



CHALMERS
UNIVERSITY OF TECHNOLOGY



ROMAn-DETR: We came, We sawed, We built a Robust Orientation-aware Model with Anisotropic attention

Increasing robustness of instance segmentation and designing machine learning solutions to reduce spill and decrease delays at sawmills

Master's thesis in Electrical Engineering

Johannes Johansson, August Wiklund

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

MASTER'S THESIS 2026

ROMAn-DETR:
We came,
We sawed,
We built a Robust Orientation-aware
Model with Anisotropic attention

Increasing robustness of instance segmentation and designing
machine learning solutions to reduce spill and decrease delays at
sawmills

Johannes Johansson
August Wiklund



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Computer vision
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

ROMAn-DETR:

We came,

We sawed,

We built a Robust Orientation-aware Model with Anisotropic attention

Increasing robustness of instance segmentation and designing machine learning solutions to reduce spill and decrease delays at sawmills

Johannes Johansson, August Wiklund

© Johannes Johansson, August Wiklund, 2026.

Supervisor: David Nordström, Department of Electrical Engineering

External Supervisor: Mattias Ulmestrand, Taigatech

Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2026

DEPARTMENT OF ELECTRICAL ENGINEERING

Division of Computer Vision,

Johannes Johansson

August Wiklund

Chalmers University of Technology

SE-412 96 Gothenburg

Telephone +46 31 772 1000

Typeset in L^AT_EX

Printed by Chalmers Reproservice

Gothenburg, Sweden 2026

ROMAn-DETR:

We came,

We sawed,

We built a Robust Orientation-aware

Model with Anisotropic attention

Increasing robustness of instance segmentation and designing machine learning solutions to reduce spill and decrease delays at sawmills

JOHANNES JOHANSSON

AUGUST WIKLUND

Department of Electrical Engineering

Chalmers University of Technology

Abstract

Objects in industrial sawmill environments tend to be in complex lighting scenarios, paired with them being highly geometrically anisotropic (long and thin) while lying close to several other similar objects. This makes them unsuitable for axis-aligned bounding boxes, which is a problem for segmentation and detection models.

To solve this we introduce our model ROMAn-DETR, an instance segmentation model made to handle anisotropic objects while also improving robustness for out-of-distribution data. By combining geometric priors for these anisotropic objects, together with a new auxiliary orientation module capable of predicting the axis of the objects' search space, the search space can be reduced for more efficient per-query sampling. Through extensive ablations we show that ROMAn-DETR both converges faster and also outperforms an equivalent state-of-the-art RF-DETR model in mAP by up to 10.3% on our out-of-distribution anisotropic dataset.

This paper also evaluates potential alternative approaches using a smaller head together with a DINO backbone which showed that while DINO is robust, it by itself not enough for instance segmentation tasks.

Keywords: computer vision, machine learning, sawmill, dino, rf-DETR, anisotropic, deformable attention

Acknowledgements

We would like to thank our supervisors David, Mattias and Matilda for helping out with our project and always being quick to answer any and all questions we have had. We would also like to thank all of Taigatech for being very welcoming and accommodating, it was a pleasure working for you these few months!

I would also like to thank my parents for helping and inspiring me to always learn new things. I would also like to thank my partner Ina for helping me to stay focused on my studies in times when other things may have been more appealing in the moment. Of course a big thank you to Johannes as well for being so easy to work with and a great overall thesis partner.

August Wiklund, Gothenburg, June 2026

I would like to dedicate this to my mum and dad for their continuous support and always being there all throughout my life, this would not have been possible without you. I also want to thank my brother and sister and wish you both the best of luck in your future.

Johannes Johansson, Gothenburg, June 2026

Contents

List of Figures	xiii
1 Introduction	1
1.1 Problem Statement	2
1.2 Scope and Limitations	3
1.3 Our Contributions	4
2 Background	5
2.1 Related work	5
2.1.1 Instance segmentation	5
2.1.2 RF-DETR	5
2.1.3 Foundational models and SSL	5
2.1.4 Frozen backbone	6
2.2 Timeline	7
3 Theory	9
3.1 Deep Learning	9
3.1.1 Stochastic Gradient Descent	10
3.1.2 Loss Function	10
3.1.3 Covariate Shift	11
3.1.4 Transformers	11
3.2 Computer Vision	12
3.2.1 Convolutional Neural Networks	14
3.2.2 Vision Transformers	15
3.2.3 Classification and regression	16
3.3 Deformable Attention	17
4 Methodology	21
4.1 Minimal Head approach	21
4.1.1 Frozen backbone	21
4.1.2 DINOv3 backbone with minimal head	22
4.1.2.1 Similarity	22
4.1.2.2 Logistic Regression	23
4.2 Anisotropic Deformable Attention with Propagated Angle Estimation	23
4.2.1 Orientation Vector Notation	23
4.2.2 Encoder-Initialised Orientation	24
4.2.3 Propagated Residual Angle Refinement	25

4.2.4	Canonical-Frame Sampling	25
4.2.5	Orientation Supervision	27
4.2.5.1	Ground-Truth orientation using PCA	27
4.2.5.2	Ground-Truth Validity Filtering	27
4.2.5.3	Geodesic L1 Loss	28
5	Results	31
5.1	DINOv3 backbone with minimal head	31
5.1.1	Similarity	31
5.1.1.1	Inserted Board	31
5.1.1.2	Manually Selected Board	32
5.1.2	Logistic regression	33
5.2	ROMAn-DETR Performance	33
5.2.1	Segmentation Model	33
5.2.2	Detection Model	34
5.2.3	Validation	36
5.3	Ablation: Angle Loss Coefficient	37
6	Discussion	39
6.1	DINOv3 backbone with minimal head	39
6.2	Model Evaluation	41
6.3	Future Work	42
7	Conclusion	43
	Bibliography	45
A	Appendix	I
A.1	Data	I

List of Figures

1.1	The input is an image that might contain boards or logs, the output is a list where each entry corresponds to a board or log.	2
1.2	Outline of the old pipeline for getting up-and-running at a new sawmill	2
1.3	Outline of the proposed pipeline for getting up-and-running at a new sawmill	3
2.1	The difference between object detection (a) and instance segmentation (b) is that instance segmentation gives a per-pixel classification of the image. This can be easily analysed to determine for example the shape or angle of an object	6
3.1	Two simple filtered images, one for edge detection and one for a box blur.	13
3.2	A closer look at a convolution with a simple filter.	14
3.3	Splitting ViT input image into patches	15
3.4	Patch embedding process	16
3.5	To determine what parts of an image contain a board, instead each <i>patch</i> can be classified. Here every patch that is more than halfway full of board is counted as board-containing.	16
3.6	The cosine similarity of the features of each patch to the features of a given patch. The patch each other patch is compared to is marked in red, a lighter color is a higher similarity	18
3.7	Cosine similarity between a user selected board patch and the other patches, a lighter patch means a higher similarity	18
3.8	Which green dot is most similar to the red one? It depends on how you define "similar"!	19
4.1	In theory, if the model perfectly labels both (a) and (b), it should also be able to label (c).	21
5.1	Visualizations of inserting a plank and using cosine and Euclidean similarity between patches, a brighter pixel corresponds to a higher similarity.	31
5.2	Visualizations of manually selecting a plank and using cosine and euclidean similarity between it and all other patches, the selected patch is marked in red and a brighter pixel corresponds to a higher similarity.	32

5.3	Logistic regression head trained to split the input image into <i>board</i> vs <i>not board</i> . A higher opacity green corresponds to a higher confidence in the patch being <i>board</i>	32
5.4	Comparison of mAP and mAP_{50} validation during training.	34
5.5	Comparison of validation loss during training.	36
5.6	Predictions of the ρ_β anisotropy ratio predictor per query in total (left). Predictions of the ρ_β anisotropy ratio predictor per query and decoder layer (right).	36
6.1	A messy bounding box	39
6.2	A simple binary classifier can have trouble differentiating between a single wide board and two boards lying next to each other, even when there is a clear border between the boards.	40
6.3	With thin boards there will be a significant amount of pixels outside of the actual board being labelled as <i>board</i>	40
6.4	Imagine a diagonal divide between a dark and light section. If we have a "patch resolution" of 2×2 we will get a very choppy edge, but if we produce two masks with "patch resolution" 2×2 where one is offset by 1 both horizontally and vertically, we can multiply them elementwise to simulate a mask with patch resolution 1×1	40
6.5	We obtain two foreground masks by shifting the original image before obtaining the DINO patches and then classifying each set of patches individually. These masks can then be multiplied together to obtain a mask with basically double the resolution.	41

1

Introduction

In the 2020s, every part of Swedish society is permeated by the use of AI and machine learning, and the forestry industry is no exception. In 2026, most Swedish sawmills implement machine learning algorithms in some form in at least one step of their factory chain. *Taigatech* is one of the companies that supplies these machine learning solutions. Taigatech develops two products, *SawTurn* and *Process Surveillance*; this thesis work is concerned with the latter.

Every modern sawmill has several cameras that monitor the facility. These cameras are connected to a control room where a human supervisor checks to make sure everything is running smoothly. Process Surveillance uses these same camera feeds as inputs to a computer vision model which can aid the human operator and, in some cases, take control over very simple tasks.

The purpose of Process Surveillance is to reduce spill and intervene before a small error turns into a big mess by sending alarms or controlling the conveyor tracks when certain conditions are met. These conditions include but are not limited to:

- boards lying at an angle
- boards lying too close to each other
- logs lying on top of each other
- too many logs lying in a specific region

Depending on the camera and region within the image from the camera, different conditions and thresholds are used; for example, when going into the planer, one of the later steps of the process of making a board, it is important that the boards lie neatly next to each other. In such a case the ruleset might be:

Measure	Limit	Action
board distance	< 10cm	Send an alarm to the operator
board distance	< 5cm	Stop the conveyor
board angle	> 5°	Stop the conveyor

Instead, the ruleset for logs that have just arrived on site and are lying in a big pile might be:

Measure	Limit	Action
# of Logs	> 40	Stop the previous conveyor
Log angle	> 45°	Reverse the conveyor
Log length	< 3m	Stop the conveyor and alert operator

All of these different constraints and thresholds require either a different solution for each scenario or a single board detector solution that can be configured depending on the scenario. Process Surveillance uses the second architecture. At runtime, an

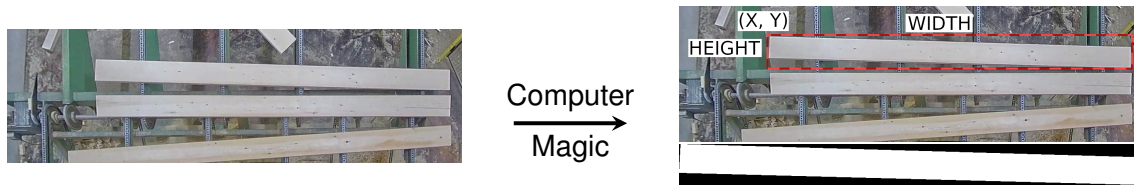


Figure 1.1: The input is an image that might contain boards or logs, the output is a list where each entry corresponds to a board or log.

image received from the camera is passed through a computer vision model to extract metrics for a bounding box around **each** visible board or log, such as bounding box height, width, coordinates, as well as a matrix representing a mask of the board within the bounding box (see Figure 1.1) which can be easily analyzed to get the necessary metrics for each board or log.

Computer vision is a very broad and active field of research within machine learning. Since vision is such a big part of the lived experience of humans, it makes sense for us to create computer programs that use visual data as input. However, this is not an easy task and countless papers are being written every year developing new techniques and methods. For more about computer vision, see Section 3.2.

1.1 Problem Statement

Previously, there has been a significant start-up time for every new process surveillance site (see Figure 1.2) which can hopefully be reduced drastically. The main time sink has been the human labelling done by an external company which can take several weeks but is necessary for accurately training a model on footage from a new sawmill.

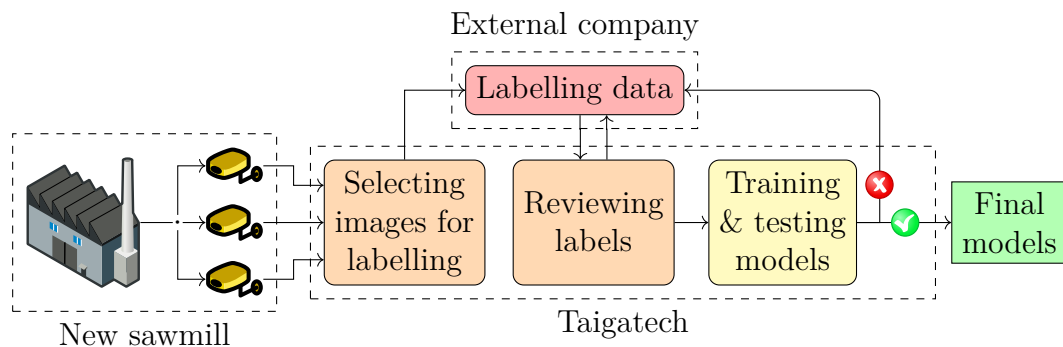


Figure 1.2: Outline of the old pipeline for getting up-and-running at a new sawmill. Normally, the bulk of the real world time is concentrated within collecting footage from the new sawmill and the external company labelling the selected data. Note that each camera trains its own model so for a sawmill with six cameras the startup time is effectively sextupled.

