```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import numpy as np
import os

# Set the dataset path
dataset_path = "/content/drive/MyDrive/UCI HAR Dataset/UCI HAR Dataset"

# Load features and activity labels
features = pd.read_csv(os.path.join(dataset_path, 'features.txt'),
                       delim_whitespace=True, header=None)
feature_names = features[1].values

activity_labels = pd.read_csv(os.path.join(dataset_path, 'activity_labels.txt'),
                              delim_whitespace=True, header=None, index_col=0)
activity_labels_dict = activity_labels[1].to_dict()

# Load training and testing data
X_train = pd.read_csv(os.path.join(dataset_path, 'train', 'X_train.txt'),
                      delim_whitespace=True, header=None)
y_train = pd.read_csv(os.path.join(dataset_path, 'train', 'y_train.txt'),
                      delim_whitespace=True, header=None)

X_test = pd.read_csv(os.path.join(dataset_path, 'test', 'X_test.txt'),
                     delim_whitespace=True, header=None)
y_test = pd.read_csv(os.path.join(dataset_path, 'test', 'y_test.txt'),
                     delim_whitespace=True, header=None)

# Add feature names for clarity (optional)
X_train.columns = feature_names
X_test.columns = feature_names

# Convert labels to one-hot encoding for classification
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train - 1)  # Labels are 1-based
y_test = to_categorical(y_test - 1)
```

```
<ipython-input-4-93679bd03e77>:9: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in
  features = pd.read_csv(os.path.join(dataset_path, 'features.txt'),
<ipython-input-4-93679bd03e77>:13: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in
  activity_labels = pd.read_csv(os.path.join(dataset_path, 'activity_labels.txt'),
<ipython-input-4-93679bd03e77>:18: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in
  X_train = pd.read_csv(os.path.join(dataset_path, 'train', 'X_train.txt'),
<ipython-input-4-93679bd03e77>:20: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in
  y_train = pd.read_csv(os.path.join(dataset_path, 'train', 'y_train.txt'),
<ipython-input-4-93679bd03e77>:23: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in
  X_test = pd.read_csv(os.path.join(dataset_path, 'test', 'X_test.txt'),
<ipython-input-4-93679bd03e77>:25: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in
