia accounts for around 16% of all deaths of children under five years worldwide. It is a common disease caused by different microbial species such as bacteria, virus, and fungi. Some of the symptoms of pneumonia include the shortness of breath, fever, cough, chest pain, etc. Moreover, the people at risk of pneumonia are elderly people (above 65 years), children (below the age of 5 years), and people with other complications such as HIV/AIDS, diabetes, chronic respiratory diseases, cardiovascular diseases, cancer, hepatic disease, etc. The pneumonia detection is usually performed through examination of chest X-Ray radiograph by highly-trained specialists. This process is tedious and often leads to a disagreement between radiologists. Deep neural network models have conventionally been designed, and experiments were performed upon them by human experts in a continuing trial-and-error method. This process demands enormous time, know-how, and resources. Computer-aided diagnosis systems showed the potential for improving diagnostic accuracy. In this work, we develop the computational approach for pneumonia regions detection based on single-shot detectors, deep convolutional neural networks, augmentations and multi-task learning.

ACKNOWLEDGEMENT

In performing the assignment, my team members and I had to take the succour and guidance of some very helpful people, who deserve our greatest gratitude. The completion of this assignment gives us much pleasure. We would like to extend our heartfelt gratitude to Prof. Sonali Sen, Project Mentor, for her valuable time in giving us good advice for the assignment and correcting us wherever possible throughout numerous consultations. The professors of the Department of Computer Science have gratified us indefinitely for their constant support and unconditional help.

Many people, especially our classmates and our friends have given us valuable suggestions on this proposal which gave us an inspiration to improve our assignment. We thank all the people for their help directly or indirectly in completing our assignment. And lastly, we would like to extend our gratitude towards our families who have stood by us all throughout the journey.

TABLE OF CONTENTS

1. INTRODUCTION

- 1.1 Context
- 1.2 Deep Learning
- 1.3 Improving Deep Neural Network: Cases of Overfitting
- 1.4 Regularisation
- 1.5 Optimization
- 1.6 Learning Rate
- 1.7 Dropout

2. DATASET

- 2.1 Description
- 2.2 Display
- 2.3 Challenges
- 2.4 Technological Advances
- 2.5 Ethics and Legalities

3. IMAGE AUGMENTATION

4. THE CNN ARCHITECTURE

- 4.1 Implementation
 - 4.1.1 Multilayer Perceptron vs Convolutional Neural Network
 - 4.1.2 The use of CNN for our model
 - 4.1.3 Use of Filters
 - 4.1.4 Creating the CNN of our model:
- 4.2 Code Details and Efficiency

5. BINARY CLASSIFICATION PROBLEM

6. INPUT MODULE

7. RESULTS AND DISCUSSION

- 7.1 Report
- 7.2 Visualisation

8. SYSTEM REQUIREMENTS

9. CONCLUSION

- 9.1 Basic Conclusion
- 9.2 Limitations
- 9.3 Future Scope

10. REFERENCES

1. INTRODUCTION

1.1 Context

The risk of pneumonia is immense for many, especially in developing nations where billions face energy poverty and rely on polluting forms of energy. The WHO estimates that over 4 million premature deaths occur annually from household air pollution-related diseases including pneumonia. COVID-19 is an extremely contagious disease caused by severe acute respiratory syndrome coronavirus 2 (SAR-CoV-2) which is the recent disease that is caused by one of the family members of Coronaviridae family. Coronaviruses invade the lung's alveoli (an organ responsible for exchange of O₂ and CO₂), thereby causing pneumonia. The pneumonia complicating recent coronavirus disease 2019 (COVID-19) is a life-threatening condition claiming thousands of lives in 2020. Pneumonia caused by COVID-19 is of huge global concern, with confirmed cases in 221 countries across five continents taking the number of cases up to 139,69,704 and the number of deaths to 3,000,000 at the time of writing this paper.

The term 'medical imaging technology' features a broad definition and encompasses any technique that helps medical professionals view the interior of the body or areas that are not visible to the naked eye. Visualisation of these structures can aid in the diagnosis of disease, treatment planning, treatment execution – such as thorough image-guided intervention, and monitoring and surveillance. The most common types of medical imaging would include X-Rays, ultrasound, CT or CAT scan, MRI, etc.

Medical imaging technology has progressed over years. From helping in rendering 3D images to artificial intelligence, it is integral to disease diagnosis and management.

With the context of medical imaging, AI can be trained to spot anomalies in human tissue – thereby aiding both in the diagnosis of diseases and monitoring their treatment. AI can not only sift through huge datasets of images and patient information and superhuman speeds that help in expediting workflows but can also be trained to detect anomalies that are too small to be discerned with the naked eye thus improving the diagnostic accuracy. X-rays are commonly used to evaluate lungs with the fat and muscle appearing as shades of grey while air in the lungs show up as black.

Initially CAD systems required a high manual development of selection of the main characteristics of the images and their interpretation, but now we are living the take-off of algorithms based on Deep Learning and more specifically based on convolutional networks (CNN) able to learn and extract characteristics and represent very complex nonlinear functions by themselves without human intervention based solely on input data after a supervised training process. This ability to learn by itself from neural networks opens up promising prospects for their application in the analysis and interpretation of radiographic images.

Pneumonia is one of the most common infections of the lung which causes inflammation in one or both lungs. While interpreting the X-ray, the radiologist usually looks for white spots in the lungs that identify an infection. It also helps determine any complications related to pneumonia such as abscesses or pleural effusions (fluid surrounding the lungs). When interpreting an X-ray by a professional, his capacity and experience are key, but other factors such as fatigue, haste, subjectivity, discrepancies regarding group consensus opinion and a long etcetera also have an impact. These factors result in significant variability among radiologists in detecting a case of pneumonia by radiography.

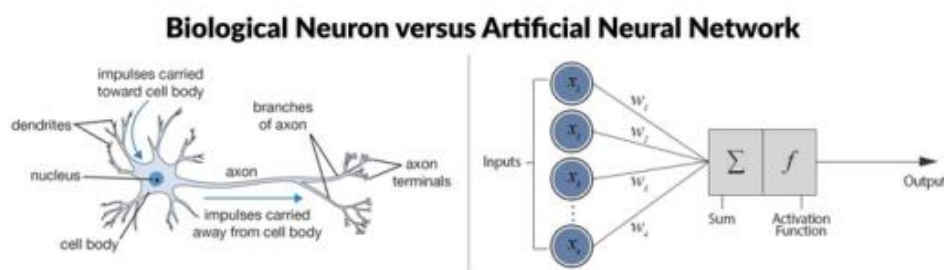
Various studies and works attest how Deep Learning algorithms have reached a performance equivalent or superior to humans in various tasks and applications. Statistical results obtained demonstrate that pre-trained CNN models employed along with supervised classifier algorithms can be very beneficial in analyzing chest X-ray images, specifically to detect Pneumonia. Deep learning approaches outperformed conventional machine learning methods in many computer vision and medical imaging tasks, including detection, classification and segmentation.

We are proposing a deep learning-based model that can quickly prognose the presence of Pneumonia. The main objective of this study is focused on building a classification model that will bring down the false negatives (people with Pneumonia but predicted otherwise) to a substantially low number along a model with high evaluation metrics (precision, accuracy, sensitivity and specificity) to determine in a precise way, from x-rays, the appearance of signs or anomalies capable of representing a case of pneumonia and to determine the location and framing of this anomaly thus providing a complete diagnostic tool for cases of pneumonia. As this is a prognosis model, we can accommodate false positives (people not having Pneumonia but predicted otherwise) to a small extent. Our approach uses deep convolutional neural networks (CNNs) and augmentations. The algorithm automatically locates lung opacities on chest radiographs and demonstrates its performance. The source code is available at <https://github.com/ShuvamAich/ChestXRay-Analysis>.

1.2 Deep Learning

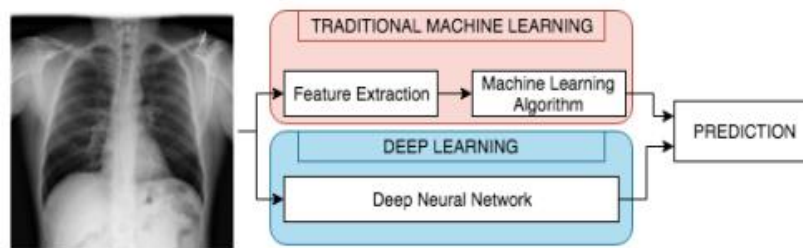
Deep Learning is a branch of Machine Learning. The main and common feature is their ability to automatically learn representative features and functions from input information, which is called "representation learning" or "feature learning".

Deep Learning carries out the training and with it the learning through multiple nodes or neurons distributed in multiple layers, simulating the functioning of the human nervous system. These nodes or neurons are organized in layers and through the interaction with other neurons distributed in different layers allows to obtain more complex and hierarchical representations. Each layer essentially acts on the characteristic constructions of the layers prior to it generating a higher level feature construction.



Supervised algorithm: They work with labeled data and the model is trained to minimize the cost function by comparing the prediction obtained by the model with the real value. Within this type we find: the feedforward neural network (FFNN) that we have commented previously, the Convolutional Neural Networks (CNN) used mainly in computer vision and Long Short Term Memory (LSTM) for applications based on time series data. In Deep Learning it is not necessary to define a rule of prediction or selection of characteristics, but the model learns them by itself. The optimization algorithm used is called gradient descent and will seek to optimize

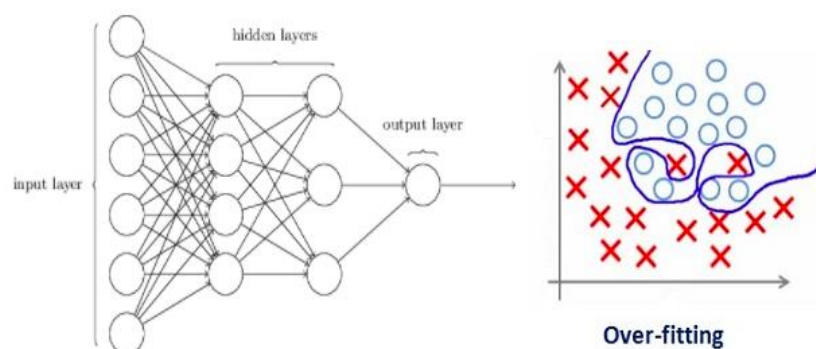
the weights of the nodes until finding a local minimum of a function, i.e., minimize the cost function.



