**Data Science Project Protocol**

*Authors:*

*Michal Gimmon*

*Liron Walter*

Contents

[Introduction 2](#_Toc2959535)

[Methodology 3](#_Toc2959536)

[Data 3](#_Toc2959537)

[**Data Sources** 3](#_Toc2959538)

[**Images Analysis & Preprocessing** 9](#_Toc2959539)

[Models 12](#_Toc2959540)

[Results 20](#_Toc2959542)

[Conclusion 25](#_Toc2959543)

# Introduction

X-ray imaging is one of the most widely used diagnostic examinations for the diagnosis and research of numerous types of illnesses, making it a high-potential source of data for the development of computer-aided detection and diagnosis tools.

Previous work has been done in the past in the field of X-ray images classification, (for example, the identification of Tuberculosis) using various deep learning techniques, yet it is still a challenge to be explored, as the available data is limited to only thousands of images for training, and poor image-labeling techniques which impact classification accuracy.

The dataset at hand is comprised of 112,120 chest X-ray images of 30,805 unique patients, which were classified into 14 different types of lung diseases (including images which demonstrate multiple diseases), and images of healthy individuals.

Several groups have addressed this multi-label dataset in attempt to classify the images according to different research goals - some classified all 14 classes (diseases) [1], [2] or part of the diseases [3], some identified patients with one specific disease vs healthy patients, and some looked for the correlation between two (or more) diseases.

Due to the complexity of multi-label classification in this dataset, we have initially decided to simplify the problem by focusing on classifying the images to patients who demonstrate ‘Effusion’ (fluid in the space around the lung), other diseases or healthy (3 classes), using deep learning methods such as different models of CNN.

The main reason for choosing ‘Effusion’ is the high frequency of the disease in the available dataset, and moreover, according to radiologist [4], this phenomenon labeling seem to be more accurate than other diseases.

In later stages, as we encountered some computing resources limitations, we have decided to work on a smaller sample of the data - a random sample of 5,606 images, as our GPU resources were limited. Further in the process, after fully analyzing the new data, we also decided to simplify the classification to two categories instead of three – ‘Healthy’ and ‘Effusion’.

As mentioned above, the outcome of this classification challenge may be affected by the fact that the disease labels in this dataset were retrieved using text mining (NLP) tools from radiologists’ reports, which may lead to seemingly false identifications and impact the model’s accuracy. As mentioned, the task is also challenging in matters of computational resources and memory, though is extremely interesting. Our goal is to find a model that predicts Effusion cases better than a random probability.

# Methodology (Project design)

## Data

### **Data Sources**

The dataset at hand, as described above, was collected, labeled and published in 2017 by the *NIH Clinical Center* – a clinical research hospital operated by the United States’ National Institute of Health.

The images and its corresponding data were retrieved from the NIH publicly available website - <https://nihcc.app.box.com/v/ChestXray-NIHCC>.

**Defining the Outcome Variable**

As mentioned before,due to the complexity of multi-label classification in this dataset, we have initially decided to simplify the problem and focus on classifying the images between one type of disease, normal or other diseases.

In order to define the disease on which we want to focus, we had to understand the data in terms of the frequency of each pathology, as described below:

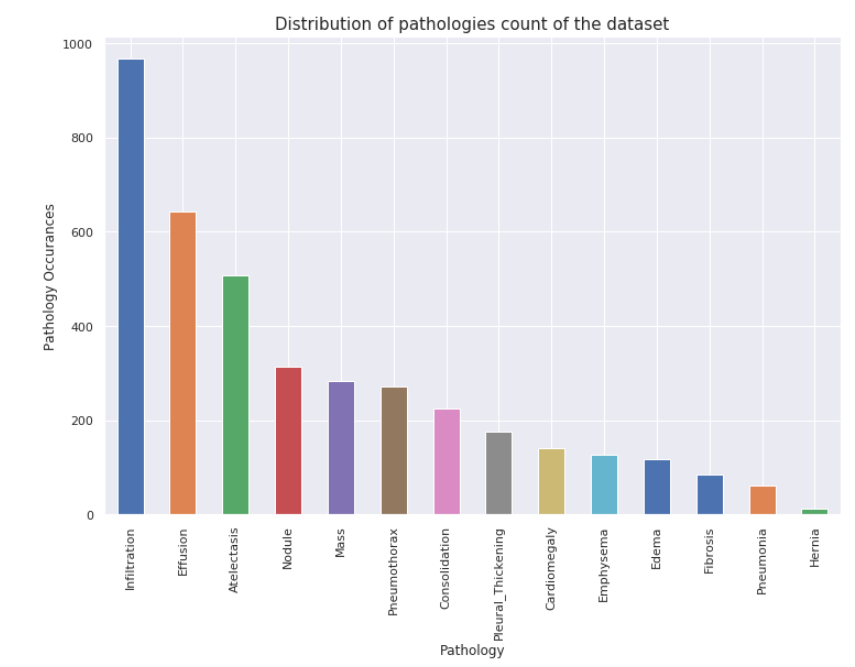


Figure 1. Distribution of pathologies in the sample dataset (5,606 images)

Based on the above, we can see that 'Effusion' is one of the most common observations in this dataset, meaning there is a higher chance for the model to classify it correctly, as it will have more images to be trained on. In addition, when defining our outcome, we had to consider the relative accuracy of the disease labeling by NLP, which may significantly affect our result.

Moreover, based on previous work done on this dataset [4], we learned that ‘Effusion’ was labeled with relatively high accuracy, therefore we have decided to define the classification to the 3 following categories: **Effusion (1), Other disease (2), Healthy (3),** by creating a new parameter in the data frame called ‘3class\_label’.

