Transformer Architecture in Object Detection

Jiayang Zheng

Abstract

Transformer, as introduced by Vaswani *et al.* [9], has proven itself to be one of the most successful deep learning models used in natural language processing, particularly the field of neural machine translation. In this project, I explored the usage and capability of transformer in object detection given a relatively scarce amount of training data. In the duration of this project, I have developed an end-to-end object detector pipeline consisting of a visual backbone, a transformer, and final regression/classification head using a linear layer or multi-layer feed-forward network. During the course of implementation, I have implemented two loss functions for the object detector pipeline to compare and contrast the final performances: a set prediction loss, and the original YOLO loss. The resulting pipeline achieved about 0.35 mAP on PASCAL VOC 2007. The following images is an example output from the detection pipeline.

Figure 1: example output from detection pipeline

1 Introduction

Object detection has been an active research area for almost as long as Computer Vision exists. It is an important vision task that tackles recognition and localization of various object in given input images. Different from image classification task, object detection requires a more comprehensive and structured understanding of the image in question. Thus, it is a much demanding and resource-exhausting problem. During the "dark ages" before deep learning, early object detection algorithms, such as Viola Jones Detectors, HOG Detectors, and Deformable Part-based Models, utilize handcrafted and vastly sophisticated feature representations [10]. A lot of algorithms during this era use fixed filter to pick out certain features and utilize them to make a prediction. Ever since the introduction of deep learning, object detection task nowadays relies more on convolution neural networks, which can robustly and autonomously learn stable high-level feature representations of images. Modern object detectors treat object detection task as more of an indirect set prediction problem where the essential goal of the task is to make a set of unique prediction given labels and bounding boxes. Detectors, such as one-staged YOLO [5], SSD [4] and two-staged Faster-RCNN [6], define surrogate regression and classification problem on a set of region proposals, anchor boxes, and window patches of the image; meanwhile, these "deep" object detectors rely heavily on post-processing to eliminate duplicate predication of labels and bounding box pairs [1]. Such post-processing techniques includes non-maximum suppression, anchor box set and heuristic functions that assign a target box to an anchor box. In fact, these techniques can be argued to contribute largely to the current state-of-the-art performance achieved.

Originally proposed by Vaswani *et al.* [9], the Transformer is a revolutionary and unconventional deep learning model in the field of neural machine translation. Instead of possessing a either recurrent or convolutional structure, the Transformer solely bases on attention mechanism to store information and learn representations of its input. Comparing to its predecessors, the Transformer can take a whole sequence as input instead of fragments of them. Then the attention mechanism can collect information from the input sequence to better aid the representation learning process. The Transformer introduces self-attention mechanism to perform an information gathering. This process essentially can be seen as learning pairwise interaction between any two tokens, making them especially better at memorizing long sequences than conventional recurrent models [1]. With current dominance of Transformers in the field of natural language processing, some researchers found that the efficiency and scalability of the Transformer could benefit the field of Computer Vision: Dosovitskiy *et al.* [3] uses the encoder part of the Transformer to do image classifications and was able to obtain a performance on ImageNet that is just a few percentage points below ResNets of relative parameters; Carion *et al.* [1] used the whole Transformer to tackle object detection and achieved a higher performance on COCO dataset than Faster-RCNN [6]. This project will mainly based on the work of Carion *et al.* in [1]. However unlike the work done in [1], where they only implement a model using direct set loss, I developed two models using direct set loss and the loss proposed in YOLO [5], as the direct set loss does not seem to perform well in a relatively small dataset such as PASCAL VOC 2007. In order to utilize the Transformer in object detection, some certain changes need to be made. Chief amongst those changes is the auto-regressive nature of the Transformer. Although largely different from its predecessors, the Transformer still shares the usage as an auto-regressive model: taking previous predicted tokens into consideration and generates token output in a sequential manner. In this auto-regressive process, the decoder part of the Transformer would take as input either the start symbol or previously predicted token at the same time masking out part of the input that has not been predicted yet. This process is highly useful in machine translations since the output can be of various length, however, in the application of object detection, sequentially predicting tokens would not make much sense for, most

likely, previously predicted bounding boxes may not have an affect on later predictions. As a result, parallel decoding, which is basically the approach taken by parallel sequence prediction in word representation learning [2], is used to support a direct prediction from the Transformer. In this case, the decoding part can thus be paralleled and gives a faster inference time than predicting sequentially.

