Hacettepe University

Artificial Intelligence Engineering Department

AIN 433 Computer Vision Lab - Assignment 4

Tiny-DETR Implementation

December 25, 2025

Student:
Mohamed Yahya Mansouri

Student Number:
2220765064

Contents

List of Figures

I Overview

This assignment aims at implementing a compact version of the original DEtection TRansformer (DETR) architecture by Facebook AI.[1] Unlike traditional object detection models that use hand-crafted components such as the Non-Maximum Suppression (as used in assignment 1), the tiny-DETR model follows the DETR's end-to-end training that is more adaptable and data-driven making it conceptually cleaner.

The Tiny-DETR begins with an ImageNet-pretrained CNN based backbone, MobileNetV2 or ResNet18, for spatial feature extraction followed by a 1x1 convolution to reduce the channel dimension of the feature maps to 256 channels. Afterwards, Sinusoidal positional encoding is applied to add information about the position of the pixels. The resulted sequential data is fed to Transformer encoder-decoder producing object queries predictions. Each query has a class logit score along with a bounding box coordinates. Query matching is achieved by Hungarian Matching enforcing one-to-one correspondance between the query predictions and the actual objects. Loss is then calculated using cross-entropy for the labels and L1 for bounding box regression loss.

Unexpectedly, the hardest and one of the most insightful part of the assignment was not the Transformer architecture itself but the set prediction via bipartite matching. The Hungarian cost matrix along with L1 and CE loss has uniquely assigned predictions to ground truth objects. In addition, the use of learned object queries interacting with global visual features through the Transformer's cross-attention mechanism was vital, as it allows the model to reason about object relations and global image context simultaneously.

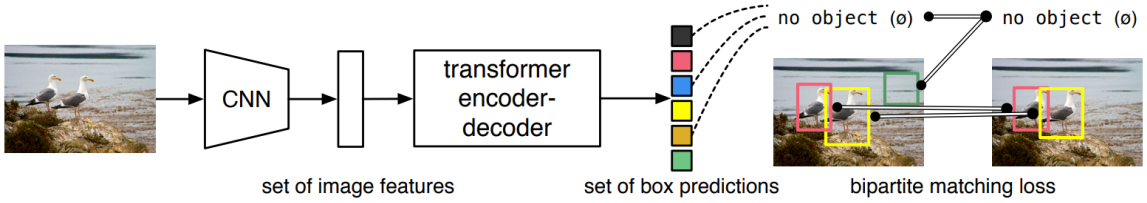


Figure 1: End-to-End DETR Architecture with Bipartite Matching

II Dataset & Setup

II.I Dataset

The dataset used in this assignment is the PennFudan dataset for Pedestrian Detection. It was jointly created by researches from the university of Pennsylvania and Fudan university. It contains 170 images with 345 labeled pedestrians. All of the images have different dimensions. The bounding box coordinate system is: $(x_{min}, y_{min}), (x_{max}, y_{max})$.

The data was split to Training, validation, and testing datasets. There are 118 samples in the training dataset, 25 samples for the validation set, and 27 samples for the testing dataset. During training, the training and validation samples are shuffled on every epoch. Shuffling the testing samples would not matter as this is an objective performance examination. Furthermore, it is easier to align the predicted images to the ground truth ones for comparsion and qualitative analysis.

II.II Preprocessing

All images were resized to the same dimensions ($512x512$) before being fed to the Tiny-DETR model as CNNs and transformers require fixed-shape tensors within a batch. This is due to

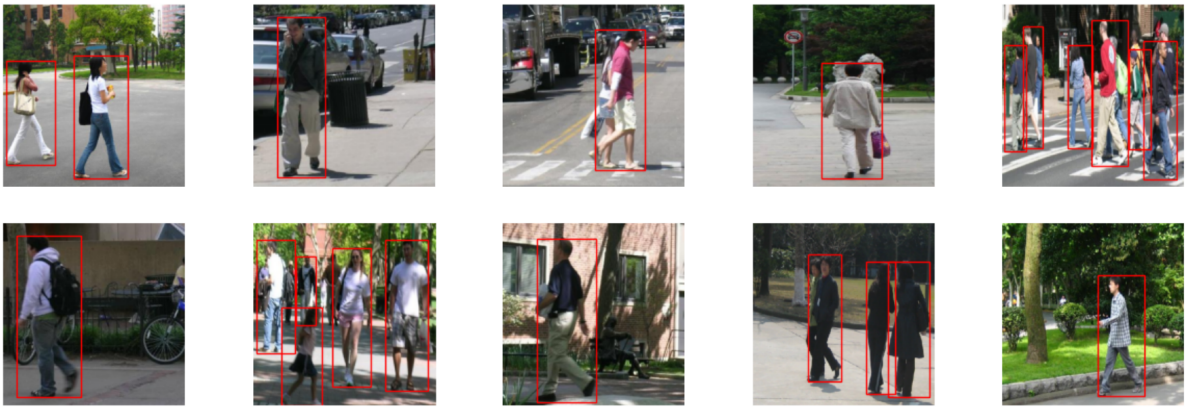


Figure 2: Samples from the dataset.

the fact that all images are stacked into a single tensor and processed efficiently on the GPU. Furthermore, as the transformer requires sequential feature maps, having inconsistent images sizes would lead to mismatched sequence lengths and broken learning mechanism.

Offline data augmentation was used on this dataset. This generates static transformations that are stored before the model training. On the contrast, online data augmentation[6] generates new data samples dynamically during the traning process increasing variation and saving memory by not storing the additional augmented samples. Offline augmentaion was favored due to its simplicity and to the small size of the dataset. Applying spatial augmentation increases labeling efforts as it requires modifying the bounding boxes. Color Jitter and Horizontal flipping were used. Color jitter[2] randomly alters brightness, contrast, saturation, and hue to make the model robust to lightning and color changes. On the other hand, horizontal flipping mirrors images and updates bounding boxes accordingly to improve invariance to object orientation. These two augmentation techniques mitigates overfitting and encourages Tiny-DETR to learn more generalizable features.

