Machine Learning Engineer Nanodegree

Capstone Project

Nitesh Vachhani
September 30th , 2019

I. Definition

Project Overview

Pneumonia is a form of acute respiratory infection that affects the lungs. The lungs are made up of small sacs called alveoli, which fill with air when a healthy person breathes. When an individual has pneumonia, the alveoli are filled with pus and fluid, which makes breathing painful and limits oxygen intake.

Pneumonia is the world's leading cause of death among children under 5 years of age, accounting for 16% of all deaths of children under 5 years old killing approximately 2,400 children a day in 2015. There are 120 million episodes of pneumonia per year in children under 5, over 10% of which (14 million) progress to severe episodes. In the US, pneumonia is less often fatal for children, but it is still a big problem. Pneumonia is the #1 most common reason for US children to be hospitalized. [1]

There were an estimated 880,000 deaths from pneumonia in children under the age of five in 2016. Most were less than 2 years of age.

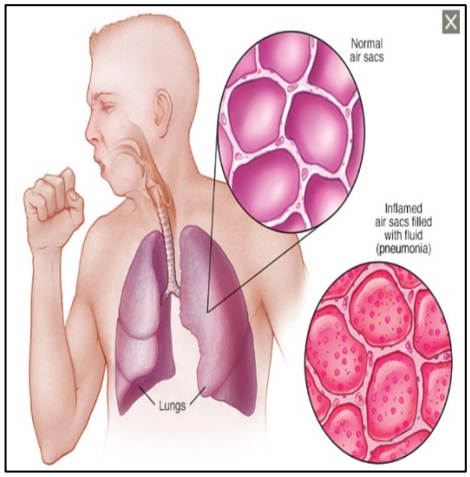


Fig 1: Details of human lung

Diagnosing pneumonia early is key to faster and healthy recovery. Pneumonia is typically diagnosed based on a combination of physical signs and a chest X-ray. Pneumonia usually manifests as an area or areas of increased opacity in an X-ray. However, the diagnosis of pneumonia from x-ray is complicated because of a number of other conditions in the lungs such as fluid overload (pulmonary edema), bleeding, volume loss (atelectasis or collapse), lung cancer, etc.

In developing countries like India, rural health services is a serious problem, with a shortage of qualified healthcare providers as a major cause of the unavailability and low quality of healthcare. Applying image-based machine diagnostic medical techniques could improve healthcare outcomes in rural areas of developing countries.

Citation of work in this field: https://www.cell.com/cell/fulltext/S0092-8674(18)30154-5

Problem Statement

Applying image processing techniques on medical images can offer an objective opinion to improve efficiency, reliability, and accuracy in the medical diagnostics process.

The problem being targeted for the capstone project is to **build an efficient and accurate machine learning model to detect if a child is suffering from pneumonia using chest x-ray images**.

Building a such a model facilitate more accurate and earlier diagnostics which in turn will help in faster patient recovery and increase the chances for successful treatment.

Metrics

The goal is to solve a binary classification problem. Confusion matrix is one of the most commonly used techniques for summarizing the performance of a classification model. Both the benchmark model and the solution model will be evaluated using the confusion matrix. More specifically, precision, recall and f-score for the models will be calculated which will serve as the evaluation metrics.

Confusion Matrix - A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. Using a confusion matrix not only into the errors being made by a classifier but more importantly the types of errors that are being made.

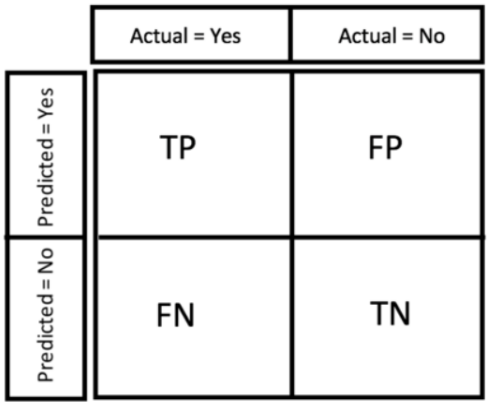


Fig 2: Confusion Matrix

- **True Positives (TP):** when the actual class of the data point was 1(True) and the predicted is also 1(True)

- **True Negatives (TN):** when the actual class of the data point was 0(False) and the predicted is also 0(False)

- **False Positives (FP):** when the actual class of the data point was 0(False) and the predicted is 1(True).

- **False Negatives (FN):** When the actual class of the data point was 1(True) and the predicted is 0(False).

Recall - Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN). Our model needs to ensure that it shouldn't predict a sick person as healthy i.e. false negatives should be low.[1] Diagnosing a sick kid as healthy can have dangerous consequences False negative is a unique characteristic for recall. The higher the false negative the lower the recall. Hence getting a very high recall would be the first goal/metric for the model.
Recall is given by the relation:

$$Recall = \frac{TP}{TP + FN}$$

Precision - To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labelled as positive

is indeed positive (small number of FP). Diagnosing a healthy kid as sick will not have as dangerous consequences but is still important especially in third-world countries where cost of medication and treatment is high. Low precision would reduce the confidence of people in this model. Hence, precision is also an important measurer for this model.

Precision is given by the relation:

$$\textbf{\textit{Precision}} = \frac{TP}{TP + FP}$$

F-Beta score – F1 score is a metric which takes into account both , precision and recall. It is the harmonic mean of precision and recall. It tells us about the balance that exists between precision and recall. As with our case, we want to have high precision and high recall but higher emphasis needs to be given to a model with higher recall. This is done using the F-Beta score metric where beta>1. This will assign more weight to recall than precision and will be an important metric for our model evaluation. For our analysis we will keep the beta value as 2.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

II. Analysis

Data Exploration

The dataset is basically a set of chest x-ray images. The data set is taken from Kaggle and is available here - https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Below are few of the important details about this dataset.

- The dataset contains 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal). The dataset itself is organized into 3 folders of train, test and validate.
- Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of paediatric 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 added to the dataset. In order to account for any grading errors, the evaluation set was also checked by a third expert.
- Distribution of the dataset. Below is a bar graph and statistics of the of the distribution of the images. As it is evident from the below information the number of images for pneumonia cases is almost three times the number of images available for normal cases. This means the dataset is imbalanced. Different techniques like data augmentation and relative weight assignment (details in data pre-processing section)will be performed to compensate for the imbalance.

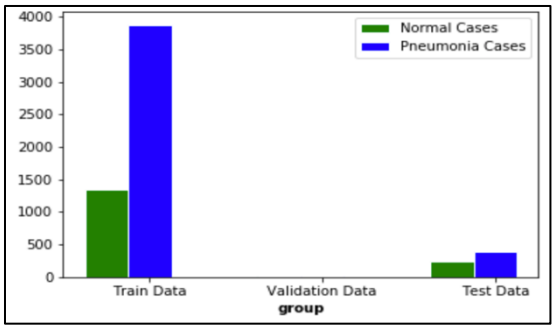


Fig 3 : Pie chart of dataset counts

Class	Training dataset	Validation Dataset	Test Dataset
Normal Cases	1342	8	234
Pneumonia Cases	3876	8	390

- Few sample Images from the data-set for both classes are extracted below. After going through samples it looks like many of the images are a bit zoomed. Many of the images, as also shown below cover a bit more than the lungs in the images. These darkened region (which is mostly on the sides), will offer no extra information/features to help in the classification. Both these issues will be handled as part of pre-processing the input.

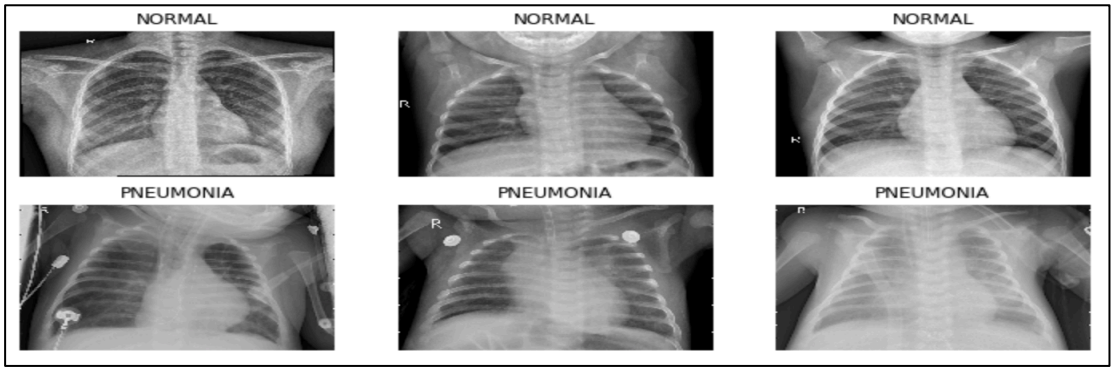


Fig 4: Sample images from dataset

Exploratory Visualization

The input data is a set of images with corresponding class classification. There are no other accompanying features. Hence in order to explore and extract features, we will try to perform image operations and try to extract information from the same. Below is the list of operations performed.

a. Checking the width and height of the images with respect to each class.

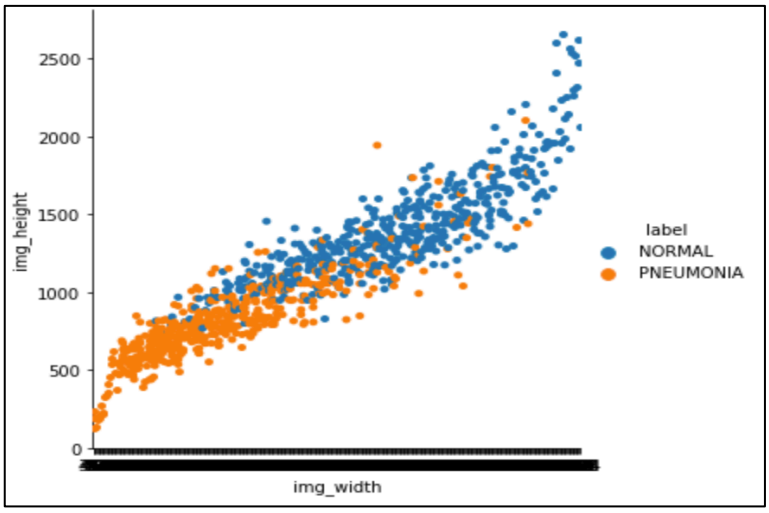


Fig 5 : Height – Width Distribution chart per class

b. Checking corresponding grey scale images, pixel histograms as well as RGB pixel histogram. Histogram is a graphical representation of the intensity distribution of an image. We will be looking at a few of the sample images and apply these operations on it to understand the underlying dataset.

