RecResNet: A Recurrent Residual CNN Architecture for Disparity Map Enhancement

Konstantinos Batsos
kbatsos@stevens.edu

Philippos Mordohai
mordohai@cs.stevens.edu

Stevens Institute of Technology

Abstract

We present a neural network architecture applied to the problem of refining a dense disparity map generated by a stereo algorithm to which we have no access. Our approach is able to learn which disparity values should be modified and how, from a training set of images, estimated disparity maps and the corresponding ground truth. Its only input at test time is a disparity map and the reference image. Two design characteristics are critical for the success of our network: (i) it is formulated as a recurrent neural network, and (ii) it estimates the output refined disparity map as a combination of residuals computed at multiple scales, that is at different up-sampling and down-sampling rates. The first property allows the network, which we named RecResNet, to progressively improve the disparity map, while the second property allows the corrections to come from different scales of analysis, addressing different types of errors in the current disparity map. We present competitive quantitative and qualitative results on the KITTI 2012 and 2015 benchmarks that surpass the accuracy of previous disparity refinement methods. Our code is available at `https://github.com/kbatsos/RecResNet`

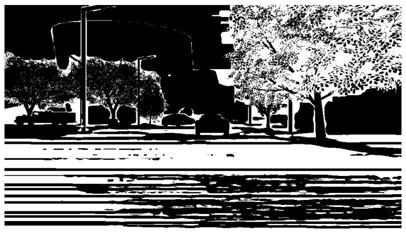
1. Introduction

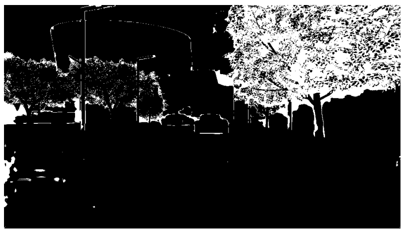
Due to the recent developments in convolutional neural networks (CNNs) and the availability of labeled data, state-of-the-art modern stereo pipelines either integrate deep convolutional neural networks as components [7, 23, 37, 47, 48, 49] or cast the problem as an end-to-end learning task, thus converting the entire stereo pipeline to a deep convolutional architecture [17, 20, 24, 27, 38]. The taxonomy introduced by Scharstein and Szeliski [33], however, still holds. It divides the stereo matching process into four main steps: matching cost computation, cost aggregation, optimization, and disparity refinement. In the context of deep learning, matching cost computation is typically implemented as a Siamese network, cost aggregation can be though of as operating on the images or in the cost volume by adjusting the receptive field of the networks, optimization has been implemented using 3D convolutional operations in the cost volume [17] and disparity refinement can be implemented with a 2D convolutional architecture operating on the disparity map [10, 27, 46].

In this paper, we address the last stage of the pipeline via a novel deep network architecture that is based on the concepts of residual networks and recurrent networks. We argue that both aspects are critical for achieving high accuracy in a label refinement task. The motivation for tackling the final stage of the stereo matching pipeline, as opposed to matching cost computation for example, comes from our observation that post-processing can have dramatic effects on accuracy in conventional stereo algorithms and also that a disparity map can reveal a considerable amount of information about where its errors are. The latter was confirmed by Poggi et al. [31] who showed that cues from the disparity map are more effective for confidence estimation than image features or intermediate results of the matching process. Figure 1 shows an example of disparity refinement by our method on a stereo pair from the Freiburg driving dataset [24].

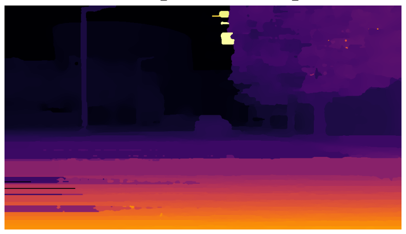
The performance of deep networks is strongly related to the number of their parameters, as long as sufficient training data are available. Training very large networks, however, is demanding in terms of computational and memory requirements. Therefore, a fair and informative comparison between different architectures is to contrast their performance for a given size of the network, imposed by the hardware. One approach is to define a network which is as large as possible and apply it to the problem at hand. Alternative approaches aim to reuse the network, or parts of it, to achieve even better performance; we call this the *iterative approach*. In a problem such as ours, a trained network that improves disparity maps can be applied to its own output. However, the errors in the outputs of the first application of the network do not follow the patterns of the original inputs and this approach is not very effective in practice. A better alternative is to feed the output of a trained network to a second instance of the same network and, then, fine tune the second network

(a) Left view


(b) Input error map


(c) Final error map

(d) Ground truth


(e) Input state Y_{init}

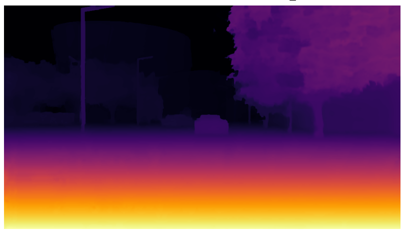

(d) Final state Y_{final}

Figure 1. An example from the synthetic Freiburg driving dataset [24]. 38.5% of the pixels in the initial disparity map have an error over 3 pixels. This is reduced to 16.2% by RecResNet. (Black and white pixels correspond to correct and wrong disparities, respectively.)

keeping the first one frozen; we call this the *cascade training approach*. The DRR algorithm [10] is an example of a cascade. Another alternative is the *recurrent training approach* in which the output of the network is fed to itself. It can be thought of as a cascade where the weights of the core architecture are shared between time steps. A cascade with N stages would have N times more parameters than the recurrent network during deployment. The main drawback of the recurrent network is the way weights and biases are updated during back-propagation; intermediate gradient contributions are stored through time steps and aggregated to update the weights and biases in a single step. Approximations of the recurrent network training formulation without the need of storing intermediate gradient results have been proposed by [44, 21]. However, they require different parts of the network to be trained independently and the training process is rather disjoint than end-to-end. On the contrary, the recurrent training approach is truly end-to-end and has a more holistic view of the task at hand.

