towhee

Build an Image-Deduplication Engine in Minutes

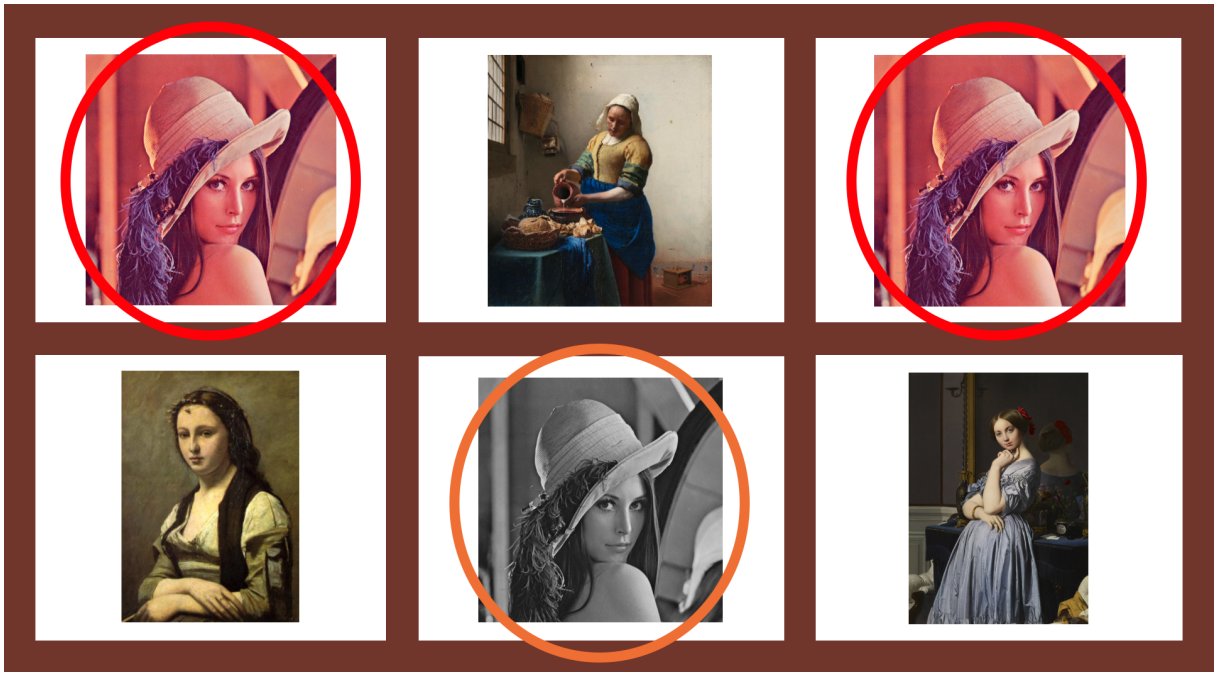
@Towhee.io

Image Deduplication

Introduction

duration: 1

Image deduplication is the process of finding exact or near-exact duplicates within a collection of images. For example:



In particular, note that the middle image in the bottom row is not identical to the other two images, despite being a “duplicate”. This is where the difficulty here lies - matching pure duplicates is a simple process, but matching images which are similar in the presence of changes in zoom, lighting, and noise is a much more challenging problem.

In this section, we go over some key technologies (models, modules, scripts, etc…) used to successfully implement an image deduplication algorithm.

Encoder-based embedding model

duration: 2

A generic embedding model turns images into dense vectors; an encoder-based embedding model outputs dense vectors which encode scale-invariant edges and corners within the input image as opposed to pure semantic information. For example, while two images of different dogs may result in two very similar encodings when using traditional object recognition embedding models, the output embeddings would be very different when using encoding-based embedding models. This blog post is a great resource for understanding contrastive loss.

To accomplish this, these encoder models shouldn't be trained on traditional image recognition/localization datasets such as CIFAR or ImageNet; instead, a siamese network trained with contrastive or triplet loss must be used. Among all these encoder-based embedding models, `resnet` is a widely applied one. In this tutorial, we take `resnet50` as an example to show Towhee's capability of comparing similar images in a few lines of code with image-processing operators and pre-built embedding models:

```
from towhee import dc

dc_1 = dc['path_1', 'path_2']([['Lenna.png', 'Lenna.png'], ['Lenna.png'
    , 'logo.png']])\
     .image_decode['path_1', 'img_1']()\
     .image_decode['path_2', 'img_2']()\
     .image_embedding.timm['img_1', 'emb_1'](model_name='resnet50')\
     .image_embedding.timm['img_2', 'emb_2'](model_name='resnet50')

dc_1.show()
```

A `resnet50` model is trained to output extremely close embeddings for two "similar" input images, i.e. zero, one, or many of the following transformations:

1. Color conversion, e.g. changes in lighting/contrast
2. Up to 150% zoom plus a random crop
3. Additive salt-and-pepper (Gaussian) noise

These transformations render the model invariant to changes in zoom, lighting, and noise.

