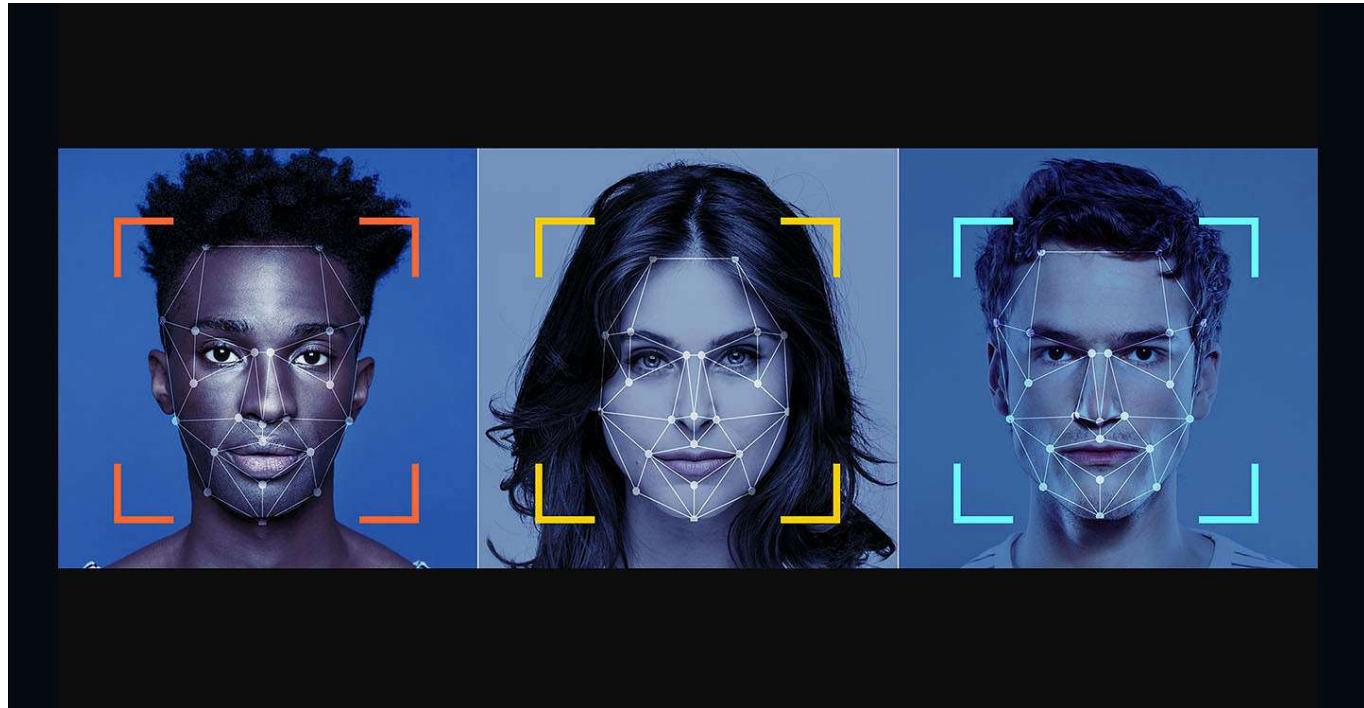


PRODUCTION ENGINEER | DATA SCIENTIST | MACHINE LEARNING ENGINEER

Rodrigo Viannini

Date: 2023-08-07

<https://www.linkedin.com/in/rodrigo-viannini-datascientist/>



Conectando ao Google Drive para obter os arquivos

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

Mounted at /content/drive

Passo 1: Importar as bibliotecas necessárias

```
1 import torch  
2 import torch.nn as nn  
3 import torch.optim as optim  
4 import torch.nn.functional as F  
5
```

