Object Detection using Transformers and CNNs

A Drone Case Study

Eduardo Dixo, Senior Data Scientist @ Continental

Talk Outline

- Background
- Faster R-CNN
- RetinaNet
- Transformers
- Detection Transformer (DETR)
- Conclusion

Background

Task: Object Detection applied to cars

VisDrone 6k training images and 500 validation images with different:

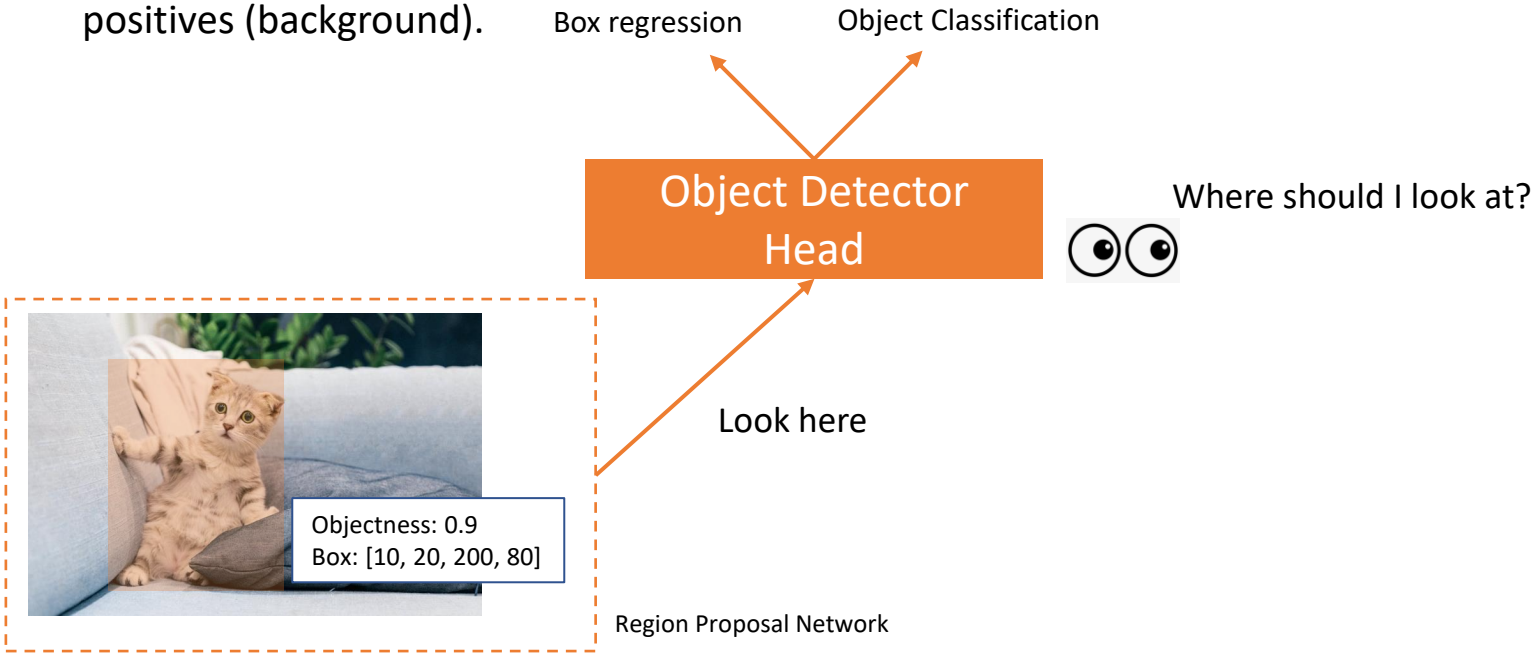
- Weather and lighting conditions
- Object Density
- Scale
- Fast-motion
- Truncation
- Occlusion

Applications:

Road Safety / Traffic Monitoring / Driving assistance

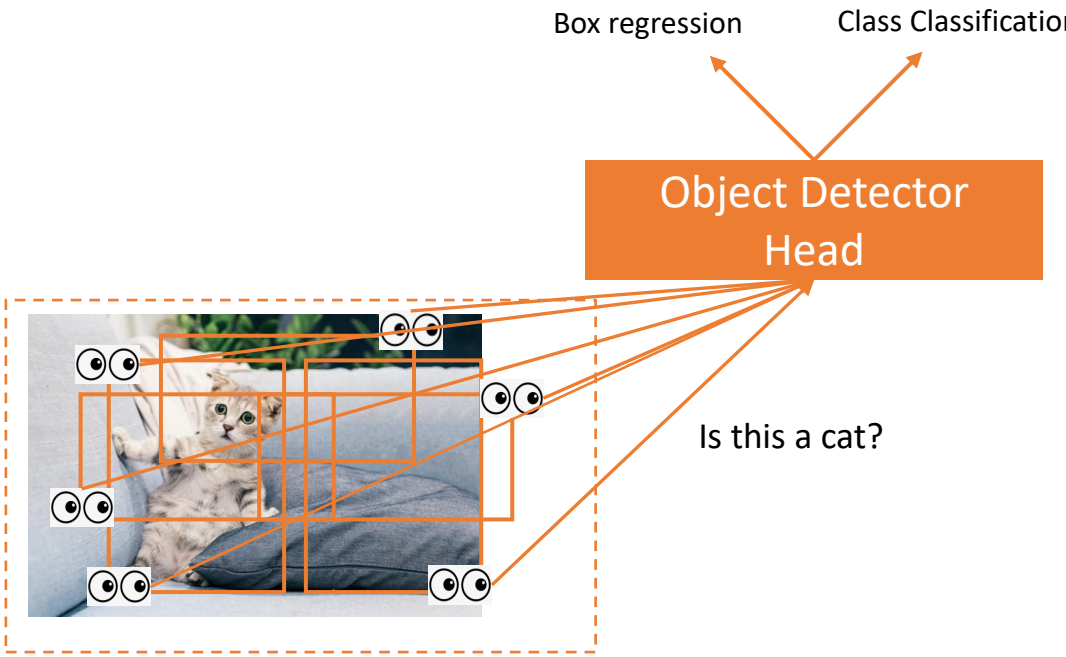
Object Detectors: One-stage vs. Two-Stage

A two-stage detector classifier would like to know where he should attend to in the original picture to find the objects. This is typically done by a Region Proposal Network that select high-confidence region proposals, eliminating many of the false positives (background).