**Exploring the data for correlative / confounding variables**

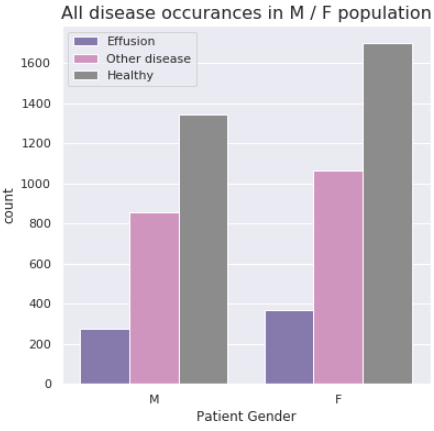
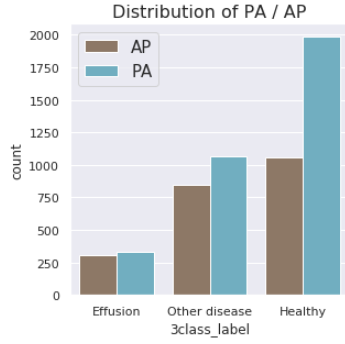
After classifying the images between the three categories we initially defined, we checked for the distribution of the images:

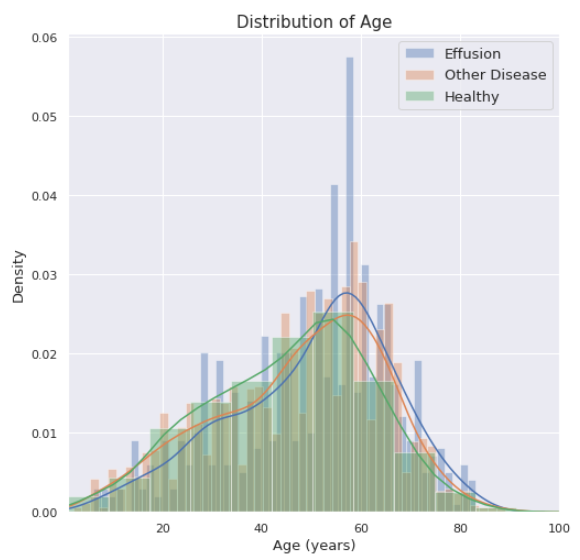


Figure 2. Distribution of the 3 classifications (‘3class\_lable’) in the new dataset

The groups are comprised of 644 images of 'Effusion', 1,918 Images of 'Other diseases' and 3,044 'Healthy' images.

In order to decide if / which variables should be included in our model, and in order to avoid creating biased sub-groups for training and validation, we checked for a relation between the parameters ‘Patient Gender’, ‘View Position’, ‘Patient Age’, ‘Follow-Up #’, and the detection of ‘Effusion’:





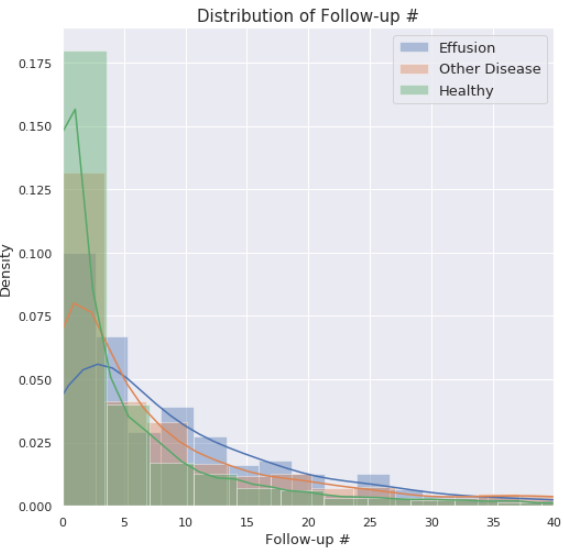


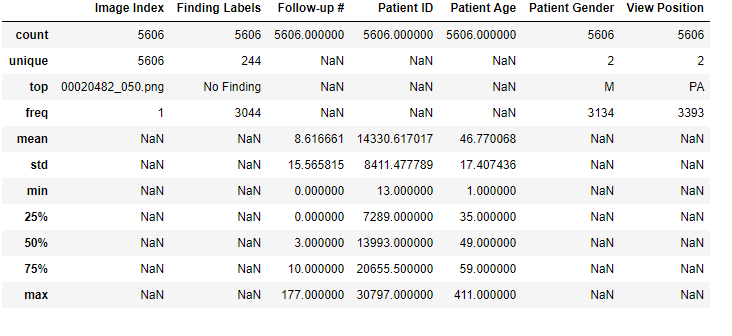
Figure 3. Distribution of data parameters in the three explored groups ‘Effusion’, ‘Other disease’ and ‘Heathy’

As we can see, the variables ‘View Position’ and ‘Patient Gender’ are distributed almost equally in the outcome population (‘Effusion’).

Moreover, we can see that ‘Patient Age’ is distributed in a relatively normal matter between ‘Effusion’ patients, which means a random selection of images for the training / validation sets will not affect the model’s accuracy.

**Treatment of Outliers and Missing Values**

Table 1. Description of data variable - count, unique, mean and IQR values



As represented in the table above, there are no missing values for all variables (both categorical and numerical).

It can also be observed that there are outliers of patient age, since 414 years is the maximal value, and that the follow-up median # is 3, while the mean is 8.5, meaning the distribution is not normal.

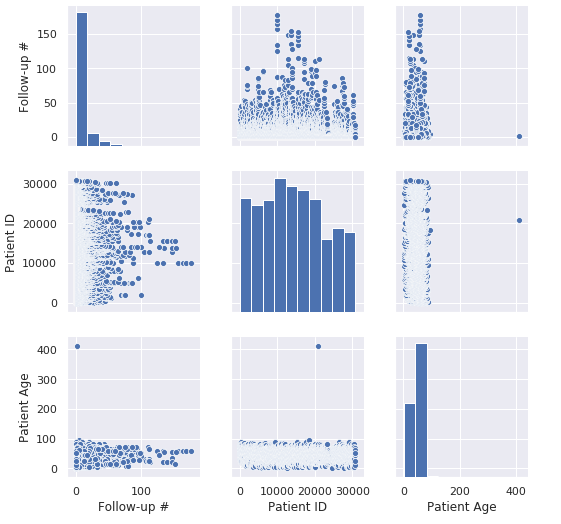


Figure 3. Pairplot visualization of the inter-parameter relations

According to *Figure 3*, there is an image with patient age over 400, and another 10 images having age above 100. Moreover, there is no significant correlation between the follow up # and the patient age.

Yet, since we currently are interested in classifying only the images data, we decided not to remove the outliers of the 'age' variables, since we do not mind that they were mistakenly documented. If we would have decided to include the ‘age’ variable in the model, we would have removed these outliers.

