

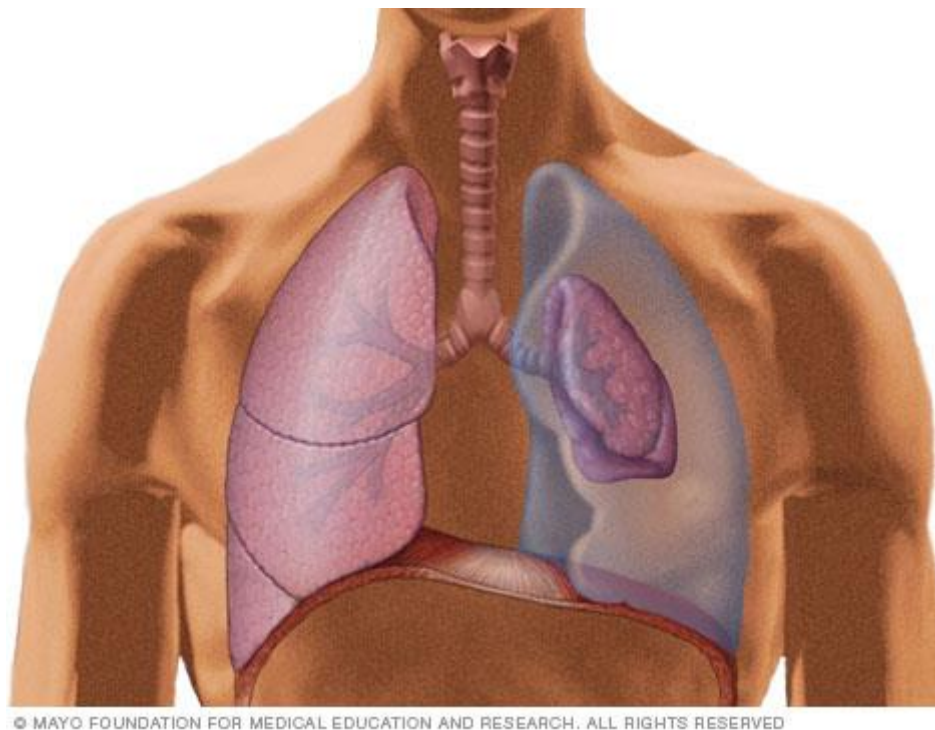
# Udacity Machine Learning Engineer Nanodegree

## Capstone Project

### Pneumothorax Segmentation in Chest X-ray Images

#### 1. Introduction

A pneumothorax is a collapsed lung. It occurs when air leaks into the space between the lung and the chest wall. This air pushes on the outside of the lung and makes it either partially or completely collapse – Fig.1 [1].



**Fig.1:** Pneumothorax: collapsed vs. normal lung. Adopted from [1]

The pneumothorax can be caused by a blunt or penetrating chest injury, certain medical procedures, damage from underlying lung disease, or most horrifying - it may occur for no obvious reason at all. The main symptoms usually include sudden chest pain and shortness of breath. On some occasions, a collapsed lung can be a life-threatening event. These symptoms can be caused by a variety of health problems, and pneumothorax is usually diagnosed by a radiologist on a chest x-ray, and can sometimes be very difficult to confirm.

Medical imaging is a rapidly evolving field due to large breakthroughs in deep learning techniques, which emerged in the last ten years. An accurate AI algorithm to detect pneumothorax would be useful in a lot of clinical scenarios. AI could be used to triage chest radiographs for priority interpretation, or to provide a more confident diagnosis for non-radiologists. Treatment for a pneumothorax usually involves inserting a needle or chest tube between the ribs to remove the excess air. However, a small pneumothorax may heal on its own.

## 2. Problem Statement, Data and Metrics

The Society for Imaging Informatics in Medicine, American College of Radiology (SIIM-ACR) and Kaggle organized a competition challenge in the 2<sup>nd</sup> half of 2019 to develop a model to classify (and if present, segment) pneumothorax from a set of chest radiographic images in order to aid in the early recognition of pneumothoraxes and save lives [2].

The original data set for this challenge was pretty large and was in the DICOM format, which is standard in medical imaging. Although the raw DICOM data is no longer available, the images converted to PNG format are available on Kaggle in different resolution [3-4]. The training set consists of images and corresponding binary masks, where pneumothorax is encoded.

There are total of 12,289 train images with corresponding binary masks, where pneumothorax is labeled, and 3205 test images, for which predictions have to be made. The size of the data in full resolution is about 7.5 GB. Several examples of the images are visualized in Fig. 2a, where pneumothorax is shown in solid dark red color. The data set is fairly imbalanced and only about 22 percent of the images contain pneumothorax.