The normalization step standardizes both the input images and bounding-box labels to make training stable and compatible with Tiny-DETR. Images are first sclaed down to $[0, 1]$ and then standardized using the mean and standard deviation of image net. This is because the Tiny-DETR backbone is based on the image net pretrained backbones. This two-step process is necessary as those image net statistics are defined on images already normalized to $[0, 1]$. Moreover, the target labels had to be provided in a specific format. The bounding boxes needed to be converted from corner coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ to the (c_x, c_y, w, h) representation and of course normalized by width and height to be in the range of $[0, 1]$

The folder of the assignment is designed as follows:

```
Assignment4/

splits/
    test.txt/                   # Names of the test images
    train.txt/                  # Names of the train images
    val.txt/                    # Names of the validation images

Pedestrian/
    FudanPed<index>.jpg         # Images
    FudanPed<index>.txt/        # Annotations

models
    <model_name>.pth            # Parameters of a model
```

```
report_img/                        # The images of the report

b2220765064.ipynb                   # Assignment Jupyter Notebok File
```

III Model Architecture

III.I Backbone

The tiny-DETR model was trained with two image net based backbones: MobileNetV2 & ResNet18.[1] These convolutional based networks produce spatial feature maps that the transformer entirely relies on for object detection. As a result, they are called backbones. They efficiently capture local patterns such as edges and textures while preserving spatial structure. The quality of these CNNs strongly influence how well the transformer can model object relationships to perform accurate detection.

- **MobileNet V2**: [5] is a convolutional neural network architecture optimized for mobile and embedded vision applications. As a result, it is a lightweight and computationally efficient CNN. It is prefferable for resource-constrained tasks, as in this assignment. MobileNetV2 uses inverted residuals to squeeze data through thin bottlenck[1] layers. Depthwise convolution is another core component as it performs spatial convolution independently over each channel. The final convolutional feature map of MobileNetV2 has 1280 output channels.

- **ResNet18**: [4] convolutional network made of 18 layers. It captures strong visual features through the use of residual skip connections. Similar to assignment 3, these residuals allows information to flow more easily through the network, making it better at capturing meaningful object-related patterns. This is achieved as skip connections lets the input of a block be added directly to its output. In fact, ResNet18 is more computationally expensive than MobileNetV2 but provides more informative feature maps which potentially improves detection performance. The final convolutional feature map of ResNet18 has 512 output channels.

After the backbone, a 1x1 convolution is applied to project the backbone's feature maps into a fixed hidden dimension expected by the transformer. The hidden size used is 256.

III.II Positional Encoding

Positional information[3][8] is essential as the transformer does not detect directions or automatically understand the relative or absolute position of the sequence items (known as tokens). It is widely used in nautral language processing to distinguish between words and capture the strucutre of phrases.Similarily, in object detection, transformers require explicit positional information to understand where features are located within an image. According to the original DETR paper, positional encoding significantly contribute to their object detection model as its removal has led to a decrease in performance, about 7.8AP to the baseline.

Learned lesson: life is more than appearances.

The original DETR uses a sinusoidal positional encoding. The location values are fully independent of the values in the image. Consequently, **they are the same across the batches.**

$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d}}})$$

[1] A layer with fewer channels or neurons than layer before and after it. Its purpose is to squeeze the data into a more compact representation to preserve only the important information.

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d}}})$$

where i is the curren channel, pos is the position index along one spatial axis. $pos = x \in \{0, 1, \ldots, \text{width} - 1\}$ or $pos = y \in \{0, 1, \ldots, \text{height} - 1\}$ and d is the transformer hidden size.

Implementation Explained: for a feature map of size $H \times W$ the method first generates a 4D grid of size $(1 \times 1 \times 1 \times W)$ to encode the horizontal positions and another 4D grid of size $(1 \times 1 \times H \times 1)$ to encode the vertical positions. These settings were derived due to the fact that pos, and so PE_x, is the same across all batches and across all rows of the feature map. Similarily, pos, and so PE_y, is the same across all batches and all columns of the feature map. Half of the channels encode the y-position and the other half encode the x-position. The channels are split so both coordinates are encoded clearly and efficiently since each channel can only store one value. The i values change alongside the channel but are the same across all feature dimensions and batches. As a result, for each coordinate, the above sinusoidal functions are computed with differente frequencies (i) producing alternating sine and cosine values. These PE_x and PE_y matrices are then concatenated to form a full positional matrix of shape (B, C, H, W).

Note: sine and cosine are used because they give each position a unique encoding across the channels alongside different positions. It prevents ambiguity and two positions are extremely unlikely to product the same encoding values.

III.III Transformer Encoder–Decoder Structure

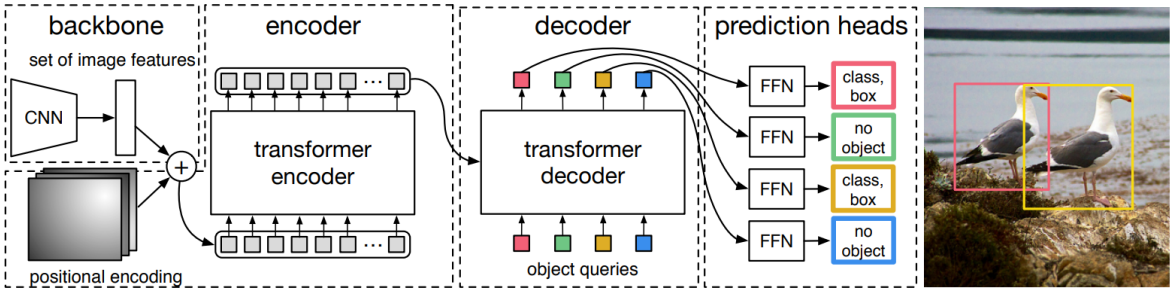


Figure 3: Original DETR (and Tiny-DETR) architecture.

