KIT315 Final Project

Kangaroo-wallaby detection

(David)

This project will do Kangaroos and Wallabies detection and classification.

1. Evironment Setup

In this project, the following packages and libraries are used.

- Python 3.8
- pytorch 1.8.0
- torchvision 0.9.0
- cudatoolkit 10.2
- pycocotools 2.0.2
- numpy 1.20.0
- detectron2

Setup

```
conda create --name detectron python=3.8
```

```
git clone https://github.com/facebookresearch/detectron2.git
```

```
conda activate detectron
```

```
conda install pytorch==1.8.0 torchvision==0.9.0 cudatoolkit=10.2 -c pytorch
```

```
pip install cython
```

```
cd detectron2
```

```
pip install -e .
```

```
pip install opencv-python
pip install pycocotools==2.0.2
```

```
pip install pafy
```

```
pip install numpy==1.20.0
```

```
cd ..
```

2. Data Preparation

For model development, dataset collection is the first step. 700 images for Kangaroos and 700 images for Wallabies are required for this project.

The following code will download the first 1000 google image search result of Kangaroo

```
#importing the library
from google_images_download import google_images_download

#class instantiation
response = google_images_download.googleimagesdownload()

#creating list of arguments
kangaroo = {"keywords":"Kangaroo","limit":1000,"size": "large","format":"jpg","chromedrive

#passing the arguments to the function
response.download(kangaroo)

#creating list of arguments
wallaby = {"keywords":"Wallaby","limit":1000,"size": "large","format":"jpg","chromedriver"

#passing the arguments to the function
response.download(wallaby)
```

For Kangaroo dataset, only around 450 images can be downloaded using the above code. More images will be collected manually on the internet.

The following code will download the first 1000 google image search result of Wallaby

```
from google_images_download import google_images_download   #importing the library

response = google_images_download.googleimagesdownload()   #class instantiation

arguments = {"keywords":"Wallaby","limit":1000,"size": "large","format":"jpg","chromedrive
response.download(arguments)   #passing the arguments to the function
```

For Wallaby dataset, only around 450 images can be downloaded using the above code. More images will be collected manually on the internet.

After dataset collection, dataset annotation is the second step need to be done.

In order to label the dataset, graphical image annotation tool *LabelImg* will be used in this project

The following code will install LabelImg

```
pip3 install labelImg
```

3. Data Analysis

The following code find the number of sample and sample that labelled

```
import os

# find number of kangaroo image
path, dirs, files = next(os.walk("./dataset/all/kangaroo-wallaby/kangaroo"))
num_kangaroo_image = len(files)
print(num_kangaroo_image) #966

# find number of kangaroo image get labelled
path, dirs, files = next(os.walk("./dataset/all/kangaroo-wallaby/kangaroo/labels"))
num_kangaroo_image = len(files)
print(num_kangaroo_label) #727

# find number of wallaby image
path, dirs, files = next(os.walk("./dataset/all/kangaroo-wallaby/wallaby"))
num_kangaroo_image = len(files)
print(num_wallaby_image) #976

# find number of wallaby image get labelled
path, dirs, files = next(os.walk("./dataset/all/kangaroo-wallaby/wallaby/labels"))
num_kangaroo_image = len(files)
print(num_wallaby_label) #727
```

In the dataset, there are 966 kangaroo images and 976 wallaby image. 727 kangaroo images and 727 wallaby images are labelled.

Since the number of images and label images are similar for kangaroo and wallaby so the data are balance.

There is no missing values in the dataset

The challenge for learning with the dataset is 1454 labeled images may not enough for the model to classify kangaroo and wallaby espically they looks similar.

4. Data Processing

Since all the images have a very different size, all the images are resized

5. Model Development

In this project, three models will be used for the model development. They are Faster R-CNN, RetinaNet and YOLOv5. All the models provides pre-trained model. Therefore, transfer learning will be applied for model development.

The following code will develop models using Faster R-CNN and RetinaNet

```
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import libraries
import numpy as np
import cv2
import random
import os

# import detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog
from detectron2.data.catalog import DatasetCatalog
from cocotrainer import CocoTrainer

from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train", {}, "./dataset/train/_annotations.coco.json",
register_coco_instances("my_dataset_val", {}, "./dataset/valid/_annotations.coco.json", ".
register_coco_instances("my_dataset_test", {}, "./dataset/test/_annotations.coco.json", ".

my_dataset_train_metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")

import random
from detectron2.utils.visualizer import Visualizer
```

```
# for d in random.sample(dataset_dicts, 3):
#     img = cv2.imread(d["file_name"])
#     visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=0
#     vis = visualizer.draw_dataset_dict(d)
#     cv2.imshow('image',vis.get_image()[:, :, ::-1])
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()
```

