ISMI: CHEST CT FISSURE INTEGRITY

Eireen Westland, Tom Janssen Groesbeek, Sven den Hartog, Pauline Lauron, Denise Klep, Ruben Kluge

ABSTRACT

Predicting fissure completeness is important in determining whether Lung Volume Reduction Surgery for COPD patients is useful. In this paper, we focus on determining fissure completeness in the oblique fissure of the left lung. We train a 3D U-net to predict lung segmentation. Afterwards we calculate the fissure completeness by counting the labels fissure as opposed to non fissure.

1. INTRODUCTION

Certain patients suffering of Chronic Obstructive Pulmonary Disease (COPD) might benefit from a surgical technique named Lung Volume Reduction Surgery (LVRS). LVRS causes the reducing of the lung volume by removing diseased emphysematous lung tissue (1). The patient has trouble with breathing because of an over-inflated lung. By removing this lung tissue the size of the over-inflated lung is reduced, which allows the more functional lung to expand more, providing the patient with more room to breath.

A predictor in determining whether the procedure will be successful is the percentage of fissure completeness. If the fissure completeness percentage is too low, the procedure is likely to fail. Mainly, as an incomplete fissure might prevent the reduction of the over-inflated lung by allowing air through. Knowing the fissure completeness can therefore avoid unnecessary medical interventions for the patient.

However, to manually determine if the fissure is indeed complete is a very demanding task. We therefore propose the use of a 3D U-net to automatically segment the fissure complete and incomplete regions in a CT scan. These segmentations are then used to immediately compute the exact percentage of fissure completeness. In order to keep the task manageable, this paper only focuses on the segmentation and classification of the fissure in the left lung lobe.

The remainder of this paper is structured as follows: the next section will provide additional information on related work which inspired us to work with the 3D U-net. Next we go into more detail on our exact methods by explaining, among other things, what data we used to train and test our network, what network design we picked and how we defined the final classification task. This section is directly followed by a section on our experiments and corresponding results. We conclude this report with a discussion of our results and methods.

Thanks to Saurabh Jain and Bram van Ginneken for supervision.

2. RELATED WORK

For our project we were inspired by previous work on fully convolutional neural networks used for image segmentation. Long et al. (2) reinterpret former classification nets as fully convolutional networks by turning the fully connected layers into convolutions with kernels that cover their entire input regions. This way the now fully convolutional network (FCN) can take input of any size and returns a corresponding classification map. In their work, the AlexNet, VGG and GoogleNet architectures are turned into FCNs. These are then also augmented to be able to perform dense prediction with in-network upsampling and a voxel-wise loss. The different networks are trained and validated on data from the PASCAL VOC 2011 segmentation challenge, where the mean pixel intersection over union (IU) is used as evaluation metric. Validating on this data resulted in a promising 56.0 mean IU score for the FCN-VGG net.

Similar work is performed by Ronneberger et al. (3), who modified and extended the FCN in order for it to be able to work with few training images while obtaining precise segmentations. The modifications include supplementing the contracting network with successive layers that include upsampling operators instead of pooling operators. These layers thus increase the output resolution. Moreover, high resolution features originating from the contracting path are passed alongside upsampled output. Based on this information, the convolutional layers in the successive expansive path can learn to produce more precise output. Consequentially, the expansive path is very symmetric to the contracting path which results in a network with a u-shaped design. Hence, they named it the U-Net. Working with U-Net, Ronneberger et al. won the ISBI cell tracking challenge of 2015.

Both the FCN and U-Net architectures described are designed to produce segmentation of 2D images. Unfortunately, we are working with 3D CT scans, which would mean that we would have to make segmentation predictions per CT layer and be able to combine these predictions. Luckily, Cicek et al. (4) introduced a 3D U-Net that is able to segment volumetric images. Their network is tested in two use cases, namely one in which just a few input image slices are annotated by the user and one in which sparsely annotations are already present. As accuracy measure, the Intersection over Union (IoU) is used to compare ground truth slices with the predicted 3D volume. In the semi-automated case study (user annotates slices) the performance of the 3D U-Net without batch normalization was 0.842 and with batch normalization was 0.863. Showing that with just a few annotations, their network can produce accurate 3D segmentations. They even compared their 3D U-net to the performance of a 2D U-Net on the fully automated use case. With this experiment, resulting in a IoU of 0.842 for the 3D U-Net against a IoU of 0.547 for the 2D U-Net, it is clear that 3D U-Net outperforms the 2D U-Net on segmenting volumetric images.

Therefore, we decided to implement the 3D U-Net with similar settings like Cicek et al. A more detailed explanation of the network architecture and different parameter settings is provided in section 3.