  y_test = pd.read_csv(os.path.join(dataset_path, 'test', 'y_test.txt'),
```

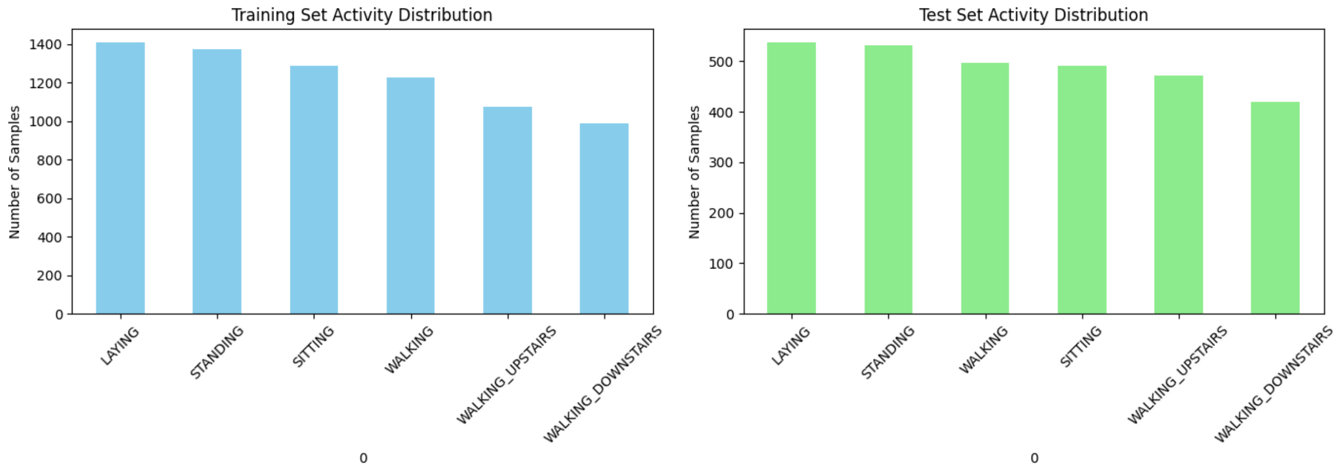
```
import os
os.listdir("/content/drive/MyDrive/UCI HAR Dataset/UCI HAR Dataset")
```

```
['features.txt',
 'features_info.txt',
 'activity_labels.txt',
 'README.txt',
 '.DS_Store',
 'test',
 'train']
```

```
import os
import pandas as pd

# ✅ Correct nested path
dataset_path = "/content/drive/MyDrive/UCI HAR Dataset/UCI HAR Dataset"

# Use sep='\s+' instead of deprecated delim_whitespace
features = pd.read_csv(os.path.join(dataset_path, 'features.txt'), sep='\s+', header=None)[1].values
activity_labels = pd.read_csv(os.path.join(dataset_path, 'activity_labels.txt'), sep='\s+', header=None, index_col=0)

# Load train data
X_train = pd.read_csv(os.path.join(dataset_path, 'train', 'X_train.txt'), sep='\s+', header=None)
```

```
y_train = pd.read_csv(os.path.join(dataset_path, 'train', 'y_train.txt'), sep='\s+', header=None)

# Load test data
X_test = pd.read_csv(os.path.join(dataset_path, 'test', 'X_test.txt'), sep='\s+', header=None)
y_test = pd.read_csv(os.path.join(dataset_path, 'test', 'y_test.txt'), sep='\s+', header=None)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import os

# Define dataset path
dataset_path = "/content/drive/MyDrive/UCI HAR Dataset/UCI HAR Dataset"

# Load activity labels
activity_labels = pd.read_csv(
    os.path.join(dataset_path, 'activity_labels.txt'),
    sep='\s+', header=None, index_col=0
).to_dict()[1]

# Load y_train and y_test
y_train = pd.read_csv(os.path.join(dataset_path, 'train', 'y_train.txt'), header=None)[0]
y_test = pd.read_csv(os.path.join(dataset_path, 'test', 'y_test.txt'), header=None)[0]

# Map labels to activity names
y_train_named = y_train.map(activity_labels)
y_test_named = y_test.map(activity_labels)

# Plotting
fig, ax = plt.subplots(1, 2, figsize=(14, 5))

# Training set distribution
y_train_named.value_counts().plot(kind='bar', ax=ax[0], color='skyblue')
ax[0].set_title('Training Set Activity Distribution')
ax[0].set_ylabel('Number of Samples')
ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=45)

# Test set distribution
y_test_named.value_counts().plot(kind='bar', ax=ax[1], color='lightgreen')
ax[1].set_title('Test Set Activity Distribution')
ax[1].set_ylabel('Number of Samples')
ax[1].set_xticklabels(ax[1].get_xticklabels(), rotation=45)

plt.tight_layout()
plt.show()
```



