인공지능개론 2023 프로젝트

박형민 교수님
20191611 유종선
2023.12.24

1. 과제목표

- 다양한 폰트로 구성된 영문자 및 숫자를 분류하는 문제를 딥러닝을 이용하여 해결한다. 아래의 예시처럼 숫자 6, 영문자 Q, C, n 이 다양한 폰트로 제공되어 있다. 딥러닝을 통해 학습하여 어떠한 문자인지를 classify한다.

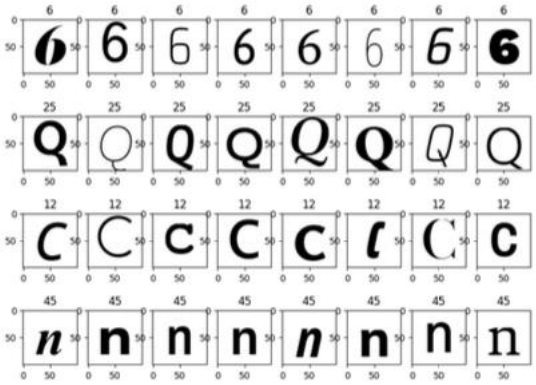


그림 1: 데이터 예시

-

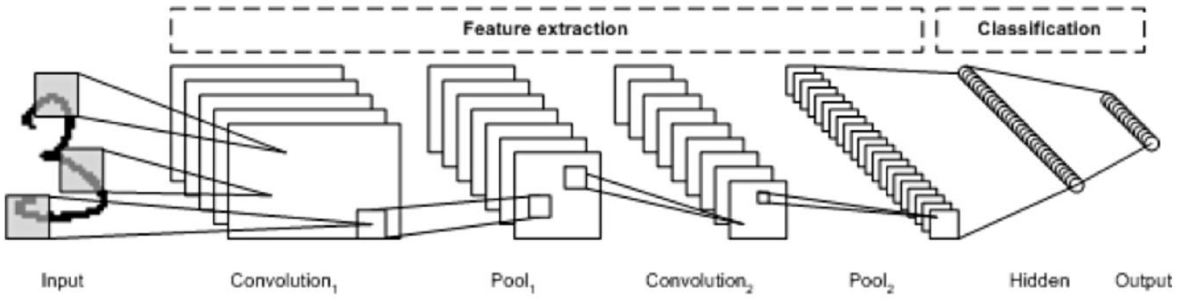
- 우리가 분류해야 할 데이터는 다음과 같다.

```
label_dict = {
    '30': 0, '31': 1, '32': 2, '33': 3, '34': 4, '35': 5, '36': 6, '37': 7, '38': 8, '39': 9,
    '41': 10, '42': 11, '43': 12, '44': 13, '45': 14, '46': 15, '47': 16, '48': 17, '49': 18,
    '4a': 19, '4b': 20, '4c': 21, '4d': 22, '4e': 23, '50': 24, '51': 25, '52': 26, '53': 27,
    '54': 28, '55': 29, '56': 30, '57': 31, '58': 32, '59': 33, '5a': 34, '61': 35, '62': 36,
    '64': 37, '65': 38, '66': 39, '67': 40, '68': 41, '69': 42, '6a': 43, '6d': 44, '6e': 45,
    '6f': 46, '71': 47, '72': 48, '74': 49, '75': 50, '79': 51,
}
```

