Exploiting Bias for Scene Recovery

D.A. Forsyth, UIUC

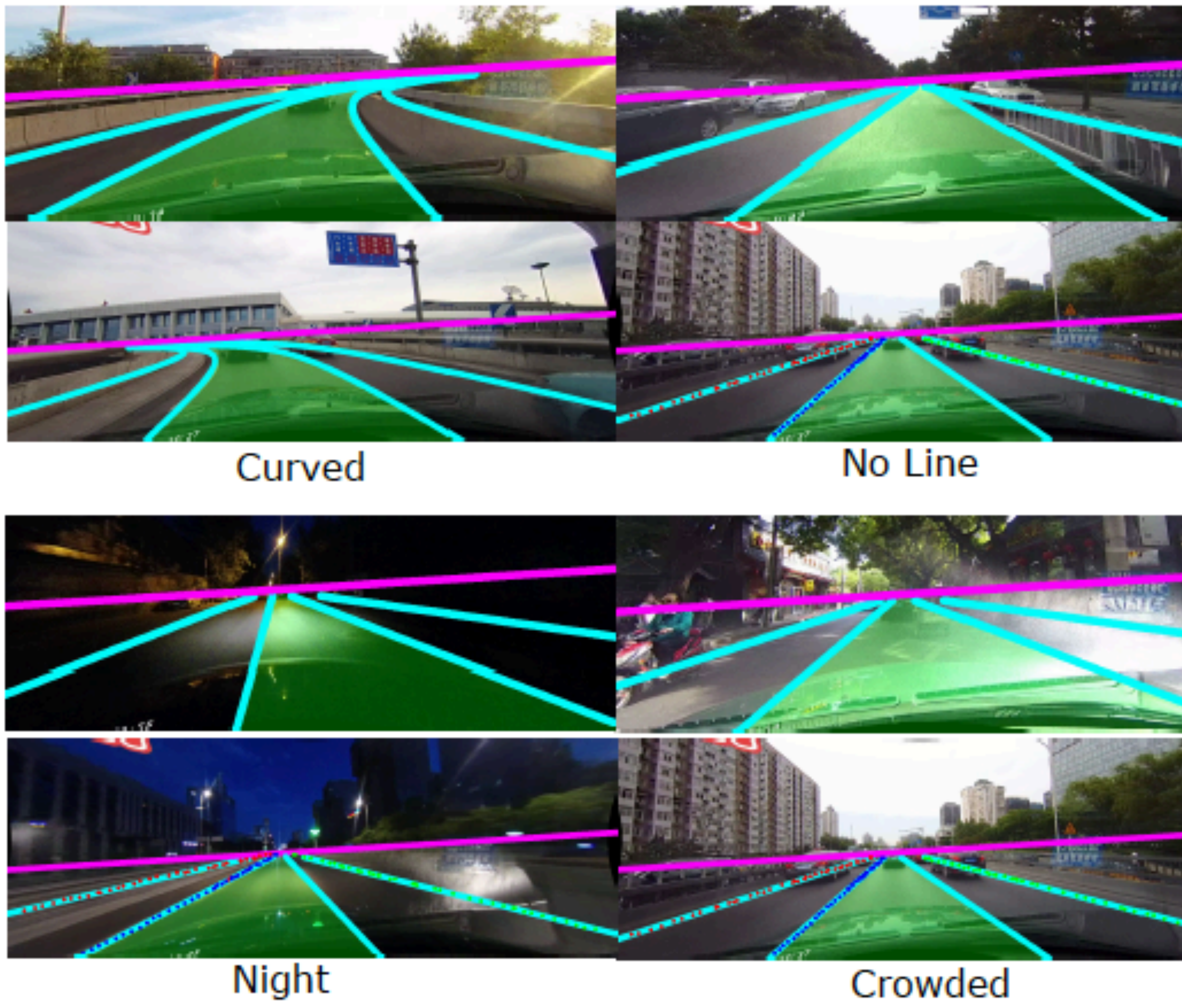


Figure 1. Sample results of our algorithm on examples from fo
different classes of CULane dataset [33] are shown here. Cy
lines are the detected lane boundaries, green region represents t
ego lane and magenta line displays the estimated horizon. In t
No Line class, there is actually no line markings on the road b
the ground truth carries the lines shown.

Goal: Road Layout Map

- With minimal/no labelling
- In nasty geometries

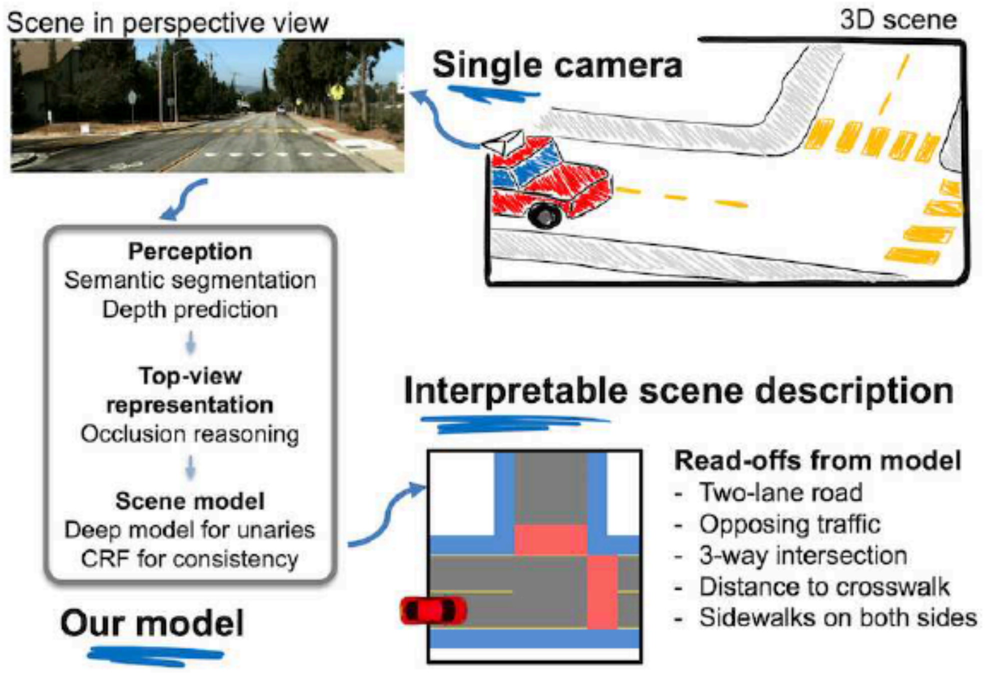


Figure 1: Our goal is to infer the layout of complex driving scenes from a single camera. Given a perspective image (top left) that captures a 3D scene, we predict a rich and interpretable scene description (bottom right), which represents the scene in an occlusion-reasoned semantic top-view.

Wang et al 19

Road layout maps

- A prediction of the layout of the main scene in front
  - distinguish between
    - transients (cars, pedestrians, etc)
    - and persistent (road, walkways, bicycle lanes, buildings)
  - including
    - intersections
    - lane boundaries
- Potential cues
  - streetview
  - openmaps
  - layout is stylized
  - persistent categories have coherent (but variable) appearance
  - scene flow/photometric consistency

Road layout map

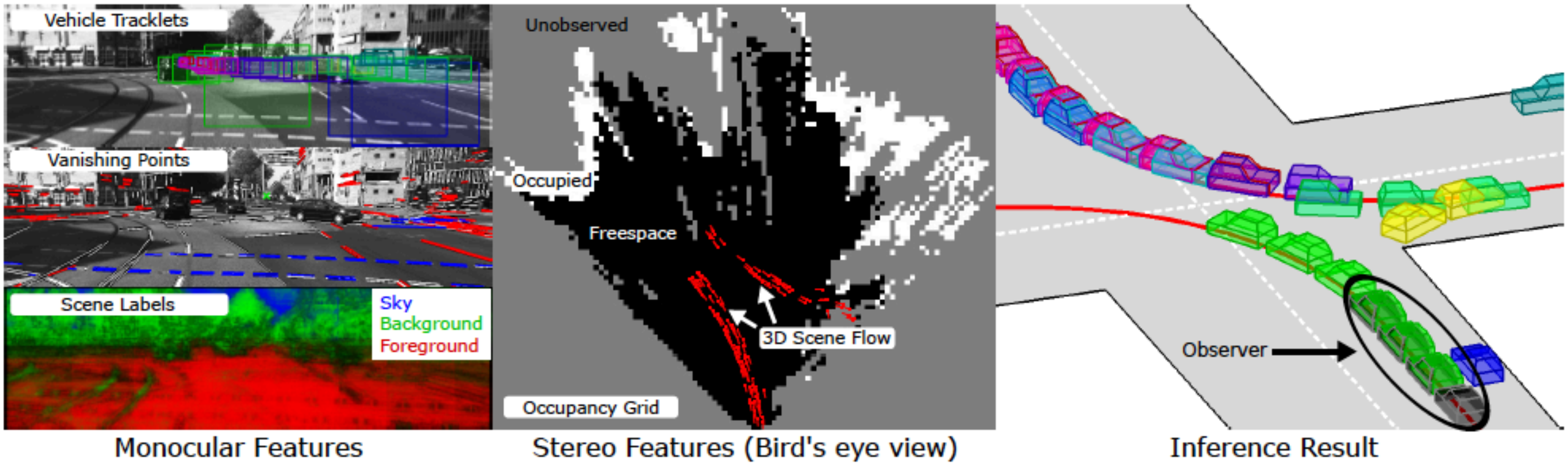


Fig. 1: **3D Intersection Understanding.** Our system makes use of monocular (left) and stereo (middle) feature cues to infer the road layout and the location of traffic participants in the scene (right) from short video sequences. The observer is depicted in black.

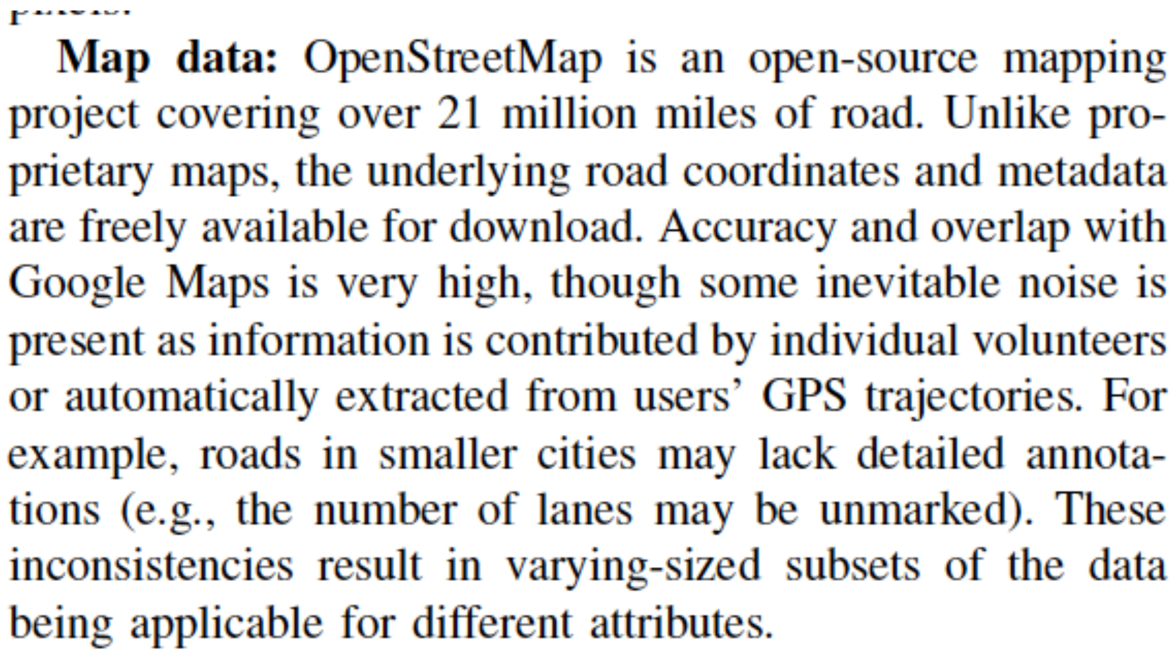
Geiger et al

Cues

- Incidental data
  - streetview+openmaps
- layout is stylized
- persistent categories have coherent appearance
- scene flow/photometric consistency

Partially supervised cues

- Open Street Maps (OSM)



Map data: OpenStreetMap is an open-source mapping project covering over 21 million miles of road. Unlike proprietary maps, the underlying road coordinates and metadata are freely available for download. Accuracy and overlap with Google Maps is very high, though some inevitable noise is present as information is contributed by individual volunteers or automatically extracted from users' GPS trajectories. For example, roads in smaller cities may lack detailed annotations (e.g., the number of lanes may be unmarked). These inconsistencies result in varying-sized subsets of the data being applicable for different attributes.

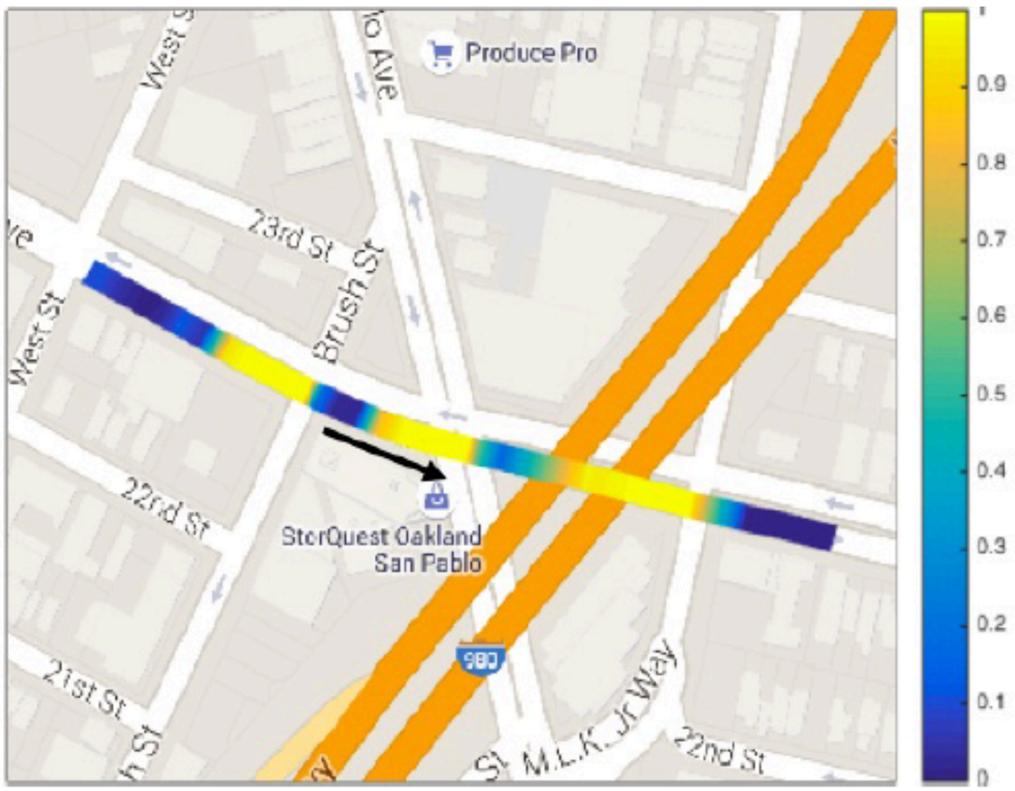


Fig. 3. Intersection detection heatmap. Images are cropped from test set GSV panoramas in the direction of travel indicated by the black arrow. The probabilities of "approaching" an intersection output by the trained ConvNet are overlaid on the road. (The images are from the ground level road, not the bridge.)

Partially supervised cues

- Google street view

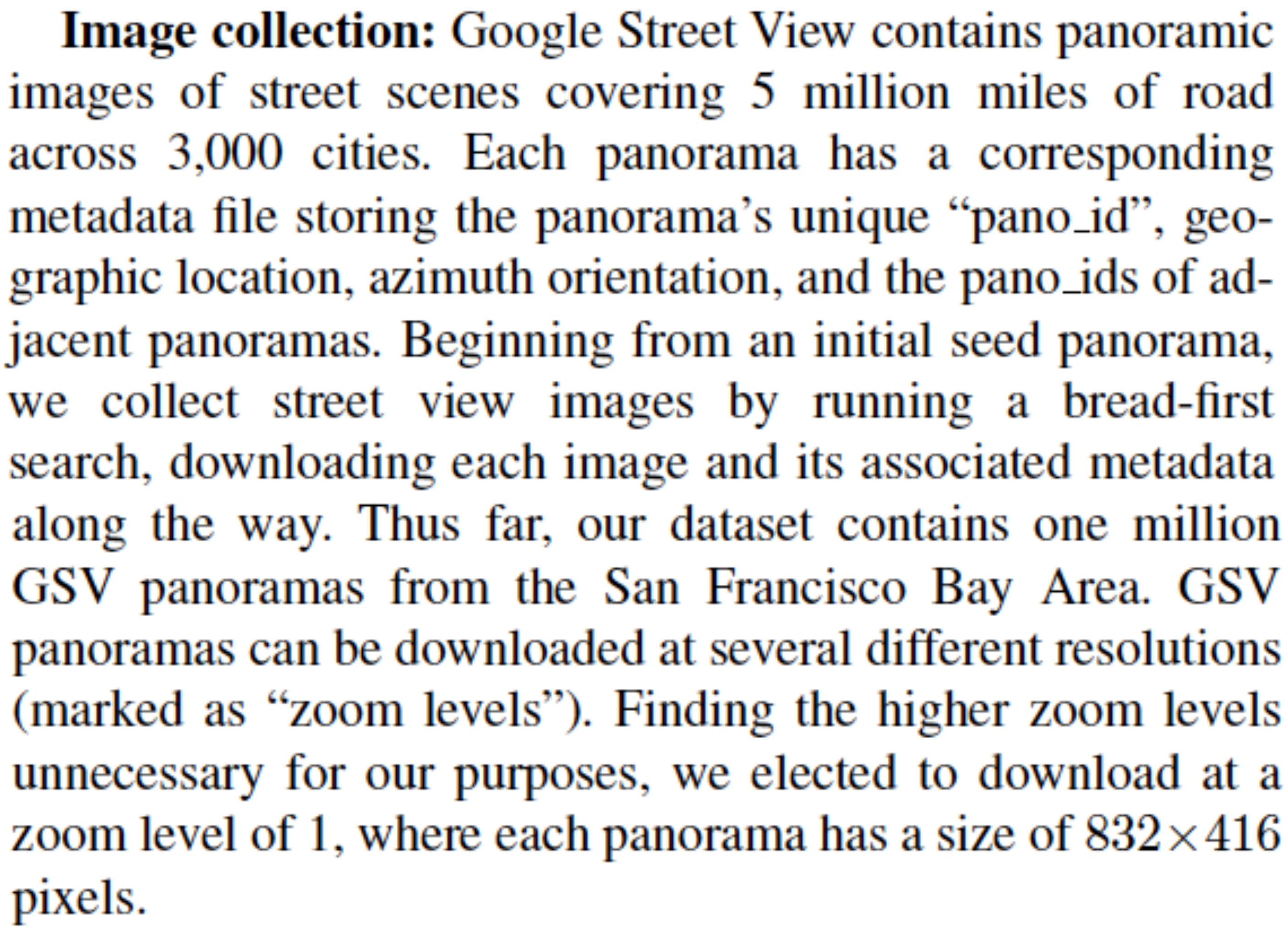


Image collection: Google Street View contains panoramic images of street scenes covering 5 million miles of road across 3,000 cities. Each panorama has a corresponding metadata file storing the panorama's unique "pano_id", geographic location, azimuth orientation, and the pano_ids of adjacent panoramas. Beginning from an initial seed panorama, we collect street view images by running a bread-first search, downloading each image and its associated metadata along the way. Thus far, our dataset contains one million GSV panoramas from the San Francisco Bay Area. GSV panoramas can be downloaded at several different resolutions (marked as "zoom levels"). Finding the higher zoom levels unnecessary for our purposes, we elected to download at a zoom level of 1, where each panorama has a size of 832×416 pixels.

Labelling - I

- Match panoramas to roads
  - panorama center location, orientation is known
  - (essentially) project to plane
  - thresholded nearest neighbor to road center polyline
    - thresholding removes panoramas inside buildings, etc.
  - some noise
    - under bridges, etc.
- Annotations
  - Intersections
  - Drivable heading
  - Heading angle
  - Bike lane
  - Speed limit, wrong way, etc.

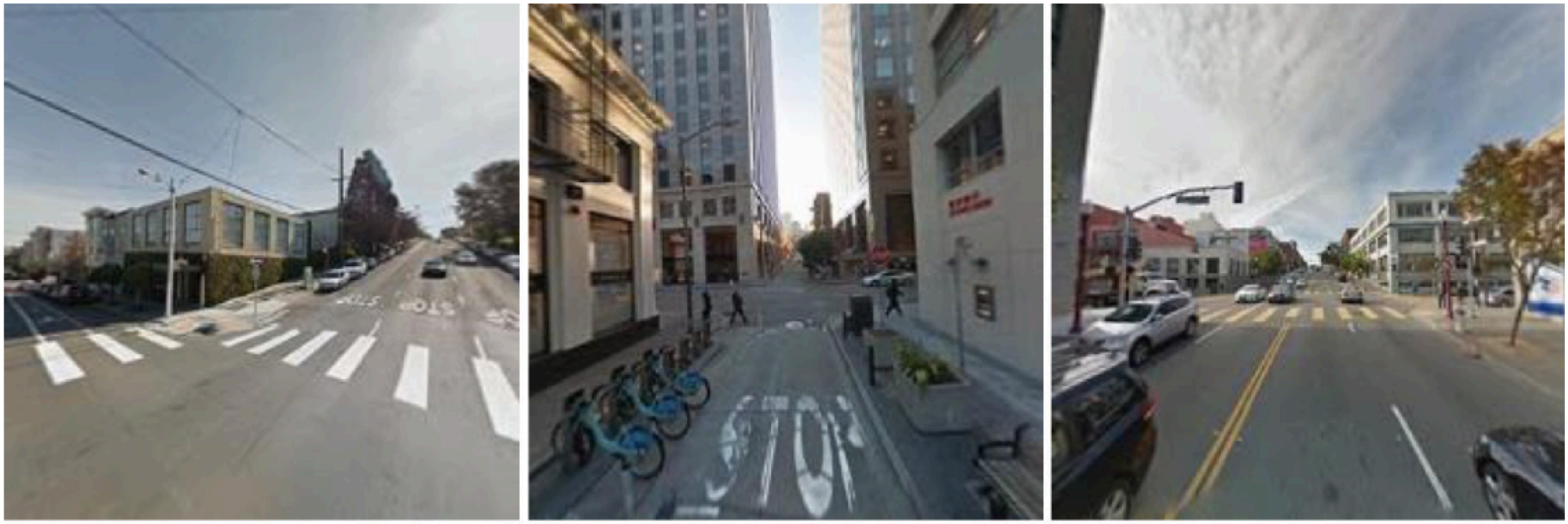


Fig. 4. Distance to intersection estimation. For images within 30 m of true intersections, our model is trained to estimate the distance from the host car to the center of the intersection across a variety of road types.

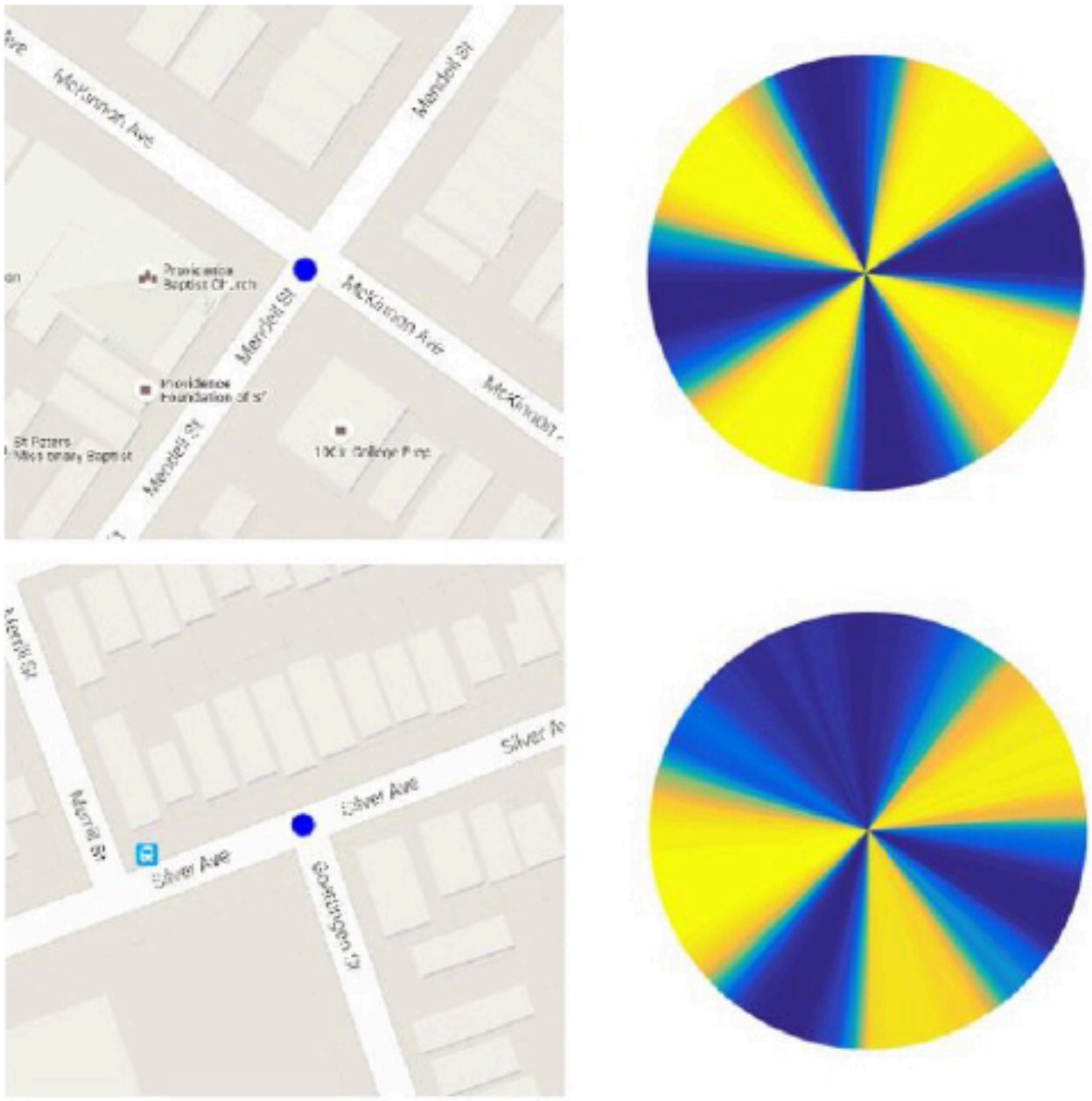


Fig. 5. Intersection topology is one of several attributes our model learns to infer from an input GSV panorama. The blue circles on the Google Maps extracts to the left show the locations of the input panoramas. The pie charts display the probabilities output by the trained ConvNet of each heading angle being on a driveable path (see Figure 3 for colormap legend).

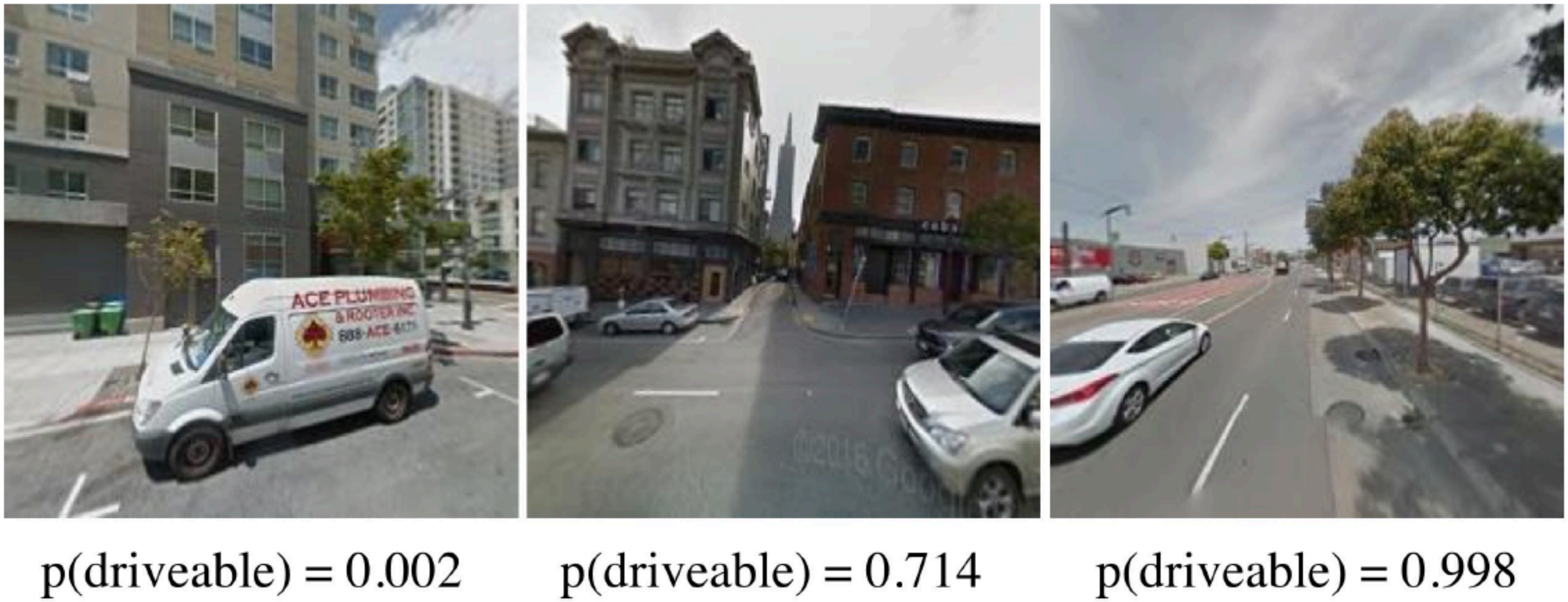


Fig. 6. Driveable headings. A ConvNet is trained to distinguish between non-drivable headings (left) and drivable headings aligned with the road (right). The ConvNet weakly classifies the middle example as drivable because the host car's heading is facing the alleyway between the buildings.

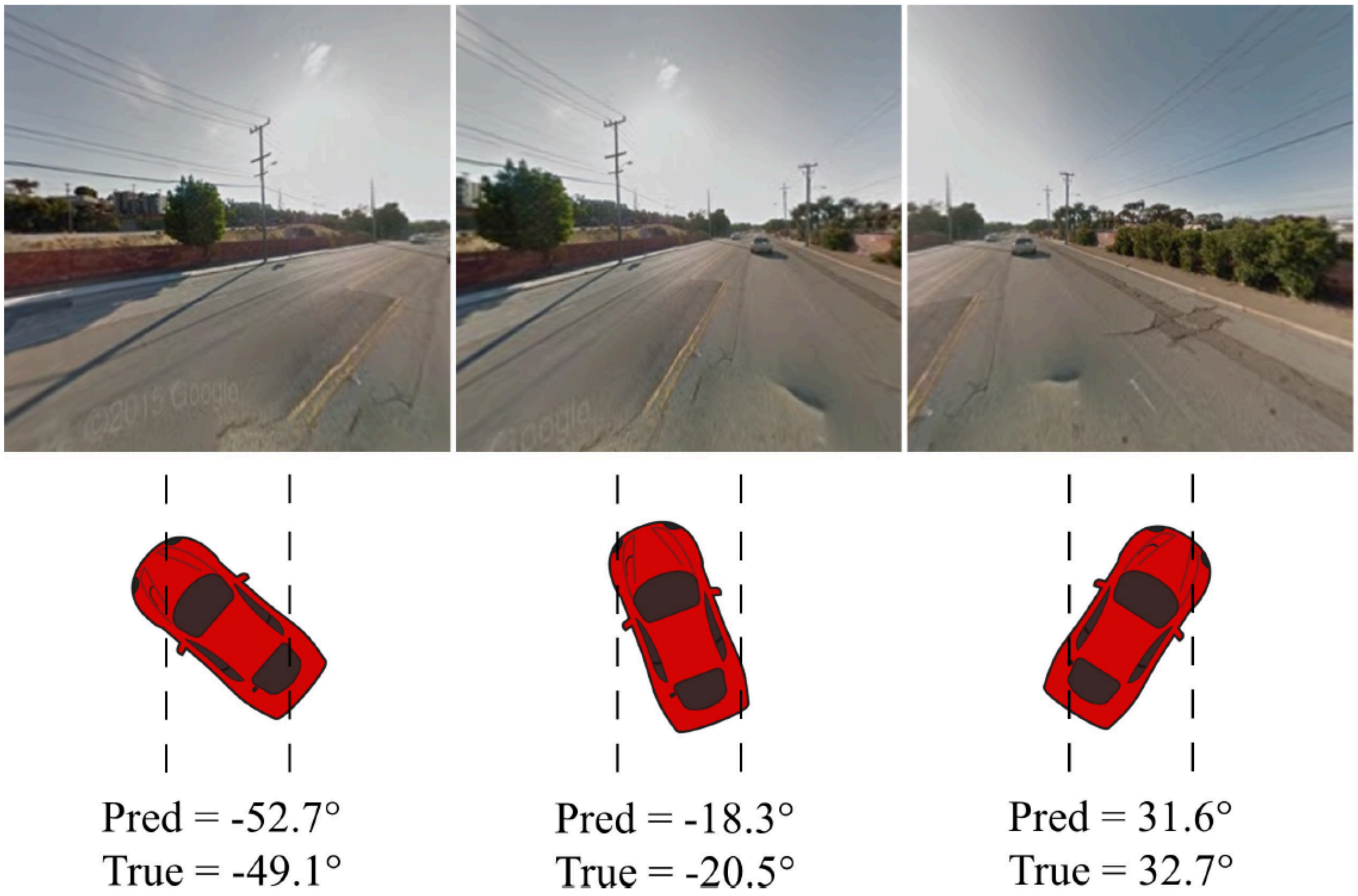


Fig. 7. Heading angle regression. The network learns to predict the relative angle between the street and host vehicle heading given a single image cropped from a GSV panorama. Below each GSV image, the graphic visualizes the ground truth heading angle.

