CMSC733 Project 4

Sigurthor Bjorgvinsson
School of Computer Science
University of Maryland, College Park
Email: Sigurthor.Bjorgvinsson@gmail.com

Ameya Patil
School of Computer Science
University of Maryland, College Park
Email: ameyap@cs.umd.edu

Jack Rasiel
School of Computer Science
University of Maryland, College Park
Email: jrasiel@cs.umd.edu

I. Overview

In this project we explored an unsupervised deep learning approach to retrieve depth and Ego-Motion from motion. We started with the code published with [1] and tried to improve it. To improve it we implemented augmentations to get better use from the provided small training dataset. We added new loss components derived from the literature we reviewed. We tested each change on its own and later tested combined changes. This is our story:

II. Methods

A. Data Augmentation

Here we discuss what augmentations we applied to the data. If the augmentations were turned on, there was a default 50% chance of the augmentation being applied to the batch.

1) Color Changes: Inspired by [2], we implemented gamma, brightness and color shift. The gamma was implemented so that every channel in the batch put into the power of a random value $[0.5, 1.5)$. The brightness factor was also the same over the channels where each channel was multiplied by $[0.5, 2.0)$. The color shift was different for each channel for the purpose of training the new loss components than only the photometric loss.

2) Gaussian Noise: We implemented Gaussian noise where we generated random values in the range $[-0.02, 0.02]$ (image values ranged from $[0, 1]$). We used a truncated normal distribution with standard deviation of 0.01 and a mean of 0 to generate these random values which we then added to the image. We made sure to clip the values afterwards to only have values $[0, 1]$.

3) Flips and Rotation: We saw flips, both vertical and horizontal, be a great simple way to include more diversity in our training data. In addition to flips, we implemented rotation. Because of the high aspect ratio of the input images, using a static scaler to fit all rotations would need to be little more than 3.4. To us, this defeated the purpose of this augmentation because of how much information we would loose from each image. Instead, after randomly picking a angle between $0 - 360\,\mathrm{deg}$, we calculated the scale we would need to apply to the image so that our center crop would have no black background.

B. Loss Function

Here, we explain the various components of our loss function.

The loss function in SfM-Learner ([1]) includes 3 components: a pixel loss, a smoothness loss, and an explainability loss. In our system, we **exclude the explainability loss**, as it was found in [1] to have little impact on performance (indeed, Zhou et al. disabled it in later revisions of their code).

1) Photometric Loss: The photometric loss (or pixel loss) measures the high-level (low frequency components) similarity of the target and warped source images. [1] uses a simple L1 norm for their photometric loss. This loss helps ensure a good match of the low frequency components in the two images. However, L1 loss may introduce blur in the images, because it minimizes the aggregate of individual pixel-level differences irrespective of whether they belong to edges or not.

Our pixel loss is a weighted sum of an L1 loss (from [1]), and an advanced photometric loss SSIM, from [2].

$$L = \alpha\left(\frac{1 - SSIM(I_t, \hat{I}_s)}{2}\right) + (1-\alpha)||I_t - \hat{I}_s||_1$$

We added a **structural similarity metric** (**SSIM**) to this loss, as was done in [2]. This performs the same function as the L1 loss, but is more efficient in doing so because it measures differences in the structure of the image as perceived by humans, and not pixel level differences [3]. The SSIM metric has separate components for luminance masking, contrast masking, and image structure, which are computed using the mean, variance and covariance of pixel values in distinct image sub-patches. Measuring within small patches helps preserve the image's high-level structure better.

2) Smoothness Loss: In SfM-Learner, the smoothness loss is simply the second order derivative of the predicted depth image. This is equivalent to removing the rapid variations of intensity in the image and keeping only the slow variations.

The smoothness loss as implemented in [2] is **edge-aware**. It uses a dot product between the gradients of the depth image, and exponential inverse of the gradients of the actual image: a weighted sum of the depth image gradients.

$$L = \sum_{pixels\ p} |\nabla D(p)|.(e^{-|\nabla I(p)|})^T$$

If an image region is smooth, the weight is identity (no change); if the region has high intensity variations, the weight is less than 1 (low weight). This lowers the contribution to loss from depth image gradients, in regions where the actual image gradients are also high, and keeps the loss contribution

from depth image gradients, in smooth regions in the actual image, intact.

Intuitively: the regions of the image with highly varying colors are more likely to have objects in them at varying depth– and thus are expected to have high depth gradients. So, these regions need not be penalized, and thus their contribution to the loss is lowered.