2 Method

In this project, I developed two object detection pipelines with mostly similar architectures. These pipelines are largely based on the work done by Carion *et al.* [1], which consists of convolutional backbone model, mildly-modified transformer model, and finally a regression/classification head to make sense of the feature representation outputted by the transformer.

2.1 Detector Architecture

The overall architecture of the detector is fairly simple as described. There are three essential components that made up the architecture: a CNN based backbone to reduce resolution of input images and extract feature representations, an encoder-decoder transformer to learn feature representation in feature maps, and eventually a linear layer or feed-forward network to classify or regress feature representations into labels and bounding boxes. The YOLO loss version of the pipeline will output for each grid a combined vector of multi-label prediction and bounding box regression.

Backbone Normal ResNet50 model, excluding the max pooling layer and linear layer, takes as input an image $x \in \mathbb{R}^{3\times H_0\times W_0}$ and return as output a feature map $z \in \mathbb{R}^{C\times H\times W}$ where C is the number of output channels from the last layer of Bottleneck in ResNet50, H and W are the height and width of the feature map. In the particular example of ResNet50, $C = 2048$ and $H, W = \frac{H_0}{32}, \frac{W_0}{32}$.

Transformer Encoder Given that the output feature map from backbone is an image $z \in \mathbb{R}^{C\times H\times W}$, the first thing to do is to project the feature map from $z \in \mathbb{R}^{C\times H\times W}$ to $h_0 \in \mathbb{R}^{d\times H\times W}$ given d is the hidden dimension of the transformer input. Then the spatial dimensions of h is collapsed into one dimension to obtain a sequence like vector $h_1 \in \mathbb{R}^{d\times HW}$. Since the Transformer encoder expects an input with first dimension being sequence length and second dimension being hidden dimension, h_1 will be permuted to $h_2 \in \mathbb{R}^{HW\times d}$. Meanwhile, given the permutation invariant property of self-attention, a 2D positional encoding is also generated and later added to the input h_2 to supply some kind of ordering information. The Transformer encoder I used in the project is mildly different from the one Pytorch supplied. Namely, positional encoding is not added to h_2 before passing to the encoder but rather added when calculating query and key. Besides, the extra layer normalization at the end of each layer of encoder is removed.

Transformer Decoder The decoder part of the Transformer remains the standard structure seen in Vaswani *et al.* [9]. Each layer of decoder expects as input N embeddings of size d and outputs N embeddings of size d vector. Unlike the original Transformer decoder, the Transformer decoder I implemented in the project takes all the N embeddings in one time step and decode them simultaneously instead of attending to each input embedding in an auto-regressive style. At the very first layer of decoder, a query embedding is fed to the self-attention. Unlike the positional encoding in the encoder, this query embedding is a learnable parameter that eventually gains

a rough understanding of the whereabouts of the N final outputs. In a way, it resembles the Region Proposal Network in Faster-RCNN [6], only that it is much more naive and thus efficient during inference. Like the encoder, query embeddings and positional encodings are only applied when calculating queries and keys in the self-attention module. The N query embeddings will be transformed by the decoder into N object proposals that represent decodings of various feature map patches encoded by the encoder. In the model that uses YOLO loss, the number of query embeddings is equal to S^2, where S is the number of grid to have in each axial direction.

Classification/Regression Head Acting as the final assembler of the object detection pipeline, classification head and regression head would act as a final interpreter of decoded vectors given by the transformer. In the model that uses set-based loss, the classification head and regression head are two different models that perform different interpretation tasks: the classification head is a linear layer that projects the decoding with hidden dimension to a vector with a dimension of the number of classes plus 1 for no object prediction; while the regression head is a 3-layer feed-forward network that projects the decoding with hidden dimension to a vector with dimension of 4 corresponding to the normalized center coordinate, width and height of the bounding boxes. In the model that uses YOLO loss, the classification head and regression head are combined into a single linear layer that transform the decoding into a vector with dimension of $5B + N_{class}$, where B is the number of bounding box to be predicted each grid and N_{class} is the number of classes. Since the Transformer would eventually given multiple number of object proposals, the classification/regression head will be shared among all these object proposals in both models.

2.2 Loss Functions

As mentioned in the previous sub section, two detection pipelines have been implemented. Both pipelines share roughly the same architecture with some small modifications. However, they are drastically different in terms of training process and final output. As a result, two loss functions has to be defined for each pipeline.

