Project Report

On

Detection of Pneumonia using CNN

By

Challa Gopala Krishna Reddy (Roll No. 21ECB0B09)

Kaveti Shrikar (Roll No. 21ECB0B21)

Done Under Provisioning of

Professor Dr. J. Ravi Kumar

Electronics and Communication Engineering Department

Course: AIML (Apr 2022 – July 2022)

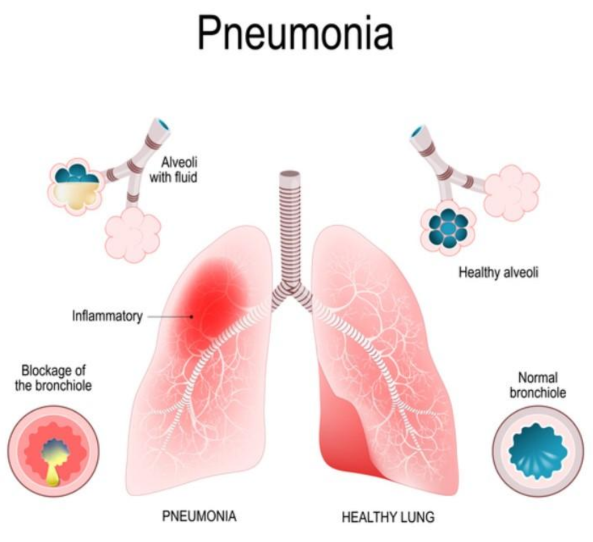
Index

I. Introduction

Pneumonia is an infection that affects one or both lungs. It causes the air sacs, or alveoli, of the lungs to fill up with fluid or pus. Pneumonia may be caused by bacterial, viral, or fungal Infections. Mild to severe symptoms might range from having a cough that produces mucus (a sticky substance), to having a fever, chills, and difficulty breathing. Your age, general health, and the source of your illness all influence severity of the condition.

Your doctor will examine you physically, go through your medical history, order tests like a chest X-ray, and diagnose pneumonia. Your type of pneumonia can be determined using this information. An X-ray aids your doctor in searching for indications of chest inflammation. Your doctor can learn more about the location and severity of any inflammation from the X-ray if it is present.

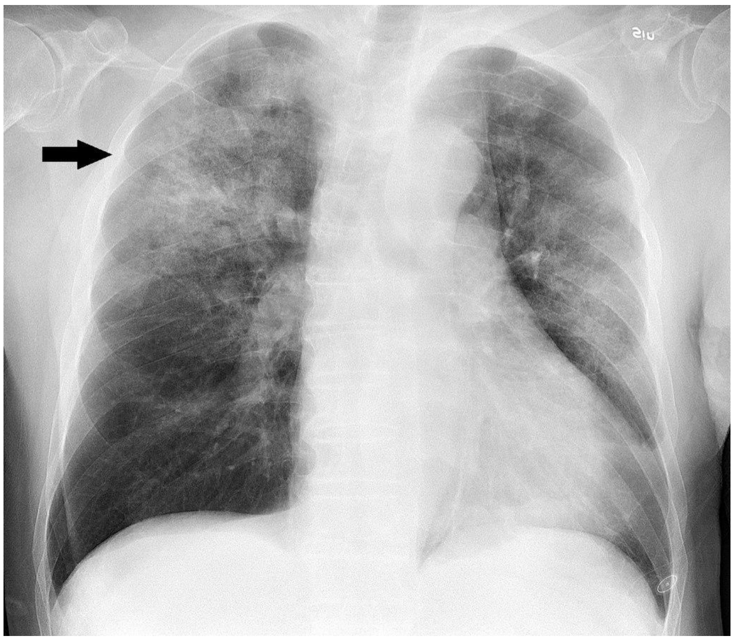
Medicines for viruses, fungi, or antibiotics may be used to treat pneumonia. The recovery from pneumonia could take several weeks. When your symptoms worsen, you should seek medical attention right soon. If you have a severe case of pneumonia, you might need to go to the hospital for oxygen therapy and intravenous (IV) antibiotics. [1] [2]



Everyone is susceptible of get pneumonia, but certain groups do have a higher risk. These groups include:

- Infants (new-born to 2 years old)
- People of Age 65 or Older
- People with weakened immune systems due to:

  1. Pregnancy
  2. HIV
  3. Use of certain medications (such as steroids or certain cancer drugs)



Chest radiograph of an 88-year-old man, about one week after onset of fever, fatigue and mild coughing. Lab tests detected both Influenza A virus and *Haemophilus influenzae*. It shows multifocal, patchy consolidation, mainly in the right upper lobe.

Chest X-Rays analysis is regarded as the most effective method in detection of pneumonia and degree of seriousness. But the examination of chest radiographs is not a simple task for radiotherapists. The appearance of pneumonia on chest X-rays might be unclear and confused with other illnesses. As many other conditions, such as congestive cardiac failure, lung scarring, etc., can resemble a pneumonia, evaluation of X-rays while examining for pneumonia can be misleading. Thus, creating an efficient algorithm for the detection of pneumonia can save a lot of time for the medical professionals and can help them take actions in order to tackle the problem as soon as possible.

II. Methodology (Algorithm)

In order to detect pneumonia, chest x-rays are widely used. Hence, we can predict the presence of pneumonia using models which can extract features from the image and then give the result which tells whether a person has pneumonia or not. In order to do this, Convolutional Neural Networks are used as they are good at picking up various features from the input image which makes it the best model for computer vision-based applications.

The whole methodology can be classified into the following steps:

1. Importing required Libraries
2. Loading and Visualising Dataset
3. Data Pre-processing
4. Data Augmentation
5. Model Training
6. Model Testing and Predictions

Importing Libraries

Libraries Used in the project -

1. Matplotlib PyPlot
2. Seaborn
3. TensorFlow
    a. Keras API
    b. In Keras - Model (Sequential Class)
    c. In Keras - layers (Dense, Conv2D, MaxPool2D, Flatten, Dropout, Batch Normalization)

4. Pre-Processing (Image Data Generator)
5. Call backs (Reduce LR on Plateau)
6. Sklearn
    a. Sklearn - Metrics (Classification Report, Confusion Matrix)

7. Open CV2 Python
8. OS Python

1. **Matplotlib**

   Matplotlib. pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

   One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc. [3]

2. Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It is used to visualize random distributions. One of the main advantages of seaborn is that it generates variety of visualization patterns using less syntax. It is mainly used in statistics visualization and for summarizing the data and to show distribution using visualizations. [4]

3. TensorFlow

TensorFlow is an open-source library for fast numerical computing. It was created and is maintained by Google and was released under the Apache 2.0 open-source license.

TensorFlow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++.

TensorFlow is mainly used for deep learning and machine learning applications such as image recognition, voice search etc.

TensorFlow 2.0 is Built on Keras API which is used to build Custom CNN Model which is explained in detail later in Model Training Section of the Report.

4. Sklearn

Scikit-learn is an important library used for machine learning in python. sklearn library contains a lot of efficient tools for testing in machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction.

Model selection (train test split) is a function included in Sklearn which divides the dataset into training data and testing data. The percentage of data given for training and testing is given as input by the user and testing data is randomly selected from the input and tested for calculating model accuracy.

5. Open-cv Python:

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

6. OS Python:

The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc. It offers many useful OS functions that are used to perform OS-based tasks and get related information about the operating system.

Data pre-processing:

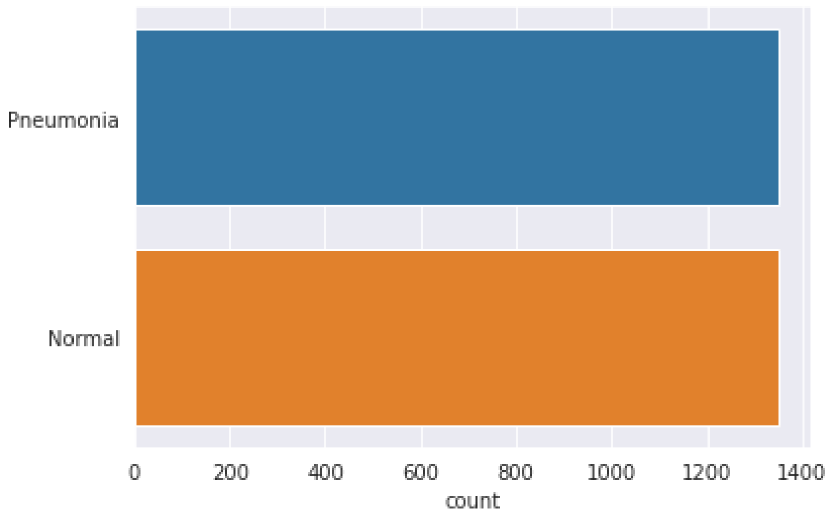
CNN is widely used in image classification problems as it can reduce the computational complexity of the model, which is likely to increase if the input is in the form of images. The pre-processing techniques used on the images can be summarised as follows:

1. Reading Image Data and Creating a NumPy array
2. Resizing Images
3. Dividing the Intensity of the Image by 255 to avoid Saturation
4. Reshaping Input NumPy array data

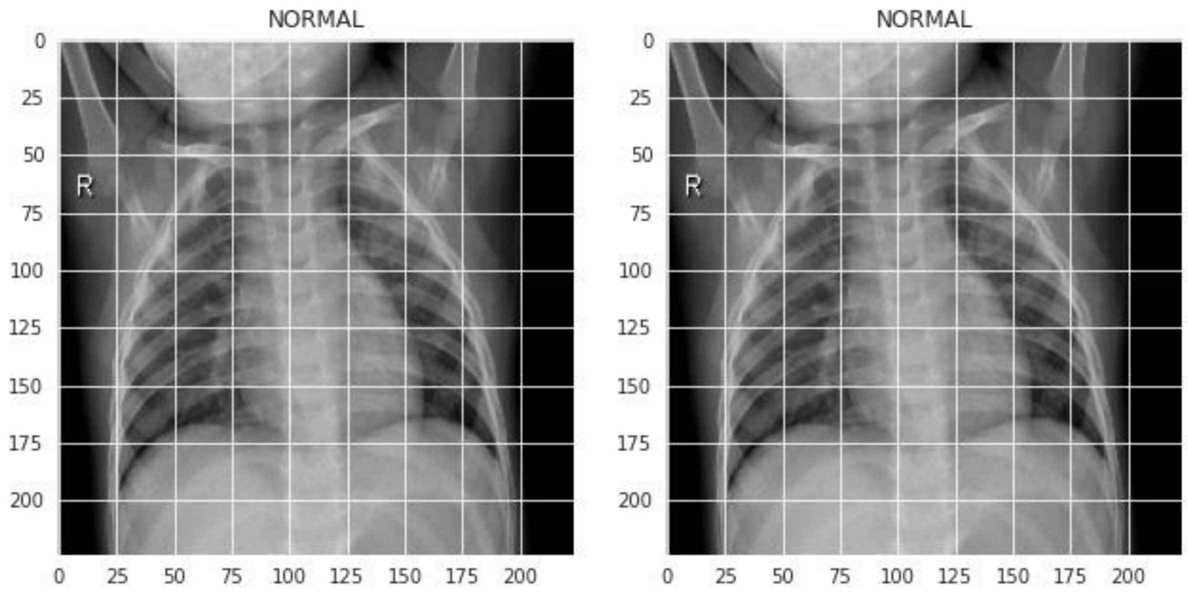
In pre-processing, we first give a label to every image. After that, we are creating a variable that stores the index of the label in the original array of labels. Then, we use function cv2.imread to convert all the images into the form of multi-dimensional arrays where each element represents the intensity of a pixel.

