# CNN(Convolutional Neural Networks)를 이용한 다중 폰트의 영문자 및 숫자문자 분류

딥러닝 개론 2022 프로젝트 보고서

20171218 김다빈

## 1. 과제 목표

CNN(Convolutional Neural Networks)를 이용한 다중 폰트의 영문자 및 숫자문자 분류

## 2. 배경 이론

CNN(Convolutional Neural Networks) :

서로 다른 폰트의 영문자와 숫자문자를 정확히 분류하는 작업은 각 알파벳과 숫자의 문자 기호의 중요한 특징을 추출하는 것이 중요한 과제다. 이에 따라 나는 이미지 학습에 특화된 합성곱 신경망 CNN을 사용하여 프로젝트를 진행하기로 결정했다.

CNN의 필터에 해당하는 Kernel은 입력 데이터로 들어온 이미지 픽셀정보를 좌측 상단에서 우측 하단까지 Stride 간격에 따라 슬라이딩하며 지역적 특징을 추출한다. 이후 Pooling Layer는 추출한 특징을 최대값이나 평균값으로 압축해 특징맵의 크기를 줄여 다음 레이어로 전달한다. 이 과정에서 덜 중요한 특징은 제외되어 연산량이 줄어드는 동시에 분류에 중요한 Feature를 강조하게 된다. 해당 프로젝트에서는 nn.MaxPool2d를 활용해 추출 특징을 최대값으로 압축하는 방식을 택했으며, 파라미터를 kernel_size=2로 설정하여 2*2 크기 선택 영역마다 Max-pooling 연산을 진행하도록 했다.

Padding은 image의 주위를 0으로 둘러싸는 과정으로, CNN 과정에서 이미지의 축소를 막고 테두리의 Pixel Data를 충분히 활용하기 위해 사용된다. 일반적으로 Kernel Size가 5일 때 Padding은 2, Kernel Size가 3일 때 Padding은 1로 설정한다. 해당 프로젝트에서는 nn.Conv2d()의 파라미터를 kernel_size=5, padding=2로 설정한 후 학습을 진행했다.

분류 단계에 해당하는 FC(Fully Connected Layer)는 이전 Layer 출력을 평탄화하여 단일 벡터로 변환한다. 이 프로젝트에서는 두 개의 FC가 사용되었는데, 첫번째 FC는 앞선 분석내용을 입력받아 정확한 Label을 예측하기 위해 가중치를 적용하는 역할을 수행하고, 두번째 FC는 각 Label의 최종 확률을 제공한다.

테스트 과정에서 중점적으로 다루어진 Dropout은 학습과정 중 지정된 비율만큼 무작위적으로 Layer과 Layer 사이의 연결을 끊어 깊은 구조의 CNN에서의 Overfitting 발생을 방지한다.

## 3. 과제 수행 방법

CNN의 기본 정확도를 높이기 위해 다양한 변형을 주어 테스트를 진행했다. 결과적으로 23 epochs에 9.63min 동안의 학습으로 Validation Set에 대한 96.8717%의 정확도를 얻은 모델을 완성했다. 해당 모델을 구현한 과정을 아래에 상세히 기록하였으며, 그 외에도 CNN 구조를 변형해 테스트를 진행하였으나, 결과값이 낮게 나타나거나 다른 문제가 발생한 경우에 대해서도 적어두었다.

---

첫번째로 torch를 비롯하여 프로젝트에 필요한 모듈을 임포트한 후, 제공된 데이터 압축파일 'Font_npy_100_train.zip', 'Font_npy_100_val.zip'을 '/content/gdrive/MyDrive/Project_dataset/' 경로에 저장한 후 압축을 해제했다.

```python
# import modules
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset
from torchvision import models, transforms

import matplotlib.pyplot as plt
import random
import time
import os

import numpy as np
```

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

```
Mounted at /content/gdrive
```

```python
[3] # 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/Project_dataset/Font_npy_100_train.zip
    !unzip /content/gdrive/MyDrive/Project_dataset/Font_npy_100_val.zip
    # !unzip /content/gdrive/MyDrive/Project_dataset/Font_npy_100_test.zip
```

```
  inflating: Font_npy_100_val/Sample015/img015-00742.npy
  inflating: Font_npy_100_val/Sample015/img015-00030.npy
  inflating: Font_npy_100_val/Sample015/img015-00346.npy
  inflating: Font_npy_100_val/Sample015/img015-00391.npy
  inflating: Font_npy_100_val/Sample015/img015-00150.npy
  inflating: Font_npy_100_val/Sample015/img015-00838.npy
```

