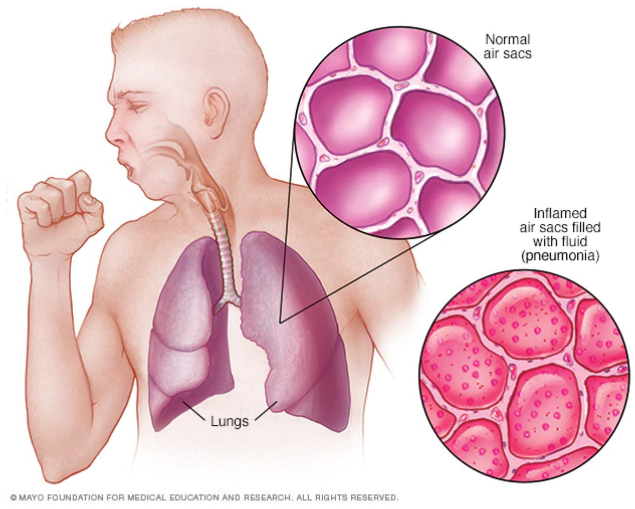
CONTENTS

Abstract

We develop an algorithm that can detect pneumonia from chest X-rays at a very good level and might be better or at least at par with practicing radiologists. Our algorithm,works on convolutional neural networks which contains three convolutional layers with 64, 128 and 216 number of feature detectors in each convolutional layer and 512 units in the hidden layer followed by the output layer with one node. Our model is trained on the dataset taken from https://data.mendeley.com/datasets/rscbjbr9sj/2**,** which has Chest X-ray images (anterior-posterior) selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou.

Introduction


Normal air sacs
Inflamed air sacs filled with fluid (pneumonia)
Lungs
© MAYO FOUNDATION FOR MEDICAL EDUCATION AND RESEARCH. ALL RIGHTS RESERVED.

Pneumonia is an infection that inflames the air sacs in one or both lungs. The air sacs may fill with fluid or pus (purulent material), causing cough with phlegm or pus, fever, chills, and difficulty breathing. A variety of organisms, including bacteria, viruses and fungi, can cause pneumonia.

Pneumonia can range in seriousness from mild to life-threatening. It is most serious for infants and young children, people older than age 65, and people with health problems or weakened immune systems.

According to WHO, pneumonia accounts for 15% of all deaths of children under 5 years old, killing 808 694 children in 2017. Pneumonia in India accounts for 20 percent of the deaths worldwide. According to the WHO, one in three deaths in India is caused by pneumonia. Every year almost 200,000 children under five die of pneumonia in India. A child dies every 2 min of pneumonia, diarrhoea in India. On a global level, pneumonia kills around 900,000 children in the world every year.

Convolutional Neural Networks

Introduction

Convolution is a mathematical concept used heavily in Digital Signal Processing when dealing with signals that take the form of a time series. In lay terms, convolution is a mechanism to combine or "blend"[10] two functions of time**3** in a coherent manner. It can be mathematically described as follows:

For a discrete domain of one variable:

$$Eq.1: \quad y[n] = x[n] * h[n] = \sum_k x[k] * h[n-k] \quad where \;\; k \in [-\infty, +\infty]$$

For a discrete domain of two variables:

$$Eq.2: \quad (f * g)(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v).g(x-u, y-v)$$

Though conventionally called as Convolutional Neural Networks, the operation performed on image inputs with CNNs is not strictly convolution, but rather a slightly modified variant called cross-correlation[10], in which one of the inputs is time-reversed:

$$Eq.3: \quad y[n] = x[n]*h[n] = \sum_k x[k]*h[n+k] \quad where \quad k \in [-\infty, +\infty]$$

Definition

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

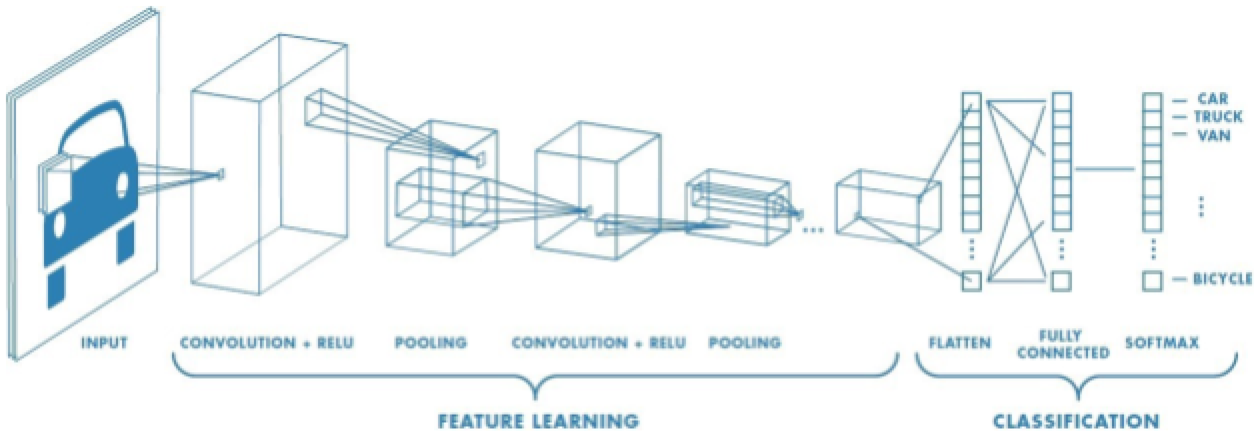


Figure : A fully connected CNN.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Why ConvNets over feed-forward Neural Nets?

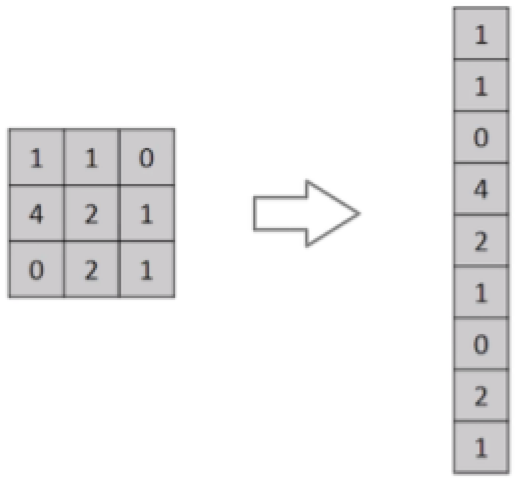


Fig: Flattening of 3x3 image into 9x1 vector.

