**ECE 508-02**

**Final Project Report**

By: Michael Schmidt

A20460468

Due: 12/4/2025

Table of Contents

[1. Introduction 3](#_Toc215670760)

[1.1 Objective: 3](#_Toc215670761)

[1.2 Dataset: 3](#_Toc215670762)

[1.3 Hyperparameter Limits: 3](#_Toc215670763)

[2. Description of Baseline Model 3](#_Toc215670764)

[2.1 Model Architecture: 4](#_Toc215670765)

[2.2 Examples of Queries: 5](#_Toc215670766)

[2.3 Evaluation of Baseline Model: 5](#_Toc215670767)

[3. Improved Methods 5](#_Toc215670768)

[3.1 DN-DETR: 5](#_Toc215670769)

[3.1.1 DN-DETR Architecture: 6](#_Toc215670770)

[3.1.2 Brief Evaluation of method Improvement: 7](#_Toc215670771)

[3.2 Lower Decoder Layer: 7](#_Toc215670772)

[3.2.1 Reducing Decoder Layers: 8](#_Toc215670773)

[3.2.2 Brief Evaluation of method Improvement: 8](#_Toc215670774)

[3.3 Lower number of Queries: 9](#_Toc215670775)

[3.3.1 Reducing Queries: 9](#_Toc215670776)

[3.3.2 Brief Evaluation of method Improvement: 9](#_Toc215670777)

[3.4 Optimizing FLOPs Calculations: 10](#_Toc215670778)

[3.4.1 Using AMP: 10](#_Toc215670779)

[3.4.2 Brief Evaluation of method Improvement: 11](#_Toc215670780)

[4. Performance Evaluation 12](#_Toc215670781)

[4.1 Model Differences: 12](#_Toc215670782)

[4.2 Detection Results: 12](#_Toc215670783)

[5. Conclusion 15](#_Toc215670784)

[6. References 15](#_Toc215670785)

## Introduction

### 1.1 Objective:

The objective of this project is to understand and implement the baseline DETR model for image classification, as well as further improve it with unique ideas. Areas of improvement that will be focused on are improving the detection accuracy, increasing the training efficiency, or reducing the computational complexity. The methods of evaluation that will be used are measuring the training speed, evaluating the IoU metrics at different percentages, measuring the inference speed, measuring the number of parameters in the model, as well as seeing examples of the model running inference on different images.

### 1.2 Dataset:

The dataset that will be used for this project is a custom version of the PASCAL VOC dataset. This contains 20 different classes of different objects and animals (such as airplane, bird, cat, etc). Unlike the original VOC, 5717 images will be used for training, and 5823 will be used for testing.

### 1.3 Hyperparameter Limits:

In order to ensure fairness with other students, certain hyperparameters will be fixed while training the model. This would include:

* Learning Rate = 5e-5
* Weight Decay = 1e-4
* Epochs = 10
* Batch size = 4

## Description of Baseline Model

[1] The DETR model is a Detection Transformer used to classify images. Unlike traditional CNNs that extract features to feed in a dense network, DETR uses transformers that were originally made for natural language processing (NLP) and uses it for classification. This is done by using a combination of CNN as a backbone, an encoder-decoder layers for processing, and Hungarian matching at the end of predictions.

### 2.1 Model Architecture:

The DETR model starts with using a CNN, such as ResNet, as a backbone to extract rich features. From there, it calculates the positional encoding of the features using sinusoids and flattens them, passing into the encoder. The job of the encoder is to build a global, context-aware representation of the entire image. This would mean that every pixel in the image has self-attention with one another, and it can understand the positional relationship of one another. The encoder layer is built with 6 layers.

A diagram of a software system

AI-generated content may be incorrect.

(Figure 1: Inside Encoder & Decoder single layer)

Once the rich feature map is produced, it is fed to the decoder, where it can start to predict objects and their bounding boxes. The decoder takes in the feature map, along with a set of learned objected queries, typically 100 by default. Each query is a slot for one possible object, with its learned pattern detector and a trainable prompt. All the queries have self-attention, where they can communicate with one another, and each query has cross-attention, where it can look at the entire feature map produced by the encoder. This allows the queries to pull spatial location information. Each query does not have to have an object detected, it can be trained to specialize in detecting a certain object, or say no object is found. The decoder layer is built with another 6 layers.

After the queries have finished processing, they are all fed into a Feed-Forward Network (FFN) that outputs a class probability as well as a bounding box for the object. The last thing the model does is use a Hungarian matching method to prevent duplicates of objects detected and figure which queries are more specialized in learning which objects.

A diagram of a transformer decoder

AI-generated content may be incorrect.

