Image Specific Feature Similarities

Ido Omer and Michael Werman

School of Computer Science,
The Hebrew University of Jerusalem,
Jerusalem 91904, Israel
{idom, werman}@cs.huji.ac.il

Abstract. Calculating a reliable similarity measure between pixel features is essential for many computer vision and image processing applications. We propose a similarity measure (affinity) between pixel features, which depends on the feature space histogram of the image. We use the observation that clusters in the feature space histogram are typically smooth and roughly convex. Given two feature points we adjust their similarity according to the bottleneck in the histogram values on the straight line between them. We call our new similarities *Bottleneck Affinities*. These measures are computed efficiently, we demonstrate superior segmentation results compared to the use of the Euclidean metric.

1 Introduction

Calculating a similarity measure between pixels is a fundamental step in many computer vision and image processing algorithms. Many of these algorithms depend on a reliable affinity (or distance measure) between pixels for their calculations. The affinities are either measured between different pixels of the same image in case of segmentation and edge detection, or between pixels from neighbouring frames in case of optical flow calculation, motion segmentation and tracking.

In most of these applications, pixel affinity is calculated as a simple function of the Euclidean distance between the pixels' features (usually $e^{-distance^2}$) in some feature space. Common feature spaces are the one-dimensional gray scales feature space, two or three dimensional colour spaces and higher dimensional ($\sim 50D$) texture feature spaces. Different researches suggest using different feature spaces for achieving optimal results in various applications, but no particular feature space is considered optimal by the whole community (A survey of the properties of different colour spaces for image segmentation can be found in [1], while [2] provides a basic survey of different texture features). Other approaches include learning pixel affinity and feature space clustering.

Fowlkes et al. suggested a high level approach for learning pixel affinity calculations using a dataset of human segmented images as ground truth [3]. Their approach for affinity calculation uses the combination of several feature spaces and information from the image itself (edges) through a high level learning mechanism. While the approach is suitable for segmentation and similar time

A. Leonardis, H. Bischof, and A. Pinz (Eds.): ECCV 2006, Part II, LNCS 3952, pp. 321–333, 2006.

consuming applications, it is less suitable for online applications or other computationally efficient applications. Another drawback is the generalization of the method, which is not straightforward. For example, generalizing the approach to handle affinity calculations between pixels in successive frames requires massive human assistance.

A different approach for providing pixel affinity is by using feature space clustering [4]. This approach tries to exploit image specific characteristics rather than learn a general rule for the affinities calculations. Although this approach can be efficient and easy for generalization, it implies clustering of the feature space (hard or soft) which is prone to errors due to noise and other difficulties.

We claim that given two feature points, u and w, with equal Euclidean distances from a third feature point v, it stands to reason that if w and v share the same cluster, while u is located within another cluster, the affinity of w and v is larger than the affinity of u and v.

The motivation for our approach is obvious when looking at the synthetic one-dimensional feature histogram in Figure 1. Our main observation is that the histogram provides us with the additional knowledge that the feature values belong to two different Gaussian distributions. While v and w are very likely belong to the same source and should be considered similar, v and u seem to belong to two different sources and should be considered dissimilar.

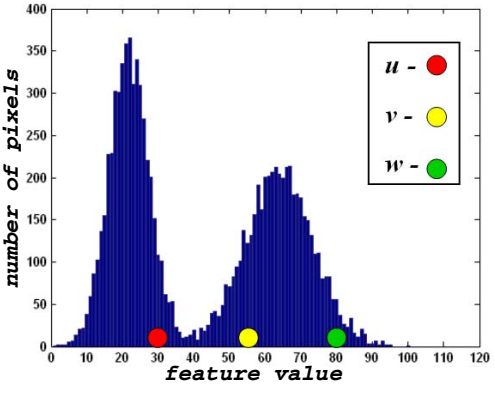


Fig. 1. (a) An example of a one-dimensional features histogram. Given the above histogram, it stands to reason to claim that a pixel having a feature value of v is similar to a pixel having a feature value of w and dissimilar to a pixel having a feature value of u although the Euclidean distance between the feature values v and w is identical to that of v and u.

