Master's Programme in Data Science

End-to-end Object Detection with Transformers

Mikko Kotola and Juuso Lassila

January 12, 2021

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

Contents

1. Introduction

Object detection is one of the main use cases of computer vision. While other use cases, such as image classification or segmentation, are straightforward to implement with neural networks, object detection has remained a relatively complicated and challenging task. Object detection models have relied a lot on different pre- and post-processing techniques in addition to one or more neural networks.

With a new paper introducing the model *Detection Transformer* (DETR) [2], a big leap in object detection research was made. The presented model does not need any additional pre- or post-processing. Instead it is a combination of a neural network and a new loss function that handles the whole task. Developers can use this model in a similar way as classification or segmentation models. They just have to show the model examples of classes and bounding boxes, and the model learns to output the same information. This big leap forward in simplicity, combined with the ease of implementation of DETR motivated us to look into this paper.

In this report we first give some background about object detection and related tasks, and earlier approaches to them. We then describe the DETR model, especially its architecture and the new loss function. We compare DETR to other object detection models, listing both advantages and problems. We also describe some experiments we did with DETR. Lastly, we introduce some follow up work.

2. Background: Object Detection Tasks and Approaches

2.1 Object Detection and Related Tasks

Object detection is the machine learning task of recognising and locating object instances within an image [13]. In this report, object recognition is defined more narrowly as the task of predicting a bounding box and a category label for each object instance of interest in an image [2, 8].

Other machine learning tasks on images related to object detection are [13, 2]: *object classification*, where the model is only required to indicate, if at least one instance of an object category is present in the image, but not required to output its location within the image; *semantic segmentation*, where all pixels are labeled to belong to a category (one of a list of target categories or possibly a "none of the target categories" label); *panoptic segmentation*, which is similar to semantic segmentation, but requires each object instance to be separately identified; and *scene understanding*, where the goal is to formulate natural language statements about what the image contains (e.g. "A young male skateboarder performing a trick"). The transformer-based object detection approach discussed in this report is relevant for all of these related tasks. It is noteworthy, that if a method can offer a per-pixel semantic segmentation of an image, it can always also offer a bounding box segmentation [8].

In addition, an interesting task is *few-shot object detection* [9], where the goal is to be able to detect objects after encountering only a small number of training instances of the object class of interest. A typical approach to this task is training a network on abundant images (containing other classes) and then fine-tuning the network with a few samples from the classes of interest.

2.2 Object Detection Approaches

The main line of division in object detection approaches is between anchor-based and anchor-free detectors. In *anchor-based object detection methods* [11], a large number of preset anchors are first placed on the image. The methods then proceed to predict the category at each anchor and refine the coordinates of these anchors one or several times. Finally, they output the refined anchors and object categories as detection results. An example of an anchor-based detector is the RetinaNet [5]. Some problems with anchor-based detectors include trouble in detecting objects with large shape variations, a large number of tunable hyperparameters related to the anchor boxes, complicated computation with calculating intersection-over-union (IoU) scores, and the fact that most of the densely placed anchor boxes are labelled as negative samples during training, causing an imbalance between negative and positive samples [8].

On the contrary, *anchor-free detectors* [11] directly find objects without preset anchors. One subtype of anchor-free detectors are *keypoint-based methods*, which first locate several pre-defined or self-learned keypoints and then bound the spatial extent of objects. Another subtype are *center-based methods*, which use the center points of objects to define positives and then predict the four distances from the positive center point to the object boundaries. An example of a center-based method is the FCOS [8]. The transformer-based method is a new addition to the family of anchor-free detectors.

Feature pyramid networks (FPN) [4] is a general method, which is used to augment/enhance several models. The basic idea in an FPN is taking advantage of convolutional features learned by the CNN at multiple layers, and building high-level semantic feature maps at all scales. Another method that can be used with different types of detectors is the Cascade Segmentation Module [13]. It uses parallel streams for stuff categories, object categories and object part categories. It can be added on top of several different convolutional base networks such as Segnet [1] and DilatedNet [3].

3. Object Detection with Transformers

Carion et al. [2] present a transformer-based approach to object detection, the *Detection Transformer* (DETR). In DETR, object detection is viewed as a direct set prediction problem. A central part of the approach is usage of a transformer encoder-decoder model and a bipartite matching loss function to force a correct amount of unique object instance predictions for an image. DETR architecture is simple: it does not require any customized layers, but can be build using a standard CNN (e.g. ResNet) and standard transformer classes[†].

3.1 Architecture

The DETR architecture [2] contains three main components: a CNN backbone, an encoder-decoder transformer and a feed forward network.

