ECE 570 Term Paper : A Review of Depth Estimation Algorithms Using Stereo Imaging

Anonymous Authors[1]

Abstract

The purpose of this paper is to review state-of-the-art methods for generating disparity maps between stereo images and evaluate the effectiveness of using the notable PatchMatch algorithm (1) to efficiently estimate disparity between stereo images. When compared to the classic stereo algorithm approach of searching over all possible disparities, my algorithm implementing PatchMatch was able to greatly reduce runtime while maintaining a comparable error rate. This paper presents a simplified approach to disparity mapping, thus more work needs to be done to build upon this algorithm before it is tested against advanced CNN disparity estimating models such as DeepPruner (3).

1. Motivation and Problem Setting

The ability to estimate absolute or relative depth based on stereo imagery is an ongoing issue that has ramifications in many fields such as autonomous vehicles, robotics, and image editing. Currently, most autonomous or semi-autonomous vehicles and robots rely on LiDAR, sonar, or other similar technologies to estimate depth for their environment. However, these solutions often have a steep cost/performance tradeoff. Further, post-processing image and video editing tools do not have the luxury of capturing distances from waveforms and add to the demand for image-based depth estimations.

The classical approach to solving the problem of depth estimation for stereo imagery is to create a mapping between patches, A, centered around each pixel in the left image with patches, B, in its paired right image while minimizing some cost function, $f(A, B)$, typically a simple error sum of squares (SSE). The horizontal pixel disparity between paired patches A and B can then be used to calculate the absolute distance between the camera and the object by using information about the positioning and resolution of the two cameras.

However, there are several issues with the classical approach, one of which is runtime. To guarantee the minimization of the cost function for a particular patch A, all patches B within range of possible disparities must be evaluated. This is computationally burdensome and results in runtimes that are unacceptable for most real-world applications.

Therefore, the exists a need for an algorithm that can quickly converge at an acceptable approximate disparity. The following paper will explore one solution for this problem which utilizes the landmark 2009 algorithm, PatchMatch (1) that was originally presented to quickly locate approximate nearest-neighbors (ANNs) for pixel patches by taking advantage of the natural structure of images and the law of large numbers.

2. Related Work

2.1. PatchMatch

Many modern image editing tools used for retargeting, completing holed, and reshuffling content rely on finding approximate nearest neighbor (ANN) matches between clusters of pixels within or between images. Efficiently locating these nearest neighbor (NN) matches is not trivial, however, and classical approaches produce runtimes inapplicable for real-time image editing tools.

In the article "PatchMatch: A randomized Correspondence Algorithm for Structural Image Editing," Barnes et al. (1) introduced a new algorithm for finding ANN matches that requires minimal memory and greatly reduces the search field for possible matches, thereby reducing the algorithm's runtime. While the introduction of PatchMatch has been effective in many image editing tools since 2009, there still exist limitations to the algorithm that have not yet been addressed.

Barnes et al. set out in their paper to provide a more efficient algorithm for finding ANN matches between image patches because, as they state, "the performance of these [image editing] tools must be fast enough that the user quickly sees intermediate results in the process of trial and error." PatchMatch greatly reduces the disparity search range for a given patch by taking advantage of the natural structure of images and the law of large numbers, seen through their propagation and random search phases respectively.

Offsets for each patch in the first image are first randomly initialized across the uniform range of all possible disparities. Playing off the law of large numbers, Barnes et al.

conclude that within a patch of random offsets, "the odds of at least one offset being correctly assigned are quite good (1 – (1 – 1/M)^M)) or approximately 1 – 1/e for large M, where M is equal to the patch sizes.

Barnes et al. assumed that, due to the natural structure of images, within an object, patches are likely to share similar disparity values and thus good patch offsets are shared with neighboring pixels in the patch. Barnes et al. state that the majority of patches will converge within the first iteration, however for improved performance the propagation and random search phases can be repeated for any k number of iterations.

Since being published in 2009, the PatchMatch algorithm has been frequently cited and incorporated in a multitude of real-time image editing tools. The strength of PatchMatch lies in its ability to quickly find approximate matches for a specific patch of pixels in a given image without necessarily finding the absolute minimum cost match within the given discrepancy range. This is particularly useful in situations when approximate matches provide valuable information, such as hole filling. However, in some edge cases when tasked with locating specific corresponding patches PatchMatch may converge at local minima some distance away from the true matches.

