PROJECT REPORT ON PNEUMONIA DETECTION FROM CHEST X-RAYS USING DEEP LEARNING

Submitted By:
Adarsh Das
MT2018501

Under The Guidance Of:
Professor Neelam Sinha

TABLE OF CONTENTS

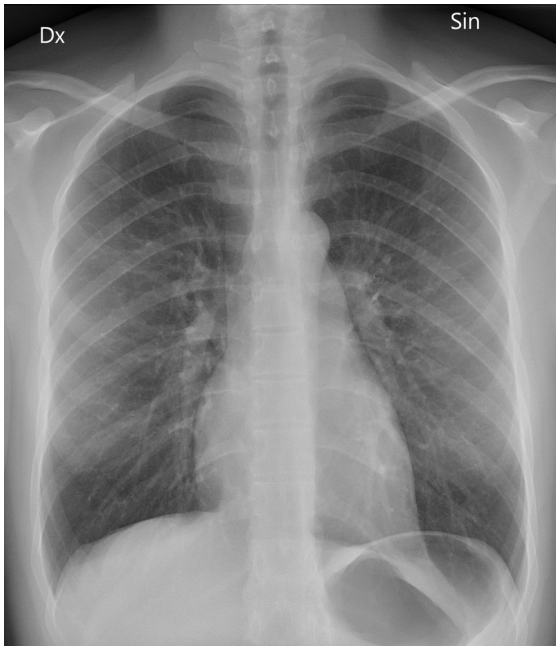
ABSTRACT

Chest X-ray images are one of the only methods which can reliably predict whether or not a person has pneumonia. Typically it is done by radiology professionals and till now has not been automated. But recent studies have shown that by using deep learning techniques results surpassing humans can also be achieved. This project explores two of such deep learning methods. One is a convolutional auto encoder and the other is a convolutional neural network.

INTRODUCTION

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.

More than 1 million adults are hospitalized with pneumonia and around 50,000 die from the disease every year in the US alone.Fifty percent of the world's pneumonia deaths occur in India which means approximately 3.7 lakh children die of pneumonia annually in India. Chest X-rays are currently the best available method for diagnosing pneumonia, playing a crucial role in clinical care and epidemiological studies. However, detecting pneumonia in chest X-rays is a challenging task that relies on the availability of expert radiologists.



A sample chest x-ray. Chest x-ray's are generally .DICOM images.

The key idea in using deep learning is to automate the task of identifying chest X-rays that show signs of pneumonia from those that don't. To do this i used two approaches.

- Using Auto-encoders for classification.

- Using CNN to classify

Autoencoders (AE) are a family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation and then reconstructing the output from this representation. We first determined the latent space representations for each x-ray image and then use these representations as an input to a binary classifier to predict whether a person has pneumonia or not.

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. We input our pre-processed data into the CNN. As we train the CNN over epochs it learns the various representations of our input data. Then we use a Fully Connected Layer to help use classify the whole network.

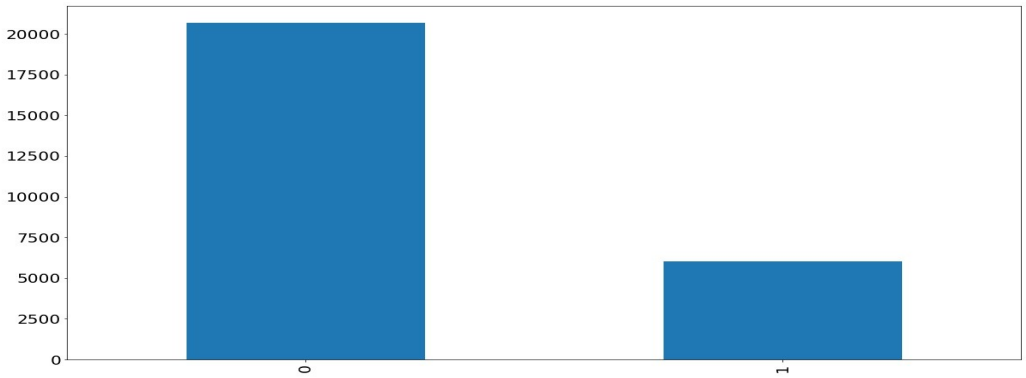
DATASET AND PREPROCESSING THE INPUT DATA

DATASET

The data was collected from RSNA Pneumonia detection challenge on kaggle. The data consisted of almost 26000 images of lung X-Rays. All these x-rays were in .DICOM format.

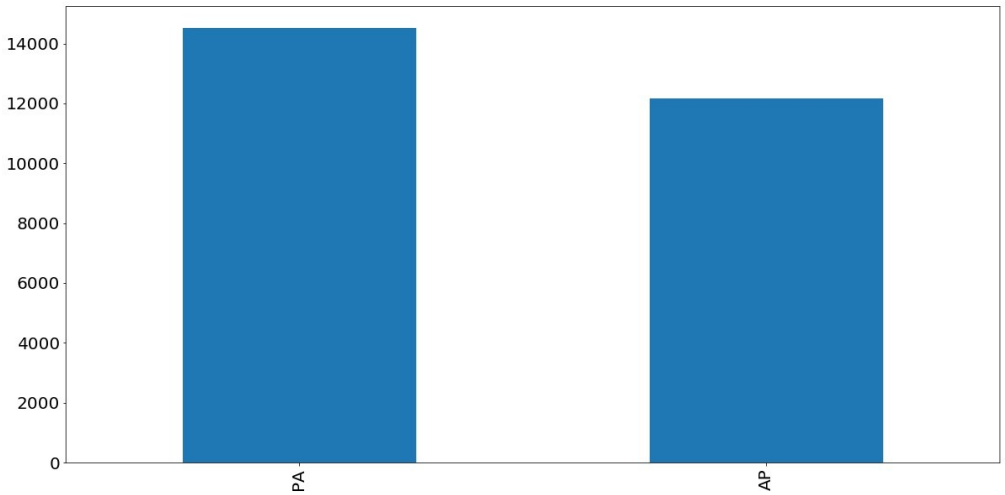
DICOM is used worldwide to store, exchange, and transmit medical images. DICOM has been central to the development of modern radiological imaging: DICOM incorporates standards for imaging modalities such as radiography, ultrasonography, computed tomography (CT), magnetic resonance imaging (MRI), and radiation therapy. DICOM includes protocols for image exchange (e.g., via portable media such as DVDs), image compression, 3D visualization, image presentation, and results reporting.

DICOM images inherently contain a lot of patient data. This data is extremely useful in doing data exploration of the dataset. Let us take a look at various kinds of information that we have gathered from it.



Approximately 20000 CXR’s do not have Pneumonia and 5000 CXR’s do have pneumonia.

Of the 26000 chest x-ray images that were present in our dataset only 5000 something are those that have tested positive for pneumonia. The data is skewed.



