Exploring DETR for Recognition and Detection of Traffic Signs

Andreas Hove Paludan
Department of Computer Science
Aalborg University
ahpa20@student.aau.dk

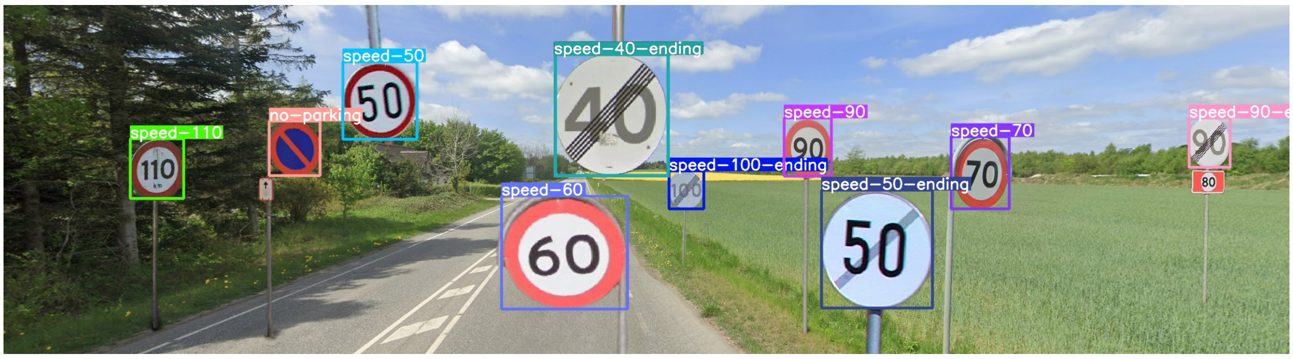
Jakob Frederik Lykke
Department of Computer Science
Aalborg University
jlykke20@student.aau.dk

Kasper Østergaard Nielsen
Department of Computer Science
Aalborg University
knie20@student.aau.dk

Martin Langgaard Jacobsen
Department of Computer Science
Aalborg University
mjacob20@student.aau.dk

Peter Schwartz Lauridsen
Department of Computer Science
Aalborg University
plauri20@student.aau.dk

June 26, 2025



Abstract

The increased attention towards Advanced driver-assistance systems, such as self-driving vehicles and automated driving, has had the interest in traffic sign recognition and detection rise to higher prevalence. This paper explores the use of DEtection TRansformer (DETR) for recognition and detection of traffic signs. We created a custom dataset on which DETR and multiple baselines were trained and evaluated to compare their performances. Our results show that DETR can not outperform state-of-the-art methods but might have the potential to do so with further research on accommodating the shortcomings of DETR. Therefore, we also evaluated the Deformable DETR, which showed significant improvements compared to the original DETR.

1. Introduction

With the increased attention towards self-driving vehicles and automated driving, the tasks of traffic sign recognition and detection (TSRD) have become evermore prevalent. A TSRD system is essential for self-driving cars and driver assistance systems. These technologies play a considerable role in traffic safety and transport efficiency. This is achieved by assisting drivers with systems that can help detect risks of accidents or automatically adjust the speed of the vehicle. Advanced driver assistance systems (ADAS), in particular, make use of TSRD by detecting speed limits and other relevant information to automatically adjust speed or assist in merging. Given the importance of the task of TSDR, research is still being done, and trends in the field change as new technologies emerge. [1] has compiled the most impactful published studies around TSRD.

Detection and classification of traffic signs have been performed with various computer vision methods following the growing interest in ADAS. However, no model is perfect, and each has its drawbacks. Models utilizing shape detection struggle when an object is partly blocked, and color-based methods struggle when faced with changes in illumination. With varying weather conditions and the diurnal cycle, both occlusion and illumination changes of traffic signs are very real issues in the task of TSRD. With vehicle-mounted sensors and cameras, blurring, and vibration from the vehicle also impair the ability of the ADAS to perform optimally.

TSRD methods can be divided into three overall categories, color-based, shape-based, and machine learning-based methods[2]. Color-based methods utilize the stark colors of traffic signs to help detect the signs. While it would seem that color-based methods have an advantage given the color-coded nature of traffic signs, they are extra sensitive to changes in illumination. Instead, shape-based methods use the fact that traffic signs are always shaped in specific ways e.g. circles or rectangles. These methods do not share the weakness of color-based approaches, as they do not rely on the colors of the signs. Some color-based approaches have combined the shape and color aspects of traffic signs to perform their detection. However, both shape and color-based methods have mostly proven not to be robust enough for real-world applications. Instead, machine learning-based approaches have become increasingly popular with neural networks, particularly convolutional neural networks (CNN), being used to create effective solutions. Single-shot detectors (SSD) have also been used to great effect in detecting traffic signs. While it is difficult to confidently evaluate which method is the definitive choice due to factors such as training time and difference in datasets used, most recent research looks at CNN and SSD-based approaches.

