FHV
University of Applied Sciences

Grading Erosion on Pelton Wheels from Cavitation with Images

Master-Computer Science

Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science, MSc.

Summer term, 2022

Submitted to
Sebastian Hegenbart

Submitted by
Thomas Kraxner

Abstract

Erosion due to cavitation is a common problem for any kind of water turbine. Most of the currently used techniques to detect cavitation are using an Acoustic Emission (AE) sensor and highspeed cameras during operation. For the pelton wheel which is subject of this thesis it is impossible to take pictures during operation, because of the splashing water and the mist. Therefore this thesis aims to explore possibilities in detecting erosion on the buckets of the pelton wheel on images taken during manual inspections. Since the provided images are snapshots taken with a mobile phone camera without a tripod, a lot of effort was invested in the preprocessing of the images. For the main task, the classification of the erosion, two methods were evaluated: *Local Binary Patterns (LBP) + kNearest neighbor classification* and the classification with a *Convolutional Neural Network (CNN)*. The given 2405 images, contained 4810 buckets on which the erosion was graded from zero to four. This means the baseline for the classification accuracy is 20%. *LBP + kNearest neighbor classification* scored 32.03%. The chosen CNN model, a light version of the *Xception* architecture outperformed the *LBP + kNearest classification* with 58,29%. The biggest issue found during research is the variance of the erosion grading by the maintainance personnel. Reasons for this are: no objective grading critera like the area of erosion in mm^2, classification by different employees, a shift in grading from overall bucket condition to erosion from cavitation and too many classes for grading. The mentioned reasons were confirmed by the manual classification experiment were an *IllwerkeVKW* employee had to perform the grading on images of the dataset. The contestants accuracy score was 36% for this task. The result of 58,29% classification accuracy indicates that an automated grading of erosion by cavitation is feasible.

Zusammenfassung

Erosion durch Kavitation tritt beim Betrieb jeder Bauart von Turbinen in Flüssigkeiten auf. Die meisten Methoden zur Detektion von Kavitation benutzen akustische Sensoren und Hochgeschwindigkeitskameras, um Kavitation während des Betriebs festzustellen. Durch die Funktion des Pelton Rades ist es unmöglich Bilder während des Betriebs zu erfassen. Gründe dafür sind spritzendes Wasser und Nebel. Ziel dieser Thesis ist es daher Methoden zu finden welche den Erosionsgrad auf den Bechern des Peltonrades auf Basis von Inspektionsfotos bestimmen. Da sämtliche Inspektionsfotos Freihand mit einer Handy Kamera erzeugt wurden, bestand ein großer Teil der Arbeit darin Methoden zur Vorverarbeitung der Bilder zu finden. Für die eigentliche Erosionsklassifizierung wurden zwei Methoden evaluiert: *Local Binary Patterns (LBP) + kNearest neighbor Klassifizierung* sowie die Klassifizierung mittels *Convolutional Neural Network (CNN)*. Der Datensatz für die Klassifizierung bestand aus 2405 Bildern. Die daraus resultierenden 4810 Becher wurden anhand des Erosionsgrades von null bis vier bewertet. Aus den fünf Klassen ergibt sich eine Baseline von 20%. Die *LBP + kNearest neighbor Klassifizierung* erreichte eine Klassifizierungsgenauigkeit von 32.03%. Das gewählte CNN Model, eine light Version der *Xception* Architektur übertraf die *LBP + kNearest neighbor Klassifizierung* mit 58,29% Klassifizierungsgenauigkeit bei weitem. Als größtes Problem wurde die Varianz in der Bewertung der Erosion durch das Wartungspersonal identifiziert. Gründe dafür sind: keine objektiven Bewertungskriterien wie etwa Erosionsfläche in mm^2, die Bewertung durch unterschiedliche Personen, der Wechsel von der Bewertung des Allgemeinzustands des Bechers hin zur "reinen" Erosionklassifizierung, sowie zu viele Klassen für die Bewertung. Die angesprochenen Punkte wurden durch die Bewertung eines Wartungsmitarbeiters auf Basis von Inspektionsbildern untermauert. Der *IllwerkeVKW* Mitarbeiter erreichte eine Klassifizierungsgenauigkeit von 36%. Das Ergebnis einer Klassifizierungsgenauigkeit von 58,29% zeigt das eine automatische Bewertung des Erosionsgrades möglich ist.

Sworn Declaration

I hereby declare that this thesis was in all parts exclusively prepared on my own, without using other resources than those stated. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis or parts of it were not previously submitted to another academic institution and have also not yet been published.

Dornbirn, June 16 2022

Thomas Kraxner

Contents

1 Introduction **1**
1.1 What is Cavitation? 1
1.2 Motivation and Goal 3
1.3 State of the art 3

2 Experiments **5**
2.1 Data 5
2.1.1 Initial data 5
2.1.2 Extracted data 6
2.1.3 Resulting data sets 7
2.2 Preprocessing 14
2.2.1 Initial Attempts 14
2.2.2 Semantic Segmentation with a U-Net 17
2.2.3 ROI Extraction 28
2.2.4 Adapted rubber sheet model 30
2.3 Erosion classification 34
2.3.1 Local Binary Pattern - k-nearest neighbors classifier 34
2.3.2 CNN Xception - Light 39
2.3.3 Manual Classification 44

3 Results **45**
3.1 LBP - k-nearest neighbors classifier 46
3.2 CNN Xception - Light 48
3.3 Manual Classification 50

4 Summary and Outlook **53**
4.1 Conclusion 53
4.2 Outlook 55

Bibliography **57**

List of acronyms **61**

List of figures **64**

List of tables **65**

List of listenings **67**

A Appendix **69**

A.1 Anaconda Environment 69
A.2 GPU-Server Hardware Spec 70
A.3 Adapted U-net model (keras) 71
A.4 Xception - light model (keras) 73

1 Introduction

IllwerkeVkw is using pumped-storage hydroelectricity to store generated electrical energy and balance the power grid. To store the electrical energy as gravitational potential energy, water is pumped to a reservoir on higher altitude. If there is demand to use this energy, water gets released from the reservoir.[1] This water then runs through pelton wheels which are powering a generator. This process leads to cavitation on the buckets of the pelton wheels manifesting in erosion. Initial erosion caused through the cavitation effect will result in stronger cavitation and therefore will accelerate the erosion of the metal (martensitic stainless steel).

1.1 What is Cavitation?

Cavitation is a phenomenon which occurs when liquids are under static pressure, mostly caused through fast moving objects. There are two types of cavitation: inertial cavitation and non-inertial cavitation. By referring to cavitation in this thesis, inertial cavitation is meant. Inertial cavitation applies to propellars, pumps and several other things including pelton wheels which are subject of this thesis. The law of Bernoulli says that the static pressure of fluids getting lower with increasing speed. If the static pressure reduces below the liquids vapour it leads to vapor-filled cavities ("bubbles"). When these bubbles are dragged to areas with a higher pressure they implode. Each implosion generates a small shock wave. The continuous implosions on the metal creates a cyclic stress which leads to wear on the metal. An example of this wear on a bucket of a pelton wheel is shown in Figure 1.2.[2]

[1] `https://en.wikipedia.org/wiki/Pumped-storage_hydroelectricity`
[2] see also `https://en.wikipedia.org/w/index.php?title=Cavitation`

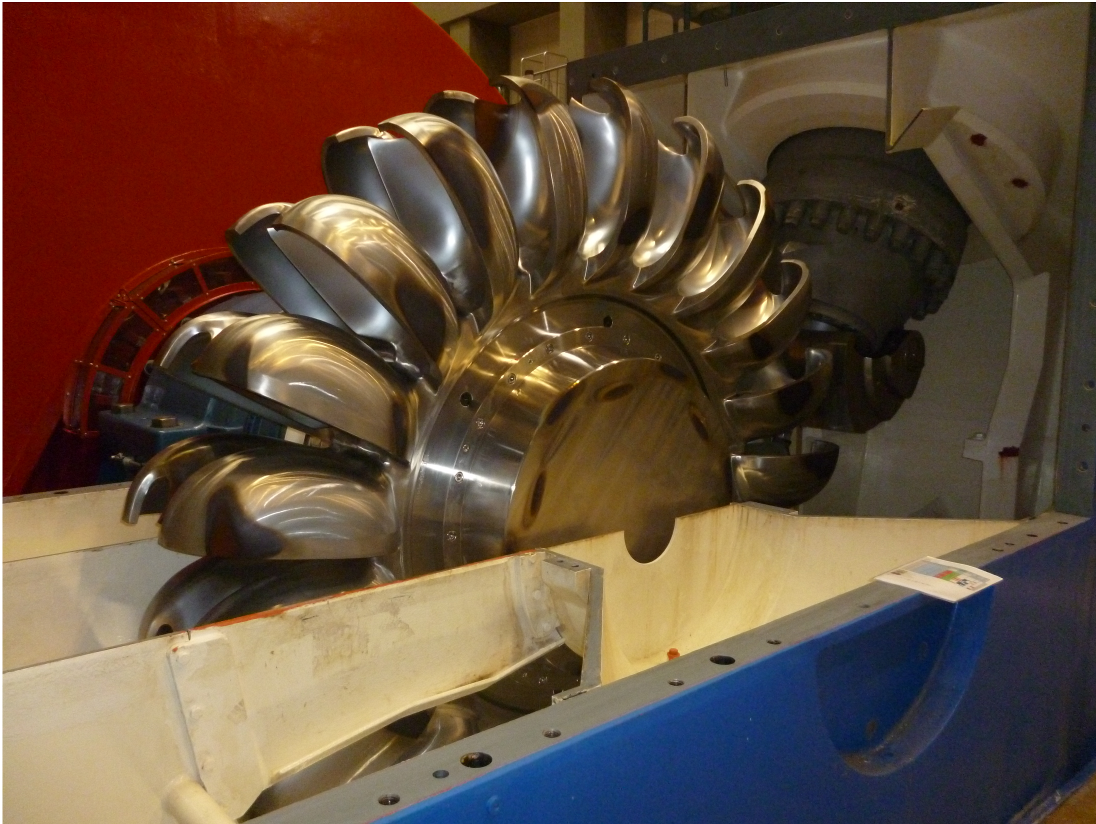

Figure 1.1: Open pelton wheel on inspection day

Figure 1.2: Erosion caused by cavitation on the backside of a bucket of a pelton wheel

1.2 Motivation and Goal

To deal with the problems caused by the cavitation the pelton wheels are inspected every three to six months. If small erosion is spotted the affected area gets polished. If there is strong erosion, material has to be removed and the geometry of the wheel must be corrected. To do so the wheel has to be removed completley. If inspections are omitted a bucket or parts can break off, resulting in high damage and harming people in the power station. There are several problems with this fixed inspection cycle. First the cavitation can increase until inspection day. Each inspection means a downtime for approximately eight hours. Because of the repetitive nature of these inspections they are error prone and time consuming for the maintainance personnel. Another thing to mention is that due to their size not all pelton turbines can be entered to take direct photos of the buckets. *IllwerkeVkw* is currently in the evaluation phase of a mounted camera installment inside the casing to take photos of the buckets during the runout of the pelton wheel.

The Goal of this thesis is it to examine ways of automated grading of erosion through cavitation with computer vision and machine learning methods, to outweight the previous mentioned problems. Creating a full industrial solution is outside the scope of this thesis.

1.3 State of the art

Current methods in detecting and grading cavitation generally uses recordings of high frequencies with a Acoustic Emission (AE) sensor of the testsubject during operation, followed by signal processing methods. Most of the methods found during research for this thesis also use high speed cameras to gather visual data. Despite the similarities, the methods differ in the test setup, mostly caused through the different characteristics of the testsubject and how the data is processed.

Research is done with additional sensors to measure the water pressure, and using the highspeed camera to detect cavitation clouds with shadow photographs by subtracting the images during operation with images when the test subject is idle (Li et al., 2021). There are methods where the highspeed camera is used to detect changes in the vortex or bubbles in the liquid (Xu et al., 2021, Katz,

2018a). The additional visual surveillance of the cavitation process is key to get robust results in cavitation detection (Zhang, F. Lu, and L. Lu, 2019). There is also research which shows that traditional signal processing of AE sensor data can be achieved with a CNN (Katz, 2018b). The main thing the methods mentioned above have in common, is that they all detect cavitation during operation.

The approach of this thesis is to detect erosion caused by cavitation, instead of detecting the process of cavitation. Images are taken when the pelton wheel is idle without any other sensors. The reason for this decision is how the pelton wheel works (see Figure 1.3). Due to the fact that the pelton wheel is not completley under water it is impossible to detect a vortex or bubbles. Getting a clear field of view onto the buckets during operation is not possible, because of the splashing water. This means detecting cavitation clouds via image substraction of the idle pelton wheel is not applicable.

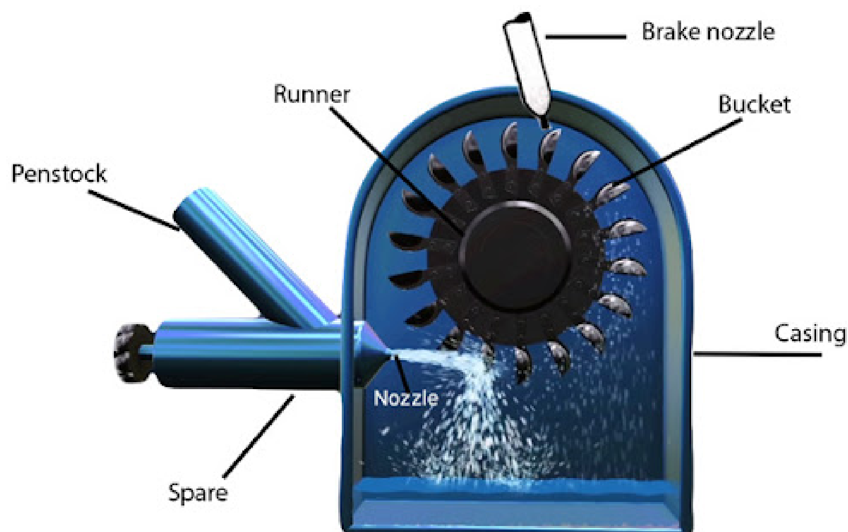

Figure 1.3: Pelton wheel component, source: `https://www.airandhydraulic.com/2020/09/components-of-pelton-turbine.html`

2 Experiments

The description of the experiments is structured by various topics which are not in chronological order. Section 2.1 is about the data provided by *IllwerkeVKW* plus a description of the erosion classes used in the classification experiments. At the end of Section 2.1 the datasets which were created through the preprocessing (see Section 2.2) are explained. These datasets embodied the ground truth in the erosion classification experiments. Since there were multiple objects on the raw images, the classification would not have been possible on the raw images. The preprocessing section outlines the way of finding methods to create usable images from the initial provided ones. Section 2.3 describes the main topic of this thesis: Classification of erosion.

2.1 Data

2.1.1 Initial data

The data for the following experiments was extracted from inspection protocols of the last five years from six specific pelton wheels. Each wheel has 21 shovels with two buckets. An inspection protocol consists of a report (pdf) containing erosion grades for every bucket, and a *jpeg* image of each shovel. If erosion was spotted during the inspection the maintenance personnel marked each area with permanent marker before taking the photo. These markings could lead to problems in the later training of the erosion classification. Nearly all images were taken with an *iPhone 8* (see Table 2.1). All images were taken manually without a tripod or anything comparable, so they all differ in angle, lighting and position. The inspection protocols stated that five grades should be used to classify the level of erosion (see Table 2.2).

Spatial Dimension	4032×3024 px
Flash model	off
Focal length	4mm
35mm focal length	28

Table 2.1: Specification of the *iPhone8* images

Grade	**Description**
0	no erosion
0.5	change in color
1	minor erosion
2	erosion
3	strong erosion

Table 2.2: Defined grading of cavitation erosion

2.1.2 Extracted data

For each inspection a digital inspection protocol (see also Section 2.1.1) was created by the *IllwerkeVKW* maintenance personnel. To create a dataset which can be used to implement an automatic erosion classification, all images were extracted from the inspection protocols. During the extraction a *csv* file was manually created to store the image filename and the corresponding erosion grades for each bucket. During the five years of inspection several new grades were introduced by the maintenance personnel (like 1.5, 3++, 2–). This led to the decision of rounding these ratings to definite grades (see Table 2.3). To get a

Grade	**Description**
0	no erosion
1	minor erosion (containing 0.5 and 1 of the original inspection protocols)
2	erosion (containing 1.5 and 2)
3	strong erosion (containing 2.5 and 3)
4	extreme erosion (containing 3.5, 4, 3++ and 3+++)

Table 2.3: Mapping of inspection protocol grades and used grades

grasp of the different classes in the dataset, some examples with labels are shown in Figure 2.5.

The final numbers of this extraction process were 2405 images in total, which means *4810 labeled buckets for training.* These images are containing the buckets

of interest and a lot of other things in the background which are not of interest for the erosion classification. In order to remove this background, semantic segmentation was used to separate the main shovel from the initial images (see Figure 2.1). After this first preprocessing step there were 2270 images left (see Limitations in Section 2.2.2). The final numbers are indicating a slight class imbalance in the provided data (see Table 2.4).

