A DEEP NEURAL NETWORK FOR ESTIMATING DEPTH FROM STEREO

Nikolai Smolyanskiy, Alexey Kamenev, Stan Birchfield

NVIDIA

Project Redtail

GTC 2018

AGENDA

Why Deep Learning for depth computation?

Our end-to-end stereo depth DNN

Supervised, unsupervised, semi-supervised training

Our stereo DNN vs mono DNN vs traditional stereo

Inference runtime

Performance metrics

WHY DL FOR DEPTH COMPUTATION?

Accurate depth is needed in

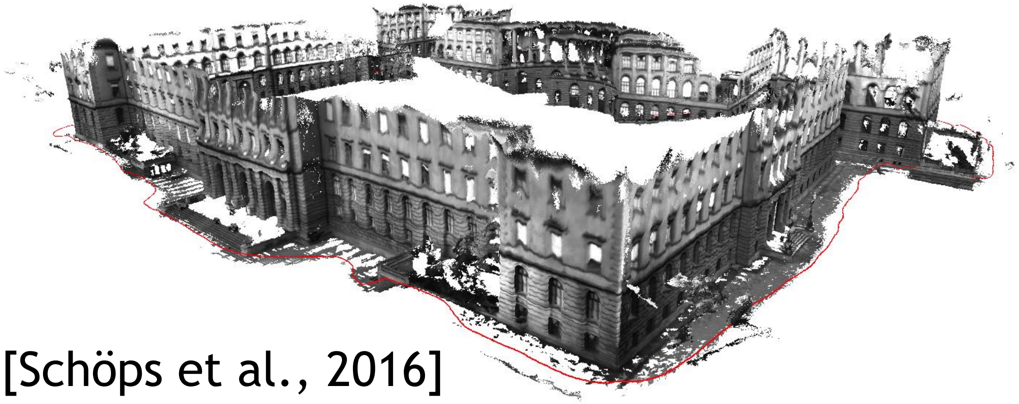
3D reconstruction

Robotic manipulation

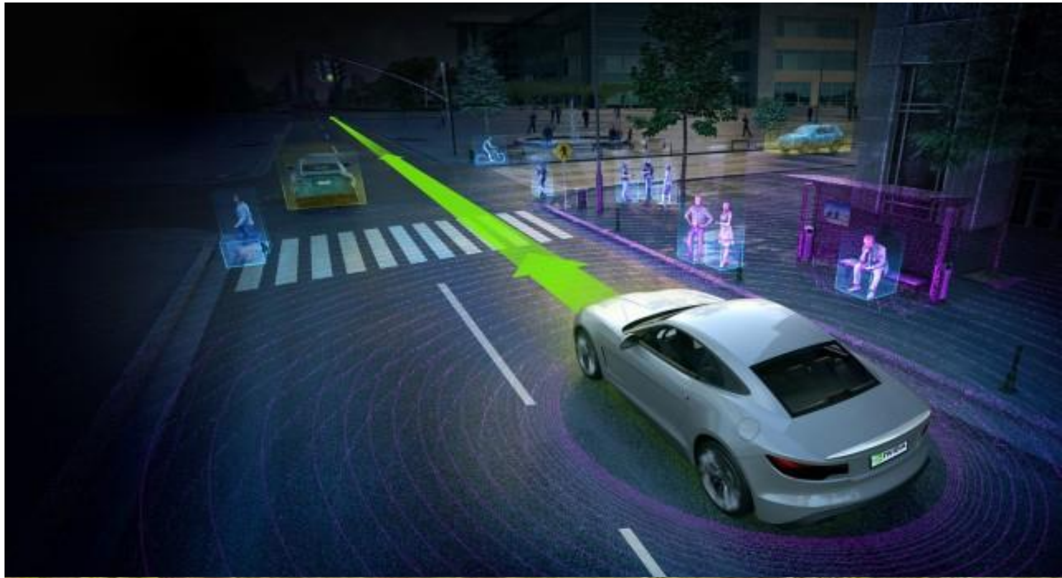
Robotic navigation

Self-driven cars

Augmented reality



[Schöps et al., 2016]



[Smolyanskiy et al., IROS 2017]

WHY DL FOR DEPTH COMPUTATION?

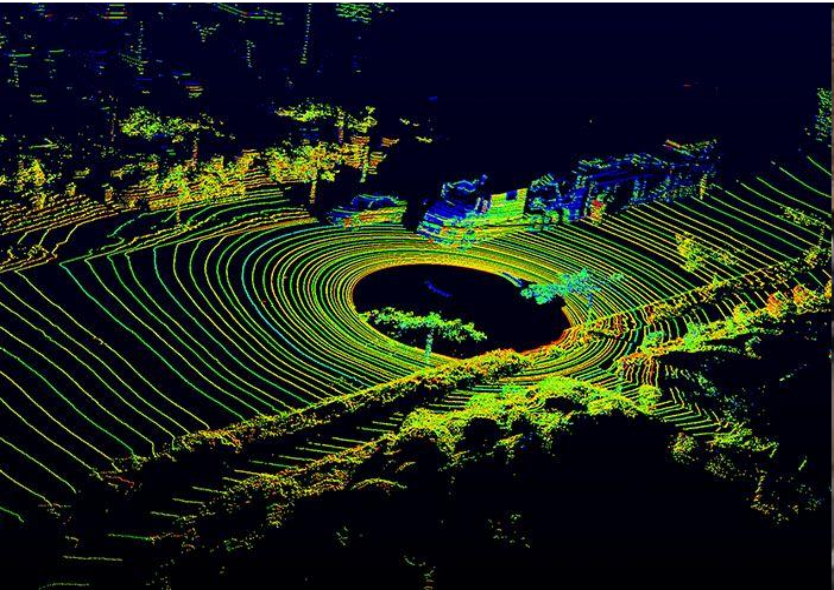
DNNs are more accurate than CV stereo and more practical than Lidars

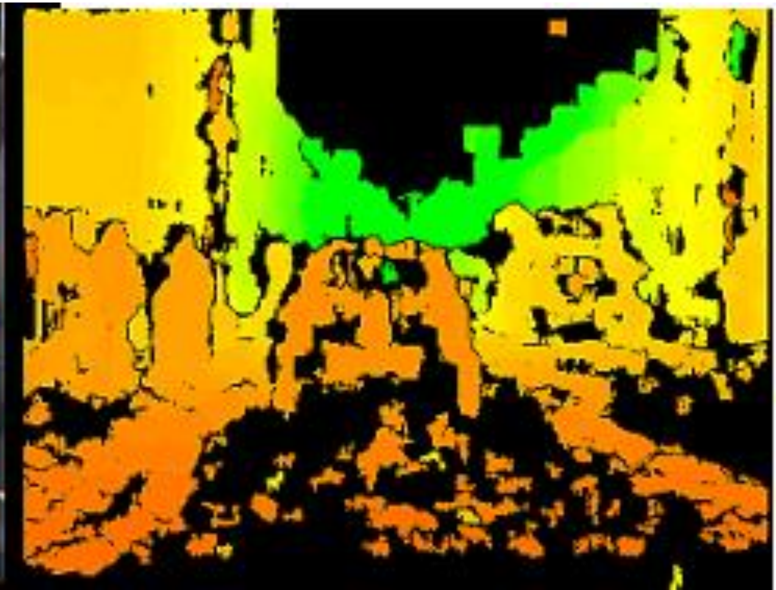
LIDARs are accurate, but bulky, expensive, have narrow angle and run at 10 FPS

