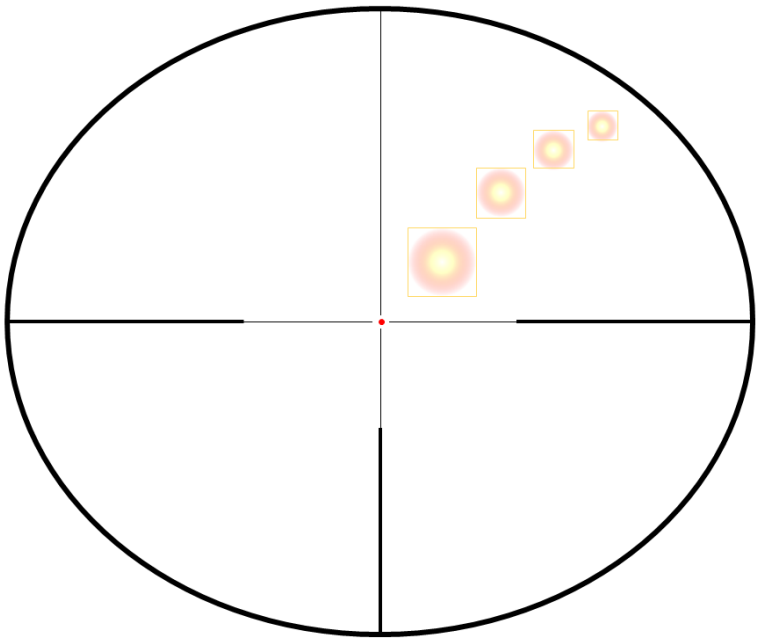

Comparative Evaluation of Detection Methods for Virtual Objects

Master's thesis

Performance Analysis of Object Detection Techniques on Simulated Projectiles and Explosions in Analog Video Sequence

Master's thesis in Systems, Control and Mechatronics

William Horngacher

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Comparative Evaluation of Detection Methods for Virtual Objects

Performance Analysis of Object Detection Techniques on Simulated Projectiles and Explosions in Analog Video Sequence

William Horngacher

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Performance Analysis of Object Detection Techniques on Simulated Projectiles and Explosions in Analog Video Sequence
William Horngacher

Supervisors:
Mādālina Aldea, SAAB Training Systems AB
Carmine Celozzi, SAAB Training Systems AB
Eric Damen, SAAB Training Systems AB
André Bezerra de Freitas Diniz, Department of Electrical Engineering

Examiner:
Erik Agrell, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An image that displays the virtual projectiles that this project will work with. It also includes a scope for aesthetic effect.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Abstract

With increasing global military spending, there is a growing need for modern training tools that can simulate realistic combat conditions without the risks and costs associated with live ammunition. SAAB Training and Simulation addresses this need with innovative laser-based systems that can be mounted on various weapon platforms.

In certain situations, there is a need to provide real-time visual feedback by displaying virtual projectiles and explosions on a video screen, such as inside vehicles, enhancing training realism. This project aims to create automated testing capabilities for the video-generating unit of this system by creating a program that can detect the projectiles and explosions in the video feed.

Two different object detection models were tested. One was an algorithmic detection model called the Round Object Detector, which utilizes the Hough circle algorithm due to its efficiency in detecting circular shapes in an image. The other model was the Faster Region-Based Convolutional Neural Network (Faster R-CNN) for its capability to identify complex features. A synthetic dataset with diverse backgrounds was created to train the Faster R-CNN, along with three manually annotated datasets from the video-generating unit for model evaluation. A selective detection technique enhanced detection accuracy by adjusting prediction confidences based on expected projectile positions. The Faster R-CNN displayed high precision across various backgrounds in the video generator tests, although recall varied, signifying challenges in consistent object detection. The Round Object Detector achieved high precision with simple backgrounds but struggled with detailed, colorful settings due to limitations in color filtering, leading to increased false detections. Selective detection marginally enhanced overall model performance, particularly improving the Faster R-CNN's recall.

The project showed that the Faster R-CNN model could transfer its knowledge well from the synthetic dataset to the analog video frames. The false positive predictions were still few, but the false negative rate increased, which showed that fine-tuning the model on an annotated dataset from the real application could be a way to improve performance. The Round Object Detector performed significantly worse than the Faster R-CNN model, and it proved to be the color filter that was the main problem to get working correctly. Selective detection proved to be a useful tool for the Faster R-CNN, where it improved recall, which was the main challenge for both models when used on the video sequence.

Keywords: Object Detection, Faster R-CNN, Synthetic Dataset, Hough Circle Algorithm, Selective Detection.

Acknowledgements

I am deeply grateful for the support and guidance I received throughout my master's thesis project. I extend a special thanks to my supervisors at SAAB: Mādālina Aldea, Carmine Celozzi, and Eric Damen. Their willingness to share their profound knowledge and expertise was invaluable. Without their input and encouragement, completing this project would not have been possible.

I am also deeply thankful to my supervisor at Chalmers, André Bezerra de Freitas Diniz, whose constructive feedback and rigorous standards significantly enhanced the quality of this project. I also owe a big thanks to my examiner, Erik Agrell, for taking on this project and whose evaluation and feedback made the project take its final form.

This project has been an important part of my personal growth both academicaly and professionaly. I am immensely appreciative of the opportunity to have worked together with such dedicated and knowledgeable individuals.

William Horngacher, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial intelligence
BGRA	Blue Green Red Alpha
CAN	Controller Area Network
CNN	Convolutional Neural network
COCO	Common Objects in Context
FN	False Negative
FOV	Field of View
FP	False Positive
FPS	Frames Per Second
HSV	Hue Saturation Value
ML	Machine Learning
NMS	Non-maximum Supression
PNG	Portable Network Graphics
R-CNN	Region-based Convolutional Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
TN	True Negative
TP	True Positive

Contents

List of Figures

1
Introduction

Amid growing military investments in Sweden and globally, the demand for advanced training tools for soldiers is also increasing. SAAB Training and Simulation is at the forefront of this development, specializing in innovative solutions that enhance combat training. Their main product is a laser system that can be mounted on any weapon of choice. These systems facilitate realistic shooting drills and war scenario simulations without using live ammunition [1].

When the system is implemented on vehicles such as tanks, the projectile trajectories, and explosions must be displayed on a video screen inside the tank to create a realistic experience for the user. The system that generates these frames will henceforth be called the "video mixer".

This chapter outlines the project's background, significance, and goals to achieve with its results. Section 1.1 provides the background to the project, and Section 1.2 the aim together with the research questions.

1.1 Background

SAAB is exploring automating some testing processes for their video mixer to reduce the time and resources required to validate the system. The initiative involves developing a program that takes a video input sampled from a customer or within SAAB and accurately detects projectiles and explosions within each frame.

The video mixer operates by receiving controller area network (CAN) messages [2]. This network protocol facilitates communication between multiple devices in a network without a central computer unit. These messages provide real-time updates on the positions of projectiles and explosions.

Based on the CAN messages, the video mixer calculates the position of the projectiles and explosions. It then adds the projectiles to the video input based on the calculated position and displays it on an analog screen.

The main challenge of this project lies in achieving a high detection accuracy. If the detection accuracy is sufficiently high, it will enable comparisons between the observed positions on the screen and the expected positions based on initial trajectory settings and viewing angles.

1.2 Aim

The project aims to create a program to detect the position of the projectiles and explosions on the output frames from the video mixer. The detected location should

be evaluated on an annotated set of frames to quantify the detection methods' performance. Another aim is to explore how a machine learning (ML) model and an algorithmic method perform the object detection task. The project also aims to understand how an ML model trained on synthetic data can transfer its performance to the real-world application [3].

The research questions chosen for this project are listed below:

- How does an ML model trained on synthetic data transfer to analog video frames?
- How do ML models compare to algorithmic methods for detecting circular objects in an analog video?
- How do ML models and algorithms perform when circular objects are becoming very small?
- How can the known approximate location of an object improve the performance of an object detector?

1.2.1 Limitations

Limitations must be defined to investigate the selected research questions for this project. This master's thesis project has time constraints due to the 20-week, full-time study format equivalent to 30 ECTS. This constraint limits the amount of time that can be put into the development and improvements of the object detection models, which may affect the performance and findings of this project. Due to the time limitations, there will not be any real-world image collection to use for the datasets, but it will instead rely on publicly available data and images created by generative artificial intelligence (AI).

There are also limitations to what can be done to increase the detection performance of the object detection methods. The animations of how the projectile and explosion look are designed by SAAB, and that will not be changed during the project. Neither will there be any changes made to the video mixer that affect the project, like changing any of the code or extracting information from inside the system.

Since this project is taking place at SAAB, which works in the military industry, rigorous security standards must be followed. It limits the type of software that can be used, and all the work has to be done and stored on the local work computer. This limits the types of open-source software and programming packages that can be utilized during the project. The licenses also had to be free to use not only for academic purposes but also for commercial use if SAAB wants to use the programs created during this project in the future.

1.3 Ethics

Given SAAB's role in the defense industry, it is important to consider the ethical implications of this project. SAAB Training and Simulation, where this project took place, manufactures training equipment for militaries [1]. A successful project could improve the training tools that SAAB sells to its customers. Improvements in the training tools would, in turn, result in a better-prepared military in the case of an

armed conflict. This ethics section is split into three subsections: training safety, environmental, and national security.

1.3.1 Training safety

A significant ethical benefit of laser training tools is their potential to reduce personal injuries during military exercises. If the soldiers can use laser shooting for training to a larger extent, the risk of personal injuries from projectiles is reduced.

1.3.2 Environmental

Another aspect of these laser training tools is the environmental impact they can have on minimizing the use of sharp ammunition during training. When firing weapons, waste is created as a bullet and shell. This waste, often left in nature, can pollute local ecosystems. If these laser weapons are improved, they can be used more often when training soldiers, which would reduce environmental impacts where training takes place.

1.3.3 National security

In Sweden, strict regulations govern the sale of military products. The organization that decides what customers the products can be sold to is called Inspektionen för Strategiska Produkter, ISP [5]. ISP is a government-controlled institution, which means it is ultimately governed by democratically elected officials, safeguarding the interests of the Swedish people. There are also laws limiting the export of military products [6]. ISP's role means that the risk of these training tools ending up in the hands of people working against the interests of Sweden and threatening Swedish security is limited.

One ethical question that must be addressed is how the use of training tools and the resulting improvement in military capabilities would impact national security. Amid increasing military aggression globally and ongoing conflict in Europe, the risk of international conflicts has escalated. Experts are torn on this question, and one common theory is the one of deterrence [7]. Deterrence theory argues that the consequences of military action by the aggressor would be too destructive and deter the enemy from a conflict. Although this theory is mainly used when discussing nuclear weapons, it can also be interpreted to include other military investments.

However, one could argue that deterrence theory, rather than preventing conflict, may actually escalate aggression and increase the likelihood of conflict. There is a risk that the strengthened defense could be seen or twisted into a provocation used to justify military action.

