Image processing. Cover "reverse image searching"

How?

We can use computer vision techniques for matching an image with another image with techniques like SIFT and ORB. We can also use Neural Networks

SIFT

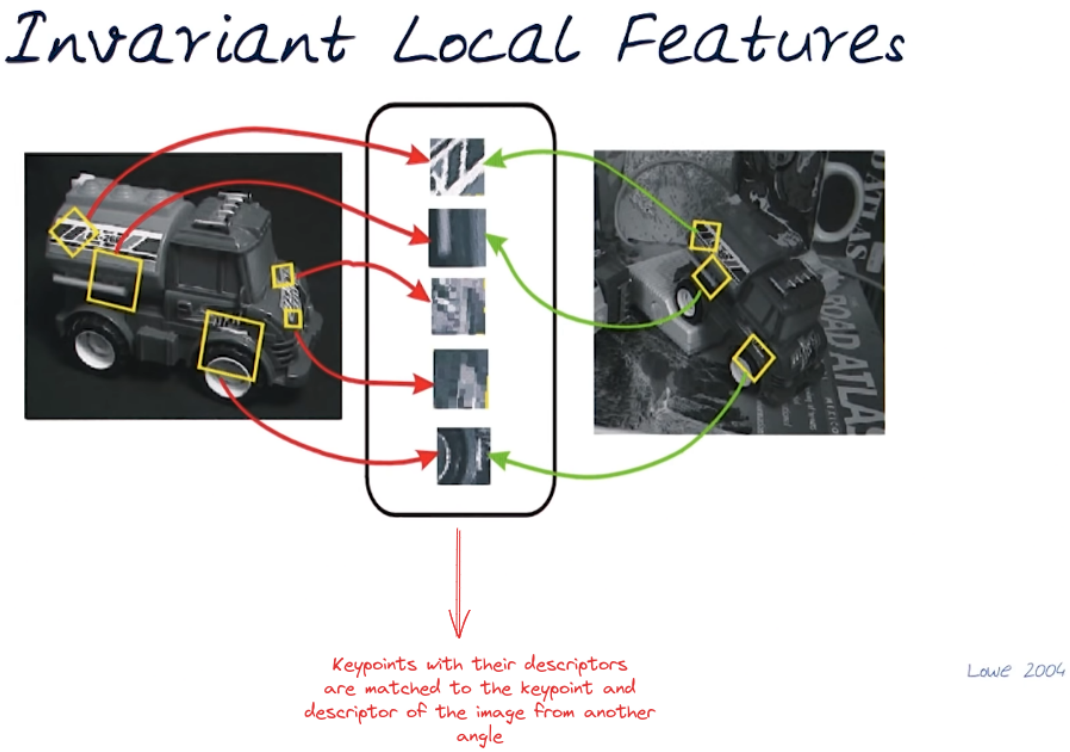
The scale-invariant feature transform (**SIFT**) is a *computer vision* algorithm to detect, describe, and match local *features* in images.

How does it work?

- SIFT starts by creating a series of blurred images at different scales (octaves) using a Gaussian filter. Each octave consists of several levels, and each level is a blurred version of the previous level.
- The difference between consecutive blurred images within each octave is computed to detect potential interest points, which correspond to regions of significant intensity changes.
- Local maxima or minima are identified in the DoG (difference of gaussian) images across scales. These extrema correspond to potential **keypoint**.
- At each **keypoint**, the gradient magnitude and orientation of the image are computed. This information is used to assign an orientation to the **keypoint**.
- A histogram of gradient orientations is created in the local neighborhood of the **keypoint**. The peak in the histogram indicates the dominant orientation, and the **keypoint** is assigned that orientation.

- SIFT creates a local image **patch** around each **keypoint** based on its scale and orientation and then in the local patch, a **descriptor** is computed based on the gradient magnitudes and orientations. This descriptor is a vector that captures the distribution of gradient information in the neighborhood of the **keypoint**.