### **Images Analysis & Preprocessing**

* **Pixels Histograms of representative images**

In order to better understand the images data, we looked at a random sample of images from each class, and also at the differences in pixels distribution between them:

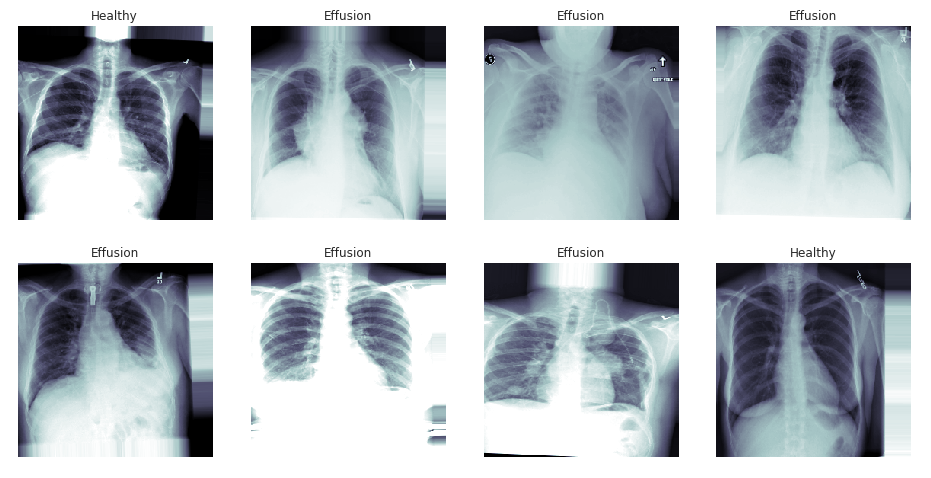
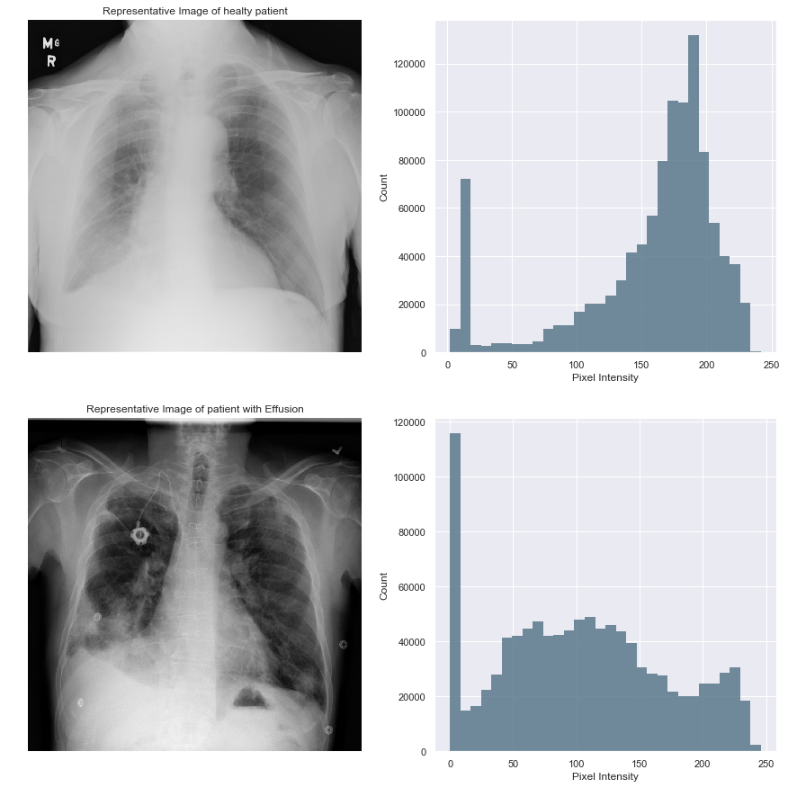


Figure 4. Random images of ‘Healthy’ and ‘Effusion’

Figure 5. Representative pixels distributions of ‘Healthy’ versus ‘Effusion’



## 

Figure 5. Pixels distribution of representative images from both classes

* **Images Preprocessing**

From exploring the representative samples, we find that the variability in images’ pixels intensity is extremely high - many of the images are highly saturated / dark, which means they may act as outliers. Moreover, from general observation its clear that the images characteristics are very random – parameters like zoom, rotation, noise, scaling and etc. are not standardized.

In order to preprocess the images to best overcome those obstacles, we used Keras’s *ImageDataGenerator* class that defines the configuration for image data preparation and augmentation.

Here are the detailed parameters which were set to the images using *ImageDataGenerator:*

height\_shift\_range=0.1, *# handle off-center objects by creating shifted versions of the train data*

width\_shift\_range=0.1,

brightness\_range= [0.7, 1.5] *# Range for picking a brightness shift value from*.

rotation\_range=3, *# Degree range for random rotations*

shear\_range=0.01 *# Shear Intensity*

fill\_mode='nearest', *# default. Type of padding outside the input boundaries*

zoom\_range=0.125, *# Standardzing zoom*

rescale=1/255.0, *# Rescaling pixels to be between 0-1.*

We have also used the *flow\_from\_dataframe* function which is part of the *ImageDataGenerator* class in order to pull random image batches for each training cycle, when calling *flow\_from\_dataframe* we also resized the images to 256x256 (originally 1024x1024).

# Models

* **Data Division**

As can be seen in *Figure 2*, the population in the data set is not equally distributed – only 644 out of 5,065 images present 'Effusion' (~12%), whereas 1,918 Images are of 'Other diseases' (~38%) and a majority of 3,044 are 'Healthy' images (~50%).

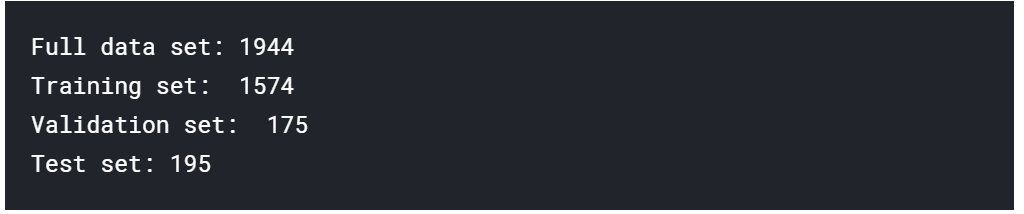
Considering the above, we have decided to further simplify the prediction, and turn it into a binary classification - where 'Effusion' occurrences are ‘1’ and 'healthy' are ‘0’.

To do so, and in order to balance the distribution, we have created a new data set, in which we included all the 'effusion' images we have (644), and twice the number of 'healthy' images (1,300), which were selected randomly. So that the final dataset now contains a total of 1,944 images.

