ML Project - Comparison of Machine Learning Models for Workout Analysis

```
pip install pandas numpy matplotlib seaborn scikit-learn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

Loading the Dataset onto Colab

```
import pandas as pd

file_path = "RecGym.csv"
data = pd.read_csv(file_path, encoding='utf-8')

print("First 10 rows of the dataset:")
print(data.head(10).to_string())

print("\nDataset Info:")
print(data.info())

print("\nSummary Statistics:")
print(data.describe())

print("\nMissing Values:")
print(data.isnull().sum())
```

```
8       1    wrist      1.0  0.500625  0.499625  0.501250  0.499025  0.500275  0.500119  0.501085    Null
9       1    wrist      1.0  0.500000  0.498625  0.502250  0.502381  0.500231  0.499962  0.499743    Null

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31155 entries, 0 to 31154
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Subject   31155 non-null  int64  
 1   Position  31155 non-null  object 
 2   Session   31154 non-null  float64
 3   A_x       31154 non-null  float64
 4   A_y       31154 non-null  float64
 5   A_z       31154 non-null  float64
 6   G_x       31154 non-null  float64
 7   G_y       31154 non-null  float64
 8   G_z       31154 non-null  float64
 9   C_1       31154 non-null  float64
 10  Workout   31154 non-null  object 
dtypes: float64(8), int64(1), object(2)
memory usage: 2.6+ MB
None

Summary Statistics:
       Subject  Session           A_x           A_y           A_z  \
count  31155.0  31154.0  31154.000000  31154.000000  31154.000000   
mean       1.0      1.0      0.502240      0.498696      0.498309   
std        0.0      0.0      0.015989      0.019553      0.017103   
```

```
min        0.102694      0.296406      0.119219      0.000000
25%        0.496406      0.497894      0.496331      0.498615
50%        0.499962      0.500081      0.499919      0.500189
75%        0.503594      0.503162      0.502878      0.501841
max        0.858631      0.669844      0.726325      1.000000

Missing Values:
Subject     0
Position    0
Session     1
A_x         1
A_y         1
A_z         1
G_x         1
G_y         1
G_z         1
C_1         1
Workout     1
dtype: int64
```

Checking the Dataset for missing values

```
import pandas as pd

file_path = "RecGym.csv"
data = pd.read_csv(file_path, encoding='utf-8')

missing_values = data.isnull().sum()

print("Missing Values in Each Column:")
print(missing_values[missing_values > 0])
if data.isnull().values.any():
    print("\nThe dataset contains missing values.")
else:
    print("\nNo missing values found in the dataset.")
```

```
Missing Values in Each Column:
C_1        1
Workout    1
dtype: int64

The dataset contains missing values.
```

Data Preprocessing Steps

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

file_path = "RecGym.csv"
data = pd.read_csv(file_path, encoding='utf-8')
data.dropna(inplace=True)

sensor_columns = ["A_x", "A_y", "A_z", "G_x", "G_y", "G_z", "C_1"]

data[sensor_columns] = data[sensor_columns].apply(pd.to_numeric, errors='coerce')
data.dropna(inplace=True)
scaler = StandardScaler()
data[sensor_columns] = scaler.fit_transform(data[sensor_columns])

pca = PCA(n_components=5)
data_pca = pca.fit_transform(data[sensor_columns])

label_encoder = LabelEncoder()
data["Workout"] = label_encoder.fit_transform(data["Workout"])

X = pd.DataFrame(data_pca)
y = data["Workout"]

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

print("Preprocessing completed with feature reduction, augmentation, and real-time adaptation!")
print("\nFirst 10 Rows of Preprocessed Data (PCA-Reduced Features):")
print(X_train.head(10))

print("\nEncoded Workout Labels (First 10 Rows):")
print(y_train.head(10))
```

```
print("\nShape of Training and Testing Sets:")
print(f"X_train: {X_train.shape}, X_test: {X_test.shape}")
print(f"y_train: {y_train.shape}, y_test: {y_test.shape}")
```

```
Preprocessing completed with feature reduction, augmentation, and real-time adaptation!

First 10 Rows of Preprocessed Data (PCA-Reduced Features):
               0         1         2         3         4
721324  0.026924  0.750032 -0.217873  0.280247 -0.055111
264856  0.074519  0.506825  0.332641 -0.743139  2.244520
479048  0.037262  0.176077 -0.036703  0.004328 -0.042720
370383  0.149144  0.059629 -0.100245 -0.079551  0.128406
542760  0.521736  0.453749  0.081733 -0.129390 -0.151991
667097  0.715391 -0.052713 -1.664123  1.298820 -0.664162
27970   0.033977  0.184751  0.005338  0.037680 -0.040637
415685  0.120025  0.165496 -0.475122  0.353599  0.181185
439624  0.046048 -0.425226  1.070706  0.560121 -0.449708
572047  0.184411  0.426513 -0.068706  0.163932 -0.627696

Encoded Workout Labels (First 10 Rows):
721324     2
264856     5
479048     5
370383     5
542760     6
667097    11
27970      5
415685     5
439624     5
572047     5
Name: Workout, dtype: int64

Shape of Training and Testing Sets:
X_train: (655356, 5), X_test: (163839, 5)
y_train: (655356,), y_test: (163839,)
```

