**Introduction to Deep Neural Networks (Spring 2021)**

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

**Student ID 2016312761**

**Name 여혁수**

**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.

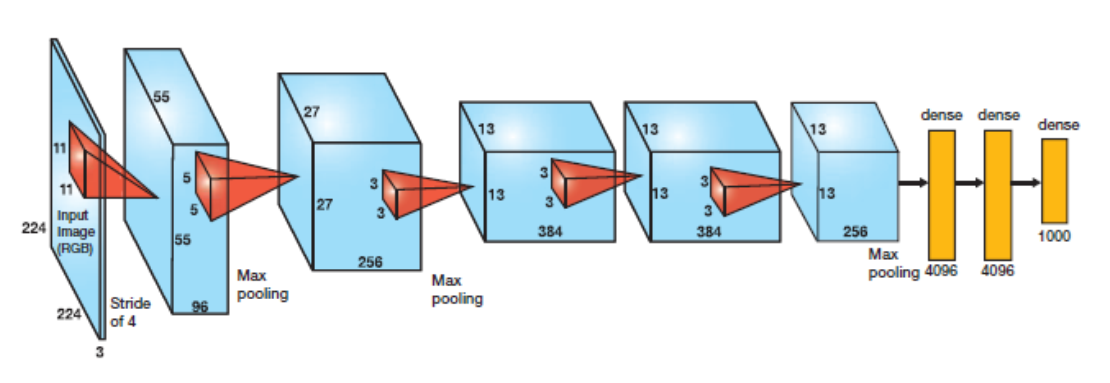
Kor 1. https://tutorials.pytorch.kr/beginner/blitz/tensor\_tutorial.html#sphx-glr-beginner-blitz-tensor-tutorial-py

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

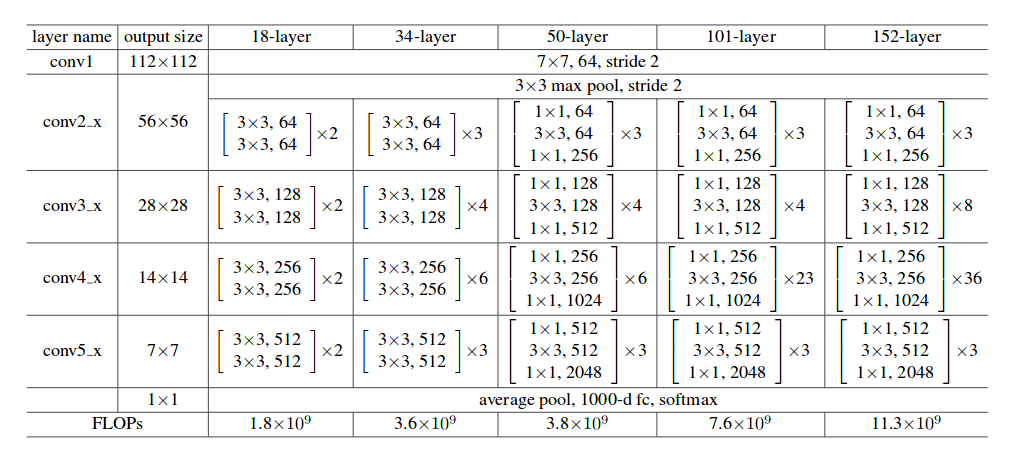
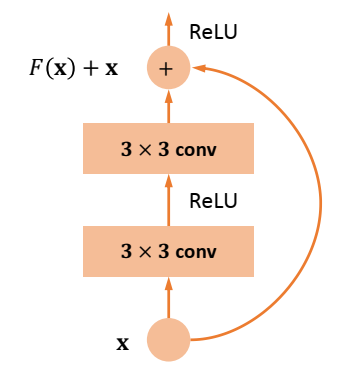
Eng 1. https://pytorch.org/tutorials/beginner/pytorch\_with\_examples.html

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

1. **[30 pts]** Implement CNN models in ‘AlexNet.py’ and ‘ResNet.py.’
2. **[AlexNet]** Implement AlexNet in ‘models/AlexNet.py’. Please refer the lecture slide ‘W09 Convolutional Neural Networks (CNNs).pdf’ 8~25 p.



1. **[ResNet]** Implement ResNet-18 in ‘model/ResNet.py’. Please refer the lecture slide ‘W10 Modern ConvNets.pdf’ 23~32 p.



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

|  |
| --- |
| **Alexnet)**  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, device):          super(AlexNet, self).\_\_init\_\_()          self.output\_dim = output\_dim          self.device = device          self.loss\_function = None          self.optimizer = None          # =============================== EDIT HERE ===============================          self.features = nn.Sequential(              nn.Conv2d(input\_channel, 96, kernel\_size = (11, 11), stride = 4, padding = 0),              nn.ReLU(),              nn.MaxPool2d(kernel\_size = 3, stride = 2),                nn.Conv2d(96, 256, kernel\_size = (5, 5), stride = 1, padding = 2),              nn.ReLU(),              nn.MaxPool2d(kernel\_size = 3, stride = 2),                nn.Conv2d(256, 384, kernel\_size = (3, 3), stride = 1, padding = 1),              nn.ReLU(),                nn.Conv2d(384, 384, kernel\_size = (3, 3), stride = 1, padding = 1),              nn.ReLU(),              nn.Conv2d(384, 256, kernel\_size = (3, 3), stride = 1, padding = 1),              nn.ReLU(),              nn.MaxPool2d(kernel\_size = 3, stride = 2),          )            self.classifier = nn.Sequential(              nn.Linear(6\*6\*256, 4096),              nn.ReLU(inplace = True),              nn.Linear(4096, 4096),              nn.ReLU(inplace = True),              nn.Linear(4096, self.output\_dim),          )          self.loss\_function = nn.CrossEntropyLoss()          self.optimizer = torch.optim.Adam(self.parameters(), lr=learning\_rate, weight\_decay=reg\_lambda)          # =============================== EDIT HERE ===============================        def forward(self, x):          out = torch.zeros((x.shape[0], self.output\_dim))          # =============================== EDIT HERE ===============================          x = self.features(x)          x = x.view(-1, 256 \* 6 \* 6)          out = self.classifier(x)          # =============================== 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, test\_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 Accuracy = %.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()  **Resnet)**  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), stride=stride, padding=1, bias=False)          self.relu = nn.ReLU()          self.conv2 = nn.Conv2d(out\_channels, out\_channels, kernel\_size=(3, 3), stride=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, device):          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.in\_planes = 64          self.relu = nn.ReLU()          self.layer1 = self.\_make\_layer(BasicBlock, 64, 2, stride=1)          self.layer2 = self.\_make\_layer(BasicBlock, 128, 2, stride=2)          self.layer3 = self.\_make\_layer(BasicBlock, 256, 2, stride=2)          self.layer4 = self.\_make\_layer(BasicBlock, 512, 2, stride=2)          self.linear = nn.Linear(512, self.output\_dim)          self.loss\_function = nn.CrossEntropyLoss()          self.optimizer = torch.optim.Adam(self.parameters(), lr=learning\_rate, weight\_decay=reg\_lambda)        def \_make\_layer(self, block, planes, num\_blocks, stride):          strides = [stride] + [1]\*(num\_blocks-1)          layers = []          for stride in strides:              layers.append(block(self.in\_planes, planes, stride))              self.in\_planes = planes          return nn.Sequential(\*layers)          # =============================== EDIT HERE ===============================      def forward(self, x):          out = torch.zeros((x.shape[0], self.output\_dim))          # =============================== EDIT HERE ===============================            out = self.relu(self.POOL1(self.CONV1(x)))          out = self.layer1(out)          out = self.layer2(out)          out = self.layer3(out)          out = self.layer4(out)          out = torch.nn.functional.avg\_pool2d(out, 4)          out = out.view(out.size(0), -1)          out = self.linear(out)            # =============================== 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, test\_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 Accuracy = %.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() |

1. **[20 pts]** Experiment results
2. **[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 = [30, 40, 50]  learning\_rate\_list = [0.001, 0.002, 0.005]  reg\_lambda\_list = [0.0005, 0.001]  batch\_size\_list = [40, 50, 60]  num\_search = 5 | 38 | 0.001 | 0.0005 | 40 | 0.974 |
| **AlexNet** | num\_epochs\_list = [30, 40, 50]  learning\_rate\_list = [0.001, 0.002, 0.005]  reg\_lambda\_list = [0.0005, 0.001]  batch\_size\_list = [40, 50, 60]  num\_search = 5 | 38 | 0.001 | 0.0005 | 40 | 0.984 |
| **ResNet** | num\_epochs\_list = [30, 40, 50]  learning\_rate\_list = [0.001, 0.002, 0.005]  reg\_lambda\_list = [0.0005, 0.001]  batch\_size\_list = [40, 50, 60]  num\_search = 5 | 37 | 0.001 | 0.0005 | 40 | 0.984 |

|  |
| --- |
| At first running time, I defined batch\_size parameter to 100. But there is no learning improvement, that is, the accuracy and loss are fixed to about 10%, 2.3 each. So I think the reason of this is dying relu effect caused by big batch\_size and backward operation. So I decreased the batch size and instead I also decreased learning rate and reg\_lambda. First I started with (learning\_rate, reg\_lambda) -> (0.1, 0.01), but I continuously decrease to (0.001, 0.0005). I define the number of epoch 30~50, and this is truly proper. |

1. **[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** | 27 | 0.001 | 0.001 | 100 | 0.911 | 0.90 |
| **2nd Best** | **Lenet** | 39 | 0.01 | 0.0005 | 48 | 0.881 | 0.87 |
| **3rd Best** | **Alexnet** | 15 | 0.01 | 0.0005 | 48 | 0.853 | 0.84 |

|  |
| --- |
| I decided all parameters based on question (a), so I keep value of parameters like learning\_rate=0.01, reg\_lambda=0.0005, batch\_size=48. In case of Alexnet, after 15 epochs, accuracy goes 10% suddenly. So learning can’t be continued. However, in case of Lenet, learning progressed stably in same parameter. Lastly, I decreased learning rate and increased batch size to stably learning with Resnet model. I don’t know why learning of model goes bad according to parameters. |