EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-
Naturwissenschaftliche
Fakultät

Bachelorarbeit

Acetabulum fracture classification on a large cohort of CT images from German hospitals using 3D convolutional neural networks

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Methods in Medical Informatics
Daniel Wilfried Dauner, daniel-wilfried.dauner@student.uni-tuebingen.de, 2021

Bearbeitungszeitraum: 01.10.2020 - 01.02.2021

Betreuer/Gutachter: Prof. Dr. Nico Pfeifer
Zweitgutachter: -

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Daniel Wilfried Dauner (Matrikelnummer 4182694), January 29, 2021

Abstract

Fractures of the acetabulum are complex injuries of the pelvic bone. Preoperative diagnostics and adequate surgical intervention are crucial in order to anatomically reconstruct the joint. The Letournel system categorizes the fractures into 10 distinct classes which allows for standardisation of the treatment. Especially clinicians with little experience have difficulties in correctly detecting and diagnosing acetabulum fractures.
In this thesis, Convolutional Neural Networks (CNNs) are applied to offer a computer-aided approach for the diagnosis of Letournel fractures. The models are trained on a large cohort of CT scans from hospitals across Germany.
Three CNNs were created to detect acetabulum fractures, classify their surgical access, and to determine their Letournel class. The fracture detection model achieved an accuracy of 93.7%. The Letournel classification was able to diagnose the fractures with a weighted F1-score of 57.3%. Furthermore, the Letournel classification achieved a weighted F1-score of 85.7% for the prediction of the surgical access.

Acknowledgments

First, I want to thank Ralf Eggeling for making this thesis possible and for the constant support throughout the development. I would like to thank Daniel Dehncke for the technical assistance in the beginning of this project.
Next, I want to thank Markus Küper and Felix Erne for helping me with the annotation of the data. Your input helped me to understand the medical perspective of this subject.
Furthermore, I want to thank my girlfriend Amy Bland, who always supported me and improved the readability of this thesis with her feedback.

Contents

1. Introduction

The acetabulum is the socket of the pelvic bone. The femoral head articulates with the pelvis at the acetabulum, which forms the hip-joint, as shown in Figure 1.1. Fractures of the acetabulum mostly occur during a trauma where the femur applies high force to the pelvic bone [1]. These fractures are among the most complex injuries treated by orthopedic surgeons [2]. Patients may suffer from disability or permanent damage if the acetabulum fractures are not treated correctly.

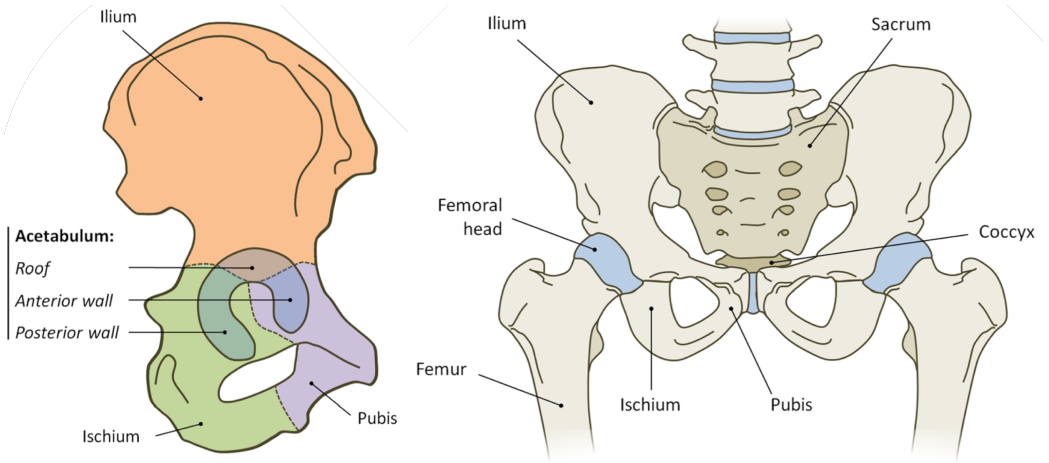

Figure 1.1.: Anatomy of the pelvis (from [3]). The pelvis bone is shown on the left. The ventral view of the hip region is shown on the right. https://creativecommons.org/licenses/by/4.0/

The goal of therapy is the anatomical reconstruction of the joint [4]. An established classification of acetabulum fractures is the Letournel system. This allows for the standardization of treatment options and necessary surgical procedures. The fractures are categorized into 10 classes, as shown in Figure 1.2. Originally Letournel proposed two main groups of the fractures. The elementary group has a single fracture line through the acetabulum, whereas the associated group is characterized by multiple elementary fractures. However, the diagnosed Letournel type also determines the access for surgical intervention [5], and are grouped accordingly in this thesis. Acetabulum fractures are either accessed anterior, posterior, or in combination when required.

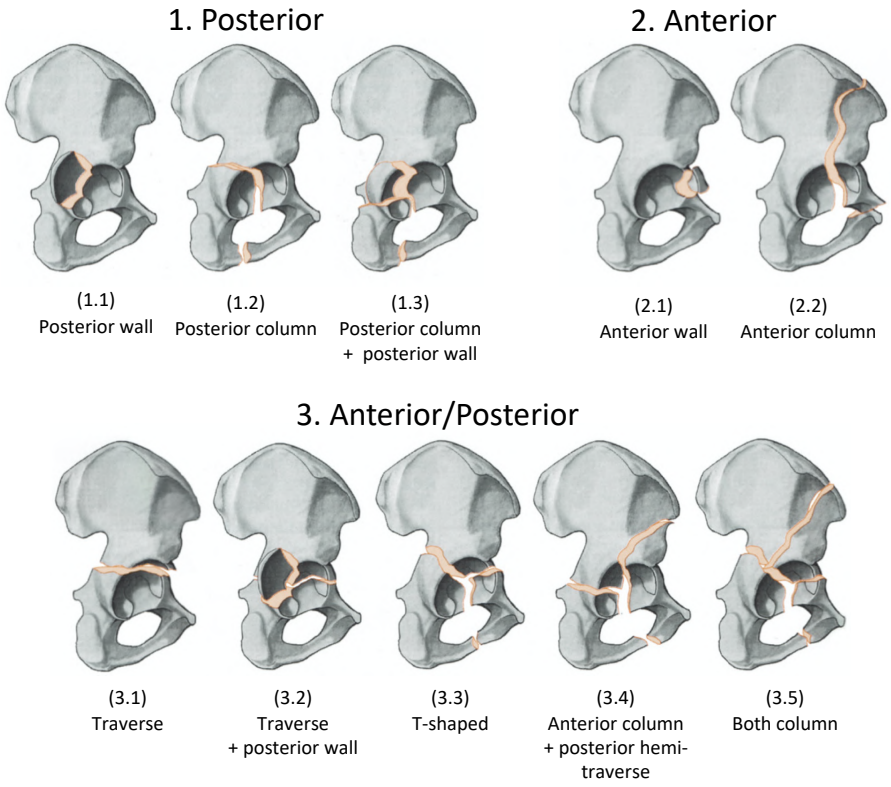

Figure 1.2.: Overview of the 10 Letournel fracture types (adapted from [6]). The fractures are grouped according to their surgical access.

Qualitative medical imaging, like Computer Tomography (CT), is necessary in order to visualize the complex structure of the joint. A CT scanner has an X-ray source and detector, that rotate around the patient [7]. A computer can generate cross sectional images of the body. CT analysis from the axial plane made the correct classification of acetabulum fractures easier, compared to conventional X-Ray images [8]. The introduction of multi-planar reconstruction from the sagittal and coronal plane further extended the visualization options.

Nevertheless, these fractures are notably difficult to correctly detect and classify, especially in the case where the physician lacks extensive experience [8]. While clinician have a variety of laboratory tests, the correct diagnosis of acetabulum fractures relies on the examination and interpretation of medical imaging. Advanced methods in Machine Learning can be applied to medical image analysis [9], to support clinicians in their decision making. A computer-aided approach could improve the patients care by providing the most suitable diagnosis.

Machine Learning is the study of developing methods that enable computers to

solve problems by learning from experience. A mathematical model can learn to solve a task for training data and apply the knowledge to unseen data. Machine Learning can be divided into several sub-categories, in which Deep Learning has recently garnered a lot of attention.
Deep Learning is based on Neural Networks, that are parameterized networks of several artificial neurons. The neurons are grouped into multiple interconnected layers, which can transform input data (e.g. images) into an output (e.g. predicted class label). By changing the parameters of the connections and units, the model can learn patterns and relationships in the data. Nowadays, Neural Networks form the state-of-the-art approach for complex tasks. These include language modeling [10], protein-structure prediction [11], or computer vision [12].

In computer vision, the most successful variant is the Convolutional Neural Network (CNN). By transforming an input image with convolutional filters in many consecutive layers, CNNs can extract high-level features. This approach was first introduced in the early eighties [13], but recent development and training on Graphics Processing Units (GPUs) made it possible to train deeper networks. In 2012 the "AlexNet" won the ImageNet competition with five convolutional layers [14]. The large ImageNet data set consist of 1.2 million images that are classified in 1000 categories [15]. Networks became deeper and more powerful, for example "ResNet" won the ImageNet competition in 2015 with 152 layers [16].
For medical images, CNNs are applied in fields such as brain tumor segmentation [17], breast cancer classification [18], or Alzheimer classification [19]. However, CNNs need an enormous amount of training data in order to make accurate predictions. The availability of medical images is often restricted due to privacy protection laws or rare deceases.

Daniel Dehncke developed a computer-aided method for the diagnosis of acetabulum fractures in his master's thesis [20]. The BG Unfallklinik Tübingen provided the data of 222 patients, classified according to the Letournel system. A pipeline was developed to evaluate and select useful CT scans. In the study, 3D CNNs were utilized to detect fractures and to classify the fractures into three categories. These categories correspond to the surgical access and are shown in Figure 1.2.
The work showed a promising approach to apply 3D CNNs for acetabulum fractures. However, the results have certain limitations. Only a small data set of single health institution was available, which restricted the classification on all 10 Letournel classes. The CT scans were only classified with the ResNet architecture and it was unclear how the models detected the fractures.

Since then, 11 health institutions across Germany provided data that was collected over several decades. The aim of this thesis is further improve the diagnostic performance of 3D CNNs with the large and diverse data set. A survey of modern

CNN architectures is conducted. These include ResNet, ResNeXt, EfficientNet, and the Inflated 3D ConvNet (I3D). The predictions are visualized with attention maps, to inspect the decision-making. The models are trained to detect fractures and to classify them into the same three categories of their surgical access. Finally, the models are trained for the classification on all 10 Letournel classes.

This rest of this thesis is structured as follows: in Chapter 2 the theoretical background for understanding Deep Learning in computer vision is introduced, in Chapter 3 the methods are explained, in Chapter 4 the results of the experiments are presented, and in discussed in Chapter 5.

2. Background

This chapter is influenced by the textbook of Goodfellow et al. [21], which offers more detailed information about the broad field of Deep Learning. In Section 2.1, the basic concepts of Neural Networks are introduced. The Convolutional Neural Network is presented in Section 2.2 and several architectures are discussed, which are used in this thesis. Furthermore, two approaches are explained in Section 2.3 which offer a visual explanation of the prediction of Convolutional Neural Networks.

2.1. Introduction to Neural Networks

Neural Networks are a group of Machine Learning models. This Section explains their components and functionality of Neural Networks. Algorithms for training and optimization are introduced, together with several techniques to further improve the prediction.

2.1.1. The Neural Network

The Neural Network consists of several interconnected units. These are called neurons and an example is shown in Figure 2.1. The neurons form a parameterized network and are loosely inspired by biological neurons [21].
Each neuron i receives multiple inputs $x_i = [x_{i1}, .., x_{in}]$, which are weighted by a factor $w_i = [w_{i1}, .., w_{in}]$. The input is processed by calculating the weighted sum of

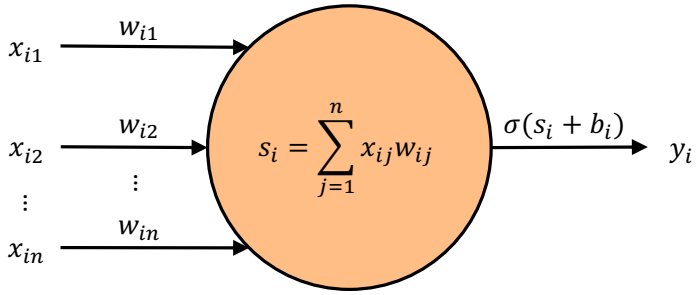

Figure 2.1.: The structure of a neuron i. The neuron receives an input x_i and calculates the weighted sum s_i. A bias b_i is added to s_i and an activation function σ is applied.

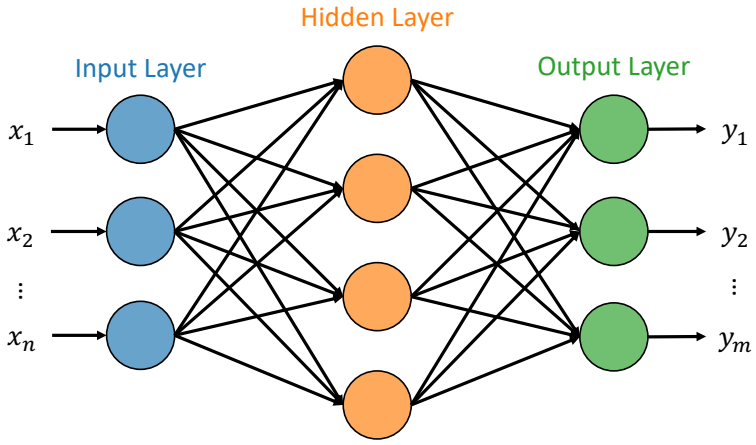

Figure 2.2.: Example of a feed-forward network, with an input layer, hidden layer and an output layer.

the input $s_i = \sum_{j=0}^{n} w_{ij} x_{ij}$. When desired, a node can add a bias b_i to the weighted sum s_i. Before the neuron outputs a value, a (usually non-linear) activation function σ is applied to the weighted sum and bias.
The topology of the network is called an architecture. In a layered architecture the neurons are be grouped in layers, as shown in Figure 2.2. When the layers are connected in between, they are called linear layers or fully connected (FC) layers. The neurons and connections mostly do not form a cycle in the graph. This is called a feed-forward network.
The first and last layers of the network are called the input and output layer, respectively. Furthermore, the network can have any number of fully connected layers in between which are called hidden layers.

The neural network calculates a function $f(\mathbf{x})$ for a data point with the feature vector $\mathbf{x}$. In the forward pass, the vector $\mathbf{x}$ is given to the input layer. In each layer, the neurons process the received input, apply an activation function and pass the output to the successive layer. At the output layer, the network returns the function $f(\mathbf{x}) = \hat{\mathbf{y}}$, which is a prediction for the target vector $\mathbf{y}$.
The goal during training is to adjust the weights and biases of the neurons, so that $f(\mathbf{x}) \approx \mathbf{y}$ for each sample in the training set. For classification tasks, the number of output nodes often coincide with the number of possible classes, where each output is an estimated likelihood.

2.1.2. Activation Functions

The activation functions in the neurons add non-linearity into the network. As a results, the Neural Network can learn more complex patterns and relationships [22]. The Sigmoid function, also called Logistic function, is a common activation function for the output neurons. It is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

where x is the weighted sum and bias of the neuron. Sigmoid is often used to compute the class probability for binary classification, because the value of the function is in the range of 0 to 1.
In multi-class classification, the probability distribution in the output layer is mainly computed with the Softmax function. Softmax is an activation function, which takes a K-dimensional feature vector x and returns a probability for each class $i = 1,...,K$ that sum up to 1.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.2}$$

One of the most widely used activation functions, is the Rectified Linear Unit (ReLU) function. Compared to Sigmoid, it is easier to optimize, it can lead to better performance, and can be calculated more efficiently.

$$ReLU(x) = max(0,x) \tag{2.3}$$

The ReLu function is mainly used in the hidden layers of the network.

2.1.3. Training & Optimization

In order to make accurate predictions, the neural network needs a set of parameters that produce the right activation on the output layer. The neural network has a defined loss function that evaluates how well the model performs for a given input. The goal of the training is to minimize the loss for the training samples. This is done by changing the parameter values until the model makes accurate predictions.

Cross Entropy Loss

The cross-entropy loss measures the performance of a classification model that outputs probabilities for each possible class. Cross-entropy quantifies the difference between the true and predicted class distribution.

$$L_i(\theta) = -\sum_{c=1}^{M} y_{i,c} log(\hat{y}_{i,c}(\theta)) \tag{2.4}$$

M is the number of classes, log is the natural logarithm, $\hat{y}_{i,c}(\theta)$ is the predicted probability that an observation i is in class c, for a given parameter set θ. The true probability $y_{i,c}$ is either 0 or 1.
A class specific factor, called weight, can be added to punish certain classifications differently.

$$L_i(\theta) = -\sum_{c=1}^{M} weight(c)[y_{i,c} \cdot log(\hat{y}_{i,c}(\theta))] \tag{2.5}$$

This can be useful when a correct classification of a class is important or when the data set is unbalanced.

The loss function only measures the performance of a single observation i from the training set. The model needs to find parameters θ that minimize the loss for every sample. This can be done by minimizing a cost or objective function $J(\theta)$ which, for example, calculates the average loss of the training set for the parameters θ.

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\theta) \tag{2.6}$$

Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) is a commonly used algorithm to minimize the objective function. SGD is an extension of the Gradient Descent algorithm.
The Gradient Descent algorithm updates the weights and biases iteratively to reduce the objective function. The algorithm computes a set of partial derivatives $\nabla_\theta J(\theta)$, which is called the gradient. Intuitively, the gradient indicates the direction the weights and biases have to be changed, in order to minimize the objective function J. In an epoch of the training, the parameters are updated as follows:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta) = \theta - \frac{\eta}{n}\sum_{i=1}^{n} \nabla_\theta L_i(\theta) \tag{2.7}$$

where n is the number of observation's i. η is called the learning rate and determines the step size in each iteration.
Gradient descent can be slow, due to the expensive calculation of the gradients for the complete training set. SGD approximately estimates the true gradient of $J(\theta)$ using a small subset of samples, called a batch. In each epoch, the weights are updated multiple times for each batch:

$$\theta \leftarrow \theta - \frac{\eta}{n'}\nabla_\theta \sum_{i=1}^{n'} L_i(\theta) \tag{2.8}$$

where n' is the batch size. The original SGD has a batch size of one. Mini-batch SGD has a batch size greater than one. However, both variants are commonly referred as

SGD.

A common technique to accelerate the training, is to add momentum to SGD. Momentum has a velocity term v which combines the gradients of the previous iteration with the current gradients. The parameter update is given by:

$$\begin{aligned} v &\leftarrow \alpha v - \frac{\eta}{n'} \nabla_\theta \sum_{i=1}^{n'} L_i(\theta) \\ \theta &\leftarrow \theta + v \end{aligned} \tag{2.9}$$

where $\alpha \in [0,1)$ is a hyper-parameter, that determines the decay of the previous gradients.

Optimizer like SGD use the back-propagation algorithm. Back-propagation is often misunderstood as a learning algorithm itself but only refers to the computation of the gradients. The algorithm iterates backwards from the last layers, which is called the backwards pass. The gradient of the loss with respect to each parameter are calculated, using the chain rule. The back-propagation algorithm is highly efficient and enables optimizer like SGD to train large Neural Networks.

2.1.4. Regularization

Neural Networks are complex structures, sometimes using several million parameters for the prediction. Often the model overfits, which means it tries to explain the data too closely, by learning irrelevant patterns (e.g. noise). As a result, the model fails to generalize the data and to give reliable predictions outside the training set.
In a broader sense, regularization is a process that hinders a machine learning model from overfitting. The aim is to reduce the generalization error and to find a more optimal parameter configuration.

L^2 Regularization

Many regularization methods constrain the model parameters θ, by adding a norm penalty $\Omega(\theta)$ to the objective function $J(\theta)$. The approach is to limit the capacity of the model.

$$\tilde{J}(\theta) = J(\theta) + \alpha \cdot \Omega(\theta) \tag{2.10}$$

where $\tilde{J}$ is the regularized objective function and $\alpha \in [0, \infty)$ a hyper-parameter, which determines the intensity of the weight penalty.
A common norm penalty is the L^2 regularization, defined as followed:

$$\Omega(\theta) = \frac{1}{2} \|w\|_2^2 \tag{2.11}$$

where w are the weights of the model. Norm penalties usually do not affect the biases of the network. L^2 regularization is also known as weight decay or ridge regression. Intuitively, by penalizing the weights, the model can learn less specific patterns. Therefore, the model improves in generalization performance.

Dropout

A simple, yet effective regularization technique is dropout. In each epoch, a random set of neurons in a layer are omitted. The probability in a layer for a neurons to "drop-out" is an additional hyper-parameter. Dropout is only applied during training and hinders a unit from relying too much on other units [23].

Data Augmentation

The best way to reduce the generalization error of a model is to add more data. In practice, the amount of data is limited. A solution for this problem is to generate fake data. This technique is particularly useful when classifying images, because of the enormous variety of operations that can be applied without changing the basic content.
Typical augmentation techniques on images are zooming, rotation, and translation. The model can learn the same features, but in different sizes and from several angles. However, it is essential that the class of the input remains the same. For example, when a letter 'd' is flipped horizontally, it results in the letter 'b'. This would be an inappropriate operation for text recognition.

2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special kind of Neural Networks for data with a grid-like topology. For example, an image can be described as a matrix, where each pixel is represented with RGB values or a grey-scale value. In a traditional feed-forward network, the layers are fully connected with separate parameters describing an interaction between input and output neurons. An image can have millions of pixels, which would lead to an extensive number of parameters. CNNs solve this problem with the use of a mathematical operation called convolution. CNNs have been successfully applied to image classification [16], video recognition [24], and in medical image analysis [9].
This section offers a introduction into CNNs and their components. Furthermore, several CNN architectures are introduced, which were used in this thesis.

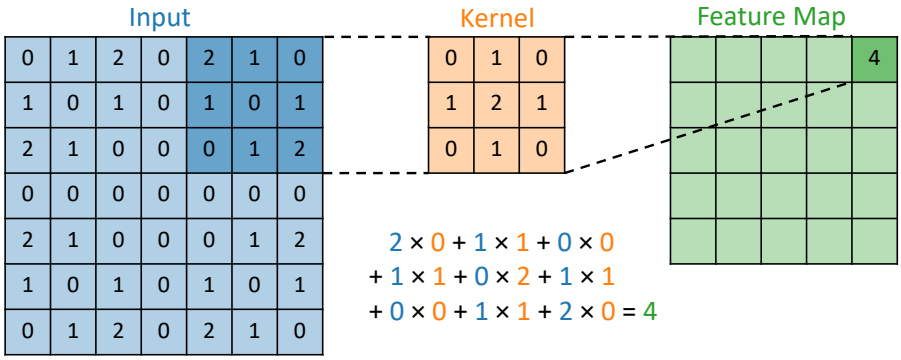

Figure 2.3.: Example of a 2D convolution operation. The kernel moves over the input map, calculates the inner product and fills out the feature map. In this example, the input and output channel is one.

2.2.1. Convolution & Pooling

In general, a convolution is a mathematical operation that combines two functions [21]. In CNNs, the first argument is called the input and the second argument the kernel. The output of the convolution is referred to as feature map.
If a 2D input image is given, the CNN can slide with the kernel matrix over the input in x and y direction. For the overlapping values, the inner product is calculated and forms a value in the feature map, as shown in Figure 2.3. The kernel size depends on the architecture and the weights of the kernel are adjusted during training. The kernel extracts essential features, like edges or lines.

A kernel is usually smaller than the input image and has less parameters than a fully connected layer between the input and output. Therefore, the layers in a CNN are locally or sparsely connected. This improves the memory requirements and enables a CNN to efficiently process large inputs, like images.

Furthermore, a convolution has a defined number of input feature maps, or input channels. For example, a RGB image has three channels and a grey scale image only one. A convolution produces several output feature maps, or output channels. For each output feature map, the convolution has a distinct kernel that moves across all the input feature maps. The resulting feature maps are summed up element-wise to produce the kernel specific output feature map [25].

A typical layer of a CNN consist of three stages. First, a convolutional layer generates several feature maps. In the second stage, a non-linear activation function, often ReLU, is applied. Lastly, a pooling function is used to modify the output.

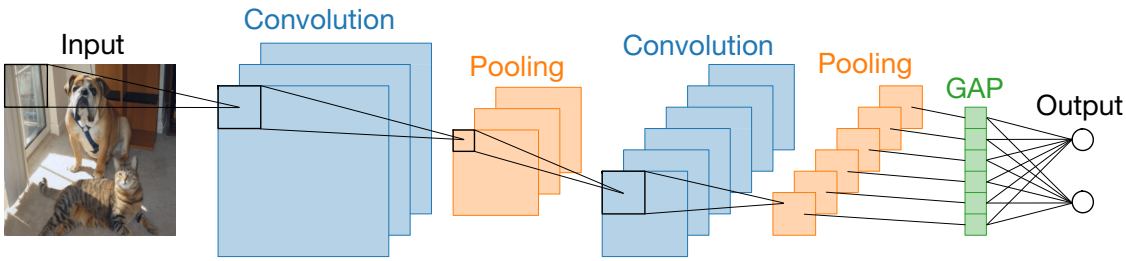

Figure 2.5.: Structure of a CNN with multiple convolution and pooling operations. Each convolution produces several output feature maps with a specific kernel. The last layers consist of Global Average Pooling (GAP) and a linear output layer. (Image from [28], inspired by [29])

A pooling function filters the input feature map, by removing unnecessary information. As a result, pooling can have a regularizing effect. Similar to the convolution, the pooling operation moves with rectangular kernel over the input. A typical operation is max pooling, where only the maximum value of the kernel is passed to the next layer. If the average value is passed, the operation is called average pooling, as shown in Figure 2.4.

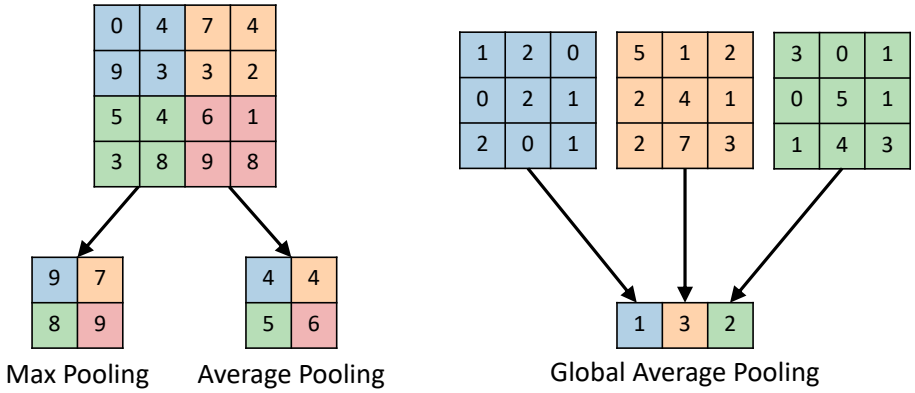

Figure 2.4.: Examples of Max Pooling, Average Pooling and Global Average Pooling (GAP). GAP has multiple input feature maps.

Global Average Pooling (GAP) is a pooling operation that calculates the average of a whole feature map. GAP is mostly applied on multiple feature maps and therefore, returns a one dimensional vector. GAP sums out spatial information, which means the network is more robust to spatial translation [26]. In some CNN architectures, GAP is applied after the last convolution layer [16][27]. A linear layer with softmax is connected, to return the probabilities for each class.

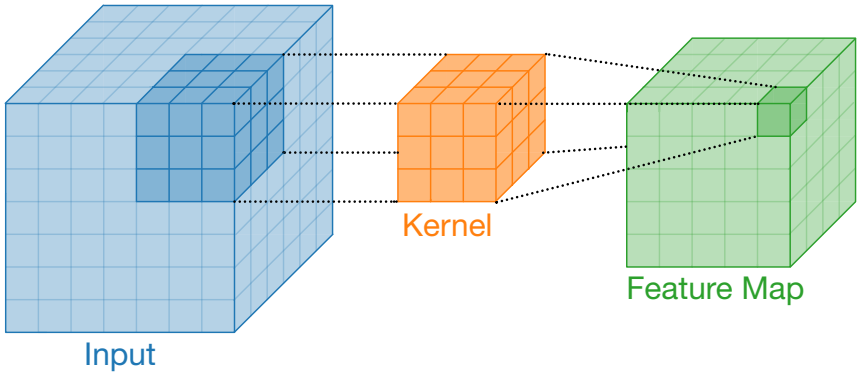

Figure 2.6.: The convolution operation extended to a third dimension. The 3D kernel moves over the input and computes the output feature map. The input and output channel is one.

