Julius-Maximilians-Universität Würzburg

Institut für Informatik
Lehrstuhl für Künstliche Intelligenz
und Wissenssysteme

Bachelorarbeit

im Studiengang Luft- und Raumfahrtinformatik

zur Erlangung des akademischen Grades
Bachelor of Science

Automated Detection of Lung Nodules in CT Scans using Convolutional Neural Networks

---

Autor:	Ivaylo Angelov Matrikelnummer 2099396
Abgabe:	28.09.2020
1. Betreuer:	Prof. Dr. Frank Puppe
2. Betreuer:	M. Sc. Amar Hekalo

Abstract

The present thesis examines the usage of neural networks, in particular Convolutional Neural Networks, for the detection of lung nodules in CT scans. The aim of this work is the implementation of a nodule detection framework that allows the integration of different datasets and models. For this purpose it initially provides a detailed overview of the current state of research, thereby focusing on image preprocessing, data augmentation and promising network architectures.
Employing the LNDb-dataset[1] released in 2019 and an auspicious model architecture, the training procedure of a lung nodule detection network is demonstrated. The model is based on the architecture presented in [2] and uses a custom 3D-Single-Shot-Detector as a detector head.
The model output is evaluated with the common object detection metrics precision and recall, as well as with the FROC-score, which is a popular metric in medical applications. We achieve a maximal recall of 0.68 at 42963 false positives per scan according to the FROC-calculation procedure in the LNDb-Challenge[3]. The calculation of precision and recall on the COCO IoU-range as well as the FROC-score for the typical set of $\{ \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8 \}$ FPs per scan yield a result of zero due to the low accuracy of box predictions and the herefrom resulting lack of true positives. Thus the method presented in this work is still not well enough trained in order to perform lung nodule detection. However, more training time and the incorporation of additional datasets, image processing procedures and training steps that are mentioned promise to lead to a significantly better performance in the future.

Zusammenfassung

Die vorliegende Bachelorarbeit befasst sich mit der Nutzung Neuronaler Netze, speziell Convolutional Neural Networks, für die Detektion von Lungenknoten in CT Scans. Ziel der Arbeit ist die Implementierung eines Frameworks, in das verschiedene Datensätze und Modelle eingebaut werden können, um die Detektion von Lungenknoten zu trainieren. Hierfür wird zunächst ein detaillierter Überblick über den aktuellen Forschungsstand mit Schwerpunkten in der Vorverarbeitung von Bildern, Datenaugmentierung und erfolgsversprechenden Netzwerkarchitekturen gegeben. Unter Nutzung des 2019 erschienenen LNDb-Datensatzes[1] wird mit einem vielversprechenden Modell der Trainingsablauf eines, auf die Erkennung von Lungenknoten spezialisierten, Netzes demonstriert. Das Modell basiert auf der unter [2] vorgestellten Architektur und benutzt einen eigenen 3D-Single-Shot-Detector als Detector-Head.
Zur Evaluation wurden die herkömmlichen Klassifikationsmetriken Precision und Recall, sowie der, für medizinische Anwendungen beliebte, FROC-Score verwendet. Das präsentierte Modell erzielt einen Recall von 0.68 bei 42963 False Positives pro Scan nach dem unter [1] beschriebenen FROC-Algorithmus. Aufgrund der geringen Genauigkeit der Box-Vorhersagen des Modells sind Precision und Recall über dem COCO IoU-Bereich[4] sowie der FROC-score für die typischen Werte von $\{ \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8 \}$ FPs pro Scan gleich null. Das vorgestellte Modell ist daher noch ungeeignet, um Lungenknoten vorherzusagen. Ein größerer Zeitrahmen und der Einbau weiterer Datensätze sowie in dieser Arbeit erwähnter Datenverarbeitungsschritte und Trainingstechniken versprechen jedoch deutlich bessere Resultate in Zukunft.

Contents

List of Symbols

ANN	Artificial Neural Network
CAD	Computer-aided diagnosis
CNN	Convolutional Neural Network
DDCB	Densely Dilated Convolutional Block
DenseNet	Densely Connected Convolutional Network
Fast-RCNN	Fast Region-based Convolutional Neural Networks
Faster-RCNN	Faster Region-based Convolutional Neural Networks
FCN	Fully Convolutional Network
FPN	Feature Pyramid Network
FPR	False Positive Reduction
FROC	Free-response Receiver Operating Characteristics
GD	Gradient Descent
HU	Hounsfield Unit
IoU	Intersection over Union
ML	Machine Learning
NN	Neural Network
RCNN	Region-based Convolutional Neural Networks
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
ROC	Receiver Operating Characteristics
ROI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SSD	Single Shot (Multibox) Detector
SVM	Support Vector Machine

List of Figures

List of Tables

1 Introduction

During the last years the development and use of deep learning for *Computer-aided diagnosis* (CAD) - systems has rapidly increased[5, 6]. One popular and promising field of application is the detection of lung nodules and cancer risk estimation in low-dose Computer Tomography (CT) scans. The aim of such systems is to improve the precision of diagnoses by providing models which, on the basis of large training sets, are able to learn to recognize various features within scans and generalize on new cases [6]. Such models can ease the work of doctors and thereby speed up the analysis and reduce medical costs. With this in mind, deep *Convolutional Neural Networks* (CNNs) have proven to be a propitious type of deep learning model for the early detection of lung cancer[6][5].

1.1 Motivation and goals

Lung cancer is among the most common and deadly types of cancer [7]. The use of deep neural networks for the detection and risk estimation of lung nodules in CT scans can be an important step towards earlier identification and prevention of a large number of potentially fatal cases. The present work deals with the detection of lung nodules, which is essentially an object detection task in three-dimensional images. A variety of approaches exist, in which models are trained based on two-dimensional inputs in the form of slices taken from the original three-dimensional scan. However, the three-dimensional model applied in this work deals with volumetric input data. A serious obstacle for the task of lung nodule detection with neural networks is the sparse availability of datasets as well as the lack of uniformity among them. They appear in various formats and with vastly different annotation processes. In this context, in November 2019 the *LNDb-Grand Challenge*[8] introduced a new dataset consisting of 294 CT scans, annotated in more detail than previous sets by at least one of a group of five radiologists.

This work aims to develop the base of an expandable framework to both train and evaluate different deep learning models on various datasets. With this goal in view, an existing and promising model architecture[2] will be trained to detect nodules in the LNDb-dataset. The resulting network is a three-dimensional feature extractor inspired by the *U-Net* architecture. Its detector head is a three-dimensional variant of the *2D-Single Shot Detector*[9] and makes use of depth data.

Having a foundation for preprocessing images and integrating models, the aim is to add more datasets and network structures in the future.

1.2 Outline

The present work is structured as follows: The upcoming chapter explains the needed theoretical foundations. Chapter 3 is devoted to analyzing related work that deals with lung nodule detection and introducing available datasets. Subsequently, chapter 4 describes the training setup applied and evaluated in this work in detail. In the following section 5 the results of the approach taken here are presented. Furthermore, the results are compared to the performance of other models and possible improvements are outlined. Finally, section 6 summarizes the work and provides an outlook into the future development of the framework and research.

2 Theoretical Framework

This chapter introduces the necessary theoretical background for the thesis.

2.1 Neurons and Neural Networks

Neural Networks (NN), often referred to as *Artificial Neural Networks* (ANN), are a highly multifaceted type of machine learning (ML) algorithm. Loosely inspired by the human brain, which contains neurons and interneuron connections and acquires knowledge through a learning process [10, Chapter 1], they are able to solve computational problems by mapping an input of data to a desired output. In this chapter the basic parts and operating principles of NNs are introduced based on the detailed explanations in [11], [12] and [10]. The fundamental parts of an NN, the neurons, are explained in section 2.1.1. Section 2.1.2 discusses how they can be combined to more complex architectures of networks. The way NNs are trained and optimized is explained in section 2.1.3. Section 2.1.4 covers hyperparameters and their importance for the training process. Finally, in section 2.1.5 the issue of *overfitting* and a possible way to prevent it, the *Dropout* technique, are presented.

2.1.1 The Neuron

A neuron, the basic building block of an NN, is an object that takes a certain amount of inputs $x_1 ... x_n$ and produces an output y, which can be continuous, binary or categorical (in a sense that it produces one of several possible outputs and therefore categorizes outputs). Figure 1 shows a useful model representation of a neuron. To each input corresponds a positive or negative weight $w_{1,...,m}$ which denotes this input's importance for the output calculation. In the introduced model, the neuron *k* firstly weights and sums the inputs according to the following rule:

$$S_k = \sum_{i=1}^{n} w_{ki} x_i + b_k \tag{1}$$

The *bias* b_k can be described as an additional degree of freedom or offset for each neuron, because it is added to the weighted sum over the inputs before producing the final output. The last part of a neuron's task is applying a so-called *activation function*, which is the topic of the next paragraph.

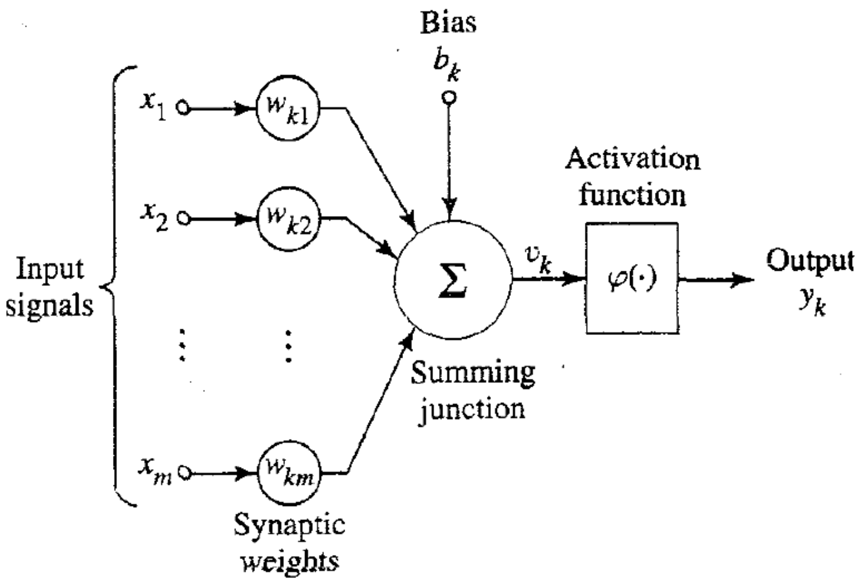


Figure 1: Model representation of a neuron as depicted in [10]. Inputs are weighted and summed over, introducing a neuron-specific *bias*. The output is then generated by applying an activation function to the weighted sum.

2.1.1.1 Activation Functions

A neuron's activation function ϕ takes in its weighted sum of inputs S_k and bias b_k and produces the output y_k:

$$y_k = \phi(S_k) \tag{2}$$

This illustrates qualitatively how the weights play a role in the neuron's output and why the bias is referred to as an offset introduced by a neuron. Quantitatively, of course, we need to know the exact form of the activation function, bearing in mind that it will have implications on the types of problems an NN can solve. Figure 2 shows a few typical activation functions, The activation function is crucial in order to make the model non-linear in the attempt to solve complex problems with minimal amount of neurons and therefore maximal efficiency [13].

Linear Function

$$\phi(x) = c * x \tag{3}$$

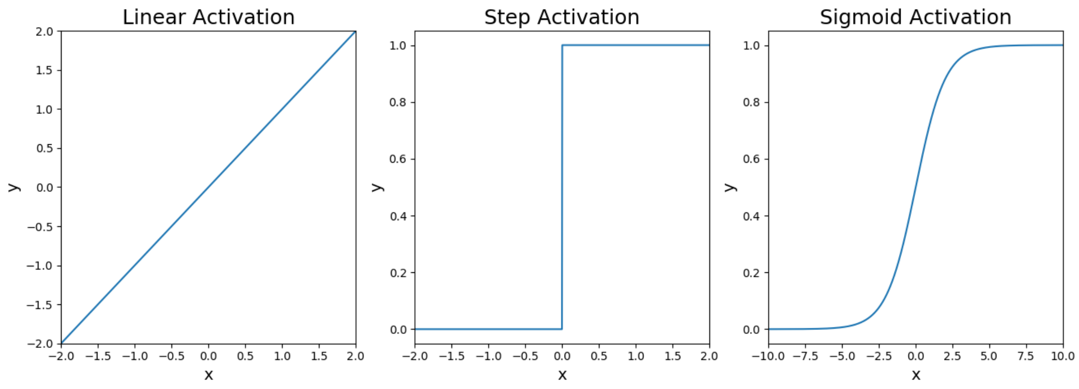


Figure 2: Different activation functions for neurons.

A linear neuron maps the input to an output by simply multiplying it with a factor *c*. An exclusive usage of linear neurons throughout a network results in a linear regression, for which a feedforward NN architecture as described in 2.1 is not necessary. Neural networks show their real potential when non-linearities are introduced. Some of the most frequently used ones will be shown now.

Step Function

$$\phi(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{4}$$

The step function is one of the simplest activation functions. It maps the input onto a binary output, 0 or 1. It is basically requiring a certain minimal value (here: 0) from the inputs in order to produce a value of 1. That corresponds to a minimal value *activating* the neuron. Nevertheless, it can be problematic for network optimization algorithms because it is not globally differentiable. This will be discussed in section 2.1.3.4.

Sigmoid Function

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

The sigmoid function is a recurringly used activation function and is especially present in predicting probability-based output, for binary classification and logistic regression problems [14]. Being differentiable everywhere, it overcomes the network optimization issues discussed in the last section.

Softmax Function

$$\phi(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \tag{6}$$

The softmax function for the j-th neuron of the output layer is defined in eq.6. The sum in the denominator goes over all neurons in the layer, normalizing the expression and therefore the sum over all softmax results in the layer is one. The result can thus be interpreted as a probability distribution, justifying its usage for classification. The network in that case learns to assign probabilities to inputs belonging to a certain class.

Rectifier Function

$$\phi(x) = \max(0, x) \tag{7}$$

The linear by part rectifier function, also referred to as a ***R**ectified **L**inear **U**nit* or ReLU, is becoming more popular in various networks. This is, among other reasons, because they are a more suiting model of a biological neuron and automatically introduce certain sparseness into the network layers due to the mapping of a range of inputs to 0 [15]. Possible modifications to the ReLU function, named *Leaky ReLU / PReLU* and *Randomized Leaky ReLU*, are shown in Fig.3.

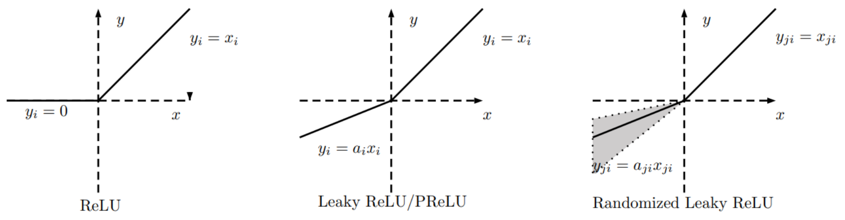


Figure 3: The ReLU, Leaky ReLU, PReLU and Randomized Leaky ReLU activation functions[16]. Unlike ReLU, the other types have a non-zero slope a_i for $x < 0$. This slope is fixed for Leaky ReLU, learnt and set to a constant value for PReLU and randomly sampled from an input range for Randomized Leaky ReLU.

2.1.2 Combining Neurons to a Network

In almost any problem in which ML algorithms are used, the goal is to obtain one or more outputs from a set of inputs. In a broader sense, the objective is to approximate a certain multidimensional function. For that, neurons are structured into *layers*, inside which neurons operate parallelly. It is convenient at this point to introduce the labeling of a neuron's output as ${a_k}^l$, denoting the output of the k-th neuron in layer l. The output of one whole layer, which is now a vector with a number of entries corresponding to the number of neurons in that layer, becomes the input of the next layer. The neurons of two consecutive layers can be connected and those connections carry the weights that were introduced in 2.1.1. Typically, the flow of information is in one direction only, going from the input to the output layer, as shown in Fig.4. This feedforward type of network is the most basic type of NN and one of many approaches to build a network for solving the problem of finding the desired function.

At this stage it is useful to elucidate the meaning of a *forward pass* — it describes the action of taking an input, propagating it through the network layer by layer and producing an output. The forwarding from a layer l to a layer $l+1$ essentially consists of multiplying a matrix containing all weights of the connections between the l-th and $l+1$-th layer onto the vector containing the outputs of the layer l. The hereby obtained vector is the input for the new layer (the $l+1$-th).

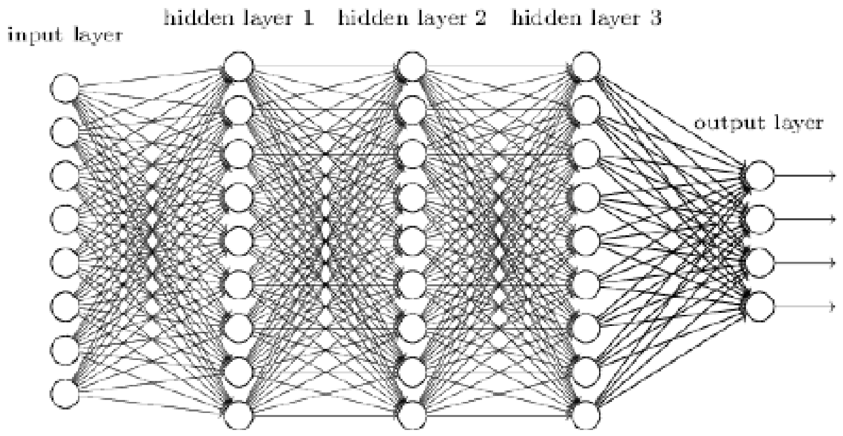


Figure 4: Example of a simple feedforward neural network [11]. The input layer takes the inputs to be mapped and the outputs of the outermost layer are those generated by the network and supposed to match the output of the desired function. The layers in between are called *hidden* layers. The flow of information is shown by the arrows and is unidirectional.

2.1.3 Training a Neural Network

The training of NNs can be supervised or unsupervised, the former describing a case in which the desired outputs are always known and used during the training process as a feedback for how well the network is performing. For networks that are supposed to categorize different inputs, this means that the input data is labeled and after each computation one can check whether the NN was able to reproduce the right label. Using labeled training data only, in the frame of this work only supervised learning will be applied.

2.1.3.1 Cost Functions

Cost functions, also referred to as *loss functions* or *error functions*, are used to quantify how well the network is performing, i.e. how close the output $\hat{Y}(w, b)$ of the network is to the ground truth Y. Being a function of the network's output, it is clear that a cost function C is also a function of the weights w and biases b of all neurons, which is essential for understanding the backpropagation algorithm in 2.1.3.3. The goal of a network's training process will be the search for a collection of weights and biases that minimizes the cost function, i.e. such that $\nabla C = 0$.

2.1.3.2 Error Surface

After looking at the cost function, it is useful to picture it as a general function consisting of a multidimensional input, its dimensionality D corresponding to the number of weights and biases existing in the network. In other words, we are looking for minima on a $(D + 1)$-dimensional surface. Its form is determined by the applied cost function, two widely used ones being the *quadratic cost*:

$$C(Y, \hat{Y}) = \frac{1}{2}(Y - \hat{Y})^2 \tag{8}$$

and *cross-entropy*:

$$C(Y, \hat{Y}) = -\sum_{i}^{N_c} Y \ln \hat{Y} \tag{9}$$

where the latter is often encountered in classification problems with N_c being the number of classes predicted, i.e. the number of output neurons. An example, that is helpful for visualization, is the error surface of a linear neuron with two input connections carrying the weights w_1 and w_2 evaluated with the quadratic cost function as shown in Fig.5. Starting with initialized weights $w_{1,start}$ and $w_{2,start}$, the training will consist of trying to bring those as close as possible to the weights in the global minimum of the error surface, creating the notion of *going down the deepest valley*.

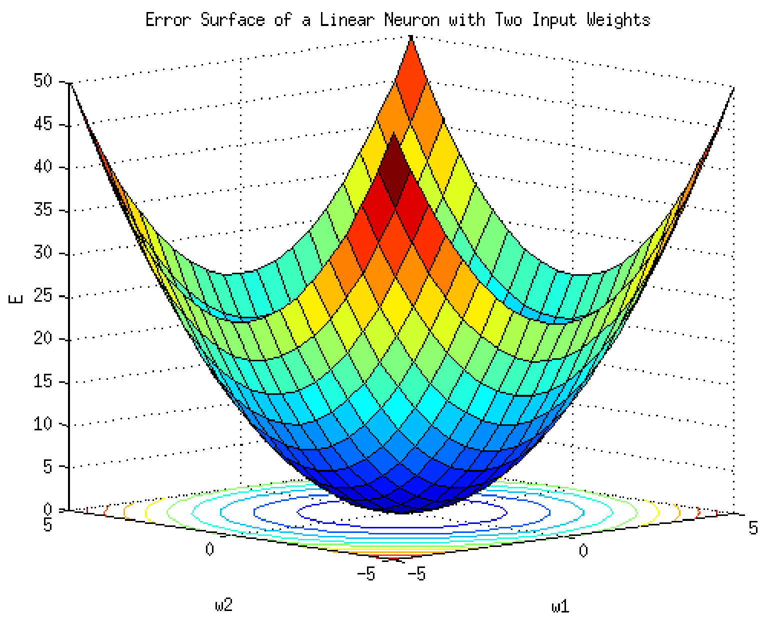


Figure 5: Error Surface of a linear neuron with two inputs, illustrated in [17].

2.1.3.3 Backpropagation

Finding the *valley* on the error surface requires an algorithm for calculating the partial derivatives of the cost function with respect to all weights and biases in the network. Loosely stated, this computation is necessary to know in which direction of the error surface one is supposed to walk to find the minimum. As demonstrated in [11], backpropagation, which can be efficiently implemented via dynamic programming, revolves around four fundamental equations and, briefly described, consists of the following steps:

1. A forward pass (see 2.1.2) is performed.

2. For each neuron in the output layer the error is calculated, containing 1) the derivative of the cost function with respect to the neuron's output activation and 2) the rate of change in the neuron's activation function for small changes in its input. The former is easily calculable because the cost function is known.

