Linköping University | Department of Computer Science
Master thesis, 30 ECTS | Information Technology
2017 | LIU-IDA/LITH-EX-A--17/024--SE

Organ Detection and Localization in Radiological Image Volumes

Detektion och lokalisering av organ i radiologiska bildvolymer

Ola Jigin
Tova Linder

Supervisor : Cyrille Berger
Examiner : Ola Leifler

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Using Convolutional Neural Networks for classification of images and for localization and detection of objects in images is becoming increasingly popular. Within radiology a huge amount of image data is produced and meta data containing information of what the images depict is currently added manually by a radiologist.

To aid in streamlining physician's workflow this study has investigated the possibility to use Convolutional Neural Networks (CNNs) that are pre-trained on natural images to automatically detect the presence and location of multiple organs and body-parts in medical CT images.

The results show promise for multiclass classification with an average precision 89.41% and average recall 86.40%. This also confirms that a CNN that is pre-trained on natural images can be succesfully transferred to solve a different task. It was also found that adding additional data to the dataset does not necessarily result in increased precision and recall or decreased error rate. It is rather the type of data and used preprocessing techniques that matter.

Acknowledgments

Thanks to all employees at Sectra Imaging IT Solutions AB for making us feel welcome and part of your organization during our thesis work. A special thanks to our supervisor **Jonas Strandstedt** for being supportive and engaged in our project. The expertise within the machine learning field provided by **Daniel Forsberg** and **Erik Sjöblom** has been extremely valuable, thanks for supporting our work. We also want to reach out to the project sponsor **Fredrik Häll** for the continous progress discussions to determine where to focus our work.

We also want to thank our supervisor and examiner at Linköping University, **Cyrille Berger** and **Ola Leifler** for keeping the structured and clear schedule for the milestones to be completed at the university. Thanks to **David Berntsen** and **Robin Abrahamsson** for valuable reviews. Finally we want to thank all of the above mentioned for your honest and encouraging comments and questions that have taken our work and thesis report to the next level.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artifical Neural Network
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CS	Code String
CT	Computed Tomography
DICOM	Digital Imaging and Communications in Medicine
GPU	Graphics Processing Unit
GUI	Graphical User Interface
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersect over Union
KL	Kullback-Leibler
MRI	Magnetic Resonance Imaging
PACS	Picture Archiving and Communication System
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SPP	Spatial Pyramid Pooling
SVM	Support Vector Machine

1 Introduction

Staff shortage within health care is a problem affecting the entire society today. The world's population is getting older and larger which means that the demand for medical examinations and treatments is increasing. Meanwhile the existing health care professionals spend an increasing amount of time on administrative activities. Manual input into different kinds of IT systems is an excellent example of such time consuming activities. One way to enable the clinicians to spend more time with the patients is to automate processes to avoid manual input and other manual tasks whenever possible.

Automation of processes that previously depended on manual input can be done by using machine learning. Machine learning is a field within computer science where computers are to learn to act on a task without being specifically programmed for that task [17]. Machine learning algorithms depend on some type of representation, often called features. A simple example is a e-mail spam-filter where the features can be a vector representing the presence of certain words in the e-mail. From the feature representation a model can be trained on many samples to be able to determine if an incoming, previously unseen e-mail is spam or not.

When it comes to detecting objects in images traditional image classifiers relied on handcrafted feature extractors to be engineered depending on the type of the input. Different domains required different extractors and the engineer had to possess domain specific knowledge [25]. The last few years a machine learning technique called representation learning has become very popular for image classification. This technique learns features that are created by transforming the raw input data and does not depend on domain specific feature extractors, making them much more flexible to use on different types of problems and input data.

One representation learning technique is *Deep Learning*. Deep learning has been applied to organ detection in Computed Tomography (CT)-images [35, 7] but also to a variety of other problems within the medical fields. A few examples are diagnosing potential skin cancer from photographs of skin lesions [11], localization of the fetal standard plane in ultrasound videos [6] and right ventricle (heart chamber) segmentation in Magnetic Resonance Images (MRI) [26].

A deep learning method has several layers which are composed of linear and non-linear modules. Each module corresponds to a layer and the module transforms the input into a more abstract representation. This enables complex functions to be learned. Using this technique there is not much engineering that is done by hand, rather it is automated in the learn-

ing of the method [25]. One deep learning method is *Convolutional Neural Networks (CCNs)*. The complex function that a deep learning model represents is learned through training with a large amount of training data. In 1996, Sahiner et al. [37] applied CNNs to medical images, the training time for the CNN was described as "computionally intensive". Today, the rapid development of GPUs (Graphics Processing Unit) enables the use of deep learning methods since more computational power is available in form of parallel computing [9].

A big challenge that still remains when using deep learning techniques is the amount of data available for training the model. If too little training data is used to train a large model there is a risk that the complex functions fit too well to the training data and do not generalize well, an issue known as *overfitting* [25].

This thesis investigates the possibility to automatically create a *table of content* for CT examination listing depicted organs and their location through image analysis with deep learning techniques. It also investigates the effect of the amount of training data when training a deep learning system.

1.1 Motivation

The project is carried out at Sectra AB, within the field of Medical Imaging IT Solutions. Sectra's main office is located in Linköping and the company offers products and provides services within the fields of Secure Communication and Medical Imaging IT Solutions.

1.1.1 Sectra

Sectra Medical Imaging IT Solutions AB develops among other things software for storage, management, transfer and visualization of radiology data, called PACS (Picture Archiving and Communication System). The universal standard DICOM is used in the PACS system. Today the meta-data for a DICOM image contain a manually added examination code describing the main organ examined, the tag is called *Body Part Examined*[1]. The tag can take predefined values or any value that conforms to the DICOM value representation Code String[2] (CS). There is always a risk of inconsistency or insufficient information when information is manually added with such freedom, for example different hospitals or even clinicians use different conventions or languages when adding the meta-data. Gueld et al. showed in a study the insufficiency of using only DICOM tags for image classification. They found that around 15% of the images in their dataset were falsely tagged and ended up being incorrectly classified [18]. In this study manually added tags are not considered as input to the deep learning system due to reliability concerns.

Sectra's software provide a feature showing *priors*. Priors are previous examinations of the same organ as the clinician is currently viewing. These priors are retrieved using complex rule sets that are based on the examination code. A table of content could simplify the complex rule sets for retrieval since information of all organs depicted in each image would be available. It would also reduce the risk of missing out on relevant priors when the organ currently being examined was depicted but was not the main organ examined in the prior examination. The clinician does not have to search for relevant images thus saving time.

When dealing with images depicting different parts of the body there are clearly many different types of tools and features in Sectra's software. With the knowledge of what organs that are depicted in an image the GUI could dynamically adapt and only display tools that are relevant for the clinician. One example is suggesting the relevant organ specific medical decision support documents depending on what organs the clinician is viewing. This can aid the clinician in making the right decisions and will also save time browsing for the right tools and decision support documents.

[1] According to the DICOM standard tag (0018, 0015),http://dicom.nema.org/dicom/2013/output/chtml/part03/sect_C.8.html
[2] DICOM Code String, ftp://dicom.nema.org/medical/DICOM/2013/output/chtml/part05/sect_6.2.html

Today there is also a possibility for the clinician to manually add an organ tag to the annotations they make in the images. This feature is not frequently used since it has to be done manually. If the organs depicted in the image are known to the system a tag can be suggested to the clinician thus improving the user experience. An increased use of the feature will also result in more data which can be used to improve the exisiting intelligent systems or to train deep learning models for new applications within the medical domain.

1.1.2 General Application

The previous section described applications of an automated organ tagging system specifically for Sectra PACS. The motivation is however applicable to the general medical field as many systems could take advantage of such feature. All PACS systems use DICOM standard the problem with inconsistency and insufficient classification by just using the DICOM tags is recurrent, making research on other classification approaches valuable to the community. Automatic tagging solves the general problem of abundant time spent on structuring, retrieving, filtering and analyzing data manually.

1.2 Aim

The goal with this project is to investigate and evaluate solutions to automate the classification and localization of organs in CT images using deep learning. The output from the thesis is a stand-alone system that is able to classify images based on the organs they contain and locate organs in CT stacks.

1.3 Research Questions

Given the introduction and background the following are the research questions that will be answered in this thesis.

1. How good accuracy, precision, recall, F-score and error rate [3] can be achieved using a CNN taking the pixel data of the image as input to do organ- or body-part-specific anatomical classification of 2D medical CT images?

   Previous studies have shown that CNNs can be used to successfully classify objects in medical images [7]. The studies have however detected individual or a few objects or organs located in disjunct parts of the body. In this study these questions will be answered for a larger set of organs where multiple organs can appear in the same images.

2. Can a CNN taking the pixel data of 2D CT images as input on its own suffice to create a *table of content* over the organs and body-parts depicted for an entire CT stack?

   The previous research question answers how well the CNN can classify individual 2D images, however the CT stacks are sets of many images [4]. This research question moves from single images to analyzing the performance in a real-world context where the images of one stack are considered as one unit. This makes it possible to pass the images of an entire stack to the classifier consecutively. It will be investigated if the individual predictions for the 2D images are enough to create the *table of content* with the depicted organ and its location within the stack or if additional processing such as using heuristics based on prior knowledge about the stacks (for example the order of the images) are needed.

[3] definitions of metrics can be found in Section 3.5

3. How does the amount of training data affect the accuracy, precision, recall and error rate when training a CNN to classify 2D CT images?

   There have been experiments on how the number of training samples affect the accuracy of a CNN in the medical domain, for example the study by Cho et al. [7]. The goal with this research question is to find out if an increased amount of training data would likely improve the accuracy and error rates of the classification network.

4. Can a sliding window technique be used together with a classifying CNN to locate and detect the spatial location of organs and body-parts within a medical 2D CT image?

   The second research question introduces the concept of a *table of content* over organs and body-parts depicted in a CT stack. Since CT stacks are 3D a classifying 2D images can aid in determining the location in one of the dimensions, however to determine the location in the other two dimensions localization of an organ or body-part within the 2D image must be done. This research question is answered by producing suggestions on how localization within 2D images can be done with a CNN. The goal is to provide a direction for future work in localization.

1.4 Delimitations

The dataset is limited to only contain CT images. Compared to natural images the availability of medical images is limited, often because of privacy regulations. A fact resulting in the availability of labelled medical data at public sources being significantly lower than the availability of labelled datasets with natural images. Due to this only data from Sectra will be considered in this study.

1.5 Thesis Outline

The next Chapter covers relevant background theory. In Chapter 3 the method is described followed by the results in Chapter 4. The method and results are discussed in Chapter 5. Finally, the conclusions are presented in Chapter 6.

2 Theory

As mentioned in the introduction machine learning algorithms depend on some type of representation or features. For a simple algorithm the features are passed as input and the algorithm can make a prediction [17]. In the traditional image classifiers where features were not automatically learned the feature extraction became a problem-specific engineering problem itself. The extracted features could then be passed to a trainable systems, for example a Support Vector Machine (SVM).

In Artificial Neural Network (ANN) the model is a feed-forward network of highly connected neurons (or units). The model uses representation learning to discover and learn more complex features from simpler features that are based on the input data [17]. This makes the data representation and feature extraction more general and applicable to many different problems and inputs. The networks are inspired by the way a biological brain works and can have one or many input units, called the visible layer (input layer). Within the network there are hidden units composed in multiple hidden layers, the final layer acts as output layer and can have multiple output units. Today a more general term, *Deep Learning*, is used to describe the field in machine learning where the models learn multiple levels of composition, such as ANNs [17]. Figure 2.1 shows an ANN network.

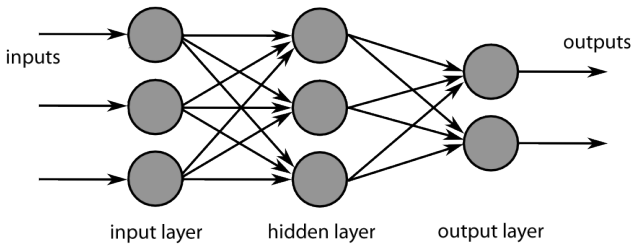


Figure 2.1: A small example of an ANN architecture with the visible input layer, one hidden layer and the output layer. Each gray circle represent a neuron. [1]

The learning procedure of a deep learning algorithm is supervised or unsupervised. In supervised learning the data sample is associated with a label ("the correct answer" for that

[1] *Diagram of a multi-layer feed-forward artificial neural network* by Chrislb licensed under CC BY-SA 3.0

sample), often called the ground truth. The algorithm learns to predict the probability $P(y|x)$ where x is the input and y is the output of the algorithm. Unsupervised learning algorithms must be able to learn a probability distribution for $p(x)$ by analyzing the structure of the input data. Supervised learning is usually used for regression or classification problems and unsupervised learning is commonly used for clustering problems [17].

The act of passing input that propagates through the network and produces an output is called *forward propagation* [17]. Each neuron has $n + 1$ (the 1 being a bias input) inputs x_i. On the input the weights w_{ij} are applied, see Figure 2.2. The set of weights for the entire network is often denoted Θ and these weights affect the output from the network. The training process adjusts these weights in order to improve the performance of the network.

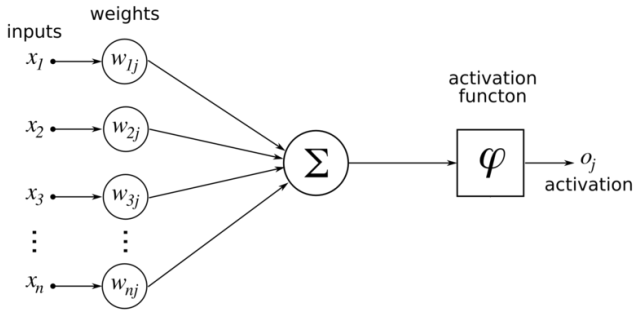


Figure 2.2: An artificial neuron with inputs x, weights w_{ij} and activation function φ. [2]

An activation function φ is applied to the sum of the inputs for a neuron j and produces the output activation o_j in Equation 2.1. More on activation functions is found in Section 2.1.1.

$$o_j = \phi\left(\sum_{i=1}^{n} w_{ij}x_i\right) \tag{2.1}$$

When using supervised learning to train the network a loss or cost $J(\Theta)$ is calculated to measure how inaccurate the output from the network is compared to the ground truth for a set of weights Θ. This is done by letting the sample forward-propagate through the network. The network improves its performance by updating the weights connecting the neurons, this is done by minimizing the cost by calculating its gradient and use it with an optimization algorithm [17]. It is expensive to numerically compute gradients but the *back-propagation algorithm* uses an inexpensive approach where the cost information flows backwards in the network and the gradients can be calculated. It is therefore common to use the back-propagation algorithm when calculating the partial derivatives in a *back-propagation process* in a multi-layer neural networks [17].

2.1 Convolutional Neural Networks

A regular artificial neural network takes features as input, which means each pixel in an image can be passed to the network [25]. Using one feature per pixel generates very heavy calculations when working with large images as input. The introduction of Convolutional Neural Network enabled a way to discover features in multi-dimensional data efficiently [25]. Instead of using each pixel as a separate input feature the convolutional neurons are responsible for convolving a spatial region of the image, called *receptive field*, as in the book *Deep Learning* by Courville et al. [17]. Courville et al. also explains that the statistical properties of an image are not sensitive to translation. They also state that due to this fact the same

[2] *Diagram of an artificial neuron* by Chrislb licensed under CC BY-SA 3.0

weights, *shared weights*, are used for all neurons in the same layer and reducing the number of parameters in the CNN.

2.1.1 CNN Architecture

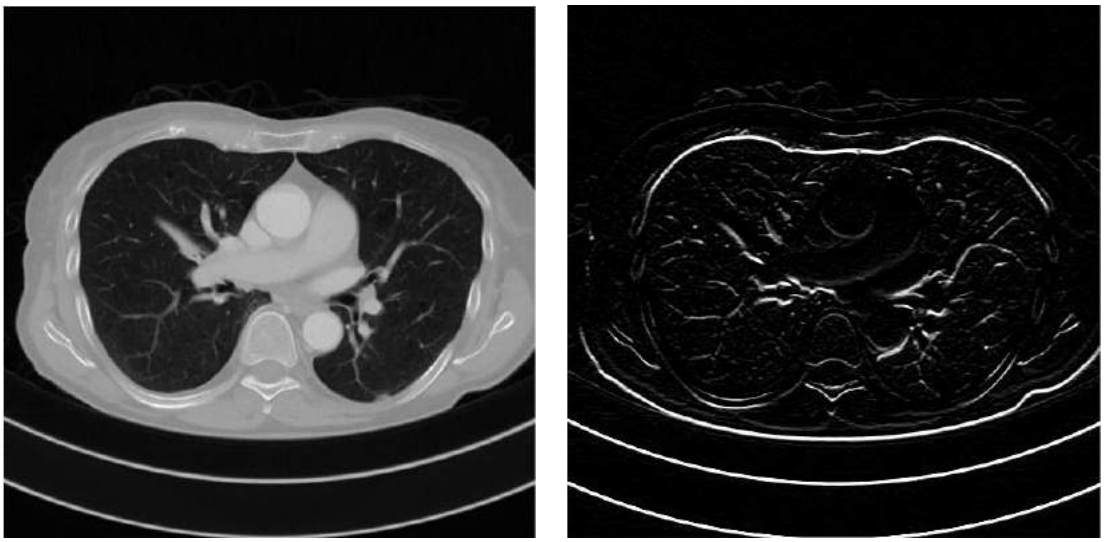
There are different types of layers that make up the architecture of a CNN. In this section convolutional layers, pooling layers and fully connected layers are introduced.

The layers in a CNN are organized into stages, with the initial stage containing convolutional layers followed by pooling layers [25]. Each convolutional layer takes a 3D volume as input and outputs a 3D volume (feature maps) by passing the input through an activation function.

A sequence of these layers is often repeated multiple times and followed by fully connected layers preceding the output layer [25].

Convolutional Layer

The role of the convolutional layer is to apply filters to the raw data in order to detect features and produce feature maps [25]. A feature map is a new representation of an image after applying some type of operation to the entire image (a convolutional operation). Each filter is responsible for applying one type of operation but applies that specific operation to all spatial regions of the image. The output from each filter is one feature map. An example of an input and the resulting feature map can be seen in Figure 2.3.



(a) Original representation (b) Resulting feature map

Figure 2.3: Example of the feature map of an image after applying an operation to the original representation of the image.

Since there are multiple filters in each convolutional layer the resulting output will be a set of 2D feature maps which are represented as a 3D volume. The convolution operation can for two-dimensional input and filters be defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.2}$$

where S is the feature map, I is the two-dimensional input and K is the two-dimensional filter [17]. The output of the convolution operation in Equation 2.2 corresponds to one cell in the output in Figure 2.4. Once the operation is performed, the filter slides to the next

spatial region and performs the same operation. This is repeated until all spatial regions are covered. A filter have a *stride* which decides how many pixels at the time the filter shall slide over the image and move to the next spatial region. The filter have a width and height which are equal for all filters in a layer. The number of filters used are referred to as the *depth* of the convolution layer and defines the depth of the output 3D volume from the layer. It is also possible to use *zero-padding* in the convolutional layer. This means that input volume is padded with zeros in order to control the spatial size of the output volume, commonly used to keep the output volume the same as the input volume. Padding with zeros is used to ensure that padding does not contribute to the convolution. Figure 2.4 displays an example of how the convolutional operation works.

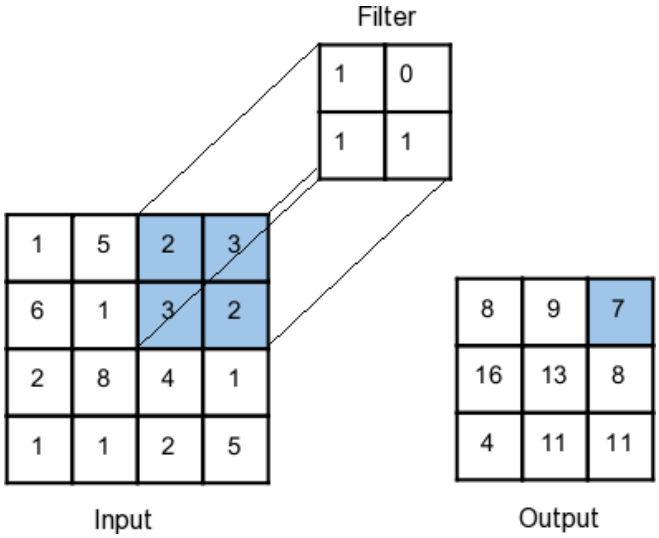


Figure 2.4: A convolution with a filter size 2x2 and input size 4x4, no zero-padding and stride 1 (sliding 1 pixel at a time) where the filter is currently operating one of the spatial regions of the image.

The filters are represented as the weights on the links from the previous layer [25]. Before the next layer in the architecture an activation function is applied to all feature maps. The activation function defines the output of a neuron. A common activation function in CNNs is the rectifier function:

$$g(z) = max\{0, z\}, \text{ where } z \text{ is the input (the feature map).} \tag{2.3}$$

Two other activation functions are the sigmoid function:

$$g(z) = \sigma(z), \text{ where } \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

and the hyperbolic tangent activation tangent function:

$$g(z) = tanh(z), \text{ where } tanh(z) = 2\sigma(2z) - 1 \text{ and } \sigma \text{ is the same in Equation 2.4.} \tag{2.5}$$

Krizhevsky et al. used the rectifier function in Equation 2.3, since training time can be reduced compared to using the sigmoid activation function in Equation 2.4 or the hyperbolic tangent activation function in Equation 2.5 [23]. Units that use the rectifier function are called Rectifier Linear Units (ReLUs) [30]. Courville et al. [17] states that the default recommendation in modern neural networks is to use ReLU .

Pooling Layer

The role of the pooling layer is to merge similar features into one to reduce the dimension of the features. One common problem when trying to fit a model is overfitting which means either that the model fits too well to the training data or that the model fits to random noise and therefore does not represent the underlying relationship between parameters as describe in a paper called *Deep Learning* by LeCun et al. [25]. An algorithm trained with a small training dataset is more prone to overfitting [6]. LeCun et al. also describe that pooling layers decrease the number of parameters and the amount of computation done in the network, hence it also reduces the risk of overfitting. It is common to use a maximum function as a pooling layer, often referred to as max pooling. The max-pooling filter simply takes the maximum pixel value within the spatial region the filter is currently operating on to produce the output pixel, Figure 2.5 shows an example of a max-pooling filter. The pooling layers have a *filter size* and a *stride*. The filter size determines the number of pixels in the original image represented by one pixel in the pooled image. The stride is similar to the stride in a convolutional layer and describes how many pixels the filter should slide at the time [17].

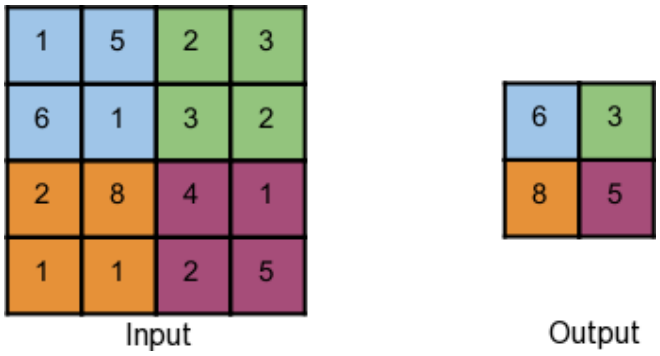


Figure 2.5: The input and output when applying a max pooling filter with size 2x2. The stride value is 2.

Fully Connected Layer

The fully connected layers are not specific for convolutional neural networks, they have the same architecture as the fully connected layers for a regular neural network. In the book *Deep Learning* Courville et al. [17] describes that each neuron in the fully connected layer have connections to all output activations of the previous layer. In a CNN that is designed to classify, this layer is responsible for the classification, the output of the last fully connected layer is the classification score for each class.

The activation function in the last fully connected layer is adjusted depending on the task the network is solving, e.g. classification or regression. Courville et al. suggest that in the case of classification according to a multinoulli distribution, i.e. only one out of many multiple is to be selected, the softmax function is shall be used. The softmax function ensures that the sum of the elements in the output vector is 1. For a multiclass-multilabel classification task, where the probabilities are independent, the sigmoid function can be used instead as described in the book *Deep Learning*. It can range anywhere from 0 to $1 \cdot k$ where k is the number of classes. The sigmoid function is defined in Equation 2.6 where $f(x_i)$ is the probability that a sample belongs to class i.

$$f(x_i) = \frac{1}{1 + e^{(-x_i)}} \forall i \tag{2.6}$$

For a regression task a linear output activation function as seen in Equation 2.7 is used, to transfer the values predicted by the network to the output. This enables output values to be any real-valued number.

$$f(x_i) = x_i \forall i \tag{2.7}$$

From Individual Layers to a Network

Figure 2.6 shows an example of a CNN. First convolutions are done on the original input image creating feature maps. The subsampling represents the pooling layers discussed above and reduces the size of the feature maps. The convolution-subsampling pattern is repeated several times until the final feature maps have been created and these are passed to the fully connected layers which produce the final output from the network.

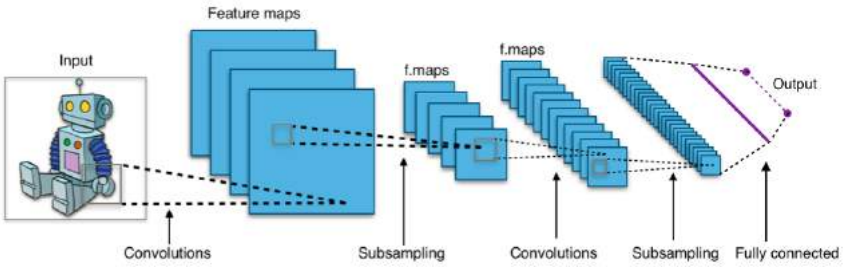


Figure 2.6: A typical CNN architecture. Convolutional operations create feature maps, the stages annotated as subsampling in the figure are the pooling operations. [3]

2.2 Training a CNN

In the book *Deep Learning*, Courville et al. [17] describes that when training a CNN the original dataset is split into a training set and a held-out test set in a process called *data generating process*. They also explain, when the training is complete the test set is used to evaluate how well the model generalizes to previously unseen data within the same domain given that the two sets are independent. In CNNs the settings that can be adjusted to effect the behaviour and performance of the network are called *hyperparameters*. When optimizing these an unseen data set is needed to confirm that the model generalizes. This set is created by splitting the training data into two disjoint sets, the training set that makes up approximately 80% of the original training data and the validation set that makes up approximately 20% of the original training data as mentioned in the book *Deep Learning*. Since the hyperparameters are tuned using the validation data the validation error will typically underestimate the error measured on the real test data when the training is complete.

2.2.1 Weight initialization

The weights (or parameters) Θ in a CNN network must be initialized to some value. It has been shown by several, such as Glorot et al. [16] and He et al. [20] that initializing the weights according to some statistical distribution gives faster convergence and lower error rates than randomly initialized weights. Glorot et al. draw a sample from a set that is normally distributed around 0 and with standard deviation σ as seen in Equation 2.8

$$\sigma = \sqrt{\left(\frac{2}{fan_{in} + fan_{out}}\right)} \tag{2.8}$$

where fan_{in} is the number of inputs into the neuron and fan_{out} is the number of outputs from the neuron.

[3] *typical CNN architecture* by Aphex34 licensed under CC BY-SA 4.0

2.2.2 Number of iterations

The iterations over the dataset are often split into *epochs*, which is a hyperparameter that can be tuned to an appropriate value to avoid overfitting. However this parameter can be optimized without barely any cost by implementing *Early Stopping* [1, 17]. Depending on how many times the training data is iterated over during one epoch the number of iterations and epochs can be the same. Early stopping terminates the training process by observing the validation error and validation accuracy after each epoch using heuristics to determine when we have seen the best model. One commonly used heuristic is implemented based on *patience*, if the validation error has not improved over a certain number (the value of patience) of epochs the training process is terminated [1, 17].

