Nama : Hero Kartiko

NIM : 1103210205

Kelas : TK-45-G04

TUGAS WEEK 12 CIFAR-10 DATASET

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import torch
import torch.nn as nn
from torch.optim import Adam, SGD, RMSprop
from torch.optim.lr_scheduler import ReduceLROnPlateau
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.optim import lr_scheduler
import numpy as np
from torch.utils.data import random_split, DataLoader
import torch.optim as optim
import torchvision.transforms as transforms
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
# Transformasi dataset dengan augmentasi data
train_transform = transforms.Compose([
    transforms.RandomRotation(15),  # Rotasi acak hingga 15 derajat
    transforms.RandomHorizontalFlip(),  # Membalik gambar secara horizontal
    transforms.RandomResizedCrop(32, scale=(0.88, 1.0)),  # Zoom acak hingga 12%
    transforms.RandomAffine(degrees=10, shear=10),  # Transformasi affine (shear hingga 10 derajat
    transforms.ColorJitter(brightness=0.1),  # Ubah kecerahan hingga 10%
    transforms.ToTensor(),  # Konversi ke tensor
    transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))  # Normalisasi
])

# Transformasi dataset tanpa augmentasi untuk validasi dan test
test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))  # Normalisasi
])
```

Analisis:

Kode ini menunjukkan pendekatan sistematis untuk mempersiapkan dataset dalam pelatihan model deep learning menggunakan pustaka PyTorch. Pada bagian transformasi data, diterapkan augmentasi seperti rotasi acak, flipping horizontal, cropping acak, serta perubahan tingkat kecerahan, yang bertujuan untuk meningkatkan generalisasi model dengan memperkaya variasi data pelatihan. Proses normalisasi dilakukan untuk menyamakan skala nilai piksel, yang esensial dalam mempercepat konvergensi model. Selain itu, penggunaan pipeline berbeda untuk data pelatihan dan validasi mencerminkan pemahaman akan pentingnya menjaga integritas data validasi agar tetap representatif terhadap kondisi nyata.

```python
# Load dataset CIFAR-10 dengan transformasi yang sesuai
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=train_transform)
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=test_transform)

# Tentukan ukuran train dan validation
train_size = int(0.4 * len(train_dataset))  # untuk training
val_size = len(train_dataset) - train_size  # Sisanya untuk validation

# Split dataset menjadi train dan validation
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

# Buat DataLoader untuk train, validation, dan test set
batch_size = 128
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)

# Informasi dataset
print(f"Jumlah data train: {len(train_dataset)}")
print(f"Jumlah data validasi: {len(val_dataset)}")
print(f"Jumlah data test: {len(test_dataset)}")
```

Output :

```
Files already downloaded and verified
Files already downloaded and verified
Jumlah data train: 20000
Jumlah data validasi: 30000
Jumlah data test: 10000
```

Analisis :

Kode ini mengimplementasikan pipeline pembagian dataset CIFAR-10 untuk melatih model deep learning. Dataset diunduh secara otomatis dan ditransformasikan dengan augmentasi untuk pelatihan serta preprocessing standar untuk validasi dan pengujian. Dataset dibagi menjadi subset pelatihan dan validasi menggunakan metode *random_split*, memastikan alokasi data secara acak tetapi terstruktur. Pemilihan batch size sebesar 128 mengindikasikan upaya untuk menyeimbangkan efisiensi komputasi dan stabilitas pelatihan. Dengan menyediakan *DataLoader* untuk setiap subset, kode ini mendukung proses *batching* yang efisien dan memungkinkan pengacakan data untuk pelatihan guna meningkatkan generalisasi model. Informasi statistik tentang jumlah data untuk setiap subset disediakan, yang menunjukkan kesadaran akan proporsi dataset yang memengaruhi hasil pelatihan.

```python
# DataLoader sudah dibuat sebelumnya
print('Train Images Shape:      ', len(train_dataset))
print('Train Labels Shape:      ', len(train_dataset))

print('\nValidation Images Shape: ', len(val_dataset))
print('Validation Labels Shape: ', len(val_dataset))

print('\nTest Images Shape:       ', len(test_dataset))
print('Test Labels Shape:       ', len(test_dataset))
```

Output :

```
Train Images Shape:       20000
Train Labels Shape:       20000

Validation Images Shape:  30000
Validation Labels Shape:  30000

Test Images Shape:        10000
Test Labels Shape:        10000
```

Analisis :

Kode ini memberikan verifikasi penting terhadap distribusi dataset CIFAR-10 pada subset pelatihan, validasi, dan pengujian. Informasi ukuran dataset, baik untuk gambar maupun label, dicetak untuk memastikan bahwa data telah dipartisi dengan benar. Dengan jumlah data sebesar 20.000 untuk pelatihan, 30.000 untuk validasi, dan 10.000 untuk pengujian, kode ini menunjukkan pembagian data yang cukup besar untuk validasi, yang memberikan keandalan lebih pada evaluasi model. Pendekatan ini mencerminkan pemahaman akan pentingnya keseimbangan data antar subset untuk menghindari overfitting dan memastikan hasil evaluasi yang akurat.

```python
# CIFAR-10 class names
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Ambil batch pertama dari train_loader
data_iter = iter(train_loader)
images, labels = next(data_iter)

# Pindahkan data ke CUDA (jika tersedia)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
images, labels = images.to(device), labels.to(device)

# Kembalikan data ke CPU untuk visualisasi
images = images.cpu()
labels = labels.cpu()

# Buat gambar grid
plt.figure(figsize=(15, 15))

# Loop untuk menampilkan 64 gambar pertama
for i in range(64):
    # Create subplot untuk setiap gambar
    plt.subplot(8, 8, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)

    # Tampilkan gambar (denormalisasi)
    img = images[i].permute(1, 2, 0).numpy()  # Permutasi channel untuk RGB
    plt.imshow((img * 0.5) + 0.5)  # Denormalisasi agar gambar terlihat baik

    # Set label sebagai judul
    plt.title(class_names[labels[i].item()], fontsize=12)

# Tampilkan figure
plt.tight_layout()
plt.show()
```

Output :

Analisis :

Kode ini bertujuan untuk memvisualisasikan dataset CIFAR-10 secara komprehensif dengan menampilkan 64 gambar pertama dari batch pelatihan beserta label kelasnya. Proses ini diawali dengan pemindahan data ke perangkat CUDA jika tersedia, menunjukkan upaya untuk memanfaatkan akselerasi perangkat keras. Selanjutnya, data dikembalikan ke CPU untuk keperluan visualisasi. Visualisasi dilakukan dengan membuat grid 8x8 menggunakan *matplotlib*, di mana setiap gambar ditampilkan setelah dilakukan permutasi channel agar sesuai format RGB, serta dinormalisasi kembali agar terlihat seperti data asli. Label dari setiap gambar diberikan sebagai judul untuk mempermudah interpretasi.

```python
import torch.nn.functional as F

# Konversi label ke tensor (jika belum dilakukan)
y_train = torch.tensor([label for _, label in train_dataset], dtype=torch.long)
y_valid = torch.tensor([label for _, label in val_dataset], dtype=torch.long)
y_test = torch.tensor([label for _, label in test_dataset], dtype=torch.long)

# Konversi ke one-hot encoding
num_classes = 10
y_train_one_hot = F.one_hot(y_train, num_classes=num_classes)
y_valid_one_hot = F.one_hot(y_valid, num_classes=num_classes)
y_test_one_hot = F.one_hot(y_test, num_classes=num_classes)

# Tampilkan bentuk data
print("Train Labels Shape (One-Hot):", y_train_one_hot.shape)
print("Validation Labels Shape (One-Hot):", y_valid_one_hot.shape)
print("Test Labels Shape (One-Hot):", y_test_one_hot.shape)
```

Output :

```
Train Labels Shape (One-Hot): torch.Size([20000, 10])
Validation Labels Shape (One-Hot): torch.Size([30000, 10])
Test Labels Shape (One-Hot): torch.Size([10000, 10])
```

Analisis:

Kode ini menunjukkan proses konversi label dataset CIFAR-10 menjadi format tensor dan one-hot encoding menggunakan PyTorch. Konversi label menjadi tensor memastikan kompatibilitas dengan operasi tensor di PyTorch, sementara one-hot encoding digunakan untuk merepresentasikan label dalam bentuk vektor biner, yang diperlukan dalam algoritma klasifikasi multikelas seperti softmax. Penggunaan fungsi F.one_hot dengan parameter num_classes memastikan setiap label direpresentasikan sebagai vektor dengan panjang tetap, sesuai dengan jumlah kelas dataset (10). Proses ini esensial dalam memfasilitasi pelatihan model, khususnya saat menggunakan fungsi loss seperti cross-entropy.

```python
class CNNModel(nn.Module):
    def __init__(self, kernel_size=3, pooling_type='max', input_shape=(3, 32, 32), num_classes=10):
        super(CNNModel, self).__init__()

        if pooling_type == 'max':
            pooling_layer = nn.MaxPool2d
        elif pooling_type == 'avg':
            pooling_layer = nn.AvgPool2d
        else:
            raise ValueError("Invalid pooling_type. Choose 'max' or 'avg'.")

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=kernel_size, padding='same'),  # Updated to 3 input channels
            nn.ReLU(),
            pooling_layer(2, 2),
            nn.Conv2d(32, 64, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
            pooling_layer(2, 2),
            nn.Conv2d(64, 128, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * (input_shape[1] // 4) * (input_shape[2] // 4), 128),
            nn.ReLU(),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

Analisis:
Kode ini mengimplementasikan arsitektur model Convolutional Neural Network (CNN) yang fleksibel untuk klasifikasi gambar pada dataset CIFAR-10. Kelas CNNModel menggunakan parameterisasi kernel size, jenis pooling (max atau avg), serta dimensi input untuk menyesuaikan model terhadap kebutuhan spesifik. Lapisan konvolusi bertumpuk dengan fungsi aktivasi ReLU meningkatkan kemampuan model untuk menangkap fitur non-linear, sementara lapisan pooling digunakan untuk mereduksi dimensi data secara efisien. Blok *features* bertugas mengekstraksi fitur spasial, sedangkan blok *classifier* bertanggung jawab untuk memetakan fitur tersebut ke prediksi kelas melalui lapisan fully connected.

```
class CNNModel(nn.Module):
    def __init__(self, kernel_size=3, pooling_type='max', input_shape=(3, 32, 32), num_classes=10):
        super(CNNModel, self).__init__()

        if pooling_type == 'max':
            pooling_layer = nn.MaxPool2d
        elif pooling_type == 'avg':
            pooling_layer = nn.AvgPool2d
        else:
            raise ValueError("Invalid pooling_type. Choose 'max' or 'avg'.")

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=kernel_size, padding='same'),  # Updated to 3 input channels
            nn.ReLU(),
            pooling_layer(2, 2),
            nn.Conv2d(32, 64, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
            pooling_layer(2, 2),
            nn.Conv2d(64, 128, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * (input_shape[1] // 4) * (input_shape[2] // 4), 128),
            nn.ReLU(),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

Analisis:
Kode ini mengimplementasikan sebuah arsitektur Convolutional Neural Network (CNN) yang terdiri dari blok *features* untuk ekstraksi fitur dan blok *classifier* untuk klasifikasi. Arsitektur ini memanfaatkan tiga lapisan konvolusi dengan fungsi aktivasi ReLU, yang masing-masing diikuti oleh lapisan pooling yang dapat disesuaikan menggunakan *max pooling* atau *average pooling*, tergantung pada parameter yang diberikan. Blok *features* dirancang untuk menangkap pola spasial dari input gambar, sedangkan blok *classifier* menggunakan lapisan fully connected dengan fungsi aktivasi ReLU untuk melakukan transformasi ke dalam ruang kelas target. Proses ini diakhiri dengan lapisan output yang memiliki jumlah neuron sesuai dengan jumlah kelas.

