Computing the Stereo Matching Cost with a Convolutional Neural Network

Jure Žbontar[1], Yann LeCun[2]
[1]University of Ljubljana. [2]New York University.

Introduction

Consider the following problem: given two images taken from cameras at different horizontal positions, the goal is to compute the disparity d for each pixel in the left image. Disparity refers to the difference in horizontal location of an object in the left and right image—an object at position (x,y) in the left image will appear at position $(x-d,y)$ in the right image. Knowing the disparity d of an object, we can compute its depth z (i.e. the distance from the object to the camera) by using the following relation: $z = \frac{fB}{d}$, where f is the focal length of the camera and B is the distance between the camera centers.

We propose training a convolutional neural network [2] on pairs of small image patches where the true disparity is known (e.g. obtained by LIDAR). The output of the network is used to initialize the matching cost between a pair of patches. Our stereo method achieves an error rate of 2.61 % on the KITTI stereo dataset [1] and was the top performing method on this dataset when the paper was first presented.

Creating the dataset

A training example comprises two patches, one from the left and one from the right image:

$$< \mathcal{P}^L_{9\times 9}(\mathbf{p}), \mathcal{P}^R_{9\times 9}(\mathbf{q}) >, \tag{1}$$

where $\mathcal{P}^L_{9\times 9}(\mathbf{p})$ denotes a 9×9 patch from the left image, centered at $\mathbf{p} = (x,y)$. For each location where the true disparity d is known, we extract one negative and one positive example. A negative example is obtained by setting the center of the right patch $\mathbf{q}$ to

$$\mathbf{q} = (x - d + o_{\text{neg}}, y), \tag{2}$$

where o_{neg} is an offset corrupting the match, chosen randomly from the set $\{-N_{\text{hi}},\ldots,-N_{\text{lo}},N_{\text{lo}},\ldots,N_{\text{hi}}\}$. Similarly, a positive example is derived by setting

$$\mathbf{q} = (x - d + o_{\text{pos}}, y), \tag{3}$$

where o_{pos} is chosen randomly from the set $\{-P_{\text{hi}},\ldots,P_{\text{hi}}\}$. The reason for including o_{pos}, instead of setting it to zero, has to do with the stereo method used later on, but o_{pos} was always less than o_{neg}. N_{lo}, N_{hi}, P_{hi}, and the size of the image patches n are hyperparameters of the method. Samples from the dataset are depicted in Figure 1.

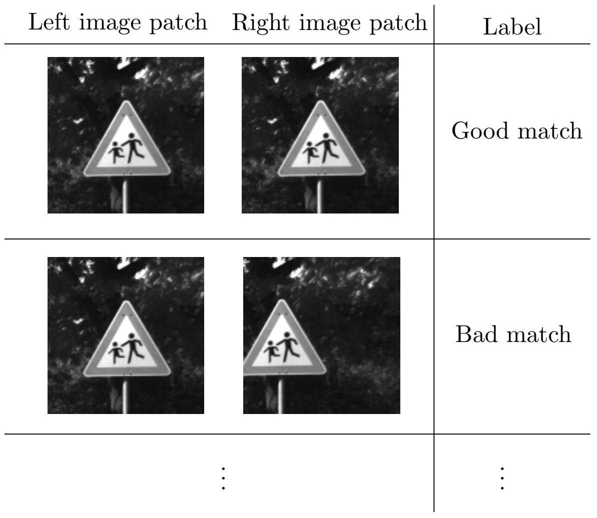

Figure 1: We constructed a dataset with 45 million training examples. Each example is a pair of image patches, one from the left and one from the right image. The positive examples are collected by aligning the two patches correctly (based on ground truth provided by LIDAR), while the negative examples are intentionally misaligned.

Network architecture

The network consists of eight layers, $L1$ through $L8$. The first layer is convolutional, while all other layers are fully-connected. The inputs to the network are two 9×9 gray image patches. The first convolutional layer consists of 32 kernels of size $5\times 5\times 1$. Layers $L2$ and $L3$ are fully-connected with 200 neurons each. After $L3$ the two 200 dimensional vectors are concatenated into a 400 dimensional vector and passed through four fully-connected layers, $L4$ through $L7$, with 300 neurons each. The final layer, $L8$, projects the output to two real numbers that are fed through a softmax function, producing a distribution over the two classes (good match and bad match). The weights in $L1$, $L2$, and $L3$ of the networks for the left and right image patch are tied. Rectified linear units follow each layer, except $L8$. We did not use pooling in our architecture. The network contains almost 600 thousand parameters. The architecture is appropriate for gray images, but can easily be extended to handle RGB images by learning $5\times 5\times 3$, instead of $5\times 5\times 1$ filters in $L1$. The best hyperparameters of the network (such as the number of layers, the number of neurons in each layer, and the size of input patches) will differ from one dataset to another. We chose this architecture because it performed well on the KITTI stereo dataset.

This is an extended abstract. The full paper is available at the Computer Vision Foundation webpage.

Matching cost

The matching cost $C_{\text{CNN}}(\mathbf{p},d)$ is computed directly from the output of the network:

$$C_{\text{CNN}}(\mathbf{p},d) = f_{\text{neg}}(< \mathcal{P}^L_{9\times 9}(\mathbf{p}), \mathcal{P}^R_{9\times 9}(\mathbf{pd}) >), \tag{4}$$

where $f_{\text{neg}}(< \mathcal{P}^L, \mathcal{P}^R >)$ is the output of the network for the negative class when run on input patches $\mathcal{P}^L$ and $\mathcal{P}^R$.

Post-processing

The post-processing steps we used were influenced by Mei et al. [3]. Matching costs were combined between neighboring pixels with similar image intensities using cross-based cost aggregation. Smoothness constraints were enforced by semiglobal matching and a left-right consistency check was used to detect and eliminate errors in occluded regions. We performed subpixel enhancement and applied a median filter and a bilateral filter to obtain the final disparity map.

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] Xing Mei, Xun Sun, Mingcai Zhou, Haitao Wang, Xiaopeng Zhang, et al. On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 467–474. IEEE, 2011.