Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches

Jure Žbontar[*]   JURE.ZBONTAR@FRI.UNI−LJ.SI
Faculty of Computer and Information Science
University of Ljubljana
Večna pot 113, SI-1001 Ljubljana, Slovenia

Yann LeCun[†]   YANN@CS.NYU.EDU
Courant Institute of Mathematical Sciences
New York University
715 Broadway, New York, NY 10003, USA

Abstract

We present a method for extracting depth information from a rectified image pair. Our approach focuses on the first stage of many stereo algorithms: the matching cost computation. We approach the problem by learning a similarity measure on small image patches using a convolutional neural network. Training is carried out in a supervised manner by constructing a binary classification data set with examples of similar and dissimilar pairs of patches. We examine two network architectures for this task: one tuned for speed, the other for accuracy. The output of the convolutional neural network is used to initialize the stereo matching cost. A series of post-processing steps follow: cross-based cost aggregation, semiglobal matching, a left-right consistency check, subpixel enhancement, a median filter, and a bilateral filter. We evaluate our method on the KITTI 2012, KITTI 2015, and Middlebury stereo data sets and show that it outperforms other approaches on all three data sets.

Keywords: stereo, matching cost, similarity learning, supervised learning, convolutional neural networks

1. Introduction

Consider the following problem: given two images taken by cameras at different horizontal positions, we wish 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 appears at position $(x - d, y)$ in the right image. If we know the disparity of an object we can compute its depth z using the following relation:

$$z = \frac{fB}{d},$$

∗. Jure Žbontar is also with the *Courant Institute of Mathematical Sciences, New York University, 715 Broadway, New York, NY 10003, USA.*

†. Yann LeCun is also with *Facebook AI Research, 770 Broadway, New York, NY 10003, USA.*

Left input image

Output disparity map

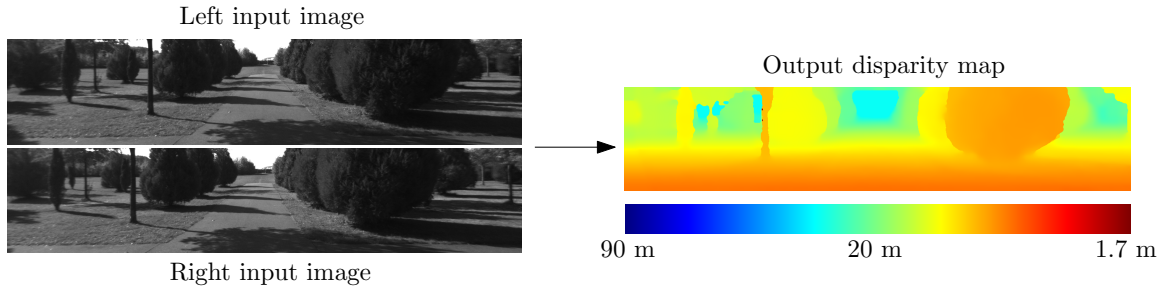Right input image

90 m   20 m   1.7 m

Figure 1: The input is a pair of images from the left and right camera. The two input images differ mostly in horizontal locations of objects (other differences are caused by reflections, occlusions, and perspective distortions). Note that objects closer to the camera have larger disparities than objects farther away. The output is a dense disparity map shown on the right, with warmer colors representing larger values of disparity (and smaller values of depth).

where f is the focal length of the camera and B is the distance between the camera centers. Figure 1 depicts the input to and the output from our method.

The described problem of stereo matching is important in many fields such as autonomous driving, robotics, intermediate view generation, and 3D scene reconstruction. According to the taxonomy of Scharstein and Szeliski (2002), a typical stereo algorithm consists of four steps: matching cost computation, cost aggregation, optimization, and disparity refinement. Following Hirschmüller and Scharstein (2009) we refer to the first two steps as computing the matching cost and the last two steps as the stereo method. The focus of this work is on computing a good matching cost.

We propose training a convolutional neural network (LeCun et al., 1998) on pairs of small image patches where the true disparity is known (for example, obtained by LIDAR or structured light). The output of the network is used to initialize the matching cost. We proceed with a number of post-processing steps that are not novel, but are necessary to achieve good results. Matching costs are combined between neighboring pixels with similar image intensities using cross-based cost aggregation. Smoothness constraints are enforced by semiglobal matching and a left-right consistency check is used to detect and eliminate errors in occluded regions. We perform subpixel enhancement and apply a median filter and a bilateral filter to obtain the final disparity map.

The contributions of this paper are

- a description of two architectures based on convolutional neural networks for computing the stereo matching cost;

- a method, accompanied by its source code, with the lowest error rate on the KITTI 2012, KITTI 2015, and Middlebury stereo data sets; and

- experiments analyzing the importance of data set size, the error rate compared with other methods, and the trade-off between accuracy and runtime for different settings of the hyperparameters.

This paper extends our previous work (Žbontar and LeCun, 2015) by including a description of a new architecture, results on two new data sets, lower error rates, and more thorough experiments.

2. Related Work

Before the introduction of large stereo data sets like KITTI and Middlebury, relatively few stereo algorithms used ground truth information to learn parameters of their models; in this section, we review the ones that did. For a general overview of stereo algorithms see Scharstein and Szeliski (2002).

Kong and Tao (2004) used the sum of squared distances to compute an initial matching cost. They then trained a model to predict the probability distribution over three classes: the initial disparity is correct, the initial disparity is incorrect due to fattening of a foreground object, and the initial disparity is incorrect due to other reasons. The predicted probabilities were used to adjust the initial matching cost. Kong and Tao (2006) later extend their work by combining predictions obtained by computing normalized cross-correlation over different window sizes and centers. Peris et al. (2012) initialized the matching cost with AD-Census (Mei et al., 2011), and used multiclass linear discriminant analysis to learn a mapping from the computed matching cost to the final disparity.

Ground-truth data was also used to learn parameters of probabilistic graphical models. Zhang and Seitz (2007) used an alternative optimization algorithm to estimate optimal values of Markov random field hyperparameters. Scharstein and Pal (2007) constructed a new data set of 30 stereo pairs and used it to learn parameters of a conditional random field. Li and Huttenlocher (2008) presented a conditional random field model with a non-parametric cost function and used a structured support vector machine to learn the model parameters.

Recent work (Haeusler et al., 2013; Spyropoulos et al., 2014) focused on estimating the confidence of the computed matching cost. Haeusler et al. (2013) used a random forest classifier to combine several confidence measures. Similarly, Spyropoulos et al. (2014) trained a random forest classifier to predict the confidence of the matching cost and used the predictions as soft constraints in a Markov random field to decrease the error of the stereo method.

A related problem to computing the matching cost is learning local image descriptors (Brown et al., 2011; Trzcinski et al., 2012; Simonyan et al., 2014; Revaud et al., 2015; Paulin et al., 2015; Han et al., 2015; Zagoruyko and Komodakis, 2015). The two problems share a common subtask: to measure the similarity between image patches. Brown et al. (2011) introduced a general framework for learning image descriptors and used Powell's method to select good hyperparameters. Several methods have been suggested for solving the problem of learning local image descriptors, such as boosting (Trzcinski et al., 2012), convex optimization (Simonyan et al., 2014), hierarchical moving-quadrant similarity (Revaud et al., 2015), convolutional kernel networks (Paulin et al., 2015), and convolutional neural networks (Zagoruyko and Komodakis, 2015; Han et al., 2015). Works of Zagoruyko and Komodakis (2015) and Han et al. (2015), in particular, are very similar to our own, differing mostly in the architecture of the network; concretely, the inclusion of pooling and subsampling to account for larger patch sizes and larger variation in viewpoint.

3. Matching Cost

A typical stereo algorithm begins by computing a matching cost at each position $\mathbf{p}$ for all disparities d under consideration. A simple method for computing the matching cost is the sum of absolute differences:

$$C_{\text{SAD}}(\mathbf{p}, d) = \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} |I^L(\mathbf{q}) - I^R(\mathbf{q} - \mathbf{d})|, \tag{1}$$

where $I^L(\mathbf{p})$ and $I^R(\mathbf{p})$ are image intensities at position $\mathbf{p}$ in the left and right image and $\mathcal{N}_{\mathbf{p}}$ is the set of locations within a fixed rectangular window centered at $\mathbf{p}$.

We use bold lowercase letters $\mathbf{p}$ and $\mathbf{q}$ to denote image locations. A bold lowercase $\mathbf{d}$ denotes the disparity d cast to a vector, that is, $\mathbf{d} = (d, 0)$. We use `typewriter` font for the names of hyperparameters. For example, we would use `patch_size` to denote the size of the neighbourhood area $\mathcal{N}_{\mathbf{p}}$.

Equation (1) can be interpreted as measuring the cost associated with matching a patch from the left image, centered at position $\mathbf{p}$, with a patch from the right image, centered at position $\mathbf{p} - \mathbf{d}$. We want the cost to be low when the two patches are centered around the image of the same 3D point, and high when they are not.

Since examples of good and bad matches can be constructed from publicly available data sets (for example, the KITTI and Middlebury stereo data sets), we can attempt to solve the matching problem by a supervised learning approach. Inspired by the successful application of convolutional neural networks to vision problems, we used them to assess how well two small image patches match.

3.1 Constructing the Data Set

We use ground truth disparity maps from either the KITTI or Middlebury stereo data sets to construct a binary classification data set. At each image position where the true disparity is known we extract one negative and one positive training example. This ensures that the data set contains an equal number of positive and negative examples. A positive example is a pair of patches, one from the left and one from the right image, whose center pixels are the images of the same 3D point, while a negative example is a pair of patches where this is not the case. The following section describes the data set construction step in detail.

Let $< \mathcal{P}_{n \times n}^L(\mathbf{p}), \mathcal{P}_{n \times n}^R(\mathbf{q}) >$ denote a pair of patches, where $\mathcal{P}_{n \times n}^L(\mathbf{p})$ is an $n \times n$ patch from the left image centered at position $\mathbf{p} = (x, y)$, $\mathcal{P}_{n \times n}^R(\mathbf{q})$ is an $n \times n$ patch from the right image centered at position $\mathbf{q}$, and d denotes the correct disparity at position $\mathbf{p}$. A negative example is obtained by setting the center of the right patch to

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

where o_{neg} is chosen from either the interval $[\texttt{dataset\_neg\_low}, \texttt{dataset\_neg\_high}]$ or, its origin reflected counterpart, $[-\texttt{dataset\_neg\_high}, -\texttt{dataset\_neg\_low}]$. The random offset o_{neg} ensures that the resulting image patches are not centered around the same 3D point.

