

Nama : Az – Zahra Chikal E

NIM : 1103213039

Kelas : TK-45-05

TUGAS WEEK 14 RNN dan DEEP RNN

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
from tqdm import tqdm
Import "tqdm" could not be resolved from source
import matplotlib.pyplot as plt
import seaborn as sns
✓ 1.3s Python

# Load and preprocess data
data = pd.read_csv('bank-full.csv', delimiter=';')
data.head()
✓ 0.0s Python
```

Output:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	outcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

Analisis:

Kode diatas digunakan untuk membuat model machine learning menggunakan PyTorch. Data yang digunakan yaitu bernama `bank-full.csv`, yang berisi informasi tentang pelanggan bank. Data ini dimuat ke dalam program menggunakan Pandas dengan pemisah `;` karena format filenya menggunakan titik koma untuk memisahkan kolom. Untuk memahami isi dan struktur data, ditampilkan beberapa baris pertama dengan `data.head()`. Selanjutnya, data ini akan diproses dan dianalisis menggunakan alat bantu seperti NumPy, Scikit-learn, serta divisualisasikan menggunakan Matplotlib dan Seaborn sebelum digunakan untuk melatih model.

```
data.info()
✓ 0.0s Python

data.describe()
✓ 0.0s Python
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
#   Column      Non-Null Count  Dtype
---  -
0    age         45211 non-null  int64
1    job         45211 non-null  object
2    marital     45211 non-null  object
3    education   45211 non-null  object
4    default     45211 non-null  object
5    balance     45211 non-null  int64
6    housing     45211 non-null  object
7    loan        45211 non-null  object
8    contact     45211 non-null  object
9    day         45211 non-null  int64
10   month       45211 non-null  object
11   duration    45211 non-null  int64
12   campaign    45211 non-null  int64
13   pdays       45211 non-null  int64
14   previous    45211 non-null  int64
15   poutcome    45211 non-null  object
16   y           45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

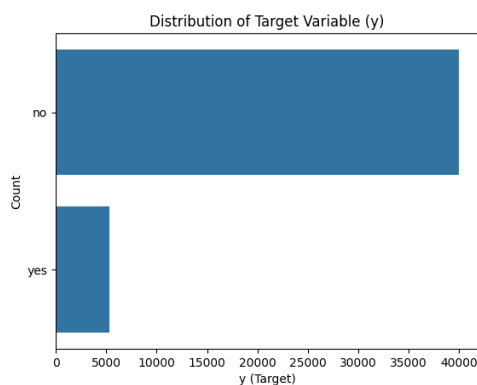
Analisis:

Kode tersebut digunakan untuk mengetahui informasi terkait dataet. Dari code tersebut dapat kita tahu bahwa terdapat 16 kolom dengan jumlah data yaitu 45211. untuk datatype terdapat 2 yaitu int dan object. Fungsi `df.describe()` digunakan untuk memberikan ringkasan statistik dasar dari data numerik dalam sebuah DataFrame. Hasilnya mencakup informasi seperti jumlah data (count), rata-rata (mean), standar deviasi (std), nilai minimum (min), kuartil (25%, 50%, 75%), dan nilai maksimum (max) untuk setiap kolom numerik. Dengan menggunakan fungsi ini, kita dapat dengan cepat memahami karakteristik umum data, seperti distribusi nilai, rentang data, dan keberadaan kemungkinan nilai ekstrem.

```
# Distribution of target variable 'y'
sns.countplot(data['y'])
plt.title('Distribution of Target Variable (y)')
plt.xlabel('y (Target)')
plt.ylabel('Count')
plt.show()
✓ 0.1s
```

Python

Output:



Analisis:

Grafik countplot menunjukkan distribusi data pada variabel target y. Sumbu horizontal menampilkan kategori dalam variabel y, sementara sumbu vertikal menunjukkan jumlah data untuk setiap kategori tersebut. Dari grafik ini, kita bisa melihat apakah data dalam variabel target seimbang atau tidak. Jika salah satu kategori memiliki jumlah data yang jauh lebih banyak dibandingkan kategori lain, itu berarti data tidak seimbang, dan hal ini perlu diperhatikan saat membuat model prediksi.

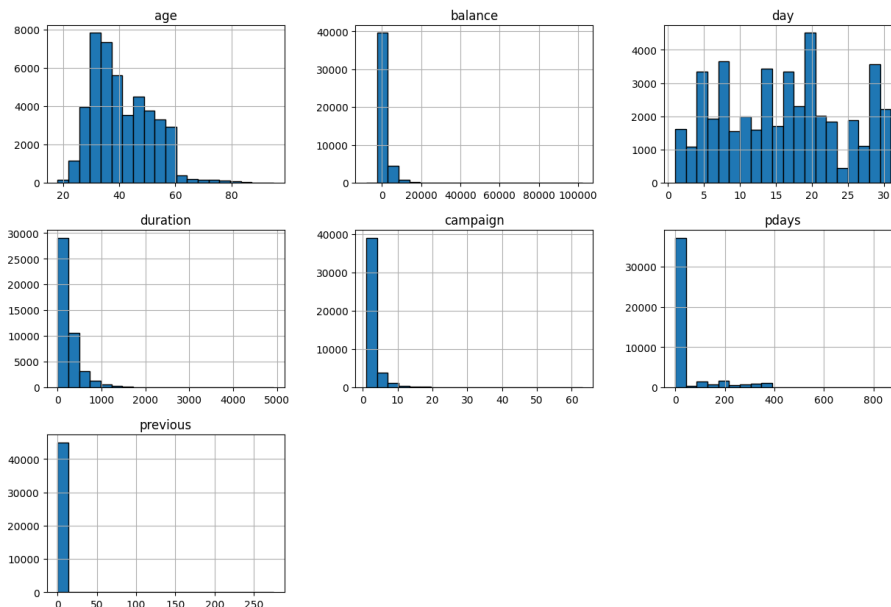
```
# Distribution of numerical variables
numerical_cols = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
data[numerical_cols].hist(figsize=(15, 10), bins=20, edgecolor='black')
plt.suptitle('Distributions of Numerical Features')
plt.show()
```

✓ 0.4s

Python

Output:

Distributions of Numerical Features



Analisis:

Histogram menunjukkan bagaimana data tersebar untuk variabel numerik seperti age, balance, day, duration, campaign, pdays, dan previous. Setiap variabel memiliki grafiknya sendiri yang menampilkan nilai-nilai data di sumbu horizontal dan jumlah kemunculannya di sumbu vertikal. Dari grafik ini, kita bisa melihat apakah distribusi data merata, condong ke satu sisi, atau memiliki nilai ekstrem (outlier). Tampilan ini membantu kita memahami pola data dan menentukan apakah ada perlakuan khusus yang perlu dilakukan, seperti normalisasi atau transformasi, sebelum digunakan dalam analisis lebih lanjut atau pembuatan model.

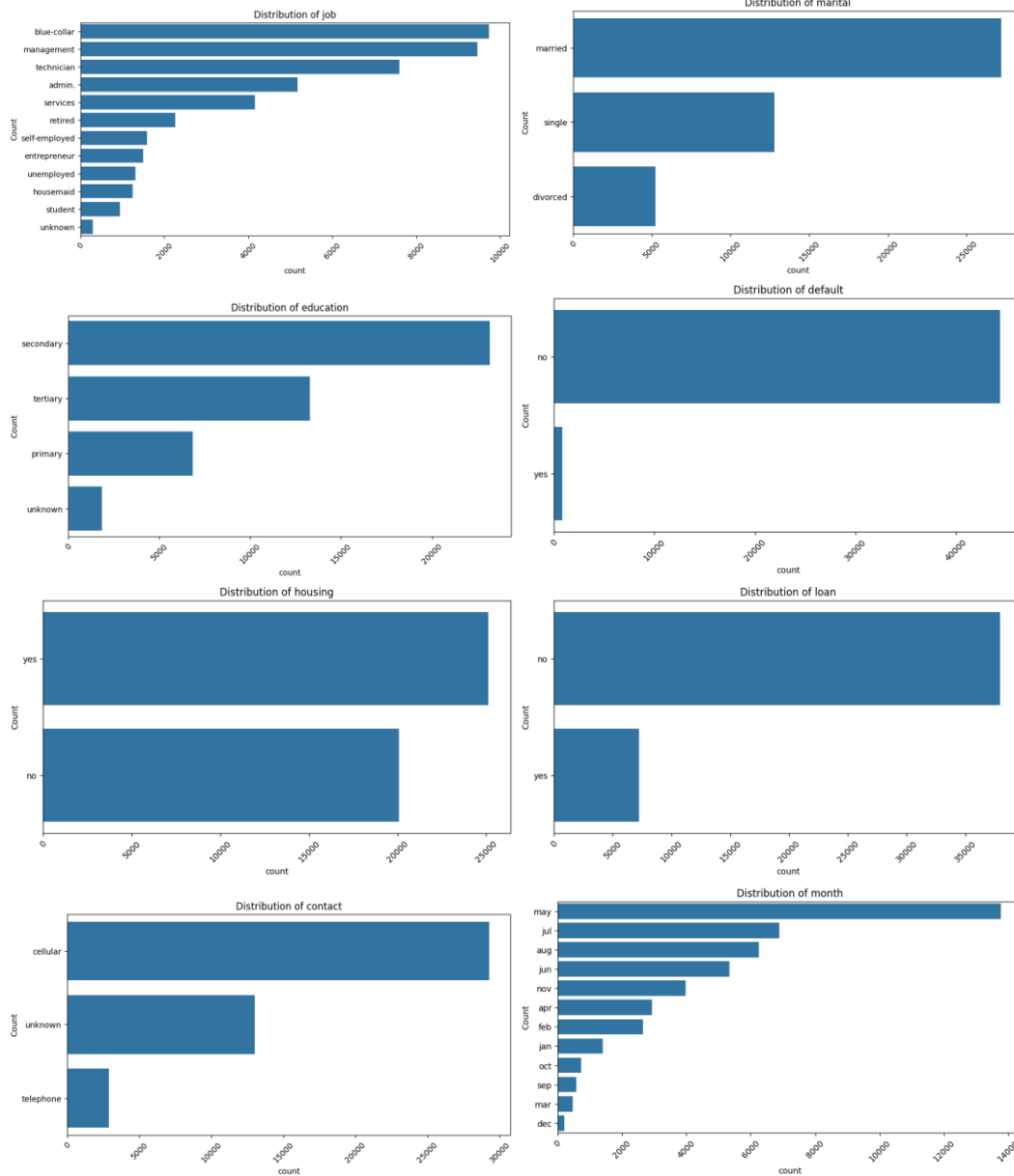
```
# Categorical variables analysis
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'outcome']

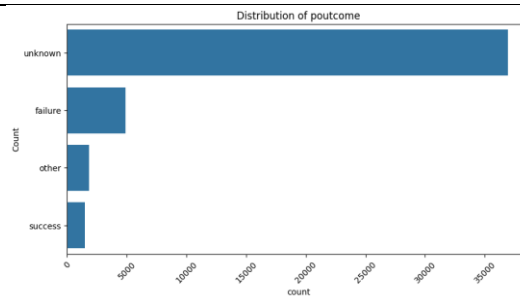
for col in categorical_cols:
    plt.figure(figsize=(10, 5))
    sns.countplot(data[col], order=data[col].value_counts().index)
    plt.title(f'Distribution of {col}')
    plt.xticks(rotation=45)
    plt.ylabel('count')
    plt.show()
```

✓ 0.9s

Python

Output:



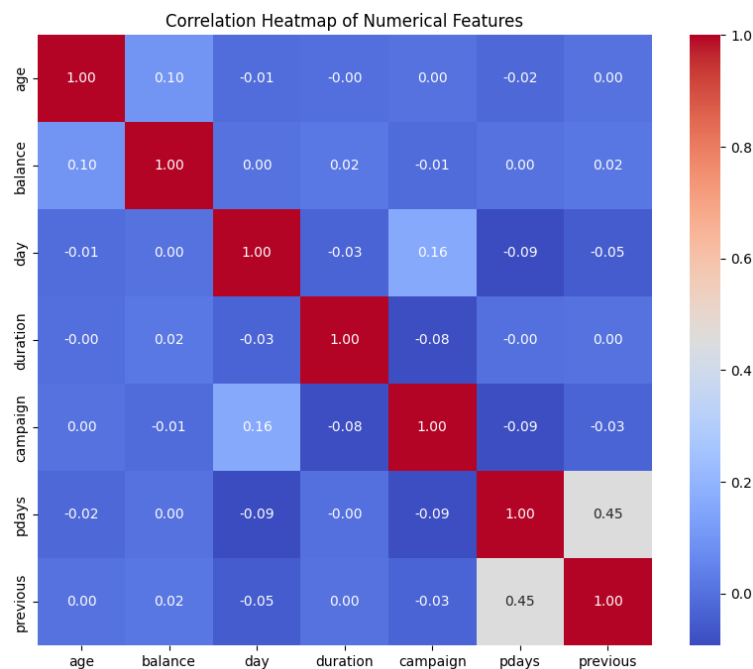


Analisis:

Grafik ini menunjukkan bagaimana data tersebar untuk variabel kategori seperti job, marital, education, default, housing, loan, contact, month, dan poutcome. Setiap variabel memiliki grafik batang (countplot) yang memperlihatkan jumlah data di setiap kategori. Sumbu horizontal menampilkan kategori, sedangkan sumbu vertikal menunjukkan jumlah data untuk masing-masing kategori. Kategori dalam grafik diurutkan dari yang paling banyak hingga paling sedikit, sehingga memudahkan kita untuk melihat kategori mana yang dominan. Label kategori dibuat miring (rotasi 45 derajat) agar tetap terbaca dengan jelas, terutama jika nama kategorinya panjang. Analisis ini penting untuk memahami pola data pada variabel kategori, seperti kategori yang sering muncul atau jarang, yang mungkin memengaruhi analisis lebih lanjut atau model yang akan dibuat.

```
# Correlation heatmap for numerical features
correlation_matrix = data[numerical_cols].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

Output:



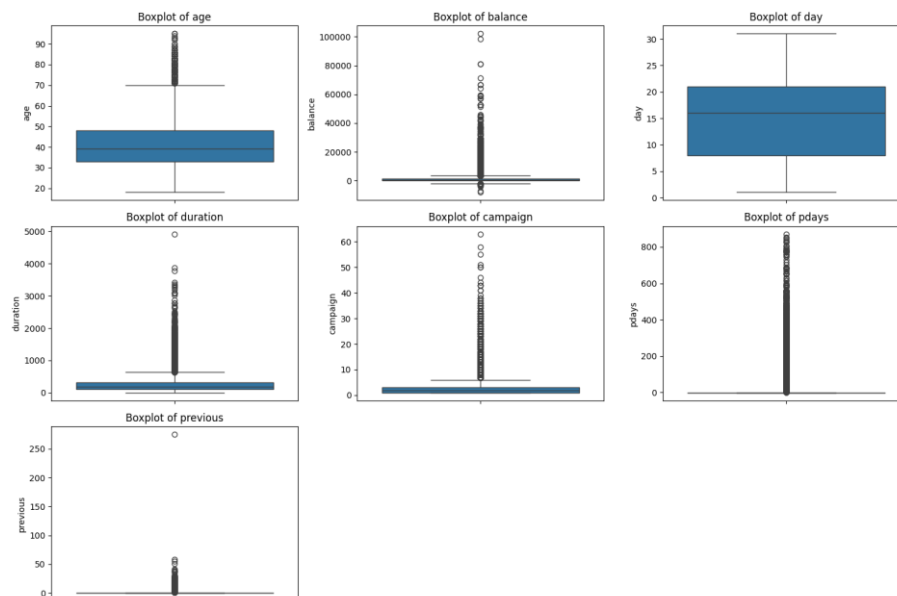
Analisis:

Heatmap ini menunjukkan hubungan antara variabel-variabel numerik dalam dataset, seperti age, balance, day, duration, campaign, pdays, dan previous. Setiap kotak dalam heatmap menggambarkan tingkat hubungan (korelasi) antara dua variabel, dengan nilai berkisar dari -1 hingga 1. Jika nilainya mendekati 1, berarti ada hubungan positif yang kuat, artinya jika satu variabel naik, variabel lain juga cenderung naik. Jika nilainya mendekati -1, berarti ada hubungan negatif yang kuat, artinya jika satu variabel naik, variabel lain cenderung turun. Nilai mendekati 0 menunjukkan tidak ada hubungan linear yang signifikan. Warna pada heatmap membantu membedakan kekuatan hubungan. warna merah menunjukkan hubungan negatif yang kuat, sementara warna biru menunjukkan hubungan positif yang kuat.

```
# Boxplot to check outliers in numerical features
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(data[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```

✓ 0.6s Python

Output:



Analisis:

Boxplot digunakan untuk melihat apakah ada outlier (data ekstrem) pada variabel numerik seperti age, balance, day, duration, campaign, pdays, dan previous. Boxplot menunjukkan ringkasan distribusi data, termasuk nilai terkecil, kuartil pertama (Q1), median, kuartil ketiga (Q3), dan nilai terbesar. Data yang dianggap outlier biasanya ditampilkan sebagai titik-titik di luar garis "whiskers".

Setiap variabel memiliki boxplot sendiri yang ditampilkan dalam grafik, sehingga kita bisa melihat pola outlier secara lebih jelas untuk masing-masing variabel. Jika banyak outlier ditemukan, langkah-langkah seperti menghapus, mengubah, atau menangani data ekstrem tersebut mungkin perlu dilakukan. Selain itu, boxplot juga membantu kita memahami pola

distribusi data, misalnya apakah data cenderung simetris atau memiliki kemiringan ke satu sisi.

```
# Preprocessing
def preprocess_data(data):
    X = data.drop('y', axis=1)
    X = pd.get_dummies(X)
    y = data['y'].apply(lambda x: 1 if x == 'yes' else 0)

    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = preprocess_data(data)

# Dataset class
class CustomDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y.values, dtype=torch.float32)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = CustomDataset(X_train, y_train)
test_dataset = CustomDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

✓ 0.1s

Python

Analisis:

Proses ini mencakup langkah-langkah untuk menyiapkan data agar bisa digunakan dalam pelatihan model. Pertama, data dipisahkan menjadi fitur (X) dan target (y). Variabel target y diubah menjadi format biner, di mana yes menjadi 1 dan no menjadi 0. Selanjutnya, fitur kategori dalam X diubah menjadi angka melalui proses one-hot encoding, sehingga bisa diproses oleh model. Semua fitur kemudian dinormalisasi menggunakan StandardScaler, yang mengatur agar setiap fitur memiliki rata-rata 0 dan standar deviasi 1, sehingga model dapat belajar lebih baik. Data ini kemudian dibagi menjadi data pelatihan (80%) dan pengujian (20%). Setelah data siap, dibuat kelas khusus bernama CustomDataset untuk mengatur data agar kompatibel dengan PyTorch. Kelas ini mengonversi data menjadi tensor PyTorch, dengan metode untuk mendapatkan ukuran dataset (__len__) dan mengakses data berdasarkan indeks (__getitem__). Dataset pelatihan dan pengujian kemudian disiapkan menggunakan kelas ini, dan data loader (DataLoader) digunakan untuk membagi dataset menjadi batch berukuran 64. Batch ini mempermudah proses pelatihan model secara bertahap, dengan data pelatihan diacak untuk meningkatkan kemampuan generalisasi model. Data pengujian tidak diacak untuk menjaga konsistensi evaluasi.

```

# Define RNN model
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, pooling='max'):
        super(RNNModel, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.pooling = pooling
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        if self.pooling == 'max':
            out = torch.max(out, dim=1).values
        elif self.pooling == 'avg':
            out = torch.mean(out, dim=1)
        out = self.fc(out)
        return out

```

✓ 0.0s Python

Analisis:

Model RNNModel adalah model jaringan saraf berulang (RNN) yang dirancang untuk memproses data berurutan, seperti teks atau data berbasis waktu. Model ini memiliki beberapa bagian utama. Pertama, data dimasukkan ke dalam lapisan RNN, yang bertugas mengenali pola dalam data sekuensial. Hasil dari lapisan RNN diringkas menggunakan metode pooling. Ada dua pilihan pooling: max pooling, yang mengambil nilai maksimum, atau average pooling, yang menghitung rata-rata hasil di sepanjang waktu. Setelah diringkas, data tersebut diteruskan ke lapisan linear untuk menghasilkan prediksi akhir. Model ini fleksibel karena memungkinkan pengguna memilih metode pooling yang sesuai dengan kebutuhan. Dengan struktur ini, model dapat mengenali pola penting dalam data sekuensial dan memberikan prediksi berdasarkan informasi yang telah diproses. Model ini cocok digunakan untuk berbagai tugas, seperti klasifikasi teks, analisis data waktu, atau prediksi dalam urutan data lainnya.

```

# Training function
def train_model(model, optimizer, criterion, scheduler, num_epochs, early_stopper, train_loader, test_loader):
    best_loss = float('inf')
    patience_counter = 0
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        for x_batch, y_batch in tqdm(train_loader):
            x_batch, y_batch = x_batch.to(device), y_batch.to(device)
            optimizer.zero_grad()
            outputs = model(x_batch.unsqueeze(1))
            loss = criterion(outputs.squeeze(), y_batch)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)

        model.eval()
        test_loss = 0
        correct = 0
        total = 0
        with torch.no_grad():
            for x_batch, y_batch in test_loader:
                x_batch, y_batch = x_batch.to(device), y_batch.to(device)
                outputs = model(x_batch.unsqueeze(1))
                loss = criterion(outputs.squeeze(), y_batch)
                test_loss += loss.item()
                predictions = torch.round(torch.sigmoid(outputs.squeeze()))
                correct += (predictions == y_batch).sum().item()
                total += y_batch.size(0)
        test_loss /= len(test_loader)
        accuracy = correct / total

```



```

scheduler.step(test_loss)

if test_loss < best_loss:
    best_loss = test_loss
    patience_counter = 0
else:
    patience_counter += 1

if patience_counter > early_stopper:
    print("Early stopping triggered")
    break

print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}, Accuracy: {
return accuracy

```

✓ 0.0s

Python

Analisis:

Fungsi `train_model` digunakan untuk melatih model dengan data pelatihan dan mengukur kinerjanya pada data pengujian. Proses dimulai dengan menempatkan model di perangkat yang tersedia, seperti GPU (jika ada) atau CPU. Pelatihan dilakukan selama beberapa epoch, di mana setiap epoch terdiri dari dua tahap: pelatihan model dengan data pelatihan dan evaluasi model dengan data pengujian. Pada tahap pelatihan, model memproses data dalam batch dari `train_loader`. Model menghasilkan output, menghitung loss menggunakan fungsi tertentu (criterion), dan memperbarui parameter model melalui backpropagation dengan bantuan optimizer. Setelah semua batch diproses, rata-rata loss dihitung sebagai ukuran kinerja model selama pelatihan. Tahap evaluasi dilakukan dengan model dalam mode evaluasi, di mana data pengujian diproses tanpa menghitung gradien. Pada tahap ini, loss dan akurasi model dihitung. Akurasi dihitung dengan membandingkan hasil prediksi model dengan label sebenarnya. Fungsi ini juga menggunakan scheduler untuk menyesuaikan laju pembelajaran berdasarkan kinerja pada data pengujian. Early stopping diterapkan untuk menghentikan pelatihan jika performa model tidak meningkat setelah sejumlah epoch tertentu, menghindari pemborosan waktu dan sumber daya. Hasil dari setiap epoch, seperti loss dan akurasi, dicetak untuk memantau kinerja model. Jika model mencapai performa terbaiknya, hasil akhir berupa akurasi pada data pengujian dikembalikan. Fungsi ini memastikan model dilatih dengan efisien dan berhenti tepat waktu saat tidak ada peningkatan signifikan.

```

# Experiment configurations
hidden_sizes = [16, 32, 64]
poolings = ['max', 'avg']
epochs_list = [5, 50, 100, 250, 350]
optimizers = ['SGD', 'RMSprop', 'Adam']
early_stopper = 10

# Run experiments for RNN and save results
results_rnn = []
for hidden_size in hidden_sizes:
    for pooling in poolings:
        for optimizer_name in optimizers:
            for num_epochs in epochs_list:
                print("=====")
                print(f"Configuration: RNN Hidden Size={hidden_size}, Pooling={pooling}, Optimizer={optimizer_name}, Epochs={num_epochs}")
                model = RNNModel(input_size=X_train.shape[1], hidden_size=hidden_size, num_layers=1, output_size=1, pooling=pooling)
                criterion = nn.BCEWithLogitsLoss()

                if optimizer_name == 'SGD':
                    optimizer = optim.SGD(model.parameters(), lr=0.01)
                elif optimizer_name == 'RMSprop':
                    optimizer = optim.RMSprop(model.parameters(), lr=0.01)
                elif optimizer_name == 'Adam':
                    optimizer = optim.Adam(model.parameters(), lr=0.01)

                scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=3, verbose=True)
                accuracy = train_model(model, optimizer, criterion, scheduler, num_epochs, early_stopper, train_loader, test_loader)

                results_rnn.append({
                    'Model Type': 'RNN',
                    'Hidden Size': hidden_size,
                    'Pooling': pooling,
                    'Optimizer': optimizer_name,
                    'Epochs': num_epochs,
                    'Accuracy': accuracy
                })

results_rnn_df = pd.DataFrame(results_rnn)
results_rnn_df.to_csv('Experiment_results_rnn.csv', index=False)

```

Output:

```
=====
Configuration: RNN Hidden Size=16, Pooling=max, Optimizer=SGD, Epochs=5
Epoch 1/5, Train Loss: 0.4404, Test Loss: 0.3387, Accuracy: 0.8854
Epoch 2/5, Train Loss: 0.2995, Test Loss: 0.2821, Accuracy: 0.8909
Epoch 3/5, Train Loss: 0.2625, Test Loss: 0.2573, Accuracy: 0.8944
Epoch 4/5, Train Loss: 0.2453, Test Loss: 0.2457, Accuracy: 0.8966
Epoch 5/5, Train Loss: 0.2362, Test Loss: 0.2399, Accuracy: 0.8986
=====

Configuration: RNN Hidden Size=16, Pooling=max, Optimizer=SGD, Epochs=50
Epoch 1/50, Train Loss: 0.4194, Test Loss: 0.3207, Accuracy: 0.8848
Epoch 2/50, Train Loss: 0.2850, Test Loss: 0.2703, Accuracy: 0.8942
Epoch 3/50, Train Loss: 0.2532, Test Loss: 0.2518, Accuracy: 0.8993
Epoch 4/50, Train Loss: 0.2415, Test Loss: 0.2451, Accuracy: 0.8987
Epoch 5/50, Train Loss: 0.2364, Test Loss: 0.2423, Accuracy: 0.9011
Epoch 6/50, Train Loss: 0.2340, Test Loss: 0.2407, Accuracy: 0.9021
Epoch 7/50, Train Loss: 0.2325, Test Loss: 0.2395, Accuracy: 0.9018
Epoch 8/50, Train Loss: 0.2313, Test Loss: 0.2384, Accuracy: 0.9019
Epoch 9/50, Train Loss: 0.2298, Test Loss: 0.2374, Accuracy: 0.9021
Epoch 10/50, Train Loss: 0.2285, Test Loss: 0.2365, Accuracy: 0.9020
Epoch 11/50, Train Loss: 0.2276, Test Loss: 0.2355, Accuracy: 0.9025
Epoch 12/50, Train Loss: 0.2266, Test Loss: 0.2347, Accuracy: 0.9027
Epoch 13/50, Train Loss: 0.2257, Test Loss: 0.2339, Accuracy: 0.9024
Epoch 14/50, Train Loss: 0.2250, Test Loss: 0.2330, Accuracy: 0.9026
...
Epoch 15/350, Train Loss: 0.1615, Test Loss: 0.2183, Accuracy: 0.9051
Epoch 16/350, Train Loss: 0.1610, Test Loss: 0.2183, Accuracy: 0.9052
Epoch 17/350, Train Loss: 0.1610, Test Loss: 0.2183, Accuracy: 0.9052
Early stopping triggered
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Analisis:

Kode ini menjalankan eksperimen untuk melatih model RNN dengan berbagai konfigurasi parameter dan menyimpan hasilnya dalam file CSV. Eksperimen ini menguji beberapa kombinasi parameter, seperti ukuran lapisan tersembunyi (`hidden_size`), metode pooling (`max` atau `average`), jenis optimizer (`SGD`, `RMSprop`, atau `Adam`), dan jumlah epoch (misalnya 5, 50, hingga 350). Tujuannya adalah menemukan konfigurasi terbaik yang memberikan akurasi tertinggi pada data pengujian. Untuk setiap kombinasi parameter, model RNN dibuat dengan satu lapisan RNN, dan metode pooling digunakan untuk merangkum output dari lapisan RNN. Fungsi loss yang digunakan adalah `BCEWithLogitsLoss` untuk menangani masalah klasifikasi biner. Optimizer disesuaikan berdasarkan parameter yang dipilih, dan scheduler digunakan untuk menyesuaikan laju pembelajaran jika performa model stagnan. Early stopping diterapkan untuk menghentikan pelatihan lebih awal jika tidak ada peningkatan performa selama 10 epoch berturut-turut. Setelah setiap eksperimen selesai, akurasi model dicatat bersama dengan konfigurasi yang digunakan. Semua hasil disimpan dalam `DataFrame` pandas dan kemudian diekspor ke file CSV bernama `Experiment_results_rnn.csv`. File ini memudahkan analisis lebih lanjut untuk menentukan kombinasi parameter terbaik bagi model.

```
# Define Deep RNN model
class DeepRNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, pooling='max'):
        super(DeepRNNModel, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.pooling = pooling
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        if self.pooling == 'max':
            out = torch.max(out, dim=1).values
        elif self.pooling == 'avg':
            out = torch.mean(out, dim=1)
        out = self.fc(out)
        return out
```

Analisis:

Model DeepRNNModel adalah jaringan saraf berulang (RNN) bertingkat yang dirancang untuk memproses data berurutan seperti teks atau data waktu. Data melewati beberapa lapisan RNN untuk mengenali pola yang lebih kompleks. Setelah itu, hasilnya diringkas menggunakan metode pooling, seperti max pooling (mengambil nilai maksimum) atau average pooling (mengambil rata-rata), lalu diteruskan ke lapisan akhir untuk menghasilkan prediksi. Model ini cocok untuk tugas seperti klasifikasi teks atau analisis data waktu karena kemampuannya menangkap pola yang lebih mendalam.

```
# Training function
def train_model(model, optimizer, criterion, scheduler, num_epochs, early_stopper, train_loader, test_loader):
    best_loss = float('inf')
    patience_counter = 0
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        for X_batch, y_batch in tqdm(train_loader):
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            optimizer.zero_grad()
            outputs = model(X_batch.unsqueeze(1))
            loss = criterion(outputs.squeeze(), y_batch)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)

        model.eval()
        test_loss = 0
        correct = 0
        total = 0
        with torch.no_grad():
            for X_batch, y_batch in test_loader:
                X_batch, y_batch = X_batch.to(device), y_batch.to(device)
                outputs = model(X_batch.unsqueeze(1))
                loss = criterion(outputs.squeeze(), y_batch)
                test_loss += loss.item()
                predictions = torch.round(torch.sigmoid(outputs.squeeze()))
                correct += (predictions == y_batch).sum().item()
                total += y_batch.size(0)
            test_loss /= len(test_loader)
            accuracy = correct / total
```

```
        scheduler.step(test_loss)

        if test_loss < best_loss:
            best_loss = test_loss
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter > early_stopper:
            print("Early stopping triggered")
            break

        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}, Accuracy: {
            accuracy:.4f}")

    return accuracy
```

✓ 0.0s

Python

Analisis:

Fungsi `train_model` digunakan untuk melatih model dengan data pelatihan dan mengukur kinerjanya pada data pengujian. Proses dimulai dengan menempatkan model di perangkat yang tersedia, seperti GPU (jika ada) atau CPU. Pelatihan dilakukan selama beberapa epoch, di mana setiap epoch terdiri dari dua tahap: pelatihan model dengan data pelatihan dan evaluasi model dengan data pengujian. Pada tahap pelatihan, model memproses data dalam batch dari `train_loader`. Model menghasilkan output, menghitung loss menggunakan fungsi tertentu (criterion), dan memperbarui parameter model melalui backpropagation.

dengan bantuan optimizer. Setelah semua batch diproses, rata-rata loss dihitung sebagai ukuran kinerja model selama pelatihan. Tahap evaluasi dilakukan dengan model dalam mode evaluasi, di mana data pengujian diproses tanpa menghitung gradien. Pada tahap ini, loss dan akurasi model dihitung. Akurasi dihitung dengan membandingkan hasil prediksi model dengan label sebenarnya. Fungsi ini juga menggunakan scheduler untuk menyesuaikan laju pembelajaran berdasarkan kinerja pada data pengujian. Early stopping diterapkan untuk menghentikan pelatihan jika performa model tidak meningkat setelah sejumlah epoch tertentu, menghindari pemborosan waktu dan sumber daya. Hasil dari setiap epoch, seperti loss dan akurasi, dicetak untuk memantau kinerja model. Jika model mencapai performa terbaiknya, hasil akhir berupa akurasi pada data pengujian dikembalikan. Fungsi ini memastikan model dilatih dengan efisien dan berhenti tepat waktu saat tidak ada peningkatan signifikan.

```
# Run experiments for Deep RNN and save results
results_deep_rnn = []
for hidden_size in hidden_sizes:
    for pooling in poolings:
        for optimizer_name in optimizers:
            for num_epochs in epochs_list:
                print("=====")
                print(f"\nConfiguration: Deep RNN Hidden Size={hidden_size}, Pooling={pooling}, Optimizer={optimizer_name}, Num Epochs={num_epochs}")
                model = DeepRNNModel(input_size=X_train.shape[1], hidden_size=hidden_size, num_layers=3, output_size=10)
                criterion = nn.BCEWithLogitsLoss()

                if optimizer_name == 'SGD':
                    optimizer = optim.SGD(model.parameters(), lr=0.01)
                elif optimizer_name == 'RMSprop':
                    optimizer = optim.RMSprop(model.parameters(), lr=0.01)
                elif optimizer_name == 'Adam':
                    optimizer = optim.Adam(model.parameters(), lr=0.01)

                scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=3, verbose=True)
                accuracy = train_model(model, optimizer, criterion, scheduler, num_epochs, early_stopper, train_loader)

                results_deep_rnn.append({
                    'Model Type': 'Deep RNN',
                    'Hidden Size': hidden_size,
                    'Pooling': pooling,
                    'Optimizer': optimizer_name,
                    'Epochs': num_epochs,
                    'Accuracy': accuracy
                })

results_deep_rnn_df = pd.DataFrame(results_deep_rnn)
results_deep_rnn_df.to_csv('Experiment_results_deep_rnn.csv', index=False)
```

Output:

```

=====
Configuration: Deep RNN Hidden Size=16, Pooling=max, Optimizer=SGD, Epochs=5
Epoch 1/5, Train Loss: 0.3949, Test Loss: 0.3591, Accuracy: 0.8794
Epoch 2/5, Train Loss: 0.3388, Test Loss: 0.3374, Accuracy: 0.8794
Epoch 3/5, Train Loss: 0.3067, Test Loss: 0.2964, Accuracy: 0.8818
Epoch 4/5, Train Loss: 0.2666, Test Loss: 0.2627, Accuracy: 0.8943
Epoch 5/5, Train Loss: 0.2437, Test Loss: 0.2471, Accuracy: 0.8974
=====

```

```

Configuration: Deep RNN Hidden Size=16, Pooling=max, Optimizer=SGD, Epochs=50
Epoch 1/50, Train Loss: 0.4028, Test Loss: 0.3413, Accuracy: 0.8794
Epoch 2/50, Train Loss: 0.3096, Test Loss: 0.2944, Accuracy: 0.8823
Epoch 3/50, Train Loss: 0.2644, Test Loss: 0.2584, Accuracy: 0.8933
Epoch 4/50, Train Loss: 0.2418, Test Loss: 0.2456, Accuracy: 0.8980
Epoch 5/50, Train Loss: 0.2336, Test Loss: 0.2402, Accuracy: 0.9015
Epoch 6/50, Train Loss: 0.2299, Test Loss: 0.2375, Accuracy: 0.9014
Epoch 7/50, Train Loss: 0.2274, Test Loss: 0.2358, Accuracy: 0.9021
Epoch 8/50, Train Loss: 0.2260, Test Loss: 0.2343, Accuracy: 0.9017
Epoch 9/50, Train Loss: 0.2243, Test Loss: 0.2330, Accuracy: 0.9020
Epoch 10/50, Train Loss: 0.2229, Test Loss: 0.2320, Accuracy: 0.9025
Epoch 11/50, Train Loss: 0.2217, Test Loss: 0.2311, Accuracy: 0.9029
Epoch 12/50, Train Loss: 0.2209, Test Loss: 0.2302, Accuracy: 0.9025
Epoch 13/50, Train Loss: 0.2202, Test Loss: 0.2294, Accuracy: 0.9026
Epoch 14/50, Train Loss: 0.2188, Test Loss: 0.2285, Accuracy: 0.9029
...
Epoch 18/350, Train Loss: 0.1564, Test Loss: 0.2242, Accuracy: 0.9010
Epoch 19/350, Train Loss: 0.1558, Test Loss: 0.2242, Accuracy: 0.9015
Epoch 20/350, Train Loss: 0.1565, Test Loss: 0.2242, Accuracy: 0.9016
Early stopping triggered
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Analysis:

Kode ini menjalankan eksperimen melatih model Deep RNN dengan berbagai kombinasi parameter, seperti ukuran lapisan tersembunyi, metode pooling (max atau average), jenis optimizer (SGD, RMSprop, Adam), dan jumlah epoch. Model dilatih menggunakan tiga lapisan RNN untuk menangkap pola kompleks, dengan fungsi loss BCEWithLogitsLoss, scheduler untuk menyesuaikan laju pembelajaran, dan early stopping untuk menghentikan pelatihan jika tidak ada peningkatan. Hasil akurasi dari setiap konfigurasi disimpan ke file CSV Experiment_results_deep_rnn.csv untuk analisis lebih lanjut.

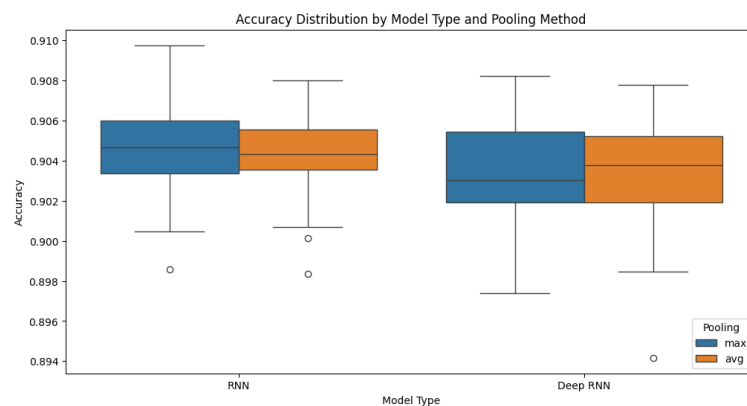
```

# Combine results and visualize
def visualize_results():
    combined_results = pd.concat([results_rnn_df, results_deep_rnn_df])

    # Plot Accuracy Distribution by Model Type
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=combined_results, x='Model Type', y='Accuracy', hue='Pooling')
    plt.title('Accuracy Distribution by Model Type and Pooling Method')
    plt.ylabel('Accuracy')
    plt.xlabel('Model Type')
    plt.legend(title='Pooling', loc='lower right')
    plt.savefig('accuracy_distribution.png')
    plt.show()
visualize_results()

```

Output:



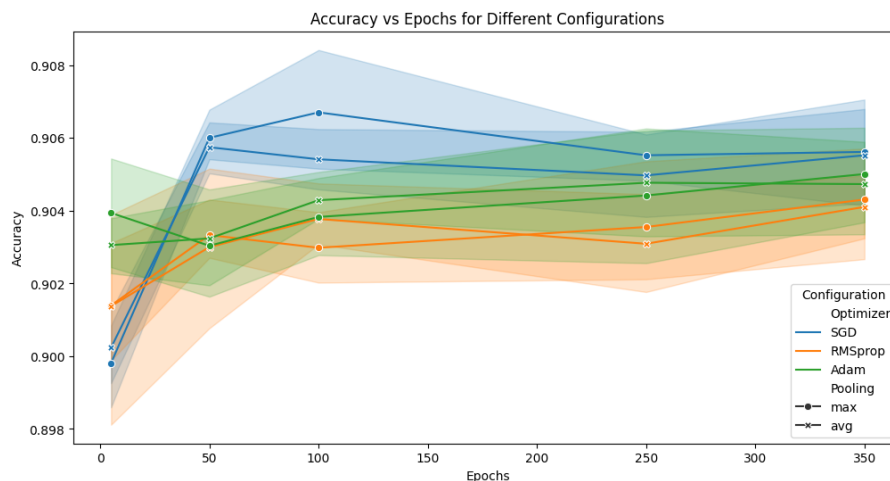
Analisis:

Fungsi ini menggabungkan hasil eksperimen model RNN dan Deep RNN, lalu membuat grafik boxplot untuk membandingkan distribusi akurasi berdasarkan jenis model dan metode pooling (max atau average). Grafik menunjukkan variasi akurasi untuk setiap kombinasi model dan pooling, membantu memahami performa terbaik. Hasil grafik disimpan sebagai file `accuracy_distribution.png` dan ditampilkan untuk analisis lebih lanjut.

```
# Combine results and visualize
def visualize_results():
    combined_results = pd.concat([results_rnn_df, results_deep_rnn_df])

    # Plot Accuracy Distribution by Model Type
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=combined_results, x='Model Type', y='Accuracy', hue='Pooling')
    plt.title('Accuracy Distribution by Model Type and Pooling Method')
    plt.ylabel('Accuracy')
    plt.xlabel('Model Type')
    plt.legend(title='Pooling', loc='lower right')
    plt.savefig('accuracy_distribution.png')
    plt.show()
visualize_results()
```

Output:



Analisis:

Kode ini digunakan untuk memvisualisasikan hasil percobaan model *Bidirectional RNN* dengan grafik. Fungsi `visualize_results` mengambil DataFrame hasil percobaan (`results_df`) dan membuat grafik garis menggunakan Seaborn. Grafik ini menunjukkan hubungan antara jumlah epoch pelatihan dan akurasi model. Garis-garis dalam grafik dikelompokkan berdasarkan jenis optimizer yang digunakan, dengan gaya garis dan tanda (*markers*) yang berbeda untuk membedakan metode pooling (max atau average). Grafik ini memberikan cara yang jelas untuk membandingkan performa model dengan berbagai konfigurasi, sehingga memudahkan untuk melihat pengaruh jumlah epoch, metode pooling, dan jenis optimizer terhadap akurasi. Hasil akhirnya ditampilkan dalam grafik yang memberikan wawasan visual untuk analisis lebih lanjut.

```
# Top 10 configurations by accuracy
combined_results = pd.concat([results_rnn_df, results_deep_rnn_df])
top_10 = combined_results.sort_values(by='Accuracy', ascending=False).head(10)
print("Top 10 Configurations by Accuracy:")

# Save as table
from tabulate import tabulate  Import "tabulate" could not be resolved from source
print(tabulate(top_10, headers='keys', tablefmt='pretty', showindex=False))
```

✓ 0.0s

Python

Output:

Top 10 Configurations by Accuracy:

Model Type	Hidden Size	Pooling	Optimizer	Epochs	Accuracy
RNN	64	max	SGD	100	0.9097644586973349
RNN	32	max	SGD	100	0.9087692137564968
Deep RNN	64	max	SGD	350	0.9082162999004755
RNN	16	avg	RMSprop	50	0.907995134358067
RNN	64	avg	Adam	250	0.9077739688156585
RNN	64	max	SGD	350	0.9077739688156585
Deep RNN	64	avg	SGD	350	0.9077739688156585
Deep RNN	32	max	SGD	50	0.9077739688156585
RNN	16	max	Adam	350	0.9076633860444543
Deep RNN	16	max	RMSprop	250	0.9073316377308416

Analisis:

Kode ini menampilkan 10 konfigurasi model dengan akurasi tertinggi dari hasil eksperimen model RNN dan Deep RNN. Data diurutkan berdasarkan akurasi secara menurun, lalu 10 konfigurasi terbaik ditampilkan dalam bentuk tabel yang rapi menggunakan library tabulate. Tabel ini memuat informasi seperti jenis model, ukuran lapisan tersembunyi, metode pooling, optimizer, jumlah epoch, dan akurasi yang dicapai. Hasil ini membantu melihat konfigurasi yang paling optimal untuk performa terbaik.