

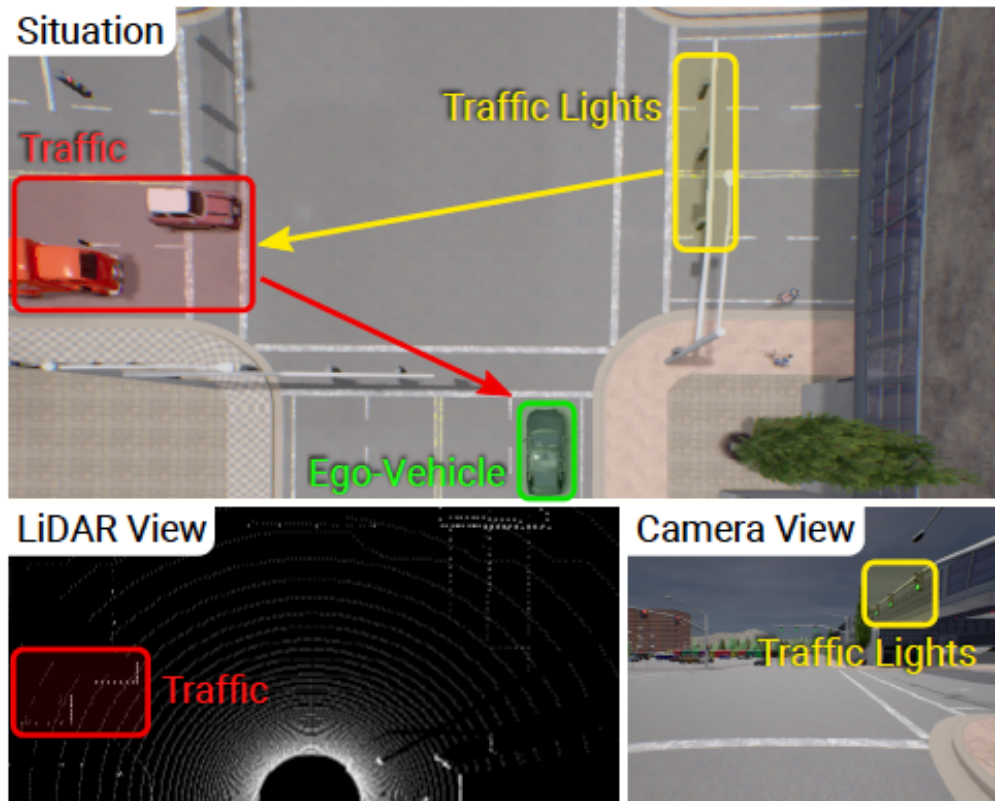
# Bird's Eye Views

D.A. Forsyth, UIUC

# Bird's Eye Views (BEVs)

- Issue:
  - what coordinate frame should we use to make decision
- Idea:
  - Ground plane is better than image
  - Why:
    - relations between objects are preserved

# Bird's Eye Views (BEVs)



Chitta et al 22

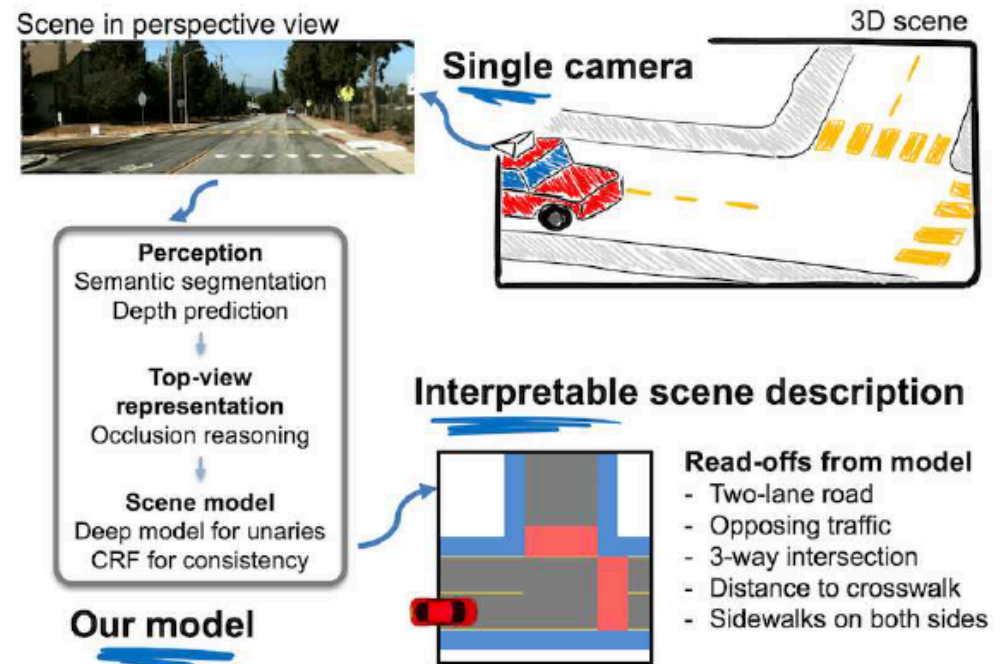
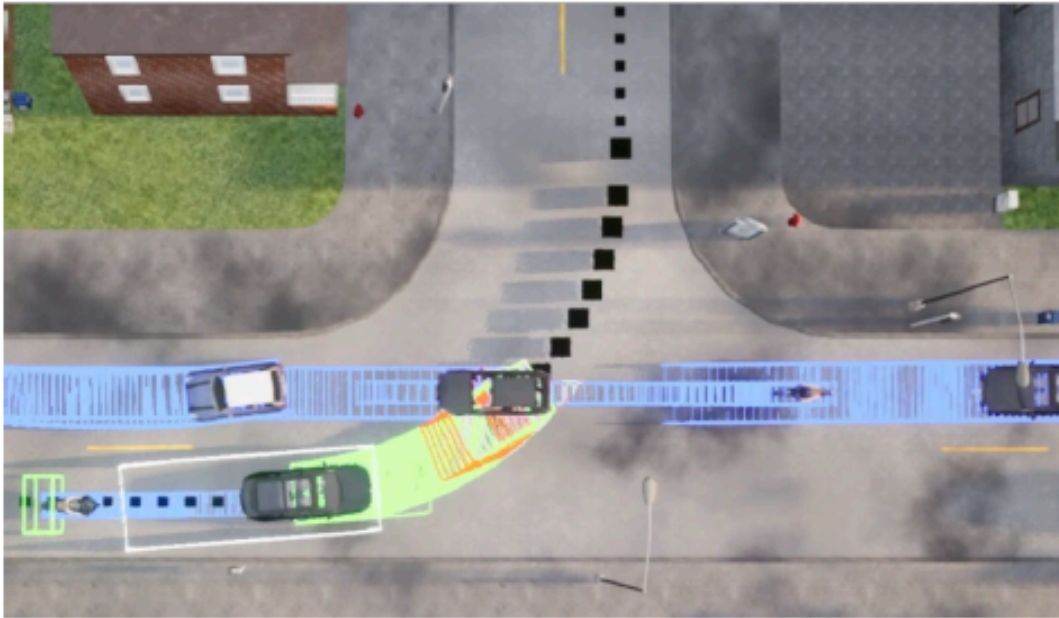
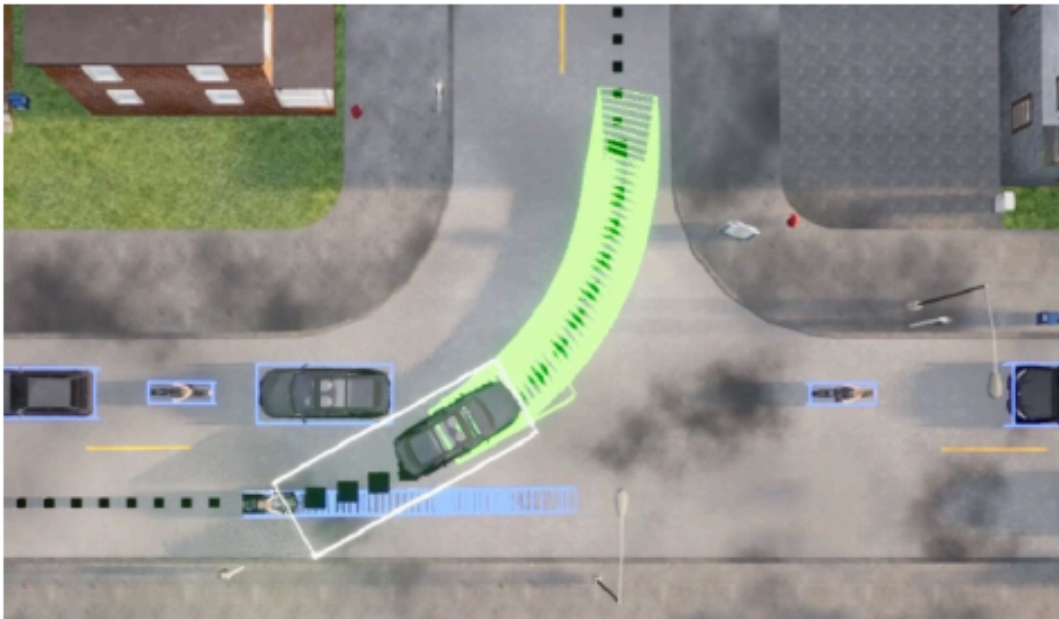


Figure 1: Our goal is to infer the layout of complex driving scenes from a single camera. Given a perspective image (top left) that captures a 3D scene, we predict a rich and interpretable scene description (bottom right), which represents the scene in an occlusion-reasoned semantic top-view.

Wang et al 19



(a) The expert waits before taking the turn because the trajectory forecasting predicts a collision if the expert would drive.



(b) After the oncoming cars have passed, the expert crosses the intersection.

# In BEV, layout is stylized

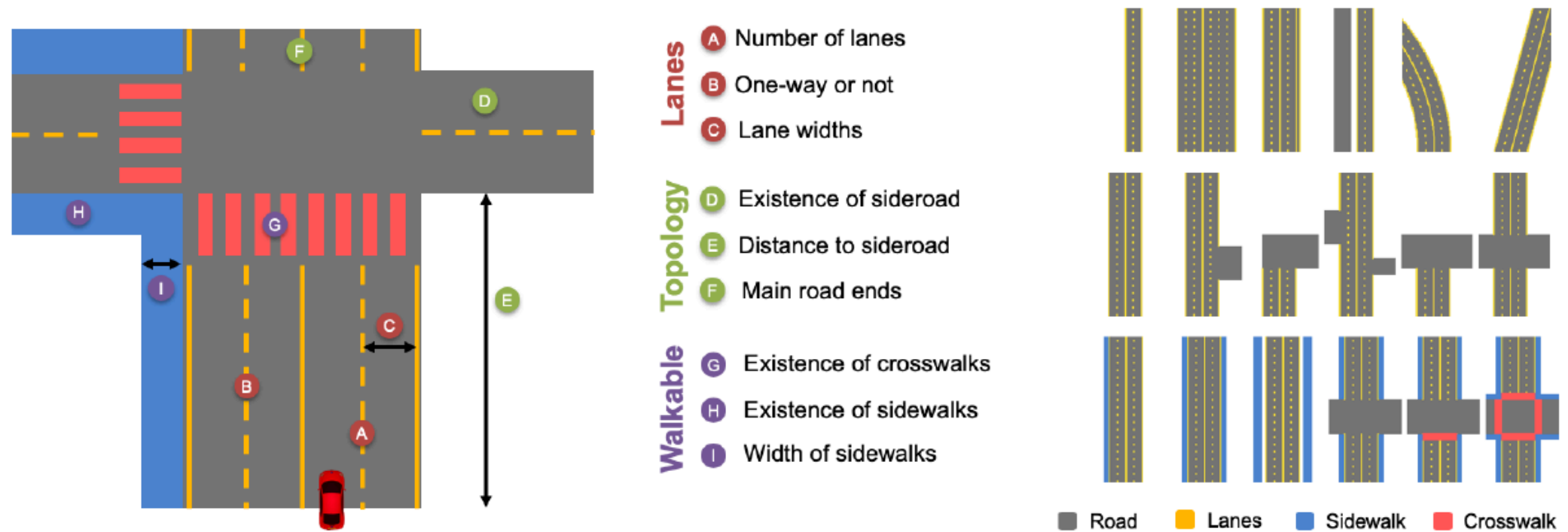


Figure 2: Our scene model consists of several parameters that capture a variety of complex driving scenes. (Left) We illustrate the model and highlight important parameters (A-I), which are grouped into three categories (middle): *Lanes*, to describe the layout of a single road; *Topology*, to model various road topologies; *Walkable*, describing scene elements for pedestrians. Our model is defined as a directed acyclic graph enabling efficient sampling and is represented in the top-view, making rendering easy. These properties turn our model into a simulator of semantic top-views. (Right) We show rendered examples for each of the above groups. A complete list of scene parameters and the corresponding graphical model is given in the supplementary.

Q: How do we get one from sensors?

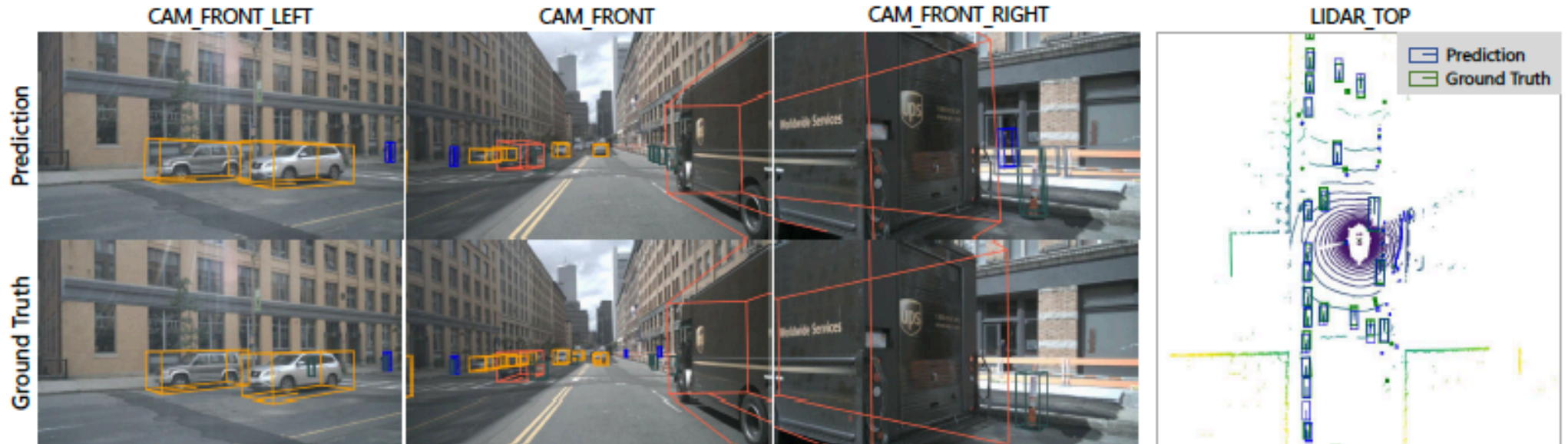
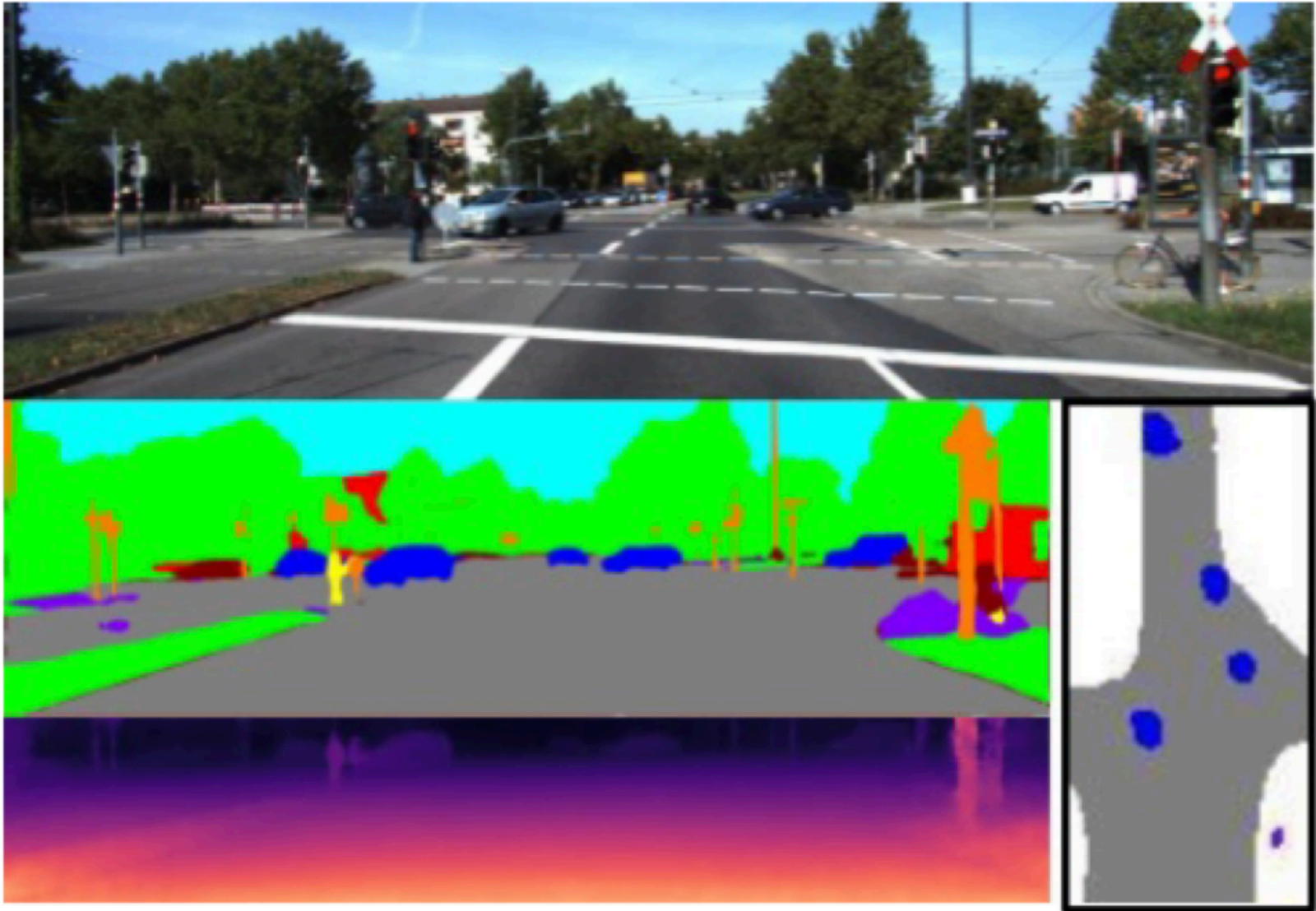


Fig. 4. Visualization results of BEVFormer on nuScenes val set. We show the 3D bboxes predictions in multi-camera images and the bird's-eye-view.

# Issues

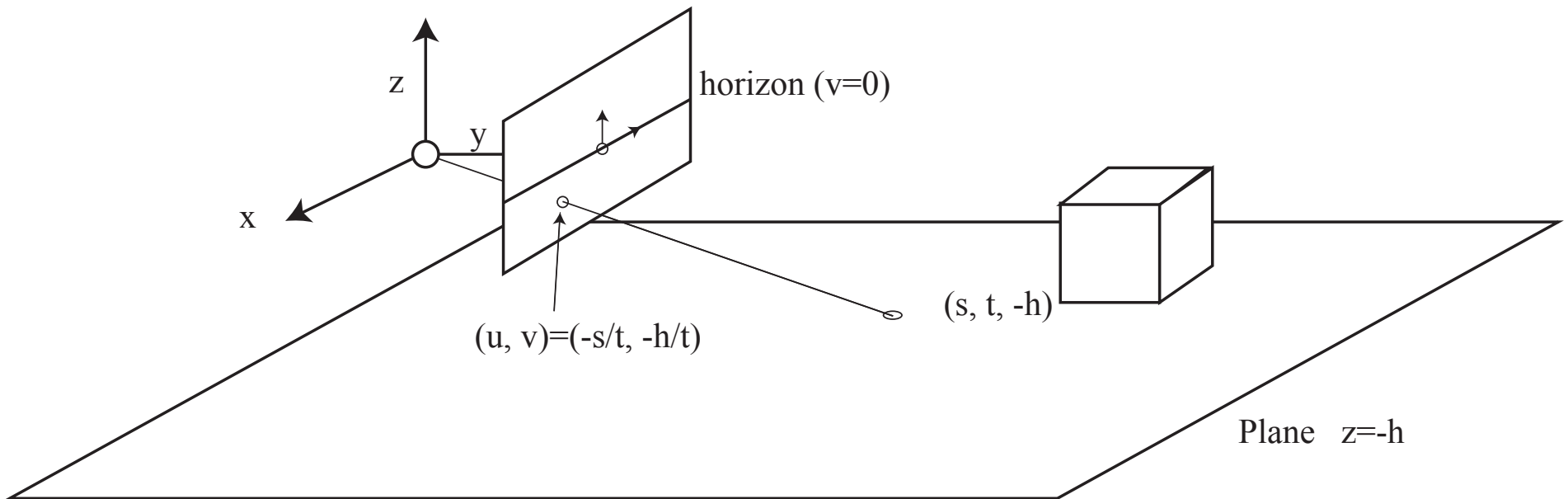


# Technologies

- Camera Geometry
- Segmentation
- Depth from single image
- Normal from single image
- Inpainting
- Registration
- Adversarial Losses



# Camera Geometry

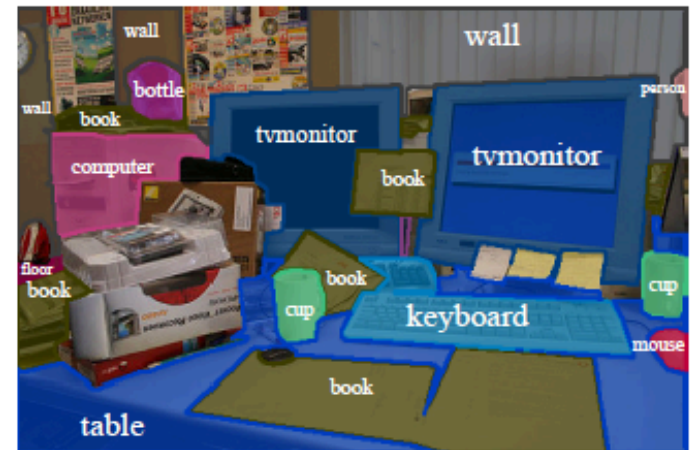
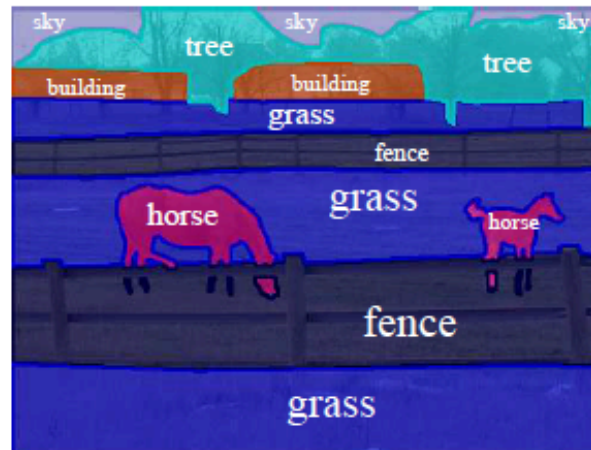
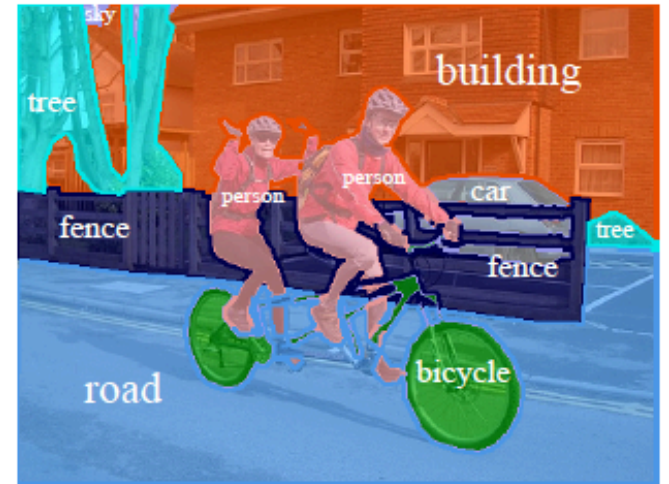
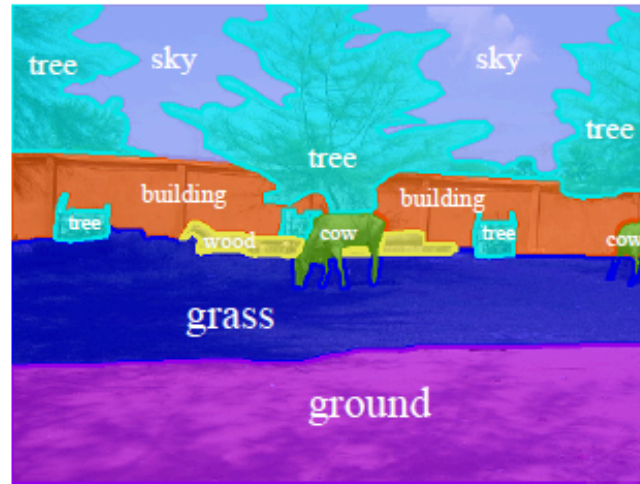


