[Project]
MNIST Classification with Representation Learning

이름	학번	학년	E-mail
강준규	20192940	4	shinysky5166@soongsil.ac.kr

1 INTRODUCTION

1.1 INTRODUCTION

본 프로젝트의 목적은 Self-Supervised Learning(SSL) 및 Representation Learning 을 활용하여 주어진 한정된 데이터셋에 대한 효과적인 MNIST 분류 모델을 개발하는 것이다. 특히, Convolutional Layer 와 Focal Loss 를 사용함으로써 모델의 특징을 더욱 효과적으로 학습하고 데이터의 표현을 향상시킬 수 있도록 설계하였다.

2 MAIN STRUCTURE

2.1 HYPOTHESIS

2.1.1 Epoch

Epoch 를 통한 학습의 반복은 필수적이라고 생각된다. 또한 Epoch 수를 늘리면 늘릴수록 좋은 성능을 확인할 수 있을 것으로 생각된다. 하지만, 일정 수 이상의 Epoch 는 의미가 없거나 과적합의 문제가 나타날 수 있을 것이다.

2.1.2 다층 Layer

딥러닝 문제를 해결하는 데 있어서 층을 분화하고 그 사이에 활성 함수와 Dropout, Max Pooling 등의 Layer 를 활용한다면 모델의 안정성을 높이며 학습에 있어서도 더욱 특징을 잘 잡을 것으로 생각된다.

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1, padding='same')
        self.conv2 = nn.Conv2d(32, 64, 3, 1, padding='same')
        self.conv3 = nn.Conv2d(64, 128, 3, 1, padding='same')
        self.dropout = nn.Dropout2d(0.25)
        self.fc1 = nn.Linear(6272, 3000)    # 7 * 7 * 128 = 6272
        self.fc2 = nn.Linear(3000, 1000)
        self.fc3 = nn.Linear(1000, 10)

    def forward(self, x):
        x = self.conv1(x) # 28 * 28 * 32
        x = F.relu(x)
        x = F.max_pool2d(x, 2) # 14 * 14 * 32
        x = self.conv2(x) # 14 * 14 * 64
        x = F.relu(x)
        x = F.max_pool2d(x, 2) # 7 * 7 * 64
        x = self.dropout(x)
        x = self.conv3(x) # 7 * 7 * 128
        x = F.relu(x)
        x = self.dropout(x)
        x = torch.flatten(x, 1) # = 6272
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        output = F.log_softmax(x, dim=1)
        return output
```

- 참조 source : https://velog.io/@skarb4788/%EB%94%A5-%EB%9F%AC%EB%8B%9D-MNIST-%EB%8D%B0%EC%9D%B4%ED%84%B0PyTorch

우선, Convolutional Layer 를 통해 이미지의 지역적 특징을 추출하고자 하였고, Fully Connected Layer 를 통해 추출된 특징을 기반으로 숫자를 구별하는데 필요한 패턴을 학습하는 방식으로 Model 을 구성하였다. 활성 함수로는 ReLU 를 통해 불필요한 정보를 제거하고, 원래의 특징을 왜곡하지 않고자 하였다. Max Pooling 을 사용하여 크기가 커진 feature 의 공간 크기를 줄여 학습 데이터의 크기를 조절하였고, filter 가 64 개가 되는 시점에서부터 모델 학습의 안정성을 부여하기 위해 Dropout 을 사용하였다. FC Layer 를 통해 추출된 특징을 10 개로 축소하여 모델을 완성하였다. 마무리에 Log Softmax 를 사용하여 이후 Cross Entropy Loss 와의 연계와 안정적인 학습의 완료를 도모하였다.

2.1.3 Focal Loss

여러 Loss 를 찾아보던 중 Focal Loss 를 접하게 되었고, 이는 본 연구에서 매우 중요한 역할을 할 것으로 사료되었다. Focal Loss 는 클래스 불균형의 문제를 다루기 위해 연구된 Cross Entropy Loss 의 보완 버전이다. 적은 데이터셋으로 인해 발생할 수 있는 클래스 불균형 문제를 Focal Loss 를 이용해 더욱 안정적으로 손실을 계산하고 이를 학습에 적용하는 결과를 얻을 수 있을 것이다.