There is a debate on whether increasing military spending is wise, but most countries now advocate increased military capabilities to protect their interests. This also applies to Sweden, and the need for a stronger military to protect Swedish values is increasing. To prepare the soldiers as well as possible in the case of an armed conflict, the training tools made by SAAB are essential for Sweden's ability to defend itself and its territory.

2

Theory

The following chapter aims to provide the necessary knowledge and methodology for the project. Section 2.1 gives the theory behind the machine learning model used for detecting projectiles and explosions; Section 2.2 introduces the theory behind some of the computer vision tools used during the project. Section 2.3 explains the purpose of the dataset and how it works; Section 2.4 gives the background knowledge needed about the video mixer system; Section 2.5 explains how the performance can be quantified for object detection models.

2.1 Machine learning

In this section, we introduce the field of machine learning, the theory behind how a neural network works, and how the network can learn patterns from a dataset. Convolutional neural networks will also be explained, and how they can extract features from an image that enable the detection of objects in an image. The theory behind the Faster R-CNN model is also introduced to provide the necessary background on how it works.

2.1.1 Neural networks

A neural network is composed of neurons structured into layers. Each neuron takes inputs from a previous layer to calculate a weighted sum. These neurons collectively form a network capable of learning patterns from training data by adjusting weights and bias parameters to minimize prediction errors. Figure 2.1 displays a single neuron and how it takes an input vector $\mathbf{x}$ and calculates the output a.

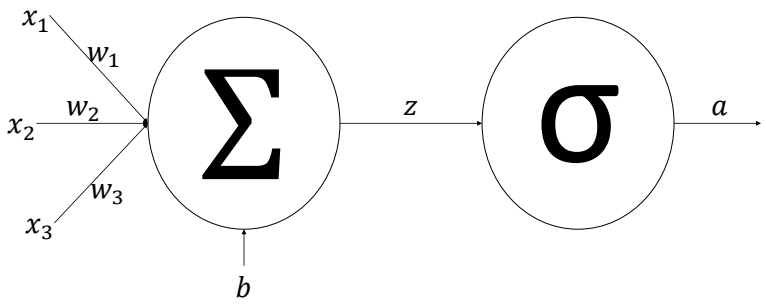

Figure 2.1: A schematic displaying a single neuron and how it calculates the output, a, from a given input $\mathbf{x} = [x_1, x_2, ..., x_n]$ where $n = 3$.

Each neuron has two types of trainable parameters: a weights vector and a bias term. The weights vector consists of real numbers, denoted as $\mathbf{w} = [w_1, w_2, w_3, \ldots, w_n]$, where n represents the number of inputs from the previous layer. The bias term variable denoted as b, is a real number and independent of the input. The first part of the neuron calculates the weighted sum z. The input is denoted $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_n]^\intercal$. The equation to calculate z is shown in Eq. (2.1)

$$z = \mathbf{w} \cdot \mathbf{x} + b. \tag{2.1}$$

The resulting sum, z, of each neuron in the layer is then combined into a vector $\mathbf{z} = [z_1, z_2, z_3, ..., z_n]^\intercal$ where n is the number of neurons in the layer. The combined vector $\mathbf{z}$ is then fed into a non-linear function calculating the output of the layer, denoted $\mathbf{a}$. The non-linear function, also called the activation function is applied to each element in the vector making $\mathbf{a}$ keep the same size as $\mathbf{z}$. Two examples of activation functions are ReLU and Sigmoid [8], which are denoted as σ. The activation function is shown in Eq. (2.2).

$$\mathbf{a} = \sigma(\mathbf{z}). \tag{2.2}$$

The resulting value of $\mathbf{a}$ is then fed forward to the next layer and used as input in Eq. (2.1) instead of $\mathbf{x}$ meaning $\mathbf{x}^{(i)} = \mathbf{a}^{(i-1)}$ for $i = [2, 3, \ldots, m]$ where m is the number of layers in the network. The network structure enables the learning of complex patterns. An example network is shown in Figure 2.2 where each circle represents a neuron from Figure 2.1 and the output is the values of the nodes in the output layer.

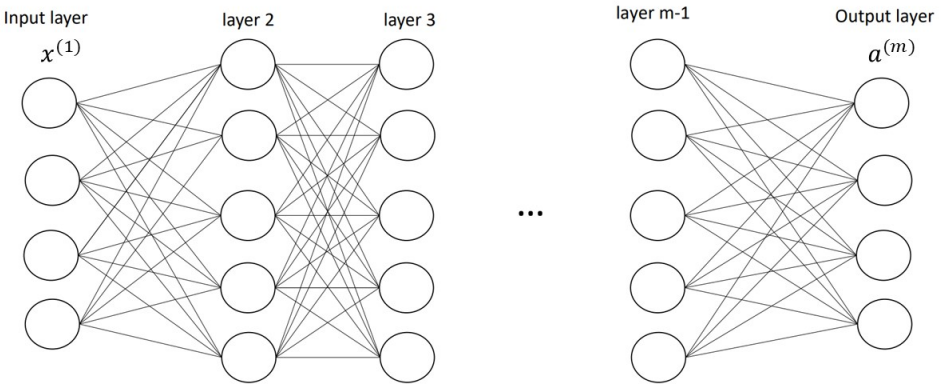

Figure 2.2: A simple structure of a neural network with 4 inputs and 4 outputs consisting of m layers.

To calculate the output $\mathbf{a}^{(m)}$ from the input $\mathbf{x}^{(1)}$ the weights and biases are reformated into matrices and vectors respectively to simplify the equations. The subscript n represents the number of neurons in a layer whereas the subscript to n specifies which layer. The weights of each layer become a matrix where each row is all the weights for a single neuron in the layer. There are corresponding weights and biases to every layer except the output layer meaning there are $m-1$ number of weight matrices and bias vectors meaning $i = [1, 2, \ldots, (m-1)]$.

$$\mathbf{W}^{(i)} = \begin{bmatrix} \mathbf{w}_1^{(i)} & \mathbf{w}_2^{(i)} & \ldots & \mathbf{w}_{n_i}^{(i)} \end{bmatrix}^\mathsf{T} = \begin{bmatrix} w_{1,1}^{(i)} & w_{1,2}^{(i)} & \cdots & w_{1,n_{i-1}}^{(i)} \\ w_{2,1}^{(i)} & w_{2,2}^{(i)} & \cdots & w_{2,n_{i-1}}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_i,1}^{(i)} & w_{n_i,2}^{(i)} & \cdots & w_{n_i n_{i-1}}^{(i)} \end{bmatrix}. \tag{2.3}$$

The bias is combined into a vector where each element represents the bias of a single neuron:

$$\mathbf{b^{(i)}} = \begin{bmatrix} b_1^{(i)} & b_2^{(i)} & \ldots & b_{n_i}^{(i)} \end{bmatrix}^\mathsf{T}. \tag{2.4}$$

Calculating the output from the input can be done by repeating Eqs. (2.5) and (2.6) where $\mathbf{a}^{(1)}$ is the input, $\mathbf{x}^{(1)}$, to the network:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)} \cdot \mathbf{a}^{(i)} + \mathbf{b}^{(i)}, \tag{2.5}$$

$$\mathbf{a}^{(i+1)} = \sigma\left(\mathbf{z}^{(i)}\right), \tag{2.6}$$

leaving the output $\mathbf{a}^{(m)}$.

2.1.1.1 Training the network

During neural network training, the weights and biases of all neurons are updated according to the patterns learned from the training dataset. The update is done by calculating the gradient for the weights and biases for each layer relative to the resulting loss function using the chain rule $\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x)$.

In the next equations, the superscript denotes the specific layer associated with the values where m is the number of layers and $i = [(m-1), (m-2), \ldots, 1]$. The loss referred to as L is a single value calculated from the error between the network output and the ground truth in the training dataset.

The gradient of the loss relative to the values in the output layer neurons referred to as the output can be calculated as:

$$\frac{\partial L}{\partial \mathbf{a}^{(m)}} = \frac{\partial L}{\partial \text{ output}}. \tag{2.7}$$

Calculating the gradient of the loss relative to the weights and biases for each layer is then done by repeating the following set of equations:

$$\frac{\partial L}{\partial \mathbf{z}^{(i)}} = \frac{\partial L}{\partial \mathbf{a}^{(i+1)}} \cdot \frac{\partial \mathbf{a}^{(i+1)}}{\partial \mathbf{z}^{(i)}}, \tag{2.8}$$

$$\frac{\partial L}{\partial \mathbf{w}_j^{(i)}} = \frac{\partial L}{\partial \mathbf{z}^{(i)}} \cdot \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{w}_j^{(i)}}, \quad \text{for } j = [1, \ldots, n_i], \tag{2.9}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(i)}} = \frac{\partial L}{\partial \mathbf{z}^{(i)}} \cdot \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{b}^{(i)}}, \tag{2.10}$$

$$\frac{\partial L}{\partial \mathbf{a}^{(i)}} = \frac{\partial L}{\partial \mathbf{z}^{(i)}} \cdot \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{a}^{(i)}}. \tag{2.11}$$

After calculating the loss gradient with respect to the weights and biases for each layer, these vectors are reformate into the matrices and vectors:

$$\Delta \mathbf{W}^{(i)} = \left[\frac{\partial L}{\partial \mathbf{w}_1^{(i)}}, \frac{\partial L}{\partial \mathbf{w}_2^{(i)}}, \cdots, \frac{\partial L}{\partial \mathbf{w}_{n_i}^{(i)}} \right]^\intercal, \tag{2.12}$$

$$\Delta \mathbf{b}^{(i)} = \frac{\partial L}{\partial \mathbf{b}^{(i)}} \tag{2.13}$$

Using $\Delta \mathbf{W}^{(i)}$ and $\Delta \mathbf{b}^{(i)}$, the weights and biases can be updated using an optimizer.

2.1.1.2 Optimizers

Several methods, known as optimizers, are used to update the weights and biases in a neural network to minimize the loss function. These optimizers can decrease the required training time, avoid local minimum points when training a model, and increase stability and accuracy during training. Below are some examples of optimizers that are commonly used when training a neural network.

Gradient descent

Gradient descent is a fundamental optimization algorithm used when training neural networks [9, pp. 82-91]. It iteratively adjusts weights and biases to minimize the loss function. This method works by moving in the opposite direction of the gradient of the loss function. This method utilizes a parameter known as the learning rate, denoted by α, which controls the size of each update step. Using an appropriate value for α is pivotal for a quick and effective learning process for the model.

Eqs. (2.14) and (2.15) demonstrate how gradient descent updates the weights and biases at each training step utilizing the gradients calculated for all weights and biases for every layer in Eqs. (2.12) and (2.13) with $i = [1, 2, \ldots, (m-1)]$:

$$\mathbf{W}^{(i)} \leftarrow \mathbf{W}^{(i)} - \alpha \Delta \mathbf{W}^{(i)}, \tag{2.14}$$

$$\mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i)} - \alpha \Delta \mathbf{b}^{(i)}. \tag{2.15}$$

Each update step calculates incremental adjustments to the model parameters, steering the network toward optimal performance by minimizing errors between predictions and ground truths.

