CLASSIFICAÇÃO DE IMAGENS

IMAGE CLASSIFICATION

CARLOS HENRIQUE GERMANO

Graduando em Sistema de informação, Centro Universitário FAESA,
willck2.ch@gmail.com

DANIEL SILLER THEBALDI

Graduando em Sistema de informação, Centro Universitário FAESA,
daniel.thebaldi@gmail.com

FLAYNER SIQUEIRA BARRETO

Graduando em Sistema de informação, Centro Universitário FAESA,
fbflayner@gmail.com

THIAGO LEITE PIMENTEL

Graduando em Ciência da Computação, Centro Universitário FAESA,
thiagolp86@gmail.com

TIAGO DOS SANTOS RODRIGUES

Graduando em Sistema de informação, Centro Universitário FAESA,
tiiago.santosr@gmail.com

HOWARD CRUZ ROATTI

Prof.º Msc.
Centro Universitário FAESA
howard.cruz@faesa.br

RESUMO

O objetivo foi demonstrar como treinar uma rede neural para identificar as características de cada item e para isso foi utilizado a biblioteca Keras de rede neural de código aberto que é uma API de aprendizado profundo escrita em Python.

Keras é executada na plataforma de aprendizado de máquina TensorFlow e, para o experimento foi utilizado uma pequena versão da rede Xception que é um modelo pré-treinado para reconhecimento de imagem. Além disso, para o experimento foi utilizado um Dataset de imagens de pessoas famosas.

Palavras-chave: Machine Learning, Deep Learning, TensorFlow , Keras, Classificação de imagem

1 INTRODUÇÃO

Este artigo apresenta a recriação do experimento de classificação de imagem usando como referência o dataset Kaggle Cats vs Dogs os arquivos de imagens utilizado foram JPEG a base recriada foi LFW Face Database, utilizando um conjunto de dados através de camadas de pré-processamento de imagens keras para padronizar as figuras e o aumento dos dados. Por que utilizar estas técnicas?

Com o aumento do poder computacional e o grande volume de dados gerados, o uso do aprendizado de máquina (Machine Learning) e aprendizado profundo (Deep Learning), está sendo utilizado cada vez mais, para construir soluções ainda mais precisas, através da utilização de redes neurais para simular o processamento do cérebro humano, com o uso de algoritmos treinados, onde as informações são passadas por camadas sendo a primeira camada a entrada para próxima, cada uma é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação.

Estas técnicas vêm permitindo aprender e reconhecer padrões visuais e executar algumas tarefas que antes, só eram feitas por humanos. Com os avanços da aprendizagem profunda e visão computacional como podemos experimentar na é possível perceber que através do treinamento destes algoritmos será possível construir soluções cada vez mais relevantes para sociedade.

2 REVISÃO DA LITERATURA

2.1 MACHINE LEARNING E DEEP LEARNING

Na última década explodiu o interesse por Machine Learning (ou aprendizado de máquina), o mundo vem passando por uma transformação onde a interação de aplicação de computador e seres humanos é crescente, mas o que se trata Machine Learning? Em resumo é a utilização de algoritmo para extrair informações de dados brutos e representá-los em forma modelo matemático, onde este modelo fará inferência (previsões) a partir de outros conjuntos de dados, hoje existem muitos algoritmos que permitem fazer isso, onde cabe ao analista de dados sabre qual se encaixa com o problema que ele quer resolver (DATA SCIENCE ACADEMY, 2021).

Em meados dos anos 2000 com a expansão exponencial do poder computacional e o mercado viu-se uma explosão de técnicas computacionais que não eram possíveis antes. Foi nesse momento que começou a surgir o aprendizado profundo (Deep Learning), com o crescimento computacional dessa década foi possível construir sistemas de Inteligência Artificial cada vez mais preciso. Sendo assim o interesse por Deep Learning não parou de crescer e hoje vemos o termo sendo usado cada vez mais e soluções surgindo a todo momento. (DEEPLARNINGBOOK, 2021).

