Høgskulen på Vestlandet

Bacheloroppgave

ELE350

Predefinert informasjon

Startdato:	08-05-2023 09:00 CEST	**Termin:**	2023 VÅR
Sluttdato:	22-05-2023 14:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	Bacheloroppgave		
Flowkode:	203 ELE350 1 O 2023 VÅR		
Intern sensor:	Endre Håland		

Deltaker

Navn:	Asbjørn Tjensvold
Kandidatnr.:	297
HVL-id:	141471@hvl.no

Informasjon fra deltaker

Egenerklæring *:	Ja
Inneholder besvarelsen konfidensielt materiale?:	Nei
Jeg bekrefter at jeg har registrert oppgavetittelen på norsk og engelsk i StudentWeb og vet at denne vil stå på vitnemålet mitt *:	Ja

Gruppe

Gruppenavn:	BO23EB-08
Gruppenummer:	8
Andre medlemmer i gruppen:	Stian Flåten, Eskil Digernes

Jeg godkjenner avtalen om publisering av bacheloroppgaven min *

Ja

Er bacheloroppgaven skrevet som del av et større forskningsprosjekt ved HVL? *

Nei

Vision System for Detection of Exposed Anode Surface

Exploring the Feasibility of Reducing CO_2 Emissions in the Aluminium Industry Using Computer Vision

Asbjørn Tjensvold, Eskil Digernes, Stian Flåten

Bachelor's thesis in Automation and Robotics

Submission date:	22nd of May, 2023
Supervisor:	Endre Håland, HVL
Co-supervisor:	Simen Vatslid Øystese, Hydro

Western Norway University of Applied Science (HVL)
Faculty of Engineering and Science

PREFACE

In this thesis, we explored Computer Vision (CV) and Machine-Learning (ML) despite not having studied them formally. Our foundation came from our robotics course book, "Robotics, Vision and Control," which guided our initial approach.

We express our gratitude to our co-supervisor at Hydro, Simen Vatslid Øystese, for his unwavering support, and Dr Morten Karlsen, who shared insights and urged simplicity with his mantra, "Why overcomplicate it?". We extend our gratitude to our supervisor at HVL, Endre Håland. His unwavering passion and enthusiasm have been contagious, and his guidance has been invaluable throughout this project—lastly, Dr Trond Eirik Jentoftsen, who introduced us to this subject via a simple email.

This thesis presents our development of a vision system for detecting exposed anode surfaces at an aluminium smelter. We hope our work contributes to the field and inspires future research.

Bergen, 22nd of May 2023

Asbjørn Tjensvold

Asbjørn Tjensvold

Eskil Digernes

Eskil Digernes

[illegible]

Stian Flåten

ABSTRACT

This thesis applies Computer Vision (CV) techniques to develop a Proof of Concept (POC) to detect exposed anode surfaces in aluminium smelters. Both basic colour, as well as edge detection algorithms and sophisticated Machine-Learning (ML) neural networks, were employed in developing this POC. A comparative analysis of the two methods revealed the neural network approach to be superior in accuracy, despite its more time-consuming development process.

The purpose of the POC is to target the adverse effects of aluminium smelting on the environment and economy. Aluminium production is among the most energy-intensive processes in Norway and accounts for 4 % of Norway's total greenhouse emissions [1]. The system is designed to identify exposed anode surfaces in the smelting process, which can significantly reduce CO_2 emissions and make aluminium production more profitable in a future with increasing CO_2 quota costs.

Furthermore, the results of this study demonstrate the feasibility and effectiveness of employing CV techniques for detecting exposed anode surfaces in aluminium smelters. The POC system developed in this research lays the groundwork for future advancements and exploration in this field, such as the automation of tasks.

SAMANDRAG

Denne oppgåva nyttar bildegjenkjenning, eller *Computer Vision*, for å utvikle eit konseptbevis i form av ein bildegjenkjenningsalgoritme, som skal oppdage eksponerte anodeoverflater i aluminiumssmelteverk. Både grunnleggjande farge- og kantgjenkjenning, samt nevrale nettverk vart nytta i utviklinga av dette konseptbeviset. Ein samanliknande analyse av dei to metodane viste at tilnærminga med nevralt nettverk hadde klart betre nøyaktighet, men ein meir tidkrevjande utviklingsprosess.

Føremålet med konseptbeviset er å takle dei negative konsekvensane aluminiumsproduksjon har på miljøet og økonomien. Aluminiumsproduksjon er blant dei mest energikrevjande prosessane i Noreg og står for 4 % av Noregs totale klimagassutslepp [1]. Systemet er laga for å identifisere eksponerte anodeoverflater i smelteprosessen, noko som kan redusere CO_2-utsleppa betydeleg, og gjere aluminiumsproduksjon meir lønsamt i ei framtid med aukande CO_2-kvotekostnader.

Resultata i denne oppgåva viser at det både er gjennomførbart og effektivt å bruke eit maskinlæringssystem for å oppdage eksponerte anodeoverflater i aluminiumssmelteverk. Konseptbeviset som er utvikla i denne oppgåva, legg grunnlaget for vidare framsteg og utforsking på dette området, til dømes automatisering av arbeidsoppgåver.

Contents

List of Figures

List of Tables

ACRONYMS

List of all acronyms in alphabetic order:

ACM	Anode Cover Material
ADC	Analogue-Digital Converter
AI	Artificial Intelligence
API	Application Programming Interface
CCD	Charge-Coupled Device
CFA	Colour Filter Array
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
FPS	Frames Per Second
GPU	Graphics Processing Unit
HSV	Hue-Saturation-Value
HVL	Western Norway University of Applied Science
IPCC	Intergovernmental Panel on Climate Change
LiDAR	Light Detection and Ranging
IoU	Intersection over Union
mAP	mean Average Precision
ML	Machine-Learning
NMS	Non-Maximum Suppression
POC	Proof of Concept
PR	Precision-Recall
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
ROC	Receiver Operating Characteristic
SOP	Standard Operating Procedure
TOS	Technology and Operational Support
YOLOv5	You Only Look Once version 5

CHAPTER

ONE

INTRODUCTION

1.1 Motivation

According to a statement by the Norwegian government [2], achieving carbon neutrality by 2050 is a global imperative. In response to this goal, Norsk Hydro, a leading Norwegian aluminium company, has initiated various strategies to curb carbon emissions [3]. One such strategy targets the reduction of CO_2 emissions stemming from insufficient anode covering in Hydro's aluminium-producing facilities, commonly referred to as *aluminium smelters*. These facilities produce aluminium via *electrolysis cells*, as seen in Figure 1.1.1.

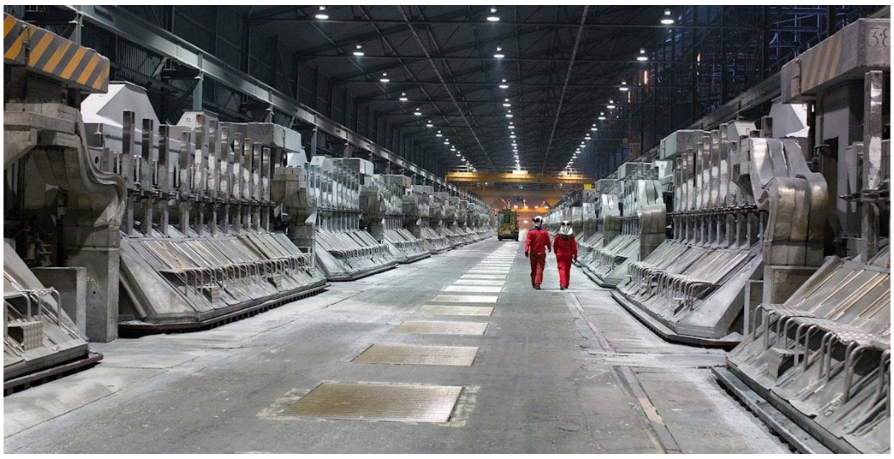


Figure 1.1.1: Rows of electrolysis cells in Øvre Årdal [4].

Currently, the anode covering operation is performed by operators, with little to no subsequent monitoring. To tackle this issue, Norsk Hydro has commissioned us to development of a Proof of Concept (POC) for an image recognition algorithm capable of identifying inadequately covered anodes that could provide feedback to the respective operators. While the research phase will involve a camera mounted on a tripod, the final objective will be to incorporate the camera onto the equipment used for the anode covering operation. However, the final object is not a part of this thesis but could be completed in a later project.

1.2 Consequences of Inadequate Covering

Inadequate anode covering will lead to exposed anode surfaces that eventually develop oxidation points, often called *hot spots*, as shown by the red circle in Figure 1.2.1. If left unaddressed, these hot spots contribute significantly to CO_2 emissions, accounting for more than 9 % of the total CO_2 generated in the electrolysis process at Norsk Hydro's smelter in Øvre Årdal.

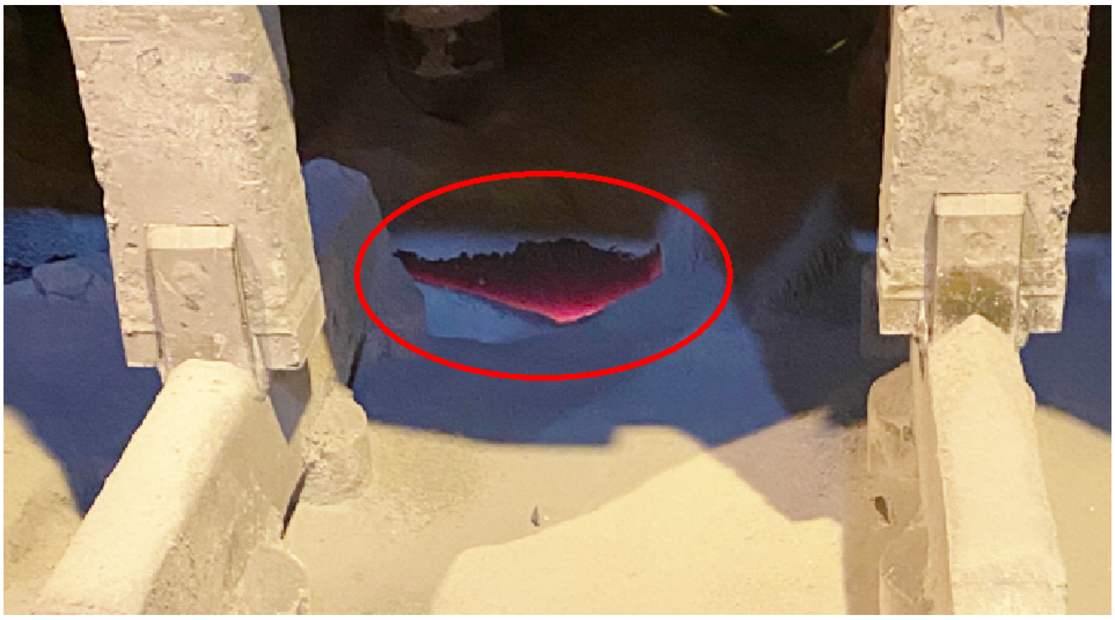


Figure 1.2.1: Inside an electrolysis cell: a hot spot on an anode.

Consequently, Hydro Årdal incurs substantial costs, exceeding **2 million NOK** per month, for CO_2 quotas related to anode burn [5]. However, the financial implications of this issue extend beyond quotas, as they do not include the loss of carbon anode mass [6].

Most importantly, CO_2 emissions have significant environmental consequences. The current climate crisis, driven predominantly by greenhouse gases such as CO_2, represents one of our society's greatest challenges. According to the Intergovernmental Panel on Climate Change (IPCC), urgent action must be taken to minimise its effects [7].

1.3 Project Description and Milestones

Aluminium smelters pose a challenging operational environment with high temperatures, magnetic fields, and corrosive dust and gases. Consequently, this thesis will involve assessing and optimising the camera's specifications and positioning to ensure optimal performance under these demanding conditions. Furthermore, we aim to develop a vision algorithm that meets three milestones outlined by Hydro Årdal:

1. Capable of detecting hot spots.
2. Expand the algorithm to identify uncovered areas that have not yet developed into hot spots.
3. Ultimately, enable the algorithm to determine the height of the Anode Cover Material (ACM).

The latter is not directly linked to anode burn, but the covering is also an important insolation layer and, thus, important for the energy efficiency of the cell. If the concept proves successful, it could yield substantial benefits and warrant further investment for continued development. This thesis is open to a range of interpretations, and the milestones set by Hydro were purposely structured to enable multiple approaches.

The project team will visit Hydro Årdal, where we will conduct multiple visits to gather images for algorithm development and take measurements of potential camera positions. In the next section, a brief overview of this thesis will be provided.

1.4 Thesis Overview

The main structure of this thesis is as follows:

Introduction: This section offers an introduction and overview of the project, along with a description of its motivating factors.

Background: This section provides necessary background information that enhances the comprehension of the project.

Methodology: This section details the project's procedures and the means of their application.

Results: This section presents and critiques the outcomes of the multiple solutions. It also includes an analysis of the images and videos used in the testing.

Discussion: This section provides a discussion of the results and offers possible solutions for future research and development.

Conclusions: This conclusive section encapsulates the project, emphasising the most effective solution and providing a summary of the work accomplished.

Table 1.4.1 briefly describes objects and industry-specific terminology associated with the aluminium sector that is central to the thesis. All of these objects are located within the electrolysis cell.

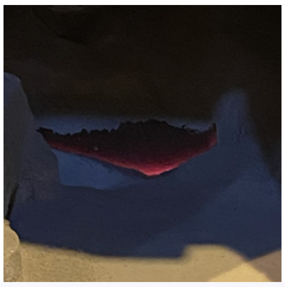
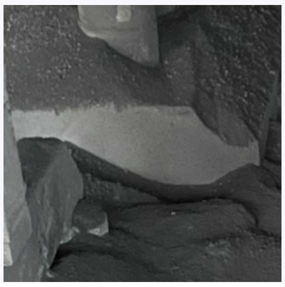
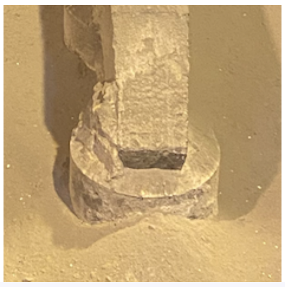
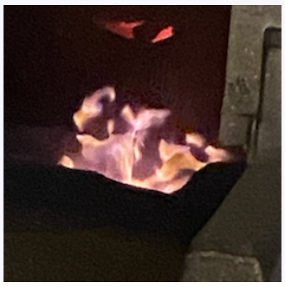
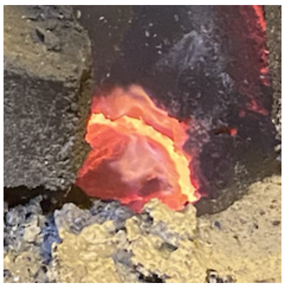
Object	**Image**	**Description**
Hot spot		Exposed anode surface that has developed into an oxidation point. The glowing red spot on the anode surface is often called *hot spot*.
Uncovered area		Exposed anode surface, or *Uncovered area*, that eventually will develop into a hot spot.
Stub		Cylindrical mounting on top of the anode.
Point flame		Flames that often spew through the cracks of the crust. The colour can vary from darker red (resembling the colour of a hot spot) to blue.
Tapping hole		Intentional holes in the anode cover material that are used to extract the aluminium from the cell. (The colour of a *tapping hole* can resemble the colour of a hot spot.)

Table 1.4.1: Overview of objects and industry-specific terminology central to the thesis.

Table 1.4.2 provides an overview of the different solutions implemented and evaluated in the thesis.

Solution	Purpose	Description	Reference	Result
Colour Detection (Solution One)	Detect hot spots	Hue-Saturation-Value colour detection (OpenCV)	Sec. 3.5.2, 4.1, 5.1, 6	Images: Inaccurate Video: N/A
Edge Detection (Solution One)	Detect uncovered areas	cv2.Canny method, edge detection (OpenCV)	Sec. 3.5.3, 4.1, 5.1, 6	Images: Inaccurate Video: N/A
Version 1.0 (Solution Two)	Detect hot spots and uncovered areas	Custom YOLOv5 model	Sec. 3.6.2, 4.2.1, 4.3, 5.1.1, 5.1.4, 6	Images: Accurate Video: Inaccurate
Version 2.0 (Solution Two)	Detect hot spots, uncovered areas, stubs, point flames, and tapping holes	Custom YOLOv5 model	Sec. 3.6.3, 4.2.2, 4.3, 5.1.2, 5.1.4, 6	Images: Accurate Video: Accurate
Version 2.1 (Solution Two)	Detect hot spots, uncovered areas, stubs, point flames, and tapping holes	Custom YOLOv5 model	Sec. 3.6.4, 4.2.3, 4.3, 5.1.3, 5.1.4, 6	Images: Accurate Video: Accurate

Table 1.4.2: Overview of the different solutions. Solution One's performance is not worth considering.

Table 1.4.3 provides a summary of the tests conducted in the thesis to evaluate the effectiveness of different solutions and factors. (The acronym *ML* in the table stands for *Machine-Learning*.)

Test	Purpose	Description	Reference	Result
Colour and Edge Detection	Assess its effectiveness	HSV colour detection, cv2.Canny method edge detection (OpenCV)	4.1.1	Inaccurate
Various ML-versions	Determine the most effective version	ML-versions V1.0, V2.0, and V2.1 tested on images and video	Sec. 4.3	V2.1: Most optimal V2.0: Comparable accuracy to V2.1 V1.0: Inaccurate
Various cameras	Determine which camera is optimal	Capturing images and video from iPhone 11 Pro and GoPro HERO11	Sec. 4.3, 5.1.5	iPhone 11 Pro marginally more optimal than GoPro HERO11
Distorted image	Moderate stress testing on the various solutions	Testing Solution One and ML-versions in Solution Two on distorted image	Sec. 4.1.1, 4.3.4	Solution One: Inaccurate V1.0: Accurate V2.0: Mostly accurate V2.1: Mostly accurate

Table 1.4.3: Overview of the tests carried out.

Table 1.4.4 compares the pros and cons of YOLOv5, Colour detection, and Edge detection.

Methods	**Description**	**Pros**	**Cons**	**Result**
YOLOv5	Real-time object detection algorithm using deep learning and convolutional neural networks.	- Real-time object detection - High accuracy	- Requires large annotated dataset - Higher computational requirements	Accurate detections in images and video
Colour Detection	Separates hue from value and saturation, enabling easier color-based operations like detection and segmentation.	- Simpler implementation - Lower computational requirements	- Less accurate - Sensitive to lighting conditions - Basic technology	Inaccurate in this instance
Edge Detection	Employs Canny technique for edge detection, analysing gradients and reducing noise for better accuracy.	- Simpler implementation - Lower computational requirements	- Less accurate - Sensitive to lighting conditions - Not very customisable - Basic technology	Inaccurate in this instance

Table 1.4.4: Comparison of YOLOv5, Colour Detection, and Edge Detection.

Table 1.4.5 provides an overview of which tools we have used and if they are premade or self-made.

Tools Used	**Description**	**Premade**	**Self-made**	**Reference**
YOLOv5 framework	Object detection algorithm that employs a single neural network for real-time object recognition and localisation in images.	✓		Sec. 3.6
YOLOv5 implementation for Solution Two	Custom -trained YOLOv5 models.		✓	Sec. 3.6, 4.2
Local implementation with images	Running the YOLOv5 models with images locally on a computer without an internet connection.		✓	Sec. 3.7.1
Local implementation with video	Running the YOLOv5 models with video locally on a computer without an internet connection.		✓	Sec. 3.7.2

Table 1.4.5: Overview of the tools used: premade or self-made.

CHAPTER

TWO

BACKGROUND

2.1 The Production of Primary Aluminium

To understand the relevance of this thesis, it is essential to understand the fundamentals of primary aluminium production. This subsection is therefore dedicated to this purpose.

2.1.1 The Electrolysis Cell

Primary aluminium is produced through the electrolysis of aluminium oxide (Al_2O_3), also known as alumina, in an electrolytic cell containing carbon electrodes: anodes and a cathode. A high electric current flows through the cell, driving the electrochemical reactions. The anodes, which are the positively charged electrodes, are submerged in an electrolyte solution, commonly referred to as *bath*, consisting of a mixture of alumina, cryolite (Na_3AlF_6), and other compounds. Cryolite serves to dissolve the alumina and decrease its melting point, while the additional compounds are employed to fine-tune the chemical properties of the electrolyte [8]. This thesis will focus on the simplified electrochemical reaction, as depicted in Equation 2.1.

$$2Al_2O_{3(\mathrm{s})} + 3C_{(\mathrm{s})} \rightarrow 4Al_{(\mathrm{l})} + 3CO_{2(\mathrm{g})} \tag{2.1}$$

This reaction is a reduction-oxidation (redox) process in which the oxygen present in the alumina reacts (oxidises) with the carbon in the anode, yielding carbon dioxide. At the same time, the liquid aluminium precipitates to the cathode (reduces) at the bottom of the cell.

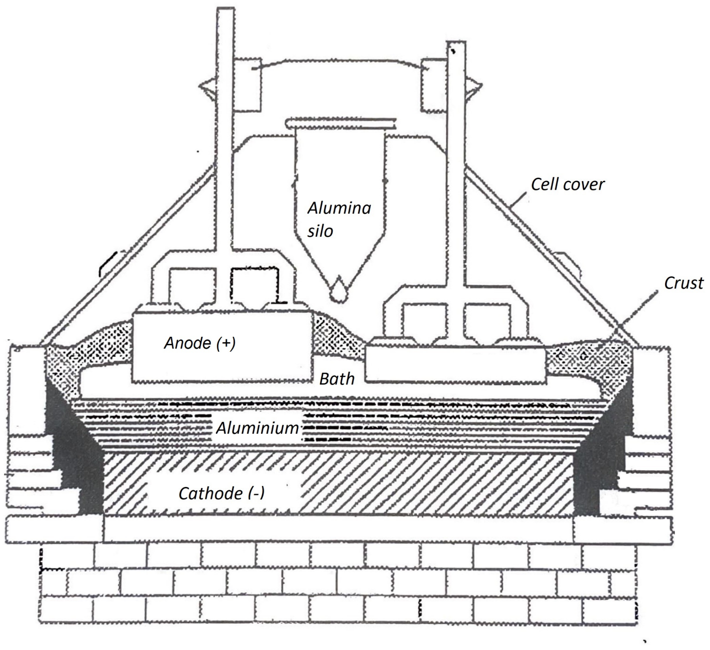


Figure 2.1.1: Cross section of an electrolysis cell [9].

The electrolysis process operates continuously, necessitating regular tapping of the molten aluminium and replacement of the anodes. The consumption rate of the anodes and the production of primary aluminium is proportional to the current strength and is described by Faraday's Law of Electrolysis as shown in Equation 2.2 [10].

$$m = \frac{Q}{F} \cdot \frac{M}{z} \tag{2.2}$$

Where:

- m is the mass of substance produced (aluminium) or consumed (carbon anode) in grams (g)
- Q is the electric charge passed through the electrolytic cell in coulombs (C)
- M is the molar mass of the material being deposited or dissolved (in grams per mole)
- F is Faraday's constant, representing the charge per mole of electrons and having a value of 96,485 C/mol for most applications.
- z represents the number of moles of electrons that are transferred during the reaction.

At Hydro Årdal, the anodes are typically replaced every 24 days. In Figure 2.1.2, a side view of a cell illustrates the consumption of anodes.

Anode one will be the next anode to be replaced, whereas anode eight is most recently changed. Used anodes are commonly referred to as *butts*.

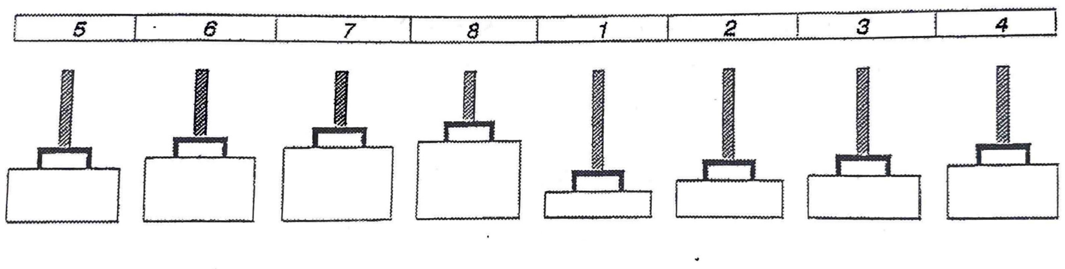


Figure 2.1.2: Illustration of the anode replacement pattern [9].

2.1.2 The Anode Covering Operation

After replacing an anode, the surfaces of the anodes are exposed, rendering them vulnerable to react with oxygen present in the ambient air rather than the oxygen in the alumina. Extended exposure to ambient air will cause the entire carbon mass of the anode to burn off, leading to decreased production and increased CO_2 emissions. Consequently, covering the anodes with Anode Cover Material (ACM) is crucial.

The ACM also serves as an insulator, enhancing the cell's energy efficiency, balancing the bath's chemical properties, and reducing fluoride gas emissions. The covering compound is a mixture of alumina and cryolite, and its composition is adjusted based on the chemical properties of the bath [11]. Anode covering should start approximately four hours after the anode replacement.

The ACM is ejected through a nozzle, as shown in Figure 2.1.3, either by a crane or a vehicle. A vehicle is utilised for the covering operation at Hydro Årdal, where this project is conducted.

In addition to covering the newly changed anodes, the covering operator is also responsible for applying additional ACM where needed, i.e. hot spots and other uncovered anode surfaces, which is one of the proposed applications of the algorithm developed in the thesis.

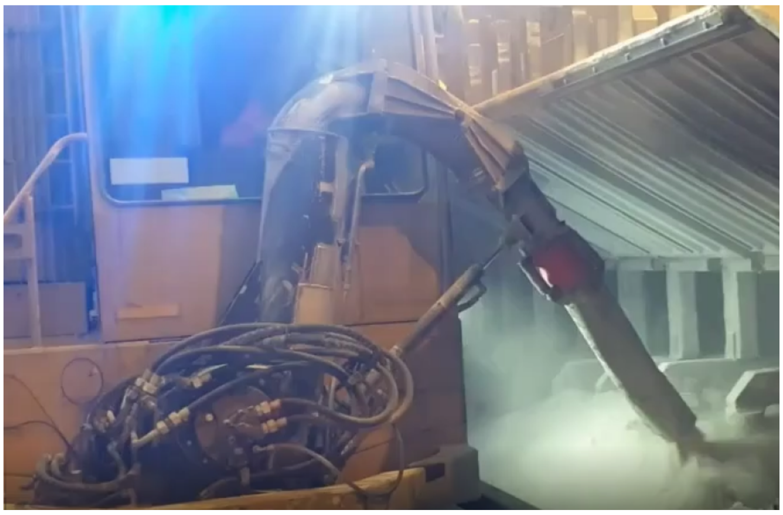


Figure 2.1.3: The covering vehicle [11].

2.1.3 The Cause of Hot Spots

An uncovered anode surface becomes a hot spot because of the tremendous electric power that passes through the anode. In Hydro Årdal's case, the cells draw a total current of 235,000 A with a voltage set to roughly 4.4 V if we use the equation for electric power 2.3.

$$P(t) = V(t) \cdot I(t) \tag{2.3}$$

Where:

- P(t) is the power consumed, measured in Watts (joules per second).
- V(t) is the electric energy potential of the cell, measured in Volts.
- I(t) is the current going through the cell, measured in Amperes.

This results in a total power consumption of over 1,000 kW per cell. If anodes are exposed to the ambient air, power consumption of this magnitude will burn the anode—emitting a distinct electromagnetic wavelength of around 650 nm, which our eyes interpret as red (depicted in Figure 2.1.4).