Throughout this project, the only models used will be supervised algorithms using mainly the CNN structure that we will explain in more detail.

1.3 Improving Deep Neural Network: Cases of Overfitting

The model has been overtrained and the nodes have ended up "memorizing" the input data. Showing a high variance, i.e. a significant difference between the error of the train set and the test set. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize. To conclude, if our model is correctly trained we must pay attention to the error obtained in both the train set and with the test set.



We suspect that our neural network is overfitting our data, that is we have a high variance problem. To solve this problem, we have two choices, either to train a bigger neural network architecture and get more data or to try out regularisation. The main cost of training a larger neural network is just computational time, so long as we are regularising it. On the other hand, getting more data pretty much reduces our variance without hurting our bias much.

1.4 Regularisation

To cope with an increase in variance when training as a result of overfitting we have several techniques to deal with it.

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models.

Therefore, it will also reduce overfitting to quite an extent. However, this regularization term differs in L1 and L2.

In L2, we have:

$$\text{Cost function} = \text{Loss} + \frac{\lambda}{2m} * \sum ||w||^2$$

Here, **lambda** is the regularization parameter. It is the hyperparameter whose value is optimized for better results. L2 regularization is also known as *weight decay* as it forces the weights to decay towards zero (but not exactly zero).

In L1, we have:

$$\text{Cost function} = \text{Loss} + \frac{\lambda}{2m} * \sum ||w||$$

In this, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it.

In *keras*, we can directly apply regularization to any layer using the regularizers:

```
from keras import regularizers
model.add(Dense(64, input_dim=64, kernel_regularizer=regularizers.l2(0.01))
```

1.5 Optimization

Choosing the correct optimization configuration of our neural network will allow us to reduce the difference between the prediction obtained and the correct value.

Gradient descent optimizer: The function of the optimizer is to update the weights of the nodes in order to minimize the error of the cost function. At the time of selecting the optimizer we find several options: SGD, Adagrad, RMSprop or Adam. We pay special attention to the Adam optimizer which combines the advantages of two optimization methods that are AdaGrad and RMSProp, obtaining a robust optimizer that adapts to a large number of problems.

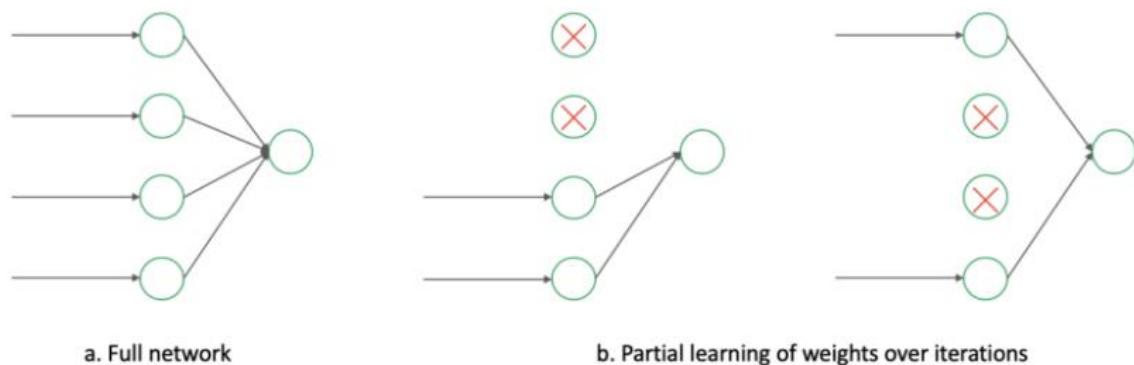
1.6 Learning rate

It is one of the hyperparameters that most affects the training process. A learning rate too small gives rise to a very slow convergence, while a learning rate that is too high cause divergence. In addition, the optimal learning rate is not a constant value throughout the entire training process. It is important to start with a sufficiently high learning rate and reduce it as the model converges to global optimum.

1.7 Dropout

Let's say we have a neural network that's overfitting. With dropout, we go through each of the layers of the neural networks and set some probability of eliminating a node in a neural network.

Let's say for each of these layers, we are, going to for each node, toss a coin and have a 0.5 chance of keeping each node and a 0.5 chance of removing each node. So after the coin tosses, we will decide to eliminate those nodes then what we do is remove all the incoming-outgoing things from the node. Thus we end up with a smaller neural network. Because we are training a much smaller network on each example, it gives us a sense why we end up being able to regularise the network. Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

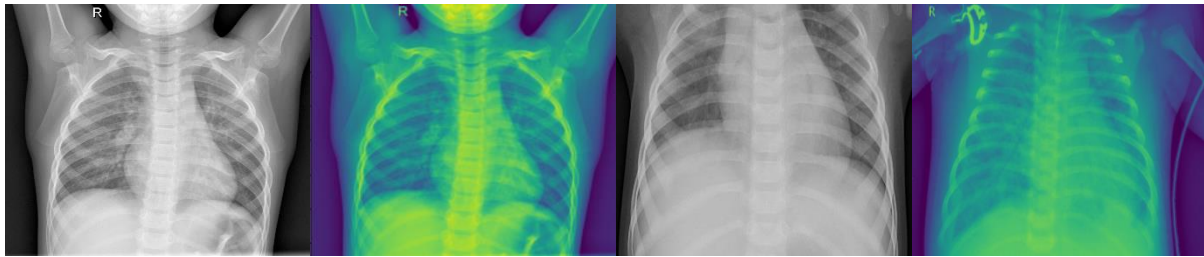


2. DATASET

2.1 Description

We have collected our dataset from [Chest X-Ray Images \(Pneumonia\) | Kaggle](#). Our dataset consists of a total of 5856 frontal-view of Chest X-Ray images which has been splitted approximately in a 90-10% ratio. The original dataset consists of three main folders (i.e., training, testing, and validation folders) and two subfolders containing pneumonia (P) and normal (N) chest X-ray images, respectively. Each image is labelled with one of two different classes from the associated radiological reports: "Normal" and "Pneumonia / Not Normal".

Name of Sets	Total Number of Images	Pneumonia(P) <i>Target: 1</i>	Normal(N) <i>Target: 0</i>
Training Set	5216	3875	1341
Test Set	624	390	234
Validation Set	16	8	8
Total	5865	3673	1583



NORMAL

PNEUMONIA

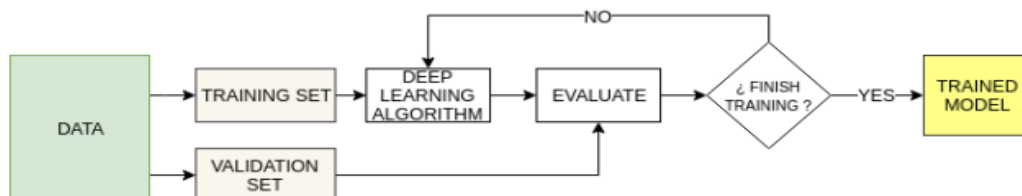
The lungs are a part of the respiratory system, and are full of air. When someone has pneumonia, the air in the lungs is replaced by other material, like fluids, bacteria, immune system cells, etc. The lung opacities refers to the areas that preferentially attenuate the x-ray beam and therefore appear more opaque on CXR than they should, indicating that the lung tissue in that area is probably not healthy.

The "Normal" class contains data of healthy patients without any pathologies found. The "Pneumonia" class has images with the presence of fuzzy clouds of white in the lungs, associated with pneumonia.

A Pandas dataframe is like a one dimensional array containing all our data. Our target is to create a dataframe, after fetching our training set with the help of the 'os- operating system interface' by Python, with the following class indices:

- Normal : 0
- Pneumonia : 1

The dataset is well-balanced with the distribution of classes as shown in the above table.



PYTHON CODE FOR DATAFRAME CREATION:

```

import os
import pandas as pd
filenames = os.listdir("/content/drive/MyDrive/chest_xray/train/NORMAL")
categories = []
for filename in filenames:
    category = filename.split('-')[0]
    if category == 'IM':
        categories.append(1)
    else:
        categories.append(0)

df1= pd.DataFrame({
    'filename': filenames,
    'category': categories})
  
```

```

import os
import pandas as pd
filenames = os.listdir("/content/drive/MyDrive/chest_xray/train/PNEUMONIA")
categories = []
for filename in filenames:
    category = filename.split('1')[0]
    if category == 'person':
        categories.append(0)
    else:
        categories.append(1)

df2= pd.DataFrame({
    'filename': filenames,
    'category': categories
})

train_generator.class_indices
{'NORMAL': 0, 'PNEUMONIA': 1}

```

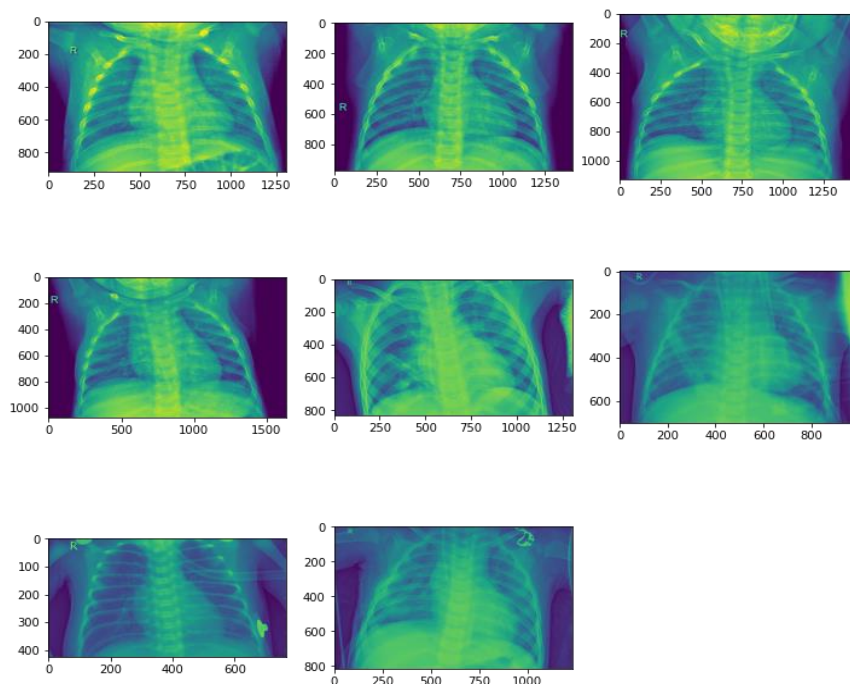
2.2 Dataset Display

Matplotlib is a comprehensive library which has been used to create static, animated, and interactive visualizations in Python. We have plotted our dataframe with the help of matplotlib and we have also created graphs to observe the performance of our model. The python code to import the library is as follows:

```

%matplotlib inline
import matplotlib.pyplot as pltimport matplotlib.image as mpimg#The image
module supports basic image loading, rescaling and display operations.