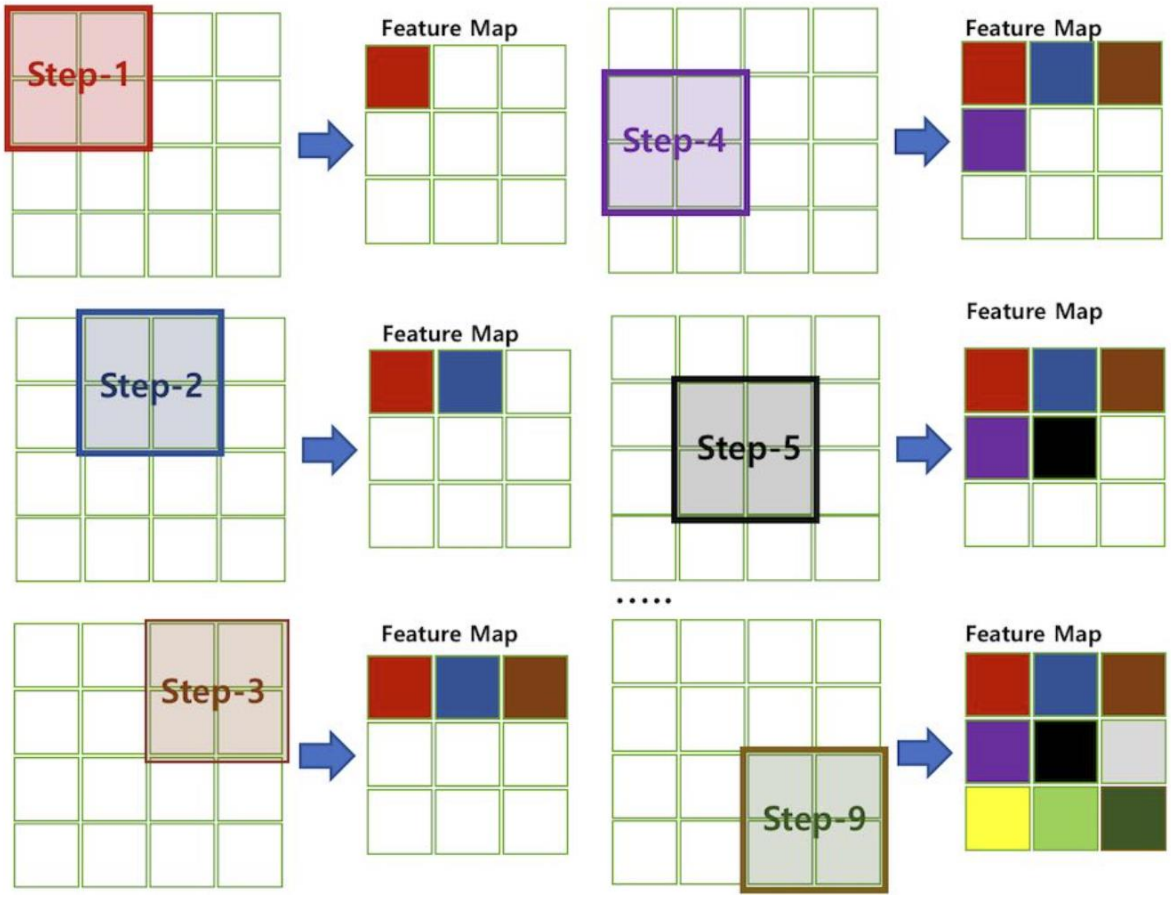
-
- 이미지 데이터는 shape이 channels = 1, height = 100, width = 100이다.
- Train data는 41,600개인데, 이를 학습시켜서 valid data = 15,00개를 통해 정확도를 test한다.

2. 배경이론

- 이미지 데이터 classification에는 CNN(Convolution Neural Network)가 다른 모델에 비해서 이점을 가지고 있다. 그렇다면 CNN이 가지는 이점을 알아보자.