In recent times, transformers[3] have gotten a lot of attention and have been applied to many areas, including computer vision, specifically object detection. While pure vision transformers (ViTs)[4] have been shown to perform worse than CNN counterparts, a combination of transformer and CNN has shown promising performance[5]. Another CNN-transformer combination that has shown excellent performance outside the field of TSRD is the DETR model[6]. The authors of [6] have shown that their model could compete with other SOTA object detection models at the time of release.

In this paper, we study the problem of TSRD by comparing different models to assert whether a transformer-based approach has the potential to perform on par with SOTA CNN and SSD-based methods. We propose to use the DETR model as a solution to TSRD, as it combines CNN backbone with a transformer. We want to assert whether the DETR model can compete with other SOTA models. To do this, we conduct experiments comparing different performance metrics of the DETR model to SOTA detection models. Our experiments show that the DETR model could not outperform the SSD-based YOLOv8s model or the Faster R-CNN model.

The remainder of the paper is structured as follows: Section 2 describes related works in the field of TSRD, including popular methods and publicly available datasets. In section 3, our problem definition is described. A description of the DETR model and its components is shown in section 4. Section 5 presents our experiments and evaluates the DETR model. Finally, we conclude our work in section 6.

2. Related Work

In this section, we provide a brief overview of traffic sign recognition and detection tasks and the most common methods used. We begin by briefly mentioning publicly available datasets used in the field of TSRD, followed by the methods currently being used.

2.1. Current Traffic Sign Databases

A traffic sign dataset is essential when training and testing a TSRD system. There exist several publicly available traffic sign datasets. Among the most popular are the German Traffic Sign Recognition Benchmark (GTSRB)[7] and the Swedish Traffic Sign Dataset (STSD)[8]. The images in both datasets are fully annotated and ready to be used for training, validation, and test data.

2.2. Methods of TSRD

There are many existing proposed methods of performing TSRD. [9] and [10] have both proposed deep learning solutions using a variation of Mask R-CNN[11] to detect and recognize traffic signs with good results. Mask R-CNN is composed of two modules, the first being a Region Proposal Network (RPN) and the second being a Fast R-CNN module. Both modules work as a single network sharing their convolutional features. Mask R-CNN also uses Feature Pyramid Network (FPN)[12] to improve object detection on smaller objects.

With the popularity of Transformers[3] and their great performance, some proposed methods for TSRD have made use of the Transformer-inspired Vision Transformer model. [13] have performed experiments with five different Vision Transformers, all being variants of the state-of-the-art (SOTA) Vision Transformer Model (ViT)[4]. A ViT firstly slices an image into patches, essentially dividing the image into sections. The input image is split into a sequence $m = hw/p^2$ patches, where h and w denotes the height and width of the input image, and p denotes both the height and width of the patch. Following the original idea of the traditional transformer, the patches are flattened to a vector of length cp^2, thus letting the patches be treated similarly to tokens in a text string. The patches are then linearly projected and summed with positional embeddings to retain the positional information of each patch. [13] found that CNN-based approaches performed better than the ViT-based methods in traffic sign classification tasks. However, they also found that with longer training time, the performance gap narrowed.

[5] uses a Pyramid Transformer architecture[14] to detect traffic signs. The Pyramid Transformer also uses a ViT and applies a pyramid structure to increase performance in dense prediction tasks, achieved by utilizing the pyramid structure's progressive shrinking strategy. In the pyramid structure, each stage of the transformer progressively shrinks the output resolution. At each shrinking stage, the pyramid transformer embeds multi-scale context into the image patches. This means that at each stage of the transformer, the features of the current-scale image are retained within a multi-scale feature map. While the method presented in [5] is in its infant state, the authors did see promising performance on the GTSDB[15] dataset.

