A Machine Learning Approach to Solving the Correspondence Problem in Computer Vision

Ronald Cruz and Paul Fisher

December 19, 2017

Abstract

The correspondence problem in computer vision refers to matching regions in pairs of stereo images to predict depth from 2-dimensional images. In this paper we approach the matching of pixels as a learning problem. We attempt to learn the matching of two pixels by modeling the correspondence problem as a binary classification problem. We then derive the displacement between corresponding pixels in the two images using a linear search to find the most confident result.

Introduction

The correspondence problem is an open problem in computer vision that refers to the matching of pixels from one image to another which differ due to slight variations in motion and/or camera movement. In tackling this problem, a smoothness assumption is made which assumes that neighboring pixels are likely to have similar *disparity*. The disparity of a pixel is denoted as the displacement of the pixel's position from one image to the other. Historically, there have been two methodologies that pushed the progress made in the correspondence problem, that of *local* and *global* methods.

Local methods utilize a window around a pixel to infer contextual information about an image. The idea behind these methods is not to match pixels themselves, but small patches of the region. The majority of current local methods utilize *adaptive* window sizes and *weight functions*. Adaptive window sizes is a technique of sampling many windows around different centers to exclude regions where disparity may be discontinuous. Weight functions determines the influence of a pixel when considering it as a match. Finding a meaningful weight function directly correlates to the performance of the local methods. Two examples of successful weight functions are the geodesic weight function [3] and a guided cost filter [4], both produced by Hosni et. al. The benefits of these proposed approaches allow for locally accurate and near real-time results.

Although local methods can achieve powerful results, they also suffer from several drawbacks. Some of these drawbacks include a poor response to noise, inability to discern untextured regions such as solid colored walls, and difficulty dealing with the occlusion of pixels. To combat these drawbacks, global methods are utilized.

Instead of implicitly relying on the smoothness assumption like local methods, global methods make this assumption explicit. Global methods define an energy function

$$E(D) = E_{\text{data}}(D) + \lambda * E_{\text{smooth}}(D)$$

that measures the quality of a disparity mapping from two images [2]. λ is a user defined parameter, $E_{\text{data}}(D)$ defines a metric for photo-consistency, and $E_{\text{smooth}}(D)$ is the explicit assumption of smoothness. This approach results in an np-complete optimization problem where we try to minimize this energy function. Two techniques that have been successful in approximately minimizing this energy function are dynamic programming (DP) and graph cuts.

If spatially neighboring points experience a smooth gradation in colors, then the structure of this smoothness is tree-like which lends itself to a DP solution ([1], [5]). Graph cuts is an iterative approach to approximating the minimization energy function by modeling the problem as a Max-flow/Min-cut problem. Both techniques have experienced success, however, the challenges of global methods are not because of drawbacks in optimization but rather energy functions that do not sufficiently model the problem.

In this paper, we attempt to learn valid matchings between two pixels by modeling the correspondence problem as a classification problem. In this attempts, our approach resembles that of a local method as we focus on the region surrounding a pixel. After learning the classification for a valid matching, we conduct a line sweep to determine the most confident matching for our corresponding pixel. The confidence in our matching is analogous to the weight function in local methods.

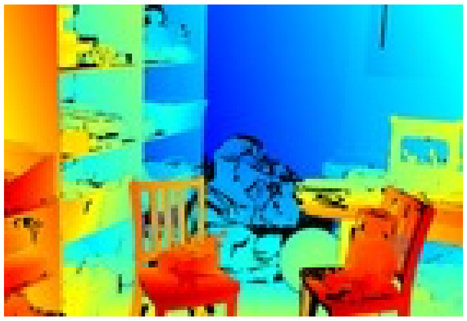
Data

Our data comes from the 2014 Middlebury stereo dataset[1] This dataset contains about two dozen pairs of high resolution images with pixel-by-pixel ground truth disparities. We chose subsets of these images as our training and testing data. The two images are taken by cameras a fixed horizontal distance apart with identical orientation and vertical position.

[1]http://vision.middlebury.edu/stereo/data/2014/

(a) Left Camera Image

(b) Right Camera Image


(c) Color map of Ground Truth Disparity

Feature Selection

A sample point in our training set represents to a pair of pixels, one in the left image and one in the right image. These pixels may or may not be a match. For this pair we construct a feature vector containing information about these pixels, and their environments. We include the values of the red, green, and blue channels for the pixels themselves and for all of the pixels in an $N \times N$ square around them. In order to detect important image features such as edges, we also convolve the image using Morlet wavelets, and include grayscale values for the same $N \times N$ square in the convolution of the images with wavelets with several sets of parameters.

