Nama: M.Rizky Fadillah (1103210259)

Kelas: TK-45-GAB

LAPORAN TUGAS 12 PEMBELAJARAN MESIN

- Cifar10

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate
# Load and split dataset
def get_data_loaders(batch_size=64, val_split=0.1):
transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
# Split train dataset into train and validation
val_size = int(len(dataset) * val_split)
train_size = len(dataset) - val_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
return train_loader, val_loader, test_loader
# Define CNN model
def create_cnn(kernel_size, pooling_type):
if pooling_type == 'max':
pooling_layer = nn.MaxPool2d(kernel_size=2, stride=2)
elif pooling_type == 'avg':
pooling_layer = nn.AvgPool2d(kernel_size=2, stride=2)
else:
raise ValueError("Pooling type must be 'max' or 'avg'")
model = nn.Sequential(
nn.Conv2d(3, 32, kernel_size=kernel_size, padding=kernel_size // 2),
nn.ReLU(),
```

```
pooling_layer,
nn.Conv2d(32, 64, kernel_size=kernel_size, padding=kernel_size // 2),
nn.ReLU(),
pooling_layer,
nn.Flatten(),
nn.Linear(64 * 8 * 8, 128),
nn.ReLU(),
nn.Linear(128, 10)
)
return model
# Train and evaluate the model
def train_model(model, train_loader, val_loader, optimizer_type, epochs, early_stop_patience):
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
if optimizer_type == 'SGD':
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
elif optimizer_type == 'RMSProp':
optimizer = optim.RMSprop(model.parameters(), lr=0.01)
elif optimizer_type == 'Adam':
optimizer = optim.Adam(model.parameters(), lr=0.001)
else:
raise ValueError("Optimizer must be 'SGD', 'RMSProp', or 'Adam'")
criterion = nn.CrossEntropyLoss()
best_accuracy = 0
patience_counter = 0
for epoch in range(epochs):
model.train()
running_loss = 0.0
for images, labels in train_loader:
images, labels = images.to(device), labels.to(device)
optimizer.zero_grad()
outputs = model(images)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()
# Validate
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
for images, labels in val_loader:
images, labels = images.to(device), labels.to(device)
outputs = model(images)
_, preds = torch.max(outputs, 1)
all_preds.extend(preds.cpu().numpy())
all_labels.extend(labels.cpu().numpy())
```