After positional encoding and flattening the 2D feature maps to a 1D vector, the original DETR feeds this data into an encoder transformer to have context-aware representation of the input. The encoder layers enable the spatial locations to interact with each other and exchange informaiton to form relations. Consequently, the encoded output represent a global scene instead of isolated parts. An accurate illustration of this can be the sentence: "Hi how are you?" The encoder takes each word as a token and potentially builds relations between *"you"* and *"are"*. This is achieved inside two submodules called Multi headed attention and FFN module. The multi-headed attention applies an attention mechanism called self-attention. The latter is the key that enables the encoder to associate relations between different spatial locations of the image creating context between close and distant regions in the image. Inside self-attention, the input is fed into 3 different FFN in parallel. Multi-headed attention[8] is reached through applying different self-attention operations in parallel after splitting the input along the channel dimensions allowing the model to attend to different aspects of the input simultaneously. The output shape is the same as the encoder input shape.

In contrast, the decoder is responsible for transforming the global image representation produced by the encoder into a fixed set of object predictions. Instead of processing image tokens directly, the decoder operates on a set of learned object queries, where each query acts as a placeholder for a potential object in the image. Through multiple decoder layers, these queries

are progressively refined into meaningful object representations. Each decoder layer contains three main components: self-attention among object queries, cross-attention between queries and the encoder output, and FFN. According to the research paper, the FFN computes the final prediction by a 3-layer perceptron activated by the ReLU function. This prediction feed-forward network predicts the normalized center coordinates, width, and height of the bounding box. Another FFN takes the (Batch, n_query, hidden_size) refined query and maps it to a class raw logits or class scores. The output shape of the FFN class is $(B \times Q \times N_b + 1)$ for B: batch, Q: number of queries, and N_b: Number of classes in the ground truth image + no-object. The output shape of the FFN box is $(B, \times Q \times 4)$.

III.IV Object Queries and Set-Based Predictions

Object queries are a fixed number of learned vectors that act as slots or potential candidates for each class in the image. It is optimized during training, similar to weights. For a single image, they are of shape: $(Q \times N_b + 1)$ for Q being the number of queries and N_b the number of classes in the ground truth image + no-object. DETR and Tiny-DETR product a fixed set of predictions where each query predicts a distribution over object classes (including no-object) and one bounding box, from which a single class label is selected.

Similar to the research paper, this assignment refers to set prediction as learning to predict an unordered set of object items where their number is fixed but only a subset corresponds to real objects. Matching these queries to the actual ground-truth objects is learned during training rather than using traditional postprocessings such non-maximal suppression.

The number of object queries define the maximum number of objects the model can detect in an image. The number is set to be larger than the typical number of objects in the input image where most of them will be assigned to no-object class. While the paper does not directly talk about the impact of the object queries, drastically decreasing them has led to performance drop in this assignment. This is potentailly explained that using too few queries limits the model to match all ground-truth objects.

III.V Hungarian Matching and Loss Formulation

DETR[1] determines which query corresponds to which actual object using bipartite Hungarian matching which enforces a one-to-one assignment. For each image a cost matrix is constructed between all predicted queries and all ground-truth objects based on class prediction and bounding-box similarity. The hungarian algorithm then finds the optimal matching between each query and one ground-truth object ensuring a one-to-one matching. The no-object class is assigned to the queries that were not matched to any ground-truth object. This one-to-one hungarian-based matching technique removes duplicate detections and discards the need of traditional postprocessings steps. In fact, it is the one of the fundamental components of DETR's object detection architecture.

$$\text{cost} \in \mathbb{R}^{(B \cdot Q) \times \left(\sum_b N_b\right)}$$

Note: flattening was used for computatioaly efficienies. Reshaping predictions from (B, Q) to (B) enables matching costs to be computed in a single vectorized operation instead of nested loops. Afterwards, the matrix is split back.

The output of the Hungarian matching is 2 tuple of indices. The first tuple represents a query list and the second tuple represents a target list. The length of each tuple is the number of objects in the real image.

In the original DETR paper, classification loss, bounding box distance loss, and GIoU[2] loss were the essential components to evaluate the model performance after the hungarian

[2]Generalized intersection over union is a bounding-box similarity metric that provides meaningful gradients even when two boxes do not overlap.

matching. The classification is essential and cannot be discarded as it measures the difference between the predicted label and the ground-truth label. According to the ablation studies of the original paper, the absence of GIoU accounts for the for most of the model performance drop (a difference of 4.4 mAP@0.5). Despite not including it in my work, the implemented Tiny-DETR reached 0.51 mAP@0.5 trained with augmented data and under the pretrained weights of the MobileNetV2 framework. The correct implementation of GIoU would potentially lead to overall model improvement.

Cross-entropy loss is used over all queries with the no-object class down-weighted to avoid class imbalance (since the majority of predictions are no-object class as previously mentioned). For bounding box distance loss, L1 distance between predicted and true boxes are used. The final loss is a weighted sum of the classification and box losses balancing object recognition and localization accuracy.

III.VI Training Under a Small Dataset Regime

Training-transformer based detectors on such small datasets is very challenging because these architectures are data-hungry. These are very complex models where less than a few hundrand samples are not adequate for a good generalizable performance. They either lead to overfitting after a long series of training epochs or underfitting and unstable convergence. In fact all trained models without any data augmentation have not reached the target 0.5 mAP@0.5 score value. This can be potentially explained by transformers relying on learning global relations through attention, which typically requires many diverse examples to generalize well. With few training samples, the model may memorize patterns instead of learning robust object representations. Applying data augmentation reached 0.51 crossing the target score.