The following code will use the pre-trained model of Faster R-CNN

```
from detectron2.config import get_cfg
#from detectron2.evaluation.coco_evaluation import COCOEvaluator
import os

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_DC5_1x.yaml
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)

cfg.DATALOADER.NUM_WORKERS = 0
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_DC5_1x.y
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.001
```

The following code will use the pre-trained model of RetinaNet

```
from detectron2.config import get_cfg
#from detectron2.evaluation.coco_evaluation import COCOEvaluator
import os

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_1x.yaml")
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)

cfg.DATALOADER.NUM_WORKERS = 0
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/retinanet_R_50_FPN_1x.yam
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.001
```

```
cfg.SOLVER.WARMUP_ITERS = 1000
cfg.SOLVER.MAX_ITER = 1000 #adjust up if val mAP is still rising, adjust down if overfit
cfg.SOLVER.STEPS = (1000, 1000)
cfg.SOLVER.GAMMA = 0.05





cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 3 #your number of classes + 1

cfg.TEST.EVAL_PERIOD = 500



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

from detectron2.data import DatasetCatalog, MetadataCatalog, build_detection_test_loader
from detectron2.evaluation import COCOEvaluator, inference_on_dataset

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.85
predictor = DefaultPredictor(cfg)
evaluator = COCOEvaluator("my_dataset_test", cfg, False, output_dir="./output/")
val_loader = build_detection_test_loader(cfg, "my_dataset_test")
inference_on_dataset(trainer.model, val_loader, evaluator)

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.DATASETS.TEST = ("my_dataset_test", )
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for this model
predictor = DefaultPredictor(cfg)
test_metadata = MetadataCatalog.get("my_dataset_test")

f = open('./output/config.yaml','w')
f.write(cfg.dump())
f.close()
```

The following code will develop a model using YOLOv5. The code is run on Colab.

Download YOLOv5 and check GPU usage

```
!git clone https://github.com/ultralytics/yolov5 #download Yolov5 into your repo
%cd yolov5
!pip install -r requirements.txt #install All the requirements
import torch
from IPython.display import Image, clear_output  # to display images

clear_output()
print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties
```

Link google drive with colab

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

Copy dataset from google drive to colab

```
!cp -r /content/gdrive/MyDrive/kangaroo-wallaby.zip /content
```

Unzip dataset

```
!unzip /content/kangaroo-wallaby.zip -d /content
```

Start Training

```
!python train.py --img 640 --batch 8 --epochs 1000 --data ./data/kangaroo-wallaby.yaml --w
```

6. Model Evaluation and Selection

Training process visualization

```
%load_ext tensorboard # Tensorboard
%tensorboard --logdir runs/train
%reload_ext tensorboard
```

The tensorboards visualize the training process of the models. The model trained using Faster R-CNN pre-trained model has the highest accuracy. Therefore, the model trained using Faster R-CNN pre-trained model is selected as the best model in the project

6. Apply Model

In order to test the model, the model will be applied to a video that contain kangaroos and wallabies.

The following code will get the apply the model to a YouTube video that contain kangaroos and wallabies.

```
from detectron2.utils.logger import setup_logger
setup_logger()
import cv2
import pafy
import glob
from detectron2.config import get_cfg
from detectron2.data import MetadataCatalog
from detectron2.engine import DefaultPredictor
from detectron2.utils.visualizer import Visualizer
```

```
cfg = get_cfg()
cfg.merge_from_file('./output/config.yaml')
cfg.MODEL.WEIGHTS = "./output/model_final.pth"
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8   # set the testing threshold for this model
predictor = DefaultPredictor(cfg)
print(predictor)
MetadataCatalog.get(cfg.DATASETS.TRAIN[0]).thing_classes = ['kangaroowallaby','kangaroo','
url = 'https://www.youtube.com/watch?v=fc-Lt6Hsgc0&t'
video = pafy.new(url)
best = video.getbest(preftype="mp4")
capture = cv2.VideoCapture(best.url)
fourcc = cv2.VideoWriter_fourcc('X','V','I','D')
out = cv2.VideoWriter('output.mp4', fourcc, 25, (640,480))
count = 0
imageCount = 0
while (True):
    grabbed, im = capture.read()
    if count == 30:
        outputs = predictor(im)
        v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1
        v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        cv2.imshow('', v.get_image()[:, :, ::-1])
        out.write(v.get_image()[:, :, ::-1])
        count=0
        imageCount = imageCount + 1
        cv2.imwrite('./image/'+str(imageCount)+'.jpg', v.get_image()[:, :, ::-1])
    count = count + 1
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
out.release()
capture.release()
cv2.destroyAllWindows()
```

Or you can run a file directly

```
python detect.py
```

This is a output of testing the model

https://www.youtube.com/watch?v=7Khzb-Nmfwc