```



2.3 Challenges

Several studies collect a series of challenges for a successful implementation of Deep Learning for medical imaging tasks and act as a barrier that delays this process.

Dataset :

- The data is not complete homogeneous, it is obtained from different measuring devices or sensors with their corresponding calibration.
- The lack of a large training dataset with precise annotations. This circumstance is accentuated in several pathologies less common.
- Despite a drive to digitize medical information and implement PACS-like systems. There is no universal and standardized consensus of such systems and terminologies, which can lead to workflow incompatibilities.
- Dataset noisy and class imbalance. A small number of samples are available for some pathologies compared to other pathologies or cases without pathologies. This may result in an imbalanced dataset
- The complexity involved in interpreting medical imaging requires expert and experienced staff to create the ground-truth data, with the cost of expert staff time involved.

2.4 Technological Advances

- Need for high computing resources: Years ago this factor limited the implementation of Deep Learning systems even more. Today, advances in GPU hardware with increasing power and computational downtime techniques have partially overcome that limitation.
- Difficulty to train: Medical imaging is characterized by its complexity in the interpretation and in certain cases data limitation, an overtraining seeking to minimize the error can cause cases of overfitting of the system, losing generalization to new data. For this reason, optimisation and regularisation techniques have been developed and improved to minimise the effect of overtraining.

2.5 Ethics and Legalities

- An implementation of Deep Learning can be seen as a black box where the complex relationships and interpretations deduced from input data are opaque producing a huge barrier of understanding these complex models by clinicians.
- Assignment of responsibilities in the event of a patient diagnostic error produced by the Deep Learning model.
- The creation and use of a dataset involves legal difficulties. It is necessary to completely anonymize the information and in the case of images this process can be difficult.

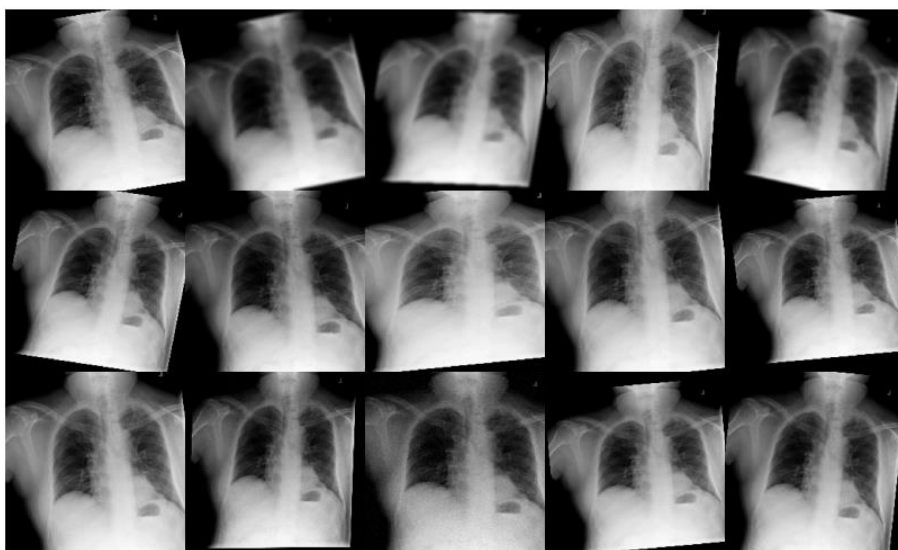
- A good reception and acceptance of such technology among medical professionals isnecessary.

3. IMAGE AUGMENTATION

A general rule when we are training any model is that the bigger the training dataset the better performance we will achieve. But sometimes the dataset is limited and it is not possible to obtain new samples with their corresponding annotations. In order to increase the size of the training data set we have the possibility to apply certain transformations to the training data such as moving the image, zooming, applying some kind of distortion or noise... in this way we increase the number of training samples.

Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc. An augmented image generator can be easily created using ImageDataGeneratorAPI in Keras.

The primary goal of using Convolutional Neural Network in most of the image classification tasks is to reduce the computational complexity of the model which is likely to increase if the input are images. So we have performed many image augmentation methods to increase the size and quality of the images. The original 3-channel images were resized into 150×150 pixels to reduce the heavy computation and for faster processing. The ‘rescale’ option helps in reducing the size or magnification of the image (rescale=1./255) and we have randomly rotated the images during training by 40 degrees. The image rotations had a measurable effect on the results, as the rotation of the bounding boxes around corners modifies the original annotated regions significant. We tested the same model with usual rotation, custom rotation and no rotation at all. The custom rotation improved our results, but the heavy augmentations without any rotation gave the best metrics on the validation. We have also added a width shift (horizontal translation) and height shift(vertical translation) of 0.2%. A shear range and a zoom range of 0.2 were also added. Finally the images were flipped horizontally.



The example of a patient chest X-ray scan with heavy augmentations and rotations.

