Abstract

Stereo reconstruction from rectified images has recently been revisited within the context of deep learning. Using a deep Convolutional Neural Network to obtain patch-wise matching cost volumes has resulted in state of the art stereo reconstruction on classic datasets like Middlebury and Kitti. By introducing this cost into a classical stereo pipeline, the final results are improved dramatically over non-learning based cost models. However these pipelines typically include hand engineered post processing steps to effectively regularize and clean the result. Here, we show that it is possible to take a more holistic approach by training a fully end-to-end network which directly includes regularization in the form of a densely connected Conditional Random Field (CRF) that acts as a prior on inter-pixel interactions. We demonstrate that our approach applied to both synthetic and real world datasets outperforms an alternative end-to-end network and compares favorably to less holistic approaches.

1. Introduction

Stereo reconstruction is a well studied problem in the field of computer vision. It is based on the notion that by taking two images of a scene from slightly different angles, one can reconstruct the depth of the scene by triangulation. This is in fact how humans observe the world. However, despite the apparent ease with which we perceive depth, the computational version of the problem is far from trivial. Many approaches have been developed to solve the problem of stereo depth perception and in general they all consist of two main stages: correspondence and regularization. The correspondence stage attempts to find a match for each of the pixels between the stereo images, while the regularization typically applies global constraints such as smoothness priors to the matching process to improve the fidelity and accuracy of the matched locations.

Convolutional Neural Networks (CNN) are an efficient and successful computer learning tool which have been demonstrated to solve many difficult problems in computer vision. The paper by Zbontar et al. [23] has recently shown that CNNs can provide state of the art results in stereo reconstruction, providing both accuracy and computational efficiency. That work however applied a neural network to the image correspondence part of the algorithm successfully but did not deal with new regularization techniques, instead relying on existing algorithms. In a similar context Mayer et al. [17] also applied a CNN methodology to the stereo matching problem, but performed the disparity prediction in an end-to-end manner without including a specific regularization stage. Their score on the KITTI [14] benchmark is somewhat lower than that of [23] which may be attributed to training with an $L_2$ norm regularizer which is prone to smoothing around object boundaries. We instead propose an end-to-end network which aims to preserve sharper object boundaries while explicitly incorporating a powerful regularization stage in the form of a Conditional Random Field (CRF). Our method uses a mean-field network layer as first shown in [25]. Doing so imposes a more natural piecewise smoothness prior, which assumes local smoothness with sparse discontinuities around object borders. The parameters controlling these priors are tunable and are learned directly from examples within the CNN framework. Furthermore we introduce a novel compatibility function for our CRF prior to better represent disparity neighborhood interactions. In this paper we show how this network can be constructed and trained using both real world and synthetic data. We discuss the benefits obtained by including the CRF stage as part of the network and compare our results to those achieved by [17, 23].

2. Related work

In recent years large stereo datasets with ground truth disparity such as KITTI [4, 14] and Middlebury [20] have been made available to the research community. These real life datasets have been supplemented by much larger synthetic datasets [17] which were specifically created to en-
able deep learning for stereo matching. This has led to a new generation of stereo reconstruction algorithms with model parameters that are tuned from real-world or synthetic data. Stereo reconstruction which is based on supervised learning can be roughly characterized into three main groups which we briefly review here. For a more general treatment of stereo algorithms we refer the reader to [20].

**Non-CNN learning based methods.** The multiple-experts method of [7,8] calculates an initial matching cost and then refines it using a classifier which is learned from data. Similarly [18] uses AD-Census [13] to calculate an initial matching cost and then learns the relation between the matching cost and final disparity using LDA. A more recent approach taken by [5] uses a random forest classifier to combine several confidence measures to improve the matching cost. In [21] the authors also use a random forest classifier for matching patches and then refine the result using an MRF model.

**CNN based methods** CNN based methods have been shown to be effective at computing the matching cost. A deep network was demonstrated in [22] which was trained to evaluate patch similarity. This idea was harnessed by [23] and [2] who applied a CNN to calculate the cost for matching two patches taken from stereo pairs. The CNNs are trained on patches taken from the labeled dataset and the locally calculated matching cost is directly used to estimate the final disparity without taking into account global priors during the training process. In contrast to [23], [17] has taken an end-to-end deep learning approach to the stereo matching problem. In this work, a deep network is used and trained in a multi-scale end-to-end manner. Additionally, [17] created large synthetic datasets for the purpose of training their network.