2.2.3 Objective

During training the weights are updated to minimize an objective function. There is a difference between pure optimization and training a deep learning model. Courville et al. [17] describes that the goal when optimizing the objective function is to optimize the result of some performance measure, but in pure optimization the goal is to optimize the objective function itself. It is therefore important to find an appropriate objective function to represent the performance measure of the task. The objective function serves as a measure of the inaccuracy of the current prediction and is denoted $J(\Theta)$ where Θ are the weights in the network. According to Courville et al. different objective functions are used depending on the type of task the model is trained to solve. Common objective functions for regression problems are *Mean Squared Error* (Equation 2.9) and *Mean Absolute Error* (Equation 2.10), x^j is the prediction for sample j and y^j is the ground truth and n is the number of samples.

$$J(\Theta) = \frac{1}{n}\sum_{j=1}^{n}(x^j - y^j)^2 \tag{2.9}$$

$$J(\Theta) = \frac{1}{n}\sum_{j=1}^{n}|x^j - y^j| \tag{2.10}$$

For classification problems the *Logistic Loss Function* is commonly used, it is often referred to as log-loss which is visualized in Figure 2.7. Log-loss and *Cross-entropy Loss Function* are very similar and share the underlaying math.

The entropy of a variable is a measure of uncertainty for a variable X with a distribution p [29]. Objective functions calculate a loss based on two distributions, p and q. The dissimilarity between the two distributions can be calculated with Kullback-Leibler divergence (KL divergence) measure. KL divergence is also called relative entropy and is defined as follows [29]:

$$\mathbb{KL}(p||q) \triangleq \sum_{n=1}^{K} p_n log\frac{p_n}{q_n} \tag{2.11}$$

When the sum is replaced by an integral for probability distribution functions it can be rewritten into [29]:

$$\mathbb{KL}(p||q) = \sum_{n} p_n log p_n - \sum_{n} p_n log q_n = -\mathbb{H}(p) + \mathbb{H}(p,q) \tag{2.12}$$

where $\mathbb{H}(p,q)$ is called cross entropy [29].

$$\mathbb{H}(p,q) \stackrel{\wedge}{=} -\sum_{n} p_n log q_n \tag{2.13}$$

To describe cross entropy in a more intuitive way one could explain it as the negative sum of the products of the logs of the predicted probabilities multiplied by the ground truth probabilities. Cross-entropy is closely related to logistic loss. When working with binary classification, $p \in \{ y, 1-y \}$ and $q \in \{ \hat{y}, 1-\hat{y} \}$, cross entropy can be written as:

$$\mathbb{H}(p,q) = -\sum_{n} p_n log q_n = -y log \hat{y} - (1-y) log(1-\hat{y}) \tag{2.14}$$

which is the definition of logistic loss for two distributions p and q as seen in Figure 2.7. p can be seen as the ground truth labels, a distribution containing just ones and zeros and q as the predicted probabilities.

Predicting a high probability $\hat{y}$ for a class with a ground truth $y = 0$ or a low $\hat{y}$ for a class with $y = 1$ one will result in a high loss value as seen in Equation 2.14. The log function between zero and one is visualized in Figure 2.7.

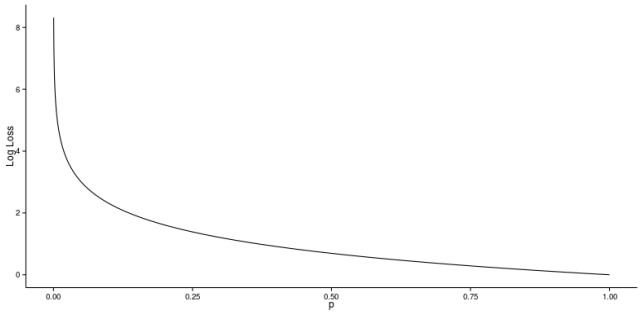


Figure 2.7: Log function for probabilities in interval $\{ 0, 1 \}$. [4]

On multi-class classification the logistic loss function can be summarized over all classes to calculate the total loss.

2.2.4 Optimizer

An optimizer is responsible for minimizing the objective function. *Stochastic Gradient Descent (SGD)* is commonly used for this purpose [7, 24, 40, 45, 48]. According to Courville et al. [17] it is the most common optimization algorithm for deep learning. A regular stochastic gradient descent updates the weights for each training sample and a batch gradient descent updates the weights once for all training samples. Courville et al. also describe mini-batch gradient descent where the gradient is computed and the weights are updated after n training samples. The mini-batch gradient descent updates according to:

$$\Theta = \Theta - \eta \cdot \Delta_{\Theta} J \left(\Theta, x^{(j:j+n)}, y^{(j:j+n)} \right) \tag{2.15}$$

Where Θ are the parameters (weights), η is the learning rate, j is a sample in the training data and $\Delta_{\Theta} J$ is the loss calculated for the mini-batch with the objective function.

In the book *Deep Learning* [17] the term *batch size* is used to describe the size of the mini-batch and has to be selected when training a model. If the batch size is small, a small learning

[4] *log-loss curve* by Bfortuner licensed under CC BY-SA 4.0

rate must be used to maintain stability. The training time will also be longer since more updates needs to be done due to the small learning rate and more batches needs to be processed in order to cover the whole dataset. According to Courville et al. the samples in the batches are run in parallel if the hardware setup allows it making the hardware the limiting factor on how large batches that can be used. Bengio states in *Practical Recommendation for Gradient-Based Training of Deep Architecture* [1] that a batch size of 32 samples is a good default value. He also claims that once the batch size has been chosen that value can be fixed while optimizing other hyperparameters.

SGD can be used with momentum which includes the previous weight change into the update to reduce the effect of gradient noise [47], where $0 \leqslant momentum < 1$.

2.2.5 Learning Rate

The learning rate (η) determines how fast the algorithm moves towards the optimal value, i.e. the step size when optimizing, as described in the book *Deep Learning* [17]. If the learning rate it is set too high there is a risk to miss the optimal solution and if it is set too low too many updates are required before reaching the optimal solution. Courville et al. suggests that it can be effectively chosen by monitoring the learning curves while training the network. Bengio [1] recommends an initial learning rate of $\eta = 0.01$ as default value for deep networks but points out that if there is only time to tune one hyperparameter the learning rate should be chosen for optimization.

According to Courville et al. [17] it is common to use a learning rate decay schedule when using Stochastic Gradient Descent as the optimizer introduces noise that will remain through the whole training process, even when optimum is close. There are various ways to implement the learning rate decay schedule. Courville et al. suggest one common approach where the learning rate is decreased linearly until epoch τ. Bengio suggests another approach where the learning rate is constant until epoch τ and then decrease it by factor $O(\frac{1}{t})$ where t is the current epoch.

2.3 Optimization of Hyperparameters

When optimizing the hyperparameters the learning process must be observed, this is used by analyzing learning curves. The learning curves are created by plotting the training accuracy, training loss, validation accuracy and validation loss for each epoch.

2.3.1 Observing the learning process

A model can suffer from two distinct problems: *overfitting* and *underfitting*, see Figure 2.8. A model overfitting the training data passes through the training data points exactly and will not generalize well, a model suffering from underfitting will not be able to capture the training data points at all [17].

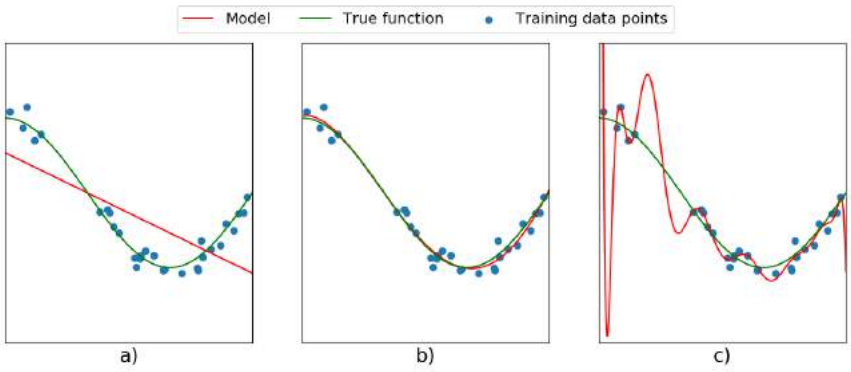


Figure 2.8: A simple example to illustrate *a)*: underfit, will neither generalize nor fit the training data well *b)*: fits true function, will generalize well *c)*: overfit, will not generalize well.

To obtain a model that generalizes the validation error (also called generalization error) and the training error are observed during the training process. Generally the following holds [17]:

- If both the training error and the generalization error are large - the model is underfitting the training data
- If the training error and the generalization error diverge - the model is overfitting the training data
- If the generalization error is just slightly higher than the training error - the model generalizes well

2.3.2 Hyperparameter optimization techniques

There are several techniques to optimize hyperparameters in a neural network, for example *Grid Search* and *Random Search*. Model-based hyperparameter optimization is another approach where for example a Bayesian regression model estimates the expected validation error before exploration and explores parameters that will likely perform well. These are however often very similar to the parameters the algorithm has already seen [17]. Courville et al. do not explicitly recommend the use of model-based approaches for hyperparameters in neural networks since it sometimes can perform very well but in other cases it fails badly.

Bengio [1] makes a point that humans can get very good at optimizing parameters manually. For reproducibility reasons it is however recommended to avoid human decisions to infer within the optimization process. Bengio also suggests the hyperparameter ranges can be specified in the paper and is therefore the only part that should include human decisions.

Grid Search

In the book *Deep Learning*, Courville et al. [17] explains that grid search is an algorithm for hyperparameter optimization which is commonly used if the number of parameters to optimize are three or fewer. For each individual parameter a set of values is set up, usually picked on a log-scale . Bengio [1] suggest that it should be ensured that the "best" values aren't the ones on the border, as this risks that better values are unexplored outside the interval. He also states that three values are insufficient for the optimization even if the best value is the middle value. The values of the parameters make up a parameter space and the goal of the grid search algorithm is to find the combination of parameters giving the optimal solution in the parameter space. Courville et al. describes that the algorithm does an exhaustive search

by training the model for each combination of hyperparameters in the cartesian product of the sets of individual parameters. It returns the combination of hyperparameters giving the best validation error. Bengio highlights that grid search has the advantage of being parallelizable. However, both Bengio and Courvile et al. means that the exhaustive search makes grid search suffer from the *curse of dimensionality* as the number of combinations of hyperparameters grow exponentially when the number of individual parameters is increased.

Random Search

In random search, as explained in the book *Deep Learning* [17], the sets of values for each hyperparameter is not explicitly defined, instead intervals in which the algorithm picks values are defined. Courville et al. and Bengio define these as a marginal distribution for each hyperparameter, a uniform distribution on log-scale is appropriate for positive real-valued hyperparameters. Bergstra et al. [2] have compared the use of grid search and random search for optimizing hyperparameters in neural networks and found that random search finds models that are better or at least as good as the models found with grid search on the same domain, but the computation time was a small fraction of the computation time for grid search. Courville et al. explains this improved computation time by the fact that no experimental runs are wasted since the individual parameters more likely get different values in each exploration. They also described that in grid search there are cases where one parameter is changed that does not make much difference on the validation error, these cases are avoided with random search since all the other hyperparameters will have different values.

2.4 Improving CNN Accuracy

To improve the accuracy of a CNN there are techniques that can be used. In this section a few such techniques are described - data augmentation, a number of regularization techniques and the use of transfer learning. The techniques described are frequently used when applying CNN models to a problem where the amount of training data is limited.

2.4.1 Data augmentation

In a study by Chatfield et al. [5] data augmentation is described as a tool to deal with overfitting and to optimize the dataset in deep learning. Roth et al. [35] showed that data augmentation reduces classification error rates. With data augmentation they achieved an error rate of 5.9% compared to 9.6% without. Chatfield et al. suggests there are different methods to do data augmentation, e.g. flipping, translation and cropping. This creates new samples of the image, transformed images. The generated samples from the original image may be used during training or testing. By using data augmentation, Chatfield et al. explains that the dataset becomes larger which reduces the likelihood of overfitting.

2.4.2 Regularization

Courville et al. [17] describes the general term *regularization* as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error". This makes for example the previously described *Early Stopping* technique a form a regularization. Courville et al. means that there is no obvious form of regularization, it has to be chosen depending on the task of the machine learning algorithm.

Weight Decay

The weight decay regularization coefficient λ can be set to a value greater than 0 to push the weights towards a specific value, this specific value is normally zero according to Bengio [1], but Courville et al. [17] point out that other values could be used. The weight decay

penalizes large weights, limiting the freedom of the model, and thus prevents overfitting. Bengio explain that weight decay is practically done by adding a regularization term to the objective function of the model. Adding the regularization term results in a new objective function $J(\Theta)$, as seen in Equation 2.16, where L represents the regularization term:

$$J_{reg}(\Theta) = J(\Theta) + L \tag{2.16}$$

The most common form of weight decay is L^2 regularization (Equation 2.17) but others such as L^1 regularization (Equation 2.18) are also used [1].

$$L^2 = \lambda \sum_j \Theta_j^2 \tag{2.17}$$

$$L^1 = \lambda \sum_j |\Theta_j| \tag{2.18}$$

Using L^2-regularization will thus result in the following objective function:

$$J_{reg}(\Theta) = J(\Theta) + \lambda \sum_j \Theta_j^2 \tag{2.19}$$

When optimizing, with for example SGD, the gradients are calculated based on the new objective function and the regularization term will propagate into the optimization algorithm. Choosing a too large λ will limit the freedom of the model excessively and result in underfitting and a too small λ will result in overfitting [1].

Bengio [1] states that using early stopping plays essentially the same role as L^2 regularization. He also means that tuning early stopping is easier than optimizing L^2 regularization, thus recommending to drop L^2 regularization if early stopping is used. However, he suggests that L^1 regularization can sometimes be useful to ensure that parameters that are not useful for the fit are driven towards 0.

Dropout

Adding dropout layers is a way to reduce the risk of overfitting. A dropout layer drops neurons in the network and their connection with a certain probability [42]. Hinton et al. [21] show that simply adding dropout to a neural network reduces the error rates for several benchmark tests.

2.4.3 Transfer Learning

Training CNNs from scratch (fully train) requires a substantial amount of labelled data which is not always available, especially within the medical domain. The limited amount of data can cause problems with overfitting. Chen et al. show in *Standard Plane Localization in Fetal Ultrasound via Domain Transferred Deep Neural Networks* [6] that even though medical images are different from natural images they do share many low-level features. Their result is promising for the use of an approach called transfer learning, where a pre-trained network is fine-tuned. A pre-trained network in this context is a CNN that have pre-trained weights which are re-used in the new network. Fine-tuning means to adjust the weights in the pre-trained network to the target task [45]. Chen et al. stated in 2015 that they believe transfer learning can be the solution to be able to use CNNs in domains where the limitation is the size of the dataset. With 11942 training images they trained a CNN to locate the standard plane in fetal ultrasound.

Furthermore, fully training a network is time-consuming and requires a considerable amount of computation power and memory resources. Several studies [45, 48, 24, 6, 15] show that the use of transfer learning can improve accuracy and reduce training time. Yosinski et al. [48] have experimentally investigated how transferable features in deep neural networks are. Even though they find that the transferability decreases with the distance between the task the network is trained for and the new task it is applied to they do find that weights from a distant task give better results than initializing random weights.

In a CNN there are l layers as described in section 2.1.1, each weight of corresponding layer l in the pre-trained network is transferred into a new network. The last fully connected layer is not transferred because every network has different desired outputs. To compute the weights of the newly constructed fully connected layer the pre-trained CNN can be used as a feature extractor and the computed features are used as input to the fully connected layer during a training process. [45].

Tajbakhsh et al. [45] refer to fine-tuning only the fully connected layers as *shallow tuning* and fine-tuning all layers as *deep tuning*.

The problem with overfitting arises once again if the entire CNN is fine-tuned. Tajbakhsh et al. suggests one approach to handle overfitting which is to work backwards from the last layer and incorporate the earlier layers if necessary. It is not always necessary to do deep tuning since the first layers learn low level features which is a very similar process for many different tasks.

Transfer learning also includes the possibility to use the pre-trained network as a feature extractor. Razavian et al. [32] explain that the last fully connected layer is removed from the pre-trained network and the entire CNN is now seen as the feature extractor. The output for the CNN is a feature vector. Razavian et al. means that this feature vector may be the input to another CNN or to a linear classifier, e.g. a SVM.

2.5 Classification, Localization and Detection with CNN

Classification, localization and detection are three tasks with increasing difficulty within computer vision. Classification treats the case where images are classified according to the main objects depicted in the image. In localization the goal is to be able to determine the location of a certain object in an image. Localization and detection tasks are very similar as detection is to locate all objects present in an image. The main differences between the two tasks are that an algorithm giving false indications of the presence of an object is penalized in detection and that a detection algorithm must be trained to recognize a background (parts of the image that does not contain any of the seeked objects). Convolutional Neural Networks have been widely used to solve these tasks on medical CT images [39, 35, 24, 7, 6].

2.5.1 Image Classification with CNN

Using neural networks for image classification became very popular when Krizhevsky et al. [23] applied neural nets in the ImageNet Large Scale Visual Recognicition Challenge (ILSVRC) in 2012 and managed to win with a classification error of 15% where the second best achieved 26% on the ImageNet dataset. The ILSVRC is a yearly competition for large scale object detection and image classification [36]. The ImageNet datasets contain in total over 14 000 000 natural images and are publically available on the ImageNet website[5]. The network presented in *ImageNet Classification with Deep Convolutional Neural Networks* by Krizhevsky et al. [23] was one of the largest to be trained on the ImageNet dataset at that time, making overfitting a major problem. They proposed several techniques to prevent overfitting which are still major techniques today, such as dropout and data augmentation. Their main limitation of the network size was the capability of the GPUs at the time and their final network contained

[5] image-net.org

five convolutional layers and three fully-connected layers. Their experiments showed that the depth of the network is critical to achieve good results. Simonyan et al. [40] also analyzed the effect of the depth of the convolutional network on the classification accuracy by keeping all parameters except for the network depth unchanged. Their conclusions in *Very Deep Convolutional Networks for Large-Scale Image Recognition* [40] in 2014 were that deeper networks are beneficial for the accuracy. They could show that using a conventional ConvNet architecture could reach state-of-the-art performance on the ImageNet challenge dataset with network depth varying from 11 to 19 layers, their architecture is today known as *VGG*. In their work they refer to the CNN by Krizhevsky et al. as "the original architecture by Krizhevsky et al." which clearly shows the impact of the paper *ImageNet Classification with Deep Convolutional Neural Networks* [23].

The CNN architectures used for image classification are becoming deeper, as in very recent architectures such as GoogLeNet proposed in *Going Deeper with Convolutions* [44] and ResNet proposed in *Deep Residual Learning for Image Recognition* [19]. GoogLeNet is a 22 layer deep architecture and was submitted to ILSVRC-2014. The network was built up by *inception modules* which was a new type of network organization based on network-in-network structure. Each inception module forms a small network on its own and the main advantage of this architecture is that depth and width of the network was increased but still keeping the computational time constant [44]. More details on GoogLeNet are found in the paper *Going Deeper with Convolutions*. In ResNet the training process is optimized by learning residual functions that reference the layer inputs instead of completely unreferenced functions. This is practically done by using *shortcut connections* that skip one or more layers. Compared to VGG they are able to use 8x as many layers, up to 152 layer, in one ResNet. It was shown that the networks are easier to optimize than the corresponding number of layers in a more traditional architecture [19]. With an ensemble of residual networks they won the ILSVRC-2015 classification task with a 3.58% error rate on the ImageNet data set. The network has also shown great success in ImageNet detection, ImageNet localization, Common Objects in Context (COCO) detection and COCO segmentation challenges [19]. As ILSVRC the COCO datasets are publically available on the website [6].

Considering the more specific case of classifying medical images it was shown in *Anatomy-Specific Classification of Medical Images Using Deep Convolutional Nets* [35] in 2015 that CNNs are effective in classifying medical images based on the organ or body-part they depict. With images from 1675 different patients they achieve results with a classification error of 5.9%. The experiments are performed with a CT stack depicting a full torso.

Machine learning applications require a large amount of data, thus the limited amount of training data within the medical domain can possibly be a problem when training a CNN. In 2016 Cho et al. published the article *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?* [7]. In the study they classify six different body-parts depicted in 2D CT images using GoogLeNet. The experiment was performed for several different training data sizes and the result for each size was used to predict that 4092 images per class are needed to reach a classification accuracy of 99.5%

Previously it has been very common to use transfer learning when classifying medical images [45, 24, 6]. This is due to the significant amount of data and training time needed when fully training a CNN. Yet another study that investigates the reuse of features is *An Ensemble of Fine-Tuned Convolutional Neural Networks for Medical Image Classification* [24]. Kumar et al. investigated the possibility to combine features from multiple pre-trained networks and then fuse the results to predict classes for new images. With their experiments they showed that in some cases the ensemble of pre-trained CNNs achieve better accuracy than previous CNNs.

[6] mscoco.org

2.5.2 Object Localization and Detection with CNN

CNNs can also be used to detect and locate objects in images. For a simpler task where only one (or exact k, where k is a number of objects) object is to be located and classified per image the last fully connected layers can be replaced with a regression head for localization and a classification head for classification. The regression head outputs coordinates and the classification head outputs a class score for each image.

Sliding Window Approach

Another approach is to use a sliding window, meaning that the classification and regression is applied to multiple locations of different scales of the image. The sliding window approach is used by Sermanet et al. in the study *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks* [39]. They won the ILSVRC-2013 localization competition with error rate of 29.9% with their localizer that uses the classification scores and a proposed bounding box for each window. The bounding boxes were merged into a localization prediction containing the coordinates of a bounding box surrounding the object. To make the sliding window approach efficient the same convolutional features are used to produce two different outputs, one classification score and one bounding box prediction [39].

Sliding Window Efficiency

Running the CNN on many scales and many location in a sliding window fashion is a commonly used approach. It becomes very computationally expensive and inefficient if the entire pipeline is calculated for each window. Using a classifying CNN together with a sliding window approach is suitable according to Sermanet et al. [39]. The overlapping spatial regions that share many common computations allows to calculate these features once for the entire image and reuse them for each window location. It is also possible to convert the last fully connected layers into convolutional layers making it possible to calculate the probabilities for all windows for one scale in a single forward pass.

Region Proposals

If the number of classes per image is not exactly k, regression can no longer be used. The need for variable sized outputs call for object detection. Using classification to detect objects as done by Sermanet et al. [39] is a possibility, however the need to test many locations and scales calls for others solutions. One alternative to the sliding window approach is to use a subsystem to produce "region proposals" (sometimes referred to as "object proposals"), which are fed to the network. This approach is used in by Girshick et al. in the study *Rich feature hierarchies for accurate object detection and semantic segmentation* [15] where the algorithm Selective Search [46] is used to produce region proposals. There are also many other algorithms that can be used to produce region proposals. *What makes for effective detection proposals?* [22] presents a comparison of Selective Search and several other such algorithms.

Producing the region proposals is slow and therefore a bottleneck in a detection system. The approach was improved with *Fast R-CNN* [14] where Girshick introduced faster region proposals by using SPPnets (Spatial Pyramid Pooling Networks). These share the same expensive convolutional computation for each region proposal, thus reducing the total time spent on producing region proposals. Many applications do however require real-time localization of objects in images. Ren et al. [33] have improved Fast R-CNN even further by introducing Region Proposal Networks (RPN) with their work in *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. The actual localizing network, Faster R-CNN, and the RPN can be trained to share the same convolutional features and the whole system can be trained end-to-end in one stage. Using 300 proposals per image the detection system

manage to achieve a frame rate of 5 fps while getting state-of-the-art detection accuracy on PASCAL VOC 2007 and 2012 (73.2% and 70.4% respectively).

2.6 Data Representation

CT scans are commonly used to obtain image data of the human body for medical purposes. It uses x-ray technology to circle the patient's body with the x-rays and produces images that represent slices of the scanned body part, the slices make up what is called a CT stack [4]. A computer can then reconstruct the CT stack into 3D volume of the entire body which helps the radiologist in diagnosing potential disease [4]. The pixel data of an image is stored together with a header containing meta-data supplied by the scanner according to the DICOM standard.

The data used for this project is medical data stored as DICOM objects. A DICOM object has a list of attributes such as identifier, modality, patient identifier, number of images in the series, series identifier and many more. There is one special element containing the pixel data. For full list of elements see The DICOM Standard [31].

Series are sets of multiple images that are captured in the same examination, also referred to as stacks [4]. These stacks can be visualized in different ways, for example as 3D volumes.

Each stack is varying in size. Each DICOM object has an attribute called *slice thickness* which is the thickness in millimeter of a slice in a stack. The slice thickness is varying thus it is possible that it exists different amount of slices in stacks that depicts the same organ or body-part. Another DICOM attribute is the *spacing between slices* which is relevant when displaying stacks as 3D volumes [31].

2.7 Evaluation Metrics

In statistics accepting a null hypothesis that is false is called *Type-I error* and rejecting a null hypothesis that is true is called *Type-II error*. In the medical field these are usually referred to as *false positive* (fp) and *false negative* (fn) [27]. Correctly classified samples are either *true positive* (tp) or *true negative* (tn). Measures based on false positives and false negatives are useful for accuracy evaluation in classification. Accuracy and error rate adds up to 1 and are defined:

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{2.20}$$

$$Error\ rate = \frac{fp + fn}{tp + tn + fp + fn} \tag{2.21}$$

Van Rijsbergen [34] presents definitions of precision and recall as follows:

$$Recall = \frac{tp}{tp + fn} \tag{2.22}$$

$$Precision = \frac{tp}{tp + fp} \tag{2.23}$$

Calculating the weighted harmonic average of precision and recall can be done through F-score [34].

$$F_\beta = \frac{(1 + \beta^2) \cdot tp}{(1 + \beta^2) \cdot tp + \beta^2 \cdot fn + fp} \tag{2.24}$$

Where β is a real positive number used to adjust the emphasis put on false negatives, that is β times more importance of recall than precision [34]. F-score ranges between 0 and 1, it is desirable to get a high F-score.

When localizing objects in images with bounding boxes the predicted bounding box (B_p) has to be evaluated against a bounding box representing the ground truth (B_{gt}). A commonly used evaluation metric to measure how well the localizer has predicted is Intersect over Union (IoU) calculated as:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{2.25}$$

IoU is used in the PASCAL VOC Challenge where an overlap $\geqslant 0.5$ is judged as a true positive [12].

3 Method

The approach used to solve the task is to use pre-trained CNNs. This Chapter begins with an overview of the frameworks and hardware used. It covers important information about the used dataset and describes the necessary pre-processing steps that have been performed. A motivation to the choices and details about the CNN implementation are covered. Finally, the experiments coducted to answer the research questions and the evaluation of the experiments are described.

3.1 Frameworks, Platforms and Hardware

The CNNs were implemented in Keras [8], a high level library for neural networks. Keras is written in Python and runs on top of Tensorflow[1] or Theano[2], in this work Tensorflow was used. The most important reason for using Keras is that it supports convolutional networks and it is used by Sectra in other deep learning projects. Additionally Keras can run both on CPU and GPU. All the training was however done on GPUs using the parallell computing platform CUDA[3]. To accelerate computation for deep neural networks cuDNN[4], which is a part of the NVIDIA Deep Learning SDK was used. The graphics card used for training was NVIDIA GeForce GTX 1060 with 6GB memory and NVIDIA GeForce GTX 1080 with 8GB memory.

3.2 Dataset

Sectra provided an unlabelled dataset with medical images. Medical images are standardized according to DICOM and there are regulations regarding their quality, these properties make them more suitable for machine learning than natural images [7].

The given data was anonymized to the extent that an anonymized patient id was given each patient, the patient age was not anonymized. The patient's age was kept since the anatomy of children and adults differ and the age information can aid when selecting which data to consider.

[1] Tensorflow, https://www.tensorflow.org/
[2] http://deeplearning.net/software/theano/
[3] CUDA, http://www.nvidia.com/object/cuda_home_new.html
[4] cuDNN, https://developer.nvidia.com/cudnn

The dataset consisted of CT stacks stored in DICOM format. In total there were 129 CT stacks distributed over 52 different patients. Stacks can depict an entire body or just part of the body. The dataset contained stacks of different sizes depicting different parts of the body and it was ensured that all relevant organs and body-parts are depicted in several stacks. The relevant organs and body-parts are the classes used for classification:

- 10 coarse body-parts: Head, Neck, Shoulders, Chest, Abdomen, Pelvis, Femur, Knee, Tibia and Ankles/Feet
- 5 additional finer organs: Right lung, Left lung, Liver, Left kidney, Right kidney

The coarse classes were chosen to cover all the entire body (except the arms). The finer classes were chosen since they are located in areas of the body where many organs exist. The total number of stacks containing a specific class is found in Table 3.2. The implemented CNNs were fed with 2D images, the slices in the CT stacks. In order to make pre-processing and labelling more efficient the CT stacks were initially treated as 3D volumes and were not converted to 2D images until the labelling was completed.

