Rodrigo Velazquez

**DETR-RESNET50 for object detection**

For the model that detects a car in the image I went for a pre-existing model for object detection that was introduced by Nicolas Carion and other facebook scientists in 2020 and it’s a object detection algorithm that uses transformers on the backbone, we specifically searched for the DETR model that also incorporate RESNET50 for the feature extraction.

Facebook DETR GitHub

<https://github.com/facebookresearch/detr/blob/main/README.md>

HuggingFace DETR Resnet 50 MODEL

<https://huggingface.co/facebook/detr-resnet-50>

A diagram of a bird

Description automatically generated

**CAR Dataset**

As our team is still working on the datascrapping I wanted to be able to train and evaluate the model im building, so while we retrieve our own dataset I downloaded a dataset of images from Kaggle to start working.

<https://www.kaggle.com/datasets/kshitij192/cars-image-dataset>

**Data Annotations and labeling with Make Sense AI**

Once I downloaded the images from Kaggle I needed to create the manual bounding boxes and class identification so I could train and evaluate my model, for this task I used makesense.ai and then I exported the annotations as xml file.

<https://www.makesense.ai/>

A black car on a white background

Description automatically generated

Once I had that xml file I needed to transform it into the COCO format that RESNET50 was trained on, to do this I wrote a python code that helped me to create the json file with the specific format I needed.

import xml.etree.ElementTree as ET  
import json  
import os  
  
  
def voc\_to\_coco(voc\_dir, output\_json\_path):  
 coco\_data = {  
 "images": [],  
 "annotations": [],  
 "categories": [{"id": 1, "name": "car"}] # Assuming 'car' is the only category  
 }  
 annotation\_id = 1 # Unique ID for each annotation  
  
 for idx, filename in enumerate(os.listdir(voc\_dir)):  
 if filename.endswith('.xml'):  
 tree = ET.parse(os.path.join(voc\_dir, filename))  
 root = tree.getroot()  
  
 # Extract image data  
 image\_data = {  
 "id": idx + 1, # Incremental ID for each image  
 "file\_name": root.find('filename').text,  
 "width": int(root.find('size/width').text),  
 "height": int(root.find('size/height').text)  
 }  
 coco\_data["images"].append(image\_data)  
  
 # Iterate through all objects in the XML  
 for obj in root.findall('object'):  
 bndbox = obj.find('bndbox')  
  
 # Calculate bounding box coordinates  
 xmin = int(bndbox.find('xmin').text)  
 ymin = int(bndbox.find('ymin').text)  
 xmax = int(bndbox.find('xmax').text)  
 ymax = int(bndbox.find('ymax').text)  
  
 annotation = {  
 "id": annotation\_id,  
 "image\_id": idx + 1, # Link annotation to the image  
 "category\_id": 1, # All objects are assumed to be 'car'  
 "bbox": [  
 xmin, ymin, xmax - xmin, ymax - ymin  
 ],  
 "area": (xmax - xmin) \* (ymax - ymin), # Width \* Height  
 "iscrowd": 0  
 }  
 coco\_data["annotations"].append(annotation)  
 annotation\_id += 1 # Increment annotation ID  
  
 # Write the COCO data to a JSON file  
 with open(output\_json\_path, 'w') as f:  
 json.dump(coco\_data, f, indent=4)  
  
  
voc\_to\_coco('C:/Users/rooyv/Documents/Loyalist/TERM 2/STEP 2/Annotations', 'output\_coco\_annotations2.json')

|  |  |
| --- | --- |
| XML Format | COCO JSON Format |
|  |  |

**Transfer Learning Videos**

From this video I understood the main characteristics of a transfer learning approach, by using this method we can rely on a pre-trained model to predict the features that we are interested in when we disconnect the last layer and tweak to predict our class of interest.

This is specially useful for deep learning models.

In this approach we use the already stored weights of already built models.

From this video I also learned that we need to send the data to the device we are using, this depends on whether we are using CPUor GPU.

<https://www.youtube.com/watch?v=K0lWSB2QoIQ&t=116s>

**Progress**

This week I managed to create a code that retrain the pre-trained model of DETR ResNet-50, obtaining the loss on each training Epoch.

A screenshot of a web page

Description automatically generated

**Challenges**

Handling multiple annotations and bboxes- so one of the challenges during this code phase was the sizes of the objects, as we are dealing with tensors, arrays and lists or dictionaries, it starts to get complex dealing with every variable. But the most complicated case was to account for the case of images that have more than 1 car on the frame, meaning that image has multiple bounding boxes, and we need to identify those cases and retrieve all the original bounding boxes.

Another challenge present was that DETR was trained using images of minimum 800 pixels by side, so I needed to resize each image, but also the bounding box created using makesense.ia was already done when I noticed that, so the bounding box had to be resized as well and then ensure that the box enclosed the car after all this manipulations.

Maping the annotations with the image id was also time consuming, but it was achieved.