Example



Key Words

- **Keypoint** - a distinctive and identifiable point or location in an image. Keypoints are often selected based on features such as corners, edges, or other points with significant local intensity variations. They serve as anchor points for further analysis and matching.
- **Descriptor** - a compact representation of the visual information in the neighbourhood of a keypoint. It captures the distinctive characteristics of the local image region surrounding the keypoint.
- **Patch** - a small, rectangular or square region extracted from an image, typically centered around a keypoint. It represents the

local area of interest and is used as the basis for computing descriptors

ORB

Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector, that can be used in computer vision tasks like **object recognition** or **3D reconstruction**. It is based on the **FAST** keypoint detector and a modified version of the visual descriptor **BRIEF** (Binary Robust Independent Elementary Features). Its aim is to provide a fast and efficient alternative to SIFT.

How it works?

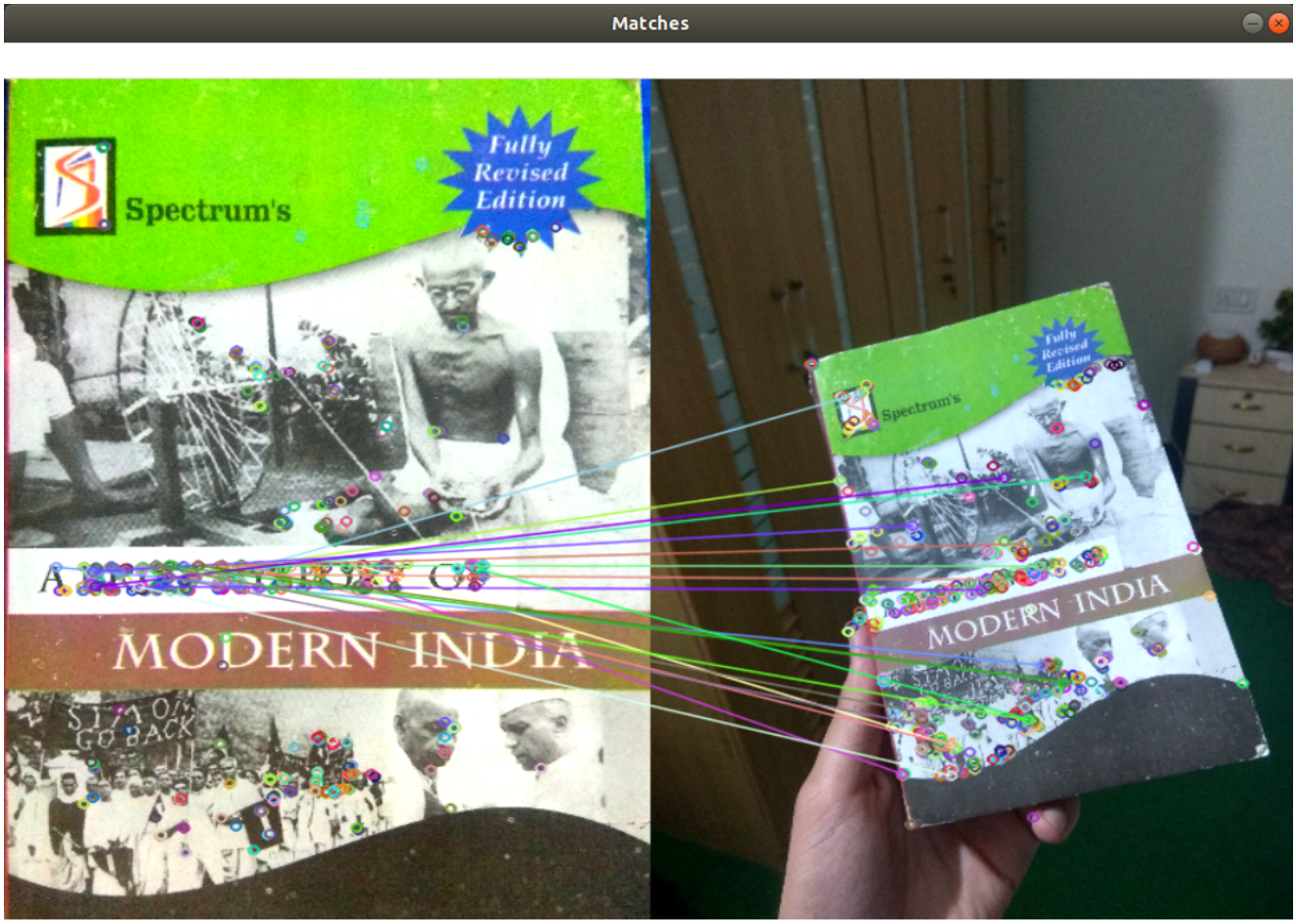
It works much similar to SIFT except it uses the FAST keypoint detection algorithm and BRIEF to make the descriptors for a patch