After this, we use cv2.resize to change the dimensions of each image array from 1302 x 1564 x 3 to 224 x 224 x 3. After resizing the image array, we create a list that consists of this and a number representing whether the image is a pneumonia sample or a normal sample.

Finally, we create a list which consists of all such lists or in other words contains list of all images and index which represents whether it is a pneumonia sample or not. This process is applied for training, testing and validation sets. These lists are used to feed the model which is to be trained or tested.



Input Data



Input Images after Resized to (224, 224, 3)

Data augmentation

Data augmentation strategies are methods that modify the training data in a way that modifies the array representation while preserving the label.

Gray scaling, vertical and horizontal flips, random cropping, colour hiccups, translations, rotations, and many other augmentations are frequently used.

We can easily double or quadruple the number of training examples and build a very robust model by applying only a few of these adjustments to our training data. By using this method, we can make sure that the algorithm doesn’t overfit to the given dataset.

We used the following augmentation techniques

1. **Rotation Images (Up to 30 Degree)**

   Images are rotated up to 30 degrees as the sample which needs to be predicted can be 30 degrees shift because of errors or camera position or patient may have some abnormalities, by training model by rotated images the model can predict sample with above mentioned abnormalities.

2. **Zooming Images**

   Images are zoomed randomly as samples which need to be predicted can be zoomed in or zoomed out. We are zooming in and zooming out image by a factor of 0.2

3. **Width shifting**

   The images are shifted by a factor of 0.1 horizontally to train the model for cases where the test sample is shifted.

4. **Height shifting**

   The images are shifted by a factor of 0.1 vertically to train the model for cases where the test sample is shifted.

5. **Horizontal flip**

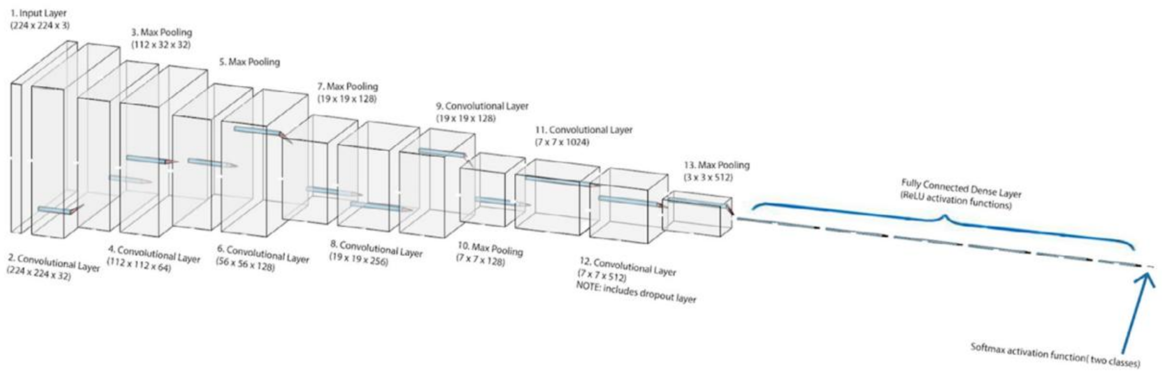
The images are flipped horizontally to train the model for cases where the test sample if flipped.

All these techniques are applied randomly on images and this increases the number of test cases.

III. Application (Model Training)

Images are turned into data arrays which is then passed through a bunch of layers before passing them through the Fully Connected Neural Network.

Total Layers numbers of Layers used in the Model are 26 and Input Shape of Each Image Data is (224 x 224 x 3)


1. Input Layer (224 x 224 x 3)
3. Max Pooling (112 x 32 x 32)
5. Max Pooling
7. Max Pooling (19 x 19 x 128)
9. Convolutional Layer (19 x 19 x 128)
11. Convolutional Layer (7 x 7 x 1024)
13. Max Pooling (3 x 3 x 512)
Fully Connected Dense Layer (ReLU activation functions)
2. Convolutional Layer (224 x 224 x 32)
4. Convolutional Layer (112 x 112 x 64)
6. Convolutional Layer (56 x 56 x 128)
8. Convolutional Layer (19 x 19 x 256)
10. Max Pooling (7 x 7 x 128)
12. Convolutional Layer (7 x 7 x 512) NOTE: includes dropout layer
Softmax activation function(two classes)

Convolution Layer

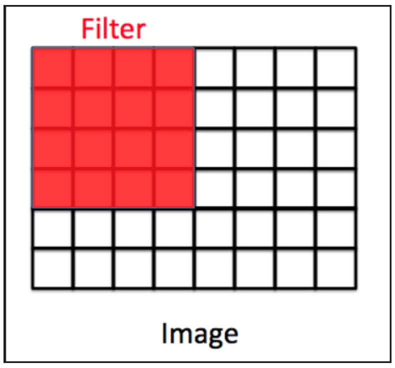
Convolution is an orderly procedure where two sources of information are intertwined; it's an operation that changes a function into something else. The first layer of a Convolutional Neural Network is always a **Convolutional Layer.**

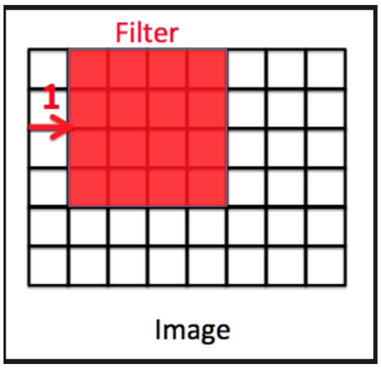
Convolutional layers apply a convolution operation to the input, passing the result to the next layer. A convolution converts all the pixels in its receptive field into a single value. The final output of the convolutional layer is a vector.

Convolution is primarily used to extract features from the image. We can use different sizes of kernels depending on the situation to train the model better and get fewer errors. [5]

Striding

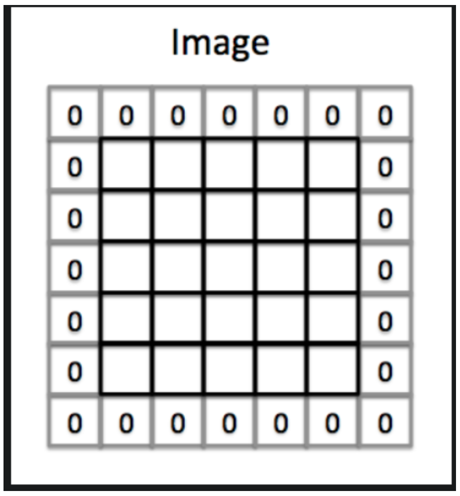
Convolutional neural networks are specialized for the compression of image and video data, include stride as a component. The neural network's filter's stride parameter determines the number of Pixels or Rows and Columns the filter is moved on the Image. The Lesser number of pixels the filter is moved on the image we get more detail of the Image. In the Model, Filters are Stride by 1 Pixel





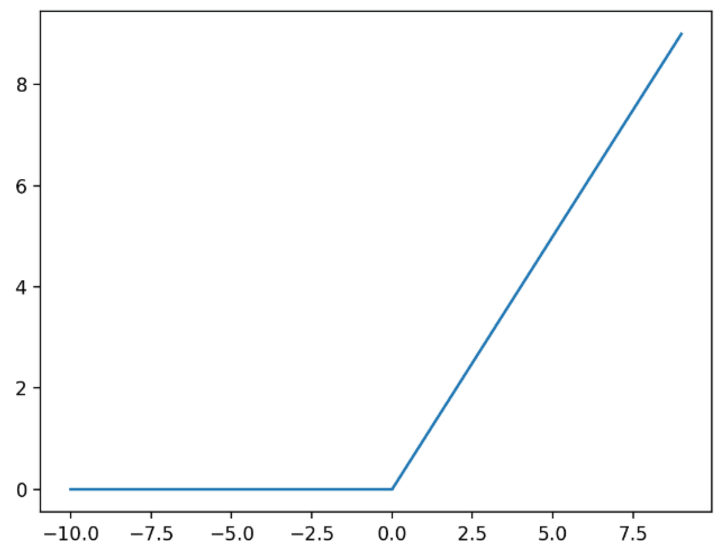
Padding

Padding refers to the number of pixels added to the border of the image before filtering. Padding Image enables filters to read the data of edge pixels more detailed. In Chest Xray, Most Important data is in centre of the Xray. Hence, Padding is not required for the Model.



ReLU Activation Function

The Rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It is the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. [6]

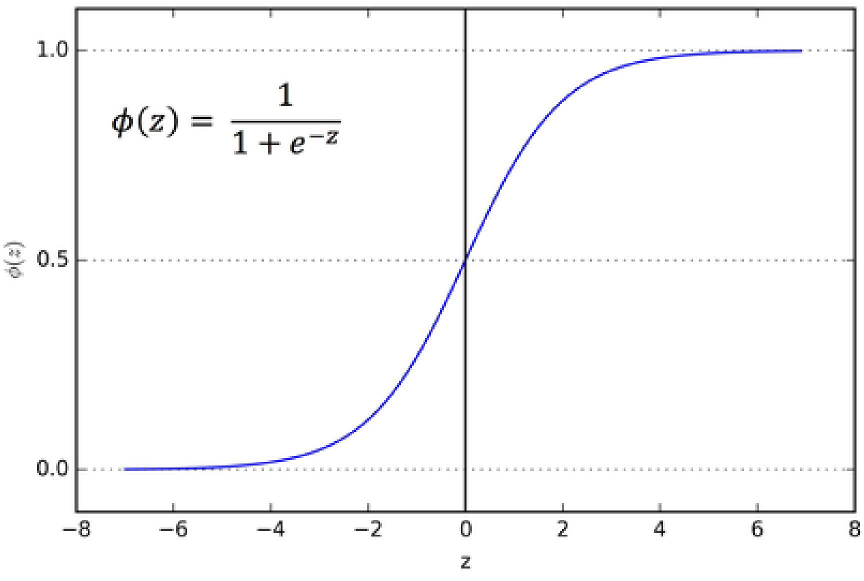


Sigmoid activation function

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve.