2.2.1 Set-based Losses

Object detection can be formulated as a direct set prediction problem, where the ulterior goal is to predict a set of unique label and bounding box pairs. The challenge is that no currently known deep learning model can make a direct set prediction. The approach taken by most of the modern object detector is to post-process output from the model to selectively include predictions that the model is fairly confident of. With such approach, non-maximum suppression often has to be performed to get a good performance. However, if direct set prediction can be achieved, post-processing would not be necessary, hereby saving a lot of resources and time. With this goal in mind, I adopt a set-based loss function consisting of several different loss introduced in [1]. Since the detection pipeline makes a fixed number of N object proposals, which will always be larger than the number of objects in the image, some kind of matching is needed to match the most appropriate prediction to the corresponding target labels and bounding boxes. In other word, an optimal permutation between the prediction and target needs to be found in order for the pipeline to train end-to-end. As introduced by Carion *et al.* [1], a bipartite matching between the prediction and target is generated by using the Hungarian algorithm. The Hungarian Algorithm looks at the class prediction probability and the similarity between the predicted boxes and the target boxes, and utilize these information to yield an optimal bipartite matching between prediction and target [1].

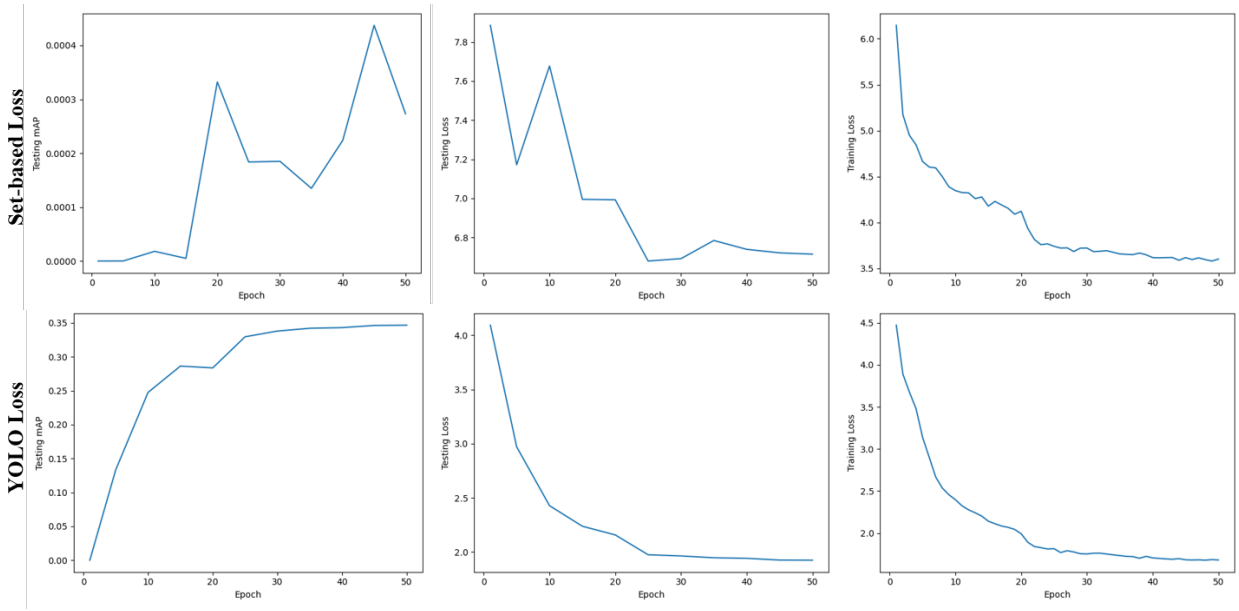

Figure 2: mean AP, Testing Loss, and Training Loss for both models. The top row is for set-based loss model while the bottom row is for YOLO loss model.

Label Loss After obtained the optimal bipartite matching, a label loss can then be calculated using cross entropy. In more detail, the matched predictions will calculate a cross entropy loss against the matched target label, while predictions that are not matched will calculate a cross entropy loss against the no object class label or "background" label. The ultimate goal is to have the pipeline predict both the correct labels and the exact number of object in the image. Since no object class label will be much more abundant than class labels, a weight is imposed on the loss of no object class label.

