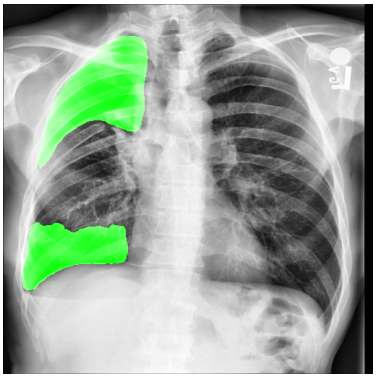
Work repport of SIIM's Kaggle competition

from 26 June to 4 September 2019

SIIM-ACR Pneumothorax Segmentation



Guillaume BALEZO

September 27, 2019

Contents

1 Introduction

The code of this project can be found at :
https://github.com/GuillaumeBalezo/Pneumothorax-Segmentation

1.1 Pneumothorax

Imagine suddenly gasping for air, helplessly breathless for no apparent reason. Could it be a collapsed lung? Pneumothorax can be caused by a blunt chest injury, damage from underlying lung disease, or most horrifying—it may occur for no obvious reason at all. On some occasions, a collapsed lung can be a life-threatening event.

Pneumothorax manifests itself as chest pain and respiratory discomfort. It is a disease of the pleura that causes the lung to collapse, resulting in respiratory consequences. Young people (between 15 and 40 years of age) and smokers are the most affected. The male sex also seems to be more exposed to this condition. Indeed, the incidence rate is about 8 to 18 cases per 100,000 in men compared to 2 to 6 per 100,000 in women. Pneumothorax can be severe as it can cause death in some cases, especially when both lungs are affected.

Pneumothorax is usually diagnosed via a radiologist on a chest x-ray, and can sometimes be very difficult to confirm. 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.

Therefore, it is not a classification problem, but more precisely a segmentation one. We want to obtain as output a mask (i.e. a binary image) of the same dimension as the radio image in input, taking the value 1 in the characteristic areas of the pneumothorax and 0 in the background.

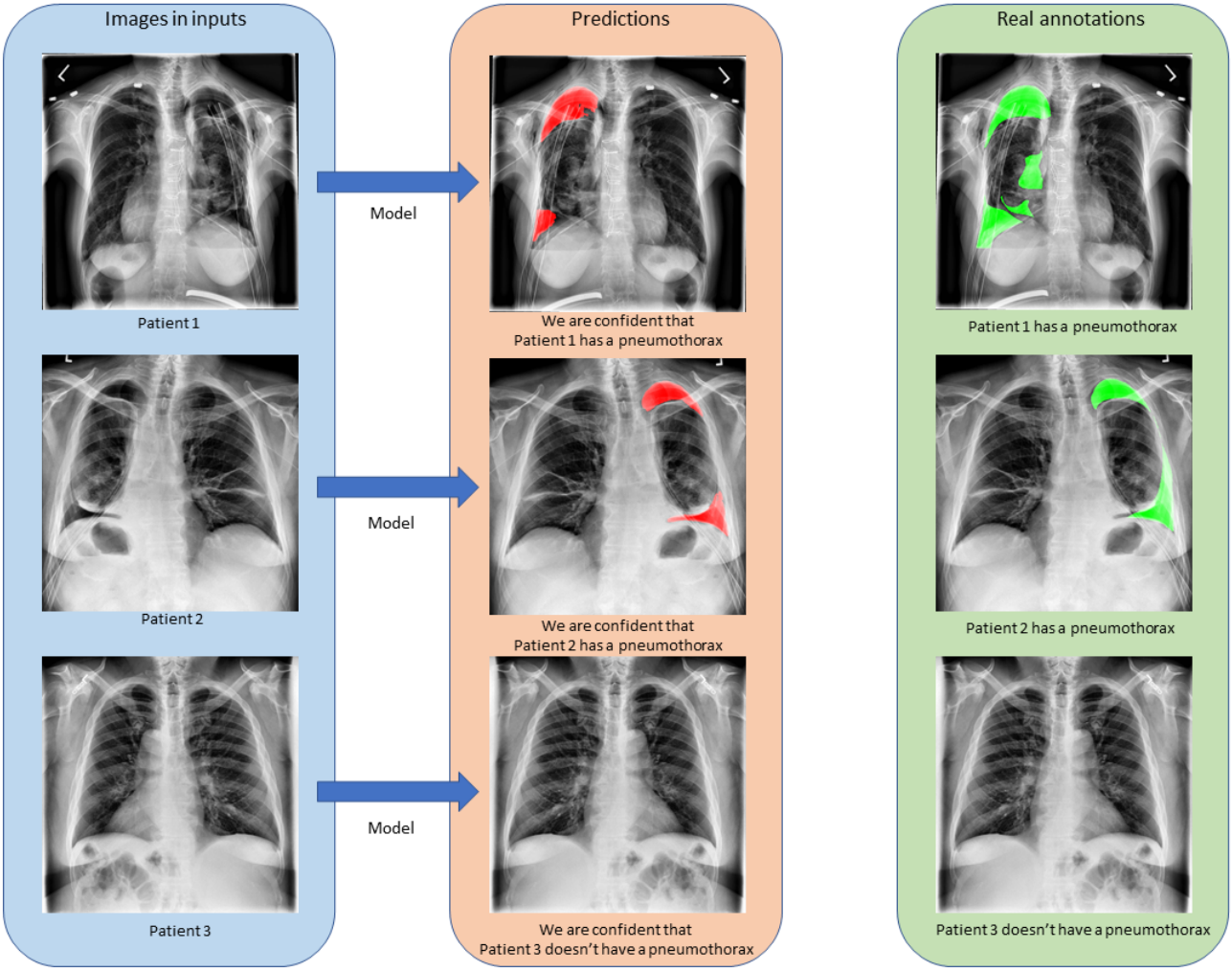


Figure 1: Some results of my single model

This document is organized as follows : after the introductory part 1, part 2 deals with data and how I handle it, part 3 is about my model Architecture and why I chose it, part 4 presents the training process and part 5 presents how I worked during this competition.

1.2 Information about the competition

The competition was organized by *Kaggle* and the *Society for Imaging Informatics in Medicine (SIIM)*. It is one of the leading healthcare organization for those interested in the current and future use of informatics in medical imaging. Their mission is to advance medical imaging informatics across the enterprise through education, research, and innovation in a multi-disciplinary community.

The evaluation metric of the models was the mean Dice coefficient. I will explain later what this metric is.