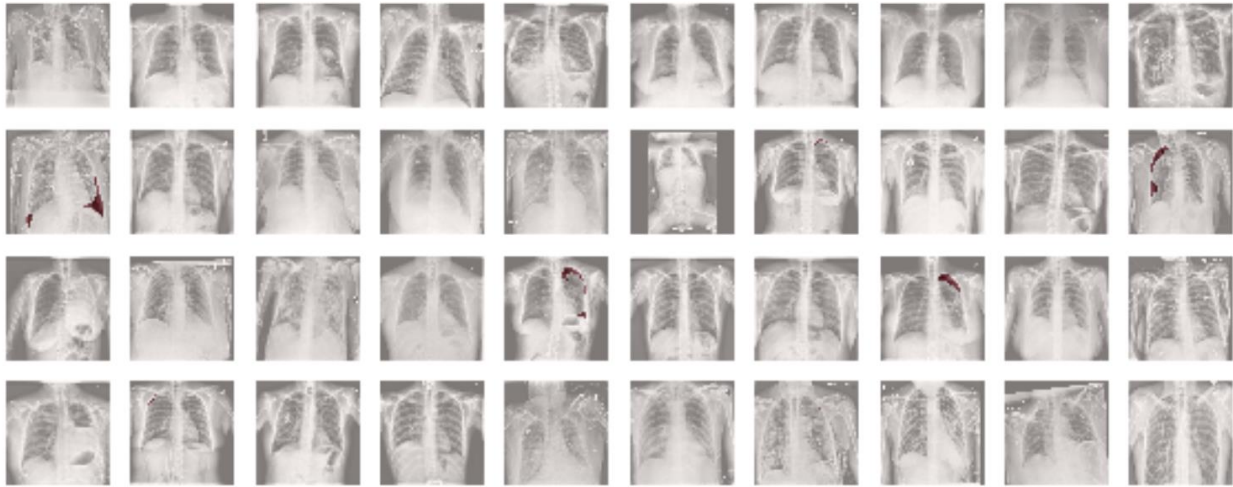
The goal of this challenge is to train a model using the training data and to make predictions on the test set. The predictions can be scored by uploading them to the Kaggle website. Late submissions are still available for this competition, even though it has already finished. I set a goal to develop a model, which will allow to place in the top 10% of the leaderboard (in the medals) out of 1475 participants, with a stretch goal to get in the top 3% on the Kaggle private leaderboard [5]. Such a goal was chosen to ensure that the developed model is on par with some of the best approaches developed during the competition.

The competition is evaluated based on the mean Dice coefficient metric. The Dice coefficient is used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth using the following formula:

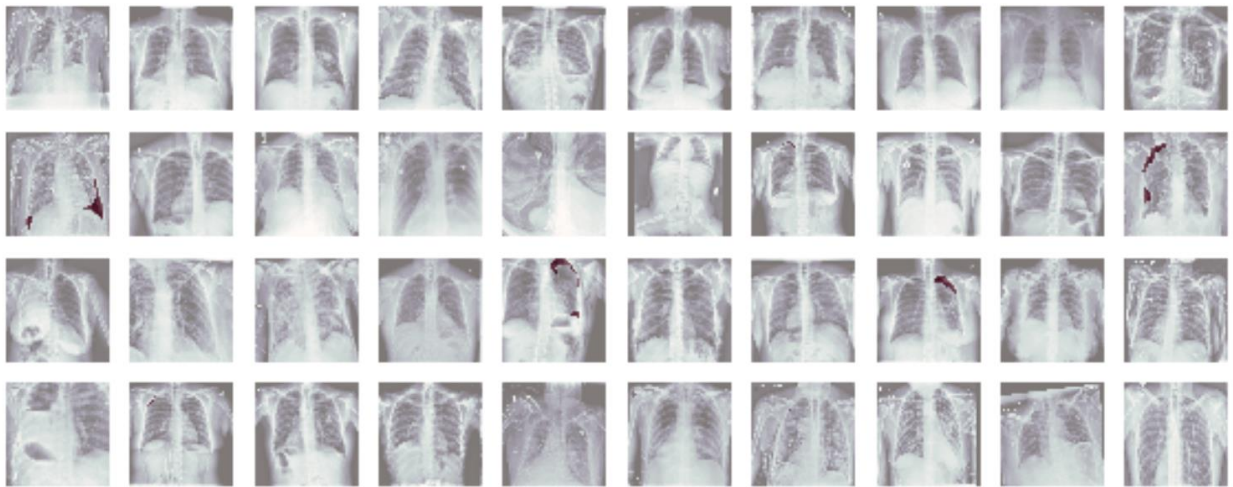
$$\frac{2 * |X \cap Y|}{|X| + |Y|},$$

where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is defined to be 1 when both X and Y are empty. The leaderboard score is the mean of the Dice coefficients for each image in the data set.

(a)



(b)

