# PNEUMONIA DETECTION USING CNN

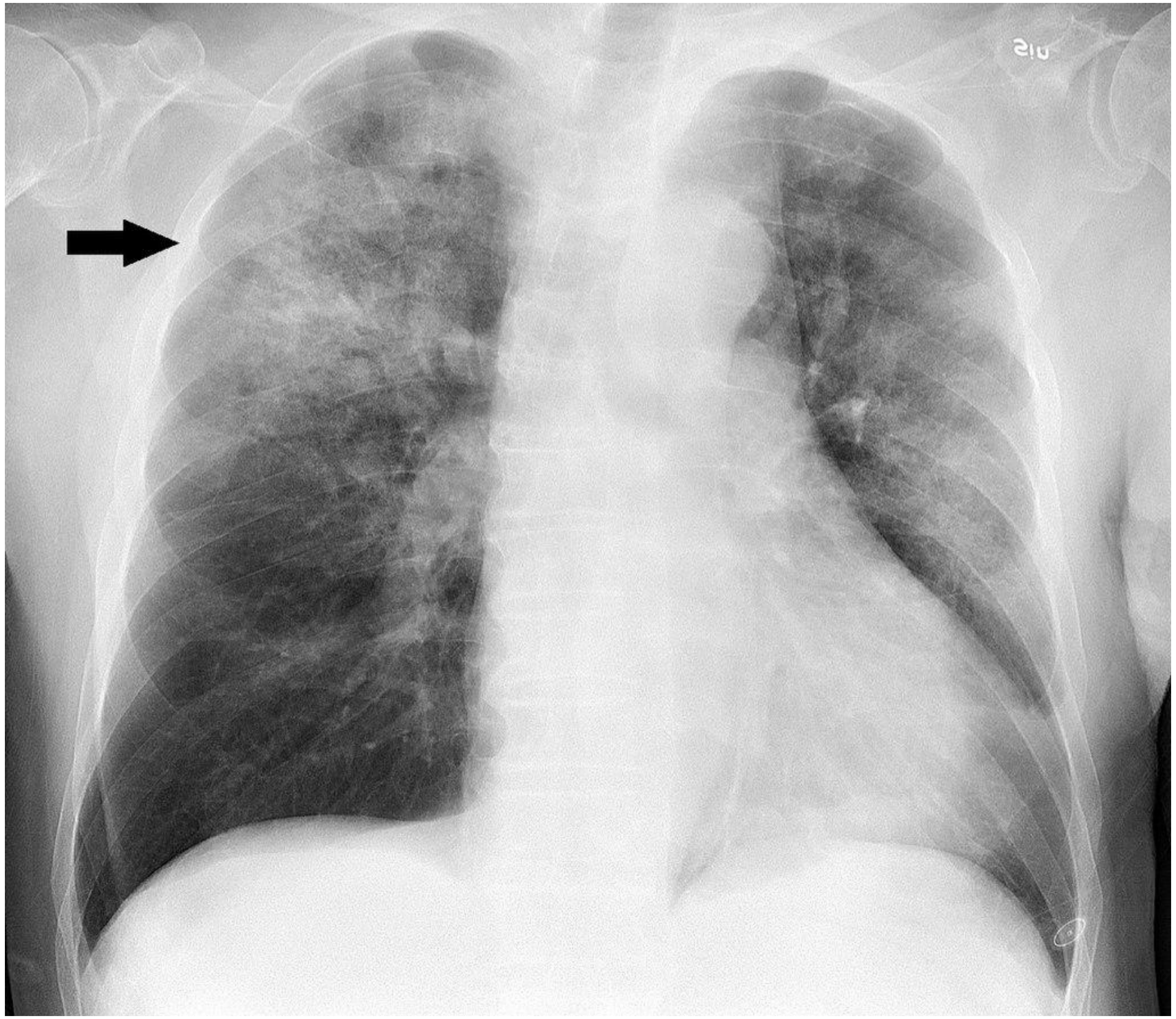
Challa Gopala Krishna
(21ECB0B09)
Kaveti Shrikar
(21ECB0B21)

# WHAT IS PNEUMONIA?

- Pneumonia is a Inflammatory condition of lung primarily affecting the small air sacs known as alveoli
- Symptoms typically include some combination of productive or dry cough, chest pain, fever, and difficulty breathing. The severity of the condition is variable.
- Pneumonia is usually caused by infection with viruses or bacteria, and less commonly by other microorganisms.
- Identifying the responsible pathogen can be difficult. Diagnosis is often based on symptoms and physical examination. Chest X-rays, blood tests, and culture of the sputum may help confirm the diagnosis.
- Each year, pneumonia affects about 450 million people globally (7% of the population) and results in about 4 million deaths.

# DETECTION OF PNEUMONIA USING CHEST X-RAY

- The Image on the Right Side Shows 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 Haemophiles influenzae. It shows multifocal, patchy consolidation, mainly in the right upper lobe.
- X-Rays can are used to understand severity of the disease but analysis of Chest X-rays is complex and can only be done by experienced Radiologists



# REFERENCE

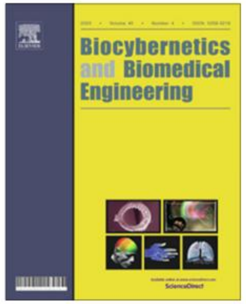
(HTTPS://WWW.SCIENCEDIRECT.COM/SCIENCE/ARTICLE/PII/S0208521622000742#:~:TEXT=THEY%20OBTAINED%20AN%20ACCURACY%20OF,OBTAINED%20A%2092.3%25%20AVERAGED%20SCORE.)

ELSEVIER

Available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/bbe




Original Research Article

## Detection of pneumonia using convolutional neural networks and deep learning

Check for updates

*Patrik Szepesi* [a], *László Szilágyi* [b,c,d,*]

[a] *Corvinus University of Budapest, Budapest, Hungary*
[b] *Computational Intelligence Research Group (CIRG), Sapientia University of Transylvania, Tîrgu Mureş, Romania*
[c] *Physiological Controls Research Center, Óbuda University, Budapest, Hungary*
[d] *Biomatics and Applied Artificial Intelligence Institution, Óbuda University, Budapest, Hungary*

# ABOUT THE DATASET BEING USED

**Content** (https://data.mendeley.com/datasets/rscbjbr9sj/2)

The dataset is organized into 3 folders (train, test, Val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

# ALGORITHM

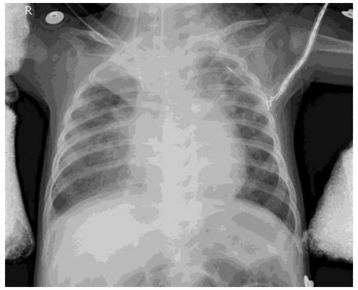
Librares used

- matplotlib.pyplot
- seaborn
- TensorFlow
- Keras
  - model (Sequential)
  - layers (Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization)
  - pre - processing (ImageDataGenerator)
- callbacks (ReduceLROnPlateau)
- Sklearn
  - Sklearn - metrics (classification report, confusion matrix)
- open-cv2 python
- os python

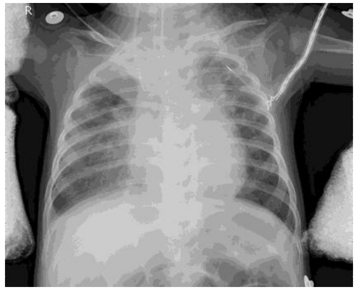
# PRE – PROCESSING

1. Reading 'Input' Directories and Reading all Image Files using open-cv and converting RGB to Grayscale Image
    1. Converting Image to Grayscale because Detecting of features in X-ray is best in Grayscale and as all X-rays are in Grayscale
2. Re – Shaping Image Data Array and Dividing Data into Training and Testing Data
    1. Splitting Input data into Test and Train data
3. Using ImageDataGenerator to randomly shift Image Pixels and Rotate Image
    1. Rotating makes sure that we get the exact result even if input image is tilted.
    2. Zoom range trains the model with different zoom levels.
    3. Horizontal flip- Makes sure x-ray is analysed irrespective of which side we keep.
4. The training data is then fed to this ImageDataGenerator function for data augmentation.

