Introduction to Deep Neural Networks (Spring 2021)

Homework #4 (50 Pts, Due Date: May 23)

Student ID 2019311195

Name 김지유

Instruction: We provide all codes and datasets in Python. Please write your code to complete models('models/AlexNet.py', 'models/ResNet.py'). Submit two files as follows:

- 'DNN_HW4_YourName_ STUDENTID.zip': ./models/*.py and your document
- 'DNN_HW4_YourName_ STUDENTID.pdf': Your document converted into pdf.

TIP 1. : Please look at PyTorch implementation of the LeNet (models/LeNet_5.py). Please refer the lecture slide 'W09 Convolutional Neural Networks (CNNs).pdf' 4 p.

TIP 2. : You can use Google Colab for using GPU.

TIP 3. : You can check how to use PyTorch.

Kor 1. https://tutorials.pytorch.kr/beginner/blitz/tensor_tutorial.html#sphx-glr-beginner-blitz-tensor-tutorial-py

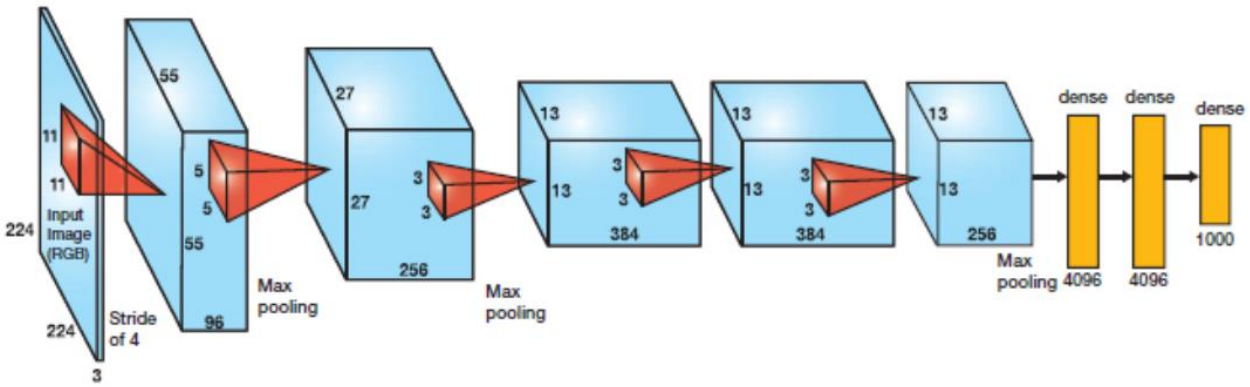
Kor 2. https://wikidocs.net/book/2788

Eng 1. https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

Eng 2. https://github.com/yunjey/pytorch-tutorial

(1) [30 pts] Implement CNN models in ‘AlexNet.py’ and ‘ResNet.py.’

(a) [AlexNet] Implement AlexNet in ‘models/AlexNet.py’. Please refer the lecture slide ‘W09 Convolutional Neural Networks (CNNs).pdf’ 8~25 p.



Answer: Fill your code here. You also have to submit your code to i-campus.

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import time
import os
import numpy as np
import matplotlib.pyplot as plt
from torchvision.transforms.functional import resize
from tqdm import tqdm

# W09 Convolutional Neural Networks (CNNs).pdf - 10 page
class AlexNet(nn.Module):
    def __init__(self, input_channel, output_dim, learning_rate, reg_lambda, devic
e):
        super(AlexNet, self).__init__()

