

CSC 665 Term Project

CNN for Chest X-Ray Diagnosis

Spring 2018

AJ Culanay
Ian Dennis
Alvin Lee

Source Code (Github):

https://github.com/idennis7/CSC665_S18

Note: The data set is not included with the code

Link to Dataset (Kaggle):

<https://www.kaggle.com/nih-chest-xrays/data>

Write-Up (Google Docs):

<https://docs.google.com/document/d/1Eq0mHZSctfR22cQ0DilXJZ331u4u2D92Nm-OoNvQhI/>

Table of Contents

Overview	2
Goal	2
Background	2
Dataset	2
Software Used	3
Design and Implementation	4
Data Cleaning	4
CNN Implementation	7
Testing Structure	9
Results	10
Base Model (Using MobileNet, 128x128 image resolution)	10
Base Model (Using MobileNet, 256x256 image resolution)	12
Simple CNN (256x256 image resolution)	14
Evaluation	16
Results	16
Improvements	16

Overview

Goal

The goal of this project was to build a CNN that could classify whether a chest X-ray presented with a disease from a limited set of pathologies with a target accuracy of 75%.

Background

This dataset, and problem originate from the NIH research paper "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases." (*Wang, et al*). The paper describes the efforts by the research team to develop to create a DCNN (deeply convoluted neural network) to do multi-classification of common chest diseases in chest x-rays under weak supervision. The team also had the DCNN generate bounding boxes around suspected disease areas identified in the chest x-rays. The paper describes the process in which the team generated their database of labeled chest x-rays using NLP, the design of their DCNN using multiple, pre-trained ImageNet models as a basis, the hyper parameters they experimented with, and the results they achieved.

Link to the Research Paper:

http://openaccess.thecvf.com/content_cvpr_2017/papers/Wang_ChestX-ray8_Hospital-Scale_Chest_CVPR_2017_paper.pdf

Dataset

The dataset is composed of 112,120 chest x-ray images that have been anonymized to remove any identifying details pertaining to the 30,000 patients from which the x-rays originate from. These images have all been scaled to a uniform resolution of 1024x1024 pixels and were supplied by the NIH in an effort to have academics and researchers develop their own automated solutions for classifying chest x-rays. In total, the dataset was about 40GB worth of data.

The dataset was labeled through the use of natural language processing on the radiology notes from the chest x-rays. The accuracy of the labels is listed as >90% with the less than perfect accuracy attributed to using NLP to generate the labels.

The labels for each image were contained in a separate CSV file, which contained the following information about each image:

- Image Index (File name)
- Disease Type
- Follow up #
- Patient Age
- Patient Gender
- X-ray Orientation
- Patient ID
- Original Image Width
- Original Image Height
- Original Image Pixel Spacing x
- Original Image Pixel Spacing y

The disease type label broke down into 15 different classifications:

- Atelectasis
- Consolidation
- Infiltration
- Pneumothorax
- Edema
- Emphysema
- Fibrosis
- Effusion
- Pneumonia
- Pleural Thickening
- Cardiomegaly
- Nodule Mass
- Hernia
- No Finding

It should be noted that some chest x-rays in the dataset presented with multiple pathologies.

We used ~12000 of the total images for our training/testing (two zipped packages from the data set on Kaggle).

Software Used

- Python 3.5.4
- TensorFlow 1.4.0
- Keras 2.1.5
- Anaconda
- Jupyter Notebook

Design and Implementation

Data Cleaning

Our team applied data cleaning in hopes of improving our CNN's accuracy.

1. Dropped images with multiple disease classifications

Our team was worried that images with multiple disease classifications might affect our CNN's ability to accurately identify a specific disease (especially since disease areas might overlap in a single area of the image). For this reason, we opted to drop these types of images from our dataset.

```
# Drop images with multiple findings
xray_labels_df = xray_labels_df[~xray_labels_df['Finding Labels'].str.contains('|')]
```

2. Dropped images with poorly represented diseases

Taking inspiration from the aforementioned research paper, our team dropped images with disease classifications that had less than 200 instances (which seemed like a fair number given the distribution of findings). We figured it would be difficult to train the CNN to accurately identify x-rays with diseases that seldom showed up in the actual data set, because the CNN would be exposed to few examples of what that disease looked like.