Typically a CNN architecture has several convolution and pooling layers, combined with activation functions. They are often hidden layers and successively applied to extract more complex features deeper in the network, as shown in Figure 2.5.

2D CNNs have been tremendously useful for predictions on images. On the other hand, there are real-world applications were the data has a 3-dimensional shape. For example, a video consisting of several images can be stacked together, resulting in a 3D volume.
In order to extract 3D features, the operations of CNN can be extended to a third dimension. Intuitively, the input and output feature maps, as well as the kernel have a cuboid shape. For the convolution, the kernel is moved over the input on the x, y and z axis, producing a 3D feature map, as shown in Figure 2.6.

2.2.2. CNN Architectures

The learning ability of 2D CNNs improved significantly over the years with the development of advanced architectures [30]. Meanwhile, successful 3D CNNs mostly emerged from 2D architectures. ResNet, ResNext and EfficientNet are originally 2D CNNs for image classification [16, 31, 27]. The I3D architecture was built action recognition but influenced by the 2D Inception-V1 architecture [24, 32].

ResNet & ResNeXt

The number of features a CNN can learn can be increased with the number of layers of the network. However, when adding too many convolutional layers, the performance will decrease eventually. One problem is, that the gradients of the error function, with respect to the weights, get too small. This hinders the optimizers

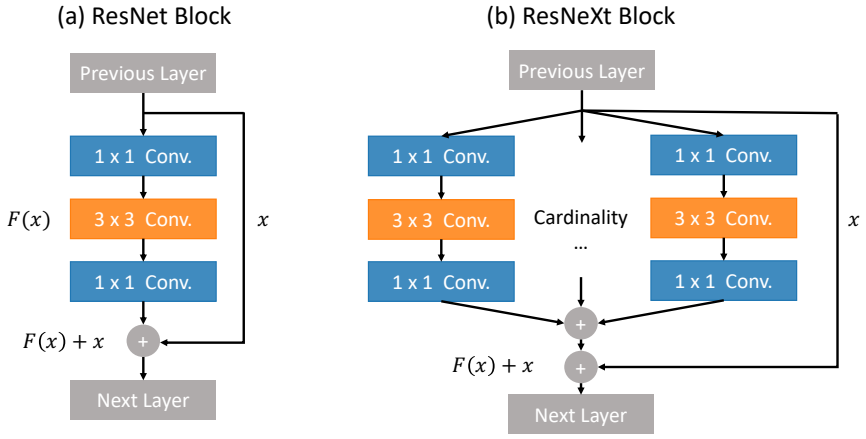

Figure 2.7.: Bottleneck blocks of (a) ResNet and (b) ResNeXt. Batch normalization and ReLU activation function are not shown.

ability to adjust the weights and train the layers. This effect is called vanishing gradient problem [16, 21].
However, a deep neural network should be equally as powerful as its shallower counterpart. When the additional layers are mapping the identity function and the earlier layers are copied, the deeper network would have an equal performance.
The ResNet architecture solves this by construction [16]. ResNet has "shortcut connections" between the layers, which add the identity function on the output of the stacked layers, as shown in Figure 2.7. The layers can learn to adjust the identity function when it is beneficial.

The ResNet architecture makes it possible to train CNNs with thousands of layers, while still improving in performance. This concept led to many evolved architectures, such as Wide ResNet [33], Inception-ResNet [34], or ResNeXt [31]. The ResNeXt architecture is largely similar to ResNet but has a different topology of the building blocks, as shown in Figure 2.7.
ResNeXt exploits the "split-transform-merge" strategy. The input of the building block is divided into several convolution paths and merged afterwards. The authors call the number of paths "cardinality", which is an additional hyper-parameter. Cardinality controls the number of transformations and is able to improve the classification performance.

EfficientNet

A technique to improve the performance of a CNN is to scale up the size. A CNN can be scaled up in depth, which means more convolutional layers are added. Another option is to increase the width of the network, which results in more feature maps at

each layer. Lastly, the resolution of the input image can be increased to capture finer details.
However, the process of scaling up an architecture was never well understood [27]. The parameters were often arbitrarily found or selected due to resource constraints.

The EfficientNet architecture introduced a systematic way to scale up CNNs, called "compound scaling" [27]. The concept is to uniformly scale up width, depth and resolution of the network, because the three parameters largely affect each other. Each dimension is scaled up by a parameter ϕ.

$$\text{depth} = \alpha^{\phi}, \quad \text{width} = \beta^{\phi}, \quad \text{resolution} = \gamma^{\phi}, \quad \text{s.t.} \quad \alpha \cdot \beta^{\phi} \cdot \gamma^{\phi} \approx 2 \tag{2.12}$$

where $\alpha, \beta, \gamma \geq 1$. The coefficients α, β, γ are determined with grid-search on a smaller model. The model can be scaled up, with respect to Equation 2.12, when more recourse's are available.

The EfficientNet architecture uses the mobile inverted block as main building block, first introduced in the MobileNetV2 architecture [35], together with squeeze and extraction optimization [36]. The EficientNet has less parameters in comparison to ResNet and ResNext, while achieving better results on large benchmark data sets, like ImageNet [15].

Inflated 3D (I3D)

The Inflated 3D ConvNet architecture, or I3D, was introduced as a action recognition model for videos. The original architecture consists of two separate 3D CNNs, which are trained on the RBG videos and the precomputed optical flow of each sample. Therefore, the I3D has a "two-stream" architecture. The predictions of the CNNs are averaged.
2D CNN architectures, like ResNet, have often been adapted for video classification [37]. The backbone of I3D uses the Inception-V1 architecture, but "inflates" the pooling kernels and the filters to a 3D shape, as shown in Figure 2.8.

The Inception module has a similar split-transform-merge strategy, which later inspired the design of ResNeXt [31].
Furthermore, the authors propose a method to bootstrap the weights of 2D pre-trained models. The $N \times N$ pre-trained filters can be repeated N times for the times dimension of the video. The weights are re-scaled by dividing by N, to ensure the same response. This enables the I3D model to be pre-trained on large image data sets, e.g. ImageNet.

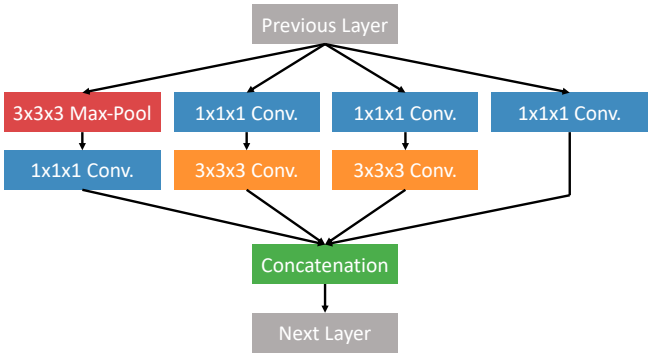

Figure 2.8.: The inflated Inception-V1 sub-module (from [24]).

2.3. Visualization Explanation

Deep Neural Networks are referred to as black box models, due to their lack of transparency [38]. The complex multi-layer structure makes the prediction untraceable for humans. There are several methods for CNNs to produce "visual explanations" of the decision-making. In this section, Grad-CAM and Guided Grad-CAM are introduced [28].

Grad-CAM

Gradient-weighted Class Activation Mapping (Grad-CAM), is a method that produces coarse localization maps. The maps show class specific activation and highlight regions that are important for the classification.

Grad-Cam utilizes the feature map activation's $A^k \in \mathbb{R}^{u \times v}$ of width u and height v, of a selected convolutional layer. The idea is to visualize the neurons of the feature map $k \in \{1, .., K\}$, which have an impact on the prediction score y^c of a class c. Therefore, Grad-CAM calculates the gradient of y^c with respect to the activation A^k of a feature map k. The gradients are global-average-pooled over the width u and height v of the map.

$$\alpha_k^c = \underbrace{\frac{1}{u \cdot v} \sum_{i=1}^{u} \sum_{j=1}^{v}}_{\text{global average pooling}} \frac{\partial y^c}{\partial A_{ij}^k} \tag{2.13}$$

The computed value α_k^c is the weight (or importance) of a feature map k for the target class c. Finally, the activation map $L^c_{\text{Grad-CAM}}$ is generated, by a weighted combination of A^k.

$$L^c_{\text{Grad-CAM}} = ReLU\left(\sum_{k=1}^{K} \alpha_k^c A^k\right) \in \mathbb{R}^{u \times v} \tag{2.14}$$

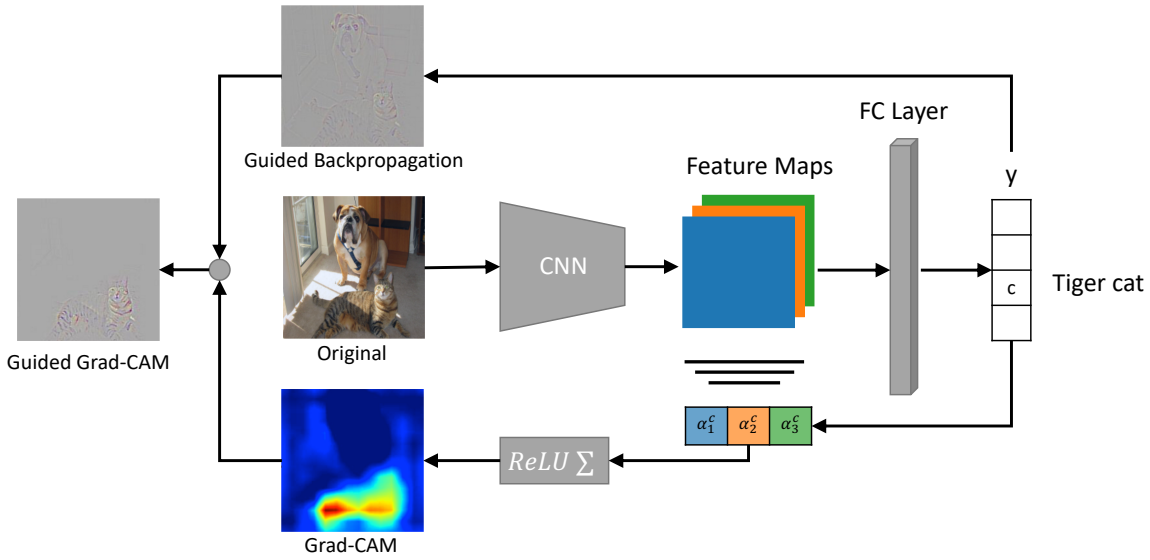

Figure 2.9.: Overview of Grad-CAM and Guided Grad-CAM, adapted from [28]. The CNN classifies an image (e.g. tiger cat). Grad-Cam back-propgates to the selected layer and combines the weighted feature maps to an activation map. Guided Back propagation can be fused, to generate Guided Grad-CAM.

The *ReLU* function is applied to only account for features, which have a positive influence on the prediction of the target class c.
The activation maps are usually generated in the last convolutional layer, then resized and visualized as a heat-map, as shown in Figure 2.9 The authors state, that the last convolutional layer of a CNN offers the best compromise between detailed spatial information and high-level semantics [28].

Guided Grad-CAM

The Grad-CAM method generates activation maps with the same size of the feature map. As a result, only important regions are visualized instead of fine-grained details of the image. The authors additionally introduced Guided Grad-CAM, which is a combination of Grad-CAM and Guided Backpropagation.
Guided Backpropagation is a method that propagates backwards in the network and visualizes the gradients with respect to the input images [39]. When backpropagating through ReLU layers of the CNN, the method suppresses negative gradients because the goal is to visualize pixels that have an activating effect on the neuron, rather than pixels that suppress the activation.
Guided Grad-CAM fuses the output of Grad-CAM and Guided Backpropagation with element-wise matrix multiplication, as shown in Figure 2.9.

3. Methods

One of the key aspects of this thesis is the utilization of a large data set. The following chapter describes the annotation and preprocessing of the CT scans. Furthermore, the specific implementation is presented.

3.1. CT Images

Initially, the BG Unfallklinik Tübingen provided 222 patients for this project that were classified according to the Letournel system. The additional data set comes from 10 health institutions across Germany.
Medical images like CT Scans are stored as DICOM files. The files include information about the series and the patient. Private tags, including the patients names, needed to be removed, in order to be legally submitted by the hospitals. In [20], an anonymization script was developed to allow hospitals to safely submit their data. Nevertheless, some hospitals anonymized the data themselves which led to disordered folder structures. Moreover, the submitted files came in a variety of formats and with different compression techniques of the CT scans.

First of all, the raw data was sorted and converted into a uniform format. A raw CT scan of a patient is stored in a dedicated folder. Each patient folder consists of several sub-folders, which contain a series of images. Often this structure was damaged and I sorted the scans with a script. The script reads all the images in the folder structure and sorts them according to the remaining meta data. Additionally, the script decompresses the images and sorts them in the correct order of the slices. This was necessary in some occasions. After assigning each patient a unique identification number, the data set was ready to be classified.

Annotation

The annotation was conducted by PD Dr. med. Markus Küper from the department of pelvis and acetabulum surgery in the BG Unfallklinik Tübingen. The medical record of the additional data was not available. As a result, CT scans of 1051 patients needed a diagnosis.
To increase the efficiency of this process, I created a macro for the open-source software ImageJ [40]. Instead of manually selecting each image series for every

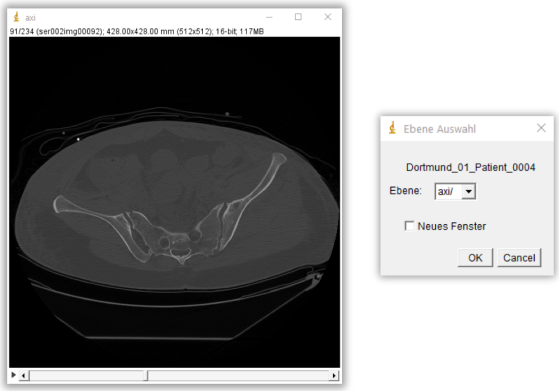

Figure 3.1.: ImageJ for the acetabulum diagnosis. A DICOM viewer displays a image series (left) and small GUI (right) enables the user to switch between the planes.

patient in the large folder structure, the macro enables the user to load the patients data with a single command. When executed, a DICOM viewer displays the first series of the patient as shown in Figure 3.1. Furthermore, a small GUI allows the user to select the desired image series of the patient. When a series is loaded, the pixel intensity is automatically re-scaled for the viewer. After the diagnosis, the user can write the fracture class into an Excel-sheet and continue with the next patient, simply by changing the ID in the command.