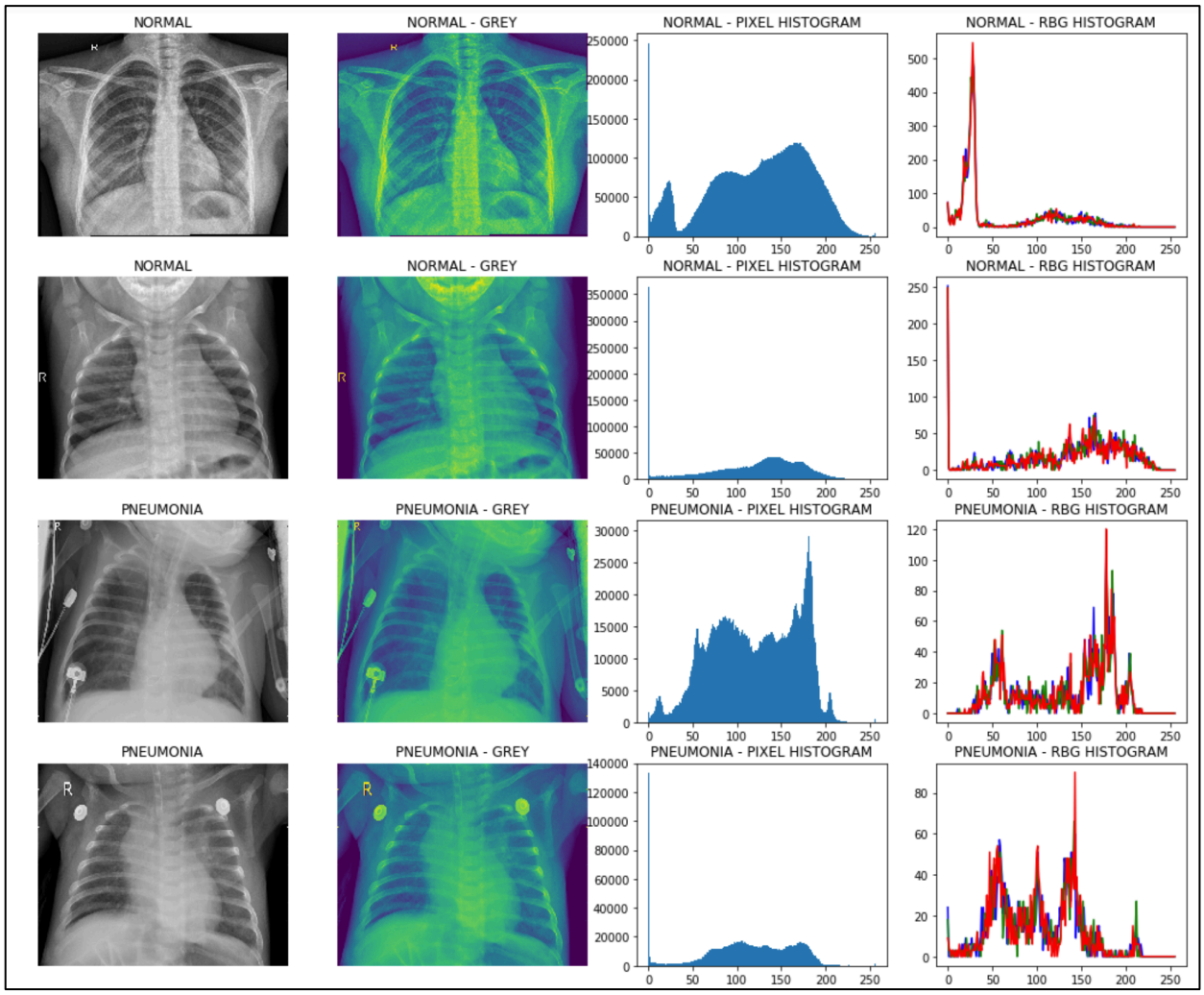


Fig 6 : Sample Images , pixel histogram and channel wise histogram

c. Extract the common properties of images and build a correlation matrix by class. This visualization will help us understand if there is any co-relation between these image features or correlation of any of the features to the output class. These image properties extracted are as follow (a smaller subset was taken to avoid Out of Memory error caused due to loading all images in RAM.)
    a. Average pixel value across the three channels (green , blue, red)
    b. Standard Deviation of pixel values across the three channels.
    c. Brightness of image.
    d. Saturation of image.

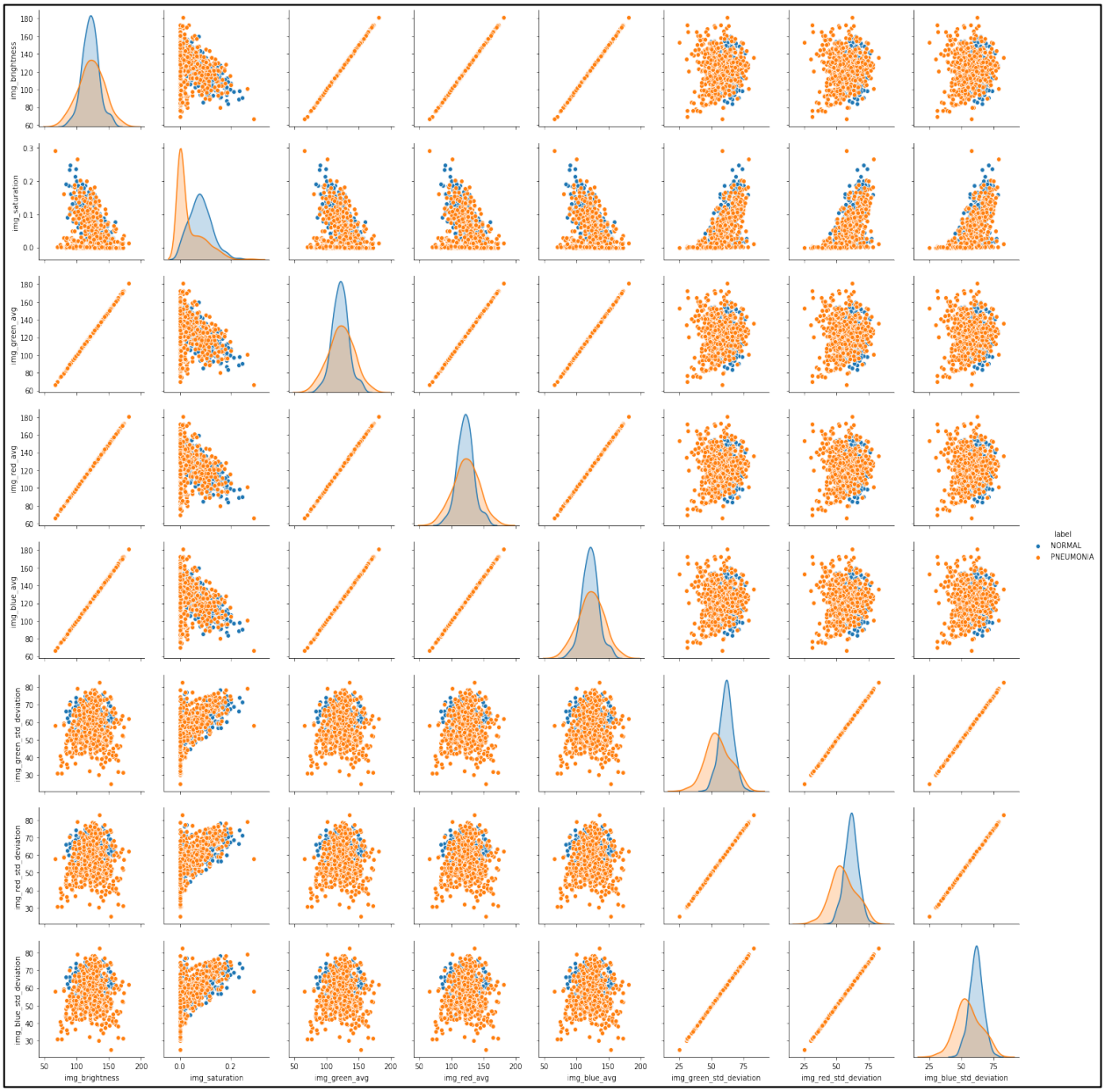


Fig 7 : Co-relation pair plot of the differnet image features (subset of total images taken for plot)

Major Observations from Visual Exploration

- The size of the images(height and width) in the dataset is highly imbalanced. The images will need to be scaled to a common value before being provided as input to the model.
- Image histograms show the imbalanced intensity distribution. Histogram equalization will need to be applied (details in Data Pre-processing section) to improve the contrast of the image. Better contract will help the underlying model extract features better.
- Looking at the different image properties, there does not seem any obvious co-relations. The values seem to be scattered throughout and are largely overlapping for both the classes of the dataset. Convoluted Neural Networks can be thought of as automatic feature extractors from the image. They covers local and global features. They also learn different features from images. Hence by using CNNs we don't have to worry about extracting image properties; they will be extracted by CNN models itself.

Algorithms and Techniques

To build the machine learning model, initially a simple Convolutional Neural Network will be built and tested/validated against the metrics. This simple convolution neural network model will be used as a baseline and new CNN model using Transfer Learning will be created. This newer CNN model will refined further to achieve good metrices. Below are the details on some of the algorithms/techniques used.

1. **Convolutional Neural Networks (CNNs)**
   Convolutional neural network are used to find patterns in an image. You do that by convoluting over an image and looking for patterns. In the first few layers of CNNs the network can identify lines and corners, but we can then pass these patterns down through our neural net and start recognizing more complex features as we get deeper. This property makes CNNs really good at identifying objects in images[1] and hence make it a good model for the solution. Below are details of some of the important concepts with respect to CNNs.

   Convolution - A convolution extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or convolved feature (which may have a different size and depth than the input feature map). Convolutions are defined by two parameters:
   - Size of the tiles that are extracted (typically 3x3 or 5x5 pixels).
   - The depth of the output feature map, which corresponds to the number of filters that are applied.