We argue that a recurrent architecture is the most appropriate for a dense label refinement problem, such as ours, since it can progressively correct large erroneous regions by being applied recursively on its own output while keeping the number of parameters low.

A novel aspect of our work compared to other disparity refinement approaches relying on deep learning [10, 46] is that it does not assign specific tasks to subnetworks, such as the detect, replace and refine operations of DRR [10]. Instead, we design a high-capacity residual network that predicts its output by a combination of multiple residuals generated at different scales of processing. This allows our approach to correct different types of errors that require different degrees of smoothness. For instance, pixels in uniform flat areas would benefit from a large scale correction, while pixels near surface discontinuities require finer corrections,

Due to the recurrent and residual aspects of our network, we named it **RecResNet**. We evaluated it on the 2012 and 2015 KITTI stereo benchmarks [9, 25] using disparity maps generated by the MC-CNN algorithm of Žbontar and LeCun [49]. RecResNet ranks third and fifth respectively, considering only published results. We envision RecResNet being adopted by other researchers as a component of a system or to refine disparity maps generated by black box algorithms.

In summary, the contributions of this paper are:

- To the best of our knowledge, the first recurrent neural network architecture for disparity map refinement.
- A novel residual network architecture that combines multiple residuals to generate the final output.
- Competitive with state-of-the-art results on the KITTI 2012 and 2015 benchmarks.

2. Related Work

In the past few years, there has been a proliferation of machine learning methods applied to stereo matching. Their success has led to a rapid increase of publications relevant to ours. We focus on supervised methods that require ground truth disparity maps, but note the recent exciting research on unsupervised methods [42, 43, 52]. We classify the methods according to the aspect of the matching pipeline they address, reviewing end-to-end methods, separately.

Cost Computation. Arguably, the first paper that presents a method for learning the cost function was published by Li and Huttenlocher [22]. It uses a structured support vector machine to learn linear discriminant functions that compute the data and smoothness terms of a Conditional Random

Field (CRF) based on discretized values of the matching cost, image gradients and disparity differences among neighboring pixels. Alahari et al. [3] formulated a similar learning problem using the same node and edge features as [22] and convex optimization to obtain the solution more efficiently.

The most impactful recent work in this area is by Žbontar and LeCun [48, 49] who trained a CNN to predict whether two image patches match or not. Two network architectures are proposed: a fast architecture, MC-CNN-fst, which is a Siamese network with two sub-networks consisting of convolutional layers and rectified linear units, and an accurate architecture, MC-CNN-acrt, in which the fast network is followed by a number of fully connected layers. The fast architecture extracts a representation from each patch and measures similarity by taking the inner product of the representations. The accurate architecture generates a similarity score as the output of the last layer. Both architectures are trained using matching and mismatching patches undergoing a multitude of changes in illumination, suffering from miscalibration and other sources of error. The CNN outputs the matching cost volume, which then undergoes a number of processing steps, including SGM optimization, to produce the disparity map.

Inspired by the accurate MC-CNN architecture, which was published first, other authors [7, 23] also proposed efficient formulations that compute patch similarity via an inner product layer operating of their representations. Related research was also carried out by Han et al. [14] and Zagoruyko and Komodakis [47] who investigated network architectures and training strategies for matching image patches. Most top performing conventional methods have either been inspired by MC-CNN or directly use it to compute the matching cost [4, 10, 18, 35, 36, 41, 50].

Park and Lee [28] enabled the network to access wider context by adding a per-pixel pyramid pooling layer that considers data over multiple scales without causing foreground fattening. The high computational cost of this approach was alleviated by a modification proposed by Ye et al. [46]. Shaked and Wolf [37] increase the capacity of the matching residual network by introducing constant highway skip connections that essentially allow matching cost computation along multiple pathways. Zhang and Wah [51] derive new deep matching costs by imposing the principle of distinctiveness in the objective function.

Cost Modulation. Recently, several authors proposed ways to inject the confidence estimates into disparity optimization to generate improved, dense disparity maps [31]. Spyropoulos et al. [39, 40] train a random forest on the cost volume to detect ground control points, which are favored during MRF-based disparity optimization. Park and Yoon [29] use the predictions of a random forest to modulate the data term of each pixel in SGM-based optimization. Poggi and Mattoccia [30] learn a confidence measure that takes into account multi-scale features and is used to weigh cost aggregation in SGM in order to reduce artifacts. Seki and Pollefeys [35] use a CNN trained on patches of the left and right disparity maps of a stereo pair to predict confidence which in turn adjusts the regularization parameters of SGM. The same authors [36] extended the approach by considering the signed disparity differences between neighboring pixels.

Cost Volume Optimization. Shaked and Wolf [37] (see above for cost computation) replace the winner-take-all disparity selection after SGM optimization with a CNN that operates on the optimized cost volume to predict disparity and confidence by combining a reflective loss with a weighted cross-entropy loss. Confidence is used for further post-processing.

Disparity Map Refinement. Poggi et al. [31] conclude that the disparity map itself contains the most valuable information for confidence estimation, and, therefore, for detecting and correcting errors. This observation explains the effectiveness of disparity refinement methods, which treat the initial disparity generation process as a black box.

A disparity refinement approach, named DRR which stands for detect, replace, refine, was recently presented by Gidaris and Komodakis [10]. It decomposes label improvement in a detection, a replacement and a refinement step based on the hypothesis that hard mistakes should be replaced, while soft mistakes can be corrected by refinement. The authors show that further improvements can be achieved if the network is applied in a cascade of two blocks. Ye et al. [46] extend [10] by using the best and second best disparity maps from an improved MC-CNN-style pipeline as inputs and adding more operations, such as two replacement subnetworks, and achieve high accuracy on the Middlebury 2014 benchmark. We take a different approach by not separating the tasks and allowing the branches of the network to learn their roles.

End-to-end Approaches. Mayer et al. [24] introduced the first end-to-end system for estimating disparity maps. They propose DispNet, which is similar to FlowNet [8], and DispNetC which includes an explicit correlation layer that leads to higher matching accuracy. Knöbelreiter et al. [20] were able to train a hybrid CNN-CRF model end-to-end using a subgradient approximation. Unlike related approaches, the CNNs are relatively shallow and the CRF is 4-connected. Slossberg et al. [38] took a similar approach with a fully connected CRF, that makes its integration into an end-to-end system more straightforward, and hand-crafted pairwise potentials.

Kendall et al. [17] developed GC-Net, an end-to-end pipeline that includes 3D convolutional layers that regress disparity from a cost volume generated by residual blocks that extract patch representations from the images. The design comprises differentiable components that correspond to the steps of conventional stereo pipelines, takes into account

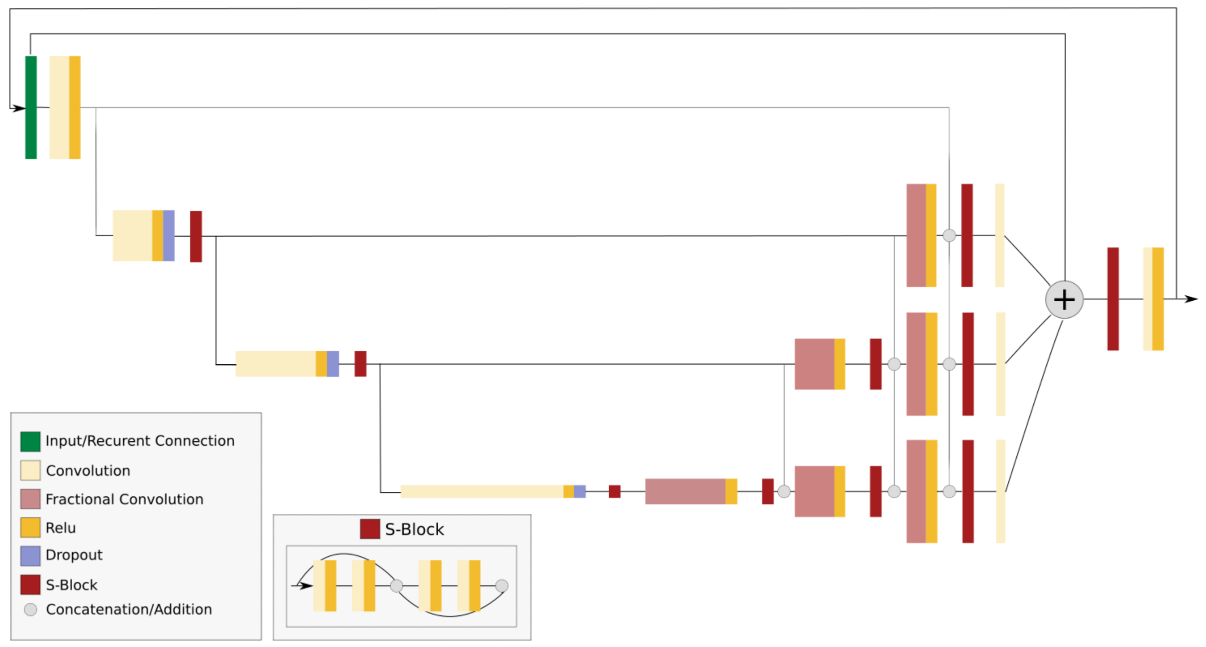


Figure 2. An abstract representation of RecResNet.

context due to the 3D convolutions, and achieves state-of-the-art accuracy on the KITTI benchmarks.

Pang et al. [27] proposed a cascade residual learning approach that comprises two cascaded networks that are trained end-to-end. The first network is similar to DispNet [24] but outputs full-resolution disparity maps. The second network, which is more relevant to our work, refines these disparity maps via residual learning with explicit supervision at each scale of the residual network. As in [10], the authors argue that residual learning is an easier problem than estimating disparity for certain pixels directly.

3. Method

Our focus is on dense labeling correction. Given an initial state of labels Y_{init} and optionally the inputs X to the label generator function G, we wish to learn a function F that can be used to predict more accurate labels. In this paper, X is the left image of a stereo pair, Y is the disparity map and G is a stereo matching algorithm that we treat as a black box. Our method can be extended to incorporate the right image of the stereo set. However, the right image does not provide any information for occluded regions, which are the most problematic areas when high accuracy stereo algorithms are used as inputs. Moreover, as presented, the proposed architecture is applicable to other label refinement problems formulated as either regression or classification.

3.1. RecResNet

Recurrent neural networks (RNN) have been applied with great success to problems involving sequential data processing like natural language processing [5, 12, 26, 32, 34]. An RNN can be drawn as a computational graph that contains cycles. Unfolding the graph converts the RNN to a directed acyclic graph similar to a feed-forward network but with an important difference, the parameters of the hidden layers inside the loop are shared across the network.