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Input Volume

CNNs are usually applied to image data. Every image is a matrix of pixel values. The range of values that can be encoded in each pixel depends upon its bit size. Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255]. However, with coloured images, particularly RGB (Red, Green, Blue)-based images, the presence of separate colour channels (3 in the case of RGB images) introduces an additional 'depth' field to the data, making the input 3-dimensional. Hence, for a given RGB image of size, say 255×255 (Width x Height) pixels, we'll have 3 matrices associated with each image, one for each of the colour channels. Thus the image in its entirety, constitutes a 3-dimensional structure called the *Input Volume* (255x255x3). In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc. You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

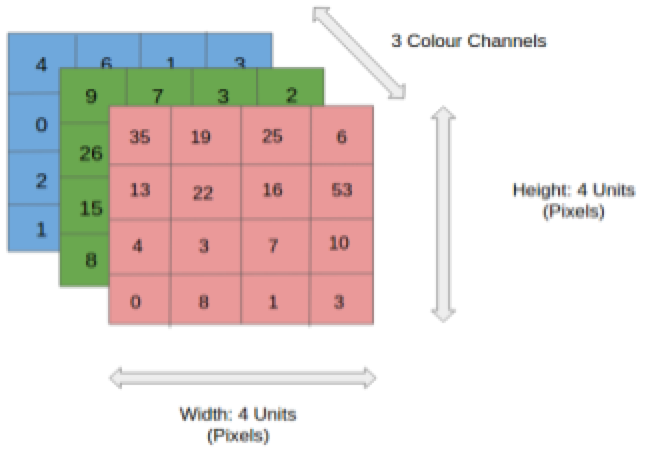


Figure: 4x4x3 RGB image.

Features

Just as its literal meaning implies, a feature is a distinct and useful observation or pattern obtained from the input data that aids in performing the desired image analysis. The CNN learns the features from the input images. Typically, they emerge repeatedly from the data to gain prominence. As an example, when performing Face Detection, the fact that every human face has a pair of eyes will be treated as a feature by the system, that will be detected and learned by the distinct layers. In generic object classification, the edge contours of the objects serve as the features.

Filters (Convolutional Kernels)

A filter (or kernel) is an integral component of the layered architecture.

Generally, it refers to an operator applied to the entirety of the image such that it transforms the information encoded in the pixels. In practice, however, the kernel is a smaller-sized matrix in comparison to the input dimensions of the image, that consists of real valued entries.

The kernels are then convolved with the input volume to obtain so-called ‘activation maps’. Activation maps indicate ‘activated’ regions, i.e. regions where features specific to the kernel have been detected in the input. The real values of the kernel matrix change with each learning iteration over the training set, indicating that the network is learning to identify which regions are of significance for extracting features from the data.

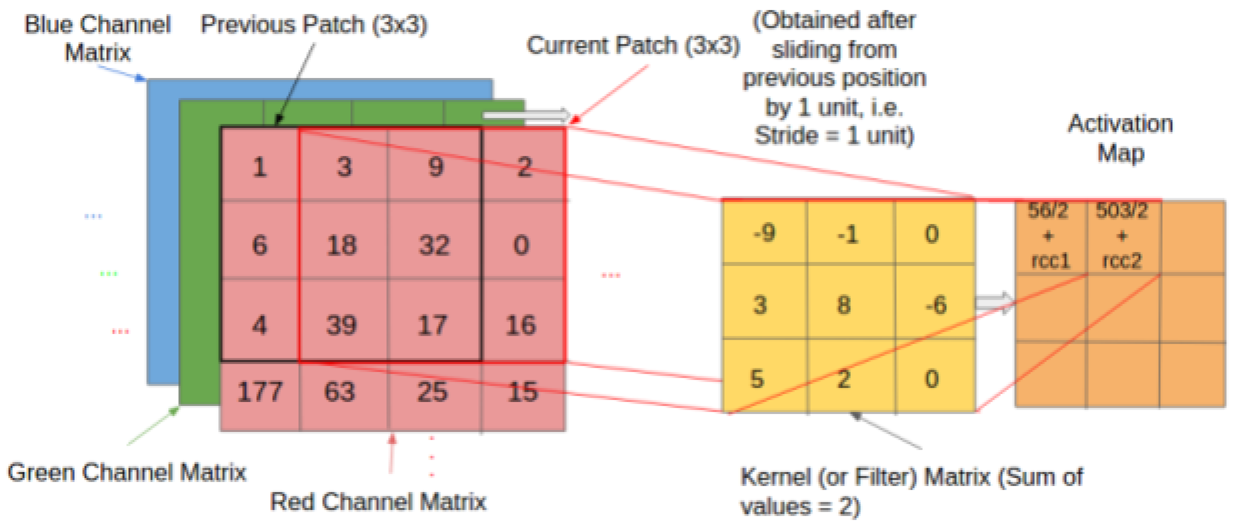
Kernel Operations

The exact procedure for convolving a Kernel (say, of size 16 x 16) with the input volume (a 256 x 256 x 3 sized RGB image in our case) involves taking patches from the input image of size equal to that of the kernel (16 x 16), and convolving (or calculating the dot product) between the values in the patch and those in the kernel matrix.

The convolved value obtained by summing the resultant terms from the dot product forms a single entry in the activation matrix. The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the 'stride' value, and the process is repeated until the entire input image has been processed. The process is carried out for all colour channels. For normalization purposes, we divide the calculated value of the activation matrix by the sum of values in the kernel matrix.

The process is demonstrated in the figure below**,** using a toy example consisting of a 3-channel 4×4-pixels input image and a 3×3 kernel matrix. Note that:

- pixels are numbered from 1 in the example.
- the values in the activation map are normalized to ensure the same intensity range between the input volume and the output volume**.** Hence, for normalization, we divide the calculated value for the 'red' channel by 2 (the sum of values in the kernel matrix).
- we assume the same kernel matrix for all three channels, but it is possible to have a separate kernel matrix for each colour channel.



In the Figure, the convolution value is calculated by taking the dot product of the corresponding values in the Kernel and the channel matrices. The current path is indicated by the red-coloured, bold outline in the Input Image volume. Here, the entry in the activation matrix is calculated as:

$$\boxed{\begin{array}{c} \textit{AM[1][2]} = \textit{Red Channel Matrix Contribution(CMC)} + \textit{Green CMC} + \\ \textit{Blue CMC)} \\ = [(3, 9, 2).(-9, -1, 0) + (18, 32, 0).(3, 8, -6) + (39, 17, 16).(5, 2, 0)]/2 \\ \text{(From-red-channel)} + \text{rcc2} \\ = [-27-9+0+54+256-0+195+34+0]/2 + \text{rcc2} = \mathbf{503/2 + rcc2} \\ \\ \textit{where rcc2 refers to the contribution from the remainder channels.} \end{array}}$$

The CNN Architecture

Now that we are familiar with the CNN terminology, let's go on ahead and study the CNN architecture in detail.