Deep Learning ou Aprendizado Profundo, consiste em uma sub-área da do Aprendizado de Máquina que utiliza algoritmos para processar dados e imitar o processamento feito pelo cérebro humano conhecido como Redes Neurais. O que são Redes Neurais Artificiais Profundas ou Dep Learning? São camadas de neurônio Matemáticos para processar dados e compreender a fala humana, reconhecer objetos visualmente. Estas informações são passadas através de camadas, sendo a saída da camada anterior fornecendo a entrada para a próxima. Sendo assim o aprendizado profundo vem sendo responsável pelo avanço da visão computacional. Onde utiliza a extração de recursos computacionais que é outro aspecto da Aprendizagem profunda, onde esses recursos usam algoritmos para construir automaticamente “soluções” significativas dos dados para fins de treinamento, aprendizado e compreensão, para conseguir reconhecer áudio, imagens, vídeo e uma variedade de categoria de dados possíveis, baseado em aprendizado profundo ou Deep Learning (Redes Neurais Artificiais). (DEEPLARNINGBOOK, 2021).

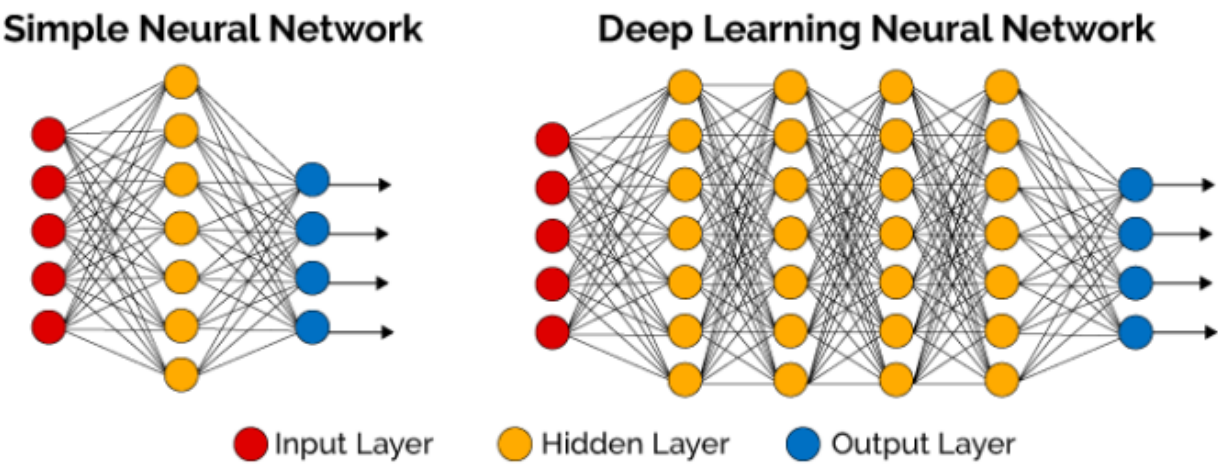


Figura 1 — Rede Neural Simples e Rede Neural Profunda (Deep Learning)

Fonte: DEEPLARNINGBOOK, 2021

2.2 TENSORFLOW E KERAS

O TensorFlow é uma plataforma de aprendizado de máquina de código aberto de ponta a ponta. Você pode pensar nisso como uma camada de infraestrutura para programação diferenciável. Ele combina quatro habilidades principais:

- Execução eficiente de operações de tensor de baixo nível na CPU, GPU ou TPU.
- Calculando o gradiente de expressões diferenciáveis arbitrárias.
- Escalonar a computação para muitos dispositivos, como clusters de centenas de GPUs.
- Exportar programas ("gráficos") para tempos de execução externos, como servidores, navegadores, dispositivos móveis e incorporados.

Keras é a API de alto nível do TensorFlow: uma interface acessível e altamente produtiva para resolver problemas de aprendizado de máquina, com foco no aprendizado profundo moderno. Ele fornece abstrações essenciais e blocos de construção para desenvolver e enviar soluções de aprendizado de máquina com alta velocidade de interação.

Para reconhecimento de imagem, Keras disponibiliza vários modelos pré-treinados, por exemplo, VGG16, VGG19, Resne50, Inception V3 e XCeption. Neste artigo foi utilizado o modelo XCeption para classificação de imagens. O modelo Xception é proposto por

François Chollet. Xception é uma extensão da arquitetura inicial que substitui os módulos iniciais padrão por convoluções separáveis em profundidade.

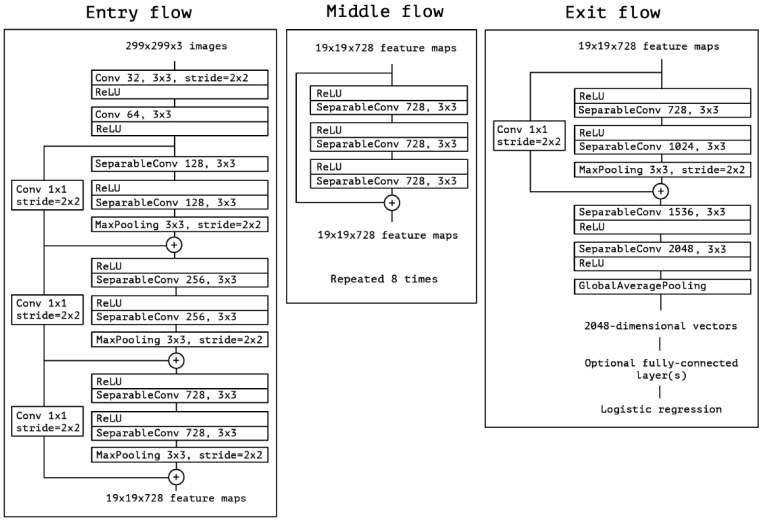


Figura 2 — Arquitetura geral do Xception (fluxo de entrada> fluxo médio> fluxo de saída)

Fonte: TOWARDS DATA SCIENCE, 2018

3 CONFIGURAÇÃO DO AMBIENTE

Uma parte do experimento foi realizada no Jupyter Notebook em um computador com Windows 10. Para isso foi necessário instalar o Python e as seguintes bibliotecas via pip: virtualenv, pandas, scipy, matplotlib, jupyter, tensorflow, pydot, pydotplus, graphviz.

Outra parte do experimento foi realizada no Google Colab, por meio dele é mais fácil configurar o ambiente, que já vem praticamente pronto para uso. Apenas requer uma configuração de hardware definindo o uso de GPU, que deixará o cálculo muito mais rápido e eficiente, mas para isso é necessário uma conta Colab Pro.

Uma vez configurado o ambiente quer seja por meio do Jupyter Notebook ou pelo Google Colab vamos a implementação do experimento.

4 IMPLEMENTAÇÃO DO EXPERIMENTO

Importando TensorFlow e Keras

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Conjunto de dados do FLW

Donwload do arquivo ltw.tgz contendo 172 MB de imagens:

```
!curl -O "http://vis-www.cs.umass.edu/lfw/lfw.tgz"
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  172M  100  172M    0     0  26.7M      0  0:00:06  0:00:06 --:--:-- 31.6M
```

Descompactando lfw.tgz.

```
!tar -xzvf lfw.tgz
```

A saída de streaming foi truncada nas últimas 5000 linhas.

Filtra imagens corrompidas

Filtra imagens mal codificadas, que não apresentam a string "JFIF" no cabecalho.

```
import os

num_skipped = 0
for folder_name in os.listdir("lfw"):
    folder_path = os.path.join("lfw", folder_name)
    for fname in os.listdir(folder_path):
        fpath = os.path.join(folder_path, fname)
        try:
            fobj = open(fpath, "rb")
            is_jfif = tf.compat.as_bytes("JFIF") in fobj.peek(10)
        finally:
            fobj.close()

        if not is_jfif:
            num_skipped += 1
            os.remove(fpath)

print("Deleted %d images" % num_skipped)
```

```
Deleted 0 images
```