Passo 2: Preparar os dados e iniciar o treinamento

Descompactar diretório

```
1 !unzip /content/drive/MyDrive/FACEMATCH/MLChallenge_Dataset.zip
```

```
A saída de streaming foi truncada nas últimas 5000 linhas.  
inflating: MLChallenge_Dataset/Data/6936/spoof/245257.jpg  
inflating: MLChallenge_Dataset/Data/6936/spoof/484566.jpg  
creating: MLChallenge_Dataset/Data/6946/  
creating: MLChallenge_Dataset/Data/6946/live/  
inflating: MLChallenge_Dataset/Data/6946/live/268809.jpg  
inflating: MLChallenge_Dataset/Data/6946/live/336113.jpg  
creating: MLChallenge_Dataset/Data/6946/spoof/  
inflating: MLChallenge_Dataset/Data/6946/spoof/103614.jpg  
inflating: MLChallenge_Dataset/Data/6946/spoof/208423.jpg  
inflating: MLChallenge_Dataset/Data/6946/spoof/374560.jpg  
creating: MLChallenge_Dataset/Data/6951/  
creating: MLChallenge_Dataset/Data/6951/live/  
creating: MLChallenge_Dataset/Data/6951/spoof/  
inflating: MLChallenge_Dataset/Data/6951/spoof/271369.jpg  
inflating: MLChallenge_Dataset/Data/6951/spoof/439942.jpg  
creating: MLChallenge_Dataset/Data/6953/  
creating: MLChallenge_Dataset/Data/6953/live/
```

```

inflating: MLChallenge_Dataset/Data/6953/live/108388.jpg
creating: MLChallenge_Dataset/Data/6953/spoof/
inflating: MLChallenge_Dataset/Data/6953/spoof/010375.jpg
inflating: MLChallenge_Dataset/Data/6953/spoof/016523.jpg
inflating: MLChallenge_Dataset/Data/6953/spoof/351071.jpg
creating: MLChallenge_Dataset/Data/6980/
creating: MLChallenge_Dataset/Data/6980/live/
creating: MLChallenge_Dataset/Data/6980/spoof/
creating: MLChallenge_Dataset/Data/6994/
creating: MLChallenge_Dataset/Data/6994/live/
inflating: MLChallenge_Dataset/Data/6994/live/120610.jpg
inflating: MLChallenge_Dataset/Data/6994/live/275919.jpg
creating: MLChallenge_Dataset/Data/6994/spoof/
inflating: MLChallenge_Dataset/Data/6994/spoof/062756.jpg
inflating: MLChallenge_Dataset/Data/6994/spoof/272804.jpg
creating: MLChallenge_Dataset/Data/6995/
creating: MLChallenge_Dataset/Data/6995/live/
inflating: MLChallenge_Dataset/Data/6995/live/364841.jpg
creating: MLChallenge_Dataset/Data/6995/spoof/
inflating: MLChallenge_Dataset/Data/6995/spoof/374741.jpg
inflating: MLChallenge_Dataset/Data/6995/spoof/379973.jpg
inflating: MLChallenge_Dataset/Data/6995/spoof/469401.jpg
creating: MLChallenge_Dataset/Data/6998/
creating: MLChallenge_Dataset/Data/6998/live/
creating: MLChallenge_Dataset/Data/6998/spoof/
inflating: MLChallenge_Dataset/Data/6998/spoof/267680.jpg
inflating: MLChallenge_Dataset/Data/6998/spoof/457271.jpg
creating: MLChallenge_Dataset/Data/7003/
creating: MLChallenge_Dataset/Data/7003/live/
inflating: MLChallenge_Dataset/Data/7003/live/056990.jpg
inflating: MLChallenge_Dataset/Data/7003/live/066594.jpg
inflating: MLChallenge_Dataset/Data/7003/live/185329.jpg
inflating: MLChallenge_Dataset/Data/7003/live/258055.jpg
creating: MLChallenge_Dataset/Data/7003/spoof/
creating: MLChallenge_Dataset/Data/7006/
creating: MLChallenge_Dataset/Data/7006/live/
creating: MLChallenge_Dataset/Data/7006/spoof/
inflating: MLChallenge_Dataset/Data/7006/spoof/103870.jpg
creating: MLChallenge_Dataset/Data/7007/
creating: MLChallenge_Dataset/Data/7007/live/

```

Copiar o dataset descompactado para o Google Drive

```
1 # %cp -r /content/MLChallenge_Dataset /content/drive/MyDrive/FACEMATCH
```

Copia do dataset original

```

1 # Criar uma cópia do dataset original
2 from distutils.dir_util import copy_tree
3 import os
4
5 # Defina o caminho do diretório original e o caminho do diretório de destino para a cópia
6 diretorio_original = '/content/MLChallenge_Dataset'
7 diretorio_destino = '/content/MLChallenge_Dataset_copia'
8
9 # Verifica se o diretório de destino já existe e, se não existir, cria-o
10 if not os.path.exists(diretorio_destino):
11     os.makedirs(diretorio_destino)
12
13 # Copia o diretório original para o diretório de destino
14 copy_tree(diretorio_original, diretorio_destino)

[ '/content/MLChallenge_Dataset_copia/annotations.json',
  '/content/MLChallenge_Dataset_copia/README',
  '/content/MLChallenge_Dataset_copia/Data/9191/spoof/511301.png',
  '/content/MLChallenge_Dataset_copia/Data/9191/spoof/509964.png',
  '/content/MLChallenge_Dataset_copia/Data/9191/spoof/558346.png',
  '/content/MLChallenge_Dataset_copia/Data/8828/live/495255.png',
  '/content/MLChallenge_Dataset_copia/Data/697/spoof/365582.jpg',
  '/content/MLChallenge_Dataset_copia/Data/697/spoof/197847.jpg',
  '/content/MLChallenge_Dataset_copia/Data/697/spoof/229535.jpg',
  '/content/MLChallenge_Dataset_copia/Data/697/spoof/030484.jpg',
  '/content/MLChallenge_Dataset_copia/Data/697/spoof/291393.jpg',
  '/content/MLChallenge_Dataset_copia/Data/581/spoof/330962.jpg',
  '/content/MLChallenge_Dataset_copia/Data/581/spoof/442402.jpg',
  '/content/MLChallenge_Dataset_copia/Data/581/spoof/069631.jpg',
  '/content/MLChallenge_Dataset_copia/Data/6504/spoof/075417.jpg',
  '/content/MLChallenge_Dataset_copia/Data/6504/spoof/050080.jpg',
  '/content/MLChallenge_Dataset_copia/Data/6504/spoof/307730.jpg',
  '/content/MLChallenge_Dataset_copia/Data/7322/live/555524.png',
  '/content/MLChallenge_Dataset_copia/Data/7322/live/499056.png',
  '/content/MLChallenge_Dataset_copia/Data/7322/live/496105.png',
  '/content/MLChallenge_Dataset_copia/Data/7969/live/096578.jpg',
  '/content/MLChallenge_Dataset_copia/Data/535/spoof/347088.jpg',
  '/content/MLChallenge_Dataset_copia/Data/535/spoof/375295.jpg',

