alexnet

# example code

|  |
| --- |
| # 파이토치와 토치비전 임포트  import torch  import torchvision  import torchvision.transforms as transforms  print(torch.\_\_version\_\_)  print(torchvision.\_\_version\_\_)  transform = transforms.Compose([      transforms.Resize(256), # 256의 크기로 이미지를 늘린다      transforms.CenterCrop(224), # 224만큼 중앙에서부터 자른다(아마 이미지 중에서도 물체가 실제로 존재하는 더 중앙부분을 잘라내는것 같습니다)      transforms.ToTensor(), # 데이터셋을 Tensor 정태로 변환(동시에 값이 0 ~ 1로 정규화됨)      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), # ImageNet Pytorch 공식 정규화 값이라고 합니다  ])  batch\_size = 4  # 토치비전을 통해 CIFAR10 데이터셋을 다운로드받고 그것을 trainset에 저장  # train 피라미터에 True 값을 넘김으로써 이것은 train 데이터셋이 됨  trainset = torchvision.datasets.CIFAR10(root='./data', train=True,                                          download=True, transform=transform)  # 50000(train dataset size) / 4(batch\_size) = 12500번의 iteration을 수행하면 모든 데이터를 다 볼 수 있음  # shuffle = True 로 데이터들을 섞음  # num\_workers는 CPU, GPU의 멀티쓰레딩과 관련된 부분인 것 같은데 자세한 내용은 https://jybaek.tistory.com/799 참고  trainloader = torch.utils.data.DataLoader(trainset, batch\_size=batch\_size,                                            shuffle=True, num\_workers=2)  # 이번엔 test 데이터셋을 다운받고 변수에 할당  testset = torchvision.datasets.CIFAR10(root='./data', train=False,                                         download=True, transform=transform)  # testloader는 shuffle = False로 데이터를 섞지 않음 (라벨과 관련이 되어있음)  testloader = torch.utils.data.DataLoader(testset, batch\_size=batch\_size,                                           shuffle=False, num\_workers=2)  # class들을 정의  classes = ('plane', 'car', 'bird', 'cat',             'deer', 'dog', 'frog', 'horse', 'ship', 'truck')  import torch.nn as nn  # torch hub에서 alexNet을 가져옴  # model = torch.hub.load('pytorch/vision:v0.9.0', 'alexnet', pretrained=True)  # 위 코드는 쉽게 모델을 가져올 수 있지만 코랩에서 Http 요청 제한을 넘겨 오류가 발생했음  # 그래서 torchvision.models.alexnet 을 사용함  model\_conv = torchvision.models.alexnet(pretrained = True)  # 클래스가 10개이므로 선형 회귀를 통해 10으로 줄어들도록 AlexNet 모델을 변경  model\_conv.classifier[4] = nn.Linear(4096,1024)  model\_conv.classifier[6] = nn.Linear(1024,10)    # 모델 구조 출력  model\_conv.eval()  # CUDA 사용  device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  print(device)  # 모델을 CUDA에 맞게 작동하도록 변경  model\_conv = model\_conv.to(device)  import torch.optim as optim  # 크로스 엔트로피 손실 함수 선언  criterion = nn.CrossEntropyLoss()  # SGD 옵티마이저를 사용  optimizer = optim.SGD(model\_conv.parameters(), lr=0.001, momentum=0.9)  for epoch in range(10): # 에포크를 2로 설정      running\_loss = 0.0      for i, data in enumerate(trainloader, 0): # 12500번 수행          # inputs와 labels에 data를 나눠서 저장          inputs, labels = data          # inputs 과 labels도 CUDA에 맞게 변환          inputs = inputs.to(device)          labels = labels.to(device)          # zero the parameter gradients          optimizer.zero\_grad()          # 모든 매개변수의 변화도 버퍼를 0으로 만듦(다음에 나오는 backward와 관련이 있음)          # 0으로 만들지 않으면 backward를 할때마다 변화도가 누적되기 때문          outputs = model\_conv(inputs) # inputs 를 model\_conv(AlexNet) 신경망에 통과시킴          # 전에 선언했던 크로스 엔트로피 손실 함수를 이용해서 loss 를 평가          loss = criterion(outputs, labels)            # 오차(손실)를 역전파(모델의 매개변수들에 대한 손실의 변화도를 계산)          loss.backward()            optimizer.step() # 모든 매개변수를 갱신          # print statistics          running\_loss += loss.item()          if i % 2000 == 1999:    # 2000번 마다 알림              print('[%d, %5d] loss: %.3f' %                    (epoch + 1, i + 1, running\_loss / 2000)) # loss의 평균 출력              running\_loss = 0.0              # running\_loss 값 초기화  print('Finished Training')  correct = 0  total = 0  with torch.no\_grad(): # 훈련중이 아니므로 no\_grad      for data in testloader:          images, labels = data[0].to(device), data[1].to(device)          outputs = model\_conv(images)          \_, predicted = torch.max(outputs.data, 1)          total += labels.size(0)          correct += (predicted == labels).sum().item()  print('Accuracy of the network on the 10000 test images: %d %%' % (      100 \* correct / total))  correct\_pred = {classname: 0 for classname in classes}  total\_pred = {classname: 0 for classname in classes}  # 여기서도 또한 필요없음  with torch.no\_grad():      for data in testloader:          images, labels = data[0].to(device), data[1].to(device)          outputs = model\_conv(images)          \_, predictions = torch.max(outputs, 1)          # collect the correct predictions for each class          for label, prediction in zip(labels, predictions):              if label == prediction:                  correct\_pred[classes[label]] += 1              total\_pred[classes[label]] += 1  # print accuracy for each class  for classname, correct\_count in correct\_pred.items():      accuracy = 100 \* float(correct\_count) / total\_pred[classname]      print("Accuracy for class {:5s} is: {:.1f} %".format(classname,                                                     accuracy)) |

# testing result

|  |
| --- |
| [1, 2000] loss: 1.157  [1, 4000] loss: 0.893  [1, 6000] loss: 0.773  [1, 8000] loss: 0.701  [1, 10000] loss: 0.676  [1, 12000] loss: 0.628  [2, 2000] loss: 0.484  [2, 4000] loss: 0.496  [2, 6000] loss: 0.492  [2, 8000] loss: 0.477  [2, 10000] loss: 0.472  [2, 12000] loss: 0.466  [3, 2000] loss: 0.317  [3, 4000] loss: 0.330  [3, 6000] loss: 0.342  [3, 8000] loss: 0.352  [3, 10000] loss: 0.331  [3, 12000] loss: 0.343  [4, 2000] loss: 0.201  [4, 4000] loss: 0.231  [4, 6000] loss: 0.242  [4, 8000] loss: 0.254  [4, 10000] loss: 0.264  [4, 12000] loss: 0.269  [5, 2000] loss: 0.162  [10, 6000] loss: 0.111  [10, 8000] loss: 0.105  [10, 10000] loss: 0.121  [10, 12000] loss: 0.107  Finished Training  Accuracy for class plane is: 87.3 %  Accuracy for class car is: 85.5 %  Accuracy for class bird is: 71.6 %  Accuracy for class cat is: 77.3 %  Accuracy for class deer is: 81.9 %  Accuracy for class dog is: 80.4 %  Accuracy for class frog is: 74.4 %  Accuracy for class horse is: 90.0 %  Accuracy for class ship is: 89.2 %  Accuracy for class truck is: 90.3 % |