Department of Informatics
Technical University of Munich

# TECHNICAL UNIVERSITY OF MUNICH

## DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

# Automatic Recognition of Crosses in Digitalized Documents

Julian Villing

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

# Automatic Recognition of Crosses in Digitalized Documents

# Automatische Kreuzerkennung in Digitalisierten Dokumenten

| | |
|---|---|
| Author: | Julian Villing |
| Supervisor: | Prof. Dr.-Ing. Georg Carle |
| Advisor: | Benedikt Jaeger, M. Sc. |
| | Dr.-Ing. Stephan M. Günther |
| Date: | April 15, 2022 |

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, April 15, 2022

Location, Date

Signature

## Abstract

Cross recognition is a very important part of form digitalization. Automating it saves time and effort necessary to evaluate which boxes have been selected. The saved time can be used to focus on more difficult tasks.

In general, image recognition is a task that has been solved many times. However, for this specific task with many individual inputs, the existing approaches are either not accurate enough or too complex. As we are only working with small images and few possible results, there are simpler solutions to this task.

Our solution to solving this problem is a modular pipeline. We can train and evaluate different approaches each of which can use a completely different technique. Additionally, we can compute metrics, compare potential solutions, and identify the ideal way to solve the task. Finally, this modular pipeline can be extended in the future to support new classification methods.

In this thesis, we research which techniques can reliably and precisely identify crosses. We evaluate, optimize and compare them to each other. We propose a modular pipeline for automated training and evaluation of different approaches. An ideal solution for this classification task is identified.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Digitalization is an important process during which analog things are converted to their digital counterparts. A part of this digitalization process is the processing and conversion of forms. This thesis focuses on the evaluation of crosses within such digitalized forms. For our research, we will test many different approaches, both with and without machine learning. The results will be compared to each other to determine which solution provides the best results. Furthermore, metrics will be evaluated which allows for a more in-depth comparison.

All of our approaches take images containing boxes as input and classify their contents. This evaluation needs to be fast and accurate while the accuracy is much more important than raw speed. The importance of accuracy is given by the use case: evaluating small batches of images at a time, but a huge number of images in total. To simplify the usage of different frameworks and algorithm types, a pipeline should be implemented. This helps us to apply necessary preprocessing to the images adjusted for each approach. It also abstracts from the implementation details of individual frameworks by creating modular, exchangeable components. This allows us to effectively use, evaluate, and compare different approaches. Offloading the classification of crosses allows us to spend our limited time on more important parts of the digitalization process.

## 1.1 Motivation

Our pipeline should be used as a research platform for the development and research of cross-evaluation approaches. It can be used to compare new approaches to existing ones and also evaluate specific metrics such as memory consumption, runtime, and accuracy. With that information, the approach that fits the current requirements the

best can be identified. By evaluating multiple metrics it is possible to easily adapt to different requirements. Furthermore, existing approaches can be modified, updated, and compared to a baseline. As such, future improvements on existing technologies or even new ones can be integrated into the pipeline. After that, they can be compared to previous approaches to determine if the new developments provide benefits to our problem.

## 1.2 Caveats

While there are a lot of things we can do with our pipeline, we also need to be aware of potential problems. We can prevent some but not all of those. The first issue we might encounter is errors introduced by the algorithms we use. Using the evaluation it is possible to identify and resolve them by adjusting the model used in the pipeline. The other potential issue is human error. Incorrectly labeled input data is an example of this type of error. We need to be aware that our pipeline is not able to detect it and therefore we are not able to recover from it. There are two options for how we can work around this issue. We can either accept that the number of errors is very small and as such does not influence our results too much and ignore it. Alternatively, we can use other methods to try to get rid of the error. This can be achieved by either manually rechecking the labeled data, adjusting it when a mistake is discovered, or using an approach to automatically identify incorrect associations.

## 1.3 Research Questions

*Q1:* Which approaches are suitable for the automatic recognition of crosses?

*Q2:* How large does the input space need to be to produce accurate results?

*Q3:* What is the necessary size for a network to produce accurate results?

## 1.4 Outline

Relevant background information is given in Chapter 2. Chapter 3 explains how the data is processed and the pipeline is built. The dataset is presented and analyzed in Chapter 4. The implementations of the different approaches are covered in Chapter 5 and the corresponding results are presented in Chapter 6. Chapter 7 covers other research related to this thesis and why it does not solve the task adequately. Finally, Chapter 8 summarizes the results of this thesis.

# Chapter 2

# Background

To reliably recognize crosses, research on available technologies is necessary. These can then be used to implement different approaches which then solve the initial task. The technologies we can use can be grouped by whether they are based on machine learning. To determine the performance of an approach and to compare it to other approaches, performance metrics are required. The selection of metrics we use for evaluation are listed in Section 2.1. Section 2.2 explains the background on machine learning-based classification technologies. Recognition using computer vision is covered in Section 2.3.

## 2.1 Performance Measurement

Performance metrics are needed to determine how well a model solves the given task. These metrics aggregate the classification results in a single value. The results are expressed using four distinct values: true positive and negative as well as false-positive and negative. Those numbers are computed for every type of box. The number of images correctly classified as a given label is the true positive. Accordingly, the number of images that are correctly classified to be different from a specific label is the true negative. Inputs that are identified as the given label but are from a different label are considered false positives. Finally, the number of predictions that should be the current label but are not is the number of false negatives.

Using those values, we can define metrics that can be used to assess the performance of the evaluated approach. Accuracy is the first of four metrics we use for measurement. It relates the number of correct predictions to the total number of predictions. Therefore, the accuracy computes the probability that the prediction is correct. As this metric considers all predictions and only the total number of mistakes is considered, a high

error rate in a small class only has a small influence on the final result. Therefore, this metric favors larger classes over smaller ones.

The second metric is precision which correlates the number of true positives to all positive results. It models the probability that an image classified as a given target label is this label. This metric is appropriate is false positives cause a severe problem. In this case, missing a correct input is better than incorrectly identifying a wrong one. A good use case would be the recognition of edible plants. Missing fruits we could consume is far better than mistakenly eating inedible crops. As only predictions for a given label are used in the computation, this metric is influenced by other classes. This metric is likely to be lower for smaller classes due to larger classes being more likely to cause false positives. However, small classes, even if only being false positives of a larger class have a much smaller effect on the final result.

Recall describes the likelihood of an image being misclassified. Maximizing this metric reduces the probability of false negatives. For scenarios where a correct result being missed is the bigger problem than a wrong result being identified, this metric should be considered. Recall would be a good choice to evaluate a dangerous situation as we never want to underestimate it. In this case, overestimating the severity could serve as a protection.

The final metric is the F1-Score which is an averaged combination of both precision and recall. It therefore aims to minimize both false positives and false negatives. Compared to accuracy, true negatives are not considered. Furthermore, due to its construction, the F1-Score is more sensitive to misclassifications as well. The formulas for the different metrics are shown below.

**Accuracy** $\frac{TP+TN}{TP+FP+TN+FN}$

**Precision** $\frac{TP}{TP+FP}$

**Recall** $\frac{TP}{TP+FN}$

**F1-Score** $\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$

## 2.2 Machine Learning

Machine learning and especially Convolutional Neural Networks (CNNs) have been used for image classification and have proven to achieve good results. This has been shown by Simoyan and Zisserman using their VGG network in [1] as well as LeCun et al. using their LeNet in [2]. Furthermore, He et al. achieved good results with their ResNet in [3]. All of those models are CNNs used for image classification. LeCuns' research achieved

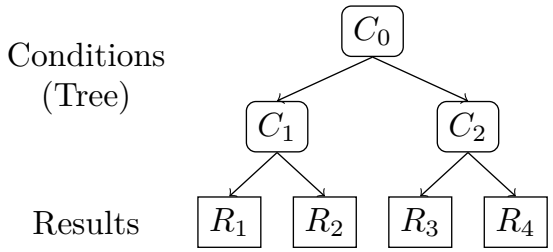



Figure 2.1: Simple example for a decisiontree

good results for the identification of handwritten digits is explained in more detail in Section 7.2.

Therefore, the state-of-the-art approach for image recognition-based tasks is to make use of machine learning. This topic aggregates approaches that automatically learn to solve the task given enough input data. In general, they identify features needed to differentiate between different classes on their own without the need for external knowledge.

In the next sections, selected approaches to classify crosses are presented. Decision forests are introduced in Section 2.2.1. Section 2.2.2 covers support vector machines and neural networks are explained in Section 2.2.3.

### 2.2.1 Decision Forests

Decision forests consist of multiple decision trees. The decision made by the forest corresponds to the most common decision made by each tree. The general architecture of decision trees is explained by Lewis in [4]. Each tree consists of several layers whose conditions branch into the next layer. The final layer does not have a decision but contains the choice made for every possible path. Every condition is based on a feature and a specific threshold. If the value of the feature is lower than the threshold, the evaluation descends into the left child and otherwise into the right child until a decision is formed. Each leaf node of the tree contains the decision made based on the previous conditions. A visualization is presented in Figure 2.1. This tree is made up of two layers and a total of three conditions. The quality of the decision depends on the quality of the provided features. It is therefore useful to extract relevant features from data in advance. Examples of such features could be the edges extracted from the image and the number of those. Other potential information could be provided in the form of the brightness of different regions of the image. In our context, this could be a subset of an image where we would suspect the marking of a cross to make a difference in perceived brightness. The forest is trained by creating trees for features and thresholds. The theoretical advantage of using a forest instead of a single tree is that overfitting is prevented as

the decision is based on the majority vote of all trees. Therefore, some wrong decisions caused by overfitting do not necessarily result in a wrong prediction. At the same time, many trees are necessary to compensate for overfitted trees. There are multiple kinds of trees or forests. When using Classification and Regression Trees (CART), a single tree is initially grown and shrunk afterward. Random forests use independent CARTs without shrinking. They were researched by Ho in [5]. As explained in their research, a benefit of their approach is that every individual tree is trained on a subset of the input data. Therefore, each of them is trained on different features. As multiple trees are used together, an individual tree does not need to identify every possible input correctly, which allows them to be smaller and therefore less complex. Contrary, gradient boosted trees also use CARTs but in this case, they are dependent on their predecessor.

### 2.2.2 Support Vector Machines

Support Vector Machines (SVMs) aims to group related inputs and separate them using support vectors. The mathematical concept is explained by Durgesh and Lehka in [6] and summarized in the following. Support vectors are computed so that they have the greatest distance to any given chunk. After these have been established, predictions are made by assigning individual inputs to any of the created groups. Support vectors are a function with parameters matching or exceeding the dimension of the input space. The extended number of dimensions might be necessary to be able to define a function in a way that a separation can be achieved. A large number of dimensions, however, especially more than that of the input space, gets increasingly expensive to compute.

A visualization is presented in Figure 2.2. The data points in this example are denoted as squares and triangles depending on the class they are associated with. Three possible distinct support vectors which can be used to separate the data are shown in blue. While the dotted and dashed lines also separate the two groups, the regular line is the best as the minimum distance from the line is the greatest.

### 2.2.3 Neural Networks

In machine learning, Neural Networks (NNs) aim to mimic the ability of our brain to recognize patterns, called features, and use them to solve complex tasks. The design of NNs is explained by Dongare et al. in [7]. They are made up of neurons that are grouped into layers. This creates several layers whose neurons are connected to all neurons of the previous layer. The value of a neuron is equal to the sum of the values of each input neuron multiplied by its weight. An example for a simple neural network is shown in Figure 2.3. It consists of three input values, eight neurons split across two hidden layers and two output values. All adjacent layers are fully-connected.

