**Tên : Nguyễn Văn Mạnh**

**MSSV: 102200024 Lớp : 20T1**

Bài làm:

# 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 = []

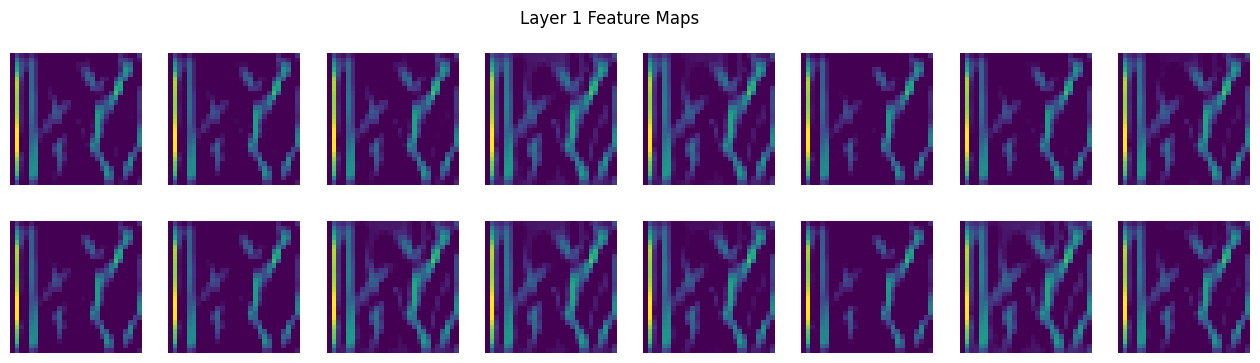
# Forward qua mô hình

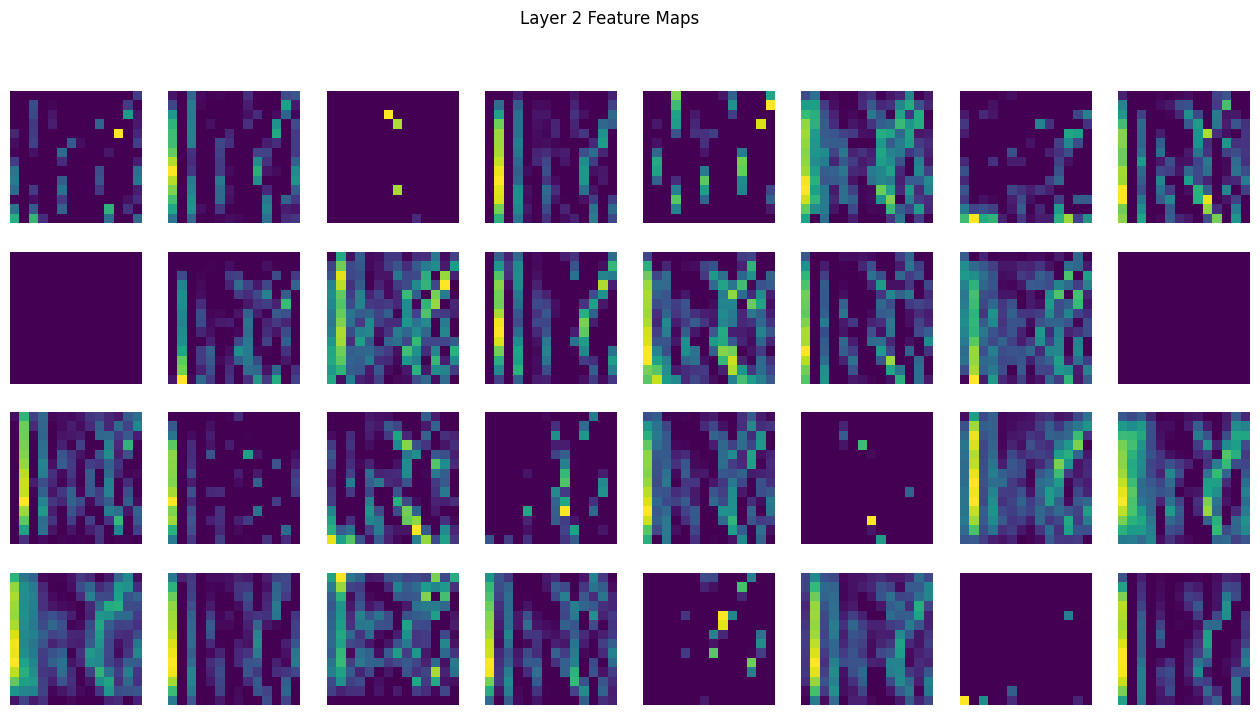
output = model(image.unsqueeze(0))

# 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.

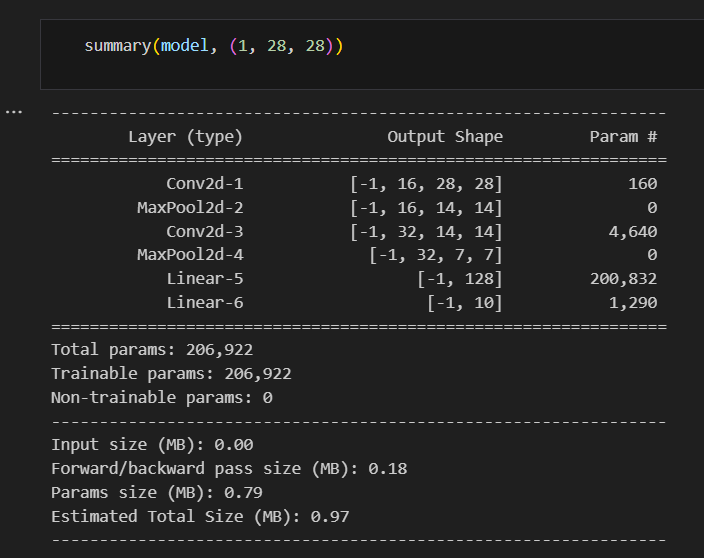
# Hiển thị kích thước vector flatting

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

* Sử dụng summary().

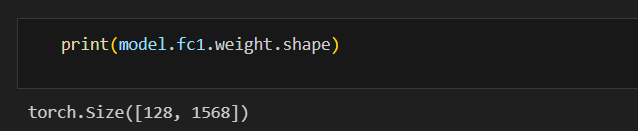
!pip install torchsummary

from torchsummary import summary



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



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

# 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

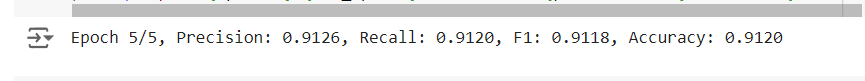
# Đánh giá mô hình sau mỗi epoch

precision, recall, f1, accuracy = evaluate(model, train\_loader)

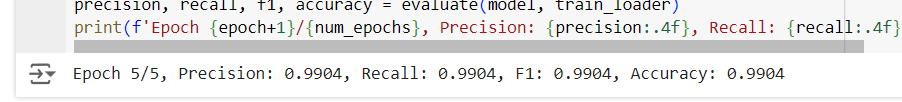
print(f'Epoch {epoch+1}/{num\_epochs}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1: {f1:.4f}, Accuracy: {accuracy:.4f}')

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

Kết quả sử dụng SGD:



Kết quả sử dụng Adam:



# 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)

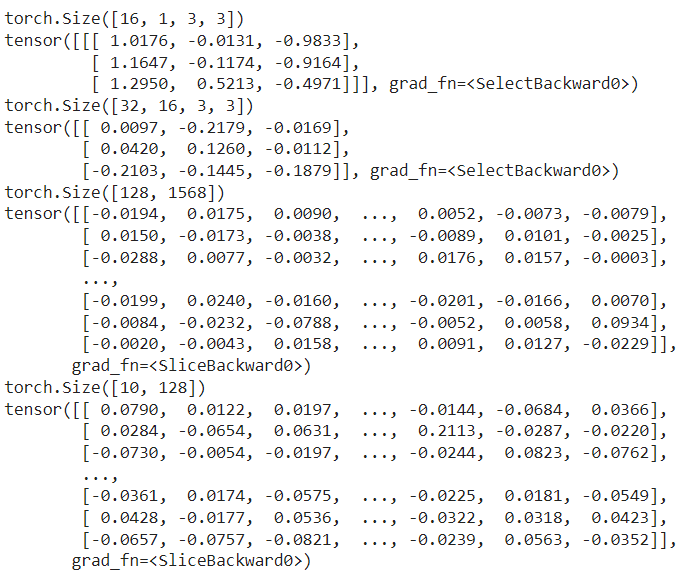
print(model.fc1.weight[0:10])

# lớp fully connected 2

print(model.fc2.weight.shape)

print(model.fc2.weight[0:10])

Kết quả:



# Dự đoán

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:

