CoasterDex

software for detection and recognition of beer-coasters from images

Authors
David Pažout & Dagur Elinór Kristinnson
RU
19.12.2023

Contents

Abstract

CoasterDex is a software for detection and classification of beercoasters in images and video. It combines fine tuned Yolov5 based CNN with custom SIFT based instance level recognition algorithm to achieve high accuracy detection of beercoasters across varied environments.

1 Introduction

1.1 Problem Definition

This problem is inspired by my friend who collects beercoasters. They have a collection of more then 200 unique beercoasters. Every time I'm in a pub and see a beercoaster I don't know if they already have it or not. They aren't always available to be asked or they sometimes don't remember and bringing them a duplicate coaster is suboptimal.

Formal definition is as follows, we have a collection $C = \{c_1, ...c_n | c_i \neq c_j;\ \forall i, j;\ i \neq j\}$ of images of unique beercoasters or coaster-scans from frontal view. Given an input image img, the goal is to find all beercoasters $B = \{b_1...b_m\}$ present in the image and determine if a beercoaster $b \in B$ is also present in C. If b isn't present in the collection we should add it to the collection ($C \leftarrow C \cup b$).

1.2 Relevant Work

We found a few papers tackling a similar or adjacent problems. The oldest is a paper from 2011 solving postage stamp recognition from photo-scans [**?**]. Since the paper was written before wide adoption of CNN it mainly uses custom, hand-crafted features and basic template matching for classification of relatively small number of stamps. The authors in HotSpotter [**?**] try to match individual animals (not species) with past photos in a larger database with multiple instances of each individual. In [**?**] CNN is used to tackle grocery product recognition. The authors approach the problem as classification of images into 100 plus classes.

Most of the approaches for the instance recognition were taken from two throughout overviews on the subject [**?**] [**?**], where authors compare older SIFT based and newer CNN based approaches to this problem.

Working with small datasets can be tricky since it can lead to overfitted models. Using smaller models increases performance in these cases.[**?**] Data augmentation is also a powerful tool for these situations due to its ability to increase dataset size artificially. [**?**]

Majority of the previous approaches work with a static databases, e.g. the whole pipeline/algorithm have to be retrained if we wand to introduce a new type of sample. They often have multiple scans/instances available for retrieval and aren't limited by the time it takes to detect and retrieve the relevant object. We have only single instance of the coaster available in the collection and want to use our system in real-time environment where prolonged waiting times are inconvenient. System also should easily scale to larger collections (thousands of items).

2 Materials and Methods

We decomposed the problem to two main parts: Finder and Matcher. Finder is an image classification and localization algorithm and its goal is to find all occurrences of beercoasters $B = \{b_1...b_m\}$ in the image img with their appropriate bounding boxes. Matcher is an instance level recognition/retrieval algorithm and its goal is to match a beercoaster from the collection $c \in C$ with a beercoaster b present in the image img or to determine that there is no matching beercoaster present in the image.

2.1 Finder - Coaster Detection

To implement a coaster finding program the utilized method was yolov5 object detection model small version. The model was fine tuned with a custom dataset specifically designed to train coaster detection. Roboflow was used to label the dataset and perform augmentations. With a functional coaster detection model, opencv was used to extract the given bounding boxes and then returned to the matcher for integration.

2.2 Matcher - Instance Recognition

There are many approaches to instance recognition/retrieval. The two most popular are ”classic” SIFT based approaches and ”modern” CCN based approaches. The goal of both is to extract one (or multiple) feature vectors of fixed length from the image and compare them to feature vector(s) extracted from the images in the collection and retrieve the most similar images based on similarity of those feature vectors. The general pipeline is summarized in fig1. It consist of three steps: feature extractions, encoding and indexing/search. We opted for a compact representation of the final encoded feature vector, this enables us to easily test both SIFT and CNN base approaches with Nearest Neighbor (NN) search.