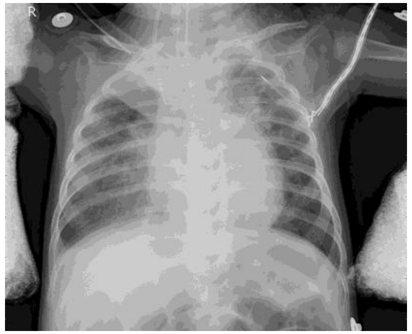
## Rotation of Image



## Image Zoom



## Horizontally Flipping Image



# LAYERS IN CNN

Total number of layers-22

The different layers that have been used are

1. Convolution layer (No of Filters = 32, 64, 128, 256, Filter Size(Kernel size) = 3x3, Activation function - ReLU)
2. Batch Normalisation
3. Maxpooling layers
4. Dropout
5. Flatten
6. Dense

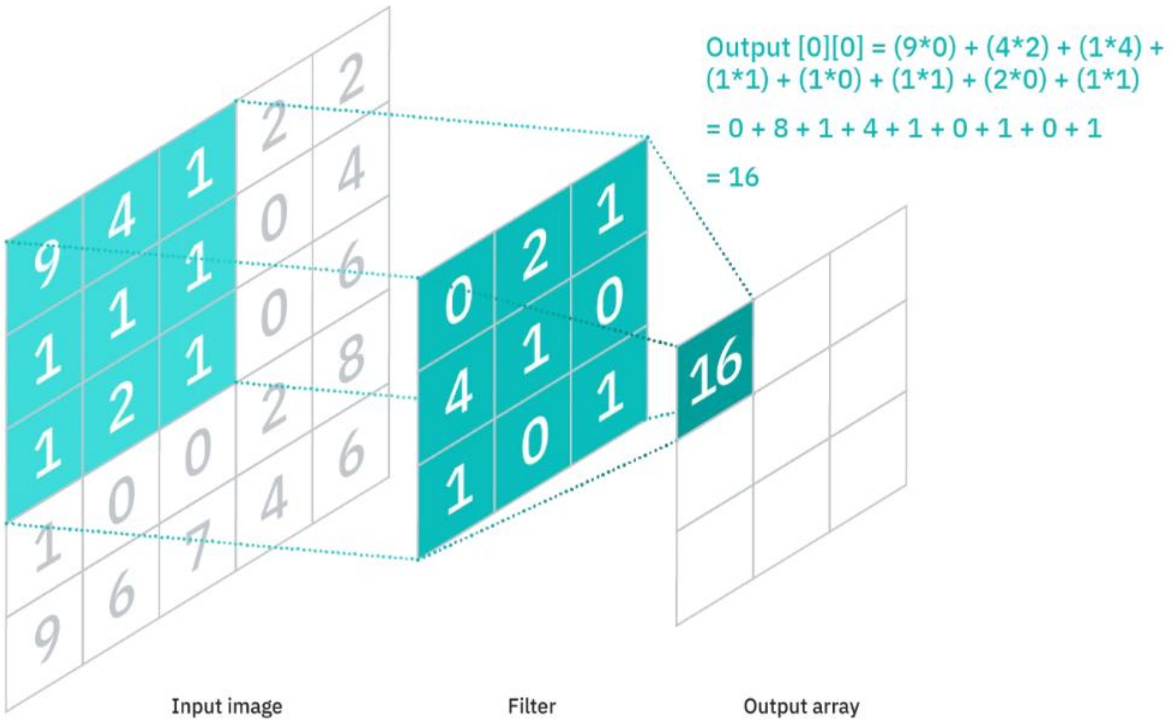
# LAYERS IN CNN

- No of Images used to Train Model – 5,262
- After Pre-processing and Data Augmentation, Final length of data - 5,262
- Batch size - 32 (No of samples processed before updating the model)
- No of epochs - 12
- No of samples trained in one epoch – 5,262

# 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 LAYER

<u>Convolution Layer parameters:</u>

1. Filters: Represent no of features to be extracted

2. Kernel Size: The kernel size refers to the width x height of the filter mask

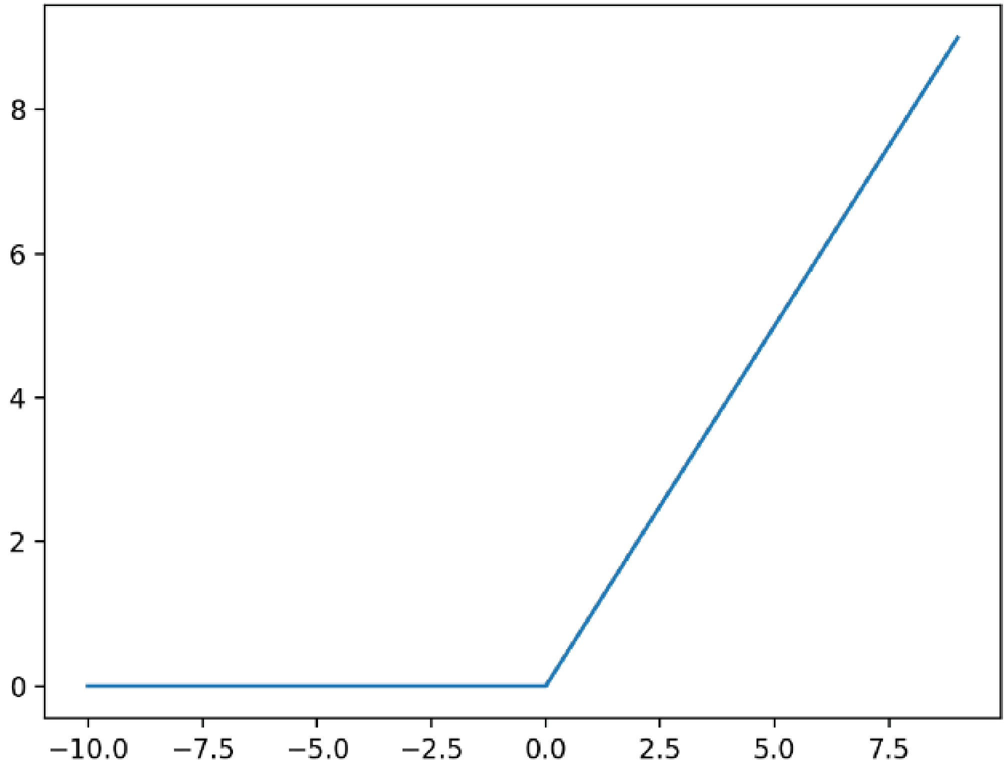
3. Strides: No of columns through which filter is moved

4. Padding: No. of Border pixels added to Image to Extract Edge pixel Characteristics

5. Activation function: ReLU

# 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.

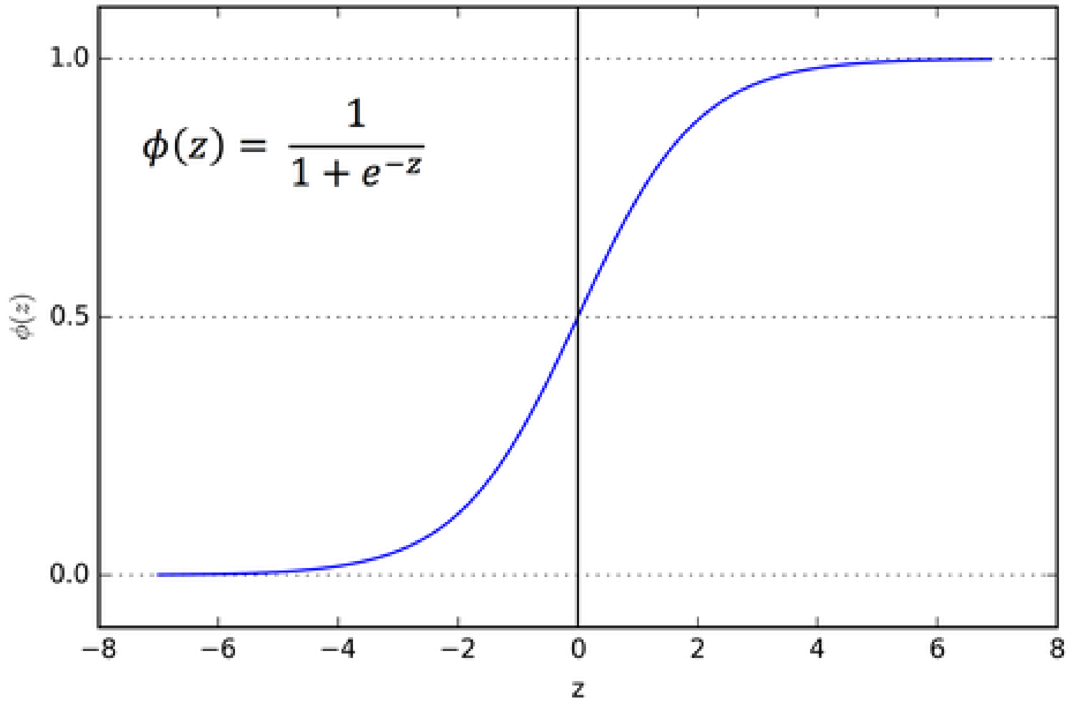


# SIGMOID ACTIVATION FUNCTION

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 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[1] and exactly one inflection point. A sigmoid "function" and a sigmoid "curve" refer to the same object.