Stochastic gradient descent

Stochastic Gradient Descent (SGD) is a variant of gradient descent that enhances optimization by randomly learning from individual data samples or small batches of the dataset for each update step [9, pp. 294-296]. This randomness helps mitigate the risk of the model remaining in a local minimum, a common issue with standard gradient descent when learning from complex data. Additionally, SGD can lead to faster convergence with large datasets, as each step is less computationally heavy, resulting in quicker updates.

Momentum

Momentum is another method that can be used in combination with any optimizer [9, pp. 296-299]. When training a neural network, momentum has the purpose of accelerating the convergence rate towards the optimal solution. It achieves this by incorporating knowledge about the previous update step into the current one. It acts as inertia that helps overcome local minimum points and smoothens erratic updates. A vector $\mathbf{v}$, often referred to as velocity, accumulates the current gradient and a portion of the velocity from the previous update. The velocity helps the update step to continue in a consistent direction even if the gradients fluctuate. Using momentum can significantly increase training speed and lead to more stable convergence. There are velocity vectors for both the weight updates, $\mathbf{v_w}$ and the bias update, $\mathbf{v_b}$. The momentum constant, β, determines how much influence the previous velocity should have on the next update step. The update steps, when applied on gradient descent, are displayed below where Eqs. (2.16) and (2.17) updates the weights and Eqs. (2.18) and (2.19) the biases with $i = [1, 2, \ldots, (m-1)]$:

$$\mathbf{v_w}^{(i)} \leftarrow \beta \cdot \mathbf{v_w}^{(i)} + (1 - \beta) \cdot \Delta \mathbf{W}^{(i)}, \tag{2.16}$$

$$\mathbf{W}^{(i)} \leftarrow \mathbf{W}^{(i)} - \alpha \cdot \mathbf{v_w}^{(i)}, \tag{2.17}$$

$$\mathbf{v_b}^{(i)} \leftarrow \beta \cdot \mathbf{v_b}^{(i)} + (1 - \beta) \cdot \Delta\mathbf{b}^{(i)}, \tag{2.18}$$

$$\mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i)} - \alpha \cdot \mathbf{v_b}^{(i)}. \tag{2.19}$$

2.1.1.3 Fitting a model to training data

When training an ML model on a dataset, the objective is to make the model generalize effectively to similar tasks beyond the training data. When doing this there is a risk that the model will either underfit or overfit the data. Underfitting occurs when the model fails to learn sufficient patterns from the training data, resulting in poor generalization to new data. Conversely, overfitting happens when the model learns patterns that are too specific to the training data, hindering its ability to generalize to other datasets.

A simple example of underfitting, appropriate capacity, and overfitting when fitting a curve to a set of points is displayed in Figure 2.3.

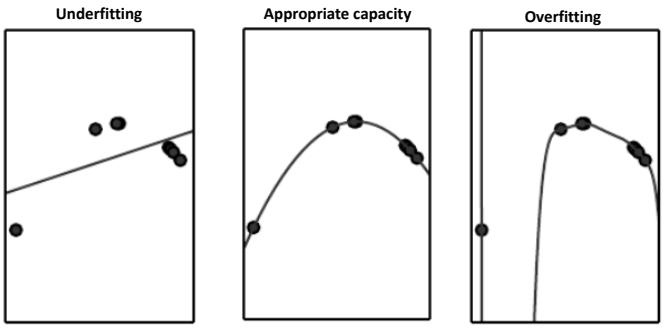

Figure 2.3: Simple example of fitting a line to a set of points, image adapted from [9, p. 113].

By analyzing the error while training the model, it is possible to determine if it reached an appropriate capacity. Figure 2.4 shows an example of where the optimal capacity is when training a model where the training and validation errors are also shown.

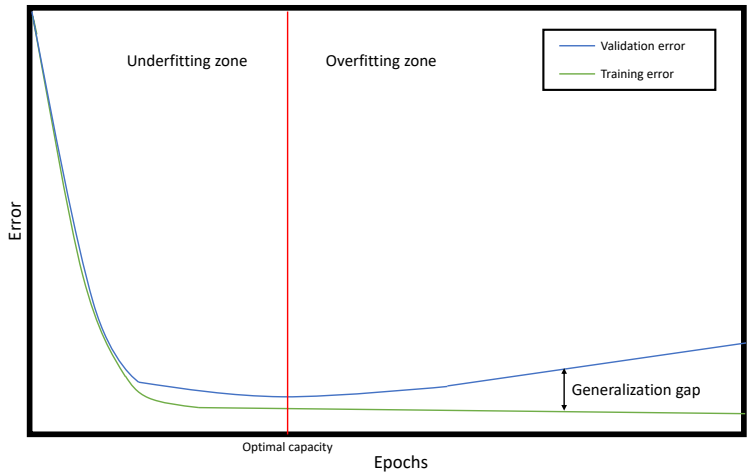

Figure 2.4: Optimal capacity reached while training a neural network.

2.1.2 Convolutional Neural Network

Convolutional neural networks (CNN) are a specialized neural network primarily used for image processing [10]. The main benefit of a CNN is how it, in an efficient way, can detect spatial patterns in an image. This makes it particularly effective in image classification and object detection. The network has a unique architecture with convolutional layers and pooling layers, which are introduced below.

Convolutional layer: A convolutional layer employs a set of trainable kernels that slide across the input image with a set step size. As the kernels are moved across the image, they perform element-wise multiplications and sum the results. The result from all the calculations of the kernels sliding across the image is a feature map representing detected features such as edges, corners, or straight lines [11, pp. 291-293]. These features are fundamental to understanding the content of an image.

Pooling layer: After extracting the features using the convolutional layer, pooling layers are typically applied. The pooling layer reduces the spatial dimensions of the input feature maps, decreasing the number of computations required. The pooling layer also increases the network's robustness to the position of features in the input [9, p. 342]. Two common pooling layers are max pooling and average pooling, which take the maximum value and the average value of a specific pool size [13]. The process of applying a pooling layer can also be called downsampling, which increases the model's receptive field of the filters, allowing the detection of more abstract features deeper in the network.

In Figure 2.5, an example of a CNN is shown. It displays how a set of kernels, called maps, is convoluted over an image. After the convolutional layer, the feature maps are fed into a pooling layer, which reduces the size of the feature map.

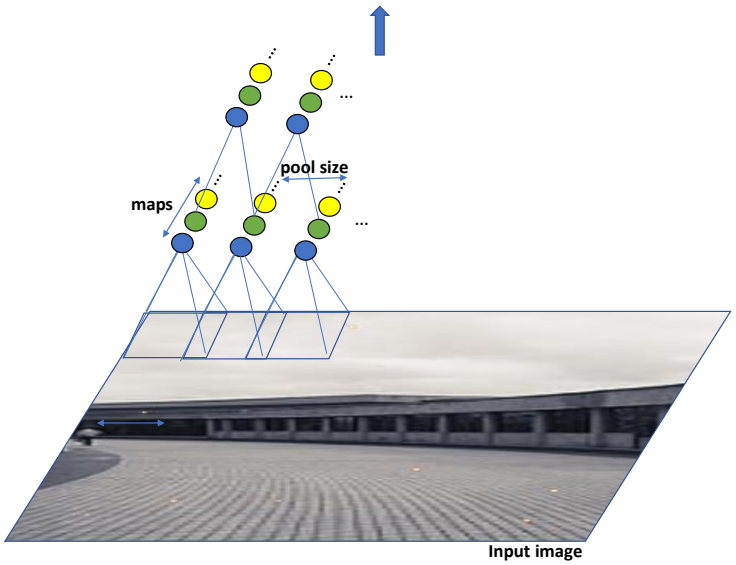

Figure 2.5: Image displaying first convolutional layers with pooling, adapted from [10].

2.1.3 Faster R-CNN

Faster R-CNN [14] is an advanced object detection model that builds upon its predecessors R-CNN [15] and the Fast R-CNN model [16]. The architecture of the Faster R-CNN model consists of several key components, each playing a crucial role in object detection. Figure 2.6 provides an overview of all components in the model.

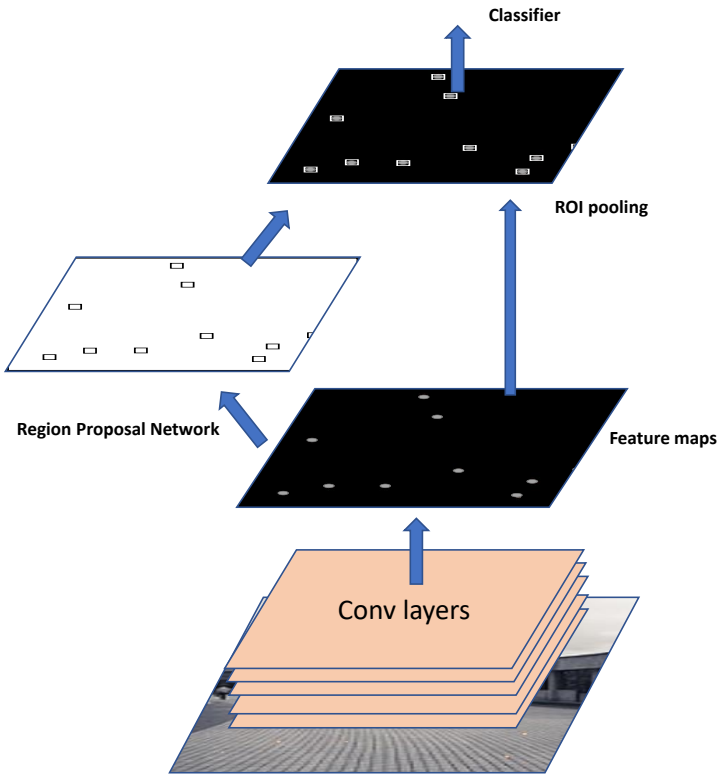

Figure 2.6: Faster R-CNN components, adapted from [14].

2.1.3.1 Feature extraction

The feature extraction, which creates the feature maps in the Faster R-CNN model, is performed by convolutional layers. A popular choice for the architecture is ResNet [17], which uses deep residual learning to train very deep networks effectively. Residual learning helps address the problem of vanishing gradients that can occur with increased network depth.

2.1.3.2 Region Proposal Network

The Region Proposal Network (RPN) in the model takes the feature map generated by the convolutional layers as input and outputs bounding box proposals for object locations [14]. A small neural network called the intermediate layer is convoluted over the feature map, as shown in Figure 2.7. The output from the intermediate layer is then fed into two fully connected layers called the box-classification layer (cls) and a box-regression layer (reg). For each sliding window position, k anchor boxes are tried.