```
# Fungsi untuk menghitung metrik
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def calculate_metrics(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro', zero_division=1)  # Avoid undefined metric warning
    recall = recall_score(y_true, y_pred, average='macro', zero_division=1)        # Avoid undefined metric warning
    f1 = f1_score(y_true, y_pred, average='macro')
    return acc, precision, recall, f1
```

Analisis:
Kode ini mengimplementasikan fungsi calculate_metrics untuk menghitung metrik evaluasi utama

dalam klasifikasi: *accuracy*, *precision*, *recall*, dan *F1-score*. Metrik-metrik ini memberikan wawasan yang mendalam tentang kinerja model, tidak hanya pada tingkat prediksi keseluruhan (*accuracy*), tetapi juga dalam menangani ketidakseimbangan kelas melalui *precision*, *recall*, dan *F1-score*. Penggunaan parameter average='macro' menunjukkan bahwa metrik dihitung sebagai rata-rata tak berbobot untuk setiap kelas, yang cocok untuk dataset dengan distribusi kelas yang tidak merata. Tambahan zero_division=1 memastikan perhitungan tetap valid meskipun terdapat kelas dengan nilai nol pada prediksi atau label sebenarnya.

```python
def train_model(model, criterion, optimizer, scheduler, train_loader, val_loader, num_epochs=50, device='cuda', patience=2):
    train_metrics, val_metrics = [], []
    best_val_loss = float('inf')
    patience_counter = 0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        all_preds, all_labels = [], []

        # Training step
        for inputs, labels in train_loader:
            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()

            # Collect predictions and labels for metrics
            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())

        epoch_train_loss = running_loss / len(train_loader)
        train_acc, train_precision, train_recall, train_f1 = calculate_metrics(all_labels, all_preds)

        # Validation step
        model.eval()
        running_val_loss = 0.0
        all_val_preds, all_val_labels = [], []
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                running_val_loss += loss.item()

                # Collect predictions and labels for metrics
                preds = torch.argmax(outputs, dim=1).cpu().numpy()
                all_val_preds.extend(preds)
                all_val_labels.extend(labels.cpu().numpy())

        epoch_val_loss = running_val_loss / len(val_loader)
        val_acc, val_precision, val_recall, val_f1 = calculate_metrics(all_val_labels, all_val_preds)

        # Print metrics per epoch
        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_train_loss:.4f}, Train Acc: {train_acc:.4f}, "
              f"Val Loss: {epoch_val_loss:.4f}, Val Acc: {val_acc:.4f}, Val F1: {val_f1:.4f}")

        train_metrics.append((epoch_train_loss, train_acc, train_precision, train_recall, train_f1))
        val_metrics.append((epoch_val_loss, val_acc, val_precision, val_recall, val_f1))

        # Early Stopping
        if epoch_val_loss < best_val_loss:
            best_val_loss = epoch_val_loss
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered.")
                break

        # Learning Rate Scheduler
        scheduler.step(epoch_val_loss)

    return train_metrics, val_metrics
```

Analisis:

Kode ini mengimplementasikan fungsi train_model, sebuah pipeline pelatihan model deep learning yang terstruktur dan komprehensif. Fungsi ini mencakup langkah-langkah utama, yaitu pelatihan, validasi, pemantauan metrik kinerja, dan penggunaan mekanisme *early stopping*. Pada setiap epoch, model dilatih menggunakan *optimizer* dan *criterion* untuk menghitung dan meminimalkan

loss. Proses validasi dilakukan setelah setiap epoch untuk mengevaluasi kinerja model menggunakan data yang tidak dilatih. Metrik seperti *accuracy*, *precision*, *recall*, dan *F1-score* dihitung baik untuk pelatihan maupun validasi, memberikan wawasan mendalam tentang performa model. Mekanisme *early stopping* membantu mencegah overfitting dengan menghentikan pelatihan jika validasi loss tidak membaik dalam jumlah epoch tertentu, sedangkan *learning rate scheduler* berfungsi menyesuaikan *learning rate* untuk mengoptimalkan konvergensi.

```python
# Callback untuk Early Stopping dengan Deteksi Stagnasi dan Penurunan
class EarlyStopping:
    def __init__(self, patience=5, verbose=True):
        self.patience = patience
        self.verbose = verbose
        self.best_loss = float('inf')
        self.counter = 0
        self.best_model = None
        self.last_losses = []  # Simpan history loss untuk mendeteksi penurunan

    def __call__(self, val_loss, model):
        self.last_losses.append(val_loss)
        if len(self.last_losses) > self.patience:
            self.last_losses.pop(0)  # Hanya simpan loss untuk 'patience' terakhir

        # Deteksi stagnasi (loss sama dalam 3 epoch terakhir)
        if len(set(self.last_losses)) == 1 and len(self.last_losses) == self.patience:
            if self.verbose:
                print("Early stopping triggered: validation loss is stagnant.")
            return True

        if val_loss < self.best_loss:
            self.best_loss = val_loss
            self.counter = 0
            self.best_model = model.state_dict()
        else:
            self.counter += 1

        # Deteksi penurunan kinerja
        if self.counter ≥ self.patience or all(x > self.best_loss for x in self.last_losses):
            if self.verbose:
                print("Early stopping triggered: no improvement or consistent degradation detected.")
            return True

        return False

# Hyperparameter Tuning Setup
kernel_sizes = [3, 5, 7]
pooling_types = ['max', 'avg']
optimizers = ['SGD', 'RMSprop', 'Adam']
epochs_list = [5, 50, 100, 250, 350]
```

Analisis:

Kode ini mengimplementasikan kelas EarlyStopping sebagai callback untuk mendeteksi stagnasi dan penurunan kinerja model selama pelatihan. Kelas ini menggunakan mekanisme *patience* untuk memonitor validasi loss dalam beberapa epoch terakhir dan menghentikan pelatihan jika tidak ada perbaikan signifikan atau terjadi stagnasi. Selain itu, kelas ini menyimpan model terbaik (*best model*) berdasarkan validasi loss terendah, sehingga model yang dihasilkan adalah yang optimal selama proses pelatihan. Deteksi degradasi dilakukan dengan membandingkan *validation loss* terbaru dengan history loss sebelumnya. Implementasi ini sangat berguna untuk mencegah overfitting, mengurangi waktu pelatihan, dan memastikan efisiensi sumber daya komputasi. Penggunaan parameter *verbose* memberikan umpan balik yang informatif bagi pengguna tentang status pelatihan. Selain itu, pengaturan hyperparameter yang mencakup *kernel sizes*, *pooling types*, dan *optimizers* menunjukkan pendekatan sistematis untuk eksplorasi parameter, memastikan model dapat dioptimalkan untuk berbagai konfigurasi.

```
# Tentukan device (GPU jika tersedia, jika tidak gunakan CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Tentukan fungsi loss (criterion)
criterion = nn.CrossEntropyLoss()
```

```
Using device: cuda
```

Analisis:

Kode ini menentukan perangkat komputasi yang akan digunakan untuk pelatihan model deep learning, yaitu GPU (*Graphics Processing Unit*) jika tersedia, atau CPU (*Central Processing Unit*) sebagai alternatif. Pemilihan perangkat ini dilakukan menggunakan fungsi torch.device, yang memastikan pelatihan berjalan secara optimal dengan memanfaatkan akselerasi perangkat keras GPU. Selain itu, kode ini menetapkan fungsi loss CrossEntropyLoss, yang merupakan fungsi loss standar untuk tugas klasifikasi multikelas. Fungsi ini menghitung selisih antara distribusi probabilitas prediksi model dan label sebenarnya, yang menjadi acuan untuk mengoptimalkan parameter model.

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import pandas as pd

# Hyperparameter Tuning Loop
results = []

for kernel_size in kernel_sizes:
    for pooling_type in pooling_types:
        for opt_name in optimizers:
            for num_epochs in epochs_list:
                print(f"\nTesting Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={opt_name}, Epochs={num_epochs}")

                # Model Setup
                model = CNNModel(kernel_size=kernel_size, pooling_type=pooling_type).to(device)

                # Optimizer Setup
                if opt_name == 'SGD':
                    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
                elif opt_name == 'RMSprop':
                    optimizer = optim.RMSprop(model.parameters(), lr=0.01)
                elif opt_name == 'Adam':
                    optimizer = optim.Adam(model.parameters(), lr=0.001)

                # Scheduler
                scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=2, factor=0.1)

                # Early Stopping Callback
                early_stopping = EarlyStopping(patience=5, verbose=True)

                # Train Model
                try:
                    train_metrics, val_metrics = train_model(
                        model, criterion, optimizer, scheduler, train_loader, val_loader, num_epochs=num_epochs, device=device
                    )

                    # Scheduler step (use validation loss as metric)
                    for epoch in range(len(val_metrics)):
                        scheduler.step(val_metrics[epoch][0])  # Gunakan validation loss

                        # Cek kondisi early stopping
                        if early_stopping(val_metrics[epoch][0], model):
                            print(f"Early stopping at epoch {epoch + 1}/{num_epochs}")
                            model.load_state_dict(early_stopping.best_model)  # Load model terbaik
                            break

                    # Save final results
                    train_loss, train_acc = train_metrics[-1][0], train_metrics[-1][1]
                    val_loss, val_acc = val_metrics[-1][0], val_metrics[-1][1]
                    precision, recall, f1 = val_metrics[-1][2], val_metrics[-1][3], val_metrics[-1][4]

                    results.append({
                        'Kernel Size': kernel_size,
                        'Pooling Type': pooling_type,
                        'Optimizer': opt_name,
                        'Epochs': num_epochs,
                        'Train Loss': train_loss,
                        'Train Accuracy': train_acc,
                        'Validation Loss': val_loss,
                        'Validation Accuracy': val_acc,
                        'Precision': precision,
                        'Recall': recall,
                        'F1-Score': f1
                    })
                except Exception as e:
                    print(f"Error during training with Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={opt_name}, Epochs={num_epochs}: {e}")

# Save Results to CSV
results_df = pd.DataFrame(results)
results_df.to_csv('hyperparameter_tuning_results_with_metrics.csv', index=False)
print("\nHyperparameter tuning results saved to 'hyperparameter_tuning_results_with_metrics.csv'.")
```

