

✓ DETR - Detection Transformer

Reference: [https://github.com/NielsRogge/Transformers-Tutorials/blob/master/DETR/Fine_tuning_DetrForObjectDetection_on_custom_dataset_\(balloon\).ipynb](https://github.com/NielsRogge/Transformers-Tutorials/blob/master/DETR/Fine_tuning_DetrForObjectDetection_on_custom_dataset_(balloon).ipynb)

Original DETR paper: <https://arxiv.org/abs/2005.12872>

Original DETR repo: <https://github.com/facebookresearch/detr>

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
!python --version
```

```
Python 3.10.12
```

```
!python -m pip install --upgrade pip
```

```
!pip install supervision==0.3.0
```

```
!pip install transformers
```

```
!pip install pytorch-lightning
```

```
!pip install timm
```

```
!pip install cython
```

```
!pip install pycocotools
```

```
!pip install scipy
```


Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.11.1)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use pipx instead.

```
!pip --version
```

```
pip 23.3.2 from /usr/local/lib/python3.10/dist-packages/pip (python 3.10)
```

```
import torch  
torch.__version__
```

```
'2.1.0+cu121'
```

```
import supervision as sv  
import transformers
```

```
# supervision.__version__ , transformers.__version__
```

```
import pytorch_lightning  
print(pytorch_lightning.__version__)
```

```
2.1.3
```

✓ Create COCO data loaders

```

import os
import torch
import torchvision.transforms as T
import torchvision

dataset = '/content/drive/MyDrive/Detect'

ANNOTATION_FILE_NAME = "annotations.json"
TRAIN_DIRECTORY = os.path.join(dataset, "train")
VAL_DIRECTORY = os.path.join(dataset, "valid")
TEST_DIRECTORY = os.path.join(dataset, "test")

# Instantiate the image processor
from transformers import DetrImageProcessor
image_processor = DetrImageProcessor.from_pretrained("facebook/detr-resnet-50")

class CocoDetection(torchvision.datasets.CocoDetection):
    def __init__(
        self,
        image_directory_path: str,
        image_processor,
        train: bool = True
    ):
        annotation_file_path = os.path.join(image_directory_path, ANNOTATION_FILE_NAME)
        if not os.path.exists(annotation_file_path):
            raise FileNotFoundError(f"Annotation file not found: {annotation_file_path}")

        super(CocoDetection, self).__init__(image_directory_path, annotation_file_path)
        self.image_processor = image_processor

    def __getitem__(self, idx):
        images, annotations = super(CocoDetection, self).__getitem__(idx)
        image_id = self.ids[idx]
        annotations = {'image_id': image_id, 'annotations': annotations}
        encoding = self.image_processor(images=images, annotations=annotations, return_tensors="pt")
        pixel_values = encoding["pixel_values"].squeeze()
        target = encoding["labels"][0]

        return pixel_values, target

# Now you can use the image_processor
try:
    TRAIN_DATASET = CocoDetection(image_directory_path=TRAIN_DIRECTORY, image_processor=image_processor, train=True)
    VAL_DATASET = CocoDetection(image_directory_path=VAL_DIRECTORY, image_processor=image_processor, train=False)

```

```

TEST_DATASET = CocoDetection(image_directory_path=TEST_DIRECTORY, image_processor=image_processor, train=False)

print("Number of training examples:", len(TRAIN_DATASET))
print("Number of validation examples:", len(VAL_DATASET))
print("Number of test examples:", len(TEST_DATASET))

except FileNotFoundError as e:
    print(f"Error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
preprocessor_config.json: 100% 274/274 [00:00<00:00, 17.4kB/s]

The `max_size` parameter is deprecated and will be removed in v4.26. Please specify in `size['longest_edge']` instead`.
loading annotations into memory...
Done (t=0.66s)
creating index...
index created!
loading annotations into memory...
Done (t=0.39s)
creating index...
index created!
loading annotations into memory...
Done (t=0.63s)
creating index...
index created!
Number of training examples: 267
Number of validation examples: 10
Number of test examples: 5

```