# BATCH NORMALISATION

Deep neural networks with tens of layers can be sensitive to initial random weights and configuration of learning algorithm. The distribution of inputs to layers deep in network may change after each mini-batch when the weights are updated. This can cause the learning algorithm to forever chase a moving target. This can be prevented using batch normalization.

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

This method helps us to avoid overfitting and makes the learning process more efficient. A typical neural network is trained using a collected set of input data called batch. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

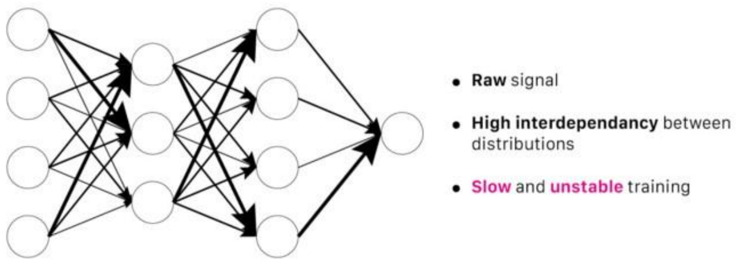
# BATCH NORMALIZATION

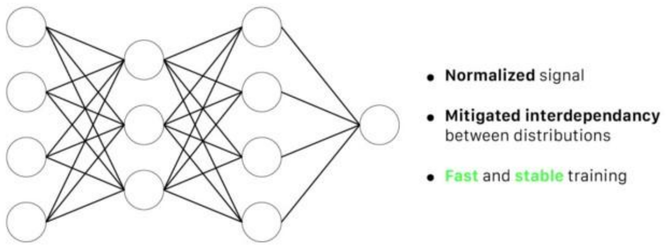
1. Batch Normalization Re-centres and Re-scales the Input layers.

2. Batch normalization transforms the Input Data and Re-Distributes the data to decrease Unstability

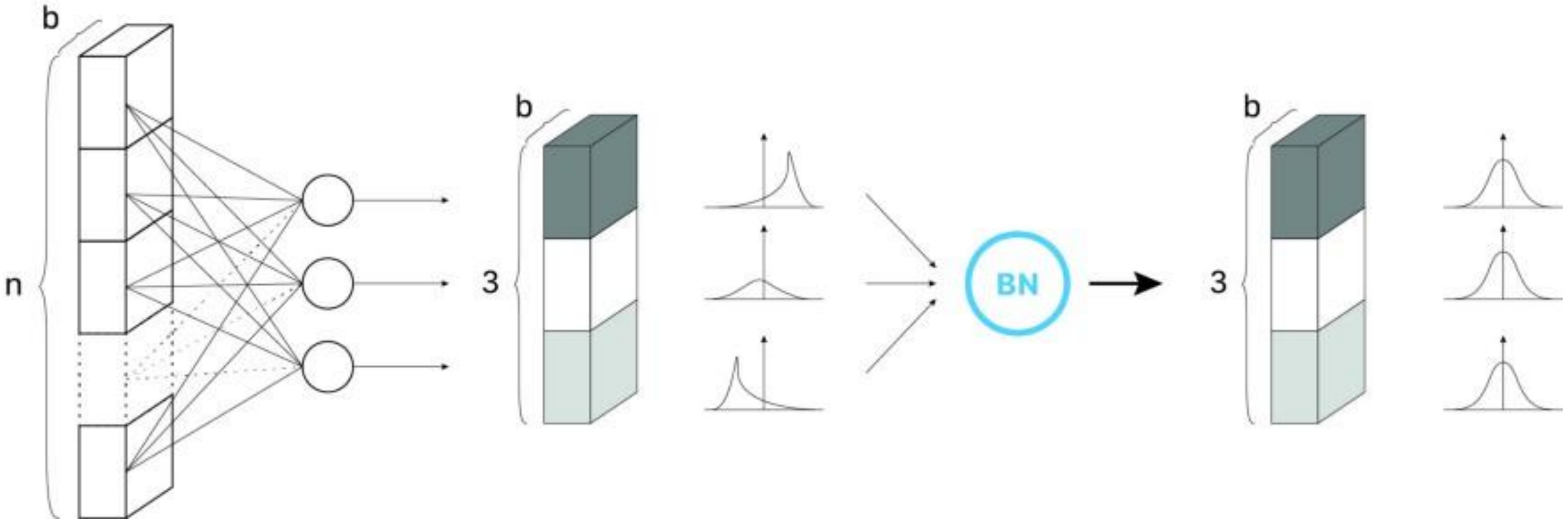
3. It changes the learning rate to optimum value to decrease losses during initialization.

4. Introduces Non-Linearity



# BATCH NORMALIZATION

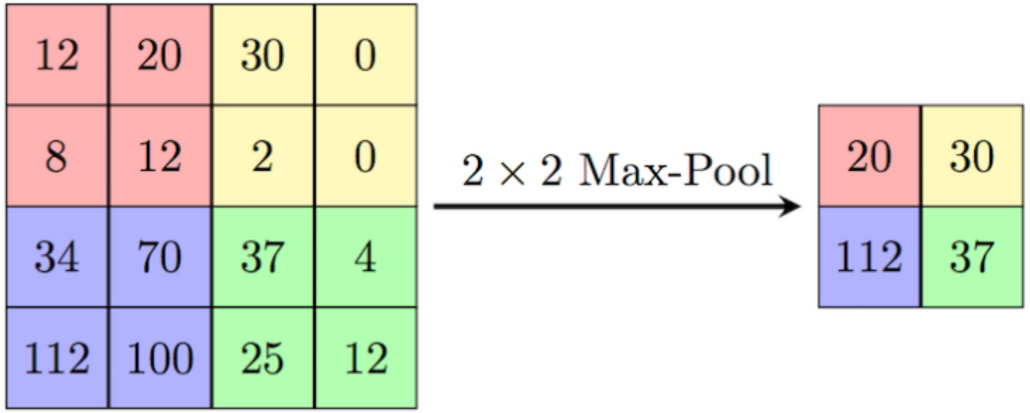



(1) $\mu = \frac{1}{n}\sum_i Z^{(i)}$ (2) $\sigma^2 = \frac{1}{n}\sum_i (Z^{(i)} - \mu)^2$ (3) $Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 - \epsilon}}$ (4) $\breve{Z} = \gamma * Z^{(i)}_{norm} + \beta$

# MAX POOLING

Max Pooling is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a down sampled (pooled) feature map. It is usually used after a convolutional layer.

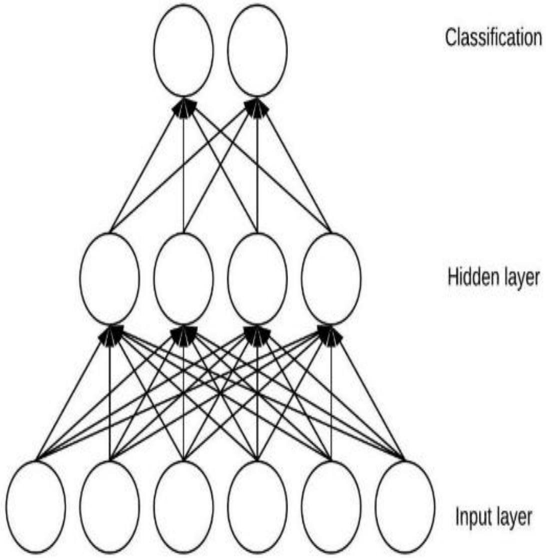
Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. It also makes the model more robust to variations in the position of the features in the input image.