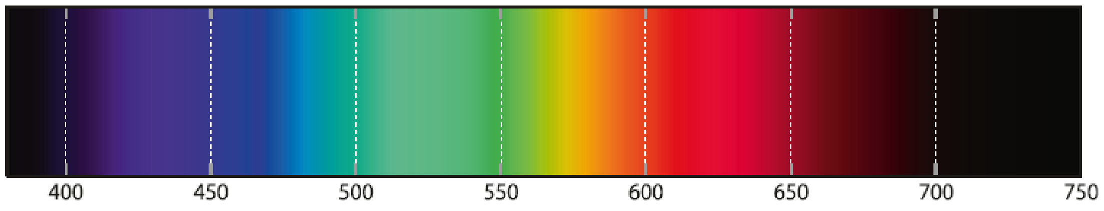


Figure 2.1.4: The visible wavelength spectrum in nanometres [12].

Using this information, we can use an affordable consumer camera to detect hot spots and possibly other uncovered anode surfaces.

2.1.4 Hydro Årdal's CO_2 Roadmap

As demonstrated by the stoichiometry of Equation 2.1, a certain amount of CO_2 will be generated during aluminium production. The theoretical minimum CO_2 emission is calculated using the three-over-four molar ratio between CO_2 and aluminium, then converting molar mass into mass. This means that for every kilogram of aluminium produced, approximately 1.23 kg of CO_2 (kg CO_2/kg Al) is generated, which is the unit used in Figure 2.1.5. However, as Figure 2.1.5 illustrates, Hydro Årdal currently emits 1.67 kg CO_2 per kilogram of aluminium and aims to reduce this to 1.55 kg by 2025. Figure 2.1.5 also depicts the composition of CO_2 emissions. However, this thesis will exclusively focus on the hot spot aspect of this objective, which Figure 2.1.5 depicts as *Airburn*.

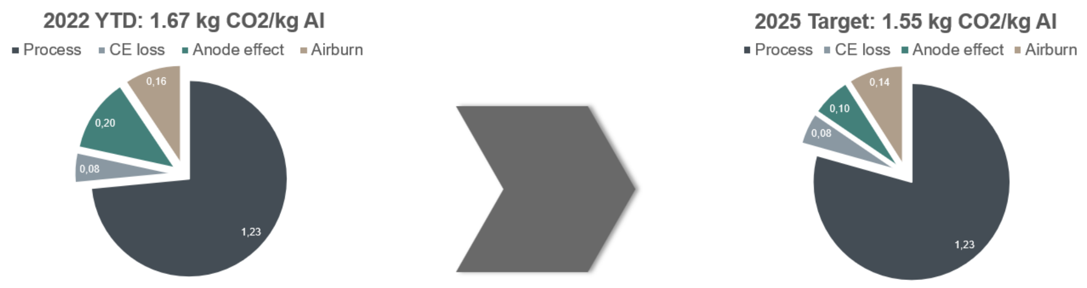


Figure 2.1.5: CO_2 target for 2025 [5].

As seen for the target for 2025, Hydro Årdal aims to reduce the hot spot aspect from 0.16 kg CO_2/kg Al to 0.14 kg CO_2/kg Al. With the current cell technology, it is challenging to lower the CO_2 past 0.14 kg CO_2/kg Al because aluminium production necessitates frequent tapping of aluminium and feeding of alumina (Al_2O_3) as depicted in Figure 2.1.6a and Figure 2.1.6b, respectively.

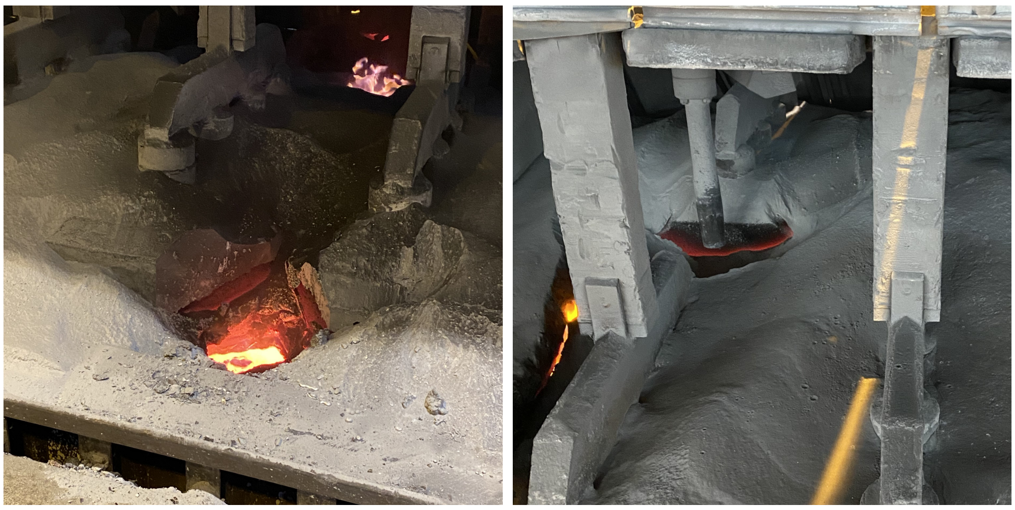

(a) Tapping hole. **(b)** Point feeder.

Figure 2.1.6: Hot spots around tapping hole and feeding point.

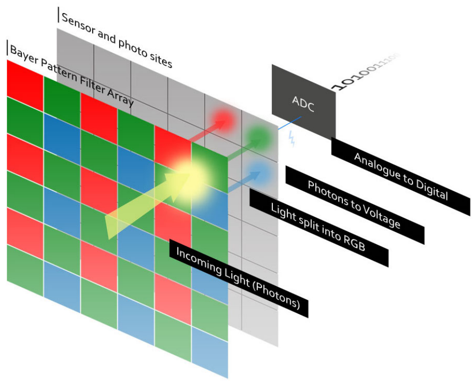
This leads to an open tapping hole and feeding points, causing some anode mass to be exposed and burned off. However, other areas have significant potential for improvement by enhancing anode covering and ensuring that unintentional anode surfaces are covered appropriately. A reduction of 20 kg of CO_2 per tonnes aluminium (0.16 to 0.14) may sound small but could, according to our calculations, yield an annual saving of approximately 3,600,000 NOK, solely from CO_2 quotas. However, the price of carbon anode mass is approximately 10,000 NOK [13], yielding an annual saving of almost 11,000,000 NOK for the 2025 goal. Even a tiny reduction, such as 3 kg of CO_2 per tonnes aluminium, will save over 1,600,000 NOK. For complete calculation, see Appendix D.

2.2 Tool Background

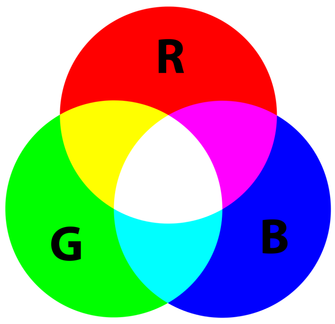
In addition to understanding the production of primary aluminium, it is also paramount to understand the theory of the tools we used during this thesis, such as image sensors and Machine-Learning (ML) concepts.

2.2.1 Image Sensor and Bit Resolution

An image sensor comprises millions of light-sensitive pixels, each acting as a tiny photosensitive diode. When light falls on a pixel, it generates a voltage charge proportional to the light intensity. To capture colour, a Colour Filter Array (CFA) is placed in front of the sensor, as depicted in Figure 2.2.1a [14]. A CFA consists of blue, red, and green squares, allowing specific wavelengths of light to pass through and forming an RGB colour profile, the most common method for capturing the visible light spectrum (Figure 2.2.1b).



(a) CFA, sensor and an ADC [15].



(b) The RGB Profile [16].

Figure 2.2.1: Illustatition of image sensor end the RGB colour profile.

The camera's processor utilises an Analogue-Digital Converter (ADC), representing the RGB colours with a given bit resolution. Bit resolution refers to the number of bits used to represent each pixel in an image. Higher bit resolutions enable more colours and shades, resulting in higher quality and more detailed images. For instance, 8-bit resolution allows $2^8(256)$ different values, while 16-bit resolution allows $2^{16}(65,536)$ values, providing smoother gradations and a wider range of RGB colours. However, higher bit resolutions demand more processing power. In our case, an 8-bit resolution should be sufficient.

2.2.2 Introduction to Machine-Learning

Machine-Learning (ML) is a branch of Artificial Intelligence (AI) to make computers learn and draw predictions using data without explicitly being programmed for the task [17]. In simple terms, ML algorithms use statistics to find patterns within the data. Then it applies these patterns to make predictions in new situations.

2.2.2.1 Datasets and Training-Test Split

An essential part of ML is the *dataset*, a collection of data points used to train the algorithm. Datasets can vary in size and complexity depending on the problem at hand. In most cases, larger datasets lead to better performance, as they provide more information for the algorithm to learn from [18].

The dataset is divided into two parts to effectively train an ML model: the *training set* and the *test set*. The training set instructs the algorithm in making predictions, whereas the test set assesses the model's performance on previously unencountered data. Typically, the dataset is divided into 80% for training and 20% for testing. However, this ratio might be adjusted based on the problem and the dataset's size [19].

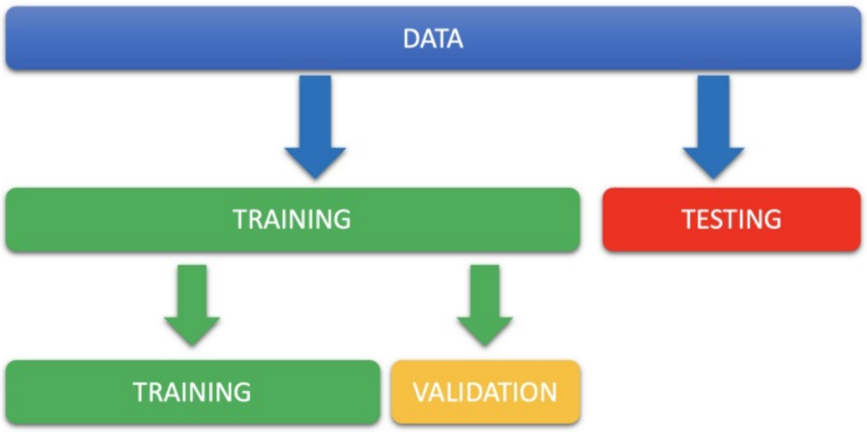


Figure 2.2.2: Illustration of data set division [20].

2.2.2.2 Preventing Overfitting in Machine Learning Models

At its core, ML relies on statistical methods to discern patterns and associations within the data. Nevertheless, it is crucial to prevent *overfitting*. This phenomenon arises when a model excels at performing on the training set but struggles to generalise previously unseen data [21]. Overfitting can result in subpar performance on the test set and may suggest that the model has learned the data's noise instead of the true underlying relationships.

Various techniques can be applied to mitigate overfitting, such as regularisation, early stopping, using a validation set (as seen in Figure 2.2.2), or fine-tuning the model's hyperparameters [22]. The number of *epochs*, which refers to the number of times the entire training set is passed through the learning algorithm, can also impact overfitting. Too few epochs may result in underfitting, while too many epochs can lead to overfitting. Hence, selecting the appropriate number of epochs is crucial for achieving optimal performance [23].

2.2.2.3 Parameters and Hyperparameters

In ML, two distinct types of adjustments can be made to the learning process: *hyperparameters* and *parameters*.

Hyperparameters are the parameters that govern the learning process. Examples include the number of epochs, the train-test split ratio, and the batch size. In simple terms, all the layers in Figure 2.2.3 are hyperparameters, except for the content inside the *Fully Connected layer*. These are known as parameters and are adjusted purely from the data as the algorithm maps features between the input and output of the neurons. Examples of parameters are the neurons' weights, coefficients, and biases. All the parameters are trainable but become frozen once the model is trained [24].

2.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning algorithms that have succeeded in image recognition and classification tasks [25]. In this application, we use a CNN to detect hot spots, uncovered areas, and other features within the cell. The architecture of the CNN used in this project consists of several layers, each responsible for a specific task, as shown in Figure 2.2.3 and are as follows:

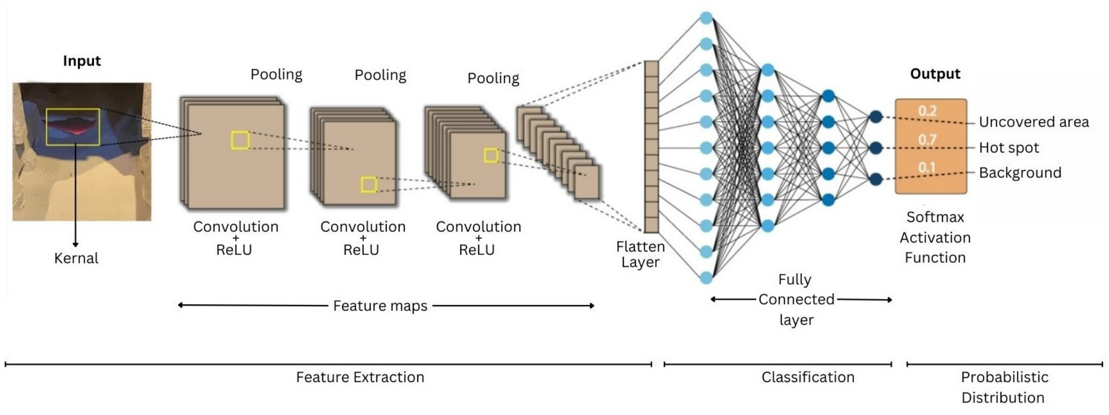


Figure 2.2.3: Architecture of our first CNN model V1.0.

Input Image: As illustrated in Figure 2.2.4, the input image I is supplied to the network, and its matrix elements correspond to pixel values. Higher values denote brighter pixels, whilst lower values signify darker ones. Subsequently, a kernel K (the yellow rectangle visible in both Figures 2.2.3 and 2.2.4) is applied. This small matrix, known as the kernel, detects features by convolving (sliding) over the input image and carrying out element-wise multiplication, followed by summing the resulting values. Mathematically, convolution is described in Equation 2.4 [26].

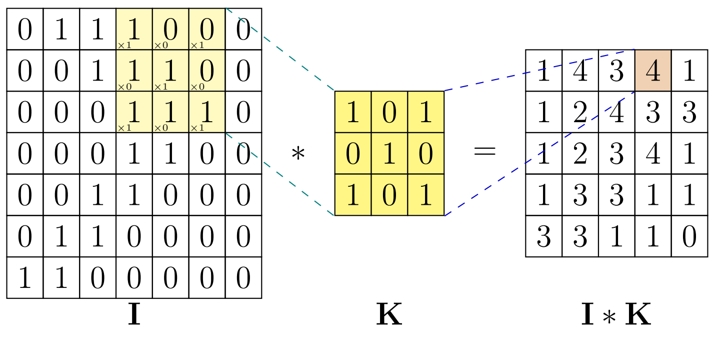


Figure 2.2.4: Convolution between input image **I** and a kernel **K** [27].

$$\text{out}(N_i, C_{\text{out}j}) = \text{bias}(C_{\text{out}j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}j}, k) * \text{input}(N_i, k) \tag{2.4}$$

Where:

- N_i: The i-th input image in a batch.
- C_{in}: The number of input channels.

- $C_{\text{out}j}$: The j-th output channel.
- weight$(C_{\text{out}j}, k)$: The weight connecting the k-th input channel and the j-th output channel.
- input(N_i, k): The value of the k-th input channel of the i-th input image.
- bias$(C_{\text{out}j})$: The bias term associated with the j-th output channel.
- out$(N_i, C_{\text{out}j})$: The value of the j-th output channel for the i-th input image after the convolution operation.

Feature Maps: This stage comprises multiple layers, including convolutional, pooling, and Rectified Linear Unit (ReLU) layers. The convolutional layers apply multiple kernels to the previous layer's output, producing feature maps representing different aspects of the image. The network refines these filters by adjusting their values during the training process, enabling the extraction of meaningful features from the image. The pooling layers help reduce the spatial dimensions of the feature maps, making the network more robust and computationally efficient. ReLU layers introduce non-linearity into the network, allowing it to learn complex patterns [28].

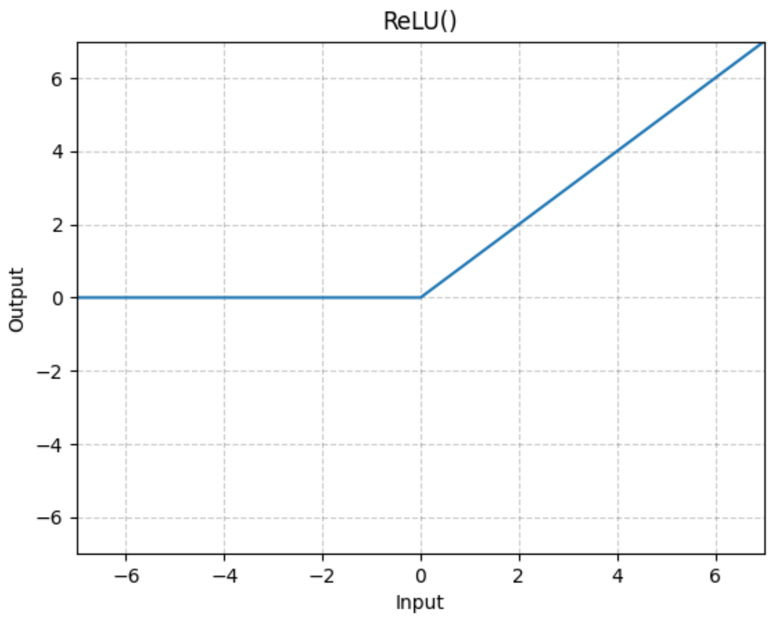


Figure 2.2.5: Graph of the ReLU function [29].

In simple terms, the ReLU sets all negative values to zero while retaining the positive values as illustrated in Figure 2.2.5. Mathematically the function is described in Equation 2.5.

$$\hat{x} = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2.5}$$

Flatten Layer: After passing through the feature extraction layers, the output feature maps are flattened into a one-dimensional vector. This vector contains the high-level features extracted from the input image and is used as input for the subsequent classification stage.

Fully Connected Layer: Following the flattening layer, the goal is to classify the input image according to the features identified earlier. The fully connected layer includes neurons that are interconnected with every component in the flattened feature vector, as demonstrated in Figure 2.2.3. This layer amalgamates the high-level features that have been extracted previously and learn to recognise patterns associated with the target classes [30].
This layer comprises a significant portion of the model's parameters, determined by the total number of neurons in the fully connected layer, the size of the flattened feature vector, and the biases. In our model, there are a total of 7,025,023 parameters, with the majority being weights of the connections between neurons. The output generated by a fully connected layer is formulated as shown in Equation 2.6:

$$y = W\hat{x} + b \tag{2.6}$$

In this equation, W denotes the weight matrix, x embodies the input vector, and b signifies the bias vector. The weight matrix W has dimensions that allow it to transform the input feature vector x into the output y, while the bias vector b is added to shift the result [31]. These weights and biases are learned during the training process, allowing the network to make accurate predictions.

Output with Softmax Activation Function: The final layer of the CNN generates an output vector with a length that matches the number of target classes. The softmax activation function is applied to this vector, transforming it into a probability distribution over the target classes. The predicted class for the input image is determined by selecting the class with the highest probability [22]. The Softmax function is defined as seen in Equation 2.7.

$$f_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.7}$$

Where x_i is the input value for the i-th class and $f_i(x)$ is the probability of the i-th class. The denominator normalises the output so that the sum of probabilities for all classes equals 1 [32].

In summary, the CNN architecture used in this project is designed to detect and classify features within the cell effectively. The feature extraction stage identifies high-level patterns in the input image, while the classification stage uses these patterns to determine the most probable class for the input image.

CHAPTER

THREE

METHODOLOGY

3.1 Challenge Analysis

One technique for detecting anode surfaces involves utilising colour recognition and edge detection, which employs an image sensor to identify specific attributes of the anode surfaces. For hot spots, detection is comparatively simple, as they emit a unique wavelength of light, as previously noted. However, identifying exposed anode areas may be considerably more difficult due to their dark colour, which is nearly indistinguishable from the ACM. Nevertheless, exposed regions frequently exhibit sharp, sudden edges where an edge-detection algorithm might be applicable.

While colour and edge detection are relatively simple, they come with challenges. For instance, the anode surface colour can vary depending on various factors, most notably the ambient lighting conditions at the smelter. Shadows can create hard edges and other colours and features inside the cell, potentially reducing the accuracy of this approach.

An alternative method for detecting anode surfaces involves using ML techniques. These algorithms can be trained on comprehensive datasets of anode surface images, enabling them to identify distinguishing features between hot spots and uncovered areas. Such an approach could offer greater robustness to variations in colour and lighting conditions and can be tailored to recognise anode surfaces across a wide range of environments.

Developing a reliable ML model for anode surface detection may necessitate considerable time and resources. This process includes gathering a large and diverse dataset of images, selecting and fine-tuning appropriate ML algorithms, and training and validating the model to ensure its accuracy and dependability.

3.2 Tool Consideration

One of the most significant challenges in this project is selecting a camera that can withstand the harsh conditions within the smelter, where temperatures during the summer months can exceed 70 °C in certain areas, in addition to the radiation heat from the cells. It is worth noting that operators work in these conditions, utilising appropriate protective equipment to ensure their safety. Additionally, the camera must resist corrosive gases and dust, which erode vehicle windows and reduce their transparency. The strong magnetic fields generated by the substantial current passing through the cell can also impair the camera's autofocus capability, as seen in Figure 3.2.1.

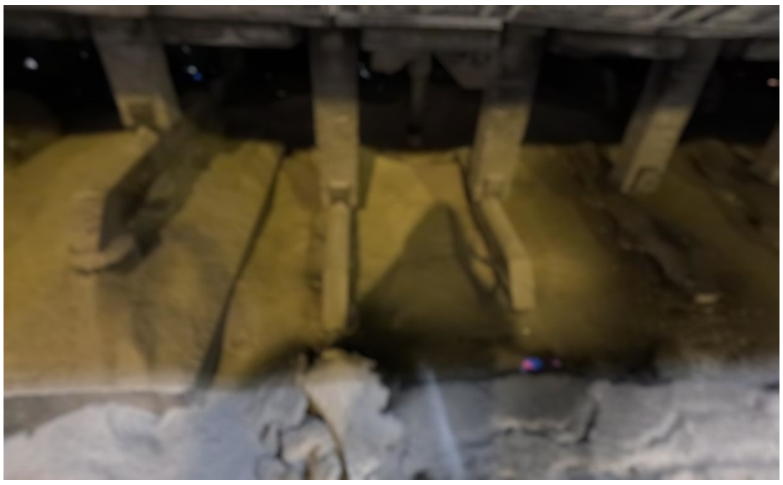


Figure 3.2.1: Out-of-focused cell image due to strong magnetic fields.

Considering these factors, we must select an affordable and durable camera system with minimal moving parts to ensure reliability and longevity in the demanding environment. In addition, there are also two primary types of image sensors used by cameras, Complementary Metal-Oxide-Semiconductor (CMOS) and Charge-Coupled Device (CCD).

For the demanding environment of an aluminium smelter, CMOS sensors are a better choice than CCD sensors. They are less susceptible to magnetic fields and consume less power [33]. The GoPro HERO11 and iPhone 11 Pro, which both use CMOS sensors, are therefore considered suitable options.

3.2.1 Hardware Selection

After concluding that the CMOS sensor is the preferred choice, we have selected the GoPro HERO11 as our primary camera, as it has proven to withstand the harsh conditions in Hydro Årdal [34]. Furthermore, its minimal moving parts render it less susceptible to interference from magnetic fields. Alongside the GoPro, we will employ an iPhone 11 Pro for testing. We hypothesise that the GoPro's 16.5 mm focal length might be too wide for our specific application. In contrast, the iPhone 11 Pro features three cameras, each with the following focal lengths:

- ultra-wide (13mm)
- wide (26mm)
- telephoto (52mm)

This choice of focal lengths allows us to experiment with various perspectives. Moreover, implementing a smartphone-based system can offer a portable solution for operators by utilising a device many people already own.

3.2.2 Software Selection

Python was chosen for our CV system for its numerous advantages. It offers various image processing and ML libraries like *OpenCV* and *PyTorch*, catering specifically to CV applications. These libraries simplify and accelerate the development process.

As a free and open-source language, Python is cost-effective and accessible. Its user-friendly nature allows for rapid prototyping and development, and its cross-platform compatibility ensures seamless execution across various operating systems, such as Windows, macOS, and Linux.

Python's popularity guarantees access to many resources and extensive community support, making it suitable for developing our CV system.

3.3 Camera Positioning

To capture a comprehensive view of the electrolysis cell, we placed the GoPro camera on a tripod 50 cm away from the cell at the height of 170 cm and angled it at -25° from the horizontal plane. This particular positioning was crucial to mitigate the impact of strong magnetic fields on the camera's autofocus capability, protect the camera from the high temperatures within the cell, and ensure effective observation of vital aspects of the cell, such as detecting hot spots, uncovered areas, and the height of ACM. The chosen setup was optimal for our specific requirements and constraints.

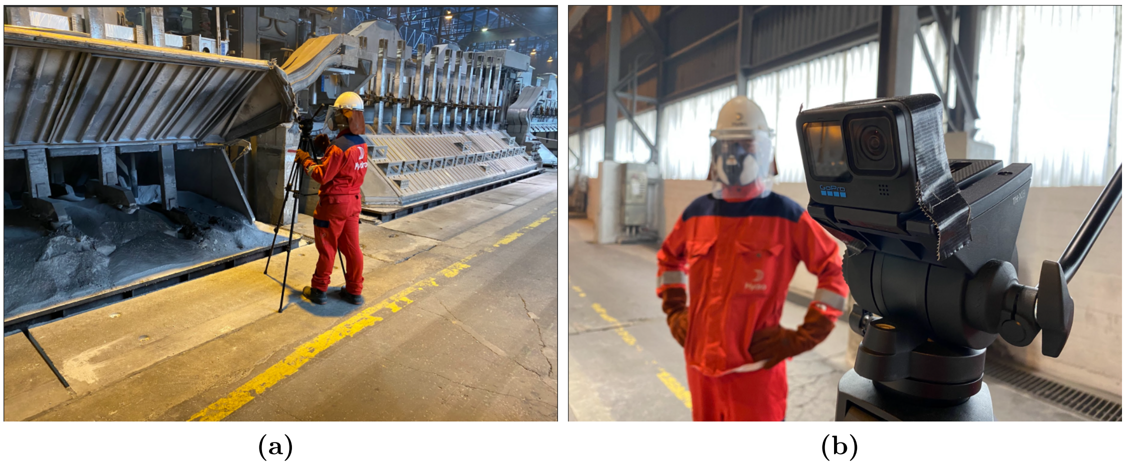

(a) (b)

Figure 3.3.1: Pictures of us gathering data for our training set.

Moreover, we took handheld iPhone photos simultaneously to capture alternative angles of the cell. After conducting multiple trials, we concluded that the tripod position was optimal for the GoPro camera, allowing for easy mounting on a vehicle or crane used in the covering operation. Meanwhile, the handheld iPhone offered the flexibility to capture additional footage from other perspectives, further emphasising the importance of considering different camera setups.

3.4 Determining Height of ACM

We hypothesise that the two first milestones can be done without any form of specialised camera. However, the same cannot be said for our third and final milestone, determining the height of the ACM. A range of systems was considered, including Light Detection and Ranging (LiDAR), which utilises laser light to generate a three-dimensional model of the cell. However, this approach might fall outside the scope of this thesis. Our co-supervisor suggested an alternative method: using the stubs as reference points. This idea presents a feasible solution worth considering.

The stubs have a known height of 19 cm, as shown in Figure 3.4.1, and according to the Standard Operating Procedure (SOP) for the covering procedure [11], the ACM should have a height of 10 to 12 cm, which means approximately 8 cm of the stub should be visible. However, the challenge in measuring this with a single camera, like our GoPro, lies in its lack of depth perception. The camera should maintain a consistent distance from the stub to measure the height accurately.

