Object Detection With Transformers

Kewal Jani, Karan Parikh, Ahmad F Ishraq
New York University
{kj2062, kap9580, afi225} @ nyu.edu

Abstract

Object Detection is a fundamental task in the domain of computer vision. Recent advances in the vision transformer has led to many breakthroughs in tasks such as image segmentation and classification. In our project, we want to extend upon a dectection transformer, known as DETR, and address the issues it faces. Our proposed methodology is to move away from the model's reliance on convolutional nuerual networks altogether and introduce a self supervised pre-training methodology. Our code be found here

1 Introduction

Object Detection is a popular and comprehensive task in computer vision. When it comes to object detection, most architectures with better performance [10,11] are based on CNNs. This is in part due to the high cost incurred for training transformer models for similar tasks. For example, the DEtection TRansformer (DETR) [1] took more training epochs to converge, at around 500 epochs, which is substantially slower than the Faster R-CNN model [10]. DETR also performs relatively low in detecting smaller objects [1,3]. These problems appeared to us in our initial study when we wanted to perform object detection for environmental and household waste, where we noticed that a CNN model was used over a transformer model [4] despite the promising properties of vision transformers. Initially, our focus was to extend on the waste detection paper [4] by optimizing the DETR architecture but we realized that such an optimization is a seperate task in itself. As a result, our primary focus for the project is to extend on the DETR model and address the issues it faces by exploiting the features and properties of vision transformers.

Since the inception of the DETR model, several improvements were tried to address the aforementioned problems faced by the model. To address the issue of high training time, Deformable DETR [3] was introduced, which introduced a deformable attention module that attends to a small set of sampling locations as a pre-filter for prominent key elements out of all the feature map pixels compared to using the entire feature map as inputs to the transformer. This architecture achieved similar to better Average Precision (AP) compared to both Faster R-CNN and DETR [3] using substantially less training epochs (50 epochs compared to 109 for Faster R-CNN and 500 for DETR). For our project, we want to incorporate new methodologies to extend on the performance issues of the DETR model, similar to that of the deformable DETR.

Problem Statement To summarize our problem statement, we have found that transformer modules are computationally expensive to train, particularly the Detection Transformer (DETR) used for object detection. The model also performs relatively low in detecting small objects. At the same time recent literature on vision transformers have highlighted the benefits of vision transformers in vision tasks, stating its ability to be highly robust to severe occlusions and its ability to encode shape representations without pixel level supervision [13]. In order to exploit these features, we want to modify the DETR model to address the issues it faces and hope to improve the model.

2 Proposed Methodology

Before we introduce our proposed methodology, it is imperative to understand the DETR model on a high level which can be demonstrated by Figure 1.

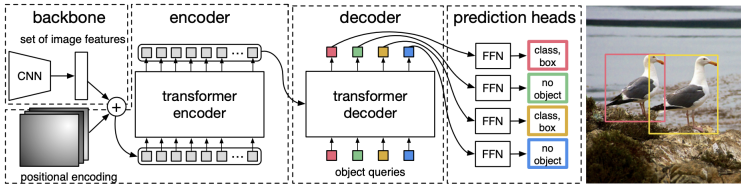


Figure 1: Original DETR model

The DETR model takes an image as an input, and passes it down to a CNN backbone which is typically a ResNet-50 to extract a feature map. Positional encoding is added to the feature map which is then fed to a standard transformer encoder module. Query and key elements are of pixels in feature maps of the backbone layer in the self-attention module which outputs a set of learned representation feature map. The transformer decoder takes in the output feature maps from the encoder and N Object queries represented by learnable positional embeddings as inputs. The attention module in the decoder block is made of cross attention and self attention. For the cross attention, Object queries extract features from the feature maps. The query elements are of object queries and key elements are of the output feature maps of the encoder. For the self-attention layer, object queries interact with themselves and the output is fed to the prediction heads which uses feed-forward networks to obtain bounding box coordinates for prediction.