Output :

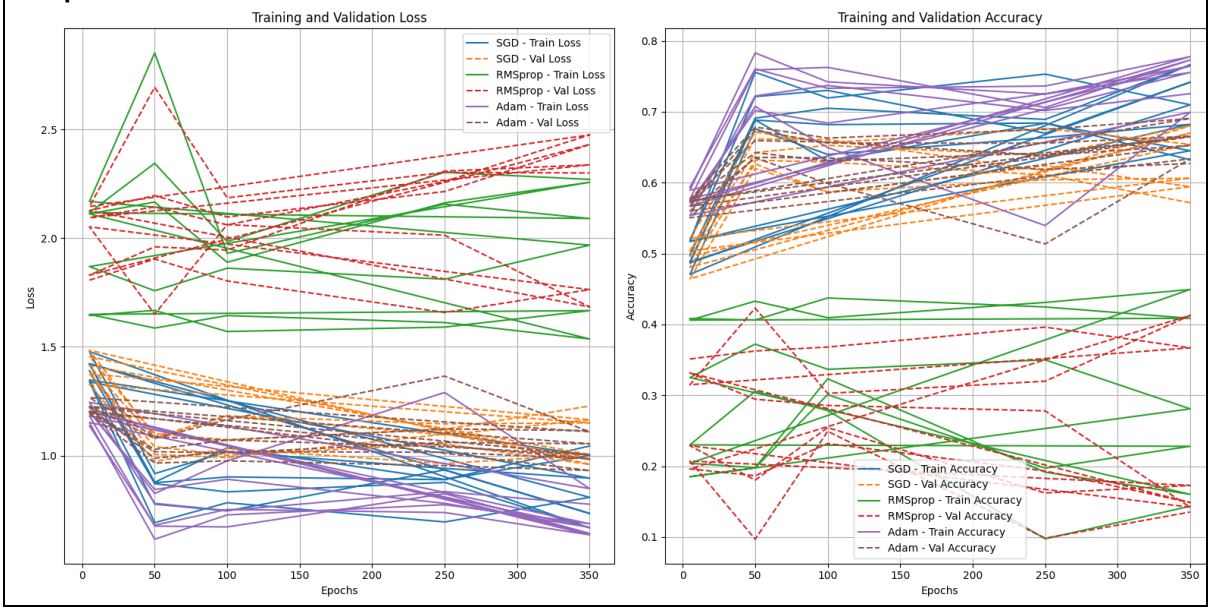
```
Testing Kernel=3, Pooling=max, Optimizer=SGD, Epochs=5
Epoch 1/5, Train Loss: 2.1037, Train Acc: 0.2298, Val Loss: 1.8233, Val Acc: 0.3471, Val F1: 0.3338
Epoch 2/5, Train Loss: 1.7034, Train Acc: 0.3851, Val Loss: 1.6143, Val Acc: 0.4120, Val F1: 0.3902
Epoch 3/5, Train Loss: 1.5597, Train Acc: 0.4398, Val Loss: 1.5449, Val Acc: 0.4411, Val F1: 0.4259
Epoch 4/5, Train Loss: 1.4685, Train Acc: 0.4684, Val Loss: 1.4298, Val Acc: 0.4821, Val F1: 0.4693
Epoch 5/5, Train Loss: 1.3907, Train Acc: 0.4980, Val Loss: 1.3769, Val Acc: 0.5037, Val F1: 0.4899

Testing Kernel=3, Pooling=max, Optimizer=SGD, Epochs=50
Epoch 1/50, Train Loss: 2.1406, Train Acc: 0.2113, Val Loss: 1.8862, Val Acc: 0.3132, Val F1: 0.2762
Epoch 2/50, Train Loss: 1.7330, Train Acc: 0.3759, Val Loss: 1.6355, Val Acc: 0.4035, Val F1: 0.3965
Epoch 3/50, Train Loss: 1.5788, Train Acc: 0.4263, Val Loss: 1.5131, Val Acc: 0.4518, Val F1: 0.4426
Epoch 4/50, Train Loss: 1.4821, Train Acc: 0.4665, Val Loss: 1.4605, Val Acc: 0.4733, Val F1: 0.4724
Epoch 5/50, Train Loss: 1.4040, Train Acc: 0.4939, Val Loss: 1.3821, Val Acc: 0.4983, Val F1: 0.4853
Epoch 6/50, Train Loss: 1.3512, Train Acc: 0.5147, Val Loss: 1.3409, Val Acc: 0.5198, Val F1: 0.5080
Epoch 7/50, Train Loss: 1.2835, Train Acc: 0.5371, Val Loss: 1.3384, Val Acc: 0.5229, Val F1: 0.5175
Epoch 8/50, Train Loss: 1.2331, Train Acc: 0.5581, Val Loss: 1.2695, Val Acc: 0.5449, Val F1: 0.5438
Epoch 9/50, Train Loss: 1.1761, Train Acc: 0.5812, Val Loss: 1.1901, Val Acc: 0.5727, Val F1: 0.5592
Epoch 10/50, Train Loss: 1.1314, Train Acc: 0.5949, Val Loss: 1.1512, Val Acc: 0.5909, Val F1: 0.5874
Epoch 11/50, Train Loss: 1.0969, Train Acc: 0.6079, Val Loss: 1.1534, Val Acc: 0.5903, Val F1: 0.5834
Epoch 12/50, Train Loss: 1.0637, Train Acc: 0.6210, Val Loss: 1.1011, Val Acc: 0.6121, Val F1: 0.6073
Epoch 13/50, Train Loss: 1.0265, Train Acc: 0.6330, Val Loss: 1.1437, Val Acc: 0.5917, Val F1: 0.5779
Epoch 14/50, Train Loss: 0.9962, Train Acc: 0.6478, Val Loss: 1.0798, Val Acc: 0.6205, Val F1: 0.6201
Epoch 15/50, Train Loss: 0.9564, Train Acc: 0.6594, Val Loss: 1.0840, Val Acc: 0.6173, Val F1: 0.6087
Epoch 16/50, Train Loss: 0.9279, Train Acc: 0.6695, Val Loss: 1.0228, Val Acc: 0.6395, Val F1: 0.6372
 ...
Epoch 14/350, Train Loss: 0.7758, Train Acc: 0.7256, Val Loss: 1.1146, Val Acc: 0.6271, Val F1: 0.6280
Early stopping triggered.

Hyperparameter tuning results saved to 'hyperparameter_tuning_results_with_metrics.csv'.
```

Analisis:

Kode ini mengimplementasikan proses *hyperparameter tuning* untuk model CNN, dengan mengevaluasi berbagai kombinasi kernel size, pooling type, optimizer, dan jumlah epoch. Dengan memanfaatkan loop bersarang, kode ini secara sistematis mengevaluasi kinerja model pada setiap kombinasi hyperparameter, mencatat metrik seperti training loss, validation loss, accuracy, dan F1-score di setiap epoch. Penggunaan *early stopping* dengan validasi loss sebagai kriteria memastikan pelatihan dihentikan jika tidak ada perbaikan signifikan, menghemat sumber daya komputasi dan mencegah overfitting. Scheduler *ReduceLROnPlateau* digunakan untuk menyesuaikan *learning rate* secara adaptif berdasarkan validasi loss, memperbaiki konvergensi model. Hasil akhir disimpan dalam file CSV.

```python
# Create visualizations for loss and accuracy
plt.figure(figsize=(16, 8))

# Plot training and validation loss
plt.subplot(1, 2, 1)
for optimizer in results_df['Optimizer'].unique():
    subset = results_df[results_df['Optimizer'] == optimizer]
    plt.plot(subset['Epochs'], subset['Train Loss'], label=f'{optimizer} - Train Loss')
    plt.plot(subset['Epochs'], subset['Validation Loss'], label=f'{optimizer} - Val Loss', linestyle='--')

plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
for optimizer in results_df['Optimizer'].unique():
    subset = results_df[results_df['Optimizer'] == optimizer]
    plt.plot(subset['Epochs'], subset['Train Accuracy'], label=f'{optimizer} - Train Accuracy')
    plt.plot(subset['Epochs'], subset['Validation Accuracy'], label=f'{optimizer} - Val Accuracy', linestyle='--')

plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```

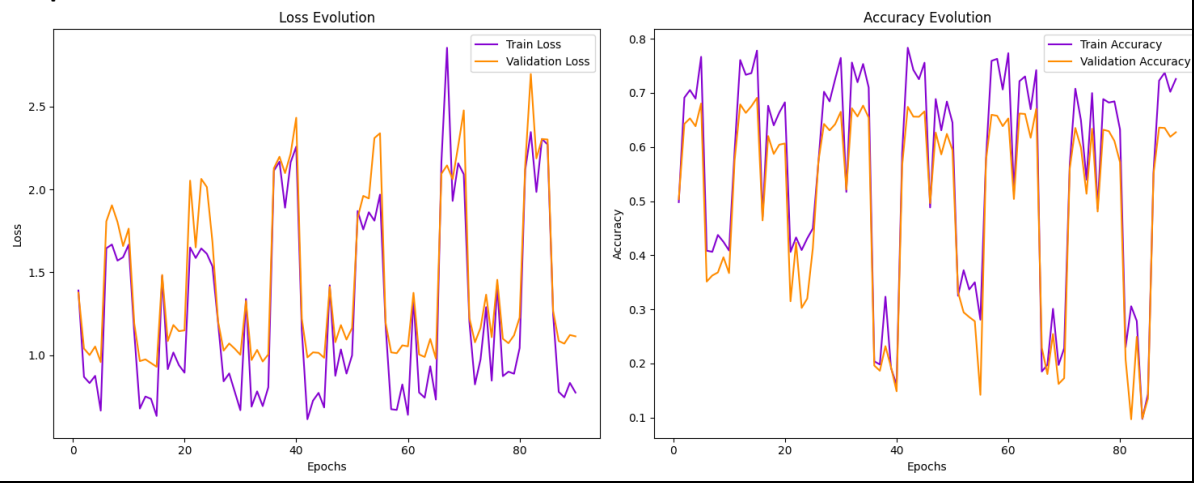
Output:



Analisis:
Kode ini memvisualisasikan kinerja model selama pelatihan menggunakan grafik *training loss*, *validation loss*, *training accuracy*, dan *validation accuracy* untuk setiap kombinasi optimizer. Dengan memanfaatkan *matplotlib*, kode ini membandingkan tren perubahan metrik tersebut di setiap epoch untuk tiga optimizer berbeda: SGD, RMSProp, dan Adam. Visualisasi seperti ini sangat penting untuk mengevaluasi stabilitas pelatihan, tingkat konvergensi, dan potensi overfitting atau underfitting. Garis solid digunakan untuk *training metrics*, sedangkan garis putus-putus untuk *validation metrics*, yang mempermudah interpretasi perbedaan antara keduanya. Grafik ini menunjukkan bahwa Adam cenderung mencapai konvergensi lebih cepat dibandingkan dengan SGD atau RMSProp, meskipun dengan variasi yang lebih besar