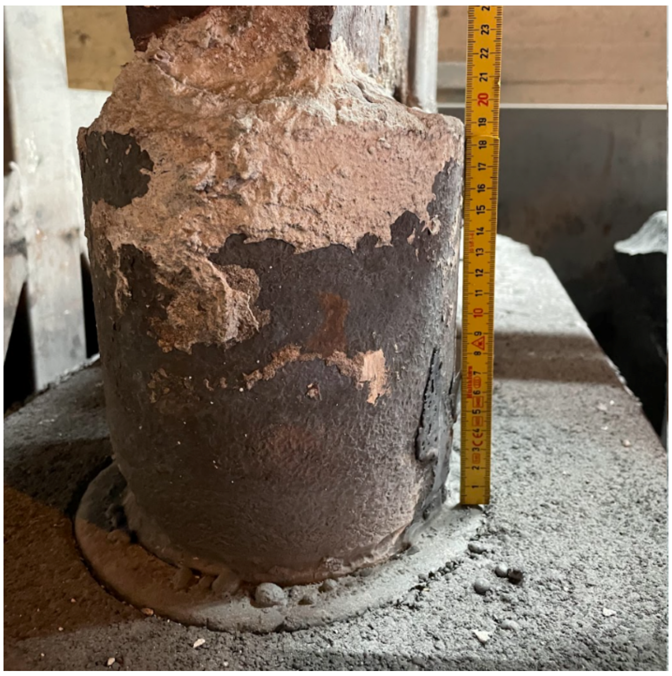


Figure 3.4.1: A ruler depicting the height of the stub.

We measured the distance between the first stub and the camera to be roughly 140 cm. Nevertheless, during the process of covering anodes at Hydro Årdal using a vehicle, it is difficult to maintain a constant distance, as the covering vehicle occasionally has to manoeuvre according to its constraints, such as the length of its nozzle and the varying degree to which the cell cover can open.

The height of cell covers does indeed fluctuate from our observations. Consequently, we hypothesised that a stereo camera would be better suited.

A stereo camera consists of two image sensors, akin to how humans possess a pair of eyes. This pair of sensors allows for depth perception via triangulation [35]. There are readily available stereo cameras, such as the *Intel® RealSense™ D435*. Alternatively, one can utilise two separate cameras, like GoPros, and calibrate their angles according to the desired specifications. Nonetheless, this approach might be more time-consuming and expensive than purchasing a pre-built 3D camera.

3.5 Solution One: Colour and Edge Detection

As previously mentioned, hot spots emit a wavelength of 650 nm, which the human eye perceives as red. This allows us to employ a simple colour-detecting algorithm to identify hot spots. Furthermore, the uncovered areas have distinct edges, which may be recognised using basic edge detection.

3.5.1 Analysing Colours with Histograms

In an RGB image, a histogram separately represents the distribution of intensities for each colour channel (i.e. Red-Green-Blue). Where:

- X-axis represents the intensity values or brightness (8-bit, 0-255).
- Y-axis is the number of pixels with that intensity value in the image.

By analysing the histograms of an image, we can obtain information about the image's brightness, contrast, and, most importantly, colour distribution. In Figure 3.5.1, a histogram is plotted next to a well-covered cell, and in Figure 3.5.2, a hot spot is present.

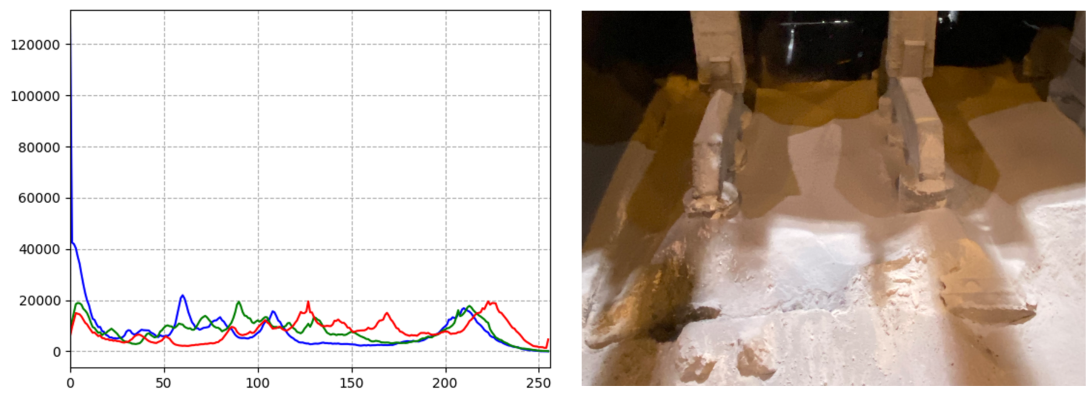


Figure 3.5.1: Histogram of well-covered anodes.

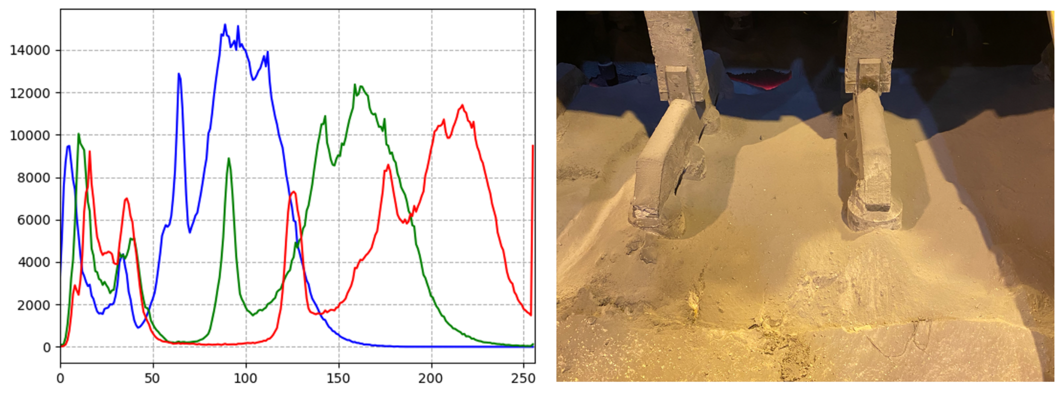


Figure 3.5.2: Histogram of anodes with a hot spot present.

As the two figures demonstrate, Figure 3.5.2 displays a notable red spike beyond $x = 250$, signifying a hot spot. However, the Hue-Saturation-Value (HSV) model is frequently favoured for colour detection, as it segregates colour information based on human colour perception. In comparison, RGB is a colour model founded on the primary colours of light, which are additive [36].

The other spikes observed in Figure 3.5.2 can be attributed to the additive nature of the RGB profile. This profile combines the RGB colours to produce the visible light spectrum, as illustrated in Figure 2.2.1b. These spikes arise because the individual RGB channels contribute to the overall colour representation. When the values of the RGB channels vary $0 \leq x \leq 200$ as seen in Figure 3.5.2, they form different shades and combinations of colours, resulting in the observed spikes in the histogram [37]. This phenomenon can be explained by considering that the RGB model is sensitive to lighting changes and other environmental factors, which can lead to variations in the colour distribution [38].

3.5.2 Detecting Hot Spots with Hue-Saturation-Value

While RGB is useful for displaying colours on screens or monitors, it is less effective in colour-based image processing tasks such as detecting hot spots. HSV, on the other hand, separates colour information into a cylindrical model as depicted in Figure 3.5.3 with three components:

- The hue is defined as an angle where 0° represents red, 120° represents green, and 240° represents blue
- Saturation represents the purity of the colour, given by the length of the vector.
- The value represents the brightness or intensity of the colour.

This separation of colour information in HSV makes it easier to isolate and segment-specific colours in an image, making it a better choice for detecting hot spots [39].

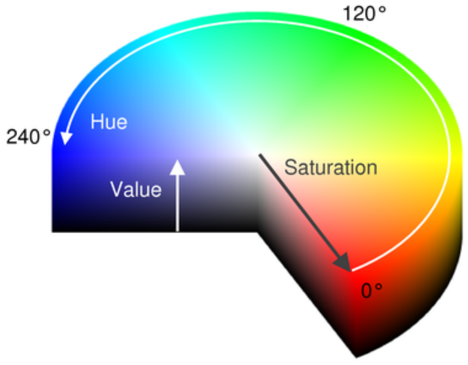
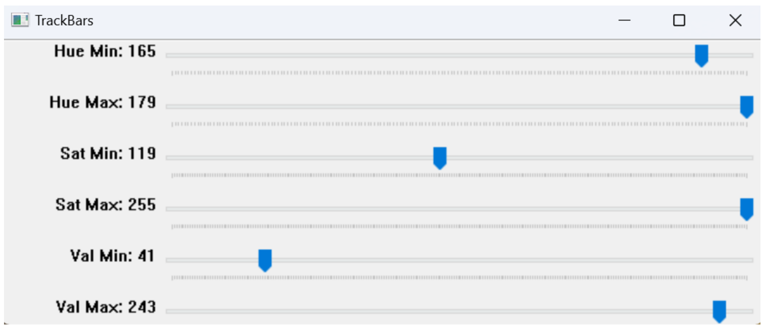


Figure 3.5.3: HSV diagram indicates the Hue-Saturation-Value values, respectively [40].

To convert from RGB to HSV, we have to establish the minimum and maximum values of Red-Green-Blue in the RGB model, which ranges from 0 to 255 in 8-bit. We accomplished this task by using OpenCV's `cv2.COLOR_BGR2HSV` function to transform the image from RGB to HSV profile. To help us generate matrices and masks, we also utilised the *NumPy* library.

Inspired by Murtaza Hassan's online lecture [41], we added trackbars. These trackbars aided us in adjusting the bit range for each channel in the HSV colour space based on each channel's minimum and maximum values. By fine-tuning the mask, we could display only the desired colours. Figure 3.5.4 illustrates the resulting trackbar values.



TrackBars

Hue Min: 165
Hue Max: 179
Sat Min: 119
Sat Max: 255
Val Min: 41
Val Max: 243

Figure 3.5.4: HSV values used to mask hot spots.

Furthermore, we developed a function that adds green rectangles around the hot spots to increase their visibility. The output depicted in Figure 3.5.5 effectively demonstrates the successful use of trackbars.

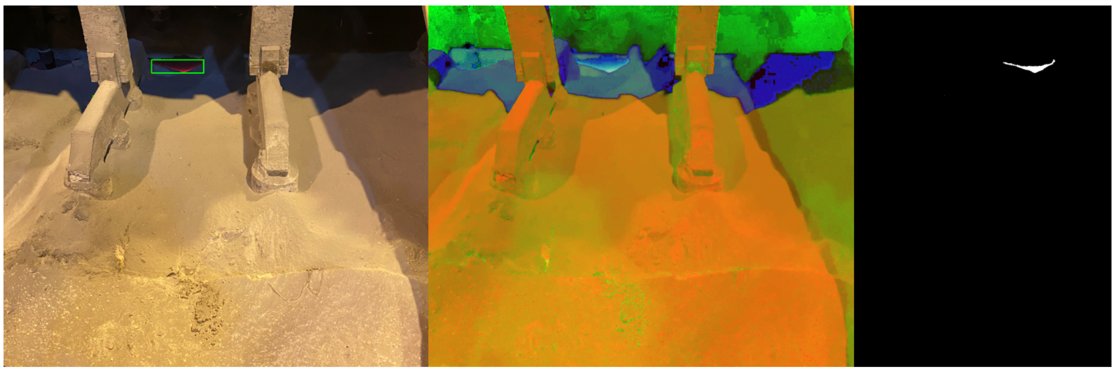


Figure 3.5.5: Left to right: RGB with hot spot detector, HSV image, and NumPy mask.

3.5.3 Detecting Uncovered Areas with Edge Detection

Since uncovered anode areas usually have noticeable corners, we hypothesised that simple edge detection could be used. Luckily, OpenCV also has built-in edge detection algorithms like the `cv2.Canny` method. However, it has a few drawbacks making it unsuitable for our task, such as:

- **Vulnerability to noise:** Resulting from other features within the cell.
- **Difficulty with tuning:** We had to use different values when tuning for each image as they did not have the same lighting

After implementing and testing it on a small data set of images, as seen in Figure 3.5.6, it became clear that we had to develop a more accurate method of detecting uncovered areas, possibly involving ML methods.

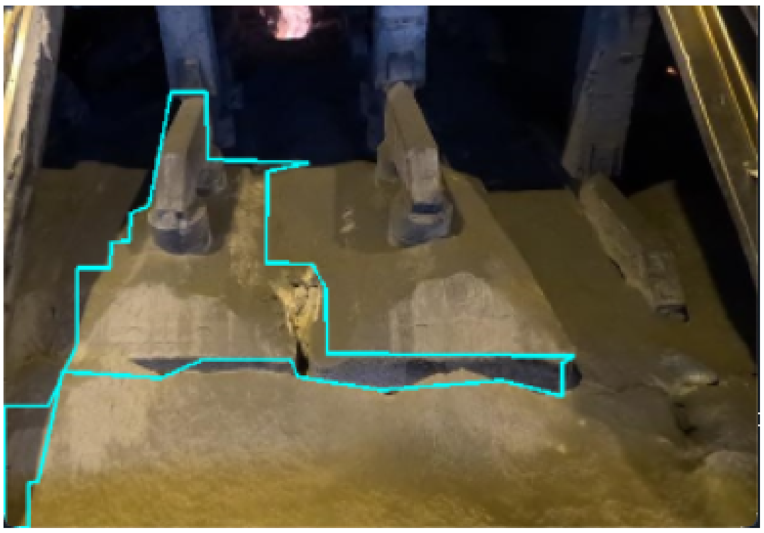


Figure 3.5.6: Edge detection results highlighting the limitations in accurately identifying uncovered areas.

3.6 Solution Two: Convolutional Neural Network

A potentially more precise solution for our problem could be found within the field of ML, specifically Convolutional Neural Networks (CNN). We selected a particular implementation of CNN called You Only Look Once version 5 (YOLOv5). It is an open-source, Python-based framework available on GitHub. YOLOv5 has been proven capable of fast and accurate detection of objects within images, making it an ideal choice for detecting hot spots and uncovered areas [42, 43].

3.6.1 Training YOLOv5 Step-by-Step

The method of training our custom YOLOv5 is illustrated in Figure 3.6.1 and consists of three main stages: setting up the necessary software, preparing the data (a collection of images), and running the training process. All our models generally followed this form with only slight adjustments further discussed in their sections, respectively.

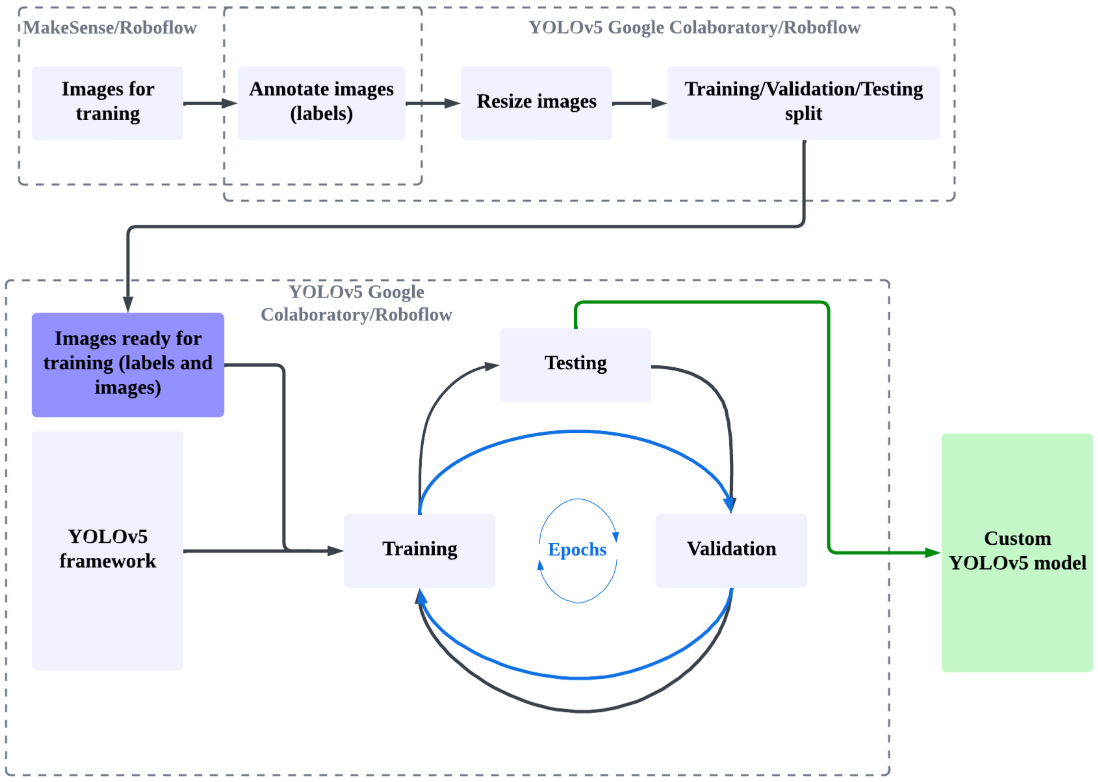

MakeSense/Roboflow
YOLOv5 Google Colaboratory/Roboflow
Images for traning
Annotate images (labels)
Resize images
Training/Validation/Testing split
YOLOv5 Google Colaboratory/Roboflow
Images ready for training (labels and images)
YOLOv5 framework
Testing
Training
Epochs
Validation
Custom YOLOv5 model

Figure 3.6.1: Simplified flow diagram of how we trained our YOLOv5 model.

3.6.1.1 Step I: Setting up the Software

We started by cloning the YOLOv5 repository from GitHub and installing the necessary additional tools. We used `pip install` to include the Roboflow Python library to manage our data. Additionally, we imported the PyTorch library to enable the heavy computational tasks of ML.

3.6.1.2 Step II: Preparing the Data

This subsequent step is dedicated to the preparation of our data for training. In our case, we annotated our collection of anode images by delineating *bounding boxes* around the intended objects of interest, such as hot spots and uncovered areas. This data was stored in a `.txt` file detailing classes alongside the corresponding *x, y coordinates* of each bounding box for the respective class.

This is the most time-consuming step, which can not be automated in the beginning. For example, model 2.1 has 1358 annotations and took an entire day of annotating bounding boxes.

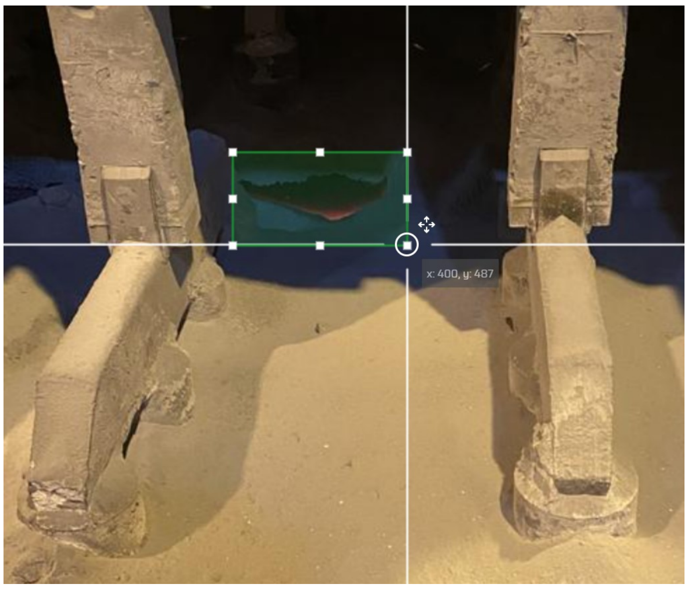


Figure 3.6.2: Sample of object annotation on a hot spot.

In deciding how to label our data, we had several options. We discovered that web-based platforms such as MakeSense [44] and Roboflow [45] provided the most user-friendly interfaces. The main difference is that Roboflow necessitates an account and offers project-saving capabilities. At the same time, MakeSense provides a rapid and lightweight alternative for label generation, allowing the option to download the labels as a .txt file immediately. Figure 3.6.2 visually represents object annotation.

Upon completing the annotation process, our attention shifted to training the YOLOv5 algorithm. It is important to emphasise that machine learning tasks are computationally intensive and require powerful hardware. Nevertheless, once trained, the models can run on lower-spec hardware [46].

3.6.1.3 Step III: Training Our YOLOv5 Model

To train our YOLOv5 model, we utilised Google Colab, a cloud-based platform that runs Jupyter Notebooks and provides the powerful *NVIDIA® T4 GPU*. The T4 GPU specifications can be found in Table 3.6.1. This allowed us to write and execute Python code efficiently. Using Google Colab was advantageous as it allowed us to focus on other project tasks while the training was ongoing. Additionally, Google Colab came pre-installed with libraries such as PyTorch and TensorFlow, which are necessary for YOLOv5.

During the training process, we had the flexibility to adjust various hyperparameters, including the size of the input images, the batch size (i.e. the amount of data processed at once), and the number of training epochs. Furthermore, we could either use pre-existing weights or create a custom model where all 7,023,610 parameters were trainable.

Throughout the training, we received regular updates through the console, providing feedback on the progress. This allowed us to monitor any potential issues or improvements. Once the training was complete, we obtained a custom YOLOv5 model that was specifically trained to detect features within the cells. We based our training on a modified version of The Roboflow team's YOLOv5 training notebook. Our version of the notebook can be found in Appendix A.

Parameter	**Value**
GPU Name	TU104
Architecture	Turing
Process Size	12 nm
Transistors	13,600 million
Shading Units (Cores)	2560
Tensor Cores	320
TeraOPS	130
RT Cores	40
Memory Size	16 GB
Memory Type	GDDR6
Memory Bus	256 bit
Base Clock	585 MHz
Boost Clock	1590 MHz
Memory Clock	1250 MHz (10 Gbps effective)
Bus Interface	PCIe 3.0 x16
TDP	70 W

Table 3.6.1: NVIDIA Tesla T4 specification sheet. [47].

3.6.2 Version 1.0: Using YOLOv5 and Roboflow

To train our first model, V1.0, we followed the step-by-step instruction on Roboflow's website [45]. Our dataset comprised 111 images encompassing hot spots and uncovered areas, our first and second milestones. Our class balance is shown in Table 3.6.2. We partitioned the dataset into training, validation, and testing sets using a 99/6/6 split, where the images were captured using iPhones and GoPros. Preprocessing and augmentation were available on Roboflow's website but not beneficial in our case.

Using PyTorch and YOLOv5, the training process began with a Google Colaboratory script downloading data from Roboflow using a unique Application Programming Interface (API) key. We employed a standard batch size of 16 and 1000 epochs. In this context, batch size refers to the number of training examples used in a single update of the model's weights. Large batch size usually means faster training time but can make the model less accurate. Epochs represent the number of times the model iterates over the entire dataset. In addition, we resized the images to a resolution of 640 pixels.

More epochs typically result in better performance but may cause overfitting if set too high, leading the model to perform worse with new unseen data. However, the training stopped at 621 epochs due to no improvement during the last 100 epochs. The best weights, at 521 epochs, were saved and uploaded back to Roboflow. The model training took approximately 37 minutes and is ready for use in the web application or offline.

Unfortunately, only two classes led to some false classifications, as seen in Chapter 4. Specifically Figure 4.3.7. To correct this issue, we created additional classes that would hopefully make our model more accurate.

Classes	**Count**
Hot spot	168
Uncovered area	83
Null examples	13

Table 3.6.2: V1.0 Classification table showing the count of each class.

3.6.3 Version 2.0: Using YOLOv5 and MakeSense

For our second iteration, we created an object detection algorithm, V2.0, to improve the distinction between tapping holes and point flames. The two were frequently misidentified as hot spots due to their similar visual attributes. Notably, point flames release hydrogen fluoride (HF), a hazardous and corrosive gas [48].

Moreover, V2.0 was designed to recognise stubs, the cylindrical mounts of the anode. This would allow the estimation of the ACM height by using the stub height as a reference. This approach, however, yielded images without any null examples, meaning there were no images without objects of interest. Although it might negatively affect accuracy as all images include stubs, this compromise was necessary to accurately identify stubs, which were occasionally misclassified as uncovered areas.

The development of V2.0 followed a YouTube tutorial by DeepLearning, which utilised the MakeSense website for image labelling [49]. A `.zip` folder containing `.txt` files with image labels, coordinates, and corresponding full-resolution iPhone images from both the *train* and *validate* folders were processed. The decision to use iPhone images was motivated by their image quality and a preferred 26 mm focal length compared to the GoPro. However, exploring how well V2.0 could perform when applied to GoPro images was also of interest. The training dataset comprised 55 images and their respective `.txt` files from the train folder. Simultaneously, the validate folder consisted of 13 images and their corresponding `.txt` files for assessing object detection accuracy during training.

The training process continued to utilise the same YOLOv5 repository on a Google Colaboratory script, where images were resized to 640 pixels. The class balance of V2.0 is shown in Table 3.6.3.

Classes	**Count**
Hot spot	79
Uncovered area	51
Stub	298
Point flame	36
Tapping hole	5

Table 3.6.3: V2.0 Classification table showing the count of each class.

3.6.4 Version 2.1: Using YOLOv5 and RoboFlow

The success of V2.0 encouraged us to extend our dataset by incorporating the complete set of images, including the GoPro images, to create V2.1.

V2.1 investigated whether the performance of V2.0 could be enhanced by employing a larger dataset, despite GoPro's less preferable focal length. Like its predecessor, V2.1 encompasses all five classes: hot spots, uncovered areas, point flames, stubs, and tapping holes. While V2.0 was developed using a dataset of 55 images, V2.1 utilised the same 111 images as in V1.0, with the additional classes annotated accordingly. Although the dataset remained the same, the class balance differed significantly due to the annotation of the three supplementary classes. V2.1 class distribution is shown in Table 3.6.4. Notably, unlike V1.0, V2.1 does not contain any null examples.

Classes	**Count**
Hot spot	91
Uncovered area	193
Stub	962
Point flame	98
Tapping hole	14

Table 3.6.4: V2.1 Classification table showing the count of each class.

Besides employing a larger dataset, we attempted to fine-tune the training parameters, such as batch size and training image size, and apply various RoboFlow filters. This was undertaken because no documentation was available for optimal training values, leading us to test for ourselves. Our model ultimately achieved the best results without any RoboFlow filters, using a batch size of 16 and a training image size of 1440 pixels.

Filter version	**Applied Roboflow filter**	**Training image resolution**	**Batch size**	**mAP 0.5**	**mAP 0.5:0.95**	**Epochs**	**Training time (hours)**
1	Split 81/21/7	1088	16	0.782	0.421	147	0.302
2	Split 89/20/0	1088	16	0.806	0.432	266	0.466
-	-	1088	10	0.808	0.427	178	0.361
-	-	1088	8	0.783	0.454	301	0.896
-	-	2240	4	0.789	0.43	242	1.556
-	-	1088	30	0.785	0.442	543	0.771
-	-	1440	16	0.807	0.46	286	0.793
-	-	1920	8	0.784	0.446	282	1.280

Table 3.6.5: Test results showing the performance of V2.1 on a *NVIDIA® T4 GPU*.

Table 3.6.5 demonstrates that reducing batch size to accommodate larger training image sizes does not enhance performance; the results are less effective. Although the version with 1440 pixels proves to be the best, the improvement is marginal. It is noteworthy that training image size primarily affects training time. Investigating the potential gains achieved by employing a more powerful GPU would be intriguing, as it would enable the use of larger images and batch sizes.

3.7 Running our Models Locally

In addition to running our models on Google Colab, we also developed Python scripts for local execution. This approach gave us greater flexibility in customising the colours of the classes and the font styles. In addition, to test the performance on a regular consumer computer.