The architecture of a typical CNN is composed of multiple layers where each layer performs a specific function of transforming its input into a useful representation. There are 3 major types of layers that are commonly observed in complex neural network architectures:

Convolutional Layer

Also referred to as Conv. layer, it forms the basis of the CNN and performs the core operations of training and consequently firing the neurons of the network. It performs the convolution operation over the input volume as specified in the previous section, and consists of a 3-dimensional arrangement of neurons (a stack of 2-dimensional layers of neurons, one for each channel depth).

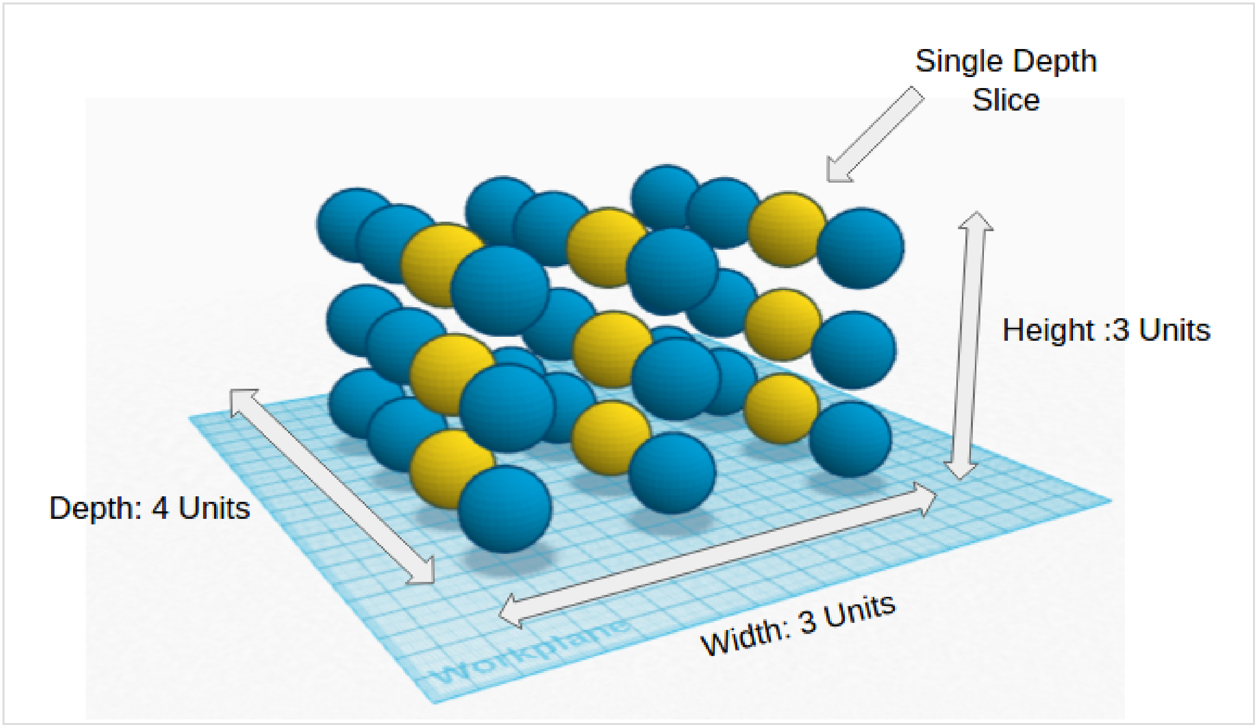


Figure 4: A 3-D representation of the Convolutional layer with 3 x 3 x 4 = 36 neurons.

Each neuron is connected to a certain region of the input volume called the receptive field (explained in the previous section). For example, for an input image of dimensions 28x28x3, if the receptive field is 5 x 5, then each neuron in the Conv. layer is connected to a region of 5x5x3 (the region always comprises the entire depth of the input, i.e. all the channel matrices) in the input volume. Hence each neuron will have 75 weighted inputs. For a particular value of **R** (receptive field), we have a cross-section of neurons entirely dedicated to taking inputs from this region. Such a cross-section is called a 'depth column'. It extends to the entire depth of the Conv. layer.

For optimized Conv. layer implementations, we may use a **Shared Weights** model that reduces the number of unique weights to train and consequently the matrix calculations to be performed per layer. In this model, each 'depth slice' or a single 2-dimensional layer of neurons in the Conv architecture all share the same weights. The caveat with parameter sharing is that it doesn't work well with images that encompass a spatially centered structure (such as face images), and in applications where we want the distinct features of the image to be detected in spatially different locations of the layer.

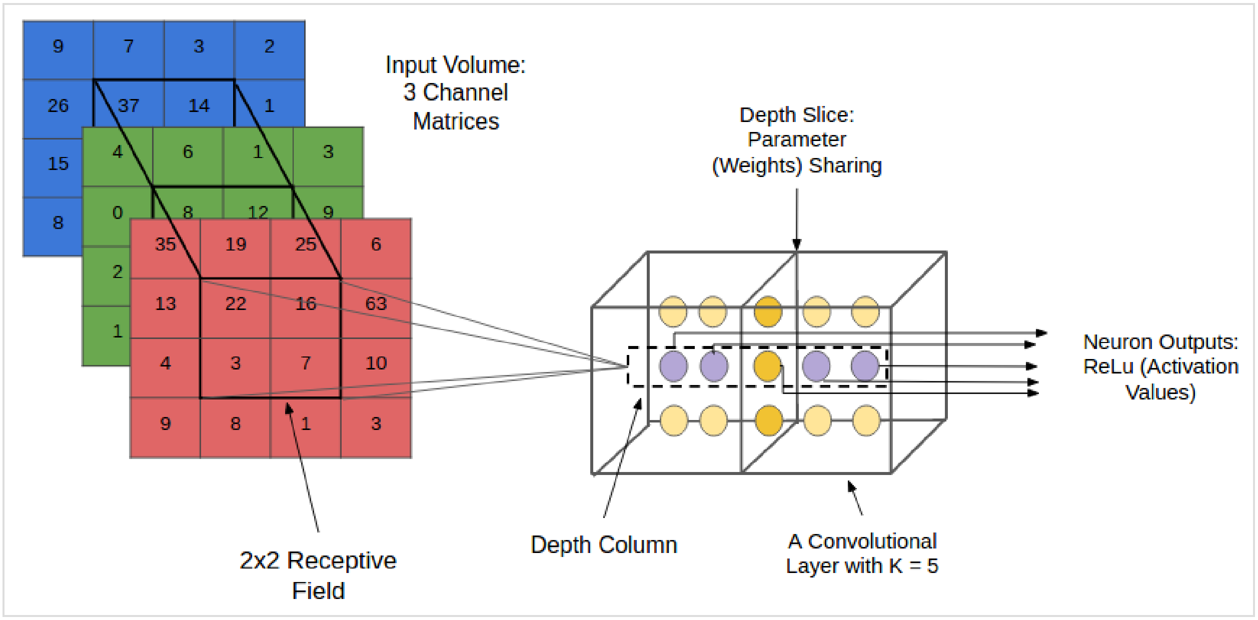

Input Volume: 3 Channel Matrices
Depth Slice: Parameter (Weights) Sharing
Neuron Outputs: ReLu (Activation Values)
2x2 Receptive Field
Depth Column
A Convolutional Layer with K = 5

Figure 5: Concept of Receptive Field.

