FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Detection of defects in X-Ray images using Neural Networks
Student:	Matúš Botek
Supervisor:	Ing. Jakub Žitný
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of winter semester 2022/2023

Instructions

Research current state-of-the-art techniques that are used for detection and segmentation tasks in the medical imaging domain, and focus on X-Ray images. Implement one or more models that will work on datasets provided by the supervisor. Compare their performance and focus on preprocessing data in order to achieve the best accuracy with chosen models. Discuss the pros and cons of the various preprocessing approaches. Publish your prototype code and make sure your results are reproducible.

Electronically approved by Ing. Karel Klouda, Ph.D. on 10 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Detection of defects in X-Ray images using Neural Networks

Matúš Botek

Department of Applied Mathematics
Supervisor: Ing. Jakub Žitný

May 12, 2022

Acknowledgements

I would like to thank my supervisor, Ing. Jakub Žitný, for his valuable insight and guidance during the writing process of my thesis. Furthermore, I would also like to thank my family and friends, who supported me the whole time.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 12, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Botek, Matúš. *Detection of defects in X-Ray images using Neural Networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato práce si klade za cíl shrnout různé techniky zlepšování a nejmodernější přístupy použitelné pro úkoly v oblasti lékařského zobrazování. S těmito znalostmi jsou implementovány a trénovány modely konvolučních neuronových sítí na datovém souboru muskuloskeletálních rentgenových snímků. Je sledován a diskutován dopad na výkon modelu použitím více metod předzpracování dat.

Klíčová slova konvoluční neuronové sítě, rentgenové snímky, MURA, klasifikace, předzpracování dat, TensorFlow, Keras

Abstract

This thesis aims to summarize different improvement techniques and state of the art approaches usable for tasks in medical imaging domain. With this knowledge, convolutional neural network models are implemented and and trained on a musculoskeletal radiographs dataset. The impact on model performance by applying multiple preprocessing methods is watched and discussed.

Keywords convolutional neural networks, X-Ray images, MURA, classification, data preprocessing, TensorFlow, Keras

Contents

List of Figures

List of Tables

Introduction

Machine learning and neural networks are not a brand new concept, but nowadays, the relatively easy access to computational resources and increasing quality and quantity of data that is being collected are causing significant growth of interest in these topics. The interest is not solely oriented on information technology as such. Machine learning is being found useful more and more in all sorts of fields like commerce, healthcare, education and others. This gives us new opportunities to research ways of utilizing machine learning approaches in fields where it can make positive difference.

Healthcare is definitely one of those fields. Most medical domain tasks require skilled professionals to solve them. Sometimes, the demand for those professionals is not satisfiable and that can lead to complications when the tasks need to be solved in a timely manner. Machine learning, specifically convolutional neural networks can be utilized to aid in these tasks. The presence of an expert is still not fully replaceable, but using convolutional neural networks as a form of assistance can be of use. For example pointing out potenial problems automatically can make the professionals decision making faster and more reliable as it provides another view into the task. There is also a possibility to develop models for these tasks and then deploy them in areas where medical help is absent to provide a partial replacement in patient diagnosis.

This thesis focuses on researching techniques that are used for detection and segmentation in medical imaging domain, especially on X-Ray images. The next goal is to implement some of those techniques and use them in order to solve a task from medical imaging domain. In this case, the implementation will try solving binary classification task of musculoskeletal X-Ray scans into normal and abnormal categories.

Chapter 1

Machine Learning

Machine learning (ML) is a computer science field focused on algorithms that adjust themselves to the given task based on experience and provided data in order to improve their performance. They are capable of extracting features and finding relationships or rules in provided data. This chapter introduces ML and describes it's fundamentals.

1.1 Supervised and Unsupervised Learning

We can divide ML algorithms into two main categories, based on the learning approach that is used. These categories are supervised and unsupervised learning

1.1.1 Supervised Learning

In supervised learning, the algorithm works with a set of labeled data. It receives n input-output pairs $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ where x_i represents the input and y_i the corresponding labeled output. The algorithm's task is to find a function that will approximate the true function $f : X \to Y$ which maps inputs X to their corresponding outputs Y. The goal is to correctly assign y values not only to inputs from provided pairs but also to new inputs for the same task that were not yet seen, by learning on the provided example pairs. We call the set of example input-output pairs training data.[1]

We can further distinguish supervised learning tasks by their domain of output values. Tasks where inputs are assigned to a small, finite set of output values are called classification problems. When the domain of output values is continuous or there are so many values that it's better to treat them as continuous, it is a regression problem. Below are some examples for supervised learning approaches:

- Logistic regression (classification problems)

- Linear regression (regression problems)
- Decision trees (both problem types)

1.1.2 Unsupervised Learning

Unlike supervised learning, for unsupervised learning there are no labels provided with the input data. The algorithm has no reference for determining the output values, so the goal is to find connections, similarities or patterns between the inputs.[2] These are some of the common unsupervised learning approaches:

- Clustering - The goal of clustering is to divide the inputs into several clusters so that the similar inputs belong to the same cluster and dissimilar inputs are in separate clusters.[2]
- Dimensionality reduction - This technique can be used to reduce the number of features for given dataset in order to improve the performance of other ML methods, while trying not to remove important information.[3]

Unsupervised learning can also be very useful at extracting features or detecting anomalies and outliers.

1.2 Metrics

Measuring the performance of models and the quality of achieved results is a crucial part in ML models training. The use of metrics can help us compare different training approaches or simply evaluate achieved outcomes. There are many different types of metrics each suitable for specific type of tasks. They usually reflect the goal of our task. This section focuses on metrics relevant to binary classification tasks.

1.2.1 Basics

Confusion Matrix visualizes the relevant values used for calculating other metrics used for binary classification.[5] It contains these values:

- True Positive (TP): number of positive samples classified as positive
- False Positive (FP): number of negative samples classified as positive
- False Negative (FN): number of positive samples classified as negative
- True Negative (TN): number of negative samples classified as negative

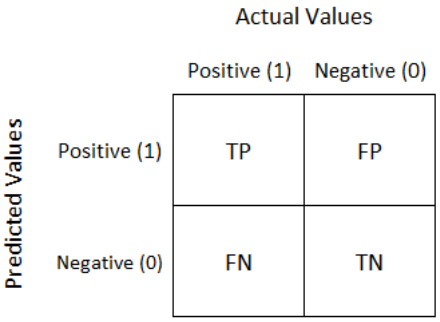

Figure 1.1: Confusion matrix[4]

After defining these 4 values, we can define the following metrics:[5][4]

Accuracy is the most common and simple metric used for measuring performance of ML algorithms in medicine, but it tends to be misleading when the dataset classes are imbalanced. If the model were to always predict the majority class, it would achieve high accuracy even though it completely ignored the minority class. We define it as

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

True Positive Rate / Sensitivity / Recall represents the fraction of positive samples that were correctly classified.

$$TPR = \frac{TP}{TP + FN}$$

True Negative Rate / Specificity represents the fraction of negative samples that were correctly classified.