A common example of a sigmoid function is the logistic function shown in the first figure and defined by the formula on the Right Side

A sigmoid function is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point and exactly one inflection point. A sigmoid "function" and a sigmoid "curve" refer to the same object. [7]

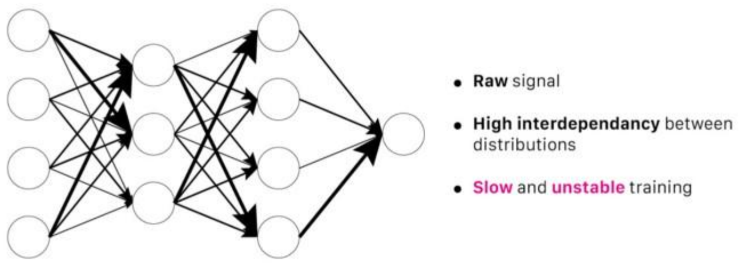


Batch Normalisation

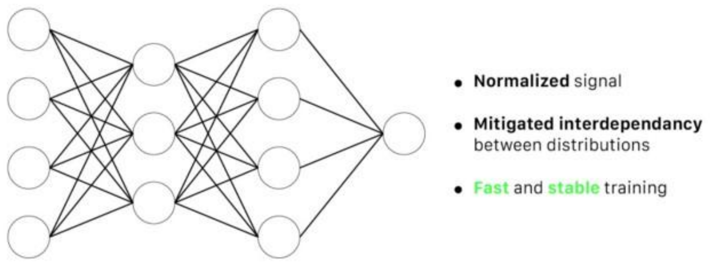
Multiple layers of deep neural networks may be sensitive to the setting of the learning algorithm and the initial random weights. After each mini-batch, when the weights are adjusted, the distribution of inputs to layers deep in the network may alter. This could result in the learning process whose aim is to chase a moving object indefinitely. The use of batch normalization can stop this.

When training deep neural networks, batch normalization is used to equalize the inputs to each layer for each mini-batch. As a result, the learning process is stabilized and the amount of training epochs needed to train deep networks is drastically decreased.

The main use of the batch normalization technique is to randomize/normalize the input in different layers of the neural network which helps in better training and also makes sure the model doesn't overfit. [8]

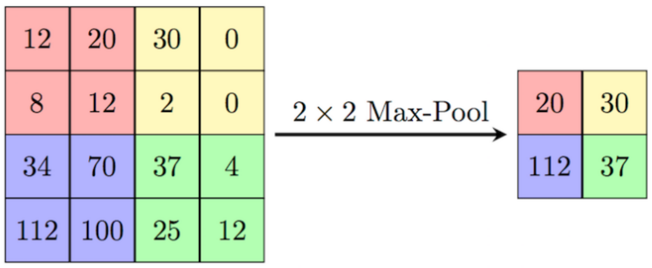


Raw signal
High interdependancy between distributions
Slow and **unstable** training



Max pooling

Max pooling is a pooling operation that calculates the maximum values for patches of a feature map and uses it to create a smaller version of the feature map. It is generally used after the convolution layer.



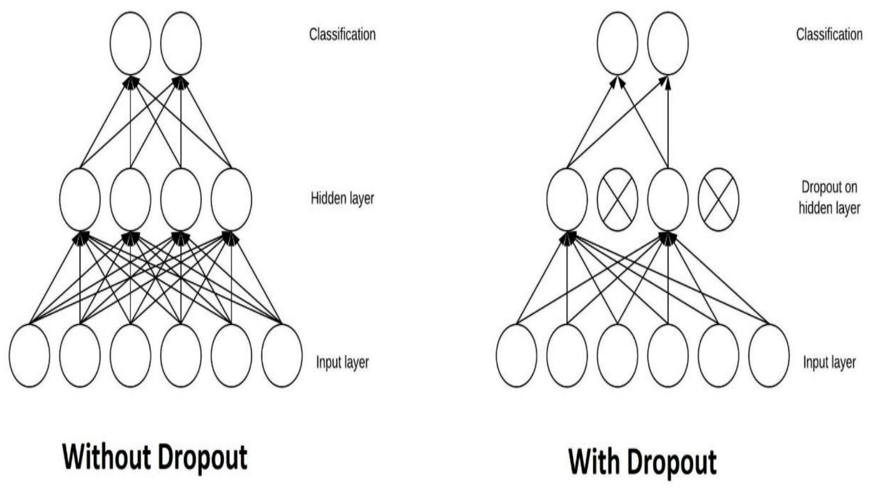
Max pooling is generally used in order to reduce the dimensions of feature maps which in turn reduces the computations performed in a network. It also makes the model more robust to variations in the position of the features in the input image.

Average pooling is another pooling operation that is generally used in neural networks.

Dropout

Dropout refers to ignoring units (i.e., neurons) during the training phase of a certain set of neurons which is chosen at random. Individual nodes are either kept in the network with probability p or pulled out with probability 1-p at the end of each training cycle, leaving behind a smaller network with its incoming and outgoing edges.

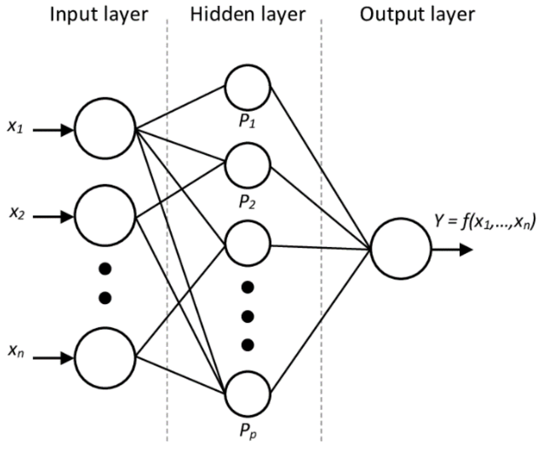
Dropout is used in order to avoid overfitting. Since a fully connected layer takes up the majority of the parameters, co-dependency between neurons during training reduces each neuron's individual power and causes training data to be overly fitted.



Dense layer

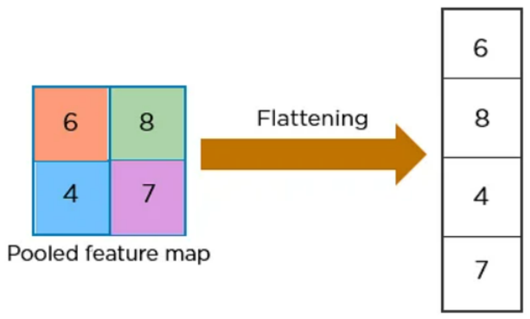
The dense layer is a deep neural network layer connected, meaning each neuron in the dense layer receives input from all neurons of its previous layer.

The dense layer multiplies matrices and vectors. Backpropagation is used to train and update the parameters that make up the values utilized in the matrix. Updated Parameters are replaced to reduce loss.



Flatten Layer

Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.



The flattened matrix is fed as input to the fully connected layer to classify the image

Optimizers

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

Adagrad is an extension of gradient descent with a changing learning rate for each individual parameter (Weights).

Adagrad finds learning of each parameter by first summing all partial derivatives seen so far in NN and then divides the initial learning rate by the square root of the sum of squares of partial derivative.

Rmsprop is an extension of Adagrad with decaying Average. Rmsprop is a Second Order Differential Equation while gradient decent is a First Order Differential Equation because of which Rmsprop does get struck at local minima and can find global minima of the loss function. [9]

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}),$$

A derivative of loss function for given parameters at a given time t.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

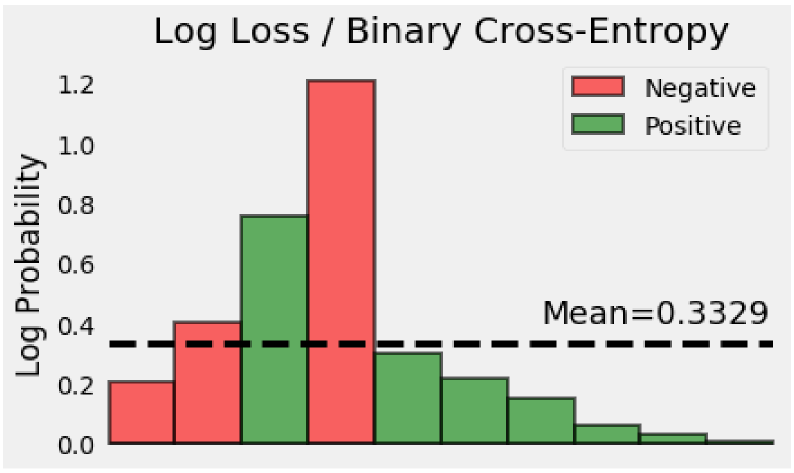
Update parameters for given input (i) and at time / iteration (t).

