Preliminary test on OpenCV feature implementations

Jonas Arnfred

March 13, 2013

Abstract

1 Project Description

In controlled settings existing face recognition algorithms have achieved good results in both classifying and clustering faces, but often these approaches break down when we consider images from real life application such as cell phone photo albums. Here we encounter non-aligned faces under different lighting conditions that might be partially obscured and turning in different directions. Even recent approaches to recognize non-aligned faces such as [Liao and Jain(2011), pp 9.] don't achieve more than 60% geniune accept rate at 1% false accept rate when comparing faces from LFW[1] database. When we consider face patches not revealing the entire face the performance drops further to around 40% genuine accept rate at 1% false accept rate.

If we relax the problem of classification to the more general problem of clustering it might turn out that this accuracy is sufficient to accurately separate a set of face patches into partitions each ideally only containing pictures belonging to one person.

1.1 A modest proposal

If we consider a graph it is often possible to successfully obtain a partitioning that adheres to a classification of the graph even if the graph has *inter*-cluster edges. This is of course only possible as long as the *inter*-cluster edges outnumber or outweigh the *intra*-cluster edges, but if we can create a link between two nodes around 40% of the time and only have 1% of them be incorrect, we are looking at a very well behaved graph.

To apply graph clustering to image clustering, consider the set of image patches as the set of nodes and the edges between the nodes as the similarity between two image patches. The ideal partitioning of the graph is then one

[1] Labeled Faces in the Wild

where all the nodes representing image patches belonging to a particular person are clustered together in their own partition.

Intuitively this is an easier problem than classification. Even if two images-patches might not resemble each other, they could both happen to resemble a common neighbor and consequently be clustered together.

1.2 How can this be done?

Because images in real world conditions are often taken under changing light and containing partially obscured or half turned faces, we can't require images to be aligned and homogenious in lighting and position in order to use them.

An alternative approach would be to sample characteristics from the face-patch in the form of feature points and compare how well these samples match feature points from other face-patches. If we can describe the feature points in a light and affine transformation agnostic way, ideally we would be able to match them to corresponding feature points on other face-patches.

By comparing feature points we can construct a similarity graph linking the face-patches to each other and use a graph clustering algorithm to obtain the final clusterings.

2 So is it done yet?

Before I present the finished algorithm there are a few practical and theoretical problems that need to be solved first.

2.1 Theoretical shenanigans[2]

Given two sets of feature points and trying to match them, it's important to realize that these two sets might be obtained from face-patches that contain some overlapping characteristics but both might also contain features that are not present in the other patch at all. If one patch contain the face from the eyes and down (the person might be wearing a cap) and the other patch contain everything from the nose and up (the same person might cover their mouth with their hand) we will have feature points from the cap, the hair, the mouth and the hand that will not match at all. This means that when we try to match two face-patches, we need to match them in a way that highlights the points that match well and ignores the points that don't.

I see two ways to possibly approach the problem. One is to find a mathematical method like sub-tree matching or similar approaches to match the two sets of feature points against each other for each image[3]. A different (and in my opinion promising) approach is to forego the matching of two images entirely:

If we consider the entire set of face-patches and the even bigger set of all the feature points, we could potentially match each feature point against the

[2] http://www.youtube.com/watch?v=MFZG8KQJni8

[3] I would be interested in hearing your input on this subject

set of all possible feature points. Instead of basing the edges in the graph on the similarity of two image patches, we base the edge on the feature points that happen to be linked between any two given face-patches when matched against all possible points. A naïve approach is obviously of quadratic complexity in the amount of feature points, which very fast would become unfeasible to compute, but since points that can be compared both euclidean distances and hamming distances form a metric space, a much faster solution might be possible using metric trees like VP and BK trees[4]. In the case of hamming distance I could potentially further Huffman encode the library of feature points to reduce their length without reducing dimensionality.