One example of these edge cases is if an object in the image is particularly smooth, i.e. most pixels in the given region are close in color. Without any constraints informing the algorithm where the patch belongs relative to other patches or objects in the image, PatchMatch will have no way of discriminating against surrounding patches with identical pixels. This is because of the simplicity of the cost function that simply finds the error sum of squares (SSE) between the two patches.

Another case where PatchMatch falls short is with repeated patterns. A good example would be a building with many similar looking windows where PatchMatch can get caught in a local minimum that matches one window with a patch from another neighboring window. Once again, this would still be valuable for some image editing tools, such as hole-filling, but would be impractical for locating absolute nearest neighbors between stereo images.

A possible way of resolving these two edge cases would be to modify PatchMatch to provide k-ANN patches and have surrounding patches vote on which disparity is most likely to be the true match. This would effectively blur the disparity map, resulting in higher continuity across surfaces. Also, more advanced CNN classification algorithms could be implemented as a pre-processing step to further improve disparity continuity across surfaces.

PatchMatch focuses on the important task of reducing the disparity search range for ANN pixel patches so that matching can be done in real-time. This algorithm has proved quite useful for many high-level image editing tools such as retargeting, completion, and reshuffling. However, there are still some limitations to this algorithm, such as dealing with smooth or non-textured images and patterned images, that still need to be addressed by future research.

2.2. DeepPruner

Since its introduction in 2009, the PatchMatch algorithm (1) has inspired and been utilized in many more complex algorithms for image processing, mostly for its ability to quickly converge on local minima thus drastically reducing runtime. PatchMatch has also been used as a way of narrowing the search field of possible candidates that are evaluated by a more robust cost aggregation algorithm. However, due to its dependence on the *arg max* function for evaluating minimum costs, the PatchMatch algorithm is not innately differentiable and has thus been left out of modern CNN models.

In the article "DeepPruner: Learning Efficient Stereo Matching via Differentiable PatchMatch," Duggal et al. (3) attempt to leverage the efficiency gains of PatchMatch to develop their own end-to-end differentiable model that provides real-time performance for stereo disparity estimation. The algorithm consists of several stages, including feature extraction followed by differentiable PatchMatch used to prune the search space of disparities. The algorithm then goes through another round of differentiable PatchMatch before conducting a 3D cost volume aggregation on the predicted ranges of disparities. Lastly, the disparity estimates are passed through a refinement phase that will attempt to reduce noise based on low-level feature information from the left image.

To evaluate the performance of DeepPruner, Duggal et al. compared two variations of their algorithm, *DeepPruner-Best* and *DeepPruner-Fast*, with some of the best-perfoming and fastest real-time models to date. The models were all compared against *SceneFlow* (10), a synthetic dataset with available ground truth disparities, as well as *KITTI 2015* (5), a collection of real-world stereo images collected alongside a Velodyne HDL-64E Lidar laser scanner used to collect ground truth disparities.

The results of their experiments show that the *DeepPruner-Best* algorithm was consistently running over 2x faster than all other leading algorithms while still yielding comparable End-Point-Error (EPE) values. *DeepPruner-Fast*, however, while having a runtime 3x quicker on average than *DeepPruner-Best*, was still running much longer compared to leading real-time models like *MAD-Net* (11) with a runtime of 60ms versus 20ms per frame.

The difference in runtime between *DeepPruner-Fast* and *MAD-Net* now raises the question, what constitutes a true

real-time algorithm? In the context of image editing software, one can assume that 60ms run-time is more than sufficient, but in the context of identifying objects surrounding a vehicle as in the *KITTI 2015* dataset, one could argue that a faster algorithm, such as *MAD-Net* would be required. To reach lower run-times, more experimentation must be done, perhaps including a further down sampling of the DeepPruner cost volume.

Another issue with DeepPruner lies in the model's implementation of Differentiable PatchMatch, specifically the omission of the random search phase. In the classical PatchMatch algorithm, each propagation phase is followed up with random samples from within the disparity search field to locate better possible matches and escape local minima. In their article, Duggal et al. chose to omit the local random resampling to further simplify the algorithm which leaves the algorithm susceptible to pruning the disparity search region around local minima that do not reflect the true disparity of a patch.

Although DeepPruner performed quite well against the *SceneFlow* and *KITTI 2015* datasets, their model may have been overfitted for these samples that contain mostly synthetic scenes or scenes from the perspective of a car on the road. To further test the rigidity of the DeepPruner, more experimentation should be done against a more diverse set of stereo images.

2.3. Efficient Deep Learning for Stereo Matching

In recent years, much progress has been made to utilize convolutional neural networks (CNNs) for stereo depth estimation. Most current algorithms utilize a siamese architecture, processing both the left image and the right image through feature extracting network layers, before later concatenating the two feature volumes to be sent through further post-processing layers. This approach has proven effective, but it oftentimes requires, according to Luo et al., "a minute on the GPU to process a stereo pair" (9).

In their article titled, "Efficient Deep Learning for Stereo Matching," Luo et al. (9) present a novel approach to improve the efficiency of the siamese CNN architecture by simply taking the inner product between the left and right feature volumes to produce the disparity estimates. Also, in their algorithm, Luo et al. maintain probability estimates over all possible disparities during training which allows the algorithm to gain insights beyond simple softmax values utilized by prior approaches.

To test their algorithm, Luo et al. compared their performance on the *KITTI 2015* and *2012* (5) benchmark datasets against those of state-of-the-art algorithms such as *MBM* (4), *SPS-St* (12), *MC-CNN* (13), and *Displets v2* (6). The results of their experiments show that their algorithm was able to achieve comparable error while showing significant decreases in runtime. However, after performing post-processing to each algorithm, Luo et al.'s approach was unable to match the error levels produced by state-of-the-art algorithms

One issue that Luo et al. faced when comparing their algorithm with the current state-of-the-art algorithms was that most of the current smoothing techniques used for post-processing are tailored to the traditional approach of concatenating the siamese networks and applying further network layers. Luo et al. made some attempts at smoothing their output, utilizing slanted-plane estimates and other sophisticated post-processing techniques to resolve conflicts between left and right image disparity estimates. Nonetheless, despite outperforming all state-of-the-art algorithms in runtime and error performance before smoothing, Luo et al.'s algorithm falls short after post-processing.

As common with many stereo matching algorithms, Luo et al.'s algorithm also fails to give accurate estimates for matching smooth patches or patches with repeated patterns. This is a difficult task to solve and often requires more computation during pre-processing and post-processing. One could also argue that the Luo et al.'s failure amongst smooth and patterned patches may stem from the information lost at the combination phase of their siamese networks through inner product instead of concatenation.

Although there exist limitations to their algorithm in its current state, Luo et al. present a novel approach to solving the issue of stereo depth estimation using convolutional neural networks. Luo et al. added to existing work by constructing probability distributions for each patch over all possible disparities, allowing inferences to be drawn between several disparities for a given patch. The team also rethought the traditional way of concatenating the siamese architecture between the right and left images by simply taking the inner product between the two. Luo et al. provided novel and valuable additions to the discussion of stereo image depth estimation and more research must now be done as a result of their article to further improve its performance against edge cases.

3. Solution Approach

The objective of this section is to describe the algorithm that I implemented and explain its rationale. My algorithm applies PatchMatch (1) in the context of disparity estimation using stereo imaging similar to what was proposed by DeepPruner (3). Instead of using PatchMatch twice to narrow the search field of possible disparities, my algorithm only runs PatchMatch once while still including the random search phase to preserve the efficacy of escaping local minima.

3.1. DeepPruner Solution

The DeepPruner algorithm contains several other elements beyond PatchMatch that improve its performance that I was unable to implement due to hardware constraints. The full DeepPruner algorithm, shown in Figure 4, consists of a feature extraction layer, pruning through differentiable patch match, and cost aggregate refinement phases that chained together for end-to-end learning. I will first explain how the solution approach to the full DeepPruner algorithm before talking specifically about my approach.

Feature Extraction The DeepPruner algorithm starts with a 2D-CNN linked to a spatial pyramid pooling module (SPP). The SPP utilizes shared parameters to create a feature volume representation, or "patch," for every pixel in the left image and the right image. By using residual blocks in their 2D-CNN and a SPP layer, the pre-processing phase can extract global feature information without losing resolution. The design of this CNN pre-processing phase is out of the scope of this paper but was modeled off of the following papers (8; 2; 7; 14).

