DSDE-Homework_3_1_object_detection_vocdetection_fasterrcnn_mobilenet_v3

1. Describe the input and output for each model, hardware requirement, data statistic, learning curve, metrics (train text val), demo the result, finetuning technique, etc.

ใช้ MobileNetV3 ในการทำ Object Detection
fine-tuned ด้วย Pascal VOC dataset ประกอบด้วย 20 class
และวัดด้วย Mean insertion over union

2. List key features for each function, including input and output. (cheat sheet)

เช็ค GPU

```
!nvidia-smi

import torch
import torchvision

device = 'cuda'
boxes = torch.tensor([[0., 1., 2., 3.]]).to(device)
scores = torch.randn(1).to(device)
iou_thresholds = 0.5

print(torchvision.ops.nms(boxes, scores, iou_thresholds))
```

โหลด pretrained Faster R-CNN with MobileNetV3 backbone และปรับให้ detect 21 class โดยเพิ่ม background เข้ามา

```
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

pretrain_weight =
torchvision.models.detection.FasterRCNN_MobileNet_V3_Large_320_FPN_Weights
model_ft =
torchvision.models.detection.fasterrcnn_mobilenet_v3_large_320_fpn(weights
=pretrain_weight)
```

```
model_ft.roi_heads.box_predictor = FastRCNNPredictor(1024, 21)
net = model_ft.to(device)
```

สร้าง mapping dict สำหรับแต่ละ class

```
class_ = ["person", "bird", "cat", "cow", "dog",
"horse", "sheep", "aeroplane", "bicycle", "boat",
"bus", "car", "motorbike", "train", "bottle",
"chair", "diningtable", "pottedplant", "sofa", "tvmonitor"]

textlabel2num = {x: i+1 for i, x in enumerate( class_)}
numlabel2text = {i+1: x for i, x in enumerate( class_)}
textlabel2num
```

```
{'person': 1,
 'bird': 2,
 'cat': 3,
 'cow': 4,
 'dog': 5,
 'horse': 6,
 'sheep': 7,
 'aeroplane': 8,
 'bicycle': 9,
 'boat': 10,
 'bus': 11,
 'car': 12,
 'motorbike': 13,
 'train': 14,
 'bottle': 15,
 'chair': 16,
 'diningtable': 17,
 'pottedplant': 18,
 'sofa': 19,
 'tvmonitor': 20}
```

สร้าง Dataset class

```
import numpy as np
class MyDataset(torch.utils.data.Dataset):
        def __init__(self, dataset):
                self.dataset = dataset
```

```
        def __len__(self):
                return self.dataset.__len__()

        def __getitem__(self, idx):
                X, y = self.dataset.__getitem__(idx)

                labels = []
                boxes = []
                for item in y['annotation']['object']:
                labels.append(textlabel2num[item['name']])
                box = item['bndbox']
                boxes.append([np.float32(box["xmin"]),
                np.float32(box["ymin"]),
                np.float32(box["xmax"]),
                np.float32(box["ymax"])])

                labels = torch.as_tensor(labels,
dtype=torch.int64).to(device)
                boxes = torch.as_tensor(boxes,
dtype=torch.float32).to(device)
                X = X.to(device)

                target = {}
                target["boxes"] = boxes
                target["labels"] = labels

                return X, target
```

สร้าง training, validation, testing loader

```
trainset = torchvision.datasets.VOCDetection(root='./data', year='2007',
image_set='train', download=True, transform=transform_train)
trainset = MyDataset(trainset)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
shuffle=True, collate_fn=collate_fn)

valtestset = torchvision.datasets.VOCDetection(root='./data', year='2007',
image_set='val', download=True, transform=transform)
valtestset = MyDataset(valtestset)
valset, testset = torch.utils.data.random_split(valtestset, [2510//2,
2510//2])#, generator=torch.Generator().manual_seed(2023))

valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size,
shuffle=False, collate_fn=collate_fn)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
shuffle=False, collate_fn=collate_fn)
```

