建立重複圖像查找系統

原創 磐焱 學習與計算機視覺 2022-09-13 23:10 發表於河南

是否要識別重複或重複的圖像？或者計算數據集中每個圖像的副本數？

如果是，這個這篇文章是給你的。

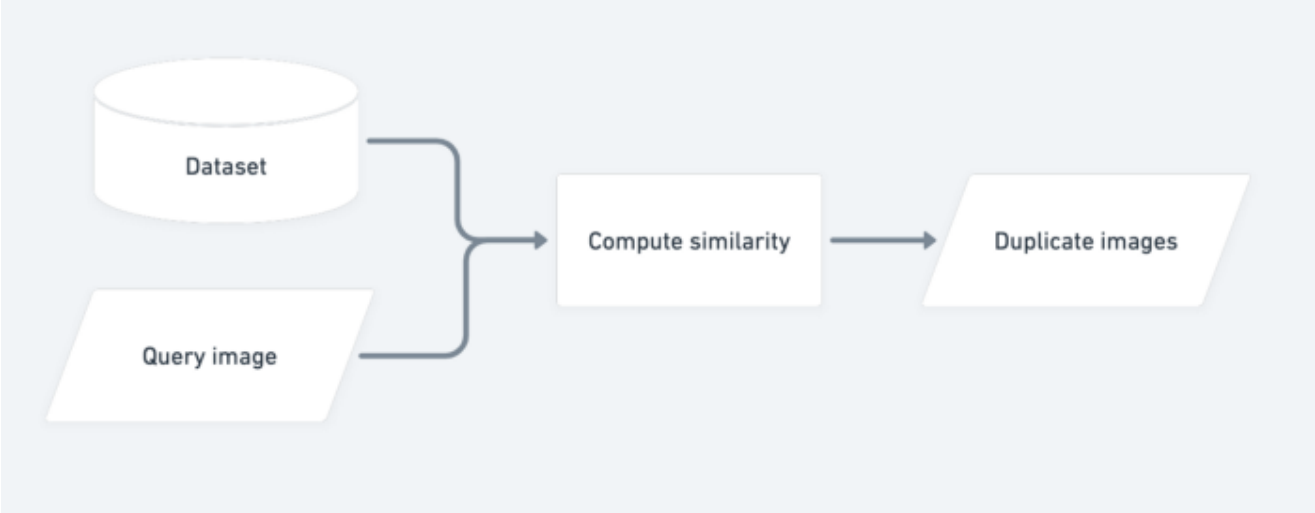
本文的目標有五個方面：

1. 理解重複圖像查找器和基於內容的圖像檢索系統之間的區別
2. 演練比較相似的5種不同方法的概念
3. 學習python中的每一種方法
4. 確定圖像變換對描述算法性能的結果
5. 為在速度和途徑選擇最佳方式選擇適合你的應用的（包括實驗）

基本架構

首先，需要定義一個術語。查詢圖像是用戶輸入以獲取信息的。系統在圖像數據集中搜索了相似的重要圖像，計算它們之間的距離。圖1步驟。

圖1-重複圖像查找系統的基本結構



在第 3 節中，我們將研究這個相似性塊，並探索實現此功能的最常見方法。

重複圖像查找器與基於內容的圖像檢索系統

系統之間的主要區別，並顯示重複圖像（搜索檢測內容和相似的圖像）。這些區域最相似的圖像（圖像3）。

圖2：重複圖像系統的輸入和輸出示例：

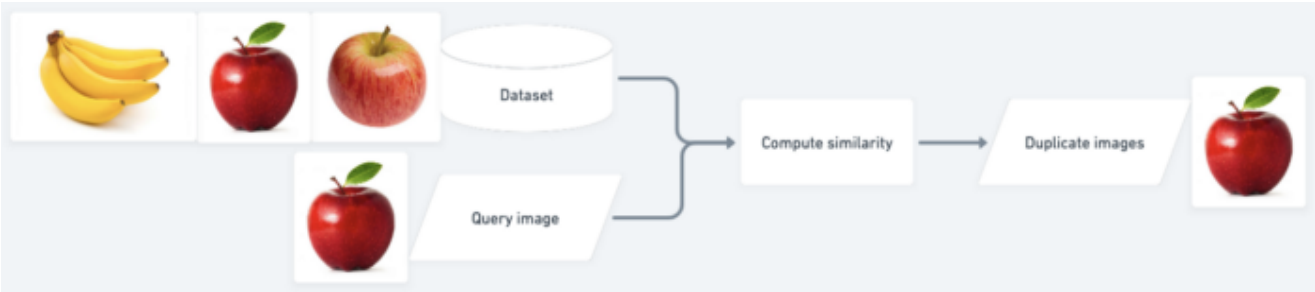
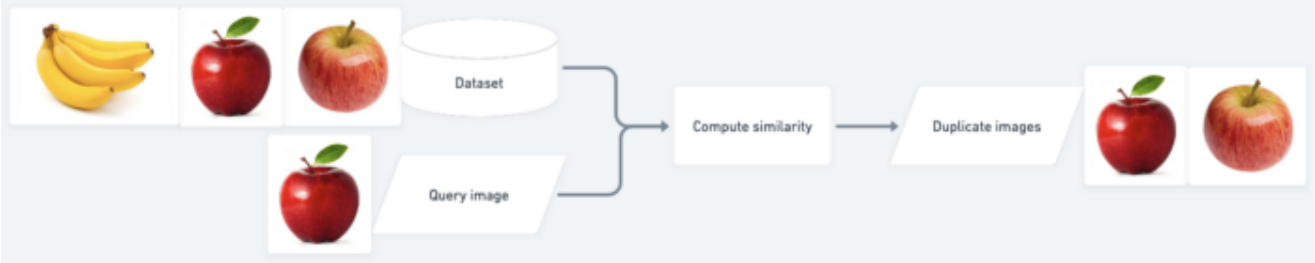


圖3-基於內容的圖像檢索系統的詳細信息示例



請注意，根據內容的圖像檢索系統如何識別蘋果並輸出不同場景的圖像。

相似比較圖像的五種常用方法

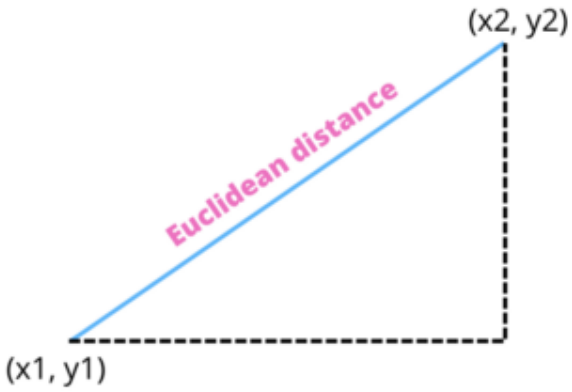
本文將介紹五種主要方法：

- 歐幾里得距離
- 結構相似性狀（SSIM）
- 圖像哈希
- 相似度
- 特徵的相似性（使用CNN）

1、歐幾里得距離

轉到第一種方法，如圖4所示，幾里得距離是2個數據點之間的距離[8]。把它稱為L2範數中距離。

圖4-歐幾里得距離



我們可以將圖像表示為對。

在向量空间中，假设我们有两张图片来比较x=[x1，x2，x3]和y=[y1，y2，y3]。虽然图5显示了一般公式[8]，但图6显示了使用示例。

图5-欧几里得距离的一般公式

$$euclidean\ distance\ (x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

图6-应用欧几里得距离公式的示例

$$euclidean\ distance\ (x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}$$

该方法公式简单明了。

在python中，实现非常简单：

- 实现1：使用Scipy库

```
import numpy as np
from scipy.spatial import distance
from PIL import Image

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image")
# Note: images have to be of equal size

# we need to flatten the image to a 1D vector
value = distance.euclidean(np.array(image1).flatten(), np.array(image2).flatte
```

实现2：使用NumPy的linalg

```
import numpy as np
from PIL import Image

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image")
# Note: images have to be of equal size

# linalg.norm
value = np.linalg.norm(np.array(image1) - np.array(image2))
```

2、结构相似性度量（SSIM）

论文《Image Quality Assessment: From Error Visibility to Structural Similarity》[1]于2004年介绍了SSIM。它计算两个给定图像之间的相似性，得出一个介于0和1之间的值。