# Semantic segmentation

D.A. Forsyth

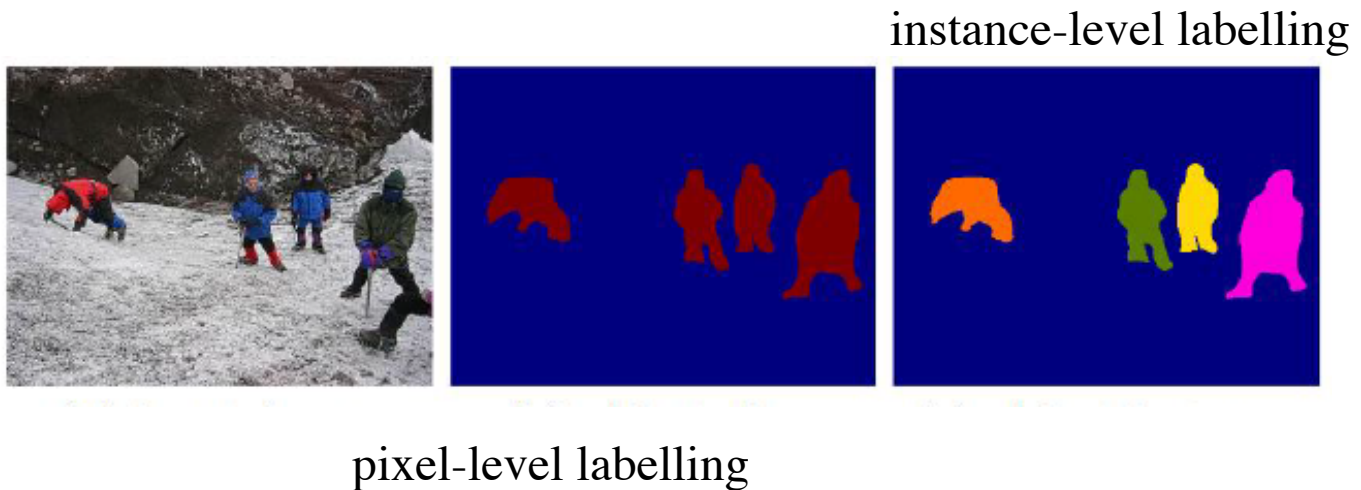
# The problem

- Tag each pixel with a class name for some set of classes



# Variants: Semantic Instance Segmentation

- Tag every pixel,
  - BUT different instances of the same class get different tags



# Variants: 3D semantic segmentation

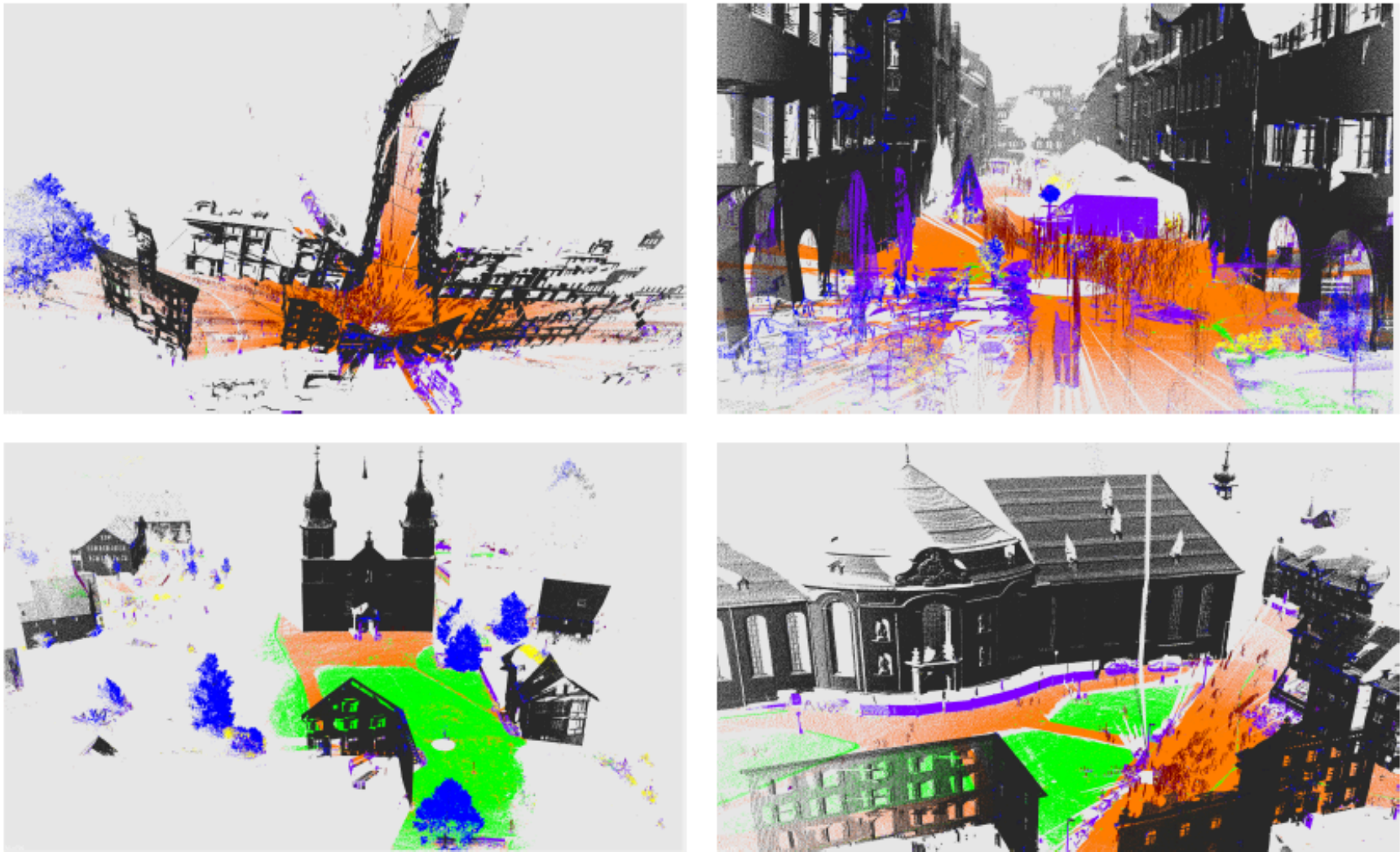


Figure 5. Results for terrestrial laser scans. *Top row:* urban street in St. Gallen (left), market square in Feldkirch (right). *Bottom row:* church in Bildstein (left), cathedral in St. Gallen (right) with classes: **man-made terrain**, **natural terrain**, **high vegetation**, **low vegetation**, **buildings**, **remaining hard scape** and **scanning artefacts**.

Hackel et al

# Variants: Map to Scene model

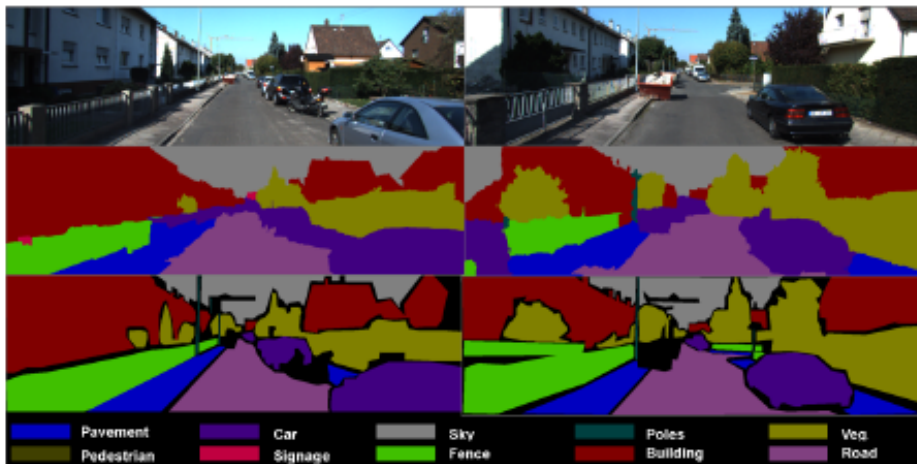


Fig. 6: *Semantic image segmentation*: The top row shows the input street-level images and the middle row shows the output of the CRF labeller. The bottom row shows the corresponding ground truth for the images.

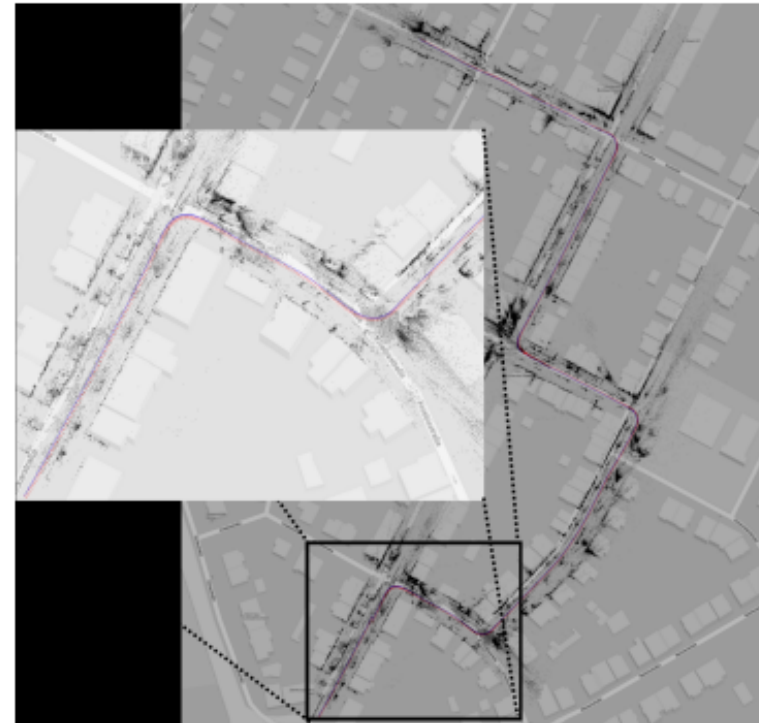


Fig. 3: Bundle adjustment results, showing camera centres and 3D points, registered manually to the Google map.

# Variants: Map to Scene model

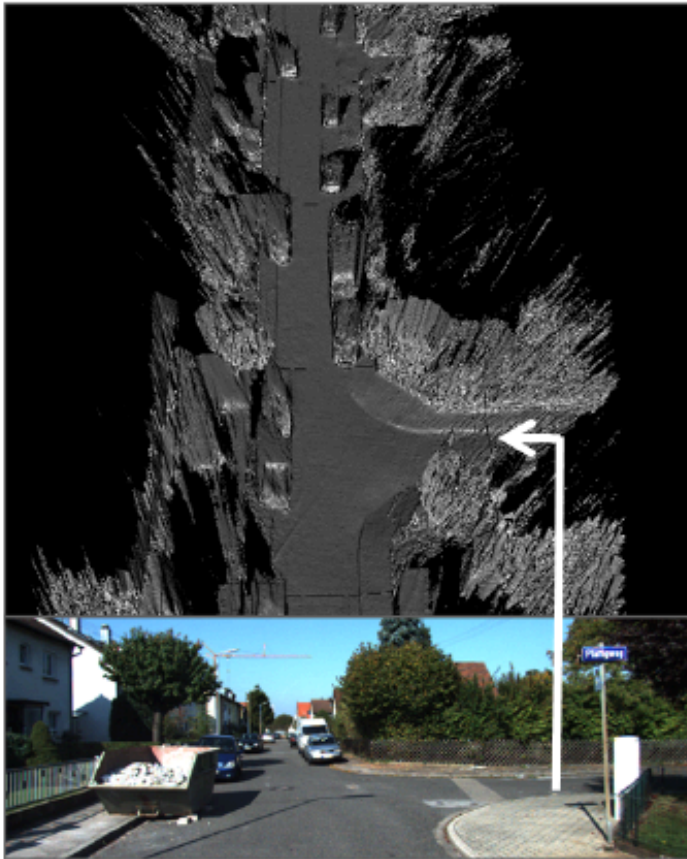


Fig. 4: *Volumetric surface reconstruction*. Top figure shows the 3D surface reconstruction over 250 frames (KITTI sequence 15, frames 1-250) with street image shown at the bottom. The arrow highlights the relief of the sidewalk which is correctly captured in the 3D model.



Fig. 8: Semantic model of the reconstructed scene overlaid with the corresponding Google Earth image. The inset image shows the Google earth track of the vehicle.

# Variants: Stixels

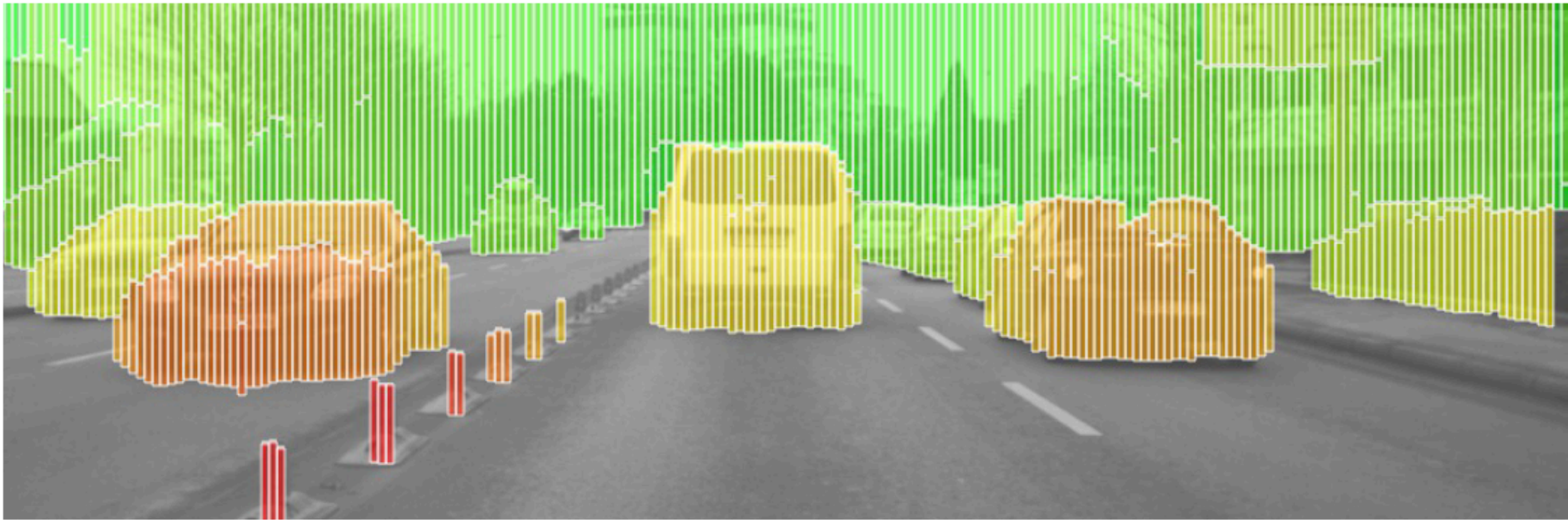


Figure 1: The multi-layer *Stixel World* result as output of the optimization. The captured scene is segmented into planar *Stixel* segments that correspond to either ground or object. The color represents the distance to the obstacle with red being close and green far away. Grey pixels belong to the ground surface.

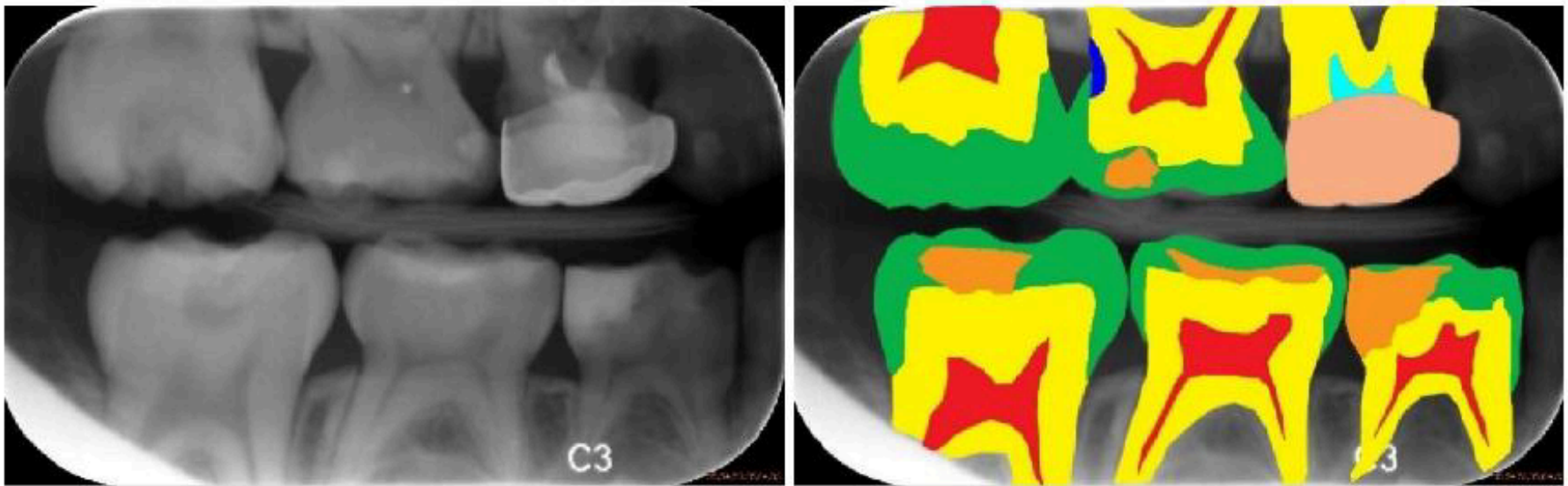


# Why bother?



Driving (maybe - why everything?)

# Why bother?



Medical applications (compelling)

# Important variants

- Partial semantic segmentation
  - some pixels unlabelled
- Thing segmentation
  - label “things”
    - count nouns (car, person, dog...)
- Stuff segmentation
  - label “stuff”
    - mass nouns (grass, sky, water...)
- Panoptic segmentation
  - each pixel gets a label
  - each instance of a count noun gets a different label (person-a, etc)
    - I \*think\* MS-COCO and Cityscapes use the term differently

# Issues

- Label distributions are skewed
  - Pascal 2010
    - from Mottaghi et al 14

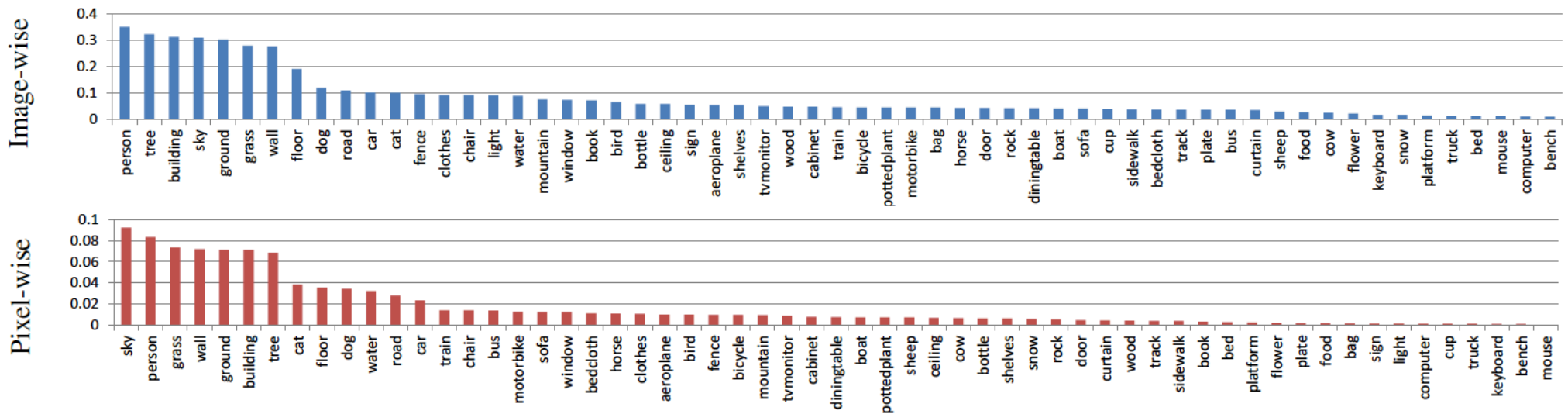
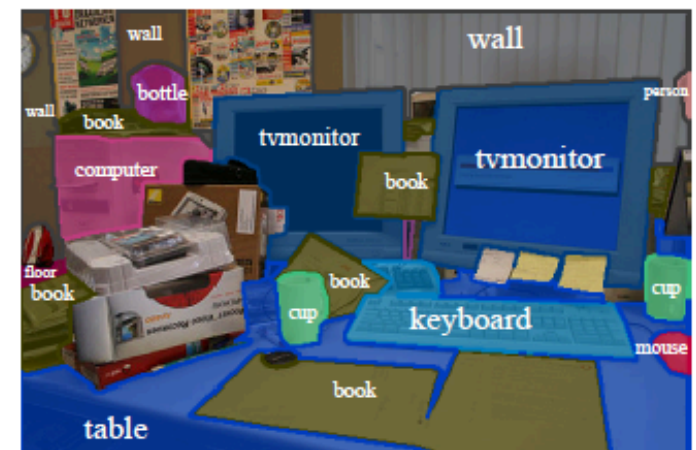
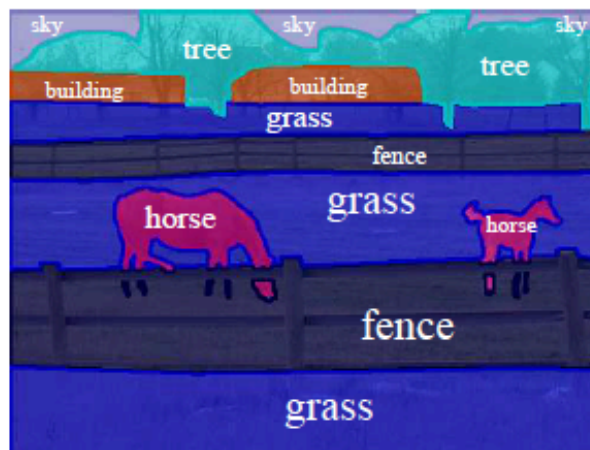
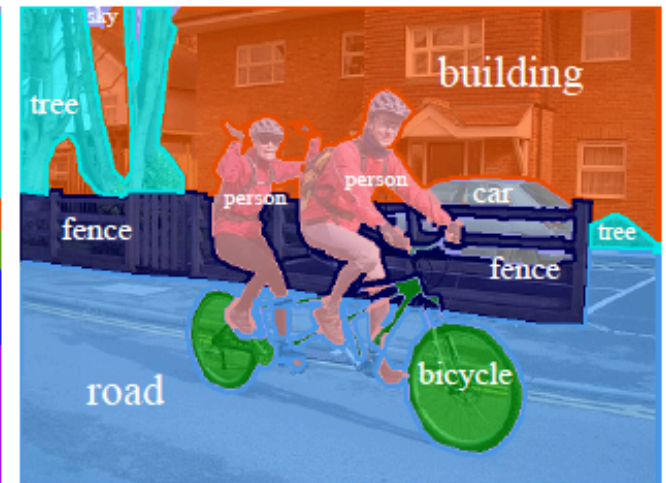
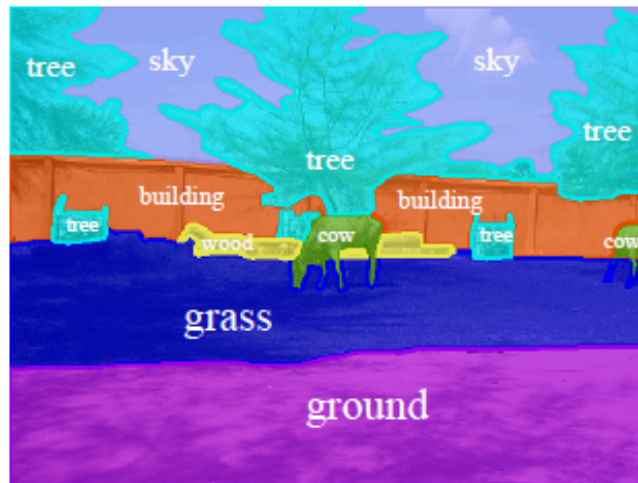


Figure 2. Distribution of pixels and images for the 59 most frequent categories. See text for the statistics.

