PA026

Artificial intelligence project

Object detection and tracking

Hynek Pavlacký 514592

May 2025

1 Introduction

Object detection is a computer vision task in which we are trying to find bounding boxes surrounding objects of interest and also classify them. Tracking is a related problem in which we are trying to follow individual object instances over time (in sequences of images). In other words, the goal is to assign a unique label to each detected object and keep the same label for the same real-world object in subsequent time frames.

Both detection and tracking are important problems and there are many proposed solutions attempting to solve them, as well as many different datasets used for benchmarking these solutions.

The goal of this project was to implement and quantitatively evaluate a pipeline for tracking objects in image sequences of traffic.

2 Dataset

The dataset used in this project was the KITTI MOTS dataset[1]. It contains 21 sequences of images of traffic with two annotated classes of objects — cars and pedestrians. Annotations in this dataset contain segmentation masks for each object instance so it is suitable for the task of instance segmentation as well, not just detection and tracking. However, the focus of this project was just on the detection and tracking.

3 Tracking pipeline

The tracking was performed in two steps. In the first step, objects were detected in each frame separately using a neural network. In the second step, the DeepSORT[2] algorithm was used to connect detections belonging to the same real-world object across subsequent image frames.

3.1 Detection

Detections were predicted in individual frames using deep neural networks. Two models were trained in this project, the Mask-RCNN[3] and a transformer-based detection model (DETR)[4].

3.1.1 Mask-RCNN

The Mask-RCNN is a two-stage object detector. In the first stage it computes image features by applying a deep convolutional neural network (backbone) to the input image[3].

Afterwards, another convolutional neural network called the Region Proposal Network (RPN) takes these computed features together with a fixed number of predefined bounding boxes called anchor boxes and predicts a relative shift, scaling factor and a confidence score for each anchor box. High confidence boxes are kept as the proposed regions[3].

In the second stage, image features inside each proposed region are individually transformed to the same size by a pooling method called ROI Align, which uses sampling and interpolation to get a fixed-size output that accurately represents the features inside the proposed boxes[3].

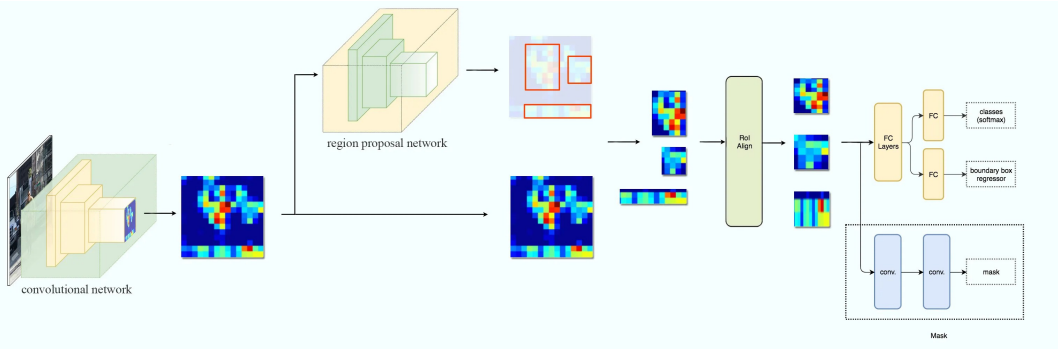

Figure 1: Mask-RCNN model, image taken from [5]

These fixed-size features one by one become input to fully-connected layers which predict the class of the object and a transformation of the proposed box into a final bounding box. A segmentation mask for the detected object can also be easily predicted by applying fully convolutional layers on these features[3].

Figure 1 shows the detection process with Mask-RCNN.