We must keep in mind though that the network operates in the same way that a feed-forward network would: the weights in the Conv layers are trained and updated in each learning iteration using a Back-propagation algorithm extended to be applicable to 3-dimensional arrangements of neurons.

The ReLu (Rectified Linear Unit) Layer

ReLu refers to the Rectifier Unit, the most commonly deployed activation function for the outputs of the CNN neurons. Mathematically, it's described as:

$$Eq.3 : max(0, x)$$

Unfortunately, the ReLu function is not differentiable at the origin, which makes it hard to use with backpropagation training. Instead, a smoothed version called the Softplus function is used in practice:

$$Eq.4 : f(x) = ln(1 + e^x)$$

The Pooling Layer

The pooling layer is usually placed after the Convolutional layer. Its primary utility lies in reducing the spatial dimensions (Width x Height) of the Input Volume for the next Convolutional Layer. It does not affect the depth dimension of the Volume.

The operation performed by this layer is also called 'down-sampling', as the reduction of size leads to loss of information as well. However, such a loss is beneficial for the network for two reasons:

1. the decrease in size leads to less computational overhead for the upcoming layers of the network.
2. it work against overfitting.

Much like the convolution operation performed above, the pooling layer takes a sliding window or a certain region that is moved in stride across the input transforming the values into representative values. The transformation is either performed by taking the maximum value from the values observable in the window (called ‘max pooling’), or by taking the average of the values. Max pooling has been favoured over others due to its better performance characteristics.

The operation is performed for each depth slice. For example, if the input is a volume of size 4x4x3, and the sliding window is of size 2×2, then for each color channel, the values will be down-sampled to their representative maximum value if we perform the max pooling operation.

No new parameters are introduced in the matrix by this operation. The operation can be thought of as applying a function over input values, taking fixed sized portions at a time, with the size, modifiable as a parameter. Pooling is optional in CNNs, and many architectures do not perform pooling operations.

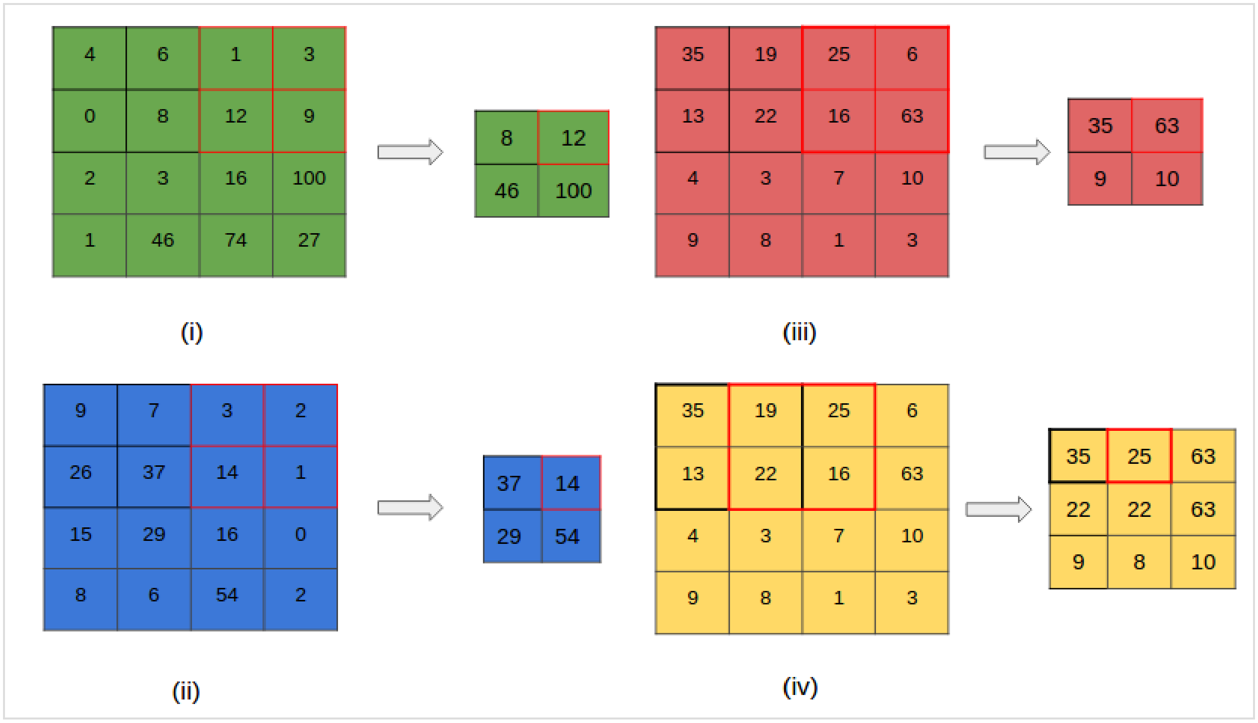
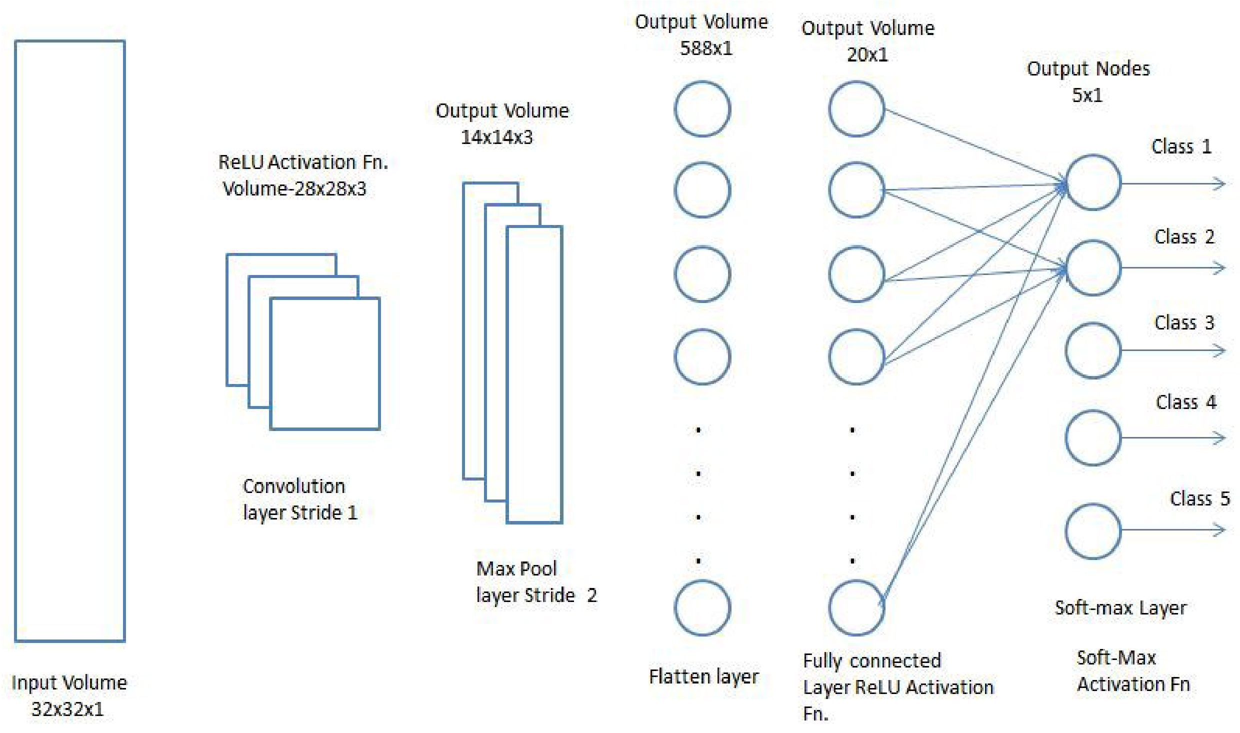