3.2. Preprocessing

Prepossessing is a crucial step to filter the data and to ensure the model learns the correct features of an image. Moreover, a high resolution of the CT scan is essential to detect small patterns like fractures. When training CNNs with large 3D images, the computation is extremely demanding. However, only the region around the acetabulum is relevant when classifying the fractures.
Daniel Dehncke solved both problems by developing a prepossessing pipeline. Several scripts evaluate and select useful CT. The important region around the acetabulum is automatically extracted, to reduce the size of the scans while maintaining the available resolution.
In this thesis, I applied the scripts on the large data set but interchanged the extraction step with the combination step, as shown in Figure 3.2. This was necessary, in order to process over 800 GB from 1.3 million images in smaller batches. I created a new combination script in order to fuse the batches to a large data set.

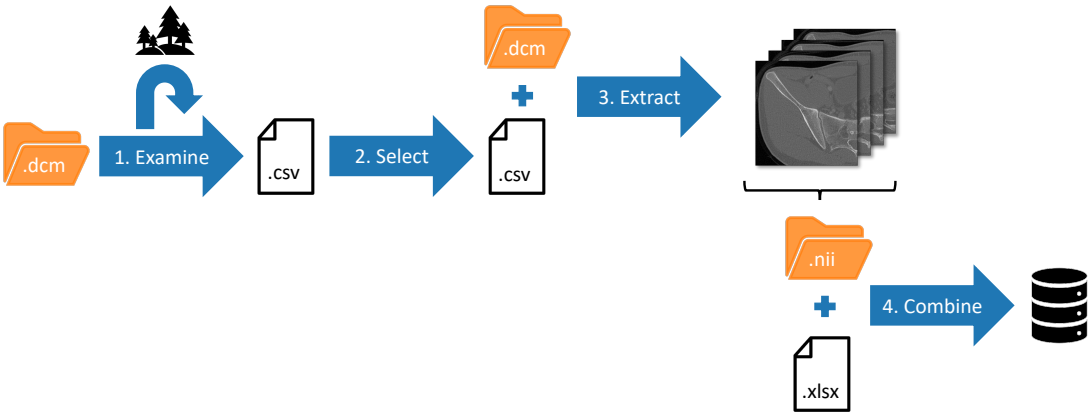

Figure 3.2.: Preprocessing pipeline. First, the images are examined and corresponding information saved. Secondly, a image series is selected for each patient. In the third step, a script extracts the actabulum from the patient's scan. Finally, the extracted images are combined with their fracture labels.

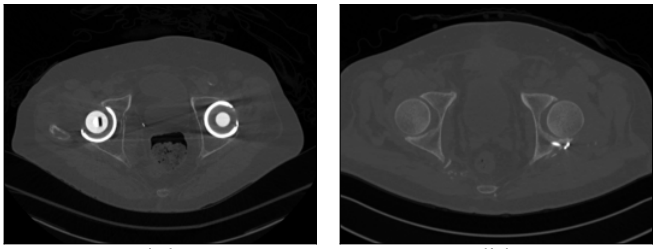

Figure 3.3.: CT scans with artefacts. (a) Patient with artificial hip joints on both sides. (b) Patient with internal fixation for the bone fragments.

1. Examine

In the first step, a script examines each image series in the folder structure. Attributes like the number of images, slice thickness, body part, plane, resolution and outliers are saved into a csv-file. The plane of an image series represents the axis of the scan. Outliers are instances either having a different plane compared to the majority of the image series, or images which have artefacts. Especially metal from internal fixation or artificial hip-joints produce large artefacts, as shown in Figure 3.3.
The model performance could suffer from learning wrong features. In [20], an Isolation Forest was trained on the brightness distribution of correct images, to detect such artefacts and mark them as outliers. Nevertheless, images without artefacts were often marked as outliers. As a result, I had to inspected all the axial scans manually to check the results of the script. A majority of the CT scans showed more than the hip of the patient. Subsequently, I extracted the hip manually for several of

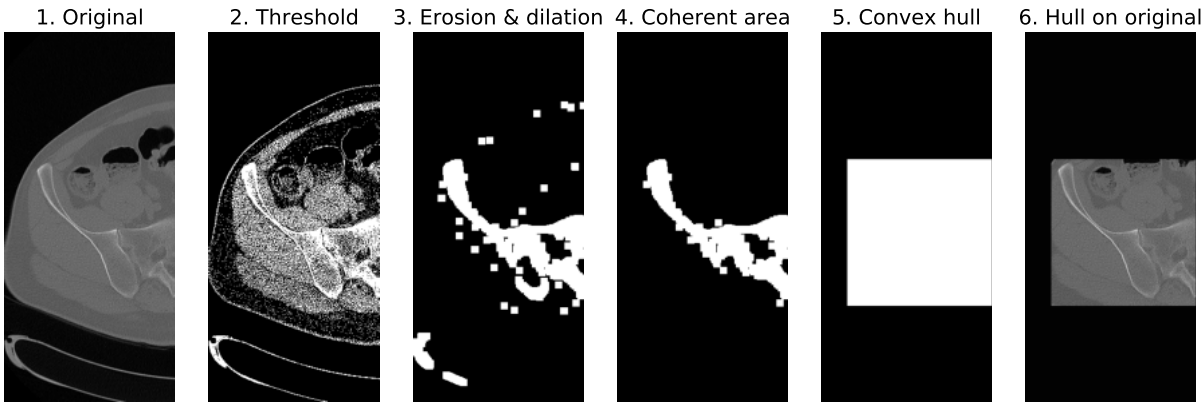

Figure 3.4.: Visualization of the extraction script. The scan is cut in half. A threshold is applied. The script marks pixels with erosion and dilation, which approximately cover the bone. The biggest area is kept and a convex hull generated. The script then masks the original slice with the hull.

the patients. Scans of a single hip were sorted out, because the processing pipeline was not adjusted for it. I repeated the examination step until the usable data was filtered.

2. Select

In the second step, a script selects an image series of a patient according to several constraints. First of all, the CT scan has been taken from the axial plane and shows the hip. Secondly, the image series contains at least 40 slices and the slice thickness does not exceed 3mm. Finally, for each patient the script selects the image series that has the closest thickness to 3mm with the lowest outliers/total-image-number ratio.

3. Extract

In the following step, an extraction script selects 60 evenly spaced slices of the selected image series. If a CT scan of a patient consists of less than 60 images, every image is retained. Acetabulum fractures mostly occur on one side of the patient. The image series is split in half to double the number of samples and to collect non-fracture instances. The script mirrors the right image so that the model only predicts samples which display the left side.

The halves are further reduced by extracting the region around the acetabulum as shown in Figure 3.4. This is done by converting the images into the Hounsfield scale. An intensity threshold of 30 is applied and the regions most likely to show bones are marked with erosion and dilation. Other incoherent structures are removed, resulting in an approximate segmentation of the bone. The script generates a convex

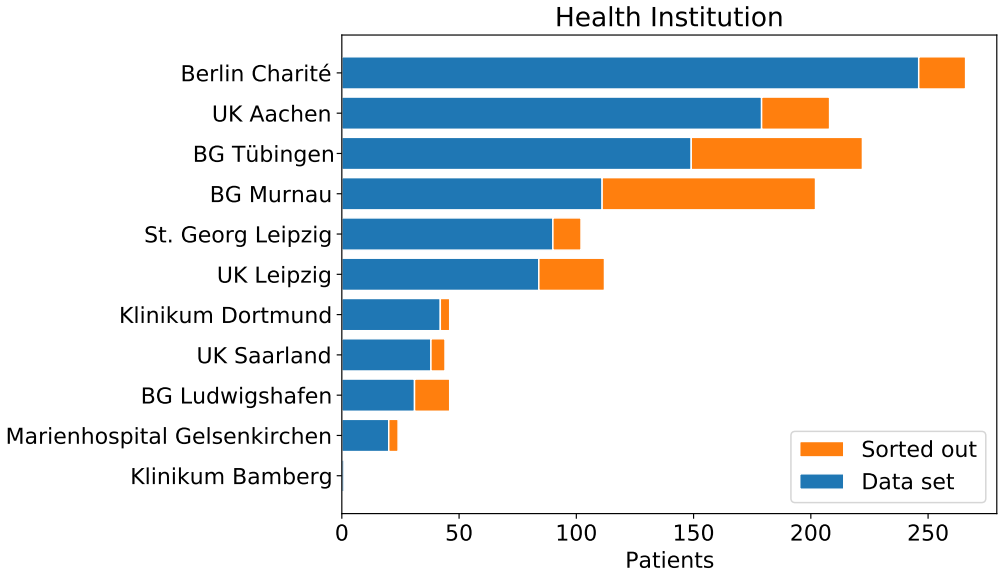

Figure 3.5.: Composition of the data set. For this study, the data set was provided by 11 hospitals. After the preprossessing, 991 out of 1273 patients remained.

hull which covers the segmented region. Finally the script creates 3D cuboid with a fixed aspect ratio for all the halves. The cuboid covers the convex hulls and is used to extract the hip joint from the original image series.
After the acetabulum is extracted, the script saves the resulting volume as a NIfTI file. In contrast to DICOM, NIfTI allows to store a 3D image in a single file instead saving each slice independently.
I inspected the extracted halves of every patient, to ensure a correct extraction. For approximately 10% of the patients, I had to adjust the cuboid and repeat the extraction. This could mostly be resolved by changing the cutting point or by increasing the threshold. In rare occasions, I framed the cuboid manually.

4. Combine

In the last step, the extracted halves are combined with the class labels, resulting in a single data set. Up to this point, the patients of each health institution were preprocessed in a batch due to the large data size. A script reads all the extracted halves and the class labels for each health institution. Extracted images are sorted out, if a diagnosis was not possible by the examiner. I removed patients if the examiner made remarks that the fracture already healed or the scans were made after surgical intervention. Additionally, I removed some patients with other remarks about severe damages in the acetabulum region (e.g. fractured ilium wing, fractured pubic branch, cancerous pelvis), which might hinder the model to learn features of healthy non-fractured pelvises or Letournel classes.

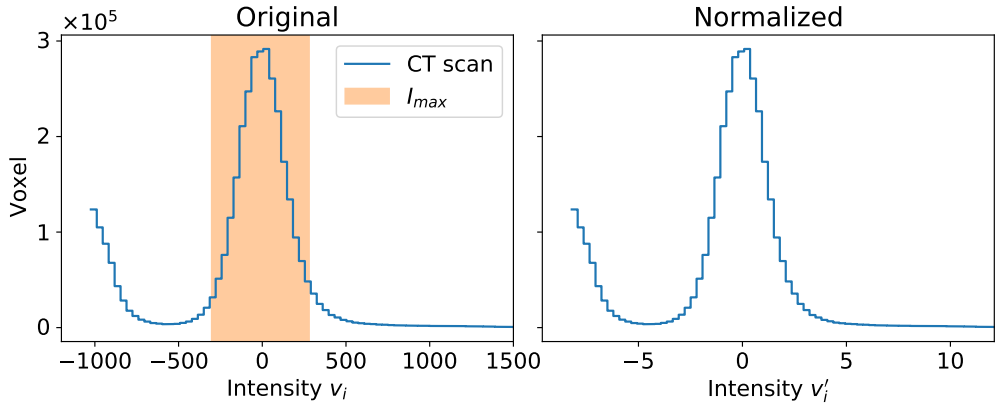

Figure 3.6.: Intensity histograms of an extracted halve (left) and the normalized counterpart (right). The interval I_{max} with the largest area under the curve, is highlighted. The scan is normalized with the mean and standard deviation of I_{max}.

The raw data set, including the patients from the BG Unfallklinik Tübingen, consisted out of 1273 patients. After the preprocessing, 991 (77.8%) patients remained, resulting in 1982 samples, as shown in Figure 3.5.
Finally, the script randomly splits the data for cross-validation, training and testing.

Intensity Normalization

Normalization is an essential step which re-scales the intensity of an image to an equal distribution. In this study, the images were generated with various CT scanners from different hospitals. Normalizing the whole data with the global mean and standard deviation removed the texture in many scans. Furthermore, each image has different characteristics. Some images show large portions of air and others come with artefacts (e.g. metal). As a result, the local mean and standard deviation were fluctuating and unreliable as a measure of central tendency and dispersion.
I created an normalization algorithm that takes the intensity of a voxel v_i and re-scales it to v_i'. First, the histogram of the input volume is generated, as shown in Figure 3.6. The algorithm records all the intensity intervals I_j, which continuously exceed 1% of the voxels. For each interval I_j, the algorithm calculates the area under the histogram. The interval I_{max} with the largest area, approximately covers the voxels showing the patients tissue. The mean $\hat{\mu}_{I_{max}}$ and standard deviation $\hat{\sigma}_{I_{max}}$ of I_{max} are calculated and used to normalize the voxel intensity v_i of the extracted CT scan.

$$v_i' = \frac{v_i - \hat{\mu}_{I_{max}}}{\hat{\sigma}_{I_{max}}} \tag{3.1}$$

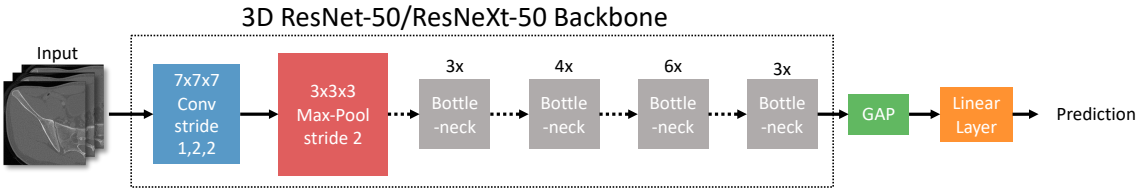

Figure 3.7.: 3D ResNet-50 and ResNeXt-50 architecture [37]. The bottleneck blocks are similar as shown in Figure 3.7 but inflated by one dimension. ResNeXt uses multiple convolution paths in the bottleneck blocks. Batch normalization and activation functions are not shown.

