

# **Chest X-Ray Medical Diagnosis with Deep Learning**

A SMALL PROJECT REPORT SUBMITTED  
IN PARTIAL – FULFILMENT OF THE REQUIREMENT  
FOR THE AWARD OF THE DEGREE  
OF

## **MASTER OF COMPUTER APPLICATION**

BY

**ASHUTOSH KUMAR SAH**

**(MCA/40043/18)**



**DEPARTMENT OF COMP. SC. & ENGG.  
BIRLA INSTITUTE OF TECHNOLOGY  
LALPUR, RANCHI-834001**

**(MO 2020)**

## DECLARATION CERTIFICATE

This is to certify that the work presented in the project entitled “***Chest X-Ray Medical Diagnosis with Deep Learning***” in partial fulfillment of the requirement for the award of the degree of **Master of Computer Application**, of Birla Institute of Technology, Lalpur, Ranchi, is an authentic work carried out under my supervision. To the best of my knowledge, the content of this project does not form a basis for the award of any previous Degree to anyone else.

**Date :** 27-12-2020

**Dr. JAYA PAL**

Dept. to Comp. Sc. & Engg.  
BIT Extn. Centre, Lalpur

**Dr. AMRITA PRIYAM**

Academic Coordinator  
Department of Comp. Sc. & Engg.  
Birla Institute of Technology, Lalpur  
Ranchi – 834001

**Dr. A.N JHA**

(In-charge)  
Birla Institute of Technology, Lalpur  
Ranchi – 834001

## **CERTIFICATE OF APPROVAL**

The foregoing project entitled, “**Chest X-Ray Medical Diagnosis with Deep Learning**” is hereby approved as a creditable study of the research topic and has been presented in a satisfactory manner to warrant its acceptance as a prerequisite to the degree for which it was submitted.

It is understood that by this approval, the undersigned does not necessarily endorse any conclusion drawn or opinion expressed therein, but approve the thesis for the purpose for which it is submitted.

**(Internal Examiner)**

**(External Examiner)**

**Dr. A.N JHA  
(In-charge)  
BITEC, Lalpur, Ranchi**

## ACKNOWLEDGEMENT

I would like to convey my heartfelt thanks to **DR. JAYA PAL** of **Birla Institute of Technology, Extension Center, Lalpur, Ranchi** who always gave me valuable suggestions and guidance for the completion of the Project. She helped me to understand and remember important details of the project. Our project has been a success only because of her guidance.

I extend my sincere thanks to **Dr. A. N. Jha**, In-charge of **BITEC, Lalpur**, for providing sufficient infrastructure and a good environment in the College to complete our course.

I extend our sincere thanks to **C K Birla** Chairman of our college, for providing the opportunity in College to complete our course.

With extreme jubilation and deepest gratitude, I would like to thank Head of the Computer Science. Department **Dr. Amrita Priyam** for her constant encouragement.

I special thanks to our Assistant Professor **Dr. P. S. Bishnu** for his support and valuable suggestions regarding project work.

I am especially intended and also beholden to my friend and finally, I am thankful to the members of my family for their support and encouragement.

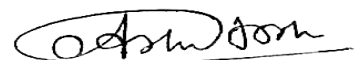
**Roll No:**

MCA/40043/18

**Name:**

Ashutosh Kumar Sah

**Signature:**



## **TABLE OF CONTENTS**

<b>Serial no.</b>	<b>Title</b>	<b>Page no.</b>
1	Abstract	6
2	Work Flow of the Project	7
3	Technologies Used	8
3.1	Deep learning	8
3.2	DenseNet	9
3.3	Dense Net 121	12
3.4	Keras	12
3.5	ROC Curve And AUROC	13
3.6	GradCAM	14
3.7	NumPy	15
3.8	Seaborn	16
3.9	Matplotlib	16
3.10	Pyplot	17
3.11	Image Data Generator	17
3.12	Pandas	17
3.13	GlobalAveragePooling2D	19
4	Diseases that can be detected	20
5	NIH data	22
6	Code	23
7	Pictures of implementaion	31
9	Conclusion	37
9	Future Scope	37
10	Bibliography	38

## 1. Abstract

# Chest X-Ray Medical Diagnosis with Deep Learning

**Gist About the Project:** This project “Chest X-Ray Medical Diagnosis with Deep Learning” used AI, is transforming the practice of medicine. It’s helping doctors diagnose patients more accurately, make predictions about patients’ future health, and recommend better treatments.

In this project I have created convolutional neural network image classification and segmentation models to make diagnoses of lung and the diagnoses our model can detect are:

- |                 |                        |
|-----------------|------------------------|
| 1. Cardiomegaly | 9. Pneumothorax        |
| 2. Emphysema    | 10. Pleural Thickening |
| 3. Effusion     | 11. Pneumonia          |
| 4. Hernia       | 12. Fibrosis           |
| 5. Infiltration | 13. Edema              |
| 6. Mass         | 14. Consolidation      |
| 7. Nodule       |                        |
| 8. Atelectasis  |                        |

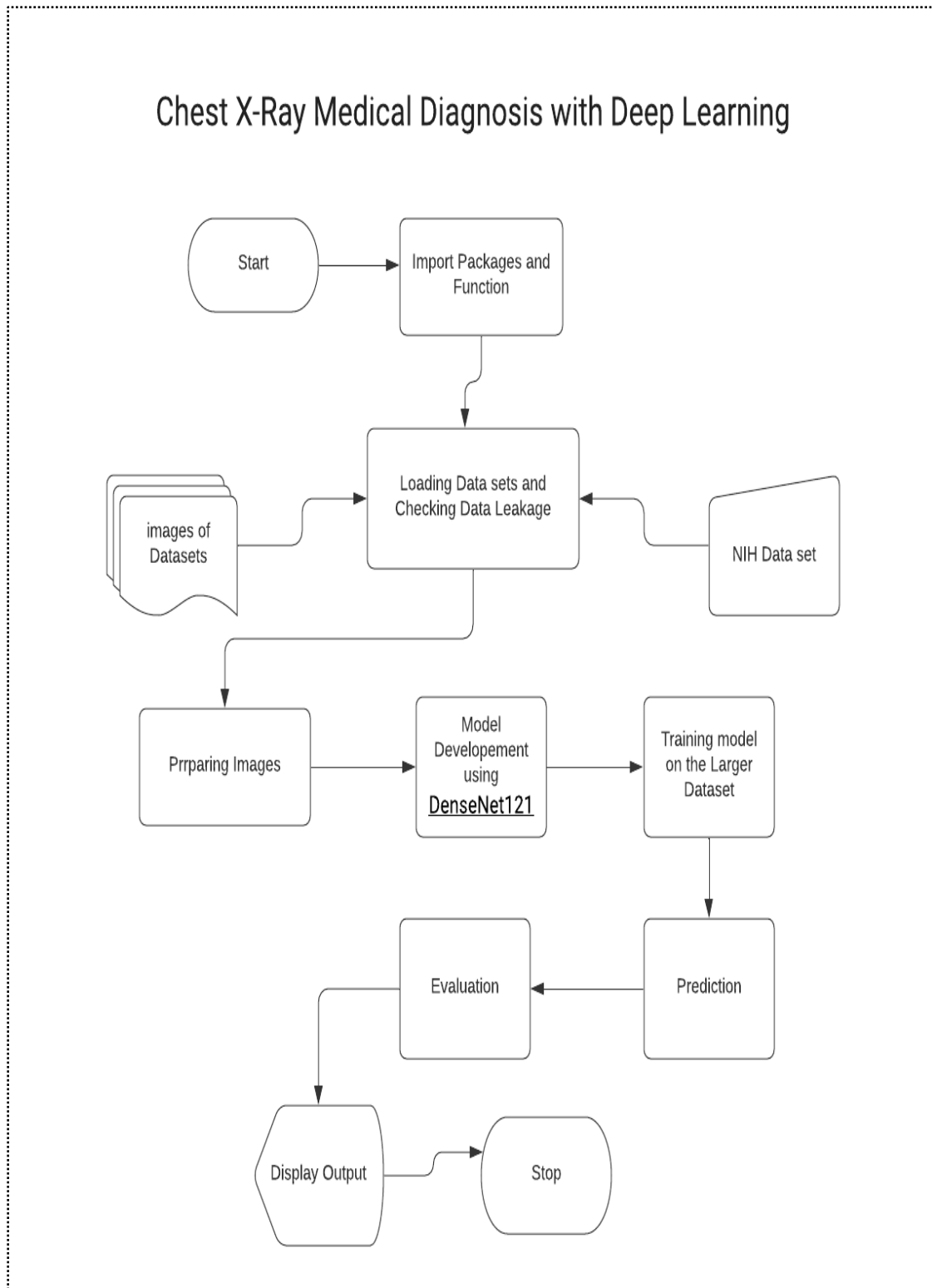
In today’s world of **automation**, we have to find ways which can help doctors in their work, checking if the patient is diagnosed with one of the 14 diseases mentioned above, this approach is reliable, quick, works 24/7 without getting stressed.