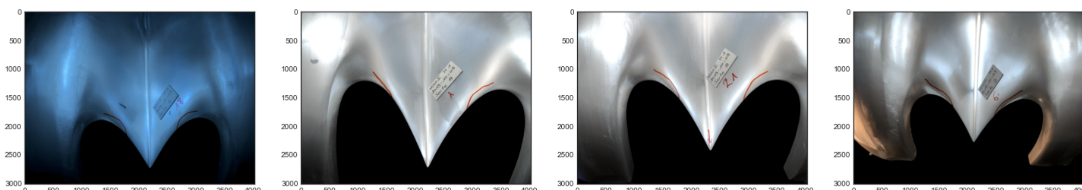

Figure 2.1: Segmented images where the background pixels are set to black, see Section 2.2.2

Grade	shovels	left bucket	right bucket	Percentage of total
0	1016	599	417	22.32%
1	939	495	444	20.63%
2	647	292	355	14.21%
3	1136	570	566	24.96%
4	814	320	494	17.08%

Table 2.4: Numbers on initial dataset

2.1.3 Resulting data sets

The resulting datasets for erosion classification are described in this section.

Test dataset

To avoid any bias through markings or other attributes on the buckets in the final scoring of the methods, a test dataset was created. Two buckets were randomly selected from each of the six pelton wheels contained in the dataset. These selected buckets were then removed from the training data in order to create a test-dataset for the final evaluation of the used methods. To ensure that they will not be used in any training this was done right after the initial extraction. This

extraction led to a test dataset of *434 buckets in total for the final evaluation*. In order to use them in the different classification methods the needed preprocessing steps were applied to use it as *final evaluation dataset*.

ROI segmented

This dataset is the result of the process described in Section 2.2.3, applied to all images of the initial dataset. The total numbers of this extraction are shown in Figure 2.3. Some sample Region of Interest (ROI) images are provided in Figure 2.2, already scaled to 360×360 px for later use with a CNN.

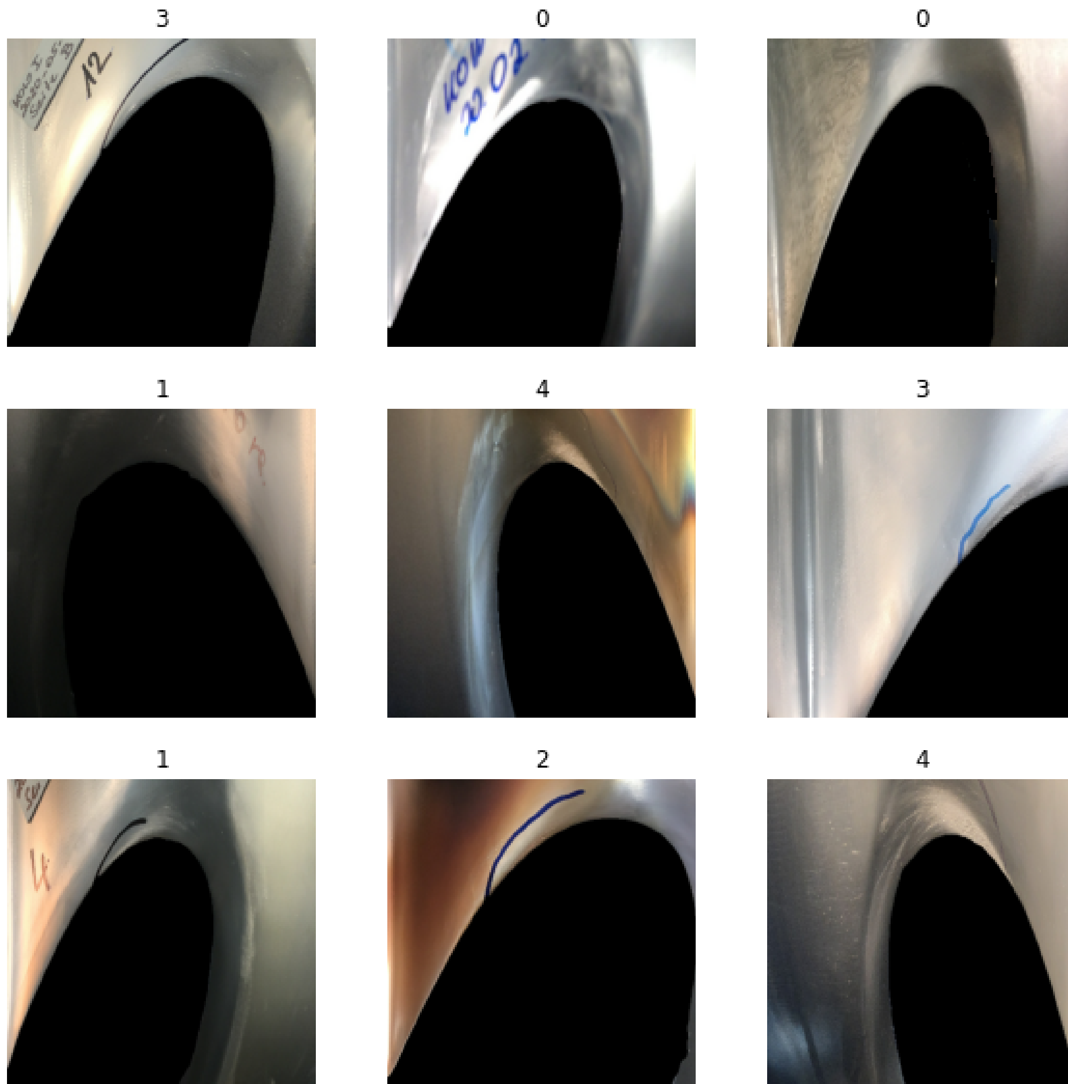

Figure 2.2: Samples ROI extraction on segmented image, with classes

To increase the number of samples for training, data augmentation was applied to the dataset. The same augmentations (rotate, flip, zoom in) as described in the U-Net Implementation (See Section 2.2.2) were used. Figure 2.4 shows the numbers of samples per class after applying augmentation. Five augmentations were created for each image, resulting in an sixfold increase of the images. The distribution stays the same.

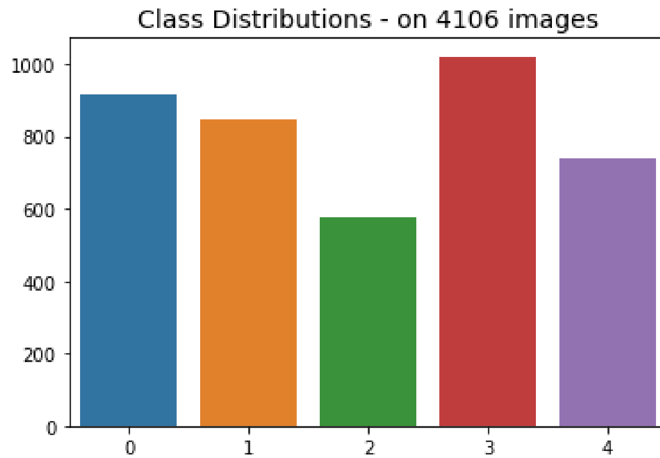

Figure 2.3: Numbers of samples per class in dataset

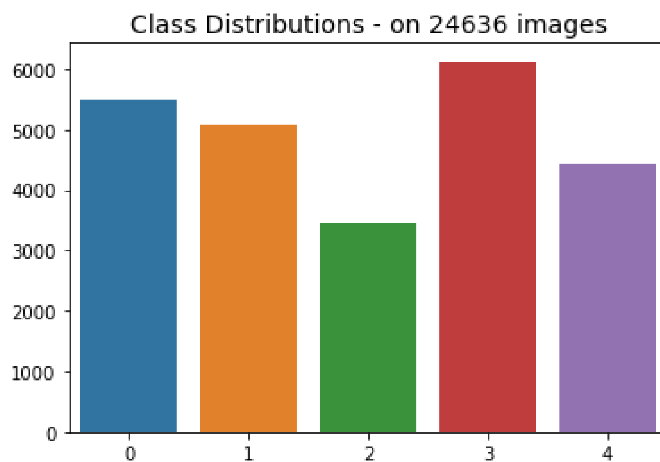

Figure 2.4: Numbers of samples per class in augmented dataset

ROI without segmentation

This dataset was created with the enhanced version of ROI extraction where the bounding boxes are cropped on the real images instead of the segmented ones. The segmented images are just guiding the extraction described in Section 2.2.3. The numbers are the same as in Section 2.1.3. Samples are shown in Figure 2.5.

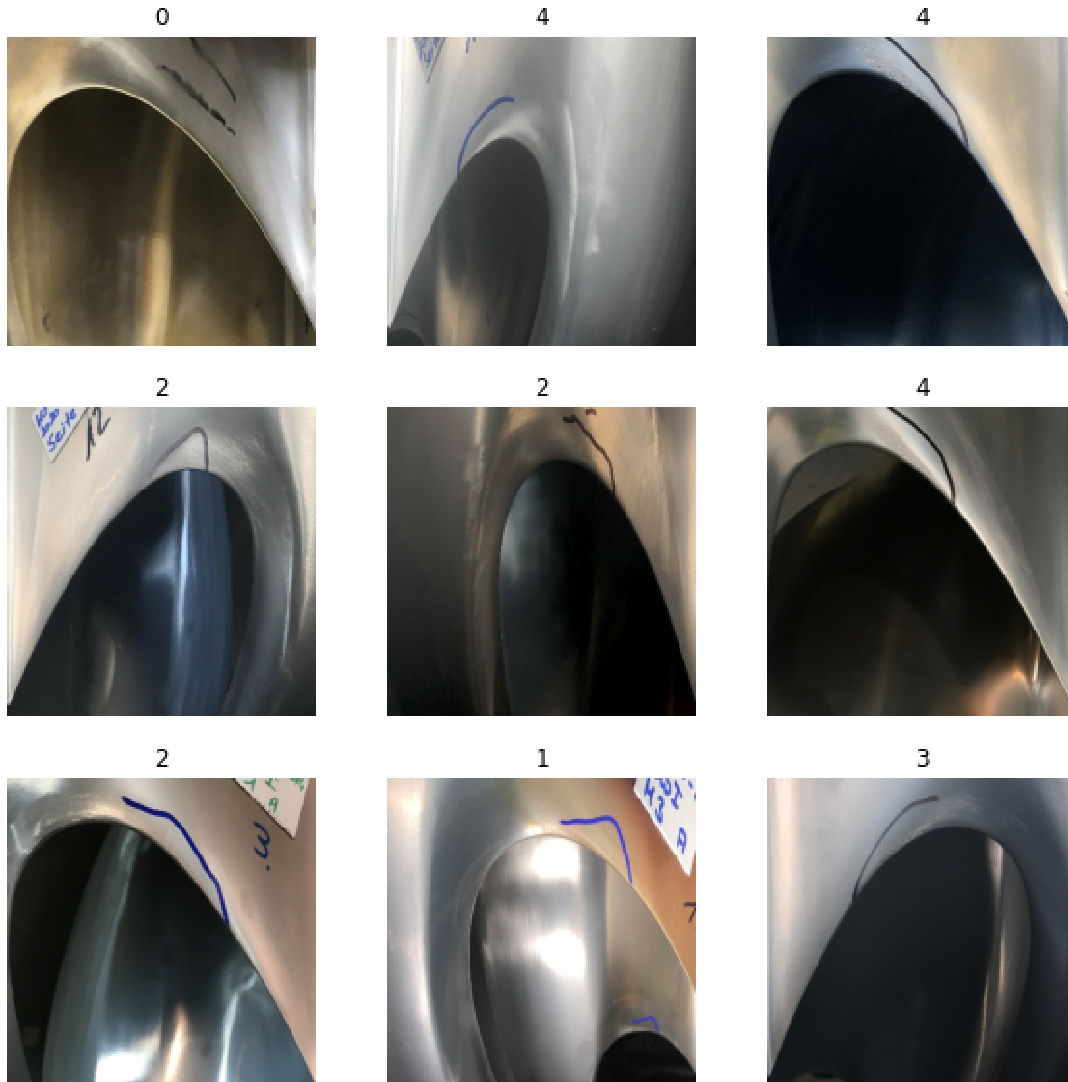

Figure 2.5: Samples ROI extraction on real image, with classes

Unwrappings

In order to get an even more precise ROI this dataset was created with the adapted Rubber sheet model (RSM) (see Section 2.2.4). The unwrapping was done with a initial offset of 20 pixels for each axis to avoid bucket background pixels in the result caused by bad ellipse fitting. Thickness (or axis increasement, y-axis in rectangular polarform) was set to 100 pixels and four samples per 1°. An example of the resulting images can be seen in Figure 2.6. Because of the limitations of the method (see Limitations in Section 2.2.4), there was some data loss in this processing step. The final numbers are presented in Figure 2.7.

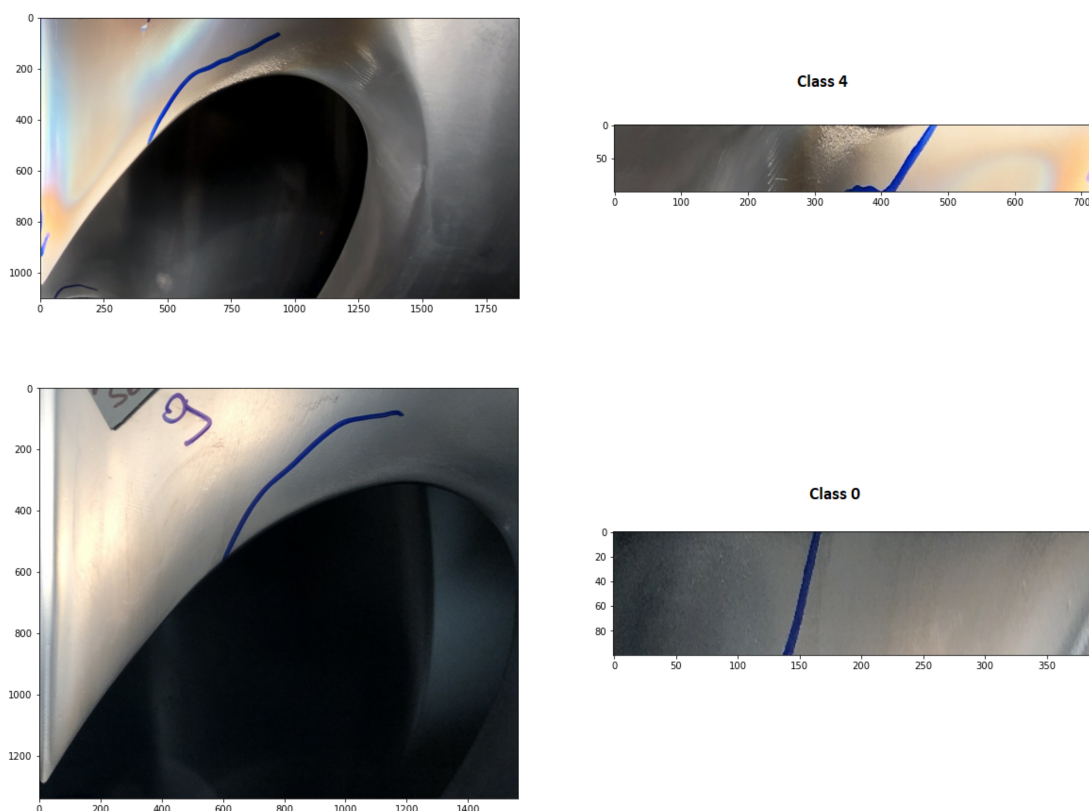

Figure 2.6: Examples of source (left) and result of unwrapping (right), for class zero and class four

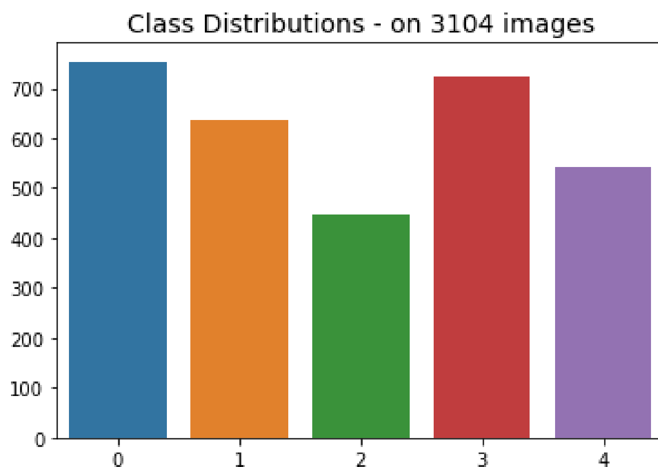

Figure 2.7: Numbers of samples per class in unwrappings dataset

As before augmentations were created. For this dataset just the flip augmentations were used. The zoomin and the rotate augmentation (containing a zoomin operation) were skipped to not create augmentation with lesser textural information. Figure 2.8 shows the numbers for each class after creating the three flipping augmentations per image.