除了寻找重复，它的许多应用之一是测量压缩图像如何影响其质量[2]。此外，它还估计了数据传输损耗如何严重降低质量[2]。

作者认为，影响该指数的主要三个因素是亮度、对比度和结构[3]。因此，如果其中一个因素发生变化，度量也会发生变化。

实现如下：

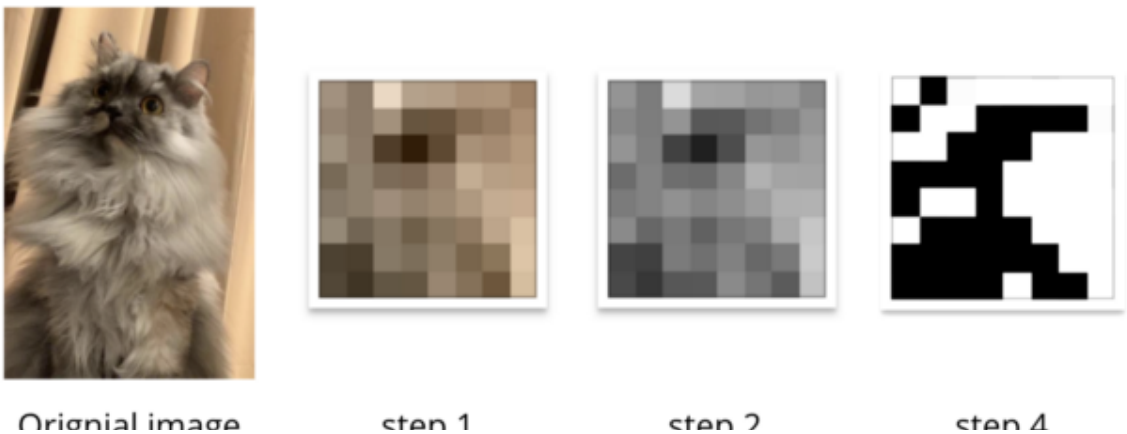
```
from SSIM_PIL import compare_ssim
from PIL import Image

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image")
# Note: images have to be of equal size
value = compare_ssim(image1, image2, GPU=False) # a value of 1 indicates stron
```

3、图像哈希

计算两幅图像之间相似性的另一种方法是图像哈希（也称为数字指纹）。它是为每个图像分配唯一哈希值的过程。然而，该方法对相同的值产生相同的结果。平均哈希是众多哈希类型之一。其工作方式如下[6]。此外，请参阅图7以了解说明。

1. 减小图像大小（例如：8x8）
2. 将其转换为灰度
3. 取均值
4. 将每个像素与平均值进行比较。如果像素高于平均值，则为其指定值1，否则为0
5. 构造哈希

图7-平均哈希步骤



Original Image Step 1 Step 2 Step 4

生成的64位整数可能如下所示：

1011111101100001110001110000111101101111100001110000001100001001

我们可以用不同的方式表示图像。从左上角开始列出0位和1（如上例所示）、左右角等等[6]。

最重要的是，如果我们改变纵横比，增加或减少亮度或对比度，甚至改变图像的颜色，其哈希值将是相同的[7]，这使其成为比较同一图像的最佳方法之一。

比较两幅图像的步骤如下[7]：

1. 构造每个图像的哈希（通过遵循上述5个步骤）

2. 计算汉明距离。零距离表示相同的图像。（下面的代码块更好地解释了这一点）

```
import imagehash
from PIL import Image

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image")
# Note: images DO NOT have to be of equal size

# Construct the hash
hash1 = imagehash.average_hash(image1)
hash2 = imagehash.average_hash(image2)

# Calculate the hamming distance
value = hash1-hash2
```

4、余弦相似性

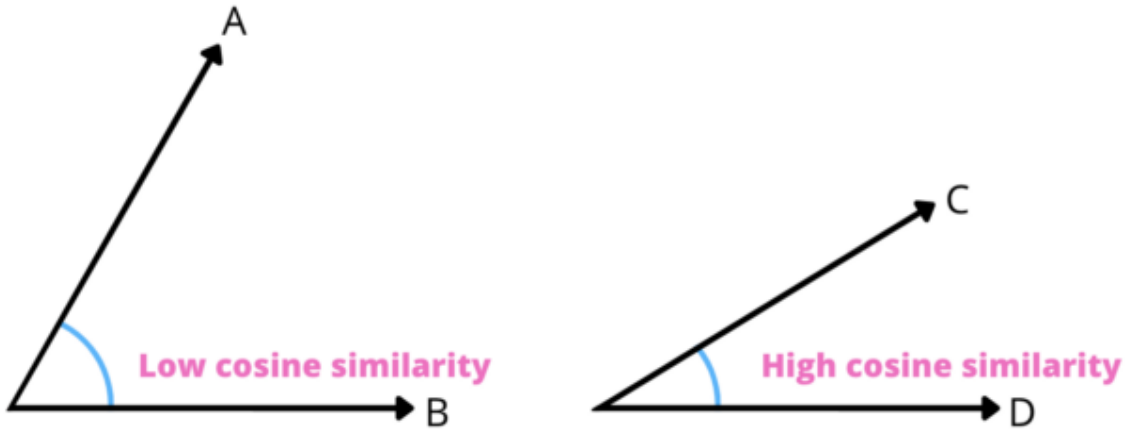
余弦相似性是一种计算两个向量（可以是图像）相似性的方法，方法是取点积并将其除以每个向量的幅值[9]，如下图8所示。