$$TNR = \frac{TN}{TN + FP}$$

Positive Predictive Value / Precision represents the chance that a sample classified as positive is actually positive

$$PPV = \frac{TP}{TP + FP}$$

Negative Predictive Value represents the chance that a sample classified as negative is actually negative

$$PPV = \frac{TP}{TP + FP}$$

False Positive Rate represents the fraction of negative samples that were incorrectly classified

$$FPR = \frac{FP}{TN + FP} = 1 - Specificity$$

1.2.2 Area Under the Curve

Receiver operator characteristic (ROC) curve is a metric that plots true positive rate against false positive rate. We can measure model performance by using the Area under the curve (AUC) ROC curve, AUC-ROC curve. AUC tells us how good is the model in distinguishing positive samples from negative samples. We want our model to have the highest AUC possible, AUC = 1 means the model always classifies the sample correctly, AUC = 0.5 means the model classifies at random and has no ability to distinguish the samples and AUC < 0.5 means the separability is worse than random.[6] Figure 1.2 shows an example for ROC curve.

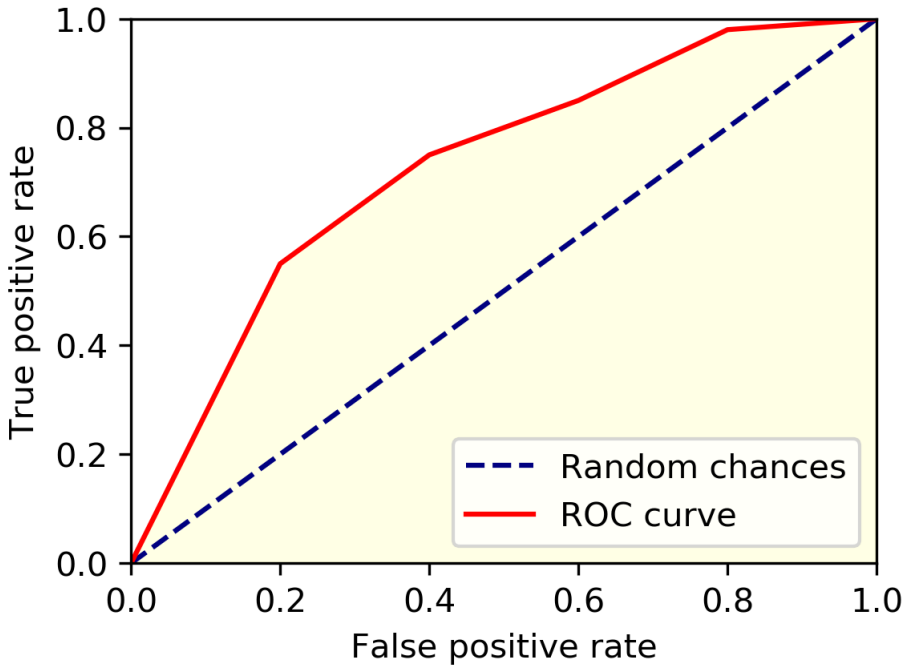

Figure 1.2: ROC curve[7]

1.3 Working with Data

When it comes to data, there are some practices that should be taken into account when working with ML models. The goal of every model is to learn from available data. The amount of data can determine how well can the model generalize the task and how good it's performance will be on unknown samples. But no matter how big our data is, we should provide only a portion of it to our model. When we provide data to the model, we want to measure it's performance, but doing that only on the training data might be unreliable as models are prone to learn too much detailed features and end up degrading their performance. It is much more reliable to simulate model response to new, unseen data. We can do a train test split. We select a small portion of data and use it as a test set. This data will be used only after the model has been trained to verify it's ability to generally solve the given task.

Another more advanced option is to do a train-validation-test split. The training set is used to train given model with specific parameters and then the model performance is measured on the validation set. The validation set acts as new data for the model and therefore should provide more accurate measurement. We can use the error from validation set to choose the correct configuration of our model. This is called model selection[8] The test set should only be used to make a final measurement of model performance. By comparing the model performance on training and test data we can spot learning problems.

1.3.1 Overfitting and Underfitting

Overfitting and underfitting are a common problem in the ML domain. When a model fits perfectly on the training data and then performs poorly on test data, we can calculate the difference in performance on train and test data, it is called the generalization gap. Big generalization gap is a sign of overfitting.[9] On the other hand, when the model is underfitting it shows that the model was not able to extract any relations from the data, it did not learn how to solve the task.

1.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are derived from traditional Artificial Neural Networks (ANNs). Their are primarily used for pattern recognitions in the imaging domain. CNNs are built using three types of layers.

- convolutional layers
- pooling layers
- fully-connected layers

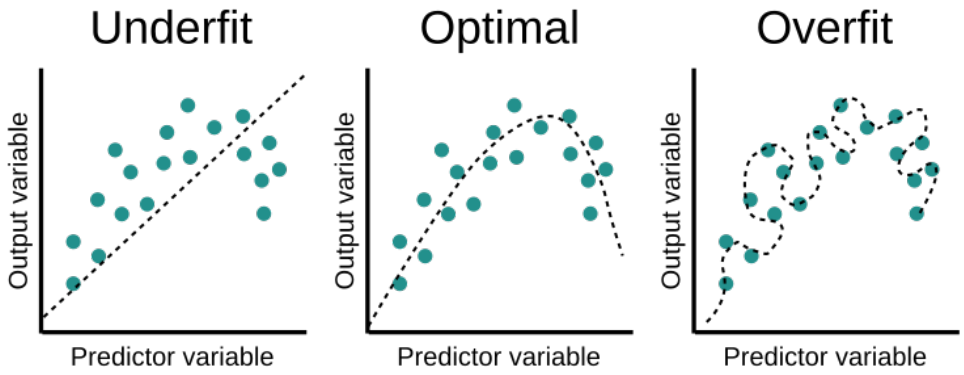

Figure 1.3: Overfitting and underfitting[10]

Basic run of a CNN consits of multiple steps. First, the input layer will obtain the input image shaped to appropriate dimensions, then operations will be applied on the image based on the layer it goes through. [11]

"The convolutional layer will determine the output of neurons which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume."[11]

The pooling layers are used for applying downsampling, then the fully-connected layers produce scores used for classification.

Chapter 2

State of the Art

This chapter covers multiple improvement techniques and state of the art approaches for working with CNNs.

2.1 Cross-validation

In section 1.3 I mentioned approaches to splitting data into distinct sets. Cross-validation is also a form of data splitting. The validation and train sets are not static. Together they are divided into K splits randomly, where for every split, all other splits are used as a training set and the specific split acts as a validation set. The model is trained like this for each split, reinitialized between them.

This way all parts of data act as new samples and the measurement is more reliable. The final model error can then be calculated by averaging all individual split errors. The approached I described is called KFold. This can be especially useful when the provided dataset would be too small if split statically.[12]

2.2 Image Preprocessing

Image preprocessing techniques focus on preparing the data prior to model training in order to improve the performance. They can try to remove unwanted defects from the images, improve some features or enhance the dataset by creating new samples.