3. Next is the *backpropagation* step, in which this error vector is propagated back to the beginning of the network. Backpropagating refers to the sum of *backward passes* performed by each layer, starting at the output and finishing at the input. In the simple example network in Fig.4, this step is performed by taking the output vector and multiplying it by the transposed weight matrices layer after layer moving in the *backward* direction. One can picture this step as obtaining a measure of the error appearing at each neuron in the network's hidden layers. In a broader sense, this procedure helps analyze how a small change in a single neuron connection, i.e. a small change of a single weight value, propagates through the network and influences the resulting error and therefore the result of the global cost function.

4. The last logical step after calculating all mentioned errors is to calculate the gradients at all connections, which tell us the direction of the desired minimum from our current point on the error surface.

The exact procedure of computing the gradients (and therefore the weight adjustments) depends on the selected *optimization* algorithm. Different optimization techniques will be discussed in the following paragraph.

2.1.3.4 Optimization Approaches

Below, a few important optimization techniques of first order, i.e. ones which take into account solely the first derivative of the loss function, will be introduced.

Gradient Descent Gradient descent (GD) is arguably the most popular attempt to optimizing the learning process, serving as a foundation for more complex algorithms. It is an answer to the question of how weights and biases should be changed after we have calculated a network output from all training inputs, obtained the loss function and thereafter executed the backpropagation step discussed above. GD's simple answer to that is to rely on the *walking down the valley*-picture that was already mentioned and change weights and biases by a value that is proportional to the loss function's derivative. With the proportionality factor η, the *learning rate*, the adjustment from old (w_i, b_j) to new (w_i', b_j') weights and biases looks as follows[11]:

$$w_i \to w_i' = w_i - \eta \frac{\partial C}{\partial w_i} \; , \tag{10}$$

$$b_j \to b_j' = b_j - \eta \frac{\partial C}{\partial b_j} \; . \tag{11}$$

With the new set of weights and biases the process of forward propagation, cost calculation, backpropagation and weight/bias adjustment is carried out repeatedly, moving closer to the multidimensional local minima of the cost function. Usually, the process is terminated when a certain termination criterion is met, e.g. when the derivative with respect to the weights falls below a threshold value. The learning rate controls the quickness of change and needs to be adjusted depending on the problem. This can be done in a global manner, that means before training, or dynamically during the training process[11].

Stochastic Gradient Descent Stochastic Gradient Descent (SGD) speeds up the fairly slow Gradient Descent algorithm by calculating the gradient for a randomly chosen subset of all training inputs, referred to as a *mini-batch*. One training *epoch* then consists of taking such a random mini-batch multiple times until the model has been given each input of the training set at least once. It turns out that such a subset is enough to estimate the gradient sufficiently well, leading to successfully finding minima with less computation time. The size of such a mini-batch is, just like the learning rate, an adjustable parameter that can influence the outcome of the calculation.
It can make a decisive difference in situations where the weights and biases are initialized in such a way that after the first forward pass the model's position on the error surface is in a *local valley*, i.e. a place that is adjacent to a local minimum. Given that for untrained models weights and biases are mostly initialized randomly, this is a probable scenario and the algorithm would result in walking down the "local valley" it is in. SGD allows for bigger jumps within the high-dimensional error surface, because a small subset of inputs introduces higher noise. *Uphill steps* are now more probable and the chance of eventually arriving at the global minimum increase. Figure 6 illustrates this scenario.

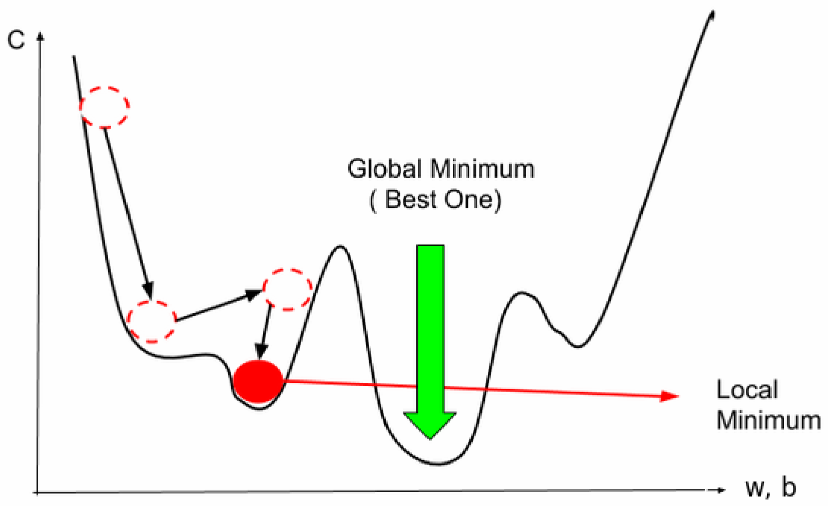


Figure 6: Finding local and global minima with GD and SGD[18]. The x-axis contains all weights and biases shrunk to one dimension, the y-axis corresponds to the cost value. The red circles schematically show the path GD would take, if the model's starting position on the error surface is the leftmost red circle. Using just a mini-batch of training inputs increases the data noise. Consequently, bigger jumps within the high-dimensional error surface are possible depending on the specific, randomly sampled subset. This increases the chance of making *uphill* steps and jumping over maxima in the process and eventually, in a figurative way, landing closer to the global minimum (green arrow). Once the algorithm is in this *global valley*, GD will *walk downwards* again, this time finding the global minimum.

Other Optimizers There are many more types of optimizers with varying complexity. A first popular change compared to GD and SGD is the introduction of a variable learning rate. The *Adaptive Gradient* (AdaGrad) [19] and *AdaDelta* [20] optimizers, which employ a decreasing learning rate, are two examples of optimizers using that technique. The *Adam* optimizer, which contains adaptive estimates of lower-order moments of the gradients [21], is a further improvement and proven to be a good choice in many cases [22],[23]. Adam is therefore the optimizer we will pick during model training. A broader overview of different optimizers and their performance can be obtained in [22] and [24].

2.1.4 Hyperparameters

The learning rate and the size of a mini-batch for GD / SGD are parameters that are not altered by the optimization algorithm but have direct consequence for the training process. Such quantities, the number and type of layers and the connections between them belong to this group as well, are called *hyperparameters*. Testing their influence on a model's behaviour is valuable and can be done in different ways. One option is to define different values for them beforehand and to train the network for all emerging hyperparameter combinations, the *grid search*. Another strategy is to assign them randomly from an underlying probability distribution. This is called *random search*.

2.1.5 Overfitting

Overfitting is a major issue that arises from training a deep learning model on a limited set of training data. It describes a situation in which the model learns to recognize certain connections between input and output that are just result of sampling noise and depend on the particular training set that is chosen (see Fig.7). Consequently, the model will perform worse on test data, even if it comes from the same distribution [25].

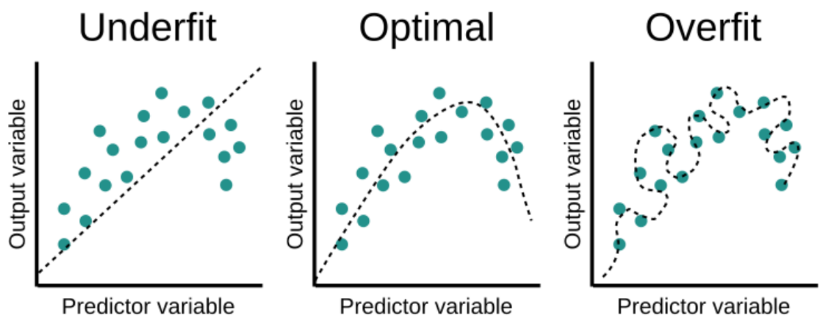


Figure 7: Underfitting and Overfitting in an estimation of a two-dimensional function[26]. If a model is underfit it is unable to reproduce the relations between input and output sufficiently well. In the case of an overfit model, the relations between input and output obtained during the training process are overly complicated and the model usually performs significantly worse on a dataset different from the one used for training.

One option to address this issue is the *dropout*-technique introduced in [25]. It consists of omitting neurons in the network with a certain probability during different training stages. Each time, a certain (and random) amount of neurons and all their incoming and outgoing connections are removed (i.e. set to zero and not updated during the backpropagation step), effectively training a different network which is smaller than the original one. What this can look like for a feedforward NN is illustrated in Fig.8.

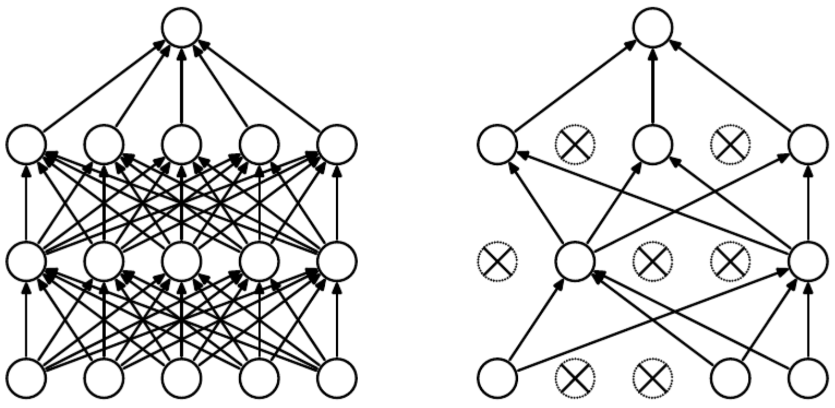


Figure 8: Example for applying dropout to a feedforward network[25]. Left: Standard feedforward neural network. Right: One possible *thinned out* network after dropout, each neuron is kept with a probability p

.

Effectively, this corresponds to training many distinct subnetworks during the training time and combining them to one whole network afterwards. The combination requires a certain weight averaging of the small, *thinned out*, networks. A computationally fast way to do this is by multiplying each weight by the probability p of the neuron it exits still being present after a *dropout*-procedure. This can be seen in Fig.9

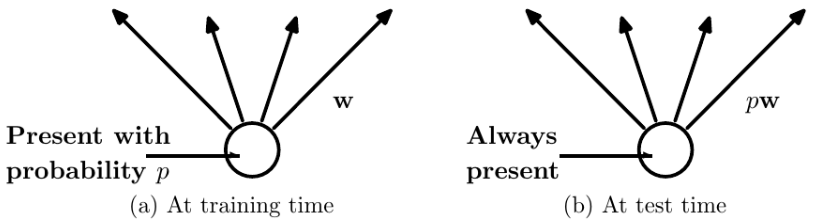


Figure 9: Weight averaging after dropout[25]. (a) At training time, dropout is applied. Every neuron and its outgoing paths are kept in the resulting subnetwork with a probability p. (b) The resulting model, used for testing and predictions, contains all neurons. Averaging is done by multiplying each weight with the dropout probability p of the neuron its path is exiting.

2.2 Convolutional Neural Networks

A special type of network which is very popular for image data processing and object recognition are CNNs (*Convolutional Neural Networks*). As emphasized in [12, Chapter 9], CNNs are a great example of biologically inspired artificial intelligence that mimick the way the primary visual cortex of a brain functions. In this section the crucial features of a typical CNN are revised.

2.2.1 The Convolution Operation

Generally speaking, convolution is a mathematical operator applied to two functions which produces a third function. It is commonly denoted with an asterisk and the continuous ($s_{cont}(t)$) and discrete ($s_{disc}(t)$) way of convolving two functions x and w into a resulting function $s(t)$ are defined in the following manner [12]:

$$s(t) = (x * w)(t) \tag{12}$$

$$s_{cont}(t) = \int_{a=-\infty}^{\infty} x(a)w(t-a)da \tag{13}$$

$$s_{disc}(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{14}$$

A possible way of picturing this operation is that the function $x(a)$ is being weighted with the help of the *weighting function* $w(a)$ over their input space. Using NN terminology, $w(a)$ is called the *kernel* and the result of this operation, $s(t)$, is referred to as the feature map. The representations of those within a CNN are shown in the next section.

2.2.2 The Convolutional Layer

Within the convolutional layer which is responsible for the convolution operation each neuron has a receptive field of given size and can be pictured as holding a weight matrix that corresponds to the kernel w with the same size. Given that CNNs are introduced with image processing in mind, it is helpful to picture the input as at least two-dimensional and the convolution can be illustrated as in Fig.10.

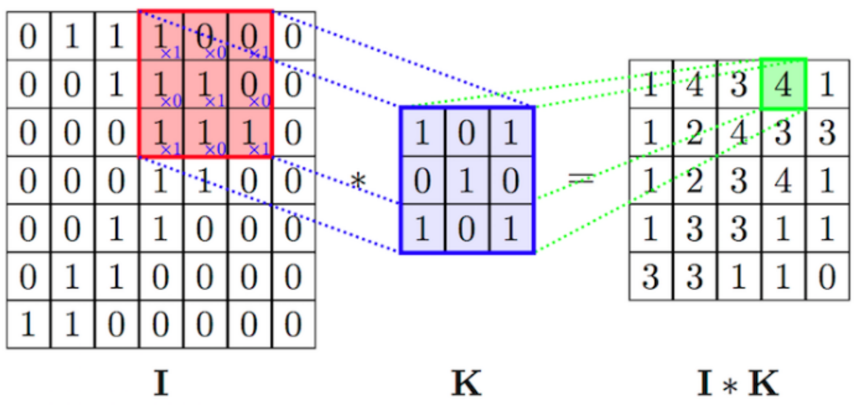


Figure 10: A single convolution operation in 2D [27]. The current receptive field of the kernel K is shown in red over the input image I. Each kernel entry is multiplied with the image value it is lying on, producing the green output *4* in this particular case. The kernel is then moved by a specified *stride length*. With an input image of size 7x7, a kernel of size 3x3 and a stride length of 1 the resulting feature map is of size 5x5. Usually, various kernels are applied using different stride lengths resulting in a stack of distinct feature maps.

The usual way of proceeding with the convolution is to slide the shown kernel over the input image, each time moving it in a certain direction with a particular *stride length*, i.e. a certain amount of pixels. Each time an activation function is applied to the current receptive field producing a single valued output. The total output size therefore depends on the input data size, the kernel size and the stride length. Additionally, the chosen *padding* also influences the output of a convolution. This term refers to how the kernel is selected to behave when reaching the ends of an input image. There are three main approaches to this problem:

- *Valid Padding*, where the kernel has to stay completely within the image
- *Same Padding*, where the kernel can exceed the image dimensions in such a way that the size of the output corresponds to the size of the input
- *Full Padding*, where the kernel is allowed to exceed the input image dimensions as long as there is at least one image pixel inside it, resulting in an output which is bigger than the input.

The latter two strategies require the kernel to perform calculations on pixels, that are not part of the image and do not exist beforehand. In those cases these *new pixels* are set to zero and the kernel performs its usual operations on the receptive field which now contains image pixels with certain values as well as the new pixels with the value 0. Figure 11 illustrates the mentioned types of padding.

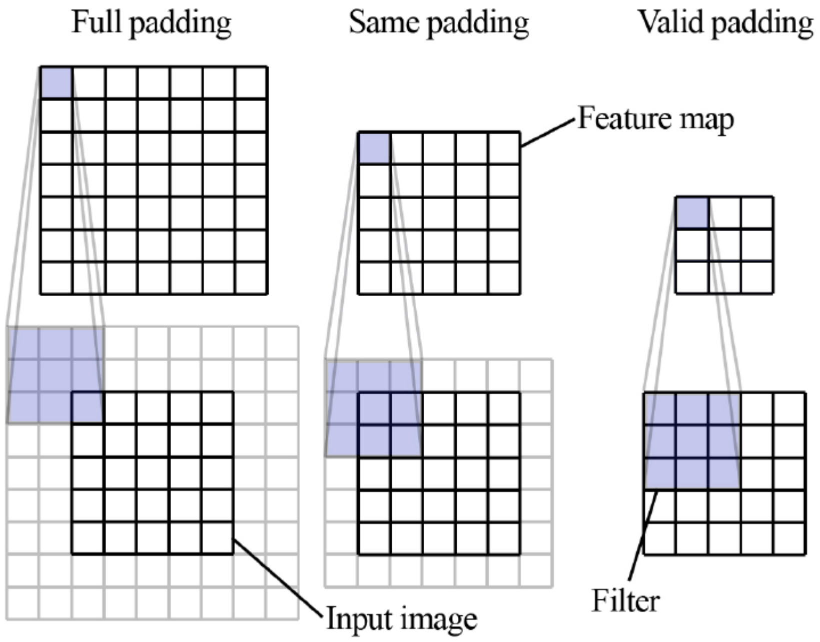


Figure 11: Different types of padding on a 2D input image of size 5×5 and a filter / kernel of size 3×3[28]. *Valid Padding* produces an output which is smaller than the input, because the kernel's receptive field shown in violet remains fully within the image. *Same Padding* ensures that input and output size are equal by enabling the kernel to have a receptive field which is partially outside of the input image. Those *new pixels* are depicted in light gray and their value is set to zero. *Full Padding* results in an output which is bigger than the input image by moving the kernel along in a way which ensures that at least one pixel of the input is within its receptive field. The newly introduced pixel values outside of the original scope are again set to zero.

Because of the fact that the kernel is substantially smaller than the image, a much smaller amount of parameters for a neuron layer have to be saved compared to a fully-connected layer. This is referred to as *sparse interaction*. It is a way of decreasing computation time but still assuring that a neuron layer affects the following layers.
Kernels are, justifiably, called *feature detectors*, since a certain type of kernel may for example help to detect edges throughout the image, while another could detect contrasts. What a CNN learns during training is to adjust its kernel's parameters to be able to extract features from the image in a way that minimizes its loss function.

The output after going over the whole input image once with a single kernel is called its *feature map*. Commonly, multiple kernels are applied on every single input creating a series of feature maps for each input. It is usual that an activation function is applied on the resulting feature maps, a very popular choice being the already introduced ReLU function (see 2.1.1.1).

2.2.3 The Pooling Layer

Pooling is a downsampling operation on the feature maps that replaces a feature map value at a single location with a value obtained from applying a function to its neighborhood. Similarly to the convolution step, you walk over the feature maps with a receptive field and apply a function to the values you are looking at at this moment. An example is *maximum pooling*, where the output of the function is the maximum of the values within the current receptive field. This is shown in Fig.12, where you can also see a comparison to another type of pooling operation, *average pooling*.

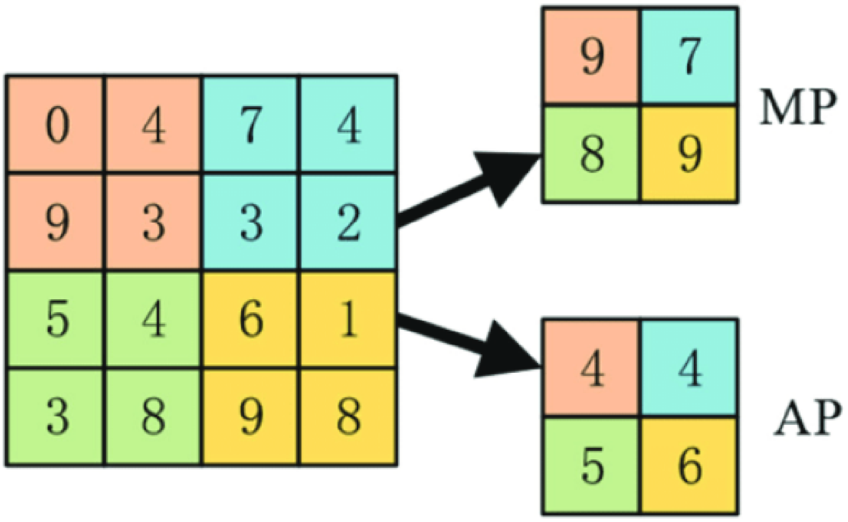


Figure 12: Comparison between Maximum Pooling and Average Pooling as depicted in [29]. Within the not necessarily uniformly shaped subregions, the algorithm picks the maximum (maximum pooling) or the average (average pooling) to take into the resulting region.

As described in [12] this makes the output approximately invariant to small translations, because you base the output of the pooling layer on a neighborhood of feature map values. This is a satisfying approach if the main goal is to detect whether a certain feature is within an image, whereas an increasing number of pooling layers gradually diminish the possibility of exactly localizing objects. In any case, the size or resolution of the output is smaller than the input size. The number of channels stays the same.

2.2.4 Flattening

Most of the times the desired overall output of a CNN will be a single value or a vector, classifying the input in a certain manner. Therefore, one or more flat, fully connected layers are added to the model after a pooling layer, connecting every neuron in the pooling layer with the next layer. One can describe this as appending a feedforward NN like the one in Fig.4 to the pooling layer, using at least one of its *flat* layers. This is often the last part of a CNN.

2.2.5 Feature Extraction

At this stage, having understood how a CNN works, it is important to point out the versatility of this architecture. It arises from the fact that a CNN produces a big number of feature maps, hence the term *feature extraction*, which afterwards can be used for many different tasks. Subsequent network structures can be added to learn, amongst other applications, classification of objects, image segmentation, and detection of objects. Due to its vital role within this work, the latter will be explained in the next chapter.

2.3 Object Detection and Recognition

The goal when constructing a model for object detection and recognition is 1) to be able to localize objects by surrounding them with a *bounding box* and 2) to assign each of them to a class, similar to the way shown in Fig.13.