However, it is worth saying that this small dataset regime was fairly compensated through using a pretrained backbone along with setting fewer transformer layers than the original DETR model. Along with data augmentation, these choices improve performance.

III.VII Summary of Experimental Findings

IV Training Procedure

The baseline configuration is summarized and justified in Table 1.

To sum up the baseline observations, the training and validation losses drop sharply after a few epochs and then stabilize. A small gap exists between the training and validation loss. However, it does not indicate overfitting as the gap is expected and not meaningful. The validation mAP increases rapidly in the first few epochs and exhibits a fluctuated behavior. Overall, it follows the stabilized behavior of the training and validation loss. The peak validation mAP is 0.35 and testing mAP approximately reached 0.21.

V Experiments & Ablation

V.I Baseline Analysis

- **Training & Validation Loss:**:
  - Both training and validation loss decrease rapidly during the first few epochs. This indicates effective learning rate and stable optimization.
  - After the first 10 epochs, the curves starts to flatten and the validation loss starts to converge. Training continues its steady decrease
  - There is a small gap between validation and training loss. This suggests good generalization and no strong signs of overfitting.

  - The slight oscillations in validation loss toward later epochs are expected in a small-dataset regime and indicate that the model is near convergence.

- **Validation mAP**:
  - The validation map increases in the first few epochs, showing the Tiny-DETR quickly learns meaningful object representations.
  - It reaches its peak at around the 15th epoch with a validation mAP of 0.35.
  - mAP fluctuates than following a smooth monotically improvement. These oscillations potentially reflect to the sensitive nature of the metric used across individual samples.
  - Overall, mAP aligns with the general behavior of the of the loss curves.

- **Testing mAP**: 0.21

V.II Depth Impact

- **Training & Validation Loss:**
  - Both training and validation loss for depth-1 decrease rapidly during the early epochs and converge smoothly. There is a small gap between the two curves. As a result, there are no clea signs of overfitting. Compared to the baseline model, the depth-1 model converged more quickly. Using more epochs in the baseline model was necessary to account for the difference in model complexity, as there will be inconsistency in model comparison. This assumption is actually verified by these observations, as the depth 1 configuration converged faster even with less epochs. Training for more would risk overfitting. With the 1 depth configuration, the testing mAP decreased compared to the baseline which is expected.
  - Dpeth-3 also demonstrates a rapid-decrease in both training and validation loss with minor fluctations in the middle of the training process. This model was trained on more epochs than previous models due to its higher complexity. Interestingly, there is a bigger (still small) gap between the validation and training loss compared to the depth-1 model. This indicates the model's increased tendency to overfitting. This is also observed in the training loss value as it is the lowest compared to previous models. In this case, the most plausible explanation lies in the number of epochs as the model complexity was higher than the previous not-overfitted models and the data is the same.

- **Validation mAP:**
  - There validation mAP has an overall increasing behavior peaking at epoch 14 with a value of 0.31. As seen before, there are fluctuations in the model, but they are expected in small-dataset regime. Relative to the baseline, this model achieves comparable similar validation mAP with better convergence patterns at the end of training.
  - The validation mAP shows noticeable fluctuations across epochs, occasionally reaching values around 0.35–0.36, without forming a consistent upward trend. This behavior reflects unstable learning dynamics and a high sensitivity to individual samples, which commonly arises when the model capacity is disproportionate to the available training data.

- **Testing mAP:** depth-1: 0.17 — depth-2: 0.18

- The training and validation loss shows that a shallower transformer was able to learn well from the given training data.In fact, the validation mAP reached a peak of approximately 0.32 mAP. However, the testing mAP shows that both the current model complexity and the data scarcity strongly affected the Tiny-DETR's generalizability and limited its ability to consistently transfer learned representations to unseen data.

V.III Data Augmentation Impact

- **Training & Validation Loss:**
  - Same fast drop behavior is exhibited.
  - Both training and validation loss are their lowest values at the end of training compared to the previous models.
  - The gap between training and validation loss remains relatively small but closely related to the depth-3 model.
  - This model indicates better learning ability due to its smoother loss decrease but a tendency to overfitting higher than the baseline.
- **Validation mAP:**
  - Validation mAP shows a clear upward trend, reaching values close to 0.50, which is significantly higher than in the non-augmented experiments.
  - Although minor fluctuations are still present, the overall stability of the mAP curve suggests that data augmentation enhances robustness to image variability.
  - These results indicate that the presented data augmentation techniques effectively compensates for data scarcity, allong the model to learn
  - These results indicate that data augmentation effectively compensates for data scarcity, allowing the model to learn representations that generalize better to unseen data.
- **Testing mAP:** 0.51

V.IV Backbone Comparison

Note: a smaller backbone learning was used since the original DETR paper [1] observed that

> having the backbone learning rate roughly an order of magnitude smaller than the rest of the network is important to stabilize training, especially in the first few epochs

- **Training & Validation Loss:**
  - The training loss decreases teadily throughout the learning process while the validation loss stabilizes at a higher loss value.
  - There are more mild oscillations in the validation mAP comapred to the MobiletNetV2 augmented model. This behavior indicates that the stronger ResNet18 backbone enables the model to fit the data more aggressively (much lower training loss) resulting in a more noticeable but controlled generalization gap.
- **Validation mAP:**
  - The validation mAP shows an overall increasing pattern.
  - The fluctuations are still present with the ResNet18 backbone. These sudden changes indicate that the learned representations are not yet fully stable and that performance is influenced by the limited diversity of the data.