3. METHOD

For the challenge, we need to classify fissure completeness into three discrete classes:

- Fissure completeness: $< 80\%$
- Fissure completeness: 80% - 95%
- Fissure completeness: $> 95\%$

3.1. Data

The training set regroups CT scans from 100 patients. This set is composed as follows:

- Raw image of the lung from the axial view (Fig. 1)
- Lung mask (Fig. 2)
- Fissure mask (Fig. 3)

For the fissure mask, there are 3 possibilities of label per voxel: background, incomplete fissure and complete fissure. Also given is a file where the exact percentages of completeness and the corresponding label can be found. We found out that this completeness percentage is calculated by:

$$\frac{N_{complete}}{N_{complete} + N_{incomplete}} \tag{1}$$

where $N_{complete}$ is the number of voxels labeled as complete fissure and $N_{incomplete}$ the number of voxels labeled as incomplete fissure. Our hypothesis is that we could accurately

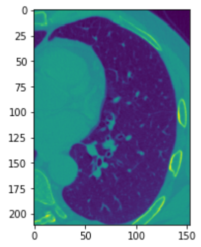
Fig. 1. Raw Slice of image

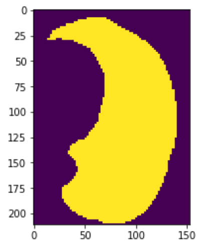
Fig. 2. Lung mask

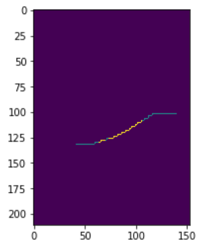
Fig. 3. Fissure mask

reconstruct the fissure mask from the image and calculate the completeness from this predicted fissure mask.

The test set is composed by CT scans from 129 patients. This time, only raw images and lung masks are provided.

3.2. The Task

At first glance, this seems a three-class classification task into discrete labels. However, the training set also contains a separate CSV file with the exact fissure completeness percentages (which are continuous values). Therefore, we decided to break down the task as follows:

1. Segmentation task
2. Calculate continuous completeness labels from segmentation map voxel labels, which are:
   (a) background
   (b) complete fissure
   (c) incomplete fissure
3. Classify continuous completeness labels into the three classes

The conversion from three discrete classes to continuous percentages made for a smoother transition between the segmentation map voxel labels and the final three classes, because the final classes are discretized bins of percentage ranges.

This step could, however, also be replaced with a supervised learning task: using the pre-existing completeness percentages from the training set as target information, the predicted segmentation maps could be classified into the three final classes, therefore negating the second step and directly converting the predicted segmentations into the final three classes.

3.3. Patch and Batch Generator

One of the challenges with this data set is that the images do not have the same size. Furthermore, the images are too large to be processed by a neural network in one go. To solve this problem we divided each image in a number of patches which are fixed-size parts of the complete image.

3.3.1. Patch Generator

The patches we feed to our 3D U-net are generated on-the-fly. This means that instead of composing one static patch set in one go, we build a simple patch generator function that is provided with an image array, the center location of the patch and patch dimensions. From these inputs it generates just one 3D patch and returns this. This way if we had to generate different patches or augment them, we could simple alter the patch generator function without having to generate an entirely new patch set.

Because each image has more patches with only background voxels than with fissure voxels, patches are labeled for their occurrence of voxel labels. The label depends on whether the patch contains only background or also fissure complete or fissure incomplete voxels. Incomplete fissure labels will overrule complete fissure labels because those occur less often and both overrule background.

3.3.2. Batch Generator

This batch generator provides patches of images with corresponding fissure mask patches. The challenge here is that the data is imbalanced, a lot of batches with background are generated but few with the fissure (complete or incomplete). To avoid that, each generation of batches is composed by a third of the background, a third of complete fissure and a third of incomplete fissure. We generated patch with a random location. But we also divided the image into 8 parts to make sure that the model see every part of the image.

Originally, we set out to combine uniform and random sampling of both background and fissure patches. This was done by writing a custom batch generator that would also divide the fissure parts in equal parts as to make sure that the entire fissure was sampled. The sampling would still happen at random in each new fissure part. This custom batch generator also incorporated a per-batch balancing, where each batch contained a balanced ratio of background, complete fissure and incomplete fissure patches.

The problem with this approach was that we had to work with large patch size to be able to feed them to the U-Net. To work with these large patch sizes caused us to establish certain boundaries between which the patches could be sampled. However, these boundaries were so restricting, that not many fissure parts were left, making it difficult to separate those parts in many equally sized chunks to perform some sort of uniform sampling.

