Video Streaming and Tracking HW2- Object Tracking Report

Author: 313581001 黃得誠

I. Experiment Setup

Data pre-process:

Reference: https://haobin-tan.netlify.app/docs/ai/computer-vision/object-detection/coco-dataset-format/

I wrote a script called "covert2coco.py" to write the dataset information into a json file that meets the coco format. I used three columns in this assignment: "categories", " images", and "annotations".

```python
def Convert2coco(label_dir, output_path):
    coco_data = defaultdict(list)
    image_id = 0
    annotation_id = 0

    # Define categories
    coco_data["categories"] = [{"id": 0, "name": "car"}]
    labels = os.listdir(label_dir)
    labels.sort()
    for label_file in labels:
        if label_file.endswith('.txt'):
            # Extract image info
            image_info = {
                "file_name": label_file.replace('.txt', '.jpg'),
                "height": HEIGHT,
                "width": WIDTH,
                "id": image_id
            }
            coco_data["images"].append(image_info)
```

```python
            # Read annotations from the label file
            with open(os.path.join(label_dir, label_file), 'r') as file:
                lines = file.readlines()

            for line in lines:
                parts = line.strip().split()
                class_id, x_center, y_center, width, height = map(float, parts)

                # Convert normalized positions to absolute (pixel) positions
                abs_x_center = x_center * WIDTH
                abs_y_center = y_center * HEIGHT
                abs_width = width * WIDTH
                abs_height = height * HEIGHT

                # Convert to COCO format (x_min, y_min, width, height)
                x_min = abs_x_center - (abs_width / 2)
                y_min = abs_y_center - (abs_height / 2)
```

```python
                annotation = {
                    "id": annotation_id,
                    "image_id": image_id,
                    "category_id": int(class_id),
                    "bbox": [x_min, y_min, abs_width, abs_height],
                    "area": abs_width * abs_height,
                    "iscrowd": 0
                }
                coco_data["annotations"].append(annotation)
                annotation_id += 1

            image_id += 1

    # Write out the COCO dataset
    with open(output_path, 'w') as json_file:
        json.dump(coco_data, json_file)
```

In addition, I change the name of the datasets.

Dataset:

```
dataset
    annotations
        instances_train.json
        instances_val.json
    test2017
    train_labels
    train2017
    val_labels
    val2017
```

Hyperparameters:

For hyperparameters, check "/exps/example/custom/yolo_s_.py".

I mostly follow the recommendation hyperparameters of the reference, which is in /exps/default/yolo_s_.py.

yolo_s_.py :

```python
class Exp(MyExp):
    def __init__(self):
        self.depth = 0.33
        self.width = 0.50
        # self.exp_name = os.path.split(os.path.realpath(__file__))[1].split(".")[0]
        self.exp_name = "yolox_s_SE"

        # Define yourself dataset path
        self.data_dir = "datasets/dataset"
        self.train_ann = "instances_train.json"
        self.val_ann = "instances_val.json"

        self.num_classes = 1

        self.max_epoch = 100
        self.data_num_workers = 4
        self.eval_interval = 1

        self.basic_lr_per_img = 0.01 / 64.0

        self.save_history_ckpt = False
```

Due to lack of computational resources, I use batch size=16 and train with 100 epochs. I've also tried to change the learning rate due to the change of batch size. However, it doesn't result in better results. Therefore, I finally use the recommended learning rate=0.01/64 * batch size as reference.

seed	None
output_dir	'./YOLOX_outputs'
print_interval	10
eval_interval	1
dataset	None
num_classes	1
depth	0.33
width	0.5
act	'silu'
data_num_workers	4
input_size	(640, 640)
multiscale_range	5
data_dir	'datasets/dataset'
train_ann	'instances_train.json'
val_ann	'instances_val.json'
test_ann	'instances_test2017.json'
mosaic_prob	1.0

mixup_prob	1.0
hsv_prob	1.0
flip_prob	0.5
degrees	10.0
translate	0.1
mosaic_scale	(0.1, 2)
enable_mixup	True
mixup_scale	(0.5, 1.5)
shear	2.0
warmup_epochs	5
max_epoch	100
warmup_lr	0
min_lr_ratio	0.05
basic_lr_per_img	0.00015625
scheduler	'yoloxwarmcos'
no_aug_epochs	15
ema	True

weight_decay	0.0005
momentum	0.9
save_history_ckpt	True
exp_name	'yolox_s'
test_size	(640, 640)
test_conf	0.01
nmsthre	0.65

II. Code Explanation

I added the SE Block and Inception Module in to "yolox/models/network_blocks.py". Then add it respectively to the CSPDarkNet, which is inside the YOLOPAFPN module. YOLO_X is contained of YOLOPAFPN and YOLOXHEAD. Since the two module is used to extract feature, I decided to add them into the FPN part.