Studies in the field of TSRD have shown that CNN-based approaches have the most promising performance, while newer technologies in the form of ViTs have been shown to not perform on par with CNN-based approaches[13]. However, [13] did show that the performance gap between ViT and CNN-based methods narrowed with increased training time. And while CNN-based approaches do solve the tasks of TSRD to significant effect, it is still worth looking into the capabilities of transformer-based approaches for TSRD. Other transformer architectures, such as a pyramid transformer architecture, have been shown, by [5], to have great performance when paired as a backbone with R-CNN. This study shows that transformer-based approaches to TSRD still have potential, and other transformer-based methods are still yet to be explored.

3. Problem Definition

In this section, we present our problem definition.

Our goal is to evaluate the performance of the DETR model for detecting and recognizing traffic signs to assess if DETR can compete with SOTA object detection and recognition models. The model takes an input image $x_{img} \in \mathbb{R}^{3\times H_0 \times W_0}$, where 3 is the number of color channels, H_0 and W_0 denotes the initial height and width of the input image with added 0-padding. The output of the DETR model is a set of bounding boxes with their coordinates B and a class label c, as shown in equation (1).

$$f(x_{img}) \mapsto B, c \in C \tag{1}$$

4. Methodology

In this section, we describe the DETR model, which we want to evaluate for traffic sign detection and recognition. We first show a model-architecture diagram followed by a more in-depth description of the model as a whole.

4.1. The DETR Model

The model architecture is shown in figure 1. The first component of the model is a CNN backbone that takes a batch of images on the form

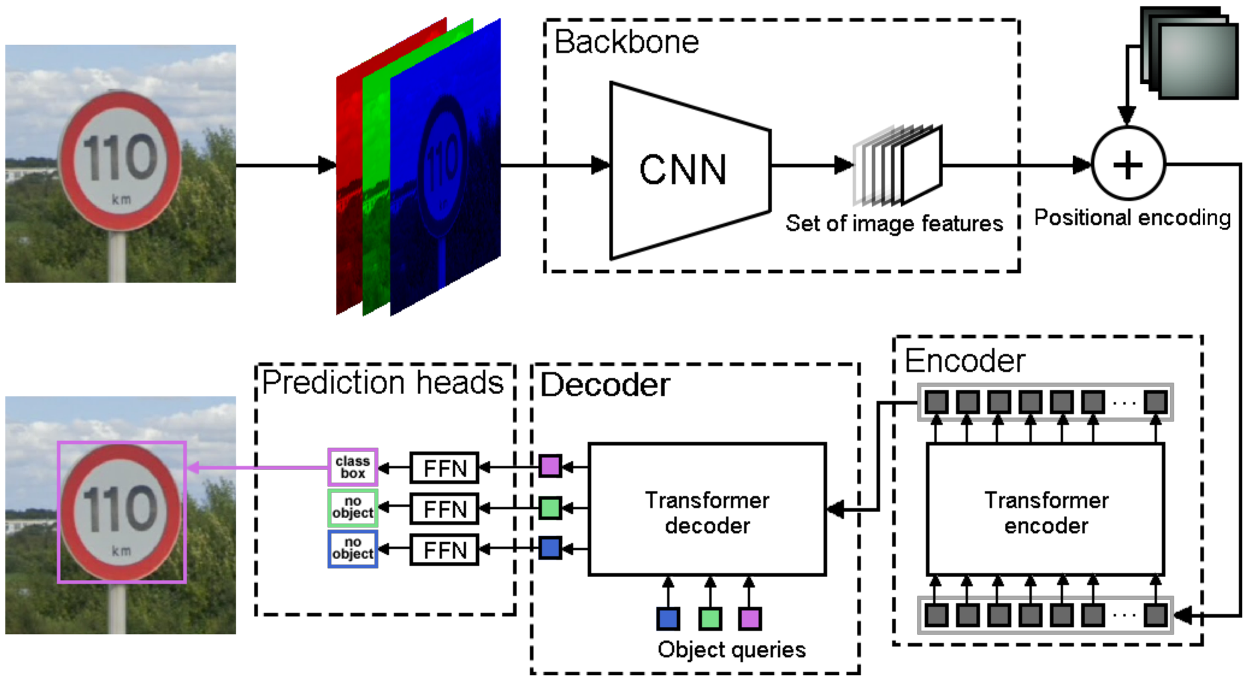


Figure 1: The DETR model framework. It contains four main steps. First, a CNN backbone takes an input image and produces a lower-resolution feature map of the image. The feature map is collapsed into a one-dimensional sequence and supplied with fixed positional encodings. Next, the collapsed feature map is fed to the transformer encoder, which outputs encoded representations of the feature map to the decoder. The decoder takes N object queries, meaning learned positional embeddings, as input alongside the output of the encoder. The decoder attends to each input and produces output embeddings, that are each passed to a prediction head as a feed-forward network. Each prediction head outputs a detection of both a bounding box and an object class, or a "`no object`" class.