3) Image Gradient loss: We also incorporated another loss from LEGO [4] which is simply the L1 difference between the image gradients of the target image and the inverse warped source image.

$$L = \sum_{pixels\ p} ||\nabla I_t(p) - \nabla \hat{I}_s(p)||_1$$

This loss further aids in the reconstruction loss, helping to maintain a high level match of the two images.

4) Depth-Normal consistency smoothness loss: LEGO [4] also implemented a depth to normal and normal to depth consistency loss. This basically tries to derive one from the other. They further applied the smoothness loss on both the predicted depth from normal and the predicted normal from depth too. The original Sfm Learner was not implemented to predict normals, but the LEGO implementation helped in improving the accuracy for depth map, so we incorporated the depth2normal and normal2depth layers and applied the smoothness losses to both of them.

5) Edge loss: LEGO [4] implemented one more loss called the edge loss which is specifically for computing edges along with the depth and normals. This involved adding an entirely new network - edgeNet to the existing Sfm learner. However, since edges are not needed for camera pose estimation, we did not incorporate the edgeNet in our code.

C. Training

Our training hyperparameters are unchanged from SfM-Learner. We trained for 20000 iterations total, with 64 epochs, and a batch size of 4. Each training session took about 4.5 hours on a PC with an RTX2080 and i7-8700K. We saved models periodically during training, and saved the model with the lowest training loss.

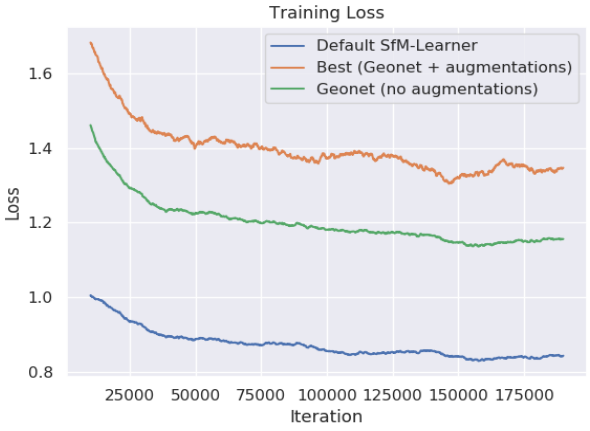
For an example of training loss, see II-C. SfM-Learner and GeoNet losses arent directly comparable because the additional term in the Geonet increases its baseline value. Notice also that adding augmentations makes training noisier and increases loss, but ultimately yields better performance (as shown in the Results section).

III. Results

We conducted a wide range of experiments, with varying loss functions and data augmentations.

A. Performance

We ran a series of ablation tests to demonstrate how each modification effects performance. For a full comparison of performance accross all experiments, see table V.



B. Overall Performance

The best performance was achieved with Geonet loss and Color augmentation (applied to 100% of training images). See III and II for the full metrics.

Overall, we achieved a 1% improvement over the default SfM-Learner. Perhaps more impressively, the highest accuracy category jumped by over 10%. Pose performance was identical.

C. Analysis: Data Augmentation

Table I shows results for the effect of different augmentations on final performance. Our rotation-and-cropping augmentation markedly decreased performance. We attribute this to the significant difference in appearance that this causes to the training images.

The other forms of augmentation uniformly increased performance. Increasing the rate of augmentation (from 50% of training images to 100%) improved performance. Note, however, that adding together multiple augmentations (e.g. color + flipping + gaussian) caused a drop in performance, even if each augmentation boosted performance on its own.

D. Output Comparison and Analysis

1) High and low-error examples:: in the supplement, we show three of the best-performing images, and three of the worst (measured by absolute relative error). As we can see, and as we might intuit, easier scenes are those with even lighting and an open road, and those which are of typical scenarios in the dataset. The best-performing images also tend to have cars towards the edge of the image, rather than in the middle. Harder scenes tend to be more crowded, especially cities with more pedestrians, or have varied lighting, or are of atypical scenarios (e.g. the scene in the third row).

2) Examples Relevant to Data Augmentation: also in the supplement, we show examples where one augmentation performs much better than another. In the first example, the color-augmented model does a much better job reconstructing the tree and pole. We could understand this as being because it is better at distinguishing image features in the low-contrast shadows. The middle two examples show where vertical and

TABLE I

DATA AUGMENTATION. Performance effect of various augmentations. All experiments were on SfM-Learner with Geonet loss. All augmentations applied with probability .5, except where otherwise noted.

	ERROR (lower is better)				ACCURACY (higher is better)			
Augmentation	abs_rel	sq_rel	RMSE	RMSE log	δ<1.25	δ<1.25^2	δ<1.25^3	
Color (p=1)	0.1815	1.573	6.6459	0.2694	0	0.7361	0.9043	0.9599
Flip	0.1831	1.589	6.376	0.2681	0	0.7346	0.9025	0.9585
Color	0.1848	1.7828	6.5579	0.2687	0	0.7406	0.9083	0.961
Gauss	0.1906	1.8348	6.5245	0.2691	0	0.7402	0.9053	0.9597
None	0.1946	2.4433	6.8793	0.2818	0	0.7401	0.9025	0.9561
All	0.3825	4.4081	11.4417	0.5372	0	0.3553	0.6335	0.8019
Rotation	0.4429	4.7569	12.0832	0.5876	0	0.3033	0.5608	0.7662

TABLE II

RESULTS - ACC. Depth performance on the KITTI test dataset, trained on the P4 training data. Legend: **SfM-Learner** - the original network. **Geonet**: without augmentations. **Geonet+Aug.**: With color augmentations (100% rate).

Exp.	ERROR (lower is better)				ACCURACY (higher is better)		
	abs_rel	sq_rel	RMSE	RMSE log	δ<1.25	δ<1.25^2	δ<1.25^3
SfM-Learner	0.2584	4.0676	8.0707	0.3356	0.6515	0.8635	0.9356
Geonet (No aug.)	0.194	2.443	6.879	0.281	0.740	0.902	0.956
Geonet + Aug.	0.181	1.57	6.645	0.269	0.736	0.904	0.9599

TABLE III

RESULTS - ACC. Pose performance on the P4 test sequences, trained on the P4 training data. Legend: **SfM-Learner** - the original network. **Geonet**: without augmentations. **Geonet+Aug.**: With color augmentations (100% rate).

Exp.	Sequence 09		Sequence 10	
	ATE Mean	ATE Std.	ATE Mean	ATE Std.
SfM-Learner	0.018	0.006	0.01	0.0085
Geonet (No aug.)	0.019	0.006	0.014	0.0089
Geonet + Aug.	0.018	0.006	0.014	0.008

TABLE IV

LOSS FUNCTIONS. (Without additional data Augmentations.) Legend: **Exp.** - Experiment . **G**. - Geonet loss. **L** - Lego loss.

Exp.	Loss		ERROR (lower is better)				ACCURACY (higher is better)		
	G	L	abs_rel	sq_rel	RMSE	RMSE log	δ<1.25	δ<1.25^2	δ<1.25^3
8	1	1	0.4429	4.7569	12.0832	0.5876	0.3033	0.5608	0.7662

horizontal flipping improved performance. The last example shows a degenerate case for the full-rate color augmentation.

E. Comparison images and 3D Reconstruction

Figure III-E shows a reconstruction of a novel scene, very unlike the testing dataset. Fig III-E shows the corresponding depth map. As we can see, the model does a reasonable job of reconstructing the basic structure of the orignal scene (shown in III-E).

F. Challenges and Future Work

Performance using the LEGO ([4]) loss is given in table V. We were unable to make this loss converge to a useful value. Further refinement is left as future work.

We would also like to further investigate the effect of augmentations on training performance.

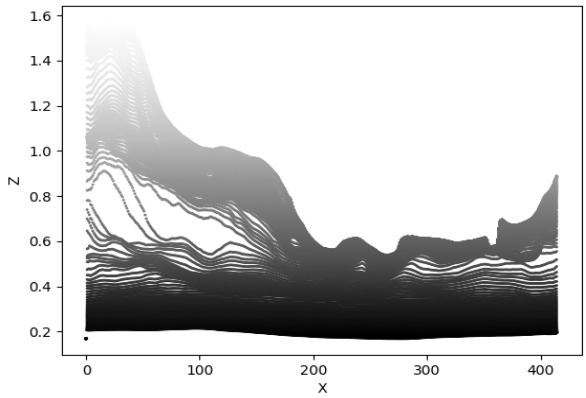


Fig. 1. here's a figure

REFERENCES

[1] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1851–1858, 2017.

[2] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proceedings of the IEEE Conference on*

TABLE V

RESULTS OF ALL EXPERIMENTS. Legend: **Exp.** - Experiment . **G**. - Geonet loss. **L** - Lego loss, **C** - Color augmentation, **R** - Rotation, **F** - Horizontal and Vertical flipping, **G** - Gaussian augmentation.

Exp.	Loss		Data Aug.			ERROR (lower is better)				ACCURACY (higher is better)			
	G	L	C	R	F	G	abs_rel	sq_rel	RMSE	RMSE log	δ<1.25	δ<1.25^2	δ<1.25^3
1	0	0	0	0	0	0	0.2584	4.0676	8.0707	0.3356	0.6515	0.8635	0.9356
2	0	0	1	0	0	0	0.4429	4.7571	12.0832	0.5876	0.3033	0.5607	0.7662
3	1	0	0	0	0	0	0.1946	2.4433	6.8793	0.2818	0.7401	0.9025	0.9561
8	1	1	0	0	0	0	0.4429	4.7569	12.0832	0.5876	0.3033	0.5608	0.7662
9	1	0	1	1	1	1	0.3825	4.4081	11.4417	0.5372	0.3553	0.6335	0.8019
10	1	1	1	1	1	1	0.4429	4.7569	12.0832	0.5876	0.3033	0.5608	0.7662
11	0	0	1	1	1	1	0.4962	5.8093	11.947	0.6358	0.2732	0.5133	0.7041
12	1	0	0	1	0	0	0.4429	4.7569	12.0832	0.5876	0.3033	0.5608	0.7662
13	1	1	0	0	0	0	0.4429	4.7569	12.0832	0.5876	0.3033	0.5608	0.7662
14	1	0	0	0	0	1	0.1906	1.8348	6.5245	0.2691	0.7402	0.9053	0.9597
15	1	0	1	0	0	0	0.1848	1.7828	6.5579	0.2687	0.7406	0.9083	0.961
16	1	0	0	0	1	0	0.1831	1.589	6.376	0.2681	0.7346	0.9025	0.9585
17	1	0	1	0	0	0	0.1815	1.573	6.6459	0.2694	0.7361	0.9043	0.9599

TABLE VI

RESULTS OF ALL EXPERIMENTS (POSE) - ACC.

Exp.	Loss		Data Aug.			Sequence 09			Sequence 10	
	G	L	C	R	F	G	ATE Mean	ATE Std.	ATE Mean	ATE Std.
1	0	0	0	0	0	0	0.0181	0.0064	0.014	0.0085
2	0	0	1	0	0	0	0.6707	0.152	0.4699	0.2065
3	1	0	0	0	0	0	0.0191	0.0062	0.0146	0.0089
8	1	1	0	0	0	0	0.6795	0.1562	0.4693	0.2017
9	1	0	1	1	1	1	0.4795	0.1319	0.3791	0.1608
10	1	1	1	1	1	1	0.0878	0.0443	0.0522	0.0347
11	0	0	1	1	1	1	0.1777	0.0988	0.1461	0.0958
12	1	0	0	1	0	0	0.6741	0.1483	0.4657	0.1961
13	1	1	0	0	0	0	0.6597	0.1528	0.4515	0.1968
14	1	0	0	0	0	1	0.0194	0.0067	0.0148	0.0094
15	1	0	1	0	0	0	0.0181	0.0063	0.0142	0.0089
16	1	0	0	0	1	0	0.0271	0.0163	0.0211	0.0141
17	1	0	1	0	0	0	0.0182	0.0063	0.0146	0.0092
18	1	0	1	0	1	0	0.0607	0.0342	0.0421	0.031
19	1	0	1	0	1	1	0.0561	0.0307	0.0384	0.028

Fig. 2. here's a figure

Fig. 3. here's a figure

Computer Vision and Pattern Recognition, pp. 1983–1992, 2018.

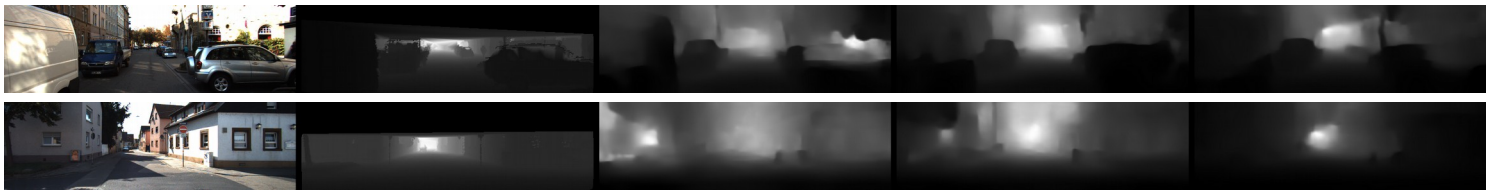
[3] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[4] Z. Yang, P. Wang, Y. Wang, W. Xu, and R. Nevatia, "Lego: Learning edge with geometry all at once by watching videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 225–234, 2018.

***Supplement**: Example Output*

Color augmentation > flipping:

Flipping augmentation > color:



Degenerate case for color:

Lowest-error examples:

Highest-error examples: