A NEW STEREO DENSE MATCHING BENCHMARK DATASET FOR DEEP LEARNING

Teng Wu¹, Bruno Vallet¹, Marc Pierrot-Deseilligny¹, Ewelina Rupnik¹

¹ LASTIG, Univ Gustave Eiffel, ENSG, IGN, F-94160 Saint-Mandé, France - firstname.lastname@ign.fr

KEY WORDS: Stereo dense matching, deep learning, benchmarking, LiDAR processing, 3D reconstruction

ABSTRACT:

Stereo dense matching is a fundamental task for 3D scene reconstruction. Recently, deep learning based methods have proven effective on some benchmark datasets, for example Middlebury and KITTI stereo. However, it is not easy to find a training dataset for aerial photogrammetry. Generating ground truth data for real scenes is a challenging task. In the photogrammetry community, many evaluation methods use digital surface models (DSM) to generate the ground truth disparity for the stereo pairs, but in this case interpolation may bring errors in the estimated disparity. In this paper, we publish a stereo dense matching dataset based on ISPRS Vaihingen dataset, and use it to evaluate some traditional and deep learning based methods. The evaluation shows that learning-based methods outperform traditional methods significantly when the fine tuning is done on a similar landscape. The benchmark also investigates the impact of the base to height ratio on the performance of the evaluated methods. The dataset can be found in https://github.com/whuwuteng/benchmark_JISPRS2021.

1. INTRODUCTION

Dense matching is a traditional topic in 3D reconstruction, which can be performed in stereo (with only two views) (Scharstein, Szeliski, 2002) or multi-view stereo (MVS) (Jensen et al., 2014). In this paper, we focus on stereo dense matching in the specific case of epipolar stereo pairs (where expected correspondences are on the same lines of the two images) as most of the recent deep learning approaches are limited to this simple configuration. The recent successes of deep learning based dense matching methods in the computer vision community (Laga et al., 2020) raise the question of their applicability in the geospatial context. This paper will investigate this question by comparing traditional and machine learning especially deep learning dense matching techniques on geospatial data.

1.1 Traditional methods

Traditional dense matching methods (Hirschmuller, 2005) are usually decomposed into four steps: hand-crafted features computation, feature matching across images (i.e., the cost volume), cost aggregation and disparity refinement. They can be divided into local and global methods. Local methods mainly take into consideration the local features (hand-crafted feature) (Hirschmuller, Scharstein, 2007) or a local region (Tombari et al., 2008). The global methods mainly add an optimization based cost aggregation step, based on dynamic programming (Van Meerbergen et al., 2002), belief propagation (Sun et al., 2003) or Graphcut optimization (Boykov, Kolmogorov, 2004). Semi global matching (SGM) is a reference method combining mutual information and dynamic programming optimization on several directions (Hirschmuller, 2005). A GPU variant of SGM (Hernandez-Juarez et al., 2016) on the full image scale is also evaluated in this paper, as well as a Graphcut based method using plane constraints (Tanai et al., 2017). Graphcut based methods are slower than SGM which is often considered to offer the best balance between efficiency and accuracy (Bethmann, Luhmann, 2015).

1.2 Learning based methods

Traditional machine learning base methods have been proposed to address the problem of dense matching. Support vector machine can be used to learn a linear discriminant function (Li, Huttenlocher, 2008). Because features have their pros and cons, a random forest (RF) can be used to fuse several feature types, e.g. census, normalized cross-correlation (NCC), zero-mean sum of Absolute Differences (SAD), SAD of Sobel feature (Batsos et al., 2018). After feature fusion, a traditional optimization is used to obtain the final result.

With development of deep learning, some steps of the traditional methods can be replaced by deep learning counterparts (Laga et al., 2020). For instance, 2D convolutional neural networks (CNN) prove effective in feature extraction (Zbontar, LeCun, 2016). In order to make CNN efficient, feature similarity calculation can be treated as a multi-class classification (Luo et al., 2016). After feature similarity calculation, traditional optimization is used to obtain the final result. SGM-Net uses a CNN network to provide learned penalties for SGM (Seki, Pollefeys, 2017). GC-Net uses a 3D CNN based network as cost aggregation (Kendall et al., 2017). Pyramid Stereo Matching network uses spatial pyramid pooling and 3D CNN (Chang, Chen, 2018). High resolution stereo network uses upsampling in the 2D CNN network, so that the 3D cost volume does not need to be down-scaled (Yang et al., 2019a). To address the high memory consumption for high-resolution image matching, a recent method (Duggal et al., 2019) proposes to prune the 3D cost volume with a differential patch match method. CNN and conditional random fields (CRF) can be combined into a hybrid CNN-CRF model which is more effective than fully-connected CRFs (Kendall et al., 2017). While these methods were developed by the computer vision community on indoor or outdoor dataset, we tested them on the ISPRS Vaihingen dataset (Knöblereiter et al., 2018), and we used Lidar data to generate ground truth disparity maps for training and evaluation.

* Corresponding author

This contribution has been peer-reviewed.
https://doi.org/10.5194/isprs-archives-XLIII-B2-2021-405-2021 | © Author(s) 2021. CC BY 4.0 License. 405
This paper evaluates the state-of-the-art deep learning based dense matching approaches for which a public implementation is available.

1.3 Benchmark data

Table 1 lists some benchmarks for stereo dense matching on real scenes proposed in Section 4. Some are popular and widely used: the indoor Middlebury dataset (Scharstein et al., 2014); the KITTI datasets in two versions, KITTI 2012 (Geiger et al., 2012) and KITTI 2015 (Mayer et al., 2016); or the ETH3D dataset containing stereo pairs (Schops et al., 2017). Using advanced computer graphics, some virtual datasets have also been generated (Yang et al., 2019a; Mayer et al., 2016). The strong ongoing research activity on autonomous driving has also resulted in several dedicated datasets such as DrivingStereo or AppolloScape (Yang et al., 2019b; Huang et al., 2019).

In aerial photogrammetry, benchmark datasets focus mainly on MVS dense matching (Cavegn et al., 2014). For satellite imagery, IARPA dataset is widely used for image dense matching evaluation (Bosch et al., 2016), the traditional evaluation method depends on DSM (Bosch et al., 2019), but the pipeline contains other steps, for example, point cloud fusion and DSM generation (Gournet et al., 2020). These steps can influence the accuracy. A recent satellite image stereo benchmark from LiDAR DSM (Patil et al., 2019) uses IARPA dataset. At the same time, for high density LiDAR, when generating the DSM, points on the walls are ignored.

Generating ground truth data from real scenes is challenging. To avoid errors introduced by grid interpolation or point cloud fusion and keep the points on walls, we propose a method to generate sparse disparity ground truth for training deep learning based methods on the proposed benchmark.

1.4 Overview

The paper is structured as follows: Benchmark data generation is described in Section 2. Evaluations are provided in Section 3. Finally, conclusions are drawn and the perspectives are proposed in Section 4.

2. GROUND TRUTH DATA GENERATION

In this paper, we use the Vaihingen dataset from the ISPRS 3D reconstruction benchmark which provides a good registration of oriented images and LiDAR point clouds. The dataset is composed of 20 images with a depth of 11 bits and the ground sample distance (GSD) of 8 cm. The median LiDAR point density is 6.7 points/m² (Rottensteiner et al., 2012). From this data, epipolar stereo image pairs and the corresponding disparity maps are generated in four steps: data preprocessing, epipolar image generation, point cloud projection, and benchmark data production.

2.1 Data preprocessing

The Vaihingen dataset contains the images and their orientation parameters, plus a LiDAR point cloud. First, we convert the orientation files into the open source MicMac format (Pierrot-Deseilligny et al., 2014) on which our workflow is based. As the images can only capture the apparent surface we keep only the first LiDAR echo. Moreover, we remove isolated points based on the point cloud density using a grid filtering approach (Cho et al., 2004).

2.2 Epipolar image generation

The first step of our process is to create epipolar image pairs from input image pairs with sufficient footprint overlap. Coarse image footprints are obtained using an approximate height of the scene and the Computational Geometry Algorithms Library (CGAL) (Flötotto, 2020) is used to compute their intersections. Only image pairs with an intersection area above half the image footprint are considered for epipolar rectification, which was also done using the MicMac library. The corresponding orientation parameter files are generated in order to allow the direct projection of 3D points into the rectified images.

2.3 Point cloud projection

The ground truth disparity maps used for both training and evaluation are computed from the LiDAR point cloud. The perspective projection model is used to project the 3D point cloud into image plane. The disparity \( d \) is defined in Equation (1).

\[
    d = x_l - x_r
\]

where \( x_l \) and \( x_r \) are the projection coordinate on \( x \) axis in the image plane after projecting the 3D point cloud to the left and right images.

Because the point cloud is sparse, it is difficult to predict the occlusions which can be important (Bevilacqua et al., 2017). An example of occlusion (in the left image) is shown in Figure 1. Some points from the ground area should be removed as they are occluded by the roof.

According to the epipolar geometry, the disparity is related to the depth of object (Jain et al., 1995), in Equation (2), \( z \) is the depth, \( b \) is base line length, \( f \) is the focal length, \( d \) is the disparity. The disparity is inversely proportional to the depth.

\[
    z = \frac{b \cdot f}{d}
\]

For the aerial images, the disparity is related to the elevation, such that ground points have a larger disparity than points on the roof. Similarly as (Biasutti et al., 2019), we use a filtering based on the density to remove occluded points: if the difference between the disparity and its median is larger than a threshold, then the point is removed. This filter removes the occluded points quite effectively, as shown in Figure 2.

2.4 Benchmark data production

After the disparity map generation, the benchmark dataset is split into a training and testing sets. In our experiment, we split the dataset according to the LiDAR point cloud area, as shown in Figure 3. Similarly to the KITTI dataset, the final data are 1024x1024 cropped images with 8bit depth color band, and the disparity images are stored on 16bits with the disparity value scaled by 256. After cropping the left image, the offset of the right image is set to the average disparity in this area in order to avoid too large disparity values. If the footprint of a cropped image intersects the training area, it is attributed to the training dataset, otherwise to the testing dataset. An example is shown in Figure 4. Using this procedure, our training set contains 585 image pairs while the testing set has 507 image pairs. In the training dataset there are 337 stereo pairs along the flight strip (along-strip), whereas the testing set contains 248 (Rottensteiner et al., 2012). From this dataset, it is attributed to the training dataset, otherwise to the testing dataset. An example is shown in Figure 4. Using this procedure, our training set contains 585 image pairs while the testing set has 507 image pairs. In the training dataset there are 337 stereo pairs along the flight strip (along-strip), whereas the testing set contains 248.
Table 1. Benchmark datasets for disparity estimation. Stereo number is the total number of stereo pairs used for both training and testing.

<table>
<thead>
<tr>
<th>Data</th>
<th>Year</th>
<th>Scene</th>
<th>Stereo number</th>
<th>Disparity density</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Middlebury V2</td>
<td>2003-2006</td>
<td>indoor</td>
<td>32</td>
<td>dense</td>
<td>(Scharstein, Szeliski, 2003)</td>
</tr>
<tr>
<td>Middlebury V3</td>
<td>2014</td>
<td>indoor</td>
<td>33</td>
<td>dense</td>
<td>(Scharstein et al., 2014)</td>
</tr>
<tr>
<td>ETH3D</td>
<td>2017</td>
<td>indoor + outdoor</td>
<td>47</td>
<td>dense</td>
<td>(Schops et al., 2017)</td>
</tr>
<tr>
<td>KITTI2012</td>
<td>2012</td>
<td>outdoor (driving)</td>
<td>389</td>
<td>sparse</td>
<td>(Geiger et al., 2012)</td>
</tr>
<tr>
<td>KITTI2015</td>
<td>2015</td>
<td>outdoor (driving)</td>
<td>1600</td>
<td>sparse</td>
<td>(Mayer et al., 2016)</td>
</tr>
<tr>
<td>DrivingStereo</td>
<td>2019</td>
<td>outdoor (driving)</td>
<td>182188</td>
<td>sparse</td>
<td>(Yang et al., 2019)</td>
</tr>
<tr>
<td>ApolloScape</td>
<td>2019</td>
<td>outdoor (driving)</td>
<td>5165</td>
<td>sparse</td>
<td>(Huang et al., 2019)</td>
</tr>
<tr>
<td>SatStereo</td>
<td>2019*</td>
<td>satellite</td>
<td>72</td>
<td>dense</td>
<td>(Patil et al., 2019)</td>
</tr>
<tr>
<td>DFC2019</td>
<td>2019</td>
<td>satellite</td>
<td>8634</td>
<td>dense</td>
<td>(Bosch et al., 2019)</td>
</tr>
<tr>
<td>Ours</td>
<td>–</td>
<td>aerial</td>
<td>1092</td>
<td>sparse</td>
<td>–</td>
</tr>
</tbody>
</table>

Similarity feature, the code is written in C++ (micmacIGN, 2020).

1. **MICMAC**: A variant of SGM implemented in MicMac (Pierrot-Deseilligny, Paparoditis, 2006), using NCC as the similarity feature, the code is written in C++ (micmacIGN, 2020).

2. **SGM (GPU)**: A variant of SGM based on GPU (Hernandez-Juarez et al., 2016), using census as similarity feature, the code is based on C++ and CUDA (Hernandez-Juarez, 2020).

3. **GraphCuts**: A Graphcut based method (Taniai et al., 2017), using intensity and gradient consistencies, and using plane as a constraint, the code is written in C++ (Taniai, 2020).

4. **CBMV**: A coalesced bidirectional matching volume based method (Batsos et al., 2018) using a random forest (RF) to fuse the features, followed by dynamic programming optimization (CBMV (SGM)) or Graphcut (CBMV (GraphCuts)) which has the same implementation with GraphCuts. The code is based on C++, CUDA (SGM implementation) and Python (Batsos, 2020).

5. **DeepFeature**: A deep learning network is used to obtain the features, then optimize with dynamic programming (Luo et al., 2016), the code is based on C++, CUDA (SGM implementation) and Lua torch (Luo, 2020).

6. **PSM net**: A pyramid stereo matching network using a spatial pyramid pooling and a 3D CNN to train an end-to-end model (Chang, Chen, 2018). The code is based on Pytorch (Chang, 2020).

7. **HRS net**: A high resolution stereo network structure improving the accuracy without downscaling the cost volume.
Figure 3. Training and testing area on Vahingen.

(Yang et al., 2019a). The code is based on Pytorch (Yang, 2020).

8. **DeepPruner**: A 3D cost volume is pruned with a differential patch match method to avoid a full cost volume calculation and evaluation (Duggal et al., 2019). The code is based on Pytorch (UberResearch, 2020).

### 3.1 Experimental setup

The disparity search range is an important parameter for stereo dense matching. Some methods do not need this parameter, i.e., **MICMAC** and **DeepPruner**. In **SGM(GPU)**, the range is set to 128 and is dictated by the implementation. For other methods, it is set to 192.

For machine learning based methods, the training data and hyper-parameters impact significantly the results. For the Random Forest based method **CBMV**, 54 epipolar pairs are used for training. For deep learning based methods, all the training data is used. For the evaluation, all the testing data is used for all methods.

In our experiments, we compare deep learning methods pre-trained on the KITTI dataset and fine tuned on our Vahingen ground truth disparity. We found that batch size has a strong impact on the memory requirements and on the accuracy, as shown in Table 2 for **PSM net**. We decided to use the default batch size proposed in the implementation: 12 for **PSM net**, 28 for **HRS net** and 16 for **DeepPruner**. For the fine-tuning experiments on Vahingen dataset, we did the same for the number of epochs: 20 for **DeepFeature**, 500 for **PSM net**, 700 for **HRS net** and 900 for **DeepPruner**.

Table 2. Influence of the batch for **PSM net**.

<table>
<thead>
<tr>
<th>Method</th>
<th>Batch size</th>
<th>&lt;2-pixel</th>
<th>&lt;3-pixel</th>
<th>&lt;5-pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSM net</td>
<td>3</td>
<td>81.988</td>
<td>87.022</td>
<td>91.501</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td><strong>84.065</strong></td>
<td><strong>88.324</strong></td>
<td><strong>92.395</strong></td>
</tr>
</tbody>
</table>

### 3.2 Pixel error

In stereo dense matching evaluation, the pixel error (error between the estimated disparity and the ground truth) is an important metric to analyze the performance. In our experiment, we compute the 2, 3 and 5-pixel error. Accuracy is the percentage of positive pixels in valid pixels within the ground truth.