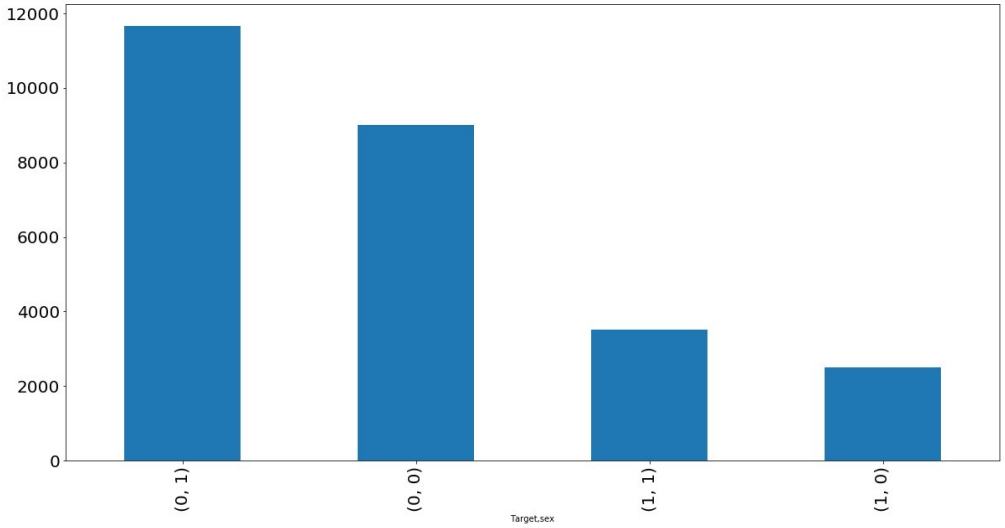
Whether the x-ray was taken in AP or PA mode

Chest x-rays are done in two views:

- Anteroposterior: From front to back. When a chest x-ray is taken with the back against the film plate and the x-ray machine in front of the patient it is called an anteroposterior (AP) view.
- Posteroanterior (PA) chest view is the most common radiological investigation in the emergency department

The data has a fairly even distribution of AP and PA chest x-ray views.

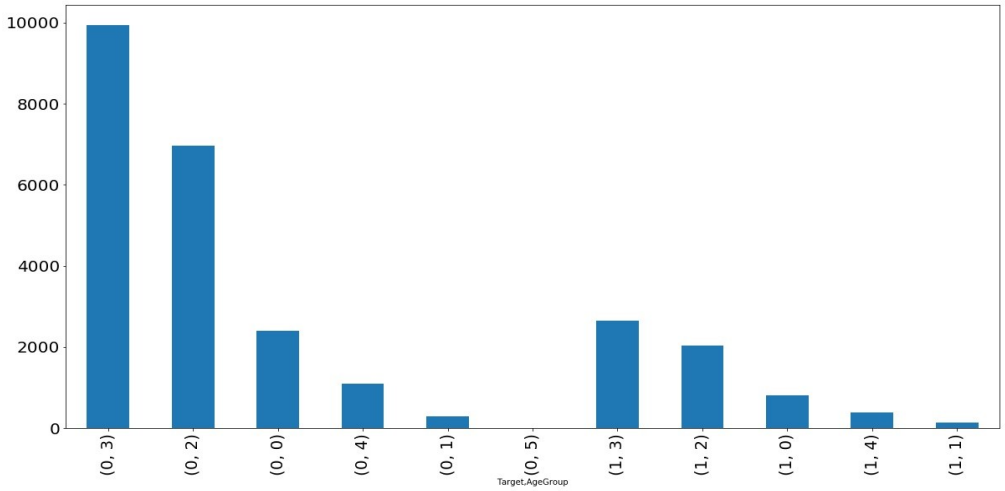
The data also is somewhat similarly distributed equally distributed for both male and female patients. But we see that of those that are affected by pneumonia the gender percentage of men is higher.



Amount of males vs females are affected.

We then divide the whole data according to age group into 5 parts:

- Infant (Age 0-10, group-0)
- Teen (Age 11-20,group-1)
- Adult (Age 21-30,group-2)
- Middle Aged (Age 31-50,group-3)
- Old (Age 51-100,group-4)



Data shows that it mostly affects the elderly that are present in group 3.

As we can see from the data that is given to us. Pneumonia mostly affects the

population that belongs to group 3 or 2. A lot of cases belonging to group 0 i.e. kids also show up.

DATA PREPROCESSING

The data had to changed from .DICOM images to .png images so that they could be fed into a CNN. Each image was reduced to 64x64 pixels only due to constraints of memory. For image preprocessing we only stuck to doing histogram equalization. Any kind of data augmentation that was required for the CNN was done by using the ImageDataGen method of keras. It will explained in detail later.

The whole data of the images was stored as a numpy array to make it easier to to load it into the cloud. We used google CoLab to train our models.

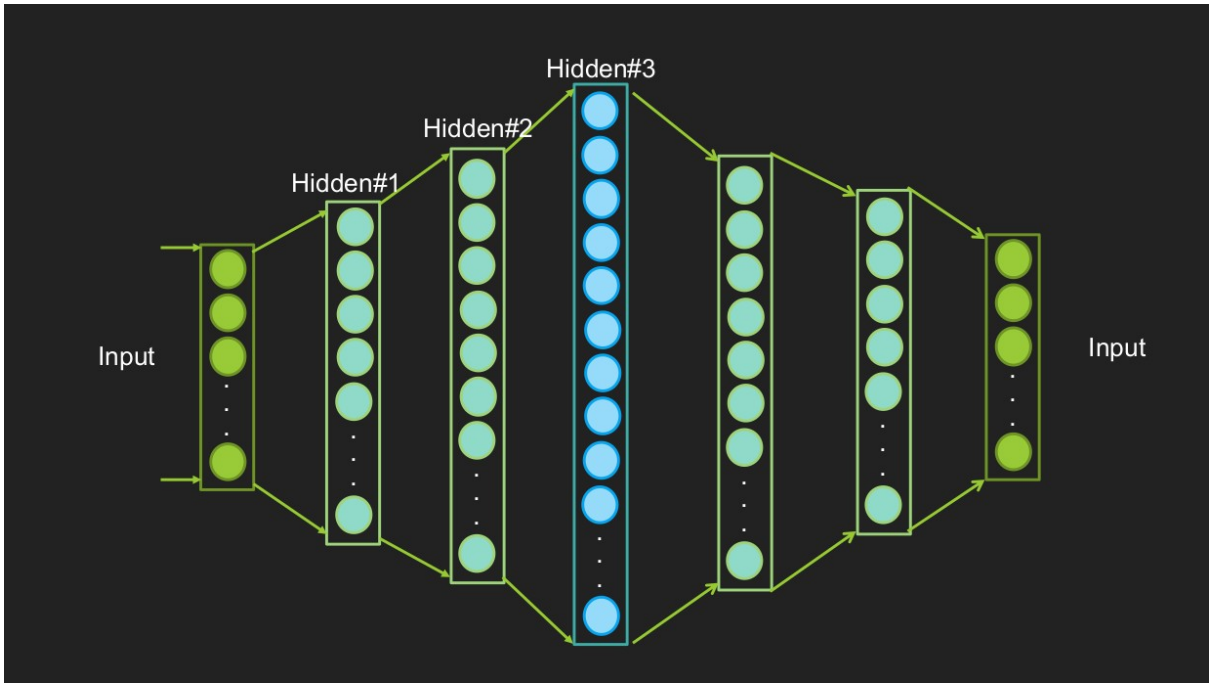
Let us now take a look at autoencoders and CNN’s in detail.

AUTOENCODERS

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer **h** that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function **h = f(x)** and a decoder that produces a reconstruction **r = g(h).**

If an autoencoder succeeds in simply learning to set **g (f(x)) = x** everywhere, then it is not especially useful. Instead, autoencoders are designed to be unable to learn to copy perfectly. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.

The structural representation of an auto-encoder is shown below.



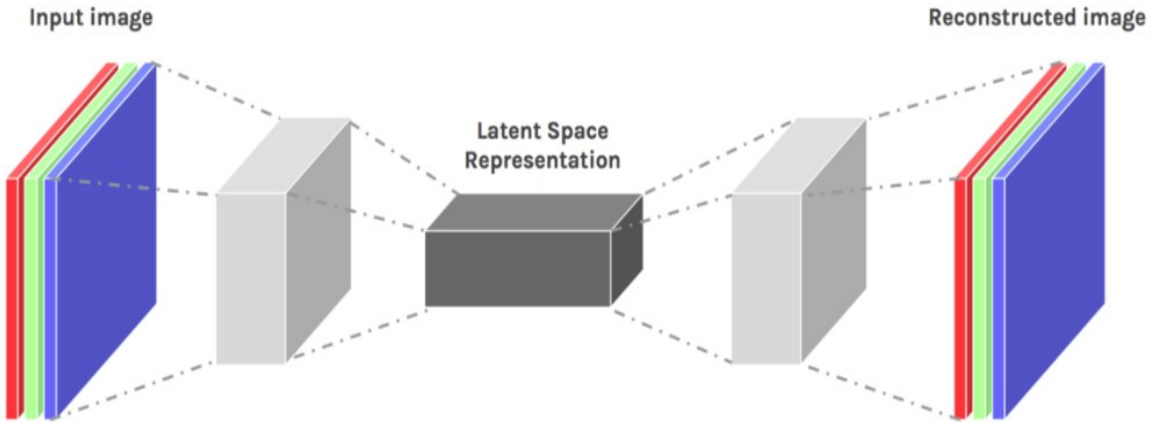
Stacked Autoencoder Architecture

We could use the above stacked autoencoder by flattening our images into vectors to get the feature vector representation of the said image. But it would not be that great for a simple reason that the image loses its structural

representation. This problem is solved by introduction of convolutional autoencoders.

CONVOLUTIONAL AUTOENCODERS

In the traditional architecture of autoencoders, it is not taken into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders (CAE), on the other way, use the convolution operator to accommodate this observation. Convolution operator allows filtering an input signal in order to extract some part of its content. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them.



Convolutional Auto Encoder for Image Data

The latent space representation is what will be used as input to the classification function. A similar kind of Convolutional Auto-Encoder is used here to extract the feature vectors from it. The CAE(Convolutional Auto-Encoder) tries to create a 256 dimensional representation of each image such that the representation best describes the CAE.

- The Encoder Layer : Consists of 4-Conv2D layers interleaved with MaxPooling2D layers. The Conv2D layers have filters each of size 3x3 and the number of filters decreases in the order 128,64,32 and 16. The MaxPooling2D layers use a stride of 2 by default and has a pooling of

(2,2). Gives us a 256-dimensional output.

- The Decoder Layer :Uses the 256 dimensional output of encoder layer as input. Consists of 4-Conv2D layers interleaved with UpSampling2D layers. The Conv2D layers have filters each of size 3x3 and the number of filters increases in the order 16,32,64 and 128. The UpSampling2D layers use a stride of 2 by default and has a upsampling of (2,2).

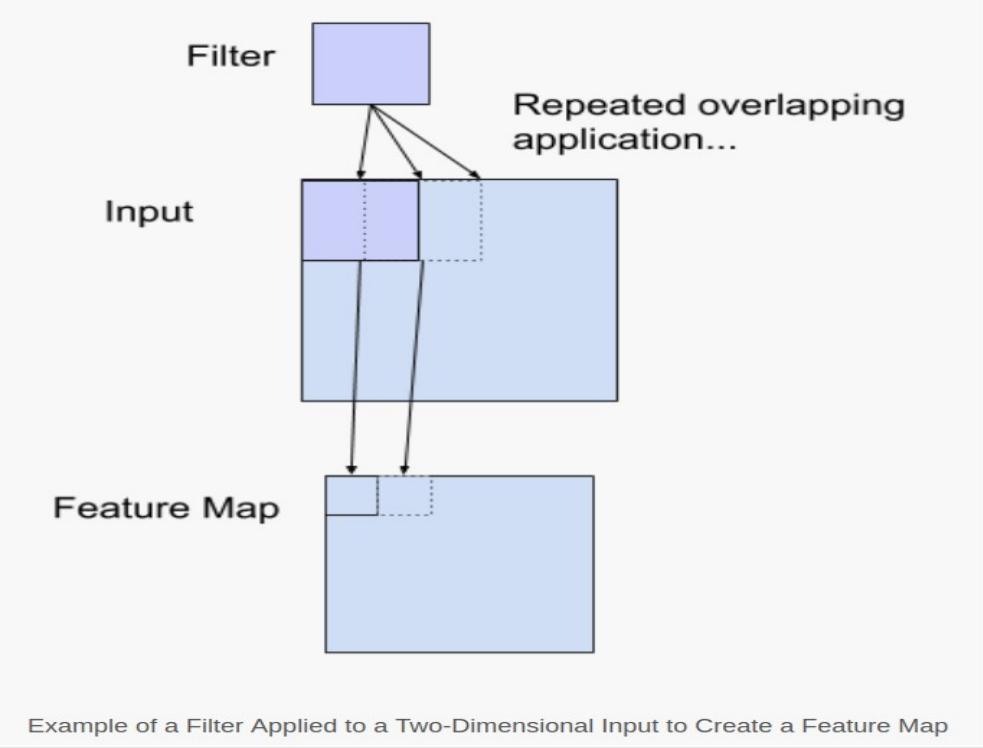
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 1)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	1280
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	73792
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 32)	18464
max_pooling2d_3 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_4 (Conv2D)	(None, 8, 8, 16)	4624
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 16)	0
conv2d_5 (Conv2D)	(None, 4, 4, 16)	2320
up_sampling2d_1 (UpSampling2	(None, 8, 8, 16)	0
conv2d_6 (Conv2D)	(None, 8, 8, 32)	4640
up_sampling2d_2 (UpSampling2	(None, 16, 16, 32)	0
conv2d_7 (Conv2D)	(None, 16, 16, 64)	18496
up_sampling2d_3 (UpSampling2	(None, 32, 32, 64)	0
conv2d_8 (Conv2D)	(None, 32, 32, 128)	73856
up_sampling2d_4 (UpSampling2	(None, 64, 64, 128)	0
conv2d_9 (Conv2D)	(None, 64, 64, 1)	1153

Total params: 198,625
Trainable params: 198,625
Non-trainable params: 0

Convolutional Auto encoder Architecture implemented for feature extraction of x-rays.

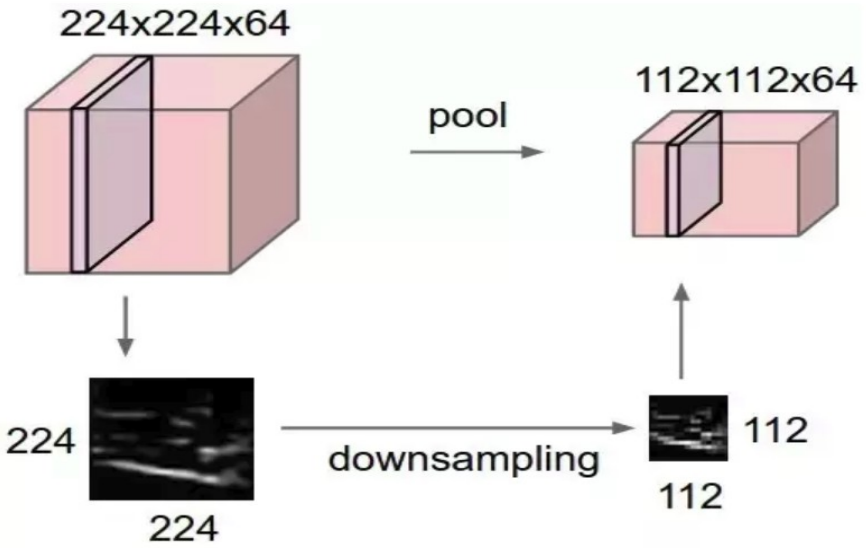
Understanding various layers of the CAE :

- Convolutional Layer : The convolution layer comprises of a set of independent filters. Each filter is independently convolved with the image and we end up with feature maps. One for each king of filter.



Convolutional Layer Working

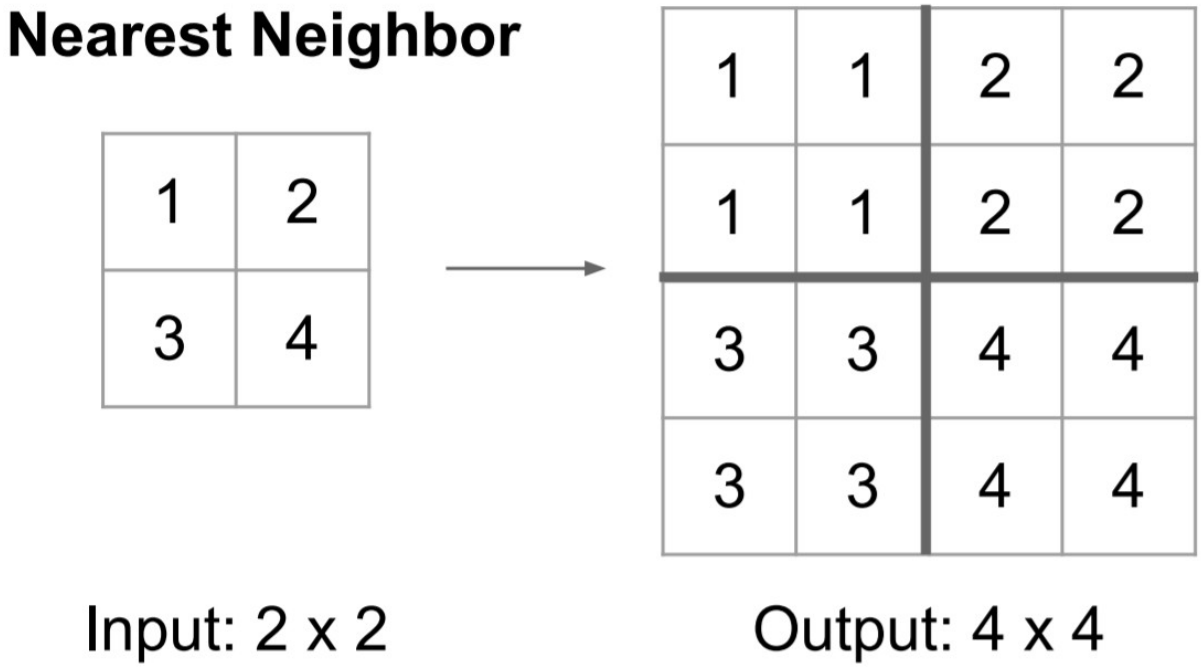
- Max Pooling Layer : Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. In MaxPooling2D filter we take the maximum value out of 2x2 pixels.



Max Pooling 2D with stride 2 and filter 2x2

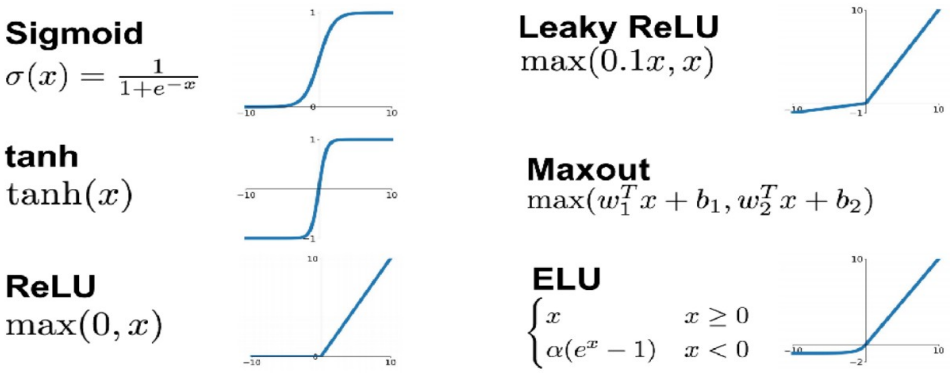
- Up Sampling Layer : Upsampling is a technique for increasing the size of an image. It is basically like interpolation. Keras by default uses nearest

neighbour upsampling.



Upsampling 2D with stride 2x2 and nearest neighbour interpolation

- Activation Function : The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output. We use **relu** as it is a good default activation function. Various kinds of activation functions are listed below.

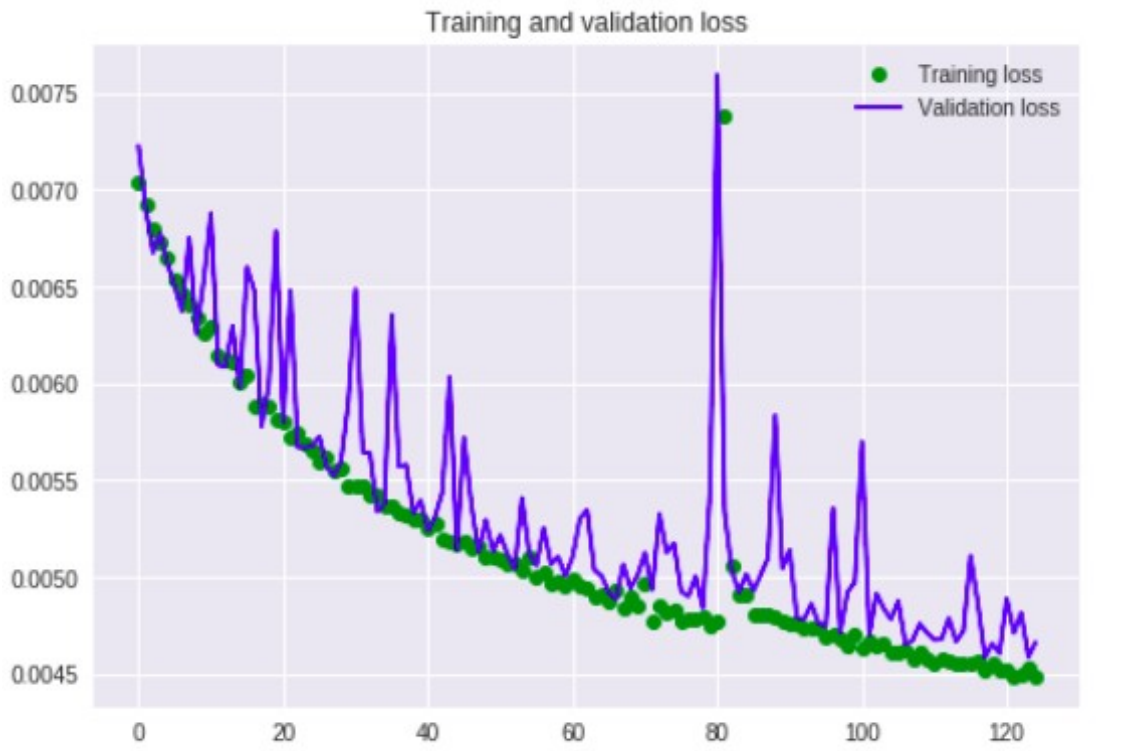


Various Activation Functions

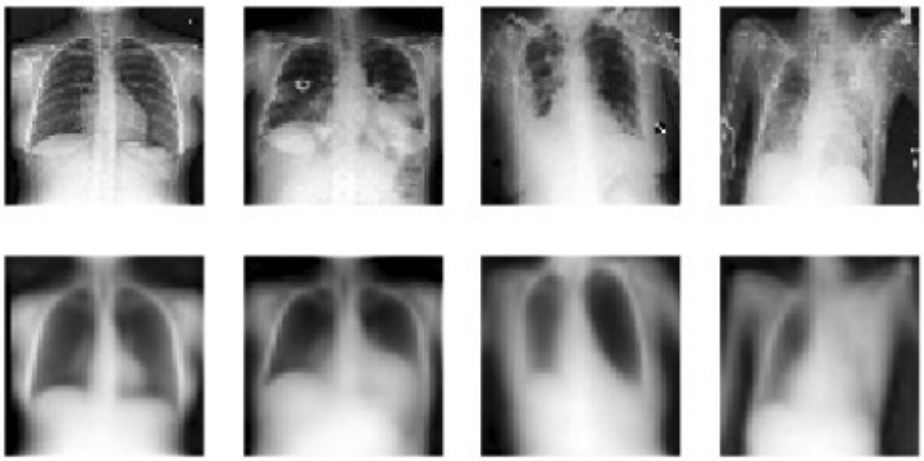
TRAINING THE CONVOLUTIONAL AUTO-ENCODER

Of the 26000 data instances we set 21000 instances for training and the rest were set to the validation set. The model uses de-facto “adam” optimizer as it seems to just work better in most of the cases. The input data is normalised and reshaped into 64x64x1 image matrix so that it can be fed into the CAE.

The model is trained for over 125 epochs. Each batch is of size 128 as is a standard in deep learning models. The loss curve is as shown below. We obtain a really low mse loss. But we actually want to use the learned features to classify the chest x-rays.



The autoencoder is trained in such a way that it can copy it’s input. The figure below shows how well the auto encoder has learned it’s encoded values.



TOP ROW : Original Images
BOTTOM ROW : Recreated Images

The autoencoder learns almost all the important features. It does not encode data related to the bones but rather focuses more on the shape and orientation of lungs and the most important feature that it encoded is whether the lungs have any type of cloudiness or not. If the lungs re clouded then it is a sign of Pneumonia.

Now the trained autoencoder has a 256 dimensional feature vector for each image. This feature vector set is used as input features to a classifier.

RESULTS

The next step is to train a classifier. Two classifier's were trained and both gave similar results. The training set consisted of the feature set extracted from the autoencoder. We train it on same number of instances as that of the autoencoder.

The accuracy achieved on the validation data is as given below.

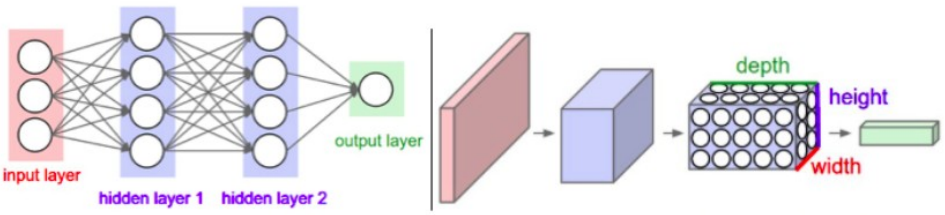
Model	Accuracy on Validation Data
Logistic Regression	82.12 %
SVM Classifier	81.92 %

The results are in no way state of the art. The next logical step is to use a CNN to try to classify the chest x-rays.

CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer—the output layer—that represent the predictions.

Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension. The figure below will better explain how we can separate regular neural network from CNN.

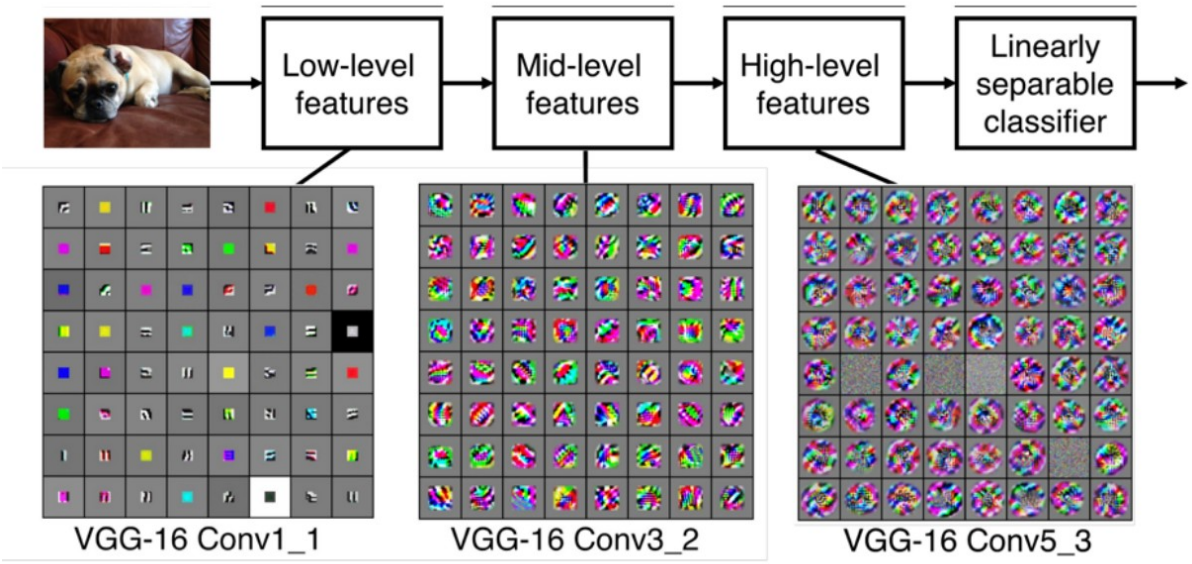


Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Normal NN vs CNN.—Source: http://cs231n.github.io/convolutional-networks/

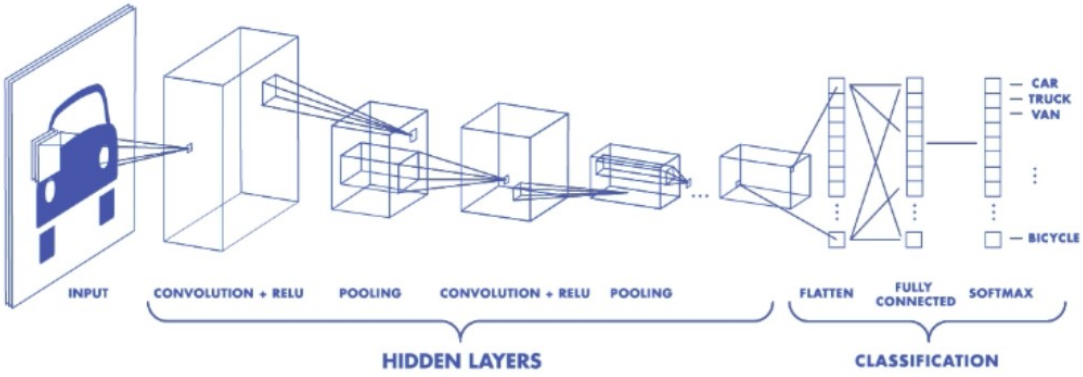
The CNN can be divided into two parts :

- **The Hidden Layer/ Feature Extraction Part** : In this part, the network will perform a series of convolutions and pooling operations during which the features are detected.
- **The Classification Part** : Here, the fully connected layers will serve as a classifier on top of these extracted features. The FC layer will assign a probability for the object on the image being what the algorithm predicts it is.



CNN as a feature extractor

The final output of a ConvNet is flattened and then is fed into a fully connected layer(Multi Layer Perceptron) to do classification. As we run through each epoch for training the CNN the network learns both the feature extractors and the weights required for classification together. This is what makes CNN so powerful. There is no need to define any kind of filter banks in a CNN. It learns those filters on it’s own.



Architecture of a CNN.—Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

A Typical CNN Architecture. Hidden Layers + Classifier

PROPOSED CNN ARCHITECTURE

The following architecture was used to help classify the chest x-ray images. Apart from the previously explained Conv2D, MaxPooloing2D and Activation functions there are a few more layers.

- Batch Normalization : Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift). Let's say an algorithm learned some X to Y mapping, and if the distribution of X changes, then we might need to retrain the learning algorithm by trying to align the distribution of X with the distribution of Y. To increase the stability/robustness of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.
- Flatten : Flattens the input matrix(MxN) into a vector so MN dimensions. This flattened vector is used as input to the fully connected layers for classification.

```
# The Model
input_img = Input(shape=(64, 64, 1))  # adapt this if using `channels_first` image data format

# Conv Layer 1
x = Conv2D(64, (3,3), padding = 'same')(input_img)
x = BatchNormalization(axis = 3)(x)
x = Activation('relu')(x)
# Conv Layer 2
x = Conv2D(32, (3,3), padding = 'same')(x)
x = Conv2D(32, (3,3), padding = 'same')(x)
x = Conv2D(32, (3,3), padding = 'same')(x)
x = BatchNormalization(axis = 3)(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size = (2,2)) (x)
# Conv Layer 1
x = Conv2D(64, (3,3), padding = 'same')(x)
x = Conv2D(64, (3,3), padding = 'same')(x)
x = Conv2D(64, (3,3), padding = 'same')(x)
x = BatchNormalization(axis = 3)(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size = (2,2)) (x)

# Conv Layer 1
x = Conv2D(32, (3,3), padding = 'same')(x)
x = Conv2D(32, (3,3), padding = 'same')(x)
x = Conv2D(32, (3,3), padding = 'same')(x)
x = BatchNormalization(axis = 3)(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size = (2,2)) (x)


# Dense
x = Flatten()(x)
x = Dense(1024, activation = 'relu')(x)
x = Dropout(0.4)(x)
x = Dense(512, activation = 'relu')(x)
x = Dropout(0.4)(x)
x = Dense(512, activation = 'relu')(x)
x = Dropout(0.4)(x)
x = Dense(256, activation = 'relu')(x)
outputs = Dense(2, activation='softmax')(x)




cnn_classifier = Model(input_img, outputs)
#adam = optimizers.Adam(lr=0.0003, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=F
#sgd = optimizers.SGD(lr=0.001, decay=1e-4, momentum=0.8, nesterov=True)
cnn_classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

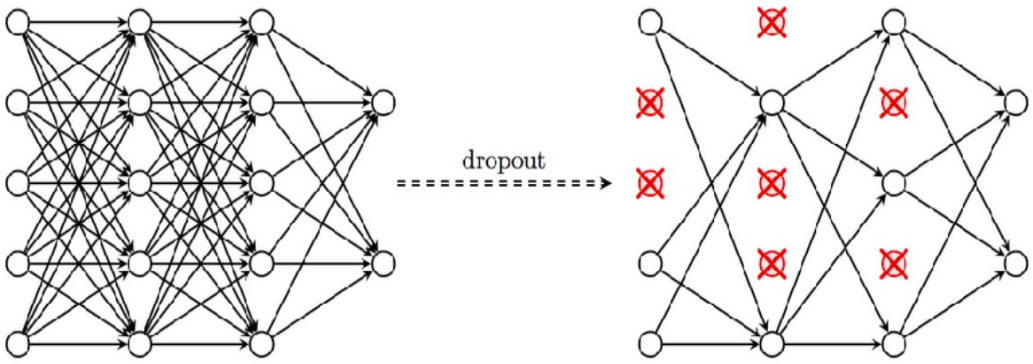
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

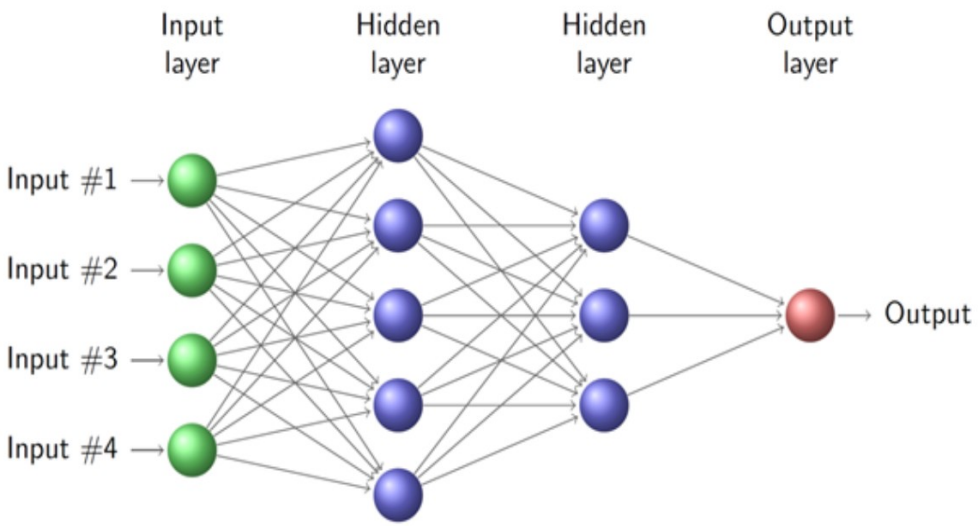
- Dropout : Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "*dropped out*." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer.



DROPOUT

- Dense Layer : A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous

layer, thus densely connected. The layer has a weight matrix **W,** a bias vector **b,** and the activations of previous layer **a.**



Multiple Dense Layer's stacked on top of another

The final dense layer must consist of the number of classes that has to be classified as it's output. We use the standard "adam" optimizer and "categorical_crossentropy" function as our loss function.

TRAINING THE CNN

The input to the CNN is same as that to the auto encoder. In the CNN the 15000 instances are taken for training and the rest kept aside for validating data. The training was not straightforward. Each batch of size 128 was passed through the ImageDataGenerator Class with parameters that help in data augmentation.

Rather than performing the operations on the entire image dataset in memory, the API is designed to be iterated by the deep learning model fitting process, creating augmented image data for the model just-in-time. This reduces the memory overhead, but adds some additional time cost during model training. Another technique that was used was a learning rate scheduler. Rather than using the same learning rate everytime, the learning rate is reduced as we progress through the epochs. Learning rate schedules seek to adjust the learning