Reshaping data for ResCNN

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv1D, BatchNormalization, ReLU, Add, GlobalAveragePooling1D, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split


num_classes = len(np.unique(y_train))
y_train_cat = to_categorical(y_train, num_classes)
y_test_cat = to_categorical(y_test, num_classes)

time_steps = 20
num_features = X_train.shape[1]

def create_time_series_data(X, y, time_steps):
    sequences, labels = [], []
    for i in range(len(X) - time_steps):
        sequences.append(X.iloc[i : i + time_steps].values)
        labels.append(y[i + time_steps])
    return np.array(sequences), np.array(labels)

X_train_seq, y_train_seq = create_time_series_data(X_train, y_train_cat, time_steps)
X_test_seq, y_test_seq = create_time_series_data(X_test, y_test_cat, time_steps)

print(f"Reshaped data for CNN: {X_train_seq.shape}")
```

```
Reshaped data for CNN: (655336, 20, 5)
```

Building ResCNN Model

```
def build_rescnn(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = Conv1D(filters=64, kernel_size=3, padding="same")(inputs)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    def residual_block(x, filters=64):
        res = Conv1D(filters, kernel_size=3, padding="same")(x)
```

```
        res = BatchNormalization()(res)
        res = ReLU()(res)
        res = Conv1D(filters, kernel_size=3, padding="same")(res)
        res = BatchNormalization()(res)
        x = Add()([x, res])  # Residual connection
        x = ReLU()(x)
        return x

    x = residual_block(x)
    x = residual_block(x)

    x = GlobalAveragePooling1D()(x)
    outputs = Dense(num_classes, activation="softmax")(x)

    model = Model(inputs, outputs)
    return model

rescnn_model = build_rescnn(input_shape=(time_steps, num_features), num_classes=num_classes)
rescnn_model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

