Investigation of Options to Handle 3D MRI Data via Convolutional Neural Networks - Application in Knee Osteoarthritis Classification

A thesis
submitted in partial fulfillment of the
requirements for the degree

Master of Science

Faculty of Mathematics
Otto-von-Guericke University Magdeburg

In collaboration with
Zuse-Institute Berlin

Presented by
Jan Krause
born on 4 April 1996
Course of study Mathematics
Field of study Mathematics

July 26, 2021

Supervised by
Prof. Dr. rer. nat. habil. Sebastian Sager
(Institute for Mathematical Optimization, OvGU)
and
Dr. ing. Stefan Zachow
(Computational Diagnosis and Therapy Planning, ZIB)

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

_____

Ort, Datum, Unterschrift

Acknowledgements

To begin with, special thanks to my first contact partner Alexander Tack, who not only always answered my questions, but also constantly motivated me during the whole process. Then I want to thank Dr. Stefan Zachow and Prof. Dr. Sager for supervising this thesis. I want to also thank my flat mate and friend Kilian Bock for helping me concerning different aspects including the exhausting repair of damaged data.

Note to figure references

Figures that are not directly referenced were obtained by the reference that is clear by context.

Contents

1 Introduction **8**
- 1.1 Scope and Delimitations . . . 9
- 1.2 Outline . . . 9

2 Foundation and Motivation **10**
- 2.1 Technical Foundation . . . 10
  - 2.1.1 Neural Networks . . . 10
  - 2.1.2 Convolutional Neural Networks . . . 15
- 2.2 Application: Knee Osteoarthritis Classification . . . 18
  - 2.2.1 Anatomy of the Knee . . . 19
  - 2.2.2 Knee Osteoarthritis . . . 19
- 2.3 Conventional Approaches . . . 24
  - 2.3.1 Heavy weight architectures . . . 25
  - 2.3.2 Input reduction or attention . . . 25
- 2.4 New Approach . . . 26
- 2.5 Objectives . . . 26

3 Related Work **27**
- 3.1 Image classification architectures in 2D . . . 27
- 3.2 Knee osteoarthritis classification on 2D X-ray Data . . . 27
- 3.3 Medical image analysis on 3D MRI Data . . . 28
  - 3.3.1 Knee osteoarthritis classification based on slicing . . . 28
  - 3.3.2 Knee osteoarthritis segmentation based on slicing . . . 28
  - 3.3.3 Abdominal image segmentation based on attention . . . 29
- 3.4 Thesis inspiring work . . . 29

4 Data and Preprocessing **30**
- 4.1 Data . . . 30
  - 4.1.1 The Osteoarthritis Initiative . . . 30
  - 4.1.2 Data insight . . . 31
  - 4.1.3 Scores . . . 33
  - 4.1.4 X-ray vs MRI . . . 34
  - 4.1.5 Additional Data . . . 35
- 4.2 Class Distribution . . . 35
  - 4.2.1 Distribution . . . 35
  - 4.2.2 Data imbalances . . . 37
    - 4.2.2.1 Introduce class weights . . . 37
    - 4.2.2.2 Balanced subsets . . . 37
    - 4.2.2.3 Oversampling . . . 37
    - 4.2.2.4 Data augmentation . . . 38

4.2.2.5 Alter perfomance metric . . . 38
4.3 Preprocessing . . . 38
4.3.1 DICOM standard . . . 38
4.3.2 Feature Scaling . . . 39
4.3.2.1 Per image normalisation . . . 39
4.3.2.2 Alternatives . . . 40
4.3.2.3 Application in OAI data . . . 41
4.3.3 Flipping . . . 41
4.3.4 Damaged data removal . . . 41
4.3.5 Others . . . 41

5 Method **42**
5.1 Experimental Setup . . . 42
5.1.1 Technical and Programming Details . . . 42
5.1.1.1 Hardware . . . 42
5.1.1.2 Software and Programming . . . 44
5.1.2 Metrics . . . 46
5.1.2.1 Loss functions . . . 46
5.1.2.2 Evaluation metrics . . . 48
5.1.3 Hyperparameter Optimisation . . . 51
5.1.3.1 Deep Learning Hyperparameters . . . 52
5.1.3.2 Hyperparameter optimisation . . . 60
5.1.3.3 Optuna framework . . . 60
5.2 Initial Approach: 3D CNN with inception blocks and residual connections 61
5.2.1 Architectural Details . . . 62
5.2.1.1 Baseline model in 2D . . . 62
5.2.1.2 Extension to 3D . . . 68
5.2.2 Binary classification: KLG 0 vs 4 . . . 69
5.2.2.1 Evaluation on test set . . . 70
5.2.3 Binary Classification: KLG 0 vs 2 . . . 70
5.2.3.1 Optuna: Convergence Objective . . . 70
5.2.3.2 Optuna: Multi Val-Loss Objective . . . 71
5.2.3.3 Optuna: Single Val-Loss Objective . . . 71
5.2.3.4 Evaluation on Test Set . . . 71
5.3 Data Reduction Approach: cropping region of interest . . . 72
5.3.1 Region of Interest . . . 72
5.3.2 Additional Data . . . 73
5.3.2.1 Segmentation Data . . . 73
5.3.2.2 Surface Triangulations . . . 74
5.3.3 Cropping Details . . . 75
5.3.3.1 Coronal crop . . . 75
5.3.3.2 Transverse crop . . . 75
5.3.3.3 Sagittal crop . . . 76
5.3.3.4 Center point . . . 76
5.3.3.5 Final routine . . . 76

5.3.4 Binary Classification: JSN 0 vs 3, JSN 0 vs 2 . . . 77
5.3.4.1 Binary Classification: JSN 0 vs 3 . . . 77
5.3.4.2 Binary Classification: JSN 0 vs 2 . . . 77
5.3.5 Siamese Network Approach . . . 78
5.3.6 Binary classification: KLG 0 vs 4 . . . 78
5.3.7 Binary Classification: KLG 0 vs 2 . . . 79
5.3.7.1 Optuna: Convergence Objective . . . 79
5.3.7.2 Optuna: Multi Val-Loss Objective . . . 79
5.3.8 Evaluation on Test Set . . . 79
5.4 Model Reduction Approach: learnable filter offsets . . . 80
5.4.1 OBELISK - One Kernel to Solve Nearly Everything: Unified 3D Binary Convolutions for Image Analysis . . . 80
5.4.1.1 Motivation . . . 81
5.4.1.2 Method . . . 81
5.4.1.3 Results . . . 84
5.4.1.4 Conclusion . . . 86
5.4.2 Adaption to image classification . . . 86
5.4.3 Binary classification: KLG 0 vs 4 . . . 87
5.4.3.1 Optuna: Convergence Objective . . . 87
5.4.3.2 Optuna: Multi Val-Loss Objective . . . 88
5.4.4 Evaluation on Test Set . . . 88
5.4.5 Binary Classification: KLG 0 vs 2 . . . 88
5.5 Explainable Artificial Intelligence . . . 88
5.5.1 Intention . . . 89
5.5.2 Different Approaches . . . 89
5.5.3 Saliency and GradCAM maps . . . 90
5.6 Experiments Overview . . . 91

6 Results 93
6.1 Initial Approach: 3D CNN with inception blocks and residual connections 93
6.1.1 KLG 0 vs 4 . . . 93
6.1.1.1 Evaluation on test set . . . 96
6.1.1.2 Explainable Artificial Intelligence . . . 97
6.1.2 KLG 0 vs 2 . . . 99
6.1.2.1 Convergence objective . . . 99
6.1.2.2 Multi val-loss objecive . . . 101
6.1.2.3 Single val-loss objective . . . 102
6.1.2.4 Evaluation on test set . . . 103
6.2 Data Reduction Approach: cropping region of interest . . . 104
6.2.1 JSN 0 vs 3, JSN 0 vs 2 . . . 104
6.2.1.1 JSN 0 vs 3 . . . 104
6.2.1.2 JSN 0 vs 2 . . . 104
6.2.2 KLG 0 vs 4 . . . 108
6.2.2.1 Evaluation on test set . . . 108
6.2.2.2 Explainable Artificial Intelligence . . . 109

6.2.3 KLG 0 vs 2 . . . 111
6.2.3.1 Convergence objective . . . 111
6.2.3.2 Multiloss val-loss objective . . . 112
6.2.3.3 Evaluation on test set . . . 114
6.3 Model Reduction Approach: learnable filter offsets . . . 114
6.3.1 KLG 0 vs 4 . . . 114
6.3.1.1 Convergence objective . . . 114
6.3.1.2 Multi val-loss objective . . . 116
6.3.1.3 Evaluation on test set . . . 117
6.3.2 KLG 0 vs 2 . . . 118
6.3.2.1 Evaluation on test set . . . 118

7 Discussion and Conclusion 119
7.1 Discussion . . . 119
7.1.1 Initial approach . . . 119
7.1.2 Data reduction . . . 119
7.1.3 Model reduction . . . 119
7.1.4 Comparison . . . 119
7.2 Conclusion . . . 120
7.3 Future Work . . . 121

1 Introduction

Artificial intelligence has become an indispensable technology for our modern society. After its turbulent development went through several ups and downs, latter are known as AI winters, the impact of AI concerns every aspect of our every day lives. Be it a recommendation system for an online shopping experience, weather forecasting or a face recognition that locks our phones, artificial intelligent based systems capture more and more areas in order to make them better, safer and faster. Especially deep learning based methods dominate the field in the last decades as they are supposed to learn highly complex patterns in huge data sets. Combined with impressive progress in hardware performance these data driven neural networks are able to solve problems that nobody thought before.
To be further precise, Convolutional Neural Networks (CNNs) inspired by vision neurons in our brain underwent a break through in image classification tasks in 2012 by the publication of AlexNet (see [1]). Since then highly accurate image class predictions has been achieved which at least in certain applications perform on eye level with humans. This raised interest in the domain of medical image analysis that is originally done by human experts as radiologists or physicians. Computer aided diagnosis systems were developed to support them, with the advantage that a computer system does not suffer from fatigue or poor concentration.

For instance knee osteoarthritis is of interest in such systems. Here my contribution attaches. Not only for the exemplary fact that the prevalence of this disease has doubled since the mid-20th in the US (see [2]), promising deep learning approaches has taken into account in order to improve the diagnostic investigation. CNNs are used to analyse X-ray or MRI data of knees. Different severity grades can be assessed in order to estimate the progress or apply certain therapeutic measures. Especially early knee osteoarthritis recognition is of high importance to delay or even avoid surgery. Most of former work focus on handling of two dimensional X-ray data. However, 3D images as 3D MRIs could provide more information.

This work can primarily be seen as a contribution intending new handling strategies for three dimensional data. As of today modern deep learning approaches indeed achieve remarkable results in increasingly complex tasks, but unfortunately due to the constraint of huge time and memory consumption. Especially for 3D data computational limits are exhausted more quickly than desired. That on the one hand leads to the fact that heavy weight machine learning models as convolutional neural networks can only be trained on high-end hardware that is not accessible for everyone. On the other hand, lots of research tend to avoid three dimensional solutions by shifting to the two dimensional case. Even deep learning APIs as tensorflow only provide two dimensional versions of prominent network architectures.

Keeping in mind that the best model only can process information that was fit to it, it would be rather advanced to completely use three dimensional data instead of slicing them to 2D or cutting off certain regions. Furthermore, scientific headway is driven by innovative minds. By inventing light weight 3D data handling strategies one endows thousands of motivated people with the possibility to develop algorithms that are no longer restricted to expensive hardware setups or complicated preprocessing steps. That in turn has the potential to accelerate progress in 3D data related research.

1.1 Scope and Delimitations

This short paragraph should emphasise that this work does not primarily focus on the development of new or better methods for knee osteoarthritis classification. The latter rather represents only one suitable application scenario for 3D image data handling strategies. Every other task which would profit of the usage of 3D image data would have been predestined to demonstrate my approaches.

However, the choice of knee osteoarthritis as application has a bunch of arguments in terms of medical urgency and priority.

1.2 Outline

The next chapter will on the one hand set the foundation of this work and on the other hand, it aims to motivate this contribution by demonstrating limitations that are related to conventional approaches and showing up alternatives. Both aspects, foundation and motivation, are also seen from a technical and from a medical perspective as well.
Afterwards, chapter 3 picks out certain relevant related work. The choice of publications is aims to give an overview about special former methods that either were developed in the field of medical image analysis or had inspirational influence on this work.
The following chapter focuses around the data and preprocessing. First, data their related topics are treated and then it is presented how the preprocessing should be done in order to achieve valuable input for the later developed approaches.
Chapter 5 can be seen as the center piece of this thesis. It begins with important question according experimental design that arise during the whole workflow. Section 2,3 and 4 then present the principal work. Each section portrays an approach that aims to solve image classification tasks. The first one is straight forward, the second one then shows an improvement as it would have been done is past and the third finally shows a new, innovative idea that aims to outperform the others in different aspects. All of these methods are analysed by a bunch of experiments. At the end possibilities of explainable artificial intelligence are introduced befor a experiment overview is given.
Chapter 6 then shows the results of the previously designed experiments. Finally, in chapter 7, the results are discussed before a conclusion and some thoughts about future work are given.

2 Foundation and Motivation

2.1 Technical Foundation

In the upcoming section the fundamental concepts of the used methods are presented in a quite formal manner. First, important aspects of neural networks are shown and then it gets more special by shifting to the theory of convolutional neural networks.

2.1.1 Neural Networks

Besides good intuitive explanations as in [3], for instance, now I introduce these networks as well.
The history of neural networks is quite young, its development is one of the fastest of our time. The idea of neural networks began as a model of how neurons in the brain function, termed 'connectionism', and used connected circuits to simulate intelligent behavior. That was first portrayed by neurophysiologist Warren McCulloch and mathematician Walter Pitts in 1943. In the following years one tried to translate the ideas to computational systems, the first so called 'Hebbian network' was implemented at MIT in 1954. Afterwards the history took its course involving ups and downs, the latter is known as AI winter which came in different episodes. (see [4])

There are two ways to describe the networks - intuitively accessible/graphically or formally, both are important for a clear understanding.
Artificial Neural Networks (ANNs) or for simplicity just Neural Networks (NNs) consist of several layers, i.e the input layer, hidden layer(s) and the output layer, which in turn include one or more so called 'neurons'. The following explanations are exemplarily orientated to the thesis' topic.

So for instance and in simple words the goal is to plug in a three dimensional MRI of a patient's knee into the input layer of a certain neural network, then the hidden layers do highly non-linear computations with this input and the output layer consisting of only one neuron indicates the certainty of having a diseased knee. Technically there are lots of two dimensional slices of the knee that constitute the three dimensional MRI. As two dimensional images consist of discrete pixels with specific intensities, three dimensional images have discrete 'voxels'. The used MRI data is in grayscale format, i.e. each voxel has an intensity between zero (black) and 255 (white), so the overall image is gray shaded.
Assume that there are $n \in \mathbb{N}$ voxels for the input MRI. Let $p = (p_1, \ldots, p_n) \in N^n$ denote the input vector of voxel intensities where $N = \{0, 1, \ldots, 255\}$ is the set of possible intensity values. Then the neural network can be seen as a mapping
$f : N^n \to [0, 1]$ where the codomain is the closed interval from 0 to 1. In the learning

process of the network called 'training' the goal is to successively change the mapping f to make better predictions. It is later specified how that works.

Different layers
As said before a network consists of input layer, hidden layer(s) and output layer whose elements are neurons. One can simply imagine a neuron as a placeholder for a single number that represents an intermediate value in the computation process. The input neurons were already clarified as placeholders for p_1 to p_n. The output neuron holds the certainty value in $[0, 1]$. So how does the network compute this output neuron? This calculation in the net is also called 'forward pass'. It is done layer by layer starting with the first hidden layer. Let the number of hidden layers in this case be three. So the $n = n_0$ input neurons are the input for a mapping $f_1 : N^{n_0} \to \mathbb{R}^{n_1}$ where n_1 is the number of neurons in the first hidden layer. Then the output of the first hidden layer is mapped by $f_2 : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$, this is the computation of the second hidden layer containing n_2 neurons. Analogously $f_3 : \mathbb{R}^{n_2} \to \mathbb{R}^{n_3}$ then computes n_3 neurons for the last hidden layer. Finally, the output layer $f_4 : \mathbb{R}^{n_3} \to [0, 1]^{n_4}$ ($n_4 = 1$ here) calculates the output neuron. It is to note that usually the output neuron is the placeholder for a certainty for the data belonging to a certain class. Not until the definition of a threshold for that certainty allows the model to interpret these predicted scores as a predicted class. So all in all the layer-wise forward pass can be formalized as a chain of mappings, i.e. $f = f_4 \circ f_3 \circ f_2 \circ f_1$ where each f_i ($i = 1, \ldots, 4$) represents a partial computation done by one layer within the whole network.

In neural networks these chained functions have a specific form. One describes f_i as $f_i = \phi_i \circ a_i$ ($i \in \{1, 2, 3, 4\}$):
$a_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ is an affine mapping, i.e. $a_i(x) = A_i x + b_i$ with $A_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and $b_i \in \mathbb{R}^{n_i}$, and $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ is the vectorial so called 'activation function' for the neurons of ith layer.

Affine mappings
The matrix A_i in $a_i(x) = A_i x + b_i$ is the 'weight matrix'. Each row contains weights that are linearly combined with the values of the neurons of layer $i - 1$ and then the linear combination is added to a corresponding 'bias' in b_i. To be more precise, let $v(i, j)$ denote the value of the j-th neuron in the i-th layer with suitable i and j, $v(i)$ should be a vector of all values of the neurons of the i-th layer. Then $v(i + 1, j)$ is computed as follows for all suitable i and j:

$$v(i + 1, j) = (\phi_{i+1})_j \left(\langle (A_{i+1})_j, v(i) \rangle + (b_{i+1})_j \right),$$

where index j in the right hand side means the j-th row (ϕ_i and b_i are considered to be column vectors) and $\langle \cdot, \cdot \rangle$ denotes the inner product. The activation $(\phi_i)_j$ is discussed later.
So the $(i + 1)$-th layer is basically dependent of the values of the i-th layer that are linearly combined with weights and then added by a bias. Weights and biases are so called 'learnable parameters' of the neural network. By slightly changing them one may obtain completely changed output. So the 'training' of the network is the task of optimizing these learnable parameters to get better outputs. It is described later

how this optimization works. To combine with previous notation it holds $f_{i+1}(v(i)) = (\phi_{i+1} \circ a_{i+1})(v(i)) = v(i+1)$ for $i \in \{1, 2, 3, 4\}$.

Activation functions
Besides affine mappings activation functions ϕ_i are involved. If there were none of these the whole forward pass would be a chained affine mapping. There are different reasons to introduce these activations. As many other aspects in neural networks they are inspired by biological neural networks as well. These functions endow the network with non-linearities which increase the complexity of the model so that learning of highly complicated tasks as image classification become possible. Furthermore, the activations prevent the model from computational issues as they might appear when there was no clipping of the outputs from each layer in certain bounded intervals.

There exists tons of distinct activations that all have their legitimation in their contexts. For convolutional neural networks that this work uses the following functions are essential:

1. Rectified Linear Unit (ReLU): $relu : \mathbb{R} \to \mathbb{R}, relu(x) = \max(x, 0)$
2. Sigmoid Function: $S : \mathbb{R} \to \mathbb{R}, S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$
3. Softmax Function:
   $softmax : \mathbb{R}^K \to \mathbb{R}^K, softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$ for $j = 1, \ldots, K$

They are specified later at the appropriate point when the concrete networks will be presented.

Overview of the workflow
In the following it is shortly discussed which single steps are required to obtain a network that hopefully predicts sufficiently well.

Assume that there is a bunch of labeled data given, i.e. for each data the corresponding class/label is known. Firstly one splits up this data into three independent (disjoint) subsets - training set, validation set and test set. There are different splitting methods which are applied for different contexts. A common choice for that split is for instance 60%/20%/20% for train/validation/test.
The training set is, clear by its name, the set of data which is used for the training process of the network. In other words, the network learns by this data. When the training is done for one epoch (epoch = one consideration of all data) the network is validated by investigating a certain metric on the validation set. These metrics (including loss functions) will be discussed later.
There are several hyper parameters whose changes potentially have impact on the quality of the model. The goal is to find the parameter combination that provides the best results on the validation set. When the performance on the validation set is satisfactory, one has the possibility to test the model on a third, independent set - called test set. These results are suitable for comparison with other models with the same task or for making them public. Training and validation set are less eligible for the latter

aspect since one trained/tuned the model according to these ones.

Training process
So how does a neural network actually learn? There was already given an explanation for the forward pass to the network. That is the part of prediction of the model. Since the ground truth labels for the training data are given one can measure in many ways how good the predictions were. The so called 'loss function', that has many forms and is dependent on prediction scores and corresponding ground truth labels, describes the amount of failure by the current model configuration. Internally, done by the ‚backward pass', the gradient of the used loss function is computed with derivatives with respect to the weights and biases of the network. As the overall network function f is chained this gradient computation which is done by an algorithm called 'backpropagation' starts from the last layer and then works backwards. With the computed gradient the weights are then updated in a certain way which will be discussed below.

So forward and backward pass through the network form one training step. The network engineer is able to decide how much data is considered at the same time. This is set by the parameter 'batch-size', one of the most important hyperparameters of the model.
To describe this process more formally let $\mathcal{T} := \{(x_i, y_i) : i \in I\}$ denote the training set, where I is the index set of training data, x_i are the data and y_i the corresponding labels for $i \in I$. Let further $\ell(\cdot,\cdot) : \mathbb{R}^2 \to \mathbb{R}$ be the loss function for single data prediction scores, for instance $\ell(y, \hat{y}) = (y - \hat{y})^2$. $s \in \mathbb{N}, 1 \leq s \leq |I|$ is the batch-size in the following.
The forward pass computes the averaged loss over one batch $B \subseteq I$,
i.e. $\mathcal{L}_B(\ell, f) = \frac{1}{s}\sum_{i\in B} \ell(f(x_i), y_i)$. It is to note that the network describing function f and therefore ℓ are successively updated through the training, however, for simplicity the notation will not be extended.
Now, it is computed how the learnable parameters (weights A_i and biases b_i in affine mappings $a_i(x) = A_i x + b_i \ \forall$ layers i) are updated. During backpropagation the derivatives $\frac{\partial \mathcal{L}_B}{\partial w}$ are determined for all $w \in \mathcal{W} := \{\text{all learnable parameters}\}$. The terms 'learnable parameters' and 'weights' are used synonym from now on, since by $A_i x + b_i = (A_i | b_i)\left(\frac{x}{1}\right)$ one can see the biases as part of the weight matrix.
Then the weights of the network are updated as follows:

$$w_{\text{new}} := w_{\text{old}} - \alpha \frac{\partial \mathcal{L}_B}{\partial w_{\text{old}}} \text{ for all } w_{\text{old}} \in \mathcal{W},$$

where α denotes the so called 'learning rate' of the network. The influence of different hyper parameter is discussed later. Now, one training step is completed, the network was trained on one batch B. This step is repeated batch-wise. When every single data is processed once, one epoch is trained. After each epoch the network validates the current weight-configuration on the validation data.

Metrics
Without diving too much into detail at least a little paragraph about metrics should not be missing in an introductory chapter for neural networks. While internally a loss

function is minimized there are several metrics that report the training, validation or testing.
The probably most prominent one is the accuracy which describes the percentage of data whose labels are predicted correctly by the considered model. Why this metric on its own is not sufficient in certain cases the following example should show:
Assume that there is a binary classification problem given, namely class zero versus class one. Let the trained network has an test accuracy of 95%. One might suppose that the approach has learned the desired coherence in the data but in fact the model was only trained on class one data and therefore it just predicts class one in every data. The test set here had 95% class one data and since the model always predicts class one the accuracy is very high (also 95%) though the model quality is maximally poor.
So on the one hand this example demonstrates that in some contexts accuracy is not a good choice for a metric and on the other hand it shows the possible effects of unbalanced data. Other metrics that deal with such balancing problems are for instance 'precision' and 'recall' and, further complex ones, 'F1 score' and 'AUC'. For this thesis relevant ones will be discussed later.

Universal Approximation Theorem [5]
Before we start to explore convolutional neural networks, I want to present the mathematical backbone of neural networks. It is a statement that theoretically satisfies that networks are universal function approximators. To be more precise, it was actually shown, that networks with only one hidden layer are able to approximate every continuous function. It is to note that the width of the hidden layer, however, is not bounded. There exist also dual versions with arbitrary deep networks, but here we rely on the arbitrary width case.
One of the first versions was proven by George Cybenko in 1989 for sigmoid acitvation functions [6]. Later in 19991 Kurt Hornik showed that it is not necessary to focus on specific activation functions. He stated in [7]:
If there is only one hidden layer and only one output unit, then the set of all functions implemented by such a network with n hidden units is

$$\mathcal{M}_k^{(n)}(\psi) = \left\{ h : \mathbb{R}^k \to \mathbb{R} : h(x) = \sum_{j=1}^{n} \beta_j \psi(a_j^T x - \theta_j) \right\},$$

where ψ denotes the activation function. The set of all functions implemented by such a network with an arbitrarily large number of hidden units is

$$\mathcal{M}_k(\psi) = \bigcup_{n=1}^{\infty} \mathcal{M}_k^{(n)}(\psi).$$

Then Kurt Hornik proved:
If ψ is continuous, bounded and non-constant, then $\mathcal{M}_k(\psi)$ is dense in $C(X)$ for all compact subsets X of $\mathbb{R}^k$, where $C(X)$ is the set of all continuous functions $X \to \mathbb{R}$.

In 1993, Moshe Leshno et al. could establish necessary and sufficient conditions for universal approximation (see [8]). In particular they showed, that a standard multi-

layer feedforward network can approximate any continuous function to any degree of accuracy if and only if the network's activation function is not polynomial. There are several extensions of the theorem, for instance it was refined for modern residual networks or for arbitrary width case (see [5]). A full characterization of the approximation property on general function spaces is given by Anastasis Kratsios at the end of 2020 [9].

2.1.2 Convolutional Neural Networks

In previous pages the basic principles of artificial neural networks were discovered and described mathematically. Now, it gets more special and topic related. 'Convolutional Neural Networks' as modern state of the art in image classification and other domains will be presented in the following lines.

As 'classic' artificial neural networks convolutional ones also consist of different layers which in turn comprise neurons. The notable difference is that CNNs proved to be more suitable for image-focused tasks. They allow to further reduce the parameters required to set up the model. Considering a 100×100 2D input image one already has 10.000 weight parameters (without biases) for one neuron in the first hidden layer in traditional neural networks. By shifting to 3-dimensional input with possibly even greater dimensions it is obvious that the computational power is quickly exhausted. Another drawback of networks with a big number of parameters is its tendency to overfit. Overfitting in short terms means that the model adjusts to close to the training data and loses the ability to generalize to other data. (In contrast, underfitting means, that the model is not complex enough to learn the training data.) So due to limitation of computational power and the problem of overfitting it is a good idea to keep the number of parameters smaller, this can achieved by convolutional neural networks.

The architecture of a CNN includes different types of layers which will be presented in the following. Before that it is important to understand how images are represented:

Image Representation
Here we will limit to two types of image representations - grayscale and RGB images. To start with the easier one, grayscale simply means that each pixel (voxel in 3D) of a considered image has an intensity value between 0 (black) and 255 (white), the higher the intensity the brighter the pixel/voxel gets. This range is a compromise between the efficacy of storing information about the image (256 values fit perfectly in 1 byte) and the sensitivity of the human eye (we distinguish a limited number of shades of the same color).
RGB images, however, not only have one intensity channel, but they even have three ones, each for the color red, green and blue. The color of the depicted pixel/voxel is therefore a mixture of different intensities of red, green and blue. This technique is called additive color synthesis and in this way every color becomes possible. So considering a 3D image with width w, height h and depth d one can describe a grayscale representation with $w * h * d$ integers (between 0 and 255) and a RGB representation has three channels with $w * h * d$ integers each. A suitable underlying data struc-

ture for storage of image information is a so called 'tensor'. Though they can be described mathematically precise an intuitive understanding from a computer science point of view is sufficient for the following chapters. A tensor is a generalization of vectors and matrices and can simply be seen as multidimensional arrays. A vector is an one-dimensional and a matrix is a two-dimensional tensor. Their operations such as addition and scalar multiplication can be reformulated for tensors as well. A 3D image representation can be stored as a 4 dimensional tensor - the first three dimensions correspond to the spatial dimensions of the image and the 4th dimension is the dimension for the color channels. The size of the image tensor that we will use for our purposes is $(w, h, d, 1)$ since we work with grayscale 3D images. To be more precise the neural network later works with an additional dimension for the use of data batches.

Overall architecture
The description of different layers was inspired by [10] and [11].
The CNN architecture essentially unites three types of layers - convolutional layers, pooling layers and fully-connected layers. The conventional ordering of these is that from the beginning of the network convolution and pooling alternate multiple times starting with convolution until several fully connected layers follow to calculate the output. The convolutional layers will determine the output of neurons which are connected to local regions of the input, the pooling layers will simply perform downsampling along the spatial dimensionality and the fully-connected layers work as in traditional NNs. It turned out that the rectified linear unit as activation function in the convolutional and fully-connected layers leads to promising results.

Convolutional Layer
As the name implies, the convolutional layer plays an important role in how CNNs operate. The learnable parameters of such a layer focus around the use of so called 'kernels'. I'm going to explain each layer type both in 2D and 3D case, so lets assume that the input tensor is of size (w, h, c) in 2D and (w, h, d, c) in 3D, where c denotes the number of channels (1 for grayscale and 3 for RGB images). It is to note here that actually each operation is performed batchwise so that it would be more precise to add the batch-dimension in the tensors, but for the sake of simplicity we will leave it out at this point. The extension to batchwise tensor operations is quite intuitive and straight forward.
The kernels are usually small in spatial dimensionality, but spread the entirety of the channel dimension, i.e. a quadratic kernel can be described as a (k, k, c) tensor in 2D and as a (k, k, k, c) tensor in 3D. The kernel glides through the whole input tensor in a certain way and calculates a scalar product at each point. All scalar products together then define the output for one convolution of the convolution layer, also called 'feature map' or 'activation map'. One convolution layer can perform several convolutions with distinct learnable kernels. To be more precise, we will describe that computation mathematically for the 2D case, 3D is analogous:
Let $F \in \mathbb{R}^{w \times h \times c}$ be the input tensor of shape (w, h, c) and let $K \in \mathbb{R}^{k \times k \times c}$ be a kernel tensor of shape (k, k, c). Then the corresponding feature map

$\widetilde{G} = (F * K) \in \mathbb{R}^{w_{\text{new}} \times h_{\text{new}}}$ of shape $(w_{\text{new}}, h_{\text{new}})$ is computed as follows:

$$\widetilde{G}[m,n] = (F * K)[m,n] = \sum_{l=1}^{c} \sum_{i=1}^{k} \sum_{j=1}^{k} K[i,j,l] F[m+i, n+j, l]$$

for all $m \in \{1, \ldots, w_{\text{new}}\}$ and $n \in \{1, \ldots, h_{\text{new}}\}$ with $w_{\text{new}} = w - k + 1$ and $h_{\text{new}} = h - k + 1$. So this holds for one specific convolution that is part of the whole output of the convolutional layer. Assume that $\mathcal{K}$ is a family of o kernels that are given in the convolutional layer. Then the overall output tensor $G \in \mathbb{R}^{w_{\text{new}} \times h_{\text{new}} \times o}$ results from $G[\cdot, \cdot, p] = F * K_p$ for all $p \in \{1, \ldots, o\}$.

So regarding the number of weights that has to be optimized we only have $k \times k \times c$ parameters for one kernel respectively $k \times k \times c \times o$ for all kernels in 2D case. Assuming that we have RGB images of size $64 \times 64 \times 3$ we can see huge differences in the number of parameters in ANNs and CNNs. If we had 16 kernels of size 6 in a convolutional layer we would only have $6 \times 6 \times 3 \times 16 = 1,728$ learnable parameters for the whole layer and in a fully connected layer there would be $64 \times 64 \times 3 = 12,288$ weights for each neuron. In 3D case the discrepancy between these numbers even gets dramatically higher.
In particular, a convolution without 'padding' and 'striding' was formalized in the previous equations. Padding means, that we expand the spatial dimensionality of the input volume in a certain way, for instance we pad the borders with zero-values ('zero-padding'). By this technique we can avoid shrinking of the output dimensions and, another reason, we can tackle the problem that the convolution puts more importance on the center pixels. The stride of a convolution is the step size in which the kernel glides through the input volume. In the equations above the stride was equal to 1. By changing this, again the dimensions of the feature map alter and, furthermore, the overlapping of the kernel locations for computing scalar products gets smaller. To generalize even further one could also specify the stride for each spatial dimension. By introducing the padding size p and the stride length s the following formulas for the output dimensions hold:

$$w_{\text{new}} = \left\lfloor \frac{w + 2p - k}{s} + 1 \right\rfloor$$
$$h_{\text{new}} = \left\lfloor \frac{h + 2p - k}{s} + 1 \right\rfloor$$

There are lots of further concepts for convolutions as grouped convolutions, use of biases, shared weights and so on that we do not discuss here.

Pooling Layer
These layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model.
A pooling layer operates over each feature map in the input and scales its dimensions using a function that is previously specified by the machine learning engineer.