2.2.1 Image Augmentation

Image augmentation is a good way to provide different samples for the model training. If the used dataset is too small, we can increase it by taking existing samples and applying transformations to them in order to create new data.

Image augmentation can also be used online during training to reduce overfitting. By randomly modifying the samples it lowers the chance of the model to learn detecting only specific input data.

2.2.2 Histogram Equalization

Histogram equalization is a method for increasing the global contrast of an image. It may not work good on images where multiple areas have huge differences in intensity, then it tends to create noise in the image. The Figure 2.1 shows the difference between applying CLAHE, which is mentioned later in this chapter and histogram equalization.

2.2.3 Contrast Limited Adaptive histogram equalization

Contrast limited adaptive histogram equalization (CLAHE) is another algorithm for increasing the contrast of images. It is an advancement to adaptive histogram equalization (AHE). AHE tends to overenhance noise in relatively homogeneous regions and CLAHE proposes a way to battle this overenhancement by clipping the histogram,on which the equalization is based, at a given value called the clip limit.[13] Any histogram bins, that are above the clip limit are clipped and redistributed uniformly among other bins before the equalization is applied.[14] It should also produce better results then standard histogram equalization. The comparison to histogram equalization is visible in Figure 2.1 A study proposing usage of "N-CLAHE" algorithm was published[15], where they normalized images using a log function with blank scan image.

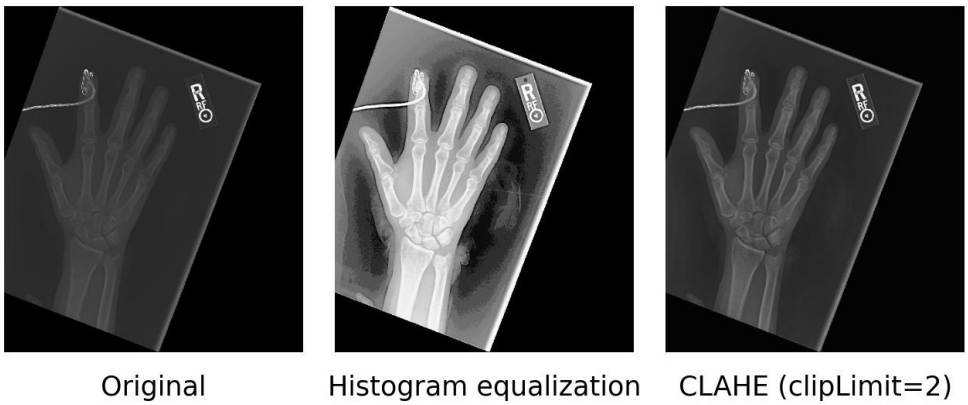

Figure 2.1: Comparison of histogram equalization and CLAHE

2.2.4 Segmentation

Image segmentation tries to segment images into multiple regions. These regions can be further used to provide additional data to images being used for training by selecting regions of interest, labeling different segments of image with custom labels. By segmenting an image one can remove unwanted regions, crop image according to found boundaries. Segmentation can be done automatically by using pre-trained models or other automated algorithms.

2.3 Gradient-weighted Class Activation Mapping

Gradient-weighted Class Activation Mapping (Grad-CAM) is a method used to better explain CNN model decisions when solving tasks. By obtaining the gradients from the last convolutional layer, it calculates signicance of each feature in the feature map for the final prediction and creates a heatmap that can visualize areas of image that took the biggest part in the final prediction. Using Grad-CAM can be beneficial when we want to find out underlying issues to which we come accross during configuration of our models. Two sample images created using my final models can be seen in Figure 5.3.

2.4 Hyperparameter Optimization

The task of successfully training ML models hugely depends on the correct choice of hyperparameters. Hyperparameters are defining the learning process and by changing them one can influence the performance. Hyperparameter optimization is a process of finding the combination of searched parameters that provide best model performance. Multiple search algorithms for hyperparameter optimization are implemented, I used a state of the art algortihm called Hyperband[16] in my experiments. It uses successive halving for filtering the search space and it adaptively assigns resources for individual trial runs.

2.5 Transfer Learning

Transfer learning is a method used to take advantage from already pretrained models. The goal of transfer learning is to transfer features learned on another dataset to a new one that is similar in order to improve the preformance. Today the most popular approach to transfer learning is taking weights from models that were trained on the ImageNet[17] dataset, which is a large competitive dataset with over 14 milion images belonging to over 20,000 categories for benchmarking. Research showed that using pre-trained weights from ImageNet dataset and transfering them to a different, not even similar problem produces satisfiable results.

Transfer learning sometimes also consists of fine-tunig. First, a CNN model pre-trained on ImageNet is created, while the top classification layers can be removed. These layers are replaced by new custom layers suitable for our task. The base architecture layers are frozen as to not disrupt the pretrained weights and only the top layers are trained. After this training ends, all or some specific layers can be fine-tuned, being trained for a small number of epochs on very small learning rate to adjust to the new task.

2.6 Ensemble

Ensemble methods are currently one of the most popular approaches to solving ML tasks. They are based on predicting the final output with the help of multiple learners. The learners can represent various machine learning algorithms and the main idea is that the use of multiple learners can overcompensate for a possible error of an individual. This approach provides several advantages. One is definitely the smaller chance of overfitting due to multiple provided predictions. Usage of different algorithms can improve the ability of the ensemble model to fit the task as wider space is can searched.[18]

The outputs from individual models can be joined into a single output by the usage of weighted methods or meta-learning. Weigthed methods assign weight to individual models outputs and meta-learning is based on learning from learners.[18]

Example usage of ensemble model can also be found in the Musculoskeletal radiographs (MURA) paper[19] as their proposed baseline model consists of multiple DenseNet models. Another paper was published on using ensemble models to classify selected studies from MURA dataset by using VGG19 and ResNet architectures.[20]

Chapter 3

Medical Imaging

This chapter summarizes my findings in regards to use of ML in medical imaging domain. I was focusing mostly on musculoskeletal applications.

Medical imaging is a field that is focused on creating body images for the purpose of diagnosis. ML algorithms and especially CNNs can contribute to this domain. For ML, quality and size of datasets is really important, some of the biggest medical imaging datasets are our used MURA dataset[19], CheXNet dataset[21] for pneumonia detection and probably the most important one these days is the COVID-19-detection dataset.[22]

In muscoskeletal research, I found several studies about MURA dataset. I found a study focused on automatic segmentation on femur bones, which contained very thourough research into the topic[23]. This study focused on detecting bone fractures in shoulder area[24] which used both individual deep learning models and ensemble models to complete the task. A paper using unsupervised methods was researching the application of those methods trained on images without anomalies for evaluating X-Ray images of hands.[25]

Even though I wanted to focus on muscoskeletal research, I wanted to include Covid-19 works as it is a totally new task widely and actively being research. Some studies are focused on data preprocessing[26] using segmentation to improve the images, even a deep convolutional neural architecture named COVID-NET was proposed[27] in late 2020. It was and still is a goal for many ML researchers to aid in the fight against Covid-19 desease and being able to detect it using ML could be very beneficial.

