CSCI-UA 0480-042 Computer Vision

Homework 3

Enter your name and NetID below.

Name:Yi Yang

NetID: yy2324

The main goals of this assignment include:

1. Giving an introduction to Mask-RCNN
2. Training the predictors for a given dataset
3. Finetuning the entire network for the same dataset

Also accompanying each part, there are a few questions (**12 questions in total**) -- 10 mandatory + 2 extra credit. The first 10 questions are worth 100 points and the extra credit questions are worth 20 points.

Please give your answers in the space provided. This homework has a mix of conceptual and coding questions. You can quickly navigate to coding questions by searching (Ctrl/Cmd-F) for `TODO:`.

Part 1: Introduction to Mask-RCNN

[Mask-RCNN (https://arxiv.org/pdf/1703.06870.pdf)](https://arxiv.org/pdf/1703.06870.pdf) is a network used for instance segmentation. Instance segmentation can be thought of as a hybrid of semantic segmentation and object detection. In other words, we don't want to just find the bounding boxes for each object in our image, we're also interested in finding the segmentation mask of *each object instance*.

Instance Segmentation as a Hybrid of Semantic Segmentation and Object Detection

Image Credits: [https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49 (https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49)](https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49)

Mask-RCNN is built on top of Faster-RCNN, which is a network used for object detection. Faster-RCNN has 2 outputs for each candidate object (Region of Interest or RoI) - a class label and a bounding box offset. Mask-RCNN adds a third branch to Faster-RCNN for predicting segmentation masks on each RoI.

We'll first briefly go over Faster-RCNN. Faster-RCNN has 2 stages:

1. **Region Proposal Network (RPN):** Given the image, it proposes candidate object bounding boxes. Previous object detection models such as RCNN and Fast-RCNN handled this separately from the CNN model. Faster-RCNN takes a different approach -- it integrates these two components into the same network to achieve speedup.
2. **Fast-RCNN:** This stages takes each candidate RoI and extracts features from the image feature vector using RoIPool. Using these RoI features, it performs classification and bounding box regression.

Mask-RCNN framework for instance segmentation

Mask-RCNN has the same 2-stage procedure, but in the 2nd stage, instead of just predicting the classification label and the bounding box offset, it predicts **in parallel** a binary mask for each RoI.

Mask-RCNN relies on a pretrained network (called the "backbone" in the paper) to extract features from the image. These features are fed into the Region Proposal Network (RPN) to generate candidate RoIs. For each RoI candidate, a fixed size RoI feature vector is generated using an RoIAlign layer. This RoI feature map is then provided to the classifier, bounding box predictor and the segmentation mask to generate the final output.

Question 1

Training uses a multi-task loss function. What are the three components in this loss function? Is the loss computed per image or per RoI?

Answer:

Classification loss, bounding box loss, and mask loss. The loss is computed on each sampled RoI.

Question 2

This blog post (https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html) by Lilian Weng gives a nice overview of the object detection (RCNN type) networks. In the blog post, it is mentioned that Mask-RCNN uses RoIAlign instead of RoIPool. Explain briefly in 3-4 lines why this is being done.

Answer

First of all, RoI pooling has a major problem in that it loses a lot of data in the process. Therefore, every time Mask-RCNN uses RoIPool, part of the information about the object is missing, which in turns lowers the precision of the model. RoI Align on the other hand, does not use quantization for data pooling. Compared to Fast RCNN that uses quantization twice, mask RCNN is faster with RoI Align, as well as more precise than RoI Pool.

Question 3

What are the different backbones explored in the Mask-RCNN paper? They are denoted in the paper using network-depth-features nomenclature. What is the advantage of using a ResNet-FPN backbone over a ResNet-C4 backbone for feature extraction?

Answer

In the paper, ResNet-101-C4, ResNet-101-FPN, ResNeXt-101-FPN are explored. FPN uses a top-down architecture with lateral connections to build an in-network feature pyramid from a single-scale input, and it is advantageous to use FPN because it gives excellent gains in both accuracy and speed.

Part 2: Training the Predictors for a New Dataset

In this section, we'll start with a pretrained Mask-RCNN model that uses Resnet-50-FPN as the backbone. This model was trained on MS-COCO dataset which is widely used for multiple vision tasks such as object detection, instance segmentation, etc.