The competition lasted 3 months and hosted 1475 teams. Each of them could accept up to 5 competitors. The total amount of cash flow was 30000$. The competition took place in 2 stages. The second stage takes place only during the last week of the competition. The test set of the first stage was included in the training data and a new test set was provided for this new phase. This tip avoids manual labelling on the first test set since it is available for a sufficiently long time. In this second phase, competitors are allowed to re-train their models, but it is strictly forbidden to modify their architecture. In the following, when I talk about dataset, I will refer to the dataset of the first stage and every score I will give except for my final scores will be the ones of the first stage. In the end, competitors are allowed to choose only 2 models for their final score.

1.3 Information about my project

I used Keras as deeplearning framework. My code can be found at:
https://github.com/GuillaumeBalezo/Pneumothorax-Segmentation. The code can be found in two different formats: in Notebooks or in .py files.

I wrote this report in order as a testimony of my work, but also in order to gather the knowledge I have accumulated during this competition, which was the first I took part in. This report exists mainly for the purpose of retrospection.

2 Data exploration

This section is dedicated to introduce the data of the competition.

2.1 File format

Exceptionally for this competition, the dataset was not fully accessible via Kaggle. Only the masks in the form of csv files were available on the competition site. For the rest (i.e. the training and test images), it was necessary to use the Cloud Healthcare API. For my part, I downloaded the data via a pre-compiled dataset uploaded by a competitor in order to avoid the tedious manipulations of the API (allowed in the competition).

The dataset consisted of 12047 DICOM files (.dcm). This file format, which stands for Digital Imaging and Communications in Medicine, is a very common type of file for the analysis of medical images. Each DICOM file contains a 1024x1024 pixels size radio image of a lung. In addition to the images, it is possible to obtain additional information about patients, such as their gender, ID or age. However, I did not use this information for time-saving purposes. Probably, it would have been astute to give a different threshold for the sigmoid in output according to the gender. However I am not sure that the results would have been drastically different.

Therefore, among all of the information stored by the DICOM files, I only used the radio images. In a preprocessing phase, I transformed all DICOM files into PNG images. During each conversion, a contrast correction was applied to the images. The code for the preprocessing part did not come from me. Having downloaded the dataset directly as images in Kaggle, I copied the notebook to reproduce the dataset from DICOM files.

In order to compress the predictions sent to the site, the masks were encoded under Run-lenght encoding. Since the masks are mostly composed of zeros (see figure mask coverage), this compression method considerably reduces the size of the submitted files.

During Kaggle competitions, there are 2 leaderboards: a public leadearboard and a private leaderboard. The public leaderboard is given in real time and was calculated on % 30 of the test data. The private leaderboard, on the other hand, is only available at the end of the competition and is calculated on % 100 of the test data.

2.2 Imbalanced classes

One of the peculiarities of the training data is the imbalanced classes that occurs on two scales. First at the patient level: There are 9378 healthy patients and 2669 with pneumothorax. On the other hand, at the segmentation level, on images of patients with the disease only a small number of pixels correspond to pneumothorax compared to the background. On the figure 2, we notice that on average the mask coverage is almost null. This imbalance problem is an essential point, since the zero function, i.e. the function that predicts the total absence of pneumothorax in each patient, is an obvious local minimum. It will therefore be necessary to find methods during training to avoid convergence towards this function.

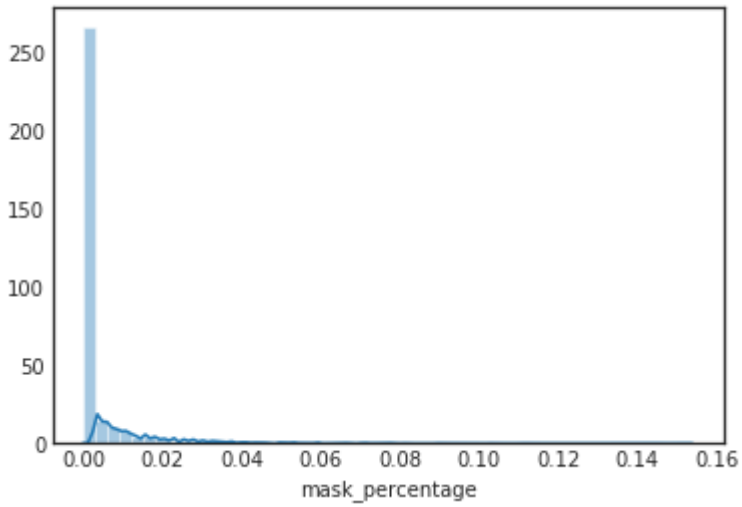


Figure 2: Average mask coverage

3 Model architecture

According to this article [1], there are plenty of models for segmentation problems. U-net [2], Mask-RCNN [3] and DeepLab [4] seemed to be the family of models with the most potential for this competition.

I chose the U-Net model because it seemed to me that starting with the simpler model among these last three was a good approach. In addition, U-Net is a very popular semantic segmentation model originally designed in 2015 for biomedical image segmentation and which has been improved by many versions since then. On the previous SAGLT Kaggle challenge, it was the U-Net architecture which had the best result, so it's one additional reason why I took this architecture as reference during all my work. Another reason I didn't choose Mask-NCRNN is because it is an instance segmentation model. In addition to being more difficult to train, this model does not adapt to the imbalance classes problem of our dataset. These types of models combine location and segmentation models, i.e. they require bounding boxes and masks. Mask-RCNNN was designed for COCO and PASCAL VOC dataset where the distribution of bounding boxes is much higher than in the competition dataset because of the imbalance classes. In a sense, all models have been tuned for higher distributions. Mask-NCRN and any other type of instance segmentation implementation may have a high number of false positives, which is highly penalized by the metric of the competition.

For reasons of understanding, I will first present the original U-Net model, then I will present the improvements made to it.

3.1 Classical U-net architecture

U-net is a semantic segmentation model initially created for the ISBI cell tracking challenge in 2015. Among its specificities, the model is composed only of convolutional layers. Thus, there are less parameters than with fully connected layers and consequently it requires less images. The last point is very important as our dataset is quite small, relatively to other used for deeplearning. In order to avoid over-fitting with a relatively small dataset, the model rely a lot on data augmentation. At the difference of other usual architecture, there is a up sampling path (cf figure) which increases the resolution of the output. It's done thanks to up convolution. We use this up-sampling path because in the end we want an output at the same dimension as the input and because it is impossible for computational reasons to keep the same dimension from the beginning to the end throught the computing process, we are forced to use pooling layers in the down sampling path in order to reduce the dimension. On the figure you can see skip connections represented by gray arrows. We concatenate a specific activation from the down sampling path with another of the up-sampling path (crop is required so that the dimensions match). This operation help localization but also for example in a Residual block, it helps a bit avoiding vanishing gradients. I skip the detail about the parameters of the convolution and pooling operations (blue and red arrows) as they will be different in my model.