Figure 6: The Max-Pooling operation can be observed in sub-figures (i), (ii) and (iii) that max-pools the 3 colour channels for an example input volume for the pooling layer. The operation uses a stride value of [2, 2]. The dark and red boundary regions describe the window movement. Sub-figure (iv) shows the operation applied for a stride value of [1,1], resulting in a 3×3 matrix Here we observe overlap between regions.

Fully Connected Layer



The Fully Connected layer is configured exactly the way its name implies: it is fully connected with the output of the previous layer. Fully-connected layers are typically used in the last stages of the CNN to connect to the output layer and construct the desired number of outputs.

After the conversion of the input image into a suitable form for the Multi-Level Perceptron, we flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

Dataset used for this project

The dataset is taken from https://data.mendeley.com/datasets/rscbjbr9sj/2

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.

The images were graded by two expert physicians before being cleared for training the AI system. The evaluation set was also checked by a third expert.

There are 5,856 X-Ray images and 2 categories out of which 4,273 characterized as Pneumonia (2,733 bacterial and 1,540 viral) and 1,583 normal.

Model Specifications

Data Splitting

The data, which included 5856 total images was divided into the following sets for the purposes of training and testing the model :

	Normal	Infected (Pneumonia)
Training set	951	2565
Validation set	316	854
Test set	316	854

Specifications and architecture

Used **Anaconda navigator** to create a new environment for using GPU and the code was written and implemented in **Spyder** using **python**.

Used Keras library with tensorflow backend to create the following CNN architecture :

Conv2D ⇒ Max Pooling ⇒ Conv2D ⇒ Max Pooling ⇒ Conv2D ⇒ Max Pooling ⇒ Flatten ⇒ Fully Connected Layer (1 hidden layer) ⇒ Output

Performed **Image augmentation** for better performance: 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.

Source code

```
# Part 1 - Building the CNN

# Importing the Keras libraries and packages

import keras

from keras.layers import Dense, Conv2D

from keras.layers import Flatten

from keras.layers import MaxPooling2D, GlobalAveragePooling2D

from keras.layers import Activation

from keras.layers import BatchNormalization

from keras.layers import Dropout

from keras.models import Sequential

from keras import backend as K

from keras import optimizers

import matplotlib.pyplot as plt

import numpy as np

from sklearn import metrics

import numpy as np

from keras.preprocessing import image



# Initialising the CNN

classifier = Sequential()


# Step 1 - Convolution

classifier.add(Conv2D(64, (3, 3), input_shape = (64, 64, 3),
activation = 'relu'))


# Step 2 - Pooling
```

```
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a two more convolutional layers
classifier.add(Conv2D(128, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(216, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 512, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.3,
                                   zoom_range = 0.3,
                                   horizontal_flip = True,
                                   vertical_flip=True)
```