Bounding Box Loss This part of the loss is constructed by two losses: bounding box regression loss and generalized IoU loss. The bounding box regression loss calculates L_1 loss between the matched bounding box and its target bounding box. The bounding boxes that are not matched will not be counted into the loss thus not contributing to the training of the model. Similar to bounding box regression loss, the generalized IOU loss calculates the $1-GIoU$ between the matched boxes and its target. As introduced by Rezatofighi *et al.* [7], generalized IOU is of the form: Given A and B to be the prediction and target bounding boxes, C to be the smallest convex hull that encloses both A and B,

$$GIoU = \frac{|A \cap B|}{|A \cup B|} - \frac{|C \setminus (A \cup B)|}{|C|} = IoU - \frac{|C \setminus (A \cup B)|}{|C|}$$

As a result it is easy to see that to minimize $1 - GIoU$ is to maximize IoU and minimize $\frac{|C \setminus (A \cup B)|}{|C|}$.

The final set-based loss will be a weighted combination of the label loss, the bounding box regression loss and the generalized IoU loss.

2.2.2 YOLO Losses

Besides from a novel set-based loss, I have also implemented a more contemporary and widely used loss proposed in the original YOLO paper [5] that I refer to as YOLO loss. Comparing to

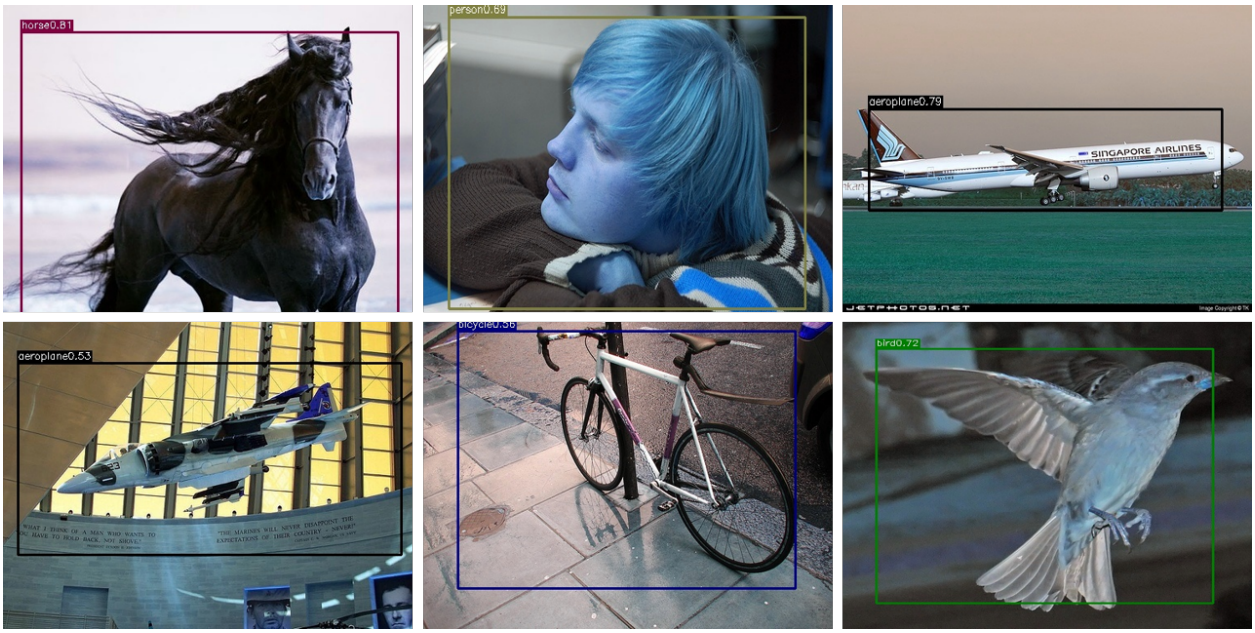

Figure 3: Example outputs from YOLO loss based model.

set-based loss, YOLO loss is much more straight forward. There is no cross entropy loss in any of the consisting part of the loss. Instead, YOLO loss utilizes sum-squared error for easy optimization. Although the YOLO loss does essentially weight localization error the same as classification error, it compensates that by giving the bounding box regression loss more weight in the final composition of the total loss. Just like set-based loss, YOLO loss also suffers from the abundance of "background" class. Namely, for each target, there are way more background labels than class labels. As a result, a weight for no object loss is also introduced in order to weigh down the no object loss. Thus, the YOLO loss can be formulated as follows:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{1}$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{y_i} - \sqrt{\hat{y}_i})^2] \tag{2}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{3}$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{4}$$

where $\mathbb{1}_i^{obj}$ describes if object appears in grid i and $\mathbb{1}_{ij}^{obj}$ denotes if the jth bounding box in grid i is responsible for generating the prediction. In this formula, $\lambda coord$ is the weight to upscale bounding box regression loss while λ_{noobj} means to weigh down no object loss.

