

Redes Neurais Artificiais e suas Principais Aplicações: Image Matching Challenge

Eduardo Albino Gonelli [1]

Matheus Kiss [2]

Rafael Marcos Tomé [3]

Renato Lins da Palma [4]

Resumo

Atualmente, conceitos ligados à Redes Neurais Artificiais estão cada vez mais em pauta, com o surgimento de diversas ferramentas que auxiliam no reconhecimento de imagens, sons, vídeos, etc. Na internet temos disponíveis diversas imagens de monumentos históricos fotografados por usuários. A ideia do projeto presente neste artigo é desenvolver uma rede neural que seja capaz de coletar um conjunto de imagens desses monumentos disponível no Kaggle (plataforma para aprendizado de ciência de dados) e através de processamento de imagens e utilização de modelos de *Deep Learning* para classificação criar uma visão tridimensional mais completa de tais imagens. Os principais conceitos de programação em Python utilizados no projeto são Convolução, Pooling, Resizing, Normalização e Augmentation.

Palavras-chave: Convolução. Pooling. Kaggle. SuperGlue. Tensorflow.

Abstract

Nowadays, concepts related to Artificial Neural Networks are increasingly on the docket, with the emergence of many tools that assist in recognition of images, sounds, videos, etc. On the Internet, we have available many photos of historical monuments taken by users. The idea of the project presented in this article is to develop a neural network that is capable of collecting an ensemble of these monuments images available on Kaggle (platform for data science learning) and by image processing and use of Deep Learning models for classification create a complete tridimensional view of these images. The main concepts of Python programming used on the project are Convolution, Pooling, Resizing, Normalization and Augmentation.

Keywords: Convolution. Pooling. Kaggle. SuperGlue. Tensorflow.

Introdução

Através de imagens que se encontram na Internet, somos capazes de conhecer diversos locais no mundo com monumentos importantes para a história humana, porém fotos bidimensionais muitas vezes não trazem a riqueza de detalhes que uma projeção em 3D proporciona.

A ideia do projeto é fornecer aos usuários maior imersão em suas pesquisas através das imagens tridimensionais, com intuito de facilitar o completo entendimento do local em estudo.

Para a criação de uma rede neural que seja capaz de agrupar um conjunto de imagens de um local e colocá-las em visão tridimensional há um processo a ser seguido, com o intuito de se atingir o melhor resultado possível.

O caminho a ser seguido é:

- Importar o conjunto de imagens, o Dataset. Neste conjunto de imagens existe a base de dados para treinamento e teste;
- Pré-processamento com ajuda de conceitos de *Resizing* e *Augmentation*;

- Extração das features com uso de pooling e convolução;
- Uso de classificadores como o DenseNet, VGG16, VGG19, Inception V3;
- Resultados das classificações e análise de performance.

Desenvolvimento

Considerando que o processo de recriação de uma imagem em três dimensões (3D) possui muitas etapas, dividiu-se o estudo dos algoritmos nas seguintes etapas:

- Configuração do ambiente
- Download e carregamento do dataset
- Treinamento de um modelo com o *dataset* do kaggle;
- Extração dos dados necessários para a recomposição da imagem;
- Geração da imagem a partir dos dados extraídos.

A configuração do ambiente se deu conforme informações extraídas do github da API oficial do Kaggle (KAGGLE, 2023).

O bloco de código responsável pela configuração do ambiente segue abaixo:

```
# upload do arquivo kaggle.json com a chave da API para uso do dataset
from google.colab import files
files.upload()
```

```
# criação do diretório para os arquivos do kaggle
! mkdir ~/.kaggle
# cópia do arquivo json para a pasta criada
! mv kaggle.json ~/.kaggle/
# segurança para visualização do arquivo json
# (4 e 2 "6" somente proprietário pode ler e modificar, 00 os demais colaboradores não têm acesso)
! chmod 600 ~/.kaggle/kaggle.json
# se deu certo, permite a visualização das listas de dataset
! kaggle datasets list
```

Com a configuração acima é possível exibir as listas de datasets do Kaggle e conferir, assim, que o ambiente foi configurado corretamente através do comando

"!kaggle datasets list". O resultado do comando pode ser observado na imagem abaixo:

Figura 1: Resultado da listagem de datasets.

ref	title	size	lastUpdated	downloadCount	voteCount	usabilityRating
salvatorestellis/spotify-and-youtube	Spotify and Youtube	9MB	2023-03-20 15:43:25	8828	319	1.0
pbb0x/country-gdp	Country GDP	7KB	2023-04-07 06:47:36	1308	38	1.0
erdenetaha/cancer-data	Cancer Data	49KB	2023-03-22 07:57:00	4033	94	1.0
omartorres25/honda-data	Honda Cars Data	184KB	2023-03-28 04:19:11	1446	34	1.0
lokesparab/amazon-products-dataset	Amazon Products Sales Dataset 2023	80MB	2023-03-26 10:45:19	4473	98	1.0
ulrikhygepedersen/fastfood-nutrition	Fastfood Nutrition	12KB	2023-03-21 10:02:41	3619	70	1.0
pbb0x/country-rates	Country Interest Rates	18KB	2023-04-07 06:28:26	1335	39	1.0
armabcdu/ai-science-salaries-2023	Data Science Salaries 2023	250KB	2023-03-13 00:57:16	23184	62	1.0
kaptsovalexander/nvidia-and-intel-asus-msi-share-prices	NVIDIA, AMD, Intel, ASUS, MSI share prices (GPU)	902KB	2023-04-13 12:15:18	569	37	1.0
riklattisak/student-performance-in-mathematics	Student performance prediction	9KB	2023-03-12 04:32:56	9435	202	1.0
ashishraut64/internet-users	Global Internet users	163KB	2023-03-29 12:25:13	2414	59	1.0
riklattisak/smart-watch-prices	Smart Watch prices	5KB	2023-04-12 06:04:23	705	23	1.0
arnabchakl/popular-video-games-1980-2023	Popular Video Games 1980 - 2023	1MB	2023-03-23 16:16:51	4014	111	1.0
tayyarhusain/best-selling-game-companies-of-all-time	Global Emissions.	31KB	2023-03-27 09:02:51	2920	58	1.0
ashishraut64/global-methane-emissions	Best Selling Gaming Consoles Dataset	1KB	2023-04-13 10:59:00	1057	36	1.0
ashishraut64/global-methane-emissions	Smartphone Specifications and Prices in India	300KB	2023-04-13 00:51:13	535	23	1.0
ashishraut64/global-methane-emissions	Cryptocurrency Prices (from start to 2023)	140KB	2023-04-09 06:07:57	886	31	1.0
richardson/the-world-university-rankings-2011-2023	THE World University Rankings 2011-2023	1MB	2023-04-03 12:43:37	2180	49	1.0
dgoenrique/netflix-movies-and-tv-shows	Netflix Movies and TV Shows	2MB	2023-03-13 18:49:00	4146	99	1.0
dansbecker/melbourne-housing-snapshot	Melbourne Housing Snapshot	451KB	2018-06-05 12:52:24	113320	1248	0.7058824

Fonte: dos autores.

Após a configuração do ambiente, torna-se possível o download do dataset através do código:

```
! kaggle competitions download -c image-matching-challenge-2022
```

Figura 2: Download do dataset concluído.

```
Downloading image-matching-challenge-2022.zip to /content
100% 2.31G/2.31G [00:26<00:00, 107MB/s]
100% 2.31G/2.31G [00:26<00:00, 92.8MB/s]
```

Fonte: dos autores.

Concluído o download, faz-se necessária a descompactação dos arquivos.

Abaixo, o código realiza a descompactação do arquivo do kaggle "image-matching-challenge-2022.zip".

```
# descompacta o dataset na raiz da pasta local
!unzip image-matching-challenge-2022.zip
```

A extração dos arquivos já disponibiliza os dados para serem carregados no dataset.

Na primeira parte deste trabalho o objetivo é adequar o exemplo disponibilizado na documentação oficial do tensorflow com o dataset do kaggle, para isso fez-se necessária a importação das bibliotecas básicas de treinamento utilizando tensorflow:

```
# importa as bibliotecas essenciais para trabalhar com tensorflow
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

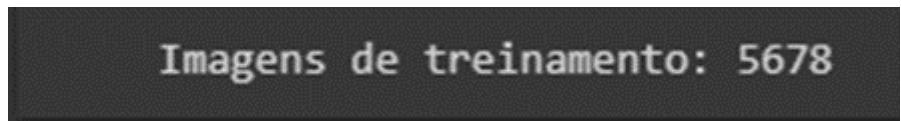
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Configura-se então o caminho das imagens e, para verificar se as informações foram encontradas corretamente, faz-se a contagem das imagens da pasta de treinamento e a pasta de testes:

```
import pathlib
train_dir = pathlib.Path('/content/train')
test_dir = pathlib.Path('/content/test_images')

image_count = len(list(train_dir.glob('*/*.jpg')))
print(f"Imagens de treinamento: {image_count}")
```

Figura 3: Imagens de treinamento.



Imagens de treinamento: 5678

Fonte: dos autores.

```
test_image_count = len(list(test_dir.glob('*/*.png')))
print(f"Imagens de teste: {test_image_count}")
```

Figura 4: Teste do caminho das imagens de teste.

```
Imagens de teste: 6
```

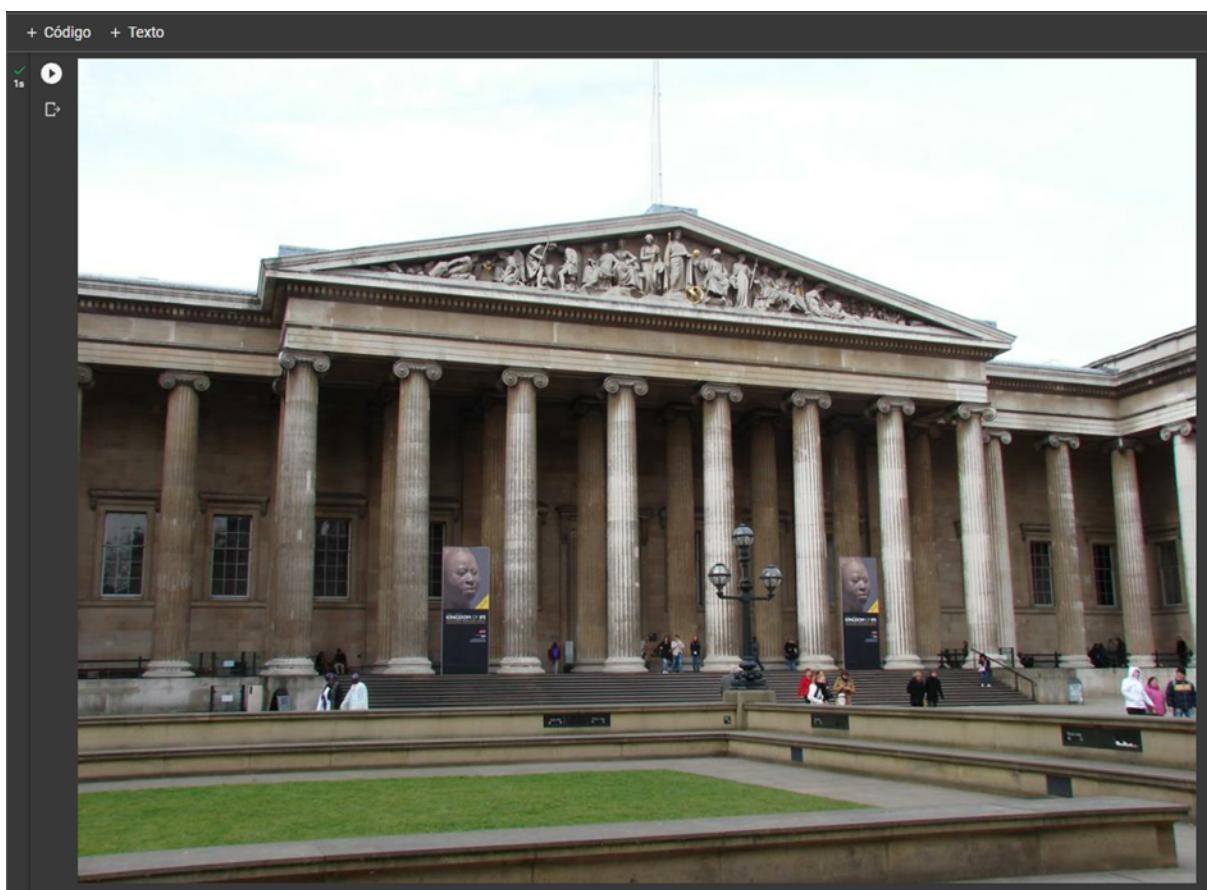
Fonte: dos autores.

Realizou-se então a verificação se as imagens foram encontradas exibindo uma das imagens através de sua localização:

```
locality = 'british_museum'  
locality_path = train_dir / locality  
locality_images = list(locality_path.glob('images/*.jpg'))
```

```
image_to_show = locality_images[0]  
PIL.Image.open(str(image_to_show))
```

Figura 5: Plot de imagem de teste.



Fonte: dos autores.

O próximo passo consiste na configuração dos dados para treinamento. Inicialmente definiu-se 32 batch_size, e redimensionamento das imagens para 180 pixels de altura e 180 pixels de largura. Separou-se também uma quantidade de 20% das imagens para validação. O código de implementação segue abaixo:

```
batch_size = 32
img_height = 180
img_width = 180
```

O uso do batch_size permite dividir o problema em partes menores, assim, evita-se o estouro de memória na geração da pilha (stack-overflow). Ali et.al. (2021) afirma que determinar o treinamento e avaliação de tamanhos de batches é um processo tedioso e não muito útil quando se tem um conjunto grande de experimentos heterogêneos para rodar. Desta forma, entende-se que a variação do valor do batch_size pode influenciar diretamente a qualidade e tempo de treinamento mas experimentar vários formatos pode significar aumento do tempo de treinamento.