Chapter 4

Analysis and Design

This chapter lists technologies used during the implementation, describes major practical part decisions, including the chosen dataset and CNN architectures.

4.1 Technologies

4.1.1 Python and Jupyter Notebook

The whole implementation is written in Python. It has high code readability and strong community support. There are many powerful libraries focused on ML for Python, which make it one of the most popular programming languages when it comes to ML tasks. The version I used is Python 3.8. For better presentation purposes, some code is written in Jupyter Notebook files. Jupyter Notebook is a web application for creating documents consisting of independent cells that can contain code, text or visualisations. These documents support running Python code, with outputs being visible directly in the document.

4.1.2 TensorFlow and Keras

Model building and training was done using open-source libraries TensorFlow and Keras. TensorFlow is a machine learning platform originally developed by Google. It can be used in multiple programming languages. It is efficient at executing low-level tensor operations on CPU, GPU, or TPU, with the possibility of scaling computation to many devices.[28] I was using the latest version of this library, TensorFlow 2.8.

Keras is a high-level deep learning API, running on top of TensorFlow. The library is written in Python and it's aim is to make model building and experimentation faster and simpler together with utilizing the scalability and efficiency of TensorFlow. Together with it's Functional API, it allows us to

build both simple and complex architectures, with many of the popular CNN architectures made available as pre-built models. It also offers data preprocessing and hyperparameter tuning methods.[28]

4.1.3 Other Libraries

In addition to libraries mentioned above, I used Pandas for dataset handling using Dataframes, Matplotlib for creating visualisations, NumPy for mathematical operations, OpenCV for image preprocessing and Scikit-learn for cross-validation.

4.1.4 Google Colab

Google Colab is a cloud-based service offering computational resources including GPUs for Python code execution. It can run Jupyter Notebooks and access files from Google Drive or GitHub. The computational resources are limited and might fluctuate. Training a CNN is a computationally demanding task, therefore I chose to run most of the training and experiments on Google Colab.

4.1.5 Code Availability

The whole implementation is shared in a public Github repository with comments and descriptions for better understanding of the project.

4.2 Dataset

The dataset I used is the MURA dataset assembled by researchers from Stanford University. It contains 14,863 studies from 12,173 patients, totalling 40,561 images. All images were obtained from the Picture Archive and Communication System of Stanford Hospital. The body parts present in the studies are the following: elbow, finger, hand, humerus, forearm, shoulder and wrist. Each study contains images of only one specific body part and was manually labeled as normal or abnormal by board-certified radiologists from the Stanford Hospital. The dataset was split into train, validation and test sets without patient overlap between them.[19] Detailed distribution of normal and abnormal studies can be seen in Table 4.1.

4.2.1 Dataset Preparation

I was only able to obtain the train and validation sets of the dataset. The test set is not publicly available, so I decided to reproduce the test set on my own, using a subset of original training data. In the MURA research paper[19], they state, that the test set contains 556 images belonging to 207 studies.

Study	Train		Validation		Total
	Normal	Abnormal	Normal	Abnormal	
Elbow	1094	660	92	66	1912
Finger	1280	655	92	83	2110
Hand	1497	521	101	66	2185
Humerus	321	271	68	67	727
Forearm	590	287	69	64	1010
Shoulder	1364	1457	99	95	3015
Wrist	2134	1326	140	97	3697
Total No. of Studies	8280	5177	661	538	14656

Table 4.1: Distribution of normal and abnormal studies, taken from MURA research paper[19]

No further information such as label or body part distribution is provided. I decided to copy the study distribution from training set by calculating the ratio of studies per body part to all studies. I then calculated the approximate number of studies using the ratio and original number of test studies and ended up with a test set consisting of 204 studies and 548 images.

4.2.2 Data Exploration

The images from the dataset vary in multiple aspects. They have different dimensions, lighting properties, portions of captured body parts. I found some images that contain multiple X-ray scans of the same body part in slightly different positions, grouped together, others with scans covering more than one body part, for example both hands in one image. There are also some inscriptions present outside the scanned areas. When the scan focuses on a smaller, specific area, the image sometimes has unnecessarily big background with the actual scan taking only a small portion of the image. A selected sample of dataset images can be seen in Figure 4.1. These mentioned properties could influence the learning process, so I tried several approaches to deal with them during experimentation.

4.3 Implementation Fundamentals

This section introduces and explains core parts of the implementation, on top of which the experimentation part is built.

4.3.1 Models

I chose two CNN architectures available directly from Keras functional API as a base for my models, ResNet50 and DenseNet169. Both of them achieved

Figure 4.1: Sample of dataset images

impressive results on the most popular competitive image datasets and are widely used in a range of classification or detection tasks. I conducted several experiments with them and compared their performance.

4.3.1.1 ResNet50

The ResNet model[29] was introduced in a paper 2015. The variant ResNet50 has 48 convolutional layers and 1 max pooling and 1 average pooling layer. It is also available from the Keras Functional API, which makes it easy to implement even with custom changes. I used the model preloaded with ImageNet weights, removed the original classification layers top and replaced it with one Global Average Pooling layer and a Dense layer with one output and Sigmoid activation. The input size for the model was (224, 224, 3). The detailed architecture can be seen in Figure 4.2.

4.3.1.2 DenseNet169

The ResNet model[31] was introduced in a paper 2015. The variant ResNet50 has 48 convolutional layers and 1 max pooling and 1 average pooling layer. It is also available from the Keras Functional API, which makes it easy to implement even with custom changes. I used the model preloaded with ImageNet weights, removed the original classification layers top and replaced it with one Global Average Pooling layer and a Dense layer with one output and Sigmoid activation. The input size for the model was (224, 224, 3). The detailed architecture can be seen in Figure 4.3.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.2: ResNet50 architecture[30]

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix}\times48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 4.3: DenseNet169 architecture[32]

4.3.2 Cohen's Kappa

The metric I used for the evaluation is Cohen's kappa statistic(κ). I chose this metric because both the model proposed in the original MURA paper and models from other researchers, who took part in a MURA dataset competition, created by the Stanford researchers, were evaluated using Cohen's kappa. That allowed me to compare my results with the baseline and competition models.

Furthermore, in the medical, and in our case, musculoskeletal research, it is often important to determine the reliability of observations made by experts. The use of Cohen's kappa is suited for that task.[33] This metric measures the agreement between two raters, also called inter-rater reliability.