Dropping these classifications left us with 7 remaining classifications: Atelectasis, Cardiomegaly, Effusion, Infiltration, Mass, Nodule, and Pneumothorax.

```
1 MIN_CASES = 200
2 all_labels = [c_label for c_label in all_labels if xray_labels_df[c_label].sum() > MIN_CASES]
3
4 print('Labels ({}).format(len(all_labels)),
5       [(c_label, int(xray_labels_df[c_label].sum())) for c_label in all_labels])
```

```
Labels (7) [('Atelectasis', 579), ('Cardiomegaly', 264), ('Effusion', 482), ('Infiltration', 1099), ('Mass', 246),
('Nodule', 397), ('Pneumothorax', 217)]
```

```
Findings Counts:
No Finding          10531
Infiltration        2292
Effusion            700
Atelectasis         692
Pneumothorax        445
Nodule              441
Mass                310
Pleural_Thickening  207
Consolidation        185
Cardiomegaly        146
Emphysema           112
Edema               90
Fibrosis            70
Pneumonia           61
Hernia              15
Name: Finding Labels, dtype: int64
```

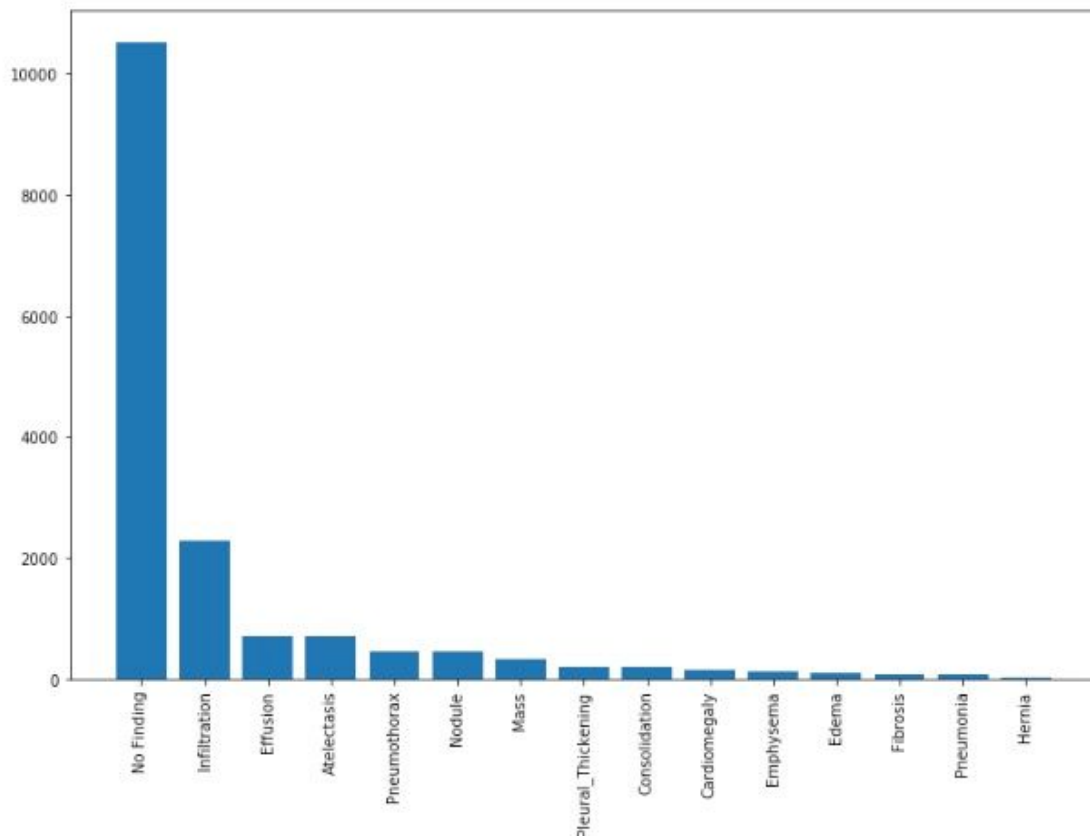


Image showing the counts and a graph of the different disease classifications after dropping images with multiple classifications, but before dropping poorly represented diseases.

3. Used one hot encoding to convert diseases classifications

The disease classifications given in the labels CSV were strings, and so needed to be converted into a type that would be usable by our CNN. To do this, we added a column for each disease type, and set the value to 1 if the disease was present in the image, otherwise it was set to 0.

```
In [51]: 1 from itertools import chain
2
3 # Replace Finding Labels with no finding to be blank
4 xray_labels_df['Finding Labels'] = xray_labels_df['Finding Labels'].map(lambda x: x.replace('No Finding', ''))
5
6 # Add columns to for each diagnosis to df
7 all_labels = np.unique(list(chain(*xray_labels_df['Finding Labels'].map(lambda x: x.split('|')).tolist()))
8 all_labels = [x for x in all_labels if len(x) > 0]
9
10 # Adds attributes for each possible finding, and assigns value of 1.0 for each finding
11 # in the Finding Label (1-Hot Encoding)
12 for c_label in all_labels:
13     if len(c_label) > 1:
14         xray_labels_df[c_label] = xray_labels_df['Finding Labels'].map(lambda finding: 1.0 if c_label in finding el
15
16 print(all_labels)
17 xray_labels_df.sample(5)
```

```
['Atelectasis', 'Cardiomegaly', 'Consolidation', 'Edema', 'Effusion', 'Emphysema', 'Fibrosis', 'Hernia', 'Infiltratio
n', 'Mass', 'Nodule', 'Pleural_Thickening', 'Pneumonia', 'Pneumothorax']
```

```
Out[51]:
```

