DSDE-Homework 1_Image_classification_CIFAR10_CNN

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.

Input เป็นรูปมี 10 classes
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
จำนวน train, test, val = (40000, 10000, 10000)

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

ดู Status ของ NVIDIA GPU

```
! nvidia-smi
```

ปิด Warning

```
from sklearn.exceptions import UndefinedMetricWarning
def warn(*args, **kwargs):
        pass
import warnings
warnings.warn = warn
```

เลือกใช้ GPU ถ้ามี

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

transformation pipeline that converts images to tensors, normalizes pixel values, resizes them to 32x32 pixels, and sets the batch size to 32 for processing the images in batches
สร้าง Transformation pipeline
เพื่อเปลี่ยนจากรูป เป็น tensors

- ทำ normalize pixel
- resize เป็น 32x32
- ตั้ง batch_size เป็น 32

```
transform = transforms.Compose( # transform is from torchvision (only for image)
[transforms.ToTensor(), # image to tensor --> divide by 255
transforms.Resize((32, 32))])
batch_size = 32
```

- Loads the CIFAR-10 dataset, applies the defined transformations
- Splits the training set into train and validation sets
- สร้าง DataLoader สำหรับ train
- สร้าง DataLoader สำหรับ validation
- สร้าง DataLoader สำหรับ test

```
trainvalset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainset, valset = torch.utils.data.random_split(trainvalset, [40000, 10000])

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, shuffle=False)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)
```

สร้าง Network ประกอบด้วย

- Conv 3 6 5
- ReLU
- MaxPool
- Conv 6 16 5
- ReLU
- MaxPool
- Flatten
- Linear 400 120
- ReLU
- Linear 120 84

- ReLU
- Linear 84 10
- Softmax

```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
        def __init__(self):
                super().__init__()
                self.conv1 = nn.Conv2d(3, 6, 5) # 3 input channels, 6
output channels, 5*5 kernel size
                self.pool = nn.MaxPool2d(2, 2) # 2*2 kernel size, 2
strides
                self.conv2 = nn.Conv2d(6, 16, 5)
                self.fc1 = nn.Linear(400, 120) # dense input 400 (16*5),
output 120
                self.fc2 = nn.Linear(120, 84) # dense input 120, output 84
                self.fc3 = nn.Linear(84, 10) # dense input 84, output 10
                self.softmax = torch.nn.Softmax(dim=1) # perform softmax
at dim[1] (batch,class)

        def forward(self, x):
                x = self.pool(F.relu(self.conv1(x)))
                x = self.pool(F.relu(self.conv2(x)))
                x = torch.flatten(x,start_dim=1) # flatten all dimensions
(dim[1]) except batch (dim[0])
                x = F.relu(self.fc1(x))
                x = F.relu(self.fc2(x))
                x = self.fc3(x)
                x = self.softmax(x)
                return x

net = CNN().to(device)
```

ใช้ torchinfo เพื่อดู Network Summary

```
from torchinfo import summary as summary_info

print(summary_info(net, input_size = (32, 3, 32, 32))) #
(batchsize,channel,width,height)

net = net.to(device)
```

ใช้ CrossEntropyLoss เป็น loss function, และใช้ Stochastic gradient descent เป็น optimizer, Learning rate = 0.01

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=1e-2, momentum=0.9)
```

ใช้ `net.train()` เพื่อเข้าสู้ train mode
ปรับค่า gradient parameter ให้เป็น 0

```
# zero the parameter gradients
optimizer.zero_grad()
```

forward ค่าผ่าน network และคำนวณ loss ด้วย criterion (CrossEntropyLoss)
จากนั้นคำนวน gradient และปรับค่า parameter

```
# forward + backward + optimize
outputs = net(inputs) # forward
loss = criterion(outputs, labels) # calculate loss from forward pass
loss.backward() # just calculate
optimizer.step() # update weights here
```

save model หลังจาก loss converge

```
#save min validation loss
if validation_loss < min_val_loss:
        torch.save(net.state_dict(), PATH)
        min_val_loss = validation_loss
```

plot statistic ระหว่าง train

```
fig, axs = plt.subplots(3, figsize= (6,10))
# loss
axs[0].plot(history_train['loss'], label = 'training')
axs[0].plot(history_val['loss'], label = 'validation')
axs[0].set_title("loss")
axs[0].legend()
# acc
axs[1].plot(history_train['acc'], label = 'training')
axs[1].plot(history_val['acc'], label = 'validation')
axs[1].set_title("acc")
axs[1].legend()
# f1-score
axs[2].plot(history_train['f1-score'], label = 'training')
axs[2].plot(history_val['f1-score'], label = 'validation')
axs[2].set_title("f1-score")
```

```
axs[2].legend()
plt.show()
```

โหลด parameter ที่ save ไว้

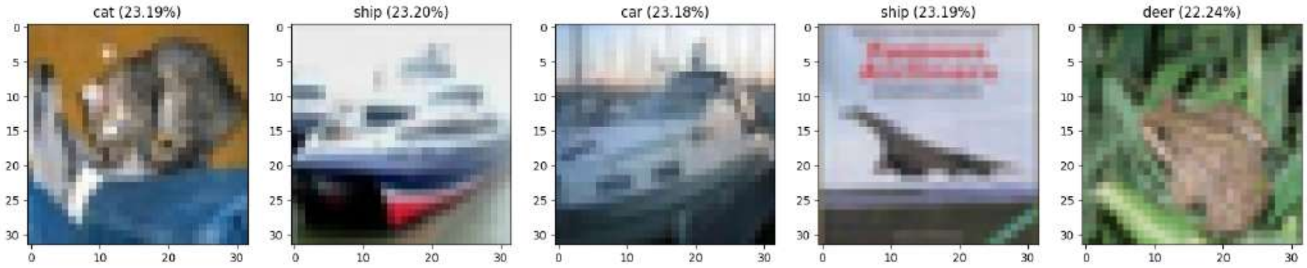
```
net = CNN().to(device)
net.load_state_dict(torch.load(PATH))
```

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

print('testing ...')
y_predict = list()
y_labels = list()
test_loss = 0.0
n = 0
with torch.no_grad():
        for data in tqdm(testloader):
                inputs, labels = data
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = net(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

                y_labels += list(labels.cpu().numpy())
                y_predict += list(outputs.argmax(dim=1).cpu().numpy())
                n+=1

        # print statistics
        test_loss /= n
        print(f"testing loss: {test_loss:.4}" )

        report = classification_report(y_labels, y_predict, digits = 4)
        M = confusion_matrix(y_labels, y_predict)
        print(report)
        disp = ConfusionMatrixDisplay(confusion_matrix=M)
        #acc = report["accuracy"]
        #f1 = report["weighted avg"]["f1-score"]
        #support = report["weighted avg"]["support"]
        #test_loss /= n
        #print(f"validation loss: {test_loss:.4}, acc: {acc*100:.4}%, f1-score: {f1*100:.4}%, support: {support}" )
```

```
disp.plot()
plt.show()
```

plot รูปและ probability ที่ classified ได้

```
plt.figure(figsize=(20,5))
dataiter = iter(testloader)
inputs, labels = next(dataiter)
with torch.no_grad():
        net.eval()
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        y_labels = list(labels.cpu().numpy())
        y_predict = list(outputs.argmax(dim=1).cpu().numpy())
        # To get probabilities, you can run a softmax on outputs
        y_probs = torch.nn.functional.softmax(outputs, dim=1)
        y_probs = list(y_probs.cpu().numpy())

# We selected a sample from the first five images for visualization
for i in range(5):
        plt.subplot(1,5,i+1)
        img = inputs[i] # unnormalize
        npimg = img.cpu().numpy()
        plt.imshow(np.transpose(npimg, (1, 2, 0)))

        most_prob = np.argmax(y_probs[i])
        label = classes[most_prob]
        prob = y_probs[i][most_prob]
        plt.title(f"{label} ({prob*100:.2f}%)")
```



DSDE-Homework 2_Image_classification_Animal_EfficientNetV2

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.

ทำ augmentation รูป animal 10 classes
ใช้ EfficientNetV2 มา fine-tune classification layer ให้เป็น 10 animal classes

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

download animal image dataset

```
!wget https://github.com/pvateekul/2110531_DSDE_2023s1/raw/main/code/Week05_Intro_Deep_Learning/data/Dataset_animal2.zip
!unzip -q -o 'Dataset_animal2.zip'
```

ทำ augmentation rotation, cropping, flipping

```
transform_train = transforms.Compose(
    [transforms.Resize((230,230)),
     transforms.RandomRotation(30,),
     transforms.RandomCrop(224),
     transforms.RandomHorizontalFlip(),
     transforms.RandomVerticalFlip(),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.507, 0.487, 0.441], std=[0.267, 0.256, 0.276])]
)

transform = transforms.Compose(
    [transforms.Resize((224,224)),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.507, 0.487, 0.441], std=[0.267, 0.256,
0.276])]
)
```

สร้าง class สำหรับ dataset

```
class AnimalDataset(Dataset):
    def __init__(self, img_dir, transforms=None):
        self.label_image = ['butterfly', 'cat', 'chicken', 'cow', 'dog',
'elephant', 'horse', 'sheep', 'spider', 'squirrel']
        self.input_dataset = [(os.path.join(img_dir, label, image_name),
label_num)
                              for label_num, label in
enumerate(self.label_image)
                              for image_name in
os.listdir(os.path.join(img_dir, label))]
        self.transforms = transforms

    def __len__(self):
        return len(self.input_dataset)

    def __getitem__(self, idx):
        img = Image.open(self.input_dataset[idx][0]).convert('RGB')
        x = self.transforms(img)
        y = self.input_dataset[idx][1]
        return x, y
```

แบ่ง train, validate, test และสร้าง Dataloader

```
trainset = AnimalDataset('./Dataset_animal2/train', transform_train)
valset = AnimalDataset('./Dataset_animal2/val', transform)
testset = AnimalDataset('./Dataset_animal2/test', transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=32,
shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=32,
shuffle=True)
```

function สำหรับ plot รูปในแต่ละคลาส

```
def PlotRandomFromEachClass(dataset, N, labels):
    Y = [label for _, label in dataset.input_dataset]
    M = len(np.unique(Y))
    plt.figure(figsize=(16, N*1.5))
```

```
  for i in range(M):
    indexes = np.random.choice(np.where(np.array(Y) == i)[0], N,
replace=False)
    for j in range(N):
      img = Image.open(dataset.input_dataset[indexes[j]]
[0]).convert('RGB')
      plt.subplot(N, M, j*M + i + 1)
      plt.imshow(img)
      plt.axis("off")
      if j == 0:
        plt.title(labels[i])
```

รูปที่ผ่านการทำ Augmentation

ดูจำนวนใน train, val และ test

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

(1400, 300, 300)

โหลด pretrain weight EfficientNetV2

```
pretrain_weight =
torchvision.models.EfficientNet_V2_S_Weights.IMAGENET1K_V1
net = torchvision.models.efficientnet_v2_s(weights=pretrain_weight)
net.classifier[1] = nn.Linear(1280, 10)
net = net.to(device)
```

ดูสรุป network parameters

```
from torchsummary import summary
summary(net, (3, 224, 224), batch_size = 64)
```

```
================================================================ Total
params: 20,190,298 Trainable params: 20,190,298 Non-trainable params: 0 --
-------------------------------------------------------------- Input size
(MB): 36.75 Forward/backward pass size (MB): 20629.03 Params size (MB):
77.02 Estimated Total Size (MB): 20742.80 -------------------------------
---------------------------------
```