2.2 Practical considerations

The success of this approach is dependent on feature points being able to describe image keypoints in a light and affine transformation agnostic way. In the paper using an alignment free approach to match images I mentioned before[5] they whip up their own feature descriptor, an approach I'm not personally very convinced of since they never show their own descriptor to be superior to anything else than sift. Given that many new feature descriptors have come out in recent years, it seems to me much more sensible to use an already tested and confirmed approach than to cook up a new method, before I can show that coming up with a new feature descriptor is the only way to solve the problem[6].

Fortunately OpenCV implements a range of feature descriptors[7]. So in order to see if any of these feature detectors might be usable and if yes, which ones, I decided to put them to the test.

3 Experiments

The goal of these experiment is to see if different feature descriptors given real life images would score the similarities between images showing the same face significantly higher than images showing the faces of different people.

3.1 Getting some data[8]

In order to test this I used the LFW dataset[9]. This is a dataset consisting of labeled photos of famous people taken by photographers under different circumstances and in different quality. The faces vary between looking straight at the

[4]See: This answe on stackoverflow.com and Wikipedia (I promise and swear I won't use wikipedia as a source for the thesis!)

[5][Liao and Jain(2011)]

[6]And in this case I'll most likely spend my thesis trying to come up with a feature descriptor instead of solving the original problem

[7]To be precise they include an interface for: SIFT, SURF, ORB, BRISK, BRIEF and FREAK. I have however not been able to get FREAK to work

[8]http://www.youtube.com/watch?v=FQ3UrLQP2ok

[9]Labeled Faces in the Wild

camera and turning their head to either side, but not to the extend of being full profile. The lighting is usually relatively good but very inconsistent as most photos are from different events, sometimes years apart.

This dataset is not a perfect match since the ideal application for this algorithm would be people's own photo albums, where the camera used is consistent and the faces often might be in profile. However it does pose similar challenges in the form of lightning and non-aligned faces that it serves well to give a rough idea of the general performance of different feature descriptors.

3.2 Experiment Setup

In general there are two measures to consider when we compare keypoints in an image; the distance measure and how unique the match is. The distance measure for the keypoint descriptors I have reviewed are either based on euclidean distance or on hamming distance. Once we have matched keypoints of one image to their highest matching counterpart in another image, we can filter these matches either by only including matches that symmetric, i.e. if we swap the two images and run the same matching algorithm we get the same match. Another method proposed in [Lowe(2004)] is to look at the ratio between the best and second best match. If the distances are too similar, the keypoint match is not unique enough and hence discarded.

Since I don't have a solution yet for how best to compare two face-patches, I have created four naïve methods that gives a score for a match of two face-patches given their keypoint descriptors. These methods might not be ideal, but it's my hope that they serve the purpose of rating the similarities between images well enough to evaluate the different types of descriptors. Notice that since a better matching function would be able to discard more bad matches and keep more good matches, the results obtained using my naïve methods sets a lower bound on how well the different descriptors perform. Hopefully this lower bound is equally tight for all of them, but this is by no means a given and my comparison can therefore not be seen as a general evaluation of their performance.

The four matching functions are given as parameters the distance for each keypoint to it's closest match and the ratio for each keypoint between the distance of the closest and second closest match. They work as follows:

- *Simple Mean*: Take the mean of all distances
- *Better half*: Take the mean of the best half of the distances
- *Treshold*: Take the mean of only the points that are both in the better half of distances and ratios
- *Limit to 10*: As above but take only the 10 points with the best distance

I also created another four similarly functions, the only difference being that the mean was over the *ratio* and not the *distance*.

Preliminary Test		Full Test	
Locator	Descriptor	Locator	Descriptor
Sift	Sift		
Surf	Surf		
Orb	Orb	Orb	Orb
Fast	Orb	Fast	Orb
Orb	Brief	Orb	Brief
Fast	Brief	Fast	Brief
Orb	Brisk		
Brisk	Brisk		

Table 1: Keypoints and descriptors used in preliminary and full testing round

To quickly weed out bad performers I designed a small test-set including three people from the LFW testset[10]. For each person 10 images were picked at random and all pairs of images compared using the eight functions mentioned above.

Since I have several different methods for selecting keypoints and several different algorithms for creating descriptors based on the keypoints available in OpenCV. The pairings I tested can be seen in table 1. Generally I used the keypoint locator that was available for the descriptor I was testing, and when this wasn't possible paired the descriptor with a keypoint locator of a similar method.

Upon finding the best performing keypoint locator and descriptor pairs I went on to test them on an imageset containing 60 people with ten images per person, that is 600 images in total.

4 Results[11]

The results from the preliminary testing can be seen in table 2. The difference displayed is the percentage wise difference between the score of pairs of faces belonging to the same person as opposed to pairs of faces belonging to two different people. The score is calculated using either the ratio or the distance with one of the four functions described above.

It is immediately obvious that the distance based measures fare much better than the ratio based measures. In particular the 'treshold' and 'limit10' methods work well. However our main concern is not with how well the different functions perform but rather the performance of different descriptors. Here we see a clear advantage of Orb, Brief and Brisk. Brisk however shows some strange fluctuations and would be surprised if these results were a true indicator of the performance of the descriptor. Brisk is also significantly slower than the others,

[10] Anna Kournikova, Ann Veneman and Bill Gates

[11] http://www.youtube.com/watch?v=tKT-eWMWXOE

	Ratio				Distance			
	Mean	Best Half	Treshold	Limit10	Mean	Best Half	Treshold	Limit10
Sift-Sift	0.4	0.7	0.8	0.4	0.7	-0.4	1.0	1.5
Surf-Surf	0.4	0.7	0.8	1.4	0.9	0.9	0.7	0.9
Orb-Orb	0.6	1.1	1.3	1.1	2.0	0.6	3.8	3.5
Fast-Orb	0.8	1.5	1.7	1.6	2.7	13.2	4.3	4.7
Orb-Brief	0.4	0.9	1.0	0.6	5.8	2.3	8.9	8.8
Fast-Brief	1.1	2.0	2.2	2.0	4.8	11.0	8.3	10.6
Orb-Brisk	1.1	2.0	2.2	2.3	4.8	7.3	6.7	6.8
Brisk-Brisk	0.0	0.0	-0.1	-0.1	0.9	-6.2	1.7	1.7

Table 2: Percentage difference in preliminary testing round

	Mean	Best Half	Treshold	Limit10
Orb-Orb	2.4	3.8	4.1	4.4
Fast-Orb	3.9	5.7	6.2	7.1
Orb-Brief	6.3	9.2	9.3	9.8
Fast-Brief	4.7	7.0	7.3	8.2

Table 3: Percentage difference in full testing round

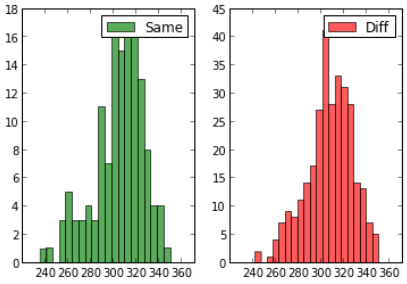
usually taking around 9-10 seconds to calculate similarities between 30 images, something than can be calculated by the other methods in around 0.3 seconds.

The difference in mean doesn't tell us much if we don't know the variance of the individual classifier, since a high variance would mean that we would need an equally big difference to use the similarity score for anything. To illustrate the behavior of the distributions I've made a few plots. In figure 1 eight histograms are shown, each displaying the behavior of a particular keypoint locator-descriptor combination using the same score function (Treshold). When taking the distributions into account it's apparent that the descriptors Orb and Brief are performing about equally well given Orb's lower variance.

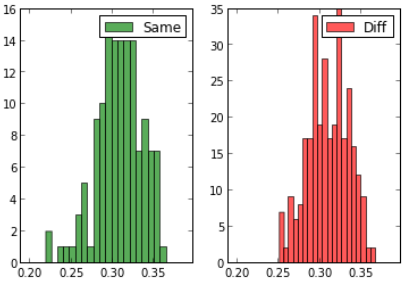
I've included figure 2 to illustrate how Orb-Brief performs over different score functions using the small testset of 30 images. With only 30 images, there is no saying if the scores are just a fluke, or if they will behave consistently at bigger datasets. Figure 3 is included to illustrate the behavior of Orb-Brief when testing on 600 images and 60 people instead. The results are encouraging since they replicate and usual exceeds the preliminary results using only 30 images. In table 3 the results from the simulation on 600 images are displayed.

