Bachelor thesis

Johann Strunck

Deep learning-based grading of HR-pQCT motion artifacts

April 15, 2024

supervised by:
Prof. Dr.-Ing. Tobias Knopp
Paul Jürß

Hamburg University of Technology
Institute for Biomedical Imaging
Schwarzenbergstraße 95
21073 Hamburg

University Medical Center Hamburg-Eppendorf
Section for Biomedical Imaging
Martinistraße 52
20246 Hamburg

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Quellen und Hilfsmittel angefertigt zu haben.

Hamburg, den ??.??.2010

Contents

Abstract

In the research field of osteoporosis High-Resolution peripheral Quantitative Computed Tomography (HR-pQCT) scans are used to image bone micro architecture in vivo at peripheral skeletal sites. A common issue of HR-pQCT scans is the appearance of motion artifacts in resulting images. These artifacts can appear due to involuntary movements like twitches and spasms which occurs due to the scan lasting between 3-4 minutes which is a long time to stay still. Depending on the severeness of those artifacts in the resulting image, it might not be suitable for medical use and a rescan is necessary. The decision of the severity is made by a qualified person which gives the image a number from 1 to 5, where 1 equals no motion artifacts and 5 equals severe motion artifacts. The decision of severity is a biased process and the score varies depending on the reviewing person. Studies show that operators disagree in up to 30% of all cases where a rescan might or might not be necessary [1]. This findings match with our data, since our data also has a disagreement in around 30% of all cases. To support the decision of the operators there has been a approach by [1] to improve the confidence of the result by using Convolutional Neural Network(CNN). The binary network achieved a high performance with a F1-Score of 86.8±2.8%. In this paper we try to replicate the network, achieving a F1-Score around 73.6 ± 1.4%. The lower results might be explained by the fact that we use XtremeCT I instead of II scans. Usually XtremeCT II scans are less likely to be in the domain of type 3 and 4 which our networks performs worse on exactly those types of data(Type 1 accuracy: 94.4 ± 5.1% ,Type 2 accuracy: 97.9 ± 2.6%, Type 3 accuracy: 72.9 ± 6.6%, Type 4 accuracy: 70.7 ± 6.2%, Type 5 accuracy: 99.7 ± 0.6%) Afterwards, based on those findings the impact of training similar networks by adding and changing layers is evaluated. The results show that introducing skip connections and inception layers to the network might enhance its performance. One network achieved slightly better peak performances and also got more stable results by switching the last convolution layer by a skip layer with 2 convolution layers. This results in a F1-score of 75.3 ± 2.8%. One of the major issues the networks were facing was the lack of training data which led them to overfit.

1 Introduction

High-Resolution peripheral Quantitative Computed Tomography (HR-pQCT) is a specialized non-invasive imaging technique that provides detailed and accurate three-dimensional images of bone and tissue micro architecture at the peripheral skeletal sites. In this paper we focus on the radius and tibia. This advanced imaging technique offers several distinct advantages like that scans provide high resolution images that allow a thorough assessment of the scanned bone micro architecture. It offers precise measurement of Bone Mineral Density(BMD) and geometric parameters such as trabecular thickness and cortical thickness. HR-pQCT has applications in both clinical and research settings and can help make more informed decisions about patient management and treatment strategies. It can provide insight into fracture or the risk of its occurrence. HR-pQCT imaging requires the patient to remain in position during the scan to avoid motion artifacts. This can be challenging for certain patient populations, such as children or individuals with limited mobility. The XtremCT I (Scanco Media AG), a device that was used to generated the test and training data for this paper, is vulnerable to patient movement for which it takes about 3-4 minutes for a scan which makes it difficult for patients to hold the chosen extremity in place for such a long time. This is also an issue when the extremity is hold in place by a cast.

Depending on the severity of the motion artifact the scan must be repeated. To determine the severity of the scan, [2] introduced a scale from 1 (no visible motion artifacts) to 5 (significant horizontal streaks) to grade the severity of motion in scans. In clinical studies it is commonly implemented, that scans with a grading of 4 or 5 have to be repeated to get another scan with possibly mitigated motion artifacts. However, even with a standardized scoring system, motion scoring remains subjective. Operator agreement has shown to remain only moderate, even with intensive training. Studies have shown that operators disagree in up to 30% of all cases [1]. Due to this issue a objective and standardized method is desirable for the grading process. Papers like [1] have taken an approach to find a suitable method for the objective grading of motion artifacts, with partial success.

In the last few years a tremendous interest in machine learning emerged. Many applications have found a way in our modern society [3]. They can be found in applications like speech to text or the recommendations on e-commerce websites. With the emerging field of deep learning in computer vision machine learning got more attention. In 2012 the ILSVRC 2012 challenge was won by the Convolutional Neural Network (CNN) based Network AlexNet outperforming the runner up with 10.8 percentage points lower top-5 error score of 15.3% .

Since then the application of deep learning structures like CNNs have seen rapid growth in fields like medical imaging. Fields including retinopathy screening, skin lesion classification and lymph node metastasis detection already have research showing that deep learning has a great potential in those fields . This holds also true for the research field of radiology there has been studies in fields like lesion detection, classification and segmentation [4].

A frequently occurring issue in medical imaging is the lack of training data. In our case we had 500 labeled examples of tibia and radius. If we compare this to the amount of data used in training state of the art networks with datasets like MNIST, CIFAR or ImageNet ranging from many thousand labeled images to over a million examples it's a small fraction. The main reason is that the labeling task in medical imaging can just be performed by professionals and therefore the process is costly and just a few people can do it. The data that we use in this paper to compare the different approaches was provided by the osteology department of the Universitätsklinikum Hamburg-Eppendorf. All 500 provided scans where generated by the Scanco XtremCT I. To ensure the correctness of the labels, three doctors labeled the data separate from each other. This allowed the values of the labels to be generalized reducing the subjective influence of the single person. To get a stable network with a somewhat high accuracy we have to find a way to augment the data so that we don't run into problems like overfitting or poor generalization of the network. This paper will compare different augmentation techniques and their impact on the training of the network given by [1].

This paper uses CNNs to train a network for grading motion artifacts. Deep learning techniques, particularly CNNs, have revolutionized medical image analysis. CNNs are a subset of Artificial Neural Networks(ANNs), with each node detecting local features from the input vector, minimizing the parameters in a process called down-sampling, and the subsequent layers combining these features into a fully connected layer.

CNNs excel at tasks such as image segmentation, object detection, and classification. With their ability to automatically learn intricate patterns and features from complex visual data, enabling more accurate and efficient diagnostic processes. Even with the simplistic structure chosen in [1] the network is reaching a F1 Score of 86.8±2.8%, this can lead to the assumption that a more sophisticated network might reach better results. This paper builds on the findings and tries to create a stronger network with state of the art CNN building blocks.

This paper tries to replicate the findings of [1]. To achieve a high accuracy we evaluate which augmentation methods and learning rates are best suited for its training, afterwards modified network structures are compared and their results are interpreted.

2
Literature review

Computer Tomography(CT) is a three dimensional imaging technique which is commonly used in the field of radiology to obtain internal images of the body. This technique uses a emitter of x-rays which rotates around the specified body part, the rays are then captured by a receiver that measures the beams attenuation. There is a high contrast between the soft and mineralized tissue due to the electron-dense bone matrix. Afterwards the radon transform can be applied to the collected values from the scan to generate a high resolution three dimensional image of the internal body structure [5].

HR-pQCT is a dedicated extremity imaging system developed to image bone micro architecture in vivo at peripheral skeletal sites [6]. This imaging method gives information on the bone structure and mineral density in bones like radius and tibia. This information allows the estimation of the bone strength and ability to resist fracture. The extraction of this information is possible due to the high resolution of HR-pQCT.HR-pQCT is a low radiation dose method with a effective radiation dose at the distal radius and tibia of 3-5 µSV depending on the scanner generation. This is significantly less when comparing it to other common medical imaging techniques like chest X-rays with 100µSV or a hip CT scan with 2000-3000 µSV. A big field of study using this technique is the field of osteoporosis. Osteoporosis causes bones to become weak and brittle, so brittle that a fall or even mild stresses such as bending over or coughing can cause a fracture. Six individual studies demonstrated that HR-pQCT variables could predict incident fractures in postmenopausal woman and old men, suggesting that the assessment of cortical and trabecular bone micro architecture by HR-pQCT could improve overall fracture prediction. There are still no widespread use of HR-pQCT since there is just a small number of devices installed (fewer than 100 in mid 2020) therefore the main use of HR-pQCT is related to research [6].

In recent years machine learning has become a popular field in the research domain due to its versatile nature. It is a branch of artificial intelligence and computer science which focuses on the use of data and algorithms to imitate the way humans learn. A machine learning algorithm operates by processing data to identify patterns and connections. It learns from the data through a training process, adjusting its internal parameters to minimize a measure called "loss". The loss quantifies the difference between the algorithm's predictions and the actual outcomes in the training data. The algorithm iterative refines its parameters to reduce this loss, making its predictions more accurate. Once trained, the algorithm can apply its

learning to new, unseen data, aiming to make predictions or classifications with minimized error based on its learned patterns.

Loss functions in machine learning can be categorized base on the task which they are used for. Most of them can either be applied to classification or regression tasks. A typical function for regression tasks and one of the early loss functions in machine learning is the L_2 (also known as Mean Squared Error) loss. This function quantifies the magnitude of the error between the prediction and actual output of a network. The squaring results in a higher penalty to higher deviations from the target value.

$$L_2(y, \hat{y}) = \frac{1}{N} \cdot \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.1}$$

Most object detectors use cross entropy-based loss functions for the classification. It is motivated by an adaptive algorithm for estimating rare events in complex stochastic network. This function is the most basic loss function for the classification task

$$loss(y, \hat{y}) = -\frac{1}{N} \cdot \sum_{i=1}^{N} y_i \cdot log(\hat{y}_i) \tag{2.2}$$

where y is the ground truth and $\hat{y}$ is the classification outcome. There are many functions derived from the cross entropy loss like Kullback-Leibler divergence and Jensen Shannon divergence. [7] shows that in a classification task, that cross entropy loss is equivalent to the other models as long as the ground truth is known. Since cross entropy loss does not involve the real distribution entropy and gets more stable results, it will be used in this thesis.

A common problem in classification tasks is the imbalance of data. This can lead to the model being biased and performing poorly on smaller classes. To mitigate this effect weighted loss can be used. This function is a modification of standard loss which assigns higher weights to minor classes. Those weight's are calculated based on the relation between the total number of samples and the number of samples in the given class.

$$w_i = \frac{total\_samples}{number\_samples\_in\_class\_i \cdot num\_classes} \tag{2.3}$$

In the scenario of a multi class function the weights are then applied to the cross entropy loss. This function is known as weighted cross entropy loss.

$$loss(y, \hat{y}) = -\frac{1}{N} \cdot \sum_{i=1}^{N} w_i \cdot y_i \cdot log(\hat{y}_i) \tag{2.4}$$

The application of the weights to the loss function makes the model more sensitive to miss classification of underrepresented classes. This leads the network to tend in a more generalized direction.

Weighted loss can also be applied to a single class classification model. Since there is a binary cross entropy loss function.

$$loss(y, \hat{y}) = \frac{1}{N} \cdot \sum_{i=1}^{N} -w_i \cdot (y_i \cdot log(\sigma(\hat{y}_i)) + (1 - y_i) \cdot log(1 - \sigma(\hat{y}_i))) \tag{2.5}$$

The process of refining the data is done by a so called optimizer. There are various different optimization algorithms like stochastic gradient descent, RMSprop, Momentum or Adam. These algorithms differ in how they calculate and apply parameter updates based on the gradients of the loss function with respect to the model parameters (weight and biases) during training. The optimizer seeks to find the optimal set of parameters by iterative updating the parameters in a way that converges the model in a direction of decreasing loss.

Gradient descent is one of the most common ways to optimize a Neural Network.[8] It is a method based on the observation that if a function $F(x)$ is defined as differentiable in the neighborhood of a, $F(x)$ will decrease the fastest if one goes from a in the direction of the negative gradient of F at a. There are three different variants of gradient descent that are differing in how much data they use to calculate the gradient of the loss function. There is stochastic gradient descent, batch gradient descent and mini batch gradient descent.

Mini batch gradient descent is a compromise between stochastic gradient descent and batch gradient descent therefore it just takes a small amount of data for calculating the gradient. It is more accurate then stochastic gradient descent since stochastic gradient descent calculates its gradient based on one example, but it isn't as fast. Compared to batch gradient descent it is faster because batch gradient descent takes all the examples into account. Therefor batch gradient descent is more accurate. Mini batch gradient descent is a compromise of the other two variants and has a good balance between computation cost and accuracy it is the commonly used gradient descent algorithm. By calculating the gradient of the function, represented by the network to update its weights which fits the function to the data, minimizing the loss for this set of data. To ensure that the algorithm can find a minimum a learning rate η is implemented. This makes it that the function is just partly fitted to the used points therefor allowing the network to converge to a minimum. To ensure that the network can get closer to optimal weights the learning rate gets reduced after the accuracy does not change for a certain amount of training epochs. This makes it possible that the gradient descent method can come as close to a minimum of the loss function as possible. Even with the adjustment of the learning rate it is nearly impossible to reach the global minimum because the learning algorithm usually gets caught in a local minimum or saddle point.

Since the introduction of gradient descent there has been a lot of progress and new, more efficient algorithms were developed.

Adaptive Moment Estimation (Adam) [9] is a first-order gradient-based optimization technique or learning algorithm that is widely used, representing the latest trend in deep learning optimization. Adam is a deep learning strategy that was specifically designed for training deep neural networks. Its main selling points are it's memory efficiency and less computational

cost compared to other optimization algorithms. It utilizes the squared gradients(v_t) to scale the learning rate like RMSprop, which is sometimes refereed to as L2-regularization and is similar to momentum by using the moving average of the gradient(m_t).

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{2.6}$$
$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{2.7}$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2.8}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance). These moving averages are initialized as vectors of 0's leading to moment estimates that are biased towards zero. The initialization bias can be counteracted resulting in bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.9}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.10}$$

Those moment estimates can then be used to update the parameters:

$$\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t(\sqrt{\hat{v}_t} + \epsilon) \tag{2.11}$$

Good default settings are step size $\alpha = 0.001$, exponential decay rates for the moment estimates $\beta_1 = 0.9\, \beta_2 = 0.999$ and $\epsilon = 10^{-10}$.

Even though Adam has the ability to adapt its learning rate there are still cases where this is not sufficient. [10] state that the L2 regularization approach of Adam is not as efficient as weight decay and therefore introduce ADAMW. This is a modified version of Adam which decouples the weight decay from the gradient based update. The main difference to Adam lies in the formula for calculating the gradient and update parameter.

$$g_t = \nabla_\theta f_t(\theta_{t-1}) + \lambda \cdot \theta_{t-1} \tag{2.12}$$
$$\theta_t = \theta_{t-1} - \eta_t(\alpha \cdot \hat{m}_t(\sqrt{\hat{v}_t} + \epsilon) + \lambda\theta_{t-1}) \tag{2.13}$$

In this formulas η_t denotes the schedule multiplier which can be fixed or decay and $\lambda \in \mathbb{R}$. The test result of the paper show a 15% relative improvement in test error compared to Adam for various image recognition datasets.

Using Adam is usually not sufficient to achieve the best results in the absence of added gradient noise. In one of the experiments conducted by [11] multiple approaches were used to train a network on the MNIST data set. Every approach was compared with and without adding Gaussian noise to the test data. The results of this experiment showed that in any of the chosen cases the best test and average test accuracy increased with the addition of noise, besides one case where the average stayed the same.