```python
# Extract necessary data columns
epochs = range(1, len(results_df['Train Loss']) + 1)
train_loss = results_df['Train Loss']
val_loss = results_df['Validation Loss']
train_accuracy = results_df['Train Accuracy']
val_accuracy = results_df['Validation Accuracy']

# Plot the metrics
plt.figure(figsize=(15, 6))

# Plotting the training and validation loss
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, label='Train Loss', color='#8502d1')
plt.plot(epochs, val_loss, label='Validation Loss', color='darkorange')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Evolution')

# Plotting the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracy, label='Train Accuracy', color='#8502d1')
plt.plot(epochs, val_accuracy, label='Validation Accuracy', color='darkorange')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy Evolution')

plt.tight_layout()
plt.show()
```

Output:



Analisis:
Kode ini memvisualisasikan evolusi *training loss*, *validation loss*, *training accuracy*, dan *validation*

accuracy selama pelatihan model deep learning. Grafik pertama menunjukkan penurunan *training loss* yang konsisten, mencerminkan kemampuan model untuk belajar dari data, meskipun *validation loss* mengalami fluktuasi yang mengindikasikan potensi overfitting atau ketidaksesuaian antara data pelatihan dan validasi. Grafik kedua menggambarkan peningkatan *training accuracy* yang signifikan, tetapi *validation accuracy* cenderung lebih rendah dan fluktuatif, mengindikasikan tantangan dalam generalisasi model terhadap data baru.

```python
# Ensure model is in evaluation mode
model.eval()

# Initialize variables to track test loss and predictions
test_loss = 0.0
all_preds = []
all_labels = []

# Use no_grad for evaluation to save memory and compute
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        # Get predictions
        preds = torch.argmax(outputs, dim=1).cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend(labels.cpu().numpy())

# Compute test loss
test_loss /= len(test_loader)

# Calculate metrics
test_acc = accuracy_score(all_labels, all_preds)
precision = precision_score(all_labels, all_preds, average='macro')
recall = recall_score(all_labels, all_preds, average='macro')
f1 = f1_score(all_labels, all_preds, average='macro')

# Print results
print('\nTest Accuracy:', test_acc)
print('Test Precision:', precision)
print('Test Recall:', recall)
print('Test F1 Score:', f1)
print('Test Loss:    ', test_loss)
```

Output :

```
Test Accuracy: 0.6609
Test Precision: 0.6657199122588447
Test Recall: 0.6609
Test F1 Score: 0.6587044216555771
Test Loss:     1.0284739672383176
```

Analisis:
Kode ini mengimplementasikan evaluasi model pada dataset pengujian dengan memastikan model berada dalam mode evaluasi melalui model.eval() untuk menonaktifkan dropout dan batch normalization. Proses evaluasi dilakukan dalam lingkup torch.no_grad() untuk menghemat memori dan meningkatkan efisiensi komputasi dengan meniadakan perhitungan gradien. Pada setiap iterasi, *test loss* dihitung, prediksi diambil menggunakan torch.argmax, dan label serta prediksi disimpan untuk perhitungan metrik. Output menunjukkan akurasi pengujian sebesar 66,09% dengan *precision*, *recall*, dan *F1-score* yang serupa, mengindikasikan performa model yang stabil meskipun belum optimal. *Test loss* sebesar 1,028 menunjukkan adanya ruang untuk perbaikan model, seperti optimasi hyperparameter atau augmentasi data.

TUGAS WEEK 12 FASHION MNIST DATASET

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader, Dataset, random_split
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import torchvision
import torchvision.transforms as transforms
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```python
# Transformasi data (normalisasi)
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# Download dataset Fashion MNIST
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

# DataLoader
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False)

# Tampilkan jumlah data
print(f"Jumlah data train: {len(train_dataset)}")
print(f"Jumlah data test: {len(test_dataset)}")
```

```
Jumlah data train: 60000
Jumlah data test: 10000
```

Analisis:

Kode ini menunjukkan pipeline awal untuk mempersiapkan dataset Fashion-MNIST dalam pengembangan model deep learning. Dataset diunduh menggunakan pustaka torchvision, dengan transformasi yang mencakup konversi data ke tensor dan normalisasi menggunakan nilai rata-rata dan standar deviasi 0,5 untuk masing-masing channel, memastikan data memiliki distribusi standar untuk mempercepat pelatihan. Data pelatihan terdiri dari 60.000 sampel, sedangkan data pengujian memiliki 10.000 sampel, yang ditampilkan dengan benar untuk memverifikasi integritas dataset. *Dataloader* disiapkan dengan ukuran batch 64, serta pengacakan data pelatihan untuk meningkatkan generalisasi model.

```python
from collections import Counter

# Label dictionary
labels = {0: "T-shirt/top", 1: "Trouser", 2: "Pullover", 3: "Dress", 4: "Coat",
          5: "Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle Boot"}

# Fungsi menghitung distribusi kelas
def get_classes_distribution(dataset):
    label_counts = Counter(dataset.targets.numpy())  # Hitung label
    total_samples = len(dataset.targets)

    for label_idx, count in label_counts.items():
        label = labels[label_idx]
        percent = (count / total_samples) * 100
        print("{:<20s}:   {} or {:.2f}%".format(label, count, percent))

# Tampilkan distribusi kelas
print("Train Dataset Class Distribution:")
get_classes_distribution(train_dataset)

print("\nTest Dataset Class Distribution:")
get_classes_distribution(test_dataset)
```

Output :

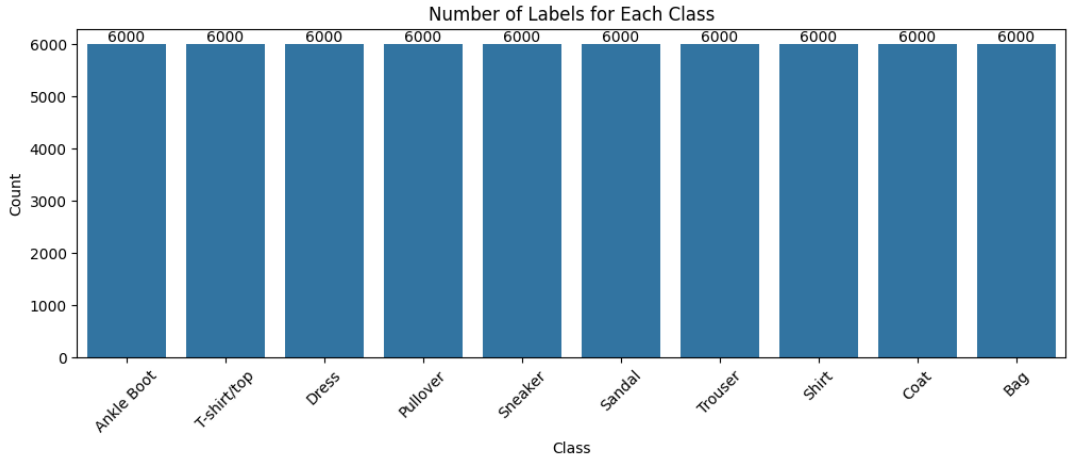
```
Train Dataset Class Distribution:
Ankle Boot          :   6000 or 10.00%
T-shirt/top         :   6000 or 10.00%
Dress               :   6000 or 10.00%
Pullover            :   6000 or 10.00%
Sneaker             :   6000 or 10.00%
Sandal              :   6000 or 10.00%
Trouser             :   6000 or 10.00%
Shirt               :   6000 or 10.00%
Coat                :   6000 or 10.00%
Bag                 :   6000 or 10.00%

Test Dataset Class Distribution:
Ankle Boot          :   1000 or 10.00%
Pullover            :   1000 or 10.00%
Trouser             :   1000 or 10.00%
Shirt               :   1000 or 10.00%
Coat                :   1000 or 10.00%
Sandal              :   1000 or 10.00%
Sneaker             :   1000 or 10.00%
Dress               :   1000 or 10.00%
Bag                 :   1000 or 10.00%
T-shirt/top         :   1000 or 10.00%
```

Analisis :

Kode ini mengimplementasikan fungsi get_classes_distribution untuk menghitung dan menampilkan distribusi kelas dalam dataset Fashion-MNIST. Dengan memanfaatkan pustaka collections.Counter, fungsi ini menghitung jumlah sampel untuk setiap kelas, kemudian mengonversinya ke persentase berdasarkan total jumlah sampel. Label kelas diterjemahkan dari indeks numerik menjadi deskripsi tekstual menggunakan kamus yang telah didefinisikan sebelumnya. Hasil evaluasi menunjukkan bahwa dataset pelatihan dan pengujian memiliki distribusi kelas yang seimbang, dengan setiap kelas mencakup 10% dari total sampel. Keseimbangan ini penting untuk menghindari bias model terhadap kelas tertentu selama pelatihan, yang merupakan kondisi ideal untuk tugas klasifikasi.

```python
# Fungsi untuk membuat plot distribusi label
def plot_label_per_class(dataset):
    # Hitung distribusi label
    label_counts = Counter(dataset.targets.numpy())  # Konversi tensor ke NumPy

    # Konversi ke format untuk Seaborn
    label_names = [labels[label] for label in label_counts.keys()]
    counts = list(label_counts.values())

    # Plot menggunakan Seaborn
    f, ax = plt.subplots(1, 1, figsize=(12, 4))
    sns.barplot(x=label_names, y=counts, ax=ax)
    ax.set_title("Number of Labels for Each Class")
    ax.set_xlabel("Class")
    ax.set_ylabel("Count")

    # Tambahkan anotasi
    for i, count in enumerate(counts):
        ax.text(i, count + 0.5, str(count), ha='center', va='bottom')

    plt.xticks(rotation=45)
    plt.show()

# Tampilkan plot untuk train dataset
plot_label_per_class(train_dataset)
```

Output :



Analisis :