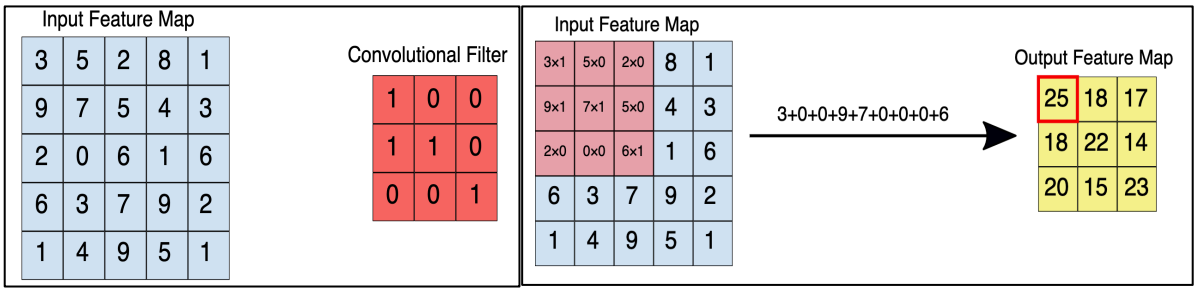


Fig 8 : The 3x3 convolution is performed on the 5x5 input feature map. In Yellow, the resulting convolved feature.

During training, the CNN "learns" the optimal values for the filter matrices that enable it to extract meaningful features (textures, edges, shapes) from the input feature map. As the number of filters (output feature map depth) applied to the input increases, so does the number of features the CNN can extract. [2]

Activation Function - The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. he activation function is the non-linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.
There are many activation functions. We will be using the two mentioned below.

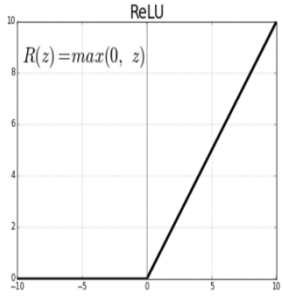


Fig 9 : ReLU Activation Function

ReLU Activation Function - ReLU function is the most widely used activation function in neural networks today. ReLU functions is that it does not activate all neurons at the same time. ReLU function converts all negative inputs to zero and the neuron does not get activated. This makes it very computational efficient as few neurons are activated per time and it does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions. [2]

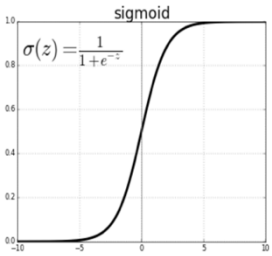


Fig 10 : Sigmoid Activation Function

Sigmoid or Logistic Activation Function - The sigmoid function exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice and is usually applied at the last layer in the network. Since we are also essentially predicting the probability if the x-ray image is normal/pneumonia, this activation function is a good fit for us.

Pooling - Pooling step is a step , in which the CNN down-samples the convolved feature (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information. A common algorithm used for this process is called max pooling.

Max pooling operates in a similar fashion to convolution. We slide over the feature map and extract tiles of a specified size. For each tile, the maximum value is output to a new feature map, and all other values are discarded.

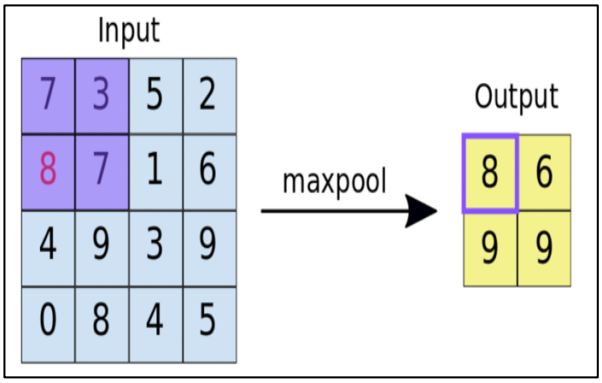


Fig 11 : Max pool operation. The Max value is picked from the filter size.

Max pooling operations take two parameters:

- Size: The size of the max-pooling filter (typically 2x2 pixels)
- Stride: the distance, in pixels, separating each extracted tile. Unlike with convolution, where filters slide over the feature map pixel by pixel, in max pooling, the stride determines the locations where each tile is extracted. For a 2x2 filter, a stride of 2 specifies that the max pooling operation will extract all nonoverlapping 2x2 tiles from the feature map.[3]

Fully Connected Layers - At the end of a convolutional neural network are one or more fully connected layers (when two layers are "fully connected," every node in the first layer is connected to every node in the second layer).

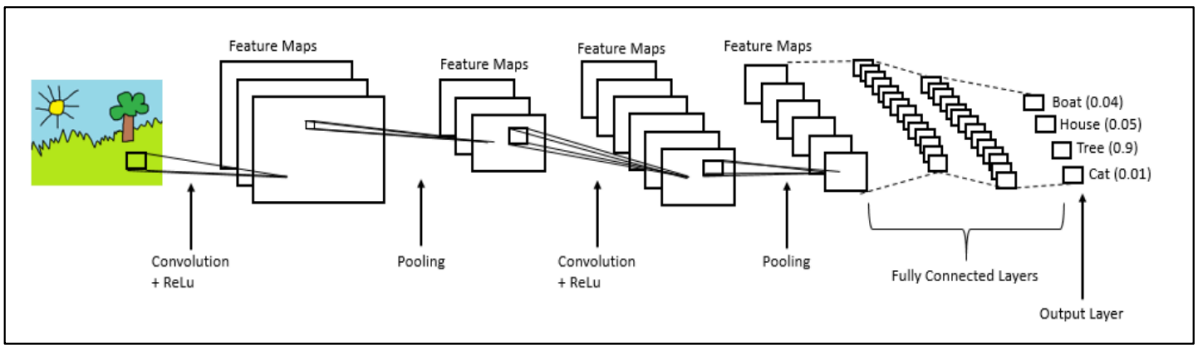
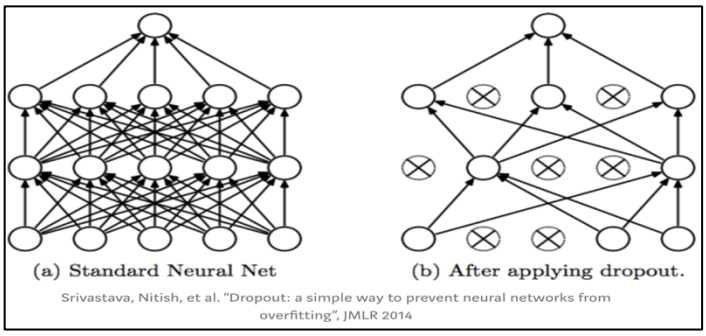


Fig 12 : Sample of a fully connected convolution network

Their job is to perform classification based on the features extracted by the convolutions. Typically, the final fully connected layer contains a softmax activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict. [3]

Batch Normalization - Batch Normalization is a technique that normalizes the mean and variance of each of the features at every level of representation during training. Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers. Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift). Batch normalization helps achieve higher learning rate and adds regularization.