3.3. Implementation

ResNet & ResNeXt

Convolutional Neural Networks for 3D medical images can be adapted from video classification models [41]. Similar to MRIs or CT scans, a video consist out of multiple frames which can be stacked together, resulting in a 3-dimensional volume. In this thesis, I used a 3D ResNet and ResNext for video classification [37][1]. The backbone of the ResNet-50 and ResNeXt-50 is shown in Figure 3.7.
The architecture remains unchanged to the original ResNet and ResNeXt architecture but the operations are extended by one dimension. However, the stride in the first convolution is set to one for the depth dimension. Stride controls with how many units the kernels moves over the input feature map at each step. GAP is applied on the feature maps of the last convolutional layer. A linear layer is connected with nodes that output the predicted probabilities of each class.

EfficietnNet

Secondly, I implemented a 3D EfficientNet[2] model for Pytorch. The backbone of the is shown in Figure 3.8. The implementation uses a stride of one in the first convolution for the depth of the input volume. Nevertheless, the remaining topology of the operations is equal to the original 2D EfficientNet but extended by one dimension. EfficientNet uses the mobile inverted block (MBConv) as main building block, from the the MobileNetV2 architecture [35] and MnasNet architecture [42], together with squeeze and extraction optimization [36]. The EfficientNet architecture has GAP after the last convolution with dropout and a linear layer for the classification.

[1] https://github.com/kenshohara/3D-ResNets-PyTorch
[2] https://github.com/shijianjian/EfficientNet-PyTorch-3D

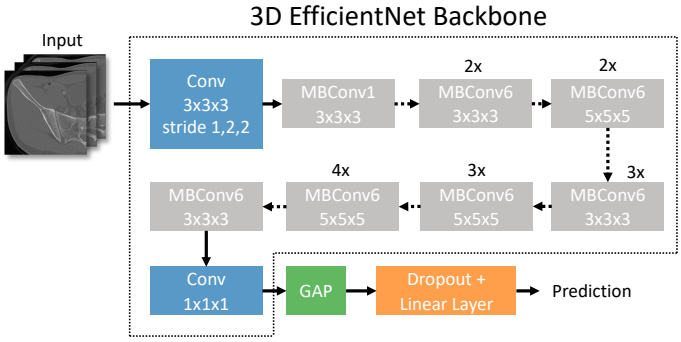

Figure 3.8.: The 3D EfficientNet architecture. The implementation extends the mobile inverted blocks (MBConv) by one dimension. Batch normalization and activation functions are not shown.

Inflated 3D (i3D)

Lastly, I implemented an I3D model for Pytorch. The repository[3] is based on the original Tensorflow implementation and offers pre-trained models from the ImageNet and Charades data set [39, 43]. The original I3D model uses two CNNs for the RBG videos and their corresponding optical flow snippets. In this thesis, I only used one CNN that was modified to receive a single channel input. Furthermore, I removed the last pooling layer and added GAP followed by a linear layer, as shown in Figure 3.9. The nodes of the last linear layer return the scores of each class.
The I3D model does not only inflate the operations of the Inception-V1 architecture but optimizes the feature extraction for videos. In the first two pooling layers, the models does not perform Max-Pooling on the frames of the video (by using a 1×3×3 kernel). As a result, the receptive field is adapted to extract more features in the height and width dimension of the input.

M3d-CAM

M3d-CAM is a library for generating activation maps in Pytorch and was specifically designed for Deep Learning with medical images [44]. The library offers various methods like Guided Backpropagation, Grad-CAM, and Guided Grad-CAM to visualize the prediction in 2D and 3D CNNs. The library also includes Grad-CAM++, which was not working reliably for 3D models. In this thesis, I only only use Grad-CAM and Guided Grad-CAM.
Generally, M3d-CAM allows an uncomplicated integration and only needs a single line of code to "inject" the model with the desired activation map. Next time the model receives an image, the activation map is automatically generated.

[3] `https://github.com/piergiaj/pytorch-i3d`

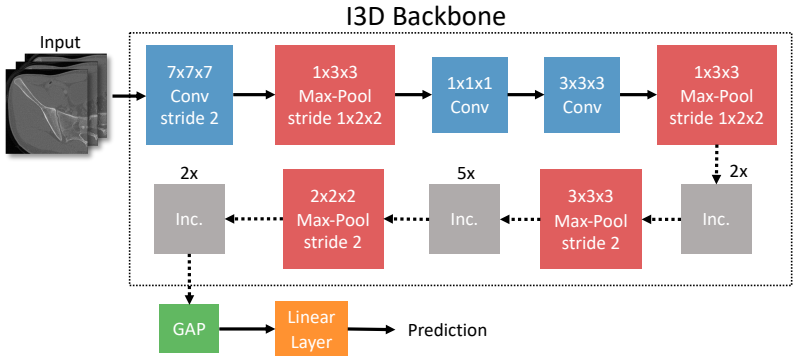

Figure 3.9.: The I3D architecture used in this thesis (adapted from [24]). The last pooling layer is interchanged with GAP and a linear layer. The inflated Inception block (Inc.) was previously introduced in Figure 2.8. Batch normalization and activation functions are not shown.

The output of methods like Grad-CAM have a low resolution. Therefore, my code resizes the attention maps with bicubic interpolation to the original input size.

Training Center for Machine Learning

Graphics Processing Units (GPUs) are commonly used in Deep Learning frameworks to accelerate the computational expensive training of CNNs.
The Training Center for Machine Learning (TCML) in Tübingen offers a large GPU cluster for scientific research and education. The cluster consists out of 40 computation nodes, each node providing a Intel XEON CPU E5-2650 v4 and four NVIDIA GeForce GTX 1080 Ti. The TCML Cluster uses the container virtualization software Singularity [45], which was developed for scientific computing on clusters. A singularity container is any Linux environment condensed to a file, which can be used to execute a program on the cluster. The user can customize the operating system, the environment and libraries while being independent of the server it is running on. This enables a high flexibility of the implementation and reassures the reproducibility of the results.
In this thesis, I used a container with Ubuntu 16.04, Pytorch 1.6.0 [46], and CUDA 9.0 [47]. The container executes the code with a Python 3.7 environment, including libraries like imgaug 0.4.0 [48] for data augmentation and medcam 0.1.6 [44] for generating attention maps.

4. Results

This chapter contains the results on the large cohort of CT scans. In the first two experiments, the models are trained to detect fractures and to classify the three treatment classes. In the last experiment, the models learn to distinguish between all 10 Letournel classes.

4.1. Fracture Detection

The data set for the fracture detection is relatively large, consisting of 997 extracted halves with an acetabulum fracture and 985 halves showing no fracture. The small imbalance comes from 16 patients with fractures on both sides and 10 patients without any fracture.
I split the data set into an 80% training set and a 20% test set. I performed 5-fold Cross Validation (CV) on the training set, to compare several architectures and to optimize corresponding hyper-parameters. The training/test split and CV splits are stratified and approximately evenly distributed across of the Letournel types and the hospitals.
The fracture detection is a binary classification with balanced classes which permits to use accuracy, sensitivity and specificity as performance measures.
Initially, I evaluated the architectures with standard hyper-parameters to acquire their baseline performance. I compared EfficientNet-b3, ResNet-50, ResNeXt-50, and I3D. Afterwards, I selected the architecture with the highest accuracy. This was done, due to the limited time of this thesis.
The models were trained with cross-entropy loss and a SGD optimizer with a learning rate of 0.001 and momentum of 0.9. I chose a batch size of four, to train all models on a single GPU without running out of memory. The ResNeXt model has a cardinality of 16 because of memory constraints. No weight decay or data augmentation were used. An exponential scheduler decreases the previous learning rate by one percent at every epoch. The weights were randomly initialized. The results are shown in Figure 4.1.

The EfficientNet architecture achieved the lowest mean accuracy of 83.7%. ResNet and ResNeXt achieved an accuracy of 87.5% and 85.9%, respectively. Even though the results are similar, it is notable that ResNeXt could not outperform ResNet. Under the given circumstances, the I3D model reaches the highest accuracy of 91.9%.

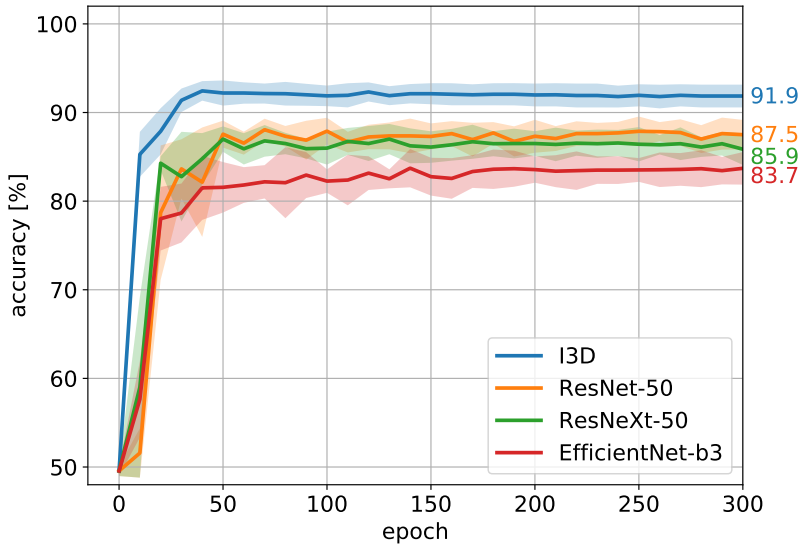

Figure 4.1.: Average validation accuracy over 300 epoch of I3D, ResNet-50, ResNeXt-50 and I3D. The shaded bands show the standard deviation of the CV splits.

Therefore, I selected the I3D architecture for the fracture detection model.

Hereinafter, I compared the influence of several parameters before evaluating the I3D model on the final test set. Only a single parameter is changed at each step and the model evaluated with cross-validation on the training set.
First of all, I trained the I3D model with the same setup as before but examined the influence of other learning rates as shown in Table 4.1.

Table 4.1.: The validation results at epoch 300 of cross-validation when training I3D with different learning rates.

	Accuracy [%]	Specificity [%]	Sensitivity [%]
lr 0.01	90.6 ± 2.2	92.9 ± 3.0	88.4 ± 2.6
lr 0.001	**91.9 ± 1.3**	**93.8 ± 1.2**	**90.0 ± 2.0**
lr 0.0001	89.9 ± 1.0	93.6 ± 1.8	86.4 ± 1.9

A learning rate of 0.001 achieved the best accuracy with 91.9% in average. However, when training with lower or higher learning rates, the I3D model still performed well with an accuracy around 90%. It indicates that the architecture is relatively robust, as long as the learning rate is in an appropriate range. Henceforth, the learning rate of 0.001 was used for the fracture detection model.

Secondly, I examined the influence of different initializations, as shown in Table 4.2. I3D was initialized with weights from models, that were trained on ImageNet and the Charades data set.

Table 4.2.: Average validation results at epoch 300. I3D models pre-trained on ImageNet and the Charades data set are compared with random initialization.

	Accuracy [%]	Specificity [%]	Sensitivity [%]
ImageNet	89.6 ± 2.2	**94.1 ± 2.2**	85.3 ± 3.9
Charades	90.0 ± 1.8	92.3 ± 2.6	87.7 ± 2.0
Random Initialization	**91.9 ± 1.3**	93.8 ± 1.2	**90.0 ± 2.0**

The models with transfer learning cannot outperform the I3D model with randomly initialized weights. It is notable, that pre-training actually has a negative effect and the validation accuracy decreases. Although the I3D model pre-trained on ImageNet reaches the highest specificity with 94.1%, it has a low sensitivity of 85.3%.
Generally, the models have more difficulty with recognising fractures, hence the sensitivity tends to be lower than the specificity. For the fracture detection model, it is preferable to achieve relatively high results in both measurements. Therefore, I decided to use the random initialization in further experiments.

In the next step, I evaluated three variants for data augmentation. The first variant is vastly similar to the augmentation used in Daniel Dehncke's thesis. Each sample is uniformly rotated between -7 and 7 degrees. Additionally, to 50% of the samples zoom, translation, shearing, noise, depth shift and blur is applied. The operations are only applied to the slices of the CT scan.
The second variant uses the same operations, but applies them to any axes of the 3D volume. For each sample, the axis is randomly selected.
The third variant is equal to the second variant, but adds random erasing. For 25% of the samples, a cuboid erases the vovels around the ilium, femur and sacroiliac joint. Additionally, a circular mask erases parts of the scan not showing the patients body. Random erasing has the purpose of ensuring that the model does not rely on features which are irrelevant for detecting acetabulum fractures. The results are presented in Table 4.3.

Table 4.3.: Average validation results of the I3D model at epoch 300. Three proposed augmentation variants are compared.

	Accuracy [%]	Specificity [%]	Sensitivity [%]
No Augmentation	91.9 ± 1.3	93.8 ± 1.2	90.0 ± 2.0
1. Variant	92.8 ± 0.8	95.8 ± 1.4	89.9 ± 1.7
2. Variant	93.1 ± 1.0	95.3 ± 0.8	90.9 ± 1.8
3. Variant	**93.5 ± 1.2**	**96.2 ± 1.5**	**90.9 ± 1.6**

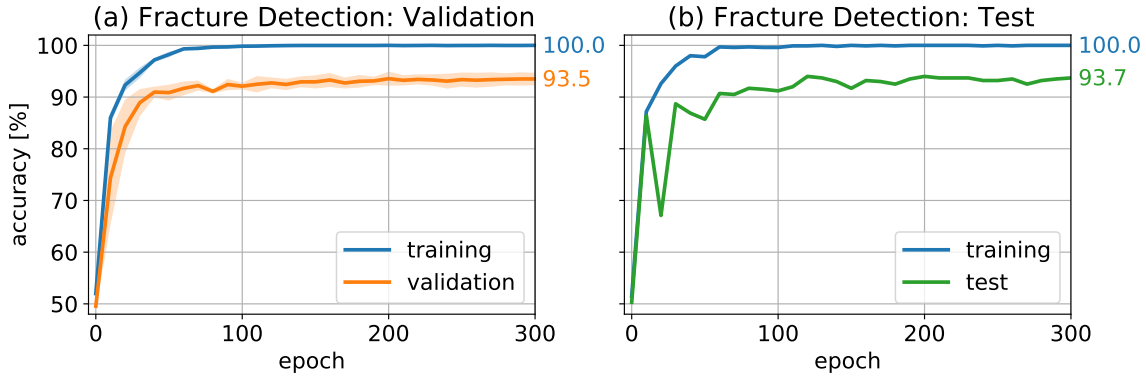

Figure 4.2.: Validation and test accuracy of the best fracture detection model over 300 epochs. (a) Average accuracy of the training and validation splits. (b) Accuracy of the whole training set and the final test set.