Analisando os dados obtidos pelo FLW

Verificando quantas pessoas existe no acervo do FLW.

```
qt = len([item for item in os.listdir("lfw")])
print(f'{qt} pessoas')
```

```
5749 pessoas
```

Verificando as pessoas com maior acervo de imagens

```
import operator

folders = []
for folder_name in os.listdir("lfw"):
    folder_path = os.path.join("lfw", folder_name)
    qt = len(os.listdir(folder_path))
    if qt >= 100:
        folders.append({'folder': folder_name, 'qt': qt})

folders.sort(key=operator.itemgetter('qt', 'folder'))
for f in folders:
    folder = f['folder']
    qt = f['qt']
    print(f'{folder:20} ==> {qt:4}')
```

```
Gerhard_Schroeder    ==>  109
Donald_Rumsfeld      ==>  121
Tony_Blair           ==>  144
Colin_Powell         ==>  236
George_W_Bush        ==>  530
```

Cria pasta de processamento

Como podemos notar o FLW fornece relamente um número muito grande de faces para serem analizadas, porem a grande maioria das pessoas não possuem um número significativo de faces. Desta forma criaremos uma pasta com o objetivo de filtrar as pessoas que serão processadas.

```
dirName = 'lfw_dados'
if not os.path.exists(dirName):
    os.makedirs(dirName)

dirName = 'lfw_dados/0'
if not os.path.exists(dirName):
    os.makedirs(dirName)

dirName = 'lfw_dados/1'
if not os.path.exists(dirName):
    os.makedirs(dirName)
```

Para fins didáticos iremos utilizar como exemplo Colin Powell como 0 e Tony Blair como 1.

```
from distutils.dir_util import copy_tree
copy_tree('lfw/Colin_Powell', 'lfw_dados/0')
copy_tree('lfw/Tony_Blair', 'lfw_dados/1')
print('ok')
```

```
ok
```

Gera os conjuntos de dados

Cria um conjunto de dados de treinamento e outro para validação, sendo que as imagens obtidas estão em um tamanho padrão (250x250).

```
image_size = (250, 250)
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory="lfw_dados", validation_split=0.2, subset="training",
    seed=1337, image_size=image_size, batch_size=32
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory="lfw_dados", validation_split=0.2, subset="validation",
    seed=1337, image_size=image_size, batch_size=32
)
```

```
Found 380 files belonging to 2 classes.
Using 304 files for training.
Found 380 files belonging to 2 classes.
Using 76 files for validation.
```

Visualize os dados

Aqui estão as primeiras 9 imagens do conjunto de dados treinados. Sendo 0 para "Colin_Powell" e 1 para "Tony_Blair".