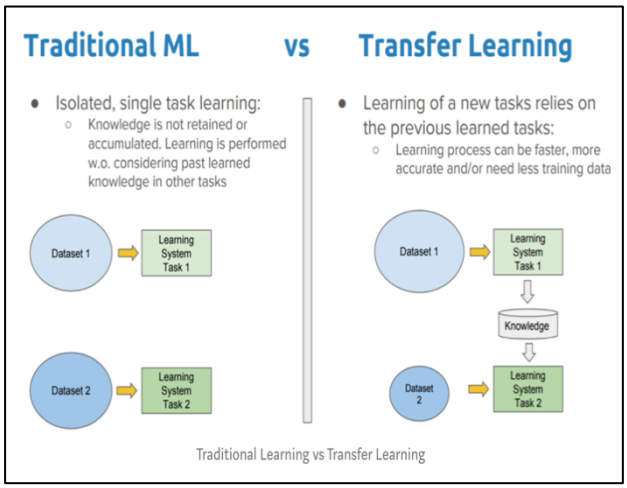
Dropout - Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By "ignoring", it means these units are not considered during a particular forward or backward pass.



Dropout layers help reduce overfitting Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.[4]

Fig 13 : Dropout Example

2. CNN with 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 second task. Transfer Learning differs from traditional Machine Learning in that it is the use of pre-trained models that have been used for another task to jump start the development process on a new task or problem. The benefits of Transfer Learning are that it can speed up the time it takes to develop and train a model by reusing these pieces or modules of already developed models. This helps speed up the model training process and accelerate results.

Fig 14: Traditional ML vs Transfer Learning

The intuition behind transfer learning is that if a model trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. A neural network is trained on a data. This network gains knowledge from this data, which is compiled as "weights" of the network. These weights can be extracted and then transferred to any other neural network. Instead of training the other neural network from scratch, we "transfer" the learned features. By using pre-trained models which have been previously trained on large datasets, we can directly use the weights and architecture obtained and apply the learning on our problem statement. This is known as transfer learning. We "transfer the learning" of the pre-trained model to our specific problem statement.

There are 3 ways in which a model can be fine-tuned using a pre-trained model

- **Train the entire model**: In this case, you use the architecture of the pre-trained model and train it according to your dataset. Here the model is learning from scratch, so it will need a large dataset (and a lot of computational power).
- **Train some layers and leave the others frozen**: Lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent). Here, we play with that dichotomy by choosing how much we want to adjust the weights of the network (a frozen layer does not change during training).

- **Freeze the convolutional base:** The main idea is to keep the convolutional base in its original form and then use its outputs to feed the classifier. The pre-trained model is being used as a fixed feature extraction mechanism, which can be useful if the dataset is small, and/or pre-trained model solves a problem very similar to the one you want to solve.[6]

For our model, we will be using the third option. We will be making use of the pre-trained model to extract features which will be fed to the custom output model to classify the images.

Resnet Transfer Learning Model - For our solution, we will be making use of the pre-trained ResNet-50 model. This is a convolutional neural network that is trained on more than a million images from the ImageNet database. The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters. [5]

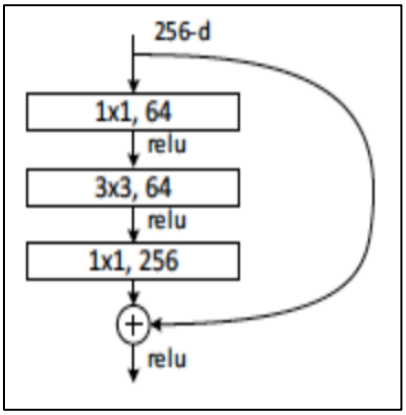


Fig 15: Bottleneck building block of Resnet50

ResNet works on the concept of skip connection. Usually a model is stacking convolution layers together one after the other. In ResNet, the model stacks convolution layers as before but now it also adds the original input to the output of the convolution block. This is called skip connection. This forms the building block of a ResNet which is called a residual block or identity block. A residual block is simply when the activation of a layer is fast-forwarded to a deeper layer in the neural network.

The benefits of ResNet model is that they mitigate the problem of vanishing gradient(caused when the gradients tends to get smaller and smaller as we keep on moving backward in the Network causing the earlier layers to learn much slowly) by allowing the alternate shortcut path for gradient to flow through. They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse.

Benchmark

For this problem of image classification, a simple **logistic regression model** (applied after few data-augmentation techniques) which uses pixel values as vectors gives a recall of **63.78%** and a precision of **72.18%.** This gives the f-beta score for the model as **1.29.**

However, pixel values are not the correct features since it loses a lot of spatial interaction between pixels which is very important. Hence as **a second baseline, a simple CNN model** will be created and measured for recall, precision and f-beta score. This score will serve as the second baseline against which the final solution model will be compared.

III. Methodology

Data Preprocessing

The Data pre-processing phase performed can be broken down into four parts.

1. **Data Loading**
    a. The images from different folders are converted and loaded into data frames.
    b. The images from NORMAL and PNEUMONIA folder have been associated with the corresponding labels/classes of 0 and 1.
    c. The dataset is randomly shuffled.

2. **Data Transformation**
    In this data pre-processing step, the input images are read, transformed and again stored in the file system. These transformed images become the new dataset for the models. Below are the transformations applied to the images as part of this transformation:
    a. **Image Scaling -** As seen in the data-exploration section, the images in the dataset are of different dimensions. A standard base image size should be used and fed to the models.. This is done via image scaling.
    b. **Image Cropping –** As observed during the data-exploration section, most of the images have corner regions(dark/black regions) which do not add any value to the features for the image. These side pixels of the images has been cropped from both x and y axis.
    c. **Data Normalization -** This is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. We divide the image by 255 so that the data is zero centred and has a range of [0-1]. This makes convergence faster while training the network.
    d. **Local Contrast Enhancement –** As seen during the visualization of histograms of sample images, there is a vast distribution of intensities/contrast in the images. Contrast enhancement and normalization will help achieve better result. This is done using an algorithm that uses histograms computed over different tile regions of the image. After applying this transformation, local details of image are enhanced even in regions that are darker or lighter than most of the image.

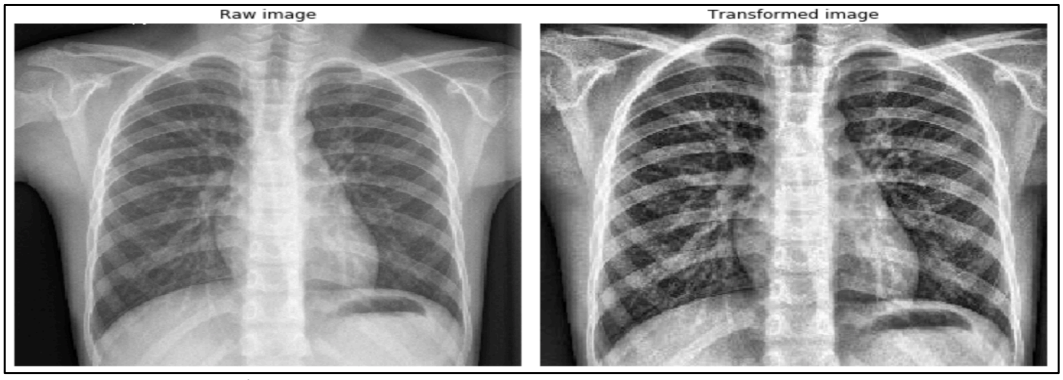


Fig 16: The raw lung image v/s the transformed image

3. **Image Augmentation**
    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.[8] To build a powerful image classifier using very little training data, image augmentation is usually required to boost the performance of deep networks.