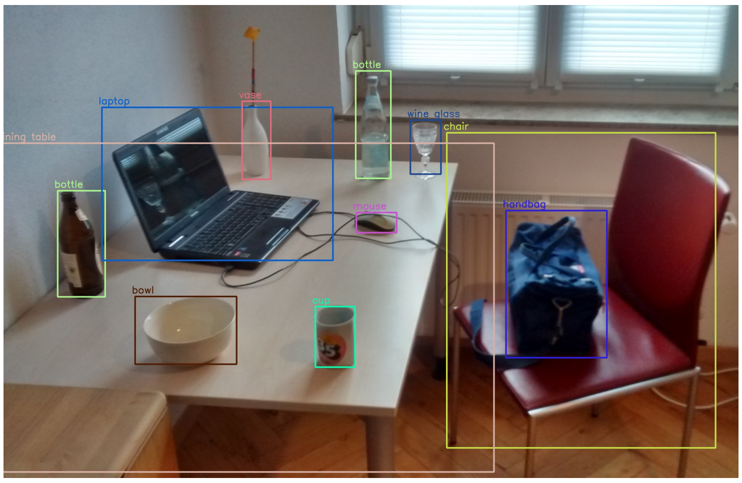


Figure 13: Example result of a two-dimensional object detection model [30]. The goal of an object detection model is to recognize the type and location of objects within an input image. The network learns to assign a bounding box and a class prediction to each object.

Classes can be characterized by integers ranging from 1 to N, with N being the total number of classes. For example: if we want to construct a network which learns to detect and differentiate between cats and dogs, we have N = 2 classes. Therefore, our classification output is a two-dimensional vector $\vec{c}$ (generally N-dimensional), assigning each class a probability to be the right one for the object in the current bounding box. On the other hand, bounding boxes (in three dimensions) are characterized by their center coordinates $x_{c,i}, y_{c,i}, z_{c,i}$, where i is the bounding box index, and their height, width, length and orientation $h_i = dx_i, w_i = dy_i, l_i = dz_i, \Theta$. We will refer to this set of network outputs as $\{\vec{c}, x_{c,i}, y_{c,i}, z_{c,i}, dx_i = h_i, dy_i = w_i, dz_i = l_i, \Theta_i\} = M_i$.

For proper training, one also needs to define a loss function L over the classification and bounding box estimations. Those will be described in section 2.3.1. This general description of the most significant detection network operations is illustrated in Fig.14.

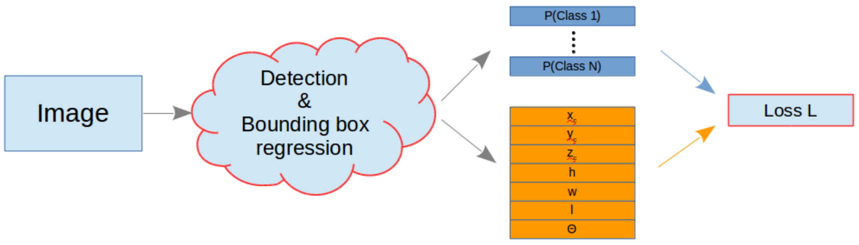


Figure 14: General information flow in an object detection network. A set of class predictions and coordinates as shown is created for a big number of bounding boxes.

Object detection networks have a typical structure that is displayed in Fig.15. Follow-

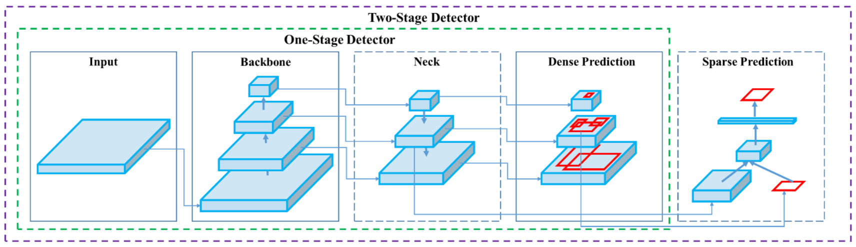


Figure 15: General structure of an object detector[31].

ing the input image, the first component of a detection network is its *backbone*, which is responsible for extracting feature maps from an input image[32]. Example backbone-structures are VGG[33] and the U-Net structure presented in 2.3.5. The *neck* refers to layers between the *backbone* and *head* which commonly collect and work on feature maps from different stages of the backbone[31]. A prominent example are *Feature Pyramid Networks* (see 2.3.6). Lastly, the *head* produces the network's final predictions.

Heads of object detection models can be grouped into *one-stage* and *two-stage* models[34]. In a one-stage model, as depicted in Fig.16, classification and localization happen in a single stage, resulting in a faster but usually less accurate prediction. Two-stage models consist of a region proposal part which outputs, as the name suggests, propositions of regions which contain objects. Those propositions are then classified. This procedure is usually more precise but comes with longer computation duration. In the following paragraphs, after shedding light on recurrently used loss functions for object detection, one-stage and two-stage models are discussed in more detail.

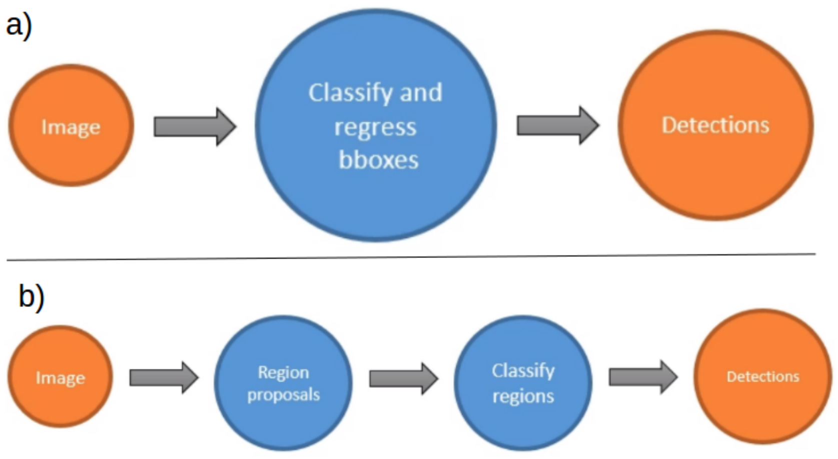


Figure 16: Different working principles of a) one-stage object detectors and b) two-stage object detectors [34].

2.3.1 Loss Functions

Given the two tasks set in this work, localization and classification, we have to make sure that after each forward propagation the network weights are adjusted in a way that improves both types of prediction. As introduced in 2.1.3.1, this is carried out via a proper loss function based on the model's N+7 outputs $\{\vec{c}, x_{c,i}, y_{c,i}, z_{c,i}, h_i, w_i, l_i, \Theta_i\} = M_i$. It is clear that we need to differentiate between cost functions for localization and classification and that both should be considered in the overall loss L. In general, L looks like the following:

$$L = \frac{1}{N_b}(L_{cls} + \alpha \cdot L_{loc}) \tag{15}$$

where L_{cls} measures the classification precision, L_{loc} the localization precision,N_b a normalization factor depending on the exact problem and α is a hyperparameter ensuring that both loss types contribute in a certain proportion to the overall loss. This section is devoted to showing popular examples of both loss types.

2.3.1.1 Classification Loss

The most popular classification loss is the cross-entropy, already presented in 9. Here, we will stress its form for a problem with two classes, the so-called *binary cross-entropy*. Its definition is:

$$L_{bc} = -\frac{1}{N}\sum_{i=1}^{N} y_i \ln\left(p(y_i)\right) + (1 - y_i) \cdot \ln\left(1 - p(y_i)\right) \tag{16}$$

Here, the index i corresponds to an input, y is the binary class label (in the context of nodule detection: 1 $\widehat{=}$ nodule and 0 $\widehat{=}$ not a nodule) and p is the predicted probability, i.e. the number our model outputs.

2.3.1.2 Localization Loss

To maintain clarity, the seven bounding box parameters for three dimensions explained in 2.3 and 2.3.1 are from now on denoted by a vector. What is needed now is a cost on such a bounding box vector $\vec{b}$, output of the neural network, in order to compare it against the "right" box the model is supposed to predict, the *ground truth box* $\vec{g}$. A popular option is the *smooth L1 loss* illustrated in Fig.17. It has slightly varying definitions

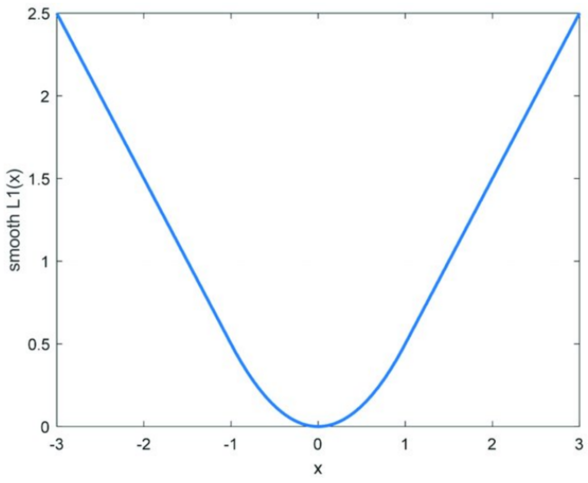


Figure 17: The smooth L1 loss in one dimension[35].

and here we will work with the version by Huber defined in [36] as:

$$L_{1,smooth}(\vec{b},\vec{g}) = \begin{cases} \frac{1}{2}(\vec{b}-\vec{g})^2 & \text{if } \left\|\vec{b}-\vec{g}\right\| \leq \delta \\ \delta\left\|\vec{b}-\vec{g}\right\| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{17}$$

In order to keep inputs and outputs for the loss dimensionless and parametrized, the vectors are not just fed into the loss like in eq.17 but are normalized. This can be seen later when specific model architectures are discussed.

After mentioning the difference between a ground truth box $\vec{g}$ and a predicted box $\vec{b}$, it is convenient to present a measure of how much two boxes overlap, which is vital during training and evaluation, the *Intersection over Union* (IoU). For two boxes A and B it is defined by

$$IoU = \frac{A \cap B}{A \cup B} \tag{18}$$

and takes a value in [0, 1].

2.3.2 Single-Stage Detection Models

The first notable type of single-stage (one-stage) detection model is the *Single Shot Multibox Detector*. The SSD is a feedforward CNN that produces bounding boxes, supposed to enclose objects of the different classes the network is designed to differentiate from[9].It assigns probability values to each of them, indicating how likely it is that a certain box encloses object instances of the different classes. The base SSD network is shown, for simplicity in two dimensions, in Fig.18.

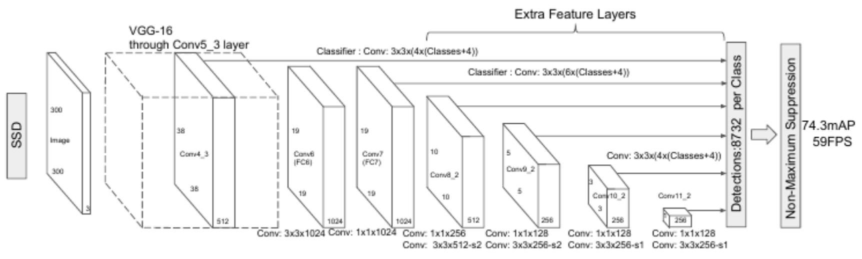


Figure 18: SSD network base for a 300x300 input [9]. The multiple feature layers of different size allow for an analysis of the input image on multiple scales.

It starts with the already shown convolution operation followed by the creation of multiple feature maps of various size. The variety in size corresponds to the possibility of extracting features on different scales. This is simply another way of saying that the resolution of a feature map is proportional to its dimensionality. Having feature maps of various size therefore means that the input is analyzed on multiple scales. The advantages of this system become clear after introducing the concept of *default / anchor boxes*.

Default Boxes To be able to detect multiple objects of different shapes, in the SSD approach one defines a number of m different *default boxes* for each pixel. This is a "human" step done before training. The form of the default boxes ought to be suitable to the problem.In a classification problem with one class typically fitting in a rectangular bounding box and another class typically described by a square box, both of those forms should be considered in the set of default boxes. We can now see how the components mentioned by now work together: With the definition of a single default box size, the model can extract features with vastly different resolutions by applying the box to all feature maps of varying sizes. It is due to this fact that SSDs usually require a smaller number of predefined anchor boxes compared to a Faster-RCNN (see 2.3.3).

Ground Truth Boxes In order to be able to train a network one naturally needs the "solutions" to the bounding box search, i.e. the *ground truth boxes* mentioned in 2.3.1.2, which in 3D are seven-dimensional vectors with center-coordinates, sizes and orientation of a bounding box correctly surrounding an object of a given class and which are our prediction goals for the model. In the case of SSD, ground truth and prediction boxes are given with *relative coordinates* to the default boxes (see eq.21). Each of these boxes can belong to N + 1 classes. N is, as it has been by now, the number of classes our initial problem has. The additional class arises from the pixel-wise processing in which we also need to introduce the additional class of the *background* if none of the default boxes centered at a pixel contains an object of any class.

Matching Of course it does not make sense to consider all default boxes during training. Therefore, the first step is a *matching* between ground truth boxes and default boxes, which means that during training we leave only the default boxes that have a minimal IoU with the ground truth boxes. An example of a 2D case with ground truth boxes of two different classes (here: cat and dog), three different default boxes per pixel and an already performed ground truth - default matching is illustrated in Fig.19.

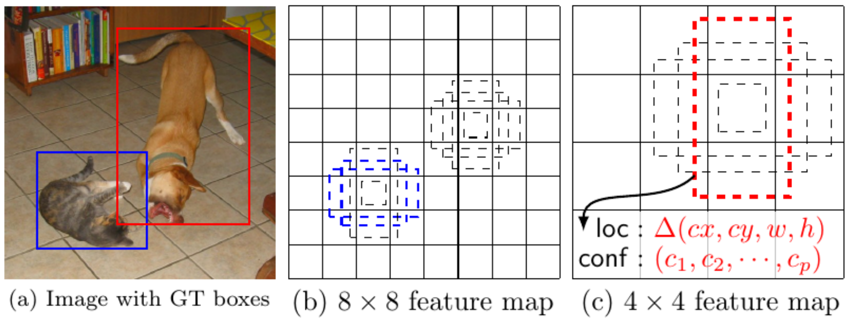

loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

Figure 19: Ground truth and default box definition in a SSD[9]. Each pixel gets a set of m default boxes that are matched with the ground truth boxes through an IoU calculation. Having default boxes on different feature map scales allows the detection of objects of different size.

With the help of the same picture, one can quantitatively see how a single default box operates on multiple scales. As depicted, the default boxes roughly fit into a 3x3 square. Applying a 8x8 feature map on the image makes them quite small compared to the objects in the image. While this is enough to have a sufficiently big IoU with the cat's ground truth box, the overlap with the dog's box is too small. This changes on applying the 4x4 feature map which now allows a matching of the dog's ground truth box to at least one of the default boxes.
A second matching also needs to be done, in which the predicted boxes (network output) are matched with the ground truth boxes, requiring a minimal IoU for a network prediction to be considered consistent with the ground truth.

Loss computation The training objective of this model is to minimize its loss as given in eq.15 with N_b being the number of matched default boxes in this particular case. It is common to pick the already described smooth L1 loss (see 17) for localization and the cross-entropy loss (see 2.3.1.1) for classification. With $x_{ij}^p = 0, 1$ indicating whether the i-th default box matches the j-th ground truth box of a class p, localization and classification losses become:

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N_b} \sum_{m \in M_i} x_{ij}^p L_{1,smooth}(l_i^m - \hat{g}_j^m) \tag{19}$$

$$L_{cls} = -\sum_{i \in Pos}^{N_b} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^{p=0}) \tag{20}$$

A positive example here (first sum) is a predicted box with a minimal IoU with a ground truth box that has been matched to at least one default box. l corresponds to a predicted box, Θ is an angle describing the bounding box orientation for a 3D problem, g to a ground truth box and $\hat{g}$ corresponds to a parametrized, unitless ground truth box regressed to offsets from the coordinates of a matching default box
$\vec{d_i} = \{d_i^{x_c}, d_i^{y_c}, d_i^{z_c}, d_i^h, d_i^w, d_i^l, d_i^\Theta\}$ as follows [37]:

$$\begin{aligned} \hat{g}_j^{x_c} &= \frac{\hat{g}_j^{x_c} - d_i^{x_c}}{d_i^h} & \hat{g}_j^h &= \ln\left(\frac{g_j^h}{d_i^h}\right) \\ \hat{g}_j^{y_c} &= \frac{\hat{g}_j^{y_c} - d_i^{y_c}}{d_i^w} & \hat{g}_j^w &= \ln\left(\frac{g_j^w}{d_i^w}\right) \\ \hat{g}_j^{z_c} &= \frac{\hat{g}_j^{z_c} - d_i^{z_c}}{d_i^l} & \hat{g}_j^l &= \ln\left(\frac{g_j^l}{d_i^l}\right) \\ \hat{g}_j^\Theta &= g_j^\Theta \end{aligned} \tag{21}$$

In the classification loss $\hat{c}_i^p$ is just the softmax output activation result, seen by replacing $x \to c_i^p$ in eq.6. The second sum, counting over examples without any match ($p = 0$), ensures that the network learns to identify the background.

You Only Look Once (YOLO) The *YOLO* architecture is another noteworthy type of one-stage detector. After the initial 2D versions[38],[39], in recent years YOLO models for 3D object detection like *Complex YOLO* and *YOLOv4* have been proposed[40],[31]. However, YOLO networks are point cloud based detectors. In contrast, this work aims to make use of models which work with volumetric data.

2.3.3 Two-Stage Detection Models

Among two-stage methods the RCNN-series is arguably the most widely used approach. In the subsequent paragraphs we will delve deeper into how these models work.

Region-based Convolutional Neural Networks - RCNN The RCNN-model is divided into three parts: a region-proposal algorithm (stage 1), a CNN extracting features from the region candidates and a *Support Vector Machine* (SVM) for classification. The last two steps comprise the network's second stage and are applied to every region proposal. From the features obtained by the CNN, the SVM learns to classify what is within a given input region. Extracting region candidates is done with a selective search algorithm [41] which is not able to learn. Here lies the biggest weakness of this approach, because it consumes time but is necessary on every new input image. An outline of the way a RCNN works is given in Fig.20.

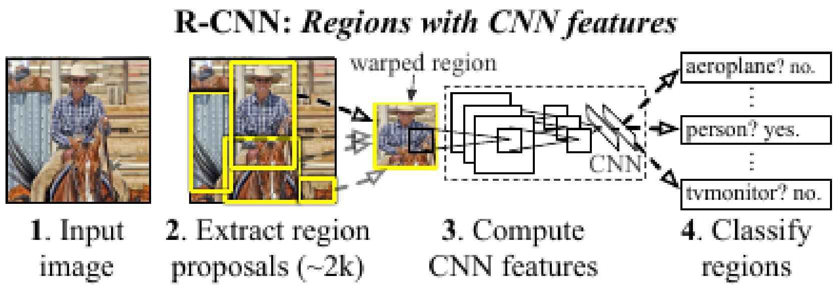


Figure 20: Working principle of a RCNN as shown in [42].

Fast-RCNN The Fast-RCNN introduces *ROI Pooling* into the original network. It starts again with selective search to obtain ROIs. Unlike in the previous setup, the following CNN takes as inputs the whole original image and the coordinates of the regions proposed by the selective search. The ROI-Pooling is now responsible for creating feature maps of fixed size, usually via maximum pooling (see 2.2.3). An example of this step is reproduced in 21. All feature maps produced in this step are fed into fully connected layers, dedicated to classification and localization. Softmax and bounding box regression (already seen for the SSD model) are then used to acquire a final prediction. Figure 22 outlines this course of action.

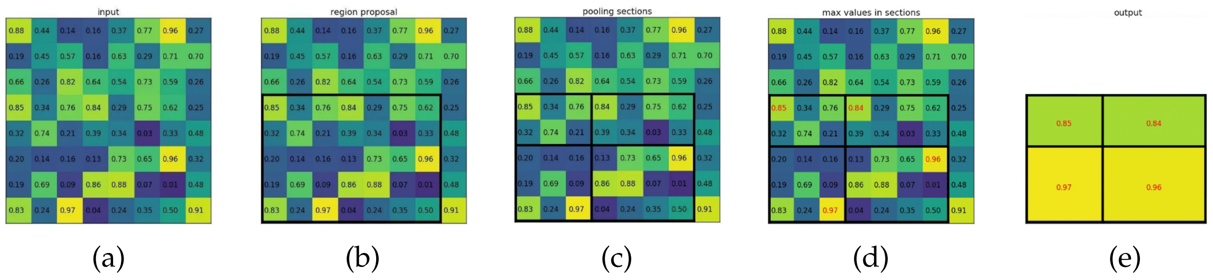


Figure 21: Basic example of an ROI pooling procedure as delineated in [43]. Each ROI(b) within an input feature map(a) is divided into pooling sections(c). Within each of those, maximum pooling is applied(d) to produce a fixed-size feature map(e), in this case 2x2. This is done for all feature maps and all ROIs.

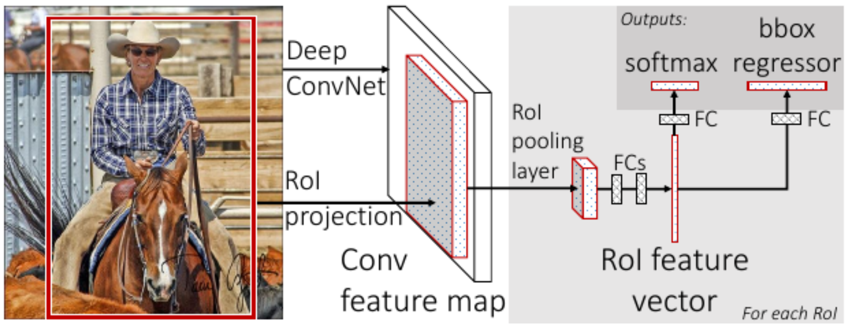


Figure 22: Outline of a Fast-RCNN from [44]. The ROIs still come from a selective search algorithm but are now a second input to the following CNN. Unlike for a RCNN model, the CNN works on the whole input image. Calculated feature maps, together with the region candidates, undergo ROI pooling, before the resulting maps enter fully connected layers. The results of those are the basis for the final softmax application / bounding box regression.

