Chair of Cyber Physical Systems

Master's Thesis

Development of a Graphical User Interface and Deep Learning Methods for Automated Inspection in a Continuous Casting Steel Plant

Melanie Elena Neubauer, BSc

February 2023

Development of a Graphical User Interface and Deep Learning Methods for Automated Inspection in a Continuous Casting Steel Plant

Master Thesis by Melanie Elena Neubauer[1]

[1]*melanie-elena.neubauer@stud.unileoben.ac.at*, *m11770415* *Montanuniversität Leoben, Austria*

Leoben, Austria, February 09, 2023

Supervisor: Univ.-Prof. Dipl.-Ing. Dr. techn. Elmar Rueckert
Co-Supervisor: M.Eng. Fotios Lygerakis

Chair of Cyber-Physical-Systems
Montanuniversität Leoben, Austria

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 11.02.2023

Unterschrift Verfasser/in
Melanie Elena Neubauer

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my heartfelt gratitude to my family for their unwavering support and encouragement throughout my entire studies. Likewise, I am extremely grateful to Univ.-Prof. Dipl.-Ing. Dr. techn. Elmar Rueckert and M.Eng. Fotios Lygerakis from the Chair of Cyber-Physical-Systems for their invaluable contributions and support in the development of this thesis. I deeply appreciate their significant role in the success of this thesis. Their guidance, knowledge, and expertise were invaluable asset during this journey. My sincere gratitude goes to Assoz. Prof. Dipl.-Ing. Dr.mont. Susanne Michelic and Ao.Univ.-Prof. Dipl.-Ing. Dr.mont. Christian Bernhard from the Chair of Iron and Steel Metallurgy for their support. I would also like to extend my heartfelt thanks to the voestalpine Stahl GmbH Linz for their contribution.

ABSTRACT

The analysis of various processes in a continuous casting plant can aid in reducing costs and defects in the production of steel slabs. As the quality of the slabs can only be determined at the end of the solidification process, this thesis focuses on analyzing the surface movements on the mold using a variety of methods.

The primary objective of this study is to develop a graphical user interface and implement deep learning methods for automated inspection in a continuous casting steel plant. The developed user interface is designed to visualize recorded image data of the mold and perform statistical analysis using techniques such as histogram and optical flow. The results of the analysis are displayed directly in the software, and tests have demonstrated its effectiveness in identifying asynchronous movements between the right and left sides of the mold.

Moreover, the study utilizes a deep neural network method on a publicly available labeled steel dataset with defects. The applied model, Mask R-CNN, can analyze steel defects and provide insight into the quality of the steel end-products. This research demonstrates the potential for combining graphical user interface and deep learning techniques to enhance the inspection process in continuous casting steel plants.

KURZFASSUNG

Die Analyse verschiedener Prozesse in einer Stranggussanlage kann dazu beitragen, Kosten und Fehler bei der Produktion von Stahlbrammen zu reduzieren. Da die Qualität der Brammen erst am Ende des Erstarrungsprozesses bestimmt werden kann, konzentriert sich diese Arbeit auf die Analyse der Oberflächenbewegungen auf der Kokille mit verschiedenen Methoden.

Das primäre Ziel dieser Studie ist die Entwicklung einer grafischen Benutzeroberfläche und die Implementierung von Deep-Learning-Methoden für die automatisierte Inspektion in einer Stahlstranggussanlage. Die entwickelte Benutzeroberfläche dient der Visualisierung der aufgezeichneten Bilddaten der Kokille und der Durchführung statistischer Analysen mit Techniken wie Histogramm und optischem Fluss. Die Ergebnisse der Analyse werden direkt in der Software angezeigt, und Tests haben ihre Wirksamkeit bei der Erkennung asynchroner Bewegungen zwischen der rechten und der linken Seite der Kokille bewiesen.

Darüber hinaus wird in der Studie ein tiefes neuronales Netzwerk auf einen öffentlich zugänglichen markierten Stahldatensatz mit Defekten angewendet. Das angewandte Modell, Mask R-CNN, kann Stahldefekte analysieren und einen Einblick in die Qualität der Stahlendprodukte geben. Diese Forschungsarbeit zeigt das Potenzial der Kombination von grafischer Benutzeroberfläche und Deep-Learning-Techniken zur Verbesserung des Inspektionsprozesses in Stahlstranggussanlagen.

Contents

List of Figures

List of Tables

1 Introduction

Contents

In the field of continuous casting, the monitoring of the mold is a critical aspect in ensuring the quality of the end-product. The evaluation of end-product quality acts as a final step in the process to verify that no defective products are sent to clients. The ability to predict defects early in the casting process can have a significant impact by avoiding the production of subpar steel, thus reducing production costs. This is a crucial area of investigation in the field and will be explored in detail in this thesis.

1.1 Continuous Casting Steel Plant

In a continuous casting plant, the mold plays a pivotal role in the production of steel slabs. The process involves pouring liquid steel into the mold to cast an 'endless' strand, which is solidified using cooling elements such as steam. The quality of the slab and the efficiency of the plant are significantly influenced by the mold. The current method of evaluating slab quality requires cutting off a piece of the slab and analyzing it at the end of the cooling process. This can be both time-consuming and wasteful. If the quality of the slab could be assessed through the structure and movements on the top level of the mold, it would lead to significant advancements in the industry. Temperature data can be used to understand the casting process and optimize further steps.

Slabs can typically be used to create other products, like flat sheet steel. Any defects of the initial steel bars can propagate defects on later products. It is crucial then to efficiently, yet accurately, detect any imperfections or anomalies in the end steel product to ensure its strength, durability, and quality.

1.2 Thesis Contribution

In this thesis, we aim to address the challenge of detecting factors that will lead steel defects early in the process of continuous casting, as well as detecting them on the end steel products. Therefore, we developed a graphical user interface (GUI) to monitor the thermal images of the liquid steel movements in the mold. This interface will provide additional information, based on statistical analysis of the thermal images. In addition, we trained a deep neural network to automatically detect flaws on the end products of a steel factory, using a publicly available dataset.

This thesis aims to address the following research questions:

- Can processes in the mold be detected by inspecting the thermal imaging data through the proposed GUI?
- Can existing asymmetries of disturbances be depicted on the statistical analysis offered by the GUI?
- Are the publicly available dataset or methods sufficient for efficiently and accurately detect steel defects on end products?

We divided the work into two parts. The first one describes the GUI for visualization of the thermal imaging data of the bath level of the mold and its analysis using statistical methods. The second chapter deals with the detection of steel defects using

a publicly available steel defect dataset with a deep neural network approach. In the second part, we describe the deep neural network architecture we used and the method we followed to train it to detect defects on images of flat steel sheets.

2 Background

Contents

In this chapter we give an overview of the background of the thesis. First, we describe the continuous casting method. Then, we present the factors that influence the quality of the steel end-products of the continuous casting steel plants. Finally, we provide ways to avoid defective products dispatching the steel plant.

2.1 Continuous Casting

The continuous casting process was developed, because steel mills produce very large quantities of steel at short intervals, which have to be cast quickly. The liquid steel coming from secondary metallurgy must be given specific shapes, dimensions and weights by casting in the steel mill. In this field, large quantities of liquid steel can be handled sooner than in ingot casting in conventional casting operations. (Degner and VDEh, 2007, pp. 78–79)

Continuous casting is a process in which molten metal solidifies into an endless strand, which is the most used method to process liquid steel. The entire continuous casting system is shown in *Figure 1*.

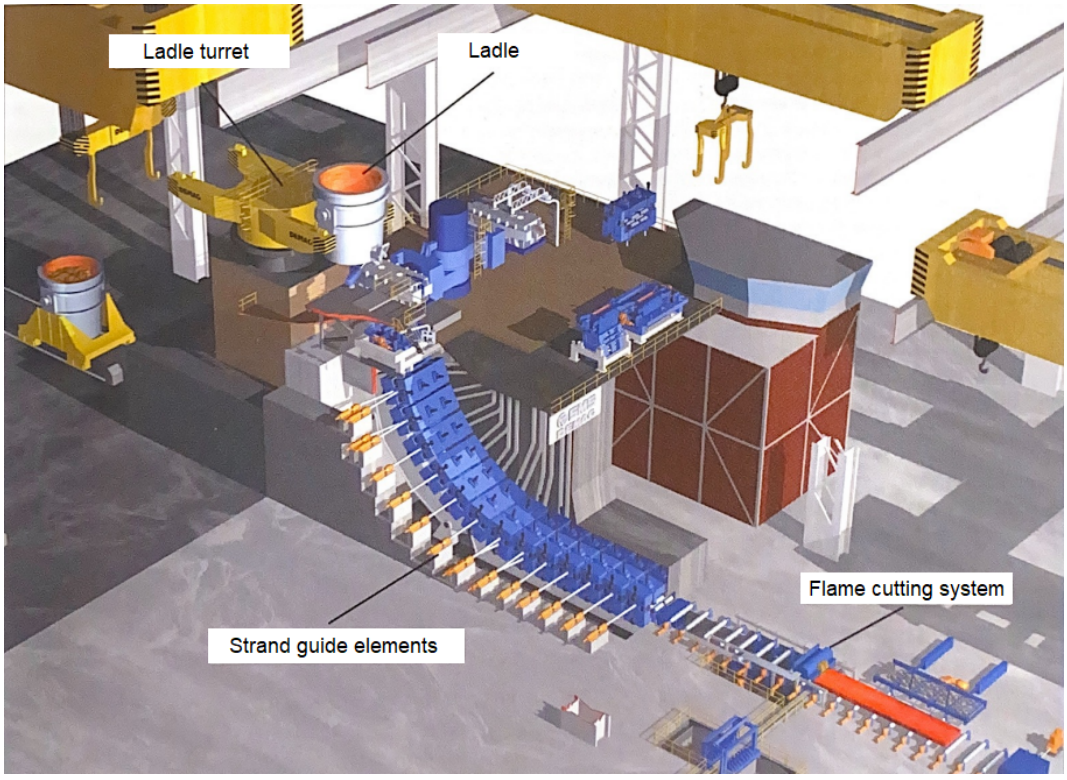


Figure 1: *Structure of the Continous Casting Plant (Degner and VDEh, 2007, p. 79)*

In *Figure 2*, a closer look at the detailed structure can be taken. During continuous casting, the molten steel passes from the ladle through the so-called shadow tube

into the tundish under exclusion of air. This is an intermediate tank which serves to distribute the liquid steel. From the tundish, the liquid steel flows into the water-cooled copper molds. (Degner and VDEh, 2007, p. 80)

The mold is the most important part of the continuous caster, for shaping and forming the strand surface. There are different types of molds, the tubular, and the slab molds. This paper focuses on the slab molds, which are used to produce slabs. These are assembled from individual cooling plates. To ensure continuous casting, the size of the mold can be changed during continuous casting. The width of the mold can be adjusted by moving the side parts.(Schwerdtfeger, 1992, pp. 12–17)

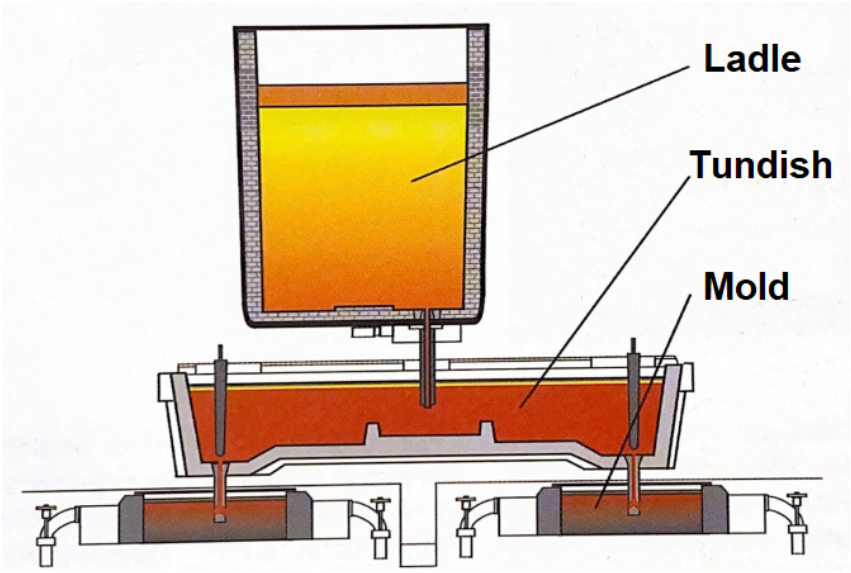


Figure 2: *Structure of the Continuous-Casting-Mold (Degner and VDEh, 2007, p. 80)*

When casting the steel, the mold is set into vertical vibration to prevent the strand from sticking to the mold wall. The strand solidified in the edge zone is then pulled out of the mold. Since this strand still has a liquid core at this point, it must be cooled with water or air. It is supported by rollers to prevent breakage in the thin edge zones. This type of cooling produces the desired uniform solidification structure. After the complete cooling process, the slab is solidified and can be cut at the desired point on the outfeed roller table with a flame cutting system that moves with it. (Degner and VDEh, 2007, p. 80)

Casting powder plays a crucial role in this process as it prevents oxidation, regulates heat dissipation between the steel strand and mold, and provides lubrication for both the strand and mold, ensuring optimal protection and control (Schwerdtfeger, 1992). The measured temperature distributions are primarily used to control the addition of casting powder. Looking at the time sequence of the recorded individual images, atypical movement patterns can be seen when playing back these image sequences, which indicate a strong underlying interfacial interaction between casting powder slag and liquid steel. Slag droplets can be drawn in by the liquid steel, which are carried by the liquid steel flow to the solidification front of the strand shell in the mold and can settle there. During hot rolling of the slabs, these slag defects become visible as line-shaped strip surface defects, so-called slivers.