图8-余弦相似方程

$$cosine\ similarity\ (A,B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

随着两个向量之间的角度变小，相似性变大[9]。如图9所示，向量C和B与A和B具有高度相似性，因为它们的角度明显较小。

图9-余弦相似性说明



以下是使用torch计算两个PIL图像的度量的代码。

```
from torch import nn
from PIL import Image
from torchvision import transforms

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image)
# Note: images have to be of equal size

# Transform the images to tensors then flatten them to 1D tensors
image1_tensor = transforms.ToTensor()(image1).reshape(1, -1).squeeze()
image2_tensor = transforms.ToTensor()(image2).reshape(1, -1).squeeze()

cos = nn.CosineSimilarity(dim=0) # dim=0 -> dimension where cosine similarity
value = float(cos(image1_tensor, image2_tensor)) # a value of 1 indicates stro
```

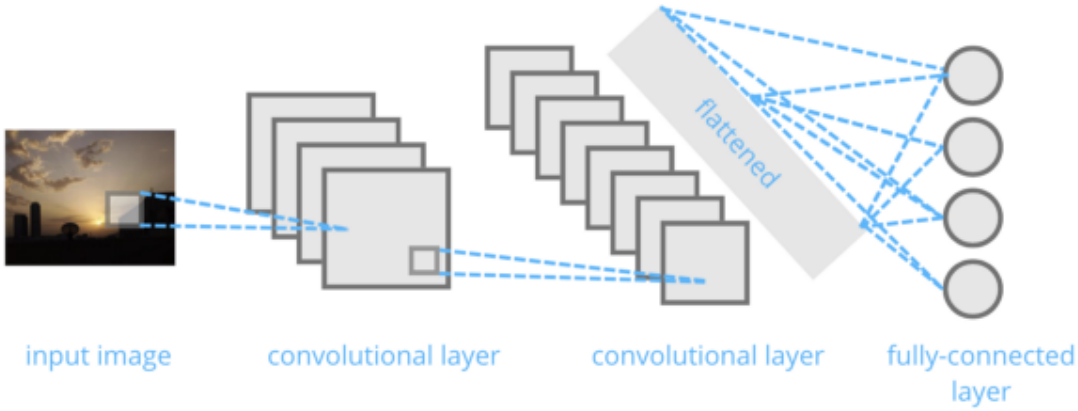
5.特征的相似性（使用CNN）

最后一种比较图像的方法是计算特征的相似性。众所周知，卷积神经网络CNN可以选择图像的模式并对其进行感知。卷积层是具有检测模式的滤波器。图像中的不同图案可以是边缘、形状或纹理。这些模式称为特征。

我们可以从CNN的卷积层中提取这些特征。

图10清楚地说明了一个示例架构。通常，我们指定网络的最后一个卷积层用于特征提取。

图10-一个简单的CNN架构



最先进的CNN架构之一是高效网（EfficientNet）。这是一种缩放方法，使用复合系数均匀缩放所有维度：深度/宽度/分辨率。我不会深入探讨它，因为它超出了本文的范围。然而，我将在以下几节中使用它。

通常，数据科学界在基于内容的图像检索系统中广泛使用特征的相似性。实验部分将解释原因。

5.1. EfficientNet-b0和欧几里得距离

在从EfficientNet中提取特征后，我应用欧几里得距离测量查询和数据集图像的特征之间的相似性，以找到最相似的特征。

```
from efficientnet_pytorch import EfficientNet
import numpy as np
from PIL import Image
from torchvision import transforms

# Load the model
model = EfficientNet.from_pretrained('efficientnet-b0')
model.eval()

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image")
# Note: images have to be of equal size

# Convert the images to tensors
image1_tensor = transforms.ToTensor()(image1)
```

```
image2_tensor = transforms.ToTensor()(image2)

# Add a fourth dimension for the batch and extract the features
features1 = model.extract_features(image1_tensor.unsqueeze(0))
features2 = model.extract_features(image2_tensor.unsqueeze(0))

# Calculate the Euclidean distance of the features
value = round(np.linalg.norm(np.array(features1.detach()) - \
                             np.array(features2.detach())), 4)
```