To split the data into training, validation and test, we used the *train\_test\_split* function from ‘sklearn’ which randomly splits the data according to a predefined group size – we chose to split 90% to the training group, and 10% to the test group.

Then we split 10% from the training group to the validation group.

So, eventually we received the following sets:



After the division, we checked for the distribution within the groups:



Figure 6. Distribution of classes in training, validation and test groups

As can be seen in *Figure 6*, the proportions of ‘Effusion’ and ‘Healthy’ are similar in all groups. However, in order to prevent a bias in the training results, we further worked on the training data to create a balanced group, by randomly reducing the number of ‘Healthy’ images to match the number of ‘Effusion’ images. This way we have equally balanced training group of 1,050 samples.

(Note: In this case, we decided to reduce the majority group ('Healthy') to match the size of the minority group ('Effusion'), and not the other way around by using data augmentation techniques, due to limited computational resources).

* **Model Deployment**

We chose to use a pretrained model as the base model (benchmark) in order to maximize accuracy. Keras offers many pretrained models which were previously trained on 'ImageNet', with different depths and performance rates.

Looking at the figure below, we picked the VGG16 model for initial training considering its high accuracy and relatively low computational cost.

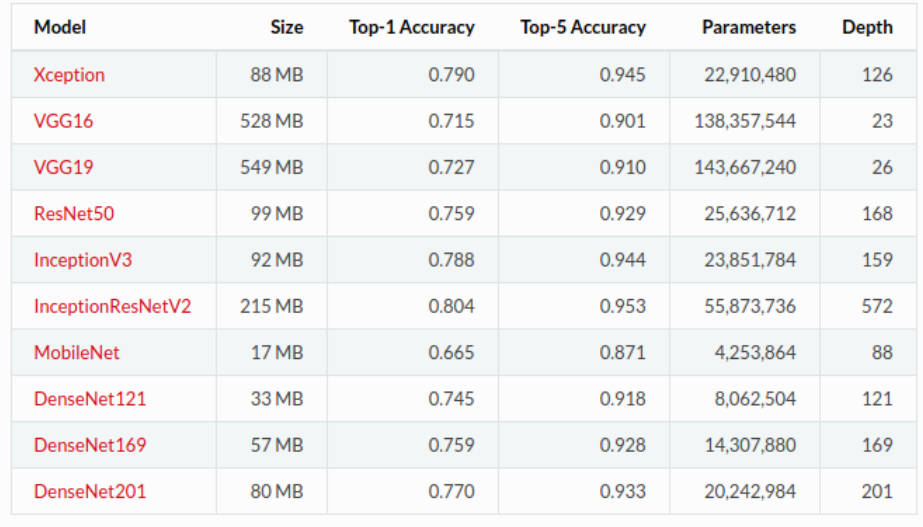


Figure 7. Keras pretrained models and their performance

The main contribution of VGG is to show that classification accuracy can be improved by increasing the depth of CNN despite using small receptive fields in the layers (especially earlier layers). Neural networks prior to VGG used bigger receptive fields (7\*7 and 11\*11) as compared to 3\*3 in VGG, but they were not as deep as VGG. There are few variants of VGG, while the deepest one is with 19 weight layers.

VGG network mainly includes 3\*3 convolutional layers stacked on top of each other in increasing depth, max pooling layer to reduce volume size and fully-connected layers that are followed by a soft max classifier. A detailed explanation of the network is available at [11].

VGG16 net consists of 16 convolutional layers and is very appealing because of its uniform architecture. Therefore, it is currently one of the most preferred choices for extracting features from images, due its simplicity. However, VGG net consists of around 140 million parameters, which may be a bit challenging to handle. The figure below represents a visualization of VGG architecture.

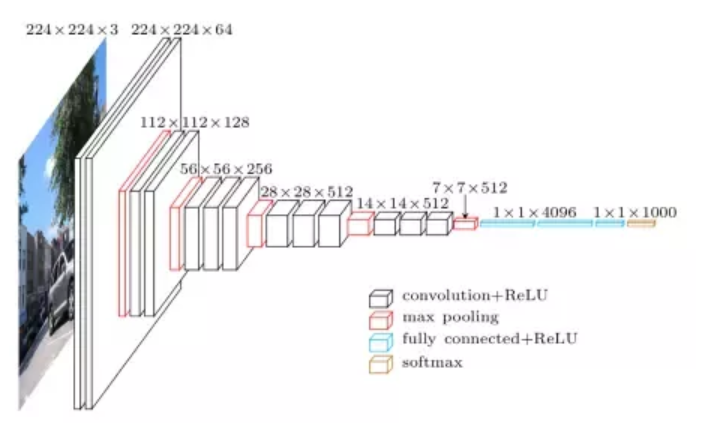


Figure 8. Visualization of VGG16 architecture

Fine-tuning the model, it is common to freeze the weights of few initial layers of the pre-trained network. This is because the first few layers capture universal features like curves and edges that are also relevant to most of the other problems. We want to keep those weights intact. Instead, we will get the network to focus on learning dataset-specific features in the subsequent layers. In our model we decided to freeze the layers of the VGG16 network, so the model will focus on features of future layers that we added in order to classify the effusion pathology. When we did not freeze these layers, however, the training process was less sufficient (not shown in the following results).

1st model

At the first attempt *(1st model*), we used the VGG16 base model for its ground weights, and added the following layers on top:

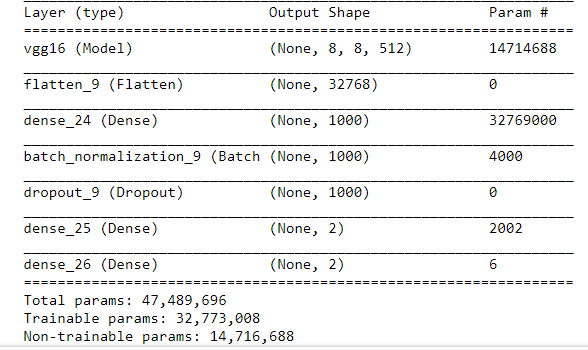


Figure 11. 1st Model Layers

The *Flatten* and *Dense* layers are added at the end of our model in order to achieve the output in the size we need it.

The *Flatten* layer flattens the pooled feature map into a column (usually after a pooling layer) and the *Dense* layer is used to change the dimensions of that vector. We apply both in order to insert the data into an artificial neural network so that we can continue process the data and finally achieve our predictions.