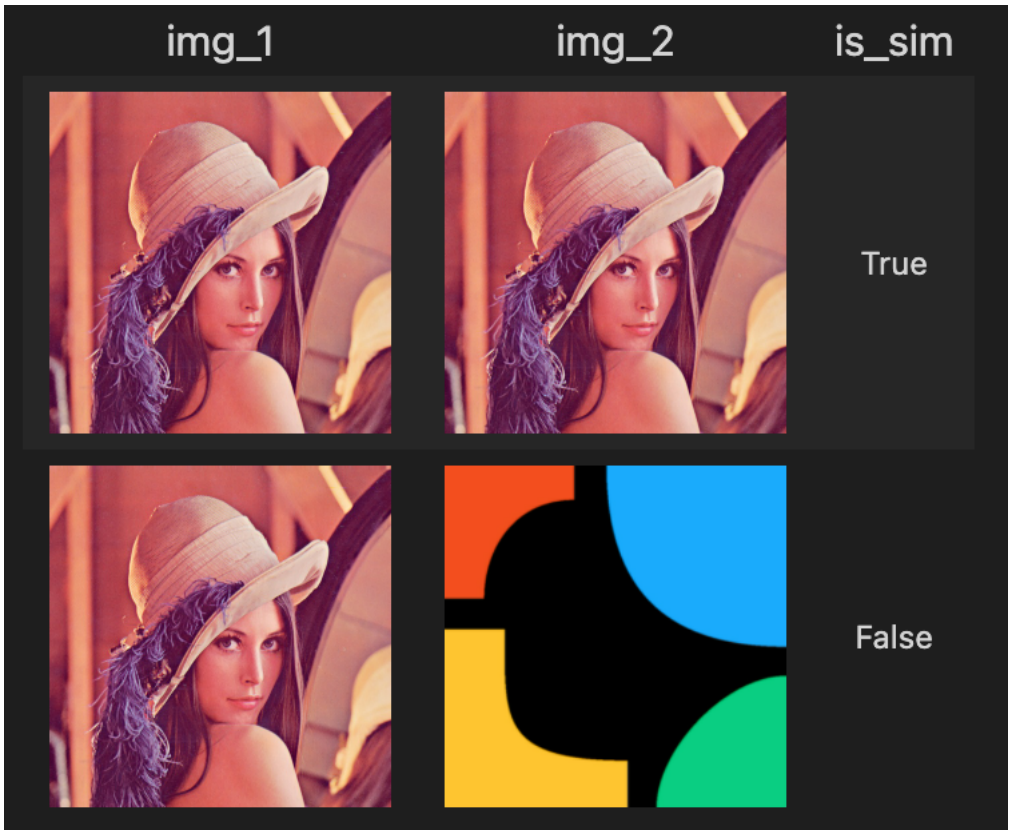
Similarity metric

duration: 2

Now we have the embedding of the images stored in dc, but the embeddings themselves are useless without a similarity metric. Here, we check if the L2 norm of the difference vector between the query and target images is within a certain threshold. If so, then the images are considered duplicates.

Towhee also support runnig a self-defined function as operator with `runas_op`, so users are allowed to run any metrics effortless once they define the metric function:

```
import numpy as np
thresh = 0.01
dc_1.runas_op[('emb_1', 'emb_2'), 'is_sim'](lambda x, y: np.linalg.norm
    (x - y) < thresh)\
     .select['is_sim']()\
     .show()
```



This is an empirically determined threshold based on experiments run on a fully trained model.

Putting it all together

duration: 1

Putting it all together, we can check if two images are duplicate with the following code snippet:

```
from towhee import dc
import numpy as np

thresh = 0.01
res = dc['path_1', 'path_2']([['path/to/image/1', 'path/to/image/2']])\
    .image_decode['path_1', 'img_1']()\
    .image_decode['path_2', 'img_2']()\
    .image_embedding.timm['img_1', 'emb_1'](model_name='resnet50')\
    .image_embedding.timm['img_2', 'emb_2'](model_name='resnet50')\
    .runas_op[('emb_1', 'emb_2'), 'is_sim'](lambda x, y: np.linalg.norm
        (x - y) < thresh)\
    .select['is_sim']()
```

And that's it! Have fun and happy embedding :)