Wavelets

We used the following definition of the discrete Morlet wavelet [2] in two dimensions:

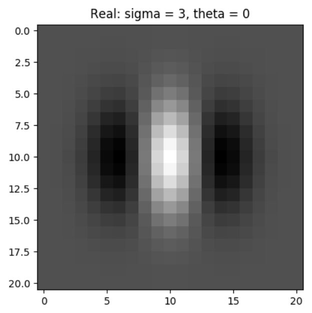
$$\psi_{\sigma,\theta}\left(\boldsymbol{u}\right) = \frac{C_1}{\sigma}\left(e^{i\frac{\pi}{2\pi}(\boldsymbol{u}\cdot e_\theta)} - C_2\right)e^{-\frac{\boldsymbol{u}^2}{2\sigma^2}}$$

where $\boldsymbol{u}$ is the displacement vector from the origin (x, y), and e_θ is the unit vector in direction θ. The parameter θ determines the orientation of the wavelet, and σ determines its width. C_1 and C_2 are constants. The constant C_2 is set

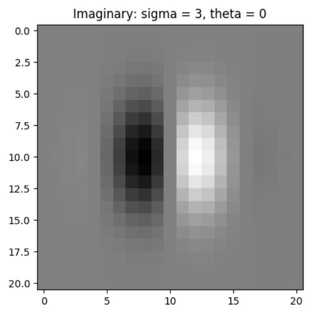
[2]Davi Geiger 2016 CSCI-GA.3033-012 "Haar Basis wavelets and Morlet Wavelets"

to make the mean of the wavelet equal to 0. The constant C_1 is set so that $\sum_x \sum_y \psi_{\sigma,\theta}(\boldsymbol{u})\, \psi^*_{\sigma,\theta}(\boldsymbol{u}) = 1$.

The above function creates a complex valued signal centered at the origin. Below is an example of the real and imaginary parts of one of these wavelets.

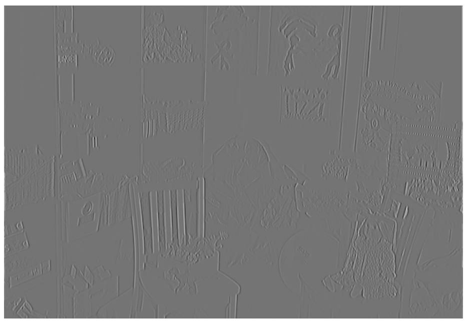


(a) Visual representation of the real part of a Morlet wavelet

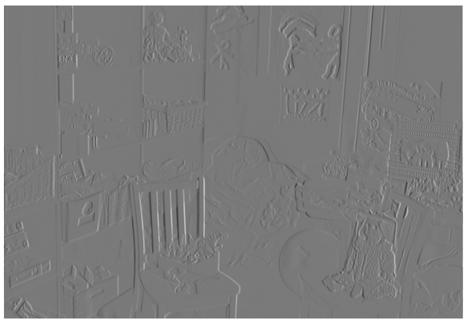


(b) Visual representation of the imaginary part of a Morlet wavelet

In the above images, white pixels represent higher (positive) values of the wavelet and black pixels represent lower (negative) values. The real part of the Morlet wavlet is perfect for detecting local extrema along its direction and the imaginary part is perfect for detecting edges perpendicular to the wavelet's direction. When convolved with an image, the wavelets produce outputs similar to the following.



(a) The playroom image convolved with the real part of the Morlet wavelet



(b) The playroom image convolved with the imaginary part of the Morlet wavelet

Building the Feature Vector

Given a pixel location in the left image and a pixel location in the right image, we create a separate vector for each. Each of these half vectors contains the RGB values of the pixel and the pixels in a 25 x 25 window centered at the pixel. We also add the grayscale values for the convolved images for the same

25 x 25 window, We use the real and imaginary parts for two convolutions with perpendicular directions. Finally, we compute the average values of the RGB and convolution images in a smaller window around the pixel (7 x 7).

Once the vectors for the individual pixels are computed, we take the element-wise absolute difference between the left vector and the right vector to create the final feature vector representing a pair of points. This pair is then labeled with a 1 if the pixels correspond to each other based on the ground truth in our data set and with a 0 otherwise.

Other Features

We attempted to included several other features, but found that they reduced our accuracy on our development set. We found that accuracy decrease when using a smaller window size than 12, but that increasing the window size further did not provide additional benefits. We also tried other features that were global to the window rather than specific to a pixel. These included the average, minimum, and maximum values over the whole window and over sub-windows. These did not lead to to an increase in accuracy. The only one of these features that was retained was the average value on the 7 x7 sub-window.

