

Experiments on Communication Networks_ Week5

2022-2학기



Overview of Experiments

WEEK1 – Python Visualization

WEEK1 – LABVIEW tutorial

WEEK1 – Channel sensor tutorial

WEEK2 – Active sensing

WEEK2 – Radar generator tutorial

WEEK2 – Making datasets

WEEK3 – Data labeling

WEEK3 – Design CNN model

Week 3 -Contents

WEEK3 – Data labeling

WEEK3 – Design CNN model

3주차 실습 개요

- ❖ 3주차 실습은 2주차에서 얻은 데이터를 라벨링하고 CNN 모델에 넣어 학습합니다.
 - 1~2주차
 - 1. USRP를 통해 Tx / Rx 예제 제작 및 IQ data로 저장
 - 3주차
 - Visualization 데이터를 통해 데이터 라벨링.
 - 라벨링된 데이터를 CNN model에 넣어 학습.
- 2주차 실습에서 충분한 데이터를 수집하지 못했을 경우 추가적으로 데이터 수집 작업이 필요합니다.

Yonsei CBRS testbed

❖ Radar signal generator

- 1차 사용자. 연방, 위성 시스템 신호를 송출

❖ CBSD LTE signal

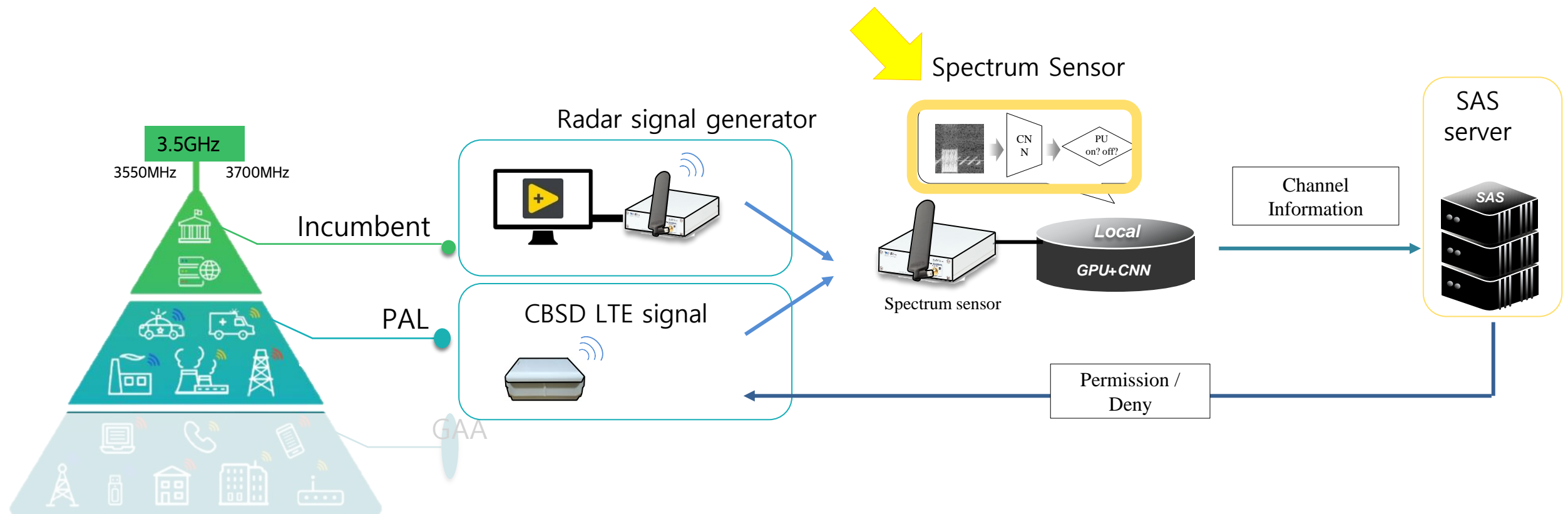
- 2차 사용자. LTE 신호를 송신

❖ Spectrum Sensor

- 3.5GHz 대역에 보내지고 있는 신호를 수신하여 1,2차 사용자의 사용 여부를 감지,
- SAS서버에 이러한 정보를 전송

❖ SAS server

- 2차 사용자에게 주파수 사용 (신호 송신) 권한을 실시간으로 부여
- Ex) 1차 사용자와 동시에 사용할 경우 주파수 사용 금지 권고



Data Labeling

❖ 습득한 데이터를 라벨링하는 작업을 진행합니다.

■ 각각의 라벨은 다음과 같은 의미를 지니고 있습니다.

— ex:

1	1	0	1
---	---	---	---

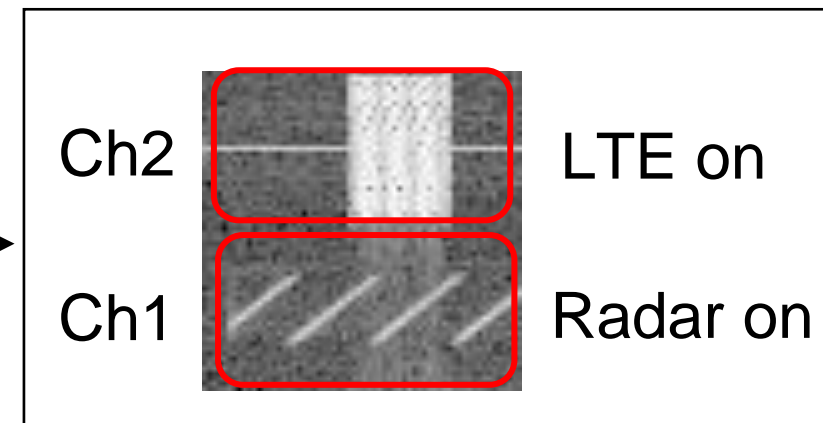
- 0000 (0) : All off
- 0001 (1) : Ch2 - Radar on
- 0010 (2) : Ch2 - LTE on
- 0011 (3) : Ch2 - LTE, Radar on
- 0100 (4) : Ch1 - Radar on
- 0101 (5) : Ch1 - Radar on / Ch2 - Radar on
- 0110 (6) : Ch1 - Radar on / Ch2 - LTE on
- 0111 (7) : Ch1 - Radar on / Ch2 - LTE, Radar on
- 1000 (8) : Ch1 - LTE on
- 1001 (9) : Ch1 - LTE on / Ch2 - Radar on
- 1010 (10) : Ch1 - LTE on / Ch2 - LTE on
- 1011 (11) : Ch1 - LTE on / Ch2 - LTE, Radar on
- 1100 (12) : Ch1 - LTE, Radar on
- 1101 (13) : Ch1 - LTE, Radar on / Ch2 - Radar on
- 1110 (14) : Ch1 - LTE, Radar on / Ch2 - LTE on
- 1111 (15) : Ch1 - LTE, Radar on / Ch2 - LTE, Radar on