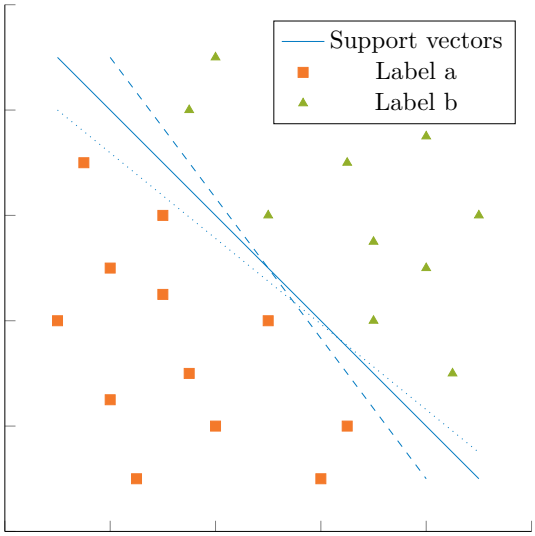



Figure 2.2: Different support vectors separating two groups of inputs, adapted from [6]

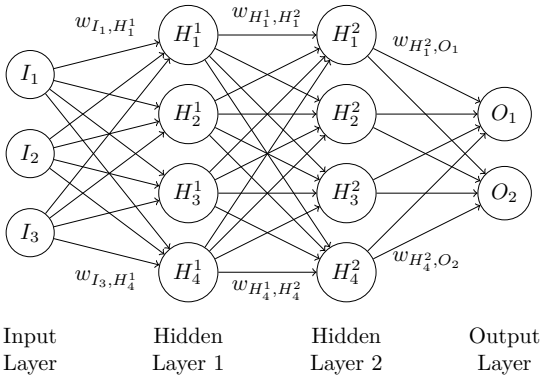



Figure 2.3: Simple example for a neural network

This relationship can be expessed as a matrix multiplication. The general formula is shown in Equation (2.1).

$$\begin{pmatrix} I_1 \\ \vdots \\ I_n \end{pmatrix}^T \cdot \begin{pmatrix} w_{I_1,O_1} & \dots & w_{I_1,O_m} \\ \vdots & \ddots & \vdots \\ w_{I_n,O_1} & \dots & w_{I_n,O_m} \end{pmatrix} = \begin{pmatrix} O_1 \\ \vdots \\ O_m \end{pmatrix}^T \tag{2.1}$$

Repeated applications represent a neural network with multiple fully-connected layers.

The values of the neurons in the input layer match the input data and the output data corresponds to the values of the neurons in the output layer. Therefore, the values

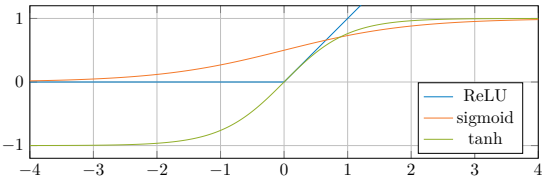



Figure 2.4: Visualization of the different activation functions

within the matrices affect how the outputs are derived from the inputs. By adjusting them, the effect of a combination of inputs on the outputs can be adjusted.

The importance of activation functions is explained by Sharma et al. in [8]. According to their research, they are required to allow the network to recognize non-linear features. Without them, the neural network could be represented as a "simple linear function". This limits the network's capability to be able to recognize complex patterns.

A very important activation function is the Rectified Linear Unit (ReLU). It maps negative values to zero and does not alter non-negative values. Apart from introducing non-linearity, this specific function has the property that not every neuron is activated at all times as they are zero. As a result, only a subset of the whole network is present in the final function. The optimization of the networks' weights during training is therefore simplified. Other possible non-linear activations are the hyperbolic tangent and sigmoid functions. Both of these functions clamp the value between negative one and one or zero and one respectively. These functions are plotted in Figure 2.4.

To improve the predictions of the neural network, training is necessary. During this process, the weights are adjusted. This is achieved using an optimizer in combination with a loss function. This function evaluates the error which is the deviation between the actual and the expected values. During training, the weights are adjusted by the optimizer so that the loss is minimized. The learning rate determines the speed of the training procedure. A higher learning rate generally converges faster while a lower value results in a more precise network.

Both of those advantages can be combined by making use of decay which was researched by You et al. in [9]. They explain that a large learning rate "avoids fitting noisy data" and by using decay, the network "learns more complex pattern". Decay enables us to decrease the learning rate as training continues. We are therefore able to start with a larger learning rate to quickly get towards an optimum. Afterward, the learning rate is decreased, allowing the optimizer to find the best weights possible.

Another important property is dropout which was researched by Srivastava et al. in [10]. Using this technique helps to mitigate overfitting. Dropout defines the probability a

neurons' value is set to zero during training. It is applied after fully connected layers in the network. Similar to the ReLU activation function, it deactivates some neurons which allows the optimizer to tune smaller parts of the network at a time. This functionality is only used during training and evaluation happens as if the dropout layers were absent.

An important technique for image recognition is the use of convolutional layers according to the research from O'Shea and Nash in [11]. They explained that the use of convolutional layers helps in recognizing distinct patterns while reducing the number of weights at the same time. These layers are then followed by pooling layers, which can reduce the dimension to 25 % of the original size. Convolutions make use of small kernels which also need to be trained. Afterward, they extract specific features. Furthermore, as these layers are not fully-connected, the network is simplified by reducing the total number of weights

A task that can be solved using NNs is the recognition of handwritten digits and is covered in more detail in Section 7.2. Solutions to that problem have been researched by LeCun et al. in [2]. As a quick summary, they identified that good results were achieved using networks with one or two hidden layers and between 300 and 1000 neurons. Many different networks have been evaluated and the accuracy ranged from 95.3 % to 99.3 % depending on the specific network architecture in use.

## 2.3 Computer Vision

Compared to machine learning, computer vision covers approaches that aim to classify images algorithmically. This could be the identification of lines, shapes, or patterns. Recognizing those within the image can then be used to find and extract the relevant content. The output can then be processed in whichever way is desired. For example, black and white dots could be extracted based on their shape and color. They could then be interpreted as ones and zeros to form a stream of binary digits.

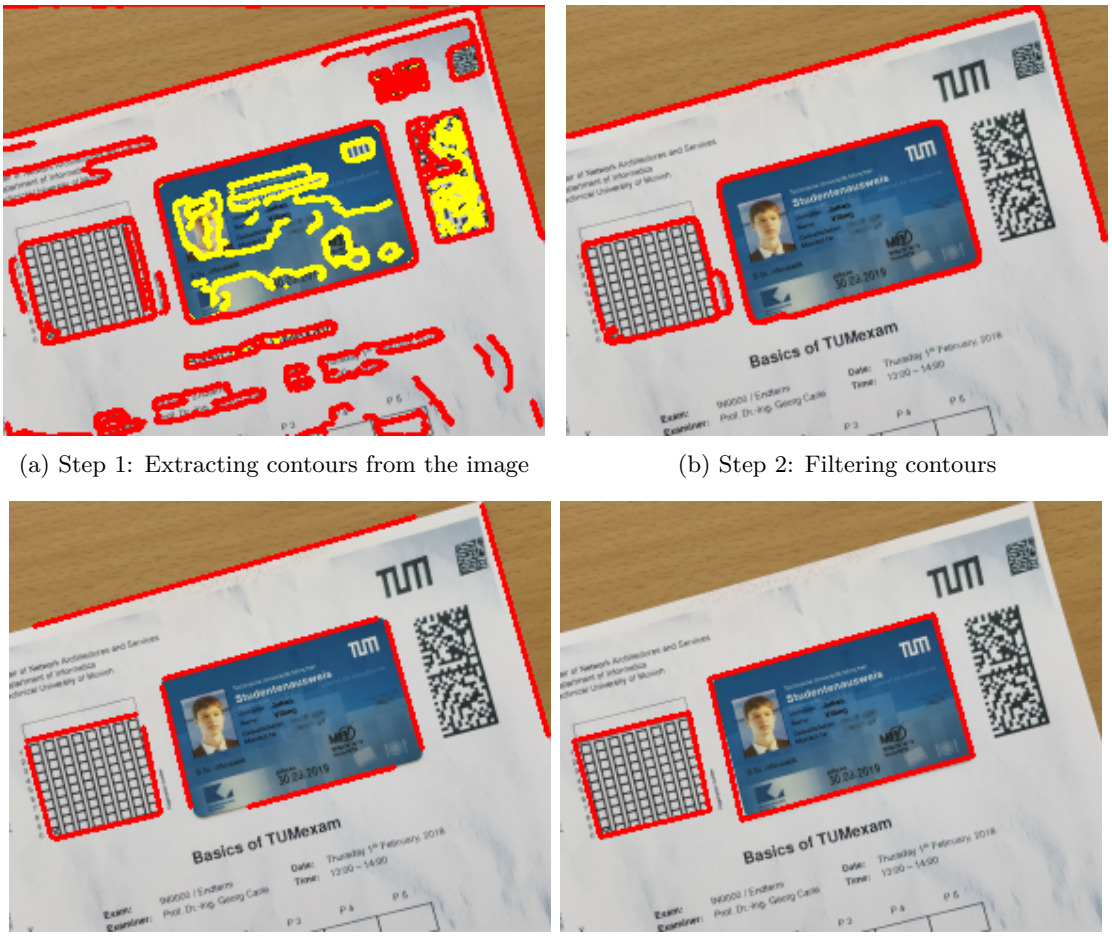
A possible use case for computer vision is the extraction of id cards and registration number boxes from an exam coverpage. This application is explained by Villing in [12]. The general procedure consists of four steps. Before any extraction is performed, the image is preprocessed using multiple filters. To amplify shadows, histogram equalization is performed. This process is necessary so that later steps retain information about shadows that would otherwise be removed. They are needed to identify the borders of the id card. The next preprocessing step is to apply blurring to the image. The effect is that slight color differences, which would otherwise be recognized as edges, are eliminated. This is also the reason why the shadows needed to be amplified. Bilateral

filtering further reduces the noise in the image. Together, these steps remove unwanted information which makes the extraction much more difficult.

The first step is to extract contours from the images using edge detection. As a lot of contours are extracted, they need to be filtered in multiple ways. The results of this step are visualized in Figure 2.5a. There is still a lot of unnecessary information in this picture which we need to get rid of. Contours contained within other contours are not needed and can be safely removed. Furthermore, contours whose shape differs too much from the known shape of the registration number box of the id card are removed as well. This cleanup is shown in Figure 2.5b. Lines are extracted from the remaining contours to filter out noise. This step additionally hides small, unwanted deviations which differ from the approximated line. The remaining lines are presented in Figure 2.5c. Finally, for every contour that resulted in four lines, a rectangle based on these lines is constructed. These boxes should now contain the relevant information very precisely and are visualized in Figure 2.5d. We are now able to extract the information we are interested in using a perspective transform. This method additionally gets rid of skewing introduced by the angle the original photo was taken at. The last postprocessing step is to orient the images correctly. This is done using the relative position of both images to each other as well as their aspect ratio. The latter limits the rotations to two possible outcomes and the former allows us to align them correctly.

Another option is to identify patterns within the image and use that information to determine what is present in it. This can be used to identify markers or codes in an image and extract information from them. As we know the position of those codes we can extract the information as before. This approach uses fewer steps and is also able to fix skewed images.

The difference to machine learning is that the extraction procedure must be implemented manually. Compared to NNs, we cannot just feed the optimizer our labeled data and automatically get results. Furthermore, we need to identify the relevant features ourselves. This could be the markers, contours, or aspect ratio from the examples before. After the extraction succeeded, we still need to interpret the data. This could be extracting the name from the id card using Optical Character Recognition (OCR) or to identify which boxes are selected in the registration number box in Figure 2.5.




(a) Step 1: Extracting contours from the image

(b) Step 2: Filtering contours

(c) Step 3: Extracting lines from countours

(d) Step 4: Constructring rectangles from lines

Figure 2.5: Object recognition performend using computer vision, taken from [12]

# Chapter 3

# Design

The goal of this thesis is to create a modular pipeline for the evaluation of crosses. Using this pipeline, it should be possible to quickly evaluate and compare different modeling approaches. It should also be possible to apply individual preprocessing to the data before it is passed to each model. The actual preprocessing used depends on the algorithm in use. For NNs, preprocessing is generally not needed but SVMs for example benefits from extracting features. Preprocessing can be used to extract additional features from this pixel data if applicable. Examples of additional features are edges extracted from the image. Other possible use cases include applying filters such as turning a grayscale image into a black and white image or computing the brightness of the image. For every input image, the pipeline should return the predicted class of that image. To achieve this flexibility, three main parts are needed.

- The first component is needed to convert images of arbitrary size and color space into standardized images. This means that the resulting images all have the same dimensions and channel count. In our case, we have decided that they should be grayscale and 14 by 14 pixels in size. This serves two purposes, firstly all images are the same size and color space, and secondly, the exact format of the input data is known and can be used when building the model. Ensuring that the input data has a consistent format is especially important for neural networks.
- The second step is a mechanism to apply necessary preprocessing ahead of time using the input images. By doing so we can transform the input data once and reuse it for multiple models. Additionally, we can easily exchange the implementations to compare their influence on the result. Abstracting this step away from the model also allows us to execute it ahead of time.