$x_{img} \in \mathbb{R}^{3\times H_0\times W_0}$ as input. An image consists of three matrices, one for each color channel. The input images are 0-padded, giving all images in a batch the same dimensions as the largest image in the batch. Then the matrices are fed into the CNN, which will output a set of activation maps containing relevant abstract parts of the images. The activation maps are represented on the form $f \in \mathbb{R}^{C\times H\times W}$. By providing the activation maps of the input images to the transformer, the CNN backbone gives a better starting point for the transformer part of the DETR model. The CNN backbone is pre-trained, which has the benefit of not needing as much data as when building a model from the ground up.

4.1.1. CNN Backbone

Before the activation maps are passed on to the encoder of the model, the activation maps are flattened into vectors and embedded with positional encodings. These positional encodings ensure that positional information on each feature is kept. The reason for flattening the activation maps is that the encoder of the model expects a sequence as input. The DETR model is heavily inspired by the architecture of the original transformer model described in [3]. Therefore, as the encoder from [3] takes a sequence of tokens, so does the DETR model.

4.1.2. The Transformer Encoder and Decoder

Figure 2, the DETR transformer architecture[6], shows the DETR encoder, consisting of four sub-components. The first is a `multi-head self-attention` component, which runs through an attention mechanism multiple times, equal to the number of heads. Each head produces an attention matrix, which is then concatenated and multiplied by another weight matrix, in order to produce a single matrix, which the following components can work with. The second component is an `add and normalize` component, which adds the concatenated matrix with a vector of input embeddings. The resulting matrix is then normalized and passed on to the third component which is a `feed-forward neural network` (FFN) whose purpose is to process the output of the encoder's attention

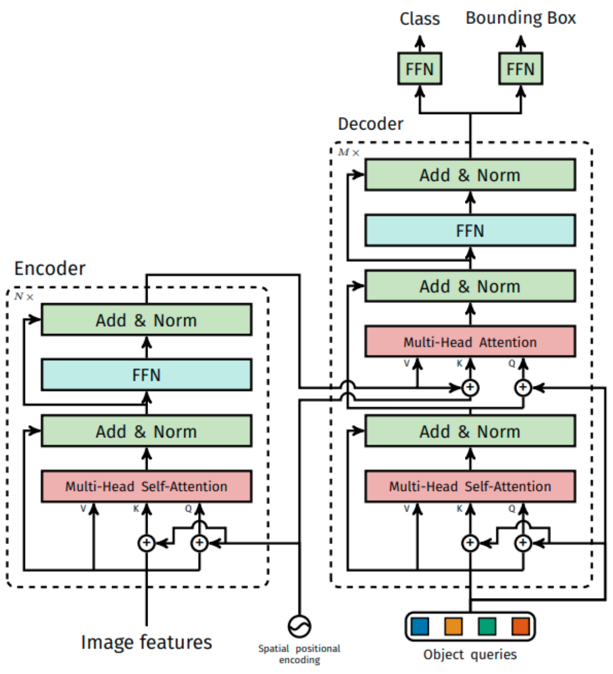


Figure 2: Architecture of the DETR transformer encoder and decoder[6]. See section 4.1.2 for a detailed description of the figure.

component to better fit the attention component in the decoder. Lastly, the output of the FFN is passed through a second `add and normalize` component, preparing the output for the decoder.

The DETR decoder is shown in figure 2[6] and is comprised of six sub-components. Four of which are nearly identical to the sub-components in the encoder. Firstly, the decoder receives N object queries, all initially with a value of 0. The decoder will produce the same number of object queries as output. The decoder takes the output from the encoder as a side input of conditioning information. The decoder firstly takes the N object queries and feature encodes them, exactly like the encoder, using a `multi-head self-attention` and an `add and normalize` component. The encoded object queries are then combined with the image information from the encoder. This means that the decoder has access to the image information from the encoder when moving on to its second `multi-head self-attention` component. From here, the decoder follows the same structure as the encoder, producing a singular attention matrix for each image, where input embeddings are then added before being passed on to an FFN.