Classification

We begin by scaling our samples to have zero mean and unit variance. For our classifier, we used a linear Support Vector Machine. We also tested the usage of a Gaussian kernel and a Polynomial kernel of degree 3 in order to perform non-linear classification. For the sake of training time, kernel approximations were used through the Nyström method.

Per sample, we had generated around 4,400 features. Due to the large amount of available samples and the dimensionality of the features, we utilized stochastic gradient descent to train our classifier. This allowed us to both train at a much quicker rate while yielding an on-line approach to allow incremental learning and circumvention of memory limitations.

Results

After training, we experienced mixed results for classification. Kernel approximations provided no benefit and in fact worsened results. This can be attributed due to having a large feature space and memory restrictions resulting in limited amounts of pairwise similarities. In other words, we were unable to utilize kernels properly due to limiting memory constraints.

In regards to linear classification, training occurred on a subset of five images from the dataset, and tested on several portions of the sixth image. Overall, we found that the results varied greatly, with accuracy ranging from $70-87\%$ and an f1-score ranging from $40-72\%$. Unsurprisingly, results for more favorable in the less realistic case of training and testing on (disjoint) parts of the same image. In this case we were able to achieve accuracies ranging from $90-93\%$ and f1-scores ranging from $79-84\%$. The f-scores are a more appropriate way to evaluate these models because the testing samples were biased towards negative examples (non-matches.)

Finding Disparities

After training a binary classifier, we used that classifier to find the disparity for each pixel in a section of the left image. For each pixel in the left image, we performed a linear search of pixels at the same height in the right image. We interpreted the output of the decision function we had learned as a confidence, and selected the pixel in the right image with the highest confidence of a match. While our results show that our model is not particularly effective as a classifier, it was still possible that they would be useful for solving the correspondence problem. For example, even if for a given pixel in the left image there are no pixels in the right image classified as a match, the "best match" (the sample closest to the decision plane) may still be correct. Further, it is not necessary to correctly tag every single pixel for the results to be useful. One could apply the assumption that disparities are mostly smooth to reduce noise. With this in mind, we evaluated our results using a few quantitative measures and one qualitative measure.

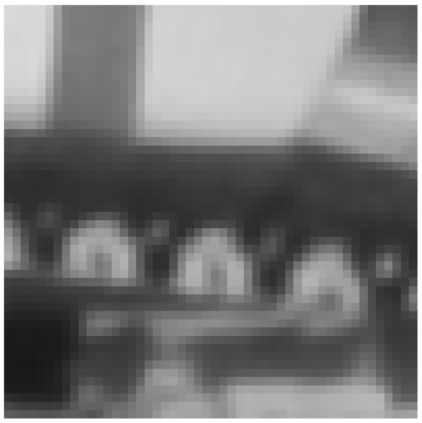
First, we calculated the fraction of pixels in the test sample that whose disparities we labeled correctly within a margin of one pixel. This is a completely correct match, since the ground truth disparities are floating point numbers and we can only predict exact integer disparities. Then we also measured the portion of disparities predicted within margins of two and five pixels. Finally, we calculated the mean square error of our predicted disparities. Each of these measures excludes pixels for which there is no ground truth disparity. That is, those which correspond to an occlusion.

In order to visually represent our results and evaluate them qualitatively, we created an image containing the pixel in the right image that we predicted for each location in the left image. If our model is working well, this should nearly reconstruct the left image. Here are a few examples.

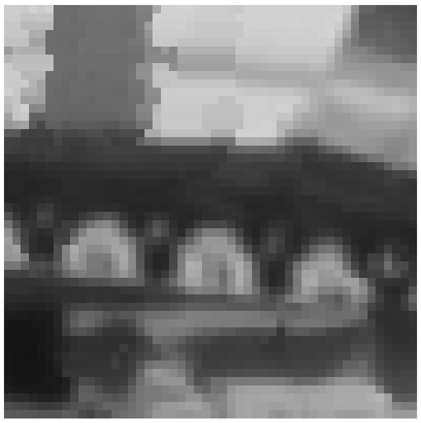
Results

Patterned Cloth:

- Predicted disparity within 1 pixel: 65.4%
- Within 2 pixels : 87.0%
- Within 5 pixels : 95.0%
- Mean square error: 2.24 pixels