To ease the training process the initial values of the networks parameters are typically normalized by initializing them with zero mean. By training on our data we would usually lose this normalization, which slows down training and amplifies changes as the network becomes deeper. To remove those effects and therefore enhance the training stability and convergence of deep neural networks, batch normalization can be employed. [12] By standardizing the inputs within each mini-batch during training, gradient related issues are mitigated and convergence is accelerated. Additionally, it acts as a form of regularization and curbs overfitting. With this method we are able to use higher learning rates and pay less attention to the initialization parameters. [8]

CNNs are a class of deep learning models specifically designed for processing structured grid-like data, such as images, by automatically learning hierarchical patterns and features. CNNs are widely used in computer vision tasks and have revolutionized the field of image recognition, object detection, and image generation. CNNs excel at tasks like image classification where the network assigns a label or category to an input image.

The fundamental building blocks of CNNs are convolution layers which perform convolution operations on input data. These layers apply a set of learnable filters (also called kernels) to the input images by sliding the filter over the data and computing element-wise multiplications and summations to produce feature maps which are shown in 2.1. This layers purpose is detecting different features like edges, textures and more complex patterns. The learned features become progressively more abstract as they pass through multiple convolution layers making it possible to classify complex structures. Two key hyper parameters that define the convolution operation are size and number of kernels. The former is typically 3 × 3, but sometimes 5 × 5 or 7 × 7. The application of a convolution layer on a matrix shrinks it in size. To get the desired output size there are two parameters, named padding and stride, which can be changed to modify the output size.

The first possibility is to add padding which adds a number of zero rows and columns to the input. This increases the resulting output size. During the convolution, the filter slides over the matrix from left to right and from top to bottom. Stride is the second changeable parameter. It defines the step size of the filter. It therefore defines how many elements the filter moves to the right or bottom per iteration.

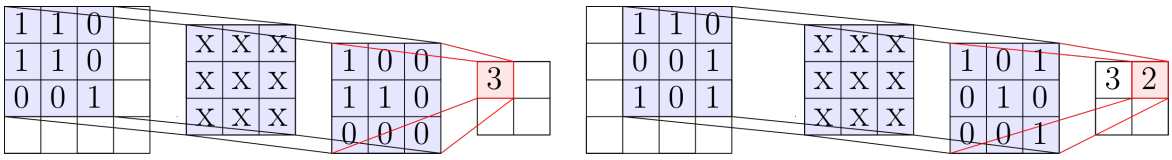

Figure 2.1: Convolutional Layer with Kernel 3x3 [1]

The convolution layer is usually followed by a pooling layer. This layers down sample the feature maps, reducing the spatial dimensions while retaining important information. Usually the stride matches the field size of the pooling operation so that no feature of the previous layer is used twice. Max pooling and average pooling are the most common applied pooling operations. In the max pooling operation the maximum value of the current view is selected.

This preserves detected features especially the most commonly used. The average pooling operation takes the averages of the values of the current view. During training in back propagation, average pooling provides a smoother gradient compared to max pooling. It also retains information from the original image since average pooling takes the collective information into account where max-pooling losses are at least 75% of the previous images data.

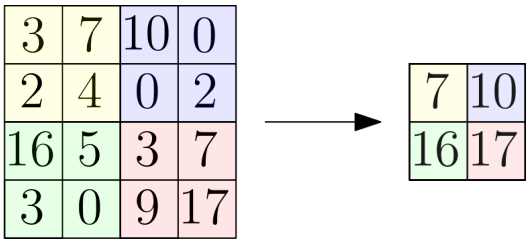

Figure 2.2: Max Pooling Layer with Kernel 2x2 and Stride 2

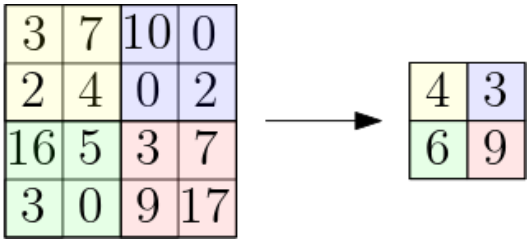

Figure 2.3: Average Pooling Layer with Kernel 2x2 and Stride 2

$$MaxPooling(X)_{i,j,k} = \max_{m,n} X_{i \cdot s_x + m, j \cdot s_y + n, k} \tag{2.14}$$

$$AveragePooling(X)_{i,j,k} = \frac{1}{f_x \cdot f_y} \sum_{m,n} X_{i \cdot s_x + m, j \cdot s_y + n, k} \tag{2.15}$$

After a few convolution and pooling layers the output gets flattened and feed into one or a set of Fully Connected (FC) layers (also known as dense layers). Each layer consists of interconnected nodes, also called neurons, where each output neuron is usually connected to every input. This feature enables the network to make high-level decisions based on the learned features. Each connection between nodes has an associated weight. To calculate the output of a FC layer the weighted sum for every single neuron is calculated. Most of the time a bias is added to the sum. Afterwards, an activation function is applied.

$$y(x) = acitvaion((\sum_{i=1}^{n} w_i \cdot x_i) + b) \tag{2.16}$$

As stated before, to calculate the output from the weighted sum of the inputs from a node, an activation function is needed. Those functions are used to map the input into the required number range like a value between 0 and 1 or 1 and -1. The choice of the activation function has a large impact on the capability and performance of the neural network. It can ensure a better detection of complicated patterns and even accelerate the learning process. [13]

[14] recommends to use ELU non-linearity without batch normalization or ReLU with it.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

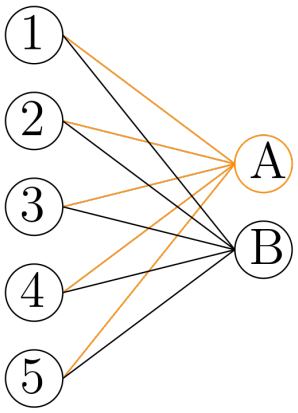

Figure 2.4: Standard FC layer

The ReLU function has significant advantages over a sigmoidal function in a neural network. The main advantage is that ReLU function is very fast to calculate. For positive x the ReLU function has a constant gradient of 1, whereas a sigmoid function has a gradient that rapidly converges to zero. This property makes neural networks with a sigmoid activation function slower to train. The occurring phenomenon is also known as vanishing gradient problem. ReLU as an activation function removes this issue because the gradient of ReLU is always one for positive x so that the learning process won't be slowed down by a vanishing gradient.

$$ReLU(x) = \max(x, 0) \tag{2.18}$$

However the zero gradient can pose the zero gradient problem. This can be compensated by adding a smaller linear term in x to give the ReLU function a nonzero slope at all points. This is solved in the implementation of ELU by adding $\alpha\ (e^x - 1)$ for all values smaller than zero. Therefore, the gradient of ELU is always bigger than 0, tending to zero for $x \to -\infty$.

$$ELU(x) = \begin{cases} x, \text{ if } x \geq 0 \\ \alpha\ (e^x - 1), \text{ otherwise} \end{cases} \tag{2.19}$$

When training a network with multiple classes where just one or a few should be classified, softmax can be used as a last layer activation function. The softmax function uses e^x to scale the values, the difference between singular values gets larger and a clearer differentiation is possible. A significant feature of the softmax function is that the sum over all the outputs equals 1. This is a useful feature in the context of image classifications since the outputs can directly be converted as percentage values. This can then be interpreted as the chance that the image shown symbolizes the class. Mathematically this means that the output O_j from a network is treated as a probability P.

$$O_j = P\ (c = j|input) \tag{2.20}$$

This requires:

$$O_j > 0 \quad and \quad \sum_{j=1}^{N} O_j = 1 \tag{2.21}$$

To enforce this stochastic constraint [15] suggests a normalized exponential output.

$$O_j = \frac{e^{I_j}}{\sum_k e^{I_k}} \tag{2.22}$$

Another last layer activation function is the sigmoid activation function which is commonly used for binary classification. The graph of the sigmoid function is a S shaped curve that is 0.5 for $\sigma(0)$. It converges to 1 for x $\rightarrow \infty$ and 0 for x $\rightarrow -\infty$. This ensures that the values range between 0 and 1.

The ImageNet Image Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition in the field of computer vision which focused on object detection and image classification. In the initial challenge held from 2010 to 2017 many breakthroughs were seen with the performance quickly rising. After 2017 the data set was updated and the challenge continues until now. It involves training and evaluating algorithms on the large ImageNet data set which contains millions of labeled images across thousands of categories. Many breakthroughs in deep learning particularly in CNN's can be traced back to this challenge.

In the history of CNNs AlexNet is a pioneering architecture that played a pivotal role in advancing the field of deep learning and computer vision. [16] In 2012 Alex net won the Image Net Large Scale Visual Recognition Challenge(ILSVRC) since it introduced several groundbreaking concepts, including deep architecture with multiple convolution and fully connected layers, ReLU activation functions, dropout for regularization, and GPU acceleration for faster training. Its success highlighted the potential of deep neural networks for image classification tasks. It influenced the design of subsequent CNN architectures and shaped the direction of modern deep learning research.

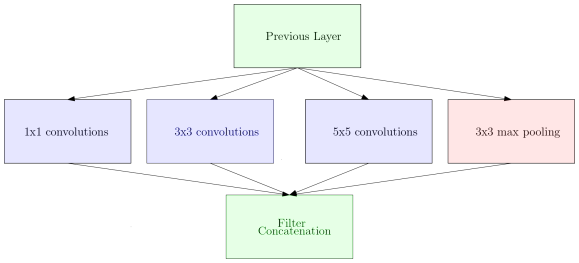

Figure 2.5: Naive Inception Module

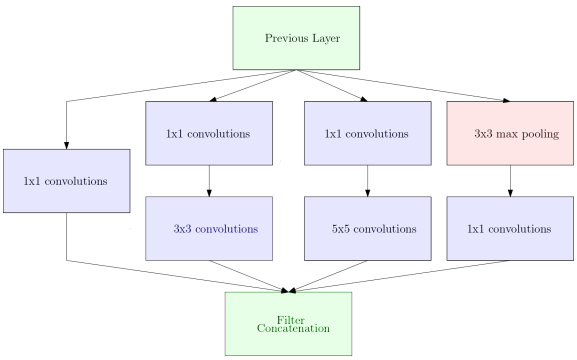

Figure 2.6: Inception Module With Reduction

GoogleNet (also known as Inception-V1) the winner of the 2014-ILSVRC competition introduced the inception block in its network achieving high accuracy with decreased computational cost. [17]

In this paper convolution layers are replaced with small blocks similar to the idea of substituting each layer with smaller networks which was proposed by network in network (NIN). It makes use of the idea of VVG that bigger filters of size 11x11 or 5x5 can be replaced by a stack of 3x3 filters. The use of smaller filters reduces computational complexity and induces the effect of large scale filter's. The inception block includes filters of size 1x1,3x3 and 5x5 to capture spatial information at different scales. Its split transform and merge concept addresses a problem related to the learning of diverse types of variations which are present in the same category of images.

Since the introduction of Inception-V1 progress has been made. The latest model Inception-V3/V4 was to minimize the computational cost with no effect on the deeper network generation by using asymmetric small-size filters (1x5 and 1x7) rather then large scale filters. Moreover they utilized the bottleneck of a 1x1 convolution layer prior to the large size filters.

In 2015,[18] won the ILSVRC competition by introducing Residual Networks(ResNet). These networks use a special component called the shortcut connection. This framework allows easier training of deeper networks. The shortcut connection allows the input data of a layer to be connected with the output of that layer and therefore input information can be kept. A great feature of this connection is not adding new parameters nor computational complexity to the network. As seen in 2.7 it is common to have more than one convolution layer between a shortcut connection. Usually, two to three convolution layers are used but more are possible. One convolution layer is uncommon since it's like using a linear layer which wouldn't benefit the network.

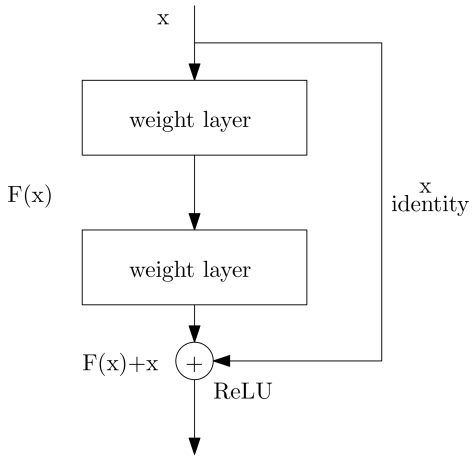

Figure 2.7: Convolutional layers with shortcut connection

In recent years the depth of network increased drastically. In fact, [19] stated that networks can be too deep. The features a convolution layer can process are limited by a parameter

which is called receptive field. It predicts which layers won't contribute qualitatively to the accuracy.

$$r_l = r_{l-1} + ((k_l - 1) \prod_{i=0}^{l-1} g_i) \tag{2.23}$$

The values needed to calculate the function are the receptive field size of the previous layer r_{l-1} which is 1 for r_0, g_i the stride of layer i and k_l the kernel size of layer l.

When the input resolution i is smaller than the previous receptive field size the so called border layer is reached.

$$r_{l-1} > i \tag{2.24}$$

This layer separates the so called unproductive layers from the productive layers. [19] states that with this layer it is possible to determine that every following layer will not contribute qualitatively to the test accuracy.

Commonly, biological data tends to be imbalanced. Often negative samples are more common, than positive ones. [16] When training a network with imbalanced data, the network is prone to bias towards the major classes, since it prioritizes learning the features for detecting those classes. This also means that the network does not get enough exposure to detect minor features and therefore can't learn its distinctive features. All this leads to a higher number of false negatives. With the network trying to capture the minor features it can run into the issue of overfitting the network. In our case there is a lack of data that is labeled with the severity type of 1 and 5. This holds a major issue, since type 5 scans are in need of a re-scan. Thus, a high accuracy score for detecting this class to be sure that healthcare professionals won't unnecessarily re-scan patients is desirable. A high accuracy score in class detection could thereby reduce the patient's exposure to unnecessary radiation.

In medical imaging the issue of imbalanced data is often combined with the lack of training data. When training a CNN with to little data we have to stop early and don't get an optimal accuracy for the network. Further training would lead the network to overfit and lose its validity. A common solution for this issue is the augmentation of the training data. Most augmentation techniques are based on basic image manipulations and generic translations. In this paper some of the techniques contained in [20] will be used.

[20] state that the most common method to augment data is the horizontal and vertical flipping of the images, where horizontal flipping is the most common implementation.

Another method is to crop the central patch of the image. This can be implemented as random cropping where a batch of variable size is cut from the main image and then scaled to the input size. This provides a quite similar effect as translation.

Translation shifts the image left, right, up and down. Because the data in our set is centered the positional bias of the network can be reduced by applying this method. Usually the remaining space would be filled up with zeros or random Gaussian noise. Since the given

dataset contains large images with the important information lying in the center a mix of cropping and translation is applied to stay in the bounds of the image so that adding padding to the image isn't required.

Rotation can also be implemented as an augmentation technique. The rotation can be chosen between 1 and 359°. This method needs to be handled with care. A common example are the numbers from the MNIST data set which would change their label when rotated to a certain degree. This method is also adapted by [21], examining cardiac magnetic resonance T1 mappings, which only adds a ± 5 ° uniform distributed rotation as their data augmentation methods.

