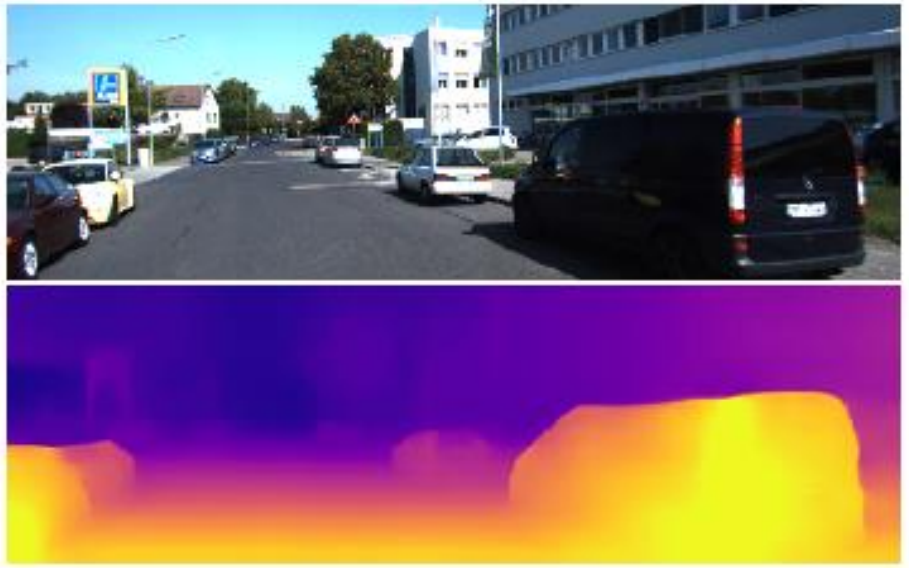
Anıl Öztürk
504181504

DEPTH PREDICTION WITHOUT THE SENSORS: LEVERAGING STRUCTURE FOR UNSUPERVISED LEARNING FROM MONOCULAR VIDEOS

by *Vincent Casser, Soeren Pirk, Reza Mahjourian, Anelia Angelova*

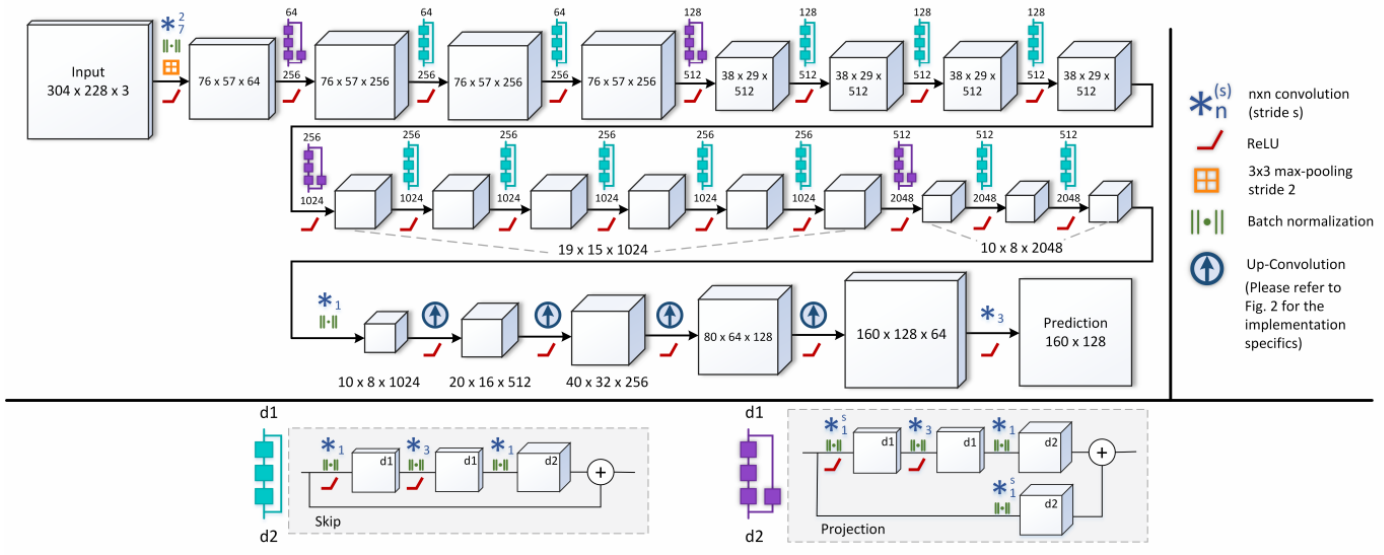
AIM

1. Give an RGB input
2. Run unsupervised methods
3. Get a depth estimation



PREVIOUS METHODS

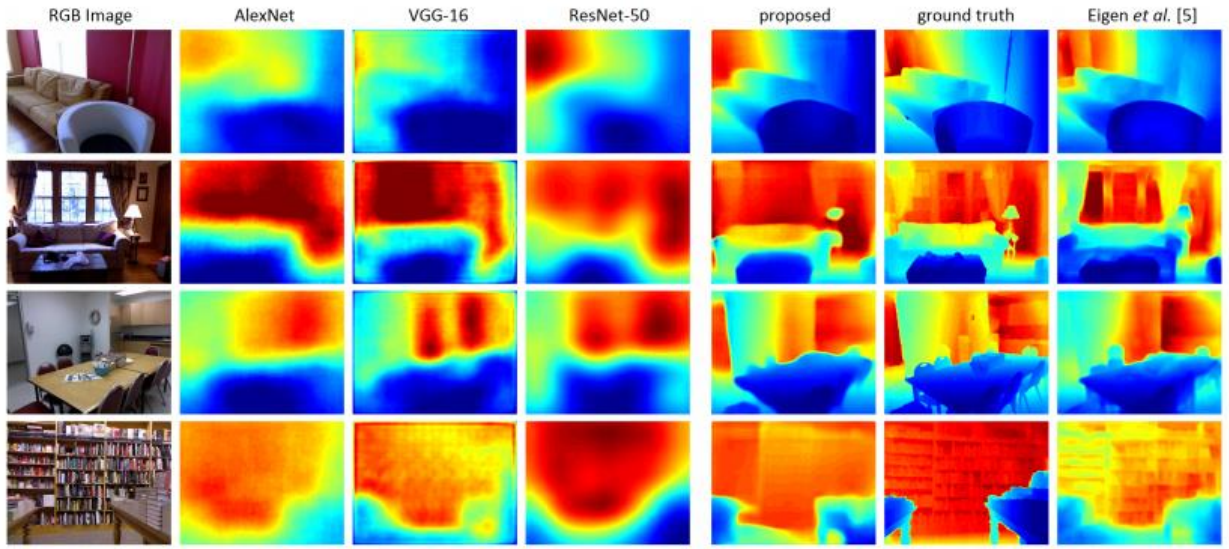
Generate rich features from raw data



Laina et al. 2016

PREVIOUS METHODS

Then, run unsupervised image-to-depth learning without depth or ego-motion supervision



Laina et al. 2016

PREVIOUS METHODS

They can’t handle dynamic scenes well!

Because they **can not explain object motion**.

To catch-up, optical flow models were trained separately.

IMPROVEMENT

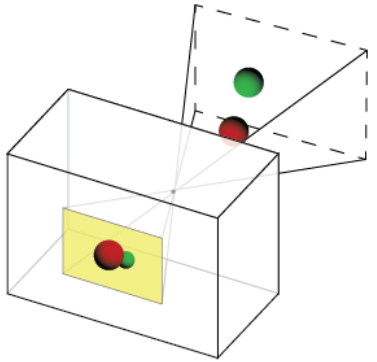
This paper suggests to use a seperate model for dynamic adaptation.

