Object Detection Using DETR

Kodavali Purnendra Sri Krishnaditya
23125017

Gunta Preetham
23125013

***Abstract*—This report details the implementation and evaluation of the Detection Transformer (DETR) model for object detection on a custom dataset of traffic scenes. DETR represents a paradigm shift in object detection, eliminating the need for hand-designed components like Non-Maximum Suppression (NMS) and anchor generation by framing the task as a direct set prediction problem. We fine-tuned a pre-trained DETR model with a ResNet-50 backbone on a dataset of approximately 31,000 images, targeting 12 distinct object classes. Due to significant computational constraints, the model could not be trained to the extent suggested by standard practice. Despite incomplete convergence, the model achieved a moderate mean Average Precision (mAP) of 0.4233 at an IoU threshold of 0.50. Our evaluation reveals strong performance on large, common classes such as "Car" and "Truck" but highlights significant challenges in detecting small objects and differentiating between visually similar, fine-grained classes like various traffic light states. This work provides a practical analysis of DETR's performance under limited computational resources and offers insights into its strengths and weaknesses on a real-world custom dataset.**

***Index Terms*—Object Detection, DETR, Transformers, Computer Vision, Deep Learning, Transfer Learning.**

I. Introduction

Object detection, a fundamental task in computer vision, has traditionally relied on complex pipelines involving region proposals, anchor boxes, and post-processing steps like Non-Maximum Suppression (NMS). While models like YOLO and R-CNN variants have achieved state-of-the-art performance, their architectures often include hand-designed components that can be intricate and difficult to tune.

The DETR (DEtection TRansformer) model, introduced by Carion et al. [1], offers a novel, end-to-end approach that simplifies this pipeline. By leveraging a transformer-based encoder-decoder architecture, DETR treats object detection as a direct set prediction problem. It eliminates the need for anchors and NMS by producing a fixed-size set of predictions, which are matched to ground truth objects using a bipartite matching algorithm. This direct, set-based objective function enforces unique predictions per object.

This project aims to implement and evaluate a DETR model on a custom traffic scene dataset. We outline the full methodology, from data preprocessing and model fine-tuning to a comprehensive quantitative and qualitative evaluation of the results.

II. Methodology

A. Dataset and Preprocessing

The initial dataset comprised a large collection of traffic scene images. A crucial first step was data cleaning; images without annotations or with clearly improper annotations were excluded from the dataset. This resulted in a final set of approximately 31,000 viable images. The data was partitioned into training, validation, and test sets with a 75%, 10%, and 15% split, respectively.

The DETR architecture requires annotations in the COCO format. As our initial dataset used YOLO-style annotations (class index and normalized bounding box coordinates: `[x_center, y_center, width, height]`), a conversion script was implemented to transform them into the COCO format, which specifies bounding boxes as `[x_min, y_min, width, height]`.

B. DETR Model Architecture

The DETR model consists of three main components: a CNN backbone for feature extraction, a transformer encoder-decoder, and a feed-forward network (FFN) for final predictions.

1) Backbone: We used a ResNet-50 model, pre-trained on ImageNet, as the CNN backbone. It extracts a lower-resolution feature map from the input image, which is then passed to the transformer.

2) Transformer and Set Prediction: The core of DETR is its transformer architecture. The encoder refines the image features using self-attention, while the decoder takes a fixed number of learned positional embeddings, referred to as "object queries," and cross-attends to the encoder's output to generate object-specific embeddings. Each of these output embeddings is then passed to two parallel FFNs: one for class prediction (including a "no object" class) and one for bounding box prediction.

3) Bipartite Matching Loss: The core innovation of DETR is its set-based loss function, which enforces unique predictions for each object. This process involves two stages: first, finding an optimal one-to-one matching between the model's predictions and the ground truth objects, and second, computing the loss for the matched pairs.

Let y be the set of ground truth objects, padded with $\emptyset$ (no object) to a fixed size N. Let $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ be the set of N predictions. Each prediction $\hat{y}_i$ consists of a class probability distribution $\hat{p}_i(c)$ and a predicted bounding box $\hat{b}_i$.

a) 1. Optimal Assignment ($\hat{\sigma}$): The model must find the optimal permutation $\hat{\sigma} \in \mathfrak{S}_N$ that pairs predictions to ground truths by minimizing a total matching cost. This is

an assignment problem solved efficiently with the Hungarian algorithm.

