Diagnosing Pneumothorax in X-Ray Images Using Deep Learning

Felix Steinberger Eriksson
felix.s.eriksson@gmail.com

under the direction of
M.Sc. Anna Broms
M.Sc. Joar Bagge
Department of Mathematics
KTH Royal Institute of Technology

Research Academy for Young Scientists
July 10, 2019

Abstract

The manual segmentation of medical images is a time-intensive process. The detection and segmentation of illnesses like pneumothorax might be accomplished using Convolutional Neural Networks. This study examines the use of neural networks for medical image segmentation. Specifically, it investigates the use of a model based on the uXception architecture and a loss function based on binary cross-entropy and log-dice loss. After training on a data set from Kaggle, it achieved a total Dice coefficient of 37.85% and a mean Dice coefficient of 30.03%. Learning rate reduction, data augmentation and image downsampling are discussed as potential reasons for the comparably low accuracy. It is concluded that the model is not reliable enough for use in medical triaging, but that the concept of neural networks for medical image segmentation is possible.

Acknowledgements

First and foremost, i would like to thank my mentors Anna Broms and Joar Bagge for their assistance and guidance during this project. Similarly, I would like to thank Anna-Karin Tornberg for her helpful explanations and insights. The greatest of thanks also to Hugo Berg, with whom it was a pleasure to cooperate on this study. Also, I would like to extend my thanks to Grant Sanderson, who, in providing beautiful illustrations and explanations, first awakened my interest for the topic of machine learning. Finally, this project would not have been possible without the dedicated work of the Rays - for excellence Organisers or without the contributions of Kjell- och Märta Beijers Stiftelse and Europaskolan Strängnäs.

Contents

1 Introduction

Diseases often show visible symptoms that can be determined by medical professionals. Many of these symptoms become clearly discernible through medical imaging techniques like Magnetic Resonance Imaging (MRI) or Computed Tomography (CT). In some cases, the affliction may be difficult to spot even for an expert [1], e.g. pneumothorax. In such instances the detection of the disease often takes longer and is more costly. Thus, automated methods for recognising and triaging pneumothoraces would be useful in many clinical scenarios. Such techniques would enable large scale screenings not possible today because of the time-intensity of screening manually. One popular way to implement such an automated method is by using neural networks created for image recognition.

1.1 Medical Background

An abnormal accumulation of air in the so-called pleural space between the chest wall and the lung is called a pneumothorax. It is sometimes informally referred to as a lung collapse, but this term might also be used to describe other lung diseases.

Pneumothoraces may arise spontaneously or traumatically (meaning from chest trauma) [2]. When occurring spontaneously, the pneumothoraces are referred to either as primary or secondary, the latter meaning that there is a connection to a previously existing lung disease. Pneumothoraces of smaller size will normally resolve without treatment, but pneumothoraces occupying more space may require surgery. In any case, monitoring the build-up, either in order to remove it or to ensure that it dissipates, is of importance. Screening for a pneumothorax is usually done by simple radiography, i.e. using techniques like X-ray imaging which creates a two-dimensional image of the patient's internals. Producing a three-dimensional image CT scan is only recommended when differentiating between a pneumothorax and similar conditions like bullous lung disease, or when the radiograph for some reason (e.g. an obstructive emphysema) does not show enough detail [1]. This study will focus only on radiography images.

1.2 Image Segmentation

Image segmentation denotes the partitioning of an image into two or more parts, each of which contains one or more coherent entities. This could include tasks such as outlining people or animals in an image or allowing self-driving cars to recognise other vehicles, and also deciding which pixels of a medical image contain an illness, for example pneumothorax.

In image segmentation, there are two main categories of methods: Semantic segmentation and instance segmentation. They share some features and differ in other regards. Semantic segmentation is usually seen as easier than instance segmentation, mainly because instance segmentation traditionally uses multiple instances of semantic segmentation to construct a solution.

Semantic segmentation refers to the labelling of pixels into different data classes, whereas an instance segmentation algorithm would separate different entities in a picture, however usually only by first applying some object detection algorithm to the image in order to find the bounding boxes of the unique entities in the image. After that, it would typically apply a smaller version of a semantic segmentation algorithm to each of the specified bounding boxes. This is possible given that the bounding box algorithm provides each of the unique objects with a separate bounding box. The semantic segmentation algorithms vary, but some of the most common and state of the art algorithms are described more closely in Section 1.4.

1.2.1 Medical Image Segmentation

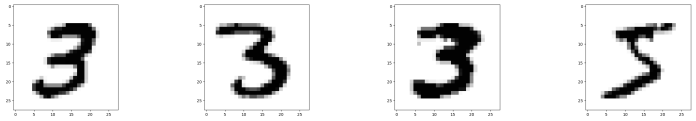
Despite being almost exclusively of a technical nature, medical scanning technologies such as MRI scans, X-ray CT scans, radiography and ultrasound imaging often require a human medical professional to classify the data, and the field of machine learning for medical image classification is still new. For most diseases, there are symptoms visible on a scan that can be segmented, either by instance or semantically. Many of the newer algorithms like U-net [3] and U-net++ [4] were developed specifically for medical image segmentation.

There is, however, still a need for new medical image segmentation algorithms, despite the large number already available.

In one implementation of computerized medical image classification, Rahman et al. [5] use support vector machines (SVMs) to classify the diverse ImageCLEFmed [6] database consisting of 11 000 radiographs grouped into 116 categories. SVMs use a learning algorithm to find a hyperplane separating the different sets of data points into two distinct categories. Therefore, SVMs are traditionally used in binary classification problems where one of only two categories may apply, unlike the multi-category problems often encountered in medical imaging problems. A straightforward approach to applying SVMs on multi-category tasks like medical imaging is to calculate the separating planes between all possible combinations of data categories. If there are n different categories, this results in $\frac{n(n-1)}{2}$ unique separating hyperplanes, from which distinct categories can be formed. This usually results in satisfactory results for very distinct categories, but it is relatively computationally expensive. Despite this, the implementation is a first step toward more advanced methods for medical image segmentation.

1.3 Neural Networks

Data sets can often be classified according to their contents. Humans are in general quite good at this type of attribute assignment and classification. Throughout this section, the classification of hand-written digits will be taken as an example. Most people will easily recognize that Figures 1a, 1b and 1c show the same digit. They will also agree that Figure 1d shows a different digit.



(a) (b) (c) (d)

Figure 1: Hand-drawn 28×28-pixel digits from the MNIST database [7].

Describing which pictures correspond to which digit with a general rule (say, in terms of pixel activation patterns), however, is a more daunting task. This task can be tackled by using a neural network and is a problem commonly used to introduce machine learning.

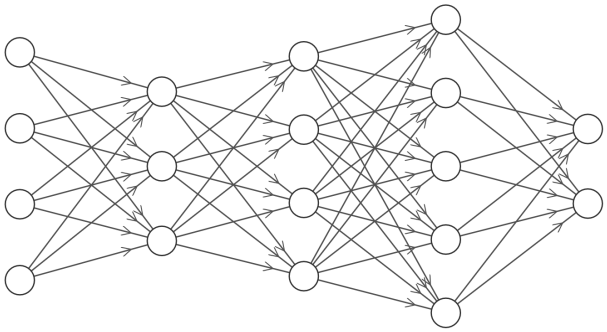


Figure 2: A basic neural network with two outputs. It is said to be fully connected, because all outputs of one layer become inputs to the next layer

A neural network is a structure that is on some level meant to have loose analogies to the structure of a biological brain. Concretely, it is made up of an input layer of neurons that feed some input into middle layers, often called the hidden layers. The hidden layers receiving an input also feed it into the successive layer. The final outputs of these hidden layers are in turn fed into the final layer L. The final layer L gives a number of outputs as illustrated in Figure 2.

Each neuron of one layer takes one or multiple inputs from the layer before, applies a transformation to them and passes them on as an output to the neurons in the next layer. In Figure 2, this is represented by arrows from one neuron to another. The transformation is described by $f(wa + b)$, where the scalar weight w and the scalar bias b are applied to the scalar activation a from a neuron to one in the next layer. In this transformation, one can also see an important step: The application of a so-called activation function $f(x)$, which is generally non-linear. In this instance, x is a scalar input. One commonly used activation function is the sigmoid (also known as the logistic function), defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. Other common activation functions include the hyperbolic tangent $\tanh(x)$, or some variation of the rectified linear unit function (ReLU), which is defined as

$$\mathrm{ReLU}(x) = \max(0, x).$$

Many activation functions share the convenient property of being easy to differentiate (at least approximately). For example, the derivative of a sigmoid in a point can be expressed as

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

In order to prevent the notation from becoming too complicated, the inputs, outputs, weights and biases between two consecutive layers $l-1$ and l are written in terms of matrices and vectors. We describe the weights using the matrix $\mathrm{W}^{[l]}$, where each row contains the scalar weights for connections between the neurons in the previous layer to one neuron in the current layer. Thus, $\mathrm{W}^{[l]}$ contains as many columns as $l-1$ has neurons; and as many rows as l has neurons. The biases and outputs from the activation function are described by vectors $b^{[l]}$ and $a^{[l]}$ respectively. Here, the i:th elements describe the bias to and activation from the i:th neuron in layer l.

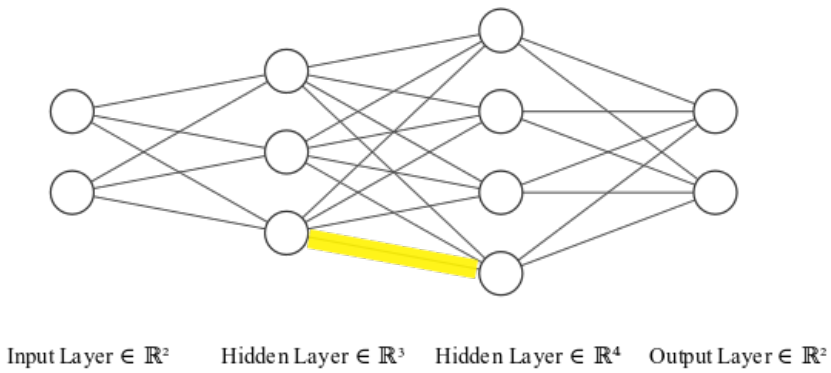


Figure 3: A network with four layers, i.e. $L = 4$. The weight corresponding to the highlighted edge is an element of the weight matrix $\mathrm{W}^{[3]}$.

Because the connections between two layers are described as vectors, we define $f(v)$, in which v is a vector, to signify the componentwise application of $f(x)$. It should be noted that the activation function can be different for different layers, despite sharing the notation. The equation $a^{[l]} = f(W^{[l]}a^{[l-1]} + b^{[l]}) \in \mathbb{R}^{n_l}$ shows the rigorous way of writing the activation vector a for layer l in terms of the weights, biases and activations from layer $l-1$. We define $a^{[0]}$ to be the input vector into the first layer and let n_l be the number of neurons in layer l. We notice that in the expression, $a^{[l-1]}$ (in this case the activation vector

of the previous layer) can similarly be expressed as a function of $a^{[l-2]}$, the activation of layer $l-2$. This gives the equation $a^{[l]} = f(W^{[l]}(f(W^{[l-1]}a^{[l-2]} + b^{[l-1]})) + b^{[l]}) \in \mathbb{R}^{n_l}$. Through repetition of this insight, we find that the entire network may be represented as a function F of the weights and biases of the different layers as well as of $a^{[0]}$, as seen in

$$f(W^{[L]}f(W^{[L-1]}f(\cdots) + b^{[L-1]}) + b^{[l]}) = F(a^{[0]}, W^{[1]}, \cdots, W^{[L]}, b^{[1]}, \cdots, b^{[L]})$$

where $W^{[1]}, \cdots, W^{[L]}$ are the weight matrices and $b^{[1]}, \cdots, b^{[L]}$ are the bias vectors. This insight is critical for understanding how the network actually "learns", something that will be explained in the next section.

1.3.1 Optimising and Gradient Descent

Learning is done by continuously adjusting the weights and biases. To do this, we first define a function that quantifies to what extent the given answer $a^{[L]}$, the activation of the outputs from layer L, differed from the ground truth, i.e. the correct answer for that input. This function, known as the loss function, should return a lower value if the network's given answer is closer to the actual answer and a higher value if the given answer is further away. The used loss function varies, but a common one is the quadratic loss function. It is defined as

$$L = \frac{1}{N}\sum_{i=1}^{N} \left\| y(x^{\{i\}}) - a_i^{[L]} \right\|_2^2$$

for N training data points $x^{\{i\}}$, each associated with a ground truth outcome $y(x^{\{i\}})$, i.e. the outcome we want the system to output for that input. When the given outputs of the network exactly correspond to the expected outputs, the loss function will be 0. As wanted, an adjustment of the weights and biases affecting $a^{[L]}$ such that the network becomes more accurate in its predictions (compared to $y(x^{\{N\}})$) also makes the output of this loss function smaller. One might similarly imagine that a minimization of this loss function through adjustments to weights and biases gives the network an increase in

prediction performance, which makes it more likely to give accurate predictions.

For a network, the loss function can be written as taking a vector $p \in \mathbb{R}^k$ as input, with each of the vector's components represent the weights and biases of the network and k is the total number of weights and biases. Changing the elements of this vector changes the output of the loss function. In order to minimize this loss function, we need to know how it changes with respect to a change in only one of the components, i.e. with respect to a change in only one weight or bias. This leads us toward the field of multivariable calculus, specifically to partial derivatives and gradients.

The concept of gradients in multivariable calculus is similar to the concept of derivatives in single-variable calculus. The gradient $\nabla L(p)$ is the vector of all partial derivatives of a multivariable function. Similarly to how one might use the derivative to find one of the minima of a single-variable function, one can use the gradient to find the change to the elements of p that gives the greatest change to the value of $L(p)$. Higham et al. [8] and Goodfellow et al. [9] derive and explain the formulae used for the computationally efficient calculations of this gradient. The vector p is then updated iteratively according to $p_{i+1} = p_i - \mu \nabla L(p)$, where μ is the *learning rate*. The learning rate is simply a scalar that scales the computed gradient to allow for larger or smaller steps for each iteration. It should be mentioned that μ can be variable, for example by using optimizer algorithms like Adam [10]. After each step, the new gradient is calculated and another step is taken along its direction. Thus, for each step, the value of the loss function decreases towards the closest local minimum.

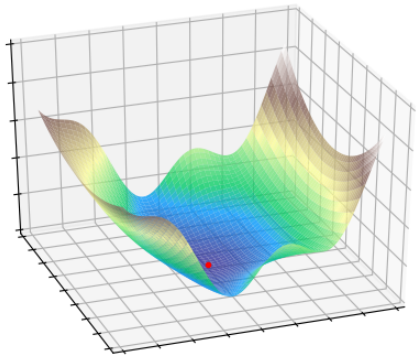


Figure 4: Example landscape in three dimensions. A red spot marks the local minimum.

A simple way to envision this gradient descent is to think about an optimization problem using a function of two variables. We can visualise both the two variables and a function of them in three dimensions. Consider the example landscape in Figure 4. For a given point in Figure 4 (say, starting at one of the peaks), we can see that a change along one direction gives a larger change in function value than a equally sized change along another direction. Thus, in this point, the gradient of the function describing this landscape is pointing in the direction for which, given an equally sized change, the function value would change more. In our visualisation, this is equivalent to a steeper slope.

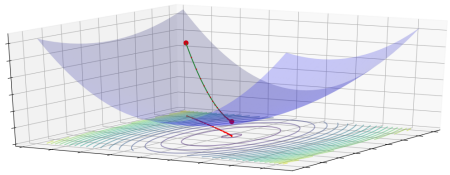


Figure 5: Ball rolling down the steepest slope.

Similar to the red ball in Figure 5 rolling down the steepest slope in a landscape towards the valley, gradient descent is the successive minimization of the loss function by changing the vector p (adjusting the weights and biases of the network) in the direction which results in the largest decrease of the loss function. In theory, because the loss function is an average over all training examples, the gradient should decrease the loss function for all training examples. In practice, however, the gradient is usually approximated. This results in better performance (or efficiency) [11], as calculating the entire gradient for each iteration is computationally intensive.

There are different methods for efficiently approximating the gradient. Many use techniques such as choosing the batch size stochastically or including a weighted version of earlier gradients to enable optimisation as good as or better than if the true gradient had been calculated. Often, multiple of these techniques are joined in an algorithm called an *optimizer*. The methods usually specified by such optimizers are calculated once per

batch. To explain batches, one first needs to explain epochs. When talking about epochs in the context of training a neural network, one means training once on each training example. One usually divides the epoch into a number of batches which are computed iteratively. The batch size denotes the number of images in a batch.

1.3.2 Convolutional Neural Networks

In image recognition, performance can be increased by preserving spatial information and not just contextual information. This is because a large part of the information in an image is spatial. This is not explicitly done in a traditional fully connected neural network described in the introductory section on neural networks, where the input image has to be flattened into an input vector to be processed by the network. Here, 1-dimensional neurons are exchanged for 3-dimensional neurons with height, width and depth [12]. These so-called convolutional layers play a vital role in the operation of a network for image input. Each layer contains filters that scan across the input. During this process, the filter outputs the componentwise multiplication of its values and the input area.

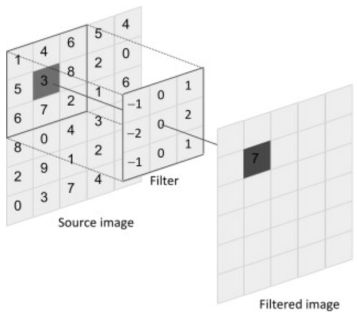


Figure 6: Application of 3×3 filter to one region of an image. [13]

For instance, we show the input, convolution and output of an image where the $3 \times 3 \times 1$ filter is particularly designed for edge detection in the vertical direction in Figure 6. The output is 7, which is the result of the componentwise multiplication of the filter's values with the corresponding pixel values. This filter gives a somewhat scaled difference in pixel values between the two sides of the filter. This difference is naturally greater if the pixel values differ more due to the existence of an edge. It is worth mentioning that in a

Convolutional Neural Network (CNN), these filters are not specifically designed, but are instead created by training the model. Each filter extracts specific features of the input. Thus, using more filters, it is possible to detect more features.

Some of the layers in a CNN are so-called pooling layers, that downsample the image in order to reduce the computational complexity of the layers. These pooling layers use different methods for pooling, but the most common ones are the average pool, outputting the average of the input values, and the max pooling layer, outputting the maximum of the input values. A max pooling layer is visualized in Figure 7.

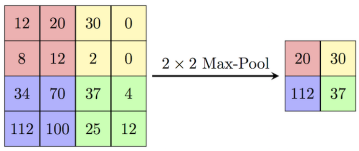


Figure 7: A max pooling layer of size 2×2, outputting the max for each input it scans and downsampling the image.

All in all, a complete CNN has a structure similar to the one pictured in Figure 8.

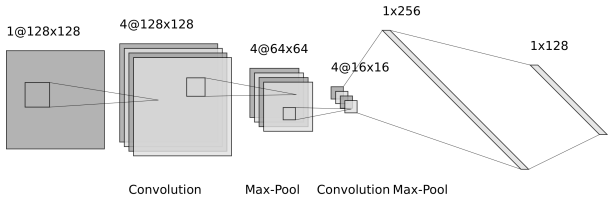


Figure 8: Example of a basic CNN. The first filter is a convolution, increasing the number of features extracted. Next, a pooling layer downsamples the image to 25% of the original size. Afterwards, a convolution is applied again, as well as an additional pooling layer. Then, the image is flattened in order to allow two fully connected layers to classify the image.

There are three main reasons for using CNNs rather than traditional fully connected neural networks for image recognition. As mentioned, information is gained by having pixels that are spatially close to each other as inputs into the same filter. In addition, traditional networks can become relatively computationally expensive with increasing pixel numbers, especially if the images are in colour (i.e. have three dimensions). Also, traditional networks are more prone to so-called overfitting, especially when they grow larger and contain more tunable parameters in order to be able to process large images

[14]. Overfitting refers to a network with too many tunable variables that becomes too finely adjusted to the training data, at the expense of bad performance on images it was not trained on. Thus, it loses its general applicability to similar data.