/ldth	Height]	OriginalImagePixelSpacing[x ...	Effusion	Emphysema	Fibrosis	Hernia	Infiltration	Mass	Nodule	Pleural_Thickening	Pneumonia	Pneumothorax
2500	2048	0.171 ...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2992	2991	0.143 ...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3012	2476	0.139 ...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3056	2544	0.139 ...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3056	2544	0.139 ...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

CNN Implementation

Using a CNN from a pre-built Kaggle Kernel

We originally used a pre-built CNN to give our team a fast-path forward in terms of being able to manipulate the data, and get a baseline for results to compare against. We decided that this would provide a solid platform from which we could iterate on. Of course, we thoroughly picked through the provided kernel, and added our own documentation so that we understood what the kernel was doing, which made it easier for us to modify and experiment with.

Link to kernel: <https://www.kaggle.com/kmader/train-simple-xray-cnn>

Research Paper on MobileNet: <https://arxiv.org/abs/1704.04861>

Layer (type)	Output Shape	Param #
mobilenet_1.00_128 (Model)	(None, 4, 4, 1024)	3228288
global_average_pooling2d_3 ((None, 1024)	0
dropout_5 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524800
dropout_6 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 14)	7182
=====		
Total params: 3,760,270		
Trainable params: 3,738,382		
Non-trainable params: 21,888		

Structure of the pre-built CNN

This pre-built CNN uses the ImageNet model MobileNet. To summarize, MobileNet is a Google made model CNN that is prized for its speed, size, and relative accuracy respective to the the two aforementioned parameters. It splits the typical convolution into a depth-wise and point-wise convolution, which ends up being less resource intensive on a system. Using MobileNet enabled us to train this CNN on our laptops without too much worry about of running out of memory.

The pre-built CNN also adds a pooling layer, and then alternates between two drop-out/dense layers to try and prevent overfitting (by dropping out nodes) while mitigating underfitting by adding fully-connected layers in-between each drop-out layer.

Our Simple CNN

```
multi_disease_model = Sequential()
multi_disease_model.add(Conv2D(32, (3,3), input_shape=t_x.shape[1:], activation='relu'))
multi_disease_model.add(MaxPooling2D(pool_size = (2,2)))
multi_disease_model.add(Flatten())
multi_disease_model.add(Dropout(0.5))
multi_disease_model.add(Dense(32))
multi_disease_model.add(Dropout(0.5))
multi_disease_model.add(Dense(len(all_labels), activation = 'sigmoid'))
multi_disease_model.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                             metrics = ['binary_accuracy', 'mae'])
```

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 254, 254, 32)	320
max_pooling2d_1 (MaxPooling2)	(None, 127, 127, 32)	0
flatten_1 (Flatten)	(None, 516128)	0
dropout_1 (Dropout)	(None, 516128)	0
dense_1 (Dense)	(None, 32)	16516128
dropout_2 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 7)	231
=====		
Total params: 16,516,679		
Trainable params: 16,516,679		
Non-trainable params: 0		

We decided to implement a simple CNN after seeing how well a simple model can perform in other image classification problems. Initially, our model consisted of a convolutional layer, max pooling layer, and a dense layer with 32 nodes. Selecting 32 nodes was simply a hardware limitation, because the GPU we used to train our model did not have enough memory for more parameters. The model seemed like it was overfitting because the validation loss was not decreasing after the first epoch, so to combat this, we added two dropout layers that each dropout half of the nodes. Once we added those two layers, our CNN started to return proper results, which are shown in the Results section of this report.

Testing Structure

Hyper-parameters:

- **Image Size** - We experimented with two image resolutions: 128x128 and 256x256. We discovered that increasing the resolution to 256x256 would improve our models' performance with some labels, and worsen performance for others.
- **Dropout Ratio** - We experimented with different dropout ratios for the dropout layers in the CNN we built.
- **Activation Function** - We experimented with using two different activation functions, ReLu and Sigmoid.

Splitting the Data

We sampled 12,000 images out of the 112,000 total images in the dataset, and split it into 75%/25% (9000/3000) for our training and test sets. We used a fraction of the dataset to limit training times, because initially we were not using a GPU to train our models. Stratified sampling was used when separating the data set so that each subset would be representative of the complete data set.

Training Set-Up

We set our models to train for 100 epochs, with early stopping set to monitor validation loss, and to terminate if validation loss does not improve after 10 epochs. We used supervised training based on the labels present for each chest x-ray. We thought at that point, a local minimum was found in the loss function and it would not be useful to continue training. Each of our models trained for about 30-40 epochs before training was stopped. We serialized interesting models and saved them under the *models/* directory. This would allow to further train the models we were interested in if we decided to use a larger portion of the dataset.