Given this architecture, we observe that the model uses a CNN backbone as a means to extract feature information over an image. If vision transformers are well equipped for extracting image features, we think it is odd that the DETR model employs this CNN backbone. Among our proposal to make DETR better, we hypothesize that the CNN model is not neccessary and that we plan to remove this backbone and rely on the transformer model entirely.

We propose using the BEiT model (BERT Pre-Training of Image Transformers), which relies on a self-supervised objective during training that has achieved competitive results on image classification and segmantic segmentation [14]. The idea is inspired from the success of Transformer architectures, in particular BERT [8], within the language processing domain that relied on self-supervised pretraining. The self-supervised objective in BEiT, is to mask out some percentage of patches and pre-training the model to predict the masked patches againsts a visual tokenizer. By forcing the model to predict masked patches, the model have shown to learn better feature representations which were later fine-tuned on downstream tasks such as image classification and image segmentation. Given its ability to perform well on downstream tasks, we hypothesize it will also perform well in object detection which hasn't been explored yet. For our project, we propose to use the BEiT encoder and incorporate with the DETR architecture to test whether it performs better or worse.

3 Experiments

We started our experiments by first seeing how DETR does with a simple inference. This was our first step to use exising code to reproduce the results. As can be seen from Figure 2, DETR does a good job in identifying most bottles, however it does poorly on some instances where the mdoel is not so confident.

DETR Implementation To get a better understanding how the model gets these results, we conducted an analysis on how the model was trained. Our analysis shows that the authors trained DETR using the following hyperparameters:

Figure 2: Inference using DETR

Model Name	Hyperparameter and Settings	Values
Backbone: ResNet	learning rate	10^{-5}
	pre-trained	True: Torchvision Library ResNet50
	batchnorm layers	frozen
Transformer	Encoder/Decoder Layers	6
	hidden dimensions	256
	number of heads	8
	dropout	0.1
DETR/ Overall Model	learning rate	10^{-4}
	batch size	2
	weight decay	10^{-4}

Table 1: DETR Configuration

We highlight the above configuration in Table 1 because we will be using these set of hyperparameters and settings throughout our experiments.

To get a better understanding of how this model works, we found a working notebook on fine tuning the DETR model on the balloon dataset. In order to fine-tune the model, we loaded the pre-trained weights that were made available to us through the torch vision hub. Following the configurations in Table 1 we fine-tuned the model for 10 epochs. We obtained the following results in Figure 3.

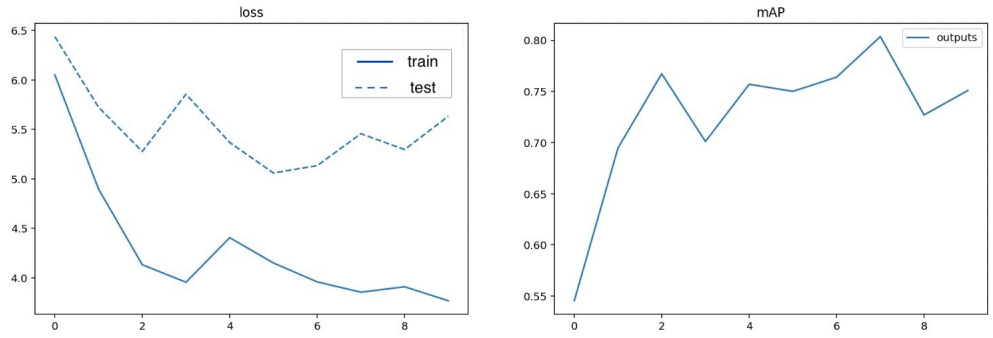


Figure 3: Torch.Hub Pre-trained weights to Fine-tune DETR for 10 Epochs

We also fine-tuned our model using another set of pre-trained weights that we found in the official repository of DETR, and conducted a similar experiment for 10 epochs with similar settings. We obtained the following results as shown in Figure 4.