1.4 Previous Research and Different Architectures

In 2017, He et al. [15] presented Mask R-CNN, an architecture for instance segmentation as an extension of the Faster R-CNN model by Ren et al. [16]. Mask R-CNN, like Faster R-CNN, divides the image that is to be segmented into many different Regions of Interest (RoI), each containing one potential bounding box. On each of these RoIs, classification is performed, features are extracted and the bounding box is fitted. In parallel to the other stages, however, Mask R-CNN also extracts information about where in the image pixels are located using a binary mask with values of True or False for each RoI. This spatial information is later merged with the fitted bounding boxes for more accurate, pixelwise segmentation. The framework is shown to achieve results comparable to other state-of-the-art models like ResNet on the COCO data set [17], containing images of common objects and animals.

Krizhevsky et al. [18] used a Deep Convolutional Neural Network (DCNN) consisting of a few convolutional layers and a few fully connected layers to classify the images of the ImageNet-LSVRC 2010 competition into 1000 different categories with higher top-1 percent and top-5 percent error rates (37.5% and 17%) than previous state-of-the-art methods. The top-n percent error metric is often used for data sets with many different categories and denotes the probability of the ground truth label not being one of the networks n top predicted labels. Being a very deep network with over 60 million tunable parameters, they avoided the high risk of overfitting mainly through the use of data augmentation [19], i.e. changing images without destroying distinguishing features, and through stochastic dropout [20] with a relatively high probability of 0.5. Stochastic dropout refers to randomly setting the activation function of neurons to zero, thereby temporarily disabling them. Techniques like the DCNN provide good evidence for general

medical classification algorithms, considering their proficiency in performing classification among a number of categories.

He et al. [21] also mention the difficulty in training networks that are too deep. They state that many of the leading classification algorithms use deep models, even though empirical evidence often shows that they are notoriously difficult to train. To counter issues with deeper networks, they introduce the concept of residual blocks containing paths that act as identity maps while skipping layers, in order to be remapped to the throughput later on. The authors show on ImageNet [22] and CIFAR-10 [23] among others that residual networks perform better than comparable networks with added depth.

To lessen the issue of features "washing out", i.e. losing their usefulness through successive layers of abstraction in deep CNNs, Huang et al. [24] proposed the DenseNet architecture that connects a layer to all successive layers with a matching feature map size by a short feed-forward path. Through this path, information from previous layers is concatenated with information changed by the convolutional layer. One such block is called a dense block. This strategy is used to counteract overemphasized abstraction by successive convolutions. To preserve the feed-forward nature of the network, dense blocks of different sizes are connected by pooling and transition layers converting between the sizes. The DenseNet structure achieved similar results on the ImageNet [22] classification task as ResNet, despite using only about a third of the number of parameters. This implies the possibility of efficient algorithms for medical classification that are also hardware efficient.

Antoniou et al. [25] state that deep convolutional networks, while they excel at image classification, are less capable at relational reasoning tasks, i.e. tasks concerning the position of objects in relation to one another. They present dilated convolutions, a type of convolutional filter where there is a space between any two neighbouring input pixels of the filter.

For example, dilation 2 seen in Figure 9 is applied only to pixels in odd-numbered rows and columns on a scanned field of 5×5 pixels. The authors apply their network on the

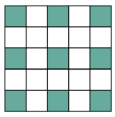


Figure 9: Dilation 2 with a space of 1 pixel between received pixels. Picture from [25]

Sort-of-CLEVR [26] data set and gain results similar to, but less computationally expensive than, the previous state of the art. Because these dilation filters can replace deeper structures requiring more calculations, this again acts as an example of how a hardware efficient algorithm for medical imaging might be implemented.

To accommodate the fact that few large image data sets are available in medical imaging scenarios, Ronneberger et al. [3] introduced the U-net architecture specifically meant to capture sparse information often more abstract than what is visible for humans. The structure consists of a traditional contracting path of simple convolutional and pooling layers, reducing the input size in each layer. This is supplemented by a subsequent multi-channelled "upsampling" path. This upsampling path, in order to preserve spatial information while increasing the resolution, is combined with high resolution features directly from the corresponding contracting layers. This is similar to a traditional auto-encoder network using an encoder to downsample the image, a latent space containing the abstract features of the encoded image and a decoder that reconstructs data from the original image. The main difference between U-net and a traditional auto-encoder is the existence of direct connections between the two paths, visualized with grey arrows in Figure 10. To ensure high performance even when large sets of data are not available, the authors use data augmentation [19] in the form of elastic deformations, rotations, shifts and sizing to allow the network to learn a robustly with respect to these features. Their tests, which include an experiment on interference contrast microscopy images of HeLa cells, show that the network beats the previous state of the art with an Intersection Over Union (IOU) metric of 77.5% compared to 46%. The IOU is defined as $\frac{|X|\cap|Y|}{|X|\cup|Y|}$, $|X|$ being the set of predicted pixels and $|Y|$ being the set of ground truth pixels.

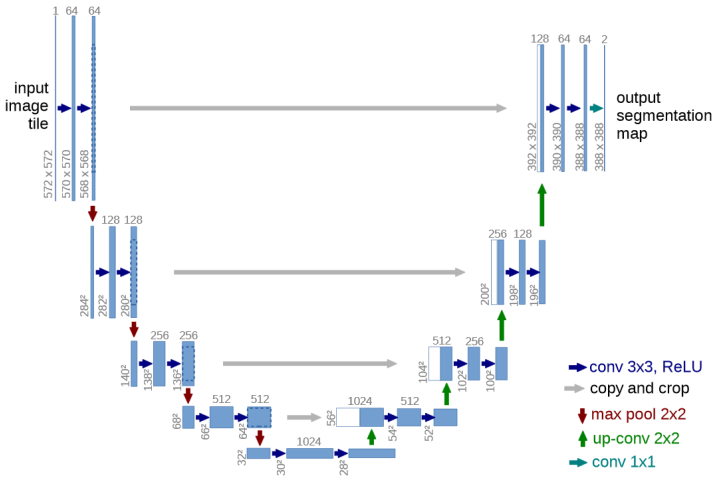


Figure 10: Example schematic of the U-net architecture [3]. A blue box corresponds to a feature map with multiple channels, with number of channels denoted on top and size denoted to the left of the box. A white box corresponds to a copy of a previous feature map, operations are denoted in the legend.

In 2017, Chollet [27] proposed the Xception architecture. It makes use of depthwise separable convolutions to extract abstract features with higher computational efficiency. A depthwise separable convolution splits one traditional three-dimensional convolution into a few smaller two-dimensional convolutions and one pointwise 1×1-convolution, contracting the outputs into an image corresponding to the output of the traditional convolution. This process is generally more computationally efficient than the regular convolution with the same output, since less calculations have to be made. Xception also contains residual connections, previously mentioned in conjunction with residual blocks, which feed outputs directly to a later layer without applying the bypassed filter. These residual connections are visible in Figure 11 as paths bypassing the larger convolutional blocks. Xception outperforms similar networks on the ImageNet [22] database, despite having the same number of tunable parameters, which suggests a more efficient use of parameters.