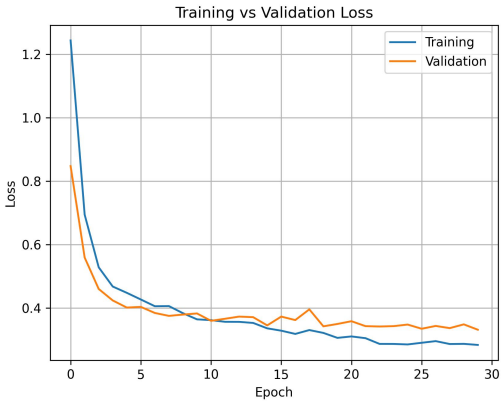
- **Testing mAP:** 0.50
- Both backbone based models had very similar results in training, validation, and testing. However, the validation mAP of the MobileNetV2 was more "explosive" and unstable than the one exhibited in the ResNet18. In fact, the validation mAP of the latter can be safely divided into two regions: one for the start of the learning process and another for the end of the learning process. The MobileNetV2 is desgined for extreme efficiency using depthwise seperable convolutions which significantly reduces the number of parameters and computational cost. However, this parameter reduction makes it more sensitive to data variability, especially after data augmentation. Ultimately, while ResNet18 offers a more stable and consistent learning process, MobileNetV2's lighter weight allowed it to generalize slightly better in the long term.

V.V Query Count Ablation

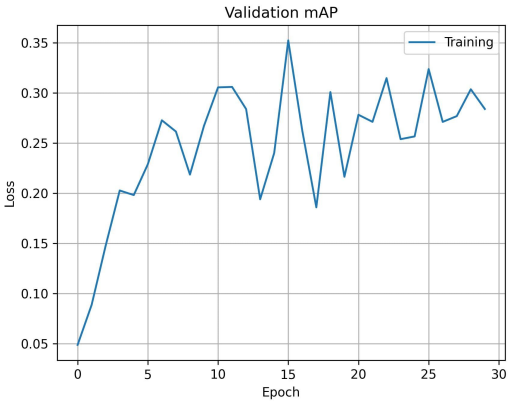
- **Training & Validation Loss:**
  - The training curve shows the classic pattern of converigng near 0.2 loss value. The validation loss follows closely but starts oscillating between 0.4 and 0.3 indicating. There is a small stable generalization gap.
- **Validation mAP:**
  - The validation mAP shows strong upward progression overall, increasing from roughly 0.17 to peaks above 0.50. It is clear that the curve exhibits some osciallations in the range of 0.4 - 0.5 similar to the 100 query based data augmented Tiny-DETR model.
  - In contrast, the model with 25 queries demonstrates a more stable and smoother learning trajectory. The validation mAP steadily increases from around 0.13 and converges near 0.38 - 0.40 with fewer extreme oscillations compared to the 50 query configuration. In fact, their magnitude is smaller. The reduced number of query has led to the most stable learning behavior across all all of the augmented models. However, it has limited the model's ability to represent complex scenes or detect multiple objects effectively.
- **Testing mAP:** Query 50: 0.45 — Query 25: 0.45

Winner Model: The Tiny-DETR model trained with the augmented data, MobileNetV2 backbone, and 100 object queries is the one that performed the best under this assignment constraints and data limitations reaching a mAP of 0.51. Only samples from the baseline and this winner model are shown in the report.

VI Results (Figures & Tables)