A positive example is derived by setting

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

Similarity score

```
              ┌─────────────────┐
              │   Dot product   │
              └─────────────────┘
   ┌──────────────────┐   ┌──────────────────┐
   │    Normalize     │   │    Normalize     │
   ├──────────────────┤   ├──────────────────┤
   │   Convolution    │   │   Convolution    │
   └──────────────────┘   └──────────────────┘
            ⋮                       ⋮
   ┌──────────────────┐   ┌──────────────────┐
   │ Convolution, ReLU│   │ Convolution, ReLU│
   ├──────────────────┤   ├──────────────────┤
   │ Convolution, ReLU│   │ Convolution, ReLU│
   └──────────────────┘   └──────────────────┘
```

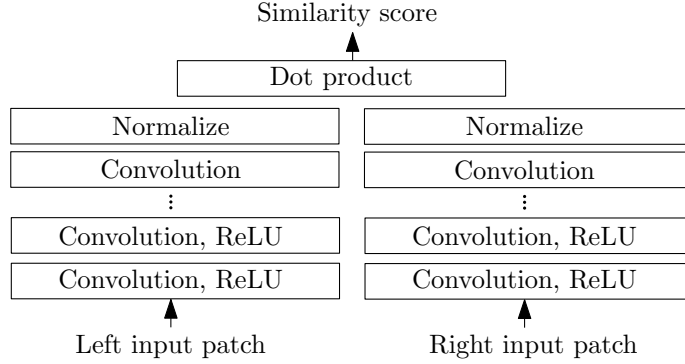Left input patch   Right input patch

Figure 2: The fast architecture is a siamese network. The two sub-networks consist of a number of convolutional layers followed by rectified linear units (abbreviated "ReLU"). The similarity score is obtained by extracting a vector from each of the two input patches and computing the cosine similarity between them. In this diagram, as well as in our implementation, the cosine similarity computation is split in two steps: normalization and dot product. This reduces the running time because the normalization needs to be performed only once per position (see Section 3.3).

where o_pos is chosen randomly from the interval $[-\texttt{dataset\_pos}, \texttt{dataset\_pos}]$. The reason for including o_pos, instead of setting it to zero, has to do with the stereo method used later on. In particular, we found that cross-based cost aggregation performs better when the network assigns low matching costs to good matches as well as near matches. In our experiments, the hyperparameter $\texttt{dataset\_pos}$ was never larger than one pixel.

3.2 Network Architectures

We describe two network architectures for learning a similarity measure on image patches. The first architecture is faster than the second, but produces disparity maps that are slightly less accurate. In both cases, the input to the network is a pair of small image patches and the output is a measure of similarity between them. Both architectures contain a trainable feature extractor that represents each image patch with a feature vector. The similarity between patches is measured on the feature vectors instead of the raw image intensity values. The fast architecture uses a fixed similarity measure to compare the two feature vectors, while the accurate architecture attempts to learn a good similarity measure on feature vectors.

3.2.1 FAST ARCHITECTURE

The first architecture is a siamese network, that is, two shared-weight sub-networks joined at the head (Bromley et al., 1993). The sub-networks are composed of a number of convolutional layers with rectified linear units following all but the last layer. Both sub-networks output a vector capturing the properties of the input patch. The resulting two vectors are

Similarity score

```
┌─────────────────────────────┐
│  Fully-connected, Sigmoid   │
├─────────────────────────────┤
│   Fully-connected, ReLU     │
└─────────────────────────────┘
            ⋮
┌─────────────────────────────┐
│   Fully-connected, ReLU     │
├─────────────────────────────┤
│   Fully-connected, ReLU     │
├─────────────────────────────┤
│         Concatenate         │
└─────────────────────────────┘
```

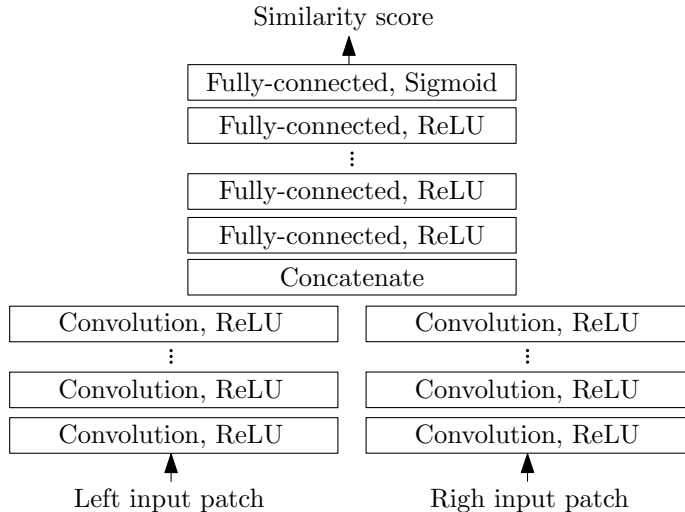| Convolution, ReLU | Convolution, ReLU |
⋮	⋮
Convolution, ReLU	Convolution, ReLU
Convolution, ReLU	Convolution, ReLU

Left input patch   Righ input patch

Figure 3: The accurate architecture begins with two convolutional feature extractors. The extracted feature vectors are concatenated and compared by a number of fully-connected layers. The inputs are two image patches and the output is a single real number between 0 and 1, which we interpret as a measure of similarity between the input images.

compared using the cosine similarity measure to produce the final output of the network. Figure 2 provides an overview of the architecture.

The network is trained by minimizing a hinge loss. The loss is computed by considering pairs of examples centered around the same image position where one example belongs to the positive and one to the negative class. Let s_+ be the output of the network for the positive example, s_- be the output of the network for the negative example, and let m, the margin, be a positive real number. The hinge loss for that pair of examples is defined as $\max(0, m + s_- - s_+)$. The loss is zero when the similarity of the positive example is greater than the similarity of the negative example by at least the margin m. We set the margin to 0.2 in our experiments.

The hyperparameters of this architecture are the number of convolutional layers in each sub-network (`num_conv_layers`), the size of the convolution kernels (`conv_kernel_size`), the number of feature maps in each layer (`num_conv_feature_maps`), and the size of the input patch (`input_patch_size`).

3.2.2 ACCURATE ARCHITECTURE

The second architecture is derived from the first by replacing the cosine similarity measure with a number of fully-connected layers (see Figure 3). This architectural change increased the running time, but decreased the error rate. The two sub-networks comprise a number of convolutional layers, with a rectified linear unit following each layer. The resulting two vectors are concatenated and forward-propagated through a number of fully-connected

layers followed by rectified linear units. The last fully-connected layer produces a single number which, after being transformed with the sigmoid nonlinearity, is interpreted as the similarity score between the input patches.

We use the binary cross-entropy loss for training. Let s denote the output of the network for one training example and t denote the class of that training example; $t = 1$ if the example belongs to the positive class and $t = 0$ if the example belongs to the negative class. The binary cross-entropy loss for that example is defined as $t \log(s) + (1 - t) \log(1 - s)$.

The decision to use two different loss functions, one for each architecture, was based on empirical evidence. While we would have preferred to use the same loss function for both architectures, experiments showed that the binary cross-entropy loss performed better than the hinge loss on the accurate architecture. On the other hand, since the last step of the fast architecture is the cosine similarity computation, a cross-entropy loss was not directly applicable.

The hyperparameters of the accurate architecture are the number of convolutional layers in each sub-network (`num_conv_layers`), the number of feature maps in each layer (`num_conv_feature_maps`), the size of the convolution kernels (`conv_kernel_size`), the size of the input patch (`input_patch_size`), the number of units in each fully-connected layer (`num_fc_units`), and the number of fully-connected layers (`num_fc_layers`).

3.3 Computing the Matching Cost

The output of the network is used to initialize the matching cost:

$$C_{\mathrm{CNN}}(\mathbf{p}, d) = -s(< \mathcal{P}^L(\mathbf{p}), \mathcal{P}^R(\mathbf{p} - \mathbf{d}) >),$$

where $s(< \mathcal{P}^L(\mathbf{p}), \mathcal{P}^R(\mathbf{p} - \mathbf{d}) >)$ is the output of the network when run on input patches $\mathcal{P}^L(\mathbf{p})$ and $\mathcal{P}^R(\mathbf{p} - \mathbf{d})$. The minus sign converts the similarity score to a matching cost.

To compute the entire matching cost tensor $C_{\mathrm{CNN}}(\mathbf{p}, d)$ we would, naively, have to perform the forward pass for each image location and each disparity under consideration. The following three implementation details kept the running time manageable:

- The outputs of the two sub-networks need to be computed only once per location, and do not need to be recomputed for every disparity under consideration.

- The output of the two sub-networks can be computed for all pixels in a single forward pass by propagating full-resolution images, instead of small image patches. Performing a single forward pass on the entire $w \times h$ image is faster than performing $w \cdot h$ forward passes on small patches because many intermediate results can be reused.

- The output of the fully-connected layers in the accurate architecture can also be computed in a single forward pass. This is done by replacing each fully-connected layer with a convolutional layer with 1×1 kernels. We still need to perform the forward pass for each disparity under consideration; the maximum disparity d is 228 for the KITTI data set and 400 for the Middlebury data set. As a result, the fully-connected part of the network needs to be run d times, and is a bottleneck of the accurate architecture.