*Batch normalization* layer was initially introduced in 2005 [5] and is usually added right before calling the activation function. As the data flows through a deep network, the weights and parameters adjust it, potentially making the data too big or too small again - a problem the authors refer to as “internal covariate shift”.

*Batch Normalization* normalizes each batch by both mean and variance, so the problem is largely avoided. The layer’s benefits are allowing using higher learning rates and therefore making the network converge faster as well as giving overall better results.

A *Dropout* layer is a technique used to tackle overfitting. Using “dropout”, randomly certain units (neurons) in a layer are deactivated with a certain probability (a hyperparameter that may be tuned). So, if you set half of the activations of a layer to zero, the neural network won’t be able to rely on particular activations in a given feed-forward pass during training. As a consequence, the neural network will learn different, redundant representations.

Using the *Softmax* layer as the final layer of our neural network in order to provide a categorical output.

For this model we used the following measures to configure the learning process:

***loss***= 'binary crossentropy'

***optimizer***= Adagrad (lr=0.01)

***metrics*** = 'binary\_accuracy'.

And defined validation loss *‘****Val loss****’* as the measure for learning progression and model accuracy.

After running the model, we noticed that ‘val loss’ was improving during the first 8/30 epochs, but at that point it got stuck at 0.5220 (models’ results are further described in the ‘Results’ section below).

Adagrad allows the learning rate to adapt based on parameters. It performs larger updates for infrequent parameters and smaller updates for frequent one, but it has the problem of monotonically decreasing learning rate.

The Adam optimizer is an adaptive method compared to the gradient descent which maintains a single learning rate for all weight updates and the learning rate does not change).

At that point, we tried to run the same model again, with a different optimizer and lower learning rate: Adam (lr=0.001). However, results did not improve (min val loss was 0.54482 at 6/30 epochs), and the learning curve seemed to be flat (see Jupyter notebook #1).

2nd model

Then we reconstructed the model architecture (‘*2nd model’*), by neglecting one of the Dense layers and changing the activation function of the first Dense layer from ReLu to Sigmoid:

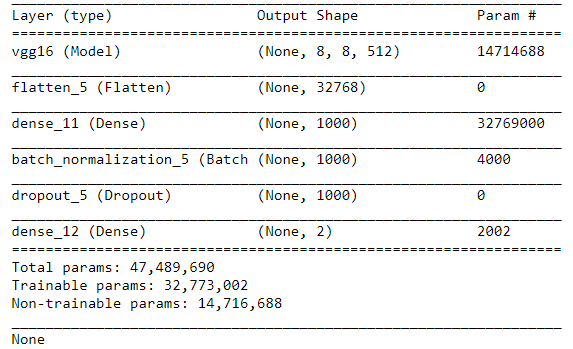


Figure 12. 2nd Model Layers

This architecture showed better loss results than our first model (with min val loss of 0.51502 at 5/30 epochs).

3rd model

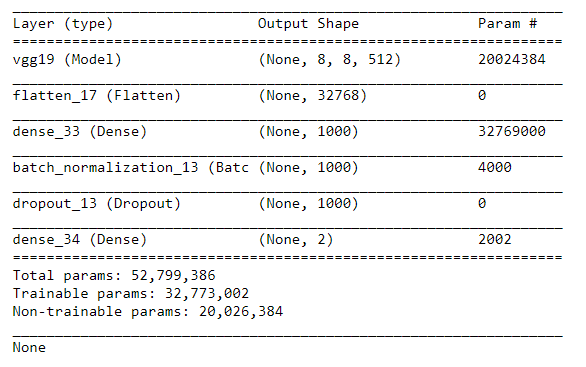
Yet, we have tried also a ‘*3rd model’*, in which we tested the impact of using a different pretrained network as benchmark – VGG19 (which is deeper and showed better performance on ‘ImageNet’): 

Figure 13. 3rd Model Layers

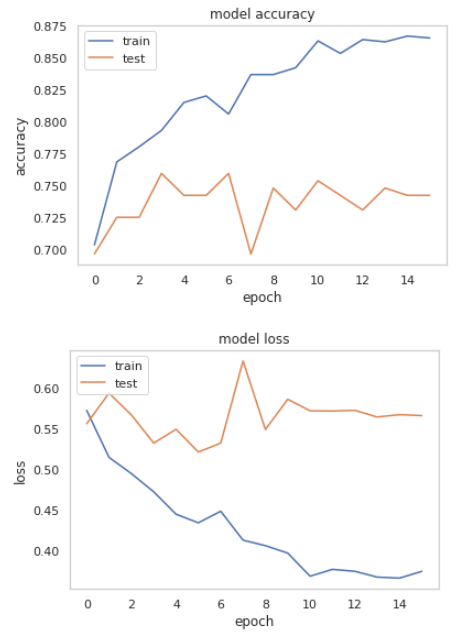
This model reached the lowest min val loss of 0.47618, see detailed accuracy results below.

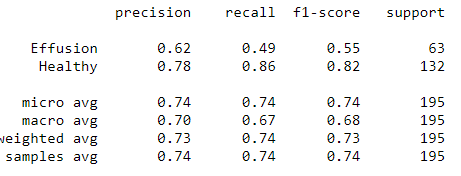
# **Results**

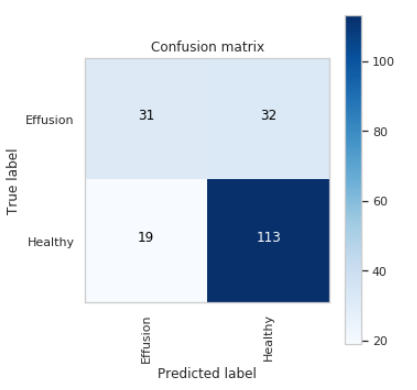
***1st model***

*Figure14* demonstrates the change of training and validation loss and accuracy during training. As epochs progress, we see an evident increase of the training accuracy and decrease in loss (Minimum Validation loss for this run was 0.5220), though for the validation group, both curves don’t show significant trends.

*Figure14* shows a true positive detection of 31/63 ‘Effusion’ images (0.49 recall), and 113/132 ‘Healthy’ images (0.86 recall), with total precision of 0.70.