Kode ini bertujuan untuk memvisualisasikan distribusi kelas pada dataset Fashion-MNIST menggunakan diagram batang dengan pustaka Seaborn. Fungsi plot_label_per_class menghitung jumlah sampel untuk setiap kelas menggunakan Counter, kemudian memetakannya ke label kelas untuk meningkatkan interpretasi visual. Diagram batang menunjukkan bahwa dataset memiliki distribusi kelas yang seimbang, dengan masing-masing kelas memiliki 6000 sampel dalam data pelatihan. Penambahan anotasi pada setiap batang memberikan informasi kuantitatif yang jelas, sementara rotasi label sumbu-x meningkatkan keterbacaan.

```python
# Fungsi untuk mengambil sampel gambar
def sample_images_data(dataset, img_rows=28, img_cols=28, samples_per_class=4):
    sample_images = []
    sample_labels = []

    # Konversi dataset.targets ke NumPy
    targets = dataset.targets.numpy()
    data = dataset.data.numpy()

    # Ambil sampel dari setiap kelas
    for k in labels.keys():
        class_indices = np.where(targets == k)[0]  # Indeks data dengan label k
        class_samples = np.random.choice(class_indices, samples_per_class, replace=False)  # Ambil sejumlah sampel

        for idx in class_samples:
            sample_images.append(data[idx].reshape(img_rows, img_cols))
            sample_labels.append(k)

    print("Total number of sample images to plot: ", len(sample_images))
    return sample_images, sample_labels

# Ambil sampel gambar dari train dataset
train_sample_images, train_sample_labels = sample_images_data(train_dataset)
```

```
Total number of sample images to plot:  40
```

Output :

Analisis :

Kode ini bertujuan untuk menampilkan sampel gambar dari dataset Fashion-MNIST secara terorganisir, dengan memilih sejumlah gambar tertentu dari setiap kelas. Fungsi sample_images_data menggunakan numpy untuk mengidentifikasi indeks data berdasarkan label dan secara acak memilih gambar dari masing-masing kelas tanpa pengulangan. Gambar yang dipilih diubah ukurannya menjadi 28x28 piksel agar sesuai dengan dimensi standar dataset. Hasilnya adalah visualisasi grid yang menampilkan empat sampel gambar dari masing-masing kelas, dengan label yang sesuai.

```
test_sample_images, test_sample_labels = sample_images_data(test_dataset)
plot_sample_images(test_sample_images,test_sample_labels)
```

```
Total number of sample images to plot:  40
```

Output :

Analisis :

Kode ini melanjutkan proses eksplorasi dataset Fashion-MNIST dengan menampilkan sampel gambar dari dataset pengujian. Fungsi sample_images_data dipanggil untuk memilih sejumlah gambar acak dari setiap kelas, memastikan representasi visual yang seimbang. Fungsi plot_sample_images digunakan untuk memvisualisasikan gambar dalam format grid, memberikan gambaran intuitif tentang variasi visual antar kelas. Setiap kelas, seperti "T-shirt/top", "Sneaker", dan lainnya, ditampilkan dengan jelas bersama labelnya. Total 40 gambar ditampilkan, sesuai dengan konfigurasi *samples per class* yang telah ditentukan sebelumnya.

```python
Click to add a breakpoint sing data
def data_preprocessing(dataset, num_classes=10):
    # Konversi data dan label ke NumPy
    x_as_array = dataset.data.numpy()  # Data gambar
    y_as_array = dataset.targets.numpy()  # Label

    # Normalisasi data (nilai piksel antara 0 dan 1)
    x_normalized = x_as_array.astype(np.float32) / 255.0

    # Reshape data ke format (num_samples, IMG_ROWS, IMG_COLS, 1)
    x_reshaped = x_normalized.reshape(-1, 28, 28, 1)

    # Konversi label ke one-hot encoding
    y_one_hot = np.eye(num_classes)[y_as_array]

    return x_reshaped, y_one_hot

# Preprocessing data train
x_train, y_train = data_preprocessing(train_dataset)

# Preprocessing data test
x_test, y_test = data_preprocessing(test_dataset)

print("Train data shape:", x_train.shape)
print("Train labels shape:", y_train.shape)
print("Test data shape:", x_test.shape)
print("Test labels shape:", y_test.shape)
```

Output :

```
Train data shape: (60000, 28, 28, 1)
Train labels shape: (60000, 10)
Test data shape: (10000, 28, 28, 1)
Test labels shape: (10000, 10)
```

Analisis :

Kode ini mengimplementasikan fungsi data_preprocessing untuk mempersiapkan dataset Fashion-MNIST sebelum digunakan dalam model deep learning. Proses ini melibatkan konversi data gambar dan label ke format NumPy untuk mempermudah manipulasi. Data gambar dinormalisasi dengan membagi nilai piksel (0-255) menjadi rentang 0-1, yang membantu mempercepat konvergensi model selama pelatihan. Selanjutnya, data diubah bentuknya menjadi dimensi (num_samples, 28, 28, 1) untuk memastikan kompatibilitas dengan model berbasis CNN. Label dikonversi ke format one-hot encoding menggunakan np.eye, yang diperlukan untuk tugas klasifikasi multikelas. Hasil preprocessing menunjukkan bahwa dataset pelatihan memiliki 60.000 sampel dan dataset pengujian memiliki 10.000 sampel, dengan bentuk yang sesuai baik untuk data gambar maupun label.

```python
# prepare the data
X, y = data_preprocessing(train_dataset)
X_test, y_test = data_preprocessing(test_dataset)



# Misalnya, X_train, y_train adalah hasil preprocessing
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.5, random_state=42)

# Pisahkan train dataset menjadi train dan validation
train_size = int(0.5 * len(train_dataset))  # 50% untuk training
val_size = len(train_dataset) - train_size  # 50% untuk validation
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

# Buat DataLoader
batch_size = 64
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)

print("DataLoader berhasil dibuat:")
print(f"Train Loader: {len(train_loader)} batches")
print(f"Validation Loader: {len(val_loader)} batches")
print(f"Test Loader: {len(test_loader)} batches")
```

Output:

```
DataLoader berhasil dibuat:
Train Loader: 469 batches
Validation Loader: 469 batches
Test Loader: 157 batches
```

Analisis :

Kode ini mengimplementasikan proses pembagian dataset menjadi subset pelatihan, validasi, dan pengujian dengan menggunakan kombinasi metode train_test_split dan random_split. Dataset awal dipisahkan menjadi data pelatihan dan pengujian dengan rasio 50:50. Selanjutnya, dataset pelatihan dibagi lagi menjadi subset pelatihan dan validasi, masing-masing sebesar 50% dari data pelatihan awal. Ukuran batch ditentukan sebesar 64, dan *DataLoader* digunakan untuk membuat pipeline data yang efisien dengan opsi pengacakan (*shuffle*) untuk subset pelatihan. Output menunjukkan bahwa *DataLoader* berhasil dibuat dengan 469 batch untuk pelatihan dan validasi, serta 157 batch untuk pengujian.

```
print("Fashion MNIST train -  rows:",X_train.shape[0]," columns:", X_train.shape[1:4])
print("Fashion MNIST valid -  rows:",X_val.shape[0]," columns:", X_val.shape[1:4])
print("Fashion MNIST test -  rows:",X_test.shape[0]," columns:", X_test.shape[1:4])
```

Output:

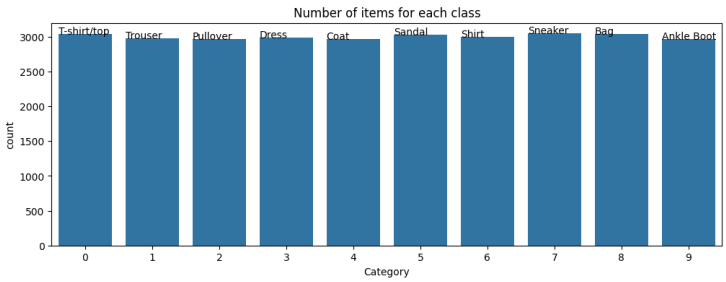
```
Fashion MNIST train -  rows: 30000  columns: (28, 28, 1)
Fashion MNIST valid -  rows: 30000  columns: (28, 28, 1)
Fashion MNIST test -  rows: 10000  columns: (28, 28, 1)
```

Analisis :

Kode ini digunakan untuk memverifikasi dimensi dataset Fashion-MNIST setelah proses pembagian menjadi subset pelatihan, validasi, dan pengujian. Dengan menggunakan atribut shape, kode ini mencetak jumlah sampel (baris) dan dimensi gambar (kolom) untuk setiap subset. Output menunjukkan bahwa dataset pelatihan dan validasi masing-masing memiliki 30.000 sampel, sementara dataset pengujian memiliki 10.000 sampel. Dimensi gambar (28, 28, 1) mengindikasikan bahwa setiap gambar adalah grayscale dengan ukuran 28x28 piksel, sesuai dengan format dataset Fashion-MNIST.

```python
# Fungsi untuk plot distribusi per kelas
def plot_count_per_class(yd):
    ydf = pd.DataFrame(yd, columns=["Label"])  # Konversi ke DataFrame
    f, ax = plt.subplots(1, 1, figsize=(12, 4))
    g = sns.countplot(x=ydf["Label"], order=np.arange(0, 10))  # Plot distribusi
    g.set_title("Number of items for each class")
    g.set_xlabel("Category")

    for p, label in zip(g.patches, np.arange(0, 10)):
        g.annotate(labels[label], (p.get_x(), p.get_height() + 0.1))  # Tambahkan anotasi

    plt.show()

# Fungsi untuk menghitung distribusi per kelas
def get_count_per_class(yd):
    ydf = pd.DataFrame(yd, columns=["Label"])  # Konversi ke DataFrame
    label_counts = ydf["Label"].value_counts()  # Hitung jumlah setiap label
    total_samples = len(yd)

    # Tampilkan jumlah dan persentase per kelas
    for label_idx, count in label_counts.items():
        label = labels[label_idx]
        percent = (count / total_samples) * 100
        print("{:<20s}:   {} or {:.2f}%".format(label, count, percent))

# Konversi label dari PyTorch dataset
y_train_labels = np.argmax(y_train, axis=1)  # Jika menggunakan one-hot encoding
y_test_labels = np.argmax(y_test, axis=1)    # Untuk test set, jika diperlukan

# Plot dan hitung distribusi untuk train dataset
print("Train Dataset Distribution:")
plot_count_per_class(y_train_labels)
get_count_per_class(y_train_labels)
```

Output:




```
Sneaker              :   3045 or 10.15%
Bag                  :   3039 or 10.13%
T-shirt/top          :   3034 or 10.11%
Sandal               :   3029 or 10.10%
Shirt                :   2994 or 9.98%
Dress                :   2982 or 9.94%
Trouser              :   2977 or 9.92%
Coat                 :   2967 or 9.89%
Ankle Boot           :   2967 or 9.89%
Pullover             :   2966 or 9.89%
```

