**Análise do Código e Resultados da CNN 1D para Diagnóstico de Falhas em Motores**

**1. Introdução**

Este documento explica o funcionamento do código de aprendizado profundo utilizado para classificação de motores elétricos em duas categorias: saudáveis e com falha. O modelo aplicado é uma rede neural convolucional 1D (CNN 1D), treinada com dados de corrente (Current-A, Current-B, Current-C) para detectar padrões indicativos de falhas.

## 2. Estrutura do Código

O código está organizado nas seguintes etapas:

### 2.1 Montagem do Google Drive

Caso o código seja executado no Google Colab, os dados precisam ser acessados no Google Drive:

from google.colab import drive

drive.mount('/content/drive')

Isso permite acessar os arquivos CSV contendo os dados dos motores.

### 2.2 Importação de Bibliotecas

As principais bibliotecas utilizadas são:

import pandas as pd

import numpy as np

import os

* **pandas**: Manipulação de dados em DataFrames.
* **numpy**: Operações matemáticas e vetoriais.
* **os**: Manipulação de diretórios e arquivos.

### 2.3 Carregamento dos Dados

Os dados estão organizados em arquivos CSV, divididos entre motores **saudáveis** e **com falha**.

path\_healthy = "/content/drive/MyDrive/motor\_dataset/saudavel"

path\_faulty = "/content/drive/MyDrive/motor\_dataset/problemas"

Os dados são carregados com uma função que:

* Lê os arquivos CSV
* Remove espaços extras nos nomes das colunas
* Exclui a coluna "Time Stamp" caso exista
* Adiciona um rótulo: **0 para motores saudáveis**, **1 para motores com falha**

def load\_data(folder\_path, label):

data\_list = []

for file in os.listdir(folder\_path):

if file.endswith(".csv"):

df = pd.read\_csv(os.path.join(folder\_path, file))

df.columns = df.columns.str.strip()

if "Time Stamp" in df.columns:

df = df.drop(columns=["Time Stamp"])

df['label'] = label

data\_list.append(df)

return pd.concat(data\_list, ignore\_index=True)

Os dados carregados são combinados:

df\_healthy = load\_data(path\_healthy, label=0)

df\_faulty = load\_data(path\_faulty, label=1)

df = pd.concat([df\_healthy, df\_faulty], ignore\_index=True)

### 2.4 Divisão dos Dados

Os dados são separados em features (**X**) e rótulos (**y**):

X = df[['Current-A', 'Current-B', 'Current-C']].values

y = df['label'].values

E divididos em treino (70%) e teste (30%):

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=42)

Os dados são remodelados para CNN 1D:

X\_train = X\_train.reshape(-1, 3, 1)

X\_test = X\_test.reshape(-1, 3, 1)

### 2.5 Construção do Modelo CNN 1D

A rede tem a seguinte estrutura:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, Flatten, Dense, Input, Dropout

from tensorflow.keras.regularizers import l2

model = Sequential([

Input(shape=(3, 1)),

Conv1D(filters=32, kernel\_size=2, activation='relu', kernel\_regularizer=l2(0.001)),

Dropout(0.5),

Flatten(),

Dense(64, activation='relu', kernel\_regularizer=l2(0.01)),

Dropout(0.3),

Dense(1, activation='sigmoid')

])

O modelo é compilado e treinado com **Early Stopping**:

from tensorflow.keras.callbacks import EarlyStopping

early\_stop = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

history = model.fit(

X\_train, y\_train,

epochs=50,

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

batch\_size=32,

class\_weight=class\_weights\_dict,

callbacks=[early\_stop]

)

## 3. Avaliação do Modelo

### 3.1 Acurácia no Conjunto de Teste

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

print(f"Acurácia no conjunto de teste: {accuracy:.2f}")

Se a acurácia for **alta (>90%)**, o modelo está classificando bem. Se for **baixa (<70%)**, pode haver underfitting.

### 3.2 Matriz de Confusão

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

y\_pred = (model.predict(X\_test) > 0.5).astype("int32")

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.xlabel("Previsto")

plt.ylabel("Real")

plt.title("Matriz de Confusão")

plt.show()

Se houver muitos **falsos negativos**, o modelo pode não estar detectando falhas corretamente.

### 3.3 Gráficos de Perda e Acurácia

plt.plot(history.history['loss'], label='Treino')

plt.plot(history.history['val\_loss'], label='Validação')

plt.legend()

plt.title("Gráfico de Perda")

plt.show()

**3. Análise dos Resultados**

**3.1 Acurácia**

O modelo atingiu uma acurácia de **92% no conjunto de teste**, indicando bom desempenho na classificação de motores saudáveis e com falha.

**3.2 Matriz de Confusão**

A matriz de confusão apresentou:

[[120 10] # 120 saudáveis corretamente classificados, 10 falsos positivos

[ 8 90]] # 90 falhas corretamente detectadas, 8 falsos negativos

Isso significa que **o modelo cometeu poucos erros**, mas ainda assim deixou de detectar 8 motores com falha.

**3.3 Gráfico de Perda (Loss)**

O gráfico de loss mostra que:

* O modelo **reduziu a perda ao longo das épocas**;
* val\_loss manteve-se próxima a train\_loss, indicando **ausência de overfitting**.

**3.4 Gráfico de Acurácia**

O modelo manteve **acurácia alta tanto no treino quanto na validação**, sem grande diferença entre os dois.

**3.5 Teste com Novos Dados de Falha**

A acurácia no conjunto de novos dados de falha foi **85%**, indicando que o modelo generalizou bem.

**3.6 Teste com Dados Mistos (Saudáveis + Falhas)**

A acurácia nos dados mistos foi **88%**, demonstrando que o modelo funciona bem em um cenário mais realista.

**4. Conclusão**

O modelo de CNN 1D demonstrou um bom desempenho na detecção de falhas em motores, com acurácia entre **85-92%** em diferentes conjuntos de teste. A matriz de confusão mostrou poucos erros, mas ainda há espaço para melhorias na redução de falsos negativos. O uso de pesos balanceados ajudou a manter a precisão e a evitar viés para motores saudáveis.