Key-Point based Image Retrieval under ISO Still Image Compression Standards

Sascha Wernegger

July 24, 2016

Abstract

In this paper, we examine the influence of JPEG, JPEG-200 and JPEG-XR on local features. Therefore we use the **VLBenchmarks** [1] software to calculate the performance of a key point detector using a content based image retrieval system. The data set we used is the **Oxford Buildings Dataset** [2]; To compare the influence of compression on the retrieval benchmark, we compress the data set to the different compression formats, and run the benchmark again. We then show the results of our survey.

1 Experiment Setup

First we will show what software we used to compressed our data, then we will show the used parameters and what compression ratio we achieved. We also discuss the fact that the data set is already compressed in JPEG and what influence it has for the survey.
Next we will discuss our setup for the retrieval benchmark, and the results it provides us, so we can compare them. We will also quickly introduce the local feature detectors we tested,and show how we set them up for our experiment.

1.1 Compression

To run our experiment we need to compress the Oxford Buildings Dataset to different formats and different compression ratios. Since both the compression, and benchmark duration heavily depends on the size of the data set we decided to use only a subset of 5062 images of the Oxford Buildings Dataset. In the end we got a random subset containing 597 images and reduced the number of queries from 55 to 18.
For compression we created a script that doubles the compression ratio until the minimum is reached. Now we want to introduce the different command line tools we used to compress the images and what parameter we used. Fore JPEG and JPEG-2000 we used the command line tool from **ImageMagick 7.0.2-5** [3]. We simply used the quality parameter to reduce the filesize. Only for JPEG we added some flags for optimization.
ImageMagick JPEG command:

```
convert <infile> -quality <quality>
  -define jpeg:dct-method=float
  -define jpeg:optimize-coding=off
  <out>.jpg
```

ImageMagick JPEG-2000 command:

```
convert <infile> -quality <quality> <out>.jp2
```

For JPEG-XR we used the **jxrlib 1.1**[4] command line tool as follws:

```
JxrEncApp -i <infile> -q <quality>
  -q <quantization> -o <outfile>
```

For details on the usage of the tools we want to refer to the documentation of the tools.
So we used the compression to generate our different data sets and observed that some of the lowest compressed data sets contained empty images so we had to remove some of the data sets. Finaly we got 25 data sets: 6 compressed to JPEG, 8 compressed to JPEG-XR, 10 compressed to JPEG-2000 and the

[1] http://www.vlfeat.org/benchmarks/
[2] http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/
[3] http://www.imagemagick.org/script/index.php
[4] https://jxrlib.codeplex.com/

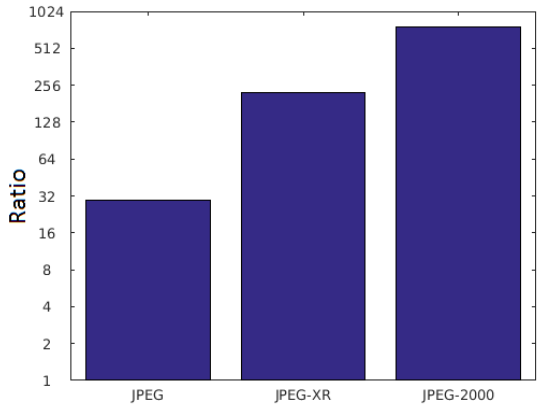


Figure 1: Achieved compression ratio for the different formats

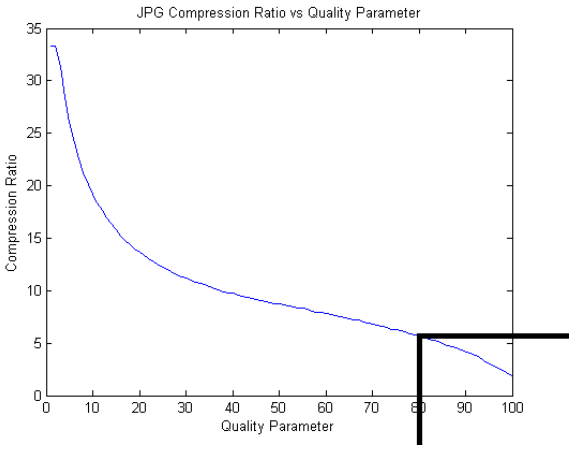


Figure 2: Estimated Quality Parameter

original data set. The following figure [1] shows the maximum achieved compression ratio for the different formats.

The next thing we discuss is the fact that the Oxford Building Dataset is already compressed in JPEG. For that reason we have calculated the average raw size of the images to compare it with the size of the Oxford Buildings Dataset. The average size as a raw image with 24bpp is about 2,296,754 bytes. The average size of an image from the Oxford Buildings Dataset is 387,864 bytes. So we come up with an initial compression ratio of about 5.92. The following figure [2] shows that the quality parameter is set to about 80% which is introducing only little information loss to the image since everything above 90% is loss less anyway, and we have much room left to compress our images.

1.2 Retrieval Benchmark

The VLBenchmatk software for Matlab provides a simple to use retrieval benchmark tool. The retrieval benchmark closely follows the work Jegou et. al [3]. First a set of local features is detected by selected feature extractor and the corresponding frame is calculated. Next the frame is used to compute descriptor for that frame. To find most similar a K-Nearest neighbours search is used to calculate the descriptor distance. Now the knn-descriptors vote for their image based on their descriptor distances, to finally sort the images.

But to compare the performance we have to calculate a value to compare the results. The proposed performance measure for such a retrieval system is the mean average precision(mAP) as described in [5]. To calculate the mAP we first need to calculate the precision and recall for each document. Since there are four categories in the Oxford Buildings Dataset, we will quickly discuss their meaning

- **Good** A nice, clear picture of the object/building.
- **OK** More than 25% of the object is clearly visible.
- **JUNK** Less than 25% of the object is visible, or there are very high levels of occlusion or distortion.
- **BAD** The object is not present.

and usage.

- **Good and OK** Are treated as relevant.
- **JUNK** Is treated as if it is not present in the database
- **BAD** Treated as not relevant.

So the query average precision for a query simply is the average of the precision values from each relevant document in the query. And the mAP for all queries is simply the average of the query average precision.

1.3 Key-Point detectors

The key-point detectors have two purposes:

- Detect key-points
- Compute descriptor

First the image is searched for key-points, mostly edges,corners or blobs. Next a frame for that key-point is computed. The frame created can be of different types: point. circle, oriented circle, ellipse, oriented ellipse. Next the feature is computed using the region defined by that frame. The resulting descriptor is stored as a vector of values, for which the distance can be computed. The following sections list the detectors and parameters used.

1.3.1 Scale Invariant Feature Transform (SIFT)

SIFT is probably the most well known key-point detector. Since SIFT is included in the VLFeat Software and its parameters seemed to work pretty good we used the default parameters for it which you can look up in the documentation for **VLFeat** [5]. The VLFeat SIFT implementation only slightly differs from the original implementation of D. Lowe [4]. We will shortly discuss the different stages of Sift, but for details we want to refer to []For key-point detection the Difference of Gaussian is used. Now frames are computed as oriented circles, therefore the orientation is assigned based on the gradient of the image. Finally the descriptor is calculated to generate a histogram of gradients. The resulting vector contains 128 fields.

1.3.2 Pyramid Histogram Of visual-Words(PHOW)

The PHOW [2] features are a variation of dense SIFT descriptors, extracted at multiple scales. Dense SIFT is a version of SIFT that calculates the descriptors for a window that is sliding across the image. A color version, extracts descriptors on the three color channels and stacks them up. When computing descriptors for many keypoints differing only by their position (and with null rotation), further simplifcations are possible. To reduce the number of descriptors we changed the **Step** parameter to 16.

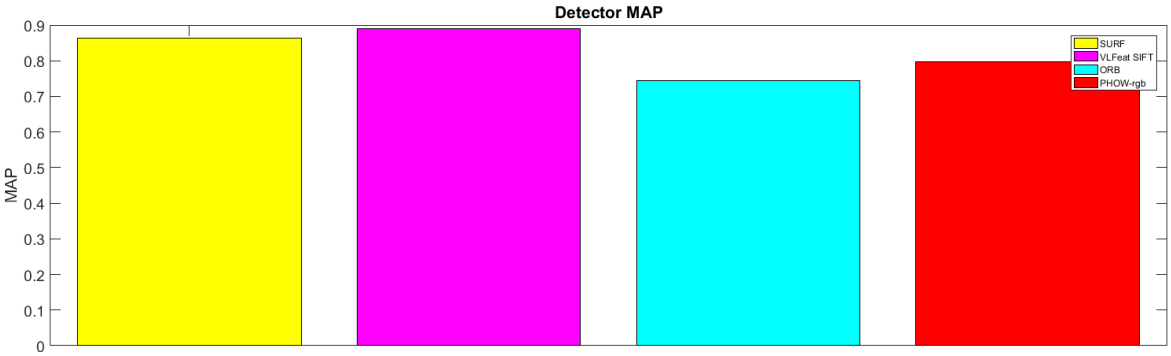
[5] `http://www.vlfeat.org/matlab/vl_sift.html`

1.4 Speed Up Robust Features(SURF)

SURF [1] is a detector similar to sift, but as the name suggests, it is faster to compute. In SURF, square-shaped Filters are used as an approximation of Gaussian smoothing. for key-point detection a Hessian matrix and Non-maximum is used. Orientation assignment uses a Sliding orientation window using Gaussian weighted Haarwavelet responses from circular neighborhood suppression. The descriptor is calculated using an oriented 4x4 grid that defines sub-region. Wavelet responses computed for 5x5 samples of the sub-region. For SURF we reduced the treshold for the keypoint-detection to get more key-points and increased both the number of octaves int he gaussian pyramid from 4 to 8 and also the number of layers per octave from 2 to 4. For details on the parameters we want to refer to the online documentation of OpenCV [6].

1.5 Oriented FAST and Rotated Brief ORB()

ORB [8] uses FAST [6] [7] to detect corners as key-points. The orientation is calculated using the centroid operator. Dimensionality reduction for the descriptors can be achieved by using hash functions that reduce SIFT descriptors to binary strings. The fast key-point detection and small descriptor size makes it well suited for devices with limited resources. The orb detector lets you specify the number of key-points, to limit the number of detected key-points. We decided to use as much descriptors as possible, setting the number of features to 8000 which is about three times the number of features created from the other descriptors, compensating the descriptor size. We also changed the scale factor from 1.2 to 1.1 and increased the number of scale levels to 16 to get more key-points.For details on the parameters we want to refer to the online documentation of OpenCV [7].



2 Results

First we want to show the mAP for the original data set shown in fig. 2. In the next sections we will see how the detectors have performed. Therefore we show plots of the mAP over the compression ratio.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Ninth European Conference on Computer Vision 2006*.

[2] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Image classification using random forests and ferns. In *IEEE International Conference on Computer Vision 2007*.

[3] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Exploiting descriptor distances for precise image search. INRIA, inria-00602325v2, 2011.

[4] David G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 2004.

[5] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. Department of Engineering Science, University of Oxford,Microsoft Research, Silicon Valley, 2007.

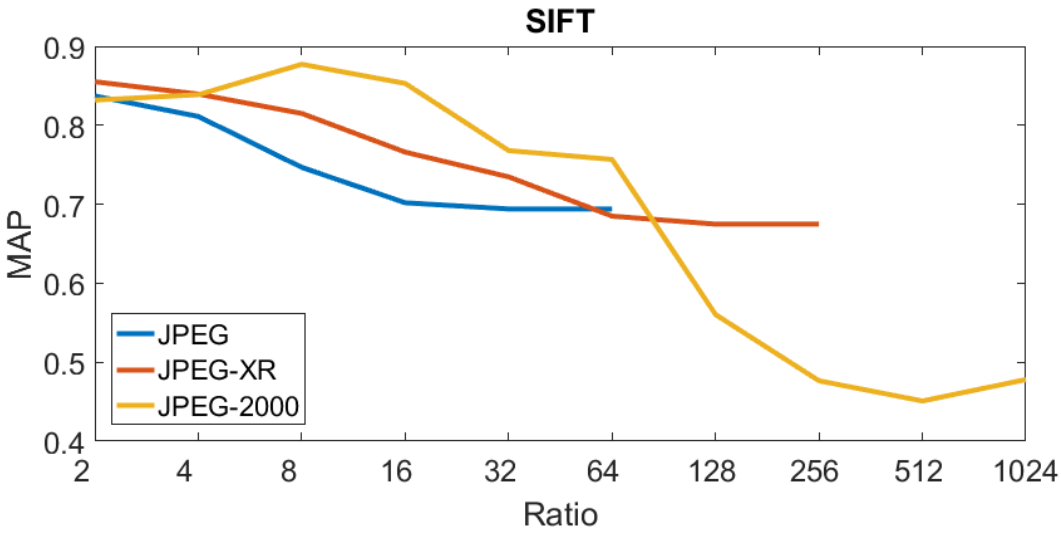
[6] Edward Rosten and Tom Drummond. Machine learning for high speed corner detection. In *9th European Conference on Computer Vision 2006*.

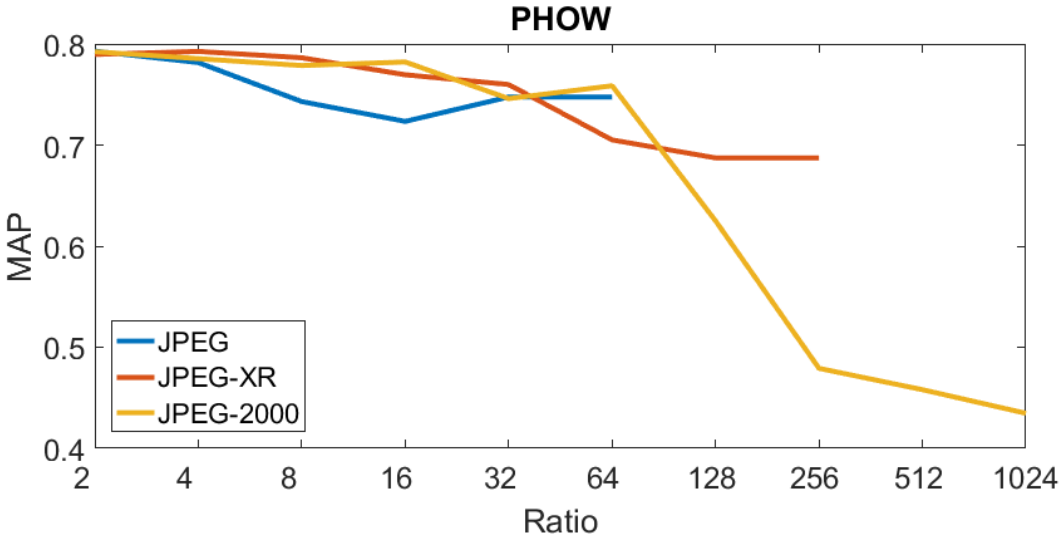
[7] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: a machine learning approach to corner detection. In *IEEE Trans. Pattern Analysis and Machine Intelligence 2010*.

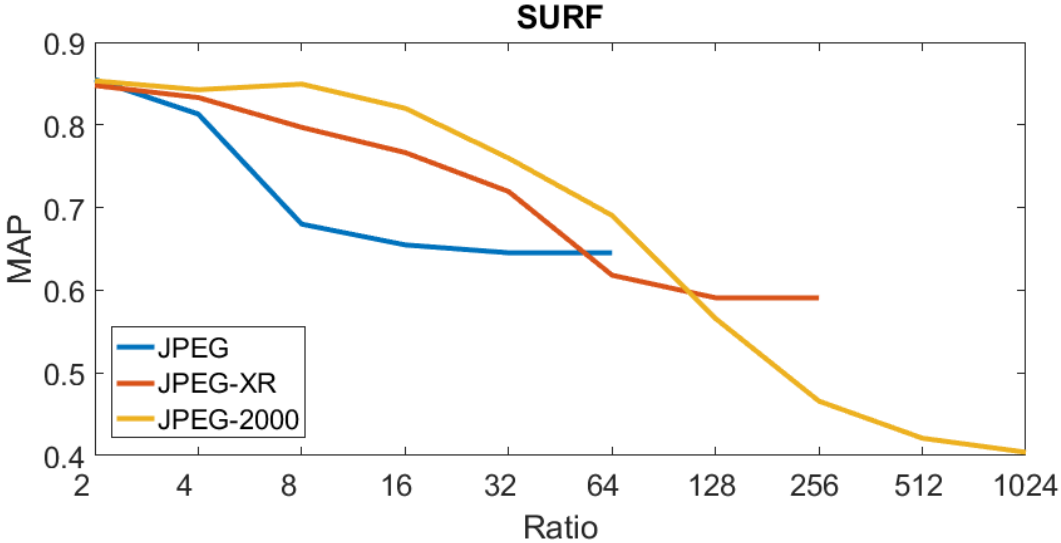
[8] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *International Conference on Computer Vision 2011*.

[6] `http://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html`

[7] `http://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html`


SIFT
MAP
Ratio
JPEG
JPEG-XR
JPEG-2000
0.9
0.8
0.7
0.6
0.5
0.4
2
4
8
16
32
64
128
256
512
1024


PHOW
MAP
Ratio
JPEG
JPEG-XR
JPEG-2000
0.8
0.7
0.6
0.5
0.4
2
4
8
16
32
64
128
256
512
1024


SURF
MAP
Ratio
JPEG
JPEG-XR
JPEG-2000
0.9
0.8
0.7
0.6
0.5
0.4
2
4
8
16
32
64
128
256
512
1024

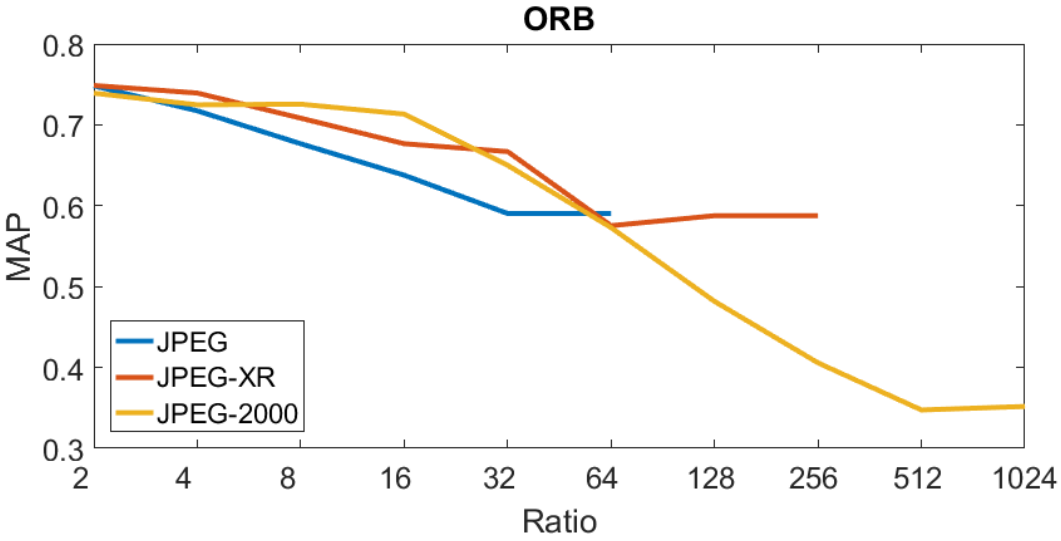


Figure 3: plot of mean average precision over compression ratio for the different detectors and formats