กำหนด loss function
ใช้ SGD เป็น optimizer
และใช้ learning rate schedule

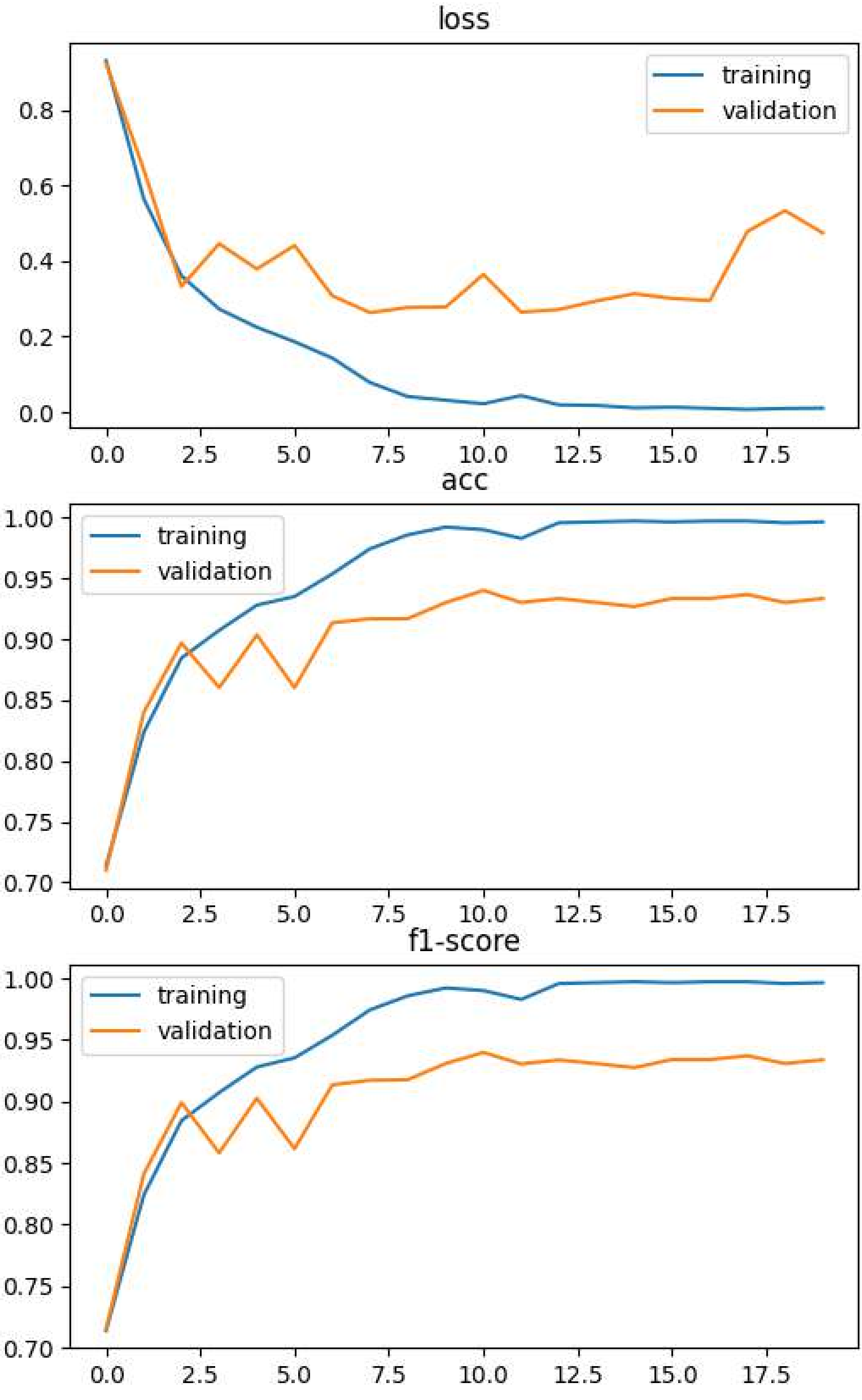
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.02, momentum=0.9)
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.5)
```

train loop

```
for epoch in range(20):  # loop over the dataset multiple times
    net.train()
    for inputs, labels in tqdm(trainloader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
    scheduler.step()
```

plot statistic การ train

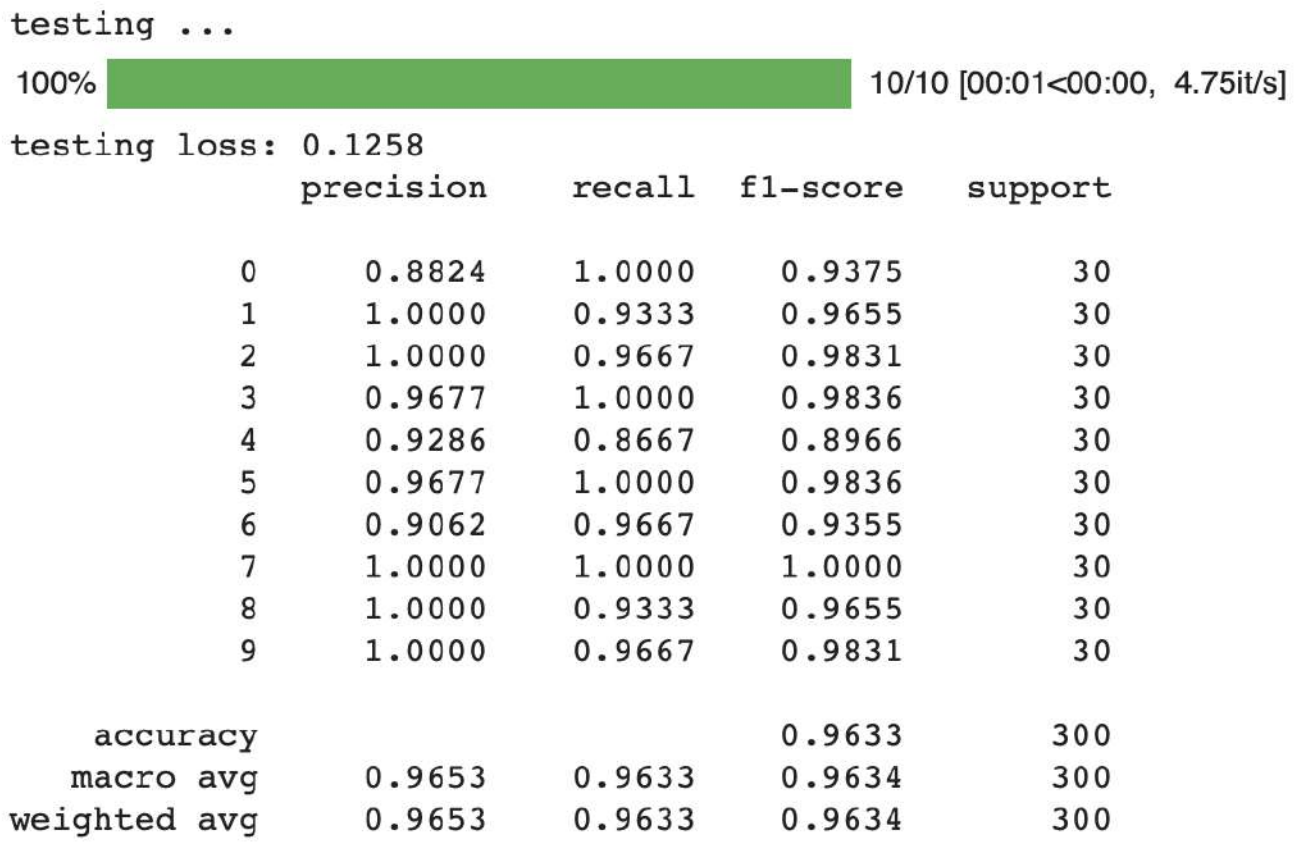
```
fig, axs = plt.subplots(3, figsize= (6,10))
# loss
axs[0].plot(history_train['loss'], label = 'training')
axs[0].plot(history_val['loss'], label = 'validation')
axs[0].set_title("loss")
axs[0].legend()
# acc
axs[1].plot(history_train['acc'], label = 'training')
axs[1].plot(history_val['acc'], label = 'validation')
axs[1].set_title("acc")
axs[1].legend()
# f1-score
axs[2].plot(history_train['f1-score'], label = 'training')
axs[2].plot(history_val['f1-score'], label = 'validation')
axs[2].set_title("f1-score")
axs[2].legend()
plt.show()
```



Evaluate model

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay


print('testing ...')
y_predict = list()
y_labels = list()
```

```
test_loss = 0.0
n = 0
with torch.no_grad():
        for data in tqdm(testloader):
                net.eval()
                inputs, labels = data
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = net(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

                y_labels += list(labels.cpu().numpy())
                y_predict += list(outputs.argmax(dim=1).cpu().numpy())
                # To get probabilities, you can run a softmax on outputs
                y_probs = torch.nn.functional.softmax(outputs, dim=1)
                y_probs = list(y_probs.cpu().numpy())
                n+=1

                # print statistics
                test_loss /= n
                print(f"testing loss: {test_loss:.4}" )

                report = classification_report(y_labels, y_predict, digits = 4)
                M = confusion_matrix(y_labels, y_predict)
                print(report)
                disp = ConfusionMatrixDisplay(confusion_matrix=M)
```



```
testing ...
100% ████████████████████████████████ 10/10 [00:01<00:00,  4.75it/s]
testing loss: 0.1258
              precision    recall  f1-score   support

           0     0.8824    1.0000    0.9375        30
           1     1.0000    0.9333    0.9655        30
           2     1.0000    0.9667    0.9831        30
           3     0.9677    1.0000    0.9836        30
           4     0.9286    0.8667    0.8966        30
           5     0.9677    1.0000    0.9836        30
           6     0.9062    0.9667    0.9355        30
           7     1.0000    1.0000    1.0000        30
           8     1.0000    0.9333    0.9655        30
           9     1.0000    0.9667    0.9831        30

    accuracy                         0.9633       300
   macro avg     0.9653    0.9633    0.9634       300
weighted avg     0.9653    0.9633    0.9634       300
```