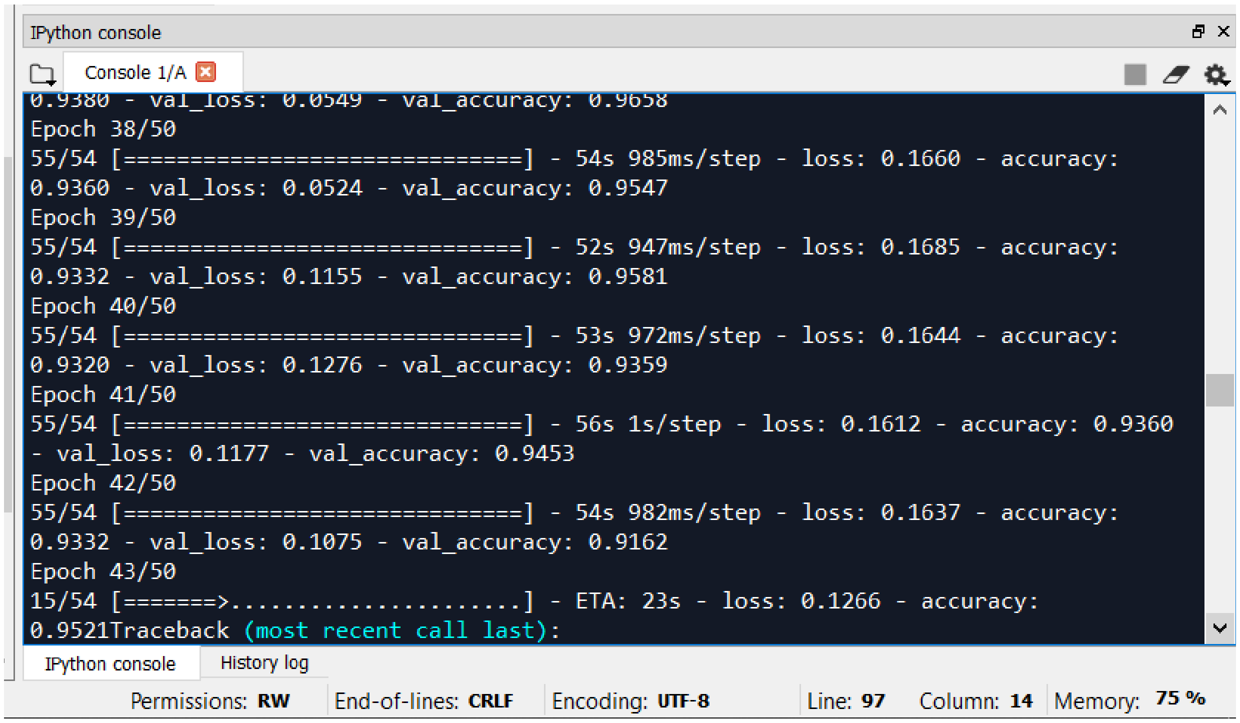
```
val_datagen = ImageDataGenerator(rescale = 1./255)

test_datagen = ImageDataGenerator(rescale = 1./255)



training_set = train_datagen.flow_from_directory('dataset/train',

                                                 target_size = (64,
64),

                                                 batch_size = 64,

                                                 class_mode =
'binary')


val_set = val_datagen.flow_from_directory('dataset/val',

                                           target_size = (64, 64),

                                           batch_size = 64,

                                           class_mode = 'binary')


history=classifier.fit_generator(training_set,

                         steps_per_epoch = 3516/64,

                         epochs = 50,

                         validation_data = val_set,

                         validation_steps = 1170/64)




test_set = test_datagen.flow_from_directory('dataset/test',

                                            target_size = (64, 64),

                                            batch_size = 64,

                                            class_mode =
'binary',shuffle=False)
```

```
pred=classifier.predict_generator(test_set,steps=1170/64)


predictedClasses = np.where(pred>0.5, 1, 0)

true_classes = test_set.classes

class_labels = list(test_set.class_indices.keys())

cm = metrics.confusion_matrix(true_classes, predictedClasses)

report = metrics.classification_report(true_classes, predictedClasses,
target_names=class_labels)

print(report)



# Check for overfitting or underfitting

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

t = f.suptitle('Basic CNN Performance', fontsize=12)

f.subplots_adjust(top=0.85, wspace=0.3)

max_epoch = len(history.history['acc'])+1

epoch_list = list(range(1,max_epoch))

ax1.plot(epoch_list, history.history['acc'], label='Train Accuracy')

ax1.plot(epoch_list, history.history['val_acc'], label='Validation
Accuracy')

ax1.set_xticks(np.arange(1, max_epoch, 5))

ax1.set_ylabel('Accuracy Value')

ax1.set_xlabel('Epoch')

ax1.set_title('Accuracy')

l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')

ax2.plot(epoch_list, history.history['val_loss'], label='Validation
Loss')

ax2.set_xticks(np.arange(1, max_epoch, 5))

ax2.set_ylabel('Loss Value')
```

```
ax2.set_xlabel('Epoch')

ax2.set_title('Loss')

l2 = ax2.legend(loc="best")


# Checking results for a new image :


img = image.load_img('test_image_normal.jpeg')

imgplot = plt.imshow(img)

print('The image that you have entered for testing :\n')

plt.show()

print('\n')

print('ACTUAL OBSERVATION : PNEUMONIA PRESENT \n')

print('----- Testing on the trained model ----- \n')

print("MODEL's OBSERVATION :")

test_image = image.load_img('test_image_normal.jpeg', target_size =
(64, 64))

test_image = image.img_to_array(test_image)

test_image = np.expand_dims(test_image, axis = 0)

result = classifier.predict_on_batch(test_image)

print(result)

if result[0][0] == 0:

 print("NORMAL")

else:

 print("PNEUMONIA PRESENT")
```

Results obtained

After running our model on 50 epochs with a batch size of 64 we obtained the following results:



IPython console

Console 1/A

```
0.9380 - val_loss: 0.0549 - val_accuracy: 0.9658
Epoch 38/50
55/54 [==============================] - 54s 985ms/step - loss: 0.1660 - accuracy:
0.9360 - val_loss: 0.0524 - val_accuracy: 0.9547
Epoch 39/50
55/54 [==============================] - 52s 947ms/step - loss: 0.1685 - accuracy:
0.9332 - val_loss: 0.1155 - val_accuracy: 0.9581
Epoch 40/50
55/54 [==============================] - 53s 972ms/step - loss: 0.1644 - accuracy:
0.9320 - val_loss: 0.1276 - val_accuracy: 0.9359
Epoch 41/50
55/54 [==============================] - 56s 1s/step - loss: 0.1612 - accuracy: 0.9360
- val_loss: 0.1177 - val_accuracy: 0.9453
Epoch 42/50
55/54 [==============================] - 54s 982ms/step - loss: 0.1637 - accuracy:
0.9332 - val_loss: 0.1075 - val_accuracy: 0.9162
Epoch 43/50
15/54 [=======>......................] - ETA: 23s - loss: 0.1266 - accuracy:
0.9521Traceback (most recent call last):
```

IPython console | History log

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 97 | Column: 14 | Memory: 75 %

Note: *We programmed our model to do 50 epochs but since we started getting redundant accuracies, we stopped at the 42nd epoch.*

Loss *is the penalty for a bad prediction. That is,* ***loss*** *is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples.*

While ***training accuracy*** *is usually the accuracy you get if you apply the model on the training data,*

Training set loss : 0.1637

Training set accuracy : 0.9332 (93.32%)

Validation set loss : 0.1075

Validation set accuracy : 0.9162 (91.62%)

Model evaluation

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future. Evaluating model performance with the data used for training is not acceptable in data science because it can easily generate overoptimistic and overfitted models. There are two methods of evaluating models in data science, Hold-Out and Cross-Validation. To avoid overfitting, both methods use a test set (not seen by the model) to evaluate model performance.

Hold-Out

In this method, the mostly large dataset is *randomly* divided into three subsets:

1. **Training set** is a subset of the dataset used to build predictive models.
2. **Validation set** is a subset of the dataset used to assess the performance of model built in the training phase. It provides a test platform for fine tuning model's parameters and selecting the best-performing model. Not all modeling algorithms need a validation set.
3. **Test set** or unseen examples is a subset of the dataset to assess the likely future performance of a model. If a model fit to the training set much better than it fits the test set, overfitting is probably the cause.

Cross-Validation

When only a limited amount of data is available, to achieve an unbiased estimate of the model performance we use *k*-fold cross-validation. In *k*-fold cross-validation, we divide the data into *k* subsets of equal size. We build models *k* times, each time leaving out one of the subsets from training and use it as the test set. If *k* equals the sample size, this is called "leave-one-out".

We have used the Hold-out method in this project.