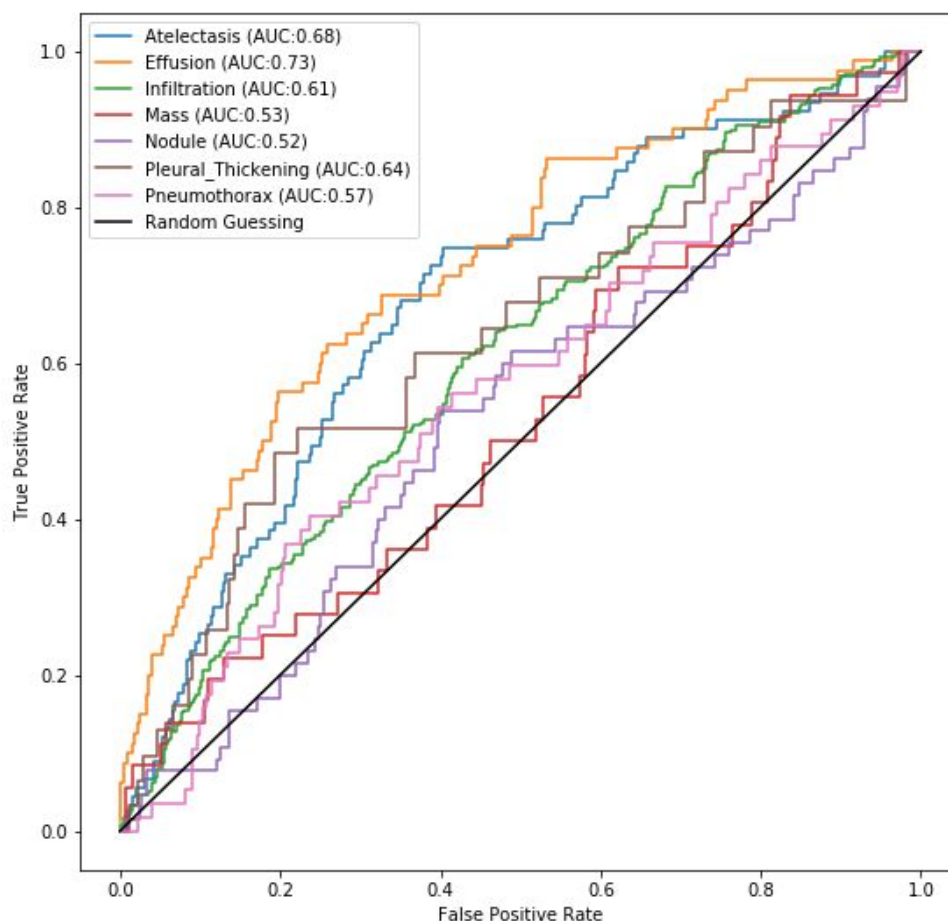
Results

Base Model (Using MobileNet, 128x128 image resolution)

Model trained for 41 epochs. We used the architecture of a pre-trained CNN called “MobileNet” as the base layer for this model.

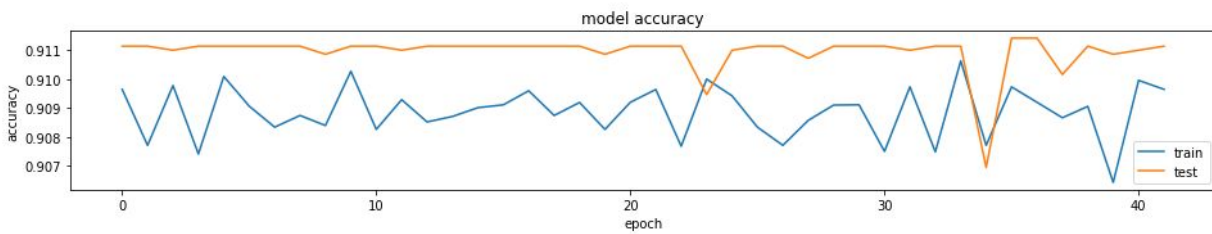
Receiver Operator Characteristic (ROC) Curve

Graphs true positive rate against false positive rate. An “Area Under the Curve (AUC)” score above 0.5 (depicted by the “Random Guessing” line) indicates that the model more often gives a true positive diagnosis compared to a false positive diagnosis.

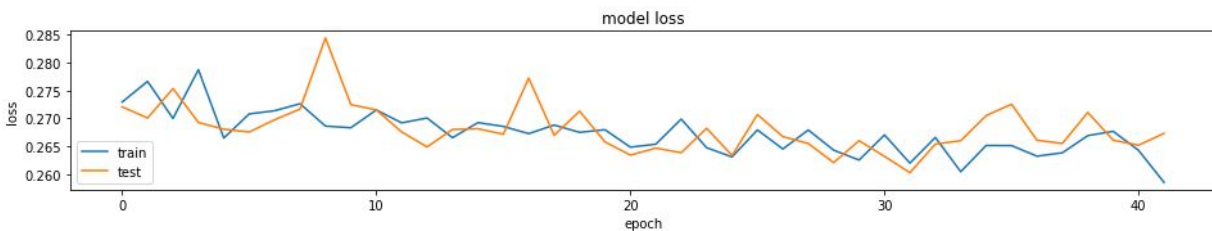


Accuracy Against Epoch

The orange line is accuracy against the validation set, and the blue line is accuracy against the training set.