PYTHON CODE FOR IMAGE AUGMENTATION:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# All images will be rescaled by 1./255
train_data = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
test_data = ImageDataGenerator(rescale=1./255)
train_generator = train_data.flow_from_directory(
    "/content/drive/MyDrive/chest_xray/train",
    target_size=(150, 150), # All images will be resized to 150x150
    batch_size=10,
    class_mode='binary')
validation_generator = test_data.flow_from_directory(
    "/content/drive/MyDrive/chest_xray/test",
    target_size=(150, 150),
    batch_size=10,
    class_mode='binary')
test_generator = test_data.flow_from_directory(
    "/content/drive/MyDrive/chest_xray/val",
    target_size=(150, 150),
    batch_size=10,
    class_mode='binary')
Found 5223 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
```

4. THE CNN ARCHITECTURE

4.1 Implementation

The term Deep Learning refers to Artificial Neural Networks with multiple layers. This particular type of network has become popular in recent Machine Learning literature because of its ability to handle very large amounts of data. Use of Deep Neural Networks have often shown very large improvement in the accuracy of a model, over classical methods, especially in pattern recognition. These biologically inspired computational models are to far exceed the performance of previous forms of artificial intelligence in common machine learning tasks.

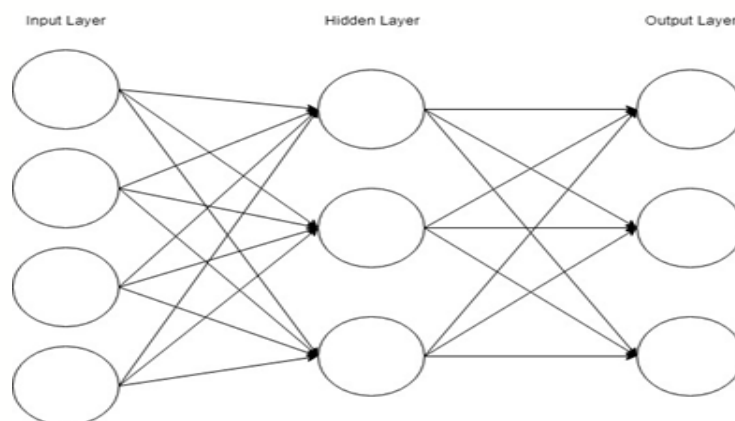
One of the most popular types of deep neural networks is Convolutional Neural Network, and we have successfully implemented this network in our model. It derives its name from convolution between matrices, a mathematical operation that can take place between matrices of any dimensions.

A Convolutional Neural Network is composed of many different types of layers, some of them being convolutional layer, non-linearity layer, pooling layer and fully connected layer. Except the pooling and the non-linearity layer, all layers take parameters. These type of networks are heavily influenced by the working of the human brain. In general, ANNs are composed of a high number of interconnected computational nodes called “neurons”, which work collectively to learn patterns from the input to produce effective and accurate outputs.

We used CNN in our model because CNNs have been known to have excellent performance in Machine Learning problems which deal with image data, for example the largest image classification data, the ImageNet dataset.

4.1.1 Multilayer Perceptron vs Convolutional Neural Network:

Traditional way to handle Image processing involved the use of “Multilayer Perceptron”. The Perceptron is made up of two fully connected layers: an input layer and an output layer. MLPs have the same input and output layers, but as shown below, they may have multiple hidden layers in between.



MLPs are the foundation of all neural networks, and when used to solve classification and regression problems, they have greatly increased the computing power of computers. Because of the multilayer perceptron, computers are no longer limited by XOR cases and can learn rich and complex models.

However, there are some problems with perceptrons, which are part of traditional neural networks. Although these are modelled on the human brain and form the basis of the CNNs and other ANNs that are in place today, these networks by themselves are inefficient when it comes to handling high dimensional data, i.e., images.

MLPs use one perceptron for each input(i.e., pixel of an image, multiplied by 3 if the image is RGB). For large images, the number of weights quickly becomes unmanageable. There are approximately 150,000 weights to train for a 224 x 224 pixel image with three colour channels! As a result, training can be difficult, and overfitting can occur.

Another common issue is that MLPs react differently to the same input (images) when shifted; they are not translation invariant. For example, if a cat appears in the top left corner of one image and the bottom right corner of another, the MLP will try to correct itself by assuming that a cat will always appear in this section of the image. Thirdly, spatial information is lost when the image is flattened into a MLP.

4.1.2 The use of CNN for our model:

CNN is a form of deep learning model for processing data with a grid pattern, such as images, that is inspired by the organisation of animal visual cortex and designed to learn spatial hierarchies of features, from low- to high-level patterns, automatically and adaptively.

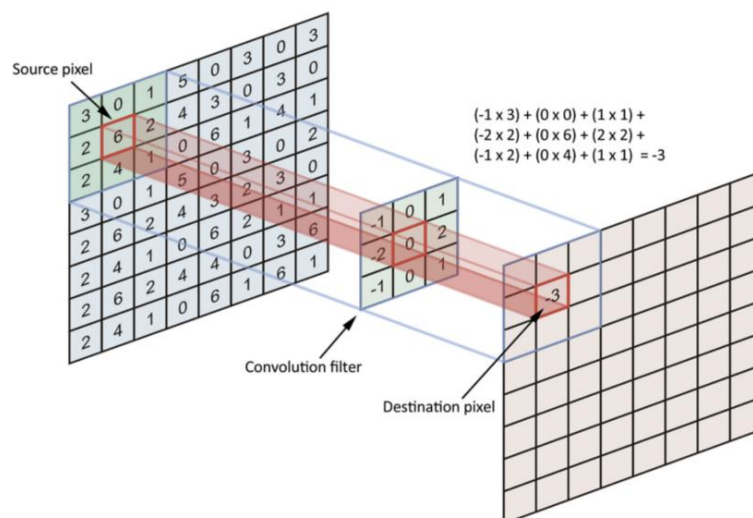
CNN is a mathematical construct typically composed of 3 layers: convolution, pooling and fully connected layers. Convolution and Pooling layers are responsible for feature extraction, whereas the fully connected layer is used to map the extracted features into a final output.

Since a feature can appear anywhere in a digital image, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers, and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image location, CNNs are highly efficient for image processing. Extracted features will become hierarchically and increasingly more complex as one layer feeds its output into the next layer. Training is the method of optimising parameters like kernels in order to reduce the difference between outputs and ground truth labels using optimization algorithms like backpropagation and gradient descent, among others.

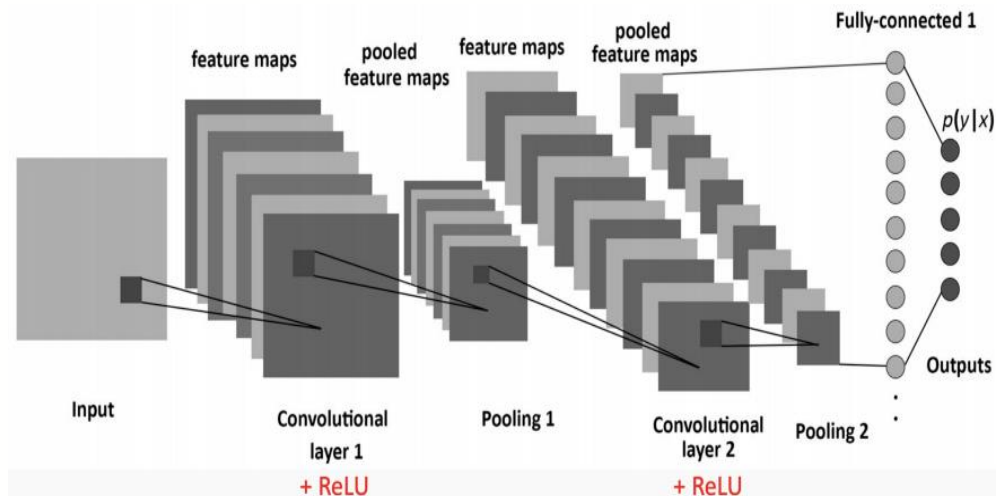
4.1.3 Use of Filters:

We determine the effect of a nearby pixel using a “filter”. A filter is a matrix of size 3x3 or 5x5, and this filter moves over the image from left to right and top to bottom. For each point on the image, a value is calculated based on the filter using a convolutional operation.

In our model, filters have been so applied such that the features required to detect pneumonia in lungs can be extracted and our model can learn accordingly. So, our filter would help us understand how strongly the features appear in our input images, including how many times and the locations where these features appear. This reduces the number of parameters that our model has to learn when compared to traditional machine learning approaches, and this also means when the location of the extracted features changes, our model does not get confused and starts giving wrong results. The extraction of features can be visualised as given in the diagram:



These filters are initialised randomly, and are updated themselves as the training progresses. No two filters can have all same values, unless the number of filters chosen for feature extraction is extremely large.



As per the diagram given above, a feature map is generated for an image once the filter passes over an image. These extracted features are then taken through a feature map, which decides upon the location of the features in the image. There can be two types of Pooling that can be used, namely AveragePooling and MaxPooling. But we have used MaxPooling in our models because we want to determine the features that are outliers, and these are the features that our network visualises and trains upon.

4.1.4 Creating the CNN of our model:

Our model used for medical imaging has multiple Convolutional (Conv2D) layers divided into separate blocks. At the end of each block, a MaxPooling2D layer has been used. These network layers when effectively combined through a trial and error process, help in feature extraction that helps to train our network effectively.

Conv2D layer:

-Syntax:

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
```

```

        use_bias=True,
        kernel_initializer="glorot_uniform",
        bias_initializer="zeros",
        kernel_regularizer=None,
        bias_regularizer=None,
        activity_regularizer=None,
        kernel_constraint=None,
        bias_constraint=None,
        **kwargs
    )

```

This layer generates a tensor of outputs by convolving the layer input with a convolution kernel. A bias vector is generated and applied to the outputs if use_bias is True. Finally, if activation is not None, the outputs are activated as well.

4.2 Code Details and Efficiency:

1st Convolutional block:

```
inputs = layers.Input(shape=(150, 150, 3))
```

Here, the input dimensions of the image are 150 x 150 and 3 is the number of colour channels present. 3 signifies it is a RGB image.

```

# First conv block

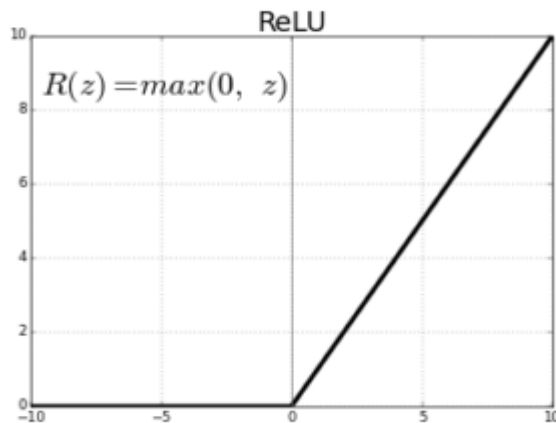
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu',
padding='same')(inputs)

x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu',
padding='same')(x)

```

This is the first of our 5 convolutional blocks. Here, x is the name of the Conv2D layer that is being used. As given above, filters are necessary for feature extraction from the images. Here, 16 such filters will be used when the model trains on the given images.

The activation function used is Relu (**R**ectified **L**inear **U**nit) activation. The graph of the function is given alongside.



Relu activation function is a linear piecewise function that produces the input as output if it is positive, otherwise it produces 0. This function has been used over other activation functions because the model is easier to train and it also achieves better performance.

`padding="same":`

We have used the “same padding” for our model. If the padding is “same”, then the input image is padded in such a way that the entire image gets covered by the movement of the filter and the specified stride length, which here, by default = 1.

```
x = layers.MaxPool2D(pool_size=(2, 2))(x)
```

We have a Max pooling layer in each one of our convolutional blocks of dimensions (2, 2).

2nd Convolutional block:

```
x = layers.SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu',
padding='same')(x)

x = layers.SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu',
padding='same')(x)
```

Here, we have used SeparableConv2D layers, with 32 filters each. The syntax of the layer can be given as:

```
tf.keras.layers.SeparableConv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    depth_multiplier=1,
    activation=None,
    use_bias=True,
    depthwise_initializer="glorot_uniform",
    pointwise_initializer="glorot_uniform",
    bias_initializer="zeros",
```

```

        depthwise_regularizer=None,
        pointwise_regularizer=None,
        bias_regularizer=None,
        activity_regularizer=None,
        depthwise_constraint=None,
        pointwise_constraint=None,
        bias_constraint=None,
        **kwargs
    )

```

A depthwise spatial convolution (which operates on each input channel separately) is performed first, followed by a pointwise convolution that combines the resulting output channels. In the depthwise step, the depth multiplier argument controls the number of output channels provided per input channel. Separable convolutions can be thought of as an extreme version of an Inception block, or a way to factorise a convolution kernel into two smaller kernels.

```

x = layers.BatchNormalization() (x)

```

Batch normalisation is a technique for standardising the inputs to a layer in a deep learning neural network automatically.

Batch normalisation has the effect of significantly speeding up the training phase of a neural network, as well as improving the model's output in some cases through a minor regularisation effect. We have effectively used the BatchNormalization in every Conv2D block.

3rd, 4th and 5th Convolutional block:

The general structure of the Convolutional block remains the same, the only difference being the number of filters that have been used in each layer. Layer 3, 4 and 5 have respectively 64, 128 and 256 filters. The number of filters have been gradually increased for effective extraction of image features as the dimension of the image decreases after every convolutional layer.

In the 4th and 5th Convolutional layer we have also used:

```

x = Dropout(rate=0.2) (x)

```

Dropout is a technique that helps prevent overfitting and also provides means to combine several different neural networks efficiently. The term “dropout” refers to the temporary removal of a neuron from the network, and removing all the incoming and the outgoing connections of that neuron. The neuron to be removed or dropped is chosen completely randomly, or they can have a probability associated with them. For the input neurons:

0.5 < the optimal probability of retention < 1

The rate signifies the fraction of the units to be dropped. Here, 20% of the neurons will be dropped randomly.

The MaxPool2D() layer:

We have used a MaxPool2D layer of dimensions 2x2 in our model. This 2x2 filter (part of the Convolutional layer) passes over each pixel in the image. As the 2x2 filter passes over 2x2 pixels at a time, 3 neighbour pixels also get covered. The filter matrix and the covered image matrix are multiplied element wise, and the values are summed up and populated in the corresponding output pixel.

