Effortlessly Recommending Similar Images

Using features from pretrained convolutional neural networks to generate comparability

Maciej D. Korzec Jun 27, 2020 · 9 min read ★

Automatically selected similar images (Source: M. D. Korzec)

recommender system that can be used for example in online-shops. To this aim, we will use transfer learning, utilizing pre-trained convolutional neural networks.

Private set of images (Source: M. D. Korzec)

These pictures from my private repository contain varying motives and when facing thousands of images, it becomes exhaustive to find the most similar ones. With the presented solution we will obtain similarity values based on image content (i.e. not on its metadata) as in the example below from the same image set. The first image depicts the queried reference image, a van that we used for a maternity leave trip through Europe, standing on a goat farm somewhere in France.

Similar images to a query image with similarity values (Source: M. D. Korzec)

In the row below we see the images that the algorithm selected to be most similar. The numbers beneath the photographs are the similarity values ranging from 0 to 1, with 1 being the highest possible similarity. Not only the images with the red van are shown, also our first rental in yellow that needed to be replaced due to a failing water pump was recommended due to all the other similarities.

Many online offerings could work with such a recommender to increase conversion rates, be it photographer's collections, movie recommendations, or other shops that could benefit from image similarities (e.g. CD covers or other products where visual similarity might be conversion increasing). These kind of item-item based system recommender systems are called content-based. For more information you may refer to [1] or for a fast entry one of the blog entries on *Towards Data Science* [2,3].

There is a remarkably simple way to achieve such recommendations if you already have basic knowledge of Python and image processing. In the next paragraphs I will provide the high-level logic behind it for a fixed set of images:

1. Get and preprocess data

2. Use trained convolutional neural networks as feature vector generators

3. Use feature vector differences to calculate similarity

4. Use top-k lists for recommendations

You can use the neural networks from the shelf, no model engineering or training required. There is no need to clean metadata, there are no hidden efforts, you only need to set-up an efficient data handling once and have no additional work.

1. Get and preprocess data

To implement even a prototype, a large image test set is needed. I worked with a part of the Coco set [4] and the results were very impressive. Within 15000 images the recommender found very similar contents. However, as the images have different licenses based on the user's selection in Flickr, I refrain from posting these results here. Instead, I created a different test set for this blog. All posted images are from my local image repository of approximately 5000 images, all in jpg format with different resolutions as Cameras changed.

Pretrained deep neural networks require input images in a certain format. For this article I used resnet18 [5] within the PyTorch library, it requires 224x224 RGB images as input, with image values in the range [0,1]. The images are normalized to have the mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225], so they were all rescaled to fit this format (be careful not to lose metadata about the image orientation during transformation). Details about a particular implementation will be presented in my next blog entry. If you want to use other convolutional neural networks, check the input requirements first and adjust the pre-processing accordingly. Furthermore, note that here the weights for resnet18 were obtained from training on ImageNet. Keep in mind that training and test sets shall be different.

2. Use trained convolutional neural networks as feature vector generators

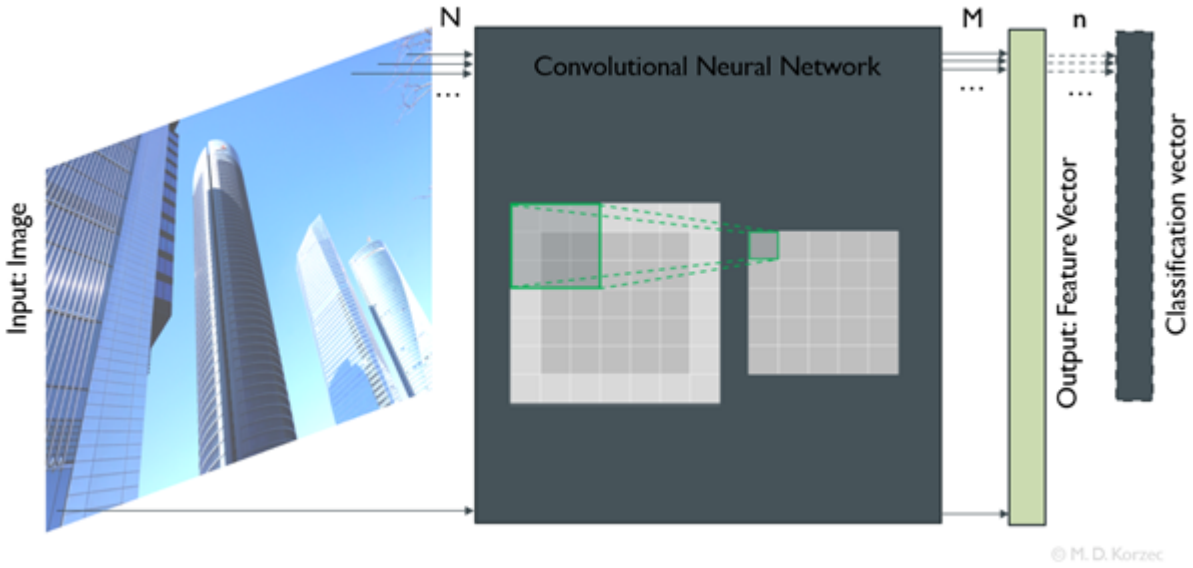
One characteristic of image related neural networks may strike you as a surprise the first time you realize it:

> ***Trained convolutional neural networks create feature vectors that can be compared for calculating similarity of images.***

Instead of struggling to find ways to characterize images with classic image processing methods that might be used to compare them (e.g. start with edge detection for different orientations and local edge densities, for example, and then extend for all cases

image sets with high motive variability. You do not need to understand any of the entries of the vectors, you just need to know that similar images will have similar entries.

The following schematic explains the calculation of feature and classification vectors when treating the core network as black box



Convolutional neural net as black box (Source: M. D. Korzec)

M>n and the feature vector contains a lot of encrypted high-level description of the image. Hence when abstracting all the layers in between and ignoring the classification vector that is not used for the recommender, we just deal with a function

$$f: R^N \to R^M, x \mapsto f(x)$$

that is familiar even from high-school. Here, x is the vectorized RGB image in

$$R^{mxpx3}, m * p * 3 = N.$$

When using a pre-trained model, all parameters are already fix, so there are no further free variables and they are not listed here. One will have to preprocess given images to

What you don't need to do

When you want to train your network from scratch, you need to

- Prepare a lot of data manually (image — classification vector — pairs)
- Spend time on longsome optimization

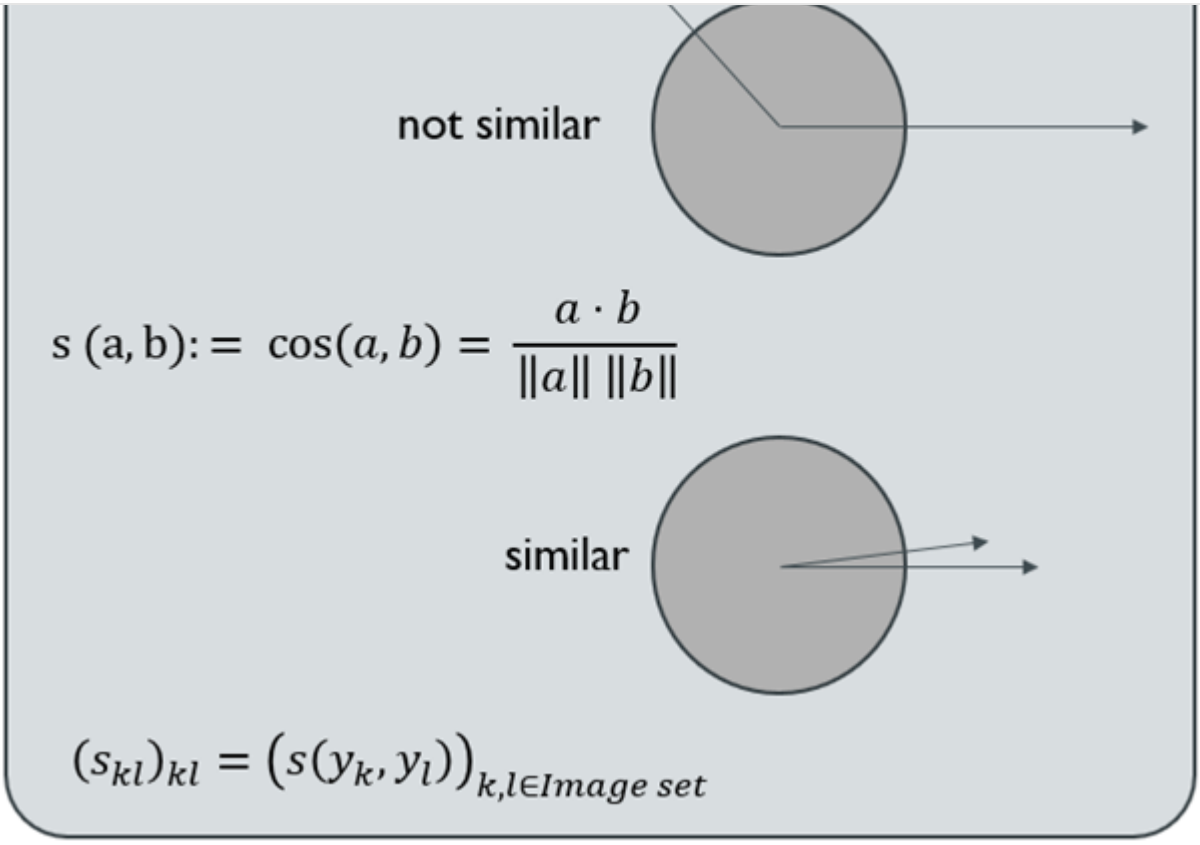
The classification vectors (e.g. "the image contains a car" may be represented by a unit vector with the one in the position representing cars) are fixed, the input is fixed, and the weights of the network are optimized to get a guessed classification vector that is as close as possible to the real one. The well-studied form of f(x) allows to optimize the weights with help of automatic differentiation that is used during the optimization iterations and makes deep neural network usage possible in the first place. If you are interested to learn more about related topics, you may refer to a former research colleague's blog post on this topic [6].