3 Experiment Result

In this project, I have trained two models on PASCAL VOC 2007 dataset, which I will abbreviate to PASCAL dataset in the following content. The PASCAL dataset contains 20 classes. There are 5011 images in the training dataset and 4950 images in the validation dataset. During experiments, the images will undergo random flip, random scaling, random image shifting, and random cropping. The images are then normalized with ImageNet normalization mean and variance and are finally resized into a shape of 448×448. Both models are trained on a device with i9-10900KF CPU and NVIDIA 2080Ti GPU for 50 epochs with a training batch size of 24. Both models have a training process of about 3 hours. In terms of number of parameters, both models have around 40 million parameters with the set-based loss model reaching 46 million and YOLO loss model reaching 41 million.

Class Name	**Average Precision**
Aeroplane	0.0
Bicycle	0.0031
Bird	0.0001
Boat	0.0001
Bottle	0.0001
Bus	0.0
Car	0.0
Cat	0.0
Chair	0.0013
Cow	0.0001
Dining Table	0.0
Dog	0.0003
Horse	0.0
Motorbike	0.0
Person	0.0
Potted Plant	0.0001
Sheep	0.0
Sofa	0.0
Train	0.0
TV Monitor	0.0003
Mean Average Precision	0.0003

Table 1: Class average precision and mean average precision at epoch 50 for set-based loss model

3.1 Set-Based Loss Model

As introduced previously, the model using set-based loss consists of a CNN backbone, an encoder-decoder transformer, and a linear layer as classification head and a multi-layer feed-forward network as regression head.

Optimizer In this project, I decided on AdamW as the optimizer since there seems to be some instability when using SGD. Adam optimizer can also be used here as it gives pretty much the same convergence rate and final performance. A weight decay of 10^{-4} is applied on the weights to keep the model from overfitting.

Learning Rate A learning rate of 10^{-4} is used for training the transformer and linear layer while a learning rate of 10^{-5} is used for training the backbone. This learning rate scheme is necessary for stable training at the first few epochs during the training process. In general, backbone learning rate needs to be a magnitude smaller than the rest of the model, otherwise, training loss will oscillate and unable to converge. At training epoch 20, and epoch 40, I further reduce the learning rate by a factor of 10.

Transformer I used the standard setup of transformer as mentioned in [9], with 6 encoder and decoder layers, self-attention with 8 attention heads, and a dimension of 2048 for the feed-forward layer after the attention. The transformer accepts input with hidden dimension of 256. I have experimented with different hidden dimension by increasing to different numbers and quickly found out that to increase hidden dimension, I have to drop batch size, which would cause training to go slowly.

Loss Weight During the calculation of set-based loss, a weighted combination of label loss, boudning box regression loss, and generalized IoU loss was calculated given the weights $\lambda_{label} = 1, \lambda_{boundingbox} = 2, \lambda_{GIoU} = 5$.

Training loss, testing loss, and testing mAP can be seen in Figure 2. A detailed testing mAP can be seen in 1. It is clear that while the model is learning during training process, the testing loss oscillate and stayed at a fairly high loss. The gap between testing loss and training loss at epoch 50 is about 3.113, which is a sign that the model is overfitting. However, the training mAP at epoch 50 is about 0.28, which suggest that the model is not doing a very good job at object detection. The following could provide some explanation as to why set-based loss model is performing poorly.

Insufficient Data and Difficulty of the Task Carion *et al.* [1] trained their transformer detector on COCO dataset, which consists of 330,000 images and over 200,000 are labeled images. This large amount of data enabled the model to be sufficiently trained. At the mean time, PASCAL dataset has only about 10,000 images, almost half of which are validation images. Besides, direct set prediction is an extremely hard task to perform since it not only requires correct class labels but also requires correct amount of class labels in each images. Given the difficulty of the task and insufficient amount of data to support the optimization of such task, it is not very surprising that the model performed poorly.

