### **Passo a Passo para Integrar MLflow no train\_model.py (Experiment Tracking)**

Conversamos sobre como o MLflow pode ser uma ferramenta poderosa para **rastrear e comparar todos os seus experimentos de treinamento de modelo**. Isso significa que, cada vez que você treinar o modelo de reconhecimento de acordes, o MLflow vai registrar automaticamente os hiperparâmetros que você usou, o desempenho do modelo (acurácia, perda) ao longo do treinamento e até mesmo salvar o próprio modelo e os pré-processadores.

Isso é fundamental para:

* **Reproduzibilidade:** Saber exatamente como um modelo específico foi treinado.
* **Comparação:** Ver qual combinação de parâmetros gerou o melhor modelo.
* **Organização:** Manter todos os modelos e seus dados associados em um só lugar.

Vamos ao passo a passo para integrar o MLflow no seu script train\_model.py.

**1. Pré-requisitos (Instalação):**

Certifique-se de que você tem as bibliotecas necessárias instaladas. Abra seu terminal (Prompt de Comando ou PowerShell no Windows, Terminal no macOS/Linux) e execute:

Bash

pip install mlflow tensorflow scikit-learn librosa

* mlflow: A biblioteca principal do MLflow.
* tensorflow: Necessário porque o MLflow tem uma integração especial (mlflow.keras.autolog()) para modelos Keras/TensorFlow.
* scikit-learn: Para LabelEncoder e StandardScaler.
* librosa: Para processamento de áudio.

**2. Modificando o train\_model.py:**

Abra o arquivo train\_model.py no seu editor de código (VS Code, Sublime Text, etc.).

**Passo 2.1: Adicione as Importações do MLflow**

No topo do seu arquivo train\_model.py, adicione as seguintes linhas junto com suas outras importações:

Python

# ... (suas outras importações existentes, ex: os, numpy, librosa, tensorflow, keras...)

# --- NOVAS IMPORTAÇÕES PARA MLflow ---

import mlflow

import mlflow.keras # Importa o módulo Keras do MLflow para auto-logging

**Passo 2.2: Defina Hiperparâmetros como Constantes (Opcional, mas Recomendado)**

Seus hiperparâmetros de treinamento (número de épocas, tamanho do batch, etc.) já estão definidos como constantes no seu script, o que é ótimo! O MLflow pode logar esses valores facilmente. Apenas verifique se eles estão claros:

Python

# ...

N\_MFCC = 40

MAX\_PAD\_LEN = 704

# Hiperparâmetros de treinamento

EPOCHS = 200

BATCH\_SIZE = 32

EARLY\_STOPPING\_PATIENCE = 20

REDUCE\_LR\_FACTOR = 0.2

REDUCE\_LR\_PATIENCE = 10

REDUCE\_LR\_MIN\_LR = 0.00001

**Passo 2.3: Envolva a Lógica de Treinamento com MLflow**

Agora, vamos modificar a função train\_model(). Você vai adicionar três coisas principais:

* **mlflow.keras.autolog():** Esta linha "liga" o registro automático para o Keras. Ela deve ser a primeira coisa dentro da função train\_model().
* **with mlflow.start\_run()::** Esta estrutura cria uma nova "run" (execução) do experimento MLflow. Tudo que estiver dentro deste bloco será registrado nessa run.
* **mlflow.log\_param():** Para registrar os parâmetros personalizados do seu experimento que não são automaticamente capturados pelo autolog (como os relacionados à extração de features e callbacks).
* **mlflow.log\_artifact():** Para salvar o LabelEncoder e o StandardScaler como artefatos da run. O modelo Keras já será salvo automaticamente pelo autolog.

Aqui está como a função train\_model() deve ficar (destaques com # --- NOVO: ...):

Python

def train\_model():

# --- NOVO: Inicia o auto-logging para Keras (DEVE SER A PRIMEIRA LINHA AQUI) ---

mlflow.keras.autolog()

# --- NOVO: Inicia uma nova run do MLflow ---

# Você pode dar um nome para a run para facilitar a identificação na interface

with mlflow.start\_run(run\_name="Treinamento\_CNN\_Acordes\_V1"): # Sugestão de nome

# --- NOVO: Logar parâmetros personalizados que o autolog não pega automaticamente ---

mlflow.log\_param("sample\_rate", SAMPLE\_RATE)

mlflow.log\_param("n\_mfcc", N\_MFCC)

mlflow.log\_param("max\_pad\_len", MAX\_PAD\_LEN)

mlflow.log\_param("early\_stopping\_patience", EARLY\_STOPPING\_PATIENCE)

mlflow.log\_param("reduce\_lr\_factor", REDUCE\_LR\_FACTOR)

mlflow.log\_param("reduce\_lr\_patience", REDUCE\_LR\_PATIENCE)

mlflow.log\_param("reduce\_lr\_min\_lr", REDUCE\_LR\_MIN\_LR)

print("Iniciando carregamento do dataset...")