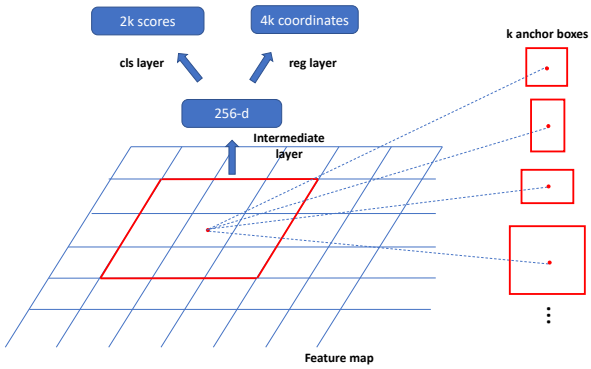

Figure 2.7: RPN simplified overview, adapted from [14].

2.1.3.3 Region of Interest pooling

The region of interest (ROI) pooling transforms the proposals from the RPN and the feature map into a feature vector of a predetermined size consisting of the regions of interest. The resulting feature vector can then be fed into the classifier.

2.1.3.4 Classifier

The classifier in Faster R-CNN makes the final predictions about where the objects are located in the input image. It takes the processed proposals from the ROI pooling layer and outputs the final predictions. For each proposal, the classifier assigns a label and a confidence score. The confidence score is a number between zero and one that indicates how certain the model is about its prediction. If the confidence of a prediction is below a predefined threshold, the prediction is removed.

Additionally, the classifier utilizes a Non-maximum suppression (NMS) step to resolve the problem of overlapping between bounding boxes. The NMS ensures that only the most probable bounding box represents each detected object. This is achieved by excluding predictions where the bounding box substantially overlaps with another box of the same class with a higher confidence score.

2.1.4 Loss functions

Optimizing a neural network involves defining and minimizing a loss function. This function quantifies the discrepancy between the model's predictions and the actual ground truths. For complex models like the Faster R-CNN, used for object detection, the loss function comprises several components that address different aspects of the model's predictions [11, p. 280].

Examples of these are the classification error and the bounding box localization error. The classification error measures how well the model predicts the correct label for the detected object.

The localization error assesses the accuracy of the bounding boxes predicted by the model, and the error evaluates how close the predicted box aligns with the ground truth box in the training data.

These losses are calculated for both the RPN and the classifier network. The classifier network will be referred to as the detection network. The total loss is the sum of the different losses the training aims to minimize.

Each loss requires a penalty constant for the localization and classification error for both the RPN and the detection network. For the classifier, these constants are denoted as λ_{cls} and $\lambda_{\text{cls}}^{'}$ for the RPN and detection network, respectively. The penalty constants for the localization are denoted as λ_{loc} and $\lambda_{\text{loc}}^{'}$ for the RPN and detection network. The losses use subscripts to show which loss it corresponds to: $L_{\text{cls, RPN}}$, $L_{\text{loc, RPN}}$, $L_{\text{cls, Det}}$ and $L_{\text{loc, Det}}$.

The losses are calculated in Eq. (2.20) to Eq. (2.22):

$$L_{\text{RPN}} = \lambda_{\text{cls}} \times L_{\text{cls, RPN}} + \lambda_{\text{loc}} \times L_{\text{loc, RPN}}, \tag{2.20}$$

$$L_{\text{Det}} = \lambda_{\text{cls}}' \times L_{\text{cls, Det}} + \lambda_{\text{loc}}' \times L_{\text{loc, Det}}, \tag{2.21}$$

$$L = L_{\text{RPN}} + L_{\text{Det}}. \tag{2.22}$$

The penalty constants λ_{cls}, $\lambda_{\text{cls}}^{'}$ and λ_{loc}, $\lambda_{\text{loc}}^{'}$ can be changed depending on how much the different errors should be penalized in the loss function.

2.2 Image processing

In this section, the different image processing techniques will be introduced. OpenCV is a popular open-source library extensively used in image processing [4], which will be used to simplify the implementation of the methods. The methods include color filtering, Canny edge detection, and the Hough circle algorithm.

2.2.1 Color filtering

OpenCV enables color filtering using the function "inRange" [18]. The function generates a mask that retains only the pixels within a selected color bounds.

HSV

HSV stands for "Hue", "Saturation", and "Value" which is a frequently used way to describe the color of a pixel. The OpenCV color format is displayed in Figure 2.8. The Hue is between 0 and 180, the saturation is between 0 and 255, and the value is between 0 and 255.

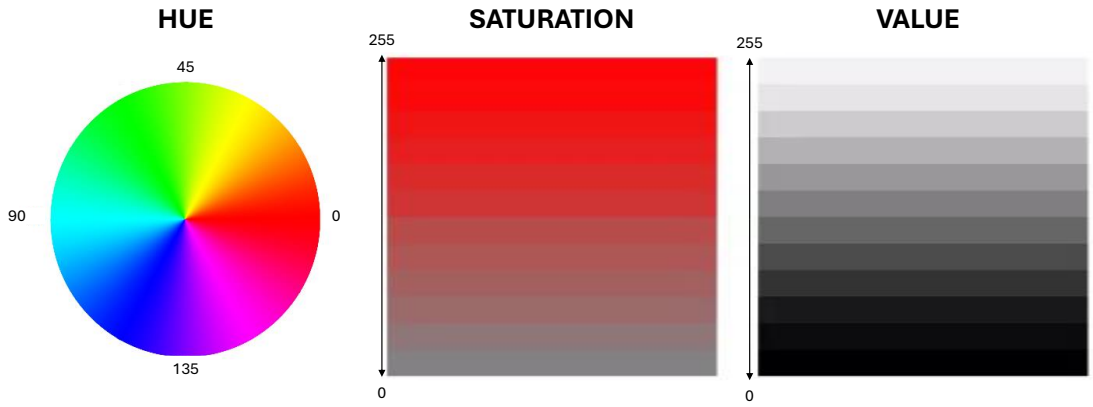

Figure 2.8: Illustration of HSV color format in OpenCV, demonstrating how different colors are encoded within this color space.

2.2.2 Canny edge detection

The Canny edge detection algorithm is a multi-stage process that efficiently detects edges in an image by analyzing the pixel gradients between the adjacent pixels [19].

OpenCV provides an implementation of this algorithm [20], which requires specifying minimum (min) and maximum (max) limits for the edge sharpness. These limits help distinguish between strong, weak, and non-edges, which is crucial for achieving accurate edge detection.

The Canny function classifies edges by comparing the magnitude of pixel gradients against the specified limits:

- **Strong edges** are those where the gradient magnitude exceeds the max limit. These are immediately considered true edges.
- **Weak edges** are those with gradient magnitudes between the minimum and maximum limit. These edges are only retained if they are connected to strong edges.
- **Non−edges** : Edges with a gradient magnitude below the min limit are discarded.

Three examples, A, B, and C, are given in Figure 2.9. The lines represent gradient values for an edge in an image. A is above the max limit, which means it has a strong edge. B is between the max and min limit but is connected to gradients over the max limit, which means it is a weak edge. Case C is below the max limit and not connected to any pixel with gradients above the max limit, which means it is a non-edge.

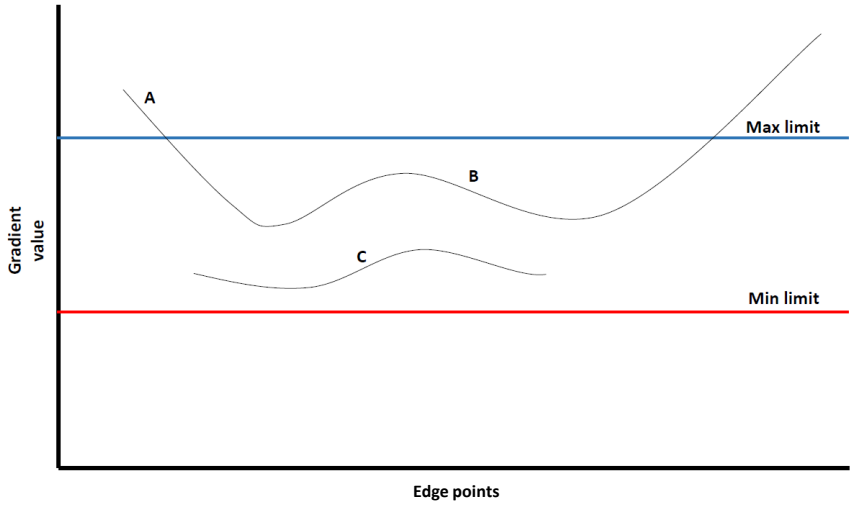

Figure 2.9: Canny edge examples of different scenarios when pixels are considered edges.

2.2.3 Hough circle algorithm

The Hough circle algorithm [21] is a computationally efficient way of detecting circles in an image. The algorithm connects edge points to possible circles and detects patterns [11, pp. 477-481]. The edge points are given from the output image of the Canny edge detector. Circles are described by the formula:

$$(x-a)^2 + (y-b)^2 = r^2.$$

Where a and b are the x and y coordinates for the center of the circle and r is the radius. The aim of the algorithm is to find (a, b) and r that best match the circles in an image. A radius interval is selected, and then every possible center point is calculated with the combination of edge point (x, y) and radius r using the formulas:

$$a = x - r \cdot \cos(\theta),$$

$$b = y - r \cdot \sin(\theta),$$

where θ is a range of angles between 0 and 2π.

If a position (a, b) for a given radius, r, has a number of edge points over the selected threshold, that position has been voted the center point with that given radius.

2.3 Dataset

Datasets play an important role in the development and evaluation of ML models. When designing a neural network, they are used to train the model to recognize patterns in a dataset. Datasets can also be used to validate the performance of an algorithm or model to quantify its performance. This section provides background on synthetic datasets and the necessity of manual labeling in the context of real-world applications.

2.3.1 Synthetic dataset

Synthetic data and its use in computer vision tasks were first investigated in [3]. The paper looked into the use of synthetically created data to test the reliability of autonomous programs under various situations without the need for extensive real-world data collection. With the advances in ML models, synthetic datasets have become increasingly important. The opportunity to create large scalable datasets is beneficial since it removes the need for the time-consuming process of manually labeling the dataset required when training complex models.

2.3.2 Manually annotating a dataset

Manual annotation of datasets involves human labeling of the data. The annotated dataset serves as the ground truth for training and testing the models. The quality of the annotated training dataset determines how well the model performs when applied to its intended task. The dataset also serves to evaluate the performance of the model, and without it, quantifying the performance metrics would be impossible.

2.4 Video mixer

This section provides an overview of the video mixer and the background information about the system. The video mixer is the system that combines the input video feed with information about projectile and explosion locations by adding an overlay with the projectile trajectory.

2.4.1 User interface

The video mixer system has a user interface that enables users to try the system. A wide array of settings can be chosen, including movement, firing frequency, maximum projectile distance, fall before explosion, and more. The output video is in analog format and can be displayed on a connected screen.