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)
```

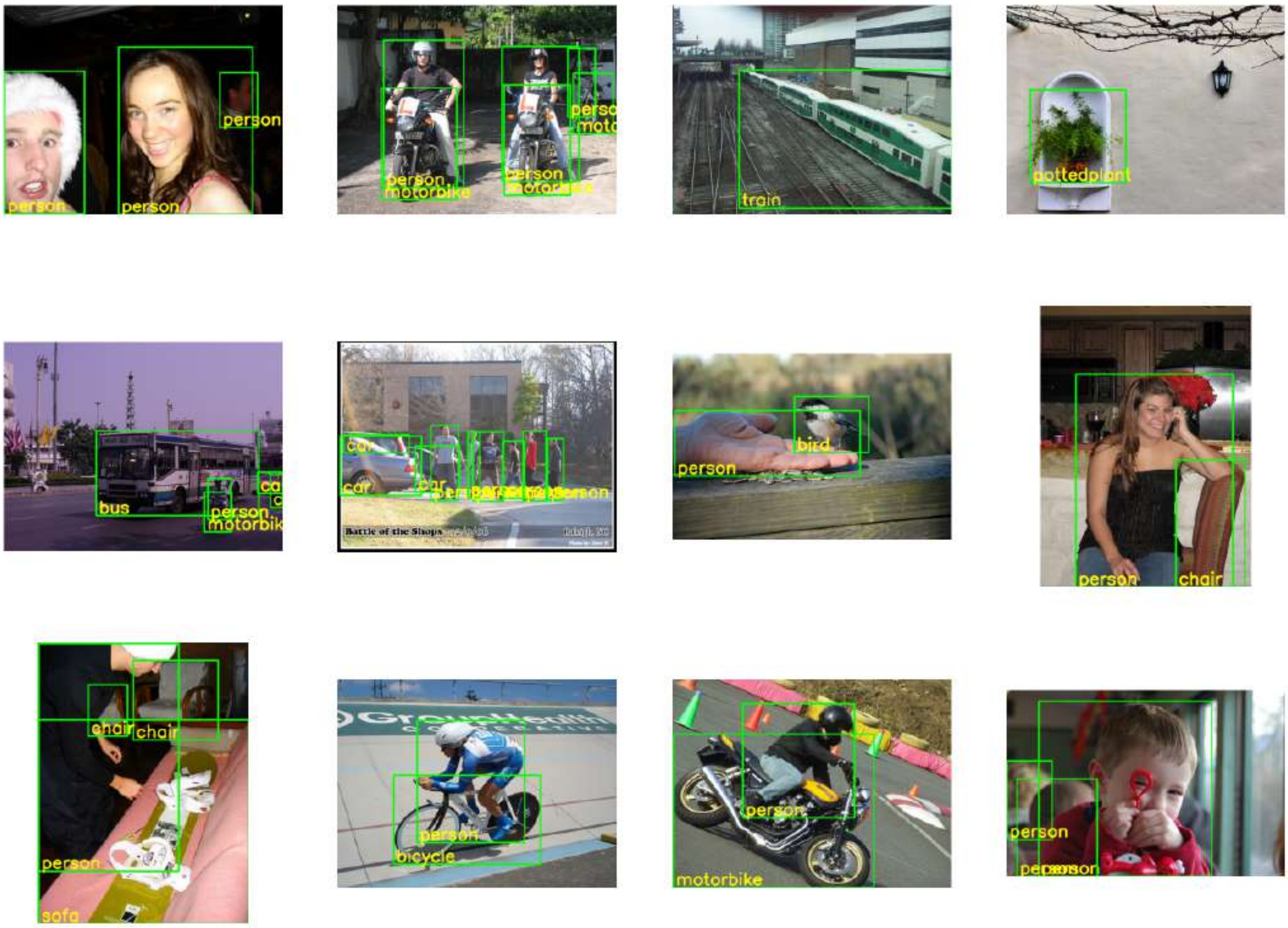
ดูจำนวน train, val, test

```
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
```

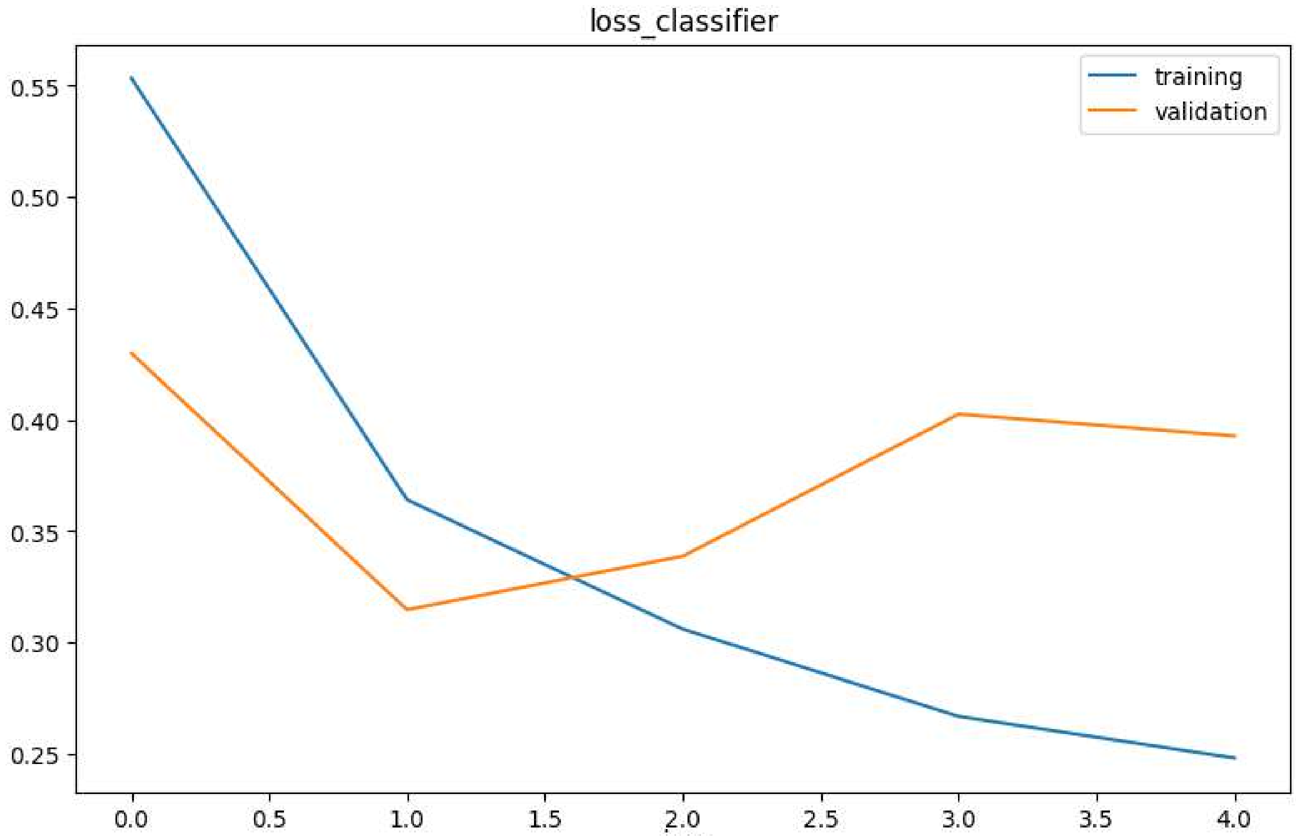
```
mIoU /= n

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)
```

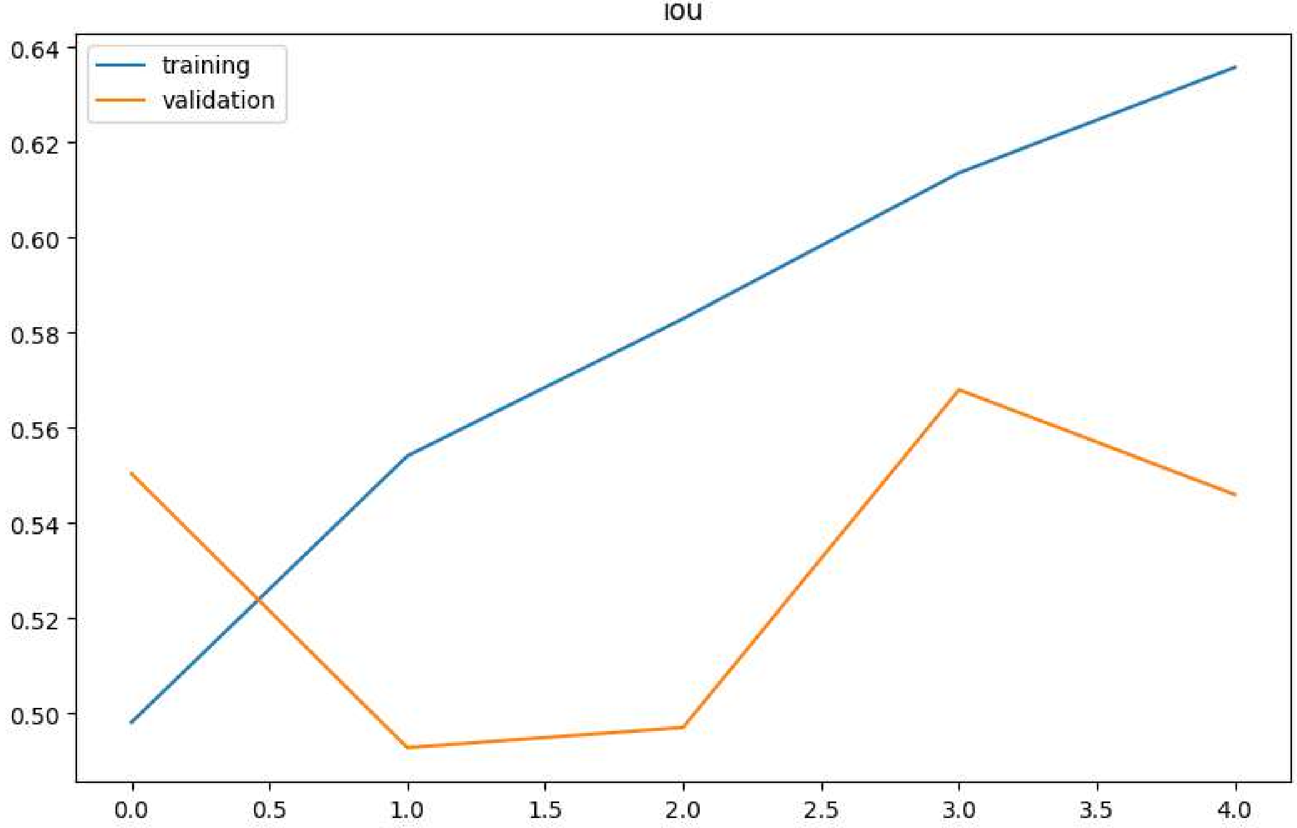
```
mIoU_sample, n_sample = miou(preds, labels)