2.2 Causes of Defects in Continuous Casting Products

Inserting casting powder to the liquid steel in the mold can cause different fluctuations on the movement of the top level of the mold. Another factor that contributes to the change of the mold's movement is the insertion of a mix of gasses that control quality of the produced steel. This movement of the liquid steel, however, can potentially lead to the production of defective steel. (Rui Liu et al., 2014)

2.3 Defect Avoidance

One way to avoid defective products leaving the plant is to monitor the movement dynamics of the liquid steel and predict whether it will lead to defects in the slabs at the end of the continuous casting procedure. Nevertheless, monitoring the movements in the mold is an extremely difficult task. This difficulty arises from the fact that is hard to put sensor in this part of the continuous casting process. In addition, the temperature fluctuations that lead to these movements usually take place in parallel with the presence of external devices, which intervene between the sensors and the mold.

Another solution to avoid defective products, is to automate end-product defect detection. This can be done by using machine vision systems. These systems use cameras to capture images of the end steel products and algorithms to analyze the images to detect any defects. This method is efficient and cost-effective because it does not require any modifications to the casting process, and the detection can be done in real-time, allowing for immediate corrective actions to be taken. Additionally, machine vision systems can detect a wider range of defects than manual inspection, and they are also less prone to human error.

3 Method

Contents

In this Chapter we provide methods to monitor different processes throughout the production line of a steel plant. First, we present a Graphical User Interface for visually inspecting the mold of a continuous caster. Then, train a deep neural network to detect defects on end-products.

3.1 Graphical User Interface

In this chapter we describe the functionality of developed Graphical User Interface (GUI). We provide a step-by-step guide for using the GUI, including how to navigate the interface, how to insert data, and how to access the results. Finally, we discuss the benefits of using the GUI to analyze the movements on the surface of the mold. The development of the GUI was done using the Tkinter python interface [1].

3.1.1 Dataset

In order to gather information from the mold, we utilized thermal imaging cameras to monitor the temperature of the top level of the mold. This was achieved using the KMON measuring system. These thermal imaging cameras allow to record the movements in the mold.

The image data was recorded with the help of two such thermal imaging cameras, which are positioned on the right and left side of the mold respectively. Every camera records one image per second. These two separately taken images are then merged and saved. *Figure 3* serves as an example of how the input data is transferred.

Figure 3: *Process Visualizer - Example Data*

The file name is used to transfer the required data, such as the time of recording or the width of the mold at the time of recording. To visualize the images correctly, they were first divided in the middle. The left part was then rotated by 180 degrees. Both parts were cropped to remove as much of the mold's surroundings as possible. Finally, the two halves were rejoined to form a whole image.

[1] https://python.readthedocs.io/en/stable/library/tkinter.html

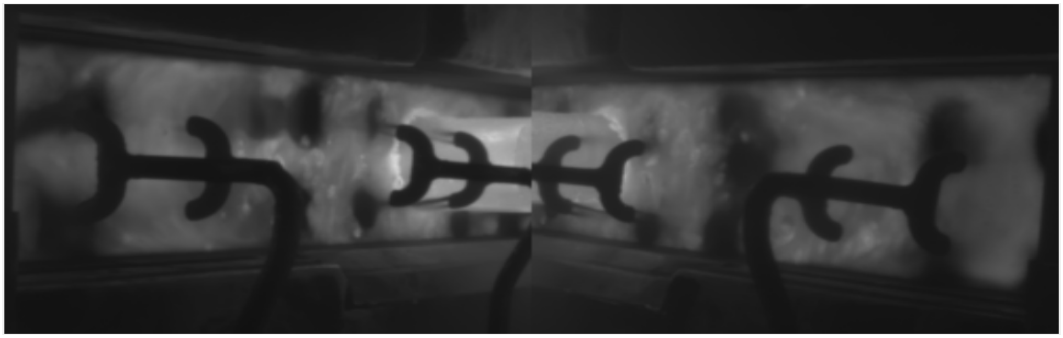


Figure 4: *Process Visualizer - Edited Data*

The resulting images have a size of 500x190 pixels with an intensity that varies between 0 and 255. The intensity measures the temperature in the mold, with 0 being the smallest temperature and 255 the highest. During data recording, all events in the mold are captured, including irrelevant information such as images taken while mold powder is being added. These images provide no insight into the movements on the surface of the mold as the temperature is much lower, resulting in dark images captured by thermal imaging cameras. This information is present but not considered in the subsequent calculations and visualizations.

3.1.2 Structure of the Process Visualizer

The structure of the GUI is designed to ensure ease of use when analysing the movements on the surface of the mold. The GUI aims to simplify mold analysis for users by clearly displaying surface movements through techniques like histograms and optical flow. It visually presents the data for easy viewing and makes it possible to point out anomalies in the data. Asynchronous movements on the bath surface of the mold can be indicative of the quality of the steel. By observing the surface movements, conclusions can be drawn about the quality. *Figure 5* shows the main window of the Process Visualizer.

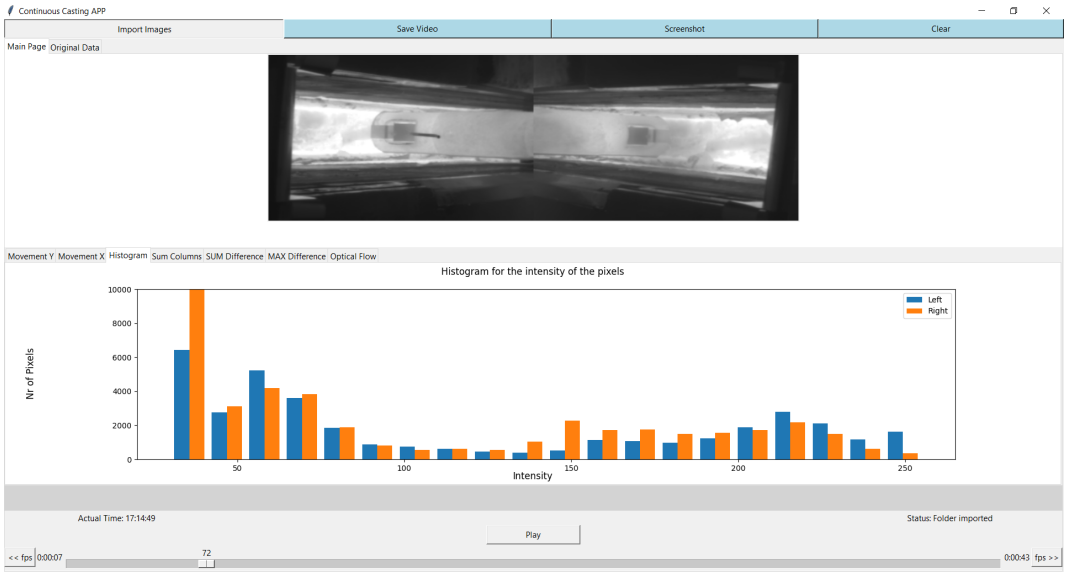


Figure 5: *Example Layout of the Process Visualizer*

Buttons for the Functionality

There are four tabs that offer a different functionality on the top of the main window of the Process Visualizer. The Process Visualizer is divided into two parts. At the top there are the buttons for the functionality and below them, there are the tabs that take up the main part of the window. The *Table 1* briefly describes the functionality buttons.

Table 1: *Description of the Functionality Buttons*

Buttons	Description
Import Images	The 'Import Images'-button is used to import the images from the needed folders, or to choose a specific time range for importing the images.
Save Video	It generates a video and saves it with the actual date and time in the folder 'Data/Videos'. The first frame of the Video is the current frame which is displayed on the image viewer. The selected fps which are used to display the sequence of images is used to generate the video.
Screenshot	The 'Screenshot'-button makes a screenshot of the whole screen and saves it in the folder 'Data/Screenshot' with the actual date-time as name.
Clear	The 'Clear'-button closes the actual window and opens a new window to make new calculations.

Import Images

Upon launch, the main window of the GUI is shown to be empty as depicted in *Figure 6*.

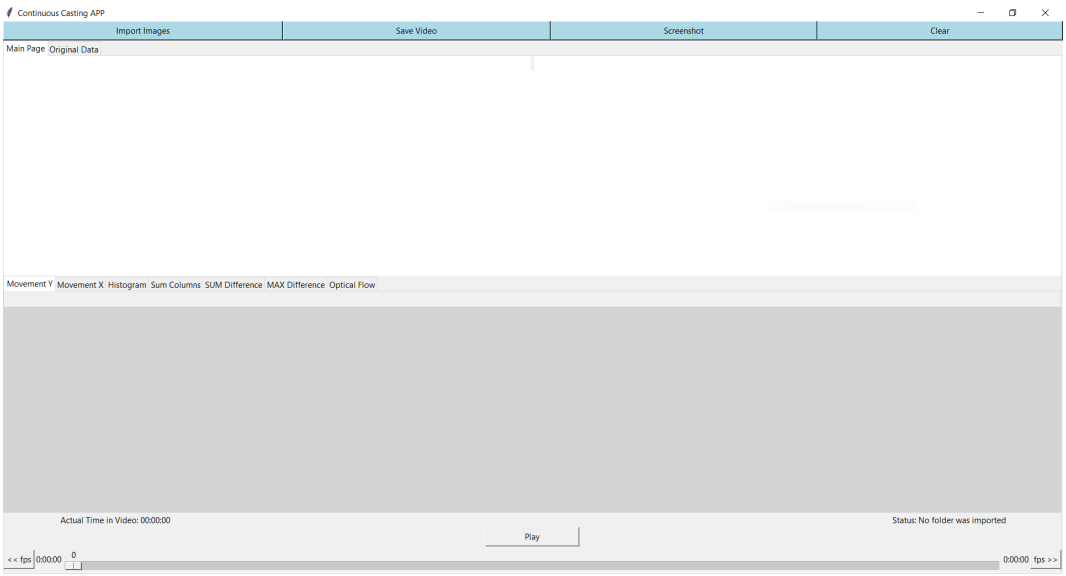


Figure 6: *Process Visualizer - Start Window*

To import the visual data, the **Import Images** button needs to be clicked, which opens the import dialog. *Figure 7* shows the import dialog with its two tabs.

As shown in *Figure 7*, the import dialog offers the possibility to import the images in two different ways depending on the selected tab. Either to import a folder of images of a whole hour or to import only images of a certain time window. These settings can be selected individually. However, it should be noted that when selecting the images of a certain time, they must also exist.

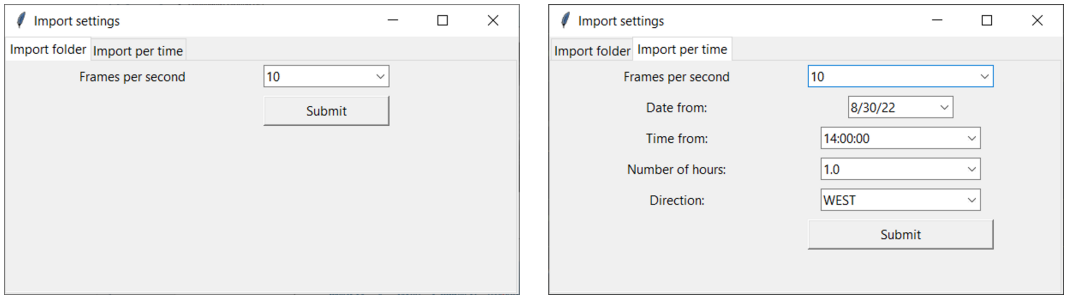


Figure 7: *Process Visualizer - Import Dialog*

After submitting the loading request, the process is displayed in a command window, which opens automatically (*Figure 8*). It should be noted that, the needed loading time depends on the amount of imported images and the performance of the computer.

Figure 8: *Process Visualizer - Loading Procedure*

After the import of the image paths, the software calculates the statistical analyses in the background. Once this is completed, the first frame is created and displayed for visualization. Afterwards the „Play“-Button is activated and can be pressed. The detailed descriptions of the individual diagrams are explained in *Chapter 3.1.3*.

To save memory space the program generates paths and saves these in a list with strings. Every string is a merge of a certain number of paths which are split by a comma.
This results in a list as follows:

[„path1,path2,path3“, „path4,path5,path6“, „path7,path8“]

At last, every single string in this list is compressed to reduce the memory. With this approach, more than 10 times less storage capacity is required when importing the paths than for non-compressed paths.

Save Video

The GUI also offers a function to create a video of consecutive images from the dataset. By pressing the **Save Video** button, the dialog shown in *Figure 9* opens. In this dialog the length of the video must be selected. It should be noted that the size of the time window of the imported data is selected and not the length of the actual video. The final length of the video depends on the actual selected number of frames per second. For instance, if the current number of frames per seconds is two, the video will be half as long as the time selected in the **Save Video** dialog window.

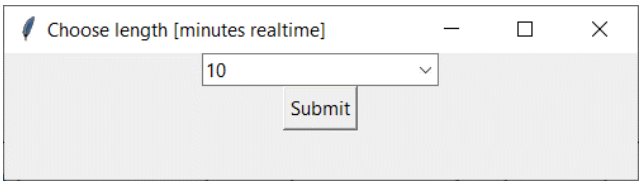


Figure 9: *Process Visualizer - Save Video Dialog*

When submitting the dialog, the program generates a video and saves it with the actual time and the actual date in the folder „Data/Videos“. The currently displayed frame in the video player is also the start frame to be seen in the video.

Main Tabs

The GUI offers two different tabs, one for the main window and one for the imported data. Both tabs are highlighted in *Figure 10*.

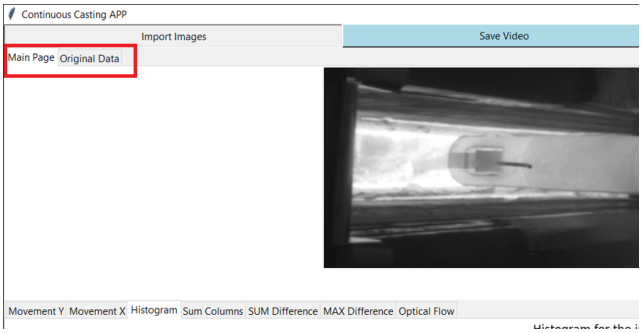


Figure 10: *Process Visualizer - Main Tabs*

Most of the time only the first tab is used. This tab contains all the necessary views of the calculations in various sub-tabs and the image viewer. Additionally, the GUI offers settings in this tab regarding the video display. The second tab is only used to view the imported original data. In *Figure 11* it can be seen the structure of the tab for the original data. On the left half of the window an image can be selected, which then is displayed on the right side.