It's value ranges from -1 to 1, with $\kappa = 0$ meaning there is only agreement expected by chance between the raters. Negative values indicate there is no agreement between the raters or that they are in disagreement. These negative values don't typically occur in practice. Values above 0 represent increasing agreement with $\kappa = 1$ being perfect agreement. Cohen's kappa can be calculated using the following formula

$$\kappa = \frac{p_a - p_e}{1 - p_e}$$

where p_a is the observed agreement between the two raters and p_e represents the expected agreement by chance. The chance agreement is calculated using the numbers of labels for each category from both of the raters. In our case, we can calculate it as

$$p_e = \frac{\frac{A_1 \cdot A_2}{T} + \frac{N_1 \cdot N_2}{T}}{T}$$

where A_i, N_i stand for the number of images labeled as abnormal or normal by rater i and T represents the total number of labeled images.[34]

Cohen's kappa statistic is not affected by imbalanced datasets as it focuses solely on the differences in predictions between raters. In our case, the image labels provided by radiologists act as one rater, with our model being the other one. If our model learned to always predict only one class due to high imbalance, Cohen's kappa would reflect that even though achieved accuracy could be seemingly good.

4.3.3 Class Weights

The numbers of abnormal and normal studies, more specifically abnormal and normal images in MURA dataset are not balanced. To address this issue, I implemented class weights that were passed to the loss function during training. I used formulas for calculating binary task weights $w_0 = N/(A + N)$, $w_1 = A/(A + N)$, where w_i is weight for predicted label i, A is number of abnormal images and N is number of normal images.

4.4 Image augmentation

Online image augmentation is a good way to limit model overfitting. I decided for this setup for all of my training runs, if it is not later explicitly stated otherwise. I used ImageDataGenerator from Keras to randomly apply augmentations that I specified. I allowed random rotations up to 20 degrees, width and height shifts by up to 0.05 proportion of the image, brightness shift value from range $(0.9, 1.1)$, random zoom from range $(0.9, 1.1)$ and horizontal flips.

4.4.1 Training Run

Training was done in a Jupyter notebook which served as a template for configuring the desired model, augmentation, preprocessing, hyperparameters. The notebook was run in Google Colab to make use of the available GPU resources. I used the train validation test split approach when dividing the dataset. The CNN model was trained for a given number of epochs on training data, then being validated after each epoch. For the loss function I chose weighted binary cross-entropy with class weights calculated as mentioned above. Cohen's kappa statistic was used as a metric. I also used checkpoints for storing the best model weights from all epochs, measured by Cohen's kappa metric on validation set at the end of each epoch. For longer training runs, early stopping was implemented to stop the training when Cohen's kappa stopped improving. Images were fed to the model in batches of given size. At the end of the training run, model was restored with it's best weights from created checkpoints and evaluated on my manually-created test set described in Section 4.2.1.

Chapter 5

Experiments and Results

This chapter reports all experiments that I conducted. The main focus was on trying different preprocessing techniques and comparing their impact on model performance. After that, some other experiments were carried out, with preprocessing methods that performed the best during last experiment. Both ResNet50 and DenseNet169 models were used in the experiments. At the end of this chapter, the final results are measured.

5.1 Image Preprocessing Comparison

The first experiment I tried was the comparison of different image preprocessing methods. As I mentioned earlier, during the data exploration I found some images with low brightness, inscriptions and other properties that could potentially make the learning process harder. Preprocessing can help the model extract features easier and therefore should bring better results.

For this experiment I decided to use a subset of the MURA dataset. I only used shoulder studies, as they had good ratio between normal and abnormal studies and second most number of studies per body part, after wrist studies, which were more imbalanced. The experiment was performed on both models, default setup for both models if not stated otherwise was:

- Adam optimizer with initial learning rate = 0.0001, $\beta_1 = 0.9, \beta_2 = 0.999$
- Batch size = 32
- Model weights initialized with weights pre-trained on ImageNet by corresponding used model
- Images were resized to 224x224 pixels and their pixel values normalized between 0 and 1, rescaling them by 1/255

5.1.1 Normalization

The first preprocessing method I used was normalization, as it is described at the start of this experiment, rescaling each pixel by 1/255. Normalization ensures that all the features have normalized scales, this is important for multiple ML models. I used normalization as the last method after every other preprocessing method as it should have only positive efect on the model performance and it's usage on input data very usual.

5.1.2 CLAHE

Another method I used was CLAHE. As desribed in 4.2.1, it improves the images contrast with limiting noise amplification. I empirically selected two values for CLAHE's clip limit parameter, 2 and 10 and kept the tile sizes (8,8) for both. Figures 5.1 and 5.2 show the comparison of applying CLAHE with both clip limit values. Initially I experimented with the clip limit value and saw that bigger values led to over-intensified portions of the image. It is also visible from the figures that both values have their disadvantages. Lower clip limit increases the contrast but on some darker images it may be insufficient. On the other hand, the higher clip limit created some over-intensified regions on bright images.

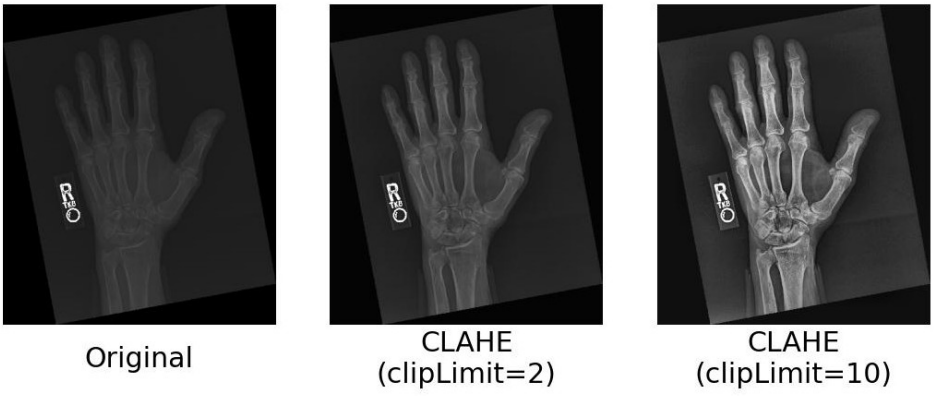

Figure 5.1: CLAHE clip limit comparison - darker original image

5.1.3 Histogram Equalization

A preprocessing method similar to CLAHE is histogram equalization. It also adjusts the contrast of the image but tends to create more noise then CLAHE overall as it only focuses on the whole image, not the local areas.

