Laporan Akhir : Convolutional Neural Network (CNN)

Mata Kuliah : Pembelajaran mesin

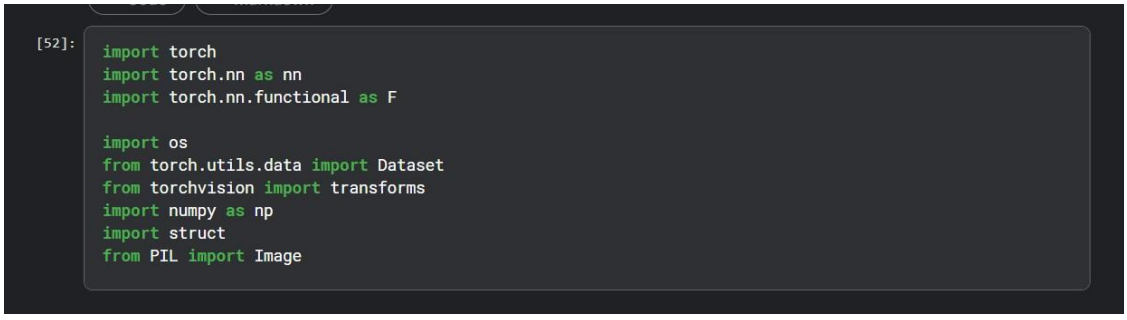
Dosen Pengampu : Dr. Oddy Virgantara Putra, S.Kom., M.T.

Kelompok :

- Rizky Cahyono Putra
- Raffa Arvel Nafi'Nadindra
- Syaifan Nur
- Irfansyah
- Muhammad Galang Fachrezy

Laporan ini menyajikan proyek deep learning yang menggunakan *PyTorch* untuk membangun dan melatih model *Convolutional Neural Network* (CNN) dalam mengklasifikasikan angka dari dataset MNIST. Berikut penjelasan dari bagian-bagian awal kode:

1. Import Library



Mengimpor berbagai library untuk manipulasi data (numpy, pandas), dan membangun model deep learning (torch, torchvision.transforms).

2. Dataset dan Preprocessing

   Dataset digunakan dalam format asli .idx3-ubyte untuk gambar dan .idx1-ubyte untuk label. Data dibaca menggunakan class custom MNISTDataset, lalu diproses ke dalam bentuk tensor PyTorch.

```
class MNISTDataset(Dataset):
    def __init__(self, images_path, labels_path, image_size=28):
        self.images_path = images_path
        self.labels_path = labels_path
        self.image_size = image_size

        self.images = self._read_images(self.images_path)
        self.labels = self._read_labels(self.labels_path)

        self.transform = transforms.Compose([
            transforms.Resize((self.image_size, self.image_size)),
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,))
        ])

    def _read_images(self, filepath):
        with open(filepath, 'rb') as f:
            magic, num, rows, cols = struct.unpack(">IIII", f.read(16))
            images = np.frombuffer(f.read(), dtype=np.uint8).reshape(num, 28, 28)
        return images

    def _read_labels(self, filepath):
        with open(filepath, 'rb') as f:
            magic, num = struct.unpack(">II", f.read(8))
            labels = np.frombuffer(f.read(), dtype=np.uint8)
        return labels
```

Fungsi di atas membuka file binary dan memparsing header MNIST untuk mendapatkan jumlah data dan dimensi gambar, kemudian membaca buffer sebagai array dan NumPy

3. Arsitektur Model CNN

Model CNN terdiri dari beberapa konvolusi dan pooling, diakhiri dengan fully connected layer untuk klasifikasi.

```python
class MnistClassifier(nn.Module):

    def __init__(self, num_classes=10):
        super(MnistClassifier, self).__init__()

        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)

        self.fc1 = nn.Linear(64 * 3 * 3, 64)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool(x)

        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)

        x = self.conv3(x)
        x = F.relu(x)
        x = self.pool(x)

        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)

        x = self.fc2(x)

        return x
```

- Tiga layer Konvolusi dengan padding mempertahankan dimensi gambar
- Max pooling mengurangi dimensi spasial
- Layer Fully Connected mengonversi fitur spasial kedalam bentuk klasifikasi

4. Traning dataset

```
import argparse
from torch.utils.data import DataLoader
import torch.optim as optim
from tqdm import tqdm
import matplotlib.pyplot as plt

train_dataset = MNISTDataset(
        images_path='/kaggle/input/mnist-dataset/train-images.idx3-ubyte',
        labels_path='/kaggle/input/mnist-dataset/train-labels.idx1-ubyte'
    )

test_dataset = MNISTDataset(
        images_path='/kaggle/input/mnist-dataset/t10k-images.idx3-ubyte',
        labels_path='/kaggle/input/mnist-dataset/t10k-labels.idx1-ubyte'
    )

batch_size = 8
test_batch_size = 8

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=test_batch_size)
```

train_dataset berisi gambar dan label angka MNIST dari file .idx, diubah menjadi tensor PyTorch dan dibungkus dalam DataLoader. Selama pelatihan, model akan membaca data dari train_loader dalam batch berukuran 8, dengan urutan yang diacak untuk generalisasi yang lebih baik.

5. Setup Perangkat & Model

Berguna untuk mengoptimalisasi Ketika program berjalan

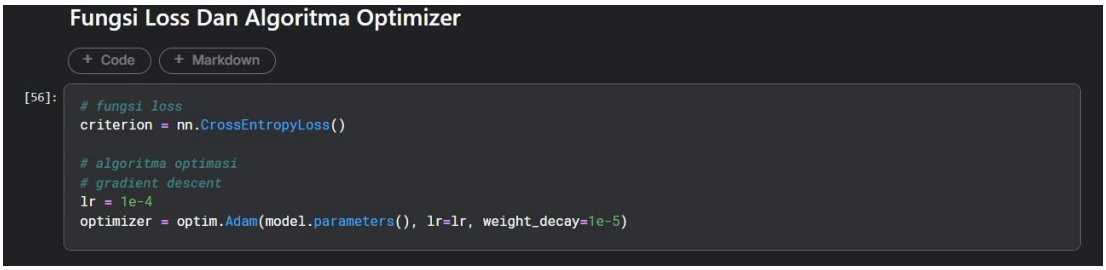
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

num_classes = 10

model = MnistClassifier(num_classes)
model = model.to(device)
```

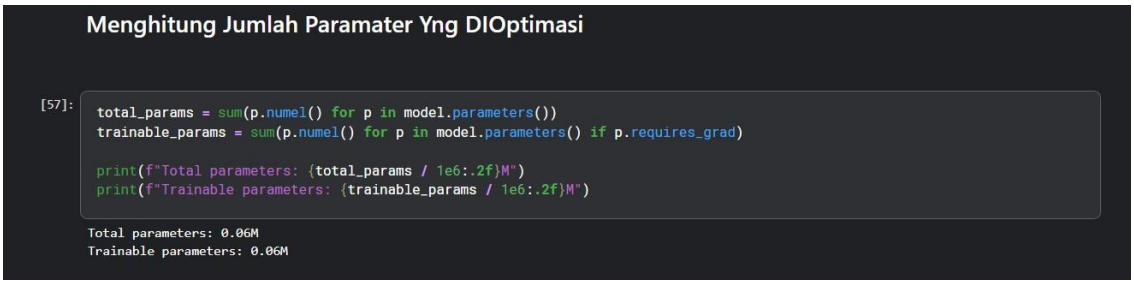
- Melakukan Pengecekan apabila GPU tersedia di dalam perangkat agar program dapat dieksekusi dengan lebih baik.

6. Loss Function dan Optimizer



- Loss function: *CrossEntropyLoss,* cocok untuk klasifikasi multi-kelas.
- Optimizer: Adam, varian dari gradient descent yang efisien.
- Learning rate (lr): 1e-4 — kecepatan pembelajaran.
- Weight decay: 1e-5, membantu mencegah *overfitting.*

7. Parameter Model



• Menampilkan Total parameter dan Parameter yang bisa dilatih (trainable)

8. Inisialisasi

```
train_losses = []
test_losses = []

epochs = 5
checkpoint = 'model_checkpoint.pth'
```

Menentukan berapa kali model akan di evaluasi, 5 epochs menandakan bahwa model akan di jalan kan selama 5 kali.

9.Evaluasi Model

```
for epoch in range(epochs):
        model.train()
        train_loss = 0

        for data, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
            data, labels = data.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(data)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

        train_loss += loss.item() * data.size(0)

        model.eval()
        correct = 0
        total = 0
        test_loss = 0
```

- model.train(): Mengaktifkan mode training (menyalakan dropout/batch norm jika ada).
- train_loss = 0: Variabel untuk menghitung total loss selama 1 epoch.

10. Loop Data Testing

- Agar bisa diproses oleh model di perangkat (device = GPU/CPU).
- outputs: Hasil prediksi model.

- Menampilkan hasil training dan evaluasi tiap epoch dengan 4 angka di belakang koma.

11. Menyimpan Model Dan Visualisasi

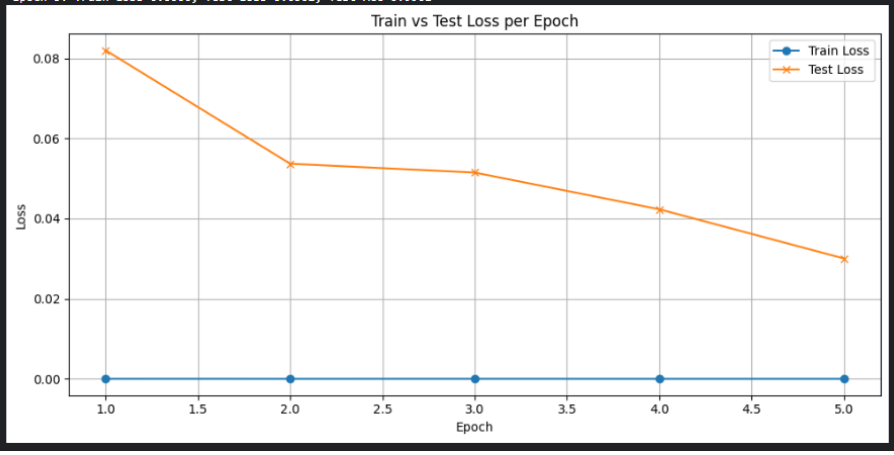
```
torch.save({
    'epoch': epochs,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'train_loss': train_losses,
    'test_loss': test_losses,
}, checkpoint)

epochs_range = range(1, epochs + 1)
plt.figure(figsize=(10, 5))
plt.plot(epochs_range, train_losses, label='Train Loss', marker='o')
plt.plot(epochs_range, test_losses, label='Test Loss', marker='x')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Train vs Test Loss per Epoch')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

• Kode ini menyimpan checkpoint model ke file dalam format dictionary.

12. Hasil visualisasi dan hasil loop testing

```
Epoch 1: 100%|██████████| 7500/7500 [00:27<00:00, 272.98it/s]
Epoch 1: Train Loss 0.0000, Test Loss 0.0820, Test Acc 0.9747
Epoch 2: 100%|██████████| 7500/7500 [00:27<00:00, 274.60it/s]
Epoch 2: Train Loss 0.0000, Test Loss 0.0537, Test Acc 0.9834
Epoch 3: 100%|██████████| 7500/7500 [00:27<00:00, 275.73it/s]
Epoch 3: Train Loss 0.0000, Test Loss 0.0515, Test Acc 0.9838
Epoch 4: 100%|██████████| 7500/7500 [00:27<00:00, 275.10it/s]
Epoch 4: Train Loss 0.0000, Test Loss 0.0423, Test Acc 0.9866
Epoch 5: 100%|██████████| 7500/7500 [00:27<00:00, 276.19it/s]
Epoch 5: Train Loss 0.0000, Test Loss 0.0301, Test Acc 0.9901
```



13. Simpulan

Model CNN yang dibuat berhasil mengenali angka MNIST dengan akurasi tinggi. Pipeline dimulai dari preprocessing hingga evaluasi akhir telah berhasil dijalankan. Proyek ini memperkuat pemahaman praktis mengenai deep learning dan arsitektur CNN.