3.2.1 Labelling

The labelling was done using Sectra's PACS by extending a part of a 3D tool used to create boxes in volumes to save these boxes as label-files. Figure 3.1 shows a screen shot of the software used to create labels.

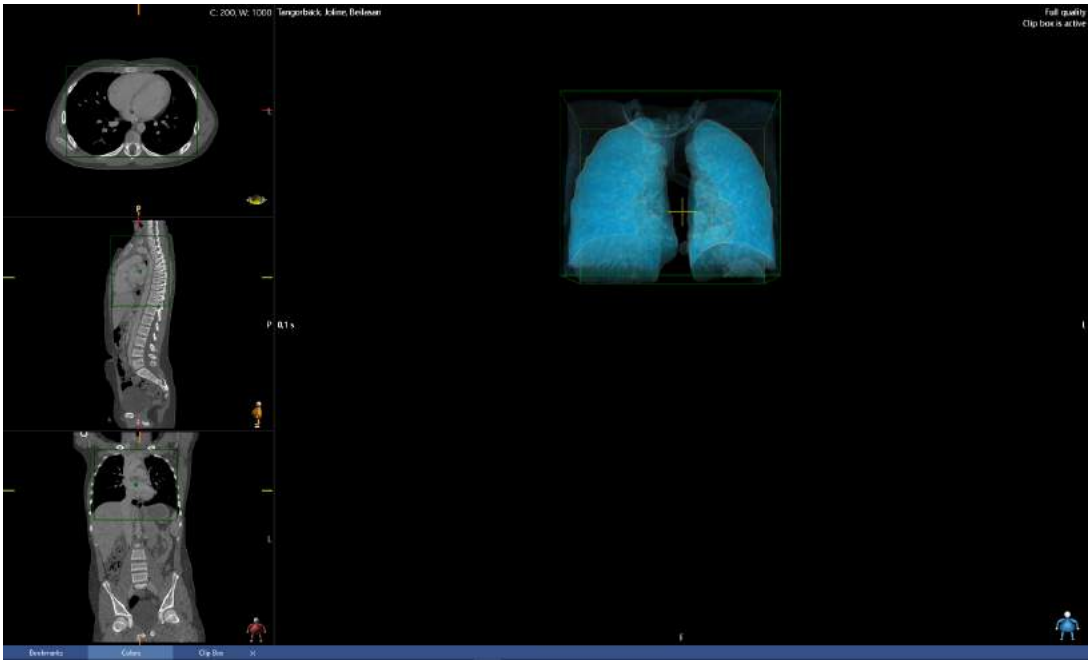


Figure 3.1: A screenshot of the tool used for labelling. The right side is a 3D representation of the stack and the three images to the left shows the stack from three different 2D projections. The green box is the box where the coordinates are transferred to a label file. This example box surrounds both lungs.

Since the labelling was done by the authors and not radiology experts organ maps of the body were used to recognize and locate the relevant organs and body-parts. Sectra representatives with more experience in working with CT images were also consulted to verify the authors knowledge of these specific organs and body-parts. Each label-file consists of a unique id

associating the label with a pixel file, coordinates representing the corners of the box and one of the classes.

There were two alternatives when the boxes that mark a class (an organ or body-part) were created in 3D, either make the label box containing or contained. A containing box had to be created around the widest part of the class in order to contain the entire organs/body-parts . The approach could result in parts of other organs or body-parts to appear in the periphery of the label box. The alternative approach was to use contained boxes, in that case the entire label would reside within the organ or body-part. This approach would guarantee that only the labelled class resides within the label box but would result in large portions of the organ or body-part to reside outside of the label box. For this study the first option was used.

3.2.2 Preprocessing

The implemented system was fed with one raw pixel file per CT stack containing the pixel data, an associated xml-file containing relevant meta-data from the DICOM headers and the label files described in the previous section. The meta-data file contained:

- **A unique identifier for the CT stack**
  The dataset was split into a training set and test set where the training set contained approximately 80% of the data and the test set contained approximately 20% of the data as recommended in the book *Deep Learning* [17]. The stack identifier was used to perform this split by stack. The split should preferably be done by patient to reflect a realistic situation where a new patient has an examination and the input to the system is completely unseen images. However the limited number of patients would require a too large portion of the data to reflect all classes in the test set if the split is done by patient. Instead the split was done by stack, when a new stack is captured the detector will not have seen parts of that data during training. It was also ensured that all classes were represented in the test set.

- **The patient's age**
  All stacks with a patient younger than 18 years were excluded from the dataset since the anatomy of children and adults differs and only adults were to be considered in this study.

- **The dimensions of the stack**
  The dimensions of the stack are used to reshape the raw-pixels which were imported as a 1-dimensional array and to filter out images with a larger ratio than $\frac{1}{3}$ between width and height. Lastly used to make all images quadratic as part of the pre-processing pipeline. An additional pre-processing step was also used to resize the images to 224x224 pixels, this additional pre-processing step is described in more detail in Section 3.3.1.

- **The orientation of the volume in patient space**
  All images stored according to the DICOM standard are related to the orientation of the human body in the scanner according to the patient space coordinate system, as illustrated in Figure 3.2. Only images aligned along the z-axis in patient space were extracted in the pre-processing step, this projection is in medical terms often referred to as the axial plane.

Figure 3.2: A CT scanner with a patient and the DICOM Patience Space Coordinate System. [5]

- **Scaling in mm/px for each dimension of the volume**
  The scaling parameter for each image gives the number of mm represented by one pixel in both dimensions of the 2D image. When the scaling was analyzed for all images in the dataset it could be concluded that all images had the same scaling in both dimensions. The smallest scaling that appeared in the dataset was $0.5mm/px$ and the largest $2.68mm/px$.

  The smallest organs considered in this study is the kidneys, see Table 3.1. In a 2D image they measure approximately $70x70mm$. In the $224x224px$ large images smallest scaling would result in the smaller of the dimensions of a kidney to be represented by $25px$. The scaling was however only considered to the extent that it was analyzed in the preprocessing step. Possible effects on the result caused by the different scaling of the images are discussed in more detailed in the Discussion in Chapter 5.

Class	Max scale (mm/px)	Min scale (mm/px)	~Size (mm)	Max size (px)	Min size (px)
head	1.339	0.732	163x182	223.0	122.0
neck	2.679	0.5	107x95	190.0	35.0
shoulders	2.679	1.246	360x197	158.0	74.0
chest	2.679	1.089	339x247	227.0	92.0
right lung	2.679	1.089	127x192	117.0	47.0
left lung	2.679	1.089	117x183	107.0	44.0
abdomen	2.679	1.089	333x261	240.0	97.0
liver	2.679	1.089	172x181	158.0	64.0
right kidney	2.679	1.089	71x66	61.0	25.0
left kidney	2.679	1.089	68x67	62.0	25.0
pelvis	2.679	1.089	351x254	233.0	95.0
femur	2.232	1.339	364x204	152.0	91.0
knee	2.232	0.61	234x147	241.0	66.0
tibia	2.232	1.339	284x125	93.0	56.0
ankles/feet	2.232	1.339	253x212	158.0	95.0

Table 3.1: The minimum and maximum scale in mm/px for an image that has been resized to 224x224 pixels. Approximate size of an organ/body-part in mm and the maximum and minimum size for each class in px. The organ/body-part sizes are calculated by taking the average size of the class in all samples in the training dataset.

[5] *CT Scanner* by BotMultichillT licensed under CC-PD-Mark

3.2.3 Resulting dataset

The volumes were converted into 2D images and the labels were translated into label vectors. All images without any label were filtered out. The resulting dataset contained:

Class	# slices	# stacks
head	4220	24
neck	3673	24
shoulders	1392	39
chest	5014	41
right lung	6237	36
left lung	6227	29
abdomen	5268	21
liver	3445	23
right kidney	2334	18
left kidney	2593	18
pelvis	5234	52
femur	5528	51
knee	3077	54
tibia	4523	36
ankles/feet	2823	41

(a) Training set

Class	# slices	# stacks
head	897	3
neck	994	4
shoulders	131	4
chest	1104	7
right lung	1229	7
left lung	1230	9
abdomen	830	7
liver	655	6
right kidney	232	5
left kidney	218	5
pelvis	1706	8
femur	1948	8
knee	730	9
tibia	1714	5
ankles/feet	1005	5

(b) Test set

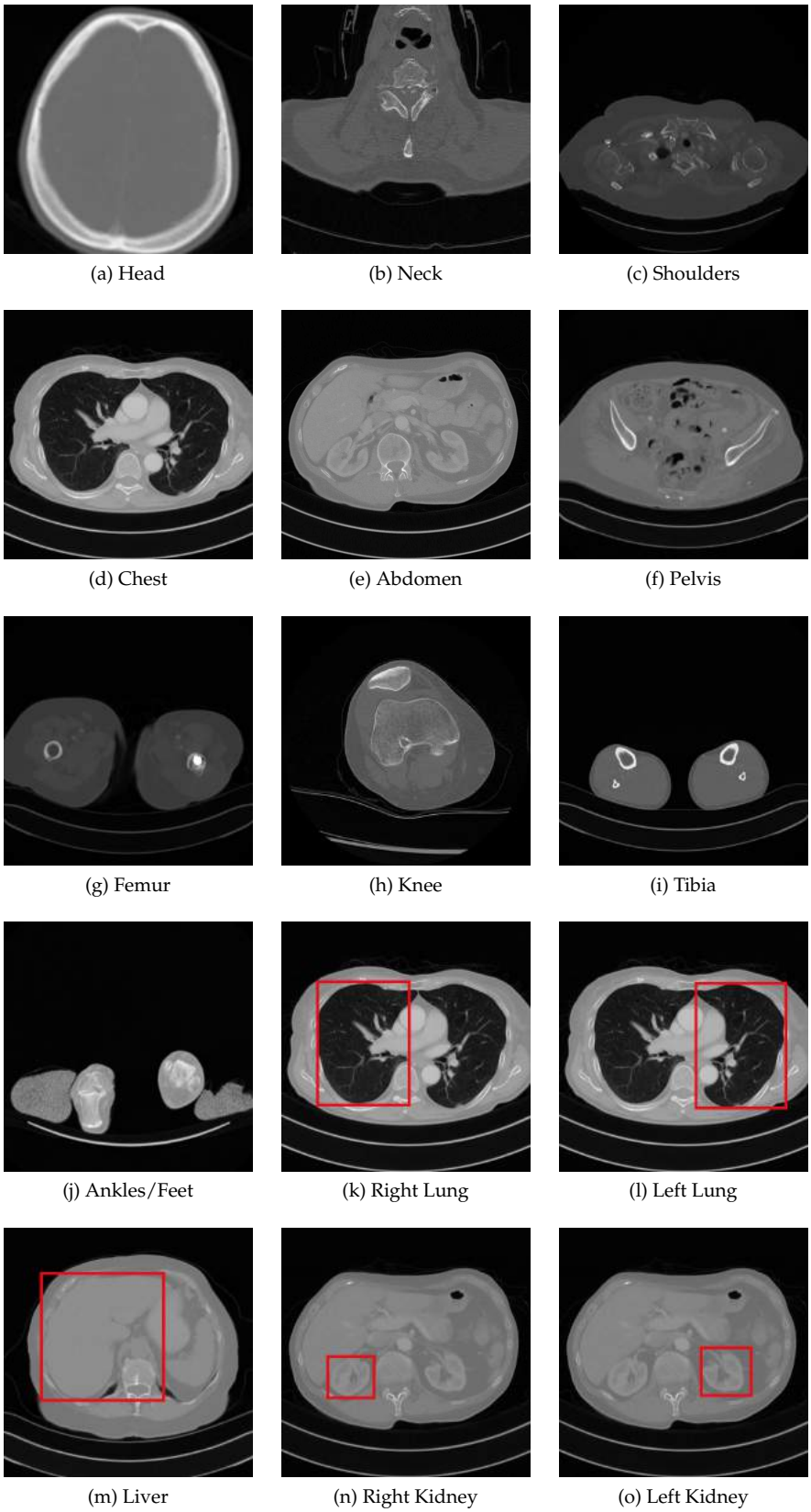
Table 3.2: The number of slices and the number of stacks containing each class in the training dataset and the test dataset.

The total number of stacks and slices in the two datasets are found in Table 3.3. It should be noted that the presence of several classes in the same slice/stack causes the number of slices/stacks per class to not add up to the total amount of slices/stacks.

Set	Total number of slices	Total number of stacks
Training	41394	108
Test	11036	21

Table 3.3: The total number of slices and stacks in the two datasets.

The results from the article *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?* [7] by Cho et al. in 2016 were used as guideline when collecting and labelling data. They showed that with around 4000 images per class a classification accuracy of 99.5% could be reached. Figure 3.3 shows examples of images belonging to each class.



(a) Head (b) Neck (c) Shoulders

(d) Chest (e) Abdomen (f) Pelvis

(g) Femur (h) Knee (i) Tibia

(j) Ankles/Feet (k) Right Lung (l) Left Lung

(m) Liver (n) Right Kidney (o) Left Kidney

Figure 3.3: Sample images for all classes.

3.3 Classification

To be able to evaluate a classifier, a CNN was implemented in Keras. The classification problem was of the type multiclass-multilabel, meaning that classification is done over multiple classes and each sample can contain multiple labels.

Due to the limited amount of data available for training, a transfer learning approach was chosen as discussed in Section 2.4.3. The classifier was implemented using a pre-trained base network, reusing the weights from all the convolutional layers. The final layers were then added and fine-tuned on top of the base network.

The classifier was implemented by using a VGG16 architecture to form a base network [40]. As discussed in Section 2.5, VGG is a set of networks and the numbers in the name of the network specifies how many layers the network has. Other popular networks used as base networks when applying transfer learning are GoogleNet [44] and AlexNet [23]. In ILSRVC-2014, VGG16 finished in the top with GoogleNet and beat the results of AlexNet from 2012 [40]. This in combination with the fact that there exists pre-trained implementations of VGG16 in Keras made it a suitable choice for this project. The implementation for this study was based on the VGG16 implementation in *Very Deep Convolutional Networks for Large-Scale Image Recognition* [40].

The network has a 16 layer deep architecture of which 13 (block 1-5 in Table 3.4) have been used as implemented in VGG16 and instead of the last three fully connected layers in the original implementation by Simonyan et al. [40], two fully connected layers adjusted to the specific classification task were used. The architecture details are presented in Table 3.4.

block	**1**	**2**	**3**	**4**	**5**	**6**	**7**
type	conv+max	conv+max	conv+max	conv+max	conv+max	fc	fc
# conv layers	2	2	3	3	3	-	-
# filters	64	128	256	512	512	256	15
conv stride	2x2	2x2	3x3	3x3	3x3	-	-
zero-pad size	1x1	1x1	1x1	1x1	1x1	-	-
pool size	2x2	2x2	2x2	2x2	2x2	-	-
pool stride	2x2	2x2	2x2	2x2	2x2	-	-
reg	-	-	-	-	-	L^2	-
input shape	(224, 224, 3)	(112, 112, 64)	(56, 56, 128)	(28, 28, 256)	(7, 7, 512)	(1, 25088)	(1, 256)
output shape	(112, 112, 64)	(56, 56, 128)	(28, 28, 256)	(7, 7, 512)	(1, 25088)	(1, 256)	(1, 15)

Table 3.4: The architecture of each block of the implemented CNN.

In Table 3.4 the **type** is either a convolutional block containing a few convolutional layers and is ended with a max pooling layer (conv+max) or a fully connected layer (fc). **# conv layers** tells how many convolutional layers a block contains. **# filters** is the number of filters used in each convolutional layer in the the block (i.e. the third dimension of an activation volume). **Conv stride** described the stride for the convolutional operation of the convolutional layers in the block. Each convolutional layer is preceeded with a *zero-padding* layer, **zero-pad size** tells how many rows and columns of zeros that are added to the input. Each block ends with one *max pooling* layer and **pool size** and **pool stride** in the table describes the settings for the max pooling operation. More information about the different layers and their settings can be found in Section 2.1.1.

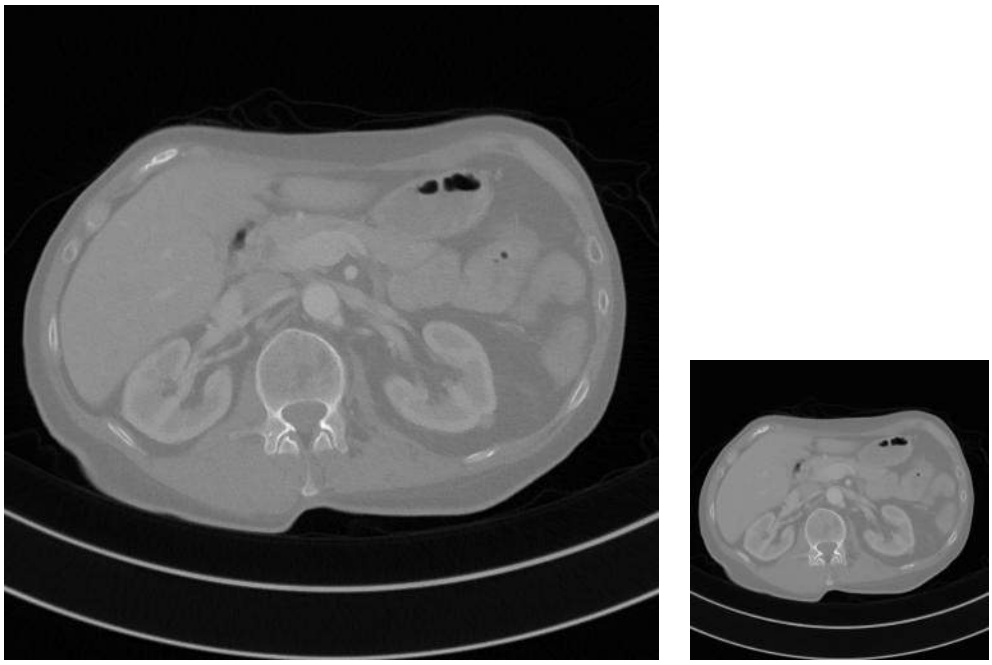
Each convolutional layer was preceeded by zero padding 1x1, adding one row and one column of zeros to the input to the layer and followed by a *max pooling* layer with filter size 2x2 and the stride 2 in both dimensions as in Figure 2.5. The image dimension was reduced to half the size by this pooling operation. Between the two fully connected layers there is a dropout layer with dropout probability 0.5.

The activation function used for each convolutional layer in the VGG16 architecture is ReLU, Equation 2.3, which is the most common in convolutional architectures according to LeCun et al. [25]. The last fully connected layer used a sigmoid activation, as seen in Equation 2.6.

The pre-trained weights loaded into the base network were saved when training the network on the ILSVRC-2014 dataset. The fully connected layers were initialized with a distribution in Keras called *glorot_normal* which was implemented as explained in Section 2.2.

3.3.1 Dataset

To be able to benefit of the existing weights for the pre-trained network the input shape had to be the same as for the original network, hence the 2D images used as input were resampled to 224x224 pixels in an additional pre-processing step. Figure 3.4 shows an example of an image before and after the resampling. VGG16 expects the pixels in the input data to be in a range 0-255 [40]. All image data was converted into this range by converting each pixel value into the new range based on the old minimum and maximum pixel value in the image. Simonyan et al. used a normalization where the mean RGB value was subtracted from each channel in the original work with VGG16 [40]. To maximize the performance of VGG16 when extracting features for the input data normalization was done in the same way as when the network was pre-trained.



(a) Original size (b) Resized to 224x224

Figure 3.4: Example of relative size between original and rescaled image.

Before fine-tuning 20% of the stacks in the training dataset were extracted into a validation set, the remaining 80% remained in the training set, see Figure 3.5. The split resulted in 10% of the individual images in the original training set to belong in the validation set and 90% in the new training set.

Figure 3.5: Illustration of dataset split. 20% of the stacks in original dataset have already been extracted to a test set. The remaining data in the training dataset is split, creating a validation dataset of 20% of the stacks and a new training dataset with 80% of the stacks.

Roth et al. used data augmentation techniques in *Anatomy-Specific Classification of Medical Images Using Deep Convolutional Net* [35] to improve the classification result. Keras provides the possibility to do data augmentation on the input using a *ImageDataGenerator*. Zooming was applied randomly within the interval of $[0.8, 1.2]$, resulting in images that are zoomed in or out $0 - 20\%$. Shear angle was applied with value $0.2rad$, giving sheared images within the interval of $-0.2rad$ to $0.2rad$. Randomly horizontal flips to the input were also used. This data augmentation was applied to the training data, no augmentation was applied to the validation data. Two different augmentations were applied per image for the training dataset, resulting in an augmented training dataset that is twice as large as the original training dataset.

3.3.2 Fine-tuning the Network

The pre-trained network was fine-tuned by freezing the weights of the 13 first layers, fine-tuning only the fully connected layers. Since the weights of the frozen layers were never adjusted during training the training time could be reduced by only predicting the feature maps for training and validation data once. This way the base network is used as a feature extractor and the computed feature maps could then be used as input training and validation data to the first non-frozen layer when fine-tuning the network.

Objective

The CNN was compiled using *binary_crossentropy*[6] objective in Keras. The objective $J(\Theta)$ is calculated with logistic loss as defined in Equation 2.14. In this setting when the probability distribution is between 0 and 1, the cross-entropy is the same as logistic loss as shown in Equation 2.14.

It is appropriate to use a *binary_crossentropy* objective as loss function since it computes sigmoid cross entropy, i.e. it measures the probability error where each class is independent and not mutually exclusive.

Optimizer

During fine-tuning, the model was compiled using the optimizer SGD (Equation 2.15) as implemented in Keras. The implementation was a mini-batch gradient descent. The size of the mini-batches was set to 32 (i.e. the batch size parameter in Keras was set to 32) which was a recommended default value for the batch size according to Bengio [1]. The momentum used with SGD was 0.9 which was used in previous work where a model was fine-tuned [48, 45]. A value between 0.8 and 0.9 was also referred to as commonly used value by Widrow et al. [47]. A learning rate decay schedule was used where $\eta_{new} = 0.95 \cdot \eta_{prev}$ where η_{prev} was the learning rate used in the previous epoch.

[6]Source code for objective functions https://github.com/fchollet/keras/blob/master/keras/objectives.py

Number of Iterations

The network was fine-tuned during 50 epochs, iterating the augmented training data once per epoch. However early stopping was implemented with $patience = 5$ on the validation loss after the 25th epoch had completed. Early stopping is implemented in multiple ways. When the training has terminated the model that was saved was the model after the epoch for which the best validation loss was observed. Meaning that the model can be saved prior to 25 epochs. The reason for forcing the model to train during 25 epochs before terminating is to avoid a few unlucky weight updates during the first few epochs to stop the model from continue training, giving the model a chance to improve. The limitation to train during 50 epochs was mainly used to avoid overfitting, training the model during too many epochs with the same training data can cause the model to overfit. Secondly it was used to ensure that training eventually stops due to time limitations. It takes approximately 8 hours for one model to train 50 epochs.

L2-regularization is not useful when applying Early Stopping according to Bengio [1], however early stopping was not used until epoch 25. L2-regularization was used to reduce the risk of overfitting during the first 25 epochs.

Parameter	Value
Optimizer	SGD
Loss	binary_crossentropy
Batch size	32
# Epochs	50 (with early stopping)
# Training samples	37051
# Training samples after augmentation	74102
# Validation samples	4343
# Validation samples after augmentation	4343
Learning rate decay	0.95
Momentum	0.9

Table 3.5: Fixed settings for fine-training the network.

3.4 Localization and Detection

The approach used in *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks* by Sermannet et al. [39] was used as inspiration when investigating the possiblity to locate organs and body-parts in the 2D images. They use a sliding window approach where they let windows of different scales slide over the original image and the content of the window at each step is the input to their CNN. They have trained a CNN with two different heads that share the same convolutional blocks, one classification head and one regression head. A class prediction score is produced by the classification head and a bounding box (four coordinates) are produced by the regression head on each window for each scale. Bounding boxes from windows where classification confidence is high are then merged using an algorithm described in their paper to produce a final prediction.

In this study one part of their pipeline has been implemented to investigate the possibility to locate or detect organs and body-parts with this approach in medical CT images. The part of their pipeline that has been implemented is a classifying CNN that is used in a sliding window manner. The window slides over the image and each part is passed as input to the CNN. The CNN produces confidence score for the presence of each class in that window.

The CNN architecture was identical to the CNN used for the classification task and was trained with the same settings. The difference is the training data that was passed to the network. Instead of passing the entire 2D image for each sample patches were extracted

based on the labels in the original samples. The CNN requires the input to be quadratic and $224x224px$. For labels that are not quadratic pixeldata outside the smaller dimension of the label was added to match the larger dimension. Finally each patch was rescaled to $224x224px$.

When running inference on the CNN it is applied to windows of the image and one consideration is the size of these windows. Since the goal is to detect different sized organs and body-parts the likelihood to find each class is larger if multiple sizes of the sliding window is used [39]. Based on the sizes in *mm* of the classes presented in Table 3.1 the sizes s are chosen.

$$s = \{\, 50, 70, 100, 150, 200, 250, 300, 350 \,\} \tag{3.1}$$

The window has a step-size, determining how many pixels at the time it should slide in both dimensions. Chosing a too large step-size will result in predictions that are not detailed enough while choosing a too small step-size will give redundant information. A sliding window approach is very computationally expensive with many models but is suitable to use with CNN. The nature of convolutional calculations is to apply the same calculation to the entire image which can be calculated once and reused for each window [39].

3.5 Evaluation metrics

In Section 2.7 the evaluation metrics were defined in a simple form. A form that is not applicable to multi-class multi-label problems. To enable the use of the metrics in the setting with multi-class multi-label task Sokolova et al. [41] presents the following modification to the metrics.

Average accuracy and error rate was defined as follows:

$$\textit{Average Accuracy} = \frac{\sum_{i=1}^{k} \frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i}}{k} \tag{3.2}$$

$$\textit{Average Error Rate} = \frac{\sum_{i=1}^{k} \frac{fp_i + fn_i}{tp_i + fn_i + fp_i + tn_i}}{k} \tag{3.3}$$

k is the number of classes. This average accuracy determines the average per-class effectiveness of the classifier.

Sokolova et al. also introduce a metric called Exact Match Ratio which considers the case of complete label matching. The metric gives the accuracy where each prediction vector L_j is considered one unit, I is the indicator function and n is the number of label vectors. L_j^d are the ground truth label vectors and L_j^c are the label vectors produced by the classifier.

$$\textit{Exact Match Ratio} = \frac{\sum_{j=1}^{n} I(L_j^d = L_j^c)}{n} \tag{3.4}$$

In *The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets* Saito et al. is investigating metrics evaluating a classifier when the dataset is imbalanced, i.e $fp + tp \neq fn + tn$ [38]. In the case of an imbalanced dataset Saito et al. noted in their experiments that precision reveals differences in performance that would be unnoticed if accuracy was used without precision. Precision, recall and F-score were evaluated in two ways, macro-averaging (denoted with M) or micro-averaging (denoted with μ). Macro-averaging takes the average over the classes for a measure that is calculated the same way for each class. The alternative is to sum the number of tp, tn, fp and fn for each class and calculate the metric from the cumulative values. Thus, micro-averaging puts a bias on larger classes. Precision, recall and F-score for macro-averaging are defined [41]

$$Precision_M = \frac{\sum_{i=1}^{k} \frac{tp_i}{tp_i + fp_i}}{k} \tag{3.5}$$

$$Recall_M = \frac{\sum_{i=1}^{k} \frac{tp_i}{(tp_i + fn_i)}}{k} \tag{3.6}$$

$$Fscore_M = \frac{(\beta^2 + 1) \cdot Precision_M \cdot Recall_M}{\beta^2 \cdot Precision_M + Recall_M} \tag{3.7}$$

$$Precision_\mu = \frac{\sum_{i=1}^{k} tp_i}{\sum_{i=1}^{k} (tp_i + fp_i)} \tag{3.8}$$

$$Recall_\mu = \frac{\sum_{i=1}^{k} tp_i}{\sum_{i=1}^{k} (tp_i + fn_i)} \tag{3.9}$$

$$Fscore_\mu = \frac{(\beta^2 + 1) \cdot Precision_\mu \cdot Recall_\mu}{\beta^2 \cdot Precision_\mu + Recall_\mu} \tag{3.10}$$

For the experiements $\beta = 1$, to put equal weight on recall and precision.