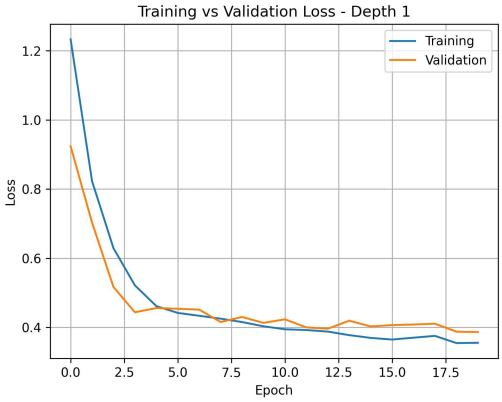


(a) Baselin Training & Validation Loss

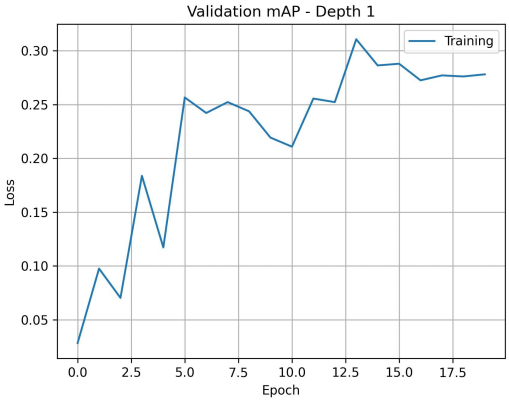


(b) Baseline Validation mAP@0.5 Curve

Figure 4: Baseline Curves

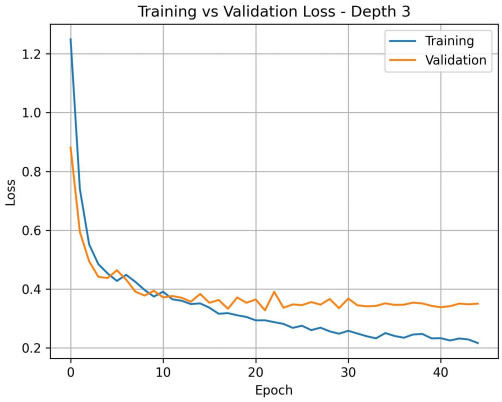


(a) Depth-1 Training & Validation Loss

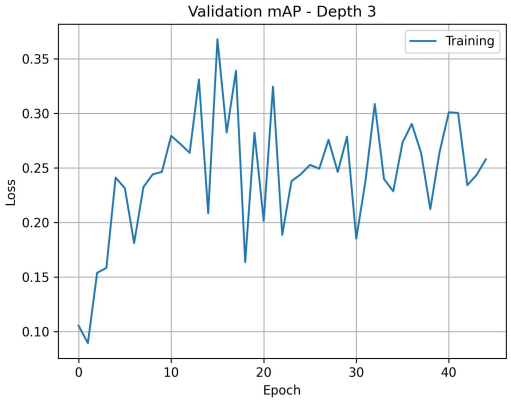


(b) Depth-1 Validation mAP@0.5 Curve

Figure 5: Dpeth-1 Tiny-DETR Curves

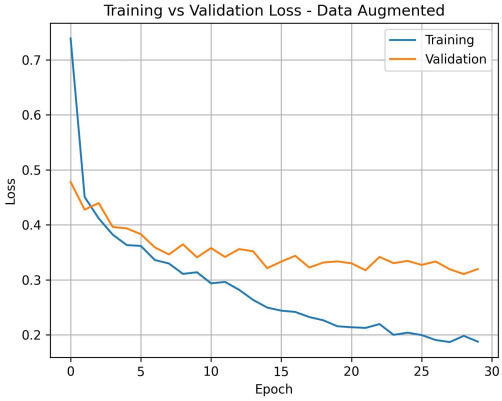


(a) Depth-1 Training & Validation Loss

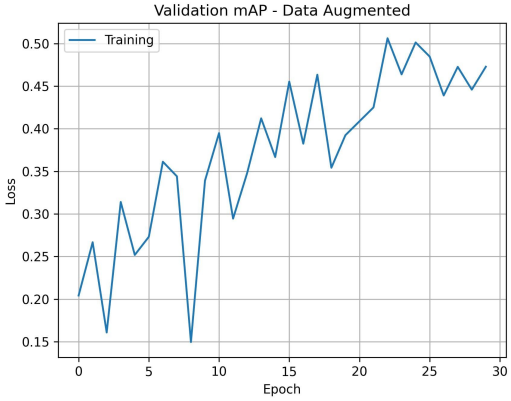


(b) Depth-1 Validation mAP@0.5 Curve

Figure 6: Dpeth-1 Tiny-DETR Curves

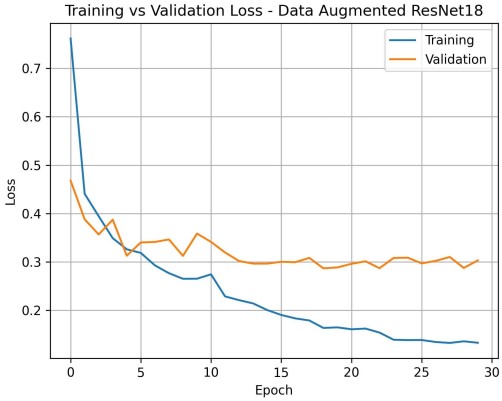


(a) Training & Validation Loss

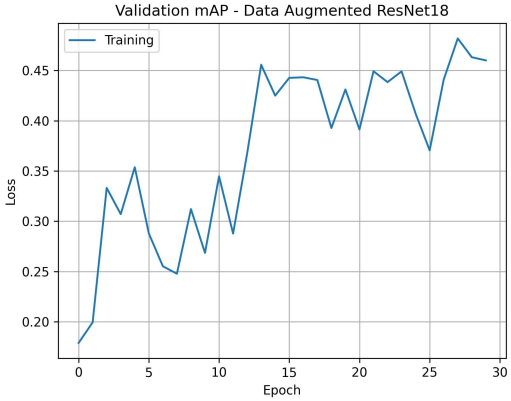


(b) Validation mAP@0.5 Curve

Figure 7: Tiny-DETR with Data Augmentation Curves and MobileNetV2 Backbone

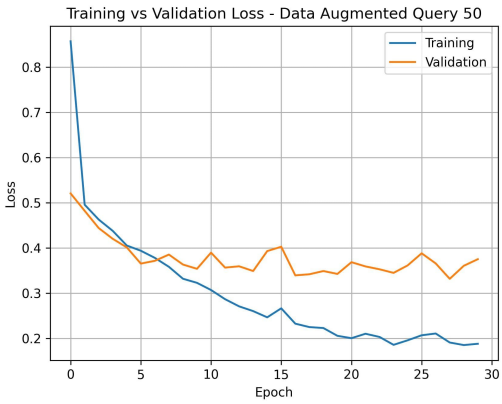


(a) Training & Validation Loss

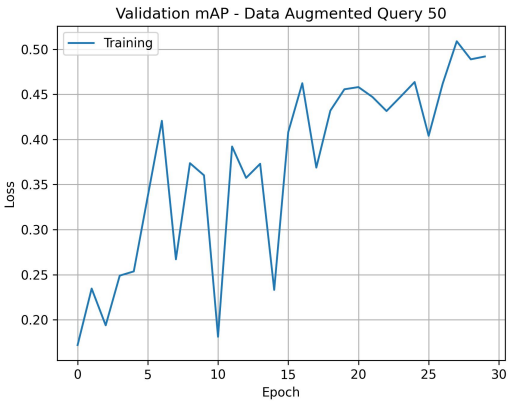


(b) Validation mAP@0.5 Curve

Figure 8: Tiny-DETR with Data Augmentation and ResNet18 Backbone Curves

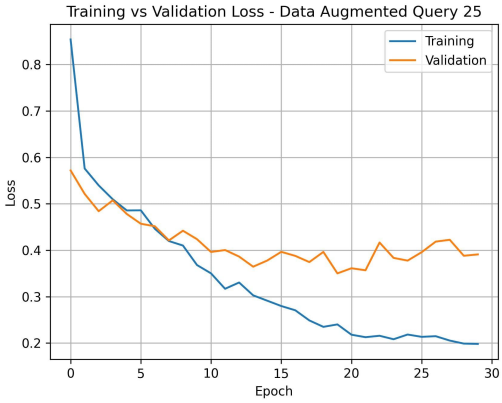


(a) Training & Validation Loss

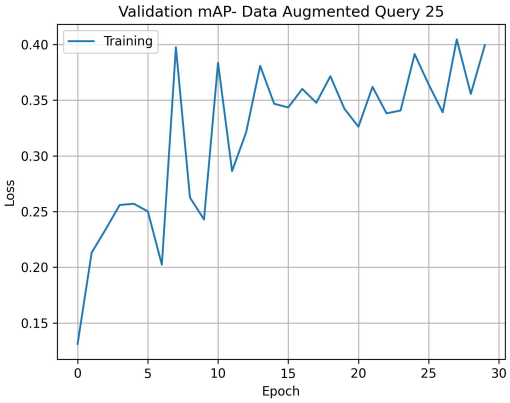


(b) Validation mAP@0.5 Curve

Figure 9: Tiny-DETR with Data Augmentation and 50 Object Queries



(a) Training & Validation Loss



(b) Validation mAP@0.5 Curve

Figure 10: Tiny-DETR with Data Augmentation and 25 Object Queries

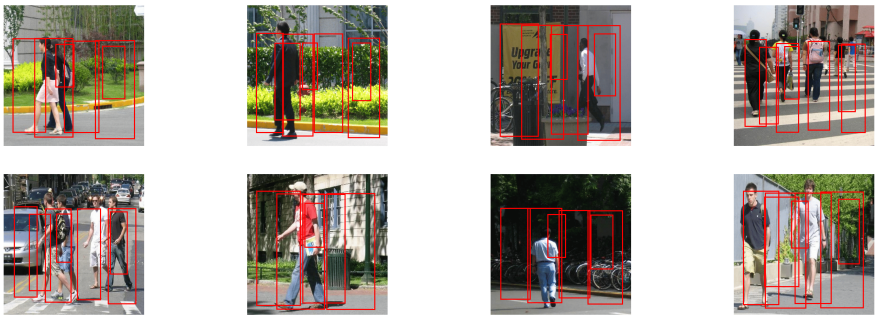


Figure 11: 8 Predicted Samples from the Baseline Model.

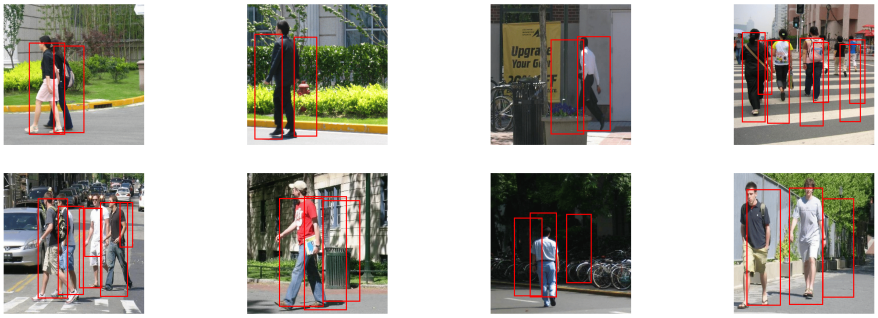


Figure 12: 8 Predicted Samples from the Best Winner Model.

VII Discussion (Accuracy, Robustness, Limitations)

The performance of the Tiny-DETR model varies noticeably depending on scene complexity and object characteristics. According to the 8 sample predicted pictures, the model performed well in images containing different number of pedestrians. However, those images with less and clearly visible pedestrians have less overlapping bounding boxes such exhibited in the second and third image in first row and the third and fourth images in the last row. In these cases, the learned object queries are able to attend to distinct spatial