Faster-RCNN In the Faster-RCNN network the selective search is being replaced by a *Region Proposal Network (RPN)* in form of an additional CNN. The input image is directly fed into a CNN to produce feature maps. Once again those feature maps are passed to a ROI-pooling layer but at the same time they serve as an input for a RPN whose final region candidates are passed to the ROI-pooling module. It is worth to point out, as sketched in Fig.23, that the RPN and the rest of the network share the same CNN. The RPN's take on being able to detect different objects at different scales differs from the methods used by now. Within the very last feature map, a small set of k *anchor boxes* are defined. They are moved along the map in a sliding-window fashion, creating multiple region proposals for each sliding-window location. An illustrative example of this for a 2D model is given in Fig.24. Now the pivotal anchor boxes have to be filtered out. This is where the ground truth boxes come into play. The IoU of the anchor boxes with the ground truth boxes decides upon the category an anchor belongs to: If it is bigger than an upper threshold, the anchor is considered positive, otherwise negative. The RPN has a resulting loss function of the already known form:

$$L_{RPN}(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \frac{1}{N_{loc}}\lambda\sum_i p_i^* L_{reg}(t_i,t_i^*) \tag{22}$$

Here, i is the anchor index, p_i the probability of it being an object and p_i^* the ground truth label, 1 for a positive and 0 for a negative anchor. t_i and t_i^* are the predicted and ground

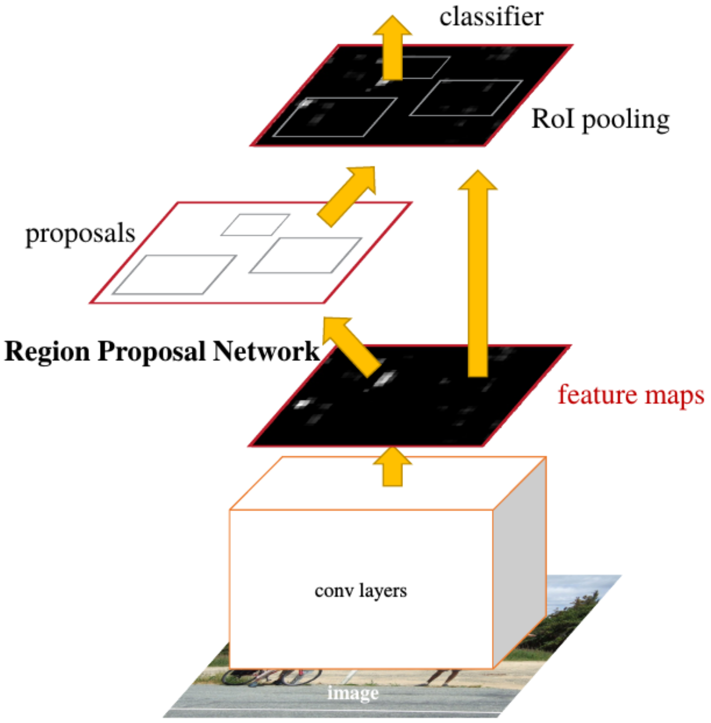


Figure 23: Structure of the Faster-RCNN model as shown in [45]. It has two improvements compared to its predecessor: instead of the selective search there is an RPN, which is responsible for creating region candidates and, unlike the selective search, able to learn. Moreover, this RPN shares the same CNN as the rest of the network. The feature maps produced by the CNN, which takes a whole image as input, are passed on to the rest of the network together with the region proposals)as it is in the case of Fast-RCNN).

truth boxes' parametrized coordinates vectors. The parametrization is performed with regards to the corresponding anchor box coordinates and works exactly like in the SSD model in 21 by replacing $\vec{d_i} \rightarrow \vec{a}_i$ where $\vec{a}_i$ is the 7D-definition of a matching anchor box. N_{cls} and N_{loc} are optional normalization factors and λ a hyperparameter used for balancing both losses. The classification loss is just a logarithmic loss over the two classes of 1) anchor contains object and 2) anchor does not contain object. The regression loss, once again, is given by smooth L1. Although it looks similar, we have to differentiate that from the loss of the overall Faster-RCNN applied on the network's classification

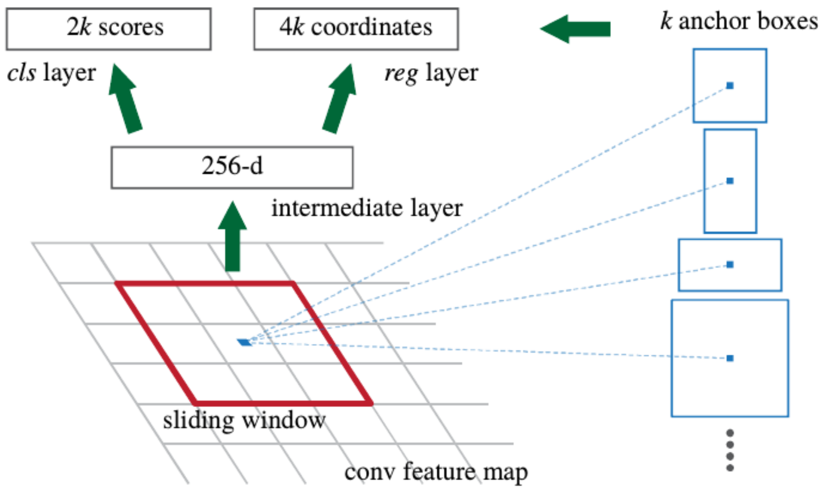


Figure 24: Anchor box definition in the Faster-RCNN model, as seen in [45]. A small number of them are defined on the last feature map and moved along it in a sliding-window manner. For each sliding-window position multiple region proposals are created with this procedure. The important ones are selected via IoU calculation with the ground truth boxes.

and localization results after the softmax / bounding box regression layers. It looks like this[34]:

$$L_{Faster}(\{p\},\{t\}) = L_{cls}(p,p^*) + \lambda \cdot (p_i, p_i^* + \frac{1}{N_{loc}}\lambda[p^* \geq 1]L_{reg}(t,t^*) \tag{23}$$

While the regression loss is again the smooth L1 loss, there are substantial differences to the RPN's loss function:

- p and t denote the class probability vector / box coordinates of the network's prediction

The normalization factors and the balancing parameter λ, as usually, depend on the specific implementation.

2.3.4 Confidence Thresholding and Non-Maximum Suppression

The final output of an object detection model is typically acquired after applying *confidence thresholding* and *non-maximum suppression* to the output vectors of the detector head. Confidence thresholding discards all model predictions with a classification confidence which is smaller than the threshold value t_{conf}. After that, there still may be multiple predictions for the same object. This is addressed by the non-maximum suppression, which ensures that to each detected object only the prediction with the highest classification confidence is assigned[46, 47, 34]. The algorithm sorts all prediction boxes with respect to their confidence values in decreasing order and, starting at the top, removes all boxes that have an IoU of $t_{\text{iou, nms}}$ or higher with the currently viewed prediction. An example is illustrated in Fig.25. A last possible strategy is *top-k filtering*, where only the k predictions with the highest confidence constitute the final output.

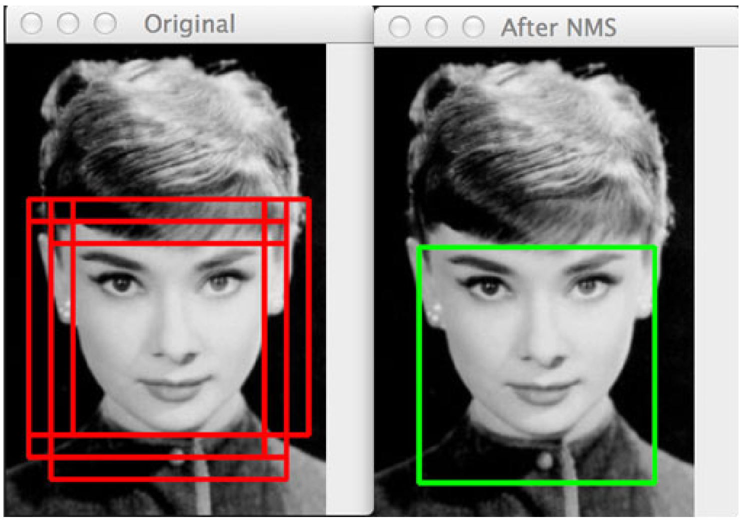


Figure 25: Non-Maximum Suppression in a model for facial recognition[47]. After confidence thresholding there are still multiple boxes detecting the same object (shown on the left in red). Non-maximum suppression sorts the remaining boxes by their confidence value. Starting at the highest one, the IoU with all remaining boxes is computed. All boxes with an IoU of $t_{\text{iou, nms}}$ or higher are then removed, because they are interpreted as boxes that have detected the same object but with a lower confidence. Ultimately, for each detected object in the image the one box with highest confidence is left (green box on the right).

2.3.5 U-Net

The *U-Net* architecture, depicted in Fig.26, is primarily used for image segmentation [48] but also in detection frameworks, whether as a base for the detector part [49] or as a backbone for the FP reduction part of the model [50]. Its outstanding feature [48] is the series of *up-convolutions* applied to the created feature maps after multiple convolutions.

With the help of those, the output is not necessarily a vector, like it usually is after creating feature maps and a subsequent pooling operation, but can be an image. A two-dimensional example architecture is shown in Fig.26.

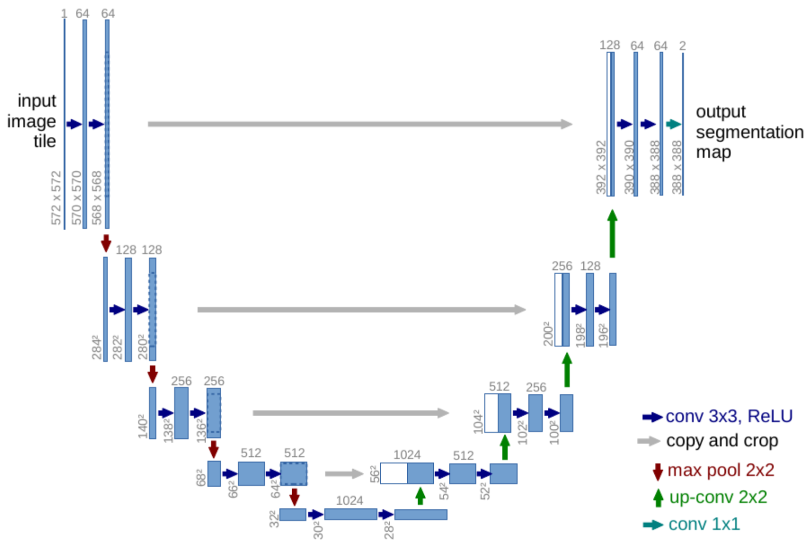


Figure 26: Example U-Net architecture in 2D for input images of size 572x572 and a lowest resolution of 32x32 pixels [48]. Instead of pooling and creating an output vector after obtaining feature maps, so-called *up-convolutions* are applied. Blue boxes are multi-channel feature maps, white boxes represent copied feature maps.

2.3.6 Feature Pyramid Networks (FPN)

Feature Pyramid Networks [51] improve the detection of objects on different scales. Their first part, the *bottom-up pathway* [52], is the typical application of convolutions resulting in feature maps on different scales. Instead of predicting directly from those, in the *top-down pathway* the highest resolution feature map is gradually upsampled and merged with a feature map of corresponding size from the bottom-up pathway. The resulting maps are the base for the model's predictions. This procedure is depicted in Fig.27.

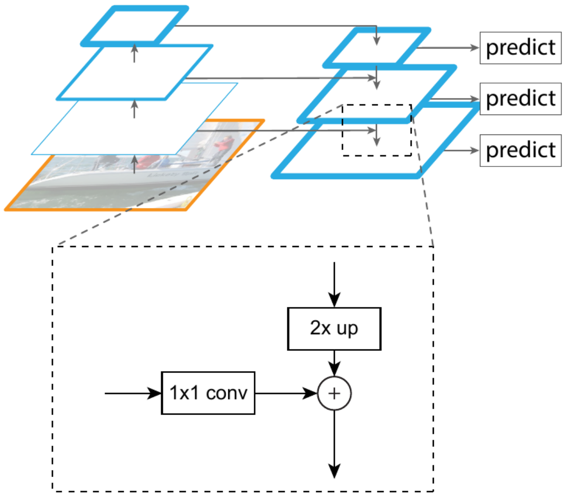


Figure 27: Working principle of Feature Pyramid Networks (from [52], originally illustrated in [51]). Multi-scale feature maps from the *bottom-up pathway* are merged with upsampled feature maps from the *top-down pathway*. The resulting maps are used for prediction.

2.3.7 Object Detection in Two and Three Dimensions

After analyzing the structure of object detection models, we have to consider the role of the dimensionality of the data that the network is supposed to work with. The architectures shown so far are widely used in 2D form for object detection, i.e. on 2D images (or single slices of 3D images) and producing 2D feature maps and proposals, which is why there exist many high-scoring approaches of this type for the task of lung nodule detection[6, 5]. Their success arises partially from the fact that they can rely on pretrained models originally used for other detection tasks. The available knowledge base regarding 3D models is more sparse, especially if we want to use models which work on volumetric data rather than ones based on point clouds, as is the case for Complex YOLO[40, 31].

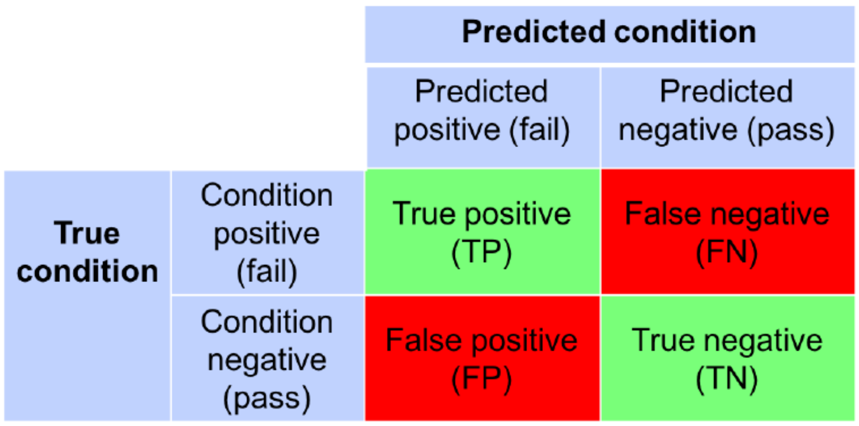
Consequently, 3D volumetric object detectors for lung nodule detection, apart from being based on the models introduced above, often require the assembly of model structures for which no pretrained weights exist. In 3 there is a variety of such examples.

2.4 Evaluation metrics

In the upcoming sections, popular performance metrics for classification are introduced [53]. They will be shown under the premise of a binary classifier but are extendable to multiclass classification.

2.4.1 Confusion Matrix

The confusion matrix is a well-arranged presentation of possible classification results from which metrics can be derived. The binary case, which is important in this work, is shown in Fig.28. In the case of nodule classification, *True Positives* are suspects that are



		Predicted condition	
		Predicted positive (fail)	Predicted negative (pass)
True condition	Condition positive (fail)	True positive (TP)	False negative (FN)
	Condition negative (pass)	False positive (FP)	True negative (TN)

Figure 28: Confusion matrix for a binary classifier as depicted in [54].

correctly predicted to be of class *nodule* or *non-nodule*, *True Negatives* are suspects which are correctly classified as not belonging to any of the two classes, *False Positives* are candidates the network wrongly classifies as nodules or non-nodules and *False Negatives* are candidates which do belong to the nodule or non-nodule class but are dismissed by the network. While the concept of TP, FP, TN, FN is crucial for all upcoming metrics, the confusion matrix itself is not particularly useful in object detection tasks due to the fact that all samples that do not contain an object of a known class are technically a TN. Therefore only TP, FP and FN are important for our metrics.

2.4.2 Precision, Sensitivity/Recall, Specificity

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{24}$$

$$\text{Sensitivity} = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{25}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{26}$$

A first thing to notice is that precision and recall give a better view over how the classifier handles positive examples while specificity describes better how well negative examples are managed. Moreover, it is clear why in medical applications the recall is of primary concern: in those situations, it is desired to have the best possible performance over the *Real Positive* values, which is exactly $TP + FN$. Missing true positives and having false negatives are the main aspects to avoid and correspond with lowering the recall. Consequently, the recall is also used in the next metric [53].

2.4.3 Receiver Operating Characteristic

Before looking at the so-called *ROC-curve*, it is helpful to define the *True Positive Rate* (tpr, the fraction of positives correctly predicted), the *False Positive Rate* (fpr, the fraction of negatives incorrectly predicted) and the *Accuracy a* as follows:

$$\text{tpr} = \text{Sensitivity} \tag{27}$$

$$\text{fpr} = \frac{\text{FP}}{\text{TP} + \text{FN}} = 1 - \text{Specificity} \tag{28}$$

$$\text{a} = \text{pos} \cdot \text{tpr} + \text{neg} \cdot (1 - \text{fpr}) \tag{29}$$

where pos = TP + FN are all positive cases and neg = FP + TN all negative cases. It is now suitable to introduce the 2D ROC-space, which is a plot of the *fpr* as a function of the *tpr* and shown in Fig.29a. It is obvious that the upper left region is desirably where we want a classifier to be, whereas the lower right part describes particularly bad performance. A second important insight, which is that accuracy isocurves, i.e. curves along which the accuracy stays constant, are straight lines in the 2D ROC-space, is shown in Fig.29b. A third thing to realize is that the accuracy equals the true positive rate along the decreasing diagonal. There are two ways which help to look at the ROC-space and how it can be useful:

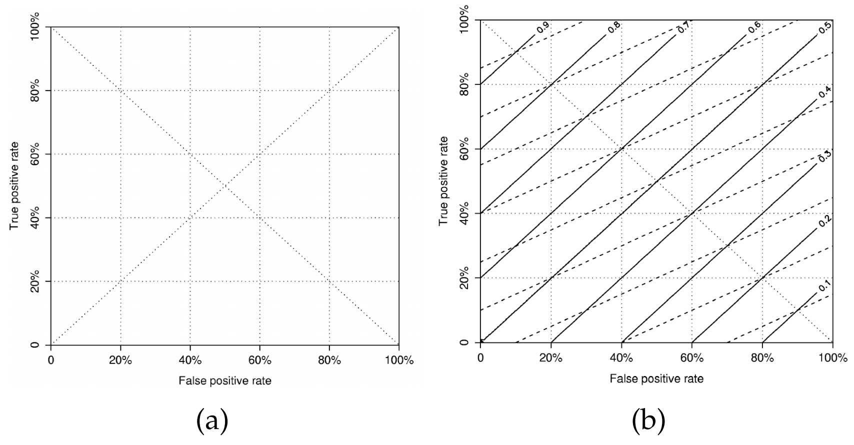


Figure 29: In (a) the ROC-space [55] is shown in pure form. The increasing diagonal corresponds to random model performance and the decreasing diagonal depicts the line along which model accuracy equals the TP rate, which is shown on the y-axis. In (b), accuracy-isocurves, which correspond to straight lines in ROC-space, are shown for negative-to-positive ratios of 1 (continuous line) and $\frac{1}{2}$ (dashed line).

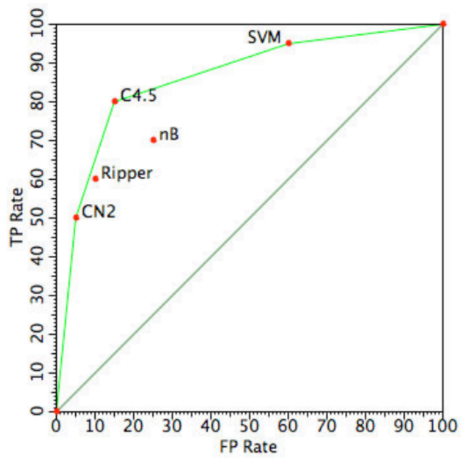


Figure 30: Different classifiers in ROC space as shown in [56]. This indicates their performance for a certain (unknown) threshold value of their corresponding output activation. Drawing the convex hull (green line) helps to later on compare the quality of the models' results.

ROC-space for classifier comparison Object classification is done by carefully selecting a threshold of our output activation (which is commonly softmax or sigmoid) for which examples above are considered positive (here: it is a nodule) and below are considered negative (here: it is not a nodule). With that, we can produce a confusion matrix for different classifiers at a threshold and plot this in ROC-space. An example is given in Fig.30. In such a plot the best classifiers lie along the *convex hull* (green line in the plot). The single best classifier now depends on the given class distribution. This is because, as it is already shown in Fig.29b, the accuracy isocurves have a different slope in ROC-space depending on the proportion of positive to negative examples np $= \frac{\text{neg}}{\text{pos}}$. The slope of the accuracy-isocurve in this representation corresponds to the given fraction, i.e. it becomes more flat with more positive examples compared to negative ones in the dataset. In Fig.29b we have already seen that the accuracy-isocurves correspond to a higher accuracy value the farther we move up and left in ROC-space. Therefore, from looking at the classifiers in ROC-space (30) we can draw conclusions about the best classifiers for different *pn*-values as shown in Fig.31.