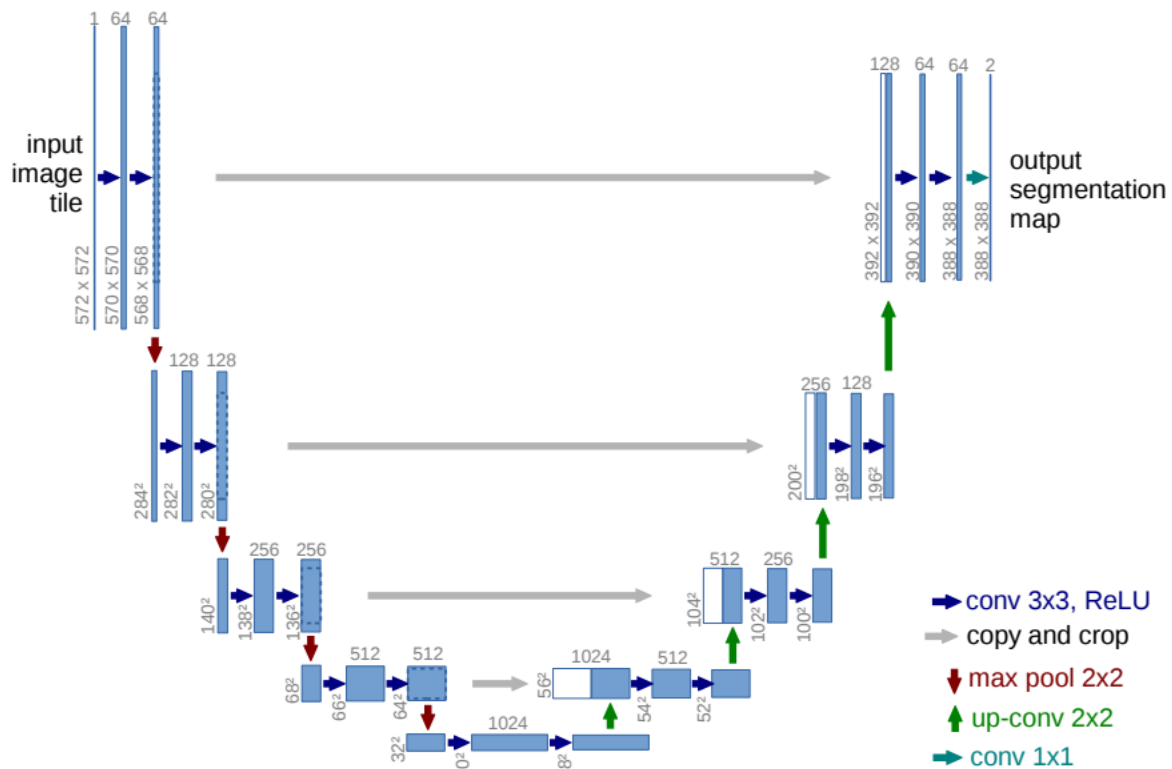


**Fig. 2:** (a) Original images from the training set and (b) their augmentations. Dark red color depicts pneumothorax.

### 3. Model Architecture and Implementation

To build a model I utilized a Unet encoder-decoder architecture [6], which is known to work well for segmentation problems in biomedical field as well as in the Kaggle competitions, where I participated before, e.g. [7]. The Unet model takes an input image and outputs a segmentation binary mask. The architecture consists of a contracting path of the encoder to capture context and a symmetric expanding path of the decoder that enables precise localization – Fig.3. In order to further improve the localization high resolution features from the contracting path are combined with the upsampled path via cross-connections.

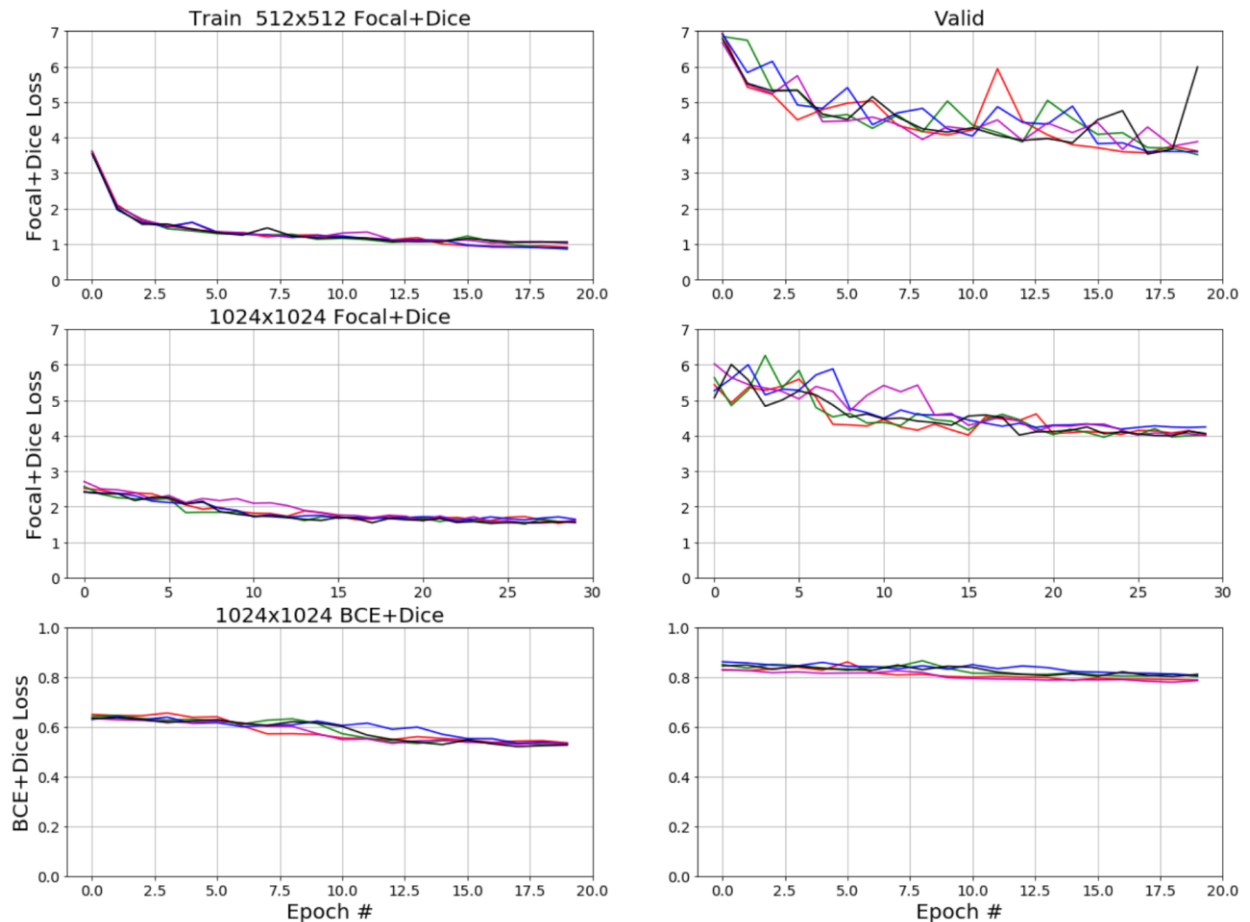
Pre-trained Unet models with different encoders are publicly available, e.g. in [8]. I used a model with ResNet34 encoder pre-trained on ImageNet [9] and implemented the model using Pytorch framework in Python.