5.2. EfficientNet-b0和余弦相似性

计算特征的余弦相似性与前一个非常相似。然而，应用余弦相似性代替欧几里得距离。

```
from efficientnet_pytorch import EfficientNet
from PIL import Image
from torchvision import transforms
from torch import nn

# Load the model
model = EfficientNet.from_pretrained('efficientnet-b0')
model.eval()

image1 = Image.open("path/to/image")
image2 = Image.open("path/to/image")
# Note: images have to be of equal size

# Transform the images to tensors
image1_tensor = transforms.ToTensor()(image1)
image2_tensor = transforms.ToTensor()(image2)

# Add a fourth dimension representing the batch number and compute the feature
features1 = model.extract_features(image1_tensor.unsqueeze(0))
features2 = model.extract_features(image2_tensor.unsqueeze(0))

# flatten the features and apply cosine similarity
cos = nn.CosineSimilarity(dim=0)
value = round(float(cos(features1.reshape(1, -1).squeeze(), \
                        features2.reshape(1, -1).squeeze())),4)
```

在本节结束之前，如果得到的相似性为250、0.04或10809，该怎么办？使一对图像相似的数字是多少？答案如下：你必须根据对所选数据集的研究或特殊测试来定义此阈值。

数据集集合

在整个实验过程中，使用了两个不同的数据集进行评估：

- Fruits360数据集的一部分（96幅多尺寸图像）
- 收集并称之为SFBench的数据集由40张图片组成（3024x4032像素）

指定了第一个数据集来评估重复/近似重复图像查找器，因为它由每个类别360度拍摄的不同水果的图像组成。这些框架略有不同；图12显示了棕色划痕是如何顺时针移动的。

图11:Fruits360数据集的样本

图12:Fruits360数据集三帧之间的差异

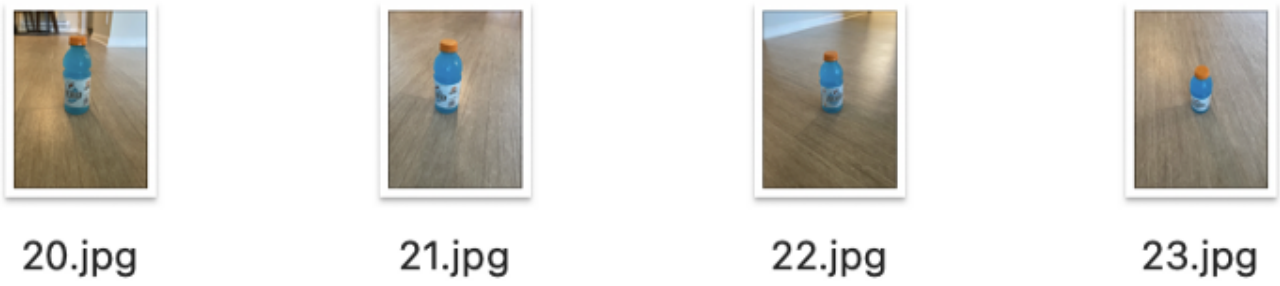
由于所有图像都是相邻帧，因此在此数据集上的测试将为我们提供关于重复图像查找器性能的良好反馈。这意味着，对于每个独特的类别，这些图片主要是相似的。

其次，SFBench是一个收集了40幅图像的数据集，用于评估基于内容的图像检索（CBIR）系统。

请注意，本文的目标不是构建或评估CBIR系统。我们使用该数据集仅测试图像变换（如3D投影和旋转）如何影响方法的性能。

数据集的一些示例图像如下图13所示。与第一个数据集一样，它由每个场景4个图像组成。

图13:SFBench数据集示例



实验

以以下方式使用两个数据集测试了每种方法：

1. 实验1：速度和准确性
2. 实验2：图像转换的弹性
3. 实验3: Scipy distance.euclidean vs. Numpy linalg.norm 速度

注意：在所有测试中都使用了2019年的MacBook Pro CPU。此外，你可以在Github存储库中找到所有测试。

https://github.com/OrjwanZaafarani/image-duplicate-finder-guide

实验1：速度和准确性

该测试为重复图像查找系统提出了速度和精度方面的最佳方法。以下步骤如下：

- 读取Fruits360数据集的图像。
- 将其转换为RGB
- 将图像大小调整为固定大小
- 使用5种方法

- 获取参考图像的3个最相似图像。
- 计算该方法用于比较一对图像的平均时间（秒）
- 计算准确率（对于每个参考图像，如果检测到3个重复/近重复，准确率为100%）