But this time: also with using **geometric structure of the scene** and **modeling all objects' motion *(including camera egomotion)***

PROBLEM SETUP

For the method to run properly, we need;

1. Sequences of at least three RGB images $(I_1, I_2, I_3) \in \mathbb{R}^{H \times W \times 3}$
2. Camera intrinsics matrix $\mathrm{K} \in \mathbb{R}^{3 \times 3}$



PROBLEM SETUP

Depth and camera ego-motion will be estimated by **learning nonlinear functions**. Neural network architecture is the best-like implementation of that kind.

We will have;

A depth function which is a fully convolutional encoder-decoder architecture $\theta: \mathbb{R}^{H\times W\times 3} \rightarrow \mathbb{R}^{H\times W}$

A dense depth map $D_i = \theta(I_i)$

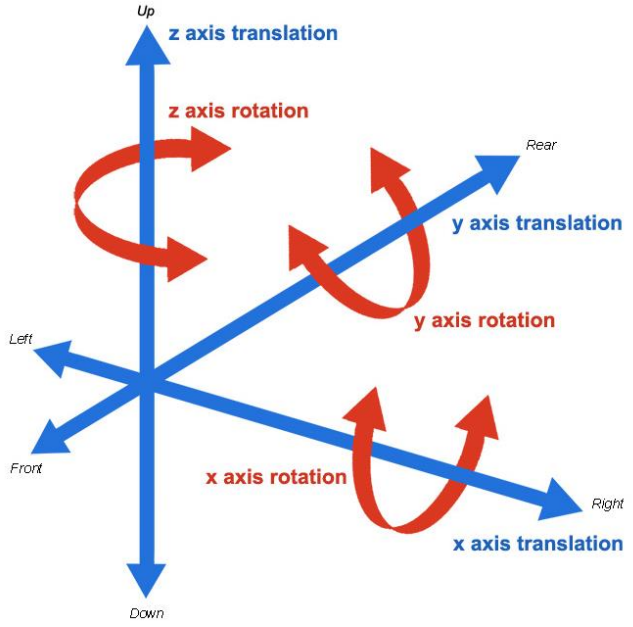
PROBLEM SETUP

Then, of course we need our **camera ego-motion model**. Our ego-motion network $\boldsymbol{\varphi_E}$ will be as follows;

$$\varphi_{\mathrm{E}}: \mathbb{R}^{2\times H\times W\times 3} \rightarrow \mathbb{R}^{6}$$

It will take a sequence of **two RGB image** and give an **output tuple of six values**. The values will represent **6-dimensional transformation vector**;

$E_{1\rightarrow 2}: \varphi_{\mathrm{E}}(I_1, I_2)$ of the form $(t_x, t_y, t_z, r_x, r_y, r_z)$ (translation and rotation values)



PROBLEM SETUP

Since the depth map is available through $\theta(I_i)$, the ego-motion to the next frame φ_E can translate the scene to next frame. This way, we can get predicted-projection of the next frame. We will call that **warping**.

Our warping operation $\emptyset = \left(I_i, D_j, E_{i\to j}\right) \to \hat{I}_{i\to j}$

It takes first RGB image, depth map of the second image, and ego-motion change between this images. Gives a predicted projection of second frame as output.

MOTION MODEL

The paper also offers an object motion model φ_M which has similar architecture with the ego-motion network φ_E. It takes **RGB image sequence** as input, but this time it will be complemented by **pre-computed instance segmentation masks**.

The motion model will be learn to predict the transformation vectors for each object in 3D space. This way, we can create observed object appearance in the target frame.

SEGMENTATION MASK

We define instance-aligned segmentation mask as;

$$(S_{i,1}, S_{i,2}, S_{i,3}) \in \mathbb{N}^{H \times W}$$

for each potential object **i** in our image sequence (I_1, I_2, I_3)

To compute ego-motion, we need to filter out these objects from the images.

