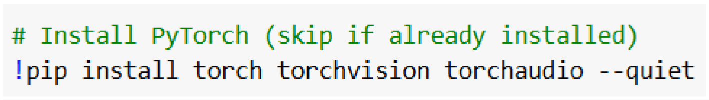
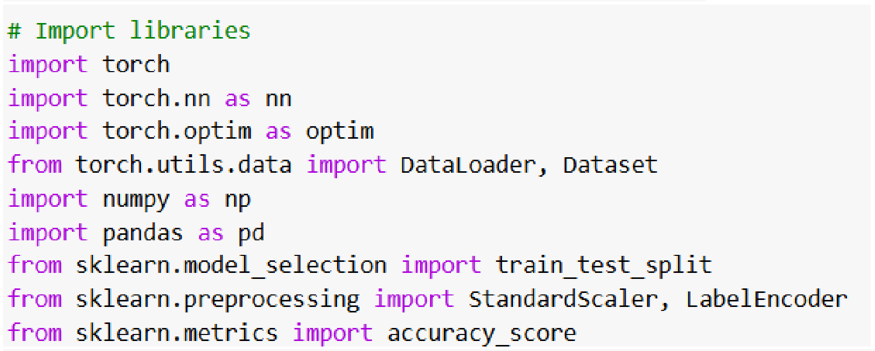
Nama: Andi Cleopatra Maryam Jamila

Nim: 1103213071

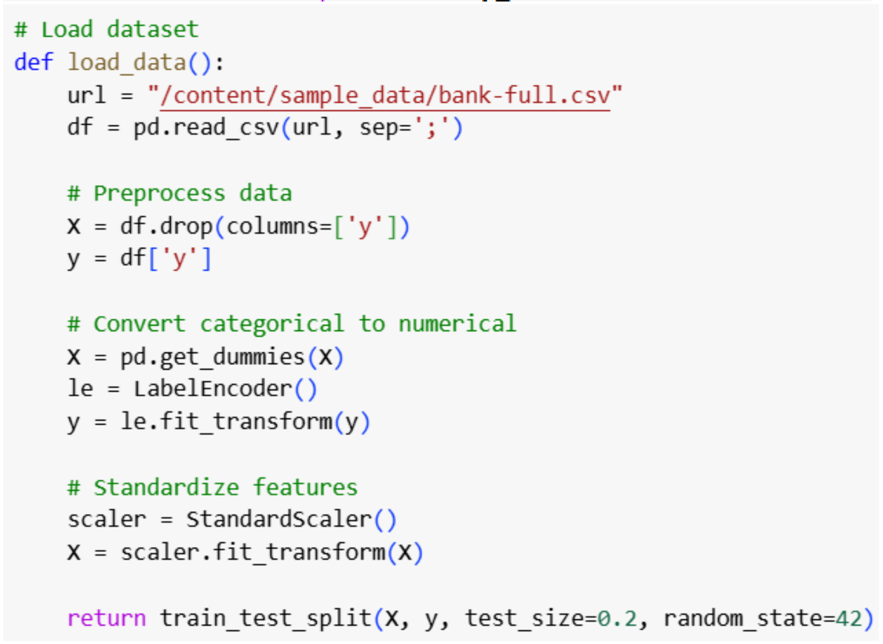
Analisis Bidirectional RNN Model Week 14



```
# Install PyTorch (skip if already installed)
!pip install torch torchvision torchaudio --quiet
```



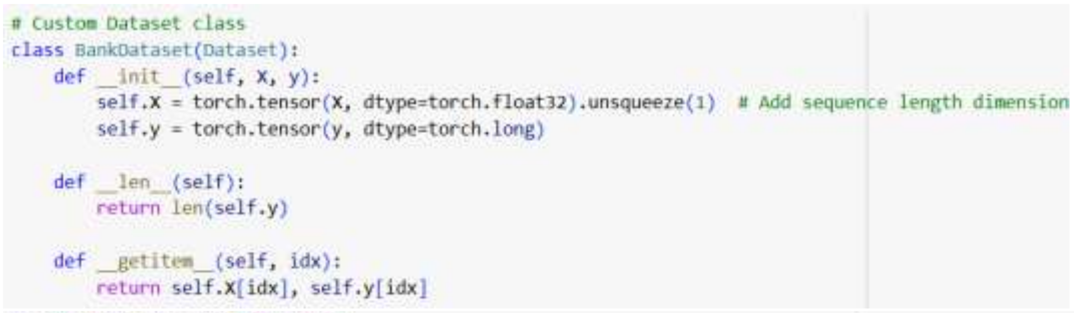
```
# Import libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
```



