Term Project 발표

홍유빈
손한솔
김시온

01

데이터 전처리

Here you could describe the topic of the section

02

CNN 모델

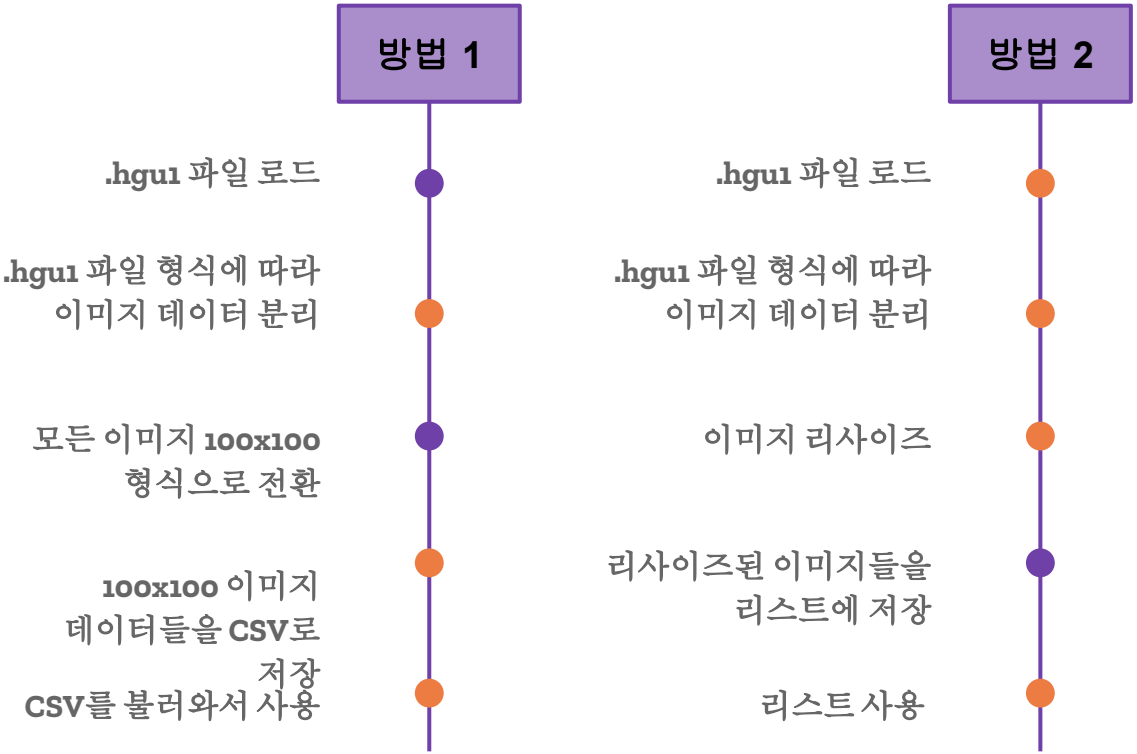
Here you could describe the topic of the section

03

결과

Here you could describe the topic of the section

데이터
전처리


방법 1
.hgu1 파일 로드
.hgu1 파일 형식에 따라
이미지 데이터 분리
모든 이미지 100x100
형식으로 전환
100x100 이미지
데이터들을 CSV로
저장
CSV를 불러와서 사용
방법 2
.hgu1 파일 로드
.hgu1 파일 형식에 따라
이미지 데이터 분리
이미지 리사이즈
리사이즈된 이미지들을
리스트에 저장
리스트 사용

데이터
전처리

```
int ReadHGU1(FILE *fp, Image *pImage)
{

    for(int i = 0; i < 100; i++)
        for(int j = 0; j < 100; j++)
            pImage->m_data[i][j] = 255;

    int ret = fread(pImage->m_code, 1, 2, fp);
    if(ret < 2)
        return FALSE;

    pImage->m_code[2] = 0;
    fread(&pImage->m_width, 1, 1, fp);
    fread(&pImage->m_height, 1, 1, fp);
    fread(&pImage->m_type, 1, 1, fp);
    fread(&pImage->m_reserved, 1, 1, fp);

    for(int y = 0; y < pImage->m_height; y++)
        fread(pImage->m_data[y], 1, pImage->m_width, fp);

    return TRUE;
}
```

1. **[100][100]** 데이터 어레이를 **255**로 채우기

2. 각 이미지의 데이터에 따라 값을 다시 변경 후 **CSV**로 저장

ex)
64x70 → 100x100
빈공간 **255**로 채우기

데이터 전처리

```
class Dataset(Dataset):
    """Korean Handwriting dataset"""

    def __init__(self, csv=None, root_dir=None):

        self._list_X = []
        self._list_Y = []
        self._root_dir = root_dir     #
        self._dir = os.path.join(self.root_dir)     #

        for root, dirs,files in os.walk(self._dir):
            for file in files:
                if file.endswith(".csv"):
                    self._csv = open(file, 'r')
        #self._csv = open(self._dir, 'r')

                    # initializae Dataset class
                    if self._csv:
                        #self._file = open(csv, 'r')     # gz File??
                        self._read_file = csv.reader(self._file)

                        self._Data_frame = pd.Dataframe(self._read_file)
                        self._Data_frame_X = self._Data_frame.iloc[1:89,72]
                        self._Data_frame_Y = self._Data_frame.iloc[0,:].str.splic('=').str[1]
                        self._dataset_X = self._Data_frame_X.astype(float)      # np.unit8?
                        self._dataset_Y = self._Data_frame_Y.astype(float)
                        self._numpy_X = self._dataset_X.to_numpy()
                        self._numpy_Y = self._dataset_Y.to_numpy()
                        self._list_X.append(torch.tensor(self._numpy_X, dtype = torch.float))
                        self._list_X.append(torch.tensor(self._numpy_Y, dtype = torch.float))

                    f.close()
                    else:
                        raise RuntimeError('Please use proper csv file')


        def __getitem__(self, index):
            #if torch.is_tensor(idx):     #
            #    idx = idx.tolist()

            return self._list_X[index], self._list_Y[index]

        def __len__(self):
            return len(self._dataset)
```