Commonly implemented candidates are max-pooling or average-pooling functions.
Again a kernel of certain dimensions that is spatially small in contrast to the input dimension but has the same depth is applied with a certain stride along the input volume. Usually CNNs use max-pooling layers with 2×2- kernels and a stride of 2 to scale the activation maps down to 25% of the original size. Overlapping pooling is also possible, for instance by 3×3- kernels and a stride of 2. Due to its destructive nature pooling with kernel sizes above 3 will usually greatly decrease the performance of the model.
For getting more formally we introduce new notation:
For a tensor $T \in \mathbb{R}^{l \times m \times n}$ and natural numbers $i < i', j < j', k < k'$ we denote a subtensor by $T[i : i', j : j', k : k'] \in \mathbb{R}^{(i'-i)\times(j'-j)\times(k'-k)}$ which is obtained by slicing the original tensor T in the following way: All elements with first dimension
less than i or greater than or equal to i' are cut off, all elements with second dimension less than j or greater than or equal to j' are cut off and, analogously, all elements with third dimension less than k or greater than or equal to k' are cut off.
So by this notation it is easy to describe the pooling operation (again in 2D case, 3D is analogous). Let $F \in \mathbb{R}^{w \times h \times c}$ be the input volume. Let further $k \in \mathbb{N}$ be the kernel size (assuming a quadratic kernel) and $s \in \mathbb{N}$ be the stride. For a pooling function f (as maximum or average) the output tensor $pool(F, k, s, f) \in \mathbb{R}^{w_{\text{new}} \times h_{\text{new}} \times c}$ of a pooling layer is computed as:

$$pool(F, k, s, f)[m, n, d] = f(F[1+(m-1)s : 1+(m-1)s+k, 1+(n-1)s : 1+(n-1)s+k, d])$$

for all $m \in \{1, \ldots, w_{\text{new}}\}$, $n \in \{1, \ldots, h_{\text{new}}\}$ and $d \in \{1, \ldots, c\}$ with $w_{\text{new}} = \left\lfloor \frac{w-k}{s} + 1 \right\rfloor$ and $h_{\text{new}} = \left\lfloor \frac{h-k}{s} + 1 \right\rfloor$. It is to note, that we can again introduce padding strategies into that operation which is analogous to its explanation for convolutions.

Fully Connected Layers
Fully connected layers or dense layers in CNNs work the same way as they work in traditional ANNs. They contain a certain number of neurons that are directly connected to the neurons in the two adjacent layers. It is to note that the input and output of these layers are one dimensional tensors respectively vectors but the output of a convolution-pooling chain is a three dimensional tensor (for 2D images, without batch dimension) or a four dimensional tensor (for 3D images, without batch dimension). To deal with that mismatch the feature volume is flattened, i.e. all elements are rolled out into a vector. To come to a final output for instance in a classification task, the last dense layer has as many neurons as classes that can be predicted.

2.2 Application: Knee Osteoarthritis Classification

This section introduces the application case where the presented methods should lead to improvements, namely knee osteoarthritis classification. It is to note here that this application is chosen only exemplarily and for the purpose of demonstration of the invented ideas. This work is not limited to that scenario and rather aims to deliver general 3D data handling strategies for CNNs. However, there were good reasons to

exactly choose that topic which the following is going to motivate from a more medical perspective.
The following medical information is obtained from [12], [13], [14], [15], [16], [17].

2.2.1 Anatomy of the Knee

The disease that we are going to examine is located in parts of knee joints. For this reason we want to take a quick glance at the anatomy of this joint which is the largest in the human body.
The knee which is the connection between the upper and lower leg joins different bones and cartilages. The bone of the thigh is called 'femur' and the bone that comes from the lower leg is called 'tibia'. The so called 'patella' or 'kneecap' is a flat and rounded triangular bone which articulates with the femur. It covers and protects the anterior articular surface of the knee. Furthermore there is a 'meniscus' that partly divides the joint cavity. Actually we could further subdivide into a lateral and a medial meniscus. 'Lateral' is a medical term of location that describes something to the side and, in contrast, 'medial' always means something close to the midline of the body. These menisci have the function of dispersing the bodyweight and reducing friction during movement. Besides this important cartilage we also have femoral and tibial cartilage which dramatically vary from person to person.
It should be mentioned here that there are more structures such as ligaments and other bones involved into knee anatomy, but for our purposes we only need the stated bones and cartilages. The physical properties of all together permit flexion and extension of the leg as well as slight internal and external rotation.

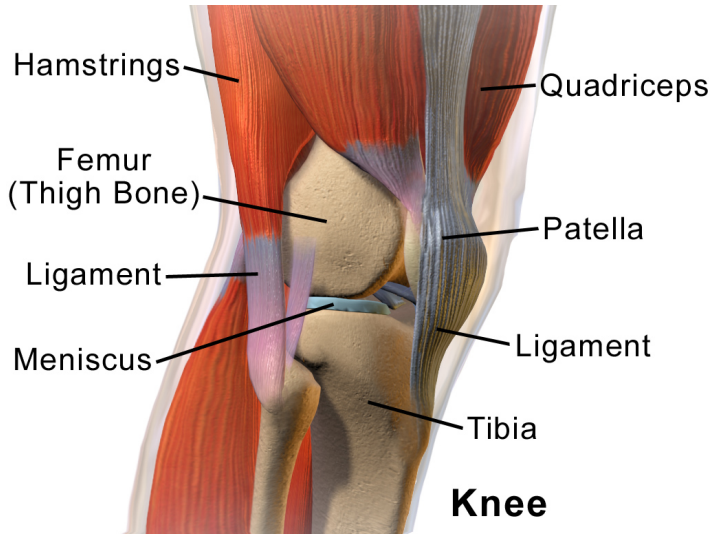


Figure 2.1: Knee Anatomy including muscles [18]

2.2.2 Knee Osteoarthritis

In the following, I am going to provide a small medical insight into the examined diesease - osteoarthritis in the knee joint.

Knee osteoarthritis is by far the most common type of arthritis. Other less common types for instance are rheumatoid arthritis, pseudogout or reactive arthritis. Osteoarthritis is specified by degeneration of articular cartilage. This condition also includes changes to the bone underneath the cartilage. Moreover it can affect nearby soft tissues.

Characteristics

To be more precise we will consider changes in the knee that possibly occur:

Cartilage → weakens, becomes damaged, attempts to heal; Meniscus damage

Bones → Bone spurs: abnormal growths, Subchondral bone sclerosis: hardening of femur/tibia, Cysts and bone marrow lesions

Joint Space → narrows

Joint fluid → composition changes, amount changes

Soft tissues → tendons and ligaments may become strained

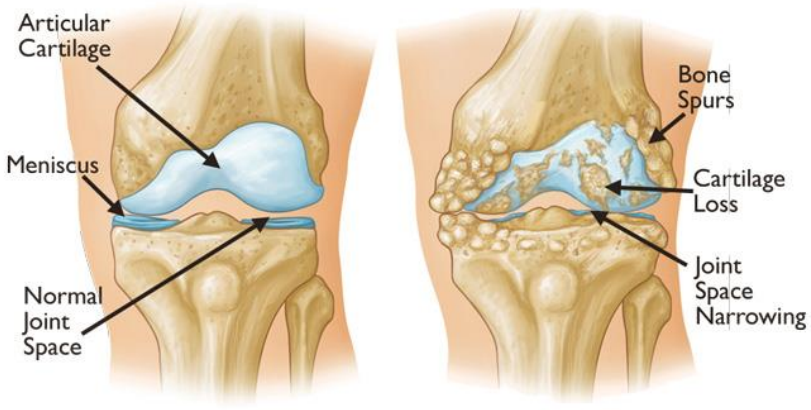


Figure 2.2: Healthy vs Diseased Knee [19]

These different types of change in the joint cause several symptoms from which patients suffer.

Symptoms

In lots of cases, the symptoms come and go, whereby it should be noted that frequency and intensity increases over time, especially if the disease is left untreated.

Knee pain
most reported, temporary or chronic, dull and aging or sharp and intense, depends on

activity

Swelling
irritation may cause production of excess joint fluid

Stiffness
friction and joint swelling cause stiffness, often for about 30 min in the morning or after sitting

Redness and warmth
skin over the knee may become red and warm, could also indicate an infection

Reduced range of motion
range of motion may become limited, less flexible, diffculty to completely bend or straighten the knee

Worsening symptoms with inactivity
knees become stiff and painful when inactive, most noticeable after sleep or long period of sitting

Popping or crunching
feel or hear when bending the knee (such as squatting), 'crepitus' is the medical term

Buckling or locking up
knee gives out or buckles

A treacherous problem in early recognition of such a disease is its potential dormant character. If there is an early diagnose of osteoarthritis possible then there is a good chance to dramatically slow or even eliminate the progression of the symptoms by an appropriate treatment.

Causes

Unfortunately it is not known nowadays which the exact cause is. It is rather supposed that several risk factors increase the likelihood of developing arthritis.

Not going too much into detail I will give a short glimpse into some important factors:

- advanced age (37% of over 60 years old are affected) *fußnote
- obesity, i.e. BMI ≥ 30 (twice as likely)
- joint trauma ('post-traumatic arthritis')
- family history (experts estimate between 40% and 65% of cases influenced by genetics)
- illness or congenital defect (gout, septic arthritis, metabolic disorders, poor bone alignment, congenital conditions)

- joint stress and chronic injury (when lot of time on feet, heavy lifting, high-impact sports, accumulated 'mini-traumas')
- lack of exercise (too little stress, fluid with nutrients has to circulate)
- poor muscle tone (hamstring, quadriceps, calf muscles too weak)
- female sex (about 40% more likely)
- biochemical changes (not clear if cause or effect of arthritis)

One has to notice at this point that advanced age and obesity are the two chief risk factors.
Furthermore it should be noted, that these properties are not absolute. One may develop the disease while fulfilling only one of them and on the other hand one may not develop arthritis while having all of them.

Diagnose

Right now there is no standard single laboratory test to verify the presence of knee osteoarthritis. There are more or less five diagnostic tools to investigate the patient for determining the medical condition:

1. Patient interview
2. Physical exam
3. X-ray $\rightarrow$ examine joint space, bone spurs ('osteophytes')
4. MRI $\rightarrow$ additional view of soft tissue
5. Lab tests for other conditions

Here one has to keep in mind that most people over age 50 have visible signs for osteoarthritis in X-ray, but many will have no symptoms.
This data is only one diagnostic tool which is not sufficient. There are patients with significant marker in X-ray but without pain and otherwise, there are patients with marginal signs in X-ray with lots of pain symptoms.

For distinct reasons that will be discussed later on MRIs are only used when X-ray is inconclusive or if the doctor suspects that there is another disease involved.

During the diagnostic process all possible causes of knee pain are considered. A gradual and systematic elimination of possible diseases has to be done until an accurate diagnosis can be made.

Treatment

Since this work aims to improve the conditions of diagnosing knee osteoarthritis more effectively and more accurately, it is not necessary for our purposes to know the therapeutic measures to deal with this disease in detail. For the sake of completeness we will however shortly mention some therapy possibilities.
Overall the treatment may include nonsurgical treatments, injections and surgery. Due to the fact that artificial knees cannot replace each aspect of a natural joint one obviously tries to avoid surgery and nonsurgical treatments are preferred.
Treatment options:

- Physical therapy
- Pain Medications
- Lose weight
- Injections
- Surgery, most common: total knee replacement

Most pharmaceuticals just treat the symptoms and not the cause.
Not until first-line treatments such as losing weight and physical therapy do not sufficiently relieve pain the physician may suggest injections. Surgery is the last step to decide for.

Prevalence

After these short medical glimpses into the characteristics, symptoms, causes and treatment strategies we will have a look into statistics which offer a better understanding about how many persons really suffer from knee osteoarthritis.
The following numbers were evaluated in a current meta-analysis (see [20]) based on 88 eligible studies with 10,081,952 participants in total. They found that the pooled global prevalence of knee OA was 16% (95% confidence intervall: 14.3% - 17.8%) in individuals aged 15 and over and was 22.9% (95% CI: 19.8% - 26.1%) in individuals aged 40 and over. The latter means that there are around 654 million invididuals (age over 40) with knee osteoarthritis in 2020 worldwide. Only these numbers dramatically show the need of early recognition of this disease to help such an huge amount of people.

Furthermore the mentioned meta analysis observed noticeable differences in the prevalence by countries or even continents. At continent-level, the prevalence was higher in Asia (19.2% [95% CI: 15.7% - 23.0%]) than in Europe (13.4% [95% CI: 10.1% - 17.0%]) and North America (15.8% [95% CI: 11.2% - 20.9%]). The lowest prevalence was found in South America (4.1% [95% CI: 2.1% - 6.9%]). There unfortunately was not enough adequate data for conclusions in Africa or Oceania.
For the differences at country-level you can look at the figure below.

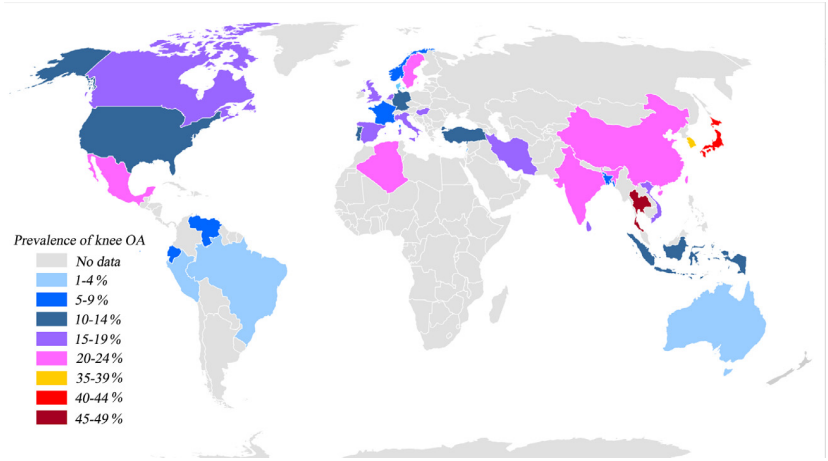


Figure 2.3: Prevalence over Countries [20]

We want to conclude this section with the correlation between prevalence and age. In the figure below the almost linear relation gets clear. More than 30% of individuals older than 70 are affected which further substantiates the fact that knee osteoarthritis is a vast danger for the health of human beings.

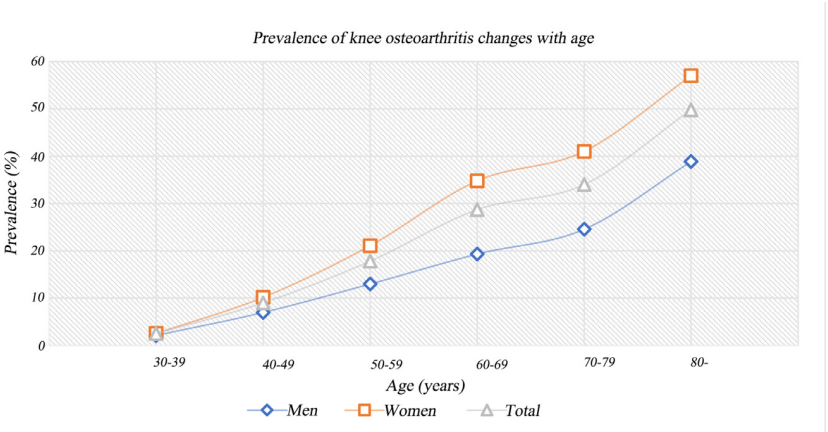


Figure 2.4: Prevalence over Age [20]

This concludes the medical introduction that is sufficient for our purposes. A big strength of data-driven approaches as neural networks is that actually no prior expertise in the considered topic is necessary. In practice and especially if you want to collaborate with medical practitioners it is of advantage to know certain vocabulary that was given in that foundation.

2.3 Conventional Approaches

After several foundations from medical and technical point of view, I want to shortly illustrate how the common trend in terms of deep learning based image data handling looks like, especially for medical image analysis. In the next chapter these general impressions are underpinned by concrete publications.

2.3.1 Heavy weight architectures

Theoretical driven by the fact that neural networks are universal function approximators assuming arbitrary width or depth, the models in practice became more and more complex by incorporating more and more neurons. This truly improves the quality of the networks. Successful architectures not only have thousands of parameters any longer, they even involves millions and hundreds of millions of learnable parameters. Since the evolution of hardware components is fast as well that does not directly lead to issues. Lots of average GPUs (which are central when it comes to train neural networks) are able to handle these big models. But the problem is that the latter actually only holds for two dimensional image data. By shifting to three dimensions GPU memory issues occur. Either one drastically downsamples the input volume or the batch size to not crash the GPU. Both of the variants are not satisfying since they have dramatic impact on the model performance.
Besides memory limitations the model training time obviously increases by the number of learnable parameters. In practice that could mean days to weeks for model training that enormously reduces the applicability. Furthermore it induces high energy consumption.
Especially in medical image analysis, 3D images as MRIs play an important role. In context of knee osteoarthritis classification this work aims to present an intitial approach incorporating a heavy weight CNN. It is shown in which way the described issues arise.

2.3.2 Input reduction or attention

When it comes to 3D image processing the prevalent strategy to tackle the memory or time limitations is to reduce the input volume. The most simple approach is to only pick a subset of 2D slices from the whole input volume. One could also process certain 2D slices for each volume in parallel architectures. Both strategies unfortunately yield a loss of spatial information. Therefore, a more sophisticated approach preserving spatial information would be the cropping of a 3D region of interest (ROI). This method, however, requires additional complicated steps to determine the exact ROI.
There are on the one hand approaches that incorporate domain knowledge. This causes time to educate oneself further or experts has to be acquired for joined work. On the other hand, also deep learning based methods for determination of important regions could be acquired.

Another technique is to learn to focus on a subset of voxels distributed over the whole image volume. Among other things, modern research develops so called attention mechanisms for CNNs that force the models to more focus on the important regions. This, in turn, boosts the models performances while one also tries to preserve the computational efficiency.

Nevertheless, additional time and effort is required for these approaches.

Followed by the initial approach, this work presents a cropping procedure with the aim of tackling time and memory constraints.

2.4 New Approach

As we saw in the last section the conventional approaches all have their drawbacks. So the goal is to explore alternatives. Keeping this aim in mind a publication from 2018 presented an innovative idea. They could develop a completely new kind of convolution that is able to deliver network architectures with only thousands of parameters that simultaneously achieve remarkable performance. This idea is presented in context of medical image segmentation.

Inspired by that, this work shows an adaption of the innovation into the context of knee osteoarthritis classification.

Additionally, but only as minor subject, simple explainable intelligence approaches are investigated. It should rather be seen as proof of concept and motivation for future work.

2.5 Objectives

The following bullet points summarise the motivation of this work, stated as questions. The conclusion chapter at the end aims to find answers.

- Is a big CNN architecture that is successful for 2D data simply extendable to 3D in context of knee osteoarthritis classification? How does it perform in 3D?
- What exactly are the technical limitations when it comes to 3D data processing by heavy weight architectures?
- What could be a suitable ROI cropping approach in context of knee osteoarthritis classification? How does it perform?
- Is the alternative idea that was successful for medical image segmentation also valuable in context in knee osteoarthritis classification?
- Which ways exist for hyperparameter optimisation to tackle time constraint?
- Which are the most important hyperparameters and how do they influence the model training in each approach?
- Are there simple approaches to make the models explainable?

3 Related Work

In the following chapter certain related work is presented. To begin with, prominent image classification architectures in two dimension that seems promising to extend to three dimensions are shown. Then a specific deep learning approach for knee osteoarthritis classification on 2D X-ray data follows. Afterwards, different conventional strategies in 3D data handling are portrayed. Finally the principal paper that inspires this work is introduced.

In order to get an overview about 3D deep learning on medical images, see [21].

3.1 Image classification architectures in 2D

In that section I want to refer to different architectures that were iteratively developed by Google Inc. They addressed several issues that arise when it comes to image classification tasks.

First, they published the Inception net version 1 in the prominent paper "Going deeper with convolutions" in 2014 [22]. Here an innovative inception module was incorporated into a convolutional neural network. That is able to capture objects belonging to the same class but given in different scales respectively zoom levels.
In 2015 their ideas further improved by altering the inception blocks essentially. Thus, "Rethinking the Inception Architecture for Computer Vision" was published [23].
Inspired by the success of residual connections they finally invented two verions of a combined Inception-Resnet, described in "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" in 2016 [24].

The Inception-ResNet v2 is the basis for my initial approach. The successive evolution of its preliminary versions is explained in furhter detail in the method chapter.

3.2 Knee osteoarthritis classification on 2D X-ray Data

Let's stick to the application scenario of knee osteoarthritis classification. The majority of work is done on two dimensional X-ray data, yet.

Exemplarily, the work of Yassine Nasser et al. in 2020 [25] shows impressive progress. In their paper "Discriminative Regularized Auto-Encoder for Early Detection of Knee OsteoArthritis: Data from the Osteoarthritis Initiative" they devoloped an auto-encoder that aims to generate feature representations that are maximally discriminative as

depicted in the figure below. These feature representations are then classified using different classifiers. This approach yielded a KLG 0 vs 2 prediction accuracy of 82.5%.

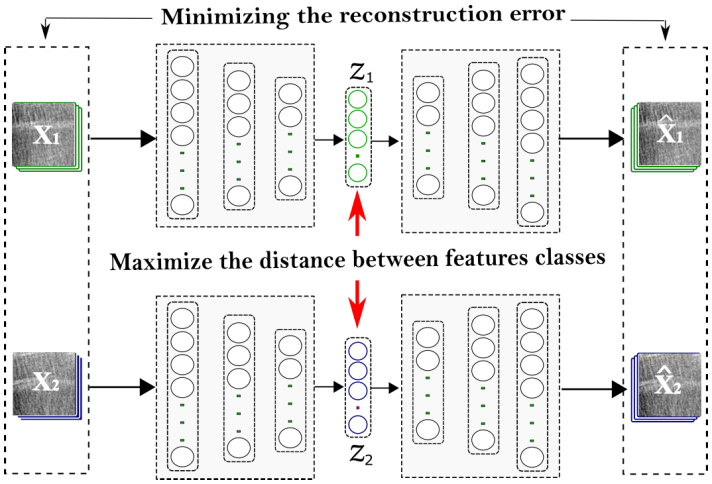


Figure 3.1: Discriminative auto encoder

3.3 Medical image analysis on 3D MRI Data

Now, methods for 3D MRI analysis are presented.

3.3.1 Knee osteoarthritis classification based on slicing

"Knee Osteoarthritis Classification Using 3D CNN and MRI" is the title of the publication from 2021 [26], that should serve as an example for that subsection. It was developed by Carmine Guida et al. Their work removes slices in order to reduce the input volume (see figure below). As a binary case, they distinguished between no arthritis cases (KLG less or equal than 1) and arthritis cases (KLG greater or equal to 2). The intuition for that separation is the aim of providing early recognition approaches. This work achieved 83 % test set accuracy.

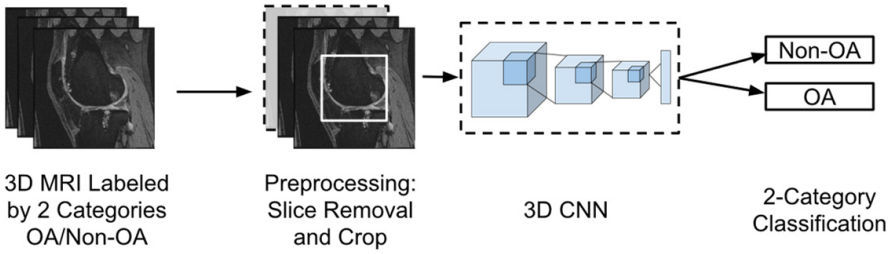


Figure 3.2: 3D CNN with slice removal

3.3.2 Knee osteoarthritis segmentation based on slicing

Here I shortly want to name the contribution that directly affects my work, since it provides me with segmenation data that is used in one of the developed approaches.

It is the work of my direct contact partner Alexander Tack and supervisor Dr. Stefan Zachow et al. from 2018 called "Automated Segmentation of Knee Bone and Cartilage combining Statistical Shape Knowledge and Convolutional Neural Networks Data from the Osteoarthritis Initiative" [27]. It is later introduced in more detail.

3.3.3 Abdominal image segmentation based on attention

"Attention Gated Networks: Learning to Leverage Salient Regions in Medical Images" [28] is a paper from Jo Schlempner et al. and was published in 2019. Among other things, it shows the beneficial influence of attention mechanisms in context of 3D CT abdominal segmentation. To be more precise, they proposed an attention gate, that is adaptable in any existing CNN for classification and segmentation tasks. This scales input features with certain attention coefficients that are computed by this gate. By incorporating that mechanism into a standard U-Net architecture, for instance, a boost in prediction performance could achieved while computational efficiency is preserved.

3.4 Thesis inspiring work

At this point I only want to name the corresponding publication, since the method chapter precisely presents this work. It is titled "OBELISK – One Kernel to Solve Nearly Everything: Unified 3D Binary Convolutions for Image Analysis" that was published by Matthias Heinrich et al. in 2018 [29].

4 Data and Preprocessing

After the motivation was set, technical and medical basics for that work were given and related work was discussed, now, the data that we are going to process is presented together with its corresponding labels. Before the actual processing in the developed approaches is possible, some important preprocessing steps has to be applied which are portrayed at the end of this chapter.

4.1 Data

This section is dedicated to the core piece that provides useful information we want to extract - the data itself.
First we will look at the study where the MRIs are acquired from. Then some examples of the database are presented and the central labels (medical scores) that we are going to predict are explained. It follows a little discussion about advantages and disadvantages from MRI respectively X-ray data and this section closes with a short glimpse into additional data, that was necessary for certain approaches.

4.1.1 The Osteoarthritis Initiative

The Osteoarthritis Initiative (OAI) [30] is a multi-center, ten-year observational study of men and women of the United States. It is sponsored by the National Institutes of Health (part of the Department of Health and Human Services). The goal of this study is to provide resources to enable a better understanding of prevention and treatment of knee osteoarthritis.
On the website https://nda.nih.gov/oai I got access to this study for scientific purposes. This permanent archive contains clinical data, patient reported outcomes, biospecimen analyses, quantitative image analyses, X-Rays and MRIs which were acquired during the study. It includes longitudinal assessments and measurements from 4,796 subjects, with data from over 431,000 clinical and imaging visits. Almost 26,626,000 images can be found in that archive. More than 400 research manuscripts that have already been generated based on this data emphasise the quality of the study and the interest in the corresponding research area.

After excluding some damaged data I could collect 40,393 MRIs including left and right knees from 4,347 subjects. The visual MRI data was generated over seven distinct timepoints - at baseline and then after 12, 24, 36, 48, 72 and 96 months follow up. These distinct timepoints for same subjects allow for instance to investigate the progression of knee osteoarthritis in further studies.

4.1.2 Data insight

Now, we want to look at the original MRI data how it was generated by the MRI scanners. Visualization respectively image processing software provides useful tools to manually get an impression of the data that one is working with. I used Amira for Life and Biomedical Sciences by ThermoFisher for that purpose. One can also view the 3D image volume as well as 2D slices from each perspective. Since the 3D volume is non-transparent it is rather improper for visualisation. Therefore certain 2D slices of the three perspectives will be shown. The perspective names will be oriented at medical terms here [31]:

- sagittal/longitudinal view: dives body into left and right halves
- coronal/frontal view: dives body into front and back halves
- transverse/ horizontal view: dives body into top and bottom halves

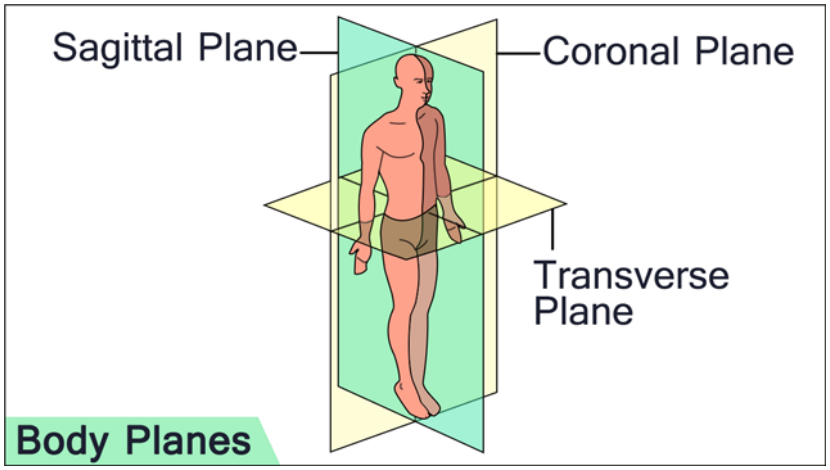
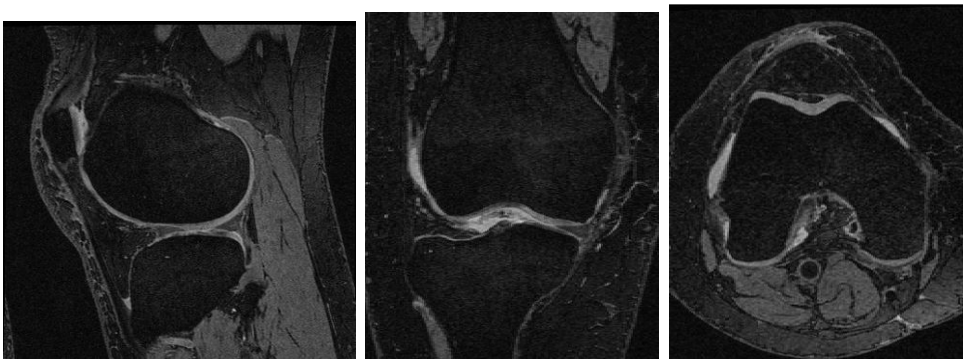


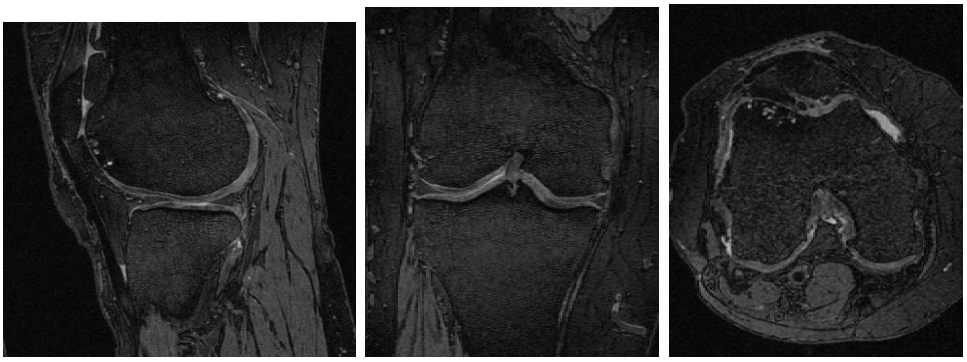
Figure 4.1: Sagittal, coronal and transverse plane [32]

So we are going to start with a healthy knee, followed by a knee with minimal and then severe arthritis.



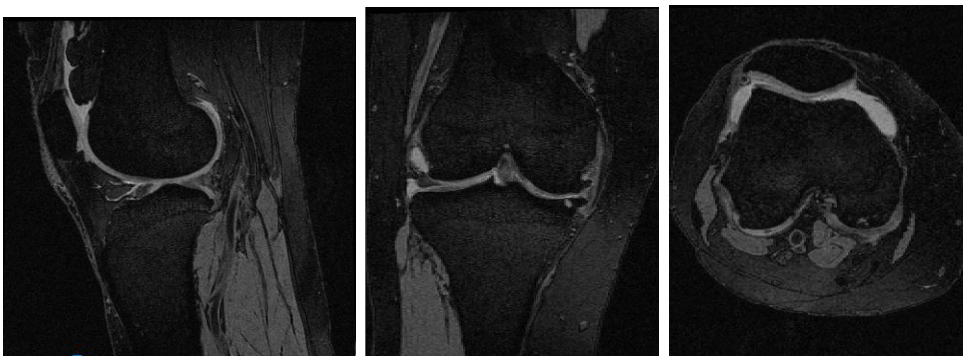
(a) sagittal view (b) coronal view (c) transverse view

Figure 4.2: different views of healthy knee



(a) sagittal view (b) coronal view (c) transverse view

Figure 4.3: different views of knee with minimal arthritis



(a) sagittal view (b) coronal view (c) transverse view

Figure 4.4: different views of knee with severe arthritis

By looking for example at the coronal view of the severely diseased knee one can see definite osteophytes close to the connection between tibia and femur bone (small bright dots at right hand side) and bone deformities (especially left hand side).

4.1.3 Scores

The labels that we are going to predict are medical scores. In medicine a 'score' is a numeric value which is determined as part of an assessment of a patient. These score computations are based on specific diagnostic parameters (see [33]).
The purpose of such a value, at least in our case, is that one gets a rough classification of different severity levels without knowing any background information about the patient.
The most essential radiographic score in the context of knee osteoarthritis is the so called 'Kellgren-Lawrence-Score' or 'Kellgren-Lawrence-Grade' (KLG). In the following we stick to the definitions in [34].

Kellgren-Lawrence-Grade
In 1957, Jonas H. Kellgren and John S. Lawrence developed this score. It is radiologically surveyed and designed to divide the severity of knee osteoarthritis into five different levels. The grade, based on conventional X-ray data, is easily applicable, well reproducible and is said to be recognised as standard for evaluation of epidemiological studies by World Health Organisation.

To be precise, there unfortunately exist different definitions or different ways to determine the grade. Here we will orient towards an atlas that was published in 1963 by Kellgren, Jeffrey and Ball. The following marker are involved:

- evidence of osteophytes (= circumscribed new bone formation)
- joint space width
- increased subchondral sclerosis (= densification of bone structures underneath articular cartilage)
- deformity of joint-building parts of bones

We will abbreviate osteophytes by OP, joint space narrowing by JSN, subchondral sclerosis by subScler and deformity of joint-building parts of bones by deformJoint. Different combinations and expressions of these characteristics result in different grades as follows:

level of severity	radiological report
0: no arthritis	normal finding
1: doubtful arthritis	doubtful JSN, discrete OP ('osteophytic lipping')
2: minimal arthritis	definite OP, possible JSN
3: moderate arthritis	multiple OP, JSN, small to moderate subScler, possible deformJoint
4: severe arthritis	large OP, progressed JSN, severe subScler, deformJoint

It is crucial to detect the early stages (grade 1 and 2) as soon as possible. Then one can timely initiate suitable treatment so that the prognosis gets better.

The Joint Space Narrowing (JSN) as a subscore of Kellgren-Lawrence-Grade (KLG) does also play a role in our investigations later.
It is to emphasise here that KLG and JSN are analysed on X-ray data nowadays. Our models, however, are based on MRI data.

4.1.4 X-ray vs MRI

See [35], [36] and [37] for the following information about these data types.
Before we are going to point out important differences we will shortly introduce the different technologies behind these data types:

X-ray
X-rays are electromagnetic radiation that differentially penetrates structures within the body and creates images of these structures on a photographic film or a fluorescent screen. This radiation passes easily through air and soft tissue of the body but it is stopped by dense material, such as tumor or bone. The greater the density of the material that the X-rays pass through, the more rays are absorbed. For this reason different types of tissue become different colors in the final picture. Dense material shows up as white, while softer tissue shows up as shades of gray and airspaces look black.

MRI
Magnetic resonance imaging (MRI) is a scanning technique for creating detailed images of the human body. It uses a strong magnetic field and radio waves.
The human body mainly consists of water. Water molecules contain hydrogen nuclei (protons), which become aligned in a magnetic field. The protons absorb the energy from the magnetic field and flip their spins. When the field is turned off the protons gradually return to their normal spin. This process generates measurable radio signals which can be converted into pictures. The beneficial property of an MRI according to its diagnosing ability is the exquisite depiction of soft tissue and anatomic detail.

So we got a small overview of these two different technologies. In the previous chapter we saw that X-rays are the first choice for diagnosis. Not until that X-ray image is not enough, a MRI can be taken into account. The reasons for that prioritization are quite simple:
On the one hand, there is a huge difference in the cost of these techniques. While X-raying of tibia, femur and knee-joint costs between 26.84 and 33.39 Euro in Germany one has to pay between 140 and 350 Euro for a knee MRI. It is quite obvious that the health insurances prefer the more than four times cheaper X-rays.
On the other hand, there are striking differences in required time for the scanning process. The X-ray scan lasts only a few seconds and the MRI needs about half an hour. Additionally one has to say that patients with specific metal implants are not suitable for MRI imaging due to the strong magnetic field. Another problem with MRI are movement artifacts, but there exist various methods to deal with that.
These are the main arguments to decide for the X-ray investigation. Let's see the other side:

To begin with, we consider the diagnose related point of view. The quality respectively the amount of information that the data can provide is undoubtedly better with MRIs. In particular, the soft tissue details improve the assessment of the pathology the doctor wants to detect. One has furthermore the possibility of generating three-dimensional image data (3D is constructed by generating several 2D slices of the knee). This is not possible with X-rays. Now we will come to an aspect, which on his own leads to the consideration to completely avoid X-ray imaging in unnecessary situations in the future. X-rays emit dangerous radiation which cannot only cause different diseases, but rather has the potential of altering the DNS which could for instance lead to birth defects. In contrast, MRI has no biological hazards, its application is completely safe.

To conclude these comparisons, there is a difference in the position of the patient in the moment of recording. While in X-ray examination the patient usually stands and therefore the knee is under load, in MRI scanners the patient lies, so the knee is not under load. As a result, the Joint Space Narrowing is not visible in MRIs, at least not directly.

All in all, in my point of view a diagnosis tool which maybe affects the health of the patient should not be longer available in a modern medical system where evidently better alternatives as MRI exist. Seen from this perspective one can understand this thesis as a further argument to supersede X-ray by MRI, at least in the context of knee osteoarthritis.

4.1.5 Additional Data

For cropping procedures that I use later I could acquire segmentation masks together with surface triangulations for all MRIs involved. This data arises from former research from the work group that I did an internship in in summer 2020 and which is supervising this master thesis. They used a method for the automated segmentation of knee bones and cartilage that combines a priori knowledge of anatomical shape with convolutional neural networks. Their approach incorporates 3D Statistical Shape Models (SSMs) as well as 2D and 3D CNNs to achieve a robust and accurate segmentation of even highly pathological knee structures. There are given more details about this data at the appropriate point in this thesis.

4.2 Class Distribution

The following section is dedicated to the labels. We will examine its distribution which will show huge imbalances. Afterwards strategies to deal with that imbalancing problem are given.

4.2.1 Distribution

To begin with it is needed to split the data into training, validation and test set. For that procedure it was crucial to keep knees from the same patient in the same subset, since these knees are likely to be very similar although they are recorded at distinct timepoints. According to that assumption I decided to set the portions of patients to

be 70%/10%/20% for train/validation/test. The reason for picking such a huge amount for the training set is that the underlying problem was said to be really complicated so the models need lots of different data to understand the abstract coherences. A further motive is that the networks are really complex, such that a big number of training samples is necessary in order to avoid overfitting. Due to that split of patients these numbers of MRIs hold for each subset:

training set	29,163 samples	72.2%
validation set	4,293 samples	10.6%
test set	6,937 samples	17.2%

In the circle diagrams below one can see how the portions of each class of KLG grade are - each for training, validation and test set.

(a) training set (b) validation set (c) test set

Figure 4.5: KLG 0 (blue), KLG 1 (red), KLG 2 (yellow), KLG 3 (green), KLG 4 (orange)

As can be clearly seen the distribution in each set is almost the same. Healthy knees (KLG 0) are overrepresented in each set and the severely diseased knees are very underrepresented. Cases of minimal and doubtful arthritis (KLG 1 and KLG 2) almost equally occur and moderate arthritis (KLG 3) is a bit less represented than the latter.

Similar results were obtained for the Joint Space Narrowing as depicted in the circles below. In this work we limit to the **medial** JSN. It is to note that these numbers are based on the count in data directories where the cropping procedure already has been done for the approach that is presented in chapter 8. Here the total numbers for training, validation and test set are slightly less (about 1,000 samples less in total) since for certain reasons some data had to be excluded.

(a) training set (b) validation set (c) test set

Figure 4.6: JSN 0 (blue), JSN 1 (red), JSN 2 (yellow), JSN 3 (green)

The portions of each class again are similarly distributed, but here the healthy knees are even more dominant in contrast to the KLG distributions.
As we are confronted with dramatic data imbalances one has to introduce methods to tackle these.

4.2.2 Data imbalances

Here I am going to show potential causes of this problem if it would left untreated and which possibilities exist to improve this setting.

So what is the actual problem with data imbalances? Concerning our case there are lots of healthy knees and very few severely diseased knees. That means if we would plug in the whole training set then the model is influenced by the imbalanced distribution and it would more likely predict healthy just due to the fact that most of the knees it saw in training were healthy. A further problem is that prominent metrics as the accuracy are no longer valuable. A model with high accuracy that only predicts healthy and did not learn any coherence is not a good model. So certain metrics as accuracy cannot be used to demonstrate the model performance anymore.
What can be done to avoid these phenomena is shown in the following. It is divided into training and test phase.

First I present four reasonable approaches at training time:

4.2.2.1 Introduce class weights

One has the possibility to define class weights which causes that the loss becomes a weighted average where the weight of each sample is depending on the weight of the corresponding class. So the effect is that the prediction of samples with higher class weights are more penalized. In that way one can put more emphasis on the minor class.

4.2.2.2 Balanced subsets

This option is as simple as its name suggests - just pick subsets that are balanced. That means that one decides for as much samples in each class as is the minor class originally are. It is applicable when the resulting new training set is still sufficiently big.

4.2.2.3 Oversampling

This technique means, that one just resamples (some of) the minor class data in each epoch such that each class is sampled the same number of times. That at least avoids that the model incorporates imbalance knowledge. However, actually no new information is given to the model. That is different with the next approach.

4.2.2.4 Data augmentation

The idea here is to artificially generate new samples by slightly altering the original data. In image processing for instance the following methods are used: flip, rotate, translate, scale, crop or Gaussian noise/distortion. This kind of new data generation is very promising as it not only tackles imbalances but also supplies the model with new information.

Now I am going to show what one can do at test time:

4.2.2.5 Alter perfomance metric

As I said before, accuracy is no longer an appropriate metric. So other ones has to be taken into account. Good simple metrics are precision and recall, more sophisticated ones F1 and AUC score. Later on they will be discussed in detail.

Although data augmentation is always a good idea I decide for simplicity only for class weights in big training runs. Since I have so much data in lots of scenarios balanced subsets were sufficient to show what I want to show.

4.3 Preprocessing

That section describes essential preprocessing steps that were done for all the original MRIs. These procedures have an huge impact on the network training which follows afterwards. I decided to do that work seperately from the model training, i.e. I buffered the processed images in order to use them directly in the model training phase. That outsources lots of time from the latter. Otherwise, by preprocessing on the fly, one could easily test different combinations of preprocessing steps in order to achieve best resuls. But since this works does not aim to beat any state-of-the-art performance in osteoarthritis classification that was not necessary in my opinion.
It is to note that when it comes to testing of the model, obviously each test data has to be preprocessed exactly the same way.
First we want to show how the medical image data format looks like in particular and then different preprocessing techniques are presented.

4.3.1 DICOM standard

[38] described how to manage DICOM images. The following refers to it. Our 3D image data is subject to the Digital Imaging and Communications in Medicine (DICOM) standard. It is the international standard for medical images and related information and is supported by almost every radiology, cardiology imaging and radiotherapy device (X-ray, CT, MRI, ultrasound etc.).
DICOM files differ from other image formats in that it groups information into data sets. They consist of a header and the image data set, all packed into a single file. The header stores demographic information about the patient, acquisition parameters for the imaging study, image dimensions, matrix size, color space and lots of additional nonintensity information required by the computer to correctly display the image. The

image data set is encoded as a series of 0s and 1s that can be reconstructed to an image according to the information from the header. Fortunately there are useful packages in python that can easily extract the relevant image data from the .dcm files. In our special modality the 3D image object is organized in 160 2D slices in separate DICOM files, each of them originally in shape 384×384.
So technically we read the DICOM data by firstly loading all .dmc slices into a list and then stitching their 2D pixel intensity arrays together into a 3D array which we can further process with (see [39] for a python API).

4.3.2 Feature Scaling

See [40] and [41] for the following.
Originally the voxel intensities can potentially range from 0 to 255. It could be that then some elements of the computed gradient (these that correspond to the input neurons) become very large and the corresponding weights are therefore changed too much in their update. That in turn means that the algorithm might miss a minimum in the loss function in the worst case. To avoid such a behaviour there are different scaling techniques that improve the convergence speed and accuracy.

First per image respectively sample-wise normalisation techniques are presented and then alternatives are shown. At the end of this subsection my choice of scaling procedure is explained.

4.3.2.1 Per image normalisation

This term refers to the fact that certain parts in the computation only relies on the current image that is considered whereas in other methods these values are computed based on other references.

Min-max normalisation
This is the simplest method and scales the range of pixel intensities to the unit interval $[0, 1]$:
Let x be an image tensor with shape $(w, h, d, 1)$ before scaling and x' afterwards, respectively. Then this kind of normalisation is computed as follows:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

To obtain a range between an arbitrary set of values $[a, b]$ the equation becomes:

$$x' = \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

Mean normalisation
This is computed as:

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

and shifts the pixel range to $\left[\frac{\min(x)-\text{average}(x)}{\max(x)-\min(x)}, \frac{\max(x)-\text{average}(x)}{\max(x)-\min(x)}\right]$. Note that negative values occur after that normalisation technique. After applying this type the new mean is zero. Therefore this method is also called 'zero-centering'.

Standardisation
This is a further strategy which aims to obtain zero-centered data with standard deviation:

$$x' = \frac{x - \text{average}(x)}{\text{std-deviation}(x)}$$

This rescales the pixel range to $\left[\frac{\min(x)-\text{average}(x)}{\text{std-deviation}(x)}, \frac{\max(x)-\text{average}(x)}{\text{std-deviation}(x)}\right]$.

Percentile normalisation
This strategy incorporates certain percentiles, say α- percentile p_α and β- percentile p_β:

$$x' = \frac{x - p_\alpha}{p_\beta - p_\alpha}, \;\; x'[> 1] = 1, \;\; x'[< 0] = 0$$

The clipping steps force a pixel range of $[0, 1]$.

All of these reported methods tackle the problem with big input numbers with rescaling the pixel range into small number areas. Mean normalisation and standardisation have the additional property that the processed data have same distributional properties (same mean respectively same mean plus deviation). When the original pixel values are on different scales, these methods have the potential to accelerate the learning and improve the performance of the model. By percentile normalisation one has the possibility to exclude outliers that are supposed to be noise in the original image.

Per image normalisation, however, is not suitable for every scenario. Imagine a classification where a model should distinguish between daylight and nightlight and which is trained on daylight and nightlight samples with the same scenery. Normalisation of pixel intensities would vanish the difference in the shading of day and night and therefore per image scaling is counterproductive here.

4.3.2.2 Alternatives

Instead of computing pixel means and deviations according to the current image one can also relate them to batches of images or even the whole data set. The latter is especially a good idea for transfer learning approaches. Phenomena as described by the daylight/nightlight example can be avoided, each image in only adjusted according to the whole data set.

Furthermore I want to mention, that in RGB images one can furhter subdivide global and local scaling, in the first one means and deviations are computed across the color channels and in the second one these are computed channel-wise.

4.3.2.3 Application in OAI data

So how was the decision in the special application in knee MRIs? Since the pixel intensity distribution in medical images is quite similar I forewent sample-wise normalisation methods. I used a percentile normalisation with 0%- and 95%-percentiles computed corresponding to 200 random MRIs. I decided to include all total black pixels (therefore 0%-percentile) but I wanted to exclude pixels that were too bright (therefore 95%-percentile). After that I zero-centered again based on 200 random MRIs. Formally these formulas summarise my feature scaling approach:

$$x' = \frac{x - p_{0.0}}{p_{0.95} - p_{0.0}}, \;\; x'[> 1] = 1, \;\; x'[< 0] = 0, \;\; x'' = x' - \text{px-mean},$$

where $p_{0.0} = 0$, $p_{0.95} = 161$ and px-mean $= 0.31928963243961334$.

4.3.3 Flipping

Since the structures of left and right knees are symmetrical about the sagittal plane, I simply flipped left knees along this plane.

4.3.4 Damaged data removal

I found 10 MRIs that had less than 160 slices. Since this number is very small in contrast to the whole data set I simply excluded them from my experiments and did not try to fix/upsample them.

4.3.5 Others

In that subsection a small overview about further preprocessing techniques for MRI data is given (see [42]). Their application could further improve the performance of developed CNNs.

- denoising: statistical noise, respiratory and body movements, aliasing
- bias field correction: low-frequency non-uniformity
- registration: align the slices correctly
- histrogram equalisation: enhance contrast
- ...

To conclude the preprocessing section I want to repeat that this topic provides tons of opportunities to improve the overall deep learning pipelines. However, feature scaling and flipping of left knees were sufficient to obtain desired results.

5 Method

5.1 Experimental Setup

5.1.1 Technical and Programming Details

To begin with that chapter I want to share an overview about the software and hardware that facilitates the theoretical approaches to realise in practice.

5.1.1.1 Hardware

The following considerations about hardware are oriented to [43] and [44]. Since deep learning requires high computational power it is a good advice to choose the hardware setup as best as possible. The central element for the computations is the graphics processing unit (GPU).

Graphics Processing Unit
There might be two questions that arise in that context:

1. Why do we use GPUs for training neural networks?
2. Are GPUs really better than CPUs (central processing units)?

To address the first question one should take a glance at the interior of the training process of neural networks. The training phase is the most resource-intensive task in the whole experiment that is performed by a coded script. So the focus is adjusted to training. As we saw before it is divided into forward and backward pass. A batch of data is fit into the model to do predictions in forward mode. Then a loss function that evaluates the performance of these predictions is derived with respect to the model weights in backward mode by an algorithm called backpropagation. Finally the weights are updated in the inverse direction of the computed gradient (gradient descent). As we learned in the foundations the forward pass is essentially done by matrix multiplications, this also applies for the backpropagation algorithm. So the operation of interest is matrix multiplication.
It would not be a big deal if the networks would have thousands of parameters. In fact, they can have up to billions of them. It would take years to train such systems by traditional approaches.

To overcome that situation parallelisation mechanisms make it possible to calculate lots of operations simultaneously. This can be achieved by GPUs.
A Graphics Processing Unit is a specialised processor with dedicated memory that conventionally performs floating point operations required for rendering graphics. GPUs devote proportionally more transistors to arithmetic logic units (ALUs) and fewer to

caches and flow control as compared to CPUs. Although a GPU is smaller than a CPU it tends to have more logical cores (ALUs, control units and memory cache) than the latter.

There are a few parameters that determine wether a CPU or GPU is more suitable to train a deep learning model: The **memory bandwidth** is one of the main reasons why GPUs are faster. Huge and complex jobs take up a lot of clock cycles in the CPU in few cores whereas a GPU has more cores and is endowed with dedicated Video RAM memory. The larger the **dataset size** is the more advantages has a GPU versus a CPU. In contrast, a CPU is the right choice when it comes to **optimization** tasks since each CPU core can perform different instructions whereas GPU cores execute the same instruction at a given time in parallel.

These theoretical considerations should answer question one and show that GPUs are the standard choice for models with high numbers of parameters.
To gain trust in the previous arguments benchmarking of state-of-the-art deep learning software tools will help. That was for instance done by a research group at Hong Kong Baptist University in 2017 ([45]). They aimed to make a comparative study of GPU-accelerated deep learning software tools including Caffe, CNTK, MXNet, TensorFlow and Torch. They benchmarked the running performance of these tools with three popular types of neural networks on two CPU platforms, three GPU platforms and some distributed versions on multiple GPUs. The following of their findings confirm the benefits of GPUs:

- In general, the performance does not scale very well on many-core CPUs. Often the use of 16 cores is only slightly better than the use of 4 or 8.
- All multi-GPU versions have a considerably higher throughput and the convergent speed is also better than in a single GPU platform.
- **GPU setups has a much better efficiency than many-core CPUs. All tools achieve significantly more speed by using modern GPUs.**

The main work for this thesis I have done on computers at Zuse Institute Berlin which I often worked on remotely. For preliminary experiments I used Nvidia GeForce GTX 1080 Ti as GPU with a VRAM of 11,178 MiB respectively a high-throughput computing cluster (HTC) including two nodes with two Nvidia Tesla P100-SXM2 with 16 GB each. For the final experiments I switched to a HTC cluster including two nodes with four Nvidia A100 PCIE with 40 GB each.

Central Processing Unit
Although we discussed the higher importance of a good GPU for our purposes that does not mean that a CPU is not important. CPUs are mainly used for data loading. More threads on a CPU means that it is possible to load more data in parallel to feed into the models for training. This is especially useful when it comes to big batch sizes to train on. The other constraint for a valueable CPU is its ability to work with multi-GPU setup, that can be very useful as we saw before. If you plan to work with reinforcement learning techniques a good CPU has to be taken into account, but we

do not need this type of learning in this thesis.

Random Access Memory
The property that is of most interest here is to have at minimum as much RAM as GPU memory since in most cases the data is first loaded into RAM before it fits into the VRAM of the GPU. Another aspect for the RAM is the demand of preprocessing power. Preprocessing steps as we see later are usually done on CPU and RAM. However, it is possible to handle this in a separate step and store the preprocessed data permanently which is then the input for the actual training phase. Therefore it is not crucial in our scenario to have a huge RAM (respectively CPU).

Storage
It is obvious to have a hard disk that is capable of storing the whole used data at minimum. Furthermore you need to have space for storing model checkpoints, plots of certain metrics, the python-code itself and maybe more. To speed up the data loading it is a good advice to use solid-state drives (SSDs).
Since my work environment is a big research institute there are several servers that provide sufficient storage capacities.
It would be possible to broaden the list of hardware discussions but GPU, CPU, RAM and storage are the principal things that has to satisfy certain constraints. Beyond that all components obviously has to be compatible.

In conclusion, I want to note that the technology producers and especially GPU producers adapted and partially specialised to the application field of deep learning. Meanwhile there exists Tensor Processing Units (TPUs) that are highly accelerated in tensor operations which are essential in neural networks.

5.1.1.2 Software and Programming

After we considered the central hardware components that realise complex deep learning applications we will have a look at the software, programming languages and libraries involved. This section focuses on the decisions that I made for the purposes in the thesis. This solution is anything but unique since there exists so many good alternative tools to achieve similar results.

Operating System
To start with the operating system I do not want to contribute to the debate on which system is most suitable to work with. The whole technical work of this thesis is done on Linux computers at Zuse Intitute. To be more precise, I worked on Ubuntu and Debian systems.

Remote Connection
Since I frequently work from home or on remote servers in general, remote connections (from another computer at the institue or from home) are necessary. This is realized within the linux terminal by Secure Shell (SSH) which is a cryptographic network protocol for operating network services securely over an unsecured network (see [46]). From my Windows computer at home I established the SSH connection via the

software PuTTY and there I also used the software WinSCP as a SFTP-client.

Distributed Version Control System
In order to share my development process with my contact person and for backuping important scripts I use the web application GitLab which is based on Git.

Data Visualization
With the aim of getting more insight into the used data it is crucial to have the possibility to manually visualise and analyse them. This functionality provides the software Amira that is actively developed by Thermo Fisher Scientific in collaboration with the Zuse Institute ([47]). It is a multifaceted platform for visualizing, manipulating and understanding life science research data from many image modalities including MRI.

GPU usage
In order to work with the GPU capacities properly the programming environment makes use of CUDA (Compute Unified Device Architecture), an application programming interface (API) created by Nvidia. This relieves the development process since the programmer can write the GPU code in a CUDA-supported language only and the API then interacts with the GPU. When it comes to neural networks the CUDA Deep Neural Network library (cuDNN) completes the GPU setup. It is a library of primitives for neural networks and provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization and activation layers.

Programming language
Python is the language of choice for this thesis. Though there might be other good alternatives it fits best to neural network implementation from my perspective. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. There are several machine learning frameworks that support easy and quick implementation of complex models as (convolutional) neural networks.

Development Environment
As mentioned before I worked at two locations - from a computer at the institute or from my home computer, both establishing a remote connection to another computer at the institute that has a python interpreter preinstalled and runs all the code. On the Linux PC at the institute I write the scripts in a simple GUI text editor, in an editor of the PyCharm IDE or sometimes simply in the console based GNU nano editor. At my Windows PC at home I used Anaconda as a Python distribution. More specially I use its included Jupyter notebooks to write the source code of the scripts.

Deep Learning frameworks
Apart from the Python Standard Library I work with frameworks that are highly specialised in deploying deep learning models. Most of the time I work with TensorFlow that was designed for dataflow programming. As the name suggests tensors are the crucial objects that are processed in the networks. These tensors in practice relate to the theoretical tensors that we discussed before. The use of TensorFlow by big com-

panies as Google, intel or PayPal emphasise its popularity ([48]). The interface Keras that is fully adopted and integrated by TensorFlow facilitates the construction of big models in just a few lines of code. In addition to that PyTorch (which bases on Torch) is another deep learning library that I work with in some cases.

Hyperparameter Optimization
When it comes to the search of promising hyperparameters as batch size or learning rate one is confronted with the following problem:
They depend on each other or more specially, when one of these parameters is altered, previously good other parameters do not need to be good anymore. The combination of all of them is relevant to the performance of the model. Since the test of each combination (grid search) consumes too much time I use the hyperparameter optimization framework optuna, that provide intelligent solutions in parameter search that rely on parameter distributions.
That topic is later discussed in further detail.

Other important libraries
Pandas allows to easily read and manipulate tabular data which are managed in so called dataframes. To work with the medical image format DICOM the libraries pydicom and dicom-numpy are beneficial for my purposes. A last one I want to mention here is NumPy which brings the computational power of languages like C or Fortran to Python. It supports lots of fast tensor operations and numeric computations.

Cluster Usage
In order to schedule jobs for the clusters I used slurm. It is an open source, fault-tolerant and highly scalable cluster management system for large and small Linux clusters.

This closes the considerations about hardware and software which is necessary to build a robust setup to work with.

5.1.2 Metrics

The upcoming subsection provides an overview of loss functions that are best suitable for the internal optimization process and metrics to reasonably evaluate the performance of the model. These observations are adjusted to the category of classification tasks done by neural networks as we exclusively deal with in this thesis. For regression tasks or other models other losses and metrics should be taken into account.
The following explanations essentially follow [49], [50] and [51].

5.1.2.1 Loss functions

As we saw in the technical foundation the gradient descend algorithm internally aims to minimize a loss that comes from a specific loss function. The loss can be interpreted as the error that the model does in its current batch of predictions. There essentially are two points to keep in mind when it comes to the decision for a specific function. Firstly, that function has to match the framing of the specific predictive modeling problem,

or in other words, the loss has to represent the prediction error in a reasonable way. Furthermore, the configuration of the output layer must also be appropriate, otherwise there might occur computational problems.
While focusing on classification problems we subdivide further into binary and multi class classification:

Binary Cross Entropy Loss

That is used for the type of problem where only two classes exist, each data is assigned to one of two labels. This problem is often framed as predicting a value of 0 or 1 for the first and second class. The implementation in neural networks is often designed by a single neuron in the output layer that attains a value between 0 and 1 that represents the probability for the input data belonging to class value 1 (the second class). By defining a threshold value the model then classifies the input as class 0 if the output value is less than or equal to the threshold or it classifies the input as class 1 if the output is greater than this threshold.
In order to satisfy the condition that the output value lies between 0 and 1, often a sigmoid function is used as activation for the last layer.

Function
Cross entropy is the default loss function to use for binary classification where the target values are 0 or 1. There is a good reason for that: by minimizing the cross entropy one gets the same result as for maximizing the (log-) likelihood estimation. Now, let's see the formula. Sticking to previous notation, let $\{(x_i, y_i) : i \in B\}$ be a batch of data and let f be the network representing function at a specific state. Then the binary cross (also called log loss) H is computed as follows:

$$H = -\frac{1}{|B|} \sum_{i \in B} y_i \cdot \log(f(x_i)) + (1 - y_i) \cdot \log(1 - f(x_i))$$

$y_i \in \{0, 1\}$ are the ground truth labels and $f(x_i) \in [0, 1]$ are the predictions in this case, i.e. the predicted probabilities to belong to class 1. Interpreting this formula, for each class 1 data the log of its predicted probability to be in class 1 is added and for each class 0 data the log of its predicted probability to be in class 0 is added. These added values are then divided by the batch-size, so all considered log-values are averaged. Since the loss should represent the (positive) error of a current prediction, the minus in front of the average is added as logarithms of values between 0 and 1 are negative.

So, if a class 0 data has a bad prediction, i.e. $f(x_i)$ is close to 1, then a logarithm of a value close to zero is added, which is highly negative. Analogously, class 1 data with bad predictions add highly negative values to the sum. The minus then flips the sign, so bad predictions result in a high positive loss. On the other hand, good predictions add values close to 0 as logarithms of values close to 1 are computed. So by these considerations it is clear that binary cross entropy loss evaluates the quality of a current batch of predictions, as lower the loss is, the better the predictions are.

Categorical Cross Entropy Loss

When it comes to classification problems with more than two classes, the categorical cross entropy loss is crucial. It is a generalisation of the binary case. For simplicity the different classes y_i are numbered from 0 to the number of classes minus 1. The output layer in a multiclass setting should consist of as much as neurons as classes in the problem. Each of them represents the probability of the input data to belong to the neuron related class. Then the class with the highest predicted probability is the predicted class of the model for that data. Since the probabilities of all classes should add up to 1 and each of them has to be between 0 and 1, usually a softmax activation function is used in the output layer. The categorical cross entropy loss H for one batch of data is computed as

$$H = -\frac{1}{|B|}\sum_{i \in B} \log(s_i),$$

where s_i denotes the predicted probability for x_i to be in the correct class y_i.

5.1.2.2 Evaluation metrics

The loss functions that we saw are less intuitive at first sight, however, they fit best to their purpose. They are crucial for a good optimization process in the model. But when it comes to evaluation of the current state of the model there are other metrics that are supposed to give an understanding about the quality of the model, its strengths and weaknesses. These metrics are evaluated with respect to a set of data with corresponding labels, e.g. the training, validation or test set.
In order to understand the upcoming metrics some important terms are relevant. Here we rely on the binary case.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 5.1: Essential Terms for Metrics

As it can be seen in the figure above, we divide into distinct classes. Since we focus on a binary classification problem here one class is referred to be the positive class and the other is referred to be the negative one. If a positive sample is predicted by the model to be in the positive class, it is named a 'true positive' sample. If a positive sample is predicted negative, it is named 'false negative'. Analogously there exist 'false positive' and 'true negative' samples. If we count the number of samples in each class and place it at the corresponding place in the table, we obtain the so called 'confusion matrix', which has a high statistical importance. In the following TP is the number of

true positives, FN the number of false negatives, FP the number of false positives and TN the number of true negatives.

Accuracy, Precision, Recall

There are some metrics that can be directly derived from the previous terms.

Accuracy
It is the most prominent classification metric, since it is highly intuitive and easy to understand. It is just the proportion of correct predictions among all predictions examined. In terms of previous considerations it is computed by

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}.$$

This metric is a valid choice for well balanced data sets. That means that each class has approximately the same size. Otherwise imagine a scenario where 95% of all data are in class 1 and only 5% are in class 0. A very bad model could just predict class 1 in every case. Then it would have a 95% accuracy although its quality is rather poor. So for this reason accuracy is not enough to assess the overall quality of a considered model.

Precision
Precision gives the proportion of truly predicted positives among all positive predicted, i.e.

$$\text{Precision} = \frac{TP}{TP + FP}.$$

This metric is a good choice when a prediction of a certain class should be true in most cases. One tries to be very sure about a prediction.

Recall
This metric, which is also often called sensitivity, has a distinct aim. It analyses the proportion of correctly classified positives among all cases in positive class, i.e.

$$\text{Recall} = \frac{TP}{TP + FN}.$$

The recall is a reasonable measure to capture as many positives as possible. In medicine it is often more important to capture the disease instead of being sure that a prediction that someone is diseased is sure. Otherwise, an obvious drawback of recall is that it is 1 if we predict positive for all examples.

So we can see that each of the previous metrics has its eligibility on the one hand but also its weaknesses in certain cases on the other hand. For this reason the next metrics utilise trade-off between precision and recall.

F1 Score

This score is the harmonic mean of precision and recall, i.e.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

and therefore again a number between 0 and 1. It maintains a balance between precision and recall. If one of them is low, the F1 score is low as well. Even if one of them is low (e.g. 0.01) and the other is really high (e.g. 1.0), then the score as a result is again low (≈ 0.2) since it is much more sensitive to low inputs. The problem with this score is that it weights precision and recall equally. In practice it is crucial to include domain specific knowledge to determine the importance of precision in contrast to recall. If one of them is more important than the other we can introduce a weighted F1 score as in the formula below:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

By that we give β times as much importance to recall as to precision.
The previous metrics offer good analyse possibilities for the quality of the predictions made by the model. In order to get to these predictions the model compares its probability outputs against a fixed threshold value and based on this the corresponding class is chosen. But how do we get a reasonable threshold? That is a question that is tackled by the following curve:

Receiver Operating Characteristic (ROC) Curve and Area Under Curve (AUC)

The ROC-curve plots two different measures against each other regarding different threshold values. To be more precise the following rates, depending on a classification threshold, are of interest:

$$\text{TPR (True Positive Rate)} = \text{Recall} = \text{Sensitivity} = \frac{TP}{TP + FN}$$
$$\text{FPR (False Positive Rate)} = \text{Fall-out} = (1 - \text{Specificity}) = \frac{FP}{TN + FP}$$

By adjusting the threshold there are different values for these rates:

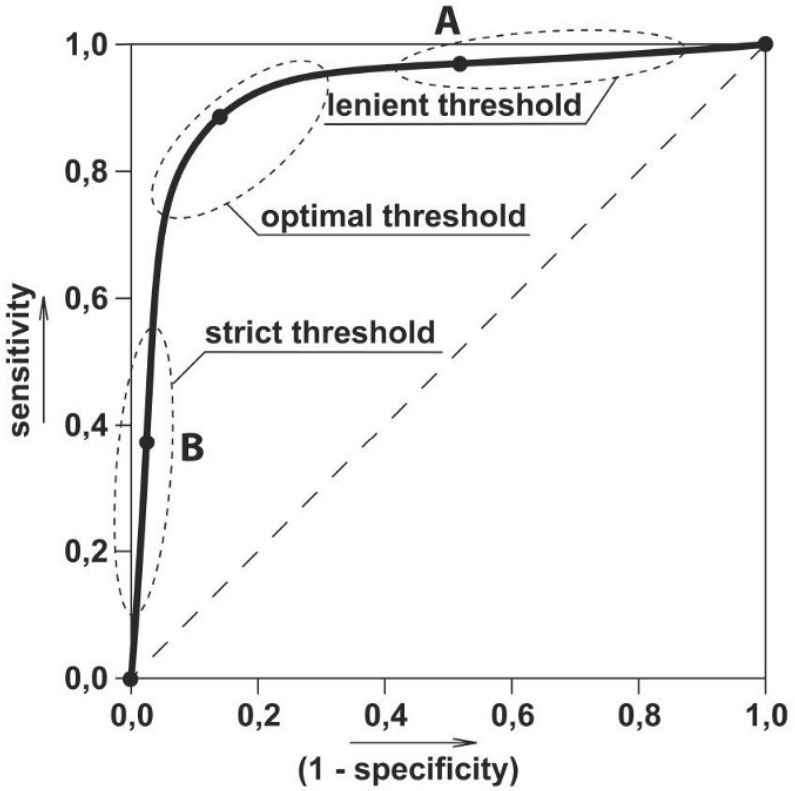


Figure 5.2: ROC-Curve

Obviously, the goal is to find a threshold that maximises the true positive rate and simultaneously minimises the false positive rate. A high TPR means that lots of cases in the positive class are captured and a low FPR means that there are few misclassified cases in the negative class and therefore lots of cases in the negative class are captured. As the figure illustrates values close to the upper left corner represent threshold choices that leverage best ratios between TPR and FPR.

The Area Under Curve (AUC) metric is just the area under the ROC-curve. It describes the correlation between TPR and FPR regarding all threshold choices in one single number between 0 and 1. The main advantage of this metric is that it is classification-threshold invariant, i.e. the score is independent of the threshold unlike other scores as F1 or accuracy. That property makes it indispensable in any classification task.