        self.output_dim = output_dim
        self.device = device
        self.loss_function = None
        self.optimizer = None
        # ================================ EDIT HERE ================================
=
        # convolution layers
        self.CONV1 = nn.Conv2d(in_channels=input_channel, out_channels=96, kernel_
size=(11, 11), stride=4)
        self.CONV2 = nn.Conv2d(in_channels=96, out_channels=256, kernel_size=(5, 5
), stride=1, padding=2)
```

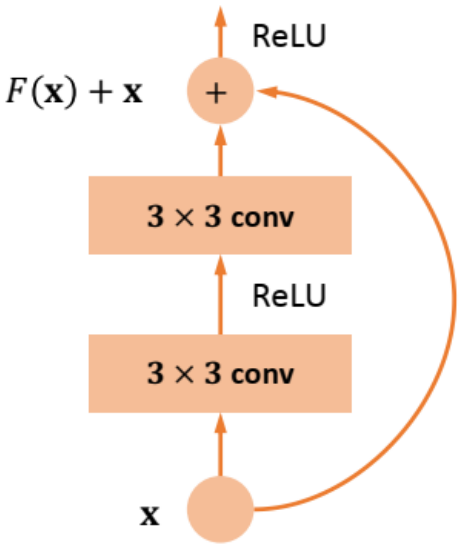
```
        self.CONV3 = nn.Conv2d(in_channels=256, out_channels=384, kernel_size=(3,
3), stride=1, padding=1)
        self.CONV4 = nn.Conv2d(in_channels=384, out_channels=384, kernel_size=(3,
3), stride=1, padding=1)
        self.CONV5 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=(3,
3), stride=1, padding=1)

        # pooling layers
        self.POOL1 = nn.MaxPool2d(kernel_size=(3, 3), stride=2)
        self.POOL2 = nn.MaxPool2d(kernel_size=(3, 3), stride=2)
        self.POOL3 = nn.MaxPool2d(kernel_size=(3, 3), stride=2)

        # Fully-connected layers
        self.FC1 = nn.Linear(9216, 4096)
        self.FC2 = nn.Linear(4096, 4096)
        self.FC3 = nn.Linear(4096, output_dim)

        # For simplicity, we can use multiple modules as a single module
        self.Conv_layers = nn.Sequential(self.CONV1, nn.ReLU(), self.POOL1, self.C
ONV2, nn.ReLU(), self.POOL2, self.CONV3, nn.ReLU(), self.CONV4, nn.ReLU(), self.CO
NV5, nn.ReLU(), self.POOL3)
        self.FC_layers = nn.Sequential(self.FC1, nn.ReLU(), self.FC2, nn.ReLU(), s
elf.FC3)

        self.loss_function = nn.CrossEntropyLoss()
        self.optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate, wei
ght_decay=reg_lambda)
        # ================================ EDIT HERE ================================
=



    def forward(self, x):
        out = torch.zeros((x.shape[0], self.output_dim))
        # ================================ EDIT HERE ================================
=
        h = self.Conv_layers(x)
        stretched_h = h.reshape(x.shape[0], -1)
        out = self.FC_layers(stretched_h)
        # ================================ EDIT HERE ================================
=
        return out

    def predict(self, x):
        pred_y = np.zeros((x.shape[0], ))
        pred_y = []
        x_tenser = torch.tensor(x, dtype=torch.float, device = self.device)
        data_loader = DataLoader(x_tenser, batch_size=self.batch_size)
```

```
        with torch.no_grad():
            for batch_data in data_loader:
                batch_x = batch_data
                batch_x = resize(batch_x, (227, 227))
                batch_pred = self.forward(batch_x).argmax(axis=1)
                pred_y.append(batch_pred.cpu().numpy())
        pred_y = np.concatenate(pred_y, axis=0)
        return pred_y

    def train(self, train_x, train_y, valid_x, valid_y, num_epochs, batch_size, te
st_every=10, print_every=10):
        self.train_accuracy = []
        self.valid_accuracy = []
        best_epoch = -1
        best_acc = -1
        self.num_epochs = num_epochs
        self.test_every = test_every

        # transfrom numpy data to torch data and make torch dataset
        x_tenser = torch.tensor(train_x, dtype=torch.float, device = self.device)
        y_tenser = torch.tensor(train_y, dtype=torch.long, device = self.device)
        dataset = TensorDataset(x_tenser, y_tenser)

        data_loader = DataLoader(dataset, batch_size=batch_size)
        self.batch_size = batch_size

        for epoch in range(1, num_epochs+1):
            start = time.time()
            epoch_loss = 0.0
            # model Train
            for b, batch_data in enumerate(data_loader):
                batch_x, batch_y = batch_data
                batch_x = resize(batch_x, (227, 227))
                pred_y = self.forward(batch_x)

                if self.loss_function is not None:
                    loss = self.loss_function(pred_y, batch_y)
                    self.optimizer.zero_grad()
                    loss.backward()
                    self.optimizer.step()
                    epoch_loss += loss

            epoch_loss /= len(data_loader)
            end = time.time()
            lapsed_time = end - start

            if epoch % print_every == 0:
```

```
                print(f'Epoch {epoch} took {lapsed_time} seconds\n')
                print('[EPOCH %d] Loss = %.5f' % (epoch, epoch_loss))

            if epoch % test_every == 0:
                # TRAIN ACCURACY
                pred = self.predict(train_x)
                correct = len(np.where(pred == train_y)[0])
                total = len(train_y)
                train_acc = correct / total
                self.train_accuracy.append(train_acc)