Another big factor in the amount of time it takes is how much footage needs to be collected in order to obtain an adequate training dataset; for example, a view of

logs of varying sizes on a conveyor belt outdoors might vary vastly in image quality, brightness, glares, and more depending on the time of day, weather, or season. On the other hand, boards of similar size and shape in a small, well-lit room without windows will look the same no matter the circumstances. Lastly, training a model on the dataset can take a couple of days.

The first of these problems, external labelling, could be solved with a few- or zero-shot model that would require little to no new labelled data before running at a new site. The second problem could be solved with a more robust model which can handle detection even with varying lighting, obstructions, weather, and camera set-up. We combine these solutions to develop a new approach: A single very powerful and robust model to handle startup at a new site with little to no labelled data and which is robust enough to handle both changes in weather and lighting. This would hopefully enable significant speed-ups (see Figures 1.2 and 1.3).

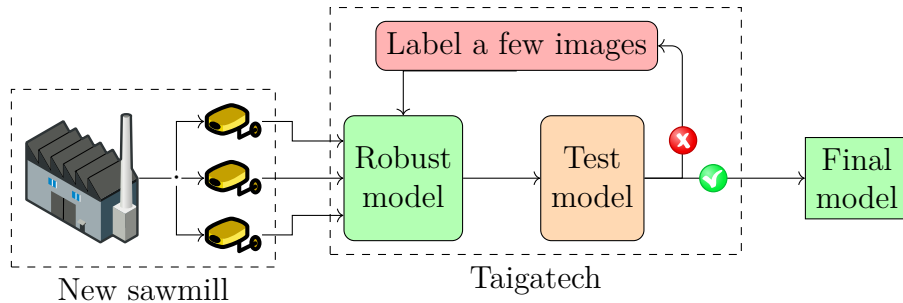


Figure 1.3: Outline of the proposed pipeline for getting up-and-running at a new sawmill. In most cases, the robust model needs little to no adaptation to get up-and-running. When it does, it will need so little labelling that it can be done in-house at Taigatech.

1.2 Scope and Limitations

As the approach is novel, there are many explorable avenues when it comes to areas like measurable metrics - e.g. number of knots on a board - or things which would have been almost impossible with the old model but which the new, more powerful model could facilitate such as depth tracking, tracking the same board over several frames, or ranking how likely a board or log is to end up causing an alarm further down the line. These, however, will not be the focus of the work. Even if some of these features are very appealing to *Taigatech* and might get developed alongside the thesis, they will mostly be left out of this paper.

Another seemingly reasonable approach that could be used is Self Supervised Learning (see Section 2.1.3), but the amount of data required is beyond what is available for this project. We do, however, use a DINO backbone, which is trained using SSL techniques on data from the entire web. In the future, such an approach might be viable to create a Foundational "Taigatech Segment Anyboard Model".

1.3 Our Contributions

We propose ROMAn-DETR, a robust orientation-aware model with anisotropic attention for real-time instance segmentation, to solving the problems introduced in the previous sections. Our contributions to the field and to Taigatech include:

- A more robust model for one of their core products.
- Deformable attention specifically tailored to work on anisotropic objects.
- An iterative orientation predictor for rotated objects.
- Showing why a simple head on a frozen ViT backbone is not a viable approach for this instance segmentation task.

2

Background

We present the background of the problem.

2.1 Related work

This thesis relies heavily on previously done work within machine learning and computer vision. Many books and papers have been written on this topic, e.g. by Goodfellow, Bengio and Courville [7]; as well as by Yildirim and Asgari-Chenaghlu [27]. Our anisotropic deformable attention integrated with RF-DETR is also heavily based on work by Xia et al. in 2022 [26] as well as the paper on deformable DETR by Zhu et al from 2021 [29].

2.1.1 Instance segmentation

Instance segmentation is a technique in computer vision that aims to recognize individual objects on the pixel level. Unlike standard object detection, instance segmentation will give a good understanding of the shape and size of a detected object (see Figure 2.1). In industrial settings this can be useful in areas from manufacturing [13] to quality control [23] and safety [28].

2.1.2 RF-DETR

In 2025, the RoboFlow team developed RF-DETR, an object detector and instance segmentation model with a good trade-off between accuracy and latency, as well as a focus on domain adaptability. This is optimal for Process Surveillance as the three main metrics of interest for choosing a model are frame rate, segmentation, and bounding box accuracy, as well as the ability of the model to adapt to a new site [15]. RF-DETR consists essentially of three pieces: a *DINOv2 backbone* [12], a *projector* built using a C2F block that 'converts' the DINOv2 features from DINOv2 feature space to the domain used by the *DETR decoder*. The decoder has two different heads, one for finding the bounding boxes and classifying the objects, and one for segmenting the objects within each bounding box [11].

2.1.3 Foundational models and SSL

Researchers from Stanford Center for Research on Foundation Models (CRFM) defined, in 2021, that "*A foundational model is any model that is trained on broad data*

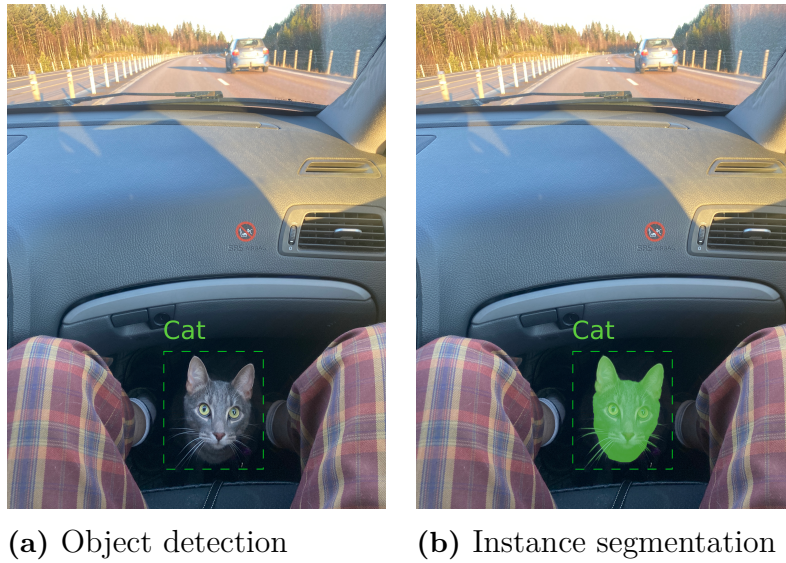


Figure 2.1: The difference between object detection (a) and instance segmentation (b) is that instance segmentation gives a per-pixel classification of the image. This can be easily analysed to determine for example the shape or angle of an object

(generally using self-supervision at scale) that can be adapted (e.g., fine-tuned) to a wide range of downstream tasks" [5]. SSL (self-supervised learning) is a good fit for these models as they require vast amounts of data; and labeling all that data would take an enormous effort. For the purposes of this thesis work, the appeal of these foundational models is their robustness. If the purpose was to classify boards from a camera view with consistent lighting and no disturbances, a foundational model would be unnecessarily complex. However, as the goal is to make a model powerful enough to tackle any board detection task thrown at it, a foundational model is a perfect fit. This adaptability property also makes foundational models such as DINO or BERT well suited as *frozen backbones*.

2.1.4 Frozen backbone

A frozen backbone architecture consists of at least two parts, the backbone and the head. The model's input goes into the backbone which processes it and extracts features from the input. These features are then sent to the head, which can interpret them and produce an output. For example, as DINOv3 is already trained and powerful, it can be utilized to extract features and interesting object from an image, that a machine learning head can then be trained to classify or otherwise modify for some desired effect such as image segmentation or object detection of objects that were not in the training dataset. Together, a custom trained head and a foundational model backbone make a powerful machine learning combo that can tackle many diverse problems.

2.2 Timeline

- * **11th century** Ibn Al-Haytham, the father of modern optics, writes the *Book of Optics*, recognizing a huge gap between the massive amounts of visual information and the well-defined separate objects we see [22].
- * **19th century** With the invention of the steam engine, logs can be cut into boards without relying on water power. This sparks enormous improvements and automation within the lumber industry [8].
- * **1950s** AI researchers create the first neural networks. Notably, neuroscientist Frank Rosenblatt creates the *perceptron*, a neural network with randomly initialized weights which were then updated through something akin to reinforcement learning to learn to recognize eight different letters [8][1].
- * **1960s** Inspired by David Hubel and Torsten Wiesel’s experiments in 1959, the first convolutional neural networks are created for use in computer vision, which would go on to be the dominant technology for computer vision for several decades [25][18].
- * **2017** Vaswani et al. publish what comes to become one of the most important computer science papers of all time in *attention is all you need*, outlining the attention mechanism and laying the groundwork for *transformers* (see Section 3.1.4), used in tools such as GPT [24]. In the same year, Dai et al. released their paper on *Deformable Convolutional Networks*. [2]
- * **2020** Dosovitskiy et al. take the transformer and adapt it for image processing, creating the first Vision Transformer (ViT) [4].
- * **2022** Xia et al. combined the ViT with the ideas from Deformable Convolutional Networks to create the Deformable Vision Transformer [26].
- * **2025** Robinson et al. release RF-DETR, a light-weight specialist detection transformer which is what we will base our deep learning model on [15].
- * **2026** We introduce ROMAn-DETR by extending RF-DETR to anisotropic objects

We will base a lot of our work on RF-DETR and the Deformable Vision Transformer, but adapt it to work less generally and more optimized for anisotropic objects which may be tilted.

3

Theory

In this section, we will cover the basics of Deep Learning (Section 3.1), Computer Vision (Section 3.2), and Deformable Attention (Section 3.3).

3.1 Deep Learning

Computers and humans often have inverse difficulty with tasks where problems that may be easy for a computer, such as rapidly multiplying enormous numbers or counting out exactly ten seconds, are difficult or close to impossible for a human. The inverse is also true, where a problem that is easy for humans, such as telling if someone is being sarcastic or whether an image contains a face, can be enormously hard for a computer. Until recently, these problems have been impossible for computers, because they are not easy to break down into simple and strict rules and algorithms. The solution to this conundrum is Deep Learning, where a system can learn to combine several small, learned transformations in order to solve one of these harder problems. For example, to detect sarcasm, the mapping from

Text $\rightarrow$ Sarcasm?

can be broken down into

Text $\rightarrow$ Context $\rightarrow$ Seriousness (3.1)

Text $\rightarrow$ Key Sentence $\rightarrow$ Semantics (3.2)

which can be used in a mapping from

Seriousness + Semantics $\rightarrow$ Sarcasm?

Here, each ' $\rightarrow$ ' represents a learned mapper. This stacking approach of breaking down a hard task into smaller and easier-to-learn subtasks is the core of deep learning, which will be explored further later in the report. To make computations more effective, each step is usually in the form of a matrix multiplication, and the entire deep learning model is essentially a number of matrix multiplications often both run in parallel¹ as well as sequentially [7].

¹Modern GPUs are built for massive parallelism, utilizing a huge amount of parallel computations. By doing the matrix multiplications in parallel, this can be exploited fully for better performance.

3.1.1 Stochastic Gradient Descent

Most modern deep learning is based on an algorithm called *stochastic gradient descent*, which in turn is based on *gradient descent*. Gradient descent is an algorithm for finding the minimum of a function f by

1. picking a random point x within the domain of f
2. calculating the slope of f at x , $\nabla f(x)$
3. go back to step 2 with a new point $x' = x - \epsilon \nabla f(x)$, where ϵ is the *learning rate*.

When the calculated slope is 0 or sufficiently small, we are in a stationary point, which is hopefully a minimum. The problem with using gradient descent for deep learning is mainly step 2, calculating the slope of f . In deep learning, the function f is actually several different functions

$$f_i \quad i = [1, \dots, m]$$

where each function f_i is based on one entry in the training set T , and where m is the size of the training set. To calculate the slope one thus needs to compute

$$\text{slope} = \frac{1}{m} \sum_{i=1}^m \nabla f_i(x) \tag{3.3}$$

which for large m can be extremely computationally heavy. In stochastic gradient descent we instead take a small subset of the training data $T' \subset T$ with size $m' \ll m$ and thus to calculate the slope we can simply calculate

$$\text{slope}' = \frac{1}{m'} \sum_{i=1}^{m'} \nabla f_i(x)$$

This explanation is mainly based on the explanation in Deep Learning by Ian Goodfellow et.al [7]. We are not done yet, however, as we have still not defined the functions f_i . In deep learning, these are called loss functions and will be explained in the next section.