Object Detectors: One-stage vs. Two-Stage

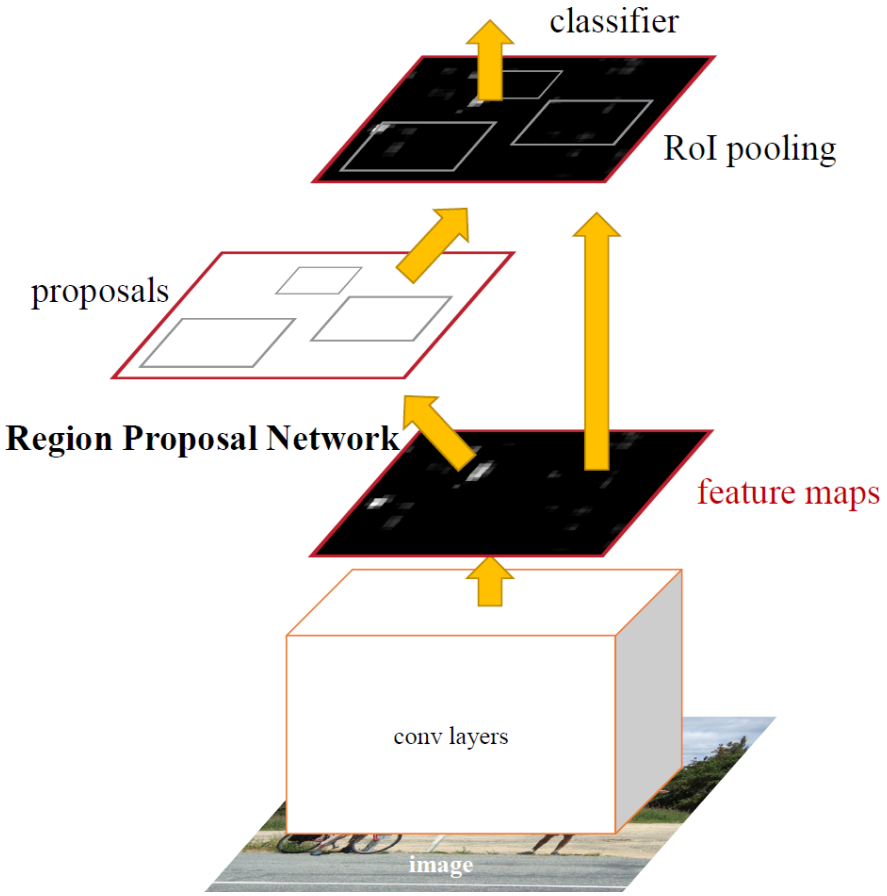
A one-stage detector processes a dense sampling of possible object locations (much larger than two-stage detectors) and learn the class labels and bounding box coordinates.



Faster R-CNN

Faster R-CNN is a two-stage detector that combines two modules:

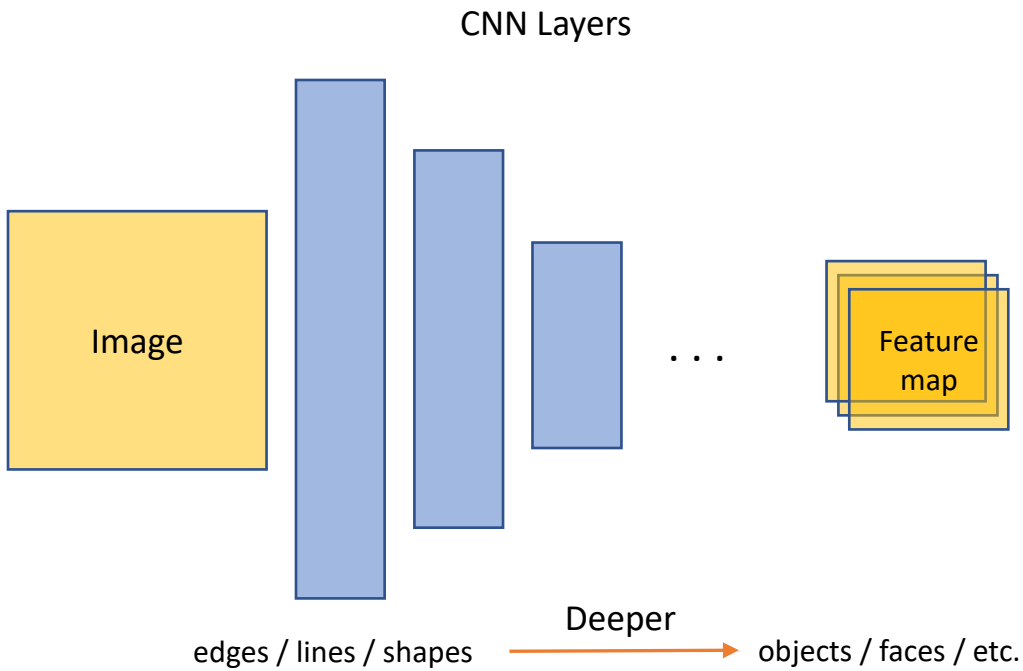
- A Region Proposal Network (RPN)
- A Head Classifier for bounding box regression and object classification

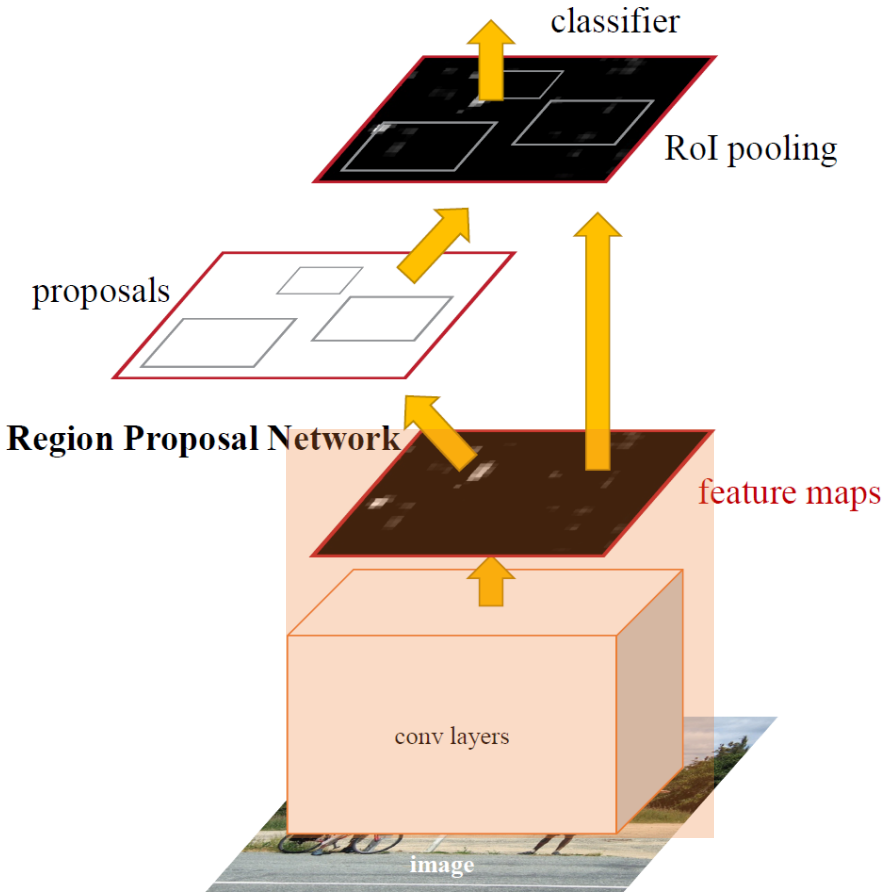


Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks

Faster R-CNN Backbone

The Backbone goal is to extract high-level semantic feature maps from the image.

