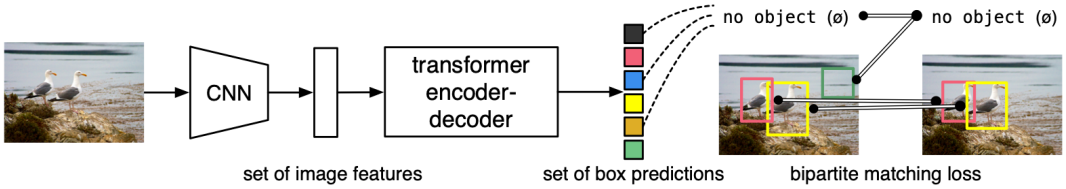
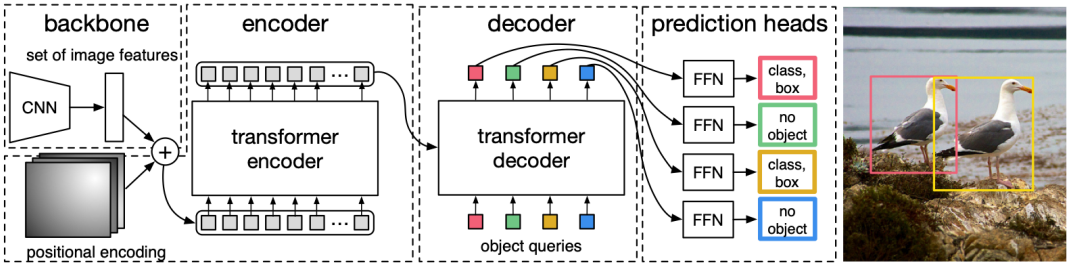
# CVPDL

# Homework #1 Object Detection

## 1. Draw the architecture of your object detecto

The model architecture depicted in the images is for DETR (Detection Transformer), a transformer-based object detection model introduced by Facebook AI Research. Here's a detailed breakdown of the architecture and its components:




## The DETR Architecture

**Backbone (CNN) for Feature Extraction:**

The first step of the architecture involves a Convolutional Neural Network (CNN) backbone (typically ResNet-50 or ResNet-101). The CNN processes the input image and extracts image features. These features form the basis for the next steps.

The output from the CNN is a set of feature maps that represents various patterns in the image such as textures, edges, and object parts.

**Positional Encoding:**

Since transformers are permutation-invariant (i.e., they do not understand the spatial relationship between pixels by default), positional encodings are added to the CNN-extracted feature maps. These encodings help the transformer to understand the spatial structure of the image.

These encodings are crucial to maintaining the relationship between object locations in the image.

**Transformer Encoder-Decoder:**

The DETR model uses a Transformer encoder-decoder architecture, originally designed for natural language processing tasks, to model the relationships between image features.

**Encoder:**

The transformer encoder processes the feature maps (with positional encoding added) and captures long-range dependencies and spatial relationships in the image.

**Decoder:**

The decoder is provided with object queries (learned positional embeddings) and predicts the object class and bounding boxes. Each query corresponds to one object in the image.

**Prediction Heads:**

After the decoder produces object representations, prediction heads are used to make the final predictions. Each prediction head is a simple feedforward neural network (FFN).

**The output consists of:**

Object class: A probability distribution over possible object classes (or "no object" in the case of background).
Bounding boxes: A set of coordinates defining the bounding box around the detected object.

**Bipartite Matching Loss:**

DETR uses a special loss function based on bipartite matching. It assigns a predicted object box to each ground truth box in a one-to-one matching scheme. This is done by minimizing the Hungarian loss, which considers both the class prediction loss and the bounding box regression loss.
By doing so, DETR avoids the need for anchor boxes or non-maximum suppression (NMS), which are commonly used in traditional object detectors.

# DETR Training and Inference Workflow

**Training:**

During training, DETR learns to predict both the class and the bounding boxes for the objects in an image.
It uses a set of object queries as inputs to the transformer decoder, and these queries allow it to predict objects present in the image.
The bipartite matching loss ensures that there is a unique correspondence between predicted and ground-truth objects.