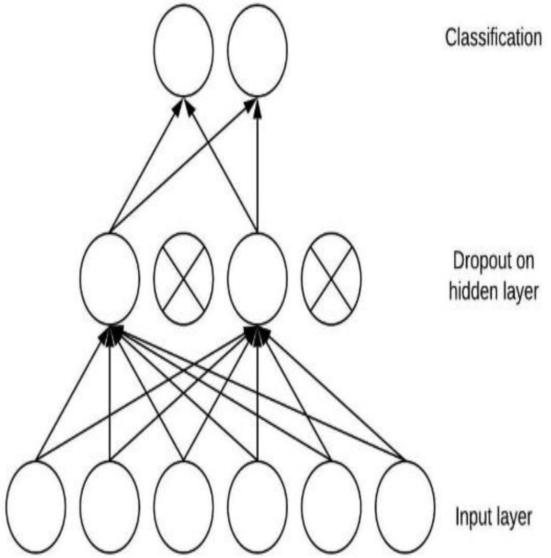
# DROPOUT

Dropout in machine learning refers to the process in which certain nodes are randomly ignored during training. This is done in order to reduce overfitting and improve generalisation error in deep neural networks.

- We used 0.1 and 0.2 rate of Frequency to set sample to Zero
- 1 / (1 – Rate) Samples are not set to zero and pass to next layer
- Sum of the Input will be the same and Re-Distributed




Without Dropout

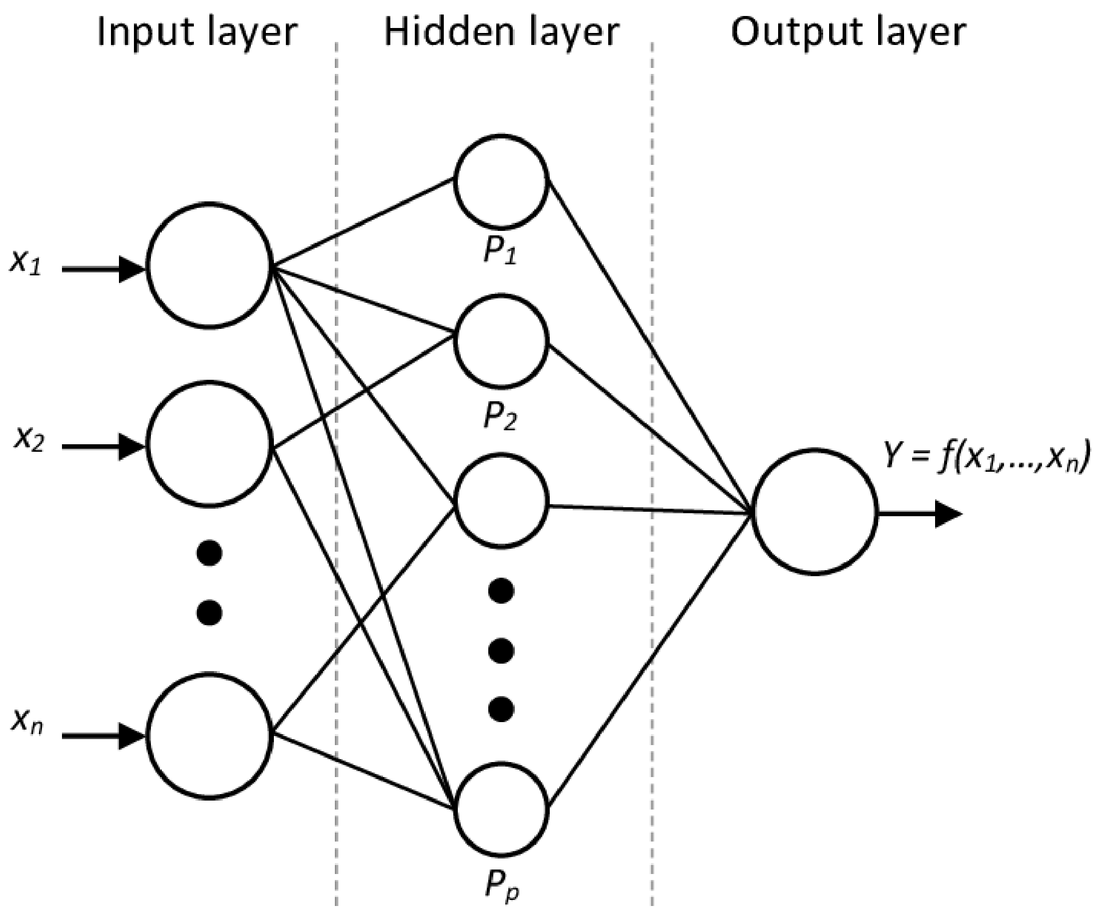



With Dropout

# DENSE (HIDDEN LAYERS)

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer.

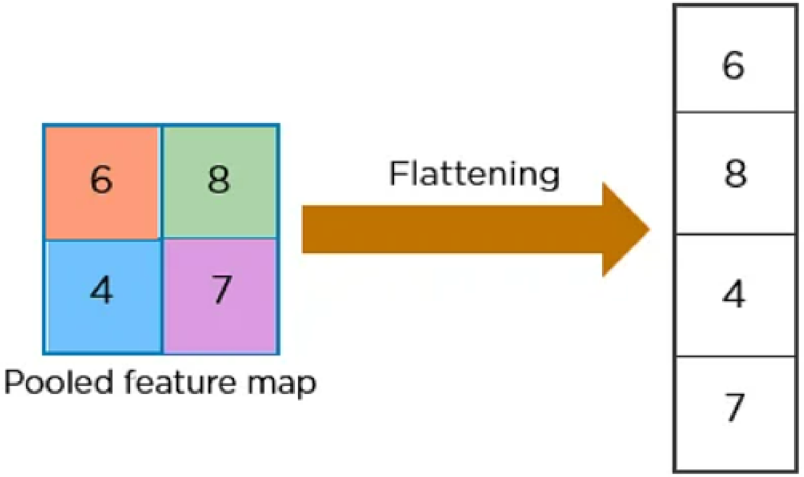
In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.



# FLATTEN LAYER

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

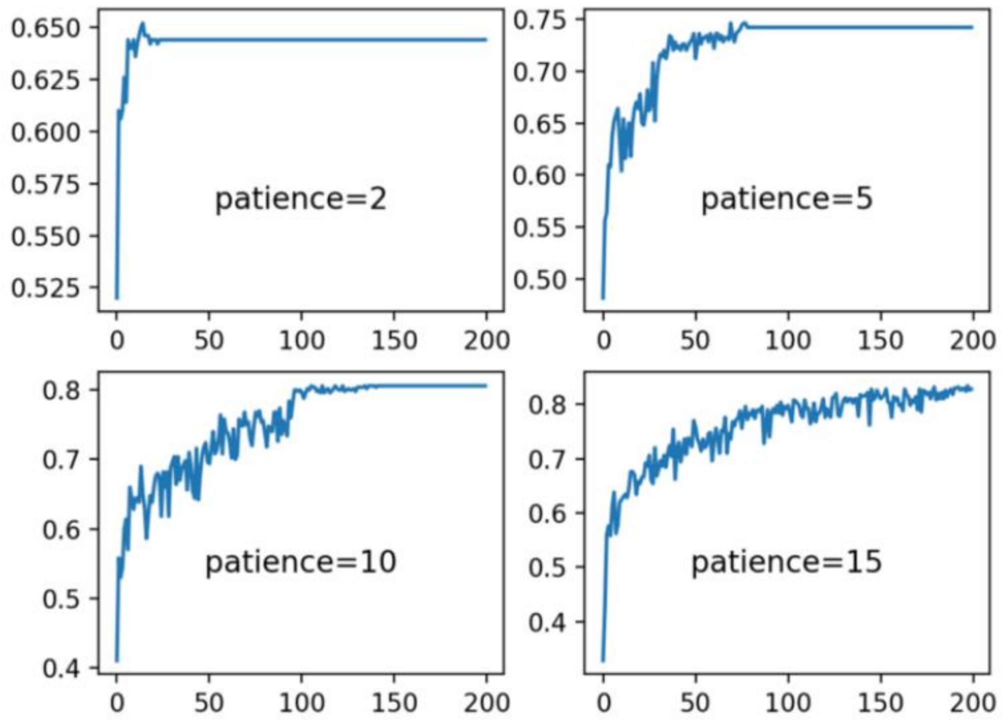
The flattened matrix is fed as input to the fully connected layer to classify the image.



# REDUCE LR ON PLATEAU

Reducelronplateau means to Reduce learning rate when a metric has stopped improving.

Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.



# COMPILING THE MODEL

After defining all the layers, we compile the model so that it can be used for training and testing. In order to compile, we use optimizers and loss functions.

Optimizers are algorithms or processes used to minimize error and maximize efficiency. This is done by changing the model's trainable parameters i.e., weights and biases.