5.1.3 Hyperparameter Optimisation

This upcoming subsection is dedicated to an increasingly important subject - the search of promising hyperparameters. These parameters are set before the model begins to train and optimize its model parameters (weights and biases of the neural network). Hyperparameters govern the whole process, whereas model parameters are updated batchwise by the chosen optimization algorithm. When you are confronted with a big data problem that you want to solve with deep neural networks there is one essential problem - you are constrained by the technical complexity of your machine that in turn involves potential time and memory issues. For that reason it is inevitable to make

reasonable decisions when it comes to the design of the overall setup. Each model training might need several hours until days so the correct configuration of the model plays an important role. Especially for image classification it turned out that several recent advances in state-of-the-art benchmarks have come from better configurations of existing techniques rather than from novel approaches to feature learning. (Ziiiit-taaaaaat)

In the following certain hyperparameters will be discussed and different approaches for their optimization will be presented. This subsection then closes with a programming library that fits our purposes.

5.1.3.1 Deep Learning Hyperparameters

We saw the separation between the model parameters and hyperparameters whereby the latter are of interest here. These hyperparameters can further be subdivided into model specific hyperparameters and optimizer related hyperparameters. These descriptions essentially follow [52].

Model hyperparameters

These are the parameters that directly concern the architecture of the considered model. Focusing on neural networks there is a huge amount of decisions to make and each of them has an influence on the performance at the end. Here some general design decisions are depicted, later the specific CNN architectures are introduced in detail.
Before we start to examine different design decisions it is crucial to understand the concepts of **underfitting** respectively **overfitting**. As explained in the foundation a network is first trained on the training data and then the performance is measured on the validation set (or later on the test set).

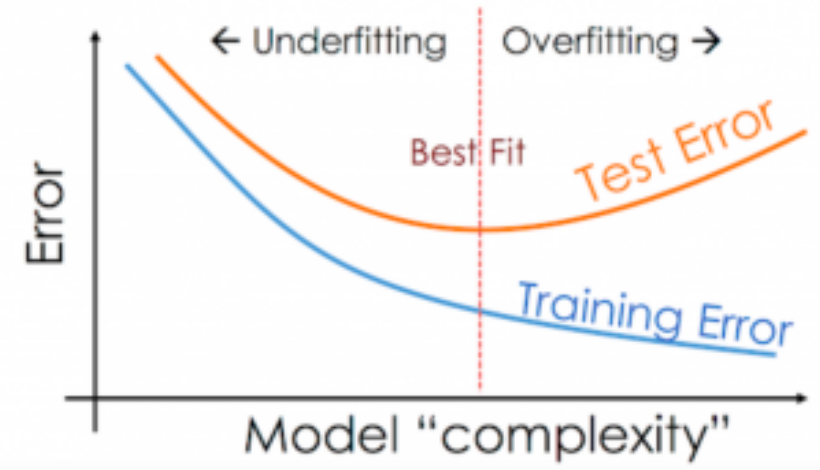


Figure 5.3: Overfitting and Underfitting (see [53])

As it can be seen in the figure the errors on the training and test set vary by different model complexities. It is to note that test error and validation error could be used interchangeably in order to demonstrate these phenomena. So if the neural network is

not complex enough both of the errors are quite high and therefore the model quality is poor. In that case we talk about an underfitted model since it has not learned the data as desired. By expanding the model complexity both errors get decreased until a certain point. Then the test error increases while the training error still decreases so the gap between these errors gets bigger. We talk about an overfitted model in that scenario. Here the model is too complex so it fits too much to the training data and looses its generalisation capabilities. In other words, the model learns the training data by hard and is not able to recognise general patterns is the test set. We obviously want to avoid underfitting as well as overfitting and try to hit the best fit, the sweet spot, as close as possible.

Number of hidden units
This number is the main measure of models learning capacity. As we saw in the Universal Approximation Theorem neural networks are able to learn every continuous function assuming an arbitrary width (or arbitrary depth in dual versions). In practice that does not hold but the capacity of the model increases by the number of hidden units. As more complex the function to learn is as more neurons are necessary. Image classification is a highly non-linear task that currently often requires millions of trainable parameters. Otherwise the model would underfit the underlying function.

Number of layers
By going deeper a convolutional neural network abstracts the input further and further. Lets demonstrate the different stages of image processing by an example, that is shown in [54]. The following cat image is used to pass through the so called 'ResNet-50', a modern CNN, that is here trained on ImageNet, a large scale image database:

Figure 5.4: Cat Image

In the following activation maps from layer 0, 20 and 48 are shown:

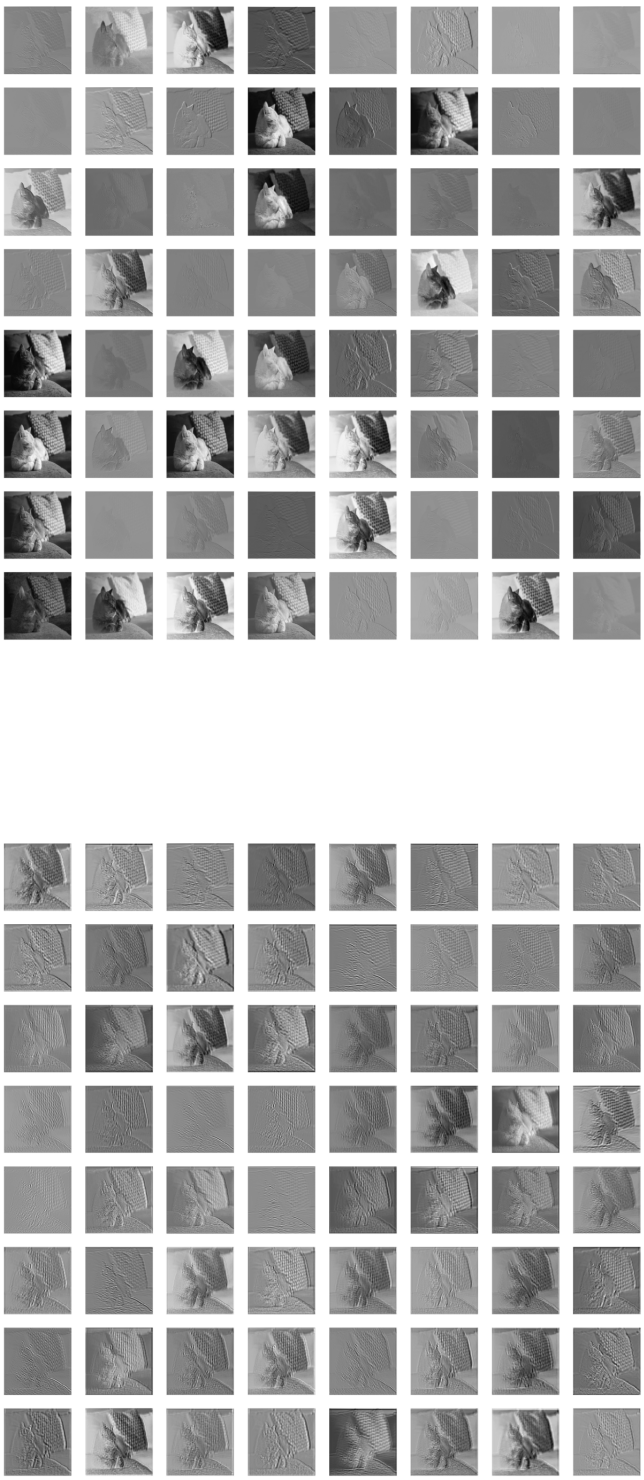


Figure 5.5: Layer 0, 20 and 48 of ResNet-50

Figure 5.5: Layer 0, 20 and 48 of ResNet-50

We can observe that as the image processes through the layers the details increasingly disappear, the patterns become more abstract.

Activation functions

The activation function brings non-linearity into the model. In modern research it is shown that the choice of activation function plays an important role for the performance of the network. Yingying Wang et al. for instance showed in 2020 that different activation functions lead to different accuracies in a facial expression recognition task (Zitat). They proposed a new piecewise activation function which was able to beat most of state-of-the-art methods on the public JAFFE data set. This example illustrates the importance of the choice of the activation function. In the past there were two popular activation functions that were used by default in neural networks - namely the logistic function (commonly named sigmoid function) and the hyperbolic tangent function:

- Logistic Function: $S : \mathbb{R} \to \mathbb{R}, S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$
- Hyperbolic Tangent: $\tanh : \mathbb{R} \to \mathbb{R}, \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x}-1}{e^{2x}+1}$

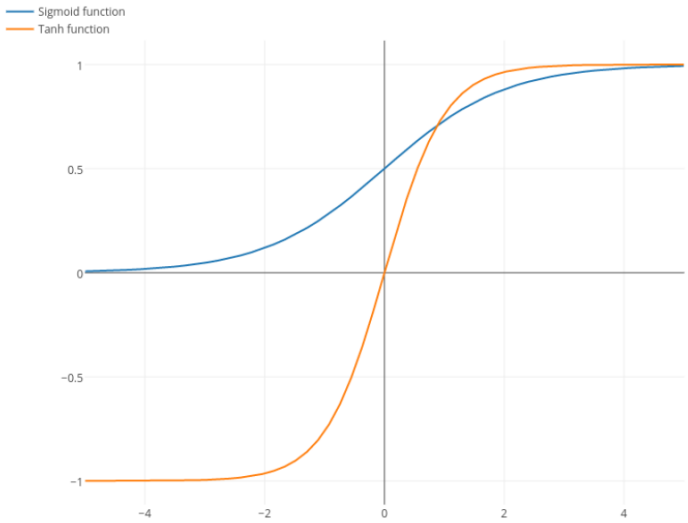


Figure 5.6: sigmoid vs tanh function

The hyperbolic tangent often converges faster than the standard logistic function for the same reason that inputs should be normalized, namely, because it is more likely to produce outputs that are on average close to zero. (Zitat) A general problem with both is that they saturate, i.e. large values snap to 1 and small ones snap to -1 for tanh or 0 for sigmoid respectively. Furthermore they are only really sensitive to changes around zero. Another prominent issue is the so called 'vanishing gradient problem' - in deep neural networks the gradient diminishes dramatically as it is propagated through the network so that it may have too small effect on the weight update. In order to address all these problems a new activation function arised - the rectified linear unit (ReLU):

- Rectified Linear Unit: $relu : \mathbb{R} \to \mathbb{R}, relu(x) = \max(x, 0)$

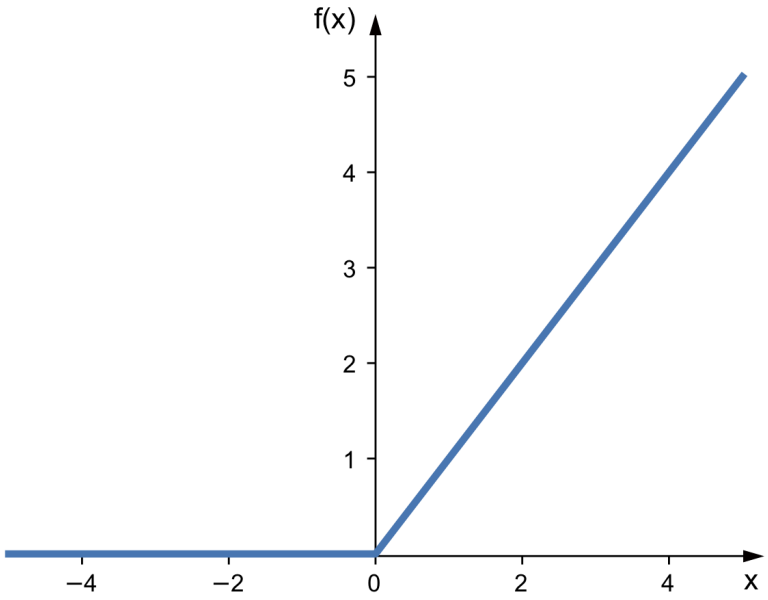


Figure 5.7: ReLU function [55]

This function prevailed and is still default in CNNs nowadays.

> "[another] major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units."
> - Page 226, Deep Learning, 2016 ([56], see [57])

Later the specific convolutional network architectures and its innovations will be discussed in detail.

Optimizer Related Hyperparameters

The following are hyperparameters that are highly related to the optimisation respectively training process.

Learning Rate
Considering the weight updating rule for a batch B

$$w_{\text{new}} := w_{\text{old}} - \alpha \frac{\partial \mathcal{L}_B}{\partial w_{\text{old}}}$$

the learning rate α influences how big the weights should change in the inverse direction of the gradient of the loss-function with respect to the old weight. In other words the learning rate determines how fast the optimisation should steer to a potential optimum. A too small learning rate would mean that it would take a much longer time (more epochs) to reach the ideal state. If the rate is too large the algorithm might overshoot the ideal state and cannot converge. Modern approaches use adaptive learning rates starting with a high value and then successively decreasing the rate when the training progresses (also called 'learning rate decay'). The following figure summarizes the different phenomena:

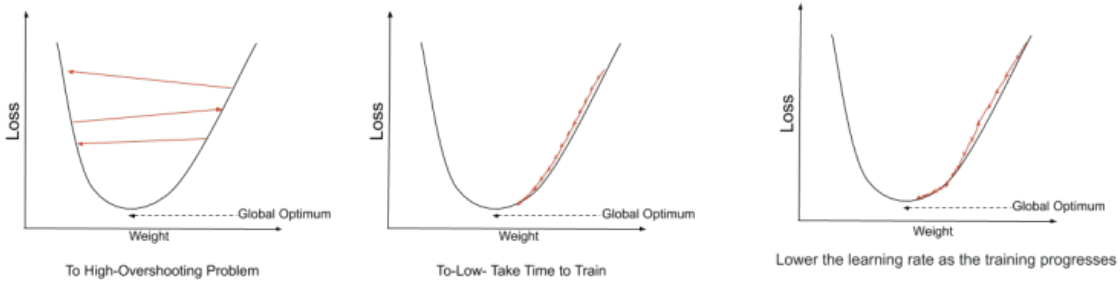


Figure 5.8: Impact of Learning Rate [58]

Batch size
The batch size is the number of training samples that are fed into the network simultaneously. This size has a non-trivial effect on the resource requirements for training and the convergence speed of the optimizer. 'Stochastic training' means a batch-size of 1 and 'batch training' means that the batch is the whole dataset. For all sizes in between we speak about 'mini batches'. A larger batch size allows a computational boost as

the forward and backward pass process more data at once. On the other hand more GPU memory is required. Smaller sizes induce more noise in the error calculations so they might prevent the algorithm from stopping at local minima. In our specific case with 3D image data the tensor objects get too big for big batches so we are technically limited to small batch sizes in most cases.

Number of epochs
The longer the optimisation proceeds the more the model fits the training data. Since the goal is to minimise the validation error it is important to find the spot where the model generalises the training data well. Similar to the relation between model complexity and over-/underfitting, the number of epochs relates to over-/underfitting as well.

Weight decay
As we saw before the number of neurons relates to the complexity of the model and the latter in turn relates to over- and underfitting phenomena. Another parameter that has potential to avoid overfitting is the so called weight decay respectively weight regularisation. The idea is to penalise huge weight values by introducing a new term in the loss function.
The conventional batch-wise loss function $\mathcal{L}_B(\ell, f)$ is now extended to $\mathcal{L}_B(\ell, f) + \frac{\lambda}{2} \sum_{w \in \mathcal{W}} w^2$. Therefore the weight updating rule is altered to

$$w_{\text{new}} := w_{\text{old}} - \alpha \frac{\partial \mathcal{L}_B}{\partial w_{\text{old}}} - \lambda w_{\text{old}}.$$

In this formula the name 'weight decay' for parameter λ gets clear. By choosing higher values for the decay the model more and more aims to avoid high weight values and in this way it prevents overfitting.
It is to note that there also exist other regularisation techniques as for instance L_1 regularisation where the sum of absolute weight values is added to the loss.

That closes the insight into the most important hyperparameters.

Hyperparameter importance

Although we have seen lots of hyperparamters there is still a question left - which of them have the most impact on the convergence and performance of our models? Could we, at best, quantify how big the impact actually is?
For this pupose I oriented at a publication "Hyperparamter Importance for Image Classification by Residual Neural Networks" from 2019 [59]. This work aims to recoginize patterns across different 2D image datasets. For each of the ten considered they analysed the hyper parameter importance by using a global hyperparmeter importance framework, namely functional ANOVA (analysis of variance). That not only reveals the meaning of individual paramaters, but also investigates interaction effects betweeen arbitrary subsets of hyperparameters. It is based on the computation of so called marginals that show how a certain value of hyperparameter performs, averaged over all possible combination of the other hyperparamters' values. They found out the

following to be most influential, in decreasing order:

1. initial learning rate
2. weight decay
3. batch size
4. momentum (feature for gradient descent)
5. number of epochs
6. initial learning rate + momentum
7. batch size + weight decay
8. batch size + number of epochs
9. patience (number of epochs without improvement after learning rate is reduced)
10. learning rate decay
11. resize crop
12. shuffle (training data after epochs)
13. horizontal flip
14. vertical flip
15. tolerance (for early stopping criterion)

The figure below shows their variance contribution:

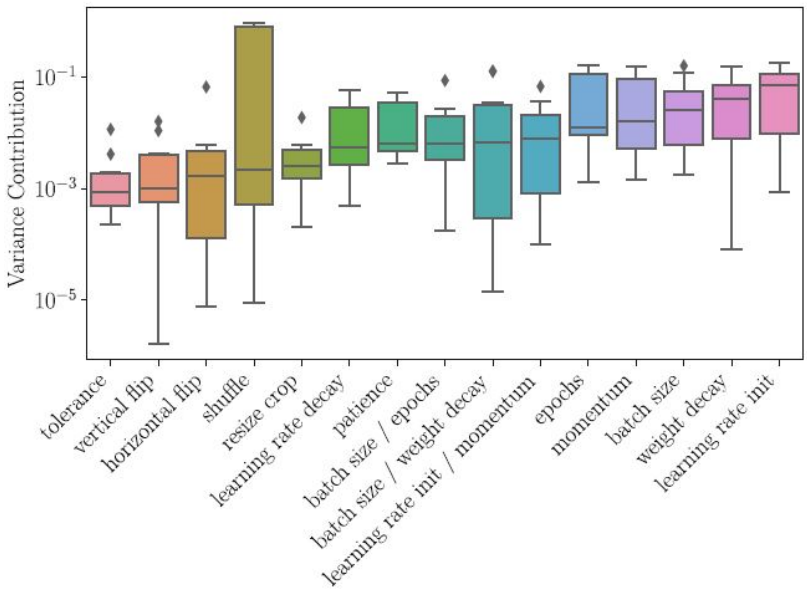


Figure 5.9: Functional ANOVA [59]

Some of parameters are controlled automated by the use of advanced optimiser algorithms as Adam Optimisatin.

5.1.3.2 Hyperparameter optimisation

When it comes to the optimisation of hyperparameters there exist different approaches.

Manual search

This method refers to manually and intuitively picking hyperparameters. That method includes certain rules of dumb and own experience.

Random search

Here, as the name suggests, simply random combinations are sampled.

Grid search

This method just brute forces every single combination from a certain combination grid of hyperparameters.

These are the conventional approaches. While the first two ones aims to reduce the invested time by either experience driven decisions or luck, the latter one yields best combinations but needs tons of hyperparameter objective evaluations as the number of combinations increases exponentially in the number of parametes.

Bayesian optimisation

These more modern approaches make use of a surrogate model that describes the real objective function. Each successive trial then incorporates knowledge from previous objective funtion evaluations into this surrogate. In this way only most promising hyperparameters are sampled. It turned out, that these approaches indeed find best combinations in less time.

5.1.3.3 Optuna framework

The concepts of bayesian approaches are intuitively shown in [60] which is summarised in the following.
Optuna is one python library among many that provides basian hyperparameter optimisation approaches. I used it for hyperparameter optimisation in my experiments. By default it uses so called tree-structured parsen estimisation, a certain variant of bayesian optimisation. One also talks about sequential model-based global optimisation.

The basic functioning of such an approach is divided into the following steps:

1. build a surrogate probability model of the objective function
2. find the hyperparameters that perform best on the surrogate

3. apply these hyperparameters to the true objective function
4. update the surrogate model incorporating the new results
5. repeat steps 2–4 until max iterations or time is reached

The following criteria are relevant when it comes to the design of a sequential model based optimisation technique. According to the their choice different methods arise:

- a domain of hyperparameters over which to search
- an objective function which takes in hyperparameters and outputs a score that we want to minimize (or maximize)
- the surrogate model of the objective function
- a criteria, called a selection function, for evaluating which hyperparameters to choose next from the surrogate model
- a history consisting of (score, hyperparameter) pairs used by the algorithm to update the surrogate model

Tree-structured parsen estimation is a special modelling of the surrogate. For the selection function one mostly uses the expected improvement criteria.
In the figure below the iterative approximation of the true objective function by a surrogate model is depicted:

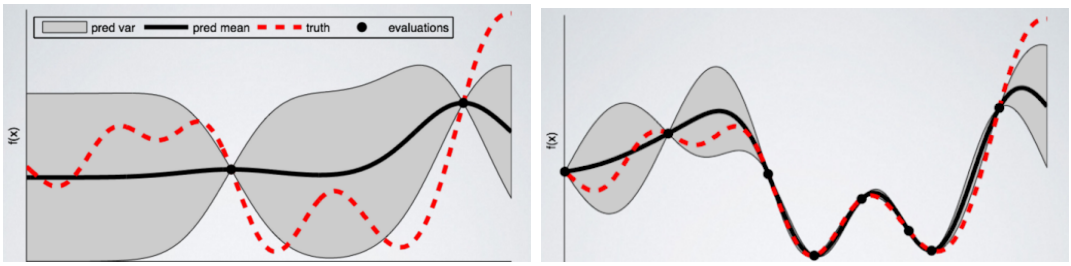


Figure 5.10: Bayesian approach using a surrogate model to approximate the true objective, after few iterations (left) and after more iterations (right)

In [61] the mathematical background of these approaches can be found.

5.2 Initial Approach: 3D CNN with inception blocks and residual connections

Now all preparations has been done in order to present the initial approach for knee osteoarthritis classification. First the network architecture is portrayed in detail and then two experimental settings are described.

5.2.1 Architectural Details

Now I am going to introduce a neural network that achieved remarkable results in 2D image classification tasks - namely the Inception Resnet in its second version.

My goal was to find a popular network with high performance but also simple extension possibilities to three dimensions. For that purpose I oriented at the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** that took place annually from 2010 to 2017. This challenge was based on a subset from ImageNet dataset with roughly 1,000 images in each of 1,000 categories. In total, there are 1.2 million training, 50.000 validation and 150,000 test images on which competitions in image classification, single-object localisation and object detection were hold. Below one can see the stunning development that since the introduction of AlexNet in 2012 was exclusively led by convolutional neural networks.

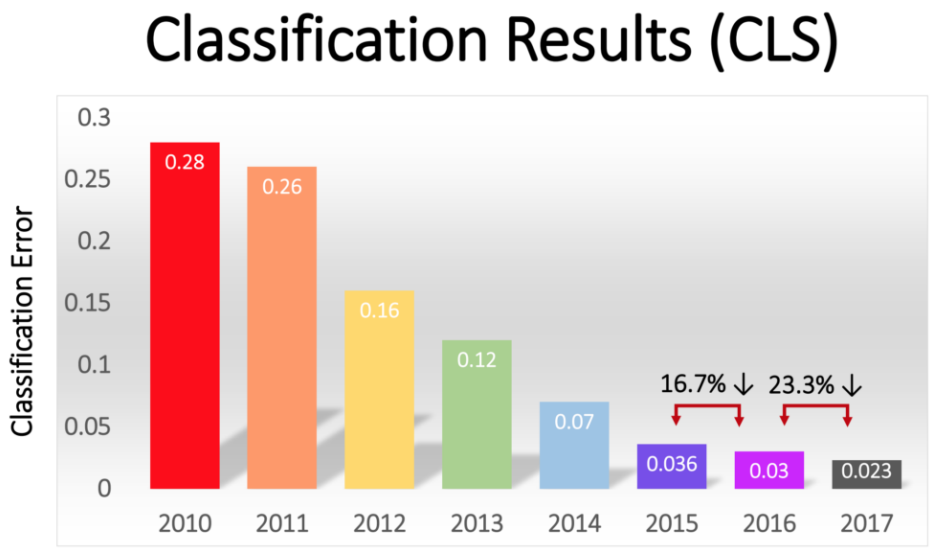


Figure 5.11: Classification results ILSVRC [62]

2014 the so called GoogLeNet won the challenge. This network developed by Google was the first that made use of inception blocks. In the following years 2015 and 2016 residual connections were inferred in the challenge leading networks.

Both of these concepts were then combined in a single network resulting in Inception-ResNet, where two different versions were published in August 2016. The second version is the basis for the upcoming approach.

5.2.1.1 Baseline model in 2D

In order to understand the innovative functionality of inception blocks and residual connections it helps to look at its chronological development steps. The following sticks to [63], also the pictures were taken from there. As claimed before the milestone was already set in 2014 by the first inception network.

Before that time most popular CNNs were going deeper and deeper by stacking several convolutional layers. This paradigm has its limitation since deep models are prone to overfitting and the vanishing gradient problem. Furthermore it gets very computationally expensive. But apart from these more technical considerations there is a general problem with complex image classification tasks - salient parts extremely vary in size but predefined convolution filters are more or less suitable for only one certain size. The example below shows images belonging to the same class but with completely different scales that are difficult to capture by conventional approaches with only one specific convolution hierarchy.

Figure 5.12: dogs occupying different portions of the whole image

A larger convolution kernel would be better for globally distributed information and a smaller one would rather be suitable for locally distributed information.

Inception v1

So the new idea was to simply apply different kernels on the same level making the network wider instead of deeper. In that way the first inception module was invented:

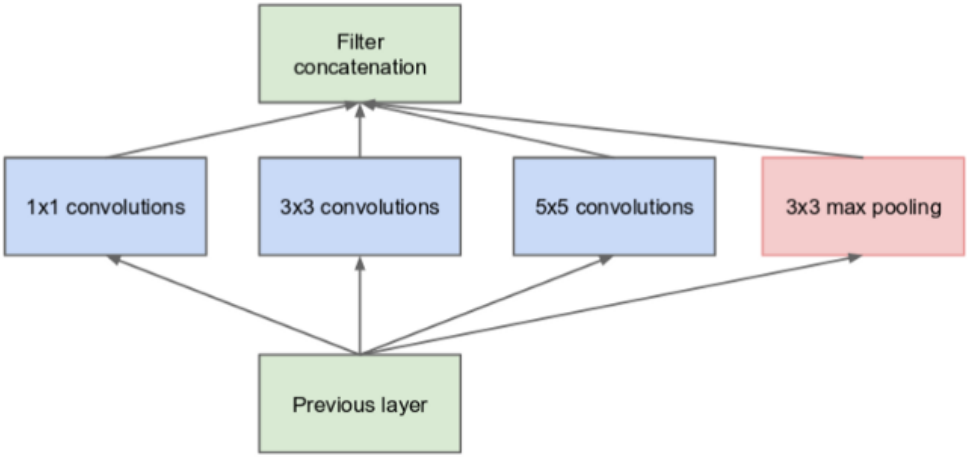


Figure 5.13: inception module, naive version

Three different convolution sizes are incorporated together with a max pooling operation. The outputs are then concatenated. In order to make this module cheaper the

authors introduced some 1×1 convolutions that reduce the number of input channels. This advanced inception module was the basis for the Inception Net version 1 (GoogLeNet) published in September 2014. Its architecture involves 9 such inception modules after some preliminary convolutions in the 'stem' of this network. Since this network is quite deep two auxiliary classifiers should prevent the middle parts from 'dying out'. The corresponding two auxiliary losses are used for training and then they are ignored during inference.

Inception v2

Advances of the GoogLeNet were then published in December 2015 realised by Inception v2 and Inception v3. These first update, Inception v2, aimed to reduce representational bottleneck and computational complexity.

The main innovation here was factorisation of convolutions. By realising distinct principles they build three new types of inception blocks:

1. Replacing one 5×5 convolution with two 3×3 convolutions drastically improves the computational complexity by a factor of 2.78 (Zitat).

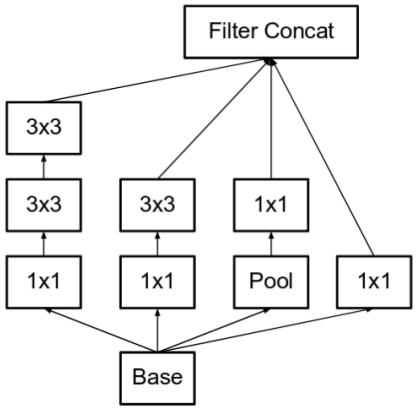


Figure 5.14: Inception Block A

2. But that was not enough, they also split up quadratic convolution filters into asymmetric convolutions, e.g. a 3×3 convolution becomes a chain of one 1×3 and one 3×1 convolution which found out to be 33% cheaper.

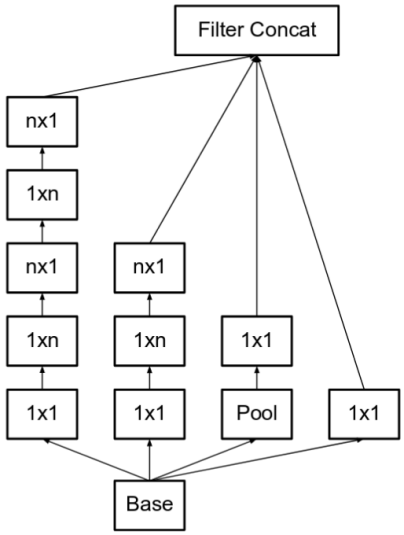


Figure 5.15: Inception Block B

3. A further new principle should tackle the representational bottleneck - they expanded the filter banks and in this way the inception modules has been made wider instead of deeper.

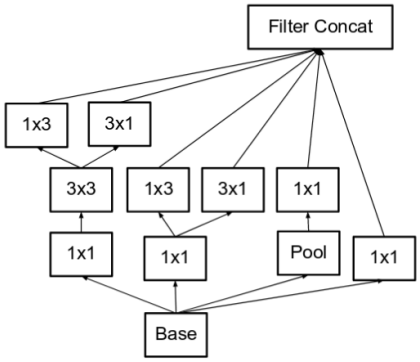


Figure 5.16: Inception Block C

The new architecture Inception v2 used all of these inception modules.

Inception v3

It turned out that the auxiliary classifiers did not contribute much until near the end of training. They can be seen as sort of **regularisation**. Next to that some other improvements has been done. The following summarises what was new in contrast to version 2:

- RMSProp Optimiser
- Factorisation of 7×7 convulutions
- BatchNorm in the auxiliary classifiers

- label smoothing (another type of regularisation)

The next steps of improvement were done in a publication from August 2016. The paper introduced the 4th version of the Inception net as well as both versions of Inception-ResNet.

Inception v4
The aim of this version was to do the modules more uniform and less complicated. On the one hand they modified the "stem" while the three main inception modules look very similar to their previous versions (v2/v3). On the other hand this version explicitly introduced so called **Reduction Blocks** in order to change the width and height of the grid which look as follows:

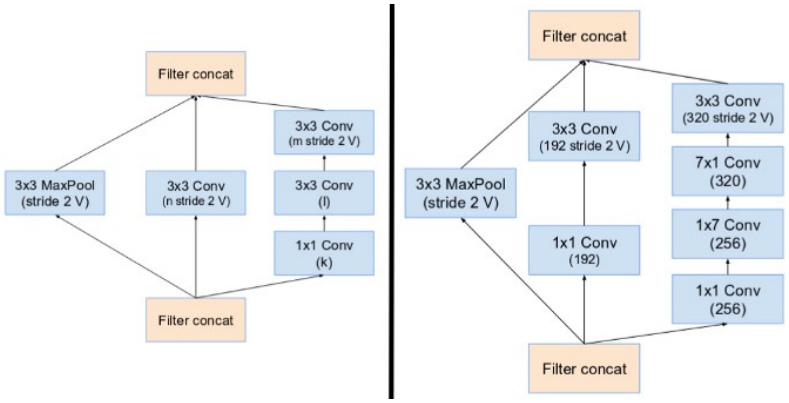


Figure 5.17: Reduction Block A (35x35 to 17x17 size reduction) and Reduction Block B (17x17 to 8x8 size reduction). Refer to the paper for the exact hyper-parameter setting (V,l,k).

Finally the Inception-ResNet in its two versions can be presented.

Inception ResNet v1 and v2
These networks are inspired by **residual connections**. The ResNet respecitvely ResNeXt who initially incorporated them won the ImageNet challenge in 2015 respectively 2016. The two new versions provide modified inception modules which utilise residual connections that add the output of the convolution operation to the input. For residual addition to work one needs same dimensions for the input and output after the convolution. That is realised by 1×1 convolutions to match the depth sizes. The new inception modules looks as follows:

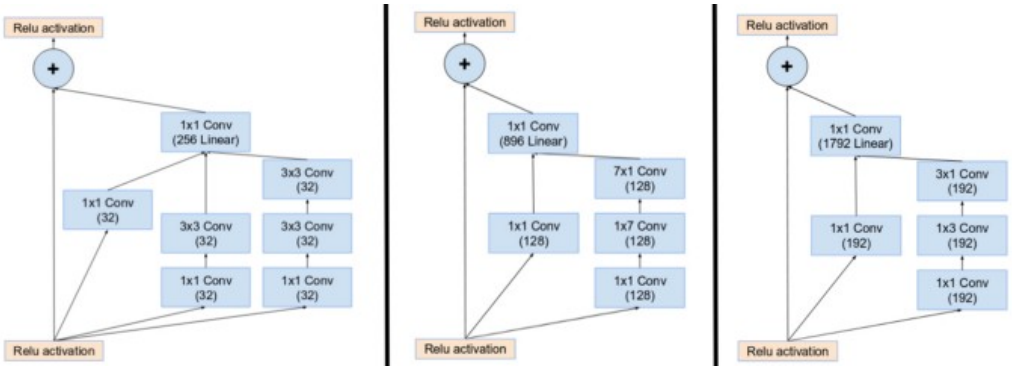


Figure 5.18: Inception Modules in Inception-ResNet v1

The pooling layer was replaced by the residual connection. However pooling still takes place in the reduction blocks where reduction block A remains the same as in Inception v4 but reduction block B is slightly altered.
Furthermore it turned out that networks with residual units deeper in the architecture let the network "die" if the number of filters became too big. Therefore a scaling of the residual activations by a value around 0.1 to 0.3 was applied. There is also to note that originally BatchNorm was not used after summation in order to fit the entire model on a single GPU.

I want to conclude that paragraph with some differences between version 1 and 2 of Inception ResNet:

- v1 has a computational cost similar to Inception v3
- v2 has a computational cost similar to Inception v4
- they use different stems
- both of them have same inception and reduction blocks

Overall architecture
After the iterative improvement steps in the development of Inception ResNet version 2 were shown, now, the overall network in two dimensional case should be presented:

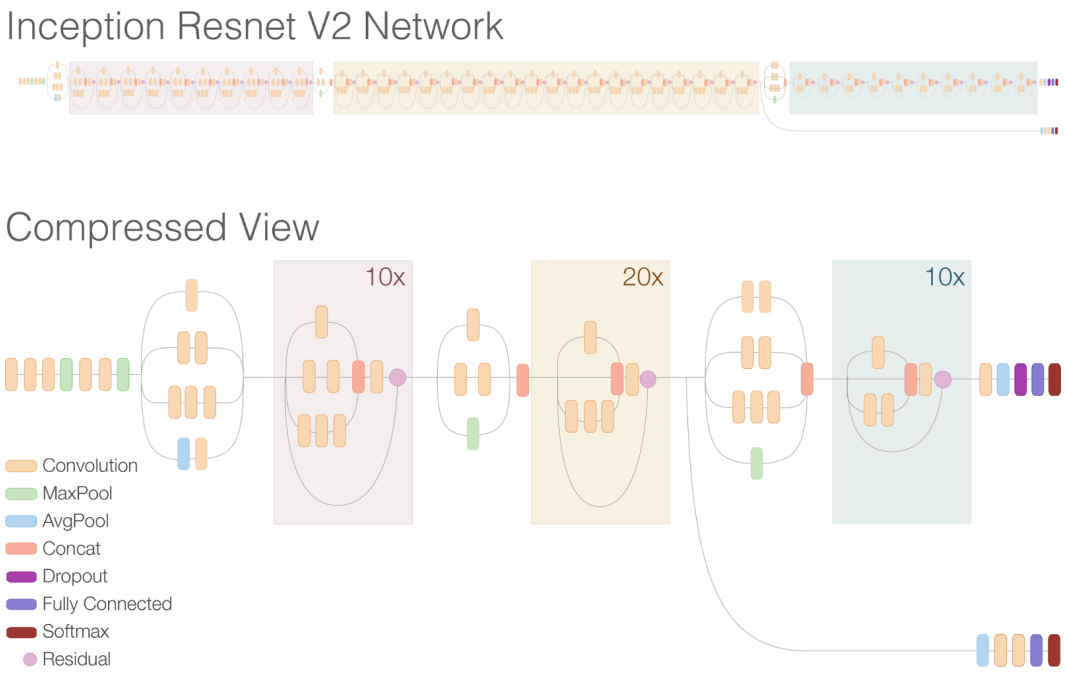


Figure 5.19: Inception ResNet V2 [64]

As Google's AI blog from 2016 reported [64], this network is more accurate than previous state-of-the-art models including Inception V2, ResNet 152 and ResNet v2 200 on the ILSVRC image classification benchmark. It obtained impressive 80.4% top-1 accuracy respecitvely 95.3% top-5 accuracy.

To close this subsection I want to underpin the progress between Inception v3 and Inception ResNet v2 by an example: While both of them are very good in identifying distinct dog breeds the Inception ResNet v2 does it noticeably better:

Figure 5.20: Alaskan Malamute (left) and a Siberian Husky (right) [64]

Whereas the old network mistakenly predicts Alaskan Malamute for the right picture the newer model correctly identifies both dog breeds.

5.2.1.2 Extension to 3D

Since the performance of this network is very convincing in two dimensional case I was wondering if the same ideas are extendable for three dimensions. It turned out that

the code adaption was straight forward. I oriented at the implementation that was provided by the keras framework and altered the following:

- each quadratic 2D convolution with filter size (k, k) became a quadratic 3D convolution with filter size (k, k, k)
- each quadratic 2D pooling operation with filter size (k, k) became a quadratic 3D pooling operation with filter size (k, k, k)
- within the inception-resnet blocks:
  - each chain of 2D 1×3 and 3×1 convolutions became a chain of 3D $1 \times 1 \times 3$, $1 \times 3 \times 1$ and $3 \times 1 \times 1$ convolutions
  - each chain of 2D 1×7 and 7×1 convolutions became a chain of 3D $1 \times 1 \times 7$, $1 \times 7 \times 1$ and $7 \times 1 \times 1$ convolutions

Furthermore it is to note that the keras implementation used batch normalisation operations after convolutions.

We are now ready to set the first experiments.

5.2.2 Binary classification: KLG 0 vs 4

To figure out if the architecture has promising potential to achieve good performance in more sophisticated settings, it is a good idea to start with an easy case initially. In that way the following experiments belonging to binary classifications between healthy and severly diseased knees can be seen as proof of concept.

It quickly turned out that this the task is quite easy for this architecture such that time consuming hyperparameter optimisations are not necessary. In this case I rather put the focus on the question how the size of the training set influences the model training. It would be interesting to know how big the training set has to be at least to learn the difference between completely healthy and completely diseased cases.

To get answers I set up the experiments below by successively increasing the size of the training set while keeping hyperparameters and validation set constant. The subsequent training sets are supersets of the previoius ones. In order to prevent imbalancing problems I always picked balanced training sets. The constant validation set was balanced as well with 250 samples in each class. The batch size was chosen to be 10 and the learning rate was left to 0.001 by default. For the backpropagation alorithm I took the Adam optimiser as I did in almost every experiment in this work.

- **Experiment 1**: training set with 25 samples in each class
- **Experiment 2**: training set with 50 samples in each class
- **Experiment 3**: training set with 100 samples in each class
- **Experiment 4**: training set with 250 samples in each class
- **Experiment 5**: training set with 500 samples in each class

5.2.2.1 Evaluation on test set

At the end the performance of the model's best epoch from experiment 5 was evaluated on test set that contains 2851 healthy and 219 severely diseased knees.

5.2.3 Binary Classification: KLG 0 vs 2

This category of experiments are of valuable interest since it represents a kind of early recognition of the osteoarthritis. Computer aided diagnosis systems would obviously be very beneficial if they captured cases of minimal arthritis.

At this point hyperparameter optimisation by the optuna framework was taken into account. Three different optimisation objectives were explored:

5.2.3.1 Optuna: Convergence Objective

With the aim of gaining some insights about the convergence behavior of the model I designed the following objective function for the hyperparameter optimisation: return the number of epochs that the network needs to exceed a certain threshold in training accuracy. The latter was set to 0.7 and when the models could not achieve this value within 50 epochs, however 50 was returned. In this way optuna is looking for hyperparameter combinations for which the model learns as fast as possible - at least according to the training data.
The initial idea was to set up three different optuna runs: one that is only optimising batch size, one for only learning rate and one for both of them. Unfortunately a mistake in programming lead to the fact that the optimiser always was compiled to the default learning rate (0.01), independent of what learning rate was sampled by optuna. Thus, the meaning respectively the intention behind the experiments (except only batch size optimisation) altered. This yields the following experiments:

- **Experiment 6**: initial idea of only batch size optimisation remains
- **Experiment 7**: initial idea of only learning rate optimisation shifted to always sampling equal combinations of batch size and learning
- **Experiment 8**: initial idea of both together optimisation shifted to only batch size optimization while confusing the model

All of these are trained on the same 200 balanced training samples and 500 balanced validation samples.
From these experiments the hyperparameter combination from the trial with the lowest objective value was supposed to run in a bigger setting then. In case of more trials with same optimal objective this one with lowest achieved validation loss was supposed to be picked.

- **Experiment 9**: study winner trained on 2000 balanced training samples and 1600 balanced validation samples

5.2.3.2 Optuna: Multi Val-Loss Objective

The second and third optuna objective are dedicated to capture parameters that yield the best overall performance. The objective in this subsection tries to additionally assert a certain kind of stability in the model while the next one only takes the performance into account.

The here considered multi val-loss objective returns the mean of the three lowest validation losses that the network achieved over 50 epochs of training. Thereby the bayesian optimisation aims to find combinations that rather yield models that achieve good results over at least three epochs instead of only one.

Orienting at the previous analysis about hyperparameter importance, weight decay was also optimised in addition to batch size and learning rate. Thus, one experiment is stated for this optuna search with:

- **Experiment 10**: influence of batch size, learning rate and weight decay on robust performance

Afterwards the study winner that was unique in this case was trained.

- **Experiment 11**: study winner trained on all training data (11,438 samples with KLG 0 vs. 7,247 samples with KLG 2) and 1600 balanced validation data

To tackle the imbalance in training data I used class weights assigning the weight of the healthy class to the number of all training samples divided by the doubled number of healthy samples and the weight of the diseased class, analogously, to the number of all training samples divided by the doubled number of diseased samples.

5.2.3.3 Optuna: Single Val-Loss Objective

As announced this objective directly focuses on the overall performance by returning the lowest validation loss achieved over 50 epochs of training, again optimising batch size, learning rate and weight decay.

- **Experiment 12**: influence of batch size, learning rate and weight decay on performance

Again the unique study winner was trained using the same class weights as above.

- **Experiment 13**: study winner trained on all training data (11,438 samples with KLG 0 vs. 7,247 samples with KLG 2) and 1600 balanced validation data

5.2.3.4 Evaluation on Test Set

At the end of these experiments the performance of the best epoch of the best study winner is evaluated on the test set that contains 2851 healthy knees and 1671 minimal diseased ones.

5.3 Data Reduction Approach: cropping region of interest

In the upcoming section we present an approach that is led by a traditional idea when it comes to computational issues - cropping the irrelevant regions and keep the region of interest. These conventional approaches almost always incorporate domain knowledge about the location of useful information in the input data. However, in modern research automated learning of region of interests (ROIs) has been made possible.

We will present a certain cropping strategy based on our application - knee OA classification.

5.3.1 Region of Interest

Before it comes to the technical details it should be medically clear on which region one could focus that potentially provides most information about the underlying classification problem. Since knee osteoarthritis has huge effects on the articular cartilage and the bones underneath it is a good idea to look at the joint cave and its surroundings. Actually the force transmission within the joint is split over two strands - the lateral and the medial condyle. Its femoral parts transmit the upper body weight over the menisci into the tibial plateau which can be seen in the figure below. The dark blue plates represent the menisci here.

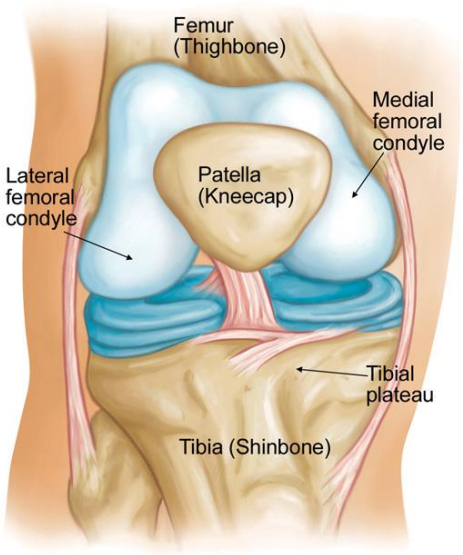


Figure 5.21: Condyles of the knee joint

For this anatomical reason my idea was to consider each condyle separately in order to extract information about knee osteoarthritis. Another fact motivates this distinction as well - the Joint Space Narrowing is also separated into lateral and medial side in the OAI Data. In that way a condyle split leverages condyle-wise considerations about

the knee.

So the goal of my cropping approach is to obtain reasonably cut condyle sides for each MRI.

5.3.2 Additional Data

Since the data set is too big to crop each MRI by hand an automated method should be applied. For that purpose segmentation data and surface triangulation data could help. As mentioned in chapter 4 it was provided by the research group at Zuse Institute. Segmentation is the task of assigning each voxel of the input image to a certain category it belongs to.

5.3.2.1 Segmentation Data

Here I will shortly present where this data come from and how it helps to crop the knee joints. The automated segmentation method was published in the journal 'Medical Image Analysis' in February 2019. It combines the advantages of Statistical Shape Model based regularisation with CNN-based classification of voxel intensities. It follows a small abstract of their work:

- approach incorporates 3D Statistical Shape Models (SSMs) as well as 2D and 3D CNNs
- variants of U-Nets are employed as CNNs
- trained using data from OAI and MICCAI grand challenge 'Segmentation of Knee Images 2010' (SKI10)
- validation on 40 validation and 50 submission datasets from the SKI10 challenge
- **accuracy equivalent to the inter-observer variability of human readers** is achieved in this challenge for the first time
- quality is assessed using various measures for data from OAI (507 manual segmentations of bone and cartilage plus 88 manual segmentations of cartilage)
- method yields sub-voxel accuracy for both OAI datasets
- method was ranked first in SKI10 challenge (at least in April 2018)

It is to note that the segmentation error directly has an impact on the quality of my cropping procedure. However, since the segmentation quality of their work is really good it was worth to utilise these data for my approach. The segmentation masks include six categories respectively six types of tissue as the following sample depicts:

Figure 5.22: Segmented Knee

This example was visualised by Amira which displays the segmented categories in different colors. As you can see the segmentation covers tibia and femur bone, femoral and tibial cartilage and medial and lateral meniscus. The knee cap and soft tissue, however, is not covered.

So by using these segmentation masks it is simple to compute bounding boxes including cartilages and meniscis but excluding big parts of the femur and tibia bone. But how is it possible to split the joint in the exact middle? Since different kinds of bones and other tissue vary in its spreading from person to person the segmentation is not enough to find a good location for the split in the middle.

5.3.2.2 Surface Triangulations

In order to compute a reasonable point for separating lateral and medial side surface triangulations of the femur bone were taken into account. Its computation was also part of the work of the work group at ZIB.
They look as follows:

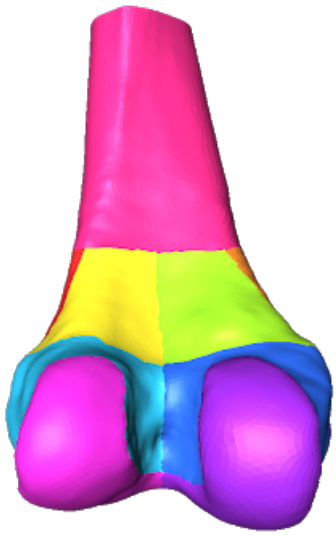


Figure 5.23: Surface triangulation of femur bone with patched colors

The whole surface of the bone consists of uniformly distributed nodes. These nodes are assigned to different numbers and nodes with same numbers have almost the same locations relative to its special knee. This allows to pick a certain reference node for the middle split.

5.3.3 Cropping Details

After all required data is introduced the cropping procedure can be explained in detail. I subdivided the process into four routines - three of them for finding lower and upper bounds in each spatial dimension and the fourth to find the central splitting point.

5.3.3.1 Coronal crop

To crop irrelevant regions in that direction I only used the segmentation masks. In particular I combined each segmented tissue to one entire object, i.e. I separated the combined segmented knee structures from the background. Afterwards a bounding box of this knee object was computed to obtain the upper and lower bounds for the coronal direction. Lets denote these boundings by $B_{c,l}$ and $B_{c,u}$ (B for 'bound', c for 'coronal', l and u for 'lower' and 'upper').

5.3.3.2 Transverse crop

For the transverse direction the crop was slightly more complex: the goal was to keep everything from articular cartilage and menisci. The problem for the lower bound (tibia side) was that in some severely diseased cases these tissues might have already been teared away. This could result in different forms of overlapping so a priori it is not clear which of articular cartilage or meniscus is at the lowest position. For that

reason I decided to join everything in the segmentation mask except for background and tibia bone. For the resulting object then a bounding box was computed to obtain the lower bound.
The upper bound could be acquired more simple just by drawing a bounding box around the femoral cartilage. Since that type of cartilage is very likely to be at the highest position besides the femur bone in all MRIs, the upper bounds were computed reasonably.
Lets denote these boundings by $B_{t,l}$ and $B_{t,u}$.

5.3.3.3 Sagittal crop

In that dimension the center point had to be computed additionally to the upper and lower bound. The latter were obtained analogous to the bounds for the coronal crop (by the same bounding box). The center point evaluation involves surface data as explained below.
Lets denote the outer boundings by $B_{s,l}$ and $B_{s,u}$ and the center by C_s.

5.3.3.4 Center point

To choose a valuable point for the center splitting I manually visualised several femur surfaces via Amira software. I concluded that vertex with number 13557 represents a good reference. It is located at the crossing of four differently patched colors at the bottom of the femur bone (see Figure 5.21). Since the surfaces and the original MRIs are aligned according to different coordinate systems a certain affine transformation has to be applied to map from the reference node to the coordinates for the original data.

5.3.3.5 Final routine

So as we saw the positions for cropping are essentially obtained by certain bounding boxes with the help of segmentations respectively by the center evaluation with the help of surface triangulations. The cropping now can be described formally as follows: Let I be the image input tensor and assume its first dimension to be the coronal, its second one to be the transverse and the third one to be the sagittal. The channel dimension is not considered here. Then for the resulting condyle tensors C_1 and C_2 hold:

$$C_1 = I[B_{c,l} - 10 : B_{c,u} + 11, B_{t,l} - 10 : B_{t,u} + 11, B_{s,l} : C_s],$$
$$C_2 = I[B_{c,l} - 10 : B_{c,u} + 11, B_{t,l} - 10 : B_{t,u} + 11, C_s : B_{s,u}]$$

Note that in coronal and transverse dimension some surrounding safety pixels were included to the resulting parts.

Below you can see an example for the condyle split. Note that I used cropped segmentation masks here for visualisation purpose only, in fact the original MRIs are split up instead of the segmentations.

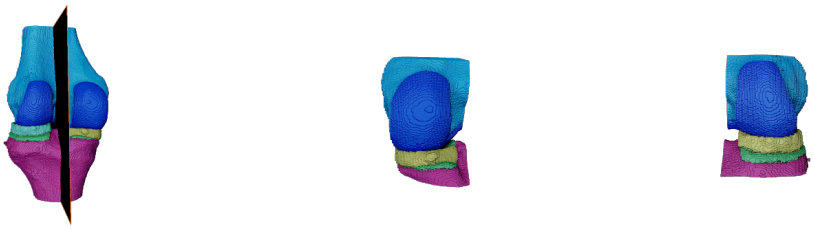


Figure 5.24: Condyle Split

Notice that the condyles are flipped as the input MRI was a left knee.

Preprocessing
At the end of that subsection I want to comment that the cropping routines are incorporated into a new preprocessing procedure. So for my whole work there exist two distinct preprocessing approaches - one that processes whole knees and one that additionally applies the condyle split. I decided to first feature scale the knees (percentile normalisation and zero-centering again), then do the split and finally flip condlyles belonging to left knees.

5.3.4 Binary Classification: JSN 0 vs 3, JSN 0 vs 2

The following experiments were principally designed as proof of concept for the cropping procedure. Since the patients knees weren't under load as the MRIs were taken the Joint Space Narrowing is not directly visible. However, the model predictions might potentially be good by examining indirect features that are related to the narrowing. Explainable intelligence approaches could explain these indirect biomarkers in future.

As input for the upcoming experiments I restricted to one condyle side which was obtained by the splitting approach before. These condyles were then processed by the inception resnet v2.

5.3.4.1 Binary Classification: JSN 0 vs 3

To begin with I classified the medial JSN by setting the learning rate to 0.001 and the batch size to 10.

- **Experiment 14**: training on 400 balanced samples, validation on 118 balanced samples

5.3.4.2 Binary Classification: JSN 0 vs 2

In this case I also predicted the lateral JSN in a separate experiment, setting same learning rate and batch size as before.

- **Experiment 15**: medial JSN: training on 400 balanced samples, validation on 200 balanced samples

- **Experiment 16**: lateral JSN: training on 400 balanced samples, validation 100 healthy and 37 diseased knees

5.3.5 Siamese Network Approach

To summarise the cropping procedure we obtained for each knee data two new data that include the cropped condyles of the knee joint. This could enormously reduce the number of voxels.

For processing of these two condyles I developed a siamese network structure including one strand for each condyle. These strands essentially are the inception resnets v2 as they was described before.

The following figure gives an overview of the new built network architecture:

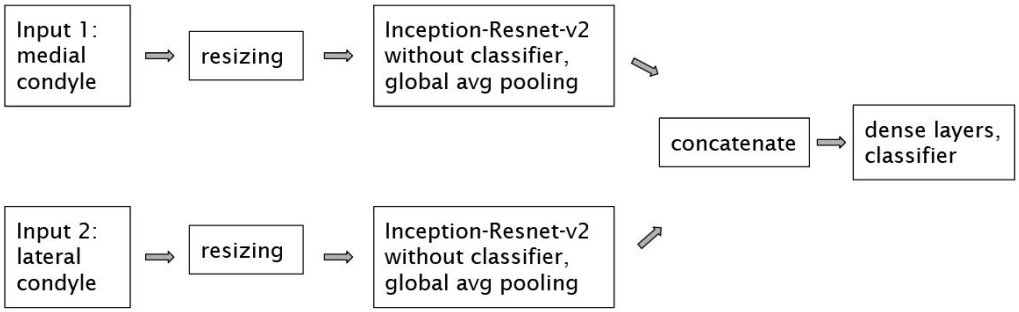


Figure 5.25: Siamese architecture

To be more precise the dense network at the end consists of five layers with 128 neurons in the first and 64 neurons in the other layers. Each of them is combined with ReLU activation. The prediction layer then consists of two output neurons with softmax acitvation.

The intuition behind this idea was that the network independently learns something about the different condyles and afterwards combines the knowledge (in dense layers) to come to a final prediction.

5.3.6 Binary classification: KLG 0 vs 4

I chose KLG 0 vs 4 due to its simplicity to poof concept of the siamese network. The training run was configured with batch size 10 and learning rate 0.001.

- **Experiment 17**: training on 200 balanced samples, validation on 500 balanced samples

Afterwards the performance of best epoch was evaluated on test set with 3396 healthy and 368 severely diseased knees.

5.3.7 Binary Classification: KLG 0 vs 2

Again that was the interesting scenario to focus on. I did analyses subdivided into the convergence and multi val-loss objective in optuna. Every trial was based on the same training respectively validation set with 200 balanced respectively 500 balanced samples.

5.3.7.1 Optuna: Convergence Objective

I did one experiment that unfortunately was again influenced by the programming mistake that wrongly compiled the optimiser to the default learning rate of 0.01. That shifts the meaning as in experiment 8.

- **Experiment 18**: initial idea of both together optimisation shifted to only batch size optimization while confusing the model

The unique study winner was trained then.

- **Experiment 19**: study winner trained on all training data (11,611 samples with KLG 0 vs. 7,307 samples with KLG 2) and 1600 balanced validation samples

5.3.7.2 Optuna: Multi Val-Loss Objective

According to the section about hyperparameter importance I included some more parameters. Besides batch size, learning rate and weight decay I also incorporated a boolean variable that determines wether the training data has to be shuffled after one epoch or not and another boolean variable that determines wether another optimiser has to be used instead of Adam or not, namely the AMSgrad algorithm that is a variant of Adam optimisation.

- **Experiment 20**: influence of batch size, learning rate, weight decay, shuffling and other optimiser on robust performance

Afterwards the unique study winner was trained.

- **Experiment 21**: study winner trained on all training data (11,611 samples with KLG 0 vs. 7,307 samples with KLG 2) and 1600 balanced validation data

5.3.8 Evaluation on Test Set

At the end the best epoch of the best study winner is evaluated on test set with 3396 healthy and 1964 minimal diseased knees.

5.4 Model Reduction Approach: learnable filter offsets

Until now we saw two approaches how they would traditionally been used. We started with a good architecture and almost directly started training. That quickly confronted us with technical limitations. We then as an improvement showed a cropping strategy that tackles these computational issues. But as a huge drawback we are reliant on acquisition of segmentation and surface triangulation data that is really topic related and complicated to generate. Now I am going to present an approach that also tackles computational issues but also is practically applicable for 3D data processing by CNNs in general and that is independent of additional data.

First it is presented where the upcoming idea has its origin, then we shift to the practical application to our scenario.

5.4.1 OBELISK - One Kernel to Solve Nearly Everything: Unified 3D Binary Convolutions for Image Analysis

This heading is the title of the publication where the used innovative principle is introduced. It is written by Matthias P. Heinrich et al. and presentet on the 1st Conference on Medical Imaging with Deep Learning (MIDL) in 2018. They invented a convolutional kernel that learns spatial filter offsets in a continous differentiable space in addition to the weight coefficients. In this way it automatically learns important sample locations and adapts to the problem specific structures.

Application context
They demonstrated their new approach at the task of Computed Tomography (CT) multi-organ segmentation. CT scan is another medical imaging technique used in radiology that is able to generate detailed 3D images of patients for diagnostic purposes [65]. Segmentation, shortly expressed, is the task of assigning each pixel/voxel of the input volume to one of the given labels.

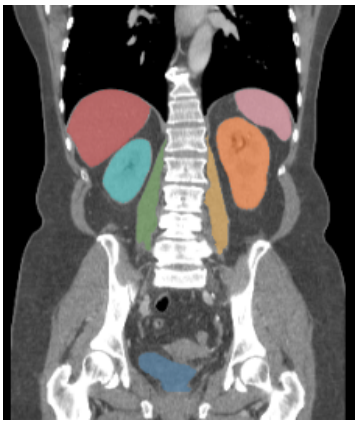


Figure 5.26: segmented organs

The figure above examplarily shows a certain segmentation of organs with seven distinct labels.

5.4.1.1 Motivation

To begin with, the issues that the considered paper aims to address will be listed:

1. Conventionally, lots of specifications of CNNs are subject to much manual design. In particular the shape and size of the receptive field for convolutional operations is a very sensitive part that differs in the underlying problem.

2. 3D multi-scale CNN architectures have huge memory requirements and need large labeled datasets. Their training speed is still quite slow.

3. A common belief has evolved that:
   - "more convolutional results are always better"
   - "small convolution filters are preferable"
   - "multi-scale or progressive dilated receptive field are necessary for dense prediction"

   None of these has really been questioned so far.

Especially related to the latter point they aim to show substantially new ideas. They want to show alternative **CNN architectures** that are **much easier to design, train and deploy**.

5.4.1.2 Method

Traditional convolution operation
In order to demonstrate the innovation it is useful to remind important properties of the conventional convolution operation:
It uses a user-defined layout for sampling locations which is for instance restricted to a 5×5 respectively $5 \times 5 \times 5$ grid for 2D respectively 3D case. Only the filter coefficients are learned here. Often it is necessary to apply a bunch of filters to capture local and regional context information.

To be more formally lets describe a 3×3 convolution in 2D case, the extension to 3D is analogous. We stick to the papers notation here:
Let d be the dilation factor, then the spatial filter offsets can be defined as $s_x = s_y = (-2d, -d, 0, +d, +2d) \in \mathbb{Z}^5$. Assume $W \in \mathbb{R}^{5\times5}$ to be the learnable weight matrix and $I(\cdot, \cdot) \in \mathbb{R}$ to be the input feature at a certain location. Then the output $F(x, y) \in \mathbb{R}$ at location $(x, y) \in \mathbb{Z}^2$ can be computed as:

$$F(x, y) = \sum_i \sum_j W(i, j) \cdot I(x + s_x(i), y + s_y(j))$$

Since summation and multiplication are differentiable with respect to the weights this overall term is differentiable as well.

OBELISK kernel
Their approach, in contrast, proposes a convolution kernel that not only learns the

filter weights but also the spatial filter offsets which comes from a continous differentiable space. By that method that is inspired by transformer networks and deformable convolutions the model itself learns how sparse and large the receptive field has to be for a given task. This single kernel has the capability to replace distinct small filters at different scales.

Mathematically it can be described as follows, again restricting to 5×5.
Let now $S_x \in \mathbb{R}^5$ and $S_y \in \mathbb{R}^5$ be the spatial offsets (continued valued here). To obtain the convolution output for inputs on a discrete grid, bilinear interpolation is applied:

$$F(x,y) = \sum_i \sum_j W(i,j) \cdot (w_1 I(\lfloor x + S_x(i) \rfloor, \lfloor y + S_y(j) \rfloor) + w_2 I(\lceil x + S_x(i) \rceil, \lfloor y + S_y(j) \rfloor) \\ + w_3 I(\lfloor x + S_x(i) \rfloor, \lceil y + S_y(j) \rceil) + w_4 I(\lceil x + S_x(i) \rceil, \lceil y + S_y(j) \rceil)),$$

where $w_1(i,j), \ldots, w_4(i,j)$ are the bilinear coefficients:

$$\begin{aligned}
w_1(i,j) &= (\lfloor x + S_x(i) + 1 \rfloor - (x + S_x(i)))(\lfloor y + S_y(j) + 1 \rfloor - (y + S_y(j))) \\
w_2(i,j) &= (x + S_x(i) - \lfloor x + S_x(i) \rfloor)(\lfloor y + S_y(j) + 1 \rfloor - (y + S_y(j))) \\
w_3(i,j) &= (\lfloor x + S_x(i) + 1 \rfloor - (x + S_x(i)))(y + S_y(j) - \lfloor y + S_y(j) \rfloor) \\
w_4(i,j) &= (x + S_x(i) - \lfloor x + S_x(i) \rfloor)(y + S_y(j) - \lfloor y + S_y(j) \rfloor)
\end{aligned}$$

Again all operations, i.e. summation, multiplication and ceiling/floor functions, are differentiable such that gradient descent still works with that new kernel.
It is to note that the extension to three dimensions is straight forward by shifting to trilinear interpolation.

In fact, the authors introduced two distinct variants - an unary and a binary version. The first one was presented in the previous description. For the binary version the model instead learns two filter offsets whose interpolated values are subtracted and then the outcome is mulitplied with joint filter weight.

Implementational/ Architectural Details
For these considerations I follow their implementation for the original model on github (https://github.com/mattiaspaul/OBELISK/blob/master/models.py). Further extensions can also be found there. The authors choose to incorporate 1024 spatial 3D offsets for the unary variant respectively 2048 for the binary one. The offsets are initially assigned to normally distributed numbers in each dimension. Due to empirical investigations the mean was set to 0 and the standard deviation was set to 0.05. So in total there are $3 \cdot 1024$ filter offsets in unary variant respectively $6 \cdot 1024$ filter offsets in binary version and 1024 weights, additionally.

By implementing the Obelisk in PyTorch the authors utilise a grid sampling procedure involving trilinear interpolation that computes the output using the input volume (in quarter resolution here) and voxel locations that are specified in a certain grid. In this grid the learnable spatial offsets are set.

So the total number of parameters for the Obelisk layer is equal to the number of spatial offsets plus the weights, i.e. $3 \cdot 1024 + 1024 = 4.096$ in unary case and $6 \cdot 1024 + 1024 = 7.168$ in binary version. To get an impression about the scale of these numbers, only a single $3 \times 3 \times 3$ convolution layer with 64 filters would already have 1.728 parameters.

Afterwards a small number of $1 \times 1 \times 1$ convolutions is utilised to to learn complex spatial patterns. At certain steps batch normalisation and the rectified linear unit (ReLU) are used, which can be explored in detail at a following summary. For the sake of readability I do not mention these types in this paragraph anymore. Although it would be rather complicated the principle of stacking $1 \times 1 \times 1$ convolutions could also be achieved with a certain architectural design of dense layers. That leads the authors to name these types of network structure as $1 \times 1 \times 1$ **DenseNet**.
So the first feature reduction is realised by a convolution layer that takes the 1024 activation maps generated by Obelisk and outputs a volume including 256 channels. That would mean that there were 256 different $1 \times 1 \times 1024$ kernels yielding a total of 262.144 parameters. To save parameters the authors decided to group this convolution into 4 parts. Grouping of convolutions here means that this overall operation is parallelised into 4 smaller convolutions with $1.024/4 = 256$ input feature maps and $256/4 = 64$ output channels each. Convolution weights are shared across these groups and the smaller output volumes are concatenated along channel dimension at the end. By this method one reduces the number of parameters by 4. That yields a layer with $262.144/4 = 65.536$ parameters for the grouped convolutions.
Then a layer taking 256 feature maps and convolving them to an output with 128 channels is employed (32.768 parameters).
At this point the architecture develops a slightly more complex $1 \times 1 \times 1$ DenseNet:
By feature concatenation it reuses previous results. This structure starts with a convolution layer reducing the channel dimension from 128 to 32 (4.096 parameters). Its output is concatenated channel-wise with the previous output with 128 channels yielding 160 channels. The next convolution generates again 32 output channels (5.120 parameters) that is sticked to the previous ouput, here with 160 channel. That process is iteratively applied until one obtains 256 channels after concatenation.

Finally a prediction layer completes the architecture that reduces 256 channels to the number of labels. It is to note that only for that layer the authors included biases, for the other ones, biases are disabled. At the end the generated segmentation masks are resized to a certain resolution by trilinear interpolation.

Below you can see a summary of the original architecture in its binary version:

layer type	input channels	output channels	parameters
OBELISK layer	1	1024	7,168
4-groups convolution	1024	256	65,536
batch normalisation	256	256	0
ReLU	256	256	0
convolution	256	128	32,768
batch normalisation 128	128	0	
convolution	128	32	4,096
ReLU	32	32	0
concatenation	128 and 32	160	0
batch normalisation	160	160	0
convolution	160	32	5,120
ReLU	32	32	0
concatenation	160 and 32	192	0
batch normalisation	192	192	0
convolution	192	32	6,144
ReLu	32	32	0
concatenation	192 and 32	224	0
batch normalisation	224	224	0
convolution	224	32	7,168
ReLu	32	32	0
concatenation	224 and 32	256	0
batch normalisation	256	256	0
convolution	256	#labels	#labels ·257

By excluding the prediction layer this network has **128,000 parameters in total** which is impressively low for complex tasks as multi organ segmentation.

5.4.1.3 Results

To begin with, I want to give an insight about the development of the spatial offset distribution. For that purpose the author's figur below clearly shows that the network automatically learns to increase the receptive field.

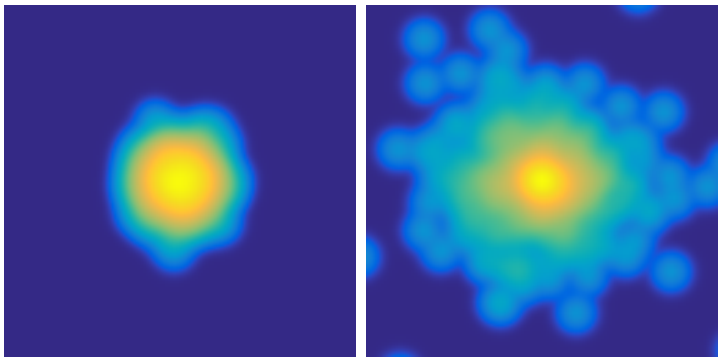


Figure 5.27: Distribution of spatial filter offsets initially (left) and after 50 epochs of training (right), shown in logarithmic colormap

It is to note that the size of extent of the figures is only half of the image domain. It is evident that many sample positions remain close to the centre as the organs that are to segment are mainly located there as well.

The published results are based on a leave-one-out cross validation on 10 contrast-enhanced 3D CT scans of the VISCERAL3 training dataset. They compared their approach with the former state-of-the-art version of the prominent multiscale U-Net which was specifically developed for biomedical images. The performance metric was chosen to be the Dice Score (also called Dics Coefficient). In simple terms it is computed as two times the area (= number of pixels) of overlapping between the ground truth segmentation and the predicted segmentation divided by the total number of pixels in both images [66]. Before we dive into numerical comparisons lets see an example CT scan that was also segmented by both of U-Net and OBELISK network. To be more precise, a two dimensional plane out of the three dimensional volume is depicted:

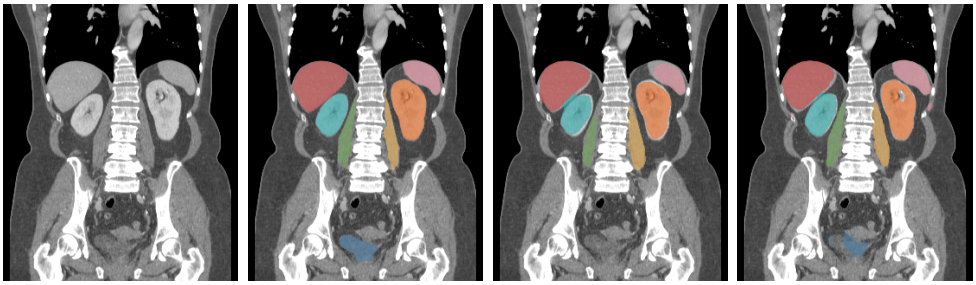


Figure 5.28: Segmentation overlays, from left to right: coronal plane from original CT scan, ground truth segmentation, multiscale U-Net and OBELISK network

Liver (red), spleen (pink), bladder (dark blue), left kidney (light blue), right kidney (orange), left psoas major muscle (green) and right psoas major muscle (yellow) are considered in this segmentation task. As it can exemplarily be seen for the spleen the U-Net recognised the upper left part while the OBELISK seems to almost capture the whole organ. Furthermore, the bladder seems to missed completely by U-Net.

To compare both of the models more general the table below numerically shows their differences:

	U-Net	OBELISK network
Dice Score	71.12%	76.68%
parameters	1,250,000	129,536
memory	$\approx 2,500$ MByte	≈ 700 MByte

More precise this type of OBELISK network additionally incorporates online hard example mining (OHEM), a technique that only selects a quantile of largest training errors for backpropagation of the loss. In that way it increases the difficulty-awareness in case of semantic segmentation. Furthermore, the authors stated that the performance can be pushed furhter to an Dice Score of 80.61% by using simple online augmentation.

According to the training speed the OBELISK approach also outperforms the U-Net by reducing the number of epochs to the half:

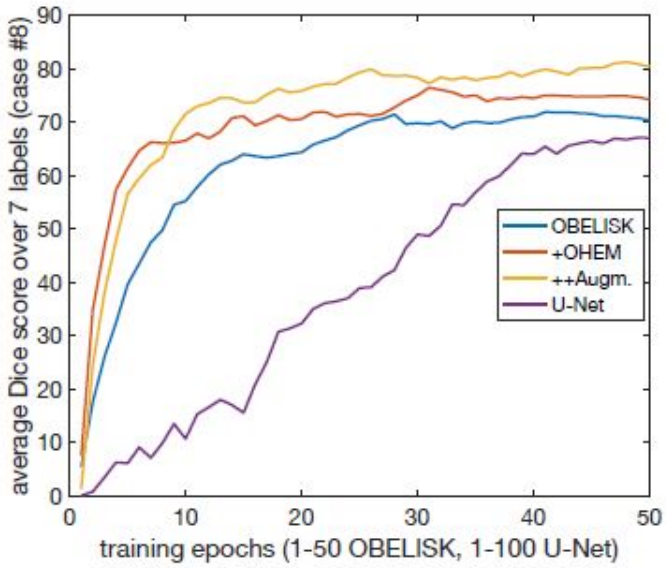


Figure 5.29: Comparison according training speed

5.4.1.4 Conclusion

The authors conclude as follows:
The OBELISK approach combining the OBELISK layer with some $1 \times 1 \times 1$ convolutions ($1 \times 1 \times 1$ Dense-Net) unifies previous multi-resolution, cascaded dilation and deformable convolution approachs into a single framework. Different aspects of architectural design as size or scale of convolutions has no longer be determined in a complicated, time consuming manner, necessarily, since the new approach automatically learns to adapt the receptive field. Moreover OBELISK network impressively outperforms multiscale U-Net by overlapping its Dice Score by 5.5% despite reducing the number of parameters by about 90%.

Relating to the mind barrier that was spoken to in their addressed issues the authors clearly showed that **deep networks, small kernels and multiscale or dilation architecture is not always the best choice**, at least for their special application case.

I want to close this subsection by noting that the presented summary does not cover all topics that are explored in the publication by Matthias P. Heinrich et al..
Not going into detail, they for instance also performed so called 'deformable registration' in 2D using OBELISK features.

5.4.2 Adaption to image classification

Aiming for consistency with the innovative idea I wanted to change the originally described OBELISK network in its binary version as slight as possible. Until the

prediction layer everything remains the same. Then my approach does some new operations that can be explored in the following table:

layer type	inputs	outputs	parameters
original OBELISK network without prediction layer	1	256	128,000
dense convolution with biases	256	128	32,896
average pooling	128	128	0
fully connected	128	64	8,192
fully connected	64	32	2,048
fully connected	32	16	512
fully connected	16	8	128
fully connected	8	# classes	# classes ·8
softmax	# classes	# classes	0

Note that inputs and outputs of the fully connected layers refer to the number of features instead of channels.
As depicted the orginal OBELISK network (without prediction layer) is followed by an average pooling that shrinks the processed volume to size $(1, 1, 1, 128)$ by global averages across the spatial dimensions. That step intends to concise spatial feature information into one single number in order to reduce the number of parameters for the following operations. The latter are defined by a fully connected network that successively decreases the number of neurons (128 to 64 to 32 to 16 to 8). These steps should weight features to generate less but more abstract ones. Finally the last 8 features are used to obtain the output layer which is combined with a softmax function that asserts reasonable prediction scores.

Including prediction layer this model at maximum (assuming 5 classes) achieves **171,816 parameters** in total. By choosing different numbers of classes this number only marginally differs.

5.4.3 Binary classification: KLG 0 vs 4

Since this work aims to introduce the OBELISK idea in context of 3D data handling I decided to do the optuna searches for this simple setting KLG 0 vs 4. Again the obejectives according convergence and multi val-loss are considered. For simplicity only batch size and learning rate were taken into account.

5.4.3.1 Optuna: Convergence Objective

Fortunately, the OBELISK approach was not affected by the programming mistake in contrast to the other approaches. Again every trial is trained on the same training set with 200 balanced samples and validated based on the same validation set with 500 balanced samples.

- **Experiment 22**: influence of batch size and learning rate on convergence

The unique study winner was trained afterwards.

- **Experiment 23**: study winner trained on 1000 balanced training samples and 300 balanced validation samples

5.4.3.2 Optuna: Multi Val-Loss Objective

The robust performance objective was investigated by the following experiment.

- **Experiment 24**: influence of batch size and learning rate on robust performance

Finally the unique study winner was trained once in a small training data setting and once including all training data.

- **Experiment 25**: study winner trained on 1000 balanced training data and 300 balanced validation data
- **Experiment 26**: study winner trained on all training data (11,438 samples with KLG 0 and 1,192 samples with KLG 4) and 300 balanced validation data

5.4.4 Evaluation on Test Set

At the end of these experiments the performance of the best epoch of the best study-winner is evaluated on the test set that contains 2851 healthy knees and 219 severely diseased ones.

5.4.5 Binary Classification: KLG 0 vs 2

The study winner of the multi val-loss objective was also trained in KLG 0 vs 2 classification setting, again on small training data and all training data.

- **Experiment 27**: study winner trained on 1000 balanced training data and 500 balanced validation data
- **Experiment 28**: study winner trained on all training data (11,438 samples with KLG 0 and 7,247 samples with KLG 2) and 1,600 balanced validation data

Finally the performance of the best epoch of experiment 28 is evaluated on the test set that contains 2851 healthy knees and 1671 minimal diseased ones.

5.5 Explainable Artificial Intelligence

This short section aims to show the importance of this quite new field. Furthermore, two simple approaches are applied.

It is to note that this topic could cover an own thesis. Here only small insights are given, that should motivate future work.

5.5.1 Intention

Explainable Artificial Intelligence (XAI) was invented due to the following problem: deep neural networks are so called blackbox models. That means that the network engineer indeed can apply theses models to solve real world problems, but however has no clue **how** the networks came to their decisions. Internally millions of neurons recognise pattern in the data but due to the size and high degree of non-linearities in a neural network it cannot be interpreted in which way they does. XAI methods are designed to bring some light into the darkness.

Especially in medicine trust in the used tools is of highest priority. A model that predicts perfectly well is not enough to convince the medical practitioners. When model based decisions affect a patients life then the models also should be explainable. Another aim of XAI is to generate domain specific knowledge. In case of medical image analysis these approaches might identify unknown biomarkers for a certain disease.

5.5.2 Different Approaches

There exist lots of distinct methods that could be categorised as follows, orienting at taxonomy of [67]:

Intrinsic vs post hoc

Intrinsic XAI approaches are related to models which are interpretable (e.g. decision trees) by its nature while post hoc approaches are applied to more sophisticated models when the training is already done (e.g. neural networks).

Result of interpretation method

- Feature summary statistic: e.g. feature importance
- Feature summary visualisation: visualise feature summary statistic
- Model internals: for intrinsically interpretable models, e.g. interprete weights of linear regression
- Data point: return already existent or new data point, e.g. return pixel that influences the CNN decision the most
- Intrinsically interpretable model: approximate black box model by an intrinsically interpretable model

Model specific vs model agnostic

Model specific tools, as the name suggests, only rely on a certain class of models as neural networks while model agnostic methods work for each machine learning approach.

Local vs global

Local refers to explanations that only rely on an individual prediction where global refers to observations of the entire model.

5.5.3 Saliency and GradCAM maps

In the following two special methods are portrayed which are practically applied in this work. According to the taxonomy above, a saliency map respectively a GradCAM map is a model specific, post hoc, local XAI tool that returns data. Both of them are especially designed to interprete CNNs for image classification tasks.

Saliency map/ Vanilla Gradient (see [68])
For a certain input image this method aims to evaluate which pixels contribute most to the CNNs prediction. In other words, saliency maps give insights about the attention of the CNN. The theory behind that method is pretty simple: one computes the gradient of the network representing function with respect to the inputs. These gradients tell how a small change in the input values would change the predicted output score. By visualising different gradients by different colors it is possible to obtain such a saliency map, for instance:

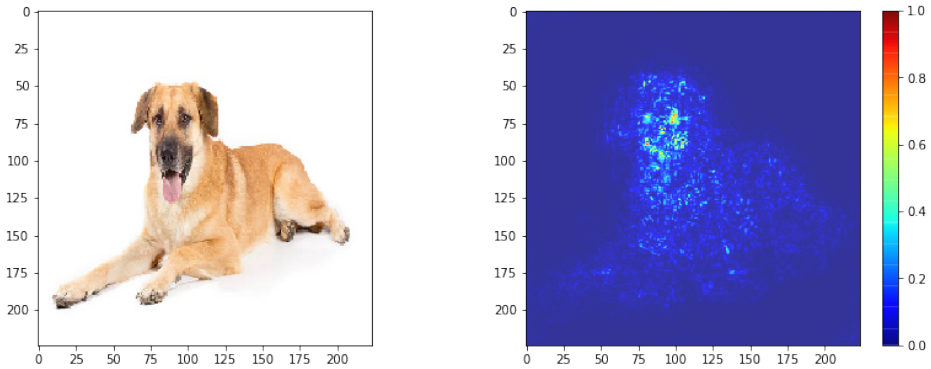


Figure 5.30: Saliency map highlighting pixels in the input image contributing the most to dog classification [69]

GradCAM map (see [70])
GradCAM is another possibility to visualise attention according to an individual input. Here, gradient information is used to generate weighted combinations of feature maps from the last convolution layer. The idea is that this convolution layer is used to get spatial information that is lost in the dense layers afterwards.
The following figure show gradCAM maps for different classes for a model that was trained only on images containing one class. This even yields the basis for a object localisation method.

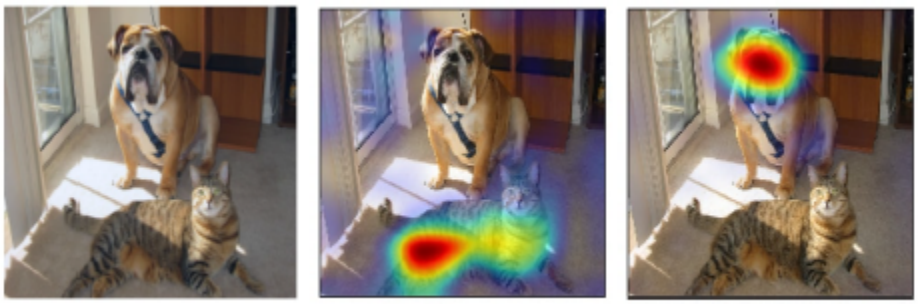


Figure 5.31: GradCAM maps for different classes [71]

To be more precise, I used the so called gradCAM++ approach that is an improved version of the original gradCAM. It is to note that I chose to apply the these methods only for models with very certain predictions.

5.6 Experiments Overview

In the following table you can find a summary of all investigated experiment settings, where bs = batch size, lr = learning rate, wd = weight decay, sh = boolean shuffle and opt = boolean AMSgrad optimiser.

no.	classification	network	type	parameters	training	validation	intention
1	KLG 0 vs 4	inc_resnet_v2	single run	fixed	50	500	influence training size
2	KLG 0 vs 4	inc_resnet_v2	single run	fixed	100	500	influence training size
3	KLG 0 vs 4	inc_resnet_v2	single run	fixed	200	500	influence training size
4	KLG 0 vs 4	inc_resnet_v2	single run	fixed	500	500	influence training size
5	KLG 0 vs 4	inc_resnet_v2	single run	fixed	1000	500	influence training size
6	KLG 0 vs 2	inc_resnet_v2	optuna convergence	bs	200	500	bs influence on conv
7	KLG 0 vs 2	inc_resnet_v2	optuna convergence	fixed	200	500	variance
8	KLG 0 vs 2	inc_resnet_v2	optuna convergence	bs	200	500	bs influence on conv (distorted)
9	KLG 0 vs 2	inc_resnet_v2	winner convergence	fixed	2000	1600	performance
10	KLG 0 vs 2	inc_resnet_v2	optuna multiloss	bs/lr/wd	200	500	robust performance
11	KLG 0 vs 2	inc_resnet_v2	winner multiloss	fixed	18685	1600	winner validation
12	KLG 0 vs 2	inc_resnet_v2	optuna singleloss	bs/lr/wd	200	500	performance
13	KLG 0 vs 2	inc_resnet_v2	winner singleloss	fixed	18685	1600	winner validation
14	JSN_M 0 vs 3	inc_resnet_v2	single run	fixed	400	118	proof of concept crop
15	JSN_M 0 vs 2	inc_resnet_v2	single run	fixed	400	200	proof of concept crop
16	JSN_L 0 vs 2	inc_resnet_v2	single run	fixed	400	137	proof of concept crop
17	KLG 0 vs 4	siamese	single run	fixed	200	500	proof of concept siamese
18	KLG 0 vs 2	siamese	optuna convergence	bs	200	500	bs influence on conv (distorted)
19	KLG 0 vs 2	siamese	winner convergence	fixed	18918	1600	winner validation
20	KLG 0 vs 2	siamese	optuna multiloss	bs/lr/wd/sh/opt	200	500	robust performance
21	KLG 0 vs 2	siamese	winner multiloss	fixed	18918	1600	winner validation
22	KLG 0 vs 4	obelisk	optuna convergence	bs/lr	200	500	convergence
23	KLG 0 vs 4	obelisk	winner convergence	fixed	1000	300	winner validation
24	KLG 0 vs 4	obelisk	optuna multiloss	bs/lr	200	500	robust performance
25	KLG 0 vs 4	obelisk	winner multiloss	fixed	1000	300	winner validation
26	KLG 0 vs 4	obelisk	winner multiloss	fixed	12630	300	winner validation
27	KLG 0 vs 2	obelisk	winner above	fixed	1000	500	winner validation
28	KLG 0 vs 2	obelisk	winner above	fixed	18685	1600	winner validation

6 Results

The following chapter is dedicated to show results and interprete describe their meaning that can directly been derived. Later on, in chapter 7, we interprete these results in a more abstract and summarising manner.

The structure is almost the same as the structure that was chosen for the method chapter. Training metrics are always depicted in blue while validation metrics are presented in red. Losses, learning and weight decay related plots are shown in logarithmic scale. That makes the visualisation more valuable.

6.1 Initial Approach: 3D CNN with inception blocks and residual connections

Let's start with the results of the initial approach.

6.1.1 KLG 0 vs 4

Experiment 1

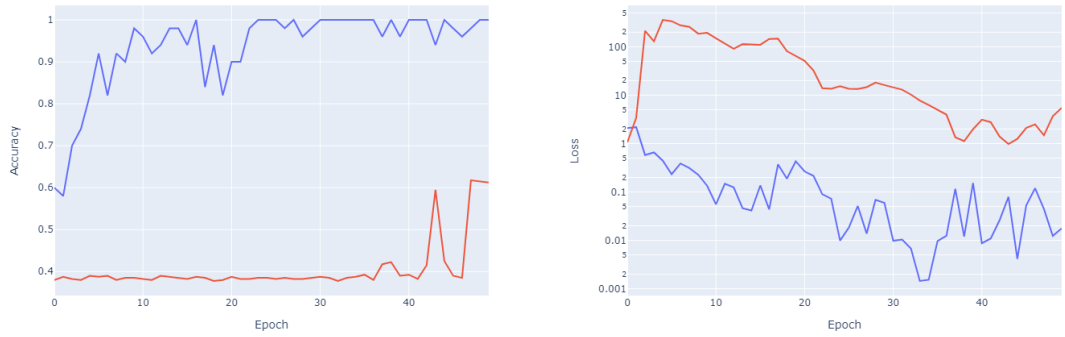


Figure 6.1: Training on 50 samples

From approximately epoch 25 the model almost predicts perfect on training set in every epoch that follows. The validation, however, remains really bad every epoch and only slightly exceeds 60% a few times.

Experiment 2

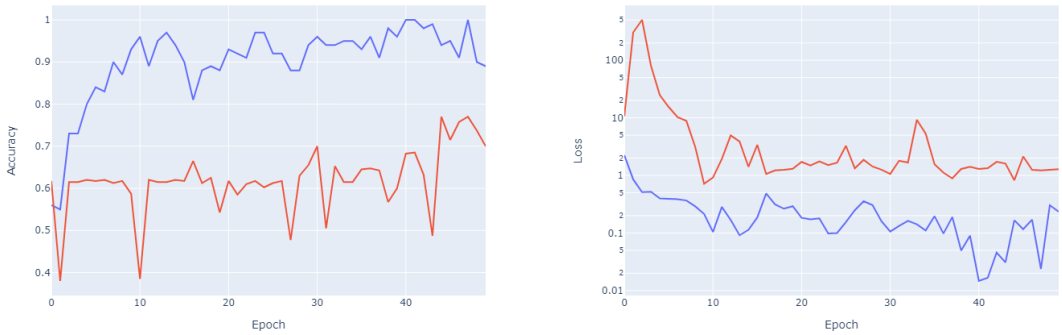


Figure 6.2: Training on 100 samples

Here the training curve is quite more fluctuating but Though the training curve is more fluctuating the training set is again perfectly learned. The validation curve is strongly fluctuating and now can 75% at the end.

Experiment 3

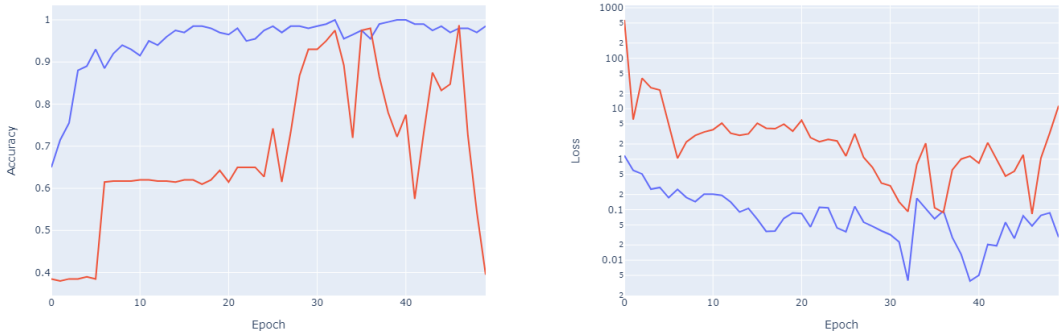


Figure 6.3: Training on 200 samples

The training progress is barely fluctuating and again increases continously. Validation again highly fluctuates but now several time reaches close to 100%.

Experiment 4

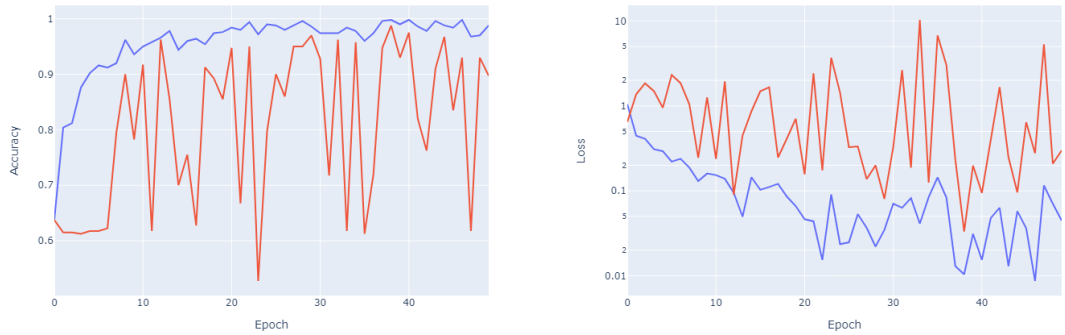


Figure 6.4: Training on 500 samples

The training curve is essentially the same as in the previous experiment. The validation progress is extremely fluctuating even from the beginning. It could reach a good performance (greater than 90%) very often.

Experiment 5

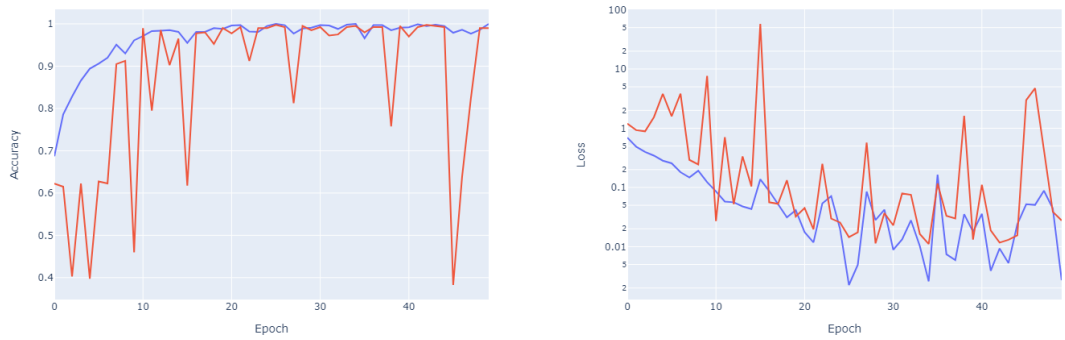


Figure 6.5: Training on 1000 samples

The training process is almost a smooth increasing curve and validation again fluctuating. In contrast to the previous experiment the validation curve here has intervals with almost no fluctuation.

In order to make these experiments more comparable they are plotted together in the figures below:

Figure 6.6: Training curves together

Figure 6.7: Validation curves together

While the training curves are quite similar for each experiment the validation curves totally differ. The bigger the training size the more early the validation achieves good results.

To conclude the 0 vs 4 setting I would say that the last experiment shows that the initial approach is able to perfectly solve (accuracy 100%) this task, at least on the validation set.

6.1.1.1 Evaluation on test set

TP	TN	FP	FN	accuracy	precision	recall	f1 score	AUC
210	2838	13	9	0.99	0.94	0.96	0.95	1.00

The evaluation of test set underpins the great quality in this setting. All metrics together confirm the high performance of the model.

6.1.1.2 Explainable Artificial Intelligence

Diseased cases

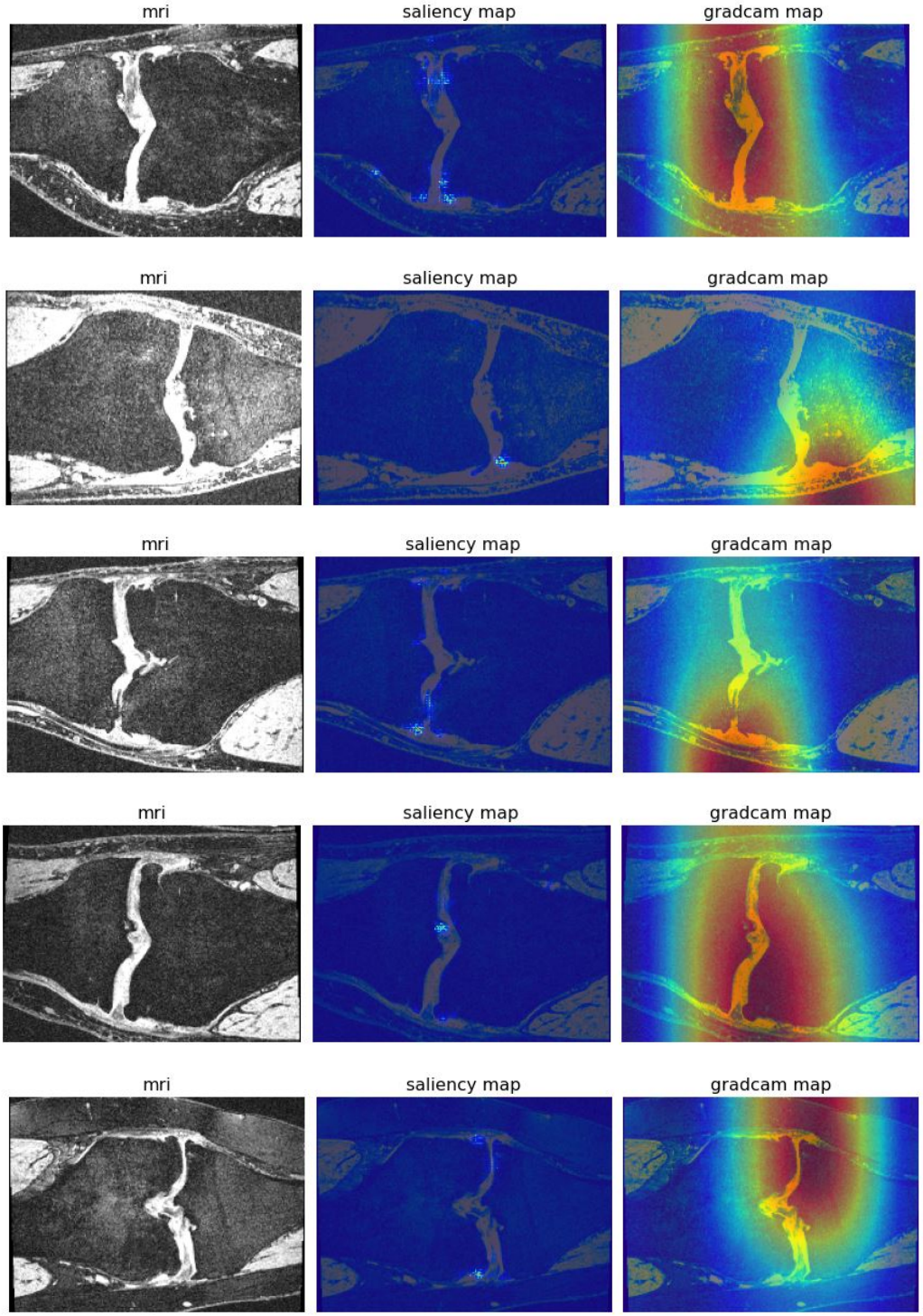


Figure 6.8: MRIs (left), saliency maps (middle) and gradCAM++ maps (right) for diseased cases

Healthy cases

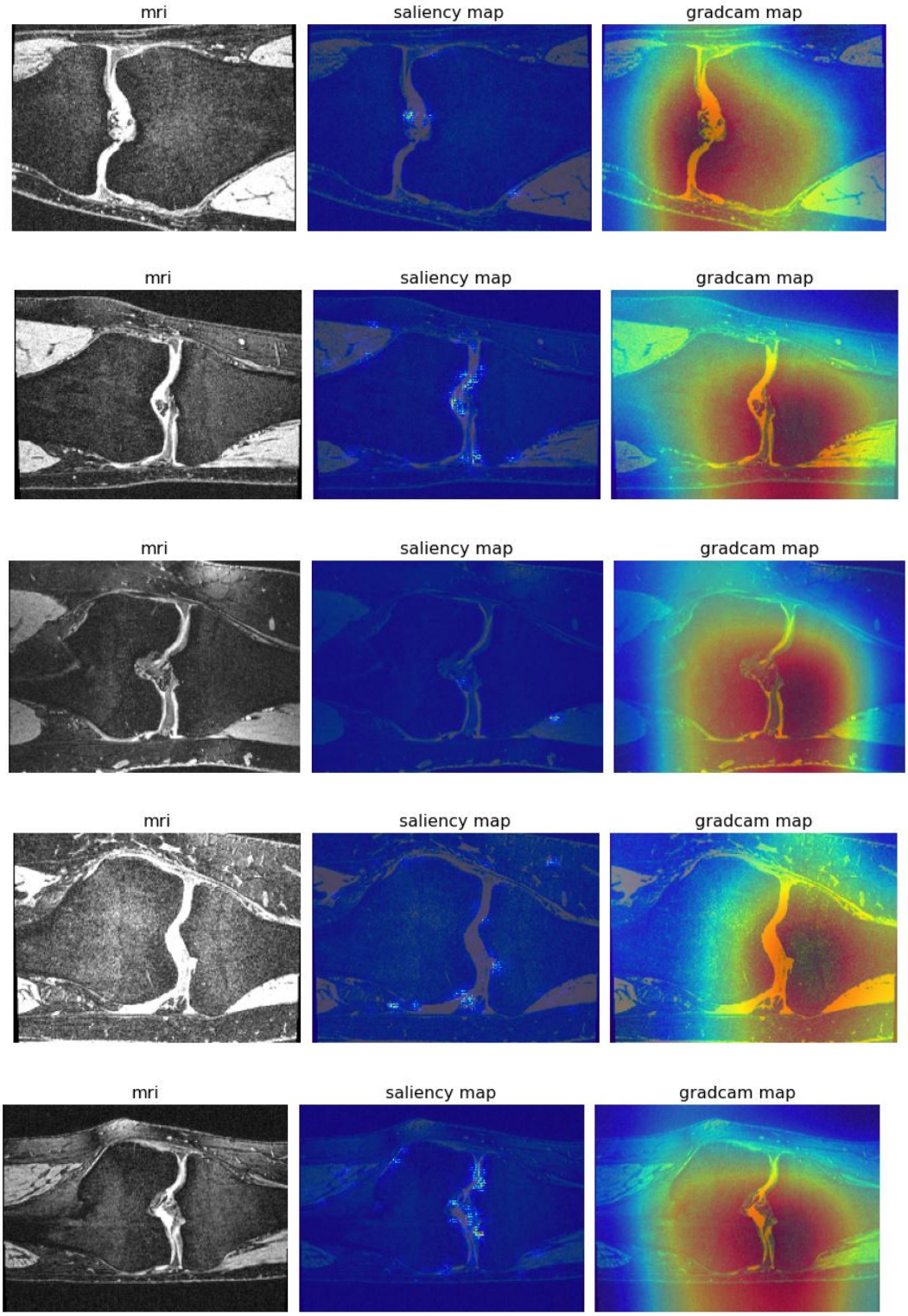


Figure 6.9: MRIs (left), saliency maps (middle) and gradCAM++ maps (right) for healthy cases

By looking at the saliency respectively gradCAM++ maps of 2D slices every sample highlights pixels in the joint line. To me the saliency maps not really differ from healthy to diseased cases while the gradCAM++ maps for the healthy knees tend to show larger important regions.

6.1.2 KLG 0 vs 2

6.1.2.1 Convergence objective

Experiment 6, 7 and 8

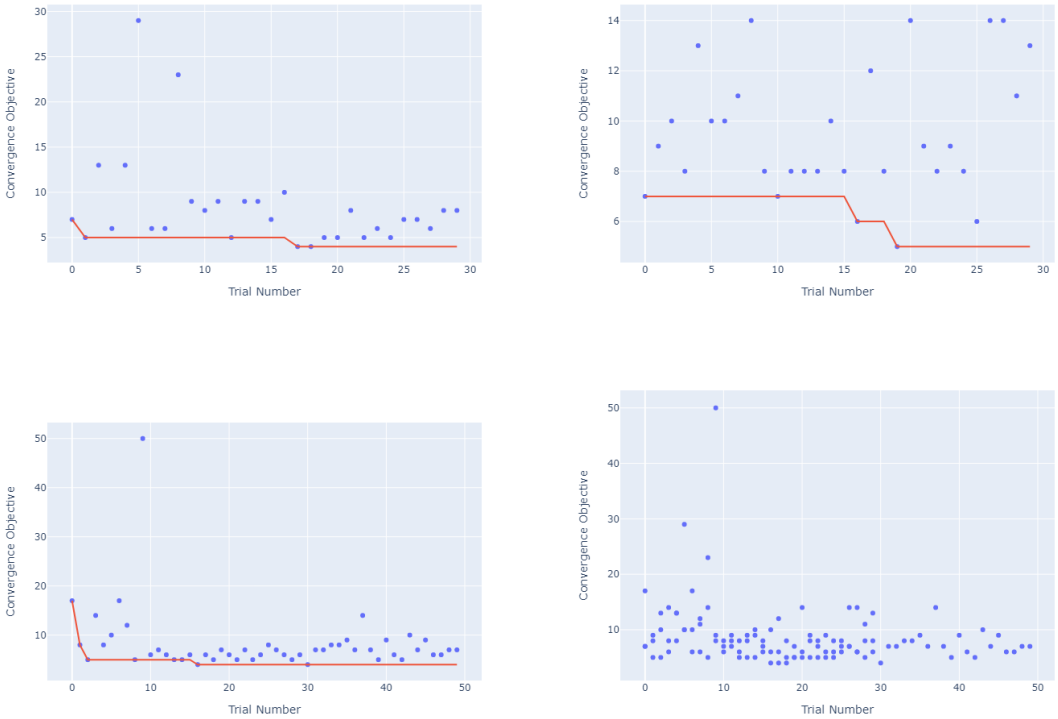


Figure 6.10: Optuna convergence objective: history for experiment 6 (up left), experiment 7 (up right) and experiment 8 (down left), joined plots (down right)

As the initial intentions for these experiments altered due to the coding mistake, experiment 6 and 8 essentially aim to optimise the batch size and experiment shows results for fixed learning rates and batch sizes with the objective of improving convergence speed. All in all, the trials differ in achieving the objective by about 5 epochs at maximum as the joined plot illustrates.

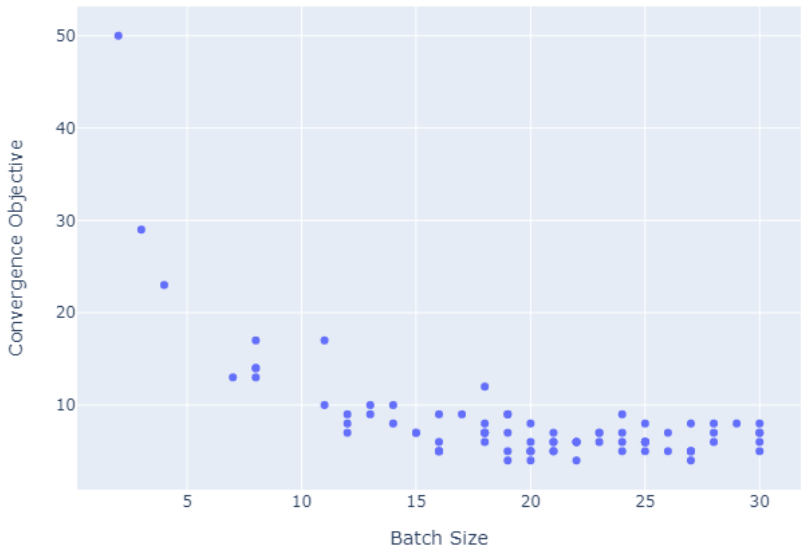


Figure 6.11: Influence of batch size

This plots shows a general trend that bigger batch sizes yield better convergence until batch size 15. Then it does not really matter which batch size is chosen.

Experiment 9

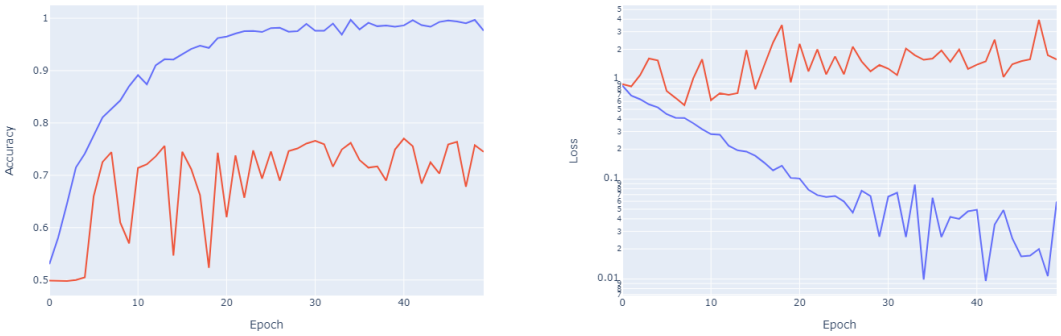


Figure 6.12: Study winner

The study winner shows a quite smooth training curve but a fluctuating validation curve that almost achieves 80% accuracy.

6.1.2.2 Multi val-loss objecive

Experiment 10

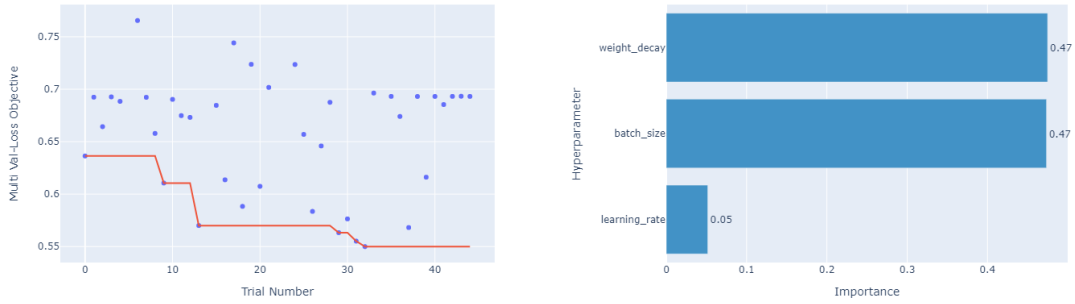


Figure 6.13: Optuna multi val-loss objective: history and parameter importance

Experiment 10 shows that weight decay and batch size are far more important than the initial learning rate in Adam optimisation concerning robust performance of the model. Over time the objective could decreased by almost 0.1.

Figure 6.14: Influence of batch size, learning rate and weight decay (from left to right)

According to parameter influence lower batch sizes and learning rates are more suitable but higher weight decay, instead.

Experiment 11

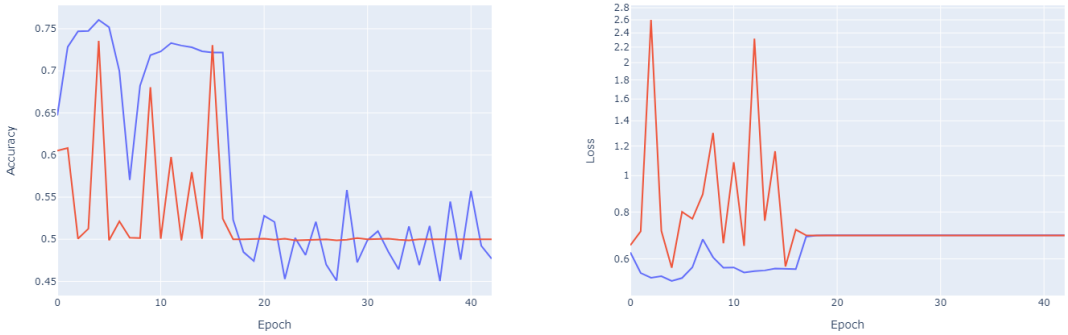


Figure 6.15: Study winner

The study winner is highly fluctuating in validation until it completely fails around epoch 20 and constantly remains at 50%.

6.1.2.3 Single val-loss objective

Experiment 12

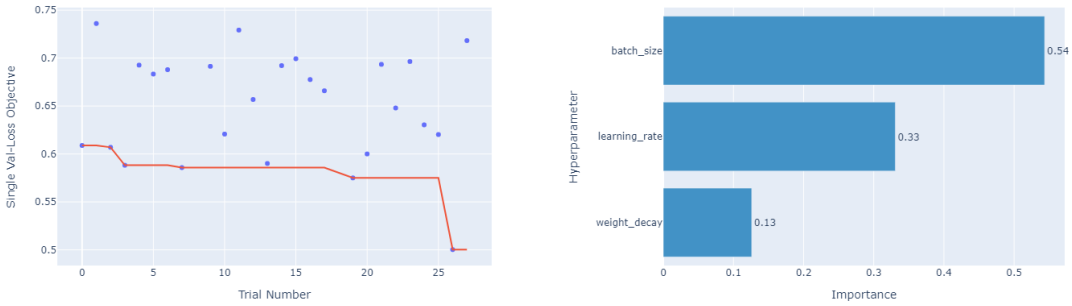


Figure 6.16: Optuna single val-loss objective: history and parameter importance

Here firstly nothing really happens until then the objective value could decreased by more than 0.06 at the end, resulting in an quality improving outlier trial. Batch size and learning rate are more important than weight decay for the performance.

Figure 6.17: Influence of batch size, learning rate and weight decay (from left to right)

No clear trends can be recognised for parameter influence.

Experiment 13

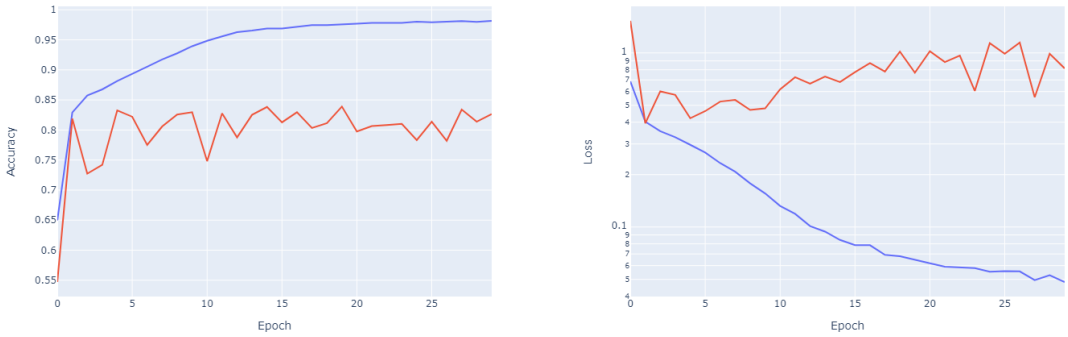


Figure 6.18: Study winner

The study winner yields a very smooth training curve and less fluctuating validation curve that achieves almost 85% at best.

6.1.2.4 Evaluation on test set

TP	TN	FP	FN	accuracy	precision	recall	f1 score	AUC
1318	2538	312	352	0.85	0.81	0.79	0.80	0.92

The test set surpasses the expectations with even 85% accuracy and quite balanced values for precision and recall.

6.2 Data Reduction Approach: cropping region of interest

6.2.1 JSN 0 vs 3, JSN 0 vs 2

6.2.1.1 JSN 0 vs 3

Experiment 14

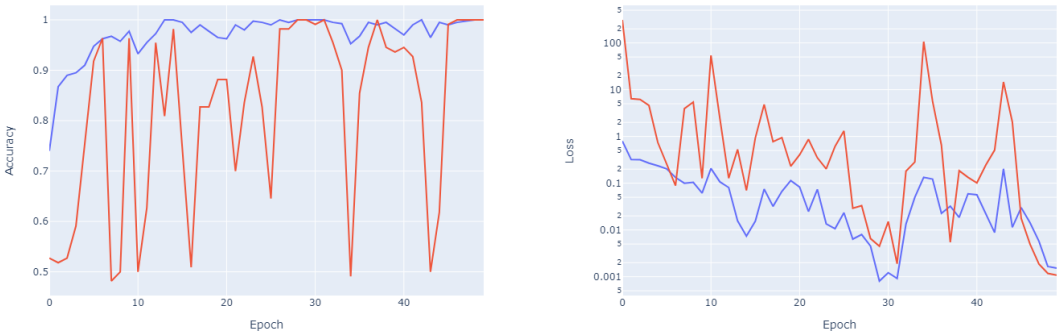


Figure 6.19: Training history for medial JSN classification

The validation curve is highly fluctuating but achieves almost perfect accuracy multiple times.

6.2.1.2 JSN 0 vs 2

Experiment 15

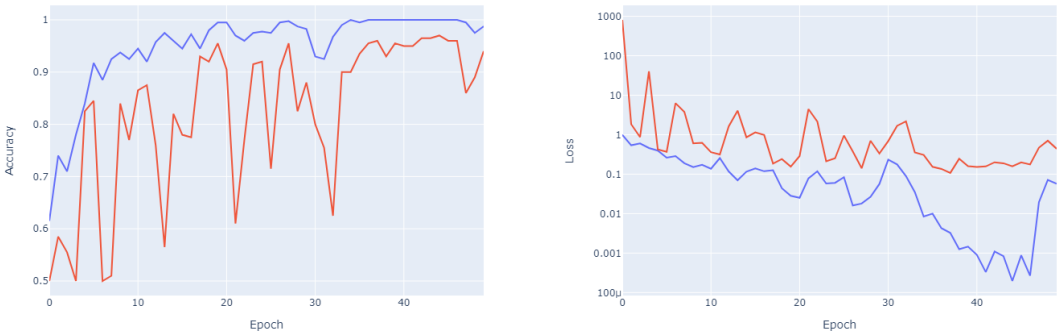


Figure 6.20: Training history for medial JSN classification

The validation is slightly less fluctuating and a bit worse in contrast to the experiment before.

Experiment 16

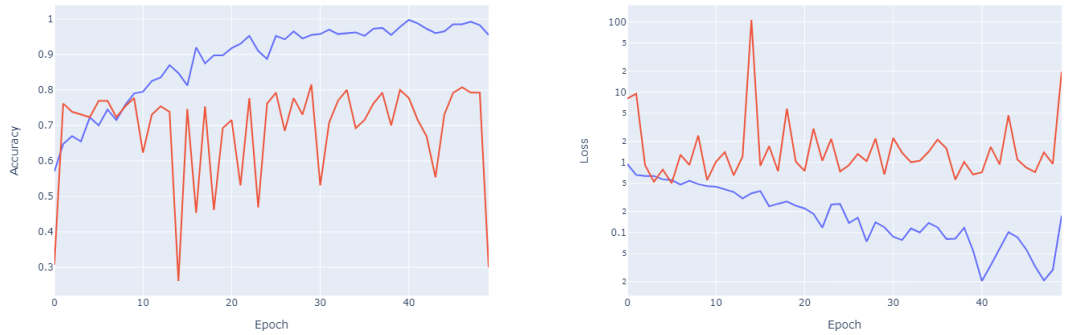


Figure 6.21: Training history for lateral JSN classification

Here the validation at best is about 15% worse. That induces the assumption that the input data were medial condyles.

Explainable Artificial Intelligence

Diseased cases

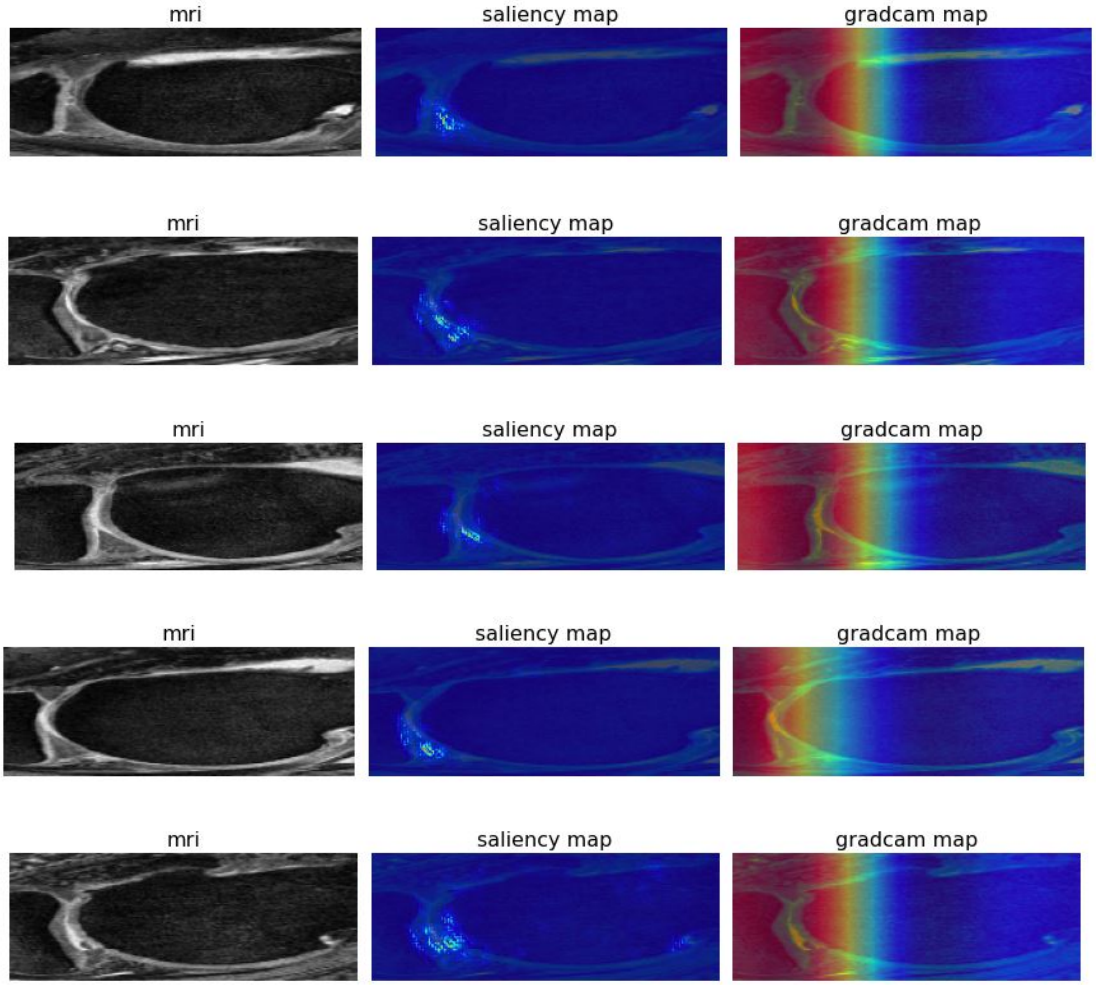


Figure 6.22: MRIs (left), saliency maps (middle) and gradCAM++ maps (right) for diseased cases

Healthy cases

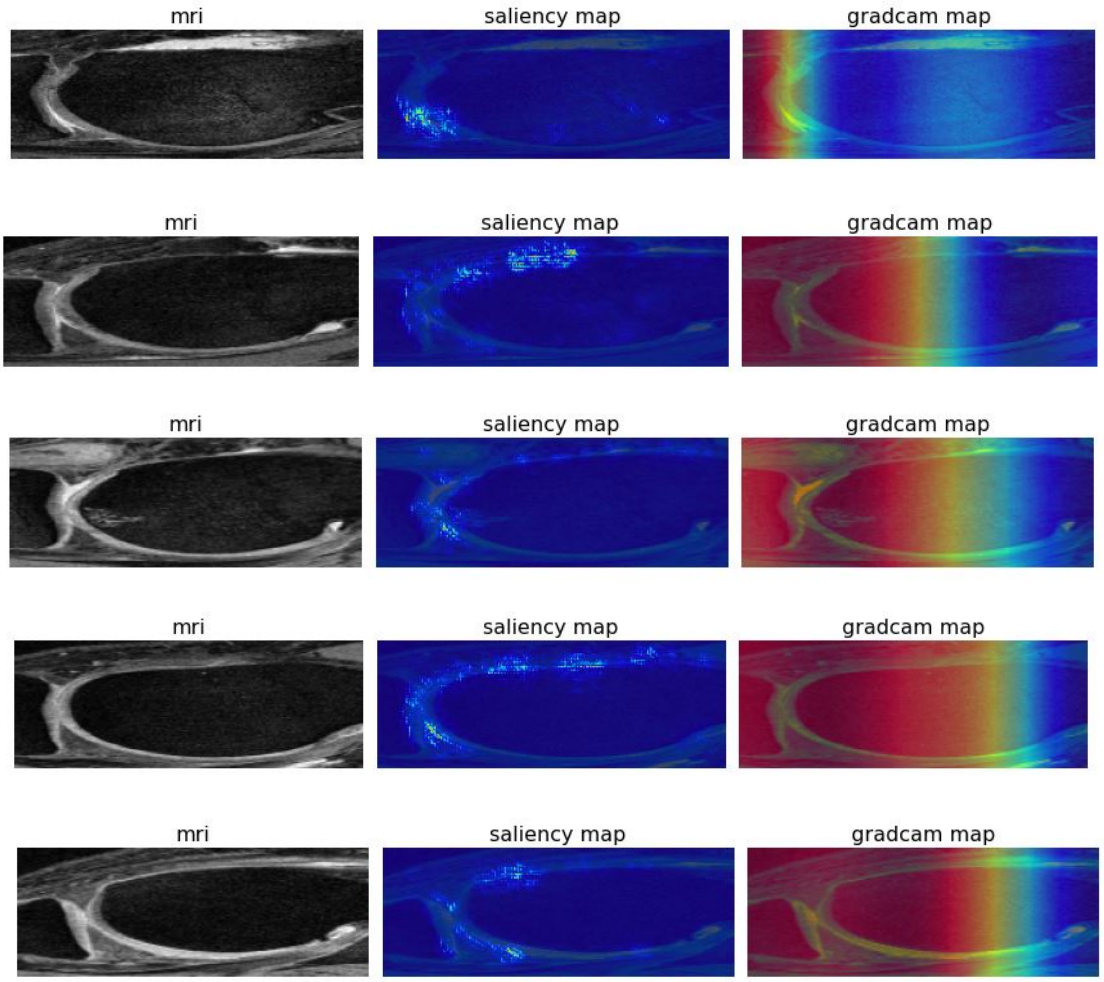


Figure 6.23: MRIs (left), saliency maps (middle) and gradCAM++ maps (right) for healthy cases

The previous maps confirm that the attention lies in the joint line while healthy knees are captured by more pixels in contrast to the diseased cases.

6.2.2 KLG 0 vs 4

Experiment 17

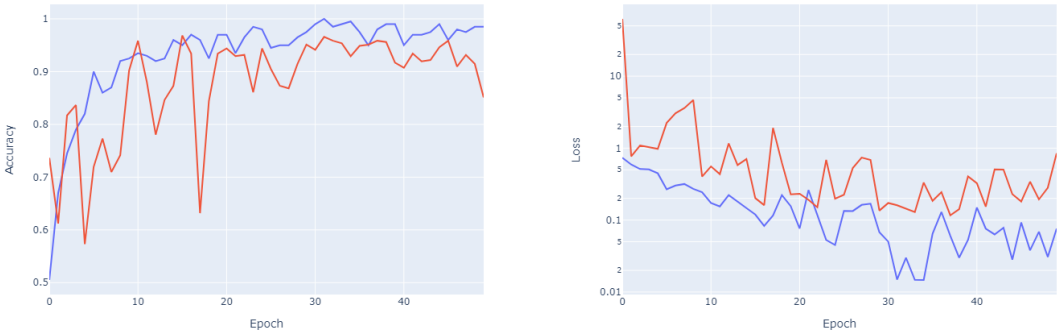


Figure 6.24: Training history

This experiment with arbitrary parameters achieved validation mostly over 90% after 20 epochs.

6.2.2.1 Evaluation on test set

TP	TN	FP	FN	accuracy	precision	recall	f1 score	AUC
356	3339	53	12	0.98	0.87	0.97	0.92	0.99

The test set performance is even better, but precision and recall differ by 10 %.

6.2.2.2 Explainable Artificial Intelligence

Diseased cases

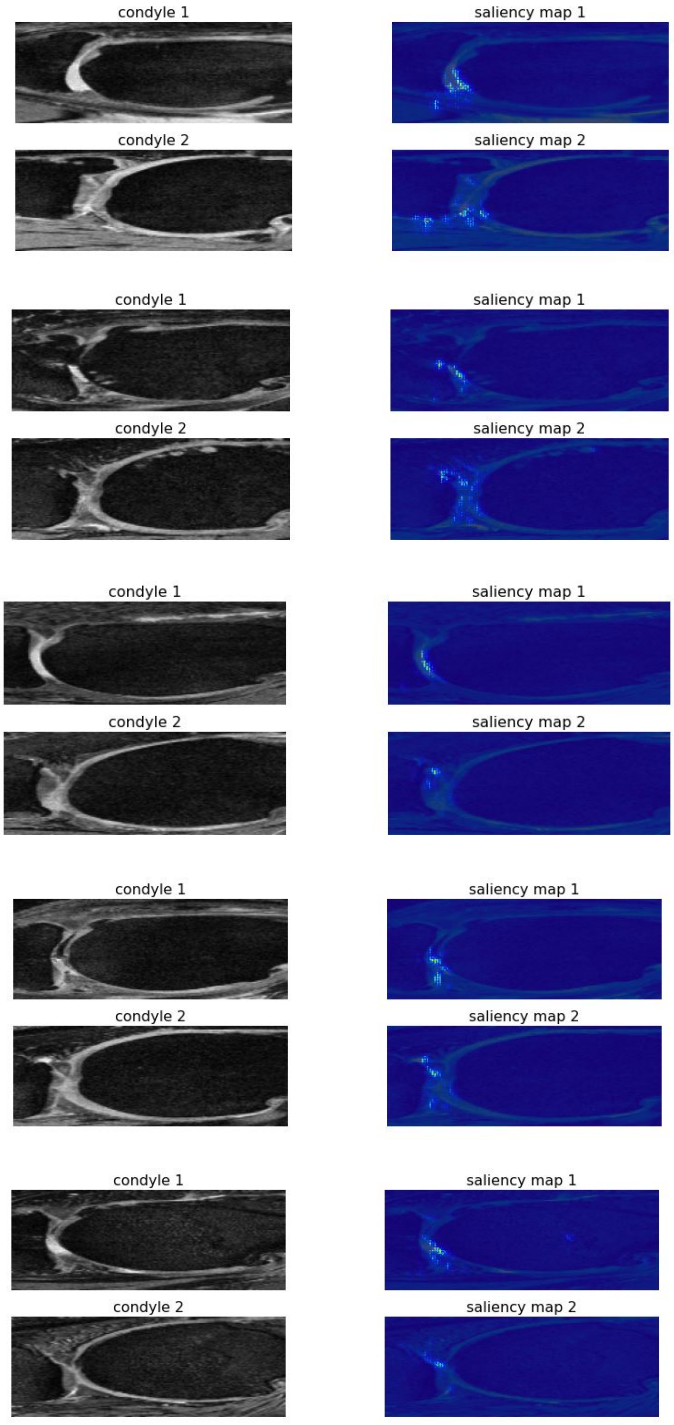


Figure 6.25: MRIs (left), saliency maps (middle) for diseased cases

Healthy cases

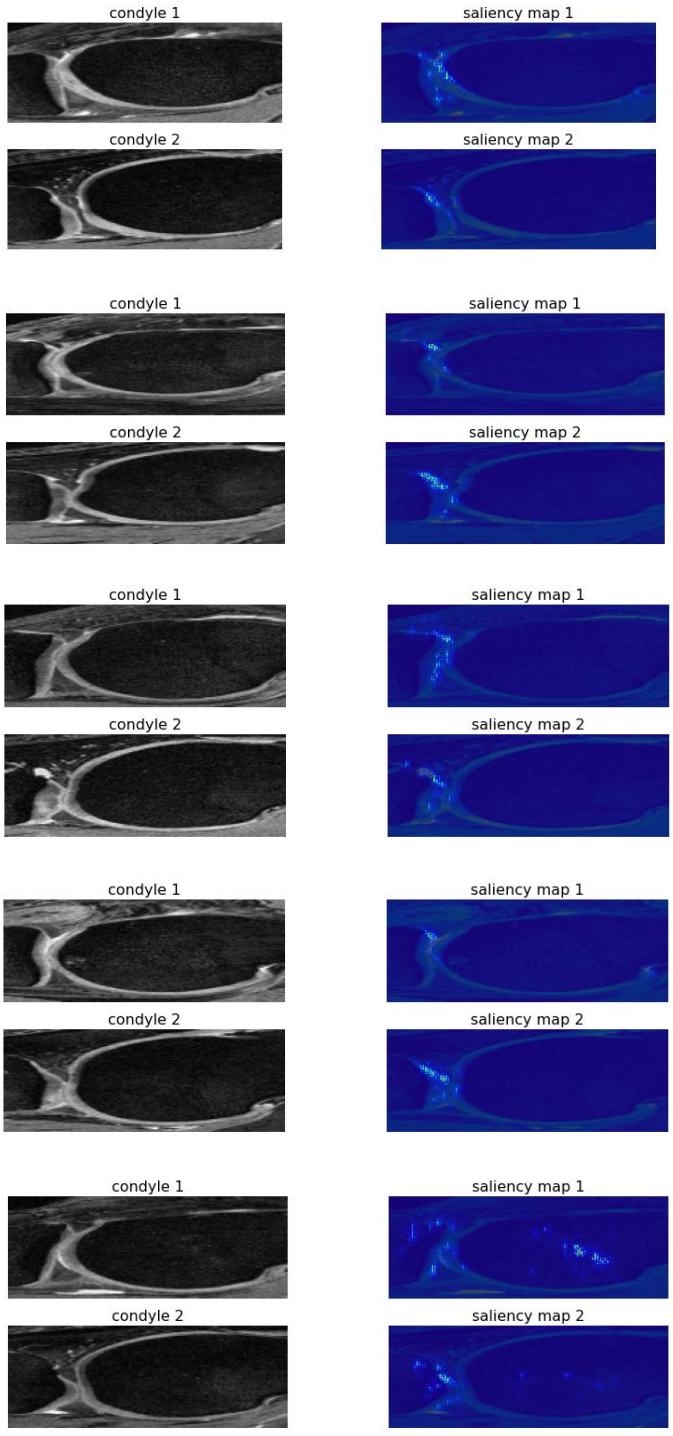


Figure 6.26: MRIs (left), saliency maps (middle) for healthy cases

Except for the last healthy sample each saliency map shows the attention on the joint line.

6.2.3 KLG 0 vs 2

6.2.3.1 Convergence objective

Experiment 18

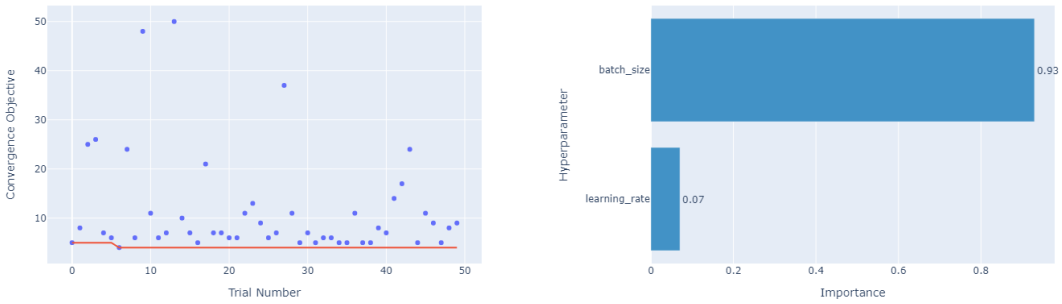


Figure 6.27: Optuna convergence objective: history and parameter importance

Since this experiment was affected by the programming mistake it now only optimised batch size according convergence speed. The diagram about parameter importance correctly confirms that by pointing out that learning rate is not important. There are several outlier cases that slowly or even not converge.

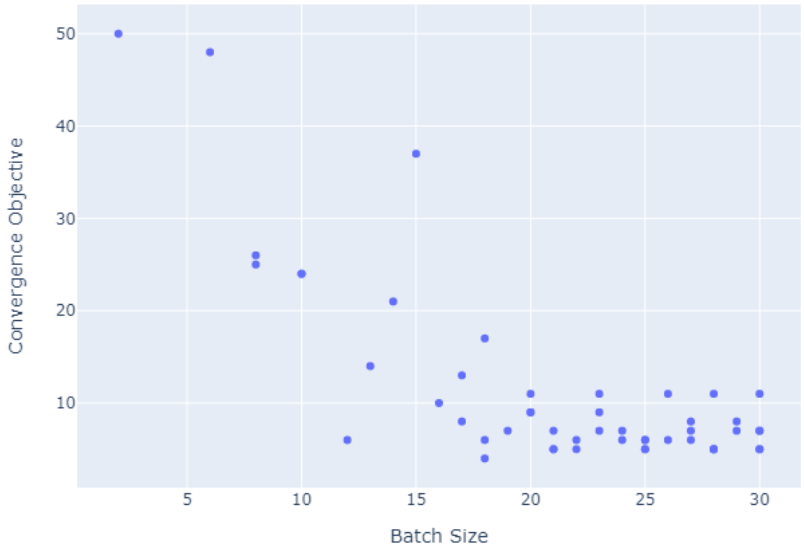


Figure 6.28: Influence of batch size

Batch sizes over 15 are preferable, but between 15 and 30 there is no much difference.

Experiment 19

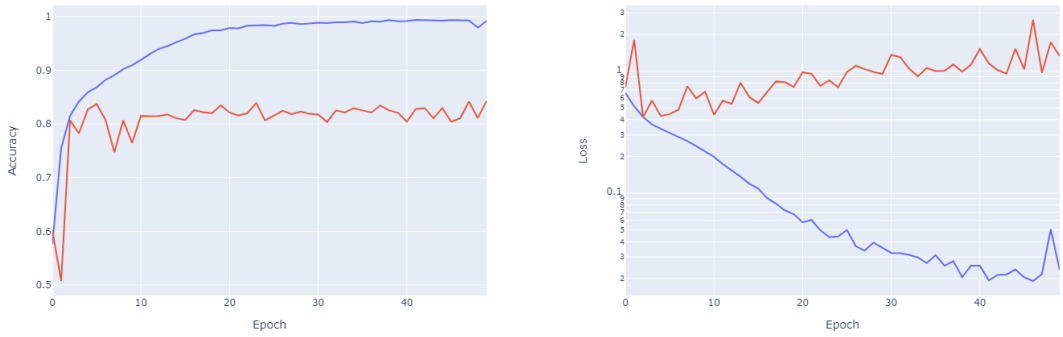


Figure 6.29: Study winner

The study winner's training curve is very smooth and the validation barely fluctuates. The model achieved about 83% at best in validation.

6.2.3.2 Multiloss val-loss objective

Experiment 20

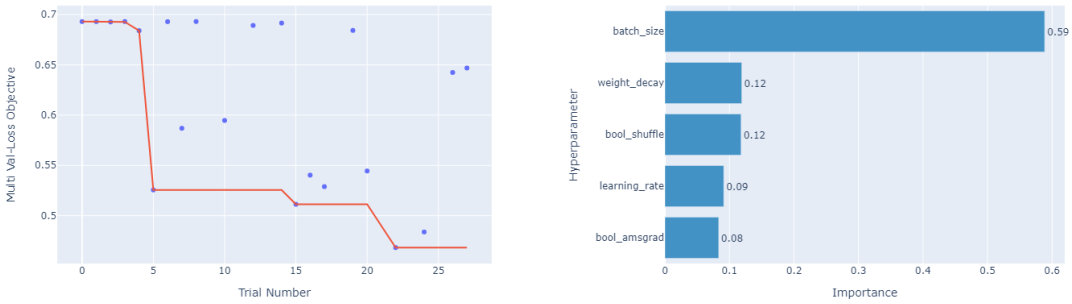


Figure 6.30: Optuna multiloss objective: history and parameter importance

That optuna run could reduce the objective value by more than 0.2. Batch size has clearly the most importance, the others are essentially the same important.

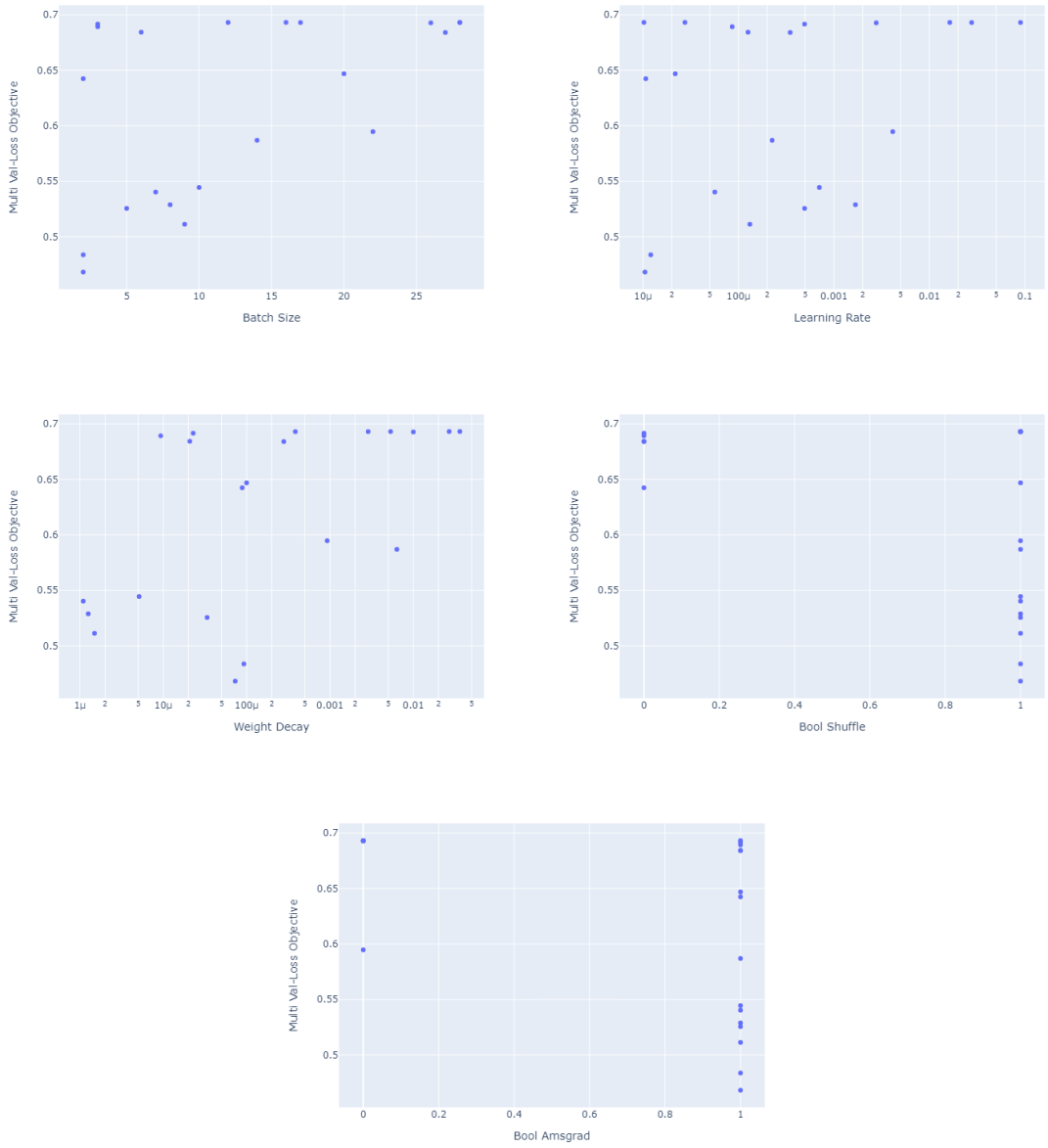


Figure 6.31: Influence of batch size, learning rate, weight decay, shuffling, other optimiser

Lower batch sizes, learning rate and weight decay is slightly preferable. Shuffling and the other optimiser seems to be good options.

Experiment 21

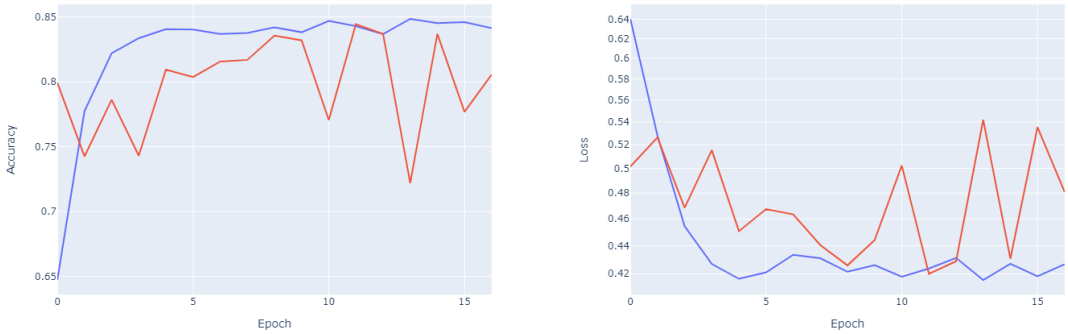


Figure 6.32: Study winner

The study winner is highly fluctuating in validation curve and even touches the training curve at two steps. The validation achieved almost 85% accuracy.

6.2.3.3 Evaluation on test set

TP	TN	FP	FN	accuracy	precision	recall	f1 score	AUC
1609	2851	545	355	0.83	0.75	0.82	0.78	0.91

The test set precisely confirms the validation accuracy. Precision is a bit lower than recall.

6.3 Model Reduction Approach: learnable filter offsets

6.3.1 KLG 0 vs 4

6.3.1.1 Convergence objective

Experiment 22

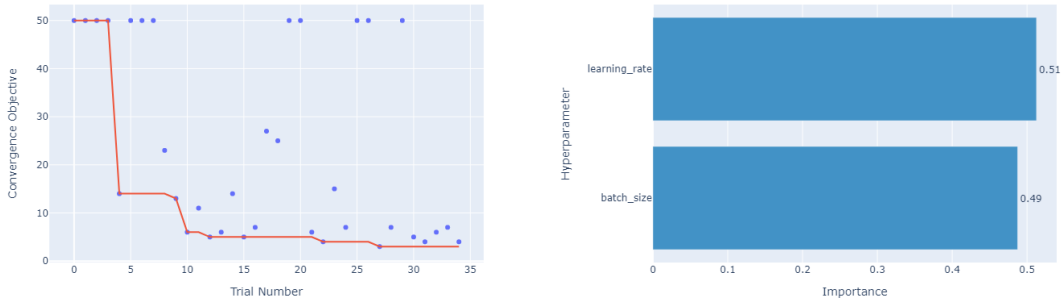


Figure 6.33: Optuna convergence objective: history and parameter importance

There are lots of outliers that could not achieve the defined accuracy threshold. Overall the differences in the objective value are quite big. Both of the parameters have the same importance.

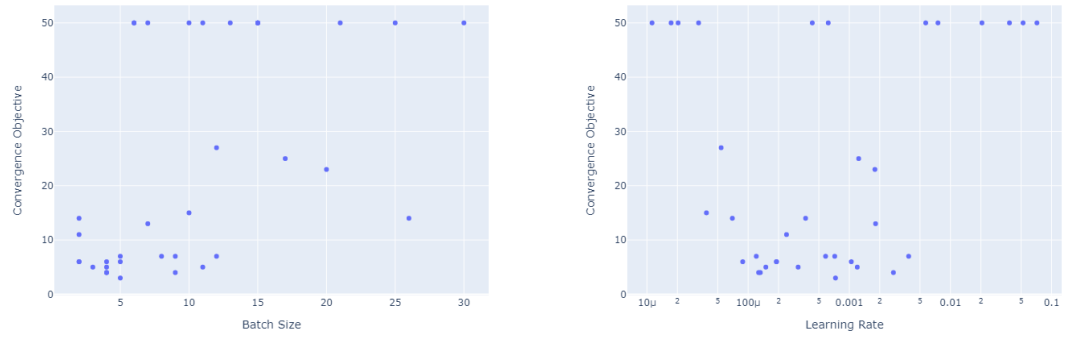


Figure 6.34: Influence of batch size and learning rate

Smaller batch sizes are preferable. For the learning rate it seems that it should not be too small and not be too big. It should rather lay in a certain interval, but due too few sampling outside of this interval this is just an assumption.

Experiment 23

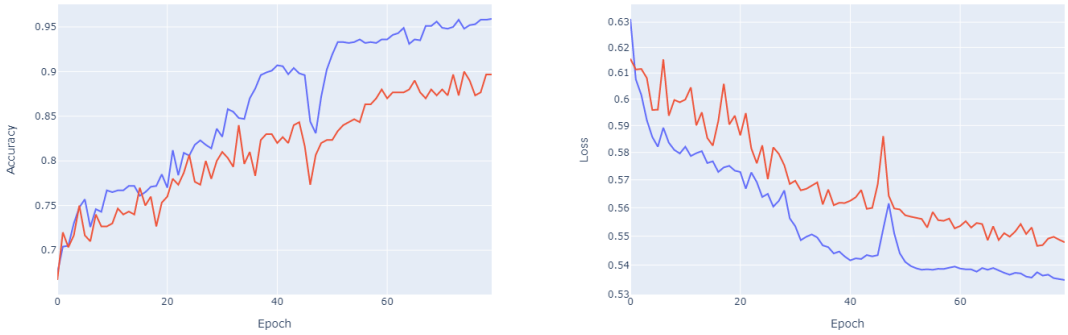


Figure 6.35: Study winner

The progress in both curves is quite linear except for one quite heavy outlier. About 90% were achieved in validation.

6.3.1.2 Multi val-loss objective

Experiment 24

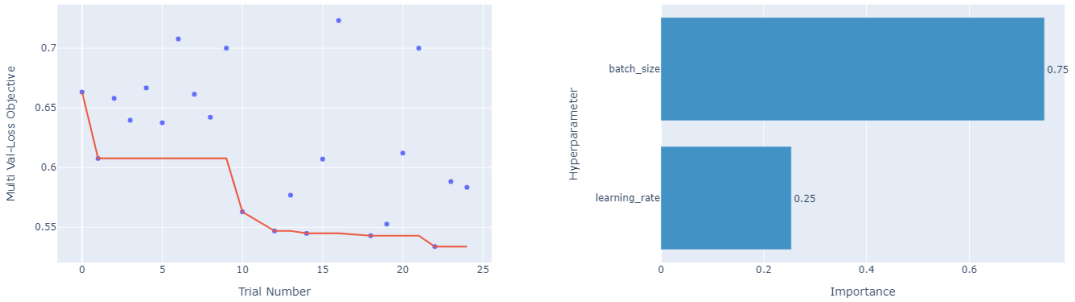


Figure 6.36: Optuna multiloss objective: history and parameter importance

The objective value could improved by more than 0.1. The learning rate turns out to be less important than the batch size here.

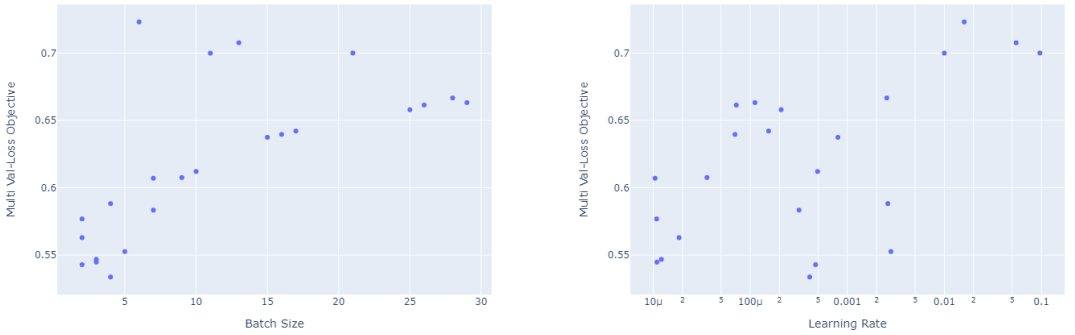


Figure 6.37: Influence of batch size and learning rate

Smaller batch sizes and learning rates seems more promising (for batch size quite clear, learning rate is not really clear).

Experiment 25

Figure 6.38: Study winner on small training data

The curves are again almost linear except for an outlier. About 87% validation accuracy was achieved.

Experiment 26

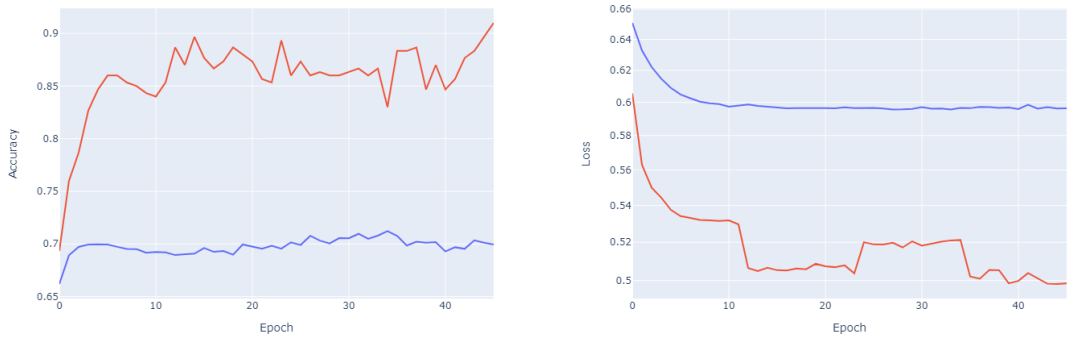


Figure 6.39: Study winner on all training data

Here the validation is much higher than the training (achieved more than 90%), seemingly due to class imbalance problems.

6.3.1.3 Evaluation on test set

TP	TN	FP	FN	accuracy	precision	recall	f1 score	AUC
213	1855	994	6	0.67	0.18	0.97	0.30	0.94

The performance has clearly decreased in contrast to validation. Precision is really bad where recall is really good.

6.3.2 KLG 0 vs 2

Experiment 27

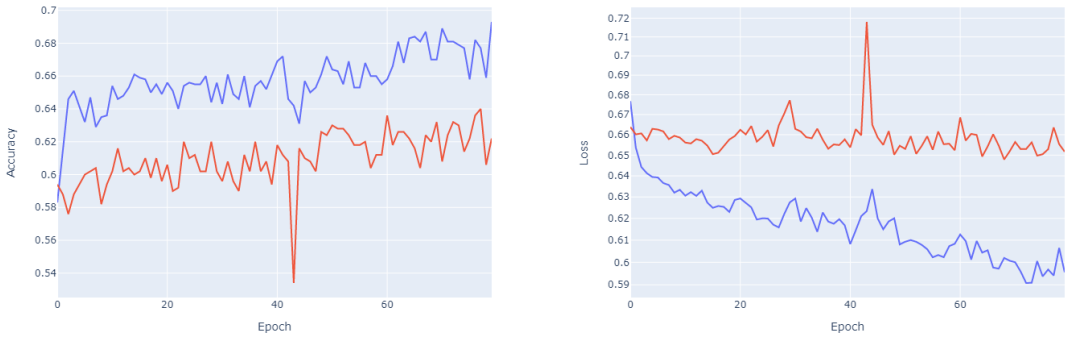


Figure 6.40: Winner from multiloss 0vs4 study on small training data

Training and validation follow similar trends upwards until about 64% validation accuracy with a small gap.

Experiment 28

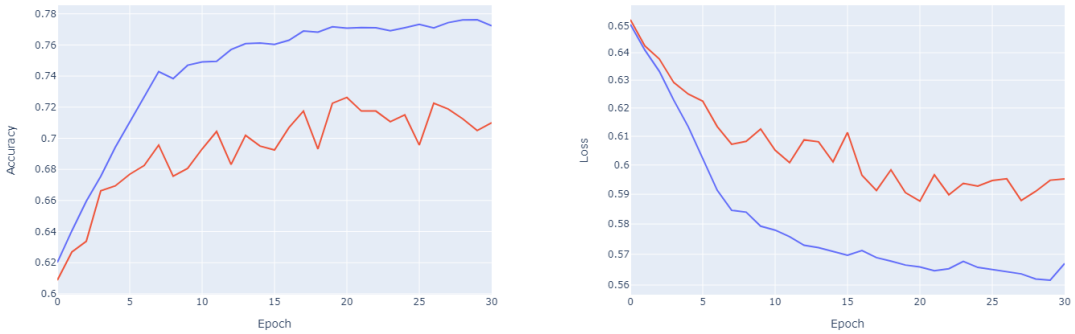


Figure 6.41: Winner from multiloss 0vs4 study on all training data

Here almost 73% were achieved invalidation. Training curve is quite smooth, validation is a bit fluctuating.

6.3.2.1 Evaluation on test set

TP	TN	FP	FN	accuracy	precision	recall	f1 score	AUC
1230	2133	717	440	0.74	0.63	0.74	0.68	0.77

The validation performance in terms of accuracy could be confirmed. Precision is slight worse than recall.
It is to note for AUC computation at OBELISK that the prediction layer does not include a softmax classifier.

7 Discussion and Conclusion

In this chapter some final considerations are made. To begin with, the latter chapter is discussed. For each approach a summary is given and afterwards a comparison between all of them is presented. Then the conclusion section aims to answer the questions that were initially stated as research objectives of this work. Finally some thoughts about potential future work are portrayed.

7.1 Discussion

7.1.1 Initial approach

KLG 0 vs 4 was perfectly been handled. For KLG 0 vs 2, batch size seems not to be really influential on convergence. For robust performance the choice of batch size, learning rate and weight decay indeed has an influence. For performance without robustness objective no trends could be seen. The best model yielded 85% accuracy on the test set.

7.1.2 Data reduction

The JSN classifications show proof of concept for the cropping procedure and let assume that medial condyles were picked for the input.
Great performance was achieved in KLG 0 vs 4 (98% accuracy). According to KLG 0 vs 2, batch size seems to be influential on the convergence objective. Batch size is also the most important parameter according robust performance. Only shuffle and other optimiser seems to have an influence on robust performance. The best model yielded 83% accuracy.

7.1.3 Model reduction

Batch sizes and learning rates seem to be influential on convergence. Batch size is the most important parameter for robust performance and clearly influences it. The best model yielded 67%.

7.1.4 Comparison

In each approach the design of batch size seems to be influential in certain setting.
Due to the accuracy, the initial and siamese approach yielded similar winning architectures while OBELISK is about 20% worse. That has to be seen in comparison to the number of parameters. The first two approaches has hundreds of millions, the Inception ResNet in 2D even has around 56 million parameters. OBELISK, in contrast, only

has 170,000 parameters. It is up to the context how to weight these properties in order to determine an overall winner. Including the fact that OBELISK shows innovative new principles and questions former wisdom, this approach is the most convincing in my personal point of view.

7.2 Conclusion

- Is a big CNN architecture that is successful for 2D data simply extendable to 3D in context of knee osteoarthritis classification? How does it perform in 3D?

Yes, indeed this simple straight forward intention could successfully realized by the Inception ResNet v2 that is known for its remarkable results in two dimensional image analysis. The extension of inception blocks was simple and turned out to work. It almost perfectly distinguishes between healthy and severely diseased cases and also yielded high performance on the binary classification between KLG 0 and KLG 2. In turn, inception blocks and residual connections are as assumed also a promising architectural design for three dimensional data.

- What exactly are the technical limitations when it comes to 3D data processing by heavy weight architectures?

But, and here is the big drawback, this convincing quality is dependent on dramatically big computational power. Since it internally uses millions of parameters average hardware settings tend to get quickly exhausted by such an architecture. This leads to the fact that the network engineer either has to dramatically reduce the input volume or the batch size. As both of them drive the overall model performance the full potential of such a network cannot be used. Anyway, in near future also standard GPU setups will be likely to handle an inception resnet v2.

- What could be a suitable ROI cropping approach in context of knee osteoarthritis classification? How does it perform?

The intention was to put the focus close to the joint space as knee OA affects this region the most, at least in terms of structural changes. In addition to that the two-stranded shape of the knee joint is predestined to cut it in the middle. In this way, two similar condyle sides were obtained for further processing. According to performance this approach is only slightly worse in contrast to the first method. But it could drastically reduce training time as the input tensors were smaller. The drawback is the enormous amount of effort to realize such an approach. It is to note that I was in the lucky case to simply utilize segmentation data is it was already there. Assuming that one has to start from scratch I would doubt that the effort is worth it, at least in terms of practical applicability.

- Is the alternative idea that was successful for medical image segmentation also valuable in context in knee osteoarthritis classification?

Here, in turn, no doubt is left in order to confirm the innovative nature of the developed OBELISK architecture. Although the model performance could not achieve the quality as in the other methods its benefits are obvious. The parameter count shows a reduction from the scale of hundreds of millions to hundreds of thousands comparing the conventional approaches with this one. That not only boosts the computational speed, but also makes this architecture accessible for almost every GPU setting, without drastically shrinking the input volume. That also means, that complicated preprocessing steps can be omitted. If that effort is invested in hyperparameter optimisations or further improvement techniques the performance also could become satisfying.

- Which ways exist for hyperparameter optimisation to tackle time constraint?

This work presented bayesian approaches for hyperparameter optimisation as a valuable alternative for time consuming approaches as grid search. Incorporating knowledge from previous objective evaluations forces the search to focus on promising combinations. From my point of view these methods are inevitable for deep learning models since the conventional approaches exceed any time constraint.

- Which are the most important hyperparameters and how do they influence the model training in each approach?

According to investigations in related work, essentially batch size, learning rate and weight decay are assumed to be crucial. Actually some others as learning rate decay are also important but by using Adam optimiser they are handled automatically. The parameter importance investigation by optuna rather shows problem specific results as general trends.

- Are there simple approaches to make the models explainable?

This work presented saliency and gradCAM++ visualizations as a first step to bring light into the blackbox models. Especially for medical imaging interpretability and explainability are of high interest since these properties generate trust in the practitioners.

7.3 Future Work

For future work this thesis suggests a bunch of possibilities. In terms of knee osteoarthritis analysis other classification scenarios would be interesting. While KLG 0 vs 2 is already quite satisfying multi class settings are rather challenging. The overall vision is to perfectly support radiologists in their decisions. Furthermore one could alter the labels. As the Osteoarthritis Initiative provides knee MRIs from same patients over distinct timepoints, one could develop computer aided prognosis besides diagnosis systems. When the progression of the disease would be predicted sufficiently well the physician could adapt his therapeutic measures even better.

To make data handling proposals one could also apply other techniques concerning 3D data handling. Multi-GPU and parallelisation setups could tackle the computational

bottleneck. Especially the siamese network architecture is promising to investigate. Furthermore attention mechanisms could be combined with the developed models. In order to avoid or at least accelerate cropping pre-procedures, deep learning based approaches could be taken into account that automatically crops region of interests.

Finally, explainability tools could be studied more extensively. For cropped condyles it would be interesting, for example, to capture relevant biomarkers for the prediction of medial and lateral joint space narrowing from knees that are not under load as in MRI data.

Bibliography

[1] Geoffrey E. Hinto Alex Krizhevsky, Ilya Sutskever. Imagenet classification with deep convolutionalneural networks. 2012. URL https://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf.

[2] David T. Felson Robert D. Jurmain Kimberly T. Wren Heli Maijanen Robert J. Woods Ian J. Wallace, Steven Worthington and Daniel E. Lieberman. Knee osteoarthritis has doubled in prevalence since the mid-20th century. 2017. URL https://www.pnas.org/content/114/35/9332.

[3] Understanding neural networks, August 2019. URL https://towardsdatascience.com/understanding-neural-networks-19020b758230.

[4] Jaspreet. URL https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec.

[5] Universal approximation theorem, July 2021. URL https://en.wikipedia.org/wiki/Universal_approximation_theorem.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. 1989. URL http://www.vision.jhu.edu/teaching/learning/deeplearning18/assets/Cybenko-89.pdf.

[7] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

[8] Allan Pinkus Shimon Schocken Moshe Leshno, Valdimir Ya. Lin. Multilayer feedforward networks with a non-polynomial activation function can approximate any function. 1992. URL https://archive.nyu.edu/jspui/bitstream/2451/14329/1/IS-92-13.pdf.

[9] Anastasis Kratsios. The universal approximation property. 2020. URL https://arxiv.org/pdf/1910.03344.pdf.

[10] Keiron Teilo O'Shea. An introduction to convolutional neural networks. 2015.

[11] Gentle dive into math behind convolutional neural networks, April 2019. URL https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

[12] URL https://en.wikipedia.org/wiki/Knee.

[13] . URL https://en.wikipedia.org/wiki/Meniscus_(anatomy).

[14] . URL https://en.wikipedia.org/wiki/Anatomical_terms_of_location.

[15] Garrett Hyman, . URL https://www.arthritis-health.com/types/osteoarthritis/what-knee-osteoarthritis.

[16] Garrett Hyman, . URL https://www.arthritis-health.com/types/osteoarthritis/knee-osteoarthritis-symptoms.

[17] URL https://www.arthritis-health.com.

[18] URL https://upload.wikimedia.org/wikipedia/commons/b/bc/Blausen_0597_KneeAnatomy_Side.png.

[19] Osteonecrosis of the knee - orthoinfo - aaos, . URL https://orthoinfo.aaos.org/en/diseases--conditions/osteonecrosis-of-the-knee.

[20] Aiyong Cui, Hui-Zi Li, Dawei Wang, Junlong Zhong, Yufeng Chen, and Hua ding Lu. Global, regional prevalence, incidence and risk factors of knee osteoarthritis in population-based studies. *EClinicalMedicine*, 29-30, 2020.

[21] Satya P. Singh et al. 3d deep learning on medical images: A review. 2020. URL https://arxiv.org/ftp/arxiv/papers/2004/2004.00218.pdf.

[22] Google Inc. Going deeper with convolutions. 2014.

[23] Google Inc. Rethinking the inception architecture for computer vision. 2015.

[24] Google Inc. Inception-v4, inception-resnet and the impact of residual connections on learning. 2016.

[25] Yassine Nasser et al. iscriminative regularized auto-encoder for early detection of knee os-teoarthritis: Data from the osteoarthritis initiative". 2020.

[26] Armine Guida et al. Knee osteoarthritis classification using 3d cnn and mr. 2021.

[27] Alexander Tack et al. Automated segmentation of knee bone and cartilagecombining statistical shape knowledge and convolutional neural networks data fromthe osteoarthritis initiative. 2018.

[28] Jo Schlempner et al. Attention gated networks: Learning to leverage salient regions in medical image. 2019.

[29] Mathhias Heinrich et al. Obelisk – one kernel to solvenearly everything: Unified 3d binary convolutions for image analysis. 2018.

[30] Oai, . URL https://nda.nih.gov/oai.

[31] whats-the-difference-between-the-sagittal-coronal-and-transverse-planes, . URL https://www.machinedesign.com/markets/medical/article/21834522/whats-the-difference-between-the-sagittal-coronal-and-transverse-planes.

[32] URL https://quizlet.com/406995436/body-planes-diagram/.

[33] . URL https://flexikon.doccheck.com/de/Score.

[34] Kellgren-lawrence-score, July 2016. URL https://flexikon.doccheck.com/de/Kellgren-Lawrence-Score.

[35] . URL https://en.wikipedia.org/wiki/X-ray.

[36] . URL http://www.healthofchildren.com/U-Z/X-Rays.html.

[37] URL https://www.livescience.com/39074-what-is-an-mri.html.

[38] Dandu Ravi Varma. Managing dicom images: Tips and tricks for the radiologist. 2012. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3354356/.

[39] . URL https://dicom-numpy.readthedocs.io/en/latest/.

[40] Feature scaling, July 2021. URL https://en.wikipedia.org/wiki/Feature_scaling.

[41] How to manually scale image pixel data for deep learning, July 2019. URL https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/.

[42] Samira Masoudi, Stephanie A. A. Harmon, Sherif Mehralivand, Stephanie M. Walker, Harish Raviprakash, Ulas Bagci, Peter L. Choyke, and Baris Turkbey. Quick guide on radiology image pre-processing for deep learning applications in prostate cancer research. *Journal of Medical Imaging*, 8(1):1 – 14, 2021. doi: 10.1117/1.JMI.8.1.010901. URL https://doi.org/10.1117/1.JMI.8.1.010901.

[43] What is a gpu and do you need one in deep learning?, December 2020. URL https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d.

[44] Build your own deep learning machine-what you need to know, June 2020. URL https://towardsdatascience.com/build-your-own-deep-learning-machine-what-you-need-to-know-85c7a0a1604a.

[45] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools, 2017.

[46] Ssh (secure shell), March 2021. URL https://en.wikipedia.org/wiki/SSH_(Secure_Shell).

[47] Amira for life biomedical sciences, . URL https://www.thermofisher.com/de/de/home/industrial/electron-microscopy/electron-microscopy-instruments-workflow-solutions/3d-visualization-analysis-software/amira-life-sciences-biomedical.html.

[48] Case studies and mentions : Tensorflow, . URL https://www.tensorflow.org/about/case-studies.

[49] How to choose loss functions when training deep learning, August 2020. URL https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

[50] The 5 classification evaluation metrics every data scientist, September 2020. URL https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226.

[51] Understanding binary cross-entropy / log loss: a visual, February 2019. URL https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[52] Hyperparameters in deep learning, May 2021. URL https://towardsdatascience.com/hyperparameters-in-deep-learning-927f7b2084dd.

[53] Overfitting underfitting concepts interview questions, December 2020. URL https://vitalflux.com/overfitting-underfitting-concepts-interview-questions/.

[54] Visualizing filters and feature maps in convolutional neural, August 2020. URL https://debuggercafe.com/visualizing-filters-and-feature-maps-in-convolutional-neural-networks-using-pytorch/.

[55] URL https://sebastianraschka.com/images/faq/relu-derivative/relu_3.png.

[56] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 9780262035613. URL https://books.google.co.in/books?id=Np9SDQAAQBAJ.

[57] A gentle introduction to the rectified linear unit (relu), August 2020. URL https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[58] Change the learning rate using schedules api in keras, June 2021. URL https://androidkt.com/change-the-learning-rate-using-schedules-api-in-keras/.

[59] Frank Hutter Abhinav Sharma1, Jan N. van Rijn and Andreas Mueller. Hyperparameter importance for image classification by residual neural networks. 2019. URL https://ada.liacs.leidenuniv.nl/papers/ShaEtAl19.pdf.

[60] A conceptual explanation of bayesian hyperparameter optimization for machine learning, July 2018. URL https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f.

[61] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel,

P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[62] Imagenet winning cnn architectures (ilsvrc): Data science and machine learning, . URL https://www.kaggle.com/getting-started/149448.

[63] A simple guide to the versions of the inception network, July 2020. URL https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202.

[64] Improving inception and image classification in tensorflow, August 2016. URL https://ai.googleblog.com/2016/08/improving-inception-and-image.html.

[65] Ct scan, July 2021. URL https://en.wikipedia.org/wiki/CT_scan.

[66] Ct scan, July 2021. URL https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2.

[67] Interpretable machine learning, June 2021. URL https://christophm.github.io/interpretable-ml-book/taxonomy-of-interpretability-methods.html.

[68] Saliency maps - keras-vis documentation, . URL https://raghakot.github.io/keras-vis/visualizations/saliency/.

[69] Saliency maps in tensorflow 2.0, . URL https://usmanr149.github.io/urmlblog/cnn/2020/05/01/Salincy-Maps.html.

[70] Class activation maps - keras-vis documentation, May 2020. URL https://raghakot.github.io/keras-vis/visualizations/class_activation_maps/.

[71] Grad-cam: Visual explanations from deep networks, May 2020. URL https://glassboxmedicine.com/2020/05/29/grad-cam-visual-explanations-from-deep-networks/.