Ch2 - Radar on / off

Ch2 - LTE on / off

Ch1 - Radar on / off

Ch1 - LTE on / off

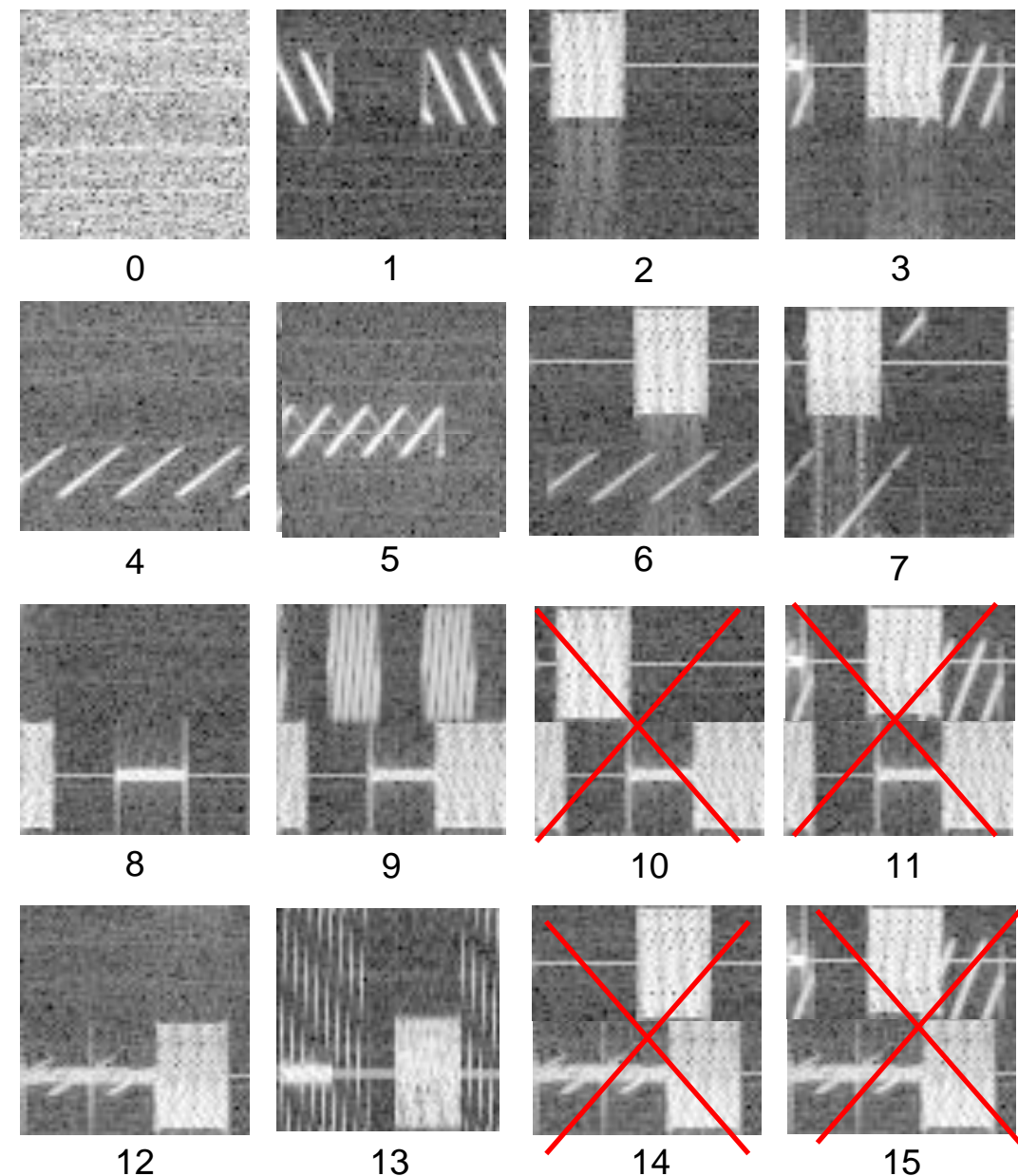
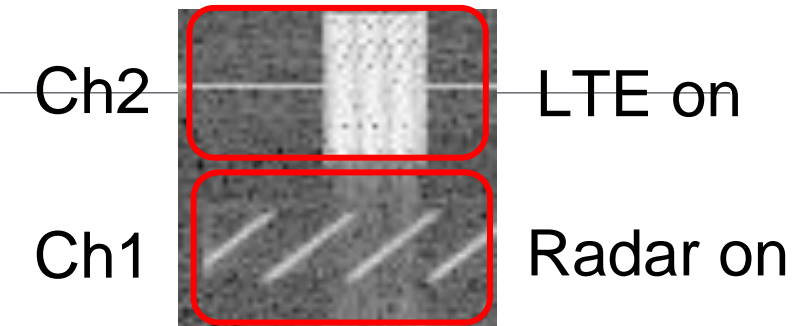


Data Labeling

❖ 습득한 데이터를 라벨링하는 작업을 진행합니다.

■ 각각의 라벨은 다음과 같은 의미를 지니고 있습니다.

- 0000 (0) : All off
- 0001 (1) : Ch2 - Radar on
- 0010 (2) : Ch2 - LTE on
- 0011 (3) : Ch2 - LTE, Radar on
- 0100 (4) : Ch1 - Radar on
- 0101 (5) : Ch1 - Radar on / Ch2 - Radar on
- 0110 (6) : Ch1 - Radar on / Ch2 - LTE on
- 0111 (7) : Ch1 - Radar on / Ch2 - LTE, Radar on
- 1000 (8) : Ch1 - LTE on
- 1001 (9) : Ch1 - LTE on / Ch2 - Radar on
- 1010 (10) : Ch1 - LTE on / Ch2 - LTE on
- 1011 (11) : Ch1 - LTE on / Ch2 - LTE, Radar on
- 1100 (12) : Ch1 - LTE, Radar on
- 1101 (13) : Ch1 - LTE, Radar on / Ch2 - Radar on
- 1110 (14) : Ch1 - LTE, Radar on / Ch2 - LTE on
- 1111 (15) : Ch1 - LTE, Radar on / Ch2 - LTE, Radar on



Data Labeling

- ❖ 이중 우리는 12개 라벨에 대한 데이터만 사용할 것입니다
 - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13
- ❖ 그리고 이전 수업에서 직접 수집한 데이터는 Radar on 입니다
 - 따라서 수집할 수 있는 데이터의 라벨은
 - 1, 4, 5

실습: Data Labeling

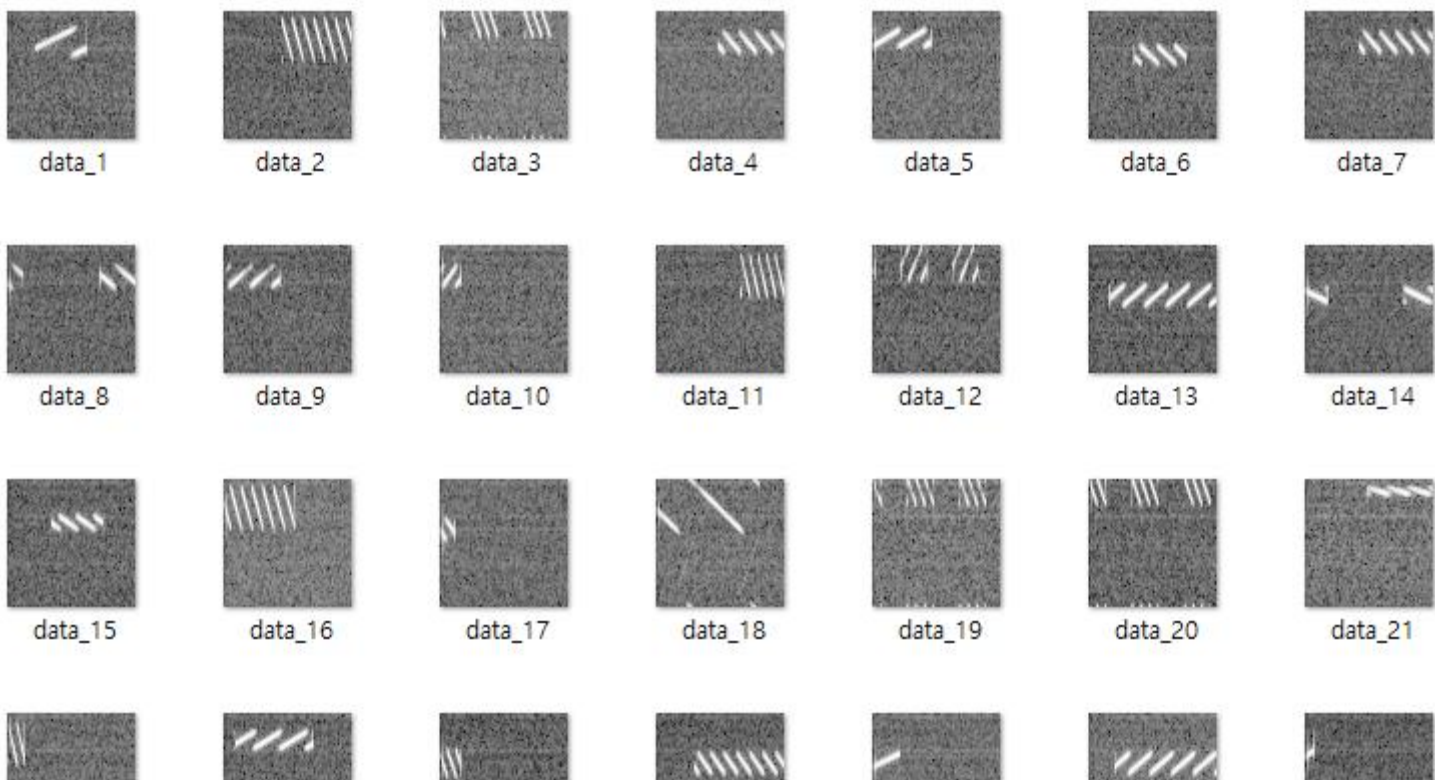
- ❖ 습득한 데이터를 라벨링하는 작업을 진행합니다.
 - Spectrogram을 통해 Visualized된 데이터를 수기로 분류합니다
 - 각 라벨 별 폴더를 만들고 각각의 라벨에 해당되는 데이터를 해당 라벨 폴더에 넣습니다.