regions, and the Hungarian matching potentially enforces more accurate one-to-one correspondance between predictions and actual ground-truth objects. The model was successfull with one occlusion case in the first image in row one. Where both of the pedestrians were detected. On the other hand, Tiny-DETR showed limitations when handling crowded scenes such as the last image of the first row and the first image of the second row. While the pedestrians were detected, some unecessary overlapping bounding boxes were appended to the image. Aside from this, the model have also struggled with very far away pedestrians. Small pedestrian occupy fewer pixels in the backbone feature maps, making precise localization difficult for the transformer architecture.

These failure cases are expected in a small-dataset regime, where the model struggles to attain a global performance and have fine-grained adjusted spatial configurations.

The limited dataset has one of the most impactful effect on both accuracy and model robustness. With only 118 training images, the transformer-based architecture has clearly exhibited sensitivity to individual samples which is translated to unstable fluctuations in the validation mAP across different epochs. Offline data augmentation partially mitigated this problem by increasing input diversity and invariance to color and spatial features. Yet, augmenation alone

cannot replace the benefits of real and larger diverse dataset. Most importantly, having a very short number of validation and testing samples is not adequate to fully trust and rely on the model performance and the exhibited metrics. Despite shuffling, random seeding, pretrained backbones, and previously tested hyperparameters (from the original DETR), it was challenging at first to have reporducible results and draw consistent conclusions. Statistical tests are necessary to assess the significance between the different Tiny-DETR configurations such as the Bonferroni-Dunn test[7] for pairwise comparison between the best Tiny-DETR and the other models where they are considered singificantly different if the average ranking difference exceeds a critical difference (CD).

