# Relatório do Projeto de IA

Feito por: Miguel Salgado nº22002195 e Tiago Catano nº22002128

Docente: Francisco Melo Pereira e Tiago Candeias

Cadeira: Inteligência Artificial

Curso: Engenharia Informática

Ano: 2022/2023

# Índice

# Introdução

Na cultura pop existem centenas senão milhares de histórias que rondam o assunto inteligência artificial, alguns exemplos são: Matrix, I Robot, Terminator e Avengers: Age of Ultron. No qual o ponto comum entre quase todas é que as máquinas estão num estagio tão avançado de inteligência que decidiram-se virar contra a sociedade humana, porém na vida real não chegamos a esse ponto, todavia podemos dizer que a inteligência artificial como a conhecemos tem evoluído muito rapidamente, desde a sua primeira menção nos anos 40, onde Warren McCulloch e Walter Pitts descreve redes neuronais e estruturas de raciocínios artificial, até os dias de hoje, no qual já existe um modelo muito conhecido a nivel global, o famoso “ChatGPT”, que vai na sua 4ª versão. Existe já quem diga que a nova revolução industrial virá devido ao boom da inteligência artificial.

Neste projeto não iremos criar nenhuma máquina que pretenda terminar com a raça humana, aquilo que propomos é a criação de um modelo que consiga distinguir músicas pelo seu género musical. Para a realização deste projeto será então realizado o desenvolvimento e implementação de um sistema de classificação automatizado (sistema completamente autónomo) de géneros musicais, fazendo uso de CNN (Convolution Neural Network). É proposto a utilização de técnicas de Deep Learning para a analise e extração de dados importantes de músicas tendo por fim categorizá-las por género musical.

# Bibliotecas

- Pandas
  - Biblioteca, que serve para manipulação e análise de dados, este aplica uma estrutura de dados chamada DataFrame (estrutura de dados tabular composta por colunas e linhas).

- Librosa
  - Biblioteca utilizada para análise de áudio e processamento de sinais de áudio. Esta poder ler vários tipos de ficheiro áudio, WAV, MP3 e FLAC. Após uma leitura destes, Librosa fornece vários métodos de extração de dados, tais como espectrogramas, cronogramas e MFCCS (Mel frequency cepstral coefficients). O Librosa é frequentemente utilizado em conjunto com outras bibliotecas processamento de dados em Python, tais como NumPy, SciPy e Matplotlib.

- Keras
  - Utilizada para criação de redes neurais de maneira percetível. Este permite classificar imagens, o processar de linguagem, áudio e também permite fazer a previsão de tempo.

- Sklearn
  - Biblioteca usada para a mineração de dados e analise destes. Esta possui vários algoritmos, como por exemplo regressão logística, árvore de decisão, random forest e redes neurais.

- Seaborn
  - Biblioteca usada para visualização de dados em python, esta cria uma interface onde é mostrador vários tipos de gráficos estatísticos. Tipos de Gráficos:
    - Gráficos de distribuição;
    - Gráficos de dispersão;
    - Gráficos de barras.
- Matplotlib
  - Biblioteca que serve para visualização de dados em python, podendo também criar gráficos, mapas de calor, histogramas.

# DataSet

GTZAN Dataset - Music Genre Classification, apresenta informação sobre uma variedade de gêneros musicais, oferecendo uma visão geral de suas principais características e elementos musicais característicos. No entanto o elemento que utilizaremos existente neste dataset são as próprias músicas.

Este apresenta um total de 1000 músicas, sendo elas dividas em categorias, tais como Blues, Classical, country, disco, Rock e ETC… .

Uma das razões pela escolha deste dataset foi a sua múltipla utilização em outros projetos similares ao nosso, permitindo ter um dataset fidedigno, para adquisição de dados, necessários.