# 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()
```

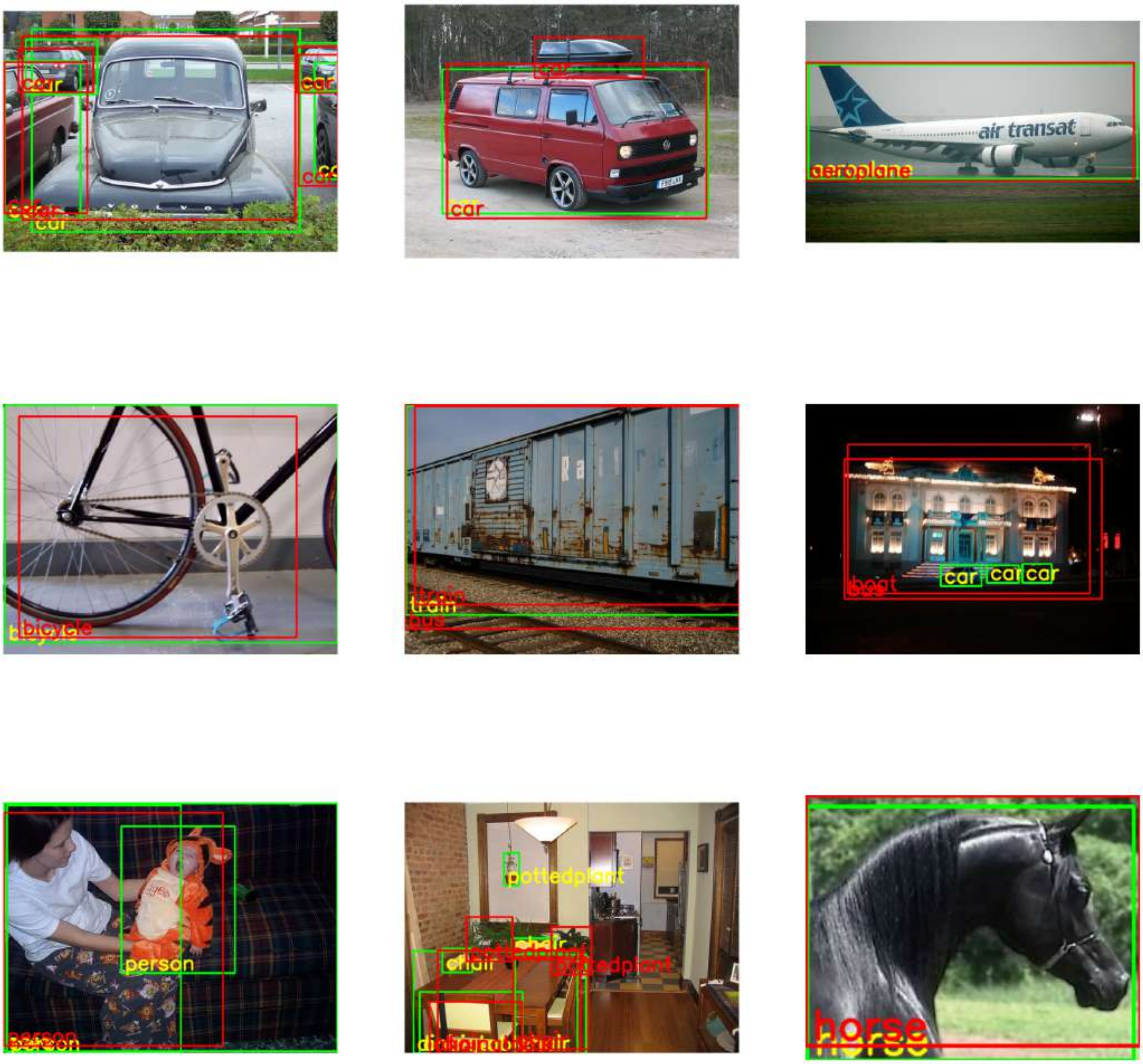




แสดง performance บน test set

```
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)
```



DSDE-Homework_3_2_Object_detection_VOCDetection_yolov8_basic

ทำ Object Detection ด้วย YOLOV8
Input data เป็น Pascal VOC Detection Dataset
และใช้ ultralytics library

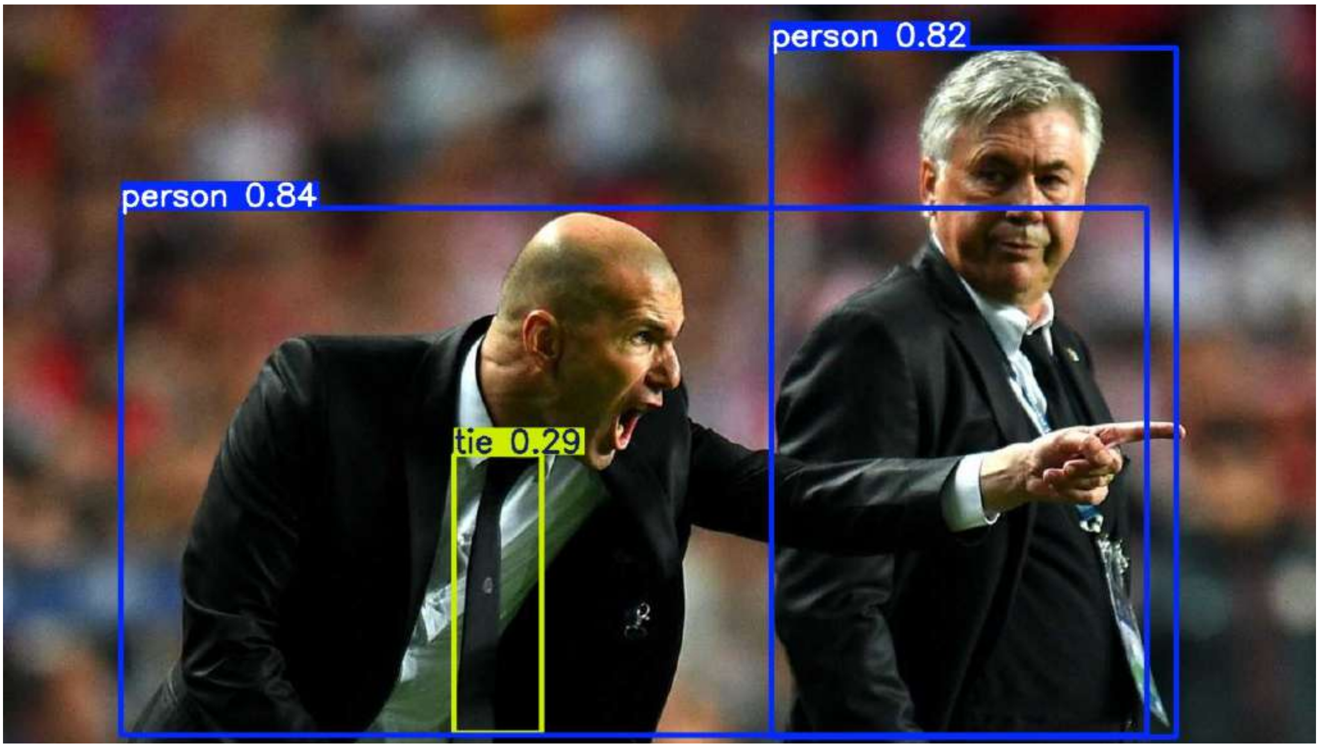
Setup Ultralytics

```
# Check for installed dependencies and available hardware
import ultralytics
ultralytics.checks()
```

ใช้ YOLOv8 ใน cli

```
# Run inference on an image with YOLOv8n
# set up device=0 refer to set cuda:0 (GPU)
!yolo predict model=yolov8n.pt
source='https://ultralytics.com/images/zidane.jpg' device=0
```

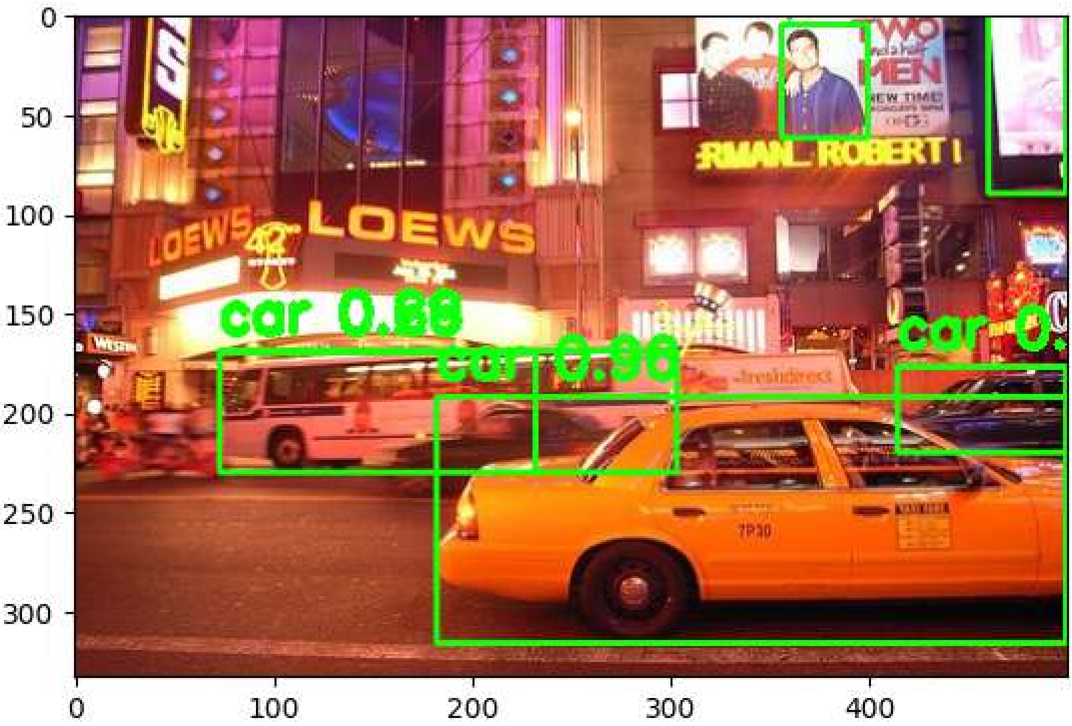
ผลลัพธ์ จะอยู่ใน runs/detect/predict



ใช้ YOLOv8 ใน Python

```
from ultralytics import YOLO



# Load a model

# model = YOLO('yolov8n.yaml') # build a new model from scratch

model = YOLO('yolov8n.pt') # load a pretrained model (recommended for
training)

model.to('cuda:0') # set up the model to GPU

# Use the model

results = model.train(data='VOC.yaml', epochs=3) # train the model

results = model.val() # evaluate model performance on the validation set
```

ผลลัพธ์



DSDE-Homework_3_3_Object_detection_VOCDetection_yolov8_advanced

Input data เป็น Pascal VOC 2007
เป็น custom dataset ที่นำมา train YOLOv8 model

Setup Ultralytics

```
# Check for installed dependencies and available hardware
import ultralytics
ultralytics.checks()
```

function download และ extract

```
def download_file(url, dest_folder):
        if not os.path.exists(dest_folder):
                os.makedirs(dest_folder)
        filename = os.path.join(dest_folder, url.split('/')[-1])
                if os.path.isfile(filename):
                print(f"{filename} already exists. Skipping download.")
                return filename
        response = requests.get(url, stream=True)
        total_size = int(response.headers.get('content-length', 0))
        with open(filename, 'wb') as f, tqdm(
                desc=filename,
                total=total_size,
                unit='iB',
                unit_scale=True,
                unit_divisor=1024,
        )as bar:
                for chunk in response.iter_content(chunk_size=1024):
                        if chunk:
                                f.write(chunk)
                                bar.update(len(chunk))
        return filename
def extract_tar(tar_file, extract_to_folder):
        with tarfile.open(tar_file, 'r') as tar_ref:
                tar_ref.extractall(extract_to_folder)
import locale

locale.getpreferredencoding = lambda: "UTF-8"
```

Download ไฟล์

```
# Download and extract VOC 2007
base_url = "http://host.robots.ox.ac.uk:8080/pascal/VOC/voc2007/"
files = {
'VOCtrainval_06-Nov-2007.tar': 'VOCtrainval_06-Nov-2007.tar',
'VOCtest_06-Nov-2007.tar': 'VOCtest_06-Nov-2007.tar'
}download_folder = '/content/voc2007'
for url, filename in files.items():
print(f"Downloading {filename}...")
tar_path = download_file(base_url + filename, download_folder)
print(f"Extracting {filename}...")
extract_tar(tar_path, download_folder)
print(f"{filename} extracted.")
```

สร้าง directory สำหรับเก็บ data YOLO format

```
# Define YOLO-compatible structure and directories
class_names = ["person", "car", "bus", "train", ... ]
voc_annot_folder = '/content/voc2007/VOCdevkit/VOC2007/Annotations'
image_folder = '/content/voc2007/VOCdevkit/VOC2007/JPEGImages'
yolo_annot_folder = '/content/yolo_annotations'
os.makedirs(yolo_annot_folder,exist_ok=True)
os.makedirs(os.path.join(yolo_annot_folder,"images"),exist_ok=True)
os.makedirs(os.path.join(yolo_annot_folder,"labels"),exist_ok=True)
```

function convert ให้เป็น yolo-text format