                # VAL ACCURACY
                pred = self.predict(valid_x)
                correct = len(np.where(pred == valid_y)[0])
                total = len(valid_y)
                valid_acc = correct / total
                self.valid_accuracy.append(valid_acc)

                if best_acc < valid_acc:
                    best_acc = valid_acc
                    best_epoch = epoch
                    torch.save(self.state_dict(), './best_model/AlexNet.pt')
                if epoch % print_every == 0:
                    print('Train Accuracy = %.3f' % train_acc + ' // ' + 'Valid Ac
curacy = %.3f' % valid_acc)
                    if best_acc < valid_acc:
                        print('Best Accuracy updated (%.4f => %.4f)' % (best_acc,
valid_acc))
        print('Training Finished...!!')
        print('Best Valid acc : %.2f at epoch %d' % (best_acc, best_epoch))

        return best_acc

    def restore(self):
        with open(os.path.join('./best_model/AlexNet.pt'), 'rb') as f:
            state_dict = torch.load(f)
        self.load_state_dict(state_dict)

    def plot_accuracy(self):
        """
            Draw a plot of train/valid accuracy.
            X-axis : Epoch
            Y-axis : train_accuracy & valid_accuracy
            Draw train_acc-epoch, valid_acc-epoch graph in 'one' plot.
        """
        epochs = list(np.arange(1, self.num_epochs+1, self.test_every))
```

```
        plt.plot(epochs, self.train_accuracy, label='Train Acc.')
        plt.plot(epochs, self.valid_accuracy, label='Valid Acc.')