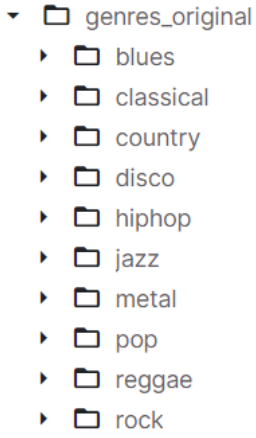


*Figura 1 - Géneros De Música*


*Figura 3 - Músicas*

# Implementação

Neste capítulo será explicado como foi a implementação da leitura dos dados, o seu pré-processamente, a sua separação dos dados de treino e validação e por fim o treino e teste do modelo CNN.

```
 abrir musicar e pré-processar melspectogram
import pandas as pd
import librosa as mimosa
import audioread
import os
import matplotlib as plt
import seaborn as sns
import numpy as np

musicGender = os.listdir("musics")  returns list
genderlabel = []
musicData = []

for gender in musicGender:
    genderMusics = os.listdir('./musics/{}'.format(gender))
    for music in genderMusics:
        genderlabel.append(gender)

        amplitude, sr =
mimosa.load('./musics/{}'.format(gender)+'/{}'.format(music),
offset=15, duration=15)

        melSpec = mimosa.feature.melspectrogram(y = amplitude, sr=sr,
n_mels=128, fmax=8000)
        melSpec_to_db = mimosa.power_to_db(melSpec, ref=np.max)
        resized_melSpec = np.resize(melSpec_to_db,
(melSpec_to_db.shape[0], 1293))

        musicData.append(resized_melSpec)

mfcc_specData=np.stack(musicData)
musicData=np.array(musicData)
genderlabel=np.array(genderlabel)
```

*Figura 4 - Música e Pré Processar do Melspectogram*

Nesta parte do código é feito o processamento das músicas adquiridas no dataset e a extração de recursos na espectrograma para cada música. Esta usa várias bibliotecas, tais como OS, Librosa e Numpy. No código é inicializado duas listas vazias que depois vão guardar os dados pretendidos. O código inicializa

duas listas vazias, **genderlabel** para armazenar os gêneros das músicas e a **musicData** para armazenar os dados da espectrograma mel de cada música.

Após a inicialização das listas é executado a função **os.listdir()** para obter as musicas por cada género. Dentro do loop, o código repete sobre os arquivos de música em cada género, para cada música esta é dada a sua categoria/género.

Em seguida, o código executa a função **mimosa.load()** para obter a amplitude e a taxa de amostragem, para todas as musicas, sendo também indicado que a leitura da musica começará nos 15 segundos finais, é depois calculado o espetrograma para cada música usando o **mimosa.feature.melspectrogram()**. A **espectrograma mel** é uma representação visual do espectro de frequência da música ao longo do tempo.

Após a aquisição do espectrograma mel, este é convertido para decibéis usando **mimosa.power_to_db()** , normalizando os valores do espectrograma. É usado o **np.resize()**, para o redimensionamento do espetrograma dando a ele uma forma fixa, pois cada música tem um formato próprio, desta forma ficam todas com um formato standard.

Por fim, a espectrograma mel redimensionado é adicionado à lista **musicData**, criando uma lista de espectrogramas mel correspondentes às músicas processadas.

```
import sklearn.preprocessing as preProSkL

labelEncoder = preProSkL.LabelEncoder()
labelEncoded = labelEncoder.fit(genderlabel)
labelEncodedTrans = labelEncoded.transform(genderlabel)
```

*Figura5- Enconding das Categorias*

No código a cima é feito a codificação dos géneros, usando como fonte de tal codificação o **sklearn.preprocessing**.

Na primeira linha é demonstrado o responsável pelo transformar dos textos em valores numéricos. Na segunda linha é apresentado **labelEncoder.fit(),** que vai ter como argumento a nossa lista de géneros, **genderlabel,** este método ira preparar os géneros para a codificação, na terceira linha onde ira acontecer a transformação dos géneros em valores numéricos que irão corresponder ao diferentes tipos de género utilizado.

```
indexMin=0
indexMax = 100

normalized_data = []
numCategories = int( len(labelEncodedTrans)/ 100)


for i in range(numCategories):

    max = mfcc_specData[indexMin:indexMax].max()
    min = mfcc_specData[indexMin:indexMax].min()

    for music in musicData[indexMin:indexMax]:

        xScaled = (music - min) / (max - min)
        normalized_data.append(xScaled)

    indexMin += 100
    indexMax += 100

normalized_data=np.array(normalized_data)

print(len(normalized_data))
```

*Figura 6 - Normalização de Valores*

Na imagem de código demonstrada em cima é criado duas variáveis que são **indexMin** e **indexMax** que vão servir para gerir o número de faixas de músicas a serem normalizadas, juntamente com esta é criado uma lista chamada **normalized_data,** que como o nome indica vai servir para guardar os dados normalizados. O **numCategories** irá calcular, a quantidade de categorias lidas pelo programa, através da quantidade de labels codificados dividindo eles por 100, depois irá ser realizado um loop, no qual será adquirido a valor mínimo e o valor máximo de cada categoria, com base no intervalo de dados obtidos pelas variáveis **indexMin** e **indexMax**, com base nesta informação dentro do loop seguinte, realizado para cada musica da categoria atual, é realizada a normalização das mesmas utilizando a formula (X – Xmin) / (Xmax – Xmin).