```

```
'/content/MLChallenge_Dataset_copia/Data/3261/live/025086.jpg',
'/content/MLChallenge_Dataset_copia/Data/3261/spoof/153598.jpg',
'/content/MLChallenge_Dataset_copia/Data/1431/live/120416.jpg',
'/content/MLChallenge_Dataset_copia/Data/1431/spoof/175939.jpg',
'/content/MLChallenge_Dataset_copia/Data/1431/spoof/181874.jpg',
'/content/MLChallenge_Dataset_copia/Data/5855/live/085328.jpg',
'/content/MLChallenge_Dataset_copia/Data/5855/spoof/029115.jpg',
'/content/MLChallenge_Dataset_copia/Data/1593/spoof/478431.jpg',
'/content/MLChallenge_Dataset_copia/Data/1387/live/152208.jpg',
'/content/MLChallenge_Dataset_copia/Data/1387/spoof/382383.jpg',
'/content/MLChallenge_Dataset_copia/Data/1387/spoof/160538.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/live/413592.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/live/132670.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/live/191043.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/spoof/097759.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/spoof/245837.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/spoof/201368.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/spoof/128714.jpg',
'/content/MLChallenge_Dataset_copia/Data/1041/spoof/303148.jpg',
'/content/MLChallenge_Dataset_copia/Data/865/live/093872.jpg',
'/content/MLChallenge_Dataset_copia/Data/865/live/310120.jpg',
'/content/MLChallenge_Dataset_copia/Data/865/live/169898.jpg',
'/content/MLChallenge_Dataset_copia/Data/865/spoof/417626.jpg',
'/content/MLChallenge_Dataset_copia/Data/865/spoof/452938.jpg',
'/content/MLChallenge_Dataset_copia/Data/865/spoof/368430.jpg',
'/content/MLChallenge_Dataset_copia/Data/5924/live/233083.jpg',
'/content/MLChallenge_Dataset_copia/Data/5924/live/084888.jpg',
'/content/MLChallenge_Dataset_copia/Data/5924/spoof/144730.jpg',
'/content/MLChallenge_Dataset_copia/Data/5924/spoof/447256.jpg',
'/content/MLChallenge_Dataset_copia/Data/3035/spoof/136446.jpg',
'/content/MLChallenge_Dataset_copia/Data/3035/spoof/045537.jpg',
'/content/MLChallenge_Dataset_copia/Data/3035/spoof/044787.jpg',
'/content/MLChallenge_Dataset_copia/Data/6486/spoof/047048.jpg',
'/content/MLChallenge_Dataset_copia/Data/6486/spoof/361755.jpg',
```

Verificar se o dataset original e o copia tem a mesma dimensão

```
1 import os
2
3 def contar_arquivos(diretorio):
4     # Inicializa o contador de arquivos
5     contador = 0
6
7     # Percorre todos os arquivos e subdiretórios do diretório
8     for _, _, arquivos in os.walk(diretorio):
9         # Incrementa o contador com a quantidade de arquivos no diretório atual
10        contador += len(arquivos)
11
12    return contador
13
14 # Defina o caminho do diretório original e o caminho do diretório de destino da cópia
15 diretorio_original = '/content/MLChallenge_Dataset'
16 diretorio_copia = '/content/MLChallenge_Dataset_copia'
17
18 # Conta o número de arquivos nos diretórios original e de cópia
19 qtd_arquivos_original = contar_arquivos(diretorio_original)
20 qtd_arquivos_copia = contar_arquivos(diretorio_copia)
21
22 # Verifica se a quantidade de arquivos é a mesma
23 if qtd_arquivos_original == qtd_arquivos_copia:
24     print("O diretório original e a cópia têm a mesma quantidade de arquivos.")
25 else:
26     print("O diretório original e a cópia têm quantidades diferentes de arquivos.")
27
```

O diretório original e a cópia têm a mesma quantidade de arquivos.

Equilibrando o dataset

```
1 import os
2 import shutil
3
4 # Diretórios
5 diretorio = "/content/MLChallenge_Dataset_copia"
6
7 # Diretórios de expurgo
8 dir_expurgo_live = "/content/expurgo_live"
9 dir_expurgo_spoof = "/content/expurgo_spoof"
10
11 # Cria os diretórios de expurgo se não existirem
12 if not os.path.exists(dir_expurgo_live):
13     os.makedirs(dir_expurgo_live)
```

```

14
15 if not os.path.exists(dir_expurgo_spoof):
16     os.makedirs(dir_expurgo_spoof)
17
18 # Função para equilibrar as imagens em uma pasta numerada
19 def equilibrar_pasta_numerada(pasta_numerada):
20     live_dir = os.path.join(pasta_numerada, "live")
21     spoof_dir = os.path.join(pasta_numerada, "spoof")
22
23     # Conta as imagens nas pastas live e spoof
24     num_live_images = len(os.listdir(live_dir))
25     num_spoof_images = len(os.listdir(spoof_dir))
26
27     if num_live_images == 0 or num_spoof_images == 0:
28         # Exclui a pasta numerada se estiver vazia
29         shutil.rmtree(pasta_numerada)
30     else:
31         if num_live_images > num_spoof_images:
32             # Move as imagens excedentes de live para expurgo_live até que live == spoof
33             for _ in range(num_live_images - num_spoof_images):
34                 live_images = os.listdir(live_dir)
35                 if len(live_images) > 0:
36                     image_to_move = live_images[0]
37                     src = os.path.join(live_dir, image_to_move)
38                     dst = os.path.join(dir_expurgo_live, image_to_move)
39                     shutil.move(src, dst)
40
41         elif num_live_images < num_spoof_images:
42             # Move as imagens excedentes de spoof para expurgo_spoof até que live == spoof
43             for _ in range(num_spoof_images - num_live_images):
44                 spoof_images = os.listdir(spoof_dir)
45                 if len(spoof_images) > 0:
46                     image_to_move = spoof_images[0]
47                     src = os.path.join(spoof_dir, image_to_move)
48                     dst = os.path.join(dir_expurgo_spoof, image_to_move)
49                     shutil.move(src, dst)
50
51 # Loop para percorrer os diretórios numerados
52 for i in range(1, 11001):
53     diretorio_numerado = os.path.join(diretorio, "Data", str(i))
54
55     # Verifica se existem as pastas live e spoof
56     live_dir = os.path.join(diretorio_numerado, "live")
57     spoof_dir = os.path.join(diretorio_numerado, "spoof")
58
59     if os.path.exists(live_dir) and os.path.exists(spoof_dir):
60         equilibrar_pasta_numerada(diretorio_numerado)
61
62 # Verifica se as pastas estão equilibradas
63 total_live = sum(len(files) for _, _, files in os.walk(os.path.join(diretorio, "Data", "live")))
64 total_spoof = sum(len(files) for _, _, files in os.walk(os.path.join(diretorio, "Data", "spoof")))
65
66 print(f"Total de imagens 'live': {total_live}")
67 print(f"Total de imagens 'spoof': {total_spoof}")
68
69 Total de imagens 'live': 0
70 Total de imagens 'spoof': 0