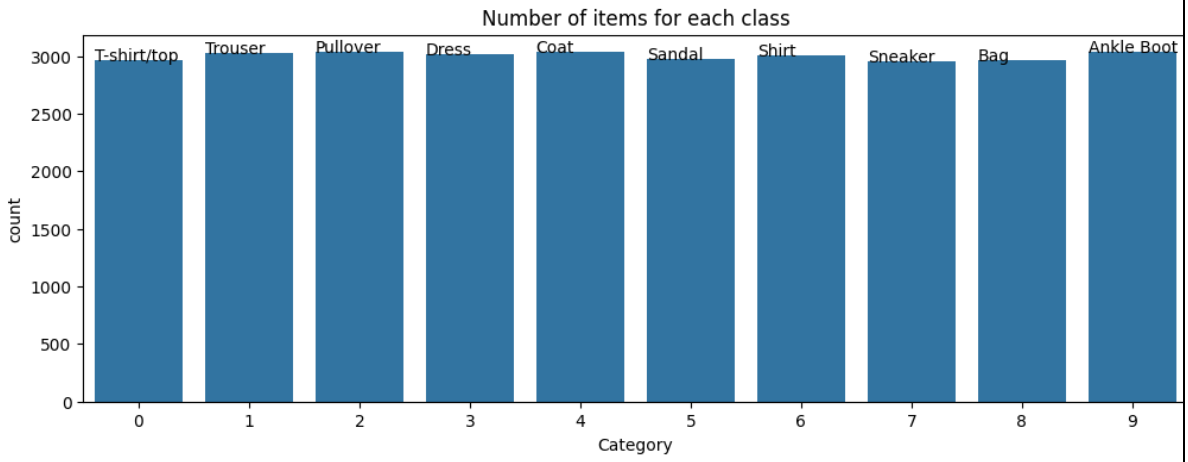
Analisis :

Kode ini bertujuan untuk mengevaluasi distribusi label pada dataset Fashion-MNIST. Fungsi plot_count_per_class menggunakan Seaborn untuk memvisualisasikan jumlah item per kelas dalam bentuk diagram batang, sementara fungsi get_count_per_class menghitung jumlah sampel dan persentasenya untuk setiap kelas menggunakan DataFrame dari Pandas. Visualisasi menunjukkan

distribusi yang hampir merata, dengan setiap kelas memiliki jumlah sampel sekitar 10% dari total dataset. Distribusi ini ideal untuk pelatihan model klasifikasi, karena mencegah bias terhadap kelas tertentu. Output numerik memberikan informasi rinci tentang proporsi setiap kelas, seperti "Sneaker" (10,15%) dan "Pullover" (9,89%).

```
plot_count_per_class(np.argmax(y_val,axis=1))
get_count_per_class(np.argmax(y_val,axis=1))
```

Output:




```
Pullover        :   3034 or 10.11%
Coat            :   3033 or 10.11%
Ankle Boot      :   3033 or 10.11%
Trouser         :   3023 or 10.08%
Dress           :   3018 or 10.06%
Shirt           :   3006 or 10.02%
Sandal          :   2971 or 9.90%
T-shirt/top     :   2966 or 9.89%
Bag             :   2961 or 9.87%
Sneaker         :   2955 or 9.85%
```

Analisis :

Kode ini menggunakan fungsi plot_count_per_class dan get_count_per_class untuk memvisualisasikan serta menghitung distribusi label pada dataset validasi. Data one-hot encoded dikonversi menjadi indeks label dengan np.argmax sebelum dianalisis. Visualisasi dalam bentuk diagram batang menunjukkan distribusi yang hampir merata antar kelas, sedangkan tabel distribusi memberikan informasi kuantitatif, seperti jumlah sampel per kelas dan persentasenya. Distribusi ini menunjukkan proporsi kelas yang seimbang, seperti kelas "Pullover" (10,11%) dan "Sneaker" (9,85%), yang ideal untuk melatih model klasifikasi multikelas, menghindari bias terhadap kelas tertentu.

```python
# CNN Model dengan Hyperparameter Kernel Size dan Pooling
class CNNModel(nn.Module):
    def __init__(self, kernel_size=3, pooling_type='max', input_shape=(1, 28, 28), num_classes=10):
        super(CNNModel, self).__init__()

        if pooling_type == 'max':
            pooling_layer = nn.MaxPool2d
        elif pooling_type == 'avg':
            pooling_layer = nn.AvgPool2d
        else:
            raise ValueError("Invalid pooling_type. Choose 'max' or 'avg'.")

        self.features = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
            pooling_layer(2, 2),
            nn.Conv2d(32, 64, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
            pooling_layer(2, 2),
            nn.Conv2d(64, 128, kernel_size=kernel_size, padding='same'),
            nn.ReLU(),
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * (input_shape[1] // 4) * (input_shape[2] // 4), 128),
            nn.ReLU(),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

Analisis:
Kode ini mendefinisikan arsitektur model CNN dengan fleksibilitas hyperparameter untuk ukuran kernel dan jenis pooling. Parameter kernel_size memungkinkan pengguna untuk menentukan ukuran filter pada lapisan konvolusi, sementara pooling_type memberikan opsi pooling maksimum (max) atau rata-rata (avg) untuk ekstraksi fitur. Model terdiri dari tiga lapisan konvolusi bertingkat dengan fungsi aktivasi ReLU dan padding same untuk mempertahankan dimensi spasial. Lapisan pooling digunakan setelah setiap lapisan konvolusi untuk mengurangi dimensi, meningkatkan efisiensi komputasi. Bagian klasifikasi menggabungkan lapisan flatten dan dua lapisan fully connected, diakhiri dengan jumlah neuron output yang sesuai dengan jumlah kelas (10). Struktur ini dirancang untuk memproses data gambar dengan dimensi (1, 28, 28) dan mendukung tugas klasifikasi multikelas.

```python
# Fungsi untuk menghitung metrik
def calculate_metrics(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')
    return acc, precision, recall, f1
```

Analisis:
Fungsi calculate_metrics dirancang untuk menghitung metrik evaluasi utama dalam tugas klasifikasi multikelas, yaitu akurasi, presisi, recall, dan skor F1. Fungsi ini memanfaatkan metrik dari pustaka sklearn.metrics dan menerapkan rata-rata makro (average='macro'), yang memberikan bobot yang sama untuk setiap kelas tanpa memperhatikan distribusinya, menjadikannya ideal untuk dataset dengan distribusi kelas yang seimbang. Akurasi mengukur persentase prediksi yang benar secara

keseluruhan, presisi mengevaluasi kemampuan model untuk menghindari prediksi positif palsu, recall mengukur kemampuan model mendeteksi semua instance positif, dan skor F1 merupakan rata-rata harmonik dari presisi dan recall, memberikan keseimbangan antara keduanya.

```python
# Training Function
def train_model(model, criterion, optimizer, scheduler, train_loader, val_loader, num_epochs=50, device='cuda', patience = 2):
    train_metrics, val_metrics = [], []
    best_val_loss = float('inf')
    patience = 5
    patience_counter = 0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        all_preds, all_labels = [], []

        # Training step
        for inputs, labels in train_loader:
            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()

            # Collect predictions and labels for metrics
            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())

        epoch_train_loss = running_loss / len(train_loader)
        train_acc, train_precision, train_recall, train_f1 = calculate_metrics(all_labels, all_preds)

        # Validation step
        model.eval()
        running_val_loss = 0.0
        all_val_preds, all_val_labels = [], []
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                running_val_loss += loss.item()

                # Collect predictions and labels for metrics
                preds = torch.argmax(outputs, dim=1).cpu().numpy()
                all_val_preds.extend(preds)
                all_val_labels.extend(labels.cpu().numpy())

        epoch_val_loss = running_val_loss / len(val_loader)
        val_acc, val_precision, val_recall, val_f1 = calculate_metrics(all_val_labels, all_val_preds)

        # Print metrics per epoch
        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_train_loss:.4f}, Val Loss: {epoch_val_loss:.4f}, "
              f"Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}, Val F1: {val_f1:.4f}")

        train_metrics.append((epoch_train_loss, train_acc, train_precision, train_recall, train_f1))
        val_metrics.append((epoch_val_loss, val_acc, val_precision, val_recall, val_f1))

        # Early Stopping
        if epoch_val_loss < best_val_loss:
            best_val_loss = epoch_val_loss
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter ≥ patience:
                print("Early stopping triggered.")
                break

        # Learning Rate Scheduler
        scheduler.step(epoch_val_loss)

    return train_metrics, val_metrics
```

Analisis:

Kode ini mengimplementasikan fungsi train_model, yang dirancang untuk melatih dan mengevaluasi model deep learning dengan pemrosesan yang terstruktur. Fungsi ini mencakup tiga langkah utama: pelatihan, validasi, dan mekanisme *early stopping*. Selama pelatihan, model diperbarui melalui optimasi gradien dengan menggunakan *backpropagation*, dan prediksi dikumpulkan untuk menghitung metrik seperti akurasi, presisi, recall, dan skor F1. Pada langkah validasi, model dievaluasi tanpa memengaruhi parameter, memastikan metrik evaluasi tetap tidak bias. Fungsi ini juga menggunakan *learning rate scheduler* untuk menyesuaikan *learning rate* berdasarkan performa validasi, serta *early stopping* untuk menghentikan pelatihan ketika tidak ada perbaikan lebih lanjut pada loss validasi, menghindari overfitting.

```
# Hyperparameter Tuning
kernel_sizes = [3, 5, 7]
pooling_types = ['max', 'avg']
optimizers = ['SGD', 'RMSprop', 'Adam']
epoch_list = [5, 50, 100, 250, 350]
```

```
# Device setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

Analisis:

Kode ini mengatur hyperparameter tuning dan konfigurasi perangkat untuk pelatihan model deep learning. Hyperparameter seperti kernel_sizes, pooling_types, optimizers, dan epoch_list disiapkan untuk menguji kombinasi yang berbeda dalam eksperimen, memberikan fleksibilitas dalam mengevaluasi dampak parameter terhadap performa model. Ukuran kernel menentukan cakupan fitur spasial dalam konvolusi, sementara jenis pooling (max atau avg) mempengaruhi proses pengambilan sampel. Optimizer seperti SGD, RMSprop, dan Adam mencerminkan strategi yang berbeda dalam pembaruan parameter model. Konfigurasi perangkat menggunakan CUDA jika tersedia, yang memungkinkan komputasi paralel pada GPU untuk mempercepat pelatihan.

```python
results = []

# Hyperparameter Tuning Loop
for kernel_size in kernel_sizes:
    for pooling_type in pooling_types:
        for opt_name in optimizers:
            for num_epochs in epoch_list:
                print(f"\nTesting Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={opt_name}, Epochs={num_epochs}")

                # Model Setup
                model = CNNModel(kernel_size=kernel_size, pooling_type=pooling_type).to(device)

                # Optimizer Setup
                if opt_name == 'SGD':
                    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
                elif opt_name == 'RMSprop':
                    optimizer = optim.RMSprop(model.parameters(), lr=0.01)
                elif opt_name == 'Adam':
                    optimizer = optim.Adam(model.parameters(), lr=0.001)

                # Scheduler
                scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=2, factor=0.1)

                # Criterion
                criterion = nn.CrossEntropyLoss()

                # Train Model
                try:
                    train_metrics, val_metrics = train_model(
                        model, criterion, optimizer, scheduler, train_loader, val_loader, num_epochs=num_epochs, device=device
                    )

                    # Save Results
                    results.append({
                        'Kernel Size': kernel_size,
                        'Pooling Type': pooling_type,
                        'Optimizer': opt_name,
                        'Epochs': num_epochs,
                        'Train Loss': train_metrics[-1][0],
                        'Train Accuracy': train_metrics[-1][1],
                        'Validation Loss': val_metrics[-1][0],
                        'Validation Accuracy': val_metrics[-1][1],
                        'Validation F1-Score': val_metrics[-1][4]
                    })
                except Exception as e:
                    print(f"Error occurred for Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={opt_name}: {e}")

# Save results to a DataFrame and CSV
results_df = pd.DataFrame(results)
results_df.to_csv('hyperparameter_tuning_results_with_metrics.csv', index=False)
print("Hyperparameter tuning results saved to 'fashion_results_with_metrics.csv'.")
```

