Fast Deep Stereo with 2D Convolutional Processing of Cost Signatures

Kyle Yee
Amazon
kyle.g.yee@gmail.com

Ayan Chakrabarti
Washington University in St. Louis
ayan@wustl.edu

Abstract

Modern neural network-based algorithms are able to produce highly accurate depth estimates from stereo image pairs, nearly matching the reliability of measurements from more expensive depth sensors. However, this accuracy comes with a higher computational cost since these methods use network architectures designed to compute and process matching scores across all candidate matches at all locations, with floating point computations repeated across a match volume with dimensions corresponding to both space and disparity. This leads to longer running times to process each image pair, making them impractical for real-time use in robots and autonomous vehicles. We propose a new stereo algorithm that employs a significantly more efficient network architecture. Our method builds an initial match cost volume using traditional matching costs that are fast to compute, and trains a network to estimate disparity from this volume. Crucially, our network only employs per-pixel and two-dimensional convolution operations: to summarize the local match information at each location as a low-dimensional feature vector, and to spatially process these "cost-signature" features to produce a dense disparity map. Experimental results on KITTI show that our method delivers competitive accuracy at significantly higher speeds—running at 48 frames per second on a modern GPU.

1. Introduction

The availability of real and synthetic datasets [5, 14, 15] and use of deep neural networks [3, 11, 20, 22] has made stereo estimation increasingly reliable. As evidenced by their performance on realistic benchmarks [15], modern algorithms are able to produce depth estimates from stereo image pairs with reliability that nearly matches depth measurements from more expensive devices such as LIDARs. However, a significant roadblock to practically using these algorithms for depth perception in robots and autonomous vehicles is their computational expense. While traditional stereo methods were able to generate dense depth estimates in real time, albeit with lower accuracy, modern neural network-based stereo methods take more than half a second (often much more) to process a single stereo pair at a standard resolution.

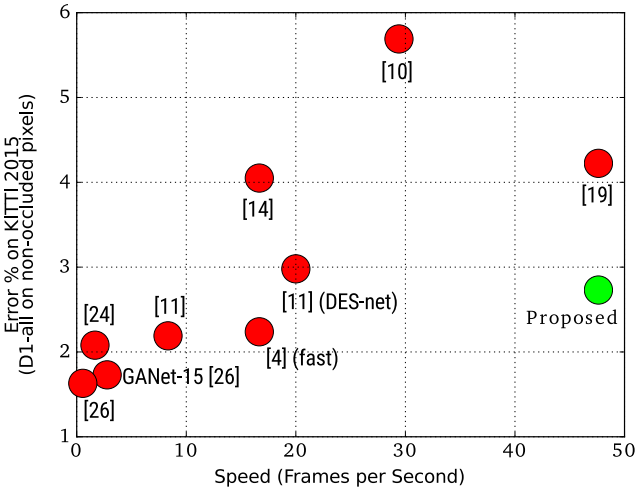

Figure 1. Accuracy-Speed Trade-off in Stereo Estimation. We show the error rate vs speed of different stereo algorithms on the KITTI 2015 [15] benchmark. Our method yields accuracy competitive with state-of-the-art networks for stereo estimation while being significantly faster, and therefore practical for real systems.

The processing pipeline of a stereo algorithm has two computational components: computing a matching cost volume based on similarities between all pairs of reference and matching candidate points in the stereo pair, and processing this volume to yield robust depth estimates by reasoning about smoothness, planarity, etc. in natural scenes. Since the cost volume itself is large (of size equal to number of pixels times the number of candidate disparity values), traditional stereo algorithms emphasized efficient operations for both constructing [25] and processing [6] this volume.

However, neural network-based methods must instantiate these computations using cascades of layers and hence incur significant expense due to the large number of floating point operations repeated across the three-dimensional cost volume. Recent methods cast these as three-dimensional (3D) convolutions [3, 11, 20] to improve parallelism and data flow, but they are still much slower than traditional

stereo methods, and indeed, than typical neural networks that use only two-dimensional (2D) convolutional layers to process regular images.

In this paper, we propose a new neural network-based method for accurate stereo estimation with an architecture highly optimized for computational efficiency. Our approach is motivated by the success of methods for depth completion [13] that are able to reconstruct highly accurate depth maps given a small number of sparse depth measurements and a reference color image as guide. These methods demonstrate that even weak or noisy depth cues can be sufficient to estimate depth accurately, when using a neural network that also learns to exploit monocular depth information present in a color image of a natural scene. This in turn suggests that precise matching with expensive learned costs may not be entirely necessary, and motivates the use of traditional matching costs that are somewhat less powerful but significantly more efficient computationally.

Our method first constructs an initial cost volume using multiple traditional matching costs that are efficient to compute. It then converts the set of all scores for different disparities at each pixel to a low-dimensional "cost-signature" feature vector. This conversion is learned as a set of independent per-pixel layers that produce a succint summary of the stereo depth information at each pixel location. An encoder-decoder network then uses 2D convolutions to process this 2D feature map, rather than the more expensive 3D convolutions on a cost volume, and produces an estimate of the final disparity map. The conversion and encoder-decoder layers are learned jointly with end-to-end training.

Experimental results on the standard KITTI [15] benchmark demonstrate that our models produces estimates with accuracy that is only slightly worse than the state-of-the-art, and higher than traditional stereo methods. But crucially, this accuracy comes at very low computational cost: our network is able to process stereo pairs at *48 frames per second* on a modern GPU. As shown in Fig. 1, our approach affords a favorable trade-off between accuracy and speed, making it both reliable and practical for deployment in robots and autonomous vehicles.

2. Background and Related Work

Depth estimation using stereo images from a calibrated camera pair requires establishing correspondences between pixels in the two images by searching over epipolar lines. When the cameras are related by only horizontal translation (or the images have been rectified to simulate this setup), all epipolar lines are horizontal and the problem reduces to finding the horizontal shift, or *disparity*, between the $x-$ co-ordinates of the projected location of a surface point in the image pair. Stereo estimation is typically cast as the problem of estimating a dense disparity map—the value of disparity at every location in the co-ordinate system of one of the two images chosen as reference. Scene depth can then be derived from disparity given knowledge of relative camera poses.

Estimating disparity by finding dense correspondences is challenging due to the presence of smooth regions, repeating textures, specular highlights, and half-occlusions. Stereo algorithms proceed by first computing an initial score of match quality between each pixel in the reference pixels and all candidate matches. These candidates are indexed by a finite discrete set of candidate disparity values common to all pixels—typically integer pixel disparities ranging from zero to some maximum value—and correspondingly, the matching scores are organized in a cost "volume" along the spatial and disparity dimensions. Traditionally, stereo algorithms used hand-crafted similarity metrics for matching that take into account local neighborhoods around pixels for robustness, while also being efficient to compute [25].

However, these matching scores are still ambiguous and thus local reasoning alone is insufficient for accurate disparity estimation. This is why stereo algorithms have a second "globalization" stage, where the local match information in the cost-volumes is aggregated while promoting properties such as smoothness, piece-wise planarity, etc. in the estimated disparity maps. This aggregation was traditionally as optimization of an energy function [2, 6, 23], again with an emphasis on computational efficiency.

Žbontar and LeCun [21, 22] demonstrated that using deep neural networks for stereo estimation could deliver significant improvements in accuracy over traditional stereo pipelines. Their work only replaced the local matching stage—they proposed learning networks that took a pair of 9×9 patches in the left and right image as input to produce a matching score. Once this network was trained, it was applied on all candidate match pairs to populate the cost volume, which was then smoothed using traditional aggregation techniques [6]. Surprisingly, by just replacing the matching cost with a learned metric, this method was able to achieve significant gains in accuracy. However, these gains came with a reduction in speed, taking more than a minute to process a single stereo pair. Žbontar and LeCun [22] also considered faster architectures, as did Luo *et al.* [12], but these were less accurate and still took 0.7 [12] and 0.8 [22] seconds to process a stereo pair.

These methods were driven by the presence of moderate-sized real stereo datasets [5, 15] with ground-truth data captured using a LIDAR. Noting the benefits of learned methods for stereo, Mayer *et al.* [14] introduced a much larger, synthetically rendered, dataset to enable training of more complex networks—with layers that carry out both matching and globalization computations (the latter replacing traditional aggregation) and are trained end-to-end. GC-Net [8] uses shared 2D convolutions to extract features from

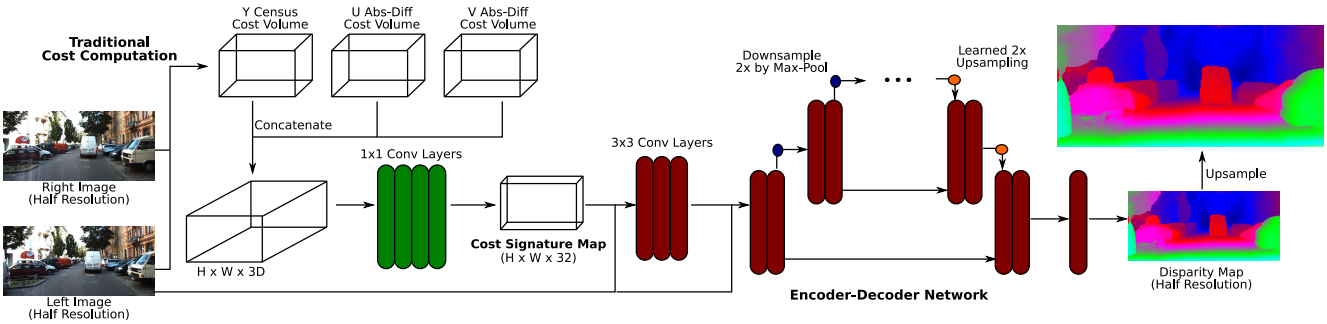

Figure 2. Proposed Stereo Estimation Pipeline. We propose a new computationally efficient architecture that avoids the use of learned matching costs and expensive 3D convolutions. We begin by constructing cost volumes at half-resolution by using different efficient traditional matching costs. We then generate a 2D cost-signature map using per-pixel layers to summarize the different costs at all disparities into a low-dimensional feature vector at each location. This is followed by spatial processing with an encoder-decoder architecture to produce an estimate of the final disparity map. Both the cost-signature and spatial processing layers are trained end-to-end.

each image in the stereo pair, concatenates them to form a 4D tensor (indexed by spatial dimensions, disparity, and features) and then uses 3D convolution layers to process this cost volume. Since then, a number of methods have used a similar approach to using 3D convolutions with innovations in network architecture for cost-volume construction and processing [3, 11, 18, 20], yielding improvements in accuracy and run-times. The fastest among these is the smaller DES-net architecture in [11], which primarily allocates layers for accurate cost computation and achieves a run-time of 0.05 seconds per stereo pair.

The goal of our work is accurate but real-time stereo estimation, which we achieve through the use of traditional matching costs and 2D (instead of 3D) convolution layers. In this context, it is useful to discuss the work of Kuzmin *et al.* [10], who also use traditional matching costs as well as a largely traditional pipeline for aggregation, using a learned deep network to control the parameters of this aggregation in different regions. This allows them to achieve a low run-time of 0.034 seconds (i.e., 29.4 frames per second) but with significantly lower accuracy than state-of-the-art methods. Meanwhile, Park *et al.* [16] use disparity maps computed entirely using traditional methods as input to a neural network for refinement—but crucially, these are combined with disparity data derived from LiDAR measurements. Their method can be seen as an extension to [13]'s approach of processing sparse and noisy depth measurements, in this case, also including information derived from simple stereo matching. However, when working with stereo data alone, disparity maps derived directly from traditional cost volumes are often too noisy, and introduce an information bottleneck by discarding distributional information about the relative costs of different disparity values. This is why the input to our spatial processing network are cost signatures, that are trained to optimally summarize this distributional information.

Other examples of fast stereo methods include the DispNetC architecture introduced in [14] that uses 2D convolutions, like us, for spatial reasoning—applied on a feature map derived from computing cross-correlations between per-image feature maps at different disparity shifts. This also leads to reduced run-times (0.06 seconds in their case) but lower accuracy. Our method is able to achieve higher accuracy as well as lower run-times (0.021 seconds) than both these methods. Very recently, Tonioni *et al.* [19] and Duggal *et al.* [4] proposed network architectures with the goal of maintaining accuracy while achieving real-time, or near real-time performance. Among these, [19] achieves a run-time similar to ours but is unable to maintain as high a level of accuracy. On the other hand, the fastest version of [4] is somewhat more accurate than ours, but is roughly three times slower. It is worth noting that their approach to speeding up computation is different from ours—since they still rely on learned features for matching—and it is likely that their architecture design innovations could be combined with our approach to achieve further speed-ups or higher accuracy at the same speed.

3. Proposed Method

We now describe our processing pipeline and network architecture for stereo estimation, summarized also in Fig. 2. Our pipeline follows the standard stereo framework of local matching followed by globalization, but with the goal of maximizing computational efficiency, we use traditional matching costs and a learned network for globalization. As our experiments will show, this allows us to produce high-quality disparity maps with high computational efficiency.

We leverage multiple traditional efficient matching costs to build initial cost volumes to represent local belief about disparity, summarize this information through a learned pixel-wise mapping as a low-dimensional 2D feature map, and finally use 2D convolutions within an encoder-decoder architecture to yield the final disparity estimates. This

allows our network to avoid repeating expensive floating point computations across different disparity candidates (i.e., 3D convolutions across a cost volume). All our computations are also carried out at half resolution (as is common) for further efficiency, with a final interpolation step in the end to yield a disparity map at the original resolution.

3.1. Initial Matching with Traditional Costs

The input to our network are the left and right images, downsampled by a factor of two. The first step in a typical stereo pipeline is to use local evidence to build a cost-volume, or a belief distribution over different possible values of disparity for each pixel in the reference left image. Most neural network-based methods attempt to learn a matching cost to build such a cost-volume. However, this proves to be much more expensive than traditional matching costs [25], since the cost functions must be approximated as a sequence of layers, with floating point operations repeated for all pairs of reference pixels and candidate disparities.