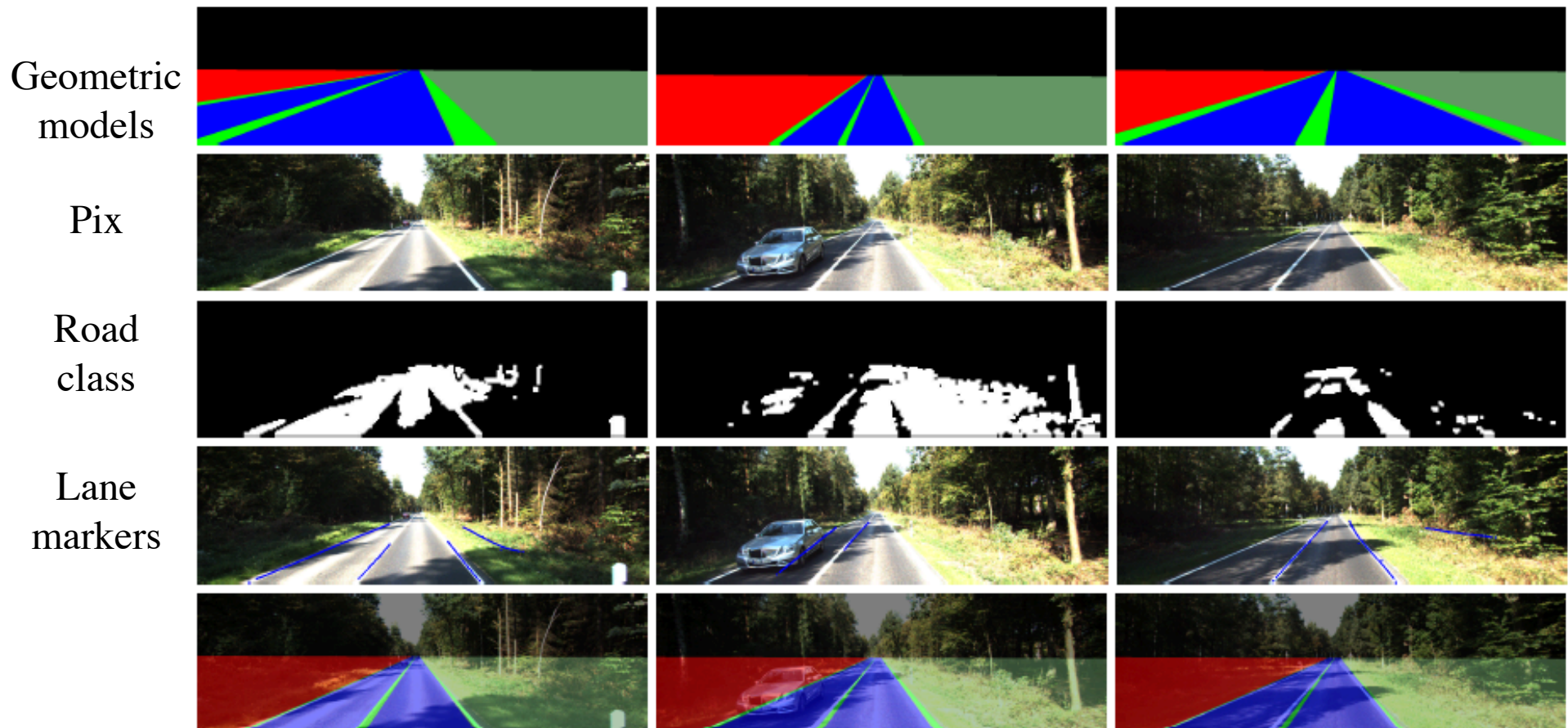
# Issues

- Some ambiguity in labelling

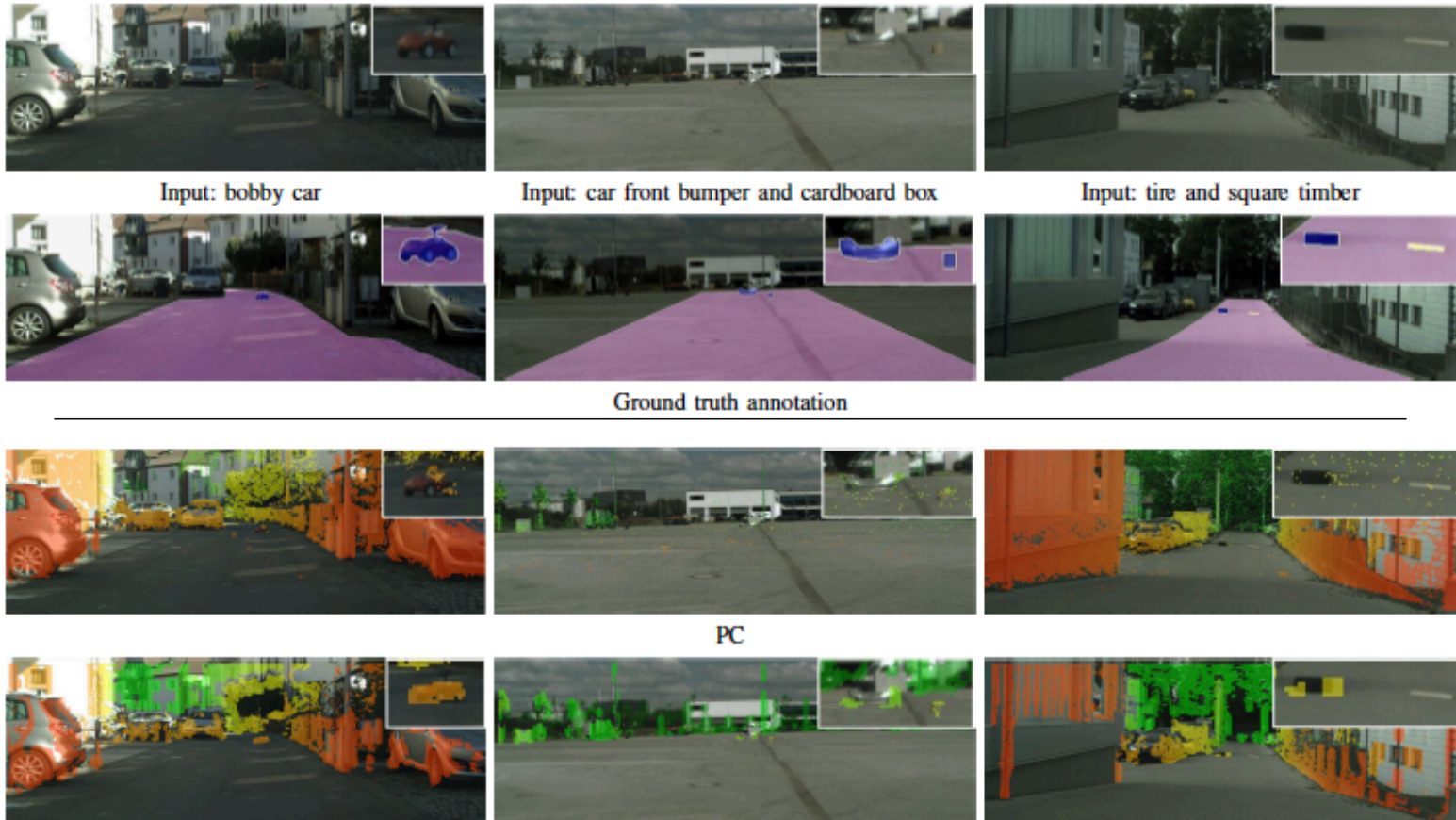


This is a pixel-level labelling

# Spatial structure is an issue



# Small things are important



# More issues

- Data
- Spatial models
- Appearance models
- Managing scale, context, etc.



# Contrast with segmentation



Learning a semantic segmenter should be **\*MUCH\*** easier  
cause you **KNOW** what label each pixel should have  
and labels transfer across images

# Evaluation

To assess performance, we rely on the standard Jaccard Index, commonly known as the PASCAL VOC intersection-over-union metric  $IoU = TP / (TP+FP+FN)$  [1], where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole test set. Owing to the two semantic granularities, i.e. classes and categories, we report two separate mean performance scores:  $IoU_{category}$  and  $IoU_{class}$ . In either case, pixels labeled as void do not contribute to the score.

# Evaluation, II

It is well-known that the global IoU measure is biased toward object instances that cover a large image area. In street scenes with their strong scale variation this can be problematic. Specifically for traffic participants, which are the key classes in our scenario, we aim to evaluate how well the individual instances in the scene are represented in the labeling. To address this, we additionally evaluate the semantic labeling using an instance-level intersection-over-union metric  $iIoU = iTP / (iTP + FP + iFN)$ . Again  $iTP$ ,  $FP$ , and  $iFN$  denote the numbers of true positive, false positive, and false negative pixels, respectively. However, in contrast to the standard IoU measure,  $iTP$  and  $iFN$  are computed by weighting the contribution of each pixel by the ratio of the class' average instance size to the size of the respective ground truth instance. It is important to note here that unlike the instance-level task below, we assume that the methods only yield a standard per-pixel semantic class labeling as output. Therefore, the false positive pixels are not associated with any instance and thus do not require normalization. The final scores,  $iIoU_{category}$  and  $iIoU_{class}$ , are obtained as the means for the two semantic granularities.

# (Some) Datasets

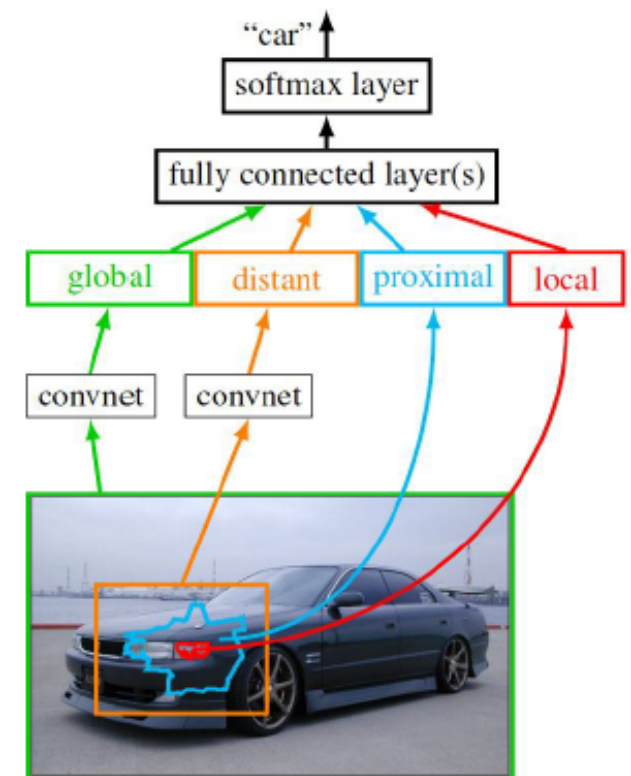
- Cityscapes
  - <https://www.cityscapes-dataset.com/benchmarks/>
- Pascal VOC 2010 context
  - <https://cs.stanford.edu/~roozbeh/pascal-context/>
- Kitti
  - [http://www.cvlibs.net/datasets/kitti/eval\\_semantics.php](http://www.cvlibs.net/datasets/kitti/eval_semantics.php)
  - also see other annotations at bottom of page
- Mapillary vistas
  - <https://research.mapillary.com/img/publications/ICCV17a.pdf>
- MS COCO
  - <http://cocodataset.org/#panoptic-2018>

# Procedure

- Produce a feature vector at each pixel
- Classify into  $k$  classes using that
- Optional
  - (apparently of declining importance)
  - Use conditional random field, etc. to clean up predictions

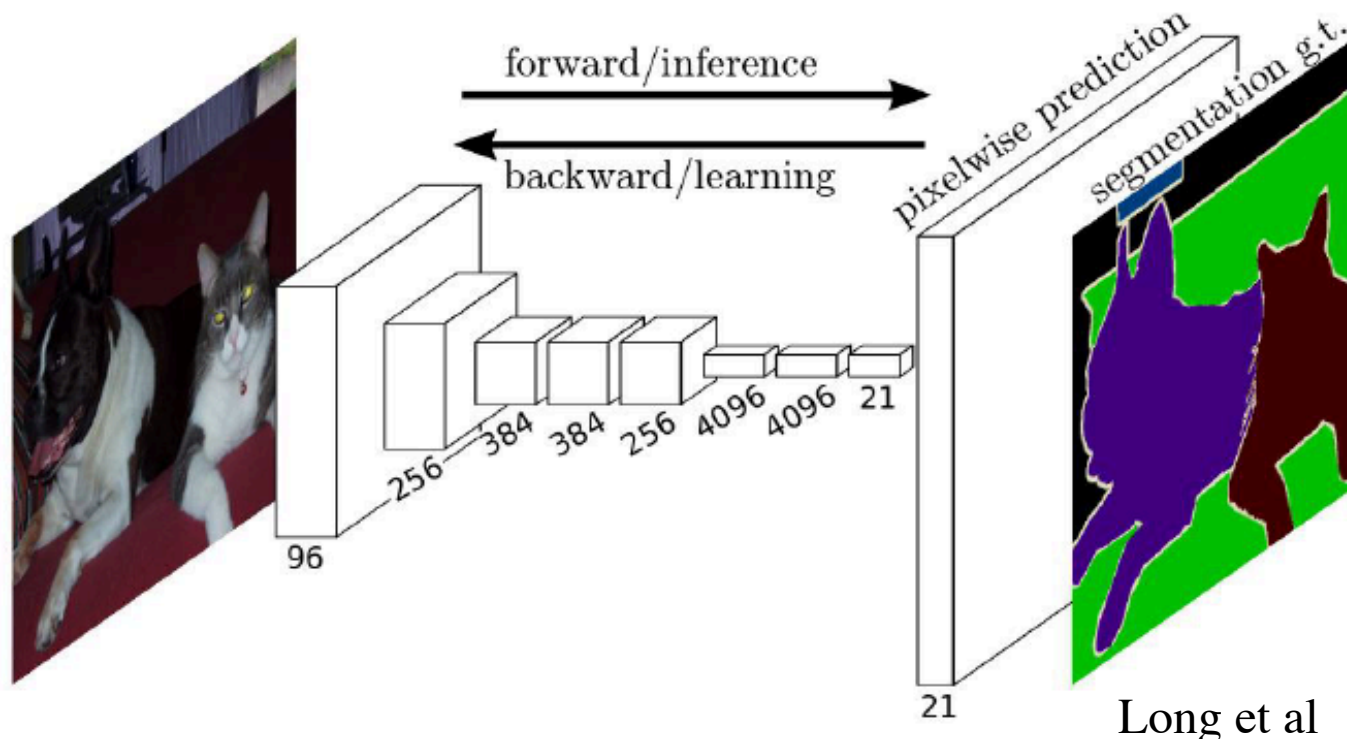
# Early ideas

- Label pixel using
  - its appearance
  - features for context, etc.
    - proximal
    - distant
    - global
  - etc



# Procedure

- Fully convolutional network
  - with very large receptive fields
  - some skip connections
- Train with cross-entropy loss



# Procedure, II

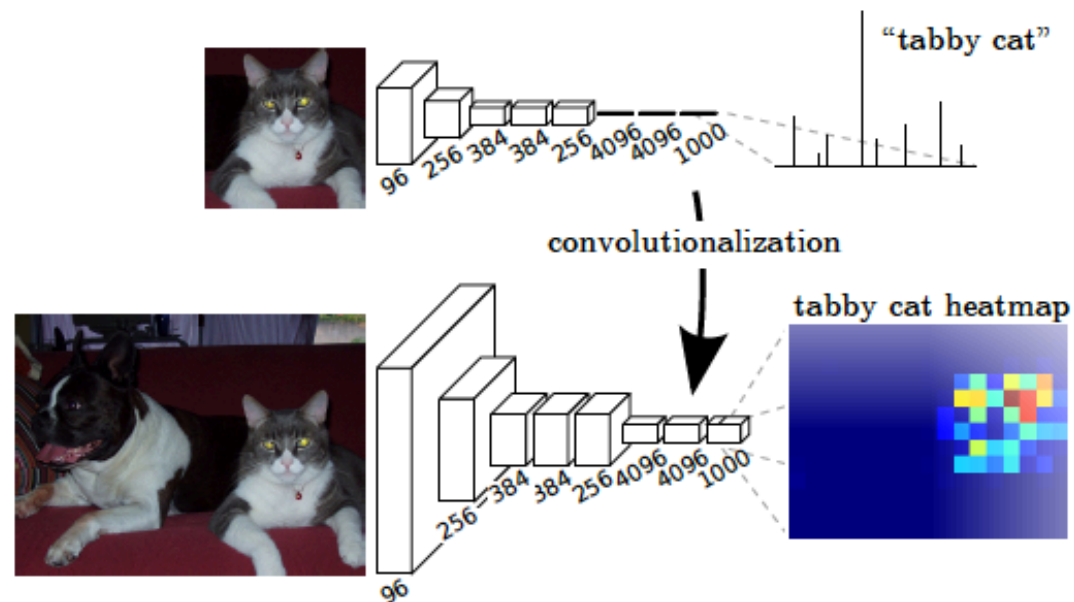


Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.



# Procedure, III

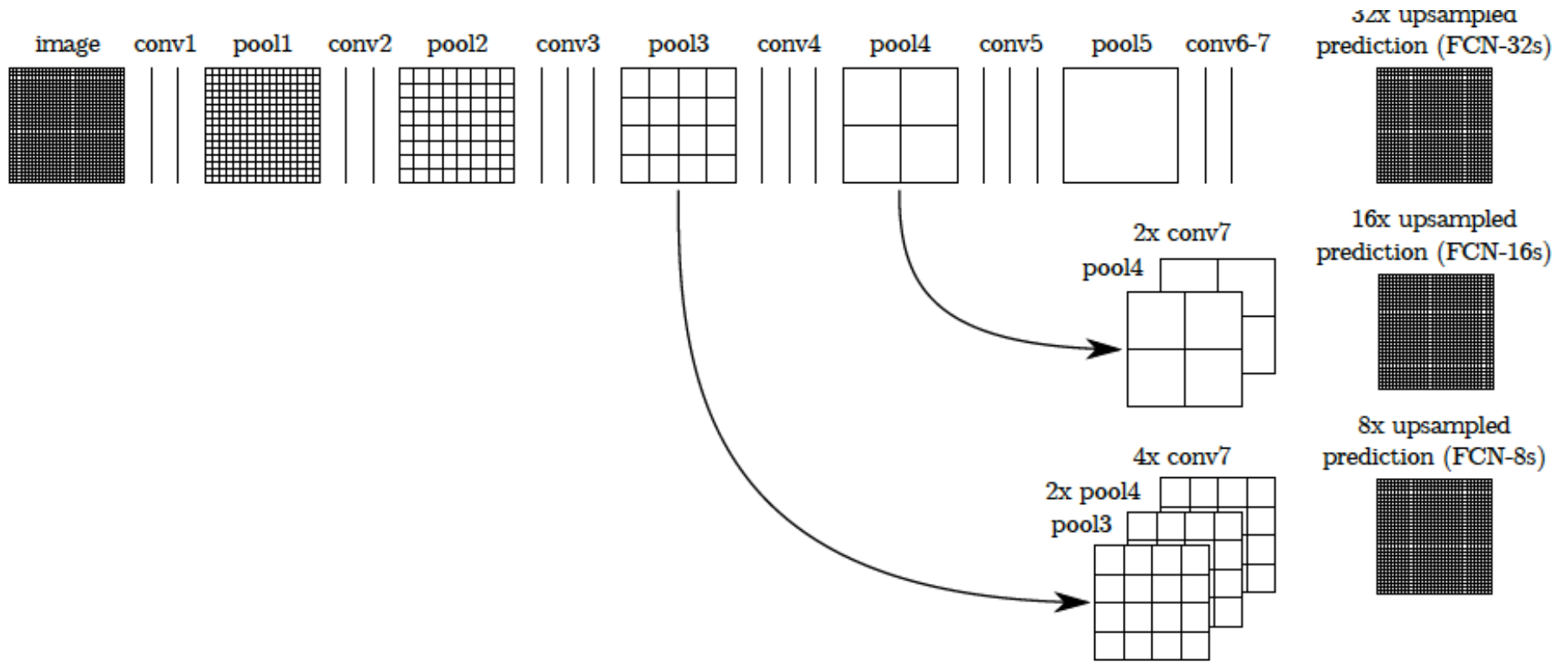


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the pool4 layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from pool3, at stride 8, provide further precision.

# Procedure, IV

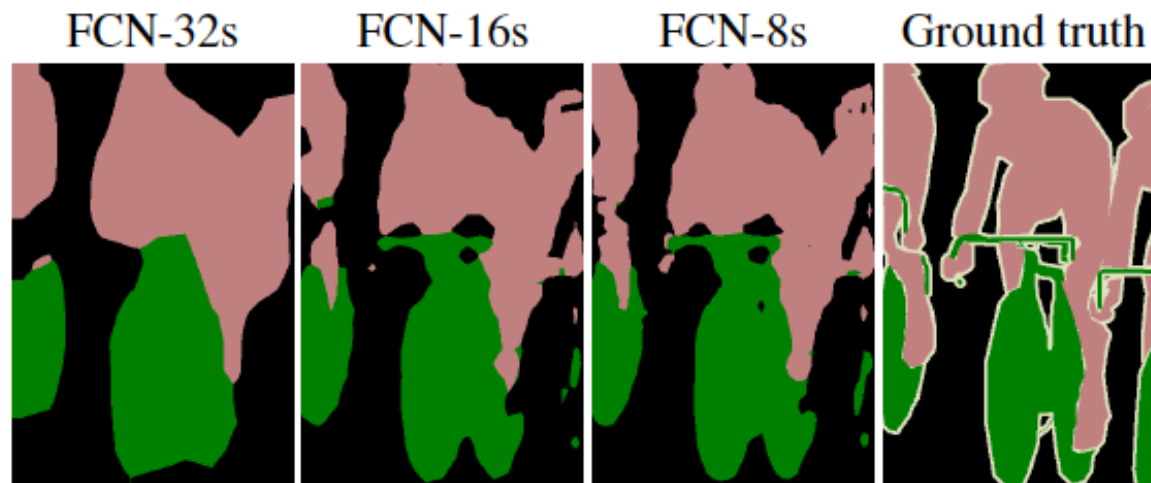


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

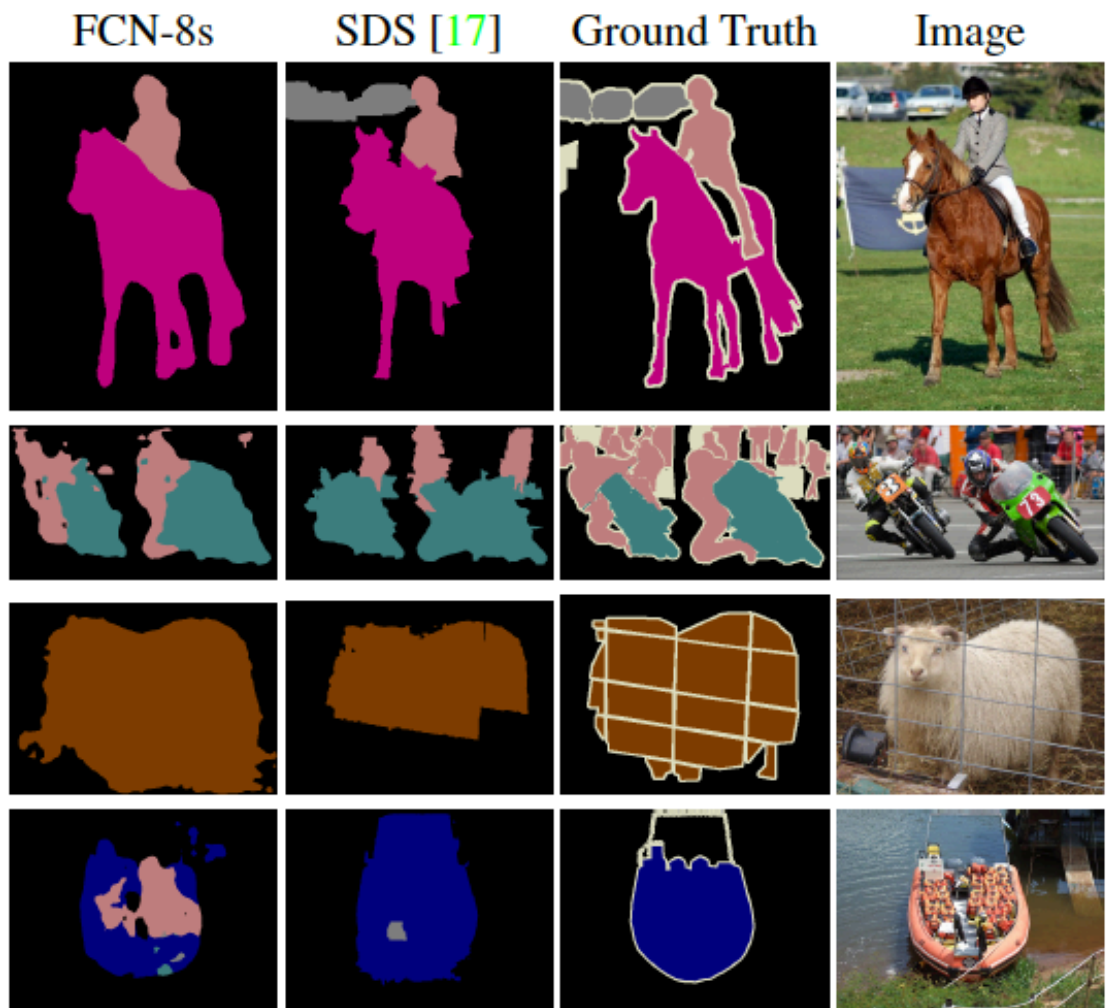


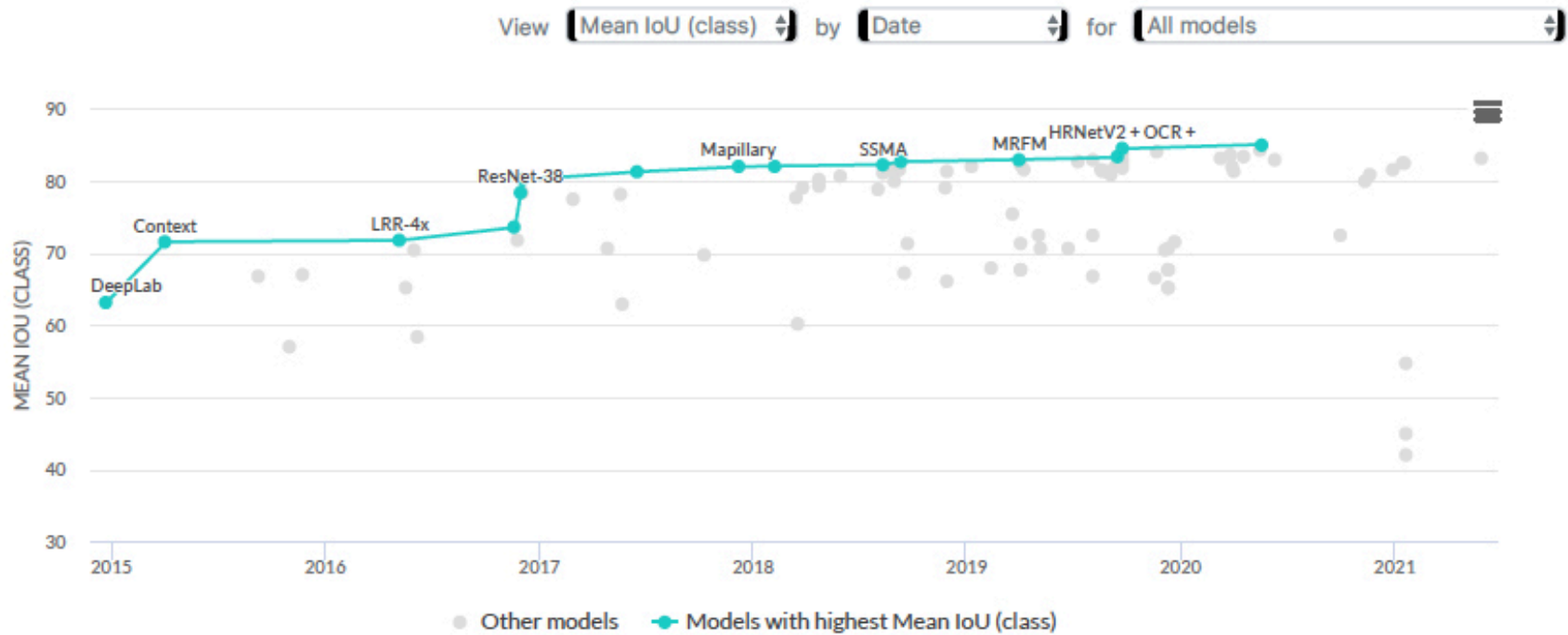
Figure 6. Fully convolutional segmentation nets produce state-of-the-art performance on PASCAL. The left column shows the output of our highest performing net, FCN-8s. The second shows the segmentations produced by the previous state-of-the-art system by Hariharan *et al.* [17]. Notice the fine structures recovered (first row), ability to separate closely interacting objects (second row), and robustness to occluders (third row). The fourth row shows a failure case: the net sees lifejackets in a boat as people.