Due to the lack of training data a possible method of enhancing the network is using transfer learning. This method is a common and effective strategy to use before training a network on a small data set. By pre-training the network on extremely large datasets like ImageNet with 1.4 million images and 1000 classes, trying to learn generic features that can be shared among networks. [4] This is a unique advantage of deep learning that makes itself useful in various domain tasks with small datasets. Despite the popularity of transfer learning in medical imaging there hasn't been a lot of work studying of its effects. Usually transfer learning is performed by taking a standard IMAGENET architecture with pre-trained weights and then fine tuning its parameters on the data set which the network is supposed to detect.

[22] shows on two large scale medical image networks that the gain of transfer learning on those networks is marginal. It also shows that transfer usually helps large scale models, with small models showing little difference. Therefore, transfer learning is not applied in this study.

When training a CNN with to little data we usually make use of the method of early stopping. This technique is used in machine learning to prevent overfitting during model training. Overfitting occurs in machine learning when a model learns to perform well on the training data, capturing noise and irrelevant patterns, but usually performs poor on new unseen data. This indicates that the model has memorized the training data instead of learning the underlying patterns, leading to reduced generalization ability and diminished predictive accuracy on real-world examples. Early stopping involves monitoring the model's performance on a validation data set and stopping training when the performance on a validation data set starts to degrade. By preventing excessive training; early stopping helps the model to generalize better to new data and to improve its ability to make accurate predictions on new data. This technique strikes a balance between training optimal performance and avoiding the point where the model starts memorizing noise in the training data.

When training networks a crucial step is to compare different network structures and their performances with each other to get the best results. One possibility is to introduce a measure that is capable to express how well a network would perform on new, unseen data. The most common way, which is employed in the majority of machine learning papers, is to calculate the accuracy of a network. This is done by splitting the data into two sets of arbitrary sizes. The first set inherits 80 percent of the data and is harnessed to train the network. The second set is then used to calculate the accuracy this is done by taking the proportion of correctly

classified samples to the whole sample space. This measure can be misleading especially in medical imaging due to the sometimes extreme uneven distribution of data.

Total Population = N+P	Predicted Positives (PP)	Predicted Negatives (PN)
Positives (P)	True Positives (TP)	False Negatives (FN)
Negatives (N)	False Positives (FP)	True Negatives (TN)

A categorical measure which is especially used in imbalanced classes is the F1 score.It is often implemented, since the measure can also be applied in a multi class environment. This measure is the harmonic mean of precision and recall:

$$Precision = \frac{TP}{TP + FP} \tag{2.25}$$

$$Recall = \frac{TP}{FP + FN} \tag{2.26}$$

$$F1 - Score = 2 \cdot \frac{precission \cdot recall}{precission + recall} \tag{2.27}$$

Hence multiplication is used in the numerator, the measure would tend to zero if one of the measures is close to zero. Therefore the balance between both scores is important to get a larger number. This means that the issue arising form the use of accuracy is eliminated, since a balanced detection is needed to get a higher score. Still categorical measures like the F1-score don't give a full insight into a networks performance.

Each data point is just counted as a correct or incorrect classification without considering the magnitude of the error because each data point counts either as a correct or incorrect without taking the magnitude of the error into account. [23]

A different measure which was initially introduced as a measure of agreement between observers of psychological behavior, is the degree of agreement between two or more people observing the same experiment. This measure is called Cohen's kappa. It's application in machine learning is quite easy to implement since there are at least two observers given by the labeled data and the results of the machine learning model.

		Observer 1		
		false	true	Total
Oberver 2	false	A	B	A+B
	true	C	D	C+D
	Total	A+C	B+D	A+B+C+D

The initial paper[24] introducing this measure suggests that for any problem in nominal scale agreement between two entity's, there are only two relevant quantity's. One being the total agreement p_0 which is equivalent to the before defined accuracy. The second measure p_c is the proportion of units for which the agreement is expected by chance. This is also supported by [25] who state that it is thought to be a more robust measure than simple percent agreement calculation.

$$p_c = \frac{(a+b)\cdot(a+c)+(c+d)\cdot(b+d)}{(a+b+c+d)^2} \tag{2.28}$$

κ is the proportion of agreement after chance agreement is removed from consideration.

$$\kappa = \frac{p_0 - p_c}{1 - p_c} \tag{2.29}$$

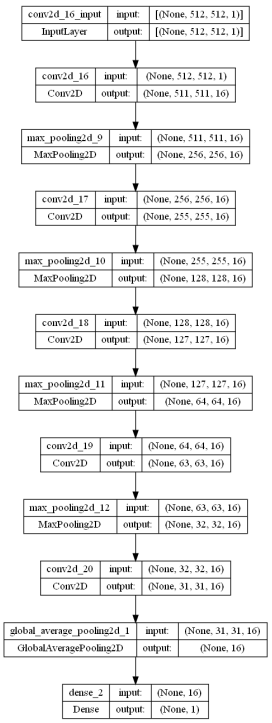

Figure 2.8: Network Structure Bone

[1] introduces a CNN structure which is trained to classify the severity of motion artifacts in HR-pQCT. The Network is trained with images (XtremeCT II, Scanco Media AG) from 90

patients. The size of scans is 512 x 512 x 168 pixels. For the training 8 equally spaced images from every scan are used resulting in a database of 3312 images. Images of the training set are randomly flipped horizontally and vertically. The implemented network structure begins with four alternating convolution and max-pooling layers followed by another convolution layer. The convolution layers employed leaky-ReLU as an activation function to enable faster learning while avoiding dead neurons. Afterwards, a global average pooling layer is implemented to reduce the output of each layer to size one which is necessary for the following dense layer. The classification is performed by a Fully Connected Layer integrating non-linear combinations of all high-level features using a standard Rectified Linear Unit (ReLU) activation function. The output layer used a softmax activation function which allowed the output to be understood as the classes probability. For training a weighted loss is implented to increase the accuracy for underrepresented classes. [1] further applied a feature visualization tool to determine the validity of the learned filters.

When training the network with an output of size one (evaluating if a re-scan was necessary), the networks reached similar agreement as the operators: F1-Score = 86.8±2.8%, precision = 87.5±2.7%, recall = 86.7±2.9% and Cohen's kappa = 68.6 ± 6.2%

3

Methods and experimental Setup

To develop the network and compare it, several steps are necessary. First of all, several measures are chosen to compare the networks structures with each other. Afterwards, different augmentation techniques are tested. To compare the performances we will look at the test and training set performance side by side. The side by side comparison can give valuable information on how the network learns, especially when identifying overfitting. Next the learning performance of the network needs to be evaluated. This includes the use of fixed and decaying learning rates. After these fundamental evaluations starts the testing of the effect different changes in the network structure have on its performance. The extracted information of those tests is then used to develop and test a final batch of networks.

Since we will compare our networks with the one from [1], it's significant to evaluate similar metrics. As the main indicator of convergence the loss metric is utilized. Going from that the focus is on the accuracy. F1-score and Cohen's kappa are calculated to get a better understanding on how the actual performance of the network looks like. Accuracy is a commonly used metric which can give a general understanding of the overall performance. Considering that the targeted issue is in medical imaging, accuracy is not sufficient, it doesn't inform about the distribution of true positives and true negatives. In the medical research field the Cohen's kappa coefficient is applied which is developed to measure the agreement between two or more raters. It's uniqueness is given by taking the agreement by chance into account. Still it can sometimes be difficult to interpret. Therefore, the F1-score is implemented as a third metric. This measure can be applied in situations where there is an imbalance between classes, which is true in the present case. It is calculated from the precision and the recall of the test set and can be applied as a convenient metric for comparing and communicating the overall performance of a classifier.

The weight initialization which is employed for all networks is the preset weight initialization technique used in the Flux.jl library. It is described in [26]. This method draws random numbers from a uniform distribution on the interval [-x,x] where:

$$x = gain \cdot \sqrt{\frac{6}{fan\_in + fan\_out}} \tag{3.1}$$

The gain factor is preset to one, fan_in is the number of input neurons connected to one output neuron and fan_out is the number of output neurons connected to one input neuron.

The initial approach of [1] set the output size to five since the scans had five severity grades. This approach only yielded a low accuracy. Considering that the most significant information is if a re-scan is necessary or not the output size was changed to one, which yielded significantly better results. Accordingly, in this thesis an output size of one is implemented for the scans.

As a loss function the weighted binary cross entropy loss is chosen. Since the input data is not equally distributed this is meant to help the network shift into a more generalized direction.

Motion Grade	Tibia rounded	Radius rounded	Sum
1	243	45	288
2	171	102	273
3	57	173	230
4	25	148	173
5	4	32	36
Sum	500	500	1000

The data utilized is provided by the osteology department of the Unfallklinikum Hamburg-Eppendorf and labeled by 3 different professionals. The labeled data contains 500 scans of the radius and 500 scans of the tibia. In 40.2% of all scans the doctors had an agreement, 41.4% for grading the tibia and 39% for grading the radius.

Considering the case that one doctor was for or against a re-scan contrary to the other 2 doctors it is observed, that this happens in 26 cases for the tibia and 109 cases for the radius. This makes it hard to compare in view of the significantly bigger amount of values in the domain of 3 and 4 for the radius compared to the tibia. By linking the amount of values where the rounded gathered rating was 3 or 4 to the possibility of a re-scan we get 31.7% for tibia and 33.9% for the radius.

For training the network the radius scans are selected as those are more equal distributed. Tibia is not considered because the use of both sets together lead to a worse accuracy.

The data is separated in a test and training set. The training set consists of 75% of the data and test set of the remaining 25%. The training data was then augmented. The test data is not augmented because in a real world example the augmented scans are unlikely to appear and would thereby perturbed the results.

For our training we used a "Nvidia Geforce RTX 2070 Super". It has 8 gigabytes of graphic memory. We decided to use a batch size of 16 images per iteration and a training set of 3200 images per Epoch.

Considering that the data set only consists of 500 radius scans the data set was enlarge by different augmentation methods. Meanwhile not every method is suitable for any type of data. Different augmentation methods are compared and will be evaluated in the results. The outcome shows that the optimal set of augmentation methods consists of:

- 90° rotation
- the use of cropped image snippets

- flipping of the image
- Gaussian noise

The implementation for Gaussian noise, 90° rotation and flipping of the images can be found in Julia libraries. The cropping of the image snippets is not as trivial. First, a minimal image size of 620 pixels is defined being the smallest frame where all the information is kept. The maximal size is limited by the input image size of 1536x1536 pixels. The decision on the frame size is made by a random normal distribution. Depending on the resulting size, the frame is randomly moved in horizontal or vertical direction, whilst always retaining the necessary information.

Commonly, Adam is used with a fixed learning rate of 0.001 which is sufficient in most cases. In the initial tests conducted with a learning rate of 0.001, the standard derivation of loss and accuracy is significant leading to the investigation on the learning rate. A strong oscillating loss or accuracy is usually the case when a network is trained with a learning rate that is too high. The learning rate determines how fast or slow the network will converge towards the optimal weights. A learning rate which is too large might lead the network to overshoot. This can either result in oscillation or in some extreme cases to a diverging network. In consequence different learning rates and methods are tested. The conducted tests used Adam and ADAMW with the possible addition of different weight decay parameters. Either a fixed value or a starting parameter which then applied an epoch wise decay function is chosen. These tests are conducted on the network of [1] and trained for 60 epochs to get a general understanding of the parameters performances.

Subsequently, the different building blocks or methods to be included in the network are tested as follows:

- Inception Block
- increasing the network depth
- addition of Skip Connections
- variation in the number of filters

Building a network from scratch can be a difficult task, hence the network from [1] was implemented as a start point. Due to the many possible combinations that could be changed, only small changes to the network are made before training it for 60 epochs. This way a general understanding on the performance of certain components could be obtained without needing to wait a long time for the training to complete.

The information gathered in this approach are used to build a set of final networks which are trained for 200 epochs and then compared with the findings of [1].

4 Results

The first milestone of this thesis was to replicate the results of [1]. The networks displayed in 2.8, which will later be refered to as the initial model, was chosen with the output size as one. Consequently, the softmax function is not applied as the last layer activation function but sigmoid. This output size is chosen because it is more valuable to detect if a re-scan is necessary than the exact severity level. Furthermore, the results of [1] show that the accuracy of the model significantly increases with the reduction of the output size.

In our first test different combinations of augmentation techniques are employed in combination and trained on the model of [1] for 200 epochs. Aiming to find the best augmentation method for the data. In general, the following augmentation techniques are applied to the training data:

- rotation by 90°,180° and 270°
- rotation by 1-359°
- flipping image horizontal or vertical
- taking image snippets and resizing them
- addition of Gaussian noise

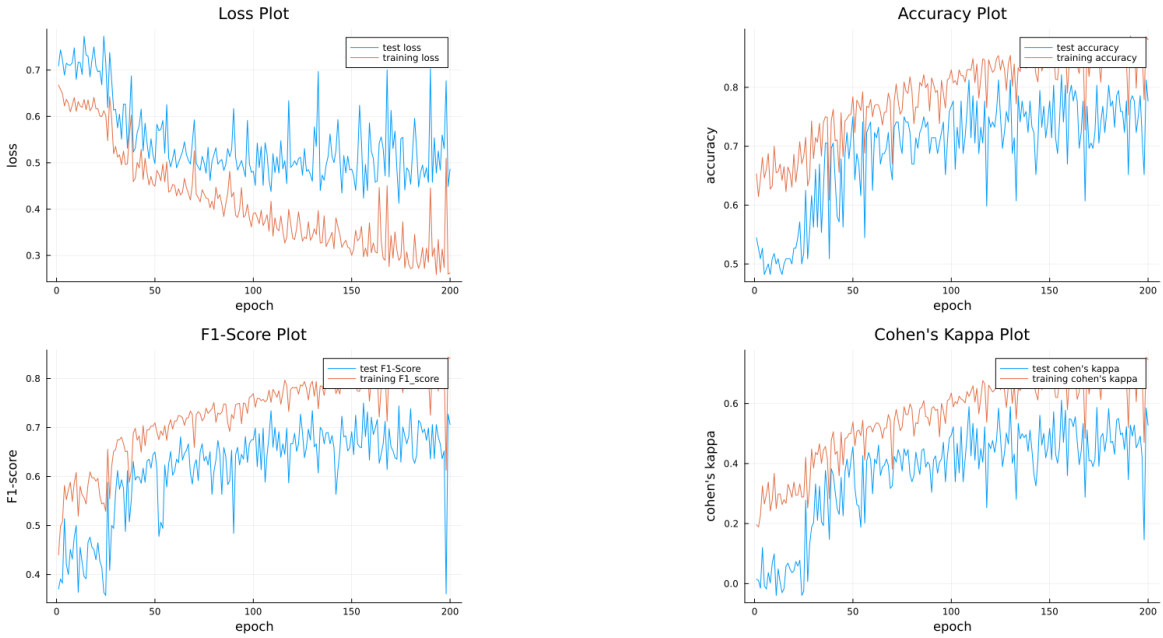

Figure 4.1: 90° rotation, flipping and Gaussian noise (epochs: 200 loss: 53.1 ± 8.12% accuracy: 74.9 ± 5.8% F1-score: 62.5 ± 13.4% Cohen's kappa: 44 ± 15.4)