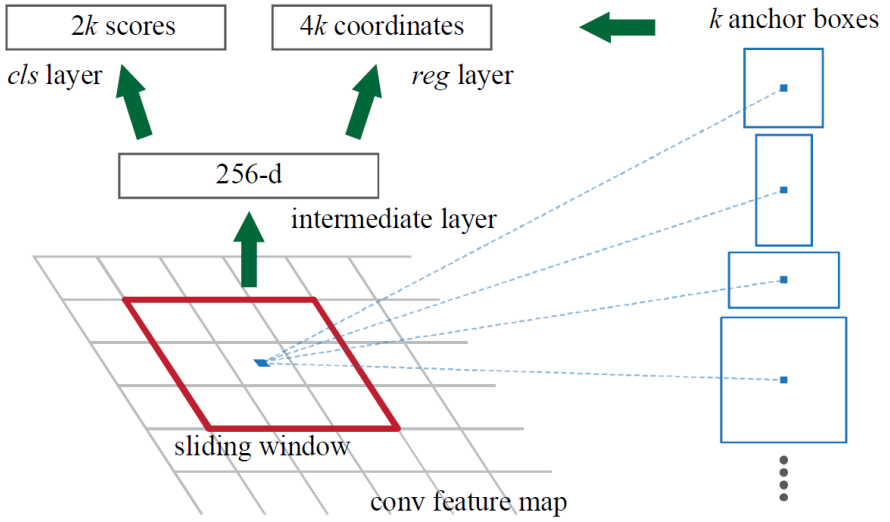




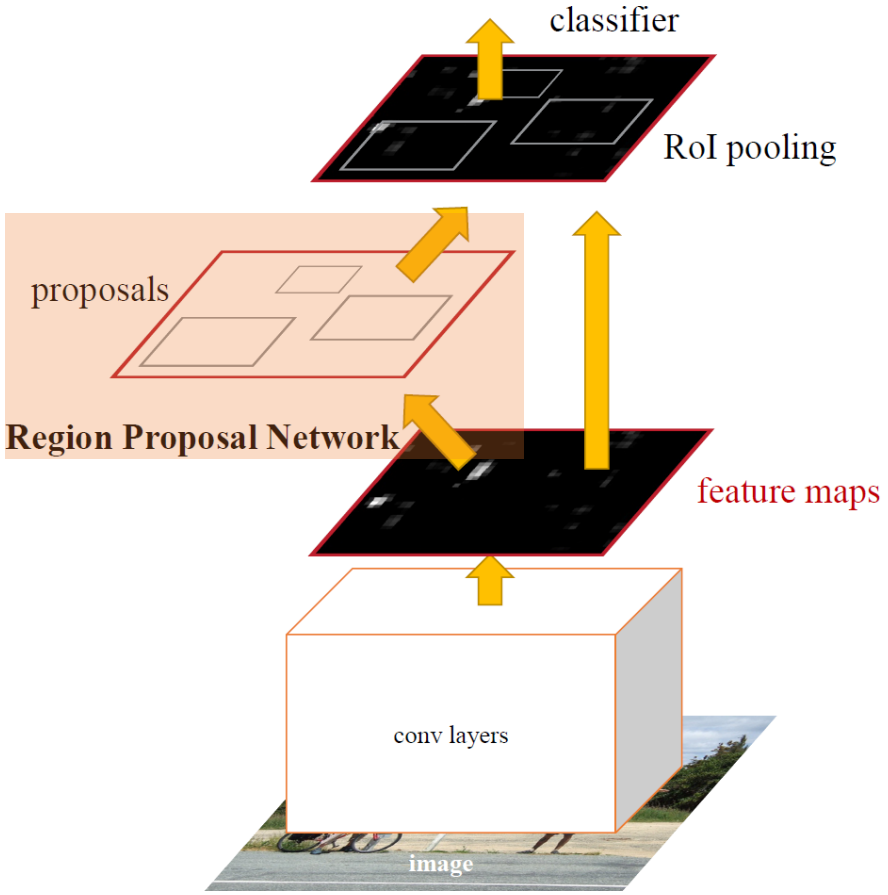
Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks

Faster R-CNN Region Proposal Network

The region proposal network predicts the object bounds and objectness scores, i.e., if the proposal region has an object (foreground) or not (background).



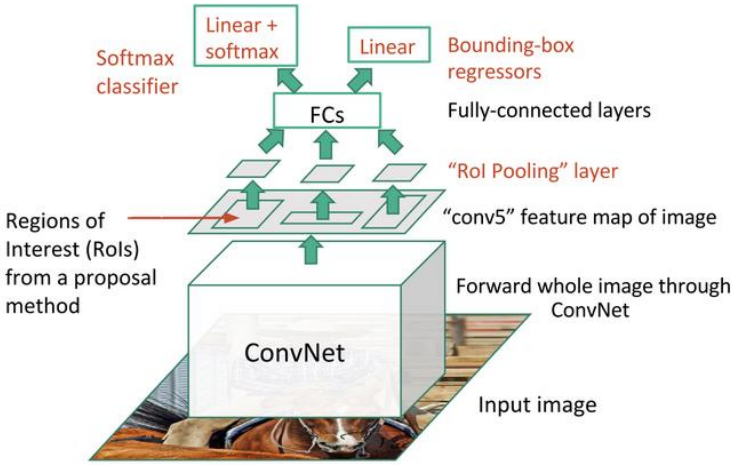
Anchor generation

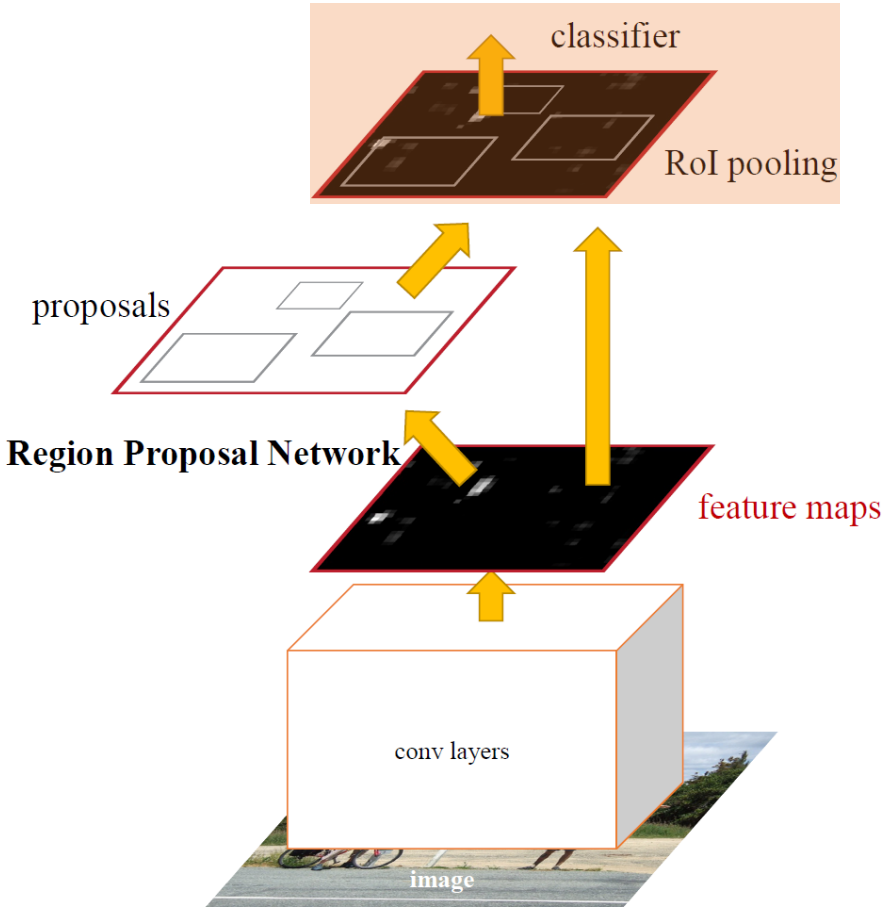


Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks

Faster R-CNN Classifier

The Region of Interest (RoI) pooling layers extracts fixed-size feature maps from each proposal. Two sibling networks are used, one for estimating the object categories and another for refining the bounding-box positions.





Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks

Faster R-CNN with Feature Pyramid Networks

Faster R-CNN:

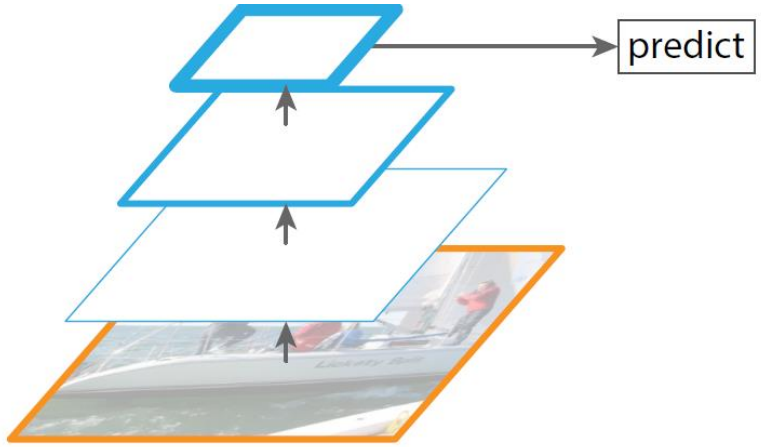
Detectron2

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

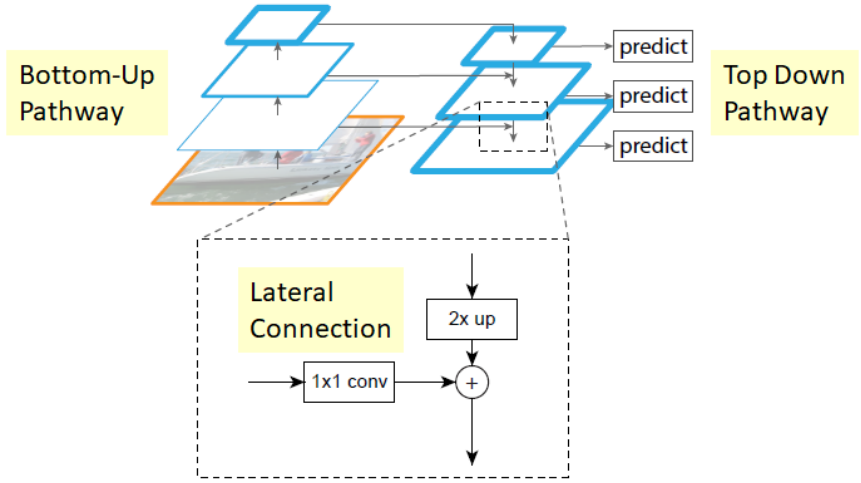
Faster R-CNN Model Zoo

Faster R-CNN Classifier – Improving Feature Representation

We can improve multi-scale object detection by using feature pyramid networks (FPN).



Before: A single feature map



Using Feature Pyramid Networks

Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection.

```python
from detectron2.data.datasets import register_coco_instances

# register train dataset
register_coco_instances(
    name="visdrone_train",
    metadata={},
    json_file="/home/visdrone/train/train_annotations.json",
    image_root="/home/visdrone/train/images",
)

# register validation dataset
register_coco_instances(
    name="visdrone_val",
    metadata={},
    json_file="/home/visdrone/val/val_annotations.json",
    image_root="/home/visdrone/val/images",
)
```

Training Faster R-CNN

Registering the dataset

Changing the config file

Launch Training

Training Faster R-CNN

```python
from detectron2 import model_zoo
from detectron2.config import get_cfg

# load default config
cfg = get_cfg()
# inherit config from faster rcnn r50 fpn
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("visdrone_train",)
cfg.DATASETS.TEST = ("visdrone_val",)
# perform multi-process data loading
cfg.DATALOADER.NUM_WORKERS = 8
# load pre-trained model weights
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.BASE_LR = 0.001
cfg.SOLVER.MAX_ITER = 17000
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.STEPS = [13260, 15800]
# number of classes
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
cfg.OUTPUT_DIR = "../experiments/faster_rcnn_r_50_fpn_3x"
```

Registering the dataset

Changing the config file

Launch Training

For changing the test threshold score:

cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST

Training Faster R-CNN

```python
import os

from detectron2.engine import DefaultTrainer

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Registering the dataset

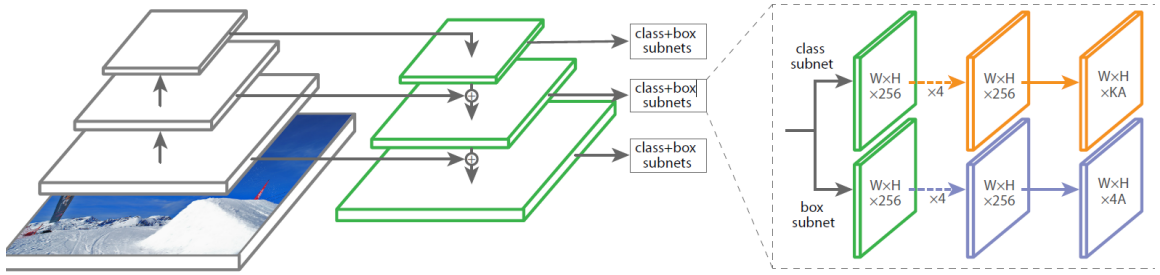
Changing the config file

Launch Training

RetinaNet

RetinaNet is a one-stage detector composed of:

- FPN backbone and anchor boxes generations.
- Two subnetworks for object classification and bounding box regression.

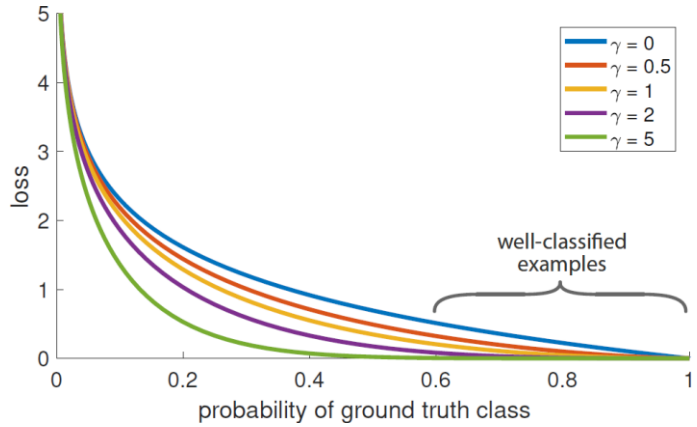


Feature Pyramid Network Backbone

Classification and regression subnetworks

Introduces focal loss used for helping one-stage detectors dealing with high class imbalance between foreground / background classes.

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$



Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection.

RetinaNet Training

RetinaNet:

Detectron2

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50	1x	0.205	0.041	4.1	37.4	190397773	model | metrics
R50	3x	0.205	0.041	4.1	38.7	190397829	model | metrics
R101	3x	0.291	0.054	5.2	40.4	190397697	model | metrics

Detectron2 Model Zoo