SEBlock and InceptionModule.

```python
class SEBlock(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SEBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Linear(channel, channel // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y.expand_as(x)

You, 5 hours ago | 1 author (You)
class InceptionModule(nn.Module):
    def __init__(self, in_channels, ksize=1, stride=1, act="silu"):
        super(InceptionModule, self).__init__()
        self.branch1x1 = BaseConv(in_channels, in_channels // 4, ksize=1, stride=stride, act=act)
        self.branch3x3 = BaseConv(in_channels, in_channels // 4, ksize=3, stride=stride, act=act)
        self.branch5x5 = BaseConv(in_channels, in_channels // 4, ksize=5, stride=stride, act=act)
        self.branch_pool = BaseConv(in_channels, in_channels // 4, ksize=1, stride=stride, act=act)

    def forward(self, x):
        branch1x1 = self.branch1x1(x)
        branch3x3 = self.branch3x3(x)
        branch5x5 = self.branch5x5(x)
        branch_pool = nn.functional.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
        branch_pool = self.branch_pool(branch_pool)
        outputs = [branch1x1, branch3x3, branch5x5, branch_pool]
        return torch.cat(outputs, 1)
```

Add SEBlock into “dark2”, “dark3”, “dark4”, “dark5”, respectively.

```python
        # dark2
        self.dark2 = nn.Sequential(
            Conv(base_channels, base_channels * 2, 3, 2, act=act),
            CSPLayer(
                base_channels * 2,
                base_channels * 2,
                n=base_depth,
                depthwise=depthwise,
                act=act,
            ),
            SEBlock(base_channels * 2),
            # InceptionModule(base_channels*2),
        )
```

After checking the validation score, I chose to use SEBlock only.

Training command:

python tools/train.py -f exps/example/custom/yolox_s.py -d 1 -b 16 --fp16 -o -c weights/yolox_s.pth --cache ram

The model is trained on NVIDIA GeForce RTX 2080 Ti with batch size=16 and pretrained weight: yolox_s.pth.

Inference command:

python tools/demo.py image -f exps/example/custom/yolox_s.py -c weights/best_ckpt_SE.pth --path ./datasets/dataset/test2017/ --conf 0.5 --nms 0.45 --save_result --device gpu --fp1

Used best_ckpt_SE.pth checkpoint to inference.

I modified tools/demo.py to get the required result format.

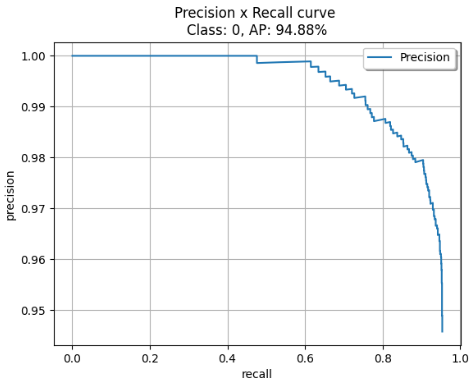
Image_demo() is modified and process_outputs() is added.

```python
def image_demo(predictor, result_folder, path, current_time, save_result):
    result_folder = './myresult/test'
    if os.path.isdir(path):
        files = get_image_list(path)
    else:
        files = [path]
    files.sort()
    for image_name in files:
        outputs, img_info = predictor.inference(image_name)
        # process the outputs to suitable format
        results = process_outputs(outputs[0], img_info, predictor)

        if save_result:
            # Save the results to a text file instead of an image
            txt_file_name = os.path.join(result_folder, os.path.splitext(os.path.basename(image_name))[0] + ".txt")
            with open(txt_file_name, 'w') as f:
                for item in results:
                    f.write("%s\n" % ' '.join(map(str, item)))
            logger.info("Results saved to {}".format(txt_file_name))
```

```python
def process_outputs(outputs, img_info, predictor):
    results = []
    ratio = img_info["ratio"]
    if outputs is None:
        return results

    outputs = outputs.cpu()

    # Process each detection
    for output in outputs:
        # Extract data
        bbox = output[0:4]  # The bounding box
        bbox /= ratio
        score = output[4] * output[5]  # The confidence score
        cls = output[6]  # The class

        left = bbox[0].item()
        top = bbox[1].item()
        right = bbox[2].item()
        bottom = bbox[3].item()

        # Add to results
        results.append([0, score.item(), int(left), int(top), int(right), int(bottom)])
```

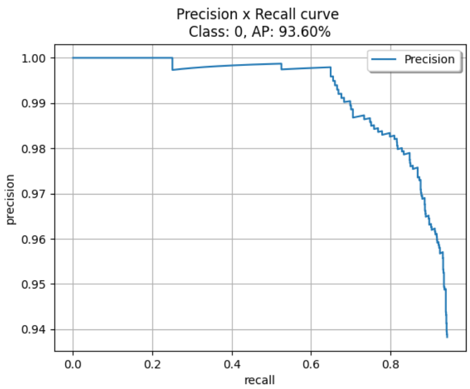
III. Screenshot of validation results

With SE block:



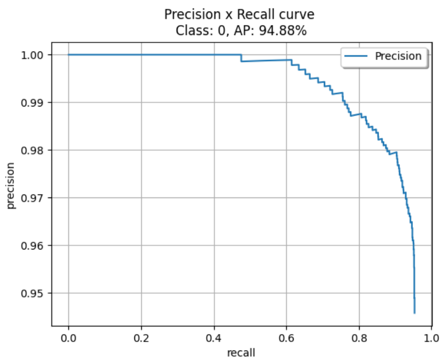

```
AP: 94.62% (0)
mAP: 94.62%
```

With Inception module:




```
AP: 93.60% (0)
mAP: 93.60%
```

Original:




```
AP: 94.88% (0)
mAP: 94.88%
```

Result: Original(Without adding extra module) > With SE module > With Inception module.

The test result is generated by the checkpoint with SE module.

IV. Reference

1. https://github.com/Megvii-BaseDetection/YOLOX
2. https://github.com/rafaelpadilla/Object-Detection-Metrics
3. https://haobin-tan.netlify.app/docs/ai/computer-vision/object-detection/coco-dataset-format/

checkpoints with SE module: Code/YOLOX/weights/best_ckpt_SE.pth
environment: Code/YOLOX/requirements.txt