Traditional stereo matching methods are inaccurate

- [Mroz and Breckon, 2012] http://breckon.eu/toby/demos/autostereo/

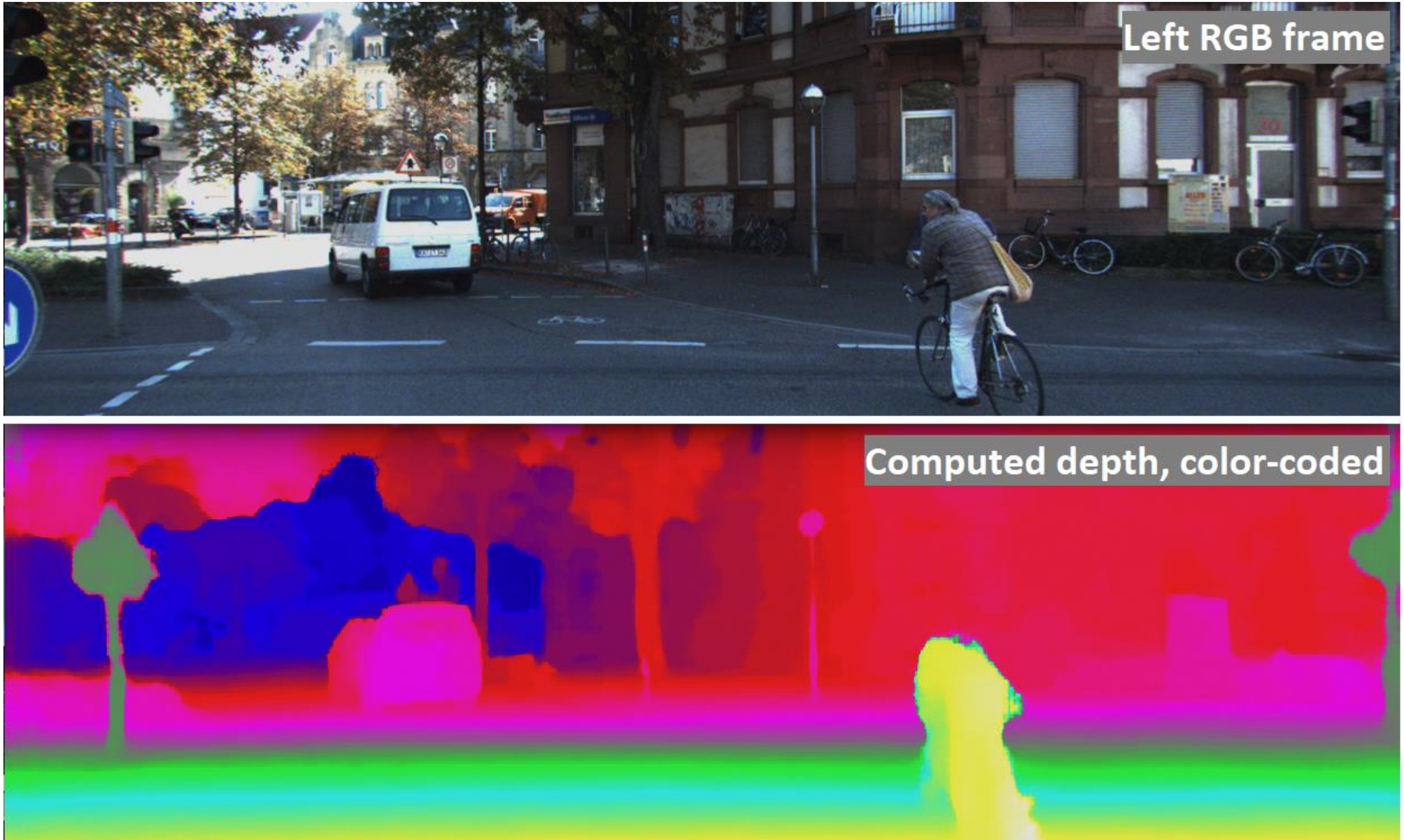
Deep learning methods provide dense accurate depth and are only bound by compute





WHY DL FOR DEPTH COMPUTATION?

Our stereo depth DNN produces accurate and clean depth



OUR DEEP LEARNING APPROACH

We were inspired by 2 DNN architectures

GC-Net: “End-to-End Learning of Geometry and Context for Deep Stereo Regression” [Kendall et al. 2017, Skydio]

Monodepth: “Unsupervised Monocular Depth Estimation with Left-Right Consistency” [Godard et al. 2017]

Our contributions:

- We can train in supervised, unsupervised and semi-supervised modes
- Simpler than GC-Net architecture: we use ELU and no batch norm.
- Novel “machine learned” argmax
- A smaller version runs in near real-time on desktop GPUs
- Our custom inference runtime allows running on Jetson TX2

DEPTH FROM DISPARITY

Short review of stereo methods

Passive stereo techniques compute depth from disparity

Disparity is a distance between corresponding points on epipolar lines

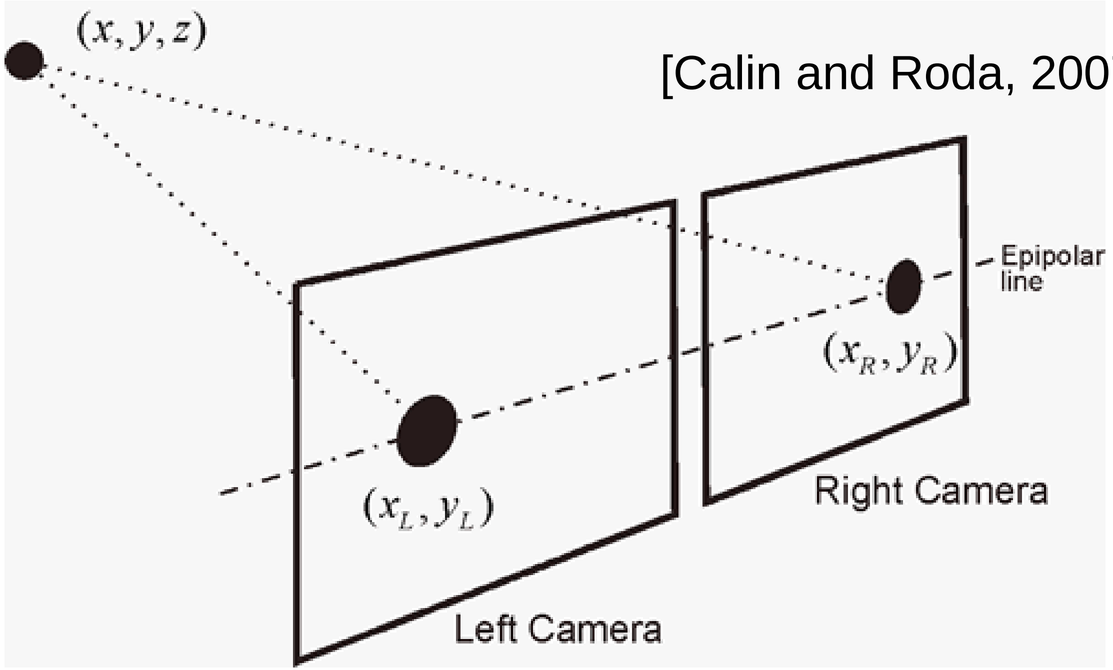
Depth is inversely proportional to disparity:

$$depth = \frac{Bf}{x_L - x_R}$$

$where$:

$B \ - camera\ baseline\ in\ meters$

$f - focal\ length\ in\ pixels$



[Calin and Roda, 2007]

STEREO MATCHING VIA COST VOLUME

Stereo matching via exhaustive search

We can build a 3D volume of match costs for all pixels for all disparities

Then a disparity for a given pixel can be computed as:

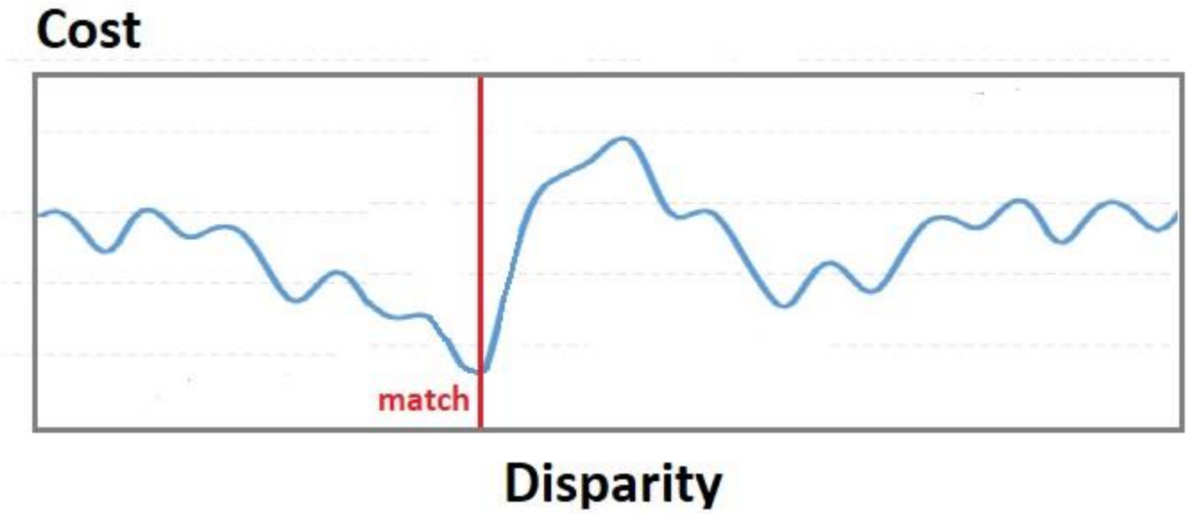
$$disparity = argmax(p_i);\ p\ is\ a\ pdf\ built\ from\ costs\ for\ this\ pixel\ (e.g. via\ softmax)$$