# SOTA early 22

## Semantic Segmentation on Cityscapes test

[Leaderboard](#)

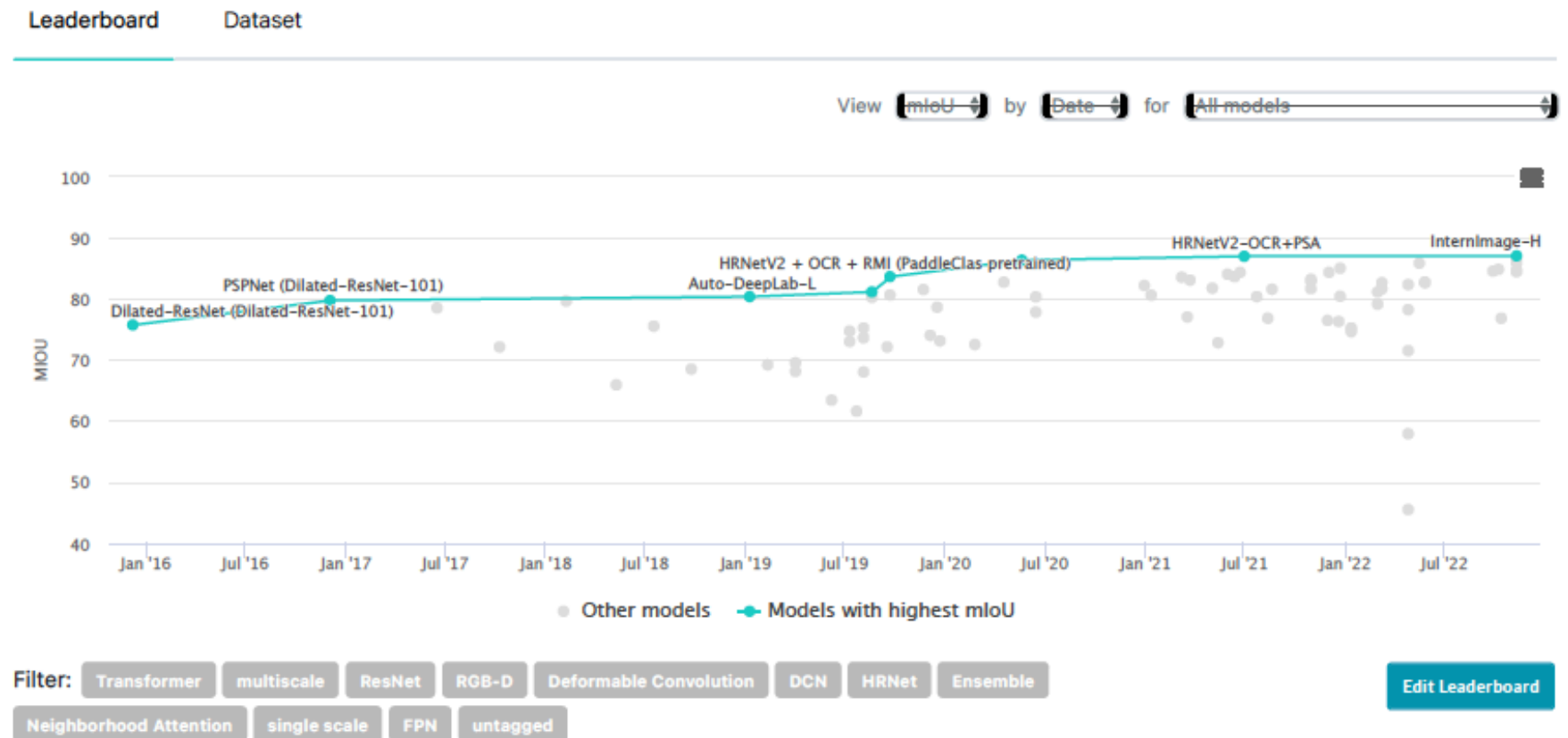
[Dataset](#)



<https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes>

# SOTA - early 23

## Semantic Segmentation on Cityscapes val



<https://paperswithcode.com/sota/semantic-segmentation-on-cityscapes>

# More SOTA

- **Kitti**
  - <http://www.semantic-kitti.org/tasks.html>
- **Robust Vision**
  - <http://www.robustvision.net/leaderboard.php?benchmark=semantic>

# Depth from one image

D.A. Forsyth

# Regression

- We must make image-like things from images
- Running example:
  - depth map from image
- A depth map has the depth to closest surface at every pixel
  - it is the same size as the image



# Monocular depth estimation

- Simplest:
  - compute feature vector at each pixel
  - predict depth from that feature vector
    - linear regression/more complex regression/classification
- Alternative:
  - impose structural model (“it’s a box”)
  - get estimates of parameters
- Current
  - Encode image
  - Decode to depth map

# History

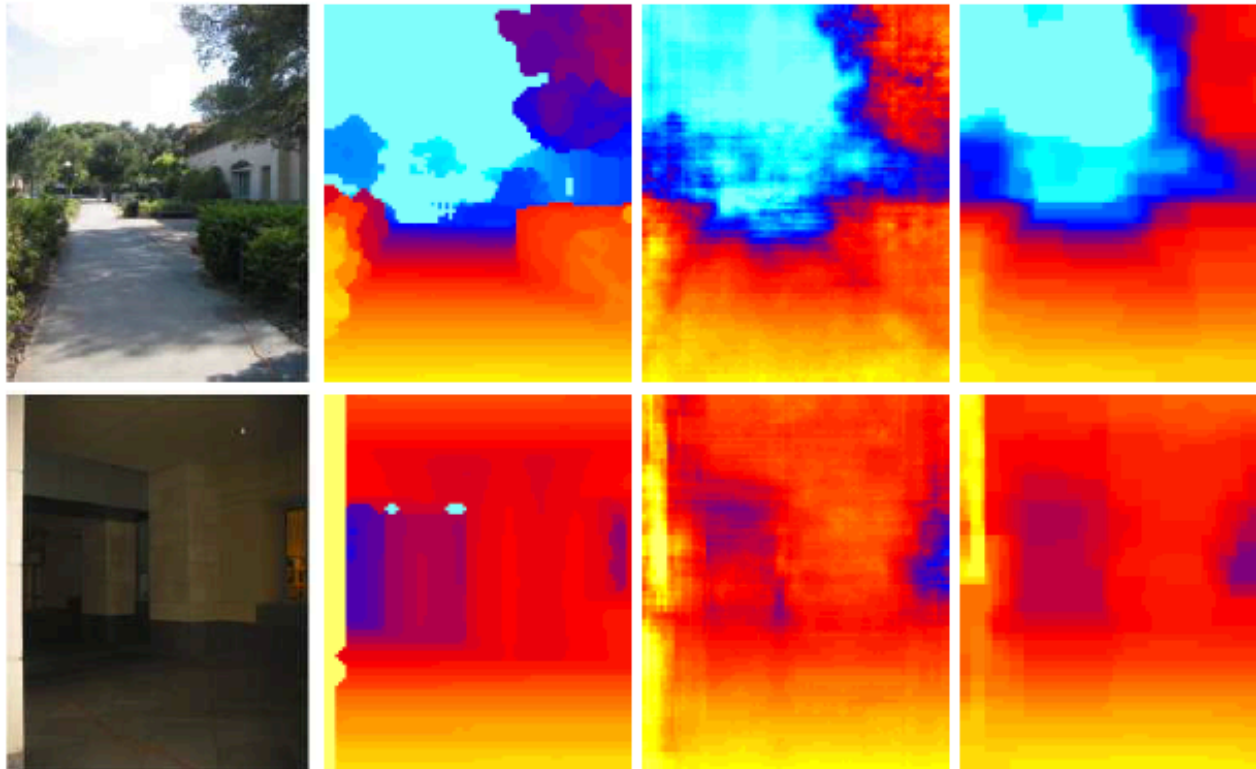


Figure 3: Results for a varied set of environments, showing original image (column 1), ground truth depthmap (column 2), predicted depthmap by Gaussian model (column 3), predicted depthmap by Laplacian model (column 4). (Best viewed in color)

# History

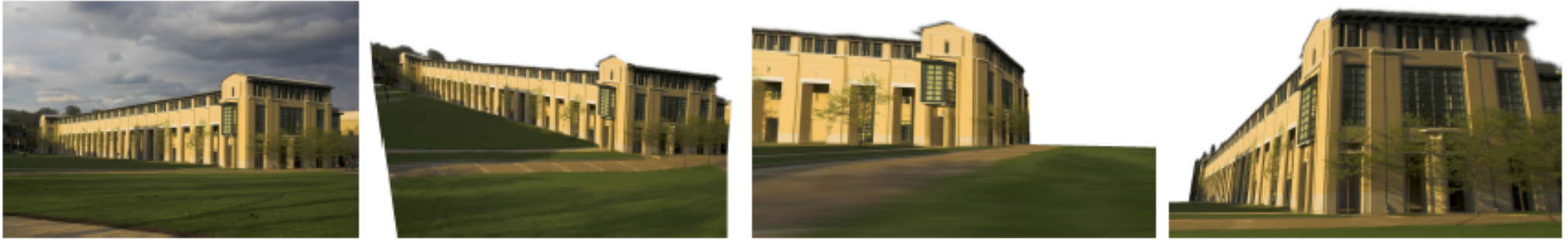


Figure 1: Our system automatically constructs a rough 3D environment from a single image by learning a statistical model of geometric classes from a set of training images. A photograph of the University Center at Carnegie Mellon is shown on the left, and three novel views from an automatically generated 3D model are to its right.

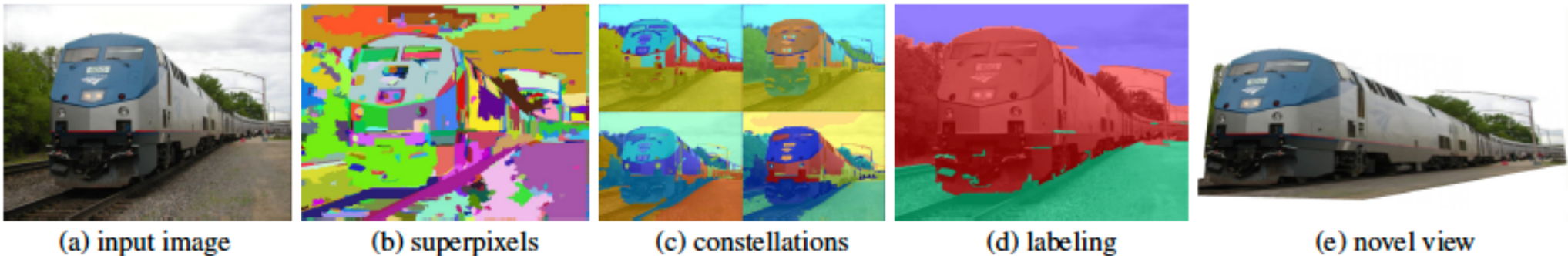


Figure 2: 3D Model Estimation Algorithm. To obtain useful statistics for modeling geometric classes, we must first find uniformly-labeled regions in the image by computing superpixels (b) and grouping them into multiple constellations (c). We can then generate a powerful set of statistics and label the image based on models learned from training images. From these labels, we can construct a simple 3D model (e) of the scene. In (b) and (c), colors distinguish between separate regions; in (d) colors indicate the geometric labels: ground, vertical, and sky.

(Essentially) Classification

Hoiem et al, 05

# History



Figure 2. Our process takes the original image, identifies long line segments (A), and uses these to find vanishing points (B). The line segments in (A) are colored with the color of the vanishing point they vote for. This information, and other features, are used to produce a set of possible layouts, which are ranked by a function learned with structure learning (four are shown in C). The top ranked layouts produce maps of label probabilities (shown in D for “left wall”, “floor”, “right wall” and “object”). In turn these maps are used to re-estimate features, and the re-estimated features are used to produce a second ranking. The top ranked layout for this image is in (E).

(Essentially) Parameter estimation

Hedau et al, 09

# Convolutional encoders

- Apply “pattern detector” to image
  - another to the result
  - another to the result
  - etc
  - occasionally reducing the spatial size of the block of data representing patterns to control redundancy
- The resulting block of data is spatially small

# Convolution

$$\mathcal{N} = \text{conv}(\mathcal{I}, \mathcal{W})$$

where

$$\mathcal{N}_{ij} = \sum_{uv} \mathcal{I}_{i-u, j-v} \mathcal{W}_{uv}.$$

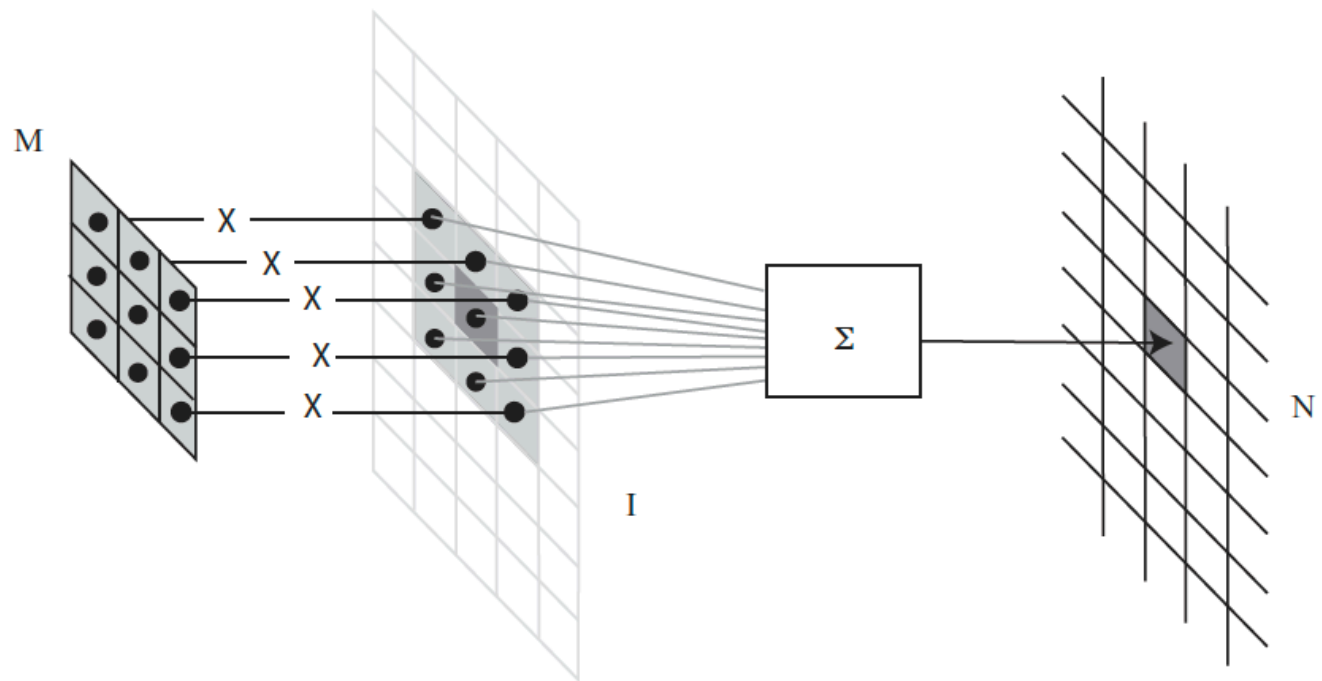


FIGURE 6.1: To compute the value of  $\mathcal{N}$  at some location, you shift a copy of  $\mathcal{M}$  to lie over that location in  $\mathcal{I}$ ; you multiply together the non-zero elements of  $\mathcal{M}$  and  $\mathcal{I}$  that lie on top of one another; and you sum the results.

# Convolution

- Think of this as a form of dot-product
  - between kernel and window
- Like dot-products
  - largest value when kernel matches window
  - smallest when kernel matches window with contrast reversal
- -> **SIMPLE PATTERN DETECTOR!**

Digits

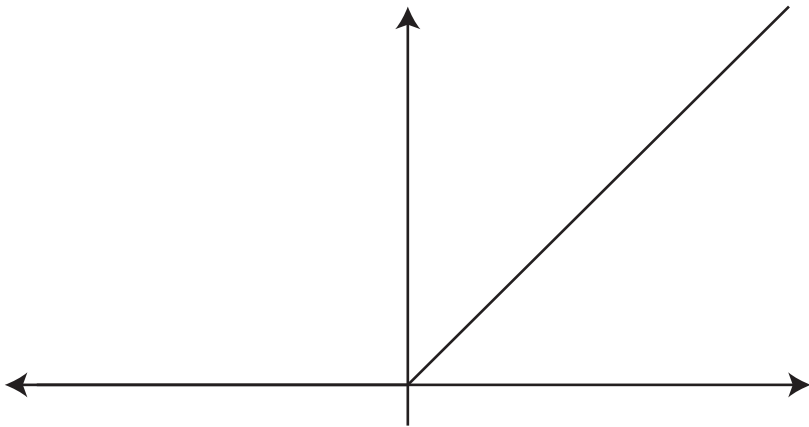
0 1 2 3 4 5 6 7 8 9  
 0 1 2 3 4 5 6 7 8 9  
 0 1 2 3 4 5 6 7 8 9  
 0 1 2 3 4 5 6 7 8 9  
 0 1 2 3 4 5 6 7 8 9

		Convolution output	Test against threshold	Superimposed
Kernels				



# The ReLU

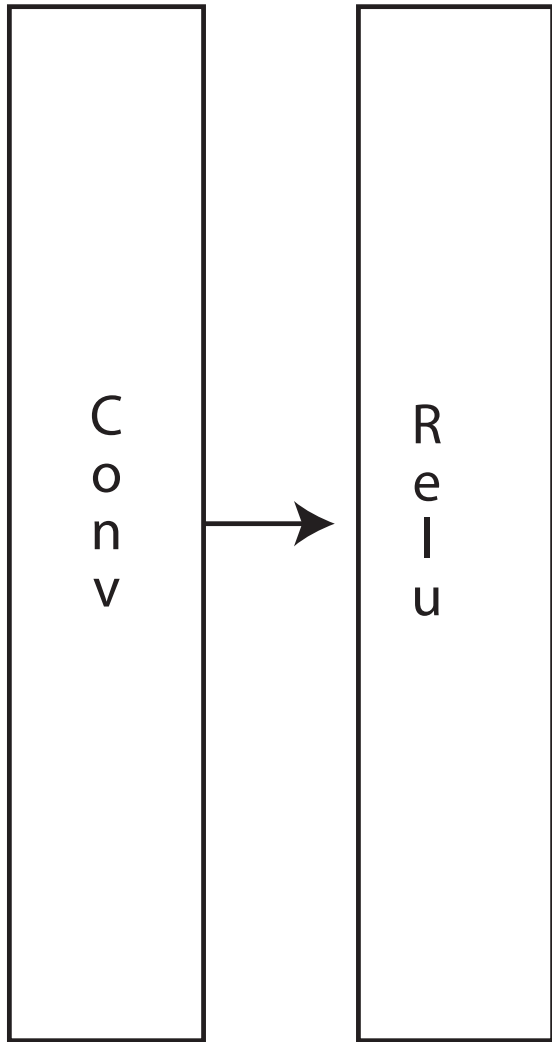
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



Issue: contrast reversal in pattern

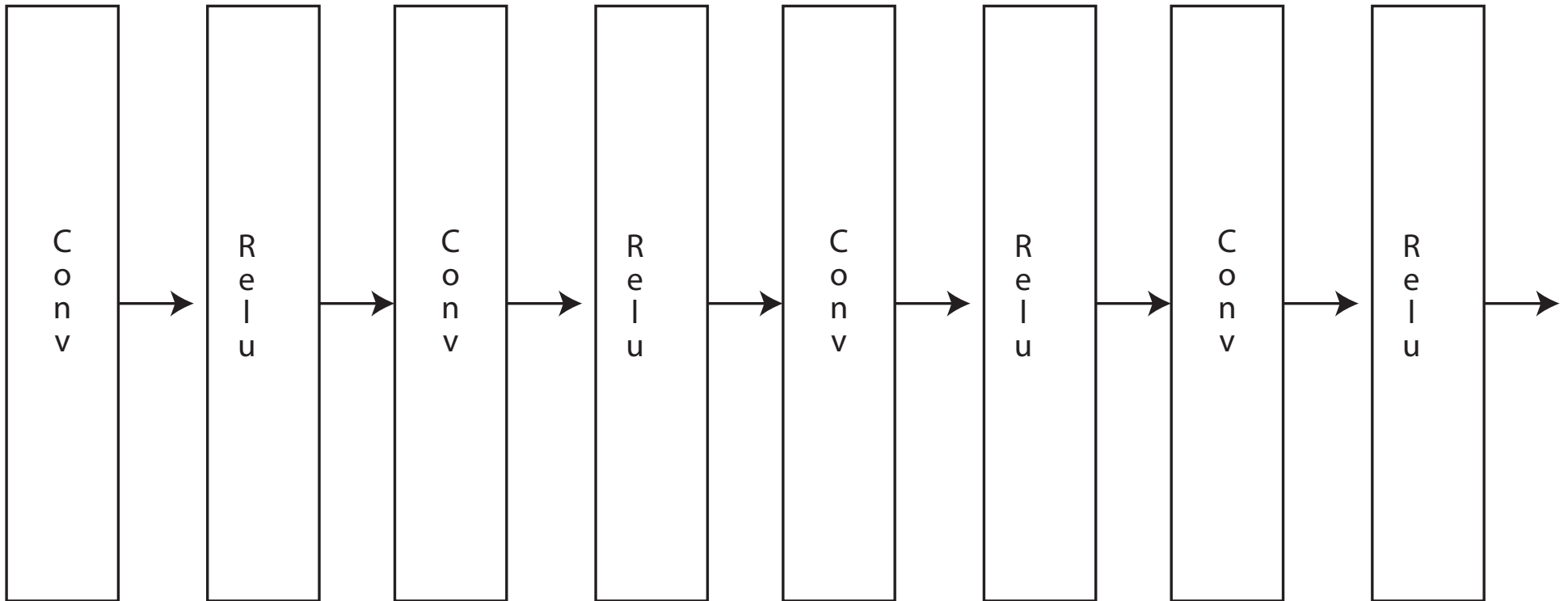
If we apply a relu to a conv, then we have a *\*signed\** pattern detector