**CRF/MRF based methods** Markov and conditional random fields have long been used to impose global priors onto noisy disparity estimations. An MRF model demonstrated in [24] learns hyper-parameters from a single stereo pair. Similarly, the work of [19] shows how constructing a labeled dataset consisting of 30 stereo pairs can be used to train the parameters of a CRF. An alternative parameter training approach for CRF based denoising is shown in [11] where the model parameters are determined using a structured support vector machine.

Within this rough taxonomy of methods ours directly combines CNN and CRF based learning into a new end-to-end pipeline. In contrast to [19,11,24] our model uses a dense CRF which is used to directly improve the matching cost volume. It can be effectively characterized as a fusion between the CRF and CNN based approaches and thus inherits the benefits of both. The primary contribution of our work lies in our exposition on how to integrate a trainable CRF into an end-to-end deep stereo matching pipeline. We now provide a brief overview of the CRF model to give a firm base to further discussion.

### 3. Conditional Random Fields

CRFs represent a family of graphical models for solving multi-label assignment problems over a field of random variables which are conditioned on observed data. In the context of computer vision, CRFs are useful for assigning labels to each image pixel, conditioned on image pixel data while imposing global constraints such as label smoothness. The field is described in terms of the random variables \( \{X_i\} \) which are associated with a labeling probability function \( P(X = x|I) \) over the set of possible labels \( \mathcal{L} = \{l_1, l_2, \ldots, l_k\} \), where \( I \) is the input image data. Given an undirected graph \( G = (V, E) \) in which each vertex is associated with a variable \( X_i \) and each edge is associated with a joint labeling probability, the joint probability distribution can be written as a Gibbs distribution of the form

\[
P(X = x|I) = \frac{1}{Z} \exp(-E(x|I))
\]

We call \( E(x) \) the energy of the configuration \( x \in \mathcal{L} \) and \( Z \) the partition function, which can be interpreted as a statistical normalization factor over all possible states. For notational convenience we will drop the explicit conditioning on \( I \). The MAP estimator for this distribution \( \hat{l} = \text{argmax} P(X = x) \) would be the labeling configuration that globally minimizes the energy \( E(x) \). In the CRF model the energy function for a label assignment is given by

\[
E(x) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j)
\]

where the unary terms \( \psi_u(x_i) \) signify the cost of assigning the label \( x_i \) to pixel \( i \), and the pairwise terms \( \psi_p(x_i, x_j) \) signify the joint energy of assigning the labels \( \{x_i, x_j\} \) to the pixel pair \((i, j)\). The unary term is set by a per node label probability distribution obtained by a labeling cost function that is dependent on the data. Similarly, the pairwise terms are used to penalize inconsistent labeling for neighboring nodes.

#### 3.1. Dense CRF with Gaussian potentials

A CRF based on a fully connected graph is called a dense CRF. The dense CRF model imposes global constraints between all the random variables \( X_i \) and is therefore expected to yield better results than a model based solely on local connections. The dense CRF optimization however is in general computationally expensive and was deemed impractical for many years until Krhenbhl et al. [9] showed that using Gaussian edge-weights for inter pixel interactions in a dense CRF enabled a linear time solution. The researchers used a mean field approximation to the CRF distribution, which substitutes the exact distribution \( P(X) \) with a distribution \( Q(X) \) such that the KL-divergence \( D(P||Q) \) is minimal among all possible dis-
tributions \( Q(X) \) and that can be expressed as \( Q(X) = \prod_i Q_i(X_i) \). This minimization is then performed iteratively using a message passing scheme and can be computed efficiently using high dimensional filtering schemes such as [1]. For our CRF model we use the same Gaussian potentials as proposed in [9]:

\[
\psi_{i,j}(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{k} w(m)k(m)(f_i, f_j), \tag{2}
\]

where each \( k(m) \) is a Gaussian kernel, \( w(m) \) is a weight for each kernel, \( \mu(x_i, x_j) \) is a labeling compatibility function and \( f_i, f_j \) are feature vectors extracted from the image at locations \( i, j \). The label compatibility function \( \mu(x_i, x_j) \) controls the likelihood of two neighboring pixels \( x_i, x_j \) to receive a specific combination of labels. Our pairwise terms consist of two Gaussian kernels defined as