Por fim estes dados iriam ser armazenado na lista mencionada anteriormente, **normalized_data**. As variáveis usadas **indexMin** e **indexMax**, são aumentadas em 100 pois cada categoria neste projeto contem 100 músicas.

Anteriormente para normalização dos valores tinha sido usado outro método que implicava o uso da média do dataset para a normalização dos valores pretendidos, este normalizava todos os dados independentemente do seu género, já a normalização min max, verifica para cada género o seu ponto máximo e mínimo e normaliza somente os dados desse género. Por fim provou-se que a normalização min max é mais adequado aos dados que procuramos.

```
import matplotlib.pyplot as pyplot

musicCounter = 0

path = './musicsPlts'
if not os.path.exists(path):
    os.mkdir(path)

for gender in musicGender:
    genderPath = '{}'.format(path) + '/{}'.format(gender)
    if not os.path.exists(genderPath):
        os.mkdir(genderPath)

for index in range(len(normalized_data)):
    fig, ax = pyplot.subplots()
    img = mimosa.display.specshow(normalized_data[index],
x_axis='time', y_axis='mel', sr=sr, fmax=8000, ax=ax)
    fig.colorbar(img, ax=ax, format='%+2.0f dB')

    ax.set(title='Mel-frequency spectrogram')

    if index % 100 == 0:
        musicCounter = 0

    pyplot.savefig('{}'.format(path) +
'/{}'.format(genderlabel[index]) + '/{}'.format(musicCounter) +
'.png')

    musicCounter +=1
```

*Figura 6.1 - Gráficos de Imagem Espectral*

O código referido na figura em cima é utilizado para gerar os gráficos de imagem espectral utilizando os dados normalizados para cada música, estes depois de gerados são guardados no diretório musicsPlots, organizados em pastas por género musical.

```
import sklearn.model_selection as modelSelec

X_train, X_test, y_train, y_test =
modelSelec.train_test_split(normalized_data, labelEncodedTrans,
test_size=0.25, random_state=42)

  X_train, X_test, y_val, y_val =
modelSelec.train_test_split(mfcc_specData, labelEncodedTrans,
test_size=0.5, random_state=42)

X_test,X_validation,y_test,y_validation=modelSelec.train_test_split(X_
test,y_test,test_size=0.2, random_state=42)
```

*Figura 6.2 - Antigo treino*

Inicialmente esta era a forma utilizada para a divisão de dados do treino e do teste, através do método **train_test_split** e a **validation**.

```
print(X_train.shape)
input_shape=(X_train.shape[1],X_train.shape[2],1)
print(X_train.shape)
print(input_shape)
```