This work suggests a straightforward and efficient approach to affinity calculations that exploits image specific attributes while not explicitly applying clustering of the feature space. We do so by introducing the *Bottleneck Affinities* - a simple mechanism for estimating the likelihood that two feature points belong to the same cluster in the feature space. We estimate this likelihood by

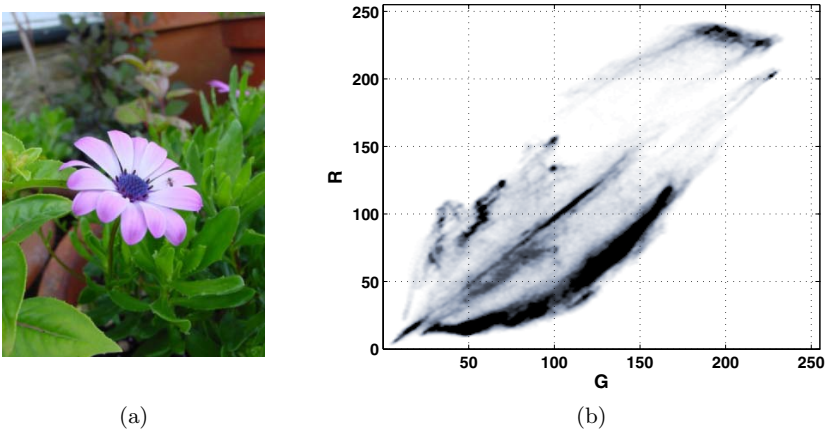
analyzing the histogram values on the straight line connecting the two feature points in the feature space histogram. A sparsely populated region along this line (in the histogram domain) is considered a "bottleneck" and indicates that the two points probably belong to two different clusters. We therefore decrease their affinity measure. Similarly, a densely populated line indicates that the points belong to the same cluster and we therefore increase their affinity measure.

Our approach utilizes the typical smooth and convex structure of clusters in the *RGB* histogram of images. This structure is the result of scene properties and the (digital) image acquisition process. We discuss the structure of the clusters in the next section (section 2). Section 3 describes our algorithm and discusses implementation issues. The results are shown in section 4, while section 5 summarizes and suggests possible extensions to this work.

2 Histogram Clusters

This section discusses the physical properties that affect cluster structure in an image feature space histogram.

Figure 2 shows a simple image, containing a small number of dominant colours along with a two-dimensional projection of its RGB histogram. The difficulty in modelling the clusters in the histogram domain is evident from this example. The clusters have no particular shape, and methods like the *Gaussian Mixture Model* are not suitable for this kind of problem. The large amount of noise makes the task of clustering a difficult one even for this simple scene. Nevertheless, it is obvious that the histogram contains different clusters. For most pairs of feature points, the problem of estimating a likelihood measure for the points to belong to the same cluster seems significantly easier than the actual clustering problem. Our algorithm is aimed at utilizing this observation.



(a) (b)

Fig. 2. (a) Sample image and (b) its RG histogram (a projection of the *RGB* histogram upon the RG axis), darker colour represents a denser histogram bin

For the sake of simplicity, this discussion refers to three-dimensional colour features (RGB values). We will address the generalization of the discussion to other feature spaces at the end of this section.

Our algorithm takes advantage of three well studied image properties. The first property is the *piecewise smooth world* assumption which has been used before in many computer vision and image processing applications such as boundary detection [5], image segmentation [6], noise reduction [7] and more. The second property is that monochromatic scene objects create nearly convex elongated clusters in the RGB histogram [8]. The third property is image blur due to the optics of the camera and the finite size of the pixel [9].

The first and second properties imply that for two feature points belonging to the same monochromatic object, all the bins along the line connecting them (in the histogram domain) are populated. Due to the third property, the same holds for textured objects as well. The justification for the last claim lies in the following fact: While locally, in the image plain, there is no difference between the blurring of edges due to texture and due to boundaries, globally - in the histogram domain there is a big difference between these phenomena. Boundaries between objects are scarce and therefore produce a small number of interpolated values. The line in the histogram between pixels from neighbouring objects is therefore scarcely populated. In textured regions the same texture components (texels) are blurred repeatedly. The line between pixels from two different texels (of the same texture) in the histogram is well populated.

Figure 3 demonstrates the difference in the histogram domain between edges due to object boundaries and edges due to texture. In the figure we show real images of a synthetic scene along with their GB histogram (a projection of their RGB histogram upon the GB plane). There are only a few pixels with interpolated values in figure (a) and the two clusters are well separated in the histogram domain (b). In figure (c) many of the pixels have interpolated values and the region between the two clusters in the histogram domain (d) is densely populated. Figure 4 shows an image along with the edge maps according to the Euclidean metric (b) and to our bottleneck distance measure (c). For visualization purposes we show the square root of the edge maps (the difference is visually prominent when looking at the squared root). Both edge maps were normalized to the range of [0..1]. Notice how in the bottleneck edge map boundaries between objects are maintained while the intensity of edges due to texture (the sculptures surface) is decreased.