# Basic pattern detector

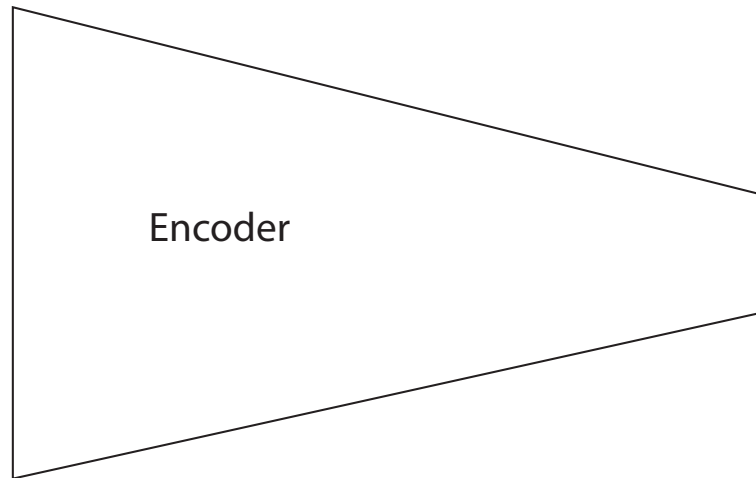
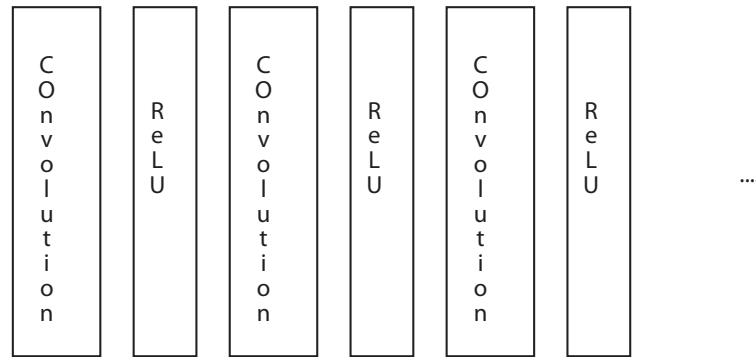


Notice - not very many parameters  
detects the same pattern at each location

# Patterns of patterns of patterns....



# Encoders



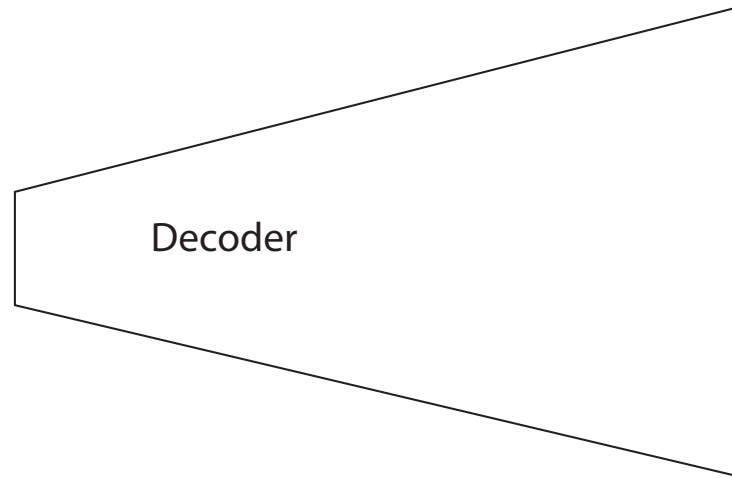
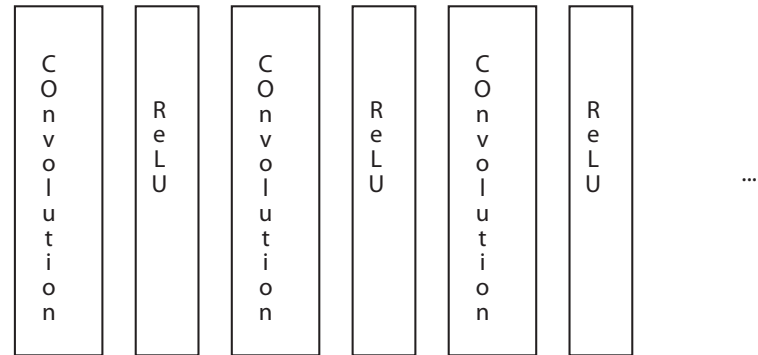
# Convolutional encoders

- Apply “pattern detector” to image
  - another to the result
  - another to the result
  - etc
  - occasionally reducing the spatial size of the block of data representing patterns to control redundancy
- The resulting block of data is spatially small

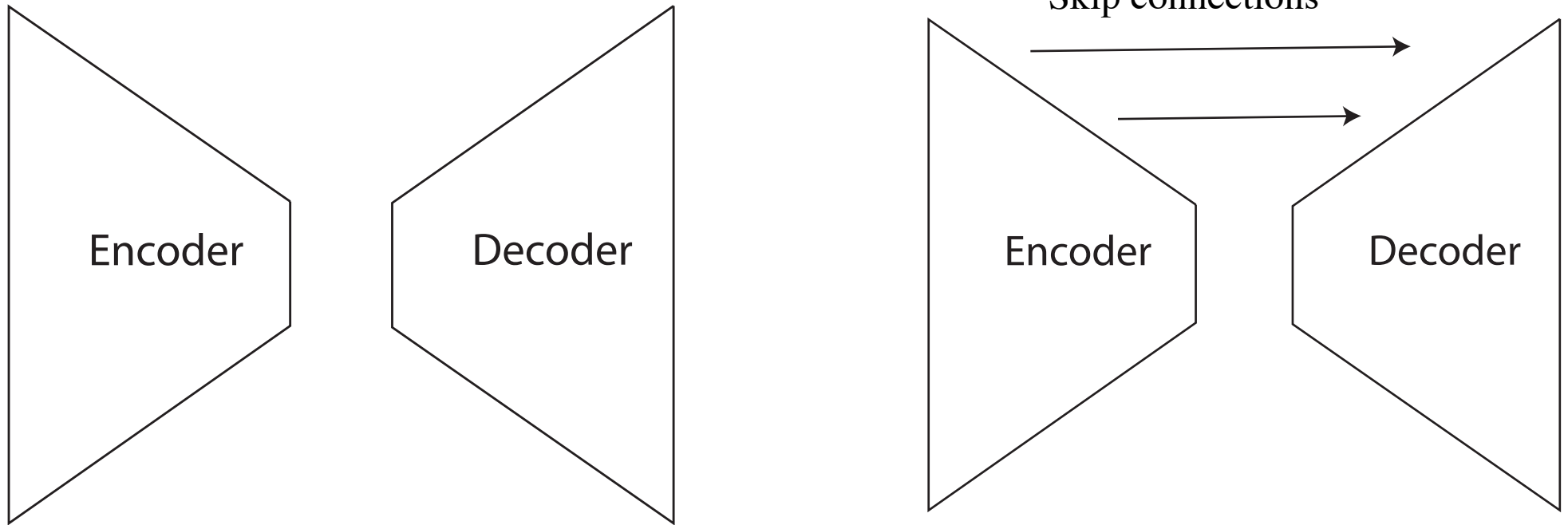
# We could now predict an image by..

- Take pattern detector results and decode into pattern
  - “pattern producer”
- Apply pattern producer to feature block
  - another to result
  - another to result
  - occasionally upsampling as required
- Pattern producer is itself a convolution
  - a feature location detects a particular pattern
  - scale that pattern by the strength of the response, and place down
  - sum at overlap
  - => convolution (sometimes called transpose convolution, inverse convolution)

# Decoders



# Regression



Sometimes known as a U-net



# Regression

- Train with pairs (image, depth)
  - Loss
    - Squared error +abs value of error+other terms as required
- Very powerful general recipe
  - depth from image
  - normal from image
  - superresolution
  - etc.
- Variants
  - more sophisticated encoder

# Monocular Depth Estimation

236 papers with code · 14 benchmarks · 20 datasets

Monocular Depth Estimation is the task of estimating the depth value (distance relative to the camera) of each pixel given a single (monocular) RGB image. This challenging task is a key prerequisite for determining scene understanding for applications such as 3D scene reconstruction, autonomous driving, and AR. State-of-the-art methods usually fall into one of two categories: designing a complex network that is powerful enough to directly regress the depth map, or splitting the input into bins or windows to reduce computational complexity. The most popular benchmarks are the KITTI and NYUv2 datasets. Models are typically evaluated using RMSE or absolute relative error.

Source: [Defocus Deblurring Using Dual-Pixel Data](#)

## Datasets

Lift from

<https://paperswithcode.com/task/monocular-depth-estimation>

### Benchmarks

Add a Result

These leaderboards are used to track progress in Monocular Depth Estimation

Trend	Dataset	Best Model	Paper	Code	Compare
	KITTI Eigen split	DwinFormer			<a href="#">See all</a>
	NYU-Depth V2	VPD			<a href="#">See all</a>
	KITTI Eigen split unsupervised	PlaneDepth			<a href="#">See all</a>
	Make3D	GCNDepth			<a href="#">See all</a>
	Mid-Air Dataset	M4Depth-d6 (VMD)			<a href="#">See all</a>
	IBims-1	LeReS			<a href="#">See all</a>
	Middlebury 2014	Miangoleh et al. (MiDaS)			<a href="#">See all</a>
	KITTI	MonoViT			<a href="#">See all</a>
	UASOL	FCRN-DepthPrediction from Iro Laina et al. (2016)			<a href="#">See all</a>
	SUN-RCBD	RPSF			<a href="#">See all</a>

Show all 14 benchmarks

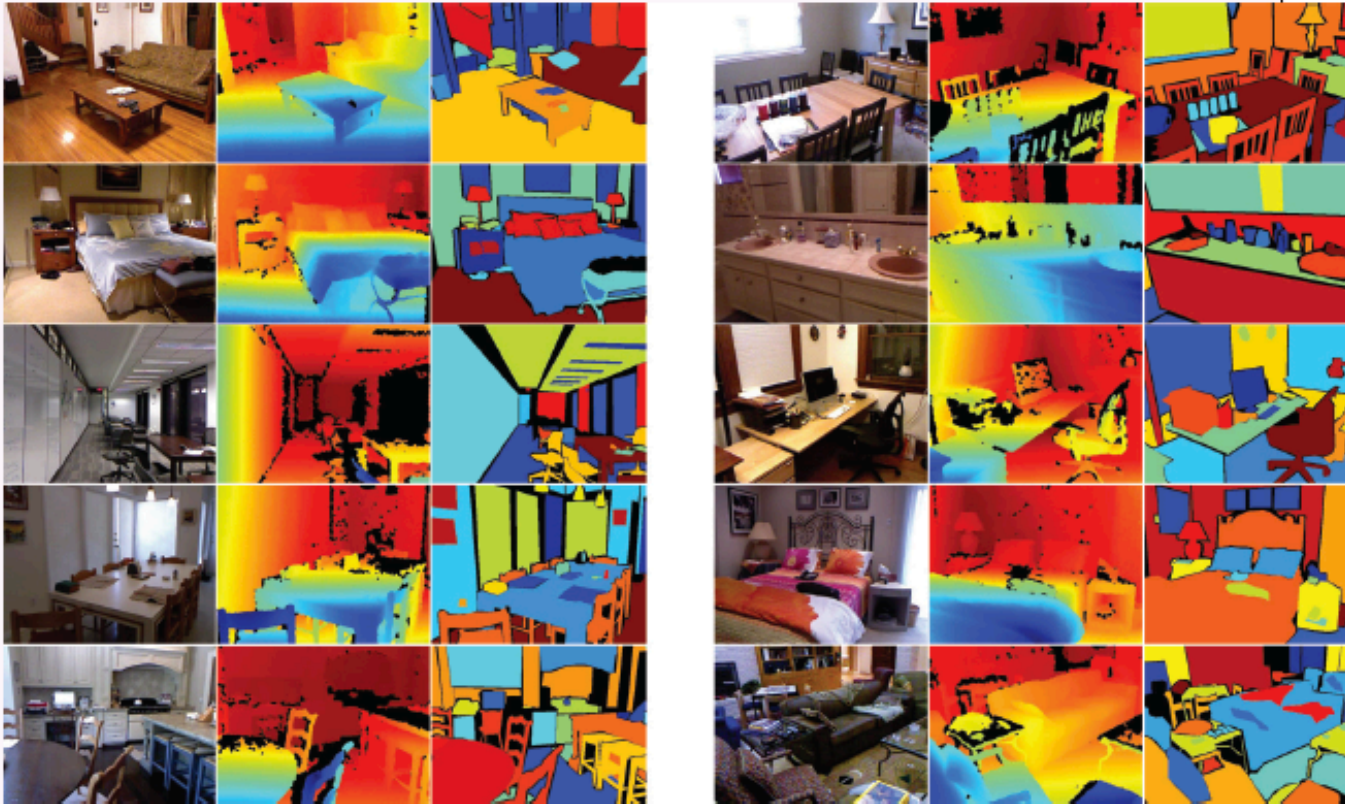
# Typical Dataset NYU-V2

## NYU Depth Dataset V2

Nathan\_Silberman, Pushmeet\_Kohli, Derek\_Hoiem, Rob\_Fergus

If you use the dataset, please cite the following work:

Indoor Segmentation and Support Inference from RGBD Images  
ECCV 2012 [PDE][Bib]



Samples of the RGB image, the raw depth image, and the class labels from the dataset.

# How to evaluate?

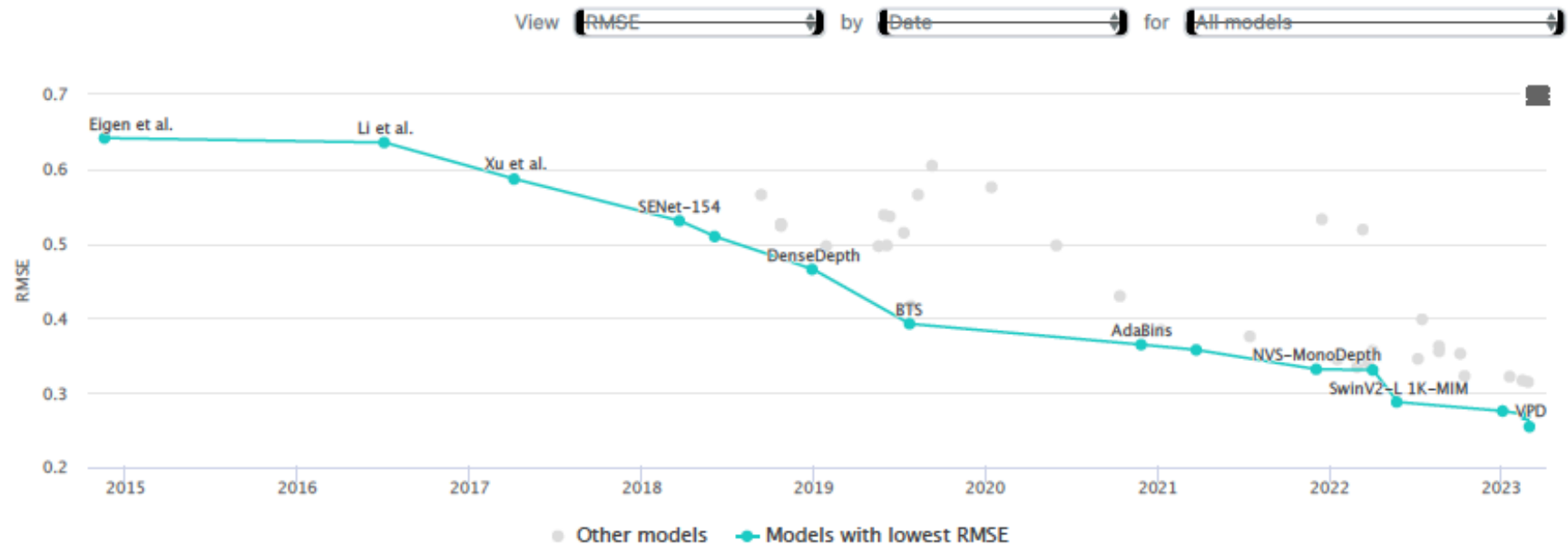
- RMSE in depth is often used
  - root mean square error
  - but presents problems:
    - typically dominated by errors in large depths
    - which may not be all that important
- AbsRel:
  - mean Abs([error/depth])
    - likely better unless depth range is relatively small

# SOTA (early 23)

## Monocular Depth Estimation on NYU-Depth V2

Leaderboard

Dataset



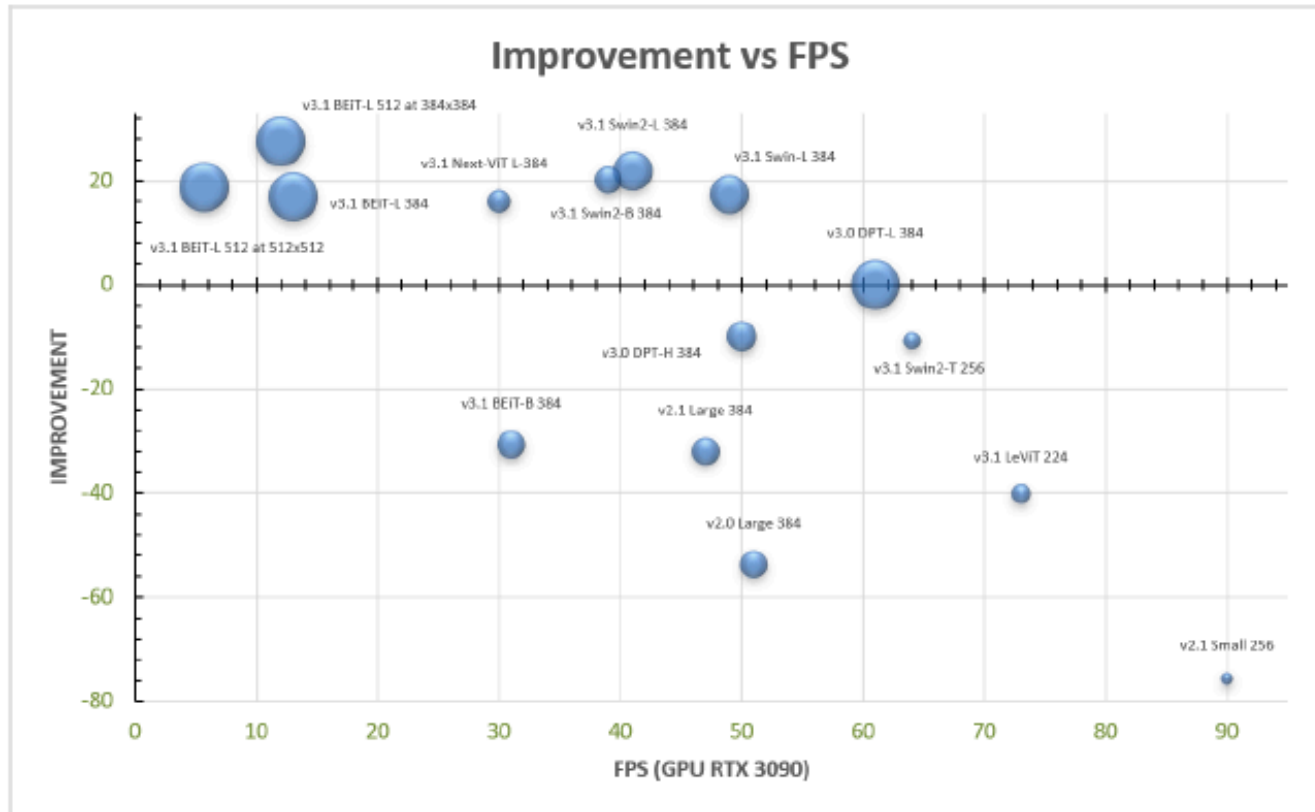
<https://paperswithcode.com/sota/monocular-depth-estimation-on-nyu-depth-v2>

# What to predict?

- Q:
  - Should we predict absolute depth?
    - on what scale?
  - Or relative depth?
  - Or what?
- Considerations:
  - somewhat depends on application and dataset
  - large depths don't really occur in some cases
  - in others, they can be relatively rare \*and\* dominate the error
  - small errors in small depths mostly worse than small errors in large depths

# Midas (one useful example, w/github)

MiDaS was trained on up to 12 datasets (ReDWeb, DIML, Movies, MegaDepth, WSVD, TartanAir, HRWSI, ApolloScape, BlendedMVS, IRS, KITTI, NYU Depth V2) with multi-objective optimization. The original model that was trained on 5 datasets (MIX 5 in the paper) can be found [here](#). The figure below shows an overview of the different MiDaS models; the bubble size scales with number of parameters.



# What to predict?

- Midas predicts
  - $a*(1/\text{depth})+b$
  - where a, b are unknown constants dependent on scene
  - So if you want true depth, you need to estimate these
    - very seldom important
  - Advantage of  $1/d$ 
    - (note: min depth  $> 0$ )
    - small errors in small depths are emphasized



# Omnimap

- Huge collection of 3D scanned data+images
- <https://omnidata.vision>
- Maybe current SOTA on many depth/normal predictions

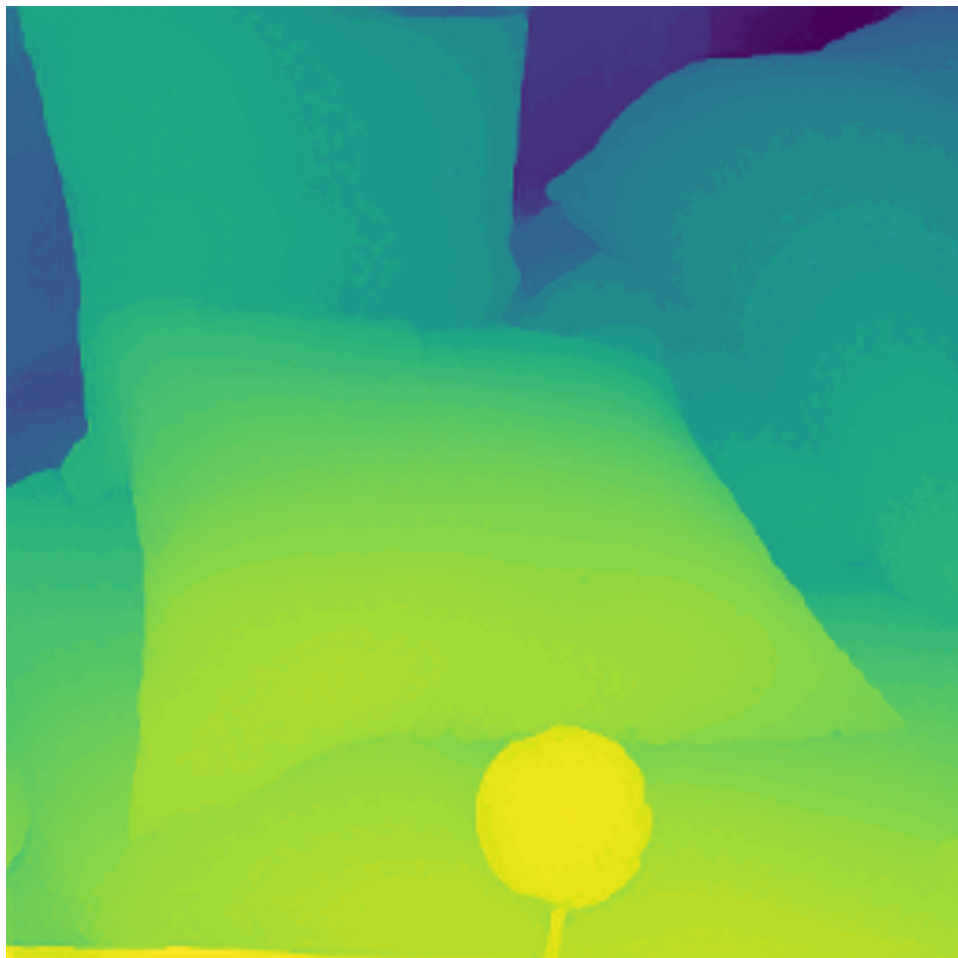
# Notes

- Depth prediction is quite strongly affected by lighting
- Implies you should be able to predict normal
  - True: usually better done using a normal predictor, below

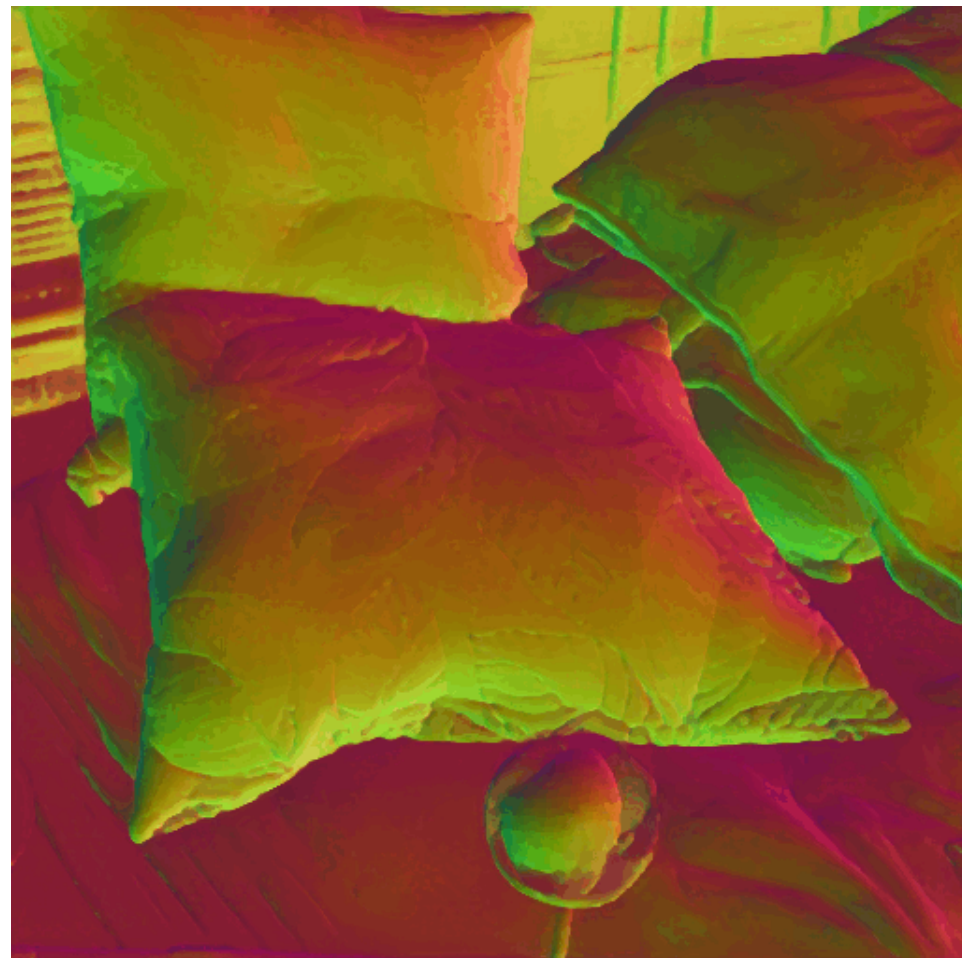


# Lighting problems

Depth (omnimap, current bestish depth est)