ดูจำนวน train, val, test

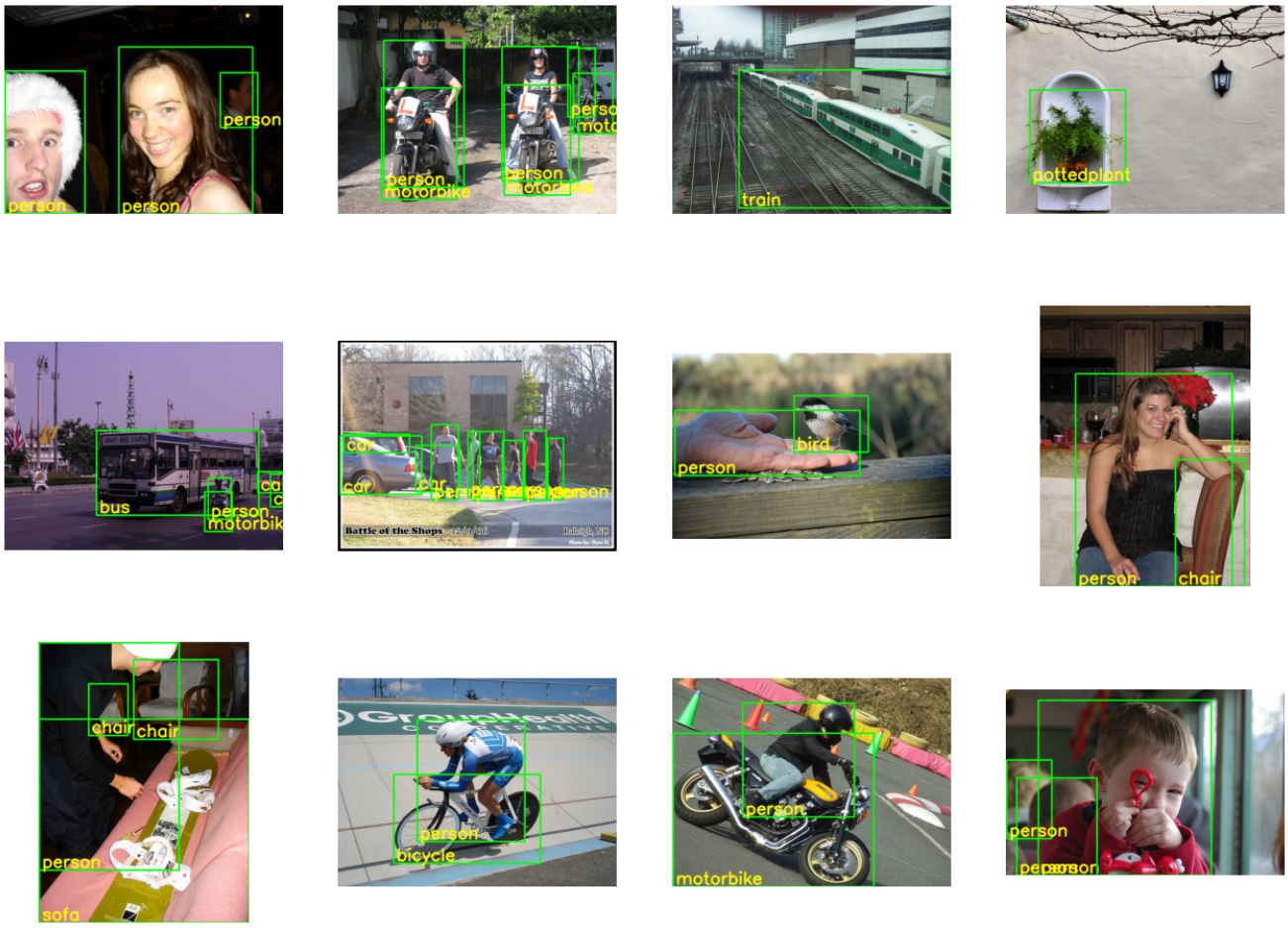
```
trainset.__len__(), valset.__len__(), testset.__len__()
```

```
(2501, 1255, 1255)
```

plot รูปใน dataset

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

# functions to show an image
def imshow(imgs, labels, ncol):
        nrow = len(imgs) // ncol

        fig, ax = plt.subplots(nrow, ncol, figsize=(ncol*4, nrow*4))
        for row in range(nrow):
                for col in range(ncol):
                        if row*ncol + col < len(imgs):
                                img = imgs[row*ncol + col].cpu()*0.5 + 0.5
                                img = img.permute((1, 2,
0)).mul(255).numpy()
                                img = np.ascontiguousarray(img,
dtype=np.uint8)