**Fig. 3:** U-net architecture (example for 32x32 pixels in the lowest resolution). Adopted from [6].

Below is a summary of some of the key features of the model:

- The training data set is split into 5 folds for cross-validation using StratifiedKFold from scikit-learn [10], to get the same fraction of images containing pneumothorax in each fold. As a result, the fraction of the training data in each fold was 80% and validation data was 20%.
- Albumentations library [11] is used for train image augmentations to reduce overfitting and improve model robustness. Fig. 2 shows examples of the original images (a) as well as augmented images (b), where rotations, crops, scaling, shifts, brightness and contrast adjustments, etc have been applied with some probability. Solid dark (red) color shows location of pneumothorax.
- Images have also been scaled using ImageNet normalization coefficients.
- Adam optimizer and ReduceLROnPlateau learning rate scheduler are used.

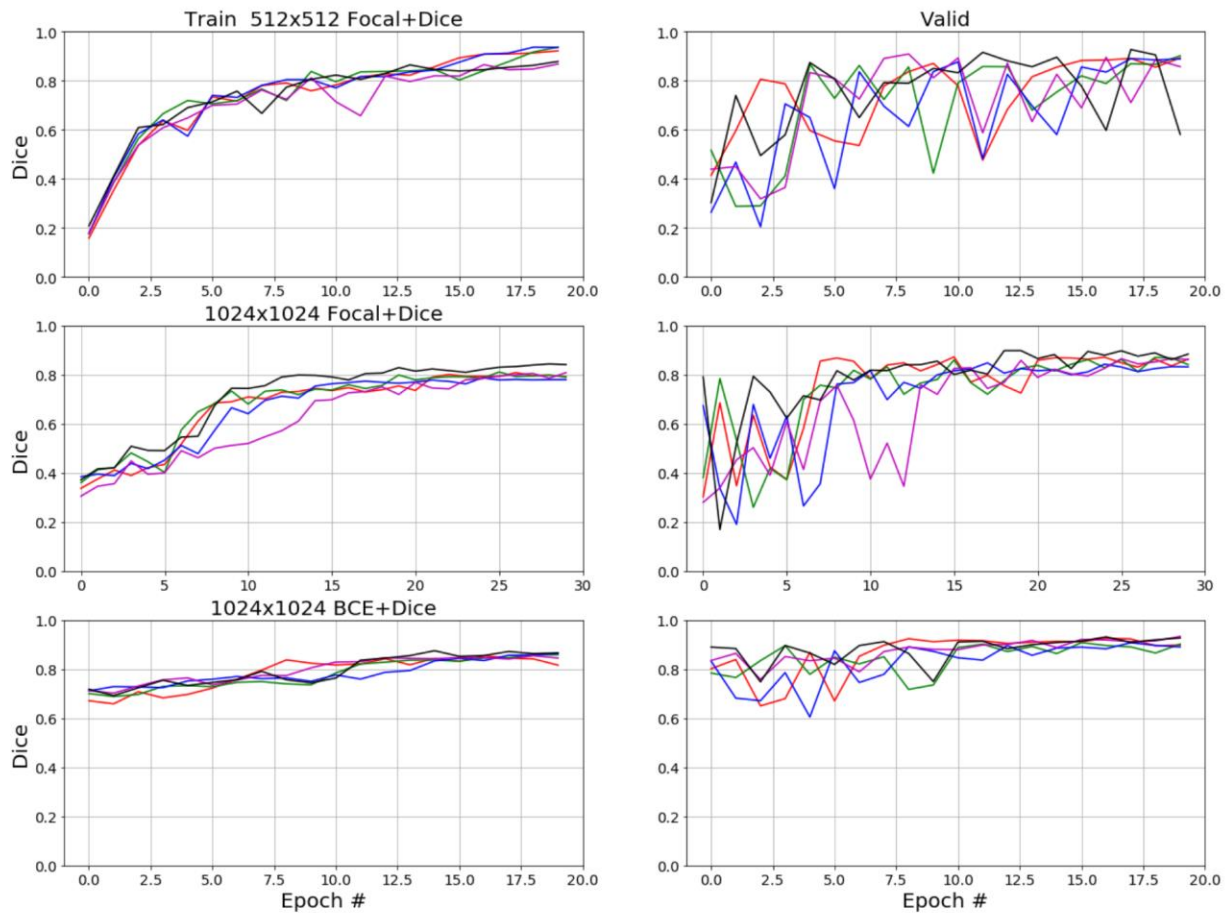


- **Fig. 4a:** Model loss vs. number of epochs during train (left) and validation (right) phases. Each plot depicts 5 folds. Top panel - 1<sup>st</sup> step: image\_size=512x512, focal+dice loss, 20 epochs; middle panel - 2<sup>nd</sup> step: image\_size=1024x1024, focal+dice loss, 30 epochs; bottom panel - 3<sup>rd</sup> step: image\_size=1024x1024, BCE+dice loss, 20 epochs.
- Two different loss functions have been employed: (1) a sum of the focal loss [12] and the dice loss and (2) a sum of the binary cross-entropy and the dice loss.
- The best model for each fold is determined (and saved) during training, when the local dice validation score is maximized.
- Only horizontal flip augmentation has been employed during inference phase. Final model predictions are made by averaging predictions (ensembling) over 5 folds on original and images flipped horizontally.
- In addition, it was crucial to zero small connected objects within a mask, identified as pneumothorax, to remove false positives.
- Also, since the size of the data is large and I only had 2 1080Ti GPUs available on my home computer, I further reduced the amount of training data by down sampling the images without pneumothorax to obtain the same number of images with and without pneumothorax, which allowed to speed up training time by 2x and still achieve good

model accuracy. I did not down sample the validation data, since inference was significantly faster and it allowed me to select the best model using initial balance ratio.

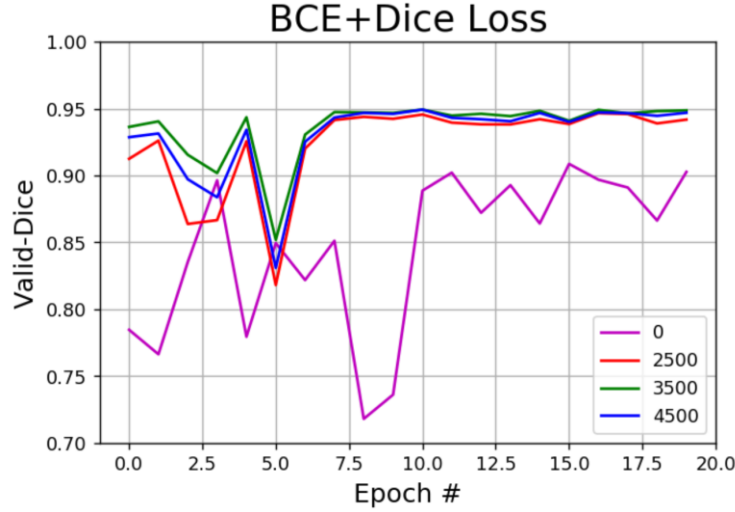
- Since when training on 1024x1024 images I could only fit a batch\_size=4 into 11GB of 1080Ti GPU memory, I used graduate accumulation technique to add together loss from several batches (size=4).

I benefited a lot from reading the Kaggle forums, where, in particular, I picked up ideas for the initial model, including, first, pre-training initial model on smaller 512x512 images for faster conversion and secondly for using a sum of focal loss and dice loss, but was able to eventually improve the model by implementing some of my own ideas.



**Fig. 4b:** Dice score vs. number of epochs during train (left) and validation (right) phases. Each plot depicts 5 folds. Top panel – 1<sup>st</sup> step: image\_size=512x512, focal+dice loss, 20 epochs; middle panel – 2<sup>nd</sup> step: image\_size=1024x1024, focal+dice loss, 30 epochs; bottom panel – 3<sup>rd</sup> step: image\_size=1024x1024, BCE+dice loss, 20 epochs.