结果（如表1所示）清楚地表明，余弦相似性占主导地位，CNN的运行时间比余弦相似性慢约250倍，但是准确率却比较接近。此外，如果速度是一个很大的因素，那么图像哈希是一个很好的选择。

表1-实验1结果

Method	Part of Fruits360 Dataset (96 images)	
	Accuracy (%)	Speed (seconds)
Euclidean distance	0.0	0.0002
SSIM	94.4444	0.0413
Average hash	90.2778	0.0005
Cosine similarity	100.0	0.001
Euclidian distance of features (CNN)	98.6111	0.2513
Cosine similarity of features (CNN)	98.6111	0.2417

实验2：图像转换的弹性

该测试遵循与实验1相同的步骤。唯一的区别是使用的数据集和大小调整因子；我使用了SFBench，注意到图像重复查找器的目的不是检测和识别相似的变换图像。我只是评估这些方法在CBIR系统中的潜在使用的弹性。

从逻辑上讲，当CNN在其层内保存空间信息时，特征相似性方法表现最好。表2总结了以下结果：

表2-实验2结果

Method	SFBench Dataset (40 image)	
	Accuracy (%)	Speed (seconds)
Euclidean distance	10.0	0.0069
SSIM	13.3333	0.48
Average hash	23.3333	0.0022
Cosine similarity	36.6667	0.0087
Euclidian distance of features (CNN)	86.6667	0.76

Cosine similarity of features (CNN)	83.3333	0.7624

实验3:Scipy distance.euclidean vs. Numpy linalg.norm 速度

最后一个实验通过重复相同的操作约2300次来考察Scipy和Numpy实现的比较。此测试是本文的额外步骤，不会影响重复图像/近复制查找系统的功能。

结果表明，它们的性能相似（表3）。

表3-实验3结果

Method	Part of Fruits360 Dataset (96 images)	
	Accuracy (%)	Speed (seconds)
Euclidian distance of features (CNN) - scipy	98.6111	0.2497
Euclidian distance of features (CNN) - linalg.norm	98.6111	0.2513

结论

总之，我们学习了欧几里得距离、SSIM、图像哈希、余弦相似性和特征相似性的概念和Python代码。此外，我们还确定了图像变换对这些算法性能的敏感性。最后，通过实验，我们根据速度和准确性的要求，得出了最佳的方法。

你可以在Github存储库中找到所有数据集、实验和结果。此外，只要对每个数据集遵循相同的命名约定，就可以测试你选择的数据集。

https://github.com/OrjwanZaafarani/image-duplicate-finder-guide

参考引用

[1] Wang, et al, Image Quality Assessment: From Error Visibility to Structural Similarity, 2004

[2] Imatest LLC, SSIM: Structural Similarity Index, v.22.1

[3] Datta, All about Structural Similarity Index (SSIM): Theory + Code in PyTorch, 2020

[4] Mathematics LibreTexts, A Further Applications of Trigonometry: Vectors. 2021

[5] Nagella, Cosine Similarity Vs Euclidean Distance, 2019

[6] The Content Blockchain Project, Testing Different Image Hash Functions, 2019

[7] Krawetz, The Hacker Factor Blog, Looks Like It, 2011

[8] Gohrani, Different Types of Distance Metrics used in Machine Learning, 2019

[9] Clay, How to calculate Cosine Similarity (With code), 2020

其他有用资源

- Kundu, Various types of Distance Metrics in Machine Learning, 2019(https://medium.com/analytics-vidhya/various-types-of-distance-metrics-machine-learning-cc9d4698c2da)
- yatu, How can the Euclidean distance be calculated with NumPy?, 2020(https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy#:~:text=Use numpy.linalg.norm%3A)

☆ END ☆

如果看到這裡，說明你喜歡這樣的，請轉向、點贊。微信搜索「uncle_pn」，歡迎添加微博信「woshic，每日朋友圈更新微博文文章。

↓掃描二維碼添加小貼↓

技術分享 | 數據不同步半同步複製下，主從高能切換後數據一致嗎？

愛可生開源社區

YOLOV7 + StrongSORT 實現目標檢測與跟踪

迷途小書僮的注意事項