```
# Visualize if dataset is loaded properly

import random
import cv2
import numpy as np

# select random image
image_ids = TRAIN_DATASET.coco.getImgIds()
image_id = random.choice(image_ids)
print('Image #{}'.format(image_id))

# load image and annotations
image = TRAIN_DATASET.coco.loadImgs(image_id)[0]
annotations = TRAIN_DATASET.coco.imgToAnns[image_id]
image_path = os.path.join(TRAIN_DATASET.root, image['file_name'])
image = cv2.imread(image_path)

# annotate
detections = sv.Detections.from_coco_annotations(coco_annotation=annotations)

# we will use id2label function for training
categories = TRAIN_DATASET.coco.cats
id2label = {k: v['name'] for k,v in categories.items()}

labels = [
    f"{id2label[class_id]}"
    for _, _, class_id, _
    in detections
]

box_annotator = sv.BoxAnnotator()
frame = box_annotator.annotate(scene=image, detections=detections, labels=labels)

%matplotlib inline
sv.show_frame_in_notebook(image, (8, 8))
```

Image #186



```
from torch.utils.data import DataLoader

def collate_fn(batch):
    pixel_values = [item[0] for item in batch]
    encoding = image_processor.pad(pixel_values, return_tensors="pt")
    labels = [item[1] for item in batch]
    return {
        'pixel_values': encoding['pixel_values'],
        'pixel_mask': encoding['pixel_mask'],
        'labels': labels
    }

TRAIN_DATALOADER = DataLoader(dataset=TRAIN_DATASET, collate_fn=collate_fn, batch_size=4, shuffle=True)
VAL_DATALOADER = DataLoader(dataset=VAL_DATASET, collate_fn=collate_fn, batch_size=4)
TEST_DATALOADER = DataLoader(dataset=TEST_DATASET, collate_fn=collate_fn, batch_size=4)
```

✓ Train model with PyTorch Lightning

The DETR model is loaded using the Hugging Face Transformers library


```

import pytorch_lightning as pl
from transformers import DetrForObjectDetection
import torch

class Detr(pl.LightningModule):

    def __init__(self, lr, lr_backbone, weight_decay):
        super().__init__()
        self.model = DetrForObjectDetection.from_pretrained(
            pretrained_model_name_or_path="facebook/detr-resnet-50",
            revision="no_timm",
            num_labels=len(id2label),
            ignore_mismatched_sizes=True
        )

        self.lr = lr
        self.lr_backbone = lr_backbone
        self.weight_decay = weight_decay

    def forward(self, pixel_values, pixel_mask):
        return self.model(pixel_values=pixel_values, pixel_mask=pixel_mask)

    def common_step(self, batch, batch_idx):
        pixel_values = batch["pixel_values"]
        pixel_mask = batch["pixel_mask"]
        labels = [{k: v.to(self.device) for k, v in t.items()} for t in batch["labels"]]

        outputs = self.model(pixel_values=pixel_values, pixel_mask=pixel_mask, labels=labels)

        loss = outputs.loss
        loss_dict = outputs.loss_dict

        return loss, loss_dict

    def training_step(self, batch, batch_idx):
        loss, loss_dict = self.common_step(batch, batch_idx)
        # logs metrics for each training_step, and the average across the epoch
        self.log("training_loss", loss)
        for k,v in loss_dict.items():
            self.log("train_" + k, v.item())

        return loss

```

```

def validation_step(self, batch, batch_idx):
    loss, loss_dict = self.common_step(batch, batch_idx)
    self.log("validation/loss", loss)
    for k, v in loss_dict.items():
        self.log("validation_" + k, v.item())

    return loss

def configure_optimizers(self):
    # DETR authors decided to use different learning rate for backbone
    # you can learn more about it here:
    # - https://github.com/facebookresearch/detr/blob/3af9fa878e73b6894ce3596450a8d9b89d918ca9/main.py#L22-L23
    # - https://github.com/facebookresearch/detr/blob/3af9fa878e73b6894ce3596450a8d9b89d918ca9/main.py#L131-L139
    param_dicts = [
        {
            "params": [p for n, p in self.named_parameters() if "backbone" not in n and p.requires_grad]},
        {
            "params": [p for n, p in self.named_parameters() if "backbone" in n and p.requires_grad],
            "lr": self.lr_backbone,
        },
    ]
    return torch.optim.AdamW(param_dicts, lr=self.lr, weight_decay=self.weight_decay)