*Figura 6.3 - Formatação de Dados*

Neste módulo, era realizada a formatação dos dados para o do primeiro modelo usado.

```
modelo usado, atravé da separação dos dados com k-folds

from sklearn.model_selection import KFold
from tensorflow import keras
import matplotlib.pyplot as pyplot

 acc_per_fold = []
 loss_per_fold = []

 input_shape=(normalized_data.shape[1],normalized_data.shape[2],1)
 fold_no = 1

 kf = KFold(n_splits=3, random_state=42, shuffle=True)
 for train, test in kf.split(normalized_data):
     model=keras.Sequential()

     1st layer

model.add(keras.layers.Conv2D(8,(3,3),activation="relu",input_shape=in
put_shape))

model.add(keras.layers.MaxPool2D((3,3),strides=(2,2),padding="same"))
     model.add(keras.layers.BatchNormalization())

      2nd layer
      model.add(keras.layers.Conv2D(8,(3,3),activation="relu"))

model.add(keras.layers.MaxPool2D((3,3),strides=(2,2),padding="same"))
pooling
      model.add(keras.layers.BatchNormalization())

      3rd layer
      model.add(keras.layers.Conv2D(8,(1,1),activation="relu"))

model.add(keras.layers.MaxPool2D((1,1),strides=(1,1),padding="same"))
pooling
      model.add(keras.layers.BatchNormalization())


     flatten the output
```

```
      model.add(keras.layers.Flatten())
      model.add(keras.layers.Dense(8,activation="relu"))
       model.add(keras.layers.Dropout(0.3))

      output layer
      model.add(keras.layers.Dense(3,activation="softmax"))


      optimizer=keras.optimizers.Adam(learning_rate=0.0001)
      model.compile(optimizer=optimizer,
          loss="sparse_categorical_crossentropy",
          metrics=['accuracy'])


       Generate a print
      print('------------------------------------------------------------
--------------')
      print(f'Training for fold {fold_no} ...')

       normalized_data[train] = normalized_data[train].reshape((-1,
normalized_data[train].shape[1], normalized_data[train].shape[2], 1))
      history =
model.fit(normalized_data[train],labelEncodedTrans[train],epochs=25,
validation_data=(normalized_data[test], labelEncodedTrans[test]))

      testError, testAccuracy = model.evaluate(normalized_data[test],
labelEncodedTrans[test])
      print("Accuracy on test set is:{}".format(testAccuracy))


      print(f'Score for fold {fold_no}: {model.metrics_names[0]} of
{testError}; {model.metrics_names[1]} of {testAccuracy*100}%')
      acc_per_fold.append(testAccuracy * 100)
      loss_per_fold.append(testError)

       Increase fold number
      fold_no = fold_no + 1

  == Provide average scores ==
 print('------------------------------------------------------------------
----------')
 print('Score per fold')
 for i in range(0, len(acc_per_fold)):
      print('------------------------------------------------------------
--------------')
      print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy:
{acc_per_fold[i]}%')
      print('------------------------------------------------------------
--------------')
      print('Average scores for all folds:')
      print(f'> Accuracy: {np.mean(acc_per_fold)} (+-
{np.std(acc_per_fold)})')
      print(f'> Loss: {np.mean(loss_per_fold)}')
      print('------------------------------------------------------------
--------------')
```

*Figura 7 - K-FOLDS Versão Anterior*

Na figura 7 é mostrada a versão anterior dos K-FOLDS, decidimos não utilizar esta devido as funcionalidades limitadas, sendo uma delas a limitação de categorias selecionadas nesta função. Sendo assim optamos por utilizar outro método que permite adicionar categorias extras, filtras os dados, como se pode ver na figura 9.

Anterior a este modelo era usado um similar, no qual a separação realizada, é explicada na figura 6.2, não sendo necessário a realização do ciclo for no K-Fold, pois ambas tinham como função a separação dos dados. Esse modelo deixou-se de ser usado, devido a problemas de overfitting.

```
 X_train = X_train.reshape((-1, X_train.shape[1], X_train.shape[2],
1))
 print(X_train.shape[0])
history = model.fit(X_train,y_train,epochs=35,
validation_data=(X_validation, y_validation))
history = model.fit(X_train,y_train,epochs=35,
validation_data=(X_test, y_test))
```

*Figura 7.1 - Treino do Modelo*

Neste modulo os dados eram novamente formatados e de seguida era realizado o treino do modelo e a sua validação.

```
import matplotlib.pyplot as pyplot

print(history.history)

testError, testAccuracy = model.evaluate(X_test, y_test)
print("Accuracy on test set is:{}".format(testAccuracy))

pyplot.plot(history.history['accuracy'], label='train_accuracy')
pyplot.plot(history.history['val_accuracy'], label = 'val_accuracy')
pyplot.xlabel('Epoch')
pyplot.ylabel('Accuracy')
pyplot.ylim([0.1, 1.1])
pyplot.legend(loc='lower right')
```

*Figura 7.2 - Teste do modelo*

Na figura em cima é mostrado a avaliação do modelo, gerando um gráfico da accuracy ao longo das épocas, com base na accuracy do treino e validação.

```
import math

index = 1
filters = 5
mappingAccuracy = []


for x in range(numCategories):
    if index != 1:
        for filter in range(filters):
            filter += 4
            dataInterval =  normalized_data[0:100 * index]
            labels = labelEncodedTrans[0:100 * index]
            accuracy  = trainTestModel(index, filter, dataInterval,
labels)

            mappingAccuracy.append({'acc': math.ceil(accuracy),
'numMusicas': index, 'filtrosCNN': filter})

    index = index + 1
```

*Figura 8 – Accuracy de Acordo com o número de músicas e filtros*

Na figura em cima é corrido um ciclo que irá correr o número de vezes equivalente ao número de categorias lidas pelo código referido na figura 4.