The output from the system is a probability vector y. To get a true or false prediction a threshold on the probability must be used for each class i. In this study the same threshold is used for all classes. This threshold is set to:

$$y_i = \begin{cases} 1 & y_i \geqslant 0.5 \\ 0 & y_i < 0.5 \end{cases} \tag{3.11}$$

3.6 Experiments

Four experiments were conducted, the first experiment: tuning a CNN, the second experiment: creating a "table of content" for CT stacks based on CNN classifications and the third experiment examining the importance of the size of training dataset. Finally, the fourth experiment investigates how a classifying CNN can be used in a sliding window manner to localize organs within single CT images.

3.6.1 Tuning a classification CNN

Some choices of the settings for the model were already set in Section 3.3.2, the remaining hyperparameters were tuned as a first experiment.

Since the learning process was extremely time consuming only two hyperparameters were chosen for optimization, the initial learning rate and the L2-regularization coefficient. If time is limited Bengio, [1] recommends to prioritize optimizing the initial learning rate as the most important hyperparameter. When the training dataset is small CNNs are more prone to overfitting [5], therefore the weight regularization is chosen as the second hyperparameter to optimize since it aids to reduce overfitting.

The tuning was done with grid search as presented in 2.3.2. Although Bergstra et al. found that random search is more effective than grid search in the work *Random Search for Hyper-Parameter Optimization* [2] grid search is chosen for optimization. Their study was conducted on a 32-dimensional configuration space and they state that for low dimensional configurations (up to three hyperparameters) grid search is reliable. It is also stated by Bengio [1] that grid search is more efficient only when the number of hyperparameters to optimize are below 2 or 3. Grid search is therefor reliable and efficient choice for optimizing the learning rate and regularization coefficient.

The following values for initial learning rate η and regularization coefficient λ were chosen:
η: 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}
λ: 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 0.0

Model	η	λ
1	10^{-2}	10^{-1}
2	10^{-2}	10^{-2}
3	10^{-2}	10^{-3}
4	10^{-2}	10^{-4}
5	10^{-2}	0.0
6	10^{-3}	10^{-1}
7	10^{-3}	10^{-2}
8	10^{-3}	10^{-3}
9	10^{-3}	10^{-4}
10	10^{-3}	0.0
11	10^{-4}	10^{-1}
12	10^{-4}	10^{-2}
13	10^{-4}	10^{-3}
14	10^{-4}	10^{-4}
15	10^{-4}	0.0
16	10^{-5}	10^{-1}
17	10^{-5}	10^{-2}
18	10^{-5}	10^{-3}
19	10^{-5}	10^{-4}
20	10^{-5}	0.0
21	10^{-6}	10^{-1}
22	10^{-6}	10^{-2}
23	10^{-6}	10^{-3}
24	10^{-6}	10^{-4}
25	10^{-6}	0.0

Table 3.6: Combinations of hyperparameters for each of the trained models.

The learning rates in the experiment were chosen by letting the network train during a few epochs and analyze the behaviour for the edge cases. The other parameter was fixed while finding the edge cases which is a common approach according to *Practical Recommendations for Gradient-Based Training of Deep Architectures* by Bengio [1]. It was found that with a lower learning rate than 10^{-6} convergence is very slow. With a higher learning rate than 10^{-2} loss does not decrease on every epoch and may not converge, 10^{-3} is the recommended initial learning rate when training a CNN according to Bengio. The lower bound for regularization, 0.0, means no regularization at all. The higher bound 10^{-4} was chosen by observing the loss and noticing that with larger regularization the regularization term dominated over the weights in the network.

Evaluation

To evaluate the result of the experiment the model, validation accuracy and validation loss were analyzed. The model with the lowest validation loss was chosen as the best model. This model were used to make predictions on the previously unseen test set. For the predictions the following metrics, as defined in Section 3.5, were calculated:

- Average accuracy
- Average error rate
- Micro and macro average recall
- Micro and macro average precision
- Micro and macro average F-score

To extend the examination of the chosen model, the misclassifications done by the model were analyzed. The analysis considered what class was predicted instead when a sample was misclassified for some class i, it was done per class. For each false positive predicted for class i the probability vector was scanned for false negatives on other classes and for each false negative predicted for class i the probability vector was scanned for false positives on other classes. By doing this it was possible to get an idea if anatomically adjacent classes were predicted instead of the correct classes when the sample was misclassified.

A new term *relevant misclassification* was introduced. A prediction that was wrong was considered a relevant misclassification if it instead predicted an anatomically adjacent class as defined in Table 3.7.

Class	Relevant Misclassification
head	neck
neck	head, shoulders
shoulders	neck, chest
chest	shoulders, abdomen
right lung	chest, left lung
left lung	chest, right lung
abdomen	chest, pelvis
liver	abdomen, left kidney, right kidney
right kidney	abdomen, liver, left kidney
left kidney	abdomen, liver, right kidney
pelvis	abdomen, femur
femur	pelvis, knee
knee	femur, tibia
tibia	knee, ankles/feet
ankles/feet	tibia

Table 3.7: Relevant misclassifications for each class.

For each class the percentage of misclassifications that was relevant misclassifications was calculated. The measure was also calculated for the case when the misclassification was a false positive and a false negative separately. The percentage of cases where no other class was predicted was also calculated.

Finally, since the classifier handles multiple classes per sample a value for Exact Match Ratio is calculated. This value gives the percentage of prediction vectors that are identical to the ground truth vector.

3.6.2 "Table of content" for CT stacks based on CNN classifications

In the previous experiment only classification of individual 2D images was considered. To generalize and put the classifier in context, seven full stacks were selected:

- One full body stack, covering all classes except head
- Two upper body stacks, covering classes in the upper body except head
- Two head stacks, covering the head.
- Two lower body stacks, covering all classes from pelvis and down.

Of these seven stacks, four stacks were selected from the test set and three were acquired for this test. The new stacks were gathered due to the lack of full body stacks in the test set and to ensure stacks of new patient's were presented to the classifier. A similar experiment has been done by Roth et al. in *Anatomy-Specific Classification of Medical Images Using Deep Convolutional Net* [35].

The best performing classifier from the previous experiment was used in this experiment. During pre-processing an index for each image in the stack was stored to make it possible to know the order of the axial slices seen from top to bottom of the body. Each image in the stack was fed to the trained classifier which outputted a probability vector containing the predicted probabilities for each class for that sample.

Evaluation

For each class a probability curve was plotted along the body. This was done for each class by extracting the desired class probabilities from every image prediction vector. The result was analyzed to see whether the odd probabilities appear in completely wrong places or if the probabilities generally are high around the location of the class. Furthermore suggestions on heuristics were discussed to explore the opportunity to enhance the result by looking at other images' predictions in the stack to see if they could aid and correct a misclassification.

3.6.3 Reducing the size of the training dataset

It remained an open question if a larger original training set would likely improve the performance in terms of accuracy, precision and recall of the classifier. Instead of adding more data to the training dataset the training set was split into several smaller training sets to investigate this. The test set was not modified at all.

To extend the training dataset it would be required to add additional stacks, no more slices can be obtained from the stacks that are already included in the training set. To reflect this realistic situation entire stacks were removed to create the smaller training sets, the percentage of stacks in each training set is found in Table 3.9.

The stacks to be included in each training set were randomly selected among all stacks but it was ensured that all classes were represented in each of the smaller training sets. It was also ensured that the smaller training sets always were a subset of the larger. Selecting random stacks from the original training set can however result in very different number of images to belong to the different sets since stacks can have different size (i.e. different amount of slices). To handle this sets containing from 10% to 100% of the stacks were created with an interval of 5%, the resulting sizes of the sets is displayed in Table 3.8

Dataset	Number of stacks	Number of slices	% of original stacks	% of original slices
1	8	6086	10%	16%
2	13	7268	15%	19%
3	17	12770	20%	34%
4	22	18607	25%	50%
5	26	21949	30%	59%
6	30	22994	35%	62%
7	35	25298	40%	68%
8	39	26775	45%	72%
9	44	28287	50%	76%
10	48	29835	55%	80%
11	52	30980	60%	83%
12	57	33218	65%	89%
13	61	34284	70%	92%
14	66	35029	75%	94%
15	70	35558	80%	95%
16	74	35846	85%	96%
17	79	36161	90%	97%
18	83	36781	95%	99%
19	88	37051	100%	100%

Table 3.8: The number and percentage of stacks and slices in each of the reduced training sets.

To get spaced values between 0% to 100% for the percentage of slices in the data training set the sets shown in Table 3.9 were selected to be used for the experiment.

Dataset	Number of stacks	Number of slices	% of original stacks	% of original slices
1	8	6086	10%	16%
2	13	7268	15%	19%
3	17	12770	20%	34%
4	22	18607	25%	50%
5	26	21949	30%	59%
7	35	25298	40%	68%
9	44	28287	50%	76%
11	52	30980	60%	83%
15	70	35558	80%	95%
19	88	37051	100%	100%

Table 3.9: Percentage of original stacks in each new training set

One model was trained with each of the training datasets using the settings that were found to give the best model in the first experiment. A similar experiment was conducted by Cho et al. in *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?* [7].

Evaluation

To evaluate the outcome of the experiment the trained CNNs were used to make predictions on the test set. For the predictions the following metrics were calculated as introduced in Section 3.5):

- Average error rate
- Micro and macro average recall
- Micro and macro average precision

By analyzing the values of these metrics as the training set gets larger it can be predicted whether adding additional stacks to the training set would improve the performance in terms of the calculated metrics of the trained CNN.

3.6.4 Towards object localization

To analyze the possibility to use a classifying CNN to locate organs and body-parts in medical CT images the CNN described in Section 3.4 was trained. A subset of the test set containing at least 10 samples of each class was extracted from the original test set. Each class was represented by at least three different stacks.

The test images were rescaled to make the sliding window sizes in mm be represented by $224px$. The rescaled image size depends on the original scaling (mm/px) and size of the image. For the experiment a step-size of $30px$ was used for both dimensions as the window slid over the image. The number of windows per scale depends on the size of the rescaled image.

The output when the entire image was covered was a heatmap with per-pixel likelihood of the presence of a specific class. The per-pixel likelihood was calculated by taking the average confidence score for each window that has convered that region (pixel) of the image.

The bounding box extraction approach differs from the approach in *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks* where a regressor was used to propose one bounding box per window and then merge these over all windows and all scales.

The approach used for this investigation was instead to extract bounding boxes based on the per-pixel likelihood heatmaps. A threshold confidence score and the minimum and maximum pixels in both dimensions that are above the threshold are set as limits for the extracted bounding box. Three different parameters with two values each have been tested for the extraction process.

Average Heatmap vs. Merged Boxes
The first parameter determines if an average heatmap was used to extract one single bounding box or if one bounding box was extracted for each scale. The average heatmap was calculated by taking the average probabilities over the heatmaps created for each scale. When extracting multiple bounding boxes the following merging algorithm was used to merge these into one.

1. Assign C to be the located class.
2. Assign B_s the predicted bounding box for C for all scales in s (see Equation 3.1).
3. Assign $B \longleftarrow \cup B_s$
4. Repeat merging until the length of B is 1:
   - $B \longleftarrow B \backslash \{ b_1, b_2 \} \cup merge\_boxes(b_1, b_2)$

In the algorithm `merge_boxes` takes the average of the bounding box coordinates which is the same approach as used in *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. There is however more steps to their algorithms that were not used in this study, they use a `match_score` to determine if the merging should continue or not. In this work there were fewer bounding boxes initially since the bounding boxes were retrieved with a different approach. The merging process was not finalized until there was only one bounding box left.

Fixed Threshold vs. Adapting Threshold
The second parameter, the threshold used to extract bounding boxes, was also varied in two different ways. The first approach was to use a fixed threshold set to 0.5 (Equation 3.12). The second approach was to find the maximum probability, `max_prob`, in the heatmap (hm) and based on this set the threshold as seen in Equation 3.13.

$$T\_fixed = 0.5 \tag{3.12}$$

$$T\_adaptive = max\left(max\_prob(hm) - 0.1, 0.1\right) \tag{3.13}$$

All Scales vs. Selected Scales
The third parameter determines which of the two settings *All Scales* or *Selected Scales* that is used when creating the average bounding boxes. To create the bounding box for the average heatmap or the bounding boxes for each scale either all scales s (see Equation 3.1) or a subset s_{sub} of s can be considered. The set s_{sub} for each class (see Table 3.10) was chosen based on the size of the organ or body-part to locate in mm, as can be seen in Table 3.1. This was done since it is more likely that an organ or body-part can be located if it fits within a window.

Class	s_{sub}
head	100, 150, 200
neck	70, 100, 150
shoulders	150, 200, 250, 350
chest	200, 250, 350
right lung	70, 100, 150, 200
left lung	70, 100, 150, 200
abdomen	200, 250, 350
liver	70, 100, 150, 200
right kidney	50, 70, 100
left kidney	50, 70, 100
pelvis	200, 250, 350
femur	150, 200, 250, 350
knee	150, 200, 250
tibia	100, 150, 200, 250, 350
ankles/feet	150, 200, 250

Table 3.10: The sliding window sizes used for each class when selected scales are used to extract bounding boxes.

Localization covers the task where it is known which object is sought and that object is located. Combining the described approach with the exisiting classifier the problem can be formulated the following way. First the original classifier (trained for the classification part) is used to classify an image, then the sliding window approach is used to locate the identified organs and body-parts in the sample. Knowing what is sought allows to select the most likely area even if probabilities overall are low, i.e using the *adapting threshold*.

The localization task ran on the selected data for each combination of the three settings resulting in 8 different combinations of settings.

The extracted subset of the data set contained 150 samples, 10 samples were extracted per class. Multiple classes can appear in the images making some of the classes appear in more than 10 samples as shown in Table 3.11. The table also shows the number of times the original classifier managed to detect the class, making it subject for localization.

Class	# of Appearences	# Times Detected
head	10	10
neck	10	9
shoulders	11	7
chest	28	23
right lung	31	29
left lung	33	29
abdomen	41	31
liver	28	16
right kidney	19	10
left kidney	20	12
pelvis	11	10
femur	10	10
knee	10	9
tibia	10	9
ankles/feet	11	10

Table 3.11: The number of times a certain class appears in the test set and the number of times it is detected by the classifier.

Evaluation

As can be seen in Table 3.11 the original classifier does not detect all samples for each class, there are false negatives. Similarily there are also false positives. The performance of the original classifier has however already been evaluated thus all false negatives and false positives are neglected and only the true positives are located.

The localization performance is evaluated using the intersect over union measurement to compare the predicted bounding box with the ground truth bounding box. The following is calculated for each setting:

- The average and median IoU for each class and the total average IoU
- The percentage of hits ($IoU \geqslant 0.5$ as defined in Section 2.7)
- The maximum and minimum IoU value for each class

Examples of good and bad predicted bounding boxes together with their respective heatmap representation are also presented as part of the evaluation of the technique.

4 Results

This Chapter presents the results used to answer how well a CNN can perform when classifying medical CT images. It also gives results that are the base for answering if a *table of content* for a full stack can be created by using a CNN taking 2D CT images as input. Lastly, results for CNNs trained on different sized datasets are presented to be able to determine if more training data would likely improve the classifiction performance of the CNN.

4.1 Tuning the CNN

This section addresses the first part of the study, the tuning of the hyperparameters *learning rate (η)* and *L2 regularization coefficient (λ)*. Even though the L2-regularization is zero for some models, regularization is still applied in form of early stopping and dropout. Table 4.1 shows the settings used and the validation accuracy and validation loss for each of the trained models obtained from the grid search. It also displays the number of epochs the model was trained before early stopping terminated the training process or 50 epochs were completed.

Model	η	λ	**#Epoch**	**Val acc**	**Val loss**
6	10^{-3}	10^{-1}	40	99.23	0.02
7	10^{-3}	10^{-2}	37	99.22	0.0205
11	10^{-4}	10^{-1}	42	99.11	0.0229
12	10^{-4}	10^{-2}	39	99.02	0.0261
2	10^{-2}	10^{-2}	37	98.87	0.0265
8	10^{-3}	10^{-3}	33	99.02	0.0302
14	10^{-4}	10^{-4}	32	99.03	0.0305
10	10^{-3}	0	38	98.86	0.0313
15	10^{-4}	0	32	98.96	0.0322
9	10^{-3}	10^{-4}	32	98.88	0.0322
3	10^{-2}	10^{-3}	37	99.21	0.0335
13	10^{-4}	10^{-3}	38	98.85	0.0368
19	10^{-5}	10^{-4}	50	98.64	0.0378
16	10^{-5}	10^{-1}	50	98.51	0.0388
20	10^{-5}	0	50	98.39	0.0443
18	10^{-5}	10^{-3}	50	98.36	0.047
17	10^{-5}	10^{-2}	50	98.43	0.0498
23	10^{-6}	10^{-3}	50	97.05	0.0862
25	10^{-6}	0	50	96.91	0.0862
22	10^{-6}	10^{-2}	50	96.91	0.0956
21	10^{-6}	10^{-1}	50	97.01	0.0991
24	10^{-6}	10^{-4}	50	96.37	0.1116
1	10^{-2}	10^{-1}	35	93.39	0.135
4	10^{-2}	10^{-4}	32	94.03	0.9616
5	10^{-2}	0	41	93.33	1.0756

Table 4.1: Results for all the models obtained from the grid search sorted on validation loss. Validation accuracy is rounded to two decimals and validation loss is rounded to four decimals. The model number is the same as in Table 3.6.

The best model is model number 6 as seen in Table 4.1. The validation loss is 0.02 and it is trained with $\eta = 0.001$ and $\lambda = 0.1$. Early stopping terminated the training after 40 epochs. Ten models (16, 17, 18, 19, 20, 21, 22, 23, 24, 25) did not early stop before reaching the 50th epoch and terminated due to reaching the limitation on number of epochs to run. These nine models are the models with the lowest value of the learning rates, $\eta = 10^{-5}$ and $\eta = 10^{-6}$.

The learning curves for the training accuracy and validation accuracy for the best model is displayed in Figure 4.1. The learning curve for the training loss and validation loss is displayed in Figure 4.2.

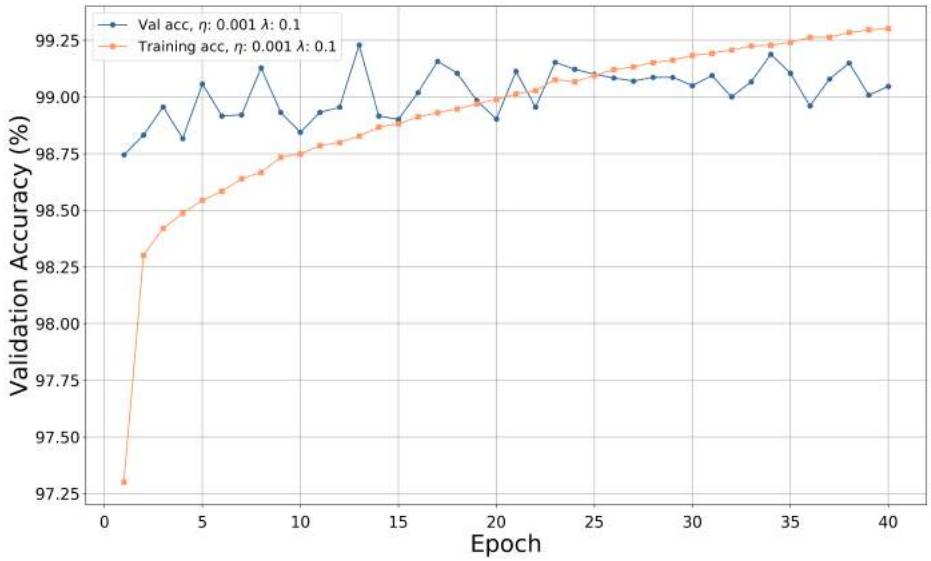


Figure 4.1: Learning curves for the training accuracy and validation accuracy for the best model, $\eta = 10^{-3}$ and $\lambda = 10^{-1}$.

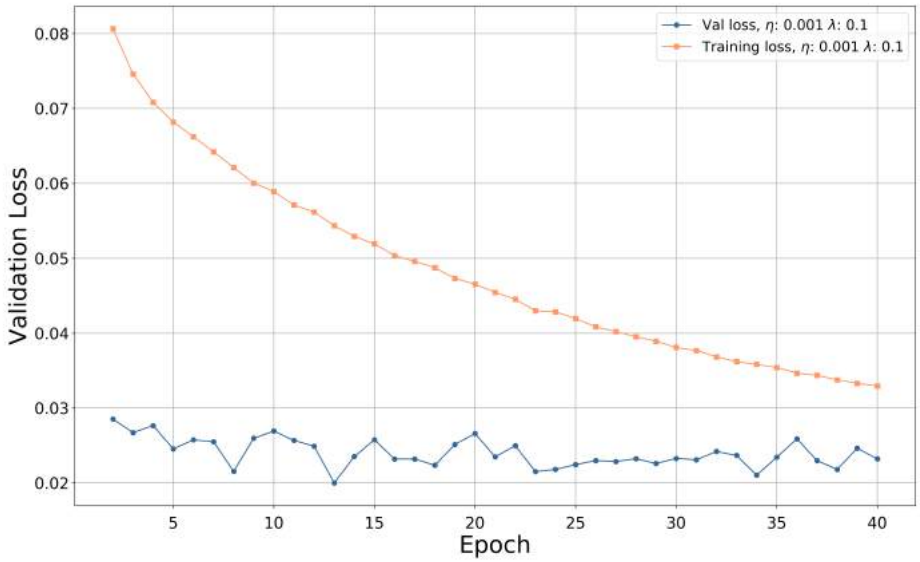


Figure 4.2: Learning curves for the training loss and validation loss for the best model, $\eta = 10^{-3}$ and $\lambda = 10^{-1}$.

As can be seen in Figure 4.2 the lowest validation loss is after epoch 12. The chosen best model is therefore the model after epoch 12 and not epoch 40.

4.2 Best Model Evaluation

The previous section only considers the validation dataset to find and select the best model among the models trained using a grid search algorithm. From this section and onwards **the model** refers to the best model that was selected. This section presents the results when the model was used to classify the data in the held out test set. Table 4.2 shows a summary of the average (both macro-average M and micro-average μ) values over all classes for the listed metrics.

Metric	Value (%)
Average accuracy	98.89
Average error rate	1.11
Precision_μ	95.37
Recall_μ	91.93
F-score_μ	0.936
Precision_M	89.41
Recall_M	86.4
F-score_M	0.879
Exact match ratio	88.24

Table 4.2: Metrics for the model, definition of metrics is found in Section 3.5.

The accuracy, error rate, recall (M and μ) and precision (M and μ) were also calculated individually for each class. These metrics are plotted in Figure 4.3, Figure 4.4 and Figure 4.5. Exact values for class specific metrics are found in Appendix A.1. For numbers of true positives, true negatives, false positives and false negatives per class see Appendix A.2.

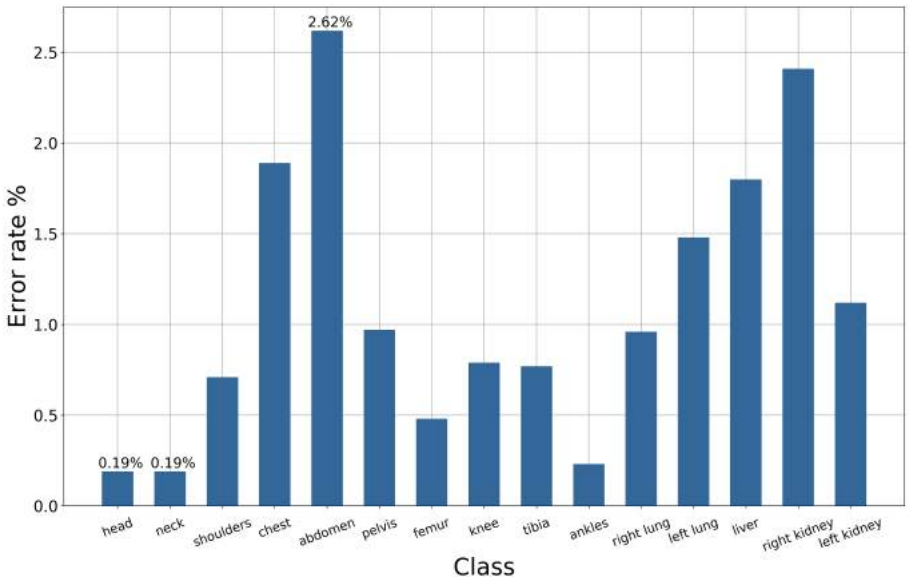


Figure 4.3: Per class error.

The class with the highest error rate is abdomen with error rate 2.62%. The classes with the lowest error rates are head and neck with an error rate 0.19%.

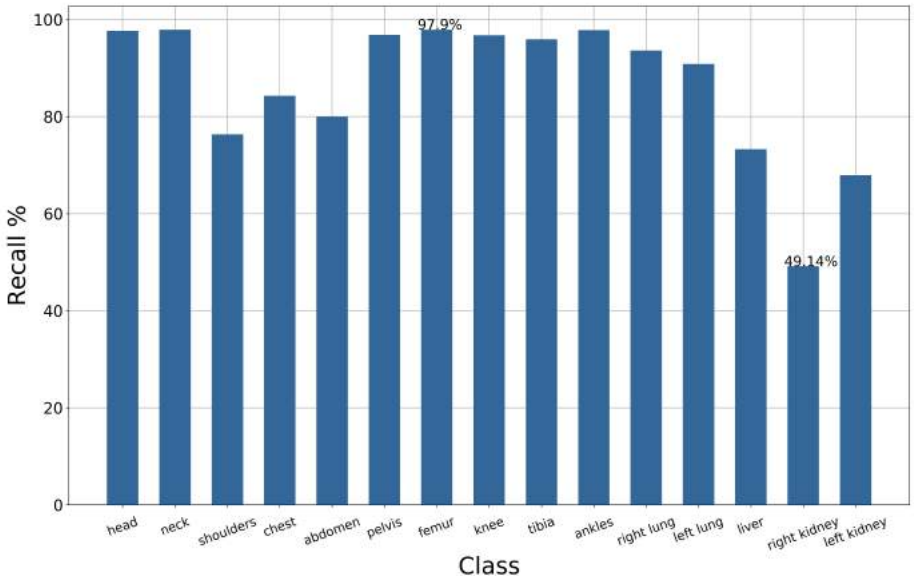


Figure 4.4: Per class recall.

The class where the largest fraction of the relevant cases are selected is femur which has a recall value of 97.9%. Neck and head follows closely behind with a recall value of 97.89% and 97.66%. The class where the least number of relevant cases are selected by the classifier is right kidney where the recall value is 49.14%, meaning that the right kidneys are not detected in more than half of all samples it is depicted in.

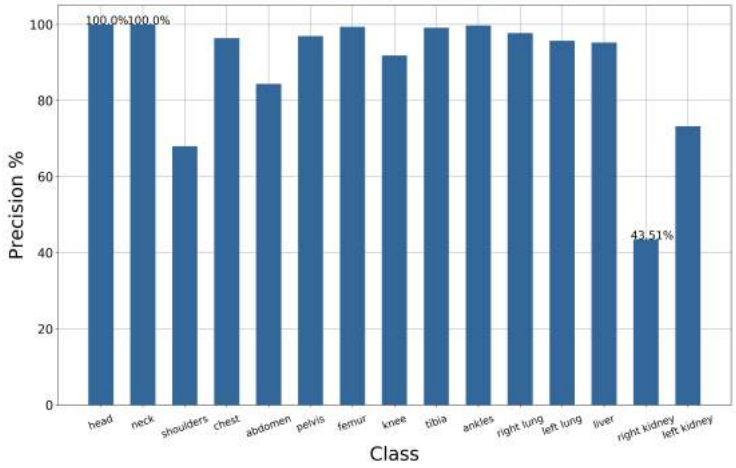


Figure 4.5: Per class precision.

The class where the largest fraction of the selected cases are relevant are head and neck for which the classifier score a precision value of 100.0%. For these two classes there are no false positives. The class where the least fraction of the selected are relevant cases is the right kidney with precision 43.51% meaning that in 56.49% of the cases where a right kidney is predicted it is a false positive.

4.3 Misclassifications

When a sample is classified incorrectly it is of interest to see what was predicted instead of the correct class. Figure 4.6 shows this information for each class. The blue bottom of the bar is the percentage of cases where nothing was predicted instead of the correct class. On top of that the orange bar represents the percentage of the cases where a *relevant misclassification* as defined in Section 3.6.1 was selected instead of the correct class.

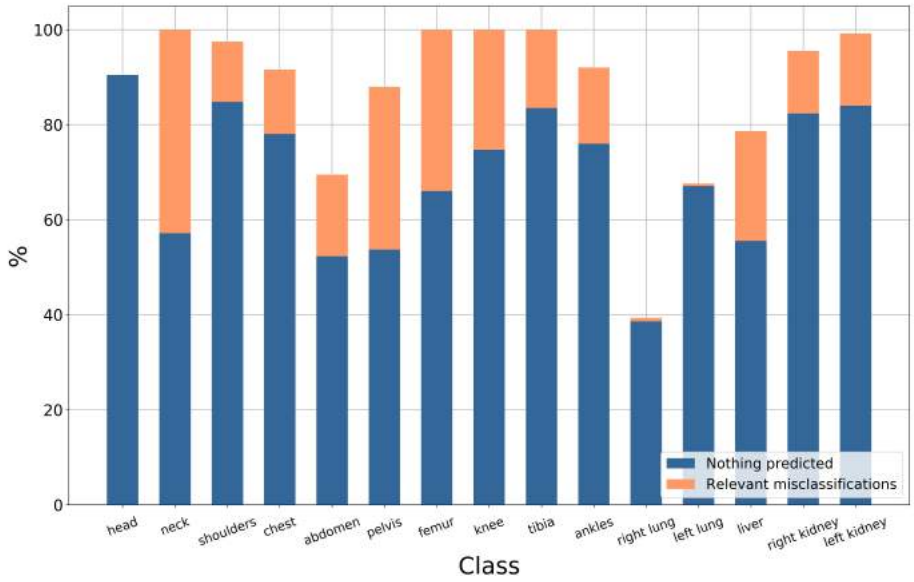


Figure 4.6: Percentage of misclassifications where the prediction is a relevant misclassification and the case where no class is predicted at all.

This is also calculated for the cases where the misclassification is a false negative in Figure 4.7 and where the misclassification is a false positive in Figure 4.8.

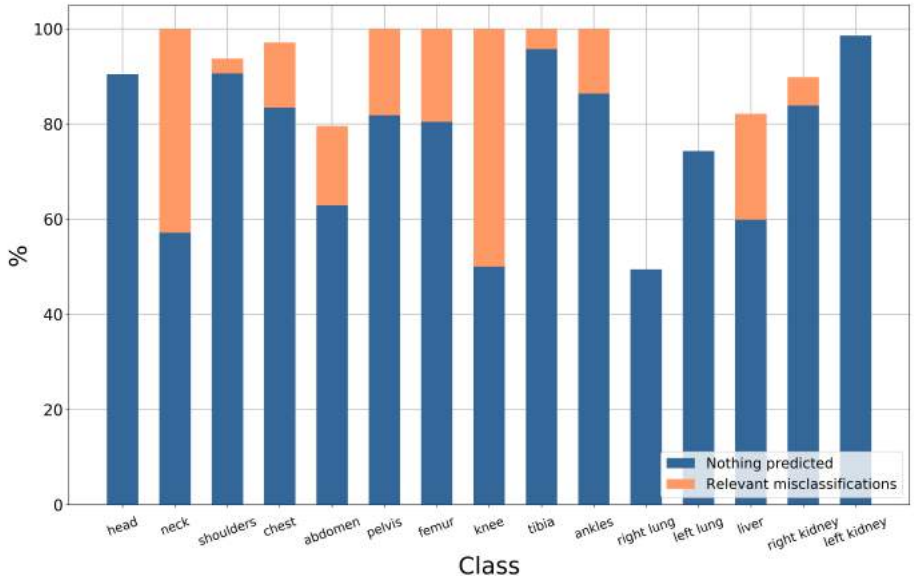


Figure 4.7: Percentage of misclassifications where the prediction is a relevant misclassification and the case where no class is predicted at all when there is a false negative for a class.

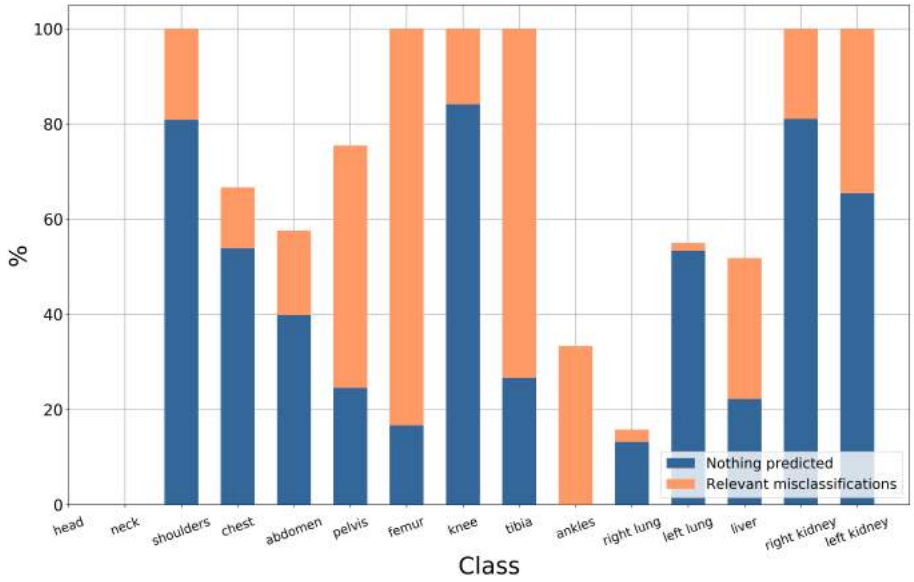


Figure 4.8: Percentage of misclassifications where the prediction is a relevant misclassification and the case where no class is predicted at all when there is a false positive for a class.

For the classes head and neck there are no false positives since they have 100% precision as shown in Figure 4.5.

Observing Figure 4.8 it is notable that when a misclassification is a false positive it is more common that it is relevant misclassification compared to misclassifications for false negatives shown in Figure 4.7. For misclassifictions that are false negatives no class predicted at all is more common than a relevant class predicted. Detailed numbers on what is predicted instead when there is a misclassification of a sample are found in A.3.

4.4 Predicting Full Stacks

This section presents the result of predicting entire stacks. Figure 4.9 shows the result for a full body stack that contains the entire body. The stacks was added to the dataset specifically for this experiment. In each graph the black line plotted at probability 0.5 represents the threshold that was used when converting a probability into a true or negative classification for a class.

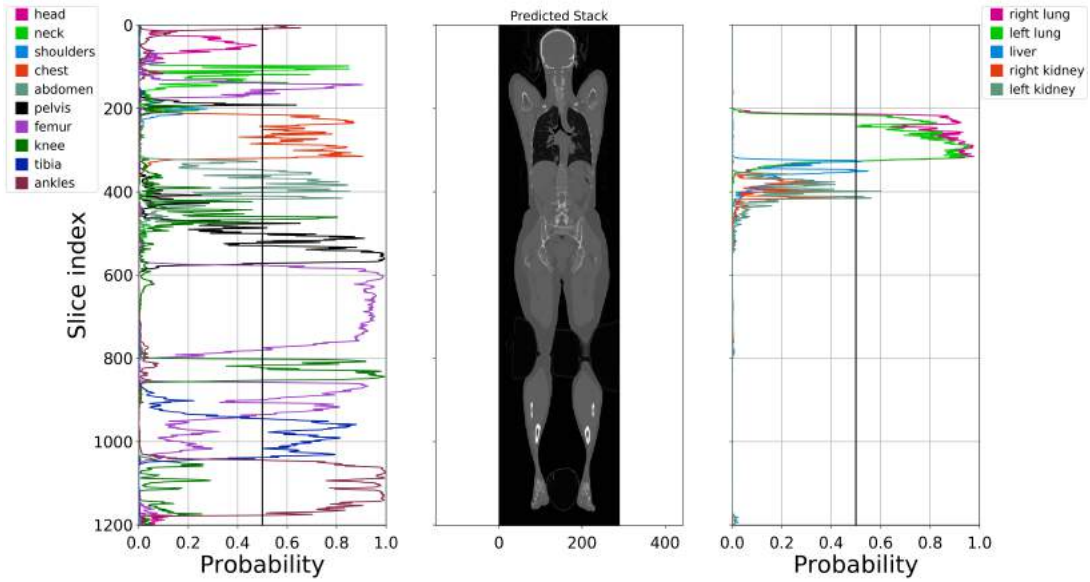


Figure 4.9: Full body stack, depicting all classes.

Figure 4.10 and Figure 4.11 shows two stacks depicting heads. Both these stacks were retrieved from the test set.

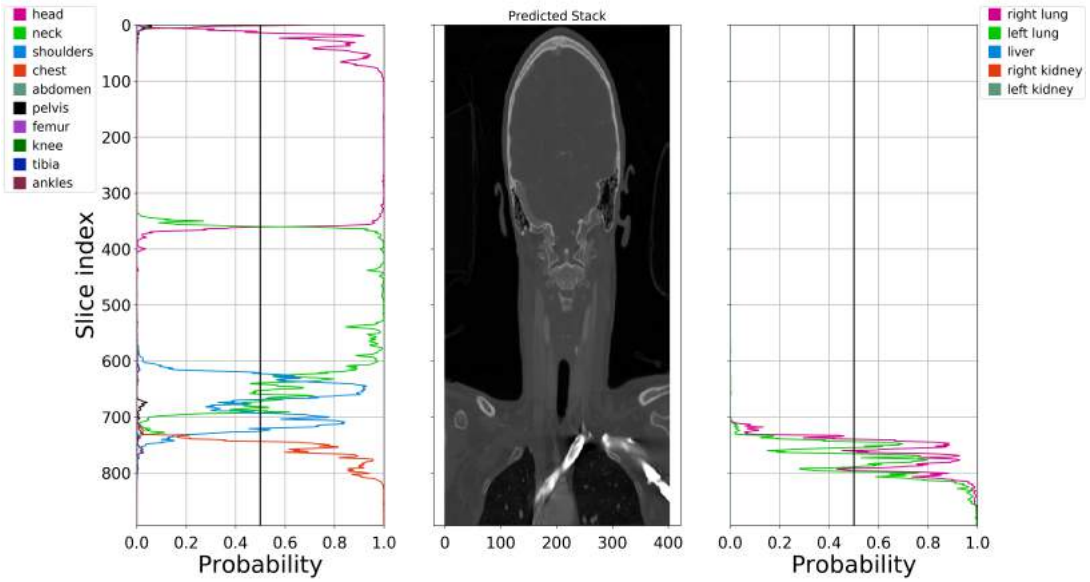


Figure 4.10: Stack depicting left and right lung, chest, shoulder, neck and head.

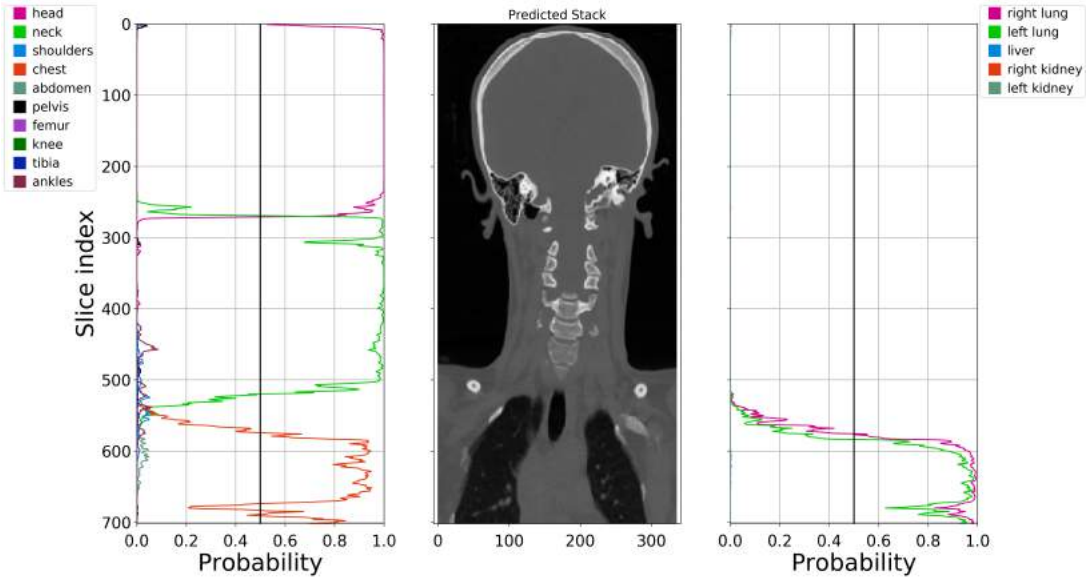


Figure 4.11: Stack depicting left and right lung, chest, shoulder, neck and head.

Figure 4.12 and Figure 4.13 depicts the result for two stacks depicting the upper body except the head. Both these stacks are new and added specifically for this experiment.

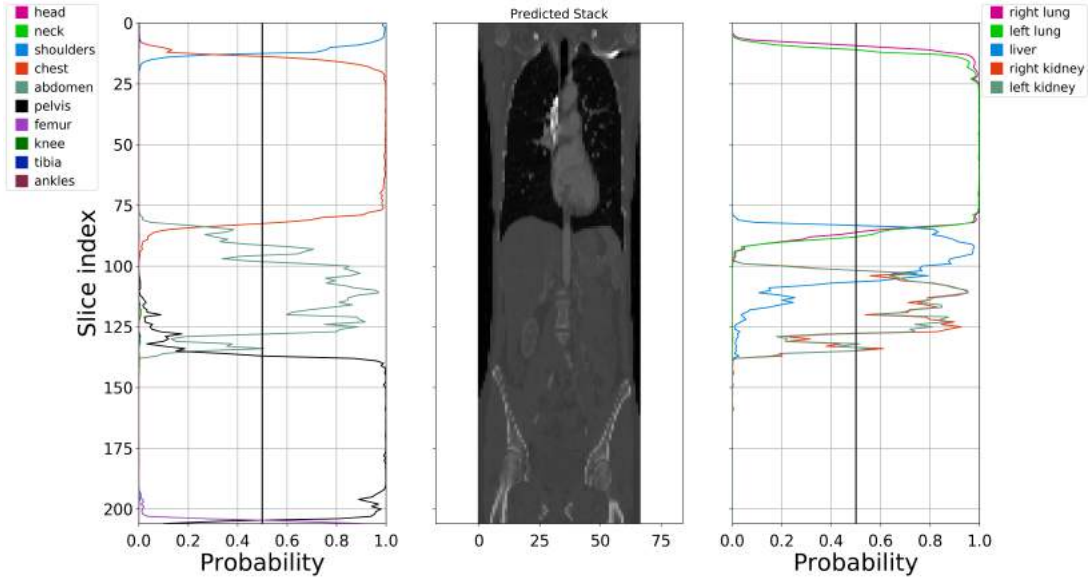


Figure 4.12: An upper body stack depicting pelvis, left and right kidney, liver, abdomen, right and left lung, chest and shoulders.

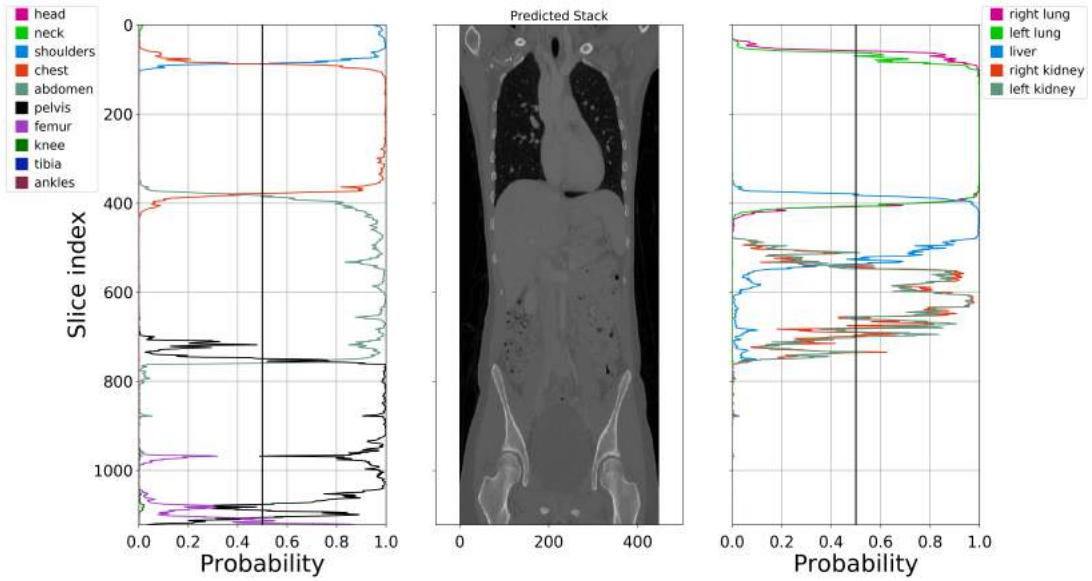


Figure 4.13: An upper body stack depicting pelvis, left and right kidney, liver, abdomen, right and left lung, chest and shoulders.

The result for the final two stacks is shown in Figure 4.14 and Figure 4.15. These two stacks were retrieved from the test set.

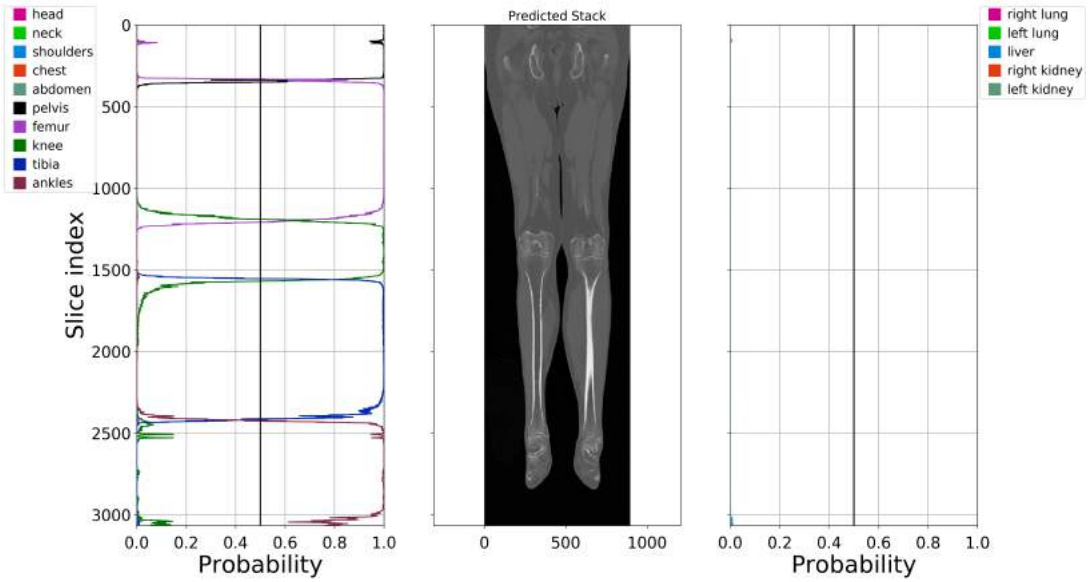


Figure 4.14: A lower body stack depicting ankles/feet, tibia, knee, femur and pelvis.

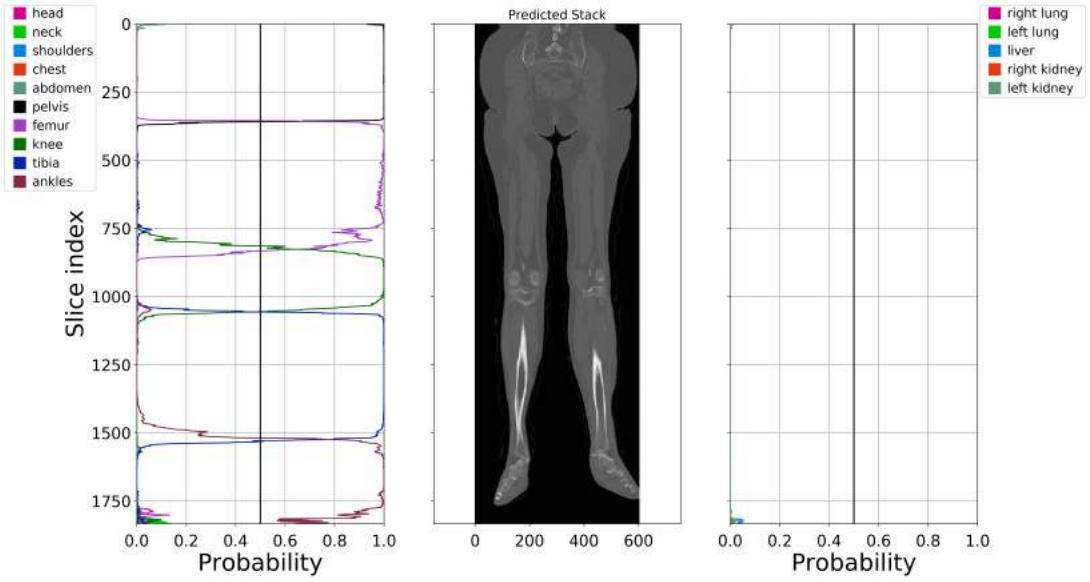


Figure 4.15: A lower body stack depicting ankles/feet, tibia, knee, femur and pelvis.

4.5 Reduced Training Set

The resulting values for different metrics when training the model with different sized datasets are shown in this section. Each graph shows the fraction of the dataset (% of samples) on the x-axis and the resulting metric value on the y-axis.

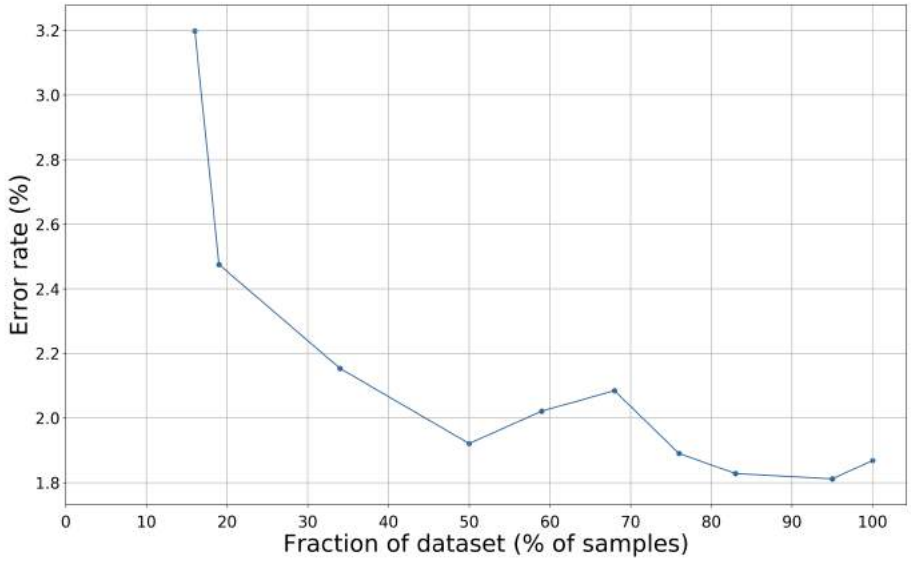


Figure 4.16: Reduced training sets error rate.

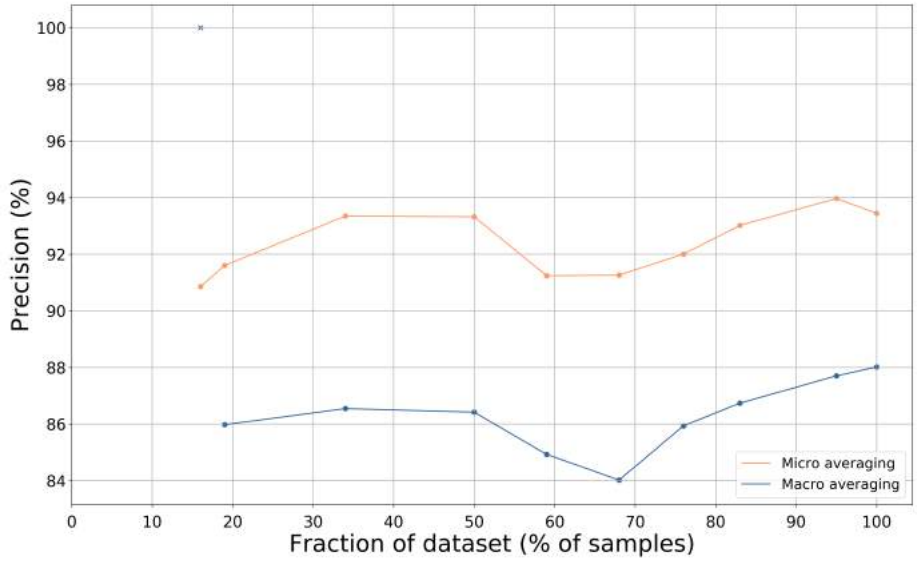


Figure 4.17: Reduced training sets precision. The cross shows that it was not possible to calculate precision for a fraction of the dataset.

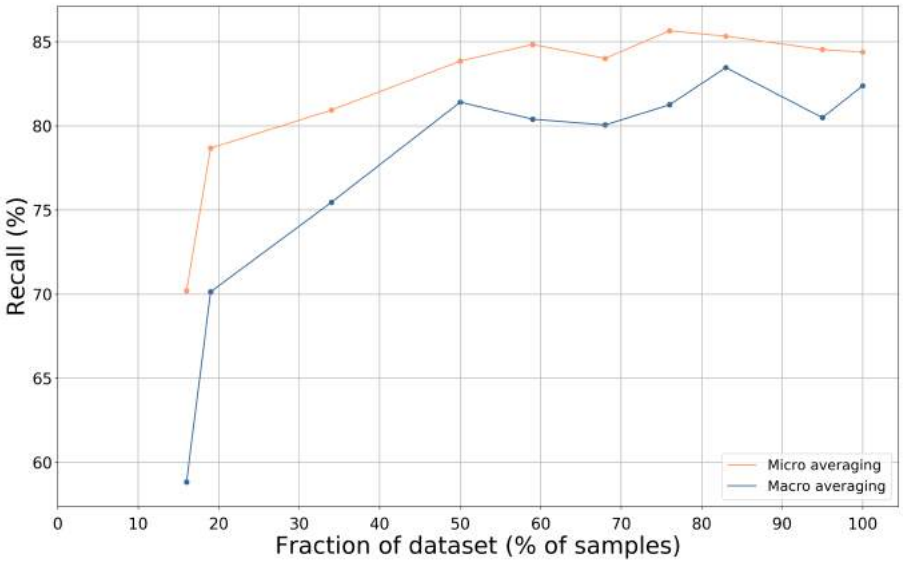


Figure 4.18: Reduced training sets recall.

4.6 Localization with Sliding Window Technique

The results of this experiment are presented for the combination of settings that gave the best percentage of hits which is using *Average Heatmap*, *All Scales* and *Adapting Treshold*. For these settings the largest image in the test set was covered by 3721 windows for the smallest size in s (Equation 3.1) and 529 for the largest size in s. The smallest image in the test set was covered by 9 windows for the smallest size in s and 1 window for the largest size in s. The following sections presents example of heatmaps and predicted bounding boxes for four selected classes and the resulting IoU scores.

4.6.1 Heatmaps

The example heatmaps chosen are for pelvis, right lung, liver and left kidney. First the best prediction for a class is presented followed by the worst prediction for the class. The best and worst prediction is determined by taking the prediction with the highest and lowest IoU value.

The first image shows the actual image with the ground truth bounding box (green) and the predicted bounding box (blue). To the right the average heatmap is displayed. The following smaller heatmaps show the predicted heatmap for each size in s. For some images the large sizes in s result in windows that is larger than the image, thus there is no heatmap for these sizes.

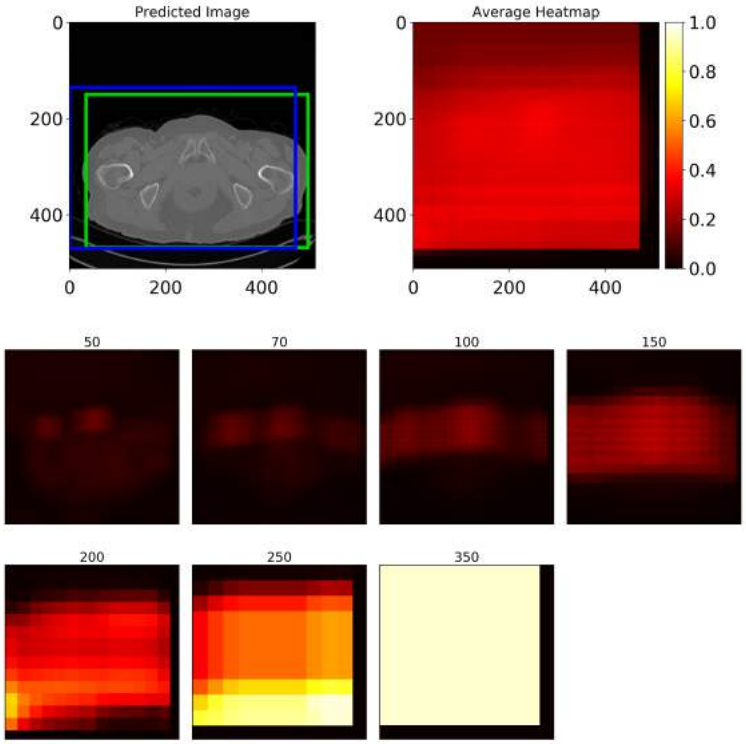


Figure 4.19: Heatmaps for the best prediction for pelvis

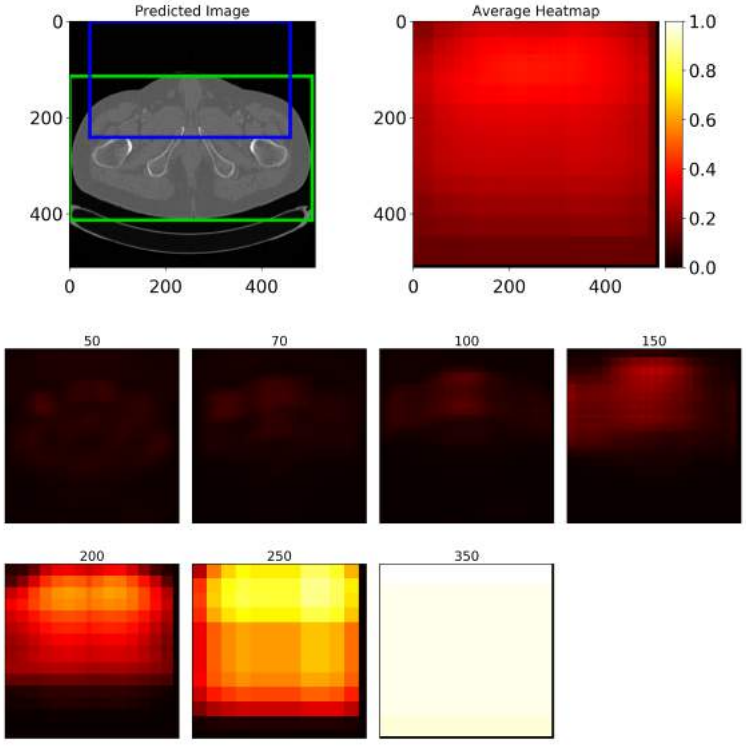


Figure 4.20: Heatmaps for the worst prediction for pelvis

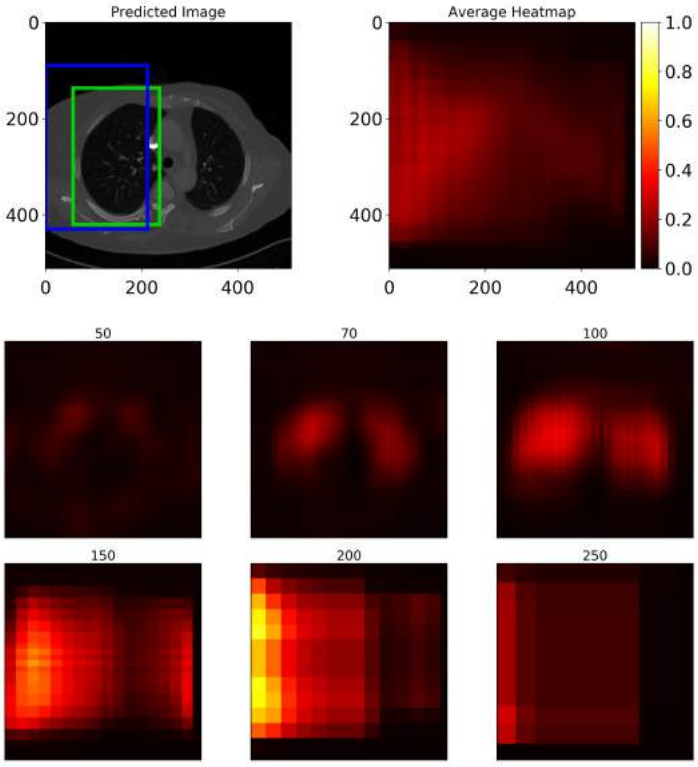


Figure 4.21: Heatmaps for the best prediction for right lung

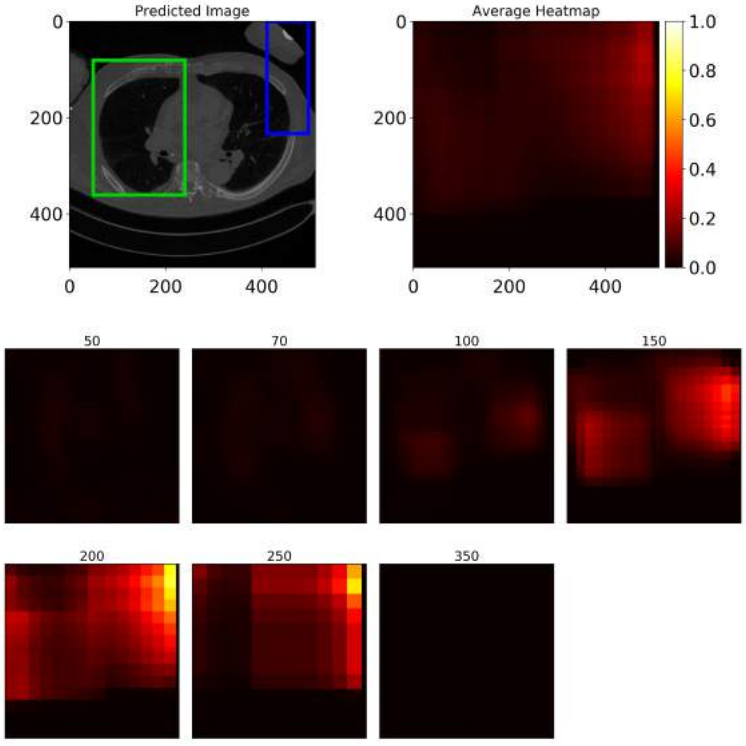


Figure 4.22: Heatmaps for the worst prediction for right lung

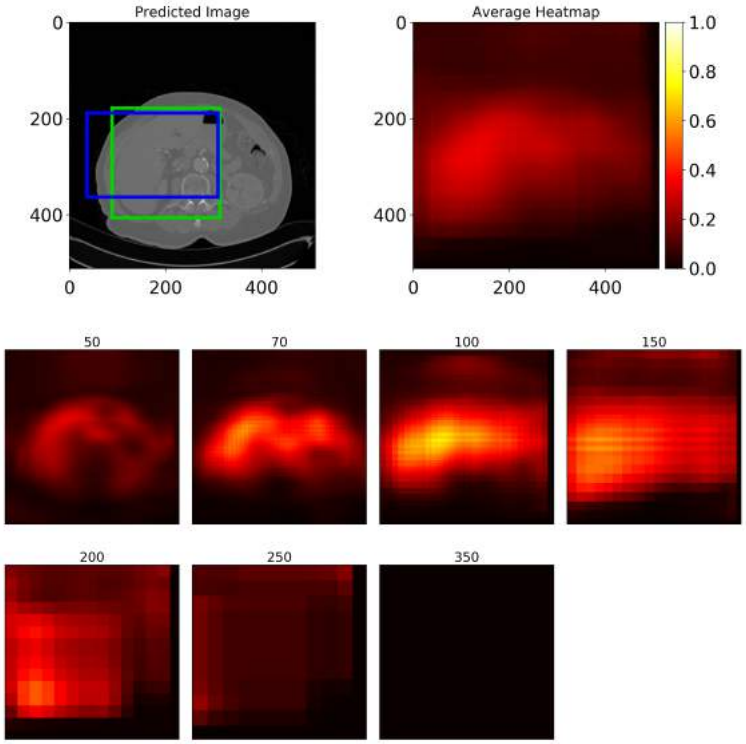


Figure 4.23: Heatmaps for the best prediction for liver

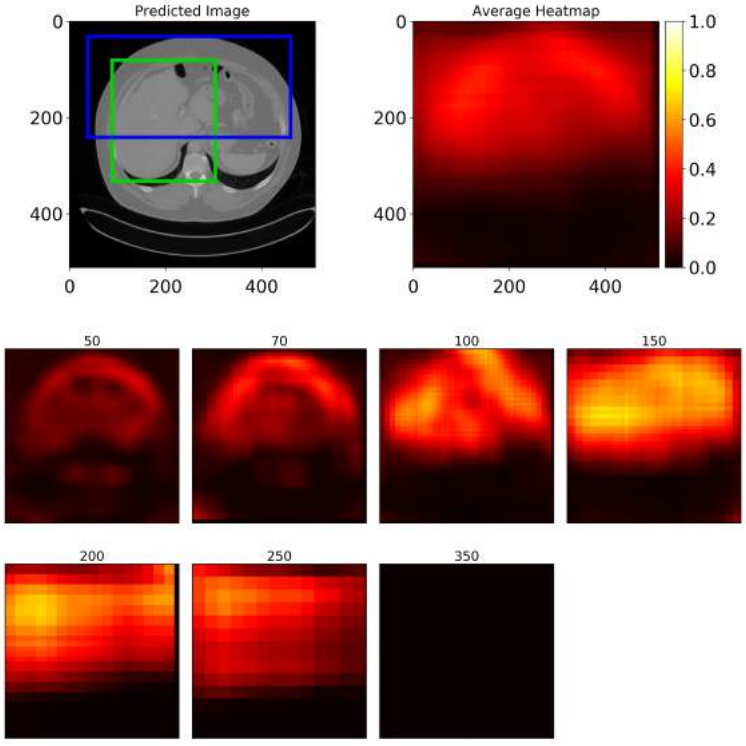


Figure 4.24: Heatmaps for the worst prediction for liver

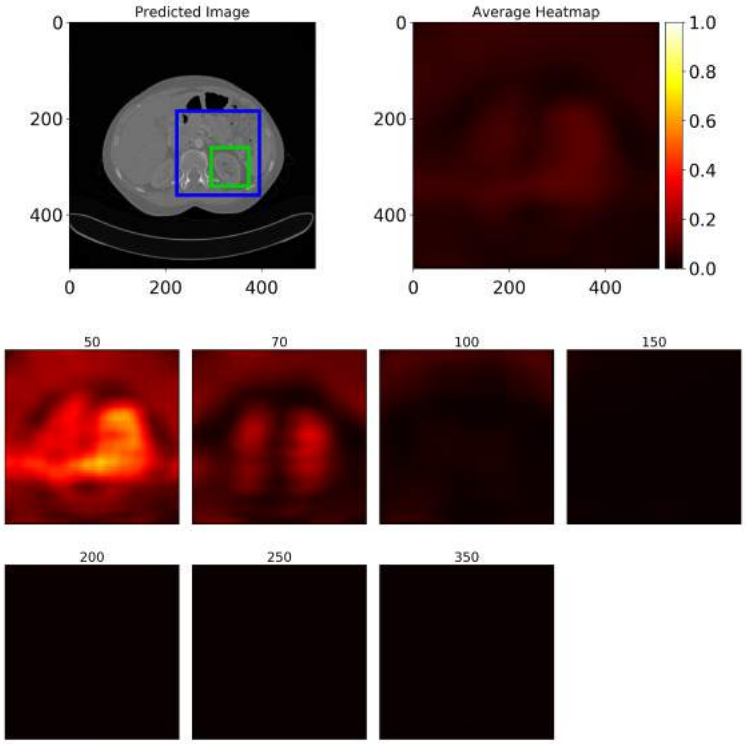


Figure 4.25: Heatmaps for the best prediction for left kidney

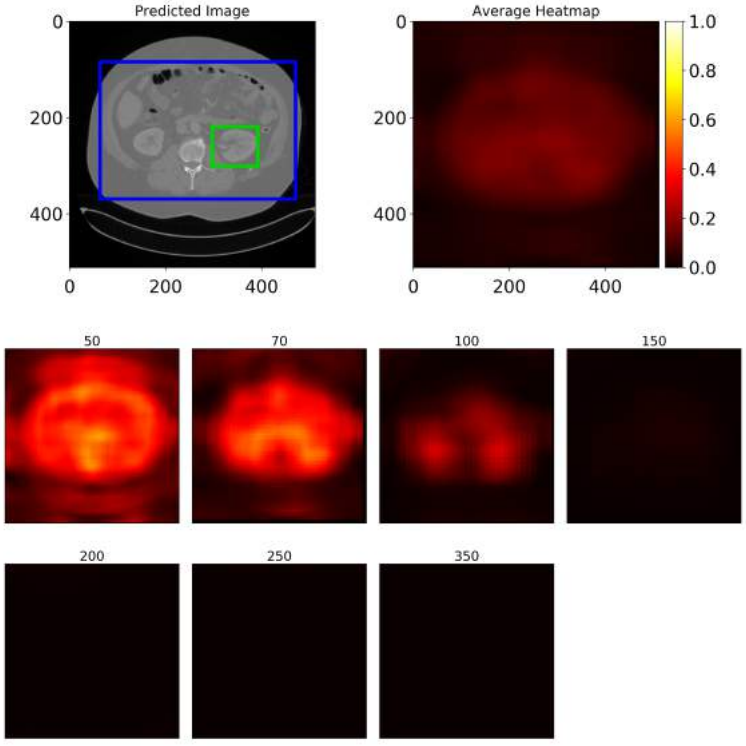


Figure 4.26: Heatmaps for the worst prediction for left kidney

4.6.2 Intersect over Union Scores

Table 4.3 shows the percentage of hits, average IoU, median IoU, maximum IoU and minimum IoU for each class and for all classes collectively. Results for all the different combinations of settings can be found in Appendix B.1.

Class	Hits (%)	Average IoU	Median IoU	Max IoU	Min IoU
head	90.0	0.66	0.69	0.81	0.49
neck	11.11	0.41	0.4	0.72	0.2
shoulders	100.0	0.61	0.59	0.75	0.5
chest	95.65	0.65	0.69	0.88	0.38
abdomen	68.75	0.55	0.56	0.92	0.0
pelvis	80.0	0.63	0.66	0.84	0.27
femur	70.0	0.56	0.59	0.66	0.47
knee	22.22	0.35	0.32	0.51	0.23
tibia	11.11	0.36	0.36	0.53	0.16
ankles	0.0	0.25	0.23	0.4	0.11
right lung	3.45	0.18	0.19	0.55	0.0
left lung	10.34	0.28	0.29	0.55	0.0
liver	56.25	0.51	0.52	0.64	0.32
right kidney	0.0	0.1	0.1	0.18	0.0
left kidney	0.0	0.12	0.13	0.21	0.0
all classes	40.35	0.42	0.4	0.92	0.0

Table 4.3: Intersect over Union scores for the settings scoring the best percentage of hits. The used settings were Average Heatmap, All Scales, Adapting Threshold

The percentage of hits is generally higher for the larger organs and body-parts than for the smaller. To understand the distribution of IoU scores for the located classes the CDF in Figure 4.27 can be consulted.

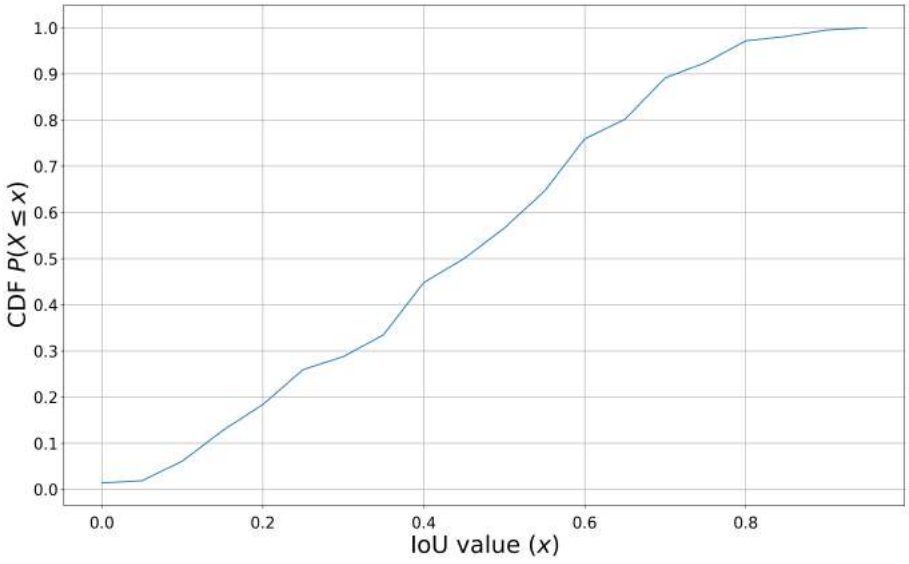


Figure 4.27: CDF for IoU score.

5 Discussion

This Chapter ties together the results and compares them with state-of-the-art within the image classification and image classification within the medical field. The practical use of the results are also discussed and possible problems that need to be considered if this type of solution is to be used in a real world setting. Additional techniques to improve the predictions are touched and the validity of the results is discussed both in terms of method choices and the dataset used.

5.1 Comparison with State-of-the-Art

The CNN classifier for medical CT images scored an average error rate of 1.11% as seen in Table 4.2. For class specific errors the class abdomen has the highest error rate 2.62% and the class head has the lowest error rate 0.19% as can be seen in Figure 4.3.

In ILSVRC-2016[1] the best image classification result had an error rate of 2.99%. That classifier was however trained on a set containing 1.2 million natural images. The classification was done over 1000 classes. Medical image classification with CNN has been done by Roth et al. [35] in 2015, their best result was an error rate of 5.9% when classifying the five classes neck, lungs, liver, pelvis and legs. These classes are a subset of the classes we have used. What differs their study from ours is that their classes can not appear in the same image and that they have used a much smaller test set. The smaller test set makes each misclassification have a larger impact on the error rate percentage. Our error rate is low compared to what Roth et al. achieved but this can be misleading since our distribution between positive and negative samples is not as even as theirs. The low error is partly a result of the fact that our classifier can determine when a sample **does not** belong to a class and that we have many negative samples for each class.

[1] Large Scale Visual Recognition Challenge 2016, image-net.org

Class	# positive samples	# negative samples
head	897	10166
neck	994	10069
shoulders	131	10932
chest	1104	9959
right lung	1229	9834
left lung	1230	9833
abdomen	830	10233
liver	655	10408
right kidney	232	10408
left kidney	218	10831
pelvis	1706	9357
femur	1948	9115
knee	730	10333
tibia	1714	9349
ankles/feet	1005	10058

Table 5.1: The distribution between positive and negative samples in the test set.

Table 5.1 shows the distribution between positive and negative samples in the test dataset for each class. It is clear that if the model can predict negative samples well the error rate will be low. It is a however a strength for the model to be able to find negative samples as well.

Another study that classifies medical CT images with CNN has been conducted by Cho et al. [7], they manage to achieve an error rate of 4.33% when testing their system on a test dataset containing 1000 images per class. The classes they used were brain, neck, shoulders, chest, abdomen and pelvis. These classes are also a subset of the classes we have used. Similar to Roth et al. the classes in their study were disjunct. Their difference between negative and positive samples in the test set is larger than in the study by Cho et al. but not as large as in our work.

Furthermore Roth et al. and Cho et al. considered different networks. Roth et al. opted to use a network consisting of 5 convolutional layers and 3 fully-connected layers while Cho et al. used an existing architecture called GoogLeNet. We chose to work with an existing network but in contrast to Roth et al. and Cho et al. we did not fully-train our network. The choice to fine-tune instead of fully-training possibly affects the results. The error rate obtained using our fine-tuning approach is comparable to the state-of-the-art results.

In *The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets* Saito et al. highlight that precision and recall reveals differences in performance that would be unnoticed if only accuracy is used when the the test set is imbalanced [38]. Our skewed distribution between negative and positive samples seen in Table 5.1 implicate that the error rate can be misleading and analyzing recall and precision metric values of the classifier gives a different perspective on performance. It is desirable to get both high recall and precision, meaning that all relevant samples are selected and all selected samples are relevant. In reality there is a trade-off between these two, lowering the threshold for considering the prediction a positive will increase the recall but might reduce precision.

Unfortunately Roth et al. and Cho et al. have not presented precision, recall or F-score results for their studies. As can be seen in Table 4.2 the metrics for macro-averaging(M) is lower compared to the corresponding micro-averaging(μ). Due to the uneven distribution of samples over the classes in the test set the macro-averaging metrics are of more interest. This way the bias caused by the larger classes is avoided.

The model scored a value of 95.37% for micro-averaging precision and 89.41% for macro-averaging precision. Recall have a lower value than precision for both macro-averaging and

micro-averaging. For micro-averaging the recall was 91.03% and for macro-averaging the recall was 86.4% as can be seen in Table 4.2. The recall is lower than precision for both metrics, indicating that the model is better at finding fewer false positives than false negatives. The F-score for micro averaging 0.936 was higher than for macro averaging 0.879.

In practice the individual classifications for each class are useful. However, to claim that multi-class multi-label classification has been done we must also highlight the result which is referred to as exact match ratio for which the model scored 88.24%, giving the much larger error rate 11.76%.

5.2 Organ Specific Discussion

The model experienced problems with certain classes. By observing Figure 4.4 and Figure 4.5 it is worth noting that the right kidney is a problematic class since it has the worst recall 49.14% and precision 43.51%. One possible reason is that the right kidney is located close to the liver. The left kidney also had one of the lowest scores in terms of precision and recall. An explanation to why the kidneys are the classes that score the worse is that they are the smallest organ in the dataset and can be represented with as little as $25px$ in the image (see Table 3.1). Another justification is the fact that the kidneys are located in the most *crowded* area of the body, where there are many other organs surrounding them. This is shown in the recall for the crowded areas, abdomen and chest. The recall for theses two body-parts is lower than other regions that are not crowded, e.g. head and lower body. The precision for abdomen is also lower than other body-parts which are not crowded which supports this theory.

There are occasions where only one kidney is visible in a sample, which could complicate the predictions. The model might have learned a pattern for the kidneys - if one exist both should, or none. This did not occur for right lung and left lung since they had a more equal number of samples in the dataset. Another class the model found troublesome in terms of precision was shoulders, with a value of 68.03%. As can be seen in Table 3.2 shoulders is the class with the least samples represented in the training and the test set which could be the reason.

In the case of a misclassification it is interesting to know what was predicted instead of the correct class. In Section 3.6.1 a term *relevant misclassification* was defined. In Figure 4.6 it is notable that if a misclassification occurred, for all classes but right lung it was either a relevant misclassification or nothing that was predicted in more than 70% of the misclassification cases. A possible reason that the misclassifications for lungs are non-relevant in a larger fraction of the cases is the fact that they are located relatively close to the liver and the abdomen, but none of these are relevant classes for lungs. All missclassifications for lungs belong to one of these classes as can be seen in Appendix A. Head is an interesting class since it does not have any relevant misclassification but a high value of nothing predicted. An explanation to this is that the head have a unique anatomical structure and thus it is less chance that it is mistaken for another body-parts.

As can be seen in Figure 4.10 between slices $600 - 700$ both neck and shoulders are predicted for the same slices. This is in a transition region between neck and shoulders. That a misclassification occur in a transistion between two classes is a possible origin for many of the the misclassifications.

5.3 Use in Practice

In a real world setting there are certain aspects that have to be considered. One have to identify if the goal of the target application is to find all positive samples thus favoring recall or if the goal is to be correct about the selected samples, thus favoring precision. This issue is called the precision-recall trade-off and is visualized in Figure 5.1.

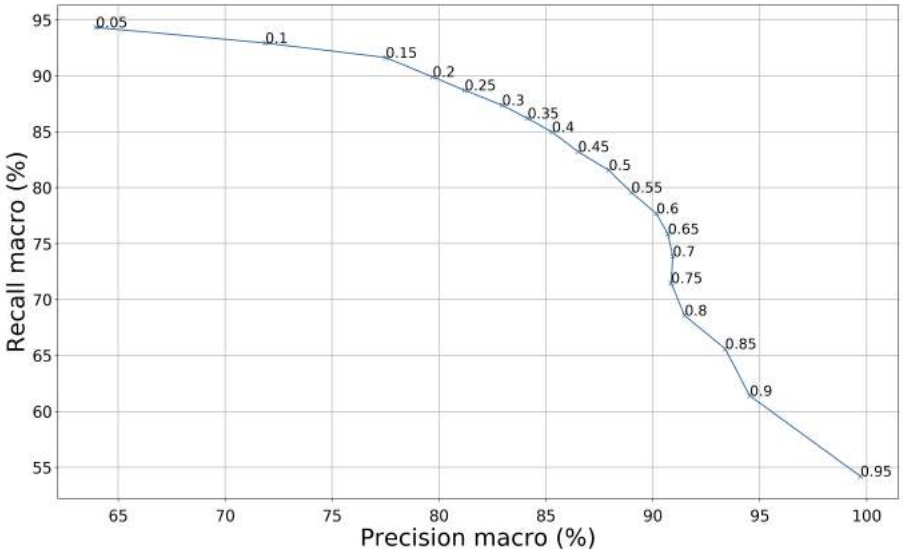


Figure 5.1: The precision-recall trade-off depending on the threshold used. The annotations in the graph show the treshold value.

Favoring either recall or precision is clearly a problem specific decision. If we reconsider the case of retrieving priors as described in the background section in Chapter 1 and imagine a situation where a clinician just received a referral about a patient and it states that the liver is to be considered. Suggesting examinations to the clinician is a problem where recall should be prioritized, to the extent that the number of false positives doesn't make the clinician's job troublesome.

Another real world setting is to have an adaptable GUI where the tools available are depending on what organ or body-part that is examined at the moment. This means that for example a shortcut pane would be populated with the tools that the clinician is most likely to use. Tools should only be added if there is a high certainty that the clinician will make use of them, enhancing the purpose of a shortcut pane. In case of false negatives the tools will still be available in their original menus for the clinician to find. Thus, precision should be the focus for this problem.

The F_β-score can aid in the selection of an appropriate threshold. β describes how many times more importance should be put on recall than precision. By plotting the threshold and F_β-score one can find an appropriate threshold for the real world application depending on what should be favored.

The misclassifications is another aspect to consider in a real world context, what is preferable when a sample is misclassified? There is no single answer to this. If the aim is to get a high precision it is preferable to get nothing predicted at all when there is a false negative. It can be observed in Figure 4.8 and 4.7 that in a larger fraction of the cases nothing is predicted instead when there is a false negative than where there is a false positive. One could claim that the classifier is almost right when classifying some class as a spatially close class instead. However it might be easier to correct mistakes in a post-processing step if the prediction is completely wrong.

5.4 Creating Tables of Content

To be able to discuss if a 2D classifier can suffice to create a table of content for a full stack the results in Section 4.4 are considered. It can be seen that some stacks (Figure 4.10, Figure 4.11,

Figure 4.14 and 4.15) are almost perfectly predicted by the classifier. One explanation to the good results on these stacks is the fact that the patient the stack is depicting also exists in the training set, although the specific stack does not.