```
class FocalLoss(nn.Module):
    def __init__(self, gamma=2, alpha=1):
        super(FocalLoss, self).__init__()
        self.gamma = gamma
        self.alpha = alpha

    def forward(self, output, target):
        ce_loss = F.cross_entropy(output, target, reduction='none')
        pt = torch.exp(- ce_loss)
        focal_loss = self.alpha * (1 - pt)**self.gamma * ce_loss
        return torch.mean(focal_loss)

criterion = FocalLoss(gamma=2, alpha=1).to(DEVICE)
```

3 EXPERIMENTAL RESULTS

3.1 TEST ENVIRONMENT

3.1.1 Deep Learning Environment

Python	v3.10.12
Colab	부족한 개인 GPU 성능을 해결하기 위해 Colab 의 원격 GPU 환경을 사용.
Cuda	v11.3

3.1.2 Library

PyTorch	Numpy 와 유사한 코드 방식을 가지는 파이썬 환경 구축을 위한 라이브러리.

Numpy	학습하는 모델의 Tensor 들을 연산하기 위한 라이브러리.
PIL	모델의 Input 인 Image 들을 로컬로부터 자유롭게 가공할 수 있는 라이브러리.

3.1.3 Dataset

프로젝트 내 주어진 각 라벨 당 50 개의 Image 를 데이터셋으로 사용했다. 이를 4:1 비율로 Split 하여 각 라벨 당 40 장의 Train 데이터셋과 10 장의 Test 데이터셋으로 구분하여 실험에 활용하였다. 각 배치 사이즈는 100 장으로 고정하였다.

```
[3]  #data load
     BATCH_SIZE = 100
     dataset = ImageFolder(root='/content/drive/MyDrive/ML/data/train-50', transform=transforms.ToTensor())
     train_size = int(0.8 * len(dataset))
     test_size = len(dataset) - train_size
     train_dataset, test_dataset = random_split(dataset, [train_size, test_size])
     train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
     test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

3.1.4 Optimizer

```
learning_rate = 0.001
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
```

Optimizer 는 Adam 으로 고정하고, Learning Rate 역시 0.001 로 고정하였다. Optimizer 에 대해서는 다양한 실험을 진행하였는데, SGD 와 RAdam 보다 Adam 이 성능이 더욱 좋아 Adam 을 사용하게 되었다.

3.2 EPOCH

Epoch 를 설정하여 학습을 여러 차례 진행하는 것이 결과 향상에 도움이 된다는 사실을 증명하고자 하였다.

Epoch = 1	Test Accuracy: 17.00
Epoch = 20	Test Accuracy: 90.00
Epoch = 50	Test Accuracy: 83.00
Epoch = 100	Test Accuracy: 83.00

3.3 Basic Net vs My CNN

Sample 코드에 포함되어 있던 기본 CNN 과 내가 직접 만든 CNN 의 성능 비교이다.

CNN 의 비교를 위해서 손실함수는 Cross Entropy Loss 로 고정하였다.

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28*3, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)

    def forward(self,x):
        x = x.view(-1, 28*28*3)
        x = self.fc1(x)
        x = F.sigmoid(x)
        x = self.fc2(x)
        x = F.sigmoid(x)
        x = self.fc3(x)
        x = F.log_softmax(x, dim=1)
        return x
```

```python
class myCNN(nn.Module):
    def __init__(self):
        super(myCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1, padding='same')
        self.conv2 = nn.Conv2d(32, 64, 3, 1, padding='same')
        self.conv3 = nn.Conv2d(64, 128, 3, 1, padding='same')
        self.dropout = nn.Dropout2d(0.25)
        self.fc1 = nn.Linear(6272, 3000)    # 7 * 7 * 128 = 6272
        self.fc2 = nn.Linear(3000, 1000)
        self.fc3 = nn.Linear(1000, 10)

    def forward(self, x):
        x = self.conv1(x) # 28 * 28 * 32
        x = F.relu(x)
        x = F.max_pool2d(x, 2) # 14 * 14 * 32
        x = self.conv2(x) # 14 * 14 * 64
        x = F.relu(x)
        x = F.max_pool2d(x, 2) # 7 * 7 * 64
        x = self.dropout(x)
        x = self.conv3(x) # 7 * 7 * 128
        x = F.relu(x)
        x = self.dropout(x)
        x = torch.flatten(x, 1) # = 6272
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        output = F.log_softmax(x, dim=1)
        return output
```

- Basic Net vs My CNN

3.3.1 Basic Net

Loss 가 줄어드는 경향을 보면, 초기 2.3 의 Loss 에서 이후 2.2, 2.1 로 Loss 가 천천히 줄어드는 것을 볼 수 있다. 이후 Test Accuracy 도 80% 중반을 선회하는 성능을 확인할 수 있다.

```
Epoch:  1       loss = 2.318725586
Epoch:  2       loss = 2.237199306
Epoch:  3       loss = 2.124117851
Epoch:  4       loss = 1.997388124
Epoch:  5       loss = 1.842265368
Epoch:  6       loss = 1.664745331
Epoch:  7       loss = 1.492476344
Epoch:  8       loss = 1.332738876
Epoch:  9       loss = 1.187525630
Epoch:  10      loss = 1.050387025
```

```
Test Accuracy: 85.00
```

3.3.2 My CNN

여러 겹의 Convolutional Layer 를 통해 더욱 세밀한 특징을 분석하고, Max Pooling 과 Dropout 등의 기법으로 학습의 안정성을 더해주고 마지막으로 이를 FC Layer 를 통해 잘 축소한 결과 모델의 Loss 가 초반부터 빠르게 감소하는 것을 확인할 수 있다. 모델의 정확도도 90%를 넘기는 것을 확인할 수 있다.

```
Epoch:  1       loss = 2.128467798
Epoch:  2       loss = 1.359390736
Epoch:  3       loss = 0.727782428
Epoch:  4       loss = 0.542027533
Epoch:  5       loss = 0.402353168
Epoch:  6       loss = 0.277197868
Epoch:  7       loss = 0.254299998
Epoch:  8       loss = 0.207985580
Epoch:  9       loss = 0.126583084
Epoch:  10      loss = 0.124184355
```

```
Test Accuracy: 91.00
```

3.4 CROSS ENTROPY LOSS VS FOCAL LOSS

MNIST 분류 문제에서 자주 활용하는 Cross Entropy Loss 와 약간의 수정을 거친 Focal Loss 의 성능을 비교하고자 한다. 손실함수의 비교를 위해 CNN 은 직접 고안한 CNN 으로 고정한다.

3.4.1 Cross Entropy Loss

시작 Loss 는 2.2 부터 시작하며 Loss 가 0.5 대에 진입하는데 Epoch 가 5 회 소모되는 것을 확인할 수 있다. 모델의 정확도는 90%를 넘기는 것을 확인할 수 있다.

```
Epoch:  1       loss = 2.238246679
Epoch:  2       loss = 1.714121342
Epoch:  3       loss = 1.083068609
Epoch:  4       loss = 0.764802098
Epoch:  5       loss = 0.543398738
Epoch:  6       loss = 0.379976958
Epoch:  7       loss = 0.291823655
Epoch:  8       loss = 0.223063841
Epoch:  9       loss = 0.180269957
Epoch:  10      loss = 0.129449874
```

```
Test Accuracy: 92.00
```

3.4.2 Focal Loss

시작 Loss 의 값이 1.7 부터 시작하면서 모델이 처음부터 방향을 더욱 잘 잡는 것을 확인할 수 있다. Epoch 가 3 회 만에 Loss 가 0.5 로 자리잡는 것을 확인할 수 있고, 모델의 성능도 95%를 넘기는 것을 확인할 수 있다.

```
Epoch:  1       loss = 1.756518841
Epoch:  2       loss = 1.133012772
Epoch:  3       loss = 0.515998244
Epoch:  4       loss = 0.338577211
Epoch:  5       loss = 0.248151422
Epoch:  6       loss = 0.182623073
Epoch:  7       loss = 0.145656571
Epoch:  8       loss = 0.111119375
Epoch:  9       loss = 0.084925339
Epoch:  10      loss = 0.053869028
```

```
Test Accuracy: 96.00
```

3.4.3 Others

실험 과정에서 SGD, Radam 등의 Loss 도 사용하여 성능을 확인하였지만, 위의 두 Loss 보다는 더욱 낮은 성능을 보였다.

4 DISCUSSION AND CONCLUSION

본 연구에서는 직접 고안한 CNN 모델과 Focal Loss 를 활용하여 MNIST 데이터셋에 대한 효과적인 분류기를 개발하였다. 실험 결과, 모델의 초기 학습 단계에서부터 더 낮은 Loss 를 확인할 수 있었으며, 빠르게 안정적인 수준으로 수렴함을 확인하였다. 또한 모델의

정확도는 95%를 넘어 MNIST 데이터셋에 대한 효과적인 분류 성능을 보여주었다. 아래 몇 가지 포인트와 한계점을 짚어보며 연구를 마무리한다.

4.1 적정 Epoch 수의 필요성

Epoch 횟수를 지정하지 않았을 때보다 지정하였을 때 압도적인 성능의 차이를 확인하였다.

Epoch = 1	Test Accuracy: 17.00
Epoch = 20	Test Accuracy: 90.00
Epoch = 50	Test Accuracy: 83.00
Epoch = 100	Test Accuracy: 83.00

표에서 보는 것 같이, 데이터셋의 규모가 적기 때문에 일정 수 이상으로 Epoch 를 올리는 것은 의미가 희미해질 수는 있지만, 여러 번 학습을 반복하게 되는 Epoch 자체는 필수적이라는 사실을 알 수 있다.

일정 수 이상의 Epoch 를 진행하게 되면 다음 사진과 같이 학습을 진전할수록 loss 의 차이는 큰 변화를 가지지 않는다는 것을 알 수 있다.

```
Epoch:  30      loss = 0.097595587
Epoch:  31      loss = 0.089088552
Epoch:  32      loss = 0.081054889
Epoch:  33      loss = 0.073933057
Epoch:  34      loss = 0.068080798
Epoch:  35      loss = 0.062574282
Epoch:  36      loss = 0.058048528
Epoch:  37      loss = 0.053743437
Epoch:  38      loss = 0.049930155
Epoch:  39      loss = 0.046616722
Epoch:  40      loss = 0.043754205
Epoch:  41      loss = 0.040867604
Epoch:  42      loss = 0.038552653
Epoch:  43      loss = 0.036184099
Epoch:  44      loss = 0.034219965
Epoch:  45      loss = 0.032389674
Epoch:  46      loss = 0.030736240
Epoch:  47      loss = 0.029133072
Epoch:  48      loss = 0.027745828
Epoch:  49      loss = 0.026418287
Epoch:  50      loss = 0.025228942
```

Loss 가 기존에 설정한 Learning Rate 인 0.001 에 근접할수록 Loss 의 변화율은 더 적다. 때문에 Epoch 수를 무한정 늘리는 것은 학습의 성능에 더욱 큰 기여를 하기는 어렵다는 결론을 도출하였다.

4.2 CNN 아키텍쳐의 효율성

직접 고안한 CNN 모델은 다층의 Convolutional Layer 와 Pooling Layer 를 통해서 이미지의 특징을 더욱 세밀하게 추출하고, 이를 Fully Connected Layer 를 통해서 분류를 수행할 수 있도록 하였다.

더욱 층을 쌓아서 분류 문제를 해결할 수도 있었지만, 데이터셋이 다양하지 않고 문제가 복잡하지 않은 MNIST 이기 때문에 복잡한 모델을 구현하지 않았다. 여러 번의 실험을 통해 성능이 괜찮아지는 적절한 층의 모델을 고안할 수 있었다.

4.3 Focal Loss 의 활용

MNIST 분류 문제에서 Focal Loss 의 적용을 통해 적은 데이터셋에서 더욱 치명적으로 나타날 수 있는 클래스 간의 불균형 문제에도 대응을 하였다. 이를 통해 초기 학습 단계에서부터 안정적이고 효과적인 모델 학습을 가능하도록 하였다.

이는 아래 표의 초기 Loss 의 값을 비교하는 것으로 알 수 있다.

Epoch: 1 loss = 2.238246679 Epoch: 2 loss = 1.714121342 Epoch: 3 loss = 1.083068609 Epoch: 4 loss = 0.764802098 Epoch: 5 loss = 0.543398738 Epoch: 6 loss = 0.379976958 Epoch: 7 loss = 0.291823655 Epoch: 8 loss = 0.223063841 Epoch: 9 loss = 0.180269957 Epoch: 10 loss = 0.129449874	Epoch: 1 loss = 1.756518841 Epoch: 2 loss = 1.133012772 Epoch: 3 loss = 0.515998244 Epoch: 4 loss = 0.338577211 Epoch: 5 loss = 0.248151422 Epoch: 6 loss = 0.182623073 Epoch: 7 loss = 0.145656571 Epoch: 8 loss = 0.111119375 Epoch: 9 loss = 0.084925339 Epoch: 10 loss = 0.053869028

4.4 한계

4.4.1 Visualization

본 연구에서 모델의 가시화 작업은 수행하지 못했다. 향후 연구에서는 CAM, t-SNE 등을 통해 모델이 어떤 방식으로 판단을 내리는지에 대해 더 자세히 살펴볼 필요가 있다.

4.4.2 Dataset

주어진 데이터셋을 통해 한정된 결론을 도출하는 것으로 연구를 마무리 지었지만, 향후 연구에서는 더욱 다양한 데이터셋으로 확실한 분류 모델을 제안할 수 있다.

4.4.3 Activation Fuction

ReLU 이외의 다양한 활성 함수를 사용하여 향후 연구에서는 적절한 활성 함수의 검증도 고려하여야 한다.

5 REFERENCES

https://wikidocs.net/60324 - Softmax Regression

https://excelsior-cjh.tistory.com/180 - Convolutional Neural Networks

https://velog.io/@skarb4788/%EB%94%A5-%EB%9F%AC%EB%8B%9D-MNIST-%EB%8D%B0%EC%9D%B4%ED%84%B0PyTorch – MNIST Classification

https://gaussian37.github.io/dl-concept-focal_loss/ - Focal Loss