```
def process_annotations(file_list, annot_folder):
    for image_filename in file_list:
        xml_file = os.path.join(voc_annot_folder,
os.path.splitext(image_filename)[0] + '.xml')
        if not os.path.isfile(xml_file):
            continue
        # Parse XML, extract bounding box, convert to YOLO format
        tree = ET.parse(xml_file)
        root = tree.getroot()
        image = cv2.imread(os.path.join(image_folder, image_filename))
        h, w, _ = image.shape
        yolo_annot_file = os.path.join(annot_folder,
os.path.splitext(image_filename)[0] + '.txt')
        with open(yolo_annot_file, 'w') as f:
            for obj in root.findall('object'):
                cls = obj.find('name').text
                if cls not in class_names:
                    continue
                cls_id = class_names.index(cls)
                xml_box = obj.find('bndbox')
                b = [int(xml_box.find('xmin').text),
int(xml_box.find('ymin').text), int(xml_box.find('xmax').text),
int(xml_box.find('ymax').text)]
                # YOLO format conversion
                x_center = (b[0] + b[2]) / 2.0 / w
                y_center = (b[1] + b[3]) / 2.0 / h
                width = (b[2] - b[0]) / w
                height = (b[3] - b[1]) / h
                f.write(f"{cls_id} {x_center} {y_center} {width}
{height}\n")
```

train test split

```
from sklearn.model_selection import train_test_split
train_files, val_files = train_test_split(image_filenames, test_size=0.2,
random_state=42)
process_annotations(train_files, train_annot_folder)
process_annotations(val_files, val_annot_folder)
```

สร้าง data.yml

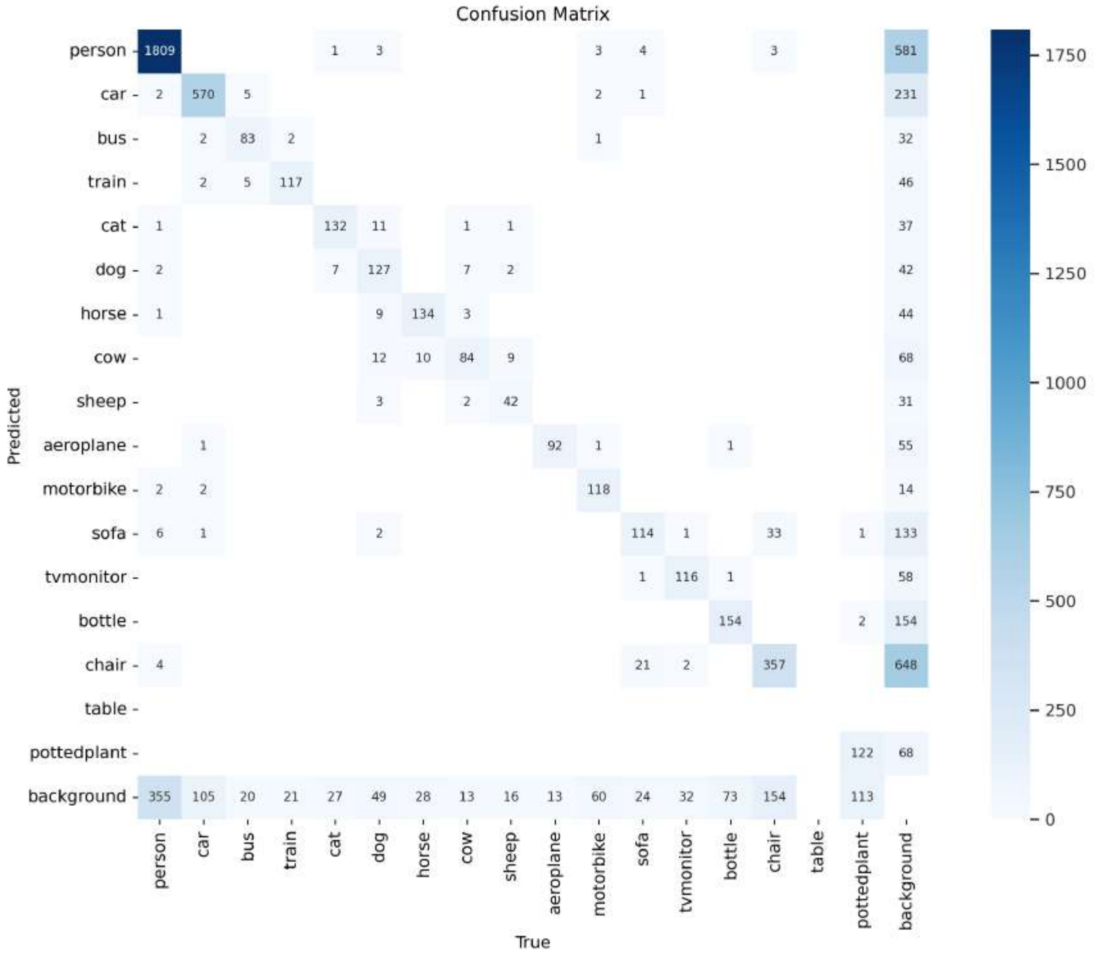
```
yaml_content = """
train: /content/yolo_annotations/images/train
val: /content/yolo_annotations/images/val
nc: 17  # Number of classes
names: ['person', 'car', 'bus', ... ]
"""
with open('/content/data.yaml', 'w') as f:
    f.write(yaml_content)
```

เริ่ม train model

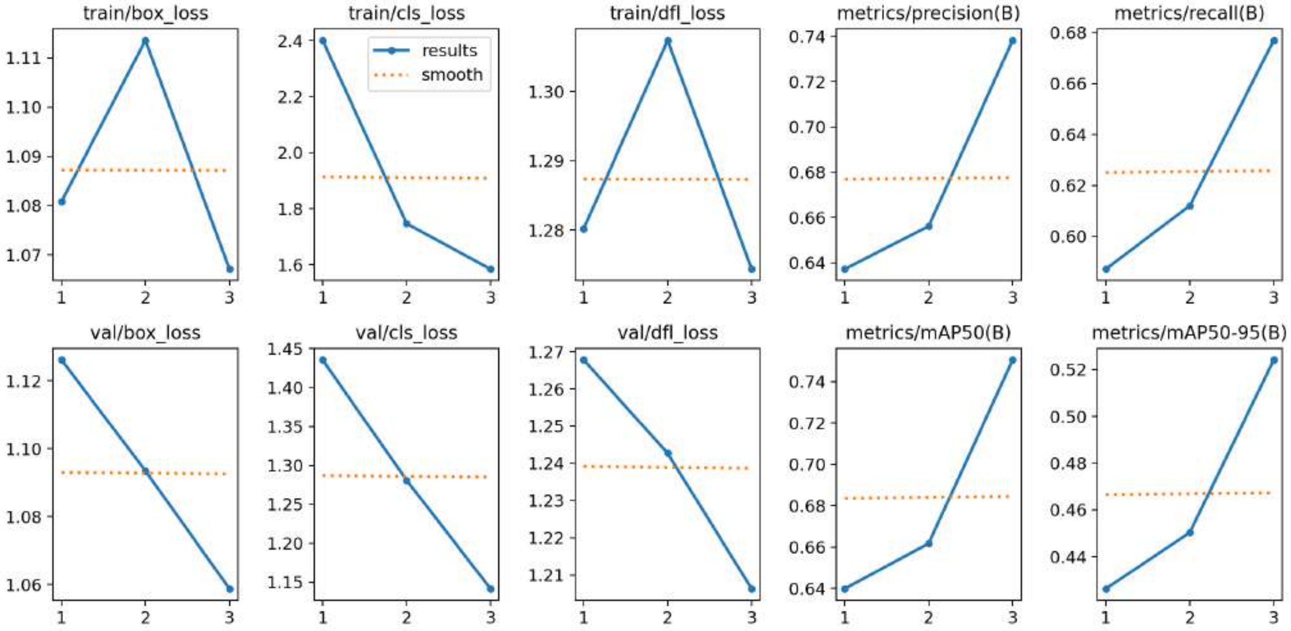
```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')  # Load pretrained model
model.to('cuda:0')  # Use GPU
results = model.train(data='data.yaml', epochs=3)  # Train for 3 epochs
```

Evaluation

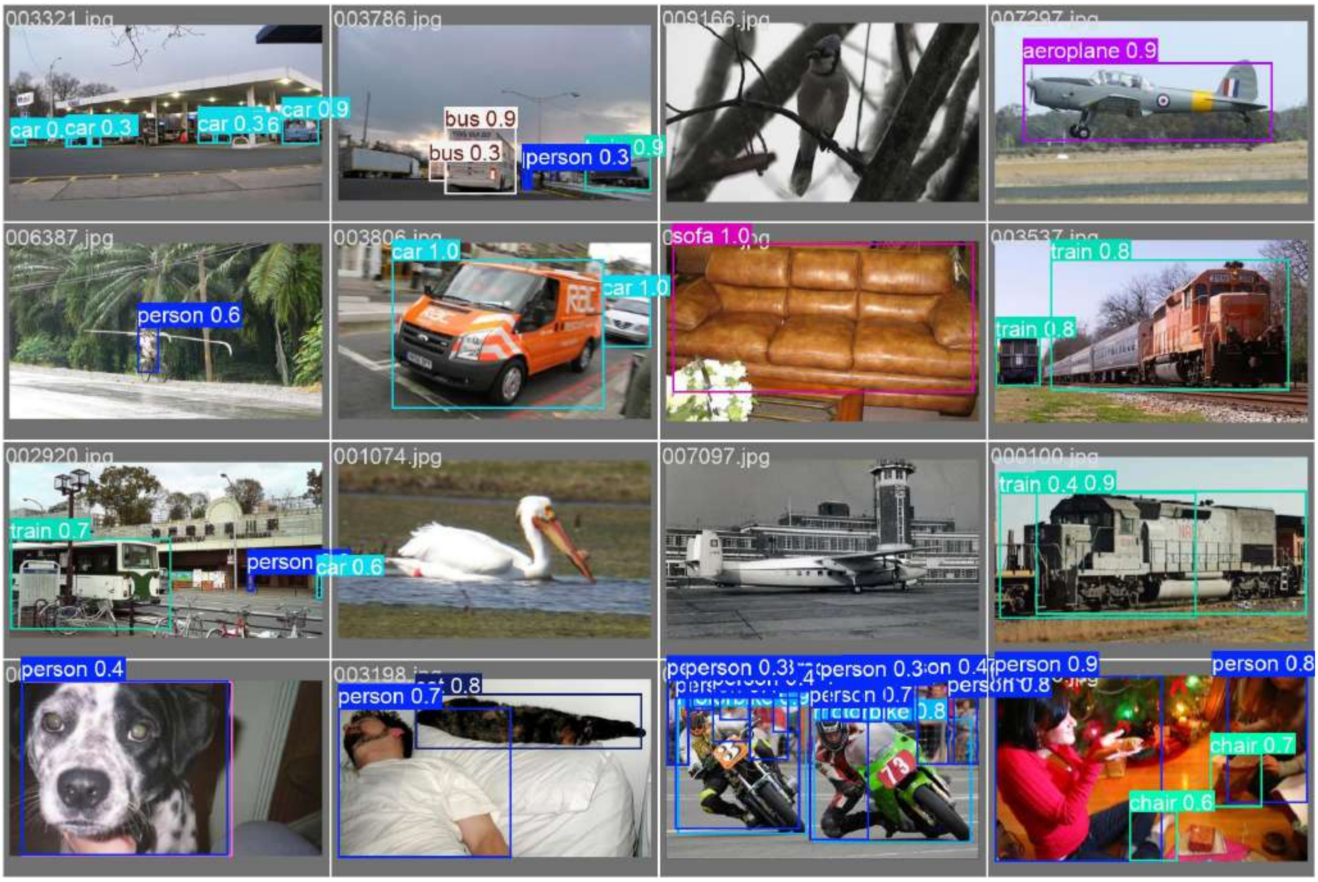
```
from PIL import Image
Image.open("/content/runs/detect/train/confusion_matrix.png")
Image.open("/content/runs/detect/train/results.png")
Image.open("/content/runs/detect/train/val_batch1_pred.jpg")
Image.open("/content/runs/detect/train/confusion_matrix.png")
```