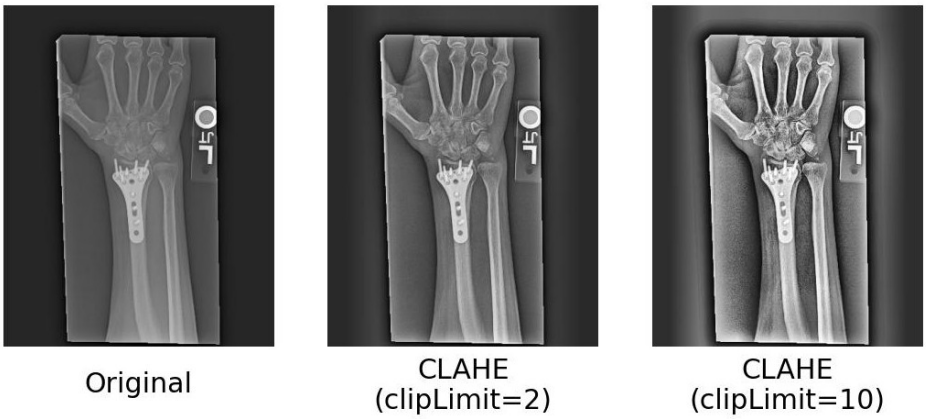

Figure 5.2: CLAHE clip limit comparison - brighter original image

5.1.4 Cropping Pipeline

As the last preprocessing method I created my own pipeline that consisted of multiple steps. At first, I determined the dominant color of the image which I transformed into a grayscale pixel intensity. Then, if this value indicated that the image is mostly white, it was inverted, because during my manual data exploration I found some images that looked as if their colors were inverted and when using binary masks on them, important image areas were masked out. I then applied CLAHE with clip limit = 4, tile size = (8, 8) and blurred the image using gaussian blur with kernel = (5, 5). During the testing of my method this proved to lead to better mask selection when applying binary threshold to the image. I determined the threshold for binary thresholding using Otsu's method[35], which determines it automatically from the image histogram. Then by using a bounding rectangle around the created mask, I cropped he image if the area of the rectangle was bigger than a proportional area of the whole image. I implemented this in order to rule out the cropping of wrongly selected areas containing bright inscriptions. After the potential cropping, another binary threshold with Otsu's method and I find the contours of the mask. Then I select the biggest one a fit a convex hull onto it. If the convex hull area is bigger than a proportion of the whole image a mask created from the convex hull area is used to remove all regions outside the convex hull. This area threshold is also implemented in order to deny removal of important areas, which usually happens if the convex hull was found only on a small portion of the image, for example a bright part of one bone. Lastly, I fit another bounding rectangle around the image and cropped it if it was possible. My aim with this method was to address multiple defects I found during data exploration4.2.2. I tried to remove unnecessary background, detect the region of interest and remove the outside regions. I also hoped this method will be

able to remove the inscriptions on the images. A sample of visualizations for this pipeline can be seen in Figure 5.4.

5.1.5 Preprocessing Results

As I mentioned before, all of the preprocessing methods were applied to both models and the images were always normalized at the end. To save computation time, I applied CLAHE and my custom cropping pipeline preprocessing offline and saved each image into a new, preprocessed dataset. Histogram equalization and normalization were not computationally expensive, so I applied them online during model training. I ran each model configuration for 10 epochs and stored the best weights. When the training was complete, I restored the weights to the best ones and evaluated the model on my custom test set. Table 5.1 contains the results of this experiment. All of the proposed methods seem to have achieved satisfiable results. It seems that the custom cropping pipeline was capable of removing defects in images and that lead to increased model performance. It achieved value κ over 0.6 in both models. In ResNet50, normalization and histogram equalization achieved worse results than both CLAHE approaches, which I expected. I was surprised to see CLAHE with clip limit = 10 outperform the other one as I expected the higher clip limit would have similar an negative effect on the performance. In DenseNet169, CLAHE with clip limit = 2 had better results than normalization and histogram equalization, but CLAHE with clip limit = 10 performed significantly worse than in ResNet50 model, achieved κ almost the same as normalization.

To address this, I conducted a quick experiment by running DenseNet169 and CLAHE with clip limit = 10 again, for the same number of epochs. The best weights achieved even worse results, $\kappa = 0.427$. I decided to consider it a bad configuration for DenseNet169 and proceeded to next experiments.

Preprocessing method	Cohen's kappa(κ)	
	ResNet50	DenseNet169
Normalization	0.510	0.522
CLAHE (clipLimit=2)	0.578	0.593
CLAHE (clipLimit=10)	**0.638**	0.526
Histogram Equalization	0.553	0.560
Cropping pipeline	0.618	**0.616**

Table 5.1: Cohen's kappa(κ) values achieved in preprocessing experiment, measured on shoulder studies test with best result for each model highlighted.

5.2 Hyperparameter Optimization

The next experiment was hyperparameter optimization. I used a framework KerasTuner available from Keras library. For each model I selected the preprocessing method which achieved best Cohen's kappa value. The parameters I chose for optimization and their possible values were:

- Batch size:
  [8, 16, 32]
- Learning rate for Adam optimizer:
  [0.001, 0.0005, 0.00025, 0.0001, 0.00005]
- Pooling layer after convolution layers:
  max or average pooling

The algorithm I chose for hyperparameter optimization was Hyperband. It uses successive halving as a subroutine for filtering out parameter configurations.[16] I set the maximum number of epochs to 80 and limited each trial run to 7 epochs. In addition, early stopping with patience of 5 epochs was used. The training was done again on shoulder studies. The best configuration of searched parameters found during hyperaparameter optimization is in Table 5.2. The run summary can be seen in corresponding Jupyter notebooks for hyperparameter optimization

These parameters combined with selected preprocessing method for each model from last experiment created my two final model configurations, which I trained and evaluated on the full dataset in the next experiment.

Parameter	ResNet50	DenseNet169
Batch size	32	16
Learning rate	0.0001	0.0001
Pooling	average	average

Table 5.2: Optimized hyperparameter values

5.3 Full Dataset Training and Cross-validation

This is the final experiment that I tried. Using the information I gained from previous experiments, I measured the performance of my final models on the full dataset.

The ResNet50 model was trained using CLAHE with clip limit = 10 and normalization, batch size of 32, Adam optimizer with learning rate = 0.0001, global average pooling layer after the last convolutional layer with weights initialized by pre-trained weights from ImageNet.

The DenseNet169 model was trained using custom cropping pipeline and normalization, batch size of 16, Adam optimizer with learning rate = 0.0001, global average pooling layer after the last convolutional layer with weights initialized by pre-trained weights from ImageNet.

At first, I trained each model for 10 epochs on the full dataset, then evaluated it on my test set. ResNet50 achieved $\kappa = 0.618$, DenseNet $\kappa = 0.712$. These results looked promising, but in order to be more confident in the results, I used cross-validation for both models. It was a KFold cross-validation with 4 folds, each run for 5 epochs with early stopping after 3 epochs with no improvement. Tables 5.3 and 5.4 show us the results from cross validation. There is the best Cohen's kappa achieved on validation set during training per each fold.

After cross-validation, I restored each model from weights obtained from best epoch across folds and evaluated it on the test set. ResNet50 achieved $\kappa = 0.591$ and DenseNet169 $\kappa = 0.683$ which is slightly worse then from the initial full dataset training run, but still very promising.