Binary Cross-Entropy

Binary cross entropy compares each of the predicted probabilities to the actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. The Loss is calculated by Negative log of probability of the Output being Pneumonia sample. [10]

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Equation of Loss Function



Visual Representation of Loss Function Calculation

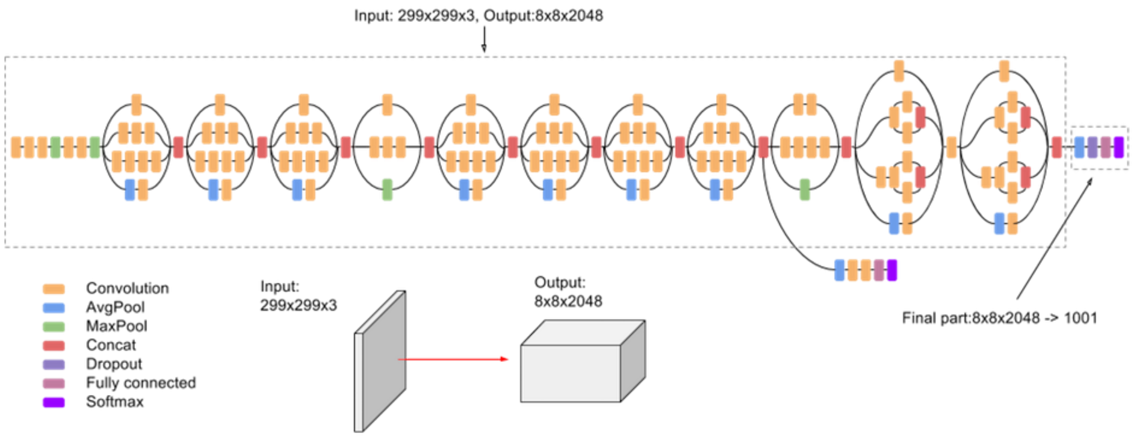
Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a similar task.

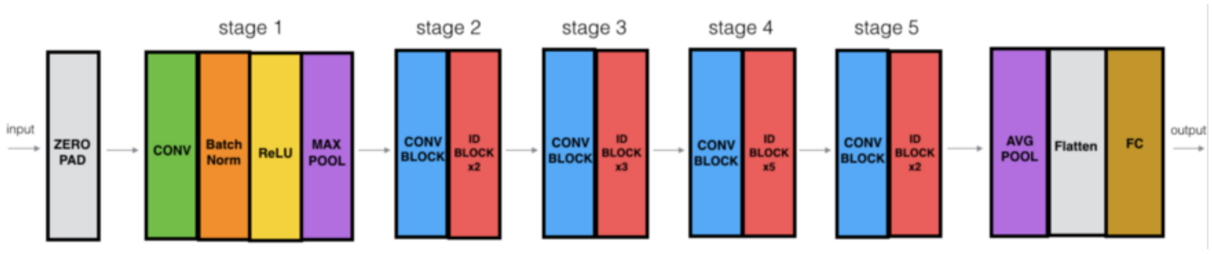
Pre-trained models are frequently utilized as the foundation for deep learning tasks in computer vision and natural language processing because they save both time and money compared to developing neural network models from scratch and because they perform vastly better on related tasks. [11]

We implemented Resnet50 and Inception V3 and compared their performances with the model implemented in the paper.

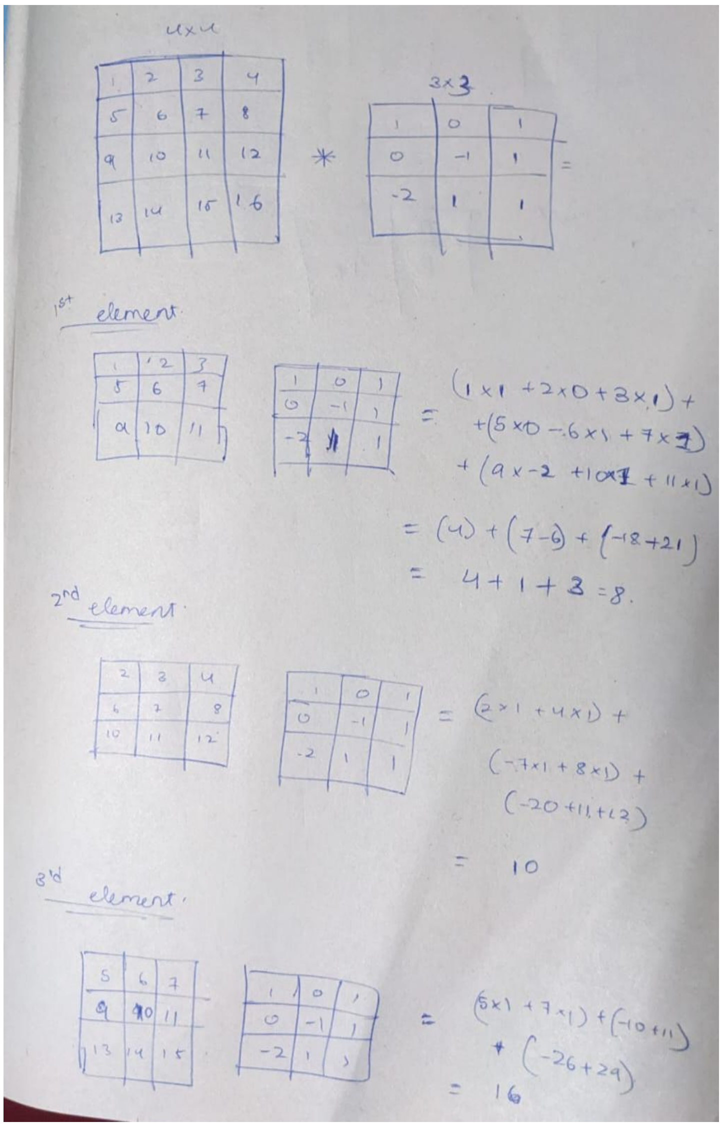
Inception V3



Resnet50



Convolution Calculation



4x4

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} * \overset{3\times 3}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ -2 & 1 & 1 \end{bmatrix}} =$$

1st element

$$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ -2 & 1 & 1 \end{bmatrix} = (1\times1 + 2\times0 + 3\times1) + {} + (5\times0 - 6\times1 + 7\times1) + (9\times-2 + 10\times1 + 11\times1)$$

$$= (4) + (7-6) + (-18+21)$$

$$= 4 + 1 + 3 = 8.$$

2nd element

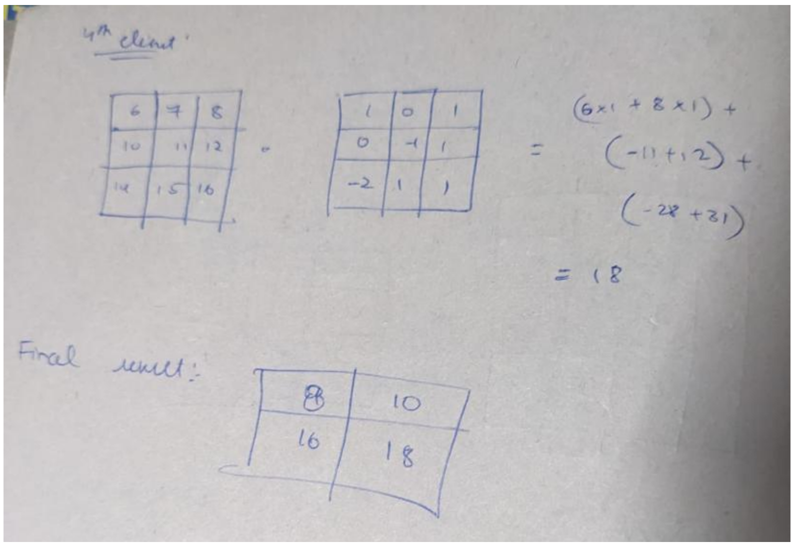
$$\begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ -2 & 1 & 1 \end{bmatrix} = (2\times1 + 4\times1) + (-7\times1 + 8\times1) + (-20 + 11 + 12)$$

$$= 10$$

3rd element

$$\begin{bmatrix} 5 & 6 & 7 \\ 9 & 10 & 11 \\ 13 & 14 & 15 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ -2 & 1 & 1 \end{bmatrix} = (5\times1 + 7\times1) + (-10+11) + (-26+29)$$

$$= 16$$



4th element

$$\begin{bmatrix} 6 & 7 & 8 \\ 10 & 11 & 12 \\ 14 & 15 & 16 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ -2 & 1 & 1 \end{bmatrix} = (6\times1 + 8\times1) + (-11+12) + (-28+31)$$

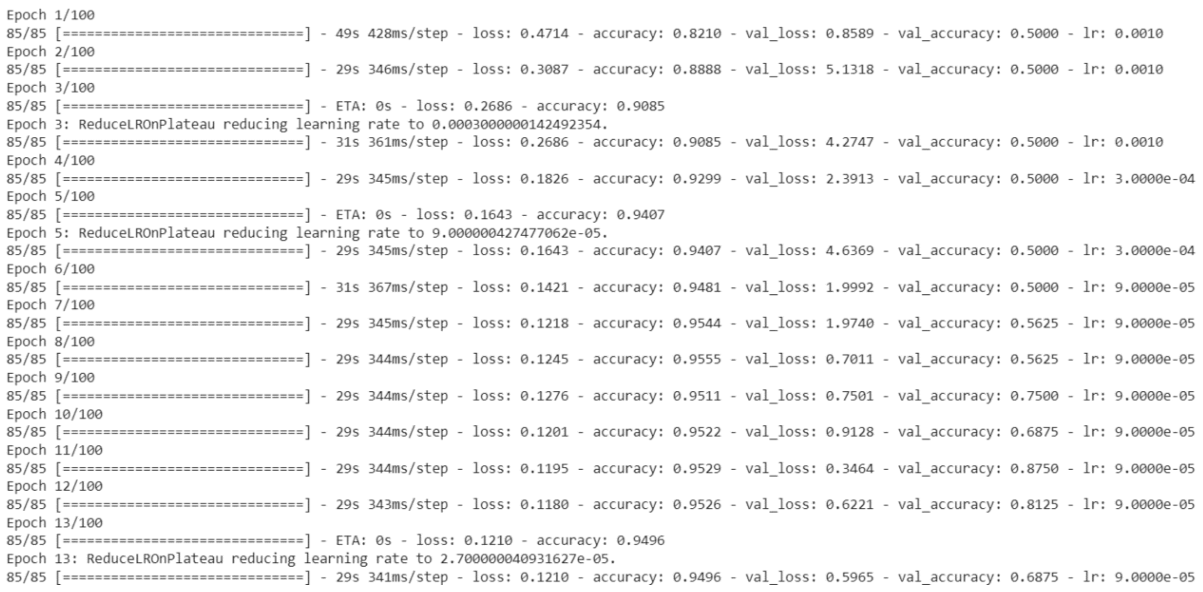
$$= 18$$

Final result:

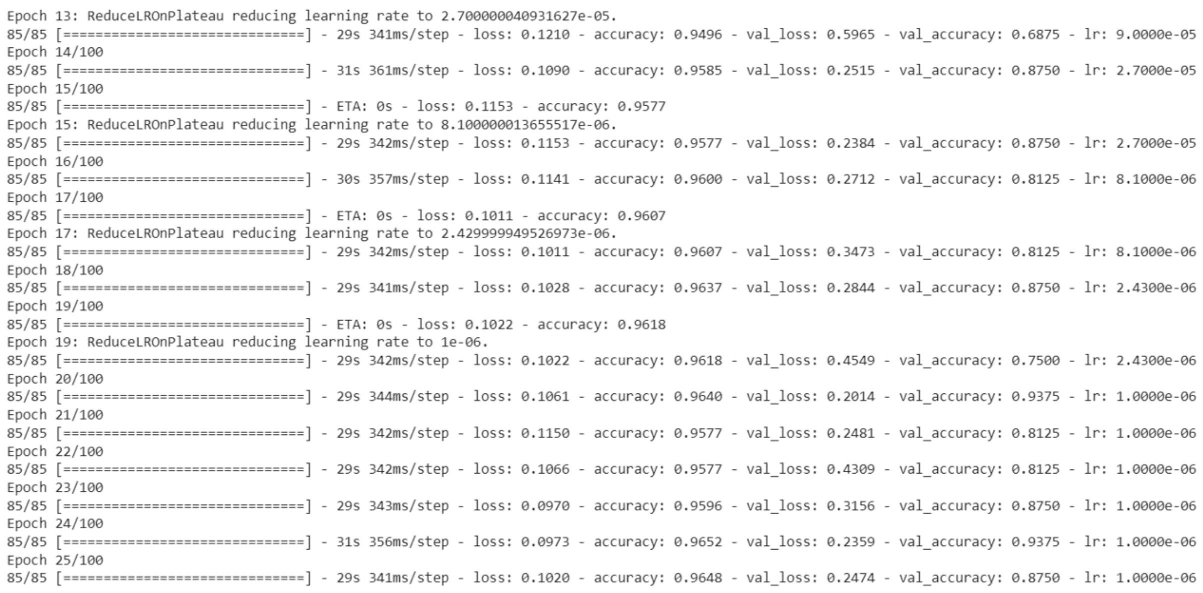
$$\begin{bmatrix} 8 & 10 \\ 16 & 18 \end{bmatrix}$$

IV. Simulations and Results

Epoch Results

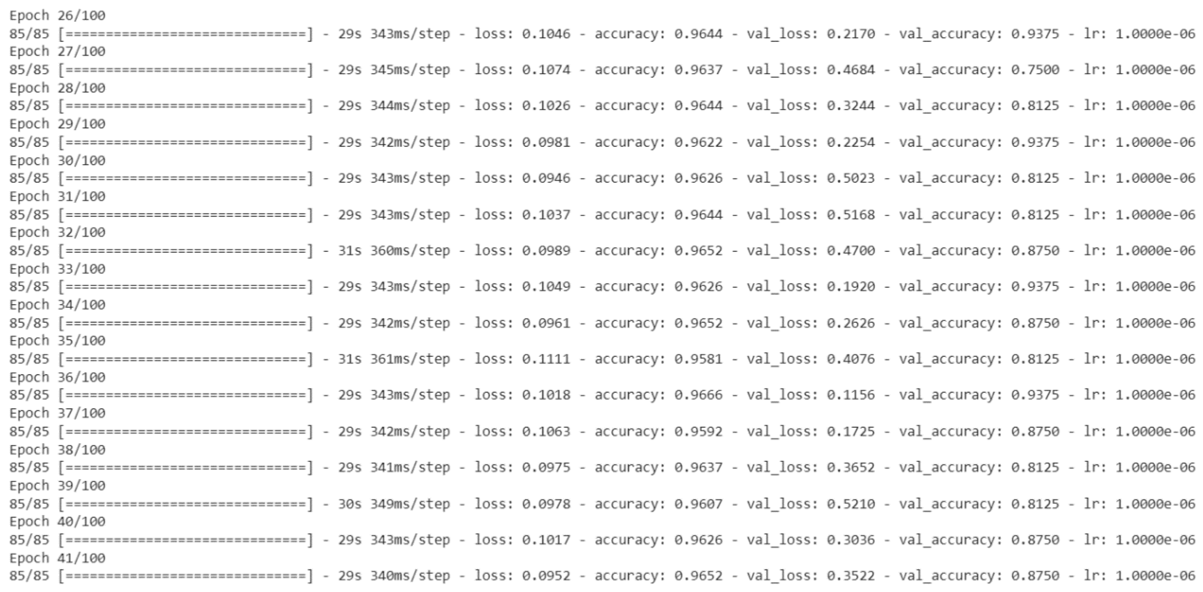


```
Epoch 1/100
85/85 [==============================] - 49s 428ms/step - loss: 0.4714 - accuracy: 0.8210 - val_loss: 0.8589 - val_accuracy: 0.5000 - lr: 0.0010
Epoch 2/100
85/85 [==============================] - 29s 346ms/step - loss: 0.3087 - accuracy: 0.8888 - val_loss: 5.1318 - val_accuracy: 0.5000 - lr: 0.0010
Epoch 3/100
85/85 [==============================] - ETA: 0s - loss: 0.2686 - accuracy: 0.9085
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
85/85 [==============================] - 31s 361ms/step - loss: 0.2686 - accuracy: 0.9085 - val_loss: 4.2747 - val_accuracy: 0.5000 - lr: 0.0010
Epoch 4/100
85/85 [==============================] - 29s 345ms/step - loss: 0.1826 - accuracy: 0.9299 - val_loss: 2.3913 - val_accuracy: 0.5000 - lr: 3.0000e-04
Epoch 5/100
85/85 [==============================] - ETA: 0s - loss: 0.1643 - accuracy: 0.9407
Epoch 5: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
85/85 [==============================] - 29s 345ms/step - loss: 0.1643 - accuracy: 0.9407 - val_loss: 4.6369 - val_accuracy: 0.5000 - lr: 3.0000e-04
Epoch 6/100
85/85 [==============================] - 31s 367ms/step - loss: 0.1421 - accuracy: 0.9481 - val_loss: 1.9992 - val_accuracy: 0.5000 - lr: 9.0000e-05
Epoch 7/100
85/85 [==============================] - 29s 345ms/step - loss: 0.1218 - accuracy: 0.9544 - val_loss: 1.9740 - val_accuracy: 0.5625 - lr: 9.0000e-05
Epoch 8/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1245 - accuracy: 0.9555 - val_loss: 0.7011 - val_accuracy: 0.5625 - lr: 9.0000e-05
Epoch 9/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1276 - accuracy: 0.9511 - val_loss: 0.7501 - val_accuracy: 0.7500 - lr: 9.0000e-05
Epoch 10/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1201 - accuracy: 0.9522 - val_loss: 0.9128 - val_accuracy: 0.6875 - lr: 9.0000e-05
Epoch 11/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1195 - accuracy: 0.9529 - val_loss: 0.3464 - val_accuracy: 0.8750 - lr: 9.0000e-05
Epoch 12/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1180 - accuracy: 0.9526 - val_loss: 0.6221 - val_accuracy: 0.8125 - lr: 9.0000e-05
Epoch 13/100
85/85 [==============================] - ETA: 0s - loss: 0.1210 - accuracy: 0.9496
Epoch 13: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
85/85 [==============================] - 29s 341ms/step - loss: 0.1210 - accuracy: 0.9496 - val_loss: 0.5965 - val_accuracy: 0.6875 - lr: 9.0000e-05
```

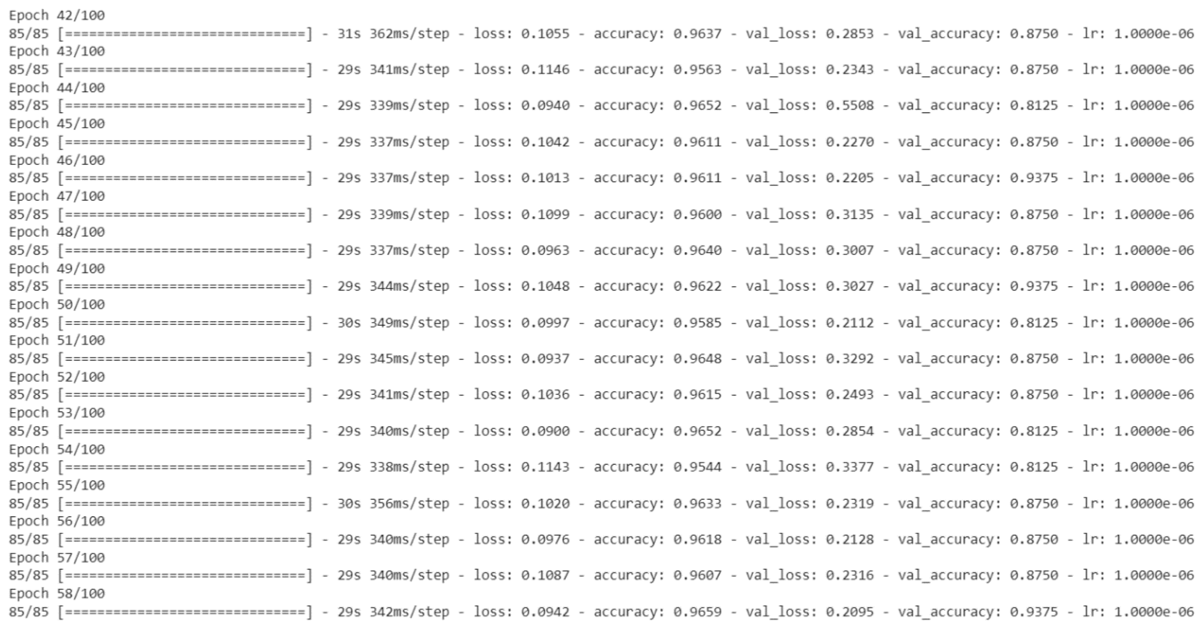


```
Epoch 13: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
85/85 [==============================] - 29s 341ms/step - loss: 0.1210 - accuracy: 0.9496 - val_loss: 0.5965 - val_accuracy: 0.6875 - lr: 9.0000e-05
```