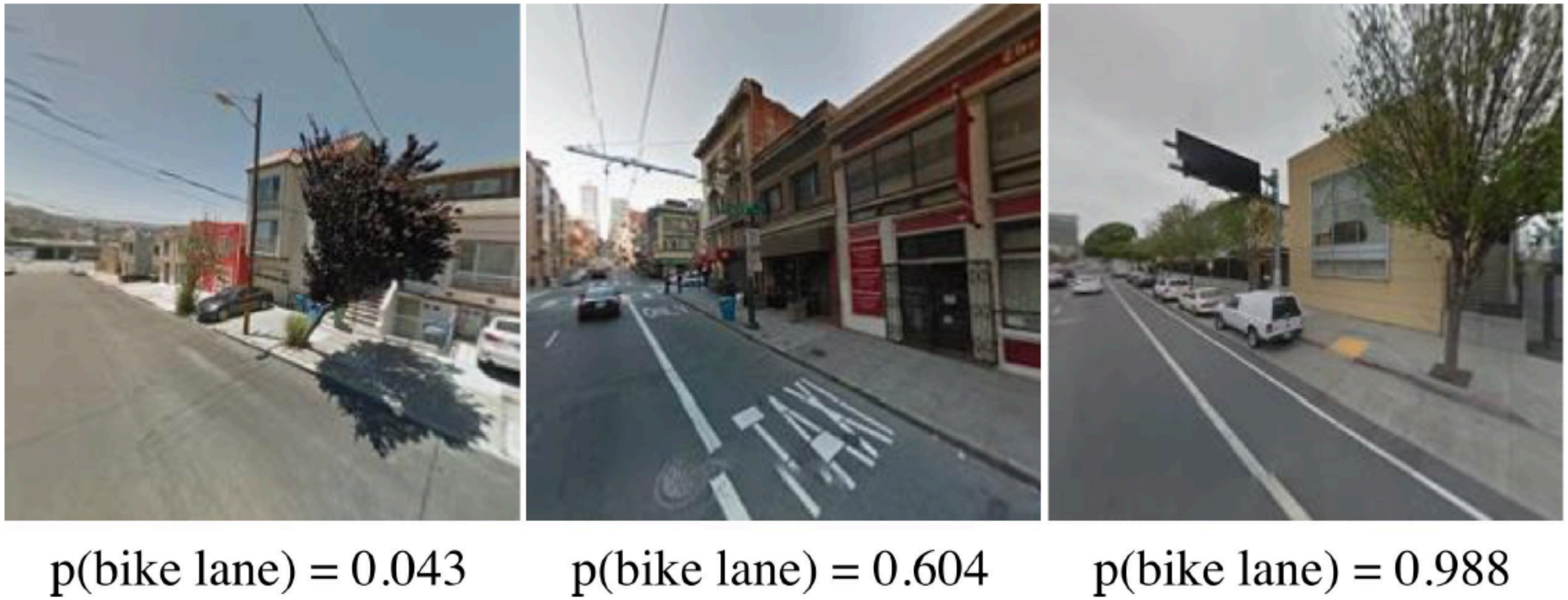


Fig. 8. The ConvNet learns to detect bike lanes adjacent to the vehicle. The GSV images are arranged from left to right in increasing order of probability output by the ConvNet of a bike lane being present (ground truth labels from left to right are negative, negative, positive). The middle example contains a taxi lane, resulting in a weak false positive.

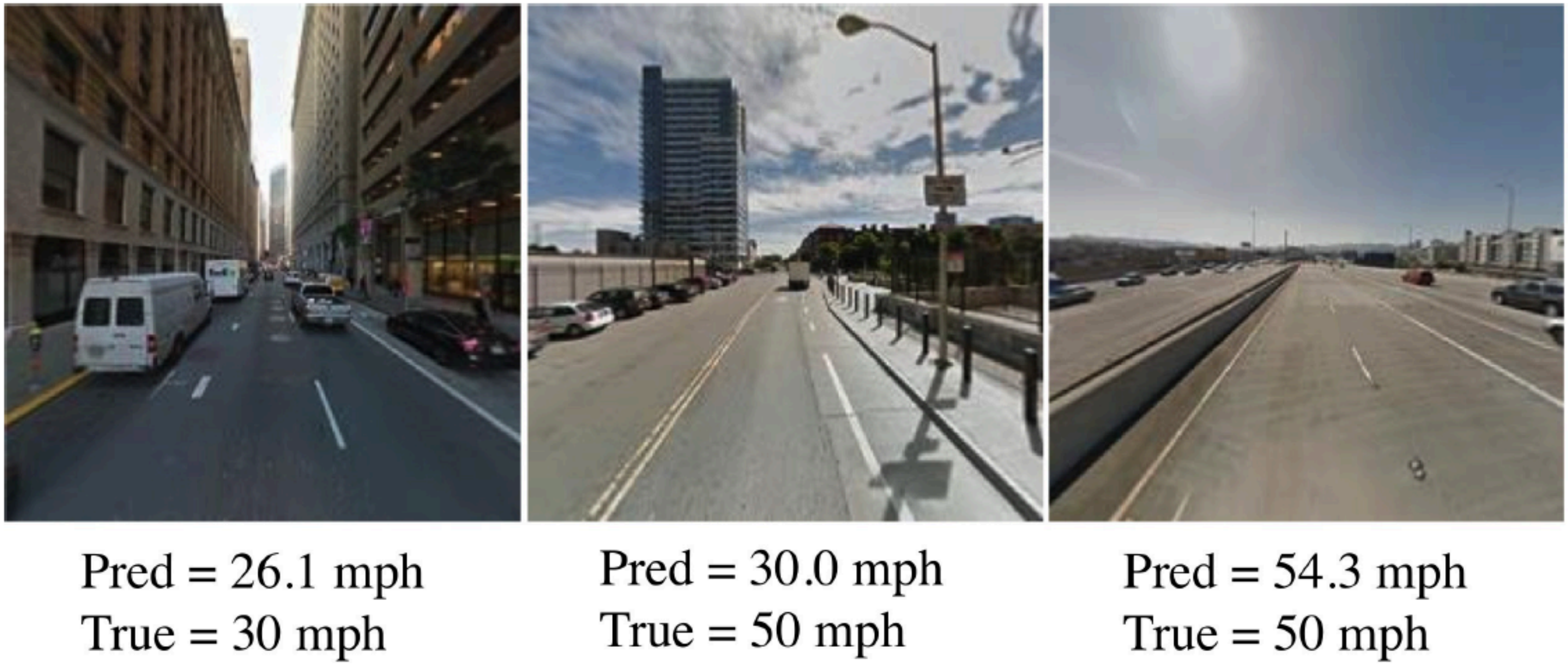


Fig. 9. Speed limit regression. The network learns to predict speed limits given a GSV image of road scene. The model significantly underestimates the speed limit in the middle example as this type of two-way road with a single lane in each direction would generally not have a speed limit as high as 50 mph.

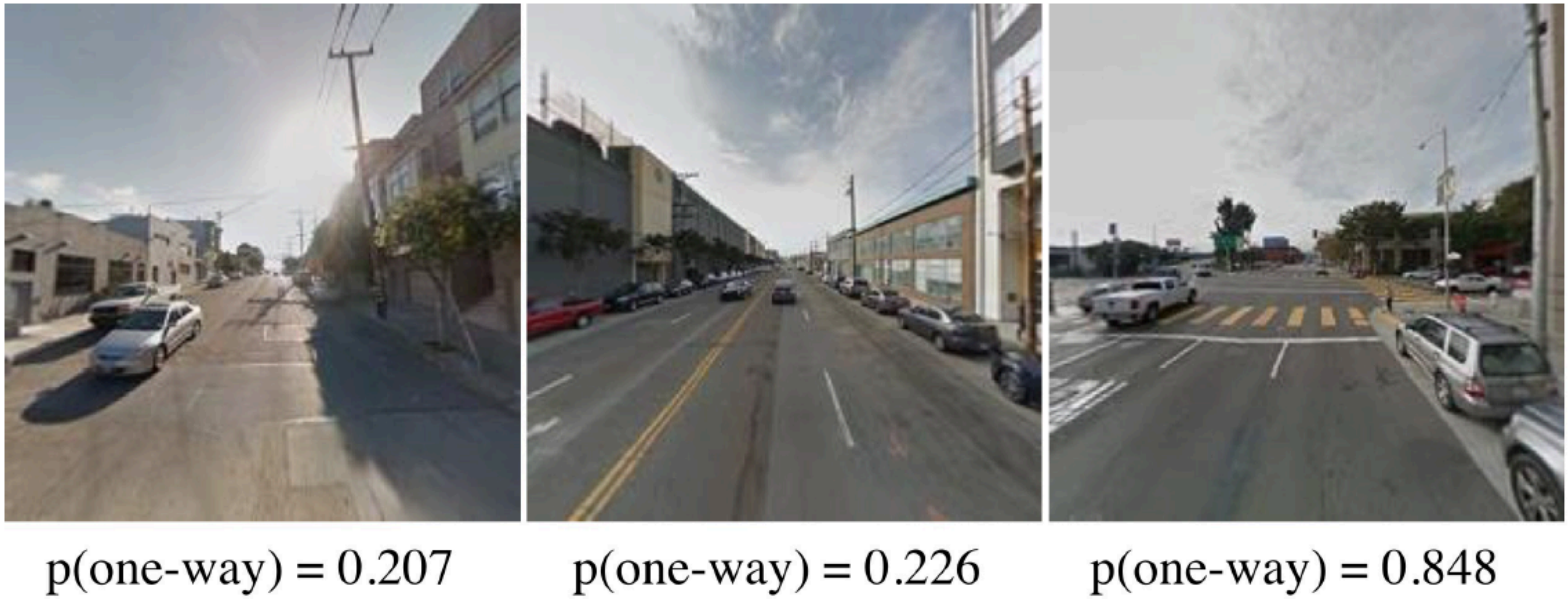
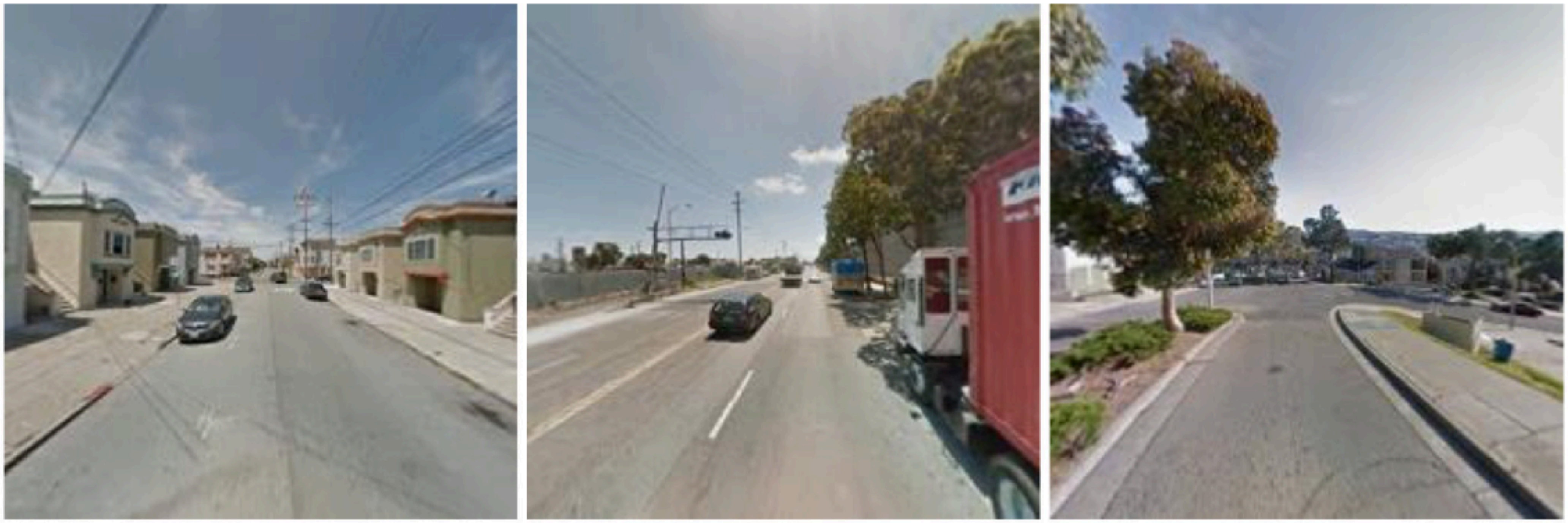


Fig. 10. One-way vs. two-way road classification. The probability output by the ConvNet of each GSV scene being on a one-way road is shown. From left to right the ground truth labels are two-way, two-way, and one-way. The image on the left is correctly classified as two-way despite the absence of the signature double yellow lines.



$p(\text{wrong way}) = 0.555$ $p(\text{wrong way}) = 0.042$ $p(\text{wrong way}) = 0.729$

Fig. 11. Wrong way detection. The probability output by the ConvNet of each GSV image facing the wrong way on the road is displayed. From left to right the ground truth labels are wrong way, right way, and right way. For two-way roads with no lane markings (left), this is an especially difficult problem as it amounts to estimating the horizontal position of the host car. The problem can also be quite ill-defined if there are no context clues as is the case with the rightmost image.

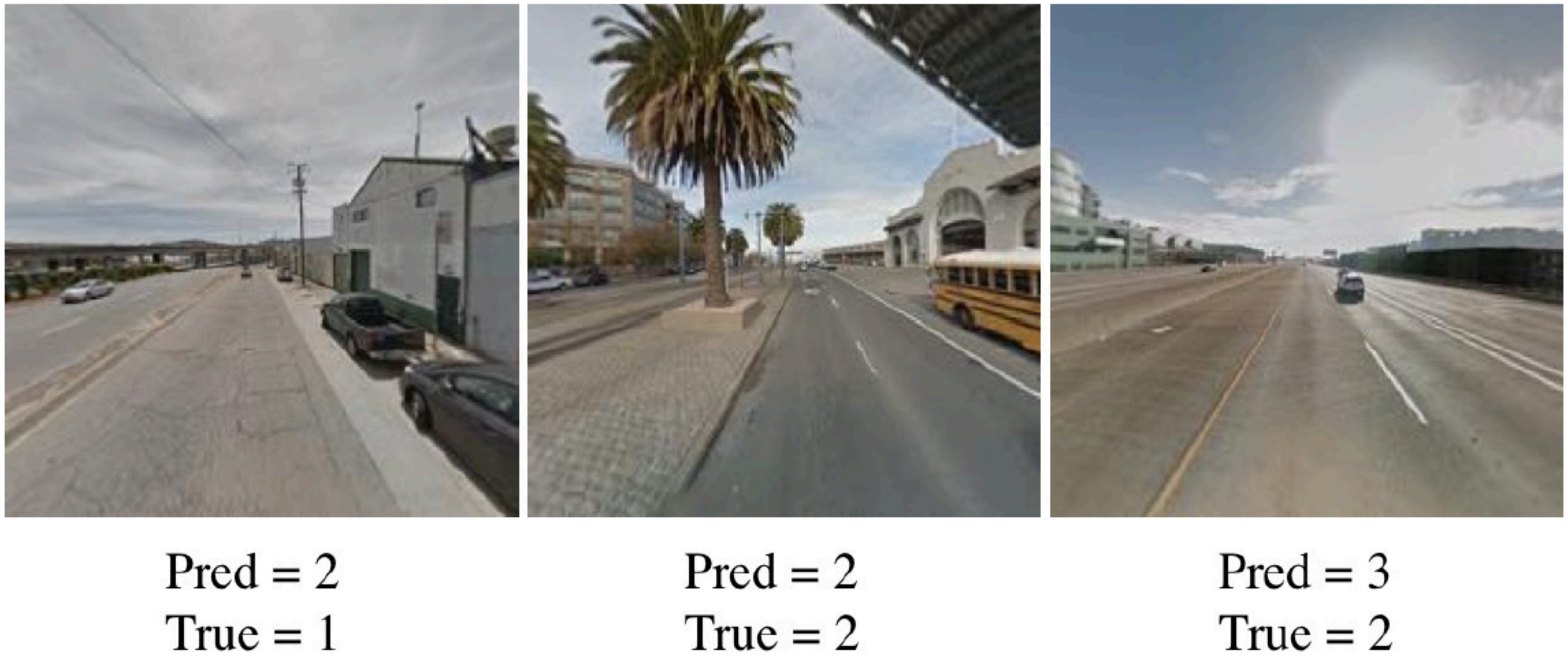


Fig. 12. Number of lanes estimation. The predicted and true number of lanes for three roads are displayed along with the corresponding GSV images. For streets without clearly visible lane markings (left), this is especially challenging. Although the ground truth for the rightmost image is two lanes, there is a third lane that merges just ahead.