2.4.2 Video feed

One component of the video mixer system is the source of the video feed, which can originate from either a computer or a camera. The perspective from which the video feed is captured will be referred to as the camera view. The camera has several

parameters to be considered when calculating the camera view. The first one is the field of view (FOV), which specifies the angle that the camera lens can encompass, determining how much of the world is seen through the camera. This parameter is critical because it affects the location of the projectile in the camera view.

Knowing the number of pixels on the screen in the horizontal and vertical direction denoted Height and Width as well as the FOV in both horizontal (FOV_H) and vertical (FOV_V) directions enables the calculation of the focal length denoted as f_x and f_y. The focal lengths are essential for calculating the 2D position of the projectile in the camera view. The focal length can be calculated as:

$$f_x = \frac{\text{Width}}{2 \times \tan\left(\frac{\text{FOV}_H}{2}\right)}, \tag{2.23}$$

$$f_y = \frac{\text{Height}}{2 \times \tan\left(\frac{\text{FOV}_V}{2}\right)}. \tag{2.24}$$

The focal length is the same in both the horizontal (f_x) and vertical (f_y) directions if the pixels taken by the image sensor are square. This uniformity ensures that the scaling is consistent in both dimensions, so it is only necessary to calculate either f_x or f_y.

2.4.3 Controller Area Network Bus

A CAN bus is a high-integrity serial bus system designed for high-speed, reliable communication among multiple intelligent devices [2].

Each device connected to the CAN bus has a CAN controller chip. The chips are responsible for handling all communications. Devices on the network receive all transmitted messages and independently decide to either process or ignore them based on relevance. Every transmitted message has an identifier that states the type of message. Every device has an acceptance filter that allows it to process the message if it passes through the filter.

To prevent a collision when multiple units try to transmit messages simultaneously, CAN has an arbitration process based on the message priority specified in the identifier. The message with lower priority is delayed until the previous message has been transmitted.

2.5 Performance metrics

Different metrics are used to evaluate the performance of an object detection model. The performance metrics enable quantitative assessment of the model's ability to detect objects correctly. Depending on the purpose of the model, the importance of the different performance metrics varies. Some standard performance metrics for object detection are listed below.

To calculate the performance metrics, the terms false positive (FP), false negative (FN), and true positive (TP) are used.

- FP is the number of instances where the model incorrectly predicts an object that is not actually present.

- FN is where the model fails to predict an object that is present.
- TP is where the model correctly predicts the presence of an object.

Examples of FP, FN, and TP predictions are shown in Figure 2.10

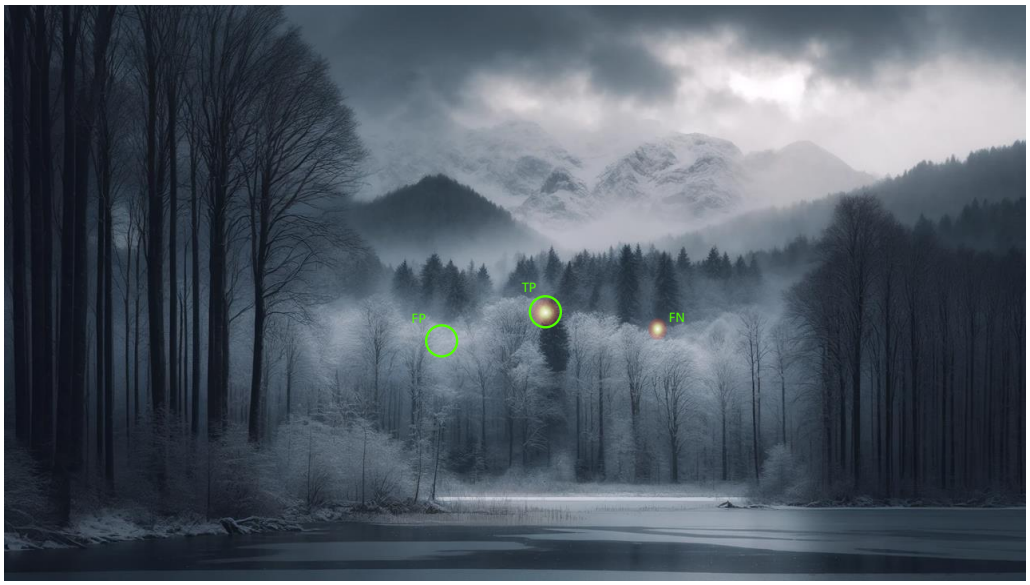

Figure 2.10: Examples of FP, FN, and TP predictions where the green circles are the predictions.

Precision

Precision is a metric that quantifies the accuracy of the positive predictions made by the model. It is a useful metric for understanding how many of the detected objects were actually correct. Precision is the ratio between the correct predictions and the total number of predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall

Recall measures the ability to detect all the objects and helps to understand the number of false negative predictions. It is calculated using:

$$\text{Recall} = \frac{TP}{TP + FN}$$

3
Methods

This chapter outlines the methodologies employed in this project to detect projectiles and explosions in recorded videos. It also explains the process of transforming recorded messages from a CAN bus to projectiles and explosions on a camera view that can assist in detecting the projectiles and explosions.

The techniques applied for object detection include the Faster R-CNN, an ML model for feature detection, and the Round Object Detector, a model that uses the Hough circle algorithm to detect circular objects. These two detection models were designed to locate projectiles and explosions in frames from the video mixer.

Additional pre- and postprocessing tools were utilized to enhance the accuracy of these models.

The chapter is structured into several key sections. Section 3.1 details the process of assembling and preparing the data necessary for training and validating the models; Section 3.2 describes the equipment and settings used to simulate and record video scenarios; Section 3.3 explains the algorithms and techniques used to identify projectiles and explosions within the videos; Section 3.4 discuss the methods used to calculate the expected true positions of the projectiles and explosions; Section 3.5 explores how knowledge about the calculated positions can improve detection performance.

3.1 Creating a dataset

Two datasets were created to train and validate the performance of the detection models. A synthetic dataset for training the model and a manually annotated dataset for validating the detection models. The predictable appearance of projectiles and explosions facilitated the use of synthetic datasets to simulate a manually labeled dataset comparable to one that would be obtained from the video mixer. Employing a synthetic dataset for training circumvented the labor-intensive process of manually labeling positions in each frame captured from the video mixer.

This approach saved time but also allowed for generating a larger and more varied dataset, which is crucial for training robust detection models.

A manually annotated dataset with three different backgrounds was also created to validate the models' performance in realistic scenarios. This made it possible to quantify the performance and investigate how to improve it.

3.1.1 Background selection

The video mixer system operates in a variety of training environments. Therefore, the backgrounds for the dataset must comprehensively represent the range of scenarios in which the system might operate. To ensure the robustness and applicability of the detection models, the backgrounds selected for the synthetic dataset need to accurately reflect real-world environments.

A dataset originally published by Intel for a natural scene classification competition on Analytics Vidhya was chosen for the backgrounds to address the requirement of backgrounds reflecting real-world environments [22]. This dataset was selected for its wide range of environment categories, each representing unique challenges relevant to the training needs of the models. The dataset includes six different categories:

- **Buildings**
- **Forest**
- **Glacier**
- **Mountain**
- **Sea**
- **Street**

3.1.2 Projectile and explosion

The video mixer generates the appearance of the projectile in C++. The projectile is depicted as a round object with a gradient transitioning from bright white at the center to transparent red at the edges, as illustrated in Figure 3.1a. This image was saved in the BGRA format using OpenCV's 'imwrite' function [23] and stored as a PNG file.

The explosion image was generated by isolating the upper half of the projectile image, which was then saved as a separate PNG file, as shown in Figure 3.1b.

(a) Projectile animation **(b)** Explosion animation

Figure 3.1: Images of the two different object animations.

The original resolutions of the projectile and explosion images were 100x100 and 100x50 pixels, respectively. However, in the video mixer recordings, the actual sizes of these objects are significantly smaller. To make the projectiles and explosions fit on the backgrounds in the dataset, they were reshaped to more accurately represent the size that they would have on a video mixer frame.

3.1.3 Synthetic dataset creation

The synthetic dataset was constructed by overlaying the generated projectiles and explosions onto the set of background images at random positions and sizes. The information about the position, bounding box, and label were stored in a corresponding text file to each image. Examples of the dataset images are displayed in Figure 3.2 together with the bounding boxes from the corresponding text files.

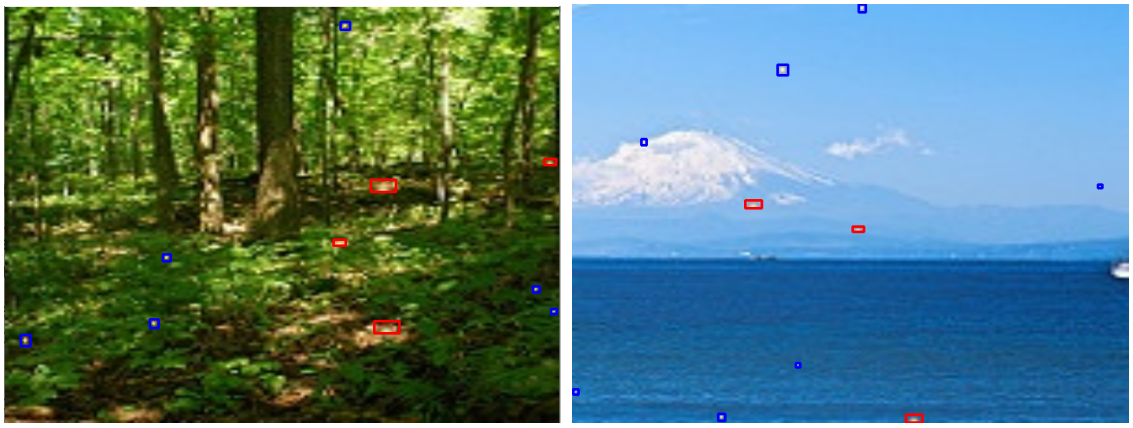

Figure 3.2: Two examples from the dataset with the corresponding bounding boxes with blue being projectiles and red explosions.

3.1.4 Data augmentation

In the video mixer, projectiles and explosions may not always appear perfectly shaped due to recording quality, resolution discrepancies, and imperfections in converting an analog image to a digital image. To accommodate this and also reduce the risk of overfitting the Faster R-CNN model on the synthetic dataset augmentation was applied. A new height for the projectiles of between 70% and 100% was randomly chosen, and then the projectiles were rotated between 0 and 2π. The new height and rotation made the projectiles deformed and could better represent how the projectiles looked in the video recording.

3.1.5 Manual labeling of video mixer frames

To determine the performance of the models on the video mixer, a dataset of labeled video frames was necessary to create. Due to the limitation stemming from security requirements, using online labeling tools was not an option, and the dataset had to remain on the local unit. Therefore, a Python program was created that took an image frame as input and enabled the user to create bounding boxes by clicking on the image. The resulting bounding boxes were stored in a corresponding text file identical to the layout of the synthetic dataset, which enabled a smooth transition to using the new dataset. The dataset's purpose was not to train the model but to validate the performance, which limited the amount of labeled data required.