**Inference:**

At inference time, the model takes an input image, processes it through the CNN backbone, and then passes the extracted features through the transformer. The model outputs a set of object classes and their corresponding bounding boxes.

## Applications of DETR

DETR is used for object detection tasks, where the goal is to identify objects in images and localize them with bounding boxes. Some applications include:
Autonomous vehicles (detecting pedestrians, vehicles, road signs, etc.)
Surveillance and security systems
Medical imaging (detecting abnormalities or specific regions in scans)
Retail (detecting products in stores)

## 2. Implement details

### Loss Function

The loss function setup in the task is designed to support the training of the DETR model, and it includes various components:

loss_ce: Cross-entropy loss for classification, which measures how well the model is predicting object classes.

loss_bbox: L1 loss for bounding box regression, measuring the error between the predicted and ground-truth bounding boxes.

loss_giou: Generalized IoU loss, which is used to evaluate the overlap between the predicted and ground-truth bounding boxes.

Auxiliary losses (aux_loss) are used, meaning losses are calculated at each layer of the Transformer. This multi-stage loss calculation helps fine-tune the model at different levels of the network.

### Parameter Settings

**Learning Rate:**

--lr 1e-4: The global learning rate is set to 1e-4, which will be decayed using a learning rate scheduler (lr_scheduler).

--lr_backbone 1e-5: A lower learning rate is set for the backbone (ResNet50 in this case) to prevent over-updating the pre-trained weights.

Additional fine-tuning was performed with lr 1e-5 to allow more careful updates during training, especially when fine-tuning with lower batch sizes.

**Batch Size:**

--batch_size 8: The batch size is set to 8, meaning eight images are processed per batch. This can be adjusted depending on the available GPU memory.

Additional fine-tuning was also done with batch_size 1, allowing the model to process one image per batch to fit into memory when using lower learning rates.

**Optimizer:**

The optimizer used is AdamW, which is well-suited for this type of task. A weight decay of 1e-4 is applied to prevent overfitting.

**Weight Decay:**

--weight_decay 1e-4: This L2 regularization term helps avoid overfitting by penalizing large weights during training.

**Epochs:**

Epochs parameter controls the total number of times the model will go through the entire training dataset.

--epochs 150: This parameter sets the number of epochs to 150, meaning the model will train over the entire dataset 150 times. Training for a larger number of epochs typically allows the model to converge better, but it can also increase the risk of overfitting if not

handled carefully.

Additionally, the model was fine-tuned with 10 epochs and 50 epochs and 50epochs+20epochs to test its performance with fewer training iterations.

**Gradient Clipping:**

--clip_max_norm 0.1: This sets the maximum norm for gradient clipping to avoid exploding gradients, ensuring stable training.

**Model Architecture:**

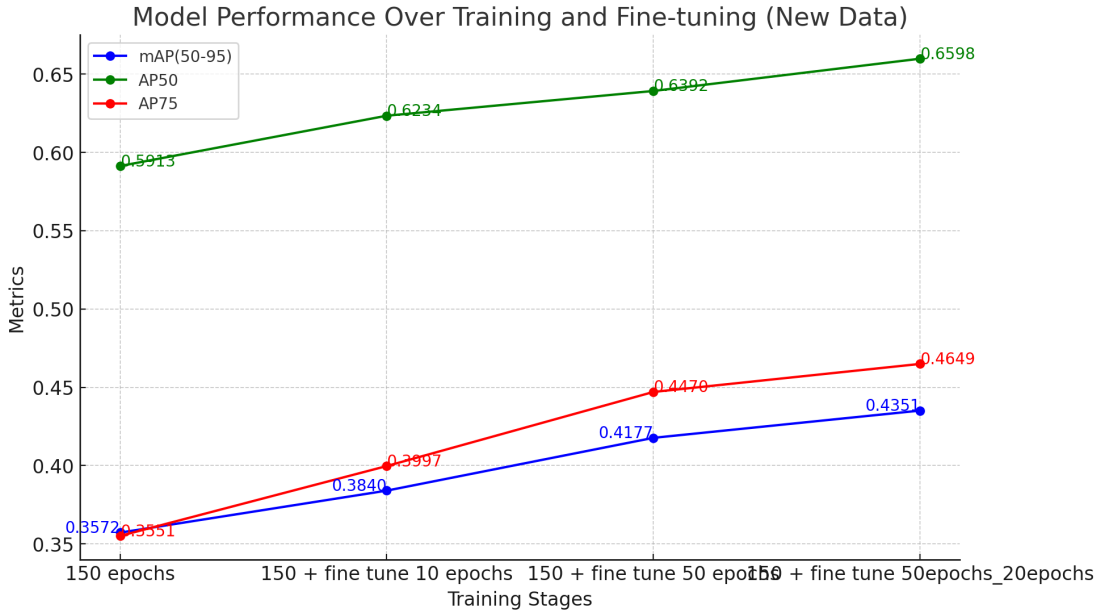
The code uses ResNet50 as the backbone and a Transformer structure with 6 encoder and 6 decoder layers (enc_layers=6, dec_layers=6).