Argmax is not differentiable. We can use “soft-argmax” instead [Kendall et al., 2017]

$$disparity = \sum_i p_i d_i\ ;\ where\ d_i - disparity\ level$$

Soft-argmax is differentiable and

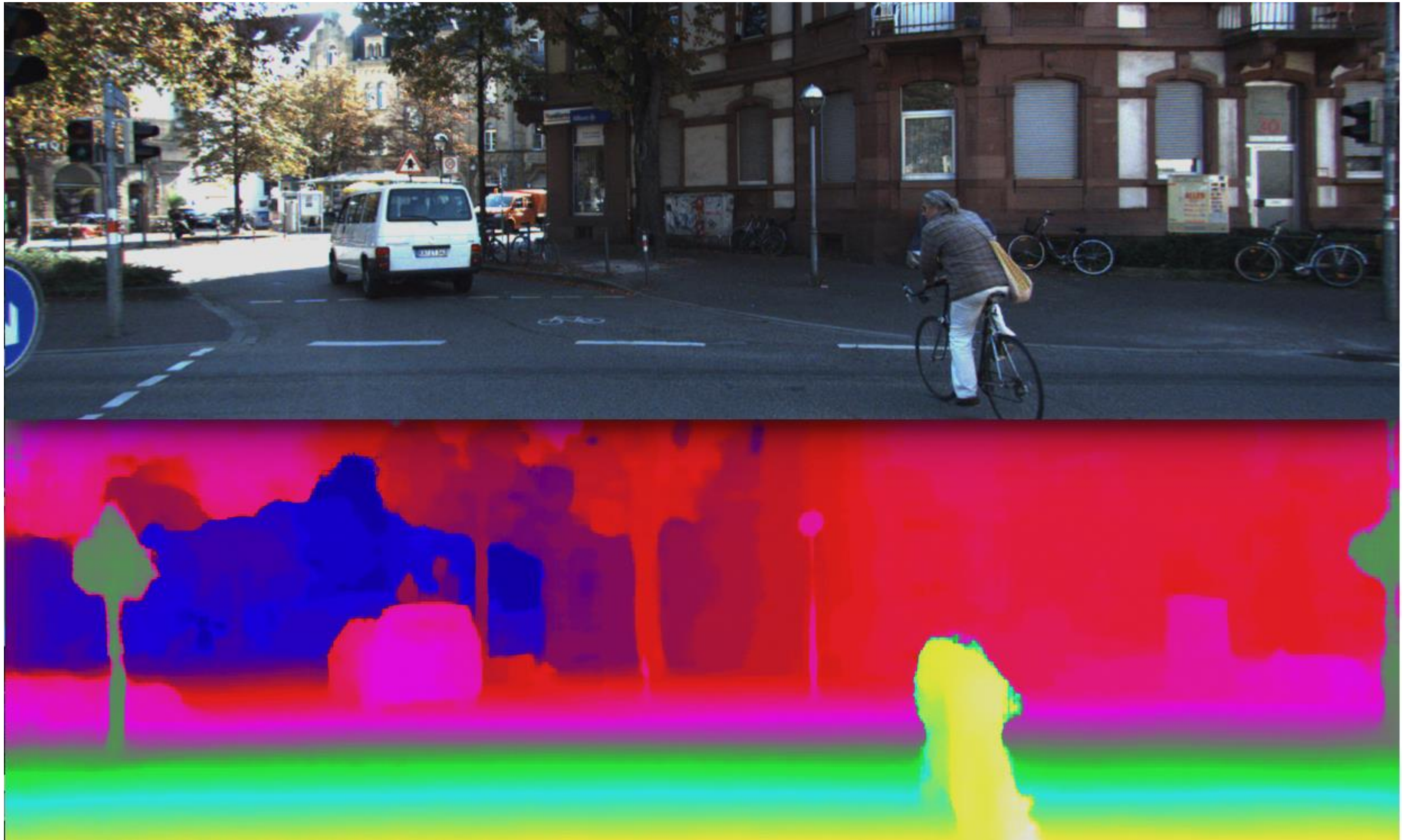
can be used to train DNNs



VIDEO DEMO

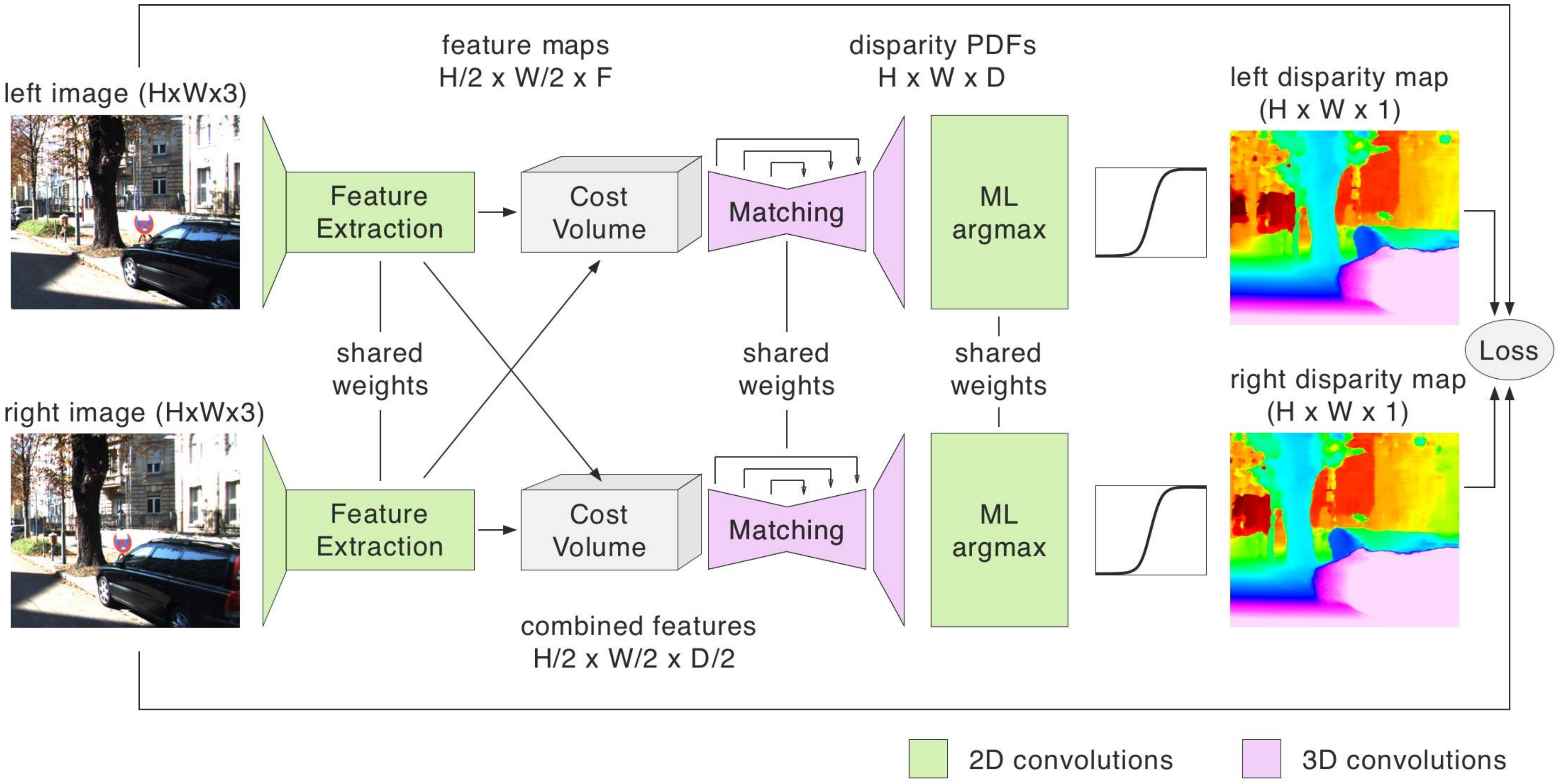
This video demonstrates our stereo DNN depth results

https://youtu.be/0FPQdVOYoAU



STEREO DNN ARCHITECTURE

Our architecture mimics traditional stereo pipeline



DIFFERENT SIZE MODELS

We created several models to test performance

Large: Resnet18 like for 2D input -> cost volume -> 3D encoder/decoder -> soft-argmax

Large: Resnet18 like for 2D input -> cost volume -> 3D encoder/decoder -> ML-argmax

Small: AlexNet as 2D input -> small cost volume -> small 3D encoder/decoder -> soft-argmax

Variations:

- Use 1 tower instead of 2 for training
- Use correlation instead of feature concatenation in cost volume
- Use different constraints in the loss

LOSS FUNCTION

Has unsupervised photometric terms and supervised L2 disparity terms

$$L = \lambda_1 E_{image} + \lambda_2 E_{lidar} + \lambda_3 E_{lr} + \lambda_4 E_{ds}, \qquad (1)$$

where

$$E_{image} = E^l_{image} + E^r_{image} \qquad (2)$$

$$E_{lidar} = |d_l - \bar{d}_l| + |d_r - \bar{d}_r| \qquad (3)$$

$$E_{lr} = \frac{1}{n}\sum_{ij} |d^l_{ij} - \tilde{d}^l_{ij}| + \frac{1}{n}\sum_{ij} |d^r_{ij} - \tilde{d}^r_{ij}| \qquad (4)$$

$$E_{ds} = E^l_{ds} + E^r_{ds} \qquad (5)$$

LOSS FUNCTION

Continued

$$E^l_{image} = \frac{1}{n}\sum_{i,j}\alpha\frac{1 - SSIM(I^l_{ij}, \tilde{I}^l_{ij})}{2} + (1-\alpha)|I^l_{ij} - \tilde{I}^l_{ij}|$$

$$E^l_{ds} = \frac{1}{n}\sum_{i,j}|\partial_x d^l_{ij}|e^{-\|\partial_x I^l_{i,j}\|} + |\partial_y d^l_{ij}|e^{-\|\partial_y I^l_{i,j}\|}$$