```

Copiando images retiradas do dataset antes do treinamento, serão utilizadas para validação!

```
1 %cp -r /content/expurgo_live /content/drive/MyDrive/FACEMATCH
```

```
1 %cp -r /content/expurgo_spoof /content/drive/MyDrive/FACEMATCH
```

Verificando se o Dataset esta equilibrado

```

1 import os
2 import glob
3
4 def contar_imagens(diretorio_base):
5     # Inicializar contadores para imagens "live" e "spoof"
6     total_live = 0
7     total_spoof = 0
8
9     # Percorrer todas as pastas numeradas de 1 a 11000
10    for pasta_numerada in range(1, 11001):
11        pasta_path = os.path.join(diretorio_base, "Data", str(pasta_numerada))
12
13        # Contar imagens na pasta "live"

```

```

14     pasta_live_path = os.path.join(pasta_path, "live")
15     live_files = glob.glob(os.path.join(pasta_live_path, "*.jpg")) # Modifique "*.jpg" para o formato das imagens se necessário
16     total_live += len(live_files)
17
18     # Contar imagens na pasta "spoof"
19     pasta_spoof_path = os.path.join(pasta_path, "spoof")
20     spoof_files = glob.glob(os.path.join(pasta_spoof_path, "*.jpg")) # Modifique "*.jpg" para o formato das imagens se necessário
21     total_spoof += len(spoof_files)
22
23     return total_live, total_spoof
24
25 diretorio_base = "/content/MLChallenge_Dataset_copia"
26 total_live, total_spoof = contar_imagens(diretorio_base)
27
28 print("Total de imagens 'live':", total_live)
29 print("Total de imagens 'spoof':", total_spoof)
30

Total de imagens 'live': 6125
Total de imagens 'spoof': 5973

```

Separando o dataset em treino e teste

```

1 import os
2 import re
3 import random
4 import shutil
5
6 def count_folders_with_numbers(starting_directory, lower_bound, upper_bound):
7     count = 0
8     matching_folders = []
9     for folder_name in os.listdir(starting_directory):
10         if re.match(r'^[1-9]\d{0,3}$|^10\d{3}$|^11\d{3}$', folder_name):
11             count += 1
12             matching_folders.append(folder_name)
13
14     return count, matching_folders
15
16 def split_folders_for_training_test(folder_list, train_ratio=0.8):
17     random.shuffle(folder_list)
18     split_index = int(len(folder_list) * train_ratio)
19     train_folders = folder_list[:split_index]
20     test_folders = folder_list[split_index:]
21     return train_folders, test_folders
22
23 # Example usage:
24 diretorio_inicial = '/content/MLChallenge_Dataset_copia/Data'
25 total_folders, matching_folders = count_folders_with_numbers(diretorio_inicial, 1, 11000)
26 print(f"Total number of folders found: {total_folders}")
27
28 # Separating folders for training and testing
29 train_folders, test_folders = split_folders_for_training_test(matching_folders)
30
31 # Printing results
32 print(f"Number of folders for training: {len(train_folders)}")
33 print(f"Number of folders for testing: {len(test_folders)}")
34
35 # Move folders to separate directories for training and testing (optional)
36 train_dir = '/content/train_data'
37 test_dir = '/content/test_data'
38
39 os.makedirs(train_dir, exist_ok=True)
40 os.makedirs(test_dir, exist_ok=True)
41
42 for folder_name in train_folders:
43     shutil.move(os.path.join(diretorio_inicial, folder_name), os.path.join(train_dir, folder_name))
44
45 for folder_name in test_folders:
46     shutil.move(os.path.join(diretorio_inicial, folder_name), os.path.join(test_dir, folder_name))
47

Total number of folders found: 4948
Number of folders for training: 3958
Number of folders for testing: 990

```

Copiando pasta de treino e teste

```
1 # %cp -r /content/test_data /content/drive/MyDrive/FACEMATCH/dataset
```

```
1 # %cp -r /content/train_data /content/drive/MyDrive/FACEMATCH/dataset
```

Passo 3: Treinar a rede neural | Avaliar o desempenho da rede

```
1 !pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.56.2)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.13)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!>4.21.1,!>4.21.2,!>4.21.3,!>4.21.4,!>4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/p
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12.3)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.1
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.7.1)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (
Requirement already satisfied: ml-dtypes>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow) (0.2.0)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow) (1.10.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tens
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensor
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tenso
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensor
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard)
Requirement already satisfied: charset-normalizer~>2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-au
```