ProjectPytorch > IQclassifier > 3rd_week > labeled_data

이름	수정한 날짜	유형
label0	2021-09-04 오전 1:30	파일 폴더
label1	2021-09-04 오전 1:30	파일 폴더
label2	2021-09-04 오전 1:31	파일 폴더
label3	2021-09-04 오전 1:31	파일 폴더
label4	2021-09-04 오전 1:31	파일 폴더
label5	2021-09-04 오전 1:31	파일 폴더
label6	2021-09-04 오전 1:31	파일 폴더
label7	2021-09-04 오전 1:31	파일 폴더
label8	2021-09-04 오전 1:31	파일 폴더
label9	2021-09-04 오전 1:31	파일 폴더
label12	2021-09-04 오전 1:30	파일 폴더
label13	2021-09-04 오전 1:30	파일 폴더

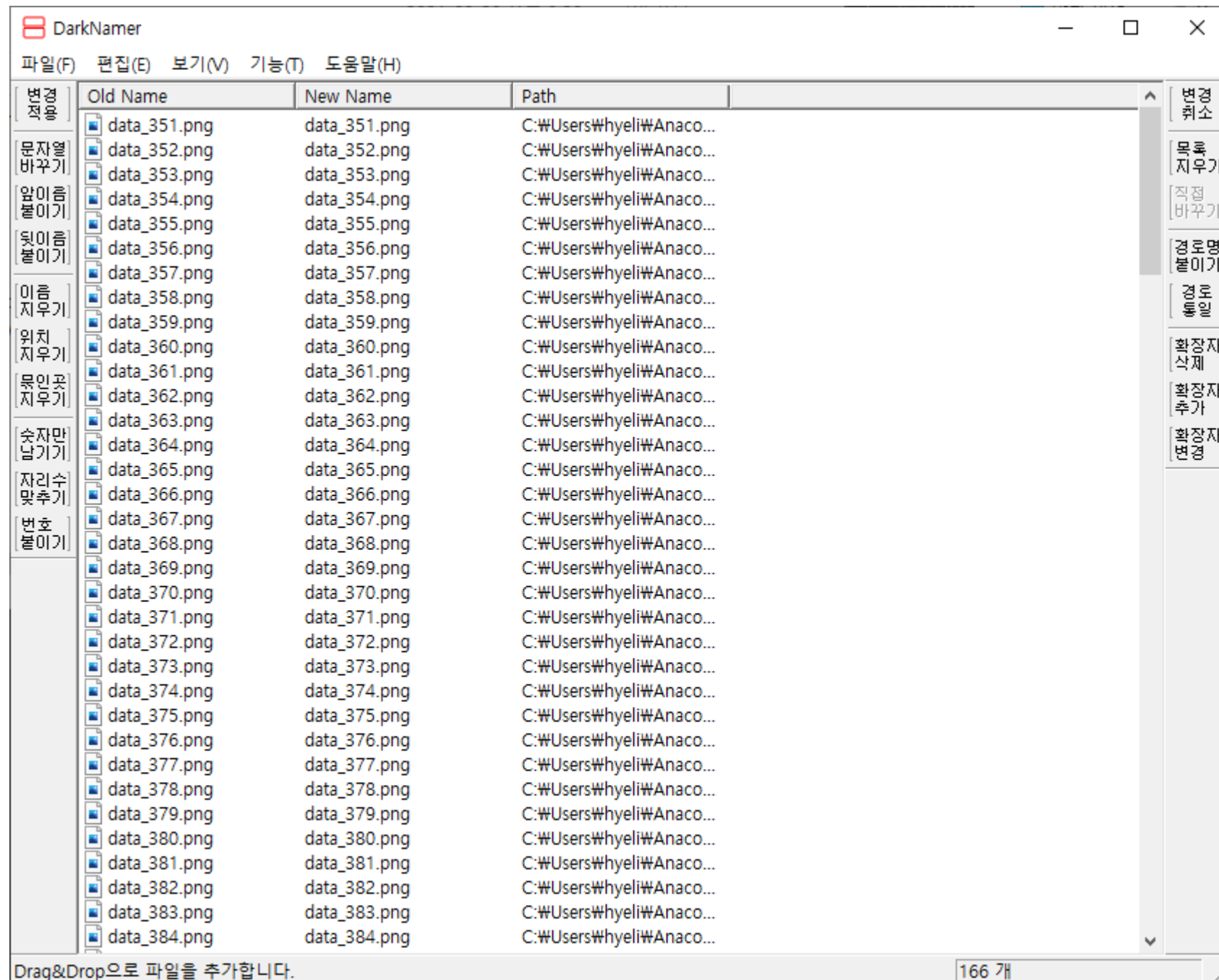
ProjectPytorch > IQclassifier > 3rd_week > labeled_data > label1