\[
k(f_i, f_j) = w^{(1)} \exp \left( \frac{-|p_i - p_j|^2}{2\theta^2} - \frac{|I_i - I_j|^2}{2\beta^2} \right) \tag{3}
\]

where \( p_i, p_j \) are the image coordinates at point \((i, j)\) and \( I_i, I_j \) are the image RGB coordinates at point \((i, j)\). \( \theta_x, \theta_y, \) and \( \theta_z \) control the width of the Gaussian for each kernel. These kernels are in fact a Bilateral kernel and a spatial Gaussian kernel respectively. The minimization of the KL-divergence \( D(P||Q) \) can be achieved by an iterative update rule written as

\[
Q_i(x_i = l) = \frac{1}{z_i} \exp \left\{ -\psi_a(x_i) - \right. \\
\sum_{l' \in (L)} \mu(l, l') \sum_{m=1}^{k} w^{(m)} \sum_{j \neq i} k^{(m)}(f_i, f_j) Q_j(l') \tag{4}
\]

as shown in [9].

### 3.2. Dense CRF for stereo

CRF methods are typically used as regularizers which impose constraints onto noisy measurements. This makes them naturally useful for the stereo matching problem in which there are noisy correlations between left and right images. We must then choose the best correlation in a way that satisfies the observed data and the piece-wise smoothness prior. It is both interesting and instructive to note that the popular Semi Global Matching (SGM) [6] method is in fact an example of a CRF approximation with semi-global constraints which is solved efficiently by dynamic programming. We instead use the fully global dense CRF with Gaussian weights from [9] described in Section 3.1. To do so we will derive the gradient in Section 4.1 and use the method first proposed in [25] to propagate error gradients. This is enabled by the iterative update scheme from equation 4 which can be broken down into five steps that require SoftMax, convolution with Gaussian filters and other basic operations which can back propagate their derivatives. The authors of [25] show how these operations can be performed in the context of a recurrent neural network (CRF-as-RNN) thereby allowing back-propagation to update the parameters of the CRF. In our work we directly embed the full MC-CNN stereo matching network of [23] together with the dense CRF layer. During the backward pass the CRF layer propagates it’s gradient updates back through the Siamese CNN layers to fine tune those weights.

### 3.3. Stereo matching network

We make use of the stereo matching cost convolutional network (MC-CNN) proposed by [23]. In this work we chose the fast version for our experimentation, although the same methodology can be applied to the accurate version as well.

MC-CNN is a patch matching network with a Siamese architecture. The network architecture is described in [23]. In short it consists of several convolutional and ReLU layers which when applied to a patch output a patch descriptor. The patch descriptor is fed into a Stereo-Join layer which computes the inner products between the descriptors of all the possible matches. The network is trained with a hinge-loss function which given examples of correct and incorrect matches, learns to distinguish between them.

The final output of the network is the matching probability for each patch and it’s possible matches. The probability is obtained by calculating the inner product between the patch descriptors at the output of the network. The probabilities are stored in memory as a 3D data structure which we call the cost volume. This cost volume contains the matching probability for every possible match and is the input to the CRF stage of our end-to-end network.

In contrast to our network, the MC-CNN is trained solely on patch examples and is therefore a purely local method. As such, it produces a noisy result which requires further regularization. A noisy estimation of the disparity map can be directly obtained by using the \textit{max} operation on the network output. The refinement step is preformed directly on the cost volume further down the pipeline.