In our design we create a recurrent connection only from the output at one time step to the hidden units at the next time step. The core architecture, that is repeated over time, contains a main branch and two auxiliary branches (see Fig. 2). Each time a down-sampling step is performed the network creates a branch which operates at that scale, and produces a residual output at the original scale. Our network performs three down-sampling steps, each reducing the size of the input by a factor of two. We denote the residual outputs as R_s where s is equal to the down-sampling and up-sampling rate. The output of the network is then the residual combination of the input labels at time step t and the residuals generated from each branch.

$$Y_t = F(Y_{t-1} + R_2 + R_4 + R_8) \qquad (1)$$

where R_2, R_4 and R_8 are the residuals computed with s equal to 2, 4 and 8, respectively. With F, we denote the last block of convolutional layers. A visualization of this procedure is depicted in Fig. 3.

3.2. Architectural Design

The core architecture of RecResNet (see Fig. 2) begins by separately convolving Y_{init} and X and concatenating the learned features. We proceed by down-sampling by a factor of two using strided convolution and pass the resulting features to a semi-dense convolution block (s-block). Our semi-

Step	Layer	Kernel	Stride	Channels	Input	Step	Layer	Kernel	Stride	Channels	Input
Main branch											
$\sigma_{1.1}$	*conv*	5x5	1	32	Y_t	First branch					
$\sigma_{1.2}$	*conv*	5x5	1	32	X	$\sigma_{1.1}$	*conv*	5x5	1/2	32	σ_5
σ_2	*conv*	5x5	2	32	($\sigma_{1.1}$,$\sigma_{1.2}$)	$\sigma_{1.2}$	*sdb*	3x3	1	96	($\sigma_{1.1}$,σ_2)
σ_3	*sdb*	3x3	1	96	$conv_2$	$\sigma_{1.3}$	*conv*	5x5	1/2	32	$\sigma_{1.2}$
σ_4	*conv*	5x5	2	64	σ_3	$\sigma_{1.4}$	*sdb*	3x3	1	96	($\sigma_{1.3}$,$\sigma_{1.1}$,$\sigma_{1.2}$)
σ_5	*sdb*	3x3	1	192	σ_4	$\sigma_{1.5}$	*conv*	1x1	1	1	$\sigma_{1.4}$
σ_6	*conv*	5x5	2	128	σ_5	Second branch					
σ_7	*sdb*	3x3	1	384	σ_6	$\sigma_{2.1}$	*conv*	5x5	1/2	32	σ_3
σ_8	*conv*	5x5	1/2	128	σ_7	$\sigma_{2.2}$	*sdb*	3x3	1	96	($\sigma_{2.1}$,$\sigma_{1.1}$,$\sigma_{1.2}$)
σ_9	*sdb*	3x3	1	256	(σ_8,σ_4)	$\sigma_{2.3}$	*conv*	1x1	1	1	$\sigma_{2.2}$
σ_{10}	*conv*	5x5	1/2	64	σ_9	Output					
σ_{11}	*sdb*	3x3	1	192	(σ_9,σ_2)	σ_{15}	*sdb*	3x3	1	130	$(Y_t + \sigma_{14} + \sigma_{1.5} + \sigma_{2.3})$
σ_{12}	*conv*	5x5	1/2	32	σ_{11}	σ_{16}	*conv*	1x1	1	1	σ_{15}
σ_{13}	*sdb*	3x3	1	160	(σ_{12},$\sigma_{1.1}$,$\sigma_{1.2}$)						
σ_{14}	*conv*	1x1	1	1	σ_{13}						

Table 1. Detailed representation of our core architecture. Down-sampling and up-sampling is performed by the layers with a stride of 2 and $1/2$ respectively. sdb mean semi-dense block and (.,.) stands for concatenation.

dense convolution block is similar to the densely connected convolution block proposed by [15]. We only concatenate features every two convolutional layers, thus the term semi-dense block. The feature growth factor of our semi-dense block is two and the following convolutional layer performs feature reduction. After each down-sampling step we create a branch in the network. Each branch performs transpose (also called fractionally-strided) convolution [1] followed by semi-dense blocks and produces a single channel output at the same scale as the initial input. A combination of the input Y_t and the estimated residuals is performed and the result is processed by another semi-dense block which is followed by the final output layer. Table 1 lists the specifications of each layer in our proposed architecture. Every convolution layer, except for the output layers, is followed by a ReLu activation function. Dropout is used only in the down-sampling layers.

ResNet. Instead of performing a residual combination at each step σ and feeding it to the next layer, our architecture

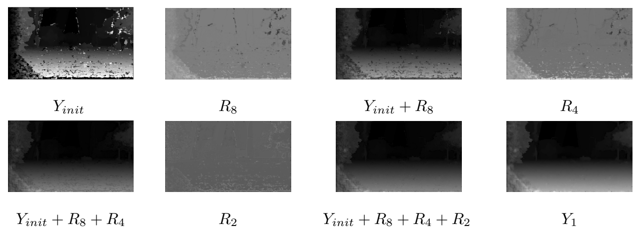


Figure 3. Visualization of the multi-scale residual corrections as applied by the network on the first iteration. The disparity values are positive. To visualize the residuals, we offset the real values, such that the minimum negative value equals zero. Notice that R_8, which is the coarsest scale, applies corrections to larger areas.

