Deep learning for lung nodule malignancy prediction Project course FMAN40

Jonatan Kronander - elt15jkr
Lunds Tekniska Högskola

April 26, 2019

1 Introduction

Deep learning has been a hot topic in image analysis for a few years now. Due to high performance GPU:s and scientific progress in deep learning, it have been used to get very good results in different fields within image analysis. This report will use deep learning, especially convolutional neural networks, to predict lung nodule malignancy from CT scans.

A lung nodule is a round growth (around 1.2 inch) in the lung that can be cancerous (malignant) or noncancerous (nonmalignant). Patients with a malignant lung nodule must get further medical examinations and if necessary remove the nodule. Image recognition and classification can make a physician more efficient and accurate when making a diagnose. [1]

The data set that has been used to train, validate and test the supervised classification model was collected from Kevin Mader on Kaggle. Mader, who is a ETH Zürich lecturer, has segmented interesting parts from whole CT scans that comes from the Kaggle Data Science Bowl 2017. [2] This project use fastai librar, built upon pytorch, to implement a deep learning model. [3] The model was run and train on a paperspace server with NVIDIA Quadro P4000 with 1792 CUDA cores.

This report is written in the course FMAN40 (Project in mathematics) at Lunds Tekniska Högskola. The purpose of the course is to develop the ability to independently understand the mathematical model and the goal is a report and presentation on a chosen topic. [4] This report is written to a person who has an interest in deep learning and have some insight in image analysis.

2 Theory

2.1 Data set

The chosen data set contained of 6691 segmented CT scans together with tags of malignant or nonmalignant (CVC file with image number and tag). Number of images tagged malignant was 4165 and nonmalignant 2526. All images had size 64x64 pixels. The data set was randomly scrabbled and then divided into train, validation and test folders (80,10,10) with malignant and nonmalignant nodules in different sub-folders. It is hard to predict, without a medical background, if there are any malignancy in the image or not, as can be seen in Figure 1 and 2.

In theory we want to classify and run or the model on a 3D data set but this was not possible since the data set only contained of single CT scans with no tagged patient. This will probably lower the result somewhat from an ideal 3D classification.

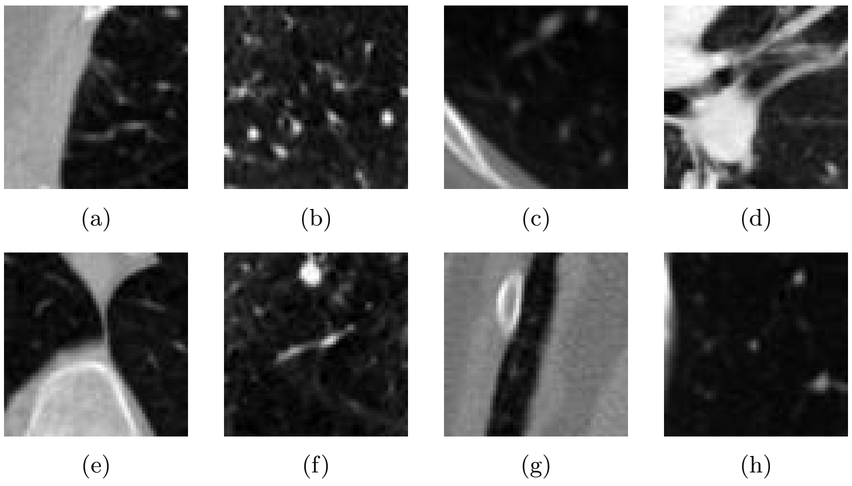
Figure 1: Nonmalignant lung nodules.

