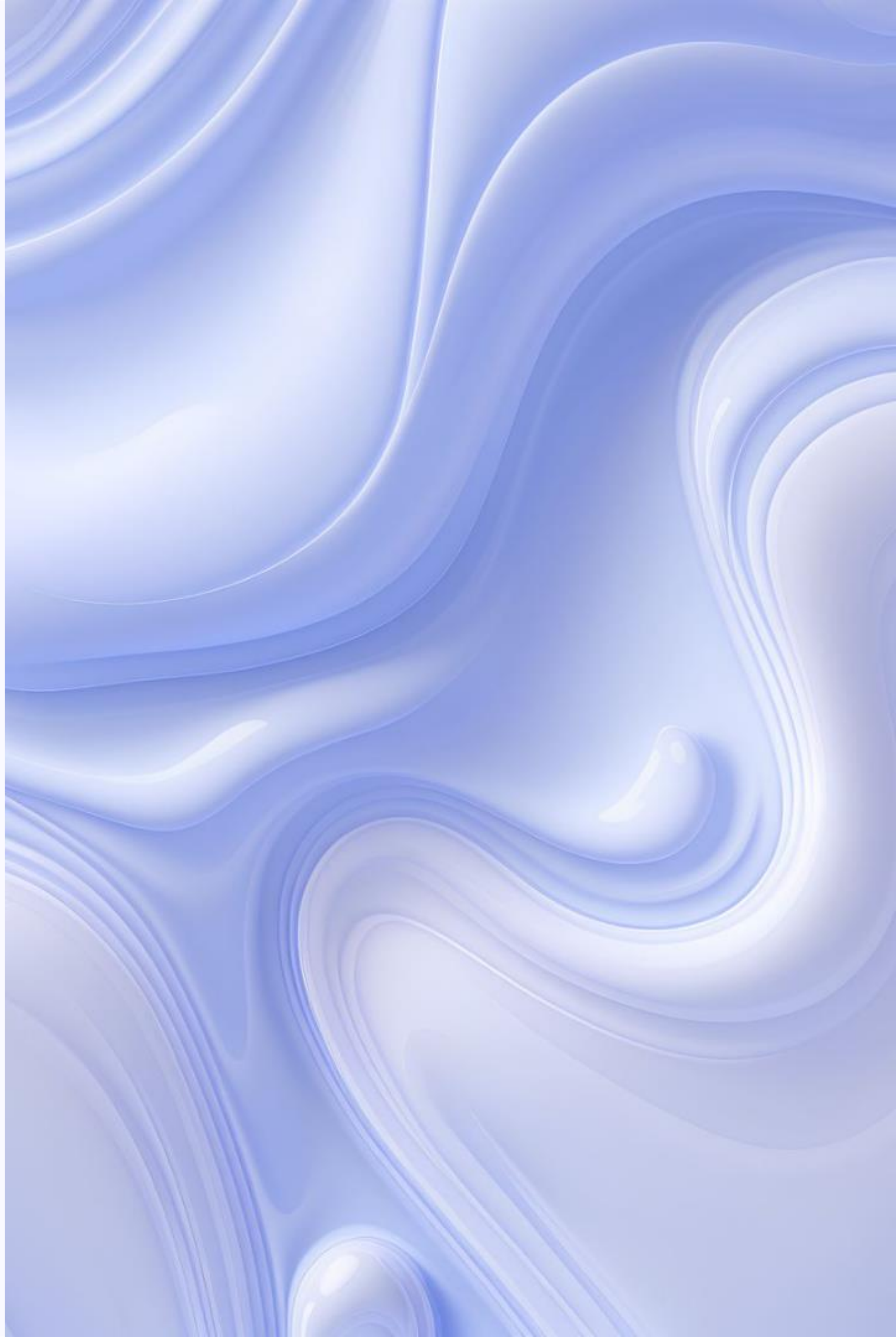
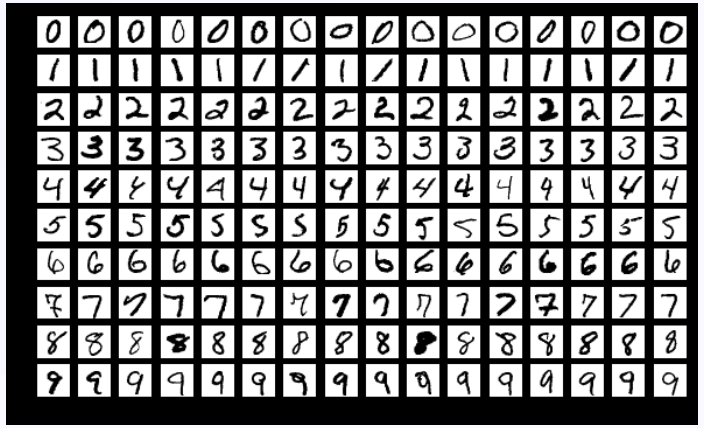
物件導向 期末專題
PyTorch

MNIST 手寫數字辨識

11111245 彭祥瑀 11111244 張景勛 11111247鄭揚融

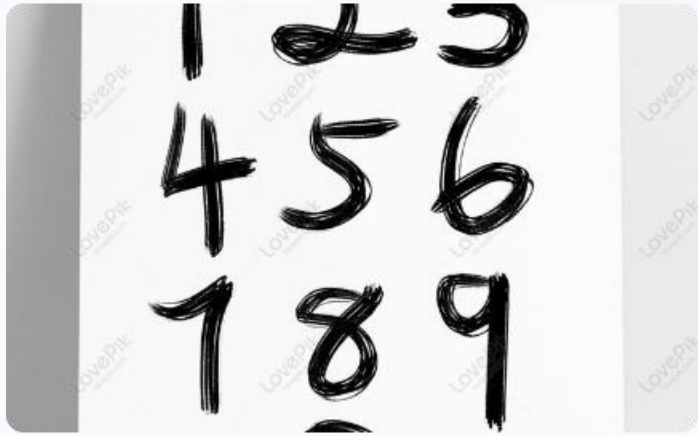


MNIST 資料集簡介



28x28像素解析度的灰度手寫數字

數據集包含 60,000 個訓練圖像
和 10,000 個測試圖像



手寫數字識別的經典問題

幾乎所有深度學習框架、CNN系
系統和圖像分類器都支持它

MNIST 是在 1998 年出現的手寫數字辨識 dataset。因為資料量小、架構簡單，很好訓練，因此被視為 deep learning 界的 hello world 專案。許多人都將它視為實作入門的教學。

模型架構

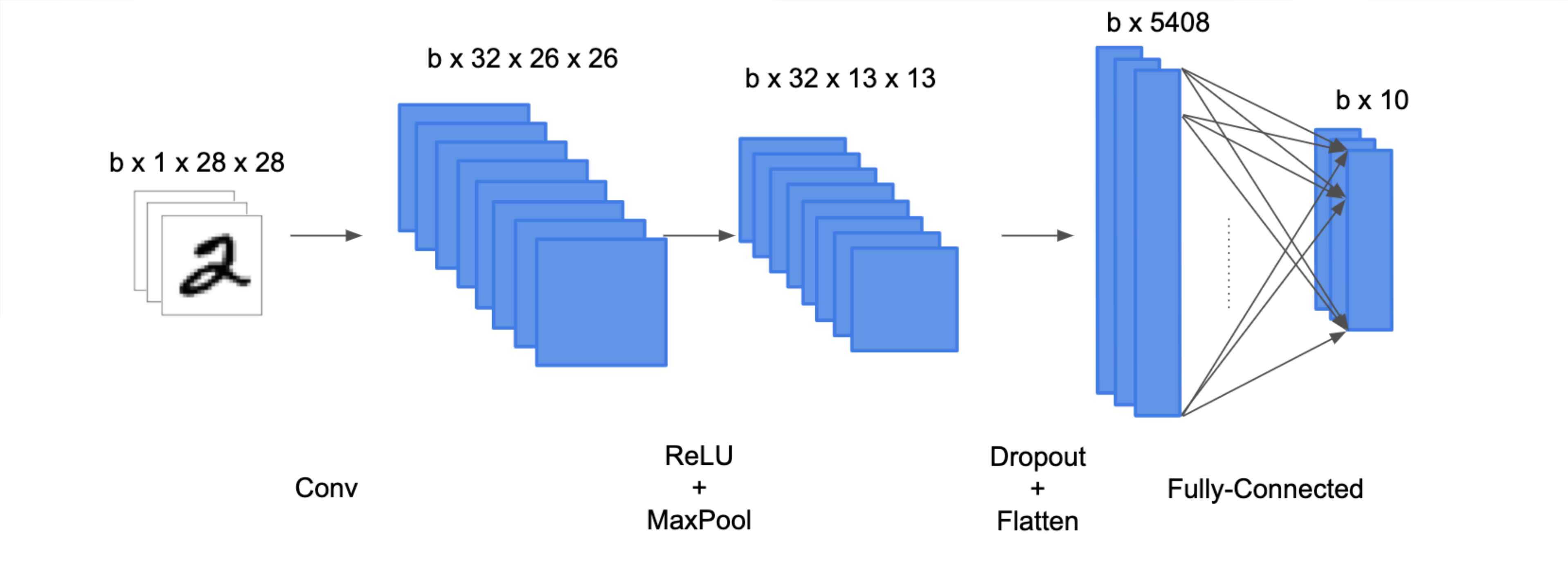
Define Network 三步驟是：繼承 Module class、overwrite **init**()、 和 overwrite forward()：

```
[ ]  class Net(nn.Module):
         def __init__(self):
             super(Net, self).__init__()
             self.conv = nn.Conv2d(1, 32, 3)
             self.dropout = nn.Dropout2d(0.25)
             self.fc = nn.Linear(5408, 10)

         def forward(self, x):
             x = self.conv(x)
             x = F.relu(x)
             x = F.max_pool2d(x, 2)
             x = self.dropout(x)
             x = torch.flatten(x, 1)
             x = self.fc(x)
             output = F.log_softmax(x, dim=1)
             return output
```