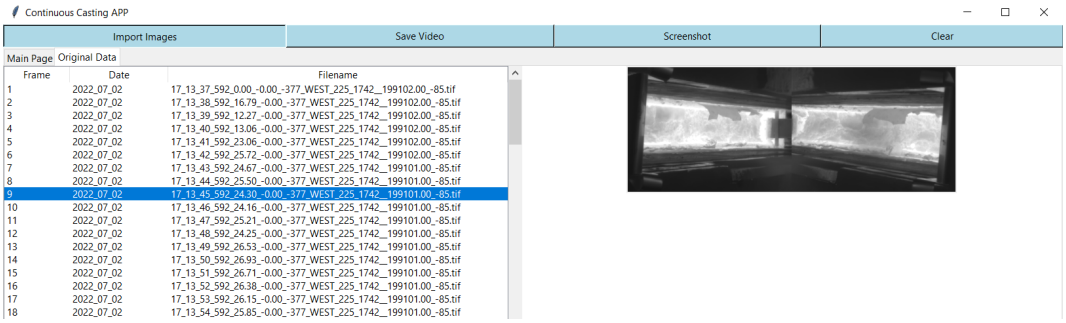


Figure 11: *Process Visualizer - Main Tab Original Data*

Frame

The term frame defines a revised input image. One frame is an image split into a left and a right picture of the mold. Each frame has a total size of 500x190 pixels.

These pixel columns and rows are used in *Chapter 3.1.3* to perform various statistical calculations. *Figure 12* shows the division and size of a frame.

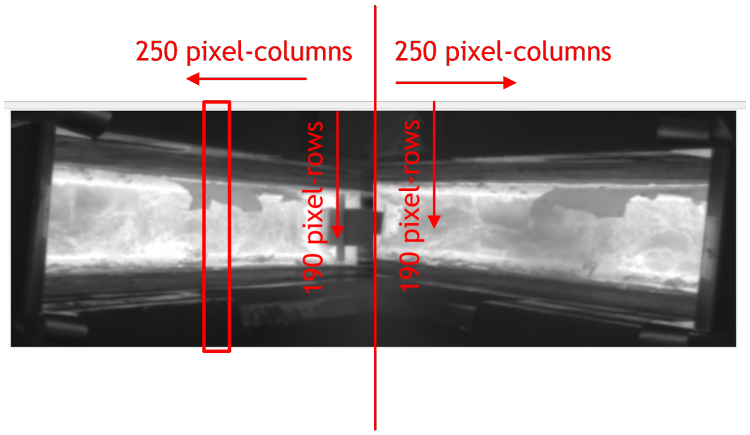


Figure 12: *Process Visualizer - Example Frame*

Image Viewer

The main part of the program is the image viewer with the control over the display of the image sequence. The following buttons control the functions for visualizing the images.

Figure 13: *Process Visualizer - Videoplayer*

The buttons described in *Table 2* can be seen in *Figure 13*.

Table 2: *Description of the Videoplayer*

Buttons	Description
a) Play/Pause	To launch or pause the image viewer.
b) <<fps	To reduce the frames per second.
c) fps >>	To increase the frames per second.
d) Scale	To search for a certain frame or time and to skip a sequence of frames in the image viewer.

The image viewer is used to visualize the imported images in the form of a video. Since the creation and storage of videos would require a large amount of memory, the image viewer displays one image after the other based on the stored paths. The display speed can be changed by increasing or decreasing the amount of frames per second. The higher the number of frames per second is selected, the more images per second are displayed consecutively. Some of the analyses of *Chapter 3.1.3* are performed directly while playing the images. Others are already calculated when importing the images.

3.1.3 Statistical Analysis of Visual Data

The following chapter describes the statistical analysis we perform and visualize within the GUI. This helps in visualizing the movements on the cover powder at the bath level of the mold of the continuous caster. We have applied two different types of chart update models in the following analysis metrics. Some metrics are created either when the frames are imported or they are continuously updated frame by frame. The diagrams that have already been calculated when importing the images, contain a vertical red line. This line indicates which of the values in the diagram belong to the current frame.

Histogram

We decided to use a histogram, because it efficiently and clearly shows the asymmetry of the two frames. The histogram is a representation of the frequency distribution of the pixel intensity of one single image. A histogram with 20 bins is calculated for each frame based on the pixel values. As shown in the histogram legend (*Figure 14*), the histogram is split into a left (blue) and right (orange). These two colors represent the left and right halves of the mold, respectively.

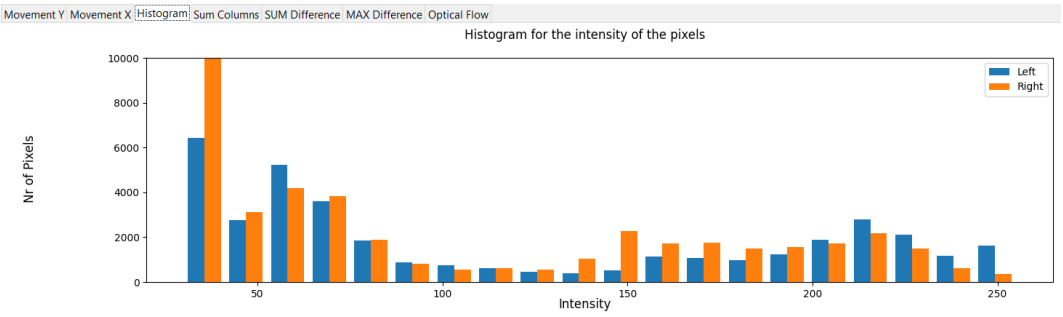


Figure 14: *Process Visualizer - Example Histogram*

Sum of the Pixel-Intensity of each Column

In this tab we display the sum of the pixel-intensity for each column of one frame. Each frame has 2x250 pixel columns (for the left and the right part of the mold) and 190 rows of pixels (see *Page 18* in *Figure 12*). This diagram shows how equal the intensities of the left and the right part of the mold are per column. This diagram is also being constantly recalculated while the image viewer is playing.

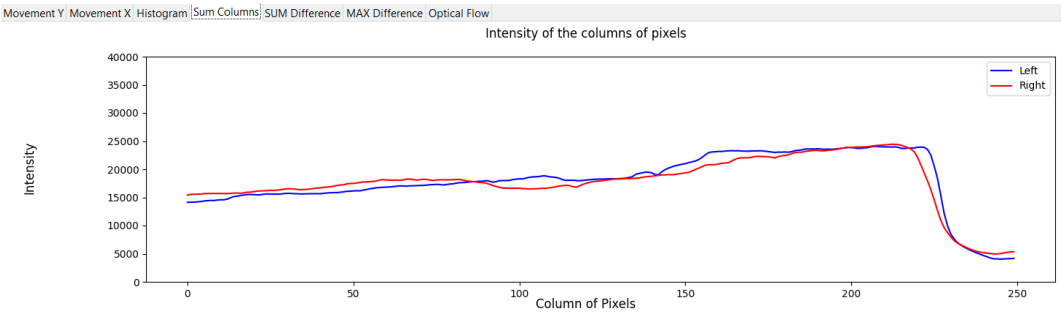


Figure 15: *Process Visualizer - Example Sum Columns*

Pixel-Intensity-Calculations retrieved from the Histogram

We decided to perform further analysis of the movements using the already calculated histograms (*Chapter 3.1.3*). Since the histograms were only visualized per frame, we used all histograms to perform the following two calculations. This has the advantage of providing an overview of asynchronous movements over the entire runtime.

$$\alpha_{Frame} = \sum_{i=1}^{N} \left((a_i - b_i)^2 \right) \tag{1}$$

In *Equation (1)*, α_{Frame} represents the calculated sum of the pixel-intensity difference per frame. N is the number of bars calculated in the histogram. The designations a_i and b_i represent two related bars (left and right side of the mold).

$$\alpha_{Frame} = MAX \left((a_i - b_i)^2 \right) \tag{2}$$

In *Equation (2)* we calculate the maximum square difference and not the sum of the histogram bars. The variable α_{Frame} represents the calculated maximum of the pixel-intensity difference per frame.

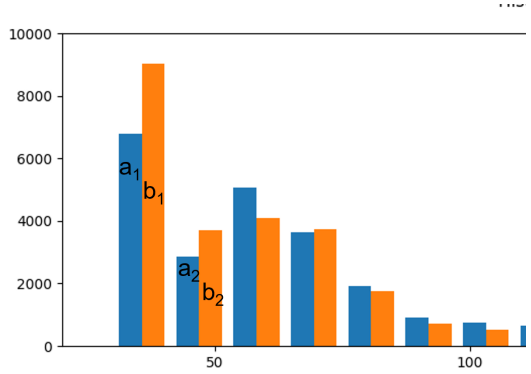


Figure 16: *Process Visualizer - Example Values from the Histogram*

Figure 16 shows a section of a histogram to give an idea of which values were used to calculate the graph shown in *Figure 17* and *Figure 18*. The resulting diagram is already created when importing the images.

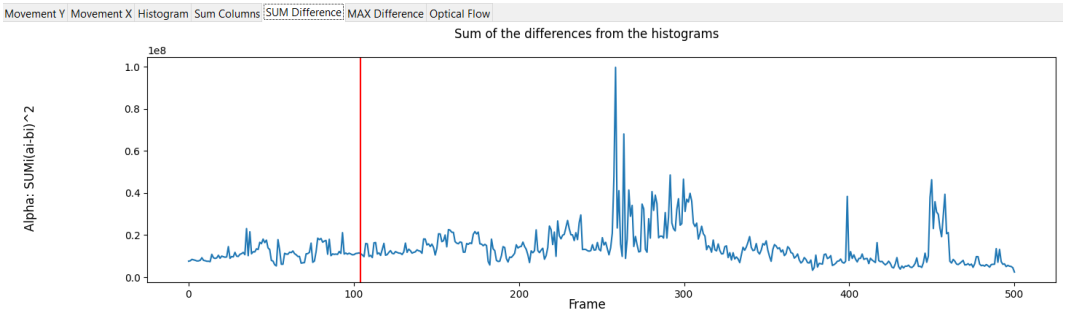


Figure 17: *Process Visualizer - Example Sum Difference*

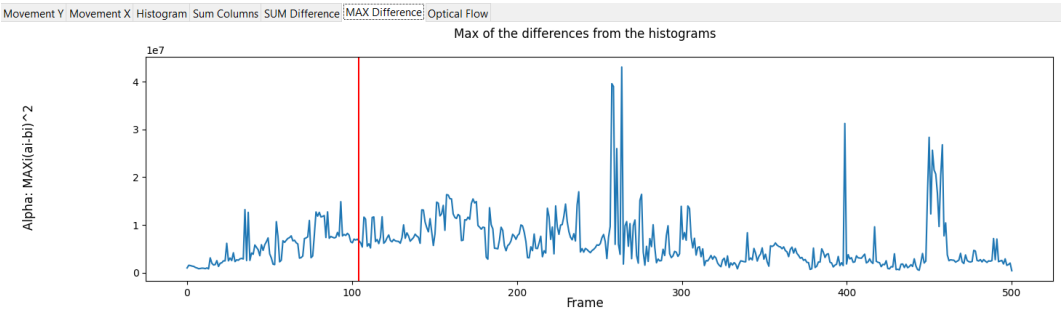


Figure 18: *Process Visualizer - Example Max Difference*

Optical Flow

The optical flow is displayed with the help of a two-frame motion estimation algorithm based on polynomial expansion. This algorithm was published by Farnebäck in 2003. It compares the previous frame with the next one. They approximate the neighborhoods of the two frames with quadratic polynomials and they apply a method for estimating displacement fields. The algorithm is given the two images as input. The result is a matrix that contains an x and a y-value for each pixel of the previous image. This x and y-value represents the point to which the pixel in the new image has moved from the previous x and y-value of the pixel. (Farnebäck, 2003)

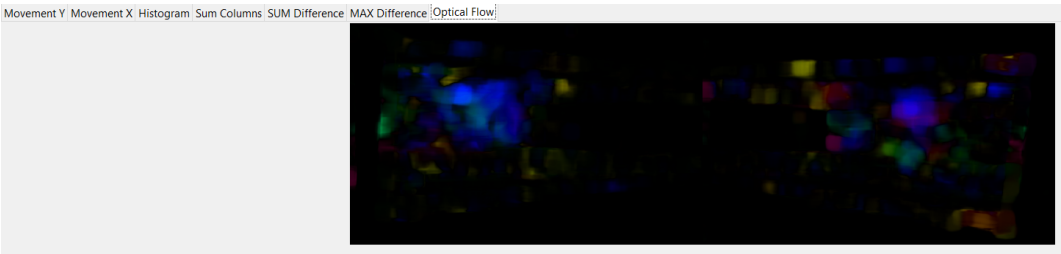


Figure 19: *Process Visualizer - Example Optical Flow*

Movements of the pixels along the X and Y axis

These diagrams describe the intensity of the movements in the x and the y-direction. We used the Farnebäck's Optical Flow Algorithm (Farnebäck, 2003) to calculate the movements on the x and the y-axis of the mold. When calculating the optical flow according to Farnebäck, we obtained two lists with the same number of values, one for the x and one for the y-values. These values represent direction vectors in which the pixels move when comparing two images. We create a separate diagram for each list to show the pixel movements summed up in x and in y-direction. The value 0 describes that either no movement has been detected on the y-axis, or the left and right images of the mold balance each other out and on average both movements are synchronous. The calculation of this diagram is performed, when importing the images. A more detailed description of Farnebäck's Optical Flow Algorithm can be found in *Chapter 3.1.3*.