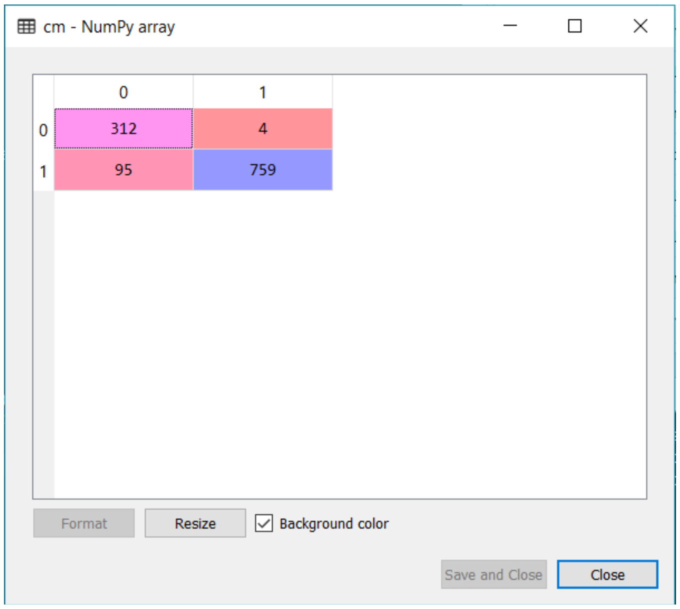
Evaluation using confusion matrix

A **confusion matrix** shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes (target value) in the data. The matrix is *N*x*N*, where *N* is the number of target values (classes). Performance of such models is commonly evaluated using the data in the matrix. The following table displays a 2x2 confusion matrix for two classes (Positive and Negative).

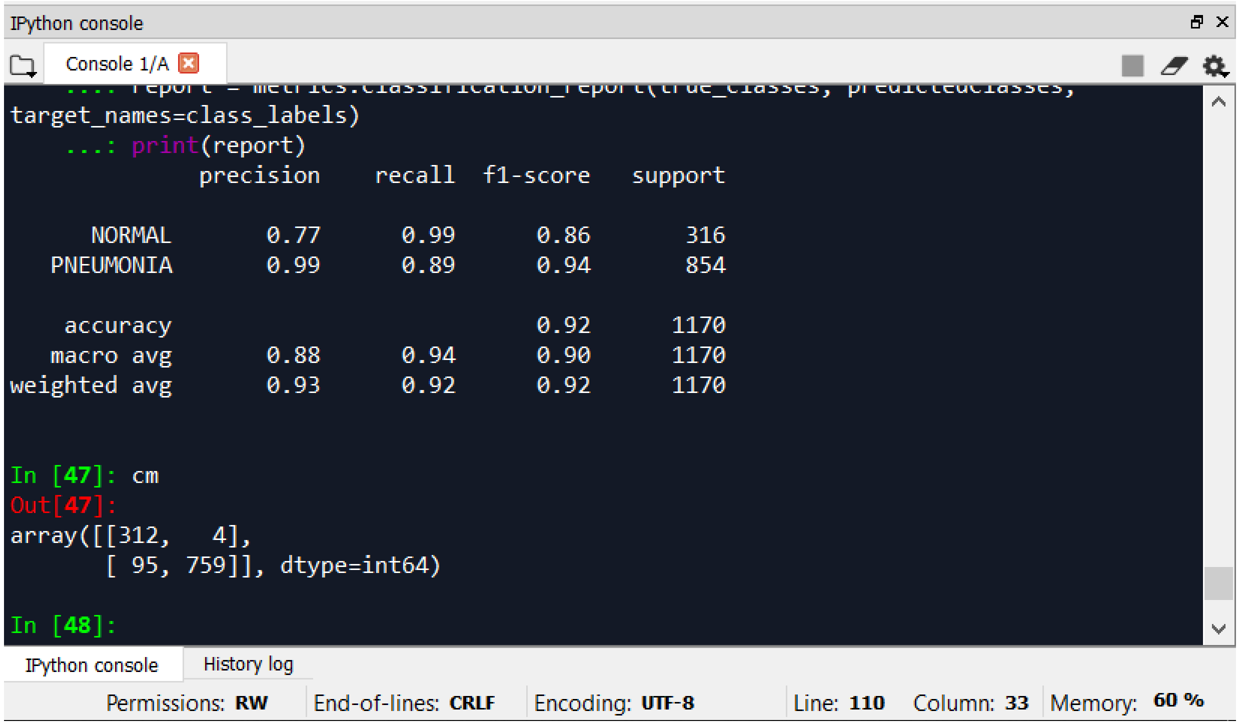
Confusion Matrix		**Target**			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

- **F1 score** is a measure of a test's accuracy. It considers both the **precision p** and the **recall r** of the test to compute the score: **p** is the number of correct positive results divided by the number of all positive results returned by the classifier, and **r** is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic mean of precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

We had the following confusion matrix (cm), precion, recall and f1 scores for our model :