The best part about this model is that it not only detects if the person has been diagnosed with the disease or not, but it maps it out to where that particular disease is, and it even dates it's that if the person is diagnosed with more than one disease.

### **Note :**

The data I used is from the National Institutes of Health (NIH)

## 2. Work Flow of the Project



### **3. Technologies Used**

I have used **Deep Learning** as the data contains images on which we train our model. I even used data augmentation, so that even if we have a small dataset, the model performs better.

#### **Tools and technologies-**

##### **3.1. Deep learning:**

For better accuracy of the model, because accuracy is the most important thing in Medical. Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example.

Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

1. Deep learning requires large amounts of **labeled data**. For example, driverless car development requires millions of images and thousands of hours of video.
2. Deep learning requires substantial **computing power**. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.



### **3.2. DenseNet:**

I have used DenseNet121 which I have used as a pre-trained model, and then we have added two layers on top of it. DenseNet is one of the discoveries in neural networks for visual object recognition. DenseNet is quite similar to ResNet with some fundamental differences. ResNet uses an additive method (+) that merges the previous layer (identity) with the future layer, whereas DenseNet concatenates (.) the output of the previous layer with the future layer.

Why Do We DenseNet?

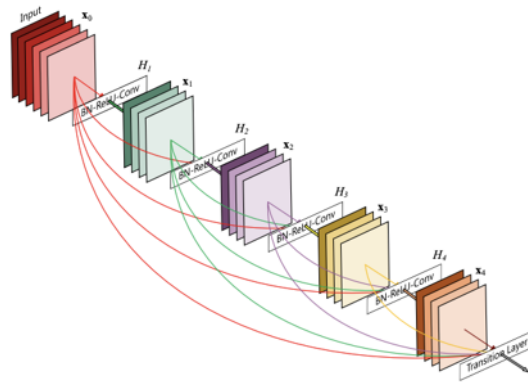
DenseNet was developed specifically to improve the declined accuracy caused by the vanishing gradient in high-level neural networks. In simpler terms, due to the longer path between the input layer and the output layer, the information vanishes before reaching its destination. DenseNet falls in the category of classic networks.

**Figure 1** shows a 5-layer dense block with a *growth rate* of  $k = 4$  and the standard ResNet structure.

An output of the previous layer acts as an input of the second layer by using composite function operation. This composite operation consists of the convolution layer, pooling layer, batch normalization, and non-linear activation layer.

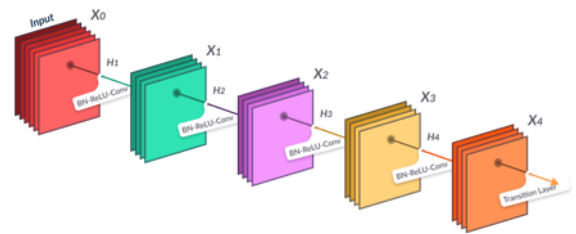
These connections mean that the network has  $L(L+1)/2$  direct connections.  $L$  is the number of layers in the architecture.

The DenseNet has different versions, like DenseNet-121, DenseNet-160, DenseNet-201, etc. The numbers denote the number of layers in the neural network.



DenseNet Structure

$$a^{[l]} = g([a^{[0]}, a^{[1]}, a^{[2]}, \dots, a^{[l-1]}])$$



ResNet Structure

$$a^{[l]} = g(z^{[l+1]} + a^{[l]})$$

**Figure 1**

The number 121 is computed as follows:

DenseNet-121:-  
 $5 + (6 + 12 + 24 + 16) * 2 = 121$

5 – Convolution and Pooling Layer  
 3 – Transition layers (6,12,24)  
 1 – Classification Layer (16)  
 2 – DenseBlock (1x1 and 3x3 conv)

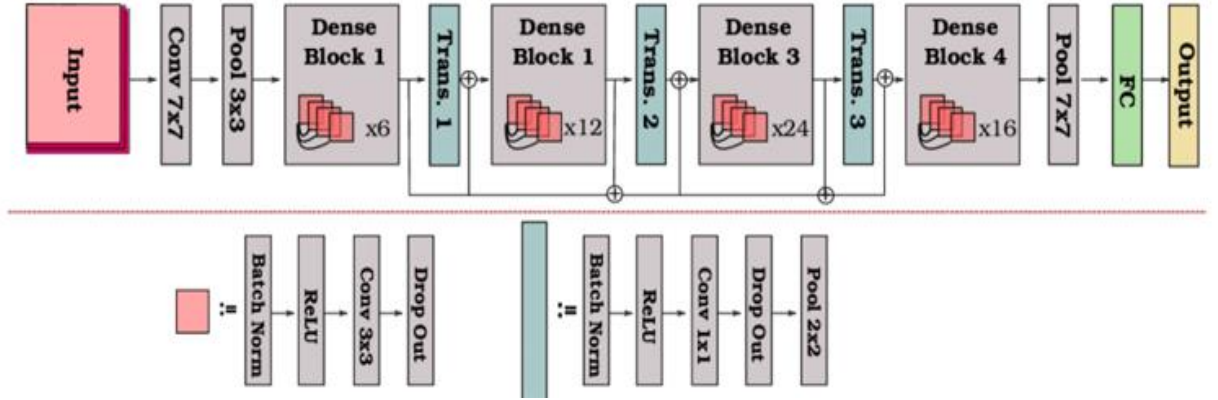
### DenseBlocks and Layers

Be it adding or concatenating, the grouping of layers by the above equation is only possible if feature map dimensions are the same.

What if dimensions are different?

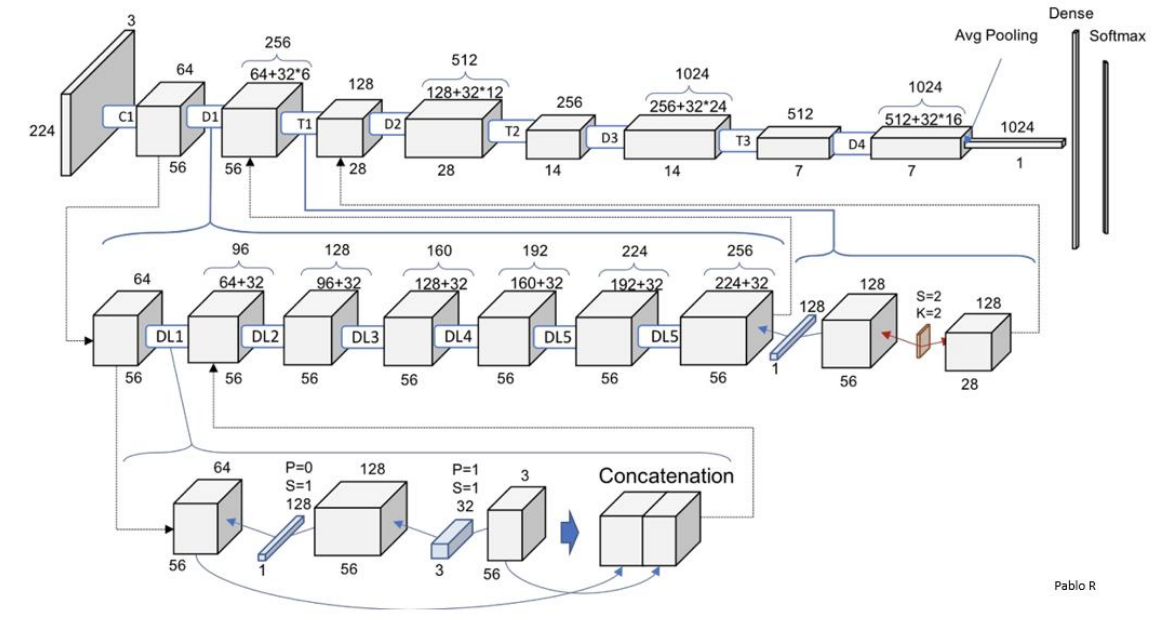
The DenseNet is divided into DenseBlocks where some filters are different, but dimensions within the block are the same. *Transition Layer* applies batch normalization using downsampling; it's an essential step in CNN.

Let's see what's inside the DenseBlock and transition layer.:



**Figure 2**

The full architecture in abstract form.:



**Figure 3**

The number of filters changes between the DenseBlocks, increasing the dimensions of the channel. The growth rate ( $k$ ) helps in generalizing the 1<sup>st</sup> layer. It controls the amount of information to be added to each layer.

$$k^{[l]} = (k^{[0]} + k(l - 1))$$

### **3.3. Dense Net 121:**

#### **Densely Connected Convolutional Networks**

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion.

Whereas traditional convolutional networks with  $L$  layers have  $L$  connections - one between each layer and its subsequent layer - our network has  $L(L+1)/2$  direct connections.

For each layer, the feature-maps of all preceding layers are used as inputs, and its feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less memory and computation to achieve high performance.

### **3.4. Keras:**

Keras is a high-level API built on TensorFlow (and can be used on top of Theano too). It is more user-friendly and easy to use as compared to TF. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation.

The core data structures of Keras are layers and models. The simplest type of model is the Sequential model, a linear stack of layers. For more complex architectures, you should use the Keras functional API, which allows you to build arbitrary graphs of layers, or write models entirely from scratch via subclassing.

### **3.5. ROC Curve And AUROC:** For model evaluation.

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters :

- a) True Positive Rate
- b) False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

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

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

A ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

TP vs. FP rate at different classification thresholds.

To compute the points in a ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

**AUC: Area Under the ROC Curve**

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

**AUC (Area under the ROC Curve).**

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:

### **3.6. GradCAM :**

To visualize where the model detects the diagnoses.

A technique for making Convolutional Neural Network (CNN)-based models more transparent by visualizing the regions of input that are "important" for predictions from these models - or visual explanations. Gradient-weighted Class Activation Mapping (Grad-CAM), uses the class-specific gradient information flowing into the final convolutional layer of a CNN to produce a coarse localization map of the important regions in the image.

Grad-CAM is a strict generalization of the Class Activation Mapping. Unlike CAM, Grad-CAM requires no re-training and is broadly applicable to any CNN-based architectures. We also show how Grad-CAM may be combined with existing pixel-space visualizations to create a high-resolution class-discriminative visualization (Guided Grad-CAM).

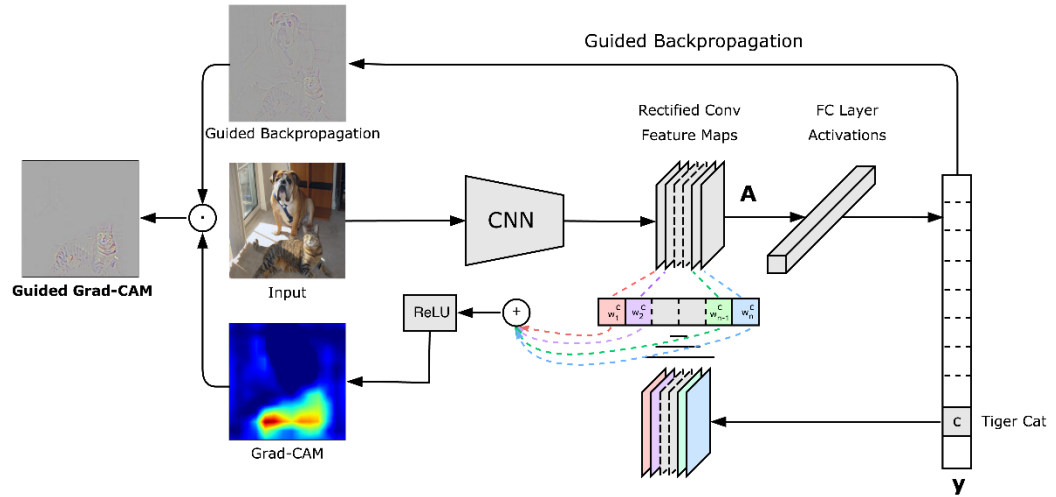
We generate Grad-CAM and Guided Grad-CAM visual explanations to better understand image classification, image captioning, and visual question answering (VQA) models. In the context of image classification models, our visualizations

- (a) lend insight into their failure modes showing that seemingly unreasonable predictions have reasonable explanations, and
- (b) outperform pixel-space gradient visualizations (Guided Backpropagation and Deconvolution) on the ILSVRC-15 weakly supervised localization task.

For image captioning and VQA, our visualizations expose the somewhat surprising insight that common CNN + LSTM models can often be good at localizing discriminative input image regions despite not being trained on grounded image-text pairs. Finally, we design and conduct human studies to measure if Guided Grad-CAM explanations help users establish trust in the predictions made by deep networks.

Interestingly, we show that Guided Grad-CAM helps untrained users successfully discern a "stronger" deep network from a "weaker" one even when

both networks make identical predictions.



***Figure 4***

### **3.7. NumPy:**

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.

The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.

A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

The points about sequence size and speed are particularly important in scientific computing. As a simple example, consider the case of multiplying each element in a 1-D sequence with the corresponding element in another sequence of the same length.

### **3.8. Seaborn:**

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them

### **3.9. Matplotlib:**

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPython or Tkinter. It can be used in Python and IPython shells, Jupyter notebook and web application servers also.

Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks. Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.



Matplotlib was originally written by John D. Hunter in 2003. The current stable version is 2.2.0 released in January 2018.

### **3.10. Pyplot:**

Pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

### **3.11. Image Data Generator:**

When working with deep learning models, I have often found myself in a peculiar situation when there is not much data to train my model. It was in times like these when I came across the concept of image augmentation. The image augmentation technique is a great way to expand the size of your dataset. You can come up with new transformed images from your original dataset. But many people use the conservative way of augmenting the images i.e. augmenting images and storing them in a numpy array or in a folder. I have got to admit, I used to do this until I stumbled upon the ImageDataGenerator class.

Keras **ImageDataGenerator** is a gem, It lets you augment your images in real-time while your model is still training! You can apply any random transformations on each training image as it is passed to the model. This will not only make your model robust but will also save up on the overhead memory

### **3.12. Pandas:**

To read the CSV file.

**Pandas** is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader

goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet

Ordered and unordered (not necessarily fixed-frequency) time series data.

Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels

Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional objects
- Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations
- Powerful, flexible **group by** functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based **slicing, fancy indexing**, and **subsetting** of large data sets
- Intuitive **merging** and **joining** data sets
- Flexible **reshaping** and pivoting of data sets

**Hierarchical** labeling of axes (possible to have multiple labels per tick)

Robust IO tools for loading data from **flat files** (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast **HDF5 format**  
**Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.

### **3.13. GlobalAveragePooling2D:**

Conventional convolutional neural networks perform convolution in the lower layers of the network. For classification, the feature maps of the last convolutional layer are vectorized and fed into fully connected layers followed by a softmax logistic regression layer . This structure bridges the convolutional structure with traditional neural network classifiers. It treats the convolutional layers as feature extractors, and the resulting feature is classified in a traditional way. However, the fully connected layers are prone to overfitting, thus hampering the generalization ability of the overall network. Dropout is proposed by Hinton et al. ,as a regularizer which randomly sets half of the activations to the fully connected layers to zero during training. It has improved the generalization ability and largely prevents overfitting. In this paper, we propose another strategy called global average pooling to replace the traditional fully connected layers in CNN. The idea is to generate one feature map for each corresponding category of the classification task in the last mlpconv layer. Instead of adding fully connected layers on top of the feature maps, we take the average of each feature map, and the resulting vector is fed directly into the softmax layer. One advantage of global average pooling over the fully connected layers is that it is more native to the convolution structure by enforcing correspondences between feature maps and categories. Thus the feature maps can be easily interpreted as categories confidence maps. Another advantage is that there is no parameter to optimize in the global average pooling thus overfitting is avoided at this layer. Futhermore, global average pooling sums out the spatial information, thus it is more robust to spatial translations of the input. We can see global average pooling as a structural regularizer that explicitly enforces feature maps to be confidence maps of

concepts (categories). This is made possible by the mlpconv layers, as they makes better approximation to the confidence maps than GLMs.

#### **4.Diseases that can be detected**

**Cardiomegaly:** it is an enlarged heart. It is not a disease, but a sign of another condition. Less severe forms of cardiomegaly are referred to as mild cardiomegaly. As mild cardiomegaly does not always cause symptoms, many people with a slightly enlarged heart are unaware of the problem

**Emphysema:** it is a lung condition that causes shortness of breath. In people with emphysema, the air sacs in the lungs (alveoli) are damaged. Over time, the inner walls of the air sacs weaken and rupture — creating larger air spaces instead of many small ones.

**Effusion:** Too much fluid, an outpouring of fluid. For example, a pleural effusion is an abnormal accumulation of fluid in the pleural space between the lungs and the chest wall, while a knee effusion is an abnormal amount of fluid in the knee joint. A hemorrhagic effusion contains blood in the fluid.

**Hernia:** is the abnormal exit of tissue or an organ, such as the bowel, through the wall of the cavity in which it normally resides. Hernias come in a number of types. Most commonly they involve the abdomen, specifically the groin. Groin hernias are most commonly of the inguinal type but may also be femoral.

**Infiltration:** is a substance denser than air, such as pus, blood, or protein, which lingers within the parenchyma of the lungs. Pulmonary infiltrates are associated with pneumonia, and tuberculosis. Pulmonary infiltrates can be observed on a chest radiograph.

**MASS syndrome:** is a medical disorder of the connective tissue similar to Marfan syndrome. MASS stands for: Mitral valve prolapse, Aortic root diameter at upper limits of normal for body size, Stretch marks of the skin, and Skeletal conditions similar to Marfan syndrome.

**Nodule:** is a growth of abnormal tissue. Nodules can develop just below the skin. They can also develop in deeper skin tissues or internal organs. Dermatologists use nodules as a general term to describe any lump underneath the skin that's at least 1 centimeter in size.

**Atelectasis:** (at-uh-LEK-tuh-sis) is a complete or partial collapse of the entire lung or area (lobe) of the lung. It occurs when the tiny air sacs (alveoli) within the lung become deflated or possibly filled with alveolar fluid. Atelectasis is one of the most common breathing (respiratory) complications after surgery

**Pneumothorax:** It is a collapsed lung. A pneumothorax occurs when air leaks into the space between your lung and chest wall. This air pushes on the outside of your lung and makes it collapse. Pneumothorax can be a complete lung collapse or a collapse of only a portion of the lung

**Pleural thickening:** It develops when scar tissue thickens the delicate membrane lining the lungs (the pleura). Pleural thickening can develop following asbestos exposure or other conditions, such as infection. It may be a symptom of a more severe diagnosis such as malignant pleural mesothelioma.

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

**Fibrosis:**, also known as fibrotic scarring, is a pathological wound healing in which connective tissue replaces normal parenchymal tissue to the extent that it goes unchecked, leading to considerable tissue remodelling and the formation of permanent scar tissue.

**Edema:** is the medical term for swelling. Body parts swell from injury or inflammation. It can affect a small area or the entire body. Medications, pregnancy, infections, and many other medical problems can cause edema. Edema happens when your small blood vessels leak fluid into nearby tissues.

**Lung Consolidation:** occurs when the air that usually fills the small airways in your lungs is replaced with something else. Depending on the cause, the air may be replaced with: a fluid, such as pus, blood, or water. a solid, such as stomach contents or cells.

## **5. NIH Data**

### **National Institutes of Health Chest X-Ray Dataset:**

Chest X-ray exams are one of the most frequent and cost-effective medical imaging examinations available. However, clinical diagnosis of a chest X-ray can be challenging and sometimes more difficult than diagnosis via chest CT imaging. The lack of large publicly available datasets with annotations means it is still very difficult, if not impossible, to achieve clinically relevant computer-aided detection and diagnosis (CAD) in real world medical sites with chest X-rays. One major hurdle in creating large X-ray image datasets is the lack resources for labeling so many images. Prior to the release of this dataset, Openi was the largest publicly available source of chest X-ray images with 4,143 images available.