Separando as imagens em live e spoof

```
1 import os
2 import shutil
3
4 def find_and_separate_files(source_dir, destination_dir, target_categories):
5     for root, _, files in os.walk(source_dir):
6         for category in target_categories:
7             if category in root:
8                 category_dir = os.path.join(destination_dir, f"new_Train_data_{category}")
9                 os.makedirs(category_dir, exist_ok=True)
10
11             for file in files:
12                 source_file_path = os.path.join(root, file)
13                 destination_file_path = os.path.join(category_dir, file)
14                 shutil.move(source_file_path, destination_file_path)
15
16 if __name__ == "__main__":
17     source_directory = "/content/train_data"
18     destination_directory = "/content"
19
20     # Replace "live" and "spoof" with the actual folder names you have in the train_data directory.
21     target_categories_to_separate = ["live", "spoof"]
22
23     find_and_separate_files(source_directory, destination_directory, target_categories_to_separate)
24
```

Treinamento da Rede Neural

Esta rede neural é um modelo de classificação binária construído usando a biblioteca TensorFlow e a API Keras. Ele é projetado para classificar imagens em duas classes: "live" e "spoof". O objetivo provável é classificar se uma imagem é uma imagem real ("live") ou uma imagem falsificada ("spoof").

A arquitetura da rede neural é a seguinte:

- Camada de entrada: Essa camada recebe as imagens de entrada com dimensões (img_width, img_height, 3), onde img_width e img_height são definidos como 128 pixels. O número 3 refere-se aos três canais de cores RGB das imagens.
- Camadas de convolução e max pooling: A rede contém três camadas de convolução, cada uma seguida por uma camada de max pooling. Cada camada de convolução tem filtros de tamanho 3x3 e as funções de ativação usadas são a função "ReLU" (Rectified Linear Unit), que é comumente usada em redes neurais para adicionar não-linearidade.
- Camada Flatten: Após as camadas de convolução e max pooling, os dados são achados em um vetor unidimensional usando a camada Flatten. Isso prepara os dados para serem passados por camadas densas (totalmente conectadas).
- Camada densa: A rede possui uma camada densa (totalmente conectada) com 128 unidades e uma função de ativação "ReLU". Isso permite que a rede aprenda características complexas dos dados.
- Camada Dropout: Para evitar o overfitting, é aplicada uma camada de Dropout com uma taxa de 0,5. O Dropout desativa aleatoriamente alguns neurônios durante o treinamento para tornar a rede mais robusta.
- Camada de saída: A última camada é uma camada densa com uma unidade de saída e a função de ativação "sigmoid". A função de ativação "sigmoid" é usada para converter a saída em um valor entre 0 e 1, que representa a probabilidade da imagem pertencer à classe "live" (0) ou "spoof" (1).
- O modelo é compilado usando o otimizador "adam" e a função de perda "binary_crossentropy", que é apropriada para problemas de classificação binária. A métrica de avaliação é a "accuracy" (acurácia), que mede a precisão da classificação.
- Durante o treinamento, são utilizados geradores de dados para pré-processamento e aumento dos dados de treinamento. O aumento de dados é realizado aplicando transformações aleatórias nas imagens para aumentar a diversidade do conjunto de treinamento.
- O modelo é treinado em duas etapas: primeiro usando dados "live" e depois usando dados "spoof". Após o treinamento, o modelo é salvo em um arquivo "trained_model4.h5" no diretório "saved_model".

Cabe mencionar que o código de treinamento poderia ser aprimorado adicionando etapas de validação para monitorar o desempenho do modelo durante o treinamento e ajustar hiperparâmetros para obter melhores resultados. Além disso, os dados "live" e "spoof" devem estar bem equilibrados para garantir uma aprendizagem adequada da rede neural.

Primeira versão

```

1 import os
2 import tensorflow as tf
3 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
4 from tensorflow.keras.models import Model
5 from tensorflow.keras.preprocessing.image import ImageDataGenerator
6
7 # Diretórios das imagens de treino
8 train_data_root_live = '/content/new_Train_data_live'
9 train_data_root_spoof = '/content/new_Train_data_spoof'
10
11 # Verificar o conteúdo dos diretórios
12 print("Conteúdo do diretório 'new_Train_data_live':")
13 print(os.listdir(train_data_root_live))
14
15 print("\nConteúdo do diretório 'new_Train_data_spoof':")
16 print(os.listdir(train_data_root_spoof))
17
18 # Dimensões das imagens para redimensionamento
19 img_width, img_height = 128, 128
20
21 # Parâmetros do treinamento
22 batch_size = 32
23 epochs = 10
24
25 # Pré-processamento e aumento de dados para imagens reais (live)
26 train_datagen_live = ImageDataGenerator(
27     rescale=1.0/255,
28     shear_range=0.2,
29     zoom_range=0.2,
30     horizontal_flip=True,
31     rotation_range=30,
32     width_shift_range=0.1,
33     height_shift_range=0.1
34 )
35
36 # Carregamento das imagens de treino reais (live)

