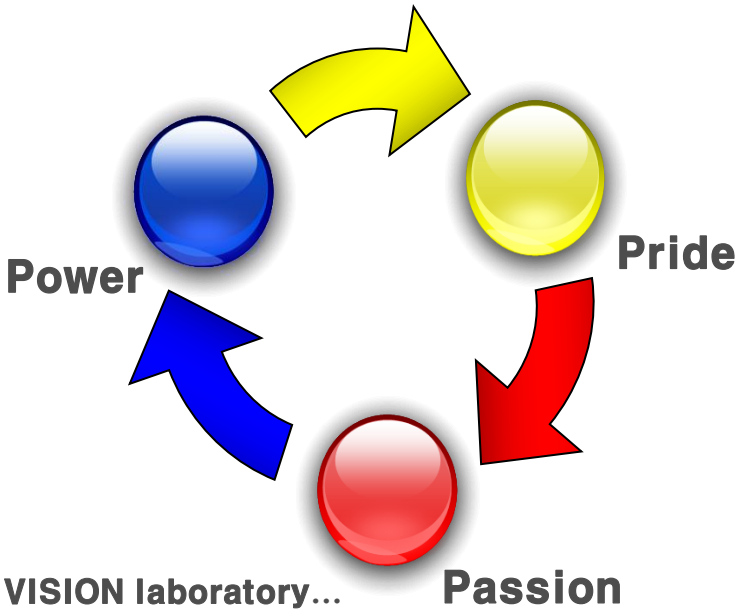



# Training a classifier
# CIFAR10 dataset 분류기 구현)

Vision System Lab, Gyumin Park
yywnnaa@gmail.com
Jan 17, 2024

Soongsil University
VISION SYSTEM LABOARTORY

# import

```
import torch
import torchvision
from torch import nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torch.utils.data import Dataset
from torchvision import datasets,transforms, models
from torchvision.transforms import ToTensor
from torchvision.io import read_image

import os
import pandas as pd
import numpy as np

from PIL import Image
```

[1] ✓ 3.0s

VISION SYSTEM LABORATORY

# 작업 환경

```python
# Get cpu, gpu or mps device for training
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

print("현재 작업 디렉토리 :", os.getcwd())
print(torch.cuda.is_available())
print(torch.cuda.device_count())  # 1 이상의 숫자가 출력되어야 함
print(torch.cuda.get_device_name(0))  # GPU 이름
print("cuda version :", torch.version.cuda)
print(torch.__version__)
```

[9] ✓ 0.0s

```
Using cuda device
현재 작업 디렉토리: c:\Users\Tzuyu\Desktop\PythonWorkspace
True
1
NVIDIA GeForce GTX 1660 SUPER
12.1
2.1.2+cu121
```

```
pip install torch==2.1.2+cu121 torchvision==0.16.2+cu121
torchaudio==2.1.2+cu121 -f
https://download.pytorch.org/whl/cu121/torch_stable.html
```

# Data Augmentation : flip

```python
# 데이터프레임 생성
df = pd.DataFrame({'img_id': [f'{i:05d}' for i in range(50000)]}) # train data 50000개
df['label_path'] = df['img_id'].apply(lambda x: os.path.join(root_dir, f'{x}.txt'))

# 레이블 파일에서 레이블을 읽어와 데이터프레임에 추가
df['label'] = df['label_path'].apply(lambda path: int(open(path, 'r').readline().strip()) if os.path.exists(path) else None)

# 레이블이 있는 데이터만 사용
df = df.dropna()

# 좌우 반전 및 새로운 파일명으로 저장 (50000~99999)
for idx in range(len(df)):
    img_id = df.iloc[idx]['img_id']
    img_path = os.path.join(root_dir, f'{img_id}.png')  # 이미지 파일 경로
    label = df.iloc[idx]['label']  # 해당 이미지에 대한 레이블

    # 이미지 불러오기
    image = Image.open(img_path).convert('RGB')

    # 좌우 반전
    flipped_image = image.transpose(Image.FLIP_LEFT_RIGHT)

    # 새로운 파일명 생성
    new_img_id = f'{idx + 50000:05d}' # 원본 인덱스 + 50000 # 0을 뒤집은 이미지 -> 50000 # 21456을 뒤집은 이미지 -> 71456
    new_img_path = os.path.join(root_dir, f'{new_img_id}.png') # root_dir = 'cifar10/train'
    new_label_path = os.path.join(root_dir, f'{new_img_id}.txt')

    # 저장
    flipped_image.save(new_img_path)
    with open(new_label_path, 'w') as label_file:
        label_file.write(str(label))  # 해당 이미지의 원본 레이블 값을 새로운 레이블 파일에 저장
```