MS-COCO has 91 classes (90 for objects + 1 for background). Some sample objects in the dataset include `person` , `car` , `bicycle` , `knife` , `train` , etc.

Along with this homework file, we have also provided another sample dataset (we'll refer to it as the [Nature dataset (https://towardsdatascience.com/custom-instance-segmentation-training-with-7-lines-of-code-ff340851e99b)](https://towardsdatascience.com/custom-instance-segmentation-training-with-7-lines-of-code-ff340851e99b)). It isn't a standard dataset, but it's small enough (600 train + 200 test images) and allows us to easily demo finetuning a pretrained Mask-RCNN model. This dataset contains only 2 classes - `squirrel` and `butterfly` .

Our goal in part2 and part3 of this assignment is to take the pretrained Mask-RCNN model and finetune/train it for this dataset. However, here in part2, instead of finetuning the entire network, we'll train only the final layers.

In Homework 2, we've shown how one could feed data into the network using `Dataset` s and `DataLoader` s. We'll use the same strategy here for finetuning the model.

We've based this homework on this [PyTorch tutorial on Object Detection Finetuning (https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html)](https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html).

In [40]:
```
import numpy as np
import torch
import torchvision

import json # for reading from json file
import glob # for listing files inside a folder
from PIL import Image, ImageDraw # for reading images and drawing masks on the
m.


# Create a custom dataset class for Nature
# dataset subclassing PyTorch's Dataset class
class NatureDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms):
        self.transforms = transforms

        # Load all image files, sorting them to
        # ensure that they are aligned with json files
        imgs = glob.glob(root + '/*.jpg')
        imgs += glob.glob(root + '/*.png') # some images are in png format
        self.imgs = sorted(imgs)

        # Mask data is stored in a json file
        masks = glob.glob(root + '/*.json')
        self.masks = sorted(masks)

        # Each image can have multiple object instances, and each
        # instance is associated with either of these 2 labels.

        # Need to convert str-labels to ids. So we'll use
        # this label-to-index mapping.
        # Note: we can't start from 0 because 0 is restricted
        # to the "background" class
        self.label_to_id = {'squirrel': 1, 'butterfly': 2}


    def __getitem__(self, idx):
        # Have already aligned images and JSON files; can now
        # simply use the index to access both images and masks
        img_path = self.imgs[idx]
        mask_path = self.masks[idx]

        # Read image using PIL.Image and convert it to an RGB image
        img = Image.open(img_path).convert("RGB")

        # TODO: Read image height, width and mask data from
        # the JSON file
        with open(mask_path, 'r') as fp:
            # TODO: Using json library read the dictionary
            # from the fp
            json_dict = json.load(fp)

            # TODO:
            height = img.height

            # TODO:
```

```
        width = img.width

        # TODO:
        poly_shapes_data = np.array(img)

    # TODO: Each image can have multiple mask instances.
    # Using the polygon points, generate the 2d-mask
    # using PIL's ImageDraw.polygon
    masks = []
    labels = []
    for shape_data in poly_shapes_data:
        polygon_points = [tuple(point) for point in shape_data['points']]

        # TODO: Using Image.new() create an image of size (width, height)
        # and fill it with 0s.
        mask_img = Image.new('L', (width, height), 0)

        # TODO: Draw the mask on the base image we just created
        ImageDraw.Draw(img).polygon(polygon_points, outline=1, fill=1)

        mask = np.array(mask_img)
        masks.append(mask)

        label = shape_data['label']
        labels.append(label)

    # Each mask instance also has an associated label which is str-type
    # Convert the str into an int using the mapping we created in __init__
    labels = [self.label_to_id[label] for label in labels]

    # TODO: Generate the bounding boxes for each instance
    # from the 2d masks
    num_objs = len(masks)
    boxes = []
    for i in range(num_objs):
        # TODO: Use np.where() to find where masks[i] == True.
        # pos will be a 2d-list of indices
        pos = np.where(masks[i])

        # In pos, find the min x- and y- indices;
        # max x- and y- indices. This will give us our box bounds.

        # TODO:
        xmin = np.min(pos[1])

        # TODO:
        xmax = np.max(pos[1])

        # TODO:
        ymin = np.min(pos[0])

        # TODO:
        ymax = np.max(pos[0])

        boxes.append([xmin, ymin, xmax, ymax])

    # Convert everything into a torch.Tensor
```

```
        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)
        masks = torch.as_tensor(masks, dtype=torch.uint8)

        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])

        # Assume all instances are not crowd
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        # Apply transforms
        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target


    def __len__(self):
        return len(self.imgs)
```

