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()` เพื่อเข้าสู่ 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}%)")
```