```

```

37 train_generator_live = train_datagen_live.flow_from_directory(
38     train_data_root_live,
39     target_size=(img_width, img_height),
40     batch_size=batch_size,
41     class_mode='binary'
42 )
43
44 # Pré-processamento e aumento de dados para imagens falsas (spoof)
45 train_datagen_spoof = ImageDataGenerator(
46     rescale=1.0/255,
47     shear_range=0.2,
48     zoom_range=0.2,
49     horizontal_flip=True,
50     rotation_range=30,
51     width_shift_range=0.1,
52     height_shift_range=0.1
53 )
54
55 # Carregamento das imagens de treino falsas (spoof)
56 train_generator_spoof = train_datagen_spoof.flow_from_directory(
57     train_data_root_spoof,
58     target_size=(img_width, img_height),
59     batch_size=batch_size,
60     class_mode='binary'
61 )
62
63 # Criação do modelo
64 def create_model():
65     input_layer = tf.keras.Input(shape=(img_width, img_height, 3))
66     x = Conv2D(32, (3, 3), activation='relu')(input_layer)
67     x = MaxPooling2D((2, 2))(x)
68     x = Conv2D(64, (3, 3), activation='relu')(x)
69     x = MaxPooling2D((2, 2))(x)
70     x = Conv2D(128, (3, 3), activation='relu')(x)
71     x = MaxPooling2D((2, 2))(x)
72     x = Flatten()(x)
73     x = Dense(128, activation='relu')(x)
74     x = Dropout(0.5)(x)
75     output_layer = Dense(1, activation='sigmoid')(x)
76
77     model = Model(inputs=input_layer, outputs=output_layer)
78     return model
79
80 # Criar o modelo
81 model = create_model()
82
83 # Compilar o modelo
84 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
85
86 # Sumário do modelo
87 model.summary()
88
89 # Treinar o modelo usando dados reais (live)
90 if len(train_generator_live) > 0:
91     model.fit_generator(train_generator_live, steps_per_epoch=len(train_generator_live), epochs=epochs)
92 else:
93     print("Não há imagens suficientes de live para treinar o modelo.")
94
95 # Treinar o modelo usando dados falsos (spoof)
96 if len(train_generator_spoof) > 0:
97     model.fit_generator(train_generator_spoof, steps_per_epoch=len(train_generator_spoof), epochs=epochs)
98 else:
99     print("Não há imagens suficientes de spoof para treinar o modelo.")
100
101 # Salvar o modelo após o treinamento
102 save_path = '/content/saved_model/'
103 if not os.path.exists(save_path):
104     os.makedirs(save_path)
105
106 model.save(os.path.join(save_path, 'trained_model4.h5'))
107 print("Modelo salvo com sucesso!")
108

Conteúdo do diretório 'new_Train_data_live':
['008186.jpg', '096105.jpg', '182647.jpg', '556745.png', '187765.jpg', '313412.jpg', '286795.jpg', '125444.jpg', '369724.jpg', '533

Conteúdo do diretório 'new_Train_data_spoof':
['498691.png', '241163.jpg', '238047.jpg', '143115.jpg', '436177.jpg', '321214.jpg', '083129.jpg', '419293.jpg', '114910.jpg', '269
Found 0 images belonging to 0 classes.
Found 0 images belonging to 0 classes.
Model: "model"

```

Layer (type)	Output Shape	Param #
<hr/>		

input_1 (InputLayer)	[None, 128, 128, 3]	0
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3211392
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 3,304,769
 Trainable params: 3,304,769
 Non-trainable params: 0

Não há imagens suficientes de live para treinar o modelo.
 Não há imagens suficientes de spoof para treinar o modelo.
 Modelo salvo com sucesso!

Segunda versão

```

1 import os
2 import tensorflow as tf
3 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
4 from tensorflow.keras.models import Model
5 from tensorflow.keras.preprocessing.image import ImageDataGenerator
6
7 # Diretórios das imagens de treino
8 train_data_root_live = '/content/new_Train_data_live'
9 train_data_root_spoof = '/content/new_Train_data_spoof'
10
11 # Verificar o conteúdo dos diretórios
12 print("Conteúdo do diretório 'new_Train_data_live':")
13 print(os.listdir(train_data_root_live))
14
15 print("\nConteúdo do diretório 'new_Train_data_spoof':")
16 print(os.listdir(train_data_root_spoof))
17
18 # Dimensões das imagens para redimensionamento
19 img_width, img_height = 128, 128
20
21 # Parâmetros do treinamento
22 batch_size = 32
23 epochs = 10
24
25 # Pré-processamento e aumento de dados para imagens reais (live)
26 train_datagen_live = ImageDataGenerator(
27     rescale=1.0/255,
28     shear_range=0.2,
29     zoom_range=0.2,
30     horizontal_flip=True,
31     rotation_range=30,
32     width_shift_range=0.1,
33     height_shift_range=0.1
34 )
35
36 # Carregamento das imagens de treino reais (live)
37 train_generator_live = train_datagen_live.flow_from_directory(
38     train_data_root_live,
39     target_size=(img_width, img_height),
40     batch_size=batch_size,
41     class_mode='binary'
42 )
43
44 # Pré-processamento e aumento de dados para imagens falsas (spoof)
45 train_datagen_spoof = ImageDataGenerator(
46     rescale=1.0/255,
```

```

47     shear_range=0.2,
48     zoom_range=0.2,
49     horizontal_flip=True,
50     rotation_range=30,
51     width_shift_range=0.1,
52     height_shift_range=0.1
53 )
54
55 # Carregamento das imagens de treino falsas (spoof)
56 train_generator_spoof = train_datagen_spoof.flow_from_directory(
57     train_data_root_spoof,
58     target_size=(img_width, img_height),
59     batch_size=batch_size,
60     class_mode='binary'
61 )
62
63 # Criação do modelo
64 def create_model():
65     input_layer = tf.keras.Input(shape=(img_width, img_height, 3))
66     x = Conv2D(32, (3, 3), activation='relu')(input_layer)
67     x = MaxPooling2D((2, 2))(x)
68     x = Conv2D(64, (3, 3), activation='relu')(x)
69     x = MaxPooling2D((2, 2))(x)
70     x = Conv2D(128, (3, 3), activation='relu')(x)
71     x = MaxPooling2D((2, 2))(x)
72     x = Conv2D(256, (3, 3), activation='relu')(x) # Nova camada convolucional
73     x = MaxPooling2D((2, 2))(x) # Nova camada de pooling
74     x = Flatten()(x)
75     x = Dense(512, activation='relu') # Aumentar o número de neurônios
76     x = Dropout(0.5)(x) # Ajustar a taxa de dropout
77     x = Dense(256, activation='relu') # Nova camada densa
78     x = Dropout(0.5)(x) # Ajustar a taxa de dropout
79     output_layer = Dense(1, activation='sigmoid')(x)
80
81     model = Model(inputs=input_layer, outputs=output_layer)
82     return model
83
84 # Criar o modelo
85 model = create_model()
86
87 # Compilar o modelo
88 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
89
90 # Sumário do modelo
91 model.summary()
92
93 # Treinar o modelo usando dados reais (live)
94 if len(train_generator_live) > 0:
95     model.fit_generator(train_generator_live, steps_per_epoch=len(train_generator_live), epochs=epochs)
96 else:
97     print("Não há imagens suficientes de live para treinar o modelo.")
98
99 # Treinar o modelo usando dados falsos (spoof)
100 if len(train_generator_spoof) > 0:
101     model.fit_generator(train_generator_spoof, steps_per_epoch=len(train_generator_spoof), epochs=epochs)
102 else:
103     print("Não há imagens suficientes de spoof para treinar o modelo.")
104
105 # Salvar o modelo após o treinamento
106 save_path = '/content/saved_model/'
107 if not os.path.exists(save_path):
108     os.makedirs(save_path)
109
110 model.save(os.path.join(save_path, 'trained_model5.h5'))
111 print("Modelo salvo com sucesso!")
112

Conteúdo do diretório 'new_Train_data_live':
['008186.jpg', '096105.jpg', '182647.jpg', '556745.png', '187765.jpg', '313412.jpg', '286795.jpg', '125444.jpg', '369724.jpg', '533

Conteúdo do diretório 'new_Train_data_spoof':
['498691.png', '241163.jpg', '238047.jpg', '143115.jpg', '436177.jpg', '321214.jpg', '083129.jpg', '419293.jpg', '114910.jpg', '269
Found 0 images belonging to 0 classes.
Found 0 images belonging to 0 classes.
Model: "model_3"
```