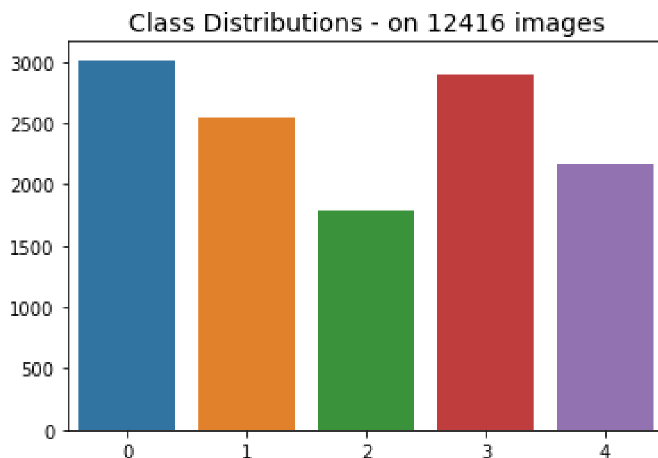

Figure 2.8: Numbers of samples per class in augmented unwrappings dataset

2.2 Preprocessing

There are two tasks to fullfil during preprocessing:

- separate left and right bucket, since every bucket is graded individually
- get a precise ROI (see green and red area on Figure 2.9)

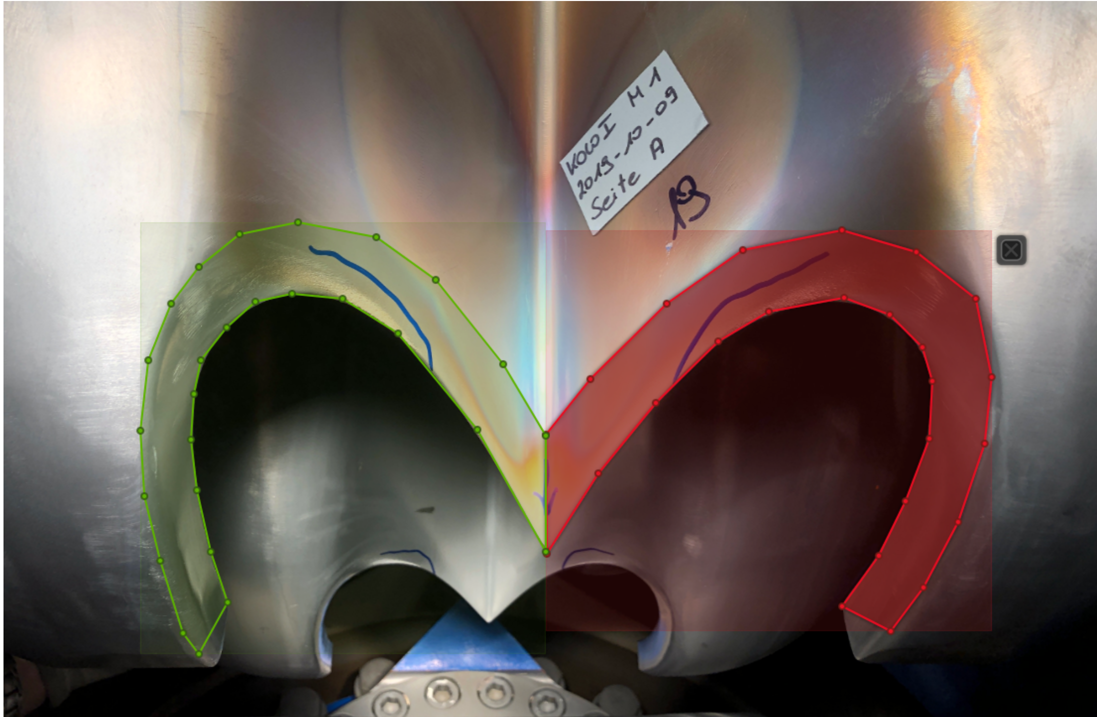

Figure 2.9: Area where erosion through cavitation manifests

Due to the reflections of the metal and other objects this process was not straight forward. The used methods are explained in the following sections.

2.2.1 Initial Attempts

OpenCV

The initial attempts to detect the outer edges of the shovel on the images with *opencv*[1] were not successful. Several attempts with roughly the same procedure were performed. This edge detection procedures consisted of:

[1] `https://docs.opencv.org/4.x/index.html`

1. use some built-in blur function (`cv2.GaussianBlur`, `cv2.medianBlur`, ...)
2. convert to grayscale (`cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`)
3. apply a threshold (`cv2.THRESH_BINARY`, `cv2.THRESH_OTSU`)
4. try to find contours (`cv2.findContours`)

All attempts failed (see Figure 2.10), because the method always detects countless contours on the threshold versions of the blurred images.

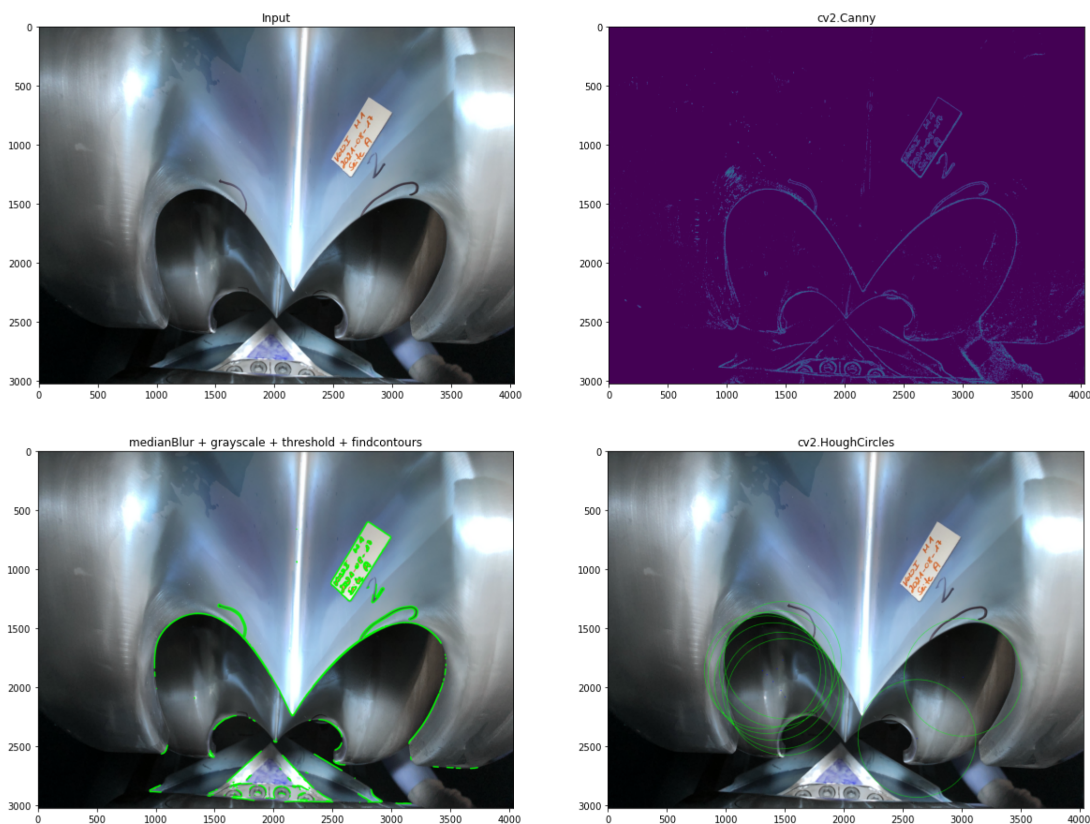

Figure 2.10: Initial tries with *opencv*

Generalized Hough Transformation

The Generalized Hough Transformation (GHT) is an improved version of the Hough Transformation (see Duda and Hart, 1972). The main concept of the Hough Transformation is to find the parameters of a shape. For instance in the

case of a line the shape is defined by $y = ax + b$, so the Hough Transformation finds the parameters a and b, via intersections in the parameter space also called Hough Space, where a and b are the axis.

The Hough Transformation is limited to simple shapes with few parameters like lines, circles and parabolas. The method tries to find mappings between the image space and the Hough space. Every pixel vote for its corresponding reference points in respect to the chosen shape (f.e.: pixels which will be on the same line). The pixels with the most votes then are representing the shape found with the Hough Transformation. The detection of complex shapes can be achieved by an composition of the mappings from simpler shapes (D. H. Ballard, 1981). This method is called Generalized Hough Transformation.

Currently there is no built-in feature for GHT in *opencv* (neither in *python* nor in *c++*). However there is an implementation in *c++* available on github[2]. At the day of writing the repository featured four commits where the last one was authored eight years ago. Since the current state is not fully compatible with the current *opencv* version *4.5.1+dfsg-5*, the repository was forked and adaptions were made[3]. With this method it was possible to find the bucket shape on another image with different scaling and different positioning, by rerunning GHT with different scaling and rotation see Figure 2.11. It took *779 seconds* for 360 rotations with $1°$ change and scaling from 0.5 to 2.0. To identify the shape of a shovel in arbitrary images the tilt has to be compensated by perspective transformations[4]. This approach could theoretically work. However, due to all the possible alterations in rotation, scaling and perspective transformations this solution would be hardly better than a brute-force approach to fit the contour of the template. For this reason it was not pursued any further.

[2]`https://github.com/jguillon/generalized-hough-tranform`
[3]`https://github.com/Kraego/generalized-hough-tranform`
[4]`https://stackoverflow.com/a/33502869/11473934`

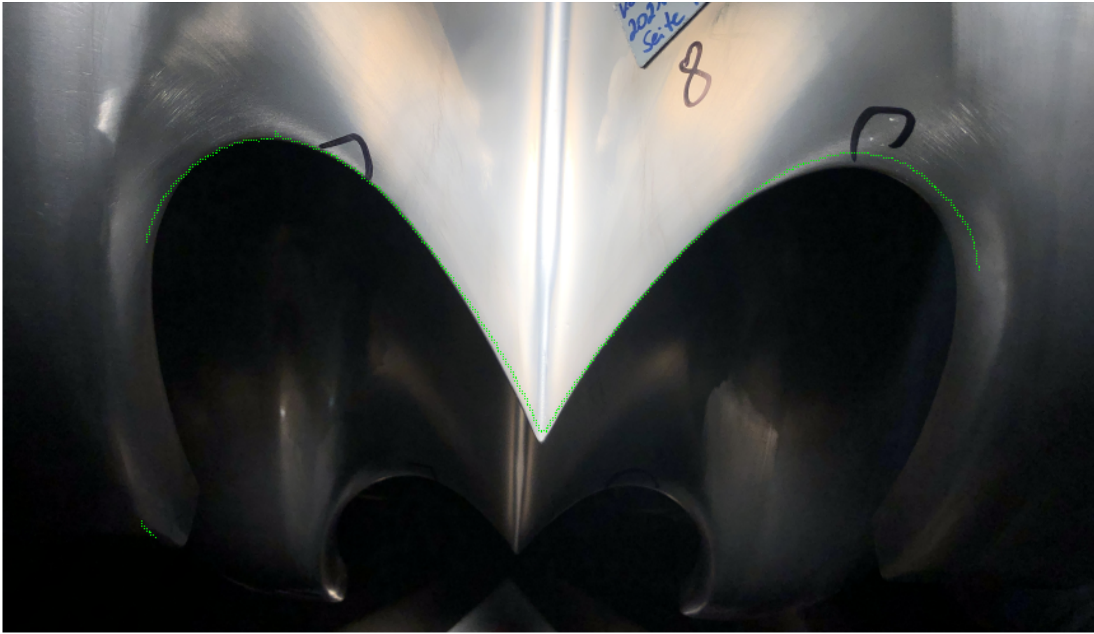

Figure 2.11: GHT on sample image with nearly same tilt

2.2.2 Semantic Segmentation with a U-Net

To counteract the problem of detecting edges or shapes with all the objects outside the ROI, the decision was made to use semantic segmentation. This method assigns a class to each pixel in the image (see also Ghosh et al., 2019 p. 1 and 2). In this case two classes are needed. Class one is foreground - in other words the main shovel. Class two is background. Most pictures also showing the next shovel, which is not of interest.

One implementation of solving semantic segmentation is the *U-Net* architecture. *U-Net* is a fully convolutional network, which has the shape an "U" (see Figure 2.12).

> The main idea is to supplement a usual contracting network by successive layers, where pooling operations are replaced by upsampling operators. Hence these layers increase the resolution of the output. A successive convolutional layer can then learn to assemble a precise output based on this information. (Ronneberger, Fischer, and Brox, 2015)

In the contracting path each layer reduces the spatial dimension and increases

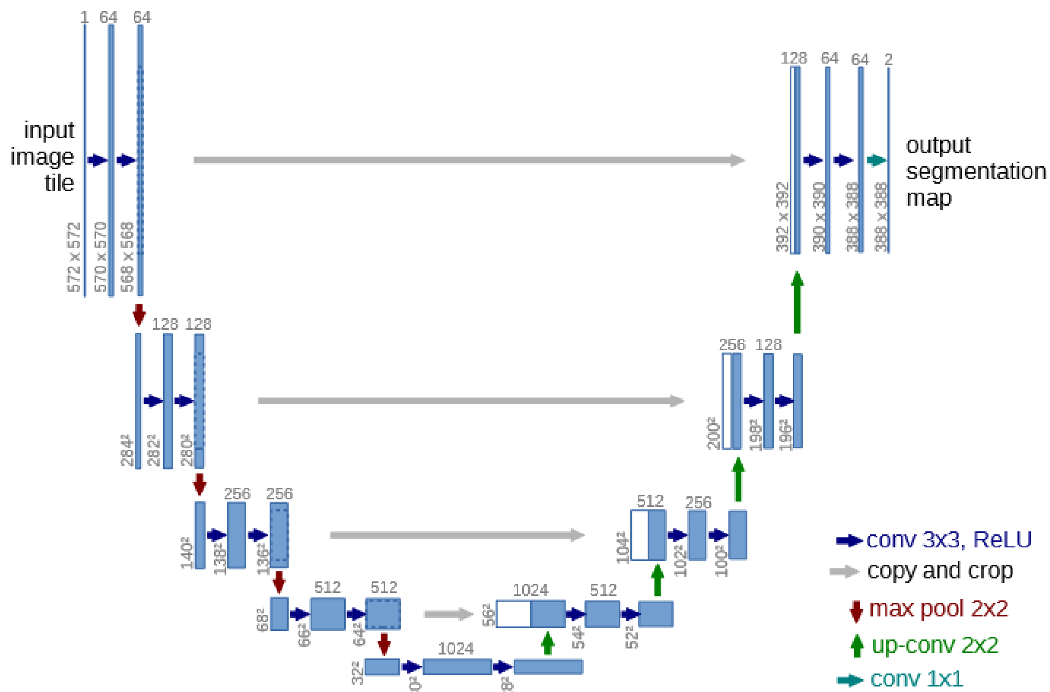

Figure 2.12: U-net architecture from original paper (Ronneberger, Fischer, and Brox, 2015)

the number of feature channels via a convolution plus a Rectified linear unit (ReLU) and a max pooling operation. The expansion path performs the upsampling with upconvolutions and concatenations of the corresponding layer from the contracting path.[5]

Implementation

In search for a suitable *U-Net* model, the implementation of *Peter Höngigschmid* was found on *kaggle*[6]. The implementation follows the model proposed in the original *U-Net* paper. It uses *keras*[7], a easy to use framework for deep learning in python. In addition to improve the performance, dropout layers were added after the pooling layers in the contracting path and after the concatenation in the expansion path. Dropout is a regularization method. The dropout layers

[5] `https://en.wikipedia.org/wiki/U-Net`
[6] `https://www.kaggle.com/code/phoenigs/u-net-dropout-augmentation-stratification/notebook`
[7] `https://keras.io/`

reduce co-adaptation of the neurons during training, which leads to overfitting, by randomly setting the activations of a selected subset of neurons to zero (Srivastava et al., 2014).

The following adaptions were made to the model from *kaggle*:

- Three color channels (RGB color model (rgb)) instead of one (grayscale), initial experiments have shown slightly better results when training with rgb images.
- Input layer size (image size) was increased from 128×128px to 512×512px, since the used images have a higher resolution (4032×3024px).
- Deeper Network: six contracting/expansion blocks vs. four in the *kaggle* implementation. This also means more parameters which results in a longer training. Since there is enough data to train the model, there is no problem with overfitting. The training and prediction with the trained model is also not time critical.

The final model with these adaptions is shown in appendix A.3.

The groundtruth for the training consists of binary masks of the shovel and the input images. To create these masks *VoTT*[8] was used. The result of the manual labeling with *VoTT* was a *json* file containing lists of points of the labeled areas similar to the datastructure of an *opencv* contour. In the need of binary masks, a python script was introduced which creates these from the *json* file (see Figure 2.13). In total 512 images were label this way, 480 images were used for training and 32 images were hold back as test dataset (see Section 2.1.3) for final scoring of the trained model.

To evaluate if the U-Net is capable of performing a semantic segmentation on the given dataset, 50 images were labeled. With the image-flip augmentation of the used implementation, the dataset for training the model consists of 100 images. Figure 2.14 shows that the provided data allows the model to learn how to perform the desired segmentation. Due to this promising results, 480 images were labeled, to train the model.

After the evaluation the model was trained with *480 images*, and the augmentation from the *kaggle* template, which was flipping the images vertical along the

[8] `https://github.com/Microsoft/VoTT`

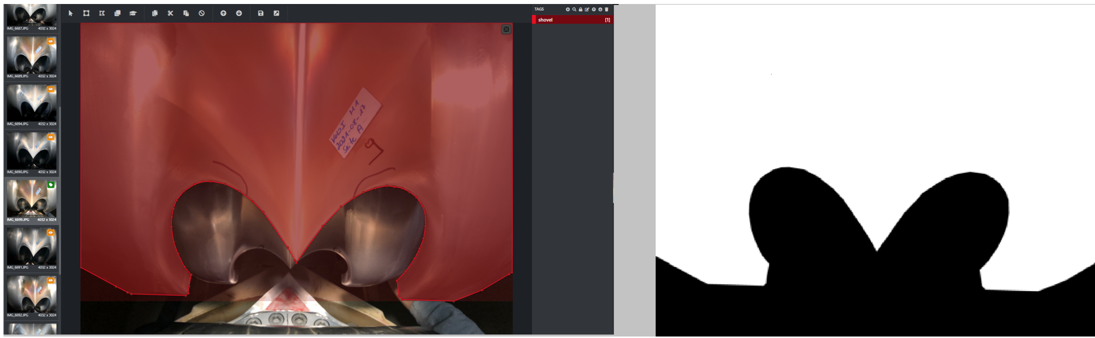

Figure 2.13: Left labeled shovel in *VoTT*, right created binary mask

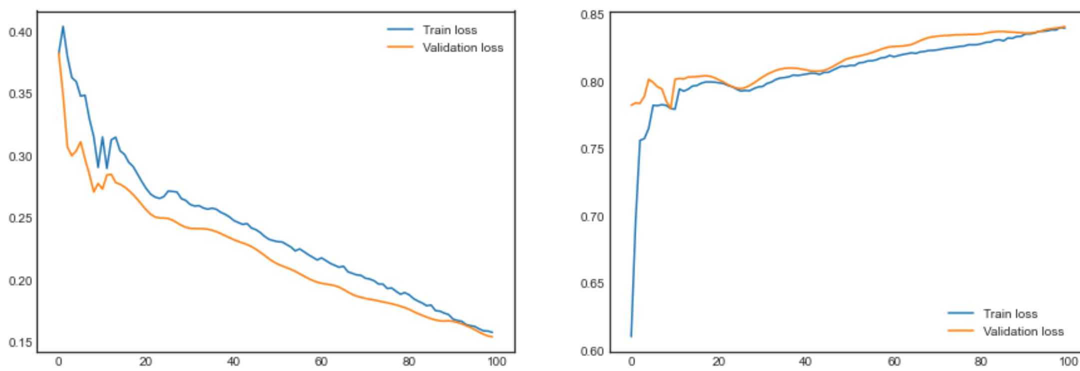

Figure 2.14: U-Net evaluation, 200 images

y-axis. With this augmentation the size of the dataset was *960 images*. The plots in Figure 2.16 are showing a consistent learning with a reducing loss and no overfitting. The training could have been stopped after epoch 40 because there was nearly no improvements in loss or accuracy. But since the training is running on a remote GPU Server (see A.2) the setting of the max epochs for the training of 100 epochs was kept.