generates multiple residual corrections and applies them to the input labels Y_{t-1} in one step. This formulation is more powerful than the conventional approach, since the network is not forced to learn the optimal residual correction in a single step. For example, let p_{wrong} be an initial prediction that needs to be replaced, a single residual output must learn the correction c that needs to be applied to p_{wrong} such that $p_{wrong} + c = p_{correct}$. This might be feasible in a discrete scenario where the label space is small, but if we consider a continuous label space, in a regression task like ours, learning the correction c gets extremely hard. With our formulation the network can use a residual output to negate p_{wrong}, a second one to predict a new label and the third residual output can either be zero or act as a refinement component. In fact because we do not restrict any of the branches of the network to a specific task, any residual output can potentially play any of the aforementioned roles. The only restriction we introduce to the network is the loss function L, which minimizes the mean absolute difference from the ground truth Eq. (2). Essentially, this dictates that the linear combination of all the residuals and the input labels reduces the end-point-error of the final prediction.

$$L = \overline{|Y_{gt} - Y_{pred}|} \tag{2}$$

This objective forces the residual layers to only agree to the same value $v \pm \epsilon \approx 0$ when the residual correction is close to zero, meaning that either the network does not change the value of the initial prediction, or slightly refines it.

3.3. Advantages of recurrent formulation

The most important advantage of our approach is the ability of the network to learn the mapping between the initial labels Y_{init}, the intermediate labels Y_t and the final prediction Y_{final} jointly. During training, back-propagation through time (BPTT) [45] updates the parameters of the network in a single pass, accumulating gradients from all recurrences of the network. Regular back-propagation and back-propagation through time are not different after the gradient contribution at each time step has been computed. Although the parameters are shared across time steps, to compute the contribution of the gradients we need to save intermediate results after each time step and operate on the fully unfolded computational graph, increasing the space complexity to $\mathcal{O}(\tau)$, where τ is the number of time steps.

$$\nabla_W L = \sum_t J(h^{(t)})(\nabla_{h^{(t)}} L) h^{(t-1)^T} \quad (3)$$

$$\nabla_b L = \sum_t J(h^{(t)}) \nabla_{h^{(t)}} L \quad (4)$$

Equations (3) and (4) show a simple computation of the gradients of the weights W and biases b, in a recurrent network. We denote the Jacobian matrix as J, the hidden units as h and the time step as t. The exact formulation is more complex since it must account for skip connections. For more details about how gradients are computed in a recurrent network we refer readers to [45] and [11].

Due to parameter sharing, the recurrent network is able to capture long range interactions through time which in turn strengthens its statistical properties. The closest alternative to a recurrent formulation would be an iterative end-to-end approach, that performs back-propagation immediately after each time step output. However, such a training approach performs poorly in practice. More complicated techniques of training an iterative network, without the need to store intermediate gradients, have been proposed by [44, 21] with good performance during inference. However, the training process is disconnected since parts of the network have to be trained independently. Another alternative formulation is a cascade network [10]. However the space complexity of a cascade network is proportional to the number of time steps during training and deployment.

On the other hand, the recurrent formulation has a more holistic view of the refinement process and the number of parameters during inference is equal to the number of parameters of the core network, which makes it practical for deployment.

4. Experimental Evaluation

In this section we present an evaluation of the proposed architecture on the task of dense disparity map enhancement.

4.1. Implementation details

For our implementation we use the Tensorflow [2] framework on an NVIDIA Titan X GPU. We use MC-CNN-acrt [48] or Content-CNN [23] as our initial label generator G, which is treated as a black box. The input to our method is the left disparity map generated by G and the corresponding left image.

Our experiments are conducted on three datasets.The synthetic driving sequence from the Freiburg dataset [24], which we will refer to as synthetic, and two real world datasets, KITTI 2012 and 2015 [9, 25]. From the available sequences of the synthetic dataset we used the forward clean-pass 15mm focal length slow subset, which consists of 800 stereo frames with dense left and right ground truth disparity maps. Each of the KITTI datasets contains about 400 stereo pairs, split evenly into a training and a test set. Sparse ground truth covering approximately 30% of all pixels concentrated in the lower part of the images is captured by LIDAR and is publicly available only for the training sets.

As described in Section 3, we pose the problem of disparity correction as a regression task. with the loss function specified in Eq. (2). To give the network larger reward when it corrects non-occludes pixels, we double the weights of the non-occluded pixels in the loss function. This is enabled by the occluded and non-occluded ground truth disparity maps provided with the KITTI data. For the synthetic dataset it is trivial to compute occlusion masks since left and right disparity maps are provided.

We use the Adam optimizer [19] with an initial learning rate of 10^{-4}. Exponential decay is applied to the learning rate every 2000 training steps with a decay factor of 0.96. The mini-batch size is set to 1, mainly to reduce the memory requirements of loading more images on the GPU memory at each training step. At the beginning of each epoch we randomly shuffle the training samples to avoid having sequences of images which are very similar. This is very important when training on the synthetic dataset since the 800 frames are sequential. We experimented with two and three recurrences over the network. To increase the number of time steps we can randomly crop the input images during training. The smallest patch size that can be fed to our network is 48×48, but we prefer to use larger patches. For the three time step network, we where able to only use cropped patches of size 256×256, while for the two time step network we used full and cropped sized images. Our experiments showed that using full or cropped sized images did not affect the performance of the network

We use two different strategies to train our models. For ablation studies, we split the three datasets to training and validation sets. From the synthetic dataset we used 150 frames for validation and the rest for training. We used a similar split for the KITTI 2012 and 2015 training sets, using the last 40 frames for validation and the rest for fine-tuning. We conducted experiments using two baselines methods, **MC-CNN-acrt architecture including all optimization and filtering operations**, and the **raw cost estimated by Content-CNN**,