X, y, labels = load\_dataset()

if len(X) == 0:

print("Nenhum dado encontrado no dataset. Verifique se a pasta 'dataset' contém subpastas com arquivos .wav de acordes.")

return None, None, None

encoder = LabelEncoder()

y\_encoded = encoder.fit\_transform(y)

y\_categorical = to\_categorical(y\_encoded)

valid\_indices = [i for i, features in enumerate(X) if features is not None]

X\_filtered = np.array([X[i] for i in valid\_indices])

y\_filtered = np.array([y\_categorical[i] for i in valid\_indices])

if len(X\_filtered) == 0:

print("Nenhum dado válido após extração de features. Verifique seus arquivos de áudio.")

return None, None, None

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_filtered, y\_filtered, test\_size=0.2, random\_state=42, stratify=np.argmax(y\_filtered, axis=1))

original\_shape = X\_train.shape

X\_train\_reshaped\_for\_scaler = X\_train.reshape(-1, X\_train.shape[-1])

X\_test\_reshaped\_for\_scaler = X\_test.reshape(-1, X\_test.shape[-1])

scaler = StandardScaler()

X\_train\_scaled\_reshaped = scaler.fit\_transform(X\_train\_reshaped\_for\_scaler)

X\_test\_scaled\_reshaped = scaler.transform(X\_test\_reshaped\_for\_scaler)

X\_train\_scaled = X\_train\_scaled\_reshaped.reshape(original\_shape)

X\_test\_scaled = X\_test\_scaled\_reshaped.reshape(X\_test.shape)

X\_train\_final = np.swapaxes(X\_train\_scaled, 1, 2)

X\_test\_final = np.swapaxes(X\_test\_scaled, 1, 2)

model = Sequential([

Conv1D(filters=64, kernel\_size=5, activation='relu', input\_shape=(X\_train\_final.shape[1], X\_train\_final.shape[2])),

MaxPooling1D(pool\_size=2),

Dropout(0.3),

Conv1D(filters=128, kernel\_size=3, activation='relu'),

MaxPooling1D(pool\_size=2),

Dropout(0.3),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.4),

Dense(len(labels), activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

print(f"Shape final de X\_train para o modelo: {X\_train\_final.shape}")

print(f"Shape de y\_train: {y\_train.shape}")

print(model.summary())

print("Iniciando treinamento do modelo de acordes (CNN)...")

early\_stopping = EarlyStopping(monitor='val\_loss', patience=EARLY\_STOPPING\_PATIENCE, restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=REDUCE\_LR\_FACTOR, patience=REDUCE\_LR\_PATIENCE, min\_lr=REDUCE\_LR\_MIN\_LR)

history = model.fit(X\_train\_final, y\_train,

epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

validation\_data=(X\_test\_final, y\_test),

callbacks=[early\_stopping, reduce\_lr],

verbose=1)

loss, accuracy = model.evaluate(X\_test\_final, y\_test, verbose=0)

print(f"Acurácia final do modelo de acordes no conjunto de teste: {accuracy:.4f}")

# --- NOVO: Salvar o encoder e o scaler como artefatos do MLflow ---

# (O modelo Keras já é salvo automaticamente pelo mlflow.keras.autolog())

os.makedirs(os.path.dirname(MODEL\_SAVE\_PATH), exist\_ok=True) # Garante que a pasta existe

joblib.dump(encoder, ENCODER\_SAVE\_PATH)

joblib.dump(scaler, SCALER\_SAVE\_PATH)

mlflow.log\_artifact(ENCODER\_SAVE\_PATH, artifact\_path="preprocessors") # Salva em uma subpasta 'preprocessors'

mlflow.log\_artifact(SCALER\_SAVE\_PATH, artifact\_path="preprocessors") # Salva em uma subpasta 'preprocessors'

print(f"Modelo de acordes (CNN), encoder e scaler salvos localmente e registrados no MLflow em: {os.path.dirname(MODEL\_SAVE\_PATH)}")

return model, encoder, scaler

**3. Executar o Treinamento com MLflow:**

**Importante:** Sempre execute o script de treinamento a partir do terminal.

1. **Abra seu terminal** (Prompt de Comando ou PowerShell no Windows, Terminal no macOS/Linux).

**Navegue até o diretório** onde o train\_model.py está salvo. Por exemplo, se ele estiver em C:\Projetos\ReconhecimentoAcordes:  
Bash  
cd C:\Projetos\ReconhecimentoAcordes

1. (Use aspas se o caminho tiver espaços)

**Execute o script:**Bash  
python train\_model.py

1. Você verá a saída normal do seu treinamento. O MLflow criará uma pasta chamada mlruns no seu diretório de projeto. É aqui que os dados de todos os seus experimentos serão armazenados.

**4. Visualizar os Experimentos com a UI do MLflow:**

Após o treinamento ser concluído:

**No mesmo terminal** (ainda no diretório do seu projeto), execute o comando:  
Bash  
mlflow ui

1. **Abra seu navegador web** e acesse o endereço que o terminal indicar (geralmente http://localhost:5000).

Lá, você verá uma interface onde poderá:

* Ver uma lista de todas as suas "runs" (execuções de treinamento).
* Clicar em cada run para ver os detalhes:
  + **Parâmetros:** Todos os hiperparâmetros que você logou, tanto os personalizados quanto os auto-logados pelo Keras.
  + **Métricas:** Gráficos interativos da perda e acurácia (treino e validação) ao longo das épocas.
  + **Artefatos:** Encontrará o modelo Keras salvo (em uma pasta model), e a pasta preprocessors com o seu label\_encoder\_chords.joblib e scaler\_chords.joblib. Você pode baixar esses arquivos diretamente da interface.
  + **Versão do Código:** O MLflow também tentará registrar a versão do seu código-fonte (se você estiver usando Git).