Refining Our Assumption: In order for our approach to separate feature points only when they belong to different clusters in the feature space histogram, the cluster should be convex. In real life, these clusters are usually not entirely convex as can be seen if figure 2, and one could claim that our heuristic should fail. Fortunately, although not entirely convex, the clusters are usually convex in a small neighborhood in the feature space histogram and are therefore locally convex. Since most applications calculate affinity only in limited neighborhoods around pixels, our heuristic rarely fails. The justification lies in the smoothness assumption. Due to this assumption, neighboring pixels that belong to the same

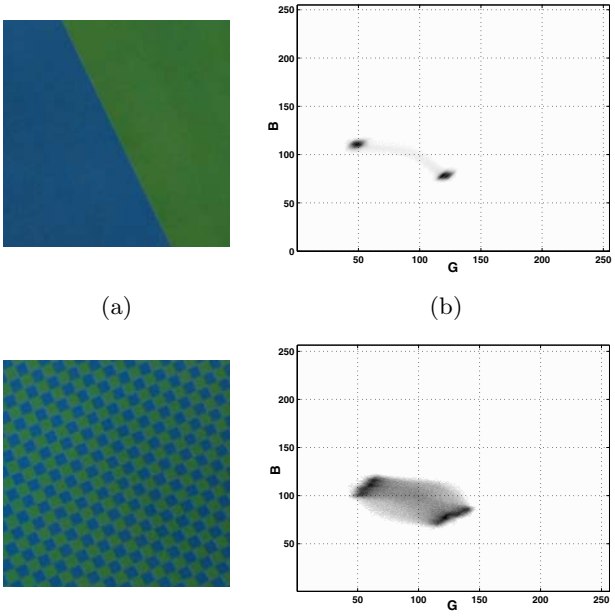


Fig. 3. (a) An image of a synthetic scene containing two objects and its GB histogram (b). (c) An image of a synthetic scene containing a texture (two *texels*) and its GB histogram (d).

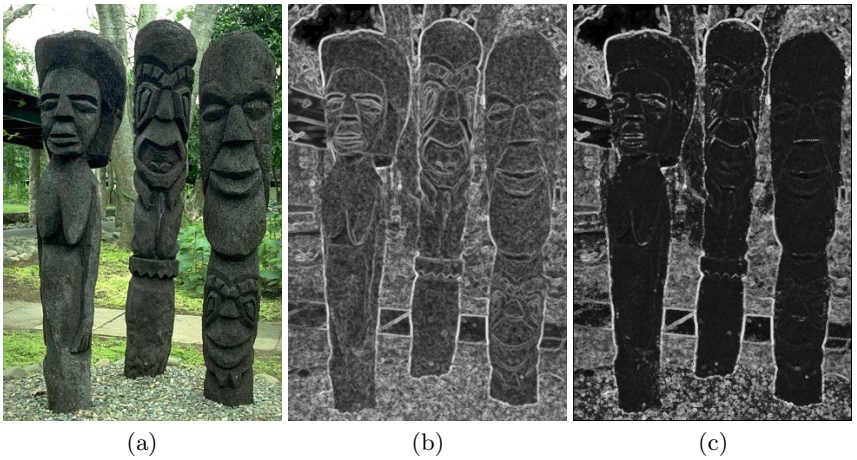


Fig. 4. (a) An image with the squared root of its Euclidean edge map (b) and of its bottleneck edge map (c) (the differences are more easily seen when looking at the squared root). The values in both maps are normalized to the range of [0..1]. Notice how the bottleneck edge map maintains edges between objects while the edge values inside the sculptures are significantly lowered.

object (or piece) have similar features (the changes are smooth within the object) and therefore reside in the same locally convex region of the cluster. Our empirical results support this claim.

Although in the entire section we referred to *RGB* features, we would like to point out that our main argument, smoothness due to scene and camera properties, is a fundamental property of natural images and is not related to a specific set of features. Our experimental results support this observation as we clearly show in the results section.