LOSS FUNCTION

Continued

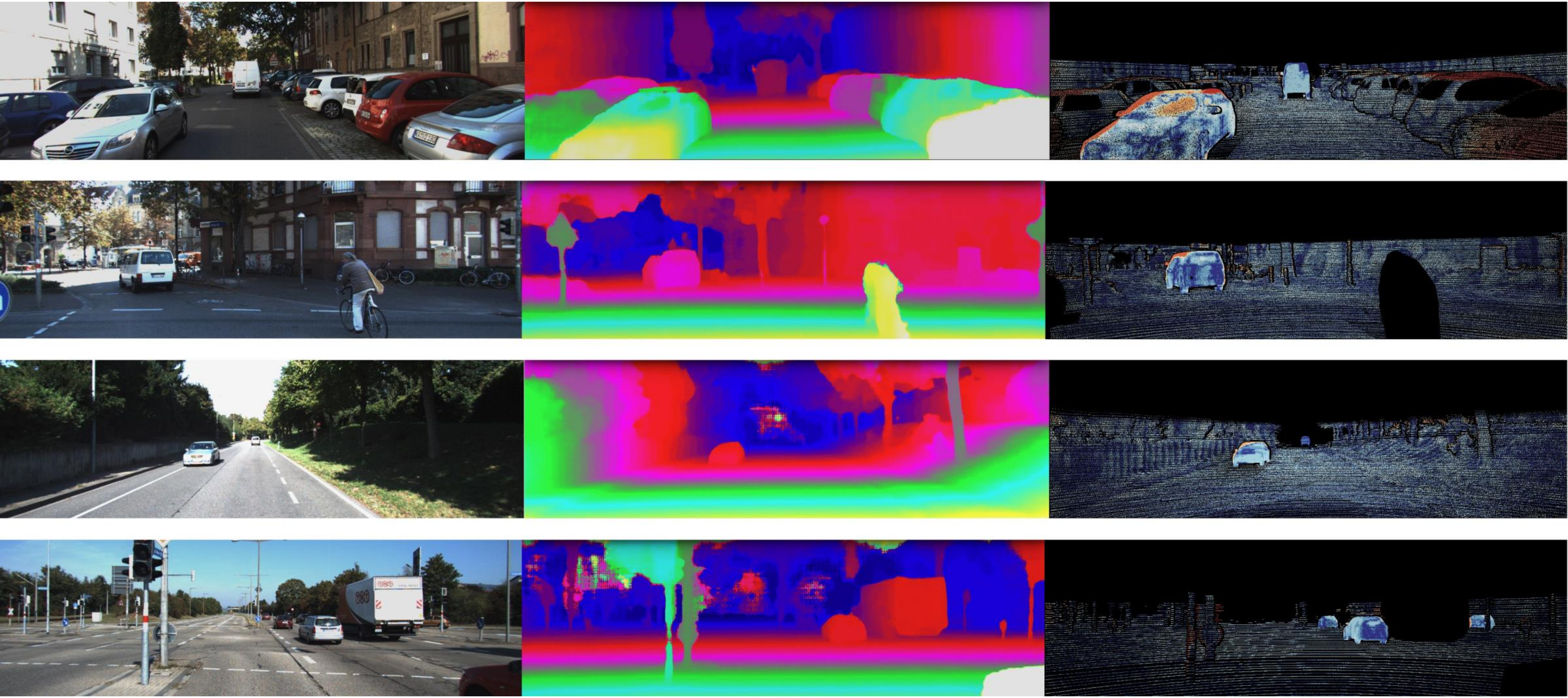
$$
\begin{aligned}
\tilde{I}^l &= w_{rl}(I_r, d_l) && (6)\\
\tilde{I}^r &= w_{lr}(I_l, d_r) && (7)\\
\tilde{d}^l &= w_{rl}(d_r, d_l) && (8)\\
\tilde{d}^r &= w_{lr}(d_l, d_r) && (9)\\
w_{lr}(I, d) &= (x, y) \mapsto I(x - d(x, y), y) && (10)\\
w_{rl}(I, d) &= (x, y) \mapsto I(x + d(x, y), y) && (11)\\
SSIM(x, y) &= \left(\frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}\right)\left(\frac{2\sigma_{xy} + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}\right) && (12)
\end{aligned}
$$

COST VOLUME CREATION

Model code in TensorFlow

```
def cost_volume_left_block(self, left, right, max_disp_steps, scope_name) :

    height = int(left.shape[1])
    width = int(left.shape[2])
    depth = int(left.shape[3])

    with tf.variable_scope(scope_name) as scope:
        right_padded = tf.pad(right,
            [[0, 0], [0, 0], [max_disp_steps-1, 0], [0,0]], "CONSTANT")
        right_disp = tf.extract_image_patches(right_padded,
            [1,height,width,1], [1,1,1,1], [1,1,1,1], padding="VALID")
        right_disp = tf.squeeze(right_disp, axis=1)
        disparity_dim = int(right_disp.shape[1])
        right_disp = tf.reshape(right_disp, [-1, disparity_dim, height, width, depth])
        right_disp = tf.reverse(right_disp, [1])

        left_disp = tf.expand_dims(left, axis=1)
        left_disp = tf.tile(left_disp,[1,disparity_dim,1,1,1])

        cost_volume = tf.concat([left_disp, right_disp], axis=4)

    return cost_volume
```

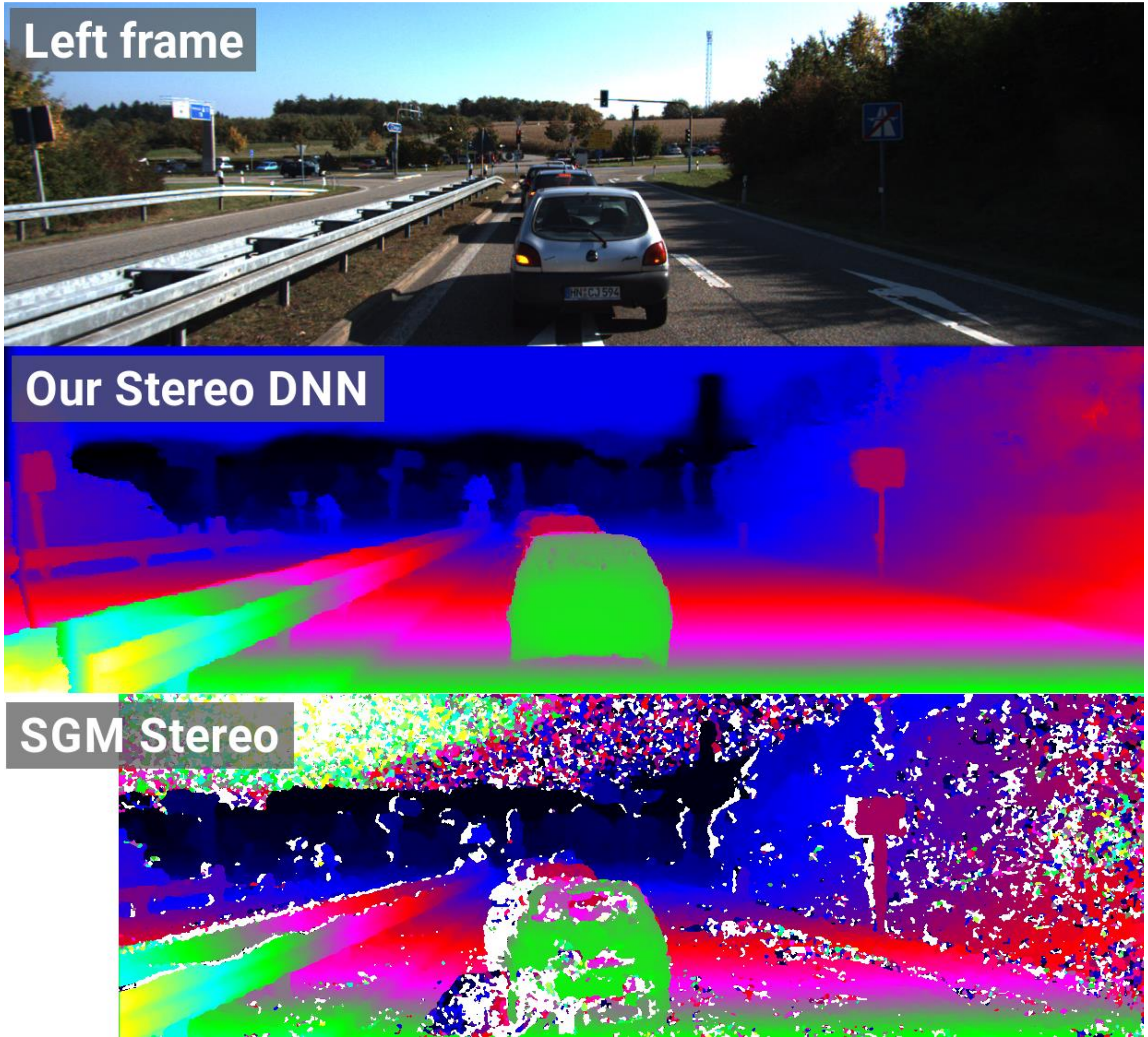
RESULTS



Left RGB frames | Computed depth, color-coded | Error maps |depth-lidar|

OUR STEREO DNN VS SEMI-GLOBAL MATCHING

Stereo DNN creates cleaner and more accurate depth



MONO DNN VS OUR STEREO DNN

Question: Can you guess what is the real geometry here?