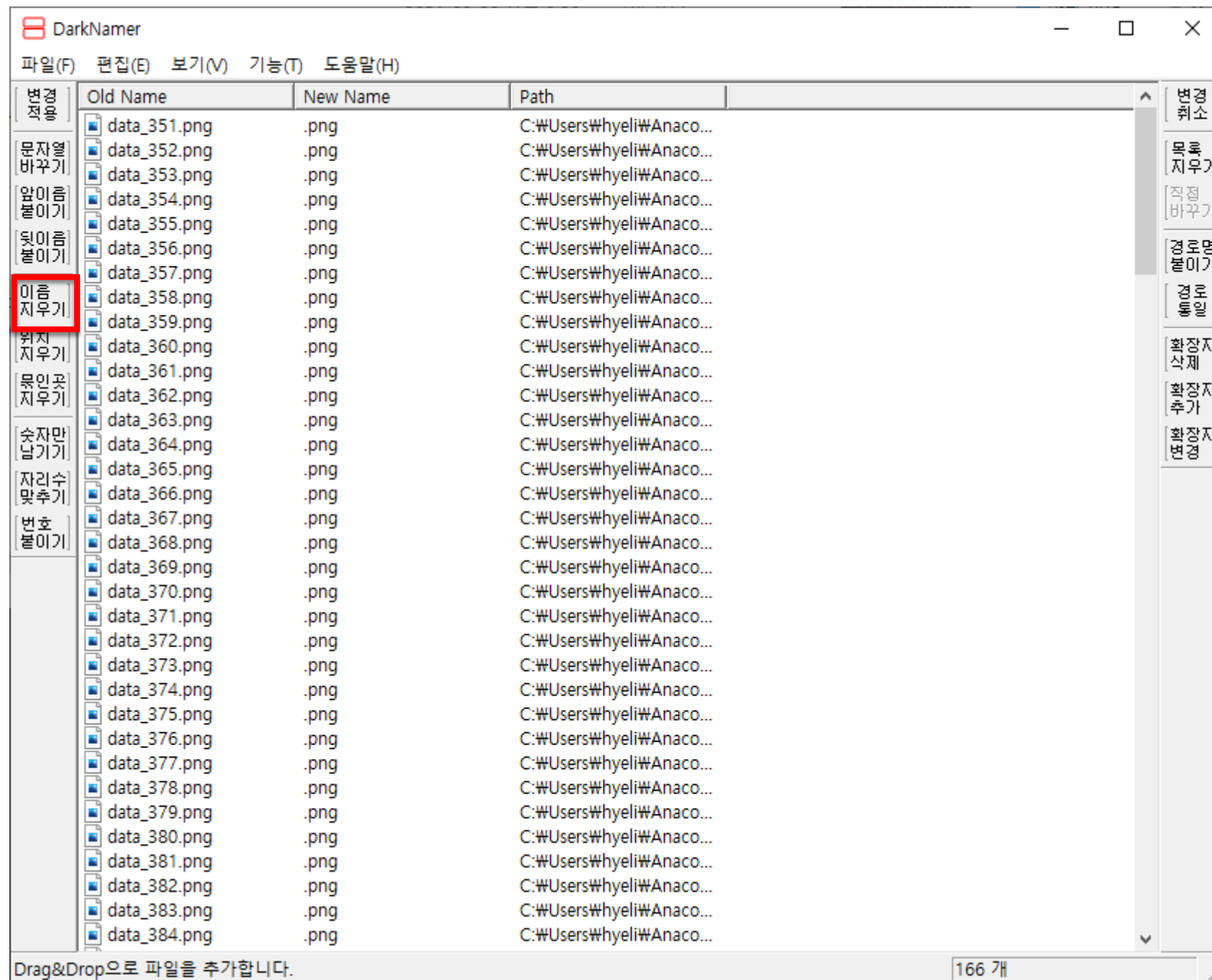
label1 검색

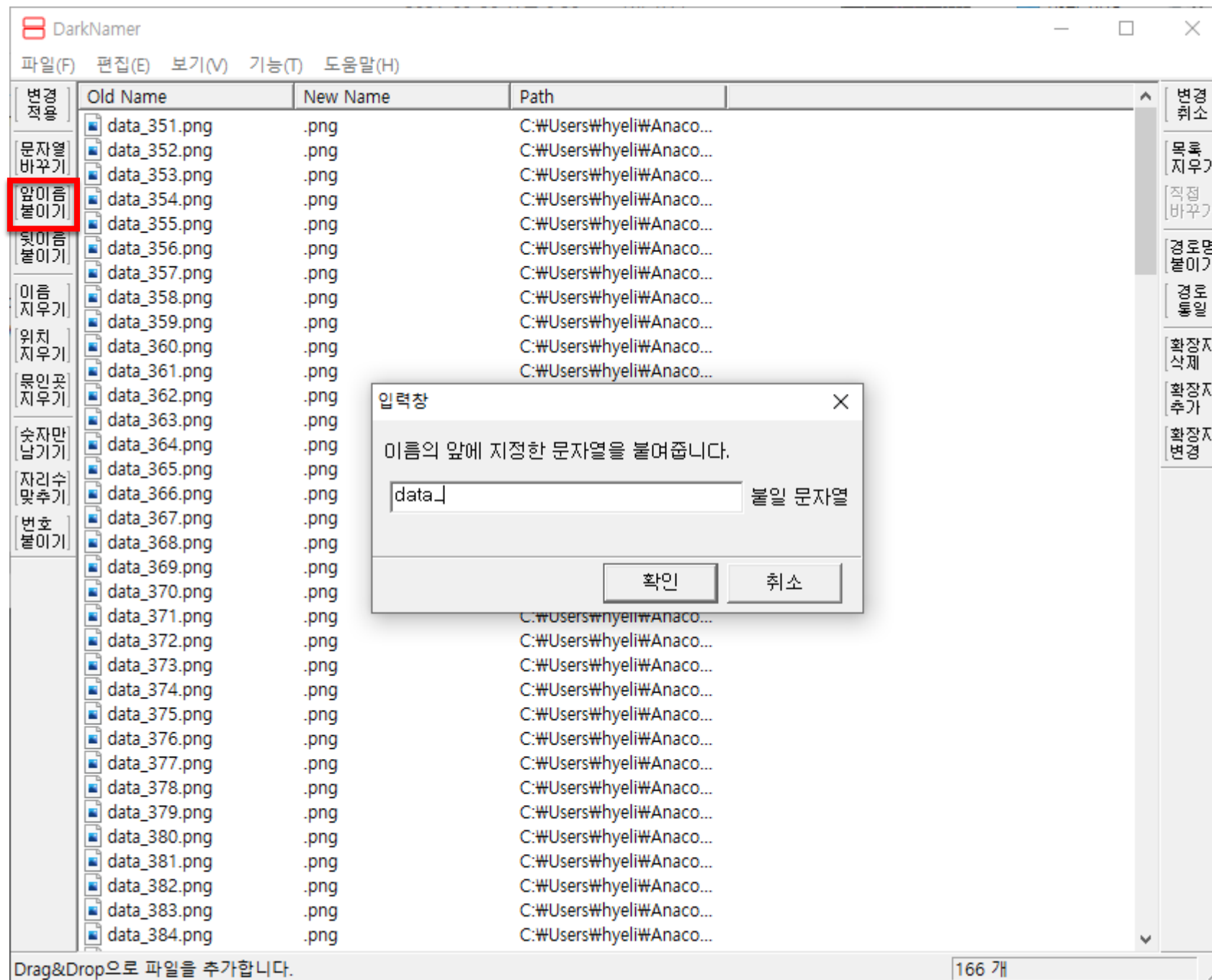


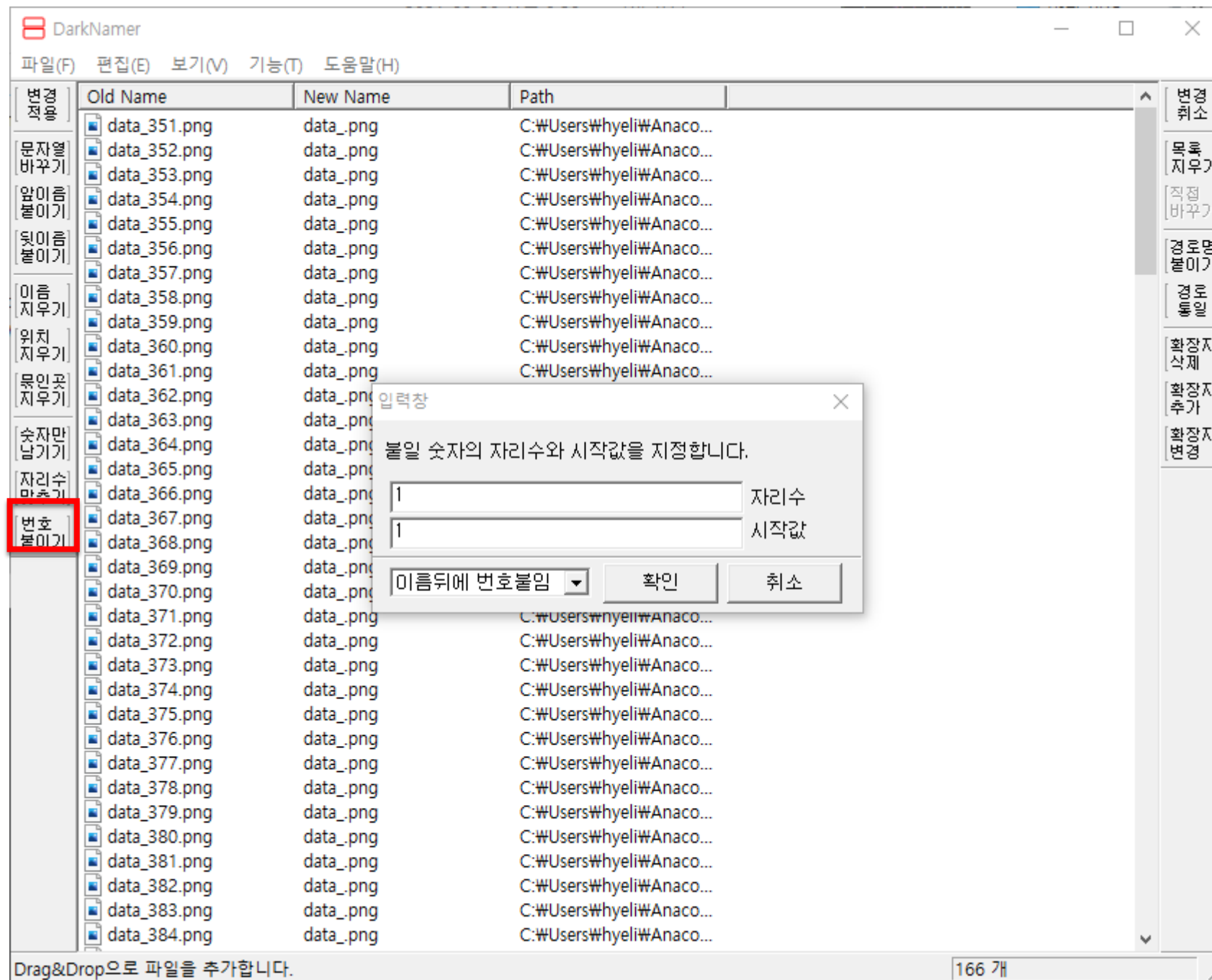
CNN model

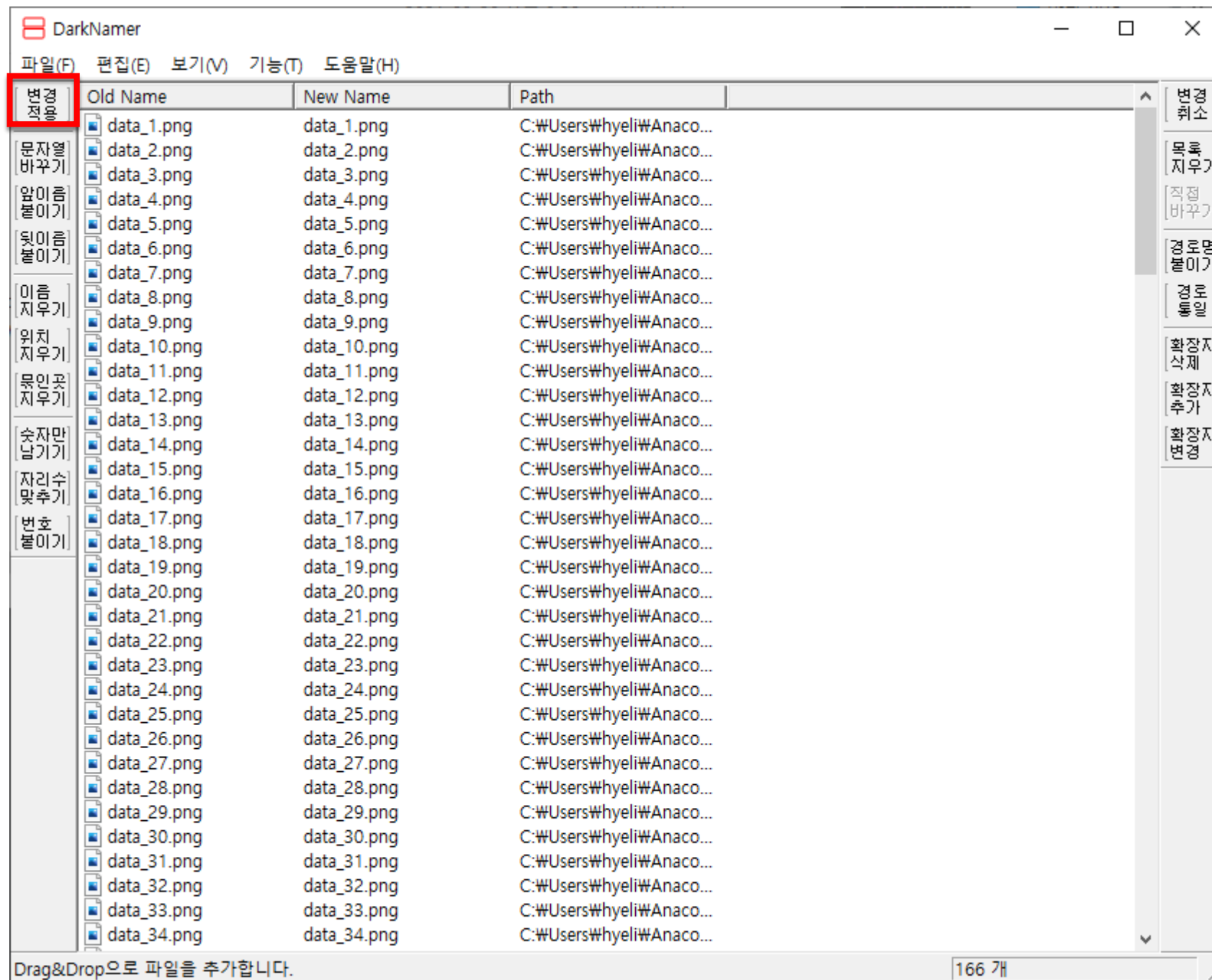
- ❖ 이미지파일들의 이름이 겹칠경우, 겹치는 데이터들 이름 일괄 변형 필요
- ❖ DarkNamer 프로그램에 해당 데이터 드래그