But this decreases the dimensions of the image. It may so happen that the filter passes over a border pixel. Then, no output will be produced as there are no neighbouring pixels for the filter to fit properly. This will appear as a blacked out pixel around an image of width:

original dimension(=150)/2 = 75.

Thus, the pooling layer is used. To add a padding around the border to prevent the shrinkage in the image size which might result in loss of image features. We have used the MaxPool2D layer, which captures the pixel with the maximum value among the 4 pixels under the layer of dimension 2x2. Thus, with a MaxPooling2D layer of size 2x2 the image size effectively decreases to 1/4th the original image size.

We can see the decrease in the dimensions of the image as follows:

1. The input dimensions of our image is 150x150x3.
2. It is passed through a Convolutional layer of 16 3x3 filters, with padding. Thus, the output dimension of this layer is 150x150x16.
3. The subsequent layer is Max Pooling of dimension 2x2, which shrinks the image by 4 times. So, the output dimension of this layer is 75x75x16.

Thus, the combination of Convolutional and Max Pooling layers decreases the dimension of the original image as illustrated above.

Fully Connected (FC) layers:

```
1. Flatten():  
x = Flatten()(x)
```

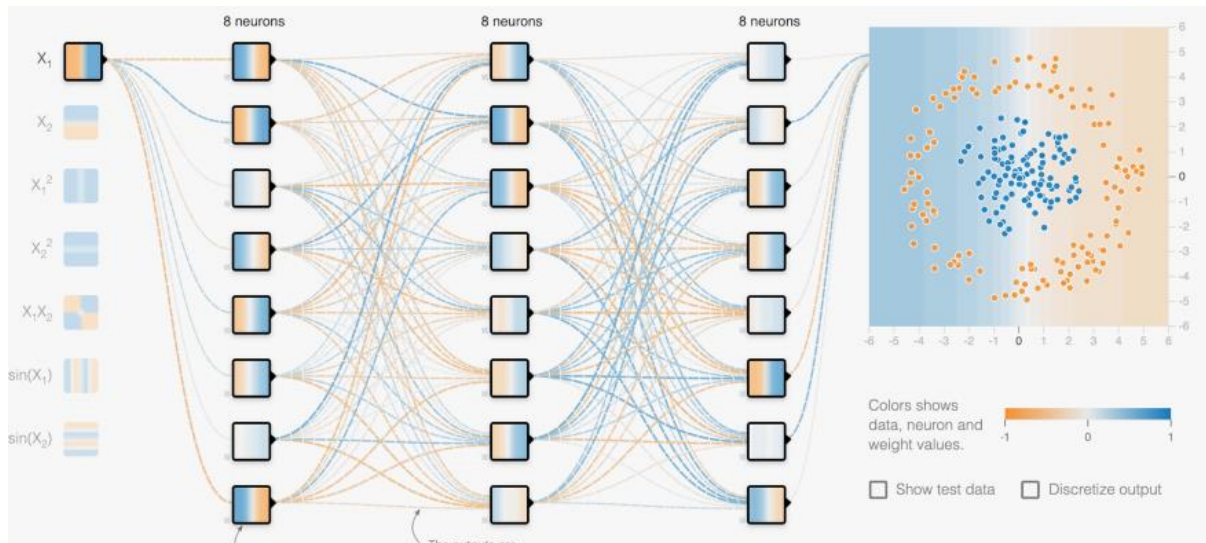
After obtaining the pooled featured map, we need to flatten it. The entire pooled feature map matrix is flattened into a single column, which is then fed to the neural network for processing.

```
2. Dense():  
x = Dense(units=512, activation='relu')(x)
```

We have used 3 Dense layers as our model is a Fully Connected model, with progressively **decreasing** number of units (512, 128 and 64) and Relu activation. The function of the Dense layer is to add the features that have been flattened into a single vector by the Flatten layer, to the network as it is a Fully connected layer.

The Dense layer signifies that each of the neurons in a particular layer receives input from the neurons in the previous layer. Hence, the term “Fully Connected”.

To put it another way, the dense layer is a completely connected layer, which means that all of the neurons in one layer are connected to those in the next.



Structure of a Fully Connected neural network

3. Dropout() :

```
x = Dropout(rate=0.7)(x)
```

Again, we have used Dropout as before. But this time, the dropout rate is significantly higher, 70%. This Dropout is for the Dense layer with 512 units and the rate has been decreased to 50% and then 30% for the successive Dense layers.

```
import keras.backend as K
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormaliza
tion
from keras.layers import Conv2D, SeparableConv2D, MaxPool2D, LeakyReLU,
Activation
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlySt
opping
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, Conv3D, BatchNormalizati
on
from keras import optimizers
from keras import backend as K
from keras import regularizers
model = Sequential()
```

```

model.add(Conv2D(16, kernel_size=(3,3), activation='relu', input_shape=
(150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(512, activation='relu',kernel_regularizer=regularizers.
l1_l2(l1=1e-5, l2=1e-4),
        bias_regularizer=regularizers.l2(1e-
4), activity_regularizer=regularizers.l2(1e-5)))
model.add(Dense(256, activation='relu',kernel_regularizer=regularizers.
l1_l2(l1=1e-5, l2=1e-4),
        bias_regularizer=regularizers.l2(1e-
4),activity_regularizer=regularizers.l2(1e-5)))
model.add(Dense(128, activation='relu',kernel_regularizer=regularizers.
l1_l2(l1=1e-5, l2=1e-4),
        bias_regularizer=regularizers.l2(1e-
4),activity_regularizer=regularizers.l2(1e-5)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid',kernel_regularizer=regularizers
.l1_l2(l1=1e-5, l2=1e-4),
        bias_regularizer=regularizers.l2(1e-4),
        activity_regularizer=regularizers.l2(1e-5)))
print(model.summary())

```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d_18 (MaxPooling)	(None, 74, 74, 16)	0
conv2d_19 (Conv2D)	(None, 72, 72, 32)	4640
max_pooling2d_19 (MaxPooling)	(None, 36, 36, 32)	0
conv2d_20 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_20 (MaxPooling)	(None, 17, 17, 64)	0
conv2d_21 (Conv2D)	(None, 15, 15, 128)	73856
max_pooling2d_21 (MaxPooling)	(None, 7, 7, 128)	0
conv2d_22 (Conv2D)	(None, 5, 5, 256)	295168

max_pooling2d_22	(MaxPooling (None, 2, 2, 256))	0
flatten_4	(Flatten) (None, 1024)	0
dense_2	(Dense) (None, 512)	524800
dense_3	(Dense) (None, 256)	131328
dense_4	(Dense) (None, 128)	32896
dropout_4	(Dropout) (None, 128)	0
dense_5	(Dense) (None, 1)	129
=====		
Total params: 1,081,761		
Trainable params: 1,081,761		
Non-trainable params: 0		
=====		
None		

5. BINARY CLASSIFICATION PROBLEM

In the second stage of the classification module we reformulate the problem as a binary classification task where in the output we will have a single binary output $y \in \{0,1\}$ that will represent the presence or absence of pneumonia in the sample. For this purpose, an end-to-end network has been built based on a feedforward structure designed and trained to detect presence of pneumonia outputs $\hat{y} = P(y=1 | x)$ estimating the probability that a sample contains signs of pneumonia

6. INPUT MODULE

To implement our model in an application, we have created an input module where the user can upload an image of a Chest X-ray and our model will infer from the picture if the lung is healthy or is infected.

The creation of the model can be segregated into the following basic steps:

1. *Importing the required modules:*

```
from keras.models import load_model
from keras.preprocessing import image
```

```
from keras.applications.vgg16 import preprocess_input

import matplotlib.pyplot as plt

import matplotlib.image as mpimg
```

- The `load_model` is used to load the model that we have trained.
- The `image` module which is a part of the `keras.preprocessing` library is used to import images as elements of an array (i.e, the user can upload several different images at once), which are then individually fed into the neural network and the result of each classification is shown.
- `Preprocess_input` is a part of VGG16 network, which is used for Transfer Learning.
- `Matplotlib.pyplot` is used to create a grid-like structure for a clean presentation of the output including the images supplied by the user and the result predicted by our model.
- `Matplotlib.image` is used to **draw** the images as part of the output

2. Importing the trained model:

```
model = load_model('cnn.h5')
```

We employ the `load_model` function which is a part of the `keras.models` library to import the model that we have trained, namely “cnn.h5”.

3. Loading and converting the image into an array, ready to be fed into the neural network:

```
img = image.load_img('', target_size = (150, 150))

x = image.img_to_array(img)
```

Here, the blank field will encompass the name of the file to be uploaded to our model and the target size indicates the dimension of the images that our input needs to be downscaled to. The `x = image.img_to_array(img)` then converts the image given by us into a numpy array.

4. *Displaying the images that are given as an input by the user:*

```
plt.imshow(img)

plt.show()
```

These are library functions simply used to display the images in a grid like format.

5. *Preprocessing the input and feeding the image into the model and printing the prediction class:*

```
img_data = preprocess_input(x)

classes = model.predict(img_data)
```

Here, x is a high dimensional variable which stores our image. It is then sent for preprocessing, and the pre-processed image is then stored in img_data.

Then, the image is fed into the model using model.predict(img_data). The output that the model gives is a list, which is stored in a variable “classes” and the 1st element of that list is the output that we concern ourselves with. Depending on whether it is 0 or 1, we conclude the lung is healthy or infected.

7. RESULTS AND DISCUSSION

7.1 Report

We have trained our model upto 10 epochs with steps per epoch = 500 since the number of images in the training set is 5216 (No. of images = Batch size(10) X No. of Steps) and no. of validation steps =62. On obtaining the accuracies, we plot graphs in order to visualise the success rate of our model using matplotlib.