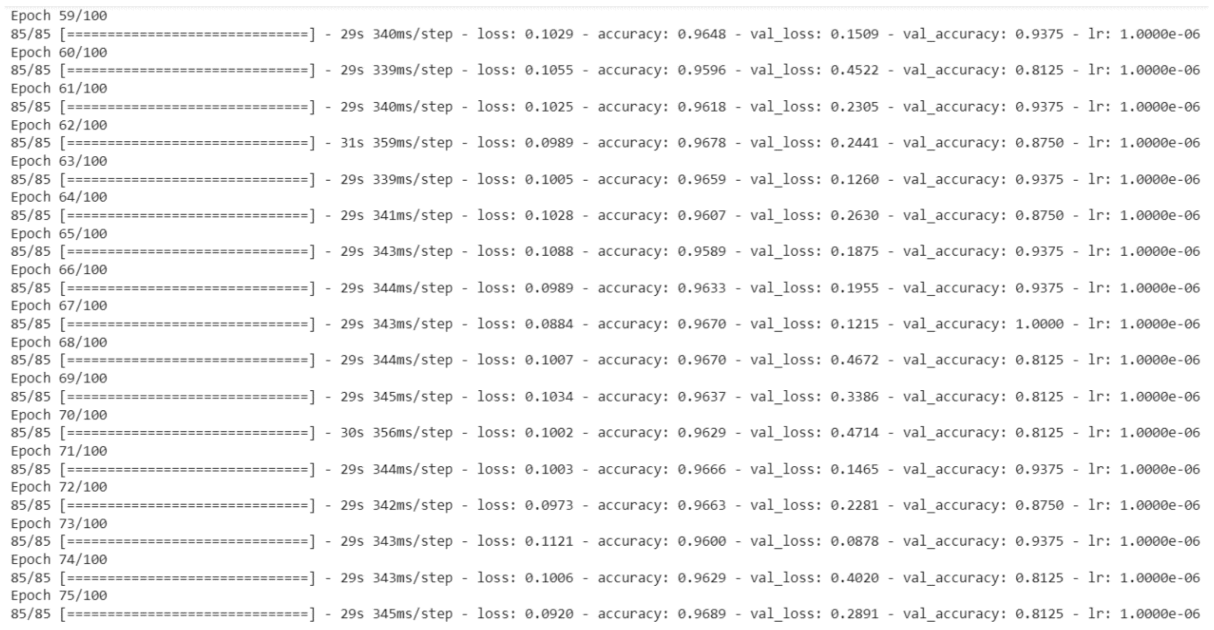
```
Epoch 14/100
85/85 [==============================] - 31s 361ms/step - loss: 0.1090 - accuracy: 0.9585 - val_loss: 0.2515 - val_accuracy: 0.8750 - lr: 2.7000e-05
Epoch 15/100
85/85 [==============================] - ETA: 0s - loss: 0.1153 - accuracy: 0.9577
Epoch 15: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
85/85 [==============================] - 29s 342ms/step - loss: 0.1153 - accuracy: 0.9577 - val_loss: 0.2384 - val_accuracy: 0.8750 - lr: 2.7000e-05
Epoch 16/100
85/85 [==============================] - 30s 357ms/step - loss: 0.1141 - accuracy: 0.9600 - val_loss: 0.2712 - val_accuracy: 0.8125 - lr: 8.1000e-06
Epoch 17/100
85/85 [==============================] - ETA: 0s - loss: 0.1011 - accuracy: 0.9607
Epoch 17: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
85/85 [==============================] - 29s 342ms/step - loss: 0.1011 - accuracy: 0.9607 - val_loss: 0.3473 - val_accuracy: 0.8125 - lr: 8.1000e-06
Epoch 18/100
85/85 [==============================] - 29s 341ms/step - loss: 0.1028 - accuracy: 0.9637 - val_loss: 0.2844 - val_accuracy: 0.8750 - lr: 2.4300e-06
Epoch 19/100
85/85 [==============================] - ETA: 0s - loss: 0.1022 - accuracy: 0.9618
Epoch 19: ReduceLROnPlateau reducing learning rate to 1e-06.
85/85 [==============================] - 29s 342ms/step - loss: 0.1022 - accuracy: 0.9618 - val_loss: 0.4549 - val_accuracy: 0.7500 - lr: 2.4300e-06
Epoch 20/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1061 - accuracy: 0.9640 - val_loss: 0.2014 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 21/100
85/85 [==============================] - 29s 342ms/step - loss: 0.1150 - accuracy: 0.9577 - val_loss: 0.2481 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 22/100
85/85 [==============================] - 29s 342ms/step - loss: 0.1066 - accuracy: 0.9577 - val_loss: 0.4309 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 23/100
85/85 [==============================] - 29s 343ms/step - loss: 0.0970 - accuracy: 0.9596 - val_loss: 0.3156 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 24/100
85/85 [==============================] - 31s 356ms/step - loss: 0.0973 - accuracy: 0.9652 - val_loss: 0.2359 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 25/100
85/85 [==============================] - 29s 341ms/step - loss: 0.1020 - accuracy: 0.9648 - val_loss: 0.2474 - val_accuracy: 0.8750 - lr: 1.0000e-06
```



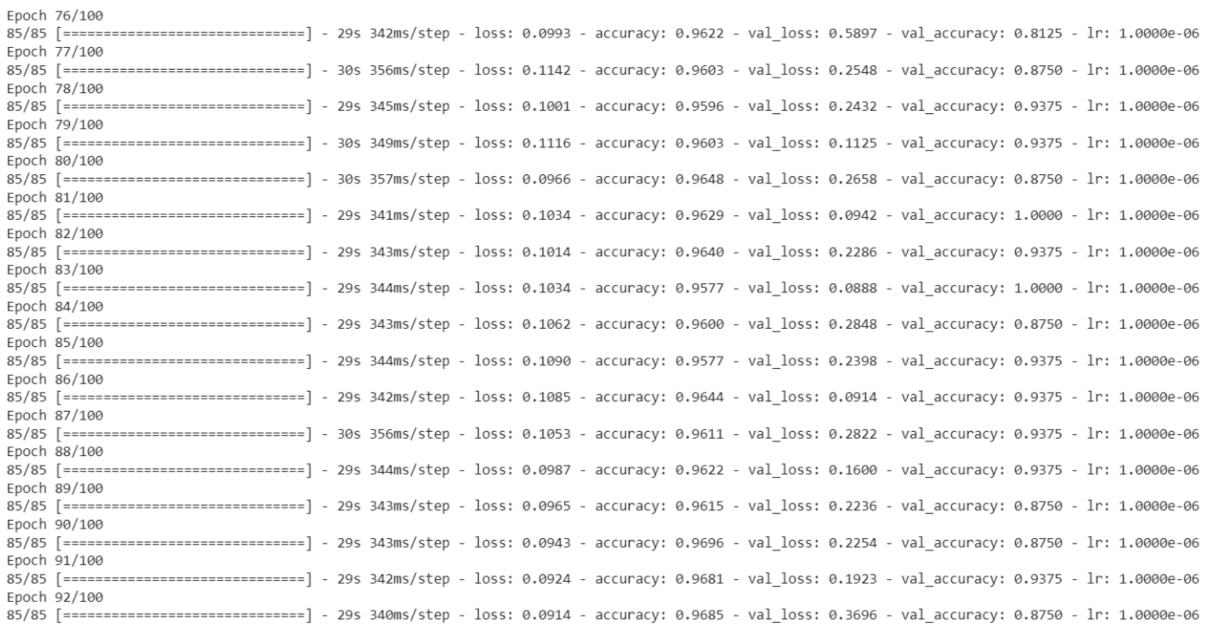
```
Epoch 26/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1046 - accuracy: 0.9644 - val_loss: 0.2170 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 27/100
85/85 [==============================] - 29s 345ms/step - loss: 0.1074 - accuracy: 0.9637 - val_loss: 0.4684 - val_accuracy: 0.7500 - lr: 1.0000e-06
Epoch 28/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1026 - accuracy: 0.9644 - val_loss: 0.3244 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 29/100
85/85 [==============================] - 29s 342ms/step - loss: 0.0981 - accuracy: 0.9622 - val_loss: 0.2254 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 30/100
85/85 [==============================] - 29s 343ms/step - loss: 0.0946 - accuracy: 0.9626 - val_loss: 0.5023 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 31/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1037 - accuracy: 0.9644 - val_loss: 0.5168 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 32/100
85/85 [==============================] - 31s 360ms/step - loss: 0.0989 - accuracy: 0.9652 - val_loss: 0.4700 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 33/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1049 - accuracy: 0.9626 - val_loss: 0.1920 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 34/100
85/85 [==============================] - 29s 342ms/step - loss: 0.0961 - accuracy: 0.9652 - val_loss: 0.2626 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 35/100
85/85 [==============================] - 31s 361ms/step - loss: 0.1111 - accuracy: 0.9581 - val_loss: 0.4076 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 36/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1018 - accuracy: 0.9666 - val_loss: 0.1156 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 37/100
85/85 [==============================] - 29s 342ms/step - loss: 0.1063 - accuracy: 0.9592 - val_loss: 0.1725 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 38/100
85/85 [==============================] - 29s 341ms/step - loss: 0.0975 - accuracy: 0.9637 - val_loss: 0.3652 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 39/100
85/85 [==============================] - 30s 349ms/step - loss: 0.0978 - accuracy: 0.9607 - val_loss: 0.5210 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 40/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1017 - accuracy: 0.9626 - val_loss: 0.3036 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 41/100
85/85 [==============================] - 29s 340ms/step - loss: 0.0952 - accuracy: 0.9652 - val_loss: 0.3522 - val_accuracy: 0.8750 - lr: 1.0000e-06
```