1. CSV파일을 불러와서 이미지 데이터 추출
2. 리스트에 저장

데이터 전처리

```
for folderName, subFolder, fileNames in os.walk('./PE92_train/'+str(File_NAME)):
    for fileName in fileNames:
        if fileName == '.DS_Store':
            continue
        fileName = './PE92_train/'+str(File_NAME)+'/'+fileName
        fp = open(fileName, 'r+b')
        print(fileName)
        header = fp.read(8)
        width = 1
        class_num += 1
        while(1):
            code = int.from_bytes(fp.read(2), "big")
            if not code:
                break
            width = int.from_bytes(fp.read(1), "big")
            height = int.from_bytes(fp.read(1), "big")
            type = fp.read(1)
            reserved = fp.read(1)
            data = np.zeros(shape=(height, width), dtype=np.int)
            for i in range(height):
                for j in range(width):
                    data[i][j] = int.from_bytes(fp.read(1), "big")
            resized = resize(data, (32, 32))
            resized = resized * pow(10, 18)
            re = []
            for i in range(len(resized[0])):
                for j in range(len(resized[1])):
                    re.append(resized[i][j])
            train_X.append(re)
            train_y.append(code)
```

1. .hgu1 파일 포맷에 따라 데이터를 자른다
2. resize (32, 32)
3. resize된 데이터는 10^-18 과 같은 형태이므로 좀 더 보기 나은 형태를 위해 10^18을 곱해준다
4. 리스트에 저장

데이터
전처리

```
BATCH_SIZE = 32

torch_X_train = torch.from_numpy(X_train).type(torch.LongTensor)
torch_y_train = torch.from_numpy(y_train).type(torch.LongTensor) # dat

# create feature and targets tensor for test set.
torch_X_val = torch.from_numpy(X_val).type(torch.LongTensor)
torch_y_val = torch.from_numpy(y_val).type(torch.LongTensor) # data ty

#torch_X_train = torch_X_train.view(-1, 1,28,28).float()
torch_X_train = torch_X_train.view(-1, 1,32,32).float()
#torch_X_test = torch_X_test.view(-1,1,28,28).float()
torch_X_val = torch_X_val.view(-1,1,32,32).float()

print(torch_X_train.shape)
print(torch_X_val.shape)

# Pytorch train and test sets
train = torch.utils.data.TensorDataset(torch_X_train,torch_y_train)
val = torch.utils.data.TensorDataset(torch_X_val,torch_y_val)

return train, val, BATCH_SIZE          # label?
```

```
train_loader = torch.utils.data.DataLoader(train_, batch_size = BATCH_SIZE, shuffle = False)
test_loader = torch.utils.data.DataLoader(test_, batch_size = BATCH_SIZE, shuffle = False)
```

1. Tensor로 변환
2. DataSet으로 변환
3. DataLoader를 통해 데이터를 BatchSize에 따라 뽑아준다

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=5)
        self.conv3 = nn.Conv2d(32,64, kernel_size=5)
        #self.fc1 = nn.Linear(3*3*64, 256)
        self.fc1 = nn.Linear(64*21*21, 1024)
        self.fc2 = nn.Linear(1024, 256)
        self.fc3 = nn.Linear(256, 16)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        #x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv3(x),2))
        x = F.dropout(x, p=0.5, training=self.training)
        #print(x.shape)
        x = x.view(-1,21*21*64 )
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)

cnn = CNN()
cnn.cuda()
```

Model design

첫 번째 모델

- 레이어 총 6개
- Conv layer의 아웃풋에
  -> max pooling 실행
  -> ReLU로 활성화
  -> dropout
- FC layer 아웃풋에 ReLU와 Dropout

Test 정확도 : 91%

```python
class Model2(nn.Module):

    def __init__(self, num_classes=14):
        super(Model2, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

Model design

두 번째 모델

- AlexNet을 변형
- 레이어 총 8개
- Conv layer의 아웃풋에
  -> ReLU로 활성화
  -> max pooling 실행
- FC layer에서 dropout과 ReLU 사용

Test 정확도 : 93.5%

Model design

```
class CNN2(nn.Module):
    def __init__(self, class_num):
        super(CNN2, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 32, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25)
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 64, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25)
        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25)
        )

        self.fc = nn.Sequential(
            nn.Linear(128, class_num)
        )


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        #print(x.shape)
        x = x.view(-1, 1*1*128)
        x = self.fc(x)
        x = F.log_softmax(x, dim=1)
        return x
```

최종 모델

- 레이어 총 8개
- Conv layer의 아웃풋에
  -> ReLU로 활성화
  -> Batch Norm
- Conv layer 3번 통과시
  Pooling과 Dropout
- FC layer를 1개만 사용

Test 정확도 : 97~98%

Model design

```
class CNN2(nn.Module):
    def __init__(self, class_num):
        super(CNN2, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 32, 3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25)
        )
```

- 32개의 3*3 사이즈 필터 커널로 입력 영상을 컨벌루션

- 컨볼루션 보폭(stride) = 1
- padding = 1

- ReLU로 활성화
- BatchNorm

  …..