ResNet50	
Fold	Cohen's kappa (κ)
Fold1	**0.560**
Fold2	0.553
Fold3	0.540
Fold4	0.533

Table 5.3: Cross-validaton results for ResNet50 with Cohen's kappa from best epoch per fold

DenseNet169	
Fold	Cohen's kappa (κ)
Fold1	0.576
Fold2	0.568
Fold3	0.558
Fold4	**0.593**

Table 5.4: Cross-validaton results for DenseNet169 with Cohen's kappa from best epoch per fold

5.4 Results Summary and Discussion

After all the conducted experiments I obtained two trained models that had the best performance throughout the experiments. The final models were trained using cross-validation with 4 folds, each running for 5 epochs and the

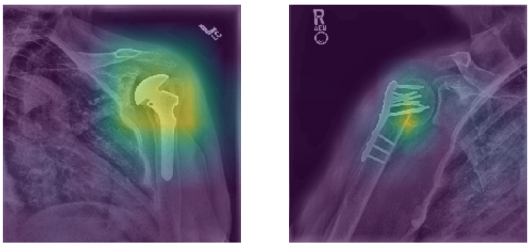

Figure 5.3: Grad-CAM visualization of detected abnormalities

best model from all folds was evaluated on my test set. I also implemented Grad-CAM for visualizing, explaining, the networks decisions. In Figure 5.3 there is one example of detected abnormality visualization for each of my models.

In Table 5.5 I compared my models to the MURA Paper baseline and best radiologist from the ones who evaluated the test set[19] and I also included the best model in the competition that was organized with the release of the dataset but is no longer open. The best model is an ensemble model. I have to state that the comparison may not be accurate as the test set is not public. Even though I tried to imitate it, I do not know study and label distribution. Also, I evaluated my models on individual images as opposed to full studies like in the MURA paper.

The results I obtained could be further improved by giving more attention to the architectures. For example experimenting with individual layer parameters, adding more layers on top of the base model or optimizing more hyperparameters for longer periods of time. The possibility to build an ensemble model, like it was also proposed in original MURA paper[19] could probably bring better results than those achieved by my experiments. When it comes to the dataset, a more complex segmentation, for example manual or automated markings of joints could help the CNN models to make better predictions.

Model	Cohen's kappa (κ)
MURA paper baseline	0.705
Best Radiologist - Stanford University	0.778
Leaderboard no.1	0.843
ResNet50	0.591
DenseNet169	0.683

Table 5.5: Results of my models compared to selected existing ones

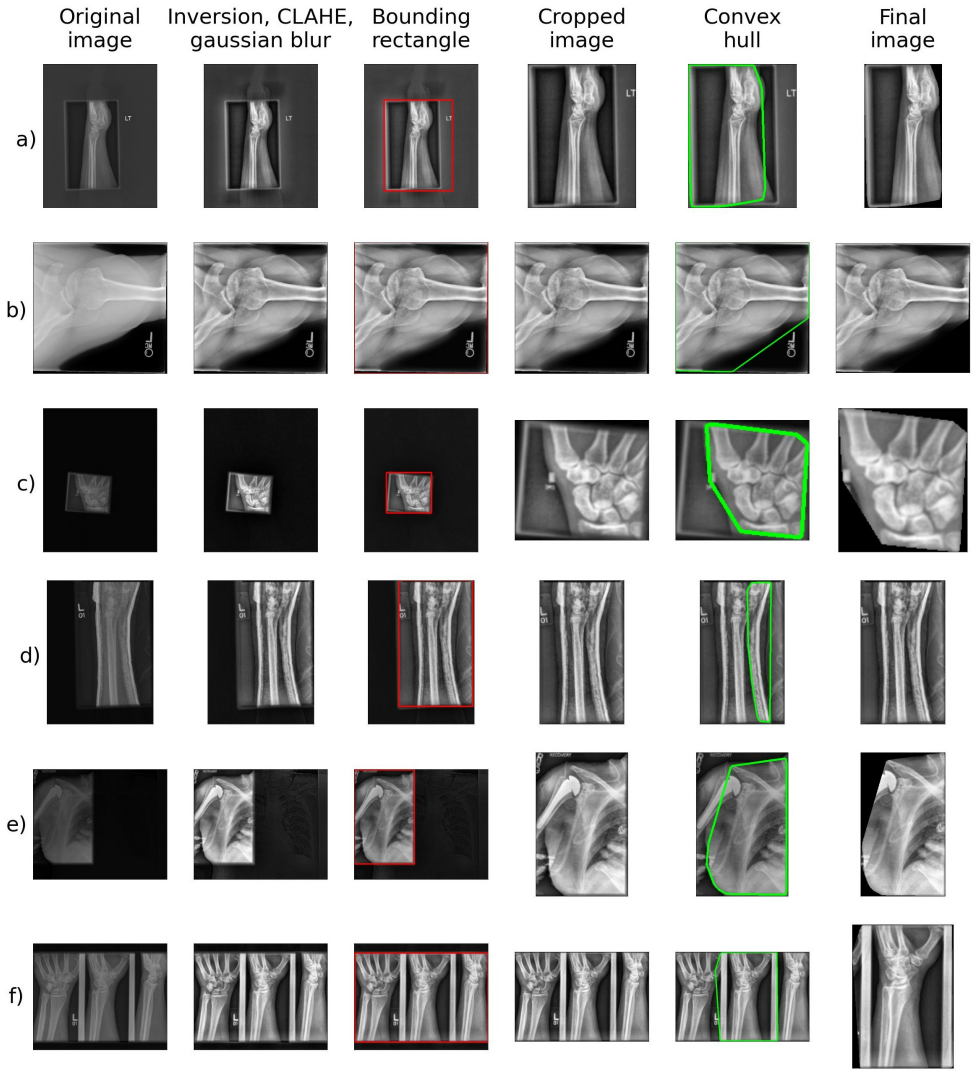

Figure 5.4: Visualisation of individual steps of my custom preprocessing cropping pipeline. Images a) and c) best represent the full functionality of the pipeline. Both images have their backgrounds removed, then the region of interest is found and images are successfully cropped again. Image b) is already positioned well by default, but has it's inscriptions removed. In image d) the region of interest is found but it is smaller than the defined threshold so the image is not cropped, which was a correct decision as the cropping would remove major part of the important area. Images e) and f) are examples of undesirable pipeline behavior. In image e) it excluded important portion of the image containing part of an abnormality. Image f) contains multiple scans of the same wrist and the pipeline detected only one and removed the others, which leads to loss of information.

Conclusion

The first part of the thesis shortly presents the fundamentals in machine learning and then focuses on individual techniques that can be used for medical imaging tasks. Some of them are selected and used in my implementation, which conducts experiments primarily on image preprocessing in order to improve model performance. The comparison of used methods is both addressed in their definition and in the experimentation part. Even though my results did not reach the scale of success achieved by existing models, they can still be consider fairly satisfiable.

All code was published on Github with comments and descriptions for better understanding of code structure in order for it to be reusable. It contains multiple interactive notebook for presenting and visualizing parts of the preprocessing steps.

More experiments could have been carried out, but the task of training CNN models is computationally expensive and more time would be needed to increase the scope of experiments. Possible improvements for future would definitely be ensemble models, adjusting of model architectures or detailed image segmentation.