- ORB starts by using the FAST (Features from Accelerated Segment Test) algorithm to identify key points in an image. FAST is a corner detection algorithm that quickly identifies areas where intensity changes significantly.
- The key points obtained from FAST are further filtered using the Harris corner response. This helps to select more stable and distinctive keypoints by considering the local intensity gradients.
- Once key points are detected, ORB computes oriented BRIEF descriptors. BRIEF (Binary Robust Independent Elementary Features) is a binary descriptor, meaning it represents the local image patch around a key point as a binary code (0s and 1s).
- ORB introduces orientation information to make the descriptors rotation-invariant. The orientation is computed based on the intensity centroid of the patch.
- To match features between different images, ORB uses a simple Hamming distance to measure the similarity between the binary descriptors. This makes the matching process efficient.

> **Important**
>
> While ORB is faster and utilises considerably less memory, and the script execution time is also faster, the results are somewhat questionable since some matches make little sense in

comparison to SIFT or Neural Networks. Will be seen in the examples

Example



Neural Networks

Neural networks can be used for feature extraction as easy as in the case of ORB or SIFT. All we need is a pre-trained model that can extract features and then we can easily compare cosine distances between descriptors of 2 images.

What kind of Networks?

In our case which is book cover recognition, the recommended type of Neural Networks are CNNs (Convolutional neural network).

Why CNN's? Because Convolutional Neural Networks (CNNs) are well-suited for image recognition due to their ability to automatically learn hierarchical features from data. Their use of convolutional layers enables them to capture spatial hierarchies and local patterns in images, making them effective for tasks like object detection and classification in computer vision.

Potential Libraries to be used

For SIFT and ORB approaches we can use one library that is well known in the computer vision space and that is OpenCV  . OpenCV allows us to easily run the SIFT and ORB algorithms on an image and calculate distances via different methods like Hamming Distance e.t.c.

In terms of Neural Networks there are a lot of options like ImageNet and so on. I have picked out one library with which I did an example with, and that is PyTorch which is a ML library based on the Torch library. It has several networks available to our disposal like ResNet-18 which is a CNN that has a pre-trained model on the ImageNet dataset