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```

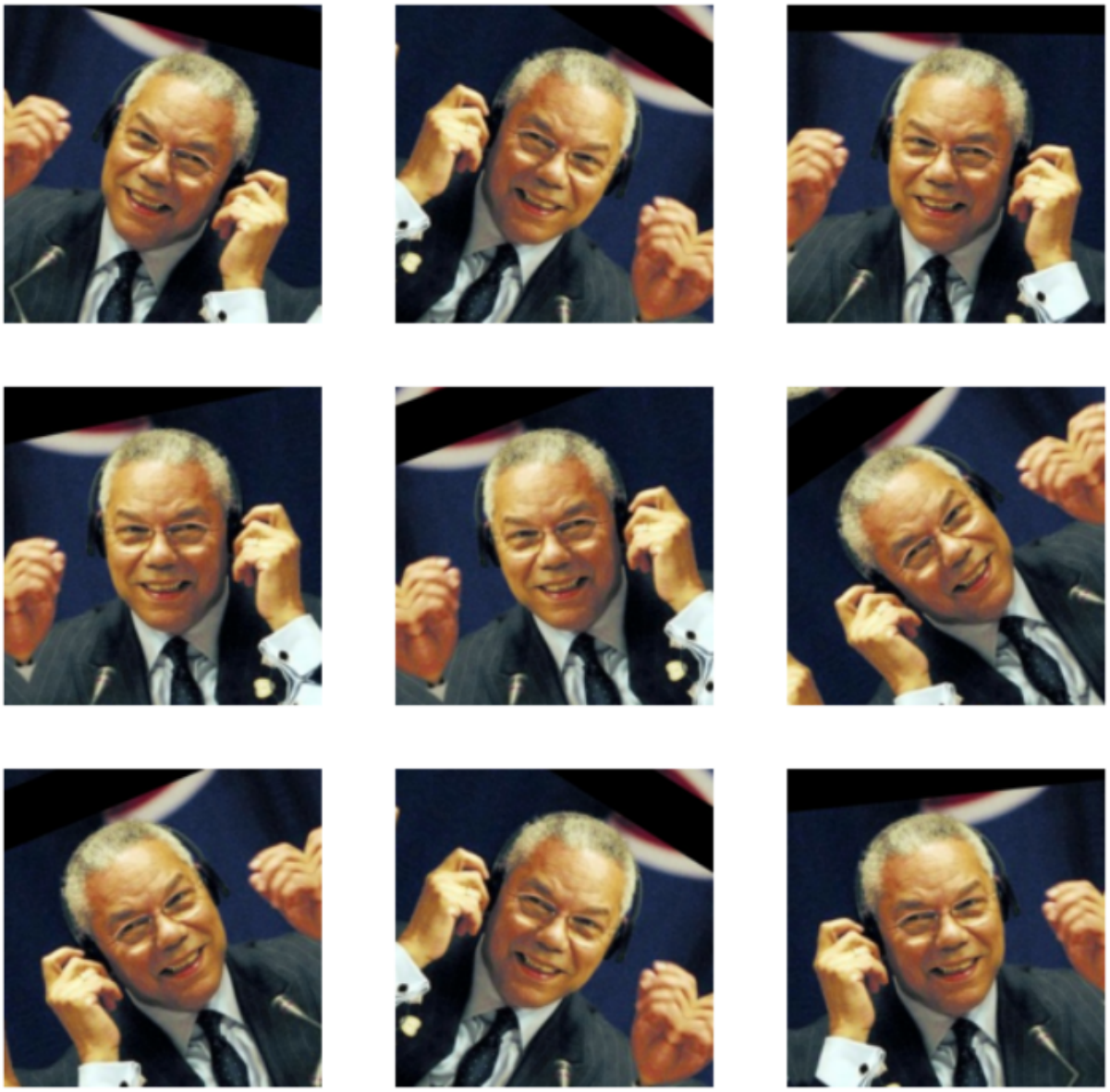
Aumenta artificialmente o condjunto de dados

Uma vez que o conjunto de dados não é muito grande, é recomendado introduzir artificialmente transformações aleatórias, porém realistas, como inversão horizontal aleatória ou pequenas rotações. Desta forma ajuda a expor o modelo a diferentes situações do dia a dia, ao mesmo tempo que desacelera o overfliting.

```
data_augmentation = keras.Sequential(
    [layers.RandomFlip("horizontal"), layers.RandomRotation(0.1),]
)
```

Exemplo de imagens introduzadas artificiamente a partir da primeira imagem do conjunto de dados.

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```



Pré-busca os conjuntos de dados visando melhorar a latência e o redimento ao custo de usar mais memória.

```
train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)
```

Cria o modelo

Cria o modelo baseado em uma pequena versão da rede Xception.

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)
```

```
    # Entry block
    x = layers.Rescaling(1.0 / 255)(x)
    x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)



model = make_model(input_shape=image_size + (3,), num_classes=2)

keras.utils.plot_model(model, show_shapes=True)
```

Treine o modelo

O modelo será treinado por 50 épocas.