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=500,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=62,
    verbose=1)
```

```

Epoch 1/10
500/500 [=====] - 273s 547ms/step - loss:
0.8985 - acc: 0.7432 - val_loss: 0.9772 - val_acc: 0.6258
Epoch 2/10
500/500 [=====] - 283s 567ms/step - loss:
0.6891 - acc: 0.7810 - val_loss: 0.7221 - val_acc: 0.6742
Epoch 3/10
500/500 [=====] - 264s 528ms/step - loss:
0.5346 - acc: 0.8327 - val_loss: 0.5921 - val_acc: 0.8210
Epoch 4/10
500/500 [=====] - 248s 495ms/step - loss:
0.4513 - acc: 0.8519 - val_loss: 0.5603 - val_acc: 0.7839
Epoch 5/10
500/500 [=====] - 203s 405ms/step - loss:
0.4002 - acc: 0.8663 - val_loss: 0.4583 - val_acc: 0.8806
Epoch 6/10
500/500 [=====] - 179s 359ms/step - loss:
0.3358 - acc: 0.8817 - val_loss: 0.3658 - val_acc: 0.8806
Epoch 7/10
500/500 [=====] - 185s 369ms/step - loss:
0.3050 - acc: 0.8861 - val_loss: 0.3237 - val_acc: 0.8935
Epoch 8/10
500/500 [=====] - 178s 356ms/step - loss:
0.2889 - acc: 0.8971 - val_loss: 0.4118 - val_acc: 0.9000
Epoch 9/10
500/500 [=====] - 178s 356ms/step - loss:
0.2612 - acc: 0.9037 - val_loss: 0.2953 - val_acc: 0.9210
Epoch 10/10
500/500 [=====] - 174s 349ms/step - loss:
0.2543 - acc: 0.9029 - val_loss: 0.7613 - val_acc: 0.7806

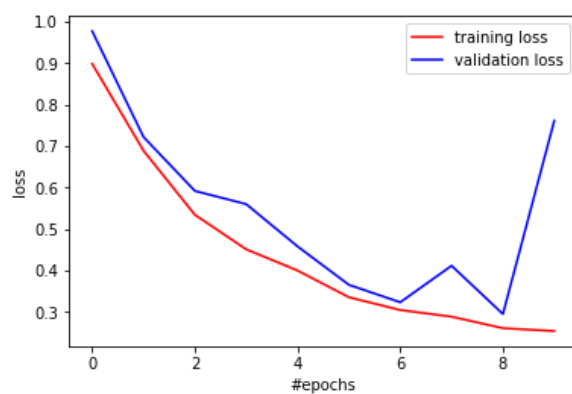
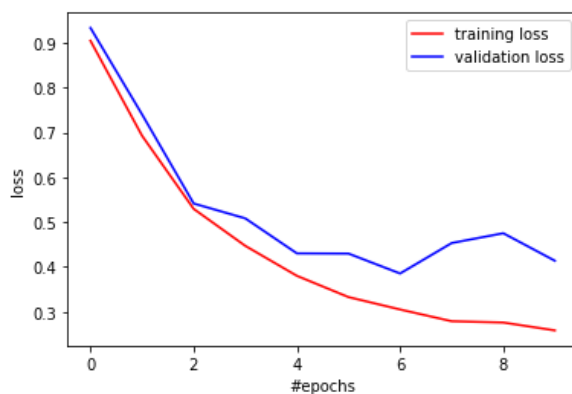
```

7.2 Visualisation – Colab vs Jupyter Notebook

```

plt.plot(history.history['loss'], 'r', label='training loss')
plt.plot(history.history['val_loss'], 'b', label='validation loss')
plt.xlabel('#epochs')
plt.ylabel('loss')
plt.legend()
plt.show()

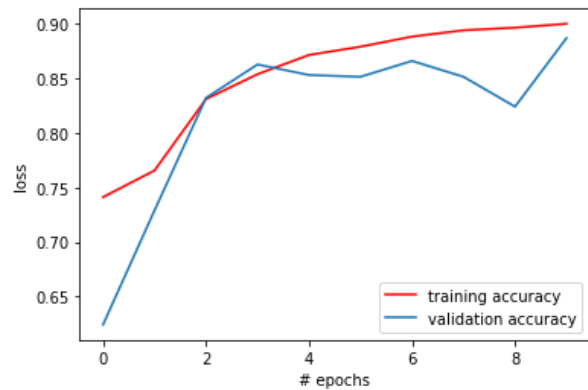
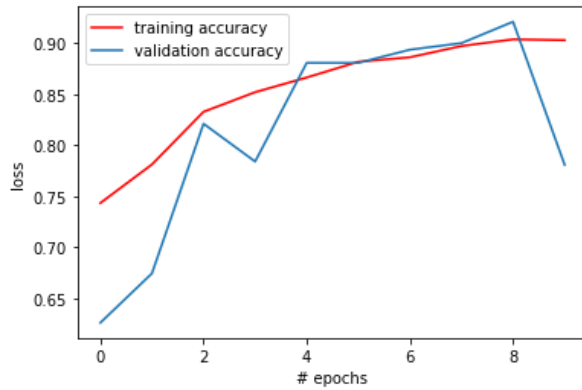
```



```

plt.plot(history.history['acc'],'r',label='training accuracy')
plt.plot(history.history['val_acc'],label='validation accuracy')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()

```



```

import numpy as np
from google.colab import files
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.models import load_model
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input