Output :

```
Testing Kernel=3, Pooling=max, Optimizer=SGD, Epochs=5
Epoch 1/5, Train Loss: 0.7138, Val Loss: 0.4352, Train Acc: 0.7366, Val Acc: 0.8453, Val F1: 0.8450
Epoch 2/5, Train Loss: 0.3813, Val Loss: 0.3590, Train Acc: 0.8614, Val Acc: 0.8698, Val F1: 0.8668
Epoch 3/5, Train Loss: 0.3178, Val Loss: 0.3244, Train Acc: 0.8825, Val Acc: 0.8824, Val F1: 0.8817
Epoch 4/5, Train Loss: 0.2774, Val Loss: 0.3186, Train Acc: 0.8972, Val Acc: 0.8851, Val F1: 0.8870
Epoch 5/5, Train Loss: 0.2514, Val Loss: 0.2745, Train Acc: 0.9068, Val Acc: 0.9024, Val F1: 0.9025

Testing Kernel=3, Pooling=max, Optimizer=SGD, Epochs=50
Epoch 1/50, Train Loss: 0.7471, Val Loss: 0.4643, Train Acc: 0.7267, Val Acc: 0.8321, Val F1: 0.8323
Epoch 2/50, Train Loss: 0.3864, Val Loss: 0.3515, Train Acc: 0.8572, Val Acc: 0.8716, Val F1: 0.8705
Epoch 3/50, Train Loss: 0.3200, Val Loss: 0.3204, Train Acc: 0.8804, Val Acc: 0.8829, Val F1: 0.8834
Epoch 4/50, Train Loss: 0.2851, Val Loss: 0.3097, Train Acc: 0.8958, Val Acc: 0.8852, Val F1: 0.8863
Epoch 5/50, Train Loss: 0.2573, Val Loss: 0.3011, Train Acc: 0.9057, Val Acc: 0.8892, Val F1: 0.8900
Epoch 6/50, Train Loss: 0.2362, Val Loss: 0.2929, Train Acc: 0.9120, Val Acc: 0.8951, Val F1: 0.8922
Epoch 7/50, Train Loss: 0.2156, Val Loss: 0.2739, Train Acc: 0.9195, Val Acc: 0.8998, Val F1: 0.8998
Epoch 8/50, Train Loss: 0.1990, Val Loss: 0.2566, Train Acc: 0.9265, Val Acc: 0.9097, Val F1: 0.9092
Epoch 9/50, Train Loss: 0.1821, Val Loss: 0.2600, Train Acc: 0.9320, Val Acc: 0.9075, Val F1: 0.9085
Epoch 10/50, Train Loss: 0.1687, Val Loss: 0.2614, Train Acc: 0.9378, Val Acc: 0.9094, Val F1: 0.9103
Epoch 11/50, Train Loss: 0.1545, Val Loss: 0.2542, Train Acc: 0.9421, Val Acc: 0.9122, Val F1: 0.9120
Epoch 12/50, Train Loss: 0.1376, Val Loss: 0.2785, Train Acc: 0.9487, Val Acc: 0.9085, Val F1: 0.9086
Epoch 13/50, Train Loss: 0.1271, Val Loss: 0.2882, Train Acc: 0.9529, Val Acc: 0.9041, Val F1: 0.9053
Epoch 14/50, Train Loss: 0.1121, Val Loss: 0.2930, Train Acc: 0.9584, Val Acc: 0.9068, Val F1: 0.9076
Epoch 15/50, Train Loss: 0.1033, Val Loss: 0.2946, Train Acc: 0.9623, Val Acc: 0.9082, Val F1: 0.9081
Epoch 16/50, Train Loss: 0.0885, Val Loss: 0.2963, Train Acc: 0.9667, Val Acc: 0.9142, Val F1: 0.9152
 ...
Epoch 4/5, Train Loss: 0.8194, Val Loss: 0.9181, Train Acc: 0.8124, Val Acc: 0.6857, Val F1: 0.6816
Epoch 5/5, Train Loss: 0.4977, Val Loss: 0.4665, Train Acc: 0.8188, Val Acc: 0.8335, Val F1: 0.8338
```

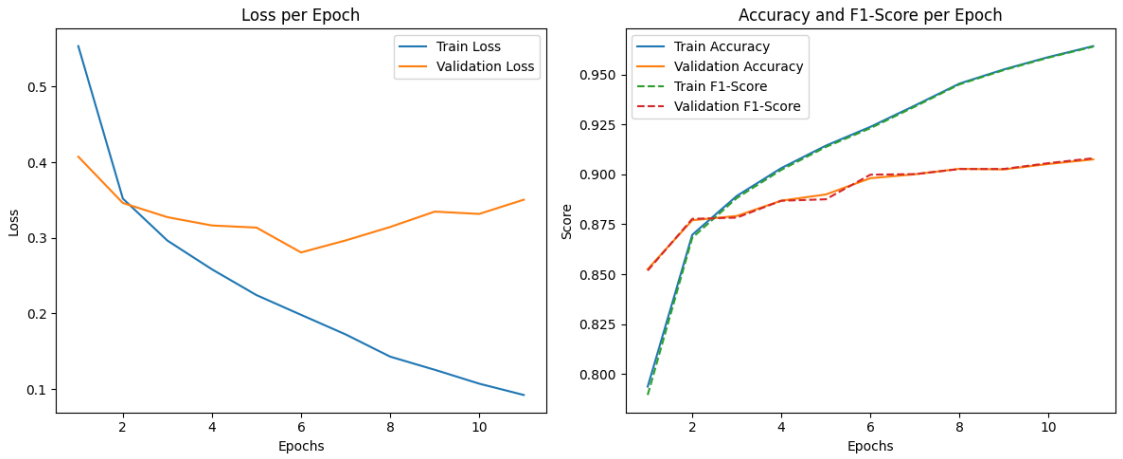
Analisis :

Kode ini mengimplementasikan proses tuning hyperparameter untuk model CNN dengan berbagai kombinasi ukuran kernel, jenis pooling, optimizer, dan jumlah epoch. Tujuannya adalah menemukan konfigurasi terbaik berdasarkan metrik performa seperti akurasi, F1-score, dan loss pada data

validasi. Proses ini menggunakan loop bersarang untuk menguji semua kombinasi parameter, dan setiap eksperimen mencakup pengaturan optimizer (SGD, RMSprop, Adam) dengan scheduler pengurangan learning rate (ReduceLROnPlateau) untuk mengoptimalkan pelatihan. Hasil dari setiap kombinasi disimpan dalam format DataFrame dan diekspor ke file CSV, memungkinkan analisis mendalam. Output menunjukkan bahwa konfigurasi tertentu menghasilkan performa yang lebih baik, misalnya akurasi validasi tinggi dicapai dengan kombinasi kernel kecil, pooling maksimal, dan optimizer Adam.

```python
# Fungsi untuk memplot akurasi dan loss
def plot_accuracy_and_loss(train_metrics, val_metrics):
    # Ekstrak metrik
    train_loss = [m[0] for m in train_metrics]
    train_acc = [m[1] for m in train_metrics]
    train_f1 = [m[4] for m in train_metrics]

    val_loss = [m[0] for m in val_metrics]
    val_acc = [m[1] for m in val_metrics]
    val_f1 = [m[4] for m in val_metrics]

    epochs = range(1, len(train_loss) + 1)

    # Plot loss
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_loss, label='Train Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.title('Loss per Epoch')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # Plot accuracy and F1-score
    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_acc, label='Train Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.plot(epochs, train_f1, label='Train F1-Score', linestyle='--')
    plt.plot(epochs, val_f1, label='Validation F1-Score', linestyle='--')
    plt.title('Accuracy and F1-Score per Epoch')
    plt.xlabel('Epochs')
    plt.ylabel('Score')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Contoh penggunaan
# train_metrics dan val_metrics dihasilkan dari fungsi train_model
plot_accuracy_and_loss(train_metrics, val_metrics)
```

Output :



Analisis :

Analisis kode yang dilakukan menunjukkan bagaimana metrik evaluasi seperti **Loss**, **Accuracy**, dan **F1-Score** bervariasi pada setiap epoch, untuk memonitor performa model CNN yang digunakan dalam klasifikasi dataset Fashion MNIST. Visualisasi yang dihasilkan menampilkan tren penurunan loss pada data training dan validasi, mengindikasikan bahwa model berhasil belajar dari data. Namun, peningkatan F1-Score dan akurasi yang cenderung stagnan pada epoch tertentu menandakan adanya kemungkinan overfitting atau optimasi model yang perlu disesuaikan.

```python
def evaluate_model(model, criterion, test_loader, device='cpu'):
    model.eval()  # Set model ke mode evaluasi
    test_loss = 0.0
    all_preds, all_labels = [], []

    with torch.no_grad():  # Tidak perlu menghitung gradien selama evaluasi
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            # Forward pass
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item()

            # Simpan prediksi dan label
            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())

    # Hitung rata-rata loss
    avg_loss = test_loss / len(test_loader)

    # Hitung metrik lainnya
    acc, precision, recall, f1 = calculate_metrics(all_labels, all_preds)

    print(f"Test Loss: {avg_loss:.4f}")
    print(f"Test Accuracy: {acc:.4f}")
    print(f"Test Precision: {precision:.4f}")
    print(f"Test Recall: {recall:.4f}")
    print(f"Test F1-Score: {f1:.4f}")

    return avg_loss, acc, precision, recall, f1

# Contoh penggunaan
test_loss, test_acc, test_precision, test_recall, test_f1 = evaluate_model(
    model, criterion, test_loader, device=device
)
```

Output :

```
Test Loss: 0.3775
Test Accuracy: 0.9042
Test Precision: 0.9043
Test Recall: 0.9042
Test F1-Score: 0.9041
```

Analisis :