3.1.2 Detection transformer – DETR

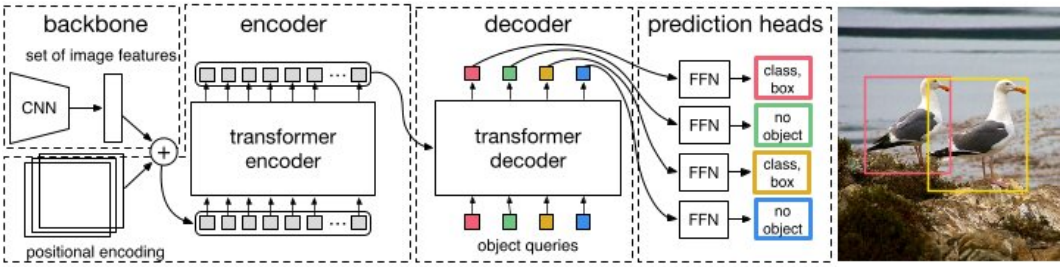

Figure 2: DETR model, image taken from [4]

The DETR model also uses a convolutional backbone to compute image features. Then the features are transformed into a sequence of embedding vectors (values in different channels form the embedding vectors for individual pixels) and positional encoding is used to keep information about the pixel's coordinates. These vectors go through a standard transformer encoder network[4].

The decoder then takes as input the output of the encoder and a fixed number of object queries, which are learnable vectors that serve a similar purpose as anchor boxes in Mask-RCNN. Its outputs are transformed embeddings for each query. Finally, these embeddings are sent to a standard neural network with fully connected layers, which predicts an object class and a bounding box.[4]

Figure 2 illustrates how DETR works.

3.2 Tracking

Tracking is done by connecting bounding box detections in subsequent frames using the DeepSORT algorithm. DeepSORT maintains information about the current object tracks and when bounding boxes from the next image frame are computed, it tries to extend the tracks by finding an optimal assignment between the tracks and the new detections. Assigning a track to a box has a particular cost and the optimal assignment is the one minimizing the overall cost[2].

The cost function used in DeepSORT combines two metrics. The first is the intersection over union (IoU) of the newly detected boxes with the predicted new position of the tracks based on their previous position and velocity. The second metric is cosine similarity of two vectors representing the image features inside the two compared boxes, these feature vectors are computed using a pre-trained neural network[2].

3.3 Training

Training was performed on 13 sequences of the annotated data, 4 additional sequences were used for validation and 4 for testing. Annotations also contained an additional "ignore" class label, which specified image regions without corresponding mask annotations, these image regions were simply masked out with a constant color during both training and evaluation, in order to prevent the networks from producing "false positives" on these unannotated regions.

Mask-RCNN implementation with a ResNet backbone is available in Torchvision[6]. The DETR implementation is from HuggingFace[7]. Pre-trained weights were used for both detection models, the output layers were replaced to match the number of classes in our dataset and then the model was fine-tuned.

Since there are around 4000 images in the training data and many of them are similar because they belong to the same image sequence, overfitting was observed after only a few epochs of training of the pre-trained models. To deal with this problem, heavy augmentation of the training data was used. Images were randomly translated, rotated, scaled, horizontally flipped and their brightness and contrast were randomly changed.

Adam optimizer was used for training and different learning rates were used for the backbones and the rest of the networks. Mask-RCNN was trained for 10 epochs, the DETR model for 20.

4 Examples

The following examples show the difference in outputs (after tracking) between the Mask-RCNN and DETR detection models.