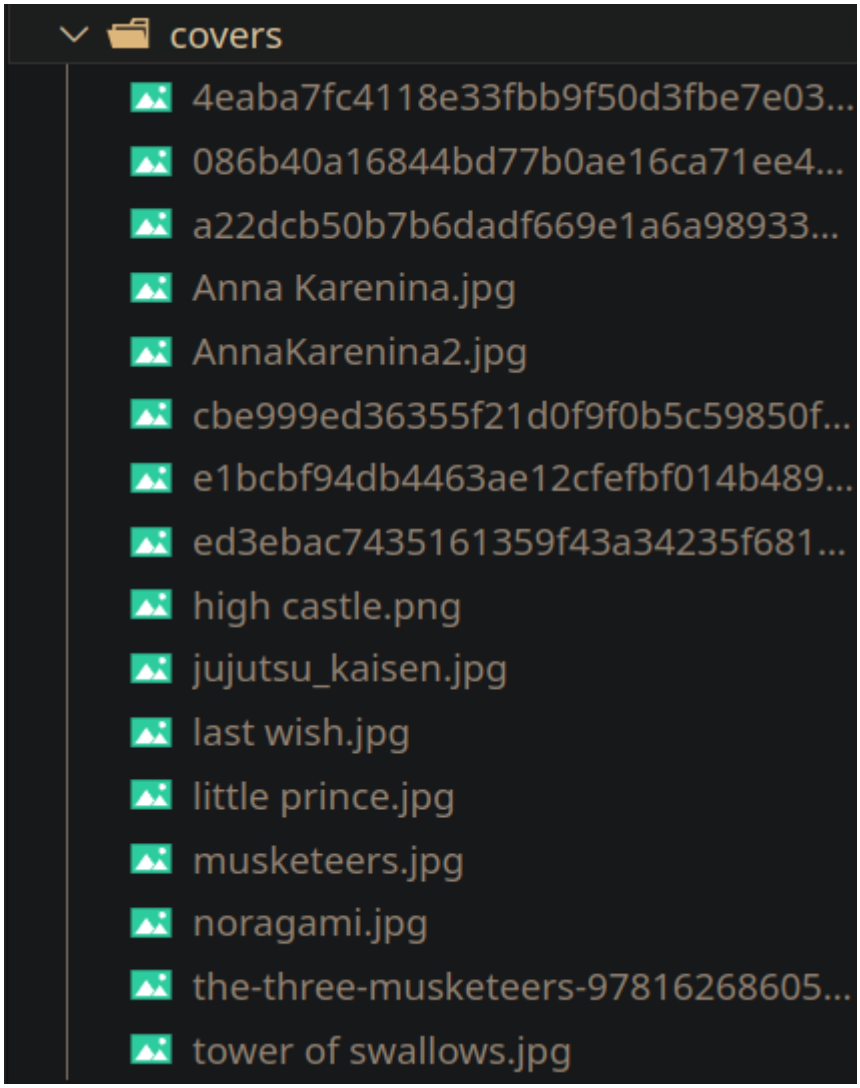
Examples with our use-case

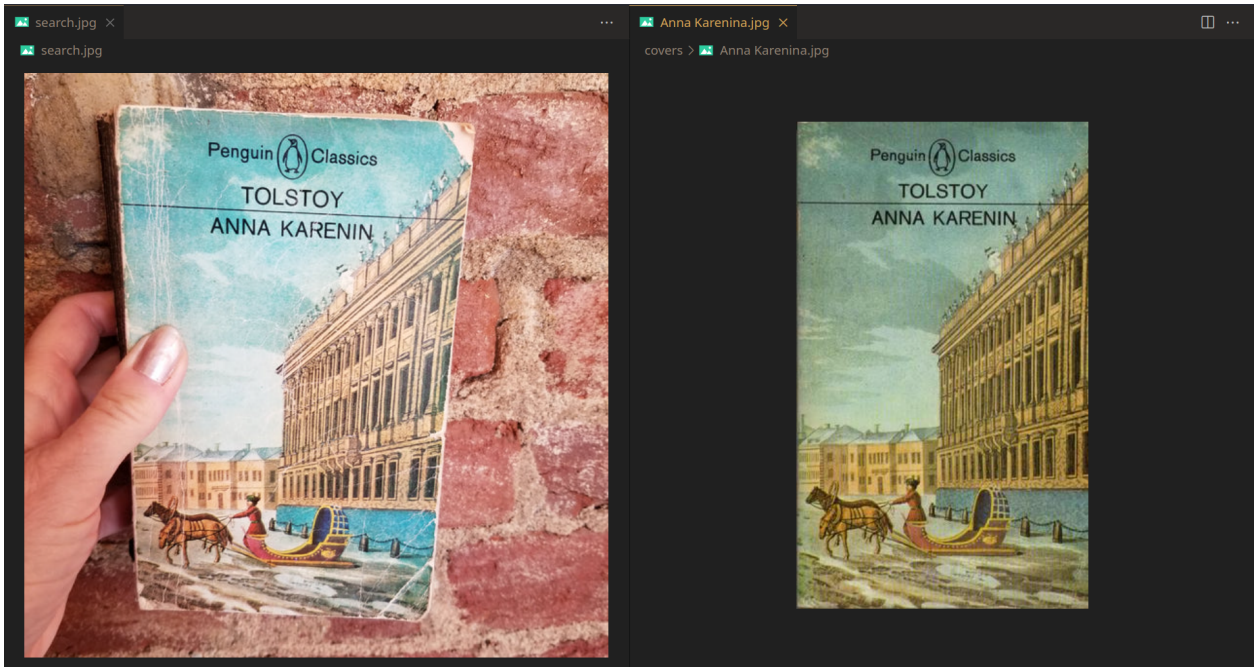
Example reqs:

- We have a database of covers which are .jpg or .png images like we would in our library
- We have a query image which would be simulating a librarian photographing the book cover of a book
- For the SIFT and ORB versions we need to have OpenCV installed as a library
- For ResNet-18 example we need to have PyTorch installed

Photo's used

- As we have established we have a folder with covers for the books that we have

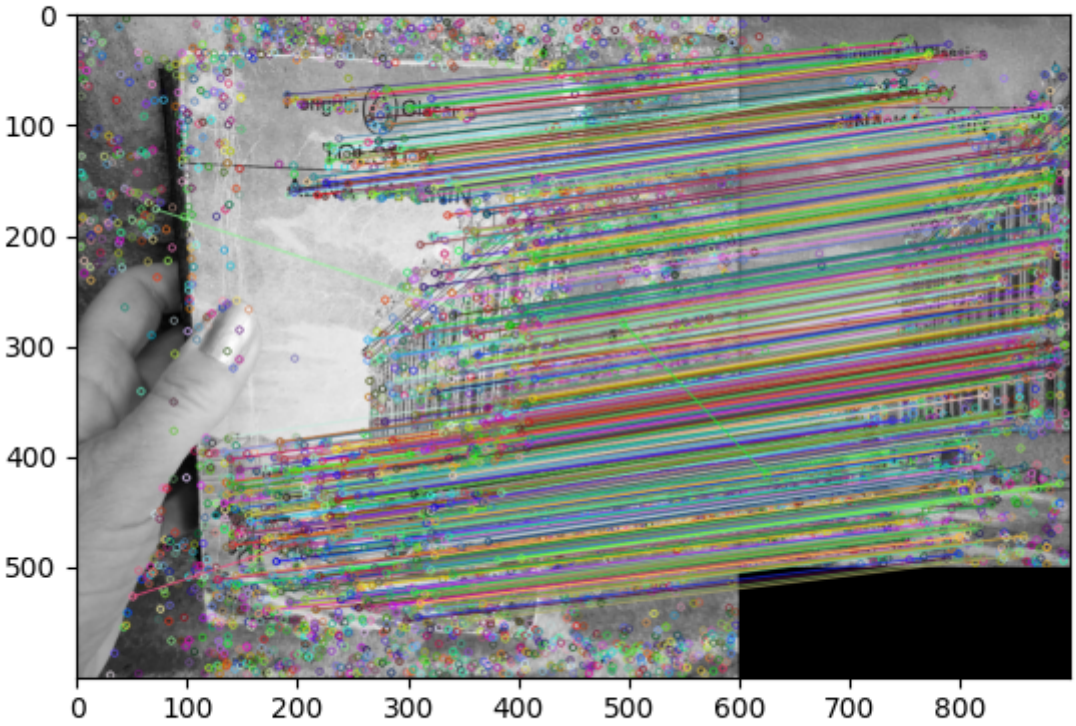




Query Image on the left, our book cover on the right that we are trying to match

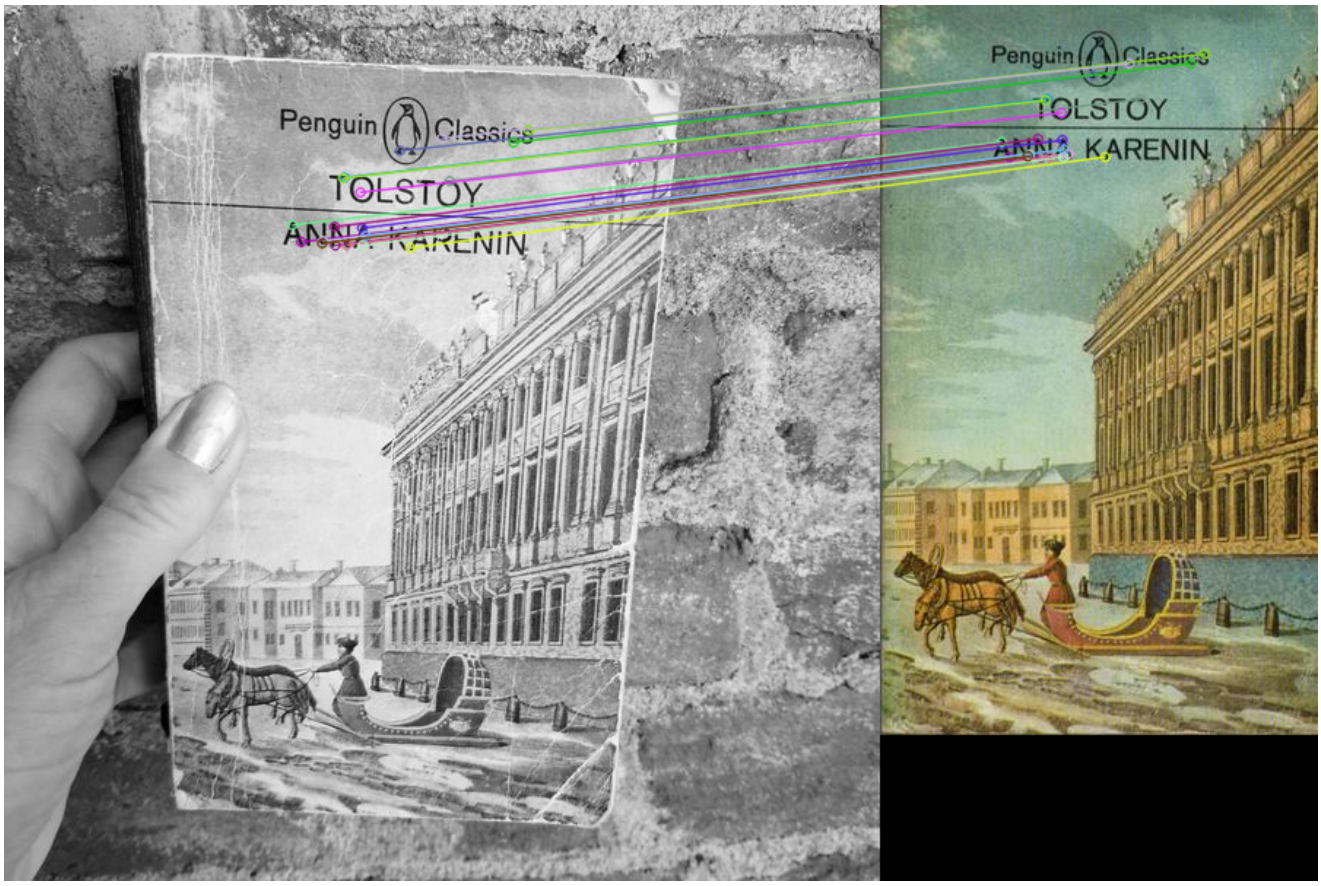
Results:

SIFT



Hard to see because of the matches being displayed but as we can see it matched the most to the correct book cover

ORB



We also get the correct matched image

ResNet-18

```
chris@archlinux ~/Programming/pyite_test
(0) > /home/chris/Programming/pyite_test/.venv/bin/python /home/chris/Programming/pyite_test/test.py
Top 5 similar images:
Anna Karenina.jpg: 70.83886861801147%
4eaba7fc4118e33fbb9f50d3fbe7e034920e3e0b9ef4ee5d7668213cc81779f8.jpg: 70.09477019309998%
ed3ebac7435161359f43a34235f6811369924a5d3b938adc2b3b13e5a46ff816.jpg: 69.5488691329956%
e1bcbf94db4463ae12cfefbf014b4898b96d55400ad4369bb544ad095959c766.png: 68.60584020614624%
a22dcb50b7b6dadf669e1a6a989333a37c4a7f674546bb9f49525e8c07b59c24.jpg: 68.27179193496704%
```

The result is as expected, the best match is the book cover

Finally, a skeleton of how things could work.

---

1. When adding a book to the database we extract it's features from the cover images and store them into a field/property
2. When we search via an image, all we will need to do, is pass the query image, to the same process, and then compare the features extracted with all of those from the database.

3. Then based on a best match display the book information/return/do whatever we want with the book recognised