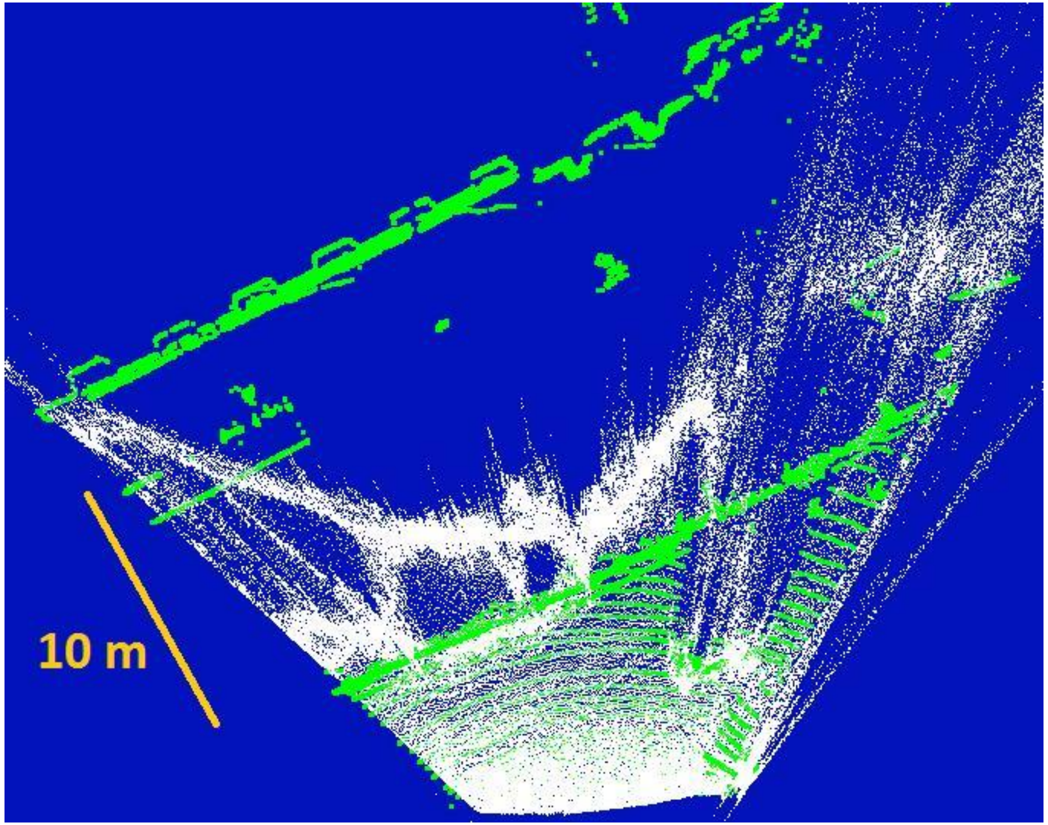
What is the distance to the fence (B)?

What is the distance from the fence (B) to the building (D)?

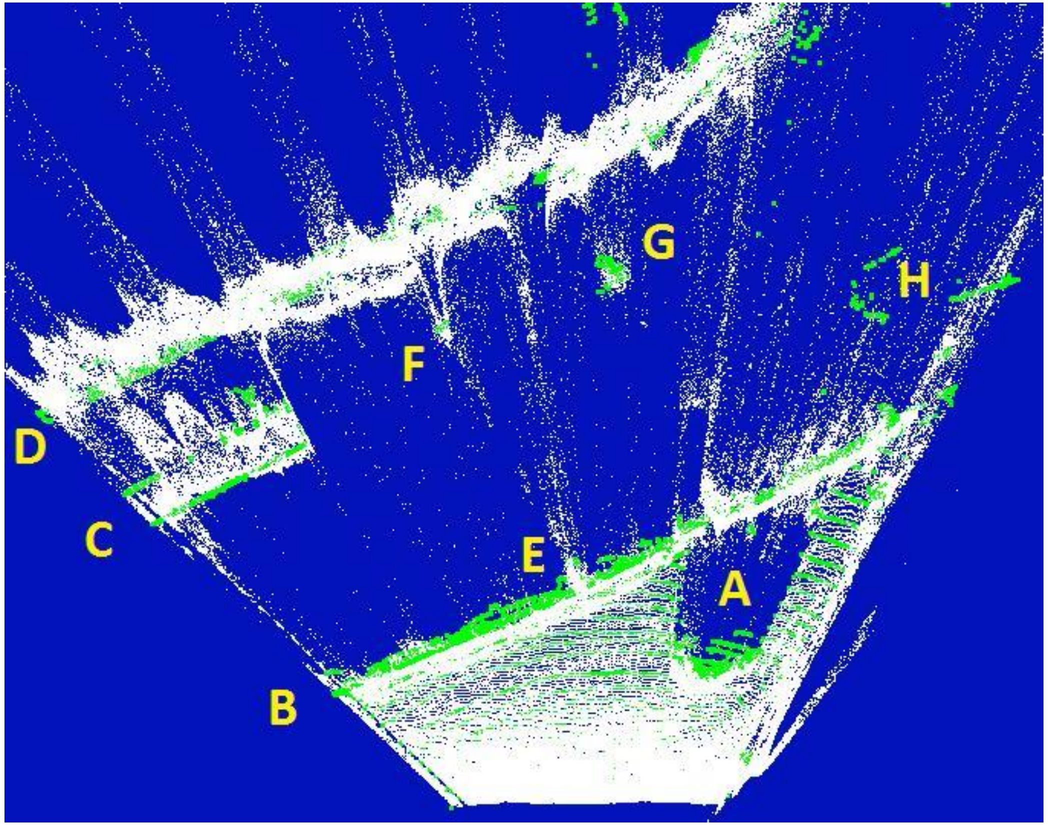
VIDEO DEMO (CONTINUED)