This NIH Chest X-ray Dataset is comprised of 112,120 X-ray images with disease labels from 30,805 unique patients. To create these labels, the authors used Natural Language Processing to text-mine disease classifications from the associated radiological reports. The labels are expected to be >90% accurate and suitable for weakly-supervised learning. The original radiology reports are not publicly available but you can find more details on the labeling process in this Open Access paper: "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases." (Wang *et al.*)

### **Data limitations:**

1. The image labels are NLP extracted so there could be some erroneous labels but the NLP labeling accuracy is estimated to be >90%.
2. Very limited numbers of disease region bounding boxes
3. Chest x-ray radiology reports are not anticipated to be publicly shared. Parties who use this public dataset are encouraged to share their "updated" image labels and/or new bounding boxes in their own studied later, maybe through manual annotation

### **Full Dataset Content**

There are 12 zip files in total and range from ~2 gb to 4 gb in size. Additionally, we randomly sampled 5% of these images and created a smaller dataset for use in Kernels. The random sample contains 5606 X-ray images and class labels.

### **Modifications to original data**

- Original TAR archives were converted to ZIP archives to be compatible with the Kaggle platform
- CSV headers slightly modified to be more explicit in comma separation and also to allow fields to be self-explanatory

## 6. Code

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.densenet import DenseNet121
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model
from keras import backend as K

from keras.models import load_model

import util

train_df = pd.read_csv("nih/train-small.csv")
valid_df = pd.read_csv("nih/valid-small.csv")

test_df = pd.read_csv("nih/test.csv")

train_df.head()

labels = ['Cardiomegaly',
          'Emphysema',
          'Effusion',
          'Hernia',
          'Infiltration',
          'Mass',
          'Nodule',
          'Atelectasis',
          'Pneumothorax',
```

```
'Pleural_Thickening',  
'Pneumonia',  
'Fibrosis',  
'Edema',  
'Consolidation']
```

```
# Checking for Data Leakage
```

```
def check_for_leakage(df1, df2, patient_col):
```

```
    df1_patients_unique = set(df1[patient_col].values)
```

```
    df2_patients_unique = set(df2[patient_col].values)
```

```
    patients_in_both_groups
```

```
=
```

```
    df1_patients_unique.intersection(df2_patients_unique)
```

```
    leakage = len(patients_in_both_groups) > 0
```

```
    return leakage
```

```
print("leakage between train and test: {}".format(check_for_leakage(train_df,  
test_df, 'PatientId')))
```

```
print("leakage between valid and test: {}".format(check_for_leakage(valid_df,  
test_df, 'PatientId')))
```

```
def get_train_generator(df, image_dir, x_col, y_cols, shuffle=True, batch_size=8,  
seed=1, target_w = 320, target_h = 320):
```

```
    print("getting train generator...")
```

```
# normalize images
```

```
image_generator = ImageDataGenerator(  
    samplewise_center=True,  
    samplewise_std_normalization=True)
```



```

# flow from directory with specified batch size
# and target image size
generator = image_generator.flow_from_dataframe(
    dataframe=df,
    directory=image_dir,
    x_col=x_col,
    y_col=y_cols,
    class_mode="raw",
    batch_size=batch_size,
    shuffle=shuffle,
    seed=seed,
    target_size=(target_w,target_h))

return generator

```

```

def get_test_and_valid_generator(valid_df, test_df, train_df, image_dir, x_col,
y_cols, sample_size=100, batch_size=8, seed=1, target_w = 320, target_h = 320):

```

```

    print("getting train and valid generators...")

```

```

# get generator to sample dataset
raw_train_generator = ImageDataGenerator().flow_from_dataframe(
    dataframe=train_df,
    directory=IMAGE_DIR,
    x_col="Image",
    y_col=labels,
    class_mode="raw",
    batch_size=sample_size,
    shuffle=True,
    target_size=(target_w, target_h))

```

```

# get data sample

```

```

batch = raw_train_generator.next()
data_sample = batch[0]


# use sample to fit mean and std for test set generator
image_generator = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True)


# fit generator to sample from training data
image_generator.fit(data_sample)


# get test generator
valid_generator = image_generator.flow_from_dataframe(
    dataframe=valid_df,
    directory=image_dir,
    x_col=x_col,
    y_col=y_cols,
    class_mode="raw",
    batch_size=batch_size,
    shuffle=False,
    seed=seed,
    target_size=(target_w,target_h))


test_generator = image_generator.flow_from_dataframe(
    dataframe=test_df,
    directory=image_dir,
    x_col=x_col,
    y_col=y_cols,
    class_mode="raw",
    batch_size=batch_size,
    shuffle=False,
    seed=seed,
    target_size=(target_w,target_h))

```

```
return valid_generator, test_generator
```

```
IMAGE_DIR = "nih/images-small/"
train_generator = get_train_generator(train_df, IMAGE_DIR, "Image", labels)
valid_generator, test_generator = get_test_and_valid_generator(valid_df, test_df,
train_df, IMAGE_DIR, "Image", labels)
```

```
#for plotting a line graph
plt.xticks(rotation=90)
plt.bar(x=labels, height=np.mean(train_generator.labels, axis=0))
plt.title("Frequency of Each Class")
plt.show()
```

```
def compute_class_freqs(labels):
```

```
    # total number of patients (rows)
    N = labels.shape[0]
```

```
    positive_frequencies = np.sum(labels, axis=0) / N
    negative_frequencies = 1 - positive_frequencies
```

```
    return positive_frequencies, negative_frequencies
```

```
freq_pos, freq_neg = compute_class_freqs(train_generator.labels)
freq_pos
```

```
data = pd.DataFrame({"Class": labels, "Label": "Positive", "Value": freq_pos})
data = data.append([{"Class": labels[l], "Label": "Negative", "Value": v} for l,v in
enumerate(freq_neg)], ignore_index=True)
plt.xticks(rotation=90)
f = sns.barplot(x="Class", y="Value", hue="Label", data=data)
```

```
pos_weights = freq_neg
neg_weights = freq_pos
pos_contribution = freq_pos * pos_weights
neg_contribution = freq_neg * neg_weights
```

```

data = pd.DataFrame({"Class": labels, "Label": "Positive", "Value":
pos_contribution})
data = data.append([{"Class": labels[l], "Label": "Negative", "Value": v}
                    for l,v in enumerate(neg_contribution)], ignore_index=True)
plt.xticks(rotation=90)
sns.barplot(x="Class", y="Value", hue="Label", data=data);

```

```

def get_weighted_loss(pos_weights, neg_weights, epsilon=1e-7):

```

```

    def weighted_loss(y_true, y_pred):

        loss = 0.0

        for i in range(len(pos_weights)):
            # for each class, add average weighted loss for that class
            loss_pos = -1 * K.mean(pos_weights[i] * y_true[:, i] * K.log(y_pred[:, i] +
epsilon))
            loss_neg = -1 * K.mean(neg_weights[i] * (1 - y_true[:, i]) * K.log(1 -
y_pred[:, i] + epsilon))
            loss += loss_pos + loss_neg

        return loss

    ### END CODE HERE ###
    return weighted_loss

```

```

# create the base pre-trained model
base_model = DenseNet121(weights='./nih/densenet.hdf5', include_top=False)

```

```

x = base_model.output

```

```

# add a global spatial average pooling layer
x = GlobalAveragePooling2D()(x)

```

```

# and a logistic layer
predictions = Dense(len(labels), activation="sigmoid")(x)

```

```
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss=get_weighted_loss(pos_weights,
neg_weights))
```

### **#Training (Optional)**

**#Note** that we have already provided a pre-trained model, so you don't need to run the following training cell (as it will take some time).

#With our model ready for training, we will use the model.fit() function in Keras to train our model.

#We are training on a small subset of the dataset (~1%).