### 4. Parameter updates

The dense CRF approach with Gaussian potentials is a flexible model which is able to impose various smoothness
constraints. The constraints are controlled by a number of parameters which should be calibrated in order to obtain good priors. We choose to learn the best parameters using a training set with ground truth disparity from the KITTI autonomous driving dataset [14] and from the FlyingThings3D dataset [17]. The parameters which control the CRF are: three Gaussian widths $\theta_\alpha, \theta_\beta, \theta_\gamma$, the appearance kernel weights $w_1(1) \in \mathbb{R}^{d_{\text{max}}}$, the spatial kernel weights $w_2(2) \in \mathbb{R}^{d_{\text{max}}}$, and the compatibility function weights $\mu \in \mathbb{R}^{d_{\text{max}} \times d_{\text{max}}}$. According to [9] the Gaussian width parameters $\theta_a, \theta_b, \theta_c$ cannot be learned efficiently through gradient methods, and we therefore resolve to learn them using the Nelder-Mead (NM) downhill simplex method [16] which is a non-gradient based optimization technique. Before we fine tune our full end-to-end MC-CNN-CRF network we first use the NM optimization stage to minimize the average 3-pixel error over a small subset of image pairs from the training data. NM minimizes the function over the space of these variables by evaluating the function at various points which is a computationally reasonable action for this small set of variables but would be infeasible for more. After the minimization is performed we obtain the Gaussian width parameters which we use throughout this work. Additionally we obtain a very rough estimation of the Gaussian kernel weights which act as a good initialization for the gradient based learning process. In fact the kernel weights are integrated into the network through linear operations which are trivially differentiable and make them highly amenable to gradient descent updates as described in [25]. However, in contrast to [25] who formulate the multiplication of the kernel weights as a linear layer by extending the weights to a full $d_{\text{max}} \times d_{\text{max}}$ matrix, we formulate this operation as a convolution layer with a $1 \times 1$ kernel and $d_{\text{max}}$ input and output channels. This allows us to realize the network in a manner that follows [9] more accurately without extraneous parameters.

### 4.1. End-to-end learning

In order to fuse the MC-CNN and CRF stages into one end-to-end network we need to fully define our CRF module in terms of how it acts in a network forward pass as well as its backward pass gradients so that we can use it during training with back-propagation. The output of the MC-CNN network is a cost volume which we consider to be noisy. The function of the CRF module is to reduce the noisiness in the disparity cost matching volume so as to improve the final matching result. In practice this means feeding the full cost volume into the CRF module and producing a cost volume as the output. To do so, we unroll the CRF meanfield loop and convert each step into a neural network action in the same manner as [25]. Although we mostly follow their approach, the specific details of our method require non-trivial adaptation. The final cost volume produced as output by the MC-CNN network is computed as

$$C_{\text{net}}(p, d) = \mathcal{P}^L(p) \cdot \mathcal{P}^R(p - d),$$

(5)

where $p$ is a pixel coordinate and $d$ is a disparity value. The depth of the volume is simply the number of disparity values we choose to test over. Therefore $C_{\text{net}}(p, d)$ is the matching cost for pixel $p$ and disparity $d$ which is produced by the network. $\mathcal{P}^L$ and $\mathcal{P}^R$ are the descriptors produced for the patches at points $p$ and $p - d$ taken from the left and right images respectively and these are the outputs of the Siamese network which forms the core of MC-CNN. In order to attach the MC-CNN and CRF networks the cost volume output $C_{\text{net}}$ must become the input volume into the CRF. This implies that the operation described in Equation 5 must be defined as a network layer with backward propagation of gradients. Recently, [12] have described a similar layer which performs forward and backward propagation for the dot product operation between a single patch and it’s candidate matches. In our case however, since we are performing global optimization on entire images, we require this layer to produce the backward gradients for all inner products between all the image patches and their matching candidates. This operation is best described as a linear operation

$$\hat{\mathcal{C}}^L(p) = \begin{pmatrix} \mathcal{P}^L(p) \\ \mathcal{P}^L(p + 1) \\ \vdots \\ \mathcal{P}^L(p + d) \end{pmatrix} \mathcal{P}^R(p).$$

(6)

By treating the left image representation $\mathcal{P}^L$ as constants, it is easy to see that the adjoint operator required for gradient back propagation is the transpose of the matrix. By this reasoning we obtain the following rule for gradient propagation

$$\hat{G}^L_{\text{in}}(p) = \begin{pmatrix} \mathcal{P}^L(p) \\ \mathcal{P}^L(p + 1) \\ \vdots \\ \mathcal{P}^L(p + d) \end{pmatrix}^T \hat{G}^R_{\text{out}}(p).$$

(7)

Here $G_{\text{in}}$ and $G_{\text{out}}$ represent the input and output gradients respectively. The adjoint operator used for the gradient propagation to the right image descriptors is attained following the same logic such that

$$\hat{G}^R_{\text{in}}(p) = \begin{pmatrix} \mathcal{P}^R(p) \\ \mathcal{P}^R(p - 1) \\ \vdots \\ \mathcal{P}^R(p - d) \end{pmatrix}^T \begin{pmatrix} G^L_{\text{out}}(p, 0) \\ G^L_{\text{out}}(p - 1, 1) \\ \vdots \\ G^L_{\text{out}}(p - d, d) \end{pmatrix}.$$  

(8)

This scheme allows us to add the necessary gradient computation mechanism to fuse the MC-CNN network to the
CRF network and train them jointly using whole images as input and disparity probabilities as the output.