**





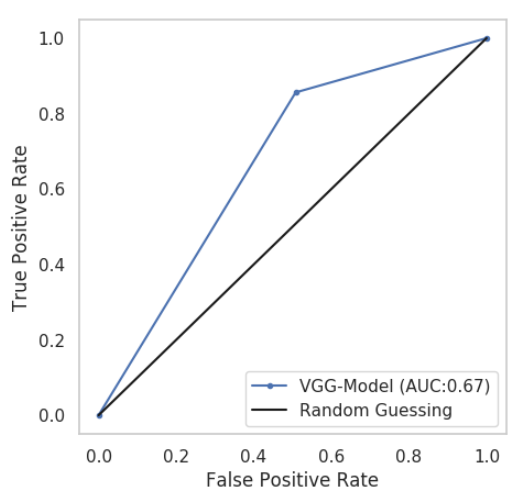


Figure 14. 1st model - Visualization of the training process

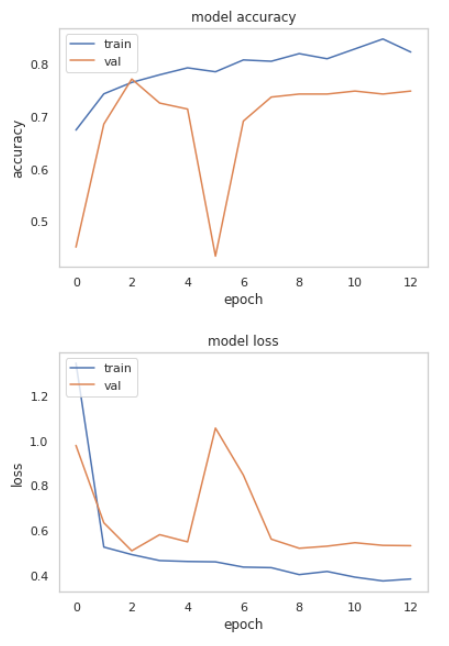
Figure 15. 1st model - Accuracy

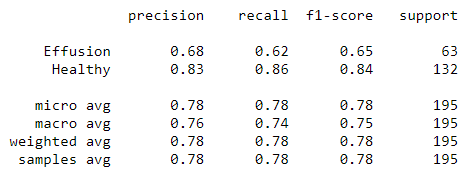
***2nd model -*** 1st training cycle:

Here we reconstructed the model architecture by neglecting the last Dense layer and changing the activation function of the first Dense layer from ReLu to Sigmoid.

*Figure16*. demonstrates increase in accuracy and decrease in loss (Minimum Validation loss for this run was 0.51052), for both the training and validation groups.

*Figure17.* shows a true positive detection of 39/63 ‘Effusion’ images (0.62 recall), and 114/132 ‘Healthy’ images (0.86 recall), with total precision of 0.76.





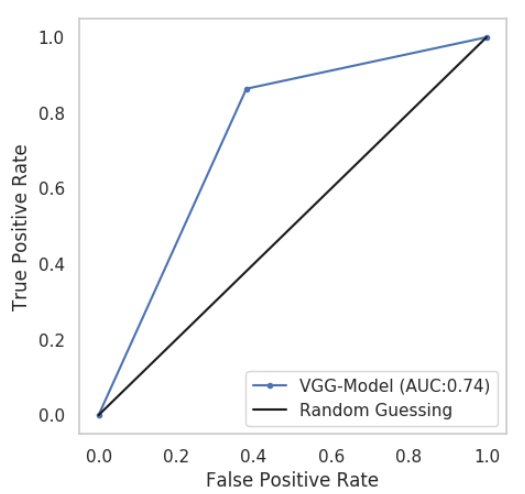
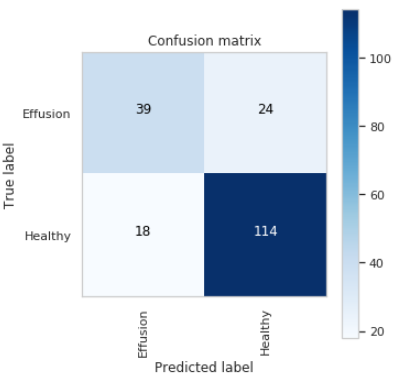


Figure 16. 2nd model, 1st run – Visualization of training

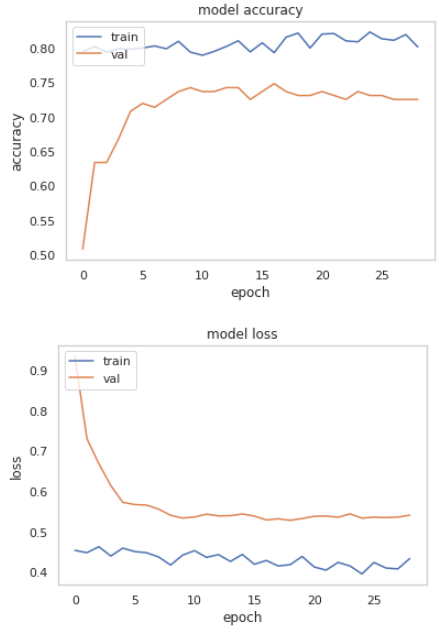
Figure 17. 2nd model, 1st run - Accuracy

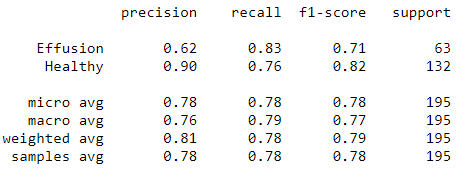
***2nd model –*** 2nd training cycle:

These are the results of training the same model for the 2nd time, with the weights from previous cycle (fitting).

In *Figure18* we see that as a result of using pretrained weights the learning curve for the training group doesn’t show significant change (Minimum val loss in this run remained 0.51052), yet the validation group shows significant increase in accuracy and loss decrease.

*Figure19.* shows improvement in true positive detection of 52/63 ‘Effusion’ images (0.83 recall), and 100/132 ‘Healthy’ images (0.76 recall), with total precision of 0.76.