```
accuracy = accuracy_score(all_labels, all_preds)
print(f"Epoch {epoch + 1}/{epochs}, Loss: {running_loss:.4f}, Val Accuracy: {accuracy:.4f}")
# Early Stopping
if accuracy > best_accuracy:
best_accuracy = accuracy
patience_counter = 0
else:
patience_counter += 1
if patience_counter >= early_stop_patience:
print("Early stopping triggered.")
break
return best_accuracy
# Main function to run the configurations
def run_experiments():
kernel_sizes = [3, 5, 7]
pooling_types = ['max', 'avg']
optimizers = ['SGD', 'RMSProp', 'Adam']
epochs_list = [5, 50, 100, 250, 350]
early_stop_patience = 5
train_loader, val_loader, test_loader = get_data_loaders()
results = []
for kernel_size in kernel_sizes:
for pooling_type in pooling_types:
for optimizer in optimizers:
for epochs in epochs_list:
print("======================================================================")
print(f"\nTesting Config: Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={optimizer},
Epochs={epochs}")
model = create_cnn(kernel_size, pooling_type)
accuracy = train_model(model, train_loader, val_loader, optimizer, epochs,
early_stop_patience)
results.append([kernel_size, pooling_type, optimizer, epochs, accuracy])
# Convert results to DataFrame and save to CSV
results_df = pd.DataFrame(results, columns=['Kernel Size', 'Pooling', 'Optimizer', 'Epochs', 'Validation
Accuracy'])
results_df.to_csv('experiment_cifar10.csv', index=False)
print("Results saved to experiment_cifar10.csv")
return results_df
# Run experiments
results_df = run_experiments()
```

Kode ini adalah implementasi eksperimen untuk menguji performa berbagai konfigurasi model Convolutional Neural Network (CNN) pada dataset CIFAR-10. Tujuannya adalah membandingkan akurasi validasi berdasarkan beberapa parameter seperti ukuran kernel, jenis pooling, jenis optimizer, jumlah epoch, dan mekanisme *early stopping*. Dataset CIFAR-10

digunakan untuk klasifikasi gambar ke dalam 10 kelas, seperti pesawat, mobil, burung, dan sebagainya.

1. Pustaka yang Digunakan

Kode ini menggunakan pustaka berikut:

• **PyTorch**: Untuk membangun, melatih, dan mengevaluasi model CNN.
• **torchvision**: Untuk memuat dataset CIFAR-10 dan melakukan transformasi data.
• **scikit-learn (accuracy_score)**: Untuk menghitung akurasi prediksi.
• **Pandas**: Untuk menyimpan hasil eksperimen dalam format tabel.
• **Matplotlib**: Untuk visualisasi data jika diperlukan.
• **Tabulate**: Mempermudah menampilkan hasil dalam bentuk tabel di konsol.

2. Fungsi-fungsi Utama

a. get_data_loaders

• **Deskripsi**: Memuat dataset CIFAR-10, menerapkan transformasi, dan membagi dataset menjadi train, validation, dan test.
• **Parameter**:
o batch_size: Ukuran batch untuk DataLoader (default 64).
o val_split: Proporsi data validasi terhadap data training (default 0.1).
• **Output**:
o train_loader: DataLoader untuk data training.
o val_loader: DataLoader untuk data validasi.
o test_loader: DataLoader untuk data pengujian.
• **Proses**:
o Dataset CIFAR-10 diunduh dan ditransformasikan dengan normalisasi dan konversi ke tensor.
o Dataset training dibagi menjadi data training dan validasi.

b. create_cnn

• **Deskripsi**: Membuat arsitektur model CNN dengan parameter konfigurasi tertentu.
• **Parameter**:
o kernel_size: Ukuran kernel untuk lapisan konvolusi.
o pooling_type: Jenis pooling, yaitu 'max' atau 'avg'.
• **Output**:
o Model CNN dengan lapisan-lapisan berikut:
▪ Dua lapisan konvolusi dengan kernel 3x3, 5x5, atau 7x7.
▪ Aktivasi ReLU setelah setiap lapisan konvolusi.
▪ Pooling (MaxPooling atau AveragePooling) setelah setiap lapisan konvolusi.
▪ Lapisan fully connected dengan 128 unit dan lapisan output dengan 10 unit (sesuai dengan jumlah kelas pada CIFAR-10).

c. train_model

• **Deskripsi**: Melatih model CNN menggunakan data training dan mengevaluasi performa pada data validasi.
• **Parameter**:
o model: Model CNN yang akan dilatih.
o train_loader: DataLoader untuk data training.
o val_loader: DataLoader untuk data validasi.
o optimizer_type: Jenis optimizer (SGD, RMSProp, atau Adam).

o epochs: Jumlah epoch untuk pelatihan.
o early_stop_patience: Batas kesabaran untuk mekanisme early stopping.
• **Proses**:
o Model dilatih selama beberapa epoch.
o Setelah setiap epoch, akurasi validasi dihitung.
o Jika akurasi validasi tidak meningkat setelah beberapa epoch, proses pelatihan dihentikan lebih awal (early stopping).
• **Output**:
o Akurasi terbaik yang dicapai selama pelatihan.

d. run_experiments

• **Deskripsi**: Mengelola eksperimen dengan mencoba berbagai konfigurasi parameter model.
• **Proses**:
o Parameter yang diuji:
▪ Ukuran kernel: 3, 5, 7.
▪ Jenis pooling: max pooling dan average pooling.
▪ Optimizer: SGD, RMSProp, Adam.
▪ Jumlah epoch: 5, 50, 100, 250, 350.
o Untuk setiap kombinasi parameter, model CNN dilatih menggunakan fungsi train_model.
o Hasil eksperimen (akurasi validasi) dicatat ke dalam list.
o Data hasil eksperimen dikonversi menjadi DataFrame dan disimpan ke file CSV (experiment_cifar10.csv).

3. Hasil dan Output

• Hasil eksperimen disimpan dalam file CSV dengan kolom berikut:
o Kernel Size: Ukuran kernel konvolusi.
o Pooling: Jenis pooling yang digunakan (max atau avg).
o Optimizer: Jenis optimizer yang digunakan.
o Epochs: Jumlah epoch pelatihan.
o Validation Accuracy: Akurasi pada data validasi.
• Selain itu, hasil eksperimen juga ditampilkan di konsol selama pelatihan.

- FashionMnist

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
In [2]:
# Load and split dataset
def get_data_loaders(batch_size=64, val_split=0.1):
transform = transforms.Compose([
transforms.ToTensor(),
```

```
transforms.Normalize((0.5,), (0.5,))
])
dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)
# Split train dataset into train and validation
val_size = int(len(dataset) * val_split)
train_size = len(dataset) - val_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
return train_loader, val_loader, test_loader
In [3]:
# Define CNN model
def create_cnn(kernel_size, pooling_type):
if pooling_type == 'max':
pooling_layer = nn.MaxPool2d(kernel_size=2, stride=2)
elif pooling_type == 'avg':
pooling_layer = nn.AvgPool2d(kernel_size=2, stride=2)
else:
raise ValueError("Pooling type must be 'max' or 'avg'")
model = nn.Sequential(
nn.Conv2d(1, 32, kernel_size=kernel_size, padding=kernel_size // 2),
nn.ReLU(),
pooling_layer,
nn.Conv2d(32, 64, kernel_size=kernel_size, padding=kernel_size // 2),
nn.ReLU(),
pooling_layer,
nn.Flatten(),
nn.Linear(64 * 7 * 7, 128),
nn.ReLU(),
nn.Linear(128, 10)
)
return model
In [4]:
# Train and evaluate the model
def train_model(model, train_loader, val_loader, optimizer_type, epochs, early_stop_patience):
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
if optimizer_type == 'SGD':
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
elif optimizer_type == 'RMSProp':
optimizer = optim.RMSprop(model.parameters(), lr=0.01)
elif optimizer_type == 'Adam':
optimizer = optim.Adam(model.parameters(), lr=0.001)
else:
raise ValueError("Optimizer must be 'SGD', 'RMSProp', or 'Adam'")
```

```
criterion = nn.CrossEntropyLoss()
best_accuracy = 0
patience_counter = 0
for epoch in range(epochs):
model.train()
running_loss = 0.0
for images, labels in train_loader:
images, labels = images.to(device), labels.to(device)
optimizer.zero_grad()
outputs = model(images)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()
# Validate
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
for images, labels in val_loader:
images, labels = images.to(device), labels.to(device)
outputs = model(images)
_, preds = torch.max(outputs, 1)
all_preds.extend(preds.cpu().numpy())
all_labels.extend(labels.cpu().numpy())
accuracy = accuracy_score(all_labels, all_preds)
print(f"Epoch {epoch + 1}/{epochs}, Loss: {running_loss:.4f}, Val Accuracy: {accuracy:.4f}")
# Early Stopping
if accuracy > best_accuracy:
best_accuracy = accuracy
patience_counter = 0
else:
patience_counter += 1
if patience_counter >= early_stop_patience:
print("Early stopping triggered.")
break
return best_accuracy
In [5]:
# Main function to run the configurations
def run_experiments():
kernel_sizes = [3, 5, 7]
pooling_types = ['max', 'avg']
optimizers = ['SGD', 'RMSProp', 'Adam']
epochs_list = [5, 50, 100, 250, 350]
early_stop_patience = 5
train_loader, val_loader, test_loader = get_data_loaders()
results = []
```

```
for kernel_size in kernel_sizes:
for pooling_type in pooling_types:
for optimizer in optimizers:
for epochs in epochs_list:
print("======================================================================")
print(f"\nTesting Config: Kernel={kernel_size}, Pooling={pooling_type},
Optimizer={optimizer}, Epochs={epochs}")
model = create_cnn(kernel_size, pooling_type)
accuracy = train_model(model, train_loader, val_loader, optimizer, epochs,
early_stop_patience)
results.append({
'Kernel Size': kernel_size,
'Pooling': pooling_type,
'Optimizer': optimizer,
'Epochs': epochs,
'Accuracy': accuracy
})
# Convert results to DataFrame and save to CSV
results_df = pd.DataFrame(results)
results_df.to_csv('experiment_FashionMnist.csv', index=False)
print("Results saved to experiment_FashionMnist.csv")
# Run experiments
run_experiments()
```

