인공지능 (딥러닝) 개론 프로젝트

Project: Letter and Digit Set Classification with Custom Dataset

학번: 20192128

학과: 전자공학과

이름: 윤성철

제출일 2022.12.18

1. 과제 목표

이번 과제에 목표는 Letter와 Digit이 함께 있는 dataset을 분류하는 과제이다. 데이터 셋의 class는 Label 0~9: 숫자 digit, Label 10~52: 영문자 letter로 구성되어 있다.

2. 배경 이론

과제에서 수행해야하는 것은 Letter와 Digit으로 구성된 이미지 dataset을 classification하는 것이다. 357232개의 train data와 7800개의 valid data가 제공되다. 학습을 하기에는 충분한 양이라고 생각된다. 하지만 학습을 실행한 결과에서 overfitting이 발생한다면, data augmentation을 진행하여 data의 양을 늘려주어 overfitting을 개선할 수 있다. Overfitting을 개선할 수 있는 방법으로는 image augmentation, drop out, regularization, model을 간소화 하는 방법 등이 있다. 모델이 복잡할수록 train data에 과적합되는 경향이 있다.

3. 과제 수행 방법

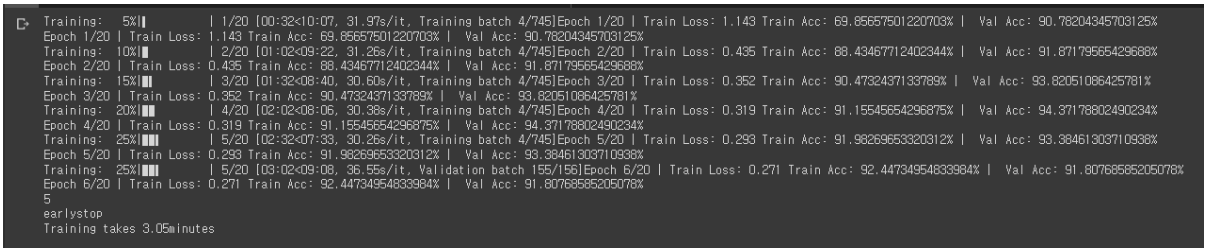
우선 과제를 처음 수행할 때 어떤 모델을 사용할 지를 생각하였다. 이론상으로 Image Classification에 적합한 모델은 CNN이라고 생각되지만, 강의에서 배웠던 RNN과 CRNN모두 실행해보았다. 그 결과 가장 높은 정확도와 overfitting이 적게 일어나는 모델은 CNN으로 확인되다. CNN 모델은 5개의 convolution layer와 2개의 dense layer로 구성하였다. hyper parameter를 다양하게 시도해 보았지만, 정확도와 학습 시간의 관점으로 분석해본 결과 가장 적합한 parameter는 다음과 같았다.

```
kernel_sizes=5
paddings=2
strides=1
max_pool_kernel=2
learning_rate=0.001
batch_size = 50
```

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

        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=in_chabnnel, out_channels=16, kernel_size=kernel_sizes, stride=1, padding=2),
            nn.BatchNorm2d(num_features=16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=kernel_sizes, stride=1, padding=2),
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )

        self.layer3 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=kernel_sizes, stride=1, padding=2),
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        self.layer4 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=kernel_sizes, stride=strides, padding=paddings),
            nn.BatchNorm2d(num_features=128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )
        self.layer5 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=kernel_sizes, stride=strides, padding=paddings),
            nn.BatchNorm2d(num_features=256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=max_pool_kernel)
        )

        self.fc1 = nn.Linear(in_features=256*3*3, out_features=416)
        self.fc2 = nn.Linear(in_features=416, out_features=52)
        self.fc3 = nn.Linear(in_features=128, out_features=52)
        self.dropout1 = nn.Dropout(0.15)
        self.dropout2 = nn.Dropout(0.5)

    def forward(self, x):
        x = self.layer1(x)
        x = self.dropout1(x)
        x = self.layer2(x)
        x = self.dropout1(x)
        x = self.layer3(x)
        x = self.dropout1(x)
```

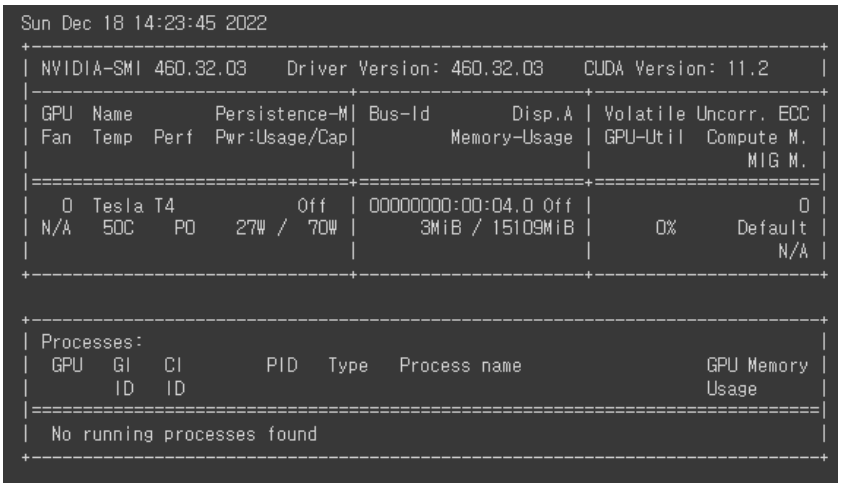
```
        x = self.layer4(x)
        x = self.dropout1(x)
        x = self.layer5(x)
        x = self.dropout1(x)