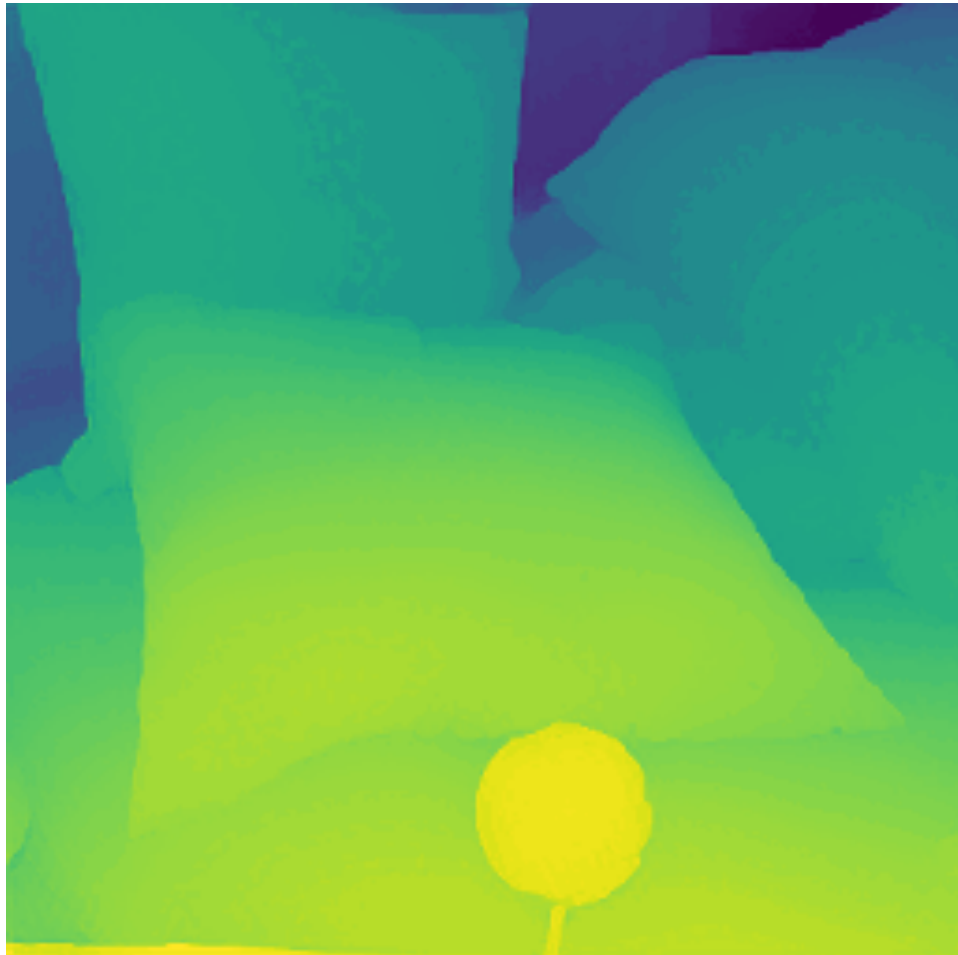
Normal (omnimap, current bestish normal est)





# Lighting problems

Depth (omnimap, current bestish depth est)



# Normal from one image

D.A. Forsyth

# Monocular Normal Estimation

- Deep history
  - line labelling

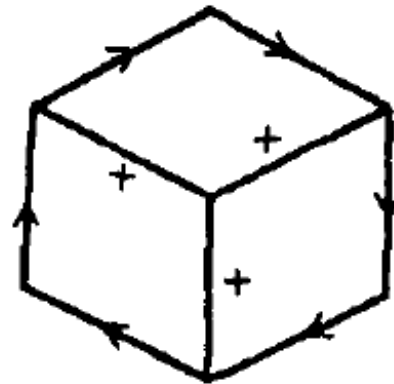
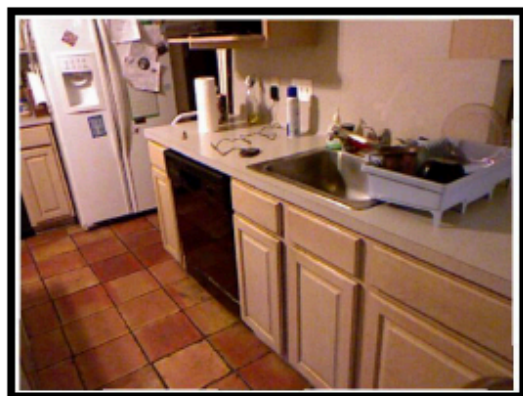


FIG. 6. Huffman-Clowes-Waltz labeling representing a cube-like configuration.

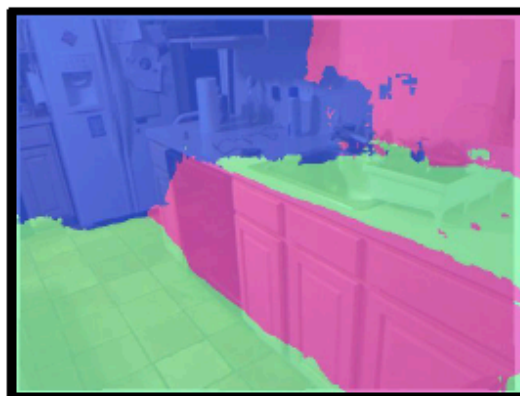
From Kanade, 1980, after Huffman, 71; Clowes, 71; Waltz 72

# Monocular normal estimation

- Simplest:
  - compute feature vector at each pixel
  - predict normal from that feature vector
    - linear regression/more complex regression/classification
- Alternative:
  - impose structural model
    - “there are only three normals”=Manhattan world
    - decide
      - what the directions are; which pixel has which dirn
- Current
  - Encode image
  - Decode to normal map



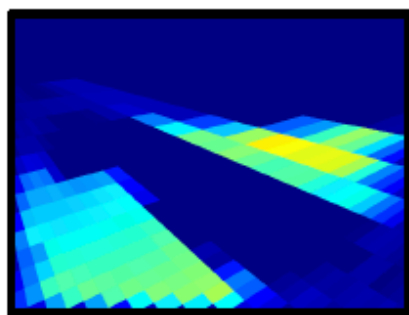
Single RGB Image



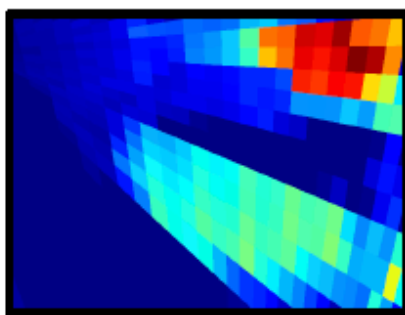
Local Surface Normals



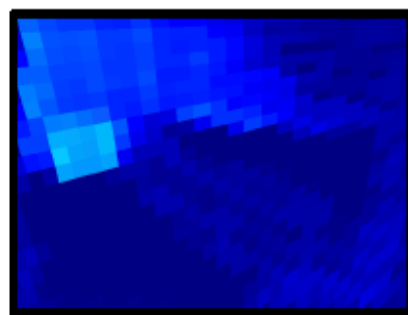
Discrete Scene Parse



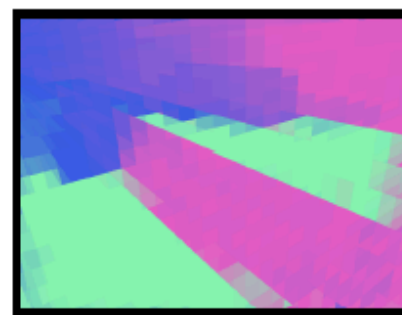
Direction 1



Direction 2



Direction 3



Continuous Interpretation

Surface Normals with Mid-level Constraints

Fig. 1. We propose the use of mid-level constraints from the line-labeling era and a parametrization of indoor layout to “unfold” a 3D interpretation of the scene in the form of large planar surfaces and the edges that join them. In contrast to local per-pixel normals, we return a discrete parse of the scene in terms of surfaces and the edges between them in the style of Kanade’s Origami World as well updated continuous evidence integrating these constraints. Normal legend: blue  $\rightarrow$  X; green  $\rightarrow$  Y; red  $\rightarrow$  Z. Edge Legend: convex +; concave -. Figures best viewed in color.



# History

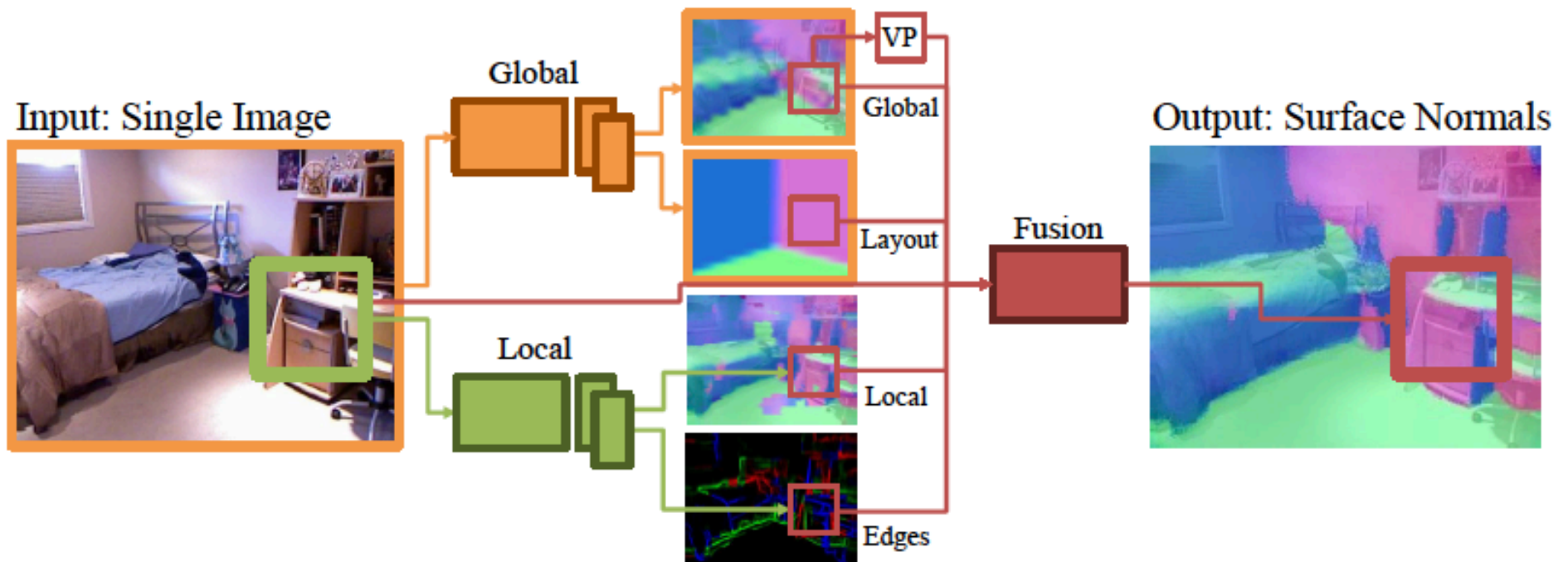


Figure 2: An overview of our approach to predicting surface normals of a scene from a single image. We separately learn global and local processes and use a fusion network to fuse the contradictory beliefs into a final interpretation. **Global processes:** our network predicts a coarse  $20 \times 20$  structure and a vanishing-point-aligned box layout from a set of discrete classes. **Local processes:** our network predicts a structured local patch from a part of the image and line-labeling classes: **convex-blue**, **concave-green**, and **occlusion-red**. **Fusion process:** our network fuses the outputs of the two input networks, the rectified coarse normals with vanishing points (VP) and images to produce substantially better results.

Make local predictions of normal, edges; global predictions of layout, normals; fuse

# Surface Normals Estimation

# Datasets

22 papers with code • 6 benchmarks • 8 datasets

Surface normal estimation deals with the task of predicting the surface orientation of the objects present inside a scene. Refer to [Designing Deep Networks for Surface Normal Estimation \(Wang et al.\)](#) to get a good overview of several design choices that led to the development of a CNN-based surface normal estimator.




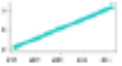













Lift from

<https://paperswithcode.com/task/surface-normals-estimation>

## Benchmarks

[Add a Result](#)

These leaderboards are used to track progress in Surface Normals Estimation

Trend	Dataset	Best Model	Paper	Code	Compare
	PCPNet	Hsurf			<a href="#">See all</a>
	ScanNetV2	Bae et al.			<a href="#">See all</a>
	NYU Depth v2	Bae et al.			<a href="#">See all</a>
	Taskonomy	X-TC (Cross-Task Consistency)			<a href="#">See all</a>
	NYU-Depth V2 Surface Normals	DSN			<a href="#">See all</a>
	PASCAL Context	InvPT			<a href="#">See all</a>

# Evaluation

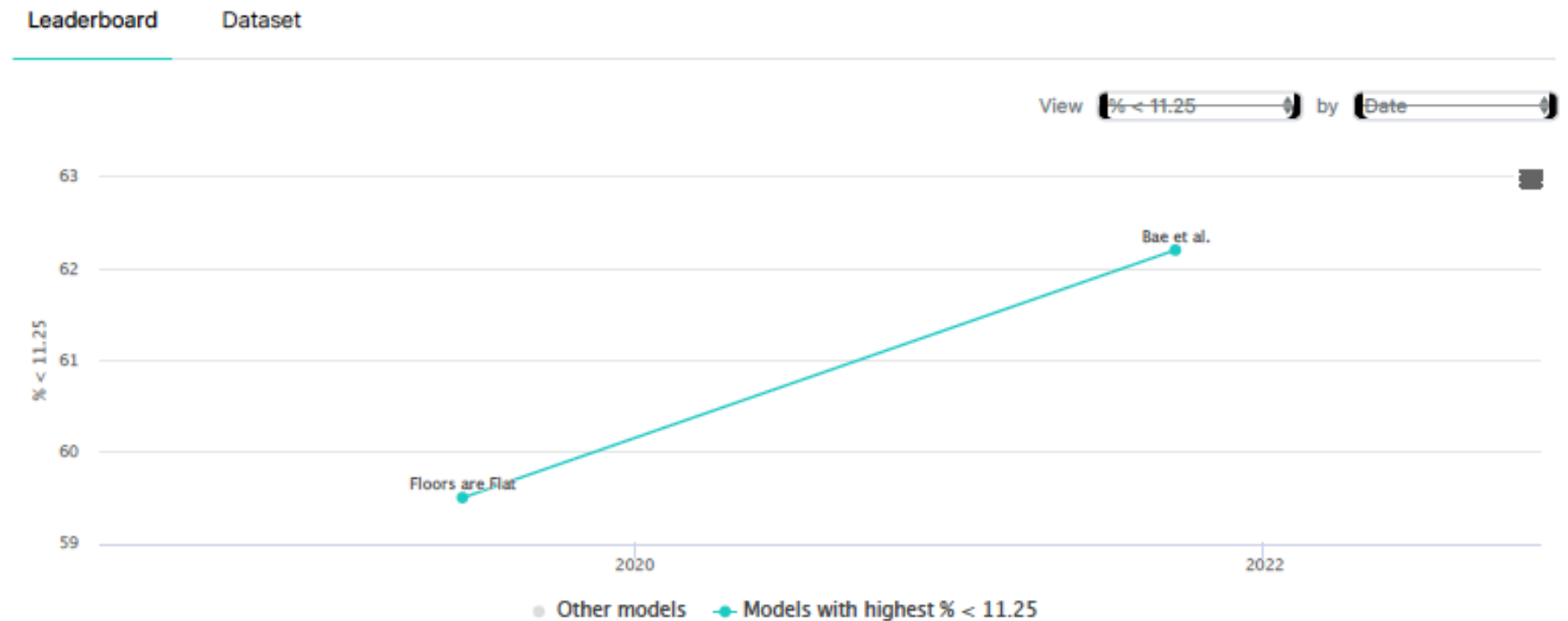
- Usual to report
  - % predicted normals within some range of angle of ground truth
  - for various ranges
- Don't get mixed up by:
  - numerical issues ( $\arccos(x)$  is complex for  $x > 1$ ,  $x < -1$ !)
  - degrees vs radians
    - usually, angles in degrees

Useful info in:

<https://web.eecs.umich.edu/~fouhey/2016/evalSN/evalSN.html>

# SOTA (early 23)

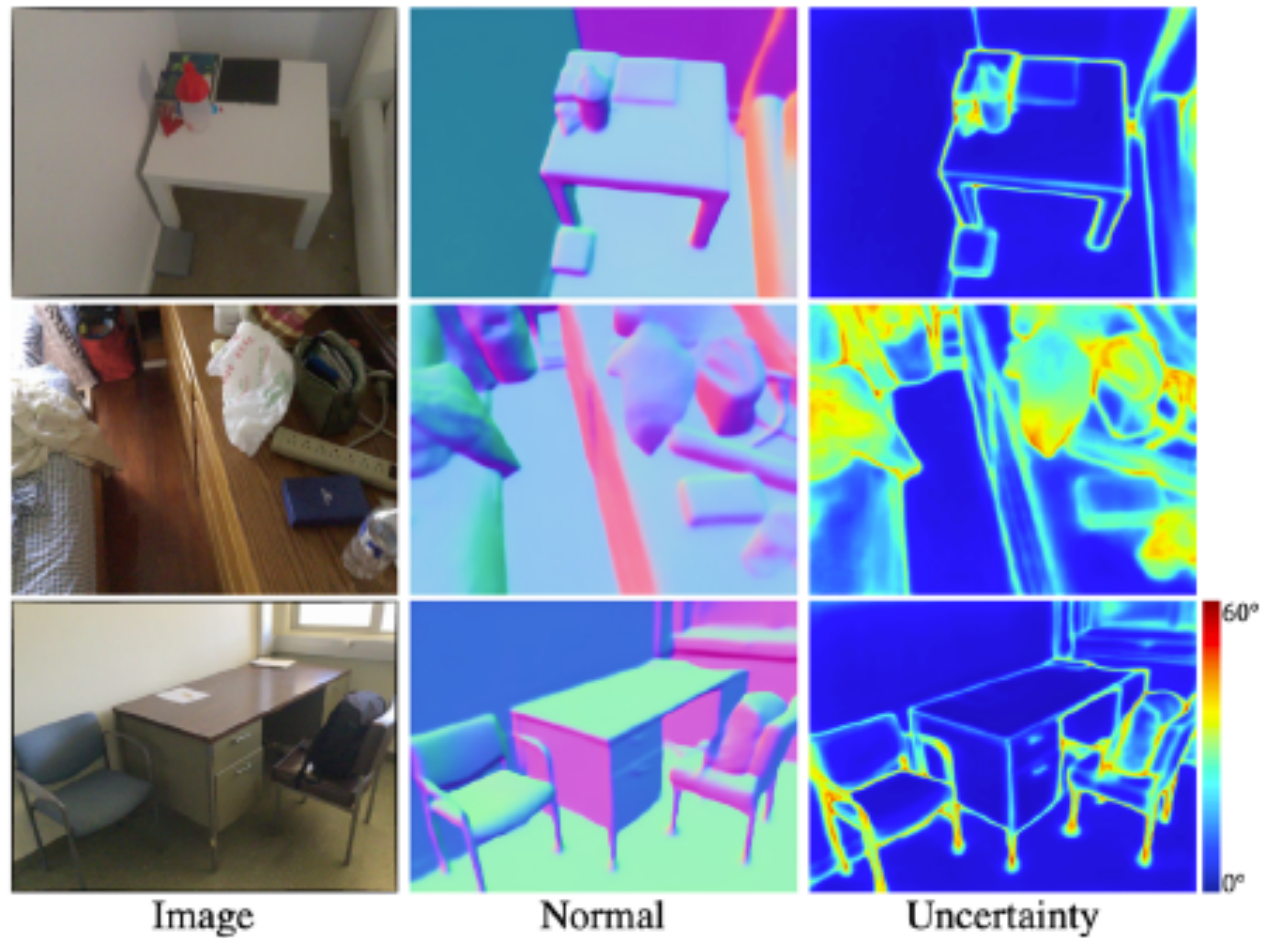
## Surface Normals Estimation on NYU Depth v2



<https://paperswithcode.com/sota/surface-normals-estimation-on-nyu-depth-v2-1>

# SOTA-ish estimator

[https://github.com/baegwangbin/surface\\_normal\\_uncertainty](https://github.com/baegwangbin/surface_normal_uncertainty)



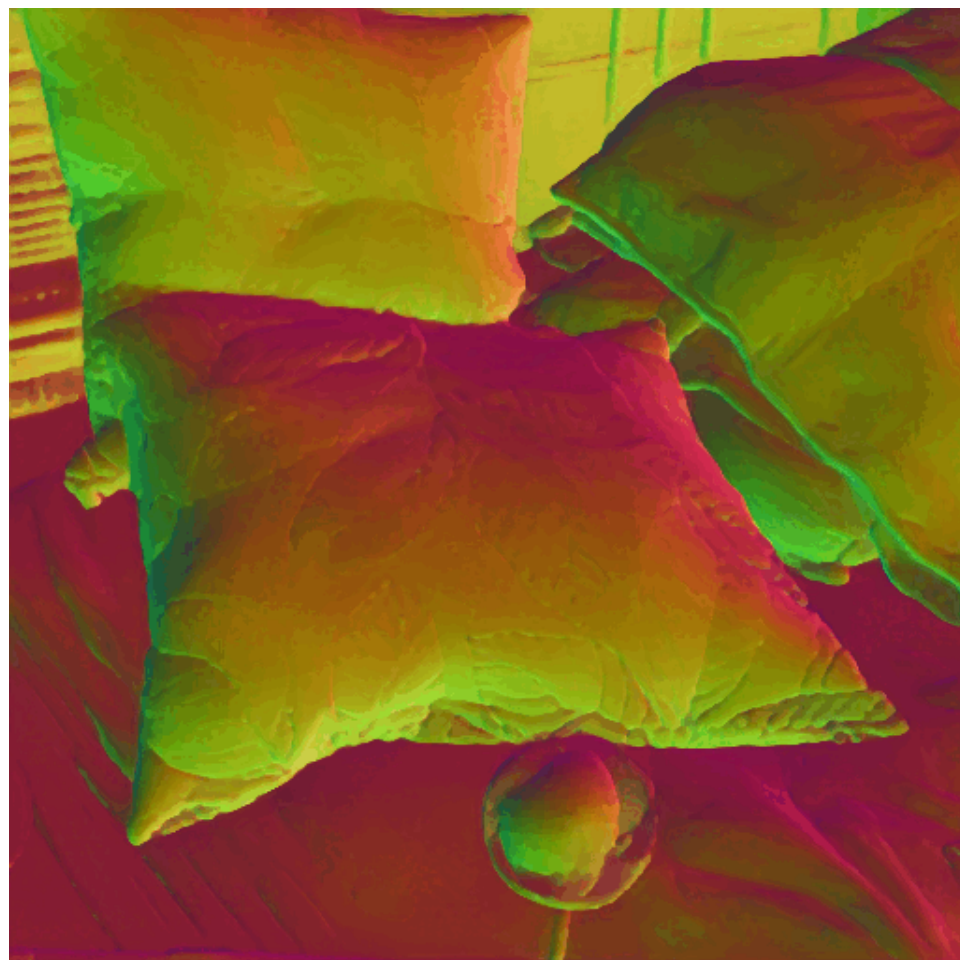
# Omnimap

- Huge collection of 3D scanned data+images
- <https://omnidata.vision>
- Maybe current SOTA on many depth/normal predictions



# Lighting problems

Normal (omnimap, current bestish normal est)



# Inpainting (super quick!)

D.A. Forsyth, UIUC



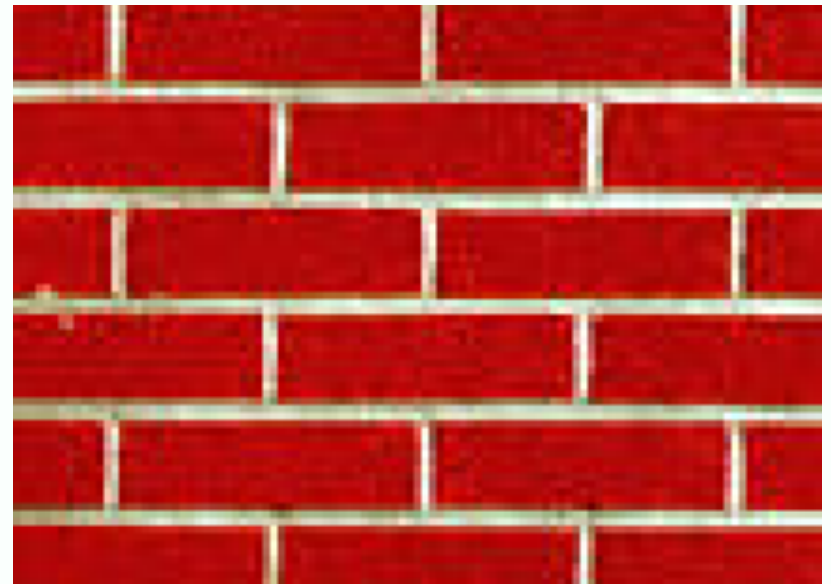
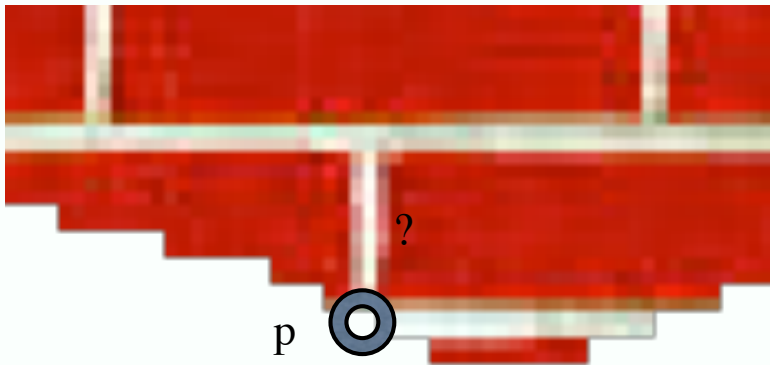
# General inpainting