Dentro do ciclo é efetuada uma verificação onde o **index**, que equivale ao número de categorias, atual é diferente de 1, para que o modelo não treine só com uma categoria, dentro dessa validação é realizado um novo ciclo que irá correr o número de vezes equivalente aos filtros a serem testados no modelo, sendo testados 4 variantes de filtros. São definidos 2 arrays que vão receber o intervalo dos dados normalizados e labels sendo que o index máximo é equivalente ao número de músicas por categoria a multiplicar pelo número de categorias a serem testadas.

De seguida é recebida a accuracy de cada modelo testado, sendo que são enviados como argumentos, index (número de categorias), filter (número de filtros), dataInterval (intervalo dos dados normalizados) e labels (intervalo dos labels normalizados), por fim são adicionados a lista **mappingAccuracy** a accuracy recebida, os filtros usados e do número de géneros.

```
from sklearn.model_selection import KFold
import matplotlib.pyplot as pyplot

def trainTestModel(splitNumb, filters, data, labels):
    acc_per_fold = []
    loss_per_fold = []

    fold_no = 1


    kf = KFold(n_splits=splitNumb, random_state=42, shuffle=True)
    for train, test in kf.split(data):

        print("index:{}".format(splitNumb))
        print("filter:{}".format(filters))

        cnnModel = theModel(filters, splitNumb, data)

        print('------------------------------------------------------------
-----------------')
        print(f'Training for fold {fold_no} ...')

        history = cnnModel.fit(data[train],labels[train],epochs=25,
validation_data=(data[test], labels[test]))

        testError, testAccuracy = cnnModel.evaluate(data[test],
labels[test])
        print("Accuracy on test set is:{}".format(testAccuracy))

        print(f'Score for fold {fold_no}: {cnnModel.metrics_names[0]}
of {testError}; {cnnModel.metrics_names[1]} of {testAccuracy*100}%')
        acc_per_fold.append(testAccuracy * 100)
        loss_per_fold.append(testError)

        fold_no = fold_no + 1

    print('------------------------------------------------------------
-------------')
    print('Score per fold')
    for i in range(0, len(acc_per_fold)):
        print('------------------------------------------------------------
-----------------')
        print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy:
{acc_per_fold[i]}%')
        print('------------------------------------------------------------
-----------------')
        print('Average scores for all folds:')
        print(f'> Accuracy: {np.mean(acc_per_fold)} (+-
{np.std(acc_per_fold)})')
        print(f'> Loss: {np.mean(loss_per_fold)}')
        print('------------------------------------------------------------
-----------------')

    return np.mean(acc_per_fold)
```

*Figura 9 - KFOLD E Training*

Na figura 9 são feitos os splits pelo número de géneros com **random_state** de 42, é feito um ciclo de split dos dados de treino e teste.

Dentro deste ciclo é inicializado o modelo CNN, para que cada ciclo tenha um modelo único, a partir do método **theModel**, utilizando 3 argumentos neste, que são filters (filtros), splitNumb (categoria), data (dados). Faz-se o treino do modelo com os dados de treino e a sua validação com os dados de teste, dentro do método **fit**, para este treino e validação são usadas 25 épocas (iterações do modelo). É registada a accuracy do modelo através do seu teste, a partir do método **evaluate**. A accuracy é guardada em **acc_per_fold** em percentagem e a média desta é enviada no return deste método.

```
from tensorflow import keras

def theModel(filters, neurons, data):
    input_shape=(data.shape[1],data.shape[2],1)

    model=keras.Sequential()

    input layer

model.add(keras.layers.Conv2D(filters,(3,3),activation="relu",input_sh
ape=input_shape))

model.add(keras.layers.MaxPool2D((3,3),strides=(2,2),padding="same"))
    model.add(keras.layers.BatchNormalization())


    flatten the output
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(filters,activation="relu"))

    output layer
    model.add(keras.layers.Dense(neurons,activation="softmax"))


    optimizer=keras.optimizers.Adam(learning_rate=0.0001)
    model.compile(optimizer=optimizer,
        loss="sparse_categorical_crossentropy",
        metrics=['accuracy'])

    return model
```