SEGMENTATION MASK

To filter out the objects, we define a binary mask for only allowing the static scene;

$$O_0(s) = 1 - \cup_i S_i$$

This will remove all image contents corresponding to potentially moving objects.

$$O_j(s) = S_j, \quad j > 0$$

Thus, this will return only the mask for the object **j**.

SEGMENTATION MASK

The mask $O_0(s)$ will be applied all images in the sequence by element-wise multiplication $\odot$, before feeding the sequence to the ego-motion model.

Our formulas for **object 0** will be;

$$\boldsymbol{V} = O_0(S_1) \odot O_0(S_2) \odot O_0(S_3)$$

$$\boldsymbol{E_{1\to2}}, \boldsymbol{E_{1\to2}} = \varphi_{\mathrm{E}}(I_1 \odot V, I_2 \odot V, I_3 \odot V)$$

OBJECT MOTION

For every object instance, the object motion estimate $M^{(i)}$ of the **i-th object** will be;

$$\boldsymbol{M}_{\boldsymbol{1\rightarrow 2}}^{\boldsymbol{(i)}}, \boldsymbol{M}_{\boldsymbol{2\rightarrow 3}}^{\boldsymbol{(i)}} = \varphi_{\mathrm{M}}\left(\hat{I}_{1\rightarrow 2} \odot O_i\left(\hat{S}_{1\rightarrow 2}\right), I_2 \odot O_i(S_2), \hat{I}_{3\rightarrow 2} \odot O_i\left(\hat{S}_{3\rightarrow 2}\right)\right)$$

$M_{1\rightarrow 2}^{(i)}, M_{2\rightarrow 3}^{(i)} \in \mathbb{R}^6$ represent **object motions**. **They are modeling how the camera would have moved in order to explain object appearance.** To get actual 3D-object-motion vectors, we track the **voxel movements** before and after the object movement transform.