Loss Against Epoch



Predictions

Dx - Label associated with the image.

PDx - Prediction scores for each label. Predictions seem to be heavily favor an "infiltration" diagnosis because the vast majority of the images in the dataset were labeled with "infiltration".

Dx: Infiltration
PDx: Atelectasis: 5%
, Effusion: 4%
, Infiltration: 20%
, Mass: 3%
, Nodule: 4%
, Pleural: 1%
, Pneumothorax: 3%

Dx: Atelectasis
PDx: Atelectasis: 1%
, Effusion: 1%
, Infiltration: 24%
, Mass: 2%
, Nodule: 3%
, Pleural: 1%
, Pneumothorax: 2%

Dx: Pneumothorax
PDx: Atelectasis: 1%
, Effusion: 0%
, Infiltration: 26%
, Mass: 2%
, Nodule: 2%
, Pleural: 0%
, Pneumothorax: 1%

Dx: Effusion
PDx: Atelectasis: 8%
, Effusion: 15%
, Infiltration: 17%
, Mass: 3%
, Nodule: 2%
, Pleural: 1%
, Pneumothorax: 2%



Dx: Pneumothorax
PDx: Atelectasis: 4%
, Effusion: 2%
, Infiltration: 25%
, Mass: 3%
, Nodule: 3%
, Pleural: 1%
, Pneumothorax: 2%



Dx: Nodule
PDx: Atelectasis: 1%
, Effusion: 0%
, Infiltration: 26%
, Mass: 2%
, Nodule: 2%
, Pleural: 0%
, Pneumothorax: 1%



Dx: Mass
PDx: Atelectasis: 3%
, Effusion: 2%
, Infiltration: 19%
, Mass: 2%
, Nodule: 3%
, Pleural: 1%
, Pneumothorax: 3%



Dx: Infiltration
PDx: Atelectasis: 3%
, Effusion: 2%
, Infiltration: 21%
, Mass: 2%
, Nodule: 3%
, Pleural: 0%
, Pneumothorax: 2%