이후 Train 시 할당받은 GPU 종류를 확인하기 위해 !nvidia-smi 셀을 출력했다. 아래의 모든 학습은 Tesla T4 GPU 환경에서 Seed를 42로 고정하여 진행했다.

```python
[4] # Check GPU Configuration
    # device configuration
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    !nvidia-smi
```

```
Wed Dec  7 14:10:54 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   43C    P0    26W /  70W |      3MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
[5]  # FIX SEED
     def fix_seed(seed):
         random.seed(seed)
         torch.manual_seed(seed)
         torch.cuda.manual_seed(seed)

     fix_seed(42)
```

주어진 MyDataset() 함수를 활용해 '/content' 경로에 저장된 'Font_npy_100_train', 'Font_npy_100_val', 'Font_npy_100_test'를 불러왔다. 각각의 데이터 크기는 순서대로 37232, 7800, 0으로 나타났다.

```
[6]  # load dataset
     import glob
     class MyDataset(Dataset):
         def __init__(self, npy_dir): # image file (*.npy) 들을 포함하고 있는 디렉토리 경로 npy_dir 로 받아야,
             self.dir_path = npy_dir
             self.to_tensor = transforms.ToTensor()

             # all npy path
             self.npy_path = glob.glob(os.path.join(npy_dir, '*','*.npy'))

         def __getitem__(self, index):
             # load data
             single_data_path = self.npy_path[index]
             data = np.load(single_data_path, allow_pickle=True)

             image = data[0]
             image = self.to_tensor(image)
             label = data[1]

             return (image, label)

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



     train_data = MyDataset("/content/Font_npy_100_train") # unzip 한 디렉토리 있는 path 그대로 넣어야, 디렉토리 옆
     valid_data = MyDataset("/content/Font_npy_100_val")
     test_data = MyDataset("/content/Font_npy_90_test")

     print(len(train_data))
     print(len(valid_data))
     print(len(test_data))
```

```
37232
7800
0
```

주어진 코드를 활용해 train_loader, valid_loader, test_loader를 정의하였으며, batch size는 50으로 설정하였다. 이에 따라 valid_loader의 image.shape는 torch.Size([50, 1, 100, 100]), label.shape는 torch.Size([50])로 나타났다. 이를 통해 주어진 흑백 이미지 데이터의 channel 값이 1이며, 100 * 100 의 크기를 가지고 있음을 다시 한번 확인하였다.

```
[7] # define dataloader
    batch_size = 50
    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)

    test_loader = torch.utils.data.DataLoader(dataset=test_data,
                                              batch_size=batch_size,
                                              shuffle=False)


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

```
    torch.Size([50, 1, 100, 100])
    torch.Size([50])
```

Hyperparameter는 아래와 같이 정의하였다. 주어진 데이터는 영문자와 숫자문자 중 대/소문자 구별이 어려운 [”c”, ”k”, ”l”, ”O”, ”p”, ”s”, ”v”, ”w”, ”x”, ”z”]가 제외되어 총 53개의 정답값을 가지고 있는데, 이에 따라 num_classes를 53으로 설정했다. 또한 앞서 살펴보았듯이, 주어진 흑백 이미지 데이터의 channel 값은 1이므로, in_channel을 1로 설정했다. max_pool_kernel와 learning_rate의 값은 각각 2, 0.001로 하였다.
num_epochs는 기본적으로 20으로 설정하고 테스트를 진행했으나, 코드 수정 시 학습시간이 10분 이내로 마치도록 조정하였다. 최종 학습 결과의 경우 epochs 값을 23으로 설정했다.

```
[8] num_classes = 53
    in_channel = 1
    max_pool_kernel = 2
    learning_rate = 0.001
    num_epochs = 20
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

# 모델 설계

CNN(Covolution Neural Networks) 신경망 아키텍처를 활용하였다. CNN의 Feature Extraction 영역은 합성곱층 Convolution Layer과 풀링층 Pooling Layer을 여러 겹 쌓는 형태로 구성하였으며, Classification 부분은 Fully Connected 학습방식을 사용해 이미지를 분류했다.
비교적 깊은 구조의 CNN을 구현하기 위해 Convolution Layer 5개와 Fully Connected Layer 2개를 사용하였다. 모든 nn.Conv2d()의 파라미터에 대해서는 kernel_size=5, stride=1, padding=2 로 설정하였다. 이를 통해 입력 데이터 외각에 0값을 2 pixel 만큼 채워넣고 (5, 5) 크기의 Kernel이 1 Pixel 간격으로 순회하며 합성곱 계산을 진행하도록 하였다.
또한 모든 Convolution Layer에 대해 nn.Conv2d() → nn.BatchNorm2d() →nn.ReLU() → nn.MaxPool2d() 구조로 연산을 진행하도록 하였다. Batch Normalization에 해당하는 nn.BatchNorm2d()의 경우, 네트워크 연산 결과가 원하는 방향의 분포대로 나오도록 하기 위해 핵심 연산을 진행하는 nn.Conv2d()의 바로 뒤에, 분포를 변형하는 Activation Function(nn.ReLU()) 앞에 적용해 정규화를 진행했다. Pooling 연산에 해당하는 nn.MaxPool2d()는 정규화가 마무리된 뒤에 적용하였다.

```python
class CNN(nn.Module):
    def __init__(self, num_classes=53):
        super(CNN, self).__init__()

        # 1*100*100
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=in_channel, out_channels=16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 16*50*50
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 32*25*25
        self.layer3 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 64*12*12
        self.layer4 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 128*6*6
        self.layer5 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 256*3*3
```

5개의 Convolution Layer의 in_channels, out_channels 할당값을 확인하면 알 수 있듯이, 채널의 변화는 1 → 16 → 32 → 64 → 128 → 256 순으로 이루어지도록 설정했다. 이에 따라서 각 Convolution Layer에 따른 Size의 변화는 다음과 같이 나타난다.

```
[12] model = CNN()
     images, labels = next(iter(train_loader))
     model.eval()
     output1 = model.forward1(images)
```

```
[13] output2 = model.forward2(output1)
     print(f"{output1.size()} --> {output2.size()}")

     torch.Size([50, 16, 50, 50]) --> torch.Size([50, 32, 25, 25])
```

```
[14] output3 = model.forward3(output2)
     print(f"{output2.size()} --> {output3.size()}")

     torch.Size([50, 32, 25, 25]) --> torch.Size([50, 64, 12, 12])
```

```
[15] output4 = model.forward4(output3)
     print(f"{output3.size()} --> {output4.size()}")

     torch.Size([50, 64, 12, 12]) --> torch.Size([50, 128, 6, 6])
```

```
[16] output5 = model.forward5(output4)
     print(f"{output4.size()} --> {output5.size()}")

     torch.Size([50, 128, 6, 6]) --> torch.Size([50, 256, 3, 3])
```

```
            self.dropout = nn.Dropout(p=0.5)
            self.fc1 = nn.Linear(in_features=256*3*3, out_features=1000)
            self.fc2 = nn.Linear(in_features=1000, out_features=num_classes) # num_classes = 53
```

이후로는 Dropout과 2개의 Fully Connected Layer를 설정했다. Dropout은 위해 학습과정에서 네트워크의 일부 뉴런을 무작위적으로 생략하는 기능으로, 깊은 구조의 CNN에서 쉽게 발생할 수 있는 Overfitting을 방지하기 위해 추가했다. 이러한 기능은 훈련 데이터에 의해 각 네트워크의 가중치들이 서로 동조화되는 현상을 방지하며, 결과적으로 합리적인 시간 내로 많은 양의 네트워크를 학습해 보다 선명한 Feature를 얻도록 한다.
Fully Connected Network로 인한 Size의 변화는 아래와 같이 나타난다.

```
[19] output6 = model.forward6(output5)
     print(f"{output5.size()} --> {output6.size()}")

     torch.Size([50, 256, 3, 3]) --> torch.Size([50, 1000])
```

```
[20] output7 = model.forward7(output6)
     print(f"{output6.size()} --> {output7.size()}")

     torch.Size([50, 1000]) --> torch.Size([50, 53])
```

모델 학습에 사용할 forward 함수에서는 CL(Convolution Layer)1 → CL2 → CL3 → CL4 → CL5 를 거친 후, Activation Function → Reshape → FC1 → Activation Function → Dropout → FC2 순서대로 연산을 진행하였다. 여기에서 Dropout의 위치는 많은 시행착오를 거치며 테스트를 진행한 후 정확도를 비교해 결정하였다. 학습에 사용한 코드는 아래와 같다.

```
[35] model = CNN().to(device)

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

```
[36] ######## Train ########

     import time
     from tqdm import tqdm
     total_step = len(train_loader)
     total_loss = []
     model.train()

     start = time.time()
     for epoch in range(num_epochs):
       epoch_loss = []
       for i, (img, label) in enumerate(train_loader):
         # Assign Tensors to Configures Devices (gpu)
         img, label = img.to(device), label.to(device)

         # Forward propagation
         outputs = model(img)

         # Get Loss, Compute Gradient, Update Parameters
         loss = criterion(outputs, label)
         optimizer.zero_grad()
         loss.backward()
         optimizer.step()

         epoch_loss.append(loss.detach().cpu().numpy())

         # Print Loss
         if i % 10000 == 0 or (i+1)==len(train_loader):
           print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, i+1, len(train_l
       total_loss.append(np.mean(epoch_loss))
       print(f"epoch{i} loss: {np.mean(epoch_loss)}")

     end = time.time()
     duration = end - start
     print("Training takes {:.2f}minutes".format(duration/60))
```

---

# 최종 학습 결과 : Tesla T4 환경 | 9.63min | Accuracy: 96.8717% (epochs=23)

<Improving neural networks by preventing co-adaptation of feature detectors, G. E. Hinton> 논문은 CNN에서 Dropout을 사용하는 가장 일반적인 방식으로 Output 직전, 각 Fully Connected Layer 뒤에 배치할 것을 제안한다.

<CNN을 사용한 공간 데이터 분류 방법, 오병우, 한국정보기술학회> 논문은 Keras 환경에서 CNN을 구현할 때 Flatten - ReLU - Dropout - Softmax 순서로 Dropout을 적용하여 Overfitting을 방지하였다.

나는 두 논문에서 제시한 구조를 참고하여 첫번째 Fully Connected Network 뒤에 Activation Function을 적용하고, 두번째 Fully Connected Network 연산이 진행되기 전에 Dropout을 적용했다. 이에 따라 CNN의 Classification 단계는 Activation Function → Reshape → FC1 → Activation Function → Dropout → FC2 순서대로 레이어 연산이 진행된다. Loss의 감소가 안정적으로 나타났으며, 정확도는 96.7817%로, 전체 테스트 결과 중 가장 높은 값이 나타났다.

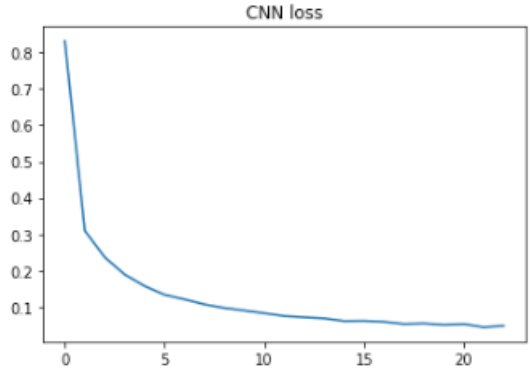
```
            self.dropout = nn.Dropout(p=0.5)
            self.fc1 = nn.Linear(in_features=256*3*3, out_features=1000)
            self.fc2 = nn.Linear(in_features=1000, out_features=num_classes) # num_classes = 53

        def forward(self, x):
            x = self.layer1(x)
            x = self.layer2(x)
            x = self.layer3(x)
            x = self.layer4(x)
            x = self.layer5(x)

            x = F.relu(x)
            x = x.reshape(x.size(0),-1)

            x = F.relu(self.fc1(x))
            x = F.dropout(x, training=self.training)
            x = self.fc2(x)
            return x
```

```
Epoch [20/23], Step [1/745], Loss: 0.0062
Epoch [20/23], Step [745/745], Loss: 0.0012
epoch744 loss: 0.050162289291620255
Epoch [21/23], Step [1/745], Loss: 0.0250
Epoch [21/23], Step [745/745], Loss: 0.0454
epoch744 loss: 0.05243309587240219
Epoch [22/23], Step [1/745], Loss: 0.0647
Epoch [22/23], Step [745/745], Loss: 0.0003
epoch744 loss: 0.04385603219270706
Epoch [23/23], Step [1/745], Loss: 0.0634
Epoch [23/23], Step [745/745], Loss: 0.0621
epoch744 loss: 0.04771029204130173
Training takes 9.63minutes
```

```
valid_model = CNN().to(device)
valid_model.load_state_dict(torch.load('model.pth'))

valid_model.eval()
with torch.no_grad():
  correct = 0

  for image, label in valid_loader:
    image = image.to(device)
    label = label.to(device)
    out = valid_model(image)
    _, pred = torch.max(out.data, 1)
    correct += (pred == label).sum().item()

  print('Accuracy of the last_model network on the {} valid images: {} %'.\
       format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 96.87179487179488 %
```

```
[13] plt.plot(total_loss)
     plt.title("CNN loss")
     plt.show()
```



아래는 최종 'test_20171218.ipynb' 파일에서 valid_loader에 대해 진행한 평가 정확도 결과이다.

```
[12] import torch
     import torchvision
     import torchvision.transforms as transforms

     # Device Configuration
     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

     valid_model = CNN().to(device)
     valid_model.load_state_dict(torch.load('20171218.pth'))

     valid_model.eval()
     with torch.no_grad():
       correct = 0

       for image, label in valid_loader:
         image = image.to(device)
         label = label.to(device)
         out = valid_model(image)
         _, pred = torch.max(out.data, 1)
         correct += (pred == label).sum().item()