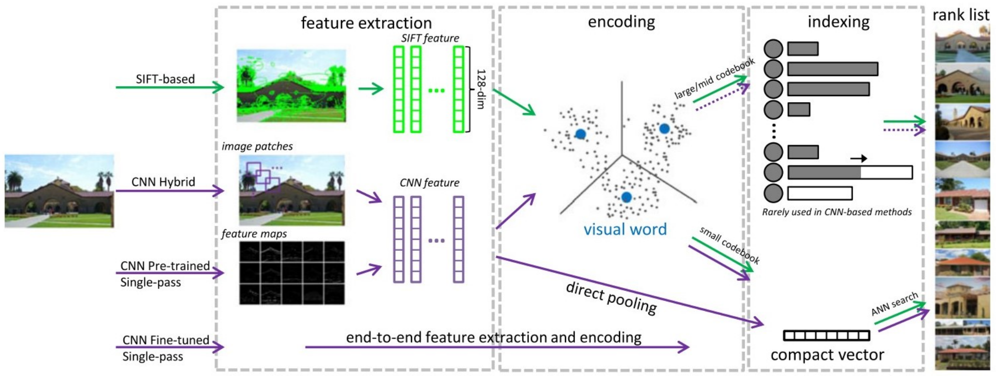

Figure 1: general instance retrieval pipeline as presented in [?]

For SIFT based recognition we tried multiple feature extractors such as SIFT, RootSIFT and ORB. Encoding used was VLAD [?] from [?]. For the CNN based recognition we limited ourselves to the Pre-trained CCNs since our dataset was relatively small and unsuitable for finetuning. The CNN method chosen was R-MAC [?] from [?] We had trouble finding suitable CNN feature extractor implementations in python (we found multiple in Matlab) so we were unable to compare multiple approaches. The search methods tried were Aproximate NN from [pynndescent], Product Quantization (PQ) from [nanopq], Locally Sensitive Hashing (LSH) from [nearpy] and KDTree and BallTree from [sklearn]. In the end we choose BallTree NN search since it outperformed most of the searches in both time needed to create the search structure and time needed for addition of new element to the search It is also suitable for search in high-dimensional spaces.

The core of the pipeline consists of SIFT feature extractor, VLAD encoding and BallTree search. For preprocessing we scale large images to maximum width of 640px while preserving the aspect ratio. We convert images to grey-scale and apply Contrast Limited Adaptive

Histogram Equalization (CLAHE) [?] to increase the number of features extracted in low-light environments. For post processing we compare a number of matching features between the inputted and retrieved images. If number of matching features is less then 16 we discard the retrieved image and determine that no matching coaster was found in the image. We also tried and discarded geometric feature matching with RANSAC [?] but this greatly increased the processing time needed and usually only let to less then 4 feature matches between the images.

2.3 Finder&Matcher Integration

We first input the image into the Finder, as output we receive a list of bounding boxes. If the list is empty we input an unaltered image to the matcher. Otherwise we do for each bounding box separably the following: we extract part of the image inside the bounding box, stretch the cropped image to a square - this helps with feature extraction since most coasters are of square-ish shape - and input the stretched image into the Matcher. As the output we have a list of best matched coasters for each bounding box or the whole image.

3 Results

Our application is in very specific field, as such we had to create our own dataset for training and testing. We tested the Finder and Matcher independently and as a unified system. The relevant metrics for finder were recall and precision. For Matcher we opted for accuracy since there are 100+ instances present.

3.1 Dataset

In making a coaster finder dataset we lacked coasters due to coaster scarcity in Iceland where this project was worked on. To solve this issue the images of coasters were printed on paper to create the percieved appearance of a coaster. The images of coasters were found on a coaster collection database found on the website coasters.eu [?]. The printed images were cut out and roughly sized proportionally to real coasters. These cutouts were placed in various locations within the authors reach and photographed. The total number of coaster cutouts for the training dataset was 120 with some additional photos taken per coaster and 5 empty photos. This provides a dataset of 184 but 2 photos including faces and 4 photos including beer in a glass were included later upon retraining to increase environment specific learning. Heavy augmentation was also used to bring the total image count to 498 images. List of used augments is Flip, 90°Rotate, Crop, Shear, Grayscale, Hue, Brightness, Exposure, Blur, Bounding Box: Rotation, Bounding Box: Shear, Bounding Box: Brightness, Bounding Box: Noise. The augments were applied randomly and automatically using roboflow.