Question 4

Complete the TODO sections in the above block. Specifically:

1. Read image height, width and polygon shapes data from the JSON file.
2. Generate 2D masks from the polygon points. You can follow this idea: https://stackoverflow.com/a/3732128 (https://stackoverflow.com/a/3732128)
3. Generate the bounding boxes from the mask data. Assume that the bounding box is a rectangle with the smallest area enclosing the mask.

You may find this json schema useful:

```
{
   "shapes": [ # list of object instances; masks are represented as polygons
       ## data for instance1
       {
           "label": "" # label for instance1
           "points": [] # 2d list of polygon points [(x1, y1), (x2, y2), ..]
       },
       ## data for instance2
       {
           "label": []
           "points": []
       },

       ..
       ..
   ],
   "imagePath": ""
   "imageData" : ""
   "imageHeight": <integer>
   "imageWidth": <integer>
}
```

In [41]:
```
# Alternatively, you could also uncomment the line below to see a sample json
 file
!cat '../shared/datasets/nature-dataset/train/s (3).json'
```

```
}
```

Having implemented our `NatureDataset` class, let's create the `Dataset` and the `DataLoader` objects. Note that we're not using `torchvision.transforms` , instead we're using transforms provided in a separate script in this directory called `transforms.py` .

In [42]:
```
import transforms as T

def get_transform(train):
    transforms = []
    transforms.append(T.ToTensor())
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)

# use our dataset and defined transformations
dataset = NatureDataset('../shared/datasets/nature-dataset/train', get_transfo
rm(train=True))
dataset_test = NatureDataset('../shared/datasets/nature-dataset/test', get_tra
nsform(train=False))

import utils

# define training and validation data loaders
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=2, shuffle=True,
    collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test, batch_size=1, shuffle=False,
    collate_fn=utils.collate_fn)
```

Now let's visualize one image from our dataset.

In [43]:
```
import matplotlib.pyplot as plt

img, targets = dataset[506]

# np.transpose docs: https://numpy.org/doc/stable/reference/generated/numpy.tr
anspose.html
# img is a PyTorch tensor, can convert it to a NumPy tensor by calling .numpy
() on it.
plt.imshow(np.transpose(img.numpy(), (1, 2, 0)));
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-43-607bc7d52e44> in <module>
      1 import matplotlib.pyplot as plt
      2 
----> 3 img, targets = dataset[506]
      4 
      5 # np.transpose docs: https://numpy.org/doc/stable/reference/generate
d/numpy.transpose.html

<ipython-input-40-ee7348cd3a9b> in __getitem__(self, idx)
     65         labels = []
     66         for shape_data in poly_shapes_data:
---> 67             polygon_points = [tuple(point) for point in shape_data['p
oints']]
     68 
     69             # TODO: Using Image.new() create an image of size (width,
height)

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`No
ne`) and integer or boolean arrays are valid indices
```

And here's the corresponding mask.

In [13]:
```
plt.imshow(np.transpose(targets['masks'].numpy(), (1, 2, 0)), interpolation='n
one');
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-34effe4de12c> in <module>
----> 1 plt.imshow(np.transpose(targets['masks'].numpy(), (1, 2, 0)), interpo
lation='none');

NameError: name 'targets' is not defined
```

We'll be training the final layers of the pretrained Mask-RCNN model (with Resnet-50-FPN backbone) available in the `torchvision` package. So let's download the model:

In [33]:
```
model_p2 = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/maskrcnn_resnet50_fpn_coco-
bf2d0c1e.pth" to /home/jovyan/.cache/torch/hub/checkpoints/maskrcnn_resnet50_
fpn_coco-bf2d0c1e.pth
66.1%IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)

85.0%IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)

100.0%
```

Question 5

Recall what we said earlier: For training for the new dataset, we need to modify its box predictor (FastRCNNPredictor) and its mask predictor (MaskRCNNPredictor) to match with the new dataset. Complete the code cell below.