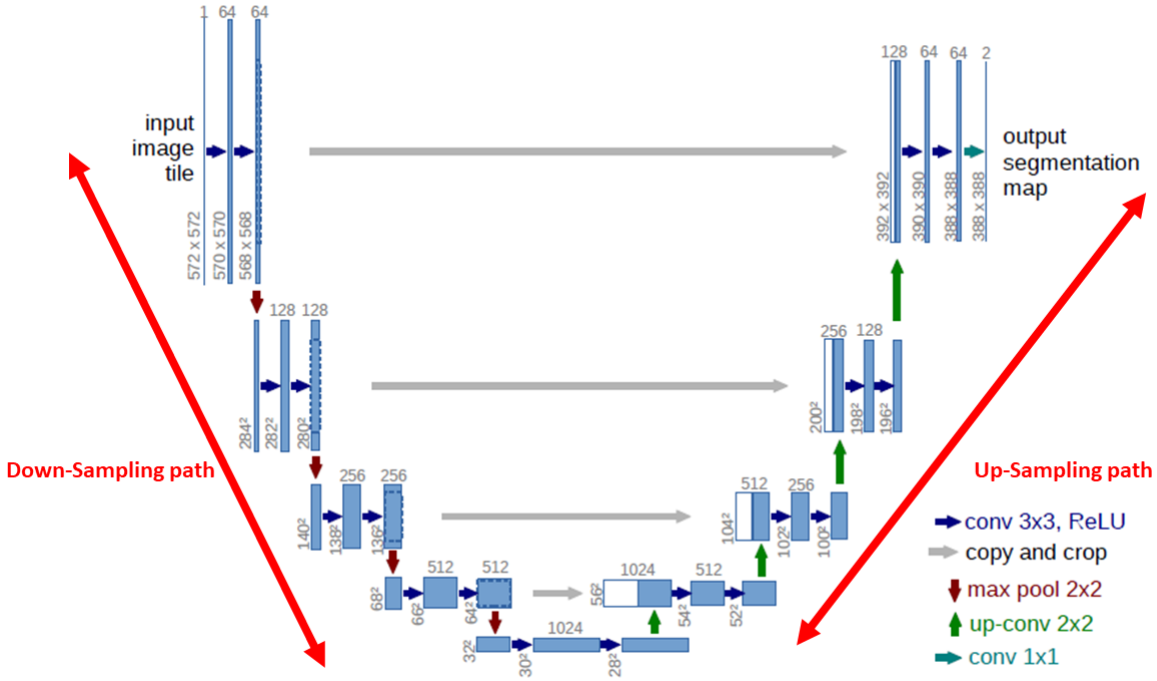


Figure 3: Classical U-net Architecture

3.2 Enhanced version of U-net

3.2.1 U-net with encoder and decoder

In the SAGLT Kaggle Challenge, many Kagglers have modified U-Net using an encoder and a decoder. The encoder is a pre-trained classification model that is used to extract features on the input image. Rather than performing a succession of 2 convolutions and then max pooling as in the down-Sampling path of classical U-Net (see figure...), we define certain layers of the encoder and we will use their activations in our UNet architecture (see figure for more clarity). We assume that by changing the simplistic encoder initially present in the U-Net model, we can opt for another more complex encoder that will be able to better extract features on the input image. We manage to have the layers of the encoder at different depths so that they extract more or less complex features. The deeper we progress in the down-Sampling path, the smaller the spatial dimensions of the activations are and the number of channels increases (see figure ...). Naturally, the pre-training facilitates convergence through transfer learning. Even before we start training, we have an encoder that is able to extract information from the images, hence its encoder name. I chose to use EfficientNet B4 pre-trained on Imagenet and will explain this choice in the next paragraph. The decoder also intervenes on the Up Sampling path. Rather than performing 2 convolutions before applying an upconvolution, it is possible to define more complex operations. For example, a common choice (and the one I used) is to use a Resnet decoder, i.e. to use Residual blocks rather than simple convolutions. For more details on the operations carried out, see figure...

The EfficientNet paper[5] was released in June 2019 at the begining of the competition. The EfficientNets family of models has a significant impact on the computer vision field, since they can match or even exceed models by taking both considerably less memory and fewer computations. There are 7 versions of EfficientNets. Among them, EfficientNet B7, the most efficient, manages to slightly exceed the state-of-the-art (as of mid 2019) GPipe on Imagenet with 9.6 times fewer parameters and being 6.1 times faster. I haven't been able to test all the EfficientNets, so the choice of EfficientNet B4 comes from the community. Many Kagglers claimed that EfficientNet B4 was the only encoder in this family to perform well. Most competitors who have used U-Net have chosen either this encoder or ResNext-50. The two models are quite comparable. ResNet50 has lower performance on Imagenet and more settings but fewer FLOPS. I chose EfficientNet B4 for its ease of use and elegance.

3.2.2 U-net++ architecture

Among the improvements of U-Net, I tried to implement U-Net+++ [6]. This version improves the skips connections in order to better address vanishing gradient problems that can be quite important in the basic model. However, the training was getting a little too long with this model.

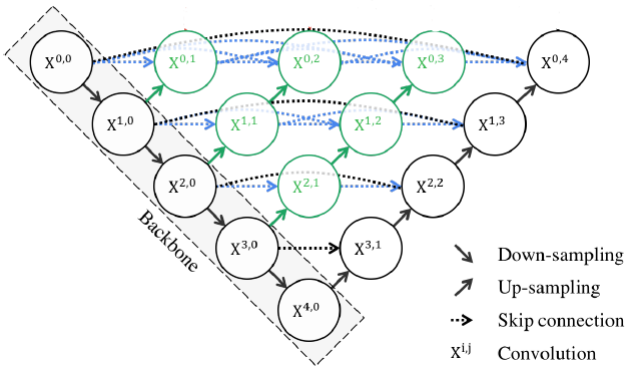


Figure 4: U-net++ Architecture (without supervision)

3.3 My U-net architecture: Simpler U-net++

Thanks to the competitor Siddhartha I opted for a simpler version of this model. The blue lines on the figure were mostly deleted from the model as it would have increased the computation. We only up-sample the deeper layer of the architecture as they are usually rich in semantic content. With the new architecture, the shallower layers can still take advantage of this semantic content. It's a good way to balance computation and use deeper layer features. You can check my architecture on the figure ... In the up-sampling path the elementary block (i.e. blue arrow) is applied to the concatenation of activation (not only the blue activation). On the other hand, the up convolution in the Hyperkernel paths are applied only to the blue activation. When there are consecutive concatenations, I did not represent the intermediate concatenations.