Since both the encoder and decoder use an attention mechanism, we will further explain how attention works. The model uses both multi-head attention and multi-head self-attention. Multi-head attention is the concatenation of M single-head attention. Single-head attention is commonly calculated in two ways, either by additive attention or by dot-product. The DETR model uses a dot-product between queries and keys. After computing the dot-product, it takes the softmax of the dot-product resulting in two features with a high dot-product score being more prominent, while those with a low score will not be as noticeable.

To further elaborate on how attention works, we have made two visualizations. Figure 3 is a simplified illustration. The pictures on the left are traffic signs without attention, and on the right, are the same traffic signs but with attention. In the pictures to the right, it can be seen how the attention only focuses on some parts of the images and the other areas are greyed-out, which are the places that are not attended to.

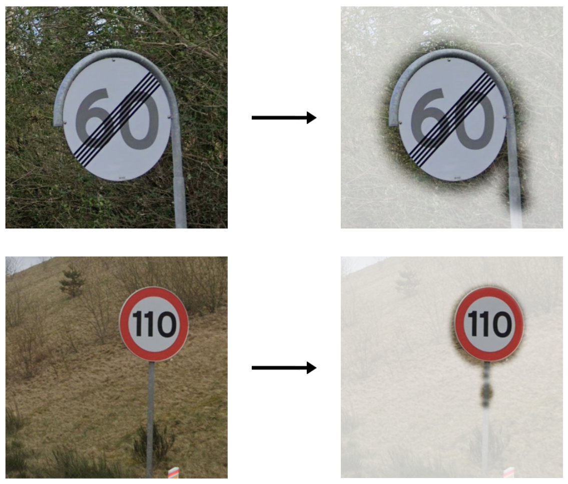


Figure 3: Visual representation of how attention can focus on specific parts of an image. Left are images before attention and right are images after attention.

The second visualization, figure 4, shows how different image features attend to each other. An input image has its features extracted (for simplicity, in this illustration, four features are extracted). Each feature has attention to every other feature, illustrated with the lines connecting the features. In reality, for self-attention, features will also have attention to themselves, but for the sake of simplicity, we have excluded that for this example. As seen, some of the lines are less pronounced than others. This is to illustrate the amount of attention between each feature, thus the more related the features are, the darker the color of the line. By being able to relate these features to each other through attention, the model can theoretically produce much better results than if they were all treated separately.

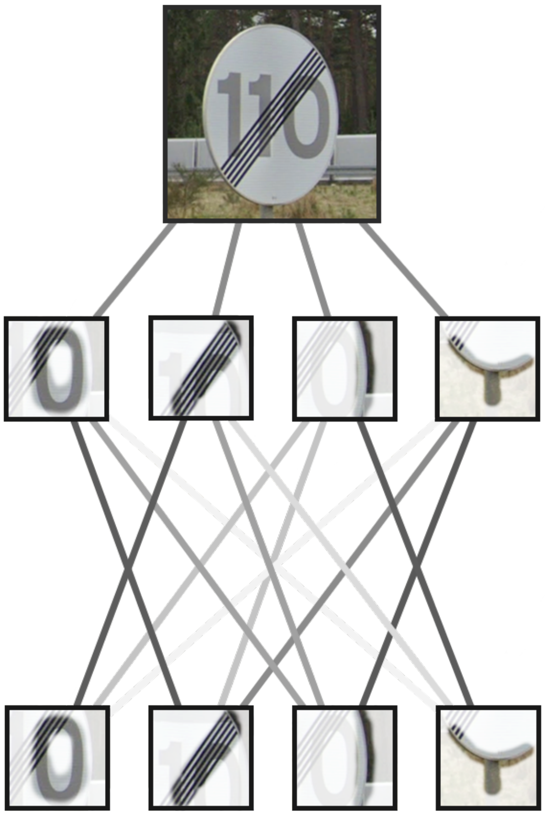


Figure 4: Visual representation of how image features attend to each other. More pronounced connective lines represent a stronger relation between features.

4.1.3. Feed-Forward Network

The feed-forward network is part of the prediction heads in figure 1, which makes the last predictions using a 3-layer perceptron with a ReLU activation function, a hidden dimension d, and a linear projection layer. The FFN predicts the normalized center point, width, and height of the bounding boxes while the linear layer predicts the class label using a softmax function. Since we make predictions on a fixed number of bounding boxes that is usually much bigger than the number of actual objects in the image, we also need a class label Ø that represents that no object was detected in that box.