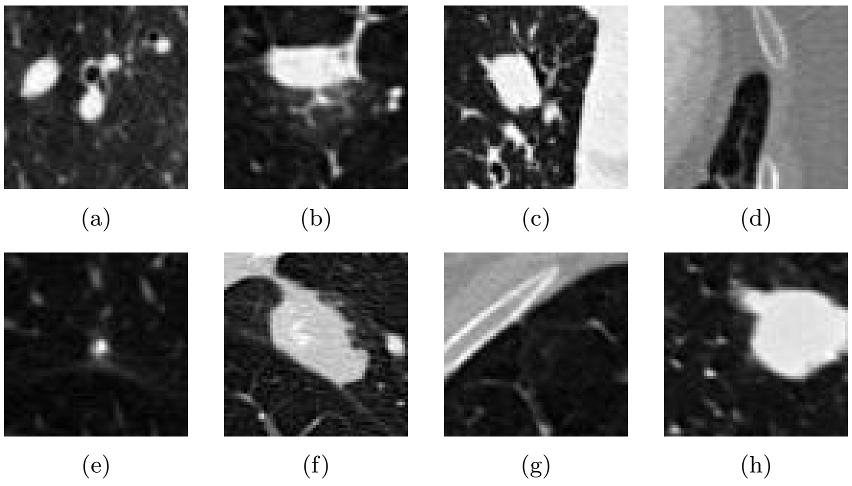
Figure 2: Malignant lung nodules.

2.2 Fastai

There are several libraries that could be used when implementing a deep neural network. The chosen library for this project, fastai, uses pytorch with creative heuristics solution to get state of the art results. Fastai was chosen because its built to be intuitive and easy to use but still reliable and gives very good result. It is created by Jeremy Howard who also lectures on fastai and deep learning. This reports and its code is written after taking fastai lectures and following Howards techniques for getting a good result.

2.3 Convolutional Neural Network (CNN)

A CNN is built up by series of convolution layers with different kernels, pooling layers and fully connected layers. The convolutional and pooling layers serve as feature extraction and the fully connected layers together with a softmax function serve classification for the image. [5] Figure 3 shows how a example CNN can be built up.

By using a pretrained CNN architecture it is not necessary to build a CNN from scratch (which will likely to perform bad) and train it. A pretrained architecture, also called covnet, is trained on a large image net, for example ImageNet [8] which is a image database that contains more than 14 billion different images. The database is mostly containing ”common” images such as images on computers, dogs, flowers etc. This becomes a problem when trying to implement the model on a non common data set such as lung nodules due to wrongly weighted CNN. Therefore its a good idea to try train all layers and not just the last layers.

The architecture that was chosen for this project was resnet34, resnet50, resnet101 and resnet152. The higher the resnet number the more parameters/weights there is in the pretrained model. Many weights can make training the network more complicated and take more time. A quick compression was made with the different resnet models in Figures 4, 5, 6 and 7. The Figures shows training of the last layers with the same learning rate (0.001). We can see in the Figures that resnet50 performed similarly to resnet101 and resnet152 even if resnet50 has less weights. [7] Resnet50 had also better accuracy than resnet34 and therefore resnet50 was chosen as the pre-trained model.

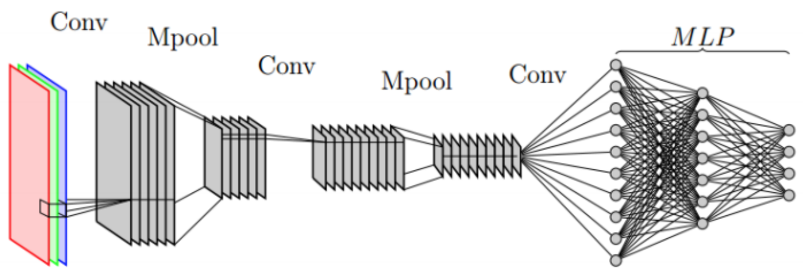

Figure 3: Convolutional neural network, Figure from Magnus Oskarssons lecture on deep learning [6].