3 Defining and Computing the Bottleneck Affinities

We implemented a simple and efficient algorithm that utilizes our observation of the typical structure of clusters in the feature space histogram for calculating the *bottleneck affinities*. Given two feature points, our distance measure is the Euclidean distance between the points multiplied by a *Bottleneck factor (bnf)*. This factor receives a low value ($bnf < 1$) for points which our algorithm decided are likely belong to the same cluster and, a high value ($bnf > 1$) for points which our algorithm decided belong to different clusters.

Given two feature points p_1 and p_2, a feature space histogram H, and $L(p_1, p_2)$ - the straight line connecting the two feature points in the histogram domain, we calculate the bnf according to the following formula:

$$bnf(p_1, p_2) = \frac{2min(H(p_1), H(p_2))}{2min(H(L(p_1, p_2))) + min(H(p_1), H(p_2))}$$

Where $H(p)$ is the histogram value at the bin whose coordinates are given by the feature point p and $min(H(L(p_1, p_2)))$ is the minimum histogram value along the line connecting the two points in the histogram domain (excluding the value at the two end points). The term $min(H(p_1), H(p_2))$ was added to the denominator for stabilization reasons. We chose the exact criteria for calculating the bnf due to its simplicity. Our experience shows that other, similar formulas produce very similar results.

A more thorough analysis of the histogram values along the line may produce yet better results, the exact formula may depend on the sparseness/denseness of the histogram, the amount of noise and other factors, but as we show in our results section, even this simple criterion yields very good results.

We believe that the **main contribution** of this paper is in introducing a new approach to computing the affinities rather than in the specific formula suggested.

We implemented the algorithm as a C routine that is called from Matlab (mex file). Since Matlab is our development environment, our implementation is only suitable for algorithms that calculate affinities or distance measures for the original input image like *Normalized Cuts (Ncut)* [10] and other spectral clustering algorithms. The implementation is not suitable for algorithms that iteratively change pixel values like the $MeanShift$ algorithm [11] since we can not efficiently update our histogram dynamically in each iteration. Nevertheless

we are confident that given a dynamic implementation of the histogram (for example in C++) iterative algorithms may benefit from our affinity calculations as well.

Since high dimensional histograms are extremely sparse, our method can not be applied directly for modifying affinity calculations of texture features. Texture is usually represented as a feature vector, holding the response of the pixel's neighbourhood to a large number of filters (typically, around 50). Even using a quantization, allowing only four possible values for each coordinate we get a feature histogram with 4^{50} bins and we hardly ever get two pixels in the same bin. We address this problem by projecting the high dimensional feature vectors onto their first three principal components. Even with this drastic dimensionality reduction, using our affinity measures with the projected histogram dramatically improves the segmentation results compared with those achieved by using Euclidean affinities in the full dimensional space (using Euclidean distance in the projected subspace produced poor results).

Computationally, calculating the histogram distance between two points, p_1 and p_2, in the feature space is linear in n - the number of bins along the line connecting the points in the histogram domain. Since most of the applications calculate distances (or affinity) only in a small neighbourhood, and neighbouring pixels tend to be similar, in average, n is very small. The average computational time for computing the affinity between one pixel and the rest of the image in a 480*320 image is 2 seconds on a Pentium4 2.4Ghz computer. Constructing a Matlab sparse matrix with histogram affinity scores of 11*11 neighbourhood around each pixel in an image of the same size took an average of 200 seconds, 80 of which were spent on the actual affinity calculations. For comparison, building the same matrix with Euclidean affinity scores took around 140 seconds, 24 of which were spent on the actual affinity calculations. In addition, the actual segmentation process (given the affinity matrix) took at least twice that time ($\sim$ 7 minutes), the computational overhead due to our method is therefore nearly negligible.

When calculating affinities between texture features, our approach even proved marginally more efficient than calculating the Euclidean distance, since we work in a projected three dimensional subspace, while the Euclidean distance was calculated in a much higher dimension.

4 Results

We demonstrate the competence of our pixel affinity through the results of segmentation algorithms using both color and texture features. We chose the *Ncut* algorithm for our demonstration since it is a well known image segmentation algorithm that is easily adopted to use with various affinity measures, affinities were computed in an 11*11 neighbourhoods around each pixel. We ran the algorithm twice over the Berkeley segmentation dataset, once using Euclidean affinities and the other using our *bottleneck affinities*. We provide both the Berkeley segmentation benchmark results for both runs and a few qualitative examples.