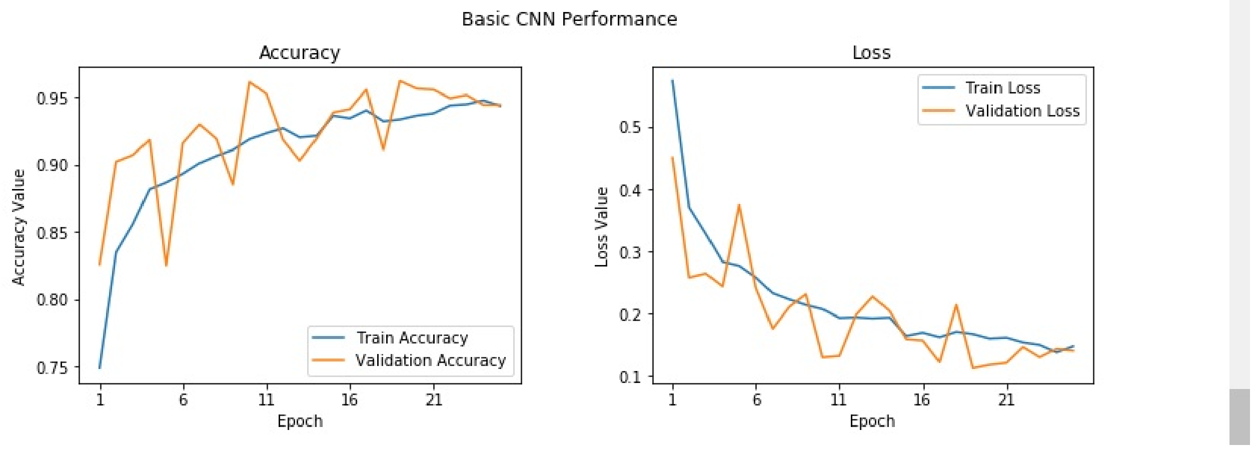


	0	1
0	312	4
1	95	759



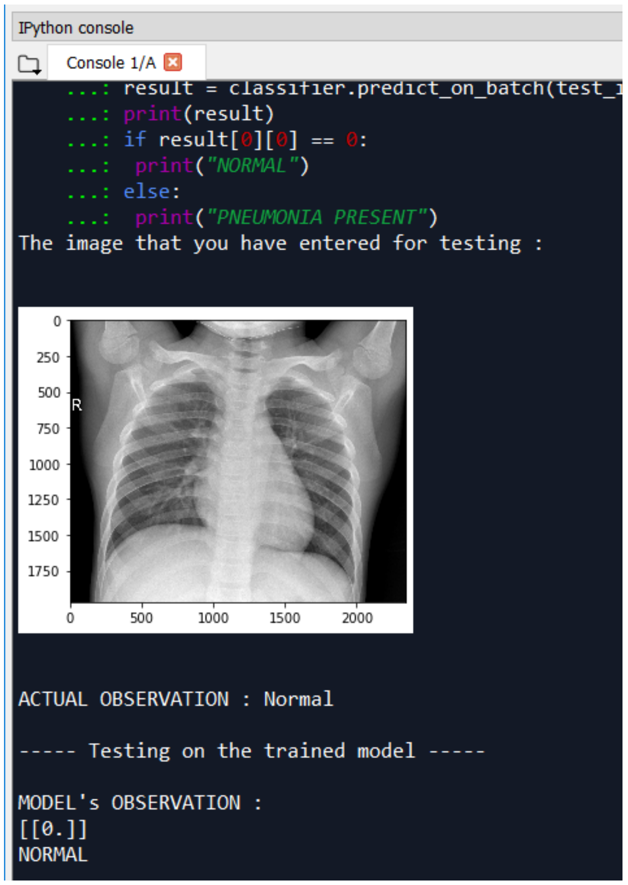
```
    ...: report = metrics.classification_report(true_classes, predictedclasses,
target_names=class_labels)
    ...: print(report)
              precision    recall  f1-score   support

      NORMAL       0.77      0.99      0.86       316
   PNEUMONIA       0.99      0.89      0.94       854

    accuracy                           0.92      1170
   macro avg       0.88      0.94      0.90      1170
weighted avg       0.93      0.92      0.92      1170


In [47]: cm
Out[47]:
array([[312,   4],
       [ 95, 759]], dtype=int64)

In [48]:
```

Checking for **overfitting** and **underfitting** using train accuracy and validation accuracy plot & train loss and validation loss plot :

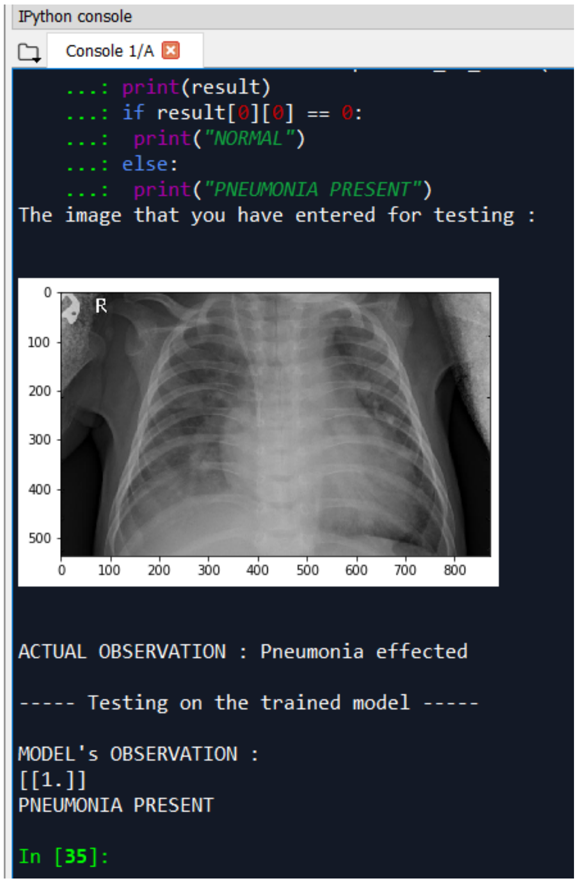


The model was then fed with two new images of chest x-rays, with one being normal and the other infected. The model had the following results for the two :

1) The normal image
2) The pneumonia affected image



```
   ...: result = classifier.predict_on_batch(test_i
   ...: print(result)
   ...: if result[0][0] == 0:
   ...:  print("NORMAL")
   ...: else:
   ...:  print("PNEUMONIA PRESENT")
The image that you have entered for testing :

ACTUAL OBSERVATION : Normal

----- Testing on the trained model -----

MODEL's OBSERVATION :
[[0.]]
NORMAL
```



```
   ...: print(result)
   ...: if result[0][0] == 0:
   ...:  print("NORMAL")
   ...: else:
   ...:  print("PNEUMONIA PRESENT")
The image that you have entered for testing :

ACTUAL OBSERVATION : Pneumonia effected

----- Testing on the trained model -----

MODEL's OBSERVATION :
[[1.]]
PNEUMONIA PRESENT

In [35]:
```

Conclusion and future work

Pneumonia accounts for a significant proportion of patient morbidity and mortality. Early diagnosis and treatment of pneumonia is critical to preventing complications including death. With approximately 2 billion procedures per year, chest X-rays are the most common imaging examination tool used in practice, critical for screening, diagnosis, and management of a variety of diseases including pneumonia. However, two thirds of the global population lacks access to radiology diagnostics, according to an estimate by the World Health Organization. There is a shortage of experts who can interpret X-rays, even when imaging equipment is available, leading to increased mortality from treatable diseases.

We have developed a model which detects pneumonia from frontal view chest-x ray images at the level of practicing radiologists. This model could be made available to the public through webapps, websites or android applications where they just need to upload or scan their chest x-rays and check whether they are infected or not. Also, it would ease the work of the doctors. With automation at the level of experts, we hope that this technology can improve healthcare delivery and increase access to medical imaging expertise in parts of the world where access to skilled radiologists is limited.

References

https://www.saedsayad.com/model_evaluation_c.htm

https://www.saedsayad.com/model_evaluation.htm

https://en.wikipedia.org/wiki/F1_score

https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://community.rapidminer.com/discussion/32499/what-is-training-accuracy-testing-accuracy

https://datascience.stackexchange.com/questions/28426/train-accuracy-vs-test-accuracy-vs-confusion-matrix

CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning

https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/

Coursera.org

https://www.mayoclinic.org/diseases-conditions/pneumonia/symptoms-causes/syc-20354204

https://data.mendeley.com/datasets/rscbjbr9sj/2

https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

https://udemy.com/machinelearning/