4.1.4. Loss Function

The loss function uses the predictions to calculate the loss, which is used to update some parameters to get better results in the next iteration of computation. The loss function calculates the loss between the predicted labels and bounding boxed compared to the ground truth. In equation (2), $y_i = \{y_i\}_{i=1}^{N}$ is the set of N ground truth objects, and $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ the set of N predictions. Provided that the amount of N is larger than the number of objects present in the image. For this there needs to be the same amount of objects in y as the amount in $\hat{y}$. There are usually more objects in $\hat{y}$ than in y to ensure there is the same amount, the model pads y with Ø. To find the matching between the two sets, the model searches for a permutation of the elements $\sigma \in \mathfrak{S}_N$ with the lowest cost:

$$\hat{\sigma} = \underset{\sigma \in \mathfrak{S}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}), \tag{2}$$

$\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ is the function that matches the predictions and ground truths, this is a pair-wise matching. The reason why the model uses pair-wise matching is that by matching every prediction to one ground truth it teaches the model to only give the best prediction instead of multiple predictions that are worse. To efficiently compute these matchings, the model uses the Hungarian algorithm [16]. The algorithm takes both the class and bounding box predictions and matches them to the ground truth. Each element in the ground truth can be seen as $y_i = (c_i, b_i)$ which contains its own class label c_i which can be Ø. The ground truth box $b_i \in [0,1]^4$ which is the center coordinates, height, and width relative to the picture. For the predictions that have the index $\sigma(i)$ the model denotes the probability of class as $\hat{p}_{\sigma(i)}(c_i)$ and the predicted bounding boxes as $\hat{b}_{\sigma(i)}$. $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ can be denoted as: $-\mathbb{1}_{c_i \neq \emptyset} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{c_i \neq \emptyset} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$

Now that the model has found all the pairs, all of the loss need to be computed, this is done with the Hungarian loss function [16], which can be seen in equation (3).

$$\mathcal{L}_{Hungarian}(y, \hat{y}) = \sum_{i=1}^{N} [-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\hat{\sigma}}(i))] \tag{3}$$

In equation (3), $\hat{\sigma}$ is the optimal assignment found in equation (2). To lessen class imbalance, the model down-weight the log-probability when the class is Ø with a factor of 10. There is a special case with the matching cost when matching between an object and Ø, where its cost would be constant. Next is the loss for the bounding boxes, which is done with the box predictions. These predictions are made directly, unlike many other detectors which use some initial guesses as a reference. However, it does produce an issue, which is the scaling of the loss. To lessen the issue, the model uses a linear combination of the ℓ_1 loss and the generalized intersection over union (IoU) loss. The reason why the model uses both is that the ℓ_1 is less sensitive to outliers but has different scales for small and large boxes. The problem with the scales is that the IoU is scale-invariant. The model's box loss can be defined as $\lambda_{iou}\mathcal{L}_{iou}(b_i,\hat{b}_{\sigma(i)}) + \lambda_{L1}\|b_i - \hat{b}_{\sigma(i)}\|_1$ where $\lambda_{iou}, \lambda_{L1} \in \mathbb{R}$ are hyperparameters. After this, the two losses are normalized based on the number of objects within the batch. For the IoU we can expand it:

$$\mathcal{L}_{iou}(b_{\sigma(I)},\hat{b}_i) =$$

$$1 - \left(\frac{|b_{\sigma(i)} \bigcap \hat{b}_i|}{|b_{\sigma(i)} \bigcup \hat{b}_i|} - \frac{|B(b_{\sigma(i)},\hat{b}_i) \setminus b_{\sigma(i)} \bigcup \hat{b}_i|}{|B(b_{\sigma(i)},\hat{b}_i)|}\right) \tag{4}$$

In equation (4), $|x|$ denotes the area of x, while the union and intersection are shorthands for the boxes. The areas of unions or intersections are computed by min/max of the linear functions of the predictions and the ground truth. $B(b_{\sigma(i)},\hat{b}_i)$ is the largest box containing both the prediction and ground truth, this is also computed with a min-max linear function.

5. Experiments

In this section, we perform various experiments on the DETR model's ability to perform TSRD. We also compare the DETR model to other state-of-the-art(SOTA) object detection models. First, we present our custom dataset of speed limit and no-parking traffic signs.