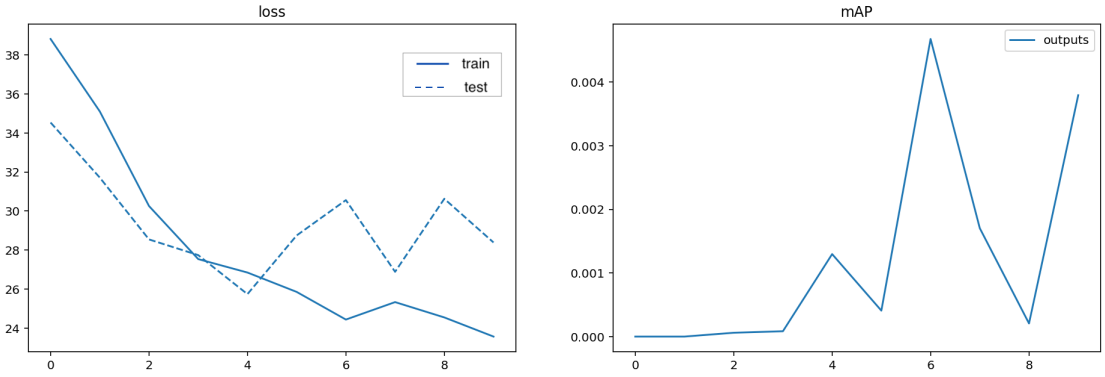


Figure 4: Official Repository Demo Pre-trained weights to Fine-tune DETR for 10 Epochs

In both the experiments, we see that the overall loss over both train and test is decreasing. However we observed descrepencies. In Figure 3, we see that the test loss is higher compared to the train loss, as opposed to Figure 4, where the difference in train and test loss are not as high. Furthermore, the mean average precision (mAP) for Figure 3 is substantially higher. The mAP increased from the one's reported in the official paper [1] from $AP = 42$ and $AP_{50} = 62$. However, the mAP with the official repository performed significantly worse. In order to see whether that was due to fact that we were only running for 10 epochs, we fine-tuned both pre-trained weights for 100 epochs and reported the results on Figure 5 and 6.

