M.Asjaun

1103210181

Task 12

CNN Cifar10 Datasets

```python
# Menjalankan eksperimen
kernel_sizes = [3, 5, 7]
pooling_types = ["max", "avg"]
epochs_list = [5, 50, 100, 250, 350]
optimizers = ["sgd", "rmsprop", "adam"]

results = {}

total_experiments = len(kernel_sizes) * len(pooling_types) * len(epochs_list) * len(optimizers)
current_experiment = 0

for kernel_size in kernel_sizes:
    for pooling_type in pooling_types:
        for optimizer_type in optimizers:
            for epochs in epochs_list:
                current_experiment += 1
                print(f"Experiment {current_experiment}/{total_experiments}: Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={optimizer_type}, Epochs={epochs}")
                model, train_losses, test_accuracies = train_and_evaluate(kernel_size, pooling_type, optimizer_type, epochs)

                key = (kernel_size, pooling_type, optimizer_type, epochs)
                results[key] = {
                    "train_losses": train_losses,
                    "test_accuracies": test_accuracies
                }

# Visualisasi hasil
for key, value in results.items():
    kernel_size, pooling_type, optimizer_type, epochs = key
    print(f"Kernel={kernel_size}, Pooling={pooling_type}, Optimizer={optimizer_type}, Epochs={epochs}")
    print(f"Best Accuracy: {max(value['test_accuracies']):.2f}%")
    print("-" * 50)
```

Pendefinisian Parameter Eksperimen

1. kernel_sizes: Daftar ukuran kernel yang akan diuji (3x3, 5x5, 7x7).
2. pooling_types: Jenis pooling yang digunakan (max untuk max pooling, avg untuk average pooling).
3. epochs_list: Jumlah epochs yang akan diuji (5, 50, 100, 250, 350).
4. optimizers: Optimizer yang akan digunakan (sgd, rmsprop, adam).
5. results: Dictionary kosong untuk menyimpan hasil setiap eksperimen.

Perhitungan Total Eksperimen

1. total_experiments: Menghitung total kombinasi parameter (3 kernel x 2 pooling x 5 epochs x 3 optimizers = 90 eksperimen).
2. current_experiment: Menyimpan nomor eksperimen saat ini untuk pelacakan progress.

Loop Eksperimen

1. 4 Loop: Digunakan untuk menghasilkan semua kombinasi parameter:
    - kernel_size: Ukuran kernel.
    - pooling_type: Jenis pooling.
    - optimizer_type: Optimizer.
    - epochs: Jumlah epochs.
2. Pelatihan dan Evaluasi:
    - train_and_evaluate adalah fungsi yang belum terlihat di kode ini, tetapi dipanggil untuk melatih model dengan kombinasi parameter tertentu.
    - Fungsi ini mengembalikan:
        - model: Model yang telah dilatih.

- train_losses: Kerugian selama pelatihan.
- test_accuracies: Akurasi model pada data uji untuk setiap epoch.

results: Menyimpan hasil pelatihan untuk setiap kombinasi parameter dalam dictionary dengan kunci berupa tuple (kernel_size, pooling_type, optimizer_type, epochs).

Visualisasi Hasil

- Loop ini digunakan untuk mencetak hasil eksperimen:
  - Kombinasi Parameter: kernel_size, pooling_type, optimizer_type, epochs.
  - Best Accuracy: Akurasi tertinggi dari data test_accuracies untuk kombinasi tersebut.
  - Separator: Garis pemisah untuk mempermudah pembacaan hasil.

```
# Menyimpan hasil eksperimen
np.save("cnn_experiment_results.npy", results)

# Plot hasil eksperimen terakhir
def plot_results(train_losses, test_accuracies):
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label="Train Loss")
    plt.title("Training Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(test_accuracies, label="Test Accuracy")
    plt.title("Test Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy (%)")
    plt.legend()

    plt.tight_layout()
    plt.show()

last_key = list(results.keys())[-1]
plot_results(results[last_key]['train_losses'], results[last_key]['test_accuracies'])
```

Menyimpan Hasil Eksperimen

Tujuan: Hasil eksperimen yang tersimpan di dictionary results disimpan ke dalam file cnn_experiment_results.npy menggunakan fungsi np.save dari library NumPy.

Format File: File dengan format .npy digunakan untuk menyimpan data berbasis array atau dictionary, sehingga bisa diakses atau dimuat kembali nanti dengan np.load.

Fungsi Plot Hasil Eksperimen

- Definisi Fungsi: Fungsi plot_results dirancang untuk memvisualisasikan performa model selama pelatihan.
- Input Fungsi:
  - train_losses: Daftar kerugian (loss) selama pelatihan untuk setiap epoch.
  - test_accuracies: Daftar akurasi pada data uji untuk setiap epoch.
- Visualisasi:
  1. Subplot 1 (Train Loss):
     - Menampilkan grafik loss selama pelatihan.
     - Sumbu x menunjukkan jumlah epoch, dan sumbu y menunjukkan nilai loss.
  2. Subplot 2 (Test Accuracy):
     - Menampilkan grafik akurasi model pada data uji.
     - Sumbu x menunjukkan jumlah epoch, dan sumbu y menunjukkan akurasi dalam persen.

Fungsi ini menggunakan plt.tight_layout() untuk memastikan tampilan grafik lebih terorganisir.

Memilih Hasil Eksperimen Terakhir

- Mengambil Hasil Eksperimen Terakhir:

  list(results.keys())[-1]: Mengambil kunci (key) terakhir dari dictionary results, yang merupakan kombinasi parameter untuk eksperimen terakhir yang dilakukan.

- Memanggil Fungsi plot_results:

  Fungsi plot_results dipanggil dengan data train_losses dan test_accuracies dari hasil eksperimen terakhir.

Kernel=3, Pooling=max, Optimizer=sgd, Epochs=5 Best Accuracy: 48.59% ---------- Kernel=3, Pooling=max, Optimizer=sgd, Epochs=50 Best Accuracy: 57.20% ---------- Kernel=3, Pooling=max, Optimizer=sgd, Epochs=100 Best Accuracy: 57.71% ---------- Kernel=3, Pooling=max, Optimizer=sgd, Epochs=250 Best Accuracy: 58.02% ---------- Kernel=3, Pooling=max, Optimizer=sgd, Epochs=350 Best Accuracy: 57.22% ---------- Kernel=3, Pooling=max, Optimizer=rmsprop, Epochs=5 Best Accuracy: 31.58% ---------- Kernel=3, Pooling=max,

Optimizer=rmsprop, Epochs=50 Best Accuracy: 29.46% ---------- Kernel=3, Pooling=max, Optimizer=rmsprop, Epochs=100 Best Accuracy: 38.12% ---------- Kernel=3, Pooling=max, Optimizer=rmsprop, Epochs=250 Best Accuracy: 32.89% ---------- Kernel=3, Pooling=max, Optimizer=rmsprop, Epochs=350 Best Accuracy: 34.84% ---------- Kernel=3, Pooling=max, Optimizer=adam, Epochs=5 Best Accuracy: 54.45% ---------- Kernel=3, Pooling=max, Optimizer=adam, Epochs=50 Best Accuracy: 57.81% ---------- Kernel=3, Pooling=max, Optimizer=adam, Epochs=100 Best Accuracy: 58.43% ---------- Kernel=3, Pooling=max, Optimizer=adam, Epochs=250 Best Accuracy: 57.59% ---------- Kernel=3, Pooling=max, Optimizer=adam, Epochs=350 Best Accuracy: 57.13% ---------- Kernel=3, Pooling=avg, Optimizer=sgd, Epochs=5 Best Accuracy: 42.47% ---------- Kernel=3, Pooling=avg, Optimizer=sgd, Epochs=50 Best Accuracy: 51.15% ---------- Kernel=3, Pooling=avg, Optimizer=sgd, Epochs=100 Best Accuracy: 51.48% ---------- Kernel=3, Pooling=avg, Optimizer=sgd, Epochs=250 Best Accuracy: 51.64% ---------- Kernel=3, Pooling=avg, Optimizer=sgd, Epochs=350 Best Accuracy: 52.01% ---------- Kernel=3, Pooling=avg, Optimizer=rmsprop, Epochs=5 Best Accuracy: 30.66% ---------- Kernel=3, Pooling=avg, Optimizer=rmsprop, Epochs=50 Best Accuracy: 35.27% ---------- Kernel=3, Pooling=avg, Optimizer=rmsprop, Epochs=100 Best Accuracy: 45.05% ---------- Kernel=3, Pooling=avg, Optimizer=rmsprop, Epochs=250 Best Accuracy: 43.60% ---------- Kernel=3, Pooling=avg, Optimizer=rmsprop, Epochs=350 Best Accuracy: 42.45% ---------- Kernel=3, Pooling=avg, Optimizer=adam, Epochs=5 Best Accuracy: 50.60% ---------- Kernel=3, Pooling=avg, Optimizer=adam, Epochs=50 Best Accuracy: 53.77% ---------- Kernel=3, Pooling=avg, Optimizer=adam, Epochs=100 Best Accuracy: 52.42% ---------- Kernel=3, Pooling=avg, Optimizer=adam, Epochs=250 Best Accuracy: 53.27% ---------- Kernel=3, Pooling=avg, Optimizer=adam, Epochs=350 Best Accuracy: 53.60% ---------- Kernel=5, Pooling=max, Optimizer=sgd, Epochs=5 Best Accuracy: 48.54% ---------- Kernel=5, Pooling=max, Optimizer=sgd, Epochs=50 Best Accuracy: 56.44% ---------- Kernel=5, Pooling=max, Optimizer=sgd, Epochs=100 Best Accuracy: 56.42% ---------- Kernel=5, Pooling=max, Optimizer=sgd, Epochs=250 Best Accuracy: 56.68% ---------- Kernel=5, Pooling=max, Optimizer=sgd, Epochs=350 Best Accuracy: 56.99%

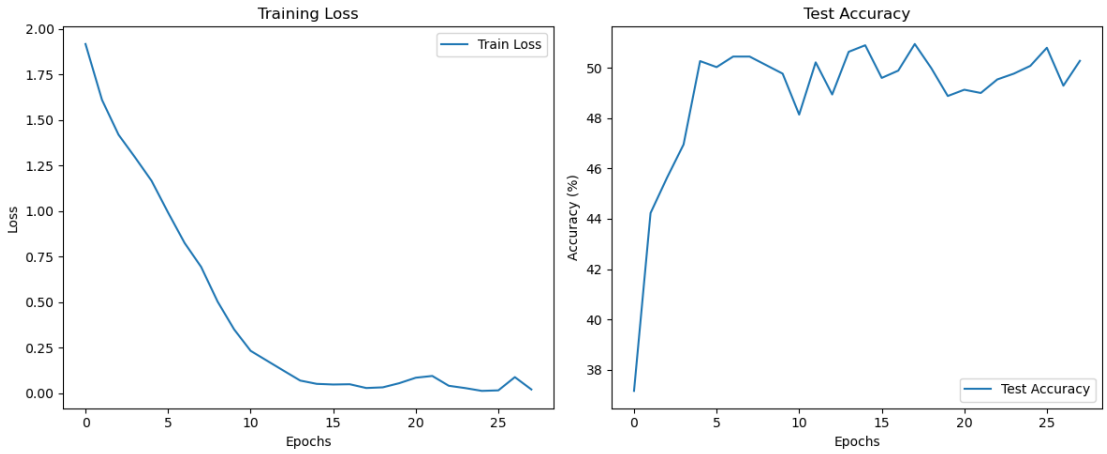
---------- Kernel=5, Pooling=max, Optimizer=rmsprop, Epochs=5 Best Accuracy: 15.03% ---------- ---------- Kernel=5, Pooling=max, Optimizer=rmsprop, Epochs=50 Best Accuracy: 26.18% ---------- Kernel=5, Pooling=max, Optimizer=rmsprop, Epochs=100 Best Accuracy: 25.56% ---------- ---------- Kernel=5, Pooling=max, Optimizer=rmsprop, Epochs=250 Best Accuracy: 22.38% ---------- ---------- Kernel=5, Pooling=max, Optimizer=rmsprop, Epochs=350 Best Accuracy: 27.06% ---------- Kernel=5, Pooling=max, Optimizer=adam, Epochs=5 Best Accuracy: 54.63% ---------- Kernel=5, Pooling=max, Optimizer=adam, Epochs=50 Best Accuracy: 55.29% ---------- ---------- Kernel=5, Pooling=max, Optimizer=adam, Epochs=100 Best Accuracy: 57.08% ---------- Kernel=5, Pooling=max, Optimizer=adam, Epochs=250 Best Accuracy: 56.48% ---------- ---------- Kernel=5, Pooling=max, Optimizer=adam, Epochs=350 Best Accuracy: 56.83% ---------- Kernel=5, Pooling=avg, Optimizer=sgd, Epochs=5 Best Accuracy: 43.43% ---------- Kernel=5, Pooling=avg, Optimizer=sgd, Epochs=50 Best Accuracy: 51.67% ---------- Kernel=5, Pooling=avg, Optimizer=sgd, Epochs=100 Best Accuracy: 53.26% ---------- Kernel=5, Pooling=avg, Optimizer=sgd, Epochs=250 Best Accuracy: 52.74% ---------- ---------- Kernel=5, Pooling=avg, Optimizer=sgd, Epochs=350 Best Accuracy: 53.15% ---------- Kernel=5, Pooling=avg, Optimizer=rmsprop, Epochs=5 Best Accuracy: 28.88% ---------- ---------- Kernel=5, Pooling=avg, Optimizer=rmsprop, Epochs=50 Best Accuracy: 30.79% ---------- Kernel=5, Pooling=avg, Optimizer=rmsprop, Epochs=100 Best Accuracy: 32.80% ---------- Kernel=5, Pooling=avg, Optimizer=rmsprop, Epochs=250 Best Accuracy: 29.88% ---------- ---------- Kernel=5, Pooling=avg, Optimizer=rmsprop, Epochs=350 Best Accuracy: 32.79% ---------- Kernel=5, Pooling=avg, Optimizer=adam, Epochs=5 Best Accuracy: 50.88% ---------- Kernel=5, Pooling=avg, Optimizer=adam, Epochs=50 Best Accuracy: 54.69% ---------- ---------- Kernel=5, Pooling=avg, Optimizer=adam, Epochs=100 Best Accuracy: 54.50% ---------- Kernel=5, Pooling=avg, Optimizer=adam, Epochs=250 Best Accuracy: 54.14% ---------- ---------- Kernel=5, Pooling=avg, Optimizer=adam, Epochs=350 Best Accuracy: 53.83% ---------- Kernel=7, Pooling=max, Optimizer=sgd, Epochs=5 Best Accuracy: 47.48% ---------- Kernel=7, Pooling=max, Optimizer=sgd, Epochs=50 Best Accuracy: 55.63% ---------- ---------- Kernel=7, Pooling=max, Optimizer=sgd, Epochs=100 Best Accuracy: 55.38% ---------- Kernel=7,

Pooling=max, Optimizer=sgd, Epochs=250 Best Accuracy: 56.16% ---------- ---------- Kernel=7, Pooling=max, Optimizer=sgd, Epochs=350 Best Accuracy: 54.61% ---------- Kernel=7, Pooling=max, Optimizer=rmsprop, Epochs=5 Best Accuracy: 15.36% ---------- Kernel=7, Pooling=max, Optimizer=rmsprop, Epochs=50 Best Accuracy: 22.47% ---------- ---------- Kernel=7, Pooling=max, Optimizer=rmsprop, Epochs=100 Best Accuracy: 12.80% ---------- Kernel=7, Pooling=max, Optimizer=rmsprop, Epochs=250 Best Accuracy: 10.56% ---------- Kernel=7, Pooling=max, Optimizer=rmsprop, Epochs=350 Best Accuracy: 17.71% ---------- ---------- Kernel=7, Pooling=max, Optimizer=adam, Epochs=5 Best Accuracy: 49.07% ---------- Kernel=7, Pooling=max, Optimizer=adam, Epochs=50 Best Accuracy: 53.86% ---------- ---------- Kernel=7, Pooling=max, Optimizer=adam, Epochs=100 Best Accuracy: 53.33% ---------- Kernel=7, Pooling=max, Optimizer=adam, Epochs=250 Best Accuracy: 54.58% ---------- Kernel=7, Pooling=max, Optimizer=adam, Epochs=350 Best Accuracy: 53.32% ---------- ---------- Kernel=7, Pooling=avg, Optimizer=sgd, Epochs=5 Best Accuracy: 42.57% ---------- Kernel=7, Pooling=avg, Optimizer=sgd, Epochs=50 Best Accuracy: 51.45% ---------- ---------- Kernel=7, Pooling=avg, Optimizer=sgd, Epochs=100 Best Accuracy: 53.21% ---------- Kernel=7, Pooling=avg, Optimizer=sgd, Epochs=250 Best Accuracy: 52.55% - ---------- Kernel=7, Pooling=avg, Optimizer=sgd, Epochs=350 Best Accuracy: 51.46% ---------- ---------- Kernel=7, Pooling=avg, Optimizer=rmsprop, Epochs=5 Best Accuracy: 11.14% ---------- Kernel=7, Pooling=avg, Optimizer=rmsprop, Epochs=50 Best Accuracy: 19.93% --------- ---------- Kernel=7, Pooling=avg, Optimizer=rmsprop, Epochs=100 Best Accuracy: 22.97% ---------- ---------- Kernel=7, Pooling=avg, Optimizer=rmsprop, Epochs=250 Best Accuracy: 24.43% ---------- Kernel=7, Pooling=avg, Optimizer=rmsprop, Epochs=350 Best Accuracy: 31.84% ---------- Kernel=7, Pooling=avg, Optimizer=adam, Epochs=5 Best Accuracy: 46.71% ---------- ---------- Kernel=7, Pooling=avg, Optimizer=adam, Epochs=50 Best Accuracy: 53.10% ---------- Kernel=7, Pooling=avg, Optimizer=adam, Epochs=100 Best Accuracy: 51.23% ---------- ---------- Kernel=7, Pooling=avg, Optimizer=adam, Epochs=250 Best Accuracy: 52.07% ---------- Kernel=7, Pooling=avg, Optimizer=adam, Epochs=350 Best Accuracy: 50.96% ----------

Output yang ditampilkan dalam kode di atas menunjukkan hasil eksperimen yang dilakukan dengan berbagai kombinasi parameter untuk melatih model.

Kombinasi parameter yang diuji meliputi ukuran kernel (3x3, 5x5, 7x7), metode pooling (max pooling dan average pooling), jenis optimizer (SGD, RMSprop, dan Adam), serta jumlah epoch (5, 50, 100, 250, dan 350). Setiap eksperimen menghasilkan akurasi terbaik (Best Accuracy) yang dicatat dan ditampilkan untuk setiap kombinasi parameter tersebut.

Hasil eksperimen menunjukkan bahwa optimizers seperti Adam cenderung menghasilkan akurasi terbaik pada sebagian besar kombinasi, sementara optimizer RMSprop menunjukkan kinerja yang lebih rendah di hampir semua kombinasi. Max pooling lebih sering memberikan hasil yang lebih baik dibandingkan average pooling. Selain itu, meskipun jumlah epochs dapat mempengaruhi akurasi, akurasi terbaik sering dicapai pada jumlah epochs sekitar 250, setelah itu akurasi cenderung stagnan atau bahkan menurun karena potensi overfitting. Dari hasil ini, dapat disimpulkan bahwa kombinasi parameter yang tepat dapat meningkatkan kinerja model, dengan Adam sebagai optimizer yang paling optimal dan max pooling sebagai metode pooling yang lebih efektif.



1. Grafik Training Loss

Grafik di sebelah kiri menunjukkan perubahan nilai kerugian (loss) selama proses pelatihan. Nilai loss secara konsisten menurun dari awal hingga mendekati nol seiring bertambahnya jumlah epoch. Penurunan loss ini mencerminkan bahwa model semakin mampu mempelajari pola dari data pelatihan dan meminimalkan kesalahan prediksi. Di akhir pelatihan, loss stabil pada nilai yang sangat rendah, menunjukkan konvergensi model terhadap data pelatihan.

2. Grafik Test Accuracy

Grafik di sebelah kanan menggambarkan perubahan akurasi model pada data uji selama pelatihan. Akurasi meningkat secara signifikan pada beberapa epoch pertama dan mencapai titik stabil setelahnya dengan sedikit fluktuasi. Tren ini menunjukkan bahwa model berhasil meningkatkan performanya pada data uji, namun fluktuasi kecil di akhir pelatihan dapat mengindikasikan adanya variasi dalam generalisasi model terhadap data baru. Akurasi stabil di sekitar angka tertentu, mencerminkan hasil akhir yang dapat diterima untuk model ini.

CNN Fashion mnist

```
# Pengaturan Hyperparameter dan Proses Training
kernel_sizes = [3, 5, 7]  # Pilihan ukuran kernel
epochs_options = [5, 50, 100  ]  # Jumlah epoch yang diuji
optimizers = [optim.SGD, optim.RMSprop, optim.Adam]  # Optimizer yang digunakan
early_stop_patience = 10  # Patokan early stopping
learning_rate = 0.001  # Tingkat pembelajaran (learning rate)

results = []

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

for kernel_size in kernel_sizes:
    for epoch_option in epochs_options:
        for optimizer_class in optimizers:
            print(f"Training dengan kernel_size={kernel_size}, epochs={epoch_option}, optimizer={optimizer_class.__name__}")

            model = CNN(kernel_size=kernel_size).to(device)
            criterion = nn.CrossEntropyLoss()
            optimizer = optimizer_class(model.parameters(), lr=learning_rate)
            scheduler = StepLR(optimizer, step_size=10, gamma=0.5)

            train_loss, val_loss = train_model(model, optimizer, scheduler, criterion, epoch_option, early_stop_patience)

            results.append({
                "kernel_size": kernel_size,
                "epochs": epoch_option,
                "optimizer": optimizer_class.__name__,
                "train_loss": train_loss,
                "val_loss": val_loss,
                "final_train_loss": train_loss[-1],
                "final_val_loss": val_loss[-1]
            })
```

Kode ini mendefinisikan proses eksperimen untuk melatih model CNN (Convolutional Neural Network) dengan berbagai kombinasi hyperparameter seperti ukuran kernel, jumlah epoch, dan jenis optimizer. Hyperparameter yang diatur meliputi:

1. **Kernel Sizes**: Ukuran kernel yang diuji, yaitu [3, 5, 7].
2. **Epoch Options**: Jumlah epoch yang diuji, yaitu [5, 50, 100].
3. **Optimizers**: Optimizer yang digunakan untuk pembaruan bobot, termasuk SGD, RMSprop, dan Adam.
4. **Learning Rate**: Nilai tingkat pembelajaran (0.001).
5. **Early Stop Patience**: Parameter untuk menghentikan pelatihan lebih awal jika performa validasi tidak membaik setelah sejumlah epoch (10 epoch).

Proses eksperimen dilakukan sebagai berikut:

- **Device Initialization**: Model akan dilatih menggunakan GPU (cuda) jika tersedia, atau menggunakan CPU jika tidak ada GPU.
- **Nested Loops**: Tiga tingkat loop bersarang digunakan untuk mencoba semua kombinasi hyperparameter, di mana ukuran kernel, jumlah epoch, dan optimizer dipilih secara iteratif.
- **Model Initialization**: Model CNN diinisialisasi untuk setiap kombinasi hyperparameter dengan menggunakan ukuran kernel yang dipilih.

- **Criterion and Optimizer**: Fungsi loss yang digunakan adalah CrossEntropyLoss(), dan optimizer diatur sesuai dengan kelas optimizer yang dipilih. Learning rate tetap pada 0.001.
- **Scheduler**: Penjadwal pembelajaran (StepLR) digunakan untuk mengatur pengurangan learning rate secara bertahap (step_size=10, gamma=0.5).
- **Training Process**: Fungsi train_model digunakan untuk melatih model dengan data training dan validasi, dengan implementasi early stopping berdasarkan validasi loss.
- **Result Storage**: Hasil dari setiap kombinasi hyperparameter, termasuk kernel size, epoch, optimizer, train loss, validation loss, final train loss, dan final validation loss, disimpan ke dalam list results dalam bentuk dictionary.

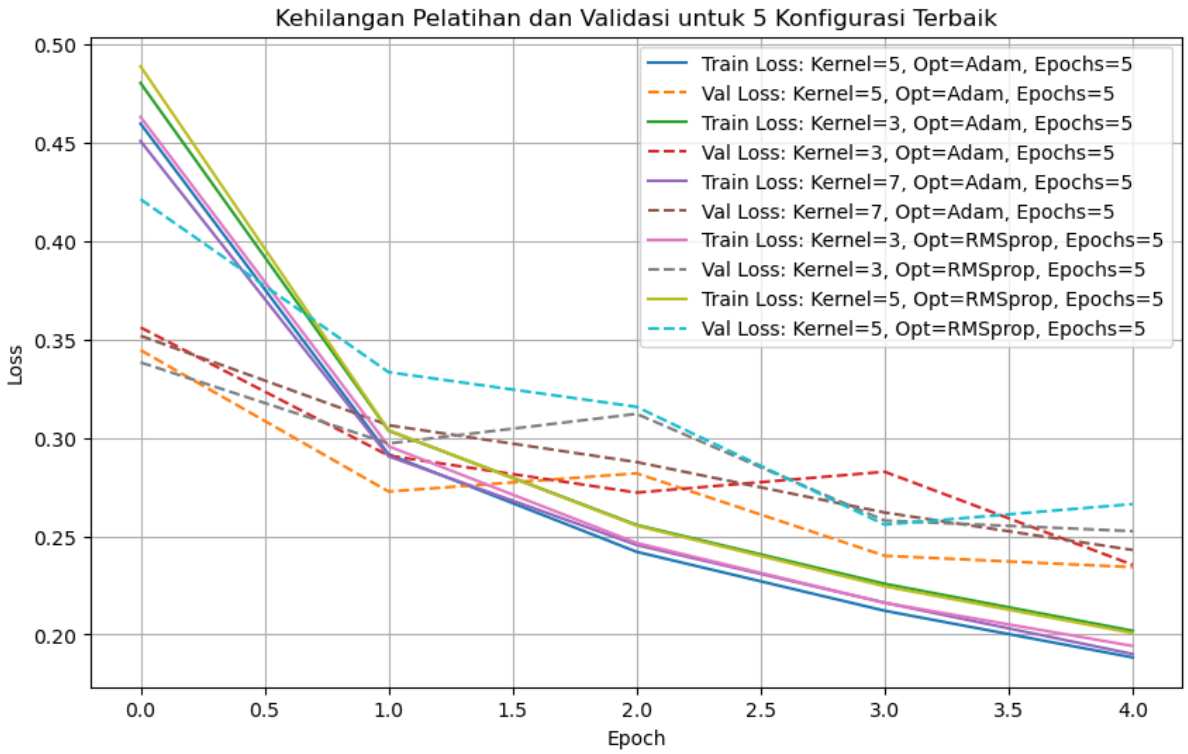
```
# Menampilkan 5 konfigurasi terbaik berdasarkan kehilangan validasi terendah
top_5_results = results_df.nsmallest(5, 'final_val_loss')

# Tampilkan 5 hasil terbaik
print("5 Hasil Hyperparameter CNN Terbaik:")
print(top_5_results)

# Visualisasi: Kehilangan Validasi untuk 5 Konfigurasi Terbaik
plt.figure(figsize=(10, 6))
for index, row in top_5_results.iterrows():
    plt.plot(
        range(len(row["train_loss"])),
        row["train_loss"],
        label=f'Train Loss: Kernel={row["kernel_size"]}, Opt={row["optimizer"]}, Epochs={row["epochs"]}',
    )
    plt.plot(
        range(len(row["val_loss"])),
        row["val_loss"],
        label=f'Val Loss: Kernel={row["kernel_size"]}, Opt={row["optimizer"]}, Epochs={row["epochs"]}',
        linestyle="--",
    )

plt.title("Kehilangan Pelatihan dan Validasi untuk 5 Konfigurasi Terbaik")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

```
5 Hasil Hyperparameter CNN Terbaik:
    kernel_size  epochs optimizer  \
11            5       5      Adam
2             3       5      Adam
20            7       5      Adam
1             3       5   RMSprop
10            5       5   RMSprop

                                      train_loss  \
11  [0.45960060394267793, 0.29155570424314753, 0.2...
2   [0.4802791292447525, 0.30364405164427594, 0.25...
20  [0.45077541028894086, 0.2906034901611078, 0.24...
1   [0.46303943072809084, 0.29570808026518647, 0.2...
10  [0.4887439204272685, 0.3037713185937674, 0.255...

                                        val_loss  final_train_loss  \
11  [0.34456243797851976, 0.27270254791732046, 0.2...          0.188384
2   [0.35607781379845493, 0.29096844526612836, 0.2...          0.201943
20  [0.3518620387763734, 0.3063930394068645, 0.287...          0.190135
1   [0.33832076334269945, 0.2971191449909453, 0.31...          0.194212
10  [0.42121875181699253, 0.3333499864882724, 0.31...          0.200774

    final_val_loss
11        0.234266
2         0.235463
20        0.242999
1         0.252573
10        0.266338
```

Code dan hasil yang ditampilkan menggambarkan proses untuk menampilkan 5 konfigurasi hyperparameter terbaik pada model CNN berdasarkan kehilangan validasi (validation loss) terendah. Berikut adalah penjelasannya:

1. **Pemilihan Konfigurasi Terbaik**:
    - Bagian kode top_5_results = results_df.nsmallest(5, 'final_val_loss') digunakan untuk memilih lima konfigurasi dengan nilai kehilangan validasi terendah (final_val_loss) dari seluruh hasil konfigurasi yang telah dijalankan. Data konfigurasi ini disimpan dalam variabel top_5_results.
2. **Visualisasi Kehilangan Pelatihan dan Validasi**:
    - Kode visualisasi menampilkan grafik untuk membandingkan kehilangan pelatihan (train_loss) dan kehilangan validasi (val_loss) dari 5 konfigurasi terbaik.
    - Setiap konfigurasi memiliki kernel size, optimizer, dan jumlah epoch yang unik. Hal ini divisualisasikan dengan menampilkan garis plot terpisah untuk setiap konfigurasi dengan keterangan legenda yang rinci.
    - Grafik menunjukkan bagaimana kehilangan berkurang seiring bertambahnya epoch, dan pola yang muncul pada data pelatihan dibandingkan data validasi.
3. **Informasi 5 Konfigurasi Terbaik**:
    - Tabel menunjukkan lima konfigurasi hyperparameter terbaik, termasuk kernel size, jumlah epoch, dan jenis optimizer yang digunakan.
    - Nilai train_loss, val_loss, final_train_loss, dan final_val_loss menunjukkan performa masing-masing konfigurasi. Dari hasil ini, konfigurasi dengan kernel_size=5, epochs=5, dan optimizer Adam memberikan nilai kehilangan validasi terendah yaitu 0.23466.

Hasil ini menunjukkan bagaimana kombinasi hyperparameter tertentu dapat memengaruhi performa model CNN dalam hal meminimalkan kehilangan validasi, yang menjadi indikator utama dalam memilih model terbaik. Analisis lebih lanjut dapat dilakukan untuk memahami pola atau tren yang muncul antara hyperparameter dan performa model.



Grafik tersebut menunjukkan perbandingan kehilangan pelatihan (train loss) dan kehilangan validasi (validation loss) untuk 5 konfigurasi hyperparameter terbaik pada model CNN. Setiap konfigurasi diwakili oleh kombinasi kernel size, optimizer, dan jumlah epoch yang telah dipilih berdasarkan performa validasi terendah. Garis solid pada grafik menunjukkan train loss, sementara garis putus-putus merepresentasikan val loss untuk setiap konfigurasi.

Dari grafik ini, dapat diamati bahwa kehilangan pelatihan dan validasi menurun secara konsisten seiring bertambahnya epoch. Namun, pada beberapa konfigurasi, val loss menunjukkan kecenderungan untuk mendatar atau bahkan sedikit meningkat pada epoch terakhir, yang bisa menjadi indikasi model mulai mengalami overfitting. Selain itu, konfigurasi dengan kernel size 5 dan optimizer Adam tampak memberikan performa yang lebih stabil dibandingkan konfigurasi lainnya.

Grafik ini mempermudah analisis visual untuk memahami bagaimana berbagai hyperparameter memengaruhi proses pelatihan dan validasi, memberikan wawasan tentang kombinasi terbaik yang dapat meminimalkan loss dan menghindari overfitting. Hasil ini juga mempertegas pentingnya pemilihan hyperparameter yang optimal untuk meningkatkan generalisasi model.