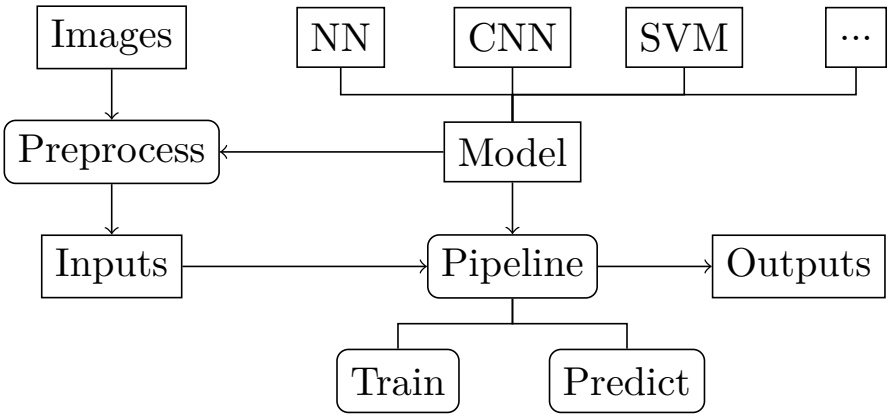



Figure 3.1: A architecture of the pipeline

- The last part is the modular pipeline which takes those images, passes them through each model, and trains it or predicts the classifications depending on the context.

Only the model and optionally the preprocessor need to be supplied by the user.

Creating the aforementioned standardized images is quite straightforward. We simply need to read the images from the disk, convert them to be black and white, and scale them so that they match our specification. The simplest form of preprocessing is the identity which does not affect images at all. It is the default when no preprocessing is necessary. Otherwise, we need to create a function to apply the necessary transformations. For the pipeline, we need a model which can be trained on those images as well as predict their results. The actual implementations for training and predictions are mostly handled by dedicated frameworks such as tensorflow or sklearn. In that case, only a layer of abstraction is necessary so that the interface to the model stays the same independently from the framework being used.

For an already implemented model, the implementation of the pipeline consists of putting those pieces together. We need to load the images and preprocess them if we did not do that ahead of time. The preprocessed data is then passed into the model which does the training or predicting. A visualization of the process is shown in Figure 3.1. The correct preprocessor is chosen by the model which affects the inputs. The chosen pipeline mode affects the output which is a trained model after training and predictions after predicting. An visualization of the pipeline is shown in Figure 3.2.

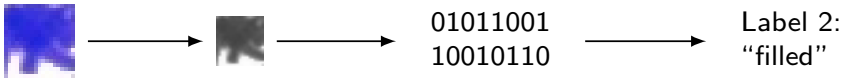



FIGURE 3.2: Visualization of the pipeline for a single image

# Chapter 4

# Dataset

As our dataset, we use data acquired from TUMexam. This has important advantages over creating that data ourselves. First of all, TUMexam has prelabeled real-world data which is better than artificially created data. Generating data on our own comes with the risk of creating perfect data. While precautions can be taken to include variation during generation, we might not think of all possible edge cases. Examples of such special artifacts could be the use of a pen with a very thin or thick stroke width.

We might only use a single pen, a single stroke width, and a single opacity while creating the images. In real data, however, we expect crosses marked by different people to differ from each other. As a result, a model might only be good at recognizing our artificial data but does not perform well on real data.

Those differences are important during the training procedure. Without them, we would not be able to reliably identify all kinds of crosses encountered in real data. Furthermore, we do not need to label the data. Our data source, TUMexam, is introduced in Section 4.1. The distribution of the crosses is analyzed in Section 4.2 and the thresholds are inspected in Section 4.3.

## 4.1 TUMexam

TUMexam is an examination software for both physical and remote examinations. It supports many tasks related to examination such as seating assignments or attendee control. Furthermore, automatic scanning, digitalization, and evaluation of exams are covered. During a scan, boxes in the document are cropped and classified. The current system can serve as a baseline and at the same time, a huge number of crosses are

| **Label** | **Absolute** | **Relative** [%] |
|---|---|---|
| Empty | 2.45 M | 78.89 |
| Selected | 639 k | 20.61 |
| Filled | 15.4 k | 0.5 |

Table 4.1: The distribution of images across different classes

available at our disposal. A visualization of these boxes can be seen in Figure 4.1. We can make use of those crosses instead of labeling data manually. This allows us to skip a tedious process and at the same time ensures that the crosses are not biased by any means, e.g. by using the same handwriting or pen.

## 4.2 Distribution

As we have labeled data from TUMexam, we want to analyze and visualize it. This helps us to better understand what we are working with. Interesting features include the distribution of the images across the different classes as well as the threshold to which an image is filled. Further interesting information we can acquire is the variation of those images of the same class. This could help us to improve our approaches. The first property of the dataset that is analyzed is the number of images for each class. The results are displayed in Table 4.1. It is somewhat expected that filled crosses only make up a very small portion of our data as human error when selecting crosses is rather unlikely. It is also expected that empty crosses have by far the highest share of the dataset as generally, only a few crosses are expected to be filled. For example, when selecting the number of points a student got on a subproblem, only one value needs to be selected and all other options do not. In TUMexam, students can enter their eight-digit registration number in a matrix of crosses which is used to make that number easily machine-readable. As example is shown in Figure 4.2. As each place of the number has ten options but only one will ever be used we immediately get a ratio of nine out of ten crosses being empty. When a mistake is made and another cross was meant to be selected, one additional cross is marked and the ratio decreases to eight out

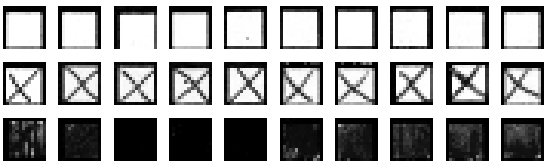


Figure 4.1: A visualization of the TUMexam dataset. From top to bottom: Crosses being empty, selected, and filled

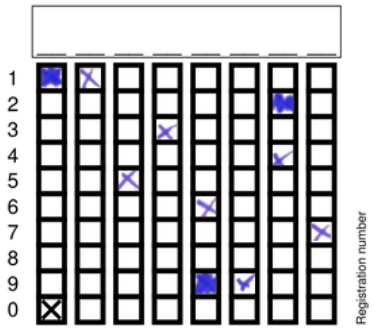



Figure 4.2: A registration number box representing the arbitrary registration number 01536947

of ten empty crosses. In either case, most of them are completely white. A similar effect appears with multiple choice questions where the majority of answers are incorrect and as a result, a majority of the crosses should be empty. As a result of both of those constraints, we expect there to be many more empty crosses than selected or filled ones.

This imbalanced distribution of images across the three possible classes is expected. We do expect to get mostly empty crosses and correctly identifying them has the single greatest impact on the final accuracy. Therefore, we also take other metrics into account. Filled images only make up 0.5 % of all images in the dataset. To improve the precision of our approaches for this label, we need to increase the relative share of those images. To mitigate a disproportionally high error rate for labels with few occurrences resulting from this imbalance, we can increase the number of images. There are two main ways to achieve this. The first is to reduce the number of other images by removing them from the training data. By reducing the number of images from a common label, we are increasing the relative amount of images for rarer labels.

Our other option is to generate more images of the rare cases. This duplication can be achieved either by augmenting already existing data or by manually creating new data. For augmentation, we can take images and rotate and flip them. For all possible combinations of 90-degree rotations and horizontal and vertical flips, we reach a total of twelve unique combinations, including the original data.

Using those techniques, we can create an augmented dataset with equal amounts of images for each label. This can be achieved by reducing all classes to the size of the smallest class and then augmenting all of those. The final distribution is shown in Table 4.2a.

If we do not want to alter the dataset this much, we can instead opt to remove fewer images from the dataset. In this case, the relative share of filled boxes not increased as much. For this approach, we reduce the size of the largest class to the size of the

| Label | Amount | Share [%] |
|---|---|---|
| empty | 148 k | 33.3 |
| selected | 148 k | 33.3 |
| filled | 148 k | 33.3 |

(a) Result from shrinking and augmenting all classes

| Label | Amount | Share [%] |
|---|---|---|
| empty | 511 k | 43.7 |
| selected | 511 k | 43.7 |
| filled | 148 k | 12.7 |

(b) Result from shrinking the largest and augmenting the smallest class

Table 4.2: The new datasets after increasing the relative share of filled crosses

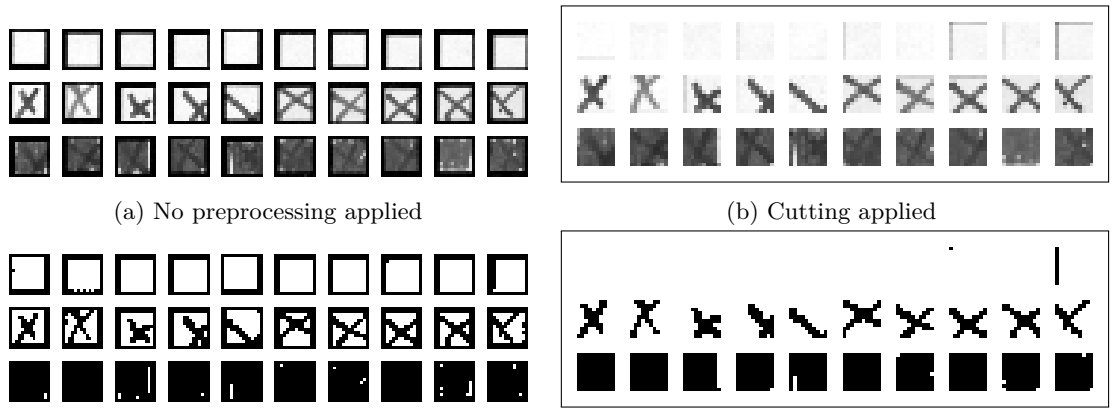
second largest. Furthermore, we augment the smallest class as before by the aforementioned factor of twelve. The new dataset resulting from these operations can be seen in Table 4.2b.

These modifications are only applied to the training part of the dataset but not to the prediction part. Using those new datasets, we can check if they provide better results compared to the original dataset. According to the research from Masko and Hensman in [13], an imbalanced dataset can have a "severely negative impact on overall performance". From their experiments, a single class being overrepresented resulted in the worst performance. They concluded that both the removal of images from an exceptionally large class as well as generation of artificial data from a small class can improve the performance of the CNN. Our modified datasets should therefore both improve the overall accuracy as well as the accuracy of the relatively rare class, filled crosses, in the dataset.

## 4.3 Threshold analysis

After we have analyzed the distribution of our labeled data on the three classes, we are now interested in the threshold to which a box is filled. This analysis might make it possible to quickly and reliably identify the different types of crosses. Furthermore, we could get better insights into the structure of the data. However, the raw data might not be ideal for this analysis due to noise and superfluous information. Examples of potentially unwanted information could be a border that is present on any given image.

To improve the results, we can transform the images by applying preprocessing steps to them. We have two basic ways to preprocess those images at our disposal: cleaning and cutting. The first turns the grayscale image into a black and white image by setting each pixel to black if it is below a certain limit and to white otherwise. This turns gray images black, allowing for an easier distinction. When tweaked correctly this can be used to separate the thresholds of the different classes. The latter is used to remove the black border of the boxes which are present in every image and therefore do not provide



(a) No preprocessing applied

(b) Cutting applied

(c) Cleaning applied

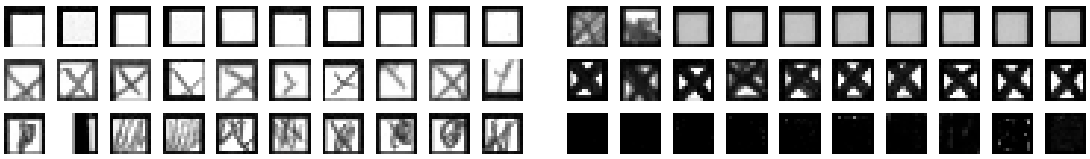
(d) Cutting and cleaning applied

Figure 4.3: A visualization of the different preprocessing optionns

meaningful information to us. As such we can remove this redundant information which reduces the threshold while also separating them from each other. In this case, empty crosses are brighter than they were with the border as that was the only black part in the image. The available preprocessing options are visualized in Figure 4.3. Note that the images to which cleaning has been applied cannot contain gray pixels. They have one of two possible values, either completely white or black. This severly limits the possible number of unique images and therefore thresholds, because the range of available colors is now much more limited after this preprocessing. Reducing the amount of possible values slightly affects visualization but is not a problem during the computations.

Besides preprocessing, we are interested in how the images look. Based on different lighting conditions or stroke widths, equally labeled images might vary greatly from each other. We expect a cross created with a pencil to be brighter than a cross drawn using a regular pen. Such examples are shown in Figure 4.4. Interestingly, the dataset contains empty boxes which appear gray even though they should be white. This could have been caused by an incorrect white balance of a scanner or a shadow being on the document while a photo was taken. In either case, such special cases need to be accounted for during the analysis. A similar difference can be identified for filled boxes. While the dark examples are virtually black, the bright examples contain a lot of white spots. As a result, we expect a large range of thresholds for a single class. Furthermore, those examples have thresholds outside of the expected range. This probably causes mistakes when classification is based solely on the threshold.