(Figure 2: Overall architecture of DETR)

### 2.2 Examples of Queries:

In figure 3 below, there are 20 images each representing what a single query node sees after training on the COCO 2017 dataset. Each query node is trained to learn the patterns of detecting a specific class in the image, in a specific area of the image. The green dots represent the center point of a bounding box of an object detected in a small box. The red dots represent the large horizontal boxes, and the blue represent the large vertical boxes.

A group of colorful splatters

AI-generated content may be incorrect.

(Figure 3: Inside of a query)

### 2.3 Evaluation of Baseline Model:

The baseline DETR model given after training for 10 epochs with the set hyperparameters have an average accuracy of 60% where the IoU threshold is set to 0.5. It took roughly 160 minutes to train on the T4 GPUS from google and has an average inference speed of 0.0844s per image. It also has a total of 41,506,778 parameters.

## Improved Methods

### 3.1 DN-DETR:

One of the problems with the baseline model is that it is slow to converge. Typically, DETR would need at least 300 epochs for the queries to learn how to detect objects. This is because unlike CNN based model that have handcrafted region proposals, anchor boxes, or hand-designed matching, DETR must learn to produce one prediction per query without duplicates from scratch based on the data, relying on the Hungarian matching. In the early stages of training, almost all predictions are garbage, therefore it takes a while for the loss functions to “nudge” the model in the right direction. A solution to this is to implement a denoising strategy called DN-DETR.

#### 3.1.1 DN-DETR Architecture:

[2] DN-DETR works by creating “easy examples” for the model to learn from, making it converge much earlier in the training. It all happens in the decoder stage of the model, where for every image given to train on, it will create a set number of duplicate groups of the image containing both a positive and negative example. The positive example in the groups will have all the correct labels of the objects in the image, and the correct bounding boxes that have a slight noise to them. The negative example in the groups will have incorrect labels for the objects, and incorrect bounding boxes. The idea is to give the model examples to learn what is a good prediction, and what is a bad one.

A diagram of a diagram

AI-generated content may be incorrect.

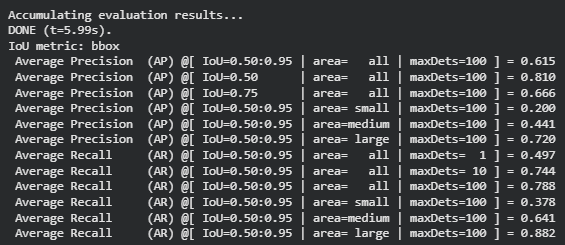
(Figure 4: DN-DETR architecture)

Figure 4 above illustrates the full architecture of implementing the DN-DETR. In addition to creating the example groups, DN-DETR also creates “denoising queries” in addition to the normal queries. These are created automatically for each specific image it is training on, where one DN-query represents an object in the image. They are deleted after it is done training on an image. The DN-queries are created for each group, and each group has self-attention, where they can see everything inside the group, but each group cannot see other groups. Nor can the normal queries see the DN-queries. This is to prevent the normal queries from “copying” the DN-queries”.

After the query calculations, the normal queries undergo Hungarian matching like normal, however the DN queries do not, as they already know their targets, thus they use a supervised loss of L1 + GIoU + cross-entropy. By combining the loss from the DN queries and the Hungarian matching from normal queries, the backpropagation of the network has more gradients, allowing the decoder to predict objects better, and for the encoder to pick up better features.

#### 3.1.2 Brief Evaluation of method Improvement:

When testing the improved model, the hyperparameters set for the DN-DETR are using 5 DN groups for each image. The noise ratio for corrupting the negative examples of the labels is 0.2, and the noise ratio for making bad bounding boxes used is 0.4.



(Figure 5: IoU metrics of DN-DETR)

After evaluating the improved model with the set hyperparameters, the overall accuracy is 0.615, which is already a great improvement over training at 10 epochs. The training time has increased to be completed in ~170 minutes. The average inference time has gone down slightly to 0.0843, which is most likely due to internal states of the GPU, so it is negligible. The number of parameters has also increased to 41,645,018.

### 3.2 Lower Decoder Layer:

Another problem with the baseline model is that it used 6 layers in the decoder to predict the object. This causes inference issues as each layer takes time and resources to compute. The loss for cross-attention also scales with the number of layers. In addition to the loss, the queries also communicate with each other too frequently, which can lead to over-correlation, removing the unique specialization of each query. The solution is to reduce the number of decoder layers.

A white paper with black text

AI-generated content may be incorrect.

(Figure 6: Formula for cross-attention)

#### Reducing Decoder Layers:

By reducing the number of decoder layers in the model, this reduces the number of FLOPs and memory, allowing the model to train faster and reduce the inference time. It can also help prevent the model from having over-correlations, and studies have shown the model still has competitive accuracy with fewer layers [3].