Bipartite Matching The optimal permutation that assign prediction to target is essential for the set-based loss to be properly attributed to corresponding parameters. However, the bipartite matching performed in the matcher are very unstable in itself due to random initialization and noisy choosing conditions during the training [8]. This unstability could hinder the training process and eventually causing the model to become unstable as well.

3.2 YOLO Loss Based Model

Similar to Set-Based loss model, the YOLO loss based model consists of a CNN backbone, an encoder-decoder transformer, and a single linear layer as classification/regression head. The training parameters are mostly the same as set-based loss model. Instead of bounding box regression loss weight and generalized IoU loss weight, these weights are replaced by coordinate loss weights $\lambda_{coord} = 5$ and no object loss weight $\lambda_{noobj} = 0.5$ as suggested in [5]. Also, the number of grid at each axial direction S is set at $S = 5$ and the number of bounding boxes predicted each grid $B = 2$.

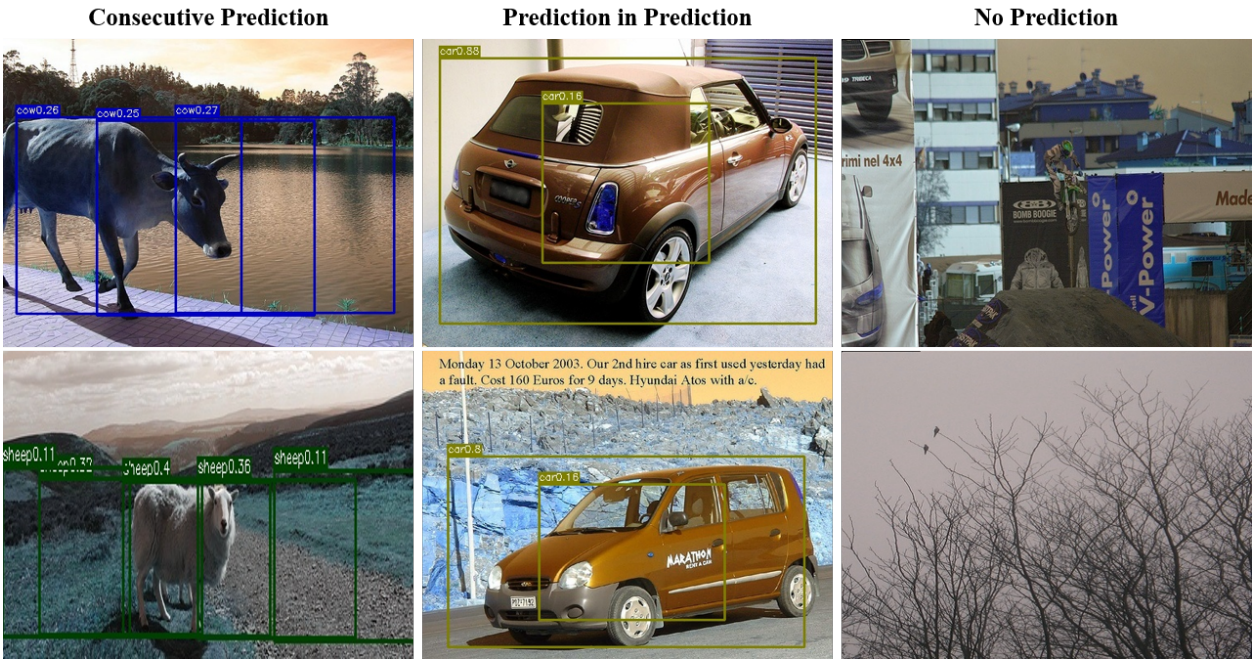

Figure 4: Failure examples from YOLO loss based model.

After 50 epochs of training, YOLO loss based model achieved a mAP score of 0.3466 as shown in table 2. Some sample outputs from the network are shown in 3. Although these outputs seems to be rather good, there are some cases in which the model does not perform very well. I have included in 4 three kinds of failure cases most common in the final outputs and named them: **Consecutive Prediction**, **Prediction in Prediction**, and **No Prediction**. In fact, these three kinds of failure cases can basically summarize all the issue in the YOLO loss based model. I postulate that the problem lies within the learnable parameter query embedding. As mentioned earlier, query embedding is a learned parameter to be passed into decoder as a hypothesized proposal of where the decoder should be attending to given the encoded feature representations. I suspect that different images activate different query embedding combinations. Thus, two similar images will activate similar query embedding combinations, results in a very patterned emerge of failure cases. Besides, since this model makes prediction in a sliding-window style, it is having a hard time detecting small objects. The two examples in no prediction category have relatively small object size comparing to most images in the dataset, hereby leading to the model thinking that there is no object in the images.