For each of the preprocessing steps, we can compute the thresholds for every image. This data can be visualized using a histogram as well as a cumulative distribution function showing the absolute and relative number of images per threshold. Our initial



(a) Bright examples from the dataset (b) Dark examples from the dataset

Figure 4.4: A visualization of the variety of the dataset

| Preprocessing | Lower Boundary | Upper Boundary | Accuracy |
|---|---|---|---|
| None | 0.43 | 0.69 | 99.603 % |
| Cutting | 0.15 | 0.57 | 99.378 % |
| Cleaning | 0.51 | 0.91 | 99.859 % |
| Cleaning & Cutting | 0.20 | 0.86 | 99.840 % |

Table 4.3: Accuracy when applied different preprocessings

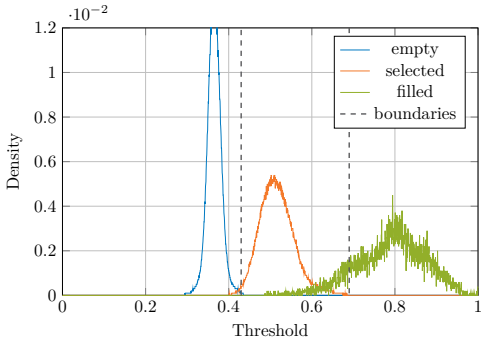
assumption, that the threshold tends to increase from empty over selected to filled boxes, can be verified by looking at these plots. As our goal is to classify the images based on their threshold, we compute two ideal limits that separate different classes from each other. Thresholds are strictly increasing for labels empty, selected, and filled. Therefore, we can compute the lower boundary between the labels empty and selected as well as the upper boundary between selected and filled and use it for identification. Images with a threshold below the lower limit are therefore considered empty, images in-between both limits are assumed to be selected, and the remaining boxes are treated as filled. To determine these boundaries, we identify the threshold with the minimal overlap. This is the value where the sum of selected crosses below the lower boundary and empty crosses above is minimized. The same principle applies to the upper boundary. Simply evaluating the overlap for both cases for every bin in the histogram yields the following results. The exact values for the determined boundaries are listed in Table 4.3. The preprocessing steps have a huge influence on the threshold distribution and therefore also on the boundaries. The distribution of thresholds for each class and preprocessing step is presented in Figure 4.5. Computed boundaries are visualized as vertical, dashed lines. We see that the curves overlap and the boundaries are therefore not able to perfectly separate the different kinds of images. This means that a classification based solely on the threshold is not possible.

As we can see from the images and the graphs, removing the border of the images by cropping them increases the spread of the thresholds. Cleaning the images increases the occurrences towards the lowest and highest thresholds while reducing them towards the middle. While this preprocessing step has a similar effect as the first preprocessing

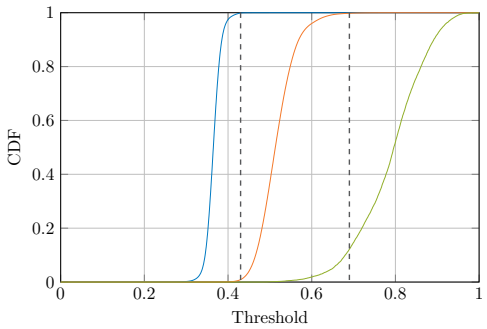
step, its results are more extreme as empty boxes now have occurrences, although only a few, at thresholds above 0.6. That might be an issue for approaches based on computer vision. The overall trend is that applying preprocessing spreads the occurrences of the different classes, making it easier to find a threshold that separates images with different labels. This threshold analysis will be used as the basis for a computer vision-based classification approach.

To allow the histograms to focus on the most important aspects, the vertical axis is limited to remove extreme spikes. With no preprocessing applied, as shown in Figure 4.5a, empty boxes have a spike at 0.365 with a value of 0.014. After the images have been cleaned, empty boxes have three spikes are at 0.382, 0.438, and 0.489 with the values 0.113, 0.313, and 0.128, respectivly. Additionally filled boxes have a spike at 0.999 of 0.580. This is shown in Figure 4.5c. Cutting produces a single spike at 0.001 of 0.109 as visualized in Figure 4.5e. Finally, with both preprocessing methods applied, the following spikes occur as presented in Figure 4.5g. Empty boxes feature two extreme values at 0.000 and 0.099 with the values 0.601 and 0.274. There is also another spike for filled boxes at 0.999 of 0.634.

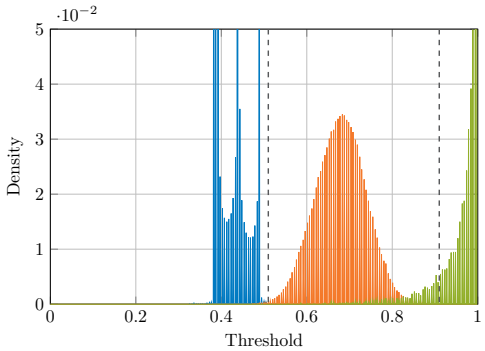
Figures 4.5a, c, e, and g display the histogram for each class while Figures 4.5b, d, f, and h display the corresponding cumulative distribution function. Taking a close look at Figures 4.5c, d, g, and h we can see that those graphs are not normal lines. The spikes, which create a striped look, are a result from the way we preprocessed the images. As cleaning has only two distinct values for each pixel, the total number of possible combinations is limited. Therefore, the thresholds are from a list of discrete values which is visualized as a series of spikes in the histogram. The preceding CDFs also suffer from the same issue and consist of a series of horizontal and vertical lines instead of a smooth curve.



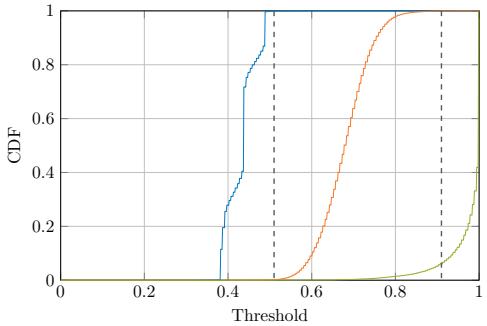

(a) Histogram, no preprocessing applied



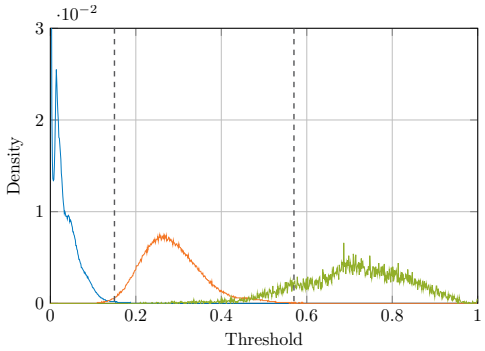

(b) CDF, no preprocessing applied



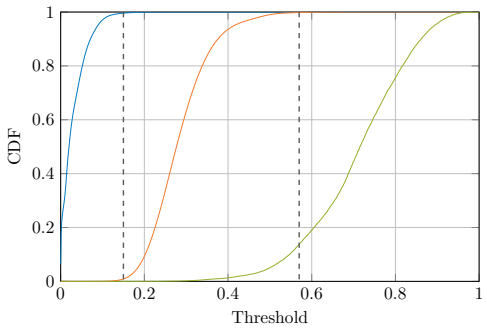

(c) Histogram, images cleaned



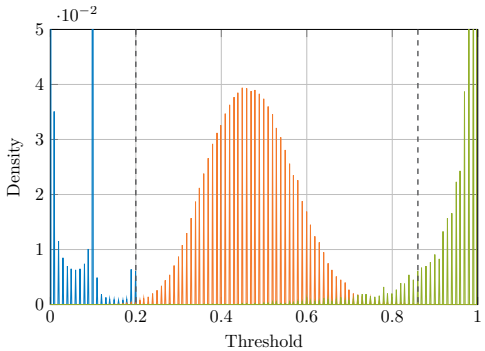

(d) CDF, images cleaned



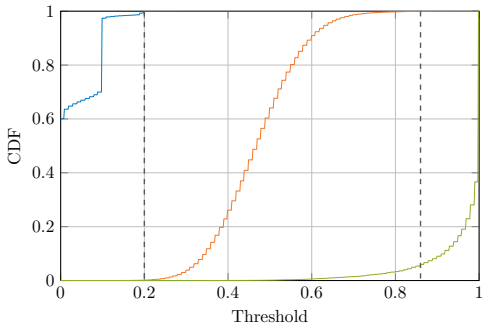

(e) Histogram, images cut




(f) CDF, images cut




(g) Histogram, images cleaned and cut




(h) CDF, images cleaned and cut

Figure 4.5: A visualization of how preprocessing allows us to separate different labels more easily

# Chapter 5

# Implementation

We need to implement different approaches to train, evaluate, and compare them using the pipeline and dataset. We solve the cross recognition both using machine learning-based approaches as well as approaches that rely on computer vision. This comparison allows us to decide whether machine learning is necessary to classify crosses reliably or if computer vision is sufficient for this task. Furthermore, we compare various machine learning methods to each other.

Machine learning-based approaches are covered in Section 5.1 and computer vision based approaches are explained in Section 5.2.

## 5.1 Machine Learning

Identifying the state of a box can be achieved in different ways. We can use different mathematical models to perform the classification. NNs try to replicate the human brain and learn to recognize complex patterns. SVMs can also be used to group images into different classes based on their properties. Lastly, we can make use of decision forests which form their decisions based on an automatically selected subset of features.

Our approach making use of SVMs is presented in Section 5.1.1. Section 5.1.2 elaborates on how we used decision forests to solve the task. Lastly, our usage of NNs is explained in Section 5.1.3.

```
# Model definition
model = sklearn.svm.SVC()

# Training Procedure
model.fit(train_images, train_labels)

# Prediction Procedure
predictions = model.predict(test_images)
```

Listing 5.1: Necessary code to train a SVM

### 5.1.1 Support Vector Machines

We built SVMs using scikit-learn[1]. This framework provides a wrapper around libsvm[2].

The implementation of this approach is pretty straightforward. Three lines of code are sufficient for training and prediction. The implementation of the pipeline separates these different steps and also covers loading and saving both the model and the data. They are shown in Listing 5.1.

While such a straightforward implementation has its advantages, it has the disadvantage that there is not much room for adjustments. The default parameters, which have been carefully chosen by the developers already provide the best results. Therefore there is no real purpose in adjusting those. In addition to that, only a limited number of parameters can be tweaked. Taking these details into account, the above implementation is about all there is to this approach.

### 5.1.2 Decision Forests

Decision forest are implemented using Tensorflow Decision Forests[3] and the following types are provided by the framework:

**CART** A single tree with shrinking

**Random Forest** Multiple, independent CARTs without shrinking

**Gradient Boosted Forest** Multiple, on each other dependent CARTs

Implementing decision trees is just slightly more complex than the straightforward implementation of SVMs. While the framework is different, the interface is very similar and three lines of code are enough to train and evaluate a forest model. Loading the

[1] https://scikit-learn.org/stable/

[2] https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

[3] https://www.tensorflow.org/decision_forests

```
# Model definition
model = tfdf.keras.RandomForestModel()

# Training Procedure
model.compile(['accuracy'])
model.fit(train_images, train_labels)

# Prediction Procedure
predictions = model.predict(test_images)
```

Listing 5.2: Necessary code to train a random forest

dataset or saving the weights is not part of the code snippet as it is handled by the modular pipeline. The relevant code is shown in Listing 5.2.

Similar to support vector machines, the implementation of this approach is straightforward as well. Together with the small number of adjustable parameters, there are not many options for improvement.

### 5.1.3 Neural Networks

Implementing a neural network is straight forward with the Tensorflow[1] framework in combination with the Keras[2] abstraction layer. Keras provides a streamlined API that is implemented by Tensorflow. This interface allows defining a model in a few lines of code. Additionally, the training and predictions procedure can be performed using a few lines of code each. A simplified example using a model with one convolutional layer and two fully-connected layers is shown in Listing 5.3. For our network, we additionally include advanced features, specifically decay and dropout and trained our model for 20 epochs. This has shown to be enough for the network to converge. We used hyperparameter tuning to approximate ideal values for certain parameters of the CNN architecture. The parameter search space is made up of layer-dependent and training-related properties. The first group contains parameters relevant to the convolutional part of the CNN. This is followed by the parameters relevant for the fully connected (dense) section. The last set of parameters is relevant for the training procedure. A full list including the possible values for each of them is shown in Table 5.1.

The number of convolution filters increases each layer by the base value. The same scaling applies to dense layers but in the opposite direction. The number of nodes in the dense layer decrease in the same fashion. As explained by Sharma et al. in [8],

[1] `https://www.tensorflow.org`

[2] `https://keras.io`

```
# Model definition
model = Sequential([
        Conv2D(32, activation='relu', input_shape=(14, 14, 1)),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(200, activation='relu'),
        Dense(100, activation='relu'),
        Dense(3)
])

# Training prodecure
optimizer = Adam(1e-4)
loss = SparseCategoricalCrossEntropy(from_logits=True)
metrics = ['accuracy']
model.compile(optimizer, loss, metrics)
model.fit(train_image, test_images, epochs=10)

# Prediction prodecure
predictions = model.predict(test_images)
predictions = softmax(test_images)
predictions = argmax(test_images)
```

Listing 5.3: Relevant code to train and predict using the Keras API

ReLU is known to be the best activation function for NNs in general. However, other candidate functions were evaluated as well to determine the best activation function for our specific approach. An added benefit of this function is the reduction of the number of nodes that are activated in the network. As explained in Section 2.2.3, this should allow the optimizer to adjust the weights more precisely.

Hyperparameter tuning was performed using Neural Network Intelligence (NNI)[1]. We used the Tree-structured Parzen Estimator (TPE) tuner, which is explained by Bergstra et al. in [14]. It is a Sequential Model-Based Optimization (SMBO) approach. Compared to a completely random choice of parameters, this tuner tests parameter combinations systematically. The core idea is to derive new parameters from previous results. This is achieved by approximating the network's accuracy for the different hyperparameters. It is used as a basis to determine which sets of parameters should be evaluated next.

Every combination of hyperparameters was trained for a total of 20 epochs on a subset of our dataset. This was done to speed up individual training procedures, allowing for more runs in the same time frame. The whole tuning procedure was run for 72 h, resulting in a total of 243 combinations being evaluated.

[1] `https://nni.readthedocs.io/en/stable/`

After the tuning procedure is complete, it is possible to reason about which parameters are the most important. To interpret the 243 tested combinations, we need to aggregate the parameter combinations by the parameter value and the corresponding networks' loss. We plot these correlations in multiple box plots and analyze them. The major insights we are interested in are parameter values associated with the lowest loss. This corresponds to the most precise model that has been tested. Additionally, we are interested in how a hyperparameter influences the final result. This should indicate both ranges which are ideal and also a trend of how the result is affected by the change of this value. Those understandings could help us to determine an ideal network architecture as well as serve as the next starting point for a more granular tuning procedure.

Figure 5.1 shows how the parameters modified during hyperparameter tuning affect the network's performance. A more detailed visualization of the tuning process is presented in Figure A.1 in the Appendix. For the purpose of readability, the results have been aggregated and uniform parameters have been combined into fewer values. This is applied to both decay and dropout, for which only the applicable values between 0 and 1 are plotted in increments of 0.1. If a specific parameter value does not match one of those eleven values, it is added to the closest one available. A similar approach was applied to the learning rate which is chosen from a uniform logarithmic distribution. The data is aggregated between $10^{-5}$ to $10^{-1}$ and a step size of an order of magniude.

The size of the network is primarily determined by the number of layers which are shown in Figure 5.1a. We can see that the best results are achieved when using two convolutional layers and one dense layer as each of those values result in the lowest loss. Figure 5.1b shows the ideal number of filters. This is determined to be 20 by our training. According to the measurements performed using NNI, the ideal kernel size is four. Figure 5.1c supports this and also shows us that a value of six appears to be a suboptimal choice. When choosing the pooling function to use after each convolution, max pooling is much better than average pooling. This can be seen in Figure 5.1d. Figure 5.1e shows that the sigmoid activation function performs the worst for our specific use case while the hyperbolic tangent achieves the best result but ReLU is not far off either. This is true both for convolutional as well as fully connected (dense) layers. The ideal number of nodes within the single fully-connected layer is 64 as it provides the best results. The other possible values perform worse with 92 nodes being in-between as presented in Figure 5.1f.

Figure 5.1g displays the ideal values for both decay and dropout. Decay performs best with small values such as 0.1 as well as big values above 0.7. A decay of 0 does not perform well at all which is expected because it sets the learning rate to 0 after the first epoch. Due to our aggregation and the tuning algorithm, not all values for decay are

| Name | Value(s) | | |
|---|---|---|---|
| | **Tested** | **Optimal** | **Final** |
| #Conv. Layers | 1, 2 | 2 | 2 |
| #Conv. Filters | $\{8, 12, \ldots, 24\}$ | 20 | 20 |
| Conv. Kernel Size | 3 to 7 | 3 to 5 | 4 |
| Conv. Pooling | avg, max | max | max |
| Conv. Activation | ReLU, sigmoid, tanh | max, tanh | tanh |
| #Dense Layers | 1 to 4 | 1 and 3 | 1 |
| #Dense Nodes | $\{32, 48, \ldots, 92\}$ | 64 | 64 |
| Dense Activation | ReLU, sigmoid, tanh | max, tanh | tanh |
| Learning Rate | $10^{-6}$ to $10^{-1}$ | $10^{-4}$ to $10^{-3}$ | $1.6 \times 10^{-4}$ |
| Decay | 0 to 1 | 0.8 to 0.9 | 0.759 |
| Dropout | 0 to 0.6 | 0 to 0.2 | 0.002 |

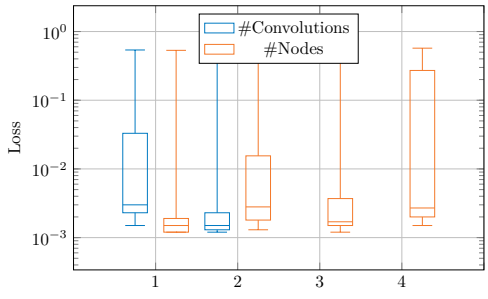
Table 5.1: The parameter values used during testing.

present in the plot. From our research, a lower dropout performs better. It appears that using dropout might not be as important for our use case. This could be due to our training procedure, i. e. it might be compensated for by the learning rate and decay. It could also be that as our dataset features very small images which are all fairly similar for the same label, overfitting might not be that much of a problem compared to other datasets and applications. Finally, the relation between the learning rate and the models' loss is shown in Figure 5.1h. We can clearly see that a value of $10^{-2}$ performs the worst and the ideal learning rate is around $10^{-4}$ to $10^{-3}$.

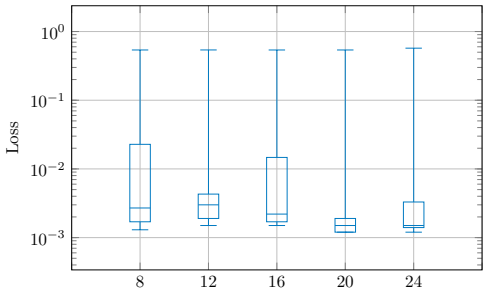
The optimal parameter values we extracted from this analysis are summarized in Table 5.1.

The final network architecture, which matches the best set of parameters during hyperparameter tuning, is also shown in Table 5.1. These values might differ slightly from the ideal values as those were both averaged and aggregated.

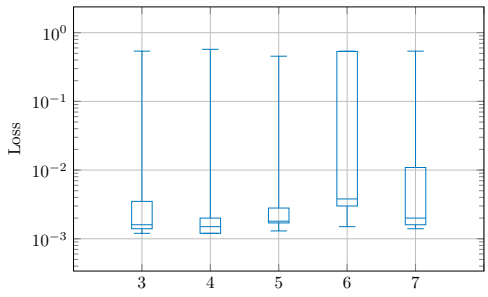
For comparison, the architecture of the convolutional neural network provided by TUMexam is shown in Table 5.2. The major difference are the much larger dense layers, around one order of magnitude bigger. Other differences are the greater number of filters, the different activation function as well as different training parameters. Contrary to our network defined above, decay is not used during training of this network.



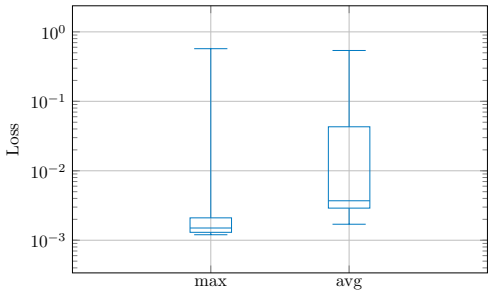

(a) Layer sizes



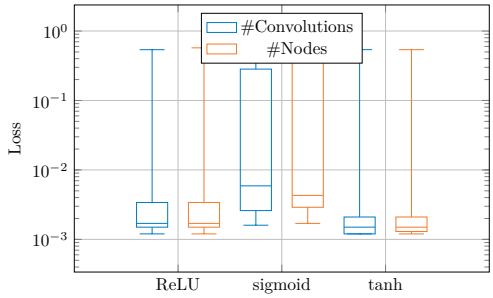

(b) Number of convolution filters



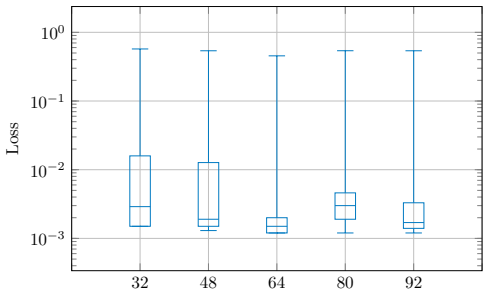

(c) Number of convolution kernel sizes



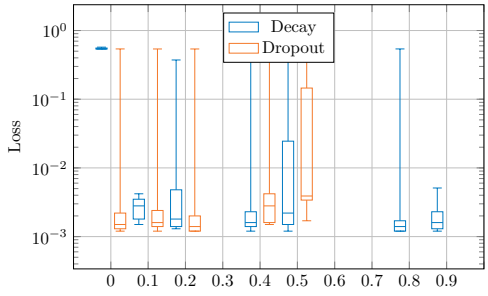

(d) Convolution pooling functions



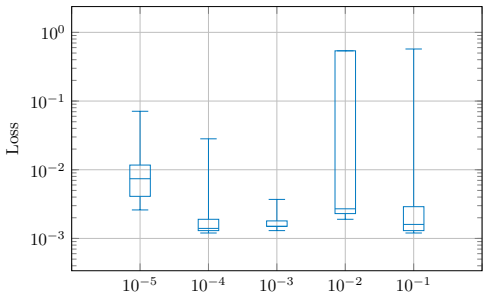

(e) Activation functions




(f) Number of fully-connected nodes




(g) Decay and dropout




(h) Learning rates

Figure 5.1: The effect different parameter values have on final loss. Lower is better.

| Parameter | Value |
|---|---|
| #Convolutions | 2 |
| #Convolution Filters$_i$ | $32, 64$ |
| Convolution Kernel Size | 5 |
| Convolution Pooling | max |
| Convolution Activation | ReLU |
| #Dense Layers | 2 |
| #Dense Nodes$_i$ | $1000, 800$ |
| Dense Activation | ReLU |
| Learning Rate | $1 \times 10^{-4}$ |
| Dropout | 0.2 |

Table 5.2: The parameters of the TUMexam neural network

## 5.2 Computer Vision

There are two different approaches we came up with. The first approach uses masks to which images are compared. This approach is based on the assumption that images of the same class look similar. We can confirm this by looking at the dataset visualization in Section 4.3 and more specifically Figure 4.3. Apart from the border, most empty boxes are completely white and filled crosses black. Boxes that are selected, have two intersecting, diagonal lines within them. As we are only interested in the average of all of those images, the actual shape does not matter that much. We only rely on similar images containing a similar pattern and different images having a different appearance.

For our first approach, we compute the average of all images of a given class. The results are three averaged masks, one each for empty, selected, and filled boxes. Images are classified according to the mask they match the most. The masks are computed as the average of all images from each class. Therefore, we get an averaged mask for empty, selected, and filled boxes. To determine which mask represents the image we want to classify, we compute the difference of the image to all masks. This result are the deviations we are interested in. Before we can compute a meaningful result, we need to take the absolute value for each pixel, so that deviations are not able to neutralize each other. Finally, we sum the deviations and can take the label corresponding to the minimal deviation as our prediction. The numeric values zero, one, and two correspond to the labels empty, selected, and filled boxes, respectively. The algorithm is shown in Listing 5.4 and visualized in Figure 5.2a with cutting being applied.