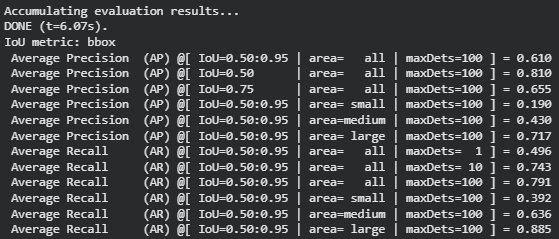
A diagram of a snow ski lift

AI-generated content may be incorrect.

(Figure 7: Reducing amount of decoder layers)

#### 3.2.2 Brief Evaluation of method Improvement:

Reducing the number of decoder layers in the model was fortunately built into the baseline script. Simply setting the hyperparameter dec\_layers = 4 will create the model to have 4 decoder layers. There were some issues loading the given checkpoint model into the training model as the number of weights do not match up, however setting the parameter “strict=false” allows the load the state of the checkpoint to match existing weights and discard unused weights.



(Figure 8: IoU metrics of DN-DETR + 4 Decoder Layers)

Figure 8 above shows that the average accuracy of the model has gone down slightly to 0.61. This is to be expected as there are fewer layers to predict the objects for each image. The training time, however, has improved back to taking ~160 minutes on the GPU, with the average inference time improving to 0.0832s, and the number of parameters greatly reducing to 38,487,514.

### 3.3 Lower number of Queries:

Another problem with the baseline model is that it used 100 queries to predict the objects in the image. Since each query learns to detect only one object and its bounding box, it can be redundant to have so many queries calculating objects in the dataset of only 20 classes. The solution is to reduce the number of queries to speed up training and inference time.

#### 3.3.1 Reducing Queries:

Reducing the number of queries to 50 can improve the training speed and inference of the model. This can help reduce the number of FLOPs and memory, as having many queries causes redundant calculations. The number 50 for the number of queries has been chosen as a good rough estimate, as the number of queries needed can be calculated by estimating the max number of objects to detect in an image, which has been chosen as 20, double it and add a little more. Reducing the number of queries can also help force specialization of the different normal queries [4].

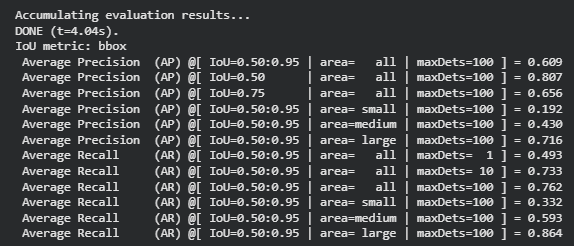
A diagram of a person's prediction

AI-generated content may be incorrect.

(Figure 9: Reducing the number of queries)

#### 3.3.2 Brief Evaluation of method Improvement:

Like with reducing the number of decoder layers, reducing the number of queries is easy to implement as it is built into the baseline model code. By setting the hyperparameter “num\_queries” to 50, the code will build the decoder layer with only 50 queries. There is another issue when running the training script, as the checkpoint model given was trained with 100 queries, causing a shape issue. Another flag was created called “resize\_queries” where when used, it will detect if the checkpoint model has more queries than the model being built, it will loop through the checkpoint, only keeping the first 50 queries and throwing away the rest, essentially creating a new checkpoint path to train on.



(Figure 10: IoU metrics of DN-DETR + 4 Decoder Layers + 50 Queries)

Figure 10 above shows that the average accuracy of the model has reduced to 0.609, which is expected as the performance has not changed too much. Most likely the model failed to predict only a handful of classes. The training time on the GPU has stayed the same at ~160 minutes, however, the average inference time per image has reduced to 0.0828s, and the number of parameters has reduced to 38,474,714.

### 3.4 Optimizing FLOPs Calculations:

Another problem with the baseline model is that it uses FP32 for all calculations. This is great as FP32 can hold a lot of information, making its numerical calculation very precise. The downside however is that it takes a lot of time and resources to calculate all the 32 bits, causing the model to have a high inference and training speed. The solution is to use FP16 where it is safe via Automatic Mixed Precision (AMP).

#### 3.4.1 Using AMP:

PyTorch has a built-in function called “autocast” where it replaces calculations to FP16 where it is safe in the forward pass. The problem with using FP16 is that since it has a much lower range to hold the data, there is a risk of underflow, where the gradients can be zero, or overflow, where the gradients can reach infinity. This can be prevented by scaling the loss to a large factor while computing the backpropagation. After calculations, the gradients are scaled back down before being passed to the optimizer. This can be handled automatically with PyTorch “GradScaler” function. As mentioned, not all calculations can be changed to FP16, as calculations like Loss computations, or Softmax require FP32 for stability and accuracy. AMP automatically decides where it can replace FP32 calculations with FP 16. Reducing calculations to FP16 where it is safe should also give negligible accuracy loss.