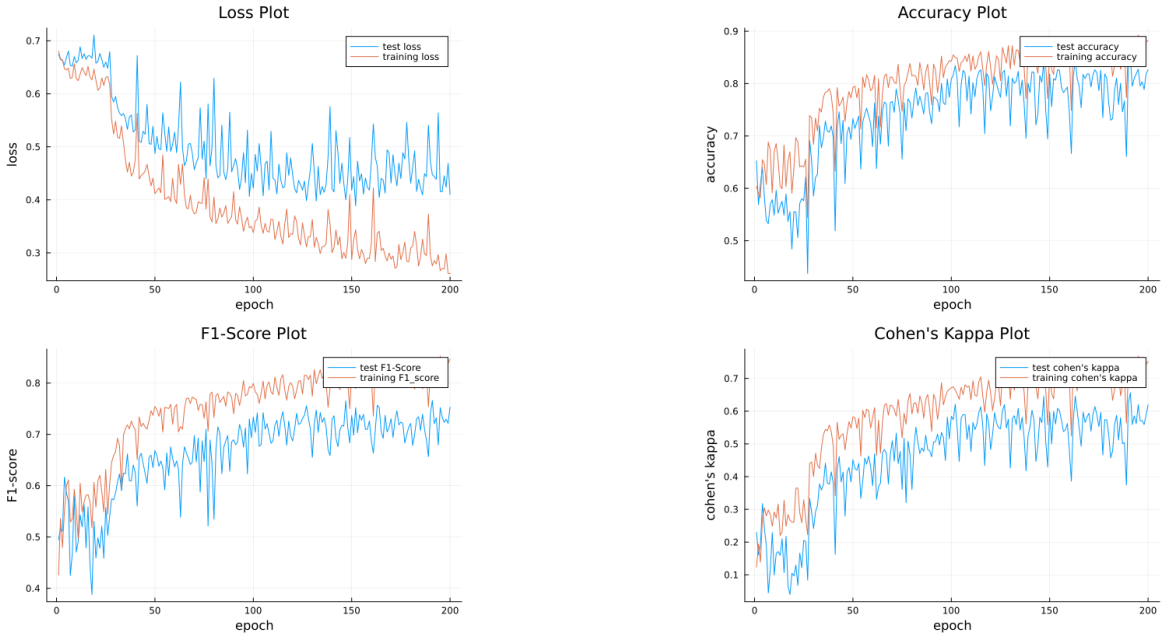

Figure 4.2: 90° rotation, resized image snippets, flipping and Gaussian noise(epochs: 200 loss: 43 ± 2.3% accuracy: 81 ± 1.6% F1-score: 73.6 ± 1.4% Cohen's kappa: 58.8 ± 2.6)

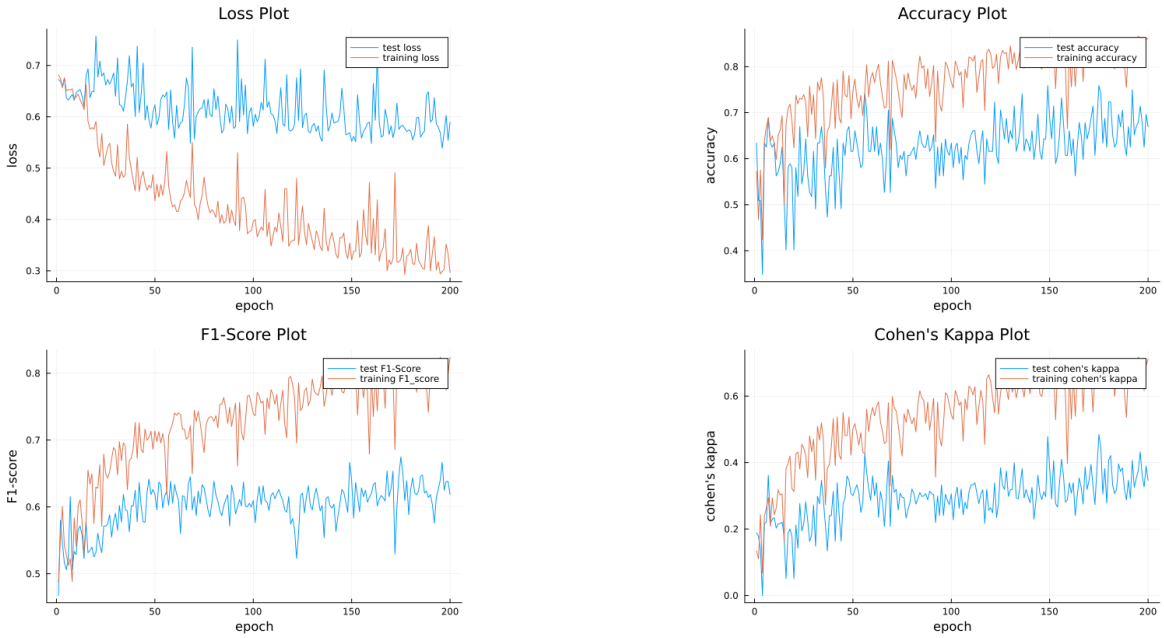

Figure 4.3: 1-360° rotation, resized image snippets,flipping and Gaussian noise(epochs: 200 loss: 57.1 ± 2.3% accuracy: 67.7 ± 3% F1-score: 63.8 ± 1.7% Cohen's kappa: 37.1 ± 3.7)

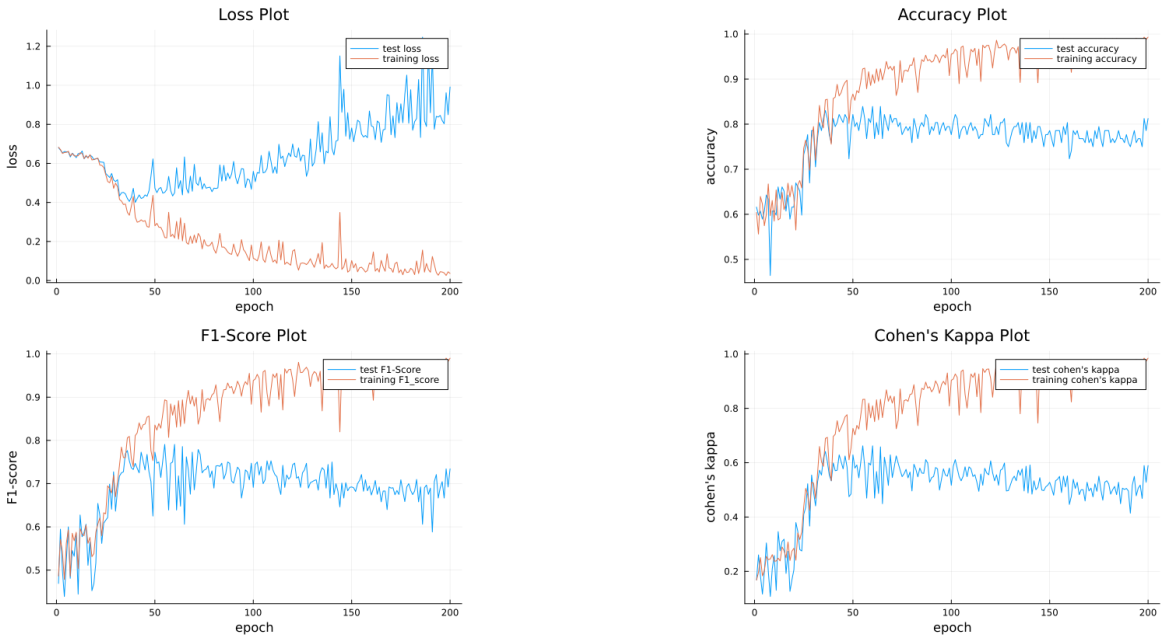

Figure 4.4: ±5° rotation, resized image snippets, flipping and Gaussian noise(epochs: 200 loss: 87.9 ± 7.8% accuracy: 78.3 ± 2.6% F1-score: 70.4 ± 2.7% Cohen's kappa: 53.3 ± 4.8)

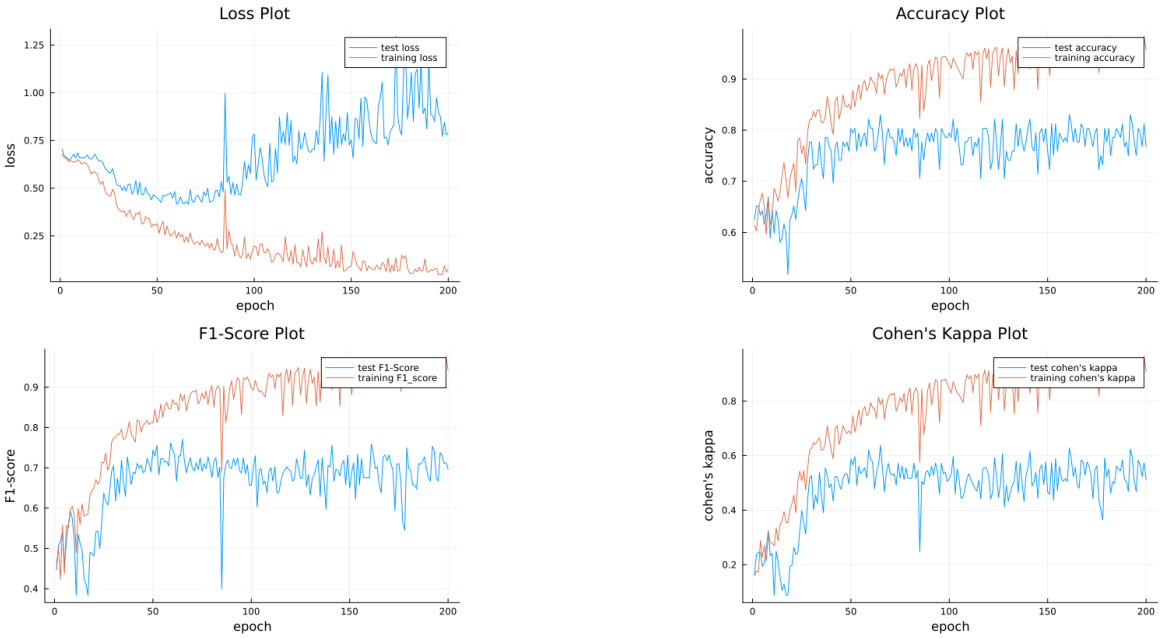

Figure 4.5: resized image snippets, flipping and Gaussian noise(epochs: 200 loss: 82.5 ± 5.3% accuracy: 78.3 ± 2.4% F1-score: 71 ± 2.5% Cohen's kappa: 53.8 ± 44.3)

The simultaneous use of rotation by 90°, 180° and 270°, flipping the image horizontal or vertical, adding Gaussian noise, taking image snippets and resizing them results in a stable training with the greatest accuracy and Cohen's kappa values. It is observed that the addition of 1-360° rotation is seemingly inefficient in training networks. When only using the ±5° rotation described in [21], the network reaches similar peak values in training but quickly starts to diverge after less than 50 epochs. It still shows little oscillation.

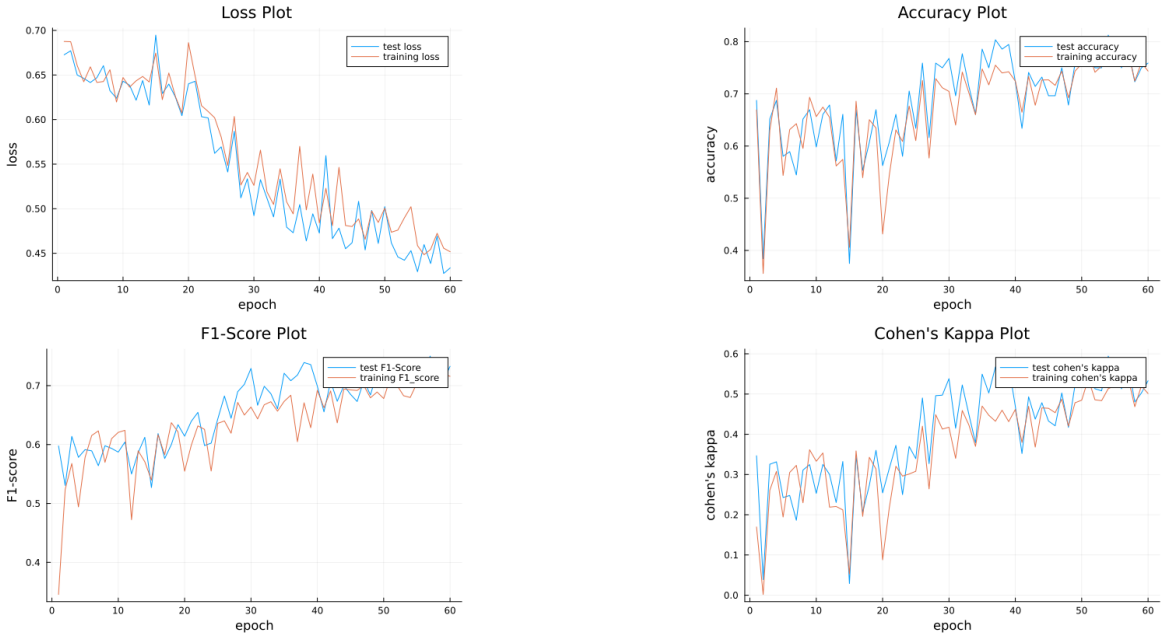

Figure 4.6: Adam without learning rate reduction(epochs: 60 loss: 44.3 ± 1.7% accuracy: 75.7 ± 2.2% F1-score: 72.6 ± 1.6% Cohen's kappa: 52.6 ± 3.4)

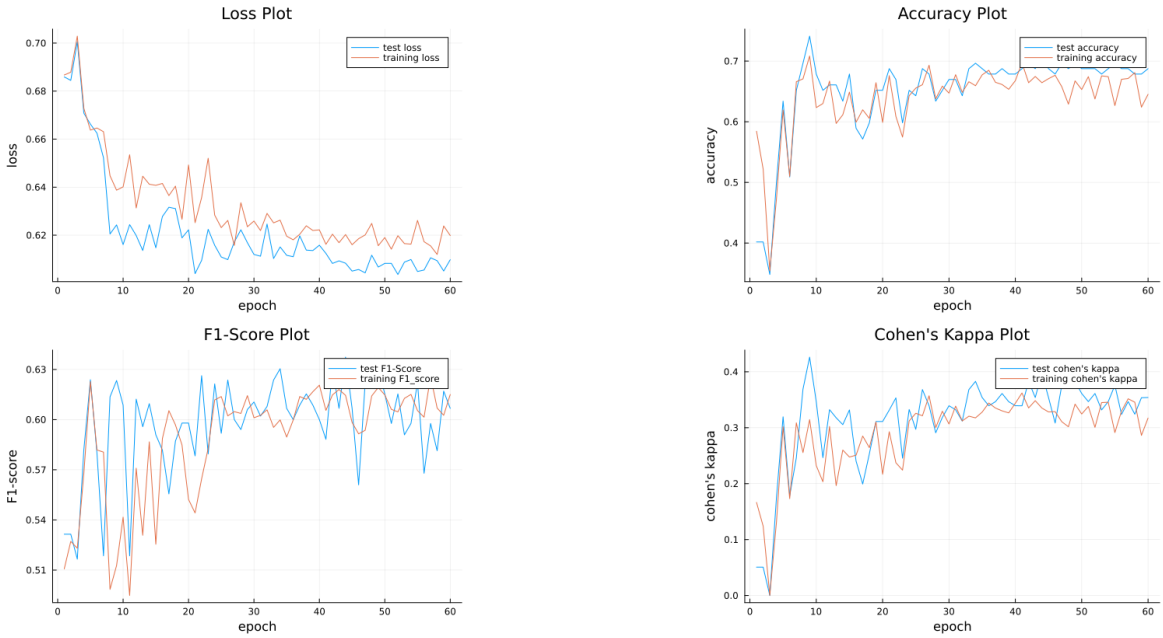

Figure 4.7: Adam with learning rate reduction of 0.94 per Epoch(epochs: 60 loss: 60.8 ± 0.3% accuracy: 68.6 ± 6.7% F1-score: 59.9 ± 2.1% Cohen's kappa: 34.6 ± 2)