Our second approach uses thresholds to classify the boxes. This approach is based on the assumption of how the boxes are filled. Again, excluding the border, boxes

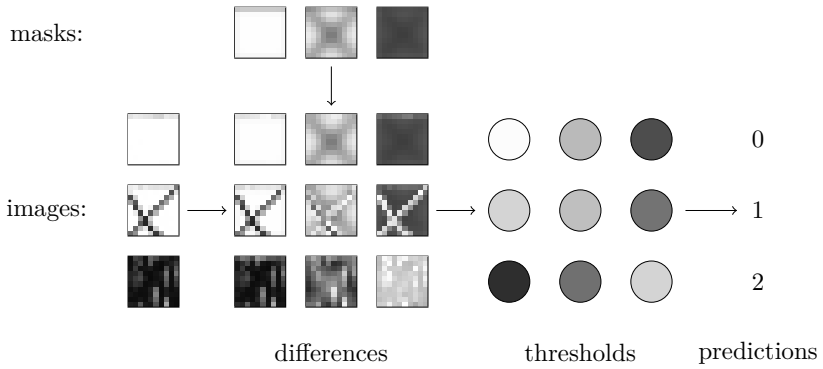
```
# Training Procedure
# Compute average mask for each label (empty, selected, and filled)
mask_empty = average(train_images_empty)
[...]

# Prediction Procedure
# Compute difference of input images to each mask
difference_empty = absolute(test_images - mask_empty)
delta_empty = sum(difference_empty)
[...]
delta = [delta_empty, delta_selected, delta_filled]
# Identify minimal difference for every image and mask
predictions = argmin(delta)
```

Listing 5.4: Algorithm to create averaged masks and perform predictions

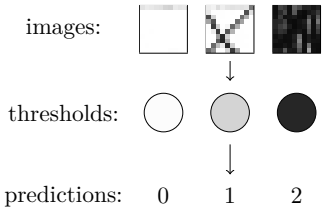
labeled as empty should be white, and filled boxes should be black. Selected boxes should be somewhere in-between. The threshold measures the amount of box which has been drawn on. Empty boxes should therefore have a threshold close to zero. This value increases for selected boxes and ends with filled boxes whose threshold should be close at around one. We can verify this assumption by looking at our analysis results in Section 4.3, more precisely in Figure 4.5. We see that the thresholds match the expectations. The effect of removing the border is clearly visible as the minimum threshold decreases from around 0.3 to 0.

For this approach, we compute the amount of the image that is filled. Using these thresholds we can identify lower and upper boundaries. Values below the lower boundary are treated as empty and thresholds between both boundaries are predicted as selected. Finally, images with a threshold above the upper boundary are classified as filled. We can evaluate and minimize the errors for every possible combination of boundaries These values are the ideal value we use classification using this approach.

To further improve the accuracy, we can preprocess the images before computing thresholds. As explained in Section 4.3, we can clean the image and we can also cut the image. The first option is implemented by applying a binary threshold to the images, after which every pixel is either fully black or white and has no values in-between. Cutting works by removing a small part of the edges of the image. This should get rid of the black border which contains no useful information for us. The ideal boundaries can now be re-evaluated with one or both preprocessing methods applied to the image and we can identify the combination with the highest accuracy. Again, the numeric values zero, one, and two correspond to the labels empty, selected, and filled, respectively. This process is visualized in Figure 5.2b with cutting being applied.




(a) Steps of the mask-based approach




(b) Steps of the threshold-based approach

Figure 5.2: Visualization of the steps for our computer vision-based approaches

# Chapter 6

# Evaluation

The evaluation was performed on the following testing hardware. It has an Intel Xeon Silver 4214 CPU with 12 cores running at 2.2 GHz and hyperthreading enabled. This is paired with an nVidia GeForce RTX 2080 Super. A total of 250 GB of memory is available. It is important to note that everything resides in this memory, including the file system. As such, loading times are expected to be faster compared to a system with a physical drive.

For the evaluation, a dataset with a 80 % train-test-split consisting of 620 k individual images is used. We evaluated the different approaches regarding their accuracy and runtime with the former being the more important property. The accuracy is measured as the rate of correctly predicted labels of all labels and the runtime measures the time needed to perform the prediction. This excludes loading times for both the frameworks used as well as loading the evaluation data and pre-trained model from disk. By doing so, we are able to compare the evaluation speed amongst different approaches. The amount of memory the different solutions consume is also taken into consideration. The resulting performance metrics are shown in Table 6.1 and Table 6.2. The runtime explicitly does neither include the time necessary to build the dataset nor the time necessary for preprocessing the data and extracting features. The CNNs are amongst

| **Framework** | **Loading Time** [s] |
|---|---|
| Tensorflow | 2.8 |
| Sklearn | 0.7 |
| TF-Decisiontrees | 3.3 |

Table 6.1: Evaluation of the different frameworks regarding loading time

| **Approach** | **Accuracy** [%] | **Runtime** [s] |
|---|---|---|
| CNN | 99.975 | 9.3 |
| TUMexam-CNN | 99.966 | 5.1 |
| SVM | 99.954 | 8.2 |
| RF | 99.926 | 3.7 |
| GBF | 99.922 | 2.2 |
| CART | 99.796 | 1.8 |
| Threshold | 99.859 | 0.2 |
| Mask | 96.566 | 0.2 |

Table 6.2: Evaluation of different classification approaches

the slowest while also delivering the best results. They are followed closely by the SVM-based approach. The inverse relationship between accuracy and speed seems to hold for all explored approaches and especially the computer vision-based approaches using thresholds or masks are not able to keep up with the alternatives regarding their precision. Decision trees are a bit better and slower than classification using computer vision but still worse than NNs or SVMs.

Furthermore, we inspected the memory consumption of the different approaches. The memory usage is split up into SMU (Specific Memory Usage), TMU (Total Memory Usage), and PDU (Prediction Delta Usage). The specific memory usage is the difference between the memory usage of the framework and the total memory usage. The prediction delta usage is the additional memory consumption caused by predicting using the dataset. This differs from the specific memory usage which measures the difference between loading the framework and additionally loading the model and its weights. For the frameworks, the effect of utilizing the GPU has been evaluated as well. In this case, the memory consumption is a lot higher. This is expected as additional libraries need to be loaded. The memory usage was measured using the Fil[1] memory profiler. It is unfortunately not able to handle subprocesses spawned by the initial program and it is possible that the reported consumption is below the total memory usage. This appears to be a problem during predictions performed by decision trees. The usage of the frameworks is presented in Table 6.3 and the consumption of each approach is presented in Table 6.4 when the model is loaded and a prediction is made on the dataset. We can see, that the framework has the greatest impact on the TMU. Another interesting result is that decision forests, compared to other approaches, have a much greater SMU. This

[1] `https://github.com/pythonspeed/filprofiler`

| **Framework** | **Memory Usage (on CPU/on GPU)** [MiB] |
|---|---|
| Numpy only | 745.0 / 2950.9 |
| Tensorflow | 1604.4 / 3783.0 |
| Sklearn | 1507.5 / 3712.4 |
| TF-Decisiontrees | 1611.1 / 3782.9 |

Table 6.3: Evaluation of the different frameworks regarding memory usage

| **Approach** | **SMU** [MiB] | **PDU** [MiB] | **TMU** [MiB] |
|---|---|---|---|
| CNN | 5.4 | 24 | 1238.1 |
| TUMexam-CNN | 19.3 | 20.1 | 1249.3 |
| SVM | 5.8 | 127.5 | 1160.1 |
| RF | 151.4 | 0 | 1398.3 |
| GBF | 121.1 | 0 | 1368.0 |
| CART | 119.4 | 0 | 1366.3 |
| Threshold | 0.2 | 0.8 | 903.3 |
| Mask | 0.2 | 1.6 | 903.3 |

Table 6.4: Evaluation of the different approaches regarding specific and total memory usage when loading the model and predicting

is partially expected a decision forest is made up of hundreds of trees. At the same time, CART uses a similar amount of memory while it should be made up of a single tree, which immediaely negates this claim. The specific memory usage is therefore a more useful metric as the differences are more obvious. This separation makes it possible to quickly compare approaches within the same framework to each other. While the framework for decision trees appears to have the largest memory footprint this is not exactly the case. It is based on tensorflow which already has a large footprint on its own and the relative footprint in that case would only be around 17 MiB.

## 6.1 Analyzing errors

As explained in Section 1.2 we are not able to recover from user errors. Firstly, this includes images that are labeled incorrectly. Additionally, this includes images that are either marked ambiguously or where the actual state is difficult to determine for humans as well. We accept that those cases are classified incorrectly as we do not aim to be better than a human regarding the classification ability. Some images where the prediction failed are shown in Figure 6.1. We can see that such empty crosses contain a single line which is half of a cross and therefore neither unambiguously empty or selected.

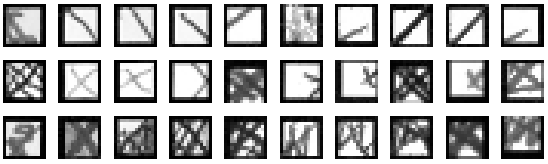


Figure 6.1: Incorrectly classified images, also difficult to recognize by humans

Selected crosses are either not crossed in the full box or they are crosses multiple times. In both cases, the intention of the user is not necessarily clear. A similar statement can be made about filled crosses: they are partially filled but not necessarily clearly filled or selected. While such images exist in our dataset we do not consider them to be predicted incorrectly for those reasons.

## 6.2 Improving the accuracy

To improve the accuracy of the CNN, we can use the modified datasets presented in Section 4.2. These datasets attempt to reduce the imbalance and improve both the overall accuracy as well as the accuracy for each individual label. For a detailed analysis we computed the accuracy, precision, recall, and F1-Score as explained in Section 2.1. The results are presented in Figure 6.2. It is important to note that the bar charts do not start at zero but at 60. Another thing to point out is that the greatest differences occur for filled crosses while the metrics for the other labels remain fairly constant.

The confusion matrix for the original, unmodified dataset is shown in Figure 6.2a. Every label has around 60 misclassifications. To put this into perspective, the "filled" label has 3025 correct predictions while the "empty" label has around 500 k. The accuracy, therefore, decreases as the number of images in each label decreases while the mistakes remain constant.

To mitigate this issue, the dataset has been altered to reduce the imbalance. Figure 6.2b presents the confusion matrix when all three labels are represented in equal amounts. Compared to Figure 6.2a the evaluation of empty crosses is off slightly worse but the classification of filled ones has greatly improved. Unfortunately, the label "selected" also encountered a degradation as now about 1400 misclassifications occur.

Taking both of the previous results into consideration, the third dataset alteration does not reduce the number of crosses that drastically but still increases the number of filled crosses. The resulting confusion matrix can be seen in Figure 6.2c. We can see combined effects from both previous attempts: filled crosses are recognized much better without a huge reduction in prediction accuracy for the other labels.

## 6.3 Analyzing feature importance

Feature importance is an important step towards understanding how a machine learning-based approach comes to a conclusion. This allows us to understand which features are the most important to the neural network. Furthermore, we can ensure that the model works as predicted, i.e. we can check if the model focuses on the content of the image or its' border.

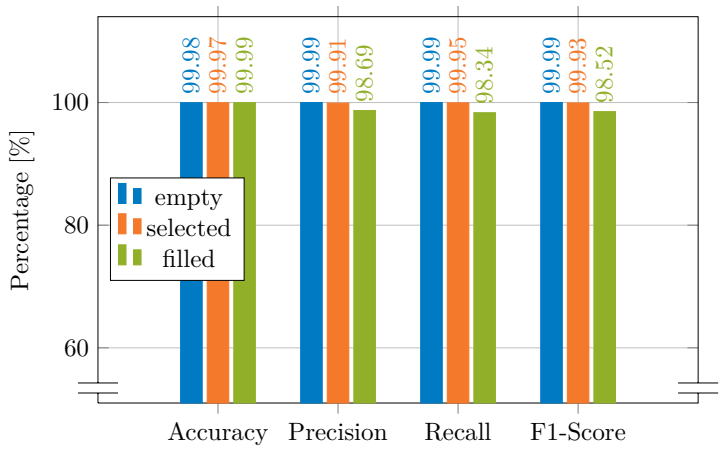
Extracting the most important features from a neural network is a rather complicated task. This is the case because each input value affects many intermediate values before a conclusion is formed. As such, identifying the influence of a single feature on the result is difficult.

At the same time, solving this problem is quite simple as there are existing frameworks designed for that exact problem. The method we used was researched by Lundberg and Lee in [15]. They developed a "game theoretic approach", SHapley Additive exPlanations (SHAP), to identify the most important features for a given machine learning model. During the evaluation, each pixel is assigned a so-called shap value identifying its importance in predicting or not predicting a specific class.

