AIN 433: DETR Replication Project

Part 1: Overview

The goal of this assignment is to replicate the DETR model using a simpler architecture and using a smaller dataset to find pedestrians. The pipeline is: Backbone model + Pos encoding -> Encoder -> Decoder -> Pred_heads.

The most important parts were the Architecture, Hungarian matching, and Bipartite loss.

Part 2: Dataset & Setup

Dataset: PennFudanPed is a small dataset with pedestrian images. Test / Val / Test sets are given points.

Preprocessing:

- 512 x 512 resolution upscale normalization on images.
- Augmentation is done with flip, jitter, and flip+jitter.

PennFudanPed

├── Splits

│ ├── Train

│ ├── Test

│ └── Val

└── 2220765014.ipynbFolder layout:

Part 3: Model Architecture

1. Components

1. **Backbone (ResNet, V2M):** Resnet got better results than V2M.
2. **1x1 Conv**
3. **Pos_encoding (Sinusoidal):** Using the pos_encoding model is learning where pixels are so that it can make sense of positions. Sinusoidal encoding is used to get position using a concrete formula so it is fast and makes sense to model.

4. **Encoder:** Since the dataset is small a basic and simple model works best but I started with 2 layers 8 head. At the end 2 layer and 4 head worked best.
5. **Decoder:** Same with encoder.
6. **Parameter (Queries):** Queries are used for specialized part image checker. Using queries it check some part of the code and give more weight to that point. Again small dataset made me use 15 queries for the best model against 50 or 100 queries.
7. **Linear (class_head)**
8. **Linear (box_pred)**

2. Hungarian Matching / Loss

It is used for predicting the boxes. Since the model predicts 100 objects in an image most of them are no class. Using bipartite matching we assign every predict to the most probable class. This way we get the right classes to lowest loss one and others are labeled as no class. For the loss we get intersection over union to learn how well we predicted.

3. Small Dataset Constraints

Like I said before transformers are not working good with small datasets. Augmentations make it better but since it is still small it doesn't work well. So backbone small queries are more impactful and simple transformer is better.

Architecture choices:

Queries = 15, 20 , 50 ,100

A.heads = 4, 8 ,16

T.Layers = 1, 2, 3

Eos coef = 0.1 , 0.2

Backbone = resnet , V2 mobilenet

I tested all of them and written in experiments . Best model was

RESNET, 4 HEADS , 2 LAYERS , 15 QUERİES , 0.2 EOS

Part 4: Training

1. Using regex get the GT boxes resize to 512 and normalize everything.
2. Like I said it the upper part I tried many architectures and got the results from baseline model comparison to other choices. Got the best model architecture. I said all the things in the Architecture choices part in upper part.

Final Model Configuration

- **Learning Rate:** 0.001 (Best). 0.0001 was too small and 0.01 was too fast.
- **Data:** Augmented (better than original).
- **Backbone:** RESNET (More weight).
- **Transformer:** Simple transformer (4 head, 2 layer), small query of 15.
- **Batch:** 16 batch training.
- **Loss:** eos_coef of 0.1 is best.
- **Optimization:** I used scaler for faster computing and scheduler for better lr calculation on every model.

  This config is used mostly because of small dataset.

Part 5: Experiments & Ablation

Data Augmentation

- Non augmented data Test mAP@0.5: **0.1414**
- Augmented Test mAP@0.5: **0.1468**
- *Conclusion:* So augmented data is better.

Backbone Comparison

- Backbone ResNet gives Test mAP@0.5: **0.1013**
- Backbone V2 gives Test mAP@0.5: **0.1104**
- *Conclusion:* They work almost same.

Query Analysis (N_queries)

- Queries = 100 Test mAP@0.5: **0.0447(Resnet)**
- Queries = 50 Test mAP@0.5: **0.1468** (ResNet)
- Queries = 20 gives Test mAP@0.5: **0.3970 (**ResNet)
- Queries = 15 gives Test mAP@0.5: **0.4314 (**ResNet)
- Queries = 15 gives Test mAP@0.5: **0.3550 (**V2)
- Queries = 12 gives Test mAP@0.5: **0.3915**
- *Conclusion:* Small queries get better results. 100 queries got 0; anything over 30 might be too big for the model.