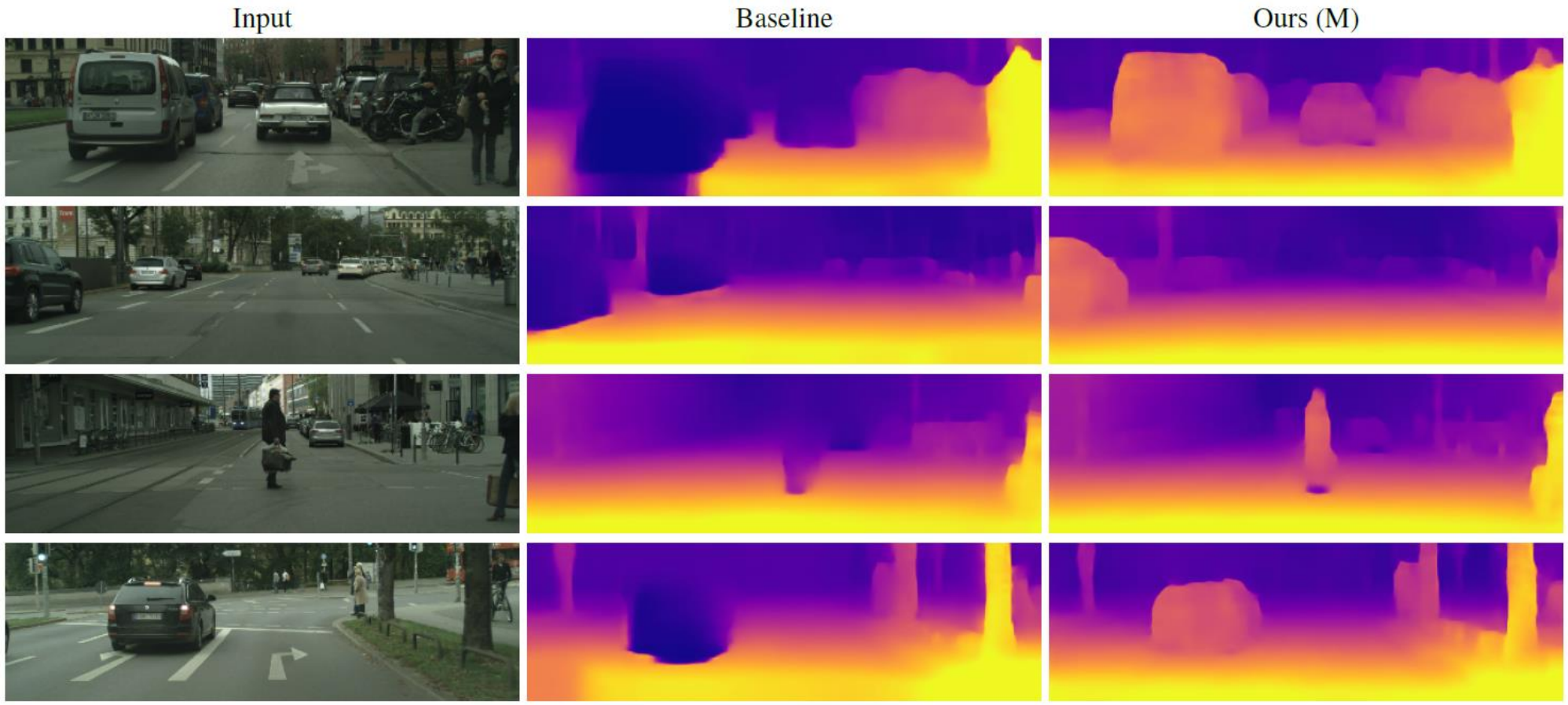
FULL WARPING

After that, **a full warping operation** will be finally executed. **This will move the objects according to the predicted motions.** The warping result is a combination of the individual warping from moving objects' motion $\hat{I}^{(i)}$ and the ego-motion $\hat{I}$.

INFINITE DEPTH

There was a common issue with the previous papers, **the cars with constant relative speed will likely be shifted to the infinite depth**. Because there is not a motion, the network can predict it as a **infinite far away**. **The reprojection error would reduce almost to the zero**.

INFINITE DEPTH PROBLEM



UNCERTAINTY

If the model has no knowledge about object scales, it could explain the same object motion by **placing an object very far away and predicting very significant motion, assuming it to be very large**, or **placing it very close and predicting little motion, assuming it to be very small**.

LEARNING SCALES

Solution is **learn objects' scales** as part of the training process. Assuming a weak prior **on the height of objects**, we can get an approximate depth estimation for them given their segmentation mask and the camera intrinsics using;

$$D_{approx}(p; h) \approx f_y \frac{p}{h}$$

Focal length: $f_y \in \mathbb{R}$

Height prior: $p \in \mathbb{R}$

Height in pixels: $h \in \mathbb{N}$

SCALE LOSS

It's **not safe** that approximating the depth with setting the constraints with hand. **We need to create to a network to tune them.** The loss for training them is below;

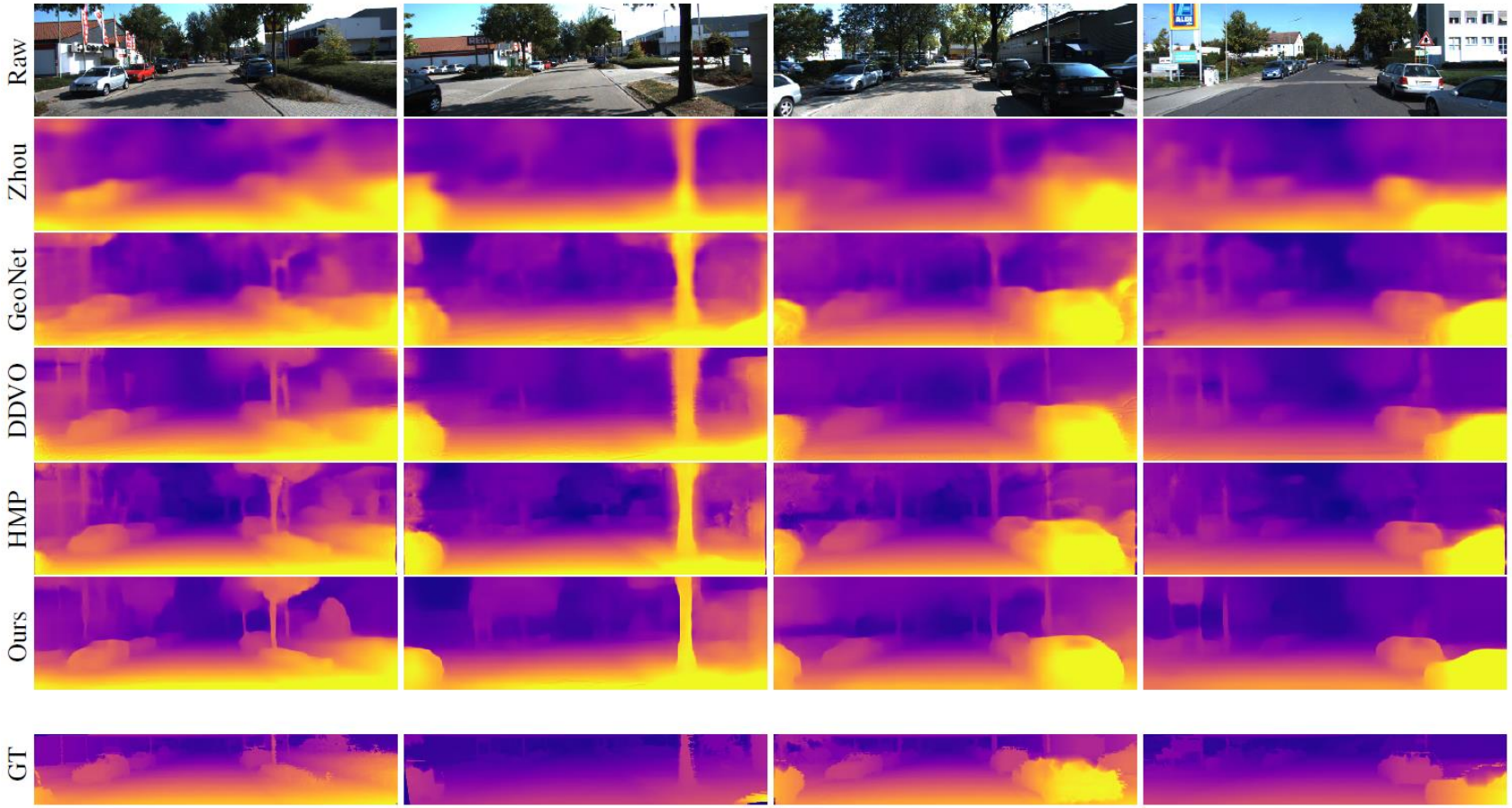
$$L_{sc} = \sum_{i=1}^{N} \left\| \frac{D \odot O_i(S)}{\overline{D}} - \frac{D_{approx}\left(p_{t(i)}; h\left(O_i(S)\right)\right)}{\overline{D}} \right\|$$

Category ID for object **i**: $t(i)$

Learnable height prior for each category **j**: p_j

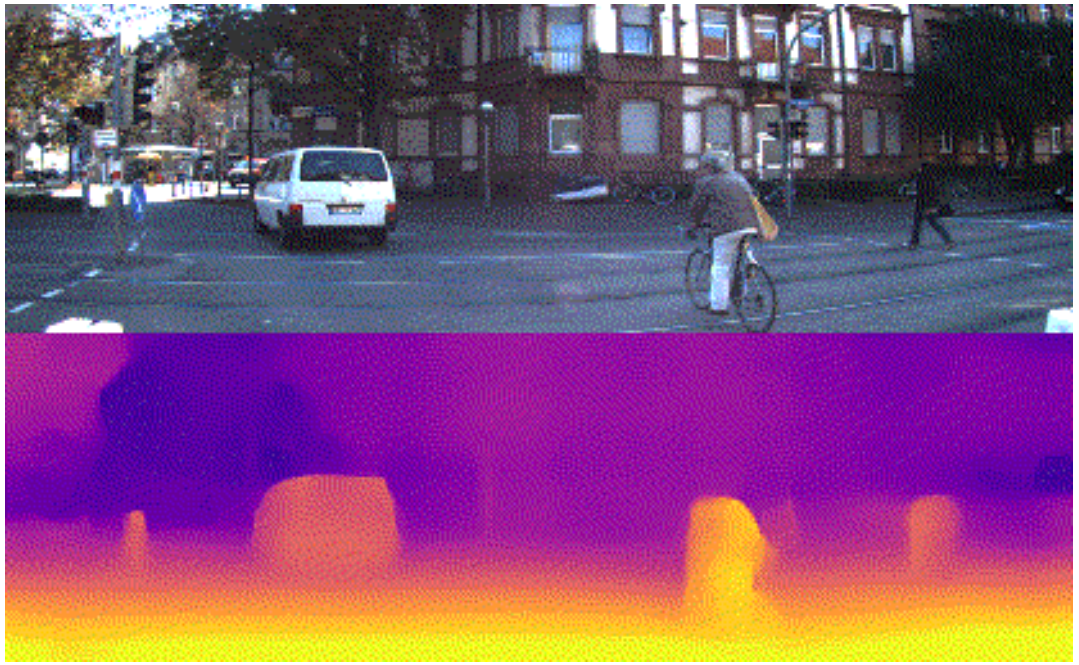
Mean estimated depth of the middle frame: $\overline{D}$

TEST RESULTS



TEST ON LOCAL PC

I have tested the code on a sample KITTI scene with **GTX1080TI**, **Python3.5** and **TensorFlow**. The result is like the following;



THANK YOU

(Q&A) FULL WARPING FORMULA

The full warp operation $\hat{\boldsymbol{I}}_{\mathbf{1\to2}}^{(\boldsymbol{F})}$ will be;

$$\boldsymbol{I}_{\mathbf{1\to2}}^{(\boldsymbol{F})} = \hat{I}_{1\to2} \odot V + \sum_{i=1}^{N} \hat{I}_{1\to2}^{(i)} \odot O_i(S_2)$$

The first part is the gradient w.r.t. φ_E, the second part is the gradient w.r.t. φ_{M}

(Q&A) OBJECT MOTION

To model object motion, we should apply ego-motion estimate to obtain $\hat{I}_{1\to2}, I_2, \hat{I}_{3\to2}$ and $\hat{S}_{1\to2}, S_2, \hat{S}_{3\to2}$. Assuming that depth and ego-motion estimates are correct. So the misalignments will **only be caused by moving objects**.

(Q&A) LOSS FUNCTION

Then, the **photometric loss** applied between projected frame $\hat{I}_{i\to j}$ and actual second frame I_j.

Ex: reconstruction loss $L_{rec} = \left\|\hat{I}_{i\to j} - I_j\right\|$

(Q&A) LOSS FUNCTION

But this paper has a strong loss function from a recent work. *(Zhou et al. 2017; Godard, Aodha, and Brostow 2018)*

Reconstruction loss: $L_{rec} = min(\|\hat{I}_{1\to2} - I_2\|, \|\hat{I}_{3\to2} - I_2\|)$

The loss is computed as the minimum reconstruction between warping from **the previous** and **next** frame.

(Q&A) LOSS FUNCTION

In addition to that loss, the paper also uses **2 more** losses;

SSIM and **depth smoothness loss**;

The final loss formula is; $L = a_1 \sum_{i=0}^{3} L_{rec}^{(i)} + a_2 L_{ssim}^{(i)} + a_3 \frac{1}{2^i} L_{sm}^{(i)}$

The alphas are hyperparameters, so they are **trainable**.