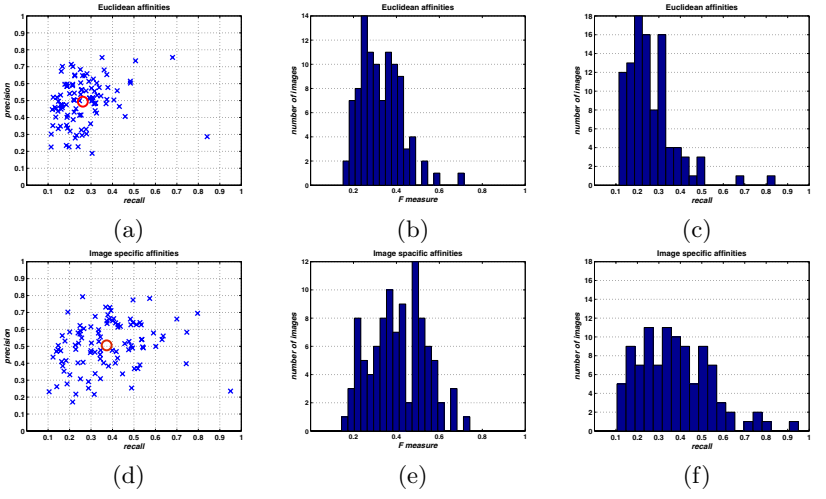


Fig. 5. Benchmark results of the *Ncut* algorithm using color features and Euclidean affinities: (a) precision/recall segmentation results (mean precision/recall in red circle), (b) F-measure histogram and (c) recall histogram. Benchmark results of the *Ncut* algorithm using color features and bottleneck affinities: (d) precision/recall segmentation results (mean precision/recall in red circle), (e) F-measure histogram and (f) recall histogram.

The Berkeley segmentation dataset contains 100 test images. All images in the dataset were manually segmented by humans and these segmentations are considered as ground truth. Berkeley's benchmark tool calculates *precision*, *recall* and *F-measure* (the harmonic mean of the precision and recall scores) for the automatically segmented images according to these human segmented images. Figure 5 provides benchmark results for images segmented by the *Ncut* algorithm using color (*RGB*) features. The results in the figure are: Precision/recall graph (a,d), the F-measure histogram (b,e) and the recall histogram (c,f). Using our affinity measure improved the F-measure from 0.33 to 0.41 - an improvement of 24%. The recall rate improved from 0.26 to 0.37 - an improvement of 42%. The improvement in precision was only marginal, from 0.49 to 0.50 - an improvement of 2%. Using *bottleneck affinities* produced better segmentation results (compared to using the Euclidean metric) for 82 out of the 100 images in the dataset. Figure 6 provides benchmark results for images segmented by the *Ncut* algorithm using texture features. The texture features used are the Leung-Malik filter bank [12] (a total of 48 filters). We also tried using the Schmidt filter bank [13] (a total of 13 filters) but received inferior results. The code for both filter banks was obtained from [14]. The results in the figure are: Precision/recall graph (a,d), the F-measure histogram (b,e) and the recall histogram (c,f). Using our affinity measure improved the F-measure from 0.2 to 0.28 - an improvement of 39%. The recall rate improved from 0.15 to 0.24 - an improvement of 56%. The precision rate has improved from 0.34 to 0.39 - an improvement of 13%. Using *bottleneck affinities* produced better segmentation results (compared to