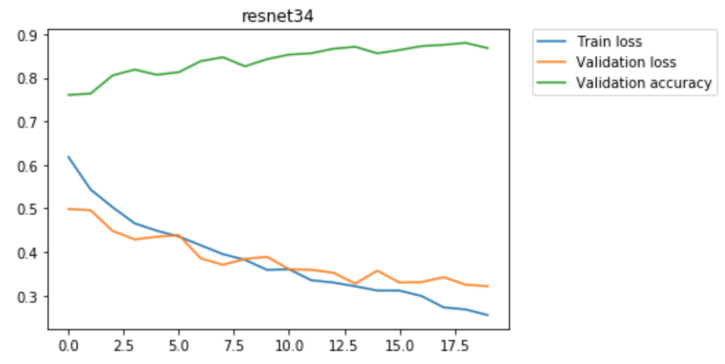

Figure 4: Resnet34

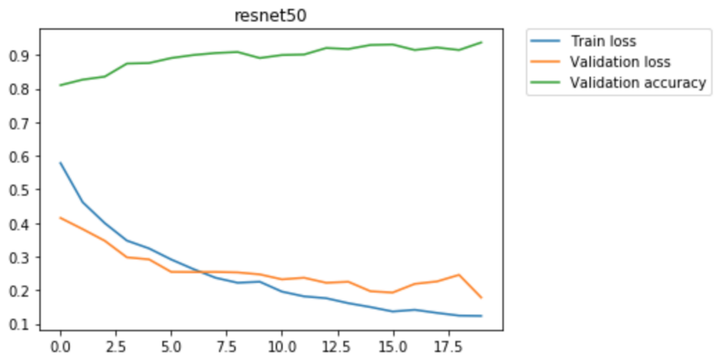

Figure 5: Training with Resnet50.

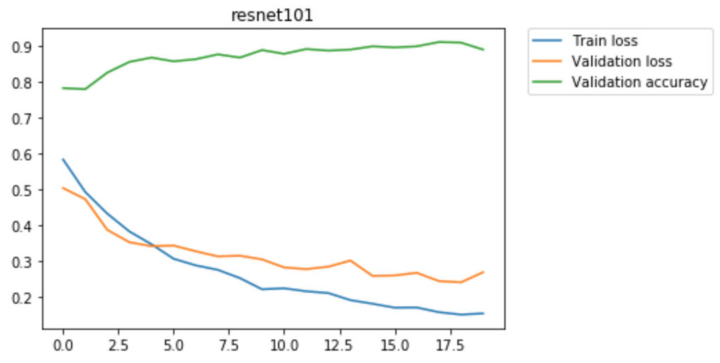

Figure 6: Training with Resnet101.

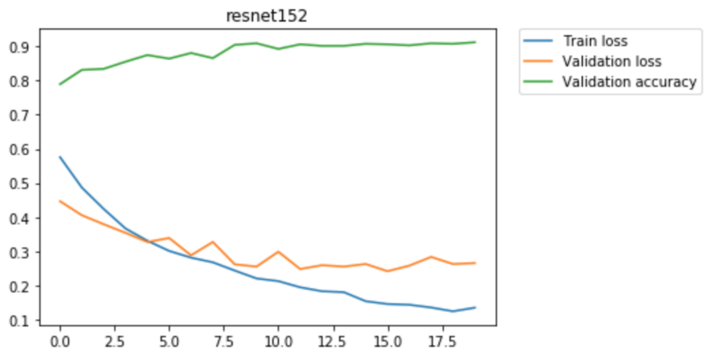

Figure 7: Training with Resnet152.

2.4 Learning rate

To minimize the loss function fastai uses stochastic gradient descent (adam optimizer). The optimizer are built to find a local minimum of the loss function by changing the weights in the network. Learning rate controls how much the parameters are changing in regards to the loss function. A low learning rate will take small steps to reach a local minimum but at the same time it will take very long time to get the right weights. Too high learning rate will take bigger steps and might miss a local minimum which will result i bad weights and a inaccurate model. [3]

To find a good learning rate fastai have a implemented function called "*learn.lr_find()*" which finds loss for each learning rate. This way the loss function against learning rate can be plotted, see Figure 8. When choosing the learning rate we want as high learning rate as possible but where the loss gradient is as high as possible and in Figure 8, this is at 10^{-2}. The learning rate should not be chosen at the validation loss function minimum because cosine annealing will be implemented.