**Matcher Cost:**

Matching costs for Hungarian matching in DETR are defined as: classification cost (set_cost_class=1), L1 box cost (set_cost_bbox=5), and generalized IoU cost (set_cost_giou=2). These determine how predicted boxes are matched with ground truth during optimization.

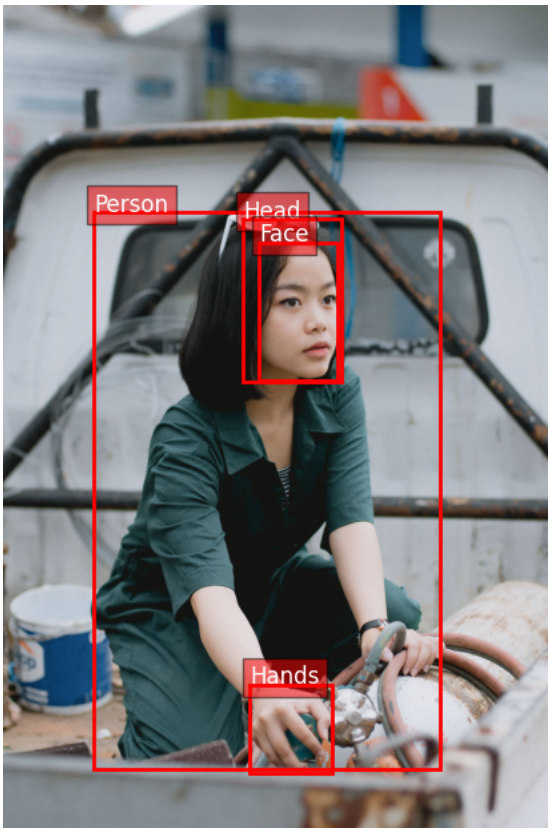
## 3. Table of your performance for validation set (mAP, $AP_{50}$, $AP_{75}$

