Tên : Nguyễn Văn Mạnh

MSSV: 102200024 Lớp : 20T1

Bài làm:

1. **Hiển thị map các feature map (tức là đầu ra của từng lớp với tấm ảnh được đưa vào).**

- Theo đề bài trước tiên mình phải đề xuất một mô hình CNN có lưu trữ các biến feature map, sau đó khởi tạo model và quy định trọng số đầu tiên của model tức là các kernel ở lớp conv1 sẽ được khởi tạo là kernel được chọn trong bảng ở mục 2, hoặc tự chọn.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1,
padding=1)
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.feature_maps = []

    def forward(self, x):
        x = F.relu(self.conv1(x))
        self.feature_maps.append(x)
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        self.feature_maps.append(x)
        x = self.pool(x)
        x = x.view(-1, 32 * 7 * 7)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

def custom_init_weights(model):
    # Custom weights for conv1
    with torch.no_grad():
        custom_kernel = torch.tensor([[[[1, 0, -1], [1, 0, -1], [1, 0, -1]]]],
dtype=torch.float32)
        repeated_kernel = custom_kernel.repeat(model.conv1.out_channels, 1, 1, 1)
        model.conv1.weight.copy_(repeated_kernel)
```

```
        # Đảm bảo rằng các trọng số còn lại được khởi tạo ngẫu nhiên hoặc theo cách bạn
muốn
        nn.init.kaiming_normal_(model.conv2.weight)
        nn.init.constant_(model.conv2.bias, 0.0)

# Khởi tạo mô hình
model = SimpleCNN()

# Tùy chỉnh kernel đầu vào
custom_init_weights(model)

# Kiểm tra kernel đầu vào
print(model.conv1.weight[1])
```

Ta có kết quả kiểm tra như sau:

```
tensor([[[ 1.,  0., -1.],
         [ 1.,  0., -1.],
         [ 1.,  0., -1.]]], grad_fn=<SelectBackward0>)
```

Đúng với kernel được chọn trong code.

- Để show các feature map, sử dụng thư viện matplotlib.

```
def plot_feature_maps(feature_maps, num_columns=8):
    for idx, fmap in enumerate(feature_maps):
        fmap = fmap.detach().cpu().numpy()
        num_filters = fmap.shape[1]

        # Tính toán số hàng cần thiết
        num_rows = (num_filters + num_columns - 1) // num_columns

        fig, axes = plt.subplots(num_rows, num_columns, figsize=(num_columns * 2, num_rows
* 2))
        fig.suptitle(f'Layer {idx+1} Feature Maps')

        for i in range(num_filters):
            row = i // num_columns
            col = i % num_columns
            ax = axes[row, col]
            ax.imshow(fmap[0, i]) #, cmap='gray'
            ax.axis('off')

        # Ẩn các ô trống nếu có
        for i in range(num_filters, num_rows * num_columns):
            row = i // num_columns
            col = i % num_columns
            ax = axes[row, col]
            ax.axis('off')

        plt.show()
```

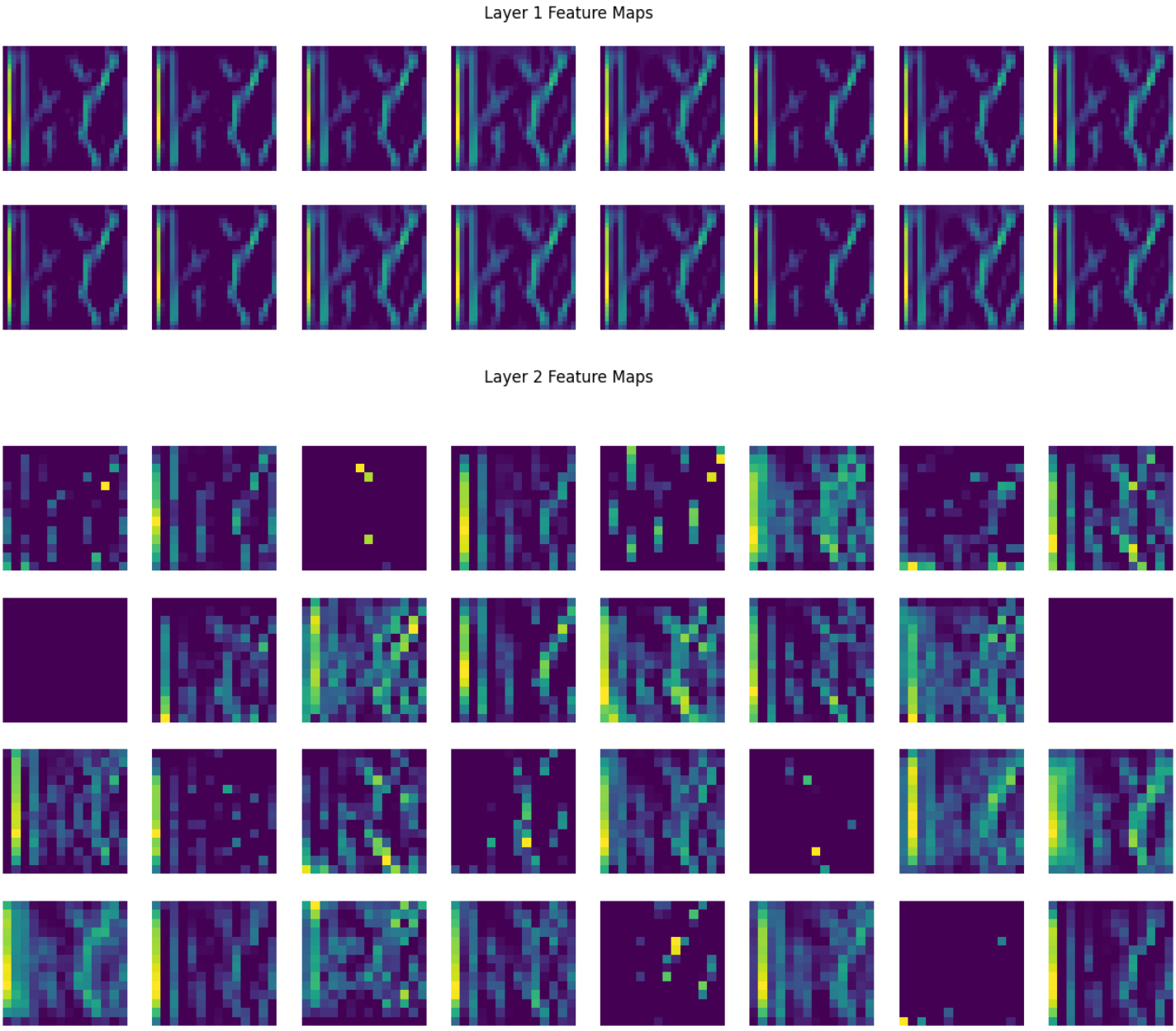
- Tiếp theo chúng ta cần hàm để load file hình ảnh của đề bài ra, chuyển sang grayscale, resize về 28x28 và chuẩn hoá để phù hợp với mô hình CNN đã định nghĩa trước đó.

```
from torchvision import datasets, transforms
from PIL import Image

def load_image(image_path):
    transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=1),  # Chuyển đổi thành ảnh xám nếu cần
        transforms.Resize((28, 28)),  # Resize ảnh về kích thước 28x28
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])

    image = Image.open(image_path)
    image = transform(image)
    return image
```

- Tiếp theo là đoạn code chỉnh, sử dụng model vừa tạo, gọi hàm load ảnh ra, đưa vào model, lấy các feature map của model đối với bức ảnh truyền vào hiển thị ra:

```
# # Khởi tạo mô hình và tải trọng số đã huấn luyện (nếu có)
# model = SimpleCNN()

# Đọc ảnh từ file
image_path = '/content/drive/MyDrive/DeepLearning/Research/OnCK/input_image.png'  # Thay
thế bằng đường dẫn tới ảnh của bạn
image = load_image(image_path)

# Xóa feature maps cũ trước khi forward
model.feature_maps = []

# Forward qua mô hình
output = model(image.unsqueeze(0))

# Hiển thị các feature maps
plot_feature_maps(model.feature_maps)
```

Ta có kết quả ở bước này như sau:



Tổng cộng có 16 feature map là output của lớp conv1 và 32 feature map là output của lớp conv2.

2. Hiển thị kích thước vector flatting

Có 2 cách để hiện thị kích thước của vector flatting.

- Sử dụng summary().

```
!pip install torchsummary

from torchsummary import summary
```

```
summary(model, (1, 28, 28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]             160
         MaxPool2d-2           [-1, 16, 14, 14]               0
            Conv2d-3           [-1, 32, 14, 14]           4,640
         MaxPool2d-4             [-1, 32, 7, 7]               0
            Linear-5                  [-1, 128]         200,832
            Linear-6                   [-1, 10]           1,290
================================================================
Total params: 206,922
Trainable params: 206,922
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.18
Params size (MB): 0.79
Estimated Total Size (MB): 0.97
----------------------------------------------------------------
```

Có thể thấy trước khi qua Linear-5 thì trong model có lớp views do đó vector flatten có kích thước là 32*7*7 = 1568

- Sử dụng in thủ công

```
print(model.fc1.weight.shape)
```

```
torch.Size([128, 1568])
```

Có thể thấy trước khi vào lớp fully connected thứ nhất thì có chiều 1568.

3. Lớp full connection sử dụng các hàm: Loss loss function, hàm optimazer:

- Sử dụng hàm mse loss cho phân loại nhiều lớp

```
def custom_mse_loss(output, target):
    loss = torch.mean((output - target) ** 2)
    return loss
```

- Sử dụng hàm SGD tự định nghĩa

```
class SimpleSGDOptimizer:
    def __init__(self, params, lr=0.01):
        self.params = list(params)
        self.lr = lr

    def step(self):
        for param in self.params:
            if param.grad is not None:
                param.data -= self.lr * param.grad

    def zero_grad(self):
        for param in self.params:
            if param.grad is not None:
                param.grad.zero_()
```

- Sử dụng hàm Adam tự định nghĩa

```
class AdamOptimizer:
  def __init__(self, params, lr=0.001, beta1=0.9, beta2=0.999):
    self.params = list(params)
    self.lr = lr
    self.beta1 = beta1
    self.beta2 = beta2

    self.iter = 0
    self.m = [0] * len(self.params)
    self.v = [0] * len(self.params)

  def step(self):
    self.iter += 1

    for i, param in enumerate(self.params):
      if param.grad is not None:
        self.m[i] = self.beta1 * self.m[i] + (1 - self.beta1) * param.grad
        self.v[i] = self.beta2 * self.v[i] + (1 - self.beta2) * param.grad ** 2

        m_hat = self.m[i] / (1 - self.beta1 ** self.iter)
        v_hat = self.v[i] / (1 - self.beta2 ** self.iter)

        param.data -= self.lr * m_hat / (v_hat**0.5 + 1e-7)

  def zero_grad(self):
    for param in self.params:
      if param.grad is not None:
        param.grad.zero_()
```

- Sử dụng dữ liệu MNIST để huấn luyện:

```
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Chuẩn bị dữ liệu MNIST
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,),
(0.5,))])
train_dataset = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
train_loader = DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)

# Khởi tạo mô hình, hàm loss và optimizer
# model = SimpleCNN()
# optimizer = SimpleSGDOptimizer(model.parameters(), lr=0.01)
optimizer = AdamOptimizer(model.parameters())

# Huấn luyện mô hình
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        model.feature_maps = []
        optimizer.zero_grad()
        output = model(data)

        # Chuyển đổi target thành one-hot vector
        target_one_hot = torch.zeros(output.size()).scatter_(1, target.view(-1, 1), 1)

        loss = custom_mse_loss(output, target_one_hot)
        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            print(f'Epoch {epoch+1}/{num_epochs}, Batch {batch_idx}, Loss: {loss.item()}')
```

- Sử dụng code đánh giá:

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score

def evaluate(model, data_loader):
    model.eval()
    all_preds = []
    all_targets = []

    with torch.no_grad():
        for data, target in data_loader:
```

```
            output = model(data)
            preds = output.argmax(dim=1, keepdim=True).squeeze()
            all_preds.extend(preds.cpu().numpy())
            all_targets.extend(target.cpu().numpy())

    precision = precision_score(all_targets, all_preds, average='weighted')
    recall = recall_score(all_targets, all_preds, average='weighted')
    f1 = f1_score(all_targets, all_preds, average='weighted')
    accuracy = accuracy_score(all_targets, all_preds)

    return precision, recall, f1, accuracy

# Đánh giá mô hình sau mỗi epoch
precision, recall, f1, accuracy = evaluate(model, train_loader)
print(f'Epoch {epoch+1}/{num_epochs}, Precision: {precision:.4f}, Recall: {recall:.4f},
F1: {f1:.4f}, Accuracy: {accuracy:.4f}')
```

4. So sánh kết quả trên khi sử dụng adam hoặc sgd.

Kết quả sử dụng SGD:

```
Epoch 5/5, Precision: 0.9126, Recall: 0.9120, F1: 0.9118, Accuracy: 0.9120
```

Kết quả sử dụng Adam:

```
Epoch 5/5, Precision: 0.9904, Recall: 0.9904, F1: 0.9904, Accuracy: 0.9904
```

5. Hiển thị weight sau khi mô hình hội tụ

```
# lớp convolution 1
print(model.conv1.weight.shape)
print(model.conv1.weight[0])
# lớp convolution 2
print(model.conv2.weight.shape)
print(model.conv2.weight[0][0])
# lớp fully connected 1
print(model.fc1.weight.shape)
print(model.fc1.weight[0:10])
# lớp fully connected 2
print(model.fc2.weight.shape)
print(model.fc2.weight[0:10])
```

Kết quả:

```
torch.Size([16, 1, 3, 3])
tensor([[[ 1.0176, -0.0131, -0.9833],
         [ 1.1647, -0.1174, -0.9164],
         [ 1.2950,  0.5213, -0.4971]]], grad_fn=<SelectBackward0>)
torch.Size([32, 16, 3, 3])
tensor([[ 0.0097, -0.2179, -0.0169],
        [ 0.0420,  0.1260, -0.0112],
        [-0.2103, -0.1445, -0.1879]], grad_fn=<SelectBackward0>)
torch.Size([128, 1568])
tensor([[-0.0194,  0.0175,  0.0090,  ...,  0.0052, -0.0073, -0.0079],
        [ 0.0150, -0.0173, -0.0038,  ..., -0.0089,  0.0101, -0.0025],
        [-0.0288,  0.0077, -0.0032,  ...,  0.0176,  0.0157, -0.0003],
        ...,
        [-0.0199,  0.0240, -0.0160,  ..., -0.0201, -0.0166,  0.0070],
        [-0.0084, -0.0232, -0.0788,  ..., -0.0052,  0.0058,  0.0934],
        [-0.0020, -0.0043,  0.0158,  ...,  0.0091,  0.0127, -0.0229]],
       grad_fn=<SliceBackward0>)
torch.Size([10, 128])
tensor([[ 0.0790,  0.0122,  0.0197,  ..., -0.0144, -0.0684,  0.0366],
        [ 0.0284, -0.0654,  0.0631,  ...,  0.2113, -0.0287, -0.0220],
        [-0.0730, -0.0054, -0.0197,  ..., -0.0244,  0.0823, -0.0762],
        ...,
        [-0.0361,  0.0174, -0.0575,  ..., -0.0225,  0.0181, -0.0549],
        [ 0.0428, -0.0177,  0.0536,  ..., -0.0322,  0.0318,  0.0423],
        [-0.0657, -0.0757, -0.0821,  ..., -0.0239,  0.0563, -0.0352]],
       grad_fn=<SliceBackward0>)
```

6. Dự đoán

```python
def show_image_with_prediction(model, data_loader, index=0):
    model.eval()
    with torch.no_grad():
        data, target = next(iter(data_loader))
        image = data[index]
        true_label = target[index].item()

        output = model(image.unsqueeze(0))
        pred_label = output.argmax(dim=1, keepdim=True).item()

        image = image.squeeze().numpy()

        plt.imshow(image, cmap='gray')
        plt.title(f'True Label: {true_label}, Predicted Label: {pred_label}')
        if (true_label==pred_label):
            print('model dự đoán đúng')
        else:
            print('model dự đoán sai')
        plt.show()
```

Kết quả:

```
# Hiển thị ảnh và dự đoán
show_image_with_prediction(model, train_loader, index=0)
```

model dự đoán đúng