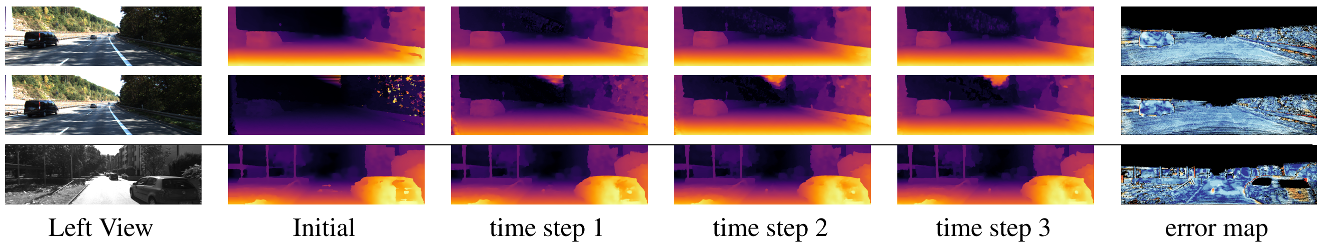


Figure 4. Qualitative results of our method RecResNet on the validation set of the KITTI 2012 and 2015 datasets. The first two rows are results for KITTI 2015. The first row shows the result when MC-CNN-acrt is used as the initial label generator G and the second row shows the final results when Content-CNN is used as the initial label generator G. The last row shows qualitative results on KITTI 2012 when the label generator function is MC-CNN-acrt.

Method	Steps	Initial	t-step 1	t-step 2	t-step 3
C-CNN+RRN	2	8.44%	4.93%	3.87%	-
C-CNN+RRN	3	8.44%	5.12%	3.71%	3.46%
MC-CNN+RNN	2	3.63%	3.06%	3.15%	-
MC-CNN+RNN	3	3.63%	3.07%	2.98%	3.04%

Table 2. Average error on the KITTI 2015 validation set for 2 and 3 time steps (t-step). Our method is able to improve MC-CNN-acrt and Content-CNN results by 18% and 59% respectively.

also used by [10]. We chose these two methods as our initial label generators G, to evaluate our method on both smooth very accurate inputs (MC-CNN-acrt) and somewhat more noisy inputs, but with less systematic errors (C-CNN). We trained RecResNet with two and three time steps and report our results on Table 2. We use the official script provided by the KITTI benchmark to evaluate all results.

The most important observation from our ablation studies, is that our method can produce approximately equivalent final estimates given as input disparity maps estimated by similar algorithms but with different levels of smoothness due to optimization and post-processing. More precisely our improvement on Content-CNN is of the order of 59%, which is on par with the 62% improvement reported by DRR. The improvement on MC-CNN-acrt is 18% since the initial disparity maps are more accurate. We also observe that adding one more recurrence leads to further improvements. Moreover, earlier time steps, in the three time step architecture make smaller improvements, but eventually the network reaches higher performance. This means that the network is able to capture long-range interactions and becomes more conservative, rather than greedy, in earlier time steps. This is an essential differentiation between the recurrent and alternative formulation, since the network does not learn to apply the best solution at each time step, which might hurt the overall performance, but tries to optimize the refinement process in a way that the final time steps can produce a better estimate.

In order to submit to the KITTI benchmarks, we pre-train our model on the synthetic dataset using both KITTI 2012 and 2015 as validation datasets. We use early stopping [11] to determine the number of epochs. After pre-training, we train on the entire training set of each benchmark, using the other dataset as an imperfect validation set. We terminate the training process when no further improvement is evident on the validation set. More precisely, to prepare for a submission to the KITTI 2012 benchmark, we trained the version of RecResNet with two time steps on the KITTI 2012 training set, using the KITTI 2015 training set for validation, and submitted to the KITTI 2012 benchmark. The same procedure was followed for the submission to KITTI 2015 benchmark.

The pre-training process takes approximately 24 hours. Then on average another 10 hours are required for the model to be fine tuned for the target dataset. Figures 4 and 5 show qualitative results for each time step of the process on all synthetic, KITTI 2012 and KITTI 2015 datasets. We can clearly observe that despite the fact that MC-CNN-acrt starts from a much better initial accuracy state, it is much harder to improve, since the errors tend to appear in structured patterns rather than noisy label predictions which is the case of Content-CNN.

The three datasets we used look very similar at first glance. There are, however, intrinsic details that can affect the performance of any learning method, especially methods like ours which act as the final stage of the stereo pipeline. The most important difference is how each dataset treats the windshields of the cars. KITTI 2012 does not provide any estimate for these areas. The synthetic dataset assigns the label of the background to them, while KITTI 2015 assigns the label of the foreground. Although it might not seem as a big difference, this small annotation change affects the consistency of the training process and forces us to use both KITTI datasets as validation sets during pre-training. Moreover, we approach the KITTI 2015 training process with caution, since the network must *re-learn* that the windshield labels must now be assigned to the foreground. Since we also want to use the entire KITTI 2015 training set to train our model, this leaves us with only KITTI 2012 training dataset to validate our training process. Because of this inconsistency, we need to not only rely on quantitative results but also on qualitative ones, to ensure that the network does indeed account for this change.