To achieve even better results more training data was needed. The first option was to label more images for training which is very time consuming. The labeling of the initial 512 images took several hours. Another option was to use heavy data augmentation. Because of the timeconsuming process of labeling, data augmentation was chosen. Data augmentation is often used during the training of CNN's to artificially create new samples from the given ones to increase

the size of the dataset.[9,10] There are several different augmentation techniques to use on image data. Image augmentation can be separated into four distinct groups which are geometric transformations (rotate, scale, ...), noise injection, color space transformations like brightness adaption or inversion of colors and mixing images (Shorten and Khoshgoftaar, 2019). Since the biggest differences in the dataset are the positions of the shovels, only geometric transformations were used. Due to the higher amount of training data obtained by augmentation the training should result in a better model. *keras* has some built-in geometric transformation augmentations[11], which can be added as layer to the model to do the augmentation in place during training. At the moment of writing this thesis, the used tensorflow version on the GPU-Server (see appendix A.2) did not support these layers. Another reason why the augmentation is implemented explicit outside of the training, was the knowledge that the training would be repeated several times when tingeling with the hyperparameters (dropout rate, network depth, ...) of the model. Moreover, each training would have done the augmentations randomly, which leads to longer trainings, plus it would have introduced some randomness. This randomness would have introduced some uncertainty in comparing the results after tuning the hyperparameters. In this experiment the hyperparameters are optimized manually, since this is just a feasibility study. But there are approaches to optimize the hyperparameters automatically, for instance with genetic algorithms (see Aszemi and Dominic, 2019). The augmentations for the dataset were implemented in *python* with *opencv*. The implemented augmentations are:

- Flipping the image horizontally (x-axis), with `cv2.flip(img, 0)`.
- Flipping the image vertically (y-axis), with `cv2.flip(img, 1)`.
- Flipping the image horizontally and vertically, with `cv2.flip(img, -1)`.
- Random zoom in (remove a frame with random thickness (15 to 60 pixel), and rescale back to original size). The frame cutaway was implemented with python array slicing[12] on the *numpy* array representation of the image. For

[9]Ronneberger, Fischer, and Brox, 2015
[10]Redmon et al., 2016
[11]`https://keras.io/api/layers/preprocessing_layers/image_augmentation/`
[12]`https://python-reference.readthedocs.io/en/latest/docs/brackets/slicing.html`

zoom in `cv2.resize(crop, (w, h), interpolation=cv.INTER_LINEAR)` was used.

- Random rotation (from minus six degrees to plus six degrees), and zoom in to avoid the black border resulting from the rotation. For this the function `cv2.warpAffine()` was used.

The augmentations were performed on the images and their corresponding labels which are binary masks. Figure 2.15 shows examples for each augmentation with the augmented binary mask in green on the alpha channel. With augmentations the new dataset size is *1920 images*. The results of the training (loss, accuracy) nearly show the same values as the initial run with lesser augmentations (see Figure 2.17).

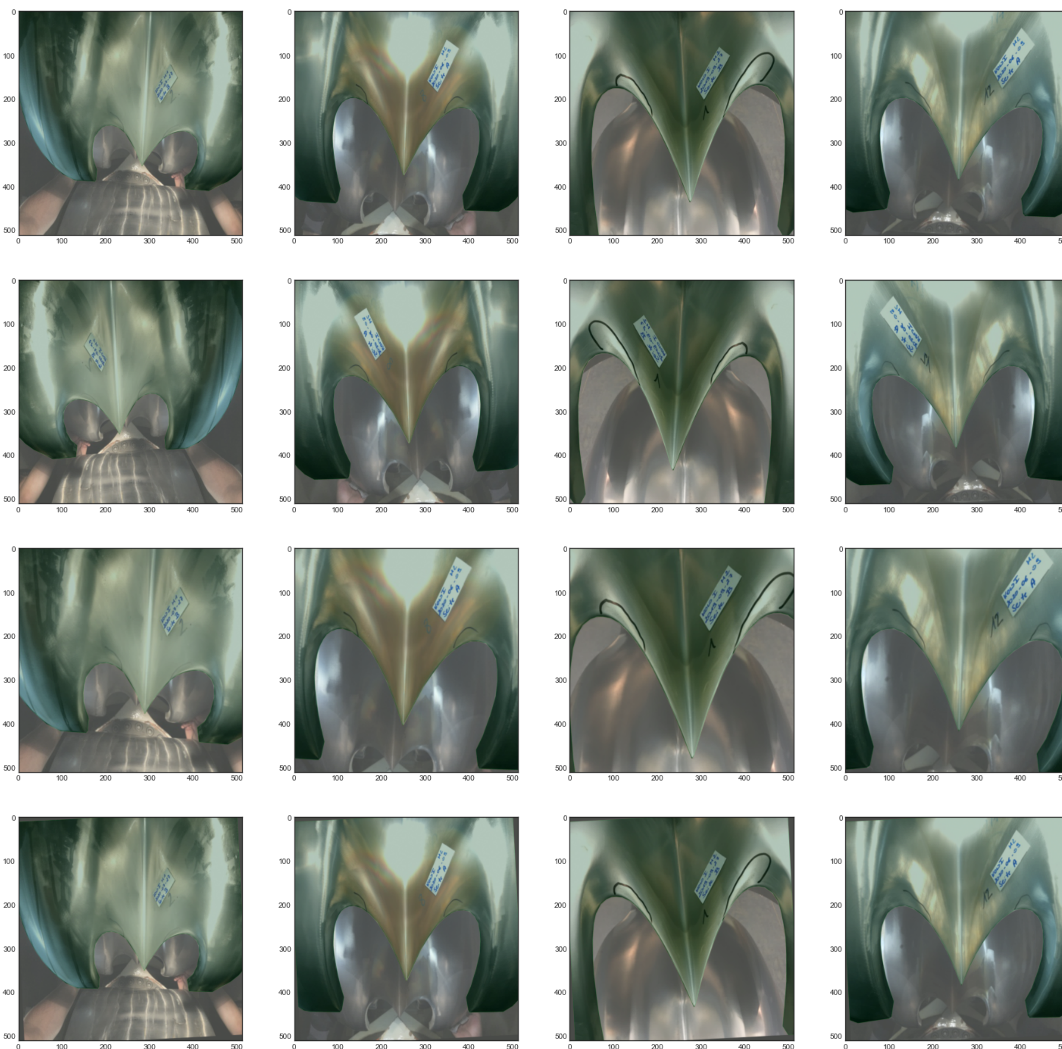

Figure 2.15: Demo of the augmentations

Training and Results

The training was performed with a batch size of 32 images. 75% of the dataset were used for training and 25% were used for loss and accuracy estimation at the end of each epoch. As loss function binary cross entropy was used with the *adam* optimizer (see Kingma and Ba, 2017). To avoid overfitting *EarlyStopping* stops the training after ten epochs without improvement in the loss. The learning

rate is reduced with a factor of 0.1 after five epochs without improvement. The complete training setup is shown in Listing 2.1.

```
model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
early_stopping = EarlyStopping(patience=10, verbose=1)
model_checkpoint = ModelCheckpoint("./keras.model",
                                   save_best_only=True,
                                   verbose=1)
reduce_lr = ReduceLROnPlateau(factor=0.1,
                              patience=5,
                              min_lr=0.00001,
                              verbose=1)

history = model.fit(x_train, y_train,
                    validation_data=[x_valid, y_valid],
                    epochs=200,
                    batch_size=32,
                    callbacks=[early_stopping,
                               model_checkpoint,
                               reduce_lr])
```

Listing 2.1: Training configuration for U-Net in keras

To get a final scoring, Intersection over Union (IoU) or Jaccard index[13] is used as metric on the testdataset consisting of 32 images. The 32 images were extracted with `random.sample()` from the initial 512 images, to have 480 images for training (15 batches with 32 images).

$$IOU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{2.1}$$

In the current scenario there are two classes: shovel and background. The IoU is calculated for each class, then the mean IoU for all classes is returned. The *Area of Intersection* counts the pixels were the groundtruth class and predicted

[13] https://en.wikipedia.org/wiki/Jaccard_index

class are equal. *Area of Union* is the union of all pixels of the class for which the IoU is calculated. An IoU was calculated for each image, after that the mean and median for all images was determined. The used python implementation of IoU can be found on *github*[14]. Since the predictions of the model are not in *One Hot Encoding* the soft version of the metric is used (see listing 2.2). The default (standard) metric type of the function expects a one hot encoded prediction.

```
iou = metrics_np(
        validation_mask, # binary 0.0 or 1.0
        prediction, # 0.0 to 1.0
        'iou',
        metric_type='soft',
        drop_last = True,
        mean_per_class=False,
        verbose=False)
```

Listing 2.2: Calculation of IoU

The first training was done with a dataset consisting of 480 images which are augmented by flipping vertically wich gives *960 training images*. The training history (see Figure 2.16) shows that there is no overfitting, training loss and accuracy curve follow each other. The lowest validation loss was 0.025 and the highest accuracy 0.965. IoU results are listed in Table 2.5.

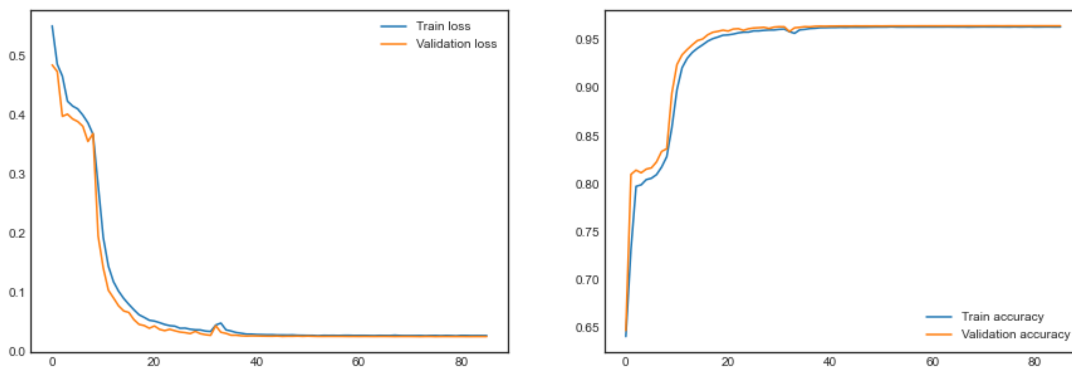

Figure 2.16: U-Net training, 960 images

A second training performed on *1920 images* showed nearly the same results.

[14] https://gist.github.com/ilmonteux/8340df952722f3a1030a7d937e701b5a

The lowest achieved value for loss during training was 0.020 and the highest accuracy 0.962. These numbers not showing any significant improvement compared to the previous training (see Figure 2.17). Table 2.5 shows, that the IoU slightly improved by 0.006. Some samples of the segmentation by the final model are presented in Figure 2.1. A double check on the random selected test dataset confirmed that there were no images with bad segmentation as shown in Figure 2.18 and Figure 2.19 used for the final scoring. This explains the high IoU score.

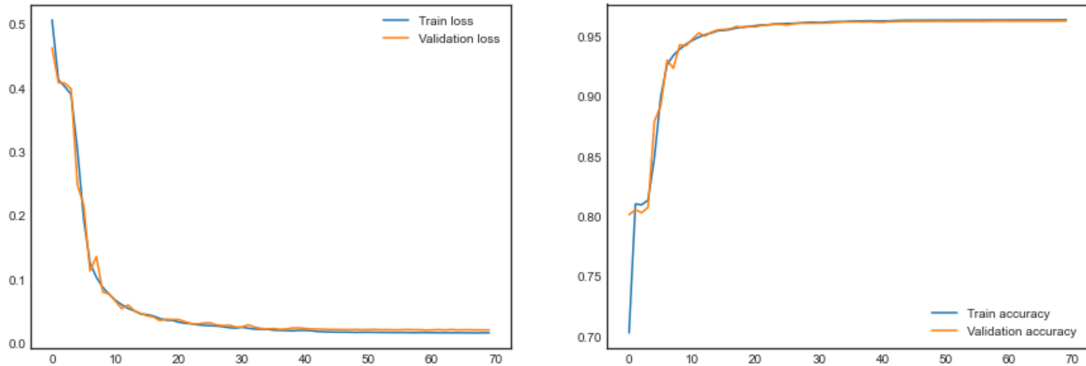

Figure 2.17: U-Net final training, 1920 images

Training	**Mean IoU**	**Median IoU**
U-Net	0.9815	0.9834
U-Net heavy augmentation	0.9874	0.9895

Table 2.5: IoU result of U-Net on 32 images

Limitations

After removing the background on all 2405 images with the prediction of the model (and a threshold of 0.5), the results were checked manually. On most of the images the results of the background segmention were nearly perfect (see Figure 2.1). On some images the segmentation was less precise, but still acceptable (see Figure 2.18).

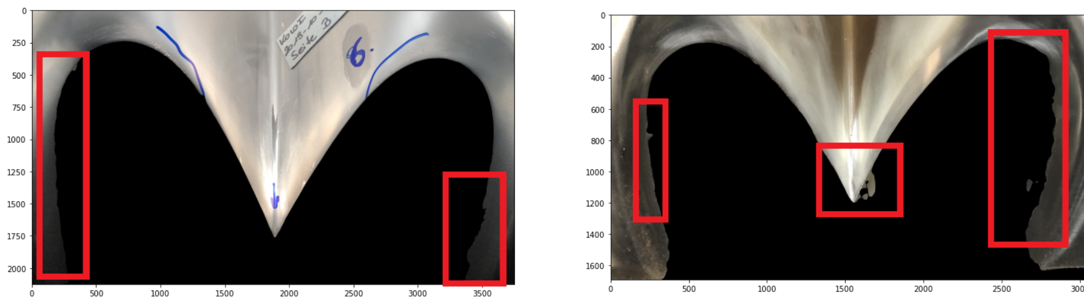

Figure 2.18: Small inaccuracy on background segmentation

The background segmentation completely fails when there is another sharp shovel below the shovel of interest (see Figure 2.19). From the training data the model has learned that there is allways one complete shovel on the images to segment. This scenario applies to 135 images out of the 2405 images. These shovels were removed from the dataset, and are not part of the further classification task.