A fully connected one-dimensional layer before the classification layer with more entries is a high-level, encrypted description of the contents of the images. Earlier layers only describe low-level features (such as edges) and the classification vector may be rather small, so that the last fully connected layer is a good choice. It will contain usable values even for images with content that was not classified during learning.

Hence we know that we can spare the efforts of training a neural network and we can still can get feature vectors with one function call f(x) from the image input x. Then we can easily decide when an image is more or less similar to another one.

3. Use feature vector differences to calculate similarity

To compare the feature vectors derived from the evaluation of the convolutional neural network for different images, employing the cosine similarity is a great and simple approach. Here, you basically just calculate the formula for cosines for all image pairs, doing this for the whole image set yields a similarity matrix.



Cosine similarity (Source: M. D. Korzec)

In two dimensions the formula is easy to interpret and visualize as it is shown above. It works in the same way in higher dimensions, consider for example the three vectors

$$a = (10, 1, 0), \qquad b = (9,1,1), \qquad c = (2, 1, 5).$$

By intuition, the similarity of a and b shall be higher than that of a and c or b and c. And indeed, when plugging the numbers into above formula, one finds

$$s(a, b) = 0.99, \qquad s(a, c) = 0.38, \qquad s(b, c) = 0{,}48.$$

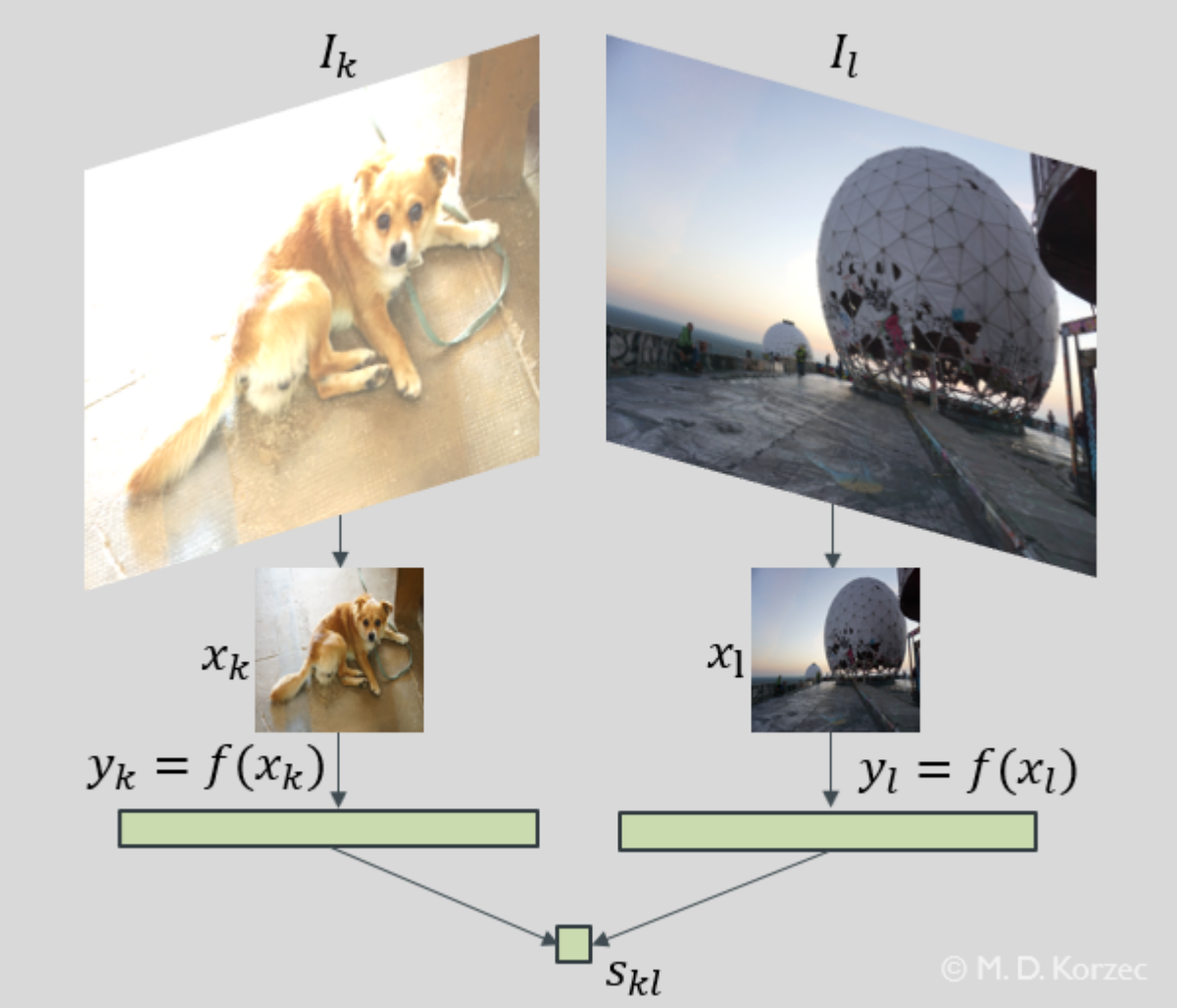
Hence, as expected,

$$s(a, b) \gg s(a, c), s(c, b) \gg s(b, c).$$

vectors.

4. Use top-k lists for recommendations

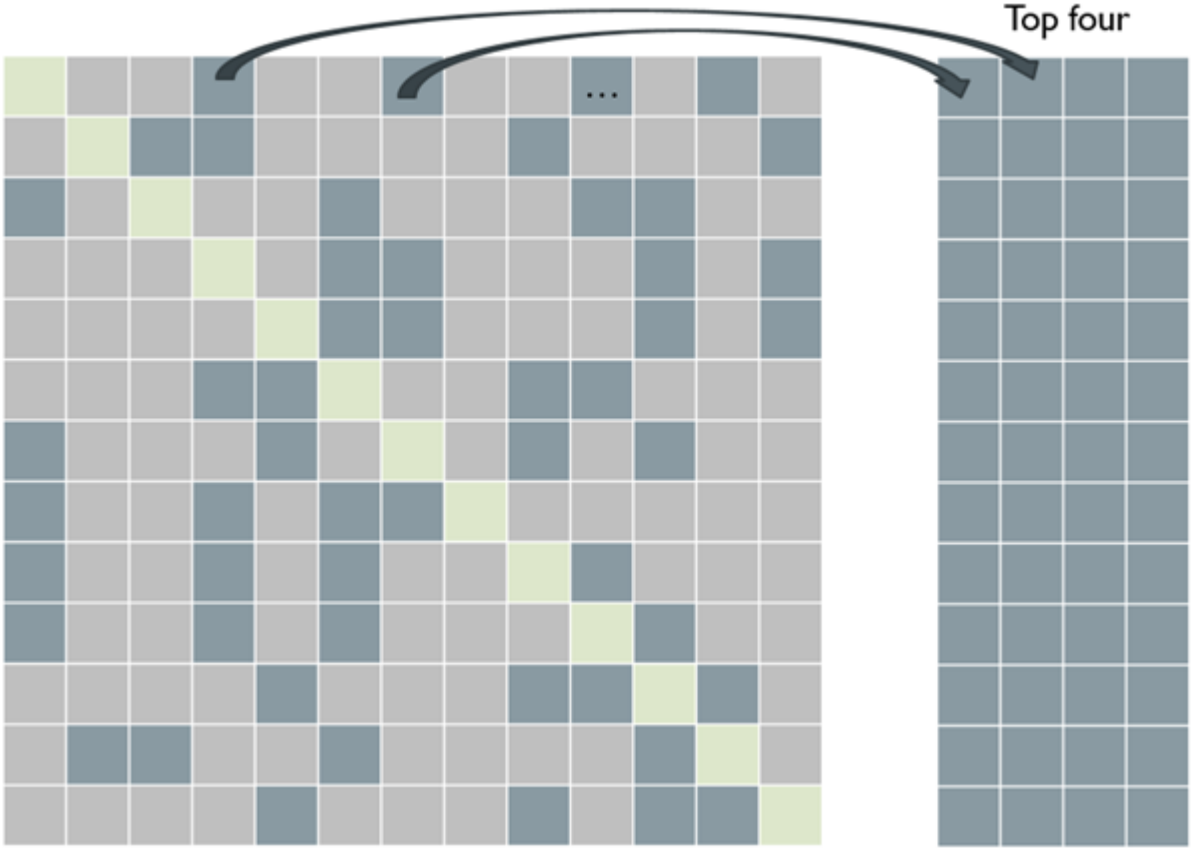
So, overall, when two images are compared, we have already derived the following logical steps to be carried out



Comparing two images (Source: M. D. Korzec)

You only need to iterate all images from the set of interest. All derived entries (s_kl) together define the full similarity matrix. However, working with the full similarity matrix is not efficient, when using the result for recommendations online. When

matrix (s_kl) is sorted from large to small and highly similar images are kept and used for recommendations. The following diagram explains the collection in smaller vectors/matrices for the generation of top-4 lists from a similarity matrix.



Similarity matrix and top 4 lists (Source: M. D. Korzec)

The darker tiles represent the four most similar images to the image represented in that row and are stored separately for efficient requests. Of course, you would use more than four in typical applications.

When you have all the sorted information in top-k lists and receive a query as input, you only go to the corresponding row of that particular image and present the images stored with the IDs stored in the corresponding row.

Takeaway messages

- We use pretrained models, so there is no need to set-up a training pipeline.
- We essentially only evaluate a function to obtain feature vectors.
- We compare feature vectors by calculating their cosine.
- When new images are added to the set, new similarity values and the top-k lists need to be updated.
- When you start with a sufficiently large image set and a well-trained model, there is no cold-start problem.

If you have Python code covering the above recommender ideas and it shall be used on a website, there are various ways to do so. In my next posts, I will explain the main points for a Python implementation of the above steps, and I will explain how a simple Flask based website can be set-up that uses the results.

Similar dandelion images from the test set (Source: M. D. Korzec)

[1] C. C. Aggarwal, Recommender Systems — The Textbook (2016), Springer

[2] P. Pandey, The Remarkable world of Recommender Systems (2019), towards data science

[3] B. Rocca and J. Rocca, Introduction to recommender systems (2019), towards data science

[4] T.-Y. Lin et al., Microsoft COCO: Common Objects in Context (2014); Used: Test data 2017

[5] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, 2015

[6] M. Köpf, High-School Math for Machines: Differentiable Programming (2020), towards data science

Thanks for reading! Liked the topic?

If you found the read interesting, you might want to catch up on my other articles on the topic:

The Core Value of Recommender Systems for Successful Internet Enterprises

An assessment of the importance of recommender systems

medium.com

Recommending Similar Images Using PyTorch

Full transfer learning implementation with Resnet18

towardsdatascience.com

A Flask App for Image Recommendations

Website prototype for similar image recommendations based convolutional neural networks in PyTorch

medium.com

Image Recommendations with PyTorch + Flask + PostgreSQL + Heroku deployment

Packaging a PostgreSQL/ PyTorch based image recommender system with Flask, importing data and running it on the Heroku...

towardsdatascience.com

Recommendations Image Similarity Convolutional Network Marketing Automation

Machine Learning