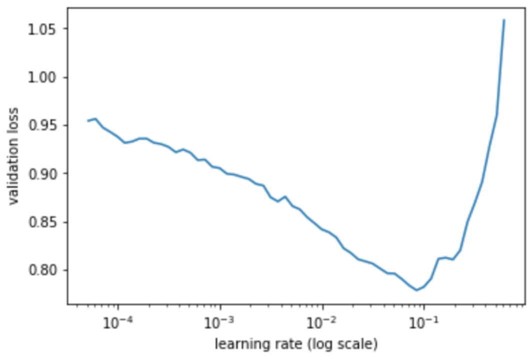

Figure 8: learn.lr_find(); learn.sched.plot()

As the model are trained after every epoch (one iteration over the training set) the model should come closer to the minimum solution. To prevent the model from overshooting and miss the minimum the learning rate should decrease after each epoch. This could be done by simply lower the learning rate after a group of epochs or by cosine annealing. In Figure 9 we can see how cosine annealing are lowering the learning rate after each epoch in a cosine rate.

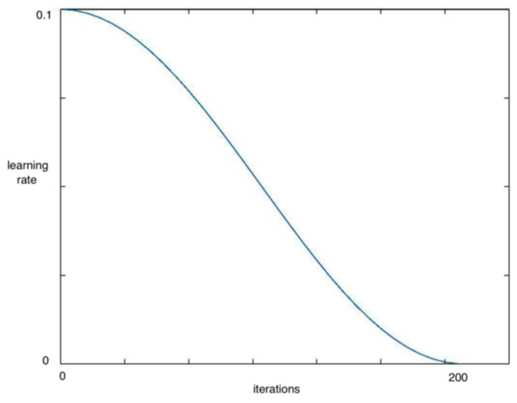

Figure 9: Learning rate and iterations.

When training the whole resnet network its a good idea to change the learning rate for different layers in the network. The networks higher (firsts) layers is consisting of feature extraction of different basic shapes such as edges and corners. These layers are typically well trained and need no additional training. The further (deeper) in the network features extractions gets more complicated such as finding human eyes or characters. As our data do not contain specific complicated structures as in ImageNet we want to train the specific layers more than the well trained shape detection layers. Therefore different learning rates for different parts of the network can be a good idea.

2.5 Data augmentation

The first try to train the last layers gave a small overfit shown in Figure 10. Overfitting occur when model is over trained to fit the specific training data and this will lead to validation loss being greater than train loss. This will lead to more specific model and not a general model. To prevent overfitting the model needs to be trained with more data. This can be done by either getting more training data or by data augmentation on the training data.

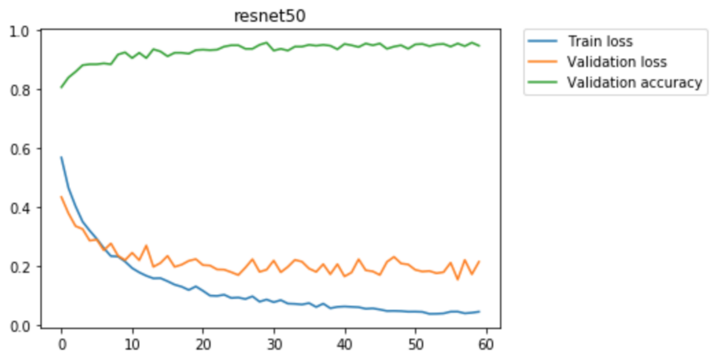

Figure 10: Last layers of resnet50 model trained with training data.

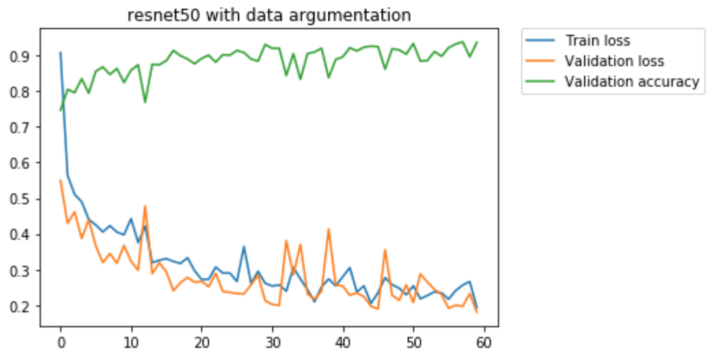

Figure 11: Last layers of resnet50 model trained with data augmentation training data.

Data augmentation is a way to prevent overfit by creating a new data set from the original data set. The original data is randomly augmentation, meaning randomly flipped, zoomed, rotated and mirrored etc, and then added to the original data set. This way we create several different images from the same image. In Figure 12 a lung nodule from training data is augmentation with random rotate and randomly mirrored. This way we get more training data and we can prevent overfitting. Figure 11 shows training and validation loss with the same data as Figure 10 but with data augmentation. As seen there is no overfit after 60 iterations training. Data augmentation is not always good for accuracy as particular images may have set features. For example size can be a features that implies malignancy or not and to zoom the training data can lead to size feature to be lost in zoom. Therefore the only data augmentation that where used in the model was randomly rotating the training data.

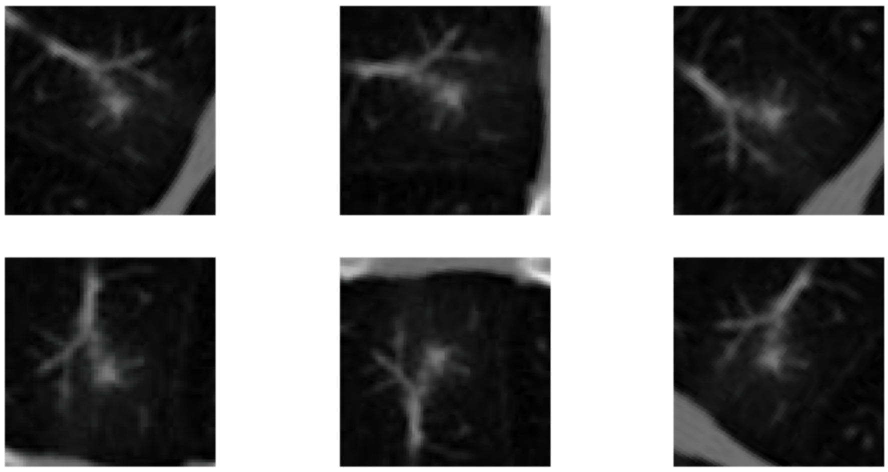

Figure 12: Data argumentation of a lung nodule.

Another method for getting better accuracy is to predict with test time augmentation (TTA). This will take the image for prediction and augment it (chop, zoom etc) and run the augmented images though the network. TTA will then take the average prediction of these images and prove that as an prediction. This will generally provide a couple of percents better accuracy.

The images in this data set was also pretty similar in contrast, sharpness, color etc which was a advantage for the test set accuracy. If we would have predicted CT scans from a different hospital, with a different CT machine, the images might look slightly different. That could make the predictions from the model worse as it was not trained for that type of images. This could might be solved with data augmentation or a different training set.

2.6 Class Activation Maps (CAM)

A interesting function when using pytorch is CAM which make it possible to see where the last layers is activated the most. By using this function we can find which sections of the image is most important when the model makes a prediction. Unfortunately this did not work with resnet50 which was the chosen model. To show this feature a small model was trained with resnet34. Figure 13 shows where the last layers did its predictions by overlaying a heat map over the left image, making the area green if there is a malignant lungnodule.

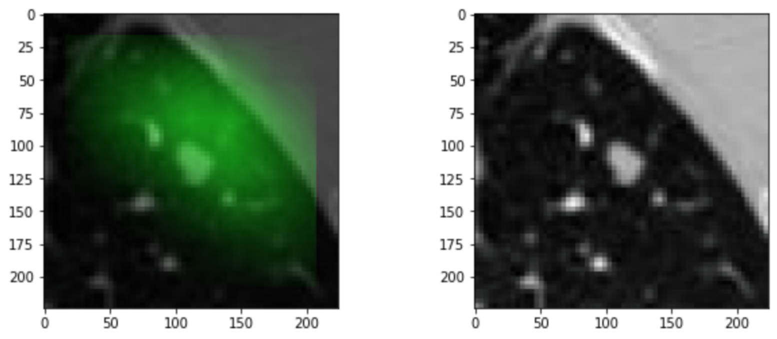

Figure 13: Malignant lung nodule and CAM layer.

3 Result

First result, when training the last layers without data augmentation, the model reached 0.93 accuracy on validation and test set after TTA. Seeing Figure 14 the model starts overfitting and data augmentation is needed to get better results. The model was trained in 1:01 minutes.

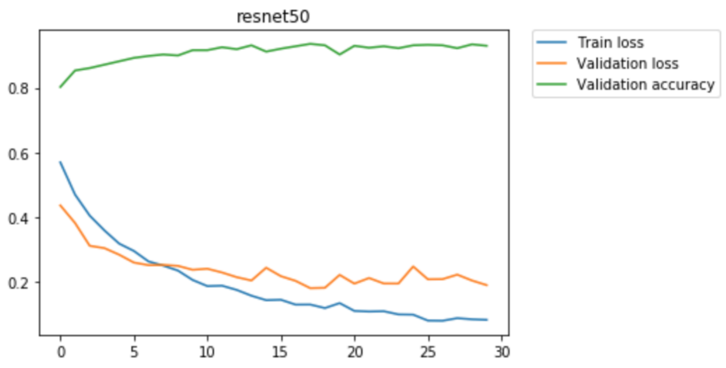

Figure 14: First result training the last layers in the resnet architecture.

With data augmentation and cosine annealing results in Figure 15 was trained. 200 epochs took 4.37 minutes to train and reached a **accuracy of 0.95**, predicted with TTA. In Figures 16, 17, 18, 19 and 20 different predictions are plotted. Figure 21 shows a confusion matrix of the predictions.

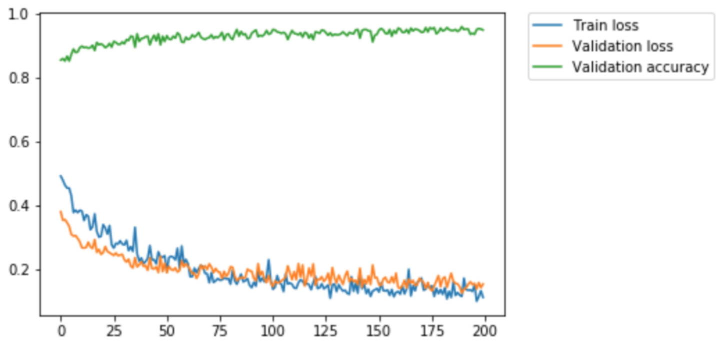

Figure 15: Second result training the last layers in the resnet architecture with data augmentation.

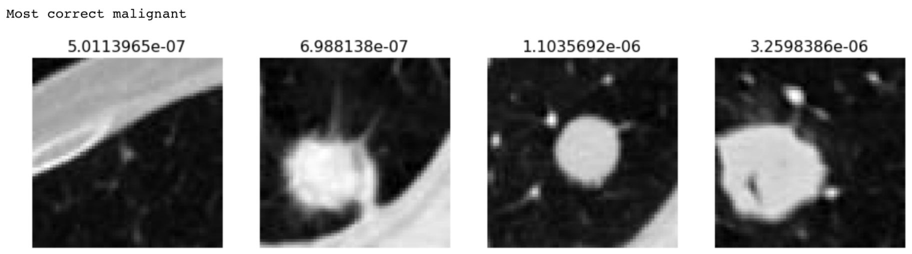

Figure 16: Most correct predicted malignant.

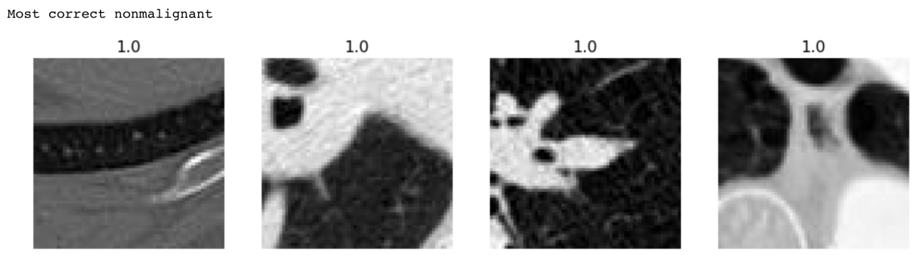

Figure 17: Most correct predicted nonmalignant.