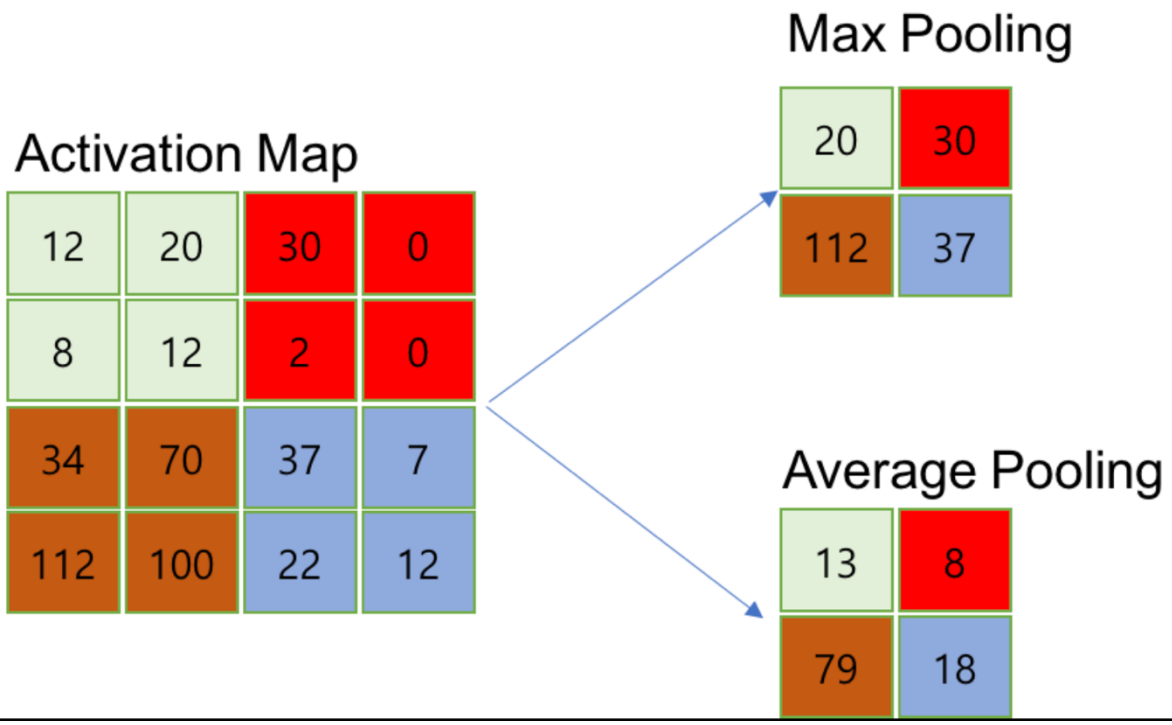


- 다음의 그림처럼, input 이미지를 convolution layer로 입력해서 input image의 특징(features)들을 잡아낸다. 예를 들어, 글씨체나 선의 굵기, 선과 배경의 경계는 어떻게 구성되어 있는지 등 이러한 특징들을 학습할 수 있다. Filter를 통해서 이미지의 특징을 모으고, pooling을 통해서 강화할 수 있다. 먼저 feature map을 통해서 어떻게 feature를 잡아내는지 알아보자.



- 다음과 같이 feature map이 진행된다. Step 에서 보이는 2*2 를 묶어서 오른쪽의 하나의 공간으로 넣어주는 것이다. 여기서 그 공간의 특징을 잡아내는 것이다. 여기서 step의 크기나 step 사이의 간격 등은 파라미터를 통해서 조절할 수 있다. 당연히 더 촘촘하게 feature를 잡게 되면, 학습에 더 오랜시간이 걸리고, 더 적게 feature를 잡게 되면 학습에 더 적은 시간이 걸리게 된다. 이렇게 파라미터를 조절해서 학습속도나 진행시간 등을 조절할 수 있다는 것 또한 CNN의 장점이다.

- 이렇게 convolution layer에서 나온 출력은 pooling layer로 들어가게 된다. 이 pooling layer는 출력 데이터의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용된다. Pooling layer에서는 3가지 방식으로 처리를 하는데, max pooling, min pooling, average pooling이 있다. 아래 그림에서 볼 수 있듯이 이름에 맞게 max, min, average 값을 선택해서 feature를 강조하는 것이다.



- 이렇게 convolution – pooling – convolution – pooling 등의 layer들을 거치고 나면 classification을 위한 fully connected layer로 들어가게 된다. 여기서는 parameter가 따로 존재하지 않고, 그저 shape을 바꾸어 주기만 한다.
- 여기서 최종 data shape은 (52, 1)이 된다. 여기서 softmax layer를 거쳐서 최종적인 결과를 갖게 된다.

3. 과제 수행 방법

코드별로 수행하는 단계를 서술하겠다.

1. 먼저 drive에 있는 data를 올려준다.

```
# mount google drive
from google.colab import drive
drive.mount('/content/gdrive')

# unzip train, valid dataset
# target 을 정하지 않으면 google drive 내의 content 드라이브에 위치시킴
# 런타임을 다시 시작할 때 마다 unzip 을 새로 해주어야 함.
!unzip /content/gdrive/MyDrive/2023_EEE4178_project/train.zip
!unzip /content/gdrive/MyDrive/2023_EEE4178_project/valid.zip
# !unzip /content/gdrive/MyDrive/Project_dataset/Font_npy_100_test.zip
```