- Max Pooling (사이즈는 2x2)
- Drop out (p = 0.25)

Model design

```
self.conv2 = nn.Sequential(
    nn.Conv2d(32, 64, 3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(64),
    nn.Conv2d(64, 64, 3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(64),
    nn.Conv2d(64, 64, 3, stride=2, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(2, 2),
    nn.Dropout(0.25)
)
```

- 64개의 3*3 사이즈 필터 커널로 입력 영상을 컨벌루션

- 컨볼루션 보폭(stride) = 1
- padding = 1

- ReLU로 활성화
- BatchNorm

  ……

- Max Pooling (사이즈는 2x2)
- Drop out (p = 0.25)

Model design

```
self.conv3 = nn.Sequential(
    nn.Conv2d(64, 128, 3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(128),
    nn.MaxPool2d(2, 2),
    nn.Dropout(0.25)
)
```

- 128개의 3*3 사이즈 필터 커널로 입력 영상을 컨벌루션

- 컨볼루션 보폭(stride) = 1
- padding = 1

- ReLU로 활성화
- BatchNorm

- Max Pooling (사이즈는 2x2)
- Drop out (p = 0.25)

```
        self.fc = nn.Sequential(
            nn.Linear(128, class_num)
        )


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        #print(x.shape)
        x = x.view(-1, 1*1*128)
        x = self.fc(x)
        x = F.log_softmax(x, dim=1)
        return x
```

- class_num개의 1x1x128 사이즈 커널을 사용해 1x1xn feature map을 얻음 (n장의 1x1 사이즈 feature map)

- n개 뉴런의 출력 값에 softmax 함수를 적용해 n개 클래스 각각에 속할 확률 구함

프로젝트 모델 구성을 위한 함수 취합

- Dataset
- Model class
- fit

효율적으로 데이터를 로딩하기 위한 파일 나누기

```
(basal) ybhong@ark9:~/Deep_learning/Eden_2/Raw_data_1$ find -type f -name 'c5*' | xargs cp -t ./c5
```

```
b1b8.hgu1  b3a4.hgu1  b4ef.hgu1  b6db.hgu1  b8c7.hgu1  bab3.hgu1  bbfe.hgu1  bdea.hgu1  bfd6.hgu1  c1c3.hgu1  c3af.hgu1  c4fa.hgu1  c6e
b1b9.hgu1  b3a5.hgu1  b4f0.hgu1  b6dc.hgu1  b8c8.hgu1  bab4.hgu1  bc         bdeb.hgu1  bfd7.hgu1  c1c4.hgu1  c3b0.hgu1  c4fb.hgu1  c6e
b1ba.hgu1  b3a6.hgu1  b4f1.hgu1  b6dd.hgu1  b8c9.hgu1  bab5.hgu1  bca1.hgu1  bdec.hgu1  bfd8.hgu1  c1c5.hgu1  c3b1.hgu1  c4fc.hgu1  c6e
b1bb.hgu1  b3a7.hgu1  b4f2.hgu1  b6de.hgu1  b8ca.hgu1  bab6.hgu1  bca2.hgu1  bded.hgu1  bfd9.hgu1  c1c6.hgu1  c3b2.hgu1  c4fd.hgu1  c6e
b1bc.hgu1  b3a8.hgu1  b4f3.hgu1  b6df.hgu1  b8cb.hgu1  bab7.hgu1  bca3.hgu1  bdee.hgu1  bfda.hgu1  c1c7.hgu1  c3b3.hgu1  c4fe.hgu1  c6e
b1bd.hgu1  b3a9.hgu1  b4f4.hgu1  b6e0.hgu1  b8cc.hgu1  bab8.hgu1  bca4.hgu1  bdef.hgu1  bfdb.hgu1  c1c8.hgu1  c3b4.hgu1  c5a1.hgu1  c6e
b1be.hgu1  b3aa.hgu1  b4f5.hgu1  b6e1.hgu1  b8cd.hgu1  bab9.hgu1  bca5.hgu1  bdf0.hgu1  bfdc.hgu1  c1c9.hgu1  c3b5.hgu1  c5a2.hgu1  c6e
b1bf.hgu1  b3ab.hgu1  b4f6.hgu1  b6e2.hgu1  b8ce.hgu1  baba.hgu1  bca6.hgu1  bdf1.hgu1  bfdd.hgu1  c1ca.hgu1  c3b6.hgu1  c5a3.hgu1  c6e
b1c0.hgu1  b3ac.hgu1  b4f7.hgu1  b6e3.hgu1  b8cf.hgu1  babb.hgu1  bca7.hgu1  bdf2.hgu1  bfde.hgu1  c1cb.hgu1  c3b7.hgu1  c5a4.hgu1  c6e
b1c1.hgu1  b3ad.hgu1  b4f8.hgu1  b6e4.hgu1  b8d0.hgu1  babc.hgu1  bca8.hgu1  bdf3.hgu1  bfdf.hgu1  c1cc.hgu1  c3b8.hgu1  c5a5.hgu1  c6f
b1c2.hgu1  b3ae.hgu1  b4f9.hgu1  b6e5.hgu1  b8d1.hgu1  babd.hgu1  bca9.hgu1  bdf4.hgu1  bfe0.hgu1  c1cd.hgu1  c3b9.hgu1  c5a6.hgu1  c6f
b1c3.hgu1  b3af.hgu1  b4fa.hgu1  b6e6.hgu1  b8d2.hgu1  babe.hgu1  bcaa.hgu1  bdf5.hgu1  bfe1.hgu1  c1ce.hgu1  c3ba.hgu1  c5a7.hgu1  c6f
b1c4.hgu1  b3b0.hgu1  b4fb.hgu1  b6e7.hgu1  b8d3.hgu1  babf.hgu1  bcab.hgu1  bdf6.hgu1  bfe2.hgu1  c1cf.hgu1  c3bb.hgu1  c5a8.hgu1  c6f
b1c5.hgu1  b3b1.hgu1  b4fc.hgu1  b6e8.hgu1  b8d4.hgu1  bac0.hgu1  bcac.hgu1  bdf7.hgu1  bfe3.hgu1  c1d0.hgu1  c3bc.hgu1  c5a9.hgu1  c6f
b1c6.hgu1  b3b2.hgu1  b4fd.hgu1  b6e9.hgu1  b8d5.hgu1  bac1.hgu1  bcad.hgu1  bdf8.hgu1  bfe4.hgu1  c1d1.hgu1  c3bd.hgu1  c5aa.hgu1  c6f
b1c7.hgu1  b3b3.hgu1  b4fe.hgu1  b6ea.hgu1  b8d6.hgu1  bac2.hgu1  bcae.hgu1  bdf9.hgu1  bfe5.hgu1  c1d2.hgu1  c3be.hgu1  c5ab.hgu1  c6f
b1c8.hgu1  b3b4.hgu1  b5         b6eb.hgu1  b8d7.hgu1  bac3.hgu1  bcaf.hgu1  bdfa.hgu1  bfe6.hgu1  c1d3.hgu1  c3bf.hgu1  c5ac.hgu1  c6f
b1c9.hgu1  b3b5.hgu1  b5a1.hgu1  b6ec.hgu1  b8d8.hgu1  bac4.hgu1  bcb0.hgu1  bdfb.hgu1  bfe7.hgu1  c1d4.hgu1  c3c0.hgu1  c5ad.hgu1  c6f
b1ca.hgu1  b3b6.hgu1  b5a2.hgu1  b6ed.hgu1  b8d9.hgu1  bac5.hgu1  bcb1.hgu1  bdfc.hgu1  bfe8.hgu1  c1d5.hgu1  c3c1.hgu1  c5ae.hgu1  c6f
b1cb.hgu1  b3b7.hgu1  b5a3.hgu1  b6ee.hgu1  b8da.hgu1  bac6.hgu1  bcb2.hgu1  bdfd.hgu1  bfe9.hgu1  c1d6.hgu1  c3c2.hgu1  c5af.hgu1  c6f
b1cc.hgu1  b3b8.hgu1  b5a4.hgu1  b6ef.hgu1  b8db.hgu1  bac7.hgu1  bcb3.hgu1  bdfe.hgu1  bfea.hgu1  c1d7.hgu1  c3c3.hgu1  c5b0.hgu1  c6f
b1cd.hgu1  b3b9.hgu1  b5a5.hgu1  b6f0.hgu1  b8dc.hgu1  bac8.hgu1  bcb4.hgu1  be         bfeb.hgu1  c1d8.hgu1  c3c4.hgu1  c5b1.hgu1  c6f
b1ce.hgu1  b3ba.hgu1  b5a6.hgu1  b6f1.hgu1  b8dd.hgu1  bac9.hgu1  bcb5.hgu1  bea1.hgu1  bfec.hgu1  c1d9.hgu1  c3c5.hgu1  c5b2.hgu1  c6f
b1cf.hgu1  b3bb.hgu1  b5a7.hgu1  b6f2.hgu1  b8de.hgu1  baca.hgu1  bcb6.hgu1  bea2.hgu1  bfed.hgu1  c1da.hgu1  c3c6.hgu1  c5b3.hgu1  c6f
b1d0.hgu1  b3bc.hgu1  b5a8.hgu1  b6f3.hgu1  b8df.hgu1  bacb.hgu1  bcb7.hgu1  bea3.hgu1  bfee.hgu1  c1db.hgu1  c3c7.hgu1  c5b4.hgu1  c7
b1d1.hgu1  b3bd.hgu1  b5a9.hgu1  b6f4.hgu1  b8e0.hgu1  bacc.hgu1  bcb8.hgu1  bea4.hgu1  bfef.hgu1  c1dc.hgu1  c3c8.hgu1  c5b5.hgu1  c7a
b1d2.hgu1  b3be.hgu1  b5aa.hgu1  b6f5.hgu1  b8e1.hgu1  bacd.hgu1  bcb9.hgu1  bea5.hgu1  bff0.hgu1  c1dd.hgu1  c3c9.hgu1  c5b6.hgu1  c7a
b1d3.hgu1  b3bf.hgu1  b5ab.hgu1  b6f6.hgu1  b8e2.hgu1  bace.hgu1  bcba.hgu1  bea6.hgu1  bff1.hgu1  c1de.hgu1  c3ca.hgu1  c5b7.hgu1  c7a
b1d4.hgu1  b3c0.hgu1  b5ac.hgu1  b6f7.hgu1  b8e3.hgu1  bacf.hgu1  bcbb.hgu1  bea7.hgu1  bff2.hgu1  c1df.hgu1  c3cb.hgu1  c5b8.hgu1  c7a
b1d5.hgu1  b3c1.hgu1  b5ad.hgu1  b6f8.hgu1  b8e4.hgu1  bad0.hgu1  bcbc.hgu1  bea8.hgu1  bff3.hgu1  c1e0.hgu1  c3cc.hgu1  c5b9.hgu1  c7a
b1d6.hgu1  b3c2.hgu1  b5ae.hgu1  b6f9.hgu1  b8e5.hgu1  bad1.hgu1  bcbd.hgu1  bea9.hgu1  bff4.hgu1  c1e1.hgu1  c3cd.hgu1  c5ba.hgu1  c7a
b1d7.hgu1  b3c3.hgu1  b5af.hgu1  b6fa.hgu1  b8e6.hgu1  bad2.hgu1  bcbe.hgu1  beaa.hgu1  bff5.hgu1  c1e2.hgu1  c3ce.hgu1  c5bb.hgu1  c7a
b1d8.hgu1  b3c4.hgu1  b5b0.hgu1  b6fb.hgu1  b8e7.hgu1  bad3.hgu1  bcbf.hgu1  beab.hgu1  bff6.hgu1  c1e3.hgu1  c3cf.hgu1  c5bc.hgu1  c7a
b1d9.hgu1  b3c5.hgu1  b5b1.hgu1  b6fc.hgu1  b8e8.hgu1  bad4.hgu1  bcc0.hgu1  beac.hgu1  bff7.hgu1  c1e4.hgu1  c3d0.hgu1  c5bd.hgu1  c7a
b1da.hgu1  b3c6.hgu1  b5b2.hgu1  b6fd.hgu1  b8e9.hgu1  bad5.hgu1  bcc1.hgu1  bead.hgu1  bff8.hgu1  c1e5.hgu1  c3d1.hgu1  c5be.hgu1  c7a
b1db.hgu1  b3c7.hgu1  b5b3.hgu1  b6fe.hgu1  b8ea.hgu1  bad6.hgu1  bcc2.hgu1  beae.hgu1  bff9.hgu1  c1e6.hgu1  c3d2.hgu1  c5bf.hgu1  c7a
b1dc.hgu1  b3c8.hgu1  b5b4.hgu1  b7         b8eb.hgu1  bad7.hgu1  bcc3.hgu1  beaf.hgu1  bffa.hgu1  c1e7.hgu1  c3d3.hgu1  c5c0.hgu1  c7a
b1dd.hgu1  b3c9.hgu1  b5b5.hgu1  b7a1.hgu1  b8ec.hgu1  bad8.hgu1  bcc4.hgu1  beb0.hgu1  bffb.hgu1  c1e8.hgu1  c3d4.hgu1  c5c1.hgu1  c7a
b1de.hgu1  b3ca.hgu1  b5b6.hgu1  b7a2.hgu1  b8ed.hgu1  bad9.hgu1  bcc5.hgu1  beb1.hgu1  bffc.hgu1  c1e9.hgu1  c3d5.hgu1  c5c2.hgu1  c7a
b1df.hgu1  b3cb.hgu1  b5b7.hgu1  b7a3.hgu1  b8ee.hgu1  bada.hgu1  bcc6.hgu1  beb2.hgu1  bffd.hgu1  c1ea.hgu1  c3d6.hgu1  c5c3.hgu1  c7a
b1e0.hgu1  b3cc.hgu1  b5b8.hgu1  b7a4.hgu1  b8ef.hgu1  badb.hgu1  bcc7.hgu1  beb3.hgu1  bffe.hgu1  c1eb.hgu1  c3d7.hgu1  c5c4.hgu1  c7b
b1e1.hgu1  b3cd.hgu1  b5b9.hgu1  b7a5.hgu1  b8f0.hgu1  badc.hgu1  bcc8.hgu1  beb4.hgu1  c0a1.hgu1  c1ec.hgu1  c3d8.hgu1  c5c5.hgu1  c7b
b1e2.hgu1  b3ce.hgu1  b5ba.hgu1  b7a6.hgu1  b8f1.hgu1  badd.hgu1  bcc9.hgu1  beb5.hgu1  c0a2.hgu1  c1ed.hgu1  c3d9.hgu1  c5c6.hgu1  c7b
b1e3.hgu1  b3cf.hgu1  b5bb.hgu1  b7a7.hgu1  b8f2.hgu1  bade.hgu1  bcca.hgu1  beb6.hgu1  c0a3.hgu1  c1ee.hgu1  c3da.hgu1  c5c7.hgu1  c7b
b1e4.hgu1  b3d0.hgu1  b5bc.hgu1  b7a8.hgu1  b8f3.hgu1  badf.hgu1  bccb.hgu1  beb7.hgu1  c0a4.hgu1  c1ef.hgu1  c3db.hgu1  c5c8.hgu1  c7b
b1e5.hgu1  b3d1.hgu1  b5bd.hgu1  b7a9.hgu1  b8f4.hgu1  bae0.hgu1  bccc.hgu1  beb8.hgu1  c0a5.hgu1  c1f0.hgu1  c3dc.hgu1  c5c9.hgu1  c7b
b1e6.hgu1  b3d2.hgu1  b5be.hgu1  b7aa.hgu1  b8f5.hgu1  bae1.hgu1  bccd.hgu1  beb9.hgu1  c0a6.hgu1  c1f1.hgu1  c3dd.hgu1  c5ca.hgu1  c7b
b1e7.hgu1  b3d3.hgu1  b5bf.hgu1  b7ab.hgu1  b8f6.hgu1  bae2.hgu1  bcce.hgu1  beba.hgu1  c0a7.hgu1  c1f2.hgu1  c3de.hgu1  c5cb.hgu1  c7b
b1e8.hgu1  b3d4.hgu1  b5c0.hgu1  b7ac.hgu1  b8f7.hgu1  bae3.hgu1  bccf.hgu1  bebb.hgu1  c0a8.hgu1  c1f3.hgu1  c3df.hgu1  c5cc.hgu1  c7b
b1e9.hgu1  b3d5.hgu1  b5c1.hgu1  b7ad.hgu1  b8f8.hgu1  bae4.hgu1  bcd0.hgu1  bebc.hgu1  c0a9.hgu1  c1f4.hgu1  c3e0.hgu1  c5cd.hgu1  c7b
b1ea.hgu1  b3d6.hgu1  b5c2.hgu1  b7ae.hgu1  b8f9.hgu1  bae5.hgu1  bcd1.hgu1  bebd.hgu1  c0aa.hgu1  c1f5.hgu1  c3e1.hgu1  c5ce.hgu1  c7b
(basal) ybhong@ark9:~/Deep_learning/Eden_2/Raw_data_1/PE92_train$ find -type f -name 'c5*' | xargs cp -t ./c5
```

모든 클래스의 데이터를 한 번에 받아오기 위한 코딩

```
def Dataset(File_NAME, label, class_num):          # label?
    #class_num = 0
    train_X = []
    train_y = []
    for folderName, subFolder, fileNames in os.walk('./PE92_train/'+str(File_NAME)):
        for fileName in fileNames:
            if fileName == '.DS_Store':
                continue
            fileName = './PE92_train/'+str(File_NAME)+'/'+fileName
            fp = open(fileName, 'r+b')
            print(fileName)
            header = fp.read(8)
            width = 1
            class_num += 1
            while(1):
                code = int.from_bytes(fp.read(2), "big")
                if not code:
                    break
                width = int.from_bytes(fp.read(1), "big")
                height = int.from_bytes(fp.read(1), "big")
                type = fp.read(1)
                reserved = fp.read(1)
                data = np.zeros(shape=(height, width), dtype=np.int)
                for i in range(height):
                    for j in range(width):
                        data[i][j] = int.from_bytes(fp.read(1), "big")
                resized = resize(data, (32, 32))
                resized = resized * pow(10, 18)
                re = []
                for i in range(len(resized[0])):
                    for j in range(len(resized[1])):
                        re.append(resized[i][j])
                train_X.append(re)
                train_y.append(code)


    print("class_num =", class_num)

    train_X_np = np.asarray(train_X, dtype='float32')
    train_y_np = np.asarray(train_y, dtype='int')
```