Cues

- Incidental data
  - streetview+openmaps
- layout is stylized
- persistent categories have coherent appearance
- scene flow/photometric consistency

Layout is stylized

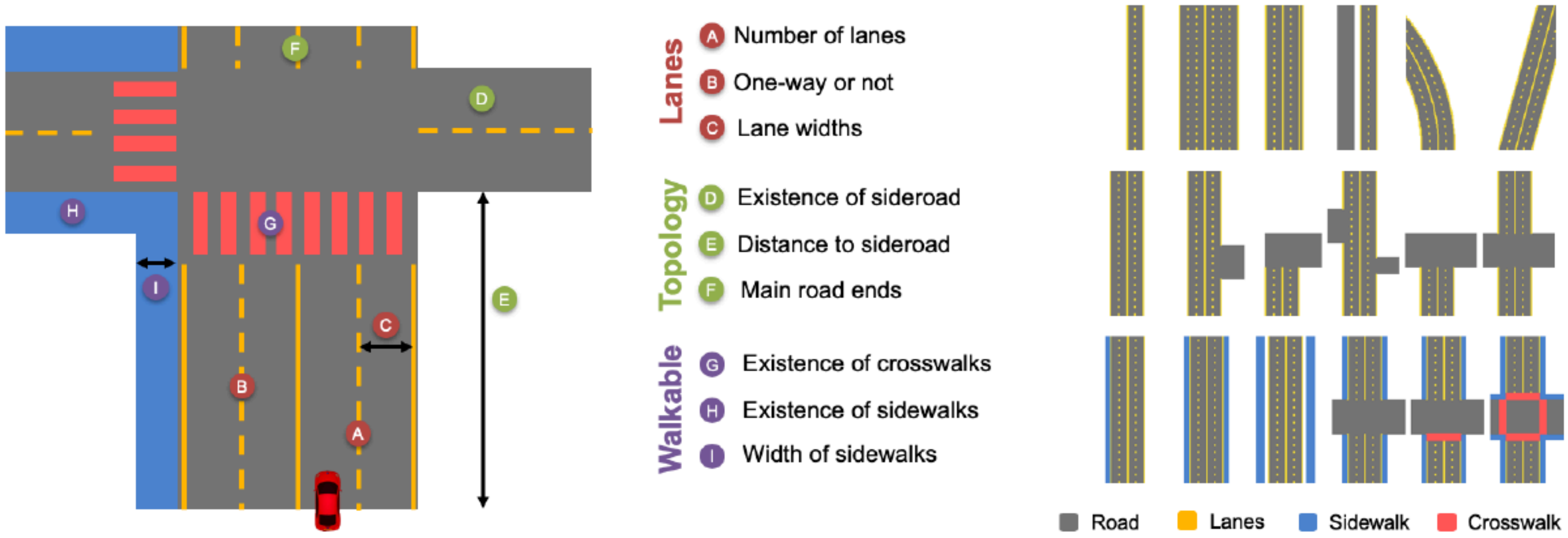


Figure 2: Our scene model consists of several parameters that capture a variety of complex driving scenes. (Left) We illustrate the model and highlight important parameters (A-I), which are grouped into three categories (middle): *Lanes*, to describe the layout of a single road; *Topology*, to model 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.

In overhead view

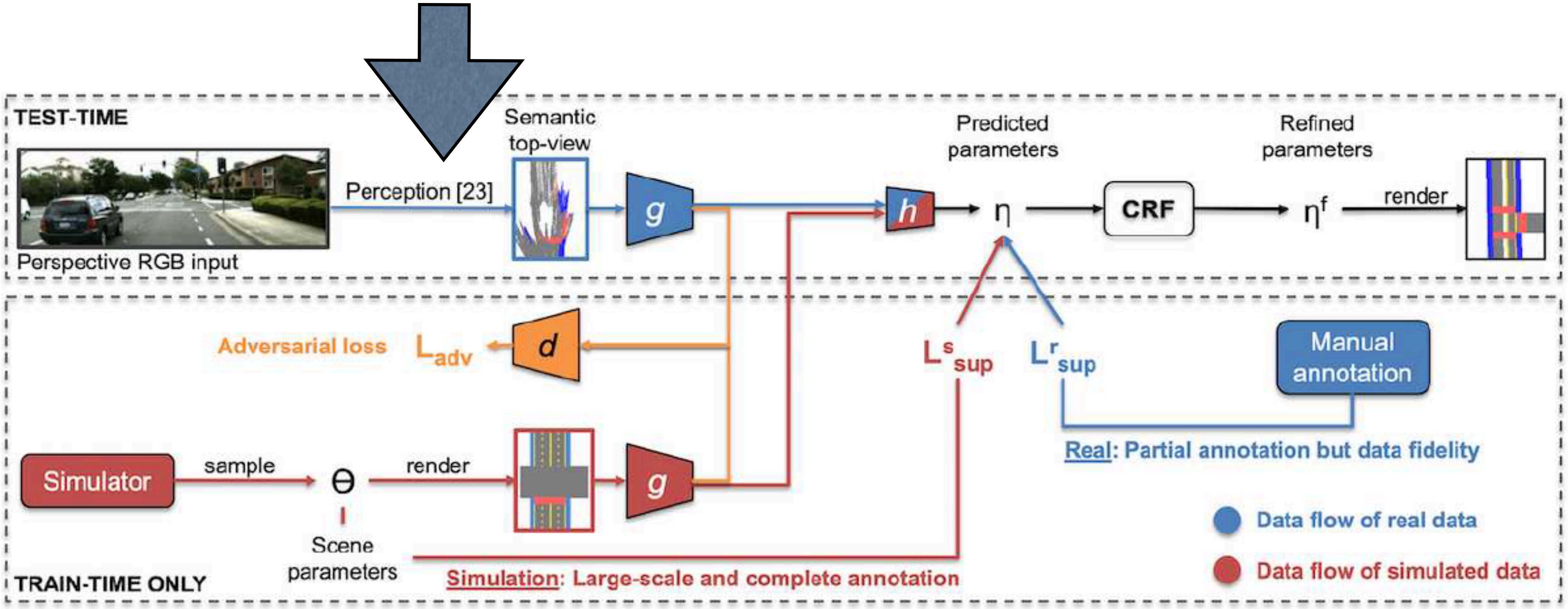
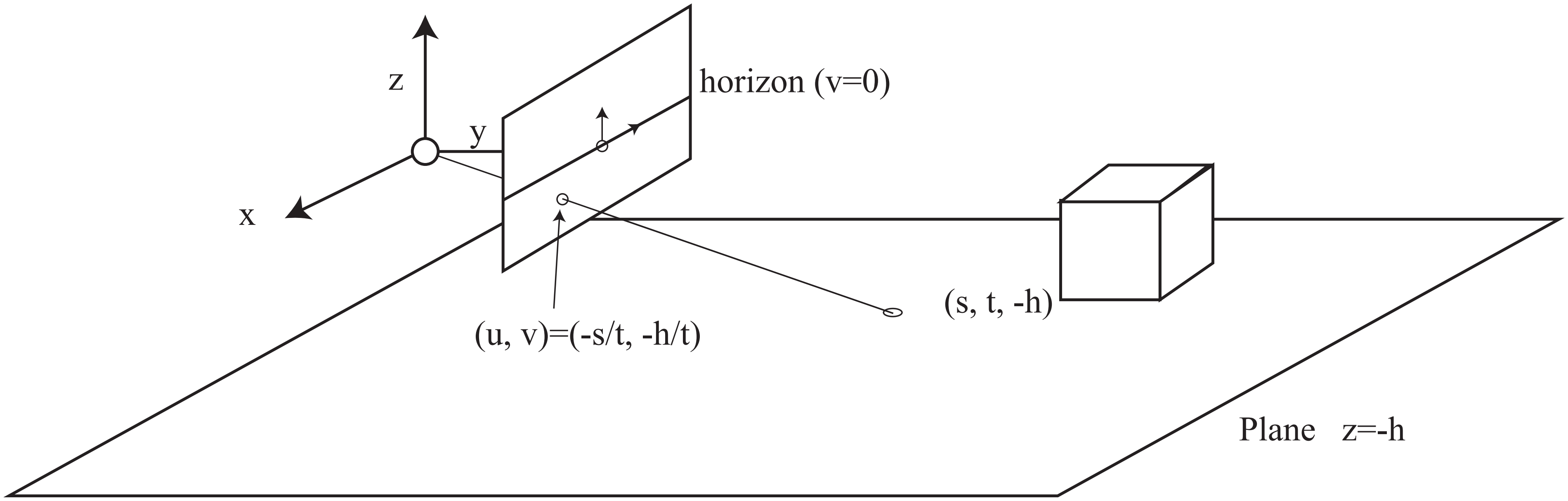


Figure 3: **Overview of our proposed framework:** At train-time, our framework makes use of both manual annotation for real data (blue) and automated annotation for simulated data (red), see Sec. 3.2. The feature extractors g convert semantic top views from either domain into a common representation which is input to h. An adversarial loss (orange) encourages a domain-agnostic output of g. At test-time, an RGB image in the perspective view is first transformed into a semantic top-view [23], which is then used by our proposed neural network (see Sec. 3.3), $h \circ g$, to infer our scene model (see Sec. 3.1). The graphical model defined in Sec. 3.4 ensures a coherent final output.

Birds eye view

- We want
  - overhead view of semantically labelled image
  - completed

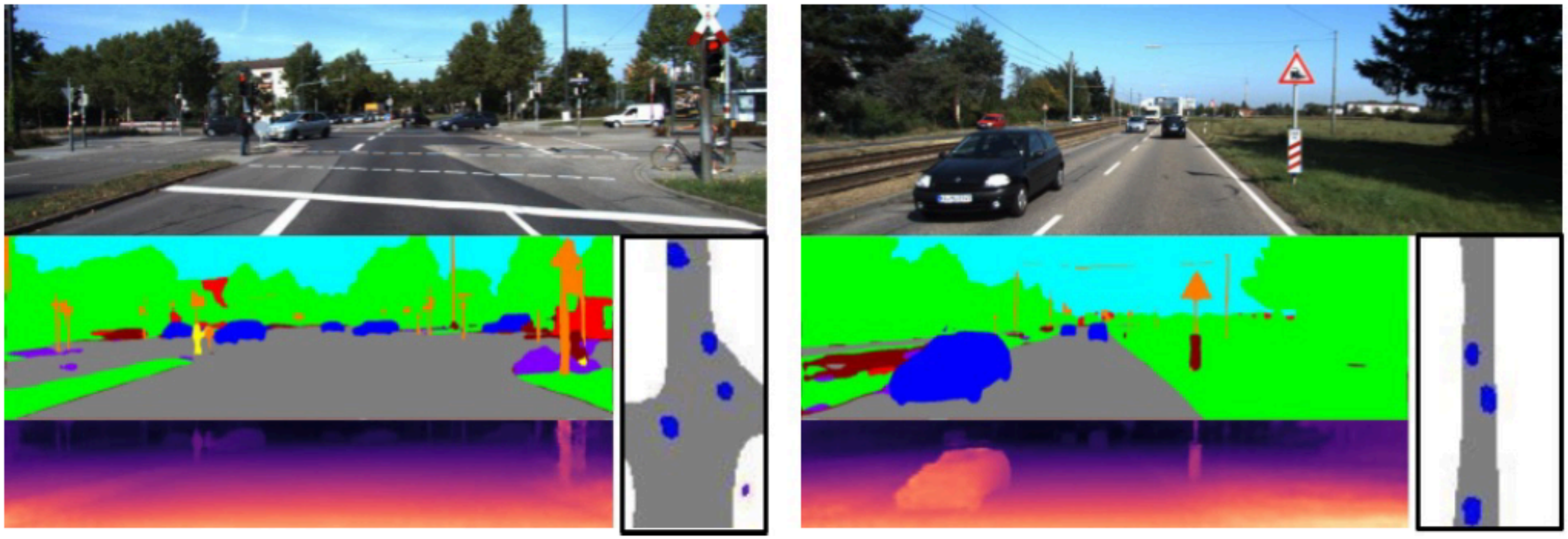


Schulter et al 18

Strategy

- Label image
- Knock out incidentals
  - cars, pedestrians, etc.
- Inpaint
- Project using depth

!?!?!?!



Birds eye view

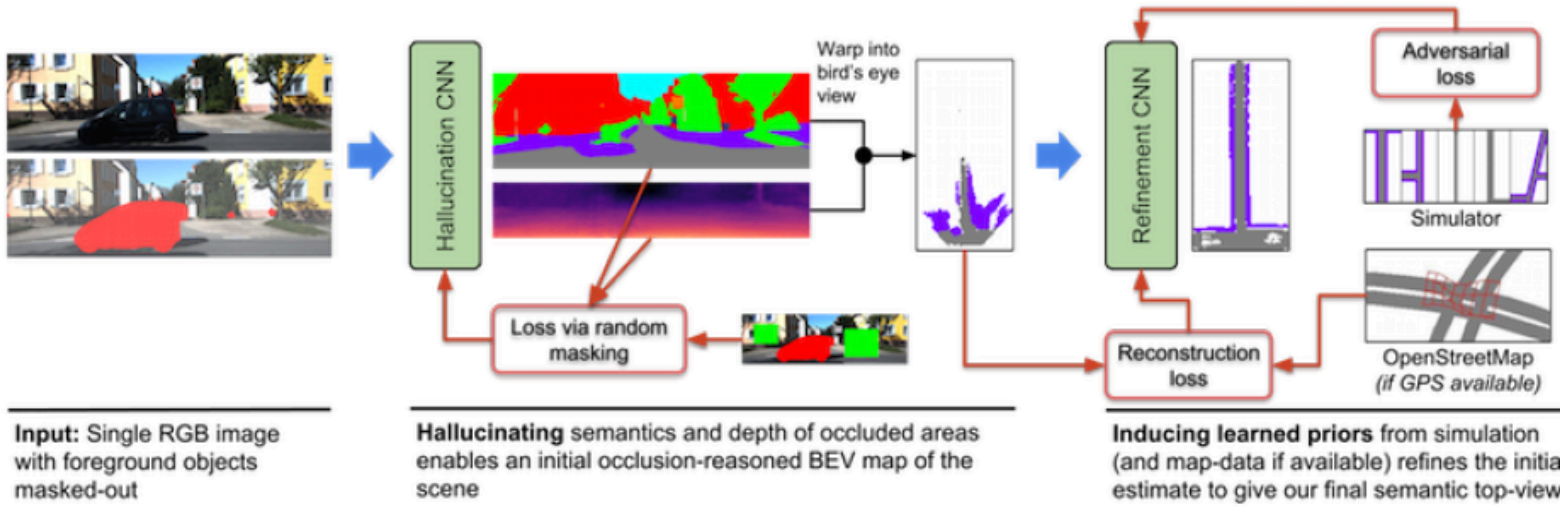
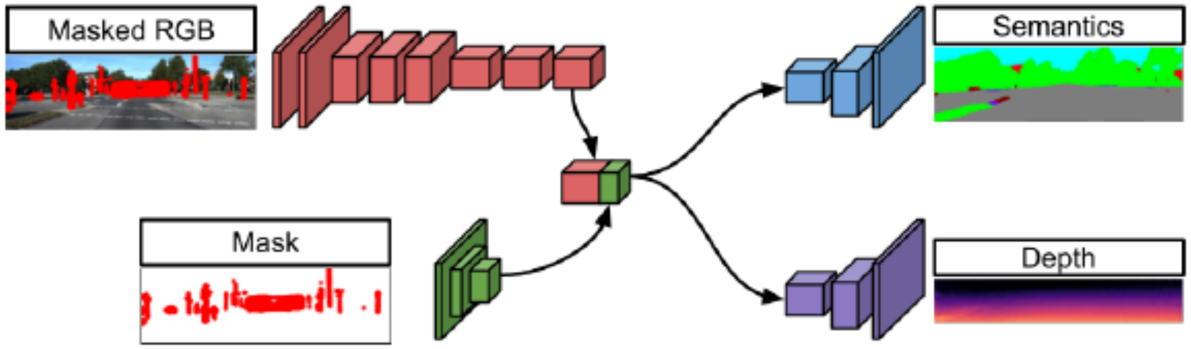

Hallucination CNN
Warp into bird's eye view
Refinement CNN
Adversarial loss
Simulator
Loss via random masking
Reconstruction loss
OpenStreetMap (if GPS available)
Input: Single RGB image with foreground objects masked-out
Hallucinating semantics and depth of occluded areas enables an initial occlusion-reasoned BEV map of the scene
Inducing learned priors from simulation (and map-data if available) refines the initia estimate to give our final semantic top-view

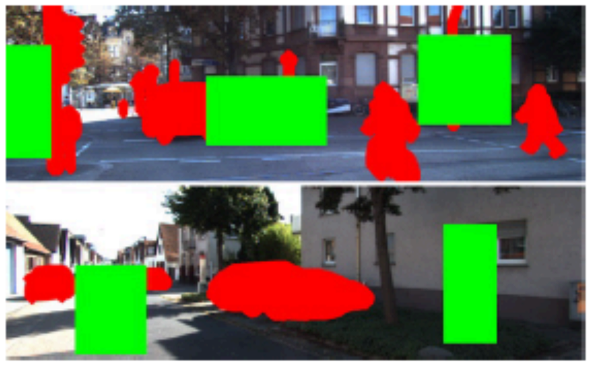
Fig. 1: Given a single RGB image of a typical street scene (left), our approac creates an **occlusion-reasoned semantic map of the scene layout in th bird's eye view**. We present a CNN that can hallucinate depth and semantics i areas occluded by foreground objects (marked in red and obtained via standar semantic segmentation), which gives an initial but noisy and incomplete estimat of the scene layout (middle). To fill in unobserved areas in the top-view, we furthe propose a refinement-CNN that induces learning strong priors from simulate and OpenStreetMap data (right), which comes at no additional annotation cost

Schulter et al 18

Inpainting



(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.