rescnn_model.summary()
```

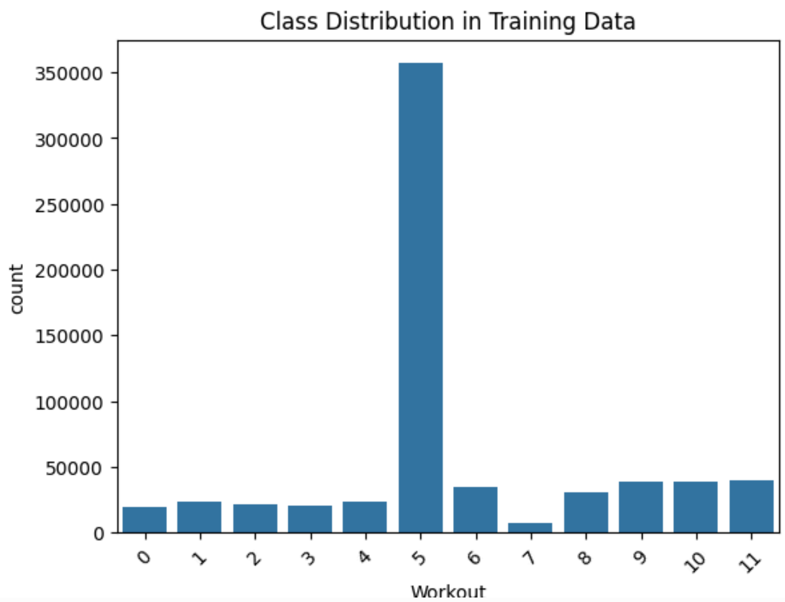
Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 20, 5)	0	-
conv1d (Conv1D)	(None, 20, 64)	1,024	input_layer[0][0]
batch_normalization (BatchNormalization)	(None, 20, 64)	256	conv1d[0][0]
re_lu (ReLU)	(None, 20, 64)	0	batch_normalization[0…
conv1d_1 (Conv1D)	(None, 20, 64)	12,352	re_lu[0][0]
batch_normalization_1 (BatchNormalization)	(None, 20, 64)	256	conv1d_1[0][0]
re_lu_1 (ReLU)	(None, 20, 64)	0	batch_normalization_1…
conv1d_2 (Conv1D)	(None, 20, 64)	12,352	re_lu_1[0][0]
batch_normalization_2 (BatchNormalization)	(None, 20, 64)	256	conv1d_2[0][0]
add (Add)	(None, 20, 64)	0	re_lu[0][0], batch_normalization_2…
re_lu_2 (ReLU)	(None, 20, 64)	0	add[0][0]
conv1d_3 (Conv1D)	(None, 20, 64)	12,352	re_lu_2[0][0]
batch_normalization_3 (BatchNormalization)	(None, 20, 64)	256	conv1d_3[0][0]
re_lu_3 (ReLU)	(None, 20, 64)	0	batch_normalization_3…
conv1d_4 (Conv1D)	(None, 20, 64)	12,352	re_lu_3[0][0]
batch_normalization_4 (BatchNormalization)	(None, 20, 64)	256	conv1d_4[0][0]
add_1 (Add)	(None, 20, 64)	0	re_lu_2[0][0], batch_normalization_4…
re_lu_4 (ReLU)	(None, 20, 64)	0	add_1[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0	re_lu_4[0][0]
dense (Dense)	(None, 12)	780	global_average_poolin…

Training the model

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x=y_train)
plt.xticks(rotation=45)
plt.title("Class Distribution in Training Data")
plt.show()
```




```
import pandas as pd
from sklearn.utils import resample

X_train_df = pd.DataFrame(X_train, columns=[f"feature_{i}" for i in range(X_train.shape[1])])
y_train_df = pd.DataFrame(y_train, columns=["Workout"])

df = pd.concat([X_train_df, y_train_df], axis=1)

df_majority = df[df["Workout"] == 5]
df_minority = df[df["Workout"] != 5]

df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=len(df_minority),
                                   random_state=42)

df_balanced = pd.concat([df_majority_downsampled, df_minority])

df_balanced = df_balanced.sample(frac=1, random_state=42)

X_train_balanced = df_balanced.drop(columns=["Workout"]).values
y_train_balanced = df_balanced["Workout"].values



import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder

gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError:
        pass

tf.keras.mixed_precision.set_global_policy('mixed_float16')

def load_and_preprocess_data(file_path):
    data = pd.read_csv(file_path)
    data.dropna(inplace=True)

    sensor_cols = ["A_x", "A_y", "A_z", "G_x", "G_y", "G_z", "C_1"]
    features = data[sensor_cols].values.astype(np.float32)

    le = LabelEncoder()
    labels = le.fit_transform(data["Workout"])

    return features, labels, len(le.classes_)

def build_fast_model(input_shape, num_classes):
    inputs = tf.keras.Input(shape=input_shape)
    x = tf.keras.layers.Conv1D(128, 3, padding='same')(inputs)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
```

```
        x = tf.keras.layers.ReLU()(x)
        x = tf.keras.layers.GlobalAveragePooling1D()(x)
        outputs = tf.keras.layers.Dense(num_classes, activation='softmax', dtype='float32')(x)
        return tf.keras.Model(inputs, outputs)

    def train_pipeline():
        X, y, num_classes = load_and_preprocess_data("RecGym.csv")

        X = X.reshape(*X.shape, 1)

        train_ds = tf.data.Dataset.from_tensor_slices((X, y))
        train_ds = train_ds.shuffle(10000).batch(4096).prefetch(2)

        model = build_fast_model(X.shape[1:], num_classes)
        model.compile(
            optimizer=tf.keras.optimizers.AdamW(learning_rate=0.001, weight_decay=0.0001),
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        history = model.fit(
            train_ds,
            epochs=200,
            verbose=1,
            callbacks=[
                tf.keras.callbacks.EarlyStopping(patience=3),
                tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=1)
            ]
        )

        return model

    if __name__ == "__main__":