To compute the matching cost of a pair of images, we run the sub-networks once on each image and run the fully-connected layers d times, where d is the maximum disparity under consideration. <mark>This insight was important in designing the architecture of the network.</mark> We could have chosen an architecture where the two images are concatenated before being presented to the network, but that would imply a large cost at runtime because the whole network would need to be run d times. <mark>This insight also led to the development of the fast architecture, where the only layer that is run d times is the dot product of the feature vectors.</mark>

4. Stereo Method

The raw outputs of the convolutional neural network are not enough to produce accurate disparity maps, <mark>with errors particularly apparent in low-texture regions and occluded areas.</mark> The quality of the disparity maps can be improved by applying a series of post-processing steps referred to as the stereo method. The stereo method we used was influenced by Mei et al. (2011) and comprises cross-based cost aggregation, semiglobal matching, a left-right consistency check, subpixel enhancement, a median, and a bilateral filter.

4.1 Cross-based Cost Aggregation

Information from neighboring pixels can be combined by averaging the matching cost over a fixed window. <mark>This approach fails near depth discontinuities, where the assumption of constant depth within a window is violated.</mark> We might prefer a method that adaptively selects the neighborhood for each pixel, so that support is collected only from pixels of the same physical object. In cross-based cost aggregation (Zhang et al., 2009) we build a local neighborhood around each location comprising pixels with similar image intensity values with the hope that these pixels belong to the same object.

The method begins by constructing an upright cross at each position; this cross is used to define the local support region. The left arm $\mathbf{p}_l$ at position $\mathbf{p}$ extends left as long as the following two conditions hold:

- $|I(\mathbf{p}) - I(\mathbf{p}_l)| < \texttt{cbca\_intensity}$; the image intensities at positions $\mathbf{p}$ and $\mathbf{p}_l$ should be similar, their difference should be less than $\texttt{cbca\_intensity}$.

- $\|\mathbf{p} - \mathbf{p}_l\| < \texttt{cbca\_distance}$; the horizontal distance (or vertical distance in case of top and bottom arms) between positions $\mathbf{p}$ and $\mathbf{p}_l$ is less than $\texttt{cbca\_distance}$ pixels.

The right, bottom, and top arms are constructed analogously. Once the four arms are known, we can compute the support region $U(\mathbf{p})$ as the union of horizontal arms of all positions $\mathbf{q}$ laying on $\mathbf{p}$'s vertical arm (see Figure 4).

Zhang et al. (2009) suggest that aggregation should consider the support regions of both images in a stereo pair. Let U^L and U^R denote the support regions in the left and right image. We define the combined support region U_d as

$$U_d(\mathbf{p}) = \{\mathbf{q}|\mathbf{q} \in U^L(\mathbf{p}), \mathbf{q} - \mathbf{d} \in U^R(\mathbf{p} - \mathbf{d})\}.$$
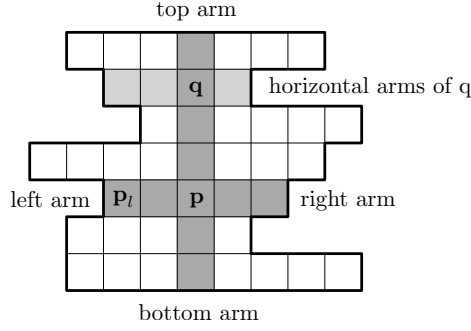
Figure 4: The support region for position $\mathbf{p}$ is the union of horizontal arms of all positions $\mathbf{q}$ on $\mathbf{p}$'s vertical arm.

The matching cost is averaged over the combined support region:

$$
C_{\text{CBCA}}^{0}(\mathbf{p}, d) = C_{\text{CNN}}(\mathbf{p}, d),
$$

$$
C_{\text{CBCA}}^{i}(\mathbf{p}, d) = \frac{1}{|U_d(\mathbf{p})|} \sum_{\mathbf{q} \in U_d(\mathbf{p})} C_{\text{CBCA}}^{i-1}(\mathbf{q}, d),
$$

where i is the iteration number. We repeat the averaging a number of times. Since the support regions are overlapping, the results can change at each iteration. We skip cross-based cost aggregation in the fast architecture because it is not crucial for achieving a low error rate and because it is relatively expensive to compute.

4.2 Semiglobal Matching

We refine the matching cost by enforcing smoothness constraints on the disparity image. Following Hirschmüller (2008), we define an energy function $E(D)$ that depends on the disparity image D:

$$
E(D) = \sum_{\mathbf{p}} \Bigg(C_{\text{CBCA}}^{4}(\mathbf{p}, D(\mathbf{p})) + \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} P_1 \cdot 1\{|D(\mathbf{p}) - D(\mathbf{q})| = 1\}
$$

$$
+ \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} P_2 \cdot 1\{|D(\mathbf{p}) - D(\mathbf{q})| > 1\} \Bigg),
$$

where $1\{\cdot\}$ denotes the indicator function. The first term penalizes disparities with high matching costs. The second term adds a penalty P_1 when the disparity of neighboring pixels differ by one. The third term adds a larger penalty P_2 when the neighboring disparities differ by more than one. Rather than minimizing $E(D)$ in all directions simultaneously, we could perform the minimization in a single direction with dynamic programming. This solution would introduce unwanted streaking effects, since there would be no incentive to make the disparity image smooth in the directions we are not optimizing over. In semiglobal matching we

9

minimize the energy in a single direction, repeat for several directions, and average to obtain the final result. Although Hirschmüller (2008) suggested choosing sixteen direction, we only optimized along the two horizontal and the two vertical directions; adding the diagonal directions did not improve the accuracy of our system. To minimize $E(D)$ in direction $\mathbf{r}$, we define a matching cost $C_{\mathbf{r}}(\mathbf{p}, d)$ with the following recurrence relation:

$$C_{\mathbf{r}}(\mathbf{p}, d) = C_{\mathrm{CBCA}}^4(\mathbf{p}, d) - \min_k C_r(\mathbf{p} - \mathbf{r}, k) + \min\Big\{ C_r(\mathbf{p} - \mathbf{r}, d), C_r(\mathbf{p} - \mathbf{r}, d - 1) + P_1,$$

$$C_r(\mathbf{p} - \mathbf{r}, d + 1) + P_1, \min_k C_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, k) + P_2 \Big\}.$$

The second term is subtracted to prevent values of $C_{\mathbf{r}}(\mathbf{p}, d)$ from growing too large and does not affect the optimal disparity map.

The penalty parameters P_1 and P_2 are set according to the image gradient so that jumps in disparity coincide with edges in the image. Let $D_1 = |I^L(\mathbf{p}) - I^L(\mathbf{p} - \mathbf{r})|$ and $D_2 = |I^R(\mathbf{p} - \mathbf{d}) - I^R(\mathbf{p} - \mathbf{d} - \mathbf{r})|$ be the difference in image intensity between two neighboring positions in the direction we are optimizing over. We set P_1 and P_2 according to the following rules:

$$\begin{aligned}
&P_1 = \texttt{sgm\_P1}, &&P_2 = \texttt{sgm\_P2} &&\text{if } D_1 < \texttt{sgm\_D}, D_2 < \texttt{sgm\_D}; \\
&P_1 = \texttt{sgm\_P1/sgm\_Q2}, &&P_2 = \texttt{sgm\_P2/sgm\_Q2} &&\text{if } D_1 \geq \texttt{sgm\_D}, D_2 \geq \texttt{sgm\_D}; \\
&P_1 = \texttt{sgm\_P1/sgm\_Q1}, &&P_2 = \texttt{sgm\_P2/sgm\_Q1} &&\text{otherwise.}
\end{aligned}$$

The hyperparameters $\texttt{sgm\_P1}$ and $\texttt{sgm\_P2}$ set a base penalty for discontinuities in the disparity map. The base penalty is reduced by a factor of $\texttt{sgm\_Q1}$ if one of D_1 or D_2 indicate a strong image gradient or by a larger factor of $\texttt{sgm\_Q2}$ if both D_1 and D_2 indicate a strong image gradient. The value of P_1 is further reduced by a factor of $\texttt{sgm\_V}$ when considering the two vertical directions; in the ground truth, small changes in disparity are much more frequent in the vertical directions than in the horizontal directions and should be penalised less.

The final cost $C_{\mathrm{SGM}}(\mathbf{p}, d)$ is computed by taking the average across all four directions:

$$C_{\mathrm{SGM}}(\mathbf{p}, d) = \frac{1}{4} \sum_{\mathbf{r}} C_{\mathbf{r}}(\mathbf{p}, d).$$

After semiglobal matching we repeat cross-based cost aggregation, as described in the previous section. Hyperparameters $\texttt{cbca\_num\_iterations\_1}$ and $\texttt{cbca\_num\_iterations\_2}$ determine the number of cross-based cost aggregation iterations before and after semiglobal matching.

4.3 Computing the Disparity Image

The disparity image $D(\mathbf{p})$ is computed by the winner-takes-all strategy, that is, by finding the disparity d that minimizes $C(\mathbf{p}, d)$,

$$D(\mathbf{p}) = \operatorname*{argmin}_d C(\mathbf{p}, d).$$

4.3.1 INTERPOLATION

The interpolation steps attempt to resolve conflicts between the disparity map predicted for the left image and the disparity map predicted for the right image. Let D^L denote the disparity map obtained by treating the left image as the reference image—this was the case so far, that is, $D^L(\mathbf{p}) = D(\mathbf{p})$—and let D^R denote the disparity map obtained by treating the right image as the reference image. D^L and D^R sometimes disagree on what the correct disparity at a particular position should be. We detect these conflicts by performing a left-right consistency check. We label each position $\mathbf{p}$ by applying the following rules in turn:

$$
\begin{array}{ll}
correct & \text{if } |d - D^R(\mathbf{p} - \mathbf{d})| \leq 1 \text{ for } d = D^L(\mathbf{p}), \\
mismatch & \text{if } |d - D^R(\mathbf{p} - \mathbf{d})| \leq 1 \text{ for any other } d, \\
occlusion & \text{otherwise.}
\end{array}
$$

For positions marked as *occlusion*, we want the new disparity value to come from the background. We interpolate by moving left until we find a position labeled *correct* and use its value. For positions marked as *mismatch*, we find the nearest *correct* pixels in 16 different directions and use the median of their disparities for interpolation. We refer to the interpolated disparity map as D_{INT}.