```
Image.open("/content/runs/detect/train/results.png")
```



```
Image.open("/content/runs/detect/train/val_batch1_pred.jpg")
```


003321.jpg
car 0.
car 0.3
car 0.3
6
car 0.9
003786.jpg
bus 0.9
bus 0.3
person 0.3
0.9
009166.jpg
007297.jpg
aeroplane 0.9
006387.jpg
person 0.6
003806.jpg
car 1.0
car 1.0
sofa 1.0
003537.jpg
train 0.8
train 0.8
002920.jpg
train 0.7
person
car 0.6
001074.jpg
007097.jpg
000100.jpg
train 0.4 0.9
person 0.4
003198.jpg
person 0.7
0.8
person 0.3
person 0.3
person 0.4
person 0.4
person 0.7
motorbike
motorbike 0.8
person 0.9
person 0.8
person 0.8
chair 0.7
chair 0.6

DSDE-Homework_4_Semantic_segmentation_Camseq_deeplabv3_DataInGD

ทำ semantic segmentation เพื่อหา shape ของ object ด้วย DeepLabV3 โดยใช้ PyTorch
Input data ที่ใช้ train เป็น CamSeq dataset
ทำ classification 32 classes

Setup

```
! pip install torchinfo
! nvidia-smi
```

unzip dataset Camseq

```
# unzip file (-q : quiet mode)
! unzip -FF Camseq_2007.zip
```

import libraries

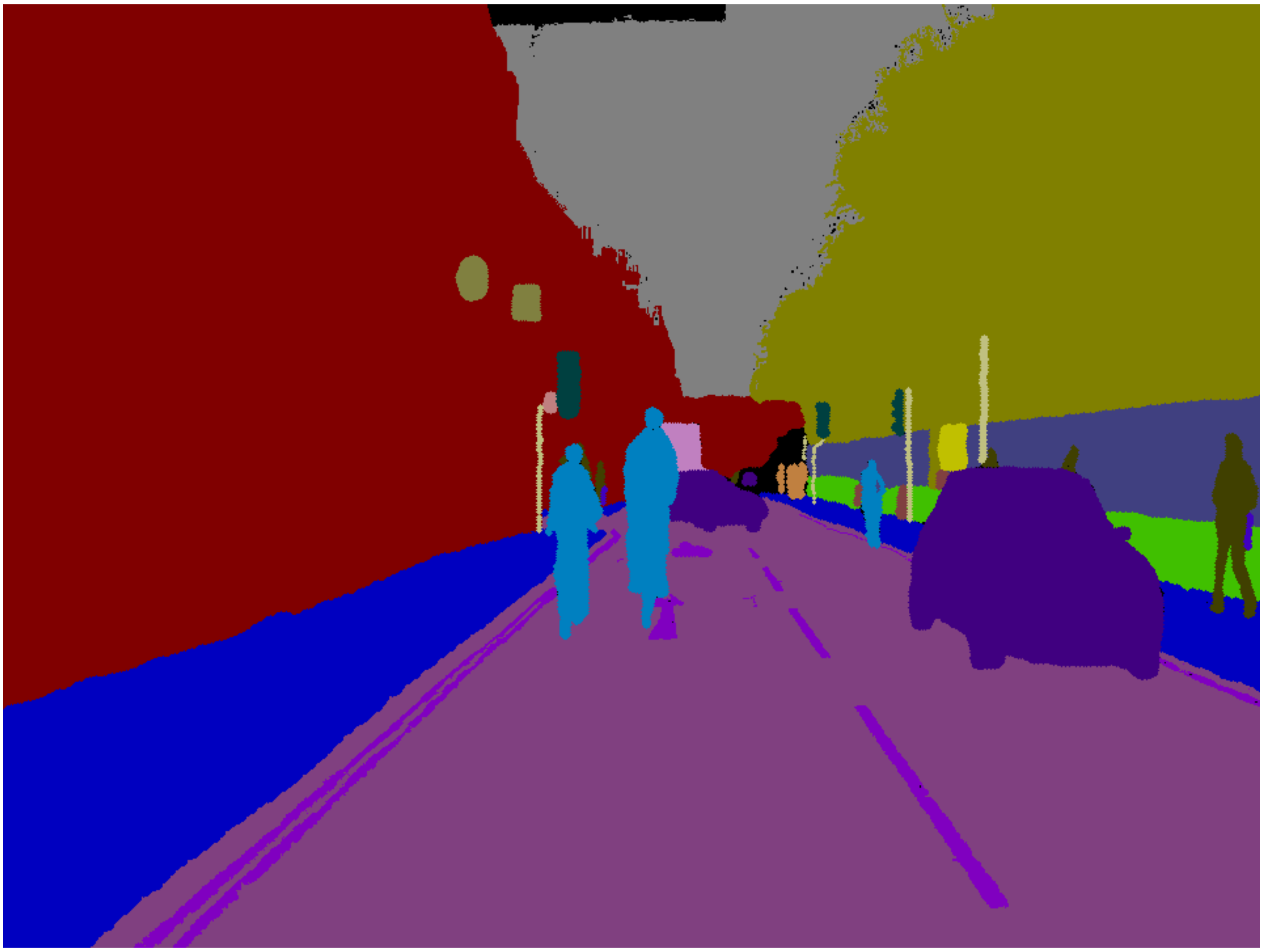
```
# Pytorch framework
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
from torchinfo import summary as summary_info
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision
from torch.utils.data import Dataset, DataLoader, random_split, Subset
#utils
import os
import numpy as np
from collections import OrderedDict
from itertools import islice
#Transformation
import albumentations as A
from albumentations.pytorch import ToTensorV2
from collections import OrderedDict
#Visualization
import matplotlib.pyplot as plt
import cv2
from PIL import Image
#Progress bar
from tqdm.notebook import tqdm
%matplotlib inline
torch.__version__
```

ตัวอย่าง input image

```
img = Image.open('./Camseq_2007/0016E5_07961.png')
img
```

ตัวที่มี segmentation จะมีชื่อไฟล์ลงท้ายด้วย _L

```
img = Image.open('./Camseq_2007/0016E5_07961_L.png')
```



load สีที่ใช้ label

```
# load colormap from label_colors.txt
colormap = OrderedDict()
with open("./Camseq_2007/label_colors.txt",'r') as f:
        for line in f:
                r,g,b,cls = line.split()
                colormap[cls] = [int(e) for e in [r,g,b]]
                list(islice(colormap.items(),8))
```

```
'Animal': [64, 128, 64],
'Archway': [192, 0, 128],
'Bicyclist': [0, 128, 192],
'Bridge': [0, 128, 64],
'Building': [128, 0, 0],
'Car': [64, 0, 128],
'CartLuggagePram': [64, 0, 192],
'Child': [192, 128, 64],
```

สร้าง DataSet Class

```
class CamSeqDataset(Dataset):

    def __init__(self,
                 img_dir,
                 colormap=colormap,
                 transforms=None):

        super().__init__()
        # sort order of frame from video sequence
        self.images = sorted([os.path.join(img_dir, e)
                              for e in os.listdir(img_dir)
                              if not e.split('.')[0].endswith('_L')])
        # remove text files
        self.images = [e for e in self.images if not e.endswith('.txt')]
        self.masks = sorted([os.path.join(img_dir, e)
                              for e in os.listdir(img_dir)
                              if e.split('.')[0].endswith('_L')])
        self.colormap = colormap
        self.num_classes = len(self.colormap) # 32 classes
        self.transforms = transforms

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):

        img = Image.open(self.images[idx])
        mask = Image.open(self.masks[idx])

        if img.mode != 'RGB':
            img = img.convert('RGB')
        if mask.mode != 'RGB':
            mask = mask.convert('RGB')

        img = np.asarray(img) # change from image to array
        mask = np.asarray(mask)
        mask_channels = np.zeros(
            (mask.shape[0], mask.shape[1]), dtype=np.int64)

        # convert RGB mask to class-pixel mask ; (R,G,B) -> (Class)
        for i, cls in enumerate(self.colormap.keys()):
            color = self.colormap[cls]
            sub_mask = np.all(mask==color, axis=-1)*i
            mask_channels += sub_mask #*i

        # transforms such as normalization
        if self.transforms is not None:
            transformed = self.transforms(image=img, masks=mask_channels)
            img = transformed['image']
            mask_channels = transformed['masks']

        mask_channels = mask_channels.astype(np.float32)
        img = img.astype(np.float32) #/255

        instance = {'image': torch.from_numpy(img.transpose(2,0,1)),
                    'mask': torch.from_numpy(mask_channels)}

        return instance

    def ___first__(self):
        return self.__getitem__[0]
```

สร้าง transform

```
# simple transform (using ImageNet norm and std) "Albumentation == torchvision.transforms for segmentation"

transform = A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
```

แบ่ง train val test

```
dataset = CamSeqDataset(img_dir='./Camseq_2007', colormap=colormap,transforms=transform)
# we split train/val/test -> 70/15/15
train_size = int(len(dataset)*0.7)
val_size = (len(dataset)-train_size)//2
# train_set, rest = random_split(dataset, [train_size, len(dataset)-train_size])
# val_set, test_set = random_split(rest, [val_size, len(rest)-val_size])
# We do not use random split because the dataset is extracted from a video sequence, so nearly every frame looks the same.
train_set = Subset(dataset, range(train_size))
val_set = Subset(dataset, range(train_size, train_size + val_size))
test_set = Subset(dataset, range(train_size + val_size, len(dataset)))

batch_size = 2

train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=True)
```

```
# loader size : 101 images (train/val/test) -> (70/16/15) -> batch size 2 (35/8/8)
```