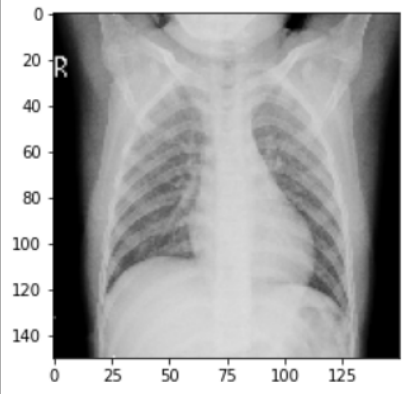
uploaded = files.upload()
model = load_model('cnn.h5')
for fn in uploaded.keys():
    #image prediction
    path = '/content/' + fn
    img = image.load_img(path, target_size = (150, 150))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis = 0)
    img_data = preprocess_input(x)
    images = np.vstack([x])
    classes = model.predict(images, batch_size = 10)
    plt.imshow(img)
    plt.show()
    classes = model.predict(img_data)
    print(classes)
if (classes) >= 1.0:
    print('PNEUMONIA')
else:
    print('NORMAL')

```

Choose Files 8 files

- **NORMAL2-IM-1427-0001.jpeg**(image/jpeg) - 253130 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1430-0001.jpeg**(image/jpeg) - 262713 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1431-0001.jpeg**(image/jpeg) - 250393 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1436-0001.jpeg**(image/jpeg) - 248826 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1437-0001.jpeg**(image/jpeg) - 213744 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1438-0001.jpeg**(image/jpeg) - 170888 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1440-0001.jpeg**(image/jpeg) - 530145 bytes, last modified: 9/28/2019 - 100% done
- **NORMAL2-IM-1442-0001.jpeg**(image/jpeg) - 475144 bytes, last modified: 9/28/2019 - 100% done

Saving NORMAL2-IM-1427-0001.jpeg to NORMAL2-IM-1427-0001 (1).jpeg
 Saving NORMAL2-IM-1430-0001.jpeg to NORMAL2-IM-1430-0001 (1).jpeg
 Saving NORMAL2-IM-1431-0001.jpeg to NORMAL2-IM-1431-0001 (1).jpeg
 Saving NORMAL2-IM-1436-0001.jpeg to NORMAL2-IM-1436-0001 (1).jpeg
 Saving NORMAL2-IM-1437-0001.jpeg to NORMAL2-IM-1437-0001 (1).jpeg
 Saving NORMAL2-IM-1438-0001.jpeg to NORMAL2-IM-1438-0001 (1).jpeg
 Saving NORMAL2-IM-1440-0001.jpeg to NORMAL2-IM-1440-0001 (1).jpeg
 Saving NORMAL2-IM-1442-0001.jpeg to NORMAL2-IM-1442-0001 (1).jpeg

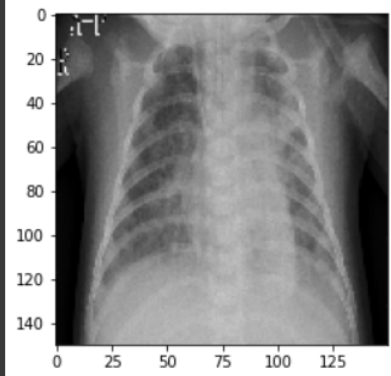


[[0.]]
NORMAL

Choose Files 8 files

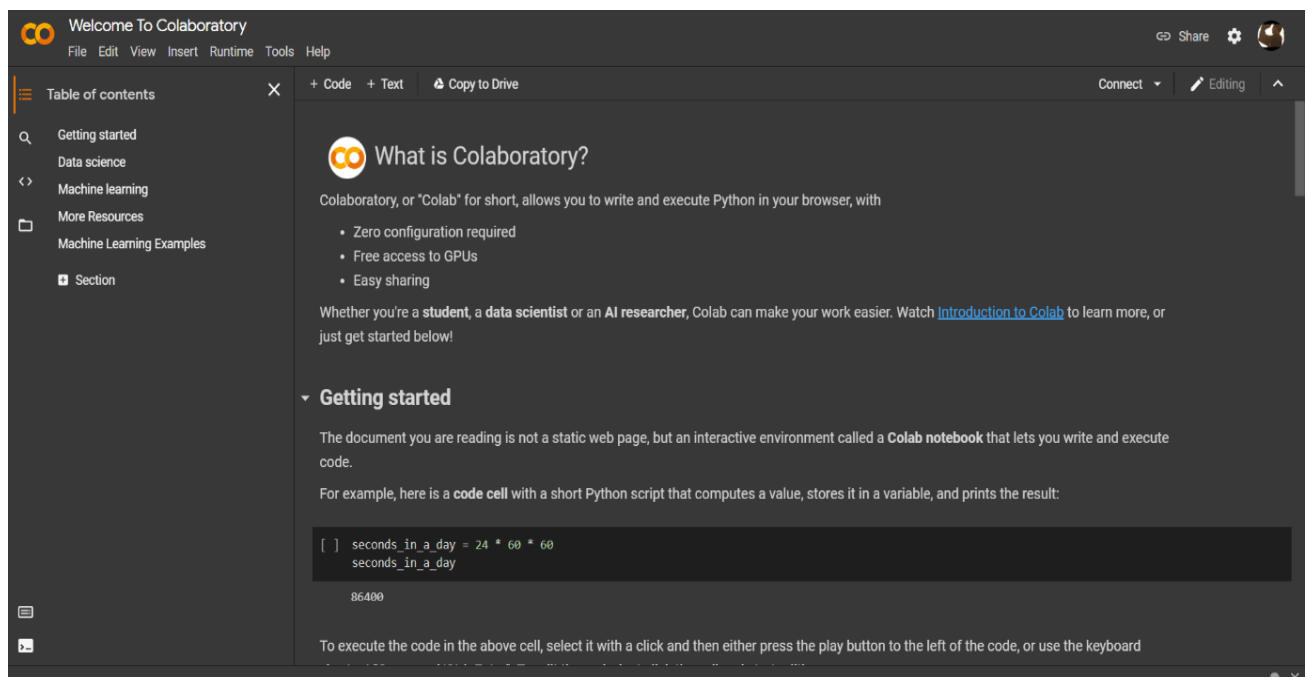
- **person1946_bacteria_4874.jpeg**(image/jpeg) - 61118 bytes, last modified: 9/28/2019 - 100% done
- **person1946_bacteria_4875.jpeg**(image/jpeg) - 59257 bytes, last modified: 9/28/2019 - 100% done
- **person1947_bacteria_4876.jpeg**(image/jpeg) - 62746 bytes, last modified: 9/28/2019 - 100% done
- **person1949_bacteria_4880.jpeg**(image/jpeg) - 99258 bytes, last modified: 9/28/2019 - 100% done
- **person1950_bacteria_4881.jpeg**(image/jpeg) - 86599 bytes, last modified: 9/28/2019 - 100% done
- **person1951_bacteria_4882.jpeg**(image/jpeg) - 49271 bytes, last modified: 9/28/2019 - 100% done
- **person1952_bacteria_4883.jpeg**(image/jpeg) - 87578 bytes, last modified: 9/28/2019 - 100% done
- **person1954_bacteria_4886.jpeg**(image/jpeg) - 119629 bytes, last modified: 9/28/2019 - 100% done

Saving person1946_bacteria_4874.jpeg to person1946_bacteria_4874 (1).jpeg
 Saving person1946_bacteria_4875.jpeg to person1946_bacteria_4875 (1).jpeg
 Saving person1947_bacteria_4876.jpeg to person1947_bacteria_4876 (1).jpeg
 Saving person1949_bacteria_4880.jpeg to person1949_bacteria_4880 (1).jpeg
 Saving person1950_bacteria_4881.jpeg to person1950_bacteria_4881 (1).jpeg
 Saving person1951_bacteria_4882.jpeg to person1951_bacteria_4882 (1).jpeg
 Saving person1952_bacteria_4883.jpeg to person1952_bacteria_4883 (1).jpeg
 Saving person1954_bacteria_4886.jpeg to person1954_bacteria_4886 (1).jpeg



[[1.]]
PNEUMONIA

8. SYSTEM REQUIREMENTS

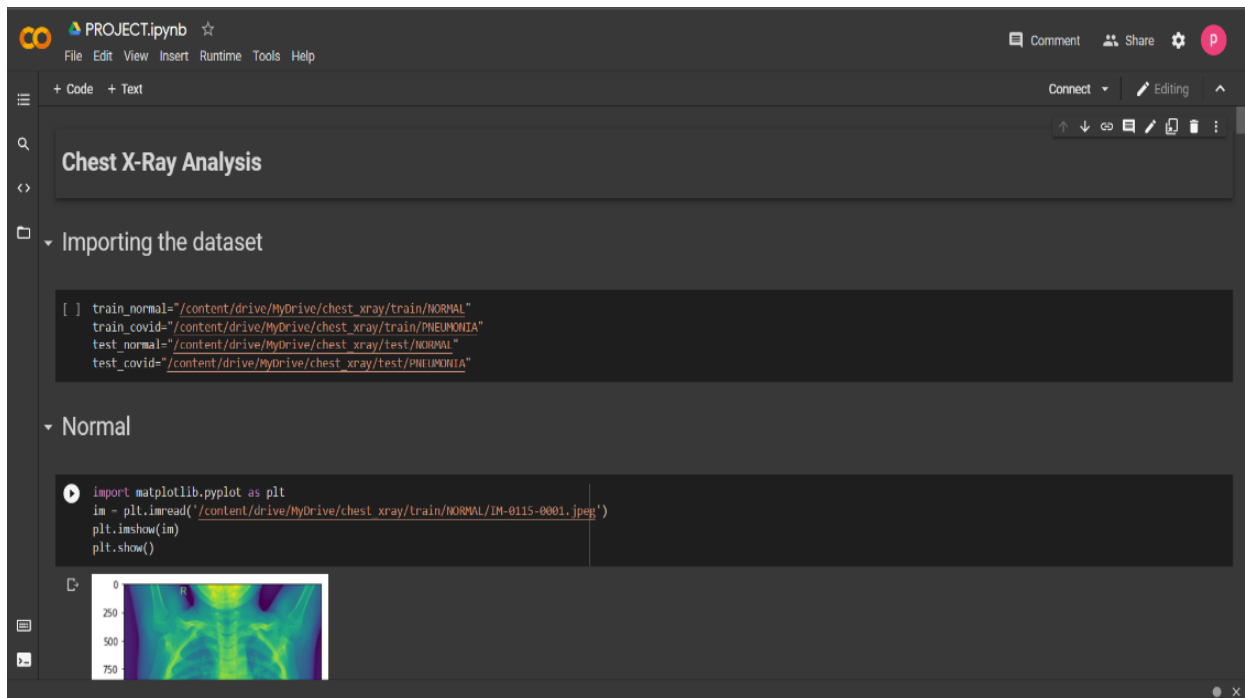


Colab is a free Jupyter notebook environment that runs entirely in the cloud. It does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. It supports many popular machine learning libraries which can be easily loaded in your notebook.

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.

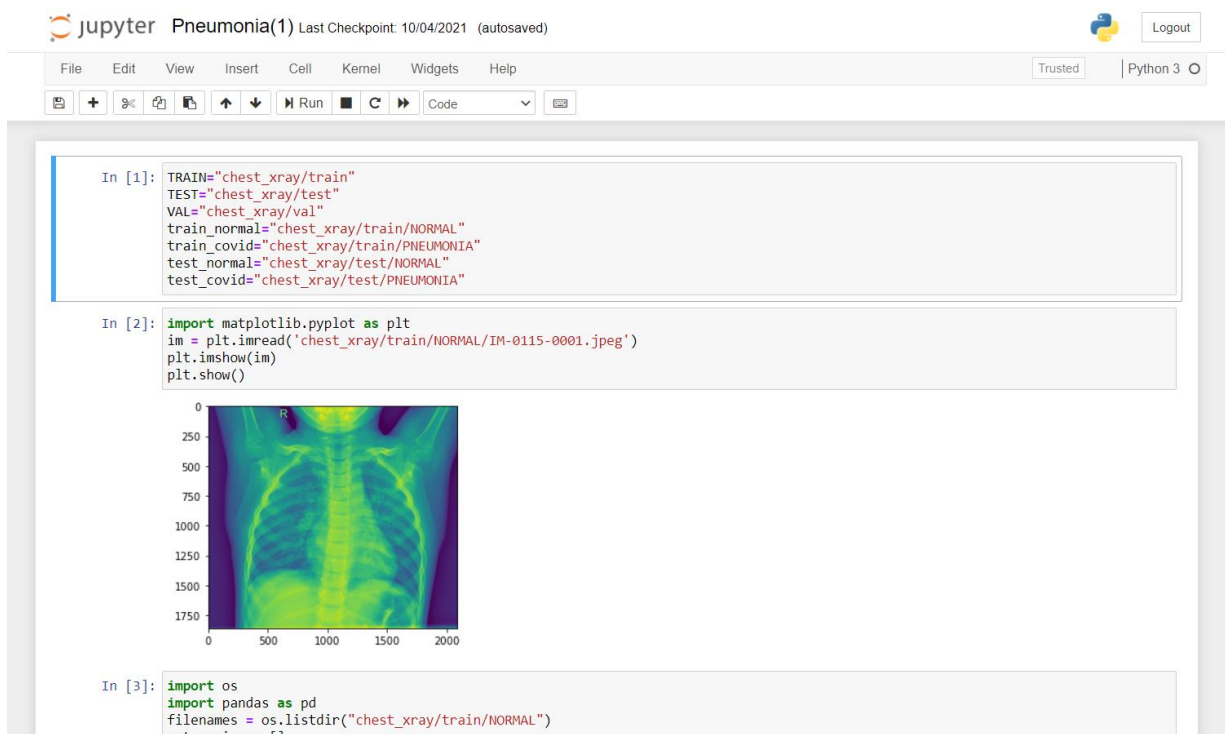
Colab notebooks allow us to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When we create our own Colab notebooks, they are stored in our Google Drive account. We can easily share your Colab notebooks with co-workers or friends, allowing them to comment on our notebooks or even edit them. To learn more, see [Overview of Colab](#).

In Colab, we mounted our dataset from Google Drive and trained our model in a hosted runtime with Hardware Accelerator as GPU. Colab offers a free RAM of 12.72 GB of which only 2.5GB was utilised and a disk space of 68.40 GB of which 38.63 GB is utilised. Here is a view of the Colab Environment for our Project (Source Code):



However, inspite of running our model in this cloud based hosted runtime, we decided to train our model in our local runtime offered by Jupyter Notebooks with system specifications as follows, which yielded slightly better results:

- RAM: 8GB
- Processor: AMD Ryzen 3
- GPU: AMD Radeon Graphics



Our dataset only consisted of 5865 Chest X-Ray images which is less than adequate. In this modern Big Data Era, uncountable data is generated and modified everyday in front of which 5865 images is a very small number. To handle such big amounts of data in the form of image files, proper mechanisms and hardware requirements need to be fulfilled to make our model absolutely successful; the kind of results which are difficult to produce in our environment.

9. CONCLUSION

9.1 Basic Conclusion

In this paper we have attempted to present an approach for designing a high precision automated computed system for diagnosing pneumonia and other related thoracic diseases among patients.

We have detailed every aspect of this task including data cleaning, model design, training and validation of the data. We have also enumerated the limitations of the system. Our system has achieved a promising degree of accuracy in determining Pneumonia among patients. We believe that our model in its current state can be deployed in a clinical setting for testing and to provide the doctor with a second opinion in a matter of minutes. A higher degree of accuracy can be achieved with a larger, higher quality dataset. It is important to note that a completely automated machine cannot replace the trust factor involved in a doctor patient relationship and total reliance on a machine built on limited data should not be relied upon in a clinical setting in its present state. However with the expansion of the datasets and the processing power of computers in accordance with Moore's law as well as the development of new ideas in the field of neural networks and machine learning ; we believe that in the future, cooperation between a radiologist and a ML based medical diagnostic system will improve the outcomes of thoracic disease diagnosis and bring benefits to doctors as well as patients

9.2 Limitations

Although an accurate performance has been achieved, we acknowledge that the proposed method reveals some limitations.

The first issue lies with the dataset itself. For instance, the deeplearning algorithm was trained and evaluated on a Mendeley data source collected from a single tertiary care institution. Therefore, it may not yield the same level of performance when applied to data from other sources such as from other institutions with different scanners. This phenomenon is called geographic variation. To overcome this, the learning algorithm should be trained on data that are diverse in terms of regions, races, imaging protocols, etc. Next, to make a diagnosis from a chest x-ray, doctors often rely on a broad range of additional data such as patient age, gender, medical history, clinical symptoms, and possibly chest x rays from different views. This additional information should also be incorporated into the dataset as it might prove beneficial for training the model to achieve a higher accuracy.

Another issue that we face in the implementation of our algorithm is overfitting of the data with the proposed model which results in lower accuracy than expected.

Overfitting or high variance in machine learning models occurs when the accuracy of the training dataset, i.e. the dataset used to “teach” the model, is greater than the testing accuracy. In terms of ‘loss’, overfitting reveals itself when your model has a low error in the training set and a higher error in the testing set. This can be identified visually by plotting the loss and accuracy metrics and seeing where the performance metrics converge for both datasets.

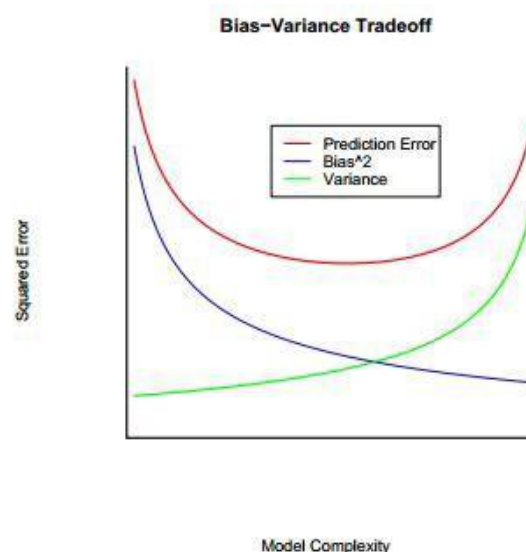
