## DETR - Detection Transformer

Reference: https://github.com/NielsRogge/Transformers-

Tutorials/blob/master/DETR/Fine\_tuning\_DetrForObjectDetection\_on\_custom\_dataset\_(balloon).ipynb

Original DETR paper: <a href="https://arxiv.org/abs/2005.12872">https://arxiv.org/abs/2005.12872</a>

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: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)

WARNING: Running bib as the 'root' user can result in broken permissions and conflicting behaviour with the system backage manager. It

!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 sy
import transformers

# supervision.__version__, transformers.__version__

import pytorch_lightning
print(pytorch_lightning.__version__)

2.1.3
```

### Create COCO data loaders

```
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 annotatons
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 = Γ
           {
                "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,
           },
        1
        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 initializ
     - class labels classifier.weight: found shape torch.Size([92, 256]) in the checkpoint and torch.Size([101, 256]) in the model instantiat
     - 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

https://colab.research.google.com/drive/1bd6i OoPfEbQsaoU6GXN4cmSY8c996Jx#scrollTo=64cf1368-1f4a-4c08-aefe-3d90d2fda638&printMode=true

```
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 EPG
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
             Trainable params
     18.1 M
     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 dataloa
     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 annotatons
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')
```