Answer: Mono DNN, Stereo DNN, LIDAR point clouds for that street view

https://youtu.be/0FPQdVOYoAU

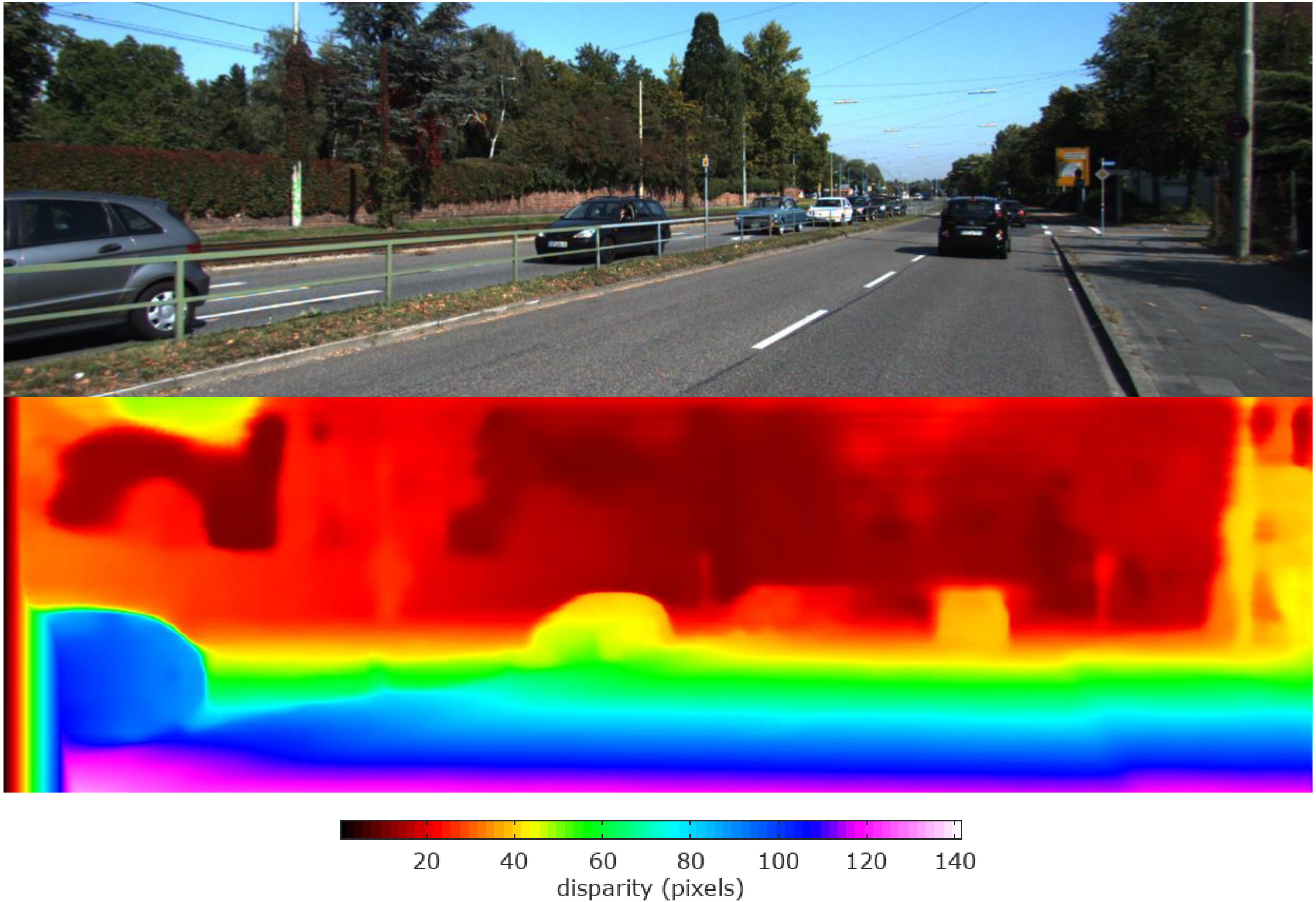


Mono DNN point cloud (white), LIDAR (green), top-down view

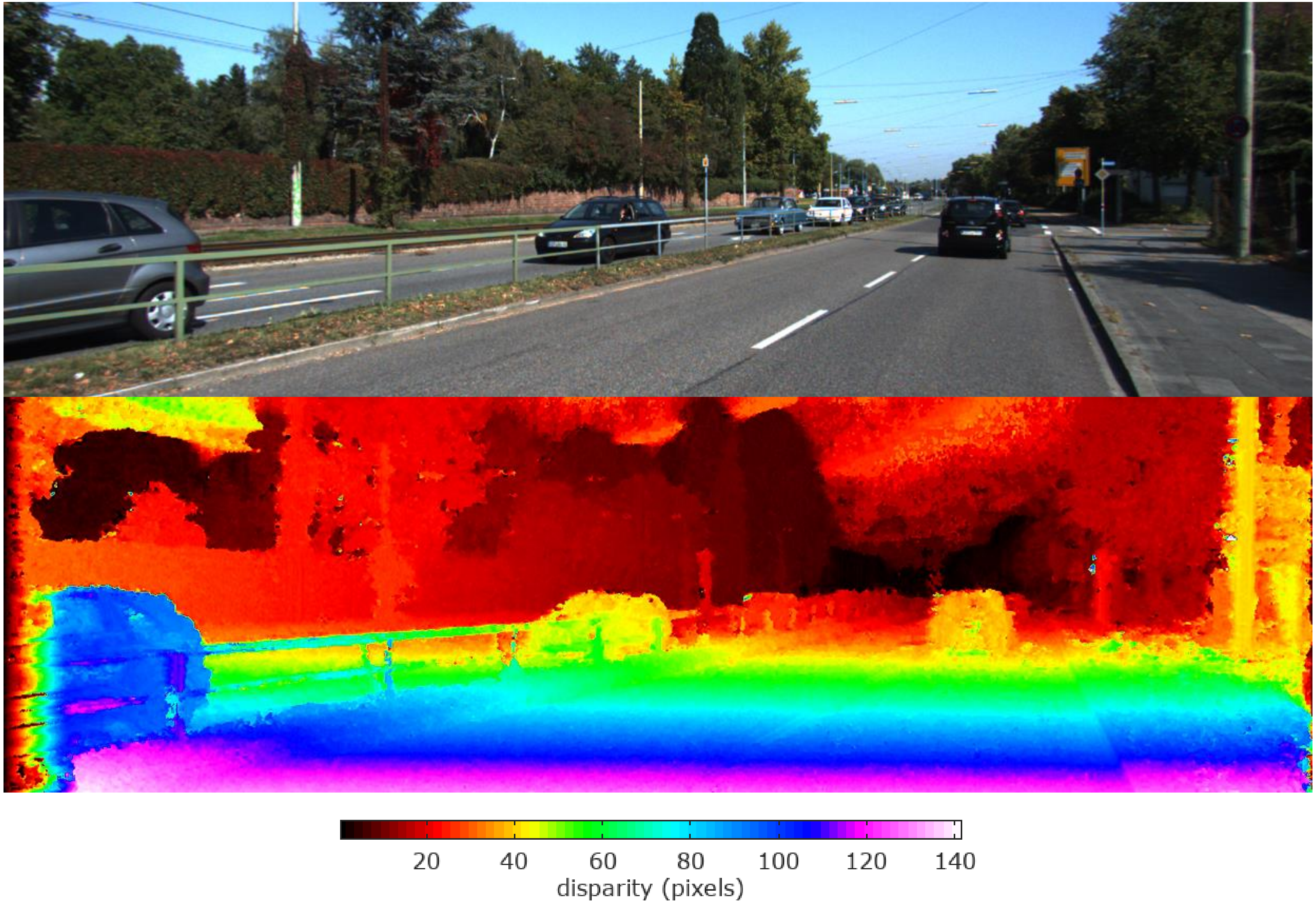


Our stereo DNN point cloud (white), LIDAR (green), top-down view

