Submitted in partial fulfilment of the requirements for qualifying

ADVANCED DIPLOMA IN BIG DATA ANALYTICS

2020

Project Report

DIAGNOSING PNEUMONIA CASES BY IMAGE BASED DEEP-LEARNING

Principal Investigator

Shri. Prasoon Kumar K G (Principal Technical Officer)

Project Mentors

Mrs. Vimala Mathew
Mrs. Reshma C B
Mr. Hari K
Mr. Tom Toby

Project Team Members

Roshna Raj T M
Vidya P
Juhi Dakhane
Rahul Raut
Akash Ghanghav
Anitya Umare
Sanket Umap

Acknowledgement

We, the students of Advanced Diploma in Big Data Analytics, are overwhelmed in all humbleness and gratefulness to acknowledge our deep gratitude to all those who have helped us put our ideas to perfection. We, whole heartedly thank **Dr. M P Pillai, Executive Director (Scientist/Engineer 'G'),** NIELIT, Calicut, for his patronage and wonderful support given to us in exposing us to the world of Big-Data Analytics through a unique program of P G Diploma in Big Data Analytics at National Institute of Electronics and Information Technology, Calicut.

It is our privilege to place on record our gratitude to **Mr. Prasoon Kumar K G (Senior Technical Officer)**, **Mrs. Vimala Mathew (Scientist/Engineer 'D'), Mrs. Reshma C B (Technical Officer)** for their immense knowledge support, guidance on our every stage of learning and while working on this project. They were and are always there to show us the right track when needed help. During the process, we got a chance to see the stronger side of our technical and non-technical aspects and strengthen our concepts. Here by, we gladly consider ourselves to be the most fortunate batch of students.

We are grateful to **Mr. Nandakumar R (Scientist/Engineer 'D')** for his relentless work in organizing Seminars and Webinars related to Data Science, Machine Learning, and AI during our course. We thank him and **Mr. Justin Joseph (Placement Coordinator)** for their help in Placements in the campus.

We also owe our gratitude and thanks to **Mr. Hari K (Senor Technical Officer)** for technical support in installation of certain software and **Mr. Tom Toby (Data Analyst)** for his constant technical support in our project and in completion of our assignments throughout our course.

Last but not the least, we whole heartedly thank all the teaching and non-teaching staff of our institute and our friends for their help in successful completion of this project work.

Abstract

According to the World Health Organization (WHO), pneumonia kills about 2 million children under 5 years old every year and is consistently estimated as the single leading cause of childhood mortality, killing more children than HIV/AIDS, malaria, and measles combined. The risk of pneumonia is especially large in developing nations where billions face energy poverty and rely on polluting forms of energy. In such regions, the problem can be further worsened due to limited medical resources and personnel. For example, in Africa's 57 nations, a gap of 2.3 million doctors and nurses exists. For these populations, accurate and fast diagnosis means everything. It can guarantee timely access to treatment and save much needed time and money for those already experiencing poverty. The dataset we collected from Kaggle consists of very few samples and that too unbalanced. The aim and purpose of this study is to develop a robust deep learning model from scratch on this limited amount of data for the diagnosis of pediatric pneumonia using chest X-ray images.

One key element of diagnosis is radiographic data. Since chest X-rays are routinely obtained as standard of care and can help differentiate between different types of pneumonia. However, rapid radiologic interpretation of images is not always available, particularly in low-resource settings where childhood pneumonia has the highest incidence and the highest rates of mortality. So, this AI model could help and assist doctors in identifying pneumonia in children and facilitate rapid referrals for children needing urgent intervention.

CONTENTS

Introduction

PNEUMONIA

Pneumonia is an infection in the lungs. It can be mild or serious. Pneumonia is generally more common in children younger than 5 years old. The WHO reports that nearly all cases (95%) of new-onset childhood clinical pneumonia occur in developing countries, particularly in Southeast Asia and Africa. Bacterial and viral pathogens are the two leading causes of pneumonia but require very different forms of management. Bacterial pneumonia requires urgent referral for immediate antibiotic treatment, while viral pneumonia is treated with supportive care. Therefore, accurate and timely diagnosis is imperative.

CAUSES OF PNEUMONIA

Common bacteria and viruses that may cause pneumonia are:

- Streptococcus pneumoniae
- Mycoplasma pneumonia. This often causes a mild form of the illness called walking pneumonia.
- Influenza virus

Pneumonia may sometimes be caused by fungi.

SYMPTOMS OF PNEUMONIA

Symptoms may be a bit different for each child. They may also depend on what is causing the pneumonia. Cases of bacterial pneumonia tend to happen suddenly with these symptoms:

- Cough that produces mucus
- Cough pain
- Vomiting or diarrhea
- Loss of appetite
- Fever and tiredness (fatigue)

Early symptoms of viral pneumonia are the same as those of bacterial pneumonia. But with viral pneumonia, the breathing problems happen slowly. The child may wheeze and the cough may get worse. Viral pneumonia may make a child more at risk for bacterial pneumonia.

In addition to the symptoms listed above, your child may have:

- Chills
- Fast or hard breathing
- Headache

DIAGNOSIS

One key element of diagnosis is radiographic data, since chest X-rays are routinely obtained as standard of care and can help differentiate between different types of pneumonia (Figure 1). However, rapid radiologic interpretation of images is not always available, particularly in the low-resource settings where childhood pneumonia has the highest incidence and highest rates of mortality.

Figure 1. Illustrative examples of Chest X-Rays in patients with Pneumonia

The other diagnosis steps include:

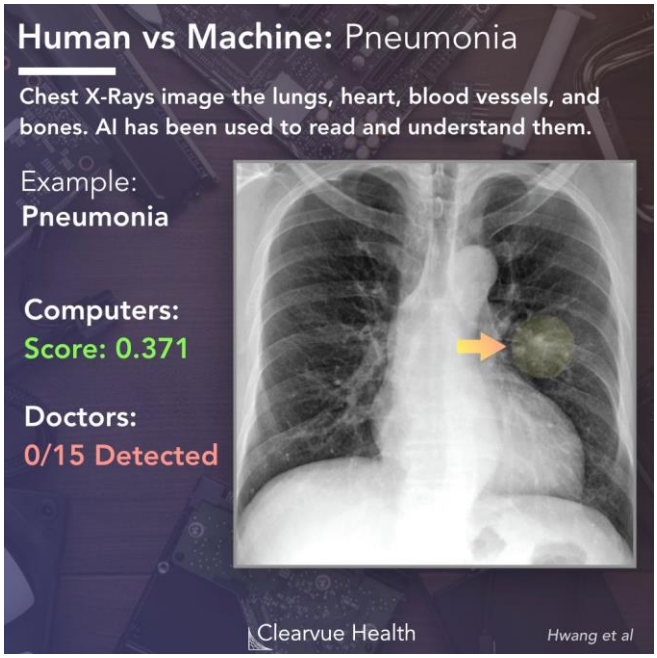
- Blood tests
- Sputum Culture
- Pulse oximetry
- Chest CT Scan
- Bronchoscopy
- Pleural fluid culture

ROLE OF ARTIFICIAL INTELLIGENCE IN MEDICAL DIAGNOSIS

Artificial Intelligence (AI) has the potential to revolutionize disease diagnosis by performing classification which is difficult for human experts and by rapidly reviewing immense amounts of images. According to National Institutes of Health (NIH), chest x-ray is the best test for pneumonia diagnosis. However, reading x-ray images can be tricky and requires domain expertise and experience. It would be nice if we can just ask a computer to read the images and tell us the results.

In a recent paper, researchers tested an AI against 15 doctors of different levels of training to see how well the AI is at reading chest x-rays. Could it beat an average doctor? A Radiologist? A Specialist Radiologist?

Consider the following example:



The AI and doctors were asked to diagnose a patient based on the above chest X-ray. The X-ray contained what doctors call a “ground glass opacity”. In this case the opacity was from a case of pneumonia. In this example, the AI was able to correctly locate the lesion with a probability score of 0.371, while all the 15/15 doctors missed it.

Despite its potential, clinical interpretability and feasible preparation of AI remains challenging.

Objective

Our objective is to build an algorithm to automatically identify whether a patient is suffering from pneumonia or not by looking at chest X-ray images. The dataset we collected from Kaggle consists of very few samples and that too unbalanced. The aim and purpose of this study is to develop a robust deep learning model from scratch on this limited amount of data for the diagnosis of pediatric pneumonia using chest X-ray images. The algorithm had to be extremely accurate because lives of people is at stake.

SCOPE OF THE STUDY

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

Installation

This section covers all the description and installation of all the tools required in this project. All the software and packages used are Open Source software and are freely available.

SYSTEM REQUIREMENTS

Table 1 describes the hardware requirements for this project. We have developed the whole system on the following hardware and ran our model on Kaggle platform using Kaggle's python notebook.

Type of Hardware	Hardware Requirements
Hardware	• Intel Core i5 8th Gen • Kaggle's NVIDIA Tesla P100 GPU • 64 – bit system
Disk Space	5 GB free disk space (minimum)
Memory	4 GB or more (recommended)

Table 1. Hardware Requirements

Table 2 describes the basic software requirements we have used.

Type of Software	Software Requirements
Operating System	Linux (Ubuntu 16.04 LTS) or greater
Web Browser	Google Chrome (79.0.3945.88)
Text Editor	Vim, gedit
Optional Software	Jupyter/Kaggle Notebook, Google Colab

Table 2. Software Requirements

TOOLS USED

Big Data Technologies Applied

Other Technologies Applied

Project Design

FLOW CHART

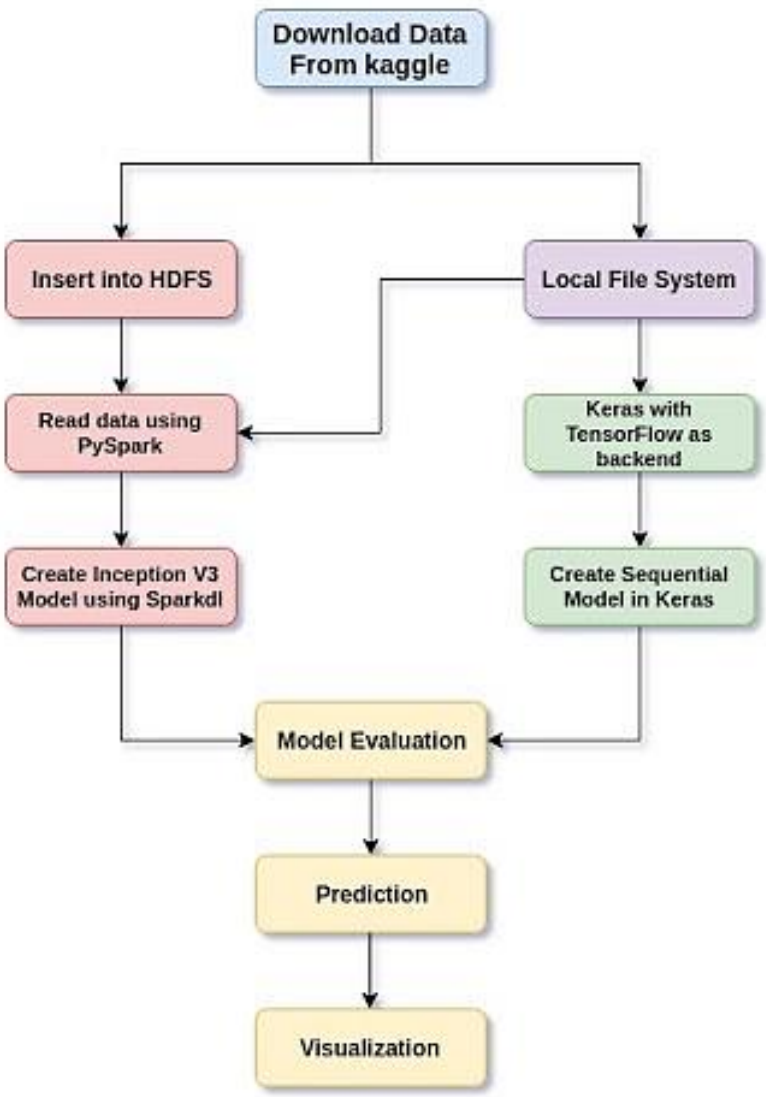


Figure 4. Flow Chart of our study

Data Analysis

DATA

To train and test the model, the 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 dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,856 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

SAMPLE SELECTION

There are a total of 5,862 labeled chest X-ray images from children, including 3,876 characterized as depicting pneumonia and 1,342 normal to train the AI system. The model was then tested with 234 normal images and 390 pneumonia images.

IMPLEMENTATION

There are two ways of analyzing this data and making the appropriate Deep Learning model if you observed our Flow Chart.

Method 1: Using Keras

Method 1 consists of implementing Convolutional Neural Networks (CNN) using Keras with TensorFlow as backend. This method loads data from the local filesystem using Keras functions. This model was run on Kaggle's Nvidia Tesla P100 GPU.

Deep Learning

Deep learning, with artificial neural networks at its core, is a new and powerful tool that can be used to derive value from big data. Most of the data today is unstructured, and deep learning algorithms are very effective at learning from, and generating predictions for, wildly unstructured data.

Since ours is a 1.2 GB data, we can classify it as a Big Data. This dataset is ideal for applying deep learning techniques. We've chosen CNN Algorithm since the dataset contains images.

CNN

In our study, we're making use of CNN to read the image data. CNN Layers are multiple processing layers to which image analysis filters, or convolutions, are applied. The abstracted representation of images within each layer is constructed by systematically convolving multiple filters across the image, producing a feature map that is used as input to the following layer. This architecture makes it possible to process images in the form of pixels as input and to give the desired classification as output. The image-to-classification approach in one classifier replaces the multiple steps of previous image analysis methods.

CNNs work by reducing an image to its key features and using the combined probabilities of the identified features appearing together to determine a classification. The reason why we chose CNN over other classification algorithms is that they require fewer hyperparameters and less supervision.

Code

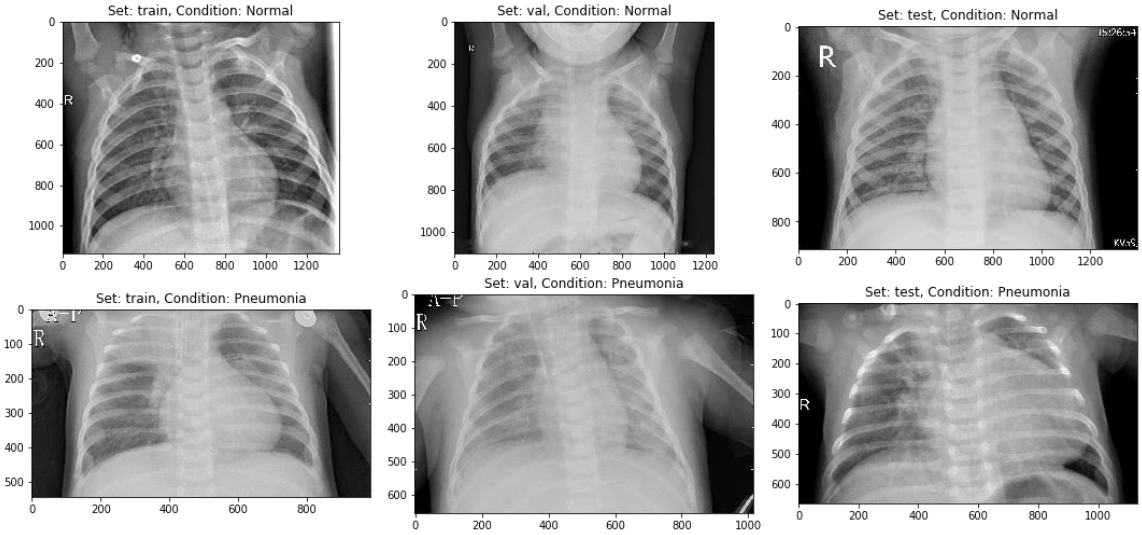
Let's start with importing necessary libraries and setting seed = 232 for reproducibility.

```
#Importing Required Libraries
import os
import numpy as np
import pandas as pd
import random
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

#Importing Keras For Model Creation
import keras.backend as K
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, SeparableConv2D, MaxPool2D
LeakyReLU, Activation
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
import tensorflow as tf

#Setting Seed Value For Random Number Generation
seed = 232
np.random.seed(seed)
tf.random.set_seed(seed)
```

```
Using TensorFlow backend.
```

Now, let's have a look at normal and pneumonia images and see if we can make out any differences between them.

```
#Path Of Kaggle Dataset
input_path = '../input/chest-xray-pneumonia//chest_xray/chest_xray/'

#For Plotting X-Ray Images
fig, ax = plt.subplots(2, 3, figsize=(15, 7))
ax = ax.ravel()
plt.tight_layout()

for i, _set in enumerate(['train', 'val', 'test']):
    set_path = input_path+_set
    ax[i].imshow(plt.imread(set_path+'/NORMAL/'+os.listdir(set_path+'/NORMAL')[0]),
                 cmap='gray')
    ax[i].set_title('Set: {}, Condition: Normal'.format(_set))
    ax[i+3].imshow(plt.imread(set_path+'/PNEUMONIA/'+os.listdir(set_path+'/PNEUMONIA')[0]),
                   cmap='gray')
    ax[i+3].set_title('Set: {}, Condition: Pneumonia'.format(_set))
```



Well, they don't seem so different!

Now, let's start processing the data. We divide the dataset into three different sets – train, test, and validation sets.

```
#Getting Number Of Images Present In Folder
for _set in ['train', 'val', 'test']:
    n_normal = len(os.listdir(input_path + _set + '/NORMAL'))
    n_infect = len(os.listdir(input_path + _set + '/PNEUMONIA'))
    print('Set: {}, normal images: {}, pneumonia images: {}'.format(_set, n_normal,
                                                                    n_infect))
```

```
Set: train, normal images: 1342, pneumonia images: 3876
Set: val, normal images: 9, pneumonia images: 9
Set: test, normal images: 234, pneumonia images: 390
```

The next few steps include:

- Data Augmentation
- Feed training and testing images to the network
- Create labels for the images

The practice of data augmentation is an effective way to increase the size of the training set. Augmenting the training examples allow the network to "see" more diversified data points during training.

Then we defined a couple of data generators: one for training data, and the other for validation data. A data generator is capable of loading the required amount of data (a mini batch of images) directly from the source folder, convert them into *training data* (fed to the model) and *training targets* (a vector of attributes — the supervision signal).

```
#Creating Function For Data reading & Processing

def process_data(img_dims, batch_size):
    # Data generation objects
    train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, vertical_flip=True)
    test_val_datagen = ImageDataGenerator(rescale=1./255)

    # This is fed to the network in the specified batch sizes and image dimensions
    train_gen = train_datagen.flow_from_directory(
    directory=input_path+'train',
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)

    test_gen = test_val_datagen.flow_from_directory(
    directory=input_path+'test',
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)



# Creating Test Data & Label Arrays Usefulfor Creating confusion matrix
test_data = []
test_labels = []

for cond in ['/NORMAL/', '/PNEUMONIA/']:
    for img in (os.listdir(input_path + 'test' + cond)):
        img = plt.imread(input_path+'test'+cond+img)
        img = cv2.resize(img, (img_dims, img_dims))
        img = np.dstack([img, img, img])
        img = img.astype('float32') / 255
        if cond=='/NORMAL/':
            label = 0
        elif cond=='/PNEUMONIA/':
            label = 1
        test_data.append(img)
        test_labels.append(label)

test_data = np.array(test_data)
test_labels = np.array(test_labels)

return train_gen, test_gen, test_data, test_labels
```

Now that we're done with defining our functions, let's create some constants.

```
#Setting Hyperparameters
img_dims = 150
epochs = 10
batch_size = 32

#calling Function process_data()
train_gen, test_gen, test_data, test_labels = process_data(img_dims, batch_size)
```

```
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

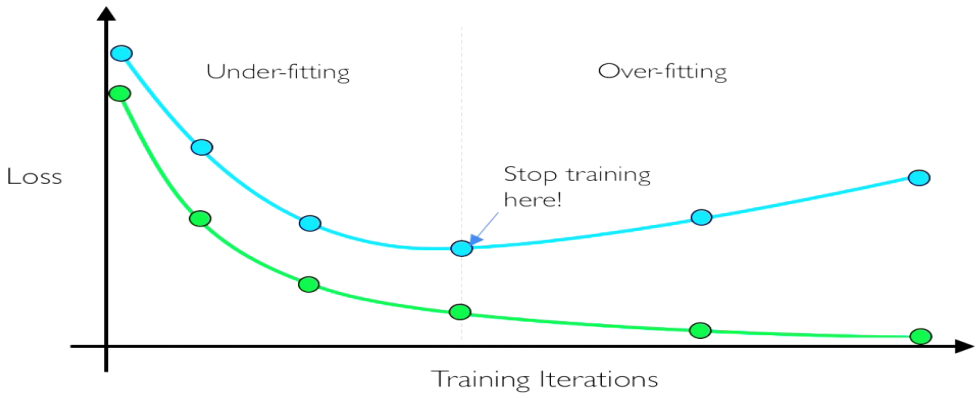
It's time to build our model. This can be done with following steps:

1. We used five convolutional blocks comprised of convolutional layer, max-pooling and batch-normalization.
2. On top of it we used a flatten layer and followed it by four fully connected layers.
3. Also in between we have used dropouts to reduce over-fitting.
4. Activation function was ReLu throughout except for the last layer where it was Sigmoid as this is a binary classification problem.
5. We have used Adam as the optimizer and cross-entropy as the loss.

Before training the model, it is useful to define one or more callbacks. Pretty handy ones are: ModelCheckpoint and EarlyStopping.

- **ModelCheckpoint**: When training requires a lot of time to achieve a good result, often many iterations are required. In this case, it is better to save a copy of the best performing model only when an epoch that improves the metrics ends.
- **EarlyStopping**: Sometimes, during training we can notice that the generalization gap (i.e. the difference between training and validation error) starts to increase, instead of decreasing. This is a symptom of overfitting that can be solved in many ways (*reducing model capacity*, *increasing training data*, *data augmentation*, *regularization*, *dropout*, etc.) Often a practical and efficient solution is to stop training when the generalization gap is getting worse. This can be done with EarlyStopping.

The below picture is to visualize when to stop the over-fitting of the model.




```
#Setting Input Shape of Training Data for CNN
inputs = Input(shape=(img_dims, img_dims, 3))

# First convolution block
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Second convolution block
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Third convolution block
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)


# Fourth convolution block
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)

# Fifth convolution block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)

# Full Convolution layer
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(rate=0.7)(x)
x = Dense(units=128, activation='relu')(x)
x = Dropout(rate=0.5)(x)
x = Dense(units=64, activation='relu')(x)
x = Dropout(rate=0.3)(x)
```

```
# Output layer
output = Dense(units=1, activation='sigmoid')(x)

# Creating model and compiling
model = Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint(filepath='best_weights.hdf5', save_best_only=True,
                             save_weights_only=True)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=2,
                              verbose=2, mode='max')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=1, mode='min')
```

In our experiments, we usually choose a batch-size of 32 and at least 10 epochs for best results. However, a higher batch size and epochs of your choice can be experimented to see the difference in results.

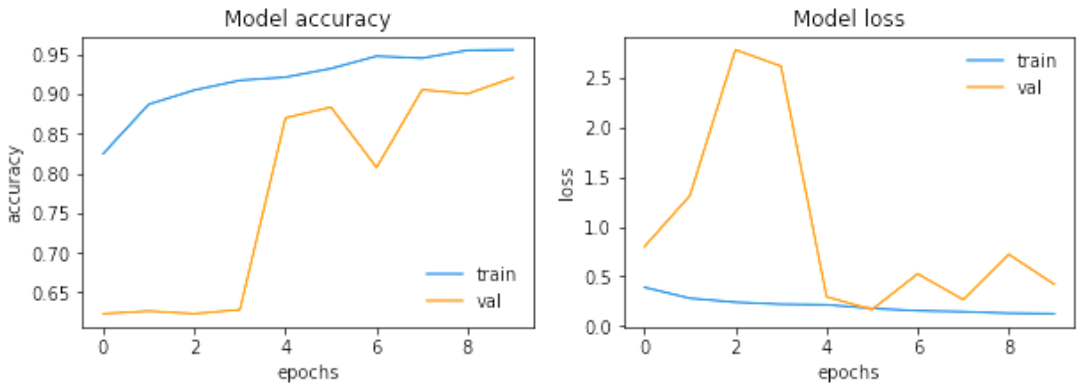
```
#Training Model and Validating
hist = model.fit_generator(
           train_gen, steps_per_epoch=train_gen.samples // batch_size,
           epochs=epochs, validation_data=test_gen,
           validation_steps=test_gen.samples // batch_size, callbacks=[checkpoint,
                                                                       lr_reduce])
```

```
Epoch 1/10
163/163 [==============================] - 109s 672ms/step - loss: 0.3907 - accuracy: 0.825
0 - val_loss: 0.7992 - val_accuracy: 0.6234
Epoch 2/10
163/163 [==============================] - 99s 605ms/step - loss: 0.2798 - accuracy: 0.8871
- val_loss: 1.3151 - val_accuracy: 0.6267
Epoch 3/10
163/163 [==============================] - 95s 584ms/step - loss: 0.2403 - accuracy: 0.9049
- val_loss: 2.7842 - val_accuracy: 0.6233
Epoch 4/10
163/163 [==============================] - 94s 579ms/step - loss: 0.2201 - accuracy: 0.9172
- val_loss: 2.6191 - val_accuracy: 0.6284
Epoch 5/10
163/163 [==============================] - 93s 570ms/step - loss: 0.2144 - accuracy: 0.9212
- val_loss: 0.2947 - val_accuracy: 0.8699

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
Epoch 6/10
163/163 [==============================] - 93s 571ms/step - loss: 0.1771 - accuracy: 0.9321
- val_loss: 0.1667 - val_accuracy: 0.8834
Epoch 7/10
163/163 [==============================] - 93s 571ms/step - loss: 0.1546 - accuracy: 0.9477
- val_loss: 0.5263 - val_accuracy: 0.8074

Epoch 00007: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
Epoch 8/10
163/163 [==============================] - 93s 571ms/step - loss: 0.1448 - accuracy: 0.9452
- val_loss: 0.2665 - val_accuracy: 0.9054
Epoch 9/10
163/163 [==============================] - 92s 566ms/step - loss: 0.1294 - accuracy: 0.9551
- val_loss: 0.7233 - val_accuracy: 0.9003

Epoch 00009: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
Epoch 10/10
163/163 [==============================] - 93s 569ms/step - loss: 0.1244 - accuracy: 0.9557
- val_loss: 0.4203 - val_accuracy: 0.9206
```

Visualization

It's time to visualize the loss and accuracy our model has got.

```
#Plotting Model Accuracy and Loss
fig, ax = plt.subplots(1, 2, figsize=(10, 3))
ax = ax.ravel()

for i, met in enumerate(['accuracy', 'loss']):
    ax[i].plot(hist.history[met])
    ax[i].plot(hist.history['val_' + met])
    ax[i].set_title('Model {}'.format(met))
    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(met)
    ax[i].legend(['train', 'val'])
```



If you carefully observe, you can see that the model is converging and there is a decrease in train loss and validation loss with epochs. Also, the model is able to reach up to 90% accuracy with just 10 epochs.

Using ScikitLearn libraries, let's plot the Confusion Matrix, Accuracy, Precision, Recall Score, F1 – Score.

```
#Importing sklearn for getting Classification Report

from sklearn.metrics import accuracy_score, confusion_matrix

#Predicting Classes of Test Data using Our Created Model
preds = model.predict(test_data)

acc = accuracy_score(test_labels, np.round(preds))*100
cm = confusion_matrix(test_labels, np.round(preds))
tn, fp, fn, tp = cm.ravel()

print('CONFUSION MATRIX ------------------')
print(cm)

print('\nTEST METRICS ----------------------')
precision = tp/(tp+fp)*100
recall = tp/(tp+fn)*100
print('Accuracy: {}%'.format(acc))
print('Precision: {}%'.format(precision))
print('Recall: {}%'.format(recall))
print('F1-score: {}'.format(2*precision*recall/(precision+recall)))

print('\nTRAIN METRIC ----------------------')
print('Train acc: {}'.format(np.round((hist.history['accuracy'][-1])*100, 2)))
```

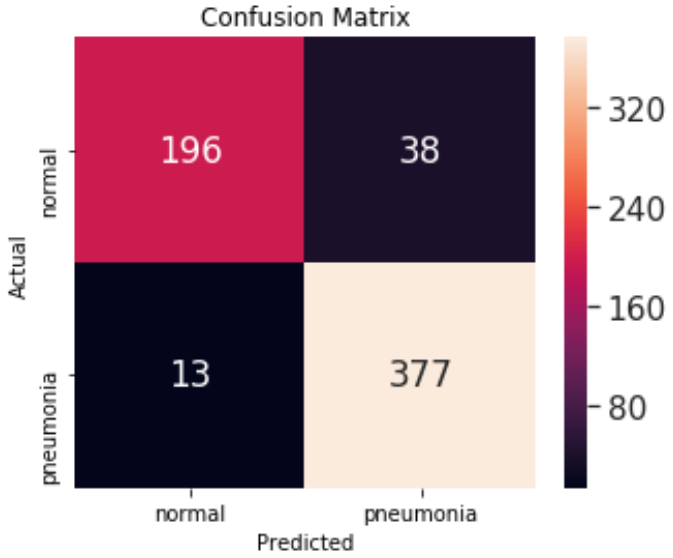
```
CONFUSION MATRIX ------------------
[[196  38]
 [ 13 377]]

TEST METRICS ----------------------
Accuracy: 91.82692307692307%
Precision: 90.8433734939759%
Recall: 96.66666666666667%
F1-score: 93.66459627329193

TRAIN METRIC ----------------------
Train acc: 95.57
```

Let's visualize the Confusion Matrix much better by importing seaborn library. Along with it, let's draw ROC Curve.

```
#Importing Seaborn and Pandas for Plotting Confusion Matrix
import seaborn as sn
import pandas as pd

#Plotting Confusion Matrix
df_cm = pd.DataFrame(cm, index = ('normal', 'pneumonia'), columns = ('normal', 'pneumonia'))
plt.figure(figsize = (5,4))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True, fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()


#Getting True and Predicted Labels
y_true = test_labels
y_pred = np.round(preds)

#Importing sklearn for Getting ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import sklearn.metrics as metrics

fpr, tpr, threshold = metrics.roc_curve(y_true, y_pred)
roc_auc=roc_auc_score(y_true, y_pred)

#Plotting ROC Curve
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.001, 1])
plt.ylim([0, 1.001])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

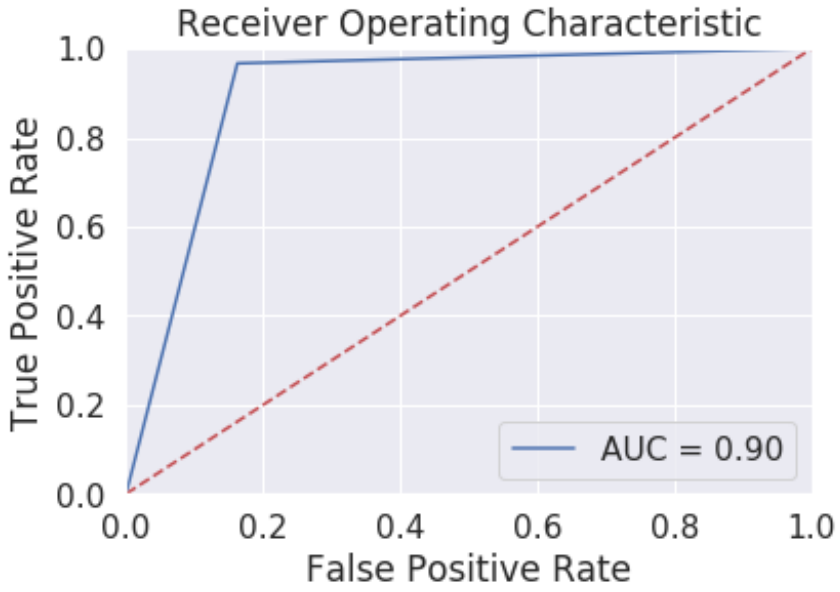
Confusion Matrix:

Confusion matrix is very much necessary for the above model because we are having unequal number of people with pneumonia and no pneumonia. In the above dataset we are having more people suffering from pneumonia than normal people.



ROC Curve:

In Medical Diagnosis, the curve is used to select optimal cut-off values for a test result, to assess the diagnostic accuracy of a test, and to compare the usefulness of different tests.



Method 2: Transfer Learning with Keras

Method 2 consists of implementing Transfer Learning using Keras with TensorFlow as backend. This method loads data from the local filesystem using Keras functions. This model was run on Kaggle's Nvidia Tesla P100 GPU.

What is Transfer Learning?

Transfer learning is an approach used in machine learning where a model that was created and trained for one task, is reused as the starting point for a secondary task. Transfer learning differs from traditional machine learning because it involves using a pre-trained model as a springboard to start a secondary task.

This approach mimics the way humans apply knowledge learned for one task to a new task. For example, John Doe learned how to read in first grade. In tenth grade, John used his reading abilities in chemistry class. The knowledge he had gained from the primary task (learning how to read) became the foundation on which he started the secondary task (learning chemistry).

InceptionV3

In this method, we will make use of Inception v3 which is a widely used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years.

The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax.

Code

The steps for pre-processing data is same as Method 1. All we need to do is, instead of writing the code to construct our Convolutional blocks, we need to import InceptionV3 from Keras and create the base for pre-trained model by downloading it's weights. Here, it's 'imagenet'.

Let's look at some code!

```
#Importing InceptionV3 from Keras but with imagenet weights.
#Also defining the necessary input shape of the resized images which were resized initially.
#The default image size is 299 X 299 for InceptionV3.

inputs = Input(shape=(img_dims, img_dims, 3))

from keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(weights='imagenet',include_top=False,input_shape=(img_dims, img_dims, 3))
x = base_model.output

#The output of the above model was being fed into the after
#layers that we are laying down. There is no rule of thumb for laying down the after layers
#after InceptionV3 but practice and intuition.
#The layers were checked multiple times for the output and finally the second last layer of BatchNormalization()
#did some good for the model.

x = Dropout(0.5)(x)
from keras.layers import GlobalAveragePooling2D
x = GlobalAveragePooling2D()(x)
x = Dense(128,activation='relu')(x)
x = BatchNormalization()(x)
output = Dense(1,activation = 'sigmoid')(x)

# Creating model and compiling
#Loading the best set of weights on the model which were attained by checkpointing
#and saving the weights in between the epochsand also prior to overfitting of the model.

model = Model(inputs=base_model.input, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint(filepath='best_weights.hdf5', save_best_only=True, save_weights_only=True)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=2, verbose=2, mode='max')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=1, mode='min')
```

We're also using Dropout ratios to reduce overfitting for our model. We've used checkpoint, lr_reduce, and early_stop to check for the best fits for our model and save them accordingly.

```
=================================================================================================
Total params: 22,065,697
Trainable params: 22,031,009
Non-trainable params: 34,688
_________________________________________________________________________________________________
```

The above picture gives us information regarding the model's summary after compiling the model.

It's time to fit our model.

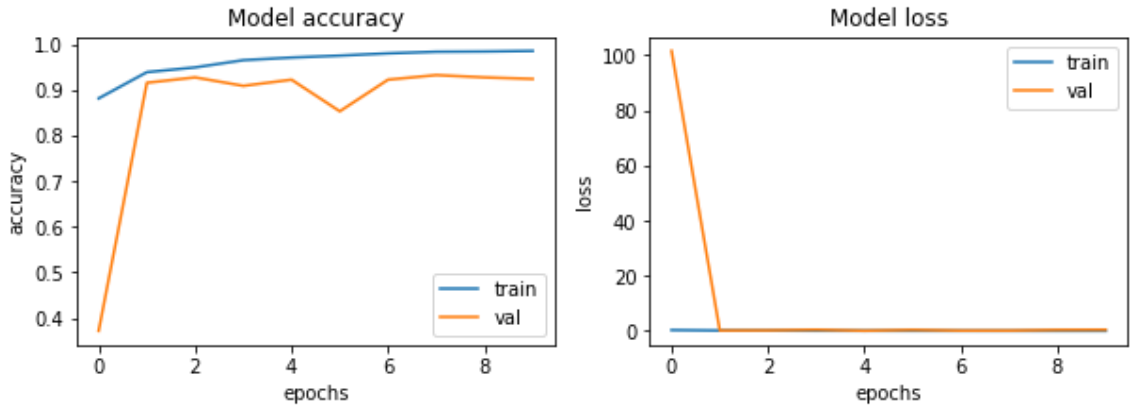
```
hist = model.fit_generator(
          train_gen, steps_per_epoch=train_gen.samples // batch_size,
          epochs=epochs, validation_data=test_gen,
          validation_steps=test_gen.samples // batch_size, callbacks=[checkpoint, lr_reduce])
```

```
Epoch 1/10
163/163 [==============================] - 140s 862ms/step - loss: 0.2863 - accuracy: 0.9089 - val_loss: 0.5051 -
val_accuracy: 0.8240
Epoch 2/10
163/163 [==============================] - 93s 568ms/step - loss: 0.1943 - accuracy: 0.9306 - val_loss: 14.3677 -
val_accuracy: 0.6250
Epoch 3/10
163/163 [==============================] - 92s 562ms/step - loss: 0.1624 - accuracy: 0.9417 - val_loss: 1.3599 - v
al_accuracy: 0.6926
Epoch 4/10
163/163 [==============================] - 93s 568ms/step - loss: 0.1202 - accuracy: 0.9557 - val_loss: 0.3897 - v
al_accuracy: 0.8699

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
Epoch 5/10
163/163 [==============================] - 92s 562ms/step - loss: 0.0838 - accuracy: 0.9711 - val_loss: 0.1818 - v
al_accuracy: 0.9105
Epoch 6/10
163/163 [==============================] - 90s 554ms/step - loss: 0.0733 - accuracy: 0.9735 - val_loss: 0.4462 - v
al_accuracy: 0.7399

Epoch 00006: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
Epoch 7/10
163/163 [==============================] - 93s 568ms/step - loss: 0.0572 - accuracy: 0.9803 - val_loss: 0.2785 - v
al_accuracy: 0.9189
Epoch 8/10
163/163 [==============================] - 92s 564ms/step - loss: 0.0468 - accuracy: 0.9829 - val_loss: 0.0567 - v
al_accuracy: 0.9527


Epoch 00008: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
Epoch 9/10
163/163 [==============================] - 91s 558ms/step - loss: 0.0501 - accuracy: 0.9826 - val_loss: 0.3555 - v
al_accuracy: 0.9358
Epoch 10/10
163/163 [==============================] - 92s 563ms/step - loss: 0.0406 - accuracy: 0.9872 - val_loss: 0.1494 - v
al_accuracy: 0.9476

Epoch 00010: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
```

As you can see, the model is stopping at every point when it is trying to overfit. The reason for overfitting is that our model is too complex and we are short on data.

Visualization

Let's visualize what our model has got to say about accuracy and model loss.



These plots have improved a lot compared to our first method. The model has reached to almost zero loss with just 1 epoch and has achieved 90+% validation accuracy by 2nd epoch.

```
TEST METRICS ----------------------
Accuracy: 94.87179487179486%
Precision: 93.65853658536587%
Recall: 98.46153846153847%
F1-score: 96.00000000000001

TRAIN METRIC ----------------------
Train acc: 98.72
```

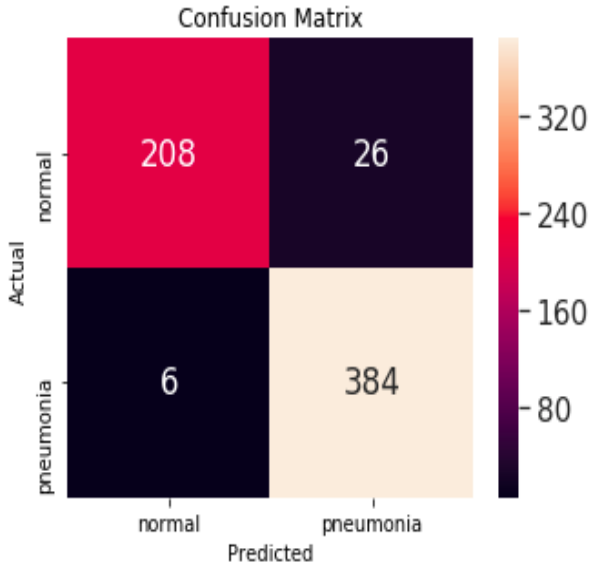
This model has greater precision than Model 1. Earlier the precision was 90.84 % but now it is having 93.65 % as its precision. Therefore, we have minimized false positive in the denominator of the formula. This makes sure that a person not suffering from pneumonia shouldn't be diagnosed as being suffering from pneumonia. This is what precision is.

Confusion matrix is very much necessary for the above model because we are having unequal number of people with pneumonia and no pneumonia. In the above dataset we are having more people suffering from pneumonia than normal people.

In datasets where there is an imbalance between classes, accuracy alone isn't importance. We have to consider things like recall score, precision, and F1-Score.

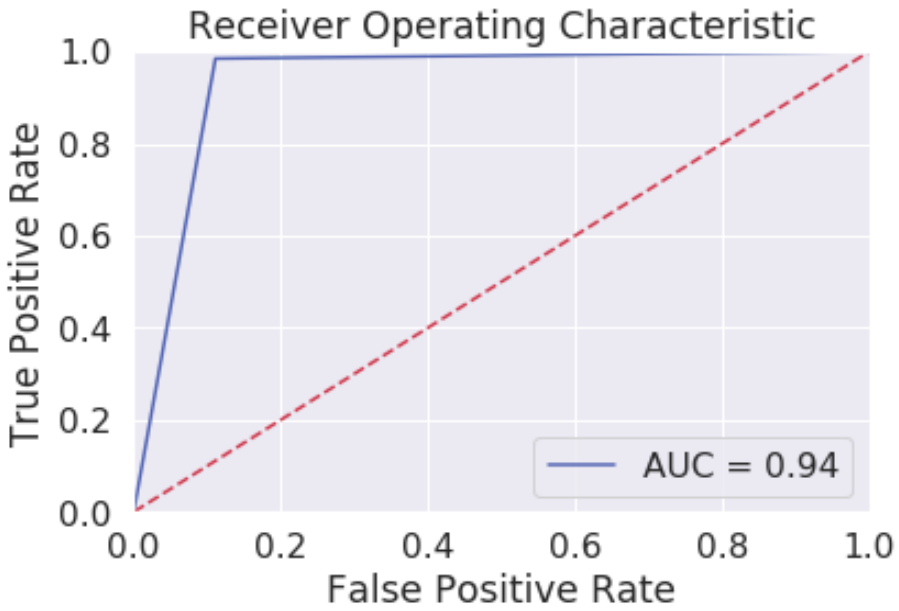
Confusion Matrix:

Notice how the confusion matrix changed from Model 1 and became even better than before.



ROC Curve:

In Medical Diagnosis, the curve is used to select optimal cut-off values for a test result, to assess the diagnostic accuracy of a test, and to compare the usefulness of different tests.



Results

The result section contains summary of Project Details, Design, and Implementation. It also contains highlight a summary of the results of our project with respect to technologies used, architecture implemented for the model. The results have been elaborated below:

Technologies used:

The project helped us in getting acquainted with both Big Data and other technologies most significantly *TensorFlow, Keras, Hadoop, Spark* amongst many others.

Architecture:

In Artificial Neural Networks, the architecture of the model affects the final results. In this project, we have used a set of CNN Layers to build the first model which gave an accuracy of 90% and then when we chose the architecture of InceptionV3 via Transfer Learning, the accuracy of the model has increased to 94%.

The Recall Score in method 1 came out to be 96% and in method 2 it increased by 2% which gave us the final recall score as 98%. Recall Score is very important, especially in medical diagnosis since we do not want the model to give us a False Positive, which means, predicting the person is not having a pneumonia when in fact he/she is suffering from pneumonia.

Data Visualization:

The very first step to any kind of Data Analysis is to visualize data. In this project, we've have visualized the data and results with state-of-the-art libraries such as SciKit Learn and Seaborn.

Discussion

In this section, we will be discussing about other possibilities of training and testing our model for better accuracy and decreasing the need to code for the CNN blocks by using Hadoop and Spark.

Hadoop

Apache Hadoop is an open-source software framework for storage and large-scale processing of datasets on clusters of commodity hardware. Since we're using 1.2 GB of data, it is wise to choose Hadoop technology. The Apache Hadoop framework is composed of the following modules:

- Hadoop Common: Contains libraries and utilities needed by other Hadoop modules
- Hadoop Distributed File System (HDFS): A distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
- Hadoop YARN: A resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.
- Hadoop MapReduce: A programming model for large scale data processing.

We used:

- Version: Hadoop 2.8.5

Start HDFS

- Write these scripts on the command line to start HDFS are:

start-dfs.sh

start-yarn.sh

- Test Hadoop Daemons by writing jps on the command line.
- Output should be:

Namenode

Datanode

Secondary Namenode

ResourceManager

NodeManager

- Make a directory to store our data in HDFS

hadoop fs -mkdir /dnn

- Put the data from local file system to Hadoop File System

```
hadoop fs -put /home/hduser/Documents/kaggle/chest-xray-pneumonia/chest_xray/chest_xray /dnn/
```

- Get configuration key

```
hdfs getconf -confKey fs.defaultFS
```

How Hadoop works?

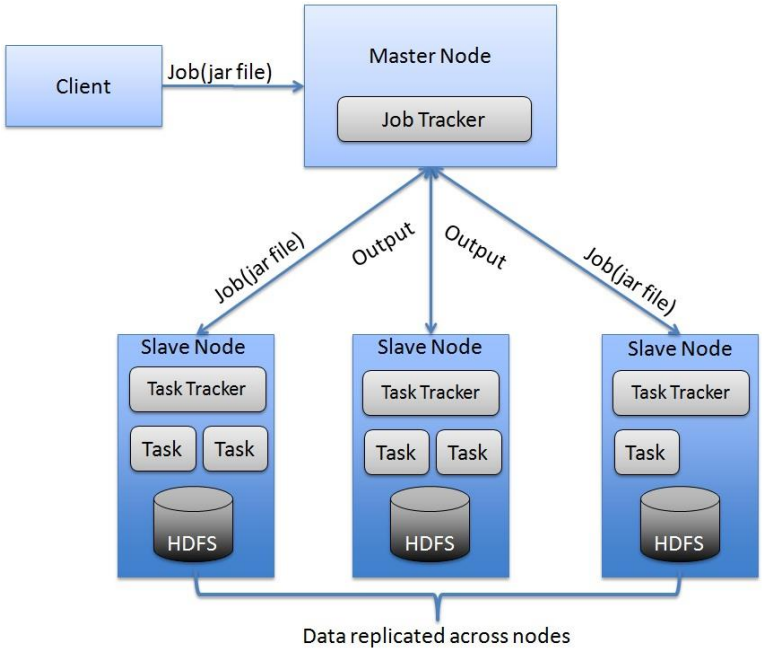


Figure 3. Multinode Hadoop Architecture

Hadoop and its various components fit together to ensure a fault-tolerant, durable and highly efficient model for storage and management of Big Data

- **Namenode**

Namenode is the node which stores the filesystem metadata i.e. which file maps to what block locations and which blocks are stored on which datanode. The namenode maintains two in-memory tables, one which maps the blocks to datanodes (one block maps to 3 datanodes for a replication value of 3) and a datanode to block number mapping. Whenever a datanode reports a disk corruption of a particular block, the first table gets updated and whenever a datanode is detected to be dead (because of a node/network failure) both the tables get updated.

- **Secondary Namenode**

The secondary namenode regularly connects to the primary namenode and keeps snapshotting the filesystem metadata into local/remote storage. It does so at a poor frequency and should not be heavily relied on.

- **Datanode**

The data node is where the actual data resides. Points to Note:

1. All datanodes send a heartbeat message to the namenode every 3 seconds to say that they are alive.
2. If the namenode does not receive a heartbeat from a particular datanode for 10 minutes, then it considers that datanode to be dead/out of service and initiates replication of blocks which were hosted on that datanode to be hosted on another datanode.
3. The datanodes can talk to each other to rebalance data, move and copy data around and keep the replication high.
4. When the datanode stores a block of information, it maintains a checksum for it as well.
5. The datanodes update the namenode with the block information periodically and before updating verify the checksums.
6. If the checksum is incorrect for a particular block i.e. there is a disk level corruption for that block, it skips that block while reporting the block information to the namenode.
7. In this way, namenode is aware of the disk level corruption on that datanode and takes steps accordingly.

- **Node Manager**

This is a yarn daemon which runs on individual nodes and receive updated information on resource containers from their individual datanodes via background daemons. Different resources such as memory, CPU time, network bandwidth etc. are put into one unit called the Resource Container. The Node Manager in turn ensures fault tolerance on the datanodes for any mapreduce jobs.

- **Resource Manager**

This is a yarn daemon which manages the allocation of resources to the different jobs apart from comprising a scheduler which just takes care of the scheduling jobs without worrying about any monitoring or status updates.

Spark

Apache Spark is an open-source data analytics cluster computing framework.

Pre-requisites: Hadoop 2.8.5 +

- Spark fits into the Hadoop open-source community, building on top of the Hadoop Distributed File System (HDFS).
- Spark is not tied to the two-stage MapReduce paradigm, and promises performance up to 100 times faster than Hadoop MapReduce, for certain applications.
- Follows the concept of a Resilient Distributed Dataset (RDD), which allows to transparently store data on memory and persist it to disc if it's needed.
- Provides Machine Learning Library MLLib for Data Analytics on the fly.