The uXCeption model, which was proposed by Curiale et al. [28], draws inspiration from both U-net and the Xception model. One of the things uXception makes use of is the Xception architecture as the "backbone" (that is, contracting path) of the traditional U-

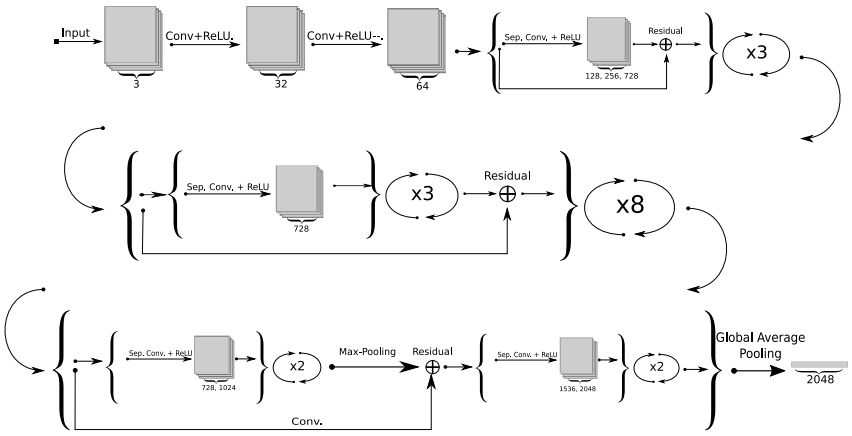


Figure 11: Schematic of the Xception architecture. Residual paths are shown as arrows bypassing convolutional layers.

net, while preserving the connections between the contracting and the expanding paths. The authors show a Dice coefficient of around 0.9 for segmenting MRI images of the myocardium. The Dice coefficient DSC is defined as

$$DSC = \frac{2 \cdot |X \cap Y|}{|X| + |Y|}. \tag{1}$$

1.5 Aim of Study

The aim of this study is to create a neural network that is able to semantically segment a set of chest radiographic images. Specifically, it will classify pixels containing and not containing pneumothorax. This information could then be used to determine the severity and location of the pneumothorax, especially in cases where other methods of detection fail.

2 Method

An implementation of the uXception model [28] was trained on chest X-ray images containing pneumothorax and the corresponding ground truth masks of the images. Subsequently, the Dice coefficient was used to evaluate the performance of the model on a set of validation data.

2.1 Data and Evaluation

The data analysed in the study were provided by Kaggle [29] in cooperation with the Society for Imaging Informatics in Medicine (SIIM) and the American College of Radiology (ACR). The data were provided in two separate sets, one training set and one test set. The training data set contained 10675 separate images and made up a significantly larger part of the total number of images than the test data set of 1377 separate images. For the training data, labels were provided in the form of pixelwise masks with a value of 1 or 0 for each pixel, depending on whether or not the corresponding pixel in the training image contained pneumothorax. However, the original set was highly unbalanced and only 2379 of the annotated images were labelled as containing instances of pneumothorax; the rest contained no pneumothorax. To facilitate the learning of pneumothorax features, the model was trained only on the subset of the total X-ray image data set containing pneumothorax. This may lessen the general applicability of the model but makes training significantly easier.

After acquisition, the training images and masks were downsampled from the original size of 1024×1024 pixels to 128×128 pixels to make the training less computationally intensive. The model was trained on 1802 images. It was subsequently validated on 200 images. Lastly, after training, it was evaluated on the remaining 377 images.

The mask images were provided in a run-length encoded (RLE) format, each mask corresponding to one X-ray image. RLE has many benefits, among them the fact that it is lossless, i.e. saves the value of each pixel, while still noticeably compressing the data volume [30]. However, it adds an extra step to the process of data decoding.

The model was evaluated on the downsampled data using the previously defined Dice coefficient DSC traditionally defined as $DSC = \frac{2 \cdot |X \cap Y|}{|X| + |Y|}$, X being the predicted set of pixels containing pneumothorax and Y being the set of ground truth pixels. In this evaluation, DSC is defined to be 1 when $X = Y = \emptyset$.

If one randomly selects pixels belonging to X with a probability of a and pixels belonging to Y with a probability of b, the expected Dice value is given by $\frac{2ab}{a+b}$. Assuming

$a = b$, i.e. that the network segments the correct number of pixels as containing pneumothorax, this Dice value can be simplified to a, which is equal to the probability of a pixel containing pneumothorax. Thus, the expected Dice value for a model segmenting pixels randomly, will be as high as the proportion of pneumothorax pixels out of total pixels. This proportion has been found empirically to be small.

The evaluation metric used in this study, however, was a version of the Dice coefficient modified specifically to accommodate for non-binary pixel values. The Dice coefficient was calculated according to

$$\frac{2\sum_i y_i \hat{y}_i + \text{Smooth}}{\sum_i y_i + \sum_i \hat{y}_i + \text{Smooth}}. \tag{2}$$

The term Smooth was added so that the Dice coefficient has a value of $\frac{\text{Smooth}}{\text{Smooth}} = 1$ when both the predicted set and the ground truth set are exactly 0. The magnitude of Smooth can be scaled according to how much weight one wants to give to sets that either contain or predict very little pneumothorax. For our model and image size, Smooth was set to be equal to 0.1.

There are two ways of interpreting the Dice coefficient for a set of predictions: using the mean and using the total number of pixels. The total Dice coefficient is calculated by treating the entire validation set like one image. Thus, the coefficient is calculated from the total number of predicted pixels and the total number of ground truth pixels. Conversely, the mean Dice coefficient is the mean of the Dice coefficients calculated individually for each picture.

2.2 Model

The model is an implementation of the uXception model previously described in Section 1.4. Our model makes use of so-called transfer learning and is initialised using weights from previous training on ImageNet [22]. Transfer learning means reusing weights from successful training on other data sets as initial weights in a new application of the same network. The model was built in the Keras [31] framework for machine learning.

2.3 Training

The network was trained on a NVIDIA GTX 1070 GPU using the previously mentioned set of 2002 images for training and the remaining 377 for validation using the Dice coefficient.