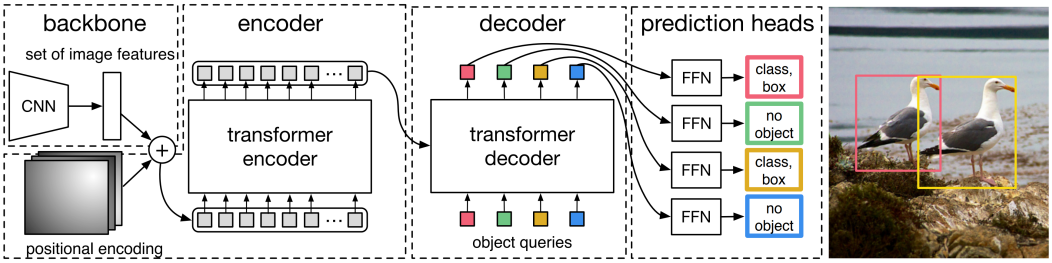

Figure 3.1: DETR architecture. Notice the blue and green "no object" detections related to the fixed number of object queries. Image source: Carion et al. [2].

The purpose of the CNN backbone is to take the image as input and generate a lower-resolution activation map $f \in \mathbb{R}^{C \times H \times W}$. A typical value of C, the number of features, is 2048 and typical values of H and W are $\frac{H_0}{32}, \frac{W_0}{32}$.

The tranformer encoder first recudes the number of channels by using a 1×1 convolution and then collapses the spatial dimensions of the input feature maps into one

[†]DETR implementation published by Carion et al. uses the Detectron2 framework [10] as a library.

dimension. All encoder layers follow a standard architecture: first adding fixed positional encodings to the input, then applying multi-head self-attention and finally a feed forward network.

The transformer decoder takes as input a small fixed number N of learned positional embeddings, *object queries*, and attends to the encoder output. All N object queries can be calculated in parallel. Each output embedding of the decoder is given as input to a shared feed forward network that predicts either an object instance (class and bounding box) or a no object class ($\varnothing$).

The final component is a 3-layer perceptron (FFN) with ReLU activation function, and a final linear layer. The FFN outputs normalized center coordinates, and the height and width of the box. Additionally, the linear layer uses a softmax function to predict the class label.

3.2 Bipartite matching loss

The bipartite matching loss function uniquely assigns each prediction instance to a ground truth object instance. It is invariant to permutations of predicted objects, so predictions of several object instances can computed in parallel [2].

DETR uses a fixed-size set on N predictions in a single pass through the decoder. N is a number significantly larger than a typical number of object instances in an image, for example 100. As N is larger than the number of object instances in the image, the set of predictions $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ is padded with special *no object* class symbols ($\varnothing$). If y is the ground truth set of objects, a bipartite matching between y and $\hat{y}$ can be found by searching for a permutation of N elements with the lowest cost:

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

where $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ is the pair-wise matching cost between ground truth y_i and the prediction with the index $\sigma(i)$ [2]. The optimal assignment of predictions to ground-truth labels can be done efficiently using the Hungarian algorithm. The matching cost considers both class prediction and similarity of boxes: $\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = -\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma}(i)(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$, where $\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$ is the loss between the ground truth and predicted boxes.

Once the bipartite matching is done – all label-box predictions are aligned one-to-one with ground truth label-boxes – the total loss, which is also called the *Hungarian loss* is computed for all the pairs using the linear combination of the negative log-likelihood of the class prediction and the box loss:

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

The bounding box loss, used in both matching loss and Hungarian loss, is defined as a linear combination of the ℓ_1 loss and the generalized IoU loss:

$$\lambda_{\text{iou}}\mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}} \left\| b_i - \hat{b}_{\sigma(i)} \right\|_1$$

where $\lambda_{\text{iou}}, \lambda_{\text{L1}} \in \mathbb{R}$ are hyperparameters.

4. Discussion

DETR brings many improvements compared to previous models that make using the model easier for developers. These improvements are balanced by a somewhat more complex model that also takes more resources to train.

4.1 Pros

The main appeal of DETR is that it can be trained end-to-end. This means that there are no parts that require handcrafted design by humans, nor does the model require multiple steps of different computations apart from the single neural network. Instead, the model just learns to produce classes and bounding boxes based on samples.

A central challenge in object detection is avoiding near-duplicate objects. Many detectors, such as FCOS [8], use postprocessing such as non-maximal suppression to address the near-duplicate issue. DETR, as a direct set prediction approach, is postprocessing-free [2]. This is in a large part thanks to the bipartite matching and related loss function discouraging duplicate detections of the same object in a built-in way, making possible a simpler and, in this part, more efficiently trainable model.

DETR can be implemented using standard modules and methods found in deep learning libraries, which makes using it a lot easier than previous methods.