5 Conclusion and Questions for tomorrow

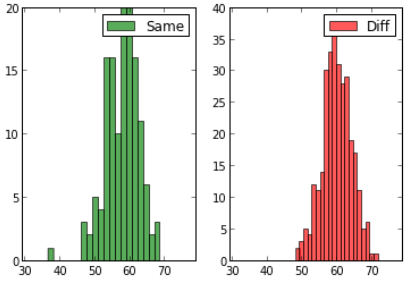
In conclusion I think that both Orb and Brief would be worth testing out for an alignment free clustering approach. The important details I think we should touch upon tomorrow are as follows:



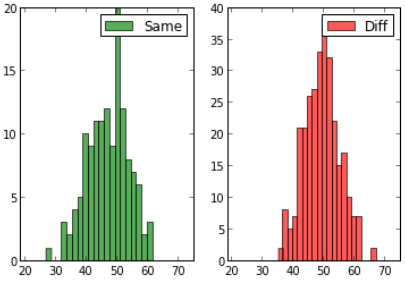
(a) Sift-Sift Treshold: 1.0% diff.



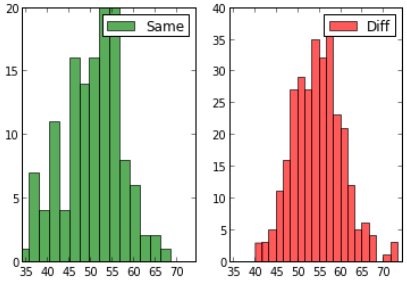
(b) Surf-Surf Treshold: 0.6% diff.



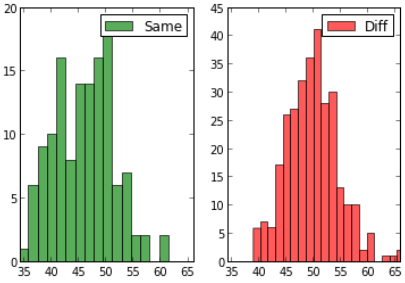
(c) Orb-Orb Treshold: 3.8% diff.



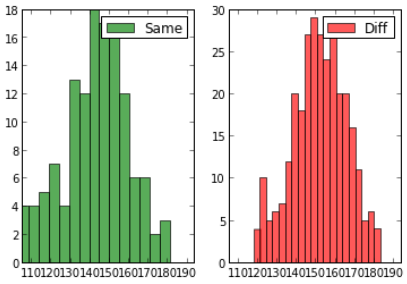
(d) Fast-Orb Treshold: 4.3% diff.



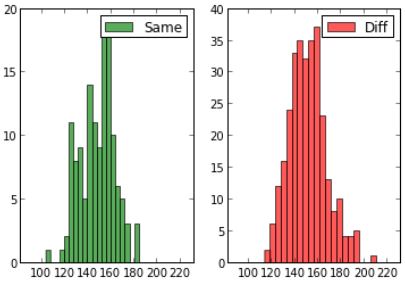
(e) Orb-Brief Treshold: 8.9% diff.



(f) Fast-Brief Treshold: 8.3% diff.

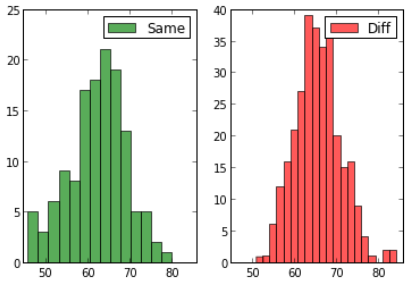


(g) Orb-Brisk Treshold: 6.7% diff.