- Find sensible replacements for missing pixels
  - eg fill in text; fill in removed object; etc
- History:
  - Local non-parametric models of pixels conditioned on neighbors
    - with many variants
  - Patch replacement
    - some variants
  - Denoising autoencoders
    - huge number of variants

# Local Non-parametric models

- Q: Conditioned on a window of known pixels
  - what value should this pixel take?
- A: Match surrounding windows to collect examples
  - choose at random from collection

# How to paint this pixel?

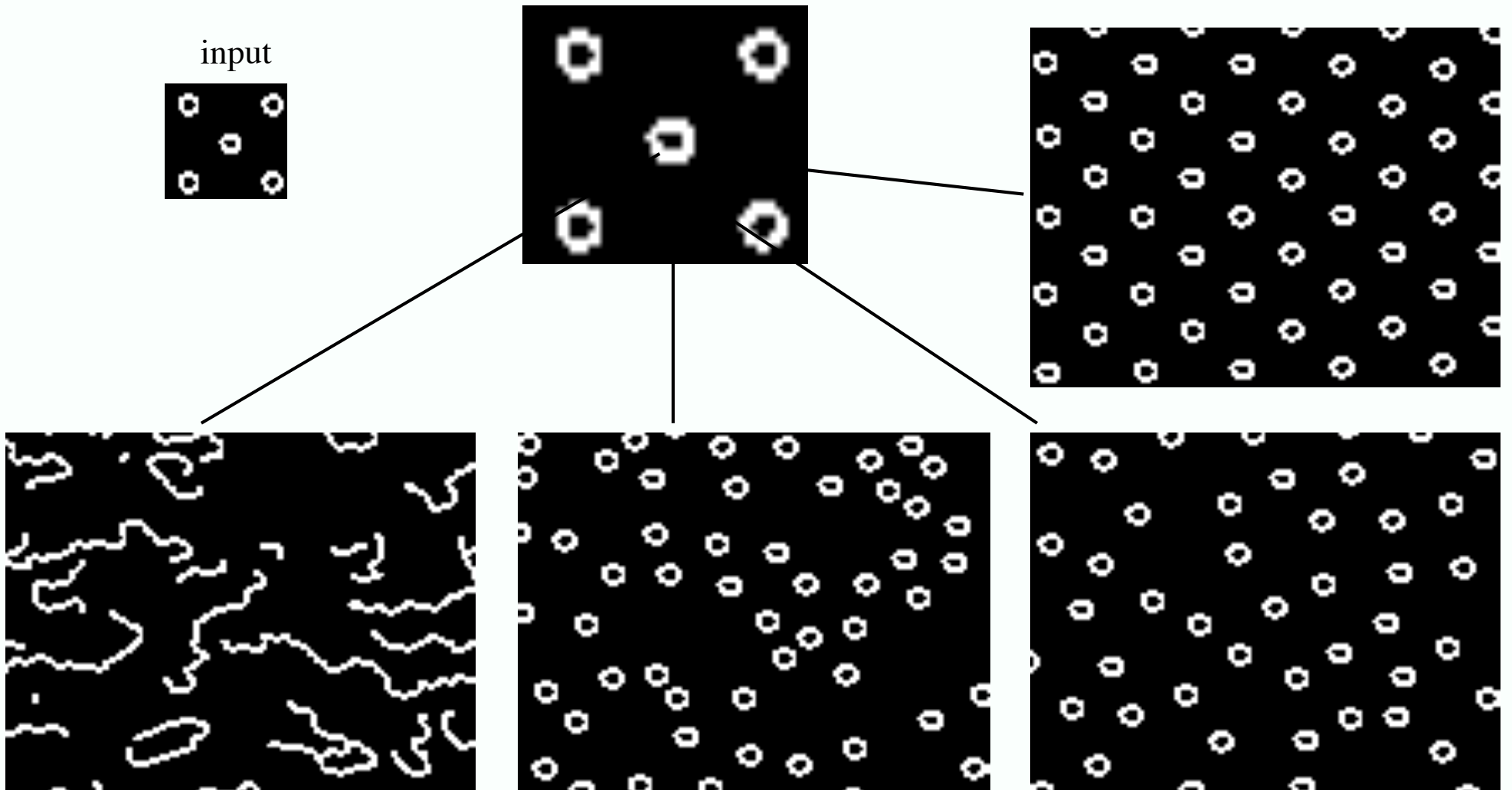


Input texture

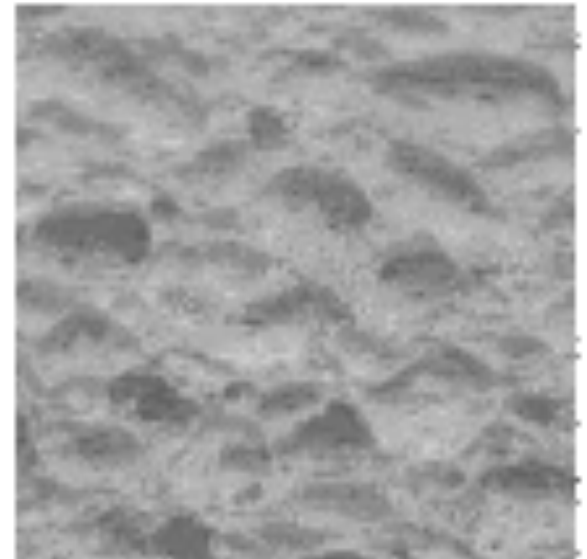
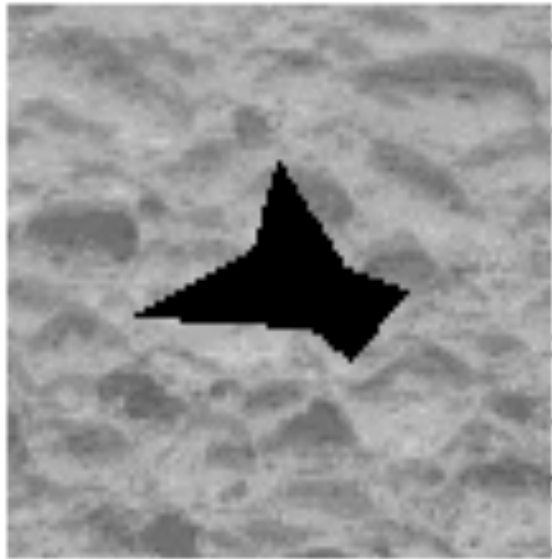
# Concerns

- Distance metric
- Neighborhood size
- Order to paint

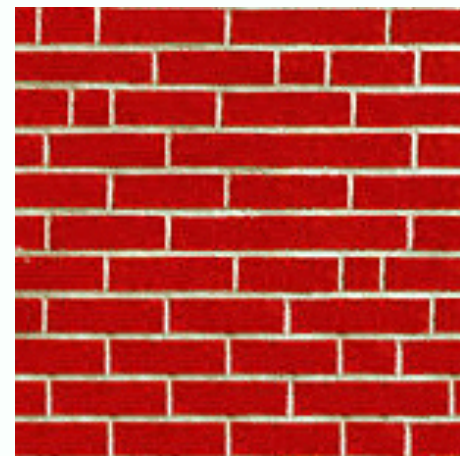
# Neighborhood size



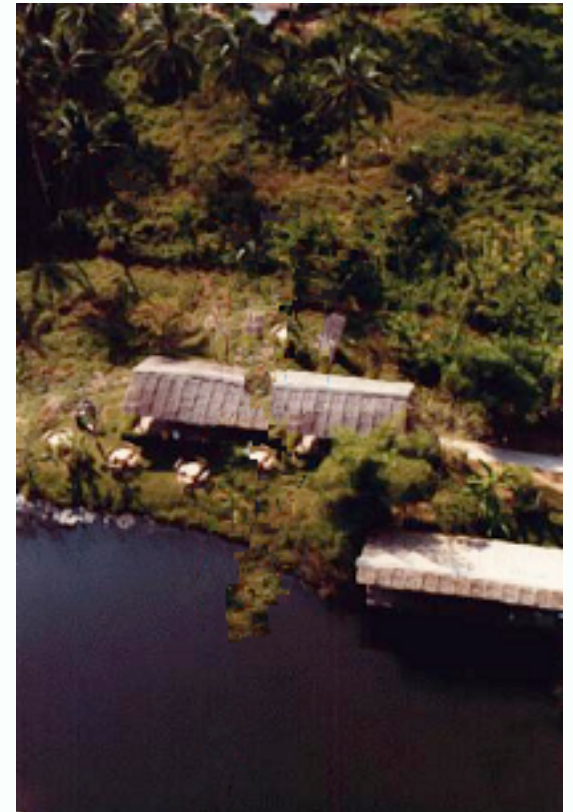
# Growing Regions Hole Filling



# Hole Filling

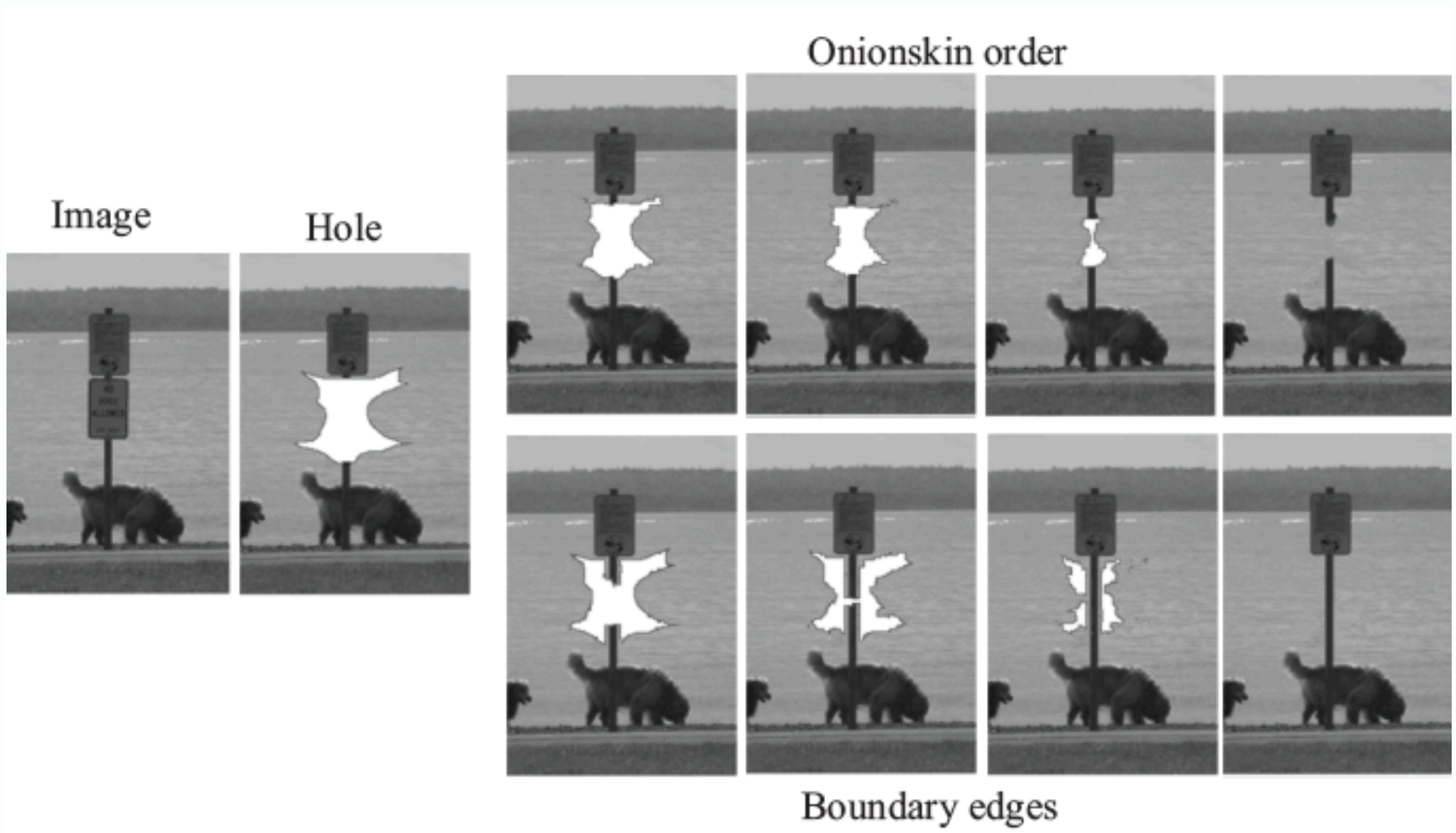


# Inpainting





# Order of inpainting matters

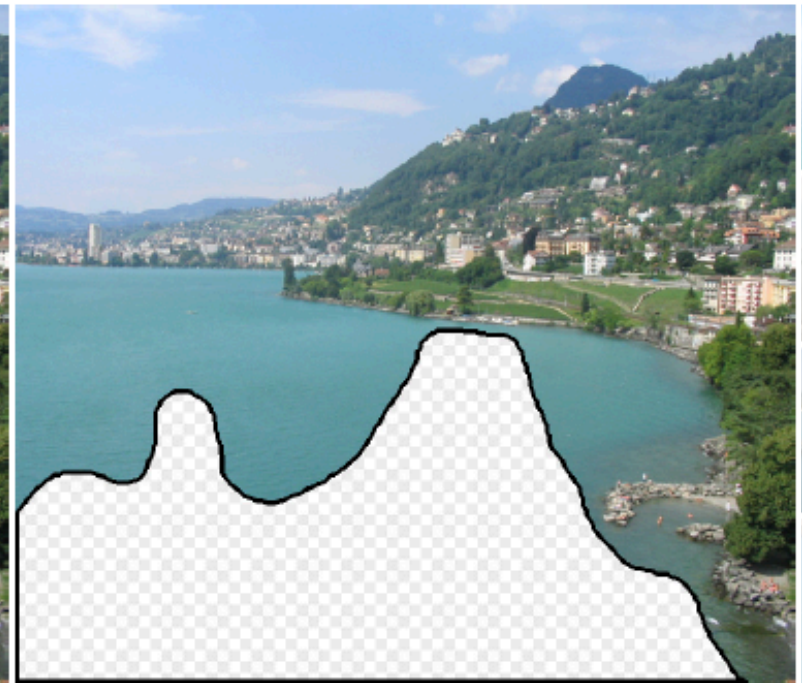


# Long Scale Non-parametric models

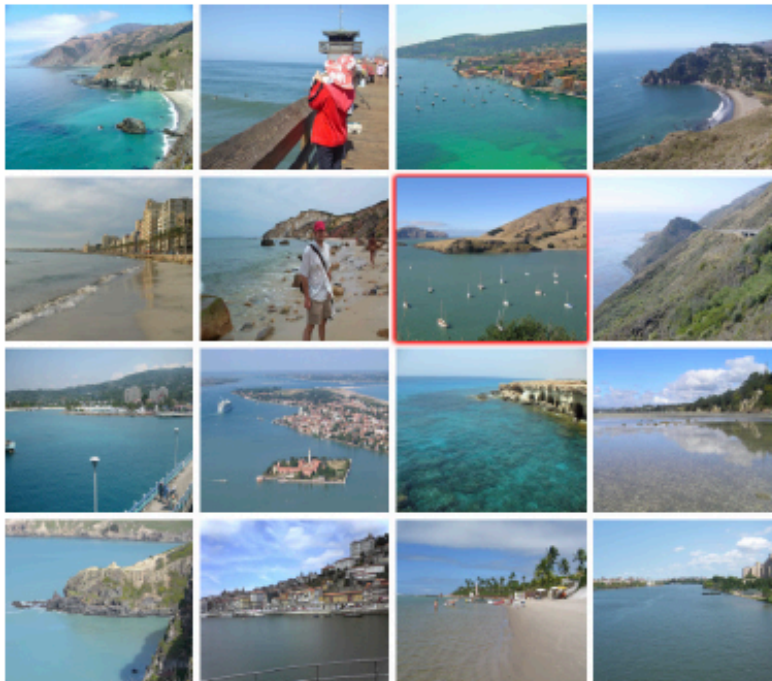
- Q: Conditioned on most of the image
  - what does the missing bit look like?
- A: Match image to others to collect examples
  - choose “at random” from collection
    - variants: how you choose; how you match



Original Image



Input



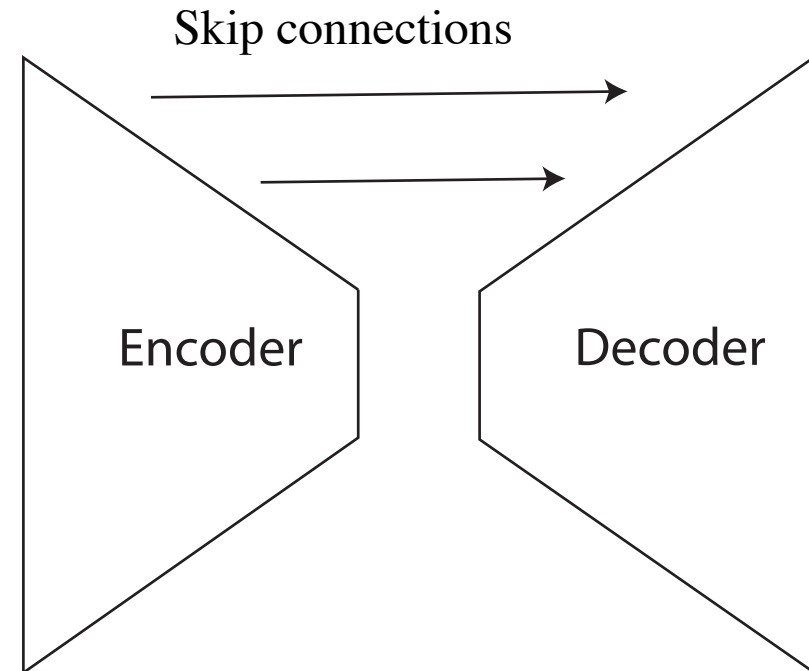
Scene Matches



Output

# Denoising autoencoders

- Use an encoder-decoder stack
  - predict denoised image from noisy image



- Training requires some care
  - skip connections useful for improved resolution BUT
    - make it easy for network to cheat
- Immense number of variants

# Issues

- Noise process
  - depends on application
- Architecture
  - the decoder could be more sophisticated
    - some form of vector quantization, eg VQVAE
  - the encoder could be more sophisticated
    - eg multiple scales; transformer, etc.
- Loss
  - (how do we compare output image to input image)
  - L1/L2
  - VGG
    - compute a set of deep features and match those

# For example...

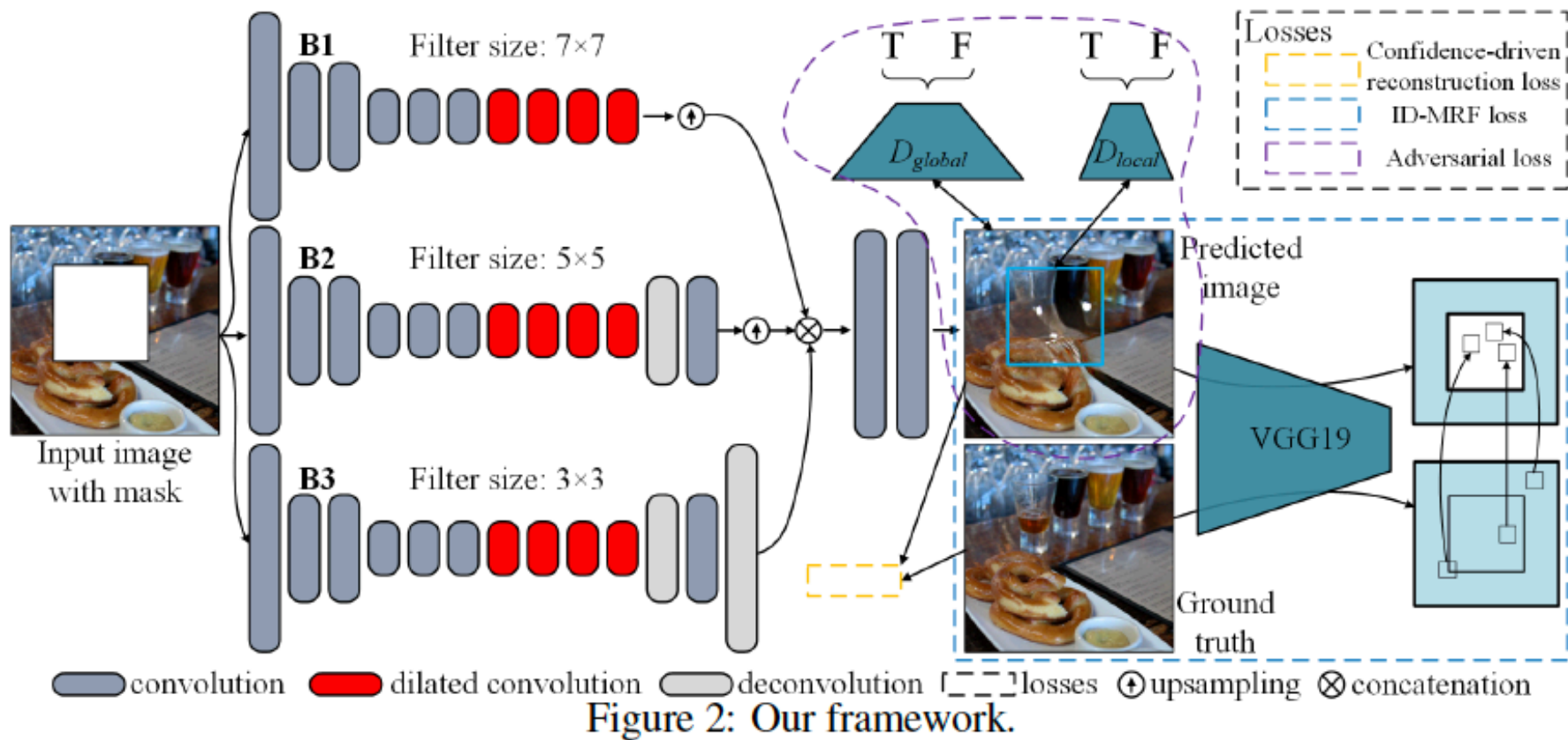
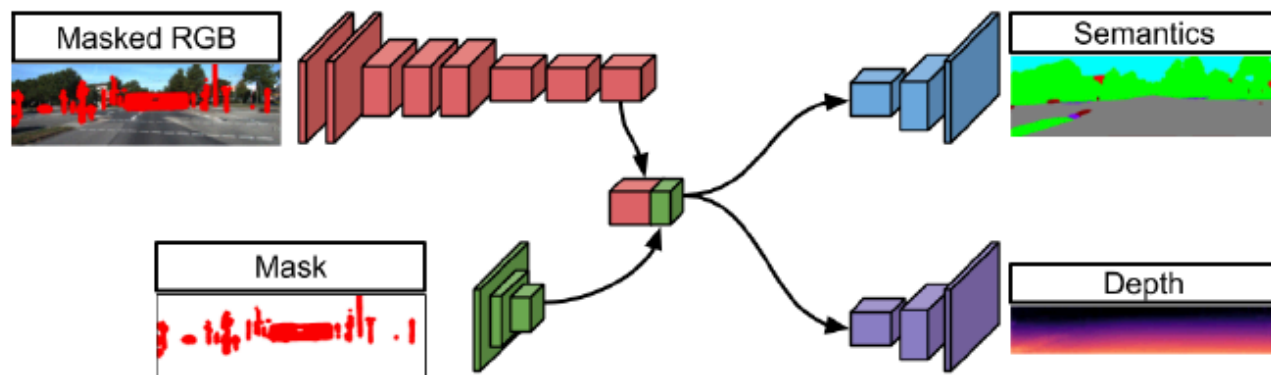




Figure 1: Our inpainting results on building, face, and natural scene.

# Inpainting for BEV purposes

Want to inpaint depth and semantic labels  
Should be easier than inpainting pixels



(a)



(b)

Fig. 2: (a) The *inpainting CNN* first encodes a masked image and the mask itself. The extracted features are concatenated and two decoders predict semantics and depth for visible and occluded pixels. (b) To train the inpainting CNN we ignore foreground objects as no ground truth is available (red) but we *artificially add masks (green)* over background regions where full annotation is already available.



# Registration

D.A. Forsyth

# Mostly, we've done this

- Useful
  - if we know where vehicle is (roughly)
  - we know a lot about likely layout in front of vehicle (openmaps, etc.)