- Image augmentation is applied to the training set for both the CNN model as well as the CNN with Transfer Learning model.
- Image augmentation techniques used include horizontal flips, tilts, increasing decreasing brightness, adding Gaussian Blur, shifting the image across both axis.
- For the simple CNN model, keras provided ImageDataGenerator is directly used and all parameters to be applied as augmentation techniques are sent to the class as parameters. In the Jupyter Notebook, a class Generators() has been created which has the methods to create ImageDataGenerator and send the different data generators to the CNN model to train , test and validate.
- For CNN with Transfer Learning model , special pre-trained model pre-processing is applied. (details in below point)..As a result the, keras provided ImageDataGenerator call cannot be used directly. A custom data generator class is created which uses other python libraries to perform augmentations. After augmentation the image is fed to the pre-processing method of the pre-trained model.

4. **Pre-processing for Transfer Learning Model (ResNet-50 Model)**
   Different pre-trained models pre-process the image differently. In order to achieve best results it is necessary that the data being used on the pre-trained model also goes through these pre-processing steps. For ResNet-50, these typically include scaling, normalization etc. These are applied to the data when using the CNN with Transfer Learning Model. In the Jupyter Notebook, the keras provided preprocess_input() method from resnet-50 model pre-processing package is called in the TranferModelGenerators() class to pre-process the data which will be used for the transfer learning model.

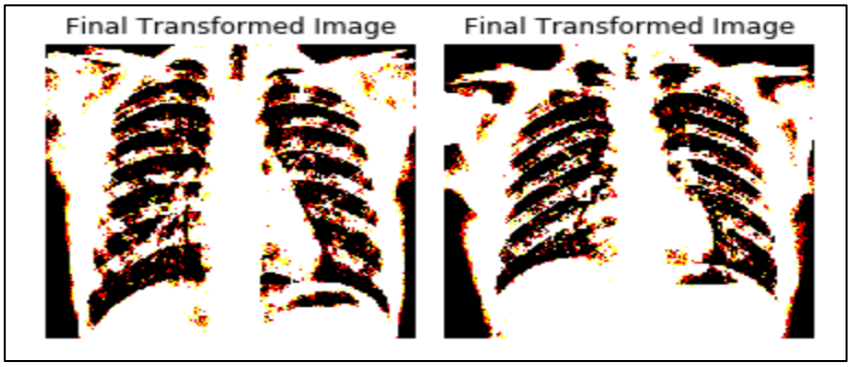


Fig 17: The final transformed image sent to the CNN with Transfer learning model.

Implementation

Below are the implementation details for both the normal CNN Model and the CNN Model with transfer learning. Both the models use the transformed image data. (details mentioned in Data Pre-processing section)

1. **Simple CNN Model Implementation**
   The goal for this implementation to build a standard CNN model which would perform the binary classification. This would serve as a baseline for the final solution. Steps performed as part of this implementation are as follows.
   a. Define the input shape for the model. For this model the image input will be of the form (256,256,3).

b. Define the CNN network architecture and the training parameters (no. of epochs and steps_per_epoch).
c. Define the model optimizer , loss function and compile the model.
d. Get the generators (train , validation and test generators) using the Generators() class (details in the Data Preprocessing section.)
e. Train the model by passing the train and validation generators and the training parameters. Also passed is the class_weight parameter with values which give more emphasis on the PNEUMONIA class.
f. Generate results/metrics using the Metrics() class.
g. Evaluate the metrics go back to step b. if the model is not satisfactory.

The final architecture details of the CNN model are as follows:

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
conv2d_1 (Conv2D)	(None, 256, 256, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 128, 128, 32)	0
batch_normalization_1 (Batch	(None, 128, 128, 32)	128
conv2d_2 (Conv2D)	(None, 128, 128, 16)	12816
batch_normalization_2 (Batch	(None, 128, 128, 16)	64
flatten_1 (Flatten)	(None, 262144)	0
dense_1 (Dense)	(None, 256)	67109120
batch_normalization_3 (Batch	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257

Total params: 67,124,305
Trainable params: 67,123,697
Non-trainable params: 608

- There are 2 convolution layers. The first one is with a 32-3X3 filter and the second one is with a 16-5X5 filter.
- A max pooling layer is applied after the first layer which is of size 2X2.
- There are 2 dense layers towards the end. One is a size of 256 and the other one is size 1. The last layer is applied a sigmoid activation function to be able to output probabilities.
- Batch normalization is applied after each of the convolution layers to help achieve higher learning rates and add regularization.
- The flatten layer is added which flattens the feature map.
- A Dropout layer is added before the final dense layer to avoid overfitting.

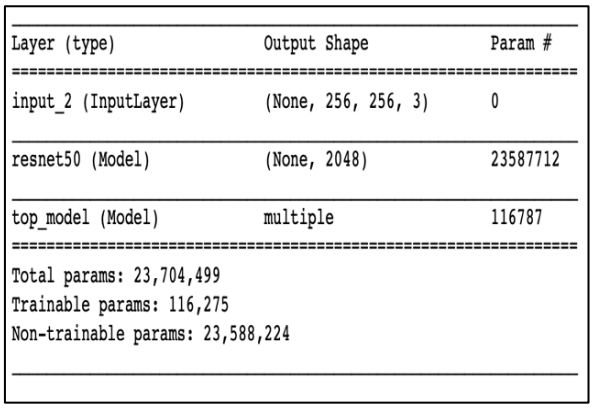
2. CNN Model with Transfer Learning Implementation

The goal for this implementation to build a accurate, robust model which would be the final solution for this binary classification problem. This would serve as a final model of solution. Steps performed as part of this implementation are as follows.

a. Define the input shape for the model. For this model, the image input will be of the form (256,256,3).
b. Get the custom train generator and data (validation and test generators) using the TranferModelGenerators () class (details in the Data Preprocessing section.)
c. Define the base model. The base model is the pre-trained ResNet-50 transfer model. We will only be training our custom top model, hence we set the train model feature for this model as false.
d. Define the custom CNN top-model.
e. Merge the two models to build the final CNN model with transfer learning.
f. Define the training parameters (no. of epochs and steps_per_epoch), the model optimizer , loss functions, callback functions (see Refinement section for callback functions used) and compile the model.

g. Train the model by passing the train generator, validation data and the training parameters. Also passed is the class_weight with values which help give more emphasis on the PNEUMONIA class during training/prediction.
h. Generate results/metrics using the Metrics() class.
i. Evaluate the metrics go back to step c. if the model is not satisfactory.