3.1.2 Loss Function

To improve a deep learning model, there has to be some definition of *improving* and thus there has to be a distinction between a good and a bad prediction. This is where the loss function comes in, if a model is supposed to make a prediction the loss function defines a score for the prediction based on how close or far off from the truth the prediction is. One common loss function is the Mean Squared Error (MSE) which is defined as

$$MSE = \frac{1}{n} \sum_{i=1}^n (x_i - y_i)^2$$

where x_i are the predictions and y_i the observed values, the truth.

3.1.3 Covariate Shift

One of the major problems with such mappings is Covariate Shift, where changes to the distribution of input data may translate to unwanted changes in the output. In the previous example (see Equation 3.2), there is a hypothetical mapping from

Key Sentence $\rightarrow$ Semantics.

If this mapper had been trained on only American English text, it might correctly see that the sentences

The kid was sad when he noticed his bike was stolen

and

The poor thing cried when someone had taken his bike

are semantically similar, but struggle with mapping the sentence

The wee lad was gutted to learn someone had nicked his bicycle

to a similar vector representation [19]. Covariate shift is an example of *domain shift* and is a very active area of research with more than 120 hits on IEEE Xplore in 2025 alone. As deep learning depends on several such mappings cooperating, *internal covariate shift*, where covariate shift happens inside the layers of the model during both training and testing, is one of the main hurdles when trying to create a robust model. Internal covariate shift can not only make the model less reliable but may also slow down training, though there is evidence to dispute this [17].

3.1.4 Transformers

Introduced in the now famous 2017 paper *Attention is all you need*, transformers have revolutionized the world of deep learning. The transformer design depends on attention, a mechanism where the relative importance of each component in a sequence to every other component is decided in a clever way. The sequence can be thought of as, for example, words in a sentence, parts of an image, or snippets of a sound. The transformer begins by embedding each component into some high-dimensional embedding space, before entering the multi-headed attention phase. The attention mechanism can be split into three steps:

1. For each component in the sequence, produce a query vector which is basically a numerical representation of the question

Q : What other parts of this sequence are important to me?, (3.4)

a key vector which is similar but instead represents an answer of

K : These are the types of queries I am relevant to, (3.5)

as well as a value vector

V : This is how your embedding should be updated. (3.6)

They are produced by a query-, key-, and value-matrix, respectively.

2. The dot products between the query and key vectors are calculated and the resulting matrix representing the relevance of each component to every other is softmaxed² and multiplied by the value vector corresponding to the relevant key vector's component.
3. All of the multiplied value vectors for each component are added, producing a total desired adjustment for that component's embedding. After addition, the adjusted embeddings can be run through another attention mechanism to further update the embeddings to represent deeper connections and meanings within the sequence.

Mathematically, attention is defined as

$$\text{Attention}(Q, K, V) = \text{SoftMax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

where SoftMax is just a normalizing function, d_k is the dimension of K , and (Q, K, V) correspond to the Query, Key, and Value, respectively [24]. Most transformers can be categorized into *encoder-only*, *decoder-only*, or *encoder-decoder*. Encoding can be thought of as *perceiving* and decoding as *producing*. For example, a transformer which analyses a piece of text such as BERT [3] is probably encoder-only, whereas generative AIs such as OpenAI's GPT series are decoder-only, and a translator is most likely an encoder-decoder.

The difference is characterized in the attention map, which is essentially what keys get queried. In a decoder-only transformer the attention mechanism only queries the earlier parts of the sequence, unlike in an encoder-only transformer where the entire sequence is queried. If the query 3.4 would be adapted for a decoder-only transformer it might instead say

Q : What parts of this sequence so far are important to me?

As we want to query the entire image and look for boards, we will use DINOv3 as an encoder-only transformer.

3.2 Computer Vision

For humans, visual input is one of the main forms of impression that shapes how we experience and form an inner model of the world. Szeliski writes more about this in his book *Computer Vision: Algorithms and Applications* but we will give a small overview below.

Even small children can understand the relative positional relation of objects, people, and landscape in a photograph, so why is it so hard for computers? One reason that computer vision is hard is that it requires predicting based on insufficient information. For example, when doing depth estimation, there is no depth information embedded in the image. The computer vision system has to somehow make that

²Softmax is a way of adjusting a number of elements with real number values such that they sum to 1.

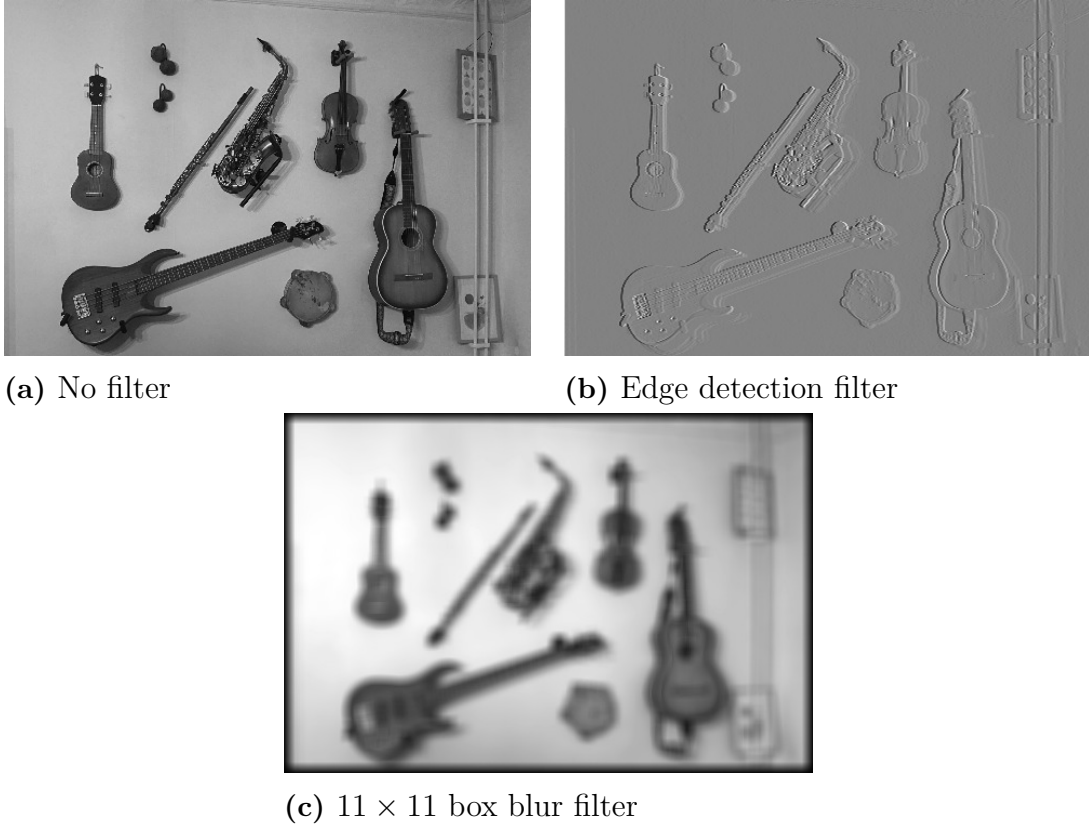


Figure 3.1: Two simple filtered images, one for edge detection and one for a box blur. Their corresponding convolution kernels are

$$(a) \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad (b) \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \quad \text{and (c) } \frac{1}{11^2} \cdot \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \Bigg\} 11 \times 11$$

information up. These systems that create new information from limited knowledge are called models and can take on a multitude of different forms, from physics simulation to probabilistic predictions or machine learning models[21].

There are several types of computer vision models but most important for the use cases in this thesis are CNNs and ViTs, two different classes of machine learning models. Traditionally, CNNs have dominated the field of computer vision, but with the recent success of transformers in natural language processing with tools like OpenAI's GPT models, Google's Gemini, and X's Grok, they (transformers) have been adapted to work with image data [24, 14]. This is what's called a Vision Transformer or ViT for short and is what "AI chatbots" like ChatGPT use for image capabilities [4].

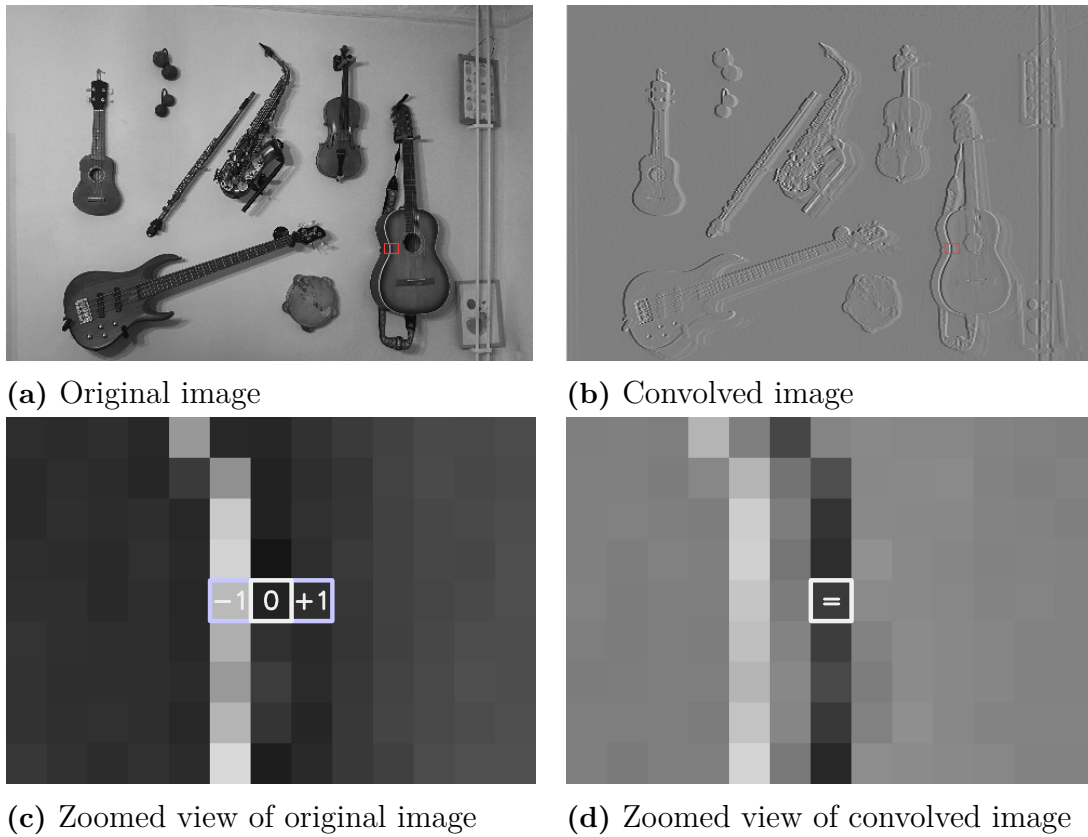


Figure 3.2: A closer look at a convolution with a simple filter $[-1, 0, +1]$. For each pixel, the new value becomes (*value of right pixel* – *value of left pixel*). For example, for the marked pixel in the middle of (c) the picture to the left has value 180 (values lie in the range $[0, 255]$) and the pixel to the right has value 27, thus the corresponding marked pixel in (d) has value $27 - 180 = -153$ (values lie in the range $[-255, 255]$)

3.2.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a machine learning architecture based on doing several *convolutions*. Convolutions are a versatile and pretty intuitive way of extracting features from an image, and have been proven and demonstrated to be able to solve various computer vision problems such as optical character recognition [6][9]. A convolution can be thought of as passing a filter across some sequence and obtaining a resulting sequence. For example, a simple filter such as $[1, 0 - 1]$ can be used to gain information about the texture and outlines of objects (see figure 3.1b) and a simple normalized matrix of ones can be used as a box blur to gain more high-level information about objects and areas in the images such as lightness in an area. Modern CNNs can have several thousands of convolutions and do several gigaFLOPs³ when running on a single image. [10]

³Floating Point OPERations, a measure of the amount of calculations done.



Figure 3.3: When making patches the input image is split into several small and square images with side length determined by the model. For example, DINOv3-Tiny wants squares with side length 14 and DINOv3-Large wants squares with side length 16. In this example, we have an image with size 640×480 and a patch size of 16, giving us $40 \times 30 = 1200$ patches of size 16×16 .

3.2.2 Vision Transformers

The transformer architecture for machine learning has proven its worth in the last couple of years with the surge of large language models. This caused researchers to start implementing the same strategies for creating vision models with great success. The vision transformer can be split into three stages: division into patches, embedding, and a transformer encoder[4]. The division into patches is quite simple (see Figure 3.3) and consists of cropping the input image into several smaller images. For example, for an image with resolution 1080×1920 and a ViT model with patch size 12 we would turn the input image into $[1080/12 = 90] \times [1920/12 = 160] \rightarrow 90 \times 160$ patches of size 12×12 , they are then flattened to a single 14400-long sequence of patches.