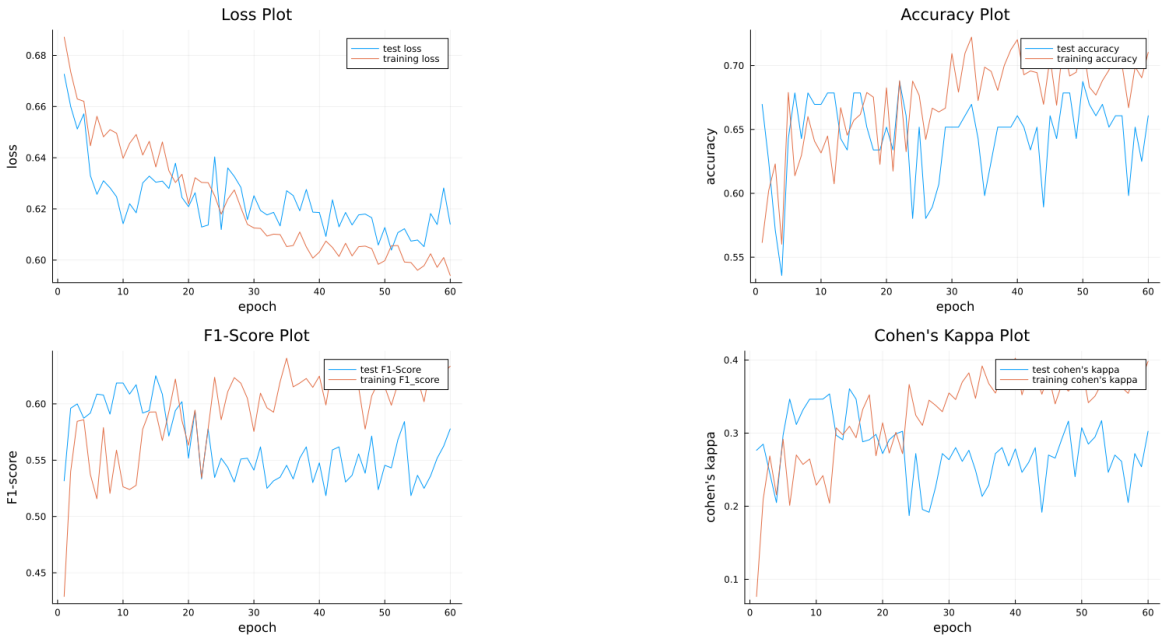

Figure 4.8: ADAMW with learning rate reduction of 0.94 per epoch(epochs: 60 loss: 61.5 ± 0.8% accuracy: 64.3 ± 2.6% F1-score: 54.8 ± 2% Cohen's kappa: 26.1 ± 3.2)

Due to the oscillation of the model the effect of adapting the learning rate over time is tested. When comparing the different learning models with each other it is clear that Adam, with a fixed learning rate, outperforms the two competing methods. After 60 epochs of training it has the lowest loss and the highest accuracy. It also seems to converge since the loss is decreasing in a somewhat linear manner. The other measures are also steadily increasing. This is different for Adam where a learning rate reduction of 0.94 per epoch is used equaling a learning rate decrease of 0.1 every 35 epochs. This method appears to quickly converge in the first 10 epochs and afterwards it slows down significant. If we look at the learning rate a major decrease in the first 10 epochs from 0.7 to 0.62 is observed. In the following 50 epochs it merely decreases by 0.02. This stagnation is also seen in the accuracy F1-score and Cohen's kappa without gaining a lot after the 10th epoch. A possible reason is that Adam's learning rate is not decoupled from the gradient based update method. Therefore, the performance of ADAMW with a learning rate decrease of 0.94 per epoch is analyzed. When looking at the loss of the network a somewhat steady decrease is detected which is not as fast as regular Adam. Although the learning rate is lower, the loss might decrease slower as well. By analyzing the other measures, it is apparent that after epoch 20 the results of the test set compared to the training set get worse. Thereby the results for the test set are decreasing while the ones for the training set increase slowly, with strong oscillation. Consequently, in the next tests Adam will be applied without a learning rate decrease. Still sometimes a learning rate of 0.0001 is implemented instead of the standard learning rate of 0.001. Due to the fact that bigger networks need a smaller learning rate because the network won't converge otherwise.

After the decision on the optimizer, it is tested which network structures performs well on the training data. In view of the many different setups to test, it is determined to train every network for 60 epochs. First of all, small changes are implemented into the network. Thereby the impact of adding and changing different layers on performance of the network can be detected. Here is a list of the building blocks:

- inception block
- shortcut connection
- additional convolution layers
- adding of padding
- changing pooling layers by strided convolution

model 1	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
Conv2D	(,30,30,16)
GAP	(,16)
Dense	(,1)

model 2	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Inception	(,128,128,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
GAP	(,16)
Dense	(,1)

model 3	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
Conv2D	(,30,30,16)
Conv2D	(,29,29,16)
GAP	(,16)
Dense	(,1)

model 4	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Inception	(,255,255,16)
MaxPool	(,128,128,16)
Inception	(,128,128,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
GAP	(,16)
Dense	(,1)

model 5	
Layer	Output
Input	(,512,512,1)
Skip1	(,512,512,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
GAP	(,16)
Dense	(,1)

model 6	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
Conv2D	(,31,31,16)
Conv2D	(,31,31,16)
GAP	(,16)
Dense	(,1)

Table 4.1: GAP: Global Average Pooling ,Skip1: Skip Connection with one 3x3 Convolution

model 7	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Inception	(,127,127,16)
MaxPool	(,64,64,16)
Inception	(,64,64,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,16)
GAP	(,16)
Dense	(,1)

model 8	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,32)
MaxPool	(,64,64,32)
Conv2D	(,63,63,32)
MaxPool	(,32,32,32)
Conv2D	(,32,32,32)
GAP	(,32)
Dense	(,1)

model 9	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Skip2	(,32,32,16)
Conv2D	(,32,32,16)
GAP	(,16)
Dense	(,1)

model 10	
Layer	Output
Input	(,512,512,1)
Conv2D	(,256,256,16)
Conv2D	(,128,128,16)
Conv2D	(,64,64,16)
Conv2D	(,32,32,16)
Conv2D	(,31,31,16)
GAP	(,16)
Dense	(,1)

model 11	
Layer	Output
Input	(,512,512,1)
Conv2D	(,511,511,16)
MaxPool	(,256,256,16)
Conv2D	(,255,255,16)
MaxPool	(,128,128,16)
Conv2D	(,127,127,16)
MaxPool	(,64,64,16)
Conv2D	(,63,63,16)
MaxPool	(,32,32,16)
Conv2D	(,31,31,32)
GAP	(,32)
Dense	(,16)
Dense	(,1)

Table 4.2: Skip2: A skip connection with two 3x3 Convolutions

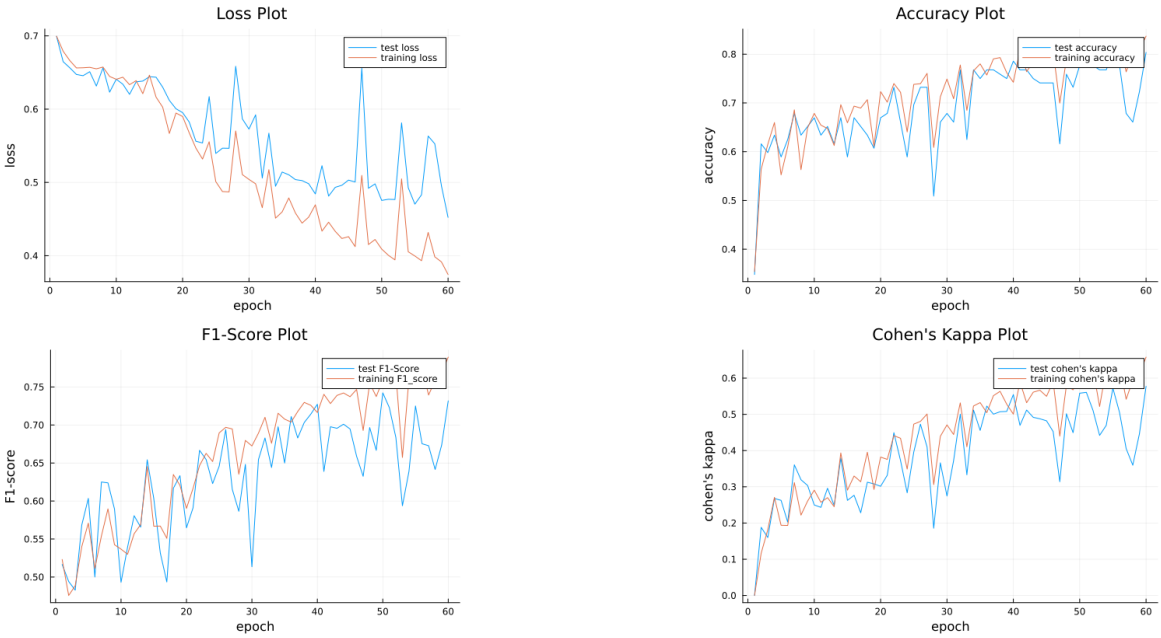

Figure 4.9: model 1 (epochs: 60 loss: 50.3 ± 4.5% accuracy: 74.1 ± 6.3% F1-score: 68.6 ± 3.5% Cohen's kappa: 47.8 ± 8.9)

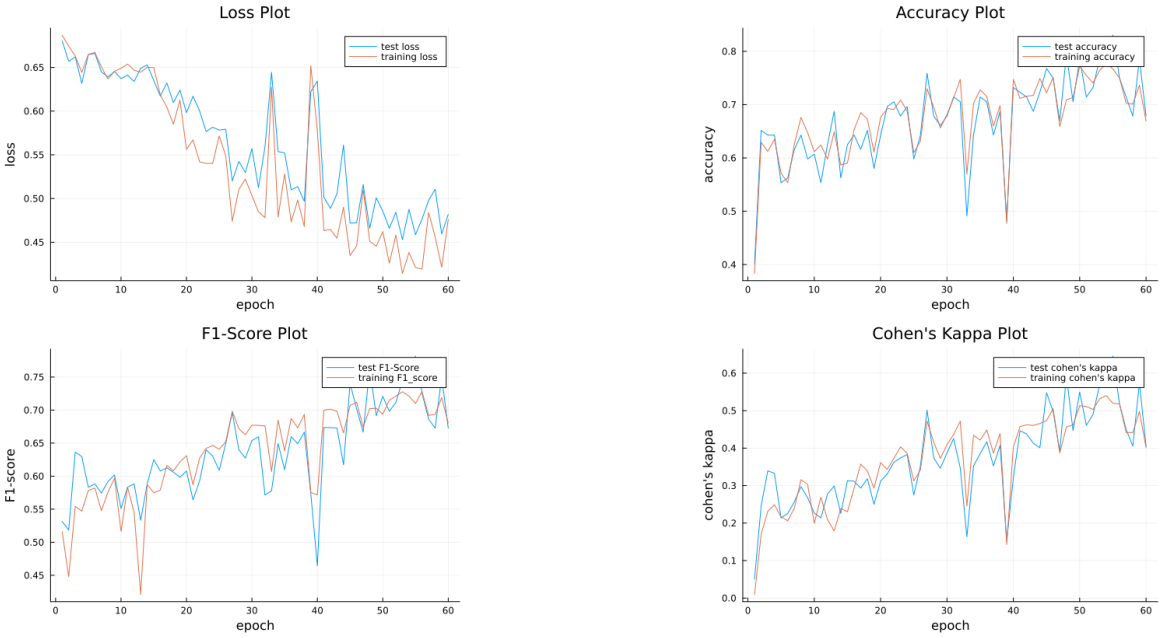

Figure 4.10: model 2 (epochs: 60 loss: 48.1 ± 2.1% accuracy: 74 ± 6.1% F1-score: 71.5 ± 4.5% Cohen's kappa: 50 ± 9.7)

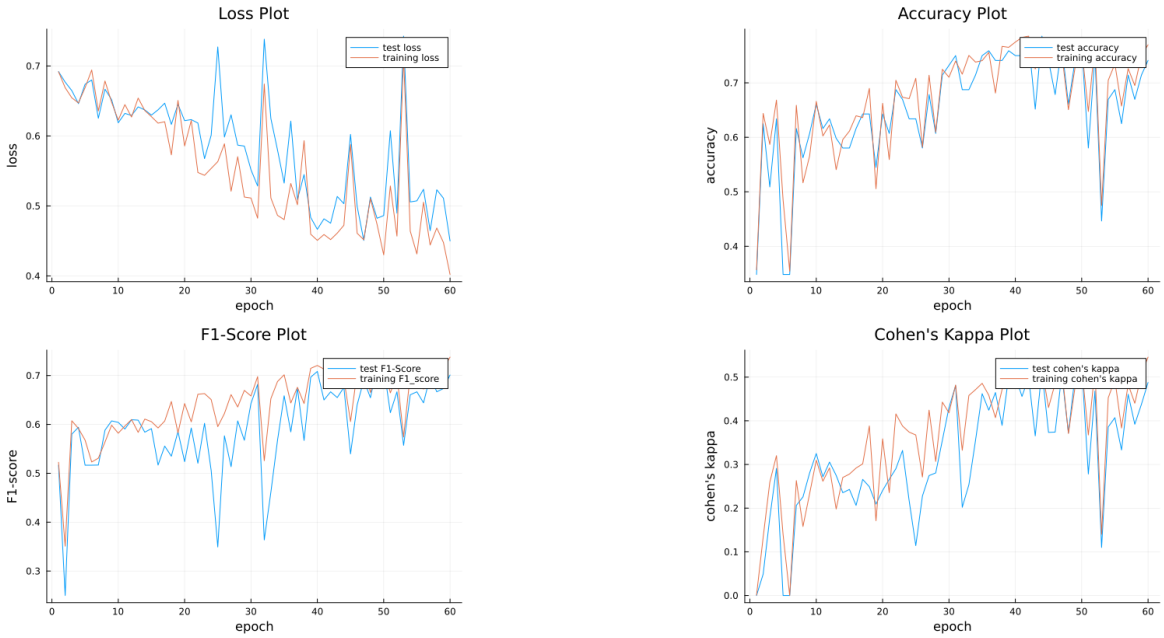

Figure 4.11: model 3(epochs:60 loss: 49.7 ± 3.1% accuracy: 69.2 ± 4.1% F1-score: 67.5 ± 2.1% Cohen's kappa: 42 ± 5.5)

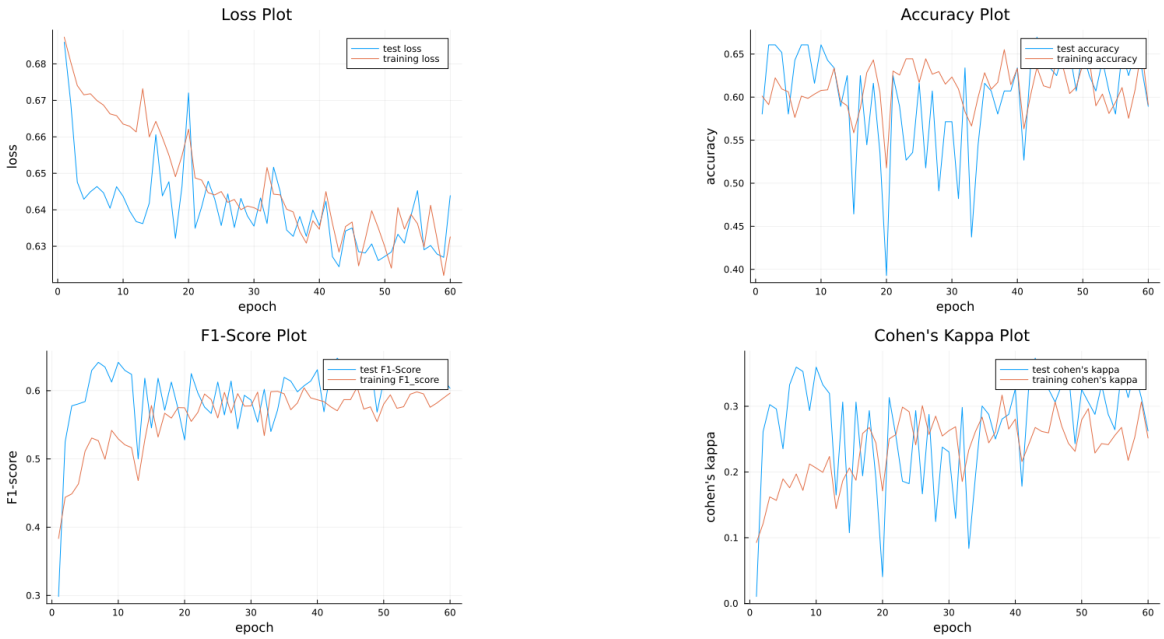

Figure 4.12: model 4(epochs: 60 loss: 63.4 ± 0.8% accuracy: 62.2 ± 3.1% F1-score: 62.4 ± 1.6% Cohen's kappa: 31 ± 4)

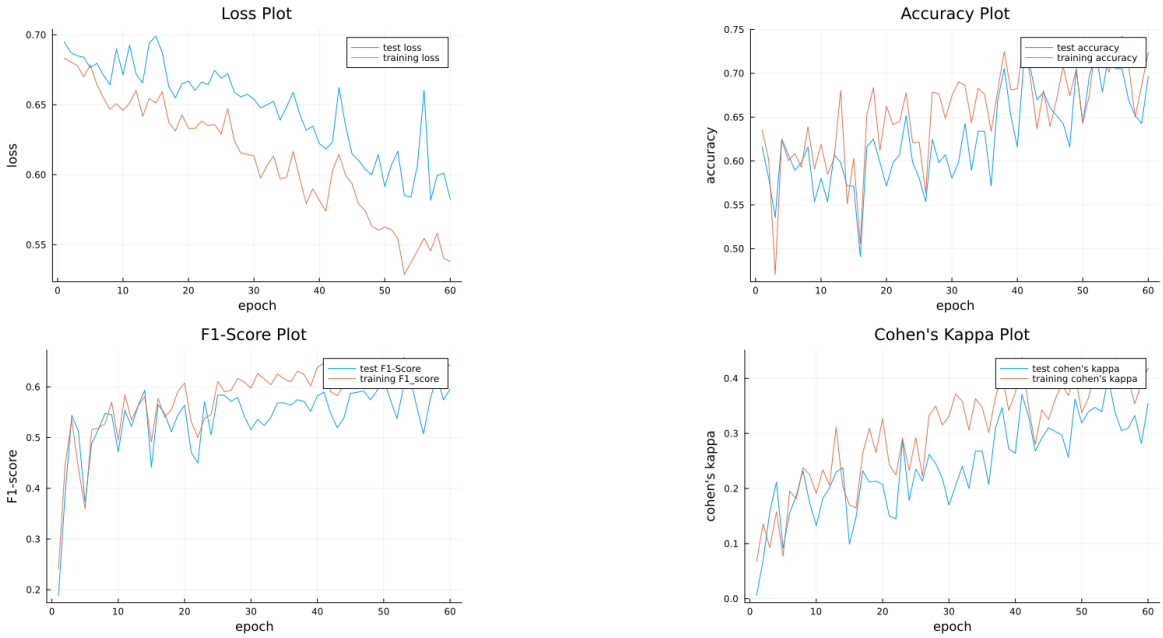

Figure 4.13: model 5(epochs: 60 loss: 60.5 ± 2.9% accuracy: 67.8 ± 2.8% F1-score: 57.2 ± 3.8% Cohen's kappa: 32 ± 2.6)

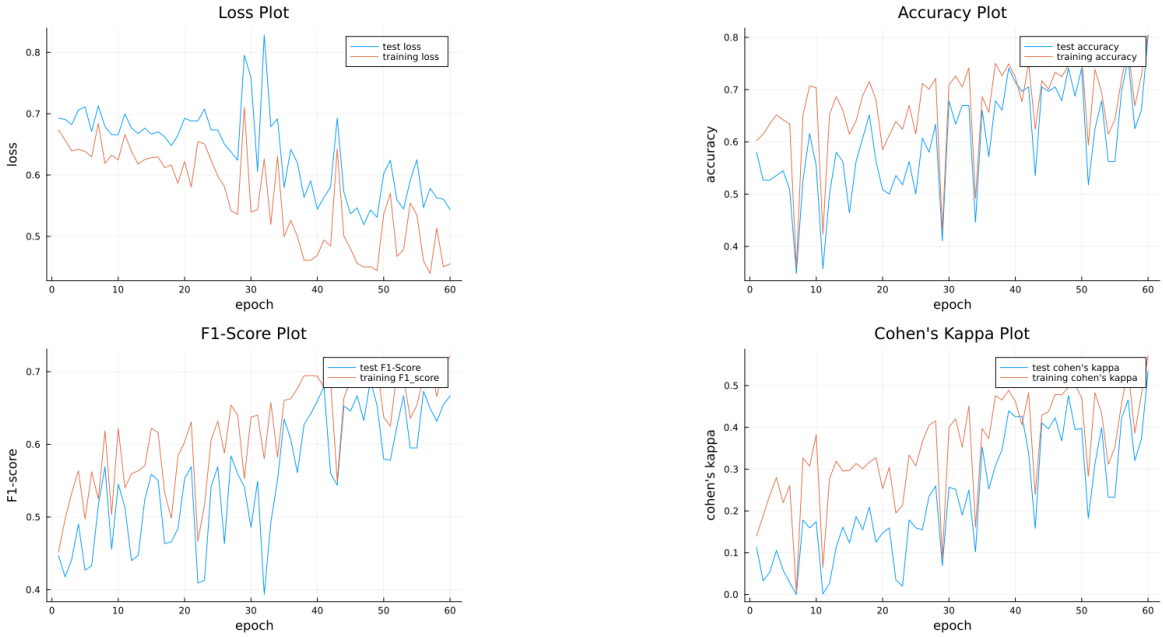

Figure 4.14: model 6(epochs: 60 loss: 57 ± 3% accuracy: 68.5 ± 8.8% F1-score: 64.5 ± 2.8% Cohen's kappa: 39.1 ± 10.7)

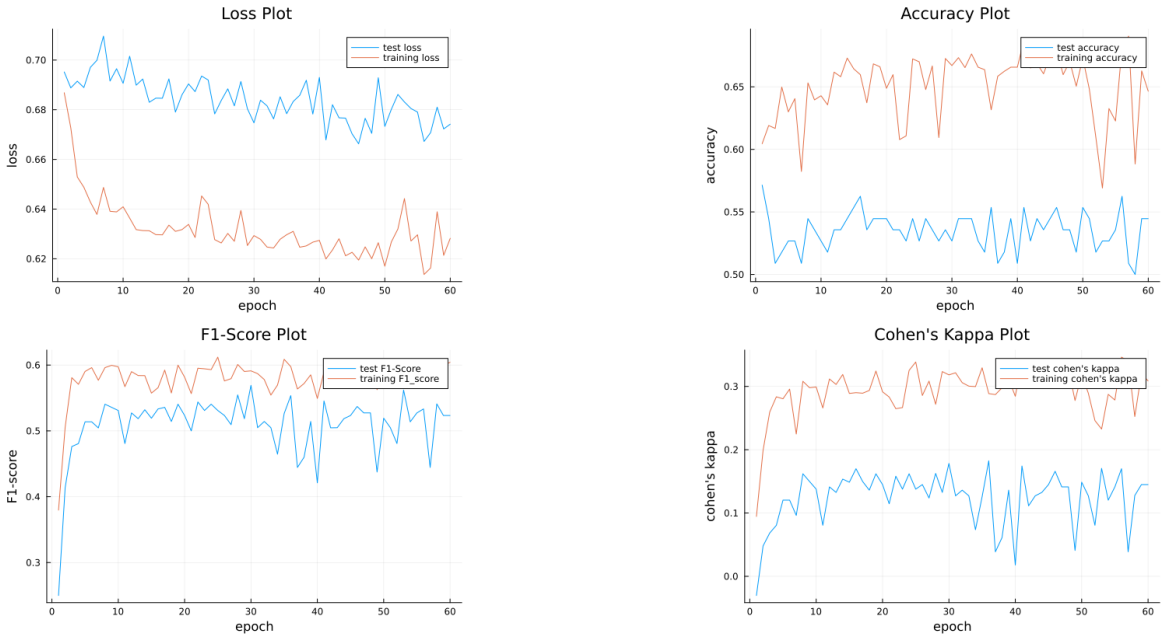

Figure 4.15: model 7 (epochs: 60 loss: 67.4 ± 0.5% accuracy: 53.3 ± 2.4% F1-score: 51.5 ± 3.5% Cohen's kappa: 12.8 ± 4.6)

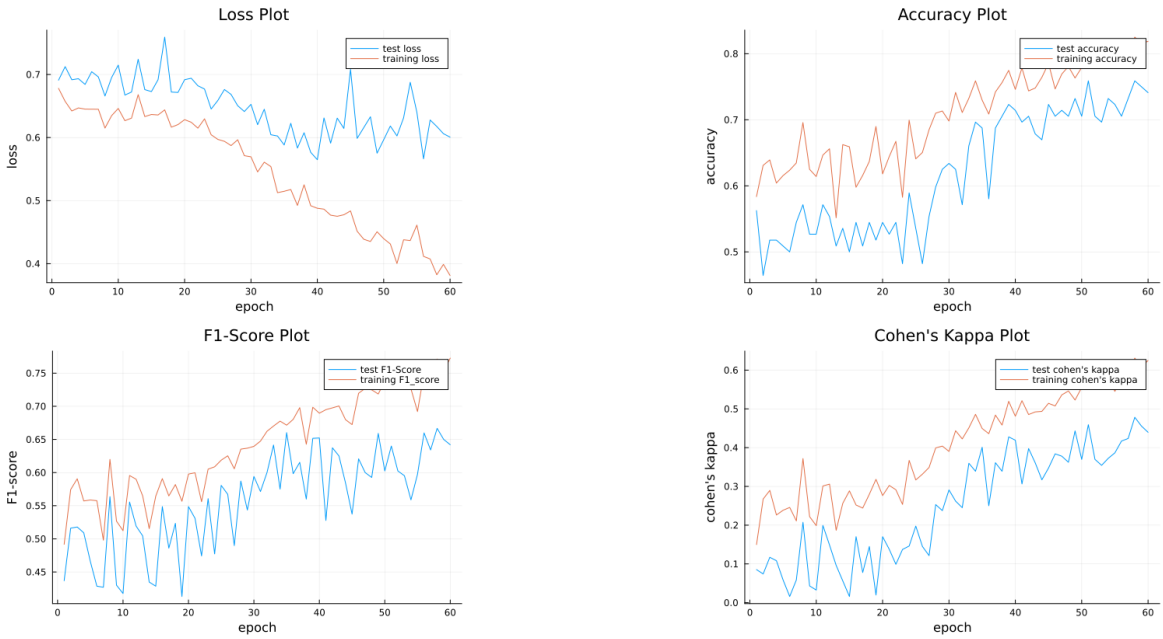

Figure 4.16: model 8 (epochs: 60 loss: 61 ± 7.6% accuracy: 73.5 ± 1.9% F1-score: 64.2 ± 2.5% Cohen's kappa: 43.3 ± 3.2)

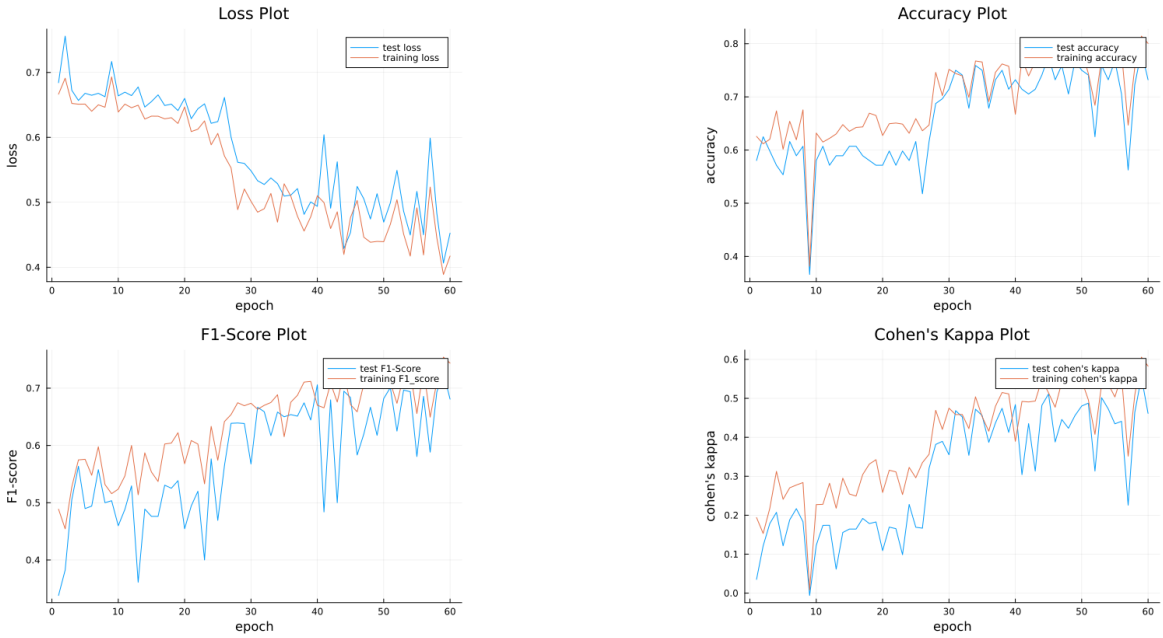

Figure 4.17: model 9 (epochs: 60 loss: 48.5 ± 6.7% accuracy: 71.3 ± 7.9% F1-score: 66 ± 6% Cohen's kappa: 43 ± 10.9)