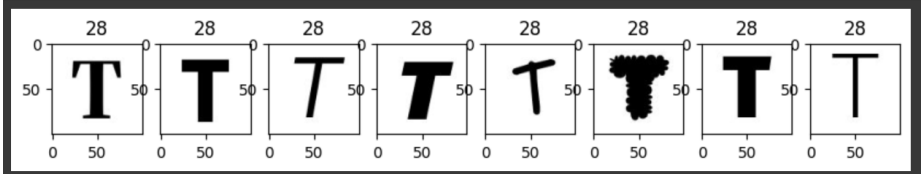
2. Train_loader와 valid_loader에 데이터를 올려준다. 원래는 test_loader이지만, 코드 구현시에는 valid_loader를 사용해서 test한다. 그럼 다음과 같이 image.shape이 나오는데 batch_size에 따라서 가장 앞의 숫자가 결정되고, 이미지는 channel = 1로 grayscale, 100 * 100이 된다.

```
batch_size = 100
train_loader = torch.utils.data.DataLoader(dataset=train_data,
                                           batch_size=batch_size,
                                           shuffle=True)

valid_loader = torch.utils.data.DataLoader(dataset=valid_data,
                                           batch_size=batch_size,
                                           shuffle=True)



# check dataloader
image,label = next(iter(valid_loader))
print(image.shape)
print(label.shape)
```

```
torch.Size([100, 1, 100, 100])
torch.Size([100])
```

3. 데이터의 형태를 열어보면 다음과 같다. 학습시에 꼭 필요한 코드는 아니다.

```
# visualize data
# image_show function : num 수 만큼 dataset 내의 data를 보여주는 함수
def image_show(dataset, num):
  fig = plt.figure(figsize=(10,10))

  for i in range(num):
    plt.subplot(1, num, i+1)
    plt.imshow(dataset[i+12200][0].squeeze(), cmap="gray")
    plt.title(dataset[i+12200][1].item())  # .item()을 사용하여 텐서에서 숫자로 변환

image_show(train_data, 8)
```



4. Seed를 설정해준다. 현 프로젝트에서는 seed = 42로 고정이다.

```
# FIX SEED
def seed_everything(seed: int = 42):
    random.seed(seed)
    np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)  # type: ignore
    torch.backends.cudnn.deterministic = True  # type: ignore
    torch.backends.cudnn.benchmark = True  # type: ignore

seed_everything(42)
```

5. Hyper – parameters를 초기화 해준다.

```
# hyper - parameters
num_classes = 52
input_size = 100
num_epochs = 13
learning_rate = 0.001
hidden_dim = 128
num_layers = 3
drop_percent = 0.2
in_channel = 1
embedding_dim = 64
dropout = 0.2
max_pool_kernel = 2
```

6. 다음은 이제 CNN layer를 구성하는 코드이다. 코드가 길어서 하나의 layer를 대표로 설명하겠다. Conv2d로 입력데이터가 들어간다. Out_channels = 32로 두었고, kernel_size = 3, padding = 2, stride = 1로 설정하였다. 이 parameter의 값들은 layer마다 조금씩 차이가 있다. BatchNorm2d를 통해서 Batch Normalization을 해준다. 활성화함수 relu를 거치고 dropout을 해준다. 그 출력이 maxpooling layer로 들어간다. Maxpooling이 끝나면 weight initialization을 해준다. 활성화 함수로 relu를 썼기 때문에 He initialization을 사용하였다.

```
self.layer1 = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=2, stride=1),
    nn.BatchNorm2d(num_features=32),  # out_channels이 그대로 들어감
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.MaxPool2d(kernel_size=max_pool_kernel, stride = 2)  # 2 * 2
)
#nn.init.kaiming_normal_(self.layer1.weight, mode='fan_out', nonlinearity='relu')
self.layer1.apply(self._init_weights)
```

7. Convolution layer가 끝나고 난 후에 fully-connected layer의 모습이다.

```
self.fc1 = nn.Linear(in_features=2304, out_features=256)
nn.init.kaiming_normal_(self.fc1.weight, mode='fan_out', nonlinearity='relu')
self.fc2 = nn.Linear(in_features=256, out_features=128)
nn.init.kaiming_normal_(self.fc2.weight, mode='fan_out', nonlinearity='relu')
#self.dropout1 = nn.Dropout(p=0.25, inplace=False)
self.fc3 = nn.Linear(in_features=128, out_features=num_classes)
```