```
# cria o conjunto de dados para treinamento e validação do treinamento
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    labels='inferred',
    label_mode='categorical',
    class_names=None,
    color_mode='rgb',
    batch_size=batch_size,
    image_size=(img_height, img_width),
    shuffle=True,
    seed=123,
    validation_split=0.2,
    subset="training",
    interpolation='bilinear',
    follow_links=False,
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    labels='inferred',
```

```

label_mode='categorical',
class_names=None,
color_mode='rgb',
batch_size=batch_size,
image_size=(img_height, img_width),
shuffle=True,
seed=123,
validation_split=0.2,
subset="validation",
interpolation='bilinear',
follow_links=False,
)

class_names = train_ds.class_names
print(class_names)

```

A imagem abaixo ilustra as classes e arquivos encontrados por classe:

Figura 6: Imagens e classes.

```

Found 5678 files belonging to 16 classes.
Using 4543 files for training.
Found 5678 files belonging to 16 classes.
Using 1135 files for validation.
['brandenburg_gate', 'british_museum', 'buckingham_palace', 'colosseum_exterior', 'gr

```

Fonte: dos autores.

Realizou-se uma última conferência para saber se todos os dados foram realmente reconhecidos na importação, plotando-se o nome das 16 classes e uma imagem de cada classe. O código utilizado foi o seguinte:

```

num_classes = len(class_names)
num_cols = 4
num_rows = num_classes // num_cols + (1 if num_classes % num_cols != 0
else 0)

plt.figure(figsize=(10, 10))

images_shown = set()

```

```

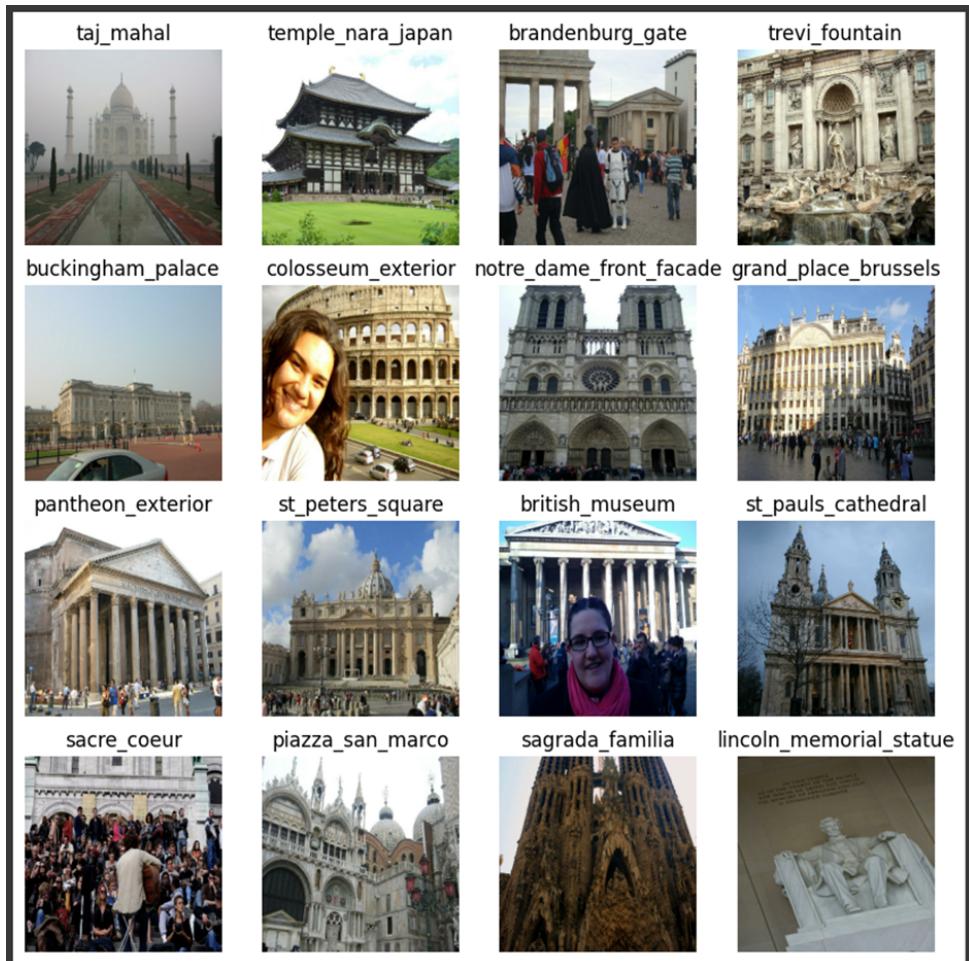
for images, labels in train_ds:
    for i in range(images.shape[0]):
        if len(images_shown) == num_classes:
            break

        label = np.argmax(labels[i])
        if label not in images_shown:
            images_shown.add(label)
            ax = plt.subplot(num_rows, num_cols, len(images_shown))
            plt.imshow(images[i].numpy().astype("uint8"))
            plt.title(class_names[label])
            plt.axis("off")

    if len(images_shown) == num_classes:
        break

```

Figura 7: Plot das 16 classes com uma imagem por classe



Fonte: dos autores.

Feita a importação e conferidos os dados, o próximo passo se deu configurando os caches e normalizando a camada de treinamento. O código responsável pela normalização segue abaixo:

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
normalization_layer = layers.Rescaling(1./255)
```

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

O modelo utilizado é Convolucional com uma camada de entrada, três camadas convolucionais, três camadas de Maxpooling2D, uma camada flatten, uma camada de ativação e uma camada de saída, com ativação softmax. O código segue abaixo:

```
num_classes = len(class_names)

model = Sequential([
    # camada de entrada
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    # camadas ocultas
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
```

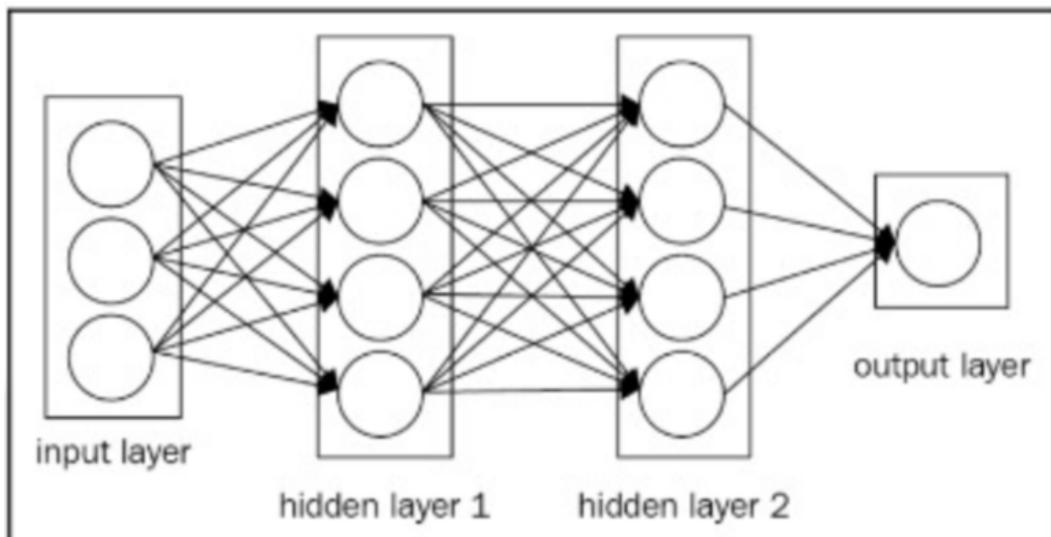
```
# camada de saída  
    layers.Dense(num_classes, activation='softmax')  
])
```

```
# compila o modelo com o otimizador adam  
model.compile(optimizer='adam',  
  
loss=tf.keras.losses.CategoricalCrossentropy(from_logits=False),  
metrics=['accuracy'])
```

Camadas convolucionais, conhecidas como CNN ou ConvNets, são similares a redes neurais regulares, feitas de neurônios com pesos que permitem aprender a partir de dados. Cada neurônio recebe informações de entradas e realiza a operação de produto escalar, podendo ainda usar uma função não linear. (SEWAK et. al., 2018).