Fungsi get_data_loaders bertujuan untuk memuat dataset FashionMNIST dan membaginya menjadi tiga subset: data pelatihan, validasi, dan pengujian.
Dataset diproses menggunakan transformasi seperti konversi ke tensor (ToTensor) dan normalisasi dengan nilai rata-rata 0,5 dan standar deviasi 0,5.
Data pelatihan selanjutnya dipecah menggunakan random_split, di mana 10% dari data digunakan sebagai data validasi. Fungsi ini juga mengatur DataLoader untuk setiap subset dengan batch size default 64, yang memungkinkan pemrosesan data secara bertahap dalam pelatihan.

Fungsi create_cnn digunakan untuk membuat arsitektur CNN (Convolutional Neural Network) dengan fleksibilitas pada ukuran kernel dan jenis pooling (maksimum atau rata-rata). Model ini mencakup dua lapisan konvolusi, masing-masing diikuti oleh fungsi aktivasi ReLU dan lapisan pooling, yang diatur berdasarkan parameter masukan.
Setelah lapisan konvolusi, data diratakan untuk diolah melalui dua lapisan fully connected (FC), dengan lapisan output menghasilkan 10 kelas sesuai dengan kategori di dataset FashionMNIST.

Fungsi train_model digunakan untuk melatih model CNN yang telah dibuat. Proses pelatihan mencakup pengoptimalan parameter model menggunakan optimizer yang ditentukan (SGD, RMSProp, atau Adam). Fungsi ini juga menggunakan CrossEntropyLoss sebagai fungsi loss untuk klasifikasi multi-kelas.
Setiap epoch melibatkan proses forward pass, backward pass, dan pembaruan parameter model. Pada akhir setiap epoch, model dievaluasi pada data validasi, dan akurasi dihitung menggunakan

accuracy_score dari scikit-learn.
Early stopping diterapkan untuk menghentikan pelatihan lebih awal jika akurasi validasi tidak meningkat selama sejumlah epoch berturut-turut, sesuai dengan nilai early_stop_patience.

Fungsi run_experiments bertujuan untuk menguji berbagai konfigurasi model dengan kombinasi parameter: ukuran kernel (3, 5, 7), jenis pooling (maksimum atau rata-rata), optimizer (SGD, RMSProp, Adam), dan jumlah epoch (5, 50, 100, 250, 350).
Untuk setiap konfigurasi, fungsi ini membuat model baru, melatihnya, dan mencatat akurasi validasi terbaik. Semua hasil disimpan dalam format dictionary, yang kemudian dikonversi menjadi DataFrame dan diekspor ke file CSV. File ini mencatat performa masing-masing konfigurasi untuk analisis lebih lanjut.

Kesimpulan:

Kode ini dirancang untuk mengevaluasi pengaruh berbagai parameter pada performa model CNN dalam tugas klasifikasi gambar menggunakan dataset FashionMNIST. Pendekatan sistematis seperti ini sangat membantu untuk menemukan konfigurasi terbaik yang menghasilkan akurasi validasi tertinggi. Dataset yang sudah diproses, fleksibilitas dalam pemilihan model, serta mekanisme pelatihan yang robust menjadikan kode ini sangat cocok untuk eksperimen pembelajaran mendalam. File CSV yang dihasilkan memungkinkan analisis komprehensif terhadap hasil eksperimen untuk pengambilan keputusan.