팀프로젝트

결과 보고서 #1

제 목: LeNet5 인공신경망 구성 및 분류기 구현

과 목 명:	딥러닝프로그래밍및실습
학 부:	AI융합학부
학번/이름:	20190501 / 김 태 균 20193125 / 김 건 수 20201802 / 이 강 룡 20211725 / 김 민 규
제 출 일:	2024년 4월 26일(금)
담당교수:	한 영 준

1. 과제 수행 내용

pytorch를 이용하여 LeNet5 인공신경망 분류기를 구현한 후 MNIST dataset을 이용하여 학습 수행.

(1) 아래 제시된 강의 자료를 참고하여 구조를 구현. 이때, 출력층에 주어지는 활성함수를 log softmax로 사용.

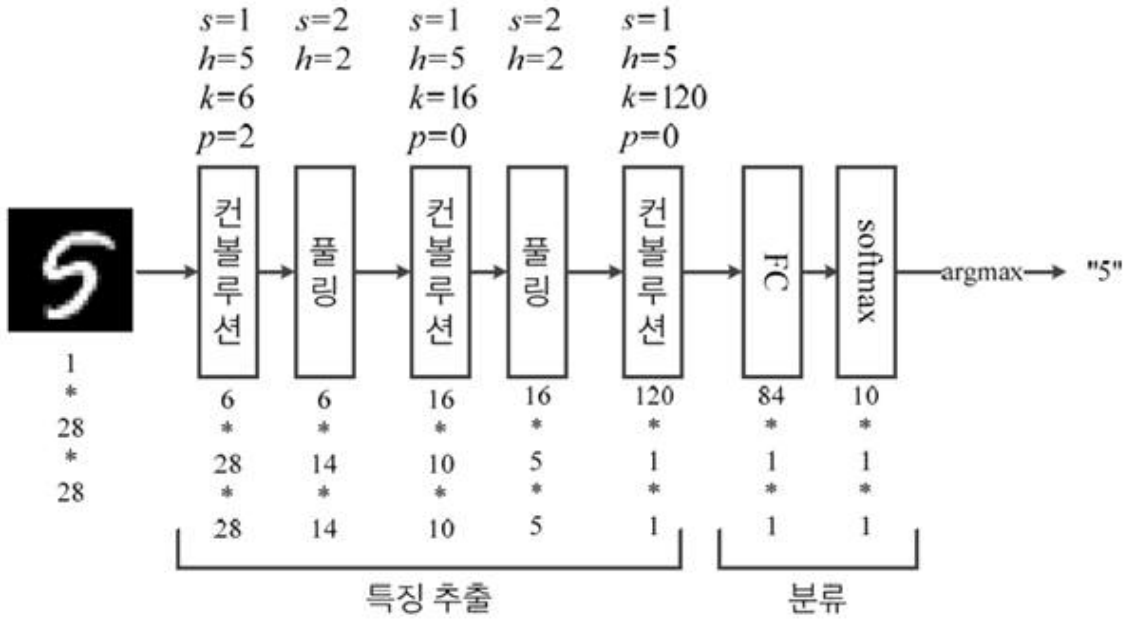


그림 4-19 LeNet-5 구조

```python
# 모델 정의
class LeNet5(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, padding=2, stride=1)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, padding=0, stride=1)
        self.conv3 = nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5, padding=0, stride=1)

        self.linear1 = nn.Linear(in_features=120, out_features=84)
        self.linear2 = nn.Linear(in_features=84,out_features=10)

        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.log_softmax = nn.LogSoftmax(dim=1)
        self.flatten = nn.Flatten()

```

```
65
66      def forward(self, x):
67          # Building Block 1
68          x = self.conv1(x)   # (6, 28, 28)
69          x = self.relu(x)
70          x = self.pool(x)    # (6, 14, 14)
71
72          # Building Block 2
73          x = self.conv2(x)   # (16, 10, 10)
74          x = self.relu(x)
75          x = self.pool(x)    # (16, 5, 5)
76
77          # Building Block 3
78          x = self.conv3(x)   # (120, 1, 1)
79          x = self.relu(x)
80
81          # Serialization for 2D image * channels
82          x = self.flatten(x) # (120, 1)
83
84          # Fully connected layers
85          x = self.linear1(x)
86          x = self.relu(x)
87
88          # output layer
89          x = self.linear2(x)
90          x = self.log_softmax(x)
91
92          return x
93
```

(2) MNIST dataset 학습 및 test dataset을 통한 accuracy 평가

```
104 # 모델 학습
105 def train(dataloader, model, loss_fn, optimizer):
106     start_time = time.time()
107
108     size = len(dataloader.dataset)
109     for batch, (X, y) in enumerate(dataloader):
110         X, y = X.to(device), y.to(device)
111
112         # 예측 오류 계산
113         pred = model(X)
114         loss = loss_fn(pred, y)
115
116         # 역전파
117         optimizer.zero_grad()
118         loss.backward()
119         optimizer.step()
120
121         if batch % 100 == 0:
122             loss, current = loss.item(), (batch + 1) * len(X)
123             print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
124
125     calculation_time = time.time() - start_time
126     print(f"\n* Calculation time on training set\n : {calculation_time:0.3f} sec\n")
127
128     return loss
129
```

```python
def test(dataloader, model, classes):
    model.eval()

    correct_predictions = 0
    total_samples = 0
    correct_samples = []

    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            outputs = model(X)

            _, predicted = torch.max(outputs, 1)
            total_samples += y.size(0)
            correct_predictions += (predicted == y).sum().item()

            # 정확하게 분류된 샘플 저장
            for i in range(len(predicted)):
                if predicted[i] == y[i]:
                    correct_samples.append((X[i], predicted[i]))

    accuracy = 100 * correct_predictions / total_samples
    print(f"* Accuracy on test dataset\n : {accuracy:.2f}%")
```

(3) LeNet5 인공신경망 분류기의 성능을 높이기 위한 방안

a. 하이퍼 파라미터 조정(batch size, learning rate, epoch)

accuracy (%)	batch	learning rate	epoch	사람	device
99.06	128	1e-1	100	김태균	cpu
98.98	256	1e-1	200	이강룡	cpu
98.94	64	1e-2	200	김태균	cpu
98.89	32	1e-2	200	이강룡	cpu
98.87	128	1e-2	200	이강룡	cpu
98.84	64	1e-2	100	김태균	cpu
98.84	256	1e-2	200	이강룡	cpu
98.62	256	1e-2	100	김건수	mps
98.56	128	1e-2	100	김태균	cpu
98.49	16	1e-1	100	김태균	cpu
98.41	128	1e-1	200	이강룡	cpu
98.37	32	1e-3	100	김태균	cpu
98.33	256	1e-2	100	김태균	cpu
97.96	64	1e-3	100	김태균	cpu
95.27	32	0.001	15	김민규	cpu
90.33	32	1e-4	100	김태균	cpu
81.16	64	1e-4	100	김태균	cpu

b. cost function, optimizer, activation function의 변경

```
# loss_fn = nn.CrossEntropyLoss()
# 주로 log softmax 출력을 사용하는 다중 클래스 분류에서 사용됨.
loss_fn = nn.NLLLoss(weight=None, ignore_index=-100, reduction='mean')
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(
        in_channels=1, out_channels=6, kernel_size=5, padding=2, stride=1)
    self.conv2 = nn.Conv2d(
        in_channels=6, out_channels=16, kernel_size=5, padding=0, stride=1)
    self.conv3 = nn.Conv2d(
        in_channels=16, out_channels=120, kernel_size=5, padding=0, stride=1)

    self.linear1 = nn.Linear(in_features=120, out_features=84)
    self.linear2 = nn.Linear(in_features=84, out_features=10)

    self.relu = nn.ReLU()
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    self.log_softmax = nn.LogSoftmax(dim=1)
    self.flatten = nn.Flatten()
```

(4) 소스 코드 내 주요 부분

위에서 제시한 모델의 구현과, 이를 기반으로 하이퍼 파라미터의 조정이 성능에 가장 큰 영향을 미치는 주요한 부분이다.

```
27  batch_size = 64
28
29  train_dataloader = DataLoader(training_data, batch_size=batch_size)
30  test_dataloader = DataLoader(test_data, batch_size=batch_size)
```

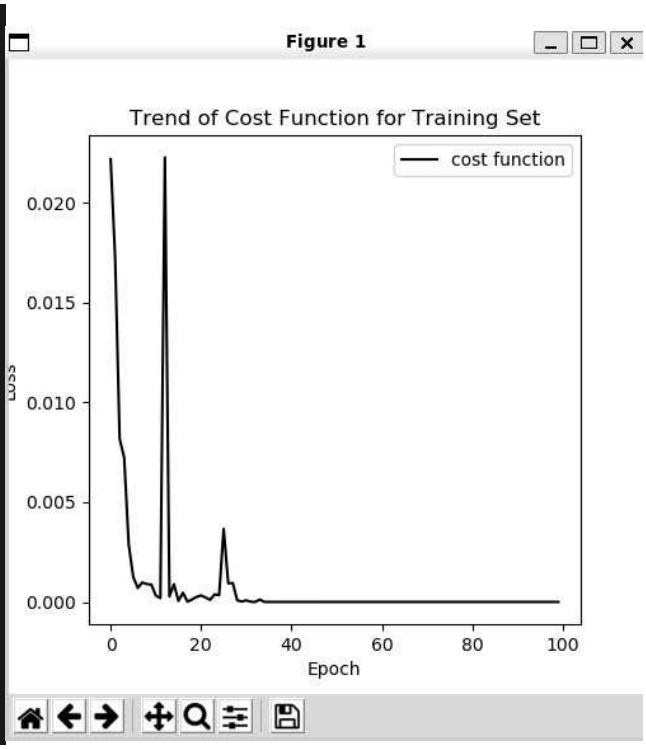
```
 99  # 모델 매개변수 최적화
100  loss_fn = nn.CrossEntropyLoss()
101  optimizer = torch.optim.SGD(model.parameters(), lr=1e-1)
```

```
212  # 에폭에 따른 결과
213  error_train = []
214  epochs = 100
215  for t in range(epochs):
216      print(f"Epoch {t+1}\n-------------------------------")
217      error = train(train_dataloader, model, loss_fn, optimizer)
218      error_train.append(error)
```

(5) 학습 중에 1 epoch 단위 training set에 대한 cost function 값과 수행시간 출력 및 학습 종료 후 cost function 추세 시각화

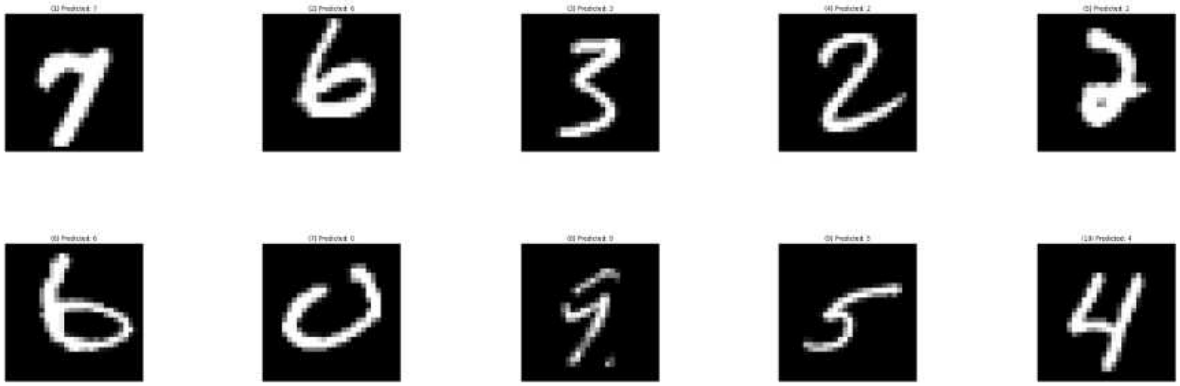
파라미터 batch size=64, learning rate=1e-1, epoch=100로 학습 및 시각화 결과 예시

```
################################

Epoch 1
-------------------------------
loss: 2.311962  [   64/60000]
loss: 2.056993  [ 6464/60000]
loss: 0.353678  [12864/60000]
loss: 0.367722  [19264/60000]
loss: 0.146474  [25664/60000]
loss: 0.117959  [32064/60000]
loss: 0.101968  [38464/60000]
loss: 0.123289  [44864/60000]
loss: 0.163022  [51264/60000]
loss: 0.128031  [57664/60000]

* Calculation time on training set
 : 9.102 sec

Epoch 2
-------------------------------
```

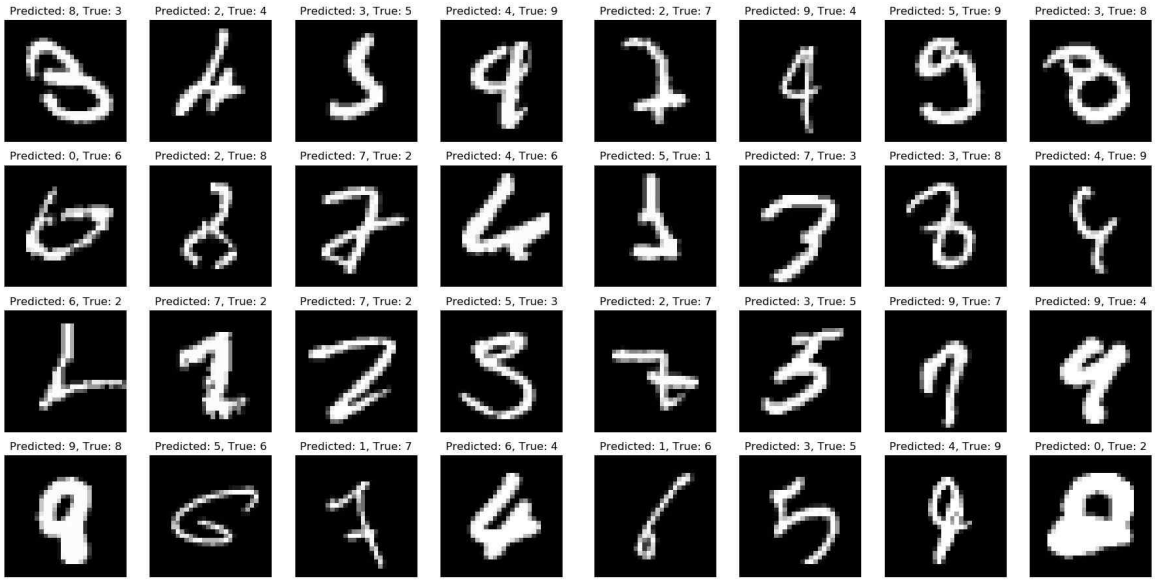


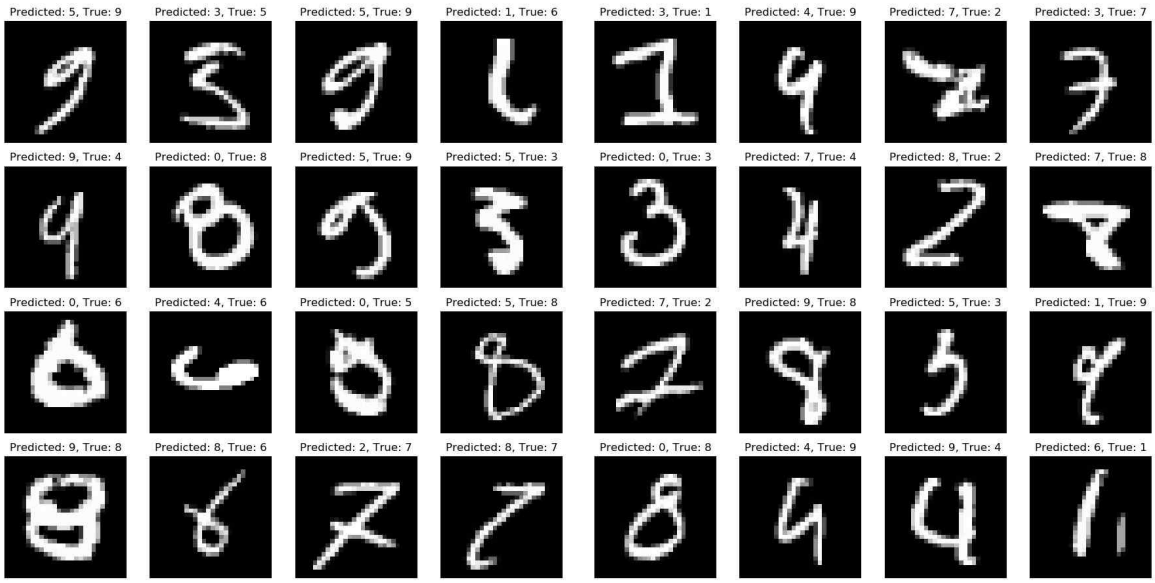
(6) test dataset에 대한 성능 출력 및 정확하게 분류한 10개의 random sample 출력

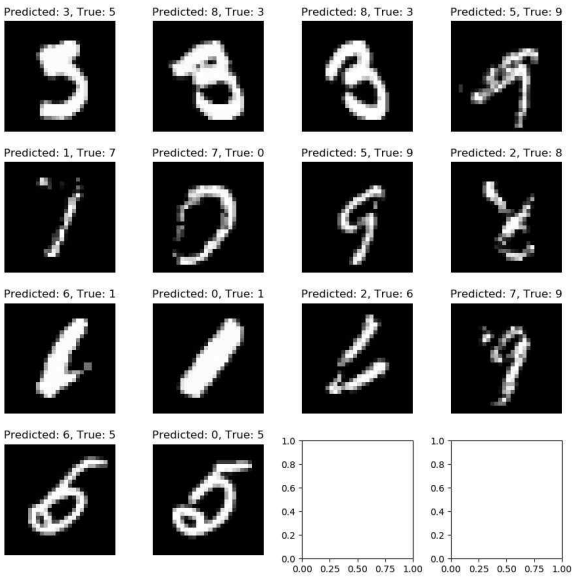
```
* Accuracy on test dataset
 : 99.22%
```



(7) test dataset 중에서 구현한 LeNet5 인공신경망 분류기가 잘못 분류한 모든 샘플 출력







총 78개(0.78%)

2. 결과 분석

(1) CrossEntropyLoss & SGD 함수 사용

CrossEntropyLoss & SGD

accuracy (%)	batch	learning rate	epoch	사람	device
99.22	64	1e-1	100	김태균	cpu
99.16	64	1e-1	200	이강룡	cpu
99.12	32	1e-2	100	김태균	cpu
99.06	128	1e-1	100	김태균	cpu
98.98	256	1e-1	200	이강룡	cpu
98.94	64	1e-2	200	김태균	cpu
98.89	32	1e-2	200	이강룡	cpu
98.87	128	1e-2	200	이강룡	cpu
98.84	64	1e-2	100	김태균	cpu
98.84	256	1e-2	200	이강룡	cpu
98.56	128	1e-2	100	김태균	cpu
98.49	16	1e-1	100	김태균	cpu
98.41	128	1e-1	200	이강룡	cpu
98.37	32	1e-3	100	김태균	cpu
98.33	256	1e-2	100	김태균	cpu
97.96	64	1e-3	100	김태균	cpu
90.33	32	1e-4	100	김태균	cpu
81.16	64	1e-4	100	김태균	cpu

학습률을 1e-1(=0.1) ~ 1e-4(=0.0001) 사이의 값으로 설정하여 성능을 측정하였을 때, 학습률이 커질수록, 즉 1e-1에 가까울수록 좋은 결과를 얻었다. 또한 batch size와 epoch 수는 성능에 크게 영향을 주지 않았다. device=cpu, batch size=64기준, 1 epoch 당 약 9~11초, 100 epoch 당 15~18분이 훈련에 사용되었다.

(2) NLLLoss & Adam 함수 사용

NLLLoss & Adam

accuracy (%)	batch	learning rate	epoch	사람	device
99.21	64	0.1	100	김건수	mps
99.16	64	0.001	100	김건수	mps
99.10	256	0.001	100	김건수	mps
99.05	128	0.001	100	김건수	mps
99.04	128	0.1	100	김건수	mps
98.81	256	0.0001	100	김민규	cpu
98.78	256	0.001	200	김건수	mps
98.77	64	0.0001	100	김민규	cpu
98.64	128	0.0001	100	김민규	cpu
98.24	128	0.01	100	김민규	cpu
98.19	256	0.01	100	김민규	cpu
92.15	64	0.01	100	김민규	cpu
11.35	256	0.1	100	김건수	mps

학습률이 클수록(0.1에 가까울수록) 배치가 작을 때, 학습률이 작을수록(0.0001에 가까울수록) 배치가 클 때 성능이 좋은 결과를 얻었다. 마찬가지로 epoch의 수는 성능에 크게 영향을 미치지 않아 100으로 실험을 하였다. device=mps, batch size=64 기준 1 epoch 당 약 7~9초, 100 epoch 당 11~14분이 훈련에 사용되었다.

3. 결론

본 프로젝트에서는 pytorch를 활용하여 LeNet5 인공신경망 분류기를 구현하고 MNIST dataset을 이용해 모델을 학습시켜 성능을 평가하였다.

첫 번째로, 서로 다른 목적 함수와 최적화 함수를 이용한 성능 실험에서 NLLLoss와 Adam 조합이 학습이 빠르고 효과적이었지만, 전체적인 성능은 CrossEntropyLoss와 SGD 조합이 MNIST dataset에 대해 가장 효과적이었다.

두 번째로, 하이퍼 파라미터의 조정을 통한 성능 실험에서 CrossEntropyLoss와 SGD 조합일 때는 학습률이 1e-1(=0.1)에 가까울수록 가장 효과적이었고, NLLLoss와 Adam 조합에서는 학습률이 클 때는 배치 크기가 작을 때, 학습률이 작을 때는 배치 크기가 클 때 효과적이었다.

이러한 실험을 통해 test dataset에 대해 최대 99.22%의 성능을 확인하고, 잘못 분류한 샘플을 출력하는 결과를 확인할 수 있다.

4. 참고문헌

(1) 오일석, 2017, 기계학습, 한빛아카데미
(2) LeNet5 구현: https://github.com/ChawDoe/LeNet5-MNIST-PyTorch
(3) pytorch 신경망 : https://9bow.github.io/PyTorch-tutorials-kr-0.3.1/beginner/blitz/neural_networks_tutorial.html
(4) 하이퍼 파라미터 조정: https://deep-learning-study.tistory.com/368
(5) LogSoftmax : https://velog.io/@olxtar/nn.LogSoftmax%EC%99%80-nn.NLLLoss