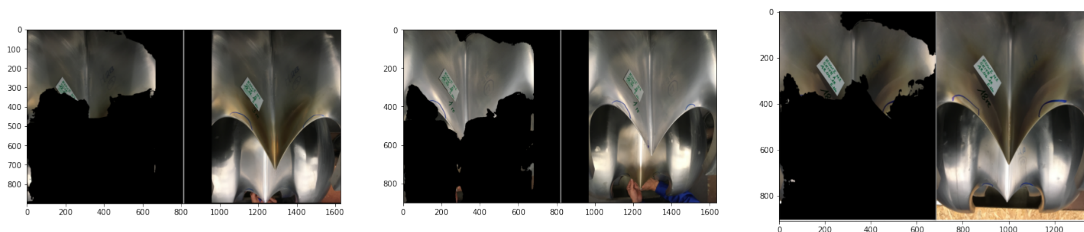

Figure 2.19: Complete segmentation fail when multiple shovels are on the image

2.2.3 ROI Extraction

Since the main shovel is segmented from the background, it is possible to successfully apply the initial approach of contour detection (described in Section 2.2.1). The final solution which creates separate images for each bucket and crop the ROI was implemented in *python* and uses *numpy*[15] and *opencv* functions. The

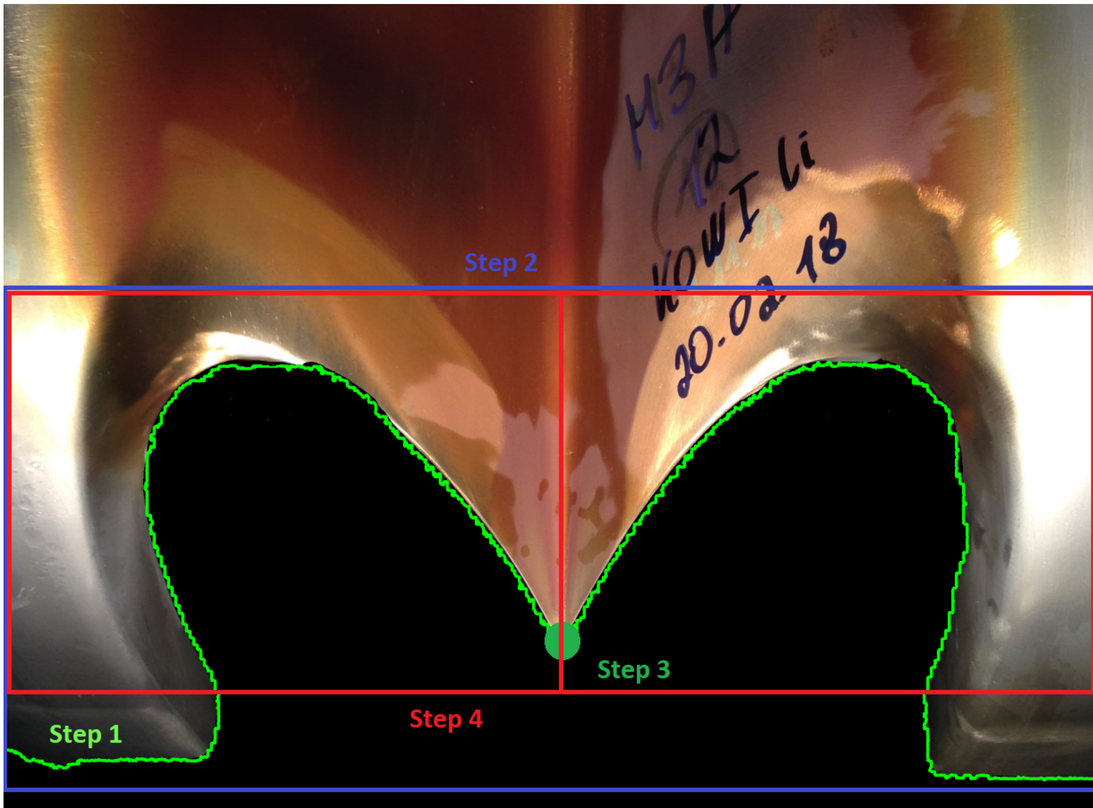

Figure 2.20: Process of bucket separation

following sections will describe the process in detail, guided by Figure 2.20.

Step one: detecting the bucket contour

The first step was the detection of the main contour on the bucket (see neon green line on Figure 2.20). For this the image undergoes a preprocessing which consists of changing the image to grayscale[16], applying a binary threshold[17] and

[15] `https://numpy.org/`
[16] cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
[17] cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)

blurring the image with a Gaussian blur (with a kernel size of three pixels)[18]. After preprocessing the countours on the image were extracted with *opencv*[19]. From the contours found, the one with the biggest bounding box is selected as bucket contour.

Step two: crop the bounding box of the contour

In this step the bounding box (see blue rectangle on Figure 2.20) of the contour[20] was cropped from the original image with python array slicing on the numpy representation of the of the image. A margin of 200 pixels is added to the top of the bounding box.

Step three: detecting the tip of the bucket

In order to detect the tip of the bucket, the countour points were searched for the pixel with the highest y coordinate in the center area (the center 40% in x direction). x refers to the pixel coordinate in width and y in height direction (see green point in the center on Figure 2.20).

Step four: crop separate bounding boxes for the left and right bucket

With the centerpoint two separate bounding boxes were extracted within the cropped bounding box from step two (see red rectangle Figure 2.20). These two cropped bounding boxes are saved under *filename_[L or R].jpg*.

The application of this procedure results in the *ROI segmented* dataset described in Section 2.1.3. Because there were artifacts from the segmentation found during implementing the unwrappings, this process was enhanced. All operations gaining the needed parameters were done on the segmented image and the final cropping, explained in step four above, was done on the original unsegmented image. With this process the artifact free dataset: *ROI without segmentation* was created.

[18] cv2.GaussianBlur(threshold, (3, 3), 0)
[19] cv2.findContours()
[20] cv2.boundingRect(contour)

2.2.4 Adapted rubber sheet model

To create the unwrappings shown in Figure 2.6, the rubber sheet model proposed by Daugman (Daugman, 2004) was altered to work on an ellipse instead of a circle.

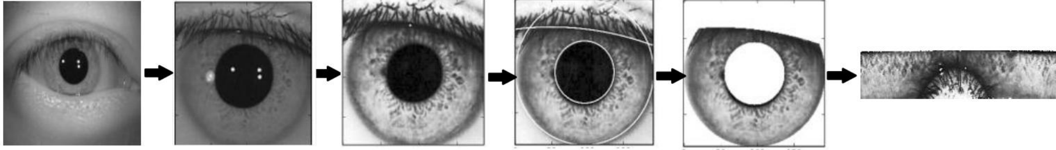

Figure 2.21: Iris Image Processing Process,
from `https://github.com/YifengChen94/IrisReco`

Rubber sheet model

The RSM was created to normalize/encode the segmented iris region (see Figure 2.21) on a image into a normalized polar form (see Figure 2.22). This normalized polar form is rectangular and therefore well suited for the usage with CNNs. The axis of this normalized polar representation are θ, which is the rotation angle around the circle, and the radius `r` on the circle inside the iris region. Dougman's RSM tries the find the source pixels coordinates `(x, y)` for each coordinate (`r`, θ) in the normalized polar form, with the mapping function (see Equation 2.2). x_p, y_p refer to the point on the pupil (inner circle), x_I, y_I specify the corresponding point on the outer iris circle in θ direction.

$$\begin{aligned} I(x(r,\theta), y(r,\theta)) &\rightarrow I(r,\theta) \\ \\ x(r,\theta) &= (1-r)x_p(\theta) + rx_I(\theta) \\ y(r,\theta) &= (1-r)y_p(\theta) + ry_I(\theta) \end{aligned} \tag{2.2}$$

Adaptations to ellipse

On the extracted ROI it is possible to detect the outer contour of the bucket and fit an ellipse[21] with *opencv* on the contour (see Section 2.23). The parameters

[21] cv2.fitEllipse(contour)

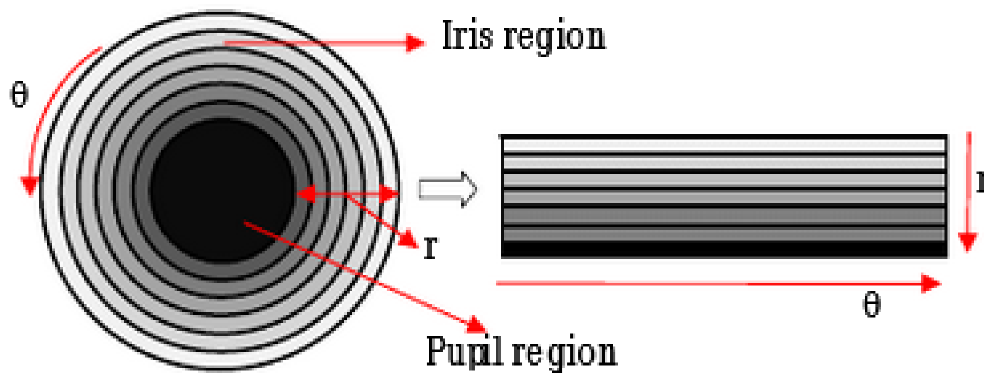

Figure 2.22: Ramkumar, R.P. and Arumugam, S. (2013). Improved Iris Segmentation Algorithm without normalization Phase. International Journal of Engineering and Technology. 5. 5107-5113.

(center, width, height and angle) gained from the fit function are used to perform the adapted RSM described in the following.

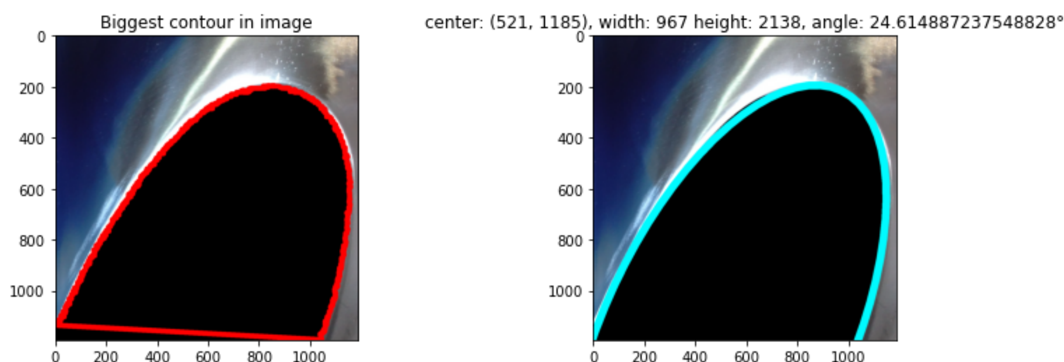

Figure 2.23: Ellipse fitting on ROI

The RSM method increases the radius and the angle of θ to calculate the source pixel `(x, y)` for this coordinate. An ellipse has no radius to increase. To emulate the radius increasement, the two axes of the ellipse are incremented simultaneously with the same value (one pixel) instead. Since there is no explicit outer boundary as in the iris, the axes were increased with a fixed margin. This method lacks the normalization proposed by Dougman and will lead to some variance through different scaling of the input images. After increasement the

radius is calculated for each rotation angle θ via the *Polar form relative to center*[22] formula

$$r(\theta) = \frac{ab}{\sqrt{(a\sin\theta)^2 + (b\cos\theta)^2}}. \tag{2.3}$$

As next step the `x` and `y` offset is calculated from r and θ with the following formulas.

$$\begin{aligned} x_r(r,\theta) &= cos(\theta) \cdot r \\ y_r(r,\theta) &= sin(\theta) \cdot r \end{aligned} \tag{2.4}$$

To take in account the rotation of the ellipse, the values `x` and `y` are adapted with the rotation matrix shown below (be aware that this θ is not the rotation angle of the normalized polar form representation used above).

$$\begin{bmatrix} x'_r \\ y'_r \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix} \tag{2.5}$$

With x'_r and y'_r and the center coordinates (x_c, y_c) of the ellipse, the final pixel coordinate on the given image is calculated.

$$\begin{aligned} x(x_c, x'_r) &= x_c + x'_r \\ y(y_c, y'_r) &= y_c + y'_r \end{aligned} \tag{2.6}$$

The final step is to get the color of the pixel at position `(x, y)` in the input image. In the rectangular result image the corresponding coordinate defined by (`r`, θ) is then set to this color.

The implementation was performed in *python* and was inspired by a *github repo*[23] containing the implementation of the original RSM. The final application of this method finds the bucket contour similar to *step one* of the ROI extraction (see Section 2.2.3) on the image. In order to get the needed parameters of the ellipse an ellipse was fitted on the contour[24]. This was done on the *ROI segmented* dataset. The actual unwrapping is done on the *ROI without segmentation* dataset to avoid segmentation artifacts in the final result. The source code of the adapted

[22] `https://en.wikipedia.org/wiki/Ellipse` - Polar form relative to center
[23] `https://github.com/YifengChen94/IrisReco`
[24] cv2.fitEllipse(contour)

RSM can be found on github[25]. The method was enhanced to find the biggest continuous part of the matched ellipse within the image space. Unwrappings where the θ range is less than 50° are dropped to keep some uniformity in the spatial dimension of the samples.

Limitations

This approach works for most of the images. There was some data loss in the removal of images with less than a 50° - θ range. On some input images *opencv* could not fit any ellipse on the provided contour which also led to some data loss. In numbers, *4069 unwrappings* were created, with a loss of 634 images due to the points mentioned before.

[25] https://github.com/Kraego/Ellipse_RSM

2.3 Erosion classification

For erosion classification three different methods are evaluated. Texture classification with LBP and a CNN. To benchmark the first two methods, images were manually classified by an amateur and an expert. Erosion classification is the main task of this thesis. All preprocessing experiments aim to support this classification through the best ROI extraction possible. The following sections will describe the erosion classification methods in detail.

2.3.1 Local Binary Pattern - k-nearest neighbors classifier

This approach consists of two parts, the feature extraction by LBP (see Ojala, Pietikainen, and Harwood, 1994) and the classification via k-nearest neighbors algorithm[26]. LBP was chosen because it is a simple but powerful feature extraction method. This experiment aims for simplicity and will act as baseline for comparison with the *CNN Xception* classification (see Chollet, 2017).

Local Binary Pattern in detail

This method was initially introduced by Wang and D.-C. He, 1990, and later refined by Ojala, Pietikainen, and Harwood, 1994 with a two-level version called *pure local binary patterns*. The initial version had $3^8 = 6561$ possible values for describing the relation between the center pixel and the pixels in a 3 x 3 neighborhood. In the refined method the possible values are reduced to $2^8 = 256$. LBP is therefore a mapping from $\mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{256}$, with $\mathbb{R}^{m \times n}$ representing the center pixel and it's neighbors, and $\mathbb{R}^{256}$ representing the extracted feature vector for the center pixel. As first step the values of the surrounding pixels are thresholded with the center pixel value (see Figure 2.24 (a),(b)). Each neighbor pixel has a *binary positional value* (clockwise or counterclockwise) $2^n, n \in [0, 7]$ (1,2,4,...). In the second step the result of the threshold from step one is multiplied with the *binary positional value* of the pixel (see Figure 2.24 (c),(d)). These two steps are then repeated for each pixel in the image. For the edge pixels of the image, different strategies can be applied for instance zero padding. There are

[26] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

several extensions to LBP (tLBP[27], OCLBP[28], ...), due to it's simplicity this thesis uses the base version from Ojala, Pietikainen, and Harwood, 1994, with a fixed radius of one pixel and eight neighbors (see Figure 2.24). LBP computes the local

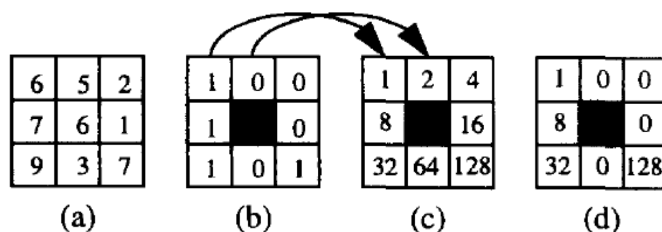

Figure 2.24: LBP proccess[29]

features for every pixel in the image isolated. There are five possible patterns to detect (see Figure 2.25). These occurring patterns are then counted, and stored in a histogram, representing the probability distribution of these patterns (see Figure 2.26). Black dots represent a higher intensity (value) than the center pixel,

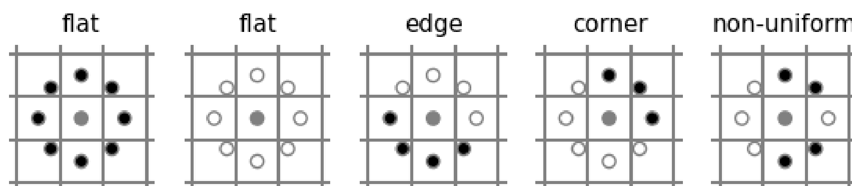

Figure 2.25: Possible LBP patterns, source: `https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html`

white dots stand for lower intensity. Flat regions can be considered featureless. Continuous groups of black and white dots are called *uniform* patterns, including the edge and corner pattern. Patterns with alternating black and white dots are called *non-uniform*. The histogram is the feature vector for the whole image.

[27]Trefný and Matas, n.d.
[28]Barkan et al., 2013.

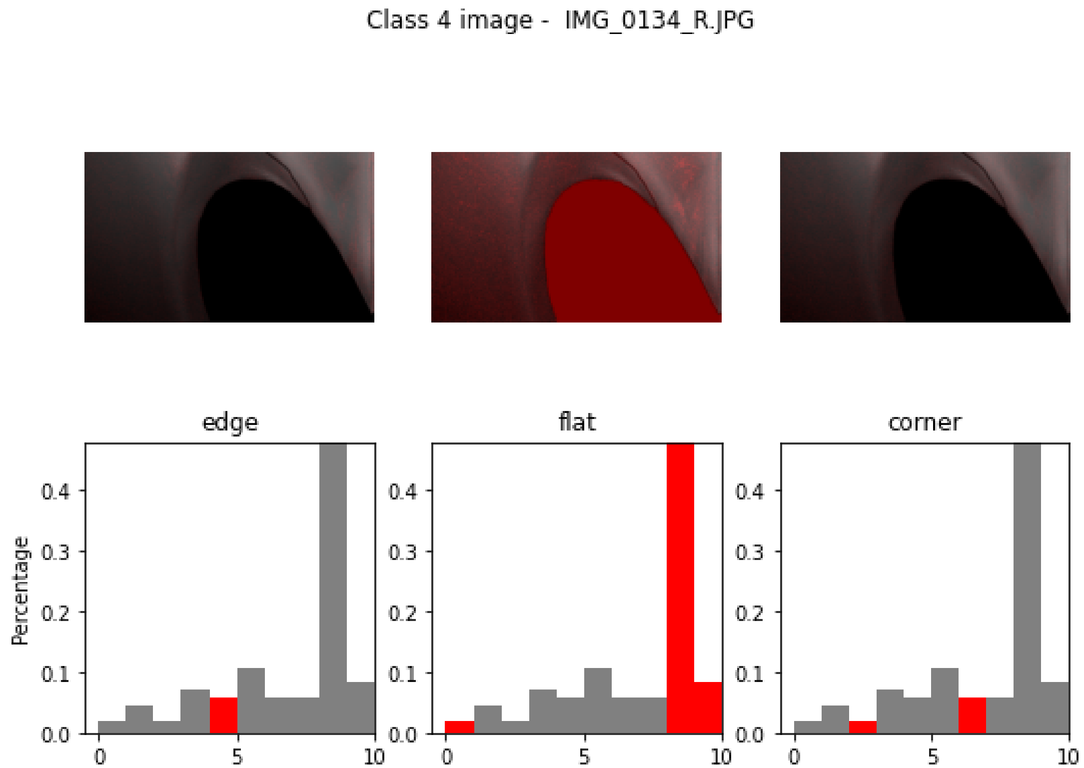

Figure 2.26: Histogram created with LBP on a bucket image

To calculate the LBP of the images the implementation from *scikit-image*[30] was used. For this purpose the images are converted to grayscale before calling `local_binary_pattern`. As initially mentioned the parameters of the original paper (Wang and D.-C. He, 1990) were used, a radius of one and eight neighbors. *Scikit-image* offers different methods to determine the patterns, with different capabilities and limitations. The chosen method is *uniform* meaning the result only includes patterns where all black dots are adjacent and all white dots are adjacent (compare to Figure 2.25), this method is also rotation invariant. The resulting LBP is then stored as histogram (see Listing 2.3).