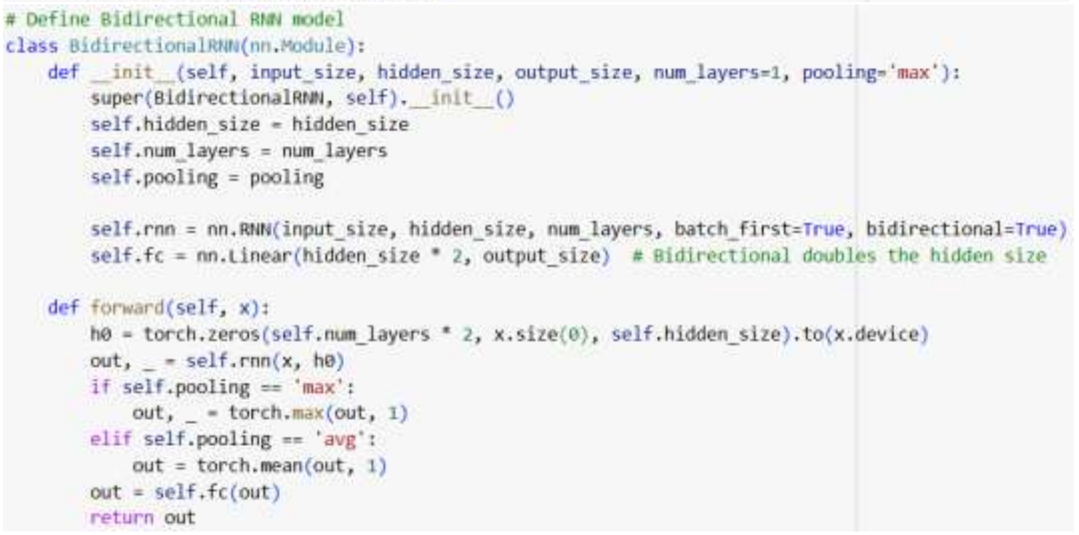
```
# Load dataset
def load_data():
    url = "/content/sample_data/bank-full.csv"
    df = pd.read_csv(url, sep=';')

    # Preprocess data
    X = df.drop(columns=['y'])
    y = df['y']

    # Convert categorical to numerical
    X = pd.get_dummies(X)
    le = LabelEncoder()
    y = le.fit_transform(y)

    # Standardize features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    return train_test_split(X, y, test_size=0.2, random_state=42)
```

Analisis:

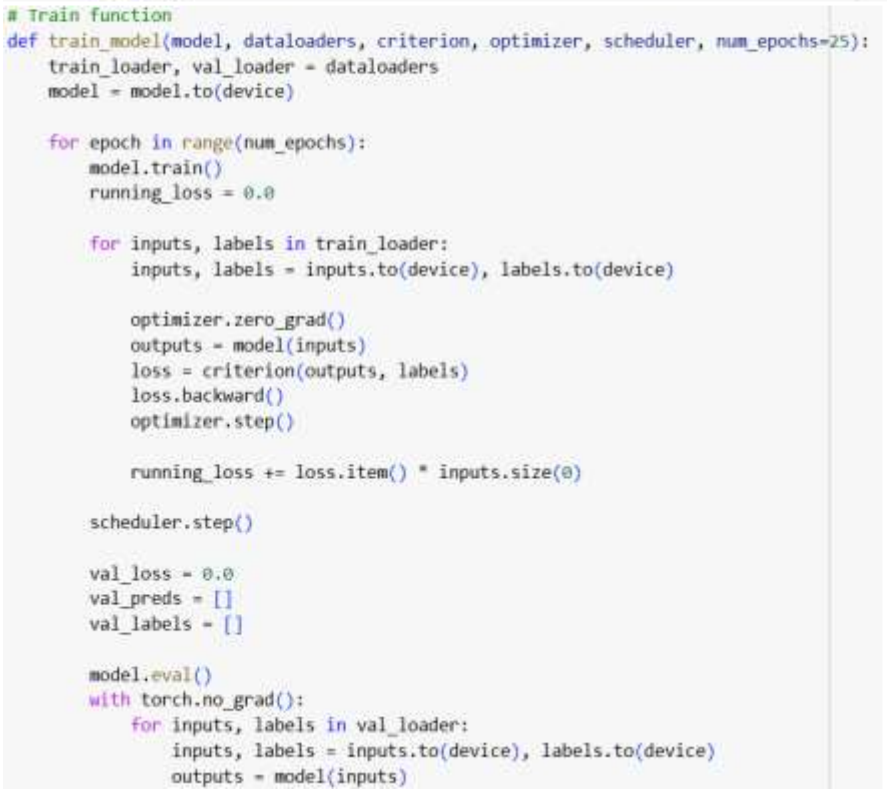
Memuat dan memproses dataset Bank Marketing dari file CSV. Dataset diproses dengan membuang kolom target (y) dari fitur input (X) dan mengubah data kategorikal menjadi format numerik menggunakan one-hot encoding melalui fungsi pd.get_dummies. Kolom target (y) dikodekan ulang menjadi nilai numerik menggunakan LabelEncoder. Selanjutnya, fitur-fitur dalam X dinormalisasi menggunakan StandardScaler untuk memastikan semua variabel berada dalam skala yang sama, yang penting untuk mempercepat konvergensi selama pelatihan model. Dataset yang telah diproses kemudian dibagi menjadi data pelatihan dan pengujian dengan rasio 80:20 menggunakan fungsi train_test_split. Proses ini memastikan dataset siap untuk digunakan dalam membangun model pembelajaran mesin dengan performa optimal.



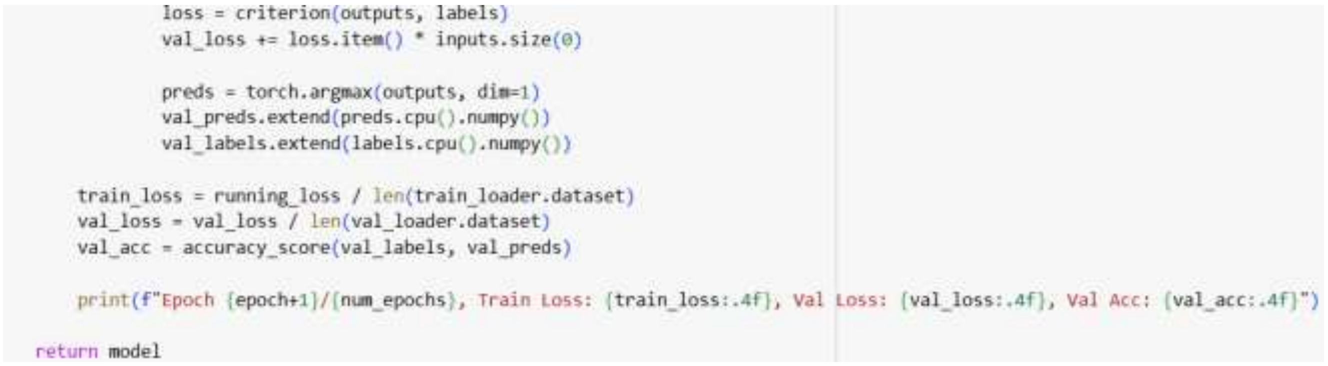
```python
# Custom Dataset class
class BankDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32).unsqueeze(1)  # Add sequence length dimension
        self.y = torch.tensor(y, dtype=torch.long)

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

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



```python
# Define Bidirectional RNN model
class BidirectionalRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1, pooling='max'):
        super(BidirectionalRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.pooling = pooling

        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)
        self.fc = nn.Linear(hidden_size * 2, output_size)  # Bidirectional doubles the hidden size

    def forward(self, x):
        h0 = torch.zeros(self.num_layers * 2, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        if self.pooling == 'max':
            out, _ = torch.max(out, 1)
        elif self.pooling == 'avg':
            out = torch.mean(out, 1)
        out = self.fc(out)
        return out
```



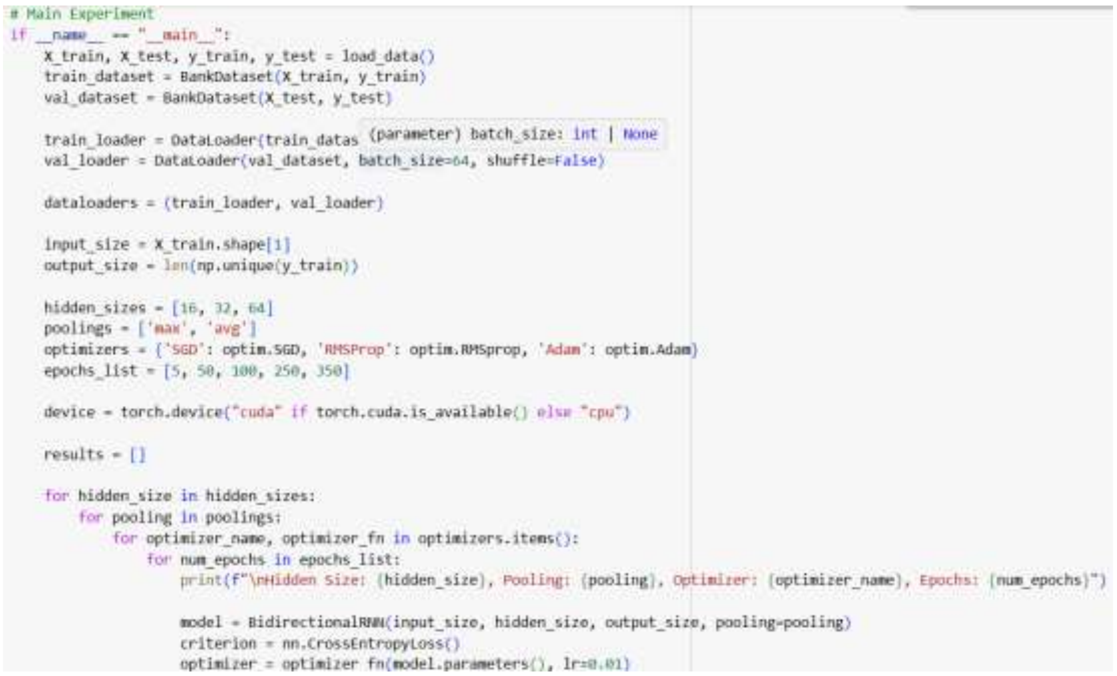
```python
# Train function
def train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs=25):
    train_loader, val_loader = dataloaders
    model = model.to(device)

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        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() * inputs.size(0)

        scheduler.step()

        val_loss = 0.0
        val_preds = []
        val_labels = []

        model.eval()
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
```



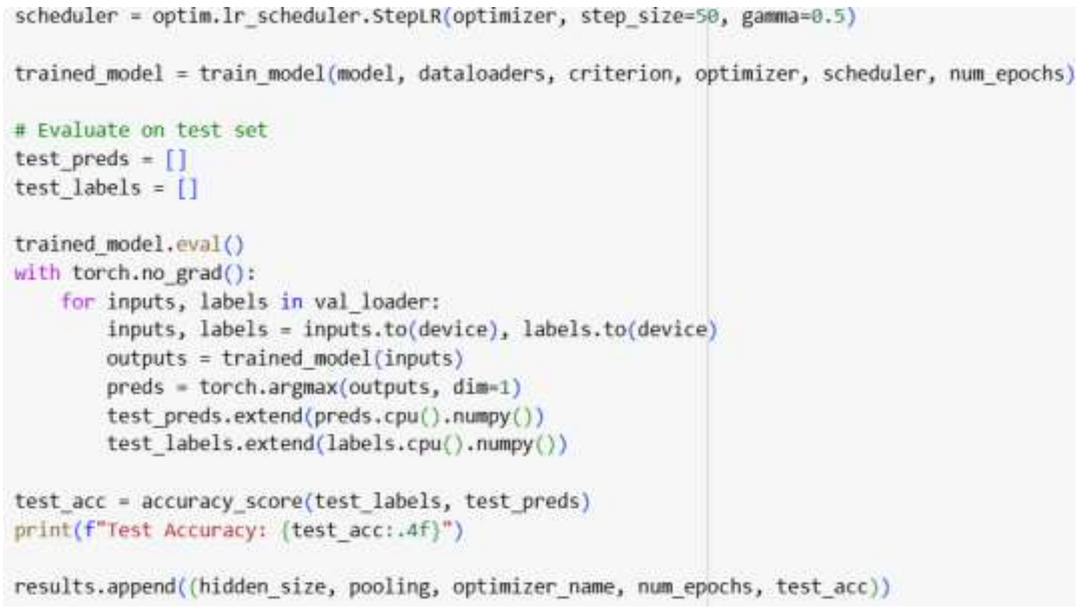
```
            loss = criterion(outputs, labels)
            val_loss += loss.item() * inputs.size(0)

            preds = torch.argmax(outputs, dim=1)
            val_preds.extend(preds.cpu().numpy())
            val_labels.extend(labels.cpu().numpy())

    train_loss = running_loss / len(train_loader.dataset)
    val_loss = val_loss / len(val_loader.dataset)
    val_acc = accuracy_score(val_labels, val_preds)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

  return model
```

Analisis:

terdiri dari tiga komponen utama: (1) Kelas Dataset: BankDataset dibuat untuk menyiapkan data input dan label dalam format yang sesuai untuk digunakan oleh PyTorch, dengan dimensi tambahan untuk panjang urutan yang diperlukan oleh model RNN. (2) Model RNN Bidirectional: Kelas BidirectionalRNN dirancang menggunakan PyTorch untuk membangun RNN dua arah, yang memungkinkan model untuk mempertimbangkan konteks dari kedua arah (ke depan dan ke belakang) dalam data sekuensial. Ukuran tersembunyi dikalikan dua karena penggabungan dua arah. Selain itu, model memiliki opsi pooling (max atau average) untuk meringkas keluaran temporal menjadi representasi tetap. (3) Fungsi Pelatihan: train_model menangani proses pelatihan dan validasi dengan optimasi gradien menggunakan kriteria loss, pengaturan langkah learning rate scheduler, dan pelacakan akurasi validasi di setiap epoch. Kode ini sangat fleksibel untuk mengevaluasi kombinasi arsitektur dan parameter pada data sekuensial.

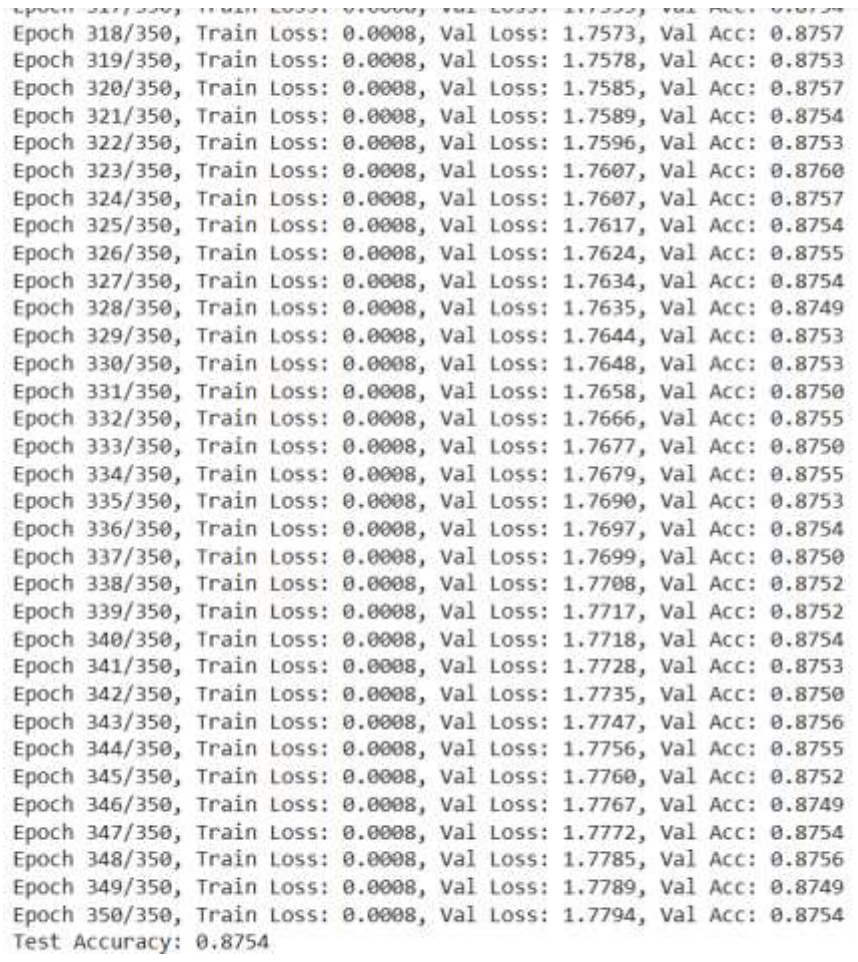


```
# Main Experiment
if __name__ == "__main__":
    X_train, X_test, y_train, y_test = load_data()
    train_dataset = BankDataset(X_train, y_train)
    val_dataset = BankDataset(X_test, y_test)

    train_loader = DataLoader(train_datas (parameter) batch_size: int | None
    val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

    dataloaders = (train_loader, val_loader)

    input_size = X_train.shape[1]
    output_size = len(np.unique(y_train))

    hidden_sizes = [16, 32, 64]
    poolings = ['max', 'avg']
    optimizers = {'SGD': optim.SGD, 'RMSProp': optim.RMSprop, 'Adam': optim.Adam}
    epochs_list = [5, 50, 100, 250, 350]

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

    results = []

    for hidden_size in hidden_sizes:
        for pooling in poolings:
            for optimizer_name, optimizer_fn in optimizers.items():
                for num_epochs in epochs_list:
                    print(f"\nHidden Size: {hidden_size}, Pooling: {pooling}, Optimizer: {optimizer_name}, Epochs: {num_epochs}")

                    model = BidirectionalRNN(input_size, hidden_size, output_size, pooling=pooling)
                    criterion = nn.CrossEntropyLoss()
                    optimizer = optimizer_fn(model.parameters(), lr=0.01)
```



```
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.5)

trained_model = train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs)

# Evaluate on test set
test_preds = []
test_labels = []

trained_model.eval()
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = trained_model(inputs)
        preds = torch.argmax(outputs, dim=1)
        test_preds.extend(preds.cpu().numpy())
        test_labels.extend(labels.cpu().numpy())

test_acc = accuracy_score(test_labels, test_preds)
print(f"Test Accuracy: {test_acc:.4f}")

results.append((hidden_size, pooling, optimizer_name, num_epochs, test_acc))
```

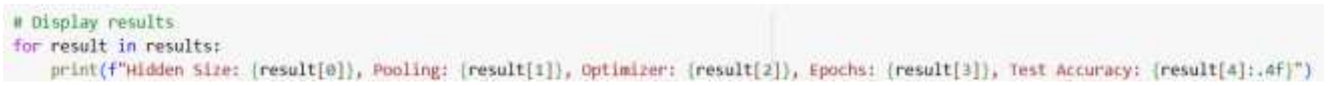
Output:



```
Epoch 318/350, Train Loss: 0.0008, Val Loss: 1.7573, Val Acc: 0.8757
Epoch 319/350, Train Loss: 0.0008, Val Loss: 1.7578, Val Acc: 0.8753
Epoch 320/350, Train Loss: 0.0008, Val Loss: 1.7585, Val Acc: 0.8757
Epoch 321/350, Train Loss: 0.0008, Val Loss: 1.7589, Val Acc: 0.8754
Epoch 322/350, Train Loss: 0.0008, Val Loss: 1.7596, Val Acc: 0.8753
Epoch 323/350, Train Loss: 0.0008, Val Loss: 1.7607, Val Acc: 0.8760
Epoch 324/350, Train Loss: 0.0008, Val Loss: 1.7607, Val Acc: 0.8757
Epoch 325/350, Train Loss: 0.0008, Val Loss: 1.7617, Val Acc: 0.8754
Epoch 326/350, Train Loss: 0.0008, Val Loss: 1.7624, Val Acc: 0.8755
Epoch 327/350, Train Loss: 0.0008, Val Loss: 1.7634, Val Acc: 0.8754
Epoch 328/350, Train Loss: 0.0008, Val Loss: 1.7635, Val Acc: 0.8749
Epoch 329/350, Train Loss: 0.0008, Val Loss: 1.7644, Val Acc: 0.8753
Epoch 330/350, Train Loss: 0.0008, Val Loss: 1.7648, Val Acc: 0.8753
Epoch 331/350, Train Loss: 0.0008, Val Loss: 1.7658, Val Acc: 0.8750
Epoch 332/350, Train Loss: 0.0008, Val Loss: 1.7666, Val Acc: 0.8755
Epoch 333/350, Train Loss: 0.0008, Val Loss: 1.7677, Val Acc: 0.8750
Epoch 334/350, Train Loss: 0.0008, Val Loss: 1.7679, Val Acc: 0.8755
Epoch 335/350, Train Loss: 0.0008, Val Loss: 1.7690, Val Acc: 0.8753
Epoch 336/350, Train Loss: 0.0008, Val Loss: 1.7697, Val Acc: 0.8754
Epoch 337/350, Train Loss: 0.0008, Val Loss: 1.7699, Val Acc: 0.8750
Epoch 338/350, Train Loss: 0.0008, Val Loss: 1.7708, Val Acc: 0.8752
Epoch 339/350, Train Loss: 0.0008, Val Loss: 1.7717, Val Acc: 0.8752
Epoch 340/350, Train Loss: 0.0008, Val Loss: 1.7718, Val Acc: 0.8754
Epoch 341/350, Train Loss: 0.0008, Val Loss: 1.7728, Val Acc: 0.8753
Epoch 342/350, Train Loss: 0.0008, Val Loss: 1.7735, Val Acc: 0.8750
Epoch 343/350, Train Loss: 0.0008, Val Loss: 1.7747, Val Acc: 0.8756
Epoch 344/350, Train Loss: 0.0008, Val Loss: 1.7756, Val Acc: 0.8755
Epoch 345/350, Train Loss: 0.0008, Val Loss: 1.7760, Val Acc: 0.8752
Epoch 346/350, Train Loss: 0.0008, Val Loss: 1.7767, Val Acc: 0.8749
Epoch 347/350, Train Loss: 0.0008, Val Loss: 1.7772, Val Acc: 0.8754
Epoch 348/350, Train Loss: 0.0008, Val Loss: 1.7785, Val Acc: 0.8756
Epoch 349/350, Train Loss: 0.0008, Val Loss: 1.7789, Val Acc: 0.8749
Epoch 350/350, Train Loss: 0.0008, Val Loss: 1.7794, Val Acc: 0.8754
Test Accuracy: 0.8754
```

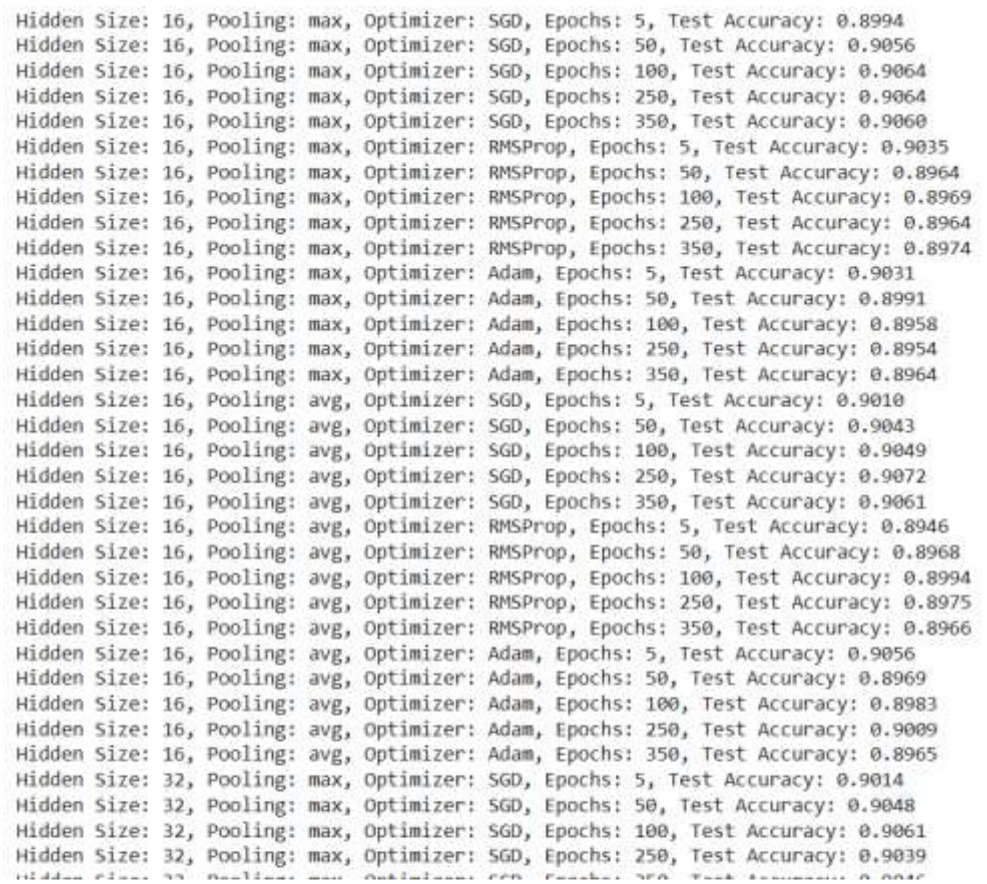
Analisis:

Mengimplementasikan eksperimen sistematis menggunakan model Bidirectional RNN untuk klasifikasi dataset Bank Marketing, dengan mengevaluasi performa model berdasarkan variasi parameter arsitektur dan pelatihan. Parameter yang diuji meliputi ukuran hidden layer (hidden_sizes), jenis pooling yang digunakan (MaxPooling atau AvgPooling), algoritma optimisasi (SGD, RMSProp, Adam), serta jumlah epoch pelatihan (5, 50, 100, 250, 350). Setiap kombinasi parameter diterapkan, dan model dilatih menggunakan *dataloader* untuk membagi data menjadi batch, dengan scheduler yang menyesuaikan learning rate setiap 50 epoch. Akurasi model diuji pada data validasi menggunakan metrik akurasi, dan hasilnya dicatat untuk setiap konfigurasi. Eksperimen ini dirancang untuk memahami dampak parameter terhadap performa klasifikasi, memungkinkan pemilihan konfigurasi optimal untuk model.



```python
# Display results
for result in results:
    print(f"Hidden Size: {result[0]}, Pooling: {result[1]}, Optimizer: {result[2]}, Epochs: {result[3]}, Test Accuracy: {result[4]:.4f}")
```

Output:

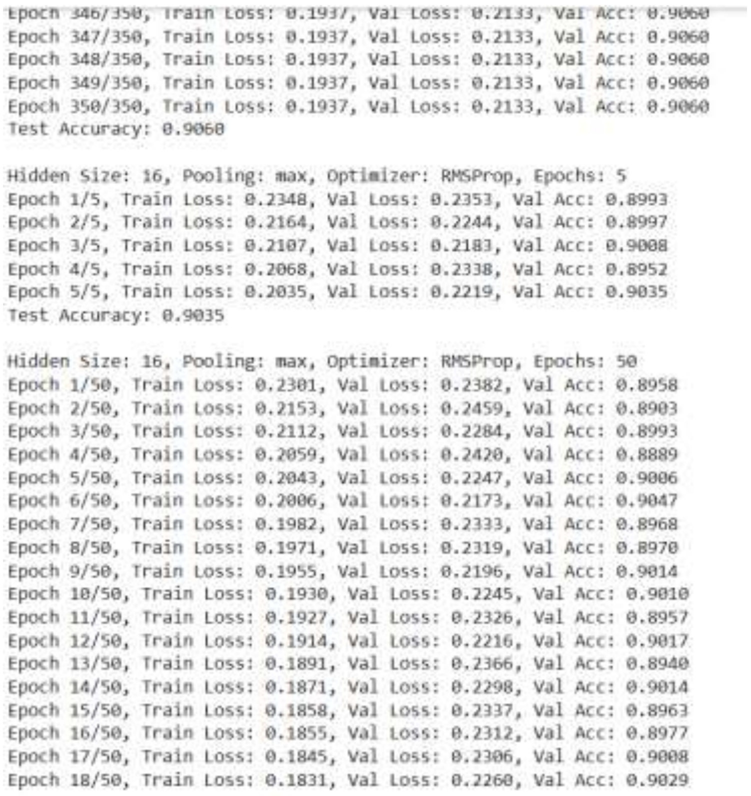


```
Hidden Size: 16, Pooling: max, Optimizer: SGD, Epochs: 5, Test Accuracy: 0.8994
Hidden Size: 16, Pooling: max, Optimizer: SGD, Epochs: 50, Test Accuracy: 0.9056
Hidden Size: 16, Pooling: max, Optimizer: SGD, Epochs: 100, Test Accuracy: 0.9064
Hidden Size: 16, Pooling: max, Optimizer: SGD, Epochs: 250, Test Accuracy: 0.9064
Hidden Size: 16, Pooling: max, Optimizer: SGD, Epochs: 350, Test Accuracy: 0.9060
Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 5, Test Accuracy: 0.9035
Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 50, Test Accuracy: 0.8964
Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 100, Test Accuracy: 0.8969
Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 250, Test Accuracy: 0.8964
Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 350, Test Accuracy: 0.8974
Hidden Size: 16, Pooling: max, Optimizer: Adam, Epochs: 5, Test Accuracy: 0.9031
Hidden Size: 16, Pooling: max, Optimizer: Adam, Epochs: 50, Test Accuracy: 0.8991
Hidden Size: 16, Pooling: max, Optimizer: Adam, Epochs: 100, Test Accuracy: 0.8958
Hidden Size: 16, Pooling: max, Optimizer: Adam, Epochs: 250, Test Accuracy: 0.8954
Hidden Size: 16, Pooling: max, Optimizer: Adam, Epochs: 350, Test Accuracy: 0.8964
Hidden Size: 16, Pooling: avg, Optimizer: SGD, Epochs: 5, Test Accuracy: 0.9010
Hidden Size: 16, Pooling: avg, Optimizer: SGD, Epochs: 50, Test Accuracy: 0.9043
Hidden Size: 16, Pooling: avg, Optimizer: SGD, Epochs: 100, Test Accuracy: 0.9049
Hidden Size: 16, Pooling: avg, Optimizer: SGD, Epochs: 250, Test Accuracy: 0.9072
Hidden Size: 16, Pooling: avg, Optimizer: SGD, Epochs: 350, Test Accuracy: 0.9061
Hidden Size: 16, Pooling: avg, Optimizer: RMSProp, Epochs: 5, Test Accuracy: 0.8946
Hidden Size: 16, Pooling: avg, Optimizer: RMSProp, Epochs: 50, Test Accuracy: 0.8968
Hidden Size: 16, Pooling: avg, Optimizer: RMSProp, Epochs: 100, Test Accuracy: 0.8994
Hidden Size: 16, Pooling: avg, Optimizer: RMSProp, Epochs: 250, Test Accuracy: 0.8975
Hidden Size: 16, Pooling: avg, Optimizer: RMSProp, Epochs: 350, Test Accuracy: 0.8966
Hidden Size: 16, Pooling: avg, Optimizer: Adam, Epochs: 5, Test Accuracy: 0.9056
Hidden Size: 16, Pooling: avg, Optimizer: Adam, Epochs: 50, Test Accuracy: 0.8969
Hidden Size: 16, Pooling: avg, Optimizer: Adam, Epochs: 100, Test Accuracy: 0.8983
Hidden Size: 16, Pooling: avg, Optimizer: Adam, Epochs: 250, Test Accuracy: 0.9009
Hidden Size: 16, Pooling: avg, Optimizer: Adam, Epochs: 350, Test Accuracy: 0.8965
Hidden Size: 32, Pooling: max, Optimizer: SGD, Epochs: 5, Test Accuracy: 0.9014
Hidden Size: 32, Pooling: max, Optimizer: SGD, Epochs: 50, Test Accuracy: 0.9048
Hidden Size: 32, Pooling: max, Optimizer: SGD, Epochs: 100, Test Accuracy: 0.9061
Hidden Size: 32, Pooling: max, Optimizer: SGD, Epochs: 250, Test Accuracy: 0.9039
```

Analisis:

Menampilkan hasil eksperimen dari setiap kombinasi parameter model Bidirectional RNN dalam bentuk yang terstruktur, mencakup ukuran hidden layer (Hidden Size), metode pooling yang digunakan (Pooling), algoritma optimisasi (Optimizer), jumlah epoch pelatihan (Epochs), dan akurasi model pada data pengujian (Test Accuracy). Dengan menyajikan hasil ini, kita dapat dengan mudah membandingkan performa model berdasarkan parameter-parameter yang berbeda. Analisis ini membantu mengidentifikasi kombinasi parameter terbaik untuk mencapai akurasi pengujian tertinggi, memberikan wawasan penting tentang bagaimana ukuran hidden layer, pooling, optimizer, dan jumlah epoch memengaruhi kemampuan model dalam menyelesaikan tugas klasifikasi dataset Bank Marketing.

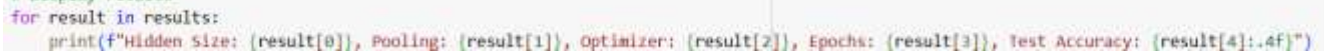
Output:



```
Epoch 346/350, Train Loss: 0.1937, Val Loss: 0.2133, Val Acc: 0.9060
Epoch 347/350, Train Loss: 0.1937, Val Loss: 0.2133, Val Acc: 0.9060
Epoch 348/350, Train Loss: 0.1937, Val Loss: 0.2133, Val Acc: 0.9060
Epoch 349/350, Train Loss: 0.1937, Val Loss: 0.2133, Val Acc: 0.9060
Epoch 350/350, Train Loss: 0.1937, Val Loss: 0.2133, Val Acc: 0.9060
Test Accuracy: 0.9060

Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 5
Epoch 1/5, Train Loss: 0.2348, Val Loss: 0.2353, Val Acc: 0.8993
Epoch 2/5, Train Loss: 0.2164, Val Loss: 0.2244, Val Acc: 0.8997
Epoch 3/5, Train Loss: 0.2107, Val Loss: 0.2183, Val Acc: 0.9008
Epoch 4/5, Train Loss: 0.2068, Val Loss: 0.2338, Val Acc: 0.8952
Epoch 5/5, Train Loss: 0.2035, Val Loss: 0.2219, Val Acc: 0.9035
Test Accuracy: 0.9035

Hidden Size: 16, Pooling: max, Optimizer: RMSProp, Epochs: 50
Epoch 1/50, Train Loss: 0.2301, Val Loss: 0.2382, Val Acc: 0.8958
Epoch 2/50, Train Loss: 0.2153, Val Loss: 0.2459, Val Acc: 0.8903
Epoch 3/50, Train Loss: 0.2112, Val Loss: 0.2284, Val Acc: 0.8993
Epoch 4/50, Train Loss: 0.2059, Val Loss: 0.2420, Val Acc: 0.8889
Epoch 5/50, Train Loss: 0.2043, Val Loss: 0.2247, Val Acc: 0.9006
Epoch 6/50, Train Loss: 0.2006, Val Loss: 0.2173, Val Acc: 0.9047
Epoch 7/50, Train Loss: 0.1982, Val Loss: 0.2333, Val Acc: 0.8968
Epoch 8/50, Train Loss: 0.1971, Val Loss: 0.2319, Val Acc: 0.8970
Epoch 9/50, Train Loss: 0.1955, Val Loss: 0.2196, Val Acc: 0.9014
Epoch 10/50, Train Loss: 0.1930, Val Loss: 0.2245, Val Acc: 0.9010
Epoch 11/50, Train Loss: 0.1927, Val Loss: 0.2326, Val Acc: 0.8957
Epoch 12/50, Train Loss: 0.1914, Val Loss: 0.2216, Val Acc: 0.9017
Epoch 13/50, Train Loss: 0.1891, Val Loss: 0.2366, Val Acc: 0.8940
Epoch 14/50, Train Loss: 0.1871, Val Loss: 0.2298, Val Acc: 0.9014
Epoch 15/50, Train Loss: 0.1858, Val Loss: 0.2337, Val Acc: 0.8963
Epoch 16/50, Train Loss: 0.1855, Val Loss: 0.2312, Val Acc: 0.8977
Epoch 17/50, Train Loss: 0.1845, Val Loss: 0.2306, Val Acc: 0.9008
Epoch 18/50, Train Loss: 0.1831, Val Loss: 0.2260, Val Acc: 0.9029
```

Analisis:

mengimplementasikan eksperimen utama untuk mengevaluasi kinerja model Bidirectional RNN menggunakan dataset Bank Marketing. Eksperimen melibatkan kombinasi berbagai parameter, termasuk ukuran hidden layer (hidden_size), jenis pooling (max dan avg), jenis optimizer (SGD, RMSProp, Adam), dan jumlah epoch pelatihan (5, 50, 100, 250, 350). Data dilatih menggunakan train_loader, sementara val_loader digunakan untuk validasi. Model dilatih dengan loss function CrossEntropyLoss, pengoptimalan learning rate menggunakan scheduler, dan pelacakan performa di setiap iterasi. Setelah pelatihan, model diuji pada data uji untuk menghitung akurasi. Hasil dari setiap kombinasi parameter dicatat untuk analisis. Pendekatan ini memungkinkan evaluasi komprehensif terhadap dampak parameter pada kinerja model, sekaligus menyoroti potensi model bidirectional dalam menangkap konteks temporal dua arah.



```python
# Display results
for result in results:
    print(f"Hidden Size: {result[0]}, Pooling: {result[1]}, Optimizer: {result[2]}, Epochs: {result[3]}, Test Accuracy: {result[4]:.4f}")
```

Analisis:

Digunakan untuk menampilkan hasil eksperimen dengan mencetak setiap kombinasi parameter dan performa model pada data uji. Untuk setiap hasil, ukuran hidden layer (Hidden Size), metode pooling (Pooling), jenis optimizer (Optimizer), jumlah epoch pelatihan (Epochs), dan akurasi pada data uji (Test Accuracy) dicatat. Output ini memberikan wawasan tentang bagaimana setiap kombinasi parameter memengaruhi kinerja model Bidirectional RNN. Dengan mengamati hasil ini, dapat diidentifikasi konfigurasi terbaik yang menghasilkan akurasi tertinggi, serta tren bagaimana parameter seperti hidden size, pooling, optimizer, dan jumlah epoch memengaruhi akurasi model pada dataset Bank Marketing. Hasil ini penting untuk memilih parameter yang optimal untuk aplikasi serupa di masa depan.