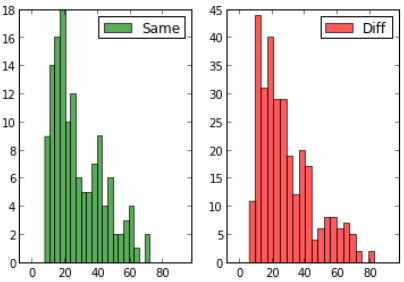


(h) Brisk-Brisk Treshold: 1.7% diff.

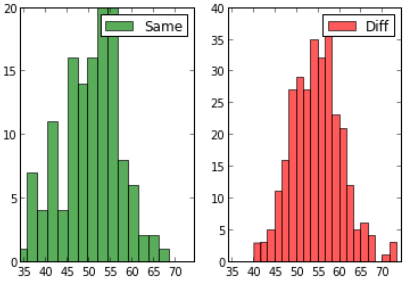
Figure 1: Treshold results on 30 images



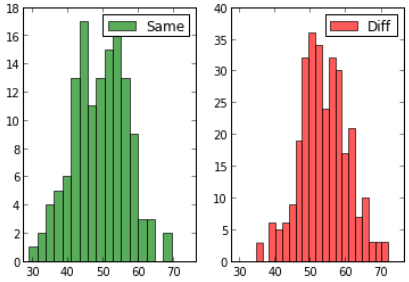
(a) Orb-Brief Simple mean: 5.8% diff.



(b) Orb-Brief Best half: 2.3% diff.

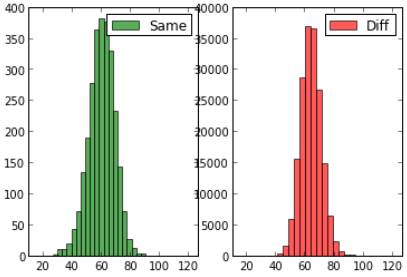


(c) Orb-Brief Treshold: 8.7% diff.

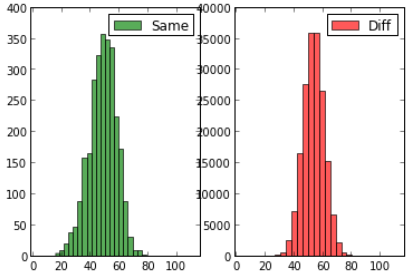


(d) Orb-Brief Limit 10: 8.8% diff.

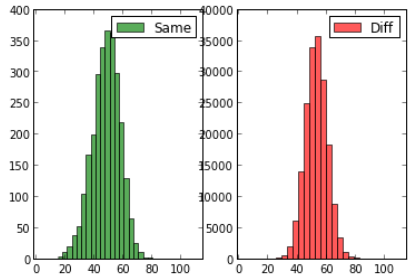
Figure 2: Orb-Brief results on 30 images



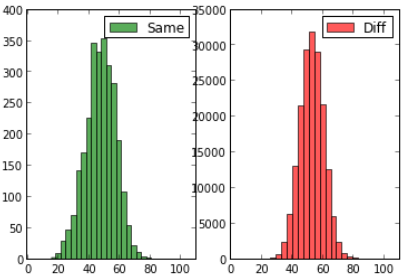
(a) Orb-Brief Simple mean: 6.3% diff.



(b) Orb-Brief Best half: 9.2% diff.



(c) Orb-Brief Treshold: 9.3% diff.



(d) Orb-Brief Limit 10: 9.8% diff.

Figure 3: Orb-Brief results on 600 images

- Do these results overall show enough discriminative power to merit further exploration?
- How can I best compare the keypoints of two face-patches?
- Does my proposal for using Metric trees to circumvent the problem sound doable?
- Is there something super important that I've completely ignored so far? (I'm mostly sitting alone with this, so I need a reality check)

References

[Liao and Jain(2011)] Shengcai Liao and Anil K Jain. Partial face recognition: An alignment free approach. In *Biometrics (IJCB), 2011 International Joint Conference on*, pages 1–8. IEEE, 2011.

[Lowe(2004)] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.