4.3.2 SUBPIXEL ENHANCEMENT

Subpixel enhancement provides an easy way to increase the resolution of a stereo algorithm. We fit a quadratic curve through the neighboring costs to obtain a new disparity image:

$$
D_{\mathrm{SE}}(\mathbf{p}) = d - \frac{C_+ - C_-}{2(C_+ - 2C + C_-)},
$$

where $d = D_{\mathrm{INT}}(\mathbf{p})$, $C_- = C_{\mathrm{SGM}}(\mathbf{p}, d-1)$, $C = C_{\mathrm{SGM}}(\mathbf{p}, d)$, and $C_+ = C_{\mathrm{SGM}}(\mathbf{p}, d+1)$.

4.3.3 REFINEMENT

The final steps of the stereo method consist of a 5×5 median filter and the following bilateral filter:

$$
D_{\mathrm{BF}}(\mathbf{p}) = \frac{1}{W(\mathbf{p})} \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} D_{\mathrm{SE}}(\mathbf{q}) \cdot g(\|\mathbf{p} - \mathbf{q}\|) \cdot 1\{|I^L(\mathbf{p}) - I^L(\mathbf{q})| < \texttt{blur\_threshold}\},
$$

where $g(x)$ is the probability density function of a zero mean normal distribution with standard deviation $\texttt{blur\_sigma}$ and $W(\mathbf{p})$ is the normalizing constant,

$$
W(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} g(\|\mathbf{p} - \mathbf{q}\|) \cdot 1\{|I^L(\mathbf{p}) - I^L(\mathbf{q})| < \texttt{blur\_threshold}\}.
$$

The role of the bilateral filter is to smooth the disparity map without blurring the edges. D_{BF} is the final output of our stereo method.

5. Experiments

We used three stereo data sets in our experiments: KITTI 2012, KITTI 2015, and Middlebury. The test set error rates reported in Tables 1, 2, and 4 were obtained by submitting

Rank	Method		Setting	Error	Runtime
1	**MC-CNN-acrt**	**Accurate architecture**		2.43	67
2	Displets	Güney and Geiger (2015)		2.47	265
3	MC-CNN	Žbontar and LeCun (2015)		2.61	100
4	PRSM	Vogel et al. (2015)	F, MV	2.78	300
	MC-CNN-fst	**Fast architecture**		2.82	0.8
5	SPS-StFl	Yamaguchi et al. (2014)	F, MS	2.83	35
6	VC-SF	Vogel et al. (2014)	F, MV	3.05	300
7	Deep Embed	Chen et al. (2015)		3.10	3
8	JSOSM	Unpublished work		3.15	105
9	OSF	Menze and Geiger (2015)	F	3.28	3000
10	CoR	Chakrabarti et al. (2015)		3.30	6

Table 1: The highest ranking methods on the KITTI 2012 data set as of October 2015. The "Setting" column provides insight into how the disparity map is computed: "F" indicates the use of optical flow, "MV" indicates more than two temporally adjacent images, and "MS" indicates the use of epipolar geometry for computing the optical flow. The "Error" column reports the percentage of misclassified pixels and the "Runtime" column measures the time, in seconds, required to process one pair of images.

the generated disparity maps to the online evaluation servers. All other error rates were computed by splitting the data set in two, using one part for training and the other for validation.

5.1 KITTI Stereo Data Set

The KITTI stereo data set (Geiger et al., 2013; Menze and Geiger, 2015) is a collection of rectified image pairs taken from two video cameras mounted on the roof of a car, roughly 54 centimeters apart. The images were recorded while driving in and around the city of Karlsruhe, in sunny and cloudy weather, at daytime. The images were taken at a resolution of 1240×376. A rotating laser scanner mounted behind the left camera recorded ground truth depth, labeling around 30 % of the image pixels.

The ground truth disparities for the test set are withheld and an online leaderboard is provided where researchers can evaluate their method on the test set. Submissions are allowed once every three days. Error is measured as the percentage of pixels where the true disparity and the predicted disparity differ by more than three pixels. Translated into distance, this means that, for example, the error tolerance is 3 centimeters for objects 2 meters from the camera and 80 centimeters for objects 10 meters from the camera.

Rank	Method		Setting	Error	Runtime
1	**MC-CNN-acrt**	**Accurate architecture**		3.89	67
	MC-CNN-fst	**Fast architecture**		4.62	0.8
2	SPS-St	Yamaguchi et al. (2014)		5.31	2
3	OSF	Menze and Geiger (2015)	F	5.79	3000
4	PR-Sceneflow	Vogel et al. (2013)	F	6.24	150
5	SGM+C+NL	Hirschmüller (2008); Sun et al. (2014)	F	6.84	270
6	SGM+LDOF	Hirschmüller (2008); Brox and Malik (2011)	F	6.84	86
7	SGM+SF	Hirschmüller (2008); Hornacek et al. (2014)	F	6.84	2700
8	ELAS	Geiger et al. (2011)		9.72	0.3
9	OCV-SGBM	Hirschmüller (2008)		10.86	1.1
10	SDM	Kostková and Sára (2003)		11.96	60

Table 2: The leading submission on the KITTI 2015 leaderboard as of October 2015. The "Setting", "Error", and "Runtime" columns have the same meaning as in Table 1.

Two KITTI stereo data sets exist: KITTI 2012[1] and, the newer, KITTI 2015[2]. For the task of computing stereo they are nearly identical, with the newer data set improving some aspects of the optical flow task. The 2012 data set contains 194 training and 195 testing images, while the 2015 data set contains 200 training and 200 testing images. There is a subtle but important difference introduced in the newer data set: vehicles in motion are densely labeled and car glass is included in the evaluation. This emphasizes the method's performance on reflective surfaces.

The best performing methods on the KITTI 2012 data set are listed in Table 1. Our accurate architecture ranks first with an error rate of 2.43 %. Third place on the leaderboard is held by our previous work (Žbontar and LeCun, 2015) with an error rate of 2.61 %. The two changes that reduced the error from 2.61 % to 2.43 % were augmenting the data set (see Section 5.4) and doubling the number of convolution layers while reducing the kernel size from 5×5 to 3×3. The method in second place (Güney and Geiger, 2015) uses the matching cost computed by our previous work (Žbontar and LeCun, 2015). The test error rate of the fast architecture is 2.82 %, which would be enough for fifth place had the method been allowed to appear in the public leaderboard. The running time for processing a single image pair is 67 seconds for the accurate architecture and 0.8 seconds for the fast architecture. Figure 5 contains a pair of examples from the KITTI 2012 data set, together with the predictions of our method.

Table 2 presents the frontrunners on the KITTI 2015 data sets. The error rates of our methods are 3.89 % for the accurate architecture and 4.46 % for the fast architecture, occupying first and second place on the leaderboard. Since one submission per paper is
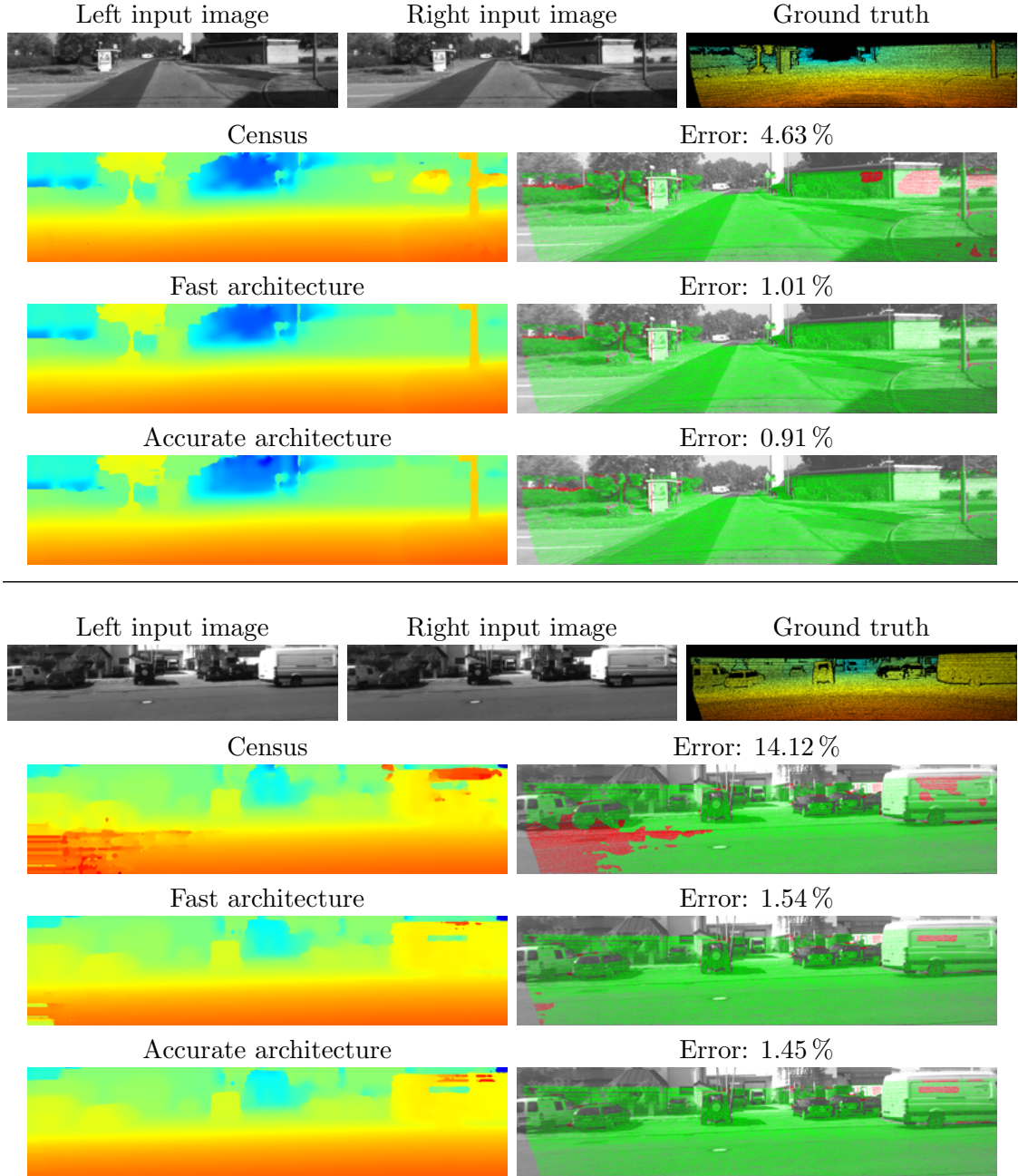
Figure 5: Examples of predicted disparity maps on the KITTI 2012 data set. Note how some regions of the image (the white wall in the top example, and the asphalt in the bottom example) cause problems for the census transform. The fast and the accurate architecture perform better, with the accurate architecture making fewer mistakes on average.