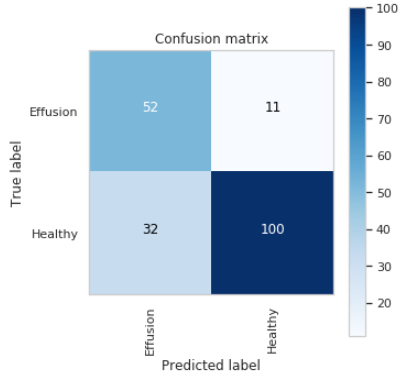
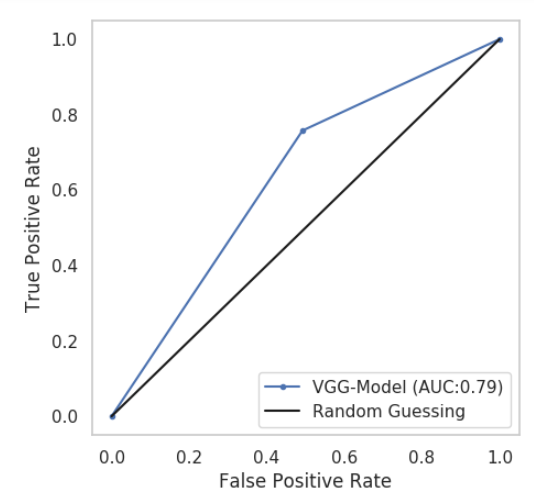


Figure 18. 2nd model, 2nd run – Visualization of training

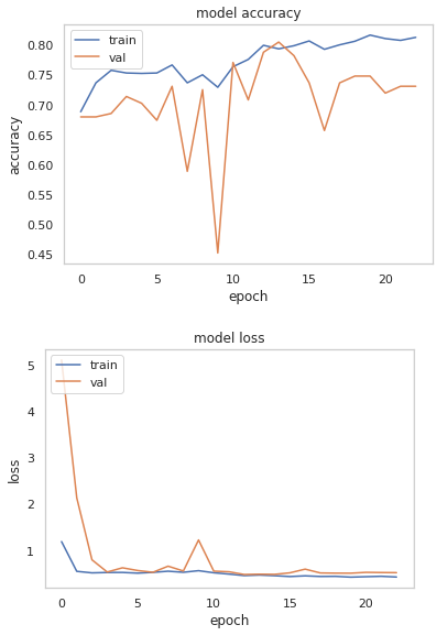
Figure 19. 2nd model, 2nd run - Accuracy

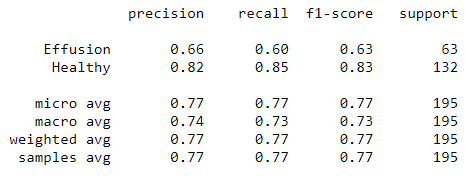
***3nd model –*** 1st training cycle:

In this model we used VGG19 as base model and added the same layers as in the model above.

In *Figure20* we see that accuracy didn’t change significantly in both training and validation groups, and loss decreased dramatically during first epochs, but then stabilized (Minimum validation loss in this run was 0.48297).

*Figure21.* shows a true positive detection of 38/63 ‘Effusion’ images (0.60 recall), and 112/132 ‘Healthy’ images (0.85 recall), with total precision of 0.74.





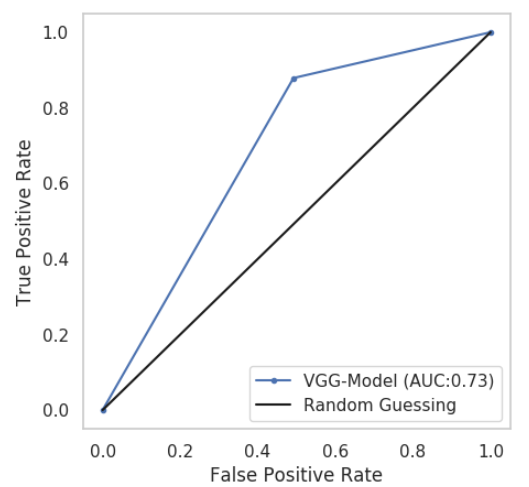
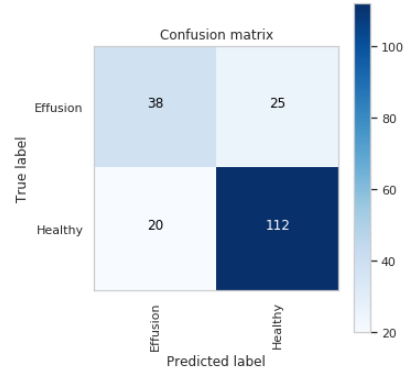


Figure 20. 3rd model, 1st run – Visualization of training

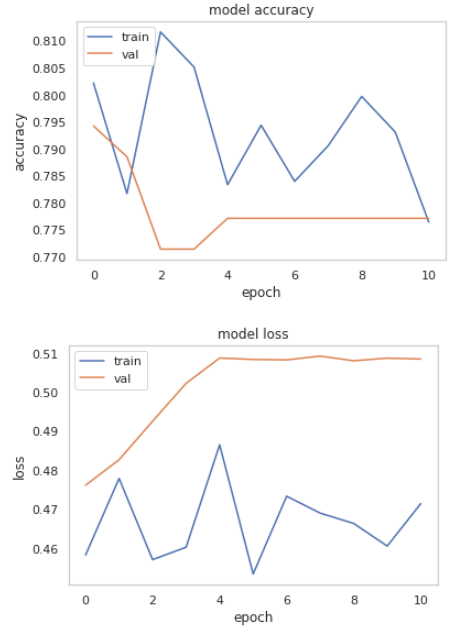
Figure 21. 3rd model, 1st run – Accuracy

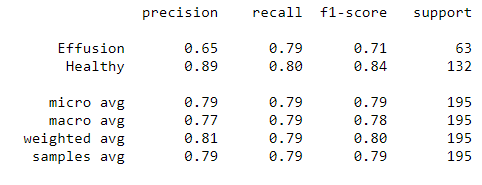
***3nd model –*** 2nd training cycle:

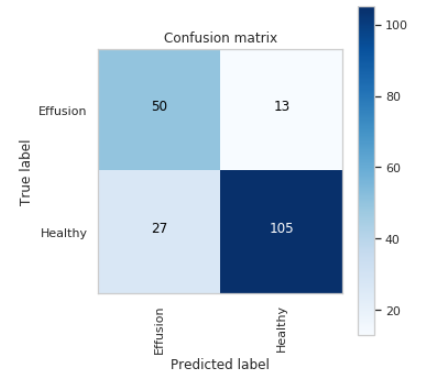
These are the results of training the same model for the 2nd time, with the weights from previous cycle (fitting).

In *Figure22* we see a decrease in the validation accuracy and increase of its loss, however these results are the opposite as we expect, for unknown reason. In the 4th epoch both evaluation methods achieve a plateau of validation accuracy of 0.777. Minimum validation loss has improved during this run from 0.48297 to 0.47618.

*Figure23.* shows a true positive detection of 50/63 ‘Effusion’ images (0.79 recall), and 105/132 ‘Healthy’ images (0.80 recall), with total precision of 0.77.







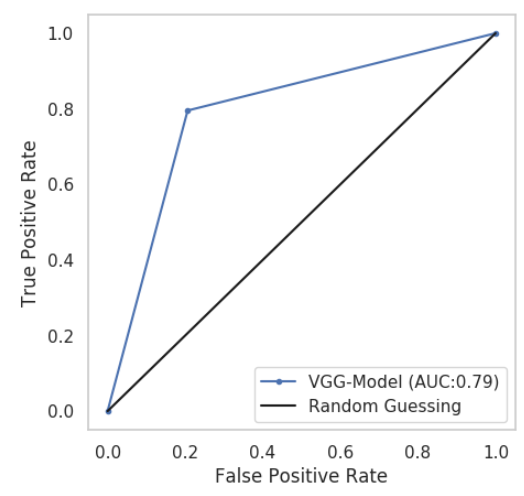


Figure 23. 3rd model, 2nd run - Accuracy

Figure 22. 3rd model, 2nd run – Visualization of training

# **Conclusion**

X-ray imaging is one of the most widely used diagnostic examinations for the diagnosis and research of numerous types of illnesses, making it a high-potential source of data for the development of computer-aided detection and diagnosis tools. This was our main motivation to investigate such a database and try to develop a model with an input of an x-ray image and output whether some pathology exists.

The full online free dataset includes 112,120 chest X-ray images of more than 30,805 unique patients, which were classified into 14 different types of lung diseases (including images which demonstrate multiple diseases), and images of healthy individuals.

The main limitation of the data is the poor image-labeling techniques (using natural language processing models to automatically label radiologists’ handwriting in clinical reports). Moreover, exploring the images we found that the variability in images’ pixels’ intensity is extremely high and that many parameters such as zoom, rotation, scaling and noise are not standardized. Obviously, these data limitations have impact on the classification task and accuracy results.

Due to the complexity of multi-label classification in this dataset and as we encountered computing resources limitations, we have finally decided to focus on a smaller sample of data - a random sample of 5,606 images – and classify two categories, the ‘Effusion’ pathology and ‘Healthy’. The main reason for choosing ‘Effusion’ is the high frequency of the phenomenon in the available dataset, and moreover, according to radiologist [4], this phenomenon labeling seem to be more accurate than other diseases.

Developing a neural network and training it with such a small number of images is an extremely challenging task, though once we build a model that seem to behave as expected and give enough accuracy, we may extend it to the larger data set or to much complicated problems.

The main goal of this project was to investigate the data set, learn how to implement neural-networks methods and find a deep-learning model that predicts Effusion cases better than a random probability.

During our research, we implemented few different models and investigated the results. The following table summarizes the results achieved (the results are discussed in details in the Results chapter).

Table 2. Accuracy evaluation methods of the different architectures

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Base Model** | **Added layers** | **Optimizer/ Learning rate** | **Learning cycle** | **AUC** | **Min Val loss** |  | **Precision** | **Recall** |
| VGG16 | Flatten, Dense (Relu), Batch-normalization, Dropout, Dense, Dense | Adagrad | 1 | 0.67 | 0.522 | Effusion | 0.62 | 0.49 |
| Healthy | 0.78 | 0.86 |
| VGG16 | Flatten, Dense (Sigmoid), Batch-normalization, Dropout, Dense | Adam | 1 | 0.74 | 0.51 | Effusion | 0.68 | 0.62 |
| Healthy | 0.83 | 0.86 |
| 2 | 0.79 | 0.51 | Effusion | 0.62 | 0.83 |
| Healthy | 0.90 | 0.76 |
| VGG19 | Flatten, Dense (Sigmoid), Batch-normalization, Dropout, Dense | Adam (0.001) | 1 | 0.73 | 0.48 | Effusion | 0.66 | 0.60 |
| Healthy | 0.82 | 0.85 |
| 2 | 0.79 | 0.476 | Effusion | 0.65 | 0.79 |
| Healthy | 0.89 | 0.80 |

As seen in the summarizing table (while best results of each evaluation method is marked in green), every model has its advantage. However, looking at the results, we understand that using VGG19 as the base model and training it second time (using the weights of the first training process), gives the best overall accuracy results out of the different models, predicting our labels with accuracy of 77.7% (see Jupyter notebooks for details). This model, however, predicts 77/195 test samples as ‘Effusion’, though only 50 are truly labeled as ‘Effusion’ occurrence, meaning the false negative test samples number is high.

Our work presented here may be further extended in many ways, besides testing the full database. Future improvements of our project may be divided to a couple of categories: pre-processing and cleansing of the data, and usage of other networks or layers. In terms of preprocessing, we can initially remove outlier images that any information cannot be extracted from (e.g. a totally white/black image). These outlier images can be found by calculating the mean and std of the gray level. Moreover, some of the images include letters and symbols on them, that may also be removed by deep learning methods or image processing methods.

Experienced research groups have investigated this problem in the past year and a half, suggesting the usage of much more deep neural networks. Try different networks and layers and fine-tuning of the parameters may improve the accuracy for predicting Effusion pathology.

Using machine and deep learning methods to solve different classification tasks is an extremely challenging, interesting and endless work.

**Bibliography**

*[1] Pranav Rajpurkar et al., CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning, (2017), URL https://arxiv.org/abs/1705.02315*

*[2] Pranav Rajpurkar et al., Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists, (2018), URL https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1002686*

*[3] Xiaosong Wang et al., ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases, (2017), URL* [*http://openaccess.thecvf.com/content\_cvpr\_2017/papers/Wang\_ChestX-ray8\_Hospital-Scale\_Chest\_CVPR\_2017\_paper.pdf*](http://openaccess.thecvf.com/content_cvpr_2017/papers/Wang_ChestX-ray8_Hospital-Scale_Chest_CVPR_2017_paper.pdf)

*[4]* [*Luke Oakden-Rayner*](https://lukeoakdenrayner.wordpress.com/)*, Exploring the ChestXray14 dataset: problems, (2017), URL* [*https://lukeoakdenrayner.wordpress.com/2017/12/18/the-chestxray14-dataset-problems/*](https://lukeoakdenrayner.wordpress.com/2017/12/18/the-chestxray14-dataset-problems/)

[5] *Sergey Ioffe and Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, (2005), URL*

*<https://arxiv.org/pdf/1502.03167.pdf>*