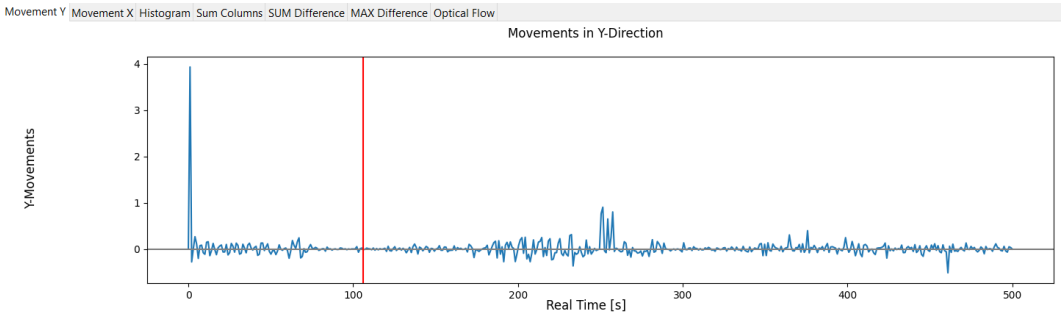


Figure 20: *Process Visualizer - Example Movement Y*

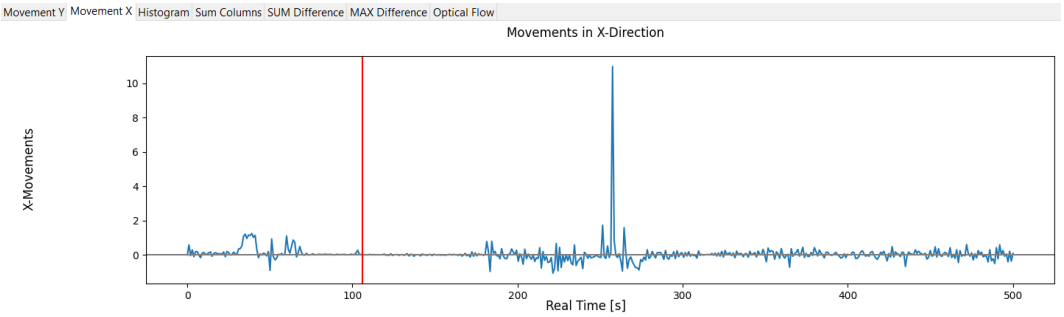


Figure 21: *Process Visualizer - Example Movement X*

3.1.4 Evaluation of the GUI

We evaluated the proposed GUI on inspecting visual data to detect anomalies. The following chapter shows which phases occur in a mold. The anomalies that lead to temperature fluctuations are also discussed. These anomalies can be easily detected by the users with the help of the GUI.

Visual Inspection of the ”Normal” Phase

As can be shown in *Figure 22, 23, 24 and 25*, interrelationships are visible in the different line graphs. All plots are computed on the same data set. It is obvious, that all metrics show similar fluctuations at the same timesteps. These diagrams give a good overview of the movements on the surface of the mold during the entire time that images were imported.

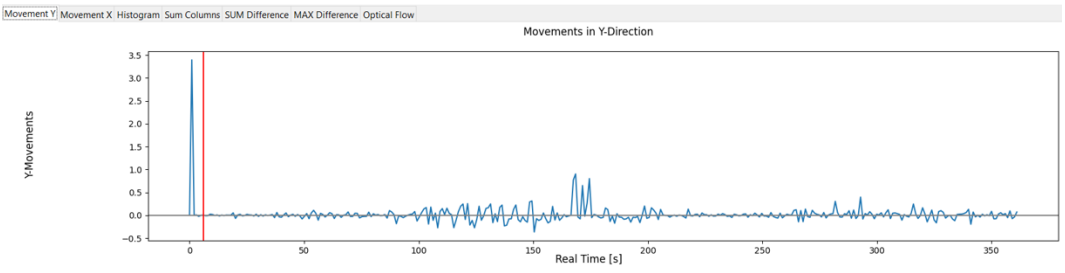


Figure 22: *Process Visualizer - Analysis Y-Movement*

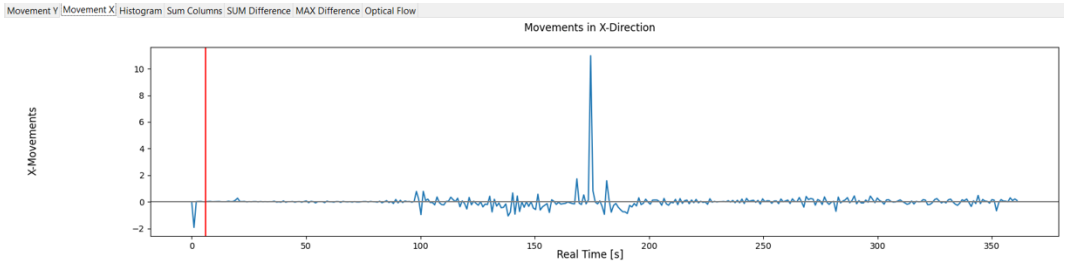


Figure 23: *Process Visualizer - Analysis X-Movement*

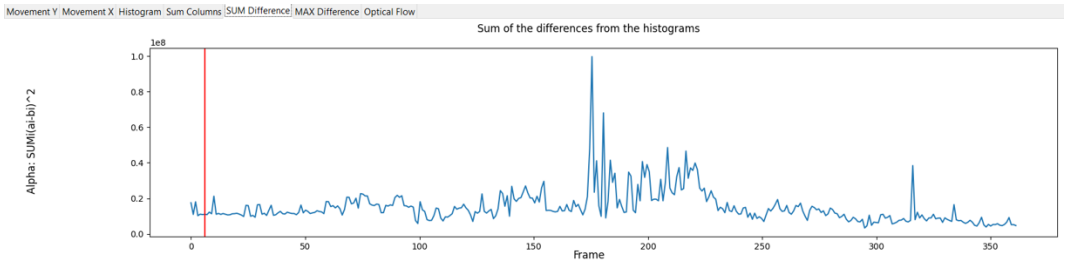


Figure 24: *Process Visualizer - Analysis Sum Difference*

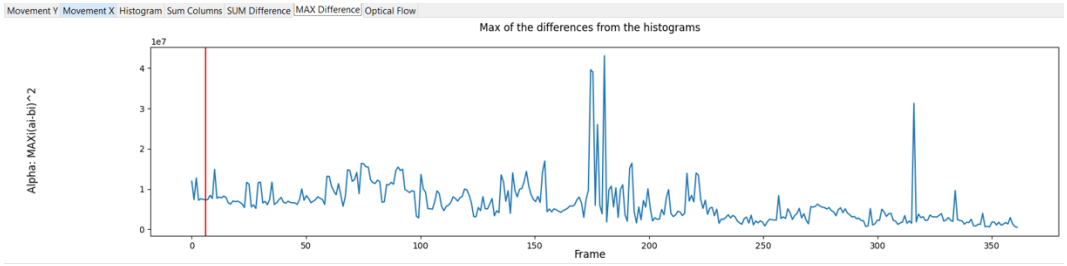


Figure 25: *Process Visualizer - Analysis Max Difference*

An example of an image of the mold without anomalies can be seen in *Figure 26*, where there are no asynchronous movements on the surface of the mold.

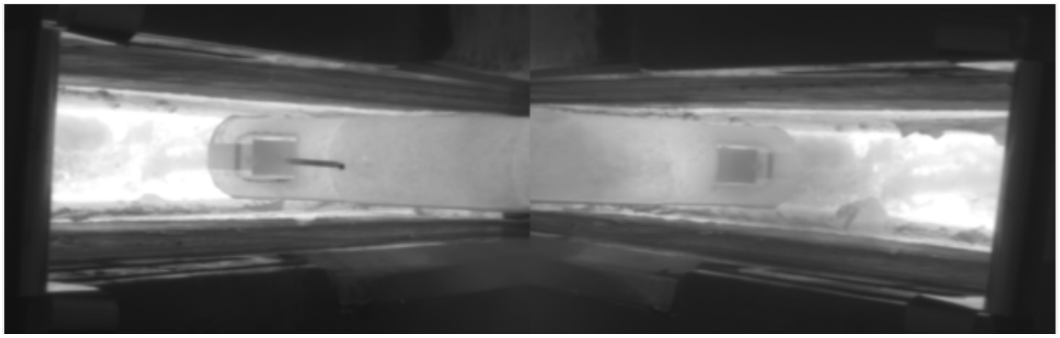


Figure 26: *Process Visualizer - Analysis Normal Phase*

The upper part of *Figure 27* shows a diagram of almost no deviations in the two sides of the mold. In contrast, a strong deviation can be seen in the lower part of this figure. Since this line diagram is in constant motion, the current movements of the surface can be easily recognized.

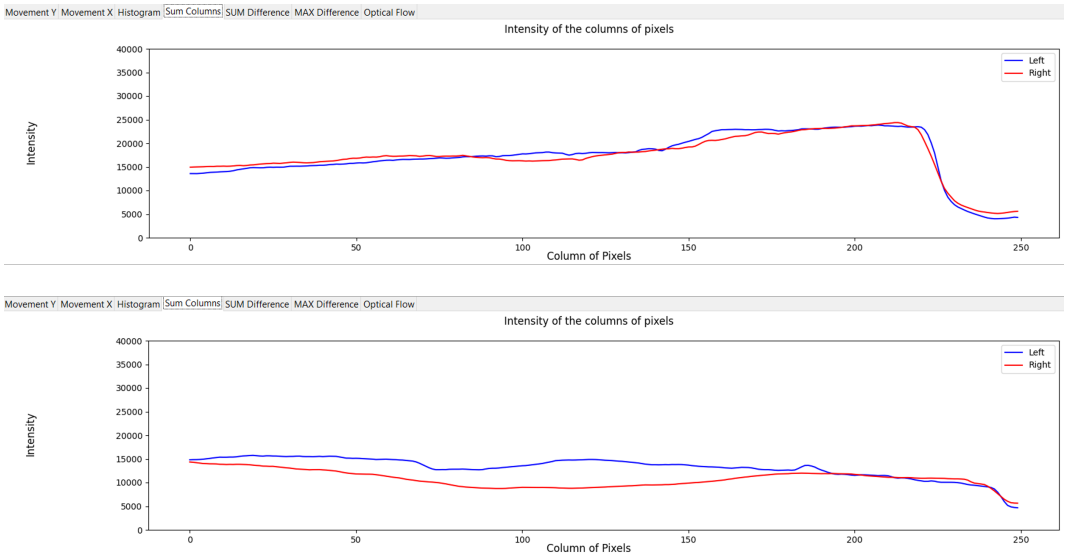


Figure 27: *Process Visualizer - Analysis Tab Sum Columns*

The visualization of the optical flow is done by displaying different colors depending on the direction in which the pixels are moving. The pixels that were classified as 'Not moving' in the calculations are displayed as black. This tool enables the observation of the liquid steel movements in the mold in real time.

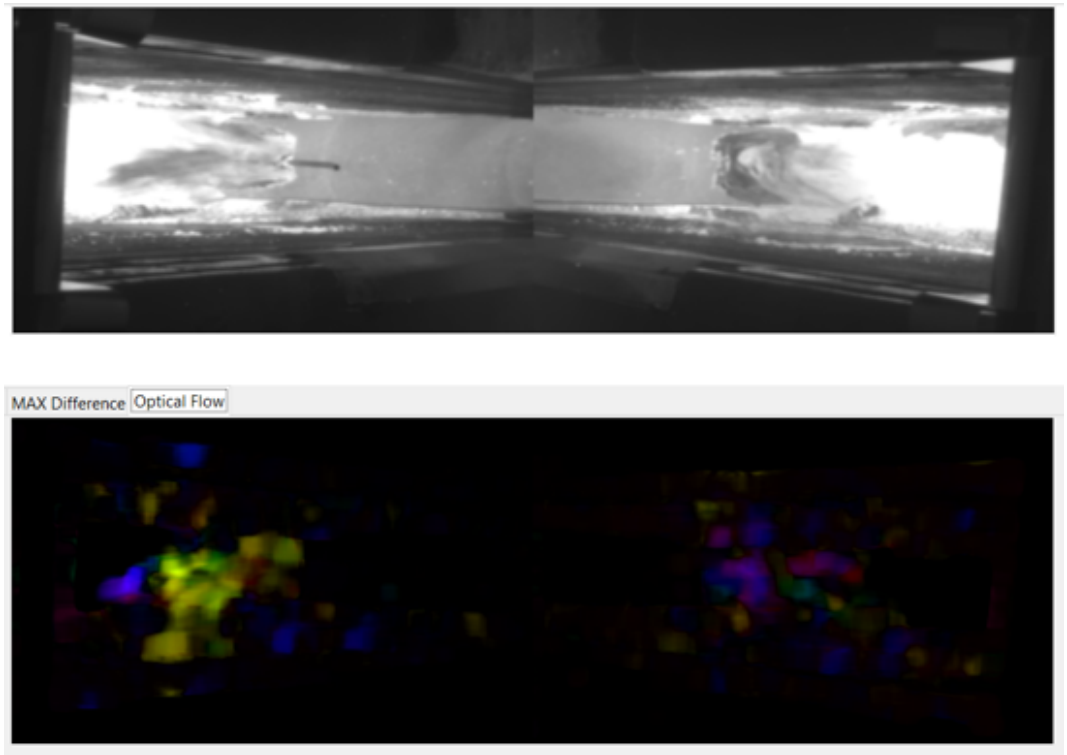


Figure 28: *Process Visualizer - Analysis Optical Flow*

Visual Inspection of Casting Powder

An event that can cause big outliers in the diagrams is the process of adding the casting powder in the mold. This causes an uneven change in temperature on the

surface of the mold. *Figure 29* shows the effect of adding the casting powder can be viewed in all graphs. The frame also clearly shows that the right side of the mold has a much lower temperature than the left side. Only the tube can be easily recognized in the picture.