```
len(train_loader), len(val_loader), len(test_loader)
```

```
(35, 8, 8)
```

function สำหรับ create model, และ train model

```
def create_model(out_channels=32):
    model =
torchvision.models.segmentation.deeplabv3_resnet50(pretrained=True)
    # decoder head
    model.classifier =
torchvision.models.segmentation.deeplabv3.DeepLabHead(
        2048, num_classes=out_channels)

    model.train()
    return model
```

```
def train_model(model,
                train_loader,
                val_loader,
                criterion= nn.CrossEntropyLoss(),
                num_epochs=1,
                device=torch.device('cpu'),
                lr=0.0002,
                model_path="./model.pth"
                ):

    model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    scheduler = lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.5)
    metrics = {
          "train_losses" : [],
          "val_losses" : [],
          "train_acc" : [],
          "val_acc" : []
          }
    min_val_loss = 1e10
    for epoch in range(1, num_epochs + 1):
        tr_loss = []
        val_loss = []
        tr_acc = []
        val_acc = []
        model.train()
        print('Epoch {}/{}'.format(epoch, num_epochs))
        for sample in tqdm(train_loader):
            if sample['image'].shape[0]==1:
                break
            inputs = sample['image'].to(device)
            masks = sample['mask'].to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            y_pred = outputs['out']
            y_true = masks
            loss = criterion(y_pred.float(), y_true.long())
            acc = (torch.argmax(y_pred, 1) == y_true).float().mean()
            loss.backward()
            tr_loss.append(loss)
            tr_acc.append(acc)
            optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
        avg_tr_loss = torch.mean(torch.Tensor(tr_loss))
        metrics["train_losses"].append(avg_tr_loss)
        avg_tr_acc = torch.mean(torch.Tensor(tr_acc))
        metrics["train_acc"].append(avg_tr_acc)
        print(f'Train loss: {avg_tr_loss}')
        print(f'Train acc: {avg_tr_acc}')

        # Validation phrase
        for sample in tqdm(val_loader):
            if sample['image'].shape[0]==1:
                break
            inputs = sample['image'].to(device)
            masks = sample['mask'].to(device)
            model.eval()
            with torch.no_grad():
                outputs = model(inputs)
            y_pred = outputs['out']
            y_true = masks
            loss = criterion(y_pred.float(), y_true.long())
            # acc using pixel accuracy. (it is easy to understand,but no
way the best metric)
            # learn more https://towardsdatascience.com/metrics-to-
evaluate-your-semantic-segmentation-model-6bcb99639aa2
            acc = (torch.argmax(y_pred, 1) == y_true).float().mean()
            val_loss.append(loss)
            val_acc.append(acc)
        avg_val_loss = torch.mean(torch.Tensor(val_loss))
        metrics["val_losses"].append(avg_val_loss)
        avg_val_acc = torch.mean(torch.Tensor(val_acc))
        metrics["val_acc"].append(avg_val_acc)
        print(f'val loss: {avg_val_loss}')
        print(f'val acc: {avg_val_acc}')
        # save model state that have best val loss
        if avg_val_loss < min_val_loss:
            torch.save(model.state_dict(), model_path)
            min_val_loss = avg_val_loss
    return model,metrics
```

สร้าง model

```
# set device
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Assuming that we are on a CUDA machine, this should print a CUDA device:
lr =0.0002
print(device)
from torchinfo import summary as summary_info
model = create_model(out_channels=dataset.num_classes)
# model architecture
model
```

ดู summary ของ network

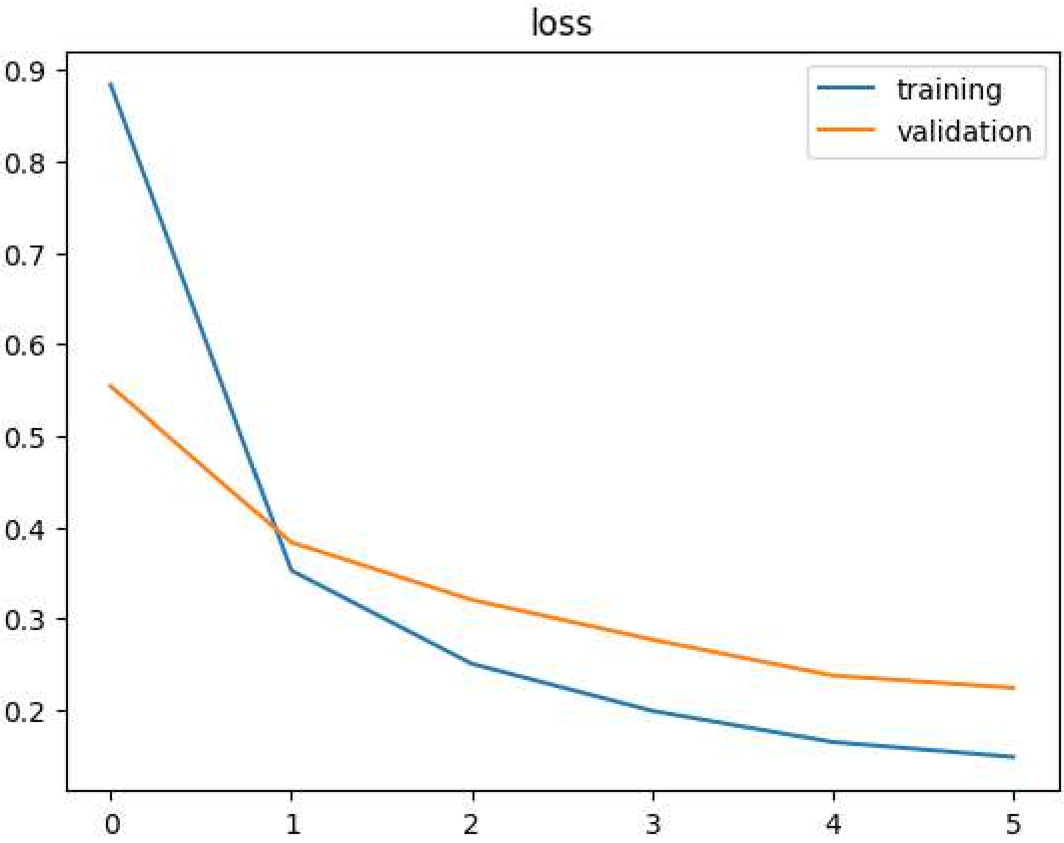
```
model.to(device)
# model summary that can pass dummy input to show output shape
# input shape desc : (batch, channel, H, W)
summary_info(model, input_size = (2, 3, 720, 960))
```

```
==========================================================================
==========================
Total params: 42,006,901
Trainable params: 42,006,901
Non-trainable params: 0
Total mult-adds (G): 913.76
==========================================================================
==========================
Input size (MB): 16.59
Forward/backward pass size (MB): 11731.92
Params size (MB): 168.03
Estimated Total Size (MB): 11916.54
==========================================================================
==========================
```

loss graph

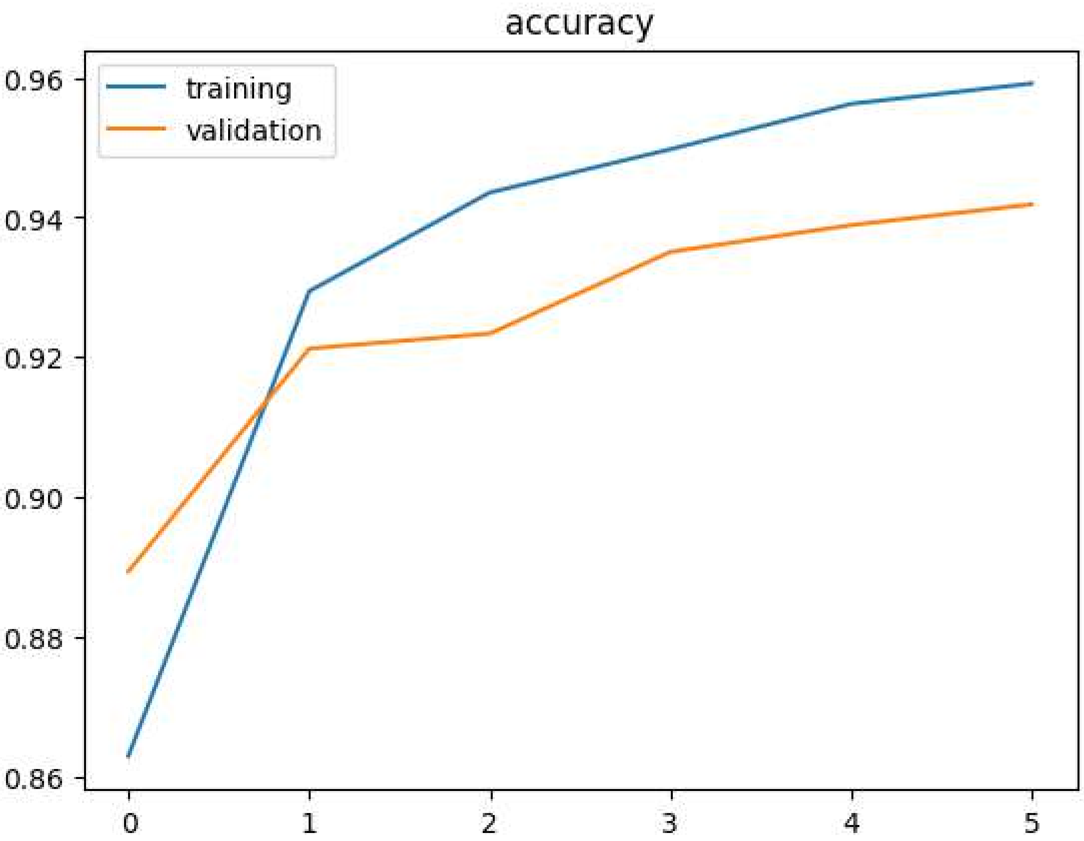
```
# loss graph
#overfitting because small dataset (only 70 image)
```

```
plt.plot(metrics["train_losses"], label = 'training')
plt.plot(metrics["val_losses"], label = 'validation')
plt.title("loss")
plt.legend()
```



plot pixel accuracy

```
# Pixel accuracy
plt.plot(metrics["train_acc"], label = 'training')
plt.plot(metrics["val_acc"], label = 'validation')
plt.title("accuracy")
plt.legend()
```



โหลด model ที่ save มาทำ evaluation

```
# load best val state

PATH = './model.pth'

model.load_state_dict(torch.load(PATH))
```

clear cache และ garbage collect

```
import gc
with torch.no_grad():
torch.cuda.empty_cache()
gc.collect()
```

```
model.eval()
accuracy = []
with torch.no_grad():
    for sample in tqdm(test_loader):
        if sample['image'].shape[0]==1:
            break
        inputs = sample['image'].to(device)
        masks = sample['mask'].to(device)

        outputs = model(inputs)
        y_pred = outputs['out']
        y_true = masks
        # acc using pixel accuracy. (it is easy to understand,but not the best metric)
        acc = (torch.argmax(y_pred, 1) == y_true).float().mean()
        accuracy.append(acc)
accuracy = torch.mean(torch.Tensor(accuracy))
print(f'accuracy: {accuracy}')
```

```
>> accuracy: 0.9269694089889526
```

ควรใช้ IoU ในการวัด performance ของ segmentation task
function สำหรับ compute IoU

```
def compute_iou_batch(predictions, ground_truths, num_classes):
    """
    Compute Intersection over Union (IoU) for semantic segmentation over a batch of images.