Notice: we inpaint labels and depth, NOT the image

Notice: depth is inferred from the image

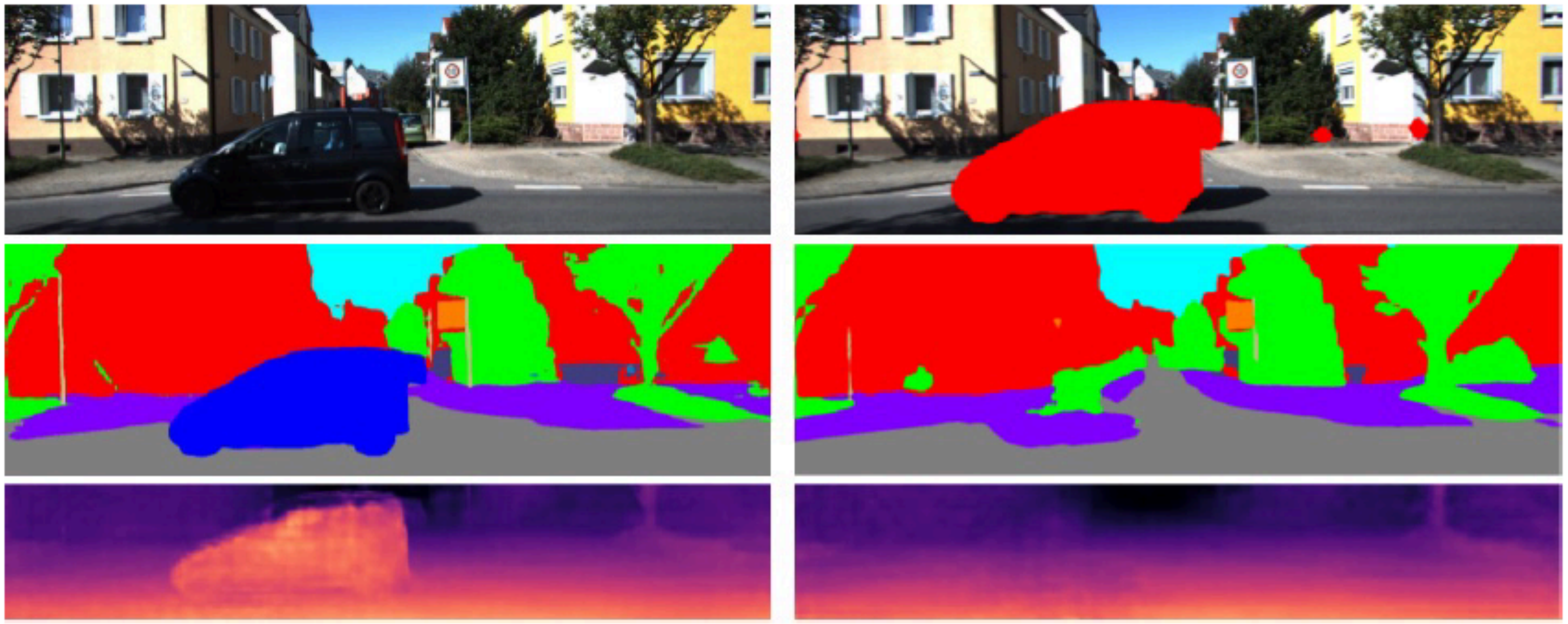


Fig. 6: Qualitative example of our hallucination CNN: Semantics and depth without (left) and with (right) hallucination.

Birds eye view from depth + labels

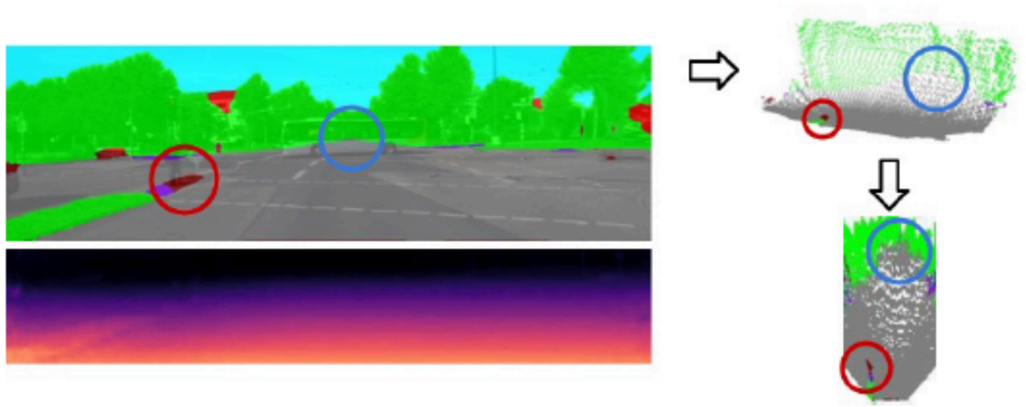


Fig. 3: The process of mapping the semantic segmentation with corresponding depth first into a 3D point cloud and then into the bird's eye view. The red and blue circles illustrate corresponding locations in all views.

Refining birds eye predictions

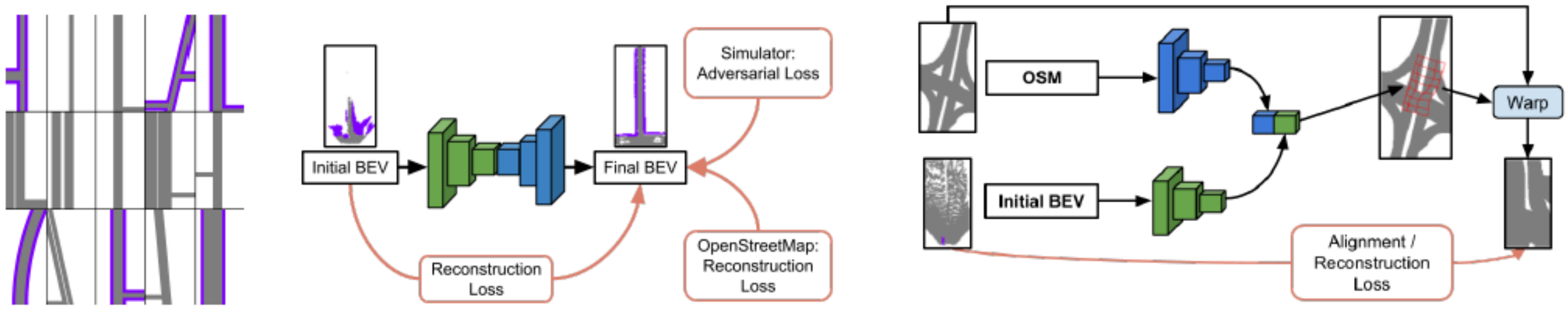
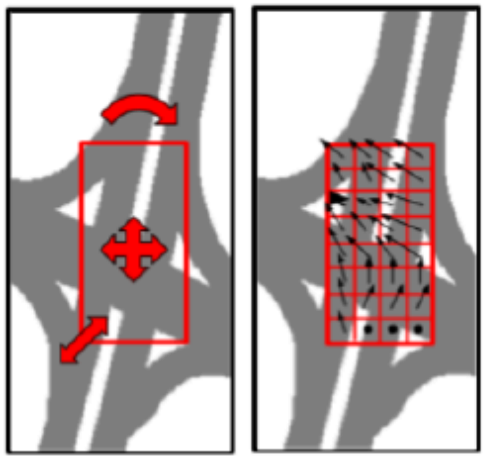
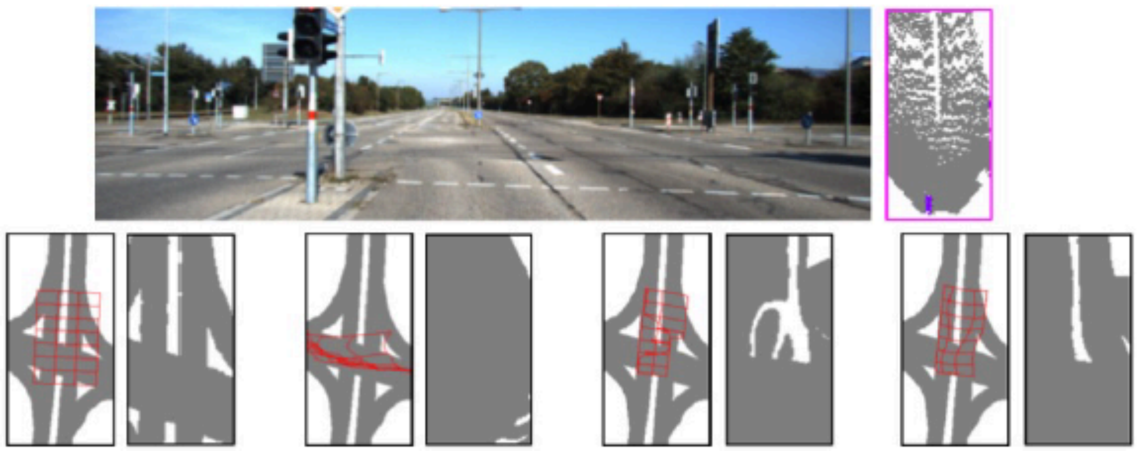

Initial BEV
Final BEV
Simulator: Adversarial Loss
Reconstruction Loss
OpenStreetMap: Reconstruction Loss
OSM
Initial BEV
Warp
Alignment / Reconstruction Loss
(a)
(b)
(c)

Fig. 4: **(a) Simulated road shapes** in the top-view. **(b) The refinement-CNN** is an encoder-decoder network receiving three supervisory signals: self-reconstruction with the input, adversarial loss from simulated data, and reconstruction loss with aligned OpenStreetMap (OSM) data. **(c) The alignment CNN** takes as input the initial BEV map and a crop of OSM data (via noisy GPS and yaw estimate given). The CNN predicts a warp for the OSM map and is trained to minimize the reconstruction loss with the initial BEV map.

Warping OSM to map layout



(a)



(b)

Fig. 5: **(a)** We use a composition of similarity transform (left, "box") and a non-parametric warp (right, "flow") to align noisy OSM with image evidence. **(b, top)** Input image and the corresponding B^{init}. **(b, bottom)** Resulting warping grid overlaid on the OSM map and the warping result for 4 different warping functions, respectively: "box", "flow", "box+flow", "box+flow (with regularization)". Note the importance of composing the transformations and the induced regularization.