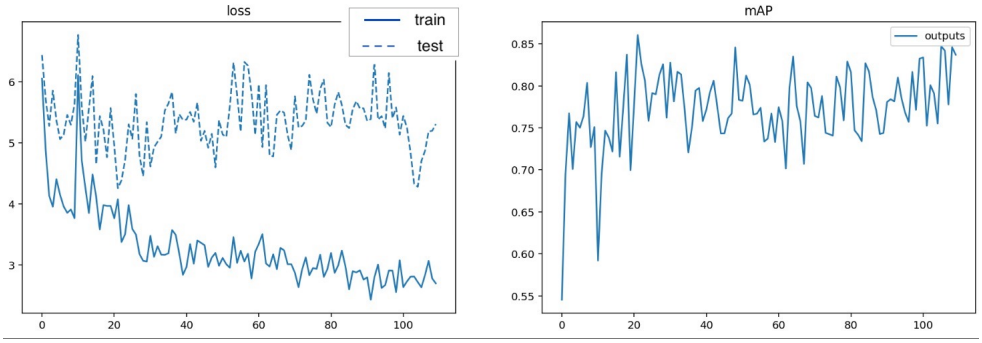


Figure 5: Torch.Hub Pre-trained weights to Fine-tune DETR for 100 Epochs

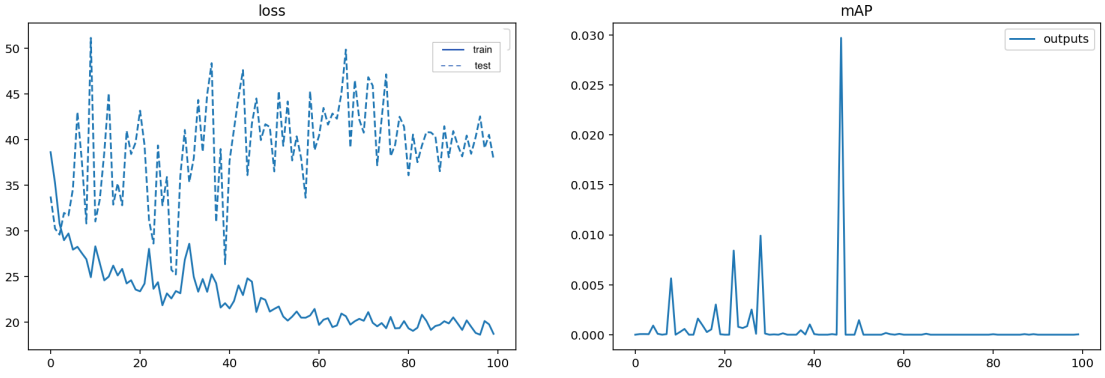


Figure 6: Official Repository Demo Pre-trained weights to Fine-tune DETR for 100 Epochs

(a) Original Image

(b) Resulting Image

In running for 100 epochs, we find the torch.hub pre-trained weights performed better than the official pre-trained weights. The mAP in Figure 5 continued increasing, whereas in Figure 6, we see mAP totally flatline indicating the model is learning almost nothing. The demo model was trained with an older version of the code, the hyperparameters were a bit different, on 8 nodes with batch size 1 for 700 epochs. It's just for simplicity, as dealing with masking would require additional code. With our baseline model (not the demo), what we observe is that if a model was trained with batch size 1, when we evaluate with more than 1 im/gpu the AP drops significantly, presumably due to boundary effects for which the model was not trained for. If we train with more than 1, and evaluate with a different number of images, including 1, there is some variation but it's not significant [17].

This performance degradation is due to the following reason based on our findings. The torch hub pre-trained weights were the pre-trained weights that were used to train the official DETR model and uploaded to the hub. Another set of pre-trained weights were also uploaded by the authors, which they called the Demo DETR, which is a minimal implementation of the original model. In the demo version, the authors used learned positional encoding (instead of sine used in the original model), positional encoding is passed at input (instead of attention), fc bbox predictor (instead of MLP). Given these differences, when we ran the fine-tune, we used the main file that had the original DETR instead of the demo version. Hence we see the degradation.

This finding presented us with a challenge. The inference we showed in Figure 2 used the pre-trained weights from the demo model, and the model used was the demo version itself. Our goal was to modify the demo version of DETR and incorporate BEiT with it. Nonetheless, we proceeded with understanding how we can remove the ResNet backbone and why it plays such a crucial role.

Backbone and CNN As mentioned before, the authors used a ResNet backbone in order to extract feature representation of the images. We propose to replace the ResNet with BEiT Encoder. Specifically, we want to use the feature representations produced by BEiT's output which outputs $H = [h_{CLS}, h_1, ...h_N]$, where h_i is the vector of the i-th image patch and h_{CLS} is the class embedding following the standard transformer output.

For us to replace the backbone, we first conducted a study of the demo code to see how an image is passed down through the model. The code and output is provided below.

```
torch.Size([1, 3, 224, 224]) #input shape:(batchsize, num_channels, h,w)
resnet conv1 torch.Size([1, 64, 112, 112]) #(batchsize, num_channels, h,w)
reslayer1 torch.Size([1, 256, 56, 56]) #(batchsize, num_channels, h,w)
reslayer2 torch.Size([1, 512, 28, 28]) #(batchsize, num_channels, h,w)
reslayer3 torch.Size([1, 1024, 14, 14]) #(batchsize, num_channels, h,w)
reslayer4 torch.Size([1, 2048, 7, 7]) #(batchsize, num_channels, h,w)
conv layer torch.Size([1, 256, 7, 7]) #(batchsize, num_channels, h,w)

#positional embeddings of shape
pos cat col= torch.Size([7, 7, 128])
pos cat row= torch.Size([7, 7, 128])
pos= torch.Size([49, 1, 256])
#transformer input shape
transformer input 1= torch.Size([1, 49, 256]) #(batchsize, seq_len=(h*w), hidden_dim)
transformer input 2= torch.Size([1, 100, 256]) #(batchsize, object_queries, hidden_dim)
torch.Size([1, 100, 256]) #output shape
```