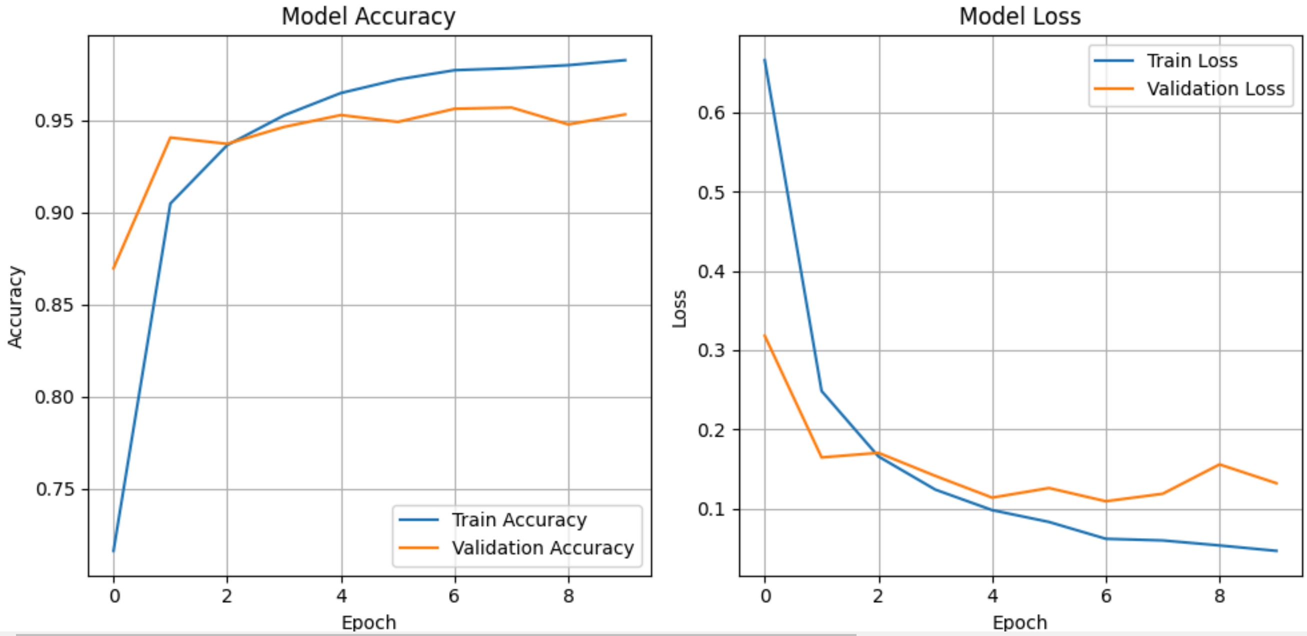

```
import numpy as np
import pandas as pd
import os
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (Input, Conv1D, MaxPooling1D, Flatten, Dense, Dropout, Add,
                                     GlobalAveragePooling1D, GlobalMaxPooling1D, Reshape, Multiply, Concatenate,
                                     Activation, Lambda)
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
import tensorflow.keras.backend as K
```

```
# Dataset path
dataset_path = "/content/drive/MyDrive/UCI HAR Dataset/UCI HAR Dataset"

# Load data
features = pd.read_csv(os.path.join(dataset_path, 'features.txt'), sep='\s+', header=None)[1].values
X_train = pd.read_csv(os.path.join(dataset_path, 'train', 'X_train.txt'), sep='\s+', header=None).values
y_train = pd.read_csv(os.path.join(dataset_path, 'train', 'y_train.txt'), sep='\s+', header=None).values
X_test = pd.read_csv(os.path.join(dataset_path, 'test', 'X_test.txt'), sep='\s+', header=None).values
y_test = pd.read_csv(os.path.join(dataset_path, 'test', 'y_test.txt'), sep='\s+', header=None).values

# Preprocessing
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

y_train = to_categorical(y_train - 1)
y_test = to_categorical(y_test - 1)

num_classes = y_train.shape[1]
input_shape = (X_train.shape[1], 1)

# -------- CBAM MODULE --------
def cbam_block(input_tensor, ratio=8):
    # Channel Attention
    channel = input_tensor.shape[-1]
    shared_dense_one = Dense(channel // ratio, activation='relu', kernel_initializer='he_normal', use_bias=True)
    shared_dense_two = Dense(channel, kernel_initializer='he_normal', use_bias=True)

    avg_pool = GlobalAveragePooling1D()(input_tensor)
    avg_pool = Reshape((1, channel))(avg_pool)
    avg_pool = shared_dense_one(avg_pool)
    avg_pool = shared_dense_two(avg_pool)

    max_pool = GlobalMaxPooling1D()(input_tensor)
    max_pool = Reshape((1, channel))(max_pool)
    max_pool = shared_dense_one(max_pool)
    max_pool = shared_dense_two(max_pool)

    cbam_feature = Add()([avg_pool, max_pool])
    cbam_feature = Activation('sigmoid')(cbam_feature)
    cbam_feature = Multiply()([input_tensor, cbam_feature])

    # Spatial Attention
    avg_pool = Lambda(lambda x: K.mean(x, axis=2, keepdims=True))(cbam_feature)
    max_pool = Lambda(lambda x: K.max(x, axis=2, keepdims=True))(cbam_feature)
    concat = Concatenate(axis=2)([avg_pool, max_pool])
    spatial_attention = Conv1D(filters=1, kernel_size=7, padding='same', activation='sigmoid')(concat)

    refined_feature = Multiply()([cbam_feature, spatial_attention])
    return refined_feature

# -------- SELECTIVE KERNEL BLOCK --------
def selective_kernel_block(input_tensor, filters):
    branch_3 = Conv1D(filters, kernel_size=3, padding='same', activation='relu')(input_tensor)
    branch_5 = Conv1D(filters, kernel_size=5, padding='same', activation='relu')(input_tensor)
    branch_7 = Conv1D(filters, kernel_size=7, padding='same', activation='relu')(input_tensor)
    branch_9 = Conv1D(filters, kernel_size=9, padding='same', activation='relu')(input_tensor)

    merged = Add()([branch_3, branch_5, branch_7, branch_9])  # Could also try Concatenate()
    return merged

# -------- BUILD ASK-HAR MODEL --------
input_layer = Input(shape=input_shape)

x = selective_kernel_block(input_layer, filters=64)
x = cbam_block(x)
x = MaxPooling1D(pool_size=2)(x)

x = selective_kernel_block(x, filters=128)
x = cbam_block(x)
x = MaxPooling1D(pool_size=2)(x)

x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
output_layer = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=input_layer, outputs=output_layer)
```

```
# Compile
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Train
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))
```