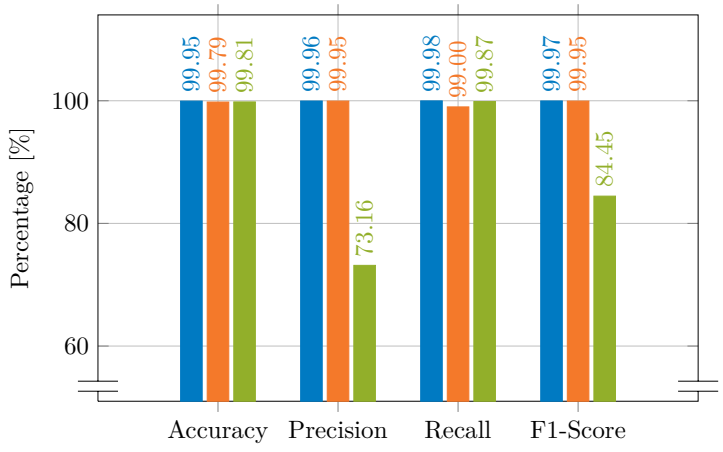
This information can be used to derive a heatmap. One heatmap is created for every image and label. Each heatmap visualizes pixels which contribute towards the respective label in red and pixels discouraging a certain prediction as blue. From this heatmap, we can identify the area which is inspected by the NN. The most relevant area is in the center of the image with a special focus on the area occupied by a cross. The heatmaps clearly show that blue pixels identify areas which have the wrong color and red pixels indicate the correct color. Color in this case refers to the pixel being white or black. Most notably, the heatmap for the correct label is mostly red. They are presented in Figure 6.3.

In addition to heatmaps, we are also able to identify the most important pixels with regards to the prediction result. To identify these, the evaluated the five highest and lowest shap values in five image per label. These are the pixels with the greatest influence on the result. Again, a higher value indicates a better that the predictions matches the correct result and vice versa. As we are only interested in relevant pixels, only those occuring multiple times were taken into consideration. The remaining pixels were ordered by the number of occurrences. We can see that the neural network focuses the most on the lower right position of the cross. This matches our expectations as it treats pixels as important the way we expected it to and it also did not focus on pixels

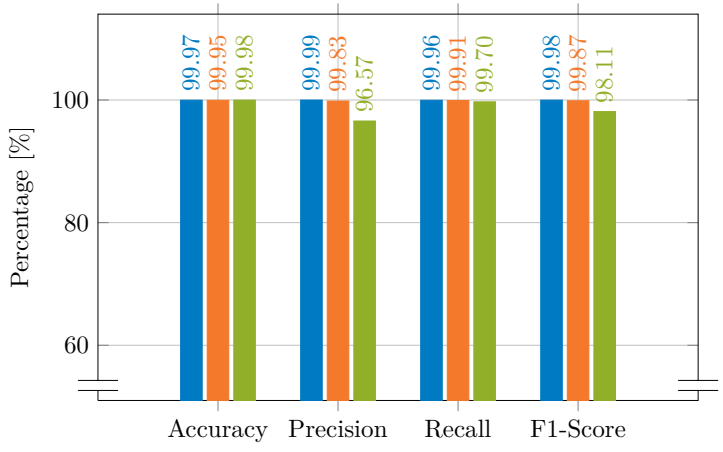
we would consider superfluous, i.e. the border of the image. The two, four, and seven most important pixels are presented in Figure 6.4.




(a) The original data




(b) The equalized data




(c) The augmented data

Figure 6.2: The confusion matrices for our altered datasets

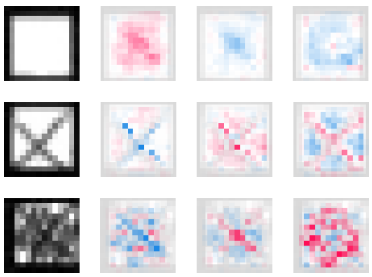


Figure 6.3: Shap-based heatmaps for an empty, selected, and filled box each

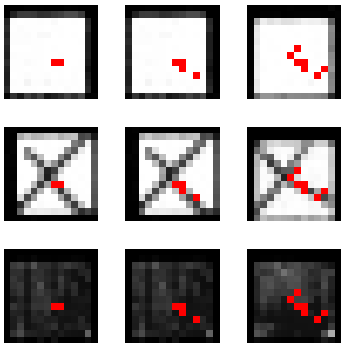


Figure 6.4: The two, four, and seven most important pixels for classification

# Chapter 7

# Related Work

In this chapter, research similar to ours in this thesis is presented. We will explain what has been researched and why it does not fit our use case.

In Section 7.1 the automatic recognition of checkboxes using computer vision is explained. The classification of handwritten digits is covered in Section 7.2. In Section 7.3 presents the identification of handwritten characters.

## 7.1 Checkbox Recognition

The automated recognition of checkboxes within digitalized forms was researched by Shengnan et al. in [16]. The goal was to identify if checkboxes have been selected with a checkmark or a cross. A visualization can be seen in Figure 7.1. The classification was performed using a computer vision-based approach. The lines in the checkbox were extracted, analyzed, and interpreted. The general idea of the algorithm is as follows: Two intersecting lines form a cross and two touching lines resemble a checkmark. Using this approach, they got an accuracy of 93 % to 95 %. The advantage of this solution is

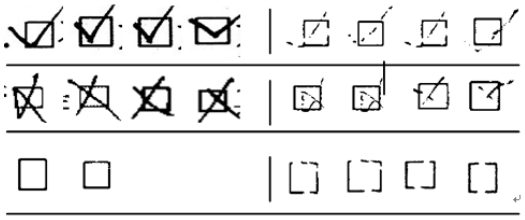


Figure 7.1: A visualization of different checkboxes. From top to bottom: checkmarks, crosses, and empty checkboxes in varying qualities. Taken from [16].

Figure 7.2: A visualization of different digits. From top to bottom: the digits 0 to 9. Taken from [2].

its simplicity compared to neural networks which are more computationally expensive. While they achieved good results, this solution does not make use of state-of-the-art technologies. This is expected as their research was done in 2014. Additionally, the accuracy is good for a computer vision-based approach but can be improved upon with machine learning. The research conducted does not fit this thesis as for the immense number of crosses a higher accuracy of at least 99.9 % is needed. Achieving those results requires us to make use of newer state-of-the-art technologies. Finally, our dataset contains only crosses to mark checkboxes as selected, so checkmark recognition is not needed.

## 7.2 Digit Recognition

The recognition of handwritten digits was researched by LeCun et al. in [2]. A visualization of those digits can be seen in Figure 7.2. They analyzed the accuracy of many approaches, including the work of others. The classification of digits was performed using various NNs and CNNs, k-nearest-neighbor algorithms, decision trees, and SVMs with a total of 69 approaches. Note however that many of those approaches are similar and some variations of the same approach are part of the same research. Using the previously mentioned approaches, they achieved an accuracy of 95.3 % to 99.77 %. Most approaches reached an accuracy of at least 98.6 % with convolutional neural networks delivering the overall best results. The advantage of this solution is the very high accuracy of some network architectures. While they achieved desired results and this

solution makes use of state-of-the-art technologies, it might be possible that simpler architectures work for our use case as well. This is a result of our data having less variance which might give us the option to use a smaller network. Therefore, the research conducted partially fits this thesis. Similar to the checkbox recognition, a slightly higher accuracy would be better.

## 7.3 Character Recognition

The recognition of handwritten characters was researched by Sharma et al. in [17]. The classification was performed using various machine learning algorithms. This includes neural networks with and without convolutional layers as well as support vector machines. Using the previously mentioned approaches, they achieved an accuracy of 81.66 % to 98.61 %. Most approaches reached an accuracy of at least 95 %. The advantage of this solution is the high accuracy of some network architectures. While they achieved quite good results and this solution makes use of state-of-the-art technologies, it might be possible that simpler architectures work for our use case as well. While alphabets consist of many different characters, this thesis only covers the evaluation of checkboxes. As a result, our solutions do not need to differentiate between as many different outcomes compared to an alphabet. Therefore it should be possible to use simpler networks and still achieve high accuracy. Therefore, the research conducted does not fit this thesis as it identifies too many different labels and the accuracy is also below our threshold of 99.9 %.

# Chapter 8

# Conclusion

After implementing and evaluating different approaches, we can now reason about which fits the task the best. We can identify solutions that are suited the best to solve the task for our use case. Our work is summarized in Section 8.1. Research questions are answered in Section 8.2 and future work is listed in Section 8.3.

## 8.1 Summary

In this thesis, we researched different techniques aiming to quickly and precisely identify crosses. This includes both various machine learning-based as well as computer vision-based approaches. We then designed a modular pipeline for automated training, tuning, evaluation, and prediction using various approaches. This modular pipeline is split into multiple steps, all of which can be modified independently of each other. We are for example able to add a new preprocessing method or framework without breaking the existing networks. Furthermore, the networks defined in terms of this pipeline are independent of the surroundings as the pipeline provides a common format e. g. for input images. We analyzed the dataset we are working with using different metrics. This includes the distribution of images across different classes as well as the threshold of the images. Preprocessing was also introduced for the latter as a way to alter the threshold for easier separation. A large number of approaches have been implemented, evaluated, and revised. This includes CNNs, SVMs, decision forests as well as custom, computer vision-based approaches. For the CNNs, hypertuning was employed to further optimize their architecture to achieve the best possible results. All of those approaches were evaluated according to their accuracy, runtime, memory usage, and other performance metrics and compared with each other. For the most promising approach, the dataset

was adjusted to compensate for its imbalance as a means to further improve the accuracy. Additionally, the most important features were analyzed to understand how the network forms its decision. This information was also used to ensure the prediction is formed reasonably.

The best approach for both raw speed and memory usage is the computer vision-based threshold analysis. Due to its simple nature and high accuracy, it serves as a good baseline other approaches need to surpass. This approach is the best choice if a minimal footprint, yet accurate result is wanted. The best solution with regards to accuracy is the tuned CNN. It provided the best result at the cost of a higher memory usage and prediction time. This approach is the correct choice if the absolute best performing solution for cross-evaluation is required and memory usage and runtime are not the main concern. It is also the recommendation to use for TUMexam. During the correction, the raw speed is not that important but due to the huge number of crosses which are evaluated, minimizing the errors is the top priority.

## 8.2 Research Questions

*Q1:* Which approaches are suitable for the automatic recognition of crosses?

We consider approaches with an accuracy of at least 99.9 % to be suitable for the recognition task. Based on the evaluation, CNNs, SVMs, and decision trees are suitable for this task. The best results were achieved using CNNs. Our computer vision-based approach that is analyzing thresholds barely misses our target accuracy by 0.041 % but is still very good. It is ideal for a quick evaluation as it provided very good results in the least amount of time.

*Q2:* How large does the input space need to be to produce accurate results?

We have identified grayscale images with a size of 14 pixels in each dimension produce great results. The threshold-based analysis also produced very good results with only four thresholds values per image, although they are derived from the same base image which is also used for other approaches.

*Q3:* What is the necessary size for a network to produce accurate results?

From our experiments using hypertuning, we have identified that a CNN with two convolutional layers and a single fully-connected layer is not only sufficient but provides the best result of all tested approaches. The two convolutional layers have 20 and 40

filters and the fully-connected layer has 64 neurons. To achieve those results, the tanh activation function and dropout are a necessity.

## 8.3 Future Work

With the pipeline being constructed, it is now ready to use. This means new approaches can be researched, implemented, and evaluated. Furthermore, it can be used to compare current and future implementations. With this information, it is possible to replace the box classification approach in use with another one that is better suited to the task and requirements. By being able to re-evaluate previously used approaches, approaches can be switched when requirements change or technologies evolve.

Extending the pipeline to support more advanced scenarios is also possible. The existing cross-evaluation could be extended to support rare edge cases. An example of this, which is used with TUMexam, would be the recognition of re-selected crosses. If a student decides that a box he filled should now be selected again, he or she is supposed to draw a small arrow beneath the box as a means to indicate their intent. Therefore, the cross-evaluation could be extended by additionally inspecting the area around the crosses. The automatic handling of such an edge case would further reduce the required manual labor for cross-evaluation.

New use cases include the recognition of handwritten digits. In addition to crosses, placeholders for digits could be added to the form as well. An example of this, which is also used with TUMexam, is the evaluation of the handwritten registration number. If this process is automated, the value can automatically be compared to the cross-encoded equivalent in the registration number box. This optimization would further aid the automated exam correction.