A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs.

```
model.compile(optimizer = "rmsprop", loss = 'binary_crossentropy', metrics = ['accuracy'])
```

# OPTIMIZER

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 a extension of gradient decent 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 square root of sum of square of partial derivative.

Rmsprop is a extension of Adagrad with decaying Average

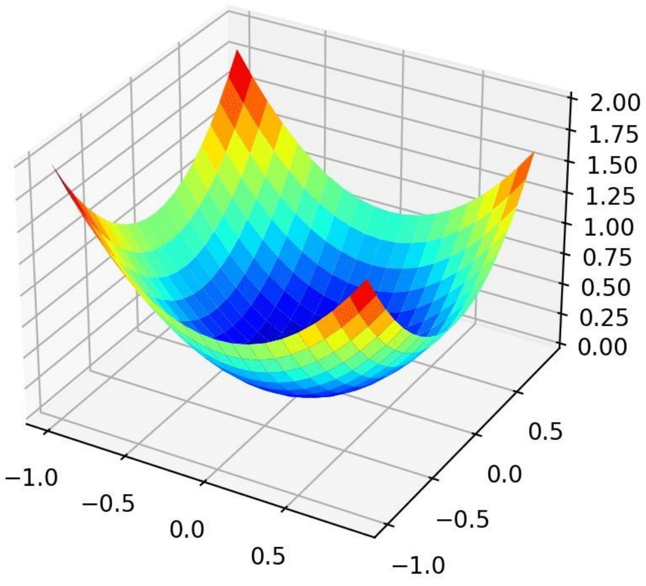
$$g_{t,i} = \nabla_\theta J(\theta_{t,i}),$$

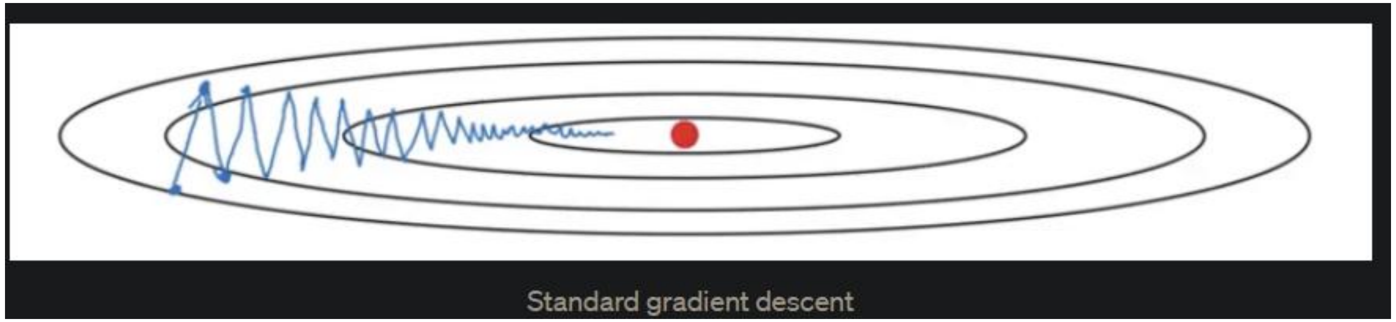
A derivative of loss function for given parameters at a given time t.

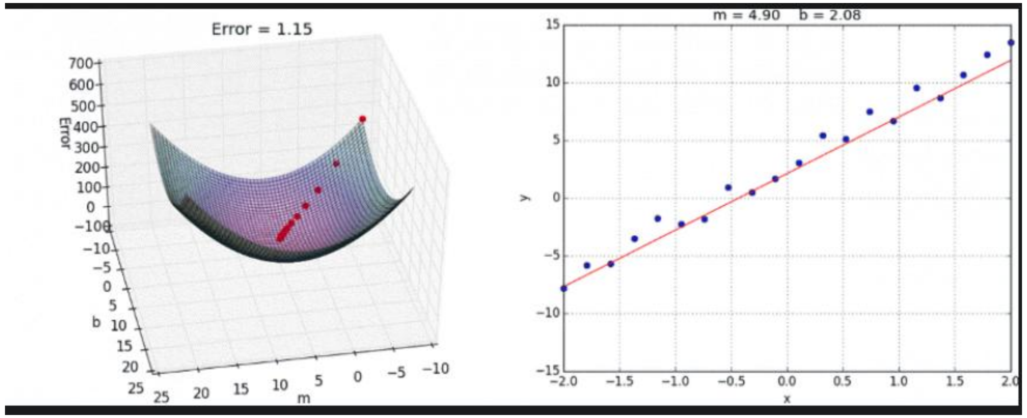
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

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

# OPTIMIZER

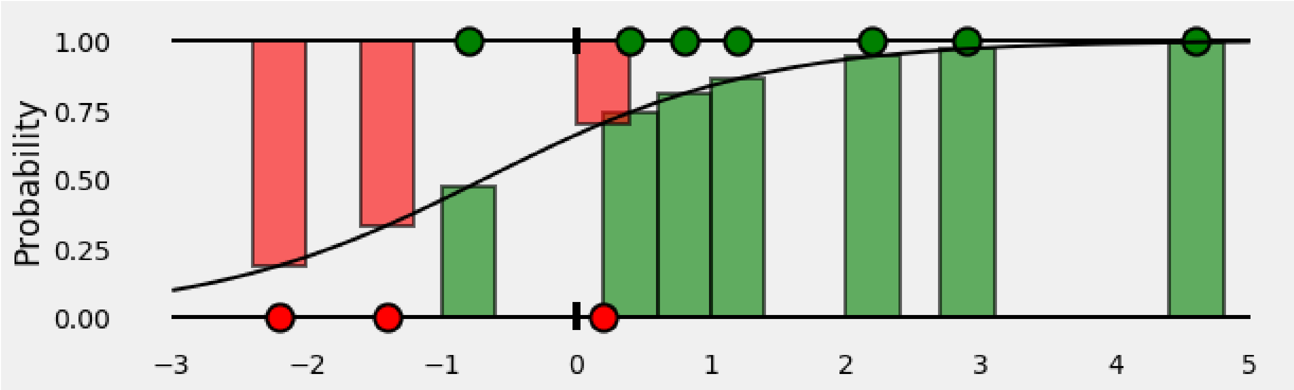

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−1.0
−0.5
0.0
0.5
0.5
0.0
−0.5
−1.0