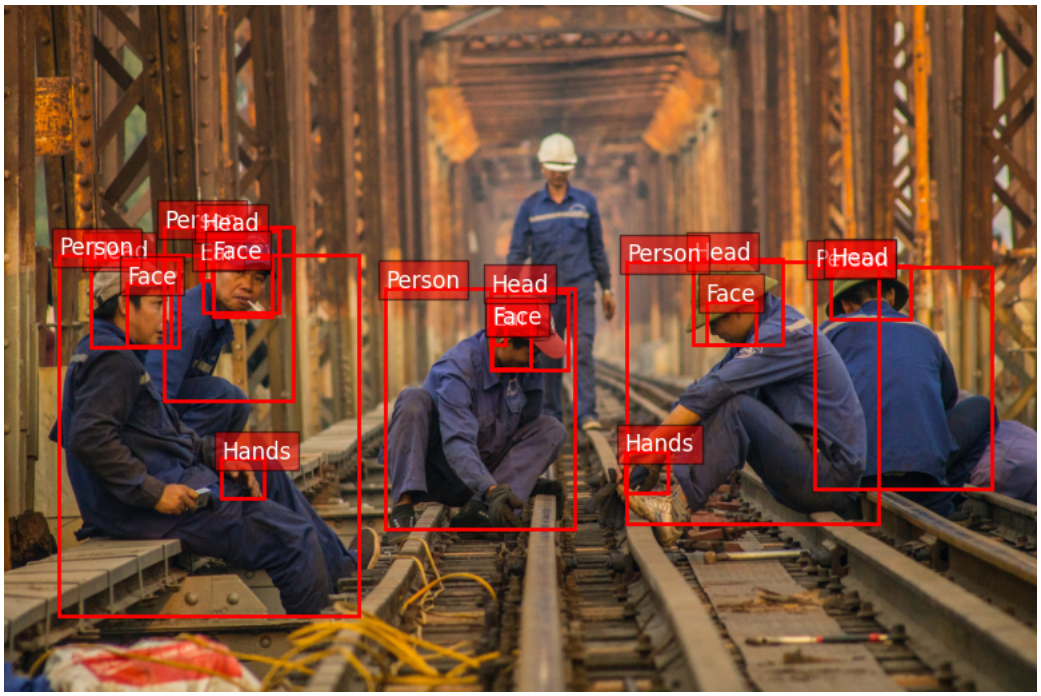
| Training Stage | mAP(50-95) | AP50 | AP75 |
|---|---|---|---|
| **150 epochs** | 0.3572 | 0.5913 | 0.3551 |
| **150 + fine tune 10 epochs** | 0.384 | 0.6234 | 0.3997 |
| **150 + fine tune 50 epochs** | 0.4177 | 0.6392 | 0.447 |
| **150 + fine tune 50epochs_20epochs** | 0.4351 | 0.6598 | 0.4649 |



## 4. Visualization and discussion

Demonstrate the detection results, discussion for the long tail effect, etc.

The Long Tail Effect refers to the phenomenon where a small number of categories or items appear very frequently, while the majority of categories or items occur much less often. This is common in tasks like object detection, classification, and recommendation systems. The long tail effect often impacts model performance, especially regarding the accuracy and recall for rare or less frequent categories.

**Why does the Long Tail Effect Occur?**

**Data Imbalance:** In training data, frequent categories (e.g., "Person") appear much more often than rare categories (e.g., "Glasses"). As a result, models tend to learn the features of common categories better while underperforming on rare ones.

**Natural Distribution:** In the real world, objects or events follow a long-tail distribution. For example, "Person" on the street appears much more frequently than a "Stop Sign"; in e-commerce, a few popular items sell far more than thousands of niche products.

**Annotation Difficulty:** Rare categories can be harder or more expensive to label. For example, annotating rare species or specialized objects may require more expertise, further exacerbating the data scarcity for these categories.

**How to Mitigate the Long Tail Effect?**

**Data Augmentation:** Apply augmentation techniques to increase the data volume for rare classes by rotating, scaling, or flipping images to generate more training samples for those categories.

**Resampling:** Use oversampling for rare classes or undersampling for common classes to balance the training data. This helps the model learn the features of underrepresented categories.

**Cost-sensitive Learning:** Assign higher error penalties (weights) for rare categories during training, making the model pay more attention to these categories and improving their detection.

**Adjusted Evaluation Metrics:** Use weighted accuracy, recall, or F1-score to reflect the imbalanced class distribution. These metrics can provide a more accurate reflection of the model's performance across all classes.

**Meta-learning Techniques:** Recent techniques such as few-shot learning or zero-shot learning can help the model better understand rare categories, even when there is limited training data available for these classes.

**Why is the Long Tail Effect Important?**

In many applications (e.g., autonomous driving, medical diagnostics, recommendation systems), detecting rare objects or events is crucial, even if they occur infrequently. For example, while "Person" is a common category in autonomous driving, detecting rare objects like "Stop Signs" or "Children's Toys" is equally important, as missing them could result in severe consequences.