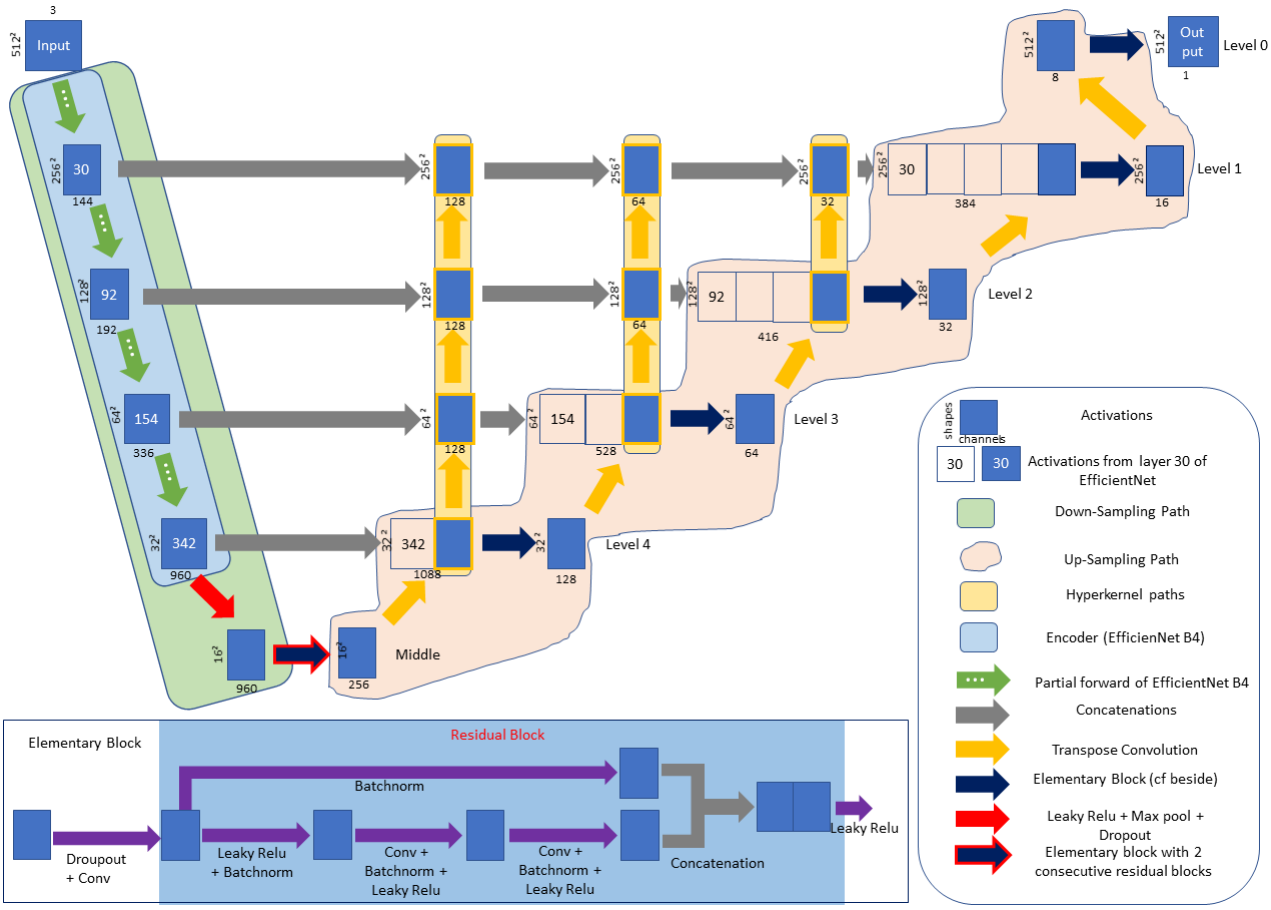
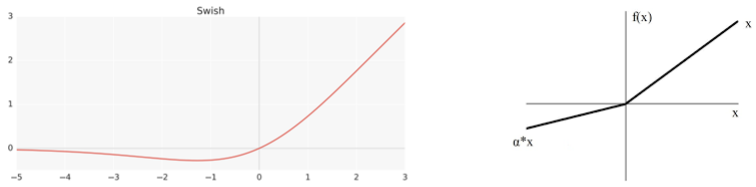


Figure 5: Architecture of my U-Net version

In the EffecientNet B4 encoder, the activations are the swish activation define like this: $f(x) = x.sigmoid(x)$. In the Unet Architecture *Leaky Relu* is used as activations. Both are rectified version of the classical Relu activation where the derivate in the negative part is not null. I used this function as often top Kaggle model use it. There is not a significant difference between Leaky Relu and Relu.



Swish activation Leaky relu activation

Figure 6: Activation functions

The input shape of my single model is 512x512 with 3 channels for the RGB coordinates.

3.4 Post processing

The techniques I will present here are some classical techniques used in competitions in order to boost the quality of prediction.

3.4.1 Test Time Augmentation

During the Post processing stage, I used Test Time Augmentation (TTA) with horizontal flip for predictions. As its name suggests, this is the equivalent of data augmentation but at test time. As horizontal

flip is applied during training and it is a rather simplistic augmentation, I also use it at test time to boost predictions. The figure below explains more precisely the TTA I used. This allowed me to have an increase in my score of about 0.001. In the post-processing phase, I also remove small predictions using the connected composent function of opencv. This allowed me to increase my score by another 0.001.