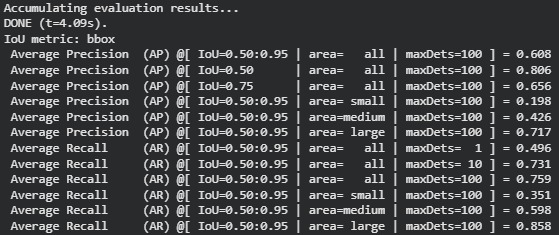
A comparison of fractions with numbers

AI-generated content may be incorrect.

(Figure 11: Difference of FP32 and FP16)

#### 3.4.2 Brief Evaluation of method Improvement:

The original paper for DN-DETR also used AMP for their training [2]. By implementing it here as well, there should be a noticeable improvement in the performance of the model. Looking at figure 12 below, the average accuracy of the model has reduced to 0.608. This is expected as since there is less range in the calculations, the model may detect a handful of objects incorrectly. The internal state of the GPUs could also have an effect there; thus, the difference is negligible. The training time of the model, however, has greatly improved, taking ~110 minutes to train. The average inference time has also greatly improved, taking an average of 0.0582s per image. The number of parameters has not changed as the model architecture has not changed.



(Figure 12: IoU metrics of DN-DETR + 4 Decoder Layers + 50 Queries + AMP)

## Performance Evaluation

### 4.1 Model Differences:

The changed made from the baseline DETR model are implementing a denoising strategy using DN-DETR, reducing the number of decoder layers to 4, reducing the number of queries to 50, and using AMP. When combined, these improvements have increased the accuracy of the model, improved the training efficiency and time, improved the average inference time per image, and reduced the number of parameters.

A screenshot of a computer screen

AI-generated content may be incorrect.

(Figure 13: Baseline Model vs Improve Model)

Looking at the metrics in figure 13 above, compared to the baseline DETR model, the improved model improved the training time to ~110 minutes. The accuracy has improved to 0.608, the average inference time is 0.0582s per image which gives about 17fps compared to the baseline model of 11.8fps. Finally, the number of parameters has been reduced to 38,474,714.

### 4.2 Detection Results:

The figures below are examples of the model running inference on certain images from the dataset. The images in the red box are the performance of the baseline model. Images in the green box are the performance of the improved model. Overall, the improved model was able to detect the same objects as with the baseline model, but with a higher confidence level. The exception is figure 17, where the improved model failed to detect the potted plant. This could be due to greatly reducing the number of parameters, causing the model to struggle to detect a complex looking object. However, with longer training, the model should be able to detect it.

A collage of images of an airplane

AI-generated content may be incorrect.

(Figure 14: Detection Comparison of airport)

A collage of images of a bicycle parked on a street

AI-generated content may be incorrect.

(Figure 15: Detection Comparison of bicycle)

A collage of a person on a motorcycle

AI-generated content may be incorrect.

(Figure 16: Detection Comparison of Women and Bike)

A table with food on it

AI-generated content may be incorrect.

(Figure 17: Detection Comparison of Dining Table)

## Conclusion

The baseline model of DETR is already exceptional as it has an acceptable model accuracy detection and was able to train without handcrafted components like in CNN based detection models. To further improve the model, a few additions were added. These included denoising targets using DN-DETR, reducing the number of decoder layers, reducing the number of queries, and adding AMP.

With these improvements, the improved model has outperformed the baseline in all areas that are targeted in this project. The training time has improved to take ~110 minutes, the training efficiency has improved causing the model to have an average accuracy of 0.608, the average inference time per image is improved to 0.0582s running at 17fps, and the number of parameters has greatly reduced to 38,474,714. With further training, this improved model can further outperform the DETR model, however as the number of epochs is limited to 10, the results achieved already show promise of the baseline model being outperformed.

## References

1. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” arXiv preprint arXiv:2005.12872, 2020.
2. F. Li, H. Zhang, S. Liu, J. Guo, L. M. Ni, and L. Zhang, “DN-DETR: Accelerate DETR Training by Introducing Query DeNoising,” arXiv:2203.01305, 2022.
3. Z. Yao, Y. Ai, L. Gao, and B. Li, “Efficient DETR: Improving End-to-End Object Detector with Dense Prior,” arXiv:2104.01318, 2021.
4. Z. Xu, S. Wang, H. Yu, and G. Zhang, “DQ-DETR: DETR with Dynamic Query for Tiny Object Detection,” arXiv:2404.03507, 2024.