You may find these docs for `FastRCNNPredictor` and `MaskRCNNPredictor` useful:

FastRCNNPredictor Docs

MaskRCNNPredictor Docs

In [34]:
```
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor

# Our new dataset has 3 classes: butterfly, squirrel and background
num_classes = 3

# Get number of input features for the classifier
in_features = model_p2.roi_heads.box_predictor.cls_score.in_features

# TODO: replace the pre-trained head with a new one
model_p2.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Get number of input features for the mask predictor
in_features_mask = model_p2.roi_heads.mask_predictor.conv5_mask.in_channels
hidden_layer = 256

# TODO: replace the mask predictor with a new one
model_p2.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask, hidden
_layer, num_classes)
```

Training these layers can take several minutes on a CPU, so we've provided GPUs to make this faster. It should take ~5 mins to run the training in Part2. But before that, we want to ensure that PyTorch is able to access the GPU by printing the device PyTorch is currently (prints `cuda` if it's using a GPU, otherwise it prints `cpu`).

Now we want to freeze all the layers below these predictors. We can do this by setting the `.requires_grad` attribute of the parameters we want to freeze to `False`. Read more about `requires_grad` from [this PyTorch page on Autograd mechanics (https://pytorch.org/docs/stable/notes/autograd.html#excluding-subgraphs-from-backward)](https://pytorch.org/docs/stable/notes/autograd.html#excluding-subgraphs-from-backward).

In order to do the computation on a GPU, we have to move the model from main memory to GPU memory. This can be done by simply calling `.to(device)` on the model. See [the docs (https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module.to)](https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module.to) for more information.

In [35]:
```
import itertools

# Freeze model and move it to device
device = torch.device('cuda') if torch.cuda.is_available() else torch.device(
'cpu')
print("Using", device)

for param in model_p2.parameters():
    param.requires_grad = False

pred_params = itertools.chain(
    model_p2.roi_heads.mask_predictor.parameters(),
    model_p2.roi_heads.box_predictor.parameters()
)

for param in pred_params:
    param.requires_grad = True

model_p2 = model_p2.to(device)
```

```
Using cpu
```

We were able to verify that PyTorch is able to access a GPU. Now let's see the layers inside the `model_p2.roi_heads` to understand what we have modified here (we just modified `box_predictor` and `mask_predictor`). You could also verify the output below from figure 4 in the Mask-RCNN paper. You'll notice that it's the exact same network on the right part of that figure.

In [36]:
```
model_p2.roi_heads
```

Out[36]:
```
RoIHeads(
  (box_roi_pool): MultiScaleRoIAlign()
  (box_head): TwoMLPHead(
    (fc6): Linear(in_features=12544, out_features=1024, bias=True)
    (fc7): Linear(in_features=1024, out_features=1024, bias=True)
  )
  (box_predictor): FastRCNNPredictor(
    (cls_score): Linear(in_features=1024, out_features=3, bias=True)
    (bbox_pred): Linear(in_features=1024, out_features=12, bias=True)
  )
  (mask_roi_pool): MultiScaleRoIAlign()
  (mask_head): MaskRCNNHeads(
    (mask_fcn1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (relu1): ReLU(inplace=True)
    (mask_fcn2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (relu2): ReLU(inplace=True)
    (mask_fcn3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (relu3): ReLU(inplace=True)
    (mask_fcn4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (relu4): ReLU(inplace=True)
  )
  (mask_predictor): MaskRCNNPredictor(
    (conv5_mask): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2,
2))
    (relu): ReLU(inplace=True)
    (mask_fcn_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

Question 6

We just printed the head architecture. From the above output, list all the layers we're training along their names.

For example, if we're training mask_fcn1 of mask_head you'll specify:

`mask_head.mask_fcn1` : 2d-Conv layer with 256 input channels, 256 output channels and kernel size = (3, 3).

Note: We're **only** asking for layers with trainable parameters.

Answer

box_head.fc6: linear with 12544 in features and 1024 out features

box_head.fc7: linear with 1024 in features and 1024 out feature

mask_head.mask_fcn1: 2d-Conv layer with 256 input channels, 256 output channels and kernel size = (3, 3)

mask_head.mask_fcn2: 2d-Conv layer with 256 input channels, 256 output channels and kernel size = (3, 3)

mask_head.mask_fcn3: 2d-Conv layer with 256 input channels, 256 output channels and kernel size = (3, 3)

mask_head.mask_fcn4: 2d-Conv layer with 256 input channels, 256 output channels and kernel size = (3, 3)

Both the dataloaders and model have been prepared for training. All that remains is to set an optimizer and a learning rate scheduler. When we create an optimizer, we have to provide it the list of trainable parameters.

In [38]:
```
# Declare optimizer and lr
params = [p for p in model_p2.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.000
5)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

Question 7

How many trainable parameters are we passing to the optimizer?