Once we understood how image is passed down the model, we moved forward with training our model from scratch with the Resnet backbone. As mentioned earlier, we were conducting our analysis of DETR through the demo version, which is a minimal implementation. Further, we saw from Figure 6 how the demo version does not converge. This led us to move our focus to the main repository and using the files produced by the authors. DETR was originally trained on the COCO-17 Dataset, which is a large dataset used for benchmarking Object Detection tasks. The dataset is about

17 GB due to the sheer amount of class labels (about 91 classes) it contains. Allocating all these memory on our local devices or Colabotary environment posed a challenge. The authors trained the baseline DETR for 300 epochs on 16 V100 GPUs which takes 3 days. The authors estimate it will take around 6 days on a single machine with 8 V100 cards. This posed a challenge for us since neither Colab nor HPC would allow us to run such computationally heavy jobs. As a result, we resorted to training on a smaller dataset.

WIDER FACE Dataset We resorted to using a face dectection dataset [15] that has around 4GB of data, consisting of only one class. We assumed this was a reasonable amount of data which will enable us to test the model. We would also like to acknowledge that we are only using this dataset to understand how DETR performs object detection and creating a working code that will enable us to remove the backbone, and run training on the original COCO dataset to benchmark. The aim of our project is not to use this dataset for any other such purposes that concern the ethical issues surrounding datasets that contain human faces.

Training We conducted our training using Wide ResNet, ResNet backbone. We used the same configuration and hyperparameter setting as the original DETR implementation in Table 1. We only ran training for 10 epochs given the limited time and resource available to us. We plot the train and test loss, and mAP of all these 3 models in the figures below.

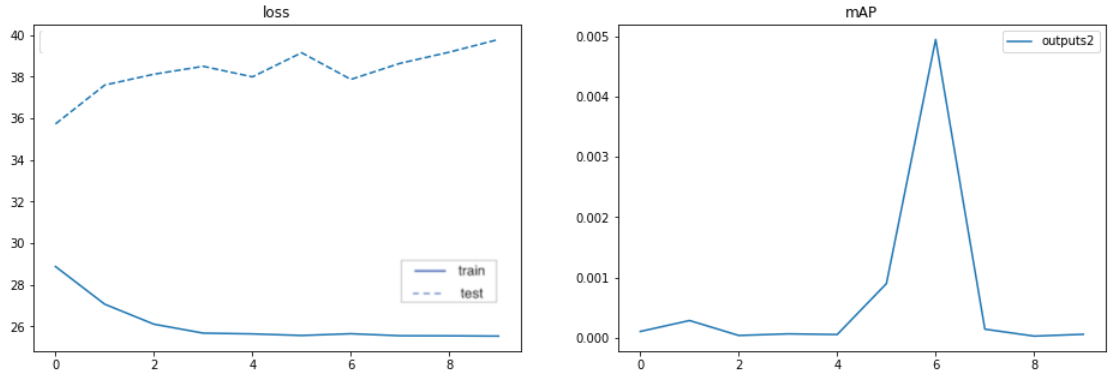


Figure 8: Wide ResNet Trained for 10 Epochs on Face Dataset

The low mAP scores are expected since we are only training for 10 epochs and with no pre-trained weights. We expect that within 100 epochs it will yield similar results to the original model.