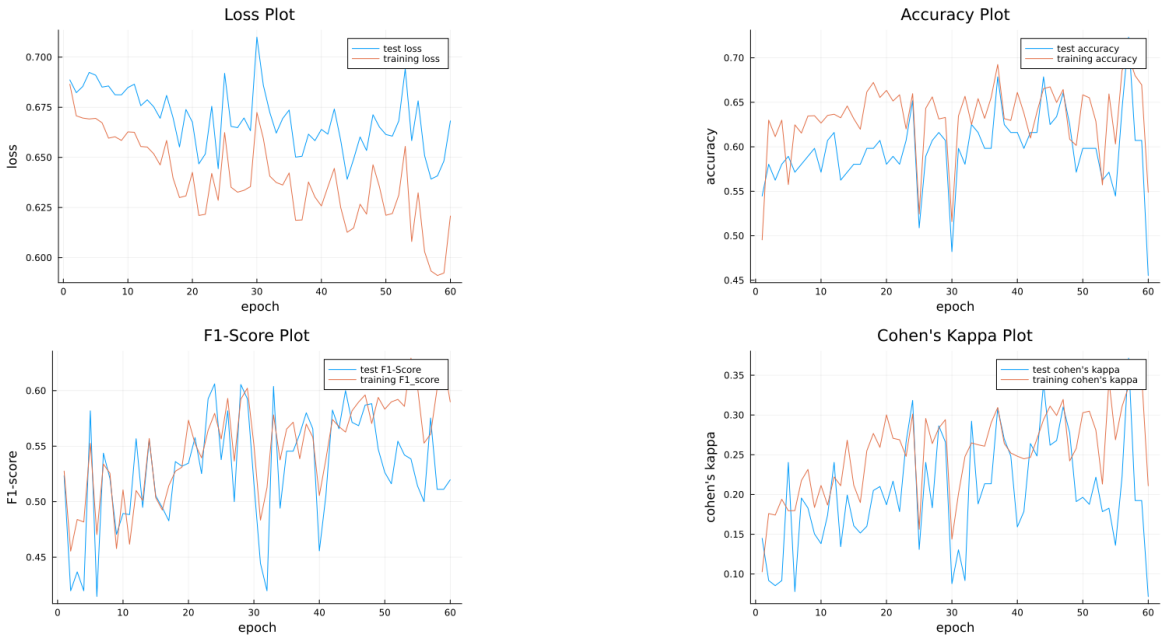

Figure 4.18: model 10 (epochs: 60 loss: 65.4 ± 1.6% accuracy: 59.7 ± 9% F1-score: 52.2 ± 2.7% Cohen's kappa: 19.8 ± 10.1)

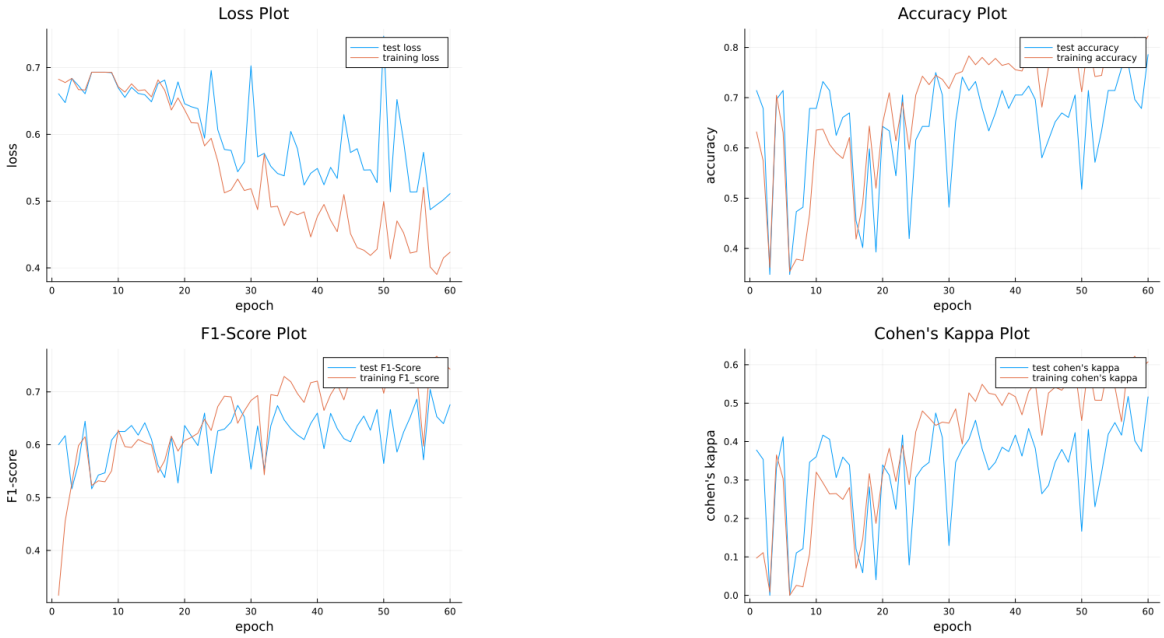

Figure 4.19: model 11 (epochs: 60 loss: 51.3 ± 3.1% accuracy: 73.4 ± 4.3% F1-score: 65.5 ± 4.7% Cohen's kappa: 44.6 ± 6)

One of the tests includes to observe the impact of adding convolution layers to the end of the network. In model 1 (4.9) one additional layer is added. Its results are very similar but the training loss starts decreasing faster than the test loss. A similar phenomenon can be seen in the F1-Score and Cohen's kappa plot where the training plot slowly diverges from the test plot. In model 3 (4.11) with two additional convolution layers this effect does not appear but the accuracy is nearly 10 % lower. The plots include some extreme outliers. In model 6 (4.14) padding was added to the layers. This increases the maximum accuracy to a similar level as without the additional convolutions but the amplitude of the oscillation increases as well.

Thereby it is concluded that adding new layers can quickly lead to a decrease in the networks performance. In contrast the addition of padding can reduce those effects.

Another test is switching convolution layers by a custom inception module. Model 2 (4.10) switches the third convolution with an inception module. The performance of it is quite similar to our initial model. The loss plot decreases linear. The accuracy, F1-score and Cohen's kappa increase linear but the amplitude of the oscillation is a bit stronger. The network still reaches accuracy values above 80 %. In model 4 (4.12) and 7 (4.15) two convolution layers are switched with inception modules. The networks accuracy, F1-Score and Cohen's kappa stay constant while oscillating. The test loss of both models is also constant while the training loss slightly decreases. In model 7 (4.15) the network's performance of the test set is significantly worse than the training set.

These tests show that adding too many inception modules can be harmful to the performance of the network while adding one inception module might increase the network's performance in the long run.

In model 5 (4.13) and 9(4.17) skip connections are attached to the network. The loss of model 5 (4.13) decreases linear and the accuracy and Cohen's kappa increase linear but the F1-score increases slightly. This model does not perform as well as the initial network but still seems to be growing. Model 9 (4.17) performs better than the initial network. Its accuracy, F1-score and Cohen's kappa are similar but the loss is 0.05 lower than the loss of the initial network. This means that the addition of skip connections can be beneficial to the overall performance of the network.

In model 8 (4.16) the number of filters of some convolution layers is increased. The training graphs look similar to the ones of the initial networks. The loss of the test set is similar to the training loss until epoch 35. Afterwards it stops decreasing, the gradient of accuracy, F1-Score and Cohen's kappa also decreases significantly. This model shows that even a slight increase of the network's parameters can have a major impact.

Model 10 (4.18) removes the max pooling layers and instead uses stride to quickly reduce the size of the input image. This was preformed to get a understanding of the importance the max pooling layer type. Looking at the results it can be observed that the accuracy plot oscillates strongly and barely reaches a value above 80%. Furthermore, the loss plot shows strong oscillation and the training and the test plot seem to diverge.

model 12		model 13		model 14	
Layer	Output	Layer	Output	Layer	Output
Input	(,512,512,1)	Input	(,512,512,1)	Input	(,512,512,1)
Conv2D	(,511,511,16)	Conv2D	(,511,511,16)	Conv2D	(,511,511,16)
MaxPool	(,256,256,16)	MaxPool	(,256,256,16)	MaxPool	(,256,256,16)
Conv2D	(,255,255,16)	Conv2D	(,255,255,16)	Conv2D	(,255,255,16)
MaxPool	(,128,128,16)	MaxPool	(,128,128,16)	MaxPool	(,128,128,16)
Inception	(,128,128,16)	Conv2D	(,127,127,16)	Inception	(,128,128,16)
MaxPool	(,64,64,16)	MaxPool	(,64,64,16)	MaxPool	(,64,64,16)
Conv2D	(,63,63,16)	Conv2D	(,63,63,16)	Conv2D	(,63,63,16)
MaxPool	(,32,32,16)	MaxPool	(,32,32,16)	MaxPool	(,32,32,16)
Conv2D	(,32,32,16)	Skip2	(,32,32,16)	Skip2	(,32,32,16)
Conv2D	(,32,32,16)	GAP	(,16)	GAP	(,16)
Conv2D	(,32,32,16)	Dense	(,1)	Dense	(,1)
GAP	(,16)				
Dense	(,1)				

Table 4.3: Skip2: A skip connection with two 3x3 Convolutions

In model 11 (4.19) the effect of multiple dense layers on the networks performance is evaluated. The loss plots are oscillating and seem to diverge. Even though a lower accuracy is reached, it does not reach the 80 % mark. The F1-score and Cohen's kappa graphs reach high values while oscillating significantly and showing signs of divergence.

In conclusion it can be stated that the addition of parameters to the network needs to be handled with care since even the addition of a few layers / kernels can lead to overfitting. In some cases, it occurs that the network does not converge even when reducing the learning rate. After these test a combination of certain methods is applied and the effect on the network's performance is evaluated.

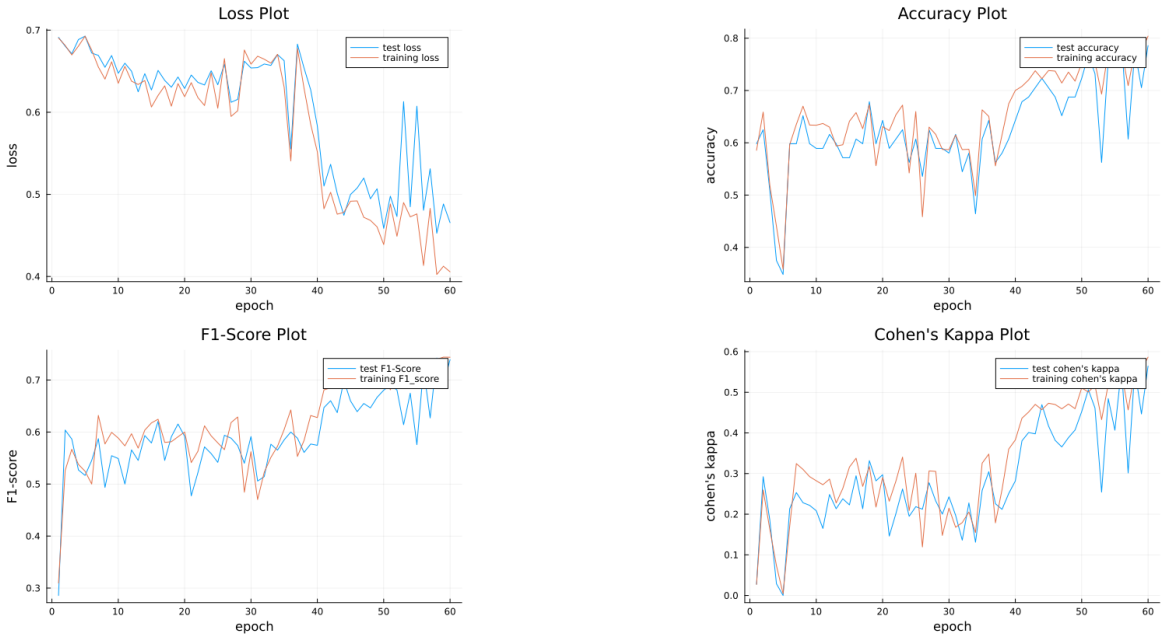

Figure 4.20: model 12 (epochs: 60 loss: ±50.4%5.7 accuracy: 73.7 ± 7.1% F1-score: 68.3 ± 6.8% Cohen's kappa: 47.3 ± 10.7)

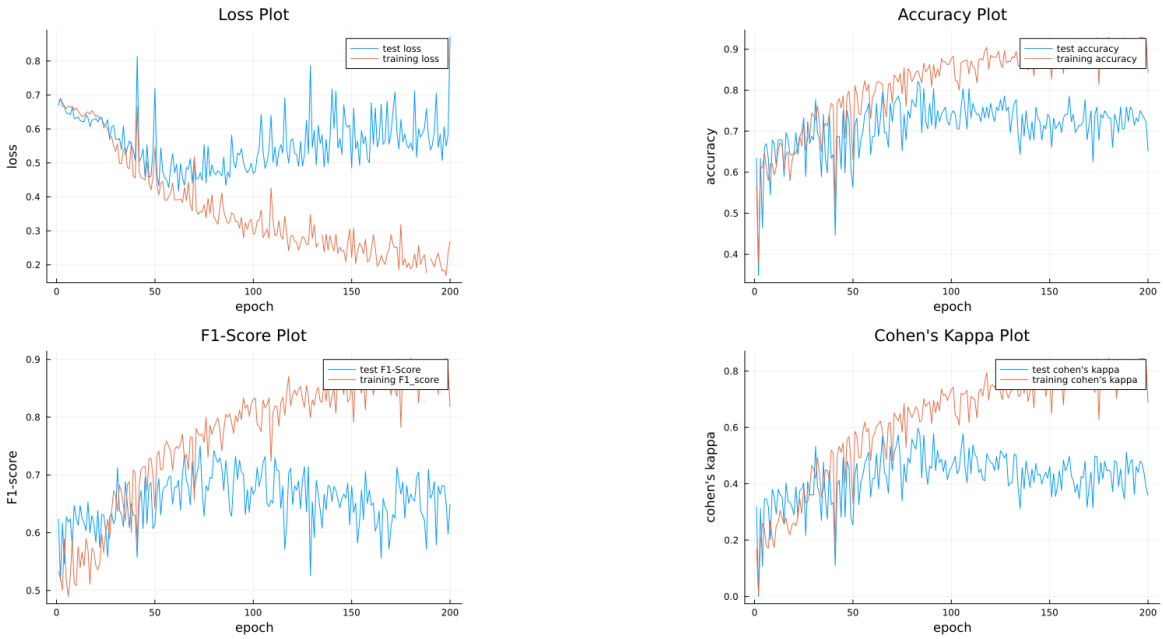

Figure 4.21: model 9 (epochs: 200 loss: 61.7 ± 12.9% accuracy: 72 ± 3.5% F1-score: 65.5 ± 3.1% Cohen's kappa: 42.9 ± 4.7)

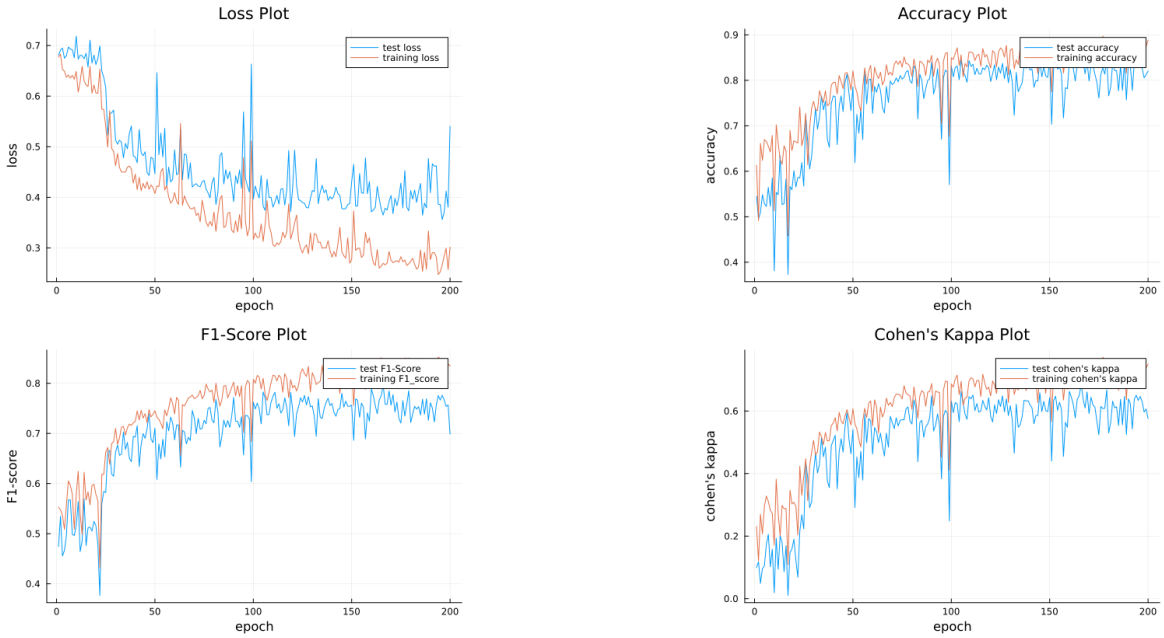

Figure 4.22: model 13 (epochs: 200 loss: 40.7 ± 6.8% accuracy: 82.2 ± 1.1% F1-score: 75.3 ± 2.8% Cohen's kappa: 61.5 ± 2.6)

