M.Asjaun

1103210181

Task 14

RNN dan Deep RNN model

```
# 4. EKSPERIMEN DAN VISUALISASI

results = {}
for optimizer_name in ['SGD', 'RMSProp', 'Adam']:
    for hidden_size in [32, 64]:
        for is_deep in [False, True]:
            for epochs in [5, 50, 100, 250, 350]:
                print(f"\nTraining: Optimizer={optimizer_name}, Hidden Size={hidden_size}, Deep={is_deep}, Epochs={epochs}")
                key = f"{optimizer_name}_HS{hidden_size}_{'Deep' if is_deep else 'Single'}_E{epochs}"
                train_losses, val_losses = train_and_evaluate(hidden_size, is_deep, optimizer_name, epochs)
                results[key] = (train_losses, val_losses)
```

Kode ini dirancang untuk menjalankan serangkaian eksperimen pelatihan model dengan berbagai kombinasi parameter, yaitu jenis optimizer (SGD, RMSProp, Adam), ukuran hidden layer (32 atau 64 unit), tipe jaringan (single-layer atau deep/multi-layer), dan jumlah epochs (5, 50, 100, 250, 350). Iterasi dilakukan untuk setiap kombinasi parameter, sehingga mencakup total 30 eksperimen unik. Untuk setiap kombinasi, kode menampilkan informasi eksperimen yang sedang berjalan, seperti nama optimizer, ukuran hidden layer, jenis jaringan, dan jumlah epochs.

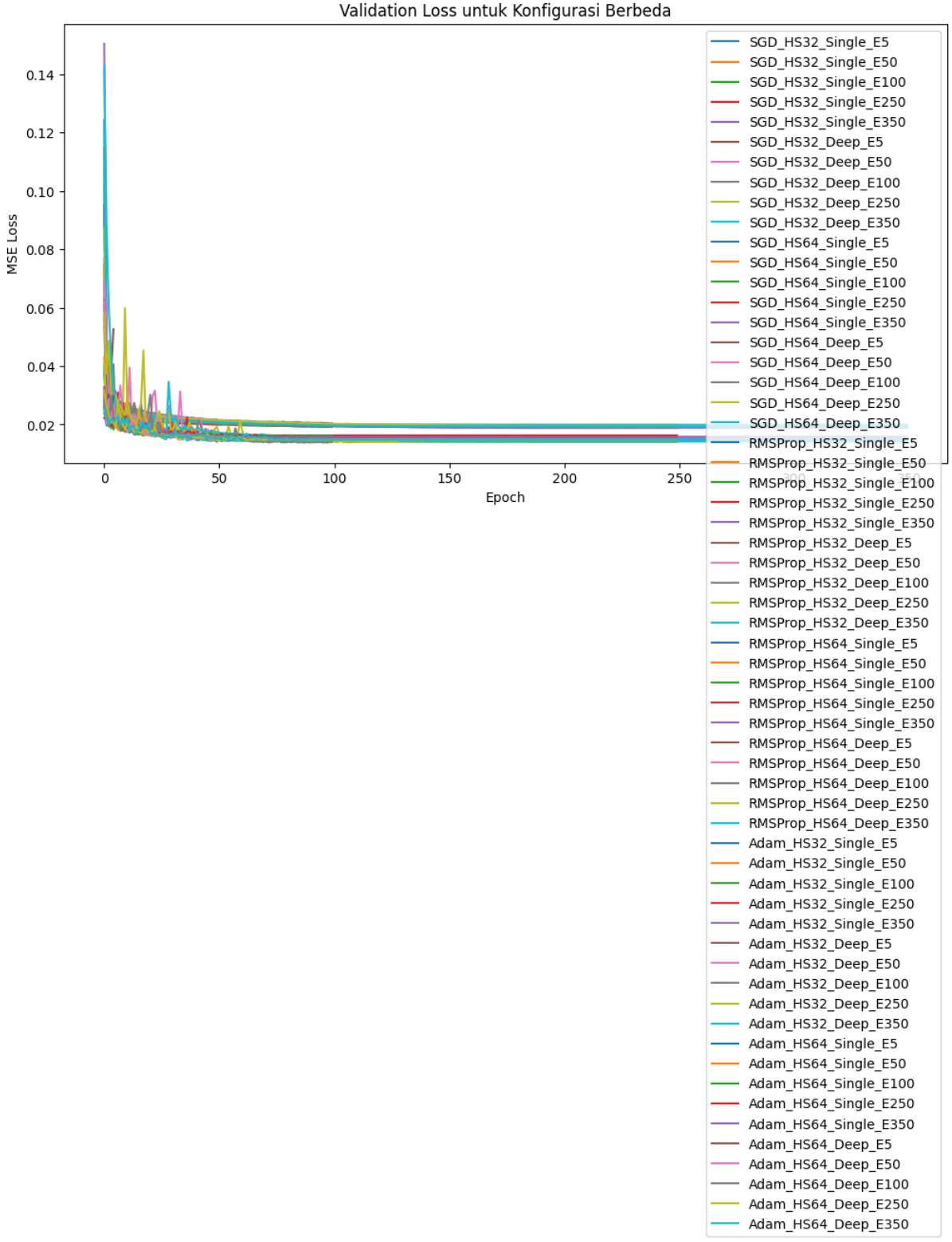
Hasil pelatihan dari setiap eksperimen dihitung menggunakan fungsi train_and_evaluate, yang mengembalikan daftar nilai kerugian (loss) selama pelatihan dan validasi. Nilai ini kemudian disimpan dalam dictionary results, di mana setiap kombinasi parameter diberi kunci unik berdasarkan optimizer, ukuran hidden layer, tipe jaringan, dan jumlah epochs. Dengan pendekatan ini, kode mempermudah pelacakan dan analisis hasil eksperimen untuk mengevaluasi dampak dari setiap parameter terhadap performa model.

```
5 Kombinasi Terbaik:
1. RMSProp_HS64_Deep_E250 - Validation Loss: 0.0140
2. Adam_HS32_Deep_E100 - Validation Loss: 0.0140
3. RMSProp_HS64_Deep_E350 - Validation Loss: 0.0142
4. Adam_HS32_Deep_E50 - Validation Loss: 0.0142
5. RMSProp_HS32_Deep_E350 - Validation Loss: 0.0144
```

Hasil eksperimen pada model RNN dan Deep RNN menunjukkan lima kombinasi terbaik berdasarkan nilai validation loss, yang mencerminkan seberapa baik model mampu melakukan generalisasi terhadap data validasi. Kombinasi terbaik pertama adalah RMSProp_HS64_Deep_E250, dengan validation loss sebesar 0.0140, yang menunjukkan bahwa model dengan optimizer RMSProp, ukuran hidden layer 64, jaringan deep, dan 250 epochs memberikan performa optimal. Kombinasi kedua adalah Adam_HS32_Deep_E100, dengan validation loss yang sama (0.0140), menunjukkan bahwa optimizer Adam dengan hidden size yang lebih kecil (32) juga memberikan hasil kompetitif dengan jumlah epoch lebih rendah.