# Adversarial Losses

D.A. Forsyth, UIUC

# Layout is stylized

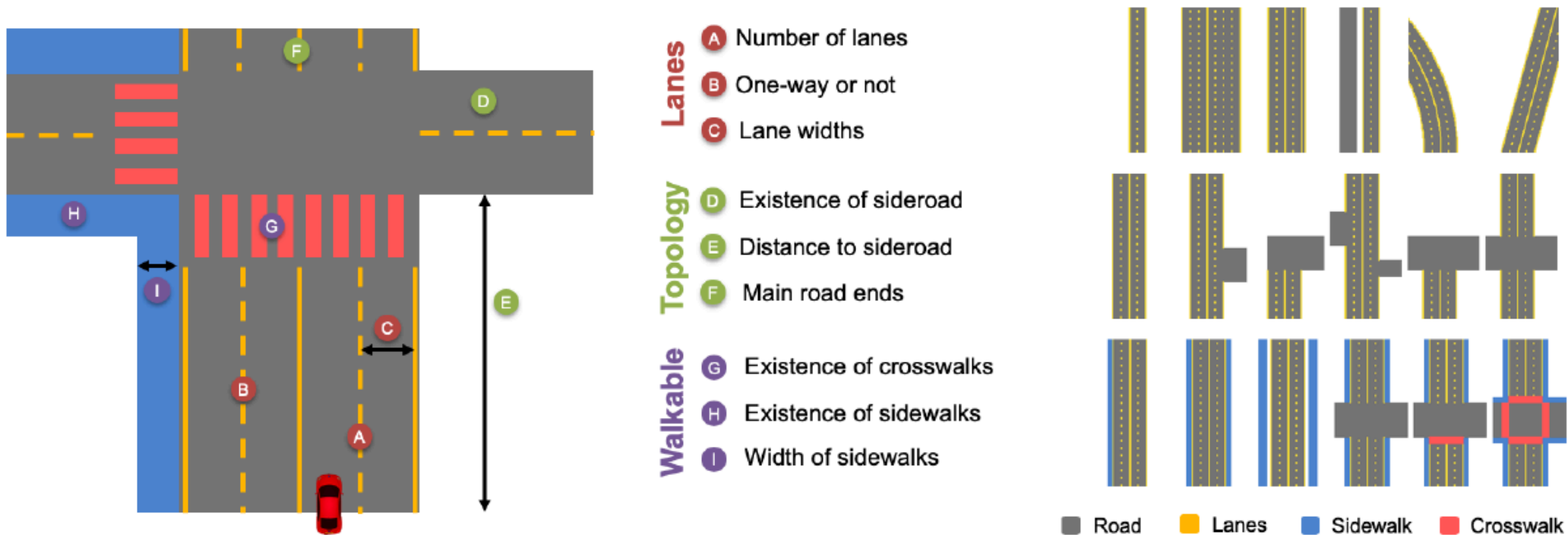


Figure 2: Our scene model consists of several parameters that capture a variety of complex driving scenes. (Left) We illustrate the model and highlight important parameters (A-I), which are grouped into three categories (middle): *Lanes*, to describe the layout of a single road; *Topology*, to model various road topologies; *Walkable*, describing scene elements for pedestrians. Our model is defined as a directed acyclic graph enabling efficient sampling and is represented in the top-view, making rendering easy. These properties turn our model into a simulator of semantic top-views. (Right) We show rendered examples for each of the above groups. A complete list of scene parameters and the corresponding graphical model is given in the supplementary.

# Q: How do we impose structure?

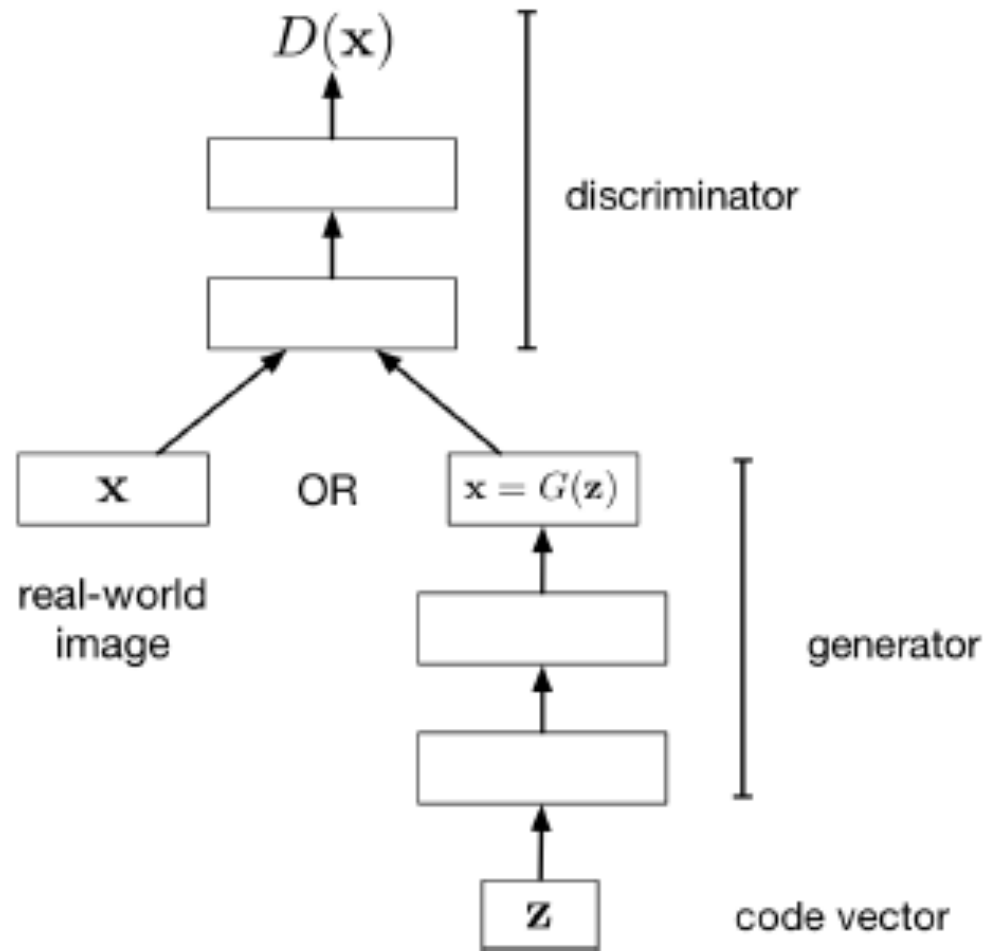
- We want to the network to produce layout maps that are “like real maps”
  - How?

# Side topic - Adversarial losses

- Issue:
  - we are making pictures that should have a strong structure
    - albedo piecewise constant, etc.
    - but we don't know how to write a loss that imposes that structure
- Strategy:
  - build a classifier that tries to tell the difference between
    - true examples
    - examples we made
  - use that classifier as a loss

# A GAN

Generative  
Adversarial  
Network



- Let  $D$  denote the discriminator's predicted probability of being data
- Discriminator's cost function: cross-entropy loss for task of classifying real vs. fake images

$$\mathcal{J}_D = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[-\log(1 - D(G(\mathbf{z})))]$$

Notice: we want the discriminator to make a 1 for real data, 0 for fake data

- One possible cost function for the generator: the opposite of the discriminator's

$$\begin{aligned} \mathcal{J}_G &= -\mathcal{J}_D \\ &= \text{const} + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))] \end{aligned}$$

- This is called the **minimax formulation**, since the generator and discriminator are playing a **zero-sum game** against each other:

$$\max_G \min_D \mathcal{J}_D$$

Solution (if exists, which is uncertain; and if can be found, ditto) is known as a saddle point.

It has strong properties, but not much worth talking about, as we don't know if it is there or whether we have found it.



Quote from the original paper on GANs:

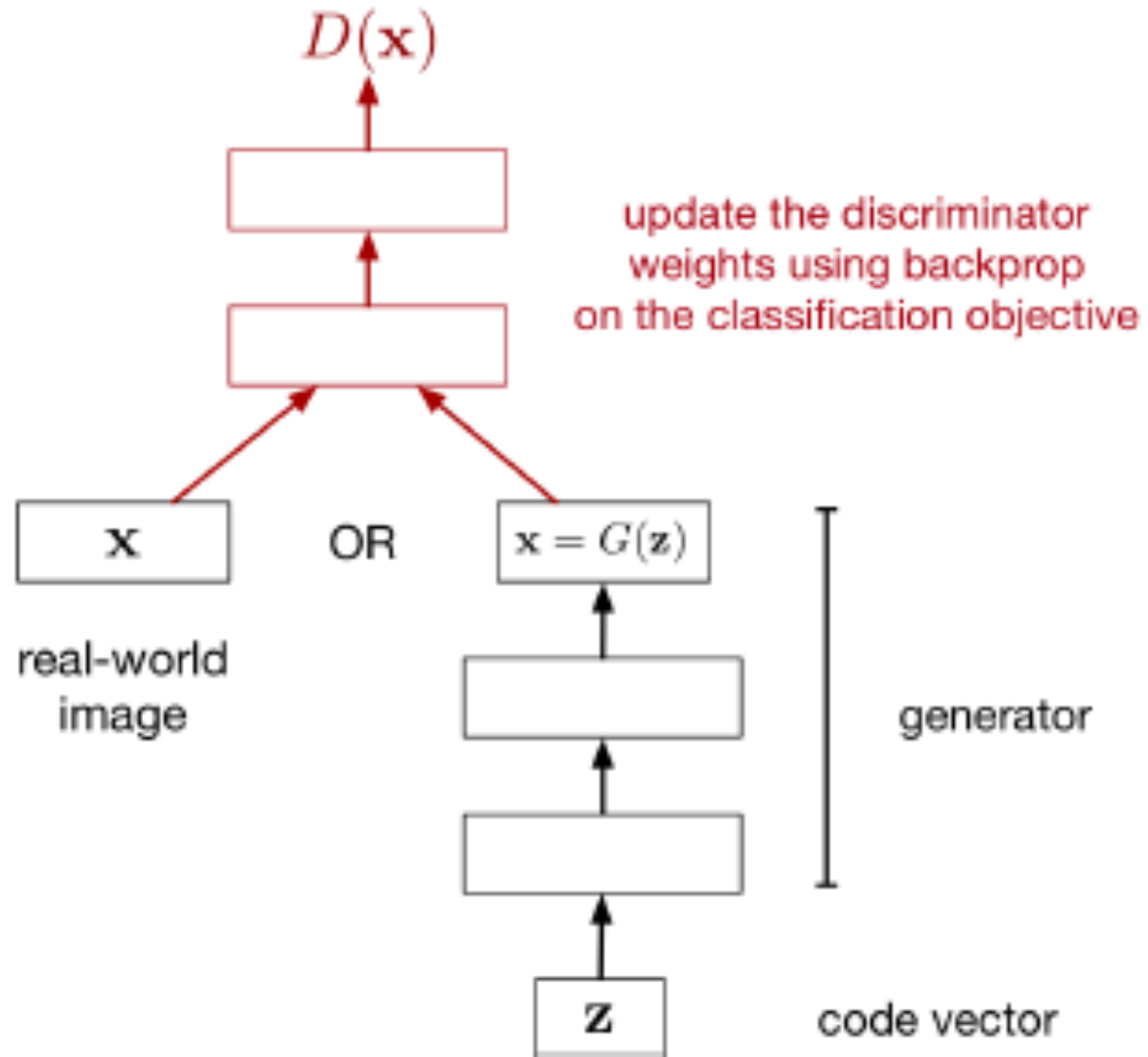
*"The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles."*

-Goodfellow et. al., "Generative Adversarial Networks" (2014)

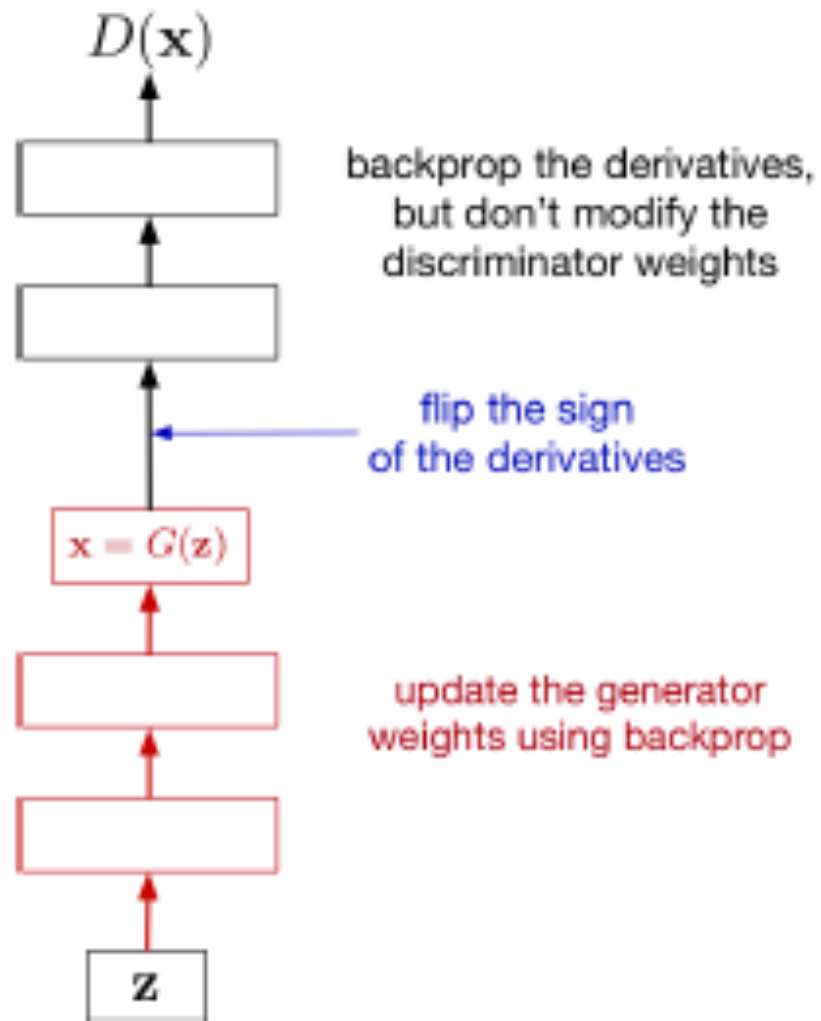
# Important, general issue

- If either generator or discriminator “wins” -> problem
- Discriminator “wins”
  - it may not be able to tell the generator how to fix examples
  - discriminators classify, rather than supply gradient
- Generator “wins”
  - likely the discriminator is too stupid to be useful
- Very little theory to guide on this point

## Updating the discriminator:



# Updating the generator:



# One must be careful about losses...

- We introduced the minimax cost function for the generator:

$$\mathcal{J}_G = \mathbb{E}_z[\log(1 - D(G(z)))]$$

- One problem with this is **saturation**.
- Recall from our lecture on classification: when the prediction is really wrong,
  - “Logistic + squared error” gets a weak gradient signal
  - “Logistic + cross-entropy” gets a strong gradient signal
- Here, if the generated sample is really bad, the discriminator’s prediction is close to 0, and the generator’s cost is flat.

# One must be careful about losses...

- Original minimax cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

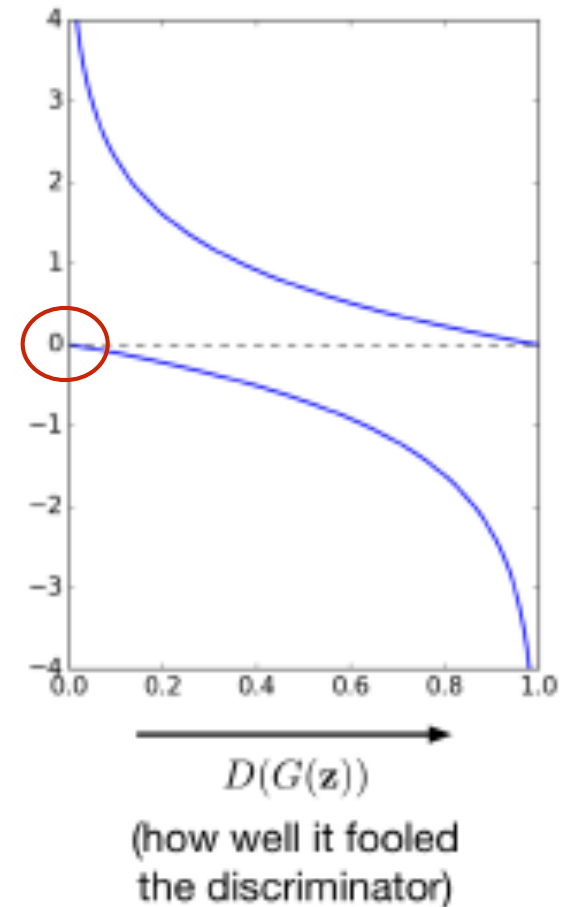
- Modified generator cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[-\log D(G(\mathbf{z}))]$$

- This fixes the saturation problem.

modified  
cost

minimax  
cost



# Alternative losses

- Hinge:

- Discriminator makes  $D(\text{im})$

- want

- real images  $\rightarrow -1$
- fake  $\rightarrow 1$

- Discriminator loss:

$$\sum_{\text{fakes and real}} \max(0, 1 - y_i D(I_i))$$

- where  $y_i = -1$  for real,  $y_i = 1$  for fake

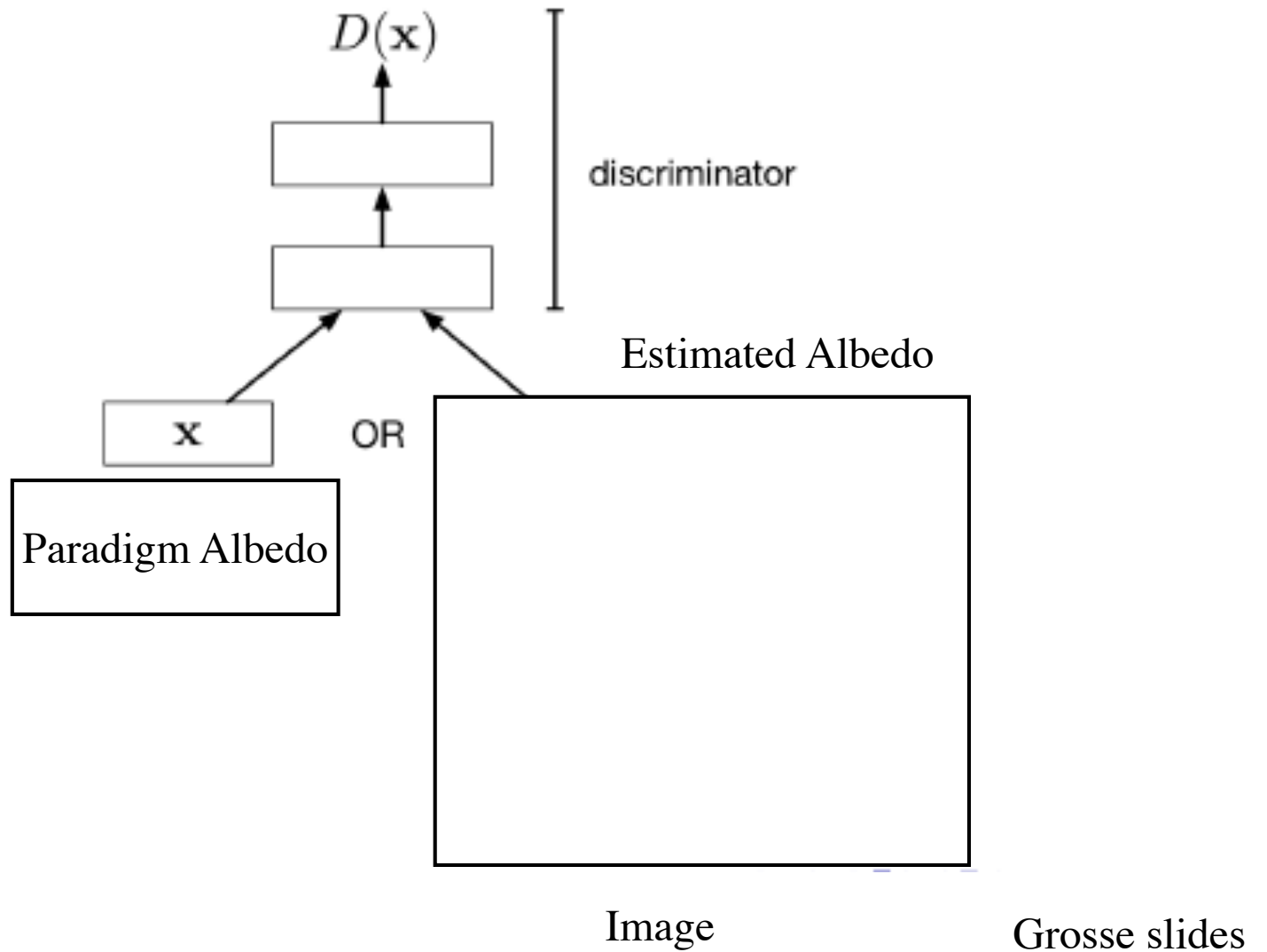
- Generator loss:

- 

$$\sum_{\text{fakes}} D(I_i)$$

# Adversarial loss

Adversarial loss





# Theory

**Proposition 2.** *If  $G$  and  $D$  have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given  $G$ , and  $p_g$  is updated so as to improve the criterion*

$$\mathbb{E}_{\mathbf{x} \sim p_{data}}[\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g}[\log(1 - D_G^*(\mathbf{x}))]$$

*then  $p_g$  converges to  $p_{data}$*

# “Theory”

**Proposition 2.** *If  $G$  and  $D$  have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given  $G$ , and  $p_g$  is updated so as to improve the criterion*

$$\mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_g} [\log(1 - D_G^*(\mathbf{x}))]$$

*then  $p_g$  converges to  $p_{data}$*

- What if they don't have enough capacity?
- What if  $p_g$  doesn't make “enough progress”?
- In what sense converges?
  - $p_{data}$  is a set of samples
  - we DON'T WANT usual convergences
  - we WANT convergence to some smoothed  $p_{data}$ 
    - how smoothed? how controlled?

# Questions

- How do we hobble an adversary in a useful way?
  - dunno
- When is an adversarial smoother helpful?
  - dunno
-

# Layout is stylized

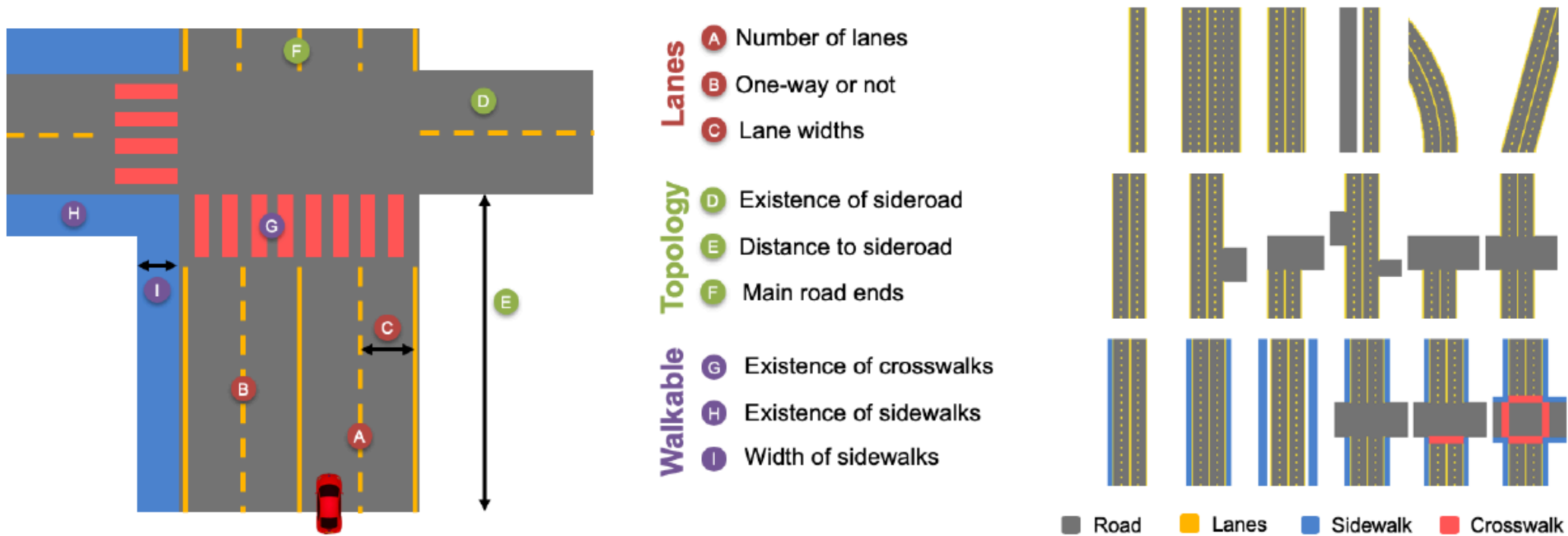


Figure 2: Our scene model consists of several parameters that capture a variety of complex driving scenes. (Left) We illustrate the model and highlight important parameters (A-I), which are grouped into three categories (middle): *Lanes*, to describe the layout of a single road; *Topology*, to model various road topologies; *Walkable*, describing scene elements for pedestrians. Our model is defined as a directed acyclic graph enabling efficient sampling and is represented in the top-view, making rendering easy. These properties turn our model into a simulator of semantic top-views. (Right) We show rendered examples for each of the above groups. A complete list of scene parameters and the corresponding graphical model is given in the supplementary.