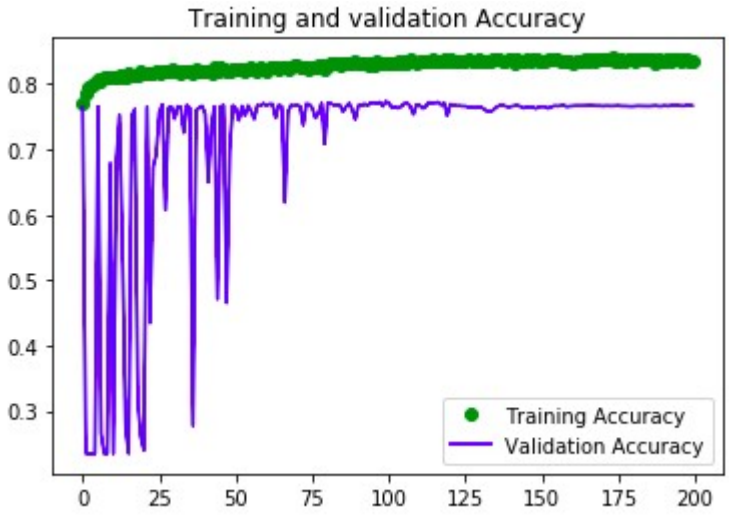
rate during training by reducing the learning rate according to a predefined schedule. Common learning rate schedules include time-based decay, step decay and exponential decay. The scheduler starts with a learning rate of 1e-3. It uses a time-based decay. We also use a function that reduces our learning rate in case our accuracy plateaus/ remains same for more than 5 epochs.

The CNN was trained for 200 epochs. The accuracy seemed to stabilise after 175 epochs. The results are documented below.

RESULTS

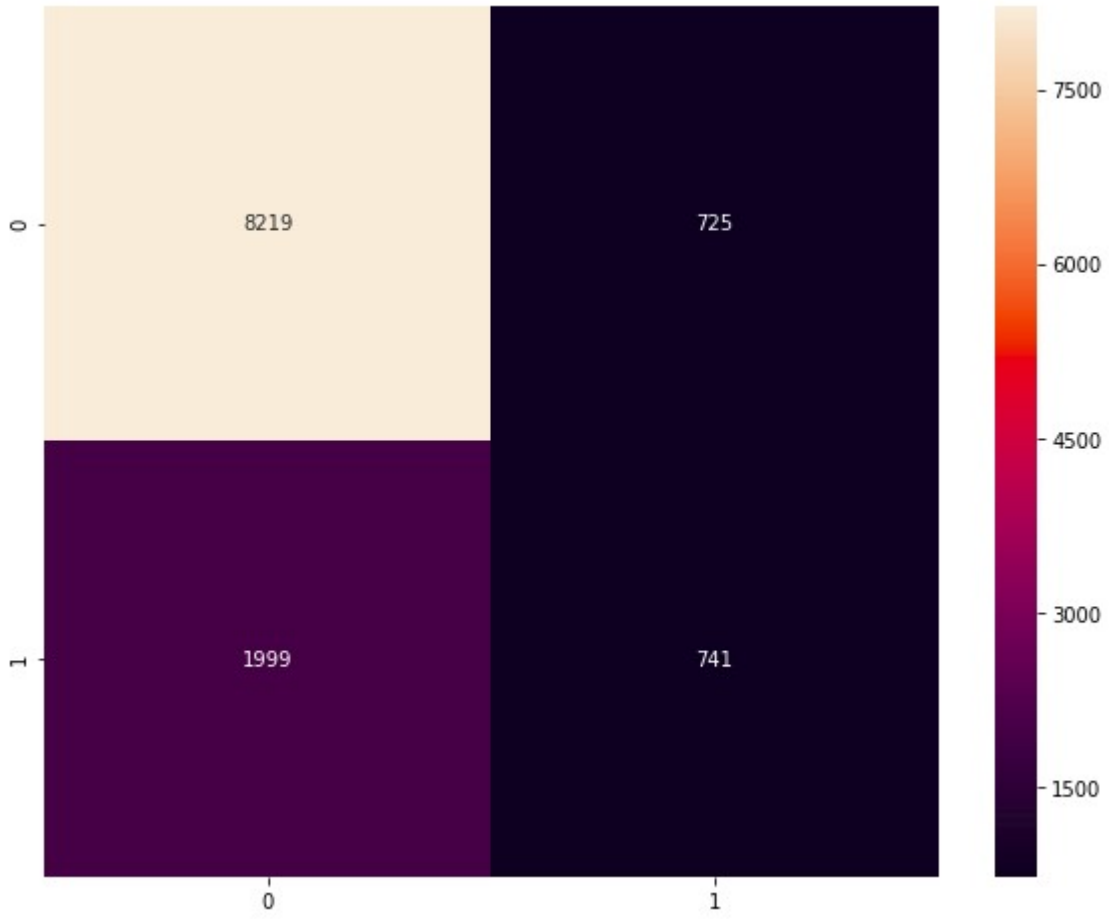
The accuracy observed was not that great and one probable reason could be that the model did not get enough data to learn all the parameters. Maybe augmenting the data and adding it to the dataset may give us better results. Another reason could be that the number of Chest X-Rays which were positive for pneumonia were only a small fraction of the total number of x-rays.

The Accuracy curve is as shown below.



Training Accuracy	83.66%
Validation Accuracy	76.69%

The confusion matrix is as shown below.



The x-axis is for predictions and the y-axis for actual values.

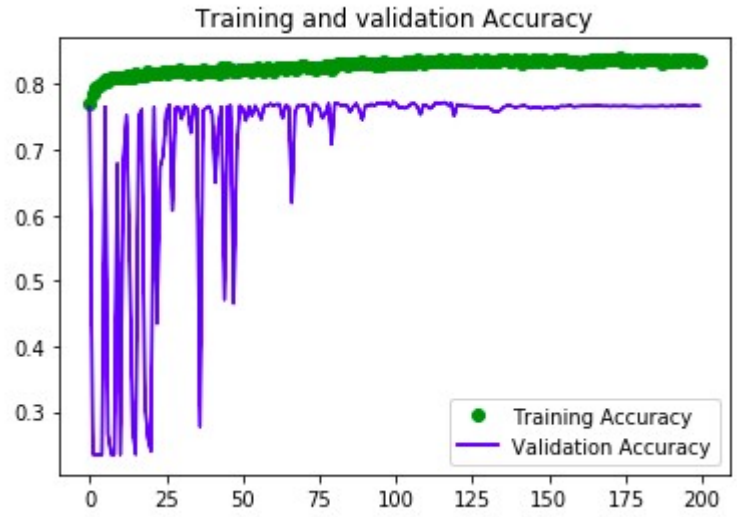
MisClassification Rate	0.23
True Positive Rate/Recall/Sensitivity	0.27
False Positive Rate	0.08
True Negative Rate/ Specificity	0.919
Precision	0.50

We observe that we get a very high TNR. This is due to the fact that we correctly classify those x-rays that do not have Pneumonia. But since we lack data on the number of x-ray's that do have pneumonia, the Recall is low. The FPR is also low. This means that the algorithm is not able to properly classify the chest x-rays with pneumonia. This could be due to low number of images pertaining to pneumonia. The misclassification rate is low which is a good sign.

ALTERNATE APPROACH - BALANCING THE DATASET

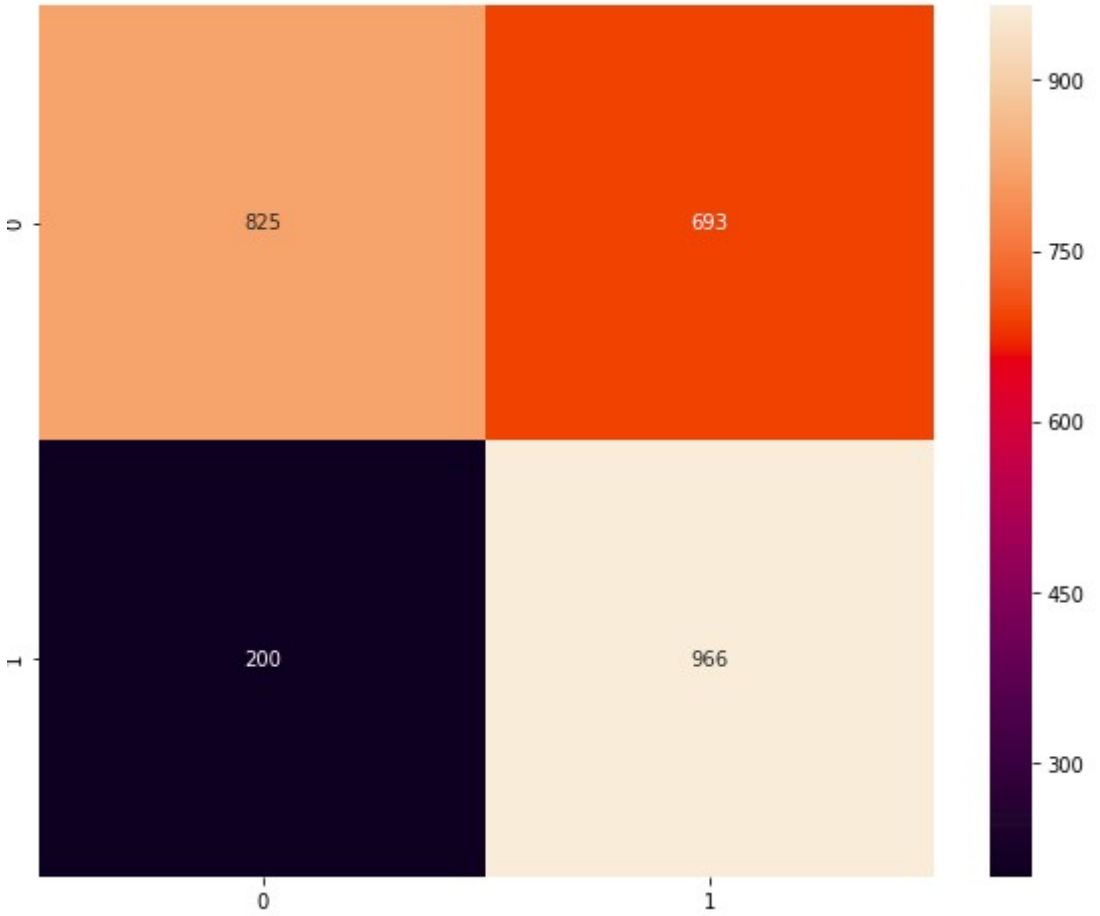
Since the dataset is extremely unbalanced, as an experiment I tried training a CNN by reducing the number of x-rays that do not contain pneumonia traces. This made the data set more balanced as it had comparable number of images from both the classes.This did not yield in a good result.

The accuracy curve is as below:



Training Accuracy	77.95%
Validation Accuracy	66.73%

The confusion matrix is as follows:



The x-axis is for predictions and the y-axis for actual values.

MisClassification Rate	0.33
True Positive Rate/Recall/Sensitivity	0.828
False Positive Rate	0.41
True Negative Rate/ Specificity	0.54
Precision	0.58

As we can see from the confusion matrix, we get a higher recall. And this is because the data is bore balanced now. But the whole network still needs a lot of data to train and we see that we have a lower precision and a somewhat higher misclassification rate.

CONCLUSION
&
FUTURE SCOPE OF WORK

Although the results are not state of the art, but these models show that deep learning is the way to go for classification of images. With a more balanced and larger dataset and with help of better hardware far better results could have been achieved.

Future work could include using transfer learning and implementing more non traditional but resource hungry networks like ResNET, Inception NET and DenseNET architectures. These architectures give state of the art accuracy on the ImageNET dataset. Transfer learning techniques can be used to fasten up the computation and use pre trained architectures as feature extractors before using dense layers for classification.

REFERENCES

- Understanding CNN
- AlexNET
- AutoEncoders
- Convolutional Auto Encoder
- ChexNET
- Densely Convoluted Neural Network
- Chest X-ray dataset
- CS231n
- Dropout: A Simple Way to Prevent Neural Networks from Overfitting