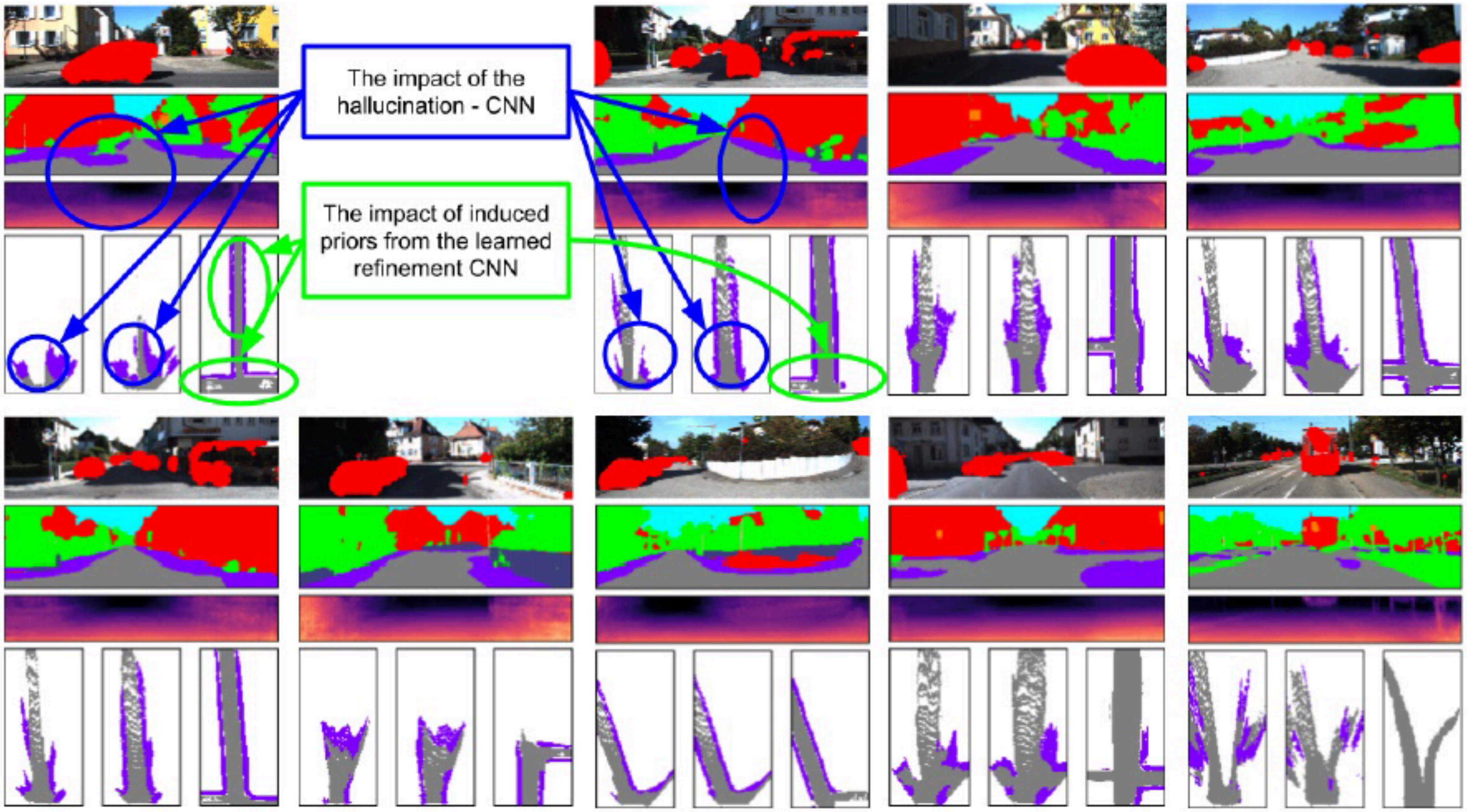
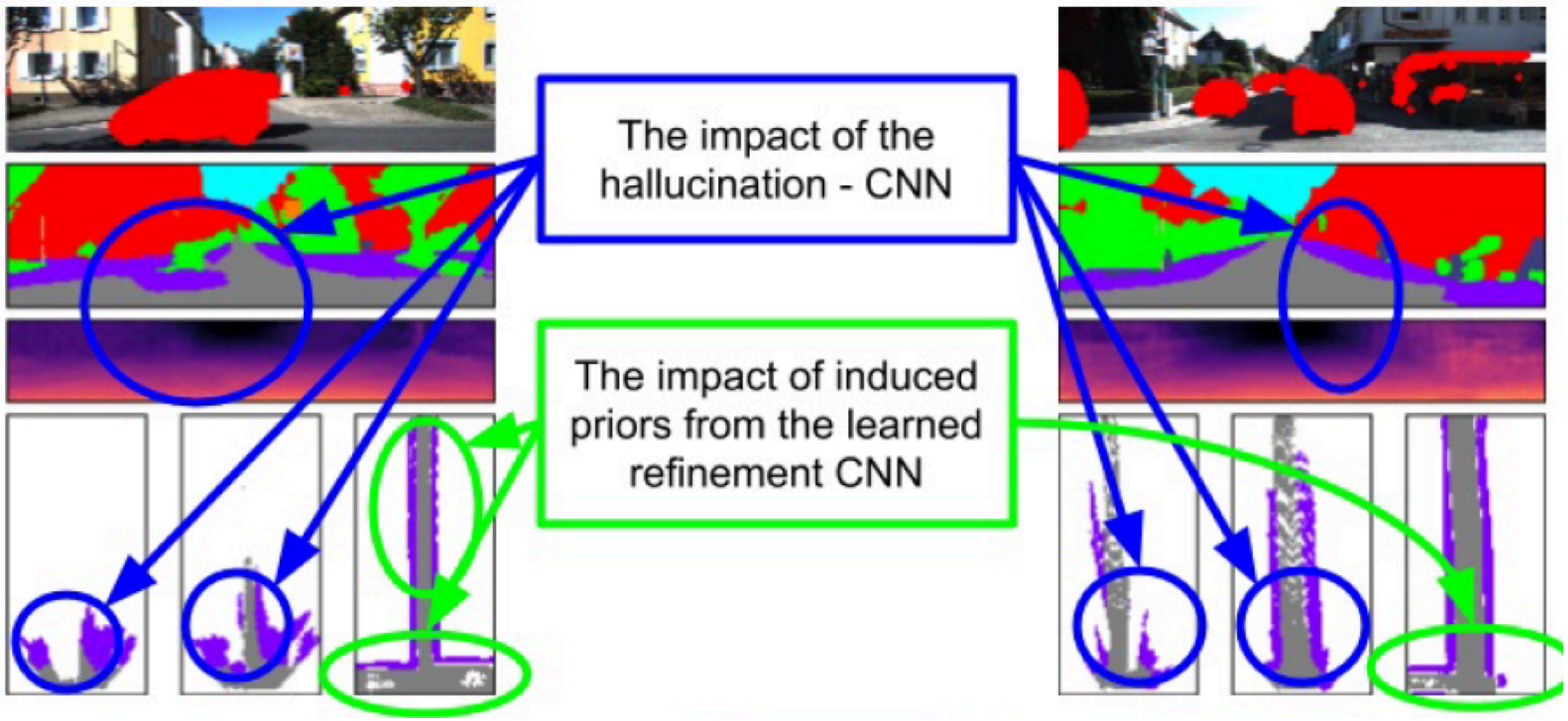
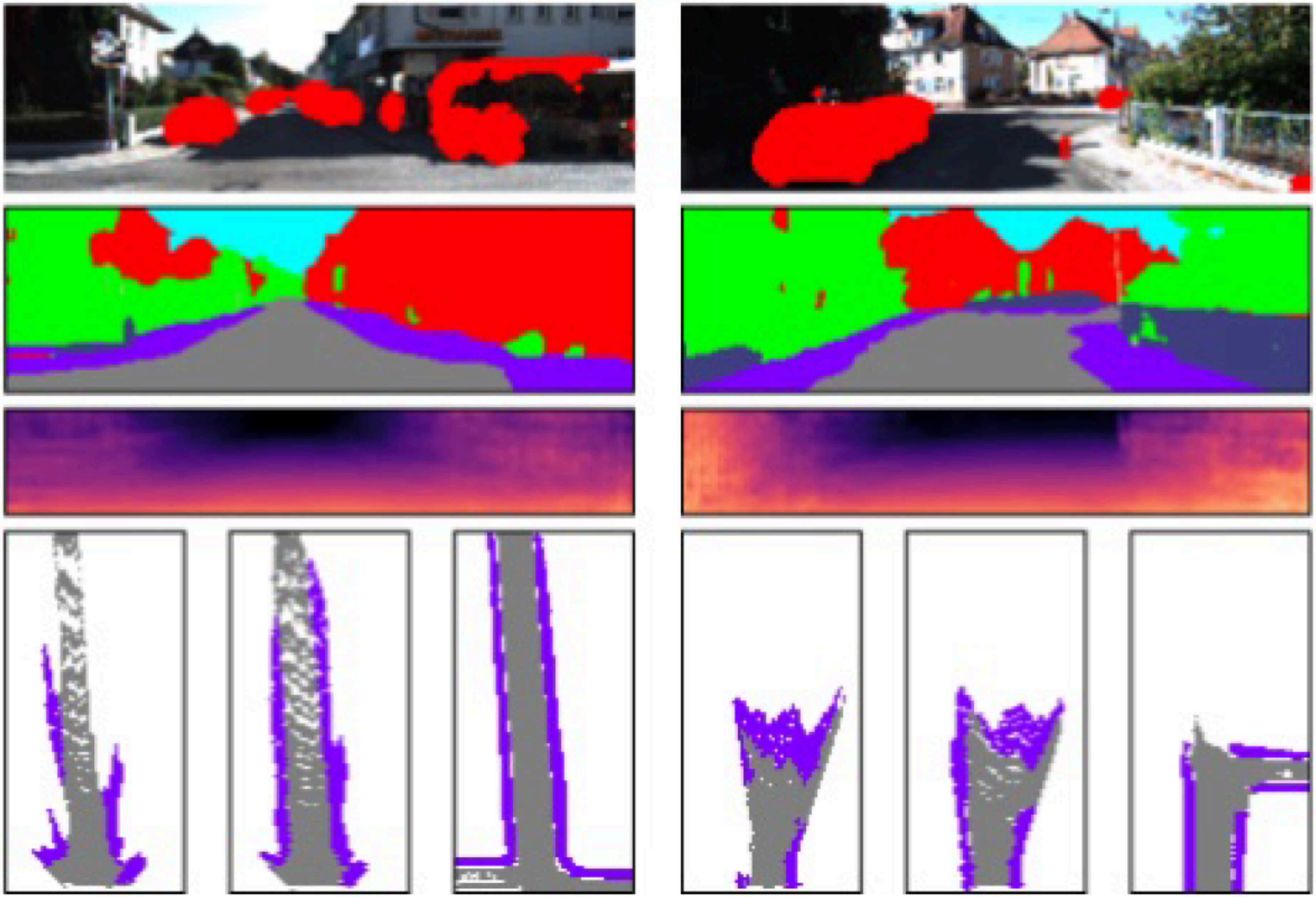

The impact of the hallucination - CNN
The impact of induced priors from the learned refinement CNN

Fig. 8: **Examples of our BEV representation**. Each one shows the masked RGB input, the hallucinated semantics and depth, as well as three BEV maps, which are (from left to right), The BEV map without hallucination, with hallucination, and after refinement. The last example depicts a failure case.


The impact of the hallucination - CNN
The impact of induced priors from the learned refinement CNN



Good + bad

- Birds eye view is a good idea
  - right place to compare labels with models
- Label inpainting is good idea
  - but why in image?
  - the warping, registration seem to help A LOT with this
- It’s clear that warping, registration, adversary are helpful

- Depth inference is a dubious idea
  - Why not use ground plane estimate?
  - and homography?

In overhead view

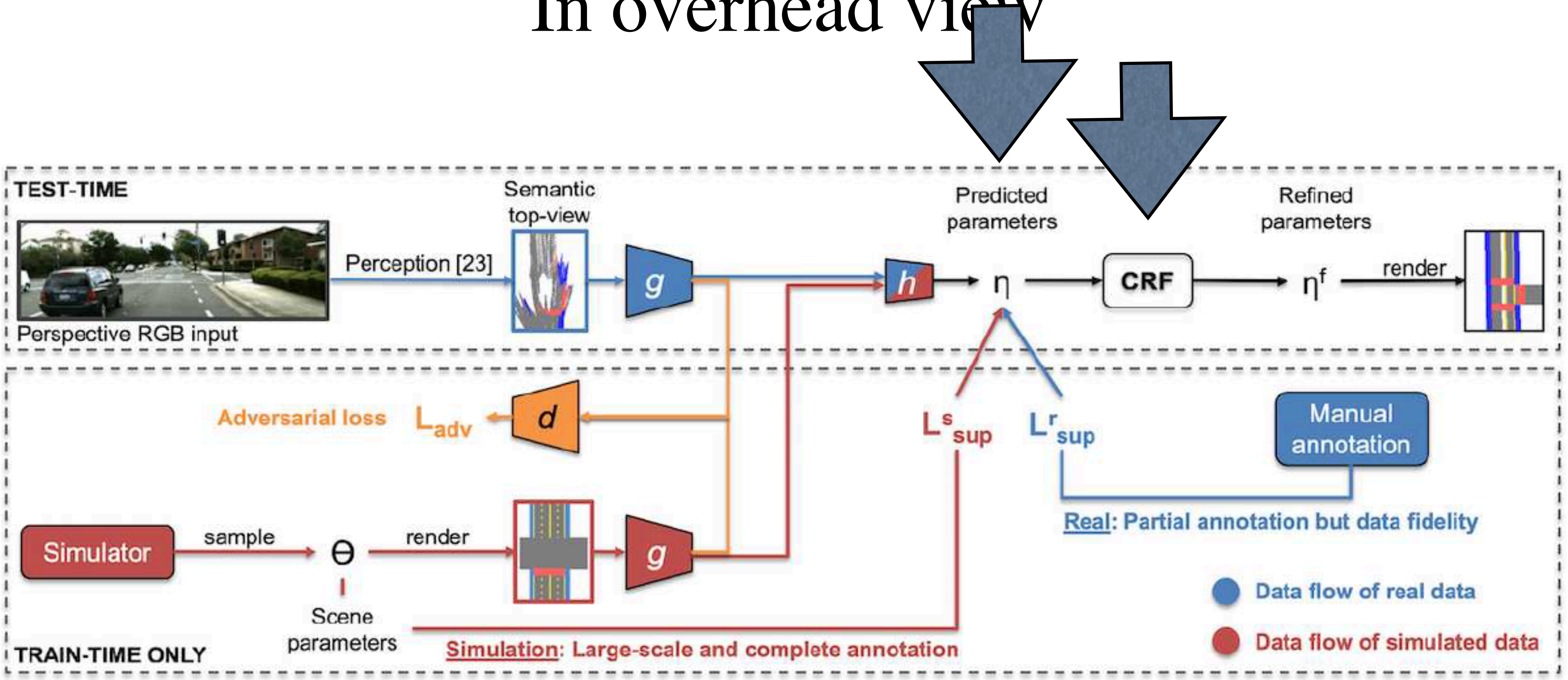


Figure 3: **Overview of our proposed framework:** At train-time, our framework makes use of both manual annotation for real data (blue) and automated annotation for simulated data (red), see Sec. 3.2. The feature extractors g convert semantic top views from either domain into a common representation which is input to h. An adversarial loss (orange) encourages a domain-agnostic output of g. At test-time, an RGB image in the perspective view is first transformed into a semantic top-view [23], which is then used by our proposed neural network (see Sec. 3.3), $h \circ g$, to infer our scene model (see Sec. 3.1). The graphical model defined in Sec. 3.4 ensures a coherent final output.

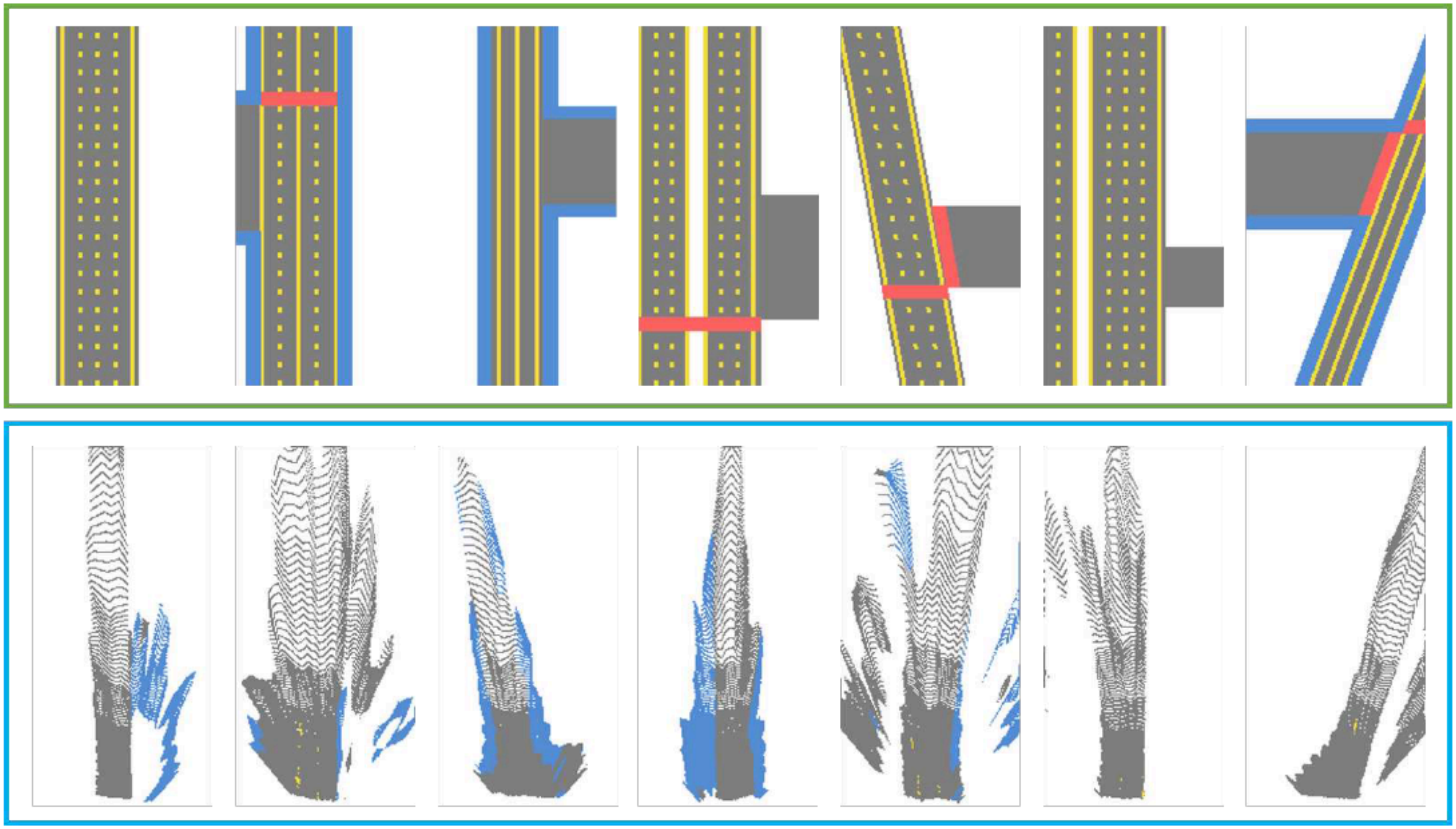


Figure 4: Unpaired examples of simulated semantic top-views (top) and real ones from [23] (bottom).

CRF

- Q: what does this apply to
  - I *think* predicted labels on “ground plane”
    - but what is discretization?

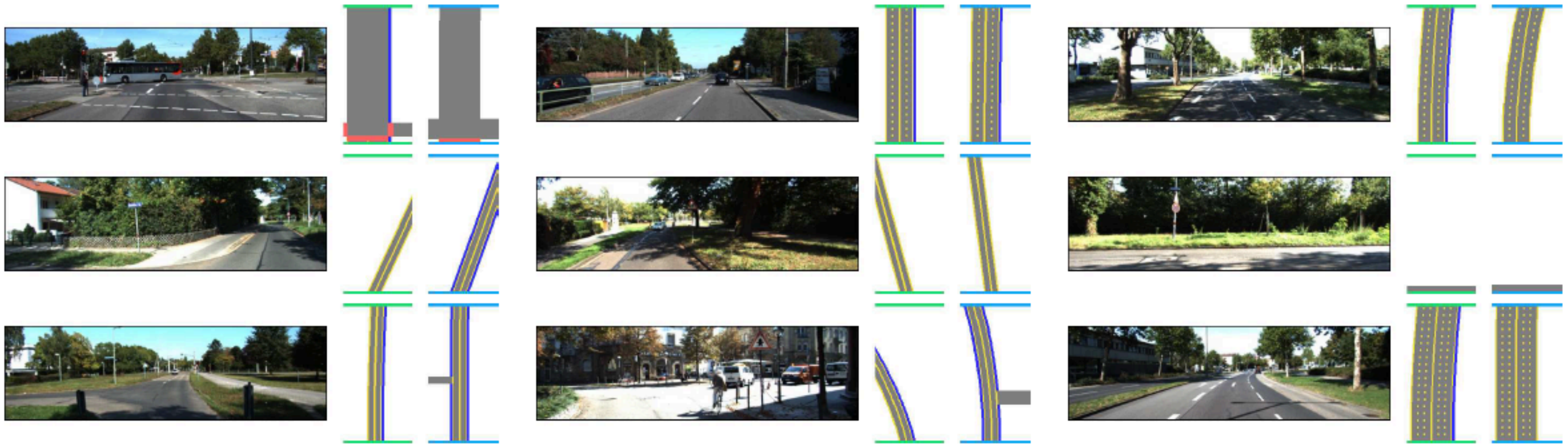


Figure 6: Qualitative results of H-BEV+DA+GM on individual frames from KITTI. Each example shows perspective RGB, ground truth and predicted semantic top-view, respectively. Our representation is rich enough to cover various road layouts and handles complex scenarios, *e.g.*, rotation, existence of crosswalks, sidewalks, side-roads and curved roads.