Three different backgrounds were chosen before starting to label the frames. The backgrounds were supposed to be of varying difficulty levels for the models. The

difficulty varied depending on how detailed and colorful the background was. The first was a completely black background that provided minimal visual complexity and tested the models' ability to detect objects without background noise. The second image shows a mountain in winter with gray and darker colors. Its gray, textured landscape provides a moderate challenge, which might make it slightly more challenging to detect the objects. The third was an image of a colorful valley with a more detailed landscape than the previous two backgrounds. It presents the highest difficulty level, challenging the models to distinguish the objects from various visual features and potentially confusing patterns.

(a) Black background

(b) Gray mountain background

(c) Colorful valley background

Figure 3.3: The three different video mixer backgrounds generated by Dall-E.

After the videos were recorded with the different backgrounds, a set of frames was stored in a separate folder for manual labeling. The selected frames had a corresponding CAN data position update message. Selecting those frames enabled the projectile detection performance to be tried while using the knowledge from the calculated position from the CAN data projectile update.

3.2 Video mixer

A standardized setting was employed for all recordings using the video mixer to facilitate comparative performance analysis.

- **Max horisontal projectile distance** : **500m**
- **Max projectile drop** : **20m**
- **Fireing vertical angle** : **0rad**
- **Initial speed** : **500m/s**
- **FOV** : **0.1875rad**

The output analog video from the video mixer was recorded using the Elgato video recorder [24]. The recorded video was saved as an mp4 file, which enabled the video to be processed in Python.

3.3 Projectile and explosion detection

This section outlines the two different methods employed for object detection, each selected due to their unique capabilities. It describes the implementation of both the ML-based Faster R-CNN model and the algorithmic Round Object Detector.

3.3.1 Faster R-CNN

The Faster R-CNN model has an implementation in the open-source PyTorch library that facilitates the implementation and customization [14]. The required sections of the model could be installed locally and then integrated into the existing project codebase.

Pretrained weights were used to speed up the training process and enhance the initial performance of the model [25]. The pretrained weights were trained on the COCO dataset, a comprehensive object detection dataset created by Microsoft [26].

These pretrained weights are beneficial since they have already been trained on a diverse set of images, allowing the model to recognize a broad spectrum of features from the outset. Utilizing these pretrained weights reduces the amount of time required for training the model to detect projectiles and explosions instead.

3.3.2 Round Object Detector

Implementing the Round Object Detector required several image processing steps. Each image had to be passed through a color filter. After the filtering, the edges had to be extracted before the Hough circle algorithm could be used. An overview of the process is illustrated in Figure 3.4.

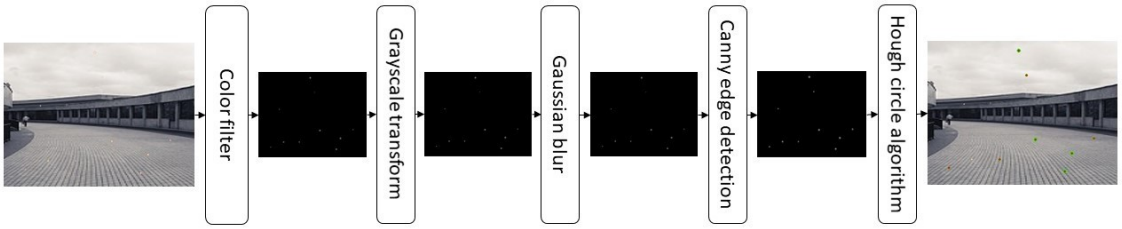

Figure 3.4: Steps in the Round Object detection model.

HSV color filter

A HSV color filter was implemented to extract the projectiles and explosions from the input image. The first step was to create a program that could read HSV pixel values from a recording of the video mixer to know the HSV color interval to let through. The two primary colors of the projectile and explosion, red and yellow, require one color filter each. The Hue is between 0 - 180, and both saturation and value are between 0 - 255, as seen in Figure 2.8. The specific ranges for hue, saturation, and value were determined from the analysis of the video mixer footage and are summarized in Table 3.1.

Table 3.1: HSV color filter

	Lower red limit	**Upper red limit**	**Yellow limit**
Hue	0 - 25	155 - 179	30 - 35
Saturation	25 - 80	25 - 80	50 - 90
Value	90 - 255	90 - 255	190 - 235

Grayscale transform

Following the color filtering, the image mask returned from the color filter was converted to grayscale. This conversion is crucial for the edge detection phase where the Canny edge detector was applied.

Gaussian blur

To reduce noise that can affect edge detection accuracy, a Gaussian blur was applied to the grayscale image. This step smoothened the image, reducing the impact of pixel-level variations irrelevant to the circle detection.

Canny edge detection

The final preprocessing step involved applying the Canny edge detector to the output from the Gaussian blur filter. The output from the Canny edge detector could then be fed into the Hough circle algorithm to create the predictions.

Hough circle algorithm

The output edges from the Canny edge detector were the input to the Hough circle algorithm. OpenCV has an implementation of the algorithm that simplified the implementation in the model [27].

3.3.3 Shared performance function

Evaluating the performance of the Faster R-CNN model and the Hough circle algorithm required a shared way to calculate how well the models locate the projectiles on the screen.

The true center point was compared to the corresponding prediction from both models. The distances between each predicted and true center position were calculated to find the corresponding prediction for each projectile. With the distances, it was possible to determine which predicted center point corresponded to which true position. If the predicted points do not match the true positions, that could be because of an FP or an FN prediction. FP means that the model predicts that there is either a projectile or explosion where there is not, and FN is when a projectile or explosion is not detected. FP and FN are penalized in the loss function by multiplying the number of FP and FN occurrences with the penalty constants λ_{FP} and λ_{FN}. The penalty constants were selected to resemble the importance of each metric and how hard each error should be penalized. The loss, L, is calculated using the equation:

$$L = \sum_{i=1}^{\text{TP}} \delta + \lambda_{\text{FP}} \cdot \text{FP} + \lambda_{\text{FN}} \cdot \text{FN}. \tag{3.1}$$

The penalty constants were chosen relative to the width of the image (640 pixels). The factor was chosen to $\lambda_{\text{FP}} = 0.2 \cdot 640 = 128$ to have a balance between penalizing FP and FN predictions as well as offsets between predicted and true center points.

3.4 True position calculations

To accurately validate the detected positions of projectiles or explosions, their true positions on the screen must be determined. This can be achieved using triangulation [11, pp. 701-702].

Figure 3.5 displays a simplified model of the system. The coordinate system originates in the cannon where z is in the pipe direction, the y-axis is horizontal, and the x-axis is downwards.

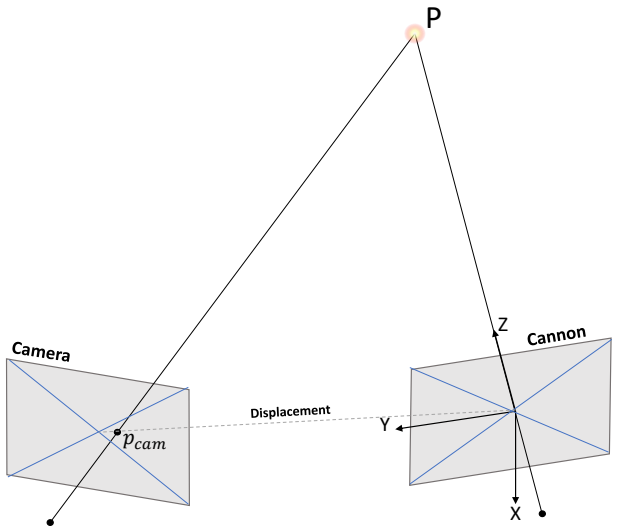

Figure 3.5: Displays a simplified view of the camera and Cannon setup and their coordinate system.

The properties of the camera, the 3D position, $\mathbf{p}$, and the relative displacement between the camera and the cannon need to be known to calculate the camera position, $\mathbf{p}_{\text{cam}}$.

The focal length is the same in both horizontal (f_x) and vertical (f_y) directions because the pixel shapes captured by the image sensor are square and therefore the scaling is uniform in both directions. The focal length f of the camera was calculated using the vertical image size, Height, and the vertical field of view, FOV_V using the eq. 2.24:

$$f = f_y = \frac{\text{Height}}{2 \cdot \tan\left(\frac{\text{FOV}_V}{2}\right)}.$$

With the known focal length, and assuming that the center of the image sensor is at coordinates $c_x = \text{Width}/2$ and $c_y = \text{Height}/2$, the intrinsic matrix is defined as:

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}.$$

The displacement of the camera is split up into rotation and translation relative to the global coordinates of the system. The rotation matrix $\mathbf{R}$ is calculated by the equations below:

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix},$$

$$\mathbf{R}_y(\psi) = \begin{bmatrix} \cos\psi & 0 & -\sin\psi \\ 0 & 1 & 0 \\ \sin\psi & 0 & \cos\psi \end{bmatrix},$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{R} = \mathbf{R}_x(\phi) \cdot \mathbf{R}_y(\psi) \cdot \mathbf{R}_z(\theta).$$

The translation vector, $\mathbf{t}$ is defined as:

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}.$$

Given the known 3D positions, $\mathbf{p}$, and the relative displacement between the camera and the cannon, calculating the camera view position, $\mathbf{p}_{\text{cam}}$, was possible using Eq. (3.2):

$$\mathbf{p}_{\text{cam}} = \mathbf{K}[\mathbf{R},\ \mathbf{t}]\mathbf{p}. \tag{3.2}$$

The matrix $[\mathbf{R},\ \mathbf{t}]$ is a concatenation of the rotation matrix, $\mathbf{R}$ and the translation vector, $\mathbf{t}$.

3.4.1 CAN

The NI-XNET bus monitor was the software used to record the CAN data [28]. The data was saved to a text file for postprocessing in Python. The CAN data consists of channels with information about the projectiles, explosions, and the parameters chosen in the user interface. Each message also has a timestamp that was used to connect messages to specific frames in the video recording.

3.4.2 Synchronizing monitored CAN data and recorded video

Due to discrepancies in the starting times of the monitored CAN traffic and the recorded video output, it was essential to align the two sources accurately. Synchronizing the data ensured that each positional update message in the CAN system had a corresponding video frame.

The synchronization process utilized the Faster R-CNN model to detect the first appearance of the projectile in the video. The detected position of the projectile in the initial frame was then compared with the calculated initial position from the

CAN data. If the positions matched within a predefined threshold, the corresponding video frame was linked to the timestamp from the recorded CAN data, synchronizing the two data streams.

The video's frame rate exceeds the message frequency from CAN, resulting in frames that lack corresponding positional information from the CAN system. The location of the projectiles and explosions are instead extrapolated to have a position update for every frame.

However, due to the lack of direct CAN data for these extrapolations, only frames that coincide with specific intervals specified in the CAN data were analyzed in conjunction with the model.