As CNNs fazem parte das camadas ocultas, geralmente seguidas de uma camada de pooling e depois a camada de saída. Várias camadas ocultas podem fazer parte de uma rede neural e seu número ideal varia com relação ao problema que se pretende solucionar. A imagem abaixo exemplifica a camada de entrada, camadas ocultas e camada de saída:

Figura 8: Exemplo da estrutura de uma rede neural convolucional.



Fonte: Sewak et. al. (2018)

A camada de MaxPooling2D funciona similar a camada CNN, em que é selecionada a intensidade máxima de um valor de pixel em uma camada de tamanho 2x2. A única diferença entre a camada convolucional e a camada de max-pooling, de acordo com Millstein (2020) é que a função aplicada ao kernel da imagem e à janela da imagem não é linear.

Treinou-se então a rede com a configuração básica com apenas 10 épocas:

```
epochs=10
history = model.fit(
    train_ds,
    validation_data = val_ds,
    epochs = epochs
)
```

Figura 9: Treinamento do modelo com 10 épocas.

```
Epoch 1/10
142/142 [=====] - 412s 3s/step - loss: 2.2004 - accuracy: 0.3249 - val_loss: 1.5096 - val_accuracy: 0.5665
Epoch 2/10
142/142 [=====] - 396s 3s/step - loss: 1.1196 - accuracy: 0.6672 - val_loss: 1.2351 - val_accuracy: 0.6396
Epoch 3/10
142/142 [=====] - 398s 3s/step - loss: 0.6245 - accuracy: 0.8160 - val_loss: 1.0674 - val_accuracy: 0.7101
Epoch 4/10
142/142 [=====] - 397s 3s/step - loss: 0.2804 - accuracy: 0.9201 - val_loss: 1.3473 - val_accuracy: 0.6987
Epoch 5/10
142/142 [=====] - 408s 3s/step - loss: 0.0936 - accuracy: 0.9747 - val_loss: 1.4065 - val_accuracy: 0.7128
Epoch 6/10
142/142 [=====] - 392s 3s/step - loss: 0.0345 - accuracy: 0.9927 - val_loss: 1.5518 - val_accuracy: 0.7093
Epoch 7/10
142/142 [=====] - 408s 3s/step - loss: 0.0369 - accuracy: 0.9905 - val_loss: 1.7676 - val_accuracy: 0.7848
Epoch 8/10
142/142 [=====] - 389s 3s/step - loss: 0.0236 - accuracy: 0.9958 - val_loss: 1.7984 - val_accuracy: 0.6996
Epoch 9/10
142/142 [=====] - 399s 3s/step - loss: 0.0182 - accuracy: 0.9960 - val_loss: 1.9676 - val_accuracy: 0.7198
Epoch 10/10
142/142 [=====] - 399s 3s/step - loss: 0.0275 - accuracy: 0.9923 - val_loss: 1.8729 - val_accuracy: 0.7339
```

Fonte: dos autores.

Para visualização do modelo utilizou-se a biblioteca matplotlib para exibir os gráficos de acurácia de treinamento e de acurácia de validação. O código utilizado para verificação foi:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)
```

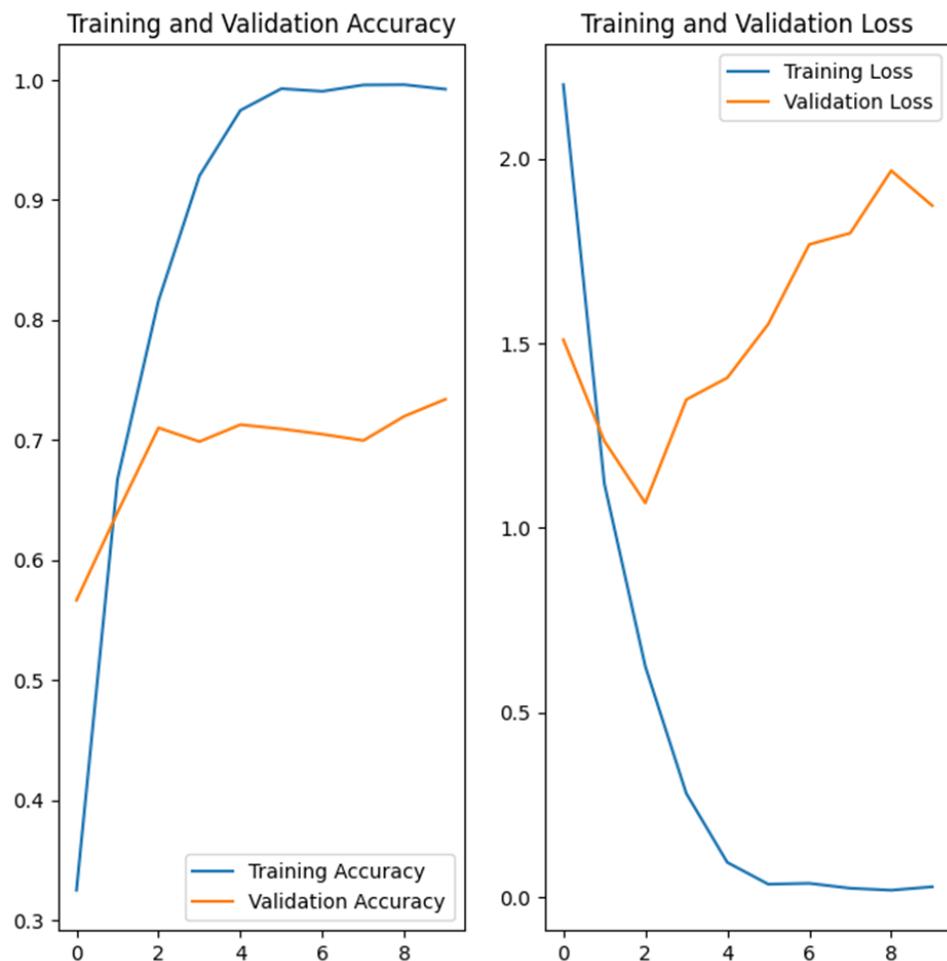
```