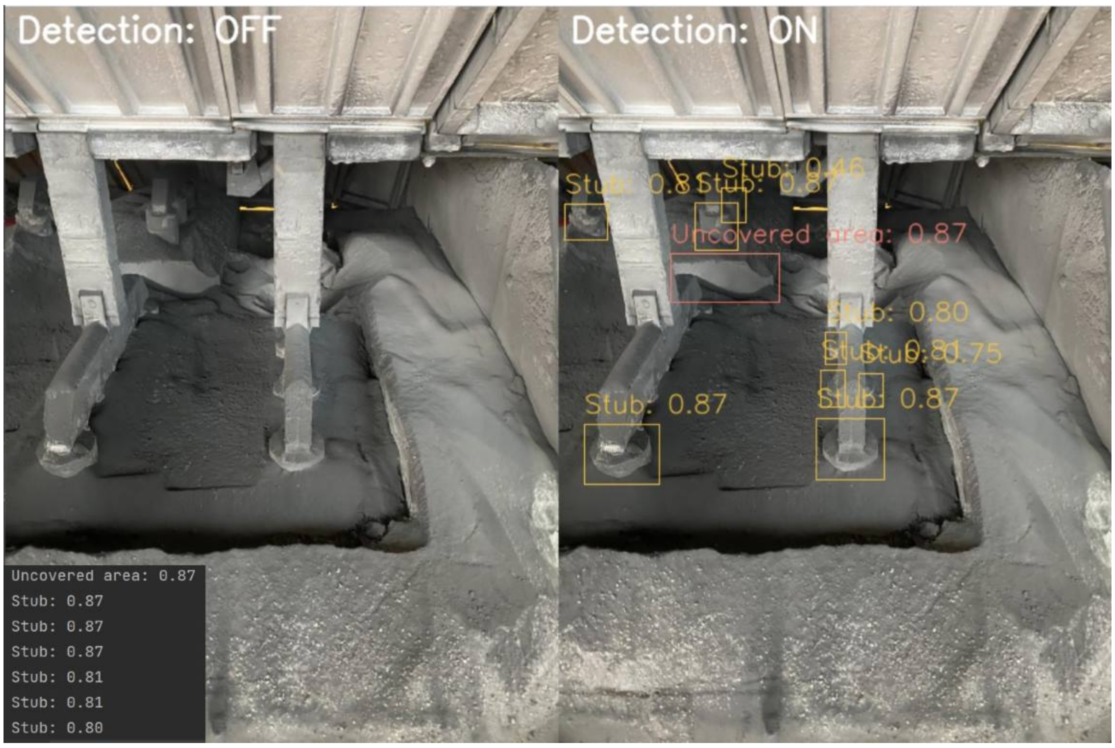
3.7.1 Implementation with Images

We started by cloning the YOLOv5 repository and creating separate Python scripts for our project. These scripts involved importing essential libraries like PyTorch, OpenCV, and NumPy. Subsequently, we loaded pre-trained models (i.e. V1.0, V2.0 and V2.1), which we downloaded from Google Colab.

We preprocessed the input images in a few key steps to achieve optimal performance. First, we maintained the aspect ratio of the images to prevent distortion. Subsequently, we resized them to a target size that is divisible by the model's stride, ensuring that the convolutional layers in the model can scan every region of the image without missing any sections. The stride refers to the number of pixels the convolutional filter moves at each step during its scan across the image. Following this, we transformed the preprocessed images into tensors, which are multi-dimensional arrays suitable for computational operations in machine learning models.

We established two thresholds, one for confidence and another for Non-Maximum Suppression (NMS), which filters out weaker detections. Each threshold was set to standard 0.4 and 0.5, respectively. The model processed the input image and generated outputs through bounding boxes and class labels. We employed a dictionary to map each class label to a unique colour for visualisation purposes.

Lastly, we used OpenCV's rectangle and text drawing functions to display the annotated image, which included the detected objects, their corresponding class labels, and their confidence scores represented in decimal probabilities. The result is as seen in Figure 3.7.1.



```
Uncovered area: 0.87
Stub: 0.87
Stub: 0.87
Stub: 0.87
Stub: 0.81
Stub: 0.81
Stub: 0.80
```

Figure 3.7.1: Interface for operators showing detection **off** and **on** with a console log of the precision point probability of each class.

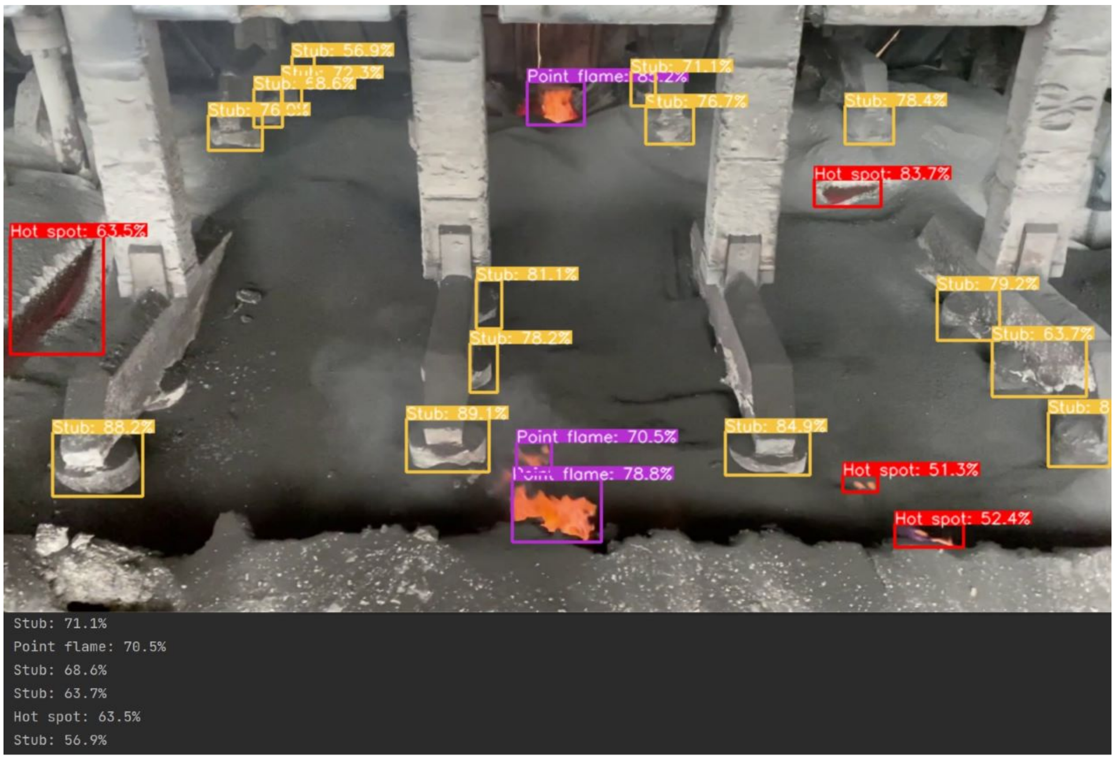
3.7.2 Implementation with Video

We observed our algorithm's strong performance on images and decided to capture videos for a more accurate representation of its application at the smelter. This change would also generate additional training material for our ML model.

We modified our code to handle video input using OpenCV's `cv2.VideoCapture` function, obtaining video properties and setting up an output video writer. The primary change involved processing input video frame-by-frame, preprocessing each frame, and converting it into a tensor. We applied the algorithm to detect objects in each frame and visualised results using OpenCV functions. Annotated frames were written to the output video, and the user could view the output in real-time.

We encountered a challenge when saving the processed video, which was resolved by ensuring the correct order of `out.release()` and `cap.release()`. Another challenge we encountered is that when the algorithm was employed locally on a Central Processing Unit (CPU), latency issues did occur. One can mitigate this issue by reducing the Frames Per Second (FPS) resolution or opting to run it on a dedicated Graphics Processing Unit (GPU).

Finally, in line with Dr Morten Karlsen's assertion that the primary objective is to help operators comprehend their tasks, we altered the probability representation from decimal to percentage—this modification aimed to enhance the operators' understanding of the bounding boxes. See Figure 3.7.2 for the end result.



```
Stub: 71.1%
Point flame: 70.5%
Stub: 68.6%
Stub: 63.7%
Hot spot: 63.5%
Stub: 56.9%
```

Figure 3.7.2: Alternative interface for operators with class probability depicted in percentage.

CHAPTER

FOUR

RESULTS

4.1 Results: Solution One

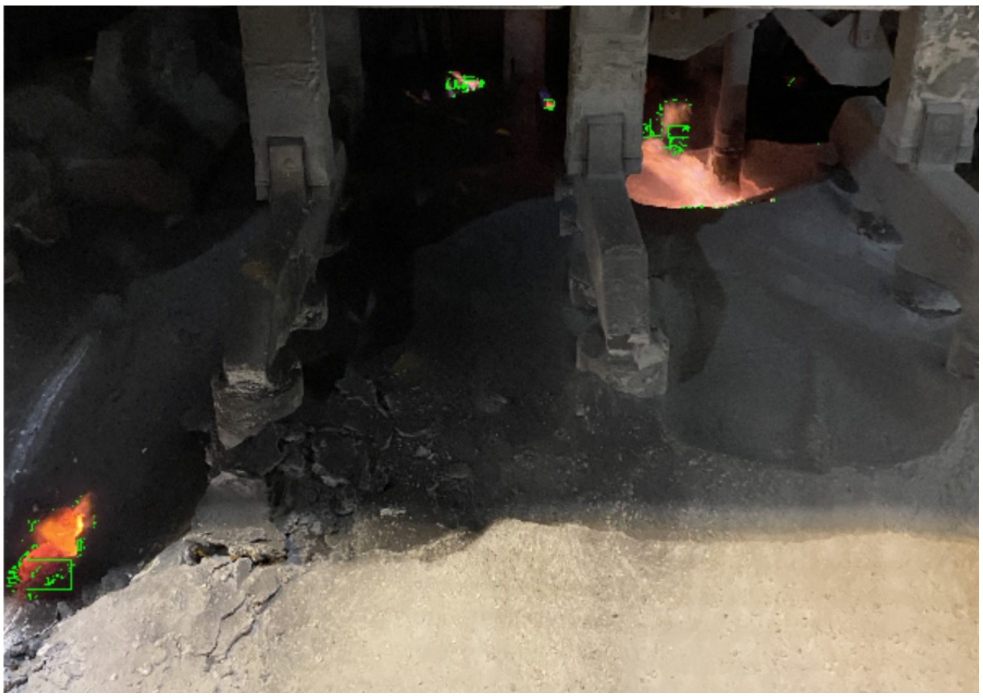
As previously mentioned, in Solution One, we devised a colour and edge-detection model to address our initial two milestones:

- Detecting hot spots through colour detection.
- Identifying uncovered areas through edge detection.

Regrettably, the results were unsatisfactory concerning detecting uncovered areas.

4.1.1 Colour and Edge Detection Examples

As demonstrated in Figure 4.1.1a, this solution detects all colours represented by the HSV values depicted in Figure 3.5.4. Consequently, tapping holes and point flames are surrounded by green detection clusters. Furthermore, upon altering the colour temperature of Figure 3.5.5, which successfully detected a hot spot, the model failed to detect anything of interest, as shown in Figure 4.1.1b.


(a) Tapping hole and point flames detect as hot spots.

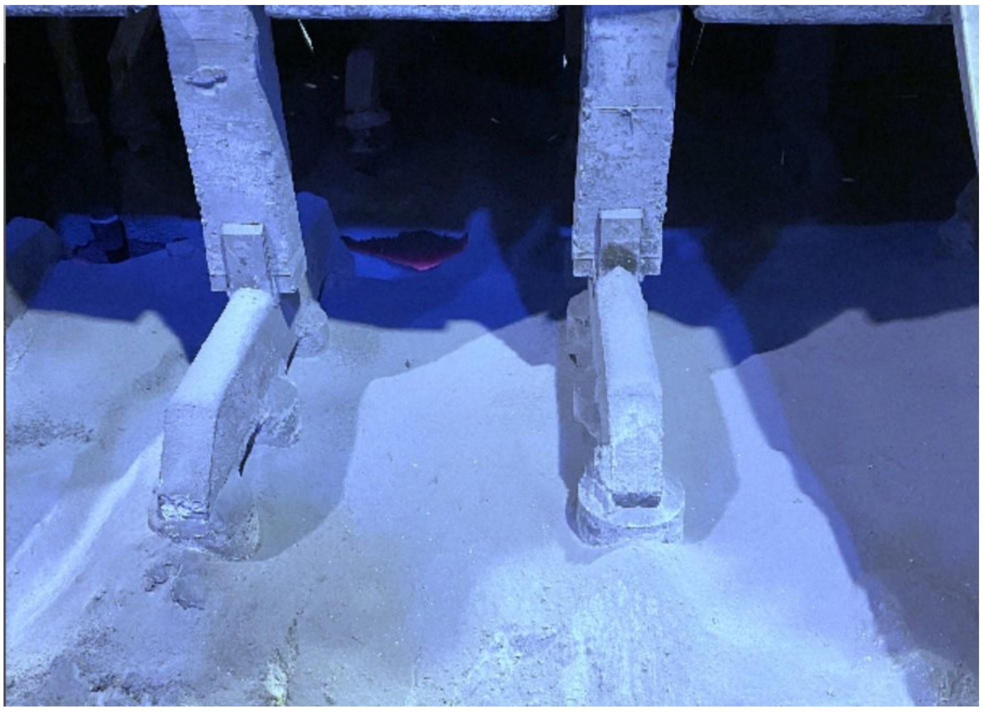

(b) Failed to detect the hot spot after colour balance adjustment.

Figure 4.1.1: Illustrating the limitations of simple colour detection.

In addition to hot spots, we tried to detect uncovered areas with the simple `cv2.Canny` method. However, as seen in Chapter 3.5.3, we were unsuccessful.

4.2 Results: Solution Two

Solution Two employs a ML approach using YOLOv5. The following subsection presents three versions of a ML model. It details their quantitative measures, such as Precision and recall, and uses images and videos to evaluate their performance in real-world applications.

4.2.1 Version 1.0: YOLOv5 and Roboflow

This version focused on just two classes: hot spots and uncovered areas. After training this model, we used TensorFlow to present our model's performance graphically, as shown in Figures 4.2.2, 4.2.3, 4.2.4 and 4.2.5.

The standard plot for ML tasks is the Receiver Operating Characteristic (ROC) curve. However, according to a paper by Takaya Saito and Marc Rehmsmeier for unbalanced data such as ours, the Precision-Recall (PR) curve is preferable [50]. Figure 4.2.1 illustrates textbook examples of PR curves.

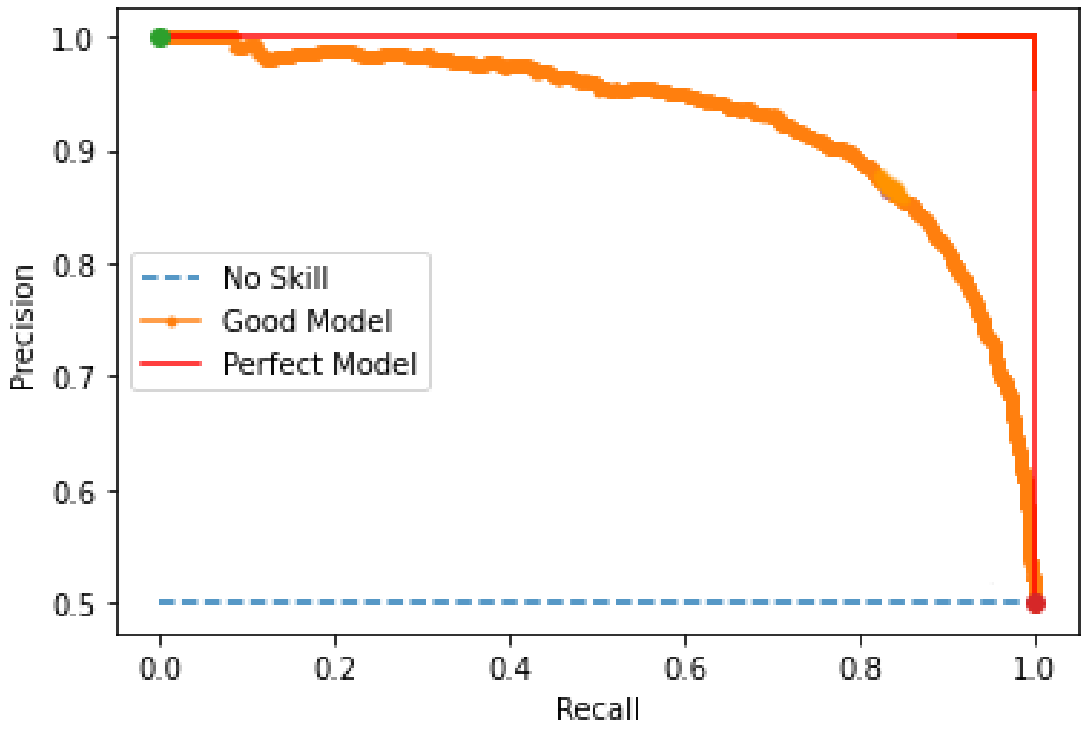


Figure 4.2.1: A textbook demonstration of PR curves for three models: *No Skill*, *Good Model*, and *Perfect Model*. Illustrating clearly the differences in performance among models [51].

The PR curve shows how *Precision* and *Recall* correlate. We must understand what the two metrics describe to understand this curve fully.

Precision is a measure of the accuracy of a model's positive predictions, indicating the proportion of correct positive predictions out of all positive predictions made by the model.

$$Precision = \frac{\text{TP}}{(\text{TP} + \text{FP})} \tag{4.1}$$

Recall is a metric that measures the proportion of correct positive predictions out of all actual positive instances in the data.

$$Recall = \frac{\text{TP}}{(\text{TP} + \text{FN})} \tag{4.2}$$

Where:

- **TP** is True Positive.
- **FP** is False Positive.
- **FN** is False Negative.

A true positive refers to correctly detecting a hot spot when there is one. A false positive, on the other hand, occurs when a hot spot is mistakenly detected without actually being present. Lastly, a false negative occurs when a hot spot that should have been detected is missed.

In other words, a higher Precision implies that the model has a lower rate of false positives and is more capable of correctly identifying positive instances. In contrast, a higher Recall indicates that the model has a lower rate of false negatives and is more capable of identifying all positive instances in the data, even at the cost of including some false positives. These metrics allow us to prioritise either high Precision or recall by adjusting the confidence threshold during the deployment of our model.

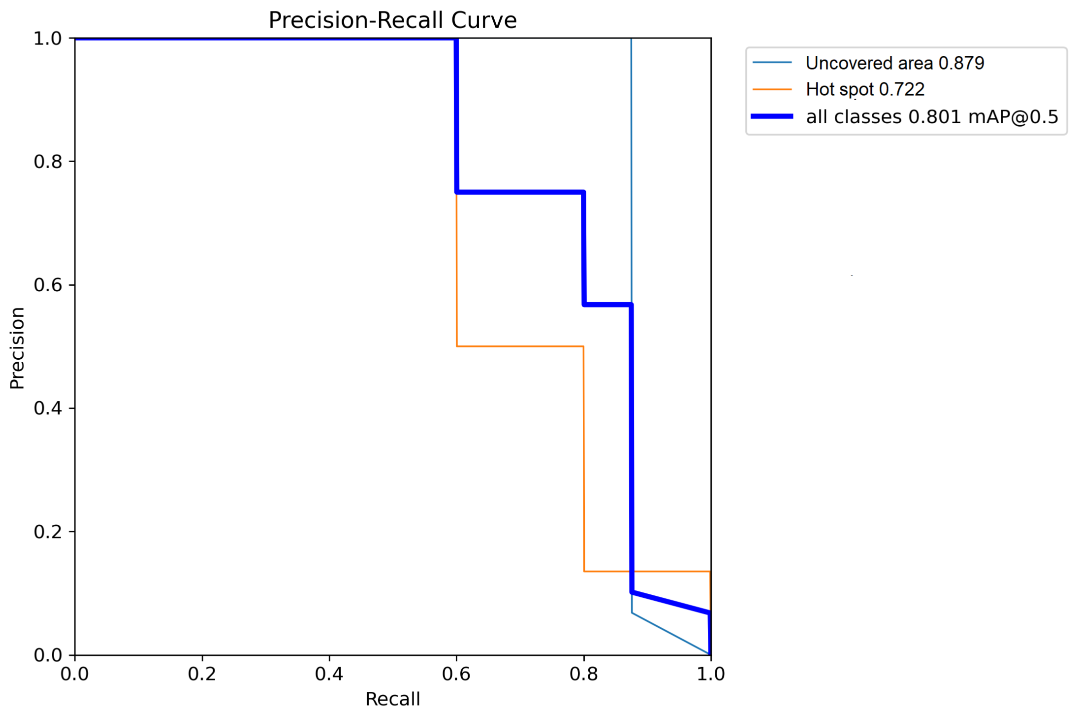


Figure 4.2.2: PR curve of Version 1.0.

Figure 4.2.2 shows the PR curve for this model. In this case, we can see there is a steep trade-off between the two metrics, clearly demonstrating a significant trade-off between Precision and recall. As Precision escalates, there is a corresponding decrease in recall. This suggests that while we can achieve high Precision, it might be at the expense of overlooking some relevant instances. However, it is decently close to the *Good Model* shown in Figure 4.2.1.

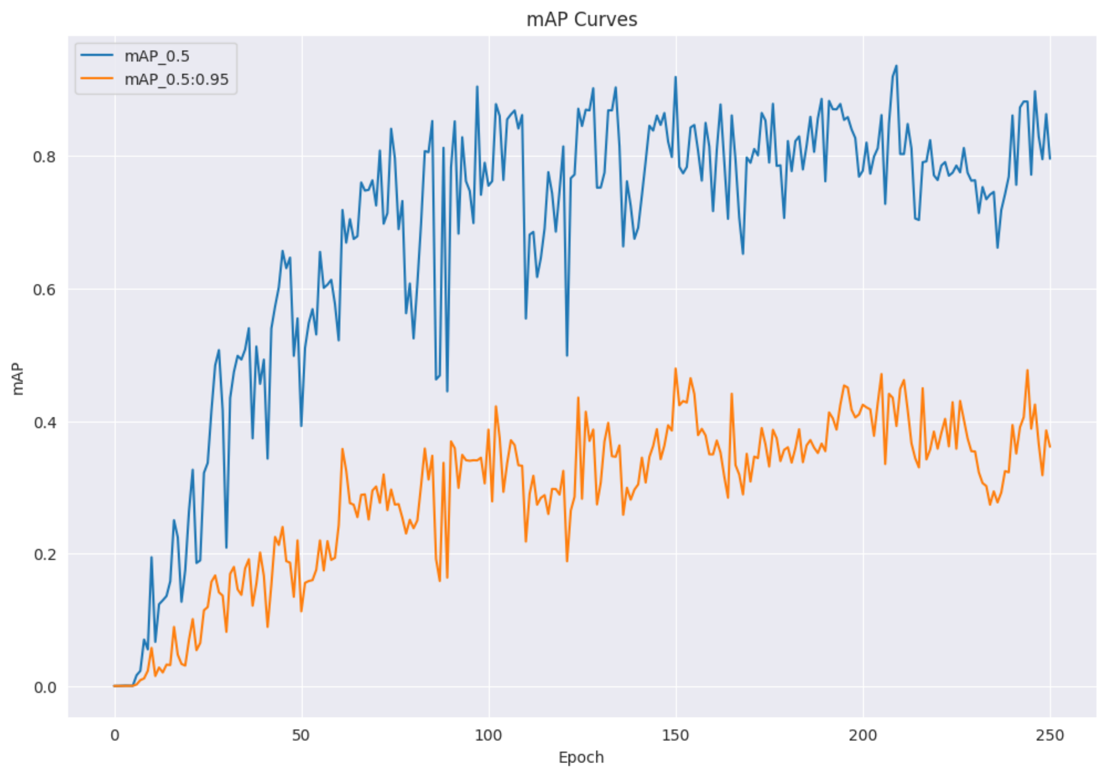


Figure 4.2.3: mAP curves of Version 1.0.

The two graphs in Figure 4.2.3, namely mAP 0.5 and mAP 0.5 : 0.95, represent the mAP for two distinct classes: *Uncovered area* and *Hot spot*. These are the same two classes depicted in Figure 4.2.2. The mAP value is calculated using the data from the confusion matrix, recall, Precision and Intersection over Union (IoU), making it a very proficient metric to evaluate our model's performance. The equations of AP, mAP and IoU is given by Equation 4.3, 4.4 and 4.5

$$\text{AP} = \sum_{k=1}^{N} \frac{P(k) \cdot rel(k)}{\text{number of relevant documents}} \tag{4.3}$$

$$\text{mAP} = \frac{1}{n} \sum_{k=1}^{k} \text{AP}_k \tag{4.4}$$

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{A \cap B}{A \cup B} \tag{4.5}$$

The mAP 0.5 curve shows the Precision for a confidence threshold of 0.5 which is not high. We want our model to be as accurate as possible, so we look to the 0.5 : 0.95 curve, which shows the model's Precision across the confidence thresholds from 0.5 to 0.95, giving a more holistic view of our model's performance. This model has mAP scores of 0.919 and 0.479, respectively. This was achieved after 150 epochs and took approximately 8 minutes.

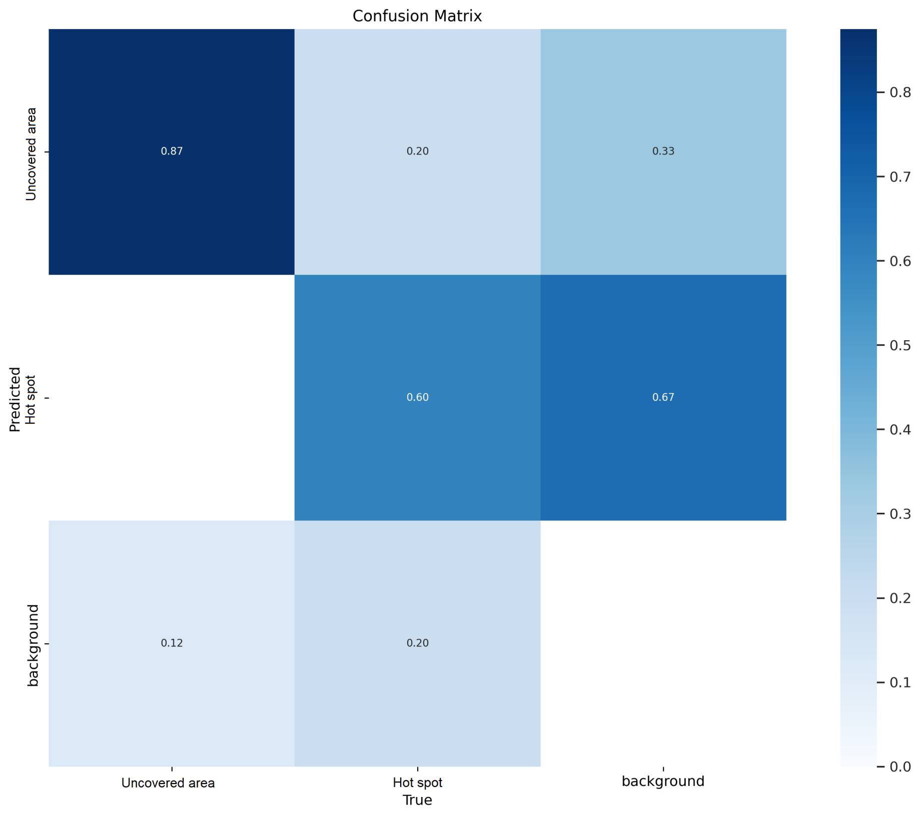


Figure 4.2.4: Confusion matrix of Version 1.0.

The confusion matrix is possibly the most important metric to understand our model's performance as it is better visualised. Along the y-axis, we have the objects predicted, crossed with objects detected along the x-axis. The diagonal shows the rate of true positives. One can examine this by observing the three squares furthest to the left. This indicates the proficiency of our model in accurately identifying *uncovered areas* while also demonstrating its ability to avoid misclassifying these as *hot spot* or *background*. Our data set did not include many images without any objects, which leads to this model separating objects from the background unreliably.

Examining Figure 4.2.4, it seems our model often mistakes hot spots for background. This raised some concerns that were later confirmed when we tested this model with video. This also showed us this version was quite often mistaking stubs to be uncovered areas which is problematic and made it evident to us that version 2.0 is preferred.

A compelling aspect of the confusion matrix is its relationship to the financial calculations detailed in Appendix D, which accounts for the combined cost/lost income from hot spots and uncovered areas. According to these calculations, Hydro Årdal could save approximately 14.5 million NOK annually if the model reliably detects all hot spots or uncovered areas. This means that the financial implications of reliable detection are substantial.