        model = train_pipeline()
        model.save("gym_activity_model.keras")
```

```
Epoch 1/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 261s 225ms/step - accuracy: 0.5328 - loss: 1.7045 - learning_rate: 0.0010
Epoch 2/200
/usr/local/lib/python3.11/dist-packages/keras/src/callbacks/early_stopping.py:153: UserWarning: Early stopping conditioned on met
  current = self.get_monitor_value(logs)
/usr/local/lib/python3.11/dist-packages/keras/src/callbacks/callback_list.py:145: UserWarning: Learning rate reduction is conditi
  callback.on_epoch_end(epoch, logs)
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 256s 223ms/step - accuracy: 0.5664 - loss: 1.5336 - learning_rate: 0.0010
Epoch 3/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 254s 221ms/step - accuracy: 0.5686 - loss: 1.5130 - learning_rate: 0.0010
Epoch 4/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 257s 224ms/step - accuracy: 0.5700 - loss: 1.4977 - learning_rate: 0.0010
Epoch 5/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 257s 224ms/step - accuracy: 0.5714 - loss: 1.4851 - learning_rate: 0.0010
Epoch 6/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 265s 226ms/step - accuracy: 0.5730 - loss: 1.4723 - learning_rate: 0.0010
Epoch 7/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 263s 227ms/step - accuracy: 0.5740 - loss: 1.4630 - learning_rate: 0.0010
Epoch 8/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 264s 230ms/step - accuracy: 0.5752 - loss: 1.4547 - learning_rate: 0.0010
Epoch 9/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 260s 226ms/step - accuracy: 0.5760 - loss: 1.4477 - learning_rate: 0.0010
Epoch 10/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 261s 227ms/step - accuracy: 0.5770 - loss: 1.4416 - learning_rate: 0.0010
Epoch 11/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 260s 226ms/step - accuracy: 0.5777 - loss: 1.4356 - learning_rate: 0.0010
Epoch 12/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 261s 225ms/step - accuracy: 0.5784 - loss: 1.4306 - learning_rate: 0.0010
Epoch 13/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 266s 229ms/step - accuracy: 0.5794 - loss: 1.4245 - learning_rate: 0.0010
Epoch 14/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 319s 227ms/step - accuracy: 0.5799 - loss: 1.4197 - learning_rate: 0.0010
Epoch 15/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 261s 227ms/step - accuracy: 0.5807 - loss: 1.4152 - learning_rate: 0.0010
Epoch 16/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 263s 229ms/step - accuracy: 0.5811 - loss: 1.4121 - learning_rate: 0.0010
Epoch 17/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 316s 224ms/step - accuracy: 0.5820 - loss: 1.4085 - learning_rate: 0.0010
Epoch 18/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 274s 238ms/step - accuracy: 0.5823 - loss: 1.4058 - learning_rate: 0.0010
Epoch 19/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 268s 233ms/step - accuracy: 0.5830 - loss: 1.4012 - learning_rate: 0.0010
Epoch 20/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 320s 231ms/step - accuracy: 0.5835 - loss: 1.3977 - learning_rate: 0.0010
Epoch 21/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 323s 232ms/step - accuracy: 0.5841 - loss: 1.3946 - learning_rate: 0.0010
Epoch 22/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 318s 229ms/step - accuracy: 0.5841 - loss: 1.3924 - learning_rate: 0.0010
Epoch 23/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 325s 231ms/step - accuracy: 0.5851 - loss: 1.3898 - learning_rate: 0.0010
```

```
Epoch 24/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 320s 230ms/step - accuracy: 0.5852 - loss: 1.3874 - learning_rate: 0.0010
Epoch 25/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 263s 229ms/step - accuracy: 0.5853 - loss: 1.3854 - learning_rate: 0.0010
Epoch 26/200
1149/1149 ━━━━━━━━━━━━━━━━━━━━ 265s 230ms/step - accuracy: 0.5858 - loss: 1.3838 - learning_rate: 0.0010
```