4.2. Training loss

We experimented with several loss functions such as MSE, KL-Divergence, Cross Entropy Loss and Piecewise linear loss [10]. From these loss functions the Multi-class Cross Entropy and Piecewise linear losses, which are applied to each pixel location separately, provided the best convergence. The PL loss is an extension of Cross Entropy which performs a linear interpolation between the discrete bins, allowing real numbers as labels. During our training attempts the network exhibited signs of lowering the loss rate while increasing the $L_1$ error relative to the ground truth. Although at first this appeared to be a symptom of overfitting, it became apparent that the loss function was encouraging the probability of correct labels to rise, but was not penalizing the bins of the incorrect labels. To remedy this we introduced an entropy penalty into our cost function which pushes the network output towards lower entropy. We also tested the $L_1$ penalty but found it to be less effective. The entropy penalty that we introduce is of the form $P = - \frac{1}{m} \sum_{i=1}^{m} \alpha p_i \log (p_i)$ and its gradient is of the form $p_i' = -\alpha (1 + \log (p_i))$. By adding this gradient to the loss gradients we achieve a more stable gradient descent which converges both in the loss rate and in the $L_1$ error domains. During training we use a penalty factor of $\alpha = 0.1$.

4.3. Implementation details

We use the open source code for the fast architecture for MC-CNN [23] made available by the authors. We then implemented the dense CRF network layers within the Torch and CUDA frameworks. We trained our network on an Nvidia TITAN-X GPU. The code for efficient Gaussian filtering on GPU was adopted from the implementations by [1] and [25]. The CRF network is attached to the output of the cost network via the inner-product layer described in [1] and [25]. The CRF network layers within the Torch and CUDA frameworks. We trained our network on an Nvidia TITAN-X GPU. We used the open source code for the fast architecture for MC-CNN [23] made available by the authors. We then implemented the dense CRF network layers within the Torch and CUDA frameworks. We trained our network on an Nvidia TITAN-X GPU. The code for efficient Gaussian filtering on GPU was adopted from the implementations by [1] and [25]. The CRF network is attached to the output of the cost network via the inner-product layer described in [1] and [25]. The CRF network layers within the Torch and CUDA frameworks. 

The values of $\mu (i, j)$ are displayed in Figure 1. The diagonal structure which arises from the learning process depicts the spatial smoothness prior which is imposed by the CRF, where nearby labels have high probability of receiving similar disparities. It is clear that the weights obtained on the KITTI dataset are not fully formed, specifically in the larger disparity region. We discuss this further in Section 7.

Figure 1: The final trained compatibility weights $\mu (i, j)$ are depicted. This diagonal structure which arises from the training depicts the learned CRF smoothness priors. On the left: weights trained from FlyingThings3D [17]. On the right: weights trained from KITTI [14].

The rule in equation 9 encodes the intuition that neighboring pixels are most likely to receive the same labels, and as the label difference becomes larger the likelihood declines linearly until it plateaus at 0 at a distance of 4. This provides a good approximation for a piece-wise smooth labeling prior. At each training iteration the network receives a stereo image pair from the training set, along with it’s corresponding ground truth disparity map. The disparity maps in KITTI are sparse and we therefore must take this into consideration. We create a mask for each ground truth image of the empty disparity locations, and use it to compute the loss function for the allowed positions only. Additionally, we exclude from the mask any occluded and over exposed pixels which might hinder the learning process. During the backward pass of the network we zero the gradients which emanate from the data-less regions and propagate only the useful gradients down the network. We found that the best results are gained using Adagrad optimization [3]. Initially we train only the CRF weights for 30 epochs while freezing the rest of the network, and later we release the weights in both parts of the network and allow them to learn jointly. We use an initial learning rate of 0.1 for the CRF weights and relay on Adagrad to set the learning rate appropriately. We train the joint network for an additional 50 epochs.

5. Experiments

We conducted our evaluation on the KITTI 2015 [15, 14] benchmark as well as on the FlyingThings3D [17] synthetic dataset. For both we set aside validation sets and trained...
on the remaining pairs. For KITTI we used 40 pairs for validation and in FlyingThings3D we used 100 pairs. We trained MC-CNN [23] on the training set alone and then trained our CRF model as described in Section 4.3. After the training process we tested the average 3-pixel error rate on the validation set using several configurations:

- MC-CNN followed by SGM as described by [23]
- MC-CNN followed by CRF in place of SGM
- MC-CNN followed by CRF and then SGM
- DispNetCorr1D as described by [17]

Each configuration was tested with and without the post processing methods described by [23].

6. Results

As shown in Table 1 we are able to outperform both methods when training and testing on the FlyingThings3D dataset. On the KITTI dataset our method outperforms [23] for 1% error but not for 3%. This indicates that our method is generally accurate, however it is affected by areas of high error in some cases. It is our belief that this is due to the fairly small size of the KITTI dataset which contains only 200 training samples compared to 20k samples in FlyingThings3D. Specifically there appears to be an underrepresentation in the number of nearby surfaces in the KITTI scenes, whereas this is not the case for the FlyingThings3D dataset. Since we are training in an end-to-end fashion using whole images as our input data, we are more susceptible to the lack of training data than [23] who train on image patches. During our work we noticed that using modern optimizer such as Adagrad [3] partially remedy this by increasing the learning rate for weights that rarely receive gradients. This training scheme dramatically improves on this issue, but still does not completely solve it in this case. Our results show that most of our errors are concentrated in large disparity regions. Examples of this can be seen in Figure 3. In other examples however we can see that our method is able to perceive finer details and also does not suffer from the streaking artifact of SGM. Examples of this are shown in Figure 2. The large disparity problem does not negatively impact our results on the FlyingThings3D dataset. Several representative examples are shown in Figure 4. The fact that our proposed method outperforms that of [23] on both datasets illustrates that including a powerful regularizer into an end-to-end trained system has a positive impact on the quantitative results. This is advantage is further highlighted when a very large dataset is available for training. This indicates that our approach is most suited to be truly beneficial to large datasets with noisy data that have a good coverage of their data space. In addition, our method coupled with SGM can increase the performance of the disparity recovery further by harnessing the advantages of both methods as shown in Table 1.

7. Conclusions

In this work we have described a novel fused CNN and CRF pipeline for stereo reconstruction that permits end-to-end training. Our results indicate that this direction is fruitful and provides a more holistic, less fragmented solution to stereo reconstruction which is a problem that has typically been highly engineered. We show that for the specific task of stereo matching we are able to beat competing CNN based methods if the dataset is large enough. When data is more scarce our approach is still able to achieve very competitive results, especially in 1% error rate. It should be noted that poorly covered characteristics in the data will yield errors in the predicted disparities (Figure 3). It is our belief that improving the training methods and data processing, our method can overcome the large disparity issue on KITTI which would greatly improve the results on future datasets. On a more general level we believe that our findings justify advocating to other researchers the benefits of focusing efforts on building specialized network architectures which implement more task specific classical algorithms as a layer in a neural network for end-to-end training.
## Table 1: 3-pixel and 1-pixel error rates shown for our method compared to [23, 17]. The post processing methods we used were taken from [23].

| Pipeline | KITTI 2015 [14] | FlyingThings3D [17] |
|----------|-----------------|---------------------|
|          | post process 3-pixel | no post p. 3-pixel | post process 3-pixel | no post p. 3-pixel |
| [23] MC-CNN — SGM | 3.78 | 12.73 | 4.74 | 14.45 | 9.66 | 11.96 | 14.01 | 15.88 |
| (ours) MC-CNN — CRF | 5.12 | 12.55 | 6.54 | 15 | 8.85 | 10.98 | 12.1 | 13.88 |
| (ours) MC-CNN — CRF — SGM | 4.7 | 11.74 | 5.32 | 13.03 | 8.85 | 11.02 | 10.88 | 12.64 |
| [17] Mayer et al. | – – | 9.97 | 31.36 | – – | 9.12 | 14.21 |

Figure 2: This figure displays an example of a scene from KITTI 2015 where our method achieves a **lower error rate** than that of [23]. On the left side are the input images and on the right are the computed disparity maps and error maps. Here we demonstrate the advantages of our method. The disparity map is produced with finer detail and without the streaking artifacts which happen when using SGM.
Figure 3: This figure displays an example of a scene from KITTI 2015 where our method achieves a higher error rate than that of [23]. On the left side are the input images and on the right are the computed disparity maps and error maps. It is clearly visible on these examples that we fail mainly on the near objects (large disparity). This is due to lacking examples for near objects in the KITTI dataset.
Figure 4: Examples of disparity prediction with error rates for our method versus [23] on the FlyingThings3D dataset [17]. Here we show that our method indeed beats [23] in many cases and does not suffer from errors in near disparities as on the KITTI dataset. This is due to the large size of the dataset and the more even distribution of disparities throughout the samples.
References

[1] A. Adams, J. Baek, and M. A. Davis. Fast high-dimensional filtering using the permutohedral lattice. In Computer Graphics Forum, volume 29, pages 753–762. Wiley Online Library, 2010. 3, 5

[2] Z. Chen, X. Sun, L. Wang, Y. Yu, and C. Huang. A deep visual correspondence embedding model for stereo matching costs. In The IEEE International Conference on Computer Vision (ICCV), December 2015. 2

[3] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Dec):2121–2159, 2011. 5, 6

[4] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In Conference on Computer Vision and Pattern Recognition (CVPR), 2012. 1

[5] R. Haeusler, R. Nair, and D. Kondermann. Ensemble learning for confidence measures in stereo vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 305–312, 2013. 2

[6] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. IEEE Transactions on pattern analysis and machine intelligence, 30(2):328–341, 2008. 3

[7] D. Kong and H. Tao. A method for learning matching errors for stereo computation. In BMVC, volume 1, page 2, 2004. 2

[8] D. Kong and H. Tao. Stereo matching via learning multiple experts behaviors. In BMVC, volume 1, page 2, 2006. 2

[9] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In Advances in Neural Information Processing Systems, pages 109–117, 2011. 2, 3, 4

[10] D. Levi, N. Garnett, E. Fetaya, and I. Herzylia. StixelNet: A Deep Convolutional Network for Obstacle Detection and Road Segmentation. BMVC, 2015. 5

[11] Y. Li and D. P. Huttenlocher. Learning for stereo vision using the structured support vector machine. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1–8. IEEE, 2008. 2

[12] W. Luo, A. G. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016. 4

[13] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang. 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. 2

[14] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In Conference on Computer Vision and Pattern Recognition (CVPR), 2015. 1, 4, 5, 7

[15] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. In ISPRS Workshop on Image Sequence Analysis (ISA), 2015. 5

[16] J. A. Nelder and R. Mead. A simplex method for function minimization. The computer journal, 7(4):308–313, 1965. 4, 5

[17] N. Mayer, E. Ilg, P. Häsßer, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2016. arXiv:1512.02134. 1, 2, 4, 5, 6, 7, 9

[18] M. Peris, S. Martull, A. Maki, Y. Ohkawa, and K. Fukui. Towards a simulation driven stereo vision system. In Pattern Recognition (ICPR), 2012 21st International Conference on, pages 1038–1042. IEEE, 2012. 2

[19] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In 2007 IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8. IEEE, 2007. 2

[20] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International journal of computer vision, 47(1-3):7–42, 2002. 1, 2

[21] A. Spyropoulos, N. Komodakis, and P. Mordohai. Learning to detect ground control points for improving the accuracy of stereo matching. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1621–1628, 2014. 2

[22] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4353–4361, 2015. 2

[23] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1592–1599, 2015. 1, 2, 3, 5, 6, 7, 8, 9

[24] L. Zhang and S. 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. 2

[25] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. In The IEEE International Conference on Computer Vision (ICCV), December 2015. 1, 3, 4, 5

[26] D. Levi, N. Garnett, E. Fetaya, and I. Herzylia. StixelNet: A Deep Convolutional Network for Obstacle Detection and Road Segmentation. BMVC, 2015. 5

[27] Y. Li and D. P. Huttenlocher. Learning for stereo vision using the structured support vector machine. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1–8. IEEE, 2008. 2

[28] W. Luo, A. G. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016. 4

[29] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang. 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. 2

[30] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In Conference on Computer Vision and Pattern Recognition (CVPR), 2015. 1, 4, 5, 7

[31] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. In ISPRS Workshop on Image Sequence Analysis (ISA), 2015. 5

[32] J. A. Nelder and R. Mead. A simplex method for function minimization. The computer journal, 7(4):308–313, 1965. 4, 5