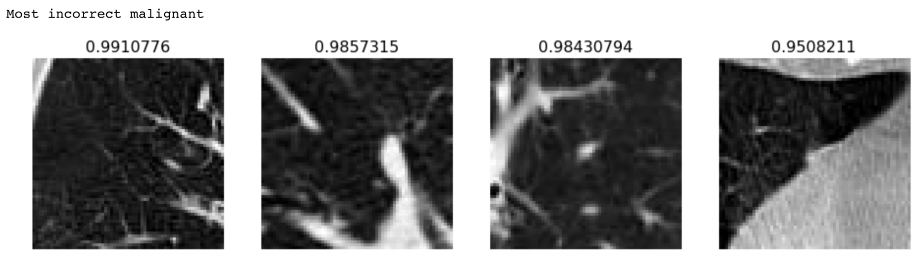

Figure 18: Most incorrect predicted malignant.

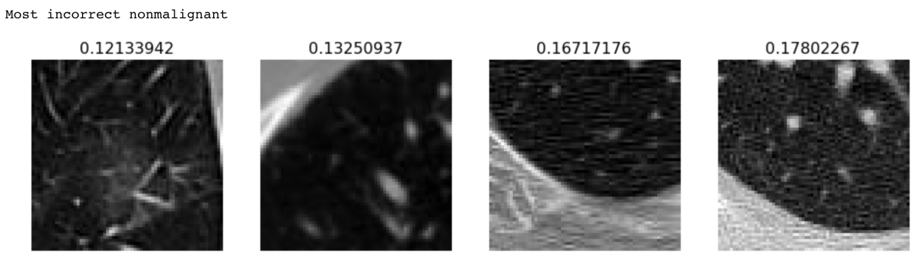

Figure 19: Most incorrect predicted nonmalignant.

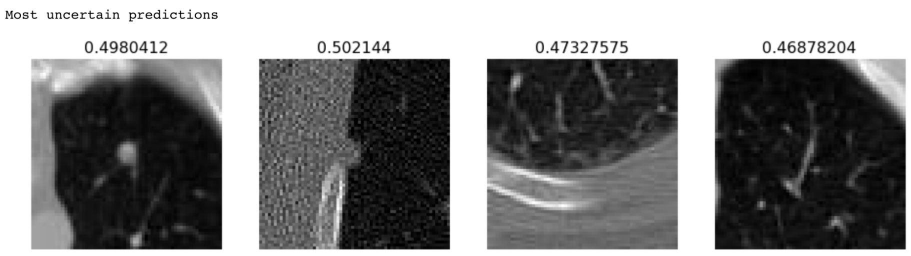

Figure 20: Most uncertain predictions.

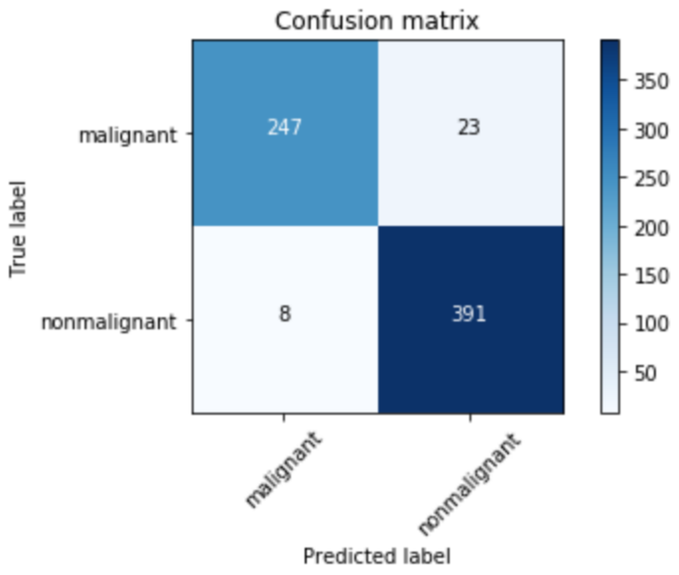

Figure 21: Confusion matrix over predicted images.