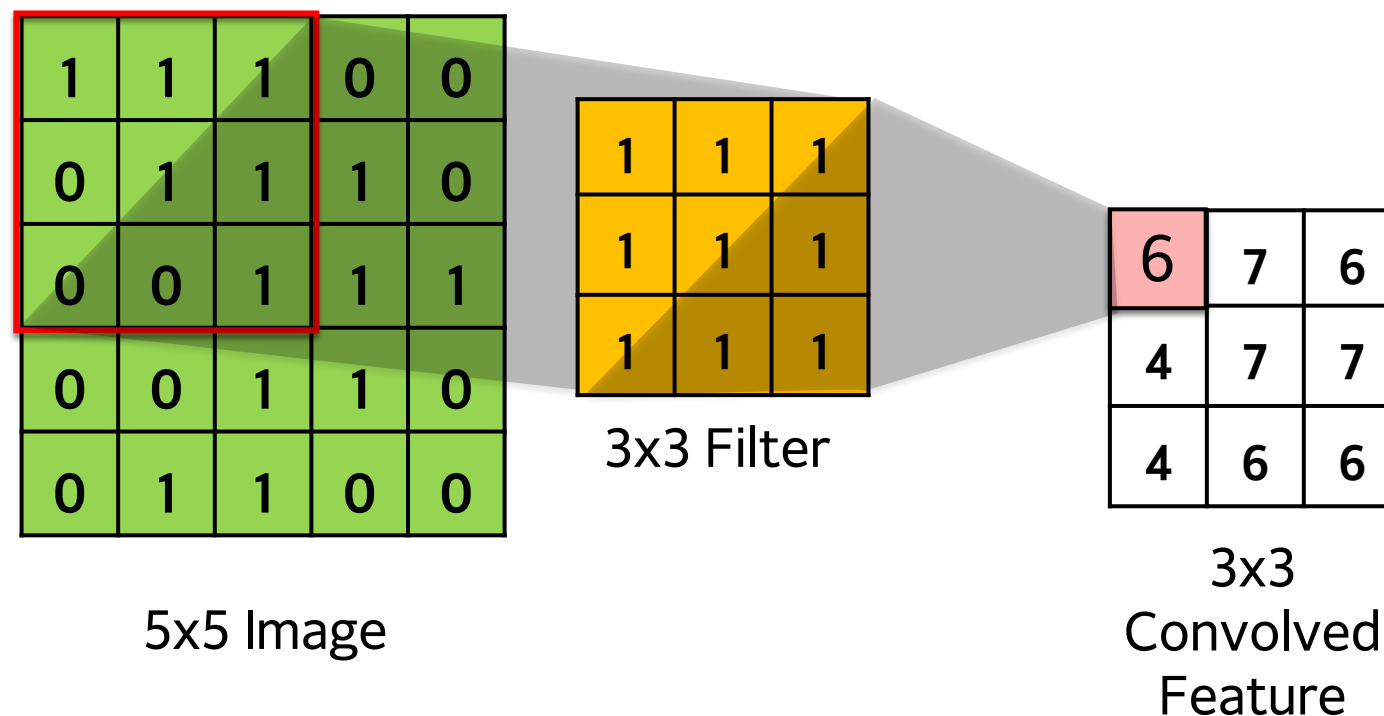






CNN model

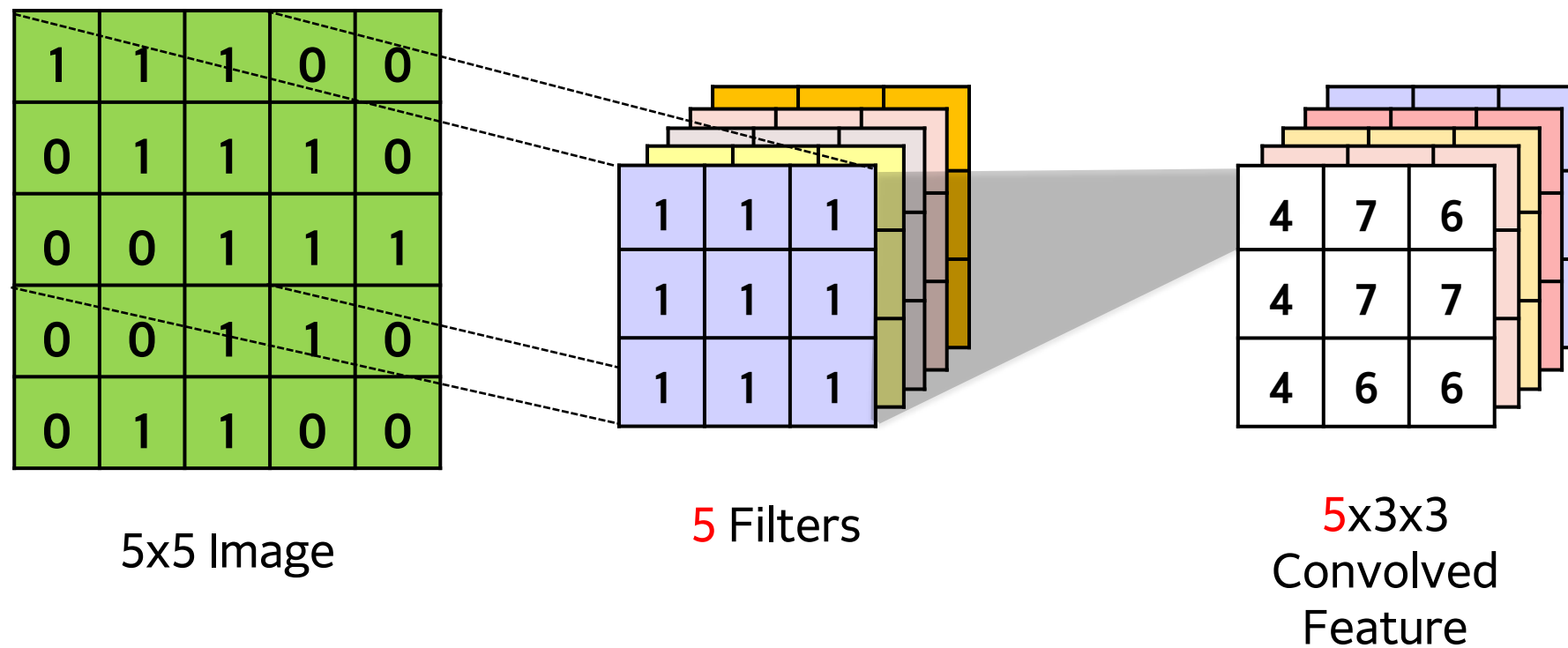
- ❖ CNN model을 통해 데이터 분류하기.
 - CNN 은 Convolutional Neural Networks의 약자로 주로 이미지 데이터를 처리할 때 사용합니다.
 - Input Image를 주어진 크기의 Filter를 통과시킵니다.
 - Filter를 통과한 데이터는 각각 Inner product되어 계산됩니다.
 - Filter의 크기는 일반적으로 3x3, 5x5, 7x7를 사용합니다.



CNN model

■ Channel

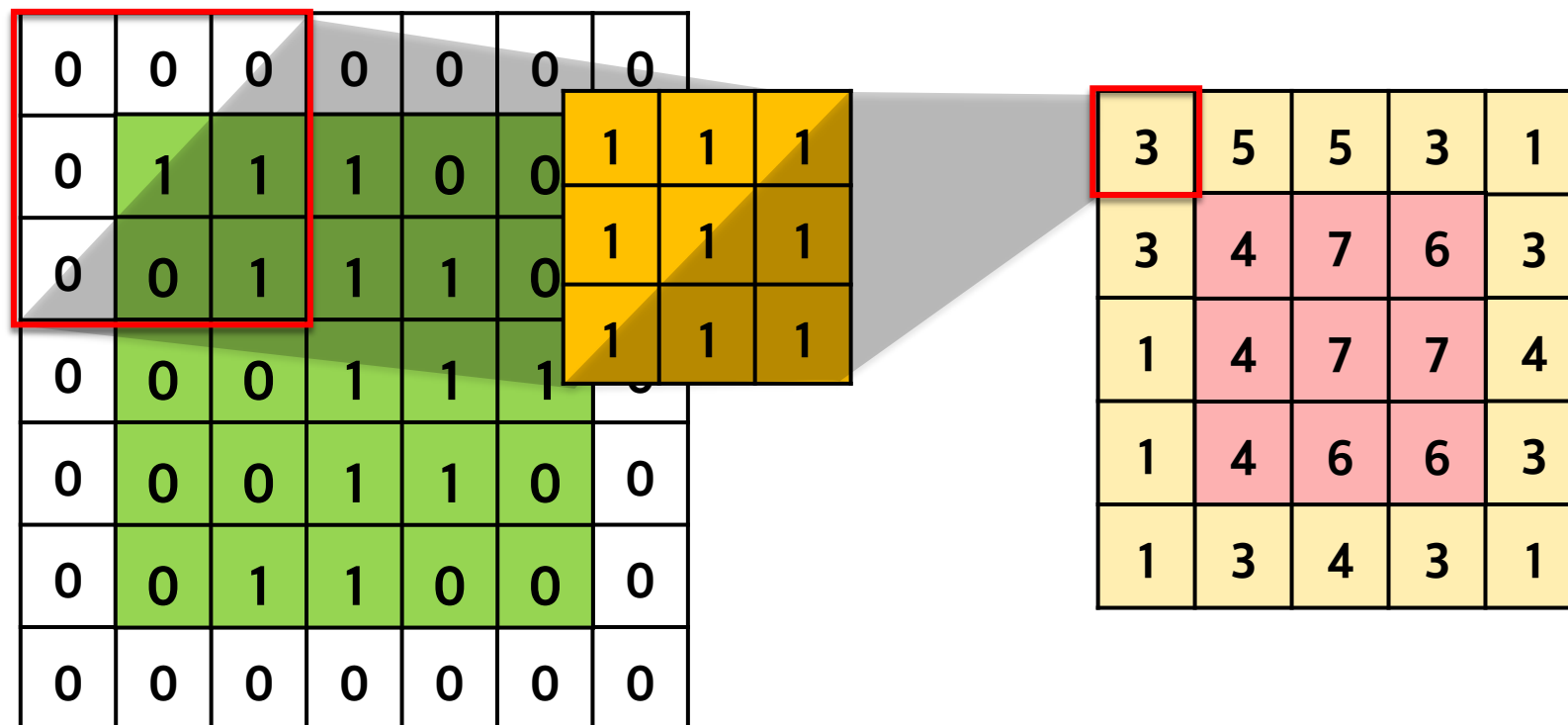
- Filter의 개수는 여러 개일 수 있습니다. Output의 크기는 Filter의 개수에 따라 달라집니다.



CNN model

■ Zero padding

- Filter 처리를 하면 Output의 크기가 Input의 크기와 다른 것을 방지하기 위해 이미지의 가장자리에 0으로 처리된 값을 추가해줍니다.



CNN model

■ Stride

- Filter를 얼마만큼 움직여주는지 알려줍니다.
- Stride값과 Padding 여부에 따라 Output의 크기가 바뀝니다.

- Output 높이 = $\frac{H+2P-FH}{S} + 1$
- Output 폭 = $\frac{W+2P-FW}{S} + 1$

H : Input Image의 높이
W : Input Image의 폭
FH : Filter의 높이
FW : Filter의 폭
P : Padding 크기
S : Stride의 크기

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Stride 1

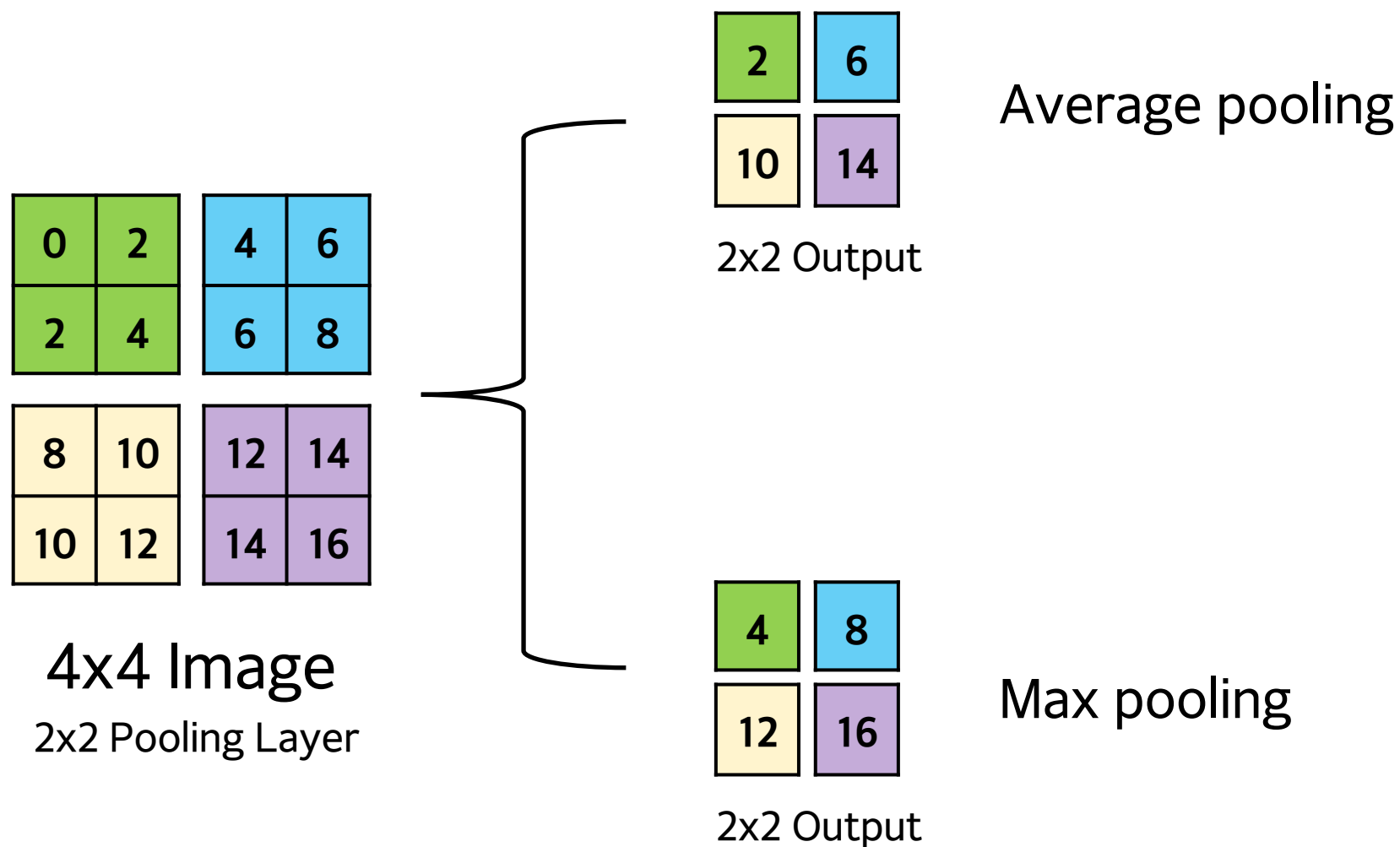
0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Stride 4

CNN model

■ Max pooling & Average pooling

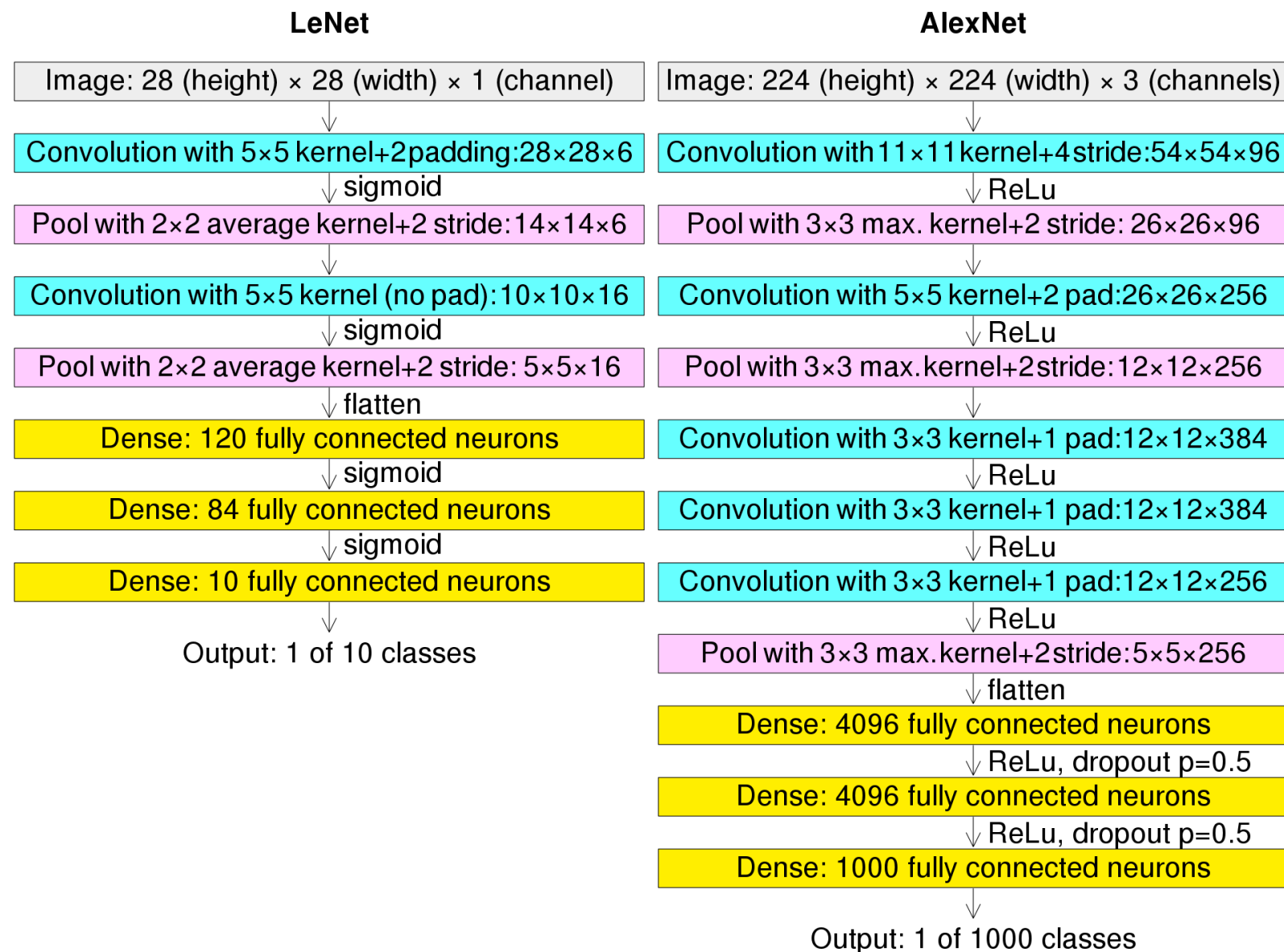
- Pooling은 출력 데이터의 크기를 줄이면서 Feature를 강조하기 위해 쓰입니다.
- 일반적으로 Pooling Layer를 거치면 크기가 작아집니다.



CNN model




■ 설계 예시

- CNN 모델은 Convolution Layer, Pooling Layer, Fully Connected Layer를 사용하여 설계됩니다. 다음은 LeNet와 AlexNet에 대한 예시 설계입니다.



CNN model

- 실습:
- week5/mainCode.py

 labeled_data	2021-09-04 오후 2:22	파일 폴더	
 models	2021-09-28 오후 2:38	파일 폴더	
 mainCode	2021-09-04 오후 2:56	PY 파일	10KB

CNN model

❖ Dataset

■ 경로 설정

```
# Load Data
print('==> Dataset selecting..')
data_root = "./labeled_data" # Set your data root
MyDataSet = CustomDataSet(data_root)
```

■ Train, validation, test 로 데이터셋 나눔

```
# Data Set Split
train_size = int(0.7 * len(MyDataSet)) # Training Size
valid_size = int(0.2 * len(MyDataSet)) # Validation Size
test_size = len(MyDataSet) - train_size - valid_size # Test Size
train_dataset, valid_dataset, test_dataset = torch.utils.data.random_split(MyDataSet, [train_size, valid_size, test_size])
```

CNN model

❖ GPU

- GPU가 없을 시에 주석처리

```
# GPU Allocating
print('==> CPU/GPU allocating..')
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Device : ', device)
print('Available devices : ', torch.cuda.device_count())
# print('Selecing GPU : ', torch.cuda.get_device_name(device))
```

CNN model

❖ Training

- epoch, batchsize 설정
- Epoch: 모든 데이터셋을 몇번 반복해서 학습을 할 것인가
 - 대부분 많을수록 시간이 오래 걸리나 최종 성능이 좋아짐
- Batchsize: 한번 training 할 때 몇 개의 데이터를 입력할 것인가
 - 대부분 조금 클수록 다양한 데이터에 대해 한번에 학습하여 다양성을 가지게 되어 성능이 좋아지나, 너무 커져도 성능이 나빠지므로 적당한 값이 좋음

```
# Model
print('==> Building model..')
nb_epochs = 30 # Set an epoch 1
torch.manual_seed(10) #Set your seed number
BatchSize = 50 # Set a batch size
```


CNN model

❖ Training

```
for epoch in range(1, nb_epochs+1):  
    #Train  
    start_epoch = datetime.now()  
    model.train()  
    train_loss = 0  
    valid_total = 0  
    valid_correct = 0  
    #label_total = list(0. for i in range(16))  
    #label_correct = list(0. for i in range(16))  
    for batch_idx, (datum, targets) in enumerate(train_dataloader):  
        start = datetime.now()  
        datum, targets = datum.to(device), targets.long().to(device)  
        img = datum.reshape([-1, 1, 64, 64]).float()  
        bsz = targets.shape[0]  
  
        # Computing loss  
        out = model(img)  
        loss = criterion(out, targets)  
  
        # Update model  
        optimizer.zero_grad()  
        loss.backward()  
        optimizer.step()  
        scheduler.step()  
  
        train_loss += loss.item()
```

Epoch iteration

Batch iteration

모델에 이미지를 적용하여
나온 출력과 label 이 얼마나 다른지에 대한 loss 계산

loss 를 줄여주기 위하여 모델 업데이트

CNN model

❖ Training 결과

■ 각 epoch 에 대하여 loss 와 accuracy 출력

- Accuracy: 입력한 데이터 중 몇 퍼센트에 대하여 모델이 label을 맞췄는지

```
Epoch: 1 | Train loss: 14.875690, Valid loss: 2.399220, Valid Acc: 93 / 424 | Time spent: 0:00:02.471457
Epoch: 2 | Train loss: 1.778478, Valid loss: 0.962740, Valid Acc: 258 / 424 | Time spent: 0:00:00.121685
Epoch: 3 | Train loss: 0.693891, Valid loss: 0.707973, Valid Acc: 307 / 424 | Time spent: 0:00:00.113677
Epoch: 4 | Train loss: 0.445977, Valid loss: 0.624138, Valid Acc: 324 / 424 | Time spent: 0:00:00.114692
Epoch: 5 | Train loss: 0.320100, Valid loss: 0.564215, Valid Acc: 330 / 424 | Time spent: 0:00:00.114695
Epoch: 6 | Train loss: 0.227674, Valid loss: 0.601691, Valid Acc: 336 / 424 | Time spent: 0:00:00.110672
Epoch: 7 | Train loss: 0.200266, Valid loss: 0.537230, Valid Acc: 341 / 424 | Time spent: 0:00:00.111702
Epoch: 8 | Train loss: 0.122314, Valid loss: 0.498873, Valid Acc: 356 / 424 | Time spent: 0:00:00.117684
Epoch: 9 | Train loss: 0.098883, Valid loss: 0.524782, Valid Acc: 355 / 424 | Time spent: 0:00:00.111702
Epoch: 10 | Train loss: 0.078438, Valid loss: 0.508019, Valid Acc: 354 / 424 | Time spent: 0:00:00.117685
Epoch: 11 | Train loss: 0.073264, Valid loss: 0.518028, Valid Acc: 364 / 424 | Time spent: 0:00:00.113694
Epoch: 12 | Train loss: 0.029191, Valid loss: 0.579043, Valid Acc: 354 / 424 | Time spent: 0:00:00.106715
Epoch: 13 | Train loss: 0.023718, Valid loss: 0.507032, Valid Acc: 363 / 424 | Time spent: 0:00:00.118682
Epoch: 14 | Train loss: 0.016634, Valid loss: 0.545996, Valid Acc: 365 / 424 | Time spent: 0:00:00.109706
Epoch: 15 | Train loss: 0.016773, Valid loss: 0.532611, Valid Acc: 360 / 424 | Time spent: 0:00:00.123668
Epoch: 16 | Train loss: 0.008779, Valid loss: 0.598558, Valid Acc: 366 / 424 | Time spent: 0:00:00.121674
Epoch: 17 | Train loss: 0.008393, Valid loss: 0.559086, Valid Acc: 364 / 424 | Time spent: 0:00:00.108709
Epoch: 18 | Train loss: 0.007278, Valid loss: 0.614458, Valid Acc: 355 / 424 | Time spent: 0:00:00.113695
Epoch: 19 | Train loss: 0.006512, Valid loss: 0.562911, Valid Acc: 361 / 424 | Time spent: 0:00:00.122671
Epoch: 20 | Train loss: 0.003090, Valid loss: 0.582635, Valid Acc: 364 / 424 | Time spent: 0:00:00.129653
Epoch: 21 | Train loss: 0.002370, Valid loss: 0.601976, Valid Acc: 367 / 424 | Time spent: 0:00:00.150596
Epoch: 22 | Train loss: 0.001949, Valid loss: 0.629529, Valid Acc: 363 / 424 | Time spent: 0:00:00.122673
Epoch: 23 | Train loss: 0.001540, Valid loss: 0.585319, Valid Acc: 364 / 424 | Time spent: 0:00:00.122672
Epoch: 24 | Train loss: 0.001382, Valid loss: 0.587620, Valid Acc: 362 / 424 | Time spent: 0:00:00.126660
Epoch: 25 | Train loss: 0.001236, Valid loss: 0.609471, Valid Acc: 362 / 424 | Time spent: 0:00:00.136634
Epoch: 26 | Train loss: 0.001134, Valid loss: 0.595928, Valid Acc: 360 / 424 | Time spent: 0:00:00.120676
Epoch: 27 | Train loss: 0.001022, Valid loss: 0.617629, Valid Acc: 363 / 424 | Time spent: 0:00:00.111703
Epoch: 28 | Train loss: 0.000944, Valid loss: 0.648720, Valid Acc: 363 / 424 | Time spent: 0:00:00.118682
Epoch: 29 | Train loss: 0.000872, Valid loss: 0.663525, Valid Acc: 362 / 424 | Time spent: 0:00:00.119681
Epoch: 30 | Train loss: 0.000804, Valid loss: 0.649500, Valid Acc: 363 / 424 | Time spent: 0:00:00.110686
```

CNN model

- 실습:
- week5/models 폴더에서 CustomModel 파일을 수정하여 사용

labeled_data	2021-09-04 오후 2:22	파일 폴더	
models	2021-09-28 오후 2:38	파일 폴더	
mainCode	2021-09-04 오후 2:56	PY 파일	10KB

pycache	2021-09-04 오후 2:22	파일 폴더	
CustomModelExample	2021-09-28 오후 2:38	PY 파일	

CNN model

■ 실습: 나만의 CNN 모델 만들기

— 예시: 4-Layer 적층 구조

```
class MyClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size = 5, stride = 1, padding = 2)
        self.layer2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 3, stride = 2, padding = 1)

        self.linear1 = nn.Linear(2048, 1024)
        self.linear2 = nn.Linear(1024, 16)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.layer1(x)), (2,2))
        x = F.max_pool2d(F.relu(self.layer2(x)), (2,2))
        x = torch.flatten(x,1)
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

- 시작 채널의 크기 1 고정 (빨간색 네모)
- __init__에서 적층시켜준 것만큼 forward에서도 동일하게 적층해야함.
- Convolution Layer (CL)의 경우 이전 Layer의 out_channels의 크기가 다음 layer의 in_channels의 크기와 동일하도록 설정 (초록색 네모)
 - Kernel_size, stride, padding의 크기는 재량껏 설정 (13 page 참고)
 - 위 파라미터에 따라 출력되는 데이터 크기가 달라지므로 주의할 것
- Forward에서 F.relu 및 max_pool2d 설정
 - max pooling의 사이즈 변경 가능 (노랑색 네모)

CNN model

- 실습: 나만의 CNN 모델 만들기
 - 예시: 4-Layer 적층 구조

```
class MyClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size = 5, stride = 1, padding = 2)
        self.layer2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 3, stride = 2, padding = 1)

        self.linear1 = nn.Linear(2048, 1024)
        self.linear2 = nn.Linear(1024, 16)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.layer1(x)), (2,2))
        x = F.max_pool2d(F.relu(self.layer2(x)), (2,2))
        x = torch.flatten(x,1)
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

- Fully connected Layer (Linear layer)의 경우 Input (빨간색 네모)과 Output (초록색 네모)의 크기를 조정
 - 본인의 parameter와 적층 구조에 따라 첫 Linear Layer의 input 크기가 달라지므로 주의할 것
- 마지막 Linear layer의 output은 16으로 고정 (Label의 갯수)
- forward에서 마지막 Linear layer는 relu를 씌우지 않음.

CNN model

- 실습: 나만의 CNN 모델 만들기
 - 검증: 4-Layer 적층 구조

```
def forward(self, x):  
    print(x.size()) # Check data size  
    x = F.max_pool2d(F.relu(self.layer1(x)), (2,2))  
    print(x.size())  
    x = F.max_pool2d(F.relu(self.layer2(x)), (2,2))  
    print(x.size())  
    x = torch.flatten(x,1)  
    print(x.size())  
    x = F.relu(self.linear1(x))  
    print(x.size())  
    x = self.linear2(x)  
    print(x.size())  
    return x  
net = MyClassifier()  
print(net)
```

```
return torch.nn.max_pool2d(input, kernel_size, stride, padding, ceil_mode)  
torch.Size([50, 16, 32, 32])  
torch.Size([50, 32, 8, 8])  
torch.Size([50, 2048])  
torch.Size([50, 1024])  
torch.Size([50, 16])  
torch.Size([50, 16])  
D:\C:\Users\Kyuwon\Dropbox\Project\Python
```

- forward에서 print(x.size())를 통해 Layer 통과 전 후의 데이터 크기를 체크하면서 파라미터 조절
- 본인의 Layer구조는 net = Myclassifer()로 선언 후 프린트하여 출력 가능

```
MyClassifier(  
  (layer1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))  
  (layer2): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))  
  (linear1): Linear(in_features=2048, out_features=1024, bias=True)  
  (linear2): Linear(in_features=1024, out_features=16, bias=True)  
)
```

CNN model

- 실습: 나만의 CNN 모델 만들기

- 검증: 4-Layer 적층 구조

- 최종적으로 임의의 tensor를 생성 후 네트워크 통과 여부 검증 후 실제 데이터에 적용!

```
td = torch.rand(50,1,64,64)
out = net(td)
print(out.size())
```

CNN model

■ 실습: 나만의 데이터셋 학습시키기

- 데이터셋 폴더 경로 설정 중요

```
import cv2
import re
import imageio
import copy

from models.CustomModelExample import MyClassifier

def main():

    # GPU Allocating
    print('==> CPU/GPU allocating..')
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print('Device : ', device)
    print('Available devices : ', torch.cuda.device_count())
    print('Selecting GPU : ', torch.cuda.get_device_name(device))

    # Load Data
    print('==> Dataset selecting..')
    data_root = "./labeled_data_" # Set your data root
    MyDataSet = CustomDataSet(data_root)

    # Data Set Split
    train_size = int(0.7 * len(MyDataSet)) # Training Size
    valid_size = int(0.2 * len(MyDataSet)) # Validation Size
    test_size = len(MyDataSet) - train_size - valid_size # Test Size
```


결과 리포트

■ 1. 주어진 데이터로 training

- 실행 결과 test accuracy 작성

■ 2. 저번주에 만든 데이터로 training

- ch1, ch2, both ch 폴더에 저장했던 이미지데이터들을 각 label 에 맞게 옮긴 후 (기존 데이터는 삭제)
- 실행 결과 test accuracy 작성
- 기존 데이터 파일 labeled_data 를 복제하여 해당 라벨만 수정하셔야 이후 문제들을 실험할 때 수월할 것입니다.

 labeled_data	2021-09-28 오후 3:30	파일 폴더
 labeled_data2	2021-09-04 오후 2:22	파일 폴더

- labeled_data: 주어진 데이터
- labeled_data2: 주어진 데이터에서 해당 라벨만 수정한 데이터셋

결과 리포트

■ 3. 모델 변경 후 주어진 데이터로 training

- 한 개 이상의 layer를 추가로 적층하여 5-layer 이상의 모델 구현하고
- 실행 결과 test accuracy 작성

■ Bonus:

- 주어진 모델 + 주어진 데이터셋
- 주어진 모델 + 만든 데이터셋
- 만든 모델 + 주어진 데이터셋
- 만든 모델 + 만든 데이터셋
- 위 4개에 대한 실행 결과 test accuracy 를 비교하시고, 차이가 존재한다면 그 이유에 대해 간단히 서술하십시오.

■ 제출양식:

- 보고서 : 형식 상관없음, 변경한 수치들 (np_epoch 등) 이 있다면 작성
- 만든 모델에 대한 week5/models/CustomModelExample.py 코드 제출

Pre Report

- ❖ SDN에 대해 조사해오기
 - SDN의 개념
 - SDN의 장점
 - SDN의 구성

- ❖ OpenFlow에 대해 조사해오기
 - OpenFlow의 개념
 - OpenFlow 기반 SDN의 동작

- ❖ Hub에 대해 조사해오기
 - Hub의 개념
 - Hub의 동작

- ❖ L2 Switch에 대해 조사해오기
 - L2 Switch 개념
 - L2 Switch 동작