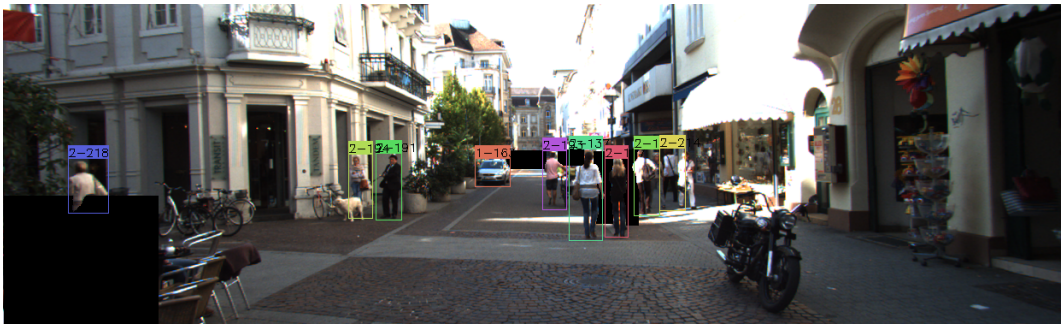

Figure 3: Mask-RCNN output — example 1

Figure 4: Transformer output — example 1

Figure 5: Mask-RCNN output — example 2

Figure 6: Transformer output — example 2

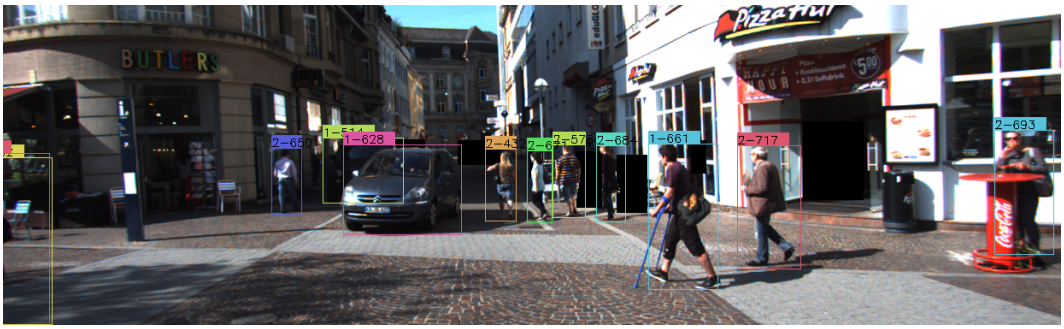

Figure 7: Mask-RCNN output — example 3

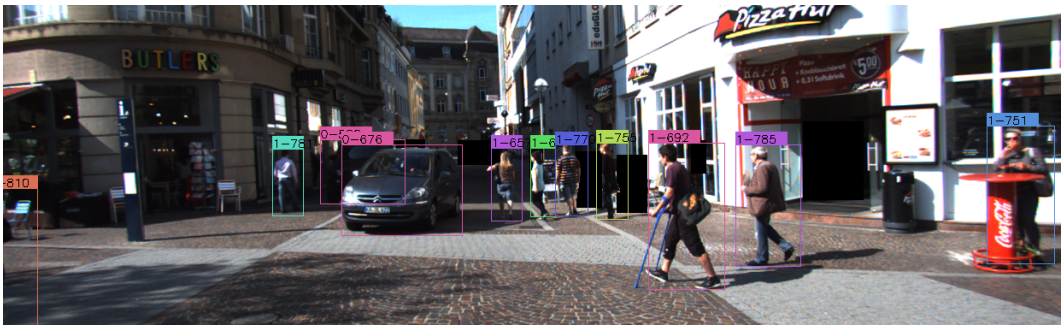

Figure 8: Transformer output — example 3

5 Evaluation

To evaluate quality of the detections (before tracking), variants of the Mean Average Precision metric (mAP) were computed.

First, it is important to describe Average Precision (AP). The AP metric is defined as an area under a Precision-Recall curve. Points on this Precision-Recall curve are computed in the following way: we iterate over predicted bounding boxes sorted from highest to lowest confidence score and at each step we try to find a ground truth box that overlaps with the predicted box the most, i.e., the pair has the highest IoU, this IoU also needs to pass a predefined threshold to be considered a valid match. Then the number of total True positives (matched boxes), False positives (predicted box with no ground truth) and False negatives (undetected ground truth boxes) are updated and the standard Precision and Recall metrics are computed (this creates one point on the curve). After all the predicted boxes are exhausted (all points computed), the area under the constructed Precision-Recall curve can be computed[8].