Trying to train the whole resnet did not turn out to give good results. The training is shown in Figure 22. Firstly the last layers was trained then the whole resnet architecture was trained. As can be seen in the Figure after 100 epochs, where the whole model started to be trained, the train and validation loss sky rocketed. This training took about one hour and this training was tried several times with different learning rates with the same result. What might had happened was that the resnet was to hard to train and needed to much more time and computational power than given. The training might turn out good if the model was set to train more. Lack of time on server prevented that. Results on the validation and test set where 0.81 which makes the model both less accurate and slower than original resnet.

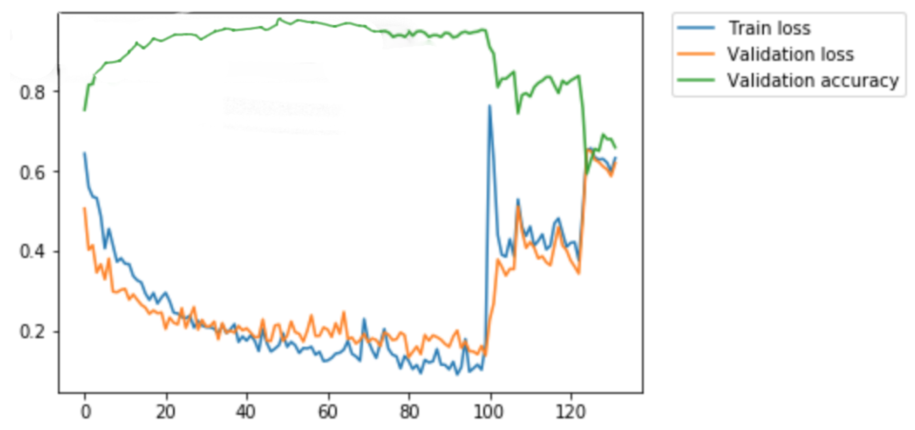

Figure 22: Model trained with data augmentation first last layers then whole model.

4 Discussion

The first thing to question is if 0.95 accuracy is a good result? We can compare to randomly picking which would give 0.5 accuracy and say that 0.95 is better. Kevin Madder, the data set creator, has two notebooks on kaggle [2]. When he ran the data set through a CNN model he reaches 0.68 accuracy and when he used a Capsulenet he reaches 0.59 accuracy. These results are old and deep learning have got more sophisticated. So with these example in mind 0.95 is a good result. Keep in mind that the dataset is not very good either. The result is firstly thanks to fastais smart solutions and state of the art library. The question of how good the result is compared to a doctor would be very interesting to look at but out of scope for this project.

The predictions in figure 16, 17, 18, 19 and 20 is interesting to look at. Both most correct predictions in figure 16 and 17 we can clearly see where the lung nodule is located. The most incorrect predictions, figure 18 and 19, is pretty hard to evaluate since we do not know exactly what a malignant nodule can look like. Here we probably want to ask a doctor and look if he/she can explain these images more. This is also the case for the most uncertain images.

5 Appendix

See attached files for code.

References

[1] American Thoracic Society, 2016, "What is a Lung Nodule?", viewed 2018-12-17, https://www.thoracic.org/patients/patient-resources/resources/lung-nodules-online.pdf

[2] Kevin Mader, 2018, "Lung Nodule Malignancy", Kaggle, viewed 2018-12-17, https://www.kaggle.com/kmader/lungnodemalignancy

[3] Fastai, 2018, viewed 2018-12-17, https://docs.fast.ai/

[4] Matematik LTH, 2018, "Projekt i matematik/tillämpad matematik", viewed 2018-12-17, http://www.maths.lth.se/matematiklth/personal/magnuso/kurser/projekt/index.php

[5] Arden Dertat, 2017, "Applied Deep Learning - Part 4: Convolutional Neural Networks", viewed 2018-12-18, https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

[6] Magnus Oskarsson, 2018, "Lecture 7 - Deep learning", viewed 2018-12-18, $http://www.ctr.maths.lu.se/media11/FMAN20/2018/F07_DeepLearning.pdf$

[7] Sergey Zagoruyko, Nikos Komodakis, 2016, "Wide Residual Networks", viewed 2018-12-19, https://arxiv.org/abs/1605.07146

[8] Stanford Vision Lab, 2018, "ImageNet", Stanford University, Princeton University, viewed 2018-12-19, http://www.image-net.org/