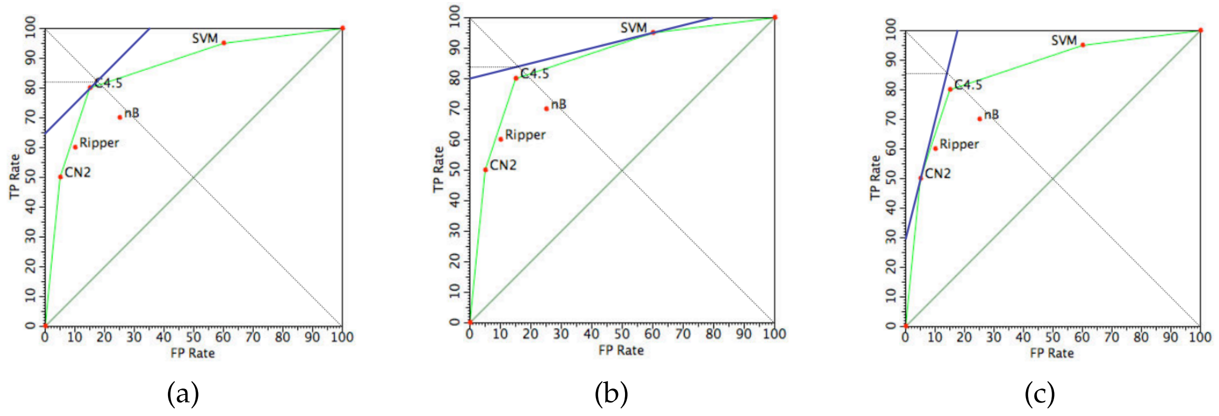
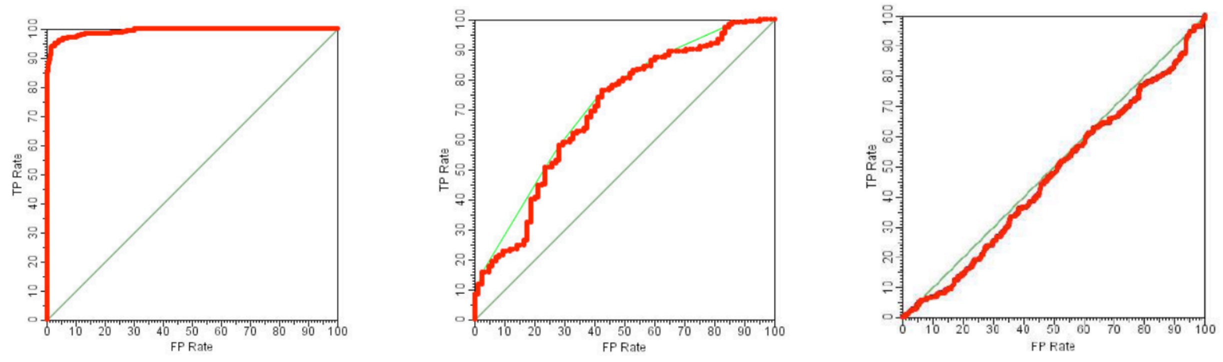


Figure 31: How to choose the best classifier in ROC-space[56] depending on the ratio of positive to negative examples $np = \frac{neg}{pos}$. This ratio corresponds to the slope of each blue accuracy-isocurve. The best performing model is always the one whose y-value (TP rate or recall) at the intersection of the blue line with the decreasing diagonal is maximal. The precise value then corresponds to the prediction's accuracy (see 29) as already stated in 2.4.3. In (a) the ratio is $np = 1$, therefore the blue accuracy curve has a slope of 1 and the best model is *C4.5* with an accuracy of around 82%. In (b), $np = \frac{1}{4}$ and the best performance is given by the *SVM* with around 84% accuracy. In (c), $np = 4$ and the *CN2*-model with around 86% accuracy is the best choice.

ROC-curves for a single classifier As already established, the chosen threshold of the output activation influences the accuracy-isocurves in 2D ROC-space. This is because it has direct influence on a classifier's confusion matrix. For example: choosing a very low threshold, in the limit thr $\to 0$, all candidates are considered positive. That makes the recall equal to one, because there are no false negatives, but also maximizes the false positives to one, because all candidates that are not truly nodules are considered to be ones. This is the upper right point in ROC-space along the increasing diagonal and does not hold any valuable information. What could now be done (there are better ways but this is good as an example) is to vary the threshold for a single classifier and compute for each threshold value its position in the plot. This defines the *ROC-curve*. Figure 32 shows different variations of ROC-curves using different parameters.



(a) (b) (c)

Figure 32: Examples of ROC-curves for a ML model as shown in [56].(a) shows a very well performing model with nearly perfect class separation. (b) depicts a poorly performing model with concavities in the ROC-curve. (c) shows a model that is not better than a random choice of classification.

Optimally, the ROC-curve has the biggest possible area underneath and therefore this area is another quality measure called the *AUC* (Area Under Curve). This implies that global convexity is a desirable feature of a model's curve. Additionally, a curve corresponding to the ascending diagonal (tpr = fpr) means that the model performs as well as a random classification. This is why locally linear ROC-curves like the one in 32 are usually considered suboptimal.

2.4.4 Free-response Receiver Operating Characteristics (FROC)

There is another meaningful performance measure for machine learning tasks with medical application[53], and logically also for lung nodule detection frameworks[6]: the *FROC-curve*[57]. It is obtained by replacing the false positive rate on the x-axis of the ROC-curve by the average number of false positives per image. As pointed out in [58], this is even more relevant to clinical practice, given that it intrinsically puts a higher value on localizing the objects we are classifying. The pivotal measure is how high the sensitivity over each FP-average is and for that reason the area under the curve states how well a model performs for a given output activation threshold over a range of average FP-numbers per image. An example is shown in Fig.33.

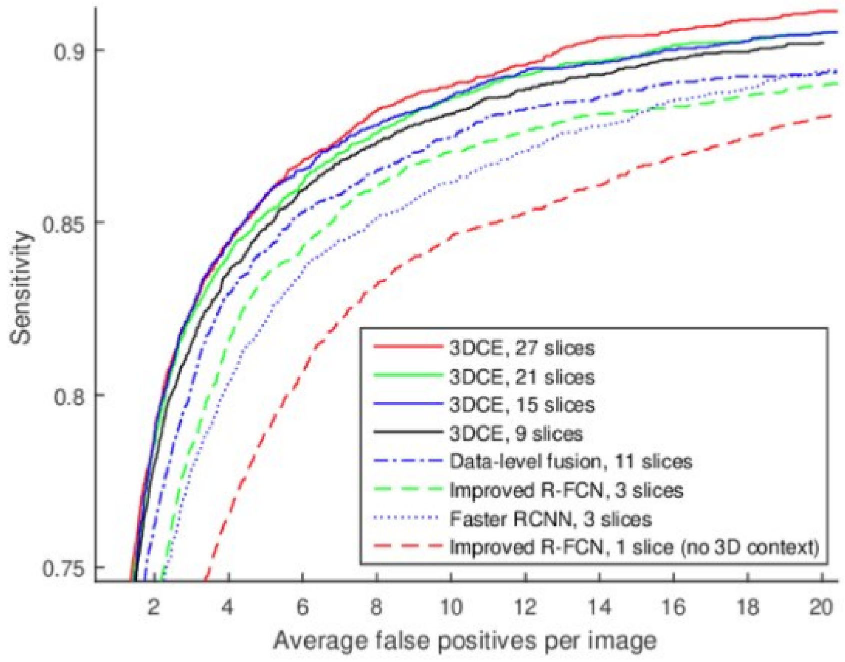


Figure 33: Example for a FROC-curve analysis of various methods on the test set of DeepLesion as shown in [59], each for a specific threshold of the output activation, deciding when a candidate belongs to a certain class. For each point, the goal is to achieve the highest possible value (normalized to [0,1]). In this particular example, we can state that the *3DCE* model with 27 slices performs significantly better than any of the *Improved R-FCN* models.

A last metric to mention is the so-called *Competition Performance Metric* (CPM), used frequently for competitions[6] and defined as the average FROC-score over the following average number of FPs per scan: $\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8$. This results in a mean sensitivity over the different FP-per-scan values s_{mean} (31). In the LNDb challenge, the *agreement level a*, i.e. the number of radiologists confirming a positive case, is also accounted for[3]. This is done in order to respect observer variability. The FROC-score and the resulting metric for nodule detection for the LNDb-challenge then become:

$$s_{mean} = \frac{1}{7}\sum_{i \in FP} s(i), \quad FP \in \{\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8\} \tag{30}$$

$$\text{score}_{\text{LNDb}} = \frac{1}{2}\sum_{a}^{2} s_{mean}(a) \tag{31}$$

3 Related Work and State of the Art

The application of object detection methods for the detection of lung nodules is a task taken upon by an increasing number of research groups over the last years. Consequently, a variety of approaches with diverse successfulness exist[6]. This can be seen on the number of new publications as well as the number of new code challenges [8, 60, 61, 62, 63] within the last 12 years, where participants are given a (usually new or newly annotated) dataset and limited time to implement the best possible model for the detection (but also segmentation and malignancy estimation) of lung nodules in CT scans. This chapter is devoted to introducing the most important currently existing datasets, the challenges that have been held over the last years and finally to showing some of the most relevant research for the topic of this thesis, coming from submissions to the challenges as well as from outside of those.

3.1 Datasets

The next sections describe the most influential datasets used for developing lung nodule detection models throughout the recent years.

3.1.1 NLST

The National Lung Screening Trial NLST [64] was a randomized controlled clinical trial in which over 54 000 people participated. They were randomly assigned to two groups: one that received low-dose CT scans, the other a single-view chest radiography. A radiologist examined the scans whereby non-calcified nodules with a diameter of more than 4mm and other abnormalities noted by the expert were annotated as officially suspicious for lung cancer. The set is not publicly available, one has to ask for permission.

3.1.2 Anode09

This dataset[65] consists of 55 CT scans, five of which have annotations, and was used for the seemingly earliest challenge on automatic nodule detection[60]. Annotations include spatial information and a binary label that states whether the nodule is malignant or not. Further information is available in the NELSON study[66], which is the dataset's origin.

3.1.3 Spie-AAPM-NCI LungX

This is a dataset with 2D - DICOM images only (single slices) put together for the LUNGx Challenge 2015[67]. It consists of a calibration set of 10 thoracic CT scans and a test set of 60 thoracic CT scans with 73 nodules. The available annotations are center coordinates of each nodule and a corresponding diagnosis in xls-format. All data is available online[63].

3.1.4 LIDC-IDRI

The Lung Image Database Consortium image collection[68] contains 1018 helical thoracic CT scans showing 1010 different patients and taken from various scanner types. The annotations, available online in an XML file, were done by four radiologists in a two-step process. In the initial phase, the radiologists independently searched for suspicious lesions and grouped them into one of the following three classes: nodule ≥ 3mm, nodule < 3mm, *non-nodule*. For all nodules in the first category diameter measurements were provided. In a second phase the radiologists reviewed the scans again, this time knowing how the other radiologists annotated each scan but without forcing consensus. The data collection process and criteria can be read under [69]. The data is available online in DICOM format.

3.1.5 LUNA

The LUNA dataset [62], used for the 2016 LUNA Grand Challenge introduced in 3.2, is a subset of the LIDC/IDRI dataset (see 3.1.4). All scans with slice thickness greater than 3mm and scans with inconsistent slice spacing were excluded, resulting in a set of 888 scans. Four radiologists were responsible for the examination. The 1186 nodules marked by at least three of them as suspicious comprise the set of positive examples. All findings with a lower consensus as well as all nodules considered having a diameter of less than 3mm were classified as *irrelevant* and not counted as neither false positives nor true positives. Annotations are provided within csv files and consist of the xyz-coordinates of each nodule's center as well as a diameter value given in mm. The data is available in DICOM format.

3.1.6 LNDb

This set[1] contains 294 CT scans with annotations from at least one radiologist released as csv files. A total of five radiologists participated in the annotation process which was performed in a single blinded fashion, i.e. in one review without knowing the others' findings. The annotation process differs slightly depending on whether a nodule's diameter is smaller or bigger than 3mm. For the latter, the procedure was similar to the one for the LIDC-IDRI dataset (see 3.1.4). Each finding is assigned a binary label *nodule* or *non-nodule*), an agreement level which corresponds to the number of radiologists who marked it as a nodule and a texture rating given as an integer between 0 (non-nodule) and 5. The latter is determined by averaging the texture classification provided by each radiologist. This year's LNDb Challenge (see 3.2) was based upon this dataset.

3.2 Recent Challenges

Code challenges play a significant role within the topic of nodule detection, the first one dating back to 2009[60]. Subsequently, the frequency of challenges increased. The two most influential ones for this work, which are also finalized and evaluated, are the LUNA 2016 Grand Challenge [62] and the Kaggle Data Science Bowl 2017 [61]. The former employs the LUNA dataset (see 3.1.5), the latter a part of the NLST set (see 3.1.1), which will be referred to as the *DSB* dataset in table 1. The latest addition to this list is the, yet to be evaluated, LNDb-Challenge mentioned in 3.1.6. Work from both of them will be mentioned in the following chapters as those challenges and the insights obtained from them are a valuable contribution to the subject.

3.3 Related Work

Typically, a detection framework is divided into two parts: the first one detecting nodule candidates and the second one responsible for reducing the number of false positive proposals coming from the first part, often referred to as the *False Positive Reduction* (FPR) network. Some works, on the other hand, use models for a combined detection and classification. Another crucial difference is whether the model's structure is set for two - or three - dimensional inputs and calculations. In the former case, an image is being fed into the model slice by slice. The latter leads to a significantly higher number of parameters and, following from that, higher computational cost and training time[6]. For all those categories a majority of the recent research relies on different forms of

CNNs or models that partially incorporate the typical layers of a CNN(2.2).
In [70] a purely two-dimensional, two-step approach is taken, in which 2D CNNs are used for detection and thereafter false positive reduction.
Mixed dimensionality methods are presented by Ding et al.[71] with the usage of a 2D Faster-RCNN for candidate proposal and a 3D CNN for the FPR-step (the former being a quite popular and successful choice in the literature[6]) and by Cheng et al.[72] with a combination of a 2D and 3D CNN for ROI proposals and a 3D CNN binary classifier taking those proposals and input and reducing the false positive rate. It is worth mentioning that this secured them the second place in the LUNA challenge.
In [73] a 2D U-Net is responsible for candidate generation and a 3D ResNet for false positive reduction.
U-Net based structures for detection and ResNet-like ones for FPR are two other favored techniques for this topic and also appear in fully three-dimensional models.
Gruetzemacher et al. [74] present such a two-step network with a ResNet-like FPR part. The U-Net used in the first part performs a *volume-to-volume prediction*, its output being a 32x32x32 cube and therefore quasi a segmentation. This is not surprising, given that U-Net structures, as discussed in 2.3.5, are broadly used for that[5].
In [75] we can see another two-step trial with a typical 3D CNN detecting nodules, in which a ResNet is used for false positive reduction, achieving better results than a DenseNet[76] but being outperformed by an FCN.
An interesting submission to the Kaggle DSB(3.2), winning the first place in this competition, is described in [77]. There, a volumetric one-stage CNN based on a U-Net backbone is used for nodule detection and a similar structure is successfully used as a classifier which predicts a cancer probability based on the detection results. Here we are interested in the detection model, especially because it is a standalone one without a dedicated FPR part. It is shown in Fig.34. This already existing model had previously influenced two very successful contributions to the LUNA challenge.
The 3rd ranked team's [49] detector was inspired by the model of the DSB-winning team and was paired with a false positive reduction step for which they used an ensemble of a standard 3D CNN, a cascaded 3D CNN and a 3D - *Wide Residual Network*.
The winning team [50] fused a one-step and a two-step approach: The detector of the second one, as well as their one-step model, are motivated by the one in Fig.34 and shown in Fig.35. The ensemble of those two models secured them the win in this chal-

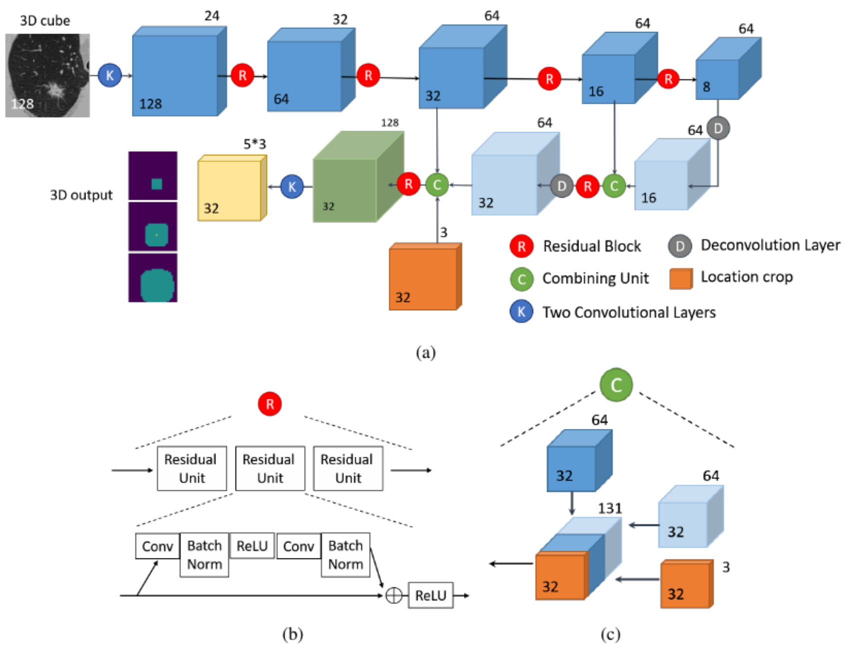


Figure 34: Detection model of the winning team [77] of the Kaggle DSB.
(a) Network structure with U-Net backbone. Each cube is a 4D-tensor. The number inside the cube stands for the spatial dimension (height = width = length), the number outside corresponds to the number of channels.(b) Structure of a Residual Block. (c) Left combining unit in the depicted model. The right combining unit is the same but does not have the location crop.

lenge.
Ensembling models is a common strategy for any type of ML task.
An example for this is given by Huang et al. [78]. Here, three 3D CNNs with different input scales are trained subsequently on the same training set, thereby adjusting the weight of the training samples before training the next CNN.
A second example is the team placed second in the Kaggle DSB 2017. In this, the weighted average of ResNet-based CNNs [79] and an adjusted form [80] of a C3D - model [81] was taken as a resulting detection. The latter is shown in Fig.36. It is important to state that in this proposal they chose a sliding-window approach. Unlike the models shown by now and the one we will use, in this one no bounding boxes are created from the image or subcrops of it (this is referred to as *bounding box regression*) but rather a window of size a^3 (in this particular case a = 32 as depicted in Fig.36) is moved over the input image. The network then learns to assign probabilities for a nodule being inside the currently placed window.

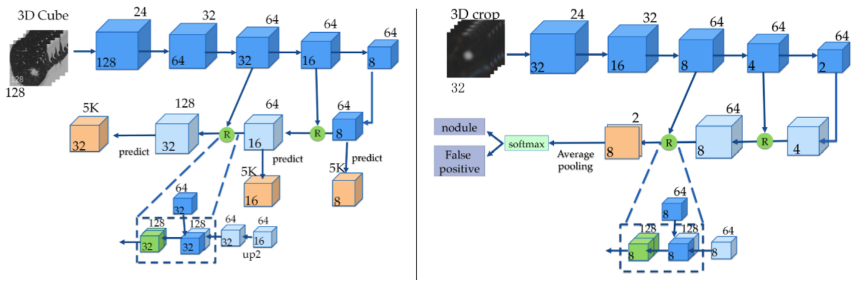
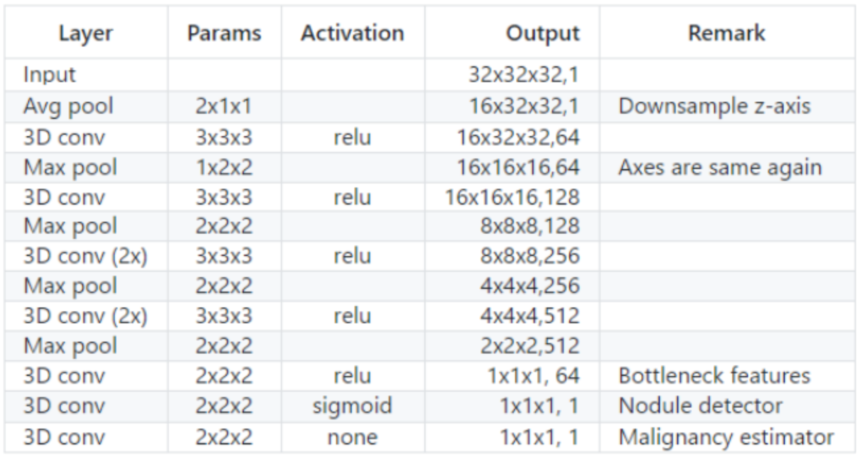


Figure 35: Two different approaches of the winning team of the LUNA challenge inspired by [77]. Left: Their detection model as part of a two-step approach in which the shown network is followed up by a false positive reduction network in the form of a 3D CNN. Right: Their one-step approach to nodule detection using a U-Net backbone. Results were obtained by fusing the two models.



Layer	Params	Activation	Output	Remark
Input			32x32x32,1	
Avg pool	2x1x1		16x32x32,1	Downsample z-axis
3D conv	3x3x3	relu	16x32x32,64	
Max pool	1x2x2		16x16x16,64	Axes are same again
3D conv	3x3x3	relu	16x16x16,128	
Max pool	2x2x2		8x8x8,128	
3D conv (2x)	3x3x3	relu	8x8x8,256	
Max pool	2x2x2		4x4x4,256	
3D conv (2x)	3x3x3	relu	4x4x4,512	
Max pool	2x2x2		2x2x2,512	
3D conv	2x2x2	relu	1x1x1, 64	Bottleneck features
3D conv	2x2x2	sigmoid	1x1x1, 1	Nodule detector
3D conv	2x2x2	none	1x1x1, 1	Malignancy estimator

Figure 36: One of the ensembled detection models for the second place contribution to the Kaggle DSB 2017. It is based on the C3D architecture [81] and utilized in a sliding-window manner.

A very recent proposal for an end-to-end, three-dimensional approach to general lesion detection with a 3D Faster-RCNN was made by Zhang et al.[82]. Here, region proposal and false positive reduction are done by different model parts but in a single phase, sharing a U-Net backbone with DenseNet blocks as shown in Fig.37. The group would have achieved 4th and 6th position in the LUNA challenge for detection and false positive reduction, taking the final scoreboard as a reference.

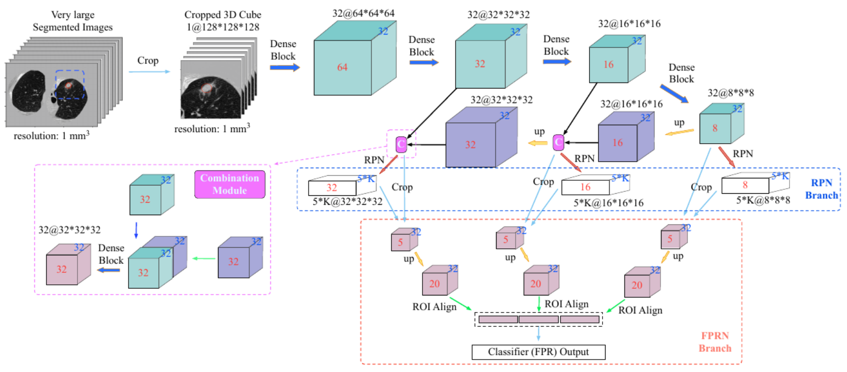


Figure 37: 3D Faster-RCNN model for lesion detection proposed in [82]. The region proposal and FPR branches are trained in a single step and share a U-Net backbone architecture.