        plt.title('Epoch - Train/Valid Acc.')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.show()
```

(b) [ResNet] Implement ResNet-18 in 'model/ResNet.py'. Please refer the lecture slide 'W10 Modern ConvNets.pdf' 23~32 p.



layer name	output size	18-layer
conv1	112×112	
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
	1×1	
FLOPs		1.8×10^9

Answer: Fill your code here. You also have to submit your code to i-campus.

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import time
import os
import numpy as np
import matplotlib.pyplot as plt
from torchvision.transforms.functional import resize
from tqdm import tqdm

# W10 Modern ConvNets.pdf - 23 page
# https://pytorch.org/assets/images/resnet.png
```

```
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=(3, 3), stri
de=stride, padding=1, bias=False)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=(3, 3), str
ide=1, padding=1, bias=False)
        if in_channels == out_channels:
            self.downsample = None
        else:
            self.downsample = nn.Conv2d(in_channels, out_channels, kernel_size=(1,
 1), stride=stride, bias=False)

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.relu(out)
        out = self.conv2(out)

        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        out = self.relu(out)

        return out

class ResNet(nn.Module):
    def __init__(self, input_channel, output_dim, learning_rate, reg_lambda, devic
e):
        super(ResNet, self).__init__()

        self.output_dim = output_dim
        self.device = device
        self.loss_function = None
        self.optimizer = None

        self.CONV1 = nn.Conv2d(in_channels=input_channel, out_channels=64, kernel_
size=(7, 7), stride=2, padding=3)
        self.POOL1 = nn.MaxPool2d(kernel_size=(3, 3), stride=2, padding=1)

        # You can implement ResNet-18 more simply using BasicBlock Module.
        # ================================ EDIT HERE ================================
=
```

```
        self.CONV2_x = nn.Sequential(BasicBlock(in_channels=64, out_channels=64),
BasicBlock(in_channels=64, out_channels=64))

        self.CONV3_x = nn.Sequential(BasicBlock(in_channels=64, out_channels=128,
stride=2), BasicBlock(in_channels=128, out_channels=128))

        self.CONV4_x = nn.Sequential(BasicBlock(in_channels=128, out_channels=256,
 stride=2), BasicBlock(in_channels=256, out_channels=256))

        self.CONV5_x = nn.Sequential(BasicBlock(in_channels=256, out_channels=512,
 stride=2), BasicBlock(in_channels=512, out_channels=512))

        self.POOL2 = nn.AvgPool2d(kernel_size=(7,7), stride=2)

        self.FC1 = nn.Linear(512, output_dim)

        self.Conv_layers = nn.Sequential(self.CONV1, nn.ReLU(), self.POOL1, self.C
ONV2_x, self.CONV3_x, self.CONV4_x, self.CONV5_x, self.POOL2)

        self.loss_function = nn.CrossEntropyLoss()
        self.optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate, wei
ght_decay=reg_lambda)
        # ============================== EDIT HERE ==============================
=

    def forward(self, x):
        out = torch.zeros((x.shape[0], self.output_dim))

        # ============================== EDIT HERE ==============================
=
        h = self.Conv_layers(x)
        stretched_h = h.reshape(x.shape[0], -1)
        out = self.FC1(stretched_h)
        # ============================== EDIT HERE ==============================
=

        return out

    def predict(self, x):
        pred_y = np.zeros((x.shape[0], ))
        pred_y = []
        x_tenser = torch.tensor(x, dtype=torch.float, device=self.device)
        data_loader = DataLoader(x_tenser, batch_size=self.batch_size)
        with torch.no_grad():
            for batch_data in data_loader:
                batch_x = batch_data
                batch_x = resize(batch_x, (224, 224))
```

```
                batch_pred = self.forward(batch_x).argmax(axis=1)
                pred_y.append(batch_pred.cpu().numpy())
        pred_y = np.concatenate(pred_y, axis=0)
        return pred_y

    def train(self, train_x, train_y, valid_x, valid_y, num_epochs, batch_size, te
st_every=10, print_every=10):
        self.train_accuracy = []
        self.valid_accuracy = []
        best_epoch = -1
        best_acc = -1
        self.num_epochs = num_epochs
        self.test_every = test_every

        # transfrom numpy data to torch data and make torch dataset
        x_tenser = torch.tensor(train_x, dtype=torch.float, device=self.device)
        y_tenser = torch.tensor(train_y, dtype=torch.long, device=self.device)
        dataset = TensorDataset(x_tenser, y_tenser)

        data_loader = DataLoader(dataset, batch_size=batch_size)
        self.batch_size = batch_size

        for epoch in range(1, num_epochs+1):
            start = time.time()
            epoch_loss = 0.0
            # model Train
            for b, batch_data in enumerate(data_loader):
                batch_x, batch_y = batch_data
                batch_x = resize(batch_x, (224, 224))
                pred_y = self.forward(batch_x)

                loss = self.loss_function(pred_y, batch_y)
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
                epoch_loss += loss

            epoch_loss /= len(data_loader)
            end = time.time()
            lapsed_time = end - start

            if epoch % print_every == 0:
                print(f'Epoch {epoch} took {lapsed_time} seconds\n')
                print('[EPOCH %d] Loss = %.5f' % (epoch, epoch_loss))

            if epoch % test_every == 0:
                # TRAIN ACCURACY
```

```
                pred = self.predict(train_x)
                correct = len(np.where(pred == train_y)[0])
                total = len(train_y)
                train_acc = correct / total
                self.train_accuracy.append(train_acc)

                # VAL ACCURACY
                pred = self.predict(valid_x)
                correct = len(np.where(pred == valid_y)[0])
                total = len(valid_y)
                valid_acc = correct / total
                self.valid_accuracy.append(valid_acc)

                if best_acc < valid_acc:
                    best_acc = valid_acc
                    best_epoch = epoch
                    torch.save(self.state_dict(), './best_model/ResNet.pt')
                if epoch % print_every == 0:
                    print('Train Accuracy = %.3f' % train_acc + ' // ' + 'Valid Ac
curacy = %.3f' % valid_acc)
                    if best_acc < valid_acc:
                        print('Best Accuracy updated (%.4f => %.4f)' % (best_acc,
valid_acc))
        print('Training Finished...!!')
        print('Best Valid acc : %.2f at epoch %d' % (best_acc, best_epoch))

        return best_acc

    def restore(self):
        with open(os.path.join('./best_model/ResNet.pt'), 'rb') as f:
            state_dict = torch.load(f)
        self.load_state_dict(state_dict)

    def plot_accuracy(self):
        """
            Draw a plot of train/valid accuracy.
            X-axis : Epoch
            Y-axis : train_accuracy & valid_accuracy
            Draw train_acc-epoch, valid_acc-epoch graph in 'one' plot.
        """
        epochs = list(np.arange(1, self.num_epochs+1, self.print_every))

        plt.plot(epochs, self.train_accuracy, label='Train Acc.')
        plt.plot(epochs, self.valid_accuracy, label='Valid Acc.')

        plt.title('Epoch - Train/Valid Acc.')
        plt.xlabel('Epochs')
```

```
        plt.ylabel('Accuracy')
        plt.legend()

        plt.show()
```

