Embedded System Lab4

Group14
61375017H 陳昕佑
61375082H 林聖倫
61375070H 黃柏瑜

Lab4:

Components used:

1. Raspberry Pi 5 & power supply

Process:

Train CNN on PC, and download the dataset from Kaggle to Pi 5.

Code:

1. Train on PC:

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

# Define the model
class Net(nn.Module):
   def __init__(self):
       super(Net, self).__init__()
       self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
       self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
       self.pool = nn.MaxPool2d(2, 2)
       self.fc1 = nn.Linear(64 * 7 * 7, 128)
       self.fc2 = nn.Linear(128, 10)
       self.relu = nn.ReLU()
       self.dropout = nn.Dropout(0.25)

   def forward(self, x):
       x = self.relu(self.conv1(x))
       x = self.pool(self.relu(self.conv2(x)))
       x = self.pool(x)  # second pooling to go from 14x14 → 7x7
       x = x.view(-1, 64 * 7 * 7)
       x = self.dropout(self.relu(self.fc1(x)))
```

```
        x = self.fc2(x)
        return x

# Set up training parameters
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
learning_rate = 0.001
epochs = 5
batch_size = 64

# Load MNIST dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
train_dataset = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

# Instantiate model, loss, optimizer
model = Net().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training loop
for epoch in range(epochs):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1}/{epochs} - Loss: {loss.item():.4f}")

# Save model
torch.save(model.state_dict(), "mnist_cnn.pth")
print("Model saved to mnist_cnn.pth")
```

2.Run on Pi 5:

```
from PIL import Image
import torch
import torch.nn as nn
from torchvision import transforms

# Define the model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(x)  # 2nd pooling layer to get 7x7 output
        x = x.view(-1, 64 * 7 * 7)
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

# Load model
model = Net()
model.load_state_dict(torch.load("mnist_cnn.pth", map_location="cpu"))
model.eval()

# Load and preprocess image
img = Image.open("/home/rueii/Graduation/First_Year/Embedded-System/WEEK9/ChatGPT Image.png").convert("L")  # grayscale
img.show()  # Show the input image

transform = transforms.Compose([
    transforms.Resize((28, 28)),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
img_tensor = transform(img).unsqueeze(0)  # Add batch dimension

# Predict
with torch.no_grad():
    output = model(img_tensor)
    pred = output.argmax(dim=1)
    print(f"Prediction: {pred.item()}")
```

Video: https://youtu.be/ah56i8S8UHY
Review of Experience: None