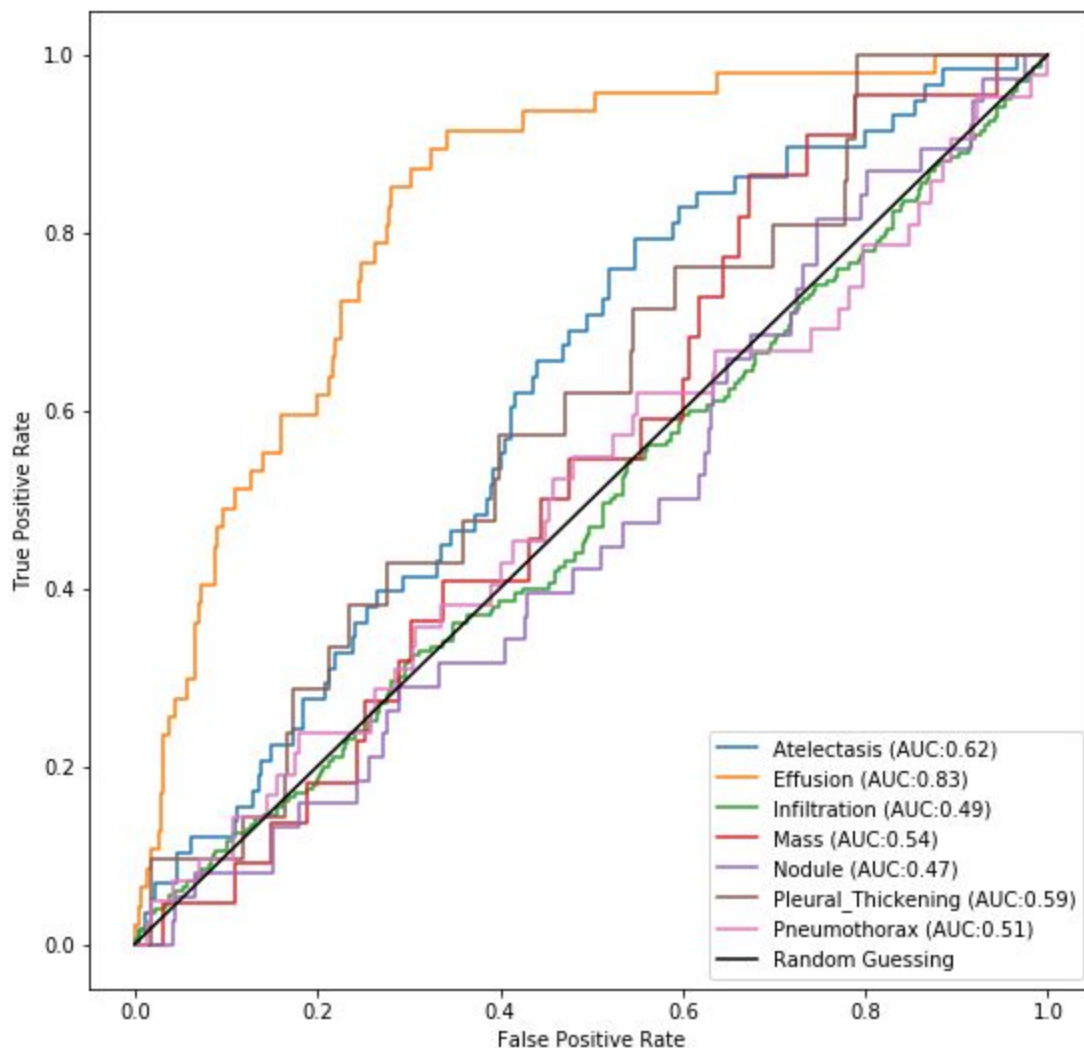


Base Model (Using MobileNet, 256x256 image resolution)

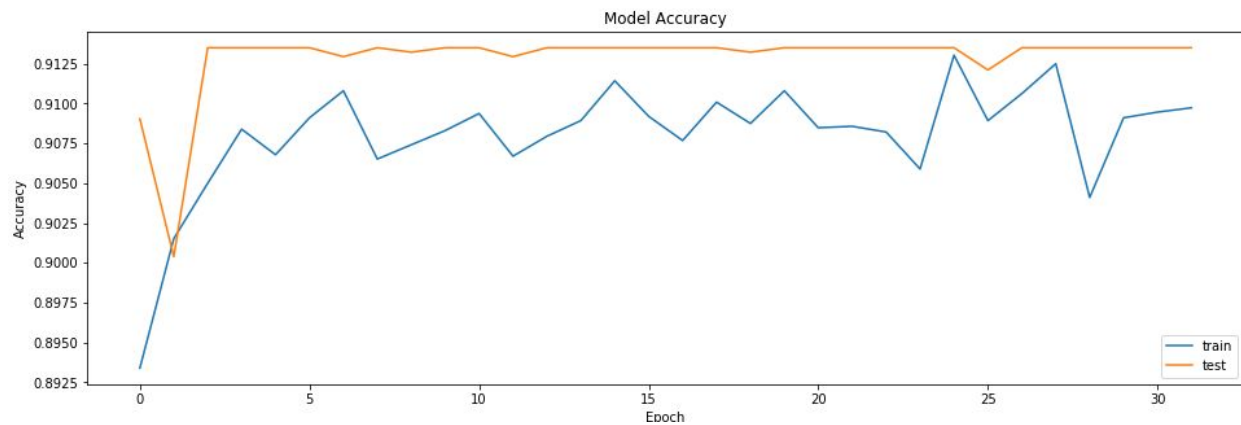
This model trained for 31 epochs before training terminated. Here, we increased the image resolution of our inputs to 256x256.

ROC Curve

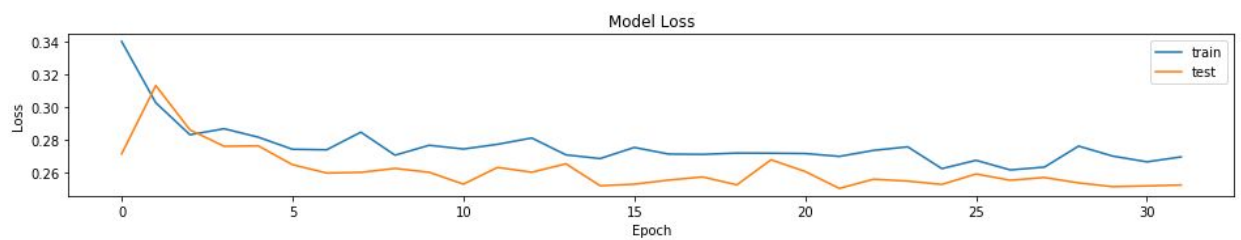
The true positive rate for some diagnoses increased, while the false positive rate for others increased, meaning random guessing would more likely yield a correct result for these diagnoses compared to using this model.