Overfitting indicates that the model is too complex for the problem that it is solving, i.e. the model has too many filters in the case of Convolutional Neural Networks, and layers in the case of overall Deep Learning Models. This causes the model to know the example data well, but perform poorly against any new data.

Overfitting occurs in our model simply because the size of the dataset is too small for the model in question.

In order to improve accuracy of the model in spite of overfitting of the data we have applied Regularization. However regularization has its own disadvantages. This is called a bias error. When we say we have an algorithm that makes a good prediction on test and training data, then, it means that the algorithm draws a regression line that minimizes the sum of squared error. The MSE is the mean of the square difference between our actual value (y or $f(x)$) and prediction.

The more complex the model $f(x)$ the more data points it will capture, and the lower the bias will be. However, complexity will make the model “move” more to capture the data points, and hence its variance will be larger.

In order to reduce the bias, we have to make sure that the expectation value of the predicted is close as much as possible to the true value which means we increase the average of the predicted value in the bias equation. As we increase the average of the predicted value, the average predicted value which is the second term in the variance formula increases. If we increase it too much, then it will be far greater than a single point of the predicted value and this will make the variance to increase more (high variance). However, if the expectation value or the mean of the predicted value is increased such that it is less than the individual single points, then the variance will be minimized in the process. As a result, as we tune our model, we have to observe the effect it has on both the variance and the bias.



Beyond these shortcomings within the algorithm of our proposed model, our model suffers from limitations that can be found among machine learning models in general. Some of these are mentioned below.

Insufficient/Low quality data: The data being provided is too limited for the model to become universally acceptable. If a certain ethnic group, race or nationality shows disproportionate degrees of susceptibility to a disease, the data being presented from that group cannot be used for a more diverse user base. Since machine learning and data acquisition is in its infancy, the majority of data available in the public sector falls under this category.

Time and resources : Machine Learning needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This means that the home computers on which our model was designed is automatically limited by its relatively less powerful system specifications. To overcome this deficiency we used the Google Colab notebook.

Machine Learning is autonomous but highly susceptible to errors. As in our case, if the dataset is small enough to not be inclusive, we end up with biased predictions coming from a biased training set(our training set has a disproportionate no of positive to negative for pneumonia). In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it

9.3 Future Scope of work

As mentioned in the previous section, this master's thesis covers the design and evaluation of a Deep Learning implementation, in the course of which a series of improvement aspects have been observed. On the one hand, with regard to the dataset, we observe 3 evident aspects of improvement:

- To use the same identifier for X-rays of the same patient. Currently, due to privacy policies, each sample is identified with a different unique identifier even if they come from the same patient.
- To record for each patient at least one frontal X-ray and one lateral X-ray. Currently this dataset only has frontal captures, losing a very valuable information and with which professionals normally have to carry out their diagnoses.
- To have available the patient's clinical information. Cases of pneumonia are usually preceded by a history of fever and cough. So, having access to this information - when it comes to making an accurate diagnosis - would give us a more accurate diagnosis.

On the other hand, we have used a variety of state-of-arts convolutional neural networks created for generic use, which we have adapted and trained for our project. In this regard, recently have appeared the first neural networks created integrally by reinforcement learning techniques. It might be interesting to study the possibility of creating from scratch the convolutional neural networks used in diagnosis, optimizing them for our project and for this type of problems through reinforced learning techniques. At the same time, in the development of the implementation we have sought to maximize the performance in terms of quality of predictions obtained through a complex architecture involving several neural networks. This

implementation itself is computationally demanding, requiring high hardware resources for its execution. It might be possible to open a line of research that filters the models that provide the least added value or the use of models with a lower calculation requirement in order to obtain a more effective application at the level of computational resources required. Finally, given the implementation developed and the results obtained, a CAD could be created that integrates this implementation, giving it an application format for use in professional environments and laboratory tests in real cases as support to the professional.

10. REFERENCES

- [1] WHO (n.d.). Pneumonia. Retrieved March 31, 2021, from <https://www.who.int/news-room/factsheets/detail/pneumonia#:~:text=Pneumonia%20is%20a%20form%20of,painful%20and%20limits%20oxygen%20intake>.
- [2] Mooney, P. (2018, March 24). Chest x-ray Images (Pneumonia). Retrieved March 31, 2021, from <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia>
- [3] Ibrahim, A.U., Ozsoz, M., Serte, S. *et al.* Pneumonia Classification Using Deep Learning from Chest X-ray Images During COVID-19. *Cogn Comput* (2021). <https://doi.org/10.1007/s12559-020-09787-5>
- [4] Enríquez, M.V. (2019). A DEEP LEARNING APPROACH FOR PNEUMONIA DETECTION ON CHEST X-RAY.
- [5] T. Gabruseva, D. Poplavskiy and A. Kalinin, "Deep Learning for Automatic Pneumonia Detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA, 2020 pp. 1436-1443. doi: 10.1109/CVPRW50498.2020.00183
url: <https://doi.ieeecomputersociety.org/10.1109/CVPRW50498.2020.00183>
- [6] Ng, Andrew (Instructor). (n.d.). *Neural networks and Deep Learning* [Video file]. Retrieved March 31, 2021, from <https://www.coursera.org/learn/neural-networks-deep-learning>
- [7] Moroney, L. (Writer). (n.d.). *Improving deep neural Networks: Hyperparameter tuning, regularization and optimization* [Video file]. Retrieved March 31, 2021, from <https://www.coursera.org/learn/deep-neural-network>
- [8] Moroney, L. (Director). (n.d.). *Introduction to tensorflow for artificial intelligence, machine learning, and deep learning* [Video file]. Retrieved March 31, 2021, from <https://www.coursera.org/learn/introduction-tensorflow>
- [9] Uniyal, M. (Instructor). (2020, April 27). *Detecting covid-19 from x-ray/ Training a convolutional neural network / Deep learning* [Video file]. Retrieved March 31, 2021, from <https://www.youtube.com/watch?v=nHQDDAAzIsI>

- [10] *Covid-19 detection using x-ray images (convolutional neural Network training)* [Video file]. (2020, July 01). Retrieved March 31, 2021, from <https://www.youtube.com/watch?v=ol0OYJoBC4A>
- [11] Jain, R., Gupta, M., Taneja, S. *et al.* Deep learning based detection and analysis of COVID-19 on chest X-ray images. *Appl Intell* 51, 1690–1700 (2021). <https://doi.org/10.1007/s10489-020-01902-1>
- [12] Jain, S. (2020, May 15). Regularization techniques: Regularization in deep learning. Retrieved March 31, 2021, from <https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/>
- [13] D. Varshni, K. Thakral, L. Agarwal, R. Nijhawan and A. Mittal, "Pneumonia Detection Using CNN based Feature Extraction," 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-7, doi: 10.1109/ICECCT.2019.8869364.
- [14] Convolutional Neural Network Architecture: CNN Architecture | Analytics Vidhya | Accessed on: 31.03.2021 | Published on 14.01.2021 | <https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/>
- [15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", 15(56):1929–1958, 2014