Extending DETR to the panoptic segmentation task is quite straightforward [2], requiring adding some components in the later part of the original architecture. In line with the bounding box object detection results, DETR excels at detecting stuff classes, which typically cover large areas, while not meeting the benchmark networks for smaller thing classes.

4.2 Cons

Compared to previous methods, DETR requires a long training time. In one reported experiment, the model was trained 500 epochs on 8 GPUs [14]. The training took 10 days. For example Faster RCNN [6] took at worst 10 times less time to train.

DETR also performs worse on small objects than previous benchmark methods [2]. This is due to the low resolution features calculated by the CNN. The authors of the model report using FPN-like approach to overcome the deficiency, but report that increases in quality of predictions are heavily offset by a much increased consumption of resources during training.

5. Experiments

We implemented the DETR model using Pytorch[†]. Our model was built to be a somewhat lighter version of the original DETR implementation[‡]. For example, We did not use the ResNet CNN for feature extraction, but instead replaced it with a small CNN. We also decreased the amount of layers in the transformers.

Implementing the model was quite an easy thing to do using Pytorch, which validated one of the presented advantages of the model. Especially the encoder-decoder transformers are implemented as a single model. In addition to the transformers, we only had to implement the CNN, positional encodings and fully connected layers, that map transformer outputs to classes and boxes for the model.

The most challenging part of coding the model was defining the loss, especially because it's formed from multiple different components, such as the combination of L1 loss and the GIoU loss of bounding boxes. The bipartite matching was implemented with a scipy function, that uses a different algorithm than the original paper, but the results are equally correct. Calculating the loss for all pairs of the queries for the matching while supporting batches was challenging.

However, our model was still quite large, and training the model took longer than we hoped for. On CSC's Puhti, training for one epoch of the COCO training dataset (118 000 images, 18 GB) using 1 GPU took 5 hours of processing time. Therefore, we could not develop a fully trained model within the given project time. Training a model from scratch (including the backbone CNN) would have required a much more advanced parallelization of training and notable use of computing resources, perhaps not appropriate for this kind of project. Due to the partial training, we could not do any of the real experiments, that we originally aimed for, on the trained model.

Our only experimental result is that training the DETR model, and also object detection in general on the COCO dataset, is very resource-intensive, just like it was stated in the paper. Our measured validation loss before the first training epoch was 0.4028, while after the first epoch, it was down to 0.3606. Typically, the loss decreases

[†]Source code of our implementation is available at https://github.com/mikkokotola/transformer-object-detection

[‡]See source code by authors of the original paper at https://github.com/facebookresearch/detr.

a lot more during the first epoch compared to the following ones, so when the first loss decreased quite a small amount, it might indicate that the loss will decrease slowly over epochs, just as is stated in the paper.

6. Follow up work

Even though DETR was published in spring 2020, it has already been followed up by multiple papers that try to solve the shortcomings, such as the long training time and issues with small objects.

Deformable DETR [14] comes up with an alternative attention mechanism called Multi-scale Deformable Attention Module. It smartly limits the keys, that get attended to in the transforms, in a way that makes the computation faster. It also uses feature maps of multiple scales, making it possible for the model to detect both big and small objects.

Methods where transformers and the bipartite losses are combined with the old methods such as FCOS or RCNN are also introduced [7]. These methods also improve on the bipartite loss by allowing a match only when the middle point of a label box is inside a predicted bounding box. This reduces the randomness of the bipartite loss and allows a faster convergence. The attention mechanisms is also improved for faster training and inference time.

Finally, one paper introduces a way to adaptively cluster the queries in the transformer [12]. This also decreases the inference time, as the model only has to attend to some human specified amount of clusters, instead of all items.

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv*, abs/1511.00561, 2016.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. *arXiv*, abs/2005.12872, 2020.

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv*, abs/1606.00915, 2017.

[4] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[6] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[7] Z. Sun, S. Cao, Y. Yang, and K. Kitani. Rethinking transformer-based set prediction for object detection. *arXiv preprint arXiv:2011.10881*, 2020.

[8] Z. Tian, C. Shen, H. Chen, and T. He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[9] X. Wang, T. E. Huang, T. Darrell, J. E. Gonzalez, and F. Yu. Frustratingly simple few-shot object detection.

[10] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[11] S. Zhang, C. Chi, Y. Yao, Z. Lei, and S. Z. Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. *arXiv*, abs/1912.02424, 2020.

[12] M. Zheng, P. Gao, X. Wang, H. Li, and H. Dong. End-to-end object detection with adaptive clustering transformer. *arXiv preprint arXiv:2011.09315*, 2020.

[13] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[14] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.