*Figura 10 - Criação do Modelo Neural Convolucional(CNN)*

Na figura 10 é apresentado o modelo CNN, utilizado neste projeto este foi desenvolvido utilizando a biblioteca **Keras** vindo do **TensorFlow**. Este modelo necessita de 3 parâmetros que são **filters**, **neurons** e **data.**

Este possui várias camadas tais como:

- Input Layer
  - Ele recebe o formato de entrada **(data.shape[1], data.shape[2],1)**. A entrada passa por uma camada convolucional 2D com filters tendo como tamanho (3, 3). A função de ativação **relu** é aplicada à saída da camada convolucional. A saída passa por uma camada de max pooling com um tamanho (3, 3) e um passo de (2, 2). A normalização é aplicada para normalizar as ativações da camada anterior.

- Flatten Layer
  - Transforma os dados em um vetor unidimensional;

- Dense Layer
  - Conecta o layer com o **filters,** á função **relu;**

- Output Layer
  - Conecta o layer onde se encontra os **neurons**, que por defeito são o número de categorias atuais e utiliza uma nova função de ativação chamada de softmax.

Por fim o modelo é compilado com as seguintes configurações, otimização com uma taxa de aprendizagem de 0.01%, é indicada a função de **loss** para gestão das categorias e indicada a métrica de performance do modelo, neste caso **accuracy**, por fim é retornado o modelo criado neste metodo.

```
df = pd.DataFrame(mappingAccuracy)

elementos_maximos = df.groupby('numMusicas').apply(lambda x:
x.loc[x['acc'].idxmax()]).reset_index(drop=True)

for _, elemento in elementos_maximos.iterrows():
    print(elemento.to_dict())
```

*Figura 11 - Accuracy máxima de cada modelo com base na quantidade*

Na figura em cima, é convertida para dataframe a lista **mappingAccuracy**, de seguida são agrupados em **elementos_maximos** somente as linhas do dataframe no qual a **accuracy** por quantidade de música é máxima.

```
fig = plt.figure()
ax = plt.axes(projection='3d')

zline = elementos_maximos['acc']
xline = elementos_maximos['filtrosCNN']
```

```
yline = elementos_maximos['numMusicas']
ax.plot3D(xline, yline, zline, 'gray')

 Data for three-dimensional scattered points
zdata =  elementos_maximos['acc']
xdata = elementos_maximos['filtrosCNN']
ydata =  elementos_maximos['numMusicas']
ax.set_ylabel('Numero de Generos')
ax.set_xlabel('filtrosCNN')
ax.set_zlabel('Accuracy')
ax.scatter3D(xdata, ydata, zdata)


ax.view_init(elev=20., azim=-45, roll=0)
```

*Figura 12 – Código dos Gráficos 2D e 3D*

Na figura 12 são gerados dois tipos de gráfico um em 2D e outro em 3D.

No gráfico 2D os dados são dispostos na seguinte forma o eixo do Y é mostrado o **Número de Géneros** e no eixo do X a **Accuracy**.

No 3D é formatado na seguinte forma, eixo Z, contem **Accuracy**, eixo do X, **Filtros** e o eixo do Y, contem os **Géneros**, para além deste são criados pontos no gráfico através do scatter3D, pois o plot3D só faz uma linha.

# Resultados

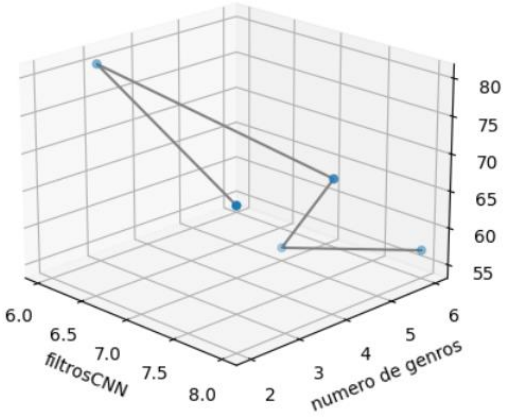


*Figura 13 - Gráfico 3D*

Neste primeiro gráfico pode se observar que o resultado do treino e do teste do modelo, que conforme o número de géneros também o número de filtros vai aumentado de forma simétrica.