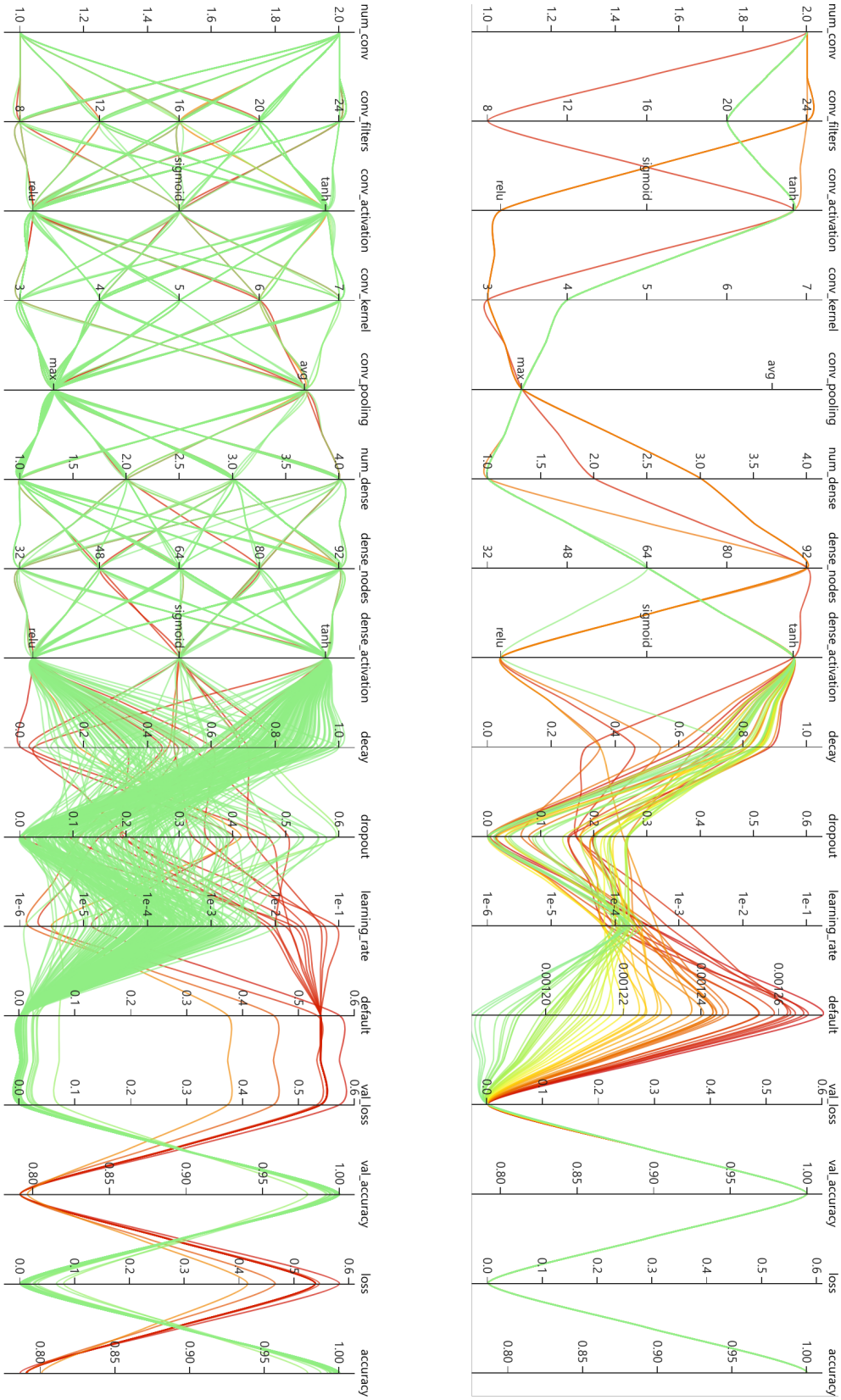
Even with the most versatile and precise approach, human error can still negatively affect the accuracy of the prediction. For this reason, it can be favorable to evaluate the performance degradation resulting from human error. If it is considered to be too big of an issue, it can be researched further. Three potential causes can be identified and measured. Firstly, the number of crosses which are labeled incorrectly and their influence on the prediction accuracy. Secondly, bad crosses entered by the user, such as an incompletely filled or a partially selected box. While it might not be possible to prevent this problem it could be feasible to detect and handle it during prediction. This would be possible by introducing one or more labels for exceptional cases. The last source of error is created by problems during digitalization. This includes but is not limited to imperfections in the paper, skewing, or shadows while a digital copy is created. Additional problems might arise during the extraction of crosses from the digitalized

form. As the cross is rather small, slight errors can severely reduce the quality of the final result.

# Chapter A

# Appendix

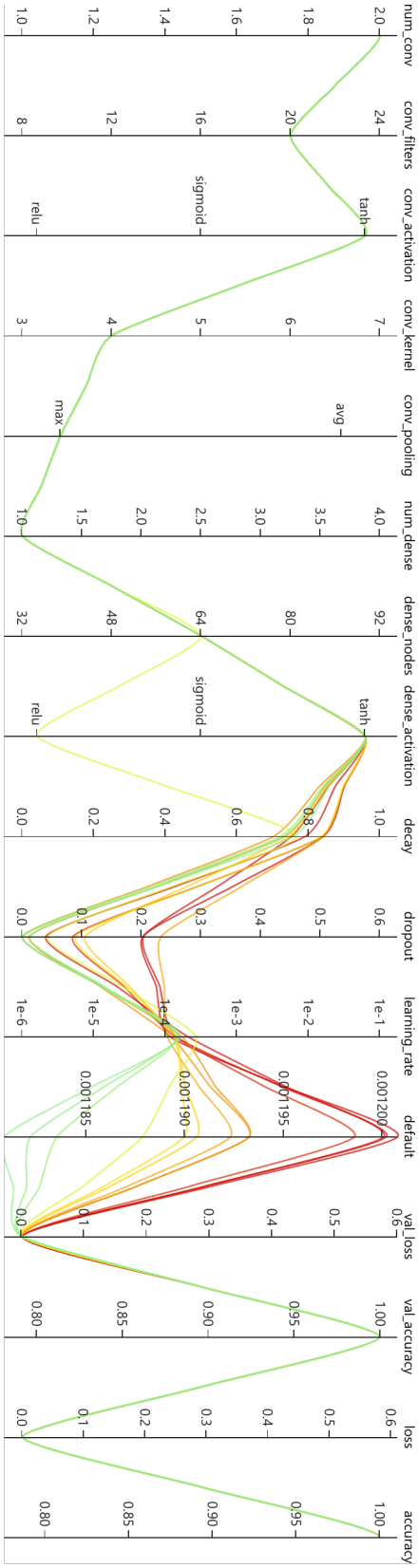
In this section, additional plots from hypertuning are presented here. They supplement the plots in Section 5.1.3. Every y-axis displays a hyperparameter or a resulting metric. The metric `default` is required by the framework and always has the same value as `val_loss`. The plot for all parameter combinations is shown in Figure A.1a. The optimal values identified in Section 5.1.3 which are presented in Table 5.1 can also be identified in the following plots. For the best 20 % trials, we can see that a lot of combinations are not tested anymore. Average convolution pooling or 32 dense nodes are examples of this. It also applies to large dropout values and learning rates. A trend towards optimal values can be seen for decay, dropout, and learning rate. For the best 5 % this trend is much easier to spot and also includes parameters relevant to the network architecture. The best 1 % trials do not differentiate at all for discrete architectural parameters and only slightly adjust training-related parameters. These steps gradially decrease to the best 20 %, 5 %, and 1 % shown in Figures A.1b to d, respectively.



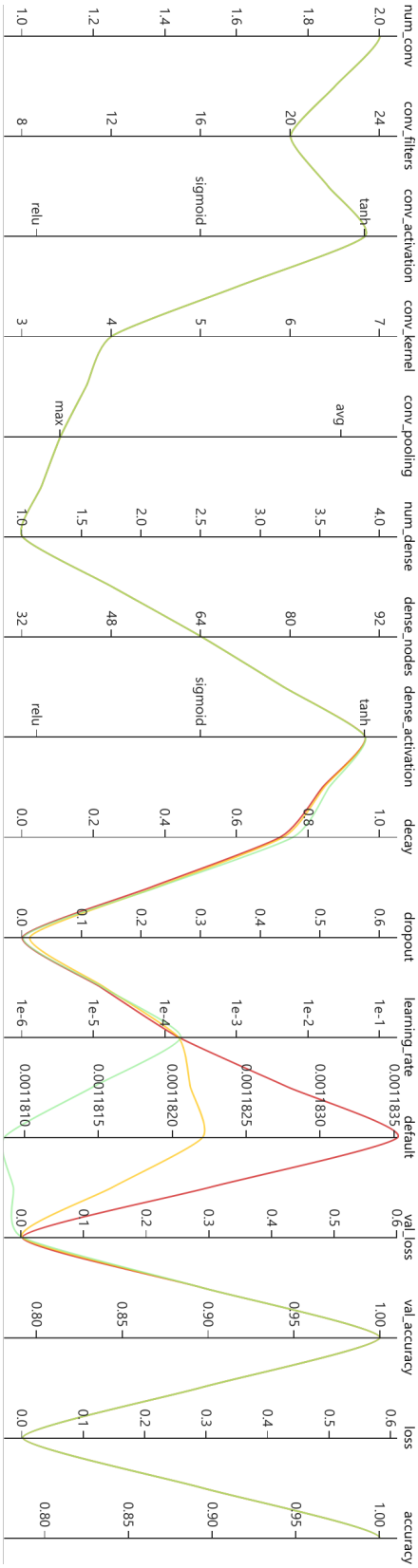

(a) All trials

(b) Best 20 % trials

Figure A.1: Trial parameters in different granularity




(c) Best 5 % trials




(d) Best 1 % trials

FIGURE A.1: Trial parameters in different granularity

## A.1 List of acronyms

**NN** Neural Network

**CNN** Convolutional Neural Network

**SVM** Support Vector Machine

**CART** Classification and Regression Trees

**ReLU** Rectified Linear Unit, an activation function

**SHAP** SHapley Additive exPlanations, a framework to explain how a machine learning approach makes a prediction

**NNI** Neural Network Intelligence, a framework for hyperparameter optimization which can also identify optimal values for individual parameters

**TPE** Tree-structured Parzen Estimator, a SMBO approach for hyperparameter optimization

**SMBO** Sequential Model-Based Optimization, an algorithm where the choice of new parameters is based on previous results

**OCR** Optical Character Recognition, a technique to extract text from images

# Literatur


[1] K. Simonyan und A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] Y. LeCun, L. Bottou, Y. Bengio und P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, Jg. 86, Nr. 11, S. 2278–2324, 1998.

[3] K. He, X. Zhang, S. Ren und J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 770–778.

[4] R. J. Lewis, "An introduction to classification and regression tree (CART) analysis," in *Annual meeting of the society for academic emergency medicine in San Francisco, California*, Citeseer, Bd. 14, 2000.

[5] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Bd. 1, 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[6] K. S. Durgesh und B Lekha, "Data classification using support vector machine," *Journal of theoretical and applied information technology*, Jg. 12, Nr. 1, S. 1–7, 2010.

[7] A. Dongare, R. Kharde, A. D. Kachare u. a., "Introduction to artificial neural network," *International Journal of Engineering and Innovative Technology (IJEIT)*, Jg. 2, Nr. 1, S. 189–194, 2012.

[8] S. Sharma, S. Sharma und A. Athaiya, "Activation functions in neural networks," *towards data science*, Jg. 6, Nr. 12, S. 310–316, 2017.

[9] K. You, M. Long, J. Wang und M. I. Jordan, "How does learning rate decay help modern neural networks?" *arXiv preprint arXiv:1908.01878*, 2019.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, Jg. 15, Nr. 1, S. 1929–1958, 2014.

[11] K. O'Shea und R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[12] J. Villing, "TUMexam: Digitalizing Attendee Control for Examinations – Recognition of Student Card and Registration Number Box," Bachelor's Thesis, Technical University of Munich, 2019.

[13] D. Masko und P. Hensman, *The impact of imbalanced training data for convolutional neural networks*, 2015.

[14] J. Bergstra, R. Bardenet, Y. Bengio und B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, Jg. 24, 2011.

[15] S. M. Lundberg und S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio u. a., Hrsg., Curran Associates, Inc., 2017, S. 4765–4774. Adresse: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

[16] Z. Shengnan, Y. Shanlei und N. Lianqiang, "Automatic Recognition Method for Checkbox in Data Form Image," in *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, 2014, S. 159–162. DOI: `10.1109/ICMTMA.2014.42`.

[17] R. Sharma, B. Kaushik und N. Gondhi, "Character Recognition using Machine Learning and Deep Learning - A Survey," in *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2020, S. 341–345. DOI: `10.1109/ESCI48226.2020.9167649`.