```python
from detectron2 import model_zoo
from detectron2.config import get_cfg

# load default config
cfg = get_cfg()
# inherit config from retinanet r50 fpn
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("visdrone_train",)
cfg.DATASETS.TEST = ("visdrone_val",)
# perform multi-process data loading
cfg.DATALOADER.NUM_WORKERS = 8
# load pre-trained model weights
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_3x.yaml")
cfg.SOLVER.BASE_LR = 0.001
cfg.SOLVER.MAX_ITER = 17000
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.STEPS = [13260, 15800]
# number of classes
cfg.MODEL.RETINANET.NUM_CLASSES = 1
cfg.OUTPUT_DIR = "../experiments/retinanet_R_50_FPN_3x"
```

Training RetinaNet

Registering the dataset (no changes)

Changing the config file

Launch Training (no changes)

For changing the test threshold score:

cfg.MODEL.RETINANET.SCORE_THRESH_TEST

Inference Visualization & Results

Faster R-CNN COCO Metrics

AP	AP50	AP75	APs	APm	APl
51.929	79.802	57.495	33.114	64.101	79.401

RetinaNet COCO Metrics

AP	AP50	AP75	APs	APm	APl
52.804	77.678	58.187	29.933	68.149	81.618

Inference Faster R-CNN

Inference RetinaNet

Data Augmentation

Enlarge the dataset that will help with the generalization of the network through tackling differences in lighting, viewpoint, scale, etc. Detectron2 defines already some transformations that may be used for building a sequence of augmentations.

Data Augmentation using Detectron2

```python
import detectron2.data.transforms as T

aug_list = T.AugmentationList(
    [
        T.ResizeShortestEdge([800, 800], 1333),
        T.RandomApply(T.RandomBrightness(0.5, 1.6), prob=0.5),
        T.RandomApply(T.RandomContrast(0.5, 1.6), prob=0.5),
        T.RandomApply(T.RandomSaturation(0.5, 1.6), prob=0.5),
        T.RandomFlip(prob=0.5, horizontal=True),
    ]
)
```

Defining our augmentation pipeline

```python
def aug_mapper(
    dataset_dict: list[dict], aug_list: T.AugmentationList, min_width_height: int = 10
) -> list[dict]:
    dataset_dict = copy.deepcopy(dataset_dict)
    image = utils.read_image(dataset_dict["file_name"], format="BGR")
    input_im = T.AugInput(image=image)
    transform = aug_list(input_im)
    (h, w, _) = input_im.image.shape
    image = torch.from_numpy(input_im.image.copy().transpose(2, 0, 1))
    anns_transformed = [
        utils.transform_instance_annotations(ann, transform, (h, w))
        for ann in dataset_dict.pop("annotations")
    ]
    instances = utils.annotations_to_instances(anns_transformed, (h, w))
    filtered_boxes_idx = filter_small_boxes(instances.gt_boxes.tensor, min_width_height)
    instances = instances[filtered_boxes_idx]
    instances = utils.filter_empty_instances(instances)
    return {
        "image": image,
        "instances": instances,
        "annotations": anns_transformed,
        "height": h,
        "width": w,
    }
```

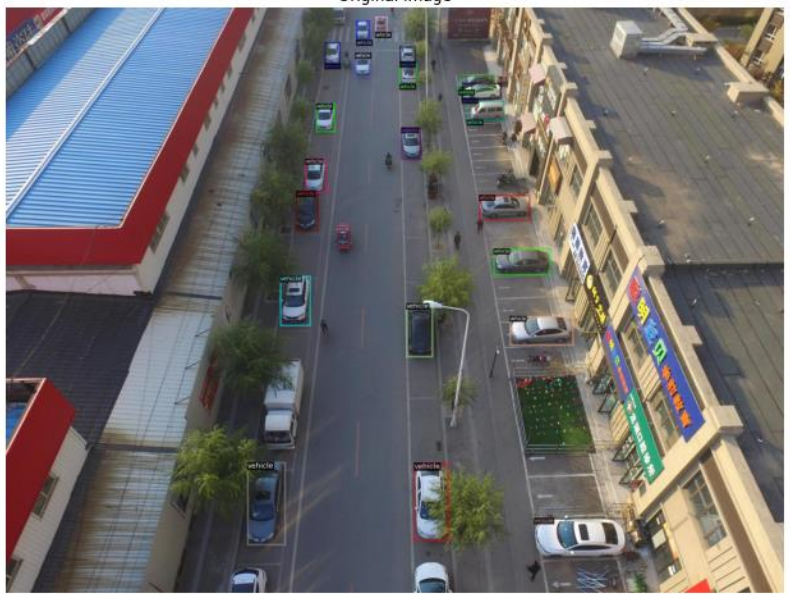
DatasetMapper that takes a dataset dict in Detectron2 dataset format and maps it into a format used by model. Takes a sequence of augmentations as input.

Data Augmentation using Albumentations

It’s also possible to integrate external libraries like Albumentations or Kornia, that provide a large collections of transformations that can be applied to computer vision.