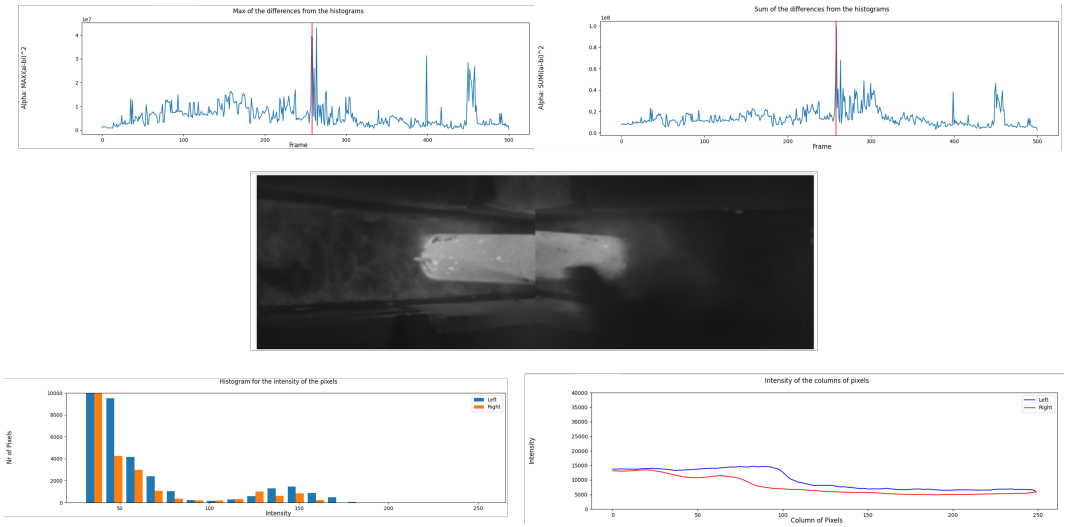


Figure 29: *Process Visualizer - Analysis adding Cover Powder*

Visual Inspection of Deer Horns

The deer horns are used to pump gas into the molten iron. The manual addition and removal of these leads to large temperature differences in the mold, which are clearly visible in the following figure. The red vertical line marks the point in time when the Deer Horns were inserted into the mold. These temperature differences lead to anomalies in the calculation of surface motions since the movements of the deer horns cannot be distinguished from the movements of the surface.

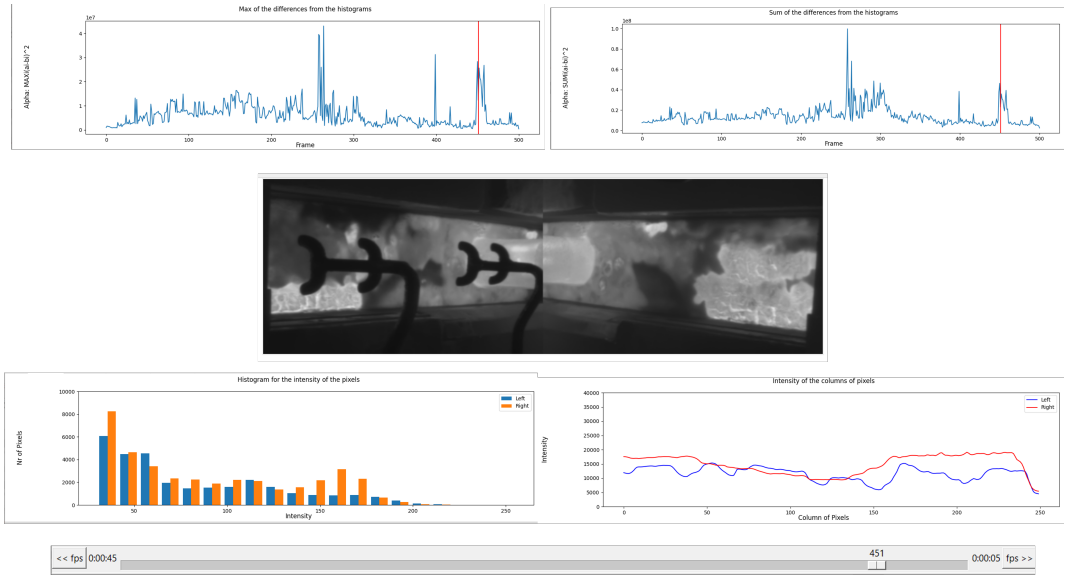


Figure 30: *Process Visualizer - Analysis adding Deer Horns*

Visual Inspection through Optical Flow

We additionally used Recurrent All-Pairs Field Transforms (RAFT) (see *Appendix A.5*) to the given image data. In contrast to the previously described statistical methods, RAFT is a deep network architecture for optical flow. RAFT has been trained to predict the optical flow directly on the data. Optical flow estimates the pixel movement of successive images in, for example, videos. (Teed and Deng, 2020)

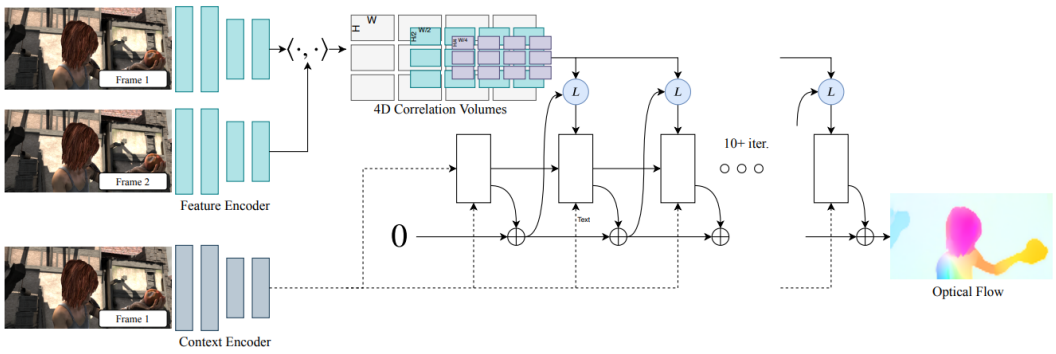


Figure 31: *Structure from RAFT (Teed and Deng, 2020)*

Figure 31 shows the structure of the RAFT model. The three main components of the model are described below.

- Feature encoder that extracts a feature vector for each pixel
- Correlation layer that generates a 4D correlation volume for all pixel pairs, with subsequent pooling to generate lower resolution volumes
- Recurrent GRU-based update operator that retrieves values from the correlation volumes and iteratively updates a flow field initialized at zero (Teed and Deng, 2020)

In this work, we deploy the RAFT model trained on Sintel dataset (Gupta, 2023). Unfortunately, the model struggles to detect any differences on the surface of the mold, as shown in *Figure 32*. Small movements could hardly be identified or not at all. Even the surroundings of the mold, which can be seen at the edges of the images, are displayed in color.

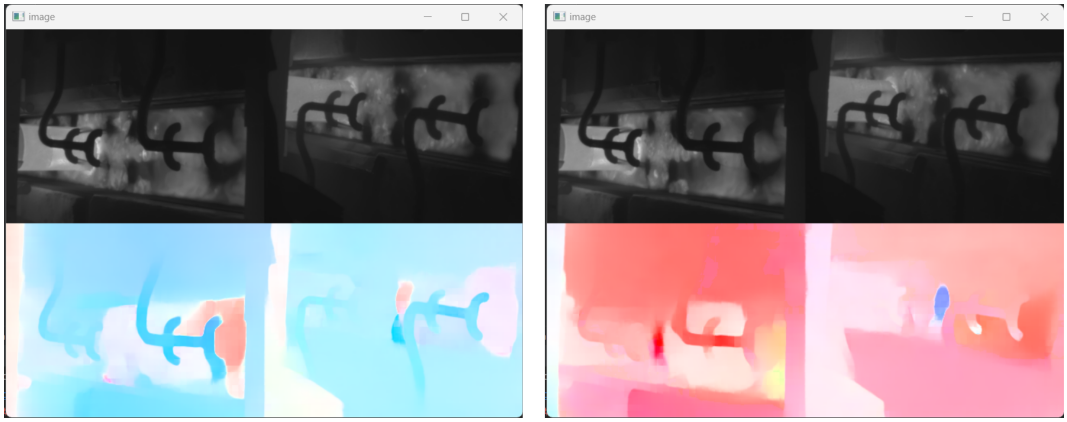


Figure 32: *Results from RAFT*

Conclusion

With the above evaluation, we showed that the proposed GUI facilitates the detection of various phenomena occurring in the mold, that can potentially lead to end-product defects. We showed that it facilitates the effortless inspection and analysis of anomalies in the mold through the inspection of metrics provided by the GUI (*Figure 33*). Such anomalies can be, for example, the addition of casting powder, or the addition of the Deer Horns. These anomalies cause asynchronous movements on the bath level of the mold which are easily detected via the GUI and clearly displayed in the form of diagrams such as a histogram or by mapping the optical flow. Users can benefit of these analyses to make decisions to avoid defects in the end-product.

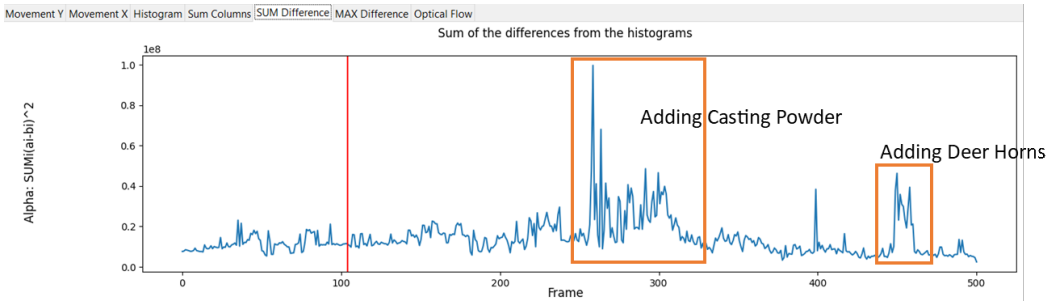


Figure 33: *Anomalies in the Mold*

3.2 Steel Product Defect Detection with a Deep Neural Network

This chapter focuses on Steel Product Defect Detection with a Mask R-CNN. This chapter describes how the Mask R-CNN model is used to detect defects in steel products. The chapter explains the architecture of the Mask R-CNN and how it works to identify and segment defects in images of steel products. The results can be viewed in *Chapter 3.2.4*.

3.2.1 Deep Neural Networks

A standard neural network consists of interconnected neurons, which generate activations based on inputs and weights. The learning process involves finding the best weights to produce the desired behavior. Deep learning is a type of neural network that focuses on accurate credit assignment over many computational steps in complex problems. (Schmidhuber, 2015)

We utilize Deep Learning as it surpasses human capabilities and achieves superhuman performance (Janiesch, Zschech, and Heinrich, 2021). Deep learning is used for steel defect detection because it offers several advantages over traditional analysis. First, Deep Learning algorithms can automatically learn complex and abstract features from large amounts of data, making them ideal for detecting subtle and varied defects in steel. Second, deep-learning models are capable of processing highly variable and noisy data, making them robust to variations in lighting conditions, image quality, and other factors that can affect steel defect detection. Third, deep learning models can be fine-tuned for a specific task, enabling accurate and efficient detection of specific steel defects. Overall, the high accuracy, robustness, and flexibility of Deep Learning make it a powerful tool for detecting steel defects and improving the quality and safety of steel products. (O'Mahony et al., 2020)

In *Figure 34* shows a basic deep neural network.

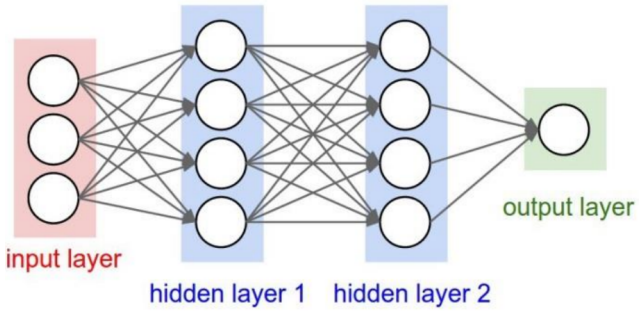


Figure 34: *Basic Deep Neural Network (Krishna et al., 2018)*

Such networks are trained by receiving an image as input and then the network is informed about the output of the image. Neural networks are basically expressed by the number of layers and the depth of the neural network. The generation of inputs and outputs is done by the layers. The most popular algorithm for implementing the Deep Learning technique is the Convolutional Neural Network (CNN). It consists of feature detection and classification layers. (Krishna et al., 2018)

This type of network is biologically inspired and is used in computer vision in the field of image classification and object detection. Each layer of the network in the CNN architecture is three-dimensional and has a spatial extent and a depth equal to the number of features. (Aggarwal, 2019, p. 40)

This chapter serves as an introduction to the field of machine learning. The network described in this thesis in *Chapter 3.2.2*, the Mask R-CNN, is also a Convolutional Neural Network. This special type of neural network is used for processing data with grid-like topology. This includes image data, for example, these have a 2D grid of pixels. The so-called convolution is a special type of linear operation. Instead of a general matrix multiplication, Convolutional Neural Networks exhibit this convolution in at least one of their layers. (Goodfellow, Bengio, and Courville, 2016, p. 326)

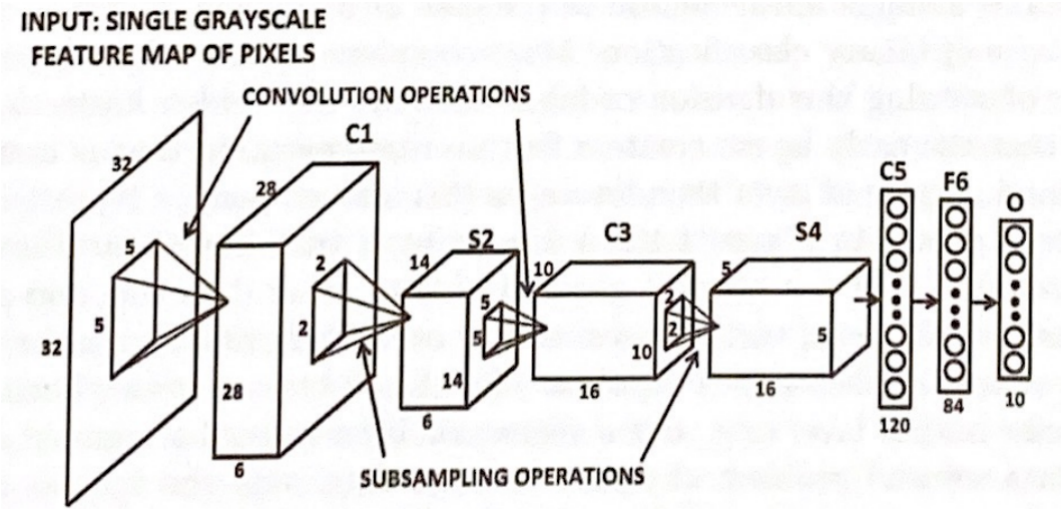


Figure 35: *Structure of the LeNet-5 (Aggarwal, 2019, p. 41)*

In the first layer of the CNN, the input layer, the number of features depends on the color channels, for example RGB. In the hidden channels, hidden feature maps are represented that encode different types of shapes in the image. *Figure 35* shows one of the first Convolutional Neural Networks, the LeNet-5. This network has only an input image in greyscale, therefore the input layer has only a depth of 1. In contrast, a three-dimensional image (RGB) has an input layer depth of three. (Aggarwal, 2019, p. 41)