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 128, 128, 3)]	0
conv2d_11 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_11 (MaxPooling2D)	(None, 63, 63, 32)	0

conv2d_12 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_12 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_13 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_13 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_14 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_14 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten_3 (Flatten)	(None, 9216)	0
dense_8 (Dense)	(None, 512)	4719104
dropout_5 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 256)	131328
dropout_6 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 1)	257

```
=====
Total params: 5,239,105
Trainable params: 5,239,105
Non-trainable params: 0
```

```
Não há imagens suficientes de live para treinar o modelo.
Não há imagens suficientes de spoof para treinar o modelo.
Modelo salvo com sucesso!
```

Copiando/ Salvando pesos do modelo

```
1 %cp -r /content/saved_model/trained_model16.h5 /content/drive/MyDrive/FACEMATCH/weights
```

Passo 6: Teste

```
1 import tensorflow as tf
2 from tensorflow.keras.models import load_model
3
4 # Caminho para o arquivo .h5 do modelo
5 model_path = '/content/saved_model/trained_model14.h5'
6
7 # Carregar o modelo
8 model = load_model(model_path)
9
10 # Resumo do modelo (opcional, mostra a arquitetura do modelo)
11 model.summary()
12
13 # Exemplo de como fazer previsões com o modelo
14 # Neste exemplo, você precisa ter uma imagem de teste (ou mais) para fazer a previsão
15 # Substitua 'caminho/para/a/imagem.jpg' pelo caminho da imagem de teste que você quer usar
16 image_path = '/content/expurgo_spoof/000090.jpg'
17 image = tf.keras.preprocessing.image.load_img(image_path, target_size=(128, 128))
18 image_array = tf.keras.preprocessing.image.img_to_array(image)
19 image_array = image_array / 255.0 # Normalização dos valores dos pixels (se necessário)
20 image_array = tf.expand_dims(image_array, 0) # Adicionando uma dimensão extra para representar o batch
21
22 # Fazendo a previsão com o modelo carregado
23 predictions = model.predict(image_array)
24
25 # Obtendo o resultado da previsão (0 ou 1 no caso de classificação binária)
26 predicted_class = int(predictions[0][0] + 0.5)
27
28 # Imprimindo o resultado da previsão
29 if predicted_class == 0:
30     print("A imagem é classificada como 'real'.")
31 else:
32     print("A imagem é classificada como 'fake'.")
33
```

Model: "model_3"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 128, 128, 3)]	0

conv2d_11 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_11 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_12 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_12 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_13 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_13 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_14 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_14 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten_3 (Flatten)	(None, 9216)	0
dense_8 (Dense)	(None, 512)	4719104
dropout_5 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 256)	131328
dropout_6 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 1)	257

```
=====
Total params: 5,239,105
Trainable params: 5,239,105
Non-trainable params: 0
```

```
1/1 [=====] - 0s 81ms/step
A imagem é classificada como 'fake'.
```

Teste - imagens da pasta LIVE

```

1 import os
2 import tensorflow as tf
3 from tensorflow.keras.models import load_model
4
5 # Caminho para o arquivo .h5 do modelo
6 model_path = '/content/saved_model/trained_model4.h5'
7
8 # Carregar o modelo
9 model = load_model(model_path)
10
11 # Função para fazer previsões com o modelo e contar os resultados
12 def predict_and_count(image_path):
13     image = tf.keras.preprocessing.image.load_img(image_path, target_size=(128, 128))
14     image_array = tf.keras.preprocessing.image.img_to_array(image)
15     image_array = image_array / 255.0 # Normalização dos valores dos pixels (se necessário)
16     image_array = tf.expand_dims(image_array, 0) # Adicionando uma dimensão extra para representar o batch
17
18     # Fazendo a previsão com o modelo carregado
19     predictions = model.predict(image_array)
20
21     # Obtendo o resultado da previsão (0 ou 1 no caso de classificação binária)
22     predicted_class = int(predictions[0][0] + 0.5)
23
24     return predicted_class
25
26 # Inicializando contadores
27 certas_live = 0
28 falsas_live = 0
29
30 # Loop para percorrer os arquivos na pasta
31 folder_path = '/content/expurgo_live'
32 for filename in os.listdir(folder_path):
33     if filename.endswith('.jpg'):
34         image_path = os.path.join(folder_path, filename)
35         predicted_class = predict_and_count(image_path)
36
37         # Atualizando contadores
38         if predicted_class == 0:
39             certas_live += 1
40         else:
41             falsas_live += 1
42

```

```
1 import os
2 import tensorflow as tf
3 from tensorflow.keras.models import load_model
4
5 # Caminho para o arquivo .h5 do modelo
6 model_path = '/content/saved_model/trained_model14.h5'
7
8 # Carregar o modelo
9 model = load_model(model_path)
10
11 # Função para fazer previsões com o modelo e contar os resultados
12 def predict_and_count(image_path):
13     image = tf.keras.preprocessing.image.load_img(image_path, target_size=(128, 128))
14     image_array = tf.keras.preprocessing.image.img_to_array(image)
15     image_array = image_array / 255.0 # Normalização dos valores dos pixels (se necessário)
16     image_array = tf.expand_dims(image_array, 0) # Adicionando uma dimensão extra para representar o batch
17
18     # Fazendo a previsão com o modelo carregado
19     predictions = model.predict(image_array)
20
21     # Obtendo o resultado da previsão (0 ou 1 no caso de classificação binária)
```

Avaliando a performance do modelo

```

1 def calcular_percentual_assertividade(real_count_live, fake_count_live, real_count_spoof, fake_count_spoof):
2     total_real = real_count_live + real_count_spoof
3     total_fake = fake_count_live + fake_count_spoof
4     total_amostras = total_real + total_fake
5
6     percentual_assertividade = (total_real / total_amostras) * 100
7
8     return percentual_assertividade
9
10 # Exemplo de uso:
11 real_count_live = certas_live
12 fake_count_live = falsas_live
13 real_count_spoof = falsas_spoof
14 fake_count_spoof = certas_spoof
15
16 percentual_assertividade = calcular_percentual_assertividade(real_count_live, fake_count_live, real_count_spoof, fake_count_spoof)
17 print(f'O percentual de assertividade é: {percentual_assertividade:.2f}%')
18
19 O percentual de assertividade é: 93.27%

```

Conclusão

```

1 """O código fornecido é um exemplo de um modelo de classificação binária usando imagens como entrada. No entanto, não está claro qua
2
3 Vou explicar o código para melhor compreensão:
4
5 Carregamento e pré-processamento dos dados de treino:
6 O código usa a biblioteca Keras com o TensorFlow backend para carregar e pré-processar os dados de treinamento. O ImageDataGenerator
7
8 Criação do modelo:
9 O modelo é criado usando uma sequência de camadas convolucionais e de pooling, seguida por camadas densas. O modelo tem a seguinte e
10
11 Camada de entrada: Aceita imagens com dimensões (img_width, img_height, 3) (3 canais de cor RGB).
12 Três camadas convolucionais com ativação ReLU e três camadas MaxPooling para redução de dimensões.
13 Camada Flatten para transformar a saída das camadas convolucionais em um vetor 1D.
14 Uma camada densa com ativação ReLU.
15 Uma camada Dropout para regularização.
16 Camada de saída com ativação sigmóide, que é a função de ativação usada em problemas de classificação binária.
17 Compilação e treinamento do modelo:
18 O modelo é compilado com o otimizador "adam" e a função de perda "binary_crossentropy" para problemas de classificação binária. Em s
19
20 Salvando o modelo treinado:
21 O modelo é salvo em formato .h5 após o treinamento.
22
23 Se o modelo está classificando todas as imagens como falsas, pode haver algumas razões possíveis:
24
25 O conjunto de dados de treinamento pode estar desequilibrado, ou seja, pode haver uma quantidade significativamente maior de exemplo
26 O modelo pode ser muito simples ou muito complexo para o problema em questão, o que pode levar a um ajuste inadequado (overfitting)
27 Os parâmetros de treinamento, como taxa de aprendizado, tamanho do lote e número de épocas, podem precisar ser ajustados para obter
28 Para obter uma análise mais aprofundada e identificar possíveis problemas com o modelo, é importante verificar o conjunto de dados c

```

'O código fornecido é um exemplo de um modelo de classificação binária usando imagens como entrada. No entanto, não está claro qual é exatamente o problema que você está enfrentando ou o que quer dizer com "esse modelo está identificando qualquer imagem como fake".\n\nVou explicar o código para melhor compreensão:\n\nCarregamento e pré-processamento dos dados de treino:\nO código usa a biblioteca Keras com o TensorFlow backend para carregar e pré-processar os dados de treinamento. O ImageDataGenerator é usado para aplicar transformações nas imagens, como normalização, rotação, deslocamento horizontal/vertical e outros aumentos de dados. Ele carrega as imagens do diretório raiz de treinamento (train_data_root) e divide as imagens em lotes para o treinamento (batch_size).\n\nCriação do modelo:\nO modelo é criado usando uma sequência de camadas convolucionais e de pooling, seguida por camadas densas.'

1

