Capstone Project  
Machine Learning Engineer  
Nanodegree  
Meskhidze Konstantin  
October 6, 2018



Definition

### Project Overview

For the final project I’ve decided to join to one of competitions on [Kaggle.com](https://www.kaggle.com/c/home-credit-default-risk).

Kaggle.com launched a competition RSNA Pneumonia Detection Challenge. In this competition kagglers need to build a model to detect a visual signal pneumonia disease in a medical images of a patient’s chest and locate it. [Radiological Society of North America (RSNA®)](http://www.rsna.org/) and National Institutes of Health Clinical Center provide this competition with datasets. The society hopes this competition would help the diagnosis of pneumonia because the disease kills 15% of patients under 5 years old internationally.

Data Description

There are the following files in a dataset:

* **stage\_1\_train\_images.zip** and **stage\_1\_test\_images.zip** – training and test images.
* **stage\_1\_train.csv** **– training labels.**
* **stage\_1\_sample\_submission.csv -** provides the IDs for the test set, as well as a sample of what your submission should look like.
* **stage\_1\_detailed\_class\_info.csv -** contains detailed information about the positive and negative classes in the training set, and may be used to build more nuanced models.

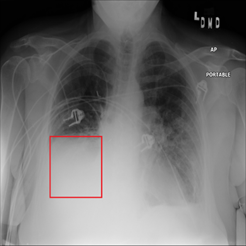
Data fields

* **patientId** \_- A patientId. Each patientId corresponds to a unique image.
* **x**\_ - the upper-left x coordinate of the bounding box.
* **y**\_ - the upper-left y coordinate of the bounding box.
* **width**\_ - the width of the bounding box.
* **height**\_ - the height of the bounding box.
* **Target**\_ - the binary Target, indicating whether this sample has evidence of pneumonia.

All files can be downloaded from <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data>

### Problem Statement

In a nutshell, I’m given a dataset of labeled patients’ chest images with bounding boxes of pneumonia locations like in the picture below:



I need to build an algorithm that finds and locates for pneumonia opacities in bounding boxes. In deep learning language, this task is called object detection. The best solution is using some kind of Convolutional Neural Network.

I’m going to train a CNN with images from **stage\_1\_train\_images.zip, using Target labels and** bounding boxes **from stage\_1\_train.csv**. Then the model is going to be tested using images from **stage\_1\_test\_images.zip. And finally a submission file with Patient’s ID, Target and bounding boxes features will be created and loaded on kaggle.com to evaluate an accuracy score.**

### Metrics

Unfortunately test images are unlabeled so to evaluate a model I’m going to use a part of train images (100-500) with Target labels. And the accuracy will be evaluated with a ROC-curve.

<https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

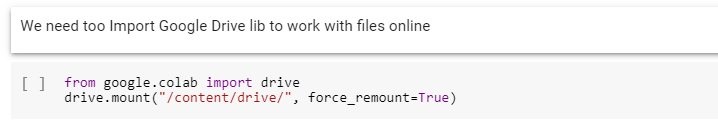
## Analysis

### Data Exploration

Dataset analysis as well as CNN training and testing was implemented using Google Colab service. It allows to launch Jupyter Notebooks online. It also provides NVidia Tesla K80 GPU for free. My notebook does not have powerful gpu so using Colab is very helpful CNN training speedup.

Dataset exploration is implemented in **cnn\_data\_analysis.ipynb** colab notebook.

First of all it’s needed to connect Google Drive because I save all information, CNN weights ect. in the google cloud.



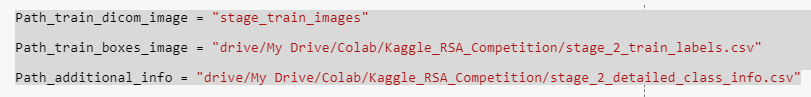
## After I download the dataset from Kaggle.com RSNA competition page:

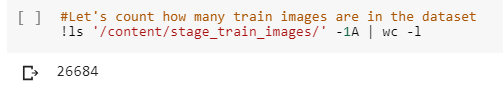
## 

Then I import needed libraries:



Then I define image train folder and path for bounding boxes and additional information csv. files:

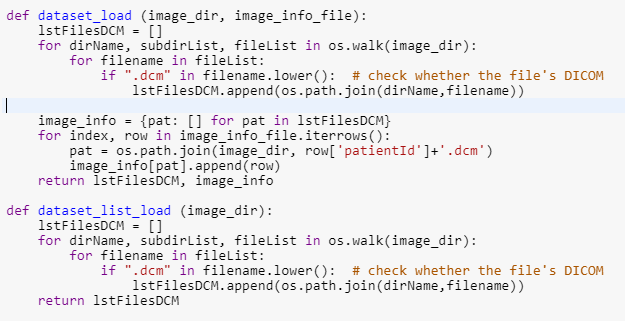




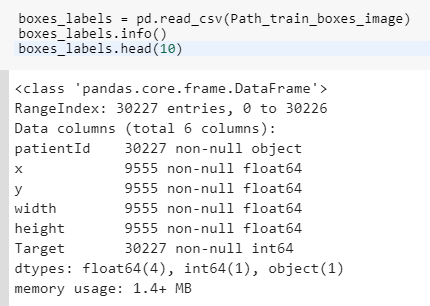
There are 2 function I prepared:

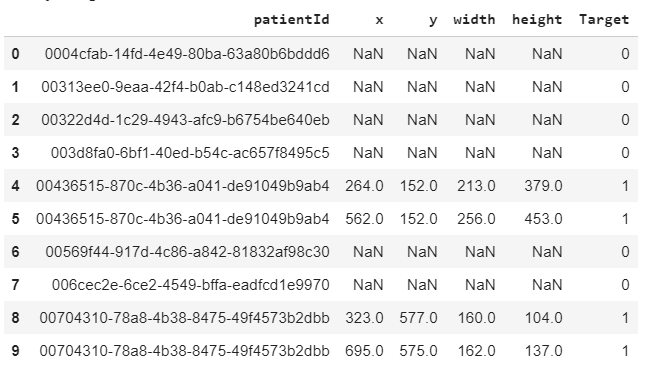
**dataset\_load()** - returns a list of dicom images in a folder and images' additional information.

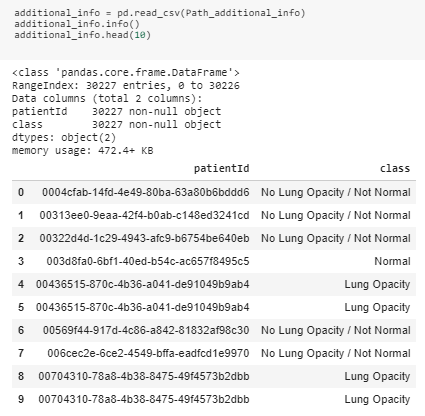
**dataset\_list\_load()** - returns a list of dicom images in a folder.



Let's analyze **stage\_2\_train\_labels.csv** and **stage\_2\_detailed\_class\_info.csv:**

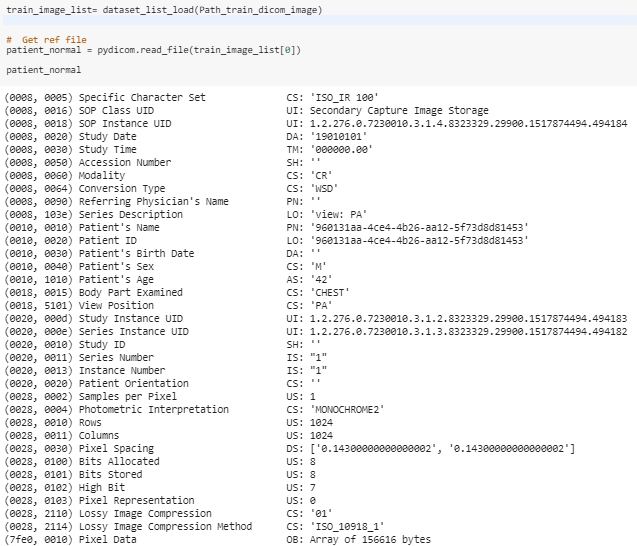






We can see that the train dataset has 26684 images but in **stage\_1\_train\_labels.csv** and **stage\_1\_detailed\_class\_info.csv** files there are 30227 patients Ids with 9555 nun-null Target features. That means that there could be situations when 1 patient has more than 1 Lung Opacity region.