3.5 Selective detection using the calculated true position

Selective detection leverages the expected positions of projectiles and explosions, as calculated from CAN data, to enhance the accuracy of object detection models within video frames. This method is designed to either filter out or add detections depending on the distance to the positions calculated from the CAN data. The two models had to use the selective search in different ways.

3.5.1 Improving the Faster R-CNN model with selective detection

The Faster R-CNN model outputs the bounding box, label, and confidence score for every prediction. The confidence score states how sure the model is that the bounding box and label are correct. Selective detection is integrated by adjusting the confidence scores of its predictions based on their proximity to the expected positions from the CAN data.

If the confidence output of a prediction from the model was under the confidence threshold, it could be increased if the detection is in proximity to the calculated CAN position. Increasing the confidence of detections close to the expected position would decrease the rate of FN predictions, increasing the recall. Because of the confidence increase for predictions close to the expected position, it also enabled an increase of the confidence threshold, which would result in fewer FP predictions and, therefore, increase the precision.

To accomplish this, the distance, δ_{center}, between the centerpoints of the prediction and the expected position from CAN was calculated. To update the confidence of the predictions, the following equation was used:

$$\text{confidence}_{\text{new}} = \min(1, \text{confidence}_{\text{old}} + \frac{\gamma}{\delta_{\text{center}}}),$$

where γ is a number that determines how important it is that the prediction is close to the CAN data position.

3.5.2 Improving the Round Object Detector with selective detection

Unlike the Faster R-CNN model, the Round Object Detector has no confidence score for its predictions. Without a confidence parameter, another approach had to be used to improve the accuracy. The Round Object Detector instead used the CAN data information to filter out unreasonable predictions. First, the distance between the predicted center point and the CAN data center point, δ_{shift}, was calculated:

$$\delta_{\text{shift}} = \sqrt{(P_{\text{pred}_x} - P_{\text{CAN}_x})^2 + (P_{\text{pred}_y} - P_{\text{CAN}_y})^2}.$$

Then, if δ_{shift} is higher than a predefined threshold, the prediction was filtered out. The threshold was selected by testing how small the threshold could be without removing too many TP predictions.

4

Results and Discussion

This chapter presents and discusses the results of this project. It is divided into two sections: the first evaluates the models using the synthetic datasets, and the second assesses their performance on the video mixer frames. Additionally, the effectiveness of selective detection is examined.

4.1 Synthetic dataset

The resulting test performance of the models on the synthetic datasets is displayed in Table 4.1 and Table 4.2. The metrics to compare the performance are precision, recall, and loss for the entire test dataset. The calculated loss from Eq. (3.1) is averaged out over the dataset shown in the table.
The different datasets are listed below and consist of 2499 labeled images each:

- **Dataset 1** : A dataset including only the projectiles and no shape augmentation
- **Dataset 2** : A dataset including only the projectiles with shape augmentation
- **Dataset 3** : A dataset including both projectiles and explosions with no shape augmentation
- **Dataset 4** : A dataset including both projectiles and explosions with shape augmentation

Faster R-CNN

The learning curves when training the Faster R-CNN model for 10 epochs on the different datasets are shown in Figures 4.1 to 4.4.

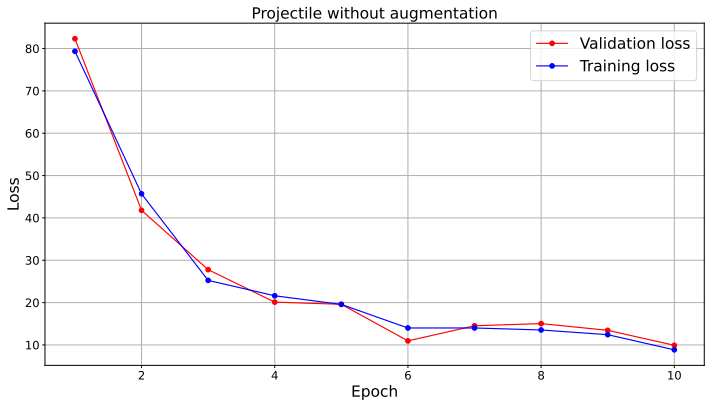

Figure 4.1: Learning curve for the dataset with only projectiles and no augmentation.

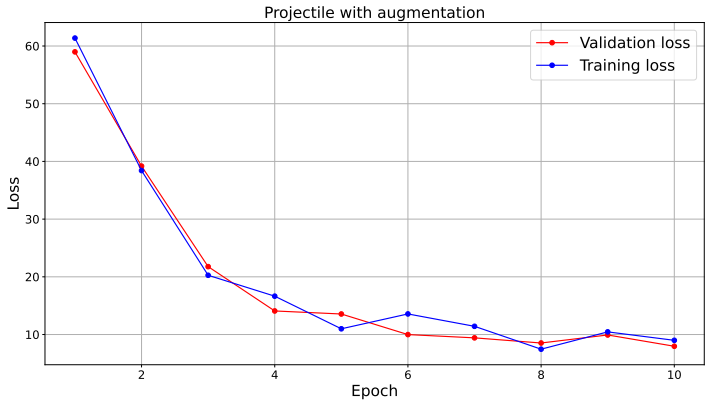

Figure 4.2: Learning curve for the dataset with only augmented projectiles.

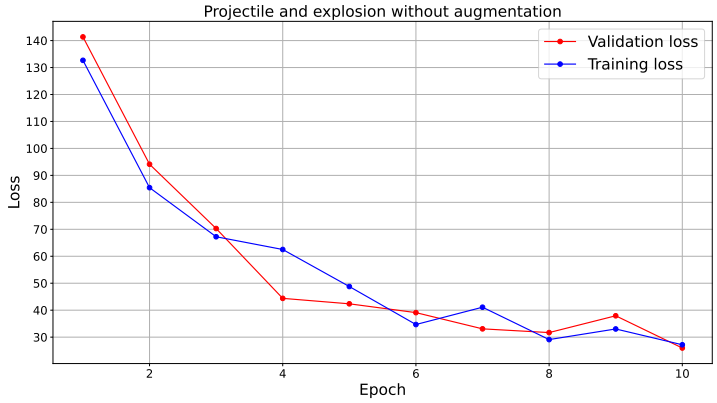

Figure 4.3: Learning curve for the dataset with projectiles and explosions without augmentation.

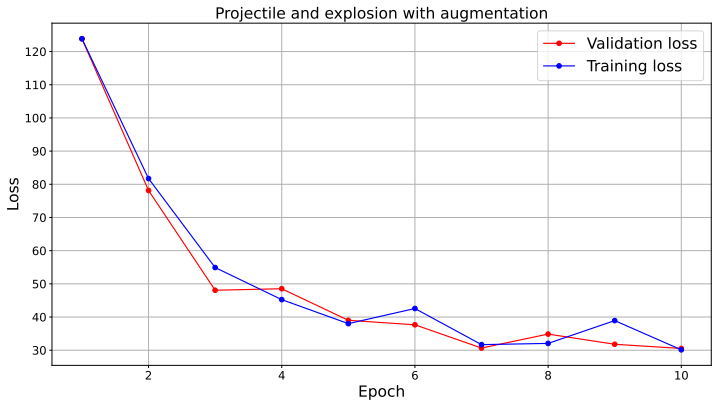

Figure 4.4: Learning curve for the dataset with projectiles and explosions with augmentation.

As it can be seen in the learning curve graphs, the model learns to detect projectiles and explosions relatively quickly and has almost stabilized after ten epochs. The validation loss follows the training loss, which indicates that the model has not overfitted the training data. Table 4.1 shows the precision, recall, and average loss on the four different datasets.

Table 4.1: The test precision, accuracy, and loss for the Faster R-CNN model on the four different synthetic datasets

Faster R-CNN	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Precision	99.84%	99.84%	99.36%	99.68%
Recall	99.48%	99.64%	98.25%	97.89%
Average loss	10.40	8.48	35.21	35.60

The precision and recall for the Faster R-CNN model on the synthetic test dataset are close to 100%, which means the model can learn to detect projectiles and explosions. The reason why the performance on datasets 3 and 4 is slightly worse than 1 and 2 is because of the added complexity of two different class labels. In such cases, an object can either be a projectile or an explosion, which increases the complexity of the predictions.

Round Object Detector

Table 4.2 shows the precision, recall, and average loss on the two datasets only containing the projectiles.

Table 4.2: The test precision, accuracy, and loss for the Round Object Detector on the two different synthetic datasets

Round object detection	Dataset 1	Dataset 2
Precision	31.26%	30.15%
Recall	44.34%	41.51%
Average loss	1695.92	1684.80

The performance of the Round Object Detector was not nearly as good as that of the Faster R-CNN model on the synthetic datasets. One of the main reasons for the lack of performance is the color filter. For the Round Object Detector to work well, the color filter must remove nearly everything from the image apart from the projectiles. The performance of the color filter is dependent on the background. If the background image has HSV-color values in the same interval as the projectile, then those regions are let through the filter, creating problems for the algorithm when locating the round objects. An example of when this occurs can be seen in Figure 4.5.

Figure 4.5: Example of noise that gets through the color filter

Another reason why the color filter does not perform perfectly is because the projectiles are partially transparent around the edges. The HSV values of those pixels values are affected by the background pixels. If the background has a distinct color, then parts of the projectile will not be let through the filter, and the projectile loses its round shape. When it is no longer circular, the algorithm will not recognize it as a projectile, and the accuracy decreases.

A third reason for the lack of performance could be because the projectiles become very small when far away from the shooter. When the projectiles become smaller, they consist of fewer pixels, making them less round and, therefore, harder to detect.

4.2 Object detection on video mixer recording

This section evaluates the performance of the object detection models using datasets annotated from video mixer recordings under varying background conditions.

The performance metrics (precision, recall, and loss) for the model are compared for both models on three different datasets. Each dataset is listed below:

- **Background 1** : A dataset of frames from the video mixer with black background shown in Figure 3.3a
- **Background 2** : A dataset of frames from the video mixer with the gray mountain background shown in Figure 3.3b
- **Background 3** : A dataset of frames from the video mixer with the colorful valley background shown in Figure 3.3c

Faster R-CNN

Tables 4.3 and 4.4 show the precision, recall, and average loss per frame for the Faster R-CNN models trained on datasets 3 and 4. Three different datasets are analyzed with frames from the video mixer recordings, where each dataset has a different background image.

Faster R-CNN trained with none augmented dataset The performance of the Faster R-CNN model trained on the non-augmented dataset is summarized in the table below. This dataset did not include any shape augmentation for the projectiles and explosions.

Table 4.3: The test precision, recall, and loss for the Faster R-CNN on frames from the three different videos

Faster R-CNN	Background 1	Background 2	Background 3
Precision	100.0%	97.92%	100.0%
Recall	93.75%	97.92%	75.51%
Average loss	11.51	6.27	32.69

Table 4.3 illustrates how the model's detection performance varies across different background complexities. Notably, the model demonstrates robust precision across all backgrounds, though recall varies, indicating challenges in consistently identifying all relevant objects, particularly against the most complex background (Background 3).