Eventually, we did not succeed to incorporate this custom batch generator with our 3d U-Net which forced us to implement a Keras generator. One that did work, but lacked several of our envisioned sampling tactics, as well as lacked batch-balaning; the overall amount of samples was balanced, but the picking per batch was random. This generator uses random oversampling from the imblearn package (5) to balance the occurrence of patch types. For the validation set we did not use any sampling method.

3.4. 3D U-Net

To create fissure mask predictions from CT images we trained a neural network on patches of images to predict patches of the fissure mask. The neural network is a 3D U-Net architecture based on this paper (4), which has 5 261 189 parameters. It is composed of a downward path that increasingly analyses and abstracts the input information, and an upward path that synthesizes a segmentation map of the abstracted information. Skip connections are added to reintroduce the spatial feature details into the upward stream that have been lost in the abstraction due to convolution and sub-sampling.

Each layer contains two 3*3*3 convolutions each followed by a ReLU activation, and then a 2*2*2 max pooling with strides of two in each dimension. After each convolution, and before each ReLU, a batch normalization is applied. All convolution and pooling operations were applied validly, i.e. without padding. This was a decision made to avoid polluting our input signal with redundant zeros.

The up-convolutions as posed by Cicek et al. (4) turned out to be meant as a 3D up-sampling. In the original 2D U-Net paper (3), the up-convolution was defined as an up-sampling followed by 2*2 convolution. However, this was not defined in this manner in the 3D U-Net paper (4), and dimension-wise following up the up-sampling by a 2*2*2 convolution did not work. Therefore, we opted for using only a 3D up-sampling. However, another possibility would have been to use a transposed convolution. The skip connections (in green in the figure 4) adds back feature details lost through max-pooling and convolutions. These features are cropped to be able to be concatenated to layers in the upward path, because of the slight size change due to the greater amount of convolutional steps.

The last layer is a 1*1*1 convolution which reduces the number of output to the number of labels which is five in our case. This 1*1*1 convolution ensures that the output is a segmentation map, which five possible voxel labels; background, fissure, non-fissure, and two labels that we did not end up using but were planned for more segmentation labels (e.g. upper and lower lung lobe).

The input patches that went into 3D U-Net were of shape 132*132*116 in voxels. We decided to keep to the patch size originally used Cicek et al. (4), because their U-Net architecture does not work well for just any input size. The output returned by U-Net is always of smaller size than the input; for input patches of 132*132*116, the output would be 44*44*28. So an input patch size that is too small would not make it through the entire U-Net structure, and for some input patch sizes, the U-Net structure does not quite work because the max pooling reduces an odd-numbered patch size to a decimal-point value instead of rounded integers. Max pooling in keras deals with this by cropping the superfluous row of voxels. Therefore, we decided to stick to 132*132*116 to minimize information loss.

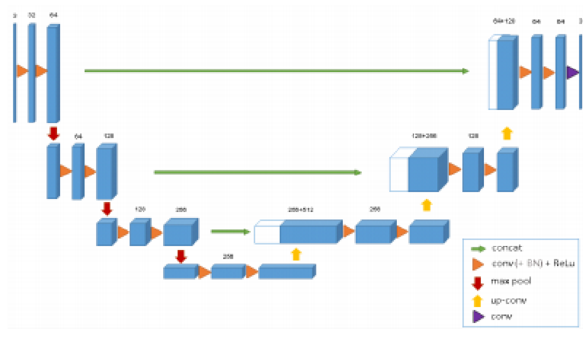

Fig. 4. The 3D u-net architecture. Blue boxes represent feature maps. The number of channels is denoted above each feature map.

3.5. Loss and Scoring function

During training the loss is calculated as weighted categorical cross entropy. This is suited to our problem as it is categorical (our network can predict five different labels), and can be weighted to help our network deal with the imbalanced labels in our data set. The part of the lung that is fissure is very small, and the part that has to be labeled as ”missing fissure” even lower. Therefore an unweighted loss function could lead to the network labeling every voxel as background, since it would score relatively well that way.

The most feasible weights implementation was to have a pre-initialized set of weights, rather than a dynamic one. Background lung labels were to be weighted by estimation of a lot less important than the (complete/incomplete) fissure voxels. Since incomplete fissure voxels occur even less than complete fissure voxels, these should also be weighted higher than the complete ones. The remaining two labels do not actually occur in the true labels, which makes their weights irrelevant because they will never be used.