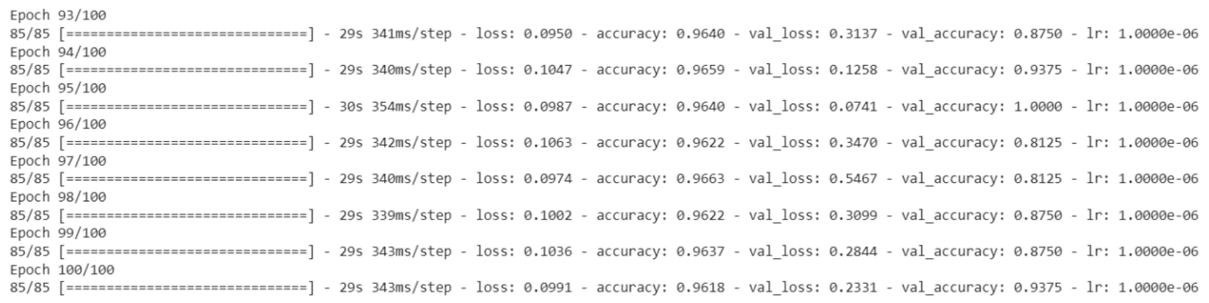
```
Epoch 42/100
85/85 [==============================] - 31s 362ms/step - loss: 0.1055 - accuracy: 0.9637 - val_loss: 0.2853 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 43/100
85/85 [==============================] - 29s 341ms/step - loss: 0.1146 - accuracy: 0.9563 - val_loss: 0.2343 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 44/100
85/85 [==============================] - 29s 339ms/step - loss: 0.0940 - accuracy: 0.9652 - val_loss: 0.5508 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 45/100
85/85 [==============================] - 29s 337ms/step - loss: 0.1042 - accuracy: 0.9611 - val_loss: 0.2270 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 46/100
85/85 [==============================] - 29s 337ms/step - loss: 0.1013 - accuracy: 0.9611 - val_loss: 0.2205 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 47/100
85/85 [==============================] - 29s 339ms/step - loss: 0.1099 - accuracy: 0.9600 - val_loss: 0.3135 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 48/100
85/85 [==============================] - 29s 337ms/step - loss: 0.0963 - accuracy: 0.9640 - val_loss: 0.3007 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 49/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1048 - accuracy: 0.9622 - val_loss: 0.3027 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 50/100
85/85 [==============================] - 30s 349ms/step - loss: 0.0997 - accuracy: 0.9585 - val_loss: 0.2112 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 51/100
85/85 [==============================] - 29s 345ms/step - loss: 0.0937 - accuracy: 0.9648 - val_loss: 0.3292 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 52/100
85/85 [==============================] - 29s 341ms/step - loss: 0.1036 - accuracy: 0.9615 - val_loss: 0.2493 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 53/100
85/85 [==============================] - 29s 340ms/step - loss: 0.0900 - accuracy: 0.9652 - val_loss: 0.2854 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 54/100
85/85 [==============================] - 29s 338ms/step - loss: 0.1143 - accuracy: 0.9544 - val_loss: 0.3377 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 55/100
85/85 [==============================] - 30s 356ms/step - loss: 0.1020 - accuracy: 0.9633 - val_loss: 0.2319 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 56/100
85/85 [==============================] - 29s 340ms/step - loss: 0.0976 - accuracy: 0.9618 - val_loss: 0.2128 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 57/100
85/85 [==============================] - 29s 340ms/step - loss: 0.1087 - accuracy: 0.9607 - val_loss: 0.2316 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 58/100
85/85 [==============================] - 29s 342ms/step - loss: 0.0942 - accuracy: 0.9659 - val_loss: 0.2095 - val_accuracy: 0.9375 - lr: 1.0000e-06
```



```
Epoch 59/100
85/85 [==============================] - 29s 340ms/step - loss: 0.1029 - accuracy: 0.9648 - val_loss: 0.1509 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 60/100
85/85 [==============================] - 29s 339ms/step - loss: 0.1055 - accuracy: 0.9596 - val_loss: 0.4522 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 61/100
85/85 [==============================] - 29s 340ms/step - loss: 0.1025 - accuracy: 0.9618 - val_loss: 0.2305 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 62/100
85/85 [==============================] - 31s 359ms/step - loss: 0.0989 - accuracy: 0.9678 - val_loss: 0.2441 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 63/100
85/85 [==============================] - 29s 339ms/step - loss: 0.1005 - accuracy: 0.9659 - val_loss: 0.1260 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 64/100
85/85 [==============================] - 29s 341ms/step - loss: 0.1028 - accuracy: 0.9607 - val_loss: 0.2630 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 65/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1088 - accuracy: 0.9589 - val_loss: 0.1875 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 66/100
85/85 [==============================] - 29s 344ms/step - loss: 0.0989 - accuracy: 0.9633 - val_loss: 0.1955 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 67/100
85/85 [==============================] - 29s 343ms/step - loss: 0.0884 - accuracy: 0.9670 - val_loss: 0.1215 - val_accuracy: 1.0000 - lr: 1.0000e-06
Epoch 68/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1007 - accuracy: 0.9670 - val_loss: 0.4672 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 69/100
85/85 [==============================] - 29s 345ms/step - loss: 0.1034 - accuracy: 0.9637 - val_loss: 0.3386 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 70/100
85/85 [==============================] - 30s 356ms/step - loss: 0.1002 - accuracy: 0.9629 - val_loss: 0.4714 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 71/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1003 - accuracy: 0.9666 - val_loss: 0.1465 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 72/100
85/85 [==============================] - 29s 342ms/step - loss: 0.0973 - accuracy: 0.9663 - val_loss: 0.2281 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 73/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1121 - accuracy: 0.9600 - val_loss: 0.0878 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 74/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1006 - accuracy: 0.9629 - val_loss: 0.4020 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 75/100
85/85 [==============================] - 29s 345ms/step - loss: 0.0920 - accuracy: 0.9689 - val_loss: 0.2891 - val_accuracy: 0.8125 - lr: 1.0000e-06
```



```
Epoch 76/100
85/85 [==============================] - 29s 342ms/step - loss: 0.0993 - accuracy: 0.9622 - val_loss: 0.5897 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 77/100
85/85 [==============================] - 30s 356ms/step - loss: 0.1142 - accuracy: 0.9603 - val_loss: 0.2548 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 78/100
85/85 [==============================] - 29s 345ms/step - loss: 0.1001 - accuracy: 0.9596 - val_loss: 0.2432 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 79/100
85/85 [==============================] - 30s 349ms/step - loss: 0.1116 - accuracy: 0.9603 - val_loss: 0.1125 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 80/100
85/85 [==============================] - 30s 357ms/step - loss: 0.0966 - accuracy: 0.9648 - val_loss: 0.2658 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 81/100
85/85 [==============================] - 29s 341ms/step - loss: 0.1034 - accuracy: 0.9629 - val_loss: 0.0942 - val_accuracy: 1.0000 - lr: 1.0000e-06
Epoch 82/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1014 - accuracy: 0.9640 - val_loss: 0.2286 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 83/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1034 - accuracy: 0.9577 - val_loss: 0.0888 - val_accuracy: 1.0000 - lr: 1.0000e-06
Epoch 84/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1062 - accuracy: 0.9600 - val_loss: 0.2848 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 85/100
85/85 [==============================] - 29s 344ms/step - loss: 0.1090 - accuracy: 0.9577 - val_loss: 0.2398 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 86/100
85/85 [==============================] - 29s 342ms/step - loss: 0.1085 - accuracy: 0.9644 - val_loss: 0.0914 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 87/100
85/85 [==============================] - 30s 356ms/step - loss: 0.1053 - accuracy: 0.9611 - val_loss: 0.2822 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 88/100
85/85 [==============================] - 29s 344ms/step - loss: 0.0987 - accuracy: 0.9622 - val_loss: 0.1600 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 89/100
85/85 [==============================] - 29s 343ms/step - loss: 0.0965 - accuracy: 0.9615 - val_loss: 0.2236 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 90/100
85/85 [==============================] - 29s 343ms/step - loss: 0.0943 - accuracy: 0.9696 - val_loss: 0.2254 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 91/100
85/85 [==============================] - 29s 342ms/step - loss: 0.0924 - accuracy: 0.9681 - val_loss: 0.1923 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 92/100
85/85 [==============================] - 29s 340ms/step - loss: 0.0914 - accuracy: 0.9685 - val_loss: 0.3696 - val_accuracy: 0.8750 - lr: 1.0000e-06
```



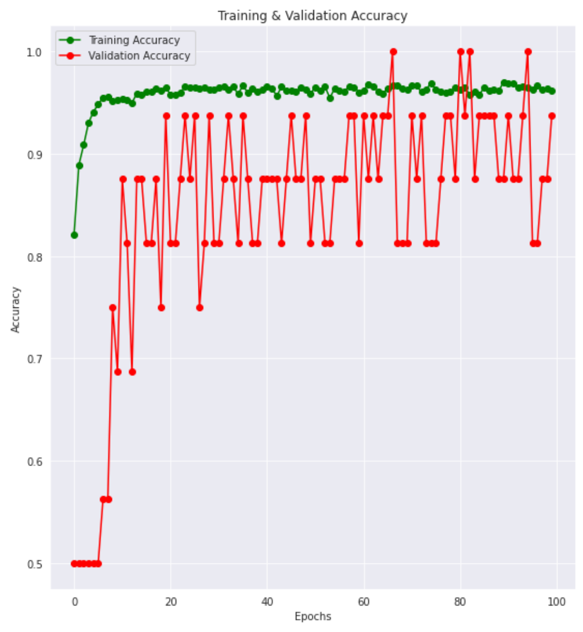
```
Epoch 93/100
85/85 [==============================] - 29s 341ms/step - loss: 0.0950 - accuracy: 0.9640 - val_loss: 0.3137 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 94/100
85/85 [==============================] - 29s 340ms/step - loss: 0.1047 - accuracy: 0.9659 - val_loss: 0.1258 - val_accuracy: 0.9375 - lr: 1.0000e-06
Epoch 95/100
85/85 [==============================] - 30s 354ms/step - loss: 0.0987 - accuracy: 0.9640 - val_loss: 0.0741 - val_accuracy: 1.0000 - lr: 1.0000e-06
Epoch 96/100
85/85 [==============================] - 29s 342ms/step - loss: 0.1063 - accuracy: 0.9622 - val_loss: 0.3470 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 97/100
85/85 [==============================] - 29s 340ms/step - loss: 0.0974 - accuracy: 0.9663 - val_loss: 0.5467 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 98/100
85/85 [==============================] - 29s 339ms/step - loss: 0.1002 - accuracy: 0.9622 - val_loss: 0.3099 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 99/100
85/85 [==============================] - 29s 343ms/step - loss: 0.1036 - accuracy: 0.9637 - val_loss: 0.2844 - val_accuracy: 0.8750 - lr: 1.0000e-06
Epoch 100/100
85/85 [==============================] - 29s 343ms/step - loss: 0.0991 - accuracy: 0.9618 - val_loss: 0.2331 - val_accuracy: 0.9375 - lr: 1.0000e-06
```

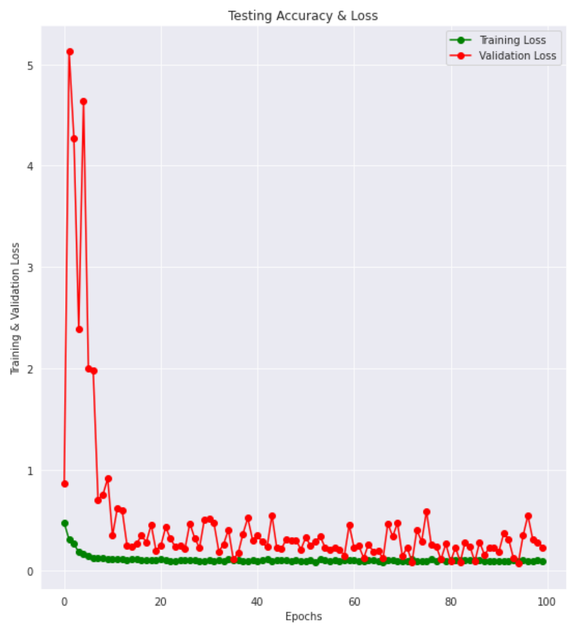
Final Result

```
20/20 [==============================] - 2s 64ms/step - loss: 0.2482 - accuracy: 0.9057
Loss of the model is -  0.24820896983146667
20/20 [==============================] - 1s 31ms/step - loss: 0.2482 - accuracy: 0.9057
Accuracy of the model is -  90.56603908538818 %
```

The final accuracy of the model after training is 90.56 percent.

