Nama: Andi Cleopatra Maryam Jamila

Nim: 1103213071

Analisis Week 12 Cifar10

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import StepLR
import time
```

```
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Transformasi untuk normalisasi dataset CIFAR-10
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
# Load dataset CIFAR-10
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=1000, shuffle=False)
```

Output:

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170M/170M [00:11<00:00, 14.7MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Analisis:

Kode di atas mempersiapkan dataset CIFAR-10 untuk digunakan dalam pelatihan dan pengujian model deep learning. Dataset CIFAR-10 terdiri dari 60.000 gambar berwarna berukuran 32x32 piksel yang terbagi menjadi 10 kelas, dengan 50.000 gambar untuk pelatihan dan 10.000 gambar untuk pengujian. Transformasi data dilakukan menggunakan torchvision.transforms, yang mencakup konversi gambar menjadi tensor dan normalisasi nilai piksel ke dalam rentang -1 hingga 1, untuk mempercepat konvergensi selama pelatihan. Data kemudian dimuat menggunakan DataLoader untuk mempermudah batch processing dengan ukuran batch 64 untuk pelatihan dan 1.000 untuk pengujian. Dataset ini cocok untuk mengevaluasi model deep learning dalam klasifikasi gambar, dengan transformasi dan pengaturan data yang memastikan model dapat dilatih secara optimal.

```python
# Arsitektur CNN
class CNN_Model(nn.Module):
    def __init__(self, kernel_size, pool_type):
        super(CNN_Model, self).__init__()

        # Convolutional layer dengan ukuran kernel yang berbeda
        if kernel_size == 3:
            self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
            self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
            self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        elif kernel_size == 5:
            self.conv1 = nn.Conv2d(3, 64, 5, padding=2)
            self.conv2 = nn.Conv2d(64, 128, 5, padding=2)
            self.conv3 = nn.Conv2d(128, 256, 5, padding=2)
        elif kernel_size == 7:
            self.conv1 = nn.Conv2d(3, 64, 7, padding=3)
            self.conv2 = nn.Conv2d(64, 128, 7, padding=3)
            self.conv3 = nn.Conv2d(128, 256, 7, padding=3)

        # MaxPooling atau Average Pooling
        if pool_type == 'max':
            self.pool = nn.MaxPool2d(2, 2)
        elif pool_type == 'avg':
            self.pool = nn.AvgPool2d(2, 2)

        # Fully connected layer
        self.fc1 = nn.Linear(256 * 4 * 4, 1024)  # setelah tiga pooling 2x2
        self.fc2 = nn.Linear(1024, 10)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 256 * 4 * 4)  # Flatten tensor
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
# Fungsi untuk melatih dan menguji model
def train_and_evaluate(model, optimizer, criterion, epochs, lr_scheduler=None, early_stopper=None):
    model.to(device)
    train_losses = []
    test_accuracy = []

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

        # Training loop
        for inputs, labels in trainloader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_losses.append(running_loss / len(trainloader))

        # Menghitung akurasi pada set pengujian
        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, labels in testloader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        accuracy = 100 * correct / total
        test_accuracy.append(accuracy)

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

        if lr_scheduler:
            lr_scheduler.step()

        if early_stopper and early_stopper(epoch, accuracy):
            print("Early stopping triggered.")
            break

    return train_losses, test_accuracy
```

```python
# Setup Optimizer dan Scheduler
def get_optimizer_and_scheduler(optimizer_name, model, lr=0.001):
    if optimizer_name == 'sgd':
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
    elif optimizer_name == 'rmsprop':
        optimizer = optim.RMSprop(model.parameters(), lr=lr)
    elif optimizer_name == 'adam':
        optimizer = optim.Adam(model.parameters(), lr=lr)

    scheduler = StepLR(optimizer, step_size=50, gamma=0.1)  # Learning rate scheduler

    return optimizer, scheduler
```

```python
# Callback Early Stopping
class EarlyStopping:
    def __init__(self, patience=10, delta=0):
        self.patience = patience
        self.delta = delta
        self.best_acc = None
        self.counter = 0

    def __call__(self, epoch, accuracy):
        if self.best_acc is None:
            self.best_acc = accuracy
        elif accuracy - self.best_acc < self.delta
            self.counter += 1
            if self.counter >= self.patience:
                return True
        else:
            self.best_acc = accuracy
            self.counter = 0

        return False
```

```python
# Melatih dan menguji model dengan pengaturan yang berbeda
kernel_sizes = [3, 5, 7]
pool_types = ['max', 'avg']
optimizers = ['sgd', 'rmsprop', 'adam']
epochs_list = [5, 50, 100, 250, 350]

for kernel_size in kernel_sizes:
    for pool_type in pool_types:
        for optimizer_name in optimizers:
            for epochs in epochs_list:
                print(f"\nTraining dengan kernel_size={kernel_size}, pool_type={pool_type}, optimizer={optimizer_name}, epochs={epochs}")

                # Inisialisasi model, optimizer, dan scheduler
                model = CNN_Model(kernel_size=kernel_size, pool_type=pool_type)
                optimizer, scheduler = get_optimizer_and_scheduler(optimizer_name, model)
                criterion = nn.CrossEntropyLoss()

                # Early Stopping
                early_stopper = EarlyStopping(patience=10, delta=0.1)

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

                print(f"Training time: {end_time - start_time} detik")
                print(f"Akurasi terbaik pada epoch terakhir: {test_accuracy[-1]}%")
```

Output:

```
Training dengan kernel_size=3, pool_type=max, optimizer=sgd, epochs=5
Epoch 1/5, Loss: 2.1586260650773794, Test Accuracy: 32.19%
Epoch 2/5, Loss: 1.749044663460968, Test Accuracy: 41.98%
Epoch 3/5, Loss: 1.5282290161723067, Test Accuracy: 47.85%
Epoch 4/5, Loss: 1.4012161522265285, Test Accuracy: 51.96%
Epoch 5/5, Loss: 1.3170155464383342, Test Accuracy: 53.42%
Training time: 2221.46337556839 detik
Akurasi terbaik pada epoch terakhir: 53.42%

Training dengan kernel_size=3, pool_type=max, optimizer=sgd, epochs=50
Epoch 1/50, Loss: 2.1750830965273824, Test Accuracy: 31.34%
Epoch 2/50, Loss: 1.7724833133275553, Test Accuracy: 41.26%
Epoch 3/50, Loss: 1.5420128846412424, Test Accuracy: 47.3%
Epoch 4/50, Loss: 1.4235040510402006, Test Accuracy: 51.19%
Epoch 5/50, Loss: 1.335592928749826, Test Accuracy: 53.52%
Epoch 6/50, Loss: 1.2596959106605072, Test Accuracy: 56.11%
Epoch 7/50, Loss: 1.187791127911614, Test Accuracy: 58.55%
Epoch 8/50, Loss: 1.1172704247714917, Test Accuracy: 60.65%
Epoch 9/50, Loss: 1.0530535072621787, Test Accuracy: 63.35%
Epoch 10/50, Loss: 0.9902292341374985, Test Accuracy: 64.48%
Epoch 11/50, Loss: 0.9362156217360436, Test Accuracy: 67.2%
Epoch 12/50, Loss: 0.8829147091606999, Test Accuracy: 67.41%
Epoch 13/50, Loss: 0.8360465486031359, Test Accuracy: 68.98%
Epoch 14/50, Loss: 0.7906907692056178, Test Accuracy: 68.59%
Epoch 15/50, Loss: 0.7507417230197536, Test Accuracy: 70.83%
Epoch 16/50, Loss: 0.7151548101773957, Test Accuracy: 71.35%
Epoch 17/50, Loss: 0.6711990189979143, Test Accuracy: 71.82%
Epoch 18/50, Loss: 0.6358775275061502, Test Accuracy: 72.68%
Epoch 19/50, Loss: 0.5944585660877435, Test Accuracy: 73.86%
Epoch 20/50, Loss: 0.5587102638943421, Test Accuracy: 73.18%
```

Analisis:

Kode ini mengevaluasi performa model CNN dengan berbagai konfigurasi hiperparameter, termasuk ukuran kernel (3, 5, 7), jenis pooling (max atau average), jenis optimizer (SGD, RMSprop, atau Adam), dan jumlah epoch (5 hingga 350). Model CNN dirancang dengan tiga lapisan konvolusi yang diikuti oleh pooling untuk ekstraksi fitur dan dua fully connected layer untuk klasifikasi gambar dari dataset CIFAR-10. Early stopping diterapkan untuk menghentikan pelatihan jika akurasi tidak meningkat dalam beberapa epoch berturut-turut, sementara scheduler digunakan untuk menyesuaikan learning rate secara bertahap. Hasil pelatihan mencatat metrik seperti train loss dan akurasi pada data pengujian, yang digunakan untuk membandingkan efektivitas setiap kombinasi hiperparameter. Eksperimen ini membantu mengidentifikasi konfigurasi terbaik yang menghasilkan akurasi tinggi dengan waktu pelatihan yang efisien, sehingga mendukung pengembangan model yang optimal untuk klasifikasi gambar.