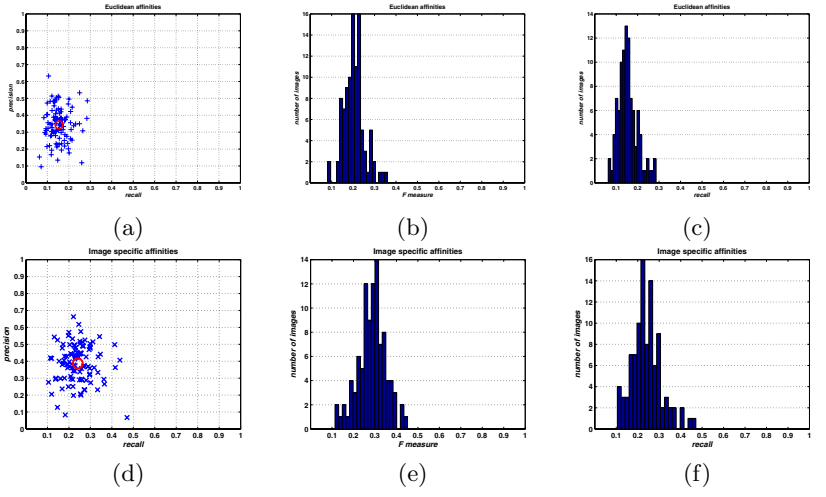


Fig. 6. Benchmark results of the *Ncut* algorithm using texture features and Euclidean affinities: (a) precision/recall segmentation results (mean precision/recall in red circle), (b) F-measure histogram and (c) recall histogram. Benchmark results of the *Ncut* algorithm using texture features and bottleneck affinities: (d) precision/recall segmentation results (mean precision/recall in red circle), (e) F-measure histogram and (f) recall histogram.

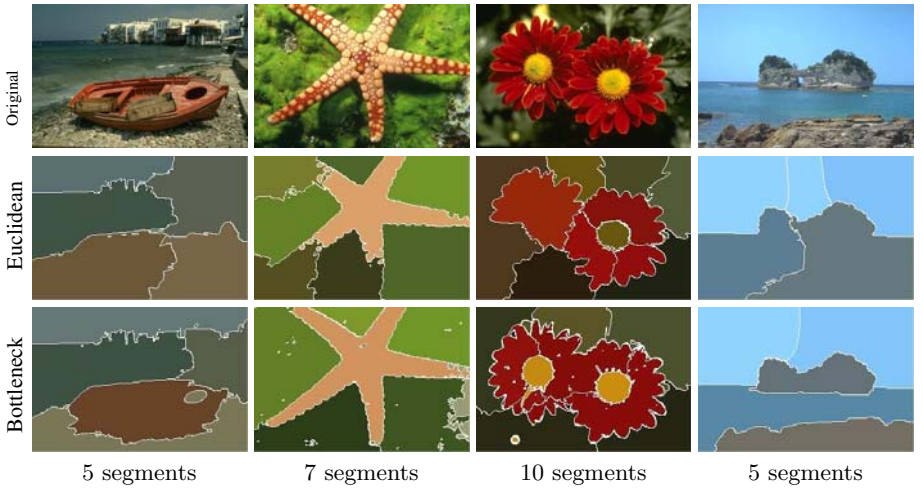


Fig. 7. Segmentation results of the *Ncut* algorithm using texture features in 15*15 neighbourhoods (each cluster is colored using its mean intensity) according to Euclidean affinities (second row) and to our bottleneck affinities (third row)

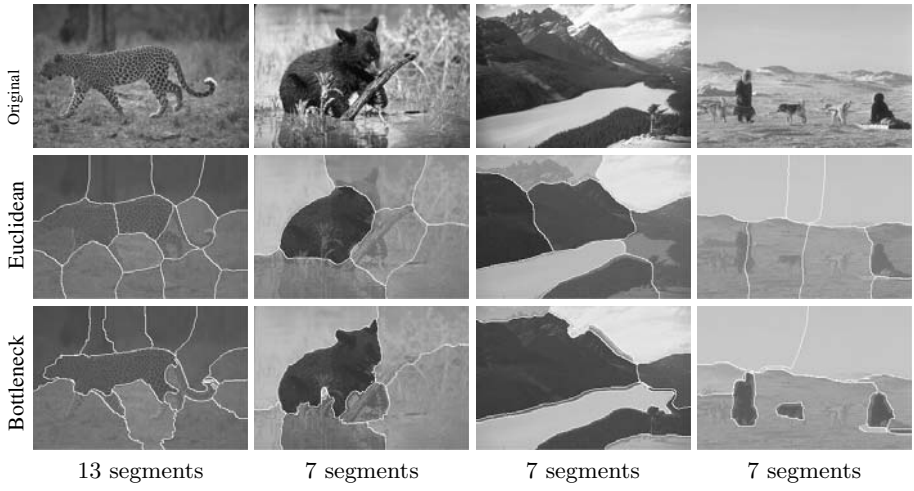


Fig. 8. Segmentation results of the *Ncut* algorithm using texture features in 15*15 neighbourhoods (each cluster is colored using its mean intensity) according to Euclidean affinities (second row) and to our bottleneck affinities (third row)

using the Euclidean metric) for 92 out of the 100 images in the dataset. It is important to mention that these results were achieved for the filter responses alone, without incorporating gray-scale information, hence they are not high.