Table 3. Evaluation of methods on testing data. For **SGM(GPU)**, only evaluate disparity smaller than 128, for other methods maximum disparity is 192.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy [%]</th>
<th>&lt;2-pixel</th>
<th>&lt;3-pixel</th>
<th>&lt;5-pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>MICMAC</td>
<td>67.169</td>
<td>74.283</td>
<td>81.429</td>
<td></td>
</tr>
<tr>
<td>SGM(GPU)</td>
<td>71.564</td>
<td>78.539</td>
<td>84.799</td>
<td></td>
</tr>
<tr>
<td>GraphCuts</td>
<td>71.704</td>
<td>76.404</td>
<td>80.951</td>
<td></td>
</tr>
<tr>
<td><strong>CBMV(SGM)</strong></td>
<td>74.941</td>
<td>80.540</td>
<td>85.342</td>
<td></td>
</tr>
<tr>
<td><strong>CBMV(GraphCuts)</strong></td>
<td>76.387</td>
<td>82.229</td>
<td>87.227</td>
<td></td>
</tr>
<tr>
<td>DeepFeature</td>
<td>78.265</td>
<td>83.982</td>
<td>88.787</td>
<td></td>
</tr>
<tr>
<td><strong>PSM net</strong></td>
<td><strong>84.065</strong></td>
<td><strong>88.324</strong></td>
<td><strong>92.395</strong></td>
<td></td>
</tr>
<tr>
<td>HRS net</td>
<td>79.135</td>
<td>85.243</td>
<td>91.238</td>
<td></td>
</tr>
<tr>
<td>DeepPruner</td>
<td>83.568</td>
<td>87.893</td>
<td>92.223</td>
<td></td>
</tr>
</tbody>
</table>

As shown in Table 3, the result shows that machine learning based methods have a significantly better performance than traditional methods. While slower, Graphcut optimization achieves better results than dynamic programming. **PSM net** has the best result among all tested deep learning methods.

As for the deep learning methods, the data used for learning highly influences the results. A comparison between the results obtained with a pretrained model and with a model finetuned on our learning set is provided in Table 4. This experiment shows that the dependency of deep learning methods on the training data depends on the method. Comparing to **DeepFeature** which
in order to investigate the effect of the $B/H$, we experimented two different trainings of the model: using all the training dataset and only the along-strip images. As shown in Table 6, the result in across-strip is not as good as along-strip. For the along-strip based training, the dataset size is smaller than the all-based training, but there is no big difference on the results of along-strip testing data. This means that the along-strip images are sufficient for the fine-tuning. However, using images across-strip to fine-tune can improve the performance on across-strip pairs. Especially, the 1-pixel error can be improved significantly for all the methods.

### 3.3 Error analysis

Pixel error is a statistical metric. When the scene is complex, errors can show different patterns on buildings (man-made structure), ground, vegetation and so on. To visualize this, in Figure 5 we provide the error maps on different scenes. Because the ground truth is sparse, the error maps are interpolated with the nearest point in order to facilitate the interpretation. The disparity map is visualized using an ambient occlusion shading implemented in MicMac as shown in Figure 6. Figures 6(p) and 6(r) demonstrate that the Graphcut based methods are smooth on building’s roofs. Figures 6(k) to 6(m) demonstrate that the deep learning based methods perform well on building’s boundary (disparity discontinuity).

**DeepPruner** proves to have the second best result among these methods, but the pre-trained result is poor as shown in Figure 7(i), which means that **DeepPruner** is highly dependent on the training dataset. Because there is a plane constraint in the GraphCut method, the result is not satisfactory on trees, as shown in Figure 7(e).

### 4. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a new stereo benchmark dataset adapted to deep learning and evaluate some existing traditional and learning based methods on this dataset. Experiments show that:

- Learning based methods perform better than traditional methods.
- Fine-tuning deep learning architectures by transfer learning on a specific dataset improves the results significantly.
- Along-strip images and across-strip images should be considered in training.