    Parameters:
    - predictions: A 4D torch tensor of shape (batch, class, height, width) with predicted class scores
    - ground_truths: A 3D torch tensor of shape (batch, height, width) with ground truth class labels
    - num_classes: Total number of classes in the segmentation task

    Returns:
    - A torch tensor of IoU values for each class
    """
    batch_size = predictions.size(0)
    iou_per_class = torch.zeros(num_classes, dtype=torch.float32)

    # Iterate over each class
    for cls in range(num_classes):
        intersection = 0
        union = 0

        # Compute IoU per image in the batch
        for i in range(batch_size):
            # Predicted mask for the current class
            pred_mask = (predictions[i, cls] > 0.5)  # Apply threshold for binary mask
            # Ground truth mask for the current class
            gt_mask = (ground_truths[i] == cls)

            # Compute Intersection and Union
            intersection += torch.sum(pred_mask & gt_mask).item()
            union += torch.sum(pred_mask | gt_mask).item()
        # Compute IoU for the current class, avoid division by zero
        if union == 0:
            iou_per_class[cls] = float('nan')  # or 0 if you prefer
        else:
            iou_per_class[cls] = intersection / union

    return iou_per_class
```

Eval

```
iou_per_class_total = torch.zeros(num_classes, dtype=torch.float32)
num_batches = 0
num_classes = 32
model.eval()
with torch.no_grad():
    for sample in tqdm(test_loader):
        if sample['image'].shape[0]==1:
            break
        inputs = sample['image'].to(device)
        masks = sample['mask'].to(device)

        outputs = model(inputs)
        y_pred = outputs['out']
        y_true = masks
        # predictions, ground_truths = batch
        # predictions = torch.softmax(predictions, dim=1)  # Convert
logits to probabilities
        batch_iou = compute_iou_batch(y_pred, y_true, num_classes)
        iou_per_class_total += batch_iou
        num_batches += 1

average_iou_per_class = iou_per_class_total / num_batches
overall_average_iou = torch.nanmean(average_iou_per_class)
print(f'average_iou_per_class: {average_iou_per_class}')
print(f'overall_average_iou: {overall_average_iou}')
```

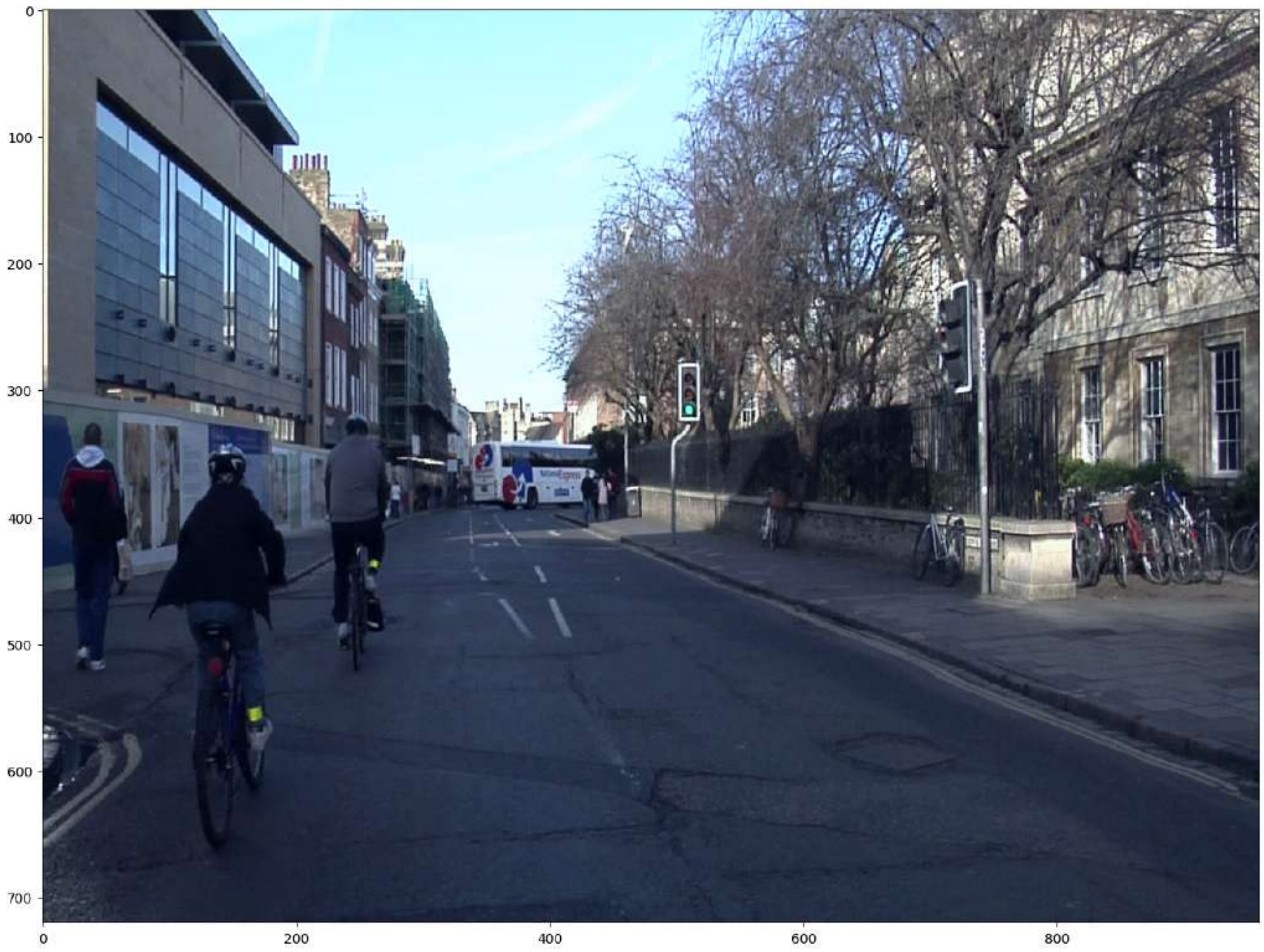
```
>> average_iou_per_class: tensor([ nan, 0.0000, 0.6856, nan, 0.7293,
0.0067, nan, 0.2761, 0.0370, 0.7496, 0.1374, nan, 0.0000, nan, 0.2079,
nan, 0.0309, 0.8530, nan, 0.4549, 0.0000, 0.7450, nan, nan, 0.5498, nan,
0.6805, 0.5452, nan, 0.0181, 0.0618, 0.6146]) overall_average_iou:
0.3515826165676117
```

```
model.eval()
img = next(iter(val_loader))
output = model(img['image'].to(device))['out']
```

function invTransofrom ใช้สำหรับแปลงรูปที่ normalized แล้ว กลับมาเป็นรูป original

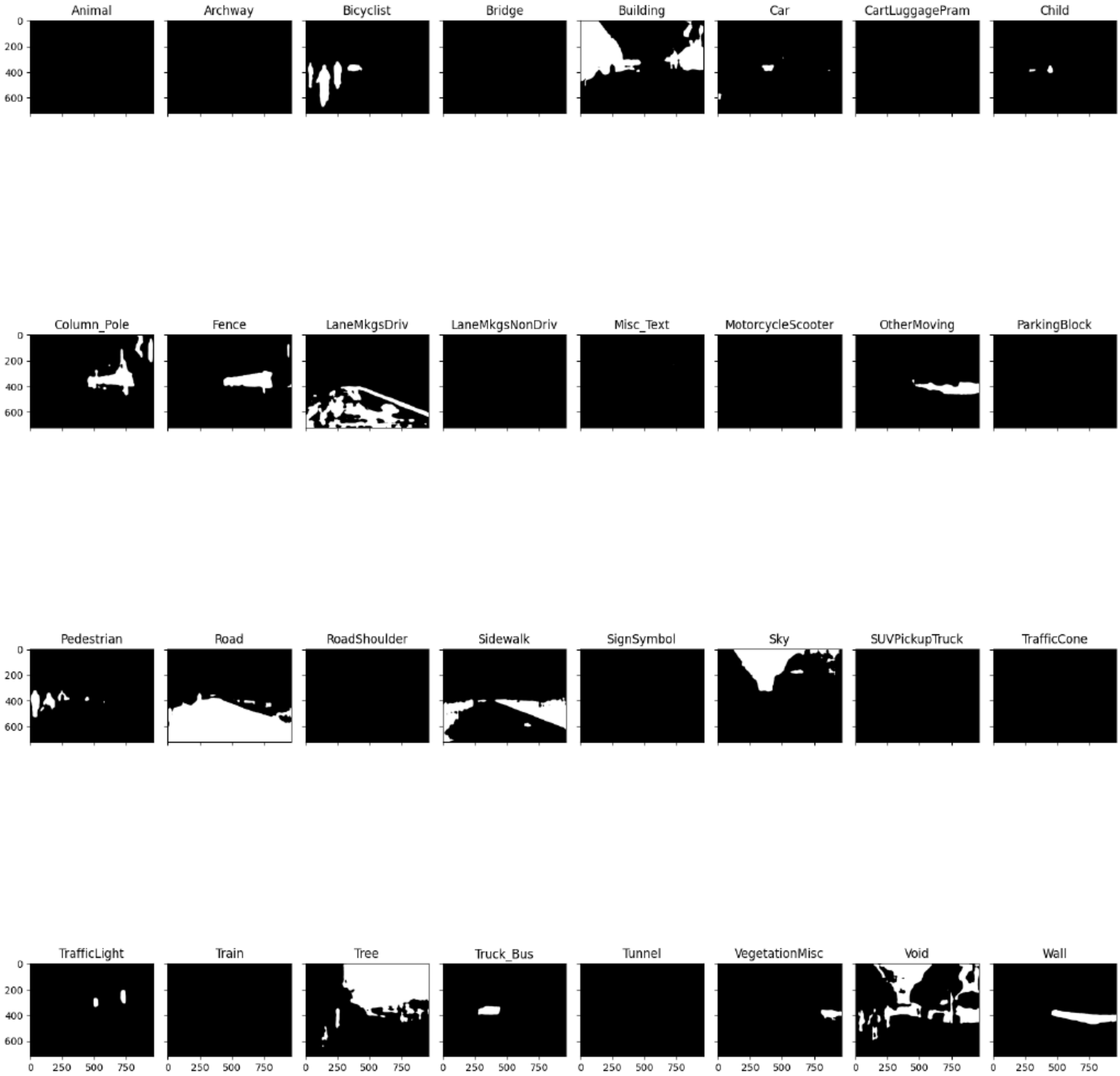
```
# inverse transform (normalized image -> original)
def invTransform(img):
    img = img*torch.tensor([0.229, 0.224, 0.225]).mean() +
torch.tensor([0.485, 0.456, 0.406]).mean()     # unnormalize
    npimg = img.numpy().clip(0,255)
    return npimg
def imshow(img):
    npimg = invTransform(img)
    plt.figure(figsize=(16,16))
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

```
image = img['image'][0]

imshow(image)
```



```
fig, axes = plt.subplots(nrows=4, ncols=8, sharex=True, sharey=True,
figsize=(16,20))
```

```
axes_list = [item for sublist in axes for item in sublist]

thresh=0.3
res = output[0].detach().cpu().numpy()
for i, mask in enumerate(res):
    ax = axes_list.pop(0)
    ax.imshow(np.where(mask>thresh, 255, 0), cmap='gray')
    ax.set_title(list(colormap.keys())[i])

for ax in axes_list:
    ax.remove()

plt.tight_layout()
```



นำ output ผ่าน argmax

```
seg = torch.argmax(output[0], 0)
seg = seg.cpu().detach().numpy()
```

```
plt.figure(figsize=(16,16))
plt.imshow(seg)  # display image
plt.show()
```

```
oiginal_image = invTransform(img['image'][0])
oiginal_image = np.transpose(oiginal_image, (1, 2, 0))
f, axarr = plt.subplots(1,2,figsize= (16,16))
axarr[0].imshow(seg)
axarr[1].imshow(oiginal_image)
plt.show()
```