Figure 7 provides a comparison of segmentation results achieved using the *Ncut* algorithm using RGB features. Results achieved using the Euclidean metric are shown in the second row, while those achieved using bottleneck affinities are in the third. Figure 8 provides a comparison of segmentation results achieved using the *Ncut* algorithm using texture features. Results achieved using the Euclidean metric are shown in the second row, while those achieved using bottleneck affinities are in the third.

We used Matlab's graph-partitioning algorithm for providing an additional evaluation of our *bottleneck affinities*. Graph vertices represented image pixels and edge weights were calculated as the sum of distance in the image plane and color-features dissimilarity (according to the Euclidean metric and to the bottleneck affinities). The algorithm is computationally expensive in terms of both time and memory, since it requires building a full graph. We therefore do not provide full benchmark results for that algorithm, rather we provide a few examples. The results are found in Figure 9. All the images in this experiment were segmented to up to 8 segments.

We did not compare our algorithm to the Fowlkes et. al. algorithm because the approaches are entirely different; our approach is a low level one that works on a single feature space, while Fowlkes et. al. use a high level approach that combines cues from several feature spaces and from the image plane itself. Moreover, their approach can use our algorithm as a subroutine. We also did not compare our approach to that of feature space clustering since this approach is rarely used and our past experience with it produced inferior results.

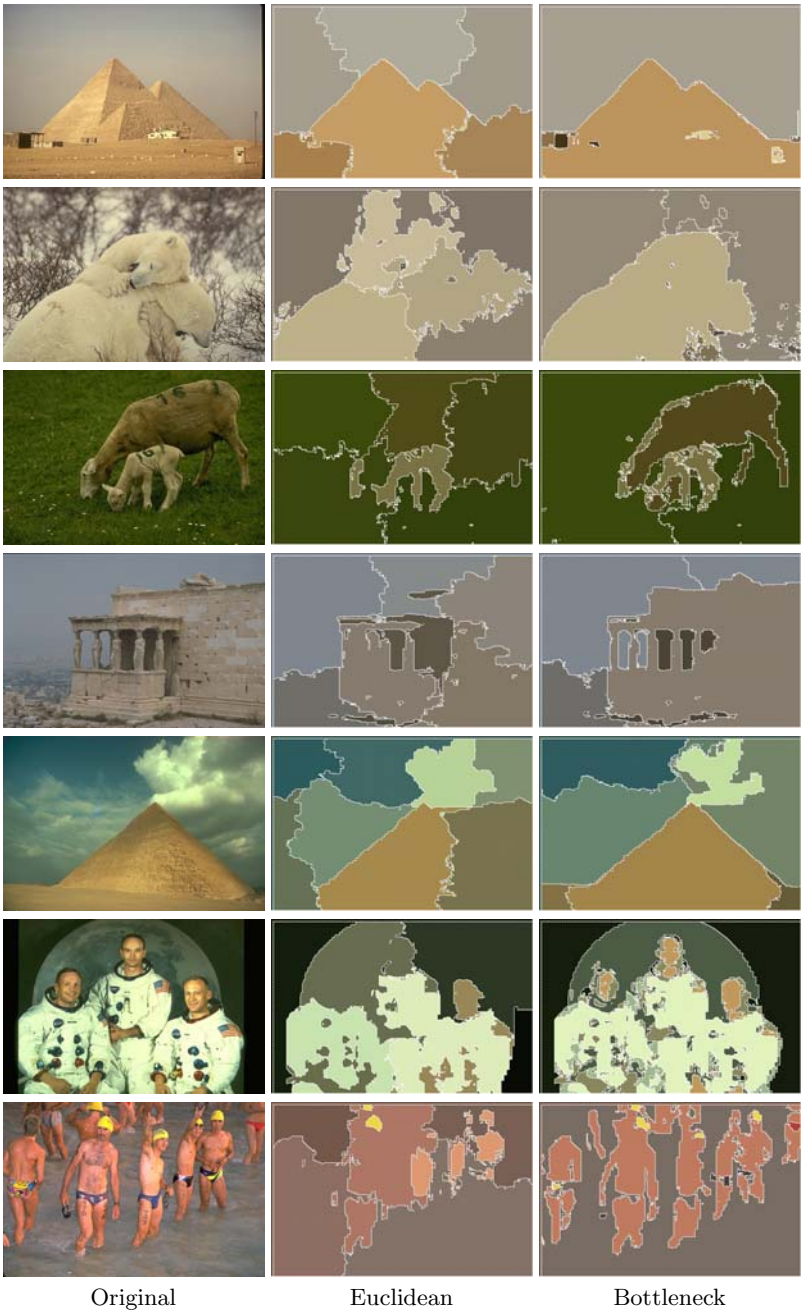


Fig. 9. Segmentation results of Matlab's average-link graph partitioning algorithm using *RGB* features. Results achieved using the Euclidean metric are in the second column. Results achieved using the bottleneck affinities are in the third column. All images were automatically segmented into up to 8 (not necessarily continuous) segments.