        x = F.relu(x)
        x=torch.flatten(x,1)
        x = F.relu(self.fc1(x))
        x = self.dropout2(x)
        x=self.fc2(x)
        #x = F.relu(self.fc2(x))
        #x = self.dropout2(x)
        #x = self.fc3(x)
        return x

model = CNN().to(device)
```

Dropout 비율을 너무 높게 설정할 경우 학습이 제대로 진행되지 않아 train_loss가 잘 감소하지 않았다. 그렇기에 반복적으로 실험해본 결과 가장 적합한 값이라고 생각되는 Conv layer의 dropout 비율은 0.15, Dense layer는 0.4로 설정하였다. Dropout을 추가하였기 때문에 학습 결과에서 validation accuracy가 train accuracy보다 높게 측정되는 것을 확인할 수 있다. 이것은 model.eval을 진행했기 때문에 evaluation에서는 nn.Dropout이 비활성화 되기 때문에 validation에서 더 높은 값이 먼저 측정되는 것이다. 또한, BatchNorm과 Gradient 연산도 생략된다. 학습이 계속 진행되면 overfitting이 발생할 수도 있기에 이것을 Early stopping을 구현하여 validation accuracy가 이전 epoch의 정확도보다 낮은 경우가 2번 발생하였을 때 학습을 종료하도록 설계하였다. 그리고 optimizer에 weight decay를 추가하여 regularization을 추가하였다.



4. 결과 및 토의

CNN모델을 구현하고 학습시킨 결과 정확도는 약 95%가 나왔다. validation은 training에 들어가지 않도록 따로 분리시켰지만 test data에서는 이것보다 낮은 정확도가 나올 것으로 예상된다. 사용한 GPU는 colab server의 것을 사용하였다.

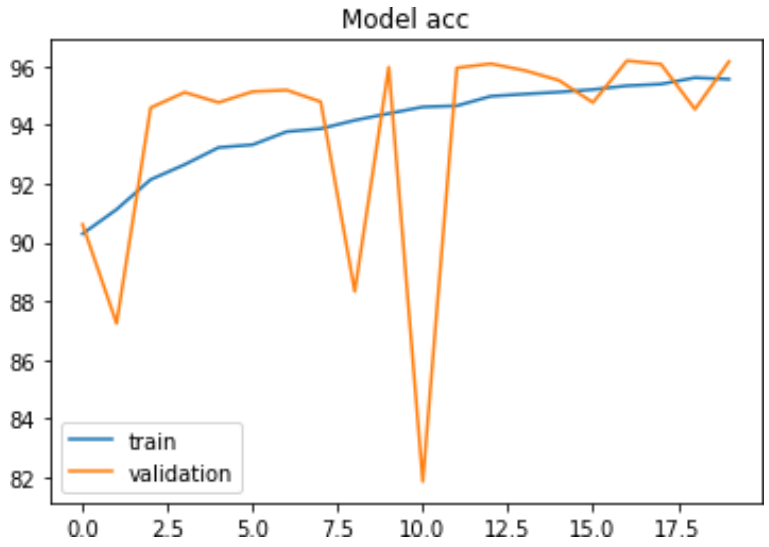


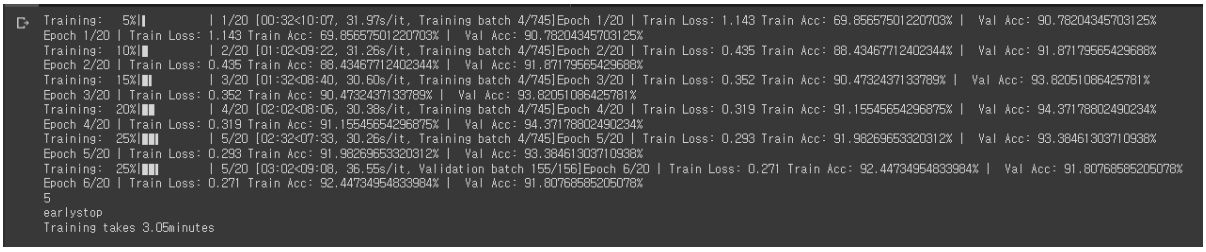
위에서 언급했던 것과 같이 결과에서 정확도는 train보다 validation에서 높게 측

정되👌다. 하지만 학습을 계속 진행시킴에 따라서 train도 점점 정확도가 올라가 validation에 가까워졌다. 하지만 이와 동시에 과적합이 발생하고 있어서 validation의 정확도는 96%를 기준으로 계속 변화하는 것을 확인 할 수 있👌다.

```
Training:   5%|▌         | 1/20 [00:29<09:26, 29.79s/it, Training batch 3/745]Epoch 1/20 | Train Loss: 0.362 Train Acc: 90.29597473144531% |  Val Acc: 90.61537933349615%
Epoch 1/20 | Train Loss: 0.362 Train Acc: 90.29597473144531% |  Val Acc: 90.61537933349615%
Training:  10%|▉         | 2/20 [01:00<09:01, 30.08s/it, Training batch 4/745]Epoch 2/20 | Train Loss: 0.318 Train Acc: 91.12590789794922% |  Val Acc: 87.24358367919922%
Epoch 2/20 | Train Loss: 0.318 Train Acc: 91.12590789794922% |  Val Acc: 87.24358367919922%
Training:  15%|█▍        | 3/20 [01:30<08:31, 30.07s/it, Training batch 4/745]Epoch 3/20 | Train Loss: 0.287 Train Acc: 92.14385223388672% |  Val Acc: 94.58973693847656%
Epoch 3/20 | Train Loss: 0.287 Train Acc: 92.14385223388672% |  Val Acc: 94.58973693847656%
Training:  20%|█▉        | 4/20 [01:59<07:57, 29.87s/it, Training batch 4/745]Epoch 4/20 | Train Loss: 0.262 Train Acc: 92.65416717529297% |  Val Acc: 95.11537933349615%
Epoch 4/20 | Train Loss: 0.262 Train Acc: 92.65416717529297% |  Val Acc: 95.11537933349615%
Training:  25%|██▍       | 5/20 [02:29<07:26, 29.75s/it, Training batch 3/745]Epoch 5/20 | Train Loss: 0.242 Train Acc: 93.23699188232422% |  Val Acc: 94.76922607421875%
Epoch 5/20 | Train Loss: 0.242 Train Acc: 93.23699188232422% |  Val Acc: 94.76922607421875%
4
earlystop
Training:  30%|██▉       | 6/20 [02:58<06:55, 29.67s/it, Training batch 4/745]Epoch 6/20 | Train Loss: 0.235 Train Acc: 93.33100128173828% |  Val Acc: 95.14102172851562%
Epoch 6/20 | Train Loss: 0.235 Train Acc: 93.33100128173828% |  Val Acc: 95.14102172851562%
5
earlystop
Training:  35%|███▍      | 7/20 [03:28<06:25, 29.63s/it, Training batch 4/745]Epoch 7/20 | Train Loss: 0.217 Train Acc: 93.77685546875% |  Val Acc: 95.19229888916016%
Epoch 7/20 | Train Loss: 0.217 Train Acc: 93.77685546875% |  Val Acc: 95.19229888916016%
6
earlystop
Training:  40%|███▉      | 8/20 [03:58<05:55, 29.65s/it, Training batch 3/745]Epoch 8/20 | Train Loss: 0.214 Train Acc: 93.87622833251953% |  Val Acc: 94.79486846923828%
Epoch 8/20 | Train Loss: 0.214 Train Acc: 93.87622833251953% |  Val Acc: 94.79486846923828%
Training:  45%|████▍     | 9/20 [04:27<05:26, 29.64s/it, Training batch 3/745]Epoch 9/20 | Train Loss: 0.203 Train Acc: 94.16361999511719% |  Val Acc: 88.33333587646484%
Epoch 9/20 | Train Loss: 0.203 Train Acc: 94.16361999511719% |  Val Acc: 88.33333587646484%
Training:  50%|████▉     | 10/20 [04:57<04:55, 29.67s/it, Training batch 4/745]Epoch 10/20 | Train Loss: 0.196 Train Acc: 94.39460754394531% |  Val Acc: 95.97435760498047%
Epoch 10/20 | Train Loss: 0.196 Train Acc: 94.39460754394531% |  Val Acc: 95.97435760498047%
Training:  55%|█████▍    | 11/20 [05:26<04:26, 29.61s/it, Training batch 4/745]Epoch 11/20 | Train Loss: 0.187 Train Acc: 94.61753082275391% |  Val Acc: 81.85897064208984%
Epoch 11/20 | Train Loss: 0.187 Train Acc: 94.61753082275391% |  Val Acc: 81.85897064208984%
Training:  60%|█████▉    | 12/20 [05:56<03:56, 29.61s/it, Training batch 4/745]Epoch 12/20 | Train Loss: 0.183 Train Acc: 94.65782165527344% |  Val Acc: 95.94871520996094%
Epoch 12/20 | Train Loss: 0.183 Train Acc: 94.65782165527344% |  Val Acc: 95.94871520996094%
Training:  65%|██████▍   | 13/20 [06:26<03:27, 29.62s/it, Training batch 4/745]Epoch 13/20 | Train Loss: 0.175 Train Acc: 94.98281097412109% |  Val Acc: 96.08973693847656%
Epoch 13/20 | Train Loss: 0.175 Train Acc: 94.98281097412109% |  Val Acc: 96.08973693847656%
Training:  70%|██████▉   | 14/20 [06:56<02:58, 29.69s/it, Training batch 3/745]Epoch 14/20 | Train Loss: 0.170 Train Acc: 95.05801391601562% |  Val Acc: 95.85897064208984%
Epoch 14/20 | Train Loss: 0.170 Train Acc: 95.05801391601562% |  Val Acc: 95.85897064208984%
Training:  75%|███████▍  | 15/20 [07:25<02:28, 29.67s/it, Training batch 4/745]Epoch 15/20 | Train Loss: 0.166 Train Acc: 95.12516021728516% |  Val Acc: 95.52564239501953%
Epoch 15/20 | Train Loss: 0.166 Train Acc: 95.12516021728516% |  Val Acc: 95.52564239501953%
Training:  80%|███████▉  | 16/20 [07:55<01:58, 29.66s/it, Training batch 4/745]Epoch 16/20 | Train Loss: 0.164 Train Acc: 95.21110534667969% |  Val Acc: 94.76922607421875%
Epoch 16/20 | Train Loss: 0.164 Train Acc: 95.21110534667969% |  Val Acc: 94.76922607421875%
Training:  85%|████████▍ | 17/20 [08:24<01:28, 29.65s/it, Training batch 4/745]Epoch 17/20 | Train Loss: 0.160 Train Acc: 95.34002685546875% |  Val Acc: 96.19230651855469%
Epoch 17/20 | Train Loss: 0.160 Train Acc: 95.34002685546875% |  Val Acc: 96.19230651855469%
Training:  90%|████████▉ | 18/20 [08:54<00:59, 29.62s/it, Training batch 3/745]Epoch 18/20 | Train Loss: 0.156 Train Acc: 95.39643096923828% |  Val Acc: 96.07691955566406%
Epoch 18/20 | Train Loss: 0.156 Train Acc: 95.39643096923828% |  Val Acc: 96.07691955566406%
Training:  95%|█████████▍| 19/20 [09:24<00:29, 29.59s/it, Training batch 4/745]Epoch 19/20 | Train Loss: 0.149 Train Acc: 95.61398315429688% |  Val Acc: 94.53845977783203%
Epoch 19/20 | Train Loss: 0.149 Train Acc: 95.61398315429688% |  Val Acc: 94.53845977783203%
Training: 100%|██████████| 20/20 [09:53<00:00, 29.67s/it, Validation batch 155/156]Epoch 20/20 | Train Loss: 0.151 Train Acc: 95.56295013427734% |  Val Acc: 96.16666412353516%
Epoch 20/20 | Train Loss: 0.151 Train Acc: 95.56295013427734% |  Val Acc: 96.16666412353516%
Training takes 9.89minutes
```

과적합이 발생한 구간에서 확인만 나오게 한 후 학습을 계속 진행시켰을 경우에 9.89분에 시간이 소요되면서 학습이 마무리 되👌다.





early stop에서 멈춘 경우에는 위와 같이 결과값이 출력되☝다.

이번 프로젝트에서는 두가지 부분에서 중점을 두☝다. 정확도를 높이는 과정과 과적합을 피하려는 부분이☝다. 정확도를 높이기 위하여 conv layer를 5개까지 쌓아서 정확도를 높였다. 하지만 현재 test data가 없기 때문에 validation data로 과적합을 판단하였는데, 그 과정에서 drop out, regularization, early stop, 모델을 간소화 하는 것까지 총 4가지 작업을 진행하였다. 프로젝트를 진행하는 과정에서 정확도나 overfitting이 발생하는 부분에서 만약 data가 더 많았다면 이러한 부분이 개선될 수도 있☝을 것이라 생각된다. 이 부분은 data augmentation을 통하여 data 개수를 늘릴 수 있다.

5. 참고 문헌

없습니다.