```python
albu_transform = A.Compose(
    [
        A.OneOf([A.Blur(), A.GaussianBlur(), A.MedianBlur()], p=0.5),
        A.Cutout(num_holes=8, max_h_size=10, max_w_size=10, p=0.5),
        A.RandomSunFlare(
            num_flare_circles_lower=2, num_flare_circles_upper=5, src_radius=250, p=0.5
        ),
    ],
    bbox_params=A.BboxParams(
        format="coco", min_area=128, label_fields=["class_labels"]
    ),
)
```

Original Image



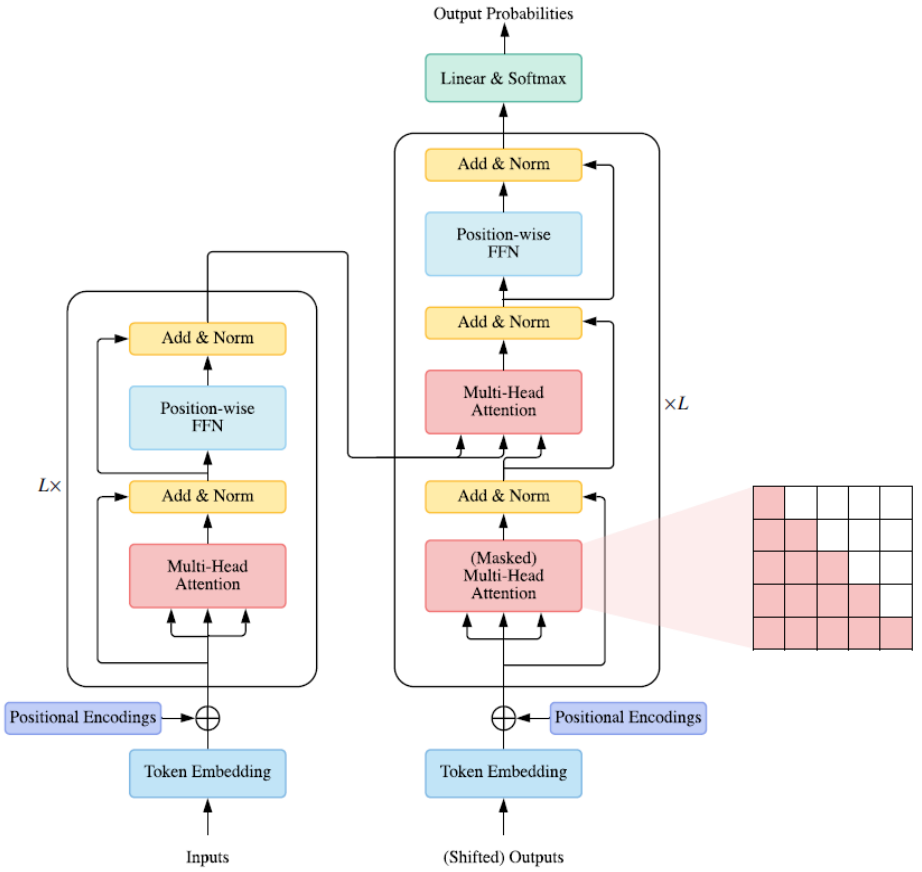
Transformed Image

Transformers

- Originally proposed as a sequence-to-sequence model for machine translation.
- General purpose-architecture with very few biases.
- Vanilla transformer has an encoder and a decoder block.
- Different scale and resolutions makes for different challenges when applying transformers to computer vision compared to NLP.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$\mathcal{O}(T^2 \cdot D)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Fully Connected	$\mathcal{O}(T^2 \cdot D^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Convolutional	$\mathcal{O}(K \cdot T \cdot D^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_K(T))$
Recurrent	$\mathcal{O}(T \cdot D^2)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$

Comparison of Self-Attention with other layers



Transformer Architecture

Vaswani, Ashish, et al. "Attention is all you need."

Transformers Self-Attention

Given an input sequence $\mathbf{z} \in \mathrm{R}^{T \times \mathrm{D}}$ we compute the queries, keys and values matrices:

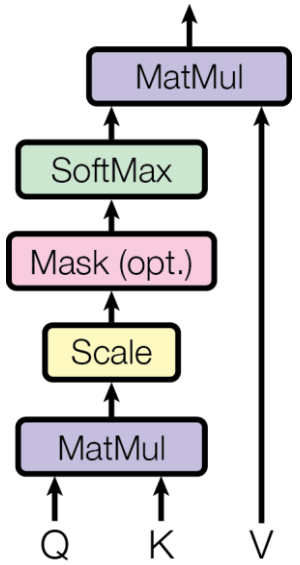
$$[\mathbf{Q}, \mathbf{K}, \mathbf{V}] = \mathbf{z}\mathbf{U}_{\mathrm{qkv}} \qquad \mathbf{U}_{\mathrm{qkv}} \in \mathrm{R}^{\mathrm{D} \times 3\mathrm{D_h}}$$

$\mathbf{Q} \in \mathrm{R}^{T \times \mathrm{D_h}}$, keys $\mathbf{K} \in \mathrm{R}^{\mathrm{T} \times \mathrm{D_h}}$ and values $\mathbf{V} \in \mathrm{R}^{\mathrm{T} \times \mathrm{D_h}}$

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\boldsymbol{K}^T}{\sqrt{D_K}}\right)\mathbf{V} = \mathbf{A}\mathbf{V}$$