Year	Number of Image Pairs	Resolution	Maximum Disparity
2001	8	380×430	30
2003	2	1800×1500	220
2005	6	1400×1100	230
2006	21	1400×1100	230
2014	23	3000×2000	800

Table 3: A summary of the five Middlebury stereo data sets. The column "Number of Image Pairs" counts only the image pairs for which ground truth is available. The 2005 and 2014 data sets additionally contain a number of image pairs with ground truth disparities withheld; these image pairs constitute the test set.

allowed, only the result of the accurate architecture appears on the public leaderboard. See Figure 6 for the disparity maps produced by our method on the KITTI 2015 data set.

5.2 Middlebury Stereo Data Set

The image pairs of the Middlebury stereo data set are indoor scenes taken under controlled lighting conditions. Structured light was used to measure the true disparities with higher density and precision than in the KITTI data set. The data sets were published in five separate works in the years 2001, 2003, 2005, 2006, and 2014 (Scharstein and Szeliski, 2002, 2003; Scharstein and Pal, 2007; Hirschmüller and Scharstein, 2007; Scharstein et al., 2014). In this paper, we refer to the Middlebury data set as the concatenation of all five data sets; a summary of each is presented in Table 3.

Each scene in the 2005, 2006, and 2014 data sets was taken under a number of lighting conditions and shutter exposures, with a typical image pair taken under four lighting conditions and seven exposure settings for a total of 28 images of the same scene.

An online leaderboard[3], similar to the one provided by KITTI, displays a ranked list of all submitted methods. Participants have only one opportunity to submit their results on the test set to the public leaderboard. This rule is stricter than the one on the KITTI data set, where submissions are allowed every three days. The test set contains 15 images borrowed from the 2005 and 2014 data sets.

The data set is provided in full, half, and quarter resolution. The error is computed at full resolution; if the method outputs half or quarter resolution disparity maps, they are upsampled before the error is computed. We chose to run our method on half resolution images because of the limited size of the graphic card's memory available.

Rectifying a pair of images using standard calibration procedures, like the ones present in the OpenCV library, results in vertical disparity errors of up to nine pixels on the Middlebury data set (Scharstein et al., 2014). Each stereo pair in the 2014 data set is rectified twice: once using a standard, imperfect approach, and once using precise 2D correspondences for perfect rectification (Scharstein et al., 2014). We train the network on imperfectly rectified

---

3. The Middlebury scoreboard: `http://vision.middlebury.edu/stereo/eval3/`

Left input image	Right input image	Ground truth

Census	Error: 4.58 %

Fast architecture	Error: 2.79 %

Accurate architecture	Error: 2.36 %

Left input image	Right input image	Ground truth

Census	Error: 5.25 %

Fast architecture	Error: 3.91 %

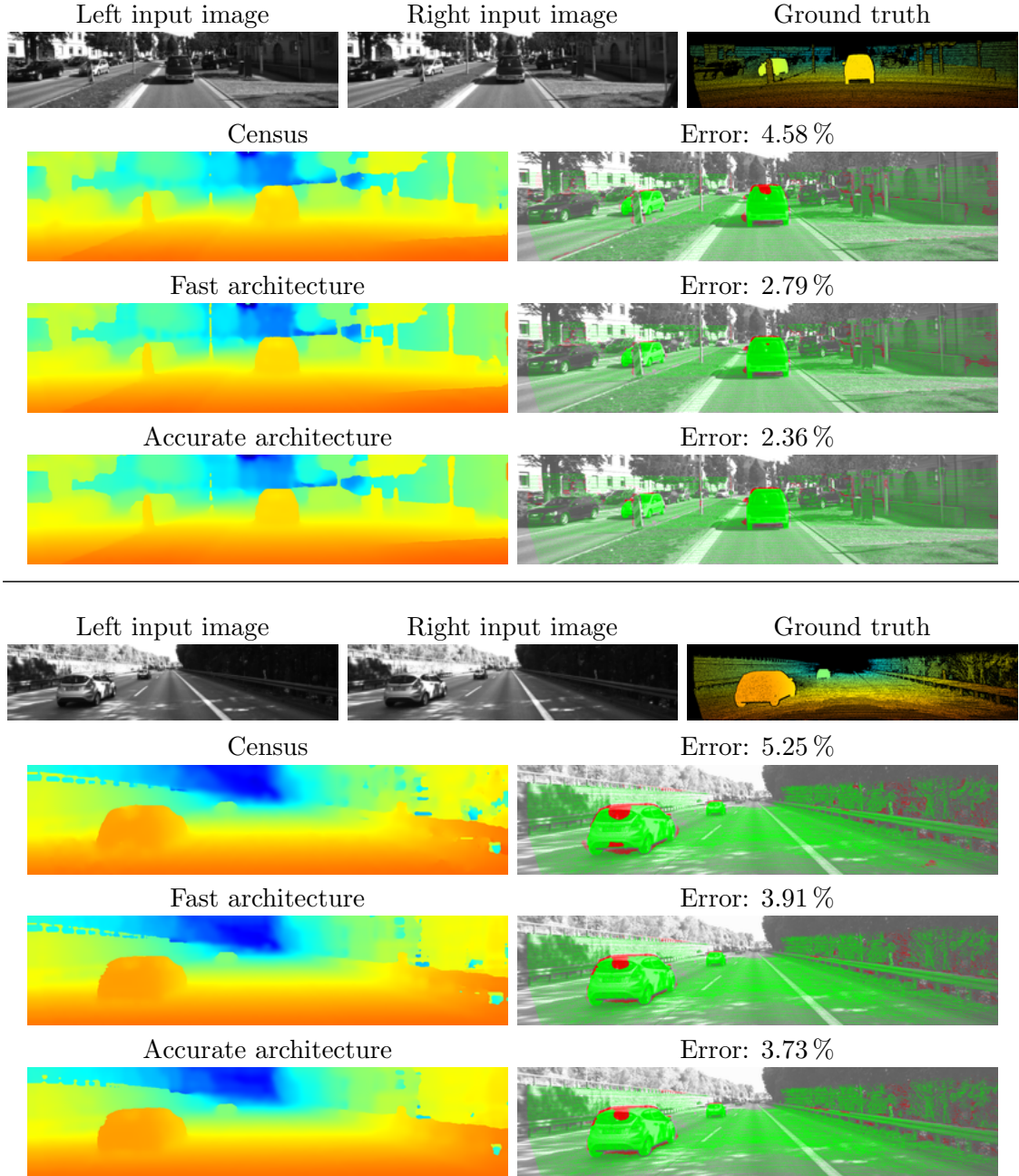| Accurate architecture	Error: 3.73 %

Figure 6: Examples of predictions on the KITTI 2015 data set. Observe that vehicles in motion are labeled densely in the KITTI 2015 data set.

Rank	Method		Resolution	Error	Runtime
1	**MC-CNN-acrt**	**Accurate architecture**	Half	8.29	150
2	MeshStereo	Zhang et al. (2015)	Half	13.4	65.3
3	LCU	Unpublished work	Quarter	17.0	6567
4	TMAP	Psota et al. (2015)	Half	17.1	2435
5	IDR	Kowalczuk et al. (2013)	Half	18.4	0.49
6	SGM	Hirschmüller (2008)	Half	18.7	9.90
7	LPS	Sinha et al. (2014)	Half	19.4	9.52
8	LPS	Sinha et al. (2014)	Full	20.3	25.8
9	SGM	Hirschmüller (2008)	Quarter	21.2	1.48
10	SNCC	Einecke and Eggert (2010)	Half	22.2	1.38

Table 4: The top ten methods on the Middlebury stereo data set as of October 2015. The "Error" column is the weighted average error after upsampling to full resolution and "Runtime" is the time, in seconds, required to process one pair of images.

image pairs, since only two of the fifteen test images (*Australia* and *Crusade*) are rectified perfectly.

The error is measured as the percentage of pixels where the true disparity and the predicted disparity differ by more than two pixels; this corresponds to an error tolerance of one pixel at half resolution. The error on the evaluation server is, by default, computed only on non-occluded pixels. The final error reported online is the weighted average over the fifteen test images, with weights set by the authors of the data set.

Table 4 contains a snapshot of the third, and newest, version of the Middlebury leaderboard. Our method ranks first with an error rate of 8.29 % and a substantial lead over the second placed MeshStereo method, whose error rate is 13.4 %. See Figure 7 for disparity maps produced by our method on one image pair from the Middlebury data set.

5.3 Details of Learning

We construct a binary classification data set from all available image pairs in the training set. The data set contains 25 million examples on the KITTI 2012, 17 million examples on the KITTI 2015, and 38 million examples on the Middlebury data set.

At training time, the input to the network was a batch of 128 pairs of image patches. At test time, the input was the entire left and right image. We could have used entire images during training as well, as it would allow us to implement the speed optimizations described in Section 3.3. There were several reasons why we preferred to train on image patches: it was easier to control the batch size, the examples could be shuffled so that one batch contained patches from several different images, and it was easier to maintain the same number of positive and negative examples within a batch.