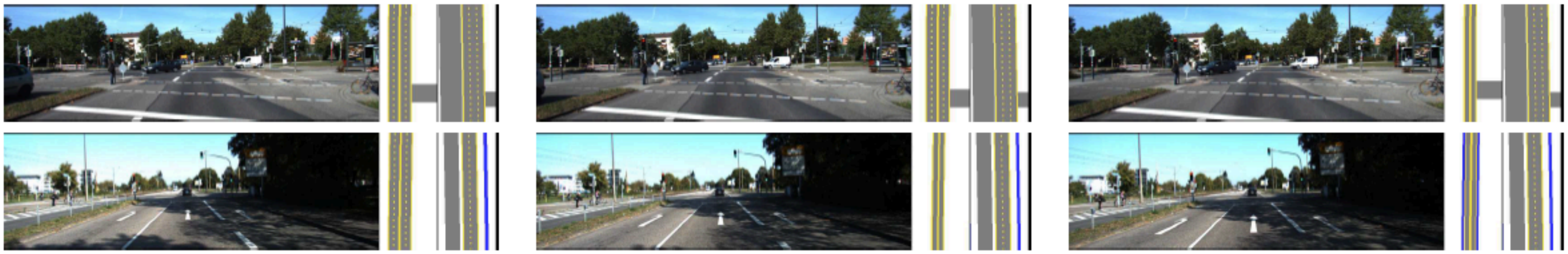


Figure 7: Qualitative results comparing H-BEV+DA and H-BEV+DA+GM in consecutive frames of two example sequences of the KITTI validation set. In each column, we have visualized the perspective RGB image, prediction from H-BEV+DA and that of H-BEV+DA+GM from left to right. Each row shows a sequence of three frames. We can observe more consistent predictions, *e.g.*, width of side-road and delimiter width, with the help of the temporal CRF.

Good + bad

- It’s clear that label fields are highly structured
  - but BEV construction is weird
- This structure is very important and valuable
  - Q: can we exploit without OSM, Streetview, etc.?

Scene Flow and Ways to Infer It

- particularly photometric consistency
  - a version of this applies to scene inference

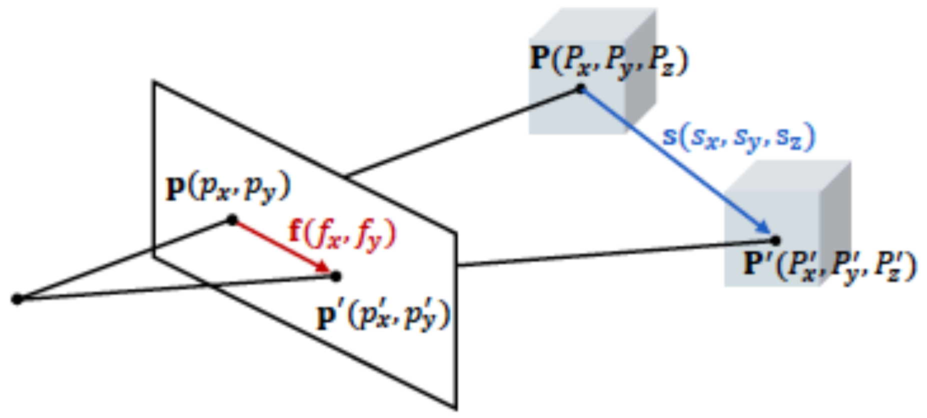
Scene Flow

- From pair of images/images+depths/stereo pairs infer
  - for each image point (x, y, z) AND (v_x, v_y, v_z)
- Rigid scene
  - easy for stereo pair/image+depth pair:
    - (v_x, v_y, v_z) follow from depth and camera ego-motion
  - harder for image pair
    - depth, scene flow ambiguity
- We assume there are moving objects

Scene Flow

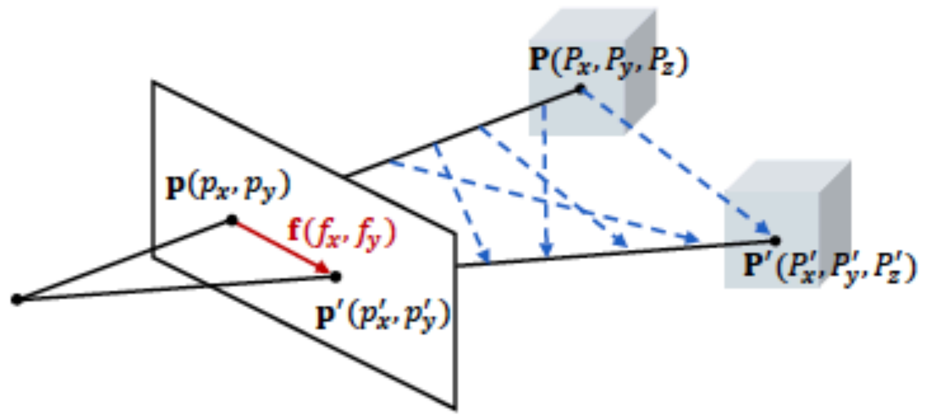
- From pair of images/images+depths/stereo pairs/lidar infer
  - for each (image) point (x, y, z) AND (v_x, v_y, v_z)
- Rigid scene
  - easy for stereo pair/image+depth pair:
    - (v_x, v_y, v_z) follow from depth and camera ego-motion
  - harder for image pair
    - depth, scene flow ambiguity
- We assume there are moving objects

Ambiguity



$\mathbf{P}(P_x, P_y, P_z)$
$\mathbf{s}(s_x, s_y, s_z)$
$\mathbf{p}(p_x, p_y)$
$\mathbf{f}(f_x, f_y)$
$\mathbf{P}'(P'_x, P'_y, P'_z)$
$\mathbf{p}'(p'_x, p'_y)$

(a) Projecting scene flow into 2D space.



$\mathbf{P}(P_x, P_y, P_z)$
$\mathbf{p}(p_x, p_y)$
$\mathbf{f}(f_x, f_y)$
$\mathbf{P}'(P'_x, P'_y, P'_z)$
$\mathbf{p}'(p'_x, p'_y)$

(b) Back-projecting optical flow into 3D space.

Figure 2. **Relating monocular scene flow estimation to optical flow:** *(a)* Projection of scene flow into the image plane yields optical flow [59]. *(b)* Back-projection of optical flow leaves an ambiguity in jointly determining depth and scene flow.

- Notice there are no problems if you know depth

Menze 2015

Typical scene flows

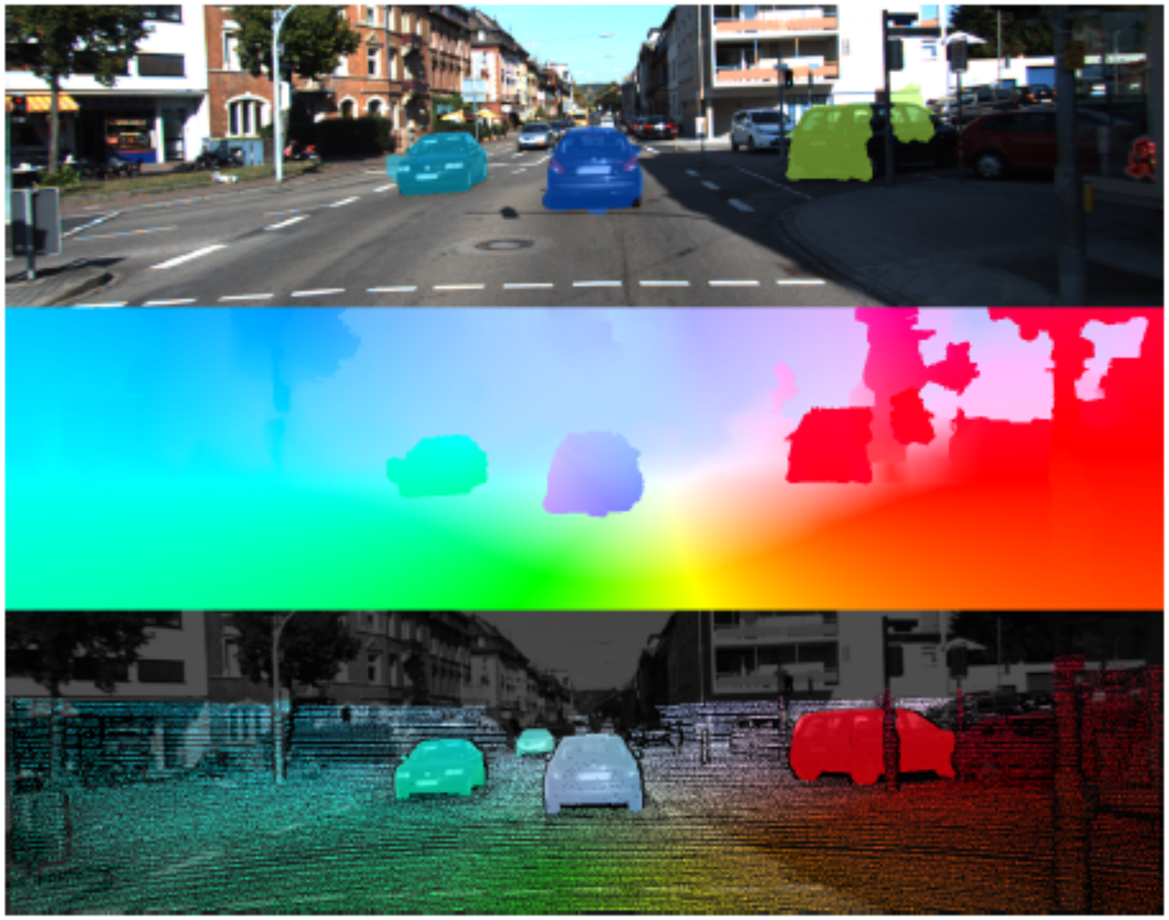


Figure 1: **Scene Flow Results on the proposed Dataset.** Top-to-bottom: Estimated moving objects with background object in transparent, flow results and flow ground truth.

Menze 2015

Estimation for stereo

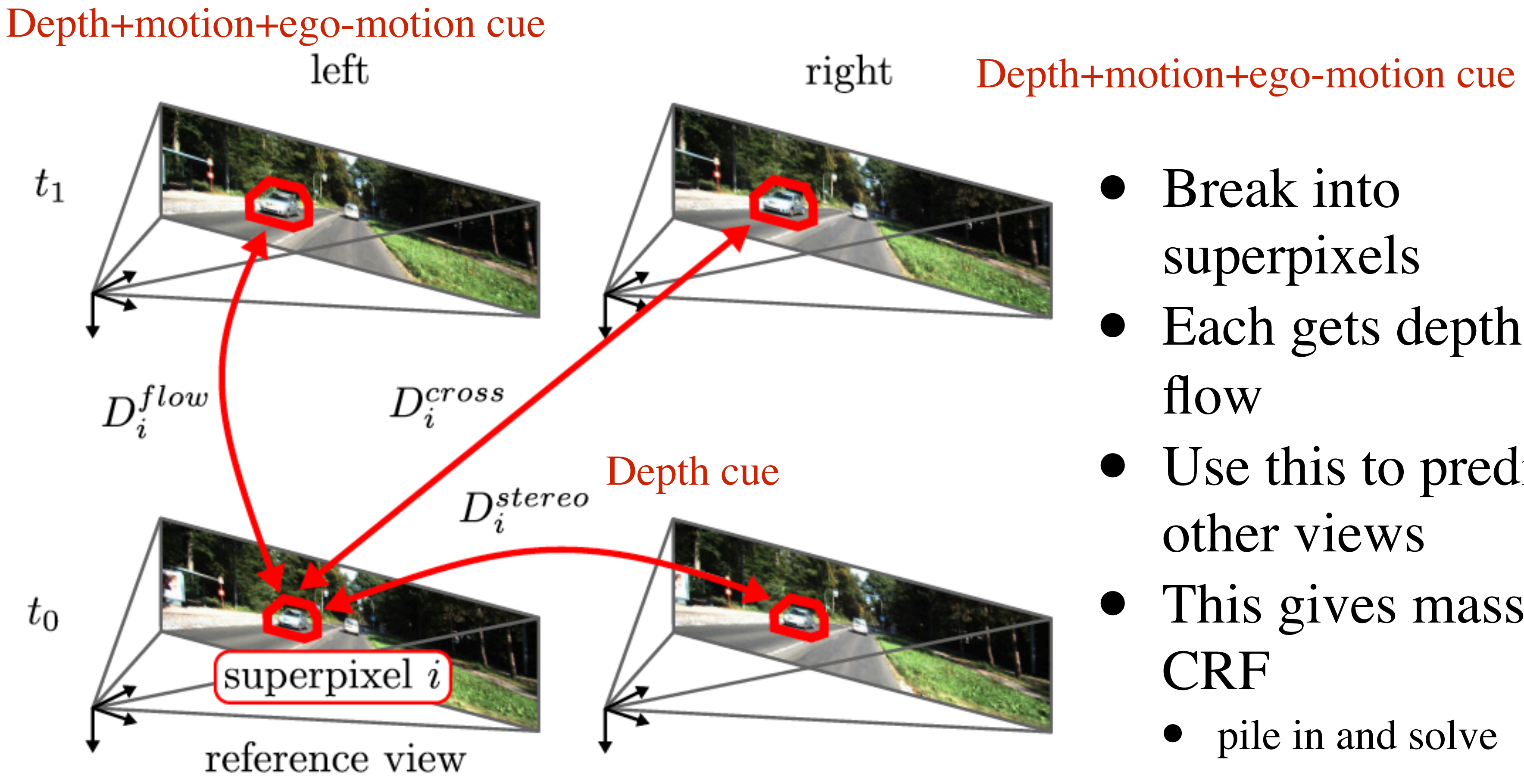


Figure 2: **Data Term.** Each superpixel i in the reference view is modeled as a rigidly moving 3D plane and warped into each other image to calculate matching costs. Each of the superpixels is associated with a 3D plane variable and a pointer to an object hypothesis comprising its rigid motion.