Adding data augmentation increased the accuracy by 0.9% to 1.6%. Applying the operations on all axes, as well as erasing parts of the scan had a small but positive influence on the performance. The best I3D model reached a high accuracy of 93.5% on the validation splits with the third variant, as shown in Figure 4.2 (a).

Generally, the I3D model is relatively stable and certain parameters have a low impact on the model. When using weight decay, the model's performance did not improve nor did the performance significantly decrease (see Table A.2). As a results, no weight decay was used.

Finally, I trained the best model on the full training set and evaluated the performance on the independent test set, as shown in Figure 4.2 (b). At epoch 300, it achieves an accuracy of 93.7%, a specificity of 96.5%, and a sensitivity of 90.9%. The confusion matrix is shown in Table 4.4.

Table 4.4.: Confusion table of the I3D model at epoch 300 on the test set.

		Actual	
		Fracture	Non-Fracture
Predicted	Fracture	180	7
	Non-Fracture	18	193

An interesting issue of the fracture detection model, can be found when analyzing the error rate for of each Letournel class independently, as shown in Figure 4.3. Fractures on the anterior wall (2.1) have an outstandingly high error rate. In total, they contribute 9 out of 18 false negative samples, even though they only make about 8% of the fractured samples in the data set and each split.
Furthermore, it is notable that the associated fractures (1.3 + 3.2-3.5) have a low error rate, even of rare Letournel classes. However, the elementary fractures (1.1-1.2 +

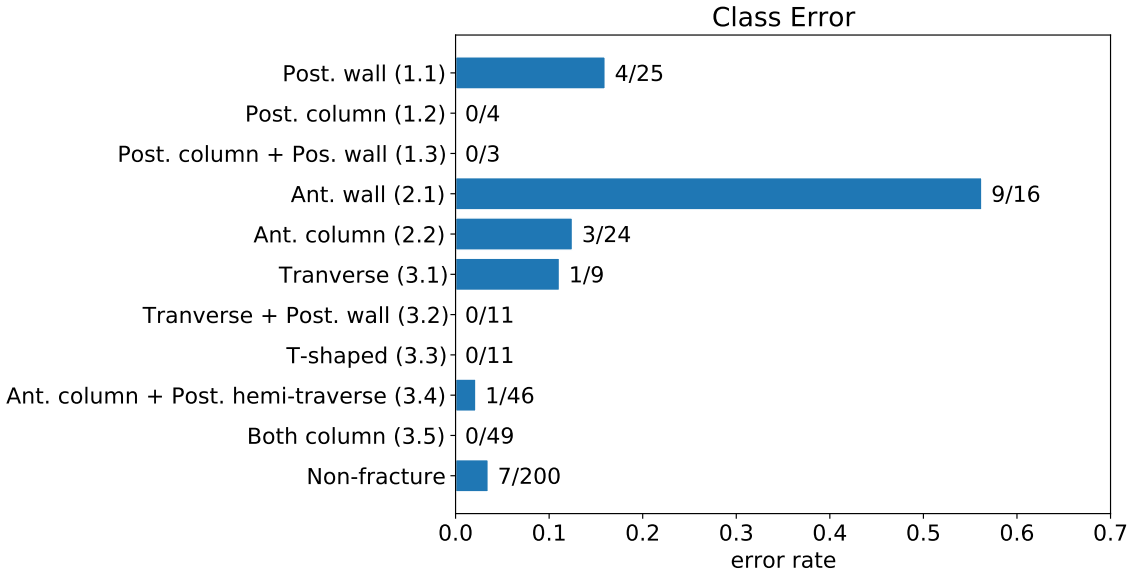

Figure 4.3.: Error rate of the fracture detection model for each Letournel class. Additionally, the error rate for the non-fractured samples is displayed at the bottom.

2.1-3.1), which tend to be smaller, are harder for the model to detect.

Subsequently, I generated attention maps with Grad-Cam and Guided Grad-Cam for some unbiased samples from the test set. With the attention maps, it is possible to comprehend the decision of the model by visualizing important regions for the predicted class. The attention maps are 3-dimensional but for purpose of demonstration only the axial section through the acetabulum is pictured in Figure 4.4.

According to Grad-CAM and Guided Grad-CAM, when a fracture is correctly detected, the model focuses on the region around it. When a sample is correctly classified as non-fracture, the model does not seem to focus on a particular region. Interestingly, the model gives attention to the the anterior wall fracture in Figure 4.4 (c) but classifies it as non-fracture. It is possible to verify whether a fracture is falsely detected because the attention likely correlates with the location of a detected fracture.

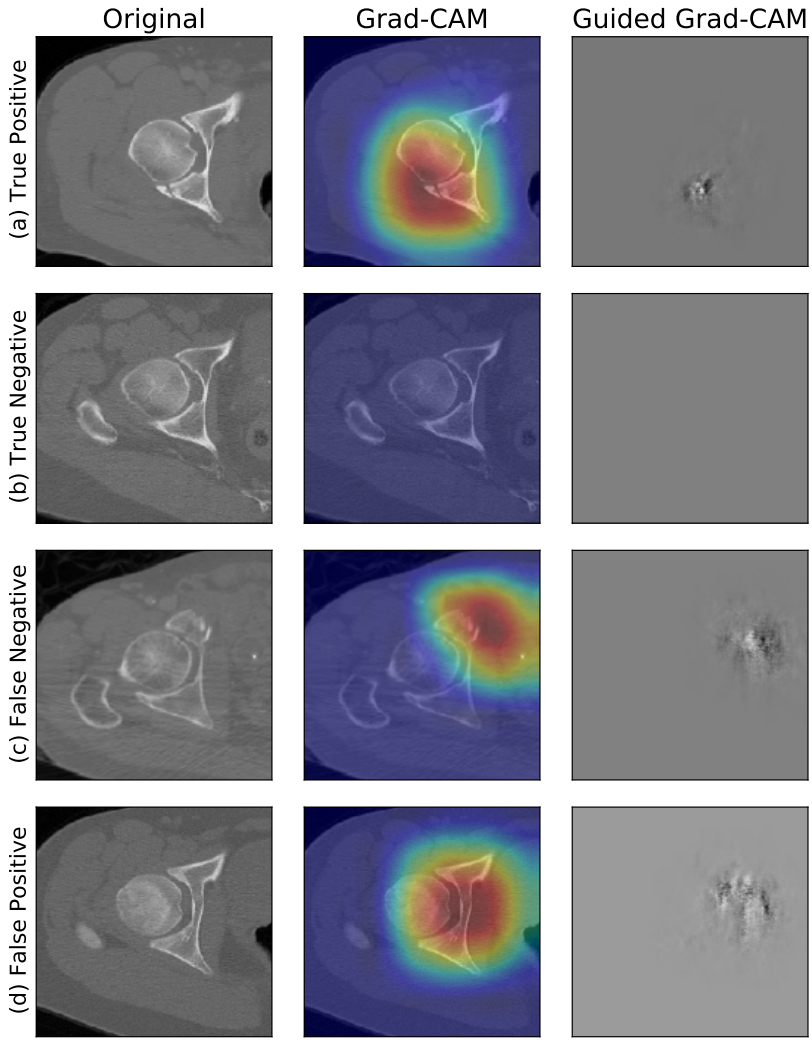

Figure 4.4.: Original scan, Grad-Cam and Guided Grad-Cam of several patients at the axial section of the acetabulum. The activation maps are generated for the predicted class at the last convolutional layer of I3D. (a) Correctly detected fracture at the posterior wall (1.1). (b) Correctly predicted non-fracture. (c) Fracture at the anterior wall (2.1), incorrectly predicted as non-fracture. (d) Non-fracture, incorrectly predicted as fracture.

4.2. Treatment Classification

In the second experiment, the models classify the fractures according to their operational access. The first class (17%) and second class (20%) are only accessed posterior and anterior, respectively. The third class (63%) is predominantly accessed anterior with a posterior combination if necessary. The distribution and Letournel classes of the three categories for the large data set are presented in Figure 4.5.

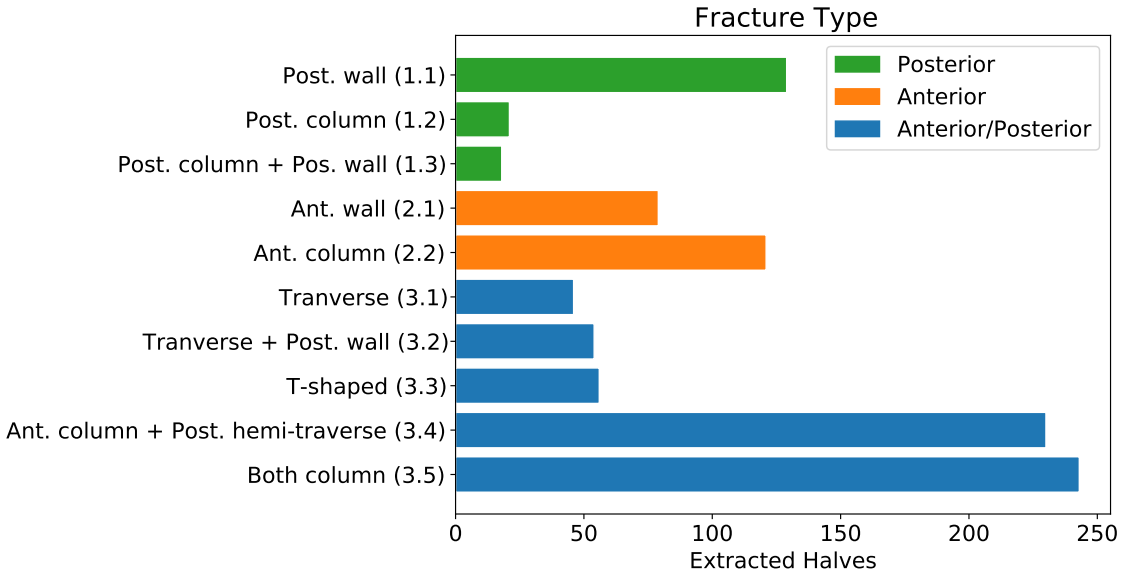

Figure 4.5.: Distribution of the Letournel classes in the data set. The fracture types are highlighted according to their treatment class.

For this experiment, only the 997 fractured samples were used. The samples are divided into an 80% training set and a 20% test set. On the training set, I perform 5-fold CV for validation. The splits (incl. training and test split) are stratified and evenly distributed across the Letournel classes and the hospitals.
Measuring the performance with accuracy when the data set is imbalanced can be misleading. I used the weighted F1-score to evaluate the models in this experiment. It is defined as followed:

$$\begin{aligned} F_1(c) &= 2 \cdot \frac{precision_c \cdot recall_c}{precision_c + recall_c} \\ weighted\ F1 &= \frac{1}{n} \sum_c number(c) \cdot F_1(c) \end{aligned} \tag{4.1}$$

where $F_1(c)$ is the F1-score of class c, $number(c)$ is the number of samples of class c, and n is the total sample size.

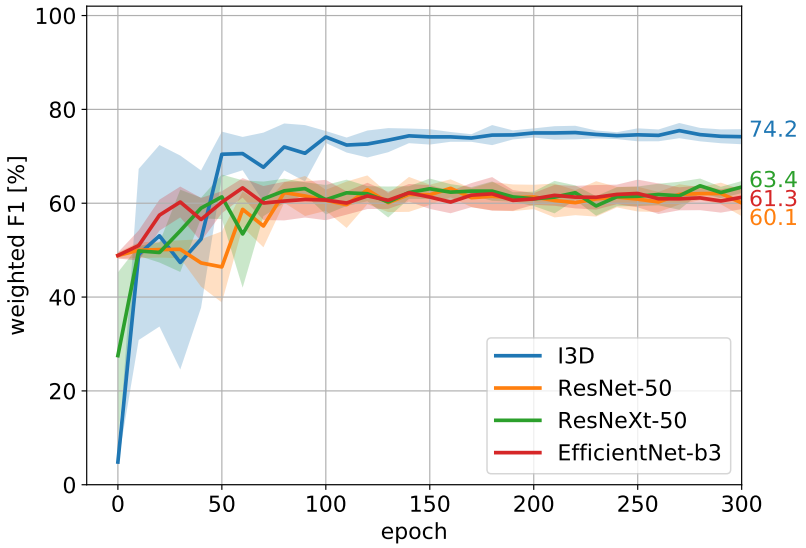

Figure 4.6.: Average weighted F1-score of the validation splits over 300 epochs. The results of I3D, ResNet-50, ResNeXt-50 and EfficientNet-b3 are compared.

Subsequently, I compared the performance of the same four architectures, to select the model with the highest weighted F1-score. The last linear layer of each architecture is extended to output three values with a softmax function. The loss function was changed to cross-entropy for multi-class problems. The SGD optimizer with the corresponding parameters remained unchanged. No data augmentation or weight decay was used. The results are presented in Figure 4.6.

The validation performance of EfficientNet, ResNet and ResNeXt are similar with a weighted F1-score between 60.1% and 63.1%. Interestingly, ResNet was outperformed by EfficientNet and ResNeXt, even though the architecture achieved better results as a fracture detection model.
Overall, the I3D had the best baseline performance with a F1-score of 74.2%. The I3D architecture seems to detect and generalize the fracture categories most effectively, given the fact that I3D outperforms the other architecture by a high margin. Consequently, I selected the I3D architecture for the treatment classification.

Moreover, I repeated the comparison of several learning rates and weight initialization. A learning rate of 0.001 achieved the best results and the random initialized model outperformed the pre-trained models (as shown in Table B.2 and Table B.2).

On the other hand, data augmentation had a regularizing effect on the model and lead to greater performance. With the third variant from Section 4.1, the model achieved a mean weighted F1-score of 77.5%, as shown in Figure 4.7 (a).

The data set for the classification is relatively unbalanced. Therefore I tried a weighted loss function and oversampling to address the problem.

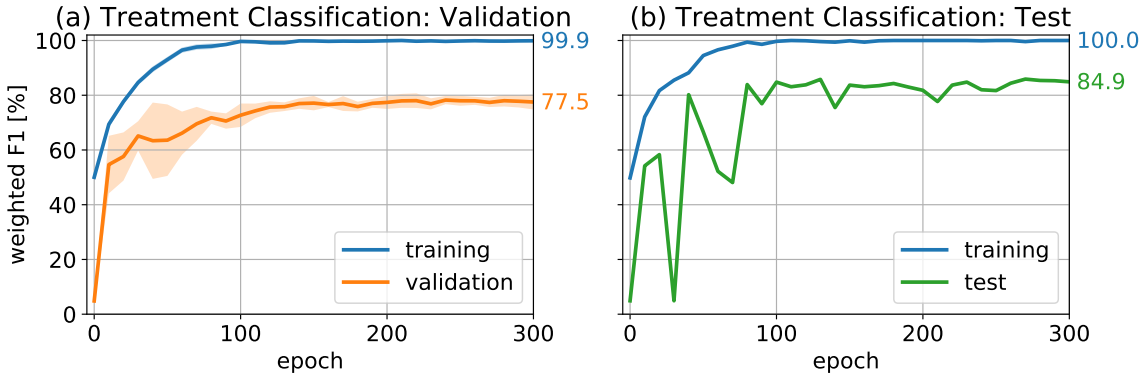

Figure 4.7.: Weighted F1-score of the best fracture classification model over 300 epochs. (a) Average results of the training and validation splits. (b) Results of the whole training set and the independent test set.

For the weighted loss function, the weight of each class c_i is computed by diving the size of the largest class c_{max} by the number of samples in c_i, as followed $w_{c_i} = \frac{|c_{max}|}{|c_i|}$. As a results, the false prediction of minority classes cause a higher loss.
With oversampling, each class has a dedicated probability, with which a corresponding sample is pulled for the training. A smaller class has a higher probability than a class with many samples. A sample can be drawn multiple times, which results in an approximately even distribution across the classes during training.
Nevertheless, the regular model achieved the best results and is used for the final evaluation. Oversampling and weighted loss did not have a positive influence under given circumstances (see Table B.4).

Finally, I trained the best I3D model on the whole training set and evaluated the performance on the test set, as shown in Figure 4.7 (b). The I3D model for the treatment classification reaches a weighted F1-score of 84.9%, and an accuracy of 84.5%. The confusion matrix is shown in Table 4.5.

Table 4.5.: Confusion table for the treatment classification. The results of the I3D model were obtained at epoch 300.

		Actual		
		Class 1	Class 2	Class 3
Predicted	Class 1	30	1	3
	Class 2	1	32	16
	Class 3	3	7	107

4.3. Letournel Classification

Finally, I studied the capabilities of the models to classify all Letournel classes. Similar to the previous experiment in Section 4.2, the models were trained with 997 fractured samples and the same stratified splits. I evaluated the models with the weighted F1-score.

As previously, I compared the four different architectures against each other in the initial step. The models were extended to output the predictions for 10 classes. Other parameters of the models and the training remain unchanged. The results are show in Figure 4.8.

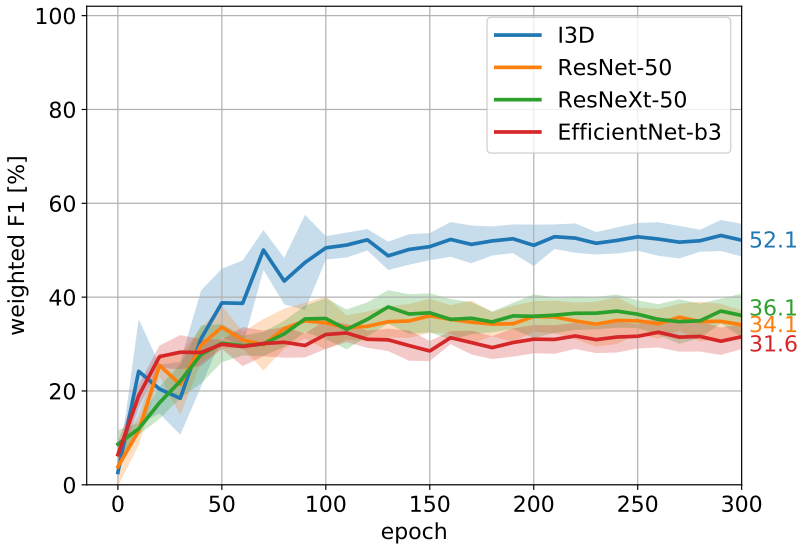

Figure 4.8.: Average weighted F1-score of the validation splits over 300 epochs. The results of I3D, ResNet-50, ResNeXt-50 and EfficientNet-b3 are compared.

On the validation splits, EfficientNet achieves the lowest performance with an weighted F1-score of 31.6%, followed by ResNet and ResNeXt with 34.1% and 36.1%, respectively. After all, I3D is the most powerful architecture and achieves a weighted F1-score of 52.1% in average. It is notable, that I3D continuously outperformed other the architecture by a wide margin. Consequently, I used the I3D architecture for the Letournel classification.

Next, I tried various learning rates, initialization with pretrained models, weighted loss, and oversampling (see Appendix C). However, I could not determine a performance advantage or other benefits, which coincides with the results of the fracture detection and treatment classification experiment.

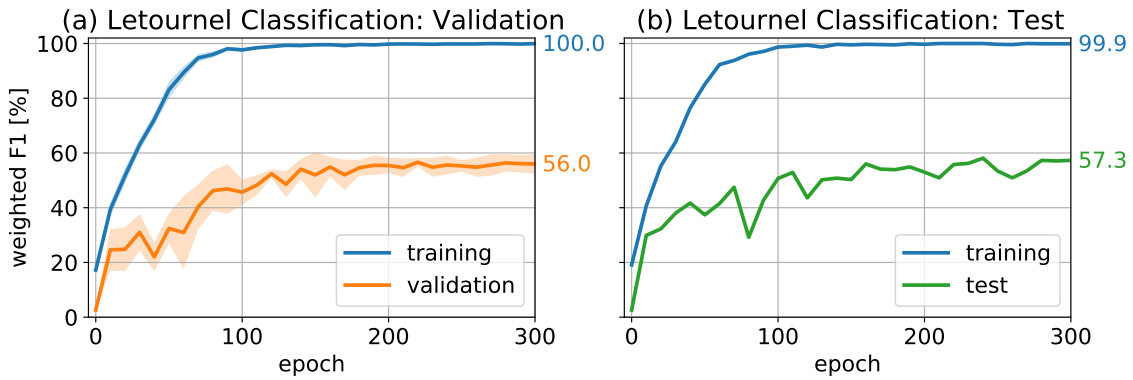

Figure 4.9.: Weighted F1-score of the best Letournel classification model over 300 epochs. (a) Average results of the training and validation splits. (b) Results of the whole training set and the final test set.

After all, data augmentation improved the performance of the Letournel classification. With the third variant of Section 4.1, the I3D model achieves a mean weighted F1-score of 56.0%, as shown in 4.9 (a). Furthermore, the model reaches a Top-1 accuracy of 56.8% and a Top-2 accuracy of 74.9% in average.
Subsequently, I trained the I3D model on the complete training set and evaluated the performance on the independent test set. The model reaches a weighted F1-score of 57.3%, as shown in 4.9 (b). In addition, the model achieves a Top-1 accuracy of 58.5%, and a Top-2 accuracy 78.0% on the test set.

A good insight into the strengths and weaknesses of the Letournel model is given by the confusion matrix in the Table 4.6.
First of all, the model seems to detect fractures in the right area but sometimes is not able to distinguish whether the column or wall of the acetabulum is affected. For example, the model never predicts the posterior column class (1.2) but classifies all the affected samples as posterior wall fractures (1.1). Furthermore, the model cannot differentiate in some instances between an anterior wall fracture (2.1) and an anterior column fracture (2.2).
Another interesting pattern can be observed for the T-shaped fractures (3.3), anterior column + posterior hemi-traverse fractures (3.4) and the two-column fractures (3.5). The model can detect that both column of the acetabulum are affected but is not able to generalize the fracture lines of each class. This leads to some miss-classifications between the mentioned Letournel types.
Due to the similarity of the classes, it is worth noticing, that five samples with a anterior column fracture (2.2) are classified as an anterior column + posterior hemi-traverse fracture (3.4).

Table 4.6.: Confusion table for the Letournel classification. The results of the I3D model were obtained at epoch 300.

			Actual									
			Class 1			Class 2		Class 3				
			1.1	1.2	1.3	2.1	2.2	3.1	3.2	3.3	3.4	3.5
Predicted	Class 1	1.1	21	4	1	0	0	0	1	0	0	0
		1.2	0	0	0	0	0	0	0	0	0	0
		1.3	1	0	2	0	0	0	2	0	0	0
	Class 2	2.1	1	0	1	12	4	3	0	1	1	1
		2.2	0	0	0	3	12	0	1	1	2	2
	Class 3	3.1	1	0	0	1	0	2	1	0	1	0
		3.2	2	0	0	0	1	0	6	0	0	0
		3.3	0	0	0	0	0	2	0	5	5	1
		3.4	0	0	0	0	5	2	0	4	20	8
		3.5	0	0	0	0	2	0	0	0	17	37

The Letournel model slightly outperforms the treatment classification model from Section 4.2, which achieved a weighted F1 score of 84.9%. When the true labels and predicted labels are converted into the three treatment classes, the Letournel model achieves a weighted F1-score of 85.7% and an accuracy of 85.5% at epoch 300 on the test set.
Although the improvement is small, it shows two characteristics. Firstly, the Letournel model may falsely classify the Letournel class but the treatment would often not change. Secondly, a separate CNN specifically trained for the treatment, is not able to achieve a superior performance compared to the Letournel model. The confusion table of the Letournel model for the treatment classes is shown in Table 4.7.

Table 4.7.: Confusion table of the Letournel model for the treatment classes. For comparison, the results of the treatment classification model are beside in brackets

		Actual					
		Class 2		Class 1		Class 3	
Predicted	Class 1	29	(30)	0	(1)	3	(3)
	Class 2	2	(1)	31	(32)	12	(16)
	Class 3	3	(3)	9	(7)	111	(107)

The classification of a fractured sample needs about 0.3 seconds in average. The performance of the Letournel model in comparison to human observers is shown in Table 4.8. In the literature, senior and junior residents achieve an accuracy between 30% and 55%. The I3D model can slightly outperform inexperienced physicians but remains inferior to experts and fellows.

Table 4.8.: Diagnostic accuracy of the Letournel classification in the literature. The studies grouped the participants according to their experience.

Study	Sample size	CT scan	Group	Diagnostic accuracy [%]	Time per fracture [min]
Riouallon et al. [49]	35	2D + 3D multiplanar	Expert	88.6	1.4
			Fellow	74.0	2.0
			Senior	50.1	2.2
			Junior	42.0	2.4
Boudissa et al. [50]	23	2D + 3D	Junior	52.0	< 2.0
Garrett et al. [51]	20	2D	Senior	42.4	-
			Junior	36.3	-
Hüfner et al. [8]	10	2D	Expert	76	-
			Senior	55	-
			Junior	30	-
Present study	200	2D	-	58.5	< 0.01

5. Discussion

In this thesis, models for acetabulum fractures were further developed and trained on a large data set. A pre-processing pipeline was used to process the enormous amount of data and extract the hip-joint in the CT scans. I compared a variety of 3D CNN architectures to evaluate their applicability for CT scans. I reevaluated the results of the fracture detection model as well as the treatment classification model. Afterwards, I created a model to distinguish between all Letournel fracture types.

The best fracture detection model achieved a high accuracy of 93.7% on an independent test set. Further development of the 3D CNN and a larger data set could improve the accuracy of 82.8% in the previous thesis [20]. The model had the most difficulties to detect anterior wall fractures. The fractures are often fissures in the acetabulum which might have been too small for the model to detect.
Generally, the activation maps of the fracture detection is conclusive. The model generates activation in a detected fracture region. This is useful when a human observer wants to understand or verify the prediction of the model. However, Grad-CAM is a heuristic technique and does not give a definite explanation of the decision.

The Letournel model achieved a weighted F1-score of 57.3% and an accuracy of 58.5%. To my knowledge, it is the first reported result concerning the automatic classification of Letournel fractures. When comparing with human observers in literature, the model outperforms medical professionals with little experience, while being much faster. Junior and senior residents achieve an accuracy between 30% and 55% with 2D reconstruction [8, 51, 49]. The model can not compete with experts which achieve an accuracy between 76% and 88% in the same studies. It was repeatedly shown that physicians reach higher rates with 3D reconstruction and segmentation without the femoral head. Consequently, junior and senior fellows achieve rates between 52.5% and 64% [8, 51]. In general, it is difficult to compare the performance, given the different data and approaches of the studies. Nevertheless, when only the 2D scans of the axial plane is given, the Letournel model is able to support the diagnosis of novice examiners.
The Top-2 accuracy of 78.0% of indicates, that the correct fracture often achieves a high score in the softmax distribution. Consequently, the model can offer further value for the diagnosis, if the estimated likelihoods of the fractures are considered.
In summary, the Letournel model has weaknesses. It detects the fractures at the

correct region but can't distinguish between the wall and column of the acetabulum. Moreover, the model misclassifies similar fracture patterns. False predictions often occur for the fracture types "T-shaped", "anterior column + posterior hemi-traverse", and "two-column". These fractures also tend to be challenging to distinguish for human observers [4]. The individual performance of smaller classes is hard to evaluate. For example, there test set only has four posterior column fractures, that were never predicted by the model. The ability to generalize the fractures should evaluated again, if more data of rare Letournel types is available.
Furthermore, the Letournel model classifies the fractures with a weighted F1-score of 85.7% according to their surgical access. The treatment classification model, specifically trained for the task, achieves weighted F1-score of 84.9%. A separate model for the surgical access does not show a benefit and might be obsolete in practice. Having more information about the fracture during training might lead to the a superior performance of the Letournel model. After all, the classification of the surgical access greatly improved in comparison to the previously ascertained F1-score of 63.3% (in [20]).