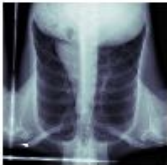




Accuracy Against Epoch



Loss Against Epoch



Predictions

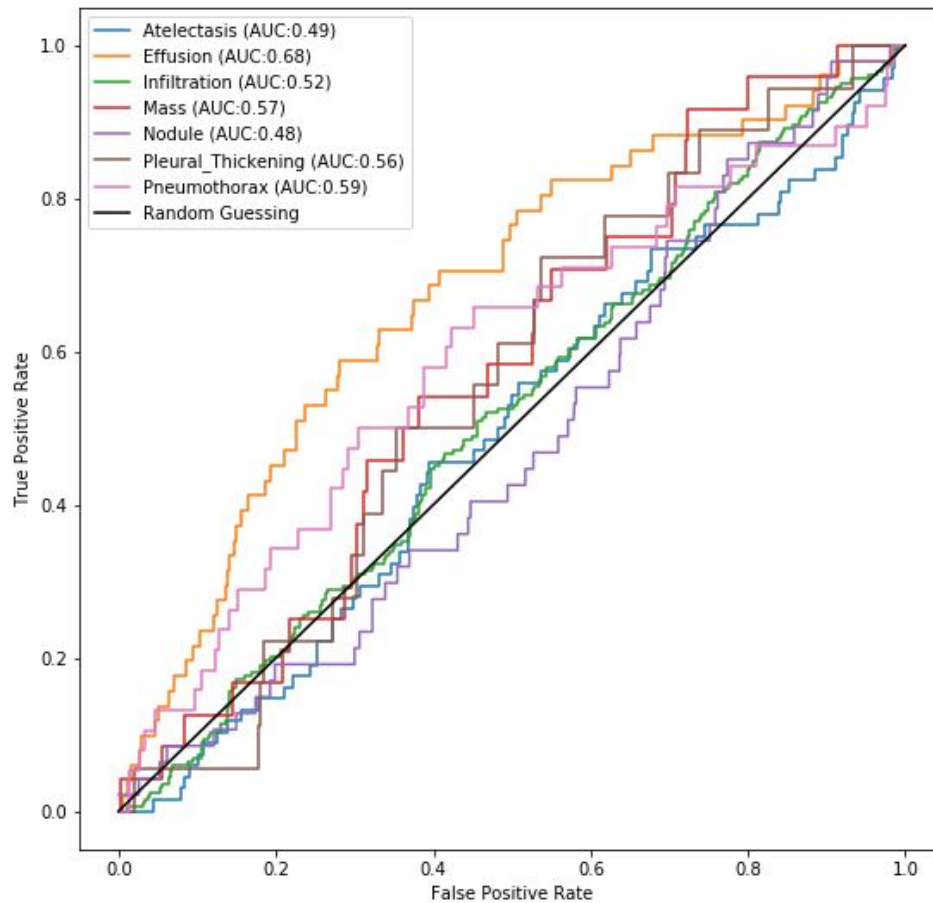
Dx: Atelec PDx: Atel: 5% , Effu: 2% , Infi: 32% , Mass: 2% , Nodu: 8% , Pleu: 1% , Pneu: 4%	Dx: Infil PDx: Atel: 9% , Effu: 6% , Infi: 31% , Mass: 3% , Nodu: 7% , Pleu: 2% , Pneu: 5%	Dx: Nodule PDx: Atel: 10% , Effu: 10% , Infi: 26% , Mass: 3% , Nodu: 5% , Pleu: 2% , Pneu: 7%	Dx: Infil PDx: Atel: 5% , Effu: 2% , Infi: 33% , Mass: 2% , Nodu: 8% , Pleu: 1% , Pneu: 4%
			
Dx: Effusi PDx: Atel: 9% , Effu: 8% , Infi: 33% , Mass: 3% , Nodu: 7% , Pleu: 2% , Pneu: 6%	Dx: Infil PDx: Atel: 5% , Effu: 2% , Infi: 32% , Mass: 3% , Nodu: 8% , Pleu: 1% , Pneu: 4%	Dx: Infil PDx: Atel: 8% , Effu: 6% , Infi: 33% , Mass: 3% , Nodu: 8% , Pleu: 2% , Pneu: 5%	Dx: Pleura PDx: Atel: 7% , Effu: 3% , Infi: 27% , Mass: 2% , Nodu: 6% , Pleu: 2% , Pneu: 4%
			

Simple CNN (256x256 image resolution)

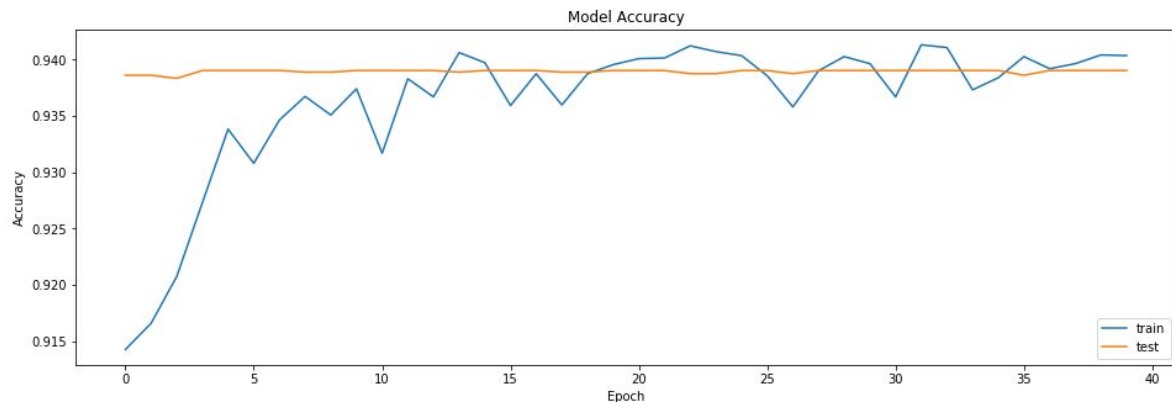
Model trained for 31 epochs before training terminated.