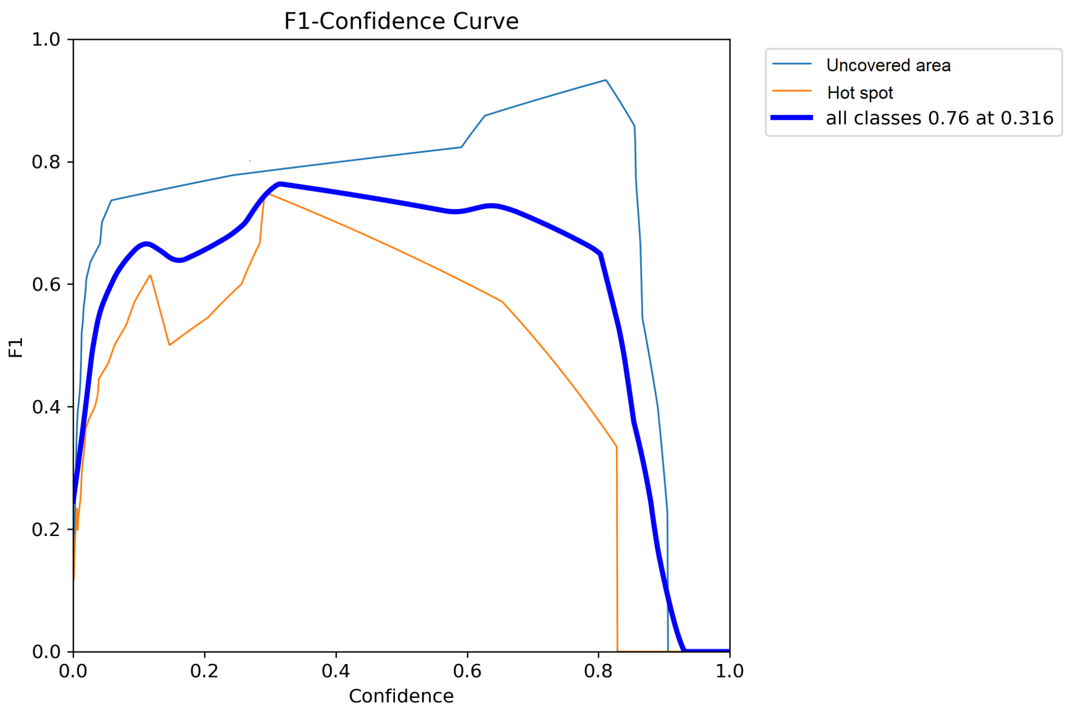


Figure 4.2.5: F1 curve of Version 1.0.

Lastly, we can review the F1 curve. The F1 score is calculated as seen in Equation 4.6. This is the combination of the Precision and recall metrics into a harmonic mean, which provides a balanced evaluation of our model's performance.

$$F1 = \frac{(1+\beta^2)\cdot(\text{precision}+\text{recall})}{(\beta^2\cdot\text{precision})+\text{recall}} \tag{4.6}$$

The F1 score tells us what the ideal threshold for our model is. When β is set equal to 1, we get the curve in Figure 4.2.5. For this model, it is given as 0.316 with an F1 score of 0.76. However, we could use Equation 4.6 to choose whether we want a higher precision or recall by tuning the value of β.

4.2.2 Version 2.0: YOLOv5 and MakeSense

The goal of V2.0, which involved the implementation of YOLOv5 for training and MakeSense for object labelling, was to demonstrate the version's capability to identify and separate diverse objects accurately. The results were noteworthy, considering the version was trained on a dataset comprising 55 images and validated on a separate set of 13 images.

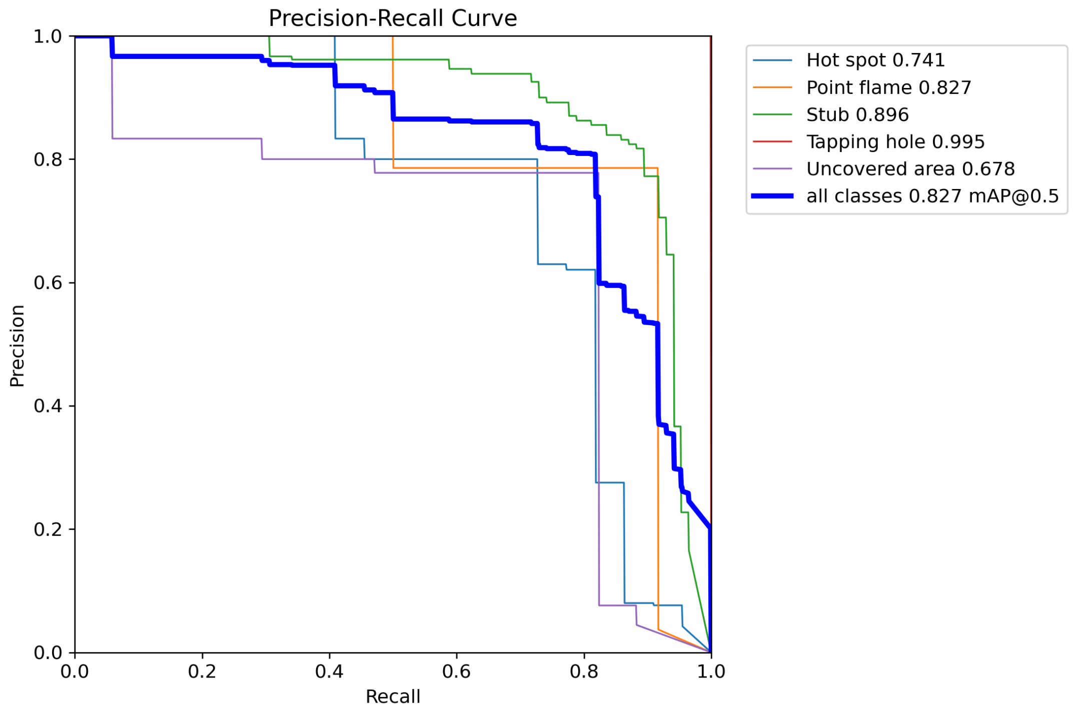


Figure 4.2.6: PR curve of Version 2.0.

This version's PR curve has a smoother trade-off between Precision and Recall. We can see that we do not have to sacrifice Precision to attain higher Recall values compared to version 1.0 as the curve for all classes is approaching the *Perfect Model* depicted in Figure 4.2.1

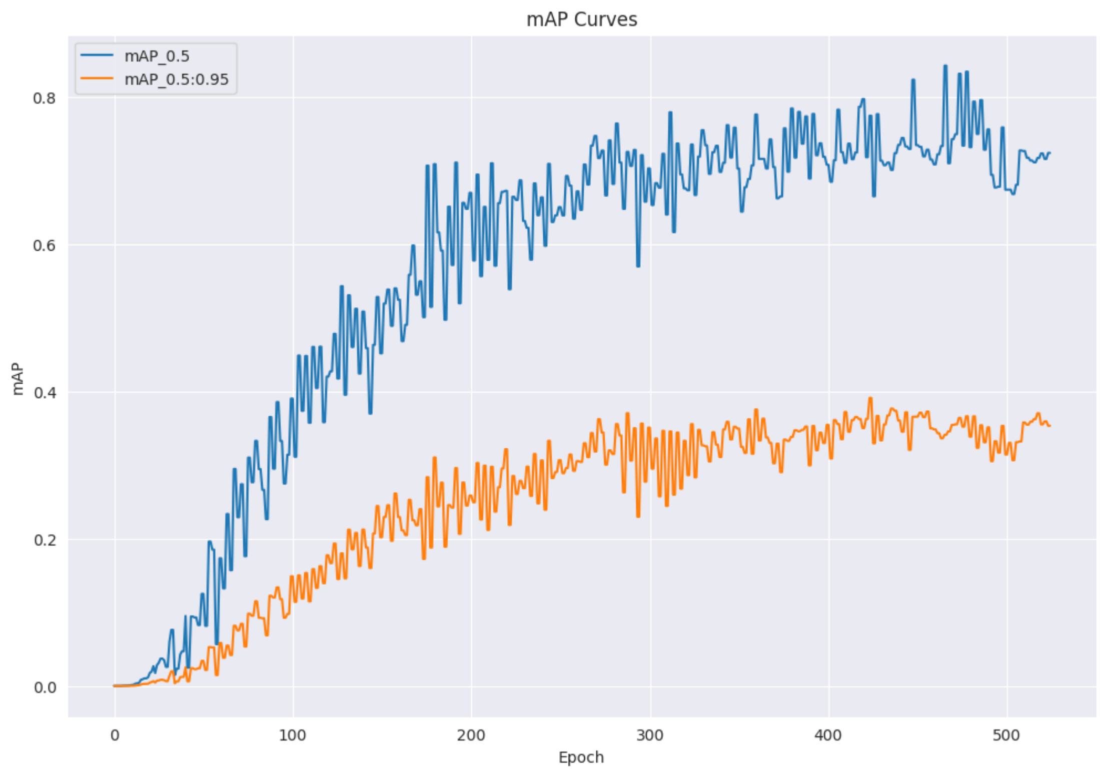


Figure 4.2.7: mAP curves of Version 2.0.

As seen in Figure 4.2.7, the mAP scores are lower than in V1.0. This is to be expected, seeing as this version includes five classes instead of two. The mAP 0.5 score of 0.775 and the mAP 0.5 : 0.95 score of 0.391, though not considerably lower, are still not as favourable as those of V1.0. This version achieved its best training result after 424 epochs and was completed in 10 minutes.

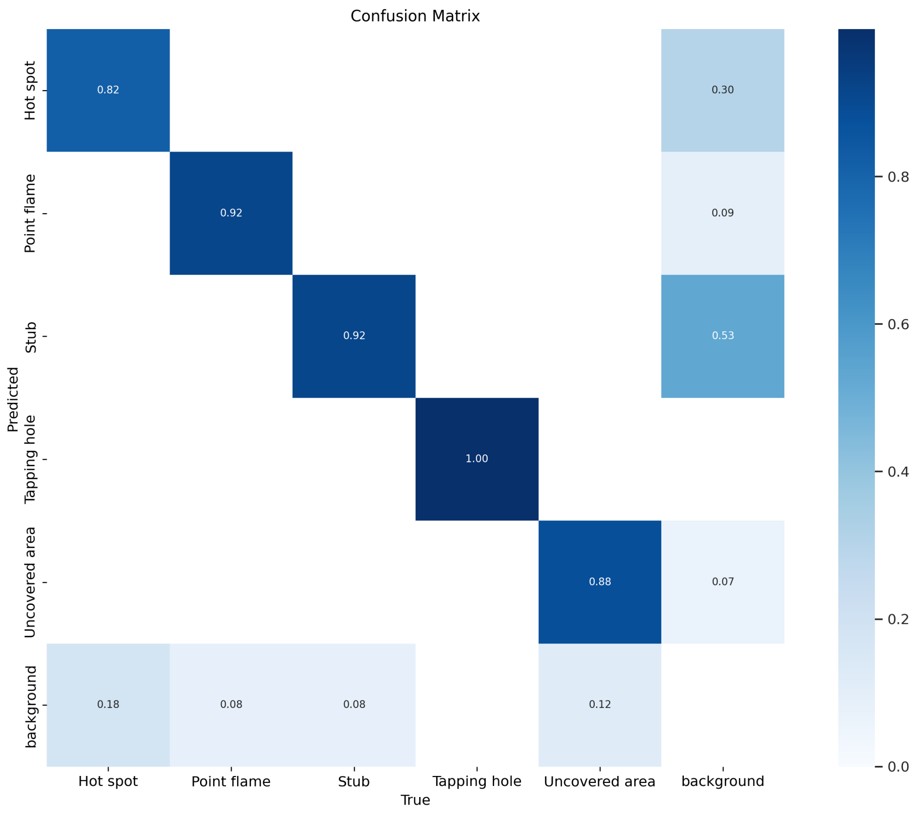


Figure 4.2.8: Confusion matrix of Version 2.0

The confusion matrix for V2.0, illustrated in Figure 4.2.8, is distinct due to additional classes, resulting in a larger grid. A notable aspect of this matrix is the significant reduction in the frequency of mistaking uncovered areas for the background, a trend observable across almost all classes. Most likely because each image contains a larger number of recognised objects, reducing the background amount. This indicates that solely relying on mAP scores may not sufficiently evaluate our model's performance, and it also helps explain why V2.0 outperformed V1.0 in video tests.

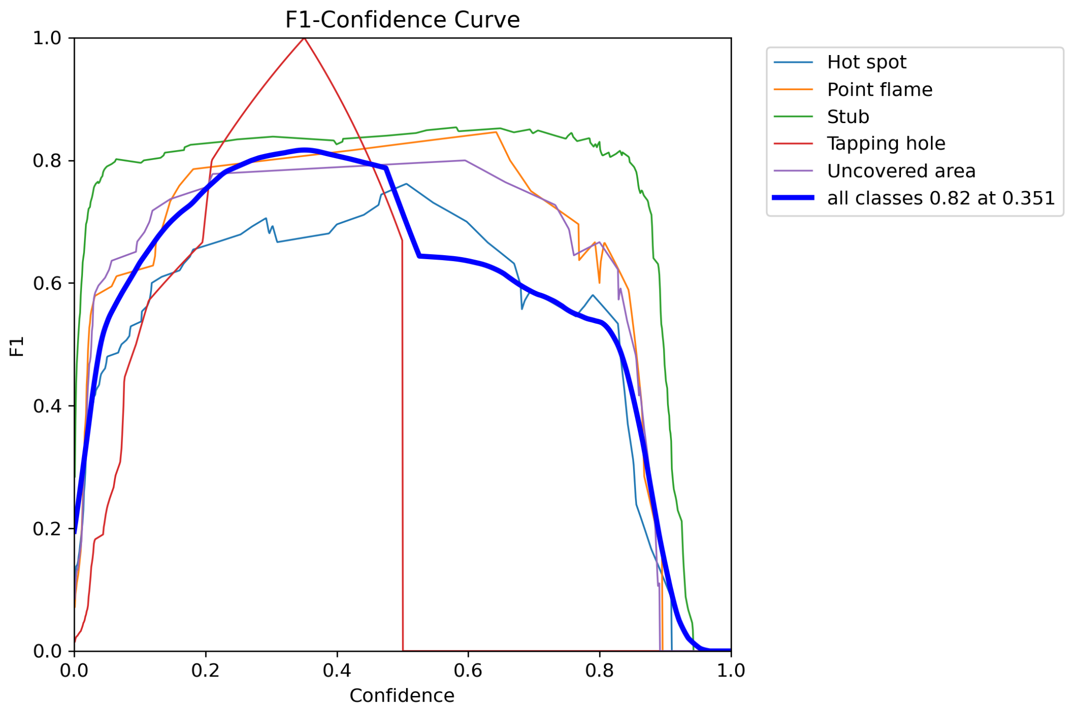


Figure 4.2.9: F1 curve of Version 2.0.

As illustrated in Figure 4.2.9, the ideal threshold for V2.0 is 0.35, which results in an F1 score of 0.82, an improvement over V1.0. However, poor tapping hole detection notably reduces the overall score.

4.2.3 Version 2.1: YOLOv5 and RoboFlow

As stated in Chapter 3.6.4, V2.1 builds upon V2.0 and aims to improve it. This version was, like the others, tested on both images and video.

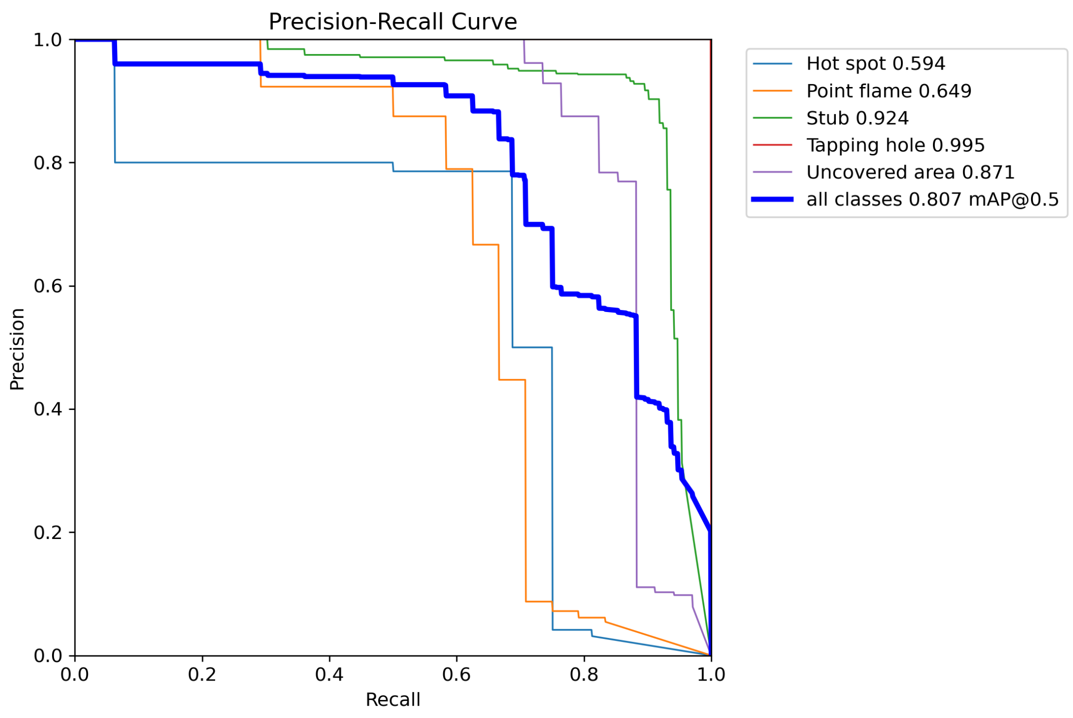


Figure 4.2.10: PR curve of Version 2.1.

The PR curve of V2.1 closely resembles its predecessors. A comparison reveals unique strengths in each version; V2.0, depicted in Figure 4.2.6, is better in detecting hot spots, while V2.1, presented in Figure 4.2.10, performs better in identifying uncovered areas. Overall, V2.0 has a similar score of 0.827 compared to V2.1's 0.807.

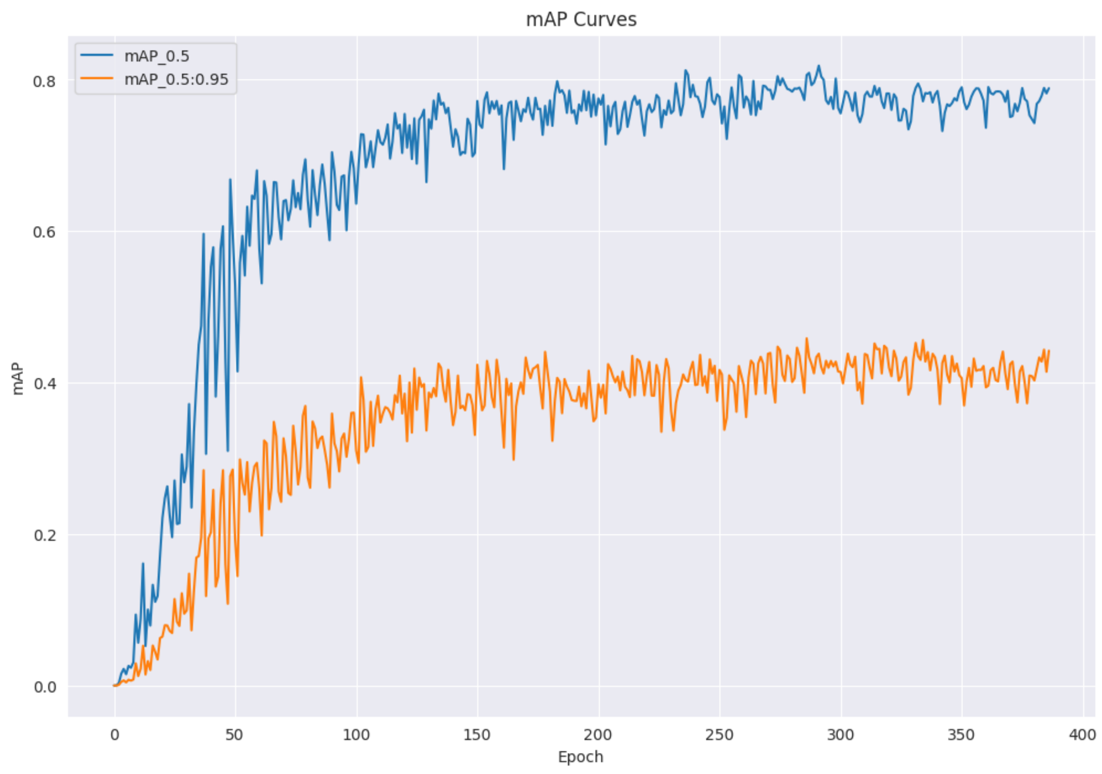


Figure 4.2.11: mAP curves of Version 2.1.

Figure 4.2.11 shows comparable results as the previous version. Comparing these results, we can clearly see from the mAP graphs that V2.1 is preferred in this regard, having an mAP 0.5 score of 0.807 and mAP 0.5 : 0.95 score of 0.46. This result was achieved after 286 epochs and was completed in approximately 47 minutes.

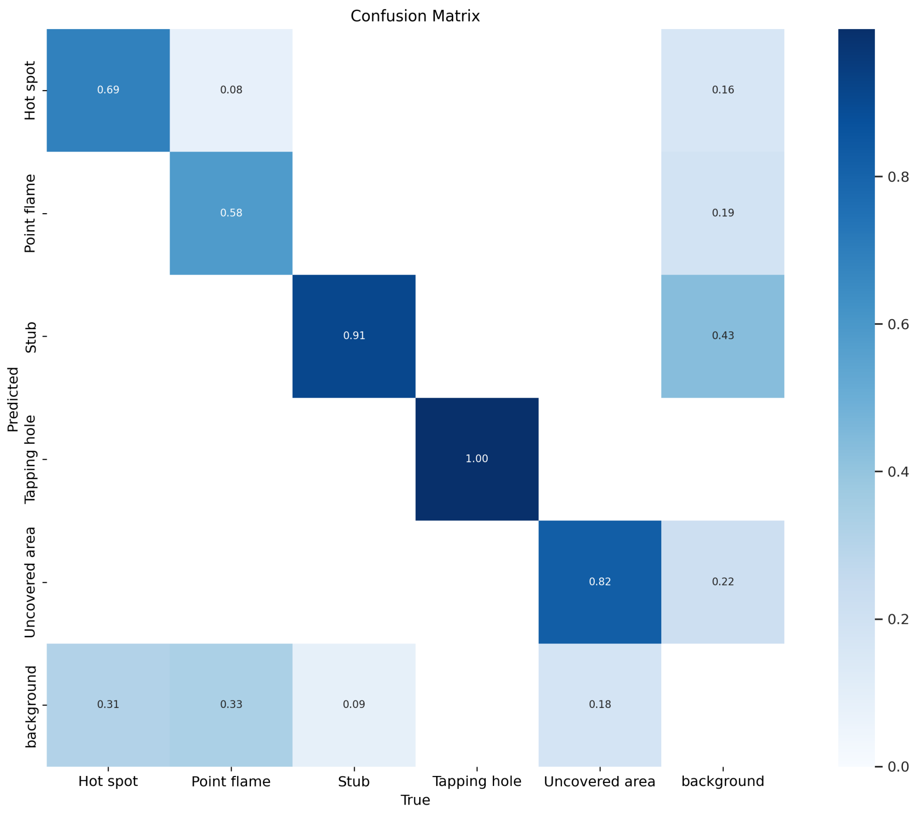


Figure 4.2.12: Confusion matrix of Version 2.1.

The comparison becomes more complex upon examining the confusion matrix. A specific point of interest is the comparison of the *Predicted Point flame* with the *True Point flame* for both versions. V2.0 scores 0.92, while V2.1 yields a lower score of 0.58. Intuitively, one might expect the larger dataset of V2.1, with twice the amount of images, to offer enhanced results. However, a side-by-side comparison of Figure 4.2.8 and 4.2.12 suggests that this is not necessarily the case.

Two potential explanations for these observations come to mind. First, the class imbalance in V2.1 might be more pronounced than in V2.0. Second, differences in image sizes between the datasets could contribute. Notably, V2.0 exclusively utilises iPhone images, all of which share the exact dimensions, while V2.1 incorporates a mixture of iPhone and GoPro images, which differ in both aspect ratio and size.

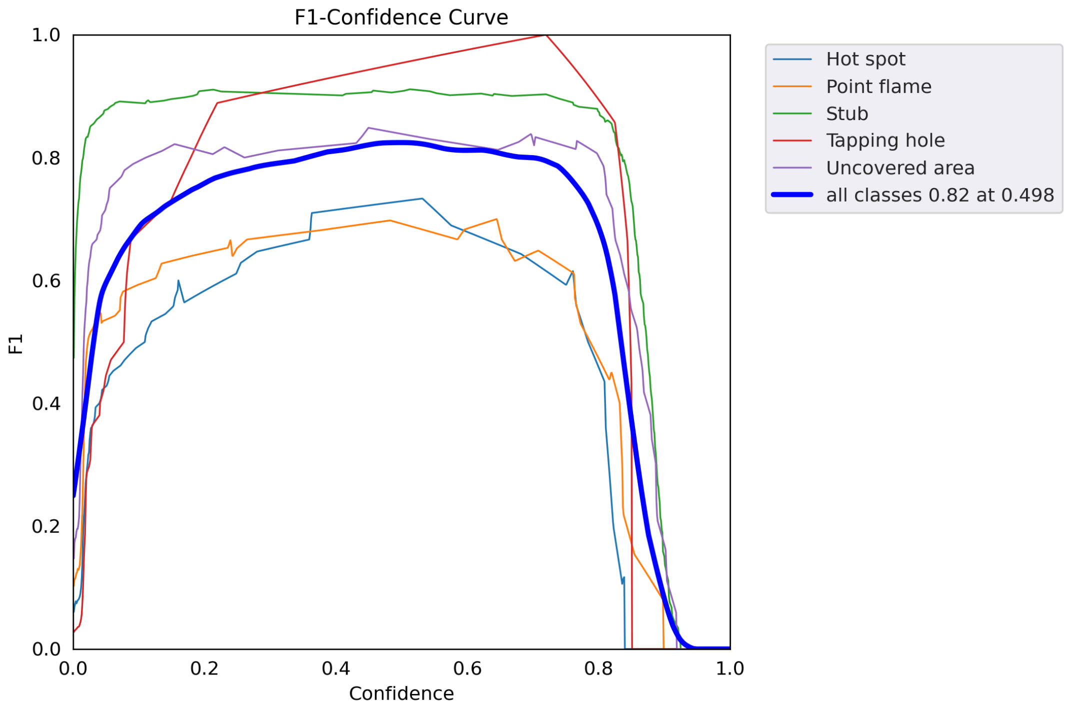


Figure 4.2.13: F1 curve of Version 2.1.

In Figure 4.2.13, we observe the F1 score for V2.1. Notably, the score for the *Tapping hole* shows an improvement, indicating that a threshold of 0.498 should be applied to achieve an F1 score similar to that of V2.0, as depicted in Figure 4.2.9. However, the information these graphs provide is somewhat contradictory, making it challenging to determine the better model. Thus, further practical testing is required to confirm the optimal model selection, thus leading to the next section.

4.3 Testing: Images and video

In this section, we look at the real-world results of our algorithm. From left to right, V1.0, V2.0 and V2.1 are tested, respectively. We used thresholds of 0.4 for confidence and an NMS of 0.5 due to the limited accuracy of our models, as seen in the mAP curves.

As mentioned, videos and images were captured using two devices: a GoPro HERO11, with a wide focal length of 16.5 mm, and an iPhone 11, with a focal length of 26 mm. Each device was placed at a distance of approximately 50 cm from the cell. Furthermore, the classification of the objects is depicted in probability precision point. However, as previously mentioned, we opted for percentage probability for the videos.