We minimized the loss using mini-batch gradient descent with the momentum term set to 0.9. We trained for 14 epochs with the learning rate initially set to 0.003 for the accurate architecture and 0.002 for the fast architecture. The learning rate was decreased
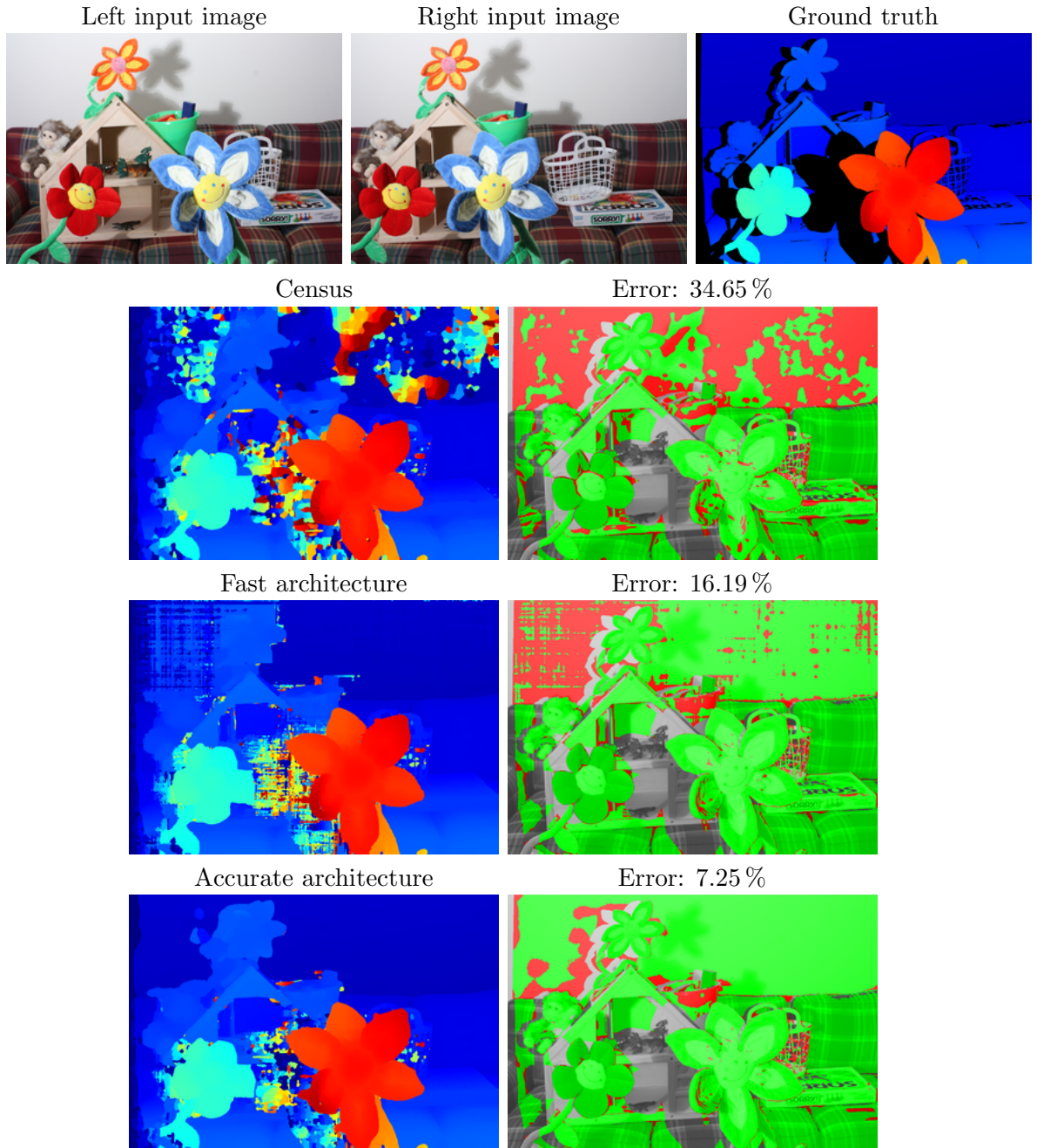
Figure 7: An example of a particularly difficult image pair from the Middlebury data set; the white wall in the background is practically textureless. The accurate architecture is able to classify most of it correctly. The fast architecture doesn't do as well but still performs better than census.

Hyperparameter	KITTI 2012		KITTI 2015		Middlebury	
	fst	acrt	fst	acrt	fst	acrt
input_patch_size	9×9	9×9	9×9	9×9	11×11	11×11
num_conv_layers	4	4	4	4	5	5
num_conv_feature_maps	64	112	64	112	64	112
conv_kernel_size	3	3	3	3	3	3
num_fc_layers		4		4		3
num_fc_units		384		384		384
dataset_neg_low	4	4	4	4	1.5	1.5
dataset_neg_high	10	10	10	10	6	18
dataset_pos	1	1	1	1	0.5	0.5
cbca_intensity		0.13		0.03		0.02
cbca_distance		5		5		14
cbca_num_iterations_1		2		2		2
cbca_num_iterations_2		0		4		16
sgm_P1	4	1.32	2.3	2.3	2.3	1.3
sgm_P2	223	32	42.3	55.8	55.9	18.1
sgm_Q1	3	3	3	3	4	4.5
sgm_Q2	7.5	6	6	6	8	9
sgm_V	1.5	2	1.25	1.75	1.5	2.75
sgm_D	0.02	0.08	0.08	0.08	0.08	0.13
blur_sigma	7.74	6	4.64	6	6	1.7
blur_threshold	5	6	5	5	2	2

Table 5: The hyperparameter values we used for the fast and accurate architectures (abbreviated "fst" and "acrt"). Note that hyperparameters concerning image intensity values (cbca_intensity and sgm_D) apply to the preprocessed images and not to raw images with intensity values in the range from 0 to 255.

by a factor of 10 on the 11th epoch. The number of epochs, the initial learning rate, and the learning rate decrease schedule where treated as hyperparameters and were optimized with cross-validation. Each image was preprocessed by subtracting the mean and dividing by the standard deviation of its pixel intensity values. The left and right image of a stereo pair were preprocessed separately. Our initial experiments suggested that using color information does not improve the quality of the disparity maps; therefore, we converted all color images to grayscale.

The post-processing steps of the stereo method were implemented in CUDA (Nickolls et al., 2008), the network training was done with the Torch environment (Collobert et al., 2011) using the convolution routines from the cuDNN library (Chetlur et al., 2014). The OpenCV library (Bradski, 2000) was used for the affine transformation in the data augmentation step.

19

The hyperparameters where optimized with manual search and simple scripts that helped automate the process. The hyperparameters we selected are shown in Table 5.

5.4 Data Set Augmentation

Augmenting the data set by repeatedly transforming the training examples is a commonly employed technique to reduce the network's generalization error. The transformations are applied at training time and do not affect the runtime performance. We randomly rotate, scale and shear the training patches; we also change their brightness and contrast. Since the transformations are applied to patches after they have been extracted from the images, the data augmentation step does not alter the ground truth disparity map or ruin the rectification.

The parameters of the transformation are chosen randomly for each pair of patches, and after one epoch of training, when the same example is being presented to the network for the second time, new random parameters are selected. We choose slightly different transformation parameters for the left and right image; for example, we would rotate the left patch by 10 degrees and the right by 14. Different data sets benefited from different types of transformations and, in some cases, using the wrong transformations increased the error.

On the Middlebury data set we took advantage of the fact that the images were taken under different lighting conditions and different shutter exposures by training on all available images. The same data set augmentation parameters were used for the KITTI 2012 and KITTI 2015 data sets.

The Middlebury test data sets contains two images worth mentioning: *Classroom*, where the right image is underexposed and, therefore, darker than the left; and *Djembe*, where the left and right images were taken under different light conditions. To handle these two cases we train, 20 % of the time, on images where either the shutter exposure or the arrangements of lights are different for the left and right image.

We combat imperfect rectification on the Middlebury data set by including a small vertical disparity between the left and right image patches.

Before describing the steps of data augmentation, let us introduce some notation: in the following, a word in `typewriter` is used to denote the name of a hyperparameter defining a set, while the same word in *italic* is used to denote a number drawn randomly from that set. For example, `rotate` is a hyperparameter defining the set of possible rotations and *rotate* is a number drawn randomly from that set. The steps of data augmentation are presented in the following list:

- Rotate the left patch by *rotate* degrees and the right patch by *rotate + rotate_diff* degrees.

- Scale the left patch by *scale* and the right patch by *scale · scale_diff*.

- Scale the left patch in the horizontal direction by *horizontal_scale* and the right patch by *horizontal_scale · horizontal_scale_diff*.

- Shear the left patch in the horizontal direction by *horizontal_shear* and the right patch by *horizontal_shear + horizontal_shear_diff*.

	KITTI 2012		Middlebury	
Hyperparameter	Range	Error	Range	Error
rotate	$[-7, 7]$	2.65	$[-28, 28]$	7.99
scale			$[0.8, 1]$	8.17
horizontal_scale	$[0.9, 1]$	2.62	$[0.8, 1]$	8.08
horizontal_shear	$[0, 0.1]$	2.61	$[0, 0.1]$	7.91
brightness	$[0, 0.7]$	2.61	$[0, 1.3]$	8.16
contrast	$[1, 1.3]$	2.63	$[1, 1.1]$	7.95
vertical_disparity			$[0, 1]$	8.05
rotate_diff			$[-3, 3]$	8.00
horizontal_scale_diff			$[0.9, 1]$	7.97
horizontal_shear_diff			$[0, 0.3]$	8.05
brightness_diff	$[0, 0.3]$	2.63	$[0, 0.7]$	7.92
contrast_diff			$[1, 1.1]$	8.01
No data set augmentation		2.73		8.75
Full data set augmentation		2.61		7.91

Table 6: The hyperparameters governing data augmentation and how they affect the validation error. The "Error" column reports the validation error when a particular data augmentation step is not used. The last two rows report validation errors with and without data augmentation. For example, the validation error on the KITTI 2012 is 2.73 % if no data augmentation is used, 2.65 % if all steps except rotation are used, and 2.61 % if all data augmentation steps are used.

- Translate the right patch in the vertical direction by *vertical_disparity*.

- Adjust the brightness and contrast by setting the left and right image patches to:

$$\mathcal{P}^L \leftarrow \mathcal{P}^L \cdot contrast + brightness \text{ and}$$
$$\mathcal{P}^R \leftarrow \mathcal{P}^R \cdot (contrast \cdot contrast\_diff) + (brightness + brightness\_diff),$$

with addition and multiplication carried out element-wise where appropriate.

Table 6 contains the hyperparameters used and measures how each data augmentation step affected the validation error.

Data augmentation reduced the validation error from 2.73 % to 2.61 % on the KITTI 2012 data set and from 8.75 % to 7.91 % on the Middlebury data set.