The model that should serve as the underlying example in this work is given in [2] and shown in 38. It combines the ResNet architecture with an FPN, the meaningful feature of the latter being the lateral connections between feature maps and output heads on each scale. With their training process which will be looked into later on they achieve very high FROC scores on the LUNA - dataset.

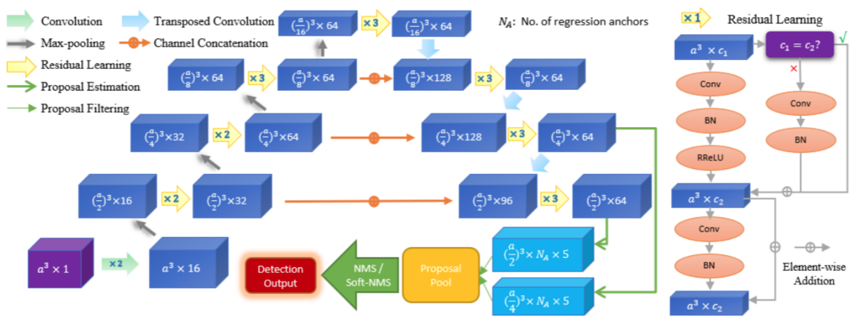


Figure 38: Single-stage detector model based on a modified ResNet and a FPN architecture. Features are being extraced on all feature map levels and directly from 3D space. The most important ResNet-property are the skip-connections.

A last example of a framework for lung nodule detection and identification is proposed by Zhang et al.[83]. The NODULe model introduced here makes use of *densely dilated convolutional blocks* (DDCB) [84]. More architectures, quantitative results as well as information on datasets can be found in [6]. The studies and the respective results presented here are outlined in table 1.

Table 1: Performance comparison of different detection models.

Ref.	Author	Year	Database	SE%	PR[a]	Acc%	AUC	CPM[a]	logloss[b]	Architecture	Dim.	Note
[70]	Setio	2016	LUNA16	85.4 90.1[c] 76.5[d]			0.996[e]	0.828/0.637[f]		Custom (based on Multi-View CNNs)	2D	
[71]	Ding	2017	LUNA16	94.6[g]						2D Faster-RCNN and 3D CNN for FPR	2D, 3D	
[72]	Cheng	2016	LUNA16					0.9499		U-Net backbone, 2D and 3D CNN for ROI-proposal, 3D CNN for FPR	2D, 3D	2nd place LUNA16
[73]	Ning	2019	LIDC-IDRI	86.5 92.3[h]				0.780		2D U-Net with 3D ResNet for FPR	2D, 3D	
[74]	Gruetze-macher	2018	LUNA16	89.29[i]			93.24[i]			U-Net backbone and ResNet-like CNN for FPR	3D	U-Net performs seg-mentation
[75]	Dou	2017	LUNA16	90.6[j]				0.839		FCN with a ResNet for FPR	3D	
[77]	Liao	2017	DSB, LUNA16[k]			81.42[l]	0.87[l]	0.8562[m]	0.39975[b]	U-Net backbone and RPN	3D	1st place DSB
[49]	Fonova	2016	LUNA16	99.1	0.0588			0.926		Ensemble (CNN, WRN-18-2, cascaded CNN)	3D	3rd place LUNA16
[50]	Bo	2016	LUNA16					0.951[n]		FPN detector with U-Net backbone	3D	1st place LUNA16
[78]	Huang	2019	LUNA16, ALT[a]					0.876		Ensemble of 3 CNNs	3D	
[79], [80]	deWit, Ham-mack	2017	DSB, LUNA16						0.40117[b]	Ensemble (ResNet-like CNNs and C3Ds)[o]	3D	2nd place DSB
[82]	Zhang	2020	LUNA16					0.939		U-Net backbone with Aggre-gated Faster-RCNN head	3D	DenseNet blocks in U-Net
[2]	Xie	2018	LUNA16					0.9351;0.9411/ **0.9767;0.9817**[p]		U-Net with Faster-RCNN head	3D	
[83]	Zhang	2018	LUNA16					0.947		LoG filter + CNN with DDCBs	3D	

[a] PR = Precision, CPM = Competition Performance Metric, ALT = Ali Tianchi (dataset can be seen under [85])

[b] The logarithmic loss (see 2.3.1) used in the Kaggle DSB ranking.

[c] For 1.0/4.0 FPs per scan respectively.

[d] On the 2D - DLCST dataset with 6 FPs per scan.

[e] For classification task.

[f] Achieved on the LUNA16 and ANODE09 datasets respectively.

[g] On 15 FPs per scan.

[h] Result of the detector-part of the model at 1.0/4.0 FPs per scan respectively.

[i] Results of the complete model (segmentation and FPR) on the LUNA16 test set with an average of 1.789 FPs per scan.

[j] Achieved at 2 FPs per scan.

[k] From the LUNA-dataset, just the nodules with a diameter of $\geq$ 6mm were taken into account.

[l] Value obtained from the test set.

[m] Evaluated on the DSB validation set.

[n] Achived 0.968 when ensembling with a two-stage FP-reduction model.

[o] The networks are small and applied in a sliding-window manner over the input image.

[p] The first pair corresponds to the result with a single model (left) and ensemble (right). The second pair corresponds to FROC-scores over {1,2,4} FPs per scan.

4 Training Methods

This chapter outlines the crucial attributes for training the neural networks. It starts with a brief description of the data used during training. Subsequently, the used architectures are introduced and the respective training process is presented.

4.1 Dataset

For training our model we made use of the LNDb dataset introduced in section 3.1.6. The whole set consists of 294 CT scans, 58 of which are withheld for the test set, i.e. their ground truth labels are not available. During preprocessing (see section 4.1.2), the remaining 236 scans were randomly assigned by us to the training and validation set with a probability of 90% to be used for training and 10% to be used for validation. This resulted in a training set containing 217 scans and a validation set containing 19 scans. Further image processing steps before the training procedure are described in the following paragraphs.
Table 2 gives an overview of significant dataset characteristics.

4.1.1 Annotations

From the 1220 available annotations already mentioned in 3.1.6, the key values for our model are the center coordinates of the nodule candidates and their volume in mm^3. Based on the latter, the diameter of a sphere with equivalent volume was calculated and this was taken as the side length of a cube containing the nodule candidate. At this point all ground truths are cubes, but how this changes during training will be discussed in section 4.1.3. The distribution of these side lengths over the dataset is shown in Fig.39

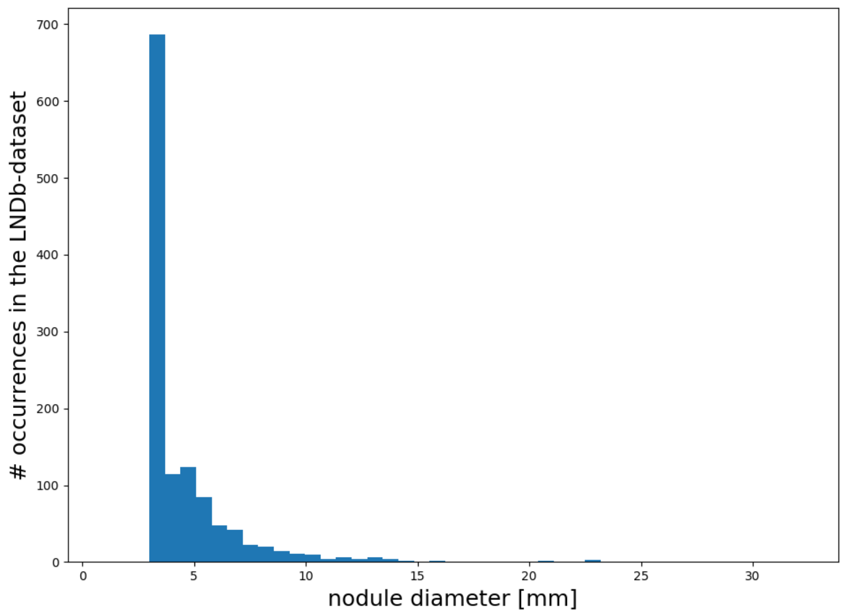


Figure 39: Size distribution of the nodule candidates in the 236 CT scans comprising the LNDb training-dataset. From the volume annotations, diameters of spheres with equivalent volume were calculated. Those diameters are interpreted as side lengths of a cube containing the nodule candidate and are the variable whose distribution is illustrated here.

4.1.2 Preprocessing

During image preprocessing the available data is homogenized as much as possible in order to enhance the model performance [86, 80, 79]. In a first standard step, the input images were resampled to a spacing of [1mm, 1mm, 1mm]. This results in every voxel representing a cube of size 1mm $\times$ 1mm $\times$ 1mm.
Pixel intensities in CT scans can be expressed in *Hounsfield Units* (HU). This unit quantifies the radiodensity of different tissue, with water arbitrarily defined at zero HU and air at -1000 HU [87]. It is common to clip the input image to a given HU range, thereby also setting boundaries for the pixel intensity depending on which kind of tissue is of importance [77, 80, 49]. Due to the fact that bone structure has HU values of over 400, we set the HU range to [-1000, 400], as done for example in [80, 79]. For further uniformity, the pixels within this range were clipped to [0, 255] which is another frequently taken step [50].
Naturally, the model inputs need to be of the same size, which in our case is 128 $\times$ 128 $\times$ 128. To achieve that, 10 cubes of this size were randomly extracted from each of the 236

scans. In order to avoid too many cubes without annotations, their center coordinates were randomly sampled from a multimodal distribution with peaks around the nodule candidate coordinates in each spatial direction. It is essentially a normalized sum of multiple normal distributions with the same standard deviation. An example is shown in Fig.40.

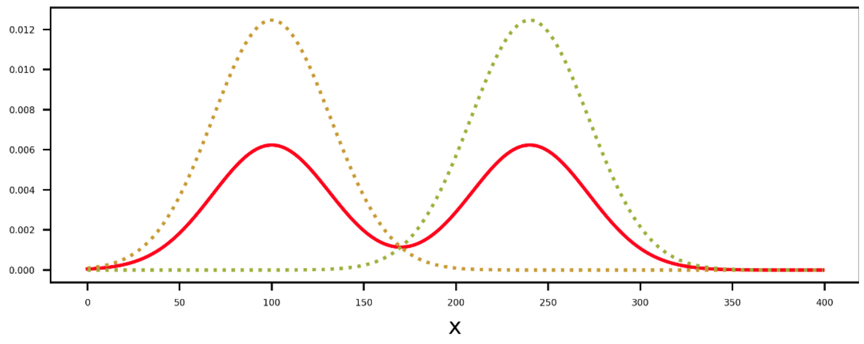


Figure 40: Example of a multimodal distribution centered around $x_1 = 100$ and $x_2 = 240$ for an input range of [0, 400]. The green and yellow curve show two normal distributions centered around x_1 and x_2, each of them with a standard deviation of 32. Those two are added and normalized to obtain the multimodal distribution depicted in red. Such distributions were used for sampling model input cubes from the preprocessed data.

We picked a standard deviation of 32 for each of the normal distributions based on the already introduced size distribution of the nodule candidates (see Fig.39).
The standard deviation was selected to ensure a sufficiently high number of model input cubes containing at least one nodule candidate without all of the possible nodules being just around the cubes' center. As a result, 2170 training cubes and 190 validation cubes were created. Out of these 2360 model inputs, 1732 contain at least one annotated nodule candidate.
The last step before supplying an image to the model consists of subtracting the training set's mean pixel value (mean = 93.451) from every pixel of every input cube and dividing each by the pixel values' standard deviation (standard deviation = 85.941). This standardizes the pixel distribution to one with zero mean and unit variance which is favourable for regression tasks [86, 72].

Table 2: Dataset overview.

nr. training scans	217
nr. validation scans	19
nr. test scans	58
nr. training cubes	2170
nr. validation cubes	190
mean pixel value (training cubes)	93.451
standard deviation of pixel values (training cubes)	85.941
mean nodule diameter (all original scans)	4.533 mm
minimal nodule diameter (all original scans)	3.0 mm
maximal nodule diameter (all original scans)	30.85 mm

4.1.3 Data Augmentation

Augmentations are image transformations used to increase the diversity of the training data [88]. Popular choices are rotations, translations, zooming or flipping of axes [88, 2, 75, 80, 72]. Applying those transformations leads to an effectively larger dataset and reduces the probability of overfitting (see section 2.1.5). In order to save storage space, augmentations were done during the training phase after loading an input cube. Figure 41 shows four examples. The following strategies were employed:

- translations in each direction by 2.5% of the input cube's side length
- random rotations by one of the following values: [0°, 90°, 180°, 270°]
- random zooming within a range of [0.75, 1.25]
- random flipping around an arbitrary axis

In the frame of this work, translations play an additional role:
As pointed out in section 4.1, the ground truth boxes are initially cubes. Because of translations, those boxes can be partially cut off, resulting in cuboids. This is important for the network's output size which is part of the next section.

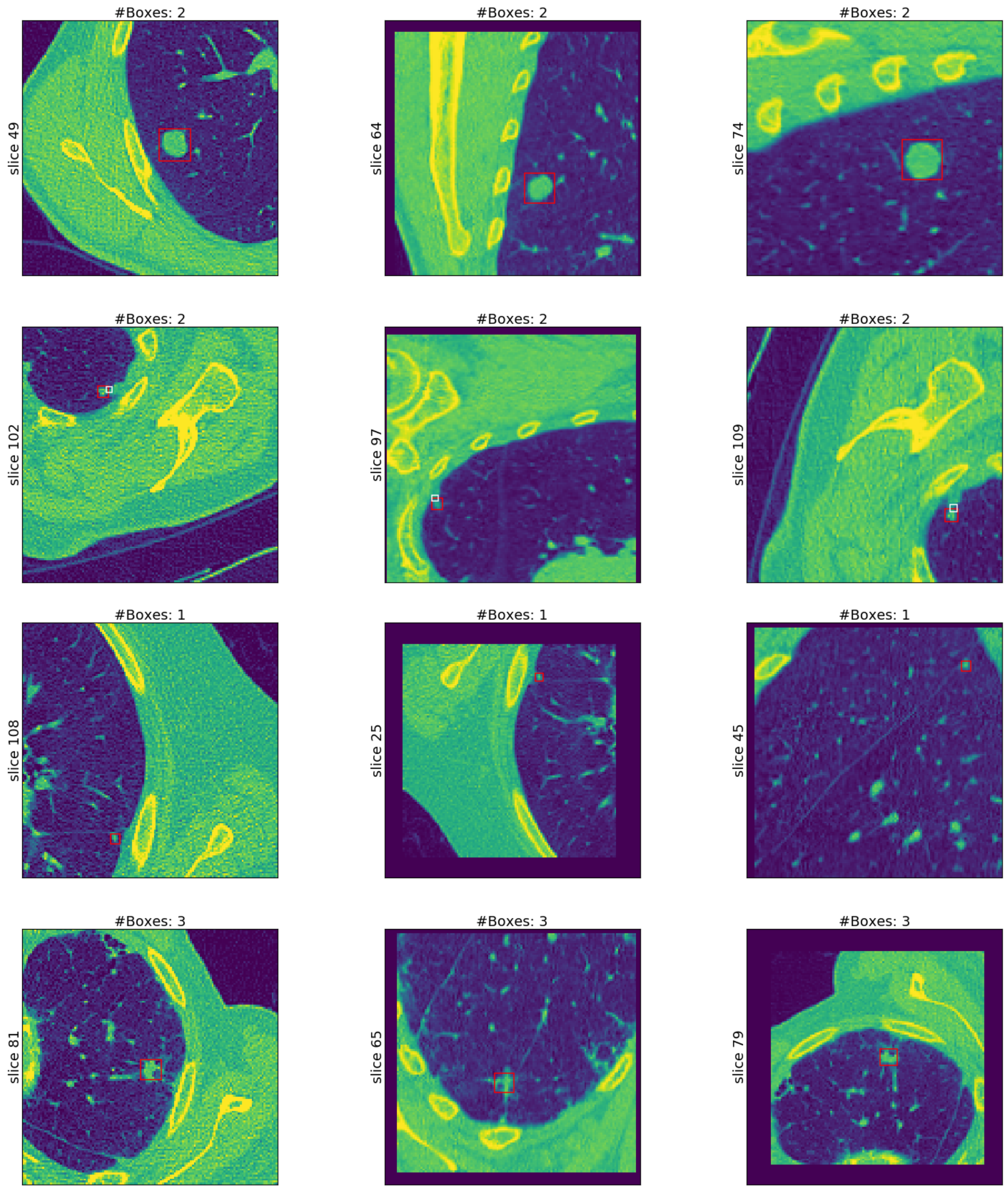


Figure 41: Example augmentations demonstrated with slices from four cubes. The left-most image in each row is from an original input cube, the other two are slices from a randomly augmented version of the same cube. Red boxes denote ground truths belonging to the *nodule* class, white boxes are *non-nodules*. The number of boxes inside a cube (indicated by the label above each image) may decrease depending on the performed transformation.

4.2 Network Architecture

The architecture chosen in this work is based on the one used by Xie [2] and shown in Fig.38. The first important variation we propose is replacing the Faster-RCNN-like head of Xie's model by an SSD-like detector head. This manifests itself as follows: instead of directly creating the whole prediction tensor indicated by the light blue boxes in Fig.38, we use separate convolutions to obtain confidence and localization predictions and apply a softmax activation (see section 2.1.1.1) to the former.
The number of predictions in this case is determined by the amount of different anchor sizes (see section 2.3.2) we choose to use on the feature maps and by the chosen stride. In this case, the anchors were centered at every feature map voxel and had the following sizes, similar to the ones used in [2]: $\langle 3, 5, 7, 10, 13, 17, 22, 30, 40 \rangle$. This selection fits our data because it captures the whole range of nodule candidate sizes illustrated in Fig.39. This approach also allows for predicting different classes, which is crucial in our case, given that we have to differentiate between *nodules* and *non-nodules* (and the automatically induced *background* class arising from the SSD architecture (see section 2.3.2)). The second difference to the model presented by Xie is the fact that, as mentioned in section 4.1.3, some ground truth boxes are cuboids. This has to be taken account for in the output vector representing each prediction. Consequently, our output vectors are of the form

$$\begin{pmatrix} \mathrm{conf}_{\mathrm{background}} \\ \mathrm{conf}_{\mathrm{nodule}} \\ \mathrm{conf}_{\mathrm{non\text{-}nodule}} \\ \mathrm{x}_{\mathrm{ctr}} \\ \mathrm{y}_{\mathrm{ctr}} \\ \mathrm{z}_{\mathrm{ctr}} \\ \mathrm{width}(\Delta \mathrm{x}) \\ \mathrm{height}(\Delta \mathrm{y}) \\ \mathrm{length}(\Delta \mathrm{z}) \end{pmatrix}$$

(where $\mathrm{conf}_{\mathrm{A}}$ denotes the confidence value for the box containing an object of class A) and allow for three distinct side lengths. In addition to that, we adopted Leaky ReLU activations (see section 2.1.1.1) after each convolution layer. The upsampling and downsampling parts of the model are delineated in tables 3 and 4. There are three residual blocks with $c_1 = c_2 = 64$ (see Fig.38) between them.

Table 3: Upsampling part of our model. The 3D-convolution parameters are kernel-sizes, max-pooling parameters correspond to pooling-sizes, the batch normalization parameters are momentum values used in the tensorflow library and c_1 and c_2 are the parameters of the residual blocks in Fig.38.

Layer	Parameters	Output
Input		$128 \times 128 \times 128, 1$
3D Convolution	$(5, 5, 5)$	$128 \times 128 \times 128, 16$
Batch Norm	0.9	$128 \times 128 \times 128, 16$
Leaky ReLU		$128 \times 128 \times 128, 16$
3D Convolution	$(5, 5, 5)$	$128 \times 128 \times 128, 16$
Batch Norm	0.9	$128 \times 128 \times 128, 16$
Leaky ReLU		$128 \times 128 \times 128, 16$
Max Pooling	$(2, 2, 2)$	$64 \times 64 \times 64, 16$
Residual Block	$c_1 = c_2 = 16$	$64 \times 64 \times 64, 16$
Residual Block	$c_1 = 16, c_2 = 32$	$64 \times 64 \times 64, 32$
Max Pooling	$(2, 2, 2)$	$32 \times 32 \times 32, 32$
Residual Block	$c_1 = c_2 = 32$	$32 \times 32 \times 32, 32$
Residual Block	$c_1 = 32, c_2 = 64$	$32 \times 32 \times 32, 64$
Max Pooling	$(2, 2, 2)$	$16 \times 16 \times 16, 64$
Residual Block	$c_1 = c_2 = 64$	$16 \times 16 \times 16, 64$
Residual Block	$c_1 = c_2 = 64$	$16 \times 16 \times 16, 64$
Residual Block	$c_1 = c_2 = 64$	$16 \times 16 \times 16, 64$
Max Pooling	$(2, 2, 2)$	$8 \times 8 \times 8, 64$

Table 4: Downsampling part of our model. *T. Conv* denotes transposed 3D convolutions.