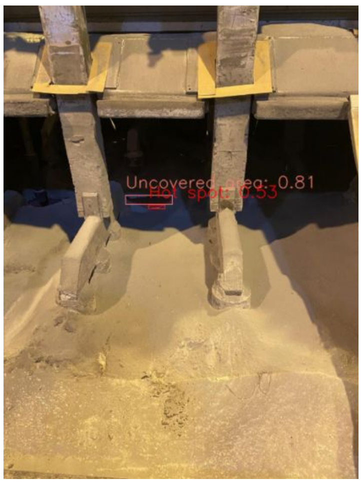
The results are discussed in sections 4.3.3 and 4.3.4. For the videos and images, the link can be found as a Google Drive link in Appendix A

The folder from which the relevant videos and images can be found is called *Test Videos and Images (with detection)*. Within this folder, there are four separate folders titled:

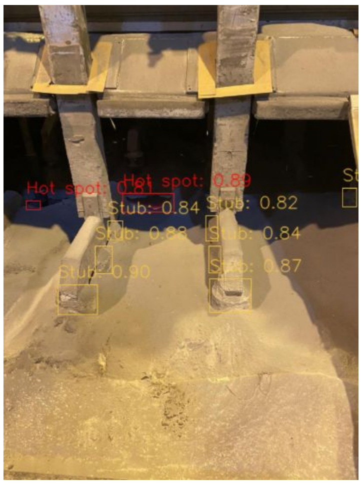
- GoPro images - contain the GoPro test images discussed for all versions.
- GoPro videos - contain the GoPro videos discussed for all versions.
- iPhone images - contain the iPhone images discussed for all versions.
- iPhone videos - contain the iPhone videos discussed for all versions.

4.3.1 iPhone Images

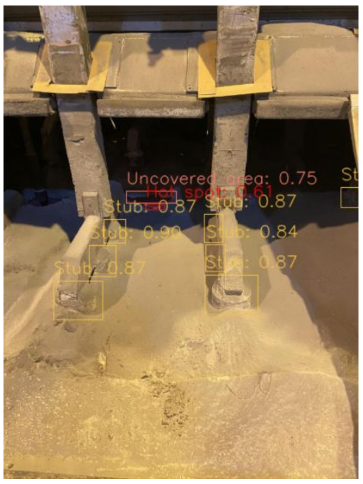
These are chosen images due to their importance and worth noting to showcase our results. The figures in this part are all taken with an iPhone 11 Pro with a focal length of 26 mm. As mentioned earlier, this focal length gives the most comprehensive view of the cell.



(a) V1.0 Console output:
```
Uncovered area:  0.81
Hot spot:  0.53
```



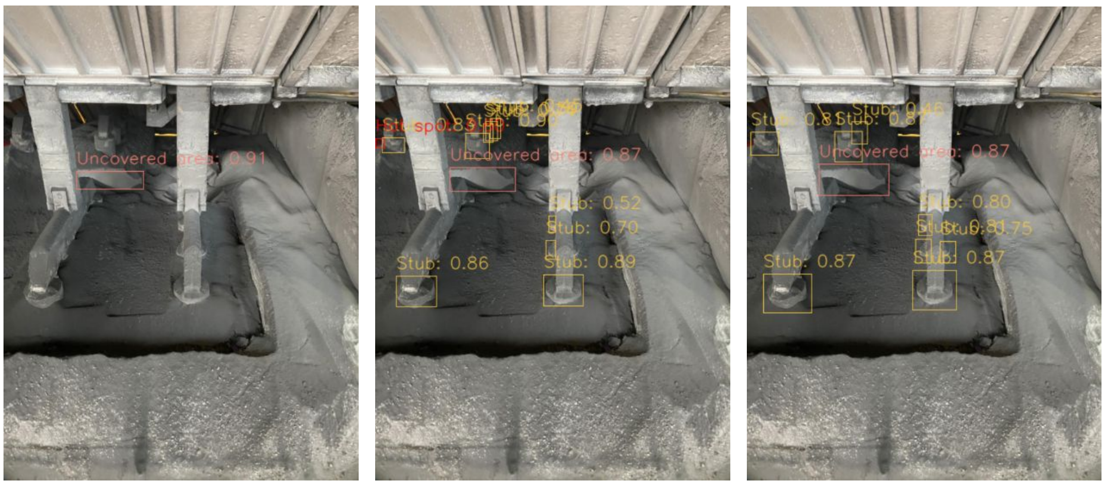
(b) V2.0 Console output:
```
Hot spot:  0.89
Hot spot:  0.81
```



(c) V2.1 Console output:
```
Uncovered area:  0.75
Hot spot:  0.61
```

Figure 4.3.1: Real-world test for detecting a hot spot.

Figure 4.3.1 was chosen because it has a hot spot that requires covering. The best performance is by V2.0 with a hot spot probability of 0.89, followed by V1.0 and V2.1. However, this hot spot is not fully formed. One can argue that it is a combination of an uncovered area and a hot spot.



(a) V1.0 Console output:

```
Uncovered area:  0.91
```

(b) V2.0 Console output:

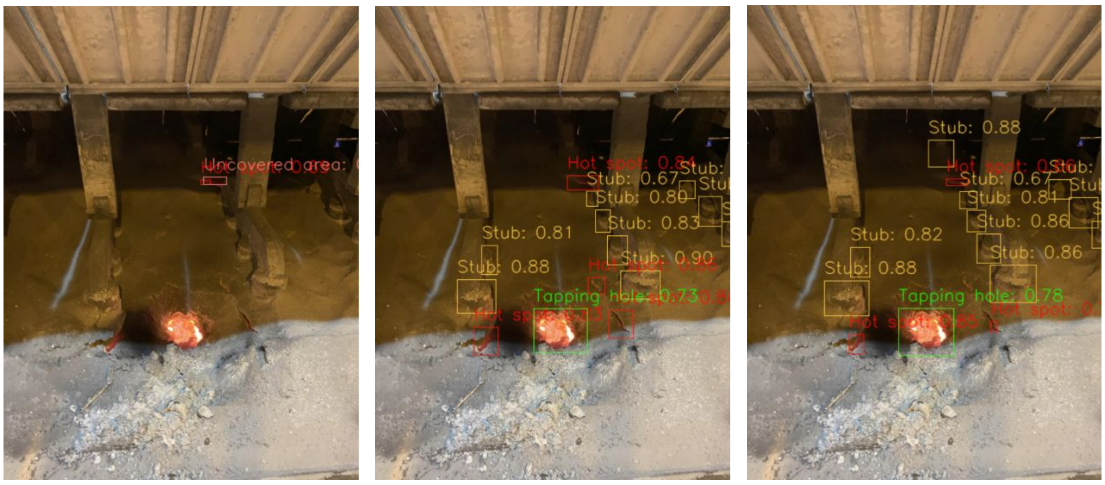
```
Hot spot:  0.59
Uncovered area:  0.87
```

(c) V2.1 Consloe output:

```
Uncovered area:  0.87
```

Figure 4.3.2: Real-world test for detecting uncovered area.

Figure 4.3.2 was chosen because it has an uncovered area that, with enough time, will become a hot spot. In addition, one can see that an operator has tried to redress the cell but did not have enough ACM to cover the uncovered area. One can tell by the scrape marks on the ACM between the stubs. Surprisingly, the highest probability score is V1.0 at 0.91 for the uncovered area, but V2.0 found a barely visible hot spot with a probability of 0.45. V2.1 and V2.0 have the same probability of the uncovered area but do not detect the hot spot.



(a) V1.0 Console output:

```
Hot spot: 0.69
Uncovered area: 0.51
```

(b) V2.0 Console output:

```
Hot spot: 0.84
Hot spot: 0.86
Hot spot: 0.82
Hot spot: 0.83
Tapping hole: 0.73
```

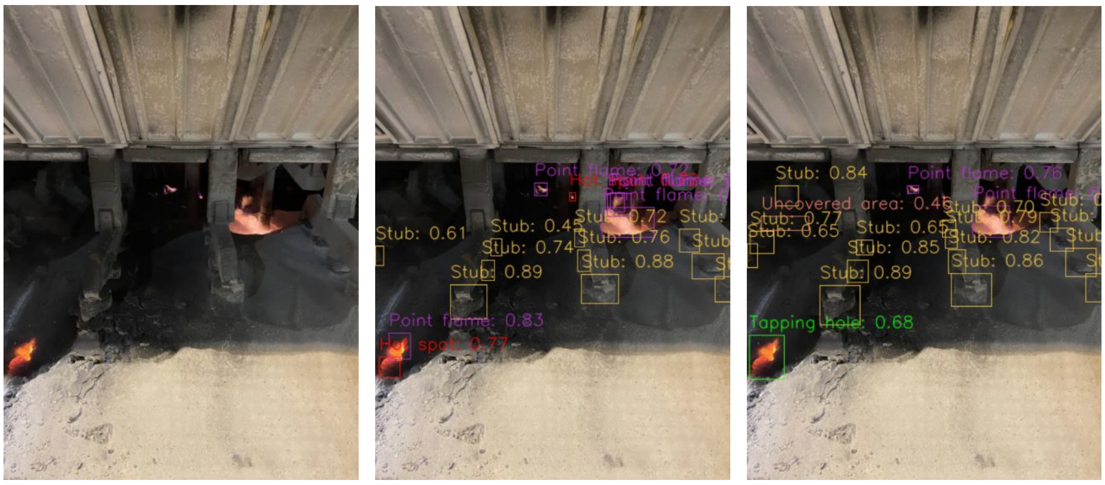
(c) V2.1 Console output:

```
Hot spot: 0.66
Hot spot: 0.76
Hot spot: 0.85
Tapping hole: 0.78
```

Figure 4.3.3: Real-world test for detecting exposed stubs around tapping hole.

Figure 4.3.3 was chosen because it has an open tapping hole with hot spots around it. As mentioned earlier, cells need an opening in the crust for the tapping of aluminium. However, in this instance, the cell has exposed stubs, and if left uncovered, the bath will start to erode the stub, which in turn will make it challenging to meet customer requirements on metal quality [52].

V1.0 failed to detect the hot spots encompassing the tapping hole, in contrast to V2.0 and V2.1 exhibited good probabilities. However, V2.0 found more hot spots around the tapping hole, and the hot spot furthest away from the tapping hole V2.0 got 0.84 compared to V2.1's probability of 0.66.



(a) V1.0 Console output:

```
N/A
```

(b) V2.0 Console output:

```
Point flame:  0.83
Point flame:  0.72
Hot spot:  0.77
Point flame:  0.83
Hot spot:  0.75
Point flame:  0.73
```

(c) V2.1 Console output:

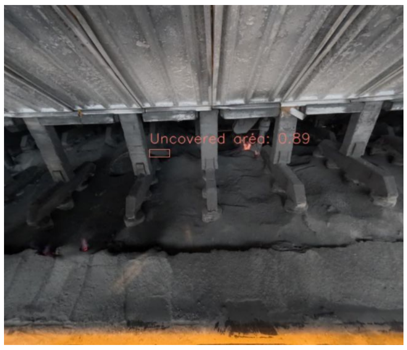
```
Point flame:  0.76
Point flame:  0.80
Uncovered area:  0.46
Tapping hole:  0.68
```

Figure 4.3.4: Real-world test for detecting hidden uncovered area.

Figure 4.3.4 was chosen since it has a barely visible uncovered area. At first glance, V2.1 is noteworthy. It correctly detected the uncovered area with a probability of 0.46 and also detected the tapping hole correctly with a score of 0.68. In contrast, V2.0 incorrectly detected a hot spot with a score of 0.77 and a point flame with a score of 0.83. However, this point flame is stemming from the tapping hole, so one could argue it is correct.

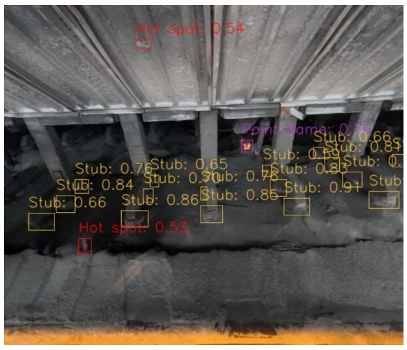
4.3.2 GoPro Images

The following images are all taken by a GoPro HERO11. As mentioned earlier, we suspect the wide focal length is not optimal. As a refresher, V2.0 is not trained on GoPro images.



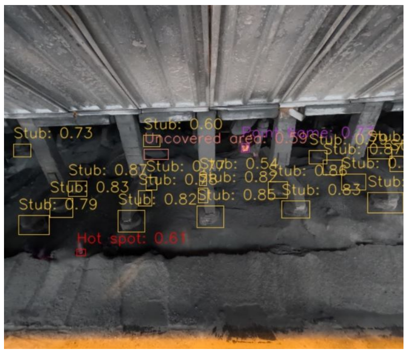
V1.0 Console output:

```
Uncovered area:  0.89
```



V2.0 Console output:

```
Hot spot:  0.54
Hot spot:  0.53
Point flame:  0.76
```

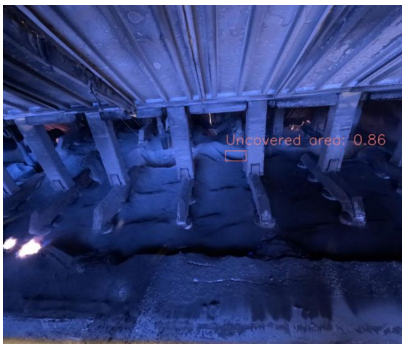


V2.1 Console output:

```
Uncovered area:  0.59
Hot spot:  0.61
```

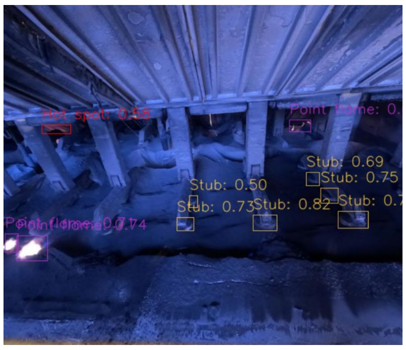
Figure 4.3.5: Real-world test for GoPro's wide focal length.

Figure 4.3.5 was selected because of its uncovered area. V1.0 had the best probability and detected the uncovered area with a score of 0.89. In contrast, V2.0 failed to detect the uncovered area and incorrectly classified a part of the cell cover as a hot spot with a probability of 0.54. For V2.1 it did indeed find the uncovered area but with a lower score, only 0.59. Both V2.0 and V2.1 incorrectly classified a point flame as a hot spot with a score of 0.53 and 0.61, respectively.



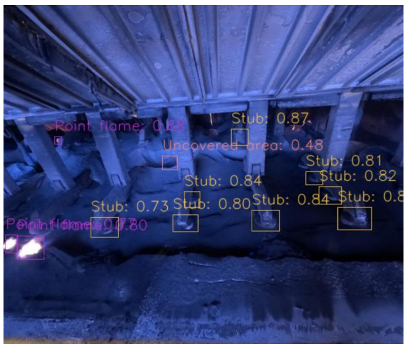
V1.0 Console output:

```
Uncovered area:  0.86
```



V2.0 Console output:

```
Hot spot:  0.58
Point flame:  0.74
Point flame:  0.71
```



V2.1 Console output:

```
Uncovered area:  0.48
Hot spot:  0.61
Point flame:  0.68
Point flame:  0.77
Point flame:  0.80
```

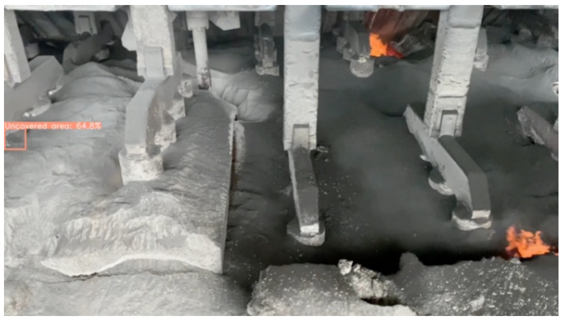
Figure 4.3.6: Real-world test on distorted image.

Figure 4.3.6 was selected because its colour balance is not set correctly and is slightly out of focus. It is essential to know how our algorithm functions in varying lighting conditions and the adverse effects of out-of-focused images, which will be prevalent due to the magnetic fields. However, all vehicles at Hydro Årdal are equipped with spotlights, which makes colour balance issues less likely.

As seen in V1.0, the uncovered area was detected with a high probability, an impressive 0.86. However, the same cannot be said of V2.0. It incorrectly classified illumination on a cell cover, possibly stemming from a tapping hole, as a hot spot. It did, however, classify almost all the point flames. V2.1 did a decent job but did not detect the crucial uncovered area. It detected an uncovered area at the feeding point with a score of 0.48. As mentioned, it is unavoidable to have uncovered areas around the feeding point.

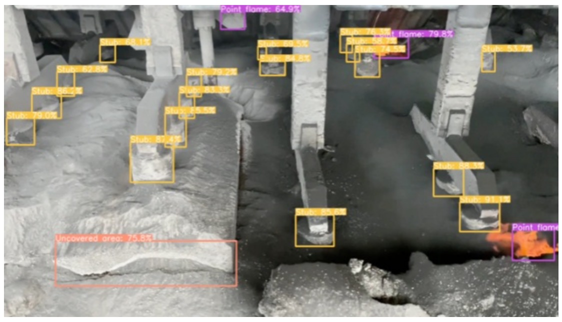
4.3.3 iPhone Video

The following images are screenshots taken from videos from an iPhone, which are also chosen due to their importance in showcasing our results. The V1.0, V2.0, and V2.1 screenshots are taken from approximately the same video frames to highlight the differences between the versions.



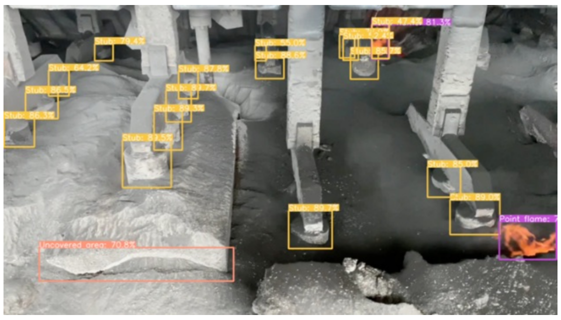
V1.0 Console output:

```
Uncovered area:  68.4%
```



V2.0 Console output:

```
Uncovered area:  74.8%
Point flame:  64.9%
Point flame:  79.8%
Point flame:  75.2%
```



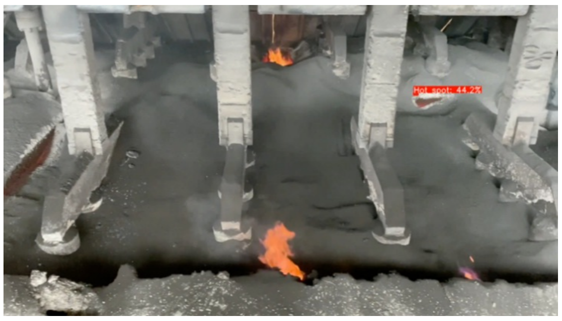
V2.1 Console output:

```
Uncovered area:  70.8%
Point flame:  81.3%
Point flame:  78.3%
```

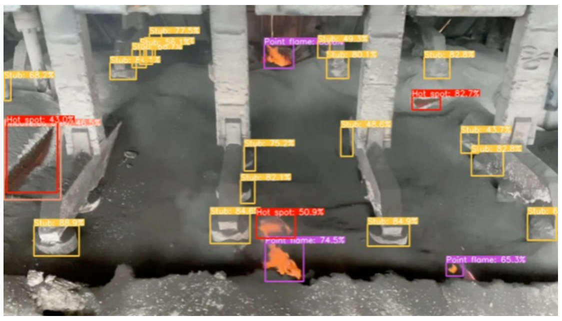
Figure 4.3.7: Real-world test on iPhone video for detecting uncovered areas.

Figure 4.3.7 shows that V1.0 classified the stub to the far left as an uncovered area, with a confidence score of 0.648 (64.8%). V1.0 has a few false detections in this video and struggles to recognise hot spots and uncovered areas in general. The detections in V2.0 are considerably more accurate. However, a part of the alumina feeder (at the top of the image) was misclassified as a point flame. The uncovered area detection to the bottom left in both V2.0 and V2.1 have high confidence scores. In this frame, V2.0 has a confidence score of 0.758 (75.8%), while V2.1 has a confidence score of 0.708 (70.8%). While playing the video, V2.1 has a better overall result regarding this particular uncovered area detection.

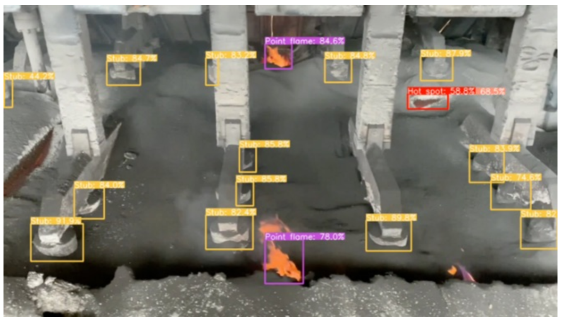
V2.1 also has a slightly better result in this video frame, with no false detections.



V1.0 Console output:
```
Hot spot:  44.2%
```



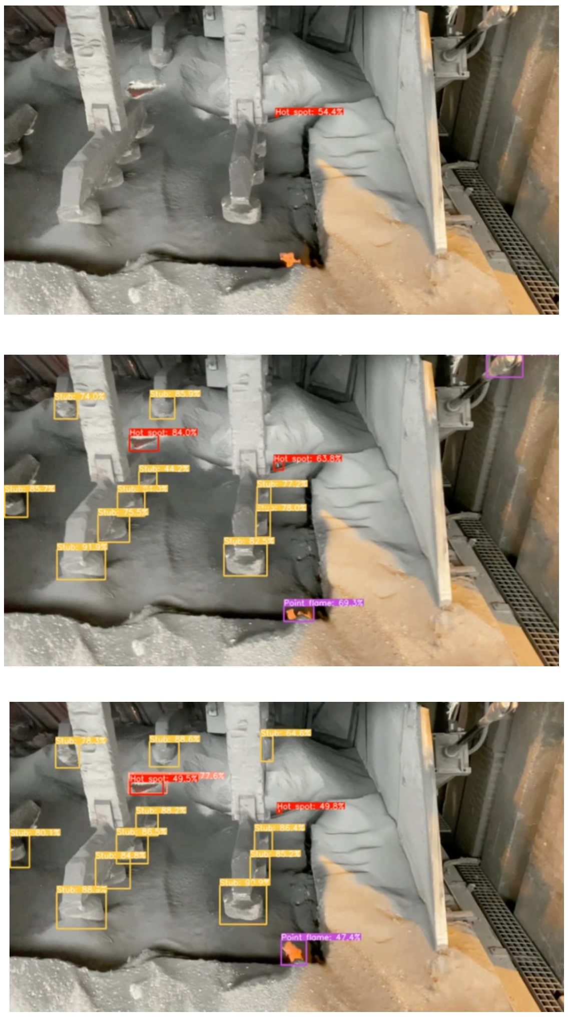
V2.0 Console output:
```
Hot spot:  43.0%
Uncovered area:  46.5%
Point flame:  88.6%
Point flame:  74.5%
Hot spot:  50.9%
Hot spot:  82.7%
Point flame:  65.3%
```



V2.1 Console output:
```
Point flame:  84.6%
Point flame:  78.0%
Hot spot:  58.8%
Uncovered area:  68.5%
```

Figure 4.3.8: Real-world test on iPhone video for detecting hot spots.

Figure 4.3.8 displays that both V1.0 and V2.1 fail to detect the critical hot spot on the left. However, they both detected the smaller, less significant hot spot on the right. V2.0 has the highest confidence score regarding that particular hot spot detection, with a confidence score of 0.827 (82.7%). V2.1 has a confidence score of 0.588 (58.8%) for that same hot spot, as well as a questionable uncovered area detection in the same spot, with a confidence score of 0.685 (68.4%). V2.0 also successfully recognises the hot spot on the left as an uncovered area and hot spot. Nevertheless, V2.0 exhibits a higher frequency of misclassifying point flames as hot spots than V2.1 (as seen in the middle lower part of Figure 4.3.8 V2.0), and the confidence scores of the hot spot and uncovered area detection to the left in V2.0 is quite low; 0.43 (43.0%) for the hot spot, and 0.465 (46.5%) for the uncovered area.



V1.0 Console output:

```
Hot spot:  54.4%
```

V2.0 Console output:

```
Hot spot:  84.0%
Hot spot:  63.8%
Point flame:  69.3%
```

V2.1 Console output:

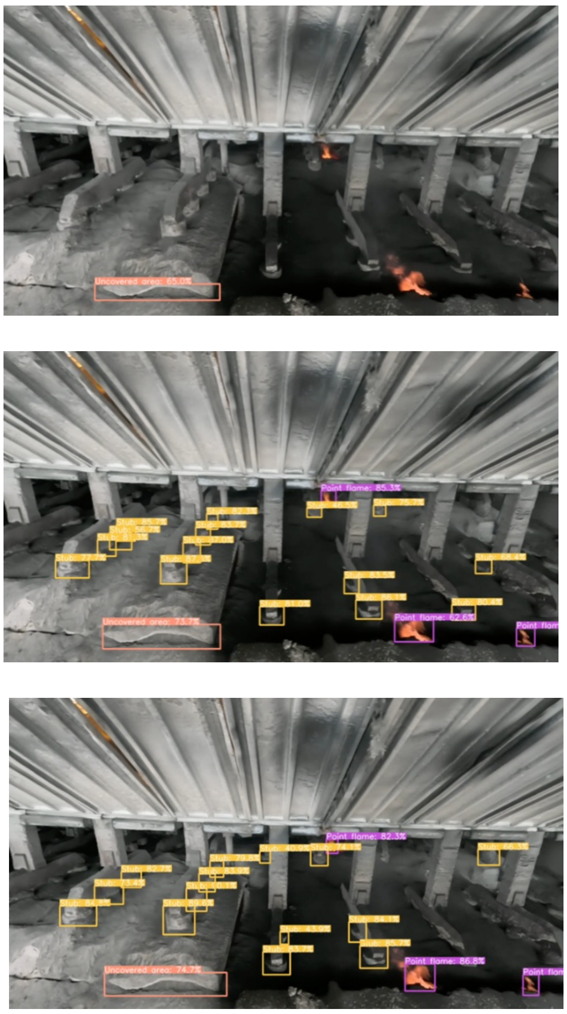
```
Hot spot:  49.5%
Uncovered area:  77.6%
Hot spot:  49.8%
Point flame:  47.4%
```

Figure 4.3.9: Real-world test on iPhone video for detecting hot spots.

Figure 4.3.9 demonstrates that V1.0 detects merely a small hot spot and fails to accurately detect the hot spot located to the upper left of the image. V2.0 and V2.1 perform well in this instance, with no false detections and high confidence scores in general. V2.0 has the highest confidence score for both the hot spot detections, with 0.84 (84%) and 0.638 (63.8%). While V2.1 has confidence scores of 0.495 (49.5%) and 0.498 (49.8%) for the same hot spots. V2.1 classifies the hot spot to the left as both a hot spot and an uncovered area in this instance as well.

4.3.4 GoPro Video

This test video, filmed with a GoPro HERO 11, is from the same cell as the iPhone video in the previous section. These images are also screenshots from approximately the same video frame, extracted as still images, showcasing the differences between the versions.