5.5 Runtime

We measure the runtime of our implementation on a computer with a NVIDIA Titan X graphics processor unit. Table 7 contains the runtime measurements across a range of hyperparameter settings for three data sets: KITTI, Middlebury half resolution, and a

new, fictitious data set, called Tiny, which we use to demonstrate the performance of our method on the kind of images typically used for autonomous driving or robotics. The sizes of images we measured the runtime on were: 1242×350 with 228 disparity levels for the KITTI data set, 1500×1000 with 200 disparity levels for the Middlebury data set, and 320×240 with 32 disparity levels for the Tiny data set.

Table 7 reveals that the fast architecture is up to 90 times faster than the accurate architecture. Furthermore, the running times of the fast architecture are 0.78 seconds on KITTI, 2.03 seconds on Middlebury, and 0.06 seconds on the Tiny data set. We can also see that the fully-connected layers are responsible for most of the runtime in the accurate architecture, as the hyperparameters controlling the number of convolutional layer and the number of feature maps have only a small effect on the runtime.

Training times depended on the size of the data set and the architecture, but never exceeded two days.

5.6 Matching Cost

We argue that the low error rate of our method is due to the convolutional neural network and not a superior stereo method. We verify this claim by replacing the convolutional neural network with three standard approaches for computing the matching cost:

- *The sum of absolute differences* computes the matching cost according to Equation (1), that is, the matching cost between two image patches is computed by summing the absolute differences in image intensities between corresponding locations. We used 9×9 patches.

- *The census transform* (Zabih and Woodfill, 1994) represents each image position as a bit vector. The size of this vector is a hyperparameter whose value, after examining several, we set to 81. The vector is computed by cropping a 9×9 image patch centered around the position of interest and comparing the intensity values of each pixel in the patch to the intensity value of the pixel in the center. When the center pixel is brighter the corresponding bit is set. The matching cost is computed as the hamming distance between two census transformed vectors.

- *Normalized cross-correlation* is a window-based method defined with the following equation:

$$C_{NCC}(\mathbf{p}, d) = \frac{\sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} I^L(\mathbf{q}) I^R(\mathbf{q} - \mathbf{d})}{\sqrt{\sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} I^L(\mathbf{q})^2 \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} I^R(\mathbf{q} - \mathbf{d})^2}}.$$

  The normalized cross-correlation matching cost computes the cosine similarity between the left and right image patch, when the left and right image patches are viewed as vectors instead of matrices. This is the same function that is computed in the last two layers of the fast architecture (normalization and dot product). The neighbourhood $\mathcal{N}_{\mathbf{p}}$ was set to a square 11×11 window around $\mathbf{p}$.

The "sad", "cens", and "ncc" columns of Table 8 contain the results of the sum of absolute differences, the census transform, and normalized cross-correlation on the KITTI 2012, KITTI 2015, and Middlebury data sets. The validation errors in the last rows of Table 8

Hyperparameter		KITTI fst	KITTI acrt	Middlebury fst	Middlebury acrt	Tiny fst	Tiny acrt
num_conv_layers	1	0.23	66.1	0.70	74.9	0.01	1.7
	2	0.26	66.2	0.82	75.2	0.01	1.8
	3	0.30	66.3	0.97	75.4	0.02	1.8
	4	0.34	66.4	1.11	75.6	0.03	1.8
	5	0.38	66.5	1.24	75.7	0.03	1.9
	6	0.42	66.7	1.37	76.0	0.04	1.9
num_conv_feature_maps	16	0.09	59.4	0.27	64.8	0.01	1.6
	32	0.15	60.4	0.51	66.2	0.01	1.6
	48	0.25	61.5	0.94	68.2	0.02	1.7
	64	0.34	62.7	1.24	70.0	0.03	1.7
	80	0.44	64.0	1.63	72.0	0.04	1.8
	96	0.53	65.3	1.93	73.9	0.04	1.8
	112	0.61	66.4	2.28	75.7	0.05	1.8
	128	0.71	67.7	2.61	77.8	0.06	1.9
num_fc_layers	1		16.3		25.3		0.5
	2		32.9		50.7		0.9
	3		49.6		75.7		1.4
	4		66.4		101.2		1.8
	5		82.9		126.4		2.3
num_fc_units	128		17.4		21.4		0.6
	256		38.5		44.9		1.1
	384		66.4		75.7		1.8
	512		101.0		113.3		2.7
No stereo method		0.34	66.4	1.24	75.7	0.03	1.8
Full stereo method		0.78	67.1	2.03	84.8	0.06	1.9

Table 7: The time, in seconds, required to compute the matching cost, that is, the time spent in the convolutional neural network without any post-processing steps. The time does include computing the matching cost twice: once when the left image is taken to be the reference image and once when the right image is taken to be the reference image. We measure the runtime as a function of four hyperparameters controlling the network architecture; for example, the first six rows contain the runtime as the number of convolutional layers in the network increases from one to six. The last row of the table contains the running time for the entire method, including the post-processing steps. As before, we abbreviate the fast and accurate architectures as "fst" and "acrt".

should be used to compare the five methods. On all three data sets the accurate architecture performs best, followed by the fast architecture, which in turn is followed by the census transform. These are the three best performing methods on all three data sets. Their error rates are 2.61 %, 3.02 %, and 4.90 % on KITTI 2012; 3.25 %, 3.99 %, and 5.03 % on KITTI 2015; and 7.91 %, 9.87 %, and 16.72 % on Middlebury. The sum of absolute differences and the normalized cross-correlation matching costs produce disparity maps with larger errors. For a visual comparison of our method and the census transform see Figures 5, 6, and 7.

5.7 Stereo Method

The stereo method includes a number of post-processing steps: cross-based cost aggregation, semiglobal matching, interpolation, subpixel enhancement, a median, and a bilateral filter. We ran a set of experiments in which we excluded each of the aforementioned steps and recorded the validation error (see Table 8).

The last two rows of Table 8 allude to the importance of the post-processing steps of the stereo method. We see that, if all post-processing steps are removed, the validation error of the accurate architecture increases from 2.61 % to 13.49 % on KITTI 2012, from 3.25 % to 13.38 % on KITTI 2015, and from 7.91 % to 28.33 % on Middlebury.

Out of all post-processing steps of the stereo method, semiglobal matching affects the validation error the strongest. If we remove it, the validation error increases from 2.61 % to 4.26 % on KITTI 2012, from 3.25 % to 4.51 % on KITTI 2015, and from 7.91 % to 11.99 % on Middlebury.

We did not use the left-right consistency check to eliminate errors in occluded regions on the Middlebury data set. The error rate increased from 7.91 % to 8.22 % using the left-right consistency check on the accurate architecture, which is why we decided to remove it.

5.8 Data Set Size

We used a supervised learning approach to measure the similarity between image patches. It is, therefore, natural to ask how does the size of the data set affect the quality of the disparity maps. To answer this question, we retrain our networks on smaller training sets obtained by selecting a random set of examples (see Table 9).

We observe that the validation error decreases as we increase the number of training examples. These experiments suggest a simple strategy for improving the results of our stereo method: collect a larger data set.

5.9 Transfer Learning

Up to this point the training and validation sets were created from the same stereo data set, either KITTI 2012, KITTI 2015, or Middlebury. To evaluate the performance of our method in the transfer learning setting, we run experiments where the validation error is computed on a different data set than the one used for training. For example, we would use the Middlebury data set to train the matching cost neural network and evaluate its performance on the KITTI 2012 data set. These experiments give us some idea of the expected performance in a real-world application, where it isn't possible to train a specialized

	KITTI 2012				
	fst	acrt	sad	cens	ncc
Cross-based cost aggregation	3.02	2.73	8.22	5.21	8.93
Semiglobal matching	8.78	4.26	19.58	8.84	10.72
Interpolation	3.48	2.96	9.21	5.96	11.16
Subpixel Enhancement	3.03	2.65	8.16	4.95	8.93
Median filter	3.03	2.63	8.16	4.92	9.00
Bilateral filter	3.26	2.79	8.75	5.70	9.76
No stereo method	15.70	13.49	32.30	53.55	22.21
Full stereo method	3.02	2.61	8.16	4.90	8.93
	KITTI 2015				
	fst	acrt	sad	cens	ncc
Cross-based cost aggregation	3.99	3.39	9.94	5.20	8.89
Semiglobal matching	8.40	4.51	19.80	7.25	9.36
Interpolation	4.47	3.33	10.39	5.83	10.98
Subpixel Enhancement	4.02	3.28	9.44	5.03	8.91
Median filter	4.05	3.25	9.44	5.05	8.96
Bilateral filter	4.20	3.43	9.95	5.84	9.77
No stereo method	15.66	13.38	30.67	50.35	18.95
Full stereo method	3.99	3.25	9.44	5.03	8.89
	Middlebury				
	fst	acrt	sad	cens	ncc
Cross-based cost aggregation	9.87	10.63	43.09	29.28	33.89
Semiglobal matching	25.50	11.99	51.25	19.51	35.36
Interpolation	9.87	7.91	41.86	16.72	33.89
Subpixel Enhancement	10.29	8.44	42.71	17.18	34.12
Median filter	10.16	7.91	41.90	16.73	34.17
Bilateral filter	10.39	7.96	41.97	16.96	34.43
No stereo method	30.84	28.33	59.57	64.53	39.23
Full stereo method	9.87	7.91	41.86	16.72	33.89

Table 8: The numbers measure validation error when a particular post-processing step is excluded from the stereo method. The last two rows of the tables should be interpreted differently: they contain the validation error of the raw convolutional neural network and the validation error after the complete stereo method. For example, if we exclude semiglobal matching, the fast architecture achieves an error rate of 8.78 % on the KITTI 2012 data set and an error rate of 3.02 % after applying the full stereo method. We abbreviate the method names as "fst" for the fast architecture, "acrt" for the accurate architecture, "sad" for the sum of absolute differences, "cens" for the census transform, and "ncc" for the normalized cross-correlation matching cost.