Instead, we use multiple traditional matching costs to construct a set of cost volumes $\{C_i\}$, one for each kind of cost, from the input images. Each volume is of size $H \times W \times D$ where H, W are the height and width of the (downsampled) image, and D is the number of possible disparity values (we consider integer disparities going from 0 to 127 pixels at the half resolution). Each element $C_i(x, y, d)$ of each volume measures the dis-similarity between pixels or regions at (x, y) in the left image and $(x - d, y)$ in the right image (when $x - d < 0$, we fill in values from the first valid pixel for d in the same row y).

To apply different matching costs for spatial and color information, we convert the input images from RGB to YUV color space. The first volume C_1 is then computed by comparing spatial variations in the Y (i.e., luminance) channels of the left and right images, as hamming distances between 5×5 census transforms [25]. The census cost found common use in traditional stereo algorithms due to its robustness to global intensity scaling, and its computational efficiency—it requires computing a census code once for each location in the left and right images based on local comparisons, following which all entries of the cost volume can be computed by computing the hamming distance between corresponding codes (which only requires counting 1s in the bit-wise XOR of the two codes).

To exploit color information, the other two cost volumes C_2 and C_3 are populated with the absolute difference of the U and V values of the corresponding pixels in the left and right image. We normalize each volume separately so that their costs have zero mean and unit-variance (as determined across volumes created from images in a training set).

3.2. Mapping to Low-dimensional Cost Signatures

The cost volumes $\{C_i\}$ summarize the depth information present in correspondences between the left and right image, and allow us to carry out all remaining remaining computations in our pipeline in the reference co-ordinate frame of the left image. However, the cost-volumes are still high-dimensional, with $3D$ numbers at each pixel.

Therefore, the next stage of our pipeline converts the set of different matching costs at all candidate disparities for each reference pixel (x, y) into a more tractable low-dimensional cost signature $S(x, y) \in \mathbb{R}^{32}$. We begin by concatenating the three cost volumes into one 3D tensor of size $H \times W \times (3D)$. This is treated as a 2D feature map $A_0(x, y) \in \mathbb{R}^{3D}$, where each $A_0(x, y)$ contains a vector of all costs at all disparities:

$$A_0(x, y) = [C_1(x, y, 0), \ldots C_1(x, y, D-1), C_2(x, y, 0), \ldots], \quad (1)$$

and learn a mapping from A_0 to the lower-dimensional S. This is different from other neural network-based approaches [3, 8, 11, 18, 20] that construct 4D tensors corresponding to 3D feature maps, to enable operations between neighboring disparities. We also do not combine the different costs with a weighted sum as in traditional stereo methods [2, 23], and let the cost-signature $S(x, y)$ be a more general functions of different costs at different disparities.

The dimensionality reduction from $3D$ (which is 384 in our setting of three costs and 128 candidate disparities) in A_0 to 32 in S is carried out by four layers that operate independently on each pixel location, and reduce dimensions to 192,96,48, and finally 32 feature channels. We instantiate these as 1×1 convolution layers, and use batch normalization [7] and ReLU activations at the output of each layer.

3.3. Globalization with CNN on Cost Signatures

The final stage of our architecture mimics the globalization step of a traditional stereo pipeline, which is typically based on performing spatial processing based on local disparity distributions as well as gradient information from the reference image. In our architecture, we use a network with only 2D convolution layers to estimate a disparity map from the cost-signature feature map S. We begin by concatenating the left input image (at half resolution) to the cost-signature map, and sending these through an initial set of three convolution layers—all with 3×3 kernels, 32 channel outputs, and batch normalization and ReLU activations. We then take this output, concatenate it again with the left image, and feed it to a UNet [17]-like encoder-decoder architecture.

This encoder-decoder network features five levels of downsampling in the encoder and upsampling in the decoder (each time by a factor of 2), with skip-connections (joined by concatenation) between corresponding scales of

the encoder to the decoder. We use two 3×3 convolution layers each in the decoder and encoder at each scale, and achieve downsampling by a 2×2 max-pool operation with stride 2, and upsampling by a learned convolution layer. While the original UNet [17] recommends doubling the number of feature channels at each scale, we choose to only increase the channels by 16 each time to reduce computation. We do not use batch-normalization in these layers.

The output of the final decoder layer is a 32-channel feature map at half-resolution. We use a single per-pixel layer to map this to a single channel disparity map, and then upsample this map to the full resolution. During training, we simply use nearest neighbor upsampling, while at test time we use a simple discontinuity-aware interpolation scheme: we produce both nearest-neighbor and bilinearly interpolated disparity maps, and at each pixel select the interpolated version when the difference between the two is less than 1 pixel, and the nearest-neighbor version otherwise.

3.4. Training Loss

Note that the first stage of our pipeline (Sec. 3.1) is fixed and need not be learned. We train the cost-signature (Sec. 3.2) and spatial processing (Sec. 3.3) layers end-to-end based on a loss defined on the quality of the final full-resolution disparity map (i.e., after upsampling). We use a robust regression loss, between estimated and ground-truth disparities $\hat{d}$ and d_{GT}:

$$L(d_{GT}, \hat{d}) = \max(\tau, d_{GT} - \hat{d})^{1/8}. \quad (2)$$

The sub-linear exponent makes the loss robust to outliers (i.e., where the disparity error is too high), with clipping by τ used to ensure that gradients of the loss are stable (we set $\tau = 1$ in our experiments). Note that the loss function is only computed over pixels with valid disparity values present in the ground truth.

4. Experiments

We implement our network architecture in Tensorflow [1], using custom GPU operations for the initial cost volume computation in Sec. 3.1, and evaluate our method on the KITTI 2012 [5] and 2015 [15] benchmarks. We report running times using an NVIDIA GTX 1080Ti GPU. For training, we adopt the standard practice of pre-training our network on the synthetic dataset of [14]—specifically, on images from the "FlyingThings3D" and "Driving" sub-sets. We then fine-tune on images from the KITTI 2012 and 2015 training sets—although, we remove a subset of 20 images from the KITTI 2015 training set and use it for validation. Our reference implementation along with trained model weights is available at `https://projects.ayanc.org/fdscs/`.

Step	Time
Traditional Cost Computation	0.0067s
Conversion to Cost Signatures	0.0087s
Spatial Processing	0.0057s
Total	0.021s

Table 1. Running Time breakdown.

Model	Avg Err.	$>$ 3px	Time
Full Model	0.72	2.41	0.021s
Only Census Cost	0.75	2.60	0.016s
Only 3-Level Enc-Dec	0.90	3.65	0.020s

Table 2. Ablation Study on Validation Set.

4.1. Training

We train our network with the loss in (2) and weight decay of 10^{-5} using the Adam optimizer [9]. We begin by training for 350k iterations on the synthetic dataset [14], with a learning rate of 10^{-4} (after initially training for 5k iterations at a lower rate of 10^{-5} for stability). Although our network is designed to only produce disparities with respect to the co-ordinates of the left image, this dataset provides dense ground-truth disparity maps with respect to both the left and right images. To train also with the right image disparity map, we form an additional pair by swapping the left and right images and flipping both images and the disparity map horizontally. We train with a batch size of four original pairs (which yields eight total pairs per batch).

We then fine-tune on images from KITTI, and since this is a smaller dataset with sparse ground-truth data, we use scale augmentation: scaling the left, right, and ground truth images by a random scale factor in $(1.0, 1.5)$, and dividing the ground truth disparities by the same value. We use a batch size of four, and train for 150k iterations with a learning rate of 10^{-4}, and additional 50k iterations at learning rates of 10^{-5} and 10^{-6} each.

4.2. Run-time Analysis and Ablation

We begin by analyzing the running time of our network on typical KITTI images (with size 1240x375) in Table 1, measuring the time taken in different steps of our pipeline. We find that the biggest contributor to running time is in fact in converting the traditional cost volume to a low-dimensional per-location cost-signature—even though we only use 1x1 convolutions here, this part of our pipeline still has the largest number of floating point operations since it takes the entire cost volume as input. Traditional cost computation comes next in the amount of computation, but this is still much faster than with learned matching costs