Let’s analyze a dicom file :

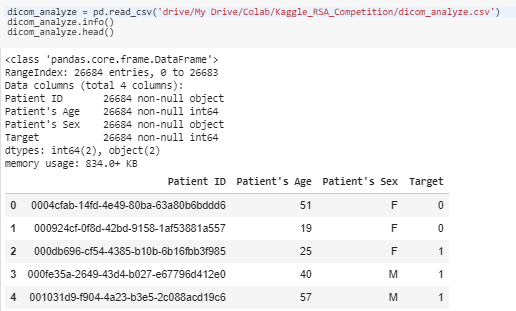


Here in file we are interested in Patient's Sex and Patient's Age features. This info might help somehow to improve the future model.

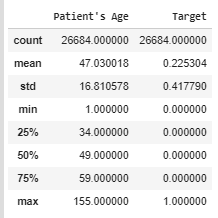
**dcmpanas.py** file has got some functions to build a new dataframe with Patient's Sex , Patient's Age and Target features . I use this new dataset to analyze the data.



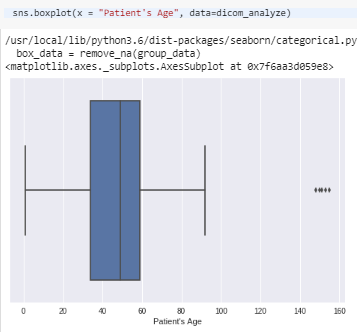
**scrape()** function passes 3 arguments: dicom images' folder, stage\_1\_train\_labels.csv location, and output.csv location.



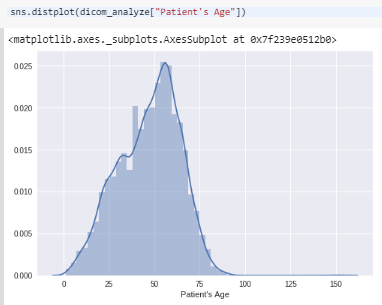
Let's do some data analysis:

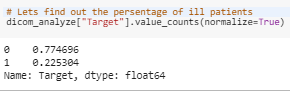


We can see that max patient's age value is 155 years. This value is definitely an outlier. But I'm a bit confused about the min value - 1 year old. Theoretically its possible that a toddler has been scanned by X-ray. But I consider this value is an outlier too.



From the boxplot we can see that we have several outliers > 100 years old. So 50% of all patients fall in a range between 34 and 59 years old.

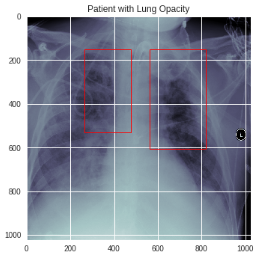
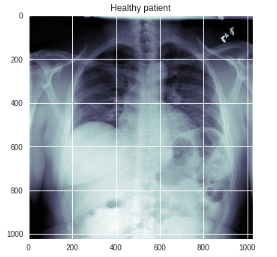




We can see that 22% of patients have pneumonia positive results.

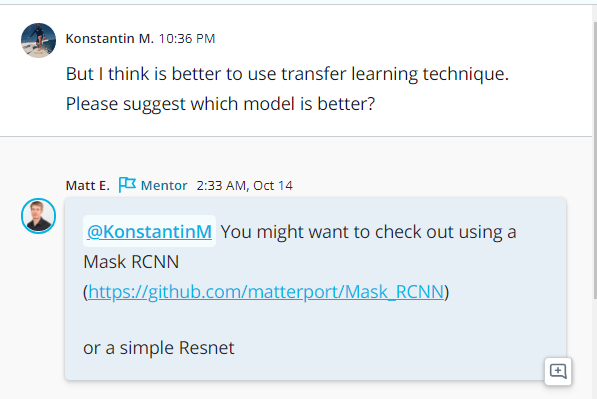
### Exploratory Visualization

Now let's plot a couple if images - with pneumonia opacity and without one. From box\_labels.head() we can see that patient **0004cfab-14fd-4e49-80ba-63a80b6bddd6** (index 0) has no lung opacity and the patient **00436515-870c-4b36-a041-de91049b9ab4** has 2 areas with lung opacity (4 and 5 indexes):