                                boxes = labels[row*ncol + col]
['boxes'].cpu().numpy()
                                in_labels = labels[row*ncol + col]
['labels'].cpu().numpy()
                                nbox, _ = boxes.shape

                                for i in range(nbox):
                                        img = cv2.rectangle(img = img,
                                                pt1 = (int(boxes[i][0]),
int(boxes[i][1])),
                                                pt2 = (int(boxes[i][2]),
int(boxes[i][3])),
                                                color = (0, 255, 0),
                                                thickness = 2)

                                        img = cv2.putText(img = img,
                                                text =
f'{numlabel2text[in_labels[i]]}',
                                                org = (int(boxes[i][0]) +
5, int(boxes[i][3]) - 5 ),
                                                fontFace =
cv2.FONT_HERSHEY_SIMPLEX,
                                                fontScale = 1,
```

```
                                                color = (255, 255, 0),
                                                thickness = 2,
                                                lineType = cv2.LINE_AA)

                                        ax[row, col].imshow(img)
                                        ax[row, col].axis('off')
                        else:
                                ax[row, col].imshow(np.zeros((200,200, 3),
dtype = np.uint8))
                                ax[row, col].axis('off')
        plt.show()

# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
ncol = 4
imshow(images[:13], labels[:13], ncol)
```



funcion ทำ miou

```
def single_iou(gt_box, pred_box):
        gt_box = gt_box.cpu()
        pred_box = pred_box.cpu()
```

```
        intersec_box = torch.tensor([
        max(gt_box[0], pred_box[0]),
        max(gt_box[1], pred_box[1]),
        min(gt_box[2], pred_box[2]),
        min(gt_box[3], pred_box[3]),
        ])

        intersec_w = max(intersec_box[2] - intersec_box[0],0)
        intersec_h = max(intersec_box[3] - intersec_box[1],0)
        intersec_area = intersec_w*intersec_h

        gt_w = gt_box[2] - gt_box[0]
        gt_h = gt_box[3] - gt_box[1]
        gt_area = gt_w*gt_h

        pred_w = pred_box[2] - pred_box[0]
        pred_h = pred_box[3] - pred_box[1]
        pred_area = pred_w*pred_h

        iou = intersec_area/(pred_area+gt_area-intersec_area)

        return iou
def miou(pred, gt):
        mIoU = 0
        for i in range(len(gt)):
        pred_ = pred[i]
        gt_ = gt[i]
        mIoU_image = 0
        n_box = 0

        for pred_box, pred_label in zip(pred_['boxes'], pred_['labels']):
        max_iou = 0
        for gt_box, gt_label in zip(gt_['boxes'], gt_['labels']):
        iou = single_iou(gt_box, pred_box)
        if iou > max_iou:
        max_iou = iou
        mIoU_image += max_iou
        n_box += 1

        for gt_box, gt_label in zip(gt_['boxes'], gt_['labels']):
        max_iou = 0
        for pred_box, pred_label in zip(pred_['boxes'], pred_['labels']):
        iou = single_iou(gt_box, pred_box)
        if iou > max_iou:
        max_iou = iou
        mIoU_image += max_iou
        n_box += 1
```

```
        if n_box:
        mIoU_image /= n_box
        else:
        mIoU_image = 0
        mIoU += mIoU_image
        mIoU /= len(gt)
        return mIoU, len(gt)
```

Optimizer และ scheduler

```
import torch.optim as optim

from torch.optim import lr_scheduler



optimizer = optim.SGD(net.parameters(), lr=0.02, momentum=0.9)

scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
```

train ด้วย 5 Epoch

```
from tqdm.notebook import tqdm

epochs = 5

history_train =
{'loss_classifier':np.zeros(epochs),'loss_box_reg':np.zeros(epochs),'loss_
objectness':np.zeros(epochs),'loss_rpn_box_reg':np.zeros(epochs),
'iou':np.zeros(epochs), 'ap@50':np.zeros(epochs)}

history_val =
{'loss_classifier':np.zeros(epochs),'loss_box_reg':np.zeros(epochs),'loss_
objectness':np.zeros(epochs),'loss_rpn_box_reg':np.zeros(epochs),
'iou':np.zeros(epochs), 'ap@50':np.zeros(epochs)}

max_val_iou = 0

PATH = './VOCDetection-FasterRCNN_MobileNet_V3.pth'



for epoch in range(epochs): # loop over the dataset multiple times



print(f'epoch {epoch + 1} \nTraining ...')
```

```
mIoU = 0

training_loss = 0.0

training_loss_classifier = 0.0

training_loss_box_reg = 0.0

training_loss_objectness = 0.0

training_loss_rpn_box_reg = 0.0

n = 0

with torch.set_grad_enabled(True):

for data in tqdm(trainloader):

# get the inputs; data is a list of [inputs, labels]

inputs, labels = data



# zero the parameter gradients

optimizer.zero_grad()



# forward

net.train()

loss_dict = net(inputs, labels)

loss = sum(x for x in loss_dict.values())



#backward

loss.backward()



#optimize

optimizer.step()
```

```
# find mIoU

with torch.no_grad():

net.eval()

preds = net(inputs)

mIoU_sample, n_sample = miou(preds, labels)


# aggregate statistics

training_loss += loss.item()*n_sample

training_loss_classifier += loss_dict['loss_classifier'].item()*n_sample

training_loss_box_reg += loss_dict['loss_box_reg'].item()*n_sample

training_loss_objectness += loss_dict['loss_objectness'].item()*n_sample

training_loss_rpn_box_reg += loss_dict['loss_rpn_box_reg'].item()*n_sample

mIoU += mIoU_sample*n_sample

n += n_sample


scheduler.step()


# print statistics

training_loss /= n

training_loss_classifier /= n

training_loss_box_reg /= n

training_loss_objectness /= n

training_loss_rpn_box_reg /= n
```

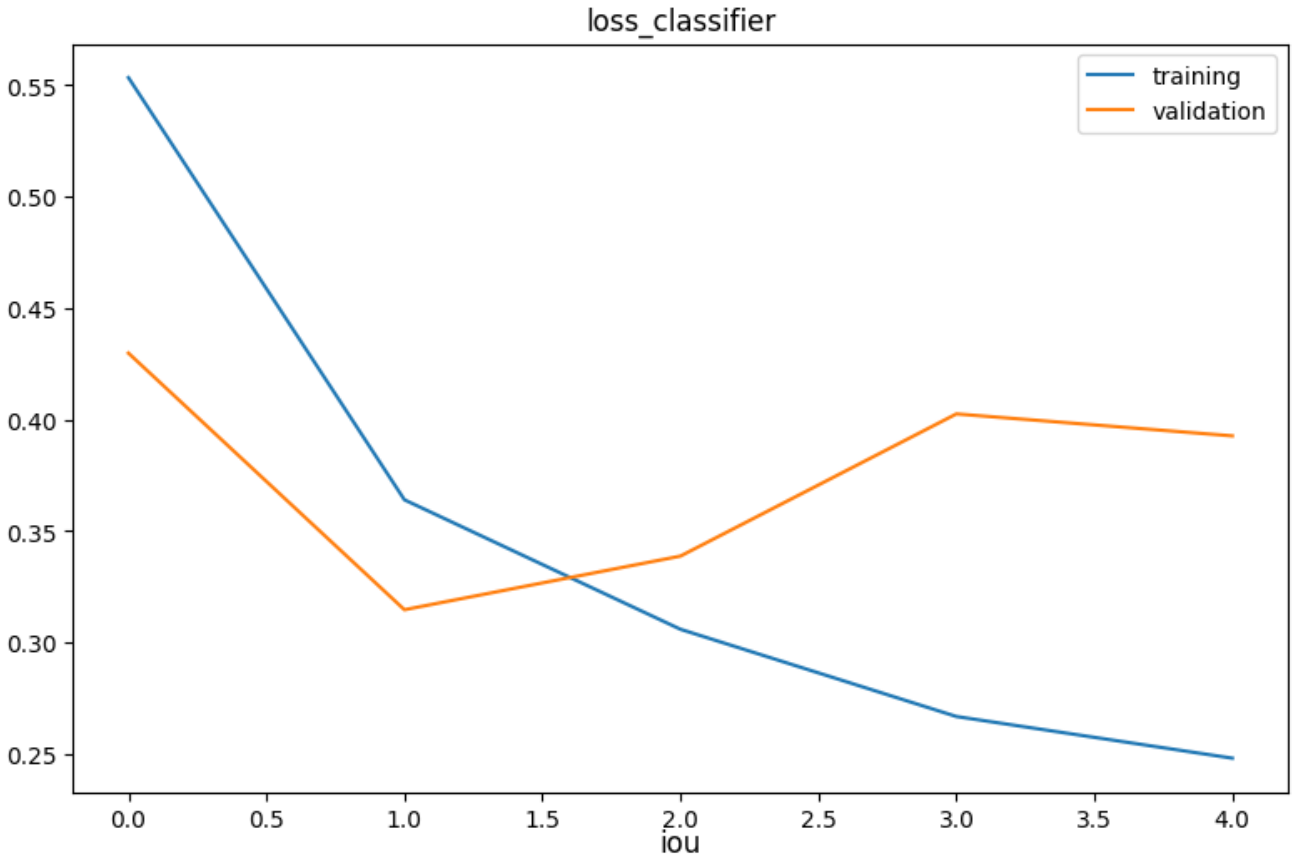
```
mIoU /= n

print(f"total_training loss: {training_loss:.4}, loss_classifier:
{training_loss_classifier:.4}, loss_box_reg: {training_loss_box_reg:.4},
loss_objectness: {training_loss_objectness:.4}, loss_rpn_box_reg:
{training_loss_rpn_box_reg:.4}, mIoU: {mIoU:.4}" )

history_train['loss_classifier'][epoch] = training_loss_classifier

history_train['loss_box_reg'][epoch] = training_loss_box_reg

history_train['loss_objectness'][epoch] = training_loss_objectness

history_train['loss_rpn_box_reg'][epoch] = training_loss_rpn_box_reg

history_train['iou'][epoch] = mIoU



print('validating ...')

mIoU = 0

validation_loss = 0.0

validation_loss_classifier = 0.0

validation_loss_box_reg = 0.0

validation_loss_objectness = 0.0

validation_loss_rpn_box_reg = 0.0

n = 0

with torch.no_grad():

for data in tqdm(valloader):

inputs, labels = data



# find mIoU

net.eval()

preds = net(inputs)
```

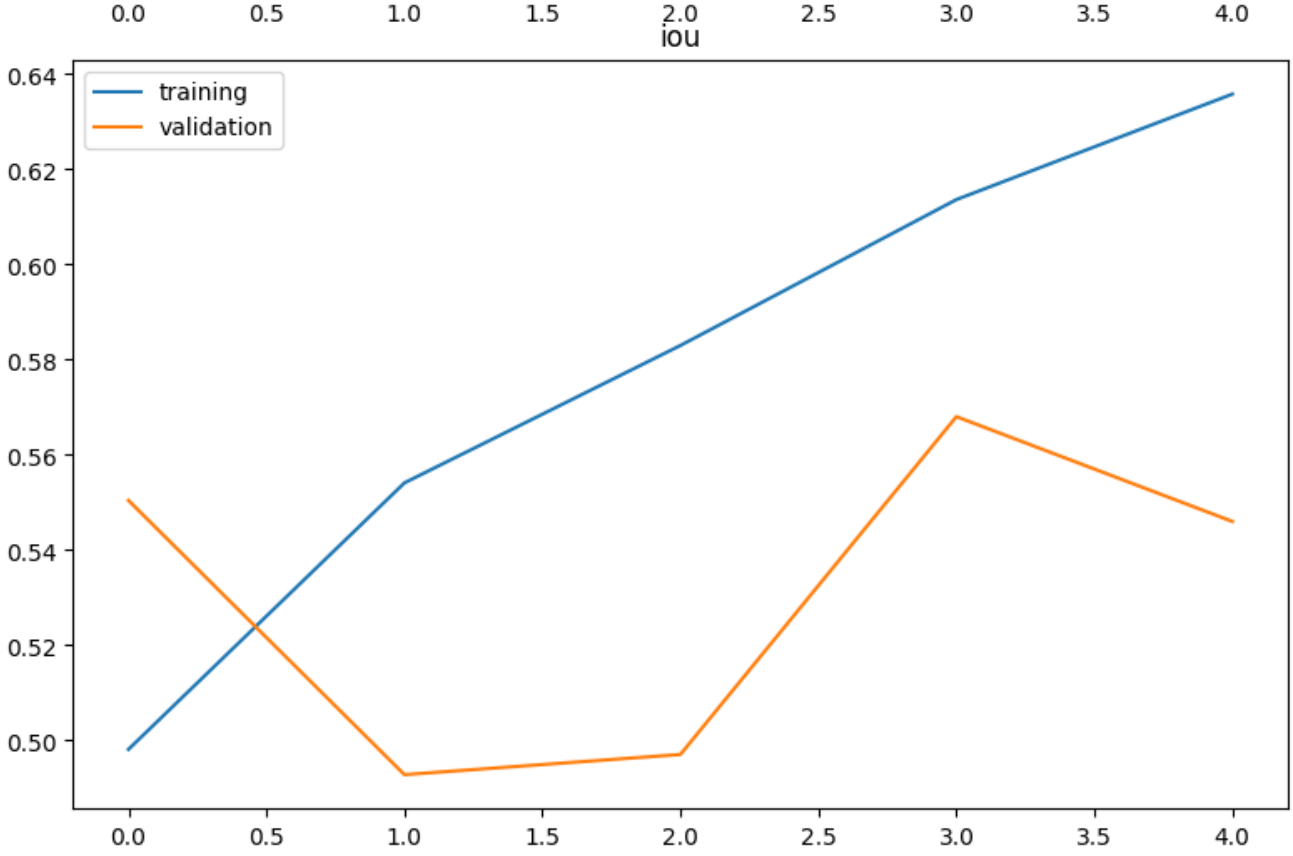
```
mIoU_sample, n_sample = miou(preds, labels)



# loss

net.train()

loss_dict = net(inputs, labels)

loss = sum(x for x in loss_dict.values())



# zero the parameter gradients

optimizer.zero_grad()



# aggregate statistics

validation_loss += loss.item()*n_sample

validation_loss_classifier += loss_dict['loss_classifier'].item()*n_sample

validation_loss_box_reg += loss_dict['loss_box_reg'].item()*n_sample

validation_loss_objectness += loss_dict['loss_objectness'].item()*n_sample

validation_loss_rpn_box_reg +=
loss_dict['loss_rpn_box_reg'].item()*n_sample

mIoU += mIoU_sample*n_sample

n += n_sample



# print statistics

validation_loss /= n

validation_loss_classifier /= n

validation_loss_box_reg /= n

validation_loss_objectness /= n
```

```
validation_loss_rpn_box_reg /= n

mIoU /= n

print(f"total_validation loss: {validation_loss:.4}, loss_classifier:
{validation_loss_classifier:.4}, loss_box_reg:
{validation_loss_box_reg:.4}, loss_objectness:
{validation_loss_objectness:.4}, loss_rpn_box_reg:
{validation_loss_rpn_box_reg:.4}, mIoU: {mIoU:.4}" )

history_val['loss_classifier'][epoch] = validation_loss_classifier

history_val['loss_box_reg'][epoch] = validation_loss_box_reg

history_val['loss_objectness'][epoch] = validation_loss_objectness

history_val['loss_rpn_box_reg'][epoch] = validation_loss_rpn_box_reg

history_val['iou'][epoch] = mIoU



#save min validation loss

if mIoU > max_val_iou:

        torch.save(net.state_dict(), PATH)

        max_val_iou = mIoU



        print('Finished Training')
```

plot statistic การ train

```
fig, axs = plt.subplots(2, figsize= (8,10))
fig.tight_layout()
# loss_classifier
axs[0].plot(history_train['loss_classifier'], label = 'training')
axs[0].plot(history_val['loss_classifier'], label = 'validation')
axs[0].set_title("loss_classifier")
axs[0].legend()
# iou
axs[1].plot(history_train['iou'], label = 'training')
axs[1].plot(history_val['iou'], label = 'validation')
axs[1].set_title("iou")
axs[1].legend()
```





แสดง performance บน test set

```
dataiter = iter(testloader)

images, labels = next(dataiter)

with torch.no_grad():
```

```
net.eval()

preds = net(images)



# show images

ncol = 3

imshow_test(images[:9], preds[:9], labels[:9], ncol)
```