The loss function utilized was a combination of binary cross-entropy loss and log-dice loss. For ground truth pixel values y_i belonging to the ground truth mask y and predicted pixel values $\hat{y}_i$ belonging to the predicted mask $\hat{y}$, binary cross-entropy loss $L_{BCE}(y, \hat{y})$ is defined as $L_{BCE}(y, \hat{y}) = -\frac{1}{N}\sum_i y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$, N being the number of pixels in the mask and i ranging through all the pixel values in the mask. The log-dice loss function, on the other hand, is defined as

$$L_{Dice}(y, \hat{y}) = -\log\left(\frac{2\sum_i y_i\hat{y}_i + \text{Smooth}}{\sum_i y_i + \sum_i \hat{y}_i + \text{Smooth}}\right). \tag{3}$$

The total loss function used is simply the sum of L_{BCE} and L_{Dice}. The main reason for using L_{BCE} as a part of the loss function was to provide an incentive for the network to be certain about given outputs. For a general overlap of predicted and ground truth areas, L_{BCE} penalizes predictions close to 0.5 more than predictions closer to 0 or to 1. The log-dice is a loss function based on the Dice coefficient, which gives a maximum of 1 for completely overlapping areas. Thus, the log-dice loss function incentivizes an overlap of predicted pixels and ground truth pixels.

The optimizer used during training was Adam, with an initial learning rate of 10^{-4} [10] Adam is straightforward to implement and uses separate learning rates for different parameters to increase performance. The batch size used was 16. Training was allowed to go on until stopped by an early stopping mechanism triggered if the mean Dice coefficient (the evaluated metric) had not increased for 50 consecutive epochs. The learning rate was also reduced by a factor of 5 whenever the loss function had not decreased for 4 consecutive epochs. In experimental training, different levels of data augmentation were used. In final training, however, no augmentation was used.

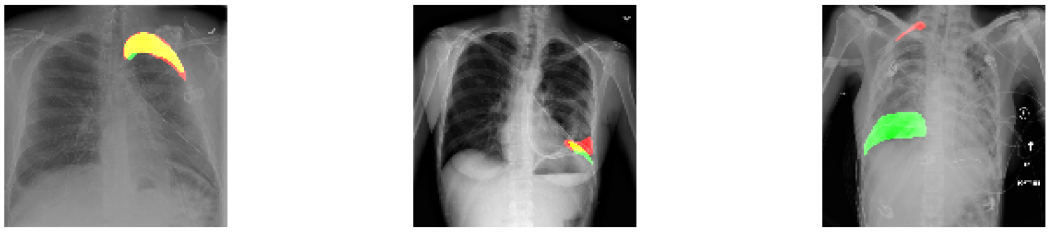
3 Results

After training and validating on 2002 labelled images containing pneumothorax, the network was evaluated on 377 labelled test images not used during training. The evaluation metrics used were the mean and total Dice coefficients, displayed in Table 1.

Table 1: The total and mean Dice coefficients

Metric	Value
Total Dice coefficient	37.85%
Mean Dice coefficient	30.03%

This model was found to achieve a total Dice coefficient



(a) A reliable prediction with a Dice coefficient of 88.14%.

(b) A representative prediction with a Dice coefficient of 36.46%.

(c) An incorrect prediction with a Dice coefficient of 0.03%.

Figure 12: Visualization of three predictions

In Figure 12, the ground truth mask is marked in red, the predicted mask is marked in green and areas where both masks overlap are shown in yellow. In some cases, like in Figure 12a, there is substantial overlap between the ground truth and predicted masks, which leads to a high Dice coefficient. In other cases, like in Figure 12b, the network has found the general location of the pneumothorax, but fails at segmenting it. Finally, there are also cases like in Figure 12c, where the network fails at predicting where the pneumothorax is located.

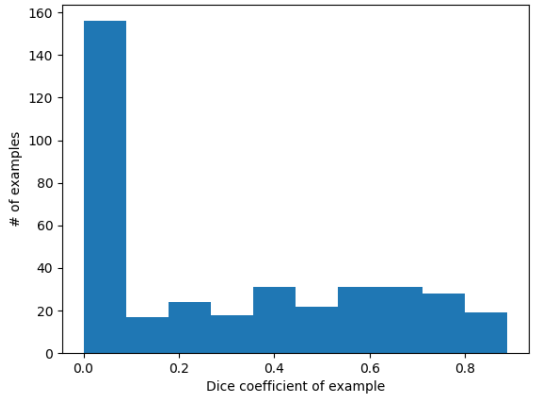
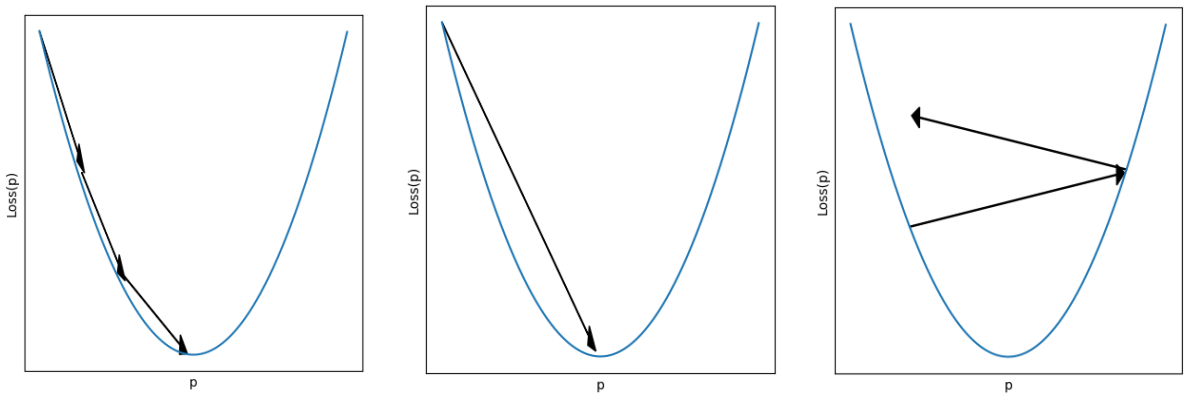


Figure 13: A histogram of the predictions given on the validation set by the model.

Figure 13 shows the distribution of the network's predictions on the validation set. Around 50% of the cases are predictions with Dice coefficients close to zero. For the rest of the predictions, the histogram is more evenly distributed between different Dice coefficients.

4 Discussion



(a) A learning rate that is too low, resulting in inefficient convergence.

(b) An optimal learning rate, finding the local minimum in one step.

(c) A learning rate that is too high, diverging from the minimum.

Figure 14: Some learning rates visualised in a two-dimensional landscape.

Choosing the right learning rate μ for the task at hand is a delicate balance. If the learning rate is too low, the loss function will decrease toward the nearest local minimum like in

Figure 14a. This will, however, take more iterations than might be necessary if one had chosen the ideal learning rate like in Figure 14b. There will also not be a possibility to "overshoot" the local minimum, to instead decrease toward another local or global minimum with a lower loss value. On the other hand, if the learning rate is too high, the loss function may fail to reach a local minimum, instead oscillating around it. This may be why the loss function failed to decrease noticeably below a certain point in our test. If the learning rate is more than twice as high as the optimal learning rate, the loss function may even diverge like in Figure 14c. These situations are visualized in a simplified, two-dimensional landscape in Figure 14.