3.2 Finder

To experiment with the finder results a new test dataset was created. The new dataset had 44 images with 50 coasters and various other objects located within certain images. Two models were tested, both were fine tuned on yolov5s. One had 184 images, specifically missing the two images containing faces and four images containing beer. The second model did contain the previously missing images and was heavily augmented. The results are as follows:

version	recall	precision	avg conf.
Synthetic	0.96	0.83	0.74
Augmented	0.9	0.85	0.78

Table 1: Finder results

3.3 Matcher

The testing of different feature extractors was done on the Synthetic dataset with VLAD encoder and BallTree search. Unfortunately we don't have numerical results for different search methods however while implementing them it was clearly apparent that BallTree and KDTree outperformed all other methods in time needed to retrieve, add and initialize the items.

features	accuracy	FPS
SIFT	0.78	1.5
RootSift	0.75	1.4
ORB	0.38	9.1
R-MAC	0.04	0.7
SIFT+Finder	0.96	1.48

Table 2: Matcher feature extractor results

3.4 Finder&Matcher

The testing of fully integrated system was done on the testing dataset of 44 images, same as testing if the Finder. The accuracy achieved with the Finder model trained on the Synthetic dataset was **0.94**.

4 Discussion

The comparison of the features used by Matcher, the basic SIFT outperformed RootSIFT which go against the general consensus on the feature extractors. We have no explanation of the poor results of R-MAC other than incorrect implementation on our side, since general consensus on R-MAC is that it outperforms standard SIFT based features.

As we can see from the results of the Finder, the training on the Augmented dataset reduced recall and improved precision. This behavior might be undesirable in the integrated system since it is better for the Matcher to focus on false positive area rather than to miss the coaster in the scene entirely. The combination of both systems leads to the best results, it allows the Matcher to focus only in potentially interesting areas, as we can se in fig2 where the addition of the Finder narrows area of interest thus decreasing total number of features in the image and prevents matcher from finding an incorrect coaster.

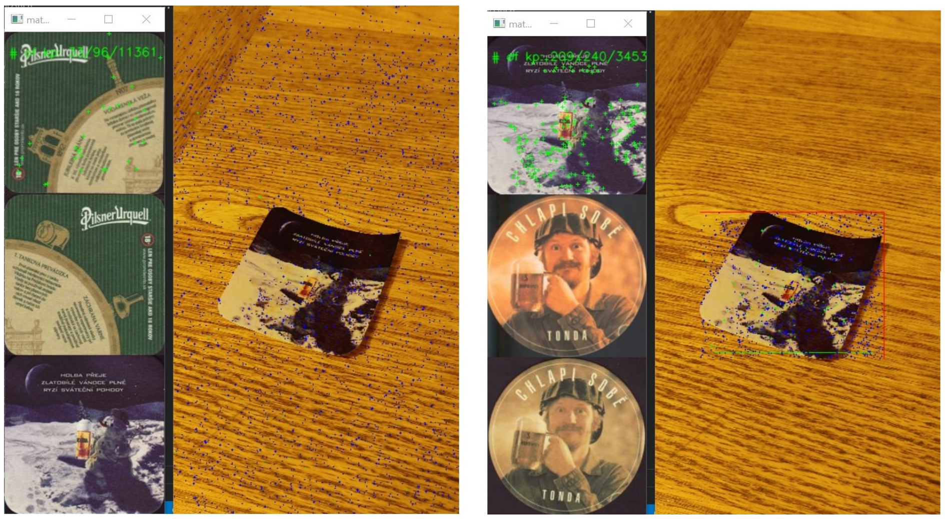
mat...
# Pilsner Urquell /96/11361
Pilsner Urquell
HOLBA PŘEJE ZLATOBILÉ VÁNOCE PLNÉ RYZÍ SVÁTEČNÍ POHODY
mat...
# of kp: 209/240/3453
CHLAPI SOBĚ
TONDA
CHLAPI SOBĚ
TONDA

Figure 2: only Matcher (left), Matcher&Finder (right)

In section 3.3 we also included results of SIFT features combined with the Finder. The use of Finder on the Synthetic dataset are not representative of Finders general performance since it was trained on this dataset. However we might view these as idealized results if the Finder finds the coaster in the image 100% of the time.

5 Conclusion

References