A is called the Attention Matrix

The scaling factor $\sqrt{D_K}$ is applied to alleviate gradient vanishing problems



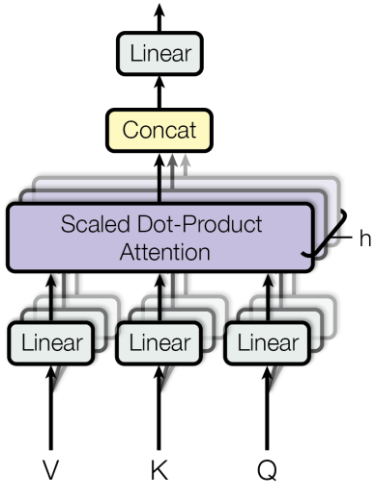
Self-Attention

Vaswani, Ashish, et al. "Attention is all you need."

From Self-Attention to Multihead Self-Attention

Multihead self-attention (MSA) is an extension of self-attention (SA) in which we run *k* self-attention operations , called "heads", in parallel.

$$\mathrm{MultiHead}(\mathrm{Q},\mathrm{K},\mathrm{V}) = [\mathrm{SA}_1(\mathrm{z}); \mathrm{SA}_2(z); \ldots; \mathrm{SA}_k(z)]\mathbf{U}_{msa} \qquad \mathbf{U}_{msa} \in \mathrm{R}^{\mathrm{k}\cdot\mathrm{D_h}\mathrm{x}\,D}$$

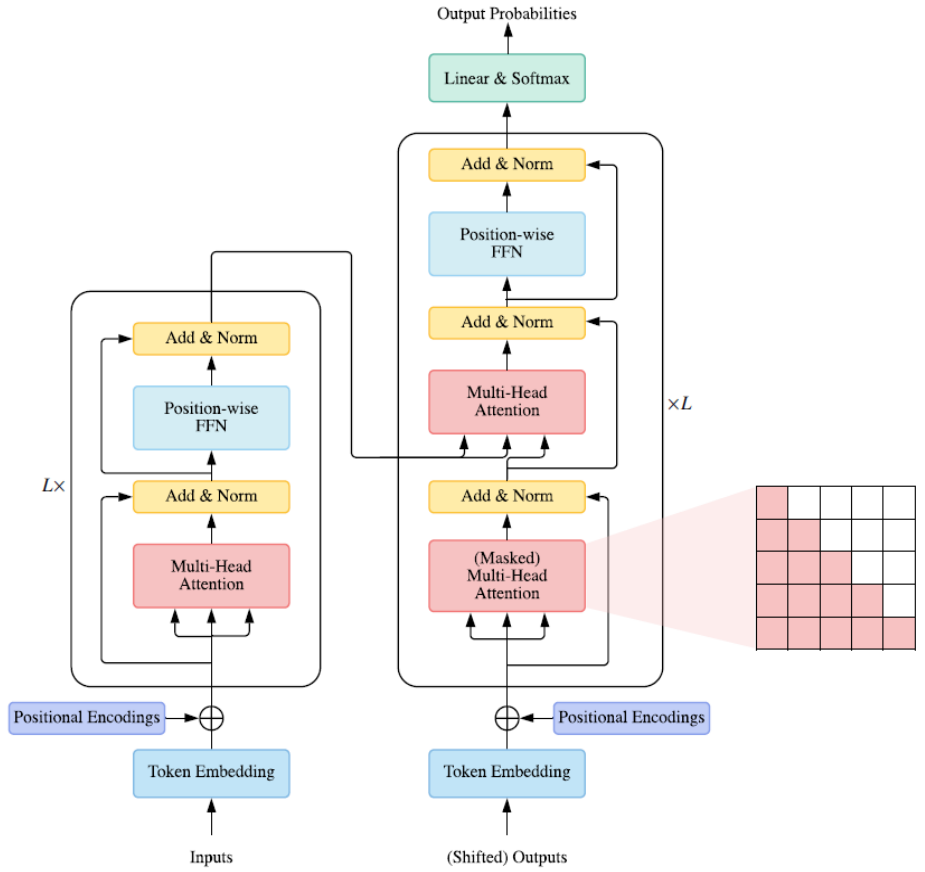


Multihead Self-Attention

Vaswani, Ashish, et al. "Attention is all you need."

Revisiting the Transformer

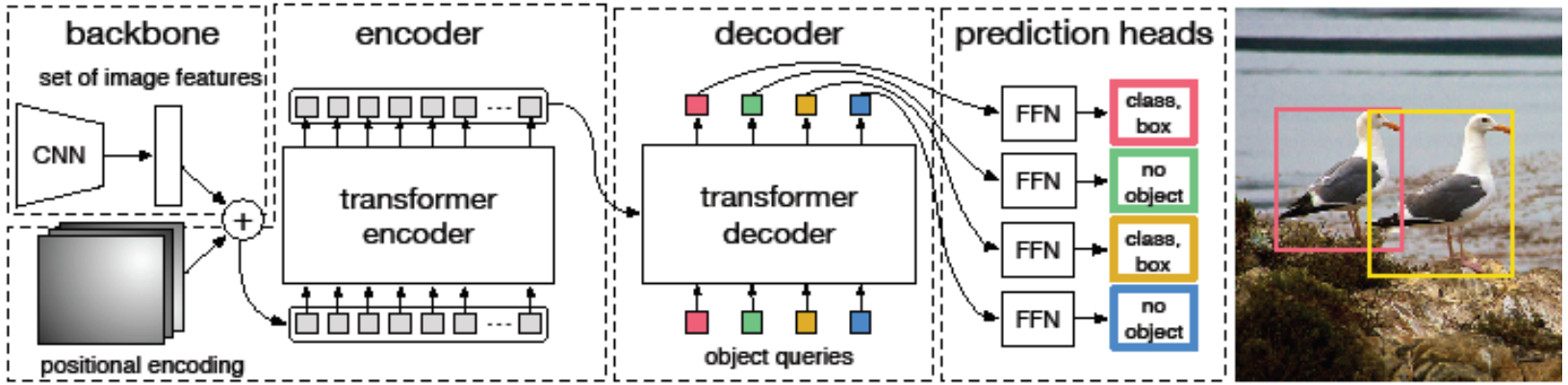
- Architectures may use only the encoder part, e.g., BERT.
- The decoder part, e.g., GPT-2.
- Both the encoder and decoder, e.g., DETR.
- Self-attention is invariant to the position of the sequence tokens so it’s common to add positional embeddings to the input.



Vaswani, Ashish, et al. "Attention is all you need."

Detection Transformer (DETR)

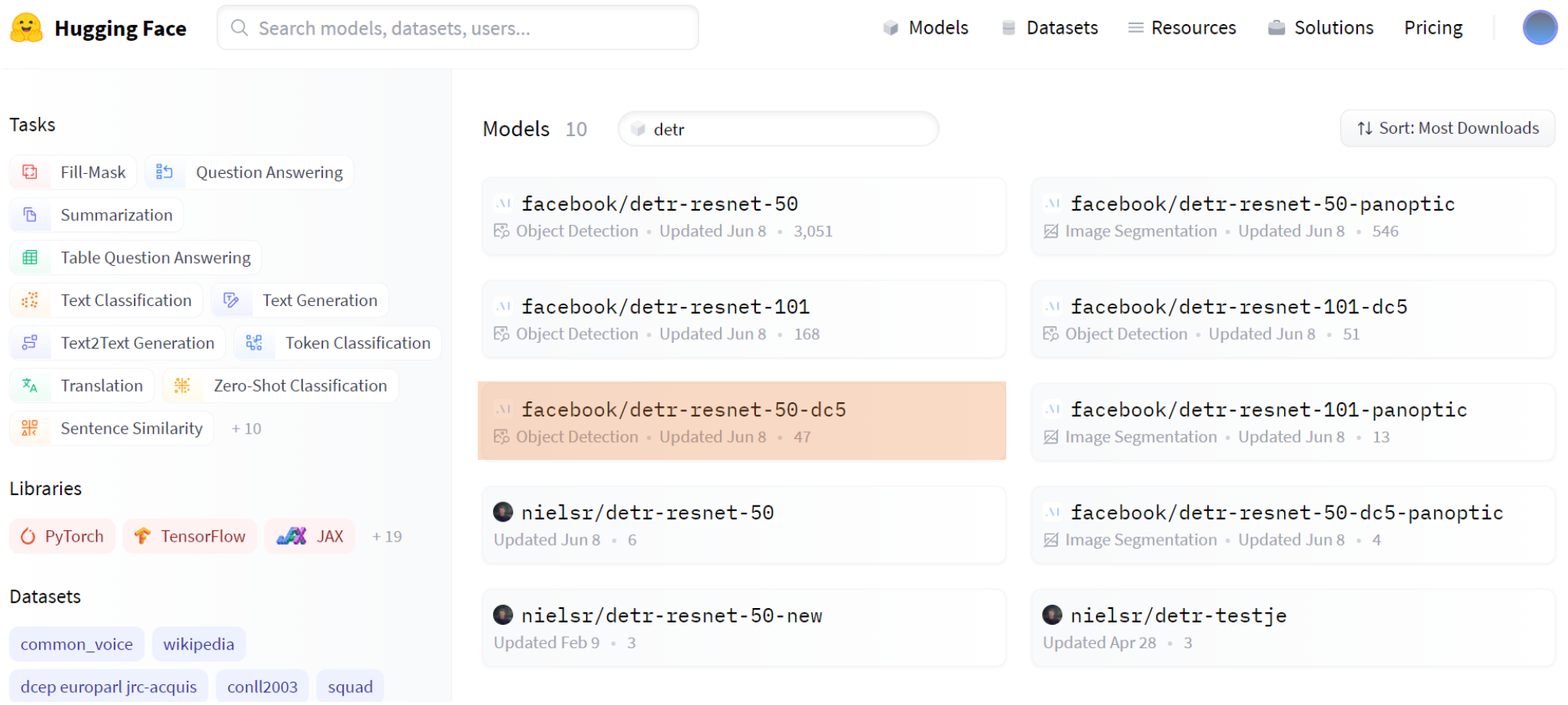
- Detection Transformer is based on a CNN backbone and a transformer decoder architecture.
- Treats the object detection problem as a direct set-based prediction.
- Forces unique assignment between ground-truth and predicted objects through a bipartite matching loss based on the Hungarian algorithm.



Detection Transformer High-Level Architecture

Carion, Nicolas, et al. "End-to-end object detection with transformers."

Detection Transformer (DETR) Training



Detection Transformer (DETR) Training

Very comprehensive documentation at HuggingFace model page and great example notebooks by NielsRogge linked at the page, explaining how to finetune DETR in our custom dataset.

Modifications from the example notebooks:

```
feature_extractor = DetrFeatureExtractor.from_pretrained('facebook/detr-resnet-50-dc5', max_size=1100)
model = DetrForObjectDetection.from_pretrained("facebook/detr-resnet-50-dc5")
config = DetrConfig.from_pretrained("facebook/detr-resnet-50-dc5", num_labels=len(id2label))
```

Resnet with dilated convolutions backbone

```
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_size=2, shuffle=True)
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=2)
```

Smaller batch size

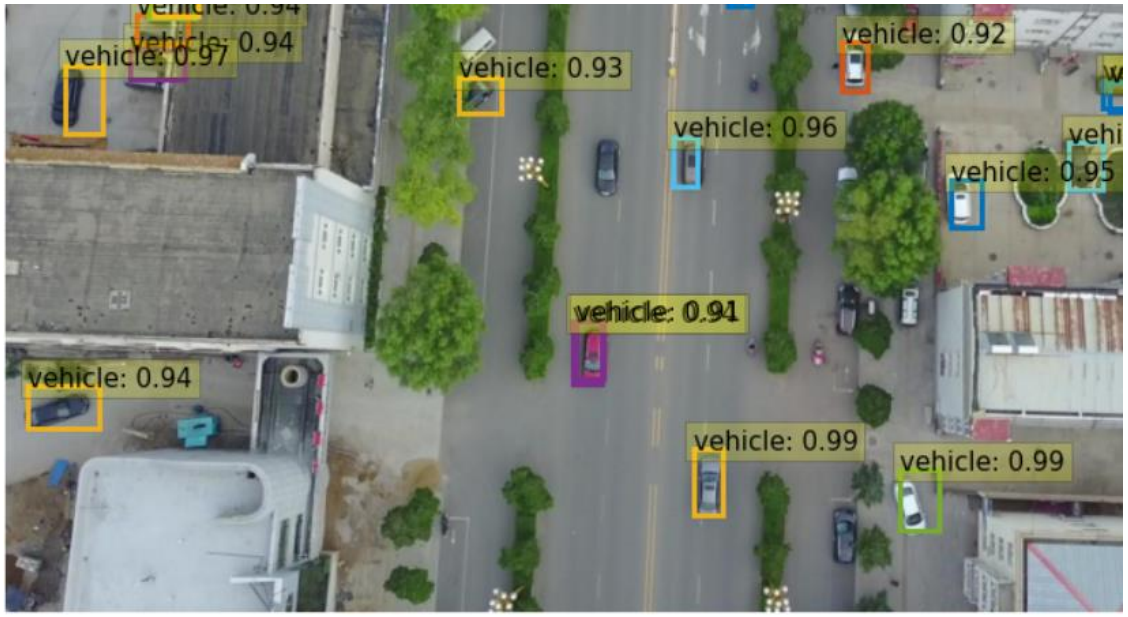
https://huggingface.co/transformers/model_doc/detr.html

https://github.com/NielsRogge/Transformers-Tutorials/tree/master/DETR

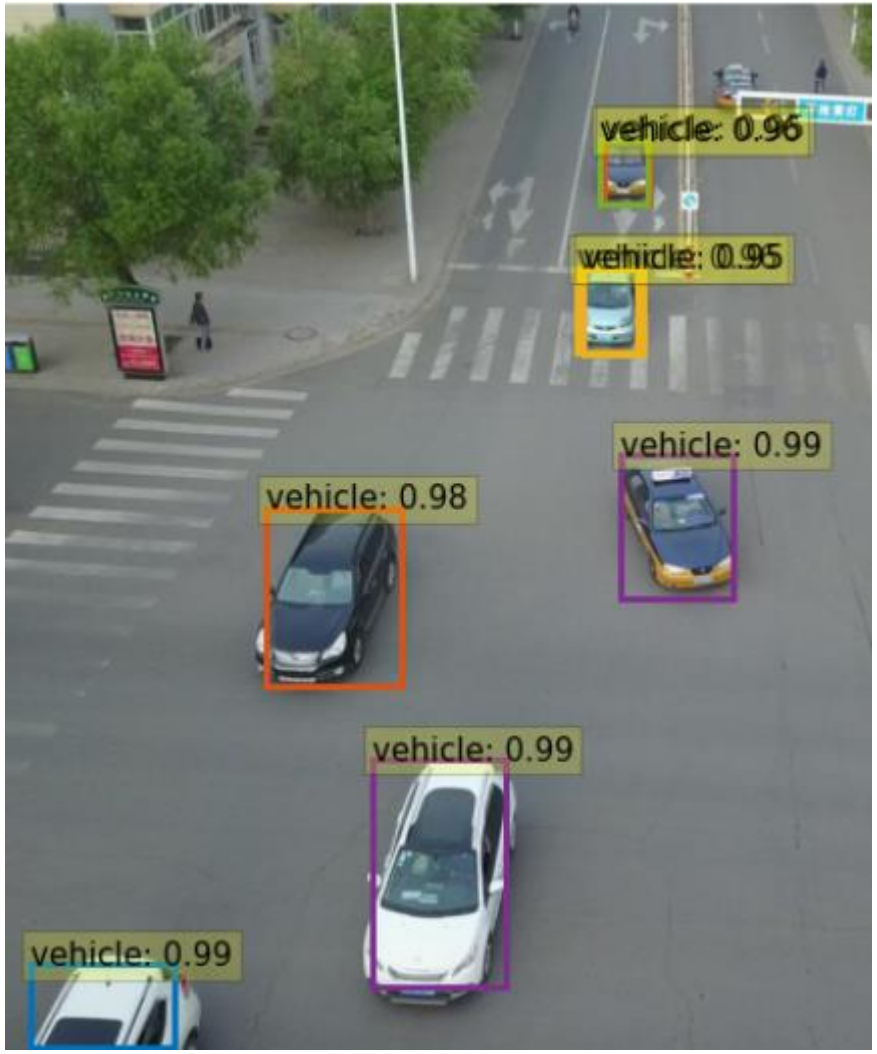
Inference Visualization DETR

DETR COCO metrics

AP	AP50	AP75	APs	APm	APl
9.5	25	5.2	1.2	1.64	46.2



Small-scale object example



Large and medium-scale object example

Where to go from here

Scaling up the backbone, e.g, from ResNet-50 to ResNet-101

Tuning the augmentation policy

Enriching the dataset with other aerial datasets (UAVDT, AU-AIR, MEVA, etc.)

Exploit temporal relationships in video object detection to reduce false positives

Other transformers architectures (Swin Transformer, Focal Transformer, etc.)

Conclusion

CNNs are powerful baselines

Transformer architectures applied to vision are becoming an increasing popular choice in research and practice

DETR is better suited for medium and large scale objects, developments like FPN may help address small-scale objects

Transformer will continue to adapted to more specific downstream tasks and applications (Object Detection, Image Segmentation, etc.)

Transformers are an unifying framework for multimodal data (text, image, video, audio)

Thank You