Model Complexity (Heads & Layers)

- Base (8 layer, 2 head): **0.4314**

- N_layers = 1: **0.2749**
- N_layers = 3: **0.4829**
- N_heads = 4: **0.5451**
- N_heads =16 **0.2399**
- *Conclusion:* This gave me the idea that transformers are not that impactful because they require more data so simple design was best.

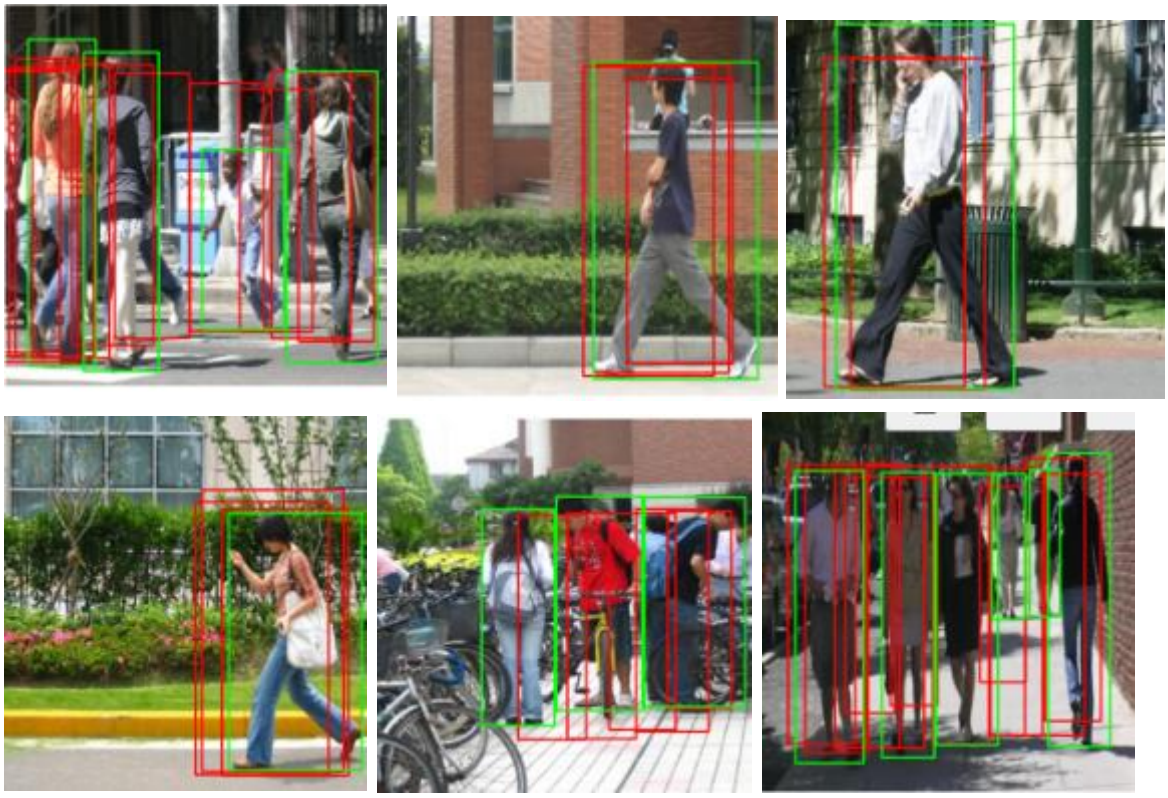
Final Conclusion

Lastly given all I tested Resnet n_queries = 15 n_heads =4 n_layers=2 it gave **Test mAP@0.5: 0.5451** after

V2 net gave **Test mAP@0.5: 0.5035**with same parameters.

Result: This is low query, simple transformer works best.