def train_dataloader(self):
    return TRAIN_DATALOADER

def val_dataloader(self):
    return VAL_DATALOADER

```

```
model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)
```

```
batch = next(iter(TRAIN_DATALOADER))
outputs = model(pixel_values=batch['pixel_values'], pixel_mask=batch['pixel_mask'])
```

Some weights of DetrForObjectDetection were not initialized from the model checkpoint at facebook/detr-resnet-50 and are newly initialized

- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the checkpoint and torch.Size([101, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoint and torch.Size([101]) in the model instantiated

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

MAX_EPOCHS: 250

```

from pytorch_lightning import Trainer

# settings
MAX_EPOCHS = "250" # @param [2, 5,10,15,50,75,100,150,200,250,300]

trainer = Trainer(devices=1, accelerator="gpu", max_epochs=int(MAX_EP

trainer.fit(model)

INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name | Type | Params
-----
0 | model | DetrForObjectDetection | 41.5 M
-----
18.1 M    Trainable params
23.5 M    Non-trainable params
41.5 M    Total params
166.108   Total estimated model params size (MB)
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/data_connector.py:441: The 'val_dataloade
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/utilities/data.py:77: Trying to infer the `batch_size` from
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/data_connector.py:441: The 'train_dataaloa
Epoch 249: 100% 67/67 [00:35<00:00, 1.91it/s, v_num=1]
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/utilities/data.py:77: Trying to infer the `batch_size` from
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=250` reached.

```

✓ Save and load model

```

MODEL_PATH = '/content/drive/MyDrive/Detect/custom-model'
model.model.save_pretrained(MODEL_PATH)

```

✓ Inference on test dataset

Let's visualize the predictions of DETR on the first image of the validation set.

```
import random
import cv2
import numpy as np
import matplotlib.pyplot as plt
from transformers import DetrForObjectDetection
import torch
import supervision as sv
import transformers

# loading model
model = DetrForObjectDetection.from_pretrained("/content/drive/MyDrive/Detect/custom-model")
# model.to(DEVICE)

# utils
categories = TEST_DATASET.coco.cats
id2label = {k: v['name'] for k,v in categories.items()}
box_annotator = sv.BoxAnnotator()

# select random image
image_ids = TEST_DATASET.coco.getImgIds()
image_id = random.choice(image_ids)
print('Image #{}'.format(image_id))

# load image and annotations
image = TEST_DATASET.coco.loadImgs(image_id)[0]
annotations = TEST_DATASET.coco.imgToAnns[image_id]
image_path = os.path.join(TEST_DATASET.root, image['file_name'])
image = cv2.imread(image_path)

# Annotate ground truth
detections = sv.Detections.from_coco_annotations(coco_annotation=annotations)
labels = [f"{id2label[class_id]}" for _, _, class_id, _ in detections]
frame_ground_truth = box_annotator.annotate(scene=image.copy(), detections=detections, labels=labels)

# Annotate detections
with torch.no_grad():

    # load image and predict
    inputs = image_processor(images=image, return_tensors='pt')
```

```
outputs = model(**inputs)

# post-process
target_sizes = torch.tensor([image.shape[:2]])
results = image_processor.post_process_object_detection(
    outputs=outputs,
    threshold=0.20,
    target_sizes=target_sizes
)[0]

detections = sv.Detections.from_transformers(transformers_results=results)
labels = [f"{id2label[class_id]} {confidence:.2f}" for _, confidence, class_id, _ in detections]
frame_detections = box_annotator.annotate(scene=image.copy(), detections=detections, labels=labels)

# %matplotlib inline # Remove this line, we won't use it anymore

# Combine both images side by side and display
fig, axs = plt.subplots(1, 2, figsize=(20, 10))
axs[0].imshow(cv2.cvtColor(frame_ground_truth, cv2.COLOR_BGR2RGB))
axs[0].axis('off')
axs[0].set_title('Ground Truth')
```