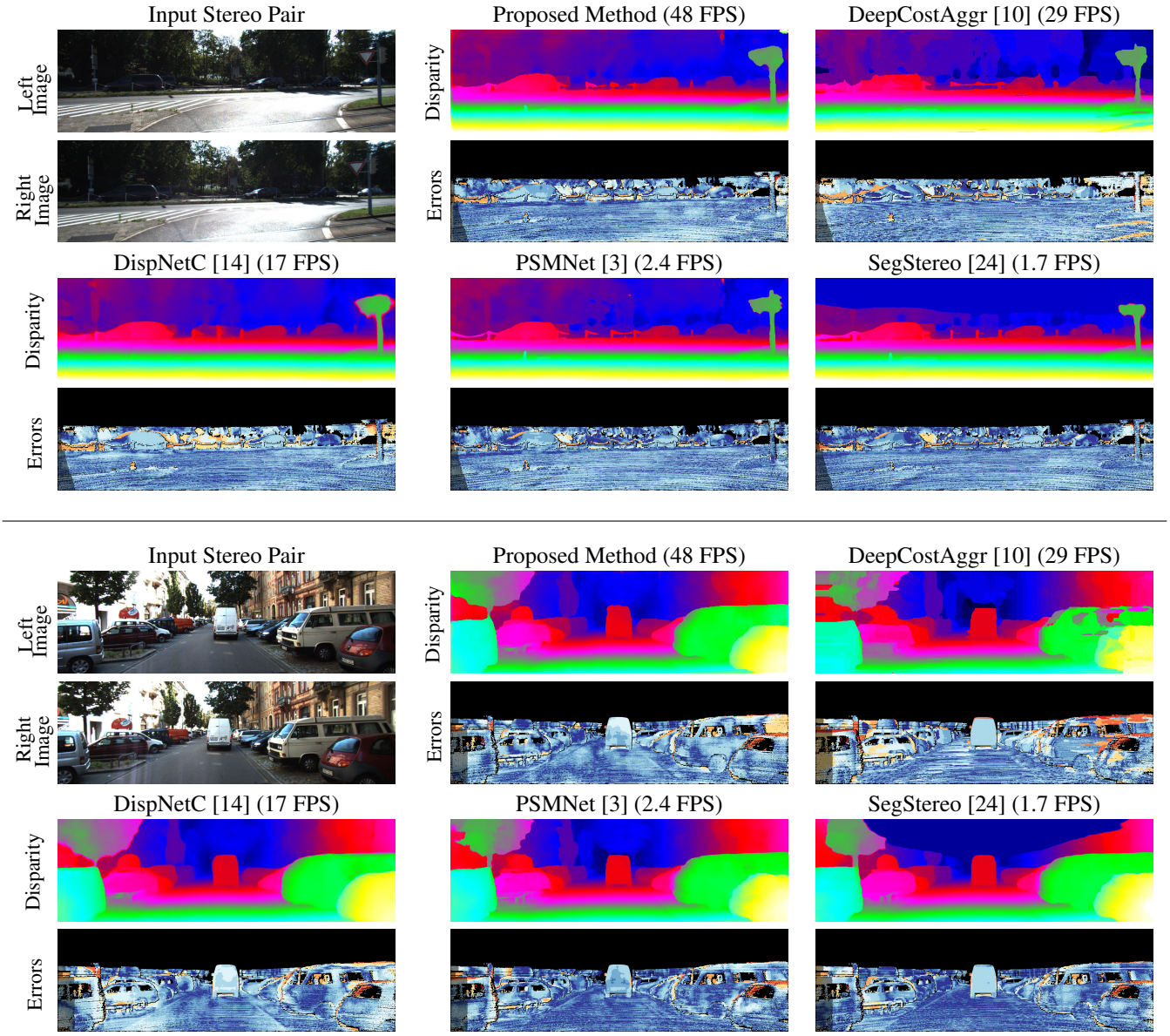

Figure 3. Example Results on KITTI 2015 [15] test images. We show disparity maps, estimated by our model as well other methods, and corresponding errors on example stereo pairs from the KITTI 2015 test set. Disparity and error maps are shown using the standard color scheme of the benchmark.

in other methods. The last part of our pipeline—spatial processing—is the least expensive computationally.

Then, we compare our full approach on the validation set to different ablated versions in Table 2. Specifically, we consider a version of our model that only uses the census cost volume and leaves out the chromaticity difference-based costs, and a versions with a smaller networks for spatial processing with only three (instead of five) scales in the encoder-decoder. We report running times (for 1240x375 images) for these versions and accuracy in terms of the average error (absolute difference between true and estimated disparity) as well as percentage of pixels where this error is greater than 3 pixels.

We see that both variations from our full model lead to higher errors and lower running times, but by different amounts. In particular, removing the color matching costs leads to a significant improvement in speed (from 48 to 62 frames per second) but only a modest drop in performance. In contrast, using a smaller spatial processing network leads to a barely measurable improvement in speed but a significant drop in performance. This demonstrates that incorporating more match information (with more costs) provides