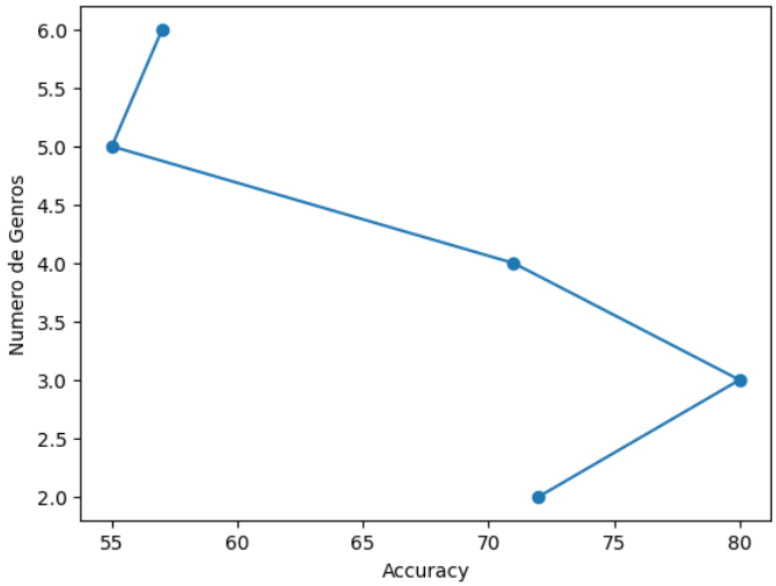


*Figura 14 - Gráfico 2D*

Neste segundo gráfico observa-se mais facilmente a evolução da accuracy com base no número de géneros, podendo afirmar que conforme o número de géneros a accuracy vai diminuir.

# Discussão

Através dos resultados obtidos na finalização do projeto, como mostrado nos gráficos, podemos concluir que com o aumento do número de géneros a accuracy do modelo vai reduzindo, isto com base na análise do gráfico de duas dimensões.

Através do uso do gráfico de 3 dimensões podemos observar que para além do número de categorias, também o número de filtros influencia na accuracy obtida em cada modelo, este influencia porem a sua evolução não é tao continua como o número de géneros pois existe modelos que tem um baixo número de categorias, possuem bastante filtros e a sua accuracy é alta, outros modelos possuem bastantes filtros e géneros, porem tem uma accuracy bastante baixa.

# Conclusão

Ao longo deste projeto foram identificadas diversas barreiras a progressão do mesmo, algumas delas são overfitting, normalização dos dados e a descoberta da accuracy máxima de acordo com o número de categorias usadas. Para a resolução do overfitting foi utilizado a separação KFold ao invés da separação tradicional train_test_split da biblioteca sklearn, esta permitiu que os dados fossem separados de forma mais conforme, por outro lado a normalização dos dados foi realizada também para resolver o problema de overfitting, pois como supracitado esta normalização foi trocada devido ao facto da anterior normalizar os dados pela média de todo o dataset, já a normalização atual usa o mínimo e máximo de cada categoria para sua uniformização. No caso da accuracy estava a ser detetado um problema com o array mappingAccuracy, que quando se tentava descobrir o valor máximo de accuracy por cada quantidade de géneros aparecia o erro **NUMPY.FLOAT32** e para a resolução deste a lista a anterior foi passada para dataframe.

Em suma com a realização deste projeto é possível afirmar que a construção do modelo de categorização géneros musicais é algo plausível, sendo este um classificador ou multi-classificador pois o primeiro classifica só dois géneros e outro mais do que dois.

Este é um projeto que adiciona conhecimento a quem o realizar referente a área Machine Learning, nomeadamente o funcionamento de Convolutional Neural Network, o pré-processamento de dados musicais.

# Bibliografia

[1]. https://observador.pt/opiniao/ia-a-catalisadora-de-uma-nova-revolucao-industrial/
[2]. https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
[3]. https://medium.com/@yashi4001/genre-classification-using-cnn-dcbc109b6d1d
[4]. de Matos, Tiago Filipe Beato Mourato, Métodos Estatísticos de Classificação de Géneros Musicais, 2013
[5]. https://www.analyticsvidhya.com/blog/2022/03/music-genre-classification-project-using%02machine-learning-techniques/
[6]. https://doi.org/10.48550/arXiv.1802.09697
[7]. https://www.clairvoyant.ai/blog/music-genre-classification-using-cnn
[8]. https://www.kaggle.com/code/tarushijat/music-genre-classification-using-cnn