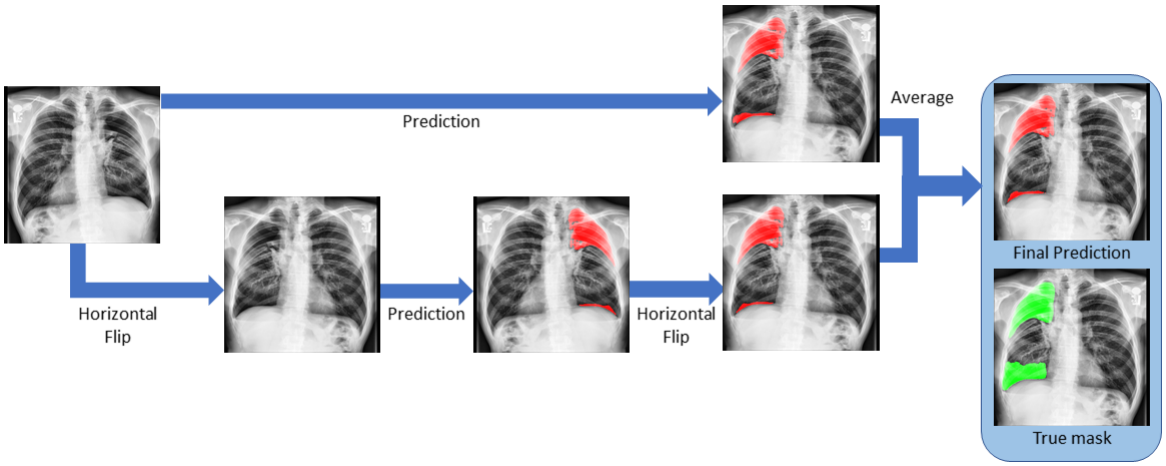


Figure 7: Test Time Augmentation flowchart

3.4.2 Ensembling Model

As I have said earlier, my best model was not the one I chose for my final score. It is in fact an ensemble between 2 models: my single best model that I have presented above and a model trained on 768x768px images. For each image, after predicting the masks with the 2 models, I average the results. This trick is popular Kaggle competitions, because it usually improves a bit the quality score. However, this tip is nearly never used in production by companies, as it required a lot of computations for the small gain. In my case, I trained another model and the prediction at test time required twice time more computations (even more so, as the input size of the second model is higher than 512x512px).

This step is optional. It doubles the computation but doesn't increase a lot the performance

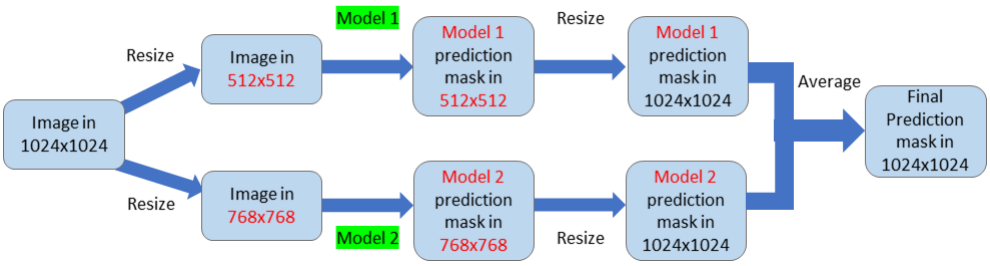


Figure 8: Ensembling flowchart

4 Training

In this section I will present the different training hyperparameters I have used, the strategies put in place to converge towards a better optimal as well as the learning curves.

For small images (i.e. 256x256px), I used Google Colab which gives free access to GPUs for 12 hours. For larger images (i.e. 512x512px, 768x768px and 1024x1024x), I used Google Cloud Plateform (GCP) instances. Each of the instances had 50GB of SSD, 1 NVIDIA Tesla T4 and 2 virtual processors with 13GB of RAM (n1-highmem-2). Kaggle offered 300$ Coupons for GCP Credit.

4.1 Training hyperparameters

4.1.1 Stratified sampling

The dataset was divided into three parts: train set, validation set and test set. As the test set was already provided in order to submit the predictions to Kaggle, it was necessary to split the labelled data into 2 to obtain the train and validation set. To do this, I carried out a stratified sampling at the beginning of each training session to maintain the proportion of patients with pneumothorax. Stratification makes the two sets more similar and makes the metrics from the test validation more consistent. 85% of the labelled dataset corresponds to the train set.

4.1.2 Training on multiple resolutions

One of the first hyperparameters that needs to be defined is the image size in inputs and output. For the choice of architectures, it is advisable to use small image sizes, i.e. 256x256px or 512x512px. Training is therefore faster, which makes it possible to conclude quickly and with a relatively high degree of confidence if one model in particular is better than another. Training hyperparameters can also be tuned easily. A fairly obvious intuition would then be that the higher the resolution of the images in inputs is, the higher the model's performance would be. However, it should be noted that the higher the resolution is, the more complicated the training becomes. On my setup it was necessary to count 1h20 per period for the resolution 768x768px and 3h for the resolution 1024x1024. Training directly on large images with my setup is unthinkable. One trick I used was to increase the training resolution as I went along. It is a form of transfer learning: we transmit what we have learned on low resolutions to higher resolutions. So I trained my model on images of sizes 256x256 then 512x512 and finally on 768x768. I also tried to train on 1024x1024 size images but the performance was too low. As a reminder, my best single model works on images of sizes 512x512px.

Note : Why did I train my model on the 256x256px, 512x512px, 768x768px and 1024x1024px resolutions ? The are two reasons. First, at each level of the U-net Architecture the spatial dimensions of the activations are divisible by two and there are 5 levels. So the resolution of the images in input has to be a multiple of 32. Secondly, the classical U-Net has been trained on 512x512px images, so this set of resolutions are relatively similar with this one. I introduced the 768x768px resolution because I thought that going from 512x512px to 1024x1024px images during training was to radical.