Input

- File_NAME_list
- label
- class_num

모든 클래스의 데이터를 한 번에 받아오기 위한 코딩

```
for id, x in enumerate(train_y_np):
    if x in label:
        continue
    else:
        label.append(x)
for id, x in enumerate(train_y_np):
    train_y_np[id] = label.index(x)

X_train, X_test, y_train, y_test = train_test_split(train_X_np, train_y_np, test_size=0.15)


BATCH_SIZE = 32

torch_X_train = torch.from_numpy(X_train).type(torch.LongTensor)
torch_y_train = torch.from_numpy(y_train).type(torch.LongTensor) # data type is long

# create feature and targets tensor for test set.
torch_X_test = torch.from_numpy(X_test).type(torch.LongTensor)
torch_y_test = torch.from_numpy(y_test).type(torch.LongTensor) # data type is long


#torch_X_train = torch_X_train.view(-1, 1,28,28).float()
torch_X_train = torch_X_train.view(-1, 1,32,32).float()
#torch_X_test = torch_X_test.view(-1,1,28,28).float()
torch_X_test = torch_X_test.view(-1,1,32,32).float()
print(torch_X_train.shape)
print(torch_X_test.shape)

# Pytorch train and test sets
train = torch.utils.data.TensorDataset(torch_X_train,torch_y_train)
test = torch.utils.data.TensorDataset(torch_X_test,torch_y_test)

return train, test, class_num, BATCH_SIZE, label          # label?
```

Output

- train TensorDataset
- test TensorDataset
- class_num
- BATCH_SIZE
- label

모든 클래스의 데이터를 한 번에 받아오기 위한 코딩

```
File_NAME_list = ['b0','b2','c0', 'c1','b3','b4','b5','b6','b7','b8','b9','ba','bb','bc','bd','be','bf','b1','c2','c3','c4','c5','c6','c7','c8']
#File_NAME_list = ['b2','c1','b3', 'b4', 'b5','b6', 'b7', 'b8', 'b9', 'ba', 'bb', 'bc', 'bd']

#File_NAME_list = ['b0', 'b2', 'c0']
label = []
class_num = 0

print(File_NAME_list)

for File_NAME in File_NAME_list:
    train, test, class_num, BATCH_SIZE, label = Dataset(File_NAME,label, class_num)          # label??

    if File_NAME == 'b0':
        train_ = train
        test_ = test

    else:
        train_ += train
        test_ += test

# data loader

train_loader = torch.utils.data.DataLoader(train_, batch_size = BATCH_SIZE, shuffle = True)
test_loader = torch.utils.data.DataLoader(test_, batch_size = BATCH_SIZE, shuffle = False)
```

Parameter 조정

```
def Dataset(File_NAME, label, train_X, train_y, class_num):          # label?


    for folderName, subFolder, fileNames in os.walk( './PE92_train/'+str(File_NAME)):
        for fileName in fileNames:
            if fileName == '.DS_Store':
                continue
            fileName = './PE92_train/'+str(File_NAME)+'/'+fileName
            fp = open(fileName, 'r+b')
            print(fileName)
            header = fp.read(8)
            width = 1
            class_num += 1
            while(1):
                code = int.from_bytes(fp.read(2), "big")
                if not code:
                    break
                width = int.from_bytes(fp.read(1), "big")
                height = int.from_bytes(fp.read(1), "big")
                type = fp.read(1)
                reserved = fp.read(1)
                data = np.zeros(shape=(height, width), dtype=np.int)
                for i in range(height):
                    for j in range(width):
                        data[i][j] = int.from_bytes(fp.read(1), "big")
                resized = resize(data, (32, 32))
                resized = resized * pow(10, 18)
                re = []
                for i in range(len(resized[0])):
                    for j in range(len(resized[1])):
                        re.append(resized[i][j])
                train_X.append(re)
                train_y.append(code)

    print(len(train_X))
    print(len(train_y))

    return train_X, train_y, class_num, label
```

resize 100, 100

- memory 부족 오류

resize 32, 32

- 정상 작동

Parameter 조정

```
class CNN2(nn.Module):
    def __init__(self, class_num):
        super(CNN2, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1),
            #nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 32, 3, padding=1),
            #nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 32, 3, stride=2, padding=1),
            #nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.25)
        )
```

- BatchNorm2d 위치 조정

Validation set Performance_30 Epochs

```
Epoch : 29 [118400/170170 (70%)]        Loss: 0.247708   Accuracy:92.721%
Epoch : 29 [120000/170170 (71%)]        Loss: 0.375208   Accuracy:92.709%
Epoch : 29 [121600/170170 (71%)]        Loss: 0.293838   Accuracy:92.703%
Epoch : 29 [123200/170170 (72%)]        Loss: 0.335486   Accuracy:92.695%
Epoch : 29 [124800/170170 (73%)]        Loss: 0.559123   Accuracy:92.689%
Epoch : 29 [126400/170170 (74%)]        Loss: 0.285273   Accuracy:92.674%
Epoch : 29 [128000/170170 (75%)]        Loss: 0.088662   Accuracy:92.663%
Epoch : 29 [129600/170170 (76%)]        Loss: 0.154596   Accuracy:92.648%
Epoch : 29 [131200/170170 (77%)]        Loss: 0.329279   Accuracy:92.627%
Epoch : 29 [132800/170170 (78%)]        Loss: 0.088634   Accuracy:92.620%
Epoch : 29 [134400/170170 (79%)]        Loss: 0.084690   Accuracy:92.618%
Epoch : 29 [136000/170170 (80%)]        Loss: 0.604876   Accuracy:92.621%
Epoch : 29 [137600/170170 (81%)]        Loss: 0.320893   Accuracy:92.621%
Epoch : 29 [139200/170170 (82%)]        Loss: 0.146950   Accuracy:92.614%
Epoch : 29 [140800/170170 (83%)]        Loss: 0.194274   Accuracy:92.603%
Epoch : 29 [142400/170170 (84%)]        Loss: 0.343211   Accuracy:92.596%
Epoch : 29 [144000/170170 (85%)]        Loss: 0.263191   Accuracy:92.586%
Epoch : 29 [145600/170170 (86%)]        Loss: 0.204490   Accuracy:92.587%
Epoch : 29 [147200/170170 (86%)]        Loss: 0.136684   Accuracy:92.580%
Epoch : 29 [148800/170170 (87%)]        Loss: 0.438482   Accuracy:92.576%
Epoch : 29 [150400/170170 (88%)]        Loss: 0.320333   Accuracy:92.575%
Epoch : 29 [152000/170170 (89%)]        Loss: 0.160604   Accuracy:92.571%
Epoch : 29 [153600/170170 (90%)]        Loss: 0.074025   Accuracy:92.573%
Epoch : 29 [155200/170170 (91%)]        Loss: 0.556360   Accuracy:92.569%
Epoch : 29 [156800/170170 (92%)]        Loss: 0.059127   Accuracy:92.571%
Epoch : 29 [158400/170170 (93%)]        Loss: 0.103897   Accuracy:92.560%
Epoch : 29 [160000/170170 (94%)]        Loss: 0.329108   Accuracy:92.568%
Epoch : 29 [161600/170170 (95%)]        Loss: 0.214389   Accuracy:92.561%
Epoch : 29 [163200/170170 (96%)]        Loss: 0.251829   Accuracy:92.549%
Epoch : 29 [164800/170170 (97%)]        Loss: 0.173202   Accuracy:92.547%
Epoch : 29 [166400/170170 (98%)]        Loss: 0.322577   Accuracy:92.541%
Epoch : 29 [168000/170170 (99%)]        Loss: 0.266276   Accuracy:92.536%
Epoch : 29 [169600/170170 (100%)]       Loss: 0.214066   Accuracy:92.534%
DEVICE = cuda
******************** Test ********************
Loss: 0.19203022122383118, Accuracy: 0.9491147497337593 %
**********************************************
```