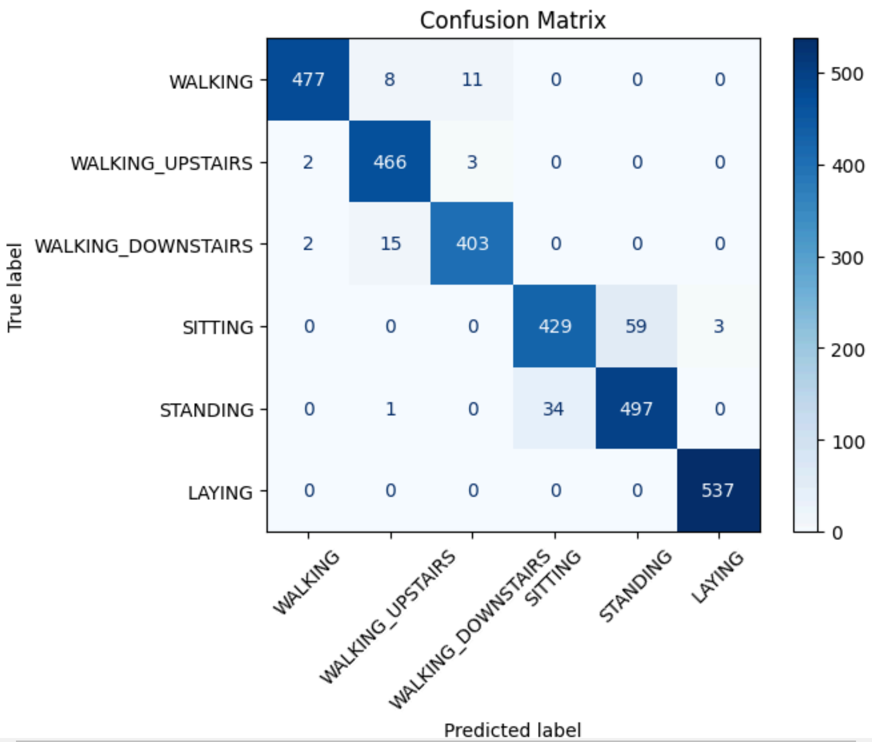
```
Epoch 1/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 118s 955ms/step - accuracy: 0.5547 - loss: 1.0187 - val_accuracy: 0.8697 - val_loss: 0.3181
Epoch 2/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 134s 891ms/step - accuracy: 0.8873 - loss: 0.2842 - val_accuracy: 0.9406 - val_loss: 0.1646
Epoch 3/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 136s 840ms/step - accuracy: 0.9379 - loss: 0.1726 - val_accuracy: 0.9372 - val_loss: 0.1704
Epoch 4/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 95s 822ms/step - accuracy: 0.9488 - loss: 0.1352 - val_accuracy: 0.9464 - val_loss: 0.1413
Epoch 5/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 137s 787ms/step - accuracy: 0.9645 - loss: 0.0944 - val_accuracy: 0.9528 - val_loss: 0.1139
Epoch 6/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 143s 799ms/step - accuracy: 0.9725 - loss: 0.0836 - val_accuracy: 0.9491 - val_loss: 0.1260
Epoch 7/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 139s 776ms/step - accuracy: 0.9790 - loss: 0.0608 - val_accuracy: 0.9562 - val_loss: 0.1094
Epoch 8/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 144s 799ms/step - accuracy: 0.9755 - loss: 0.0673 - val_accuracy: 0.9569 - val_loss: 0.1188
Epoch 9/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 141s 787ms/step - accuracy: 0.9799 - loss: 0.0493 - val_accuracy: 0.9477 - val_loss: 0.1558
Epoch 10/10
115/115 ━━━━━━━━━━━━━━━━━━━━ 146s 822ms/step - accuracy: 0.9809 - loss: 0.0509 - val_accuracy: 0.9532 - val_loss: 0.1321
```

```
import matplotlib.pyplot as plt

# Plot accuracy and loss
def plot_training_history(history):
    plt.figure(figsize=(10, 5))

    # Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Call the function
plot_training_history(history)
```




```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

# Predict
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
labels = [
    'WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS',
    'SITTING', 'STANDING', 'LAYING'
]

# Plot
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
plt.title("Confusion Matrix")
plt.grid(False)
plt.show()
```

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 15s 161ms/step
```



```
from sklearn.metrics import classification_report

# Assuming y_true and y_pred_classes are already defined
labels = [
    'WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS',
    'SITTING', 'STANDING', 'LAYING'
]

# Generate and print the classification report
report = classification_report(y_true, y_pred_classes, target_names=labels)
print("Classification Report:\n")
print(report)
```

```
Classification Report:

                    precision    recall  f1-score   support

           WALKING       0.99      0.96      0.98       496
  WALKING_UPSTAIRS       0.95      0.99      0.97       471
WALKING_DOWNSTAIRS       0.97      0.96      0.96       420
           SITTING       0.93      0.87      0.90       491
          STANDING       0.89      0.93      0.91       532
            LAYING       0.99      1.00      1.00       537

          accuracy                           0.95      2947
         macro avg       0.95      0.95      0.95      2947
      weighted avg       0.95      0.95      0.95      2947
```