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

```

Gerando o seguinte resultado:

Figura 10: Acurácia de treino, acurácia de validação e perdas.



Fonte: dos autores.

Observa-se no gráfico acima a disparidade da acurácia de treinamento e de validação e, além disso, a baixa incidência de falhas no treinamento, porém, alta incidência de falhas na validação. Trata-se claramente do fenômeno de "Overfitting".

Overfitting ocorre quando o erro de treinamento é baixo enquanto o erro de teste é alto. Underfitting ocorre quando os erros são altos no treinamento e na validação, sendo que o ideal (Good fit) é quando os erros para o treinamento e validação são relativamente pequenos. (GU et. al., 2016).

Tabela 1: MAD (mean absolute difference - diferença média absoluta) entre o treinamento e validação.

	Overfitting	Underfitting	Good Fit
MAD	$MAD_{training} < MAD_{testing}$	Highest $MAD_{training}$ and highest $MAD_{testing}$	Lowest $MAD_{training}$ and lowest $MAD_{testing}$
SD	High	High	Relatively low

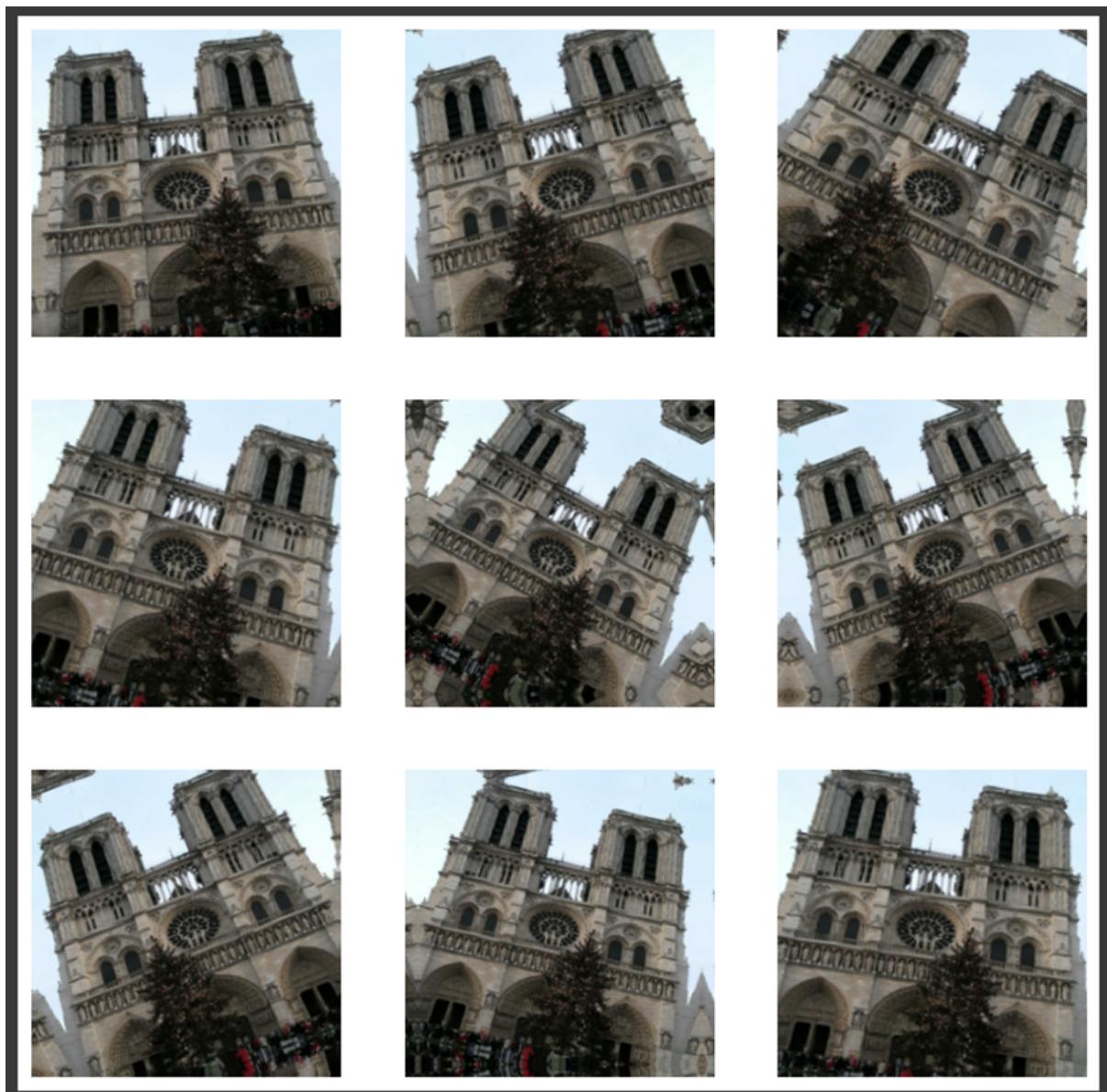
Fonte: Gu et. al. (2016)

Na tentativa de melhorar a acurácia de validação, aumentou-se o tamanho da imagem para 256x256 pixels e utilizou-se a técnica de "Data Augmentation". O código para o aumento do dataset levou em consideração a inversão da imagem no eixo horizontal, rotação randomizada e ampliação randomizada. Abaixo o código utilizado e o resultado de uma imagem:

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal",
                           input_shape=(img_height,
                                       img_width,
                                       3)),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
    ]
)
```

```
# exibe uma imagem aplicando o data_augmentation nele.
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")
```

Figura 11: Imagem com Data Augmentation.



Fonte: dos autores.

Adaptou-se então o modelo para o treinamento com Data Augmentation:

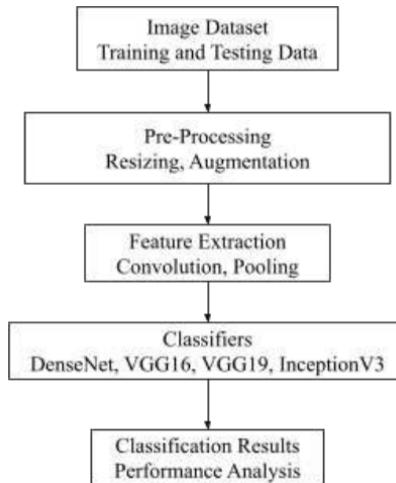
```
model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])
```

```
# compila o modelo com o otimizador adam
model.compile(optimizer='adam',
               loss=tf.keras.losses.CategoricalCrossentropy(from_logits=False),
               metrics=['accuracy'])
```

Tripathi (2021) afirma que a técnica de data augmentation é usada para expandir e treinar o dataset, com isso melhorando a performance e permitindo generalizar o modelo.

Observa-se que o bloco de códigos acima segue o sistema proposto por Tripathi (2021), que consiste no Image Dataset (treinamento e teste dos dados), Pre-Processing (Redimensionamento e Ampliação), Feature Extraction (Convolução e Pooling), Classificação (DenseNet, VGG16, VGG19, Inception V3) e Classification Results (Análise de Performance), conforme imagem abaixo:

Figura 12: Sistema proposto por Tripathi (2021).