使用convolution layer 和 pooling layer 來擷取 image 的 feature ，再把這些 feature map 成 10 個 node 的 output。flatten 把 feature 集中成 vector，再用 fully-connected layer map 到 output layer。

架構圖



b x 1 x 28 x 28
b x 32 x 26 x 26
b x 32 x 13 x 13
b x 5408
b x 10
Conv
ReLU
+
MaxPool
Dropout
+
Flatten
Fully-Connected

訓練過程

```python
def train(model, train_loader, optimizer, epochs, log_interval):
    model.train()
    losses = [] # to store losses
    for epoch in range(1, epochs + 1):
        for batch_idx, (data, target) in enumerate(train_loader):
            # Clear gradient
            optimizer.zero_grad()

            # Forward propagation
            output = model(data)

            # Negative log likelihood loss (log prob + nll loss = prob + cross entropy loss)
            loss = F.nll_loss(output, target)

            # Back propagation
            loss.backward()

            # Parameter update
            optimizer.step()

            # Log training info
            if batch_idx % log_interval == 0:
                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    epoch, batch_idx * len(data), len(train_loader.dataset),
                    100. * batch_idx / len(train_loader), loss.item()))
                losses.append(loss.item()) # append current loss

    # plot the losses
    plt.figure()
    plt.plot(losses)
    plt.title('Training loss over time')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.show()
```

每個 epoch 會 train 過整個 training set，每個 dataset 會做 batch training。

基本的步驟：

1. clear gradient 清空梯度

2. feed data forward

3.取 loss、back propagation 算 gradient

4.最後 update parameter

測試和評估

```python
def test(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad(): # disable gradient calculation for efficiency
        for data, target in test_loader:
            # Prediction
            output = model(data)

            # Compute loss & accuracy
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item() # how many predictions in this batch are correct

    test_loss /= len(test_loader.dataset)

    # Log testing info
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

為了有更好計算速度Disable gradient calculation，首先加總所有的loss，在對每一個樣本的預測結果存儲在 pred 中，然後計算測試集的準確率，最終將loss總數 / 樣本總數=平均loss。

主程式和import

```
def main():
    # Training settings
    BATCH_SIZE = 64
    EPOCHS = 2
    LOG_INTERVAL = 1

    # Define image transform
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,)) # mean and std for the MNIST training set
    ])

    # Load dataset
    train_dataset = datasets.MNIST('./data', train=True, download=True,
                                   transform=transform)
    test_dataset = datasets.MNIST('./data', train=False,
                                  transform=transform)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE)

    # Create network & optimizer
    model = Net()
    optimizer = optim.Adam(model.parameters())

    # Train
    train(model, train_loader, optimizer, EPOCHS, LOG_INTERVAL)

    # Save and load model
    torch.save(model.state_dict(), "mnist_cnn.pt")
    model = Net()
    model.load_state_dict(torch.load("mnist_cnn.pt"))

    # Test
    test(model, test_loader)

if __name__ == '__main__':
    main()
```

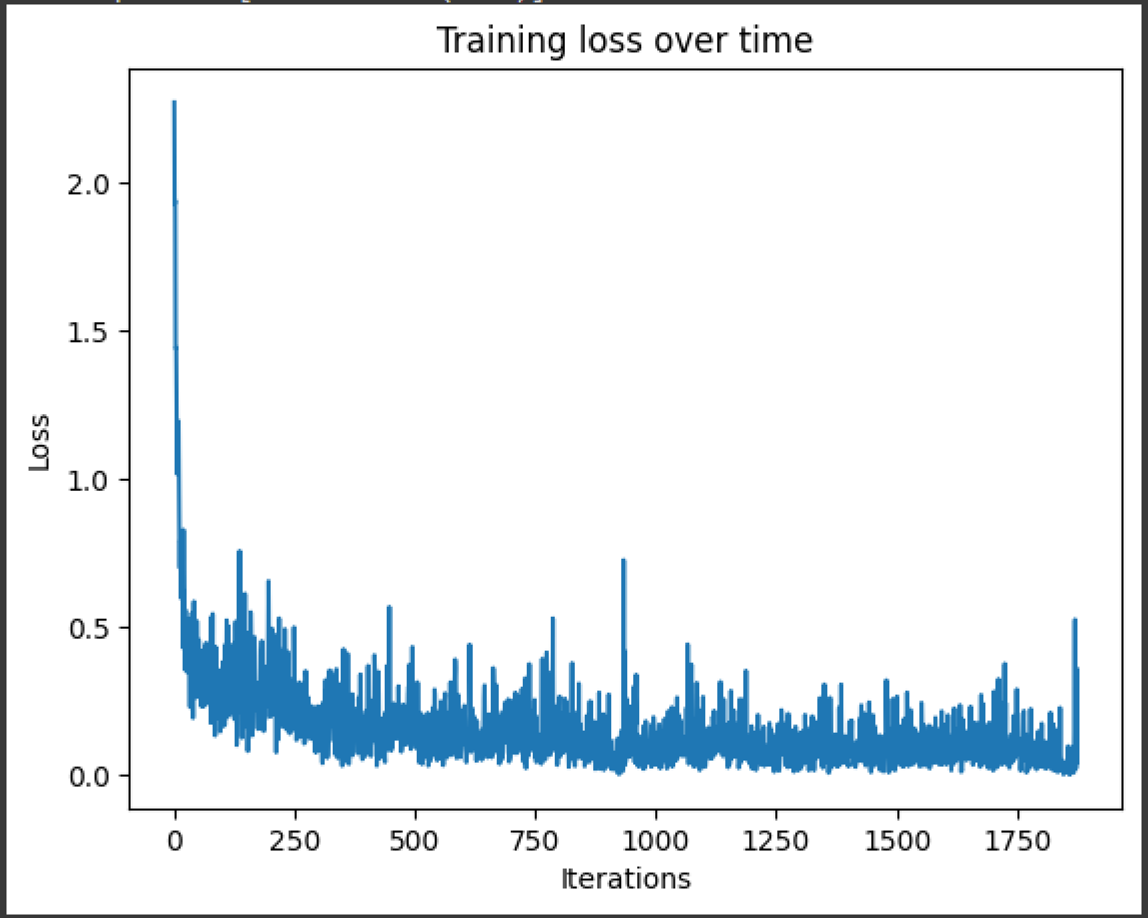
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
```

我們將data loading、training、testing 合在一起，首先設置BATCH_SIZE = 64　EPOCHS = 2　LOG_INTERVAL = 1，將圖像轉換為Tensor後，再去正規化。

利用torchvision.transforms 可以把 dataset 的 data pre-processing。torchvision.transforms功能包括tensor、resize、crop、normalization 。
我們這邊使用 tensor 和 normalization 兩個功能。

成果展示

```
Train Epoch: 2 [59776/60000 (100%)]     Loss: 0.034457
Train Epoch: 2 [59840/60000 (100%)]     Loss: 0.021121
Train Epoch: 2 [59904/60000 (100%)]     Loss: 0.361606
Train Epoch: 2 [29984/60000 (100%)]     Loss: 0.040706
```




```
Test set: Average loss: 0.0705, Accuracy: 9775/10000 (98%)
```

透過在 MNIST 訓練資料集上的兩個 epoch 訓練，我們的模型在測試集上達到了 98% 準確率。

結論

本簡報展示了PyTorch實現MNIST手寫數字辨識的方式和過程，提供了一種種學習深度學習的基礎方法。未來可以嘗試其他的CNN模型，或將模型應用用在其他圖像辨識任務上。