Layer	Parameters	Output
Input		$8 \times 8 \times 8, 64$
T. Conv + Concatenation	$(5, 5, 5)$	$16 \times 16 \times 16, 128$
Residual Block	$c_1 = 128, c_2 = 64$	$16 \times 16 \times 16, 64$
Residual Block	$c_1 = c_2 = 64$	$16 \times 16 \times 16, 64$
Residual Block	$c_1 = c_2 = 64$	$16 \times 16 \times 16, 64$
T. Conv + Concatenation	$(5, 5, 5)$	$32 \times 32 \times 32, 128$
Residual Block	$c_1 = 128, c_2 = 64$	$32 \times 32 \times 32, 64$
Residual Block	$c_1 = c_2 = 64$	$32 \times 32 \times 32, 64$
Residual Block	$c_1 = c_2 = 64$	$32 \times 32 \times 32, 64$
T. Conv + Concatenation	$(5, 5, 5)$	$64 \times 64 \times 64, 96$
Residual Block	$c_1 = 96, c_2 = 64$	$64 \times 64 \times 64, 64$
Residual Block	$c_1 = c_2 = 64$	$64 \times 64 \times 64, 64$
Residual Block	$c_1 = c_2 = 64$	$64 \times 64 \times 64, 64$

4.3 Training

There is a variety of hyperparameters (see section 2.1.4) that need to be set for training such a model. Given that the available time frame did not allow for detailed hyperparameter tuning, reasonable settings had to be picked with respect to the available literature.
The network was set to train for up to 500 epochs as was done by Xie [2], thereby keeping track of the weight combination currently producing the lowest validation loss. The batch size was set to 1 (due to memory restrictions) with 2170 steps per epoch and 190 validation steps. This assures that the model sees exactly one variation of each training and validation cube per epoch. In the upcoming paragraphs we describe other important training parameters. Finally, the three setups with the lowest validation loss are summarized in table 5.

Matching Strategy For the anchor based approach it is pivotal to define the IoU thresholds for the default anchors with the ground truth boxes used in the matching process (see section 2.3.2). We need an upper threshold t_+, indicating a successful match of an anchor with a ground truth and a lower threshold t_-. All anchors which do not exceed the latter with any ground truth are considered to be *negatives* (background boxes, see section 2.3.2) during the loss computation and the ones in between those thresholds are *neutral boxes*, i.e. they are not part of the loss computation. Following the parameters selected in [2], $\mathrm{t}_+ = 0.3$ and $\mathrm{t}_- = 0.001$ were picked.
A last value to mention for all anchor boxes of sizes ⟨3, 5, 7, 10, 13, 17, 22, 30, 40⟩ is the standard deviation of their center coordinates and side lengths, denoted by the vector $\vec{\sigma}_{\mathrm{anchor}} = (\sigma_{\mathrm{x}}, \sigma_{\mathrm{y}}, \sigma_{\mathrm{z}}, \sigma_{\mathrm{w}}, \sigma_{\mathrm{h}}, \sigma_{\mathrm{l}})$. It is incorporated during the encoding (see eq.21) by dividing each regression parameter by a preset standard deviation value[89]. The values we chose were $\sigma_{\mathrm{x}} = \sigma_{\mathrm{y}} = \sigma_{\mathrm{z}} \in \{\sqrt{0.1}, 0.1, 1\}$ for the center coordinates and $\sigma_{\mathrm{w}} = \sigma_{\mathrm{h}} = \sigma_{\mathrm{l}} \in \{\sqrt{0.2}, 0.2, 1\}$ for the side lengths. The absolute values and their application follow the discussions in [89, 90, 91].

Optimizer The Adam optimizer presented in section 2.1.3.4 was used in all training runs. It is a popular choice [72, 2] and promises faster convergence.
We employed an initial learning rate of 0.001 with up to two reductions by a factor of 10 (minimal learning rate = 0.00001) on plateaus of the validation loss with a patience of

10 epochs [92]. Learning rates within this range have been used repeatedly for similar tasks [50, 2, 72, 78].

Loss computation The loss was computed as is typical for SSD networks and discussed in section 2.3.2. An issue that can emerge during training is the imbalance between positive and negative examples arising from the fact that there are at most a few nodule candidates per input image and the majority of default boxes will be negatives. This would result in the learning process being much more focused on the negative class. Measures taken to prevent this imbalance are referred to as *hard negative mining* and can be found in most of the related models [2, 50, 77, 72].
A first hard negative mining technique consists of setting a maximal *negative to positive ratio* $r_{n:p}$ for each batch[9]. In this way, not more than $\mathrm{r_{n:p}} \times \mathrm{num_{pos}}$ negative examples ($\mathrm{num_{neg}}$) are used for computing the loss. The negative examples entering the loss are the ones with the highest confidence loss[9].
In batches without any positive box, this would lead to learning only background information. This can be prevented by setting a minimal number of negatives $\mathrm{min_{neg}}$ to be considered in the loss. Based on recommendations in [2, 91], reasonable $\{\mathrm{r_{n:p}}, \mathrm{min_{neg}}\}$ pairs utilized during training were {3,3} and {5,5}. As suggested in [9], the α-value in eq.15 was set to $\alpha = 1$.

Table 5: Hyperparameters for training runs.

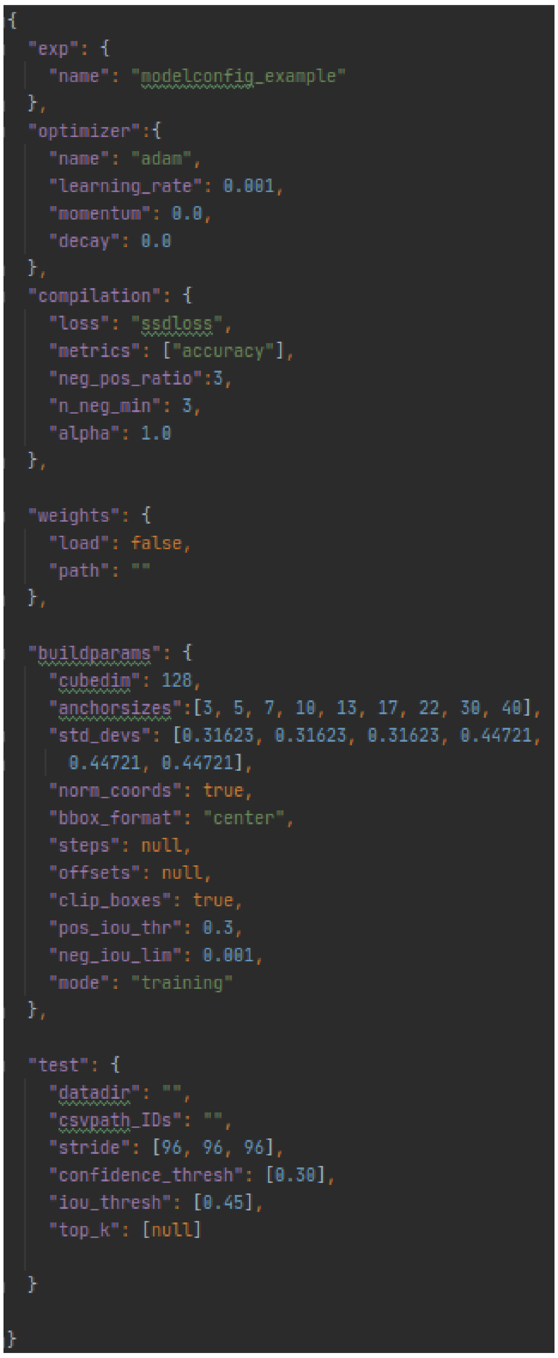
Hyperparameter	Value		
batch size	1		
maximal epoch number	500		
training steps per epoch	2170		
validation steps per epoch	190		
initial learning rate	0.001		
minimal learning rate	0.00001		
$\mathrm{r_{n:p}}$	3	5	5
$\mathrm{min_{neg}}$	3	5	5
$\mathrm{t_+}$	0.3		
$\mathrm{t_-}$	0.001		
$\sigma_\mathrm{x}, \sigma_\mathrm{y}, \sigma_\mathrm{z}$	$\sqrt{0.1}$	0.1	0.1
$\sigma_\mathrm{w}, \sigma_\mathrm{h}, \sigma_\mathrm{l}$	$\sqrt{0.2}$	0.2	0.2
augmentations	all	all	None

4.4 Inference and Postprocessing

After the model has been trained, it naturally needs to be tested and evaluated on original scans. Their size is sometimes more than 500 voxels in each dimension and therefore incompatible with the network's input size of $128 \times 128 \times 128$. For this reason, during inference a window of size $128 \times 128 \times 128$ is slided over the scan. The model creates proposals for each such cube and they are transformed into world coordinates in accordance to the original ground truth annotations they will be compared against. The stride of the sliding window was set to 96, so that the overlap with the previous window is always at least 32 voxels. This resulted in approximately 100 cubes per input scan and a time of around one hour for each scan. Given that the spacing of the scans in the dataset is often around 0.5mm in x- and y-direction and 1.0mm in z-direction, 32 voxels seems to be a reasonable compromise to ensure that every nodule candidate is properly inside the window at least once (the size distribution is depicted in Fig.39) and at the same time the inference time remains sensible.
In each prediction process, the model's raw output is decoded, reversing the box encoding in eq.21. Afterwards, boxes with unreasonable parameters, e.g. side lengths which are negative or much larger than the image, center coordinates outside of the image frame etc., are discarded if they exist. They do not necessarily appear but can justifiably be removed, given that they are obviously not useful and potentially harm the further postprocessing procedure. The latter happens if some of those boxes belong to either the *nodule* or the *non-nodule* class. Their existence implies that the anchor sizes need to be adjusted.
As a last postprocessing step for every single prediction we used confidence thresholding with $t_{\text{conf}} = 0.3$ and non-maximum suppression with $t_{\text{iou, nms}} = 0.45$ (see section 2.3.4). The confidence threshold was chosen from observations of the approximate amount of proposals left after applying it. At $t_{\text{conf}} = 0.3$, between 20 and 3000 predictions are left for every cube. Having 100 cubes per scan, this is a sufficient number of candidates.
Finally, we employed two top-k filtering strategies (see section 2.3.4) for each scan: We separately evaluated the $k = 200$ highest confidence candidates as well as all predictions yielded from the procedure described in this paragraph.

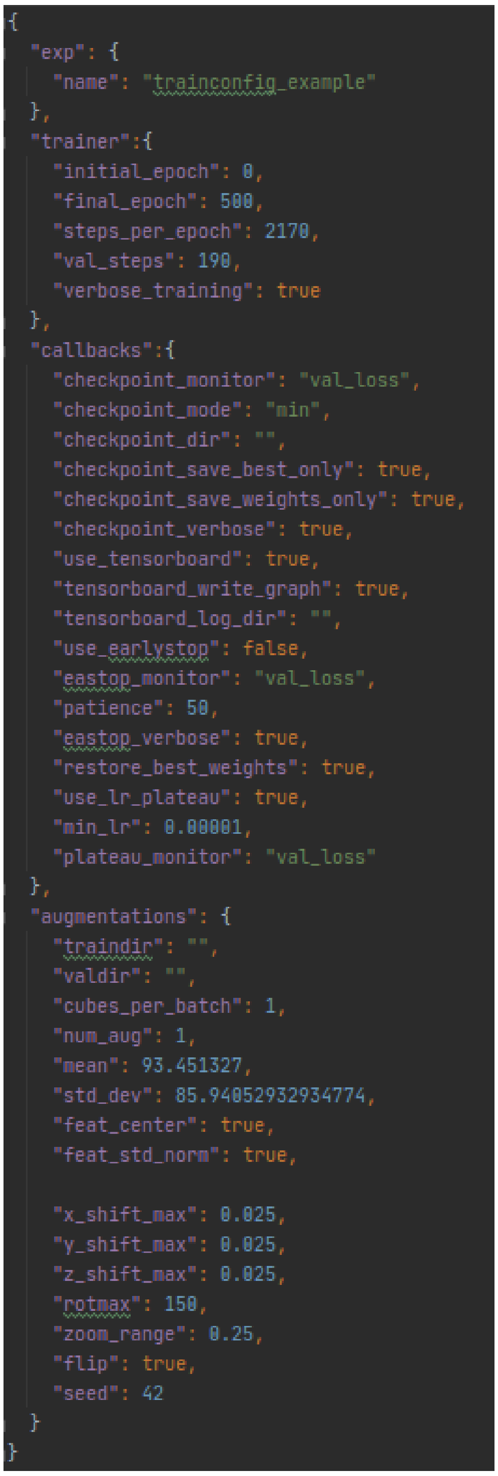
4.5 The Framework

The framework, inside of which everything described in this chapter happens, is implemented in Keras[93] with Tensorflow backend[94]. It is modular and makes it possible to execute each step of the pipeline individually, with the option of adjusting many parameters conveniently.
The image preprocessing and input cube creation can be performed at once or separately and can both be customized via distinct configuration dictionaries. These contain all necessary parameters, including file paths, data sets to load, Hounsfield units to clip the images to, the number of cubes to create per scan, parameters for the distribution from which the center coordinates of cropped cubes are sampled, the training-validation ratio and many more. Having started the training process, a custom image data generator loads and augments the created cubes during run-time.
The training procedure and the model used for it depend on configuration files that are stored in json format. The training configuration holds hyperparameters like the batch size or epoch number, variables regarding the callbacks[95] we use during training and all augmentation parameters.
Model configurations contain optimizer information, variables that are necessary for the applied loss function, crucial network architecture parameters and thresholds concerning the inference time in order to be able to perform tests. Figure 42 shows examples of both configuration types. Individual testing procedures for determining precision, recall and FROC-scores are also implemented. Lastly, a variety of scripts with visualizations of different fragments of the whole pipeline is provided.
The code is available under [96].



```
{
  "exp": {
    "name": "modelconfig_example"
  },
  "optimizer":{
    "name": "adam",
    "learning_rate": 0.001,
    "momentum": 0.0,
    "decay": 0.0
  },
  "compilation": {
    "loss": "ssdloss",
    "metrics": ["accuracy"],
    "neg_pos_ratio":3,
    "n_neg_min": 3,
    "alpha": 1.0
  },

  "weights": {
    "load": false,
    "path": ""
  },

  "buildparams": {
    "cubedim": 128,
    "anchorsizes":[3, 5, 7, 10, 13, 17, 22, 30, 40],
    "std_devs": [0.31623, 0.31623, 0.31623, 0.44721,
      0.44721, 0.44721],
    "norm_coords": true,
    "bbox_format": "center",
    "steps": null,
    "offsets": null,
    "clip_boxes": true,
    "pos_iou_thr": 0.3,
    "neg_iou_lim": 0.001,
    "mode": "training"
  },

  "test": {
    "datadir": "",
    "csvpath_IDs": "",
    "stride": [96, 96, 96],
    "confidence_thresh": [0.30],
    "iou_thresh": [0.45],
    "top_k": [null]

  }

}
```

(a)



```
{
  "exp": {
    "name": "trainconfig_example"
  },
  "trainer":{
    "initial_epoch": 0,
    "final_epoch": 500,
    "steps_per_epoch": 2170,
    "val_steps": 190,
    "verbose_training": true
  },
  "callbacks":{
    "checkpoint_monitor": "val_loss",
    "checkpoint_mode": "min",
    "checkpoint_dir": "",
    "checkpoint_save_best_only": true,
    "checkpoint_save_weights_only": true,
    "checkpoint_verbose": true,
    "use_tensorboard": true,
    "tensorboard_write_graph": true,
    "tensorboard_log_dir": "",
    "use_earlystop": false,
    "eastop_monitor": "val_loss",
    "patience": 50,
    "eastop_verbose": true,
    "restore_best_weights": true,
    "use_lr_plateau": true,
    "min_lr": 0.00001,
    "plateau_monitor": "val_loss"
  },
  "augmentations": {
    "traindir": "",
    "valdir": "",
    "cubes_per_batch": 1,
    "num_aug": 1,
    "mean": 93.451327,
    "std_dev": 85.94052932934774,
    "feat_center": true,
    "feat_std_norm": true,

    "x_shift_max": 0.025,
    "y_shift_max": 0.025,
    "z_shift_max": 0.025,
    "rotmax": 150,
    "zoom_range": 0.25,
    "flip": true,
    "seed": 42
  }
}
```

(b)

Figure 42: Example of a model configuration(a) and training configuration(b) in json format.

5 Results and Discussion

In this section the results obtained from the training process described in chapter 4.3 are presented. The lowest validation loss of $\text{loss}_{\text{val}} = 2.51$ was achieved in the 67th epoch after four days of training with the leftmost parameter combination in table 5. The setup in the middle column delivered a loss of $\text{loss}_{\text{val}} = 5.13$ after 80 epochs, the one in the right column was stopped after reaching a loss of $\text{loss}_{\text{val}} = 5.57$ in the 50th epoch. It is worth mentioning that the following parameter combinations led to early termination due to lack of improvement or errors during training:

- A setup as in column 2 in table 5 but with $\text{r}_{\text{n:p}} = \text{min}_{\text{neg}} = 3$ resulted in $\text{loss}_{\text{val}} =$ nan (not a number) after a few epochs.
- A setup as in column 2 in table 5 but with $\sigma_{\text{x}} = \sigma_{\text{y}} = \sigma_{\text{z}} = \sigma_{\text{w}} = \sigma_{\text{h}} = \sigma_{\text{l}} = 1$ did not lead to any improvement compared to the best performing runs.
- Any training run in which the default anchors (see sections 2.3.2 and 4.2) were not clipped to the image boundaries but allowed to exceed them led to $\text{loss}_{\text{val}} = \text{nan}$.

Precision, Recall The precision and recall metrics introduced in eq.26 were calculated for the 19 validation scans with thresholds (see section 2.3.4)

$$\text{t}_{\text{iou}} \in \{0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$$

according to the standard range defined in the COCO challenge[4]. In its current state, the model achieved a highest IoU to any ground truth box of 0.2, leading to a precision and recall of zero on the COCO-range for the top-k filtered as well as for the non-filtered detection results (see section 4.4).

FROC score The FROC score (explained in section 2.4.4) was calculated for the 19 validation scans based on the algorithm described in [3]. There, the confidence threshold over the predictions is varied. For each threshold, the recall and number of FPs per scan are calculated, only taking into account predictions of type *nodule*. The recall calculation is based on the following rule: Every candidate that matches a ground truth is considered a true positive, where a *match* means that the distance between the center coordinates of the nodule and the ground truth is smaller or equal than the maximum equivalent ground truth diameter. A nodule candidate that matches a non-nodule

ground truth is a false positive and ground truths that were not matched are false negatives.
We calculated the FROC-score over a confidence threshold range of $t_{\text{conf, froc}} \in [0.3, 0.99]$ with a step size of 0.03. The result, with a highest recall of 0.68 at 42963 FPs per scan for the non-filtered detection output, is shown in Fig.43. After top-k filtering with $k = 200$ there were no true positives among the candidates, leading to a score of zero.

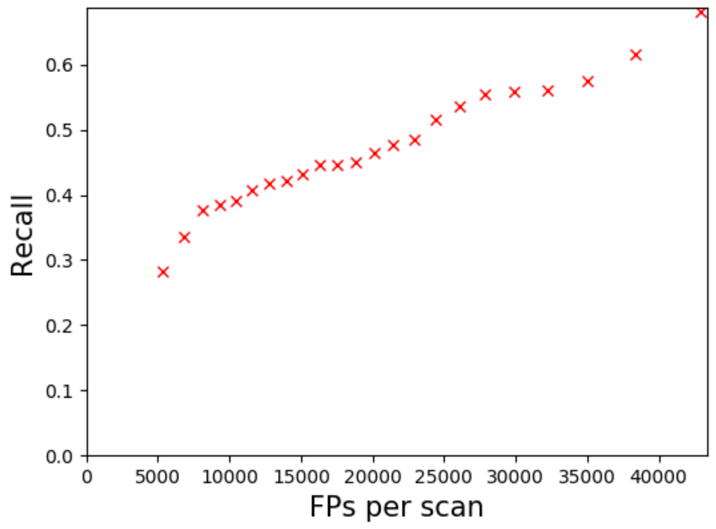
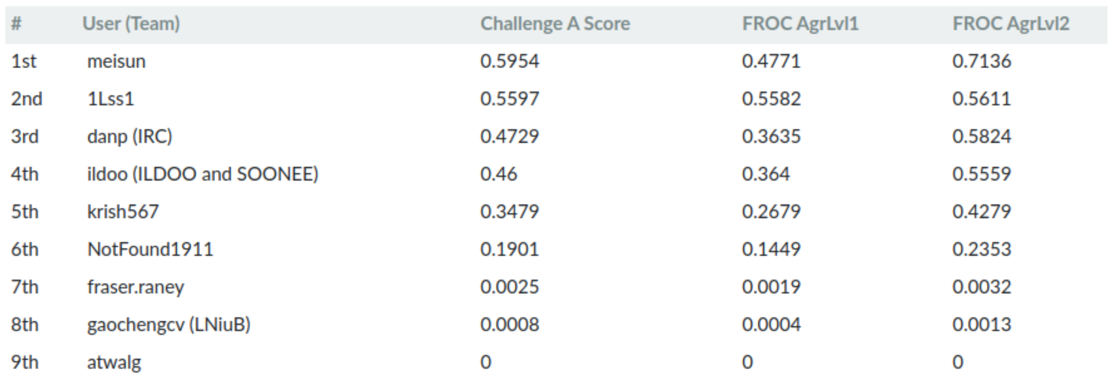


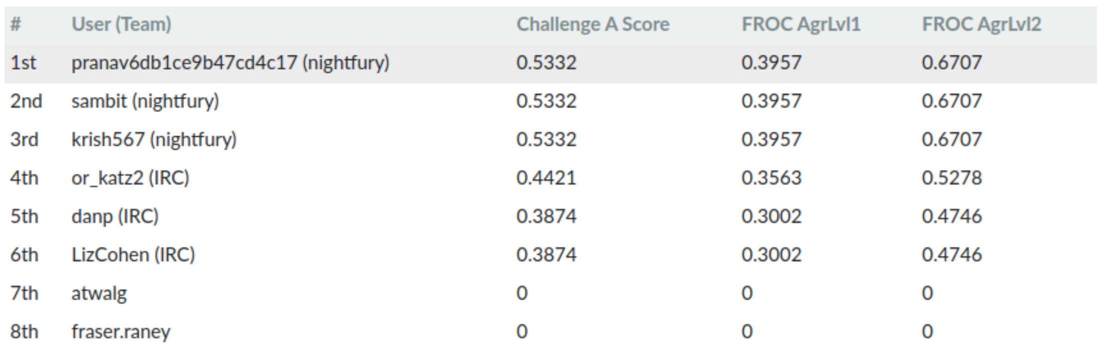
Figure 43: FROC analysis result for our model obtained from a confidence threshold range of [0.3, 0.99] with a step size of 0.03.