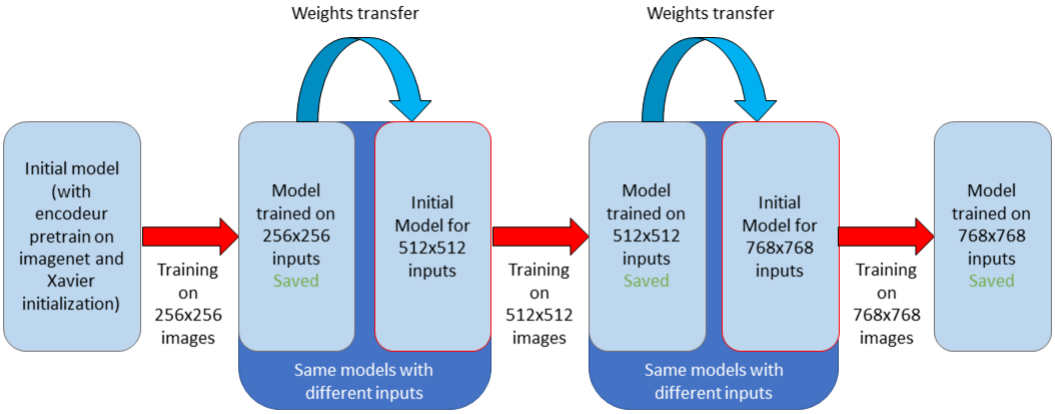


Figure 9: Multiple resolution training

4.1.3 How to manage small batch sizes

In addition, because of the up-sampling path, the model takes a lot of space in memory, which means that a very small batch size is required. For the resolution 512x512px the batch size was 4, for 768x768px 2 and finally 1 for the resolution 1024x1024px. It should be noted that a low batch size, i.e. less than 8, induces instability at the BatchNorm level and increases the error produced by the model. For small batch sizes, the GroupNorm layer [7] was introduced to compensate this problem, but I didn't have time to implement it. In addition, some community members seemed to complain that this type of standardization does not increase their performance in the competition. Moreover, having a small batch size can introduce noisy contradicting gradients near the optimal point and for this issue I used gradient Accumulation. In fact with this tip, we want to simulate that we have a bigger batch size. Instead of updating the parameters after each batch backward pass, we update the parameters after each *accumulatedsteps* backwards which is an hyperparameter. During the intermediary steps, the gradients are summed, and the loss is normalized (divided by *accumulatedsteps*). This technique allows to simulate a batch size of *batchsize* * *accumulatedsteps*. However, this technique does not allow to address the problem associated with the BatchNorm, since the simulation of a higher batch size take place only during the backward pass and the instability of the BatchNorm comes from the forward pass. For the 512x512px model I used an accumulation step of 8 in order to simulate a batch size of 32 while the real size was 4. For the 768x768px model, the accumulation step was 12. On the other hand, gradient checkpointing is a very effective trick that considerably reduce the amount of memory that training takes in exchange for a slightly longer calculation time. During backward pass all gradients are cached and it takes up a lot of memory space. To reduce the size of the gradient cache, only some of these gradients, called checkpoints, are saved. Thanks to the chain rule, to obtain the value of a given gradient, all gradients are recalculated from the last checkpoint. The time added by this technique during the backward pass therefore depends on the choice of the checkpoint. Some techniques have been developed to choose these checkpoints in order to optimize the computation time vs. memory problem. I did not use this method because it was complex to use in Keras and for time reasons.

4.1.4 Data augmentation

Data augmentation is a very important step and especially in computer vision and biomedical imaging. As explained earlier, the model relies heavily on this technique. Among the usual one I used: horizontal flipping, random contrast, random gamma, random brightness and random cropping with a zoom of approximately 0.8. As the images are medical images, elastic transforms, grid distortion and optical distortion are also applied because they can represent some variations between the images.

4.1.5 Metrics

The metrics I used were the Intersection over Union (iou) metric (or Jaccard index) and the dice coefficient. This two metrics are equivalent which means that if one model has a better score with one metric

against another one, it will be the same with the other metric. I could have used only one of them, but used them both out of curiosity. These two metrics are classical metrics used in segmentation and object localisation problems. Here are the formula of the metrics (with TP for True positive etc.):

Dice score:

$$\frac{2TP}{2TP + FP + FN}$$

IoU :

$$\frac{TP}{TP + FP + FN}$$

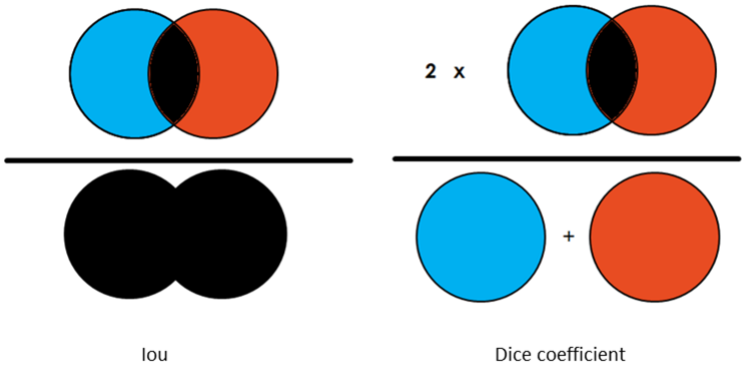


Figure 10: Metrics calculation

4.1.6 Losses

There are a multitude of loss for segmentation problems. Thanks to this article (ref below), I was able to get an exhaustive list of them. The 4 main ones I have chosen are binary cross entropy (BCE), dice loss, focal loss [9] and finally Lovasz loss [10]. For most of my training, I used a combination between the BCE and dice loss which is a quite popular loss. Dice loss maximizes the dice coefficient. Similarly, the Lovasz loss is a function defined directly to maximize the Jaccard index (iou). The paper explains how this function is constructed to have a smooth extension of the discrete metric iou. This last loss did not seem to work. Indeed, with this metric my model converged towards the null function. I could have tried a weighted version of this Lovase but I did not because of time constraints. As for the focal loss, I used it instead of the ECB at the end of the training. This function is an improved version of the BCE that gives a high weight to the easy classified element. This is a loss that is applied in cases of imbalance classes.

4.1.7 SWA and Learning rate scheduler

In the deeplearning field where we deal with very complex models, converging quickly towards a local minimum is not necessarily a good solution since it will probably be very far from the global minimum. Instead, we must encourage the exploration of local minima. I will present here the methods I have used to encourage exploration. First, I used cosine anealing as Learning Rate Scheduler (see Figure ...). This allows to have a high learning rate at the beginning which allows to converge quickly enough towards an optimal zone and then to approach more and more the minimum of the zone by small steps as the learning rate decreases. To encourage exploration, a trick is to use a cyclical learning rate but I didn't use it because it would have increase the training time. I also used Stochastic Weight Averaging (SWA) [11]. Rather than directly updating the model weights after the backprop, the average is calculated between the new calculated weights and the old ones. The paper shows that SWA allows the model to generalize better, by giving much flatter solutions than SGD and can approximate Fast Geometric Ensembling (FGE) by using only one model which can considerably reduce the computational cost. The method must be used as the end of the training. I usually used SWA only for the last 3 epochs.