Image Classification

The division into groups and categories based on characteristics is called classification. Image Classification divides images into groups or categories based on features. By training the program with labeled data, features are first filtered out by mean of feature extraction (where features such as corners and edges are extracted) and then a classification module performs the classification based on the previously extracted features. The goal for this task is to assign a class/label to an image. The possible classes/labels are previously defined. (Krishna et al., 2018)

Object Detection

We assume to have a *fixed* set of objects which are represented in an image. Now we want to identify a rectangular region in this image in which the object is located. Then this problem would be solved with object localization. This rectangular region can be identified with four numbers uniquely and is called bounding box. By default, these four numbers consist of the coordinates of the upper left corner of the box and the two dimensions. If there is a *variable* number of objects of different classes in the image, object detection is used to solve this problem. This is very similar to object

localization. The goal here is to identify all objects of an image and assign them to the different classes. (Aggarwal, 2019, pp. 364–365)

Instance Segmentation

When using image segmentation, the entire image is divided into several segments. This approach makes it possible to locate objects and boundaries in images. (Tan, 2016, p. 167) Image segmentation is an important phase in the analysis of images. It can be applied to a set of images as well as to videos. Generally, the basic goal is to reduce the data of an image in order to simplify the subsequent analysis process. The image should be simplified so that only those objects are analyzed in the object analysis phase that are of importance. Region-based segmentation is one of the most well-known techniques in image segmentation. This technique is applied in Mask R-CNN, which was used in this thesis to analyze the data of the steel defects dataset. (Abdulateef and Salman, 2021)

In *Chapter 3.2.2* information about Mask R-CNN can be found. Semantic segmentation and instance segmentation are two different techniques of image segmentation. The distinctions between Classification, Object Detection, Instance and Semantic Segmentation are illustrated in *Figure 36*.
Instance segmentation is a technique of image segmentation in which individual instances of objects that belong to the same class are labeled differently. The problem of object detection as well as that of semantic segmentation is solved by instance segmentation. (Hafiz and Bhat, 2020)

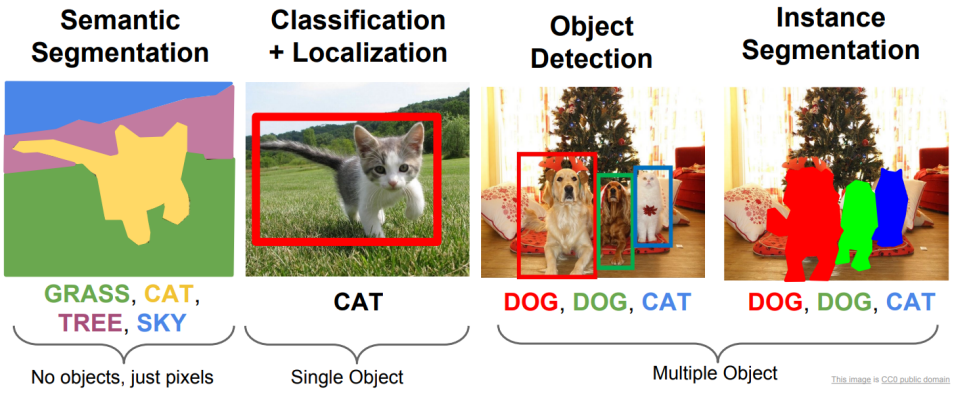


Figure 36: *Differences between Classification, Object Detection, Semantic and Instance Segmentation (Rieder and Verbeet, 2019)*

3.2.2 Mask R-CNN

Mask R-CNN (Mask Region-based Convolutional Neural Network) is based on the Faster R-CNN (Faster Region-based Convolutional Neural Network).

It is a framework for object instance segmentation. The existing branch for bounding box recognition from the Faster R-CNN is extended in the Mask R-CNN by adding a branch for predicting an object mask. The Mask R-CNN is easy to generalize and includes instance segmentation, bounding-box object detection and person keypoint detection. The correct detection of all objects in an image is required for instance segmentation. The goal is to classify each pixel into a fixed set of categories without

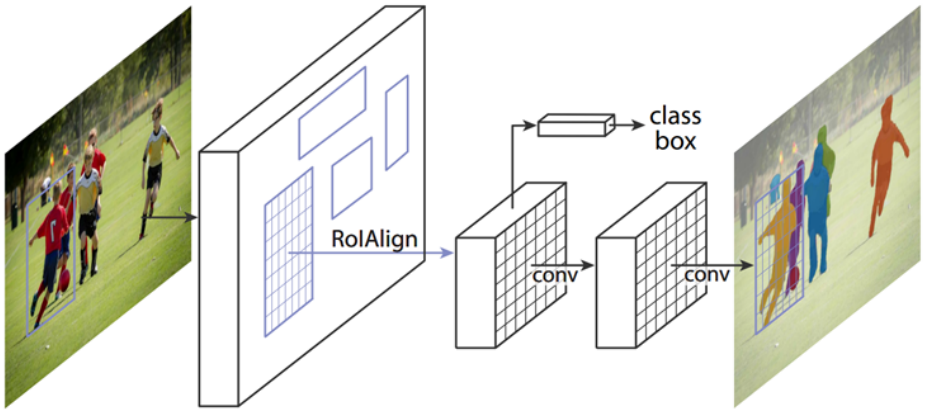


Figure 37: *Structure of the Mask R-CNN (He et al., 2017)*

differentiating object instances. To understand the Mask R-CNN approach, the two stages of Faster R-CNN are described first. (He et al., 2017)

Stage 1 of the Faster R-CNN

The Region Proposal Network (RPN) is the first stage of the Faster R-CNN which proposes candidate object bounding boxes. A Region Proposal Network requires an arbitrary sized image as input and outputs a series of rectangular object proposals, each with an object value. (Ren et al., 2015) The structure of the RPN is shown in *Figure 38*.

Stage 2 of the Faster R-CNN

In the second stage features are extracted from each candidate box by using RoIPool and classification and bounding box regression is performed. (He et al., 2017)
The Region of Interest (RoI) is a rectangular window defined by four numbers, like the bounding box. RoIPool is used to convert this region of interest of an image into a smaller feature map. (Girshick, 2015)

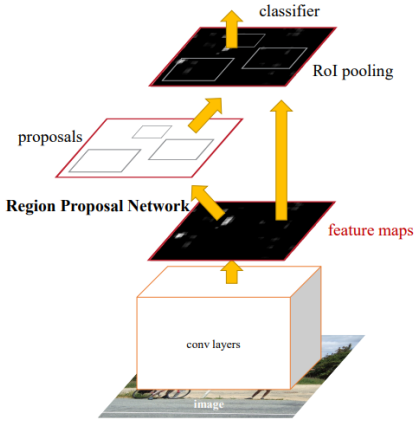


Figure 38: *Structure of the Region Proposal Network (Ren et al., 2015)*

Extension from Faster R-CNN to Mask R-CNN

These two stages are taken over by the Mask RCNN, with the difference that in the second stage a third output is added in parallel. In addition to predicting the class and bounding box, the Mask R-CNN outputs a binary mask for each region of interest. In comparison to most recent systems, the classification does not depend on the mask prediction in this case.

Faster R-CNN extends Mask R-CNN intuitively, good results are achieved by the correct construction of the mask branch. Faster R-CNN cannot be used to design pixel-to-pixel matching between the inputs and outputs of the network.

RoIPool is the most important operation in Faster R-CNN for processing instances, which performs coarse spatial quantization for feature extraction. To address the misalignment, the Mask R-CNN uses RoIAlign instead. RoIAlign is a quantization-free

layer that maintains accurate spatial positions. (He et al., 2017) In *Figure 37* shows the structure of the Mask R-CNN.

Loss-Function

The total loss L_{total} for each sampled RoI is calculated from three losses (He et al., 2017):

$$L_{total} = L_{cls} + L_{box} + L_{mask} \tag{3}$$

- L_{cls} - Classification Loss - cross entropy loss
- L_{box} - Bounding-Box Loss - smooth L1 loss
- L_{mask} - Mask Loss - average binary cross entropy loss

The classification loss and the bounding box loss are calculated as follows, these are calculated the same as for the Faster R-CNN. (Ren et al., 2015)

$$L_{cls} = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) \tag{4}$$

Where $L_{cls}(p_i, p_i^*)$ is calculated as follows in *Equation (5)*.

$$L_{cls}(p_i, p_i^*) = -p_i^* \; log \; p_i - (1 - p_i^*) \; log \; (1 - p_i) \tag{5}$$

The loss for the bounding boxes is a smooth L1 loss, for which *Equation (6)* is used for the calculation.

$$L_{box} = \frac{\lambda}{N_{box}} \sum_i p_i^* \; * \; L_1^{smooth}(t_i - t_i^*) \tag{6}$$

- N_{cls} - normalization term
- N_{box} - normalization term
- i - the index of an anchor in a mini-batch
- p_i - the predicted probability of anchor i being an object
- p_i^* - the ground-truth label
- λ - a balancing parameter
- L_1^{smooth} - the smoothed L1 loss
- t_i - a vector representing the 4 parameterized coordinates of the predicted bounding box
- t_i^* - a vector representing the 4 parameterized coordinates of the ground-truth box associated with a positive anchor
  (Ren et al., 2015)

The equation of the loss for the mask is an addition to the Mask R-CNN and is calculated as in *Equation (7)*. (Tao Wang et al., 2021)

$$L_{mask} = -\frac{1}{x} \sum_i x_i^* \; log \; p(x_i) - (1 - x_i^*) \; log \; (1 - p(x_i)) \tag{7}$$

- x - the number of pixels
- x_i^* - the category label where the pixel is located
- $p(x_i)$ - the probability of the x_i predicted category
  (Tao Wang et al., 2021)

3.2.3 Training Dataset

The data set we used to train the model contains four types of images with information about different steel defects. The complete data set contains a total of 12568 images. We used only the 6666 labeled images from the dataset to train the network. We split the 6666 images into a training and a test dataset. We used 80% of the data for training and the remaining ones for testing the model. Each of the images has a size of 256x1600 pixels. (Grishin et al., 2019)

The dataset consists of images along with their associated classification labels, bounding boxes, and masks. A smallest possible rectangle was placed around the color-coded pixels to represent the bounding box. We calculated the bounding boxes as described in *Appendix B.1*. The following are the examples on which the included pixels are highlighted. More than classes of defects may appear on the sample images.

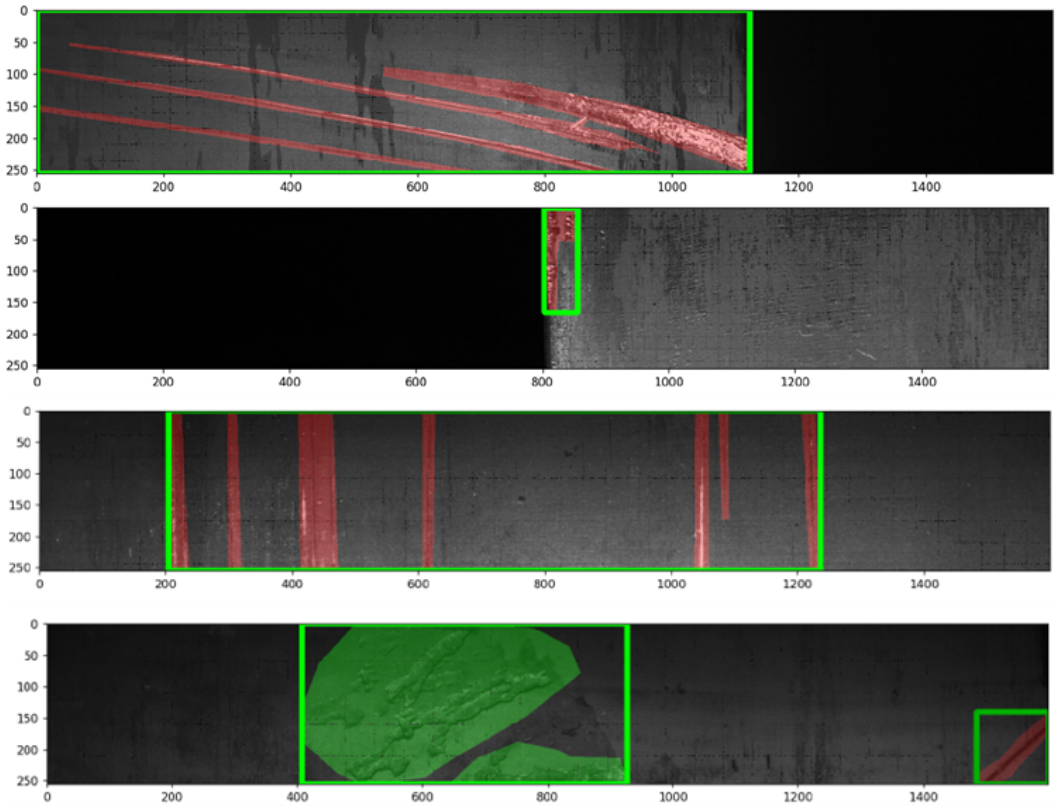


Figure 39: *Data Example - Steel Defection*

Backbone Architecture

The backbone architecture we used for the Mask R-CNN model is the ResNet50 Feature Pyramid Network (FPN) version 2 from (Li et al., 2021). The original backbone Girshick, 2015 model required more GPU memory which limited the batch size allowed by the used system 3.2.3.