```
# """
```

# OPTIONAL: uncomment this code to practice training the model.

# This is optional because we have loaded pre-trained weights after this.

```
# """
```

```
# history = model.fit_generator(train_generator,
#                               validation_data=valid_generator,
#                               steps_per_epoch=100,
#                               validation_steps=25,
#                               epochs = 3)
```

```
# plt.plot(history.history['loss'])
# plt.ylabel("loss")
# plt.xlabel("epoch")
# plt.title("Training Loss Curve")
# plt.show()
```

```

model.load_weights("./nih/pretrained_model.h5")

predicted_vals = model.predict_generator(test_generator, steps =
len(test_generator))

auc_rocs = util.get_roc_curve(labels, predicted_vals, test_generator)

```

```

df = pd.read_csv("nih/train-small.csv")

IMAGE_DIR = "nih/images-small/"

```

```

#only show the lables with top 4 AUC

labels_to_show = np.take(labels, np.argsort(auc_rocs)[::-1])[4]

```

```

util.compute_gradcam(model, '00008270_015.png', IMAGE_DIR, df, labels,
labels_to_show)

```

```

util.compute_gradcam(model, '00011355_002.png', IMAGE_DIR, df, labels,
labels_to_show)

```

```

util.compute_gradcam(model, '00029855_001.png', IMAGE_DIR, df, labels,
labels_to_show)

```

```

util.compute_gradcam(model, '00005410_000.png', IMAGE_DIR, df, labels,
labels_to_show)

```

.....

## 7. Pictures of implimentation

```
In [3]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.densenet import DenseNet121
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model
from keras import backend as K
from keras.models import load_model

import util
```

```
In [4]: train_df = pd.read_csv("nih/train-small.csv")
valid_df = pd.read_csv("nih/valid-small.csv")

test_df = pd.read_csv("nih/test.csv")

train_df.head()
```

```
Out[4]:
```

	Image	Atelectasis	Cardiomegaly	Consolidation	Edema	Effusion	Emphysema	Fibrosis	Hernia	Infiltration	Mass	Nodule	PatientId	Pleural_Tl
0	00008270_015.png	0	0	0	0	0	0	0	0	0	0	0	8270	
1	00029855_001.png	1	0	0	0	1	0	0	0	1	0	0	29855	
2	00001297_000.png	0	0	0	0	0	0	0	0	0	0	0	1297	
3	00012359_002.png	0	0	0	0	0	0	0	0	0	0	0	12359	
4	00017951_001.png	0	0	0	0	0	0	0	0	1	0	0	17951	

```
In [8]: labels = ['Cardiomegaly',
                  'Emphysema',
                  'Effusion',
                  'Hernia',
                  'Infiltration',
                  'Mass',
                  'Nodule',
                  'Atelectasis',
                  'Pneumothorax',
                  'Pleural_Thickening',
                  'Pneumonia',
                  'Fibrosis',
                  'Edema',
                  'Consolidation']

In [4]: def check_for_leakage(df1, df2, patient_col):

    df1_patients_unique = set(df1[patient_col].values)
    df2_patients_unique = set(df2[patient_col].values)

    patients_in_both_groups = df1_patients_unique.intersection(df2_patients_unique)

    leakage = len(patients_in_both_groups) > 0

    return leakage
```

```
In [5]: # test
```

```

In [10]: # test
print("test case 1")
df1 = pd.DataFrame({'patient_id': [0, 1, 2]})
df2 = pd.DataFrame({'patient_id': [2, 3, 4]})
print(df1)
print(df2)
print("df1:")
print(df1)
print("df2:")
print(df2)
print(f"leakage output: {check_for_leakage(df1, df2, 'patient_id')}")
print("-----")
print("test case 2")
df1 = pd.DataFrame({'patient_id': [0, 1, 2]})
df2 = pd.DataFrame({'patient_id': [3, 4, 5]})
print(df1)
print(df2)
print("df1:")
print(df1)
print("df2:")
print(df2)
print(f"leakage output: {check_for_leakage(df1, df2, 'patient_id')}")

test case 1
df1
  patient_id
0          0
1          1
2          2
df2
  patient_id
0          2
1          3
2          4
leakage output: True
-----
test case 2
df1:
  patient_id
0          0
1          1
2          2
df2:
  patient_id
0          3
1          4
2          5
leakage output: False

```

#### Expected output

```

test case 1
df1
  patient_id
0          0
1          1
2          2
df2
  patient_id
0          2
1          3
2          4
leakage output: True
-----
test case 2
df1:
  patient_id
0          0
1          1
2          2
df2:
  patient_id
0          3
1          4
2          5
leakage output: False

```

```

In [6]: print("leakage between train and test: {}".format(check_for_leakage(train_df, test_df, 'PatientId')))
print("leakage between valid and test: {}".format(check_for_leakage(valid_df, test_df, 'PatientId')))

leakage between train and test: False
leakage between valid and test: False

```

```

In [7]: def get_train_generator(df, image_dir, x_col, y_cols, shuffle=True, batch_size=8, seed=1, target_w = 320, target_h = 320):
        print("getting train generator...")
        image_generator = ImageDataGenerator(
            samplewise_center=True,
            samplewise_std_normalization=True)
        generator = image_generator.flow_from_dataframe(
            dataframe=df,
            directory=image_dir,
            x_col=x_col,
            y_col=y_cols,
            class_mode="raw",
            batch_size=batch_size,
            shuffle=shuffle,
            seed=seed,
            target_size=(target_w, target_h))
        return generator

```

```

In [8]: def get_test_and_valid_generator(valid_df, test_df, train_df, image_dir, x_col, y_cols, sample_size=100, batch_size=8, seed=1, tar
        print("getting train and valid generators...")
        raw_train_generator = ImageDataGenerator().flow_from_dataframe(
            dataframe=train_df,
            directory=IMAGE_DIR,
            x_col="image",
            y_col=labels,
            class_mode="raw",
            batch_size=batch_size,
            shuffle=True,

```



```

        batch_size=batch_size,
        shuffle=False,
        seed=seed,
        target_size=(target_w,target_h))
    return valid_generator, test_generator

```

```

In [9]: IMAGE_DIR = "nih/images-small/"
train_generator = get_train_generator(train_df, IMAGE_DIR, "Image", labels)
valid_generator, test_generator = get_test_and_valid_generator(valid_df, test_df, train_df, IMAGE_DIR, "Image", labels)

getting train generator...
Found 1000 validated image filenames.
getting train and valid generators...
Found 1000 validated image filenames.
Found 200 validated image filenames.
Found 420 validated image filenames.

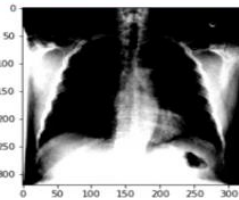
```

```

In [10]: x, y = train_generator.__getitem__(0)
plt.imshow(x[0]);

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

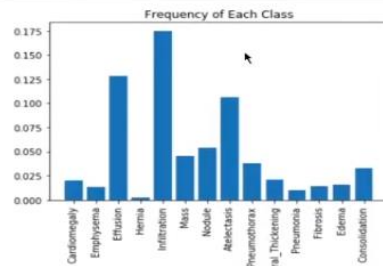
```



```

In [16]: plt.xticks(rotation=90)
plt.bar(x=labels, height=np.mean(train_generator.labels, axis=0))
plt.title("Frequency of Each Class")
plt.show()

```



```

In [17]: def compute_class_freqs(labels):
N = labels.shape[0]

positive_frequencies = np.sum(labels, axis=0) / N
negative_frequencies = np.sum(labels==0, axis=0) / N

return positive_frequencies, negative_frequencies

```

```

In [13]: # Test
labels_matrix = np.array(
[[1, 0, 0],
[0, 1, 1],
[1, 0, 1],
[1, 1, 1],
[1, 0, 1]]
)
print("labels:")
print(labels_matrix)

test_pos_freqs, test_neg_freqs = compute_class_freqs(labels_matrix)
print(f"pos freqs: {test_pos_freqs}")
print(f"neg freqs: {test_neg_freqs}")

labels:
[[1 0 0]
[0 1 1]
[1 0 1]
[1 1 1]
[1 0 1]]
pos freqs: [0.8 0.4 0.8]
neg freqs: [0.2 0.6 0.2]

```

#### Expected output

```

labels:
[[1 0 0]
[0 1 1]
[1 0 1]
[1 1 1]
[1 0 1]]
pos freqs: [0.8 0.4 0.8]
neg freqs: [0.2 0.6 0.2]

```

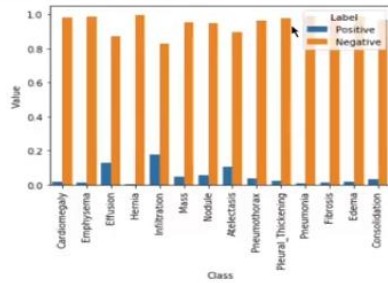
```

In [14]: freq_pos, freq_neg = compute_class_freqs(train_generator.labels)
freq_pos

```

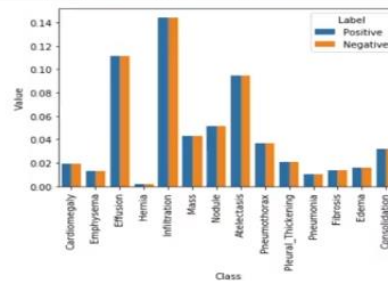
```
Out[19]: array([0.02, 0.013, 0.128, 0.002, 0.175, 0.045, 0.054, 0.106, 0.038,
0.021, 0.01, 0.014, 0.016, 0.033])
```

```
In [15]: data = pd.DataFrame({"Class": labels, "Label": "Positive", "Value": freq_pos})
data = data.append([{"Class": labels[l], "Label": "Negative", "Value": v} for l,v in enumerate(freq_neg)], ignore_index=True)
plt.xticks(rotation=90)
f = sns.barplot(x="Class", y="Value", hue="Label", data=data)
```



```
In [21]: pos_weights = freq_neg
neg_weights = freq_pos
pos_contribution = freq_pos * pos_weights
neg_contribution = freq_neg * neg_weights
```

```
In [17]: data = pd.DataFrame({"Class": labels, "Label": "Positive", "Value": pos_contribution})
data = data.append([{"Class": labels[l], "Label": "Negative", "Value": v} for l,v in enumerate(neg_contribution)], ignore_index=True)
plt.xticks(rotation=90)
sns.barplot(x="Class", y="Value", hue="Label", data=data);
```



```
In [18]: def get_weighted_loss(pos_weights, neg_weights, epsilon=1e-7):
def weighted_loss(y_true, y_pred):
loss = 0.0
for i in range(len(pos_weights)):
loss += -1 * K.mean(pos_weights[i] * y_true[:, i] * K.log(y_pred[:, i] + epsilon) +
neg_weights[i] * (1 - y_true[:, i]) * K.log(1 - y_pred[:, i] + epsilon)))
return loss
return weighted_loss
```

```
In [19]: # Test
sess = K.get_session()
# import tensorflow as tf
# sess = tf.compat.v1.keras.backend.get_session()
with sess.as_default() as sess:
print("Test example:\n")
y_true = K.constant(np.array(
[[1, 1, 1],
[1, 1, 0],
[0, 1, 0],
[1, 0, 1]]
))
print("y_true:\n")
print(y_true.eval())
...

w_p = np.array([0.25, 0.25, 0.5])
w_n = np.array([0.75, 0.75, 0.5])
print("\nw_p:\n")
print(w_p)

print("\nw_n:\n")
print(w_n)

y_pred_1 = K.constant(0.7*np.ones(y_true.shape))
print("\ny_pred_1:\n")
print(y_pred_1.eval())

y_pred_2 = K.constant(0.3*np.ones(y_true.shape))
print("\ny_pred_2:\n")
```

```

print("\nIf we weighted them correctly, we expect the two losses to be the same.")
l1 = L(y_true, y_pred_1).eval()
l2 = L(y_true, y_pred_2).eval()
print(f"\nL(y_pred_1)= {l1:.4f}, L(y_pred_2)= {l2:.4f}")
print(f"Difference is L1 - L2 = {l1 - l2:.4f}")

```

Test example:

```

y_true:
[[1. 1. 1.]
 [1. 1. 0.]
 [0. 1. 0.]
 [1. 0. 1.]]

w_p:
[0.25 0.25 0.5 ]

w_n:
[0.75 0.75 0.5 ]

y_pred_1:
[[0.7 0.7 0.7]
 [0.7 0.7 0.7]
 [0.7 0.7 0.7]
 [0.7 0.7 0.7]]

y_pred_2:
[[0.3 0.3 0.3]
 [0.3 0.3 0.3]
 [0.3 0.3 0.3]
 [0.3 0.3 0.3]]

```

If we weighted them correctly, we expect the two losses to be the same.

```

L(y_pred_1)= -0.4956, L(y_pred_2)= -0.4956
Difference is L1 - L2 = 0.0000

```

```

In [20]: base_model = DenseNet121(weights='./nih/densenet.hdf5', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(len(labels), activation="sigmoid")(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss=get_weighted_loss(pos_weights, neg_weights))

```

WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow\_core/python/ops/resource\_variable\_ops.py:1630: calling BaseResourceVariable.\_\_init\_\_ (from tensorflow.python.ops.resource\_variable\_ops) with constraint is deprecated and will be removed in a future version.  
Instructions for updating:  
If using Keras pass \*\_constraint arguments to layers.  
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow\_backend.py:4070: The name tf.nn.max\_pool is deprecated. Please use tf.nn.max\_pool2d instead.  
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow\_backend.py:4074: The name tf.nn.avg\_pool is deprecated. Please use tf.nn.avg\_pool2d instead.

```

In [ ]: ...
history = model.fit_generator(train_generator,
                             validation_data=valid_generator,
                             steps_per_epoch=100,
                             validation_steps=25,
                             epochs = 3)

plt.plot(history.history['loss'])
plt.ylabel("loss")
plt.xlabel("epoch")
plt.title("Training Loss curve")
plt.show()
...

```

```

In [21]: model.load_weights("./nih/pretrained_model.h5")

```

```

In [22]: predicted_vals = model.predict_generator(test_generator, steps = len(test_generator))

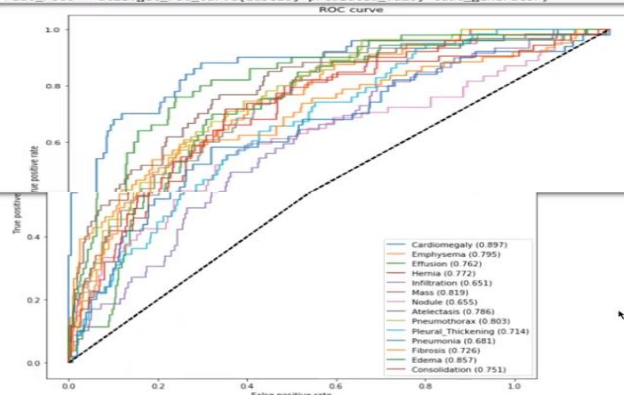
```

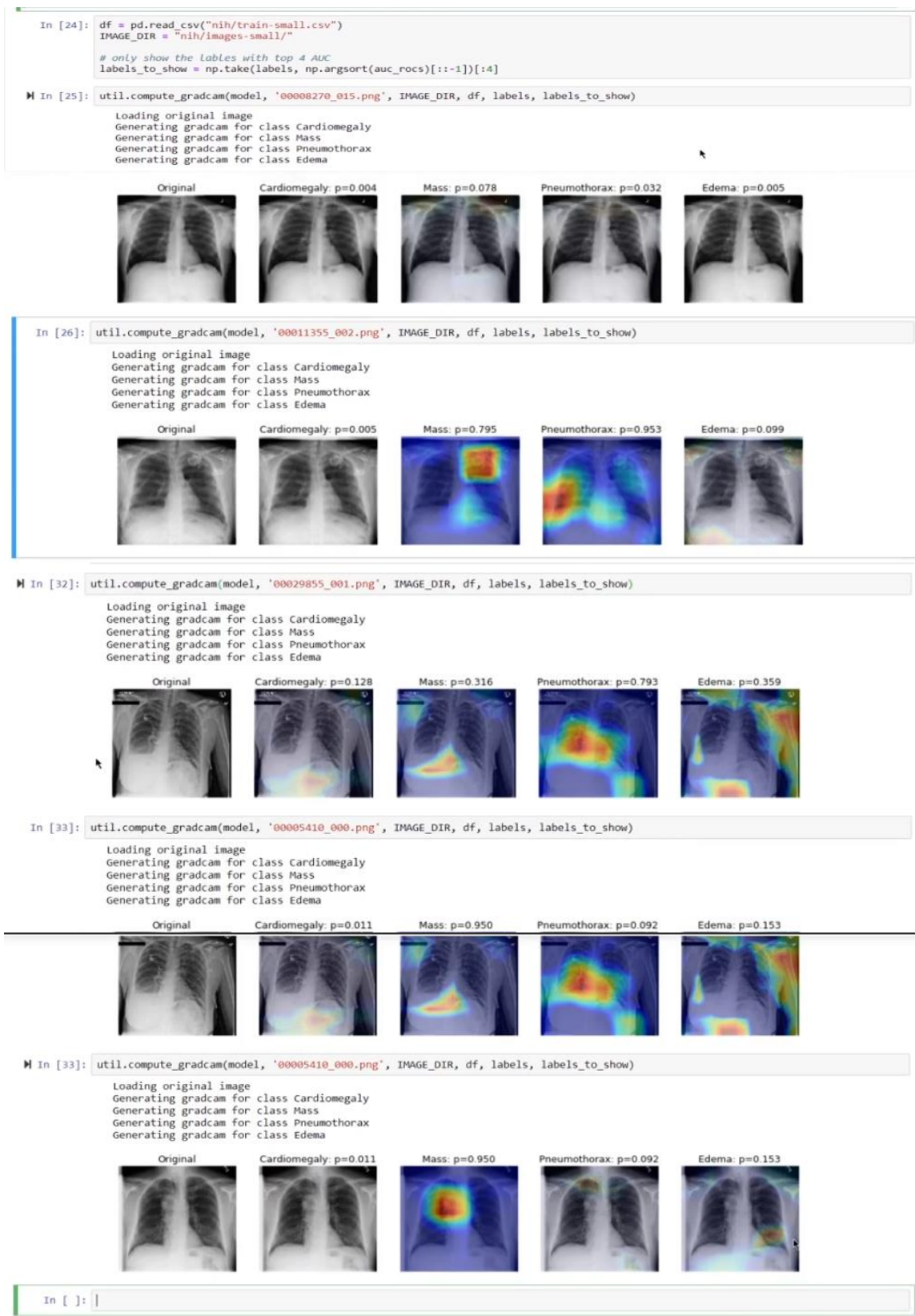
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow\_backend.py:422: The name tf.global\_variables is deprecated. Please use tf.compat.v1.global\_variables instead.

```

In [23]: auc_rocs = util.get_roc_curve(labels, predicted_vals, test_generator)

```





## **8. Conclusion**

I have built a state of the art chest X-Ray classifier using Keras, which will help doctors in detecting if the patient has been diagnosed with 14 different lung diseases. This project help using the NIH data set which is very good for the data processing. So we can say the data is also reliable. Here we are neglecting those disease having accuracy 0.09 or less than the this (may be three or more decimal number) we consider it as normal. The model is very helpful for speedy trials on patients because some time human mind get confused after see the x-ray reports. I am not saying we, should depend upon this result but it can be helpful to find quicker than traditional method . So doctors can visit some more number of patients. The queue in the hospitals may be shorter while they were standing for check ups. At least for these 14 disease I hope this project may be helpful.

## **9.Future Scope**

Yes, it has a future scope as we can train the model with the different datasets to detect different diseases, like **COVID-19**, and a GUI version can be made or even a **Mobile App** can be made. With the world experiencing different diseases, which are unheard of before, this model can be trained and tuned to detect them too. Or we can make a web-app with centralized processing system so that whole branches of the same hospital or medical institutions can save there data at one place so that will be helpful for future detection of diseases. And the number of disease we are detecting can also increased.

## **10. Bibliography**

- <https://www.pluralsight.com/guides/introduction-to-densenet-with-tensorflow>
- Figure 1: Sources: *DenseNet Structure* - G. Huang, Z. Liu, and L. van der Maaten, “Densely Connected Convolutional Networks,” 2018; *Resnet Structure* - [Missinglink.ai](https://www.missinglink.ai)
- Figure 2 : Source: G. Huang, Z. Liu, and L. van der Maaten, “Densely Connected Convolutional Networks,” 2018.
- Figure 3: Source: Pablo R
- DenseNet 121 : <https://www.kaggle.com/pytorch/densenet121>
- <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>
- <http://gradcam.cloudcv.org/>
- Figure 4: <http://gradcam.cloudcv.org/>
- [https://numpy.org/devdocs/user/what\\_is\\_numpy.html](https://numpy.org/devdocs/user/what_is_numpy.html)
- [https://www.tutorialspoint.com/matplotlib/matplotlib\\_introduction.htm](https://www.tutorialspoint.com/matplotlib/matplotlib_introduction.htm)
- <https://matplotlib.org/tutorials/introductory/pyplot.html>
- <https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/>
- <https://pypi.org/project/pandas/>
- <https://arxiv.org/pdf/1312.4400.pdf>
- <https://www.kaggle.com/nih-chest-xrays/data>
- Disease definitions: <https://www.google.com>