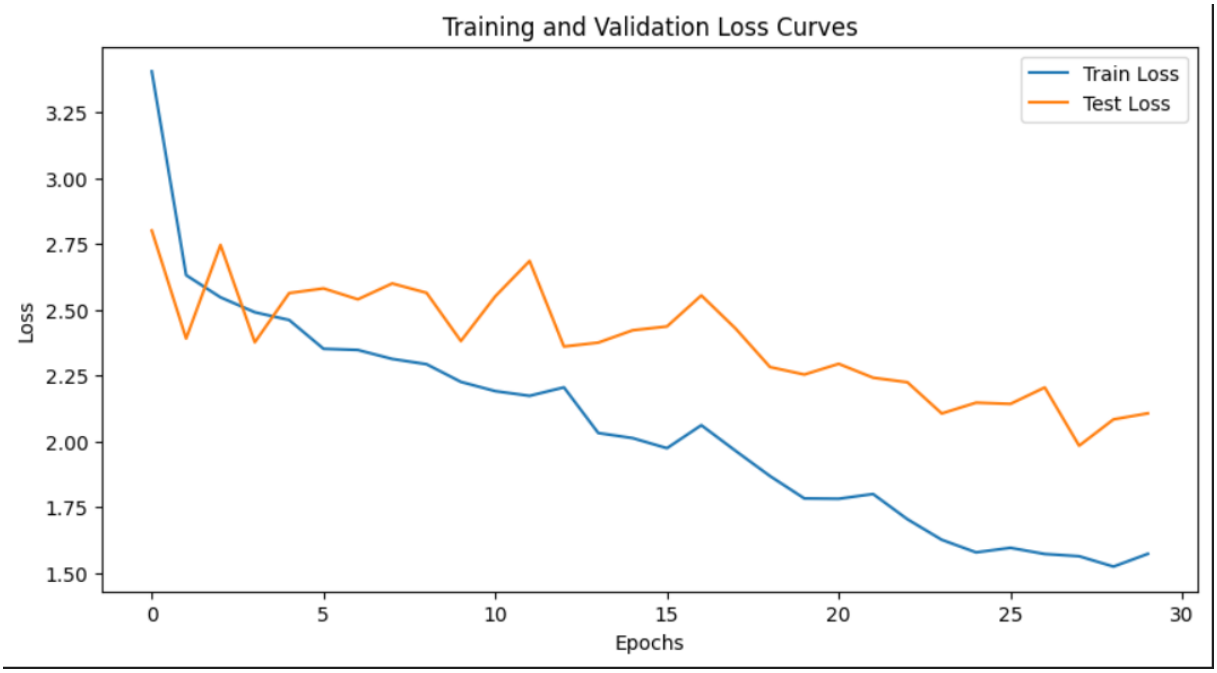
Part 6 : Results (Figures & Tables)



It can predict some simple pictures and predits mostly true boxes.

Base model loss

Last model loss



15 query best according to experiments

Augmented data is better according to experiments

Resnet is best according to experiments

mAP@0.5: 0.5451

Part 7 : Discussion (Accuracy, Robustness, Limitations)

It performs good on simple pictures and non occulated persons. Occlusion makes the box union bigger so it can predict many classes there. This is why when there is a group of people tons of boxes are in the group.

A bigger dataset would be much better so that the model would have been more complex but having a simpler model on small dataset seems better.

Transformers need big data to learn and this dataset is not enough or that. This is why I tried to keep it simple as possible.

Part 8 : Reproducibility Notes

- **Learning Rate:** 0.001 (Best). 0.0001 was too small and 0.01 was too fast.
- **Data:** Augmented (better than original).
- **Backbone:** Resnet (More weight).
- **Transformer:** Simple transformer (4 head, 2 layer), small query of 15.
- **Batch:** 16 batch training.
- **Loss:** eos_coef of 0.2 is best cost class = 1 cost box = 5 cost giou=2
- **Optimization:** I used scaler for faster computing and scheduler for better lr calculation on every model.
- **Random seed 42**
- **4060 laptop GPU**