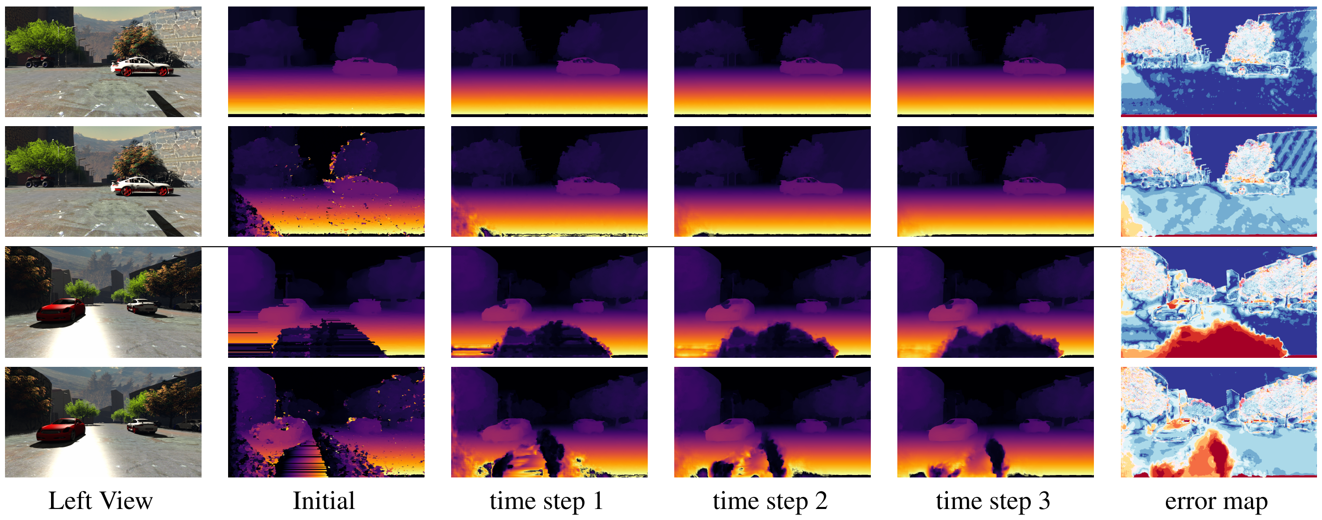


Figure 5. Qualitative comparison on the outputs of RecResNet during pre-training, using MC-CNN-acrt (rows 1 and 3) and Content-CNN (rows 2 and 4) on two different frames selected from the validation set of the synthetic dataset. On the first frame we observe that RecResNet is able to eliminate large noisy erroneous regions and converge to a similar output when compared with MC-CNN-acrt. On the second frame, we observe that RecResNet is able to handle large erroneous regions better when the initial labels are noisy rather than smooth.

	KITTI 12		KITTI 15		
Method	Out-Noc	Out-All	Out-Noc	**Out-All**	Description
PSMNet* [6]	1.49%	1.89%	2.14%	2.32%	End-to-end
CRL [27]	-	-	2.32%	2.48%	End-to-end
GC-NET [17]	1.77%	2.30%	2.61%	2.86%	End-to-end
LRCR* [16]	-	-	2.55%	3.03%	Component
RecResNet	**2.21%**	**2.94%**	**2.75%**	**3.10%**	Component
DRR [10]	-	-	2.76%	3.16%	Component
L-ResMatch [37]	2.27%	3.40%	2.91%	3.42%	Component
Displets v2 [13]	2.37%	3.09%	3.09%	3.43%	Semantic
PBCP[35]	2.36%	3.45%	3.17%	3.61%	Component
SGM-Net [36]	2.29%	3.50%	3.09%	3.66%	Component
MC-CNN [48]	2.61%	3.84%	3.33%	3.89%	Baseline

Table 3. Top published methods on KITTI 2012 and 2015 at the time of submission. The table is sorted based on KITTI 2015 main metric.

Finally, to evaluate our method, we compare it to the state of the art published methods on the KITTI 2012 and 2015 leaderboards (Table 3). We divide these methods into two categories: end-to-end and component. The end-to-end category includes three complete stereo learning pipelines [17, 27], while the component category includes improvements or components integrated to the baseline method (MC-CNN), or to a closely related method. Our method belongs to the component category and outperforms all other component methods on KITTI 2012, and ranks second, behind the soon-to-be published method LRCR.

The top-performing published methods are GC-NET [17] and CRL [27], both of which are complete end-to-end stereo pipelines. Our method lags a little behind them in accuracy. It should be noted that RecResNet outperforms DRR [10], which is a cascade method, while using a smaller network and significantly less data (DRR is trained on the whole synthetic dataset of [24] which contains $34k$ stereo images), and advances the state of the art for disparity map refinement, admittedly by a slim margin. For completeness, we also include the recently published methods PSMNet [6] and LRCR [16]. LRCR is a competing method to RecResNet, which incorporates the right image and the cost-volume to refine the estimate of L-ResMatch [37]. However, while using more information and a stronger baseline method as input from both RecResNet and DRR, the reported improvement is marginal, which is a strong indicator that our approach of using only the left image of the stereo pair is sufficient.

5. Conclusions

In this paper, we present RecResNet, which, to the best of our knowledge, is the first recurrent network to be applied to the problem of stereo matching. We believe that our recurrent, multi-scale, residual design enables RecResNet to make substantial improvements to the input disparity maps. Clearly, it is much harder to improve very accurate inputs than noisy ones. We have shown large error rate reduction on disparity maps produced by the accurate MC-CNN architecture [49], in the order of 18% on the KITTI 2012 benchmark and 20% on KITTI 2015 when all pixels with ground truth are considered.

Our future work will focus on integrating RecResNet into an end-to-end stereo matching pipeline and on applying this architecture on other pixel-wise labeling problems. Our code and trained models can be downloaded from `https://github.com/kbatsos/RecResNet`.