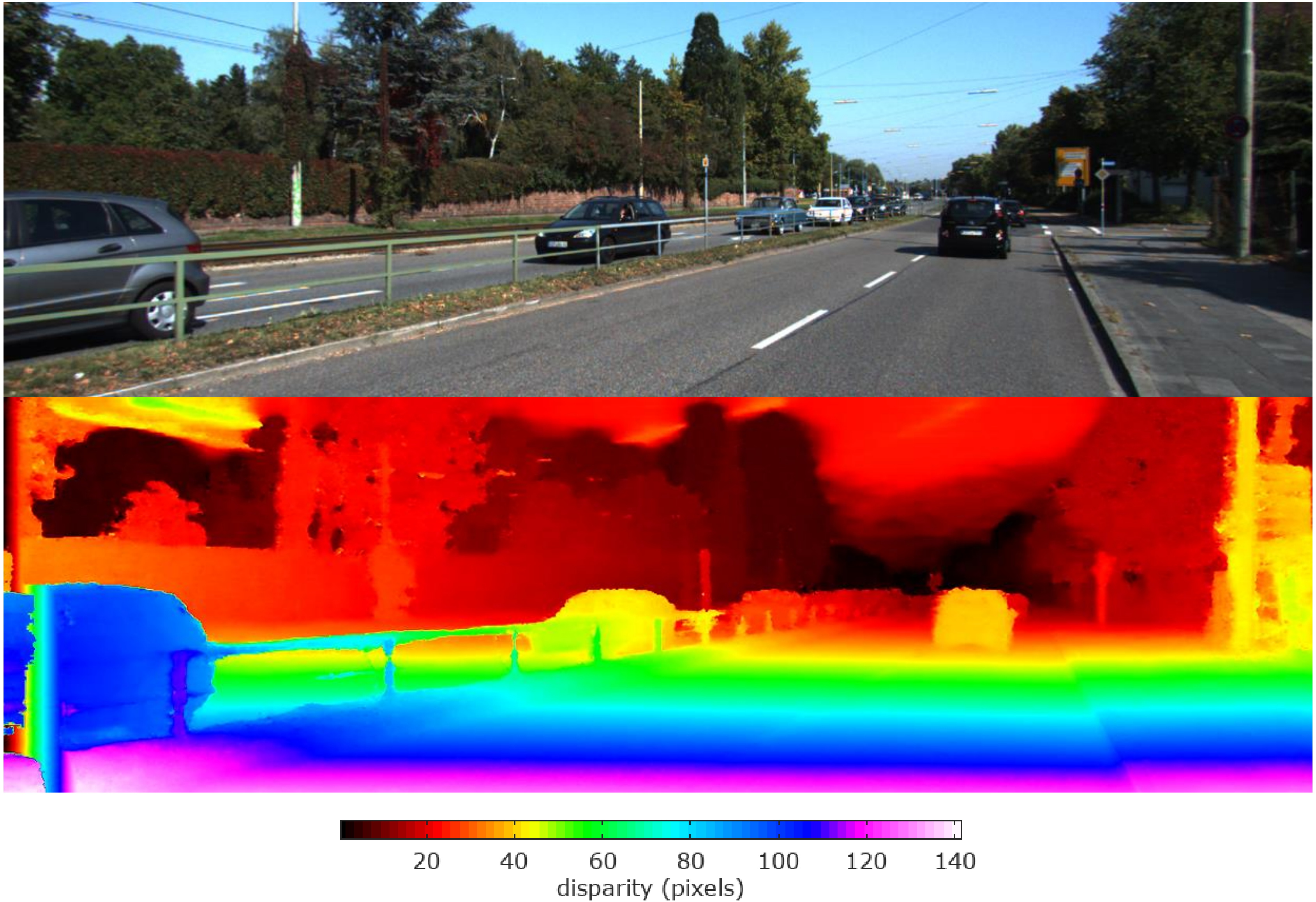
SUPERVISED BY LIDAR



UNSUPERVISED

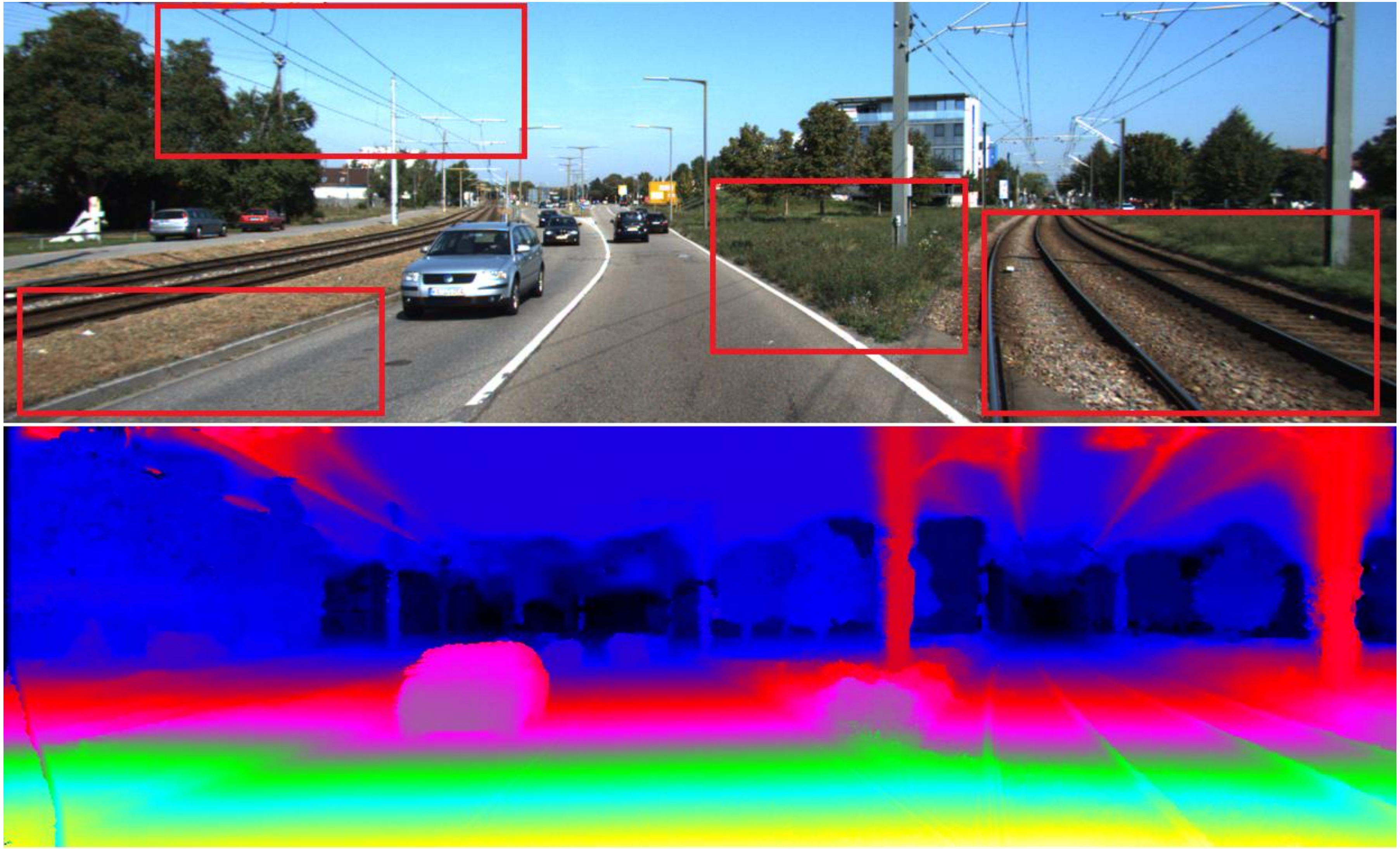


SEMI-SUPERVISED

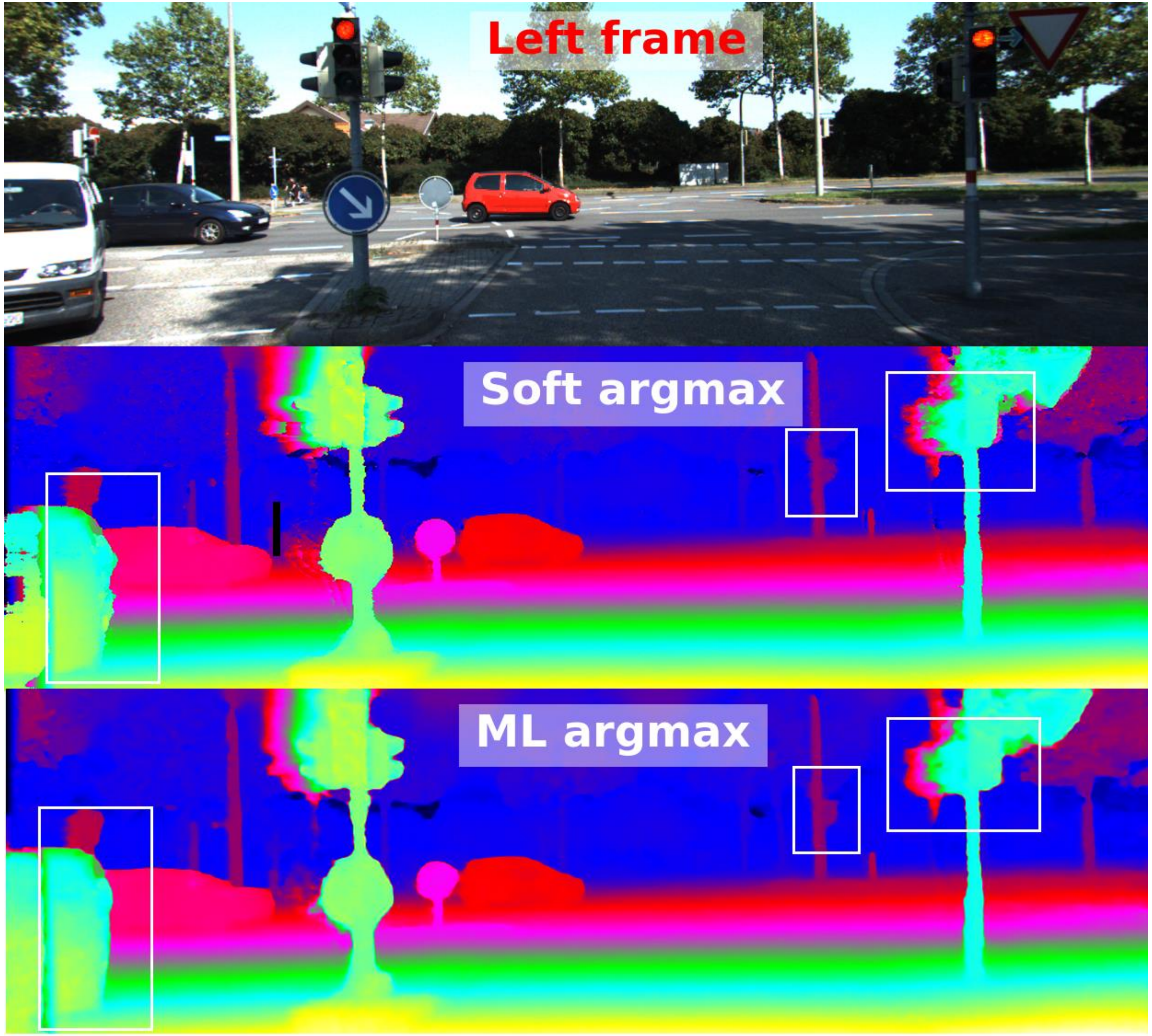


FINE GRAIN DETAILS

Stereo DNN is capable of capturing wires, rails, curbs, grass, etc.