[30] see: `https://scikit-image.org/docs/dev/api/skimage.feature.html`

```python
from skimage.feature import local_binary_pattern

lbp = local_binary_pattern(image, 8, 1, 'uniform')
n_bins = int(lbp.max() + 1)
hist, _ = np.histogram(lbp, density=True,
                      bins=n_bins, range=(0, n_bins))
```

Listing 2.3: Call of skimage.feature.local_binary_pattern

k-nearest neighbors classification in detail

As in the *pure local binary patterns* paper proposed, the k-nearest neighbors algorithm is used for classification (see Ojala, Pietikainen, and Harwood, 1994). It is a non-parametric supervised learning method based on instance learning. First the distance of the input to all training samples is calculated (instance learning). The input is then classified by it's `k` nearest neighbors. On the example shown in Figure 2.27, the voted class would be triangle for the inner circle ($k = 3$) and square for the outer one ($k = 5$). To avoid tie situations in the class membership voting `k` should be an odd number.

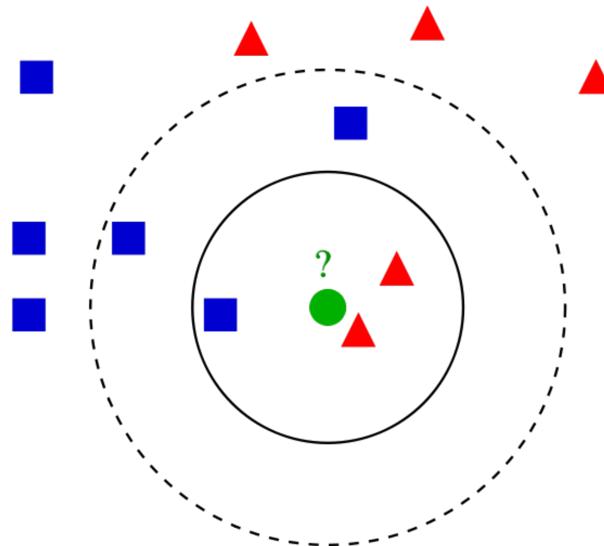

Figure 2.27: k-NN classification, source: `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`

The original proposal of Ojala, Pietikainen, and Harwood, 1994 uses *Kullback-Leibler divergence*[31] to define the distance between the input to predict and the training samples. For simplicity this experiment uses *histogram intersection* proposed by Swain and Dana H. Ballard, 1991 to calculate this distances. The implementation is shown in listing 2.4.

```
1 def histogram_intersection(h1, h2):
2     minima = np.minimum(h1, h2)
3     intersection = np.true_divide(np.sum(minima), np.sum(h2))
4     return intersection
```

Listing 2.4: Histogram Intersection

The chosen number of neighbors to vote for the class is three (**k=3**).

[31] https://en.wikipedia.org/wiki/Kullback-Leibler_divergence

2.3.2 CNN Xception - Light

The second experiment used a CNN to classify the images. In contrast to the *LBP-k-nearest neighbor* experiment, the feature extraction was learned by the convolutional layers of the model, during training, instead of the *hand-crafted* feature extraction by LBP. *Keras* ships with some prominent deep learning models[32] for instance *VGG16[33] and InceptionV3[34]* for image classification. There are also the weights from trainings with different datasets (f.e.: *imagenet*) available, for instant usage, retraining or transfer learning of these models. The downside of these models is that they are very deep which results in an enormous number of trainable parameters (f.e.: Inception ResNet V2: 54,283,877 trainable parameters). However, a tutorial on the *keras* homepage[35] features a simpler model with way less trainable parameters than the built in models (2778013). Due to the limited amount of available training data (about 4000 pictures) this model is better suited and results in reduced time needed for training the model. An initial comparison of the keras tutorial and Inception ResNet V2 on the *ROI without segmentation* dataset (see *ROI without segmentation* in Section 2.1.3) showed nearly the same outcome on unseen data after training (see Table 2.6). Therefore the model proposed in keras image classification tutorial was used for this experiment.

Model	**trainable parameters**	**accuracy on testdataset**
Inception Resnet V2	54,283,877	60.83%
keras tutorial	2,778,013	59.22%

Table 2.6: Inception Resnet V2 vs. keras image classification tutorial

The Xception network

The Keras image classification tutorial uses a stripped down version of the *Xception* network proposed by Chollet, 2017.

We propose a convolutional neural network architecture based entirely

[32]`https://keras.io/api/applications/`
[33]Simonyan and Zisserman, 2015.
[34]Szegedy, Vanhoucke, et al., 2015.
[35]`https://keras.io/examples/vision/image_classification_from_scratch/`

> on depthwise separable convolution layers. In effect, we make the following hypothesis: that the mapping of cross-channels correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled. (Chollet, 2017)

The Xception architecture can be seen as evolution of the inception architecture described by Szegedy, Vanhoucke, et al., 2015. To understand how the Xception architecture works it is usesfull to take a closer look at his predecessor.

First CNN architectures used stacks of convolution layers followed by max-pooling layers (see Krizhevsky, Sutskever, and Hinton, 2012), later approaches consisted of stacks with multiple convolutions followed by a single pooling layer (see Simonyan and Zisserman, 2015). After that Szegedy, Vanhoucke, et al., 2015 proposed the inception architecture which consists of stacks of inception modules (see Figure 2.28). This architecture has several offsprings like Inception V2[36], Inception V3[37] or Inception-ResNet V2[38]. The basic idea of the inception modules is

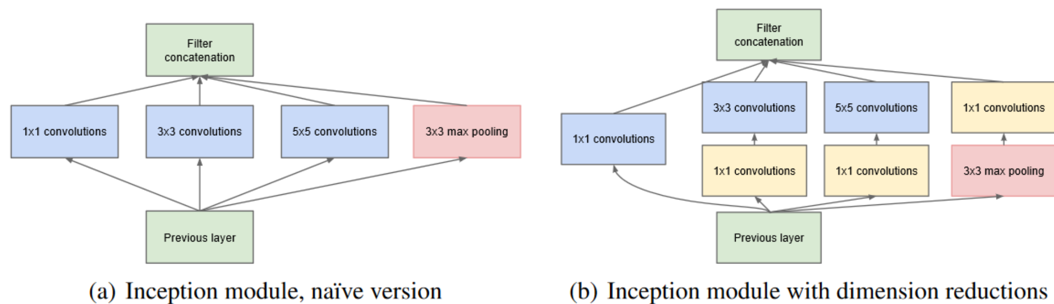

Figure 2.28: Inception module, (source: Simonyan and Zisserman, 2015)

to create deep feature maps, by stacking different filters of different convolutions and spatial reductions with max pooling by filter concatenation, to extract richer features of the input (see naive version in Figure 2.28). This means that the convolutions and pooling are not done sequentially as in previous architectures, they are done simulationsly on the same input. Deep feature maps will result in a huge number of parameters to learn in the next inception module for the convolutions.

[36]Ioffe and Szegedy, 2015.
[37]Szegedy, Vanhoucke, et al., 2015.
[38]Szegedy, Ioffe, et al., 2016.

To counteract that a 1×1 convolution is added before these convolution layers. A 1×1 convolution is also used to increase the feature channels after the pooling operation to match with the depth of the filters learned in the convolution paths (see Inception module with dimension reductions in Figure 2.28).

To get a better understanding of these 1×1 convolutions they are explained in detail in the following. They can be used in three different ways: dimension reduction, dimension increasement and projections without spatial transformation (see below, with w as width, h as height, n is the initial number of channels and k presents the resulting channel depth).

$$\begin{array}{lll} \text{Input} & \text{Convolution} & \text{Result} \\ (w,h,n) & \rightarrow(1\times1\times k) & \rightarrow(w,h,k) \end{array} \tag{2.7}$$

$$\text{reduction:}n>k;\ \text{projection:}n=k;\ \text{increasement:}n<k$$

Inception modules use a 1×1 convolution for dimension reduction with $n > k$ before the convolutions. This is often referred as feature map pooling, pointwise convolution, or in case of the Xception architecture: *a mapping of cross-channels correlations.* For example a 1×1×1 ($k = 1$) convolution can be seen as single neuron learning to produce a single output with the value of every pixel position on all feature maps in depth direction (see Figure 2.29). In other words it summarizes the given n feature maps into one single feature map. To match the depth of the feature map learned in the 3×3 maxpooling a 1x1 convolution is used to increase the dimension. This is done with repeated projections.

The Xception architecture uses stacks of depthwise separable convolution layers instead of inception modules to perform the convolution.

> A depthwise separable convolution, commonly called "separable convolution" in deep learning frameworks such as TensorFlow and Keras, consists in a depthwise convolution, i.e. a spatial convolution performed independently over each channel of an input, followed by a pointwise convolution, i.e. a 1×1 convolution, projecting the channels output by the depthwise convolution onto a new channel space. (Chollet, 2017)

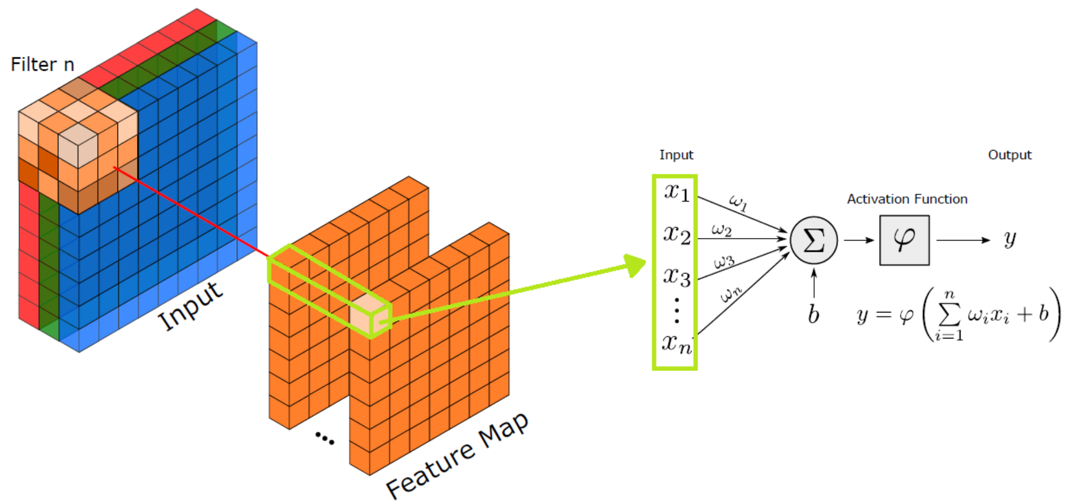

Figure 2.29: Feature pooling described as neuron

Each of the 14 modules has a linear residual connection[39] around them to allow skipping of the surrounded module, except of the first and last modules (see Figure 2.30). This shortcut connection also transforms the input to match with the dimension of the next module. For spatial transformation padding and for a matching depth a 1×1 convolution is used.

Implementation

As stated, the tutorial on keras includes a small version of the Xception network. The differences to proposed architecture (see Figure 2.30) are explained in this section. The entry flow is implemented according to the proposal of the original paper, except the additional module with a filter size of 512. Middle flow and exit flow are replaced by a simpler implementation (see Listing 2.5). The inputsize of 180×180 pixels from the tutorial was changed to 360×360 pixels, because the samples in the created datasets have a higher spatial resolution than the images used in the tutorial. For the unwrapped datasets which had at least a width of 200 pixels (50deg with four samples per degree) and a fixed height of 100 pixels, the inputsize was set to 200×100 pixels.

[39] K. He et al., 2015.

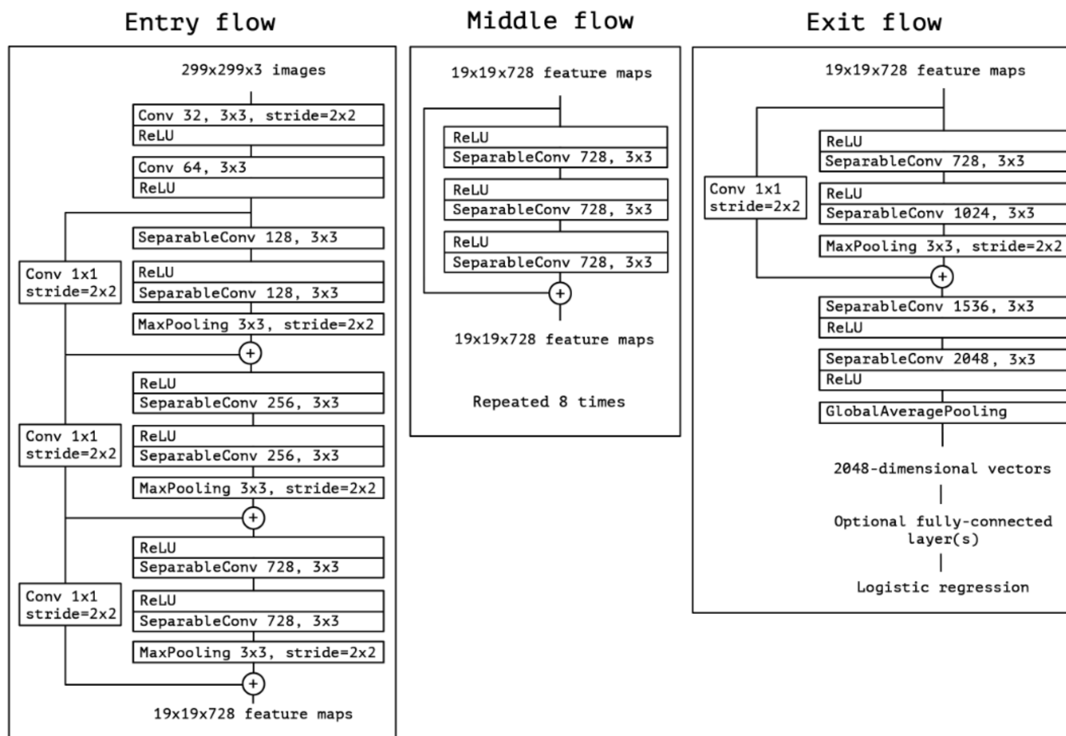

Figure 2.30: Xception architecture, source: Chollet, 2017

```
...
# x is the last layer of the entry flow
x = layers.SeparableConv2D(1024, 3, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)

if classes == 2:
  outputs = layers.Dense(1, activation="sigmoid")(x)
else:
  outputs = layers.Dense(classes, activation="softmax")(x)
```

Listing 2.5: Replacement for middle and exit flow

For the complete model implementation with keras see appendix A.4.

2.3.3 Manual Classification

To get an idea how a human participant performs on doing the classification based on images, a subset of ten images per class was excluded from the two datasets *ROI without segmentation* and *Unwrappings* (described in Subsection 2.1.3). So each participant has 50 images per dataset (100 in total) to classify. The first human classifier is an amateur who had never seen a real pelton wheel, the second person is the employee who coordinates the inspections of pelton wheels at *IllwerkeVKW*. To make things fair the amateur got 20 demo images of each class, before doing the grading.

3 Results

This chapter covers the results of applying the three classification methods: *LBP + k-nearest neighbors classifier*, *CNN Xception light* and *manual classification* on the three datasets *ROI segmented*, *ROI without segmentation* and *Unwrappings* explained in Section 2.1.3. The following sections are ordered by method. Each section contains the results gained by an applicable metric and a short interpretation of the results. To get a fair comparison between the performance of the *LBP + kNearest neighbors classifier* and *Xception light network* approach, different alterations regarding the training and test datasets were made. These alterations are:

- **Binary classification** with a dataset containing only class zero and class four
- **Second binary classification** with class one versus class four
- **Dataset without class imbalance** via downsampling

For binary classification class zero and class four were selected initially because they should have the greatest difference which means the classification should be easier. The good results of *LBP + k-nearest neighbors classifier* and *CNN Xception light* in this binary classification lead to some doubt, especially in the deep learning method. The mistrust in the result arose from the markers on the buckets, where the maintainance personnel has marked each erosion region. Is the trained network just a marker detector similar to the glorified snow detector[1], where some students had tried to built an husky/wolf classifier? This classifier has learned that when there is snow on the ground it must be a wolf. To avoid this a second binary classification was done with class one versus class four, because class one images also contain the cavitation markings where most of the class zero

[1]`https://innovation.uci.edu/2017/08/husky%2Dor%2Dwolf%2Dusing%2Da%2Dblack%2Dbox%2Dlearning%2Dmodel%2Dto%2Davoid%2Dadoption%2Derrors/`

images did not. The results showed nearly the same outcome. For the *unwrapped* dataset the mean accuracy was 2.49% lower and for the *ROI segmented* dataset it was even 1.17% higher (see Table 3.2). This strongend the belief that no marker detector was created. There is also a new research branch dealing with this kind of problems called Explainable AI (XAI).

> However, the use of complex AI algorithms like Deep Learning, Random Forests, etc., could result in a lack of transparency to users which is termed black/opaque box models. Thus, For AI to be confidently rolled out by industries and governments, there is a need for greater transparency in explaining the AI decision making process to users to generate "White /Transparent Box" models which can also be termed Explainable AI (XAI). (Hagras, 2018,)

The last dataset alteration is a downsampling of all classes to the least present class in the dataset to avoid a bias introduced through class imbalance in the datasets. This was done for the full dataset (class zero to four), and the two binary classifications (class zero vs. four, class one vs. four). The samples were picked randomly with `random.sample`, from each class.

3.1 LBP - k-nearest neighbors classifier

For scoring the *LBP - k-nearest neighbors classifier* method, the accuracy on the test dataset was calculated (see Section 2.1.3). Because LBP extracts micro features in a rotation invariant way, no augmentations were applied in the process.

Table 3.1 contains the results of the *LBP + k-nearest neighbors classifier* method. The columns are the dataset used, the alteration on the dataset, the number of training samples and test samples (samples which are not used during training) and the accuracy on the test dataset (see 2.1.3). All experiments with the full class set (one to four) gained a accuracy between 24% and 36%. This is not very high, regarding a baseline of 20% with guessing one out of five classes. The main reason for this could be that the manual classification was subjective. There was no objective criteria like eroded area in mm^2 or length in mm (compare to Table 2.2). The result shows a high accuracy in the classification between class zero and class four. This supports the thesis that there was a fluctuation in the

Dataset	Alteration	train/test	Accuracy
ROI unsegmented	-	3324/434	24.65%
ROI segmented	-	3324/434	32.03%
ROI segmented	0 vs. 4	1645/165	82.42%
ROI segmented	**1 vs. 4**	**3324/162**	**77.78%**
Unwrapped	-	2571/331	29.31%
Unwrapped	0 vs. 4	1206/124	83.87%
Unwrapped	**1 vs. 4**	**1204/122**	**77.05%**
ROI unsegmented	downsampled	2674/434	30.41%
ROI segmented	downsampled	2674/434	36.05%
Unwrapped	downsampled	2571/331	29.31%

Table 3.1: Results LBP + k-nearest neighbors classifier

classification during the inspections over the five years. Another reason, learned during discussing the results with the person who is in charge for the inspections, is that the grading not entirely consists of erosion through cavitation, it also contains other damages on the bucket like rock chips which were not in the ROI. This could also explain why there was no performance gain on the unwrappings, since the extracted region on these images were even smaller. The result also shows that there is a slight performance loss in the binary classification between class one vs. class four in comparison to class zero vs. class one. Remember most samples of class zero did not contain any markings. For this reason class one vs. class four experiments are considered to be the most representative binary classification. Confusion matrices of the experiments with all classes underpin these

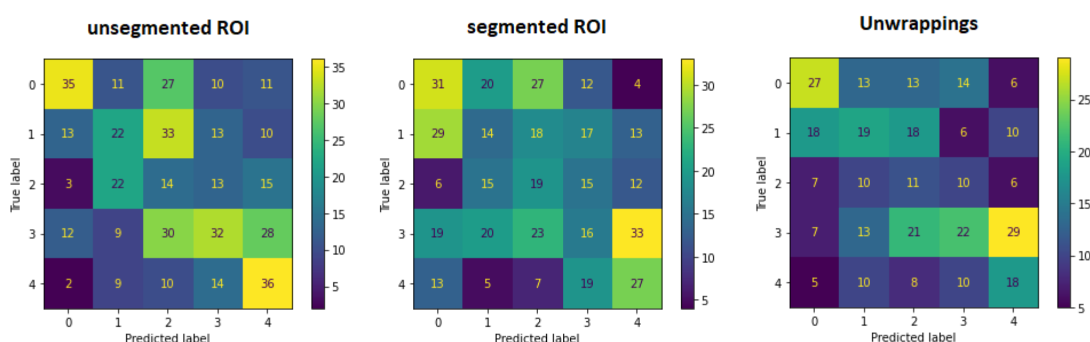

Figure 3.1: Confusion matrices on experiments with all classes (LBP + k-nearest neighbors classifier)

allegations (see Figure 3.1). They also show problems in classifying between class

zero vs. class one, and class three vs. class four. Benefits of the unwrapped data are a smaller memory footprint storing these representations (approximately 300 - 600 kB vs. 5 - 20 kB) and faster training plus classification through the reduced spatial dimensions (approximately 1500×1500 pixel vs. 500×100 pixel).

3.2 CNN Xception - Light

The datasets and the alterations on them are the same ones as in Section 3.1, except the downsampling per dataset. This experiment uses the classification accuracy on the hold back test dataset (see Section 2.1.3) as metric. In addition to compare the results from the different datasets, a global downsampling to the least presented class (1792 images per class) in all three augmented datasets was performed (see Section 3.3). The results outperformed the *LBP + kNearest neighbors classifier* approach on all created datasets. An average accuracy of approximately 60% on the data sets with all classes underpin this, especially when minding that the model used was extracted from a starter tutorial to classify cats and dogs. The assumptions regarding the variance in the manual classification in Section 3.1 are confirmed again by the margin between the multi and binary classification results (see also the confusion matrices in Figure 3.2). Since it is impossible to know what the model has learned during training only assumptions can be made (see also XAI in chapter 3). One assumption for the better results is that the model had learned the variance of the manual classification in the dataset. The model trained on the *unwrappings* falsely classified class one to three as class zero 37 times (see Figure 3.2). This could be explained with the ROI extraction

Dataset	**Alteration**	**train/test**	**Accuracy**
ROI unsegmented	-	24636/434	59.22%
ROI segmented	-	24636/434	60.14%
ROI segmented	0 vs. 4	9954/165	96.97%
ROI segmented	**1 vs. 4**	**9534/162**	**98.14%**
Unwrapped	-	12416/331	54.98%
Unwrapped	0 vs. 4	5180/124	98.39%
Unwrapped	**1 vs. 4**	**4921/122**	**95.90%**

Table 3.2: Results Xception light model

(from the adapted RSM, that also could just be done in a limited θ-range in some cases), which leads to an information loss regarding that stone chips and other things contribute to the classification. If the classification would just consist of erosion damage from cavitation this would not be such a disadvantage.

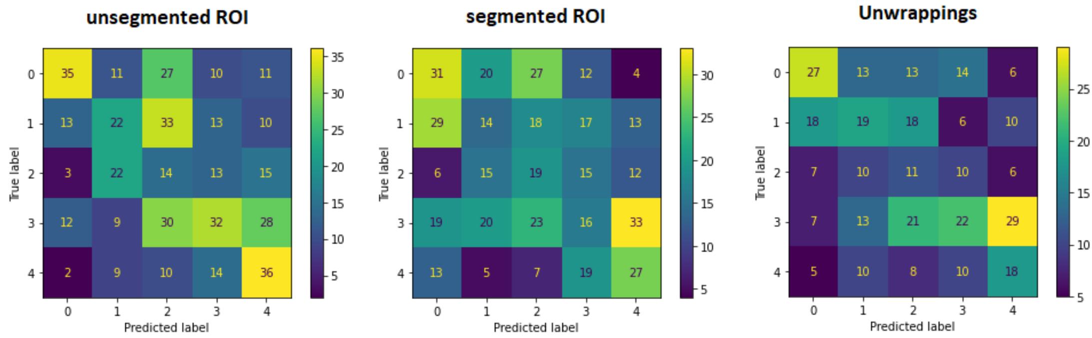

Figure 3.2: Confusion matrices on experiments with all classes (Xception light)

To compare the influence of the different representation of the groundtruth, the experiments were repeated with the reduced datasets (1792 images per class for every dataset). The results in Figure 3.3 show that the unwrappings had nearly the same accuracies in comparison to the ROI's in all experiments. This is remarkable despite the drawbacks: of the information loss mentioned before and being squeezed horizontal during the rescaling from up to 758×100 pixels to 200×100 pixels. The ROI's in comparison were rescaled in a more proportional manner from approximately 1500×1500 pixels to 360×360 pixels. If the images would have been taken with a mounted camera the width of all unwrappings would have been the same. This width could have been used as dimension for the input layer of the model, meaning no rescaling at all.

Dataset	**Alteration**	**train/test**	**Accuracy**
ROI unsegmented	-	8960/434	58.29%
ROI segmented	-	8960/434	55.07%
ROI segmented	0 vs. 4	3584/165	97.58%
ROI segmented	**1 vs. 4**	**3584/434**	**97.53%**
Unwrapped	-	8960/331	53.58%
Unwrapped	0 vs. 4	3584/124	95.97%
Unwrapped	**1 vs. 4**	**3584/122**	**97.54%**

Table 3.3: Results Xception light model, with global downsampling

Due to the benefits of the unwrapped representation (see Section 3.1), the training and classification needed lesser time (see also Table 3.4).

Dataset	train/test	epoch duration	predict duration
ROI segmented (1 vs. 4)	9534/162	57s	21s
Unwrapped (1 vs. 4)	4921/122	9s	551ms

Table 3.4: Comparison of training duration and duration for predicting the whole test data set of *ROI segmented* against *Unwrapped* data

3.3 Manual Classification

This experiment gained some interesting results. The first thing was that the expert achieved a lower result than the amateur on both datasets. The confusion matrix of the expert in Figure 3.3 shows that most of the false classified samples were graded to low, meaning predicting a lesser erosion grade. One reason for this could be that one of the original grading criteras was the roughness on the bucket surface. This was also the impression during labeling of the data. On some images the missing focus of the images make the erosion completley invisible or at least less visible. So the assumption is that with haptic feedback on the real object the grading especially from the expert would have been different. Therefore the training of the amateur which exclusively consisted of seeing 20 images per class could be seen as advantage in comparison to the expert. Both of the contestants did not classify any class zero sample with four and vice versa. This matches with the good result of the binary classification (class zero and class four) of the automated classification with LBP + kNearest neighbor classifier and the Xception light model. This can be seen as confirmation that there are too many erosion grades in between zero to four, in order to perform a accurate classification.

Participant	train/test	Mean-Accuracy
Expert	0/50	36%
Amateur	100/50	48%

Table 3.5: Results of manual classification on *ROI without segmentation* dataset

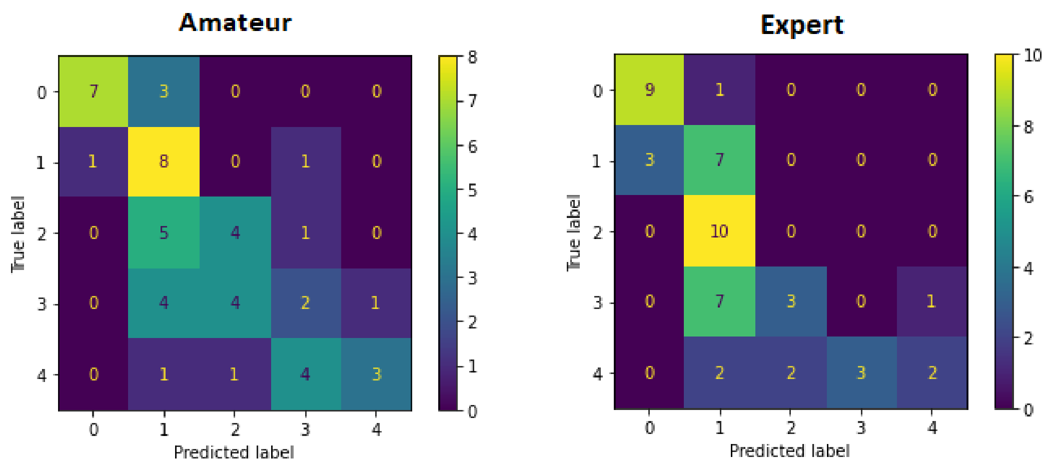

Figure 3.3: Confusion matrices on *ROI without segmentation* dataset for manual classification

The results on the *Unwrappings* dataset (see Unwrappings in Section 2.1.3), were lower for both contestants (see Table 3.6). They both mentioned that the process was more guessing than classifying the samples, which is reflected by the confusion matrices of their classification in Figure 3.4. One reason for this could be that because of the unwrapping the absolute position where most erosion appears is variable in contrast to the uniformity of the ROI's. All samples also differed in width, which was a result of the unwrapping process trying to unwrap the most continuous part of the ellipse inside the image space. These added variance on the unwrapped images make a manual classification harder. The confusion matrices in Figure 3.4 show that similar to the manual classification on the *ROI without segmentation* dataset none of the edge classes (class one and four) were classified opposite to their true label. The automated methods in contrast predicted the opposite class in some cases (see Figures 3.2, 3.1).

Participant	**train/test**	**Mean-Accuracy**
Expert	0/50	28%
Amateur	0/50	30%

Table 3.6: Results of manual classification on *unwrapping* dataset

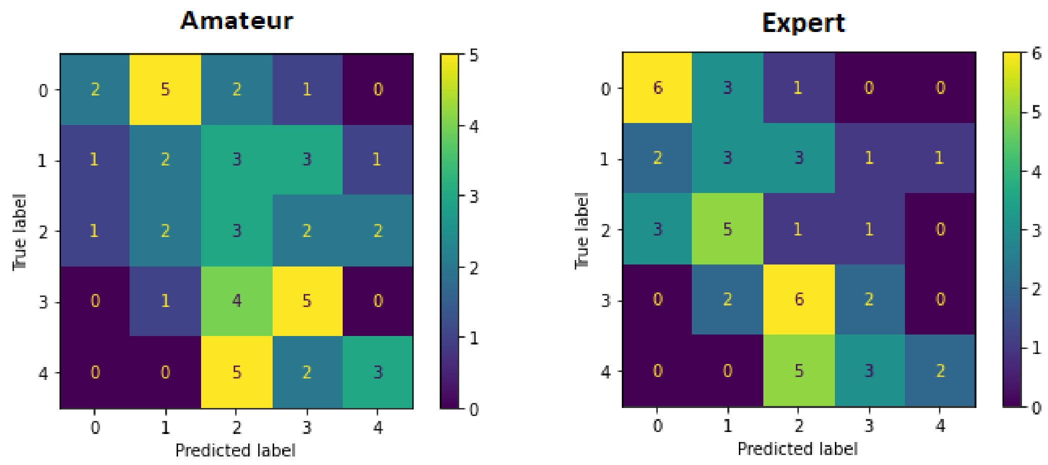

Figure 3.4: Confusion matrices on *unwrapping* dataset for manual classification

4 Summary and Outlook

4.1 Conclusion

The experiments in Chapter 2 had shown that an automated grading of erosion from cavitation with computer vision and machine learning is possible. The needed separation of the right and left bucket, due to the fact that they were graded individually, would have been impossible without the semantic segmentation by a U-Net (see Section 2.2.2). Since all experiments were done on images taken by hand the results can be seen as baseline for the industrial grade implementation with a mounted camera which takes photos at a defined shutter speed, exposure time and a constant lighting. At first glance the best accuracy of approximately 60% in erosion grade classification from zero to four did not seem impressive. But with the limitations in size and quality of the dataset, the results imply that the chosen approach works. The deep learning method (CNN) will benefit from more and better labeled data. The previous mentioned fact and the superior results in the erosion classification show how powerfull these deep learning techniques are (see Section 2.3). Results in the erosion classification of the CNN approach in comparison to the LBP approach show that the CNN learned a better/stronger feature representation as the handcrafted feature extraction by the LBP method. The CNN even outperformed the human cavitation expert by a margin of approximately 22%. Results of all methods on the full class set (zero to four) are shown in Table 4.1. One of the lessons learned is the importance of the quality of the ground truth (inspection images and inspection protocols). This starts with the metrics for the manual scoring. They were to subjective, which made it impossible for the maintainance personnel to do a consistent grading over time. There was also a shift in the grading from overall bucket condition to level of erosion. The format of the ground truth (pdf inspection protocols, inconsistent folder layouts with images) made the preperation of the data a time consuming

Method	train/test	Accuracy
LBP + k-nearest neighbors	3324/434	32.03%
Xception Light (CNN)	8960/434	58.29%
Human Amateur	100/50	48%
Human Expert	0/50	36%

Table 4.1: Erosion classification results of all methods (on unsegmented ROI's, all classes)

process. This leads to the conclusion that in future inspections (not just limited to pelton wheel inspections), the data should be stored in a consistent folder layout, which allows an easier extraction and labeling of the data. For example folders for each class containing the images could be used (see Figure 4.1, following the keras dataset layout[1]). Experiments with reduced class sets (binary

```
training_data/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
etc.
```

Figure 4.1: Folderlayout proposed by keras

classification) and the confusion matrices have shown that five classes are too much, even for the maintainance personnel. This is amplified by the subjective grading due to the missing objective class specification. Table 4.2 shows the results of the automated methods performing a binary classification (class one and four) on the unwrapped data set (see Unwrappings). The unwrapping method

Method	train/test	Accuracy
LBP + k-nearest neighbors	1204/122	77.05%
Xception Light (CNN)	3584/122	97.54%

Table 4.2: Erosion classification results of all methods (on unwrapped images, class one vs. four)

[1] https://keras.io/api/preprocessing/

proposed in this thesis creates a good ROI extraction of the area where the cavitation manifests in erosion (see Section 2.2.4). Other benefits of the application of this method are that less storage is needed to store the data and the training plus classification will be faster (see Table 3.4). With a better ground truth (images from a mounted camera, less classes, objective classification just for cavitation), the results of the unwrappings could lead to better results compared to images without unwrapping. This has to be examined in the later implementation of the industrial grade automatic classification of erosion.

4.2 Outlook

At the time of writing this the new mounted camera system is installed and starts collecting images. The techniques and methods discussed in this thesis can not be used direct on the new data gathered by the new camera system, especially the trained deep learning models. But they are providing a good starting point for implementing a industrial grade classification of the erosion. The next step is to collect the data in a way so that it can be used with Machine learning (ML) methods (class folders). As mentioned previously the proposal is made to use lesser classes with objective specification of the grades, for instance erosion area in mm^2. Since all erosion was polished away when installing the camera system, all samples will show flawless buckets. It will take approximately one to two years until the first erosion due to cavitation will be seen. Therefore this time can be used to create a preprocessing pipeline which stores the collected images appropriately. If just a binary classification for the bucket health is required, using a Generative Adversarial Network (GAN), for anomaly detection should be considered (see Akcay, Atapour-Abarghouei, and Breckon, 2018 and Di Mattia et al., 2021). The approach could also detect stone chips and other anomalies on the bucket.

> We propose anomaly detection based on deep generative adversarial networks. By concurrently training a generative model and a discriminator, we enable the identification of anomalies on unseen data based on unsupervised training of a model on healthy data. (Schlegl et al., 2017)

This means that the training of a GAN-model like *AnoGAN* or *GANomaly* could start after collecting many healthy samples. After collecting an sufficient amount of data (with erosion due to cavitation), the classification of the erosion should then be repeated with a better suited and optimized CNN model as the one used in this thesis, which is a rather simple one based on a keras tutorial.

Bibliography

Akcay, Samet, Amir Atapour-Abarghouei, and Toby P. Breckon (Nov. 2018). *GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training*. en. Number: arXiv:1805.06725 arXiv:1805.06725 [cs].

Aszemi, Nurshazlyn Mohd and P.D.D Dominic (2019). "Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms". en. In: *International Journal of Advanced Computer Science and Applications* 10.6. ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2019.0100638.

Ballard, D. H. (Jan. 1981). "Generalizing the Hough transform to detect arbitrary shapes". en. In: *Pattern Recognition* 13.2, pp. 111–122. ISSN: 0031-3203. DOI: 10.1016/0031-3203(81)90009-1.

Barkan, Oren et al. (Dec. 2013). "Fast High Dimensional Vector Multiplication Face Recognition". In: *Proceedings of the 2013 IEEE International Conference on Computer Vision*. ICCV '13. USA: IEEE Computer Society, pp. 1960–1967. ISBN: 978-1-4799-2840-8. DOI: 10.1109/ICCV.2013.246.

Chollet, Francois (July 2017). "Xception: Deep Learning with Depthwise Separable Convolutions". en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, pp. 1800–1807. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.195.

Daugman, J. (Jan. 2004). "How iris recognition works". In: *IEEE Transactions on Circuits and Systems for Video Technology* 14.1. Conference Name: IEEE Transactions on Circuits and Systems for Video Technology, pp. 21–30. ISSN: 1558-2205. DOI: 10.1109/TCSVT.2003.818350.

Di Mattia, Federico et al. (Sept. 2021). *A Survey on GANs for Anomaly Detection*. en. Number: arXiv:1906.11632 arXiv:1906.11632 [cs, stat].

Duda, R. and P. Hart (1972). "Use of the Hough transformation to detect lines and curves in pictures". In: *CACM*. DOI: 10.1145/361237.361242.

Ghosh, Swarnendu et al. (July 2019). "Understanding Deep Learning Techniques for Image Segmentation". en. In: *arXiv:1907.06119 [cs]*. arXiv: 1907.06119.

Hagras, Hani (Sept. 2018). "Toward Human-Understandable, Explainable AI". In: *Computer* 51.9. Conference Name: Computer, pp. 28–36. ISSN: 1558-0814. DOI: 10.1109/MC.2018.3620965.

He, Kaiming et al. (Dec. 2015). "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]*. arXiv: 1512.03385.

Ioffe, Sergey and Christian Szegedy (Mar. 2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]*. arXiv: 1502.03167.

"Assessment of Remote Cavitation Detection Methods with Flow Visualization in a Full Scale Francis Turbine" (2018a). en. In: *Proceedings of the 10th International Symposium on Cavitation (CAV2018)*. Ed. by Joseph Katz. ASME Press, pp. 119–124. ISBN: 978-0-7918-6185-1. DOI: 10.1115/1.861851_ch24.

"Detection and Level Estimation of Cavitation in Hydraulic Turbines with Convolutional Neural Networks" (2018b). en. In: *Proceedings of the 10th International Symposium on Cavitation (CAV2018)*. Ed. by Joseph Katz. ASME Press, pp. 293–296. ISBN: 978-0-7918-6185-1. DOI: 10.1115/1.861851_ch56.

Kingma, Diederik P. and Jimmy Ba (Jan. 2017). *Adam: A Method for Stochastic Optimization*. Number: arXiv:1412.6980 arXiv:1412.6980 [cs]. DOI: 10.48550/arXiv.1412.6980.

Krizhevsky, A., Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet classification with deep convolutional neural networks". en. In: *undefined*.

Li, Beibei et al. (Feb. 2021). "Analysis method of the cavitation vibration signals in poppet valve based on EEMD". en. In: *Advances in Mechanical Engineering* 13.2. Publisher: SAGE Publications, p. 1687814021998114. ISSN: 1687-8132. DOI: 10.1177/1687814021998114.

Ojala, T., M. Pietikainen, and D. Harwood (Oct. 1994). "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions". In: *Proceedings of 12th International Conference on Pattern Recognition*. Vol. 1, 582–585 vol.1. DOI: 10.1109/ICPR.1994.576366.

Redmon, Joseph et al. (May 2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv:1506.02640 [cs]*. arXiv: 1506.02640.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (May 2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". en. In: *arXiv:1505.04597 [cs]*. arXiv: 1505.04597.

Schlegl, Thomas et al. (Mar. 2017). *Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery*. Tech. rep. arXiv:1703.05921. arXiv:1703.05921 [cs] type: article. arXiv. DOI: 10.48550/arXiv.1703.05921.

Shorten, Connor and Taghi M. Khoshgoftaar (July 2019). "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1, p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0.

Simonyan, Karen and Andrew Zisserman (Apr. 2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556 [cs]*. arXiv: 1409.1556.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. ISSN: 1533-7928.

Swain, Michael J. and Dana H. Ballard (Nov. 1991). "Color indexing". en. In: *International Journal of Computer Vision* 7.1, pp. 11–32. ISSN: 1573-1405. DOI: 10.1007/BF00130487.

Szegedy, Christian, Sergey Ioffe, et al. (Aug. 2016). "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *arXiv:1602.07261 [cs]*. arXiv: 1602.07261 version: 2.

Szegedy, Christian, Vincent Vanhoucke, et al. (Dec. 2015). “Rethinking the Inception Architecture for Computer Vision”. en. In: DOI: 10.48550/arXiv.1512.00567.

Trefný, Jiˇrí and Jiˇrí Matas (n.d.). “Extended Set of Local Binary Patterns for Rapid Object Detection”. en. In: (), p. 7.

Wang, Li and Dong-Chen He (Jan. 1990). “Texture classification using texture spectrum”. en. In: *Pattern Recognition* 23.8, pp. 905–910. ISSN: 0031-3203. DOI: 10.1016/0031-3203(90)90135-8.

Xu, Zhenfa et al. (Nov. 2021). “Research on Visualization of Inducer Cavitation of High-Speed Centrifugal Pump in Low Flow Conditions”. en. In: *Journal of Marine Science and Engineering* 9.11. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, p. 1240. ISSN: 2077-1312. DOI: 10.3390/jmse9111240.

Zhang, Chuanhong, Fang Lu, and Linzhang Lu (Apr. 2019). “High-speed visualization of cavitation evolution around a marine propeller”. en. In: *Journal of Visualization* 22.2, pp. 273–281. ISSN: 1875-8975. DOI: 10.1007/s12650-018-00540-7.

List of Acronyms

AE Acoustic Emission

CNN Convolutional Neural Network

GAN Generative Adversarial Network

GHT Generalized Hough Transformation

IoU Intersection over Union

LBP Local Binary Patterns

ML Machine learning

ReLU Rectified linear unit

rgb RGB color model

ROI Region of Interest

RSM Rubber sheet model

XAI Explainable AI

List of Figures

1.1 Open pelton wheel on inspection day 2
1.2 Erosion caused by cavitation on the backside of a bucket of a pelton wheel 2
1.3 Pelton wheel component, source: `https://www.airandhydraulic.com/2020/09/components-of-pelton-turbine.html` 4

2.1 Segmented images where the background pixels are set to black, see Section 2.2.2 7
2.2 Samples ROI extraction on segmented image, with classes 9
2.3 Numbers of samples per class in dataset 10
2.4 Numbers of samples per class in augmented dataset 10
2.5 Samples ROI extraction on real image, with classes 11
2.6 Examples of source (left) and result of unwrapping (right), for class zero and class four 12
2.7 Numbers of samples per class in unwrappings dataset 13
2.8 Numbers of samples per class in augmented unwrappings dataset . 13
2.9 Area where erosion through cavitation manifests 14
2.10 Initial tries with *opencv* 15
2.11 GHT on sample image with nearly same tilt 17
2.12 U-net architecture from original paper (Ronneberger, Fischer, and Brox, 2015) 18
2.13 Left labeled shovel in *VoTT*, right created binary mask 20
2.14 U-Net evaluation, 200 images 20
2.15 Demo of the augmentations 23
2.16 U-Net training, 960 images 25
2.17 U-Net final training, 1920 images 26
2.18 Small inaccuracy on background segmentation 27
2.19 Complete segmentation fail when multiple shovels are on the image 27

2.20 Process of bucket separation 28
2.21 Iris Image Processing Process, from https://github.com/YifengChen94/IrisReco 30
2.22 Ramkumar, R.P. and Arumugam, S. (2013). Improved Iris Segmentation Algorithm without normalization Phase. International Journal of Engineering and Technology. 5. 5107-5113. 31
2.23 Ellipse fitting on ROI 31
2.24 LBP proccess[1] 35
2.25 Possible LBP patterns, source: https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html 35
2.26 Histogram created with LBP on a bucket image 36
2.27 k-NN classification, source: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm 37
2.28 Inception module, (source: Simonyan and Zisserman, 2015) 40
2.29 Feature pooling described as neuron 42
2.30 Xception architecture, source: Chollet, 2017 43

3.1 Confusion matrices on experiments with all classes (LBP + k-nearest neighbors classifier) 47
3.2 Confusion matrices on experiments with all classes (Xception light) 49
3.3 Confusion matrices on *ROI without segmentation* dataset for manual classification 51
3.4 Confusion matrices on *unwrapping* dataset for manual classification 52

4.1 Folderlayout proposed by keras 54

A.1 GPU Server - CPU 70
A.2 GPU Server - GPU 70
A.3 GPU Server - RAM in GB 70

[1] Ojala, Pietikainen, and Harwood, 1994.

List of Tables

2.1 Specification of the *iPhone8* images 6
2.2 Defined grading of cavitation erosion 6
2.3 Mapping of inspection protocol grades and used grades 6
2.4 Numbers on initial dataset 7
2.5 IoU result of U-Net on 32 images 26
2.6 Inception Resnet V2 vs. keras image classification tutorial 39

3.1 Results LBP + k-nearest neighbors classifier 47
3.2 Results Xception light model 48
3.3 Results Xception light model, with global downsampling 49
3.4 Comparison of training duration and duration for predicting the whole test data set of *ROI segmented* against *Unwrapped* data . . 50
3.5 Results of manual classification on *ROI without segmentation* dataset 50
3.6 Results of manual classification on *unwrapping* dataset 51

4.1 Erosion classification results of all methods (on unsegmented ROI's, all classes) 54
4.2 Erosion classification results of all methods (on unwrapped images, class one vs. four) 54

Listings

2.1 Training configuration for U-Net in keras 24
2.2 Calculation of IoU 25
2.3 Call of skimage.feature.local_binary_pattern 37
2.4 Histogram Intersection 38
2.5 Replacement for middle and exit flow 43

A Appendix

A.1 Anaconda Environment

Below is a listening of all relevant packages installed in *conda 4.10.3*.

```
python                     3.7.11
tensorboard                2.4.0
tensorboard-plugin-wit     1.6.0
tensorflow                 2.4.1
tensorflow-base            2.4.1
tensorflow-estimator       2.6.0
tensorflow-gpu             2.4.1
....
```

A.2 GPU-Server Hardware Spec

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz
Stepping:              7
CPU MHz:               2288.478
BogoMIPS:              5600.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              22528K
NUMA node0 CPU(s):     0-15,32-47
NUMA node1 CPU(s):     16-31,48-63
```

Figure A.1: GPU Server - CPU

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
+-------------------------------+----------------------+----------------------+
|   6  Quadro RTX 6000     Off  | 00000000:B1:00.0 Off |                  Off |
| 33%   26C    P8    14W / 260W |  23186MiB / 24220MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   7  Quadro RTX 6000     Off  | 00000000:B2:00.0 Off |                  Off |
| 33%   26C    P8    12W / 260W |  24061MiB / 24220MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figure A.2: GPU Server - GPU

```
              total        used        free      shared  buff/cache   available
Mem:            377          30          76           0         270         344
```

Figure A.3: GPU Server - RAM in GB

A.3 Adapted U-net model (keras)

The final *tensorflow keras* model used for bucket segmentation.

```
def build_model(input_layer):
    filters = 16
    # 512 -> 256
    conv0 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(input_layer)
    conv0 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(conv0)
    pool0 = MaxPooling2D((2, 2))(conv0)
    pool0 = Dropout(0.25)(pool0)
    # 256 -> 128
    conv1 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(pool0)
    conv1 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(conv1)
    pool1 = MaxPooling2D((2, 2))(conv1)
    pool1 = Dropout(0.5)(pool1)
    # 128 -> 64
    conv2 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(pool1)
    conv2 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(conv2)
    pool2 = MaxPooling2D((2, 2))(conv2)
    pool2 = Dropout(0.5)(pool2)
    # 64 -> 32
    conv3 = Conv2D(filters * 2, (3, 3), activation="relu", padding="same")(pool2)
    conv3 = Conv2D(filters * 2, (3, 3), activation="relu", padding="same")(conv3)
    pool3 = MaxPooling2D((2, 2))(conv3)
    pool3 = Dropout(0.5)(pool3)
    # 32 -> 16
    conv4 = Conv2D(filters * 4, (3, 3), activation="relu", padding="same")(pool3)
    conv4 = Conv2D(filters * 4, (3, 3), activation="relu", padding="same")(conv4)
    pool4 = MaxPooling2D((2, 2))(conv4)
    pool4 = Dropout(0.5)(pool4)
    # 16 -> 8
    conv5 = Conv2D(filters * 8, (3, 3), activation="relu", padding="same")(pool4)
    conv5 = Conv2D(filters * 8, (3, 3), activation="relu", padding="same")(conv5)
    pool5 = MaxPooling2D((2, 2))(conv5)
    pool5 = Dropout(0.5)(pool5)
    # Middle
    convm = Conv2D(filters * 16, (3, 3), activation="relu", padding="same")(pool5)
    convm = Conv2D(filters * 16, (3, 3), activation="relu", padding="same")(convm)
    # 8 -> 16
    deconv5 = Conv2DTranspose(filters * 8, (3, 3), strides=(2, 2), padding="same")(convm)
    uconv5 = concatenate([deconv5, conv5])
    uconv5 = Dropout(0.5)(uconv5)
    uconv5 = Conv2D(filters * 8, (3, 3), activation="relu", padding="same")(uconv5)
    uconv5 = Conv2D(filters * 8, (3, 3), activation="relu", padding="same")(uconv5)
    # 16 -> 32
    deconv4 = Conv2DTranspose(filters * 4, (3, 3), strides=(2, 2), padding="same")(uconv5)
    uconv4 = concatenate([deconv4, conv4])
    uconv4 = Dropout(0.5)(uconv4)
    uconv4 = Conv2D(filters * 4, (3, 3), activation="relu", padding="same")(uconv4)
    uconv4 = Conv2D(filters * 4, (3, 3), activation="relu", padding="same")(uconv4)
    # 32 -> 64
    deconv3 = Conv2DTranspose(filters * 2, (3, 3), strides=(2, 2), padding="same")(uconv4)
    uconv3 = concatenate([deconv3, conv3])
    uconv3 = Dropout(0.5)(uconv3)
    uconv3 = Conv2D(filters * 2, (3, 3), activation="relu", padding="same")(uconv3)
    uconv3 = Conv2D(filters * 2, (3, 3), activation="relu", padding="same")(uconv3)
    # 64 -> 128
    deconv2 = Conv2DTranspose(filters * 1, (3, 3), strides=(2, 2), padding="same")(uconv3)
    uconv2 = concatenate([deconv2, conv2])
    uconv2 = Dropout(0.5)(uconv2)
    uconv2 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(uconv2)
    uconv2 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(uconv2)
    # 128 -> 256
    deconv1 = Conv2DTranspose(filters * 1, (3, 3), strides=(2, 2), padding="same")(uconv2)
    uconv1 = concatenate([deconv1, conv1])
    uconv1 = Dropout(0.5)(uconv1)
    uconv1 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(uconv1)
```

```
uconv1 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(uconv1)
# 256 -> 512
deconv0 = Conv2DTranspose(filters * 1, (3, 3), strides=(2, 2), padding="same")(uconv1)
uconv0 = concatenate([deconv0, conv0])
uconv0 = Dropout(0.5)(uconv0)
uconv0 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(uconv0)
uconv0 = Conv2D(filters * 1, (3, 3), activation="relu", padding="same")(uconv0)
output_layer = Conv2D(1, (1,1), padding="same", activation="sigmoid")(uconv0)
return output_layer
```

A.4 Xception - light model (keras)

This is the implementation to create the model.

- input shape for ROIs: 360x360x3
- input shape for unwrappings: 200x100x3

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)

    # Entry block
    x = inputs
    x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)
```