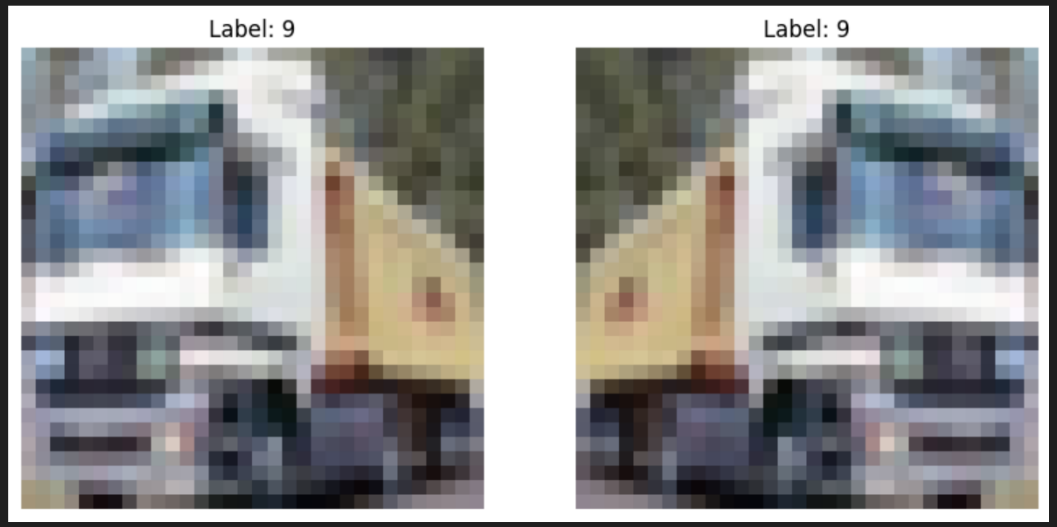
이미지 전체를 좌우로 뒤집은 후 새로 저장

-> train 데이터 50000+50000 = 100000개

# Data Augmentation : flip (**결과 확인**)

```python
# 제대로 실행되었는지 확인
import matplotlib.pyplot as plt
# 이미지 불러오기
img_id_0 = '00001'
img_path_0 = os.path.join(root_dir, f'{img_id_0}.png')
label_0 = int(open(os.path.join(root_dir, f'{img_id_0}.txt'), 'r').readline().strip())
image_0 = Image.open(img_path_0).convert('RGB')

img_id_50000 = '50001' # 00001 + 50000
img_path_50000 = os.path.join(root_dir, f'{img_id_50000}.png')
label_50000 = int(open(os.path.join(root_dir, f'{img_id_50000}.txt'), 'r').readline().strip())
image_50000 = Image.open(img_path_50000).convert('RGB')

# 이미지와 레이블 표시
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(image_0)
axes[0].set_title(f"Label: {label_0}")
axes[0].axis('off')

axes[1].imshow(image_50000)
axes[1].set_title(f"Label: {label_50000}")
axes[1].axis('off')

plt.show()
```

[4] ✓ 0.7s




99989
99990
99990
99991
99991
99992
99992
99993
99993
99994
99994
99995
99995
99996
99996
99997
99997
99998
99998
99999
99999

크기: 225MB (236,809,987 바이

디스크 할당 크기: 390MB (409,550,848 바이

내용: 파일 200,000, 폴더 0

만든 날짜: 2024년 1월 12일 금요일,

특성: ■ 읽기 전용(폴더의 파일

□ 숨김(H)

확인

2024-01-
2024-01-
2024-01-

# 데이터셋 불러오기 (코드 수정 과정)

```
class CustomDatasetCombined(Dataset):
    def __init__(self, root_dir, subset, transform=None, target_transform=None):
        self.root_dir = root_dir
        self.subset = subset  # train 또는 test로 지정하여 데이터셋을 선택 근데 root_dir로 판단해도 될듯
        self.transform = transform
        self.target_transform = target_transform
```

수정 전

```
        # 데이터프레임 생성
        if self.subset == 'train':
            df = pd.DataFrame({'img_id': [f'{i:05d}' for i in range(100000)]})
            df['label_path'] = df['img_id'].apply(lambda x: os.path.join('cifar10/train', f'{x}.txt'))
        elif self.subset == 'test':
            df = pd.DataFrame({'img_id': [f'{i:05d}' for i in range(10000)]})
            df['label_path'] = df['img_id'].apply(lambda x: os.path.join('cifar10/test', f'{x}.txt'))
```

수정 후

```
        if self.root_dir == 'cifar10/train':
            range_num = 100000
        elif self.root_dir == 'cifar10/test':
            range_num = 10000

        df = pd.DataFrame({'img_id': [f'{i:05d}' for i in range(range_num)]})
        df['label_path'] = df['img_id'].apply(lambda x: os.path.join(root_dir, f'{x}.txt'))

        # 레이블 파일에서 레이블을 읽어와 데이터프레임에 추가
        df['label'] = df['label_path'].apply(lambda path: int(open(path, 'r').readline().strip()))

        self.df = df
```

# 데이터셋 불러오기

```python
class CustomDatasetCombined(Dataset):
    def __init__(self, root_dir, transform=None, target_transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.target_transform = target_transform

        if self.root_dir == 'cifar10/train':
            range_num = 100000
        elif self.root_dir == 'cifar10/test':
            range_num = 10000

        df = pd.DataFrame({'img_id': [f'{i:05d}' for i in range(range_num)]})
        df['label_path'] = df['img_id'].apply(lambda x: os.path.join(root_dir, f'{x}.txt'))

        # 레이블 파일에서 레이블을 읽어와 데이터프레임에 추가
        df['label'] = df['label_path'].apply(lambda path: int(open(path, 'r').readline().strip()))

        self.df = df

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

    def __getitem__(self, idx):
        img_id = self.df.iloc[idx]['img_id']
        img_path = os.path.join(self.root_dir, f'{img_id}.png')  # 이미지 파일 경로
        label = self.df.iloc[idx]['label']  # 해당 이미지에 대한 레이블

        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        if self.target_transform:
            label = self.target_transform(label)

        return image, label
```

✓ 0.0s

# Data Augmentation : crop

```python
batch_size = 32
# 큰 배치 크기는 한 번에 더 많은 데이터를 처리 - 계산 속도 상승 가능, 메모리 많이 필요
# 작은 배치 크기는 모델이 각 배치에 노출되는 데이터가 다양하게 - 모델 더 빠르게 수렴, 더 나은 일반화 성능 가능

def random_crop_with_replicate_padding(img, size):
    # Replicate Padding
    img_array = np.array(img) # 입력 이미지를 numpy 배열로 변환

    img_padded = np.pad(img_array, ((4, 4), (4, 4), (0, 0)), mode='edge')
    # 첫 번째 차원(높이)에 위아래로 각각 4픽셀 / 두 번째 차원(너비)에 좌우로 각각 4픽셀 / 세 번째 차원(채널)에는 패딩을 추가 X
    # mode='edge': 패딩을 할 때 가장자리의 값을 반복하여 사용하는 모드, 'edge' 모드에서는 가장자리 값을 반복하여 패딩

    img_padded = Image.fromarray(img_padded) # 패딩된 numpy 배열을 다시 PIL 이미지로 변환

    # Random Crop
    # 패딩된 이미지에서 크롭할 영역의 시작점 (i, j)와  크기 (h, w)를 반환
    i, j, h, w = transforms.RandomCrop.get_params(img_padded, output_size=size)

    img_cropped = transforms.functional.crop(img_padded, i, j, h, w)
    # transforms.functional.crop : 이미지에서 주어진 좌표 (i, j)와 크기 (h, w)로 정의된 영역을 자르는 함수
    # crop(img: PIL.Image.Image, top: int, left: int, height: int, width: int)
    # i: 크롭을 적용할 영역의 top 좌표 # j: 크롭을 적용할 영역의 left 좌표
    # h: 크롭할 영역의 높이 (height) # w: 크롭할 영역의 너비 (width)

    return img_cropped

######### train

transform_crop = transforms.Compose(
    [transforms.Lambda(lambda x: random_crop_with_replicate_padding(x, size=(32, 32))),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

aug_dataset_2 = CustomDatasetCombined_2(root_dir='cifar10/train', transform=transform_crop)
aug_dataloader_2 = torch.utils.data.DataLoader(aug_dataset_2, batch_size=batch_size, shuffle=True, num_workers=0)
print(f"Number of training samples: {len(aug_dataset_2)}")
```

실수한 부분 : 모든 이미지를 crop해서 사용하는 코드가 만들어짐

(전체 이미지 중 랜덤 비율로 몇 개의 이미지만 crop하도록 만들어야했음)

최종적으로 crop 사용 X

(실험 결과 성능 낮음+ 개인적으로 생각하기에 이미지 크기가 32*32로 너무 작음)

# test_dataset, test_dataloader

crop을 사용하지 않았기 때문에
이 transform을 train dataset에도 적용하였음

```
######### test

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# torchvision 데이터셋의 출력(output)은 [0, 1] 범위를 갖는 PILImage 이미지
# 이를 [-1, 1]의 범위로 정규화된 Tensor로 변환 (공식 문서)

test_dataset_2 = CustomDatasetCombined_2(root_dir='cifar10/test', transform=transform)
test_dataloader_2 = torch.utils.data.DataLoader(test_dataset_2, batch_size=batch_size, shuffle=False, num_workers=0)
print(f"Number of test samples: {len(test_dataset_2)}")
```

✓ 16.7s

```
Number of training samples: 100000
Number of test samples: 10000
```

VISION SYSTEM LABORATORY

# CustomModel

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

# __init__은 모델의 구조를 초기화
# forward는 입력 데이터를 받아 모델을 통과시켜 예측값을 계산

class CustomModel(nn.Module):
    def __init__(self, num_classes):
        super(CustomModel, self).__init__()

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1, padding_mode='replicate')
        # 입력 채널 3, 출력 채널 32, 커널 크기 3, 패딩 1
        # 입력 채널 3 : 컬러 이미지 - 빨강, 초록, 파랑 (RGB) 세 가지 채널
        # 출력 채널 32 : 특징 맵의 수
        # 커널 크기 3 : 각 픽셀에 대해 3x3 크기의 커널 사용
        # 패딩 1 : 커널 크기가 3x3이므로 자연스럽게 패딩 1 / 커널 크기 5x5라면 패딩 2
        # 패딩 파라미터를 입력하지 않으면 기본값으로 0이 사용, 출력이 줄어들게 됨
        # Replicate-padding: 가장자리에 있는 픽셀을 복사하여 패딩을 채움

        # 기본 패딩 (제로패딩) 정확도 : % (100에포크, adam)
        # replicate 패딩 정확도      : % (100에포크, adam)

        self.batch_norm1 = nn.BatchNorm2d(32)
        # 32 : 배치 정규화 레이어에 입력되는 채널의 수
        # 32개의 채널 각각에 대해 배치 정규화 수행
        # 각 채널이 독립적으로 정규화되어 네트워크가 더 안정적으로 학습되도록 도와주는 효과

        self.conv2 = nn.Conv2d(32, 32, kernel_size=3)
        self.batch_norm2 = nn.BatchNorm2d(32)

        self.maxpool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        # 2x2 크기의 풀링 윈도우, 보폭 2 - 입력 데이터를 2픽셀씩 이동하면서 최댓값을 추출

        self.dropout1 = nn.Dropout(0.25)
        # 25%의 노드가 랜덤하게 비활성화
```

# CustomModel

```
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.batch_norm3 = nn.BatchNorm2d(64)

        self.conv4 = nn.Conv2d(64, 64, kernel_size=3)
        self.batch_norm4 = nn.BatchNorm2d(64)

        self.maxpool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout2 = nn.Dropout(0.25)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64 * 6 * 6, 512)
        # 64 * 6 * 6 : 앞선 레이어를 거치면서 얻은 출력 특징 맵의 크기
        # 512: 이 fully connected 레이어의 출력 크기, 512개의 노드로 연결

        self.batch_norm5 = nn.BatchNorm1d(512)
        self.dropout3 = nn.Dropout(0.5)

        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = F.relu(self.batch_norm1(self.conv1(x)))
        x = F.relu(self.batch_norm2(self.conv2(x)))
        x = self.maxpool1(x)
        x = self.dropout1(x)

        x = F.relu(self.batch_norm3(self.conv3(x)))
        x = F.relu(self.batch_norm4(self.conv4(x)))
        x = self.maxpool2(x)
        x = self.dropout2(x)

        x = self.flatten(x)
        x = F.relu(self.batch_norm5(self.fc1(x)))
        x = self.dropout3(x)

        x = self.fc2(x)
        return F.softmax(x, dim=1)
```

✓ 0.0s Python

# 손실 함수, optimizer

```python
gmnet_2 = CustomModel(num_classes=10).to(device)

# 손실 함수 및 optimizer 설정
criterion = nn.CrossEntropyLoss()

#optimizer = optim.SGD(gmnet.parameters(), lr=0.001, momentum=0.9)
#optimizer = optim.RMSprop(gmnet.parameters(), lr=0.001, alpha=0.9, weight_decay=0.01)
optimizer = optim.Adam(gmnet_2.parameters(), lr=0.001)

print(gmnet_2)
```

✓ 0.0s

```
CustomModel(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=replicate)
  (batch_norm1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (batch_norm2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout(p=0.25, inplace=False)
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batch_norm3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (batch_norm4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout2): Dropout(p=0.25, inplace=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=2304, out_features=512, bias=True)
  (batch_norm5): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)
```

# Train 함수 정의

```python
# 훈련 함수 정의
def train(gmnet_2, aug_dataloader_2, criterion, optimizer, device):
    gmnet_2.train() # 모델을 학습 모드로 설정 : 드롭아웃과 같은 학습 중에만 활성화되어야 하는 연산들을 활성화
    running_loss = 0.0 # 현재 미니배치까지의 누적 손실을 저장

    for inputs, labels in aug_dataloader_2: # 입력 데이터와 레이블을 가져옴
        inputs, labels = inputs.to(device), labels.to(device) #  데이터를 GPU로 이동

        optimizer.zero_grad() # 기울기 초기화

        # 모델을 통과한 결과를 얻고 손실을 계산
        outputs = gmnet_2(inputs)
        loss = criterion(outputs, labels)
        loss.backward()  # 역전파를 수행하여 기울기를 계산
        optimizer.step() # 최적화를 수행하여 가중치를 업데이트

        running_loss += loss.item() # 현재 미니배치의 손실을 누적

    return running_loss / len(aug_dataloader_2) # 전체 훈련 데이터에 대한 평균 손실을 반환
```

# Test 함수 정의

```python
# 테스트 함수 정의
def test(gmnet_2, test_dataloader_2, criterion, device):
    gmnet_2.eval()            # 모델을 평가 모드로 설정
    correct_predictions = 0 # 올바르게 예측된 총 샘플 수를 저장
    total_samples = 0       # 총 테스트 샘플 수를 저장

    # 그래디언트 계산을 비활성화하고 테스트 데이터로부터 입력 데이터와 레이블을 가져옴
    with torch.no_grad():
        for inputs, labels in test_dataloader_2:
            inputs, labels = inputs.to(device), labels.to(device)

            # 모델을 통과한 결과를 얻고, 가장 높은 확률을 가진 클래스로 예측
            # 정확한 예측 수와 전체 샘플 수를 업데이트
            outputs = gmnet_2(inputs)

            _, predicted = torch.max(outputs, 1)
            # torch.max(outputs, 1) : 모델의 출력에서 각 행마다 최대값과 해당 최대값의 인덱스를 반환
            # _ : 최대값, predicted : 각 입력 샘플에 대한 예측된 클래스의 인덱스를 나타냄

            correct_predictions += (predicted == labels).sum().item()
            # 예측된 클래스와 실제 레이블 labels를 비교하여 올바르게 예측된 샘플 수를 누적
            # (predicted == labels) : 각 위치에서 예측이 맞으면 True, 틀리면 False를 가지는 텐서를 생성
            # sum().item() : True의 개수를 합산하여 올바르게 예측된 전체 수를 얻음

            total_samples += labels.size(0)
            # 테스트된 전체 샘플 수를 업데이트
            # labels.size(0) : 현재 미니배치의 샘플 수

    # 전체 테스트 데이터에 대한 정확도를 계산
    accuracy = correct_predictions / total_samples * 100

    return accuracy
```

✓ 0.0s

# 훈련 및 테스트

```python
# 훈련 및 테스트
num_epochs = 80

for epoch in range(num_epochs):
    train_loss = train(gmnet_2, aug_dataloader_2, criterion, optimizer, device)
    test_accuracy = test(gmnet_2, test_dataloader_2, criterion, device)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')

# 최종 모델의 정확도 출력
final_accuracy = test(gmnet_2, test_dataloader_2, criterion, device)
print(f'Final Test Accuracy: {final_accuracy:.2f}%')
```

# 결과

이미지 crop하지 않고 수행 결과

```
correct = 0
total = 0

# 학습 중이 아니므로, 출력에 대한 변화도를 계산할 필요가 없음
with torch.no_grad():
    for data in test_dataloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)

        # 신경망에 이미지를 통과시켜 출력을 계산
        outputs = gmnet(images)

        # 가장 높은 값(energy)를 갖는 분류(class)를 정답으로 선택
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

```
...   Accuracy of the network on the 10000 test images: 87 %
```

VISION SYSTEM LABORATORY

# 결과

```python
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
# 각 분류(class)에 대한 예측값 계산을 위해 준비
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# 변화도는 여전히 필요하지 않음
with torch.no_grad():
    for data in test_dataloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)

        outputs = gmnet(images)
        _, predictions = torch.max(outputs, 1)
        # 각 분류별로 올바른 예측 수를 모음
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# 각 분류별 정확도(accuracy)를 출력
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

✓ 15.2s

```
Accuracy for class: plane is 88.6 %
Accuracy for class: car   is 94.6 %
Accuracy for class: bird  is 78.6 %
Accuracy for class: cat   is 74.3 %
Accuracy for class: deer  is 85.6 %
Accuracy for class: dog   is 81.8 %
Accuracy for class: frog  is 91.7 %
Accuracy for class: horse is 91.9 %
Accuracy for class: ship  is 91.6 %
Accuracy for class: truck is 91.7 %
```

VISION SYSTEM LABORATORY