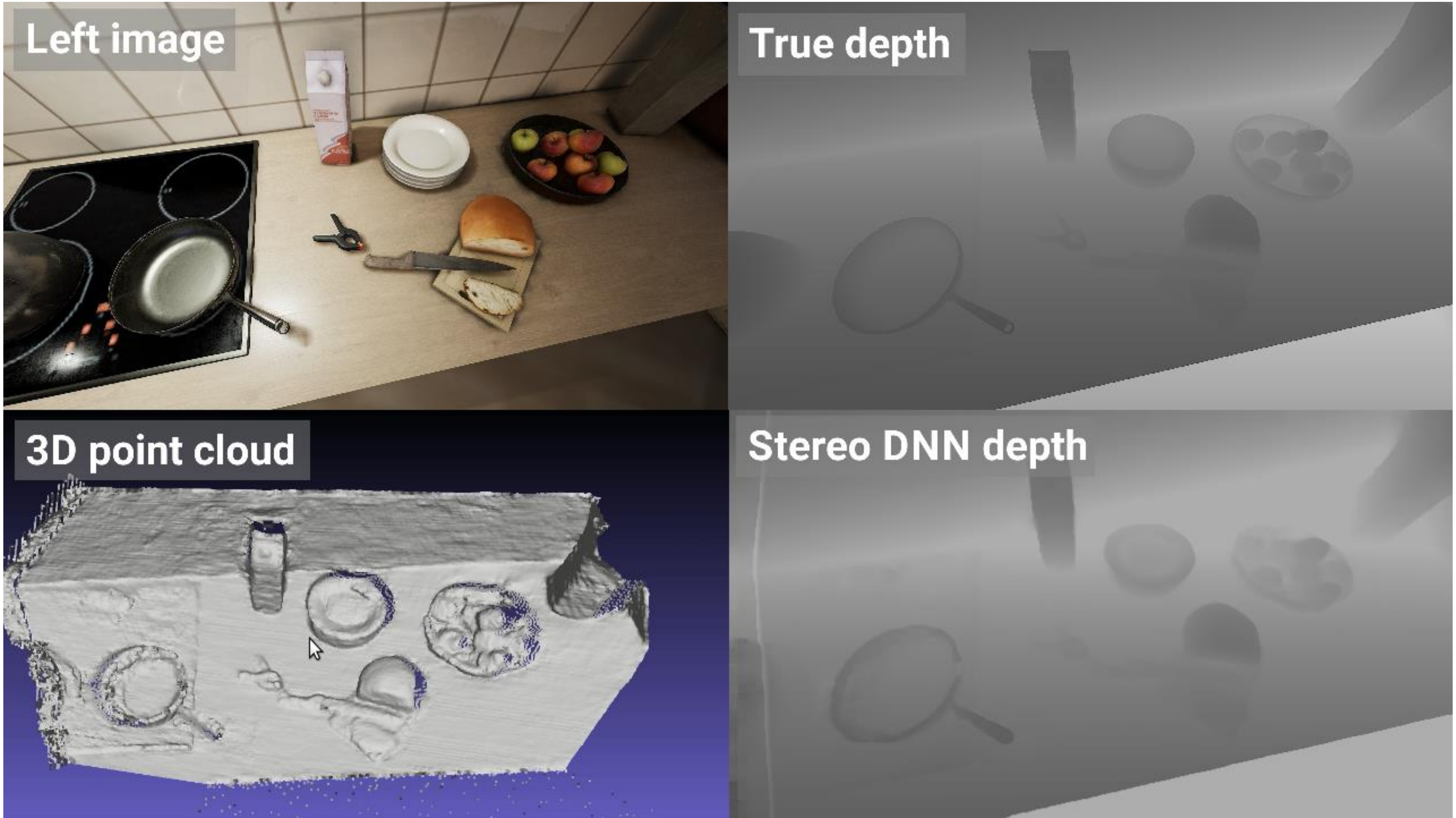


SOFT ARGMAX VS ML-ARGMAX



SYNTHETIC MODEL

We also trained and tested on synthetic 3D scenes



COMPARING MODELS ON KITTI 2015

This table shows our **KITTI D1 error:** % of pixels where disparity error is more than 3 pixels close range or more then than 5% further out

We show D1 error for models trained on **raw dataset** (sparse LIDAR, 29K frames)

Model / Training mode	Supervised w/ Lidar	Unsupervised Photometric	Semi-supervised Lidar + Photo
Mono Depth		32.8%	
Correlation based	14.6%	13.3%	12.9%
Our stereo DNN	15.0%	12.9%	8.8%

Semi-supervised mode with Lidar + Photo yields better results

COMPARING MODELS ON KITTI 2015

Numbers in red are for models trained on 200 scenes with **densified LIDAR depth**

Most papers report models trained on 200 scenes with densified LIDAR depth

When we fine-tune on those 200 dense scenes, we are in top 10 KITTI 2015 stereo

Model	size	Lidar + photo D1 error
No bottleneck	0.2M	14.5%
Correlation	2.7M	12.9%
Small	1.8M	9.8%
Tiny (near real-time)	0.5M	11.9%
Single tower	2.8M	10.1%
Resnet18 based (our baseline)	2.8M	8.8%
ML-argmax	3.1M	8.7%
ML-argmax + dense depth	3.1M	3.5%
Resnet18 based + dense depth	2.8M	3.4%

KITTI 2015 BENCHMARK

Most papers report models trained on 200 scenes with densified LIDAR depth

When we fine-tune on those 200 dense scenes, we are in top 10 KITTI 2015 stereo

Model	D1-background	D1-foreground	D1-All
DispNetC	4.3%	4.4%	4.3%
SGM-Net	2.7%	8.6%	3.7%
GC-Net	2.2%	6.2%	2.9%
CRL	2.5%	3.6%	2.7%
L-ResMatch	2.7%	7.0%	3.4%
Ours (no-finetuning)	3.2%	14.8%	5.1%
Ours (finetuned on dense 200)	**2.7%**	**6.0%**	**3.4%**

INFERENCE RUNTIME

Project Redtail has runtime inference lib on GitHub

The library implements operations currently not available in TensorRT

Operations are implemented as custom TensorRT plugins

To run, use 2-step process:

- Convert TensorFlow binary model to TensorRT C++ API code
- Use generated C++ code in your TensorRT inference code

Note: TensorFlow runtime is NOT required to run our stereo DNN

INFERENCE RUNTIME

Our custom TensorRT plugins

3D convolutions and transposed 3D convolutions aka deconvolutions

- TensorFlow and cuDNN have different implementations of 3D convolution
- TensorFlow’s 3D convolution can be represented in cuDNN by reshaping and proper stride/padding calculation

INFERENCE RUNTIME

Our custom TensorRT plugins

ELU activation function

Cost volume plugin (stereo DNN specific)

Multidimensional soft-argmax plugin

Auxiliary plugins necessary for stereo DNN model:

- Tensor transforms/transpose
- Padding (due to asymmetric padding in TensorFlow)
- Slicing (same reason as for padding)

INFERENCE PERFORMANCE

On Different NVIDIA GPUs

Model	Resolution	D1 error	Titan XP performance (ms)		Jetson TX2 perf (ms)
		%	TensorFlow	TensorRT	TensorRT
ResNet-18	1025x321	3.4	950	650	11000
NVSmall	1025x321	9.8	800	450	7800
NVTiny	513x161	11.1	75	40	360

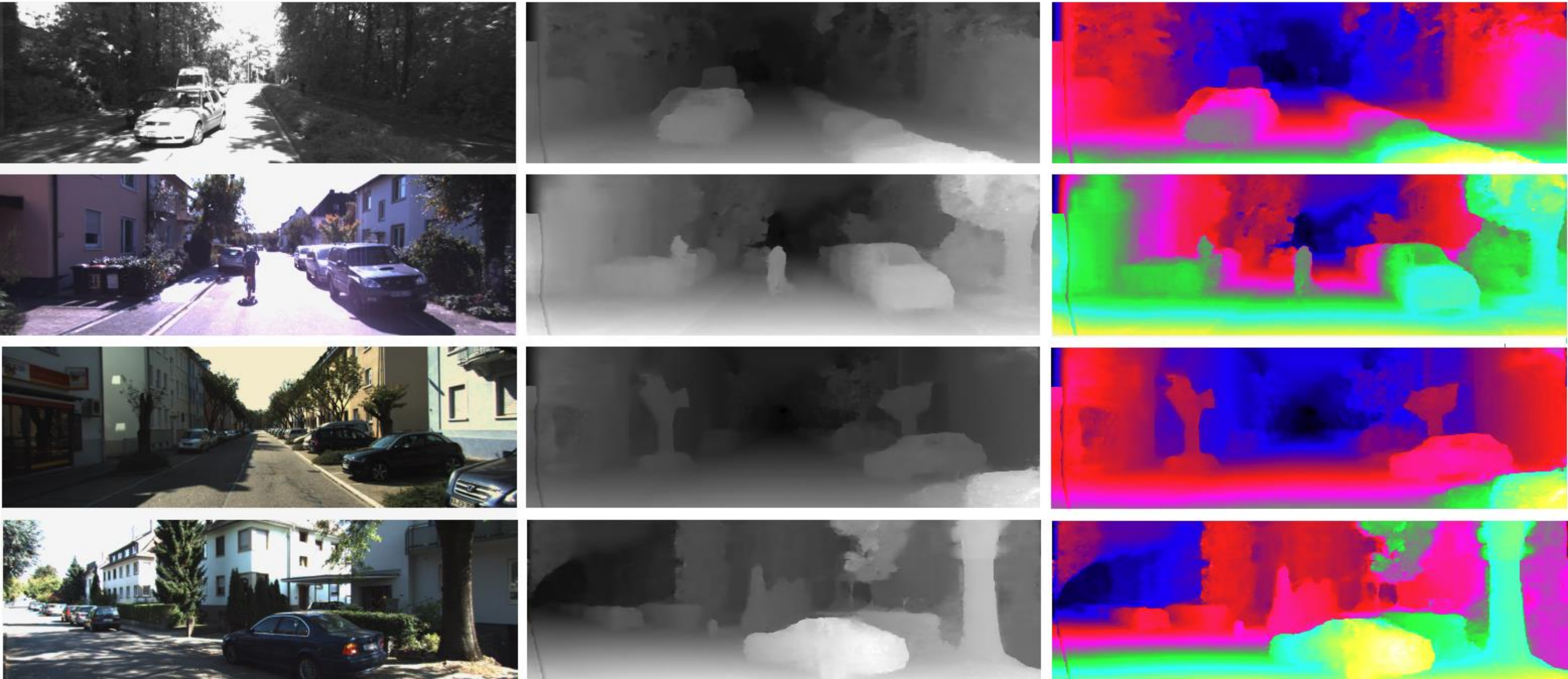
Notes:

- D1 error for ResNet-18 was measured on KITTI 2015 benchmark 200 test images. The model was fine-tuned on 200 train images (with dense LIDAR) after training on full KITTI
- D1 error for NVSmall and NVTiny was measured on KITTI 2015 benchmark 200 training images. These models were trained on full KITTI (with sparse LIDAR)
- FP16: at the moment, 3D convolutions in cuDNN are not optimized for FP16

INFERENCE ON JETSON

NVTiny model runs at 3 FPS on TX2

Stereo DNN depth output with left frame input. Model: "NVTiny", semi-supervised, 512x160, 48 max disparity, trained on KITTI



CONCLUSIONS AND FUTURE WORK

We can train fairly accurate stereo DNN end-to-end

Stereo DNNs not only do matches, but also understand context

Better accuracy is needed around fine branches, poles, etc.

Cannot yet estimate depth of textureless objects at infinity

More work needed to run depth DNNs on embedded GPUs in real-time

NVIDIA