### Algorithms and Techniques

# I was a little bit confused finding an appropriate CNN for this competition. And finally I was suggested using Mask R-CNN for Object Detection and Segmentation:



This is an implementation of [Mask R-CNN](https://arxiv.org/abs/1703.06870) on Python 3, Keras, and TensorFlow. The model generates bounding boxes and segmentation masks for each instance of an object in the image. It's based on Feature Pyramid Network (FPN) and a ResNet101 backbone: <https://github.com/matterport/Mask_RCNN>

Explanation how Mask R-CNN works:

<https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272>

I’m going to use a transfer learning technique:

* Initializing CNN model with pretrained weights.
* Training model with my train dataset.
* Testing model my test data.

The model is trained on 2 datasets:

1. COCO dataset: <http://cocodataset.org/#home>
2. ImageNet dataset: <http://www.image-net.org/>

I’m going to use COCO weights to train my model.

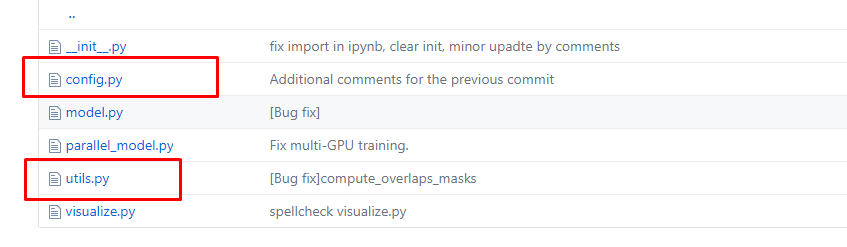
As an example I used **shapes** example on the github: <https://github.com/matterport/Mask_RCNN/blob/master/samples/shapes/train_shapes.ipynb>

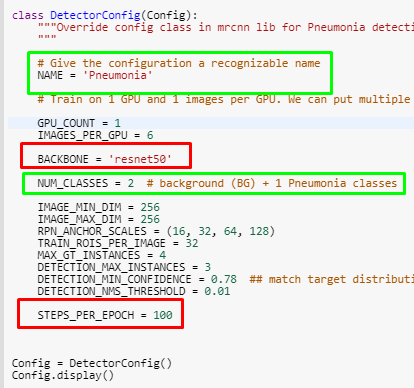
## Methodology

### Data Preprocessing

First of all we need to do some data processing before we start training our model

We need to extend 2 Classes in MRCNN model according the dataset we have: **Config** and **Utils.Dataset**

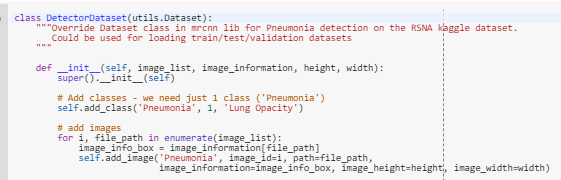




*Backbone* and *STEPS\_PER\_EPOCH* are the main parameters for model evaluation

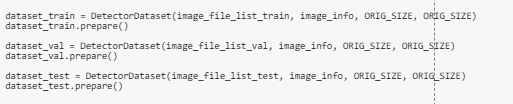
Cause for model training it asks object’s mask information we extend some functions in **Utils.Dataset** to create lung opacity masks.

-



Then I divide the train dataset into 3 smaller ones:

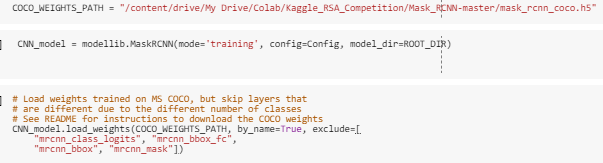
* Train
* Validation
* Test (local)



Every dataset now has original lung images and opacity masks for training step

### Implementation

In this step I train the model with configured parameters.

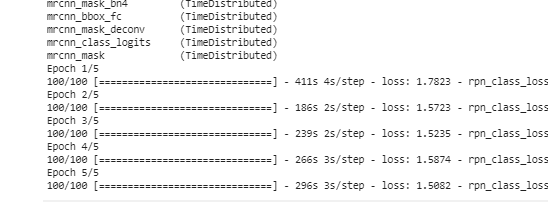
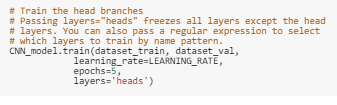


*Loading COCO weights into the model*

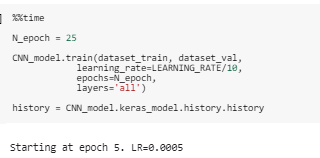
There two options here:

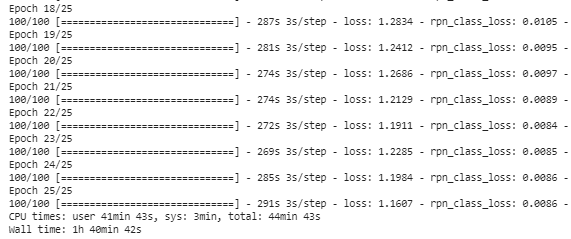
- Training Head layers.

- Training All layers.



*Training heads layers*



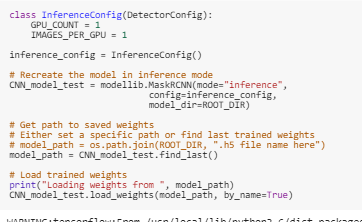


*Training All layers*

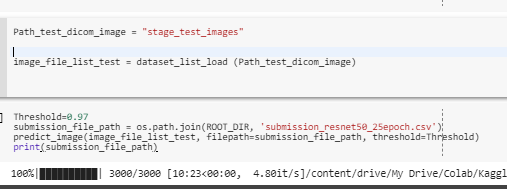
Model Weights are saved after each epoch on my Google Drive. From time to time I have to clean the old ones cause I just have about 5 GB of free space in the cloud.

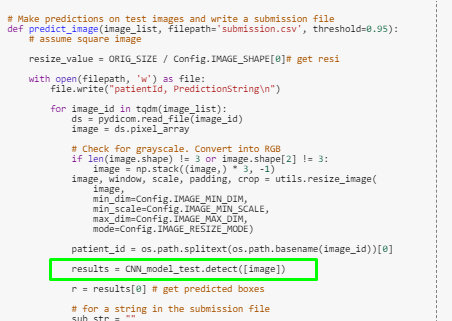
After I test trained model using test data:

I create a new **config** instance for testing and load trained wights:



Then I predict lung opacities on test image and create a submission file for Kaggle:

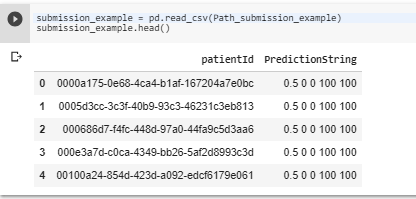




*Predict image function*

In *predict\_image* function I use *model.detect* function (internal MASK RCNN func) for testing. It boundary boxes as result.

A submission file for Kaggle looks like this:

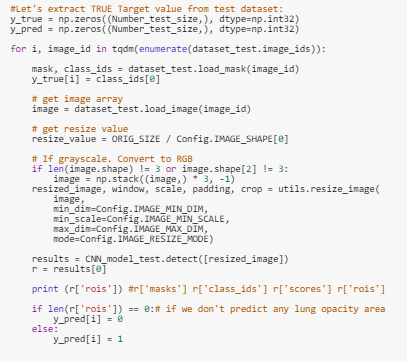


Prediction string has the next format: confidence x-min y-min width height

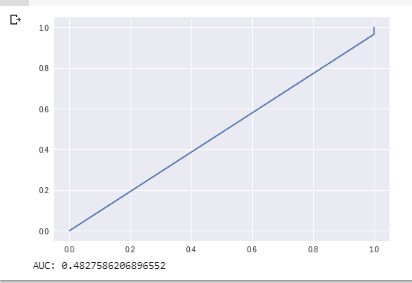
For local model evaluation I use 500 images from training dataset to plot the ROC curve:



We need to get Y\_TRUE and Y\_PRED Target vectors to plot ROC curve

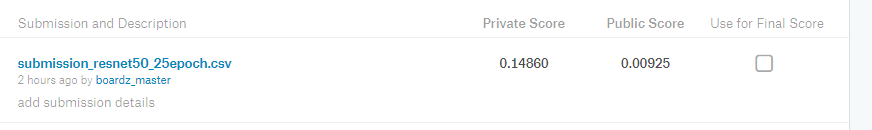


As a result I for RCNN training for 25 epochs I’ve got the next results AUC = 0.48:



*Model’s ROC curve*

And on Kaggle I’ve got 0.14860 accuracy score



### Refinement

Previous results showed that the model shows bad results. It’s no better than random guessing.

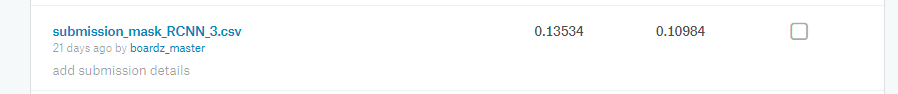
To improve the performance we could adjust the next parameters:

* Epoch numbers
* Batch step size
* Backbone of the model: **resnet50** or **resnet101**
* Data augmetation

## Results

### Model Evaluation and Validation

I've been playing with parameters for 4 weeks now and have tried different variants. And I have not reached somehow appropriate results – the model still does good accuracy. Google Colab sessions last for 12 hours so I could not train the model for a long time. The best result I have reached so far is 0.135 Public score, 0.10984 Private score:



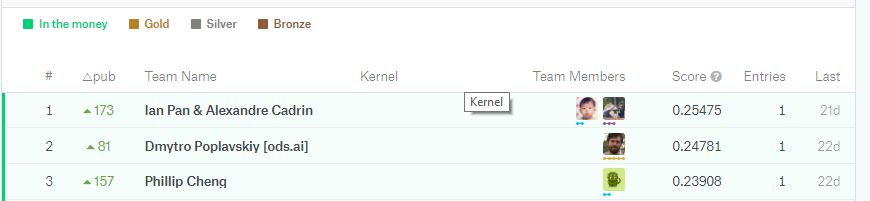
Training time was about 8 hours. Model parameters were:

* Epoch numbers 200.
* Batch step size 100.
* Backbone of the model: **resnet101.**

The ROC curve was still like 50% random guessing.

Conclusion

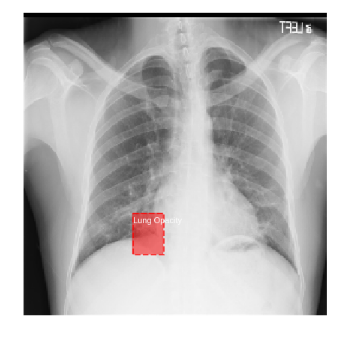
The model does not work well so far. Comparing with another results in the competition I state that the best result is also far away from satisfied detection:

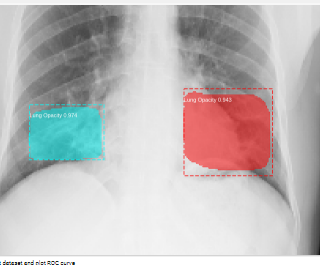


The best accuracy score is 0.25475.

I think the main reason of such poor results is quite small dataset for such type of images. Just 26684 train dicom files.

In the picture below there is a test image prediction example





*Original image*

*Predicted image*

We can see that original image has just one lung opacity region as well as predicted has 2 bigger ones.

Even for a human eye it’s hard to tell the difference between an ill patient and healthy one. For a CNN convolution layers it’s also hard to make a set of features cause there are no shape borders between lung opacity regions and other part of an image. So there is need a much bigger train dataset.

So to significantly improve the performance we need:

* Big train dataset (more 100k)
* Another structure of a CNN ( for fuzzy boundaries, homogenous image colors)
* More calculation recourses: powerful cluster of GPU, TPU, FPGA (Xilinx Aveo) for long time neural network training

<https://www.xilinx.com/products/boards-and-kits.html>

<https://www.xilinx.com/products/design-tools/acceleration-zone/aws.html>