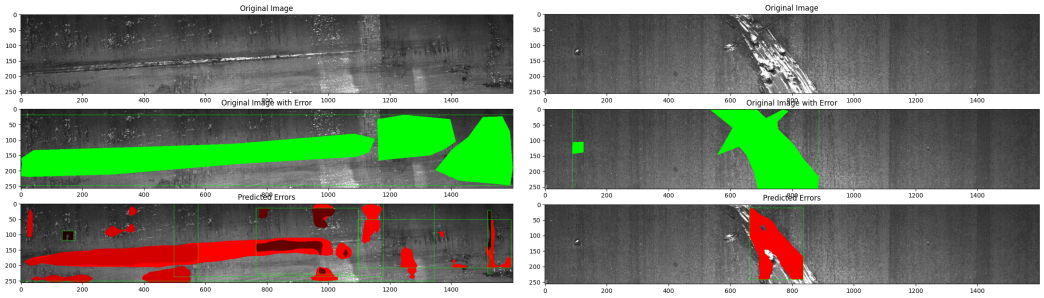
Training Set Up

We performed the training of the network on an NVIDIA GeForce RTX 3090ti GPU. The following hyperparameters specify the configuration of a machine learning model. We set the batch size to 16, which means that 16 images are processed simultaneously

during training. The learning rate was set to 0.001, which determines the step size at which the optimizer updates the model parameters. Finally, we set the image size to $[1600, 256]$.

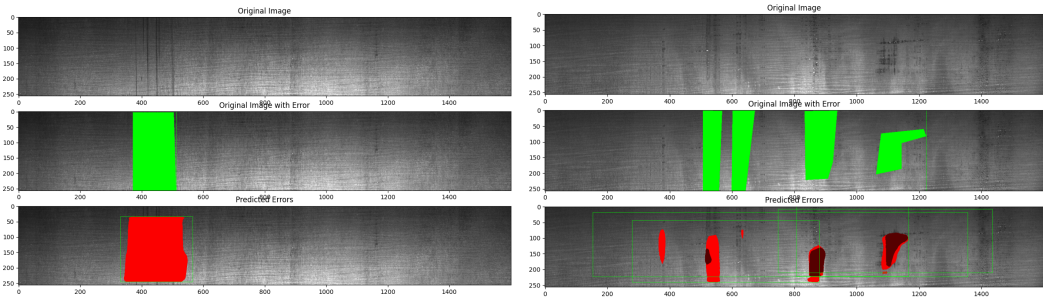
3.2.4 Results

We tested the model to a wide variety of images from the test steel dataset. The results were visualized and compared to the original image in the following figures. The score-output determines how likely this mask should be applied to the image. This score can reach from 0 (not needed) to 1 (needed). For the tests in this thesis, the score was expressed using the brightness of the mask. The brighter the red of the predicted mask, the higher the score.



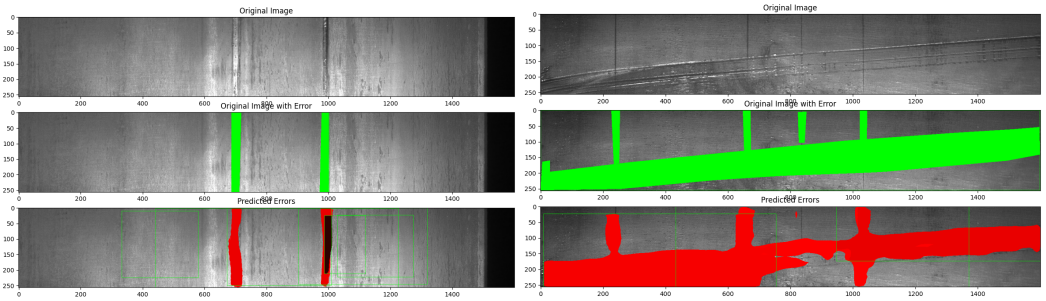
(a) Example Result 1

(b) Example Result 2



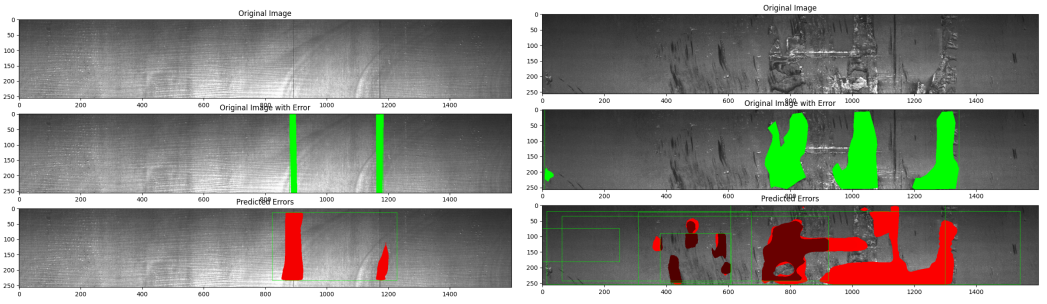
(c) Example Result 3

(d) Example Result 4



(e) Example Result 5

(f) Example Result 6

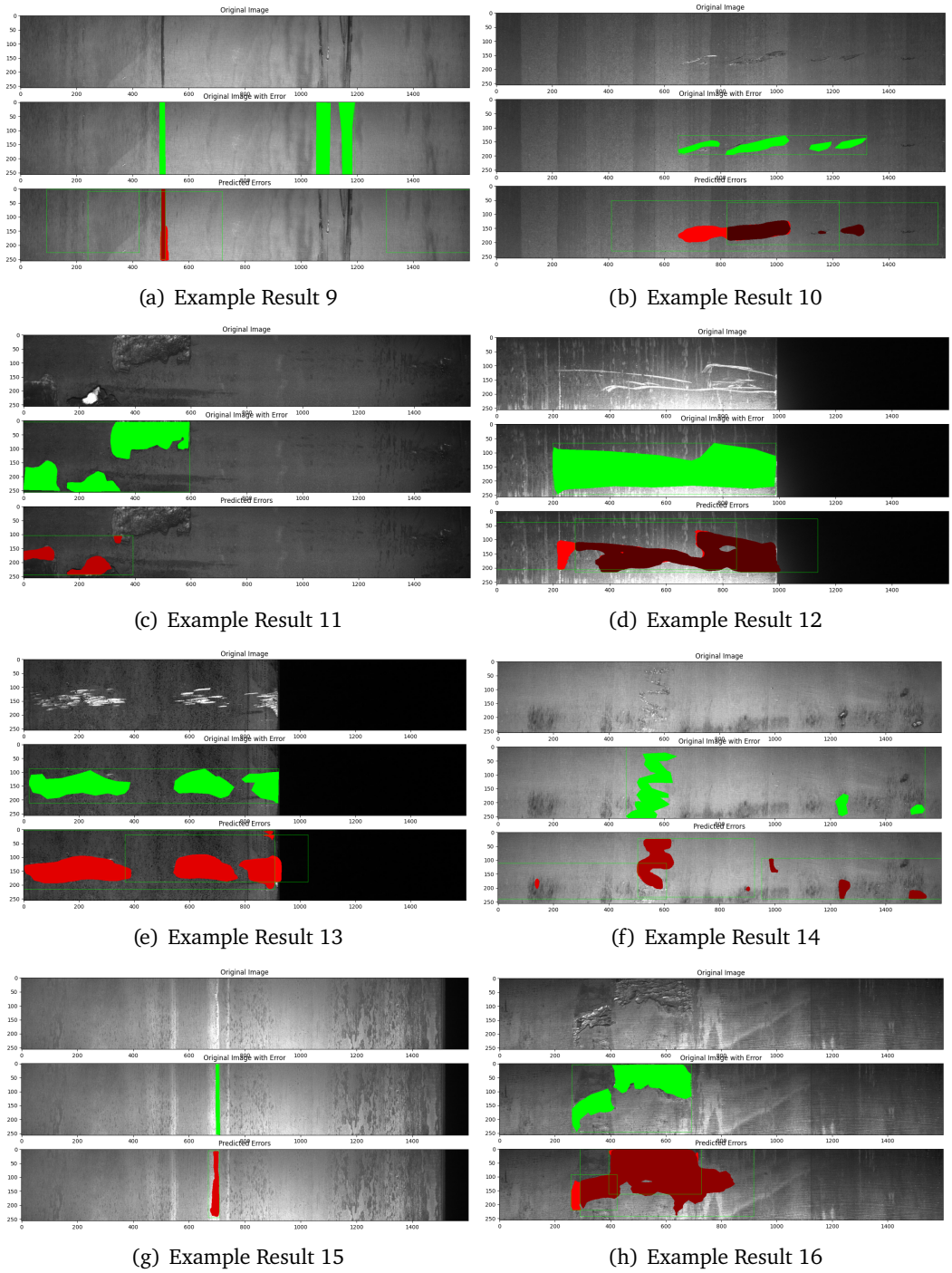


(g) Example Result 7

(h) Example Result 8

Figure 40: *Mask R-CNN Results Part 1*

As shown in *Figure 40* and *Figure 41*, Mask R-CNN correctly predicts the defects on the test images. The green segmentation shows the original defects on the image and the red shows the predicted defects. The comparison shows that the two overlap. The greater the agreement between the original defects marked in green and the predicted defects marked in red, the better the prediction of our model. In our case, as seen in the images, the agreement is very high.



(a) Example Result 9

(b) Example Result 10

(c) Example Result 11

(d) Example Result 12

(e) Example Result 13

(f) Example Result 14

(g) Example Result 15

(h) Example Result 16

Figure 41: *Mask R-CNN Results Part 2*

The diagram in *Figure 42* is a line plot, showing the performance of a machine learning model during training (red line) and testing (green line). The x-axis repre-

sents the number of epochs, while the y-axis shows the total Loss L_{total}. The red line shows how the model's performance changes during the training phase, as it updates its parameters based on the input data. As seen in the figure, the diagram overfits after about 25 epochs. It is obvious that the model greatly overfits. This is attributed to the small size of the dataset and the memory limitations of the available hardware.

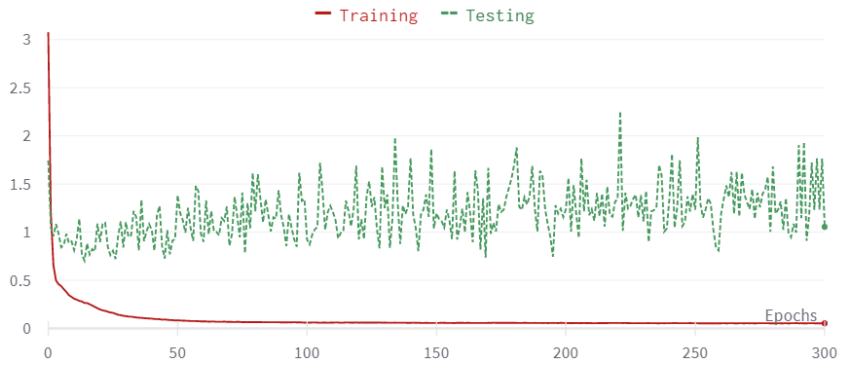


Figure 42: *Total Loss* L_{total}

3.2.5 Conclusion

In this section, we have shown that it is possible to apply Mask R-CNN with ResNet50-FPN architecture in detecting defects in steel. We found that the performance of the model depends on several factors. Since we had only a limited amount of training data available, this negatively affected the performance of the network. The limitations of the dataset and the hardware refrained is from acquiring a stronger model. Nevertheless, Mask R-CNN is a powerful model for example segmentation and object detection, and is thus a viable solution for steel defect detection.

4 Discussion

This thesis demonstrates the potential of a proposed GUI in facilitating the analysis of bath level in a mold. By using the software's calculations, anomalies in the mold can be easily detected and analyzed, leading to conclusions about the quality of the steel. We also found that Mask R-CNN with ResNet50 architecture can be effectively used to detect defects in steel, but the model's performance is impacted by the limited availability of training data. In conclusion, Mask R-CNN is a strong model for object detection and segmentation, particularly in steel defect detection.

4.1 Future work & Limitations

The GUI developed in this thesis can be extended with several machine leaning tools. Follow-up work can investigate whether and how Mask-RCNN can be applied to the continuous caster image data. If this is successful, the next goal could be to connect the application for visualizing the images from the mold with Mask R-CNN.

This study highlights some limitations in the use of Mask R-CNN for steel defect detection. One of the biggest limitations is the limited availability of labeled datasets, which affects the training and performance of the model. Another factor is the limited GPU resources, which can limit the processing power and overall efficiency of the network. These limitations suggest that further research is needed to improve the availability and quality of training data and to enhance the processing capabilities of the model.

Bibliography

Abdulateef, Salwa and Mohanad Salman (2021). "A Comprehensive Review of Image Segmentation Techniques". In: *Iraqi Journal for Electrical and Electronic Engineering* 17.2, pp. 166–175. ISSN: 1814-5892. DOI: 10.37917/ijeee.17.2.18.

Aggarwal, Charu C. (2019). *Neural networks and deep learning: A textbook*. New York: Springer. ISBN: 978-3-319-94462-3.

Degner, M. and Stahl-Institut VDEh (2007). *Stahlfibel*. Verlag Stahleisen. ISBN: 9783514007413. URL: https://books.google.at/books?id=LgX3GAAACAAJ.

Farnebäck, Gunnar (June 2003). "Two-Frame Motion Estimation Based on Polynomial Expansion". In: vol. 2749, pp. 363–370. ISBN: 978-3-540-40601-3. DOI: 10.1007/3-540-45103-X_50.