V1.0 Console output:

```
Uncovered area:  65.0%
```

V2.0 Console output:

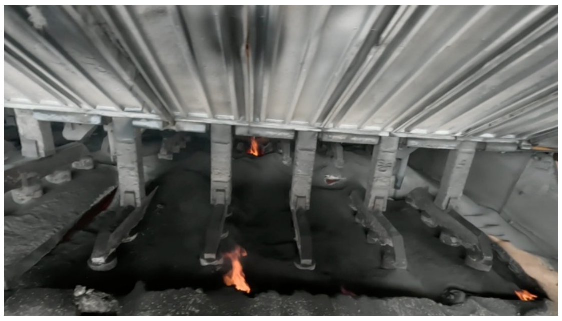
```
Uncovered area:  73.7%
Point flame:  85.3%
Point flame:  62.6%
Point flame:  71.8%
```

V2.1 Console output:

```
Uncovered area:  74.7%
Point flame:  82.3%
Point flame:  86.8%
Point flame:  78.8%
```

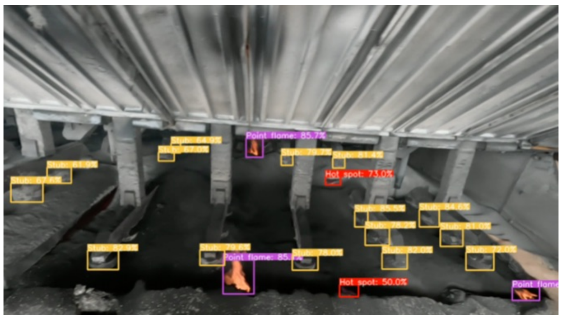
Figure 4.3.10: Real-world test on GoPro video for detecting uncovered areas.

Figure 4.3.10 displays that V1.0 successfully detects the uncovered area in the bottom left, with a reasonable confidence score of 0.65 (65%). V2.0 and V2.1 also detect that same uncovered area, with V2.0 comprising a confidence score of 0.737 (73.7%) and V2.1 having 0.747 (74.7%). There is minimal difference in the results of V2.0 and V2.1 in this video frame, except for the confidence scores being slightly higher overall in V2.1. In addition, V2.1 detects a higher number of stubs, although they both fail to detect every single stub.



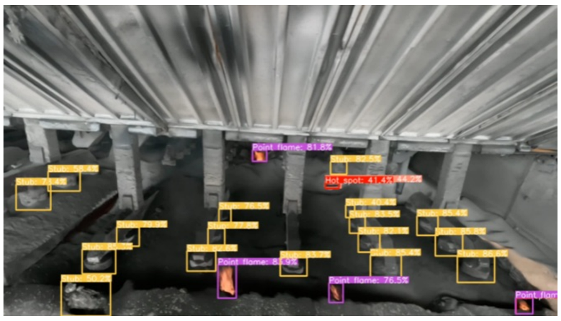
V1.0 Console output:

```
N/A
```



V2.0 Console output:

```
Point flame:  85.3%
Point flame:  85.7%
Hot spot:  73.0%
Hot spot:  50.0%
Point flame:  65.3%
```



V2.1 Console output:

```
Point flame:  81.8%
Point flame:  83.9%
Hot spot:  41.4%
Uncovered area:  44.2%
Point flame:  76.5%
Point flame:  72.5%
```

Figure 4.3.11: Real-world test on GoPro video for detecting hot spots.

Figure 4.3.11 show that V1.0 does not detect anything in this instance, although a hot spot is located slightly to the right in the middle of the image. V2.0 and V2.1 successfully detect that hot spot, where V2.0 has the highest confidence score of 0.73 (73.0%), while V2.1 has a combined hot spot and uncovered area detection with confidence scores of 0.414 (41.4%) and 0.442 (44.2%). V2.0 classified one of the point flames at the bottom of the image as a hot spot, while V2.1 classified the point flame correctly with a confidence score of 0.765 (76.5%). Apart from this, V2.0 and V2.1 performed rather similarly.

CHAPTER

FIVE

DISCUSSION

5.1 Results from Solution One and Two

The results obtained from both Solution One and Solution Two exhibit varying levels of success in detecting hot spots and uncovered areas.

Solution One, which relied on a colour and edge-detecting model, demonstrated reasonable success in identifying hot spots. However, the method's high sensitivity to lighting conditions and inability to distinguish between hot spots and point flames limit its practical application. Additionally, the edge detection method for identifying uncovered areas failed to produce reliable results due to interfering edge features inside the cell.

In contrast, despite their limited training data set, the versions obtained in Solution Two worked better than expected. In the following subsections, the results from each version will be discussed.

5.1.1 Version 1.0

The object detection algorithm V1.0 exhibits a relatively high level of accuracy when applied to test images. However, it did not perform well when tested on video. V1.0 was able to identify some objects in different lighting conditions, but its performance was not as precise as in later versions.

V1.0's performance varies throughout the test cases. While tested on images, V1.0 sometimes outperformed the other versions in detecting uncovered areas or hot spots. However, it struggled to provide accurate results consistently.

The confidence scores for detected objects were generally low when tested on video, with inconsistent results and several false detections. V1.0 was trained using 111 images from both the iPhone and the GoPro, so the poor results on video files were surprising.

In summary, V1.0's performance was satisfactory when tested on images but could not perform while being tested on video. By only being able to recognise hot spots and uncovered areas, V1.0 was seemingly more receptive to false detections. However, it provided a foundation for developing V2.0 and V2.1.

5.1.2 Version 2.0

The object detection algorithm V2.0 demonstrates a high degree of accuracy when applied to test videos captured using both a GoPro HERO11 and an iPhone 11. V2.0 could accurately detect objects in various lighting conditions, with almost no false detections. It successfully detected a high number of objects, consistently exhibiting relatively high confidence scores.

One notable feature of V2.0 is its ability to separate objects in crowded images, even when objects are partially obscured. This is particularly impressive given that the number of images used in training the object detection algorithm was relatively low (68 images in total).

Although instances of false detections were observed, they were infrequent and typically exhibited low confidence scores. The most common occurrence of false detections was point flames being misclassified as hot spots. These false detections often last only a few frames, but they can still be problematic and attention-stealing when they occur consistently.

Overall, V2.0 performed surprisingly well, given its relatively low amount of training data. Its ability to recognise and separate objects in crowded images is particularly noteworthy. However, the biggest weakness of V2.0 is false detections, especially in the case of point flames being misclassified as hot spots.

5.1.3 Version 2.1

V2.1 performs well in both video and image testing, with almost non-existent false detections. While being tested on iPhone and GoPro video, successful object detections often hold high confidence scores in varying lighting conditions. However, in some instances, V2.1 classify hot spots as both a hot spot and an uncovered area.

V2.1 is accurate when tested on video files; few objects go unseen, and false detections are rare. Being trained on 111 images might contribute to its ability to detect objects even though they are obscured or have bad lighting. However, as seen in the test videos, V2.1 sometimes struggles to detect certain hot spots and uncovered areas instantly. Yet, except for the case in Figure 4.3.8, hot spots and uncovered areas are eventually detected.

V2.1's most significant weakness is the occasional classification of hot spots as hot spot and uncovered area. Although these combined classifications might be considered a drawback, hot spots are inherently uncovered areas; thus, these detections can be argued as not being false.

V2.1 demonstrates a consistently high degree of accuracy, with minimal false detections. Few objects go undetected. Besides the combined hot spot and uncovered area detections and the occasionally uncovered area initially going unseen, V2.1 is robust and reliable.

5.1.4 Comparing Versions

In video testing, V1.0 failed to detect anything of interest except for brief hot spot detections and uncovered area detections. V1.0 also exhibits numerous false detections. In contrast, V2.0 and V2.1 both performed well. However, V2.0 is more likely to misclassify point flames as hot spots, even though such false detections often only last a few frames. V2.1 more frequently classify hot spots as both a hot spot and an uncovered area.

Nevertheless, since a hot spot is inherently an uncovered area, it can be argued that these types of detections are not false. Seemingly, V2.0 and V2.1 each possess their advantages and drawbacks. V2.0 detects hot spots more frequently, and V2.1 is more accurate in detections, with fewer false detections and a reduced likelihood of misclassifying point flames as hot spots.

Even though V2.1 has combined hot spot and uncovered area detections, rather than V2.0's single hot spot detections with higher confidence scores, V2.1 seems sturdier with consistently fewer false detections. Both V2.0 and V2.1 handle the wide angle from the GoPro well. However, V2.1 has no false detections outside the cell. In contrast, V2.0 has occasional false detections, such as red paint being misclassified as hot spots and other equipment misclassified as point flames or uncovered areas.

The confusion matrices in Chapter 4 reveal that our image dataset scarcely included images without objects, also known as *background* images. If the model has not been trained with such images, it could struggle to identify them when presented accurately. The fact that we did not specifically train the versions with verified backgrounds (except for a handful in V1.0) might contribute to the observed misclassifications.

V1.0 has some instances of successful detections, but it generally performs poorly in test videos. V2.0 and V2.1 are the more comparable versions, each with its own advantages and drawbacks. However, V2.1 is generally more reliable, with fewer false detections and typically accurate detections.

Table 5.1.1 gives an overview of what type of images the various versions were trained on, the number of images, and the labelling tools used.

Version	GoPro images	iPhone images	Total	MakeSense	Roboflow
Version 1.0	43	68	111		✓
Version 2.0	0	68	68	✓	
Version 2.1	43	68	111		✓

Table 5.1.1: Overview of the quantity and types of images, as well as the labelling tools used in training the various ML-models.

5.1.5 Wide-Angle Impact on Detection from the GoPro

Potential issues were initially suspected due to the wide-angle focal length of the GoPro HERO11. It could complicate the object detection algorithms' ability to discern between different objects. However, these concerns were largely proven unfounded following further tests.

Although the GoPro's wide angle is not detrimental, its footage generally appears darker. This could diminish the accuracy of hot spot detection, resulting in some hot spots being missed. The darker footage may be attributed to effects such as vignetting, a common phenomenon with wide focal lengths. This could also be due to a less light-sensitive sensor.

5.1.6 Latency Issues and GPU Requirements

The object detection algorithms V1.0, V2.0, and V2.1 were executed and tested on video files within two different environments: Google Colab, which used a powerful GPU (often a Tesla T4 GPU) [53], and Python scripts running locally on a personal computer.

Both environments produced identical results. However, the locally executed script was more prone to latency issues, which resulted in a slower FPS rate when running the videos live. These latency issues can largely be attributed to the limited CPU capabilities of the computer running the script locally. As a point of note, the training phase of machine learning models demands powerful GPUs. However, when deploying pre-trained models, the requirements are not as demanding, and a less powerful GPU can suffice.

Figure 5.1.1: The budget-friendly Nvidia GeForce RTX 2070 GPU [54].

Dedicated Graphics Processing Units (GPUs) can mitigate latency issues and enhance performance, compared to Central Processing Units (CPUs). Economical GPUs, such as the NVIDIA Jetson series, offer a beneficial balance between cost and performance. They are specifically designed to handle ML tasks [55]. Alternatively, a cost-effective solution is to use an affordable consumer gaming computer equipped with a modestly priced GPU, like the *Nvidia GeForce RTX 2070* (see Figure 5.1.1). Despite being less powerful than its counterpart used in the training phase, this GPU has the necessary 8 GB of memory to run pre-trained ML operations effectively. As of 2023, it can be acquired for around 6500 NOK, a relatively small cost considering the potential savings highlighted in Appendix D.

5.2 Potential Solutions

In addition to finding the correct method for detecting hot spots and other objects on the anode, we must discuss how to implement the system.

5.2.1 Corrosion Protection

The environment at Hydro Årdal is, as mentioned, harsh. This includes corrosive dust and gases such as hydrogen fluoride (HF) and aluminium oxide (Al_2O_3). These substances, as mentioned, severely corrode any form of glass, making it essential to find ways to protect the lens to maintain decent image quality.

At Hydro Årdal, different protective measures are already in place to minimise corrosion. For example, vehicles have replaceable films on their windows to prolong the glass's lifespan. Moreover, some flue gas sensors use a compressor that continuously generates an overpressure at the lens of the laser beam, reducing lens corrosion [56].

Our study suggests using cost-effective glass filters or cases that could be replaced during regular service intervals as the most viable solution. This approach has multiple benefits, such as being easy to implement, simple in design, and having few potential failure points. However, it is important to consider that the protective glass may have a limited lifespan, potentially lasting only one or two shifts. This is similar to the initial performance of laser beam lenses before introducing the overpressure method, which only yielded reliable measurements for a few hours [56].

Given the challenges associated with movement, applying an overpressure system on a moving vehicle might not be as effective as for stationary flue gas sensors. Factors such as changes in pressure due to varying speeds or external environmental influences could impact the system's performance.

5.2.2 Real-Time Detection

Providing live feedback during the anode covering operation offers immediate, precise responses, although it introduces certain challenges. Initially, real-time feedback might increase the operator's workload, potentially posing a challenge to covering all the anodes within a single shift. Therefore, it becomes essential to define tolerances and ascertain the criticality of hot spots to prevent the operator from being overburdened.

Secondly, real-time feedback necessitates an enhanced processing capacity for immediate image analysis and suggestions. This requirement considers the need for more powerful hardware, such as dedicated Graphics Processing Units (GPUs), which excel in image processing tasks compared to Central Processing Units (CPUs), as discussed in Section 5.1.6.

5.2.3 Cloud-Based Prioritisation

At Hydro Årdal, each operator in the electrolysis area is tasked with redressing a designated number of cells. In a cloud-based system, these operators would receive prioritisation notifications for their cells that require redressing. This system could also initiate tasks for the upcoming shift. For instance, if a cell lacks sufficient ACM, the covering operator can refer to a list of cells that need additional ACM.

A cloud-based system offers numerous benefits, such as image processing in a secure and centralised location and keeping hardware safe from the hazardous environments found in Hydro Årdal. Additionally, it can monitor conditions and provide more reliable documentation than random checks. This could assist quality control, which is currently conducted through manual visualisation checks [57].

However, this system also presents some drawbacks. To be a viable solution, the vehicle on which the system is mounted must maintain a stable wireless connection to the database, enabling the transfer of images for processing. Furthermore, the wireless connection must possess sufficient bandwidth to transmit data wirelessly, which could pose a challenge.

If establishing a stable wireless connection proves too difficult, an alternative approach would be physically transferring the recordings using the memory card or via cable when the covering vehicle refills the ACM. However, we hypothesise that a physical transfer solution might become more of a burden than an aid for the operators, as it could lead to the loss of memory cards.

A potential solution to address the mentioned challenges could involve implementing a robust and reliable wireless communication system, such as a dedicated private network or 5G infrastructure. This can ensure stable connections and sufficient bandwidth for data transfer.

5.2.4 Vehicle Considerations

While the anode-covering vehicle is an obvious choice for implementing our system, several limitations must be addressed. Large quantities of dust are generated during the anode covering operation, which we hypothesised can pose challenges to the vision system's accuracy. As our system has only been tested under ideal conditions with controlled illumination and lighting sources, real-world scenarios with varying levels of dust and debris may lead to decreased performance.

An alternative approach would be to integrate the system with the *crust-breaking operations*. In this phase, the used anode, or *butt*, is broken free from the crust by a crust breaker before the anode is replaced. This process requires the entire cell cover to be opened, providing a clear view of the cell for the system to scan, see Figure 5.2.1. By mounting the system on the crust breaker, it can acquire images of the entire cell, which can then be relayed to a centralised database for further processing.

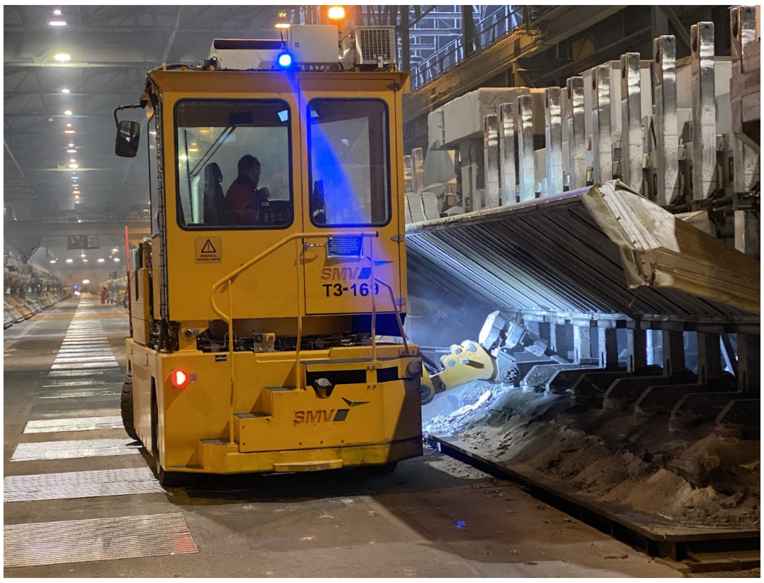


Figure 5.2.1: A crust breaker chiselling out a butt.

This approach offers several advantages. Firstly, it allows the system to gather information on the cell's condition before the covering operation, enabling operators to prioritise cells that require immediate attention.

Secondly, the system may exhibit improved performance and accuracy as the crust breaker operates in a relatively dust-free environment compared to the anode covering process. However, integrating the system with the crust breaker introduces new challenges. The system's mounting, stability, and protection from potential damage due to the crust breaker's operation must be considered. Which consist of the vibration caused by the chiselling of the crust. These vibrations could inhibit the system's image quality, making it less accurate. Despite these challenges, integrating the vision system on the crust breaker presents a promising solution.

5.3 Further Work

In this section, we suggest the potential for further work, which includes three phases. Each phase builds upon the previous one, laying the groundwork for the next step while addressing new challenges and expanding the scope of the project.

5.3.1 Phase I: Refining the Algorithm

Upon establishing that the POC is viable under ideal conditions, it is necessary to expand the dataset. To achieve a robust algorithm, it is advisable to have more images. However, it is difficult to determine the number of images since it depends on the complexity of the task. Our findings indicate that a focal length of approximately 26 mm is optimal, with a resolution of 640 pixels is sufficient. Expanding the training set is facilitated by filming, and the tedious annotation task could possibly be automated. To achieve this, the video feed would first undergo *inference*, which refers to making predictions on new unseen data. The processed output would then be fed back into Roboflow, which subsequently divides the video into a specified number of still images. After the split, these images would be manually reviewed to determine which among them would be selected for further automated training. After collecting sufficient data, we recommend finalising the height measurement for the ACM. A stereo camera is suggested for improved accuracy, but one could continue using a single-sensor camera, albeit with potential height inaccuracies for the ACM. Furthermore, develop a scoring system to evaluate ACM coverage and instances of exposed anode surfaces, operators with intuitive feedback.

5.3.2 Phase II: Implementation

The system's implementation presents the most significant challenge. As previously mentioned, our tests have been conducted under ideal conditions.

We initially propose implementing the system on the covering vehicle, as image acquisition and algorithm testing do not require powerful hardware on board the vehicle. Data can be gathered and analysed elsewhere. The crust breaker solution may be considered if dust generated during the covering process hinders image clarity. It currently needs to be determined whether cloud-based or real-time feedback is preferable; both options have their merits and drawbacks, as discussed in Section 5.2. We advise making this decision after completing Phase I.

5.3.3 Phase III: Autonomous Covering Crane

Suppose Phase I and II are successfully implemented. In that case, it can be used in the final goal, which is to integrate a vision system into an autonomous robotic crane that automatically covers anodes. A monocular camera could be adequate for this application based on the crane operating along a predefined track. The distance between the anode and the camera frame can be calculated using inverse and forward kinematics to transform coordinate frames. According to our hypothesis, a monocular camera should offer sufficient performance, thus negating the need for a 3D camera in this context. For a comprehensive understanding of robotic vision techniques and principles, refer to the book "Robotics, Vision and Control" by Peter Cork [58].

CHAPTER

SIX

CONCLUSIONS

This project aimed to develop a POC for a CV system to detect exposed anode surfaces at aluminium smelters. Two different solutions were explored: Solution One, a colour and edge-detecting model, and Solution Two, a ML object detection algorithm.

Solution One showed potential in detecting hot spots but struggled with distinguishing between hot spots and other red-coloured objects in the cell, i.e. point flames and tapping holes. Moreover, it faced challenges in identifying uncovered areas due to interfering edge features inside the cell. On the other hand, Solution Two presented three versions of the object detection algorithm. V1.0 showed satisfactory results on images, but its video performance was subpar. V2.0 demonstrated a fairly high degree of accuracy but had some issues with false detections, such as point flames being misclassified as hot spots. V2.1 performed the best, with minimal false detections and consistent accuracy in detecting hot spots and uncovered areas.

The wide-angle focal length of the GoPro HERO11 camera did not pose significant challenges to object detection. Our results showed that a resolution of 640 is sufficient at a distance of 50 cm with a focal length of 26 mm. However, dimmer footage from the GoPro could lead to less accurate detections of hot spots. Latency issues were observed when running object detection algorithms on a CPU. However, no latency issues were encountered when executing these pre-trained models on a dedicated GPU. This means it is advantageous to have a dedicated GPU for real-time detection.

Various potential solutions were discussed, including corrosion protection for the camera, real-time detection using dedicated GPU, cloud-based prioritisation, and vehicle considerations. These solutions highlighted the importance of balancing system performance, practicality, and operator workload. Ultimately, integrating the system with crust breaker operations emerged as a promising approach, providing a clearer view of the cell and possibly more efficient allocation of resources.

In conclusion, this project demonstrates the potential of machine learning and computer vision in detecting exposed anode surfaces within aluminium-producing cells. The further development and implementation of our proposed system could reduce emissions, enhance anode-covering quality, and thereby yield economic benefits for Hydro Årdal. As Appendix D indicates, potential savings could amount to 14.5 million NOK annually. These financial savings and reduction in CO_2 emissions can enhance Hydro Årdal's profitability and support a more sustainable future for aluminium production.

REFERENCES

[1] Enova. *Skal produsere aluminium utslippsfritt.* Accessed: 2023-04-12. 2021. URL: https://kommunikasjon.ntb.no/pressemelding/skal-produsere-aluminium - utslippsfritt ? publisherId = 17848299 & releaseId = 17958182.

[2] Norwegian Ministry of Climate and Environment. *Norway's Climate Action Plan for 2021–2030.* https : / / www . regjeringen . no / contentassets / a78ecf5ad2344fa5ae4a394412ef8975 / en - gb / pdfs / stm202020210013000engpdfs.pdf. p. 32. 2020.

[3] Norsk Hydro. *Climate.* https://www.hydro.com/en/sustainability/our-approach/environmental/climate/. 2022.

[4] Norsk Hydro. *PotRoom.* Accessed: February 13, 2023. 2022. URL: https://www.hydro.com/en/media/news/2022/hydro-investerer-80-millioner-i-ny-produksjonsteknologi-i-ardal/.

[5] Simen Vatslid Øystese. *CO2 Strategi Hydro og Elektrolysen i Årdal.* Unpublished PowerPoint Presentation. slide 10. 2022.

[6] Simen Vatslid Øystese. *CO2 Strategi Hydro og Elektrolysen i Årdal.* Unpublished PowerPoint Presentation. slide 13. 2022.

[7] Pachauri, R. K. and Meyer, L. A. *Climate Change 2014: Synthesis Report. Contribution of Working Groups I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change.* https://www.ipcc.ch/site/assets/uploads/2018/05/SYR_AR5_FINAL_full_wcover.pdf. pp. 13-16. Geneva, Switzerland, 2014.

[8] W. Haupin. "Electrochemistry of the Hall-Heroult Process for Aluminum Smelting". In: *Journal of Chemical Education* 60.4 (1983), p. 279. DOI: 10.1021/ED060P279. URL: https://doi.org/10.1021/ED060P279.

[9] *Hovedkurs i videreutdanning Fagoperatør Aluminium.* Course material. pp. 3, 13. Oslo, Norway, 2006.

[10] Carl H Hamann, Andrew Hamnett, and Wolf Vielstich. *Electrochemistry.* Wiley-VCH, 2007. Chap. 3, pp. 57–60. ISBN: 9783527310692.

[11] *AA 03 04 03 SOP Dekking.* Unpublished standard operating procedure. Developed by Elin Haugland. Approved by Simen Vatslid Øystese. 2022.

[12] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* 2nd. New York, NY: Springer, 2017, p. 287. ISBN: 978-3-319-54413-7. DOI: 10.1007/978-3-319-54413-7.

[13] Trond Eirik Jentoftsen. *Personal communication: Price of anode mass.* Personal communication with Eskil Digernes. 2023.

[14] Junichi Nakamura. *Image Sensors and Signal Processing for Digital Still Cameras.* CRC Press, 2005. ISBN: 978-0-8493-3545-7.

[15] Silent Peak Photography. *How Do Camera Imaging Sensors Work?* Accessed: February 6, 2023. 2021. URL: https://silentpeakphoto.com/gear/cameras/camera-guides/how-do-camera-imaging-sensors-work/.

[16] Various. *The Three Primary Colors of RGB Color Model (Red, Green, Blue).* Online. Accessed on March 2, 2023. 2019. URL: https://en.wiktionary.org/wiki/RGB#/media/File:The_three_primary_colors_of_RGB_Color_Model_(Red,_Green,_Blue).png.

[17] Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997. ISBN: 0070428077.

[18] Alon Halevy, Peter Norvig, and Fernando Pereira. "The unreasonable effectiveness of data". In: *IEEE Intelligent Systems* 24.2 (2009), pp. 8–12.

[19] Sebastian Raschka. "Model evaluation, model selection, and algorithm selection in machine learning". In: *arXiv preprint arXiv:1811.12808* (2018).

[20] AI Graduate. *Use of Cross-Validation in Machine Learning.* https://aigraduate.com/use-of-cross-validation-in-machine-learning/. Accessed: 2023-05-10. 2021.

[21] Douglas M Hawkins. "The problem of overfitting". In: *Journal of chemical information and computer sciences* 44.1 (2004), pp. 1–12.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* Cambridge, MA: MIT Press, 2016. ISBN: 978-0262035613. URL: https://www.deeplearningbook.org/front_matter.pdf.

[23] Lutz Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade* (1998), pp. 55–69.

[24] Towards Data Science. *Parameters and Hyperparameters.* Year. URL: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[26] PyTorch Team. *torch.nn.Conv2d.* 2021. URL: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

[27] Janosh Riebesell. *Convolution Operator.* 2022. URL: https://tikz.net/conv2d/.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Advances in neural information processing systems.* 2012, pp. 1097–1105. DOI: 10.1145/3065386.

[29] PyTorch Team. *torch.nn.ReLU.* 2021. URL: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html.

[30] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[31] PyTorch Team. *Defining a Neural Network in PyTorch.* 2021. URL: https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html.

[32] PyTorch Team. *torch.nn.Softmax.* 2021. URL: https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html.

[33] A. El Gamal and B. Fowler. "CMOS vs. CCD Imagers". In: *Handbook of Image and Video Processing.* Ed. by A.C. Bovik. 2nd ed. Elsevier Academic Press, 2005, pp. 83–98. ISBN: 9780121197926.

[34] Simen Vatslid Øystese. *Personal communication: The use of GoPros with vehicles at Hydro Årdal.* Personal communication with Eskil Digernes. Apr. 2023.

[35] ClearView Imaging. *Stereo Vision for 3D Machine Vision Applications.* https://www.clearview-imaging.com/en/blog/stereo-vision-for-3d-machine-vision-applications. [Online; accessed 19-April-2023]. 2023.

[36] P. Polakis and A. Karakos. "RGB and HSV color models for image analysis: A survey". In: *2014 International Conference on Telecommunications and Multimedia (TEMU).* IEEE. 2014, pp. 1–6. DOI: 10.1109/TEMU.2014.6955153.

[37] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall, 2001. ISBN: 978-0-13-030796-5.

[38] Richard Szeliski. *Computer Vision: Algorithms and Applications*. London, UK: Springer, 2010. ISBN: 978-1-84882-934-3.

[39] R. J. Cintra et al. “Color-based segmentation for forest fire detection: A comparative study”. In: *Journal of the Brazilian Computer Society* 12.3 (2007), pp. 7–16.

[40] Yuanyuan Ma and Ognjen Arandjelović. “Classification of Ancient Roman Coins by Denomination Using Colour, a Forgotten Feature in Automatic Ancient Coin Analysis”. In: *Sci* 2.2 (2020), p. 37. DOI: 10.3390/sci2020037. URL: https://www.mdpi.com/2413-4155/2/2/37.

[41] Murtaza Hassan. *OpenCV Python Tutorial For Beginners 1 - Introduction to OpenCV*. Chapter 7: Color Detection. Murtaza’s Workshop - Robotics and AI. 2021. URL: https://www.youtube.com/watch?v=WQeoO7MI0Bs&ab_channel=Murtaza/27sWorkshop-RoboticsandAI.

[42] Glenn Jocher. “YOLOv5: Improved Performance, Faster Speed”. In: (2021). URL: https://blog.roboflow.com/yolov5-improvements-and-speed/.

[43] J. Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/pdf/1506.02640.pdf.

[44] MakeSense. *MakeSense*. https://www.makesense.ai. Accessed: April 24, 2023.

[45] Roboflow. *Roboflow*. https://www.roboflow.com. Accessed: April 24, 2023.

[46] Priya Goyal et al. “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”. In: *arXiv preprint arXiv:1706.02677* (2017).

[47] *NVIDIA Tesla T4 Specs | TechPowerUp GPU Database*. Accessed: 2023-05-18. 2023. URL: https://www.techpowerup.com/gpu-specs/tesla-t4.c3316.

[48] Daniel L. Schwerin and Jason D. Hatcher. “Hydrofluoric Acid Burns”. In: *StatPearls [Internet]* (Jan. 2023). Last Update: March 7, 2023. URL: https://www.ncbi.nlm.nih.gov/books/NBK441829/.

[49] DeepLearning. *YOLOv5 training with custom data*. Accessed: 2023-02-23. YouTube. Dec. 2020. URL: https://www.youtube.com/watch?v=GRtgLlwxpc4&t=1072s.

[50] Takaya Saito and Marc Rehmsmeier. “The Precision-Recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets”. In: *PloS one* 10.3 (2015), e0118432. DOI: 10.1371/journal.pone.0118432.

[51] Analytics India Magazine. *Complete Guide to Understanding Precision and Recall Curves.* Accessed on Date of access. Year of publication. URL: https://analyticsindiamag.com/complete-guide-to-understanding-precision-and-recall-curves/.

[52] Simen Vatslid Øystese. *Personal communication: Iron content in aluminium.* Personal communication with Eskil Digernes. 2023.

[53] Tutorials Point. *Google Colab - Using Free GPU.* https://www.tutorialspoint.com/google_colab/google_colab_using_free_gpu.htm. 2021.

[54] *Nvidia GeForce RTX 2070.* Accessed: 2023-05-20. URL: https://prisguiden.no/produkt/nvidia-geforce-rtx-2070-337182.

[55] *NVIDIA Jetson: Hardware for Machine Learning Tasks.* https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. Accessed: April 27, 2023.

[56] Simen Vatslid Øystese. *Personal communication: Use of overpressure to protect laserbeam lens from corrosive HF gas.* Personal communication with Eskil Digernes. 2023.

[57] Simen Vatslid Øystese. *Personal communication: Quality control of anode covering.* Personal communication with Eskil Digernes. 2023.

[58] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* 2nd. New York, NY: Springer, 2017. ISBN: 978-3-319-54413-7. DOI: 10.1007/978-3-319-54413-7.

[59] Nordnet. *Norsk Hydro (NHY).* Online. 2020. URL: https://www.nordnet.no/market/stocks/16105667-norsk-hydro?details.

[60] Wikipedia. *Norsk Hydro logo.* Online. URL: https://no.wikipedia.org/wiki/Norsk_Hydro#/media/Fil:Norsk_Hydro.svg.

[61] Hydro. *Hydro Aluminium AS Årdal.* Online. 2020. URL: https://www.hydro.com/no-NO/om-hydro/hydro-locations-worldwide/europe/norway/ardal/hydro-aluminium-as-ardal/.

APPENDICES

A - GITHUB AND GOOGLE LINKS

The GitHub links contain all the Python scripts and our modified YOLOv5. The Google links contain all the videos and images.

GitHub repository links

- `https://github.com/eskilDigernes/BO23EB-08.git`
- `https://github.com/eskilDigernes/yolov5`

Google Drive

- `https://drive.google.com/drive/folders/1nOwRtTs7T5r0G1gH5H4RB3v8b6mdj2Rh`

Google Colaboratory

- `https://colab.research.google.com/drive/1G9E1bBhzeM191Yu0eaMa0o25EpD22J3K#scrollTo=eaFNnxLJbq4J`

B - PYTHON SCRIPTS

B1 - Google Colaboratory

```
# -*- coding: utf-8 -*-
"""Yolov5_training.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/
    1G9E1bBhzeM191Yu0eaMa0o25EpD22J3K

# Custom Training with YOLOv5

In this tutorial, we assemble a dataset and train a custom
YOLOv5 model to recognize the objects in our dataset.
To do so we will take the following steps:

* Gather a dataset of images and label our dataset
* Export our dataset to YOLOv5
* Train YOLOv5 to recognize the objects in our dataset
* Evaluate our YOLOv5 model's performance
* Run test inference to view our model at work



![](https://uploads-ssl.webflow.com/5f6bc60e665f54545a1e52a5/
615627e5824c9c6195abfda9_computer-vision-cycle.png)

# Step 1: Install Requirements
"""
```

```
# Commented out IPython magic to ensure Python compatibility.
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
# %cd yolov5
# %pip install -qr requirements.txt # install dependencies
# %pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to
display images

print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if
torch.cuda.is_available() else 'CPU'})")

"""# Step 2: Assemble Our Dataset

In order to train our custom model, we need to assemble a
dataset of representative images with bounding box
annotations around the objects that we want to detect.
And we need our dataset to be in YOLOv5 format.

In Roboflow, you can choose between two paths:

* Convert an existing dataset to YOLOv5 format.
Roboflow supports over [30 formats object detection formats]
(https://roboflow.com/formats) for conversion.
* Upload raw images and annotate them in Roboflow with
[Roboflow Annotate](https://docs.roboflow.com/annotate).

# Annotate

![](https://roboflow-darknet.s3.us-east-2.amazonaws.com/
roboflow-annotate.gif)

# Version

![](https://roboflow-darknet.s3.us-east-2.amazonaws.com/
robolfow-preprocessing.png)
```

```
"""

# set up environment
os.environ["DATASET_DIRECTORY"] = "/content/datasets"

#Replace these lines with lines from your own Roboflow project

from roboflow import Roboflow
rf = Roboflow(api_key="L6PC4NZ0Yj3c6sKKprQ0")
project = rf.workspace("object-detection-5csc8").
project("black_red_detection")
dataset = project.version(1).download("yolov5")

"""# Step 3: Train Our Custom YOLOv5 model

Here, we are able to pass a number of arguments:
- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs.
(Note: often, 3000+ are common here!)
- **data:** Our dataset locaiton is saved in the
‘dataset.location‘
- **weights:** specify a path to weights to start transfer
learning from.
Here we choose the generic COCO pretrained checkpoint.
- **cache:** cache images for faster training
"""

!python train.py --img 640 --batch 16 --epochs 1000 --data
{dataset.location}/data.yaml --weights yolov5s.pt --cache

"""# Evaluate Custom YOLOv5 Detector Performance
Training losses and performance metrics are saved to
Tensorboard and also to a logfile.

If you are new to these metrics, the one you want to focus on
is ‘mAP_0.5‘-learn more about mean average precision [here]
(https://blog.roboflow.com/mean-average-precision/).
"""

# Commented out IPython magic to ensure Python compatibility.
# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
# %load_ext tensorboard
# %tensorboard --logdir runs

"""# ***Plot and save mAP 0.5 and 0.5:0.95 curves***"""

#import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#load results file
df=pd.read_csv('/content/yolov5/runs/train/exp3/results.csv')



#shows indexes
print(df.columns)

#specify indexes
df = df[['epoch', 'metrics/mAP_0.5',
'metrics/mAP_0.5:0.95']]

#show and save image
sns.set_style('darkgrid')
plt.figure(figsize=(12, 8))
plt.plot(df['epoch'], df['metrics/mAP_0.5'],
label='mAP_0.5')
plt.plot(df['epoch'], df['metrics/mAP_0.5:0.95'],
label='mAP_0.5:0.95')
plt.xlabel('Epoch')
plt.ylabel('mAP')
plt.title('mAP Curves')
plt.legend()
plt.show()
plt.savefig('/content/yolov5/runs/train/exp/mAP_0.5_Curve.jpg')

"""#Run Inference  With Trained Weights
Run inference with a pretrained checkpoint on contents of
`test/images` folder downloaded from Roboflow.
```

```
"""

!python detect.py --weights runs/train/exp/weights/best.pt
--img 416 --conf 0.5 --source {dataset.location}/test/images

#display inference on ALL test images

import glob
from IPython.display import Image, display

for imageName in
glob.glob('/content/yolov5/runs/detect/exp/*.jpg'):
#assuming JPG
    display(Image(filename=imageName))
    print("\n")

"""# Conclusion and Next Steps

Congratulations! You've trained a custom YOLOv5 model to
recognize your custom objects.

To improve your model's performance, we recommend first
interating on your datasets coverage and quality.
See this guide for [model performance improvement]
(https://github.com/ultralytics/yolov5/wiki/
Tips-for-Best-Training-Results).

To deploy your model to an application, see this guide on
[exporting your model to deployment destinations]
(https://github.com/ultralytics/yolov5/issues/251).

Once your model is in production, you will want to
continually iterate and improve on your dataset and model via
[active learning](https://blog.roboflow.com/what-is-active-
learning/)."""

#Deploy Model back to Roboflow
project.version(dataset.version).deploy(model_type="yolov5",
model_path=f"/content/yolov5/runs/train/exp/")

#download full zipped folder
```

```
from google.colab import files
!zip -r /content.zip /content
files.download("/content.zip")
```

B2 - Colour Detection

```
    import  cv2
import numpy  as  np


def empty(a): # empty function for trackbars
    pass

def stackImages(scale, imgArray): # see ch6 for explanation
    rows = len(imgArray)
    cols = len(imgArray[0])
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    if rowsAvailable:
        for x in range ( 0, rows):
            for y in range(0, cols):
                if imgArray[x][y].shape[:2] == imgArray[0]
                [0].shape [:2]:
                    imgArray[x][y] = cv2.resize(imgArray[x][y],
                    (0, 0), None, scale, scale)
                else:
                    imgArray[x][y] = cv2.resize(imgArray[x][y],
                    (imgArray[0][0].shape[1], imgArray[0]
                    [0].shape[0]), None, scale, scale)
                if len(imgArray[x][y].shape) == 2: imgArray[x][y]=
                cv2.cvtColor(imgArray[x][y], cv2.COLOR_GRAY2BGR)
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
        ver = np.vstack(hor)
    else:
        for x in range(0, rows):
            if imgArray[x].shape[:2] == imgArray[0].shape[:2]:
                imgArray[x] = cv2.resize(imgArray[x], (0, 0),
                None, scale, scale)
            else:
                imgArray[x] = cv2.resize(imgArray[x],
                (imgArray[0].shape[1], imgArray[0].shape[0]),
                None, scale, scale)
            if len(imgArray[x].shape) == 2: imgArray[x] =
            cv2.cvtColor(imgArray[x], cv2.COLOR_GRAY2BGR)
        hor= np.hstack(imgArray)
        ver=hor
    return ver

# Add your image names to the list
path = 'resources/Full_Image_set/IMG_3260.jpg'




cv2.namedWindow('TrackBars') # create a window for trackbars
cv2.resizeWindow('TrackBars', 640, 240) # resize the window
to 640x240

#""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# "Important! Define these values FIRST.
# find the values for your image using the trackbars.
and then place them here.
#"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

cv2.createTrackbar('Hue Min', 'TrackBars', 165, 179, empty)
cv2.createTrackbar('Hue Max', 'TrackBars', 179, 179, empty)
cv2.createTrackbar('Sat Min', 'TrackBars',119, 255, empty)
cv2.createTrackbar('Sat Max', 'TrackBars', 255, 255, empty)
cv2.createTrackbar('Val Min', 'TrackBars', 41, 255, empty)
cv2.createTrackbar('Val Max', 'TrackBars', 243, 255, empty)

while True: # create a loop to keep the trackbars open
    img = cv2.imread(path)
    imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # convert the image to HSV
    h_min = cv2.getTrackbarPos('Hue Min', 'TrackBars')
```

```
# get the value of the trackbar each time it moves
h_max = cv2.getTrackbarPos('Hue Max', 'TrackBars')
s_min = cv2.getTrackbarPos('Sat Min', 'TrackBars')
s_max = cv2.getTrackbarPos('Sat Max', 'TrackBars')
v_min = cv2.getTrackbarPos('Val Min', 'TrackBars')
v_max = cv2.getTrackbarPos('Val Max', 'TrackBars')
print(h_min, h_max,s_min,s_max,v_min, v_max)
# prints the values to the consol

lower = np.array([h_min, s_min, v_min])
# create an array with the lower values
upper = np.array([h_max, s_max, v_max])
# create an array with the upper values
mask = cv2.inRange(imgHSV, lower, upper)
# create a mask with the lower and upper values
imgResult = cv2.bitwise_and(img, img, mask=mask)
# create a new image with the mask

contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE) # find the contours


for cnt in contours: # loop through the contours
    x, y, w, h = cv2.boundingRect(cnt) # get the x, y,
    width and height of the contour
    cv2.rectangle(img, (x, y), (x + w, y + h),
    (0, 255, 0), 2) # draw a rectangle around the contour
# cv2.imshow('Original', img)
# show the original image, ect.
# cv2.imshow("HSV", imgHSV)
# cv2.imshow("Mask", mask)
# cv2.imshow("Result", imgResult)



imgStack = stackImages(.2, ([img, imgHSV, mask]))
cv2.imshow("Stacked Images", imgStack)
# cv2.imshow('Original', img)


cv2.waitKey(1)
```

```
    # wait for 1ms, CRUCIAL while in a loop, otherwise
    it will freeze.
```

B3 - Edge Detection

```
import cv2
import numpy as np

def process(img):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_blur = cv2.GaussianBlur(img_gray, (11, 11), 10)
    img_canny = cv2.Canny(img_blur, 0, 65) #img_blur
    is the threshold, 0
    kernel = np.ones((19, 19))
    img_dilate = cv2.dilate(img_canny, kernel, iterations=4)
    img_erode = cv2.erode(img_dilate, kernel, iterations=4)
    return img_erode

def draw_contours(img):
    contours, hierarchies = cv2.findContours(process(img),
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    cnt = max(contours, key=cv2.contourArea)
    peri = cv2.arcLength(cnt, True)
    approx = cv2.approxPolyDP(cnt, 0.004 * peri, True)
    cv2.drawContours(img, [approx], -1, (255, 255, 0), 2)

path = 'Resources/Full_image_set_resized/IMG_3273.jpg'
img = cv2.imread(path)
h, w, c = img.shape

draw_contours(img)



img = cv2.resize(img, (w // 2, h // 2))

cv2.imshow("Image", img)
cv2.waitKey(0)
```

B3 - Object Detection: Image

```python
import torch
import cv2
import numpy as np
from models.experimental import attempt_load
from utils.general import non_max_suppression, scale_coords
from utils.torch_utils import select_device

# Load the YOLOv5 model
weights = 'V21.pt'
device = select_device('cpu') # or 'cuda:0' for GPU
model = attempt_load(weights, device)

# Count the number of free and locked parameters
num_free_params = sum(p.numel()
    for p in model.parameters() if p.requires_grad)
num_locked_params = sum(p.numel()
    for p in model.parameters() if not p.requires_grad)

print('Number of free parameters:', num_free_params)
print('Number of locked parameters:', num_locked_params)

# Set the model to evaluation mode
model.eval()

# Define the classes
classes = ['Hot spot'
    ,'Point flame'
    ,'Stub'
    ,'Tapping hole'
    ,'Uncovered area']

# Define the input image size
def maintain_aspect_ratio(img, target_size, stride=32):
    img_height, img_width = img.shape[:2]
    aspect_ratio = img_width / img_height

    # Check if the image is horizontal or vertical
    if aspect_ratio >= 1:  # Horizontal
        new_width = target_size
        new_height = int(new_width / aspect_ratio)
    else:  # Vertical
        new_height = target_size
        new_width = int(new_height * aspect_ratio)

    # Ensure dimensions are divisible by stride
    new_width = (new_width // stride) * stride
    new_height = (new_height // stride) * stride

    img_resized = cv2.resize(img, (new_width, new_height))

    return img_resized

# Define the confidence threshold and
# non-maximum suppression threshold
conf_threshold = 0.4
nms_threshold = 0.5

# List of input images
image_paths = [
    'Resources\Full_image_set\IMG_3260.JPG',
    'Resources\Full_image_set\IMG_3282.JPG',
    'Resources\Full_image_set\IMG_7437.JPG',
    'Resources\Full_image_set\GOPR0117.JPG',
    'Resources\Full_image_set\GOPR0125.JPG'
]

# Loop through all the images
for image_path in image_paths:
    # Load the input image
    img = cv2.imread(image_path)

    # Preprocess the input image
    img = maintain_aspect_ratio(img, 640)
    img = img[..., ::-1]  # BGR to RGB
    img = np.ascontiguousarray(img)

    # Convert the input image to a tensor
    img_original = img.copy()
    img_vis = img.copy()
    img = img.transpose((2, 0, 1))
img = torch.from_numpy(img).to(device)
img = img.float()
img /= 255.0
if img.ndimension() == 3:
    img = img.unsqueeze(0)

# Run object detection on the input image
outputs = model(img)
results = non_max_suppression(outputs,
conf_threshold, nms_threshold)

# Add "Before" label to the original image
cv2.putText(img_original, "Detection: OFF",
            (10, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 1,
            (255, 255, 255), 2, cv2.LINE_AA)

# Create a dictionary to map class names to
#their corresponding BGR colours
class_colors = {
    'Hot spot': (255, 0, 0),  # Red in RGB
    'Uncovered area': (250, 128, 114), # Salmon in RGB
    'Stub': (245, 197, 66),  # Orange in RGB
    'Point flame': (182, 48, 209),  # Purple in RGB
    'Tapping hole': (0, 255, 0),  # Green in RGB
}
# Visualize the results
for result in results:
    if result is not None:
        result[:, :4] = scale_coords(img.shape[2:],
        result[:, :4], img.shape[2:]).round()
        for x1, y1, x2, y2, conf, cls in result:
            label = classes[int(cls)]
            print(f'{label}: {conf:.2f}')
            color = class_colors[label]
            cv2.rectangle(img_vis, (int(x1), int(y1)),
            (int(x2), int(y2)), color, 1)
            cv2.putText(img_vis, f'{label}:
            {conf:.2f}',
            (int(x1), int(y1) - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                               color, 1, cv2.LINE_AA)

    # Add "After" label to the image with object detection
    cv2.putText(img_vis, "Detection: ON", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1,
                (255, 255, 255), 2, cv2.LINE_AA)

    # Concatenate the original image and output
    # image side by side
    combined_image = cv2.hconcat(
    [img_original[..., ::-1],
    img_vis[..., ::-1]])

    # Show the combined image
    cv2.imshow(f'combined - {image_path}', combined_image)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

B4 - Object detection: Video

```
import torch
import cv2
import numpy as np
from models.experimental import attempt_load
from utils.general import non_max_suppression, scale_coords
from utils.torch_utils import select_device

# Load the YOLOv5 model
weights = 'V21.pt'
device = select_device('cpu')  # or 'cuda:0' for GPU
model = attempt_load(weights, device)

# Set the model to evaluation mode
model.eval()

# Define the classes
classes = ['Hot spot','Point flame','Stub', 'Tapping hole',
'Uncovered area']
```

```
# Define the input image size
def maintain_aspect_ratio(img, target_size):
    img_height, img_width = img.shape[:2]
    aspect_ratio = img_width / img_height

    if img_width >= img_height:
        new_width = target_size
        new_height = int(new_width / aspect_ratio)
    else:
        new_height = target_size
        new_width = int(new_height * aspect_ratio)

    # Ensure dimensions are multiples of 32
    new_width = (new_width // 32) * 32
    new_height = (new_height // 32) * 32

    img_resized = cv2.resize(img, (new_width, new_height))
    return img_resized

# Define the confidence threshold and non-maximum
suppression threshold
conf_threshold = 0.4
nms_threshold = 0.5

# Open the input video
#input_video_path = 'Resources\GoPro_Video\GX010282.MP4'
input_video_path = 'Resources\iPhone_Video\iphone8.mp4'
cap = cv2.VideoCapture(input_video_path)

# Verify if the video file was opened successfully
if not cap.isOpened():
    print('Error: Could not open the video file.')
    exit()

# Get video properties
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Set up the output video writer
output_video_path = 'Resources/output_video.mp4'
```

```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
# Use 'mp4v' codec for .mp4 video format
out = cv2.VideoWriter(output_video_path, fourcc, fps,
(frame_width, frame_height))


# Create a dictionary to map class names to their
corresponding BGR colors
class_colors = {
    'Hot spot': (255, 0, 0),  # Red in RGB
    'Uncovered area': (250, 128, 114), # Salmon in RGB
    'Stub': (245, 197, 66),  # Orange in RGB
    'Point flame': (182, 48, 209),  # Purple in RGB
    'Tapping hole': (0, 255, 0),  # Green in RGB
}


# Process the input video frame by frame
while True:
    ret, frame = cap.read()

    if not ret:
        break

    # Preprocess the frame
    img = maintain_aspect_ratio(frame, 1080)
    img = img[..., ::-1]  # BGR to RGB
    img = np.ascontiguousarray(img)

    # Convert the frame to a tensor
    img_vis = img.copy()  # Create a copy of the original
    frame for visualization
    img = img.transpose((2, 0, 1))  # Move channels to the
    first dimension
    img = torch.from_numpy(img).to(device)
    img = img.float()
    img /= 255.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)

    # Run object detection on the frame
```

```
outputs = model(img)
results = non_max_suppression(outputs, conf_threshold,
nms_threshold)

# Visualize the results
for result in results:
    if result is not None:
        result[:, :4] = scale_coords(img.shape[2:],
        result[:, :4], img.shape[2:]).round()
        for x1, y1, x2, y2, conf, cls in result:
            label = classes[int(cls)]
            conf_percent = conf * 100  # Convert
            confidence to percentage
            print(f'{label}: {conf_percent:.1f}%')
            color = class_colors[label]
            # Get the color for the current class label
            cv2.rectangle(img_vis, (int(x1), int(y1)),
            (int(x2), int(y2)), color, 2)

            # Create a background rectangle for the text
            (text_width, text_height), _ =
            cv2.getTextSize(f'{label}: {conf_percent:.1f}%',
            cv2.FONT_HERSHEY_SIMPLEX, 0.50, 1)
            cv2.rectangle(img_vis, (int(x1), int(y1) -
            text_height),(int(x1) + text_width, int(y1)),
            color, -1)

            # Put the white text on the background
            rectangle
            cv2.putText(img_vis, f'{label}:
            {conf_percent:.1f}%',
            (int(x1), int(y1)), cv2.FONT_HERSHEY_SIMPLEX,
                        0.50,
                        (255, 255, 255), 1, cv2.LINE_AA)

# Write the visualized frame to the output video
out.write(cv2.resize(img_vis[..., ::-1], (frame_width,
frame_height)))

# Show the output frame
cv2.imshow('output', img_vis[..., ::-1])
```

```
    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video writer and the video capture objects
out.release()
cap.release()

# Close all OpenCV windows
cv2.destroyAllWindows()
```

C - REPORT ORGANISING AND EMPLOYER

C1 - Report Organising

This thesis is written by a team of three final-year bachelor students, each of whom holding a certificate of apprenticeship in electrical, chemical, and automation, respectively. To ensure sound management, we divided ourselves into the following roles:

- Communication: Eskil Digernes
- Financial: Stian Flåten
- Technical: Asbjørn Tjensvold

Figure C.1: From left to right: Asbjørn, Stian, Eskil.

C2 - Employer

Norsk Hydro is a global industrial conglomerate and one of Norway's largest companies. The company specialises in producing aluminium solutions and is active throughout the entire value chain, from mining to producing rolled and extruded aluminium products. In addition to its core business, Norsk Hydro also operates hydroelectric power plants that produce aluminium. The company was established in 1905 and has its headquarters located in Oslo [59].

Hydro

Figure C.2: Norsk Hydro's emblem [60].

This thesis is conducted at Hydro Årdal, under the Technology and Operational Support (TOS) branch. The smelter in Øvre Årdal is staffed by 520 employees and had a profit of 912 million NOK in the second quarter of 2022 [61].

Figure C.3: The aluminium smelter in Øvre Årdal [61].

D - CALCULATION OF SAVINGS

NOTE: This hypothetical calculation, derived from Figure 2.1.5 provided by Hydro Årdal, demonstrates the potential for significant annual savings. By implementing these measures, Hydro could save up to 3,600,000 NOK in CO_2 quotas and an additional 10,909,091 NOK in carbon anode costs.

D1 - Current Scenario (2022)

- Quota price per tonnes CO_2: 900 NOK / tonne CO_2
- The current ratio is 16/167 airburn to total CO_2 emissions.
- Production: 200,000 tonnes Al/year, 1.67 tonnes CO_2/tonnes Al

Current annual CO_2 emissions:

$200{,}000\,\text{t Al/yr} \cdot 1.67\,\text{t CO}_2\text{/t Al} = 334{,}000\,\text{t CO}_2\text{/yr}$

Current annual CO_2 quota cost:

$334{,}000\,\text{t CO}_2\text{/yr} \cdot 900\,\text{NOK/t CO}_2 = 300{,}600{,}000\,\text{NOK/yr}$

Current annual cost of airburns:

$300{,}600{,}000\,\text{NOK/yr} \cdot \dfrac{16}{167} = 28{,}800{,}000\,\text{NOK/yr}$

D2 - CO_2 Target for 2025

- Target ratio for 2025: 14/155 airburn to total CO_2 emissions
- Assuming same aluminium production and CO_2 quota price.

Calculate the 2025 target annual CO_2 emissions:

$200{,}000\,\text{t Al/yr} \cdot 1.55\,\text{t CO}_2\text{/t Al} = 310{,}000\,\text{t CO}_2\text{/yr}$

Calculate the current annual CO_2 quota cost:

$310{,}000\,\text{t CO}_2\text{/yr} \cdot 900\,\text{NOK/t CO}_2 = 270{,}000{,}000\,\text{NOK/yr}$

Calculate the current annual cost of airburns:

$270{,}000{,}000\,\text{NOK/yr} \cdot \frac{14}{155} = 25{,}200{,}000\,\text{NOK/yr}$

Hypothetical annual CO_2 quota savings

$28{,}800{,}000\,\text{NOK/yr} - 25{,}200{,}000\,\text{NOK/yr} = 3{,}600{,}000\,\text{NOK/yr}$

D3 - Savings of Carbon Anode Mass

Given the following:

- Price of carbon anode mass: 10,000 NOK/t
- Current CO_2 emissions: 334,000 CO_2/yr
- $M_\text{C} = 12\,\text{g/mol}$, $M_{\text{CO}_2} = 44\,\text{g/mol}$
- Carbon-to-CO_2 ratio: $\frac{12}{44}$

Goal 2025: Reduce CO_2 emissions by 20 kg per metric tonne Al

$334{,}000\,\text{t CO}_2/\text{yr} - 200{,}000\,\text{t Al/yr} \cdot 1{,}65\,\text{CO}_2/\text{Al} = 4{,}000\,\text{t CO}_2/\text{yr}$

Annual carbon mass savings for 2025

$4{,}000\,\text{t CO}_2/\text{yr} \cdot \dfrac{12}{44} \cdot 10,000\,\text{NOK} = 10{,}909{,}091\,\text{NOK/yr}$

Short term goal: Reduce CO_2 emissions by 3 kg per metric tonne Al

$334{,}000\,\text{t CO}_2/\text{yr} - 200{,}000\,\text{t Al/yr} \cdot 1{,}667\,\text{CO}_2/\text{Al} = 600\,\text{t CO}_2/\text{yr}$

Annual carbon mass savings short term

$600\,\text{t CO}_2/\text{yr} \cdot \dfrac{12}{44} \cdot 10,000\,\text{NOK} = 1{,}636{,}364\,\text{NOK/yr}$

The price of carbon anode mass was given by Dr Trond Eirik Jentoftsen, Head of Operations IOS Electrolysis, Norsk Hydro.