Standard gradient descent

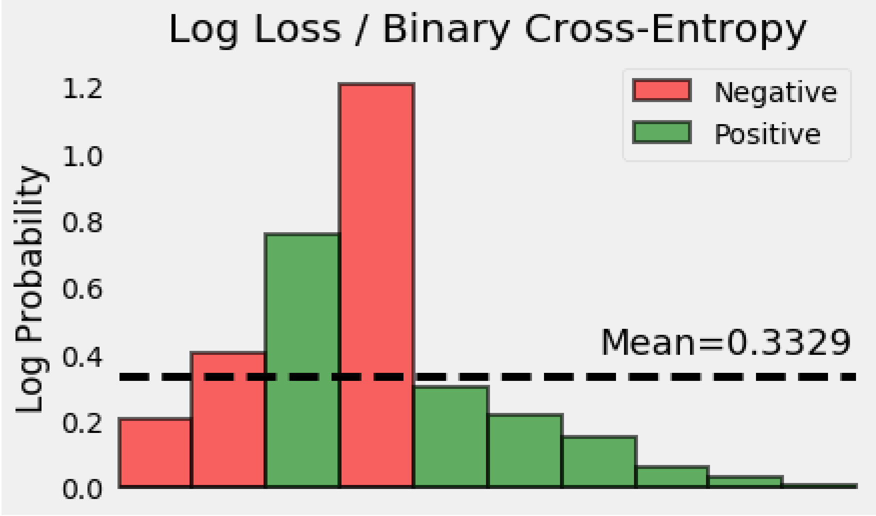

Error = 1.15
Error
b
m
m = 4.90 b = 2.08
y
x

# BINARY CROSS-ENTROPY

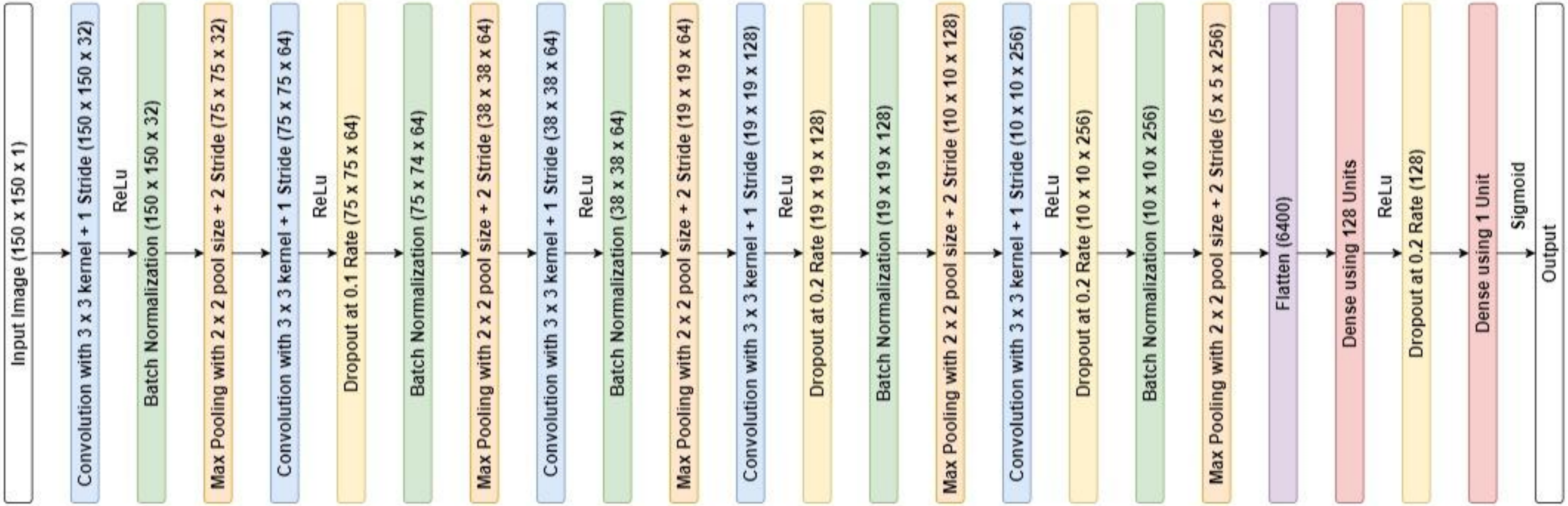
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$




Binary cross entropy compares each of the predicted probabilities to 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.



# SUMMARY OF THE CONVOLUTION LAYER

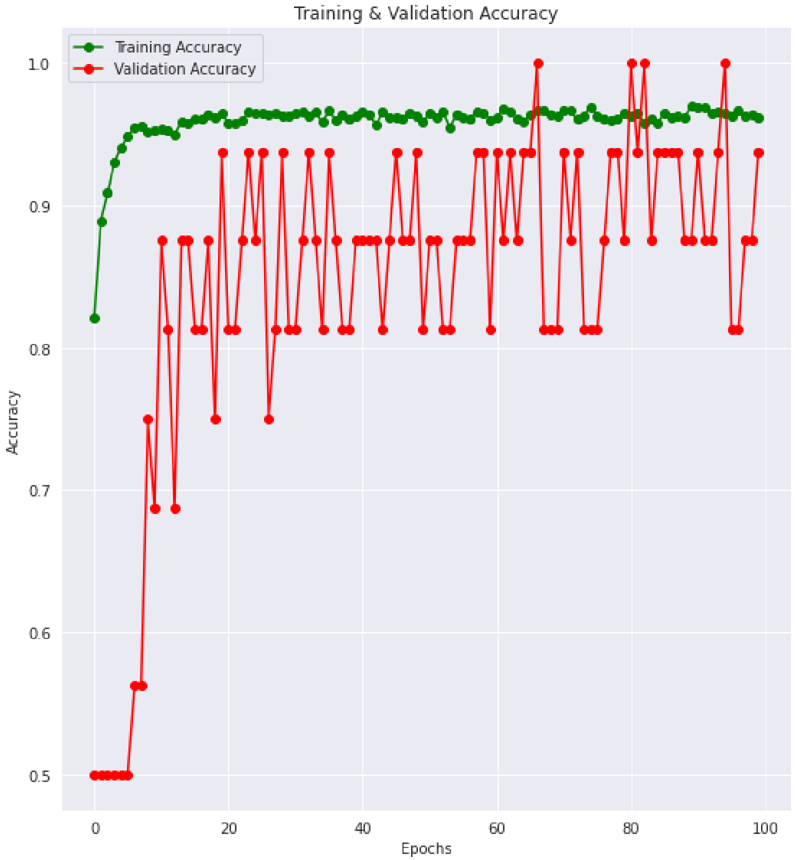


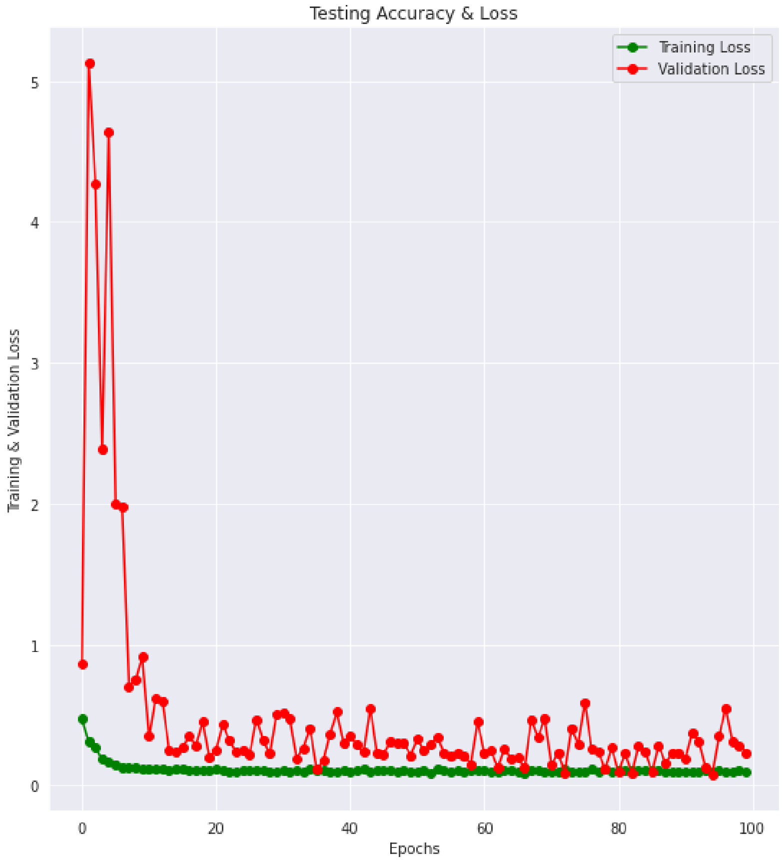
# TESTING LOSS AND ACCURACY

```
[13] print("Loss of the model is - ", model.evaluate(x_test, y_test)[0])
     print("Accuracy of the model is - ", model.evaluate(x_test, y_test)[1] * 100, '%')
     model.save('/content/drive/MyDrive/Colab Notebooks/Custom')

20/20 [==============================] - 1s 25ms/step - loss: 0.2733 - accuracy: 0.9277
Loss of the model is -  0.2732964754104614
20/20 [==============================] - 0s 14ms/step - loss: 0.2733 - accuracy: 0.9277
Accuracy of the model is -  92.76729822158813 %
```