Here's an example to calculate # of trainable parameters in a fully connected layer with:

1. an additive bias
2. in_channels = 1024
3. out_channels = 10

The number of trainable parameters here will be 1024*10 + 10 = 10250.

Answer

12544*1024+1024+1024*1024+1024+256*256*4=14,157,824

Now we can finally start the training process.

In [39]:
```
num_epochs = 3

from engine import train_one_epoch, evaluate

for epoch in range(num_epochs):
    print("Epoch", epoch)
    # train for one epoch, printing every 10 iterations
    train_one_epoch(model_p2, optimizer, data_loader, device, epoch, print_fre
q=10)
    # update the learning rate
    lr_scheduler.step()
    # evaluate on the test dataset
    evaluate(model_p2, data_loader_test, device=device)
```

```
Epoch 0
```

```
---------------------------------------------------------------------------
UnidentifiedImageError                    Traceback (most recent call last)
<ipython-input-39-fdfcd67d49af> in <module>
      6     print("Epoch", epoch)
      7     # train for one epoch, printing every 10 iterations
----> 8     train_one_epoch(model_p2, optimizer, data_loader, device, epoch,
 print_freq=10)
      9     # update the learning rate
     10     lr_scheduler.step()

~/hw3/engine.py in train_one_epoch(model, optimizer, data_loader, device, epo
ch, print_freq)
     24         lr_scheduler = utils.warmup_lr_scheduler(optimizer, warmup_it
ers, warmup_factor)
     25 
---> 26     for images, targets in metric_logger.log_every(data_loader, print
_freq, header):
     27         images = list(image.to(device) for image in images)
     28         targets = [{k: v.to(device) for k, v in t.items()} for t in t
argets]

~/hw3/utils.py in log_every(self, iterable, print_freq, header)
    207             ])
    208         MB = 1024.0 * 1024.0
--> 209         for obj in iterable:
    210             data_time.update(time.time() - end)
    211             yield obj

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/dataload
er.py in __next__(self)
    433         if self._sampler_iter is None:
    434             self._reset()
--> 435         data = self._next_data()
    436         self._num_yielded += 1
    437         if self._dataset_kind == _DatasetKind.Iterable and \

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/dataload
er.py in _next_data(self)
    473     def _next_data(self):
    474         index = self._next_index()  # may raise StopIteration
--> 475         data = self._dataset_fetcher.fetch(index)  # may raise StopIt
eration
    476         if self._pin_memory:
    477             data = _utils.pin_memory.pin_memory(data)

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/_utils/f
etch.py in fetch(self, possibly_batched_index)
     42     def fetch(self, possibly_batched_index):
     43         if self.auto_collation:
---> 44             data = [self.dataset[idx] for idx in possibly_batched_ind
ex]
     45         else:
     46             data = self.dataset[possibly_batched_index]

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/_utils/f
etch.py in <listcomp>(.0)
     42     def fetch(self, possibly_batched_index):
```

```
     43         if self.auto_collation:
---> 44             data = [self.dataset[idx] for idx in possibly_batched_ind
ex]
     45         else:
     46             data = self.dataset[possibly_batched_index]

<ipython-input-25-8921081bf879> in __getitem__(self, idx)
     57
     58             # TODO:
---> 59             poly_shapes_data = np.array(Image.open(mask_path))
     60
     61         # TODO: Each image can have multiple mask instances.

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/PIL/Image.py in open(fp,
 mode, formats)
   2956     for message in accept_warnings:
   2957         warnings.warn(message)
-> 2958     raise UnidentifiedImageError(
   2959         "cannot identify image file %r" % (filename if filename else
 fp)
   2960     )

UnidentifiedImageError: cannot identify image file '../shared/datasets/nature
-dataset/train/s (227).json'
```

In order to analyze the output, you'll need to know what an IoU score is. You can read this blog: [https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)](https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)

Question 8

In an image (grey box) of size 12x16, there is an object whose ground truth mask is the box with the dashed edge (green color) and the model's predicted mask is the box with the dash-dotted edge (red color). Calculate the IoU score for this prediction.

Image

Answer

Area of overlap = 5x4=20, area of union = 7x5+5x1=40. So IoU=20/40=1/2

Let's see the model's prediction for a sample from the test set.

In [44]:
```
img, target = dataset_test[1]

# put the model in evaluation mode
model_p2.eval()

with torch.no_grad():
    prediction = model_p2([img.to(device)])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-44-609c428988d9> in <module>
----> 1 img, target = dataset_test[1]
      2 
      3 # put the model in evaluation mode
      4 model_p2.eval()
      5 

<ipython-input-40-ee7348cd3a9b> in __getitem__(self, idx)
     65         labels = []
     66         for shape_data in poly_shapes_data:
---> 67             polygon_points = [tuple(point) for point in shape_data['p
oints']]
     68 
     69             # TODO: Using Image.new() create an image of size (width,
height)

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`No
ne`) and integer or boolean arrays are valid indices
```

Here's the sample image.

In [ ]:
```
Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

And here's the mask generated by the finetuned model.

In [ ]:
```
Image.fromarray(prediction[0]['masks'][0, 0].mul(255).byte().cpu().numpy())
```

Part 3: Finetuning the Entire Network

Let's see what happens when we fine-tune the entire network. That is, instead of just learning the weights in the final layers, we'll let the weights of the entire network change during the training process.

In [45]:
```
model_p3 = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)
```

Question 9

Just like in question 5, we're interested in fine-tuning the pretrained model for our new dataset, so we will again need to modify its box predictor (FastRCNNPredictor) and its mask predictor (MaskRCNNPredictor) to match with our new dataset. Complete the code cell below. Hint: This is exactly the same as question 5.

In [46]:
```
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor

# Our new dataset has 3 classes: butterfly, squirrel and background
num_classes = 3

# Get number of input features for the classifier
in_features = model_p3.roi_heads.box_predictor.cls_score.in_features

# TODO: replace the pre-trained head with a new one
model_p3.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Get number of input features for the mask predictor
in_features_mask = model_p3.roi_heads.mask_predictor.conv5_mask.in_channels
hidden_layer = 256

# TODO: replace the mask predictor with a new one
model_p3.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask, hidden
_layer, num_classes)
```

Again, let's ensure that we're using the GPU by printing the device info. This finetuning process takes even longer because we have more trainable parameters, hence there will be more computations.

In [47]:
```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device(
'cpu')
print("Using", device)

model_p3 = model_p3.to(device)
```

```
Using cpu
```

We'll use the same optimizer and learning rate scheduler as before.

In [48]:
```
params = [p for p in model_p3.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.000
5)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

Let's start the finetuning process.

In [49]:
```
num_epochs = 3

from engine import train_one_epoch, evaluate

for epoch in range(num_epochs):
    print("Epoch", epoch)
    # train for one epoch, printing every 10 iterations
    train_one_epoch(model_p3, optimizer, data_loader, device, epoch, print_fre
q=10)
    # update the learning rate
    lr_scheduler.step()
    # evaluate on the test dataset
    evaluate(model_p3, data_loader_test, device=device)
```

```
Epoch 0
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-49-79b4d7944962> in <module>
      6     print("Epoch", epoch)
      7     # train for one epoch, printing every 10 iterations
----> 8     train_one_epoch(model_p3, optimizer, data_loader, device, epoch,
 print_freq=10)
      9     # update the learning rate
     10     lr_scheduler.step()

~/hw3/engine.py in train_one_epoch(model, optimizer, data_loader, device, epo
ch, print_freq)
     24         lr_scheduler = utils.warmup_lr_scheduler(optimizer, warmup_it
ers, warmup_factor)
     25 
---> 26     for images, targets in metric_logger.log_every(data_loader, print
_freq, header):
     27         images = list(image.to(device) for image in images)
     28         targets = [{k: v.to(device) for k, v in t.items()} for t in t
argets]

~/hw3/utils.py in log_every(self, iterable, print_freq, header)
    207             ])
    208         MB = 1024.0 * 1024.0
--> 209         for obj in iterable:
    210             data_time.update(time.time() - end)
    211             yield obj

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/dataload
er.py in __next__(self)
    433         if self._sampler_iter is None:
    434             self._reset()
--> 435         data = self._next_data()
    436         self._num_yielded += 1
    437         if self._dataset_kind == _DatasetKind.Iterable and \

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/dataload
er.py in _next_data(self)
    473     def _next_data(self):
    474         index = self._next_index()  # may raise StopIteration
--> 475         data = self._dataset_fetcher.fetch(index)  # may raise StopIt
eration
    476         if self._pin_memory:
    477             data = _utils.pin_memory.pin_memory(data)

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/_utils/f
etch.py in fetch(self, possibly_batched_index)
     42     def fetch(self, possibly_batched_index):
     43         if self.auto_collation:
---> 44             data = [self.dataset[idx] for idx in possibly_batched_ind
ex]
     45         else:
     46             data = self.dataset[possibly_batched_index]

/opt/conda/envs/cv_sp21/lib/python3.8/site-packages/torch/utils/data/_utils/f
etch.py in <listcomp>(.0)
     42     def fetch(self, possibly_batched_index):
```

```
     43         if self.auto_collation:
---> 44             data = [self.dataset[idx] for idx in possibly_batched_ind
ex]
     45         else:
     46             data = self.dataset[possibly_batched_index]

<ipython-input-40-ee7348cd3a9b> in __getitem__(self, idx)
     65         labels = []
     66         for shape_data in poly_shapes_data:
---> 67             polygon_points = [tuple(point) for point in shape_data['p
oints']]
     68 
     69             # TODO: Using Image.new() create an image of size (width,
height)

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`No
ne`) and integer or boolean arrays are valid indices
```

We'll generate the output for the same image, but this time with the new finetuned model.

In [50]:
```python
img, target = dataset_test[1]

# put the model in evaluation mode
model_p3.eval()
with torch.no_grad():
    prediction = model_p3([img.to(device)])

Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-50-10948511ac9d> in <module>
----> 1 img, target = dataset_test[1]
      2 
      3 # put the model in evaluation mode
      4 model_p3.eval()
      5 with torch.no_grad():

<ipython-input-40-ee7348cd3a9b> in __getitem__(self, idx)
     65         labels = []
     66         for shape_data in poly_shapes_data:
---> 67             polygon_points = [tuple(point) for point in shape_data['p
oints']]
     68 
     69             # TODO: Using Image.new() create an image of size (width,
height)

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`No
ne`) and integer or boolean arrays are valid indices
```

Here's the predicted image.

In [51]:
```
Image.fromarray(prediction[0]['masks'][0, 0].mul(255).byte().cpu().numpy())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-51-1fca18724991> in <module>
----> 1 Image.fromarray(prediction[0]['masks'][0, 0].mul(255).byte().cpu().nu
mpy())

NameError: name 'prediction' is not defined
```

Question 10

Does this model perform better than the trained model in part2? Explain why.

Answer

As I am unable to find the result due to a mistake in the code, I do believer it will perform better than the trained model in part 2. Because this model uses more data to train, and more data to train will result in better performance.

Part 4: Extra Credit Questions

Question 11 [15 points]

Can fully convolutional networks (FCN) be used for object detection? In Mask-RCNN we have 3 branches — mask, classification, and bounding box regression — out of which the last 2 have fully connected (FC) layers. Can this entire pipeline be replaced by a fully convolutional network? If possible, give 1 or 2 networks to support your claim.

Answer

I think FCN can be used for object detection. More specifically, regional based FCN, or R-FCN has been used in studies to achieve good results in object detection, such as this paper: https://arxiv.org/pdf/1605.06409.pdf (https://arxiv.org/pdf/1605.06409.pdf)

In terms of replacing the entire pipeline by a fully convolutional network, it should be possible if we use the ideas of R-FCN, utilizes position-sensitive score maps to address a dilemma between translation-invariance in image classification and translation-variance in object detection.

Question 12 [5 points]

What is the advantage of using CONV layers over FC/Dense layers?

Answer

Conv layers are very good at extracting useful information, it also retains the data shape, making it easy to visualize and detect local features.

In [ ]:

```

```