Fonte: Tripathi (2021)

Os parâmetros com os dados anteriores beiravam três milhões de parâmetros. Com o ajuste com Data Augmentation e aumento do tamanho da imagem, a quantidade de parâmetros beira a oito milhões e quinhentos mil.

```
model.summary()
```

Figura 13: Sumário dos parâmetros.

Model: "sequential_8"		
Layer (type)	Output Shape	Param #
sequential_7 (Sequential)	(None, 256, 256, 3)	0
rescaling_8 (Rescaling)	(None, 256, 256, 3)	0
conv2d_18 (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d_18 (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_19 (Conv2D)	(None, 128, 128, 32)	4640
max_pooling2d_19 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_20 (Conv2D)	(None, 64, 64, 64)	18496
max_pooling2d_20 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_2 (Dropout)	(None, 32, 32, 64)	0
flatten_6 (Flatten)	(None, 65536)	0
dense_12 (Dense)	(None, 128)	8388736
dense_13 (Dense)	(None, 16)	2064
<hr/>		
Total params: 8,414,384		
Trainable params: 8,414,384		
Non-trainable params: 0		

Fonte: dos autores.

Realizou-se novamente o treinamento do modelo com quize épocas e chegou-se a um resultado mais satisfatório, conforme pode ser observado abaixo:

```
epochs = 15
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

O treinamento gerou uma acurácia de treinamento de 0.89, enquanto a acurácia de validação ficou em 0.77, como pode ser observado no gráfico abaixo:

Figura 14: Gráfico de acurácia e perda.



Fonte: dos autores.

Realizando teste com uma imagem da pasta branderbunr_gate, retornou que a imagem pertence a Branderbung Gate com 15.29% de certeza.

Realizando teste com uma imagem do Gran Place Brussels, extraída da internet, retornou que a imagem pertence ao Palácio de Buckingham com 15.34% de certeza.

Ainda, em outro teste utilizando uma das imagens de treinamento da Trevi Fountain, o algoritmo foi capaz de identificar de onde pertence a imagem com confiança de 15.31%.

Com uma imagem da Fontana di Trevi, com baixa resolução, outro equívoco, indicando Notre Dame Front Facade com 12.47% de confiança. Já com uma imagem da Fontana di Trevi com alta resolução (4k), apontou corretamente como a Trevi Fountain com 15.22% de confiança.

Isto indica que, ou o treinamento foi insuficiente, ou a qualidade das imagens para treinamento precisa de mais informações (tamanho maior que 256x256), ou ainda que as imagens precisam estar em alta qualidade para a comparação.

Estudo do 3D - Point-e

Nesta etapa do processo o reconhecimento de imagens já foi comprovado através do uso do tensorflow e keras, com resultado que, mesmo que não tenham o máximo de confiança, ainda assim realizam o reconhecimento dos locais quando a imagem apresentada possui qualidade suficiente.

Estuda-se então, as formas de criação de imagens 3D a partir das imagens de origem.

Como primeira alternativa, estudou-se a possibilidade de gerar as imagens através do sistema de Machine Learning Point-e, da OpenAI.

De acordo com Nichol et. al (2022), o objetivo do Point-e é utilizar os benefícios da geração textual de imagens com a geração de modelos 3D com base nas imagens.

Seguiu-se as instruções disponibilizadas no github da página oficial do Point-e para extração das informações de seu uso e implantação. O código utilizado foi:

```
# versão do six necessária para utilização do point-e
!pip install six --upgrade
```

```
# instalação do point-e
!pip install git+https://github.com/openai/point-e.git
```

```
# import das bibliotecas necessárias
import torch
from tqdm.auto import tqdm

from      point_e.diffusion.configs      import      DIFFUSION_CONFIGS,
diffusion_from_config
from point_e.diffusion.sampler import PointCloudSampler
from point_e.models.download import load_checkpoint
from point_e.models.configs import MODEL_CONFIGS, model_from_config
from point_e.util.plotting import plot_point_cloud
```

```
# configuração do dispositivo
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

print('creating base model...')
base_name = 'base40M-textvec'
base_model = model_from_config(MODEL_CONFIGS[base_name], device)
base_model.eval()

base_diffusion = diffusion_from_config(DIFFUSION_CONFIGS[base_name])

print('creating upsample model...')
upsampler_model = model_from_config(MODEL_CONFIGS['upsample'], device)
upsampler_model.eval()
```

```
upsampler_diffusion =  
diffusion_from_config(DIFFUSION_CONFIGS['upsample'])  
  
print('downloading base checkpoint...')  
base_model.load_state_dict(load_checkpoint(base_name, device))  
  
print('downloading upsampler checkpoint...')  
upsampler_model.load_state_dict(load_checkpoint('upsample', device))
```

```
# configuração do sampler  
sampler = PointCloudSampler(  
    device=device,  
    models=[base_model, upsampler_model],  
    diffusions=[base_diffusion, upsampler_diffusion],  
    num_points=[1024, 4096 - 1024],  
    aux_channels=['R', 'G', 'B'],  
    guidance_scale=[3.0, 0.0],  
    model_kwargs_key_filter=('texts', ''), # Do not condition the  
upsampler at all  
)
```

```
# recebe o prompt para a geração da imagem  
prompt = 'a red motorcycle'  
  