Training and Validation Accuracy & Testing Loss per Epoch





Classification Report

Classification Report is a tool in Sklearn Python. It calculates f score, accuracy and precision after testing the model which are useful to evaluate whether the model is overfitting the data.

A classification report consists of the following parameters:

Precision: precision is the number of true positive results divided by the number of all positive results, including those not identified correctly.

Precision = TP / (TP + FP)

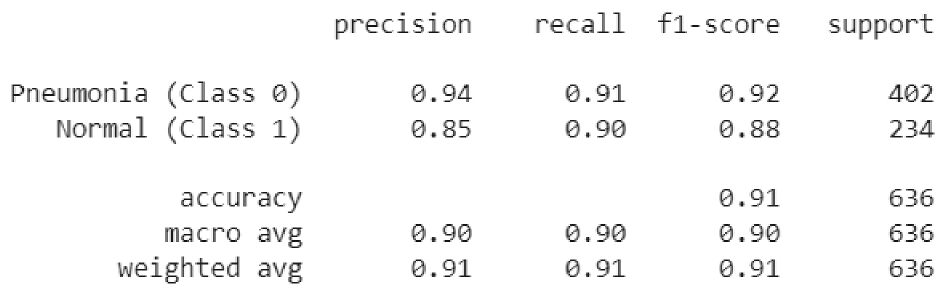
Recall: Recall is the number of true positive results divided by the number of all samples that should have been identified as positive.

Recall = TP / (TP + FN)

F1 score: It is the harmonic mean of precision and recall. The best possible score is 1 and the worst is 0. It is used to compare the performance of any 2 models.

f1 score = (2 * (Recall * Precision) / (Recall + Precision))

Support: Specifies the number of actual occurrences of class in the given dataset.



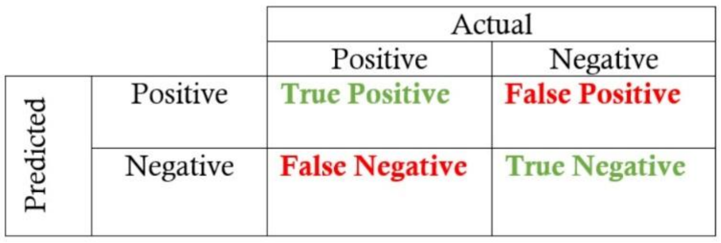
```
                    precision    recall  f1-score   support

Pneumonia (Class 0)      0.94      0.91      0.92       402
   Normal (Class 1)      0.85      0.90      0.88       234

           accuracy                          0.91       636
          macro avg      0.90      0.90      0.90       636
       weighted avg      0.91      0.91      0.91       636
```

Confusion Matrix

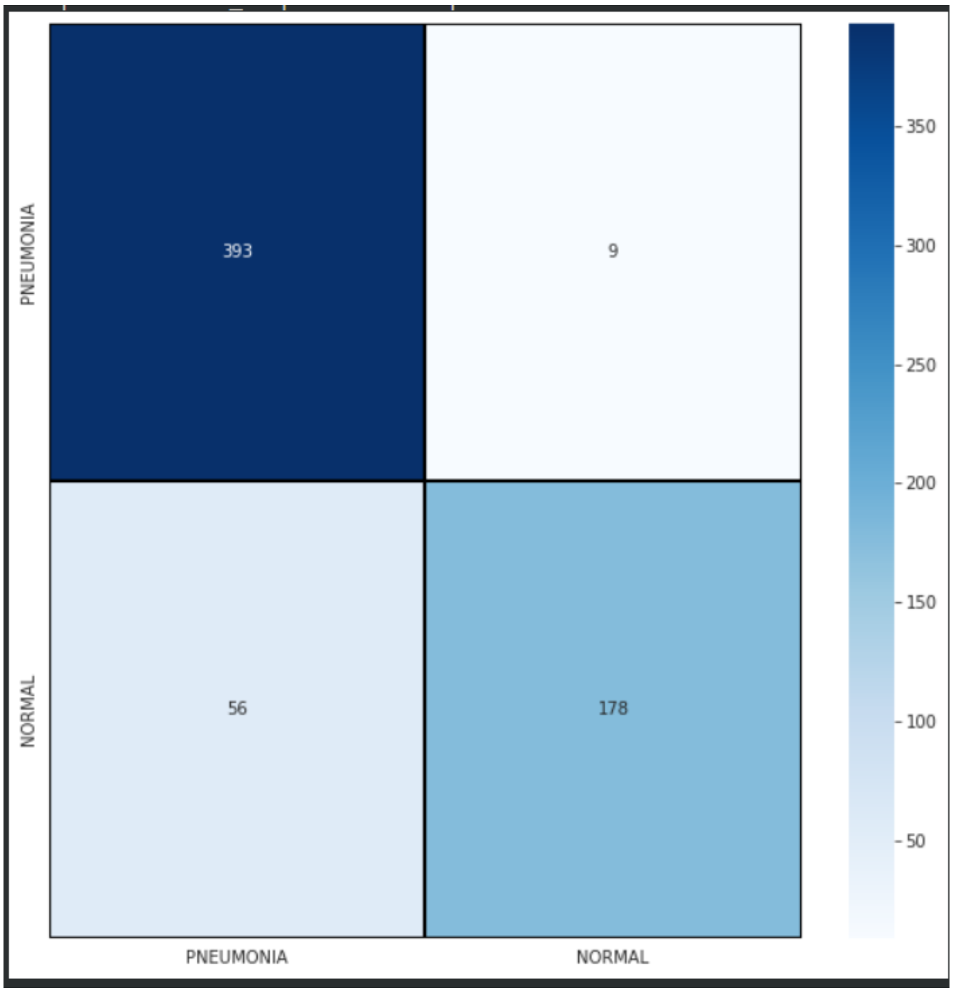
Confusion matrices are a widely used measurement when attempting to solve classification issues. Confusion matrices show counts between expected and observed values. The result "TN" stands for True Negative and displays the number of negatively classed cases that were correctly identified. Similar to this, "TP" stands for True Positive and denotes the quantity of correctly identified positive cases. The terms "FP" and "FN" stand for false positive and false negative values, respectively, and denote the number of real negative cases categorized as positive and real positive examples classified as negative, respectively. [12]



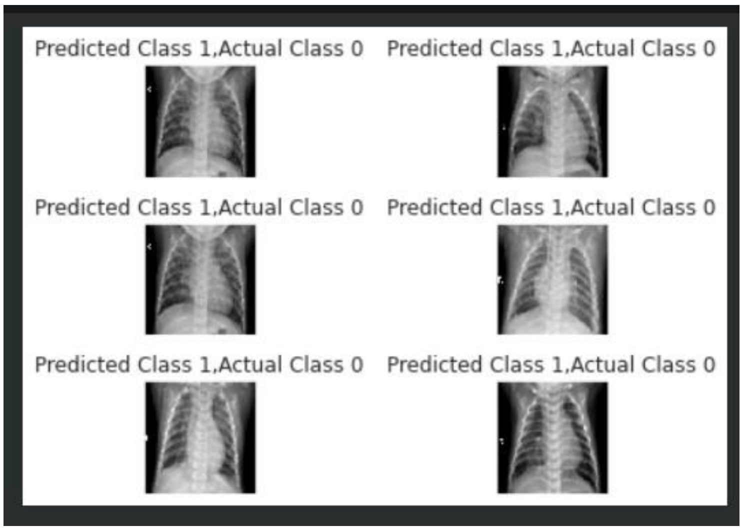
		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

The accuracy can be calculated from confusion matrix as follows

$$\text{Accuracy} = \frac{\text{TN}+\text{TP}}{\text{TN}+\text{FP}+\text{FN}+\text{TP}}$$



Wrongly Predicted Test Samples


Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0
Predicted Class 1,Actual Class 0

V. Conclusion

It is necessary that a properly trained radiologist is present when identifying thoracic diseases. However, the development of algorithms in this particular domain can be helpful in the identification of these diseases in remote areas. At present, not much work has been done in the identification of these kinds of diseases. In the future, if more accurate algorithms can be developed, one can easily classify the type of disease a person has within seconds using the chest X-rays.

VI. References

[1] "What is Pneumonia?," [Online]. Available: https://www.nhlbi.nih.gov/health/pneumonia.

[2] "Everything You Need to Know About Pneumonia," [Online]. Available: https://www.healthline.com/health/pneumonia.

[3] "Introduction to Matplotlib," [Online]. Available: https://www.geeksforgeeks.org/python-introduction-matplotlib/.

[4] "Introduction to Seaborn – Python," [Online]. Available: https://www.geeksforgeeks.org/introduction-to-seaborn-python/.

[5] IBM Cloud Education, "Convolutional Neural Networks," [Online]. Available: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[6] "A Gentle Introduction to the Rectified Linear Unit (ReLU)," [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[7] "Activation Functions in Neural Networks," [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[8] "Why is Batch Normalization useful in Deep Neural Networks?," [Online]. Available: https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf.

[9] "A Look at Gradient Descent and RMSprop Optimizers," [Online]. Available: https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b.

[10] "Understanding binary cross-entropy / log loss: a visual explanation," [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[11] "A Gentle Introduction to Transfer Learning for Deep Learning," [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

[12] "Confusion Matrix," [Online]. Available: https://www.sciencedirect.com/topics/engineering/confusion-matrix.

[13] "Keras API reference," [Online]. Available: https://keras.io/api/.