We used:

- Version: Spark 2.4.4

Let's start Spark Session by importing it from pyspark.sql

```
#Starting a spark session
from pyspark.sql import SparkSession

#Creating a spark context
spark = SparkSession.builder.appName("DL with Spark Deep Cognition").getOrCreate()
sc = spark.sparkContext

sc
```

SparkContext

Spark UI

Version
v2.4.4
Master
local[*]
AppName
PySparkShell

Let's check if we have successfully loaded data into our HDFS.

```
#listing out the data loaded into hdfs
URI = sc._gateway.jvm.java.net.URI
Path = sc._gateway.jvm.org.apache.hadoop.fs.Path
FileSystem = sc._gateway.jvm.org.apache.hadoop.fs.FileSystem
fs = FileSystem.get(URI("hdfs://localhost:9000"), sc._jsc.hadoopConfiguration())

for f in fs.listStatus(Path('/dnn/chest_xray')):
    print(f.getPath())
```

```
hdfs://localhost:9000/dnn/chest_xray/.DS_Store
hdfs://localhost:9000/dnn/chest_xray/test
hdfs://localhost:9000/dnn/chest_xray/train
hdfs://localhost:9000/dnn/chest_xray/val
```

Now, let's read in the NORMAL and PNEUMONIA images from HDFS.

```
#importing pyspark libraries required for reading images
from pyspark.ml.image import ImageSchema
from pyspark.sql.functions import lit

#reading normal x ray images from hdfs
nor_df = ImageSchema.readImages("hdfs://localhost:9000/dnn/chest_xray/train/NORMAL").withColumn("label", lit(1))

nor_df.select("image.origin", "image.width", "image.height").show(truncate=False)
```

```
+-----------------------------------------------------------------------+-----+------+
|origin                                                                 |width|height|
+-----------------------------------------------------------------------+-----+------+
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0115-0001.jpeg|2090 |1858  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0117-0001.jpeg|1422 |1152  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0119-0001.jpeg|1810 |1434  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0122-0001.jpeg|1618 |1279  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0125-0001.jpeg|1600 |1125  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0127-0001.jpeg|1974 |1306  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0128-0001.jpeg|1528 |1013  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0129-0001.jpeg|1384 |1167  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0131-0001.jpeg|1450 |1144  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0133-0001.jpeg|1468 |993   |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0135-0001.jpeg|1724 |1581  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0137-0001.jpeg|1346 |1044  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0140-0001.jpeg|1156 |1237  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0141-0001.jpeg|1740 |1453  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0143-0001.jpeg|2138 |1928  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0145-0001.jpeg|1596 |1156  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0147-0001.jpeg|2102 |1974  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0149-0001.jpeg|1542 |1152  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0151-0001.jpeg|1518 |1156  |
|hdfs://localhost:9000/dnn/chest_xray/train/NORMAL/IM-0152-0001.jpeg|1612 |1133  |
+-----------------------------------------------------------------------+-----+------+
only showing top 20 rows
```

```
#reading pneumonic x ray images from hdfs
pnu_df = ImageSchema.readImages("hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA").withColumn("label", lit(0))

pnu_df.select("image.origin", "image.width", "image.height").show(truncate=False)
```

```
+-----------------------------------------------------------------------------------------+-----+------+
|origin                                                                                   |width|height|
+-----------------------------------------------------------------------------------------+-----+------+
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1000_bacteria_2931.jpeg|1152 |760   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1000_virus_1681.jpeg   |1072 |768   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1001_bacteria_2932.jpeg|1244 |863   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1002_bacteria_2933.jpeg|1242 |940   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1003_bacteria_2934.jpeg|1488 |1280  |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1003_virus_1685.jpeg   |1008 |616   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1004_bacteria_2935.jpeg|1184 |816   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1004_virus_1686.jpeg   |856  |480   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1005_bacteria_2936.jpeg|1048 |888   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1005_virus_1688.jpeg   |1216 |792   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1006_bacteria_2937.jpeg|1553 |1044  |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1007_bacteria_2938.jpeg|992  |776   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1007_virus_1690.jpeg   |1048 |664   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1008_bacteria_2939.jpeg|1240 |792   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1008_virus_1691.jpeg   |1576 |1328  |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1009_virus_1694.jpeg   |1520 |936   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person100_virus_184.jpeg     |1104 |792   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1010_bacteria_2941.jpeg|1328 |1368  |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1010_virus_1695.jpeg   |502  |307   |
|hdfs://localhost:9000/dnn/chest_xray/train/PNEUMONIA/person1011_bacteria_2942.jpeg|1128 |520   |
+-----------------------------------------------------------------------------------------+-----+------+
only showing top 20 rows
```

Let's join both normal and pneumonia X-Ray images into training set and read the test data.

```
#combining both normal and pneumonic x ray images into train_df
train_df = nor_df.unionAll(pnu_df)
# If data is fully loaded in memory, training may be expensive.
# This ensure that each of the paritions has a small size.
train_df = train_df.repartition(100)


#reading test data from hdfs
nor_test_df = ImageSchema.readImages("hdfs://localhost:9000/dnn/chest_xray/test/NORMAL").withColumn("label",lit(1))
pnu_test_df=ImageSchema.readImages("hdfs://localhost:9000/dnn/chest_xray/test/PNEUMONIA").withColumn("label",lit(0))
test_df = nor_test_df.unionAll(pnu_test_df)
test_df = test_df.repartition(100)
```

We import required spark libraries to fit the model with our data. Deep Learning Pipelines provides utilities to perform transfer learning on images, which is one of the fastest ways to start using deep learning. Using Deep Learning Pipelines, it can be done in just few lines of code.

```
#importing pyspark libraries required for fitting the model
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline


#importing sparkdl deep image featurizer
from sparkdl import DeepImageFeaturizer

Using TensorFlow backend.
```

Deep Learning Pipelines enables fast transfer learning with the concept of a Featurizer. The following example combines the InceptionV3 model and logistic regression in Spark to adapt InceptionV3 to our specific domain. The DeepImageFeaturizer automatically peels off the last layer of a pre-trained neural network and uses the output from all the previous layers as features for the logistic regression algorithm. Since logistic regression is a simple and fast algorithm, this transfer learning training can converge quickly using far fewer images than are typically required to train a deep learning model from ground-up.

```
#defining InceptionV3 model
featurizer = DeepImageFeaturizer(inputCol="image", outputCol="features", modelName="InceptionV3")

#defining final output classifier
lr = LogisticRegression(maxIter=10, regParam=0.05, elasticNetParam=0.3, labelCol="label")

#compiling the model
p = Pipeline(stages=[featurizer, lr])

#fitting the model
p_model = p.fit(train_df)

#importing pyspark libraries for model evaluation
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

tested_df = p_model.transform(test_df)
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(tested_df.select("prediction", "label"))))
```

When the scalability increases, you can use the above code to train the model. The main difference between Hadoop and Spark is that Spark processes the huge amount of Data in RAM unlike how MapReduce processes it in Disk. That is the reason why Spark is 100 times faster than MapReduce.

Conclusion and Future Work

An estimated nearly 1·2 million children younger than 5 years died in 2011 from pneumonia. Most of these deaths occurred in developing countries where access to care is limited and interventions that have improved care in developed countries are scarce. In this study, we have developed a model which can assist doctors in fast diagnosing which means everything for the children who suffer from pneumonia in developing countries.

Future Work

The next step for our model is to deploy into the cloud like AWS, Google Cloud, Azure, and Heroku as a web application where doctors can access it readily. We can also train and deploy our model for real-time detections offline in Google Coral Dev Board and NVIDIA Jetson Nano.

References

[1] “Data” https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

[2] “Python Documentation” https://docs.python.org/3.8/

[3] “Keras Documentation” https://keras.io/layers/convolutional/

[4] “TensorFlow Documentation” https://www.tensorflow.org/api_docs/python/tf

[5] “InceptionV3” https://arxiv.org/abs/1512.00567

[6] “Spark Documentation” https://spark.apache.org/docs/latest/rdd-programming-guide.html

[7] “Hadoop Documentation” https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html

[8] “Transfer Learning” https://missinglink.ai/guides/neural-network-concepts/transfer-learning-overview/

[9] “Solving errors” https://stackoverflow.com/

[10] “Conceptual doubts” https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/