Girshick, Ross (Apr. 2015). "Fast r-cnn". In: DOI: 10.1109/ICCV.2015.169.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Grishin, Alexey et al. (2019). *Severstal: Steel Defect Detection*. URL: https://kaggle.com/competitions/severstal-steel-defect-detection.

Gupta, Vikas (2023). Visited on 6th Feb. 2023. URL: https://github.com/spmallick/learnopencv/tree/master/Optical-Flow-Estimation-using-Deep-Learning-RAFT.

Hafiz, Abdul Mueed and Ghulam Mohiuddin Bhat (2020). "A survey on instance segmentation: state of the art". In: *International Journal of Multimedia Information Retrieval* 9.3, pp. 171–189. ISSN: 2192-662X. DOI: 10.1007/s13735-020-00195-x.

He, Kaiming et al. (2017). "Mask R-CNN". In: *CoRR* abs/1703.06870. arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

Janiesch, Christian, Patrick Zschech, and Kai Heinrich (2021). "Machine learning and deep learning". In: *Electronic Markets* 31.3, pp. 685–695. ISSN: 1422-8890. DOI: 10.1007/s12525-021-00475-2.

Krishna, M et al. (Mar. 2018). "Image classification using Deep learning". In: *International Journal of Engineering & Technology* 7, p. 614. DOI: 10.14419/ijet.v7i2.7.10892.

Li, Yanghao et al. (2021). "Benchmarking Detection Transfer Learning with Vision Transformers". In: *CoRR* abs/2111.11429. arXiv: 2111.11429. URL: https://arxiv.org/abs/2111.11429.

O'Mahony, Niall et al. (2020). "Deep Learning vs. Traditional Computer Vision". In: *Advances in Computer Vision*. Ed. by Kohei Arai and Supriya Kapoor. Cham: Springer International Publishing, pp. 128–144. ISBN: 978-3-030-17795-9.

Ren, Shaoqing et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497. arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

Rieder, Mathias and Richard Verbeet (Sept. 2019). "Robot-Human-Learning for Robotic Picking Processes". In: DOI: 10.15480/882.2466.

Rui Liu et al. (2014). "Measurements of Molten Steel Surface Velocity and Effect of Stopper-rod Movement on Transient Multiphase Fluid Flow in Continuous Casting". In: *ISIJ International* 54.10, pp. 2314–2323. DOI: 10.2355/isijinternational.54.2314.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003. URL: https://arxiv.org/pdf/1404.7828.pdf.

Schwerdtfeger, Klaus, ed. (1992). *Metallurgie des Stranggiessens: Giessen und Erstarren von Stahl*. Düsseldorf: Stahleisen. ISBN: 3514003505. DOI: Klaus.

Tan, Ying (2016). "Chapter 11 - Applications". In: *Gpu-Based Parallel Implementation of Swarm Intelligence Algorithms*. Ed. by Ying Tan. Morgan Kaufmann, pp. 167–177. ISBN: 978-0-12-809362-7. DOI: https://doi.org/10.1016/B978-0-12-809362-7.50011-X. URL: https://www.sciencedirect.com/science/article/pii/B978012809362750011X.

Tao Wang et al. (2021). "Tea picking point detection and location based on Mask-RCNN". In: *Information Processing in Agriculture*. ISSN: 2214-3173. DOI: 10.1016/j.inpa.2021.12.004. URL: https://www.sciencedirect.com/science/article/pii/S2214317321000962.

Teed, Zachary and Jia Deng (2020). "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow". In: *CoRR* abs/2003.12039. arXiv: 2003.12039. URL: https://arxiv.org/abs/2003.12039.

A APPENDIX ONE - Software

The code listed in the following chapter and the following image serves as an extension to the referenced chapters.

A.1 Method for Calculating the Histogram for the Intensity of the Pixels

This chapter is an extension to *Chapter 3.1.3*.

Listing 1: *Create the Histogram for the Intensity of the Pixels in the Application*

```
def refresh_figure2(self, left_pic, right_pic):
    """Refresh of the histogram for the intensity of the
       pixels after every image"""
    # left_pic, right_pic -> ndarray:(190,250,3)
    # last dimension = 3x the same number
    left_pic_f2 = left_pic[:, :, 0].ravel()
    right_pic_f2 = right_pic[:, :, 0].ravel()

    self.ax2.remove()
    self.ax2 = self.figure2.add_subplot()
    self.ax2.hist([left_pic_f2, right_pic_f2], bins=20,
       range=[30, 255])
    self.ax2.set_ylim([0, 10000])
    self.ax2.legend(['Left', 'Right'])
    self.ca_figure2.draw_idle()
```

A.2 Method for Calculating the Sum of the Pixel-Intensity for each Column

This chapter is an extension to *Chapter 3.1.3*.

Listing 2: *Create the Line Chart for the Sum of the Pixel-Intensity for each Column*

```
def refresh_figure3(self, left_pic, right_pic):
    """Refresh of the line-chart for the intensity of the
       columns of pixels"""
    # left_pic, right_pic -> ndarray:(190,250,3)
    # last dimension = 3x the same number
    left_pic = np.sum(left_pic[:, :, 0], 0)
    right_pic = np.sum(right_pic[:, :, 0], 0)

    self.ax3.remove()
    self.ax3 = self.figure3.add_subplot()
    self.ax3.plot(np.arange(len(left_pic)), left_pic, color=
       'b')
    self.ax3.plot(np.arange(len(right_pic)), right_pic,
       color='r')
    self.ax3.set_ylim([0, 40000])
    self.ax3.legend(['Left', 'Right'])
    self.ca_figure3.draw_idle()
```

A.3 Method for Calculating Alpha

This chapter is an extension to *Chapter 3.1.3*.

Listing 3: *Create the Charts for Tab Sum Difference and Tab Max Difference*

```
def create_figure4_5(self):
    """Method for calculating the difference alpha: SUMi(ai-bi)^2 and MAXi(ai-bi)^2"""
    list_max = []
    list_sum = []
    for paths_str in self.paths:
        paths_str = zlib.decompress(paths_str).decode("utf-8").split(',')
        for path in paths_str:
            if path == '':
                continue
            try:
                path = self.path_fix_part + path

                img = cv2.imread(path)
                # original shape = 640x240)
                # image = image[0:320, 0:120]

                '''split into left and right and turn left side 180 degree'''
                left_pic = img[50:, 50:300]
                right_pic = img[:190, 390:640]
                left_pic = left_pic[:, :, 0].ravel()
                right_pic = right_pic[:, :, 0].ravel()

                y_left, x_left = np.histogram(left_pic, 50)
                y_right, x_right = np.histogram(right_pic, 50)

                diff = y_left - y_right
                diff = diff * diff
                maximum_hist = diff.max()
                sum_hist = diff.sum()

                list_max.append(maximum_hist)
                list_sum.append(sum_hist)

            except TypeError:
                continue
```

A.4 Method for Calculating the Optical Flow

This chapter is an extension to *Chapter 3.1.3*.

Listing 4: *Create the Optical Flow in the Application*

```
def refresh_figure7(self, image):
```

```
    """Generate a new image with the optical flow from
       Farneback"""

    # Split and rotate the left part to have the same color
       map
    left_pic = image[:, :480]
    right_pic = image[:, 480:]

    left_pic = cv2.rotate(left_pic, cv2.ROTATE_180)
    image = cv2.hconcat([left_pic, right_pic])

    # Create the optical flow from Farneback for the whole
       image
    new_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    flow = cv2.calcOpticalFlowFarneback(self.figure7_prev,
       new_image, None, 0.5, 3, 15, 3, 5, 1.2, 0)

    # Computes the magnitude and angle of the 2D vectors
    magnitude, angle = cv2.cartToPolar(flow[..., 0], flow
       [..., 1])

    # Sets image hue according to the optical flow direction
    self.mask_figure7[..., 0] = angle * 180 / np.pi / 2

    # Sets image value according to the optical flow
    # magnitude (normalized)
    self.mask_figure7[..., 2] = cv2.normalize(magnitude,
       None, 0, 255, cv2.NORM_MINMAX)

    # Converts HSV to RGB (BGR) color representation
    rgb = cv2.cvtColor(self.mask_figure7, cv2.COLOR_HSV2BGR)

    # Split into left and right
    left_pic = rgb[:, :480]
    right_pic = rgb[:, 480:]

    # turn left pic around, and put it together
    left_pic = cv2.rotate(left_pic, cv2.ROTATE_180)
    rgb = cv2.hconcat([left_pic, right_pic])

    rgb = ImageTk.PhotoImage(image=Image.fromarray(rgb))

    # Changes the image in figure 7
    self.figure7.configure(image=rgb)
    self.figure7.image = rgb
```

A.5 Method for using RAFT

This chapter is an extension to *Chapter 3.1.4*. The original code (Gupta, 2023) was changed and used in this thesis to apply RAFT to the mold image data.

A.6 Structure of the Software-Code

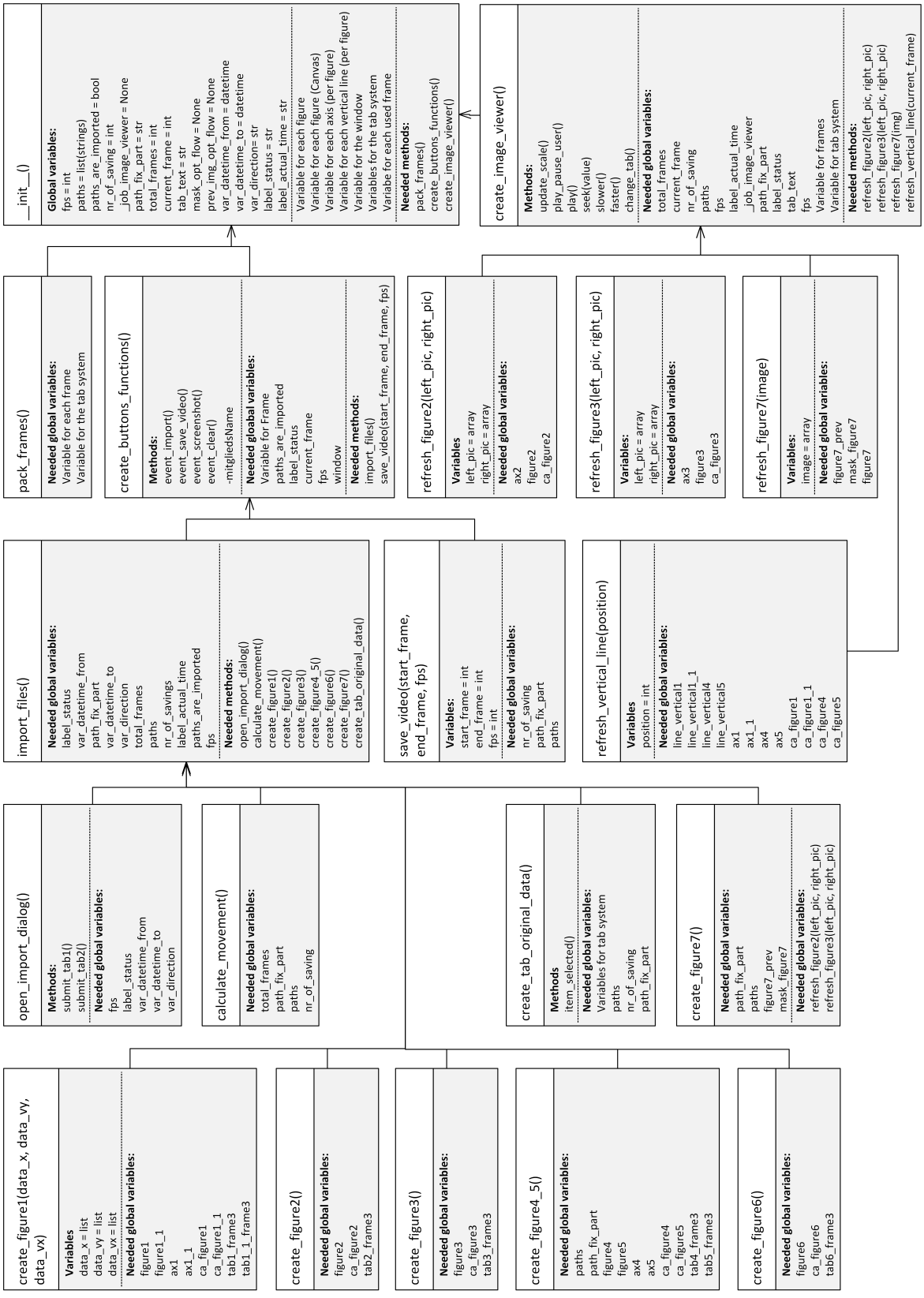


Figure 43: *Process Visualizer - Class Diagram of the Code*

B APPENDIX TWO - Mask R-CNN

The code listed in the following chapter serves as an extension to the referenced chapters.

B.1 Method for Creating the Bounding Boxes

This chapter is an extension to *Chapter 3.2.3*.

Listing 5: *Creating the Bounding Boxes*

```
# use the masks to generate a bounding box for each object
# num_objs = the number of defects (number of masks for one image)
boxes = torch.zeros([num_objs, 4], dtype=torch.float32)
for i in range(num_objs):
    x, y, w, h = cv2.boundingRect(masks[i])
    boxes[i] = torch.tensor([x, y, x + w, y + h])
    # x,y: left top coordinate; w,h: width and height of bb
```

B.2 Description of the Steel Defection Dataset

The information for the test image is given as follows:

Table 3: *Description of the Dataset Steel Defection (Grishin et al., 2019)*

	ImageId	ClassId	EncodedPixels
0	0002cc93b.jpg	1	29102 12 29346 24 29602 24 29858 24 301...
1	0007a71bf.jpg	3	18661 28 18863 82 19091 110 19347 110...
...	...	...	...

In order to save memory, the affected pixels of the respective image containing a defect were coded. The column 'EncodedPixels' consists of start positions and run length. Every second number (starting with the 1st) is a start position that belongs together with the number (run length) immediately following it. Example for using the EncodedPixels row: For example, '1 4 9 3' implies the pixels '$1, 2, 3, 4, 9, 10, 11$' which are included in the mask. (Grishin et al., 2019)