Nama: Andi Cleopatra Maryam Jamila

Nim: 1103213071

Analisis Week 12 Fashion mnist

```
import torch
import torch.nn as nn
import torch.optim as optim
import time
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import StepLR
```

```
# Kelas Early Stopping
class EarlyStopping:
    def __init__(self, patience=5, delta=0):
        self.patience = patience
        self.delta = delta
        self.best_loss = None
        self.counter = 0
        self.stop = False

    def should_stop(self, val_loss):
        if self.best_loss is None:
            self.best_loss = val_loss
        elif val_loss > self.best_loss + self.delta:
            self.counter += 1
            if self.counter >= self.patience:
                self.stop = True
        else:
            self.best_loss = val_loss
            self.counter = 0
        return self.stop
```

```
import torch
import torch.nn as nn
```

```python
class CNNModel(nn.Module):
    def __init__(self, kernel_size=3, pool_type='max'):
        super(CNNModel, self).__init__()

        # Tentukan layer konvolusi
        self.conv1 = nn.Conv2d(1, 32, kernel_size=kernel_size, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=kernel_size, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=kernel_size, padding=1)

        # Tentukan pooling
        if pool_type == 'max':
            self.pool = nn.MaxPool2d(2, 2)
        elif pool_type == 'avg':
            self.pool = nn.AvgPool2d(2, 2)

        # Menghitung ukuran output setelah lapisan konvolusi dan pooling
        self._to_linear = None
        self.convs(torch.randn(1, 1, 28, 28))  # Hitung ukuran output untuk gambar 28x28

        # Fully connected layer
        self.fc1 = nn.Linear(self._to_linear, 512)
        self.fc2 = nn.Linear(512, 10)  # Output layer untuk 10 kelas

    def convs(self, x):
        # Fungsi untuk menghitung ukuran output setelah konvolusi dan pooling
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))

        # Menyimpan ukuran output yang akan digunakan pada fc1
        if self._to_linear is None:
            self._to_linear = x.numel()  # Menyimpan jumlah elemen setelah pooling
```

```python
        return x

    def forward(self, x):
        # Forward pass melalui konvolusi dan pooling
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))

        # Flatten tensor
        x = torch.flatten(x, 1)

        # Forward pass melalui fully connected layer
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
# Fungsi Training dan Evaluasi
def train_and_evaluate(model, optimizer, criterion, epochs, trainloader, testloader, lr_scheduler=None, early_stopper=None):
    train_losses = []
    test_accuracies = []

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        # Langkah pelatihan
        for inputs, labels in trainloader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        train_losses.append(running_loss / len(trainloader))

        # Langkah evaluasi (test)
        model.eval()
        with torch.no_grad():
            for inputs, labels in testloader:
                outputs = model(inputs)
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        accuracy = 100 * correct / total
        test_accuracies.append(accuracy)

        # Learning rate scheduler (opsional)
        if lr_scheduler:
            lr_scheduler.step()

        # Early stopping (opsional)
        if early_stopper and early_stopper.should_stop(running_loss / len(trainloader)):
            print(f"Early stopping di epoch {epoch}")
            break

        print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss / len(trainloader):.4f}, Accuracy: {accuracy:.2f}%")

    return train_losses, test_accuracies
```

```python
# Memuat Dataset Fashion-MNIST
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
trainset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
testset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64, shuffle=False)
```

```python
# Hyperparameter
epochs_list = [5, 50, 100, 250, 350]
kernel_sizes = [3, 5, 7]
pool_types = ['max', 'avg']
optimizers = ['sgd', 'adam', 'rmsprop']
```

```python
# Penyimpanan Hasil
results = {}
```

```python
# Loop Eksperimen
for kernel_size in kernel_sizes:
    for pool_type in pool_types:
        for optimizer_type in optimizers:
            for epochs in epochs_list:
                print(f"Pelatihan dengan kernel_size={kernel_size}, pool_type={pool_type}, optimizer={optimizer_type}, epochs={epochs}")

                # Definisikan model, kriteria, dan optimizer
                model = CNNModel(kernel_size=kernel_size, pool_type=pool_type)
                criterion = nn.CrossEntropyLoss()

                if optimizer_type == 'sgd':
                    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
                elif optimizer_type == 'adam':
                    optimizer = optim.Adam(model.parameters(), lr=0.001)
                elif optimizer_type == 'rmsprop':
                    optimizer = optim.RMSprop(model.parameters(), lr=0.001)

                # Learning Rate Scheduler
                scheduler = StepLR(optimizer, step_size=5, gamma=0.7)

                # Early Stopping
                early_stopper = EarlyStopping(patience=10)

                # Latih dan evaluasi
                start_time = time.time()
                train_losses, test_accuracies = train_and_evaluate(
                    model, optimizer, criterion, epochs, trainloader, testloader, lr_scheduler=scheduler, early_stopper=early_stopper
                )
                end_time = time.time()
```

```python
# Simpan hasil
results[(kernel_size, pool_type, optimizer_type, epochs)] = {
    'train_losses': train_losses,
    'test_accuracies': test_accuracies,
    'time_taken': end_time - start_time
}

print(f"Pelatihan selesai dalam {end_time - start_time:.2f} detik")
```