```
epochs = 50

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

```
Epoch 1/50
10/10 [==============================] - 12s 956ms/step - loss: 0.2526 - accura
cy: 0.9079 - val_loss: 0.7345 - val_accuracy: 0.4474
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: Custom
MaskWarning: Custom mask layers require a config and must override get_config.
When loading, the custom mask layer must be passed to the custom_objects argume
nt.
  category=CustomMaskWarning)
Epoch 2/50
10/10 [==============================] - 10s 919ms/step - loss: 0.1677 - accura
cy: 0.9309 - val_loss: 0.7361 - val_accuracy: 0.4474
...
Epoch 50/50
10/10 [==============================] - 9s 899ms/step - loss: 0.0436 - accurac
y: 0.9836 - val_loss: 0.1723 - val_accuracy: 0.8947
<keras.callbacks.History at 0x7f070bdc8510>
```

Executar inferência de novos dados.

Agora para uma imagem de Colin Powell o modelo preve 99% de chance de ser o Colin.

```
img = keras.preprocessing.image.load_img(
    "lfw/Colin_Powell/Colin_Powell_0003.jpg", target_size=image_size)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis
predictions = model.predict(img_array)
score = predictions[0]
print("A imagem é %.2f%% Colin_Powell e %.2f%% Tony_Blair."
      % (100 * (1 - score), 100 * score))
```

```
A imagem é 99.81% Colin_Powell e 0.19% Tony_Blair.
```

Enquanto uma imagem do Tony Blair, tem 99% de chance de ser o Tony.

```
img2 = keras.preprocessing.image.load_img(
    "lfw/Tony_Blair/Tony_Blair_0013.jpg", target_size=image_size)
img_array2 = keras.preprocessing.image.img_to_array(img2)
img_array2 = tf.expand_dims(img_array2, 0)  # Create batch axis
predictions2 = model.predict(img_array2)
score2 = predictions2[0]
print("A imagem é %.2f%% Colin_Powell e %.2f%% Tony_Blair."
      % (100 * (1 - score2), 100 * score2))
```

```
A imagem é 0.10% Colin_Powell e 99.90% Tony_Blair.
```

5 RESULTADOS E DISCUSSÃO

Os resultados obtidos a partir do Google Colab foram de fato muito satisfatórios, sendo capaz de inferir as imagens treinadas com um grão de precisão e assertividade muito grande. Certamente melhor do que muitos humanos no que tange ao reconhecimento facial, vale destacar que as duas pessoas utilizadas no experimento não são tão diferentes umas das outras, muito pelo contrário, dependendo da iluminação e do ângulo da foto provavelmente nós iremos confundi-los.

Vale destacar ainda que não foi uma tarefa fácil atingir este grau de precisão com as poucas imagens que tínhamos à disposição. Uma vez que começamos nosso experimento por meio do Jupyter Notebook em um computador cujo TensorFlow não estava devidamente configurado para executar na GPU, então os cálculos iniciais não só demoraram muito como estavam dando muitos falsos positivos. Estes por sua vez só foram eliminados deixando o experimento rodar por muitas eras, passo este que só fomos capazes de executar por meio do Google Colab.

CONCLUSÃO

O experimento permitiu mergulharmos um pouco no mundo do Machine Learning e Deep Learning utilizando a api Keras do TensorFlow para realizarmos a classificação de imagens. Lembramos que apesar disso ainda existem muitos outros modelos que não foram aqui explorados. Para mais informações recomenda-se a leitura de todas as referências.

REFERÊNCIA

DATA SCIENCE ACADEMY. 2021. Disponível em: <https://www.datascienceacademy.com.br/path-player?courseid=inteligencia-artificial-fundamentos&unit=inteligencia-artificial-fundamentos_1534427169931_0Unit>. Acesso em: 11 set. 2021.

DEEPLARNINGBOOK. 2021. Disponível em: <https://www.deeplearningbook.com.br/reconhecimento-de-imagens-com-redes-neurais-convolucionais-em-python-parte-3/>. Acesso em: 11 set. 2021.

TOWARDS DATA SCIENCE. 2018. Disponível em: <https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568>. Acesso em: 11 set. 2021.

MEDIUM. 2019. Disponível em: <https://medium.com/analytics-vidhya/image-recognition-using-pre-trained-xception-model-in-5-steps-96ac858f4206>.Acesso em: 11 set. 2021.

KERAS. 2021. Disponível em: <https://keras.io/about/>. Acesso em: 11 set. 2021.