The final model consists of 3 part as shown below

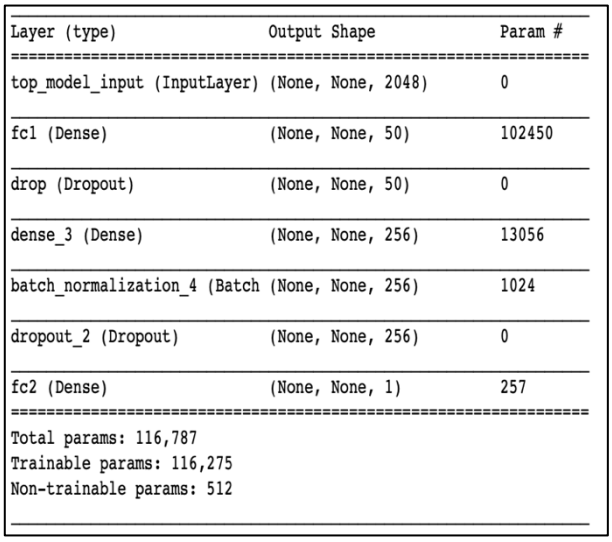


Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 256, 256, 3)	0
resnet50 (Model)	(None, 2048)	23587712
top_model (Model)	multiple	116787

Total params: 23,704,499
Trainable params: 116,275
Non-trainable params: 23,588,224

- The Input Layer – This gives inputs in the form of images whose size is 256,256,3.
- The resnet50 Model – This is the pretrained model which will act as the feature extraction part of the model. The training for this layer is frozen.
- The Top Model – This is the custom model which will be used to output the correct classification using resent50 model as the base.

The details of the top-model for the model are as follows.



Layer (type)	Output Shape	Param #
top_model_input (InputLayer)	(None, None, 2048)	0
fc1 (Dense)	(None, None, 50)	102450
drop (Dropout)	(None, None, 50)	0
dense_3 (Dense)	(None, None, 256)	13056
batch_normalization_4 (Batch	(None, None, 256)	1024
dropout_2 (Dropout)	(None, None, 256)	0
fc2 (Dense)	(None, None, 1)	257

Total params: 116,787
Trainable params: 116,275
Non-trainable params: 512

- The input to the top model is the output from the feature extraction layer of the resent model.
- There are 2 dense layers of size 50 and 256 units.
- There is a batch normalization layer before the last layer which adds regularization.
- The last dense layer is applied a sigmoid activation function to be able to output probabilities. The dense layer is 1 unit as we have to predict only 2 categories.
- Dropout layers are added in the middle layer to avoid overfitting.

Refinement

As the first step a simple CNN model was created. This served as the baseline model for the transfer Learning Model. An initial CNN Model with Transfer Learning was created which achieved a recall of around 80%. Below are the steps performed as part of the refinement process. The goal was to achieve a higher recall while having a respectable score of precision.

1. **Changing no. of Layers:** As part of refinement for the CNN Transfer model, different number and different types of layers were added to the custom model. When more layers were added (>3), the model started overfitting which could be seen by the training/validation loss graphs. Finally after many combinations, the target top-model was finalized.

2. **Reduce Learning Rate on Plateau** – Whenever the learning stagnated (loss function stopped decreasing), the learning rate was reduced. This was done using keras callback functions provided to the model fit function.

3. **Changing Optimizers** – Initially an RMSprop optimizer was used with the model. However this was later changed to Adam Optimizer. Adam Optimizer can be seen as a variant on the combination of RMSProp and momentum, the update looks like RMSProp except that a smooth version of the gradient is used instead of the raw stochastic gradient. Also the full Adam update also includes a bias correction mechanism[8]. Using Adam optimizer, provider better result and hence was chosen as the final optimizer.

4. **Combinations of Epochs** – Early stopping(where model stops training when the improvement is not satisfied) callback function was used to detect the optimum number of epochs to be used for training. At the same time, the model was trained with different combinations of epochs and steps per epoch. Combination giving best results was selected.

5. **Changing the batch sizes** – Smaller batch sizes are noisy, offering a regularizing effect and lower generalization error and hence used more often but they require more CPU/memory. Different batch sizes we tried and finally the default batch size (32) was chosen.

IV. Results

Model Evaluation and Validation

The final model is a CNN Model using Transfer Learning. Below are the reasons I think this is a robust model and can handle changes in training data.

1. The final solution uses CNN Model - A CNN works by extracting features from images. This eliminates the need for manual feature extraction. CNN deep learning models extremely accurate for computer vision tasks.

2. The final solution uses pre-trained Model - ResNet models are one of the most widely used models for transfer learning. Using such a robust model as the base of our model for feature extraction definitely helps make the final solution more robust and hence the final results from the model can be trusted.

3. The final solution is solution built using image augmentation. There are many image augmentation techniques used before feeding the data to the model. The features of these different kinds of images (not actually part of dataset) are also considered while building the model. Hence the model will be able to work reasonably well in-case of small perturbations in input data.

Justification

Model Metrics Table:

Model Name	Recall	Precision	F-Beta score
Logistic Regression Model	63%	72%	1.29
Simple CNN Model	78%	79%	1.56
CNN with Transfer Learning Model	96%	84%	1.87

Above is a table of the different models and their results. As it can be clearly seen the final solution combining CNN with Transfer Learning provides the best solution.

It has the best recall rate, at the same time the precision is also the highest. Also it has the highest F-beta score of all the models. Below figure shows the loss and accuracy plots between the training and validation data. The loss plot clearly indicates that the model has converged and there is no-overfitting.

