Laporan Tugas 12

1. Library yang Digunakan

python

Copy code

```
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, random_split
from torch.optim.lr_scheduler import StepLR
import torch.nn.functional as F
```

Penjelasan mengenai fungsi masing-masing library:

- **torch & torch.nn**: Untuk membangun arsitektur jaringan saraf dan proses pelatihan.
- **torch.optim**: Mengandung algoritma optimasi seperti SGD, Adam, dan RMSProp.
- **torchvision**: Menyediakan dataset standar seperti CIFAR-10, serta alat transformasi gambar.
- **transforms**: Digunakan untuk transformasi data, misalnya normalisasi gambar ke rentang tertentu.
- **DataLoader**: Untuk memuat data dalam batch secara efisien.
- **StepLR**: Scheduler untuk menurunkan *learning rate* secara bertahap selama pelatihan.

2. Implementasi Early Stopping

python

Copy code

```
class EarlyStopping:
    def __init__(self, patience=5, delta=0):
        # Logika untuk early stopping
```

```
def __call__(self, val_loss):
    # Proses untuk menghentikan pelatihan
```

Fungsi Early Stopping:

- *Early Stopping* digunakan untuk menghentikan pelatihan lebih awal ketika validasi loss tidak menunjukkan perbaikan selama sejumlah epoch.
- **Manfaatnya**: Menghindari overfitting dan menghemat waktu pelatihan.

3. Dataset CIFAR-10

python

Copy code

```
def load_data(batch_size=64):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    return train_loader, val_loader, test_loader
```

Deskripsi Dataset:

- **CIFAR-10** terdiri dari 60.000 gambar berukuran 32x32 piksel dengan 10 kategori.
- **Transformasi Data**:
    - *ToTensor()* mengubah gambar menjadi tensor PyTorch.
    - *Normalize()* untuk normalisasi piksel ke rentang [-1, 1].
- Dataset dibagi menjadi data latih (80%) dan validasi (20%) menggunakan *random_split*.

4. Arsitektur CNN

python

Copy code

```
class CNN(nn.Module):
    def __init__(self, kernel_size=3, pooling='max'):
        super(CNN, self).__init__()
        # Definisi layer
```

```
def forward(self, x):
    # Proses forward
```

Detail Arsitektur:

- **Layer Convolutional**:
  - conv1: Layer pertama dengan 32 filter.
  - conv2: Layer kedua dengan 64 filter.
  - Setiap layer convolution menggunakan fungsi aktivasi ReLU.
- **Pooling**: Bisa memilih antara *MaxPooling* atau *AveragePooling*.
- **Fully Connected Layers**:
  - fc1: Menghubungkan hasil convolution ke 128 neuron.
  - fc2: Layer akhir untuk prediksi 10 kelas.

5. Fungsi Pelatihan

python

Copy code

```
def train_model(model, train_loader, val_loader, optimizer, scheduler, criterion, num_epochs=50, patience=5):
    # Logika pelatihan model
```

Komponen Utama:

- **Model**: CNN yang akan dilatih.
- **Optimizer**: Menggunakan algoritma optimasi seperti SGD, RMSProp, atau Adam.
- **Scheduler**: Mengatur perubahan *learning rate*.
- **Early Stopping**: Penghentian pelatihan dilakukan jika validasi loss tidak membaik setelah beberapa epoch.

Proses:

1. **Training**: Pelatihan model pada data latih dan memperbarui bobot menggunakan optimizer.
2. **Validation**: Mengevaluasi model dengan data validasi untuk mengukur loss dan memutuskan early stopping.

6. Evaluasi Model

python

Copy code

```
def evaluate_model(model, test_loader):
    # Evaluasi pada test set
```

Tujuan:

- Evaluasi dilakukan pada *test set* yang belum pernah dilihat model.
- Menghitung akurasi prediksi.

7. Optimizer

Tiga jenis optimizer yang diuji:

1. **SGD**:

python

Copy code

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

   - Keunggulan: Cepat dan sederhana.
   - Kekurangan: Rentan terhadap jebakan local minima.

2. **RMSProp**:

python

Copy code

```
optimizer = optim.RMSprop(model.parameters(), lr=0.01, alpha=0.9)
```

   - Keunggulan: Menyesuaikan *learning rate* secara dinamis.
   - Kekurangan: Sensitif terhadap pengaturan hyperparameter.

3. **Adam**:

python

Copy code

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

   - Keunggulan: Kombinasi terbaik antara momentum dan adaptasi *learning rate*.
   - Kekurangan: Bisa konvergen terlalu cepat ke solusi suboptimal.

8. Learning Rate Scheduler

python

Copy code

scheduler = StepLR(optimizer, step_size=10, gamma=0.1)

- Scheduler ini menurunkan *learning rate* setiap 10 epoch dengan faktor 0.1 untuk membantu stabilitas konvergensi setelah fase awal pembelajaran.

9. Analisis dan Performa

Hasil yang perlu dicatat setelah menjalankan eksperimen:

1. **Akurasi pada Test Set**: Perbandingan akurasi model dengan berbagai optimizer.
2. **Grafik Loss**: Visualisasi *train loss* dan *validation loss* untuk melihat pola konvergensi.
3. **Durasi Pelatihan**: Perbandingan waktu pelatihan untuk melihat efisiensi masing-masing optimizer.

Kesimpulan

1. **Arsitektur CNN**:
    - Terdiri dari dua layer convolutional dengan ReLU dan pooling.
    - Layer fully connected digunakan untuk klasifikasi.
2. **Eksperimen Hyperparameter**:
    - Ukuran kernel diuji dengan 3x3 (default), 5x5, dan 7x7.
    - *MaxPooling* vs *AveragePooling*.
    - Tiga jenis optimizer: SGD, RMSProp, dan Adam.
    - Jumlah epoch diuji pada 5, 50, 100, 250, dan 350.
3. **Hasil**:
    - Visualisasi akurasi dan loss untuk semua kombinasi hyperparameter.
    - Laporan durasi pelatihan dan konvergensi tiap optimizer.

Dengan struktur yang lebih beragam ini, konten tetap serupa tetapi lebih bervariasi.