**Fig. 5:** Comparison of dice score during validation phase for different minimum object size parameters.

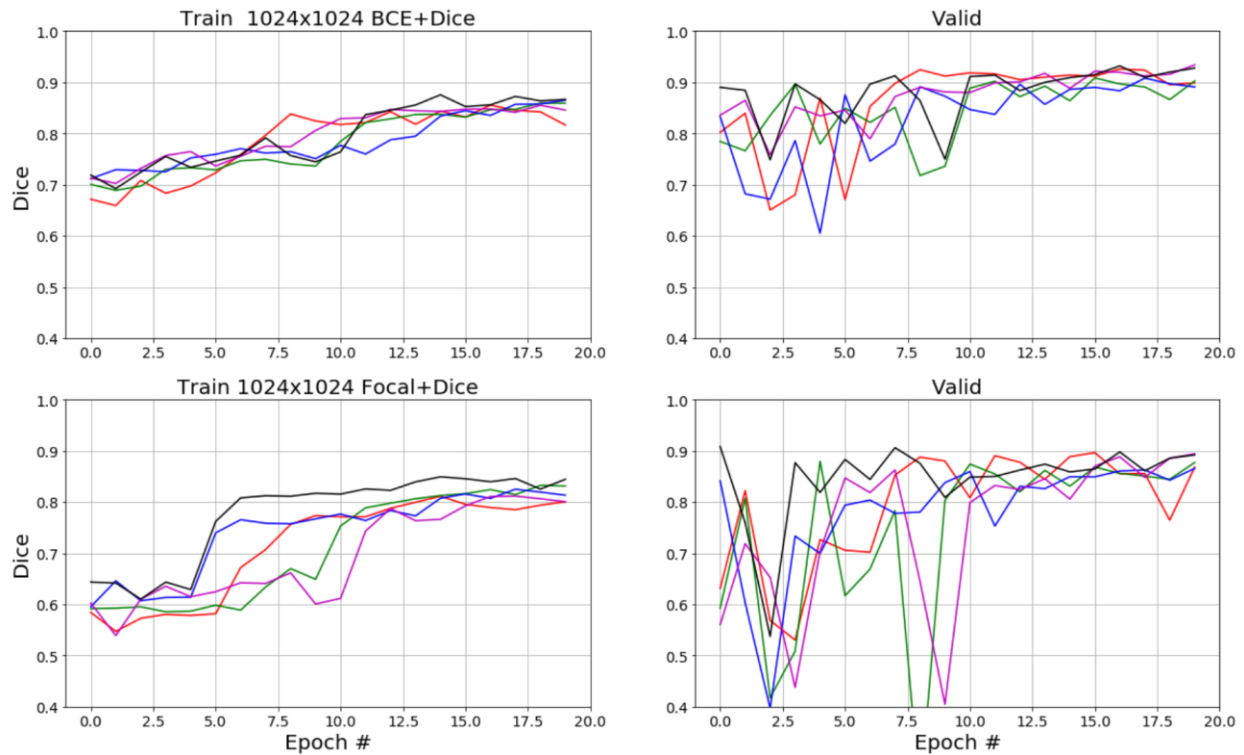
I kept refining the code during the course of my work on model development and improvements and finally got to the point where the code is modular, all model parameters are defined through configuration files and all the training steps and model artifacts (outputs) are logged, so that all the experiments are well documented and can be reproduced. The code is available on Github [13] and description of the code structure is provided in Appendix I.

#### 4. Model Training, Validation and Improvements

As a 1<sup>st</sup> step to help with model conversion, I initially pre-trained the Unet model on reduced size images of 512x512 for 20 epochs. I used a sum of the focal loss and dice loss, batch size equal 16 and graduate accumulation parameter equal 2.

For model evaluation, I relied on the local dice validation based on individual folds, which turned out to be a good predictor for the eventual result on the **private** leaderboard used to score this competition.

Training on 512x512 images, allowed me to obtain the score of 0.7912 on the private leaderboard, which was only enough to get to the top ~20% in the completion. Then in the 2<sup>nd</sup> step, I upscaled the model to 1024x1024 size images and trained it for another 30 epochs. Since Unet is a fully-convolutional model, one can first train the model on smaller images and then progressively increase image size for faster conversion using weights from the previous step, since convolutional filters are not dependent on the image size. However, the result for 1024x1024 image size initially dropped somewhat on both local validation as well as on Kaggle.



**Fig. 6:** Dice score for model using BCE+Dice loss (top panel) vs. model using Focal+Dice loss during 3<sup>rd</sup> step. Model using BCE+Dice performs better.

Top two panels in Fig. 4a show model loss for each of the 5 folds on the training (left) and the validation (right) data and top two panels in Fig. 4b show how dice loss changes on training and validation for 512x512 and 1024x1024. One can see that the training loss and validation loss reach plateau during training on 1024x1024 and the validation dice as well as the training dice are actually higher for the 512x512 model. I attempted to further train 1024x1024 model at these conditions for another 20 epochs, but it did not improve the score, which was very puzzling.

At this point I tried several other things to improve the model and made few discoveries. First of all, the model was making a lot of false positives while predicting small pneumothorax objects in the mask. Since we know from looking at the raw images that pneumothorax is unlikely to be only few pixels large in size, we can attempt to zero small connected positive predictions within the mask to see if this helps.

Fig.5 is presenting an example of how changing the minimum pneumothorax object size helps to improve the dice metric on validation. There is a clear jump in performance, after this cut off is introduced. The optimal min object size is about 3500 pixels although the differences with 2500 and 4500 are fairly small. Introducing this cutoff in the model during inference (test) phase allowed to significantly improve the score of the 1024x1024 model to 0.8395 and get into top 7% on the Kaggle private leaderboard.



(a)

54	▼ 27	[ods.ai] formemorte		0.8483
55	▲ 176	A.Spct		0.8482
56	▲ 50	gcpyobcmst		0.8481
57	▲ 112	VinBDI		0.8480
58	▼ 45	OmerS		0.8477
59	▼ 39	Artem Toporov		0.8475

(b)

<a href="#">submission_pytorch_5fold_ave_Wflip_0p5th_FineTunedM7withBCE.csv</a>	0.8479
2 days ago by <a href="#">AlexeyK</a>	
Model 7 tuned with BCE+Dice for 20 epochs	

**Fig. 7:** (a) Snapshot of the Kaggle private leaderboard [5], (b) Final performance of the model on Kaggle (Private Leaderboard score). Obtained score would allow to take the 58<sup>th</sup> place out of 1,475 teams, which participated in this completion, which corresponds to the top 4% result or a Silver medal.

Apparently before the cutoff was introduced the 512x512 model was creating smaller number of false positives than 1024x1024 model thus resulting in a better score, but after the cutoff, the 1024x1024 model started giving better performance.

The value of the dice score seems to depend fairly significantly on how balanced the data is. Since it is probably easier for the model to predict empty masks, this likely explains why observed dice score is typically higher on the validation, which contains larger proportion of empty images compared to the “balanced” training set.

The 2<sup>nd</sup> big improvement came from trying a different loss function. It turned out that adding training with a sum of binary cross-entropy (BCE) loss and dice loss as a 3<sup>rd</sup> step allowed to further improve the model (bottom panel of Fig.4 a,b) compared to the initially used focal plus dice losses, as is also evident from Fig. 6, which compares training and validation dices. The training of the models was performed from the same initial state (after 2<sup>nd</sup> step) for 20 epochs for 5 different folds. This increase in the local score also translated to an increase in the leaderboard score of 0.8479 and 58<sup>th</sup> place (out of 1475 participants) or top 4% - Fig. 7.

I have also tried using Lovasz loss, which worked well in previous Kaggle competition related to salt segmentation [7], which I participated in, but I could not get it to work properly here.

Overall training of all 3 steps took about 10.5 hours per fold (~1 hour for the 1<sup>st</sup> step, 5.5 hours for the 2<sup>nd</sup> step, and 4 hours for the 3<sup>rd</sup> step) or 52.5 hours for all 5 folds on a single 1080Ti Nvidia GPU (or half of that on 2 1080Tis). The inference on the test data took about 1 hour.

I have not tried to optimize prediction threshold for the binary mask, which determines when pixel is categorized as having pneumothorax and when it is not, since default value of 0.5 already provided good model performance and retraining the model with different thresholds would take pretty long and be costly, but given available resources one can try to optimize it as well.

As I later found out some of the winning solutions (top-10 – gold medal level) in this competition used large ensembles of much deeper and heavier neural nets like se-resnext50, se-resnext101, SENet154, which were trained on multiple GPUs or even on Nvidia DGX workstations, however the 1<sup>st</sup> place score was only higher by 0.02, thus we can conclude that model developed here achieves pretty good performance.

## **5. Conclusions**

We have developed a relatively light-weight Unet model for Pneumothorax Segmentation in Chest X-ray Images using a ResNet34 encoder pre-trained on ImageNet and Albumentation library for image augmentations. The final model ensembled over 5 folds allowed to place in top 4% on the Kaggle leaderboard (a Silver medal level), which exceeds initial goal of placing in the top 10%.

## References:

- [1] Pneumothorax – Mayo Clinic: <https://www.mayoclinic.org/diseases-conditions/pneumothorax/symptoms-causes/syc-20350367>
- [2] [www.kaggle.com/c/siim-acr-pneumothorax-segmentation](https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation)
- [3] <https://www.kaggle.com/iafoss/siimacr-pneumothorax-segmentation-data-512>  
<https://www.kaggle.com/iafoss/siimacr-pneumothorax-segmentation-data-1024>
- [4] <https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/data>
- [5] <https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/leaderboard>
- [6] Olaf Ronneberger, Philipp Fischer, Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, <https://arxiv.org/abs/1505.04597>
- [7] Kaggle – TGS salt Identification Challenge: <https://www.kaggle.com/c/tgs-salt-identification-challenge>
- [8] Pavel Yakubovskiy, Segmentation Models for Keras and Pytorch, Github:  
[https://github.com/qubvel/segmentation\\_models](https://github.com/qubvel/segmentation_models) and  
[https://github.com/qubvel/segmentation\\_models.pytorch](https://github.com/qubvel/segmentation_models.pytorch)
- [9] ImageNet: <https://en.wikipedia.org/wiki/ImageNet>
- [10] [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.StratifiedKFold.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html)
- [11] Albumentations: Fast and Flexible Image Augmentations: <https://github.com/albumentations-team/albumentations>
- [12] T.-Y. Lin, et al, "Focal Loss for Dense Object Detection", <https://arxiv.org/pdf/1708.02002.pdf>
- [13] <https://github.com/akuritsyn/udacity-ml-nanodegree/tree/master/pneumothorax>

## Appendix I: Model Code Structure and Software Requirements

**Requirements:** The model has been developed using Python (see requirements.txt):

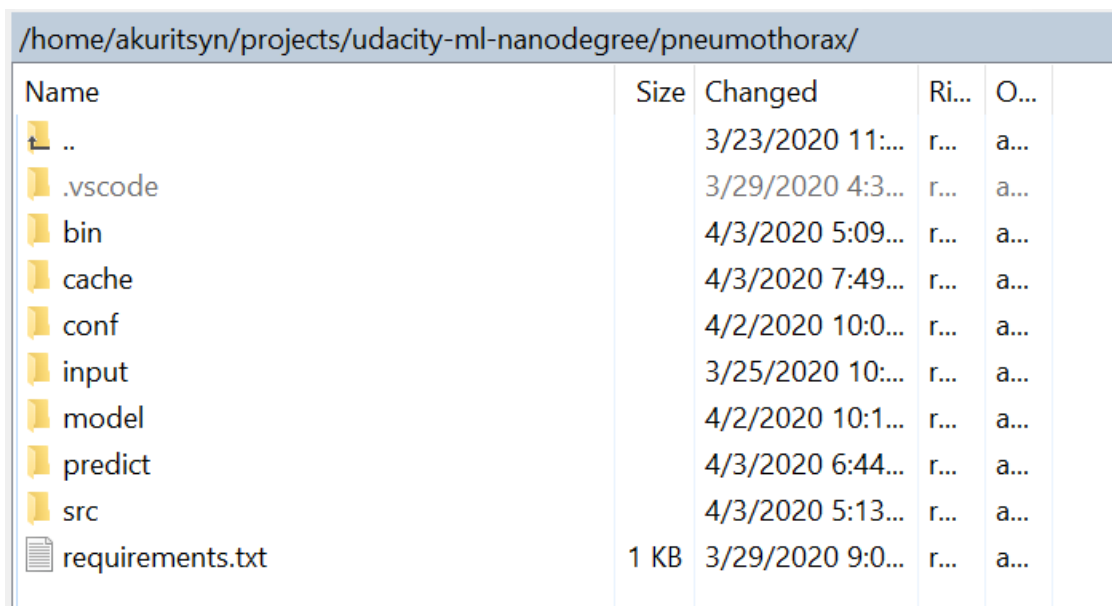
- Python=3.7.5
- Pytorch=1.3.1
- albumentations=0.3.0
- segmentation-models-pytorch=0.1.0

**Code directory structure** is shown in Fig. 8. Folders `./bin`, `./conf`, `./src` can be downloaded from Github [11]. Folders `./cache`, `./model` will be created automatically.

**Data:** The train and test data should be downloaded from [3] and saved in the `./input` directory at the root level and contain train and test data at 512x512 and 1024x1024 resolution as well as annotations for the train data `stage_2_train.csv` and a sample submission file `stage_2_sample_submission.csv`, which can be downloaded directly from Kaggle [4] (registration is required). The path to the data can be modified in the model config file, if necessary (see below).

**Scripts:** All the training and inference (predict) scripts are stored in the `./bin` directory.

**Configs:** All the model configuration files are stored in `./conf` directory. They contain all the model parameters, including training function, optimizer, loss function, training scheduler as well as paths and other necessary parameters.



Name	Size	Changed	Ri...	O...
..		3/23/2020 11:...	r...	a...
.vscode		3/29/2020 4:3...	r...	a...
bin		4/3/2020 5:09...	r...	a...
cache		4/3/2020 7:49...	r...	a...
conf		4/2/2020 10:0...	r...	a...
input		3/25/2020 10:...	r...	a...
model		4/2/2020 10:1...	r...	a...
predict		4/3/2020 6:44...	r...	a...
src		4/3/2020 5:13...	r...	a...
requirements.txt	1 KB	3/29/2020 9:0...	r...	a...

**Fig. 8:** Code directory structure.

**Model artifacts** are stored in `./model/model00X` directory. A separate directory is automatically created for each new model.

**Model source files:** All the source files are stored in `./src` directory.

- `main.py` contains model initialization and training, validation and test loops.
- `dataset.py` contains training and validation data loaders.
- `factory.py` contains routines for loading the model, optimizer, loss function and scheduler, all defined in the model config file.
- `Losses.py` contains loss functions.
- `metrics.py` contains dice metric calculations.
- `predictor.py` contains validation data loader and other auxiliary routines for making predictions that are called from the test loop in the `main.py`.
- `./preprocess/make_folds.py` contains routine for making folds at the beginning of the training process, which are later used by the model.
- `./utils` folder contains auxiliary files for reading config files, logging, etc.

### Running the model:

Please, make sure you run all the scripts from the parent directory of `./bin`.

- **Preprocessing:** run `$ sh ./bin/preprocess.sh` to make folds.
- **Training:** `$ sh ./bin/train00[X].sh`
  - Models 006, 007, 008 needs to be executed on all the folds 0...4, which need to be specified in the `train00[X].sh` file along with the GPU number. Also, for models 007 and 008 one need to specify corresponding folds from the previous model, which will be used for weights initialization in the model configuration file.
- **Inference:** `$ sh ./bin/test008.sh`
  - Model predictions on the test data will be saved in `./model/model008` (file name can be changed in config) and can submitted to Kaggle (registration is required).