4 Discussion

Judging from the performance of both models, it is quite clear that comparing to conventional CNN-based approaches, the transformer approach seems to be lacking some fundamental capability to reason confidently about object detection with relatively small dataset. It seems like the transformer's lacking of translation equivariance and locality, that are inherent to CNN, is causing it to not generalize very well when insufficient amount of data is used for training [3]. Apart from the inherent problem of the transformer itself, training loss is also a big contributing factor to performance. In the set-based loss, due to the randomness of the bipartite matching, the training steps are not very stable and starting to show signs of convergence at a still very high loss. Comparing

Class Name	Average Precision
Aeroplane	0.5092
Bicycle	0.4151
Bird	0.3237
Boat	0.2145
Bottle	0.0646
Bus	0.4831
Car	0.4177
Cat	0.5951
Chair	0.0788
Cow	0.2547
Dining Table	0.4405
Dog	0.5184
Horse	0.5621
Motorbike	0.3986
Person	0.2616
Potted Plant	0.0779
Sheep	0.2286
Sofa	0.3502
Train	0.6142
TV Monitor	0.1241
Mean Average Precision	0.3466

Table 2: Class average precision and mean average precision at epoch 50

to the COCO dataset Carion *et al.* [1] trained their model on, which has over 200k labeled images, PASCAL is a very small dataset with only 5000 images for training. This insufficient amount of data combined with a very unstable and hard to optimize set-based loss cause the model to underfit at the same time not generalizing very well to validation set. On the other side, the model in YOLO loss shows a somewhat satisfying result, giving an mAP of 0.3466. Since YOLO loss consists of only sum-squre errors, it is very easy to optimize and seems to be a better choice on a small datasets such as PASCAL.

This project means to explore the usage of transformers in object detection task, and so far my founding has been bleak. One of the most important aspect of using transformers to do object detection is to eliminate post-processing steps such as non-maximum suppression. To achieve this goal, set-based loss is introduced specifically to make object detection pipeline to not only predict correct class labels but also correct number of objects in the image [1]. However, this task is much more intricate than sliding window predictions since there are no non-maximum suppression to eliminate duplicate predictions. This means such task needs a large amount of data to train sufficiently, is very hard to optimize, and is very likely to overfit since it will require a fairly large model to process all these data. Besides, the transformer itself does not seem to be very stable in object detection task. In both of the model I developed, I have noticed that the transformer would give extremely similar result among query embeddings in the initialization stage and results in all the object proposal to be almost identical. This process will persists for the first several epochs and eventually disappear as more training epochs are introduced. Sun, Cao *et al.* [8] performed a sparsity check into the cross-attention mechanism of the transformer and conclude that the sparsity of cross-attention consistently increase and does not even reach a plateau after 100 epochs. Such

poor sparsity of the cross-attention layer may prevent the decoder from extracting accurate context information from the encoder, causing the model to miss out information and introduce localization error on small objects in particularly.

The future of the transformer in object detection is still unclear. Although the efficiency of the transformer would allow the existence of a much larger model, its inherent lack of locality and translation equivarience [3] coupled with the unstability of its cross-attention layer [8] could eventually prevent it from become as dominant as CNN in object detection task.

5 External Resource

In this project, I largely based my work on the work [1] done by Carion *et al.*. They have a github repository that fully implemented their version of detection transformer: facebookresearch/detr. In its full glory, the model is trained on COCO dataset for 300 epochs and would last 6 days. I implemented a simpler version of the model and switched to YOLO loss when found out that it does not work on PASCAL.

The PASCAL dataset code is adopted from Assignment 3 where I changed some of it to make the images and target come out as expected by my models.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers, 2020.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.

[6] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[7] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. Generalized intersection over union: A metric and a loss for bounding box regression, 2019.

[8] Z. Sun, S. Cao, Y. Yang, and K. Kitani. Rethinking transformer-based set prediction for object detection, 2020.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[10] Z. Zou, Z. Shi, Y. Guo, and J. Ye. Object detection in 20 years: A survey, 2019.