The patches are then embedded by first turning each patch into a sequence (Figure 3.4), each of which becomes the row of a matrix, and then multiplying the matrix of patch sequences by an embedding matrix to obtain the final embedded patches. This matrix, along with positional encodings, will act as input to the transformer which will process the patches and output a vector, which can be seen as a point in domain space, for each patch. In a way, the ViT reduces the resolution by a factor of its patch size.

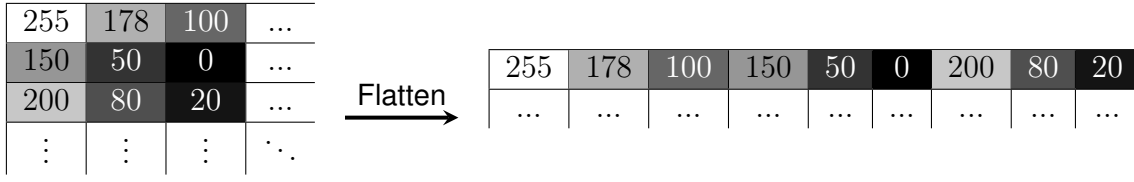


Figure 3.4: The first step of the patch embedding is to flatten the matrix of pixel values in the patch. In this example, a 3×3 matrix of lightness values in the interval $[0, 255]$ is flattened to a 9 long sequence of lightness values in the interval $[0, 255]$. Each patch is "laid below" the previous to form a matrix of dimension $\text{patch size}^2 \times \#\text{patches}$.

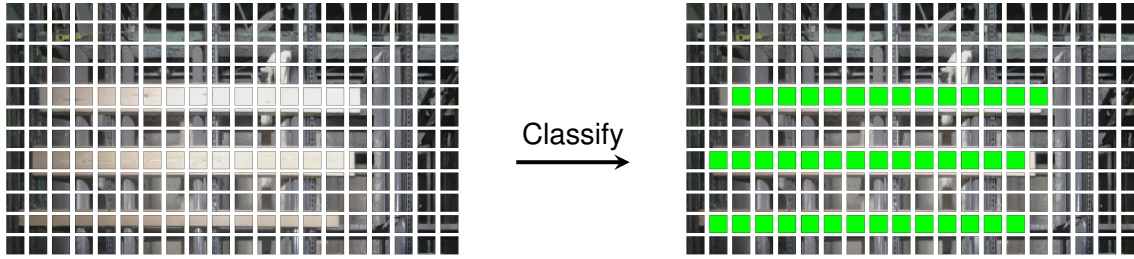


Figure 3.5: To determine what parts of an image contain a board, instead each *patch* can be classified. Here every patch that is more than halfway full of board is counted as board-containing.

3.2.3 Classification and regression

In this project it is necessary to know when part of an image contains a board. When using a vision transformer there are several ways of taking the output of the model and classifying each patch as either "has board" or "has no board" (see Figure 3.5).

The most common alternative to classification is regression, where instead of doing a binary *in/out* decision there is a continuous grading, for example a probability that a patch contains a board.

Another obvious idea is to take some patch which we know⁴ to contain a board and check how similar or dissimilar each other patch is to that one board-containing patch. Two common ways to measure similarity are *Cosine Similarity* and *Euclidean Distance* (See Figure 3.8). The more intuitive one is Euclidean distance; for two vectors

$$X = [x_1, x_2, \dots, x_n], \quad Y = [y_1, y_2, \dots, y_n]$$

it is defined as

$$\text{Euclidean Distance}(X, Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$$

or, simply,

$$\text{Euclidean Distance}(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

⁴The patch is chosen by a human but it could just as well be chosen in some other way

In the two-dimensional case, this is simply the Pythagorean theorem. Cosine similarity, on the other hand, is defined as

$$\frac{X \cdot Y}{\|X\| \|Y\|} = \frac{x_1 y_1 + x_2 y_2 + \cdots + x_n y_n}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \cdot \sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}}$$

which can be written

$$\text{Cosine Similarity}(X, Y) = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \cdot \sqrt{\sum_{i=1}^n y_i^2}}.$$

It is intuitive to see how the Euclidean distance in the feature space would be a good measure of similarity. If two patches are similar and they are in similar contexts, they will have similar patch outputs and thus have a low Euclidean distance between them. The problem with distance is that it does not measure similarity but dissimilarity, the higher the distance, the less similar two points are. This is where cosine similarity comes in; because of the nice property of cosine where two vectors along the same line have a cosine between their angles of 1, while two opposite vectors have a cosine of -1 , cosine similarity is a reasonable and intuitive metric for similarity between two vectors where they can be similar (1), non-similar (0) or anti-similar (-1). [20] This means that for every patch, which can be seen as a point in our feature space, there is a cosine similarity between it and every other patch (see Figure 3.6) ranging between -1 and 1 .

Of course this makes cosine similarity nicer to work with, with values in the range $[-1, 1]$, instead of Euclidean distance's range of $[0, \infty)$. Another notable difference between cosine similarity and Euclidean distance is that if a point is along the same direction but twice as far from the origin, the cosine similarity will still be 1 whereas the Euclidean distance will of course be more than 0.

3.3 Deformable Attention

Here we will go over the difference between deformable attention and the regular attention used in Vision Transformers (ViTs). A good understanding of attention and ViTs is needed to fully grasp this section. We recommend Section 3.1.4 for a brief overview of transformers and attention and the YouTube channel 3Blue1Brown's video series on neural networks and transformers [16] for a deeper but still easily digestible dive. When an ordinary Vision Transformer "looks at" an image, it queries each patch against all other patches in the image. Deformable attention implements positional queries with positional offsets creating several different sample points for each positional query, enabling the model to learn to be better at recognizing objects of certain shapes while also improving performance. In the seminal paper on deformable DETR by Zhu [29], deformable attention is defined as

$$\text{DeformAttn}(\mathbf{z}_q, \mathbf{p}_q, \mathbf{x}) = \sum_{m=1}^M \mathbf{W}_m \left[\sum_{k=1}^K A_{mqk} \cdot \mathbf{W}'_m \mathbf{x}(\mathbf{p}_q + \Delta \mathbf{p}_{mqk}) \right] \quad (3.7)$$

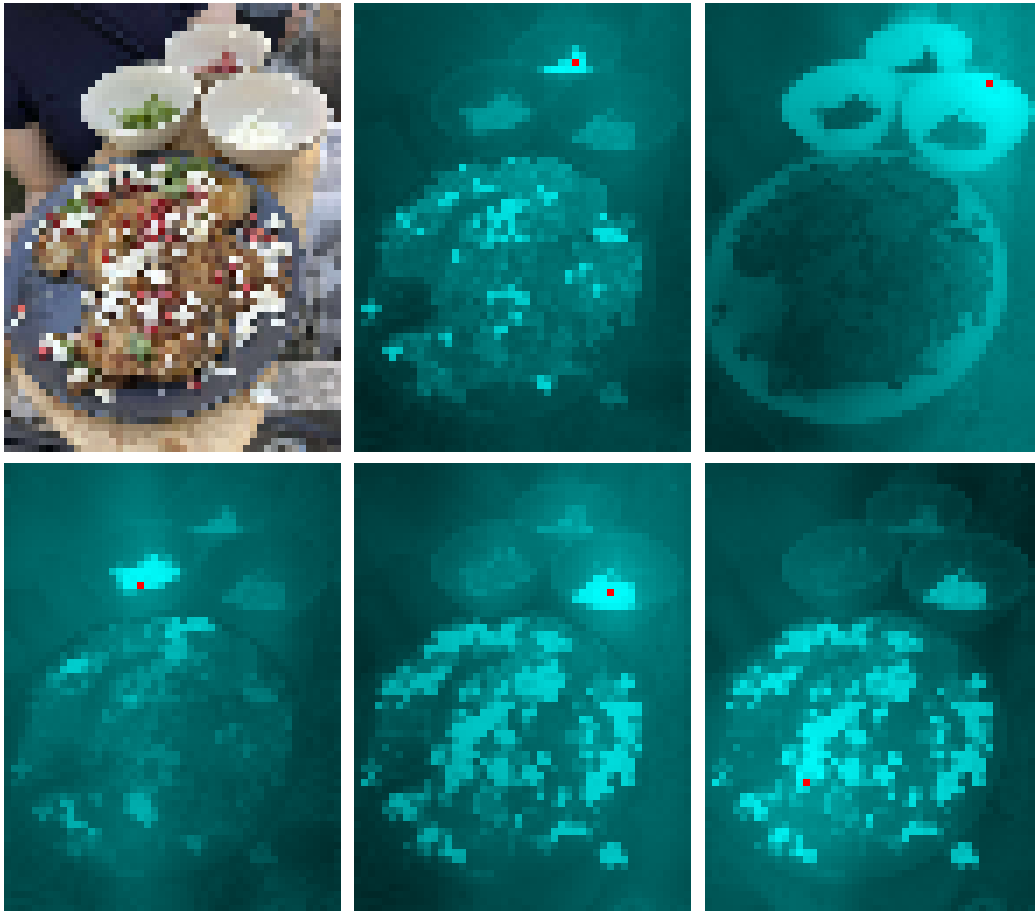
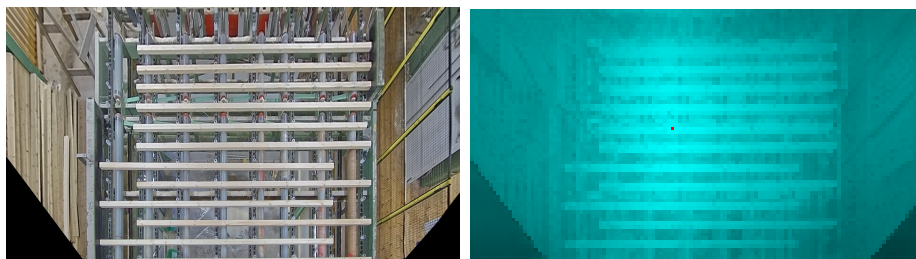


Figure 3.6: The cosine similarity of the features of each patch to the features of a given patch. The patch each other patch is compared to is marked in red, a lighter color is a higher similarity



(a) boards at a sawmill

(b) Cosine similarity

Figure 3.7: Cosine similarity between a user selected board patch and the other patches, a lighter patch means a higher similarity

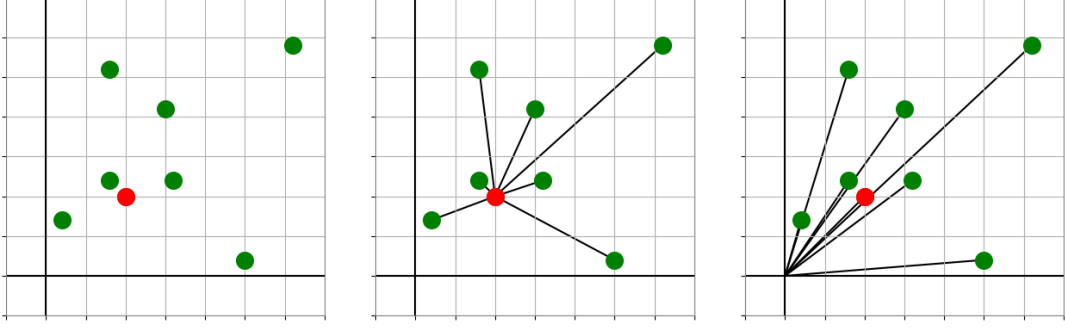


Figure 3.8: Which green dot is most similar to the red one? It depends on how you define "similar"! If you take it to mean "being close by (Euclidean) distance" you get one answer; but if you take it to mean "in the same direction" you get a very different answer.

for the aggregate of all attention heads; in the same paper they describe the usual multi-headed attention in transformers as

$$\text{MultiHeadAttn}(\mathbf{z}_q, \mathbf{x}) = \sum_{m=1}^M \mathbf{W}_m \left[\sum_{k \in \Omega} A_{mqk} \cdot \mathbf{W}'_m \mathbf{x}_k \right] \quad (3.8)$$

For a single attention head the formulae are then

$$\text{DeformAttnSingle}(\mathbf{z}_q, \mathbf{p}_q, \mathbf{x}) = \sum_{k=1}^K A_{qk} \cdot \mathbf{W}' \mathbf{x}(\mathbf{p}_q + \Delta \mathbf{p}_{qk}) \quad (3.9)$$

$$\text{SingleHeadAttn}(\mathbf{z}_q, \mathbf{x}) = \sum_{k \in \Omega} A_{qk} \cdot \mathbf{W}' \mathbf{x}_k \quad (3.10)$$

where A_{qk} and $\mathbf{W}'$ are weights and projection matrices, respectively, K is the number of samples for each positional query ($K \ll \#\text{patches in image}$). Here the differences become clear, deformable attention takes in a positional query and obtains $\Delta \mathbf{p}_{qk}$ via linear projection over $\mathbf{z}_q$. It then takes a A_{qk} -weighted sum of the $\mathbf{W}'$ -projections of the K samples positioned with offset $\Delta \mathbf{p}_{qk}$ in relation to $\mathbf{p}_q$. On the other hand, ordinary attention takes a A_{qk} -weighted sum of the $\mathbf{W}'$ -projections of all patches in the set Ω , which in a ViT is the number of patches.

Ordinary queries thus look at the entire input, which in our case could be confusing due to the abundance of similar boards in a single image, and by training the sampling offsets to follow the expected shape of a board we can reduce this confusion and increase the accuracy of the predicted board shape. In many cases, the deformations made by the sampling offsets can be quite complicated as in the paper by Xia et al. where they consider deformable attention for animals and people [26]. Theoretically, the board case should be a lot simpler as they are all more or less the same shape, only differing in the angle.

4

Methodology

The aim is to reduce the time between accessing a new dataset and being able to confidently run the model on unseen parts of the new dataset. As the point is to simulate getting access to a new camera, it would not be fair to use all of the data in the new dataset for training immediately; instead, either zero-shot learning, few-shot learning on the first couple of images in the dataset¹, or waiting for some time after gaining access to the dataset before training a model on it will be necessary.

In the zero-shot learning case, the assumption is that the model is good enough after training on all of the old datasets to generalize to the new dataset. This could be due to the new dataset being similar to some older datasets. For example, if the model can label both the images in the dataset with images like 4.1a and the dataset with images like 4.1b, it *should* be able to label the images in the dataset with 4.1c, assuming all the images in dataset c are similar to the images in dataset a and b. This depends on the model not over-fitting to the training data as well as learning the general look and shape of the boards instead of something intangible or site-specific.

4.1 Minimal Head approach

Here we explore the approaches using of a frozen backbone with a minimal head.

4.1.1 Frozen backbone

A model using this architecture is divided into (at least) two parts:

¹We will assume the new datasets are chronological snapshots of the camera feed



(a) Some boards, not very ordered



(b) Many well ordered boards



(c) Many boards, not very ordered.

Figure 4.1: In theory, if the model perfectly labels both (a) and (b), it should also be able to label (c).

- A backbone which can neither be trained nor modified, it is simply run on the inputs, and its outputs are the inputs to the *head*. For example, this could be DINOv3 which takes a video frame and splits it into a grid of 16x16 pixel *patches*. These patches are arrays of length 1024.
- A head is a (typically smaller) model which takes the backbone’s outputs as inputs and computes the final output. For example this could be a semantic segmenter which takes DINOv3 feature vectors as inputs and outputs a probability of each *patch* containing board.

4.1.2 DINOv3 backbone with minimal head

Several different approaches with a DINOv3 backbone and a very simple head will be tested, and although there are big technical differences between the techniques, some core ideas and problems remain the same:

- + A very small head would take very little time to run and would thus allow for the use of a bigger and better DINOv3 model.
- + A small head is easier to understand and implement and the same idea could then be tested and evaluated on several different backbones to find a good fit.
- These small heads cannot use all the information in the DINOv3 features effectively and might thus lose out on important information.
- Similar to the previous point, the DINO patching severely reduces the resolution of the image which might cause problems, for example: when two boards are lying close to each other they might get mistaken for a single broad board.

The results of some of these are described in the **Results** chapter. Even though the goal was instance segmentation, most of the testing with a minimal head was regarding semantic segmentation, simply because it is easier to set up and test while also giving a good idea of the capability of the architecture to differentiate object from background. If any head is found which is capable of classifying the DINOv3 patches accurately, it will be used to develop an instance segmentation system.

We hope to use the head to either detect a single board so that it can be run iteratively on the same output, finding each board one by one, or to find all boards and then using some known method for splitting the board mask into several smaller masks of a single board each.

4.1.2.1 Similarity

The similarity between a given patch and a patch definitely containing a board should give a pretty good indication of whether the given patch contains a board. Therefore, we will try to compare every patch to some patch which we know to contain a board. For testing this we came up with two main approaches, *board insertion* and *manual selection*, where manual selection is mainly seen as a placeholder for an eventual future plank patch detection system.

We will compare the patches using cosine similarity as well as euclidean similarity defined as

$$\text{EuclideanSimilarity}(p_1, p_2) = 1 - \frac{\text{EuclideanDistance}(p_1, p_2)}{12}.$$

We get the constant 12 by taking the maximum distance between a patch and all others a few times and checking what the largest distance occurring between patches is.

4.1.2.2 Logistic Regression

Another idea was to train a simple logistic regression classifier to essentially split the feature space into a "board region" and a "non-board region", which could then be analysed to get the individual boards.

4.2 Anisotropic Deformable Attention with Propagated Angle Estimation

Since we know the general shape of the objects, in this case long rectangles, we can produce good deformed attention clouds given only the orientation of each board. In order to achieve good deformable attention, conditioned on the orientation of each object, a number of techniques are combined. This will enable cheaper and more accurate predictions because there is less confusion between different objects that are close but not aligned, as well as fewer sample points outside the object. We also update the standard object detector to use a bounding box fitted to the mask at evaluation instead of using the learnt bounding box.

4.2.1 Orientation Vector Notation

We will use 4D reference points

$$\mathbf{b} = (c_x, c_y, w, h)$$

representing axis-aligned bounding boxes with the centerpoints (c_x, c_y) with width w and height h . We denote normalisation to unit length by

$$N(\mathbf{z}) = \frac{\mathbf{z}}{\|\mathbf{z}\|}$$

The objects have a tilt rather than a direction; in other words, they are the same if you flip them along their length, so

$$\theta \sim \theta + \pi. \quad (4.1)$$

For this reason we instead represent this orientation with a double-angle parametrisation

$$\hat{\mathbf{u}} = N(\tilde{\mathbf{u}}) = \begin{bmatrix} \cos 2\theta \\ \sin 2\theta \end{bmatrix}. \quad (4.2)$$

Equivalence is built-in since

$$\begin{bmatrix} \cos 2(\theta + \pi) \\ \sin 2(\theta + \pi) \end{bmatrix} = \begin{bmatrix} \cos 2\theta \\ \sin 2\theta \end{bmatrix} = \hat{\mathbf{u}}. \quad (4.3)$$

We can extract the magnitudes of $\cos \theta$ and $\sin \theta$ using:

$$|\cos \theta| = \sqrt{\frac{1 + \cos 2\theta}{2}}, \quad (4.4)$$

$$|\sin \theta| = \sqrt{\frac{1 - \cos 2\theta}{2}}. \quad (4.5)$$

When a signed rotation matrix is required we recover the angle via half-angle atan2 identity

$$\theta = \frac{1}{2} \operatorname{atan2}(\sin 2\theta, \cos 2\theta), \quad (4.6)$$

This is stable for any unit-length $\hat{\mathbf{u}}$, and we denote its corresponding rotation matrix as

$$R(\hat{\mathbf{u}}) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad (4.7)$$

where θ is the half-angled recovered from Equation 4.6.

4.2.2 Encoder-Initialised Orientation

In two-stage mode, RF-DETR uses the encoder to generate a set of proposal queries where the top scoring proposals are selected as queries q . We use the corresponding encoder features e_q for each q to generate an initial proposed angle for the object using a linear head

$$\tilde{\mathbf{u}}_q^{enc} = \varphi_{enc}(\mathbf{e}_q), \quad \varphi_{enc} : \mathbb{R}^d \rightarrow \mathbb{R}^2. \quad (4.8)$$

The two-dimensional angle is then normalised to a unit orientation vector.

$$\hat{\mathbf{u}}_q^{(0)} = N(\tilde{\mathbf{u}}_q^{enc}). \quad (4.9)$$

Before training is started, the weights and biases for φ_{enc} are initialised as

$$W_{enc} = \mathbf{0}_{2 \times d}, \quad \mathbf{b}_{enc} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \quad (4.10)$$

This gives every query an initial orientation to begin training with of:

$$\tilde{\mathbf{u}}_q^{enc} = \varphi_{enc}(\mathbf{e}_q) = W_{enc}\mathbf{e}_q + \mathbf{b}_{enc} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad (4.11)$$

which is already normalized, so we get

$$\hat{\mathbf{u}}_q^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (4.12)$$

4.2.3 Propagated Residual Angle Refinement

In each attention layer the angle for the next layer is predicted based on a linear head φ_Δ . We call this iterative residual angle refinement. At layer ℓ the decoder has the current orientation $\hat{\mathbf{u}}_q^{(\ell)}$ as well as, after cross attention and FFN, a feature vector

$$\mathbf{h}_q^{(\ell)} \in \mathbb{R}^d. \quad (4.13)$$

From $\mathbf{h}_q^{(\ell)}$ a linear head predicts a correcting vector

$$\tilde{\mathbf{r}}_q^{(\ell)} = \varphi_\Delta^{(\ell)}(\mathbf{h}_q^{(\ell)}), \quad \varphi_\Delta^{(\ell)} : \mathbb{R}^d \rightarrow \mathbb{R}^2. \quad (4.14)$$

The residual is added to the identity-rotation vector

$$\mathbf{r}_q^{(\ell)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \tilde{\mathbf{r}}_q^{(\ell)}. \quad (4.15)$$

Before training is started, we initialise the weights and biases such as

$$W_\Delta^{(\ell)} = \mathbf{0}_{2 \times d}, \quad \mathbf{b}_\Delta^{(\ell)} = \mathbf{0}_2, \quad (4.16)$$

so that the residual vector before normalisation becomes the identity-rotation vector

$$\mathbf{r}_q^{(\ell)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \mathbf{0}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad (4.17)$$

which is already normalised, so $N(\mathbf{r}_q^{(\ell)}) = (1, 0)^\top$. The rotation matrix then corresponds to a zero-angle rotation, the identity matrix

$$R(N(\mathbf{r}_q^{(\ell)})) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{\text{init}}. \quad (4.18)$$

Every layer then propagates the encoder’s angle estimate unchanged and we then get the predicted orientation for layer $\ell + 1$ from

$$\hat{\mathbf{u}}_q^{(\ell+1)} = R(N(\mathbf{r}_q^{(\ell)})) (\hat{\mathbf{u}}_q^{(\ell)})_{\text{sg}}. \quad (4.19)$$

Here, $(\cdot)_{\text{sg}}$ stands for stop-gradient, meaning that the gradient for the Δ -head does not propagate through the decoder layers during training, but instead each layer supervises its own learning.

4.2.4 Canonical-Frame Sampling

We use canonical frame sampling, that is, sampling from a per-object rotated bounding box using the per-query object’s axis and cross-axis as a reference frame for the sample points. The canonical coordinates of the query are then (α, β) where α is along the predicted oriented spine and β is along the orthogonal cross-axis.

As we only have knowledge of the axis-aligned bounding box and the angle of the object, we need to extract its scale in the canonical frame. As we use the anisotropic priors for this work, the scale factor s_α can be well approximated using

$$s_\alpha = |\cos \theta_q| w_q + |\sin \theta_q| h_q \quad (4.20)$$

only overshooting the scaled length by $w \ll l$. Determining s_β is a less well-defined problem as $\theta \rightarrow \frac{\pi}{4}$ since we then would not be able to extract any knowledge of its width and only having to rely on priors. To solve this problem we try to find the cross section of the object using

$$c_{\text{proj}} = \left| |\cos \theta_q| h_q - |\sin \theta_q| w_q \right| = c \cdot |\cos 2\theta_q|. \quad (4.21)$$

This works well for axis-alignment but quickly becomes a problem for diagonal cases where the cross section would be as large as the length of the object, which is a worst case scenario with our priors.

To tackle this we start by implementing an information weight from the bounding-box which is used as an uncertainty factor of s_β

$$\sigma = |\cos 2\theta_q| \quad (4.22)$$

is 1 at alignment and 0 at diagonal predictions and gates with

$$s_\beta = c_{\text{proj}} + (1 - \sigma) \rho_\beta(\mathbf{q}) s_\alpha. \quad (4.23)$$

To then preserve full information from $\mathbf{b}$ at axis-alignment and allow for flexibility across queries as $\sigma \rightarrow 0$, we also implement the per-query learnable anisotropy ratio ρ_β which acts as a learned value using every query's embedding

$$\rho_\beta(\mathbf{q}) = \text{sigmoid}(\mathbf{w}_\rho^\top \mathbf{q} + b_\rho). \quad (4.24)$$

This allows for the model to learn the anisotropic priors of the low-information predictions itself instead of having to manually tune it after the dataset. The scaled local offsets in canonical coordinates are then

$$\alpha_{q,m,k,p} = \frac{\alpha_{q,m,k,p}^{\text{raw}}}{P} \cdot \frac{s_\alpha}{2}, \quad \beta_{q,m,k,p} = \frac{\beta_{q,m,k,p}^{\text{raw}}}{P} \cdot \frac{s_\beta}{2}. \quad (4.25)$$

The rotation back to image-space from the canonical frame is then performed by the rotation matrix $R(\theta_q)$

$$\begin{pmatrix} \Delta x_{q,m,k,p} \\ \Delta y_{q,m,k,p} \end{pmatrix} = R(\theta_q) \begin{pmatrix} \alpha_{q,m,k,p} \\ \beta_{q,m,k,p} \end{pmatrix}. \quad (4.26)$$

Finally, the sampling for the reference points is given by

$$\mathbf{p}_{q,m,k,p} = (x_q, y_q) + (\Delta x_{q,m,k,p}, \Delta y_{q,m,k,p}) \quad (4.27)$$

where P is the number of sampling points per head and $(\Delta \mathbf{x}_{q,m,k,p}, \Delta \mathbf{y}_{q,m,k,p})$ are vectors of coordinate offsets that define the distribution.

4.2.5 Orientation Supervision

When training the orientation predictor on an image, there is a need for both obtaining a ground truth orientation and determining how well the model predicts orientation. In the following three subsections we will go over how to obtain a ground truth orientation from a ground truth mask, how to make sure the ground truth orientation is good enough, and define a metric for measuring loss.

4.2.5.1 Ground-Truth orientation using PCA

We obtain the ground truth orientation for use during training from the ground truth masks using PCA. For each provided ground-truth instance p with binary mask $M_p \in \{0, 1\}^{H \times W}$, let $\mathbf{X}$ be the set of foreground pixels. The mean position for instance p is then

$$\bar{\mathbf{x}} = \frac{1}{N_p} \sum_{i=1}^{N_p} \mathbf{x}_i \quad (4.28)$$

where N_p is the number of foreground pixels and $\mathbf{x}_i$ is an element in $\mathbf{X}$. The 2×2 sample covariance matrix is then

$$\Sigma_p = \frac{1}{N_p - 1} \sum_{i=1}^{N_p} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix}, \quad (4.29)$$

Instead of performing eigendecomposition of Σ_p we use the double-angle representation with the closed-form expression for the principal axis of a symmetric 2×2 matrix

$$\hat{\mathbf{u}}_p^{\text{gt}} = \frac{1}{D} \begin{bmatrix} \sigma_{xx} - \sigma_{yy} \\ 2\sigma_{xy} \end{bmatrix} = \begin{bmatrix} \cos 2\theta_p \\ \sin 2\theta_p \end{bmatrix} \quad (4.30)$$

with normalisation $D = \sqrt{(\sigma_{xx} - \sigma_{yy})^2 + (2\sigma_{xy})^2}$. This is equivalent to taking the dominant eigenvector $\mathbf{u}_p = (\cos \theta_p, \sin \theta_p)^\top$ of Σ_p , and representing its orientation as $(\cos 2\theta_p, \sin 2\theta_p)^\top$, but avoids explicit eigendecomposition and removes the eigenvector sign ambiguity.

4.2.5.2 Ground-Truth Validity Filtering

The reliability of the PCA-derived orientation depends on the elongation of the foreground in the mask. For near-isotropic objects the principal eigenvector is poorly defined and is left out as to not affect the angle loss. We define the *anisotropy ratio* for an object p with eigenvalues λ_1 and λ_2 as

$$\rho_p = \frac{\lambda_1}{\lambda_2 + \epsilon}, \quad \epsilon = 10^{-6} \quad (4.31)$$

Where L_p and B_p are the length and breadth of the object. For an approximately rectangular² mask the eigenvalues are proportional to the squared physical dimen-

²Without any holes or pieces missing.

sions and so

$$\sqrt{\rho_p} \approx \frac{L_p}{B_p}. \quad (4.32)$$

Each instance is assigned a validity flag to indicate whether it is sufficiently reliable for supervision

$$v_p = \begin{cases} 0 & \text{if } \rho_p \leq \tau_\rho \\ 1 & \text{if } \rho_p > \tau_\rho \end{cases} \quad (4.33)$$

A $\tau_\rho = 9$ was chosen, which corresponds to a target aspect ratio of $\sqrt{9} = 3$. Instances less elongated than this are ignored during training of the angle predictor.

4.2.5.3 Geodesic L1 Loss

During training we need a metric to determine how well the predicted orientation matches the ground-truth axis derived from the mask using Equation 4.30. We use the geodesic L1 loss, the shortest-path arc length on the unit circle between the predicted and ground-truth double-angle vectors. This was chosen for two reasons.

(i) It has a constant gradient magnitude with respect to the axis error, providing supervision even when predicted angles are close to the ground-truth.

(ii) It is robust to angular outliers in the image. Since the dataset is skewed to have many objects in the same axis, the L1 loss provides equal gradient support to all instances which handles rare off-axis cases instead of learning the major distribution of angles in the image.

Both the ground truth and prediction are represented in double-angle form as

$$\hat{\mathbf{u}}_q^{(L-1)} = \begin{bmatrix} \cos 2\hat{\theta}_q \\ \sin 2\hat{\theta}_q \end{bmatrix}, \quad \hat{\mathbf{u}}_p^{\text{gt}} = \begin{bmatrix} \cos 2\theta_p \\ \sin 2\theta_p \end{bmatrix}. \quad (4.34)$$

For each matched pair (q, p) , we compute the dot product

$$d_{q,p} = \hat{\mathbf{u}}_q^{(L-1)} \cdot \hat{\mathbf{u}}_p^{\text{gt}} = \cos(2\theta_{\text{err}}) \quad (4.35)$$

and the scalar cross product

$$c_{q,p} = \hat{u}_{q,0}^{(L-1)} \hat{u}_{p,1}^{\text{gt}} - \hat{u}_{q,1}^{(L-1)} \hat{u}_{p,0}^{\text{gt}} = -\sin(2\theta_{\text{err}}), \quad (4.36)$$

where $\theta_{\text{err}} = \hat{\theta}_q - \theta_p$ is the underlying axis error. Since both vectors lie on the unit circle, $d_{q,p}^2 + c_{q,p}^2 = 1$.

The unsigned geodesic angular error is then

$$\ell_{q,p} = \frac{1}{2} \text{atan2}(|c_{q,p}|, d_{q,p}) = |\theta_{\text{err}}|. \quad (4.37)$$

Given the set of all matching pairs

$$\mathcal{V} = \{(q, p) : q \text{ matches } p \text{ and } v_p = 1\}, \quad (4.38)$$

the final angle loss for the image is

$$\mathcal{L}_{\text{angle}} = \frac{1}{|\mathcal{V}|} \sum_{(q,p) \in \mathcal{V}} \ell_{q,p}. \quad (4.39)$$

Since $\ell_{q,p} = |\theta_{\text{err}}|$ is in radians, the loss can naturally be interpreted as the mean per-query axis error.

5

Results

Here, we will go through the results from the frozen backbone and minimal head and see that they are, spoiler alert, not satisfactory. We will then move on to going over the more advanced method which also garnered better results.

5.1 DINOv3 backbone with minimal head

For the frozen backbone with minimal head approach, no satisfactory method was found. Therefore, the results presented are not for an instance segmenter, but for a prerequisite, accurate plank detection.

5.1.1 Similarity

The results of the similarity heads are presented below.

5.1.1.1 Inserted Board

Some images that we consider representative of the performance of this method can be seen in Figure 5.1.

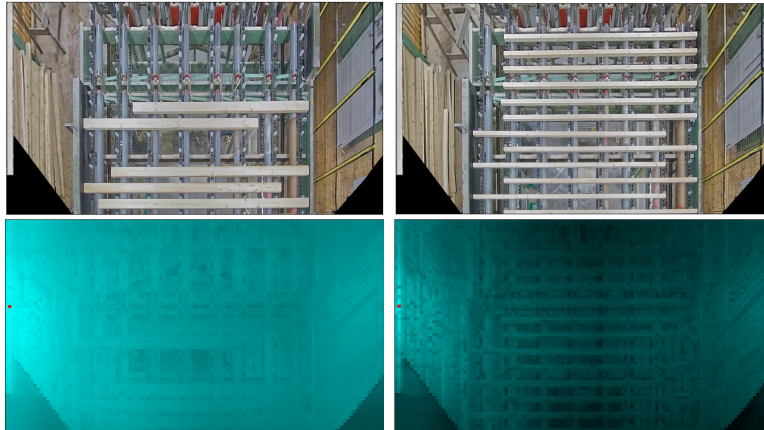


Figure 5.1: Visualizations of inserting a plank and using cosine and Euclidean similarity between patches, a brighter pixel corresponds to a higher similarity.

5.1.1.2 Manually Selected Board

Some results that we consider representative of the performance of this method can be seen in Figure 5.2.

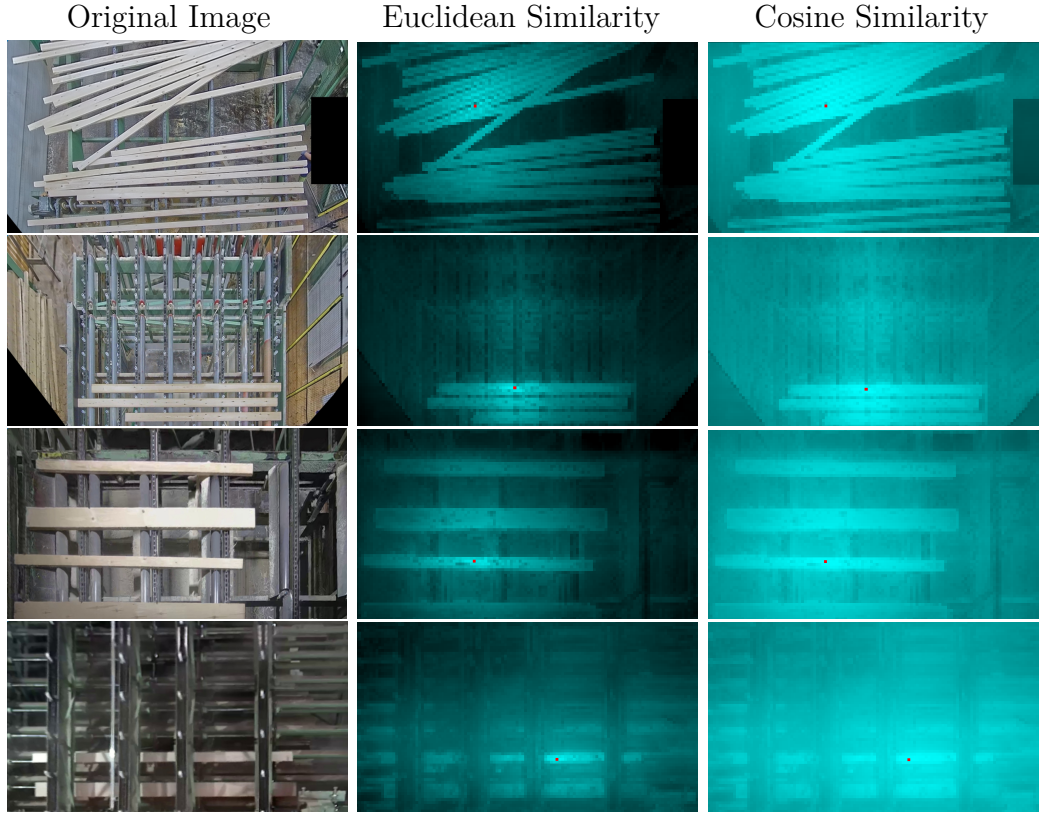


Figure 5.2: Visualizations of manually selecting a plank and using cosine and euclidean similarity between it and all other patches, the selected patch is marked in red and a brighter pixel corresponds to a higher similarity.

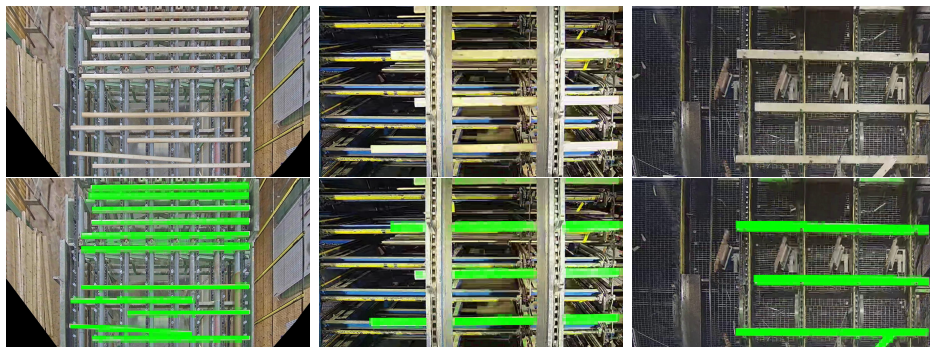


Figure 5.3: Logistic regression head trained to split the input image into *board* vs *not board*. A higher opacity green corresponds to a higher confidence in the patch being *board*.

5.1.2 Logistic regression

Some images which we deem representative of the performance of a logistic regression head can be seen in Figure 5.3.

5.2 ROMAn-DETR Performance

In this section, we will present how well the model performed for the segmentation and bounding box tasks. The results come from the hold-out test set which was left unseen until evaluation and consists of unseen images from the same distribution as the train and validation sets. As the data is horizontal-biased we also evaluate over multiple splits in which each image is rotated by $\pm\theta$ degrees.

5.2.1 Segmentation Model

For the main segmentation task we see an improvement across all metrics for our model.

Table 5.1: Segmentation AP improvement of ROMAn-DETR nano over the baseline RF-DETR nano.

Split	Baseline AP	Ours AP	Δ (pp)	Δ (rel.)
In-distribution	0.7315	0.7512	+1.97	+2.7%
$ \theta = 22.5^\circ$	0.5010	0.5525	+5.14	+10.3%
$ \theta = 45.0^\circ$	0.3747	0.3967	+2.20	+5.9%
$ \theta = 67.5^\circ$	0.5331	0.5481	+1.50	+2.8%
$ \theta = 90.0^\circ$	0.7425	0.7508	+0.83	+1.1%
Mean rotated	0.5378	0.5620	+2.42	+4.5%

Our model outperforms the baseline RF-DETR model on every distribution, boasting a $+1.97$ in mAP improvement in-distribution, and an increase of as high as 10.3% in out-of-distribution mAP when $\theta = 22.5^\circ$.

The trend follows here, our model outperforms the baseline in every single COCO metric across all distributions except one metric at one rotation. In particular following Table 5.1 at $\theta = 22.5^\circ$ we see an improvement in mAP_{75} from .5350 to .6238 yielding an accuracy increase of 17% in that OOD split.

From Figure 5.4 it is clear that our model learns faster and achieves higher validation segmentation scores. For mAP_{50} the score peaks after around 100 epochs and then holds a steady state. Compare this to the baseline model which has a much flatter curve and does not converge until around epoch 300, so 200 epochs slower than ours combined with a lower mAP.

Table 5.2: Full COCO segmentation metrics on the in-distribution test set and across the OOD rotation sweep. AP, AP₅₀, and AP₇₅ are the mean average precision at IoU thresholds [0.5, 0.95], 0.50, and 0.75 respectively; AR₁₀₀ is the average recall over the top-100 detections.

Split	Model	AP	AP ₅₀	AP ₇₅	AR ₁₀₀
In-distribution	Baseline	0.7315	0.9540	0.8802	0.7841
	Ours	0.7512	0.9603	0.9005	0.7986
$ \theta = 22.5^\circ$	Baseline	0.5010	0.8228	0.5350	0.5708
	Ours	0.5525	0.8634	0.6238	0.6182
$ \theta = 45.0^\circ$	Baseline	0.3747	0.6815	0.3758	0.4583
	Ours	0.3967	0.7182	0.3893	0.4742
$ \theta = 67.5^\circ$	Baseline	0.5331	0.8511	0.6101	0.6011
	Ours	0.5481	0.8711	0.6418	0.6025
$ \theta = 90.0^\circ$	Baseline	0.7425	0.9521	0.8938	0.7965
	Ours	0.7508	0.9548	0.8955	0.8011

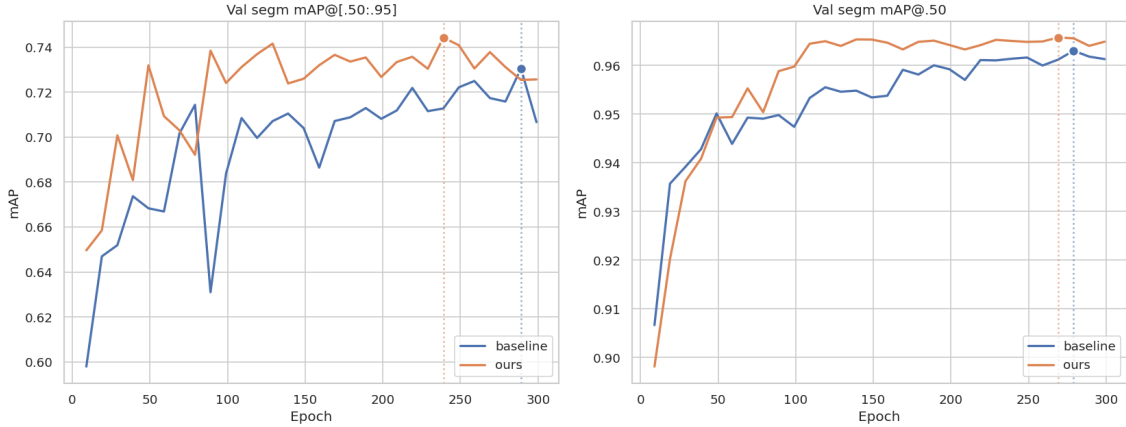


Figure 5.4: Comparison of mAP and mAP_{50} validation during training.

5.2.2 Detection Model

When evaluating the bounding boxes we must take into account that the rotated dataset is not as good of a metric for model comparison as it is for the segmentation. As the dataset contains almost exclusively long, thin objects, achieving a high mAP is much easier on the rotated objects as the boxes will almost exclusively be filled with background and thus giving a much lower IoU. For this reason we mainly focus on the axis-aligned cases where a bounding box must be able to fit to the object much more tightly for high accuracy. For fair comparison in latency during inference, we use our ROMAn-DETR nano with $3.4ms$ latency, against the larger RF-DETR small object detector with $3.5ms$ latency. Since our model has no heavy

added computations during inference, all model latencies are based on the numbers from the RF-DETR paper [15].

Table 5.3: Detection AP improvement of the ROMAn-DETR nano over the baseline RF-DETR small object detector.

Split	Baseline AP	Ours AP	Δ (pp)	Δ (rel.)
In-distribution	0.6213	0.7734	+15.21	+24.5%
$ \theta = 22.5^\circ$	0.7734	0.7900	+1.65	+2.1%
$ \theta = 45.0^\circ$	0.7286	0.7435	+1.49	+2.0%
$ \theta = 67.5^\circ$	0.7818	0.7931	+1.14	+1.5%
$ \theta = 90.0^\circ$	0.6185	0.7898	+17.13	+27.7%
Mean rotated	0.7256	0.7791	+5.35	+7.4%

The results for the detection model in Table 5.3 follow our previous analysis of the axis-alignment compared to rotated data. On the rotated data our model slightly outperforms the baseline with a range of [1.14, 1.65] pp mAP increase. The main point of interest here though is for the axis-aligned in-distribution and 90° rotation. We see that our mask-based model vastly outperforms the larger detection model with an mAP increase of 24.7 and 27.7 for in-domain and 90° rotation respectively.

Table 5.4: Full COCO detection metrics for both models.

Split	Model	AP	AP ₅₀	AP ₇₅	AR ₁₀₀
In-distribution	Baseline	0.6213	0.9283	0.7118	0.6989
	Ours	0.7734	0.9527	0.8989	0.8436
$ \theta = 22.5^\circ$	Baseline	0.7734	0.9103	0.8523	0.8219
	Ours	0.7900	0.9177	0.8539	0.8442
$ \theta = 45.0^\circ$	Baseline	0.7286	0.8838	0.8132	0.7914
	Ours	0.7435	0.8941	0.8176	0.8048
$ \theta = 67.5^\circ$	Baseline	0.7818	0.9220	0.8652	0.8250
	Ours	0.7931	0.9168	0.8537	0.8390
$ \theta = 90.0^\circ$	Baseline	0.6185	0.9399	0.6923	0.6948
	Ours	0.7898	0.9557	0.9016	0.8562

Looking at the entire COCO evaluation we see similar results to the semantic one. Other than two metrics in $\theta = 67.5^\circ$, our model again outperforms the larger baseline model everywhere. Not only does the quality of the box improve, but we see an equally large increase in recall in AR_{100} suggesting that the usage of masks can

improve the robustness of a model at a smaller cost than lightweight detection models.

5.2.3 Validation

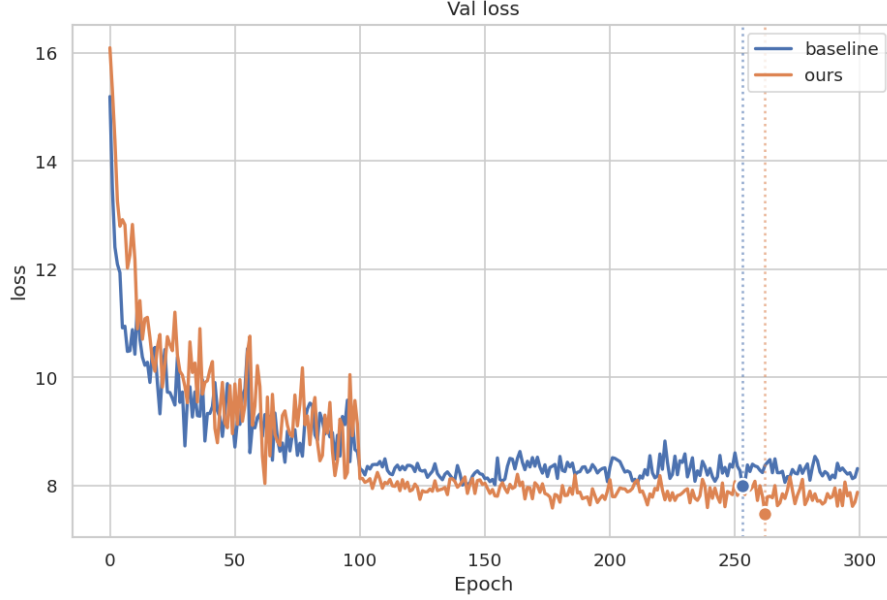


Figure 5.5: Comparison of validation loss during training.

Here we see that our model achieves a lower validation score as well, despite having an additional angle loss. This is aligned with what we have seen in the validation metrics where ours beats the baseline in all but the bbox mAP.

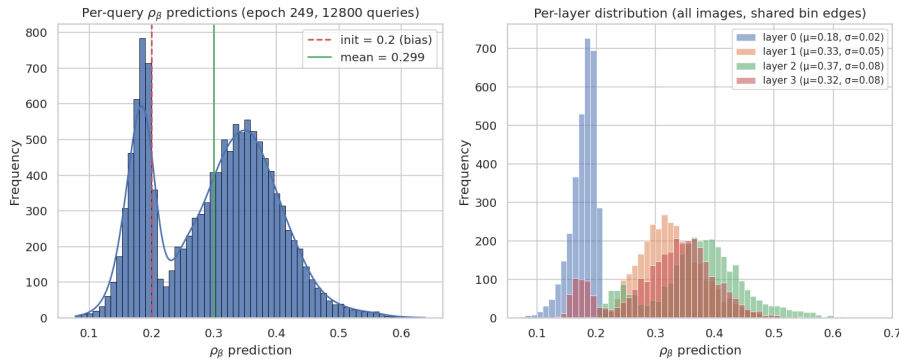


Figure 5.6: Predictions of the ρ_β anisotropy ratio predictor per query in total (left). Predictions of the ρ_β anisotropy ratio predictor per query and decoder layer (right).

In Figure 5.6 the distribution of ratio predictions is bimodal. This follows the known behaviour of the training dataset which has an overwhelming majority of objects with a ratio of less than 0.1 which quickly drops off to a long tail. This highlights that the function learns actual values based on the underlying distribution for its

predictions. Another interesting result is that the distribution per decoder layer varies a lot, with the mean and variance of the distribution of the predictions per layer increasing until the final layer.

5.3 Ablation: Angle Loss Coefficient

As the main tuning knob our model introduced, we wanted to see how the model performed in and out of distribution for different angle coefficients λ , as well as compare them to the baseline which uses no angular supervision.

Table 5.5: Angle loss coefficient ablation. Values are segm AP gain over the baseline (pp). $\lambda_{\text{angle}} = 0.5$ is the rotation-generalization optimal value: lower values under-train the angle head and reduce in-distribution gain; higher values can assist at in-distribution data but falters at generalization and may even perform worse than the baseline.

Split	$\lambda = 0.25$	$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 2.0$
In-distribution	+0.76	+1.97	+2.48	+0.87
$ \theta = 22.5^\circ$	+3.62	+5.14	+3.77	+3.82
$ \theta = 45.0^\circ$	+0.95	+2.20	-0.24	+1.05
$ \theta = 67.5^\circ$	+1.28	+1.50	+1.17	+0.44
$ \theta = 90.0^\circ$	+1.97	+0.83	+0.18	+0.08
Mean rotated	+1.95	+2.42	+1.22	+1.35

Looking at the table we see that $\lambda = 0.5$ generalizes best and outperforms the other coefficients in all but one out-of-distribution datasets. At the same time, there is no clear monotone change of mean AP as λ changes, suggesting that the true optimal value in this case may lie somewhere below or above 0.5.

6

Discussion

This chapter will go over in detail and provide a deeper insight into the causes and implications of the results presented in the previous chapter.

6.1 DINOv3 backbone with minimal head

We tried a couple of different approaches with a DINOv3 backbone and a small head, and we will discuss the results of two main approaches:

- Using some similarity metric between a known foreground patch and other patches.
- Using a classifier head atop the backbone for foreground segmentation.

We tried using these techniques for semantic segmentation, which we deem to be a good yardstick for whether the technique is viable for instance segmentation. If we cannot precisely segment boards, we will not be able to find bounding boxes and segment boards within them. It might seem like segmenting a single board within a bounding box is a much easier task than segmenting all boards in an image. However, while this might be true in most cases, in some of the most interesting cases where boards are very close together or lying at an angle, the bounding boxes can be essentially as difficult to segment as a full image (see Figure 6.1).

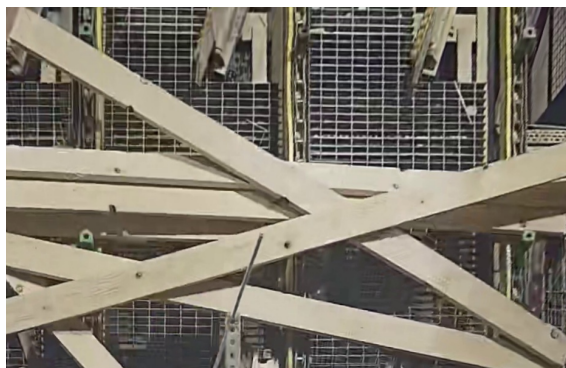


Figure 6.1: A messy bounding box. Sometimes, segmenting the boards within a bounding box can be just as challenging as segmenting the boards in an entire image.

In the early testing phase, our results were promising. Unfortunately, there were some glaring issues with both cases. Firstly, when several boards lie close to each other, the classifier cannot tell them apart. This is because it simply assigns each

patch to be either *board* or *not board*, so if a patch is comprised of only board, but from two different boards, it will be classified to be 100% board, the same as if it was just the middle of a single board (see Figure 6.2). To avoid this issue, a logistic



Figure 6.2: A simple binary classifier can have trouble differentiating between a single wide board and two boards lying next to each other, even when there is a clear border between the boards.

classifier was trained on a dataset where the masks were split so that each image contained a single board. The hope was that this would create a classifier capable of finding a single board from an image, but the results were insufficient. The reason for this is probably that instead of every input image having a single ground truth mask, every input image had several ground truth masks, one for each board in the image. A logistic classifier in this case is essentially a function from patch embedding space to $\{0, 1\}^{H \times W}$. The problem is that a function must have a singular output for each input, which we do not have when each image has several possible *single plank segmentations*.

Another problem appears with very thin boards. As the patches are 16×16 pixels, if there is a very thin board (less than 16 pixels wide), some pixels outside the board will inevitably be labelled as being *board* (see Figure 6.3). We did find a way to



Figure 6.3: With thin boards there will be a significant amount of pixels outside of the actual board being labelled as *board*.

mitigate this somewhat by creating two masks, where one is offset by half of the patch width and height; these can then be multiplied together to get, essentially, a mask with higher resolution (see Figure 6.4). This method works quite well (see Figure

0.2	0.2	0.6	0.6
0.2	0.2	0.6	0.6
0.8	0.8	1	1
0.8	0.8	1	1

 $\times$

0	0	0	0.2
0.4	0.7	0.7	1
0.4	0.7	0.7	1
0.8	1	1	1

 $=$

0	0	0	0.12
0.08	0.14	0.42	0.6
0.32	0.56	0.7	1
0.64	1	1	1

Figure 6.4: Imagine a diagonal divide between a dark and light section. If we have a "patch resolution" of 2×2 we will get a very choppy edge, but if we produce two masks with "patch resolution" 2×2 where one is offset by 1 both horizontally and vertically, we can multiply them elementwise to simulate a mask with patch resolution 1×1 .

6.5) but has a big problem. In order to obtain the second set of patch embeddings, we need to run the backbone again, thus completely nullifying the entire point of using a light-weight head atop a large backbone.

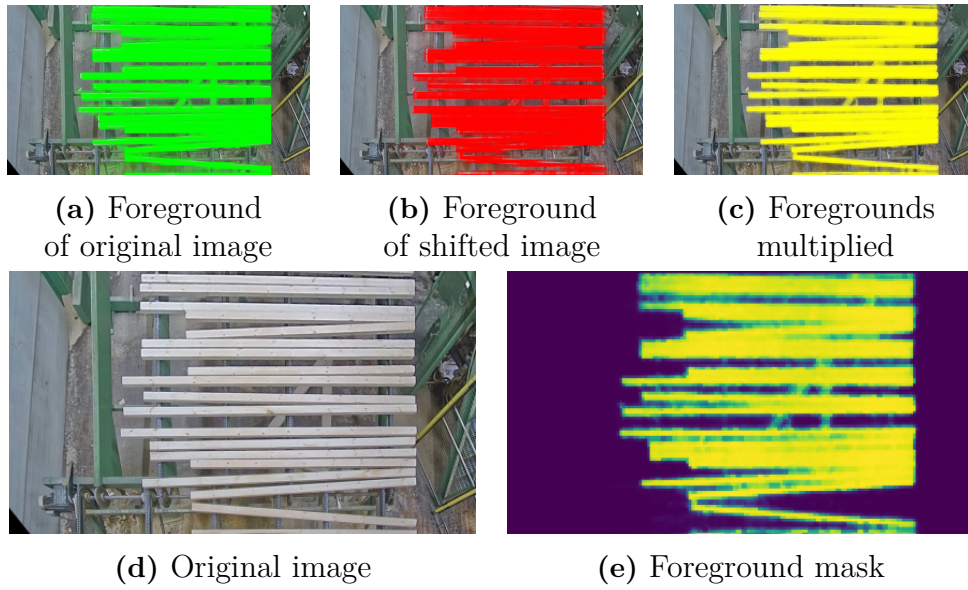


Figure 6.5: We obtain two foreground masks by shifting the original image before obtaining the DINO patches and then classifying each set of patches individually.

These masks can then be multiplied together to obtain a mask with basically double the resolution.

Both of these issues are inherent to approaches treating each DINO output patch as a single data point, meaning any such approach would not be suitable for process surveillance.

6.2 Model Evaluation

By the results, it is clear that our model outperforms a standard RF-DETR model on our dataset. What is interesting is how the behaviour of the λ_{angle} coefficient affects not only the total result, but how it interacts with the in-distribution and rotated data differently. One could imagine that it would be potentially interesting to only tune the coefficient to maximize the in-distribution data if you know that the majority of your objects are not going to behave differently, meaning that from Table 5.5 a $\lambda_{angle} = 1$ could potentially be optimal for some use cases.

If we analyse the anisotropy ratio predictor ρ_β in Figure 5.6 together with the wanted behaviour of the residual angle refinement, we can see that it has an interesting emergent property. While the original idea of ρ_β was to predict a consistent width matching that of the object, the distribution of ratios in Figure 5.6 (right) shows that layer 0 learns to always sample a tight area, which then expands in layer 1 to a wider sample space. The behaviour then changes in the final 2 layers as it seems to spread out to a bimodal distribution again, seemingly following the distribution of the objects in the validation set. Another interesting point to highlight is that the mean of both peaks appears to be greater than the mean of the original dataset. This could suggest that the optimal sampling space may not be to get as close a fit as possible to the object, but rather that the model wants to at some point see some

other objects or empty space around the query.

6.3 Future Work

While the anisotropic priors and angle predictions have led to a positive result, there are still possible future improvements on the model. The most obvious place to start would be to apply the orientation-conditioning to the mask head, as it is only applied to the query right now.

Another possible route would be to work on the canonical frame sampling and ratio predictor. Right now, we see that the predictor is very flexible and actually learns different sampling patterns per decoder layer. Originally, the idea behind this implementation was that the model should learn angled objects in the same way as axis-aligned ones. Since the beta scale is gated by the angle, it means that angles closer to 45 degrees are affected more by the flexibility, whereas axis-aligned boxes are fully rigid. It could be interesting to try making the predictor more rigid by only allowing a single width across all layers and comparing this to the removal of the gating to allow for boxes of all angles to have a fully flexible sample space.

7

Conclusion

For handling instance segmentation in industrial environments where objects are known to be long and thin and strong robustness is required, we propose ROMAn-DETR — a novel approach for robust instance segmentation using anisotropic priors and orientation-conditioning which outperforms a same-size state-of-the-art RF-DETR segmentation model by up to 10.3% mAP on our out-of-distribution dataset. To achieve this, we combine the priors and per-query orientation to use per-query canonical frames, which allows the model to learn the orientation of the same object in the same way. This is done by rotating the sampling space to that of the object and then applying scale factors to convert it to its new coordinate system, which minimizes the search space for attention sampling, speeds up convergence, and improves mAP .

The usage of a minimal head with a DINO backbone for segmentation was also explored, but we deem a frozen ViT backbone with a minimal head unsuitable for the application of instance segmentation on boards in a sawmill.

Bibliography

- [1] H. D. Block. The perceptron: A model for brain functioning. i. *Rev. Mod. Phys.*, 34:123–135, Jan 1962.
- [2] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks, 2017.
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
- [5] Rishi Bommasani; et al. On the opportunities and risks of foundation models, 2022.
- [6] Kunihiro Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.
- [7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [8] Cari Jean. A brief history of sawmills, May 2025.
- [9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [10] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.
- [11] David Redó Nieto. Rf-detr under the hood: The insights of a real-time transformer detection, 2025.
- [12] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision, 2024.
- [13] Md Raisul Islam, Md Zakir Hossain Zamil, Md Eshmam Rayed, Md Mohsin Kabir, M. F. Mridha, Satoshi Nishimura, and Jungpil Shin. Deep learning and computer vision techniques for enhanced quality control in manufacturing processes. *IEEE Access*, 12:121449–121479, 2024.

- [14] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [15] Isaac Robinson, Peter Robicheck, Matvei Popov, Deva Ramanan, and Neehar Peri. Rf-detr: Neural architecture search for real-time detection transformers, 2026.
- [16] Grant Sanderson. Transformers, the tech behind llms | deep learning chapter 5. YouTube, 3Blue1Brown, 2024. https://www.youtube.com/watch?v=wjZofJX0v4M&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=6.
- [17] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2019.
- [18] Philipp Schmitt. A whiggish history of computer vision, Jun 2021.
- [19] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90:227–244, 10 2000.
- [20] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24:35–43, 2001.
- [21] Richard Szeliski. Vision transformer with deformable attention, 2022.
- [22] Abdelghani Tbakhi and Samir S. Amr. Ibn al-haytham: Father of modern optics. *Annals of Saudi Medicine*, 27(6):464–467, 2007.
- [23] Yustina S Tjahjaningsih, Moses L. Singgih, Putu Dana Karningsih, and Chastine Fatichah. Deep learning-based quality control for surface defect detection in particle board using cnn and resnet50 toward zero defect manufacturing. In *2025 International Conference on Artificial Intelligence's Future Implementations (ICAIFI)*, pages 143–148, 2025.
- [24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [25] Robert H. Wurtz. Recounting the impact of hubel and wiesel. *The Journal of Physiology*, 587(12):2817–2823, Jun 2009.
- [26] Zhuofan Xia, Xuran Pan, Shiji Song, Li Erran Li, and Gao Huang. Vision transformer with deformable attention, 2022.
- [27] Savaş Yıldırım and Meysam Asgari-Chenaghlu. *Mastering transformers: The journey from Bert to large language models and stable diffusion*. Packt Publishing Ltd, 2024.
- [28] Ibrahim Yousif, Jad Samaha, JuHyeong Ryu, and Ramy Harik. Safety 4.0: Harnessing computer vision for advanced industrial protection. *Manufacturing Letters*, 41:1342–1356, 2024. 52nd SME North American Manufacturing Research Conference (NAMRC 52).
- [29] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.

A

Appendix

A.1 Data

Table A.1: Augmentations used on the dataset during training.

Augmentation	Parameters	Probability p
<i>Geometric</i>		
HorizontalFlip	—	0.5
Affine	rotation $\in [-90^\circ, +90^\circ]$	1.0
	scale $\in [0.85, 1.15]$	
	border = reflect	
<i>Photometric</i>		
RandomBrightnessContrast	brightness ± 0.2 , contrast ± 0.2	0.5
GaussianBlur	kernel limit = 3	0.3
GaussianNoise	per-channel std $\in [0.01, 0.05]$	0.3

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



CHALMERS
UNIVERSITY OF TECHNOLOGY