Method	All Pixels			Non-Occluded Pixels			Run
	D1-bg	D1-fg	D1-all	D1-bg	D1-fg	D1-all	Time
MC-CNN-acrt [22]	2.89	8.88	3.89	2.48	7.64	3.33	67s
GANet-Deep [26]	1.48	3.46	1.81	1.34	3.11	1.63	1.8s
GC-Net [8]	2.21	6.16	2.87	2.02	5.58	2.61	0.9s
Content-CNN [12]	3.73	8.58	4.54	3.32	7.44	4.00	0.7s
SegStereo [24]	1.88	4.07	2.25	1.76	3.70	2.08	0.6s
PDSNet [20]	2.29	4.05	2.58	2.09	3.68	2.36	0.5s
PSMNet [3]	1.86	4.62	2.32	1.71	4.31	2.14	0.41s
GANet-15 [26]	1.55	3.82	1.93	1.40	3.37	1.73	0.36s
EdgeStereo [18]	2.27	4.18	2.59	2.12	3.85	2.40	0.27s
DeepPruner (best) [4]	1.87	3.56	2.15	1.71	3.18	1.95	0.18
iResNet-i2 [11]	2.25	3.40	2.44	2.07	2.76	2.19	0.12s
DispNetC [14]	4.32	4.41	4.34	4.11	3.72	4.05	0.06s
DeepPruner (fast) [4]	2.32	3.91	2.59	2.13	3.43	2.35	0.06
DES-net [11]	3.13	3.87	3.25	2.94	3.21	2.98	0.05s
DeepCostAggr [10]	5.34	11.35	6.34	4.82	10.11	5.69	0.034s
MADnet [19]	3.75	9.20	4.66	3.45	8.41	4.27	0.02
SPS-St [23]	3.84	12.67	5.31	3.50	11.61	4.84	(CPU) 2s
Proposed	2.83	4.31	3.08	2.53	3.74	2.73	0.021s

Table 3. Results on the KITTI 2015 [15] Benchmark

Method	$>$ 2px		$>$ 3px		$>$ 4px		$>$ 5px		Run
	Out-Noc	Out-All	Out-Noc	Out-All	Out-Noc	Out-All	Out-Noc	Out-All	Time
MC-CNN-acrt [22]	3.90	5.45	2.43	3.63	1.90	2.85	1.64	2.39	67s
GANet-Deep [26]	1.89	2.50	1.19	1.60	0.91	1.23	0.76	1.02	1.8s
GC-Net [8]	2.71	3.46	1.77	2.30	1.36	1.77	1.12	1.46	0.9s
Content-CNN [12]	4.98	6.51	3.07	4.29	2.39	3.36	2.03	2.82	0.7s
SegStereo [24]	2.66	3.19	1.68	2.03	1.25	1.52	1.00	1.21	0.6s
PDSNet [20]	3.82	4.65	1.92	2.53	1.38	1.85	1.12	1.51	0.5s
PSMNet [3]	2.44	3.01	1.49	1.89	1.12	1.42	0.90	1.15	0.41s
GANet-15 [26]	2.18	2.79	1.36	1.80	1.03	1.37	0.83	1.10	0.36s
EdgeStereo [18]	-	-	1.73	2.18	1.30	1.64	1.04	1.32	0.27s
iResNet-i2 [11]	2.69	3.34	1.71	2.16	1.30	1.63	1.06	1.32	0.12s
DispNetC [14]	7.38	8.11	4.11	4.65	2.77	3.20	2.05	2.39	0.06s
DES-net [11]	4.88	5.54	2.66	3.12	1.78	2.11	1.33	1.59	0.05s
SPS-St [23]	4.98	6.28	3.39	4.41	2.72	3.52	2.33	3.00	(CPU) 2s
Proposed	4.54	5.34	2.61	3.20	1.86	2.33	1.46	1.85	0.021s

Table 4. Results on the KITTI 2012 [5] Benchmark

comparatively less value to more complex spatial processing (with a larger network and receptive field). At the same time, this spatial processing is relatively cheap since it involves only 2D convolutional operations, while acquiring

more match information incurs a higher computational cost because it adds to the number of operations that must be repeated across the disparity dimension.

4.3. Results on KITTI Benchmark

Next, we report the official results as returned by the test server on the KITTI 2015 and 2012 benchmarks along with running time in Tables 3 and 4 respectively, and compare these to a number of methods: including state-of-the-art methods with high accuracy [3, 24, 26], as well as those with relatively low running times (such as [10, 14, 19]). For KITTI 2015, all errors correspond to percentage of pixels with errors greater than 3 pixels for non-occluded and all pixels, reported separately for all pixels (D1-all) and those corresponding to background (D1-bg) and foreground (D1-fg) objects. For the KITTI 2012 benchmark, errors are measured as percentage of pixels with disparity error above different thresholds, computed over non-occluded (Out-Noc) and all (Out-all) pixels. We also include example estimated disparities and corresponding errors from the KITTI 2015 test set in Fig. 3.

We find that our method is both faster and has a clear advantage in accuracy over the only other method with real-time performance [10]. On KITTI 2015, it also performs better than the other methods that take less than 0.05 seconds per stereo pair: [19] and the DES-net version of [11], although it does have slightly lower accuracy than DES-net on KITTI 2012. At the same time, the proposed method's performance is competitive to state-of-the-art methods [3, 24]: e.g., on the D1-all metric on non-occluded pixels in KITTI 2015, it is worse by only 1% and 0.59% compared to GANet-15 and PSMNet, while being between 17 and 20 times faster. Therefore, our method provides a new trade-off in accuracy vs speed in stereo estimation. It allows practical real-time usage while yielding estimates with competitive depth accuracy.