The results of three experiments were similar. When comparing different architectures, the I3D architecture repeatably outperformed ResNet, ResNeXt and EfficientNet. The original I3D model was trained with videos that have 64 frames. In contrast to the other architectures, I3D does not use symmetric kernels for the the first pooling layers. The receptive field is adapted to extract more features in the slices of the volume. This might have been beneficial for the feature extraction of the CT scans.
The batch size of four is small in relation to the training set. The parameters are updated more often in each epoch. Subsequently, the model has to do smaller update steps, which explains why a small learning rate of 0.001 achieved the best results. The learning rate should be adjusted, when more GPU memory is available and training with larges batches is possible.
Pre-training on natural images and videos did not improve the performance of I3D. The features of the fractures might been substantially different compared to patterns on natural footage. Ultimately, training from scratch led to better performance.
Heavy data augmentation improved the performance in every experiment by a small amount. The reason could be the usage of GAP, which sums out spatial information before the last linear layer and already acts as a structural regularizer [26].
However, the tuning has certain limitations. First of all, only a singly parameter was changed at each step. A grid search with the parameters would offer more insight into their relation. Secondly, only the I3D architecture was tuned and evaluated on the test set. In order to make a fair comparison, the hyper-parameter tuning should be conducted for every architecture.

Conclusion & Outlook

The correct classification of acetabulum fractures remains a challenging problem. Adequate expertise and spatial perception are crucial in order to make a sufficiently accurate diagnosis. The application of 3D Concolutional Neural Networks for the analysis of CT scans can be substantially beneficial. The large data set and a more advanced architecture further improved the diagnostic performance of the proposed computer-aided approach. The models offer a fast and fairly accurate method to detect acetabulum fractures and to classify their surgical access. Given enough data, the models can distinguish between all Letournel types and are able to assist doctors with little experience.

Nevertheless, the consolation of experts remains essential, because the diagnosis is not yet reliable. However, the approach has room for future improvement.
There is little research concerning 3D CNNs for medical data. The I3D model led to great results but was designed for videos. It would be interesting to construct a CNN that is specifically designed for the analysis of CT scans. Natural videos and CT scans are vastly different. A customized architecture might be important for further development.
Moreover, the models only predicted the fractures based on 60 axial slices. For physicians, the coronal and sagittal reconstruction can make the decisive difference for the correct decision [52, 8]. Therefore, additional planes might be crucial for a dependable classification of model. An interesting approach would be to train three separate CNNs for each plane and combine the classification scores to a single output. This would be similar to the original "two-stream" architecture of I3D. Another option is the training with more axial slices to achieve a higher resolution of the 3D volume. It could improve the detection of smaller fractures but requires a GPU with more memory.
A limitation of this thesis lies in the supervised task itself. First of all, the fractures were classified by a single expert for acetabulum surgery. The models are not trained with the true fracture labels because the medical records are regulated by privacy law. In the future, more experts should analyse the CT images in order to discuss the most suitable diagnosis.
Secondly, the model just predicts a class labels but is not able to detect corner cases or transitional forms, which can occur for Letournel fractures [8]. Physicians are responsible for the individual treatment planning, even if the model achieves a high performance in the course of time.

In the future, it is planned to implement the models into the back end of a website. Hospitals without a specialist can upload the data and receive the results of the model for a second opinion. A CT viewer could visualize the extracted acetabulum and the coarse activation maps of Grad-CAM. By receiving more data, the service could be continuously improved.

A. Appendix: Fracture Detection

Table A.1.: Averaged validation accuracy, sensitivity and specificity from various architectures. The results were observed at epoch 300.

	Accuracy [%]	Specificity [%]	Sensitivity [%]
EfficientNet-b3	83.7 ± 1.8	89.3 ± 2.9	78.3 ± 4.6
ResNeXt-50	85.9 ± 1.8	86.4 ± 4.8	85.4 ± 2.4
ResNet-50	87.5 ± 1.7	89.0 ± 4.6	86.1 ± 1.8
I3D	**91.9 ± 1.3**	**93.8 ± 1.2**	**90.0 ± 2.0**

Table A.2.: Average validation results of the I3D model with several values for weight decay. Data augmentation was used

Weight decay	Accuracy [%]	Specificity [%]	Sensitivity [%]
10^{-3}	93.2 ± 1.4	**96.2 ± 1.2**	90.3 ± 2.2
10^{-4}	93.0 ± 1.6	95.4 ± 1.6	90.6 ± 2.4
10^{-5}	93.3 ± 1.4	96.0 ± 2.1	90.6 ± 1.6
10^{-6}	93.2 ± 1.3	96.6 ± 1.1	89.9 ± 1.9
0	**93.5 ± 1.2**	96.2 ± 1.5	**90.9 ± 1.6**

B. Appendix: Treatment Classification

Table B.1.: Average results of the validation splits at epoch 300. The weighted F1-score and accuracy of I3D, ResNet-50, ResNeXt-50 and EfficientNet-b3 are recorded along with the standard deviation across the CV splits. No data augmentation was used.

	weighted F1 [%]	Accuracy [%]
EfficientNet-b3	61.3 ± 2.5	62.7 ± 3.0
ResNeXt-50	63.4 ± 1.3	63.2 ± 1.4
ResNet-50	60.1 ± 2.8	58.6 ± 3.9
I3D	**74.2 ± 1.6**	**73.7 ± 1.8**

Table B.2.: Comparison of different learning rate for the I3D model. The weighted F1-score and accuracy of the validation splits are averaged at epoch 300. No data augmentation was used.

	weighted F1 [%]	Accuracy [%]
lr 0.01	70.2 ± 0.6	70.1 ± 1.0
lr 0.001	**74.2 ± 1.6**	**73.7 ± 1.8**
lr 0.0001	71.9 ± 2.5	70.7 ± 2.5

Table B.3.: Comparison of the I3D performance, when the model is initialized with pre-trained weights. The weighted F1-score and accuracy of the validation splits are averaged at epoch 300. No data augmentation was used.

	weighted F1 [%]	Accuracy [%]
ImageNet	73.6 ± 2.2	73.3 ± 2.4
Charades	73.3 ± 1.5	72.5 ± 1.5
Random Initialization	**74.2 ± 1.6**	**73.7 ± 1.8**

Table B.4.: Comparison of weighted loss and oversampling to the regular I3D model. The weighted F1-score and accuracy of the validation splits are averaged at epoch 300. Data augmentation was used.

	weighted F1 [%]	Accuracy [%]
weighted loss	76.6 ± 3.1	76.4 ± 3.1
oversampling	76.3 ± 3.4	76.9 ± 3.3
regular	**77.5 ± 2.7**	**77.4 ± 2.8**

C. Appendix: Letournel Classification

Table C.1.: Average results of the validation splits at epoch 300. The weighted F1-score and accuracy of I3D, ResNet-50, ResNeXt-50 and EfficientNet-b3 are recorded along with the standard deviation across the CV splits. No data augmentation was used.

	weighted F1 [%]	Top-1 Accuracy [%]	Top-2 Accuracy [%]
EfficientNet-b3	31.6 ± 2.6	31.7 ± 3.1	52.8 ± 2.8
ResNeXt-50	36.1 ± 4.7	37.9 ± 4.2	56.7 ± 3.9
ResNet-50	34.1 ± 3.1	34.5 ± 3.8	55.2 ± 5.6
I3D	**52.1 ± 3.5**	**53.2 ± 3.3**	**74.4 ± 2.5**

Table C.2.: Average results of the validation splits at epoch 300. Comparison of several learning rate (lr) of the I3D model for the Letournel classification. No data augmentation was used.

	weighted F1 [%]	Top-1 Accuracy [%]	Top-2 Accuracy [%]
lr 0.01	47.1 ± 4.9	47.3 ± 5.0	67.6 ± 5.0
lr 0.001	**52.1 ± 3.5**	**53.2 ± 3.3**	**74.4 ± 2.5**
lr 0.0001	47.4 ± 1.4	48.8 ± 1.6	69.4 ± 2.9

Table C.3.: Average results of the validation splits at epoch 300. Comparison of pre-trained weights with random initialized weights of the I3D model for the Letournel classification. No data augmentation was used.

	weighted F1 [%]	Top-1 Accuracy [%]	Top-2 Accuracy [%]
ImageNet	51.6 ± 3.0	52.9 ± 3.1	74.2 ± 1.6
Charades	51.6 ± 3.1	52.3 ± 3.3	73.0 ± 2.8
Random Initialization	**52.1 ± 3.5**	**53.2 ± 3.3**	**74.4 ± 2.5**

Table C.4.: Average results of the validation splits at epoch 300. Weighted loss and oversampling in comparison to the regular I3D model for the Letournel classification. Data augmentation was used.

	weighted F1 [%]	Top-1 Accuracy [%]	Top-2 Accuracy [%]
weighted loss	54.8 ± 2.0	55.6 ± 2.7	77.4 ± 2.7
oversampling	53.1 ± 4.7	55.7 ± 5.3	75.9 ± 4.0
regular	**56.0 ± 3.5**	**56.8 ± 3.9**	**74.9 ± 3.7**

Bibliography

[1] M. Tile. *Fractures of the Acetabulum*, pages 291–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[2] Navid Ziran, Gillian L. S. Soles, and Joel M. Matta. Outcomes after surgical treatment of acetabular fractures: a review. *Patient Safety in Surgery*, 13(1):16, Mar 2019.

[3] Henri M. de Bakker, Melanie Tijsterman, Bela Kubat, Vidija Soerdjbalie-Maikoe, Rick R. van Rijn, and Bernadette S. de Bakker. Postmortem radiological case series of acetabular fractures after fatal aviation accidents. *Forensic Science, Medicine and Pathology*, 14(1):62–69, Mar 2018.

[4] A. Schäffler, F. Fensky, D. Knöschke, N. P. Haas, U. Stöckle, B. König, and AG Becken III DGU. Ct-basierten klassifikationshilfe für acetabulumfrakturen. *Der Unfallchirurg*, 116(11):1006–1014, Nov 2013.

[5] T. Pohlemann, P. Mörsdorf, U. Culemann, and A. Pizanis. Behandlungsstrategie bei azetabulumfraktur. *Trauma und Berufskrankheit*, 14(2):125–134, May 2012.

[6] H. Tscherne and T. Pohlemann. *Tscherne Unfallchirurgie: Becken Und Acetabulum*. Tscherne Unfallchirurgie. Springer, 1998.

[7] G.T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Advances in Computer Vision and Pattern Recognition. Springer London, 2009.

[8] T. Hüfner, T. Pohlemann, A. Gänsslen, P. Assassi, M. Prokop, and H. Tscherne. Classification of acetabular fractures. a systematic analysis of the relevance of computed tomography. *Der Unfallchirurg*, 102(2):124–131, Feb 1999.

[9] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60 – 88, 2017.

[10] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[11] Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan,

Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, Jan 2020.

[12] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018:7068349, Feb 2018.

[13] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[17] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron C. Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *CoRR*, abs/1505.03540, 2015.

[18] Juan Zhou, Lu-Yang Luo, Qi Dou, Hao Chen, Cheng Chen, Gong-Jie Li, Ze-Fei Jiang, and Pheng-Ann Heng. Weakly supervised 3d deep learning for breast cancer classification and localization of the lesions in mr images. *Journal of Magnetic Resonance Imaging*, 50(4):1144–1151, 2019.

[19] Manhua Liu, Fan Li, Hao Yan, Kundong Wang, Yixin Ma, Li Shen, and Mingqing Xu. A multi-model deep convolutional neural network for automatic hippocampus segmentation and classification in alzheimer's disease. *NeuroImage*, 208:116459, 2020.

[20] Daniel Dehncke. Acetabulum fracture classification with 3d deep learning on ct images. [Unpublished master's thesis], Eberhard Karls Universität Tübingen, 2020.

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

[22] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Mar-

shall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[24] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018.

[25] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[26] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[27] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[28] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, Dec 2020.

[31] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[33] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[34] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[35] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[36] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[37] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? *CoRR*, abs/1711.09577, 2017.

[38] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey, 2019.

[39] K. Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[40] Caroline A. Schneider, Wayne S. Rasband, and Kevin W. Eliceiri. Nih image to imagej: 25 years of image analysis. *Nature Methods*, 9(7):671–675, Jul 2012.

[41] Sihong Chen, Kai Ma, and Yefeng Zheng. Med3d: Transfer learning for 3d medical image analysis. *CoRR*, abs/1904.00625, 2019.

[42] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile, 2019.

[43] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. *CoRR*, abs/1604.01753, 2016.

[44] Karol Gotkowski, Camila Gonzalez, Andreas Bucher, and Anirban Mukhopadhyay. M3d-cam: A pytorch library to generate 3d data attention maps for medical deep learning, 2020.

[45] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):1–20, 05 2017.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[47] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, and Amit Bawaskar. GPGPU processing in CUDA architecture. *CoRR*, abs/1202.4347, 2012.

[48] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. `https://github.com/aleju/imgaug`, 2020. Online; accessed 01-Feb-2020.

[49] Guillaume Riouallon, Amer Sebaaly, Peter Upex, Mourad Zaraa, and Pomme Jouffroy. A new, easy, fast, and reliable method to correctly classify acetabular

fractures according to the letournel system. *JB & JS open access*, 3(1):e0032–e0032, Feb 2018. 30229234[pmid].

[50] M. Boudissa, B. Orfeuvre, M. Chabanas, and J. Tonetti. Does semi-automatic bone-fragment segmentation improve the reproducibility of the letournel acetabular fracture classification? *Orthopaedics Traumatology: Surgery Research*, 103(5):633 – 638, 2017.

[51] Jeffrey Garrett, Jason Halvorson, Eben Carroll, and Lawrence X. Webb. Value of 3-d ct in classifying acetabular fractures during orthopedic residency training. *Orthopedics*, 35(5):e615–e620, May 2012. 22588400[pmid].

[52] James F. Kellam and Andrew Messer. Evaluation of the role of coronal and sagittal axial ct scan reconstructions for the imaging of acetabular fractures. *Clinical Orthopaedics and Related Research®*, 305, 1994.