ROC Curve

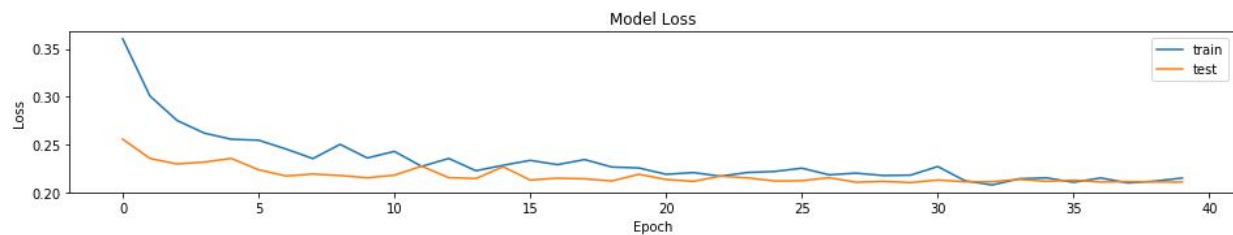
ROC shows similar patterns to the base model using 256x256 image resolution, with the AUC for a few diagnoses dropping below 0.5.



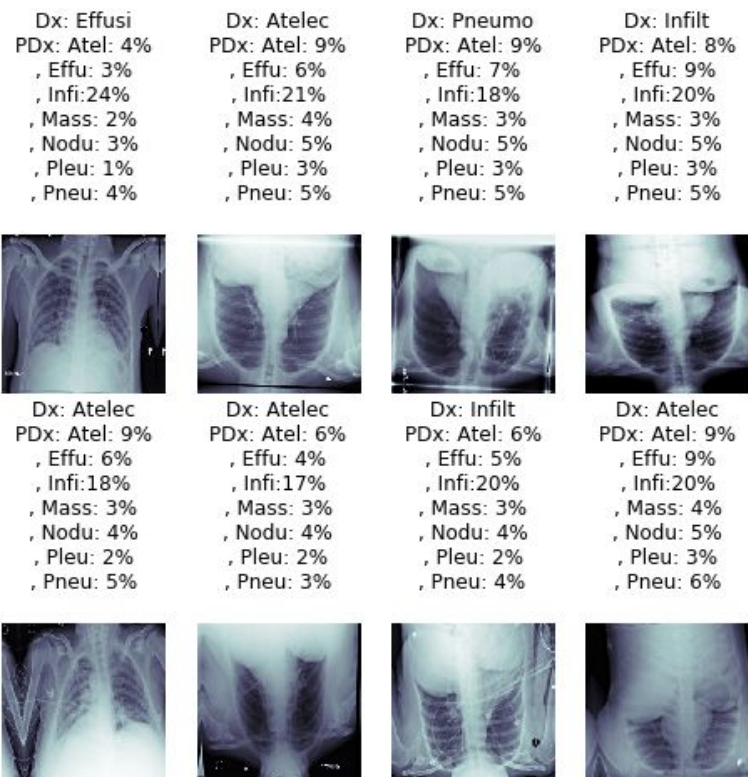
Accuracy Against Epoch



Loss Against Epoch



Predictions



Evaluation

Results

Some patterns we noticed with each model:

- In the Receiver Operator Characteristic (ROC) curve, it appears that the label “Effusion” pulls away from the “Random Guessing” line the most.
 - All the models we trained classified this category the best.
- Validation accuracy reached about 0.9 in each model, which is much higher than we expected. We believe that it is because the label “Infiltration” made up a significant portion of the dataset, so if the model predicted “Infiltration” it would be correct more often than not.
- We believe the majority of our dataset being labeled “Infiltration” also has to do with the prediction scores when testing our model heavily favoring “Infiltration.”
- The poor prediction rate on “Nodule” and “Mass” classifications mimics the research paper about the ChestX-ray8 system (which also performed poorly for these classifications with much more robust CNNs).
 - In the paper, they explain the difficulty with classifying these diseases because of large in-class variation, and because the disease areas associated with these classifications is relatively small.

We failed to hit our target accuracy for all classifications of 75%, but after comparing our results with the ChestX-ray8 results, we feel we did a decent job given that we used less accurate CNN models.

Improvements

- Balance the number of records for each classification. The majority of our dataset being labeled “Infiltration” affected our model’s training. If we had an equal amount of each label, we may have gotten more accurate results for our validation accuracy, and in our predictions.
- Use more of the dataset. If we used a bigger portion, or even all of the available dataset, we would have more images to train some of the less frequent labels.
- Experiment with other pre-trained CNNs such as VGGNet-16, using cloud computing.