5.1. Data Processing

We have created a custom dataset since we could not find any datasets that were representative of the speed limit signs in Denmark. We wanted to work with Danish traffic signs since that is what we are most familiar with. The dataset contains speed limit and no-parking signs and is a combination of images from other datasets and images we have annotated ourselves. These images consist of various traffic signs taken in different scenes where multiple speed limits and their respective endings[1] are placed.

This dataset contains images from the GTSRB[7], the Chinese Traffic Sign Database[17], and the DFG Traffic Sign[9] dataset. The dataset contains a total of 2403 images which are split into training, validation, and test sets at a ratio of 70:20:10. The image size varies a lot, as can be seen in figure 5. The median size of all images is 695x552.

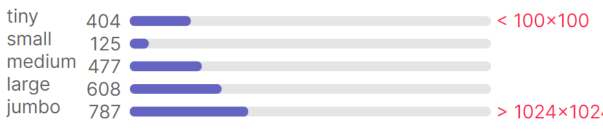


Figure 5: Distribution of size in our dataset.

Figure 6 shows the distribution of class labels in our dataset. As is shown, our dataset is slightly imbalanced and has an uneven distribution of labels.

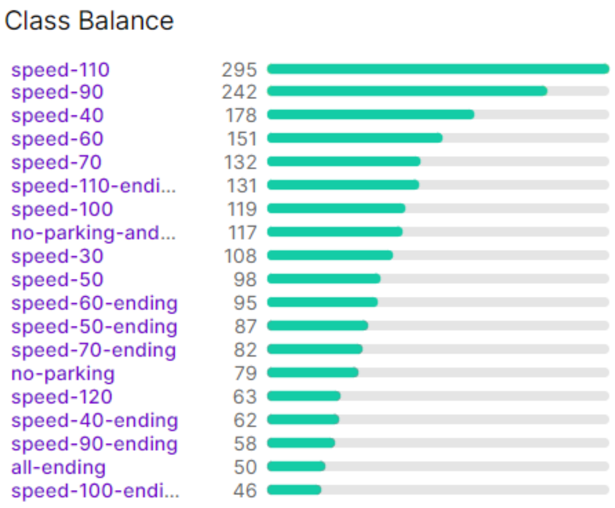


Figure 6: Distribution of class labels in our dataset.

5.2. Other models

We compare the DETR model to other SOTA models like Faster R-CNN and YOLOv8. Let us first give a brief description of both. Despite the original model being released in 2015, Faster R-CNN can still achieve great accuracy with the right backbone. We chose a combination of ResNet and a feature pyramid network from 2022[18] as the backbone. We also wanted

[1]When speed limitations are ceasing

to compare against a more recent model, so we chose YOLOv8, which was released in January 2023 and is available in five models of different sizes. We selected YOLOv8s because its inference time is similar to DETR.

5.3. DETR

The DETR model can be used with any conventional CNN backbone. However, since Meta AI provided pre-trained models for DETR, we used detr-resnet-50 and detr-resnet-101-dc5. The detr-resnet-50 model is small in size but less accurate, while the detr-resnet-101-dc5 is significantly larger with a minor improvement in accuracy.

5.4. Evaluation

We trained all models on an Nvidia A40 GPU for 250 epochs. The models were evaluated on the IoU metric for bounding box Average Precision (AP) and Average Recall (AR). The training results can be seen in table 1.

Model	AP50-95	AP50	AR50-95	Training hours
YOLOv8s	0.949	0.981	0.962	4
Faster R-CNN R50-FPN	0.896	0.952	0.905	18
Deformable DETR-R50	0.863	0.914	0.885	10
DETR-R101-DC5	0.740	0.907	0.777	10
DETR-R50	0.716	0.804	0.750	9

Table 1: Comparison of object detection models trained on our own data set.

As seen in table 1, neither the DETR-R50 model nor the DETR-R101-DC5 model can compete with the YOLOv8 model. However, since we are using the YOLOv8 model from 2023 and the DETR from 2020, it is expected that DETR would be at a disadvantage. As described previously, the DETR model uses a CNN backbone, while YOLO is based solely on CNN. We have also described how adding components from transformers can theoretically increase precision and recall. Therefore we expect that there is potential for using transformers for object detection, and new designs could improve on the weaknesses of DETR.

[6] noted that DETR has worse accuracy on small objects than other models. However, it is better at detecting large objects and shows an 11% improvement on the COCO validation set over Faster R-CNN in the AP_L metric[6]. Since the release of DETR in 2020, several improved models have been developed. We have tested the Deformable DETR model[19], which should be more accurate on smaller objects. The model introduces a multi-scale deformable attention module that learns to attend to features at different scales. In addition, it only attends to a small number of relevant features, thus reducing training time. We will not go into more detail with the Deformable DETR model since it is not the focus of this paper.