(a) A small portion of the actual left image in gray

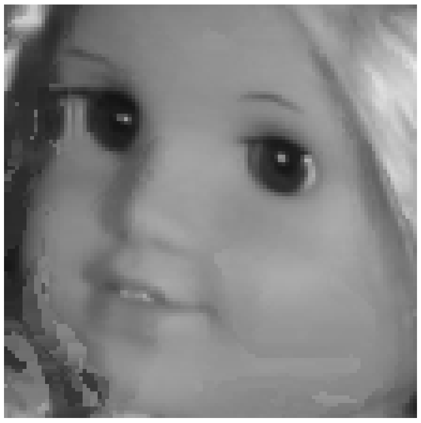


(b) The corresponding pixels in the right image, as predicted by our model

Doll's Face:

- Predicted disparity within 1 pixel: 44.8%
- Within 2 pixels : 68.1%
- Within 5 pixels : 88.6%
- Mean square error: 8.36 pixels

(a) A small portion of the actual left image in gray



(b) The corresponding pixels in the right image, as predicted by our model

Long Strip:

- Predicted disparity within 1 pixel: 36.5%
- Within 2 pixels : 49.9%
- Within 5 pixels : 63.0%

- Mean square error: 35.81 pixels

(a) A thin strip of the actual left image in gray

(b) The corresponding pixels in the right image, as predicted by our model

Discussion

Overall, the results that this approach yielded were not optimal in comparison to the previous state of the art in local methods. Techniques such as [3] and [4] achieved various average accuracies ranging within 5% for several images of the dataset.

We found that the same drawbacks for local methods renders learning of the weight function for local methods to be ill-posed when using classical algorithms. The frequent presence of *occlusions*, pixels appearing in one image but not the other, resulted in a large amount of noise added during training. In addition to occlusions, the presence of large areas of nearly uniform color posed problems for training and for the resulting model. Patches of uniform color present a challenge because there will be many incorrect matches that are very similar to correct matches. Patches that contain edges between regions of different depth do not have this problem, but do have the problem that the background and foreground shift relative to each other in the two stereo images, This means that samples labeled as matches will sometimes (but not always) have large regions that are completely different between the two images. This is the problem that adaptive window sizes attempt to solve, but this was not possible with the way we constructed feature vectors. Our inclusion of wavelets to detect edges attempts to compensate for this.

One difficulty about this problem is that while the question is posed in a way that sounds like a ranking (of best matches) or regression (on the disparity) problem, the data does not lend itself to these learning approaches, Early in the development of this project we attempted to model it as a ranking and as a regression problem. The problem is that there labeled data that allows us to train a model for these approaches. We have the true disparities for each pixel, but pairs of pixels that are not matches do not necessarily match better if their disparity is closer to the actual one. Thus, looking at two pixel frames doesn't tell us anything about how far apart they are, just whether or not they match. This means that while we can classify pairs of pixels as a match or not, we cannot say anything about how close (spatially) they are to a match.

We found that our approach to matching pixels worked best on flat surfaces with a lot of color variation. This gave the feature vectors a lot of information to work with in distinguishing between nearby pixels. In addition, the fact that disparities are smooth along flat surfaces and that there are no occlusions make the entire window used in the feature vector useful. We can see these conditions present in the patterned cloth example above. The flat surface and colorful pattern of the cloth allowed us to nearly perfectly reconstruct the patten. There is noise present in the upper portion of the image where the wall behind the cloth is occluded in the right image. In the image of the doll's face, we see that our method has weaker results with areas that are more uniform in color. Most of the face is mapped correctly. It is close to flat and has plenty of features such as shadows to help match. The lower right portion of the cheek, however, is more uniform in color, making it more difficult to match with precision. The errors on the left side of the image are caused by the occlusion of the pixels by the dolls face in the foreground of the right image. Our largest and most varied test sample is the long strip, which contains several regions of occlusions and uniform color. However, notice that even with the high error rate, much of the image is recognizable, especially in regions with a lot of edges and color variations. These results also suggest that the mean square error is not the best method of evaluation, because it heavily weights large errors, which in this context are really no more harmful than moderate errors.

References

[1] Stan Birchfield and Carlo Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3):269–293, Dec 1999.

[2] Michael Bleyer and Christian Breiteneder. *Stereo Matching—State-of-the-Art and Research Challenges*, pages 143–179. Springer London, London, 2013.

[3] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann. Local stereo matching using geodesic support weights. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 2093–2096, Nov 2009.

[4] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):504–511, Feb 2013.

[5] O. Veksler. Stereo correspondence by dynamic programming on a tree. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 384–390 vol. 2, June 2005.