Data Set Size (%)	KITTI 2012		KITTI 2015		Middlebury	
	fst	acrt	fst	acrt	fst	acrt
20	3.17	2.84	4.13	3.53	11.14	9.73
40	3.11	2.75	4.10	3.40	10.35	8.71
60	3.09	2.67	4.05	3.34	10.14	8.36
80	3.05	2.65	4.02	3.29	10.09	8.21
100	3.02	2.61	3.99	3.25	9.87	7.91

Table 9: The validation error as a function of training set size.

		Test Set					
		KITTI 2012		KITTI 2015		Middlebury	
		fst	acrt	fst	acrt	fst	acrt
	KITTI 2012	3.02	2.61	4.12	3.99	12.78	11.09
Training Set	KITTI 2015	3.60	4.28	3.99	3.25	13.70	14.19
	Middlebury	3.16	3.07	4.48	4.49	9.87	7.91

Table 10: The validation error when the training and test sets differ. For example, the validation error is 3.16 % when the Middlebury data set is used for training the fast architecture and the trained network is tested on the KITTI 2012 data set.

network because no ground truth is available. The results of these experiments are shown in Table 10.

Some results in Table 10 were unexpected. For example, the validation error on KITTI 2012 is lower when using the Middlebury training set compared to the KITTI 2015 training set, even though the KITTI 2012 data set is obviously more similar to KITTI 2015 than Middlebury. Furthermore, the validation error on KITTI 2012 is lower when using the fast architecture instead of the accurate architecture when training on KITTI 2015.

The matching cost neural network trained on the Middlebury data set transfers well to the KITTI data sets. Its validation error is similar to the validation errors obtained by networks trained on the KITTI data sets.

5.10 Hyperparameters

Searching for a good set of hyperparameters is a daunting task—with the search space growing exponentially with the number of hyperparameters and no gradient to guide us. To better understand the effect of each hyperparameter on the validation error, we conduct a series of experiments where we vary the value of one hyperparameter while keeping the others fixed to their default values. The results are shown in Table 11 and can be summarized by observing that increasing the size of the network improves the generalization

Hyperparameter		KITTI 2012		KITTI 2015		Middlebury	
		fst	acrt	fst	acrt	fst	acrt
num_conv_layers	1	5.96	3.97	5.61	4.06	20.74	12.37
	2	3.52	2.98	4.19	3.45	12.11	9.20
	3	3.10	2.72	4.04	3.27	10.81	8.56
	4	3.02	2.61	3.99	3.25	10.26	8.21
	5	3.03	2.64	3.99	3.30	9.87	7.91
	6	3.05	2.70	4.01	3.38	9.71	8.11
num_conv_feature_maps	16	3.33	2.84	4.32	3.51	11.79	10.06
	32	3.15	2.68	4.12	3.35	10.48	8.67
	48	3.07	2.66	4.06	3.32	10.20	8.47
	64	3.02	2.64	3.99	3.30	9.87	8.12
	80	3.02	2.64	3.99	3.29	9.81	7.95
	96	2.99	2.68	3.97	3.27	9.62	8.03
	112	2.98	2.61	3.96	3.25	9.59	7.91
	128	2.97	2.63	3.95	3.23	9.45	7.92
num_fc_layers	1		2.83		3.50		8.52
	2		2.70		3.31		8.33
	3		2.62		3.30		8.06
	4		2.61		3.25		8.00
	5		2.62		3.29		7.91
num_fc_units	128		2.72		3.36		8.44
	256		2.65		3.28		8.03
	384		2.61		3.25		7.91
	512		2.60		3.23		7.90
dataset_neg_low	1.0	3.00	2.76	3.97	3.35	9.84	8.00
	1.5	3.00	2.71	3.97	3.33	9.87	7.91
	2.0	2.99	2.63	3.98	3.31	9.98	8.08
	4.0	3.02	2.61	3.99	3.25	10.20	8.66
	6.0	3.06	2.63	4.05	3.28	10.13	8.86
dataset_neg_high	6	3.00	2.72	3.98	3.30	9.87	8.59
	10	3.02	2.61	3.99	3.25	9.97	8.23
	14	3.04	2.61	4.02	3.25	10.00	8.05
	18	3.07	2.60	4.06	3.23	9.98	8.11
	22	3.07	2.61	4.05	3.24	10.16	7.91
dataset_pos	0.0	3.04	2.67	4.00	3.26	9.92	7.97
	0.5	3.02	2.65	3.99	3.28	9.87	7.91
	1.0	3.02	2.61	3.99	3.25	9.86	8.04
	1.5	3.04	2.62	4.04	3.27	10.00	8.34
	2.0	3.04	2.66	4.04	3.29	10.16	8.51

Table 11: Validation errors computed across a range of hyperparameter settings.

performance, but only up to a point, when presumably, because of the size of the data set, the generalization performance starts do decrease.

Note that the `num_conv_layers` hyperparameter implicitly controls the size of the image patches. For example, a network with one convolutional layer with 3×3 kernels compares image patches of size 3×3, while a network with five convolutional layers compares patches of size 11×11.

6. Conclusion

We presented two convolutional neural network architectures for learning a similarity measure on image patches and applied them to the problem of stereo matching.

The source code of our implementation is available at `https://github.com/jzbontar/mc-cnn`. The online repository contains procedures for computing the disparity map, training the network, as well as the post-processing steps of the stereo method.

The accurate architecture produces disparity maps with lower error rates than any previously published method on the KITTI 2012, KITTI 2015, and Middlebury data sets. The fast architecture computes the disparity maps up to 90 times faster than the accurate architecture with only a small increase in error. These results suggest that convolutional neural networks are well suited for computing the stereo matching cost even for applications that require real-time performance.

The fact that a relatively simple convolutional neural network outperformed all previous methods on the well-studied problem of stereo is a rather important demonstration of the power of modern machine learning approaches.

References

G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.

Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1): 43–57, 2011.

Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):500–513, 2011.

Ayan Chakrabarti, Ying Xiong, Steven J. Gortler, and Todd Zickler. Low-level vision by consensus in a spatial hierarchy of regions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Zhuoyuan Chen, Xun Sun, Yinan Yu, Liang Wang, and Chang Huang. A deep visual correspondence embedding model for stereo matching costs. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. URL http://arxiv.org/abs/1410.0759.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

Nils Einecke and Julian Eggert. A two-stage correlation method for stereoscopic depth estimation. In *Digital Image Computing: International Conference on Techniques and Applications (DICTA)*, pages 227–234, 2010.

Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Proceedings of the 10th Asian Conference on Computer Vision - Volume Part I*, ACCV'10, pages 25–38. Springer-Verlag, Berlin, Heidelberg, 2011.

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

Fatma Güney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Ralf Haeusler, Rahul Nair, and Daniel Kondermann. Ensemble learning for confidence measures in stereo vision. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. MatchNet: Unifying feature and metric learning for patch-based matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008.

Heiko Hirschmüller and Daniel Scharstein. Evaluation of cost functions for stereo matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

Heiko Hirschmüller and Daniel Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1582–1599, 2009.

Michael Hornacek, Andrew Fitzgibbon, and Carsten Rother. SphereFlow: 6 DoF scene flow from RGB-D pairs. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Dan Kong and Hai Tao. A method for learning matching errors for stereo computation. *British Machine Vision Conference (BMVC)*, 2004.

Dan Kong and Hai Tao. Stereo matching via learning multiple experts behaviors. *British Machine Vision Conference (BMVC)*, 2006.

Jana Kostková and Radim Sára. Stratified dense matching for stereopsis in complex scenes. *British Machine Vision Conference (BMVC)*, 2003.

Jedrzej Kowalczuk, Eric T Psota, and Lance C Perez. Real-time stereo matching on CUDA using an iterative refinement method for adaptive support-weight correspondences. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(1):94–104, 2013.

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.

Yunpeng Li and Daniel P Huttenlocher. Learning for stereo vision using the structured support vector machine. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008.

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

Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.

Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronin, and Cordelia Schmid. Local convolutional features with unsupervised training for image retrieval. In *IEEE International Conference on Computer Vision (ICCV)*, pages 91–99, 2015.

Martin Peris, Atsuto Maki, Sara Martull, Yasuhiro Ohkawa, and Kazuhiro Fukui. Towards a simulation driven stereo vision system. In *21st International Conference on Pattern Recognition (ICPR)*, pages 1038–1042, 2012.

Eric T Psota, Jedrzej Kowalczuk, Mateusz Mittek, and Lance C Perez. Map disparity estimation using hidden markov trees. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Deepmatching: Hierarchical deformable dense matching. *ArXiv e-prints*, 1(7):8, 2015.

Daniel Scharstein and Chris Pal. Learning conditional random fields for stereo. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2007.

Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3): 7–42, 2002.

Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2003.

Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. *German Conference on Pattern Recognition (GCPR)*, September 2014.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1573–1585, 2014.

Sudipta N Sinha, Daniel Scharstein, and Richard Szeliski. Efficient high-resolution stereo matching using local plane sweeps. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Aristotle Spyropoulos, Nikos Komodakis, and Philippos Mordohai. Learning to detect ground control points for improving the accuracy of stereo matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

Tomasz Trzcinski, Mario Christoudias, Vincent Lepetit, and Pascal Fua. Learning image descriptors with the boosting-trick. In *Advances in neural information processing systems*, pages 269–277, 2012.

Christoph Vogel, Konrad Schindler, and Stefan Roth. Piecewise rigid scene flow. *IEEE International Conference on Computer Vision (ICCV)*, 2013.

Christoph Vogel, Stefan Roth, and Konrad Schindler. View-consistent 3D scene flow estimation over multiple frames. *European Conference on Computer Vision (ECCV)*, September 2014.

Christoph Vogel, Konrad Schindler, and Stefan Roth. 3D scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, pages 1–28, 2015.

Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. *European Conference on Computer Vision (ECCV)*, September 2014.

Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. *European Conference on Computer Vision (ECCV)*, 1994.

Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Jure Žbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Chi Zhang, Zhiwei Li, Yanhua Cheng, Rui Cai, Hongyang Chao, and Yong Rui. Meshstereo: A global stereo model with mesh alignment regularization for view interpolation. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009.

Li Zhang and Steven M Seitz. Estimating optimal parameters for MRF stereo from a single image pair. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2): 331–342, 2007.