Our future work will focus on extending the benchmark to satellite images, but also studying how training on different scenes and sensor types affect the performance of learning-based methods.

### ACKNOWLEDGEMENTS

This research has been funded by AI4GEO project. The Vaihingen data set was provided by the German Society for Photogrammetry, Remote Sensing and Geoinformation (DGPF).

### REFERENCES


<table>
<thead>
<tr>
<th>Method</th>
<th>Training data</th>
<th>Testing data</th>
<th>Accuracy [%]</th>
<th>&lt; 1-pixel</th>
<th>&lt; 2-pixel</th>
<th>&lt; 3-pixel</th>
<th>&lt; 5-pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepFeature</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(along-strip)</td>
<td>69.552</td>
<td>82.135</td>
<td>86.723</td>
<td>90.588</td>
<td></td>
</tr>
<tr>
<td>DeepFeature</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(across-strip)</td>
<td>33.781</td>
<td>65.030</td>
<td>75.806</td>
<td>83.247</td>
<td></td>
</tr>
<tr>
<td>DeepFeature</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(along-strip)</td>
<td>67.854</td>
<td>81.529</td>
<td>86.588</td>
<td>90.750</td>
<td></td>
</tr>
<tr>
<td>DeepFeature</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(across-strip)</td>
<td>52.742</td>
<td>71.710</td>
<td>78.750</td>
<td>85.119</td>
<td></td>
</tr>
<tr>
<td>PSM net</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(all)</td>
<td>76.609</td>
<td>85.402</td>
<td>89.178</td>
<td>92.837</td>
<td></td>
</tr>
<tr>
<td>PSM net</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(across-strip)</td>
<td>44.010</td>
<td>74.271</td>
<td>81.456</td>
<td>87.230</td>
<td></td>
</tr>
<tr>
<td>PSM net</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(all)</td>
<td>76.054</td>
<td>85.392</td>
<td>89.350</td>
<td>93.067</td>
<td></td>
</tr>
<tr>
<td>PSM net</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(across-strip)</td>
<td>69.389</td>
<td>81.399</td>
<td>86.264</td>
<td>91.046</td>
<td></td>
</tr>
<tr>
<td>HRS net</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(all)</td>
<td>68.510</td>
<td>80.904</td>
<td>86.603</td>
<td>92.093</td>
<td></td>
</tr>
<tr>
<td>HRS net</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(across-strip)</td>
<td>39.744</td>
<td>68.830</td>
<td>78.006</td>
<td>86.036</td>
<td></td>
</tr>
<tr>
<td>HRS net</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(all)</td>
<td>68.856</td>
<td>81.140</td>
<td>86.809</td>
<td>92.298</td>
<td></td>
</tr>
<tr>
<td>HRS net</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(across-strip)</td>
<td>58.567</td>
<td>75.118</td>
<td>82.099</td>
<td>89.110</td>
<td></td>
</tr>
<tr>
<td>DeepPruner</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(all)</td>
<td>75.780</td>
<td>84.775</td>
<td>88.669</td>
<td>92.629</td>
<td></td>
</tr>
<tr>
<td>DeepPruner</td>
<td>Vaihingen(along-strip)</td>
<td>Vaihingen(across-strip)</td>
<td>40.539</td>
<td>73.143</td>
<td>80.525</td>
<td>86.683</td>
<td></td>
</tr>
<tr>
<td>DeepPruner</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(all)</td>
<td>76.045</td>
<td>84.961</td>
<td>88.972</td>
<td>92.937</td>
<td></td>
</tr>
<tr>
<td>DeepPruner</td>
<td>Vaihingen(all)</td>
<td>Vaihingen(across-strip)</td>
<td>68.926</td>
<td>80.772</td>
<td>85.728</td>
<td>90.790</td>
<td></td>
</tr>
</tbody>
</table>


Figure 6. Error map and disparity visualization on building area. From (a)-(m) is the error map, and from (n)-(z) is the disparity map visualization. The left image is shown in Figure 5(a).


Figure 7. Error map and disparity visualization on tree area. From (a)-(m) is the error map, and from (n)-(z) is the disparity map visualization. The left image is shown in Figure 5(b).