Differentiable Patch Match As discussed in Section 2.1, PatchMatch was a pivotal algorithm used to quickly locate approximate near neighbor patches within an image or between two images. However, PatchMatch as it was originally presented in 2006 relies on many random offset probes used in the random search phase which can be computationally burdensome, and it determines which offsets to push forward by utilizing the non-differentiable *arg max* algorithm.

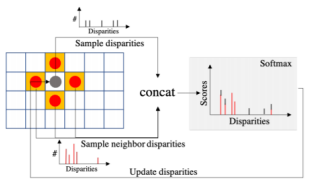

Figure 1. **Overview:** DeepPruner's differentiable PatchMatch (3)

DeepPruner presents a differentiable version of PatchMatch that removes the random search phase and replaces the *arg max* function with a soft version (8) to maintain differentiability. Also, in the propagation phase instead of alternating propagation from top left to the bottom right and from the bottom right to top left, the differentiable PatchMatch algorithm presented by Duggal et al. utilizes one-hot filter banks for all four surrounding pixels (Figure 1).

Confidence Range Prediction The confidence range prediction phase of the DeepPruner algorithm exploits a convolutional encoder-decoder network to narrow the search range of likely disparities for each patch. The network takes the density estimations from the initialization and propagation phase of the differentiable patch match layer and produces a confidence range of disparities, $R_i = [l_i, u_i]$, for each pixel i.

3D Cost Aggregation / Refinement Using the reduced disparity range produced by the confidence range prediction layer, a 3D cost analysis is done once with the soft *arg max* algorithm to evaluate the best disparity. In the refining layer, the best disparity values for each pixel are passed into a "lightweight fully convolutional refinement network" (3). This refining phase improves the performance of the algorithm by considering the global information formed by the feature extraction layer. The refinement layer serves to reduce noise within objects that may be smooth or contain repeated patterns.

3.2. My Solution

The first assumption I made was that the stereo images being processed have been rectified through pre-processing to aligned objects along the vertical plane. This important assumption greatly reduces the cost volume of my algorithm by eliminating the y-axis component, thus reducing the disparity range to a 3D array along the same row of any given pixel.

Disparities for each pixel in the left image are chosen by locating the cost-minimizing disparity between a 7x7 pixel patch surrounding the pixel and corresponding pixel patches in the right image within the disparity range. For example, the cost distance function, $f(A, B)$, between patch A in the left image and patch B in the right image is calculated by doing a simple SSE between the pixel information in each of the two patches.

Initialization In the initialization phase I start by running a convolutional pooling algorithm on each image to initialize what will be considered the feature patch for each pixel. Then, for each patch in the left image, a random offset between zero and the maximum disparity uniformly.

Propagation For each iteration, good guesses from the initialization or random search phase are shared with the patch's right and bottom neighbors (or top and left neighbors for even iterations) during the propagation phase. All three disparities are translated onto the right image patches and run through the SSE cost function with the left patch. The *arg min* function is used to assign the new lowest-cost disparity.

Random Search The random search phase operates similarly to the propagation phase, but instead of using neighboring offsets a random offset is selected and compared to the current offset within a decreasing radius around the current best disparity. For this algorithm, I started with a radius of 100 that would be halved after each random search. After running several tests, however, I came to the same conclusion as Duggal et al. (3), that the random search

phase is computationally burdensome and that most patches converge without it. Instead of removing the random search phase entirely, as Duggal et al., I still preserved one random search across the entire disparity range of 100 for each iteration to cut down on runtime while still maintaining the ability to escape local minima.

4. Implementation

The following will describe in detail how I implemented my stereo depth estimation algorithm using differentiable PatchMatch. The algorithm was written in Python 3.8 and utilizes Torch 1.3.0+cpu, Numpy 1.17.3, and Matplotlib 3.0.2. The algorithm begins by pooling each image into pixel patches and initializing random offsets for each patch in the left image. Then during each of the k iterations of the training phase, a nested loop will perform propagation and random search for every patch in the left image.

Initialization The convolutional pooling algorithm described in the previous section converts an image represented by a 3D numpy array to a 5D numpy array containing 7x7 pools of the pixels surrounding each image pixel. This is done through iterating through every pixel in the image with nested loops and pooling its neighboring pixels.

The function used to initialize the offsets returns a numpy array of the same height and width of the right and left images with random offsets uniformly distributed between zero and the maximum disparity. Offsets are initialized via the numpy random module using the *randint* function. The array used to cache the current cost or distance values for each patch's offset is also initialized at the beginning and calculated using the cost function.

Propagation The propagation phase is implemented for each pixel by applying the accepted offset for the patch above and the patch to the left and comparing the two costs to the currently accepted offset for the pixel. Numpy's *arg max* function is called to determine which of the three offsets would remain for the patch. By considering accepted offsets above and to the left of a given patch, good offsets are effectively propagated down and to the right.

For odd iterations, the pixels are looped through in reverse order, from the bottom right to the top left. This gives the chance for good offsets in the lower right corners of objects to be propagated up throughout the object. To implement this function the propagate patch function is split into two separate functions for propagating up and propagating down for even and odd iterations respectively.

Random Search As mentioned in Section 4, for the random search phase I randomly chose offsets for each pixel patch distributed uniformly across the entire disparity range. I initialized random offsets by calling the same function used to initialize the offsets at the beginning of the program outside of the patch loop to cut down on the number of times numpy *randint* was called. For each patch, its currently accepted offset's cost was compared to the new randomly selected offset and *arg max* was used again to select between the two.

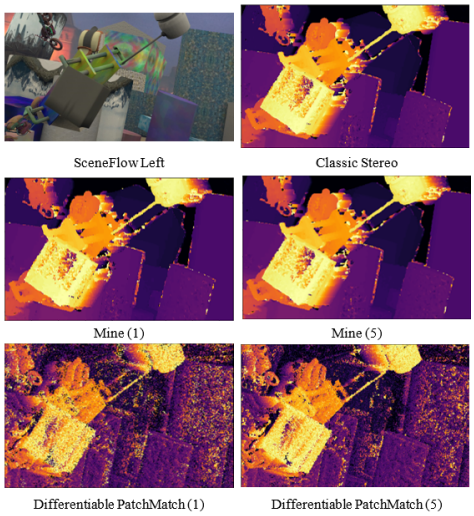

Figure 2. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1 and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1 and 5 iterations on SceneFlow sample (10)

		Driving			Flying Objects		
		0001.png	0002.png	0003.png	1001.png	1002.png	1003.png
PM	1 iter	27.922	45.469	46.391	30.281	51.391	60.672
	2 iter	46.719	100.109	86.219	54.328	86.500	99.672
	5 iter	110.563	190.625	181.688	178.438	201.203	226.875
DPM	1 iter	18.281	28.516	27.719	25.750	25.625	31.000
	2 iter	37.547	50.641	49.469	51.375	52.719	60.109
	5 iter	114.063	126.953	128.281	129.078	133.828	144.266
Classical		1044.828	1224.375	1320.578	840.594	931.609	547.375

Figure 3. Runtimes (in seconds) of all Disparity mappings of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on the SceneFlow Samples (10)

5. Evaluation / Experiments

The goal of my algorithm was to implement the notable PatchMatch (1) algorithm across a pair of stereo images, similar to the work done in DeepPruner (3). What makes the PatchMatch algorithm significant is its ability to prune the search space of possible disparities for a given patch through its random search phase and its ability to propagate good guesses to neighboring patches. By reducing the search range of disparities, the runtime can be greatly reduced, with varying error rates based on the number of iterations performed by the algorithm.

To test the efficacy of my algorithm I implemented three

similar algorithms, the classic stereo algorithm that would perform a cost-minimization over the entire range of possible disparities for each pixel, my algorithm that would scan the range of possible disparities using the PatchMatch algorithm described in Section 4, and the Differentiable PatchMatch (DPM) algorithm presented by Duggal et al. I tested all three models against six sample stereo pairs from the *SceneFlow* (10) sample dataset. Runtimes were compared using the built-in Python Time module and the disparity maps were compared visually (Figure 2).

It did not make sense to compare my algorithm's runtime with any previous works or the full DeepPruner algorithm because not only was my algorithm a greatly simplified version of DeepPruner, but my algorithm was also being tested on a CPU with strict memory limitations. In the future, more extensive testing can be done with improved hardware and more components of the DeepPruner design integrated into my model.

The results of my experiment showed that for a set of images from the *SceneFlow* (10) sample dataset, my algorithm was able to greatly outperform the classical stereo depth algorithm in terms of runtime while quickly converging for the majority of patches within the first few iterations. Although the differential PatchMatch model outperformed my algorithm's runtime, from visual inspection it is clear that the disparity estimates produced by my algorithm significantly outperform those of the differential PatchMatch model. Average runtimes can be seen in Table 3 and the disparity maps are shown in Figure 2.

6. Discussion

As shown in Section 5, the results of my experiments have shown that PatchMatch (1) has the potential to significantly reduce runtime for stereo depth estimating algorithms by grossly pruning the search space for potential disparities. However, it is clear from visual inspection that using PatchMatch as a stand-alone tool to estimate disparities would produce error rates much higher than current state-of-the-art algorithms such as DeepPruner (3). In this section, I will go over how Duggal et al. expanded upon the PatchMatch algorithm with DeepPruner and how these expansions could have improved upon the results I saw in my experiments.

To begin, DeepPruner implemented a differentiable version of PatchMatch that allowed their model to implement end-to-end learning. Duggal et al. took advantage of the large and diverse *SceneFlow* (10) and *KITTI 2015* (5) datasets to train their model with a cost function that compared their disparities directly against known disparity values. Doing so gave DeepPruner the ability to gain insights beyond just the simple SSE between neighboring pixels.

Another significant advantage to DeepPruner is the feature extraction network layer implemented on both images during the pre-processing phase. There are several points within all of the disparity maps produced in Section 5 (Figure 2) where there seem to be discontinuous disparity estimations on a single surface. The disparity estimations on the flat surface of the box in the front of the scene are clear examples of this. Had the patches of this area of the image had the global context of knowing that they all existed within the same feature, an element of voting or averaging could be done between patches to result in a more continuous estimation.

Finally, Duggal et al. used their version of DPM, as the name DeepPruner implies, like a pruning method for narrowing the search field of likely disparity values. However, the DeepPruner algorithm later performs a 3D cost aggregation across the entirety of these disparity ranges that allows for a finer tuned output. This would explain why the DMM model that I tested performed so poorly compared to my model. A lot of the error seen in in the DMM model can likely be explained by the removal of the random search phase necessary to escape local minima.

7. Conclusion

The purpose of this paper was to review a few methods of generating disparity maps between sets of stereo images and to evaluate the effectiveness of using PatchMatch (1) to prune disparity ranges between stereo images. When compared to the classic stereo algorithm approach of searching over all possible disparities, my algorithm implementing PatchMatch was able to cut down runtime by a magnitude of 5 without any significant decrease in disparity estimation.

Although the results of my experiments showed much promise for the use of PatchMatch into stereo depth estimating algorithms to cut down on runtime, more research still needs to be done to implement and test the feasibility of integrating PatchMatch into various forms of deep learning algorithms such as DeepPruner (3).

As a means of estimating approximate near-neighbor patches PatchMatch is an effective tool, but apart from a feature extraction network or other forms of pre- and post-processing PatchMatch is too coarse of a method for mapping disparities compared to state-of-the-art algorithms such as those presented by DeepPruner and Luo et al. (9). A possible next step to this paper could be the integration of a spatial pyramid pooling network as a pre-processing step attached to Differentiable PatchMatch to create an end-to-end differentiable network that may take advantage of the learning properties of backward propagation. Doing so may reduce error rate amongst cases of texture-less surfaces and repeated patterns by providing each pixel patch with a more global scope.

References

[1] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), 2009. 1749 citations.

[2] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[3] Shivam Duggal, Shenlong Wang, Wei-Chiu Ma, Rui Hu, and Raquel Urtasun. Deeppruner: Learning efficient stereo matching via differentiable patchmatch. *International Conference on Computer Vision (ICCV)*, pages 4321–4330, 2019. 0 citations.

[4] Nils Einecke and Julian Eggert. A multi-block-matching approach for stereo. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 585–592. IEEE, 2015.

[5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[6] Fatma Guney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4165–4175, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[8] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017.

[9] Wenjie Luo, Alexander G. Schwing, and Raquel Urtasun. Efficient deep learning for stereo matching. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5695–5703, 2016. 291 citations.

[10] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016.

[11] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Real-time self-adaptive deep stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 195–204, 2019.

[12] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *European Conference on Computer Vision*, pages 756–771. Springer, 2014.

[13] Jure Zbontar, Yann LeCun, et al. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2, 2016.

[14] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

8. Appendix

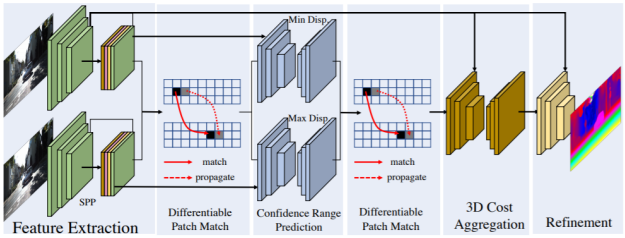

Figure 4. High-level overview of DeepPruner (3)

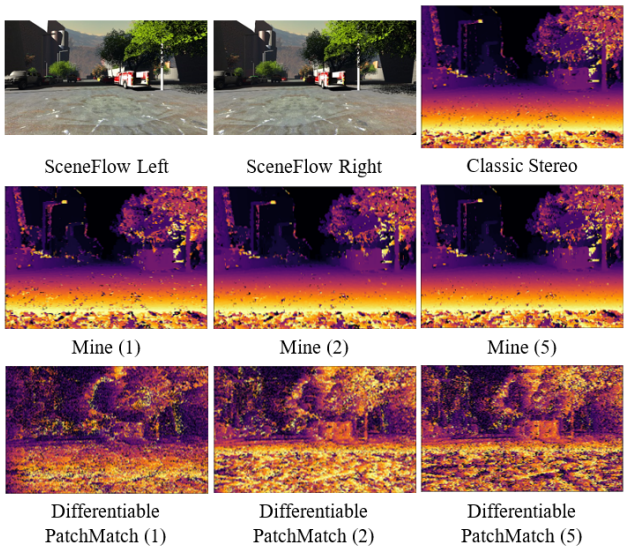

Figure 5. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on SceneFlow Driving Sample 0001.png (10)

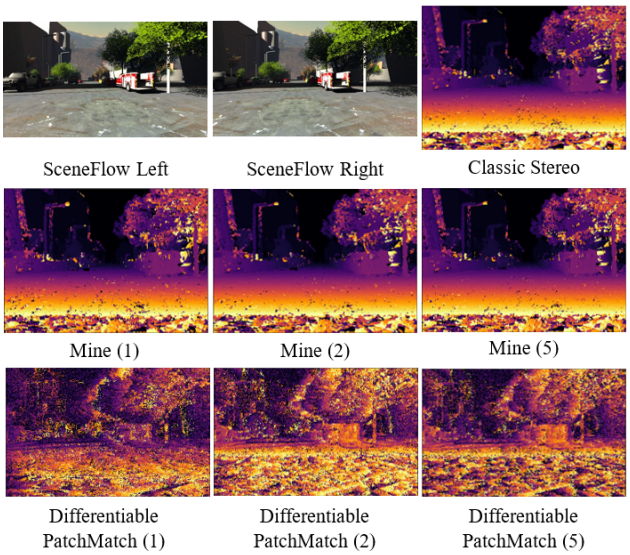

Figure 6. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on SceneFlow Driving Sample 0002.png (10)

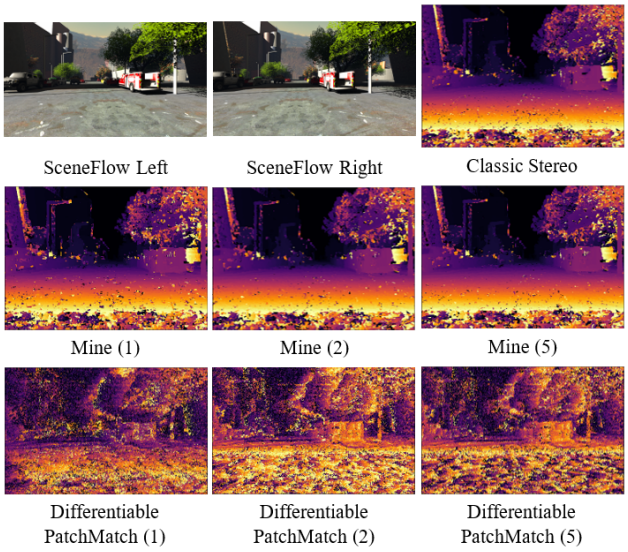

Figure 7. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on SceneFlow Driving Sample 0003.png (10)

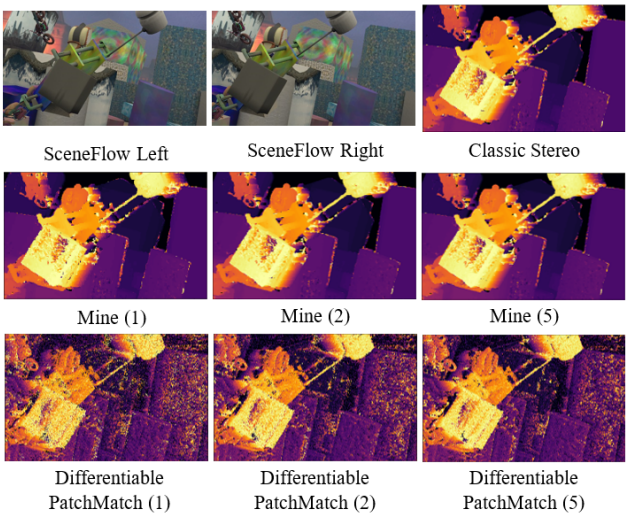

Figure 8. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on SceneFlow Floating Objects Sample 1001.png (10)

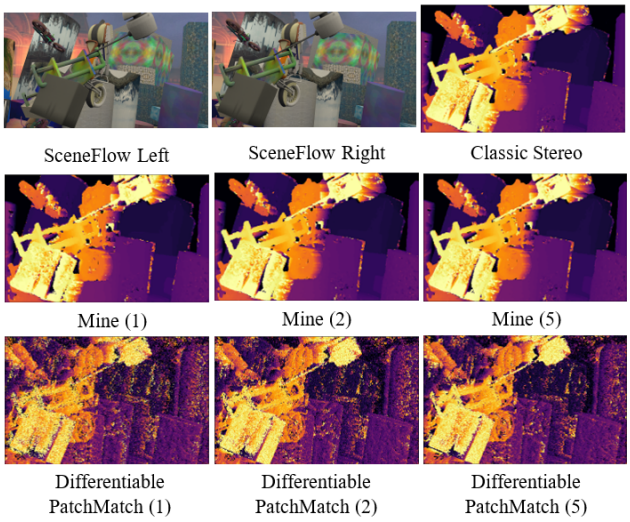

Figure 9. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on SceneFlow Floating Objects Sample 1002.png (10)

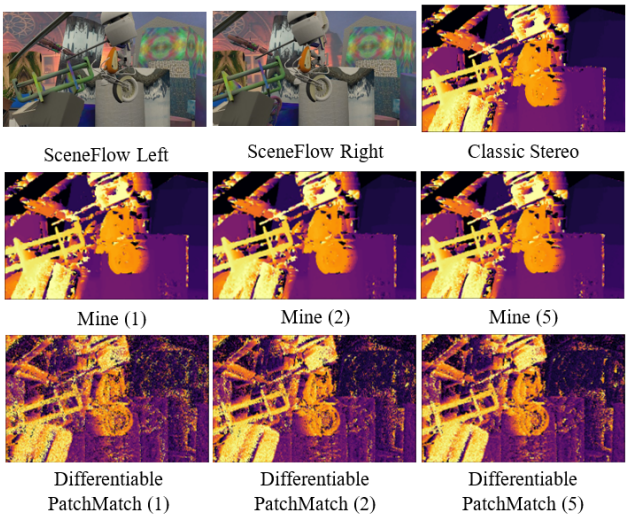

Figure 10. Disparity mapping of the Classical Stereo algorithm, my algorithm after 1, 2, and 5 iterations, and DeepPruner's Differentiable PatchMatch algorithm after 1, 2, and 5 iterations on SceneFlow Floating Objects Sample 1003.png (10)