The respective scoreboards of the LNDb Challenge A, which is the nodule detection challenge we are interested in, are shown in Fig.44 and Fig.45.



#	User (Team)	Challenge A Score	FROC AgrLvl1	FROC AgrLvl2
1st	meisun	0.5954	0.4771	0.7136
2nd	1Lss1	0.5597	0.5582	0.5611
3rd	danp (IRC)	0.4729	0.3635	0.5824
4th	ildoo (ILDOO and SOONEE)	0.46	0.364	0.5559
5th	krish567	0.3479	0.2679	0.4279
6th	NotFound1911	0.1901	0.1449	0.2353
7th	fraser.raney	0.0025	0.0019	0.0032
8th	gaochengcv (LNiuB)	0.0008	0.0004	0.0013
9th	atwalg	0	0	0

Figure 44: FROC score ranking over the train/validation set of the LNDb Challenge A[97]



#	User (Team)	Challenge A Score	FROC AgrLvl1	FROC AgrLvl2
1st	pranav6db1ce9b47cd4c17 (nightfury)	0.5332	0.3957	0.6707
2nd	sambit (nightfury)	0.5332	0.3957	0.6707
3rd	krish567 (nightfury)	0.5332	0.3957	0.6707
4th	or_katz2 (IRC)	0.4421	0.3563	0.5278
5th	danp (IRC)	0.3874	0.3002	0.4746
6th	LizCohen (IRC)	0.3874	0.3002	0.4746
7th	atwalg	0	0	0
8th	fraser.raney	0	0	0

Figure 45: FROC score ranking over the test set of the LNDb challenge A[98].

Discussion The results reveal that the model is still not well enough trained in order to perform lung nodule detection. On the one hand, the bounding box center coordinates are predicted inaccurately. Following from this, the FROC calculation showed a huge number of false positives. On the other hand, it is clear that the shapes of the candidate boxes are also unsatisfactory. This shows itself in the fact that, even without top-k filtering, the highest IoU between prediction and ground truth for the validation set was 0.2. Therefore, even the boxes that are close enough to a ground truth to be considered TPs in the FROC paradigm can not assure an IoU value that is large enough for a meaningful calculation of precision and recall.
The biggest issue was the narrow time frame, which did not allow for sufficient optimization of the training process. With over one hour of training time per epoch, there was not enough time to train the model for the planned number of 500 epochs and its currently best checkpoint had to be utilized to carry out the evaluation procedure. The unsuitable predicted box measures give out a second possible issue: the choice of anchor sizes. Misshaped boxes can appear due to anchor sizes that do not fit the dataset. A different choice may lead to much better results.
Besides that, there is a variety of ideas mentioned in different papers which could enhance the model's performance significantly. The implementation of these methods exceeded the scope of this work, so we will only discuss their potential improvements here. The arguably biggest improvement that can be done during preprocessing is the segmentation of the lung volume. This procedure removes unnecessary tissue outside of the lung from the scan which may slow down the learning process of the model and is used in many approaches [77, 80, 78, 82, 74, 83]. It is worth mentioning that in [80] and [74] U-Nets are used for this task and in [77] it is pointed out that simultaneous training of segmentation and detection models usually increases the performance of both tasks.

A next possible improvement strategy is the employment of model ensembles. As was discussed in 3, many of the well performing models are the result of a weighted average of multiple models trained for object detection. This is something that can be done fairly easily within the framework in the future.

In the context of the model's architecture, an important note is that the model was trained from scratch, which introduces a higher risk of insufficient results, especially in such early training phases. A reason for that is the already mentioned lack of 3D models that are working with volumetric data and trained on similar tasks. Not only are there many more pretrained 2D network structures, but among the already tested 3D models, ones that use volumetric data instead of point clouds and consist of a single, end-to-end trainable stage (rather than separately trained parts for detection and FP reduction) are rare.

Further modifications which could positively influence the training process include the replacement of the Leaky ReLU activations with Randomized ReLU, which has been employed in well-performing models[2, 82].
Moreover, modified loss functions seem to be able to improve the performance, the *Focal Loss* being a prominent example utilized in successful architectures[50, 82]. In [99] it is pointed out that this fairly new type of loss function is helpful with regards to the class imbalance in detection models and can be of particular use in one-stage detectors like the one presented in this work. This can be crucial for single-phase detection models, as they are usually outperformed by multi-phase solutions (with separate detection and FP reduction parts).
The application of *Soft NMS* instead of the common NMS, mentioned in [2] and [82], is another possible adjustment that has proven to be effective in detection networks[100].

In addition, the training procedure can profit from many custom adaptations and a variety of such can be found in the relevant literature. They range from applying morphological operations to the input image in the preprocessing stage[77, 83], over anchor-based sampling during training introduced by Xie[2], to adding local magnification layers which can result in better sensitivity and FROC-scores[82].
Including those into the present framework, especially in combination with more datasets, should help to improve the model performance.

6 Conclusion

The present work analyzed the application of Convolutional Neural Networks to the task of lung nodule detection in CT scans. It aimed to integrate a single-stage, end-to-end trainable network which learns from volumetric data into a framework that incorporates image preprocessing, model training and evaluation procedures. Using basic image preprocessing on the LNDb-dataset[1], this pipeline was executed for a model that is based on a U-Net-like 3D feature extractor with a 3D detector head adapted from the 2D-SSD architecture. We achieved a maximal recall of 0.68 at 42963 false positives per scan. Due to the low accuracy of bounding box predictions, precision and recall calculations on the COCO IoU-range and the calculation of the FROC-score over the typically used $\{ \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8 \}$ FPs per scan can not be performed meaningfully at this stage.
Although the herein presented approach is not yet suitable for accurate lung nodule detection, the theoretical background and state-of-the art analysis together with a larger time frame for model training promise to be keys for further development and an enhanced performance. As explained in section 5, there are numerous methods that can be added to the pipeline in order to improve its performance, ranging from additional image processing steps, to a variety of network architectures, to the portrayed enhancements of the training procedure. With the ability to easily include those into the framework, we can be optimistic that we will see better results based on this work in the future.

Literature

[1] João Pedrosa et al. *LNDb: A Lung Nodule Database on Computed Tomography*. 2019. arXiv: 1911.08434 [eess.IV].

[2] Zhongliu Xie. *Towards Single-phase Single-stage Detection of Pulmonary Nodules in Chest CT Imaging*. 2018. arXiv: 1807.05972 [cs.CV].

[3] *Evaluation in the LNDb grand challenge 2019/20*. https://lndb.grand-challenge.org/Evaluation/. (Visited on 09/13/2020).

[4] *COCO Evaluation Metrics*. https://cocodataset.org/#detection-eval. (Visited on 09/21/2020).

[5] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical Image Analysis* 42 (2017), pp. 60 –88. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.07.005. URL: http://www.sciencedirect.com/science/article/pii/S1361841517301135.

[6] Diego Riquelme and Moulay Akhloufi. "Deep Learning for Lung Cancer Nodules Detection and Classification in CT Scans". In: *AI* 1 (Jan. 2020), pp. 28–67. DOI: 10.3390/ai1010003.

[7] K. Hinkelmann. *Hyperparameter Optimization*. https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d. (Visited on 09/21/2020).

[8] *Homepage of the LNDb grand challenge 2019/20*. https://lndb.grand-challenge.org/Home/. (Visited on 08/11/2020).

[9] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[10] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Second Edition. Pearson Education, 1999. ISBN: 978-0132733502.

[11] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

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

[13] K Hinkelmann. *Lecture on Neural Networks*. http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf. (Visited on 03/02/2020).

[14] Chigozie Nwankpa et al. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. arXiv: 1811.03378 [cs.LG].

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323. URL: `http://proceedings.mlr.press/v15/glorot11a.html`.

[16] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015. arXiv: `1505.00853 [cs.LG]`.

[17] Wikimedia Commons. *Error surface of a linear neuron with two input weights*. File: `Error surface of a linear neuron with two input weights.png`. Feb. 2013. URL: `https://commons.wikimedia.org/wiki/File:Error_surface_of_a_linear_neuron_with_two_input_weights.png`.

[18] *Stochastic Gradient Descent- A Super Easy Complete Guide!* Apr. 2019. URL: `https://www.mltut.com/stochastic-gradient-descent-a-super-easy-complete-guide/`.

[19] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.

[20] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: `1212.5701 [cs.LG]`.

[21] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: `1412.6980 [cs.LG]`.

[22] Dami Choi et al. *On Empirical Comparisons of Optimizers for Deep Learning*. 2020. URL: `https://openreview.net/forum?id=HygrAR4tPS`.

[23] Guodong Zhang et al. *Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model*. 2019. arXiv: `1907.04164 [cs.LG]`.

[24] Satyam Kumar. *Overview of various Optimizers in Neural Networks*. `https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5`. (Visited on 09/14/2020).

[25] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: `1207.0580 [cs.NE]`.

[26] *Overfitting and underfitting*. `https://www.educative.io/edpresso/overfitting-and-underfitting`. (Visited on 09/12/2020).

[27] Anh Vo. *Dot-product between Filter and Input*. Feb. 2018. URL: `https://subscription.packtpub.com/book/game_development/9781789138139/4/ch04lvl1sec31/convolutional-neural-networks`.

[28] Tyrone Carlisle Nowell. "Detection and Quantification of Rot in Harvested Trees using Convolutional Neural Networks". MA thesis. Norwegian University of Life Sciences, Ås, 2019.

[29] Shuihua Wang et al. "Multiple Sclerosis Identification by 14-Layer Convolutional Neural Network With Batch Normalization, Dropout, and Stochastic Pooling". In: *Frontiers in Neuroscience* 12 (Nov. 2018), p. 818. DOI: 10.3389/fnins.2018.00818.

[30] Wikimedia Commons. *Objects detected with OpenCV's Deep Neural Network module (dnn). Reading a network model stored in Darknet model files.It uses a YOLOv3 model trained on COCO dataset capable of detecting 80 common objects in context.* Jan. 2019. URL: https://commons.wikimedia.org/wiki/File:Detected-with-YOLO--Schreibtisch-mit-Objekten.jpg.

[31] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[32] Ritesh Kanjee. *YOLOv4 — Superior, Faster & More Accurate Object Detection*. https://medium.com/@riteshkanjee/yolov4-superior-faster-more-accurate-object-detection-7e8194bf1872. (Visited on 09/14/2020).

[33] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[34] Amar Hekalo. *Lecture notes in Programming with Neural Networks*. June 2020.

[35] Zhong Chen, Ting Zhang, and Chao Ouyang. "End-to-End Airplane Detection Using Transfer Learning in Remote Sensing Images". In: *Remote Sensing* 10 (Jan. 2018), p. 139. DOI: 10.3390/rs10010139.

[36] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *Ann. Math. Statist.* 35.1 (Mar. 1964), pp. 73–101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[37] Qianhui Luo et al. *3D-SSD: Learning Hierarchical Features from RGB-D Images for Amodal 3D Object Detection*. 2017. arXiv: 1711.00238 [cs.CV].

[38] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640 [cs.CV].

[39] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[40] Martin Simon et al. *Complex-YOLO: Real-time 3D Object Detection on Point Clouds*. 2018. arXiv: 1803.06199 [cs.CV].

[41] Jasper Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[42] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. arXiv: 1311.2524 [cs.CV].

[43] Tomasz Grel. *Illustration of an ROI-pooling operation*. https://deepsense.ai/region-of-interest-pooling-explained/. (Visited on 08/14/2020).

[44] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[45] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: 1506.01497 [cs.CV].

[46] R. Girschick. *voc-dpm*. https://github.com/pierluigiferrari/ssd_keras. 2012.

[47] A. Rosebrock. *Non-Maximum Suppression for Object Detection in Python*. https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/. (Visited on 09/21/2020).

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[49] Fonova. "3D Deep Convolution Neural Network Application in Lung Nodule Detection on CT Images". LUNA 2016 challenge submission paper. URL: https://grand-challenge-public.s3.amazonaws.com/f/challenge/71/bea787d4-5cb3-4669-a48b-caa0a3048d66/20171128_034629_LUNA16FONOVACAD_NDET.pdf (visited on 08/16/2020).

[50] "3DCNN for Lung Nodule Detection And False Positive Reduction". LUNA 2016 challenge submission paper. URL: https://grand-challenge-public.s3.amazonaws.com/f/challenge/71/8ac994bc-9951-420d-a7e5-21050c5b4132/20180102_081812_PAtech_NDET.pdf (visited on 08/16/2020).

[51] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2016. arXiv: 1612.03144 [cs.CV].

[52] Sik-Ho Tsang. *Review: FPN — Feature Pyramid Network (Object Detection)*. https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610. (Visited on 08/23/2020).

[53] David Powers and Ailab. "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *J. Mach. Learn. Technol* 2 (Jan. 2011), pp. 2229–3981. DOI: 10.9735/2229-3981.

[54] Hongge Chen. "Novel machine learning approaches for modeling variations in semiconductor manufacturing". PhD thesis. Jan. 2017.

[55] Peter Flach. "The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC Isometrics". In: vol. 1. Jan. 2003, pp. 194–201.

[56] Peter Flach. *Tutorial on "The Many Faces of ROC Analysis in Machine Learning"*. http://people.cs.bris.ac.uk/~flach/ICML04tutorial//. (Visited on 08/13/2020).

[57] Harold Miller. "The FROC Curve: A Representation of the Observer's Performance for the Method of Free Response". In: *The Journal of the Acoustical Society of America* 46 (1969). DOI: 10.1121/1.1911889.

[58] Arnau Oliver. *FROC Analysis*. http://eia.udg.edu/~aoliver/publications/tesi/node147.html. (Visited on 08/13/2020).

[59] Ke Yan, Mohammadhadi Bagheri, and Ronald Summers. "3D Context Enhanced Region-Based Convolutional Neural Network for End-to-End Lesion Detection". In: Sept. 2018, pp. 511–519. DOI: 10.1007/978-3-030-00928-1_58.

[60] *Homepage of the ANODE09 grand challenge*. https://anode09.grand-challenge.org/. (Visited on 08/11/2020).

[61] *Homepage of the Kaggle Data Science Bowl*. https://www.kaggle.com/c/data-science-bowl-2017. (Visited on 08/12/2020).

[62] *Homepage of the LUng Nodule Analysis 2016 grand challenge*. https://luna16.grand-challenge.org/. (Visited on 08/11/2020).

[63] *Homepage of the LUNGx Challenge 2015*. https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI. (Visited on 08/11/2020).

[64] *Summary Of The National Lung Screening Trial NLST*. https://cdas.cancer.gov/learn/nlst/trial-summary. (Visited on 08/11/2020).

[65] Bram van Ginneken et al. "Comparing and combining algorithms for computer-aided detection of pulmonary nodules in computed tomography scans: The ANODE09 study". In: *Medical Image Analysis* 14.6 (2010), pp. 707 –722. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2010.05.005. URL: http://www.sciencedirect.com/science/article/pii/S1361841510000587.

[66] Ying Ru Zhao et al. "NELSON lung cancer screening study". eng. In: *Cancer imaging : the official publication of the International Cancer Imaging Society* 11 Spec No A.1A (Oct. 2011), S79–S84. ISSN: 1470-7330. DOI: 10.1102/1470-7330.2011.9020. URL: https://pubmed.ncbi.nlm.nih.gov/22185865.

[67] Justin S. Kirby et al. "LUNGx Challenge for computerized lung nodule classification". In: *Journal of Medical Imaging* 3.4 (2016), pp. 1 –9. DOI: 10.1117/1.JMI.3.4.044506. URL: https://doi.org/10.1117/1.JMI.3.4.044506.

[68] *The Lung Image Database Consortium image collection LIDC-IDRI*. https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI. (Visited on 08/11/2020).

[69] Michael F. McNitt-Gray et al. "The Lung Image Database Consortium (LIDC) Data Collection Process for Nodule Detection and Annotation". In: *Academic Radiology* 14.12 (2007), pp. 1464 –1474. ISSN: 1076-6332. DOI: https://doi.org/10.1016/j.acra.2007.07.021. URL: http://www.sciencedirect.com/science/article/pii/S1076633207004497.

[70] Arnaud Setio et al. "Pulmonary Nodule Detection in CT Images: False Positive Reduction Using Multi-View Convolutional Networks". In: *IEEE Transactions on Medical Imaging* 35 (Mar. 2016), pp. 1–1. DOI: 10.1109/TMI.2016.2536809.

[71] Jia Ding et al. *Accurate Pulmonary Nodule Detection in Computed Tomography Images Using Deep Convolutional Neural Networks*. 2017. arXiv: `1706.04303 [cs.CV]`.

[72] Guohua Cheng et al. "Deep Convolution Neural Networks for Pulmonary Nodule Detection in CT imaging". In: 2018. URL: `https://grand-challenge-public.s3.amazonaws.com/f/challenge/71/101b150c-88c5-4f9d-a374-d8d3fc166aff/20171222_073722_JianpeiCAD_NDET.pdf` (visited on 08/16/2020).

[73] Jiaxu Ning et al. "A Computer-Aided Detection System for the Detection of Lung Nodules Based on 3D-ResNet". In: *Applied Sciences* 9 (Dec. 2019), p. 5544. DOI: `10.3390/app9245544`.

[74] Ross Gruetzemacher, Ashish Gupta, and David B. Paradice. "3D deep learning for detecting pulmonary nodules in CT scans". In: *Journal of the American Medical Informatics Association* 25 (2018), 1301–1310.

[75] Qi Dou et al. *Automated Pulmonary Nodule Detection via 3D ConvNets with Online Sample Filtering and Hybrid-Loss Residual Learning*. 2017. arXiv: `1708.03867 [cs.CV]`.

[76] Gao Huang et al. *Densely Connected Convolutional Networks*. 2016. arXiv: `1608.06993 [cs.CV]`.

[77] Fangzhou Liao et al. "Evaluate the Malignancy of Pulmonary Nodules Using the 3-D Deep Leaky Noisy-OR Network". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.11 (Nov. 2019), 3484–3495. ISSN: 2162-2388. DOI: `10.1109/tnnls.2019.2892409`. URL: `http://dx.doi.org/10.1109/TNNLS.2019.2892409`.

[78] Wenkai Huang, Yihao Xue, and Yu Wu. "A CAD system for pulmonary nodule prediction based on deep three-dimensional convolutional neural networks and ensemble learning". In: *PLOS ONE* 14.7 (July 2019). DOI: `10.1371/journal.pone.0219369`.

[79] Daniel Hammack. "Forecasting Lung Cancer Diagnoses with Deep Learning". Data Science Bowl 2017 Technical report. Apr. 2017.

[80] Julian de Wit. *2nd place solution for the 2017 national datascience bowl*. `http://juliandewit.github.io/kaggle-ndsb2017/`. (Visited on 08/16/2020).

[81] Du Tran et al. *Learning Spatiotemporal Features with 3D Convolutional Networks*. 2014. arXiv: `1412.0767 [cs.CV]`.

[82] Ning Zhang et al. *3D Aggregated Faster R-CNN for General Lesion Detection*. 2020. arXiv: `2001.11071 [cs.CV]`.

[83] Junjie Zhang et al. "NODULe: Combining constrained multi-scale LoG filters with densely dilated 3D deep convolutional neural network for pulmonary nodule detection". In: *Neurocomputing* 317 (2018), pp. 159 –167. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2018.08.022`. URL: `http://www.sciencedirect.com/science/article/pii/S0925231218309378`.

[84] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2015. arXiv: `1511.07122 [cs.CV]`.

[85] *Ali Tianchi dataset*. `https://tianchi.aliyun.com/competition/entrance/231601/information`. (Visited on 09/15/2020).

[86] Jason Brownlee. *Best Practices for Preparing and Augmenting Image Data for CNNs*. `https://machinelearningmastery.com/best-practices-for-preparing-and-augmenting-image-data-for-convolutional-neural-networks/`. (Visited on 08/20/2020).

[87] Tami D. DenOtter and Johanna Schubert. *Hounsfield Unit information sheet*. `https://www.ncbi.nlm.nih.gov/books/NBK547721/`. (Visited on 09/20/2020).

[88] *Tutorial on Data Augmentation with Tensorflow*. `https://www.tensorflow.org/tutorials/images/data_augmentation`. (Visited on 09/20/2020).

[89] Lei Mao. *Bounding Box Encoding and Decoding in Object Detection*. `https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610`. (Visited on 09/21/2020).

[90] *Discussion Thread on Variances in the Priorbox Layer*. `https://github.com/weiliu89/caffe/issues/155`. (Visited on 09/21/2020).

[91] P. Ferrari. *Keras SSD v0.9.0*. `https://github.com/pierluigiferrari/ssd_keras`. 2018.

[92] *Tensorflow documentation of the ReduceLROnPlateau callback*. `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau`. (Visited on 09/21/2020).

[93] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[94] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[95] *Tensorflow documentation of the Callback class*. `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback`. (Visited on 09/27/2020).

[96] *GitLab repository of the introduced framework*. `https://gitlab2.informatik.uni-wuerzburg.de/ext702819/lungnoduledetection`. (Visited on 09/27/2020).

[97] *Train/Validation Ranking of the LNDb grand challenge 2019/20*. `https://lndb.grand-challenge.org/TrainValidationRanking/`. (Visited on 09/21/2020).

[98] *Test Ranking of the LNDb grand challenge 2019/20*. https://lndb.grand-challenge.org/TestRanking/. (Visited on 09/21/2020).

[99] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2017. arXiv: 1708.02002 [cs.CV].

[100] Navaneeth Bodla et al. *Soft-NMS – Improving Object Detection With One Line of Code*. 2017. arXiv: 1704.04503 [cs.CV].

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

$Unterschrift :$ $Ort, Datum :$