These predictions can easily be converted into a *table of content* for the stack that can store the organs and body-parts depicted and the z-coordinates in the stack where it is located. Solutions to take the localization of the organ or body-part a step further and locate it in each 2D slice of the stack, giving a full 3D bounding box, are discussed in Section 5.7 and Section 6.1.

Some stack predictions are not as good, for example the full body stack in Figure 4.9. The full body stack is a completely new stack where the patient has not been previously seen in the training data. Although Figure 4.12 and Figure 4.13 are also depicting previously unseen patients the predictions are much more accurate making it likely that the impact of the patient being unseen is not the main reason and not as large as initially suspected.

One possible explanation to these varying results is the varying image scales in mm/px as explained in Section 3.2.2 in the method chapter. The training data has different scales and if a scale has been seen many times during training it seems that the classifier can more easily classify new images with the same scale. Figure 5.2 shows the distribution of samples for different scalings in the training set and the test set. The crosses indicate to which scaling interval the test stacks used for the stack experiment belong. It is clear that the full body stack belongs to an interval ($> 2.2mm/px$, it has scaling $2.23mm/px$) where we do not have a lot of training data while the other two unseen stacks are located in intervals where there exist a lot of training data.

This can also be noted for the two head stacks (4.10 - H1, Figure 4.11 - H1). Even though both stacks origin from examinations of patients who are present in the training set the classifier fails to predict the shoulders in stack H2 but succeeds in H1. Inspecting Figure 5.2 it is clear that H2 belongs to a scaling interval that differs more from most of the data than H1 does. This supports that scaling is of greater importance than the fact that the patient has appeared in the training set and suggests that the features are not scale invariant.

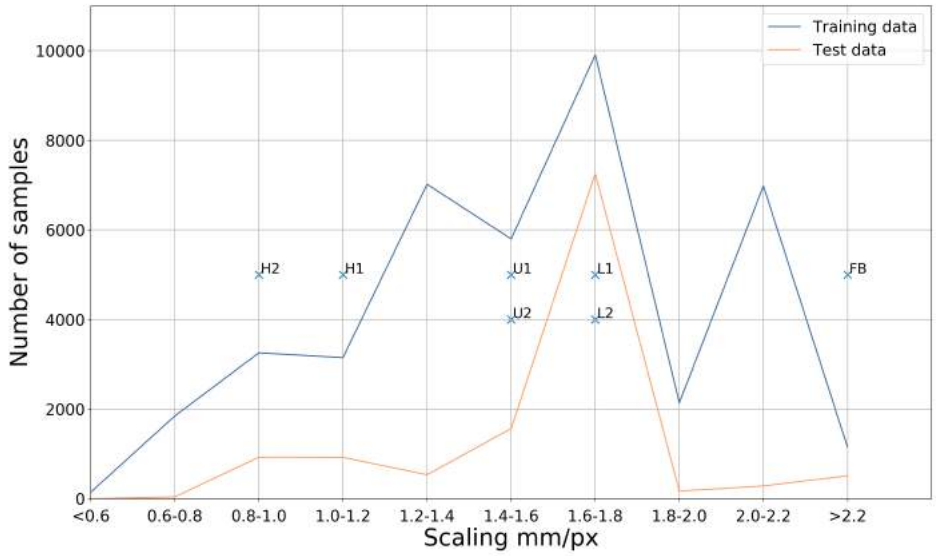


Figure 5.2: The amount of samples per scaling interval in the training dataset and the test dataset. Crosses mark the scaling of the test stacks used for the stack experiment. FB = full body stack 4.9, H1 - head stack 1 4.10, H2 - head stack 2 4.11, U1 = upper body stack 1 4.12, U2 = upper body stack 2 4.13, L1 - lower body stack 1 4.14, L2 - lower body stack 2 4.15.

It can also be observed that the majority of our test set belongs to scaling intervals that are well represented in the training set. And we have peeks of data with scaling $1.2 - 1.4$, $1.6 - 1.8$ and $2.0 - 2.2$. One could suspect that the zoom level used where these peeks are located is also the most common zoom levels to use when capturing a CT scan since our data is randomly selected from real patient data.

To verify the hypothesis that the predicted features are not scale invariant the full body stack in Figure 5.3 was rescaled and cropped to match a scale that is more frequently represented in the training data. The chosen scale was 1.6 and the resulting predictions are shown in Figure 5.4. It is clear that the predictions generally (for most classes) are more consequent and accurate when using the resampled stack. The predictions for knee and tibia are exceptions to this, as the predictions get worse as the stack is rescaled. This further highlights the importance of the scaling as the new scaling is less frequently represented than the original scaling for these specific classes. For knee there are approximately 500 training samples that have scaling around the value 2.2, but there are only around 100 samples that have scaling around 1.6. For tibia there are around 1200 samples with scaling value 2.2 and there are only 200 samples around scaling value 1.6.

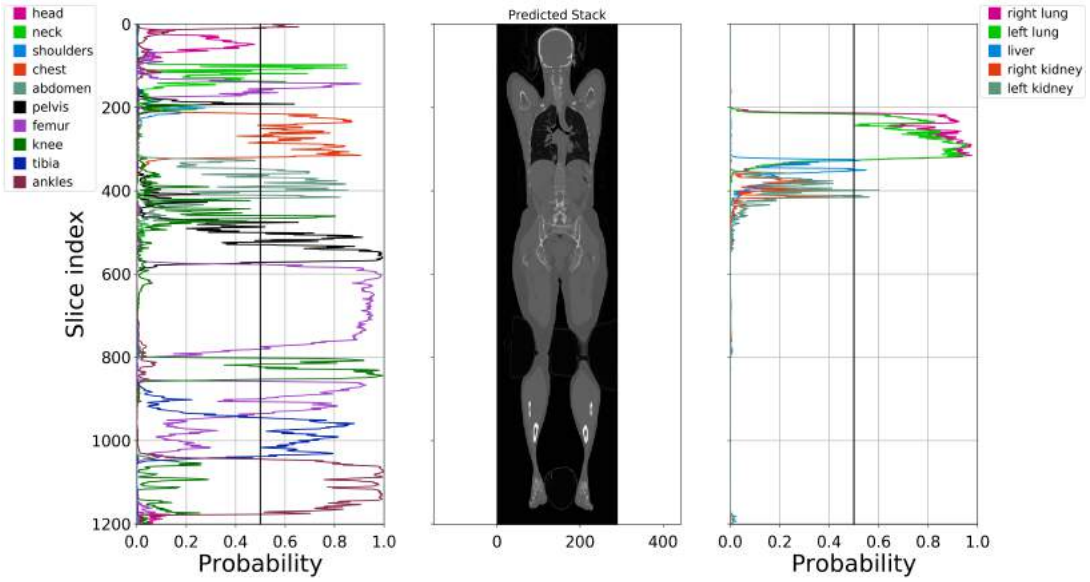


Figure 5.3: The full body stack as presented in the result section.

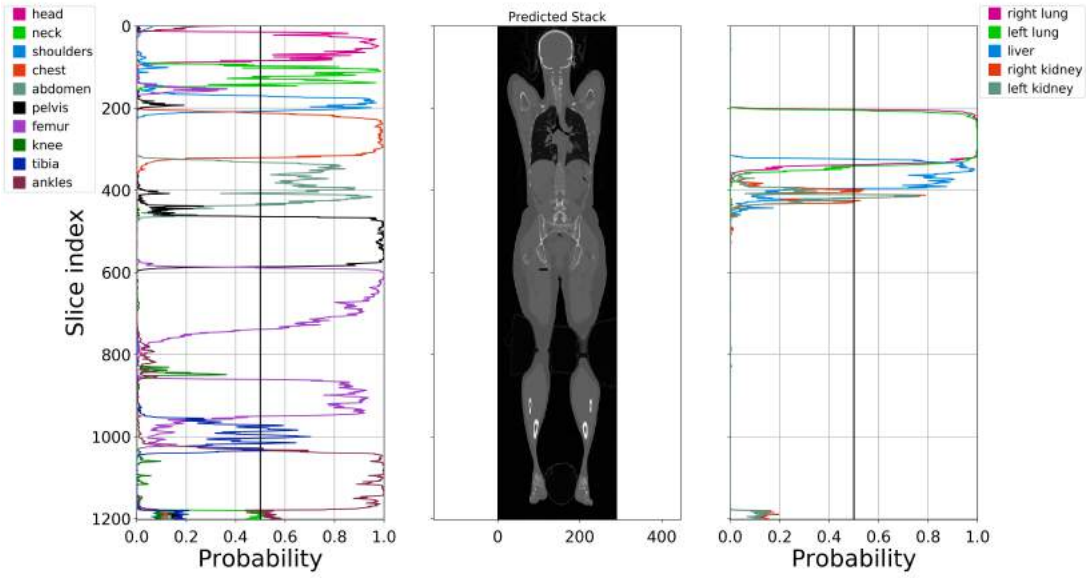


Figure 5.4: The full body stack used in the stack experiment but resampled to have scale 1.6.

5.5 Size of Training Dataset

The graphs in Section 4.5 indicate that using up until 50% of the training samples, for training, a significant improvement of the error rate can be observed. Training with dataset beyond that size does not give as much improvement.

Zooming in on the 16% training set (a), the 50% training set (b) and the 100% training set (c) we see that the distribution of samples is the following:

Training dataset	(a)	(b)	(c)
head	322	1064	2717
neck	578	830	3030
shoulders	179	559	1325
chest	1094	1801	4780
abdomen	1030	2155	5124
pelvis	859	2276	4801
femur	1027	3298	5056
knee	724	1620	2481
tibia	1030	3165	4287
ankles	562	1911	2725
right lung	1204	2315	5947
left lung	1207	2223	5936
liver	510	1199	3306
right kidney	604	942	2253
left kidney	559	952	2717

Table 5.2: The distribution of samples in (a) 16% of original data, (b) 50% of original data and (c) 100% of orignial data.

It was found by Cho et al. in *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?* [7] that around 1000 samples per class should give an accuracy of 97.25%. Analyzing dataset (a) in Table 5.2 some classes have approximately 1000 samples and some that have less data. With this training set the classifier scores 97.37% which is a small improvement compared to what was found by Cho et al. for 1000 samples.

In dataset (b) the number of samples for most classes are in the range 1000-3000 samples and the accuracy is 98.66%. According to what was found by Cho et al. the accuracy for this number of samples should be in the interval $97.25 - 99.5\%$ which maps well with the accuracy scored.

The final dataset (c) contain 4000 or more samples for around half of the classes, but some classes have significantly fewer samples. Cho et al. found that 4092 samples per class should give an accuracy of 99.5%, with the final dataset we score 98.87%. We have more than twice as many classes and we do not have 4000 samples for each class, which can be reasons why 99.5% accuracy is not reached.

As previously noted the error rate does not improve much with the datasets larger than 50% (dataset (c) in Table 5.2) of the original dataset. This resembles the results by Cho et al. where they have predicted the accuracy based on the size of the training data used to train the CNN. Above 1000 samples the accuracy is not increasing much compared to below 1000 samples.

These points together with the results presented about the precision and recall in Section 4.5 indicate that increasing the size of the full training dataset is not likely to improve accuracy, precision and recall. To affect the result one might consider what type of data (spread over many patients) is used and possible pre-processing steps that can improve the data.

5.6 Improvement of the Predictions

In this section we present a few techniques that can be used to improve the predictions without replacing the actual model.

5.6.1 Pre-Processing

It was previously discussed that the scaling of the stack possibly affects the results. This issue can be circumvented with two different approaches. Either by training the model with additional data to have an even distribution over all scales. Possible sources for additional data are:

- Collecting additional stacks and ensure that all scalings are represented equally.
- Use additional data augmentation to zoom in on and zoom out on/crop the images to produce samples of varying scales to ensure that all classes are represented by all used scales.

Since the first approach would require the model to *know* many scalings it is probably harder to train than a second proposed approach. The second approach is to rescale all images in the training dataset to have the same scaling. When using the trained classifier it would always be required to rescale the images before predicting, but the classifier would only have to be trained for one scale.

5.6.2 Weighted Training Samples

Observing figure 4.4 the kidneys are the classes with the lowest recall. Not being able to detect a class can be an issues when the number of positive samples is low compared to the number of negative samples. One approach to improve the recall is to weight samples. Weighted samples gives a higher loss during training for some samples (for example all samples that belong to a class with few positive samples) compared to others even if the actual error is equally large. Weighted samples can aid in reducing the impact of an imbalanced dataset.

5.6.3 Post-Processing

One possible heuristic to use as a post-processing step is to use the knowledge of the slice order in a stack. The *slice index* itself does not provide much information since a stack can be captured of any part of the body and slice 0 will be the first slice in any stack. However, since there is a context of a full stack, relative position of a slide can be used to correct mistakes. For example if slice 0-100 are predicted to depict head and slice 110-200 are predicted to depict neck slice 101-110 are predicted to depict knee we can be pretty certain that the slices 101-110 are misclassified. These *single misclassifications* where there is just one or a few slices misclassified in the context of many correctly classified can be corrected. The approach is also applicable when we see the case as in the stack presented in Figure 4.13, the slices just below slice 1000. The prediction is *right* but the probability goes down at one point and then comes back up. Such a false negative can probably be recovered with the same technique.

Another option is to lower the threshold for classes where there are many false negatives. Usually lowering a threshold gives better recall and worse precision. By analyzing class specific recall-precision trade-off plots we find that the class right kidney would get both better recall and precision by lowering the threshold to 0.4 instead of 0.5, see Figure 5.5. This is probably a result of that the probability of right kidney rarely reaches above 0.5 and as the recall is increased by selecting more samples the model manages to select few false positives among these samples.

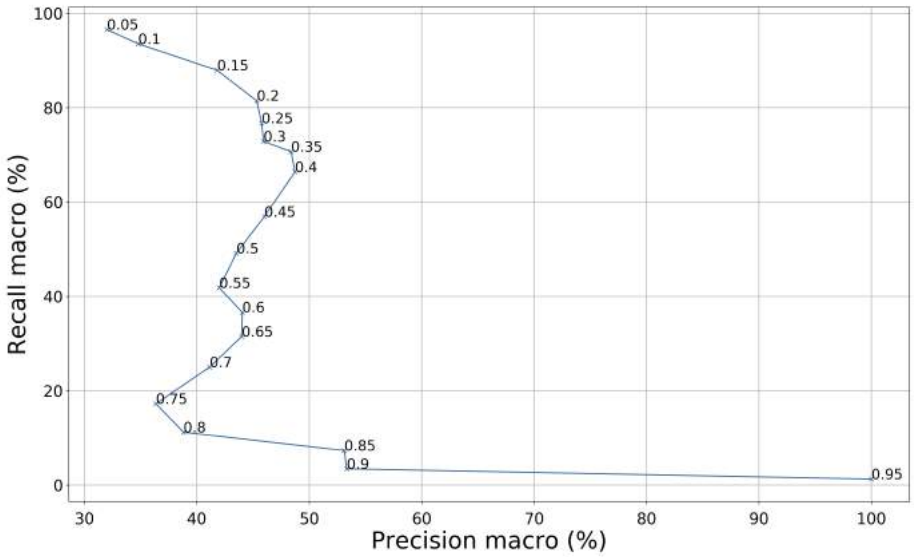


Figure 5.5: Class-specific precision-recall trade-off for right kidney.

5.7 Towards Localization and Detection

The work on localization and detection was just an initial investigation and there are many possible improvements that could be done. As only the true positives were considered to be subject to the sliding window algorithm the problem solved is of type localization. No false negatives or false positives are penalized. However, the pipeline that has been used can easily be extended to solve the detection task as well, by introducing punishment for false positives and false negatives.

5.7.1 Result for Bounding Box Extraction

The predicted bounding boxes were in the experiment extracted with a very naive approach. There are possibly much more sophisticated ways of extracting bounding boxes that can give better results. In *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks* [39] a regression network is used to output coordinates representing the location of an object in each window. Using a regression network is an approach that has not been used at all in this work.

Overall the results are bad, with hit rates that are 0% for multiple of the classes as can be seen in Table 4.3. Compared to Sermanet et al. [39] that reach an error rate of 29.9% for localization in natural images some of our classes are comparable but mainly because of the fact that the labels of these body-parts cover almost the entire image. The liver class scores 56% hits even though it is not one of the large body-parts that cover the entire image, a result that shows some promise for using this approach to locate organs in medical images. State-of-the-art in object detection scored by Ren et al. in *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [33] is around 70% detection accuracy, where the algorithm is also penalized for false positives and false negatives. A huge difference between their work and our work is the fact that they can process 300 frames per second (fps) where we at the moment complete the prediction for one frame in around 1.5 minutes.

The fact that larger organs and body-parts generally have higher hit percentage can be explained by their size. Many of these labels include almost the whole image since the labelling was done with containing labels and not contained labels. The largest windows get

very high probabilities (for example size 350 in Figure 4.19) thus a predicted bounding box covering almost the whole image is extracted. The smaller organs and body-parts generally have lower probabilities in the heatmaps indicating that the classifier trained on patches is not as certain when it detects these organs. This can be seen for the left kidney in Figure 4.25 and Figure 4.26.

The containing labels also make the widest part of an organ in the 3D visualization become the width of the label. Therefore there are examples where the labels are not representative for the organ they are supposed to highlight. An example of this can be seen in Figure 5.6 where the image depicts the top of the skull but the label has been based on the widest part of the head.

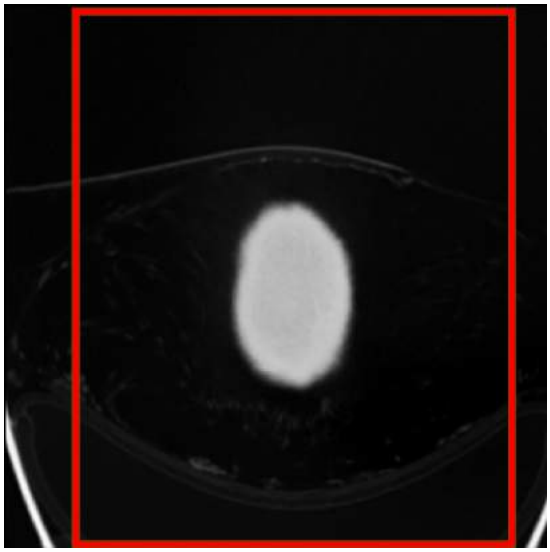


Figure 5.6: A labelled image depicting the top of the skull.

The generation of the patches that the CNN is trained with is one possible explanation to the above described issue. The patches are based on a label but if the label is rectangular its smaller dimension is expanded to match the larger dimension meaning that background or other classes are included in the patch. To get higher probabilities when running the sliding window algorithm it is suggested to have labels that more tightly surrounds the organ or body-part in the image, preferrably even use segmentation. Another potential improvement is to extract multiple patches for the organs that are rectangular, to produce multiple quadratic training images which depict different parts of the same organ. The CNN requires quadratic images in the training data.

5.7.2 Validity

This experiment is presented as an initial pre-study to analyze the feasibility to use a CNN and a sliding window technique to locate given organs and body-parts in medical CT images. However the test set is small, only containing 150 images, thus inferring reliability concerns. The small test set was used due to the long computation time needed to get predictions for all windows (improvements that make the process much faster are explained in Section 5.7.3). The results should not be interpreted as the answer to the localization task, rather they show that there is potential to develop this technique and make it more accurate. The heatmaps and results presented show indications on important factors that must be considered when further developing this technique.

Another consideration is the fact that a fixed threshold is used for some of the settings. When all probabilities are below the fixed threshold no bounding boxes are generated. This is also the reason behind the identical results for some of the settings (see Appendix B.1):

- *Merged Boxes, All Scales, Fixed Threshold* and *Merged Boxes, Selected Scales, Fixed Threshold*: The result of the fixed threshold is that only the most suitable scales (the ones used as selected scales) are the ones with probabilities above the threshold. Thus the two settings result in the same algorithm since no bounding boxes are generated for the set $\{ AllScales \backslash SelectedScales \}$.
- *Merged Boxes, All Scales, Adapting Threshold* and *Merged Boxes, Selected Scales, Fixed Threshold*: Even with one setting with adapting threshold the results are identical. The reason for this is that all heatmaps in the set $\{ AllScales \backslash SelectedScales \}$ have a maximum probability lower than the minimum threshold (0.1), thus no bounding boxes are generated for these.

Finally there are cases where not all scales results in a predicted heatmap since the image size might be smaller than the window size.

5.7.3 Efficiency

As already mentioned the time spent to predict the bounding boxes for the classes in one single image is around 1.5 minute on a NVIDIA GeForce GTX 1080 graphics card. The root cause for this is that each window is forward propagated through the entire network. However, if the algorithm is to be used in practice the network can be reconfigured to greatly improve efficiency. As described in Section 2.5.2 the fully connected layers can be converted into convolutional layers. The features can then be calculated for the entire image once and the result for each spatial region can be extracted from the output volume, meaning that the computation wasted on overlapping region is eliminated. The maximum number of windows that were forward propagated through the entire network for one image in the used test set is 3721, this number would be reduced to 1 if the CNN was rewritten.

5.8 Validity of Study

5.8.1 Data set

There are several concerns related to the dataset used in this study. One validity concern of the results of the study is the fact that the dataset was labelled by the authors and not by medical experts within radiology. The authors did however use organ maps and more experienced representatives at Sectra to verify the labels.

In the introduction a study by Gueld et al. [18] was presented which proved the incorrectness of manually added meta-data in DICOM data. The actual input to the CNN is not manually added data from the DICOM files since only the pixel data has been used. During pre-processing we did use manually added information to sort out some data which can possibly be a reliability issue. It should not be an issue that significantly affects the results.

A clear limitation to the work is the number of different patients and stacks that were used to create the dataset. The number is limited but this limitation had to be done in order to finish labelling within the time frame of the project. The fact that the test set and training set were split on stacks can also infer validity concerns since a stack depicting the same patient can exist both in the training dataset and the test dataset. We have seen examples where we get much better results if the patient has been previously seen even though the specific stack has not been previously seen 5.3. It is therefore recommended to split the dataset on patients, an adjustment that would require a much more extensive data collection and labelling phase.

The images in the dataset are real patient data meaning that the reason for the examination is some kind of disease state. Since there are diseases that can cause displacement or distortion of the organs and body-parts that could effect the results. The cases where a state like this was visible to us, for example an amputated leg, were removed from the dataset. However, there might be many conditions that we do not recognize which means that such samples might have ended up in the dataset.

The dataset is not publicly available but the fact that all data used conforms to the DICOM standard is useful for replicability purposes, CT images from any CT scanner have the same characteristics [4]. In the report the number of samples and stacks for all different classes and the labelling procedure are clearly presented in the method section. The fact that CT stacks are standardized according to the DICOM standard makes it likely that a different dataset based on the same type of data would perform similar to our dataset.

5.8.2 Method

When optimizing the model for this work some limitations have been done. The major limitation is that only two parameters were considered for optimization. The validity of values for other parameters is motivated by literature in the theory found in Chapter 2 and the method choices found in Chapter 3. The parameters chosen for optimization have also been motivated to be reasonable choices in these chapters. However supporting the choices isn't enough to claim that the optimal model has been found which means there might exist better training configurations that would achieve better results. Secondly the training phase is limited to a maximum of 50 epochs, for low learning rates the training does not terminate before this epoch as can be seen in 4.1. There might exist a configuration with a low learning rate that eventually would result in a better model if the number of epochs was increased.

By analyzing Table 4.1 it can be seen that many of the settings perform very well and that the top-5 validation losses do not differ much. Comparing model number 2 and 5 they score a significant different validation loss, 0.0265 and 1.0756 respectively, with the same learning rate but using different regularization. Regularization has significant impact on the generalization error. It is also interesting that the models with $\eta = 10^{-2}$ are spread out in place 5, 11 and the last three in place 15, 23 and 24. The last models have regularization $0, 10^{-1}$ and 10^{-4} which includes both the highest and lowest regularization used in the experiment, suggesting that the interval for the regularization coefficients was appropriate.

From the learning curves in Figure 4.1 and Figure 4.2 it is observed that the validation accuracy and validation loss are pretty close to their final value already after the first epoch, this could be a result of using transfer learning. Yosinski et al. [48] showed that transferring weights from a different task is better than random initialization of the network. They also showed that transferability of weights decreases as the task gets more distant but it has been shown by Chen et al. [6] that medical images do share many low level features with natural images which can be an explanation to why the validation error is low already after the first epoch.

5.8.3 Source criticism

The literature have been collected from different scientific databases, mostly from Google Scholar, IEEE Xplore:digital library and ACM digital library. The theory about CNN is mainly based on three sources, a book *Deep learning* by Courville et al. [17], a paper *Practical Recommendeation for Gradient-Based Training of Deep Architectures* by Bengio [1] and a paper *Deep Learning* by Le Cun et al. [25]. All these sources are well cited (300+ times). The authors are also well established within the area of deep learning, hence these sources are reliable. The authors of the book Deep Learning are Ian Goodfellow - a research assistant at Google Brain, Aaron Courville - Assistant Professor of Computer Science at the Université de Montréal and Yoshua Bengio - Université de Montréal employee and project co-director and senior fel-

low of the Learning in Machines & Brains at the Canadian Institute for Advanced Research[2]. Yann Le Cun is the director of the AI Research Facebook community and founding director of the NYU Center for Data Science. He has also been elected to the National Academy of Engineering for his advances in AI [3].

The approach used for localizing objects in the images is mainly based on the work in *Overfeat: Integrated recognition, localization and detection using convolutional networks*. The paper is cited 1387 times and among the authors we find Yann Le Cun.

When analyzing the work in wider context in Section 5.9 the thoughts are mainly based on online magazine articles that reflect opinions. These are not as reliable as the rest of the sources that have been used but the main purpose of including these is to raise the thoughts that are currently circulating in the community about the future of deep learning in the medical domain.

5.9 The Work in a Wider Context

With the development of technology and computerization that has taken part over the last decades jobs are changing. Tasks that previously depended on a human to be completed can today be automated. With the introduction of algorithms based on big data more complex no-routine will also change. Frey et al. [13] have estimated the probabilities of many different jobs to be automated. They found that the tasks that are least susceptible to computerization are the tasks that creativity and social skills.

5.9.1 Will doctors be replaced by algorithms?

A common thought when applying deep learning to the medical domain is "will algorithms replace the doctors?". Many claim that they will not, however the tasks of a doctor will radically change.

The assumption that computers do not have judgment and lack the ability of being creative or empathic and that these assumptions are necessary to perform a professional service are according to R. Susskind and D. Susskind [43] the base when stating that professions can't be replaced by computers. However, they find in their research that if the profession is broken down into many parts, a great part of the tasks do not require judgment, creativity or empathy. Naturally, as a doctor has human interaction with a patient, there are still some tasks in the profession that require these skills. In the article *Can An Algorithm Diagnose Better Than a Doctor?* in The Medical Futurist [28] it is stated that we are not at the point where an algorithm can diagnose more accurately than a doctor, yet, but we will get there. They highlight the fact that even in that scenario algorithms would not replace the doctors, but emphasize that the role of the doctor will radically change. This change has also been embraced as motivation for this study, if the tasks can be automated the doctors can spend additional time with the patient, the task that actually requires empathy and social skills.

Dr. Eliot Siegel from the University of Maryland says in the article *Part 2: How will AI affect radiology?* published on AuntMinnie.com [10] that radiologists will expect far more from their image display to in the future in terms of automation and intelligent systems. He compares the future AI systems in radiology systems to spelling and grammar checkers. "Instead of making the diagnosis, identifying all of the findings, or making recommendation, AI algorithms will make the interpretation process more efficient and less stressful for the radiologist."

[2]https://www.cifar.ca/profiles/yoshua-bengio/
[3]https://www.nae.edu/MembersSection/MemberDirectory/165608.aspx

5.9.2 Can we trust the artificially intelligent systems?

A concern when applying intelligent systems is requirements to be able to trust the decisions made by the system. The deep learning system is a black box and there is no way to look inside and determine how it works. The neurons and layer interconnection embeds the behaviour of the network. Researchers are currently working on creating systems that would explain its reasoning and making it more understandable and trustworthy to humans. There is a Defence Advanced Research Projects Agency (DARPA) program called *Explainable Artificial Intelligens (XAI)*[4] where the goals are to create more explainable models while maintaining the state-of-the-art accuracy and to enable humans to understand and trust the artificially intelligent systems. In an article called *Deep Learning Is a Black Box, but Health Care Won't Mind* [3] Monique Brouillette has interviewed Nicholson Price who is a legal scholar from the University of Michigan with health law as main subject. Price believes that the black box nature of deep learning system does not necessarily impose a threat to its application in medical applications and further compares it to drugs that are effective in treating disease by unknown means that have been introduced in the past. The same article states that the Food and Drug Administration (FDA) in the US are exposed to more applications that are driven by deep learning and have noted that companies creating these algorithms are allowed to keep the details confidential. One big challenge is clearly to justify that the algorithms work as intended and contribute to better health care.

5.9.3 Who controls the data?

Another important consideration is that fact that all the data used to train the algorithms originate from real examinations of real patients. Companies and research institutes will need to acquire data to make progress in research and applications of intelligent systems within health-care. The patients and privacy advocates will have to make sure it is not done without the correct consent. One must also discuss the objective of creating intelligent health care solutions. Will it actually make the treatment better for each individual or is it rather a societal gain reducing the overall cost of medical care that can be achieved. Clearly the interest will differ depending on the stakeholder.

5.9.4 Considerations in this work

Working with medical data the patient integrity is clearly an important consideration. To ensure that no sensitive data leaked during this project the CT stacks in the dataset were anonymized. It was also ensured that all tests ran on hardware within Sectra's facilities. The rights to publish the medical data published in the report has been approved by Sectra.

[4] http://www.darpa.mil/program/explainable-artificial-intelligence

6 Conclusion

This study evaluates how well a CNN can be used to do automatic organ- and body-part-specific anatomical classification of 2D CT images. Automating such tasks, that have previously depended on manual user input, can aid in many ways to reduce the workload of clinicians. Allowing them to spend more time with patients.

To solve the task the fully connected layers of a pre-trained VGG16 network were adjusted to fit the task, and the entire network was fine-tuned using CT data provided by Sectra. Several experiments were conducted to investigate the performance of the implemented solution. The resulting error rates are comparable to state-of-the-art using CNN to classify medical images when classifying individual organs or body parts. The error rate to get an exact match when considering it as a multi-class multi-label problem is significantly lower but no previous work to compare it to was found.

Using this approach to create a *table of content* of the depicted organs and body-parts in an entire stack shows promise. It was found that the scaling of the stack is of importance when training the CNN but if addressing this issue it is feasible to use the CNN to create *tables of content* for stacks.

It was investigated whether a larger dataset is likely to give better results and the finding is that a larger dataset does not necessarily mean better results. Analyzing the results it seem more likely that efforts to retrieve different types of data, for example spreading the collected stacks over a larger amount of patients, will give better results. Using more clever additional pre-processing steps will also improve the results.

Lastly, an initial attempt to use a classifying CNN together with a sliding window approach to locate organs and body-parts was conducted. The achieved result are not satisfying but show promise to use such approach for localization in medical CT images. As discussed in the previous chapter there are major improvements to be done in terms of labelling and extraction of training data that would likely improve the localization results.

6.1 Future Work

This study presents a CNN for classification of medical CT images. Several limitations to the study have been discussed in Chapter 5. In this section ideas of related further work are presented based on the limitations presented.

6.1.1 Further localization

CT data is as described in the thesis aligned into stacks. By implementing a 2D classifier each image in such stack could be passed into the system and be classified. When all images in one stack have been classified the classification can be transferred into a table of content for the stack that contains the organs or body-parts that are depicted and the z-coordinates (slice index of stack) of the images within the stack. To take this one step further localization techniques as in Section 2.5.2 can be used to locate the object in each 2D plane giving the x- and y-coordinate. Parts of a sliding window approach has been investigated in this work but to improve and analyze this approach further is left as future work. Using regression for localization has also been proposed in the thesis but has not been investigated, an approach that can be experimented with as future work.

An accurate localization in the 2D image would improve the table of content to not only provide z-coordinates for the location but a complete 3D bounding box describing the location of organ or body-part within a stack.

6.1.2 Altering the study

It was found in this study that splitting the dataset by stack might effect the result since images of the same patient can reside both in the training and test set. As further work it would be interesting to split the dataset by patient to investigate the effect on the results.

The dataset used for this study was labelled by the authors. The limited knowledge of radiological images when labelling limited the number of classes to 15. If experts can be consulted to do the labelling the case with additional classes can be used to investigate the effect of more fine labelling. A great addition to the scientific community would be a project where annotated public datasets for medical images were created.

The CNN in this study was fed with pixeldata only. CNNs can however process multiple inputs. There is a lot of information in the DICOM meta-data that can be added as input to the system. One option is to simply pass all the meta-data parameters to a CNN. A layer can have multiple input, for example VGG16 can still be used as base network to generate features from the images. An additional CNN can be constructed to process the text from the DICOM tags. The first fully connected layer in the implementation can then take two inputs, one from each of the described CNNs, and produce the output from all the data. To do this one must find a suitable CNN architecture for the DICOM tags, possibly investigate feature extraction and feature selection techniques that can aid classification. Suitable pre-processing techniques for the DICOM tags should also be investigated before feeding the data to the network.

Another alteration that can be done to the architecture of the CNN is to introduce 3D convolutional layers and use a full stack as input instead of 2D slices. Since the stacks are aligned in 3D volumes it would be appropriate to use as input to a 3D convolutional neural network. Using this approach the training dataset in this study would likely be too small since each stack would represent one input sample.

Bibliography

[1] Yoshua Bengio. "Practical Recommendations for Gradient-Based Training of Deep Architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478. arXiv: arXiv:1206.5533v2. URL: https://arxiv.org/pdf/1206.5533.pdf.

[2] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305. URL: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

[3] Monique Brouillette. *Deep Learning Is a Black Box, but Health Care Won't Mind - MIT Technology Review*. 2017. URL: https://www.technologyreview.com/s/604271/deep-learning-is-a-black-box-but-health-care-wont-mind/.

[4] Thorsten M. Buzug. *Computed Thomography From Photon Statistics to Modern Cone-Beam CT*. Springer-Verlag Berlin Heidelberg, 2008, p. 522. DOI: 10.1007/978-3-540-39408-2.

[5] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Return of the Devil in the Details : Delving Deep into Convolutional Nets". In: *arXiv preprint arXiv:1405.3531* (2014), pp. 1–11. arXiv: arXiv:1405.3531v4.

[6] Hao Chen, Dong Ni, Jing Qin, Shengli Li, Xin Yang, Tianfu Wang, and Pheng Ann Heng. "Standard Plane Localization in Fetal Ultrasound via Domain Transferred Deep Neural Networks". In: *IEEE Journal of Biomedical and Health Informatics* 19.5 (2015), pp. 1627–1636. ISSN: 21682194. DOI: 10.1109/JBHI.2015.2425041.

[7] Junghwan Cho, Kyewook Lee, Ellie Shin, Garry Choy, and Synho Do. "How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?" In: *arXiv preprint arXiv: 1511.06348* (2015). arXiv: arXiv:1511.06348v2.

[8] Francois Chollet. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[9] Anders Eklund, Paul Dufort, Daniel Forsberg, and Stephen M M. LaConte. "Medical image processing on the GPU – Past, present and future". In: *Medical Image Analysis* 17.8 (2013), pp. 1073–1094. ISSN: 1361-8415.

[10] Ridley Erik L. *Part 2: How will AI affect radiology?* 2017. URL: http://www.auntminnie.com/index.aspx?sec=sup&sub=aic&pag=dis&ItemID=117245.

[11] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. "Dermatologist-level classification of skin cancer with deep neural networks". In: *Nature* 542.7639 (Jan. 2017), pp. 115–118. ISSN: 0028-0836. DOI: 10.1038/nature21056. URL: http://www.nature.com/doifinder/10.1038/nature21056.

[12] Mark Everingham, Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. "The PASCAL Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.180.3435.

[13] Carl Benedikt Frey and Osborne Michael A. "The Future of Employment: How Susceptible are Jobs to Computerisation?" In: *Technological Forecasting and Social Change* 114 (2017), pp. 254–280. URL: http://www.oxfordmartin.ox.ac.uk/downloads/academic/The_Future_of_Employment.pdf.

[14] Ross Girshick. "Fast R-CNN". In: *Proceedings of the IEEE International Conference on Computer Vision*. Apr. 2015, pp. 1440–1448. arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Nov. 2014, pp. 580–587. arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[16] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. 2010, pp. 249–256. URL: http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf.

[17] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[18] Mark O. Gueld, Michael Kohnen, Daniel Keysers, Henning Schubert, Berthold B. Wein, Joerg Bredno, and Thomas M. Lehmann. "Quality of DICOM header information for image categorization". In: *Proc. SPIE*. Ed. by Eliot L. Siegel and H. K. Huang. International Society for Optics and Photonics, May 2002, pp. 280–287. DOI: 10.1117/12.467017. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=880364.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. URL: https://arxiv.org/pdf/1512.03385.pdf.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034. URL: https://arxiv.org/pdf/1502.01852.pdf.

[21] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012), pp. 1–18. ISSN: 9781467394673. arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[22] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. "What makes for effective detection proposals?" In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.4 (Feb. 2016), pp. 814–830. DOI: 10.1109/TPAMI.2015.2465908. arXiv: 1502.05082. URL: http://arxiv.org/abs/1502.05082.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems* (2012), pp. 1–9. ISSN: 10495258. arXiv: 1102.0183.

[24] Ashnil Kumar, Jinman Kim, David Lyndon, Michael Fulham, and Dagan Feng. "An Ensemble of Fine-Tuned Convolutional Neural Networks for Medical Image Classification". In: *IEEE Journal of Biomedical and Health Informatics* 21.1 (Jan. 2017), pp. 31–40. ISSN: 2168-2194. DOI: 10.1109/JBHI.2016.2635663. URL: http://ieeexplore.ieee.org/document/7769199/.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539. URL: http://www.nature.com/doifinder/10.1038/nature14539.

[26] Gongning Lou, Ran An, Kuanquan Wang, Suyu Dong, and Henggui Zhang. "MRI, A deep learning network for right ventricle segmentation in short-axis". In: *2016 Computing in Cardiology Conference (CinC)*. 2016, pp. 485–488.

[27] J.N.S. Matthews, Geoffrey Berry, and Peter Armitage. *Statistical Methods in Medical Research*. Wiley, 2008, p. 139. ISBN: 9780632052578.

[28] Mesko Bertalan. *Can An Algorithm Diagnose Better Than A Doctor?* -. 2016. URL: http://medicalfuturist.com/can-an-algorithm-diagnose-better-than-a-doctor/.

[29] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020, 9780262018029.

[30] Vinod Nair and Geoffrey E Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning* 3 (2010), pp. 807–814.

[31] The Association of Electrical Equipment NEMA and Medical Imaging Manufacturers. *DICOM*. 2016. URL: http://dicom.nema.org/.

[32] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: An astounding baseline for recognition". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 512–519. ISBN: 9781479943098. DOI: 10.1109/CVPRW.2014.131. arXiv: 1403.6382.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[34] C.J Rijsbergen. *Information Retrieval, 2nd edition*. 1979. ISBN: 978-1-4020-3004-8. DOI: 10.1007/978-1-4020-3005-5.

[35] Holger R. Roth, Christopher T. Lee, Hoo-Chang Shin, Ari Seff, Lauren Kim, Jianhua Yao, Le Lu, and Ronald M. Summers. "Anatomy-specific classification of medical images using deep convolutional nets". In: *IEEE 12th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 2015, pp. 101–104. DOI: 10.1109/ISBI.2015.7163826. arXiv: 1504.04003.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252. ISSN: 0920-5691. DOI: 10.1007/s11263-015-0816-y. URL: http://link.springer.com/10.1007/s11263-015-0816-y.

[37] Berkman Sahiner, Heang Ping Chan, Nicholas Petrick, Datong Wei, Mark A. Helvie, Dorit D. Adler, and Mitchell M. Goodsitt. "Classification of mass and normal breast tissue: A convolution neural network classifier with spatial domain and texture images". In: *IEEE Transactions on Medical Imaging* 15.5 (1996), pp. 598–610. ISSN: 02780062. DOI: 10.1109/42.538937.

[38] Takaya Saito and Marc Rehmsmeier. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". In: *PloS one* 10.3 (Mar. 2015). Ed. by Guy Brock. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0118432. URL: http://www.ncbi.nlm.nih.gov/pubmed/25738806.

[39] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *arXiv preprint arXiv:1312.6229* (Dec. 2013). arXiv: 1312.6229. URL: http://arxiv.org/abs/1312.6229.

[40] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556* (Sept. 2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[41] Marina Sokolova and Guy Lapalme. "A systematic analysis of performance measures for classification tasks". In: *Information Processing and Management* 45.4 (July 2009), pp. 427–437. ISSN: 03064573. DOI: 10.1016/j.ipm.2009.03.002. URL: http://linkinghub.elsevier.com/retrieve/pii/S0306457309000259.

[42] N Srivastava, GE Hinton, and A Krizhevsky. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958. URL: http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf.

[43] Richard E. Susskind and Daniel Susskind. *The future of the professions : how technology will transform the work of human experts*. 2015, p. 346. ISBN: 9780198713395.

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf.

[45] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. "Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?" In: *IEEE Transactions on Medical Imaging* 35.5 (May 2016), pp. 1299–1312. ISSN: 0278-0062. DOI: 10.1109/TMI.2016.2535302. URL: http://ieeexplore.ieee.org/document/7426826/.

[46] J R R Uijlings, K E A Van De Sande, T Gevers, and A W M Smeulders. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171.

[47] Bernard Widrow and Michael A Lehr. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation". In: *Proceedings of the IEEE*. Vol. 78. 9. 1990, pp. 1415–1442. URL: https://pdfs.semanticscholar.org/8b73/adda15fa71b0a35ffedb899d6a72d621923b.pdf.

[48] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

A Class Specific Results

This presents the results for each class more in depth. Firstly the values of the metrics for each class are presented in Section A.1. The next Section contains confusion matrices displaying the exact number of true positives, true negatives, false positives and false negatives for each class. Lastly, Section A.3 contain tables showing what has been predicted instead of the right class in the case where a misclassification occur.

A.1 Metrics

Metric	Value (%)
Accuracy	99.81
Error	0.19
Precision	100.0
Recall	97.66

(a) Head

Metric	Value (%)
Accuracy	99.81
Error	0.19
Precision	100.0
Recall	97.89

(b) Neck

Metric	Value (%)
Accuracy	99.29
Error	0.71
Precision	68.03
Recall	76.34

(c) Shoulders

Metric	Value (%)
Accuracy	98.11
Error	1.89
Precision	96.37
Recall	84.24

(d) Chest

Metric	Value (%)
Accuracy	97.38
Error	2.62
Precision	84.37
Recall	80.0

(e) Abdomen

Metric	Value (%)
Accuracy	99.03
Error	0.97
Precision	96.95
Recall	96.78

(f) Pelvis

Metric	Value (%)
Accuracy	99.52
Error	0.48
Precision	99.37
Recall	97.9

(g) Femur

Metric	Value (%)
Accuracy	99.21
Error	0.79
Precision	91.81
Recall	96.71

(h) Knee

Metric	Value (%)
Accuracy	99.23
Error	0.77
Precision	99.1
Recall	95.92

(i) Tibia

Metric	Value (%)
Accuracy	99.77
Error	0.23
Precision	99.7
Recall	97.81

(j) Ankles

Metric	Value (%)
Accuracy	99.04
Error	0.96
Precision	97.71
Recall	93.57

(k) Right Lung

Metric	Value (%)
Accuracy	98.52
Error	1.48
Precision	95.72
Recall	90.81

(l) Left Lung

Metric	Value (%)
Accuracy	98.2
Error	1.8
Precision	95.24
Recall	73.28

(m) Liver

Metric	Value (%)
Accuracy	97.59
Error	2.41
Precision	43.51
Recall	49.14

(n) Right Kidney

Metric	Value (%)
Accuracy	98.88
Error	1.12
Precision	73.27
Recall	67.89

(o) Left Kidney

Table A.1: Metrics for each class separately.

A.2 Confusion Matrices

876	21
0	10139

(a) Head

973	21
0	10042

(b) Neck

100	31
47	10858

(c) Shoulders

930	174
35	9897

(d) Chest

664	166
123	10083

(e) Abdomen

1651	55
52	9278

(f) Pelvis

1907	41
12	9076

(g) Femur

706	24
63	10243

(h) Knee

1644	70
15	9307

(i) Tibia

983	22
3	10028

(j) Ankles

1150	79
27	9780

(k) Right Lung

1117	113
50	9756

(l) Left Lung

480	175
24	10357

(m) Liver

114	118
148	10656

(n) Right Kidney

148	70
54	10764

(o) Left Kidney

true positives	false negatives
false positives	true negatives

(p) Confusion Matrix Interpretation

Table A.2: Confusion matrices for each class. Table (p) shows the interpretation for each cell in the tables.

A.3 Per Class Misclassification

The predictions are stored in the format of a vector with zeros for non-existing classes and ones for existing classes. Example [1 0 0 1] means that class 1 and 4 have been predicted to true and 2 and 3 to false. The ground truth follows the same format.

For each prediction, if a class is set to 1 in the ground truth vector but to 0 in the predicted vector there is a miss (false negative). For each miss in each class the prediction and ground truth vectors are scanned to find false positives that might have been predicted instead of the right class. These occurences are counted as the alternative class/classes predicted for each miss.

Since multi-class multi label classification was used in this study, it is possible that one miss may have resulted in multiple false positives being counted. It is also possible that if there are two misses in the predicted vector for one sample, the existing false positives in that predicted vector will be counted for both the misses. A short example:

The ground truth vector is [1 1 0 0 0]. The predicted vector is [0 0 1 1 0]. The first and second indices are not predicted correctly, which means there might exists false positives (index 3 and 4). In this example the classes with index 3 and 4 will be counted as the alternative class predicted for both the class with index 1 and index 2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
head (1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	**19**
neck(2)	0	0	**9**	0	0	0	0	0	0	0	0	0	0	0	0	**12**
shoulders (3)	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	**29**
chest (4)	0	0	0	0	**1**	**1**	**24**	**3**	0	0	0	0	0	0	0	**146**
r lung (5)	0	0	0	0	0	0	**35**	**10**	0	0	0	0	0	0	0	**44**
l lung (6)	0	0	0	0	0	0	**29**	0	0	0	0	0	0	0	0	**84**
abdomen (7)	0	0	0	**4**	**17**	**16**	0	**3**	**1**	**1**	**27**	0	0	0	0	**117**
liver (8)	0	0	0	**13**	**14**	**10**	**2**	0	**27**	**17**	0	0	0	0	0	**124**
r kidney (9)	0	0	0	0	0	0	**1**	**5**	0	**1**	**12**	0	0	0	0	**99**
l kidney (10)	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	**69**
pelvis (11)	0	0	0	0	0	0	**4**	0	0	0	0	**6**	0	0	0	**45**
femur (12)	0	0	0	0	0	0	0	0	0	0	0	0	**8**	0	0	**33**
knee (13)	0	0	0	0	0	0	0	0	0	0	0	**4**	0	**8**	0	**12**
tibia (14)	0	0	0	0	0	0	0	0	0	0	0	0	**2**	0	**1**	**67**
ankles (15)	0	0	0	0	0	0	0	0	0	0	0	0	0	**3**	0	**19**

Table A.3: The rows represent the class for which we have seen a false negative and the columns show what was predicted instead of the actual class. The columns are indexed the same way as the rows with numbers, with column *16* being no class at all (the rest of the prediction vector is correct).

The same information as shown in Table A.3 above is shown in Table A.4. However, instead of finding false negatives in the prediction vector false positives are found and occurrences of false negatives are counted instead of occurrences false positives.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
head (1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
neck (2)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shoulders (3)	0	**9**	0	0	0	0	0	0	0	0	0	0	0	0	0	**38**
chest (4)	0	0	**1**	0	0	0	**4**	**13**	0	0	0	0	0	0	0	**21**
r lung (5)	0	0	**1**	**1**	0	0	**17**	**14**	0	0	0	0	0	0	0	**5**
l lung (6)	0	0	**1**	**1**	0	0	**16**	**10**	0	0	0	0	0	0	0	**32**
abdomen (7)	0	0	0	**24**	**35**	**29**	0	**2**	**1**	0	**4**	0	0	0	0	**63**
liver (8)	0	0	0	**3**	**10**	0	**3**	0	**5**	0	0	0	0	0	0	**6**
r kidney (9)	0	0	0	0	0	0	**1**	**27**	0	0	0	0	0	0	0	**120**
l kidney (10)	0	0	0	0	0	0	**1**	**17**	**1**	0	0	0	0	0	0	**36**
pelvis (11)	0	0	0	0	0	0	**27**	0	**12**	**1**	0	0	0	0	0	**13**
femur (12)	0	0	0	0	0	0	0	0	0	0	**6**	0	**4**	0	0	**2**
knee (13)	0	0	0	0	0	0	0	0	0	0	0	**8**	0	**2**	0	**53**
tibia (14)	0	0	0	0	0	0	0	0	0	0	0	0	**8**	0	**3**	**4**
ankles (15)	**2**	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Table A.4: The rows represent the class for which we have seen a false positive and the columns show what was predicted instead of the actual class. The columns are indexed the same way as the rows with numbers, with column *16* being no class at all (the rest of the prediction vector is correct).

B Localization with Sliding Window Results

This appendix contains results related to the localization task.

B.1 Intersect over Union Scores

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	60.0	0.5	0.6	0.81	0.0
neck	0.0	0.08	0.0	0.44	0.0
shoulders	0.0	0.0	0.0	0	0
chest	0.0	0.0	0.0	0	0
abdomen	0.0	0.0	0.0	0	0
pelvis	0.0	0.0	0.0	0	0
femur	0.0	0.0	0.0	0	0
knee	0.0	0.0	0.0	0	0
tibia	0.0	0.01	0.0	0.09	0.0
ankles	0.0	0.33	0.34	0.44	0.2
right lung	0.0	0.0	0.0	0	0
left lung	0.0	0.0	0.0	0	0
liver	0.0	0.0	0.0	0	0
right kidney	0.0	0.0	0.0	0	0
left kidney	0.0	0.0	0.0	0	0
all classes	2.52	0.06	0.0	0.81	0.0

Table B.1: Average Heatmap, All Scales, Fixed Threshold

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	100.0	0.71	0.77	0.82	0.55
neck	11.11	0.1	0.0	0.52	0.0
shoulders	0.0	0.08	0.0	0.46	0.0
chest	60.87	0.44	0.6	0.76	0.0
abdomen	42.42	0.35	0.36	0.88	0.0
pelvis	90.0	0.63	0.7	0.82	0.0
femur	0.0	0.02	0.0	0.09	0.0
knee	0.0	0.0	0.0	0.02	0.0
tibia	10.0	0.25	0.26	0.58	0.0
ankles	0.0	0.4	0.41	0.44	0.23
right lung	0.0	0.0	0.0	0	0
left lung	0.0	0.01	0.0	0.14	0.0
liver	0.0	0.06	0.0	0.33	0.0
right kidney	0.0	0.0	0.0	0	0
left kidney	0.0	0.0	0.0	0	0
all classes	20.76	0.2	0.0	0.88	0.0

Table B.2: Average Heatmap, Selected Scales, Fixed Threshold

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	70.0	0.62	0.75	0.9	0.26
neck	11.11	0.23	0.15	0.59	0.05
shoulders	42.86	0.5	0.46	0.66	0.37
chest	39.13	0.36	0.27	0.74	0.01
abdomen	41.94	0.37	0.3	0.76	0.03
pelvis	40.0	0.46	0.43	0.72	0.11
femur	70.0	0.49	0.59	0.62	0.22
knee	0.0	0.15	0.09	0.4	0.05
tibia	11.11	0.34	0.35	0.51	0.01
ankles	20.0	0.44	0.43	0.58	0.28
right lung	6.9	0.23	0.24	0.72	0.0
left lung	17.24	0.31	0.33	0.69	0.04
liver	0.0	0.33	0.33	0.42	0.21
right kidney	0.0	0.22	0.18	0.41	0.09
left kidney	0.0	0.14	0.14	0.19	0.07
all classes	24.11	0.35	0.31	0.9	0.0

Table B.3: Average Heatmap, Selected Scales, Adapting Threshold

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	80.0	0.63	0.65	0.8	0.37
neck	22.22	0.27	0.13	0.72	0.0
shoulders	85.71	0.6	0.55	0.75	0.48
chest	91.3	0.65	0.68	0.82	0.23
abdomen	62.5	0.53	0.54	0.93	0.0
pelvis	90.0	0.62	0.65	0.7	0.42
femur	70.0	0.55	0.59	0.73	0.0
knee	0.0	0.06	0.05	0.16	0.0
tibia	0.0	0.38	0.37	0.46	0.35
ankles	0.0	0.4	0.42	0.44	0.24
right lung	0.0	0.04	0.0	0.21	0.0
left lung	3.45	0.15	0.05	0.51	0.0
liver	6.25	0.28	0.27	0.52	0.11
right kidney	0.0	0.15	0.14	0.42	0.0
left kidney	0.0	0.16	0.17	0.22	0.0
all classes	33.04	0.36	0.37	0.93	0.0

Table B.4: Merged Boxes, All Scales, Fixed Threshold

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	80.0	0.63	0.65	0.8	0.37
neck	22.22	0.27	0.13	0.72	0.0
shoulders	85.71	0.6	0.55	0.75	0.48
chest	91.3	0.65	0.68	0.82	0.23
abdomen	62.5	0.53	0.54	0.93	0.0
pelvis	90.0	0.62	0.65	0.7	0.42
femur	70.0	0.55	0.59	0.73	0.0
knee	0.0	0.06	0.05	0.16	0.0
tibia	0.0	0.38	0.37	0.46	0.35
ankles	0.0	0.4	0.42	0.44	0.24
right lung	0.0	0.04	0.0	0.21	0.0
left lung	3.45	0.15	0.05	0.51	0.0
liver	6.25	0.28	0.27	0.52	0.11
right kidney	0.0	0.15	0.14	0.42	0.0
left kidney	0.0	0.16	0.17	0.22	0.0
all classes	33.04	0.36	0.37	0.93	0.0

Table B.5: Merged Boxes, Selected Scales, Fixed Threshold

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	70.0	0.58	0.57	0.84	0.36
neck	0.0	0.13	0.13	0.31	0.02
shoulders	14.29	0.3	0.35	0.56	0.06
chest	34.78	0.4	0.38	0.79	0.09
abdomen	19.35	0.34	0.35	0.57	0.04
pelvis	40.0	0.46	0.41	0.77	0.25
femur	10.0	0.43	0.44	0.53	0.29
knee	0.0	0.14	0.11	0.33	0.05
tibia	22.22	0.36	0.38	0.55	0.14
ankles	40.0	0.46	0.43	0.86	0.24
right lung	13.79	0.27	0.27	0.66	0.0
left lung	6.9	0.3	0.33	0.52	0.06
liver	6.25	0.33	0.34	0.54	0.19
right kidney	0.0	0.21	0.2	0.3	0.16
left kidney	0.0	0.03	0.0	0.18	0.0
all classes	17.86	0.32	0.31	0.86	0.0

Table B.6: Merged Boxes, All Scales, Adapting Threshold

Class	Hits (%)	Average iou	Median iou	Max iou	Min iou
head	70.0	0.58	0.57	0.84	0.36
neck	0.0	0.13	0.13	0.31	0.02
shoulders	14.29	0.3	0.35	0.56	0.06
chest	34.78	0.4	0.38	0.79	0.09
abdomen	19.35	0.34	0.35	0.57	0.04
pelvis	40.0	0.46	0.41	0.77	0.25
femur	10.0	0.43	0.44	0.53	0.29
knee	0.0	0.14	0.11	0.33	0.05
tibia	22.22	0.36	0.38	0.55	0.14
ankles	40.0	0.46	0.43	0.86	0.24
right lung	13.79	0.27	0.27	0.66	0.0
left lung	6.9	0.3	0.33	0.52	0.06
liver	6.25	0.33	0.34	0.54	0.19
right kidney	0.0	0.21	0.2	0.3	0.16
left kidney	0.0	0.03	0.0	0.18	0.0
all classes	17.86	0.32	0.31	0.86	0.0

Table B.7: Merged Boxes, Selected Scales, Fixed Threshold