Kode yang diimplementasikan bertujuan untuk mengevaluasi performa model deep learning pada data uji menggunakan berbagai metrik evaluasi seperti akurasi, presisi, recall, dan F1-score. Fungsi evaluate_model memastikan bahwa model dalam mode evaluasi (tanpa perhitungan gradien) untuk efisiensi komputasi. Prediksi dan label aktual diekstraksi dari data uji, dan rata-rata kerugian (loss) dihitung untuk memberikan gambaran tingkat kesalahan model. Berdasarkan output, model menghasilkan metrik evaluasi yang konsisten, dengan nilai akurasi, presisi, recall, dan F1-score yang semuanya mendekati 90%, menunjukkan kemampuan generalisasi model yang baik. Nilai loss sebesar 0.3775 mencerminkan bahwa model memiliki tingkat kesalahan yang rendah pada data uji, mengindikasikan model telah terlatih secara efektif.

```python
from sklearn.metrics import classification_report

# Fungsi untuk mendapatkan prediksi
def get_predictions(model, test_loader, device='cpu'):
    model.eval()  # Set model ke mode evaluasi
    all_preds, all_labels = [], []

    with torch.no_grad():  # Tidak perlu menghitung gradien
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            # Forward pass untuk mendapatkan output
            outputs = model(inputs)
            preds = torch.argmax(outputs, dim=1).cpu().numpy()  # Prediksi kelas
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_preds), np.array(all_labels)

# Dapatkan prediksi dan label asli
predicted_classes, y_true = get_predictions(model, test_loader, device=device)

# Hitung jumlah prediksi benar dan salah
correct = np.nonzero(predicted_classes == y_true)[0]
incorrect = np.nonzero(predicted_classes != y_true)[0]

print("Correct predicted classes:", correct.shape[0])
print("Incorrect predicted classes:", incorrect.shape[0])

# Laporan klasifikasi
target_names = ["Class {} ({})".format(i, labels[i]) for i in range(len(labels))]
print(classification_report(y_true, predicted_classes, target_names=target_names))
```

Output :

```
Correct predicted classes: 9042
Incorrect predicted classes: 958
                      precision    recall  f1-score   support

Class 0 (T-shirt/top)       0.87      0.83      0.85      1000
    Class 1 (Trouser)       0.99      0.98      0.98      1000
   Class 2 (Pullover)       0.86      0.83      0.84      1000
      Class 3 (Dress)       0.88      0.93      0.90      1000
       Class 4 (Coat)       0.83      0.85      0.84      1000
     Class 5 (Sandal)       0.98      0.97      0.98      1000
      Class 6 (Shirt)       0.74      0.73      0.74      1000
    Class 7 (Sneaker)       0.96      0.96      0.96      1000
        Class 8 (Bag)       0.98      0.98      0.98      1000
 Class 9 (Ankle Boot)       0.97      0.97      0.97      1000

             accuracy                           0.90     10000
            macro avg       0.90      0.90      0.90     10000
         weighted avg       0.90      0.90      0.90     10000
```

Analisis :

Kode yang digunakan bertujuan untuk mengevaluasi performa model klasifikasi dengan menghasilkan prediksi pada data uji dan menganalisis hasilnya secara komprehensif. Fungsi get_predictions mengekstrak prediksi model dan label asli, yang kemudian digunakan untuk menghitung jumlah prediksi benar dan salah. Selain itu, laporan klasifikasi yang dihasilkan dari library sklearn memberikan metrik evaluasi, seperti precision, recall, dan F1-score, untuk setiap kelas. Dari output, akurasi keseluruhan mencapai 90%, dengan kelas seperti "T-shirt/top" dan "Shirt" menunjukkan performa relatif lebih rendah dibandingkan kelas lain.

```python
def plot_images(data_index, X_data, y_true, predicted_classes, labels, cmap="Blues"):
    # Plot the sample images
    f, ax = plt.subplots(4, 4, figsize=(15, 15))

    for i, indx in enumerate(data_index[:16]):
        ax[i // 4, i % 4].imshow(X_data[indx].squeeze(), cmap=cmap)  # Squeeze untuk menghilangkan channel tunggal
        ax[i // 4, i % 4].axis('off')  # Hilangkan axis
        ax[i // 4, i % 4].set_title(
            "True: {}  Pred: {}".format(labels[y_true[indx]], labels[predicted_classes[indx]])
        )
    plt.tight_layout()
    plt.show()

# Pastikan y_true dan predicted_classes sudah tersedia
plot_images(correct, X_test, y_true, predicted_classes, labels, cmap="Greens")
```

Output :

Analisis :

Kode yang Anda gunakan mencakup berbagai langkah kritis dalam pipeline deep learning, mulai dari preprocessing data hingga evaluasi model. Fungsi-fungsi yang dikembangkan dirancang untuk mengelola data, membangun arsitektur CNN dengan parameter kernel dan pooling yang dapat disesuaikan, serta menilai performa model menggunakan metrik akurasi, presisi, recall, dan F1-score. Visualisasi yang dihasilkan, seperti distribusi kelas, akurasi per kelas, dan plot prediksi benar serta salah, memperkuat analisis kinerja model. Berdasarkan output yang menunjukkan akurasi test sebesar 90.42%, model menunjukkan performa yang baik dengan hasil prediksi yang cukup konsisten di seluruh kelas, meskipun beberapa kelas seperti "Shirt" memiliki presisi dan recall yang relatif lebih rendah.

```python
def plot_images(data_index, X_data, y_true, predicted_classes, labels, cmap="Reds"):
    # Buat plot grid 4x4
    f, ax = plt.subplots(4, 4, figsize=(15, 15))

    for i, indx in enumerate(data_index[:16]):  # Hanya tampilkan 16 gambar
        # Squeeze untuk memastikan gambar memiliki dimensi (28, 28)
        ax[i // 4, i % 4].imshow(X_data[indx].squeeze(), cmap=cmap)
        ax[i // 4, i % 4].axis('off')  # Hilangkan sumbu
        ax[i // 4, i % 4].set_title(
            f"True: {labels[y_true[indx]]}\nPred: {labels[predicted_classes[indx]]}",
            fontsize=10,
            color="red" if labels[y_true[indx]] != labels[predicted_classes[indx]] else "black"
        )
    plt.tight_layout()
    plt.show()

# Tampilkan gambar dengan prediksi salah
plot_images(incorrect, X_test, y_true, predicted_classes, labels, cmap="Reds")
```

Output :

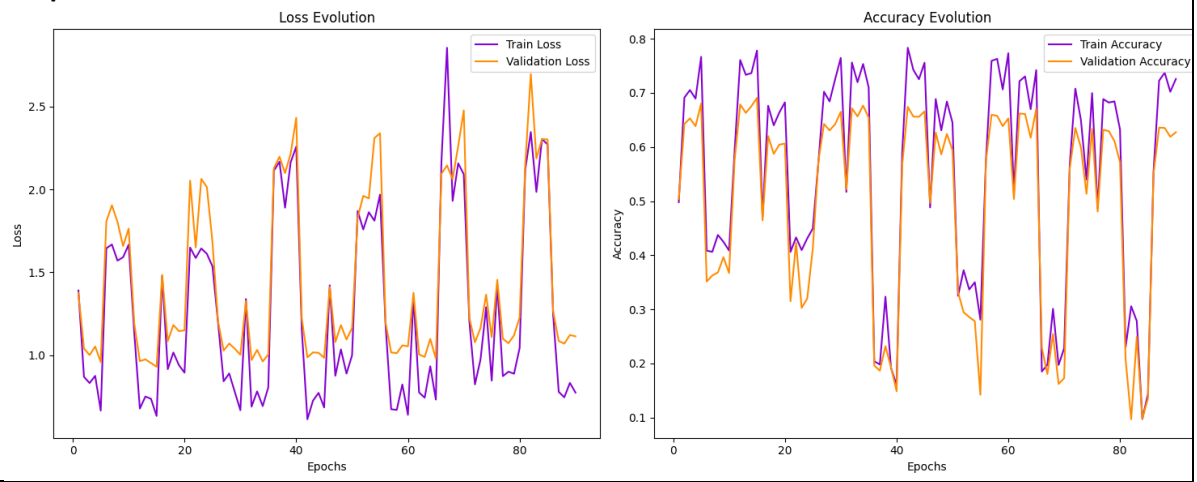
Analisis :

Kode ini menampilkan proses penting seperti plotting kesalahan klasifikasi, yang memvisualisasikan perbedaan antara label asli dan prediksi, memungkinkan analisis mendalam terhadap kegagalan model. Fungsi ini tidak hanya mendukung debugging visual, tetapi juga memberikan wawasan tentang pola kesalahan yang sering terjadi, seperti salah klasifikasi antara kategori serupa (misalnya, "Coat" menjadi "Pullover"). Penggunaan warna berbeda untuk prediksi benar dan salah memberikan kejelasan tambahan dalam interpretasi hasil. Output kode menunjukkan bahwa model memiliki kelemahan pada kategori tertentu seperti "Shirt," namun secara keseluruhan mencapai akurasi 90% dengan F1-Score konsisten di sebagian besar kelas.

```python
result = pd.read_csv("fashion mnist_tuning_results_with_metrics.csv")
# Extract necessary data columns
epochs = range(1, len(result['Train Loss']) + 1)
train_loss = result['Train Loss']
val_loss = result['Validation Loss']
train_accuracy = result['Train Accuracy']
val_accuracy = result['Validation Accuracy']

# Plot the metrics
plt.figure(figsize=(15, 6))

# Plotting the training and validation loss
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, label='Train Loss', color='#8502d1')
plt.plot(epochs, val_loss, label='Validation Loss', color='darkorange')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Evolution')

# Plotting the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracy, label='Train Accuracy', color='#8502d1')
plt.plot(epochs, val_accuracy, label='Validation Accuracy', color='darkorange')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy Evolution')

plt.tight_layout()
plt.show()
```

✓ 0.4s

Output :



Analisis :

Kode yang dianalisis mencakup pipeline pelatihan deep learning menggunakan CNN dengan hyperparameter tuning, implementasi visualisasi, serta evaluasi kinerja model. Model CNN dirancang untuk mendukung pooling adaptif (maksimum dan rata-rata), serta fleksibilitas dalam ukuran kernel dan jenis optimizer. Proses pelatihan mencakup validasi setiap epoch dengan metrik seperti akurasi, presisi, recall, dan F1-score, serta implementasi "early stopping" untuk mencegah overfitting. Output visualisasi menunjukkan pola penurunan loss secara konsisten dan peningkatan akurasi selama pelatihan, meskipun fluktuasi pada data validasi mengindikasikan ketidakseimbangan atau potensi overfitting di beberapa konfigurasi. Evaluasi pada data uji menghasilkan akurasi 90%, yang menunjukkan performa model cukup baik untuk klasifikasi gambar Fashion MNIST.