We have decided to train the Deformable DETR model multiple times. First, the model was trained for 145 epochs, then for an additional 80 epochs with a lower learning rate. By doing this, we were able to get the model to converge fully. As seen in table 1, the Deformable DETR model performed better than the DETR models coming close to Faster R-CNN. Therefore, we see the potential to improve the DETR model with newer technologies. It should be noted that the Deformable DETR model is from March 2021, whereas YOLOv8 is from January 2023.

Generally, in table 1, the AP and AR evaluation results are all relatively high compared to standard COCO validation sets[20]. For exploring future models on this dataset, it might be necessary to increase the difficulty of the dataset with more challenging images in order to improve generalization. Our custom dataset does not contain any images captured in extreme weather conditions or other challenging scenarios.

6. Conclusion

This paper examines the DETR model for traffic sign detection and recognition by comparing the model with SOTA object detection models. The models were trained and evaluated on a custom dataset, representing Danish speed signs and no-parking signs. The dataset could be expanded upon, including more diverse images, such as images from poor weather conditions. In our work, DETR did not show better AP and AR when compared to newer SOTA models, such as YOLOv8. However, we showed that the Deformable DETR model[19] from 2021 can achieve higher AP and AR than DETR. Deformable DETR shows that DETR is improvable with newer technologies.

In order to approach YOLOv8's performance, other models could be explored in future work, such as the DETR-based DINO model[21]. According to [21], DINO can get an AP of up to 49.4 on the COCO val2017 dataset, whereas the Deformable DETR gets an AP of 41.1.

Bibliography

[1] Ayoub Ellahyani, Ilyas El jaafari, and Said Charfi. Traffic sign detection for intelligent transportation systems: A survey. *E3S Web of Conferences*, 229:01006, 01 2021.

[2] Ellahyani, Ayoub, El Jaafari, Ilyas, and Charfi, Said. Traffic sign detection for intelligent transportation systems: A survey. *E3S Web Conf.*, 229:01006, 2021.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[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] Omid Nejati Manzari, Amin Boudesh, and Shahriar B. Shokouhi. Pyramid transformer for traffic sign detection, 2022.

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020.

[7] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 0(0):–, 2012.

[8] Fredrik Larsson and Michael Felsberg. Using fourier descriptors and spatial models for traffic sign recognition. In Anders Heyden and Fredrik Kahl, editors, *Image Analysis*, pages 238–249, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[9] Domen Tabernik and Danijel Skočaj. Deep learning for large-scale traffic-sign detection and recognition, 2019.

[10] Rajesh Kannan Megalingam, Kondareddy Thanigundala, Sreevatsava Reddy Musani, Hemanth Nidamanuru, and Lokesh Gadde. Indian traffic sign detection and recognition using deep learning. *International Journal of Transportation Science and Technology*, 2022.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[12] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[13] Yuping Zheng and Weiwei Jiang. Evaluation of vision transformers for traffic sign classification. *Wireless Communications and Mobile Computing*, 2022.

[14] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *CoRR*, abs/2102.12122, 2021.

[15] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

[16] Russell Stewart and Mykhaylo Andriluka. End-to-end people detection in crowded scenes.

[17] Yi Yang, Hengliang Luo, Huarong Xu, and Fuchao Wu. Towards real-time traffic sign detection and classification. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):2022–2031, 2016.

[18] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[19] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.

[20] Papers with Code. Object detection benchmarks. `https://paperswithcode.com/task/object-detection`, 2023.

[21] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection, 2022.

[22] Safat B. Wali, Majid A. Abdullah, Mahammad A. Hannan, Aini Hussain, Salina A. Samad, Pin J. Ker, and Muhamad Bin Mansor. Vision-based traffic sign detection and recognition systems: Current trends and challenges. *Advanced Computational Intelligence for Object Detection, Feature Extraction and Recognition in Smart Sensor Environments*, 2019.

[23] Jonathan O'Callaghan. Trusting self-driving cars is going to be a big step for people.

[24] Nick Babich. Transformer architecture: is it the future of ai?

[25] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, Zeng Yifu, Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, November 2022.

[26] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[27] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. *CoRR*, abs/1911.03584, 2019.

[28] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.