Supplementary Material for Learning to Zoom and Unzoom

Chittesh Thavamani[1] Mengtian Li[†1] Francesco Ferroni[‡2] Deva Ramanan[1]
[1]Carnegie Mellon University [2]Argo AI
tchittesh@gmail.com mtli@cs.cmu.edu fferroni@nvidia.com deva@cs.cmu.edu

A. Appendix

A.1. Bilinear Transformations

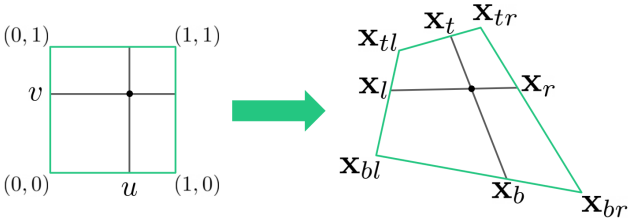

Figure 1. Geometric interpretation of bilinear transformations. Suppose we have such a transformation from the unit square to an arbitrary quadrilateral. Given coordinates (u, v) in the unit square, if we draw lines in the quadrilateral such that $u = \frac{|\mathbf{x}_b\mathbf{x}_{bl}|}{|\mathbf{x}_{br}\mathbf{x}_{bl}|} = \frac{|\mathbf{x}_t\mathbf{x}_{tl}|}{|\mathbf{x}_{tr}\mathbf{x}_{tl}|}$ and $v = \frac{|\mathbf{x}_l\mathbf{x}_{bl}|}{|\mathbf{x}_{tl}\mathbf{x}_{bl}|} = \frac{|\mathbf{x}_r\mathbf{x}_{br}|}{|\mathbf{x}_{tr}\mathbf{x}_{br}|}$, they will intersect at $\text{BilinearTransformation}(u, v)$.

Our construction in Section 4.1 assumes prior knowledge of bilinear transformations. Bilinear transformations have actually been widely studied in the context of computer graphics [10]. Here, for unfamiliar readers, we outline its definition and inverse formulation.

For simplicity, consider a bilinear transformation that maps the unit square to the quadrilateral with corners $\mathbf{x}_{bl}, \mathbf{x}_{br}, \mathbf{x}_{tl}, \mathbf{x}_{tr}$. True to its name, the transformation is defined *within* the square via bilinear interpolation:

$$\begin{aligned}\text{BilinearTransformation}(u, v) = \mathbf{x}_{bl} + (\mathbf{x}_{br} - \mathbf{x}_{bl})u \\ +(\mathbf{x}_{tl} - \mathbf{x}_{bl})v + (\mathbf{x}_{tr} - \mathbf{x}_{br} - \mathbf{x}_{tl} + \mathbf{x}_{bl})uv. \quad (1)\end{aligned}$$

Interestingly, this transformation also has a geometric interpretation, shown in Figure 1.

Now, consider the inverse of this mapping. Given a point (x, y) in the quadrilateral, we want to find the point (u, v) in the unit square that maps to it. A full derivation is given in [10], but if we define the following scalars

$$(a_0, b_0) = \mathbf{x}_{bl} \quad (2)$$
$$(a_1, b_1) = \mathbf{x}_{br} - \mathbf{x}_{bl} \quad (3)$$
$$(a_2, b_2) = \mathbf{x}_{tl} - \mathbf{x}_{bl} \quad (4)$$
$$(a_3, b_3) = \mathbf{x}_{tr} - \mathbf{x}_{br} - \mathbf{x}_{tl} + \mathbf{x}_{bl} \quad (5)$$
$$c_0 = a_1(b_0 - y) + b_1(x - a_0) \quad (6)$$
$$c_1 = a_3(b_0 - y) + b_3(x - a_0) + a_2b_1 - a_2b_1 \quad (7)$$
$$c_2 = a_3b_2 - a_2b_3, \quad (8)$$

then the solution (u, v) must satisfy

$$c_2v^2 + c_1v + c_0 = 0 \quad (9)$$

and

$$u = \frac{x - a_0 - a_2v}{a_1 + a_3v}. \quad (10)$$

Applying the quadratic formula on Equation 9, we can solve for v. Then, we can substitute into Equation 10 to find u. Given a point (x, y) in the quadrilateral, this will produce exactly one pair of solutions (u, v) in the unit square (there may be extraneous solutions with u or v negative or greater than 1).

Although these results assume a mapping from the unit square, they extend naturally to our use case. Recall from Section 4.1 that $\widetilde{\mathcal{T}}_{ij}$ is a bilinear transformation from rectangle R_{ij} to quadrilateral $\mathcal{T}[R_{ij}]$. We can apply all previous results, simply by normalizing the coordinates within R_{ij}.

A.2. Efficient Inversion of Nonseparable Warps

In Section 4.2, we detail how to efficiently invert separable zooms $\mathcal{T}_{\text{LZ,sep}}$. To invert nonseparable zooms $\mathcal{T}_{\text{LZ}}$, it no longer suffices to invert each axis. We must instead reason in the full 2D space.

Suppose we have a nonseparable zoom $\mathcal{T}_{\text{LZ}}$. We compute $\mathcal{T}_{\text{LZ}}[\text{Grid}(h, w)]$ for small h, w and use this to approximate the forward zoom as $\widetilde{\mathcal{T}}_{\text{LZ}}$, an $(h-1)\times(w-1)$ piecewise tiling of bilinear maps. Now, to unzoom to a desired output resolution of $H'' \times W''$, we must evaluate $\widetilde{\mathcal{T}}_{\text{LZ}}[\text{Grid}(H'', W'')]$. That is, for each $\mathbf{x} \in \text{Grid}(H'', W'')$, we must determine which of the $(h-1)(w-1)$ quadrilateral pieces it falls in

[†]Now at Waymo.
[‡]Now at Nvidia.

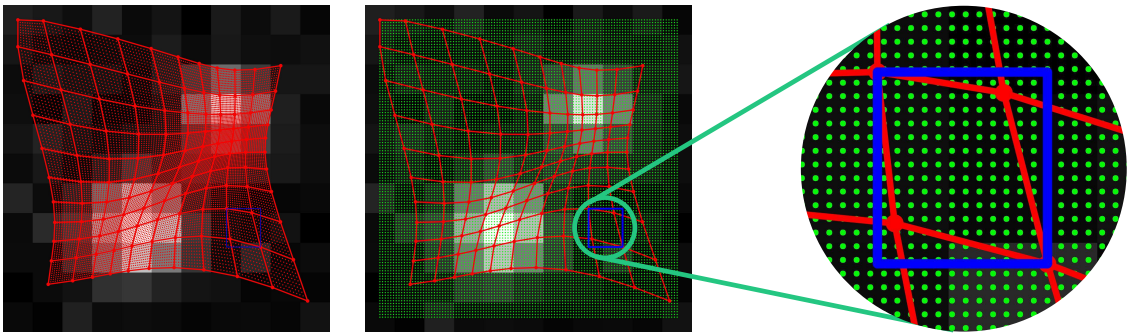

Figure 2. Unzooming in the nonseparable case. $\mathcal{T}_{\text{LZ}}$ is approximated as $\widetilde{\mathcal{T}}_{\text{LZ}}$, which is a $(h-1) \times (w-1)$ tiling of bilinear transformations (left). We wish to compute $\widetilde{\mathcal{T}}_{\text{LZ}}^{-1}(\mathbf{x})$ at each green point $\mathbf{x} \in \text{Grid}(H'', W'')$, where $H'' \times W''$ is the desired output size (middle). For the ij-th tile, we consider the set of all candidate green point $\mathbf{x}$ within the enclosing blue box (right) and apply the corresponding inverse bilinear transformation. We set $\widetilde{\mathcal{T}}_{\text{LZ}}^{-1}(\mathbf{x}) = \text{BilinearTransformation}_{ij}^{-1}(\mathbf{x})$ only if it falls in the ij-th grid rectangle $R(i,j)$.

Algorithm 1 Inverting nonseparable zooms $\mathcal{T}_{\text{LZ}}$.

In practice, we make the following optimizations. We vectorize the loop on line 13. We also fix B_{ij} to be the max size over choices of (i,j), allowing us to implement the loop on line 4 using batch-processing.

1: ▷ See Appendix A.2 for the algorithm setup and meaning of variables.
2: **function** UNZOOM($\mathcal{T}_{\text{LZ}}[\text{Grid}(h,w)], (H'', W'')$)
3: Initialize $\mathcal{T}_{\text{LZ}}^{-1}(\mathbf{x}) = (0,0)$ for all $\mathbf{x} \in \text{Grid}(H'', W'')$
4: **for** $(i,j) \in [h-1] \times [w-1]$ **do**
5: ▷ corners of $R(i,j)$
6: $\mathbf{x}'_{tl}, \mathbf{x}'_{tr} = \left(\frac{i}{h-1}, \frac{j-1}{w-1}\right), \left(\frac{i}{h-1}, \frac{j}{w-1}\right)$
7: $\mathbf{x}'_{bl}, \mathbf{x}'_{br} = \left(\frac{i}{h-1}, \frac{j-1}{w-1}\right), \left(\frac{i}{h-1}, \frac{j}{w-1}\right)$
8: ▷ corners of ij-th quadrilateral tile
9: $\mathbf{x}_{tl}, \mathbf{x}_{tr} = \mathcal{T}_{\text{LZ}}^{-1}(\mathbf{x}'_{tl}), \mathcal{T}_{\text{LZ}}^{-1}(\mathbf{x}'_{tr})$
10: $\mathbf{x}_{bl}, \mathbf{x}_{br} = \mathcal{T}_{\text{LZ}}^{-1}(\mathbf{x}'_{bl}), \mathcal{T}_{\text{LZ}}^{-1}(\mathbf{x}'_{br})$
11: ▷ top-left and bottom-right corners of rectangle
12: ▷ enclosing the quadrilateral tile
13: $\mathbf{c}_{tl} = \min(\mathbf{x}_{tl}, \mathbf{x}_{tr}, \mathbf{x}_{bl}, \mathbf{x}_{br})$
14: $\mathbf{c}_{br} = \max(\mathbf{x}_{tl}, \mathbf{x}_{tr}, \mathbf{x}_{bl}, \mathbf{x}_{br})$
15: ▷ set of all candidate points in the ij-th tile
16: $B_{ij} = \{\mathbf{x} \in \text{Grid}(H'', W'') : \mathbf{c}_{tl} \le \mathbf{x} \le \mathbf{c}_{br}\}$
17: **for** $\mathbf{x} \in B_{ij}$ **do**
18: $\mathbf{x}' = \text{BilinearTransformation}_{ij}^{-1}(\mathbf{x})$
19: **if** $\mathbf{x}'_{tl} \le \mathbf{x}' \le \mathbf{x}'_{br}$ **then**
20: $\mathcal{T}_{\text{LZ}}^{-1}(\mathbf{x}) = \mathbf{x}'$
21: **return** $\mathcal{T}_{\text{LZ}}^{-1}$

and apply the corresponding inverse bilinear map. Recall from Appendix A.1 that applying an inverse bilinear map amounts to solving a quadratic.

In our actual implementation, we parallelize operations as much as possible. For the ij-th tile, instead of first determining which points $\mathbf{x} \in \text{Grid}(H'', W'')$ are inside of it and then applying the ij-th inverse bilinear map, we implement it the other way around. We consider a set of candidate interior points, apply the ij-th inverse bilinear map to all of them, and keep only those with a valid solution. The candidate points are those falling inside the axis-aligned rectangle enclosing that tile. The full procedure is described in Algorithm 1 and visualized in Figure 2.

Our implementation takes about 12.6ms to invert a nonseparable warp with $(h,w) = (31,51)$ and an output shape $(H'', W'') = (600, 960)$, as in our Argoverse-HD [6] experiments. While this is not fast enough to support favorable accuracy-latency tradeoffs, we believe that further optimization (*e.g.* using custom CUDA operations) may change this.

A.3. Analysis of our Warping Approximations

We make two approximations in our formulation. First, to ensure that the composition of forward and inverse warps is truly the identity function, we use the approximate forward warp $\widetilde{\mathcal{T}}$ in place of the true forward warp $\mathcal{T}$. This trades latency for how well $\widetilde{\mathcal{T}}$ zooms in on the intended regions of interest. See Figure 3 for a visualization of this effect.

Second, we use bilinear downsampling to approximate the inverse at lower resolutions after feature pyramid networks [8]. This approximation is surprisingly effective! For our fixed LZU model on 2D detection, we calculate that the error between $\widetilde{\mathcal{T}}^{-1}$ and the doubly approximated inverse $\widehat{\mathcal{T}}_d^{-1}$ given by

$$\underset{\mathbf{x} \in \text{Grid}(H'/d, W'/d)}{\text{avg}} \left\| \widetilde{\mathcal{T}}^{-1}(\mathbf{x}) - \widehat{\mathcal{T}}_d^{-1}(\mathbf{x}) \right\|_2 \tag{11}$$

is only 0.0274, 0.0065, 0.0028 pixels at $d = 2$, 4, and 8!

A.4. Sensitivity to Saliency

Fixed LZU is quite robust to choice of saliency. Details on how we computed our saliency maps are given in Appendix A.6. For 2D detection, we performed a grid search at 0.5x scale to determine the saliency hyperparameters. The

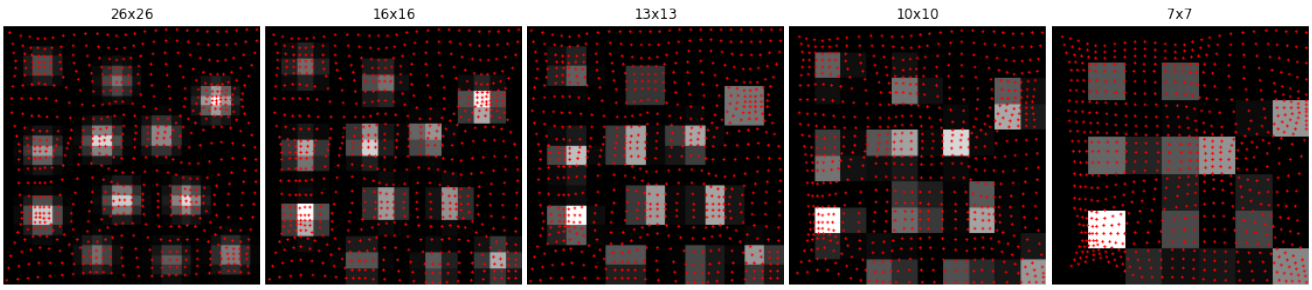

Figure 3. Quality of the approximate forward warp $\widetilde{\mathcal{T}}$ as we decrease the dimensions $h \times w$ of our piecewise bilinear approximation. $\widetilde{\mathcal{T}}$ degrades noticeably once h, w decrease beyond a certain threshold.

LZU, fixed				
	Attraction Kernel fwhm			
Amp.	4	10	16	22
1	34.6	34.8	35.4	35.7
5	34.5	34.7	32.6	34.6
10	34.8	34.4	32.3	34.5
50	35.0	33.5	31.4	33.7
100	35.0	33.8	31.3	33.2

LZU, adaptive				
	Attraction Kernel fwhm			
Amp.	4	10	16	22
1	34.9	36	35.6	35.5
5	34.3	35.1	33.3	35.1
10	33.4	34.3	32.2	34.0
50	33.8	33.2	32.3	34.2
100	34.0	33.6	32.3	33.7

Table 1. Grid search over hyperparameters for 2D object detection (see Appendix A.6) shows that LZU is quite robust to choice of saliency. For comparison, uniform downsampling yields an AP of 32.2.

results show that LZU is quite robust to choice of saliency, as long as the warp is not too strong (see Table 1). For semantic segmentation and 3D detection, we chose saliency one-shot, which suggests that these fixed LZU models are also robust to choice of saliency.

Our grid search suggests that adaptive LZU is also robust to choice of hyperparameters than fixed LZU. However, these saliency hyperparameters, chosen at 0.5x scale, struggle to generalize to 0.75x and 1x scale (see Table 1). As a result, for adaptive warps, it may be necessary to tune saliency at each scale.

A.5. Additional Results

In Figure 4, we plot the spatial accuracy of our CityScapes [5] models.

A.6. Implementation Details

Our experiments are implemented using open-source libraries MMDetection [2], MMSegmentation [4], and MMDetection3D [3], all released under the Apache 2.0 License. We use GeForce RTX 2080 Ti's for training and training, which takes at most 5 GPU-days for any given model, but the precise amount varies by model and task. We perform all timing experiments with a batch size of 1 on a single GPU.

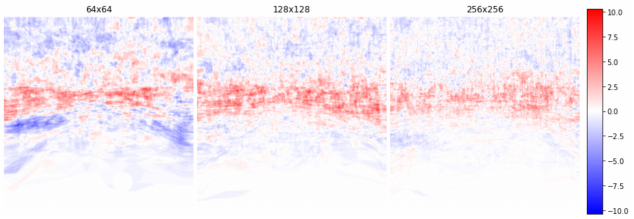

Figure 4. Visualizations of the per-pixel accuracy difference between the uniform and the LZU models at each resolution. The largest gains occur at the central horizontal strip, largely corresponding with the saliency map used for segmentation. However, performance also *decreases* at the top and bottom of the image. Thus, using a fixed saliency map must be conscious decision to trade performance at low saliency regions for high saliency regions. We hypothesize that adaptive saliency maps could lead to more uniform gains.

Argoverse-HD [6] Detection As done in FOVEA [9], for our uniform downsampling experiments, we finetune a COCO-pretrained model for 3 epochs with the random left-right image flip augmentation, a batch size of 8, momentum SGD with a learning rate of 0.005, momentum of 0.9, weight decay of 1e-4, learning rate warmup for 1000 iterations, and a per-iteration linear learning rate decay [7].

For both LZU models, we use a learning rate of 0.01 and keep all other hyperparameters identical to the baseline. To "zoom", we use a 31×51 saliency map and the separable anti-cropping formulation $\mathcal{T}_{\text{LZ,sep,ac}}$ (as proposed in [9] and discussed in Section 3.2).

For the fixed saliency LZU model, we use Gaussian distance kernels k_x and k_y of full-width-half-maximum (fwhm) 22. To generate the fixed saliency map, we use kernel density estimation (KDE) on all training bounding boxes with hyperparameters amplitude $a = 1$ and bandwidth $b = 64$. For details on the effects of a and b, refer to [9].

For the adaptive saliency LZU model, we use Gaussian distance kernels k_x and k_y of fwhm 10. To generate adaptive saliency, we use KDE on detections from the previous frame with $a = 1$ and $b = 64$. When training, to simulate motion,

we jitter bounding boxes by $\mathcal{N}(0, 7.5)$ pixels horizontally and $\mathcal{N}(0, 3)$ pixels vertically.

For each LZU experiment, we run a grid search at 0.5x scale over separable/nonseparable, amplitude $a = 1, 5, 10, 50, 100$, and distance kernel's fwhm $= 4, 10, 16, 22$ to determine optimal settings. This is done using an $80/20$ split of the train set, so as to not overfit on the real validation set. We generate this split such that locations between splits are disjoint. All other hyperparameters are chosen one-shot.

Cityscapes [5] Segmentation We train the uniform downsampling baseline using mostly the default hyperparameters from MMSegmentation [4]. The only changes are to the data augmentation pipeline and the evaluation frequency. Comprehensively, we train with just the photometric distortion augmentation (random adjustments in brightness, contrast, saturation, and hue), a batch size of 16, momentum SGD with a learning rate of 0.01, momentum of 0.9, weight decay of 5e-4, and a polynomial learning rate schedule with power 0.9. To account for overfitting, we validate our performance at 10 equally spaced intervals on a 500-image subset of the training dataset (and train on the others). For all experiments, we train for 80K iterations, with the exception of 64×64, which we train for 20K iterations due to rapid overfitting.

For the fixed LZU model, we use the same training hyperparameters as the baseline. To "zoom", we use the separable anti-cropping formulation $\mathcal{T}_{\mathrm{LZ,sep,ac}}$ with a 45×45 saliency map and Gaussian distance kernels k_x, k_y of fwhm 15. To generate the fixed saliency, we aggregate ground truth semantic boundaries over the train set. Precisely, we define boundaries to be pixels which differ from at least one of its eight neighbors. We compute semantic boundaries for each 256×256 ground truth segmentation, assign boundaries an intensity of 200 and background an intensity of 1, and average pool down to a 45×45 saliency map. The semantic boundary intensity value was chosen qualitatively (for producing a reasonably strong warp) and tested one-shot.

nuScenes [1] 3D Detection We train the uniform downsampling baseline using all default hyperparameters from MMDetection3D [3], except the learning rate, which we reduce for stability. Specifically, we train for 12 epochs with the random left-right flip augmentation, a batch size of 16, momentum SGD with a learning rate of 0.001, momentum of 0.9, weight decay of 1e-4, doubled learning rate on bias parameters with no weight decay, L2 gradient clipping, a step learning rate schedule with drops at epochs 8 and 11, and a linear learning rate warmup for the first 500 iterations.

For the fixed LZU model, we use the same training hyperparameters as the baseline. To "zoom", we use the separable anti-cropping formulation $\mathcal{T}_{\mathrm{LZ,sep,ac}}$ with a 27×48 saliency map and Gaussian distance kernels k_x, k_y of fwhm 10. To generate the fixed saliency, we project 3D bounding boxes into the image plane and reuse the same KDE formulation with the same hyperparameters ($a = 1$ and $b = 64$) as used in 2D detection. These are all chosen and evaluated one-shot.

References

[1] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 4

[2] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 3

[3] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020. 3, 4

[4] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020. 3, 4

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 3, 4

[6] Mengtian Li, Yu-Xiong Wang, and Deva Ramanan. Towards streaming perception. In *European Conference on Computer Vision*, pages 473–488. Springer, 2020. 2, 3

[7] Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. *arXiv preprint arXiv:1905.04753*, 2019. 3

[8] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 2

[9] Chittesh Thavamani, Mengtian Li, Nicolas Cebron, and Deva Ramanan. Fovea: Foveated image magnification for autonomous navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15539–15548, 2021. 3

[10] George Wolberg. *Digital image warping*, volume 10662. IEEE computer society press Los Alamitos, CA, 1990. 1