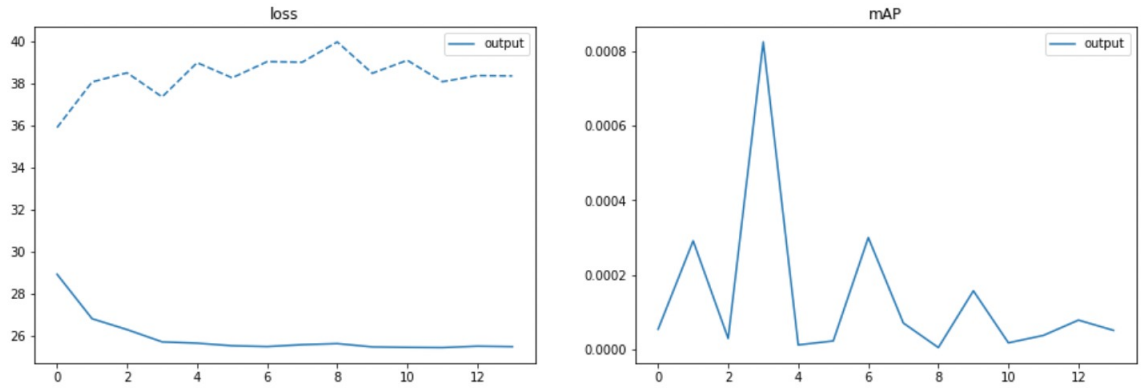


Figure 9: Resnet trained for 13 epochs on Face Dataset

Using BEiT Backbone Now that we have figured out a way to train the model from the author's repository, we move forward by removing the ResNet backbone altogether. We pay attention to the

image shape that is passed into the DETR transformer and modify the DETR file. Figure 8 illustrates our model.

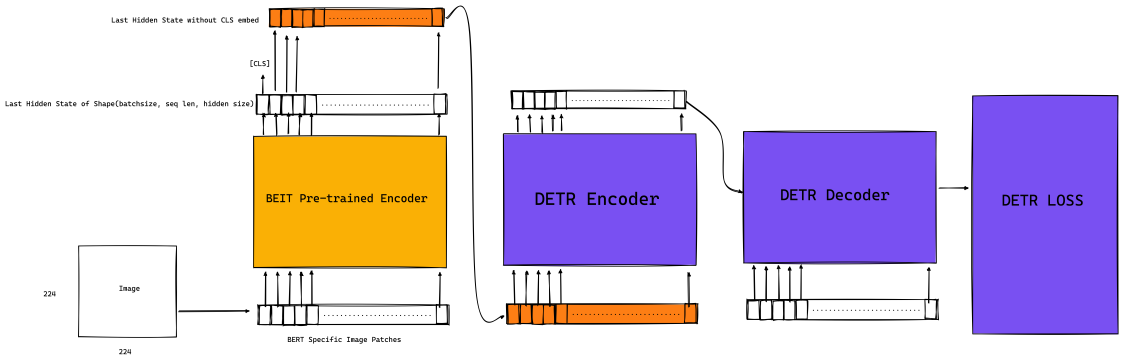


Figure 10: BEiT-DETR Implementation

For our implementation, we take the pre-trained BEiT model used for Image Classification that was available through the Hugging Face Library, and take the model's second last output. We also remove the class embedding layer, since we are only interested in the image representation. We feed this output as an input to DETR's transformer. In doing so, we remove the positional embeddings used in the original DETR, since we hypothesize that the BEiT output has mapped the relevant positional encoding to the images. We freeze the BEiT layer during training, and keep the rest of the DETR implementation as is. The model is demonstrated in Figure 8. However, given that we had limited time, we resorted to using the demo DETR's version of the DETR class definition, which means that our model is limited to the poor outcomes we saw in Figure 6. We also changed the batch size to 1. We further resized the image to 224 by 224 as this is what the BEiT encoder expects to recieve as input. We also changed the hidden dimensions of the DETR Encoder-Decoder to 768 in order to match the hidden dimension of the BEiT Encoder. We keep the remaining hyperparameters highlighted in Table 1 the same throughout the rest of our model. We trained our model on the Face Dataset for 10 epochs and plotted our results in Figure 9.