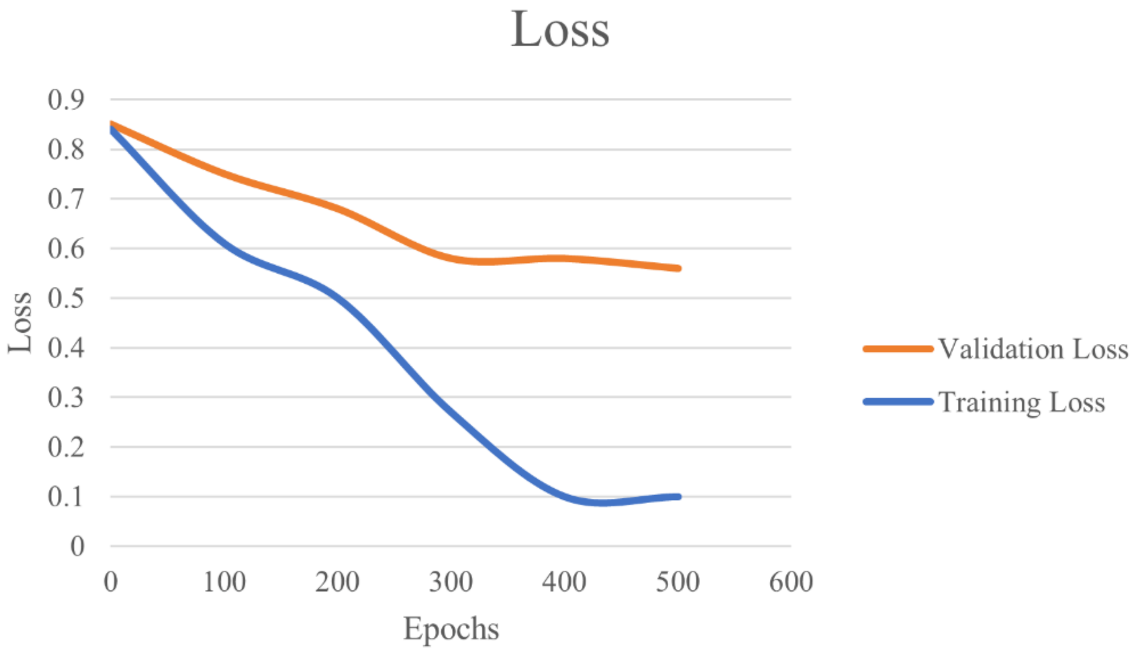
# TRAINING LOSS VS VALIDATION LOSS



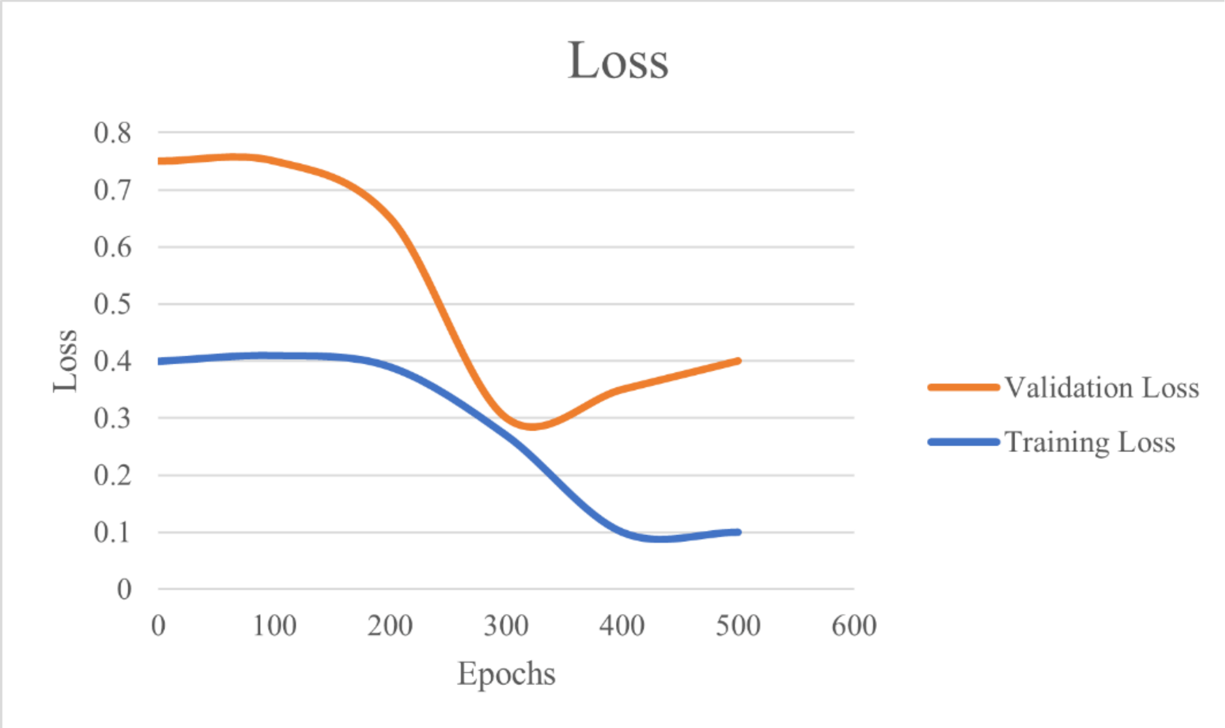
# TRAINING AND VALIDATION LOSS

At times, the validation loss is greater than the training loss. This may indicate that the model is underfitting. **Underfitting** occurs when the model is unable to accurately model the training data, and hence generates large errors.



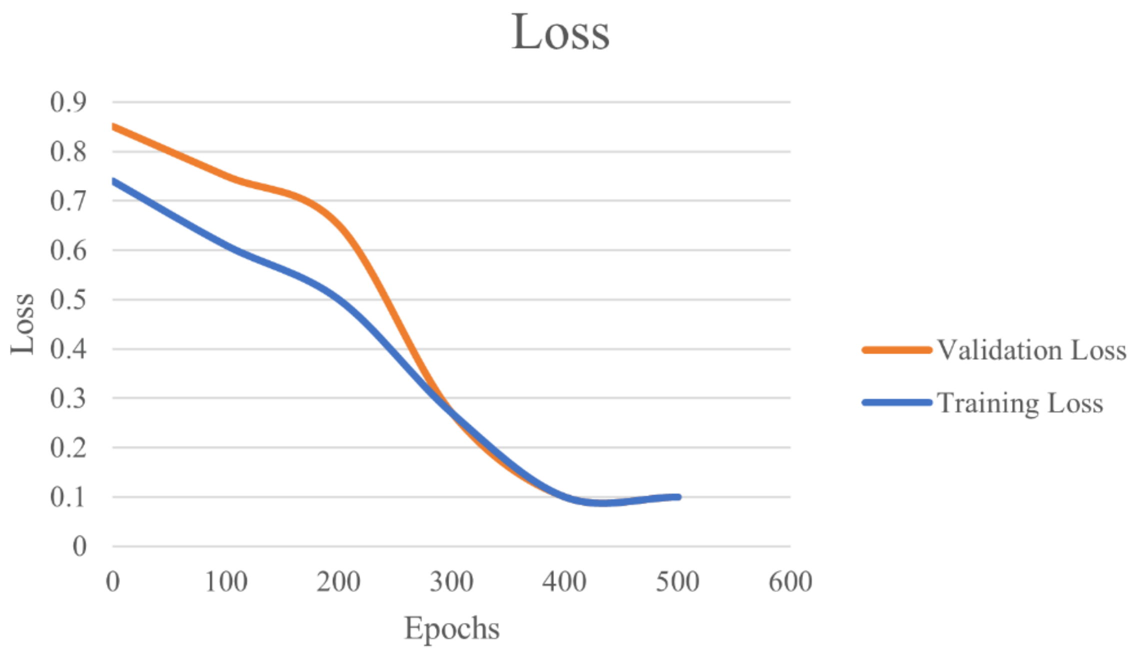
# TRAINING AND VALIDATION LOSS

This usually indicates that the model is **overfitting**, and cannot generalize on new data. In particular, the model performs well on training data but poorly on the new data in the validation set. At a point, the validation loss decreases but starts to increase again.



# TRAINING AND VALIDATION LOSS

This indicates an optimal fit, i.e a model that does not overfit or underfit.



# PREDICTING TESTING DATA USING MODEL

Predict is used to test the model on testing data. It predicts the label of a new set of data when given a trained model.

This function helps us to know how well the model has been trained by testing it with test and validation sets. Predict is can be used in test a image of X-ray and classify it as Pneumonia or Normal Sample.

We have Tested the Model by Using around 636 Images which are not used for training and 590 Images are predicted correctly

# CLASSIFICATION REPORT

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) | 0.93 | 0.96 | 0.94 | 402 |
| Normal (Class 1) | 0.92 | 0.88 | 0.90 | 234 |
| accuracy | | | 0.93 | 636 |
| macro avg | 0.93 | 0.92 | 0.92 | 636 |
| weighted avg | 0.93 | 0.93 | 0.93 | 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 name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e. commonly mislabeling one as another).