VIII Reproducibility Notes

- **Hyperparameters:** *All general hyperparameters are included in Table 2.*
- **Random seeds:** To ensure reproducible result across multiple runs, random seed was set to 40 and was applied to Python's built-in random module, Numpy, and Pytorch (CPU and CUDA). Additionally, deterministic behavior was enforced in cuDNN by disabling benchmarking and enabling deterministic operations. With these settings, rerunning the code yields consistent and repeatable results.

```
seed = 40
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

- **Hardware Used:** This assignmend was trained on a Google Collab T4 GPU with 15GB of VRAM. However, the full memory capacity was not utilized. With a batch size of 16, the GPU memory usage averaged around 7GB. The system RAM was 51GB, but this assignment did not exceed 7GB. With a smaller batch size (8 or 4), this work can be run on a 4-6GB GPU, althrough training would take longer. On average, model training took roughly 5 minutes.
- **exact command to run training/evaluation**: Only a jupyter notebook was provided. As a result, no exact commands are needed to run training or the evaluation. It is worth noting that all models are saved. The code to save and load models is also provided in the jupyter notebook.

References

[1] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV]. URL: https://arxiv.org/abs/2005.12872.

[2] PyTorch Documentation. *ColorJitter*. 2017. URL: https://docs.pytorch.org/vision/main/generated/torchvision.transforms.ColorJitter.html.

[3] GeeksforGeeks. *Positional Encoding in Transformers*. 2025. URL: https://www.geeksforgeeks.org/nlp/positional-encoding-in-transformers/.

[4] GeeksforGeeks. *ResNet18 from Scratch Using PyTorch*. 2025. URL: https://www.geeksforgeeks.org/deep-learning/resnet18-from-scratch-using-pytorch/.

[5] GeeksforGeeks. *What Is Mobilenet V2?* 2025. URL: https://www.geeksforgeeks.org/computer-vision/what-is-mobilenet-v2/.

[6] NVIDIA. *OOffline Data Augmentation*. 2025. URL: https://docs.nvidia.com/tao/tao-toolkit/latest/text/data_services/augment.html.

[7] Yuntao You et al. "Data Transformation-Driven Fuzzy Clustering Neural Network With Layerwise and End-to-End Training". In: *IEEE Transactions on Fuzzy Systems* 33.10 (2025), pp. 3542–3556. DOI: 10.1109/TFUZZ.2025.3596790.

[8] YouTube. *Illustrated Guide to Transformers Neural Network: A step by step explanation*. 2020. URL: https://www.youtube.com/watch?v=4Bdc55j80l8.

Table 1: Baseline Tiny-DETR Configuration

Hyperparameter	**Value**	**Justification**
Backbone	MobileNetV2 (pretrained)	Lightweight architecture with pretrained features to improve convergence on a small dataset
Hidden size	256	Provides sufficient model capacity while reducing overfitting risk
Encoder layers	2	Reduced depth stabilizes training under limited data
Decoder layers	2	Enables query refinement with minimal computational overhead
Number of object queries	100	Exceeds typical object count, allowing unmatched queries to represent no-object (followed Original DETR)
Number of classes	1 (+ no-object)	Task-specific setup with explicit handling of background
Backbone learning rate	1×10^{-5}	Preserves pretrained backbone features during fine-tuning ((followed Original DETR))
Transformer learning rate	1×10^{-4}	Allows faster adaptation of transformer layers ((followed Original DETR))
Optimizer	Adam	Stable optimization for transformer-based architectures
Learning rate scheduler	Cosine Annealing	Smooth learning rate decay for improved convergence
Batch size	16	Balances gradient stability and memory constraints
Classification weight α	0.1	Down-weights no-object class to address class imbalance ((followed Original DETR))
Box loss weight λ	5	Emphasizes accurate bounding-box regression ((followed Original DETR))
Gradient clipping	0.5	Prevents exploding gradients during training
Epochs	30	Sufficient training duration for a small dataset

Table 2: Model Performance Summary

Model Name	**Configuration**	**Testing mAP**	**Stability**	**Limitations**
Baseline	MobileNetV2, depth-2, no data augmentation	0.21	Moderate stability with validation mAP fluctuations	Limited generalization due to small dataset and lack of augmentation
Depth-1	Reduced encoder–decoder depth	0.17	Better stability and smoother convergence	Limited representational capacity
Depth-3	Increased encoder–decoder depth	0.18	Low stability with extreme oscillatory behavior	Overfitting caused by excessive epochs and too complex for the current amount of data.
Data-Augmented Tiny-DETR	Color jitter and horizontal flipping applied	0.51	Much better stability with consistent performance improvement	Offline augmentation limits variability
ResNet18 Backbone	ResNet18 backbone instead	0.50	Stable learning with controlled generalization gap	Higher computational cost
Query-50 Tiny-DETR	50 Object Queries	0.45	Moderate	Sensitive to sample variation
Query-25 Tiny-DETR	25 Object Queries	0.45	Moderate Stability	Limited capacity for complex scenes
Final Selected Model	Data Augmented Tiny-DETR	0.51	Balanced stability and accuracy	No GIoU loss and limited dataset size

Table 3: Training Hyperparameters and Model Configuration

Hyperparameter	**Value**	**Description**
Epochs	30 / 45	Number of full passes over the training dataset
Batch Size	16	Number of samples fed per training iteration
Backbone Learning Rate	1×10^{-5}	Learning rate for the CNN backbone parameters
Transformer Learning Rate	1×10^{-4}	Learning rate for the transformer encoder and decoder
Backbone	MobileNetV2	CNN used for feature extraction
Hidden Size	256	Dimensionality of the transformer embeddings
Encoder Layers	2	Number of transformer encoder layers
Decoder Layers	2	Number of transformer decoder layers
Number of Queries	100	Number of object queries in the Tiny-DETR decoder
Number of Classes	1	Number of object categories (without background)
λ	5	Weight of the bounding box regression loss
α	0.1	Loss weight for the no-object class
Filter Coefficient	0.2	Coefficient used to filter low-confidence predictions