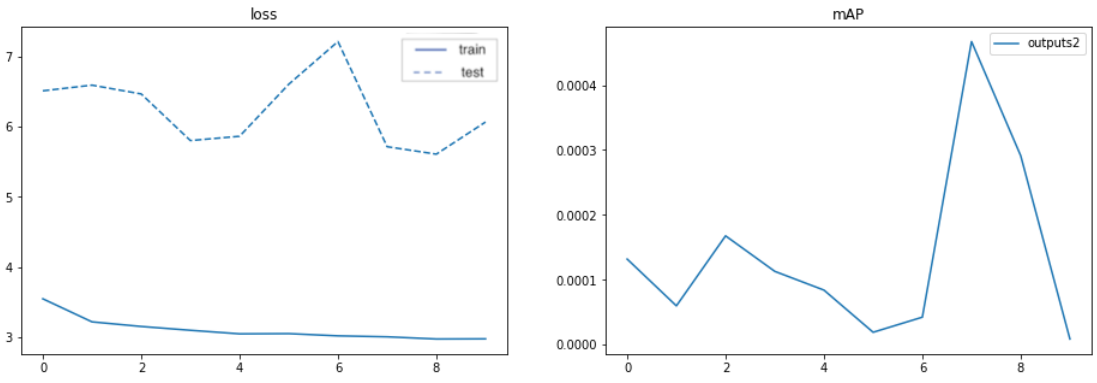
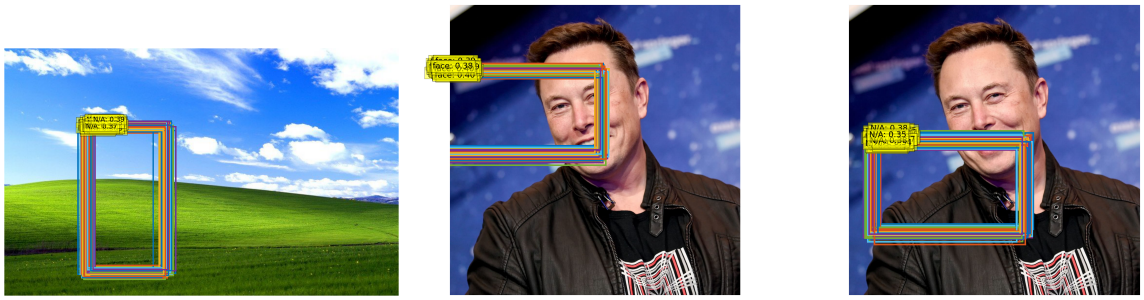


Figure 11: BEiT-DETR 10 Epochs Training Outcome

We can see from Figure 9 that we obtain similar results to Figure 7, which was trained with a Wide ResNet backbone. We observe that our model starts with a lower error compared to Wide ResNet backbone model, which we find is odd since we expected it to start from similar level of loss compared to the ResNet backbone. To compare these models further, we took note of the amount of time it took to train both these models and the total number of trainable parameters these models had.



(a) no object (b) face predicted (c) no object

Figure 12: Experiment Outcome over 2 Images

model	total parameters	training time
DETR-ResNet	41,149,510	04:03:26
DETR-Wide ResNet	84,149,510	5:22:35
BEiT-DETR	80,450,329	4:05:52
Faster R CNN + FPN	42,000,000	–

Table 2: Model Comparison for 10 Epochs except for Fast RCNN

Another observation in our experiment was that the train class error logged errors at 100 throughout the training set whereas the Resnet architecture logged substantially lower class error. This indicates that all the images have 100% error rate. We hypothesize that this could be due to the fact that we modified our DETR implementation with the demo DETR class instead of the original DETR class. We also hypothesize the problem could be due to us using a Face dataset. BEiT was pre-trained using ImageNet which has a varying amount of images whereas the Face dataset would only have images of faces. This indicates that in the future we may need to first fine-tune BEiT on the Face dataset or use our model to train only on the COCO dataset, given that we are presented with enough resources that the model demands. It is also important that we note the time taken to train our models. The ResNet background took about 5 hours to train for 10 epochs whereas the BEiT-DETR model took around 4 hours, but also with less parameters. This highlights the problem that transformer models are indeed costly to train. If we unfreeze the BEiT layers, then the parameters could double making it extremely costly over the ResNet counterpart. Especially with Faster RCNN models that have around 42 million parameters [3] and original DETR with ResNet50 which has around 41 million parameters.