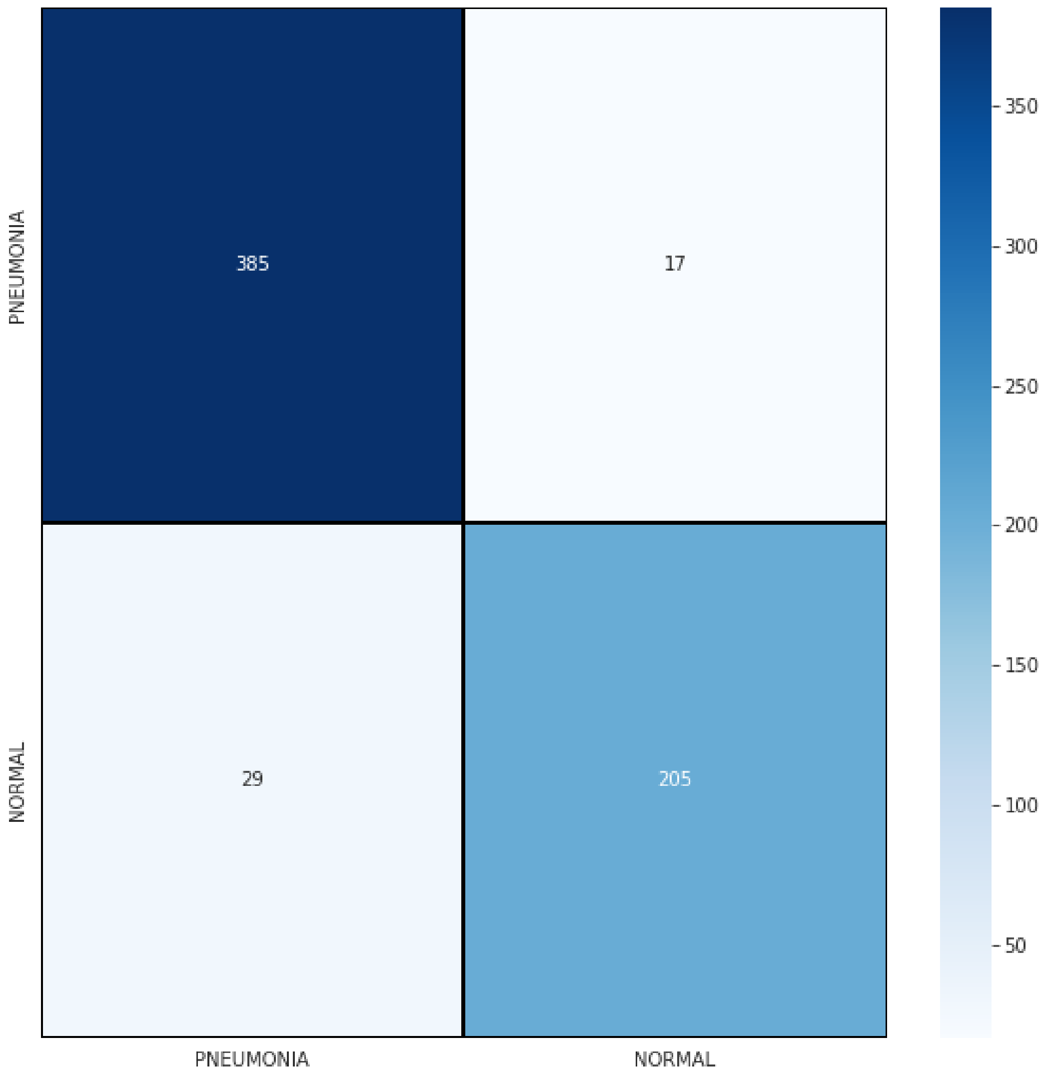
$$\text{Accuracy} = \frac{\text{TN+TP}}{\text{TN+FP+FN+TP}}$$

TP- correctly identified positive cases

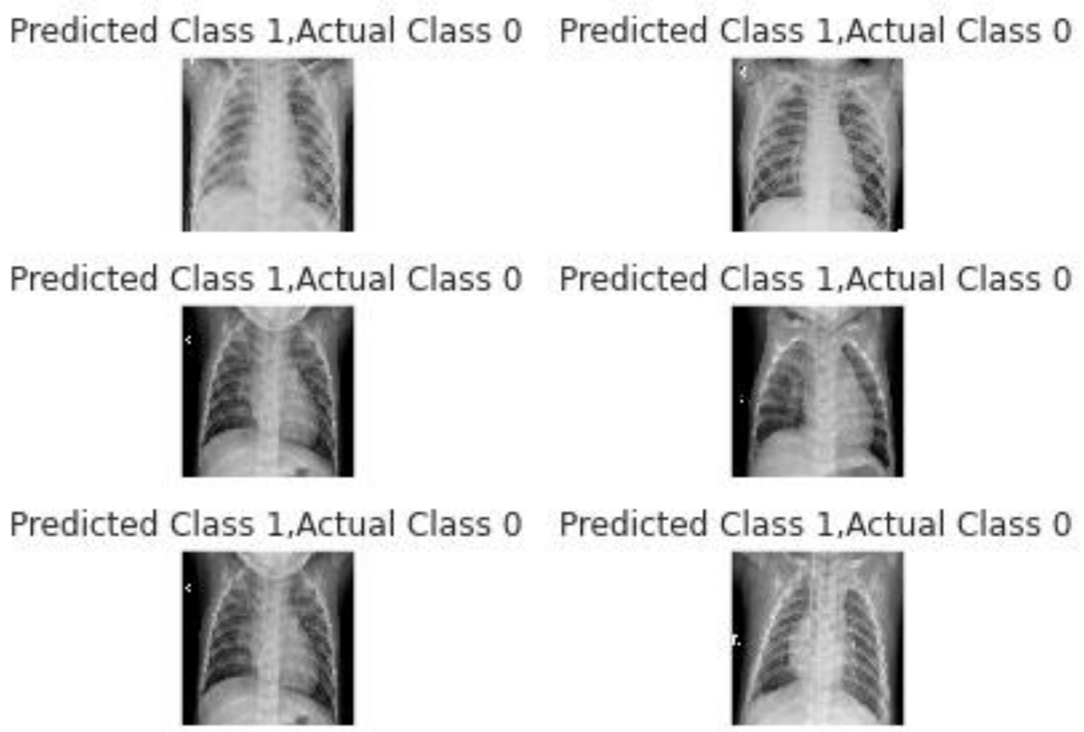
TN- correctly identified negative cases

FP- Falsely identified positive cases

FN- Falsely identified negative cases



# CASES WHERE ERROR HAS BEEN MADE WHILE PREDICTING RESULTS



# STATE-OF-THE-ART METHODS

| Paper | Year | Method | Data | $F_1$ score | Runtime |
|---|---|---|---|---|---|
| Brunese et al. [4] | 2020 | VGG-16 | 6523 CXR | 97% | 2.5 s |
| Panwar et al. [5] | 2020 | VGG-19 + GradCAM | 2482 CT + 6382 CXR | 95.61% | 2 s |
| Mahmud et al. [13] | 2020 | customized CNN (CovXNet) | 6161 CXR | 97.4% | N/A |
| Ouchicha et al. [14] | 2020 | customized CNN (CVDNet) | 2905 CXR | 96.7% | N/A |
| Wang et al. [15] | 2020 | 3D-ResNet | 4697 CXR | 93.3% | N/A |
| Choudhury et al.[31] | 2020 | DenseNet201 | 3487 CXR | 97.94% | N/A |
| Ren et al. [32] | 2020 | CNN + Bayesian Network | 35,389 CXR | 87% | N/A |
| Arias et al. [33] | 2020 | CNN | 79,500 CXR | 91.5% | N/A |
| Sakib et al. [34] | 2020 | customized CNN (DL-CRC) | 5367 CXR | 94% | N/A |
| Ozturk et al. [35] | 2020 | YOLO via DarkNet | 1000 CXR | 87–98% | $<$ 1 sec |
| Alhudjaif et al. [10] | 2021 | DenseNet-201 | 1218 CXR | 94.96% | "within seconds" |
| Nikolaou et al. [36] | 2021 | EfficientNet models | 15,153 CXR | 95% | N/A |
| Das et al. [37] | 2021 | CNN + transfer learning | 1004 CXR | 95% | "few seconds" |
| Munusamy et al. [38] | 2021 | FractalCovNet | 473 CT + 11,934 CXR | 92–98% | N/A |
| Joshi et al. [39] | 2021 | DarkNet-53 | 6884 CXR | 97.11% | 0.137 s |
| Singh and Tripathi [40] | 2022 | Quaternion CNN | 5856 CXR | 93.75% | N/A |
| Dash and Mohapatra [41] | 2022 | CNN + transfer learning | 1272 CXR | 97.12% | N/A |
| Gour and Jain [42] | 2022 | VGG-19, Xception | 4645 CT + 3040 CXR | 97.5% | 0.029–3.66 s |
| Proposed method | 2022 | CNN + modified dropout | 5856 CXR | 97.4% | 0.122 s |

THE END