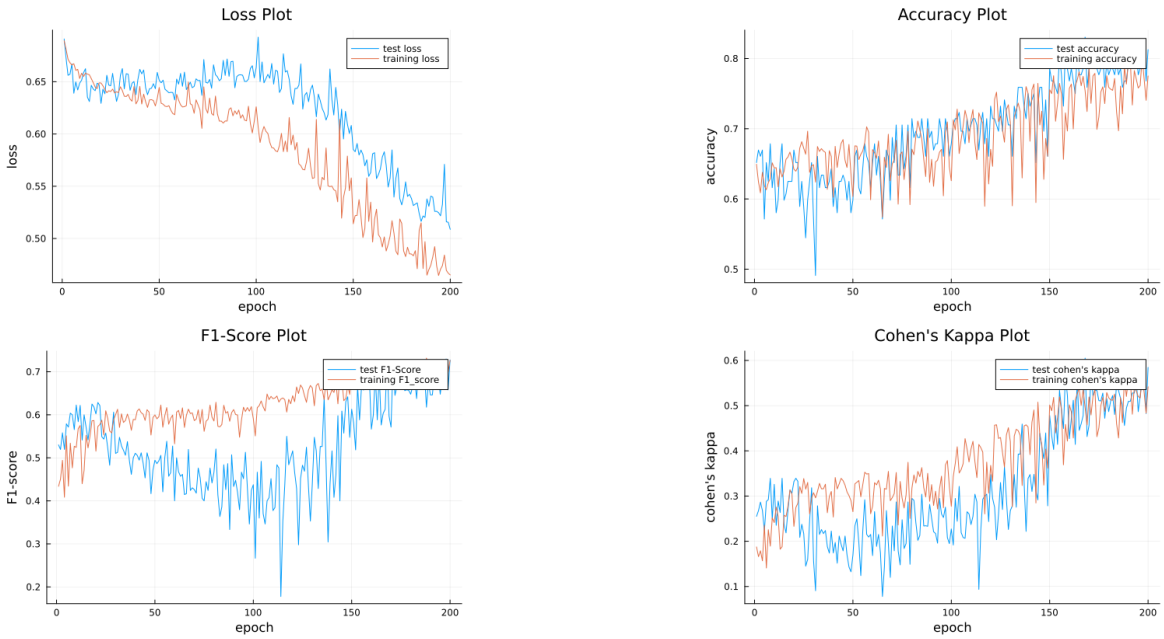

Figure 4.23: model 14 (epochs: 200 loss: 52.8 ± 2.3% accuracy: 79.8 ± 2.4% F1-score: 68.6 ± 3.5% Cohen's kappa: 53.9 ± 4.9)

During testing of the final models, the model is trialed for 60 epochs and if it performes well, it is trained for 200 epochs. The only model which didn't perform well enough for trialing it for 200 epochs was model 12 (4.20). This model stayed similar for the 40 epochs and then made a huge jump but still only achieved a max accuracy of around 70 percent which didn't seem to be able to contend the initial model.

The models that are trained for 200 epochs always achieved similar peak performances compared to the initial network. In model 9 (4.21) shortly after reaching its peak performance, the training and test performance graphs start to diverge. While the performance of the training set becomes better, the test set performance slightly decreases. This is interpreted as overfitting of the network due to a lack of training data.

The only model that reaches a higher peak accuracy then the initial network is model 13 (4.22) with 85.6 %. Its performance is significantly more stable as its oscillation is minor.

model 14 (4.23) on the other hand has a very slow performance growth which reaches its peak around epoch 200. This indicates that this model might outperform the initial model. However, when training the same network for 300 epochs the result was different than expected with the accuracy only reaching around 71% after 290 epochs even after running this simulation multiple times its results didn't change. The good results when training for 200 epochs are consequently contributed to a lucky choice of the initial parameters.

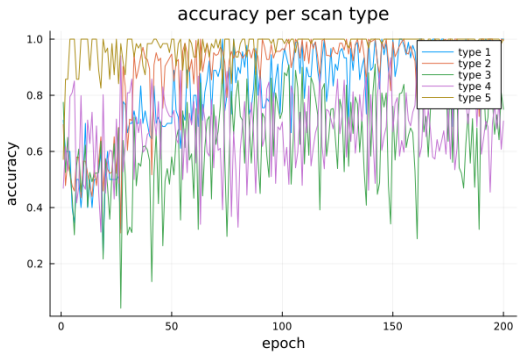

Figure 4.24: accuracy of scan type 1-5 of the initial model
(Type 1 accuracy: 94.4 ± 5.1%,
Type 2 accuracy: 97.9 ± 2.6%,
Type 3 accuracy: 72.9 ± 6.6%,
Type 4 accuracy: 70.7 ± 6.2%,
Type 5 accuracy: 99.7 ± 0.6%)

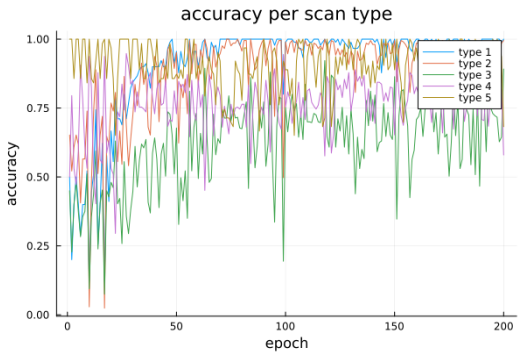

Figure 4.25: accuracy of scan type 1-5 of model 13
(Type 1 accuracy:99.4 ± 0.9%,
Type 2 accuracy: 97.6 ± 2.1%,
Type 3 accuracy: 72.2 ± 9.4%,
Type 4 accuracy: 75.9 ± 9.2%,
Type 5 accuracy: 92.3 ± 12.1%)

Subsequently, the accuracy of the initial model and model 13 are analyzed for every single scan type. This figure is chosen to visualize the difference in accuracy between the different scan types. The networks especially perform well on type 1,2 and 5 scans which achieve a mean accuracy of over 90% . If compared to type 3 and 4 scans, they only reach mean accuracy values around 70-76 % which is a significantly lower.

5 Discussion

The object of this thesis was to replicate the results of [1] and try to validate its findings for data generated by the Scanco XtremeCT I. Its model can be used to support operators in determining the severity of motion in radius and tibia scans while reducing the operator's bias. In the field of medicine it is desirable to minimize false assessments. Therefore, a high accuracy of the model is beneficial. In these cases this would lead to some patients not needing another re-scan which would reduce the radiation the patient is exposed to. Based on the findings of [1] a more sophisticated network is developed while sticking to a general CNN approach.

When trying to replicate the results of [1], the impact of different augmentation techniques on the data set was evaluated.Furthermore, the impact of static and dynamic learning rate on the training process was reviewed. On one hand traditional augmentation techniques were implemented. On the other hand the work of other researchers in similar fields was taken into consideration. One method used by [21] is to rotate the images by 1-5° which seemed to be promising, since the use of 1-360° rotation did not achieve satisfying results. This method returned a high accuracy but after its peak around epoch 50 the test and training accuracy started to diverge which might be due to the network overfitting on the data. The most promising results for this network is by a $n \cdot 90°$ rotation with n $\in$ 0-3$\mathbb{N}$, Gaussian noise, image snippets, horizontal and vertical flipping.

Additionally, different learning rates for optimizers of the Adam family were implemented. The results of these tests show the superiority of basic Adam without any learning rate decay. Even when training the network with a combination of the best augmentation technique and optimizer we still didn't get close to the performance of [1]. In the research following this assessment the same augmentation and optimizers are used. In some cases, especially when using larger networks, the learning rate had to be reduced to 0.0001 instead of 0.001 which needs to be considered when building on this research.

The results of this thesis are difficult to compare to the results of the initial paper since its scans where generated by the XtremeCT II. [1] This scanner has a scan time of 120 seconds which is less than the XtremeCT I which has a scan time of at least 168 seconds. [27] The smaller scan time reduces the risk of a patient moving and therefore the distribution of the severity level is different compared to the data used in this paper. This leads to more type 1 and 2 scans whilst reducing type 3 and 4 scans. While there is no information in the paper [1] about this distribution, it is suspected that this influences the overall accuracy. Looking at

the performance of our network for the different severity types, the network performes well on type 1,2 and 5 scans with mean accuracy values of over 90% while type 3 and 4 scans have a significantly lower accuracy of around 70-76%. This supports the assumption that the different distribution of scan types caused by the newer scanner improves the accuracy of the network significantly. This theory should be validated by using data generated by the XtremeCT II scanner. Furthermore, type 1,2 and 5 scans often reach 100 percent accuracy which is a sign that the network overfits on those data types.

Another reason why our network might have performed worse is that the initial paper includes a visual examination of the network by using the Grad-CAM method. The Grad-CAM method is used to interpret the network's decisions. This method wasn't applicable in this thesis due to the lack of healthcare professionals to interpret if the network looks at the correct regions to determine the severity of scans. Also, Grad-CAM is not implemented. Since the implementation of it would have exceeded the scope of this work, it is excluded.

Based on the findings of the replication of [1], the object has been to find a model that performs better on the given data. This was done by adding additional layers, kernels or switching layers with skip connections and inception modules. Sometimes, due to a lack of training data even small changes on the networks structure had major effects on the network performance. Most of the models showed no real improvement to the initial network. Most of the networks seemed to struggle with the amount of given data or did not show any significant difference in performance to have a major effect on the results. Only one model outperformed the initial model (loss: 40.7 ± 6.8% accuracy: 82.2 ± 1.1% F1-score: 75.3 ± 2.8% Cohen's kappa: 61.5 ± 2.6) with the highest recorded accuracy being 85.62%. This is 1.13 percent higher than the highest accuracy of the initial network. Additionally, it seems to be more stabled since it has less oscillation.

Even though not all of the layer performed as well, the addition of inception and skip connections seemed to have a positive effect on the network's performance. One major contributor to the networks not reaching higher performance is attributed to the size of the training set. To prove this theory, testing on a larger data set is necessary.

6 Conclusion

As a conclusion it can be said that the network described in [1] is a good fit for the amount of training data used. Small data sets often bring the issue with them, that even networks with a few layers can be prone to overfitting. This can be seen in this paper. Even the addition of a few more parameters leads the network to overfit or result in significantly worse results. This is not as visible when training for only 60 epochs, but when training for 200 epochs it becomes obvious.

It is hard to compare our findings with the results of [1] since the paper used scans of the XtremeCT II. The network achieve significantly lower performances on the used data from the XtremeCT I scanner. When looking at the performance of the different scan types, the network has a mean accuracy of over 90% for type 1,2 and 5 scans and only 70-76% mean accuracy for type 3 and 4 scans. Since the XtremeCT I has a longer scan time, scans are more likely to contain type 3 and 4 motion artifacts. The reduction of those scan types, which might be possible by using the XtremeCT II, might lead our network to achieve better performances. To prove this theory a test on XtremeCT II scans is recommend.

By analyzing the performance of operators on the data it becomes apparent that the main issue is the detection of type 3 and 4 scans. Therefore, an overall high accuracy won't support the operators in their work if they can't rely on the network's performance for detecting type 3 and 4 scans. A network which is only trained on this type of data (type 3 and 4 scans) might be able to achieve higher performances and consistently be able to reduce the human bias when its influence is the strongest.

Even with the small amount of data it was possible to achieve a small increase in the peak accuracy and stability of the network. This was accomplished by switching the fifth convolution layer with a skip connection containing two convolution layers. In other tests the training and test graphs often started to diverge which indicates that the networks overfitted. It still showed signs that the addition of inception layers and skip connections might be able to benefit the network when trained on a bigger data set. Therefore, it is proposed to repeat the tests on a larger data set to evaluate which impact this changes have on the network's performance.

Bibliography

[1] Matthias Walle, Dominic Eggemann, Penny R. Atkins, Jack J. Kendall, Kerstin Stock, Ralph Müller, and Caitlyn J. Collins. Motion grading of high-resolution quantitative computed tomography supported by deep convolutional neural networks. *Bone*, 166:116607, jan 2023.

[2] D.E. Whittier, S.K. Boyd, A.J. Burghardt, J. Paccou, A. Ghasem-Zadeh, R. Chapurlat, K. Engelke, and M.L. Bouxsein. Guidelines for the assessment of bone density and microarchitecture in vivo using high-resolution peripheral quantitative computed tomography. *Osteoporosis International*, 31(9):1607–1627, may 2020.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[4] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, jun 2018.

[5] Andrew J. Burghardt, Thomas M. Link, and Sharmila Majumdar. High-resolution computed tomography for clinical imaging of bone microarchitecture. *Clinical Orthopaedics Related Research*, 469(8):2179–2193, aug 2011.

[6] J.P. van den Bergh, P. Szulc, A.M. Cheung, M. Bouxsein, K. Engelke, and R. Chapurlat. The clinical application of high-resolution peripheral computed tomography (HR-pQCT) in adults: state of the art and future directions. *Osteoporosis International*, 32(8):1465–1485, may 2021.

[7] Yingjie Tian, Duo Su, Stanislao Lauria, and Xiaohui Liu. Recent advances on loss functions in deep learning for computer vision. *Neurocomputing*, 497:129–158, August 2022.

[8] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. November 2017.

[11] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks, 2015.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. February 2015.

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

[14] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, aug 2017.

[15] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pages 227–236. Springer, 1990.

[16] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1), mar 2021.

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[19] Mats L. Richter, Julius Schoning, Anna Wiedenroth, and Ulf Krumnack. Should you go deeper? optimizing convolutional neural network architectures without training. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, December 2021.

[20] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), July 2019.

[21] Qiang Zhang, Evan Hann, Konrad Werys, Cody Wu, Iulia Popescu, Elena Lukaschuk, Ahmet Barutcu, Vanessa M. Ferreira, and Stefan K. Piechnik. Deep learning with attention supervision for automated motion artefact detection in quality control of cardiac t1-mapping. *Artificial Intelligence in Medicine*, 110:101955, nov 2020.

[22] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[23] Richard Dinga, Brenda W.J.H. Penninx, Dick J. Veltman, Lianne Schmaal, and Andre F. Marquand. August 2019.

[24] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, April 1960.

[25] Susana M. Vieira, Uzay Kaymak, and Joao M. C. Sousa. Cohen's kappa coefficient as a performance measure for feature selection. In *International Conference on Fuzzy Systems*. IEEE, July 2010.

[26] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[27] Doris My-Lan Tran, Nicolas Vilayphiou, and Bruno Koller. Clinical in vivo assessment of bone microarchitecture with ct scanners: An enduring challenge. *Journal of Bone and Mineral Research*, 35(2):415–416, December 2019.