We also tested our trained model on a set of images to see how we have done. Figure 10 illustrates what our model predictions.

For Figures 12.b and 12.c, we see that our model has detected a face and also detected no face. This happened as we initialized our model twice and ran inference on the same image. It produced two different results which demonstrates how we have a 100% class error. Figure 12.a was only initialized once and it detected no face at first go, however we expect it to produce a different result if we re-initialize our model and run inference again.

Conclusion Overall, we were able to remove the backbone of the DETR model with BEiT's encoder. However, we still need to conduct further experiements. We need to either fine-tune BEiT on the Face dataset or run training over the original COCO dataset. We will need to run training on the COCO dataset nonetheless since COCO is used as the current industry benchmark for all object detection task. That way we could really evaluate our model. We also want to acknowledge that our BEiT incorporation with DETR was done rather hastily given the short span of time we had to implement this task. We will ultimately need to use the original DETR class implementation as opposed to the demo version, and study the role that positional embeddings play. We hope to continue working on this in the future as there is a lot that can be explored. Given the relative success of transformers in computer vision, it is remarkable that there is no fully transformer based architecture for object detection task.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov & S. Zagoruyko. End-to-End Object Detection with Transformers. In arXiv:2005.12872 [cs.CV], 2020.

[2] H. Bao, L. Dong & F. Wei. BEiT: BERT Pre-Traning of Image Transformers. In arXiv:2106.08254 [cs.CV], 2021.

[3] X. Zhu, W. Su, L. Lu, B. Lin, X. Wang & J. Dai. Deformable DETR: Deformable Transformers for End-to-End Object Dection. In arXiv:2010.04159 [cs.CV], 2021.

[4] S. Majchrowska, A. Milkolajczyk, M. Ferlin, Z. Klawikowska, M. Plantykow, A. Kwasigroch & K. Majek. Waste Detection in Pomerania: Non-Profit Project for Detecting Waste in Environment. In arXiv:2105.06808 [cs.CV], 2021.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser & I. Polosukhin. Attention Is All You Need. In arXiv:1706.03762 [cs.CL], 2017.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit & N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In arXiv:2010.11929 [cs.CV], 2021.

[7] A. Krizhevsky, I. Sutskever & G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS, 2012.

[8] J. Devlin, M. Chang, K. Lee & K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In arXiv:1810.04805 [cs.CL], 2019.

[9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever & D. Amodei. Language Models are Few-Shot Learners. In arXiv:2005.14165 [cs.CL], 2020.

[10] S. Ren, K. He, R. Girshick & J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In arXiv:1506.01497 [cs.CV], 2016.

[11] J. Redmon, S. Divvala, R. Girshick & A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In arXiv:1506.02640 [cs.CV], 2016.

[12] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu & M. Pietikäinen. Deep Learning for Generic Object Detection: A Survey. In arXiv:1809.02165 [cs.CV], 2019.

[13] M. Naseer, K. Ranasinghe, S. Khan, M. Hayat, F. Khan, M. Yang. Intriguing Properties of Vision Transformers. In arXiv:2105.10497 [cs.CV], 2021.

[14] H. Bao, L. Dong & F. Wei. BEiT: BERT Pre-Training of Image Transformers. In arXiv:2106.08254 [cs.CV], 2021.

[15]Yang, Shuo and Luo, Ping and Loy, Chen Change and Tang, Xiaoou. WIDER FACE: A Face Detection Benchmark. In arXiv:1511.06523 [cs.CV], 2016.

[16] S. Zagoruyko, N. Komodakis. Wide Residual Networks. In arXiv:1605.07146 [cs.CV], 2017.

[17] https://github.com/facebookresearch/detr/issues/63