(2) [20 pts] Experiment results

(a) [Random Search with MNIST] Adjust the model settings (# of hidden layers, # of hidden nodes, # of epochs, learning rate, etc.) with random search to get the best results over MNIST dataset using 'main_random_search.py'. Report your best valid accuracy, the model setting, and the search space. Explain how you determined the search space of hyperparameters in a couple of lines.

[Model Hyperparameters]

	Search Space	# of epochs	Learning rate	L2 reg. lambda	Batch size	Best Validation Acc.
LeNet-5	**num_epochs_list = [10, 15, 20, 25, 30]** **learning_rate_list = [0.01, 0.001, 0.0001]** **reg_lambda_list = [0.01, 0.001, 0.0001]** **batch_size_list = [100]** **num_search = 30**	30	0.01	0.0001	100	0.982
AlexNet	**num_epochs_list = [5, 10, 15]** **learning_rate_list = [0.001, 0.0005, 0.0001]** **reg_lambda_list = [0.01, 0.001, 0.0001]** **batch_size_list = [100]** **num_search = 30**	15	0.0001	0.0001	100	0.982
ResNet	**num_epochs_list = [10, 15]** **learning_rate_list = [0.001, 0.0001]** **reg_lambda_list = [0.001, 0.0001]** **batch_size_list = [32, 64]** **num_search = 30**	15	0.001	0.0001	64	0.982

각 model 별로 main_classification.py를 이용하여 적절한 learning rate를 찾은 후, 그 결과를 바탕으로 random_search.py를 이용하여 best parameter를 찾았다. LeNet-5는 AlexNet, ResNet가 다르게

높은 learning rate에서 학습이 잘 되었고, AlexNet, ResNet은 LeNet-5처럼 learning rate가 높으면 학습이 잘 안되었다. batch size는 줄여도 비슷한 성능이 나왔고, googlge colaboratory에서 제공하는 RAM 크기의 제한으로 100에서 더 늘려서 실험해볼 수는 없었다.

(b) [CNN with Fashion MNIST] Choose a model and adjust the model settings (# of hidden layers, # of hidden nodes, # of epochs, learning rate, etc.) to get the best results over FashionMNIST dataset using 'main_classification.py.' Report your best test accuracy with your model and fine-tuned hyperparameters. Explain how you determined the model structure or parameters in a couple of lines.

[Model Hyperparameters]

	Model	# of epochs	Learning rate	L2 reg. lambda	Batch size	Best Validation Acc.	Final Test Acc.
1st Best	**ResNet**	**30**	0.0001	0.0001	100	0.93	0.92
2nd Best	**AlexNet**	30	0.0001	0.0001	100	0.93	0.92
3rd Best	**AlexNet**	30	0.001	0.001	100	0.92	0.91

# of epochs를 100으로 설정하고 실험을 해본 결과, epoch=30부터 loss가 크게 변하지 않아서 # of epochs는 30으로 설정하였다. (2)(a)를 진행하면서 ResNet과 AlexNet 모두 learning rate가 큰 경우 학습이 잘 안 된다는 사실을 깨닫고 learning rate를 상대적으로 낮게 설정하였다. (높게 설정해서 실험해본 결과, 실제로 성능이 떨어졌다.) batch size는 정확도에 큰 영향을 주지 않는 것으로 확인되어 모두 100으로 설정하였다.