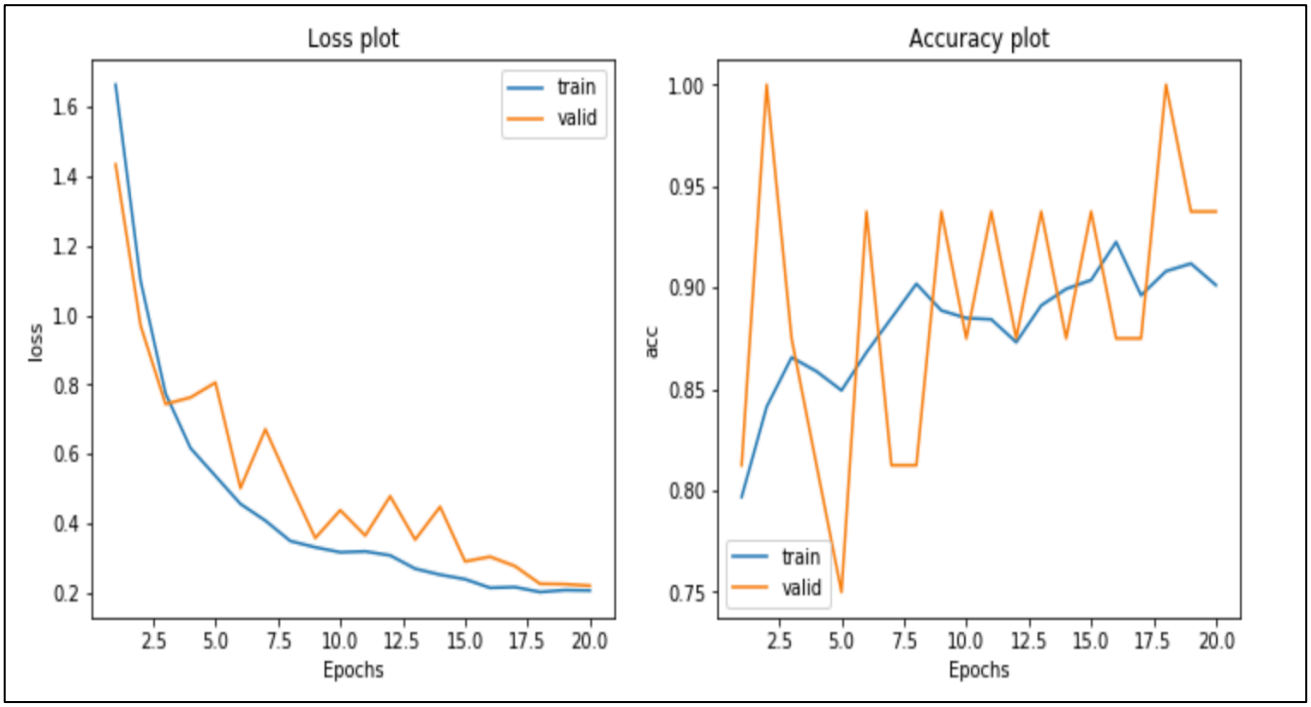
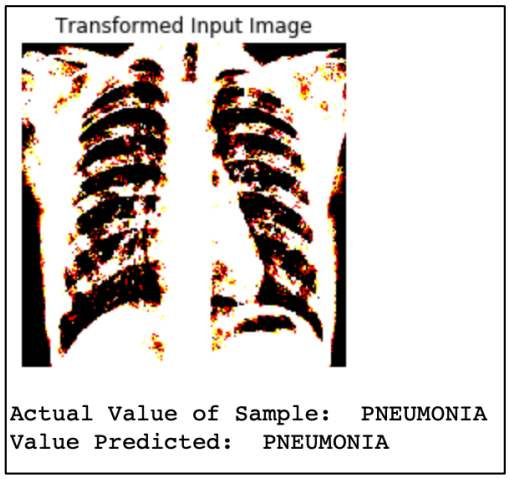


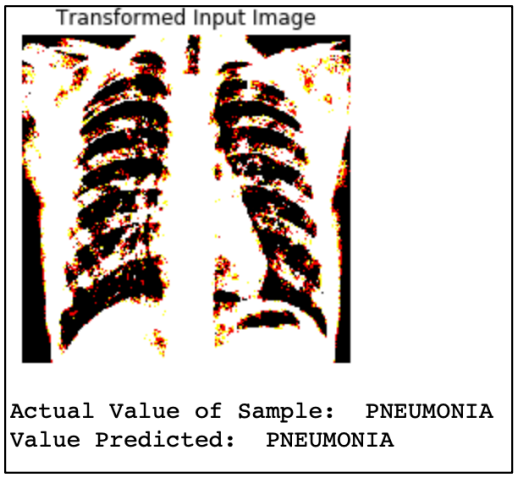
Fig 18: The loss and accuracy plots for the CNN with Transfer learning model.

This solution model is significant to solve the problem for the given limited dataset or dataset of the same domain and can aid in the process of diagnosis. To apply it in a generalized environment, many improvements will be needed, some of which are discussed in the last section.

V. Conclusion

Free-Form Visualization





- The two examples above show the transformed image and its predictions. The values are correctly predicted by the model.
- The image above looks different from the original image because the image is the transformed image. We can see that the transformation has significant effect on the image and it converts the image in a form suitable for the CNN to extract features and classify correctly.

Reflection

The process used for this project can be summarized as below:

1. Identification of relevant, interesting domain and a specific problem (with publicly available dataset) where machine learning can be applied.
2. Downloading the dataset , exploring the dataset and applying some exploratory analysis on the images.
3. Using image transformation techniques to transform the given images to a format better suited for CNN and then applying data augmentation techniques.
4. Creating a simple CNN model which will serve as one of the baselines..
5. Creating a CNN model with transfer learning by using ResNet-50 pre-trained model.
6. Performing multiple rounds of refinement to the model to get desired results.
7. Evaluating the model using the different matrices defined.
8. Using the final model to predict against the unseen-test data.

I found step 3, the transformation and data augmentation step the hardest. In order to transform the image in a way useful for the model, image properties and their effects needed to be understood. Understanding these image properties like contract, hue , brightness channel-wise distribution etc. and techniques to improving image quality was a challenge. Also data augmentation for the simple CNN model was easy however, for pre-trained model, image pre-processing specific for the ResNet model had to be applied and hence custom data generators had to be created. Trying to figure out that not applying the

model pre-processing was causing the model to give sub-standard result was difficult. Also building the custom data generator was a time-consuming process.

The step I found most interesting was the model refinement phase. There are a number of properties which can be tuned and even slight changes in them can affect the training of the model significantly. Trying to tune the model, gave me chance to understand the impact of these properties which in turn gave me better insight and understanding on the inner workings of CNNs.

The solution fits my expectation for the problem and dataset but before using it in a general environment, there are different improvements which can be targeted. (Details in section below)

Improvement

The following points can be considered as improvements

1. Dataset Improvement
    a. Currently a single data source is used as the input dataset. Using multiple datasets from different sources will help make the solution more robust and more accurate.
    b. Medical diagnosis is often a combination of understanding symptoms and checking images like x-ray. Having a dataset of symptoms combined with the provided images can also make the model more robust and provide better results.
2. For fine tuning models, I chose among the most commonly used optimizers and loss functions. I would want to use different combinations of loss functions with optimizers and check if it improves performance.
3. ResNet-50 was chosen as the model of choice for the pre-trained model. The solution could be tried with other well-known pretrained models like Xception, VGG-19, DenseNet etc. Trying the solution with such pretrained models can help identify and use the best pre-trained model for our solution.
4. In our transfer learning model, the feature-extraction base was frozen. We can try to train some of those layers and freeze others and see the results.
5. Although the precision of the model is good, it can be improved further by trying techniques like L1-L2 regularizations.

References

1. https://www.thoracic.org/patients/patient-resources/resources/top-pneumonia-facts.pdf
2. https://becominghuman.ai/what-exactly-does-cnn-see-4d436d8e6e52
3. https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks
4. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
5. https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17
6. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751
7. https://towardsdatascience.com/hitchhikers-guide-to-residual-networks-resnet-in-keras-385ec01ec8ff
8. https://towardsdatascience.com/image-augmentation-for-deep-learning-histogram-equalization-a71387f609b2
9. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
10. https://arxiv.org/abs/1512.03385