4.4. Results on SceneFlow

For completeness, we also report results on the SceneFlow [14] test set. For this, we use a model without finetuning on KITTI. Instead, we continue training the model for another 100k iterations on a subset of our training set without any images from the "driving" images from [14], the last 50k of which are at a lower learning of 10^{-5}. The accuracy in terms of End-Point-Error (EPE) is reported in Table 5. Note that while still competitive, we find that the method performs slightly worse relatively on this synthetic dataset. We believe this to be likely due to the poorer performance of the Census transform as a cost measure on the relatively smoother textures.

Method	Avg. EPE
DeepPruner (best) [4]	0.86
DeepPruner (fast) [4]	0.97
PSMNet [3]	1.09
PDSNet [20]	1.12
SegStereo [24]	1.45
DispNetC [14]	1.68
GC-Net [8]	2.51
Proposed	2.01

Table 5. Evaluation on SceneFlow Test Set [14].

5. Conclusion

We introduced a new stereo estimation method that is able to generate accurate dense depth estimates from stereo image pairs at faster than real time speeds—48 frames per second—on a modern GPU. Our method achieves this by using traditional matching costs instead of their more expensive learned counterparts, and focusing its computations on spatial processing with 2D convolutions, in contrast to recent neural network-based methods that seek to explicitly mimic the cost-volume computations of traditional stereo pipelines. Given its accuracy and speed, our method is feasible to deploy on actual robots and autonomous vehicles, possibly as an alternative for depth perception to more expensive LIDARs. While our work focused on the binocular stereo case in this paper, in future work we propose extending our approach to multi-view stereo—we believe the higher computational efficiency will make it possible to reason about correspondences across multiple cameras in real time, while bringing gains in depth estimation accuracy.

Acknowledgments

This work was supported by the NSF under award no. IIS-1820693, including with an REU supplement for KY's participation in the project at Washington University.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[2] A. Chakrabarti, Y. Xiong, S. J. Gortler, and T. Zickler. Low-level vision by consensus in a spatial hierarchy of regions. In *Proc CVPR*, 2015.

[3] J.-R. Chang and Y.-S. Chen. Pyramid stereo matching network. In *Proc. CVPR*, 2018.

[4] S. Duggal, S. Wang, W.-C. Ma, R. Hu, and R. Urtasun. Deeppruner: Learning efficient stereo matching via differentiable patchmatch. In *Proc. ICCV*, 2019.

[5] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. CVPR*, 2012.

[6] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Trans. PAMI*, 2008.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[8] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proc. ICCV*, 2017.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

[10] A. Kuzmin, D. Mikushin, and V. Lempitsky. End-to-end learning of cost-volume aggregation for real-time dense stereo. In *IEEE Intl. Workshop on Machine Learning for Signal Processing*, 2017.

[11] Z. Liang, Y. Feng, Y. Guo, H. Liu, W. Chen, L. Qiao, L. Zhou, and J. Zhang. Learning for disparity estimation through feature constancy. In *Proc. CVPR*, 2018.

[12] W. Luo, A. G. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. In *Proc. CVPR*, 2016.

[13] F. Mal and S. Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *Proc. ICRA*, 2018.

[14] N. Mayer, E. Ilg, P. Hausser, 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 *Proc. CVPR*, 2016.

[15] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Proc. CVPR*, 2015.

[16] K. Park, S. Kim, and K. Sohn. High-precision depth estimation with the 3d lidar and stereo fusion. In *Proc. ICRA*, 2018.

[17] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. Intl. Conf. on Medical Image Computing and Computer-assisted Intervention*, 2015.

[18] X. Song, X. Zhao, H. Hu, and L. Fang. Edgestereo: A context integrated residual pyramid network for stereo matching. In *Proc. ACCV*, 2018.

[19] A. Tonioni, F. Tosi, M. Poggi, S. Mattoccia, and L. D. Stefano. Real-time self-adaptive deep stereo. In *Proc. CVPR*, 2019.

[20] S. Tulyakov, A. Ivanov, and F. Fleuret. Practical deep stereo (pds): Toward applications-friendly deep stereo matching. In *Proc. NeurIPS*, 2018.

[21] J. Žbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. In *Proc. CVPR*, 2015.

[22] J. Žbontar, Y. LeCun, et al. Stereo matching by training a convolutional neural network to compare image patches. *JMLR*, 2016.

[23] K. Yamaguchi, D. McAllester, and R. Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *Proc. ECCV*, 2014.

[24] G. Yang, H. Zhao, J. Shi, Z. Deng, and J. Jia. Segstereo: Exploiting semantic information for disparity estimation. In *Proc. ECCV*, 2018.

[25] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proc. ECCV*, 1994.

[26] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. In *Proc. CVPR*, 2019.