       print('Accuracy of the last_model network on the {} valid images: {} %'.\
            format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 96.87179487179488 %
```

---

# 테스트 과정 기록 (96.8717% 미만의 정확도)

1) 테스트 1 : CNN without Dropout
   : Tesla T4 환경 | 8.42min | Accuracy: 95.9872% (epochs=20)

Dropout을 적용하지 않고 실습 수업에서 구현하였던 CNN 구조를 참고하여 모델을 만들었다. 총 5층의 CL(Convolutional Layer)을 사용하였으며, Feature Extraction이 완료된 이후로는 Activation Function → Reshape → FC1 → Activation Function → FC2 순으로 연산을 진행했다. 이 경우 epochs를 23으로 설정할 경우 학습시간이 10.0min을 초과하여 epochs를 20으로 조정하여 학습을 진행했다.

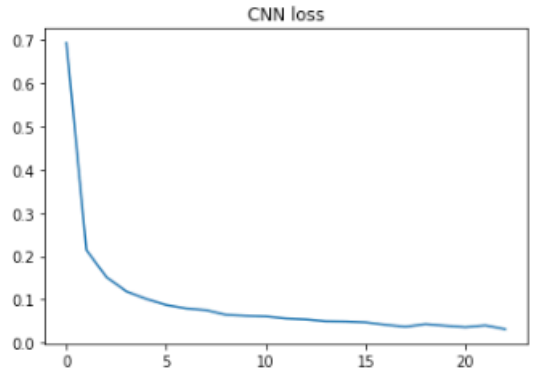
```
            self.fc1 = nn.Linear(in_features=256*3*3, out_features=1000)
            self.fc2 = nn.Linear(in_features=1000, out_features=num_classes) # num_classes = 53

        def forward(self, x):
            x = self.layer1(x)
            x = self.layer2(x)
            x = self.layer3(x)
            x = self.layer4(x)
            x = self.layer5(x)

            x = F.relu(x)
            x = x.reshape(x.size(0),-1)

            x = F.relu(self.fc1(x))
            x = self.fc2(x)
            return x
```

```
Epoch [18/20], Step [1/745], Loss: 0.0414
Epoch [18/20], Step [745/745], Loss: 0.0539
epoch744 loss: 0.04185490310192108
Epoch [19/20], Step [1/745], Loss: 0.0064
Epoch [19/20], Step [745/745], Loss: 0.0774
epoch744 loss: 0.039020635187625885
Epoch [20/20], Step [1/745], Loss: 0.0044
Epoch [20/20], Step [745/745], Loss: 0.0124
epoch744 loss: 0.04069037362933159
Training takes 8.42minutes
```

```
[22] model.eval()
     with torch.no_grad():
       correct = 0

       for image, label in valid_loader:
         image = image.to(device)
         label = label.to(device)
         out = model(image)
         _, pred = torch.max(out.data, 1)
         correct += (pred == label).sum().item()

       print('Accuracy of the last_model network on the {} valid images: {} %'.\
            format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 95.98717948717949 %
```

```
[15] plt.plot(total_loss)
     plt.title("CNN loss")
     plt.show()
```




2) 테스트 2 : Dropout을 2회 적용한 경우

: Tesla T4 환경 | 9.63min | Accuracy: 96.4745% (epochs=24)

<Dense Layer Dropout Based CNN Architecture for Automatic Modulation Classification, P Dileep; Dibyaiyoti Das; Prabin Kumar Bora, IEEE> 논문을 참고한 결과, Dropout을 각 Fully Connected Layer(Dense Layer) 뒤에 적용한 경우 보다 높은 정확도를 보였다. 해당 논문에서는 Input - Conv2d - PReLU - Conv2d - PReLU - Flatten - Dense+PReLU - Dropout - Dense+PReLU - Dropout - Dense+Softmax 순서대로 Dropout을 Feature Extraction 단계에 적용했는데, 나는 이를 참고하여 ReLU - Dropout - FC1+ReLU - Dropout - FC2+ReLU 순으로 Feature Extraction 단계를 구성했다.

그 결과 안정적으로 학습이 이루어졌으나, 최종 학습보다 오랜 시간(9.63min)이 소요되었으며, 정확도 또한 보다 낮은 96.4745%가 나타났다.

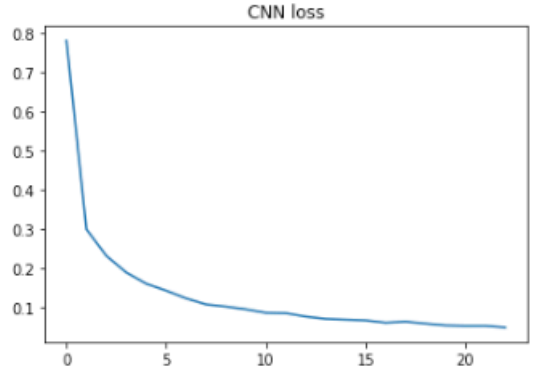
```
            self.dropout1 = nn.Dropout(p=0.25)
            self.dropout2 = nn.Dropout(p=0.5)

            self.fc1 = nn.Linear(in_features=256*3*3, out_features=1000)
            self.fc2 = nn.Linear(in_features=1000, out_features=num_classes) # num_classes = 53

        def forward(self, x):
            x = self.layer1(x)
            x = self.layer2(x)
            x = self.layer3(x)
            x = self.layer4(x)
            x = self.layer5(x)

            x = F.relu(x)
            x = self.dropout1(x)
            x = x.reshape(x.size(0),-1)
            x = F.relu(self.fc1(x))
            x = self.dropout2(x)
            x = self.fc2(x)
            return x
```