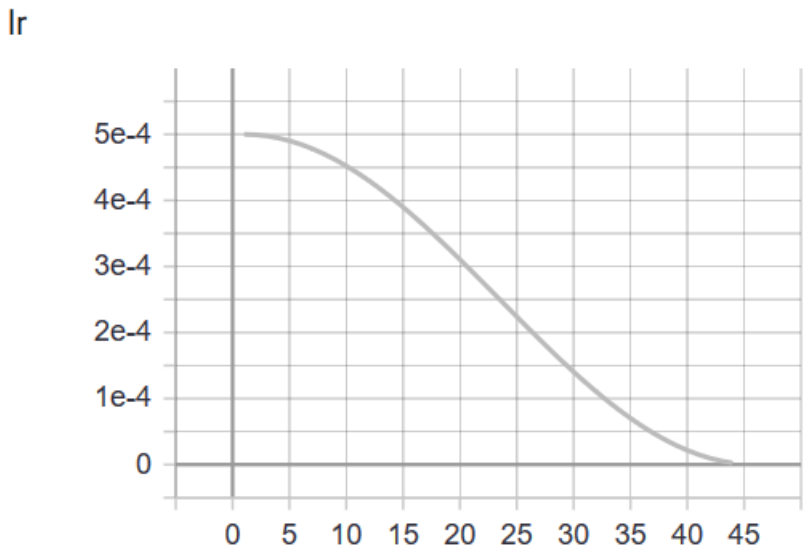


Figure 11: Cosine annealing (on 45 epochs with intial learning rate 5e-4)

4.1.8 Callbacks

Among the callbacks, I used Model checkpoint on the loss and the iou and dice coeff metrics in order to save the models which maximize them. I also used SWA and cosine annealing presented above. To visualize the learning curves, I used Tensorboard.

4.2 Learning curves

Unfortunately, I only managed to save the learning curves of the model trained on 768x768px resolution. For lower resolution models, I forgot to transfer the Tensorboard logs from the GCP instance to my PC. However, the approach I implemented for each resolution was the same every time: a long training over many periods, then a finer and shorter training with a lower initial learning rate and a different loss allowing to converge towards a better optimum.

I started the training for 75 periods for the 256x256px model with the Bce dice loss. I then trained this model on the 512x512px resolution in two steps. First for 50 epochs with dice loss and an initial learning rate of 1e-3. Then secondly with focal loss for 10 steps and with SWA. With this new model, I obtained a score of 0.8608 on the public leaderboard.

The training of the model on the 768x768 resolution took place over 45 periods for 1 day, 7 hours 50 min and 16 s. The initial learning rate was 5e-4.

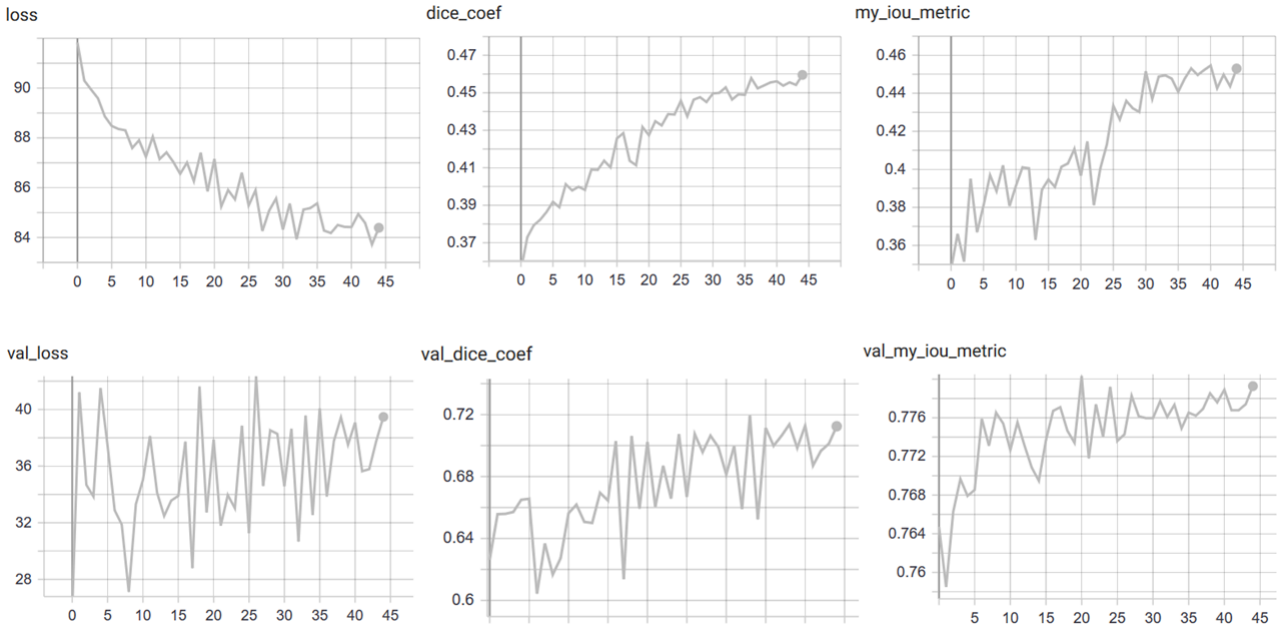


Figure 13: Learning curves of the first training on 768x768 resolution

The model I kept on this training is the one of the last epoch since it had almost the best iou and dice at the same time. The model had a loss of 39.48, a dice coefficient of 0.7125 and an iou of 0.7793 on the validation set.

I retrained this model more finely for 19 eras for 13hours 34 min and 33 s. The initial learning rate here was 1e-5 and SWA was applied from epoch number 8.