# produz o modelo com base no prompt fornecido  
samples = None  
for x in tqdm(sampler.sample_batch_progressive(batch_size=1,  
model_kwargs=dict(texts=[prompt])):  
    samples = x
```

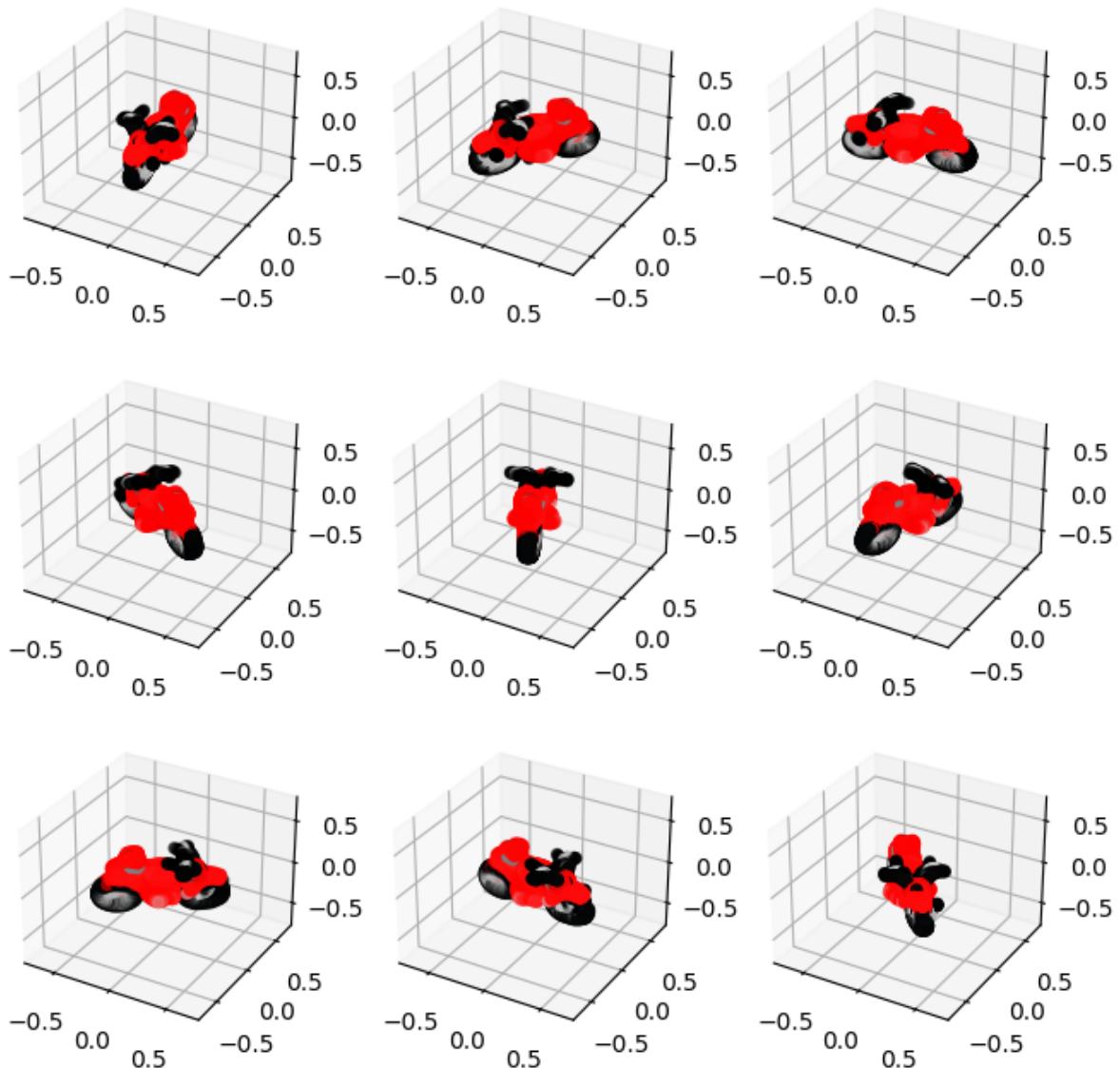
```
# renderiza o prompt no colab  
pc = sampler.output_to_point_clouds(samples)[0]  
fig = plot_point_cloud(pc, grid_size=3, fixed_bounds=(-0.75, -0.75,  
-0.75), (0.75, 0.75, 0.75)))
```

Observou-se que na criação do modelo utiliza-se um prompt de texto e a imagem gerada é produzida com vários pontos com volume na cena. A imagem

gerada do prompt acima, indicado para testes pela página oficial do Point-e, gerou a imagem abaixo:

O processo é realizado primeiro produzindo uma amostra em uma imagem usando o modelo text-to-image, depois produzindo uma amostra 3D condicionada à amostra de imagem gerada. (NICHOL et.al, 2022)

Figura 15: Imagem gerada através do prompt “a red motorcycle”.



Fonte: dos autores.

Constata-se, portanto, que tal solução não atende a geração de imagens 3D a partir de fotogrametria, partindo-se para outros estudos.

Estudo do 3D - SuperGlue

Pesquisando-se desafios semelhantes a este na plataforma Kaggle (GOOGLE, 2023), percebe-se que alguns participantes utilizaram o modelo pré-treinado SuperGlue (MAGIC LEAP, 2020).

Tal modelo permite a detecção de pontos de referências entre imagens fazendo correlação entre suas características, extraíndo assim informações importantes para a geração do modelo 3D.

A função do SuperGlue é gerar correspondências entre pontos em imagens para estimar uma estrutura 3D e posicionamento de câmera em uma visão computacional geométrica (SARLIN et. al., 2020). Gerados estes parâmetros, essas informações podem ser passadas para sistemas como *Simultaneous Localization and Mapping* (SLAM) e *Structure-from-Motion* (SfM). Estes programas decompõe o problema em extração de feature visual e ajustamento ou estimativa de pose, e o SuperGlue entra domo *middle-end* neste processo. (SARLIN et. al., 2020)

Devido à limitação da plataforma Kaggle, foi necessário configurar um ambiente local para o estudo da plataforma. O editor de texto utilizado foi o Visual Studio Code, rodando no ambiente “venv”. A versão do Python foi a 11.

Realizados downloads necessários do dataset do Image Matching Challenge 2022 e do SuperGlue analisou-se o conteúdo e, após várias tentativas de adequar o código para o dataset de imagens, verificou-se que o SuperGlue já possui um prompt para a leitura e geração das imagens com as devidas referências.

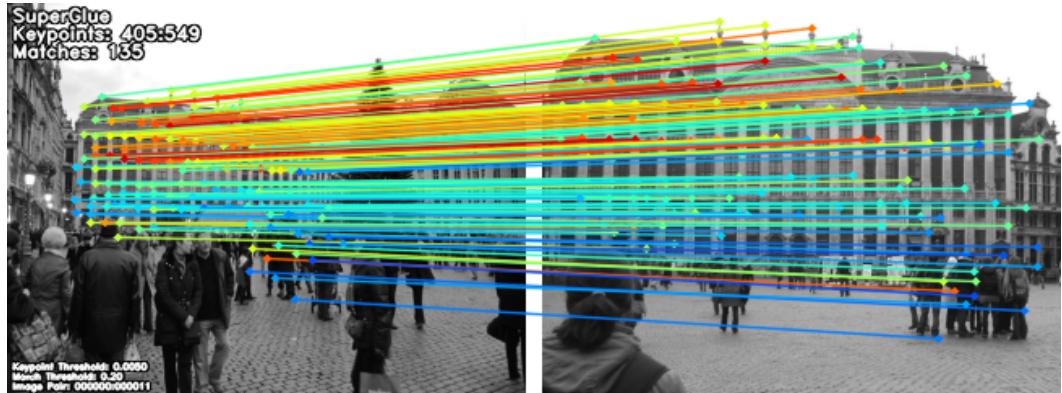
Passou-se o prompt então de algumas pastas como exemplo abaixo:

```
python ./demo_superglue.py --input imc2022/train/grand_place_brussels/images/  
--output_dir dump_demo_sequence2 --resize 320 240 --no_display
```

O prompt recebe as imagens de origem localizadas dentro de cada localidade no dataset do Kaggle, realiza a análise com o modelo pré-treinado e faz um despejo

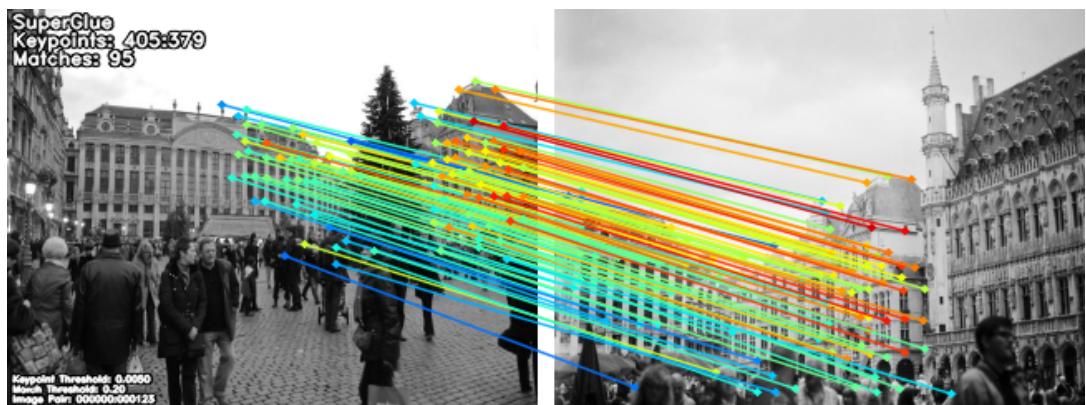
de pares de imagens com as características em comum encontradas entre elas. Abaixo seguem exemplos das localidades “grand_place_brussels” e “sacre_coeur”.

Figura 16: 135 ocorrências em par de imagens de grand_place_brussels:



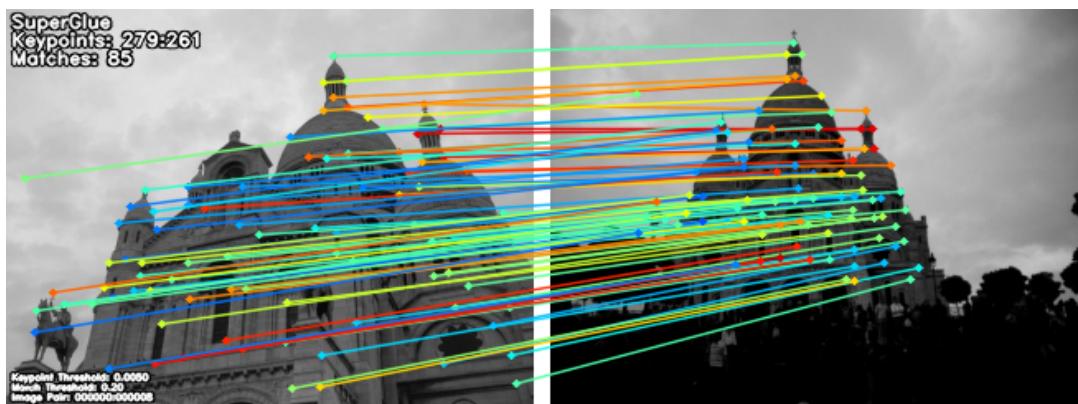
Fonte: dos autores.

Figura 17: 95 ocorrências em pares de imagens de grand_place_brussels.



Fonte: dos autores.

Figura 18: Ocorrências de pares de imagens de sacre_coeur.



Fonte: dos autores.

Figura 19: Ocorrências de pares de imagens de sacre_coeur.



Fonte: dos autores.

Esta, porém, é apenas uma das etapas da geração de uma imagem em 3D, sendo os próximos passos:

- Extração das correspondências entre os pares de imagens em um arquivo json;
- Uso de uma biblioteca para criação de modelo 3D (SfM, SLAM ou COLMAP);
- Renderização do modelo em 3D com a criação de um GIF exemplificando sua construção.

Considerações Finais

Na aplicação do reconhecimento de imagens utilizando redes convolucionais, o baixo número de treinamento se deu pelo curto espaço de tempo entre o desenvolvimento e a entrega desta obra, entretanto, aplicando-se mais épocas no treinamento, ajustando a quantidade de camadas de entrada e saída e aplicando mais técnicas de data augmentation é possível aumentar consideravelmente o resultado do treinamento.

Dada a limitação de conhecimento dos autores e dificuldade de implantação das bibliotecas necessárias para o desenvolvimento, trabalhou-se nesta obra apenas as etapas de reconhecimento de imagens com tensorflow e extração de características das imagens com o SuperGlue.

Mais estudos se fazem necessários para real entendimento da biblioteca SuperGlue e extração das informações em 3D para posterior uso de bibliotecas 3D para conversão dessas informações em dados úteis para a geração de uma imagem em 3D, podendo até mesmo permitir a manipulação através de interações com o teclado do objeto gerado.

Referências

ALI, Mehdi; BERRENDORF, Max. HOYT, Charles Tapley; VERMUE, Laurent; SHARIFZADEH, Sahand; TRESP, Volker; LEHMANN, Jens. **PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings**. Journal of Machine Learning, 2021. Disponível em:
<https://dl.acm.org/doi/pdf/10.5555/3546258.3546340>. Acesso em: 23 abr. 2023.

GU, Yingxin; WYLIE, Bruce K.; BOYTE, Stephen P.; PICOTTE, Joshua; HOWARD, Daniel M. SMITH, Kelcy; NELSON, Kurtis J.. **An Optimal Sample Data Usage Strategy to Minimize Overfitting and Underfitting Effects in Regression Tree Model Based on Remotely-Sended Data**. MDPI, 2016. doi:10.3390/rs8110943.

KAGGLE. **Kaggle**. Disponível em: <https://www.kaggle.com/>. Acesso em: 23 abr. 2023.

MAGIC LEAP. **SuperGlue Inference and Evaluation Demo Script**. Disponível em: <https://github.com/magicleap/SuperGluePretrainedNetwork>. Acesso em 23 abr. 2023.

MILLSTEIN, Frank. **Convolutional Neural Networks In Python: Beginner's Guide To Convolutional Neural Networks In Python**. ISBN-10: 1986264025. Createspace Independent Publishing Platform (2018).

NICHOL, Alex; JUN, Heewoo; DHARIWAL, Prafulla; MISHKIN, Pamela; CHEN, Mark. **Point-E: A System for Generating 3D Point Clouds from Complex Prompts**. arXiv, 2022. Disponível em: <https://arxiv.org/pdf/2212.08751.pdf>. Acesso em: 23 abr. 2023.

SARLIN, Paul-Edouard; DETONE, Daniel; MALISIEWICZ, Tomasz; RABINOVICH, Andrew; ZURICH, ETH; MAGIC LEAP INC. **SuperGlue: Learning Feature Matching with Graph Neural Networks**. arXiv, 2020. Disponível em: <https://arxiv.org/pdf/1911.11763.pdf>. Acesso em: 23 abr. 2023.

SEWAK, Mohit; KARIM, Md. Rezaul; PUJARI, Pradeep. **Practical Convolutional Neural Networks**: Implement advanced deep learning models using Python. Packtpub, 2018.

TENSORFLOW. **Carregar e pré-processar imagens**. Disponível em:
https://www.tensorflow.org/tutorials/load_data/images?hl=pt-br. Acesso em 23 abr. 2023.

TRIPATHI, Milan. **Analysis of Convolutional Neural Network based Image Classification Techniques**. Tribhuvan University, 2021. DOI:
<https://doi.org/10.36548/jiip.2021.2.003>

[1] Especialista em Desenvolvimento de Games e Docente no Centro Universitário SENAC - Santo Amaro.

[2] Graduando em Engenharia Mecânica pela Universidade Paulista.

[3] Tecnólogo em Redes de Computadores pelo Centro Universitário SENAC - Santo Amaro.

[4] Engenheiro da Computação pela Universidade São Judas Tadeu.