8. 다음과 같이 forward 함수를 구성한다.

```
def forward(self, x):
  x = self.layer1(x)
  x = self.layer2(x)
  x = self.layer3(x)
  x = self.layer4(x)

  #x = x.reshape(x.size(0), -1, embedding_dim)

  #x, _ = self.lstm(x)

  x = x.reshape(x.size(0), -1)  # fully connected에 넣어주
  x = self.fc1(x)
  x = self.dropout(x)
  x = F.relu(x)
  x = self.fc2(x)
  x = self.dropout(x)
  x = F.relu(x)
  x = self.fc3(x)

  return x  # 10개의 출력 return
```

9. Model을 만들었으니, model을 선언해주고 to.device 해준다. Criterion과 optimizer를 선언해준다.

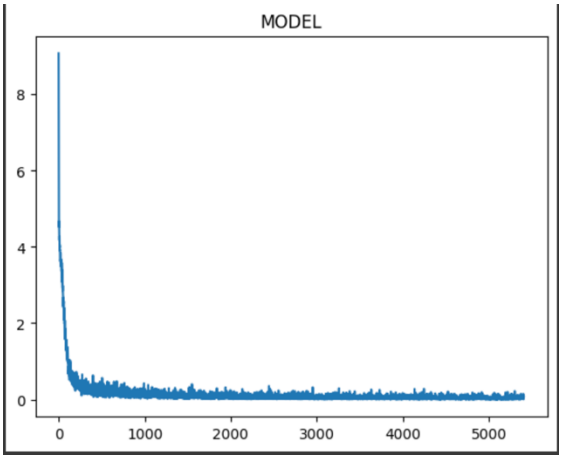
```
model = CNN(embedding_dim, hidden_dim, num_layers).to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

best_epoch = 0
best_loss = float('inf')
total_step = len(train_loader)
loss_list = []
```

10. 이제 학습을 시키고 결과를 받으면 끝이다.

```
start = time.time()
for epoch in range(num_epochs):
  for i, (image, label) in enumerate(train_loader):
    image = image.to(device)
    label = label.to(device)

    # Forward
    output = model(image)
    loss = criterion(output, label)

    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    loss_list.append(loss.item())

    if loss.item() < best_loss:
      best_loss = loss.item()
      torch.save(model.state_dict(), "model.pth")

    #if (i+1) % 1000 == 0:
    print("Epoch [{}/{}], Step[{}/{}], Loss:{:.4f}".format(epoch+1, num_epochs, i+1, total_step, loss.item()))

end = time.time()
print("\n>> Train takes {:.2f} minutes".format((end-start)/60))
```

4. 결과 및 토의

1. Loss graph



2. Valid accuracy

```
Accuracy of the last_model network on the 15600 valid images: 98.26923076923077 %
```

3. 걸린시간

```
Training takes 13.12minutes
```

Tesla T4 환경에서 13.12초 소요

4. Model parameter

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 102, 102]             320
       BatchNorm2d-2         [-1, 32, 102, 102]              64
              ReLU-3         [-1, 32, 102, 102]               0
           Dropout-4         [-1, 32, 102, 102]               0
         MaxPool2d-5           [-1, 32, 51, 51]               0
            Conv2d-6           [-1, 64, 26, 26]          51,264
       BatchNorm2d-7           [-1, 64, 26, 26]             128
              ReLU-8           [-1, 64, 26, 26]               0
           Dropout-9           [-1, 64, 26, 26]               0
        MaxPool2d-10           [-1, 64, 13, 13]               0
           Conv2d-11            [-1, 128, 8, 8]          73,856
      BatchNorm2d-12            [-1, 128, 8, 8]             256
             ReLU-13            [-1, 128, 8, 8]               0
          Dropout-14            [-1, 128, 8, 8]               0
        MaxPool2d-15            [-1, 128, 4, 4]               0
           Conv2d-16            [-1, 256, 6, 6]         295,168
      BatchNorm2d-17            [-1, 256, 6, 6]             512
             ReLU-18            [-1, 256, 6, 6]               0
          Dropout-19            [-1, 256, 6, 6]               0
        MaxPool2d-20            [-1, 256, 3, 3]               0
           Linear-21                  [-1, 256]         590,080
           Linear-22                  [-1, 128]          32,896
           Linear-23                   [-1, 52]           6,708
================================================================
Total params: 1,051,252
Trainable params: 1,051,252
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 12.77
Params size (MB): 4.01
Estimated Total Size (MB): 16.81
----------------------------------------------------------------
```

5. 참고문헌

- 인공지능개론 강의자료
- Batch_size와 learning_rate의 관계논문 : The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset , Ibrahem Kandel