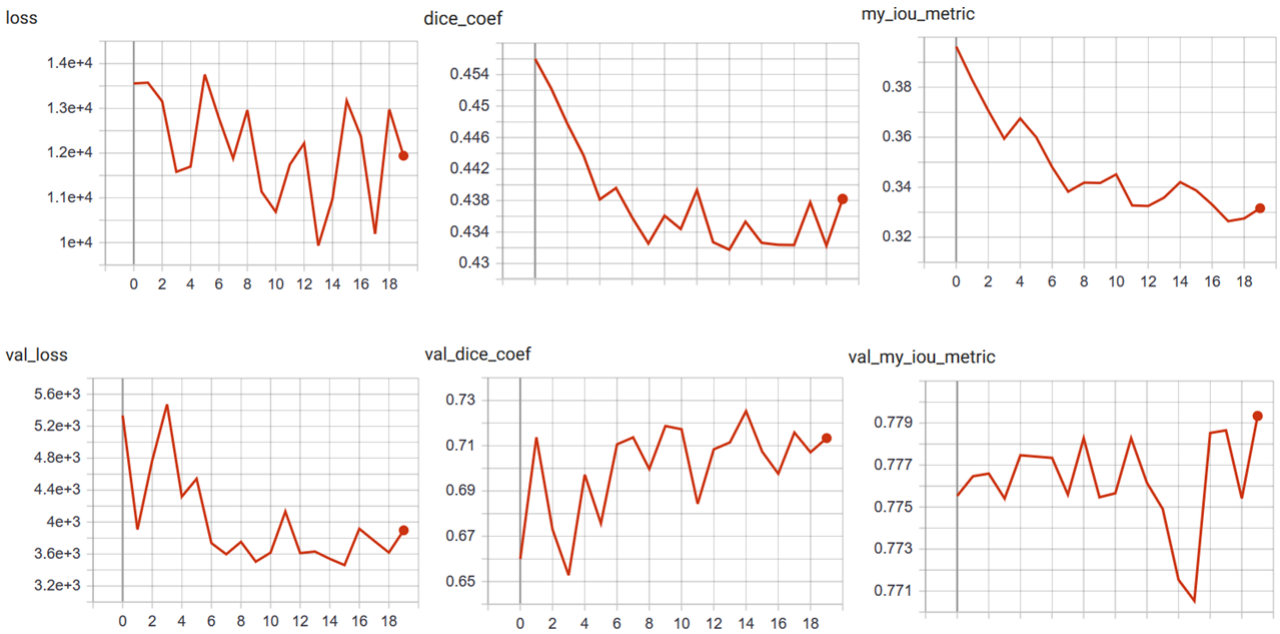


Figure 14: Learning curves of the second training on 768x768 resolution

The model I kept on this last training is the one of the last epoch since it had the best iou. The model had a loss of 3898, a coefficient of 0.7134 and an iou of 0.7793 on the validation set. I tried to re-train other models from this one, but I couldn't get any better.

5 Personal organization

I did this competition during my school holidays. I entered the competition a few days after it started and worked on it until the end except for a week in early August. My work was divided into two phases. A first two-month phase which consisted in working on the architecture of my model and then a second training phase. This split comes from the fact that I was only able to access my Google Cloud credits from July 22nd.

The first phase consisted of reading scientific papers, notebooks and discussions on the competition forum. It is also in this phase that I developed most of the code. I was able to train my models only on low resolution (256x256) on google collab. I was operating by one-week sprints. Go on this link https://trello.com/b/pzJikM3q/siim-acr-challenge to have access to my Trello. As for working hours, I worked from 9am to 12pm and then from 2pm to 5pm from Monday to Friday.

In the second phase, I hardly changed my U-Net architecture. I only modified the training hyperparameters in order to make my model converge. This phase was quite tedious since it required a lot of method and patience. My work was not as intense as in the first phase since it depended a lot on the training times which were quite long, especially on the high resolutions.

6 Acknowledgements

I would like to thank the Society for Imaging Informatics in Medicine (SIIM) and Kaggle for organizing this competition, which allowed me to learn so much. I would also like to thank Google for providing me with GCP credits that have been essential to me throughout this competition.

I also wanted to thank some competitors: Iafoss for making the dataset available directly on Kaggle. Heng CherKeng whose contribution to the community have been very valuable to me. And especially Siddhartha which I was particularly inspired by, especially on the architecture of my model as well as on the code in general.

7 Conclusion

I am particularly proud to have been able to participate in this competition and to have won a bronze medal for my first Kaggle competition. It made me realize how passionate I am about the AI field of work. I look forward participating in other competitions in the future and, I hope, collect more medals.

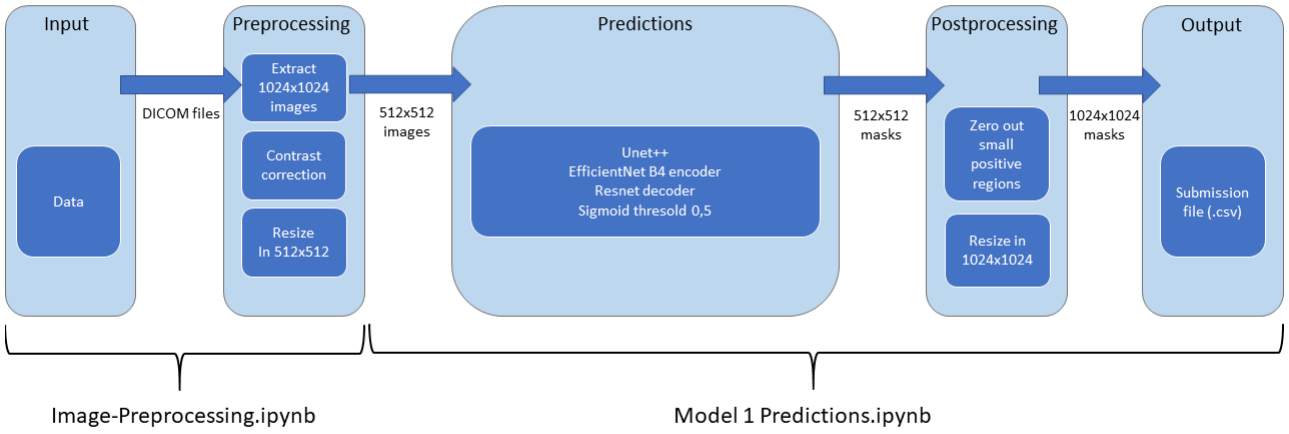


Figure 15: Flowchart

Summary of my model architecture

- Unet ++ Simplified with encoder: EfficientNet B4 & Resnet decoder
- Input shape 512x512
- Leaky Relu activations
- TTA with horizontal flip
- Zero out small predictions
- Threshold 0.5 for sigmoid output

References

[1] Arthur Ouaknine. **Review of Deep Learning Algorithms for Image Semantic Segmentation**. https://medium.com/arthur_ouaknine/review-of-deep-learning-algorithms-for-image-semantic-segmentation-509a600f7b57

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. **U-Net: Convolutional Networks for Biomedical Image Segmentation**. Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick. **Mask R-CNN**. Facebook AI Research.

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam. **Rethinking Atrous Convolution for Semantic Image Segmentation**. Google Inc.

[5] Mingxing Tan, Quoc V. Le. **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks**. Google Brain, Mountain View, CA.

[6] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. **UNet++: A Nested U-Net Architecture for Medical Image Segmentation**. Arizona State University.

[7] Yuxin Wu, Kaiming He. **Group Normalization**. Facebook AI Research.

[8] Lars Nieradzik. **Losses for Image Segmentation**. https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

[9] Tsung-Yi, Lin Priya, Goyal Ross, Girshick Kaiming, He Piotr, Dollar. **Focal Loss for Dense Object Detection**. Facebook AI Research.

[10] Maxim Berman Amal Rannen Triki Matthew B. Blaschko. **The Lovasz-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks**. Dept. ESAT, Center for Processing Speech and Images KU Leuven, Belgium.

[11] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, Andrew Gordon Wilson. **AveragingWeights Leads to Wider Optima and Better Generalization**. Cornell University, Higher School of Economics, Samsung-HSE Laboratory, Samsung AI Center in Moscow, Lomonosov Moscow State University