Bibliography

1. RUSSELL, Stuart J.; NORVIG, Peter. Artificial Intelligence: A Modern Approach. In: 3rd ed. Prentice Hall, 2009, pp. 695–696. ISBN 0-13-604259-7.
2. HU, Fei; HAO, Qi. Intelligent sensor networks: the integration of sensor networks, signal processing and machine learning. In: Taylor Francis, 2012, pp. 16–17. ISBN 978-1-4398-9281-7. Available from DOI: 10.1201/b14300.
3. EDUCATION, IBM Cloud. *What is unsupervised learning?* [Online]. [N.d.] [visited on 2022-05-10]. Available from: https://www.ibm.com/cloud/learn/unsupervised-learning.
4. KAYYATH, Aatish. *Confusion Matrix : Let's clear this confusion*. Medium, 2021. Available also from: https://medium.com/@aatish_kayyath/confusion-matrix-lets-clear-this-confusion-4b0bc5a5983c.
5. HICKS, Steven A. et al. On evaluation metrics for medical applications of artificial intelligence. *medRxiv*. 2021. Available from DOI: 10.1101/2021.04.07.21254975.
6. NARKHEDE, Sarang. *Understanding AUC - ROC Curve*. Towards Data Science, 2018. Available also from: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.
7. BUI, Huy. *ROC Curve Transforms the Way We Look at a Classification Problem*. Towards Data Science, 2020. Available also from: https://towardsdatascience.com/a-simple-explanation-of-the-roc-curve-and-auc-64db32d75541.
8. XU, Yun; GOODACRE, Royston. On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of analysis and testing*. 2018, vol. 2, no. 3, pp. 249–262.

9. MURPHY, Kevin P. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. Available also from: probml.ai.

10. *Overfitting and underfitting*. [N.d.]. Available also from: https://www.educative.io/edpresso/overfitting-and-underfitting.

11. O'SHEA, Keiron; NASH, Ryan. *An Introduction to Convolutional Neural Networks*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1511.08458.

12. HEATON, Jeff. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning. *Genetic Programming and Evolvable Machines*. 2018. ISSN 1389-2576. Available from DOI: doi:10.1007/s10710-017-9314-z.

13. PIZER, Stephen M. et al. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*. 1987, vol. 39, no. 3, pp. 355–368. ISSN 0734-189X. Available from DOI: https://doi.org/10.1016/S0734-189X(87)80186-X.

14. ITSEEZ. *Histograms - 2: Histogram equalization* [online]. [N.d.] [visited on 2022-05-12]. Available from: https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html.

15. KOONSANIT, Kitti; THONGVIGITMANEE, Saowapak; PONGNAPANG, Napapong; THAJCHAYAPONG, Pairash. Image enhancement on digital x-ray images using N-CLAHE. In: 2017, pp. 1–4. Available from DOI: 10.1109/BMEiCON.2017.8229130.

16. LI, Lisha et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2016. Available from DOI: 10.48550/ARXIV.1603.06560.

17. DENG, Jia et al. ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. Available from DOI: 10.1109/CVPR.2009.5206848.

18. SAGI, Omer; ROKACH, Lior. Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*. 2018, vol. 8, no. 4, e1249. Available from DOI: https://doi.org/10.1002/widm.1249.

19. RAJPURKAR, Pranav et al. *MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs*. 2017. Available from arXiv: 1712.06957.

20. MONDOL, Tusher Chandra; IQBAL, Hasib; HASHEM, MMA. Deep CNN-Based Ensemble CADx Model for Musculoskeletal Abnormality Detection from Radiographs. In: *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*. 2019, pp. 392–397. Available from DOI: 10.1109/ICAEE48663.2019.8975455.

21. RAJPURKAR, Pranav et al. *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*. arXiv, 2017. Available from DOI: 10.48550/ARXIV.1711.05225.

22. medicalsegmentation.com. *COVID-19 CT Images Segmentation* [online]. [N.d.] [visited on 2022-05-12]. Available from: https://www.kaggle.com/competitions/covid-segmentation/data.

23. DING, Feng; LEOW, Wee Kheng; HOWE, Tet. Automatic Segmentation of Femur Bones in Anterior-Posterior Pelvis X-Ray Images. In: 2007, pp. 205–212. ISBN 978-3-540-74271-5. Available from DOI: 10.1007/978-3-540-74272-2_26.

24. UYSAL, Fatih et al. Classification of Shoulder X-ray Images with Deep Learning Ensemble Models. *Applied Sciences*. 2021, vol. 11, no. 6. ISSN 2076-3417. Available from DOI: 10.3390/app11062723.

25. DAVLETSHINA, Diana et al. *Unsupervised Anomaly Detection for X-Ray Images*. arXiv, 2020. Available from DOI: 10.48550/ARXIV.2001.10883.

26. GIEŁCZYK, Agata et al. Pre-processing methods in chest X-ray image classification. *PLOS ONE*. 2022, vol. 17, no. 4, pp. 1–11. Available from DOI: 10.1371/journal.pone.0265949.

27. WANG, Linda; LIN, Zhong Qiu; WONG, Alexander. COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images. *Scientific Reports*. 2020, vol. 10, no. 1, p. 19549. ISSN 2045-2322. Available from DOI: 10.1038/s41598-020-76550-z.

28. KERAS. *Keras documentation: About keras* [online]. [N.d.] [visited on 2022-04-27]. Available from: https://keras.io/about/.

29. HE, Kaiming et al. *Deep Residual Learning for Image Recognition*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1512.03385.

30. KAUSHIK, Aakash. *Understanding Resnet50 architecture* [online]. OpenGenus IQ: Computing Expertise, 2020 [visited on 2022-05-12]. Available from: https://iq.opengenus.org/resnet50-architecture/.

31. HUANG, Gao et al. *Densely Connected Convolutional Networks*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1608.06993.

32. PYTORCH. *Pytorch DenseNet* [online]. [N.d.] [visited on 2022-05-12]. Available from: https://pytorch.org/hub/pytorch_vision_densenet/.

33. SIM, Julius; WRIGHT, Chris C. The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements. *Physical Therapy*. 2005, vol. 85, no. 3, pp. 257–268. ISSN 0031-9023. Available from DOI: 10.1093/ptj/85.3.257.

34. MCHUGH, Mary. Interrater reliability: The kappa statistic. *Biochemia medica*. 2012, vol. 22, no. 3, pp. 276–282. Available also from: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/`.

35. OTSU, Nobuyuki. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, vol. 9, no. 1, pp. 62–66. Available from DOI: `10.1109/TSMC.1979.4310076`.

Appendix A

Acronyms

AHE Adaptive Histogram Equalization

AUC Area Under the Curve

CLAHE Contrast Limited Adaptive Histogram Equalization

Grad-CAM Gradient-weighted Class Activation Mapping

CNN Convolutional Neural Network

ML Machine Learning

MURA Musculoskeletal Radiographs

ROC Receiver Operator Characteristic

Appendix B

Contents of Enclosed CD

- `logs` ..... the directory with logged training runs
- `weights` ..... the directory of saved model weights
- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of implementation source codes
- `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format