These are the variants of mAP used in the evaluation:

1. AP (cars) — AP for the car class, averaged over IoU thresholds 0.5, 0.55, ..., 0.9
2. AP (pedestrians) — same as above but for pedestrian class
3. mAP@50 — AP with IoU threshold 0.5, averaged over the two classes
4. mAP@75 — AP with IoU threshold 0.75, averaged over the two classes
5. mAP — AP averaged over thresholds 0.5, 0.55, ..., 0.9 and then averaged over both classes

To evaluate the tracking, the predicted tracks were compared with ground truth tracks and the following metrics were computed: Precision, Recall, MOTA and MOTP[9].

Notation[9]:

- TP_t — number of matches (IoU > threshold) in frame t
- FP_t — number of false positive boxes in frame t
- FN_t — number of unmatched ground truth boxes in frame t
- TP, FP, FN — values accumulated over all frames
- G_t — number of ground truth objects in frame t
- S_t — number of ID switches in frame t (a ground truth box corresponding to a particular real-world object was matched to a box with a different track ID than in the previous frame)

Precision and Recall are defined in the standard way:

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

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

MOTA is defined as[9]:

$$MOTA = 1 - \frac{\sum_t FP_t + FN_t + S_t}{\sum_t G_t}$$

MOTP is defined as the average distance (1 - IoU) of all the matched pairs of bounding boxes (True positives)[9].

Results of the evaluation for both the models can be seen in Table 1.

Metric	**Model**	
	Mask-RCNN	DETR
mAP	0.52	0.44
mAP@50	0.89	0.83
mAP@75	0.57	0.43
AP (cars)	0.63	0.58
AP (pedestrians)	0.42	0.29
Precision	0.73	0.74
Recall	0.78	0.81
MOTA	0.30	0.31
MOTP	0.40	0.40

Table 1: Evaluation

The DETR model struggles more with detecting pedestrians and sometimes even misclassifies them as cars, which contributes to the low mAP score. Overall, cars are easier to detect for both models, as the pedestrians are smaller and often appear in more confusing scenes or far in the background.

As for the tracking metrics, the MOTP metric is 0.4 for both models, which means the average IoU of matched boxes is 0.6. This explains the significant dropoff between mAP@50 and mAP@75. MOTA is a lot lower than both Precision and Recall because of identity switches.

6 Implementation and startup instructions

The project was implemented in Python (3.10.9). The PyTorch[10] library was used to create a custom pipeline to load the data and train the models. The Python scripts used for training and evaluation (`maskrcnn_evaluation.py`, ...) are provided together with the trained model weights. A complete list of all the libraries used is specified in the "requirements.txt" file.

Before running the scripts used for training or evaluation with Python, it is necessary to:

1. Install Python 3.10.9
2. Create and activate a Python virtual environment
3. Install dependencies (pip install -r requirements.txt)
4. Set the project directory as the working directory

References

[1] Paul Voigtlaender et al. "MOTS: Multi-Object Tracking and Segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[2] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *CoRR* abs/1703.07402 (2017). arXiv: 1703.07402. URL: http://arxiv.org/abs/1703.07402.

[3] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[4] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *CoRR* abs/2005.12872 (2020). arXiv: 2005.12872. URL: https://arxiv.org/abs/2005.12872.

[5] Jonathan Hui. *Image segmentation with Mask R-CNN*. 2018. URL: https://jonathan-hui.medium.com/image-segmentation-with-mask-r-cnn-ebe6d793272.

[6] URL: https://docs.pytorch.org/vision/main/models/mask_rcnn.html.

[7] URL: https://huggingface.co/docs/transformers/en/model_doc/detr.

[8] URL: https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/.

[9] Keni Bernardin and Rainer Stiefelhagen. "Evaluating multiple object tracking performance: the CLEAR MOT metrics". In: *J. Image Video Process.* 2008 (Jan. 2008). ISSN: 1687-5176. DOI: 10.1155/2008/246309. URL: https://doi.org/10.1155/2008/246309.

[10] URL: https://pytorch.org/.