Acknowledgments This research has been supported by the National Science Foundation under Awards IIS-1527294 and IIS-1637761

References

[1] Transpose convolution (fractionally strided convolution). `http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html`.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] K. Alahari, C. Russell, and P. Torr. Efficient piecewise learning for conditional random fields. In *CVPR*, pages 895–901, 2010.

[4] J. T. Barron and B. Poole. The fast bilateral solver. In *ECCV*, 2016.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[6] J.-R. Chang and Y.-S. Chen. Pyramid stereo matching network. *arXiv preprint arXiv:1803.08669*, 2018.

[7] Z. Chen, X. Sun, L. Wang, Y. Yu, and C. Huang. A deep visual correspondence embedding model for stereo matching costs. In *ICCV*, pages 972–980, 2015.

[8] A. Dosovitskiy, P. Fischer, E. Ilg, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, pages 2758–2766, 2015.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, pages 1229–1235, 2013.

[10] S. Gidaris and N. Komodakis. Detect, replace, refine: Deep structured prediction for pixel wise labeling. In *CVPR*, 2017.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[12] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[13] F. Guney and A. Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *CVPR*, 2015.

[14] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*, pages 3279–3286, 2015.

[15] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[16] Z. Jie, P. Wang, Y. Ling, B. Zhao, Y. Wei, J. Feng, and W. Liu. Left-right comparative recurrent model for stereo matching. *arXiv preprint arXiv:1804.00796*, 2018.

[17] 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 *ICCV*, 2017.

[18] K.-R. Kim and C.-S. Kim. Adaptive smoothness constraints for efficient stereo matching using texture and edge information. In *International Conference on Image Processing (ICIP)*, pages 3429–3433, 2016.

[19] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] P. Knöbelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock. End-to-end training of hybrid cnn-crf models for stereo. In *CVPR*, 2017.

[21] K. Li, B. Hariharan, and J. Malik. Iterative instance segmentation. In *CVPR*, 2016.

[22] Y. Li and D. Huttenlocher. Learning for stereo vision using the structured support vector machine. In *CVPR*, 2008.

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

[24] N. Mayer, E. Ilg, P. Häusser, 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 *CVPR*, 2016.

[25] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[26] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[27] J. Pang, W. Sun, J. S. Ren, C. Yang, and Q. Yan. Cascade residual learning: A two-stage convolutional neural network for stereo matching. In *ICCV workshop on Geometry Meets Deep Learning*, 2017.

[28] H. Park and K. M. Lee. Look wider to match image patches with convolutional neural networks. *IEEE Signal Processing Letters*, 24(12):1788–1792, 2017.

[29] M.-G. Park and K.-J. Yoon. Leveraging stereo matching with learning-based confidence measures. In *CVPR*, pages 101–109, 2015.

[30] M. Poggi and S. Mattoccia. Learning a general-purpose confidence measure based on O(1) features and a smarter aggregation strategy for semi global matching. In *International Conference on 3D Vision (3DV)*, 2016.

[31] M. Poggi, F. Tosi, and S. Mattoccia. Quantitative evaluation of confidence measures in a machine learning world. In *ICCV*, 2017.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[33] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1-3):7–42, 2002.

[34] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[35] A. Seki and M. Pollefeys. Patch based confidence prediction for dense disparity map. In *BMVC*, 2016.

[36] A. Seki and M. Pollefeys. SGM-Nets: Semi-global matching with neural networks. In *CVPR*, 2017.

[37] A. Shaked and L. Wolf. Improved stereo matching with constant highway networks and reflective confidence learning. In *CVPR*, 2017.

[38] R. Slossberg, A. Wetzler, and R. Kimmel. Deep stereo matching with dense CRF priors. *arXiv preprint arXiv:1612.01725v2*, 2017.

[39] A. Spyropoulos, N. Komodakis, and P. Mordohai. Learning to detect ground control points for improving the accuracy of stereo matching. In *CVPR*, pages 1621–1628, 2014.

[40] A. Spyropoulos and P. Mordohai. Correctness prediction, accuracy improvement and generalization of stereo matching using supervised learning. *IJCV*, 118(3):300–318, 2016.

[41] T. Taniai, Y. Matsushita, Y. Sato, and T. Naemura. Continuous 3D label stereo matching using local expansion moves. *PAMI*, 2017.

[42] A. Tonioni, M. Poggi, S. Mattoccia, and L. Di Stefano. Unsupervised adaptation for deep stereo. In *ICCV*, 2017.

[43] S. Tulyakov, A. Ivanov, and F. Fleuret. Weakly supervised learning of deep metrics for stereo reconstruction. In *ICCV*, 2017.

[44] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. In *CVPR*, 2017.

[45] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[46] X. Ye, J. Li, H. Wang, H. Huang, and X. Zhang. Efficient stereo matching leveraging deep local and context information. *IEEE Access*, 5:18745–18755, 2017.

[47] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *CVPR*, 2015.

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

[49] J. Žbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *The Journal of Machine Learning Research*, 17(65):1–32, 2016.

[50] C. Zhang, Z. Li, Y. Cheng, R. Cai, H. Chao, and Y. Rui. Meshstereo: A global stereo model with mesh alignment regularization for view interpolation. In *ICCV*, pages 2057–2065, 2015.

[51] F. Zhang and B. W. Wah. Fundamental principles on learning new features for effective dense matching. *IEEE Transactions on Image Processing*, 2017.

[52] Y. Zhong, Y. Dai, and H. Li. Self-supervised learning for stereo matching with self-improving ability. *arXiv preprint arXiv:1709.00930*, 2017.