Output:

```
Pelatihan dengan kernel_size=3, pool_type=max, optimizer=sgd, epochs=5
Epoch 1/5, Loss: 0.6239, Accuracy: 86.40%
Epoch 2/5, Loss: 0.3309, Accuracy: 86.68%
Epoch 3/5, Loss: 0.2802, Accuracy: 89.73%
Epoch 4/5, Loss: 0.2457, Accuracy: 90.11%
Epoch 5/5, Loss: 0.2251, Accuracy: 90.37%
Pelatihan selesai dalam 604.62 detik
Pelatihan dengan kernel_size=3, pool_type=max, optimizer=sgd, epochs=50
Epoch 1/50, Loss: 0.6194, Accuracy: 85.54%
Epoch 2/50, Loss: 0.3280, Accuracy: 87.25%
Epoch 3/50, Loss: 0.2767, Accuracy: 89.12%
Epoch 4/50, Loss: 0.2503, Accuracy: 89.99%
Epoch 5/50, Loss: 0.2236, Accuracy: 90.30%
Epoch 6/50, Loss: 0.1939, Accuracy: 90.69%
Epoch 7/50, Loss: 0.1811, Accuracy: 91.29%
Epoch 8/50, Loss: 0.1686, Accuracy: 91.57%
Epoch 9/50, Loss: 0.1575, Accuracy: 91.42%
Epoch 10/50, Loss: 0.1475, Accuracy: 91.75%
Epoch 11/50, Loss: 0.1227, Accuracy: 92.02%
Epoch 12/50, Loss: 0.1149, Accuracy: 92.03%
Epoch 13/50, Loss: 0.1070, Accuracy: 91.53%
Epoch 14/50, Loss: 0.0981, Accuracy: 91.89%
Epoch 15/50, Loss: 0.0910, Accuracy: 91.94%
Epoch 16/50, Loss: 0.0724, Accuracy: 91.79%
Epoch 17/50, Loss: 0.0645, Accuracy: 92.13%
Epoch 18/50, Loss: 0.0590, Accuracy: 92.19%
Epoch 19/50, Loss: 0.0523, Accuracy: 91.57%
Epoch 20/50, Loss: 0.0480, Accuracy: 92.03%
Epoch 21/50, Loss: 0.0354, Accuracy: 92.30%
Epoch 22/50, Loss: 0.0307, Accuracy: 92.04%
Epoch 23/50, Loss: 0.0272, Accuracy: 92.12%
Epoch 24/50, Loss: 0.0250, Accuracy: 91.92%
Epoch 25/50, Loss: 0.0215, Accuracy: 92.18%
```

Analisis:

Simulasi ini dirancang untuk mengevaluasi performa model CNN dalam klasifikasi gambar menggunakan dataset Fashion-MNIST dengan menguji variasi hiperparameter seperti ukuran kernel, jenis pooling, optimizer, dan jumlah epoch. Model CNN yang digunakan terdiri dari tiga lapisan konvolusi dan dua lapisan fully connected. Tiga jenis optimizer yang diuji adalah SGD, Adam, dan RMSprop, yang masing-masing memiliki keunggulan dalam cara mereka mengupdate bobot selama pelatihan. Selain itu, dua jenis pooling yang diuji adalah max pooling dan average pooling, yang mempengaruhi cara model mengurangi dimensi data dan menangkap fitur utama. Untuk menghindari overfitting, diterapkan early stopping, yang menghentikan pelatihan jika tidak ada perbaikan dalam loss pada sejumlah epoch berturut-turut. Learning rate scheduler juga digunakan untuk menurunkan laju pembelajaran setelah beberapa epoch untuk meningkatkan konvergensi.

Hasil eksperimen disimpan dalam bentuk dictionary yang mencakup train loss, akurasi pada data pengujian, serta waktu pelatihan yang dibutuhkan untuk setiap kombinasi hiperparameter. Dengan melakukan eksperimen terhadap berbagai kombinasi kernel, pooling, optimizer, dan epoch, simulasi ini memberikan wawasan tentang pengaruh masing-masing faktor terhadap kinerja model. Pembandingan hasil akurasi dan efisiensi waktu pelatihan untuk tiap konfigurasi memungkinkan pemilihan model yang optimal. Hasil dari simulasi ini dapat digunakan untuk memilih kombinasi hiperparameter yang memberikan akurasi terbaik dengan waktu pelatihan yang lebih efisien, yang sangat penting dalam pengembangan model deep learning untuk aplikasi dunia nyata.