Originally we planned to adapt the weights of the loss function to each individual patch. This would mean that a patch with an almost complete fissure would weigh mistakes in missing fissures a little higher than patches with a lot of incomplete fissure in them. The advantage of weighing these individual patches is that the weights are more accurate and generate consistent results. For instance a patch without any fissures, would score lower than a patch with mostly fissure, when both are labeled 100% correct. This inconsistency would be resolved when adapting the weights per patch. This functionality failed to work when implemented, which is why we ended up using set weights.

In our first version of the model (Network 20-06) we naively used 0.001, 0.4 and 0.6 as weights for background, complete fissure and incomplete fissure. In our second model (Network 24-06) we used the inverse mean of the pixel frequencies of all the images in the training set as weights for the loss function; we believed the weights to be inversely proportional to the label occurrence, as a misclassification should be weighted heavier the less frequently a label occurs. These weights were 1.0015, 752.33, 5001.4 for background, complete fissure and incomplete fissure respectively.

As a scoring function we used the dice coefficient. Unlike the loss function, this scoring function does not take into account the different frequencies of the labels.

4. EXPERIMENTS

4.1. Training

To be able to check intermediate performance of our network, we decide to split the data into a training set (90%) and a validation set (10%). In order to do so, we made use of the python library sklearn and its ’Stratified ShuffleSplit cross-validator’. Given some data, this will return stratified randomized folds. But the folds are made such that they preserve the percentage of sample for each class (the three fissure completeness classes in our case).

We worked with patches of the input image instead of providing the network with the entire image at once. This also means that no matter the original image size, the input for our network were all of size 132*132*116 (patch size). Because of the different convolution and max pooling layers, we ended up with an output of size 44*44*28 with 5 output channels. However, only 3 of these channels are related to the actual labels and 2 were kept unused. This is because the original fissure mask had 3 different label values, namely 0, 2, and 4. By making use of 5 output channels, the network could immediately assign the correct label without the need of any custom label mapping function. Currently, it learns to ignore the unused labels. The final model is trained for 3 epochs with 10000 steps per epoch and a batch size of 4. With this setting one epoch took approximately 15 hours.

Table 1. Training results

Network	Dice	Val. Dice	Loss	Val. Loss
20-06 (3 epochs)	0.7731	0.8644	$1.364e^{-3}$	$5.981e^{-4}$
24-06 (1 epoch)	0.7576	0.8099	4.347	2.07

Eventually the final loss function weights converged to 0.001 for the background label, 0.6 for the incomplete fissure, and 0.4 for the complete fissure. At the end of the training, the dice score on the training set was 0.7731, on the validation set it was 0.8644.

We also trained a model with other weights given to each label, but only for 1 epoch due to lack of time. For the loss function, the following coefficient are applied: 1.0015 for the background, 752.33 for the incomplete fissure and 5001.4 for the complete fissure. At the end of the training, the dice score on the training set was 0.7576. These results can be found in table 1. If we would have had more time to train our network this would probably have outperformed the other model.

4.2. Shift and stitch

We noticed that the image predicted by our network has a lower resolution then the input image. This is due to the pooling operations. We applied the network multiple times to the same image but shifted by a number of voxels for every dimension to obtain a full resolution image. Because the output patches are smaller than the input patches there will be a border of voxels with no predictions left around the image. For simplicity we just assumed those voxels to be background since they are likely not part of the lung.

4.3. Predictions

The initial network that was trained for 3 epochs with weights 0.001, 0.4 and 0.6 for background, fissure complete and fissure incomplete labels, predicted too many fissure complete voxels on other areas in the lung. This caused the output to look like Figure 5 while it should actually look like Figure 6. The fissure line can be seen in the predicted fissure mask but there are also a lot of false positives in general. Generating predictions from a trained model took 1.5 - 2 hours.

5. RESULTS

We created a submission to grand challenge by generating the complete fissure masks using our trained 3D U-net model. For each predicted mask we calculated the completeness of the fissure with Equation 1. This completeness was then binned in one of the three discrete classes. Our first submission gives us the kappa score of -0.0047. A negative score implies that there is no agreement at all, or that the calculated agreement seems worse than chance level. Most images were

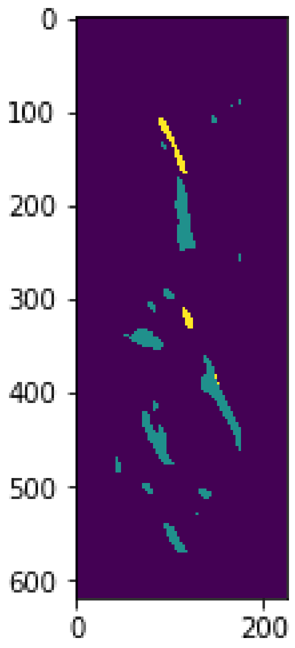

Fig. 5. Predicted fissure mask from the first version of our 3D U-net.

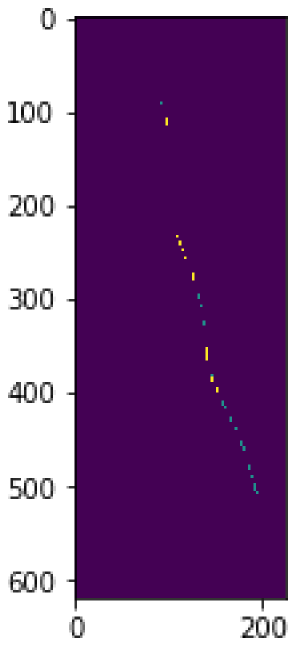

Fig. 6. Actual fissure mask retrieved from the raw data.

labeled "Fissure completeness $>$ 95%", only some "Fissure completeness: 80% - 95%" and none "Fissure completeness: $<$ 80%". For our second submission we trained our network with different loss weights. The kappa score for this submission was 0.0 and most of the images were labeled as "Fissure completeness: $<$ 80%". A kappa score of zero implies that our model performed at around chance level.

6. DISCUSSION

Our results are not yet very satisfying, yet we can still conclude a few things. Firstly we see that during training our network 24-06 scores about as good after one epoch, as our 20-06 network does after three epochs. It would be interesting to see what happens when this network is trained longer. Better results could also be achieved by tweaking the loss function weights some more. One of the possible reasons for the failure in the fissure segmentation is that the weight of the background could be too low so a lot of background is detected as incomplete fissure.

The predicted fissure masks are far from perfect, so there is still lots of opportunity for improvement. These improvements vary between some additional preprocessing steps to our final classification method. First of all the input images are not normalized. The raw input images differed a lot in their brightness, which is most likely due to different CT scan settings or the use of different machines. Normalization is a method of dealing with these differences, but it was not considered for this project. Also, we did not consider any additional augmentations of the input patches. Mainly as we did not have the exact understanding of the different parts of the lung to be certain if transformations like mirroring or flipping would be wise. But these transformations could have helped us create more fissure complete and incomplete data, which was now very unbalanced in comparison to data on the rest of the lung.

Instead of segmenting the actual fissure, another approach that we could have considered is the separation of the upper and lower lobe. By separating these lobes, we can determine where the border of the fissures lie. This could give us additional information about where the area of the (non) existing fissure should reside. Although upper and lower lobe can be hard to differentiate, it is more important to find the fissures for our final completeness calculation. Hopefully this would give the network a better understanding of the structure of a lung.

A possible improvement for U-Net could be to use 3D transposed convolution instead of a regular 3D up-sampling as used now. This would introduce an extra convolutional operation that might abstract more information. However, this would also involve introducing zero-padding to the input signal in order to end up with a larger image after the up-convolution. Given more time, we could test different methods of up-convolution and investigate which works best.

As for classifying the segmentation maps into the final three classes, a more sophisticated viable approach would have been to implement a convolutional neural network for the classification task, rather than first calculate completeness percentages and then classify into either of three classes using these percentages. This approach could have worked by training on the segmentation maps as returned by U-Net as input, and have the predicted three-class labels as output or use the pre-existing completeness percentages from the training set as target data. Instead of having it use a categorical cross-entropy loss, we could have used the kappa-score to more smoothly integrate it with the loss used for the test labels in the submission. The challenge with this approach is that the segmentation maps returned by the U-net are not ground truth and are of different sizes. The former would hopefully be learned by the convolutional neural network while the latter could be fixed by using a global average pooling layer after the convolutional layers.

Another setback was that we did not get our network up and running until the very last couple of weeks. Unfortunately, in these few weeks we had some trouble getting access to the external cluster on which we could run our experiments. This probably due to other teams reserving more than the advised 2 nodes some time in the last weekend.

7. CODE

All code and notebooks used for this project can be found on Github:
https://github.com/SvenDH/ISMI-Fissure-Detection

References

[1] F. J. Martinez, J. K. Stoller, and H. Hollingsworth, “Lung volume reduction surgery in copd,” 2018.

[2] Jonathan Long, Evan Shelhamer, and Trevor Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[4] Özgün Cicek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger, “3d u-net: learning dense volumetric segmentation from sparse annotation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2016, pp. 424–432.

[5] “Imblearn,” http://contrib.scikit-learn.org/imbalanced-learn/stable/api.html.