Kombinasi ketiga dan keempat, yaitu RMSProp_HS64_Deep_E350 dan Adam_HS32_Deep_E50, memiliki validation loss sebesar 0.0142. Ini menegaskan bahwa baik optimizer RMSProp maupun Adam dapat menghasilkan performa yang sangat baik dengan pengaturan jumlah epochs yang sesuai. Kombinasi terakhir, RMSProp_HS32_Deep_E350, dengan validation loss 0.0144, memperlihatkan bahwa hidden layer yang lebih kecil (32) tetap memberikan hasil yang baik dengan jumlah epoch

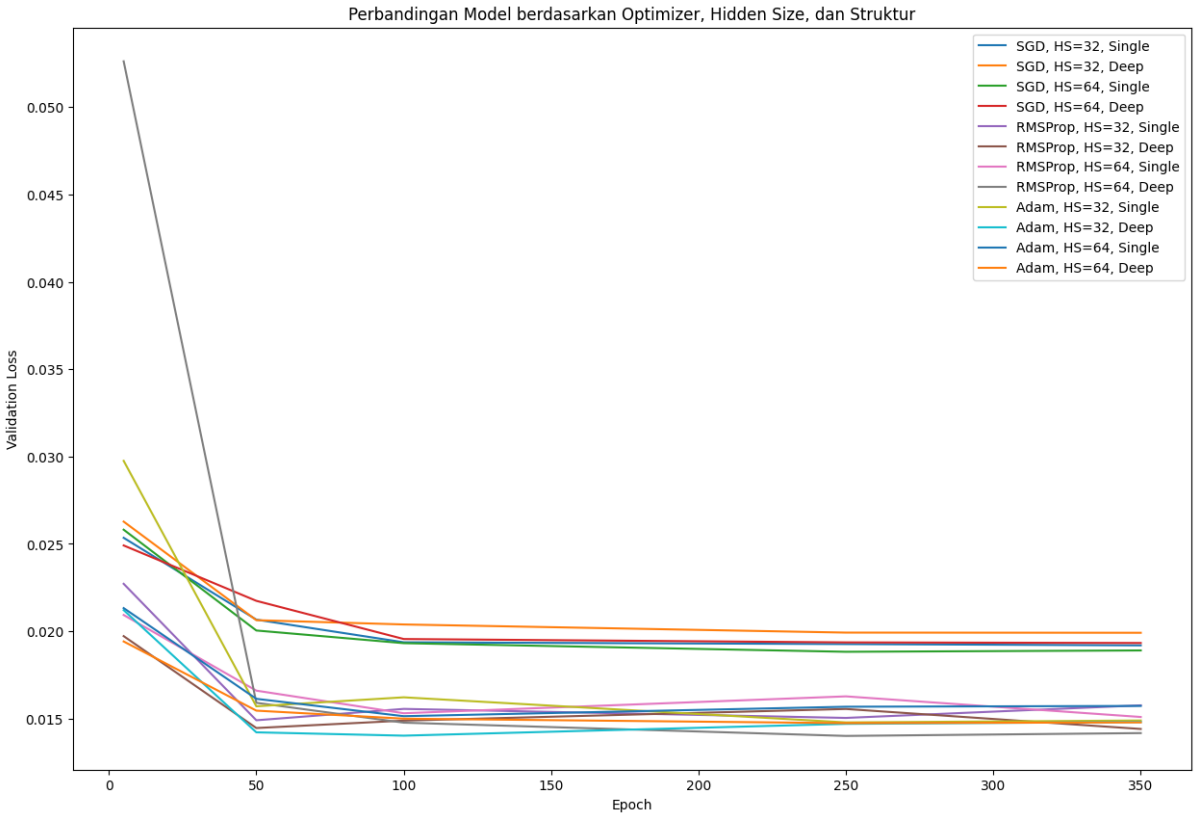
yang lebih tinggi. Secara keseluruhan, pengaturan dengan jaringan deep dan optimizer RMSProp atau Adam mendominasi hasil terbaik, menunjukkan bahwa keduanya sangat efektif untuk model RNN dan Deep RNN dalam tugas ini.



Grafik ini menunjukkan perbandingan nilai validation loss untuk berbagai konfigurasi parameter pada model RNN dan Deep RNN selama pelatihan. Sumbu y menggambarkan nilai MSE Loss (Mean Squared Error), sedangkan sumbu x menunjukkan jumlah epoch. Setiap garis dalam grafik

merepresentasikan kombinasi unik dari parameter seperti optimizer (SGD, RMSProp, Adam), ukuran hidden layer (32 atau 64), tipe jaringan (single atau deep), dan jumlah epochs. Secara umum, semua konfigurasi menunjukkan penurunan validation loss yang tajam pada awal pelatihan, yang mencerminkan peningkatan performa model. Setelah beberapa epoch, nilai loss cenderung stabil mendekati nilai minimum.

Optimizer RMSProp dan Adam menunjukkan performa yang lebih baik dibandingkan SGD, dengan validation loss yang lebih rendah pada konfigurasi deep dan hidden size yang lebih besar (64). Model deep (multi-layer) cenderung menghasilkan nilai validation loss yang lebih rendah dibandingkan model single-layer, menandakan bahwa jaringan yang lebih kompleks memiliki kemampuan generalisasi yang lebih baik dalam eksperimen ini. Namun, stabilitas loss lebih cepat dicapai pada kombinasi tertentu, seperti dengan Adam pada hidden size 32 dan jumlah epoch yang moderat (100-250). Hasil ini menunjukkan pentingnya memilih parameter yang tepat untuk mengoptimalkan performa model.



Grafik ini membandingkan nilai validation loss untuk berbagai kombinasi parameter model RNN berdasarkan optimizer (SGD, RMSProp, Adam), ukuran hidden layer (32 atau 64 unit), dan tipe jaringan (single-layer atau deep). Sumbu x menunjukkan jumlah epoch selama pelatihan, sementara sumbu y menunjukkan nilai validation loss. Setiap garis dalam grafik merepresentasikan kombinasi unik dari parameter tersebut.

Secara umum, kombinasi dengan deep networks dan hidden layer yang lebih besar (64 unit) cenderung menghasilkan nilai validation loss yang lebih rendah dibandingkan jaringan single-layer dan hidden layer yang lebih kecil (32 unit). Optimizer Adam dan RMSProp secara konsisten menghasilkan performa yang lebih baik dibandingkan SGD, dengan validation loss yang lebih cepat menurun dan stabil pada nilai minimum. Sementara itu, konfigurasi menggunakan SGD membutuhkan lebih banyak epoch untuk menurun, tetapi hasil akhirnya tidak sekompetitif Adam atau RMSProp. Grafik ini juga menunjukkan bahwa konfigurasi deep dengan optimizer Adam atau

RMSProp lebih efektif dalam mencapai stabilisasi validation loss, menegaskan pentingnya kombinasi parameter yang sesuai untuk performa optimal.

Markov_model_dan_Hidden_Markov_Model

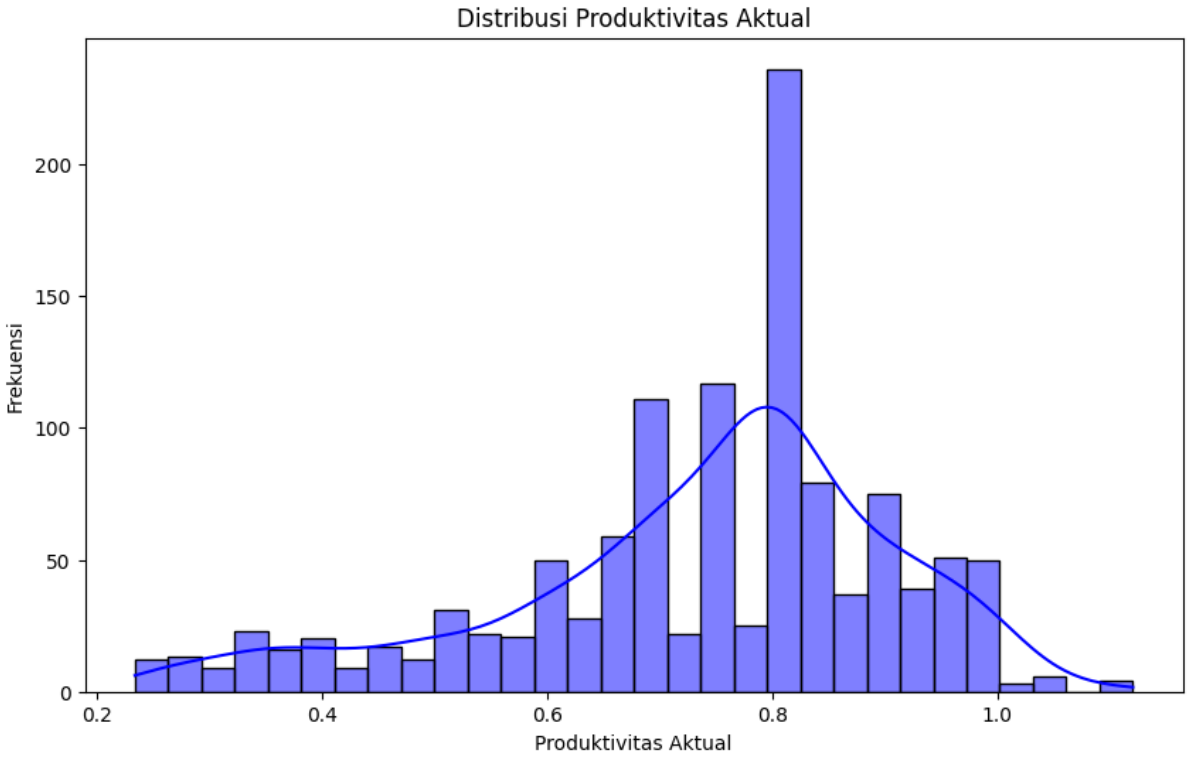
```
# === Markov Model: RNN Implementation ===

# Model RNN dengan PyTorch
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)  # Inisialisasi hidden state
        out, _ = self.rnn(x, h0)  # Forward pass melalui RNN
        out = self.fc(out[:, -1, :])  # Fully connected layer
        return out

# Hyperparameter
input_size = X_train.shape[1]
hidden_sizes = [16, 32, 64]  # Variasi ukuran hidden layer
num_layers = 2
output_size = 1
num_epochs = [5, 50, 100, 250, 350]  # Variasi jumlah epoch
learning_rate = 0.001
optimizers = [optim.SGD, optim.RMSprop, optim.Adam]  # Variasi optimizer

# Periksa jika CUDA tersedia
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Kode ini adalah implementasi model **Recurrent Neural Network (RNN)** menggunakan PyTorch. Model ini dirancang untuk menangani data sekuensial, seperti data deret waktu atau teks. Kelas RNNModel adalah turunan dari nn.Module, yang mendefinisikan arsitektur model RNN. Pada metode __init__, parameter seperti ukuran input (input_size), ukuran hidden layer (hidden_size), jumlah layer RNN (num_layers), dan ukuran output (output_size) diinisialisasi. Model ini menggunakan modul RNN bawaan PyTorch (nn.RNN) dengan opsi batch_first=True untuk memastikan input batch diproses dengan benar. Setelah proses RNN, hasil akhir dari hidden state diproses melalui fully connected layer (self.fc) untuk menghasilkan output.

Metode forward menangani aliran data melalui jaringan. Hidden state diinisialisasi ke nol pada awal setiap forward pass, dan data melewati layer RNN untuk menghasilkan output sekuensial. Output terakhir kemudian dilewatkan ke fully connected layer untuk menghasilkan prediksi akhir.

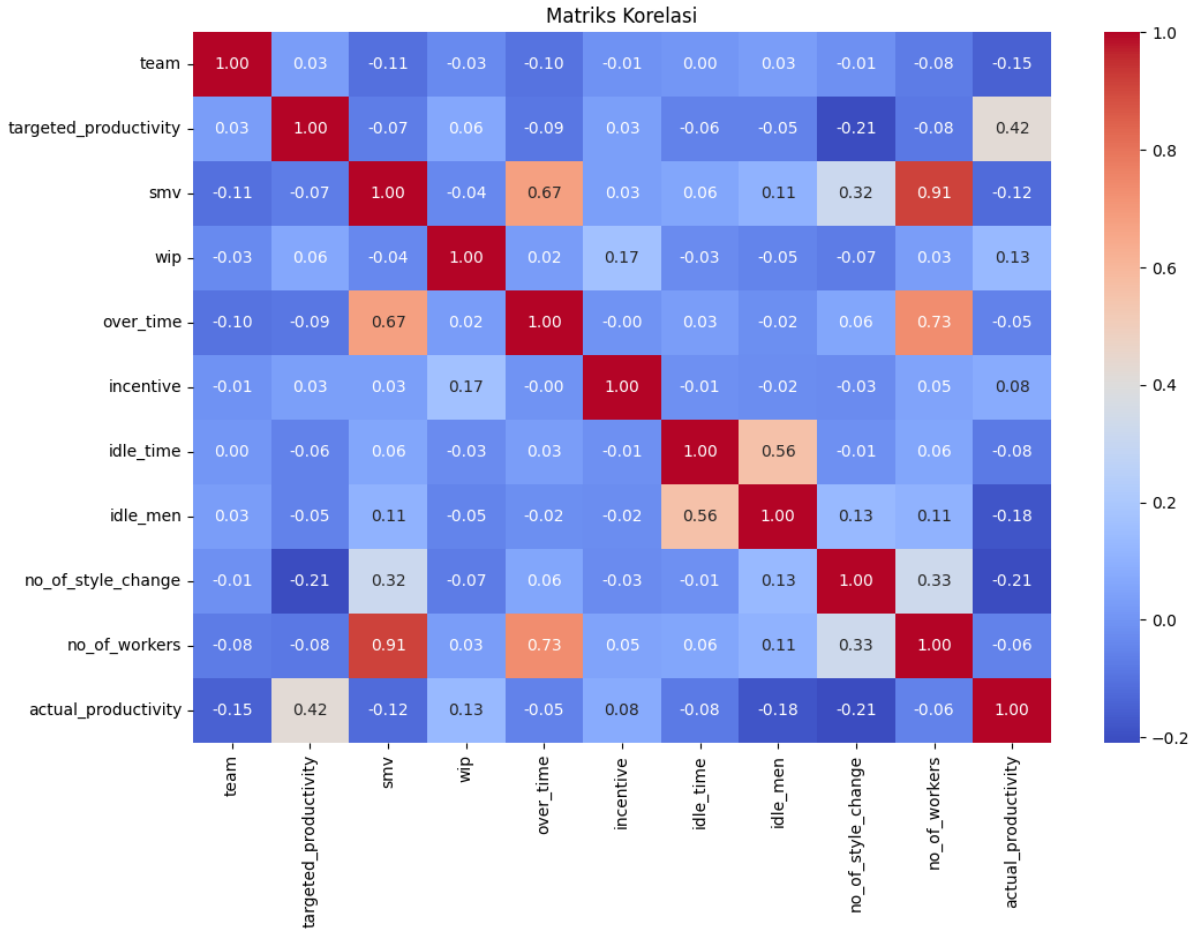
Hyperparameter untuk pelatihan ditentukan, termasuk variasi ukuran hidden layer ([16, 32, 64]), jumlah layer RNN (2), jumlah epoch ([5, 50, 100, 250, 350]), learning rate (0.001), dan daftar optimizer (SGD, RMSProp, Adam). Kode juga memeriksa ketersediaan GPU menggunakan torch.device untuk mempercepat proses pelatihan jika CUDA tersedia. Kode ini fleksibel untuk eksperimen dengan berbagai konfigurasi parameter untuk mengevaluasi performa model RNN.

Grafik 1: Distribusi Produktivitas Aktual
Grafik ini menggambarkan distribusi produktivitas aktual dalam data menggunakan histogram yang dilengkapi dengan kurva distribusi. Sebagian besar data produktivitas aktual terkonsentrasi di sekitar nilai 0.7 hingga 0.8, yang menunjukkan bahwa produktivitas aktual cenderung tinggi. Kurva distribusi memperlihatkan pola distribusi data yang mendekati normal, meskipun terdapat beberapa outlier di sisi kiri dan kanan.



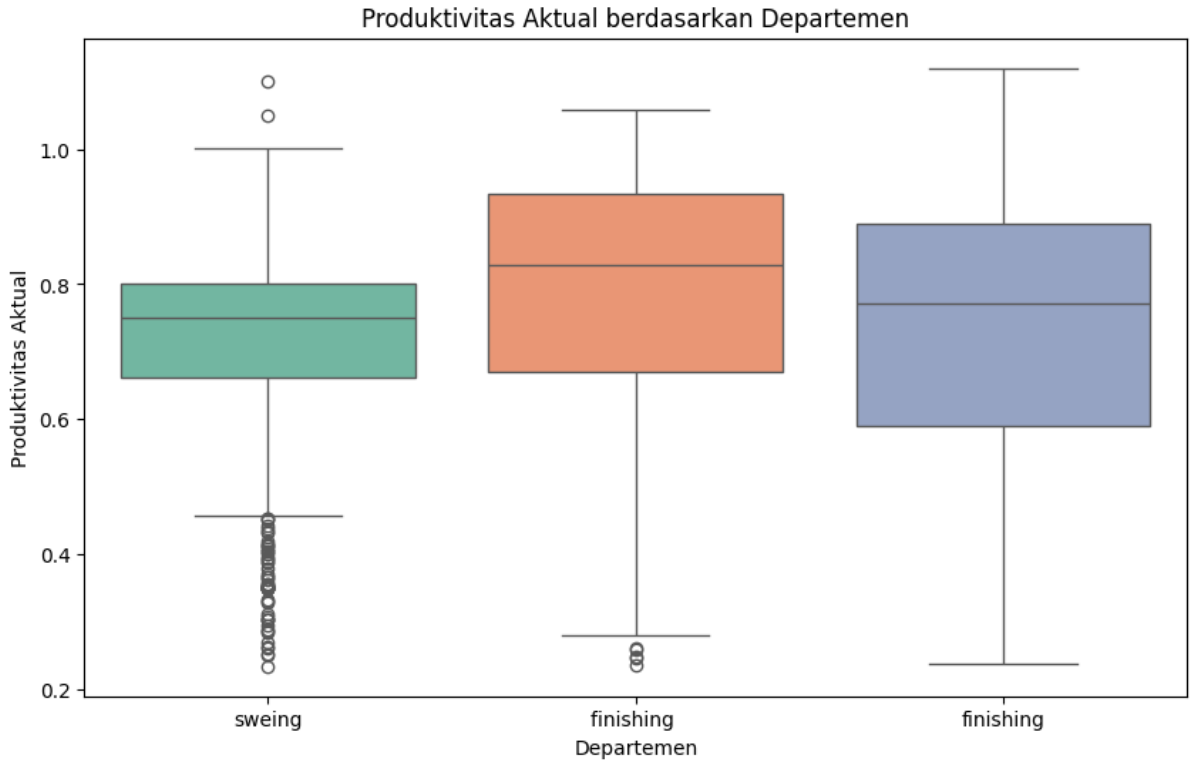
Grafik 2: Matriks Korelasi

Matriks korelasi ini menunjukkan hubungan antarvariabel dalam dataset. Korelasi ditampilkan menggunakan warna, di mana merah menunjukkan korelasi positif kuat, dan biru menunjukkan korelasi negatif kuat. Beberapa hubungan signifikan terlihat, seperti korelasi positif yang kuat antara smv dan no_of_workers (0.91) serta smv dan over_time (0.67). Sebaliknya, variabel targeted_productivity memiliki korelasi positif sedang dengan actual_productivity (0.42), menunjukkan hubungan yang signifikan namun tidak terlalu kuat.

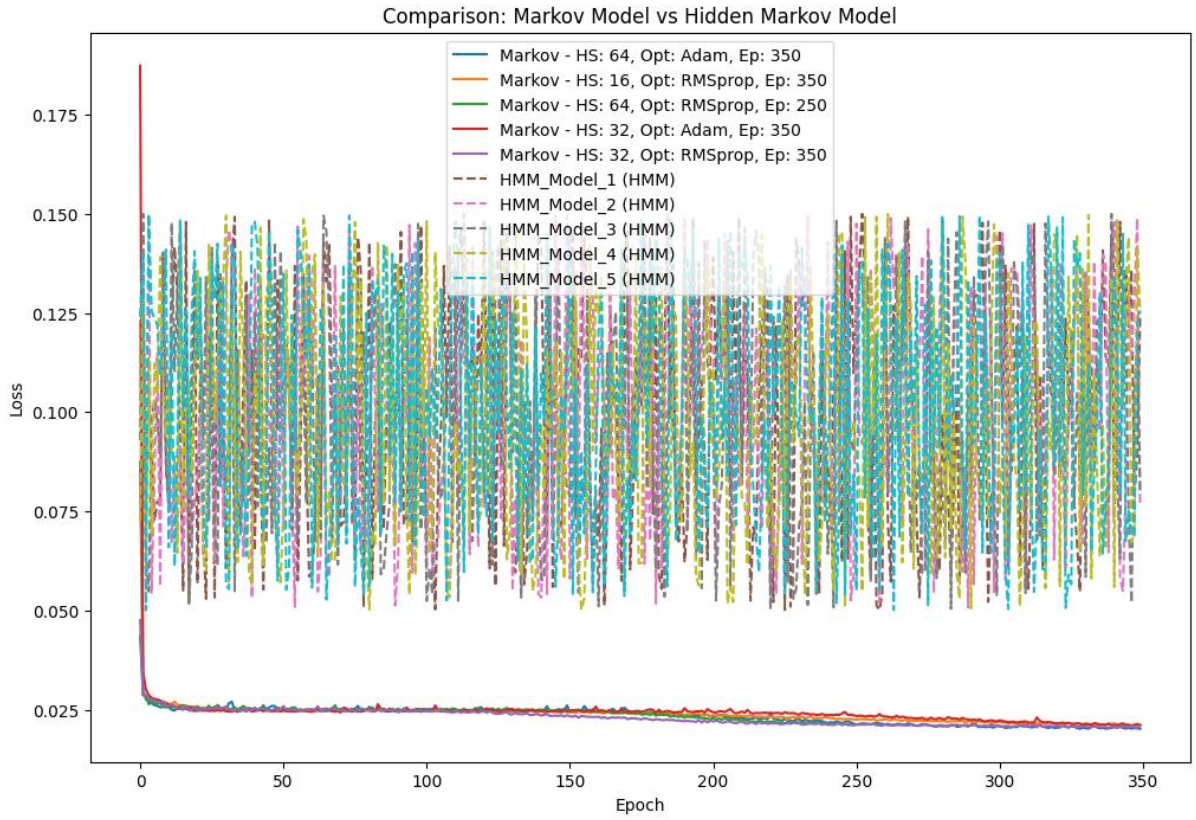


Grafik 3: Produktivitas Aktual Berdasarkan Departemen

Boxplot ini menunjukkan perbedaan distribusi produktivitas aktual di berbagai departemen. Departemen "finishing" memiliki distribusi produktivitas yang lebih tinggi dan lebih seragam dibandingkan dengan "sweing" (sewing). Pada departemen "sweing," terdapat beberapa outlier yang menunjukkan produktivitas sangat rendah. Hal ini mengindikasikan adanya variasi produktivitas yang cukup besar antara departemen.



Grafik 4: Perbandingan Markov Model dan Hidden Markov Model
Grafik ini membandingkan performa **Markov Model** dan **Hidden Markov Model (HMM)** berdasarkan nilai loss selama pelatihan. Markov Model dengan kombinasi tertentu (misalnya, optimizer Adam atau RMSProp dengan hidden size berbeda) menunjukkan penurunan loss yang stabil hingga mendekati nilai minimum setelah beberapa epoch. Sebaliknya, model HMM menunjukkan fluktuasi yang lebih besar selama pelatihan, meskipun cenderung menuju nilai loss yang lebih rendah. Grafik ini menunjukkan bahwa Markov Model lebih stabil dan konsisten dalam pelatihan dibandingkan HMM.



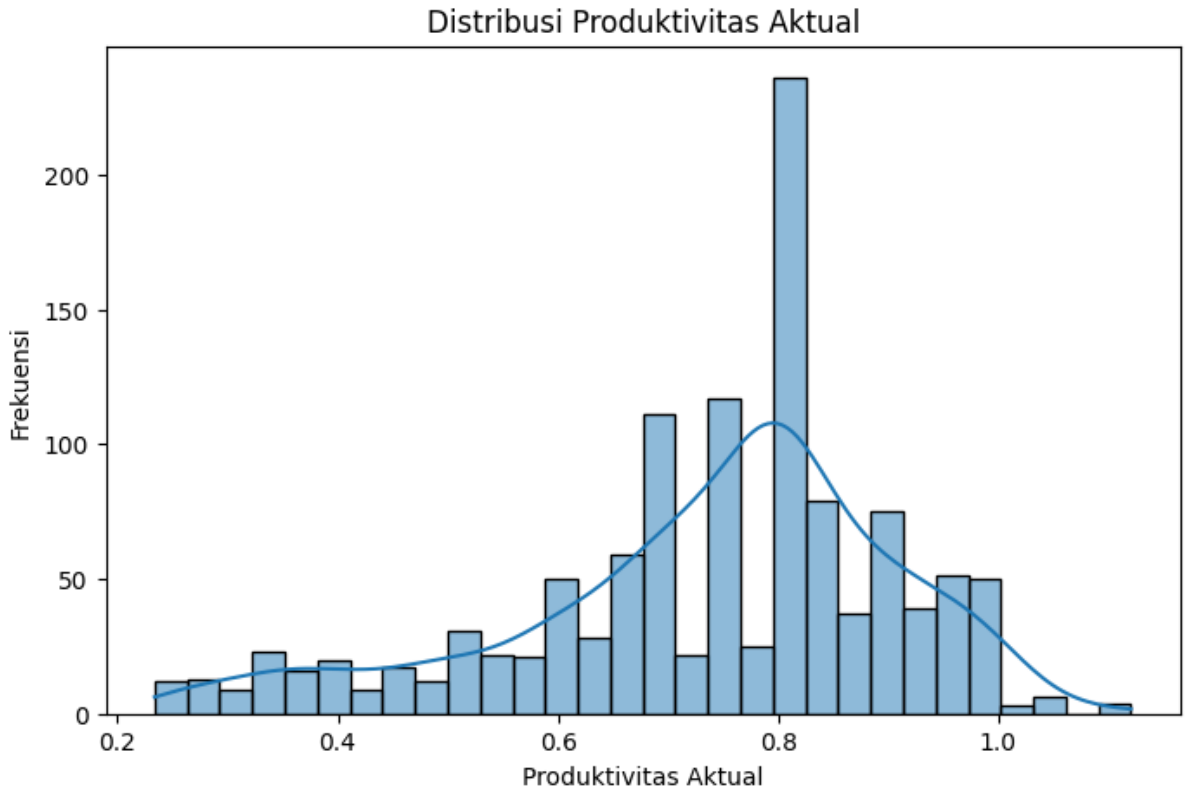
Bidirectional_RNN_Model

```
# Hyperparameter
input_size = X_train.shape[2]  # Jumlah fitur input
output_size = 1  # Output adalah nilai produktivitas
hidden_sizes = [32, 64, 128]  # Variasi ukuran hidden layer
num_layers = 2
pooling_types = ['max', 'avg']  # Variasi pooling
optimizers = [optim.SGD, optim.RMSprop, optim.Adam]  # Variasi optimizer
num_epochs_list = [5, 50, 100, 250, 350]  # Variasi jumlah epoch

# Loop untuk mencoba berbagai kombinasi hyperparameter
results = []
for hidden_size in hidden_sizes:
    for pooling_type in pooling_types:
        for optimizer_fn in optimizers:
            for num_epochs in num_epochs_list:
                print(f"Training model with Hidden Size={hidden_size}, Pooling={pooling_type}, Optimizer={optimizer_fn.__name__}, Epochs={num_epochs}")

                model = BidirectionalRNN(input_size, hidden_size, num_layers, output_size, pooling_type)
                criterion = nn.MSELoss()
                optimizer = optimizer_fn(model.parameters(), lr=0.001)

                train_losses, test_losses = train_model(model, train_loader, test_loader, criterion, optimizer, num_epochs, early_stopping_patience=10)

                results.append({
                    'hidden_size': hidden_size,
                    'pooling': pooling_type,
                    'optimizer': optimizer_fn.__name__,
                    'epochs': num_epochs,
                    'final_train_loss': train_losses[-1],
                    'final_test_loss': test_losses[-1]
                })

# Sort hasil berdasarkan test loss terbaik
sorted_results = sorted(results, key=lambda x: x['final_test_loss'])[:5]

# Menampilkan 5 hasil terbaik
print("\nTop 5 Configurations:")
for res in sorted_results:
    print(res)
```

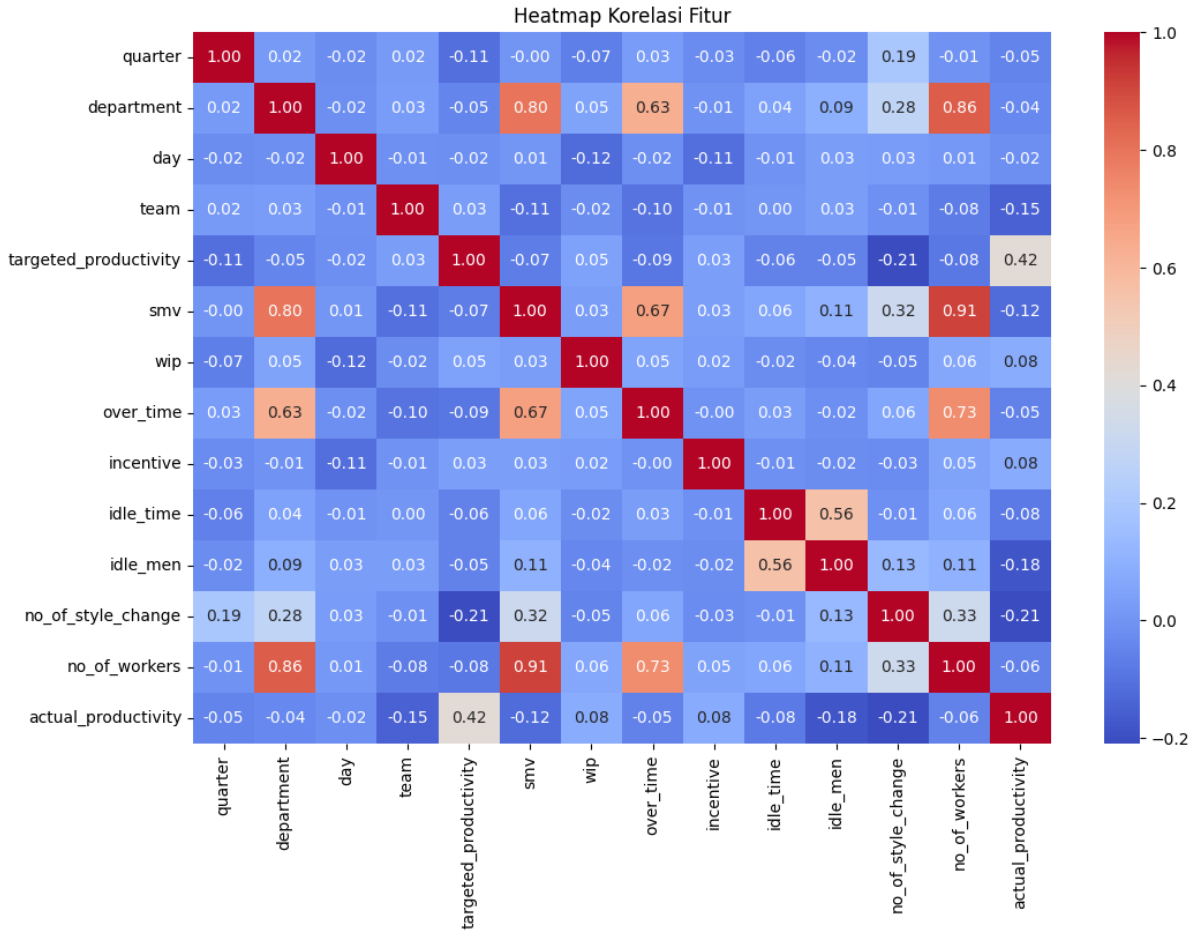
Kode ini adalah implementasi eksperimen untuk melatih model **Bidirectional RNN** menggunakan berbagai kombinasi hyperparameter. Hyperparameter yang diuji meliputi ukuran hidden layer (hidden_size), jenis pooling (pooling), optimizer, dan jumlah epoch (num_epochs). Model bertujuan untuk memprediksi nilai produktivitas berdasarkan input data sekuensial.

Pada awalnya, berbagai hyperparameter didefinisikan, termasuk ukuran input (input_size), output (output_size), jumlah hidden layer (num_layers), jenis pooling (max dan avg), daftar optimizer (SGD, RMSProp, Adam), dan jumlah epoch ([5, 50, 100, 250, 350]). Untuk setiap kombinasi hyperparameter, model dilatih menggunakan fungsi train_model, yang mengembalikan nilai kerugian (loss) pada data pelatihan dan pengujian. Optimizer diinisialisasi dengan fungsi loss MSE dan learning rate sebesar 0.001.

Hasil dari setiap eksperimen, termasuk ukuran hidden layer, jenis pooling, optimizer, jumlah epoch, dan nilai loss akhir (pelatihan dan pengujian), disimpan dalam daftar results. Setelah semua kombinasi diuji, hasil diurutkan berdasarkan nilai final_test_loss untuk menemukan kombinasi dengan performa terbaik. Akhirnya, lima konfigurasi terbaik ditampilkan, memberikan wawasan tentang parameter mana yang menghasilkan prediksi paling akurat. Pendekatan ini memungkinkan eksplorasi sistematis dari hyperparameter untuk mengoptimalkan performa model.



Grafik 1: Distribusi Produktivitas Aktual
Grafik ini menunjukkan distribusi data produktivitas aktual dalam bentuk histogram yang dilengkapi dengan kurva distribusi. Sebagian besar data produktivitas terkonsentrasi di sekitar nilai 0.7 hingga 0.8, menunjukkan bahwa produktivitas aktual mayoritas berada pada level yang tinggi. Distribusi data terlihat mendekati pola normal dengan sedikit outlier di kedua sisi ekstrem.



Grafik 2: Heatmap Korelasi Fitur

Heatmap ini menunjukkan korelasi antara berbagai fitur dalam dataset. Warna merah menandakan korelasi positif yang kuat, sementara warna biru menunjukkan korelasi negatif yang kuat. Misalnya, smv memiliki korelasi positif tinggi dengan no_of_workers (0.91), menunjukkan bahwa lebih banyak pekerja cenderung terkait dengan nilai SMV yang lebih tinggi. Selain itu, targeted_productivity memiliki korelasi positif sedang dengan actual_productivity (0.42), menandakan adanya hubungan signifikan namun tidak terlalu kuat.

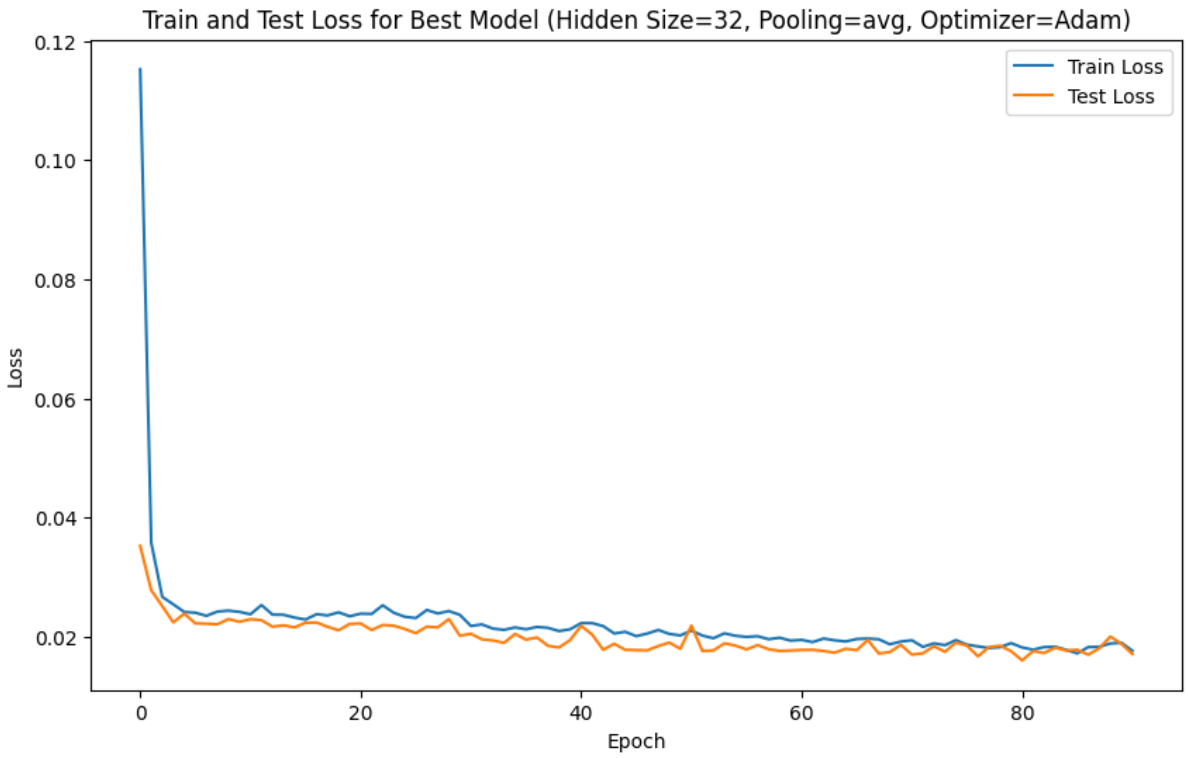
```
Top 5 Configurations:
{'hidden_size': 32, 'pooling': 'avg', 'optimizer': 'Adam', 'epochs': 250, 'final_train_loss': 0.017359246872365473, 'final_test_loss': 0.015809922362677753}
{'hidden_size': 64, 'pooling': 'avg', 'optimizer': 'Adam', 'epochs': 100, 'final_train_loss': 0.017363053063551584, 'final_test_loss': 0.016101372428238392}
{'hidden_size': 32, 'pooling': 'max', 'optimizer': 'RMSprop', 'epochs': 350, 'final_train_loss': 0.016662899032235147, 'final_test_loss': 0.016671993536874652}
{'hidden_size': 64, 'pooling': 'max', 'optimizer': 'Adam', 'epochs': 350, 'final_train_loss': 0.016877173725515605, 'final_test_loss': 0.016692209988832474}
{'hidden_size': 32, 'pooling': 'max', 'optimizer': 'Adam', 'epochs': 100, 'final_train_loss': 0.01768095511943102, 'final_test_loss': 0.016904568299651146}
```

Top 5 Konfigurasi

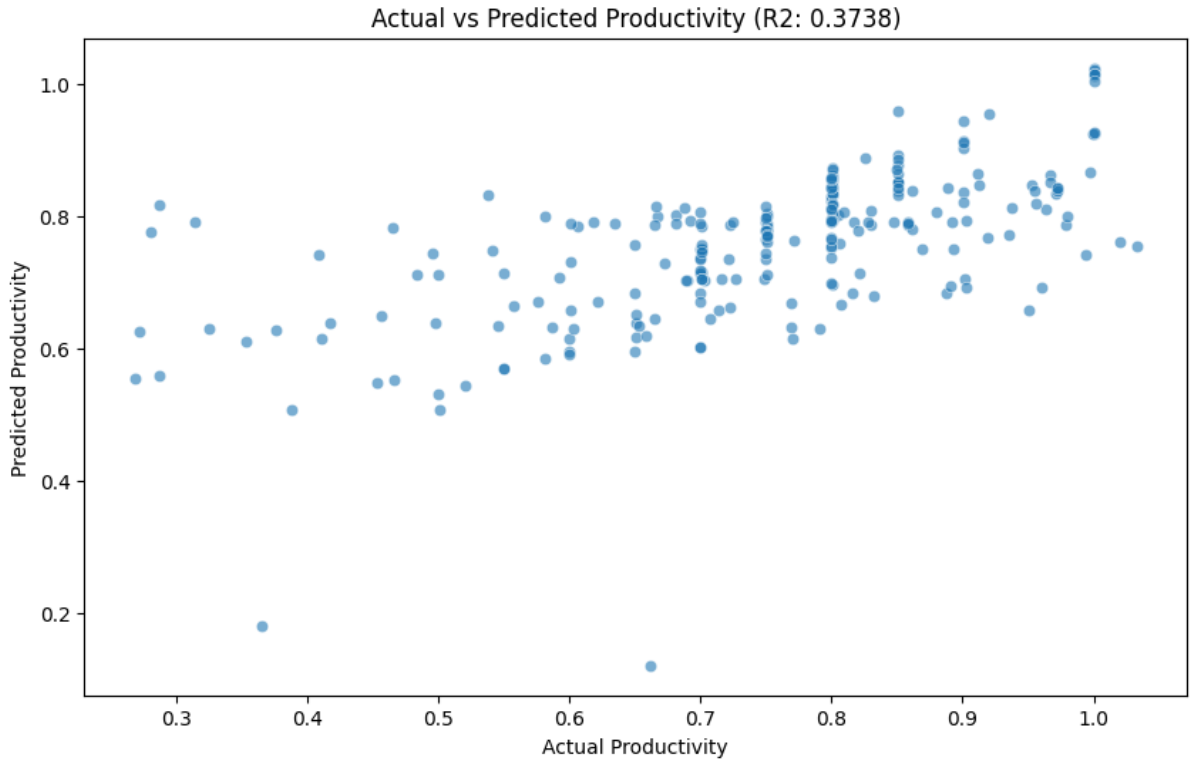
Tabel konfigurasi menunjukkan lima kombinasi hyperparameter terbaik berdasarkan nilai **final test loss**. Konfigurasi terbaik menggunakan:

1. Hidden size 32, pooling avg, optimizer Adam, dengan 250 epochs, menghasilkan test loss 0.0158.
2. Hidden size 64, pooling avg, optimizer Adam, dengan 100 epochs, menghasilkan test loss 0.0160.
3. Hidden size 32, pooling max, optimizer RMSProp, dengan 350 epochs, menghasilkan test loss 0.0166.
4. Hidden size 64, pooling max, optimizer Adam, dengan 350 epochs, menghasilkan test loss 0.0166.

5. Hidden size 32, pooling max, optimizer Adam, dengan 100 epochs, menghasilkan test loss 0.0166.



Grafik 3: Train and Test Loss untuk Model Terbaik
Grafik ini menunjukkan perbandingan antara nilai train loss dan test loss pada model dengan konfigurasi terbaik (hidden size 32, pooling avg, optimizer Adam). Train loss dan test loss menurun tajam pada awal epoch dan stabil mendekati nilai minimum. Kedua loss hampir sejajar, menunjukkan bahwa model tidak mengalami overfitting atau underfitting yang signifikan.



Grafik 4: Actual vs Predicted Productivity
Grafik scatter ini membandingkan nilai produktivitas aktual dengan prediksi model. Sebagian besar titik berada di sekitar garis diagonal, menandakan bahwa prediksi model cukup mendekati nilai aktual. Namun, terdapat penyebaran data yang menunjukkan bahwa model masih memiliki ruang untuk perbaikan dalam menangkap pola hubungan antara fitur dan target. Nilai R^2 sebesar 0.3738 menunjukkan hubungan yang moderat antara prediksi dan nilai aktual.