5 Discussion and Future Work

We introduced *bottleneck affinities*, a straightforward and efficient approach that utilizes image specific characteristics for calculating similarity measures between pixel features. Our algorithm decreases the affinity between two feature points when it estimates that they belong to two different clusters, while increasing their affinity when estimating they belong to the same cluster. We do so without explicitly clustering the data and with only weak assumptions on the structure of these clusters.

Although we have justified our approach with the claim that the data is both smooth and nearly convex in nature, we believe that for most applications the smoothness requirement is the important of the two, since for smooth data, linearity in a small neighbourhood is obtained automatically according to Taylor's theorem and most applications calculate affinity only in a small neighbourhood around pixels.

We demonstrated the advantages of our affinities compared to the Euclidean distance measure for segmentation both in a three-dimensional color space and in a high dimensional texture space. The improved segmentation results were achieved for only a small additional computational cost, compared with the use of the Euclidean metric, in the case of the three-dimensional colour features. In the case of the high-dimensional texture features our algorithm proved slightly more efficient than the Euclidean metric. We are confident that other applications that rely on pixel affinity measures will benefit from our algorithm.

Better results may be obtained through using a more thorough analysis of the feature space and allowing for more general paths between the feature points (and by this, giving up the convexity requirement). Representing the feature space using a graph, where vertices store density measurement of the neighborhood around each feature point and edges represent Euclidean distance between neighboring features enables calculating (dis)similarities using shortest path algorithms that consider both the paths's length (distance) and the density along the path in the feature space. We are currently working in this direction and already achieve better results but for the cost of a larger computational time. We currently studying how to combine the density information with the distance information and seeking for an efficient algorithm to do so. The graph representation has other advantages as well, probably the most important of which is that it enables working in an arbitrary high dimension without difficulties, which is useful for calculating affinities between texture features.

We further believe that the method may be applied to clustering and affinity measuring for different kinds of data and we intend to try it in different domains.

References

1. Cheng, H.D., Jiang, X., Sun, Y., Wang, J.: Color image segmentation: advances and prospects. Pattern Recognition **34** (2001) 2259–2281
2. Tuceryan, M., Jain, A.K.: Texture analysis. (1993) 235–276

3. Fowlkes, C., Martin, D., Malik, J.: Learning affinity functions for image segmentation: Combining patch-based and gradient-based approaches. Computer Vision and Pattern Recognition (CVPR) (2003)
4. Comaniciu, D., Meer, P.: Robust analysis of feature spaces: color image segmentation. In: CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), Washington, DC, USA, IEEE Computer Society (1997) 750
5. Mumford, D., Shah, J.: Boundary detection by minimizing functionals, I. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition. (1985) 22–26
6. Vese, L.A., Chan, T.F.: A multiphase level set framework for image segmentation using the mumford and shah model. International Journal of Computer Vision **50** (2002) 271–293
7. Spokoiny, V.: Estimation of a function with discontinuities via local polynomial fit with an adaptive window choice. Annals of Statistics (1998) 1356–1378
8. Gevers, T., Smeulders, A.W.M.: Color-based object recognition. Pattern Recognition **32** (1999) 453–464
9. Park, S., Schowengerdt, R.: Image sampling, reconstruction, and the effect of sample scene phasing. Applied Optics **21** (1982) 3142–3151
10. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) (2000)
11. Comaniciu, D., Meer, P.: Mean shift analysis and applications. In: ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2, Washington, DC, USA, IEEE Computer Society (1999) 1197
12. Leung, T., Malik, J.: Representing and recognizing the visual appearance of materials using three-dimensional textons. Int. J. Comput. Vision **43** (2001) 29–44
13. Schmid, C.: Constructing models for content-based image retrieval. In: International Conference on Computer Vision & Pattern Recognition. (2001)
14. (http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html) Visual Geometry Group, University of Oxford.