Selected Topics in Visual Recognition using Deep Learning Homework 2

Github link of the codes

Project Repository: VRDL_21au

Reference

Github repository MMdetection and related pretrained models are used and modified in this

project.

Introduction

The project is an experiment of implementing deep learning model to deal with objection detection task. The dataset used here is The Street View House Numbers (SVHN) Dataset, which is obtained from house numbers in Google Street View images. Each image of the datasets contains multiple objects (number0~9) and its position (bounding box). For experimental sake, a subset of it is applied to our model, with 33,402 images as training data and 13,608 images as testing data. The main feature of the task is that compared to typical image classification problem, object detection task requires models to learn the spatial information of potentially multiple objects. Also, both mean Average Precision (mAP) and inference speed are important metrics which should be considered when optimizing the model. In this project, we analyze the predictability of several commonly-used models such as Faster-RCNN, Cascade-RCNN and Yolov3, and also examine the performance of some latest methodology like Deformable DETR.

Data Preprocessing

Some image preprocessing methodology, such as scaling, normaliztion and data augmentation, is shared with image classification task, so here we skip those detailed preprocessing description. For more information, pytorch documentation or my homework report is the reference.

Another thing is that we do format transfer to label information. The raw label file is stored in *.mat* format, which is MATLAB file that stores workspace variables. In this project, the data is transformed into *.json* format in Coco dataset style. It is a commonly-used format when

training object detection model and shall work well with those interface that MMdetection or other libraries offer.

```
annotation{
    "id": int,
    "image_id": int,
    "category_id": int,
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height],
    "iscrowd": 0 or 1,
}

categories[{
    "id": int,
    "name": str,
    "supercategory": str,
}]
```

Basic Deep Learning Techniques

We also apply transfer learning and learning rate scheduling techniques, by MMdetection default settings, to our object detection framework. Pretrained ResNet-50 model is used as our initial backbone weights to utilize the domain knowledge learned from other tasks. Only last layer (fully-connected part) is replaced to finetune the whole model. For the scheduling part, step-LR is used so that smaller learning rate will be used in the last few epochs.

Deformable DETR: Deformable Transformers for End-to-End Object Detection

Deformable DETR, based on Detection with Transformer (DETR), is an object-detection model appling transformer ideas. It was proposed in ICLR 2021 to solve the problem DETR suffers, such as slow convergence speed and limited feature spatial resolution. We apply this method by finetuning pretrained model with SVHN datasets, and examine its capability of detecting house number object.

Experiment Result

We summarize the mAP and inference speed of all the models mentioned above, and run time is evaluated on Nvidia Tesla K80 GPU offered by Google Colab. Inference speed may fluctuate according to the enviornment setting of Colab, hence relative speed may be a better metric.

Model	mAP (%)	Inference Speed (ms)	Relative Speed
Faster RCNN	39.29	308.3	1x
Cascade RCNN	41.12	434.7	0.71x
Yolov3	37.10	75.5	4.08x
Deformable DETR	36.59	351.4	0.88x

Summary

As we could see above, the basic implementation of RCNN family models could achieve a decent mAP, with Faster RCNN 39.29% and Cascade RCNN 41.12% respectively, the inference speed, however is slower. On the other hand, Yolov3 runs four times speed of Faster RCNN while accuracy is lower. Obviously there exists a trade-off between accuracy and inference speed, so user should choose models according to the priority of their needs.

Deformable DTER performs poor in accuracy and speed, one possible reason for this is that due to limited time, 25 epochs (half of the default setting) are trained to this model, and some further analysis should be done here.

Screenshot of Inference Speed

- Faster RCNN

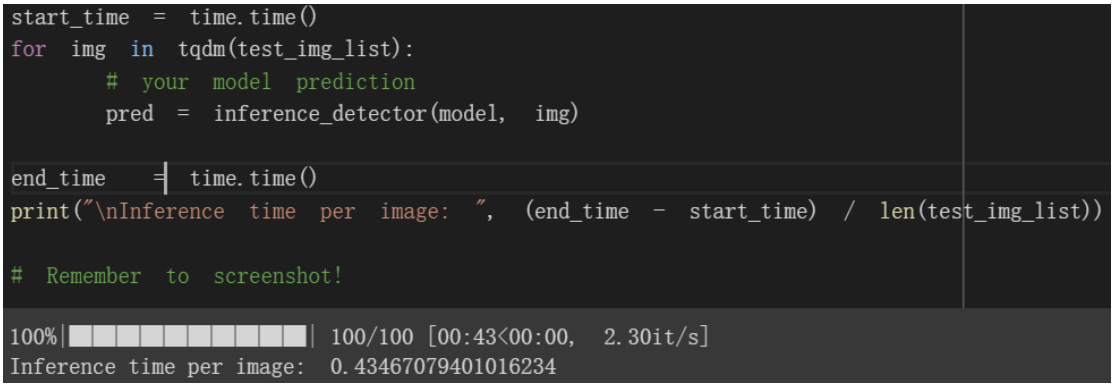
```
start_time = time.time()
for img in tqdm(test_img_list):
        # your model prediction
        pred = inference_detector(model, img)

end_time = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))

# Remember to screenshot!
```

```
100%|██████████| 100/100 [00:30<00:00,  3.24it/s]
Inference time per image:  0.30825069427490237
```

- Cascade RCNN



```
start_time = time.time()
for img in tqdm(test_img_list):
        # your model prediction
        pred = inference_detector(model, img)

end_time = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))

# Remember to screenshot!
```

```
100%|██████████| 100/100 [00:43<00:00,  2.30it/s]
Inference time per image:  0.43467079401016234
```

- Yolov3

```
start_time = time.time()
for img in tqdm(test_img_list):
        # your model prediction
        pred = inference_detector(model, img)

end_time = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))

# Remember to screenshot!
```

```
100%|██████████| 100/100 [00:07<00:00, 13.25it/s]
Inference time per image:  0.07553750514984131
```

- Deformable DETR

```
start_time = time.time()
for img in tqdm(test_img_list):
        # your model prediction
        pred = inference_detector(model, img)

end_time = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))

# Remember to screenshot!
```

```
  0%|          | 0/100 [00:00<?, ?it/s]/content/mmdetection/mmdet/models/utils/positional_enc
  dim_t = self.temperature**(2 * (dim_t // 2) / self.num_feats)
/content/mmdetection/mmdet/models/utils/transformer.py:883: UserWarning: __floordiv__ is depr
  dim_t = temperature**(2 * (dim_t // 2) / num_pos_feats)
/content/mmdetection/mmdet/models/dense_heads/detr_head.py:666: UserWarning: __floordiv__ is
  bbox_index = indexes // self.num_classes
100%|██████████| 100/100 [00:35<00:00,  2.85it/s]
Inference time per image:  0.3513914918899536
```