Validation set Performance_40 Epochs

```
Epoch : 39 [126400/170170 (74%)]        Loss: 0.134154   Accuracy:93.594%
Epoch : 39 [128000/170170 (75%)]        Loss: 0.216264   Accuracy:93.576%
Epoch : 39 [129600/170170 (76%)]        Loss: 0.404076   Accuracy:93.582%
Epoch : 39 [131200/170170 (77%)]        Loss: 0.095116   Accuracy:93.574%
Epoch : 39 [132800/170170 (78%)]        Loss: 0.148743   Accuracy:93.563%
Epoch : 39 [134400/170170 (79%)]        Loss: 0.050936   Accuracy:93.568%
Epoch : 39 [136000/170170 (80%)]        Loss: 0.043081   Accuracy:93.564%
Epoch : 39 [137600/170170 (81%)]        Loss: 0.047600   Accuracy:93.567%
Epoch : 39 [139200/170170 (82%)]        Loss: 0.424792   Accuracy:93.569%
Epoch : 39 [140800/170170 (83%)]        Loss: 0.064635   Accuracy:93.558%
Epoch : 39 [142400/170170 (84%)]        Loss: 0.722676   Accuracy:93.553%
Epoch : 39 [144000/170170 (85%)]        Loss: 0.464606   Accuracy:93.544%
Epoch : 39 [145600/170170 (86%)]        Loss: 0.198203   Accuracy:93.552%
Epoch : 39 [147200/170170 (86%)]        Loss: 0.471816   Accuracy:93.548%
Epoch : 39 [148800/170170 (87%)]        Loss: 0.991798   Accuracy:93.552%
Epoch : 39 [150400/170170 (88%)]        Loss: 0.077154   Accuracy:93.565%
Epoch : 39 [152000/170170 (89%)]        Loss: 0.193998   Accuracy:93.563%
Epoch : 39 [153600/170170 (90%)]        Loss: 0.495810   Accuracy:93.544%
Epoch : 39 [155200/170170 (91%)]        Loss: 0.108671   Accuracy:93.533%
Epoch : 39 [156800/170170 (92%)]        Loss: 0.531261   Accuracy:93.533%
Epoch : 39 [158400/170170 (93%)]        Loss: 0.234160   Accuracy:93.522%
Epoch : 39 [160000/170170 (94%)]        Loss: 0.071752   Accuracy:93.521%
Epoch : 39 [161600/170170 (95%)]        Loss: 0.161748   Accuracy:93.524%
Epoch : 39 [163200/170170 (96%)]        Loss: 0.807206   Accuracy:93.525%
Epoch : 39 [164800/170170 (97%)]        Loss: 0.050177   Accuracy:93.530%
Epoch : 39 [166400/170170 (98%)]        Loss: 0.240794   Accuracy:93.525%
Epoch : 39 [168000/170170 (99%)]        Loss: 0.298214   Accuracy:93.523%
Epoch : 39 [169600/170170 (100%)]       Loss: 0.267562   Accuracy:93.529%
DEVICE = cuda
******************** Test ********************
Loss: 0.25350421667099, Accuracy: 0.9505457933972311 %
**********************************************
```

Test dataset Peformance

```
CNN2(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.25, inplace=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=128, out_features=2350, bias=True)
  )
)
DEVICE = cuda
******************** Test ********************
Loss: 0.0030983483884483576, Accuracy: 984.2185792349727 %
**********************************************
```

Test dataset Peformance

```
CNN2(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.25, inplace=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=128, out_features=2350, bias=True)
  )
)
DEVICE = cuda
******************** Test ********************
Loss: 0.0012723742984235287, Accuracy: 984.1311475409836 %
**********************************************
```

Test dataset Peformance

```
print('Loss: {}, Accuracy: {} %'.format(loss.item(), acc/len(test_loader)*BATCH_SIZE))
```

```
print('Loss: {}, Accuracy: {} %'.format(loss.item(), acc/(len(test_loader)*BATCH_SIZE)))
```

Test dataset Peformance_30 Epochs model

```
CNN2(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.25, inplace=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=128, out_features=2350, bias=True)
  )
)
DEVICE = cuda
******************** Test ********************
Loss: 0.0012723742984235287, Accuracy: 0.9610655737704918 %
**********
```

Test dataset Peformance_40 Epochs model

```
CNN2(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Dropout(p=0.25, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.25, inplace=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=128, out_features=2350, bias=True)
  )
)
DEVICE = cuda
******************** Test ********************
Loss: 0.0019221074180677533, Accuracy: 0.9638405054644809 %
**********************************************
```

Member	Role	Is leader?	Contribution (%)
손한솔	전처리/모델 만들기	o	33%
홍유빈	전처리/테스트/최종디버깅		33%
김시온	전처리		33%