Learning rate reduction partly counteracts the effects of a learning rate that is too high, while keeping the benefits. It allows the learning rate to start out high and find an adequate local minimum. If the loss function does not decrease for a number of epochs, perhaps because it is oscillating around a minimum, the learning rate is reduced and the loss function is allowed to converge. The patience of our early stopping mechanism was 50 epochs. This value was chosen to be significantly higher than 4, which was the patience of our learning rate reduction mechanism. Thus, the model is allowed many epochs to find a path that decreases the loss function further. Empirically, this did not work as well as expected. The reasons for this are uncertain, but may entail the learning rate decreasing fast enough to prevent parameters from changing much at all.

During conception, the model was tested with different levels of data augmentation. In many cases, this drastically impaired performance. At first glance, this seems counter-intuitive: The main case for augmenting data is to create a larger set of data for the network to train on. In theory, this should allow for better performance through a larger pool of features to extract. Indications like these also conflict with the empirical data of Krizhevsky et al. [18] who successfully made extensive use of augmentation in their Dense Convolutional Neural Network. It may be that medical images are not as easy to augment without distorting defining features. The number of possible augmentations is also smaller. For example, because all medical X-ray images come oriented in a very simi-

lar way, all rotational augmentation would add information that one does not necessarily want the network to learn. The degree of augmentation may also play a role. For example, when cropping the X-ray images, it is possible that the area containing pneumothorax was cropped out. This would mean that the augmented image did not contain pneumothorax, which would lead to worse performance when using a model trained exclusively on images containing pneumothorax. This might imply that the network used was not optimal for use in conjunction with data augmentation.

Another factor that may contribute to somewhat disappointing results is the image downsampling performed in conjunction with the data acquisition. Downsampling from 1024×1024 to 128×128 pixels reduces the total number of pixels by a factor of 64. Doing this, one risks losing a lot of details. These details may well be critical in the segmentation of pneumothorax. Of course, different methods for image downsampling vary in both efficiency and the amount of details preserved. Both bilinear and bi-cubic interpolation were tested during training. Bi-cubic interpolation, while preserving more detail, was found to allow for low negative pixel values in the masks. This led to minor complications at first, as they were expected to be in the interval $[0, 1]$. This could, of course, be resolved by clipping the values to the interval $[0, 1]$, but doing so risks losing detail.

The uXception model uses depthwise separable convolutions. Since they reduce the number of individual calculations needed for a traditional convolution, they effectively reduce the computational complexity of the model. There are, however, also disadvantages to this decrease. Because depthwise convolutions reduce the total number of trainable parameters in the model, it might fail to learn if the layers do not contain enough total filters. However, considering the depth of the Xception backbone, this seems unlikely.

Despite a large part of the predictions having Dice coefficients close to 0%, it is noticeable when regarding the histogram in Figure 13 that a number of the predictions also have Dice coefficients that are neither close to 0% nor close to 100%. This seems to indicate that a substantial part of the predictions find the general location of the ground

truth mask, but fail to predict the exact shape of the ground truth mask.

4.1 Further Studies

This study has one chief limitation: computational power. The training of a neural network takes time. Because of the limited time frame, this study does not aim to present a network perfectly capable of classifying cases of pneumothorax. It merely aims to act as a proof of concept of the use of similar architectures for medical image classification in general, using pneumothorax classification as an example.

Already established architectures and ones that are yet to be proposed might be applied to the task of segmenting X-ray images containing pneumothorax. It is possible that other types of architectures are more proficient at extracting the kind of abstract features that make pneumothorax segmentation possible. Other loss functions better adjusted to unbalanced data sets might perform better.

One possibility is that weights pre-trained on a general image database such as ImageNet [22] do not convert well to medical segmentation tasks. It might be more efficient to train the weights on the pneumothorax data set from the beginning. Naturally, this also entails a longer and more computationally intensive training process, requiring more time and perhaps even more specialized hardware.

Recently, Kervadec et al. [32] proposed the new boundary loss function specifically for the segmentation of highly unbalanced sets, e.g. medical image sets. It is based on the distance of boundaries rather than the overlap of regions. It is shown to improve Dice Score by 8% on two medical image test sets when used in conjunction with a Dice loss function. An interesting application of this loss function would be the segmentation of pneumothorax in xX-ray images.

5 Conclusion

The uXception model is at least somewhat proficient at segmenting instances of pneumothorax. At around 35 % Dice score, the model is not yet reliable enough to be used in medical triaging. The study shows that the concept of training a neural network to segment medical images is possible, albeit difficult.

References

[1] Henry M, Arnold T, Harvey J. British Thoracic Society Guidelines for the Management of Spontaneous Pneumothorax. *Thorax*. 2003;58(Suppl 2):ii39.

[2] Sahn SA, Heffner JE. Spontaneous Pneumothorax. *New England Journal of Medicine*. 2000;342(12):868–874.

[3] Ronneberger O, Fischer P, Brox T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. In: International Conference on Medical Image computing and Computer-Assisted Intervention. Springer; 2015. p. 234–241.

[4] Zhou Z, Siddiquee MMR, Tajbakhsh N, Liang J. Unet++: A Nested U-Net Architecture for Medical Image Segmentation. Springer; 2018.

[5] Rahman MM, Desai BC, Bhattacharya P. Medical Image Retrieval with Probabilistic Multi-Class Support Vector Machine Classifiers and Adaptive Similarity Fusion. *Computerized Medical Imaging and Graphics*. 2008;32(2):95–108.

[6] Müller H, Deselaers T, Deserno T, Clough P, Kim E, Hersh W. *Overview of the ImageCLEFmed 2006 medical retrieval and medical annotation tasks*. In: Workshop of the Cross-Language Evaluation Forum for European Languages. Springer; 2006. p. 595–608.

[7] LeCun Y, Cortes C, Burges CJC. MNIST;. Retrieved 10 of July 2019. Available from `http://yann.lecun.com/exdb/mnist/`.

[8] Higham CF, Higham DJ. Deep Learning: An Introduction for Applied Mathematicians. *arXiv*. 2018 January;.

[9] Goodfellow I, Bengio Y, Courville A. *Deep Learning*. MIT Press; 2016. `http://www.deeplearningbook.org`.

[10] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. *arXiv*. 2014;.

[11] LeCun YA, Bottou L, Orr GB, Müller KR. Efficient BackProp. Springer; 2012.

[12] O'Shea K, Nash R. An Introduction to Convolutional Neural Networks. *arXiv*. 2015;.

[13] Vincent OR, Folorunso O, et al. *A descriptive algorithm for sobel image edge detection*. In: Proceedings of Informing Science & IT Education Conference (InSITE). vol. 40. Informing Science Institute California; 2009. p. 97–107.

[14] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*. 2014;15(1):1929–1958.

[15] He K, Gkioxari G, Dollár P, Girshick R. *Mask R-CNN*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017. p. 2961–2969.

[16] Ren S, He K, Girshick R, Sun J. *Faster r-cnn: Towards real-time object detection with region proposal networks.* In: Advances in neural information processing systems; 2015. p. 91–99.

[17] Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, et al. *Microsoft coco: Common objects in context.* In: European conference on computer vision. Springer; 2014. p. 740–755.

[18] Krizhevsky A, Sutskever I, Hinton GE. *Imagenet classification with deep convolutional neural networks.* In: Advances in neural information processing systems; 2012. p. 1097–1105.

[19] Simard PY, Steinkraus D, Platt JC, et al. *Best practices for convolutional neural networks applied to visual document analysis.* In: Icdar. vol. 3; 2003. .

[20] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:12070580.* 2012;.

[21] He K, Zhang X, Ren S, Sun J. *Deep residual learning for image recognition.* In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770–778.

[22] ImageNet. ImageNet Image Database;. Retrieved 10 of July 2019. Available from `http://www.image-net.org/`.

[23] University of Toronto. Cifar-10;. Retrieved 10 of July 2019. Available from `https://www.cs.toronto.edu/~kriz/cifar.html`.

[24] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. *Densely Connected Convolutional Networks.* In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017. p. 4700–4708.

[25] Antoniou A, Słowik A, Crowley EJ, Storkey A. Dilated DenseNets for Relational Reasoning. *arXiv.* 2018;.

[26] Santoro A, Raposo D, Barrett DG, Malinowski M, Pascanu R, Battaglia P, et al. *A simple neural network module for relational reasoning.* In: Advances in neural information processing systems; 2017. p. 4967–4976.

[27] Chollet F. *Xception: Deep learning with depthwise separable convolutions.* In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2017. p. 1251–1258.

[28] Curiale AH, Colavecchia FD, Mato G. Automatic quantification of the LV function and mass: A deep learning approach for cardiovascular MRI. *Computer methods and programs in biomedicine.* 2019;169:37–50.

[29] Kaggle. *SIIM-ACR Pneumothorax Segmentation*;. Retrieved 10 of July 2019. Available from `https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/overview`.

[30] Hauck EL. Data compression using run length encoding and statistical encoding. Google Patents; 1986. US Patent 4,626,829.

[31] Chollet F, et al. Keras;. Retrieved 10 of July 2019. Available from `https://github.com/keras-team/keras`.

[32] Kervadec H, Bouchtiba J, Desrosiers C, Granger É, Dolz J, Ayed IB. Boundary loss for highly unbalanced segmentation. *arXiv preprint arXiv:181207032*. 2018;.

A A Selection of Predicted Images

The ground truth mask is marked in red, the predicted mask is marked in green and areas where both masks overlap are shown in yellow.

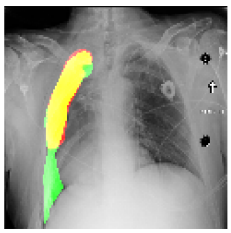


Figure 15: Dice 75.62%

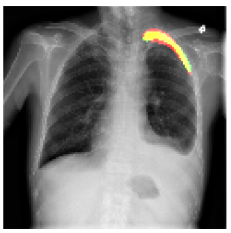


Figure 16: Dice 76.95%

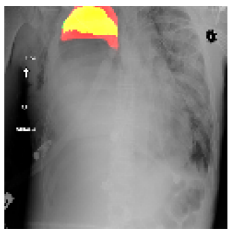


Figure 17: Dice 79.37%

Figure 18: Dice 0.02%

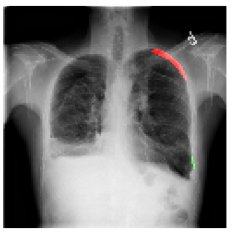


Figure 19: Dice 0.003%

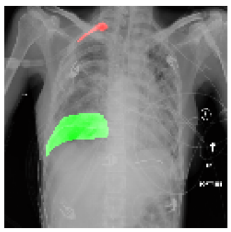


Figure 20: Dice 0.03%

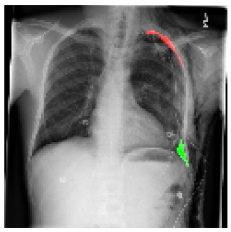


Figure 21: Dice 0.12%

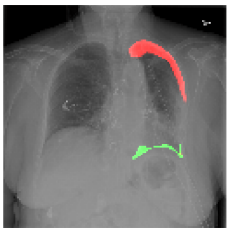


Figure 22: Dice 0.46%

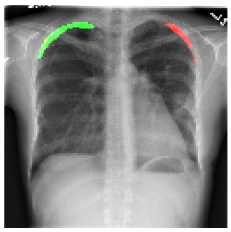


Figure 23: Dice 2.70%

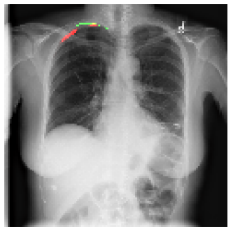


Figure 24: Dice 7.16%

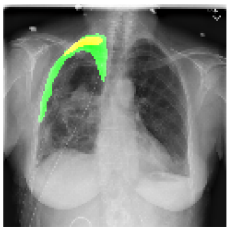


Figure 25: Dice 33.41%

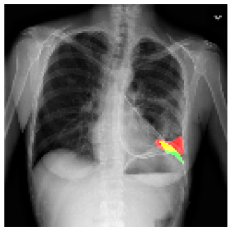


Figure 26: Dice 36.46%n

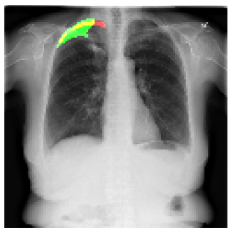


Figure 27: Dice 38.33%

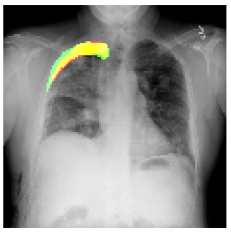


Figure 28: Dice 78.78%

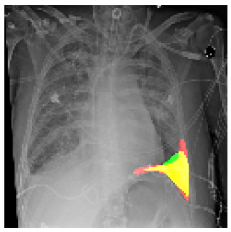


Figure 29: Dice 80.53%

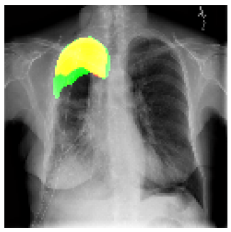


Figure 30: Dice 82.40%

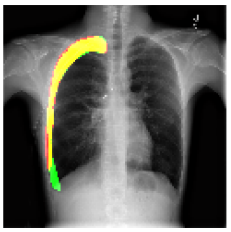


Figure 31: Dice 82.37%

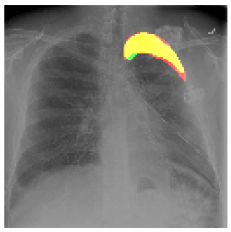


Figure 32: Dice 88.14%