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i=1}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) \tag{1}$$

Here, $\mathcal{L}_{\text{match}}$ is the pair-wise matching cost between a ground truth y_i and a prediction with index $\sigma(i)$.

b) 2. Pair-wise Matching Cost ($\mathcal{L}_{match}$): The cost used to find the optimal assignment considers both class prediction and bounding box similarity. For a ground truth $y_i = (c_i, b_i)$, where c_i is the class label and b_i is the box, the matching cost is defined as:

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = \mathbf{1}_{\{c_i \neq \emptyset\}} \left[-\hat{p}_{\sigma(i)}(c_i) + \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) \right] \tag{2}$$

The indicator function $\mathbf{1}_{\{c_i \neq \emptyset\}}$ ensures that the cost is only computed for actual objects, not the padded $\emptyset$ slots. The cost encourages pairing predictions that are confident about the correct class (term $-\hat{p}_{\sigma(i)}(c_i)$) and have a small box localization error (term $\mathcal{L}_{\text{box}}$).

c) 3. Bounding Box Loss ($\mathcal{L}_{box}$): The box loss, used in both the matching cost and the final loss, is a linear combination of the L1 loss and the Generalized Intersection over Union (GIoU) loss.

$$\mathcal{L}_{\text{box}}(b, \hat{b}) = \lambda_{\text{iou}} \mathcal{L}_{\text{GIoU}}(b, \hat{b}) + \lambda_{\text{L1}} \|b - \hat{b}\|_1 \tag{3}$$

Here, λ_{iou} and λ_{L1} are hyperparameters that balance the two loss terms. The L1 loss provides scale-sensitivity, while the GIoU loss addresses the weaknesses of standard IoU.

d) 4. Generalized IoU Loss ($\mathcal{L}_{GIoU}$): The GIoU loss is used to provide a meaningful gradient even when boxes do not overlap. It is defined as:

$$\mathcal{L}_{\text{GIoU}}(b, \hat{b}) = 1 - \left(\text{IoU}(b, \hat{b}) - \frac{|C \setminus (b \cup \hat{b})|}{|C|} \right) \tag{4}$$

where C is the smallest convex shape enclosing both the predicted box $\hat{b}$ and the ground truth box b. Unlike IoU, which is 0 for non-overlapping boxes, GIoU provides a non-zero loss that pushes the boxes towards each other.

e) 5. Hungarian Loss ($\mathcal{L}_{Hungarian}$): After finding the optimal assignment $\hat{\sigma}$ using Eq. (1), the final loss is computed as a sum over all matched pairs. This is the loss that is backpropagated to train the model.

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \Bigg[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbf{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}(i)}) \Bigg] \tag{5}$$

This loss combines a negative log-likelihood for class prediction (similar to cross-entropy) and the $\mathcal{L}_{\text{box}}$ for the bounding boxes. The class loss is computed for all pairs, including the "no object" class, while the box loss is only computed for pairs corresponding to actual objects.

C. Model Training

We utilized a pre-trained DETR model from the Hugging Face 'transformers' library, which comes with a ResNet-50 backbone pre-trained on ImageNet. The final classification head was replaced with a new one configured for our 12 distinct object classes.

To accelerate training and leverage learned features, we employed a fine-tuning strategy. The parameters of the entire ResNet-50 backbone were frozen. This meant that only the transformer encoder, decoder, and the final prediction heads were updated during training, reducing the number of trainable parameters to approximately 18 million (43.5% of the total parameters).

Training was conducted on a single NVIDIA P100 GPU provided by the Kaggle platform. Training DETR is notoriously slow to converge. The original paper reports a 300-epoch baseline schedule requiring ∼6 days on 8 V100 GPUs [1]. In our setup, a single epoch on our dataset took approximately 1.5 hours. Due to platform time limits, we were only able to train the model for 38 epochs, totaling approximately 52 hours. This limited training duration meant the model did not fully converge, which is a critical factor in interpreting the final evaluation results.

III. Evaluation and Results

A. Mean Average Precision (mAP)

The model's overall performance was evaluated using the standard COCO evaluation metrics. The primary metric, mAP@0.50, reached 0.4233, indicating a moderate level of performance. A breakdown by object size reveals a significant weakness in detecting small objects.

TABLE I: Overall Mean Average Precision (mAP) Metrics

Metric	Value
map_50 (IoU=0.50)	**0.4233**
map_small	0.0635
map_medium	0.2595
map_large	0.4052

The low `map_small` value of 0.0635 suggests the model struggles to resolve fine-grained details or that there was a lack of high-quality labels for small objects in the training set. Performance clearly improves with object size, as seen in the `map_large` score of 0.4052.

B. Per-Class Analysis

We further analyzed performance on a per-class basis. Table II shows the mAP for each of the 12 classes, while Table III provides the corresponding precision, recall, and F1-scores. The data from these tables is visualized in Figure 1 and Figure 2 for easier comparison.

The results show a clear hierarchy in performance. "Truck" and "Car" are the best-performing classes, likely due to their large size, distinct features, and high frequency in the dataset. Conversely, fine-grained traffic light states, such as "traffic light-green left", perform very poorly. This can be attributed to

TABLE II: Mean Average Precision (mAP@0.50) per Class

Class ID	Class Name	mAP per Class
11	truck	0.3930
2	car	0.3587
1	biker	0.3105
4	traffic light	0.2792
7	traffic light-red	0.2655
3	pedestrian	0.2142
12	arret	0.1914
9	traffic light-yellow	0.1605
10	traffic light-yellow left	0.1448
8	traffic light-red left	0.1403
5	traffic light-green	0.0776
6	traffic light-green left	0.0426

TABLE III: Precision, Recall, and F1-Score per Class

Class Name	Precision	Recall	F1 Score
truck	0.5256	0.7303	0.6113
car	0.4695	0.7459	0.5763
traffic light	0.4009	0.6835	0.5054
biker	0.4188	0.6490	0.5091
traffic light-red	0.4009	0.6487	0.4955
pedestrian	0.3379	0.6301	0.4399
arret	0.3352	0.6000	0.4301
traffic light-yellow	0.2717	0.4694	0.3442
traffic light-red left	0.2567	0.4253	0.3201
traffic light-green	0.1880	0.3758	0.2506
traffic light-yellow left	0.1826	0.3643	0.2433
traffic light-green left	0.0757	0.1812	0.1067
Micro-Average	**0.4038**	**0.6730**	**0.5048**

a combination of factors: small object size, high visual similarity between classes, and potentially fewer training examples.

A consistent trend across many classes is that recall is significantly higher than precision. The micro-averaged recall of 0.6730 versus a precision of 0.4038 indicates a global tendency for the model to produce more false positives than false negatives. It is effective at finding objects but often misclassifies them or detects non-existent objects.

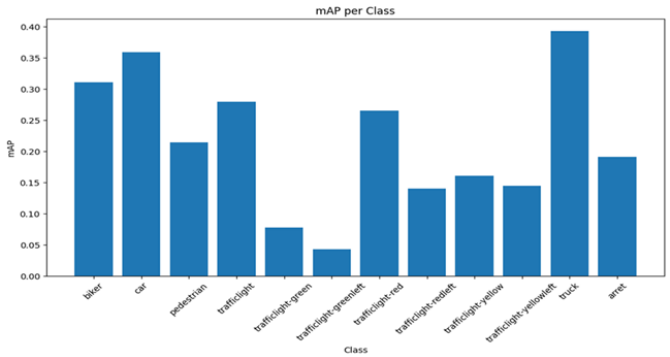

Fig. 1: Visual comparison of Mean Average Precision (mAP@0.50) across all 12 classes. This chart visualizes the data presented in Table II, highlighting the superior performance on classes like "Truck" and "Car" and the significant challenge in detecting specific traffic light states.

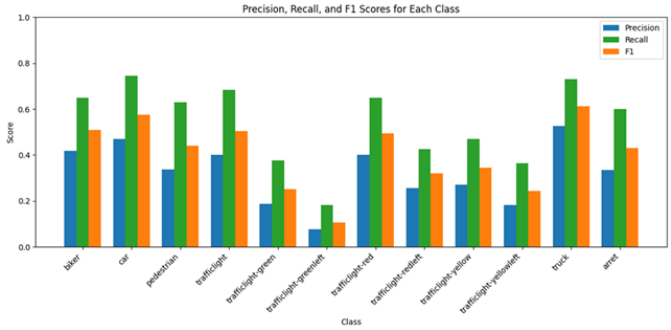

Fig. 2: Per-class Precision, Recall, and F1-scores. This visualization corresponds to the metrics in Table III. The generally higher recall compared to precision is evident across most classes, indicating a tendency for more false positives.

C. Confusion Matrix

The confusion matrix (Fig. 3) provides a visual representation of class-wise errors. The diagonal represents correct classifications, while off-diagonal cells show misclassifications. We observe confusion between different states of traffic lights, which aligns with the poor per-class metrics for these categories.

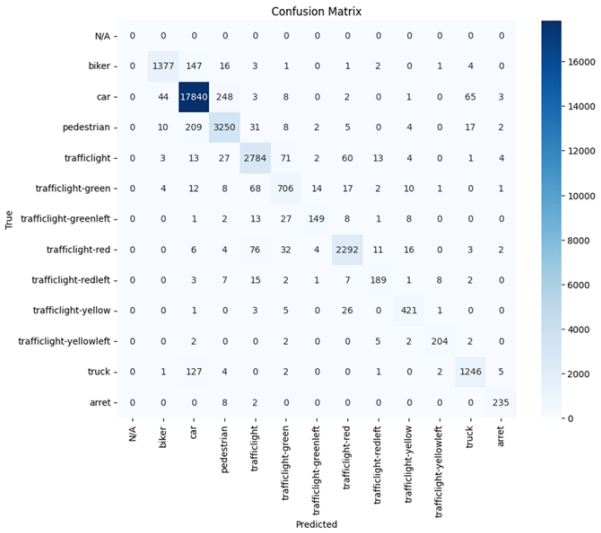

Fig. 3: Confusion matrix visualizing class-wise prediction accuracy. Values on the diagonal represent true positives, while off-diagonal cells indicate misclassifications.

D. Localization Accuracy

The Intersection over Union (IoU) distribution histogram for true positive predictions is shown in Fig. 4. The histogram is strongly skewed to the right, with a majority of true positives achieving an IoU between 0.7 and 0.9. This is a positive indicator, suggesting that when the model correctly detects and classifies an object, its bounding box localization is generally accurate.

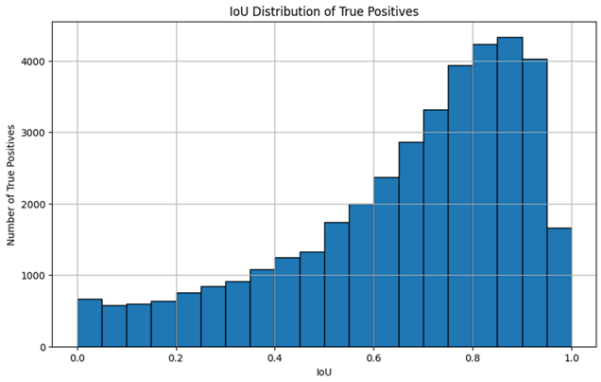

Fig. 4: IoU distribution histogram for true positive predictions. The high concentration of IoUs ¿ 0.7 indicates accurate bounding box localization for correctly classified objects.

IV. Qualitative Results

Qualitative analysis of predictions on test images (Fig. 5) corroborates the quantitative findings. The model reliably detects large objects like cars and trucks with well-fitted bounding boxes. However, smaller and more distant objects, such as pedestrians or traffic lights, are often missed or incorrectly classified, highlighting the challenges identified in our metric-based evaluation.

Fig. 5: Example predictions on test set images. The model demonstrates proficiency in detecting prominent objects like cars but struggles with smaller, distant, or fine-grained objects.

V. Conclusion

In this project, we successfully implemented and fine-tuned a DETR model for object detection on a custom dataset. Despite significant limitations in computational resources and training time, which prevented the model from fully converging, we achieved a moderate overall mAP@0.50 of 0.4233.

Our evaluation demonstrated that the model performs well on large, distinct classes like ”Car” and ”Truck”, and provides accurate bounding box localization for its correct predictions. However, its performance degrades substantially for small objects and for classes requiring fine-grained distinction, such as different traffic light states. The model’s tendency for high recall at the cost of lower precision suggests that with further training, it could learn to better discriminate between true and false positives.

Future work should prioritize training for the full 300-epoch schedule to allow for proper convergence. Additionally, techniques like data augmentation specifically targeting small objects, or employing a more advanced backbone like Swin Transformer, could help address the identified weaknesses and unlock the full potential of the end-to-end DETR framework.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” *CoRR*, vol. abs/2005.12872, 2020. [Online]. Available: https://arxiv.org/abs/2005.12872