```
Epoch [20/23], Step [1/745], Loss: 0.0187
Epoch [20/23], Step [745/745], Loss: 0.0320
epoch744 loss: 0.05354810878634453
Epoch [21/23], Step [1/745], Loss: 0.0351
Epoch [21/23], Step [745/745], Loss: 0.0054
epoch744 loss: 0.05246822535991669
Epoch [22/23], Step [1/745], Loss: 0.0777
Epoch [22/23], Step [745/745], Loss: 0.0659
epoch744 loss: 0.05253403261303902
Epoch [23/23], Step [1/745], Loss: 0.0077
Epoch [23/23], Step [745/745], Loss: 0.0784
epoch744 loss: 0.04829147830605507
Training takes 9.63minutes
```

```
[30] model.eval()
     with torch.no_grad():
       correct = 0

       for image, label in valid_loader:
         image = image.to(device)
         label = label.to(device)
         out = model(image)
         _, pred = torch.max(out.data, 1)
         correct += (pred == label).sum().item()

       print('Accuracy of the last_model network on the {} valid images: {} %'.\
            format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 96.47435897435898 %
```

```
[31] plt.plot(total_loss)
     plt.title("CNN loss")
     plt.show()
```



3) 테스트 3 : Convolutional Layer에 Dropout(p=0.5)를 적용한 경우
: Tesla T4 환경 | 9.62min | Accuracy: 96.7564% (epochs=24)

<Analysis on the Dropout Effect in Convolutional Neural Networks, Sungheon Park; Nojun Kwak, Soul National University> 논문의 내용에서 Convolution - Batch Normalization - Activation Function - Dropout - Pooling 순서대로 네트워크를 구성하는 것이 적절하다는 내용을 참고하였다. 이에 따라 layer1부터 layer5까지 각각의 Covolution Layer를 nn.Conv2d() → nn.BatchNorm2d() → nn.ReLU() → nn.Dropout() → nn.MaxPool2d() 순서대로 구성했다. 또한 최종 코드와 동일하게 두 개의 Fully Connected Layer 사이에 하나의 Dropout을 배치했다. 해당 테스트 과정에서는 모든 nn.Dropout()의 파라미터는 p=0.5로 설정했다.
23 epochs에서 9.62min의 학습시간에 비교적 높은 정확도(96.7563%)가 나타났으나, loss가 안정적으로 감소되지 않아 사용을 보류했다.

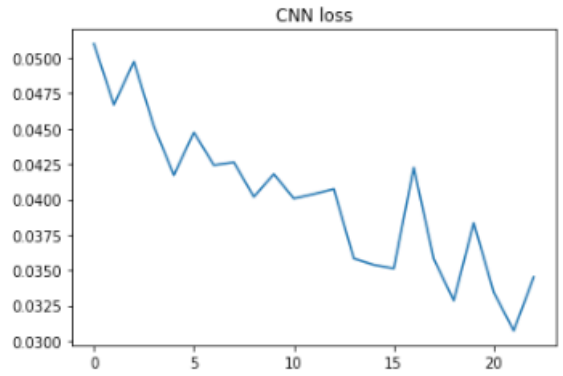
```
        # 1*100*100
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=in_channel, out_channels=16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=16),
            nn.ReLU(),
            nn.Dropout(p=0.5, training=self.training),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 16*50*50
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            nn.Dropout(p=0.5, training=self.training),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        # 32*25*25
        self.layer3 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
            nn.Dropout(p=0.5, training=self.training),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
```

```
Epoch [20/23], Step [745/745], Loss: 0.1043
epoch744 loss: 0.038346726447343826
Epoch [21/23], Step [1/745], Loss: 0.2225
Epoch [21/23], Step [745/745], Loss: 0.0049
epoch744 loss: 0.033442411571741104
Epoch [22/23], Step [1/745], Loss: 0.0091
Epoch [22/23], Step [745/745], Loss: 0.0585
epoch744 loss: 0.03072836622595787
Epoch [23/23], Step [1/745], Loss: 0.0148
Epoch [23/23], Step [745/745], Loss: 0.0046
epoch744 loss: 0.03451189398765564
Training takes 9.62minutes
```

```
[35] model.eval()
     with torch.no_grad():
       correct = 0

       for image, label in valid_loader:
         image = image.to(device)
         label = label.to(device)
         out = model(image)
         _, pred = torch.max(out.data, 1)
         correct += (pred == label).sum().item()

       print('Accuracy of the last_model network on the {} valid images: {} %'.\
            format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 96.75641025641026 %
```

```
[36] plt.plot(total_loss)
     plt.title("CNN loss")
     plt.show()
```




4) 테스트 4 : Convolutional Layer에 Dropout(p=0.2)를 적용한 경우

<Analysis on the Dropout Effect in Convolutional Neural Networks, Sungheon Park; Nojun Kwak, Soul National University> 논문에는 Convolutional Layer에 p=0.1 혹은 0.2 값을 부여한 Dropout을 Activation Function 뒤에 적용하는 것이 효과적이라고 나타났다. 이 점을 참고하여 각 Convolutional Layer의 nn.ReLU() 뒤에 nn.Dropout(p=0.2)를 작성했다.

각각의 Convolution Layer에 Dropout(p=0.5)를 적용했을 때와는 달리 Loss가 안정적으로 감소하였으나, 23 epochs에서 9.81min의 학습시간에 95.5%의 정확도를 보여 비교적 낮은 성능을 보였다. 두 차례의 시도 끝에 Convolution Layer에 Dropout을 적용하는 것은 현 단계에서 부적절하다고 판단하였다.

추가 조사 결과, Dropout을 Convolution Layer에 적용하지 않는 것이 성능 증가에 크게 도움되지 않으며, 오히려 더 낮은 정확도를 도출한다는 프로그래머들의 의견을 확인하였다. CNN은 Convolution 연산을 통해 데이터의 Spatial Feature를 추출하기 때문에, 단순히 노드 몇 개를 지우는 것으로는 추출한 일부 Correlated

Information을 완벽하게 지울 수 없다는 것이 이유였다. 이에 따라 Convolution Layer에 Dropout을 적용하려는 시도를 멈추었다.

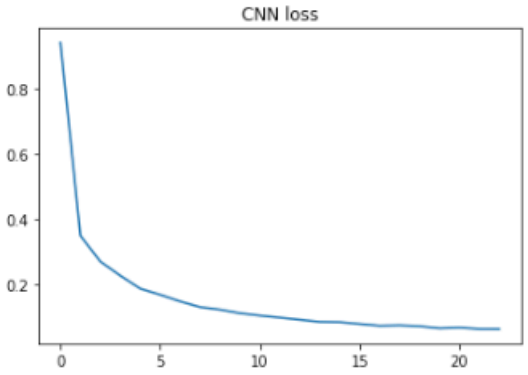
```
            # 1*100*100
            self.layer1 = nn.Sequential(
                nn.Conv2d(in_channels=in_channel, out_channels=16, kernel_size=5, stride=1, padding=2),
                nn.BatchNorm2d(num_features=16),
                nn.ReLU(),
                nn.Dropout(p=0.2),
                nn.MaxPool2d(kernel_size=max_pool_kernel)
            )
            # 16*50*50
            self.layer2 = nn.Sequential(
                nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=2),
                nn.BatchNorm2d(num_features=32),
                nn.ReLU(),
                nn.Dropout(p=0.2),
                nn.MaxPool2d(kernel_size=max_pool_kernel)
            )
            # 32*25*25
            self.layer3 = nn.Sequential(
                nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=1, padding=2),
                nn.BatchNorm2d(num_features=64),
                nn.ReLU(),
                nn.Dropout(p=0.2),
                nn.MaxPool2d(kernel_size=max_pool_kernel)
            )
```

```
Epoch [20/23], Step [745/745], Loss: 0.0070
epoch744 loss: 0.06310462951660156
Epoch [21/23], Step [1/745], Loss: 0.0504
Epoch [21/23], Step [745/745], Loss: 0.0738
epoch744 loss: 0.06508632749319077
Epoch [22/23], Step [1/745], Loss: 0.0948
Epoch [22/23], Step [745/745], Loss: 0.0886
epoch744 loss: 0.0612277053296566
Epoch [23/23], Step [1/745], Loss: 0.0505
Epoch [23/23], Step [745/745], Loss: 0.0509
epoch744 loss: 0.06095157191157341
Training takes 9.81minutes
```

```
model.eval()
with torch.no_grad():
  correct = 0

  for image, label in valid_loader:
    image = image.to(device)
    label = label.to(device)
    out = model(image)
    _, pred = torch.max(out.data, 1)
    correct += (pred == label).sum().item()

  print('Accuracy of the last_model network on the {} valid images: {} %'.\
       format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 95.5 %
```

```
[13] plt.plot(total_loss)
     plt.title("CNN loss")
     plt.show()
```




5) 테스트 5 : CRNN

추가로 테스트한 CRNN 구조는 가장 불안정한 Loss 감소와 가장 낮은 정확도를 보였다.

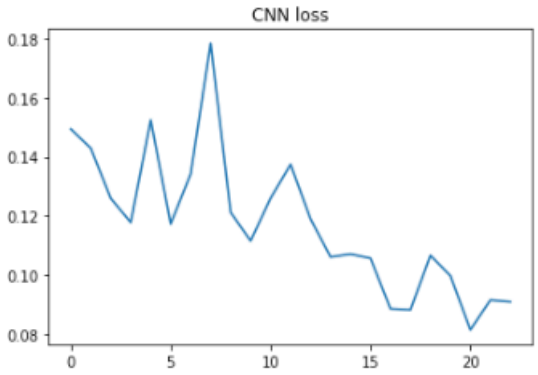
```
            self.rnn = nn.RNN(3*3, hidden_size, num_layers, batch_first=True)
            self.fc = nn.Linear(hidden_size, num_classes)

        def forward(self, x):
            x = self.layer1(x)
            x = self.layer2(x)
            x = self.layer3(x)
            x = self.layer4(x)
            x = self.layer5(x)
            x = F.relu(x)

            x = x.reshape(x.size(0), -1, 3*3)

            x = x.float()
            h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
            out, _  = self.rnn(x, h0)
            out = self.fc(out[:,-1,:])
            return out
```

```
Epoch [21/23], Step [1/745], Loss: 0.0242
Epoch [21/23], Step [745/745], Loss: 0.2516
epoch744 loss: 0.08139048516750336
Epoch [22/23], Step [1/745], Loss: 0.0591
Epoch [22/23], Step [745/745], Loss: 0.0198
epoch744 loss: 0.09147042781114578
Epoch [23/23], Step [1/745], Loss: 0.0802
Epoch [23/23], Step [745/745], Loss: 0.1357
epoch744 loss: 0.09087355434894562
Training takes 13.11minutes
```

```
model.eval()
with torch.no_grad():
  correct = 0

  for image, label in valid_loader:
    image = image.to(device)
    label = label.to(device)
    out = model(image)
    _, pred = torch.max(out.data, 1)
    correct += (pred == label).sum().item()

  print('Accuracy of the last_model network on the {} valid images: {} %'.\
        format(len(valid_data), 100 * correct / len(valid_data)))
```

```
Accuracy of the last_model network on the 7800 valid images: 94.87179487179488 %
```

```
[55] plt.plot(total_loss)
     plt.title("CNN loss")
     plt.show()
```




4. 결과 및 토의

이번 프로젝트에서 설계한 CNN 모델은 총 5개의 Convolutional Layer와 2개의 Fully Connected Layer를 사용하였다. 10분 내로 학습을 진행해야 하는 CNN 모델 중 비교적 깊은 구조를 가지고 있다. 따라서 나는 해당 모델의 학습에 과적합의 위험이 존재할 것이라고 예상했으며, 이에 따라 효율적인 학습 구조를 만들어 성능을 개선하기 위해 CNN에 Dropout을 활용하기로 결정했다. 해당 딥러닝 개론 2022 프로젝트 진행 중 풀어야 했던 가장 학생적인 과제는 CNN 내에 Dropout을 어느 위치에 적용하는 것이 가장 좋은 결과값을 만들어내는지에 대한 것이다.

테스트 결과에 따르면 CNN의 Feature Extraction 단계의 첫번째 Fully Connected Layer에 Activation Function을 적용한 후 1개의 Dropout을 적용하는 것이 가장 높은 정확도를 보였다. 반면 Feature Extraction 단계에 해당하는 Convolutional Layer에 Dropout을 적용하는 것은 부적절하게 나타났다. 파라미터 p의 값을 0.5로 설정한 경우, Convolutional Layer에 적용한 Dropout 또한 유의미한 결과를 보였지만, Classification 단계에 하나의 nn.Dropout(p=0.5)를 적용한 경우보다 Loss가 불안정하게 감소하는 모습을 보여 해당 모델을 사용하지 않기로 결정했다.

결과적으로 CNN으로 서로 다른 폰트의 영문자와 숫자문자 이미지를 학습해 분류하는 데에 Activation Function → Reshape → FC1 → Activation Function → Dropout → FC2 순서로 Classification 단계를 구성하는 것이 가장 높은 정확도를 보이는 것으로 나타났다.

5. 참고 문헌


a. 오병우, CNN을 사용한 공간 데이터 분류 방법 (한국정보기술학회, 2022)
b. P Dileep; Dibyaiyoti Das; Prabin Kumar Bora, Dense Layer Dropout Based CNN Architecture for Automatic Modulation Classification (IEEE, 2020)
c. Sungheon Park; Nojun Kwak, Analysis on the Dropout Effect in Convolutional Neural Networks (Springer International Publishing, 2017)