Faster R-CNN trained with augmented dataset Table 4.4 shows the performance of the Faster R-CNN model trained on the dataset with augmented projectiles and explosions.

Table 4.4: The test precision, recall, and loss for the Faster R-CNN on frames from the three different videos

Faster R-CNN	Background 1	Background 2	Background 3
Precision	100.0%	97.92%	100.0%
Recall	93.75%	97.92%	79.59%
Average loss	8.80	6.27	27.35

The result indicates a slight improvement in recall for background 3 compared to the model trained on the non-augmented dataset. The improvement suggests that the model, when trained on an augmented dataset, is slightly better equipped to handle variations in projectile and explosion shapes, enhancing its ability to perform on the video mixer frames.

Performance transfer from the synthetic datset

The knowledge gained from the synthetic dataset was transferred well to the video mixer frames. The precision is slightly higher on backgrounds 1 and 3 and lower on background 2 compared to the performance on the synthetic dataset. The slight increase for two background datasets could be explained by the lack of background features resembling either a projectile or an explosion, resulting in fewer FP predictions. The recall is down for all three background datasets, which was to be expected. The Synthetic dataset would not mimic the video mixer frames perfectly, resulting in a model that is not completely optimized for the task. Background 1 and 2 still have a high recall, especially comparing the performance to the Round Object Detector in Table 4.6.

Faster R-CNN with selective detection

Since the model trained on the dataset with augmentation performed slightly better on the video mixer frames, that model is used when applying the selective detection method.

Table 4.5: The test precision, recall, and loss for the Faster R-CNN on frames from the three different videos using selective detection

Faster R-CNN	Background 1	Background 2	Background 3
Precision	100.0%	97.92%	100.0%
Recall	97.92%	97.92%	81.63%
Average loss	3.48	6.27	24.69

The selective detection method improved the recall rates across the video frames by effectively reducing FN predictions. By increasing the confidence for detections near expected projectile positions from the CAN data, the model creates more true positive predictions.

The biggest improvement can be seen in background 1. With a black background, the models should be able to detect an object in the position of the projectile or

explosion. When the projectiles and explosions are small, the confidence level of the prediction might be quite low. Using selective detection could increase the confidence to exceed the threshold and make the prediction.

The performance also increased in background 3, which could be explained by the fact that the more difficult background could decrease the prediction confidence, which the selective detection method might counteract.

Round Object Detector

Table 4.6 shows the precision, recall, and average loss per frame from the video mixer recordings with the three different backgrounds.

Table 4.6: The test precision, recall, and loss for the Round Object Detector on frames from the three different videos

Round Object Detection	Background 1	Background 2	Background 3
Precision	95.35%	90.48%	1.60%
Recall	85.41%	39.58%	6.12%
Average loss	25.80	83.95	528.83

The results show the performance is significantly better on frames with a black background than the other two.

A flawed color filter could explain the performance difference for the different backgrounds. With a black background, the only things passed through the filter are the projectiles, which could be likened to a perfect color filter.

Background 2 does not share HSV values with the projectiles or explosions. The partially transparent edges of these elements blend with the background, altering their HSV values. This effect can prevent the projectiles from maintaining a complete and recognizable shape after the filter, impacting detection accuracy as parts of the projectiles get removed.

Background 3 is the most challenging due to its vibrant colors. The diverse range of hues often passes through the color filter unintentionally, leading to a significant increase in both FP and FN predictions. This scenario shows the limitations of the current color-filtering approach when faced with complex backgrounds.

Round Object Detector with selective detection

Table 4.7 shows the performance of the round object detection model using the selective detection method where all predictions further than 10 pixels away from the CAN data-position are removed.

Table 4.7: The test precision, recall, and loss for the Round Object Detector on frames from the three different videos using selective detection

Round Object Detector	Background 1	Background 2	Background 3
Precision	100.0%	100.0%	100%
Recall	83.33%	37.5%	4.08%
Average loss	22.58	80.90	125.45

The results indicate a notable increase in precision across all backgrounds. Selective detection effectively removes most FP predictions by ensuring that only detections close to the calculated CAN positions are considered valid.

However, the decrease in recall suggests that the synchronization between the CAN data and the video frames is not always accurate. Some correct detections are likely being discarded because their calculated positions from the CAN data do not align closely enough with the actual detections in the video frames. This misalignment might be due to timing errors between frames and CAN messages.

The improvement in recall is particularly significant in backgrounds 2 and 3, where the complex background creates a significant amount of FP predictions.

However, since selective detection does not create more TP predictions for the Round Object Detector as it does for the Faster R-CNN, the performance improvement is still not enough to close the performance gap with the ML-based method.

5
Conclusion

During this project conclusions could be drawn for both object detection models that were used. The performance of the two models depended significantly on the type of background the recorded video used. The most significant metric affected by the background was the recall, which means the rate of TP predictions changed.

Even though the Faster R-CNN model was only trained on a synthetic dataset, it outperformed the Round Object Detector in all categories. The discrepancy was the largest when the background image consisted of pixel values that got through the color filter. The Faster R-CNN did not have a higher rate of FP predictions with more detailed backgrounds, which signifies it is a more robust model than the Round Object Detector, better handling a wider range of background environments.

Both models performed worse when the projectiles got smaller. Since they consist of fewer pixels, they lost their circular shape and were harder to locate in the frames. The change in shape affected the Round Object Detector especially since it relies on the objects being circular. Even though the ML model was trained to detect small projectiles and explosions in synthetic data it still had problems when applied on the video frames. An explanation could be that the model has not learned how the object deforms in the video recordings.

Selective detection improved the performance of both models. It was most significant for the Faster R-CNN model because it could be implemented to affect the confidence level rather than only filtering out the detections like for the Round Object Detector.

To summarize, Faster R-CNN is the better option unless computational power is very limited and the background is not detailed. The synthetic dataset effectively served its purpose of training the Faster R-CNN model, yielding results that were good enough to consider it a viable alternative to manually labeled datasets. The advantages of using a synthetic dataset include reduced time spent on dataset creation and the flexibility to retrain the model with new animations and appearances of projectiles and explosions.

5.1 Future work

For further development of this project, some improvements can be carried out. The main objective would be to improve the detection performance of the Faster R-CNN model.

One improvement would be to create a manually annotated dataset of frames from the video mixer recordings. This dataset could then be used to fine-tune the model

and optimize the current weights that have only been trained on the synthetic dataset.

Another method would be to create a program that calculates a prediction of the object's location using the previously detected locations. This prediction could be used if the model is not able to detect any object in close proximity to the predicted position. This would enable the creation of a trajectory where the frames that have FN predictions use the predicted location instead. Examples of methods to accomplish this could be to use a Kalman filter [29].

Bibliography

[1] Saab AB, "Training and Simulation," Accessed: Feb. 22, 2024. [Online]. Available: https://www.saab.com/products/land/training-and-simulation.

[2] "Controller Area Network (CAN) Overview," National Instruments. Accessed on: Feb. 22, 2024 [Online]. Available: https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html#section--640856706.

[3] W. Burger and M.J. Barth, "Virtual Reality for Enhanced Computer Vision," in *Springer US, Boston, MA*, pp. 247–257, 1995. https://doi.org/10.1007/978-0-387-34904-6_19.

[4] "OpenCV - Open Source Computer Vision Library," OpenCV, 2024. [Online]. Available: https://opencv.org/. [Accessed: Jun. 11, 2024].

[5] "About the ISP," Swedish Inspectorate of Strategic Products, Accessed on: Feb.22, 2024 Published April 19, 2017, Last updated March 15, 2023. [Online]. Available: https://isp.se/eng/about-the-isp/.

[6] Utrikesdepartementet, "Lag (1992:1300) om krigsmateriel", Accessed: Feb. 22, 2024. issued on Dec. 10, 1992, amended up to SFS 2023:371, effective from Jan. 1, 1993. [Online]. Available: https://rkrattsbaser.gov.se/sfst?bet=1992:1300.

[7] The Editors of Encyclopaedia Britannica, "Deterrence," *Encyclopedia Britannica*, Dec. 17, 2023. [Online]. Available: https://www.britannica.com/topic/deterrence-political-and-military-strategy.

[8] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. on Machine Learning (ICML'10)*, Haifa, Israel, 2010,

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Cambridge, MA, USA: MIT Press, 2016.

[10] A. Ng, J. Ngiam, C. Y. Foo, et al., "Convolutional Neural Network UFLDL Tutorial," *Stanford*, September 2015. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[11] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., Springer, 2022, ISBN 123-4567890123. [Online]. Available: http://www.springer.com/gp/book/1234567890123

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Cambridge, MA, USA: MIT Press, 2016. p. 342

[13] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485*, 2020.

[14] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, doi: 10.1109/CVPR.2014.81

[16] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, doi: 10.1109/ICCV.2015.169

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 2015, arXiv preprint arXiv:1512.03385 [cs.CV].

[18] OpenCV, “Tutorial: Thresholding with InRange - OpenCV.” Available: https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html. [Accessed: 13-May-2024].

[19] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Dec. 1986, doi: 10.1109/TPAMI.1986.4767851.

[20] OpenCV, "Canny Edge Detector," in *OpenCV 3.4 Documentation*, 2023. [Online]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[21] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Commun. ACM*, vol. 15, no. 1, Jan. 1972. [Online]. Available: https://doi.org/10.1145/361237.361242

[22] Analytics Vidhya. "DataHack." [Online]. Available: https://datahack.analyticsvidhya.com/. [Accessed: 29-Apr-2024].

[23] OpenCV, "Image file reading and writing – OpenCV 4.x," 2024. [Online]. Available: https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html. [Accessed: April 15, 2024].

[24] Elgato. "Video Capture." Elgato, [Online]. Available: https://www.elgato.com/us/en/p/video-capture. [Accessed: 29-Apr-2024].

[25] Torch Contributors, "fasterrcnn_resnet50_fpn," PyTorch Vision Documentation, Copyright 2017-present. [Online]. Available: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html. [Accessed: April 17, 2024].

[26] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft COCO: Common Objects in Context,” 2015, arXiv preprint arXiv:1405.0312 [cs.CV].

[27] OpenCV, "Hough Circle Transform," OpenCV Documentation, Accessed on: Feb.23, 2024 [Online]. Available: https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html.

[28] National Instruments, "NI-XNET CAN, LIN, and FlexRay Platform Overview." [Online]. Available: https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/ni-xnet-can--lin--and-flexray-platform-overview.html. [Accessed: Feb. 27, 2024].

[29] C. Montella, "The Kalman Filter and Related Algorithms: A Literature Review," , May 2011. [Online]. Available: https://www.researchgate.net/publication/236897001_The_Kalman_Filter_and_Related_Algorithms_A_Literature_Review [Accessed: June. 17, 2024].

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY