```
# Basic setup
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU, SimpleRNN, Dropout
```

```
# Step 1: Upload the ZIP file
from google.colab import files
uploaded = files.upload()   # Choose your 'archive.zip' or similar file

# Step 2: Unzip (use the correct filename shown after upload)
!unzip -q "archive (6).zip" -d "/content/HAR_Dataset"

# If Colab renamed it to something like archive (6) (1).zip, then run:
# !unzip -q "archive (6) (1).zip" -d "/content/HAR_Dataset"

# Step 3: Check extracted folders
!ls "/content/HAR_Dataset"
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving archive (6).zip to archive (6) (1).zip
replace /content/HAR_Dataset/test.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: trupti
error:  invalid response [trupti]
replace /content/HAR_Dataset/test.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: no
replace /content/HAR_Dataset/train.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: one
error:  invalid response [one]
replace /content/HAR_Dataset/train.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: 1
error:  invalid response [1]
replace /content/HAR_Dataset/train.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: test.csv  train.c
```

```
!ls /content/HAR_Dataset
```

```
test.csv  train.csv
```

```
import pandas as pd

# Load your CSV files
train_df = pd.read_csv("/content/HAR_Dataset/train.csv")
test_df = pd.read_csv("/content/HAR_Dataset/test.csv")

print("✅ Data Loaded Successfully!")
print("Training shape:", train_df.shape)
print("Testing shape:", test_df.shape)

# Display the first few rows
train_df.head()
```

```
✅ Data Loaded Successfully!
Training shape: (7352, 563)
Testing shape: (2947, 563)
```

| | tBodyAcc-mean()-X | tBodyAcc-mean()-Y | tBodyAcc-mean()-Z | tBodyAcc-std()-X | tBodyAcc-std()-Y | tBodyAcc-std()-Z | tBodyAcc-mad()-X | tBodyAcc-mad()-Y | tBodyA mad( |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.288585 | -0.020294 | -0.132905 | -0.995279 | -0.983111 | -0.913526 | -0.995112 | -0.983185 | -0.923 |
| 1 | 0.278419 | -0.016411 | -0.123520 | -0.998245 | -0.975300 | -0.960322 | -0.998807 | -0.974914 | -0.957 |
| 2 | 0.279653 | -0.019467 | -0.113462 | -0.995380 | -0.967187 | -0.978944 | -0.996520 | -0.963668 | -0.977 |
| 3 | 0.279174 | -0.026201 | -0.123283 | -0.996091 | -0.983403 | -0.990675 | -0.997099 | -0.982750 | -0.989 |
| 4 | 0.276629 | -0.016570 | -0.115362 | -0.998139 | -0.980817 | -0.990482 | -0.998321 | -0.979672 | -0.990 |

```
X_train = train_df.drop("Activity", axis=1)
y_train = train_df["Activity"]

X_test = test_df.drop("Activity", axis=1)
y_test = test_df["Activity"]
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Encode labels (convert to one-hot)
encoder = OneHotEncoder(sparse_output=False)
y_train_encoded = encoder.fit_transform(y_train.values.reshape(-1,1))
y_test_encoded = encoder.transform(y_test.values.reshape(-1,1))
```

```
X_train_seq = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_seq = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

print("Train reshaped:", X_train_seq.shape)
print("Test reshaped:", X_test_seq.shape)
```

```
Train reshaped: (7352, 1, 562)
Test reshaped: (2947, 1, 562)
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, GRU, Dropout
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import numpy as np

# Helper function for plotting accuracy and loss
def plot_history(history, title):
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title(f'{title} Accuracy')
    plt.xlabel('Epochs')
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f'{title} Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# For comparison
results = {}

# 🌀 Simple RNN Model
rnn_model = Sequential([
    SimpleRNN(64, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2]), activation='tanh'),
    Dropout(0.3),
    Dense(y_train_encoded.shape[1], activation='softmax')
])

rnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_rnn = rnn_model.fit(X_train_seq, y_train_encoded, epochs=10, batch_size=64, validation_
results['RNN'] = rnn_model.evaluate(X_test_seq, y_test_encoded, verbose=0)

plot_history(history_rnn, "RNN")

# ⚡ LSTM Model
lstm_model = Sequential([
    LSTM(64, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])),
    Dropout(0.3),
    Dense(y_train_encoded.shape[1], activation='softmax')
])

lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_lstm = lstm_model.fit(X_train_seq, y_train_encoded, epochs=10, batch_size=64, validatio
results['LSTM'] = lstm_model.evaluate(X_test_seq, y_test_encoded, verbose=0)

plot_history(history_lstm, "LSTM")

# ⚙️ GRU Model
gru_model = Sequential([
    GRU(64, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])),
    Dropout(0.3),
    Dense(y_train_encoded.shape[1], activation='softmax')
])

gru_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history_gru = gru_model.fit(X_train_seq, y_train_encoded, epochs=10, batch_size=64, validation_
results['GRU'] = gru_model.evaluate(X_test_seq, y_test_encoded, verbose=0)

plot_history(history_gru, "GRU")

# 📊 Compare model performance
print("\n=== Model Accuracy Comparison ===")
for model, score in results.items():
    print(f"{model} → Loss: {score[0]:.4f}, Accuracy: {score[1]*100:.2f}%")

# 📉 Evaluate best model (LSTM or GRU usually)
best_model = max(results, key=lambda x: results[x][1])
print(f"\n🏆 Best Model: {best_model}")
```

```
# Confusion Matrix for best model
model_obj = {'RNN': rnn_model, 'LSTM': lstm_model, 'GRU': gru_model}[best_model]
y_pred = model_obj.predict(X_test_seq)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test_encoded, axis=1)

plt.figure(figsize=(6,5))
sns.heatmap(confusion_matrix(y_true, y_pred_classes), annot=True, fmt='d', cmap='Blues')
plt.title(f'{best_model} Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Classification report
print(f"\n📋 {best_model} Classification Report:")
print(classification_report(y_true, y_pred_classes))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pa
  super().__init__(**kwargs)
Epoch 1/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - accuracy: 0.6748 - loss: 0.7817 - val_accuracy:
Epoch 2/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9396 - loss: 0.1789 - val_accuracy:
Epoch 3/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9634 - loss: 0.1144 - val_accuracy:
Epoch 4/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9733 - loss: 0.0839 - val_accuracy:
Epoch 5/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9735 - loss: 0.0780 - val_accuracy:
Epoch 6/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9812 - loss: 0.0633 - val_accuracy:
Epoch 7/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9821 - loss: 0.0544 - val_accuracy:
Epoch 8/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9833 - loss: 0.0494 - val_accuracy:
Epoch 9/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9836 - loss: 0.0439 - val_accuracy:
Epoch 10/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9822 - loss: 0.0504 - val_accuracy:
```