- Break into superpixels
- Each gets depth, flow
- Use this to predict in other views
- This gives massive CRF
    - pile in and solve

Menze 2015

Lagniappe: Scene flow in LIDAR

- Learn without labelled data
- ICP isn’t quite enough
  - objects might contract, for example
  - use a cycle consistency loss
    - f_ab = 3Da -> 3Db
    - we must have f_ba(f_ab(x))=x
    - trick:
      - as stated, this is unstable
      - instead, f_ba(0.5 f_ab(x)+ 0.5 NN(f_ab(x))) close to x
      - this also avoids problems with zero flow

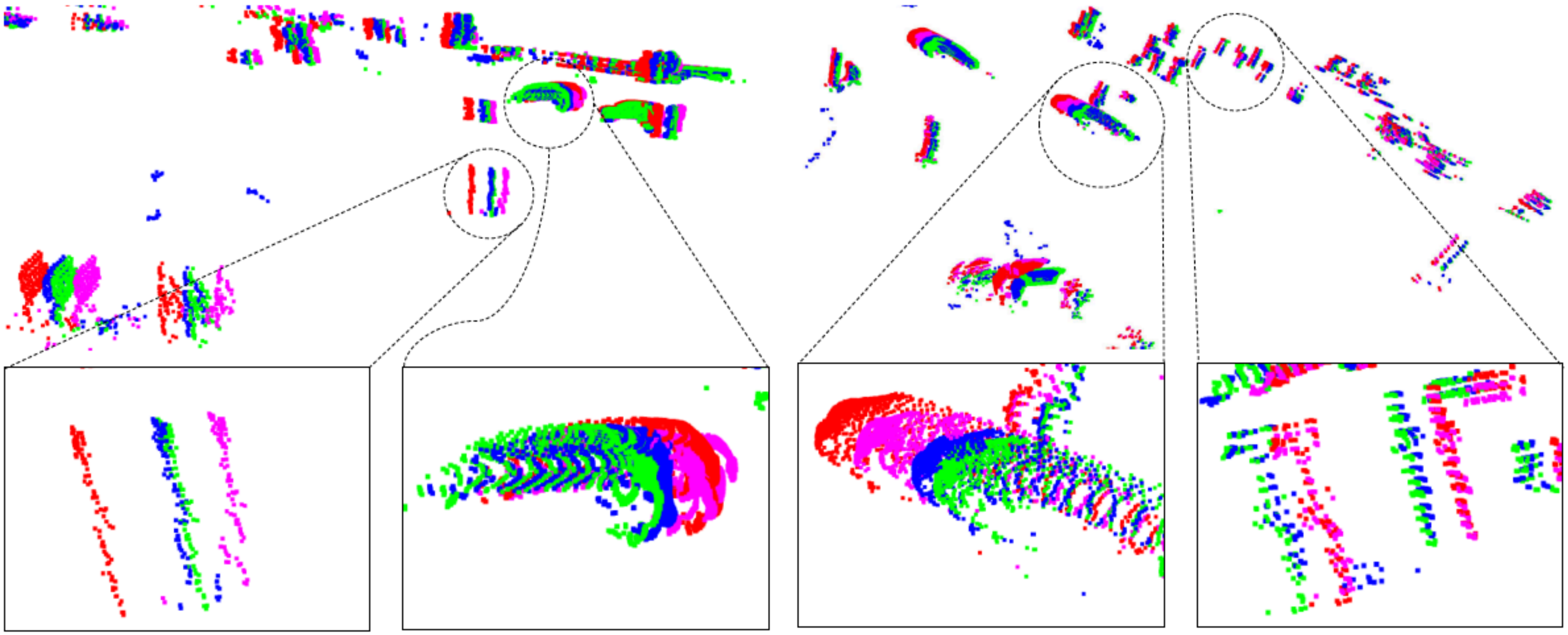
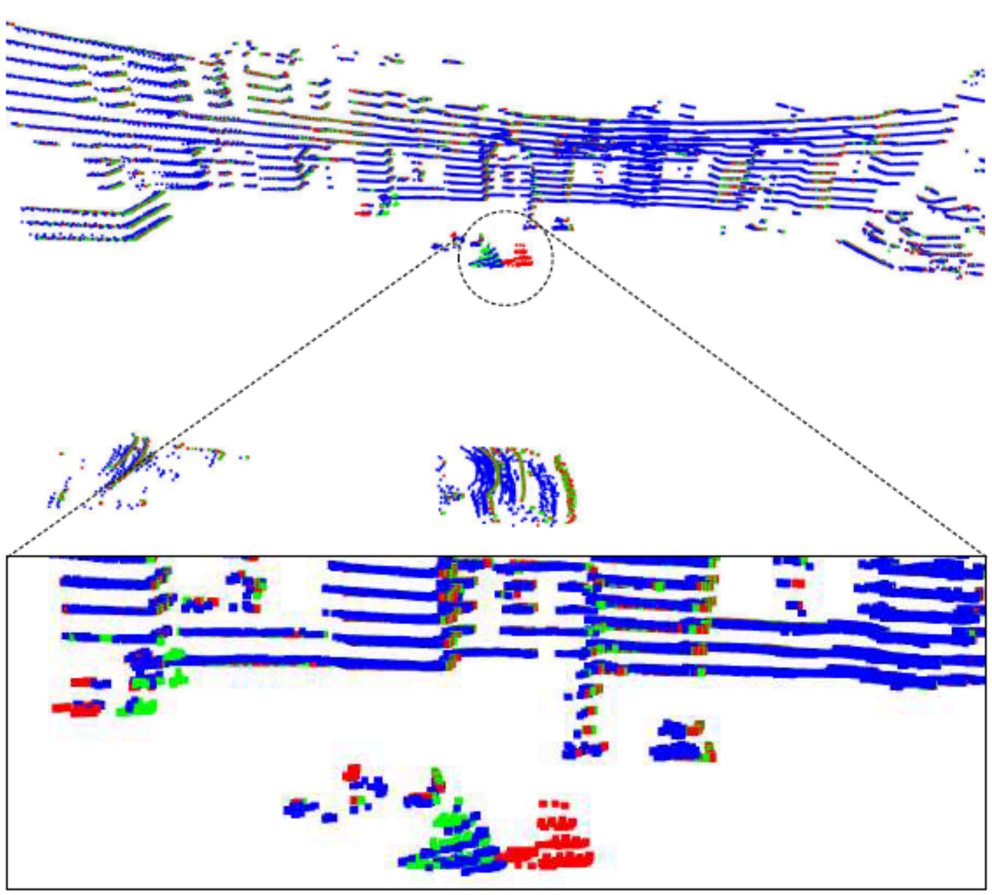


Figure 4: Scene flow estimation between point clouds at time t (red) and $t+1$ (green) from the KITTI dataset trained without any labeled LIDAR data. Predictions from our self-supervised method, trained on nuScenes and fine-tuned on KITTI using self-supervised learning is shown in blue; the baseline with only synthetic training is shown in purple. In the absence of real-world supervised training, our method clearly outperforms the baseline method, which overestimate the flow in many regions. (Best viewed in color)

Mittal 20


(a) Ours (Self-Supervised Fine Tuning)

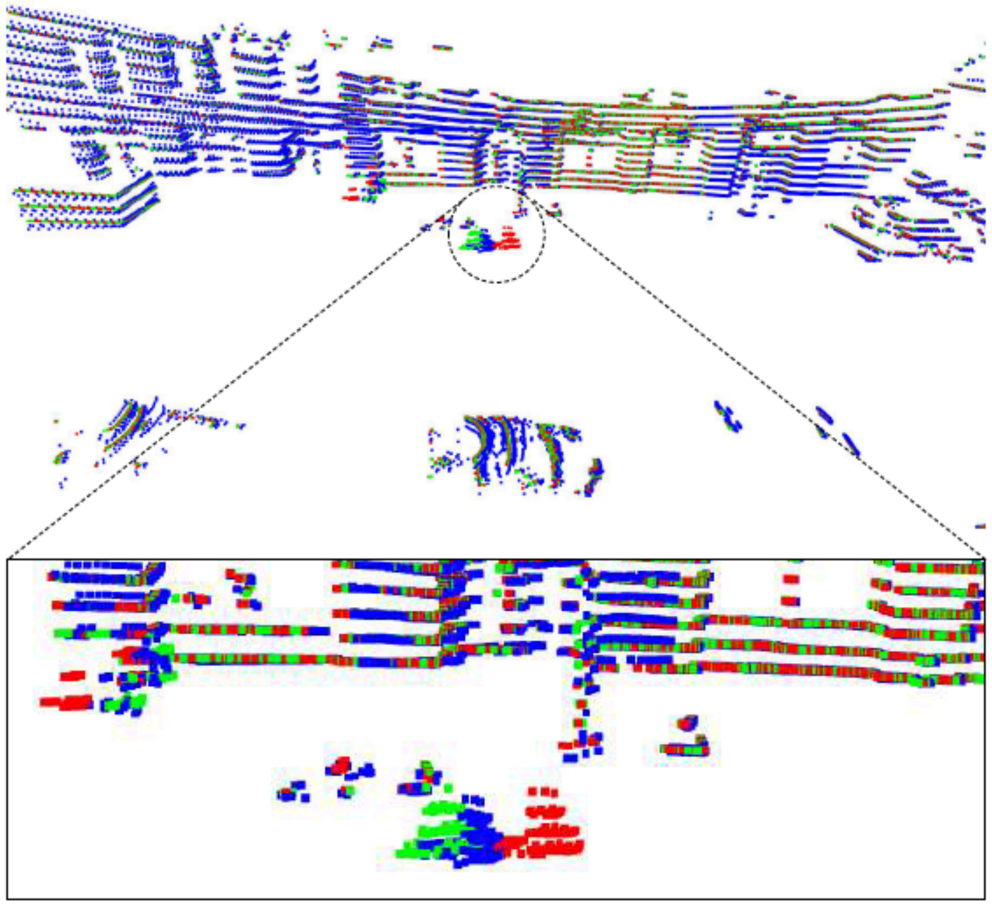
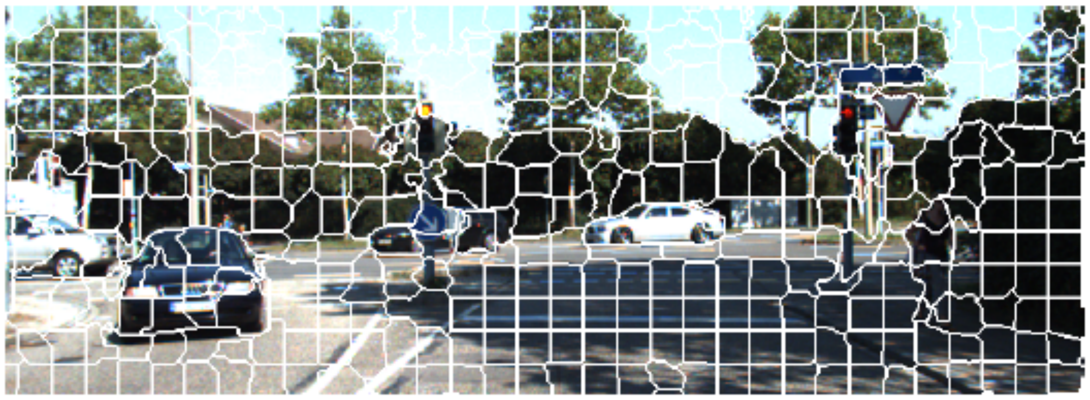

(b) Baseline (No Fine Tuning)

Figure 5: Comparison of our self-supervised method to a baseline trained only on synthetic data, shown on the nuScenes dataset. Scene flow is computed between point clouds at time t (red) and $t + 1$ (green); the point cloud that is transformed using the estimated flow is in shown in blue. In our method, the predicted point cloud has a much better overlap with the point cloud of the next timestamp (green) compared to the baseline. Since nuScenes dataset does not provide any scene flow annotation, the supervised approaches cannot be fined tuned to this environment.

Mittal 20

Scene flow in single images

- Predict depth from single image
  - using network which makes mixture of normals in depth at location
  - trained using existing image-depth data
- Break image into superpixels
  - each is a plane section that moves rigidly
    - to infer: plane params, motion params (9 total per superpixel)
-



Brickwedde 19

Scene flow in single images

Photometric consistency

$\downarrow$

- CRF
  - unary losses:
    - plane section motion should predict next frame pixel values well
    - plane section should model predicted depth well
  - binary losses:
    - plane sections should have compatible depths on boundary
    - normals of neighbors should be similar

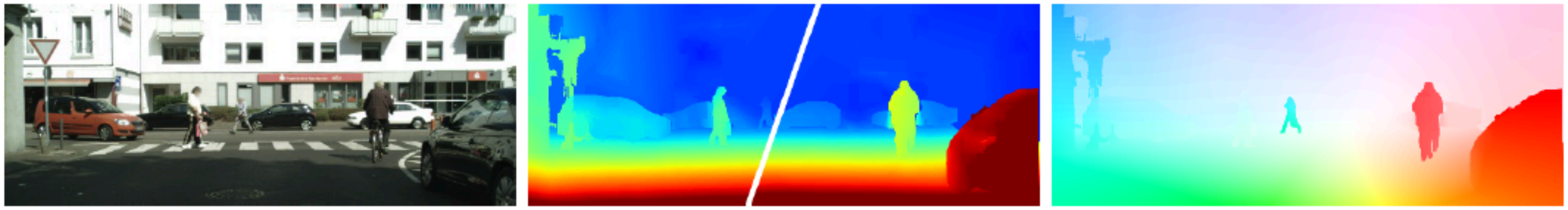
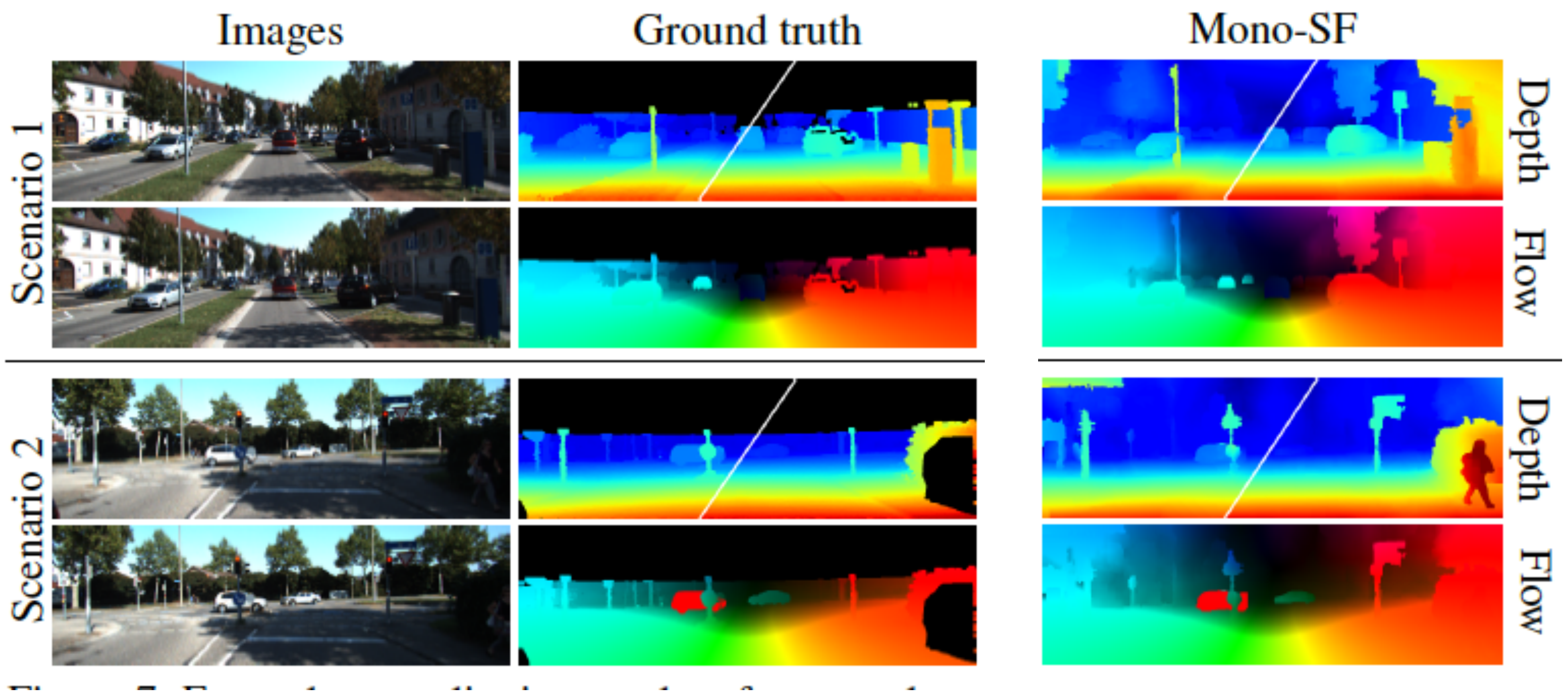


Figure 8. Exemplary qualitative result of Mono-SF on a crop of Cityscapes (removing car hood); left: first input image, middle: estimated depth values at time $t = 0$ (left half) and $t = 1$ (right half), right: estimated optical flow

Brickwedde 19


Images
Ground truth
Mono-SF
Scenario 1
Scenario 2
Depth
Flow
Depth
Flow

Scene flow in single images

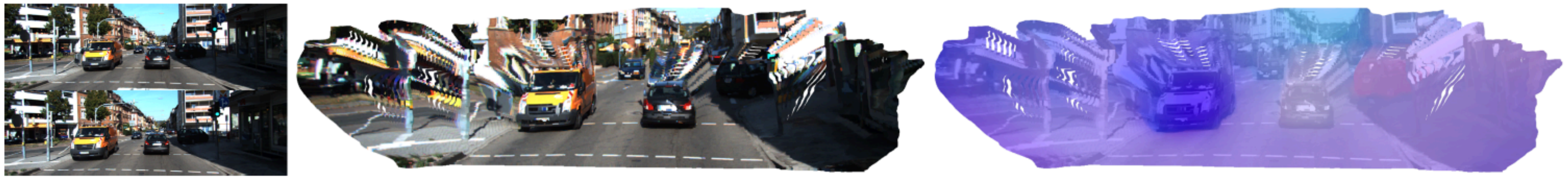


Figure 1. **Results of our monocular scene flow approach on the KITTI dataset [11].** Given two consecutive images *(left)*, our method jointly predicts depth *(middle)* and scene flow *(right)*. (x,z)-coordinates of 3D scene flow are visualized using an optical flow color coding.

- Straightforward network prediction of scene flow
  - depth ambiguity?
    - semantics, etc. resolve
    - *train* with stereo pairs
  - cues
    - single image depth cues (texture)
    - photometric consistency
      - optic flow

Hur 20

Single image depth estimator for disparity

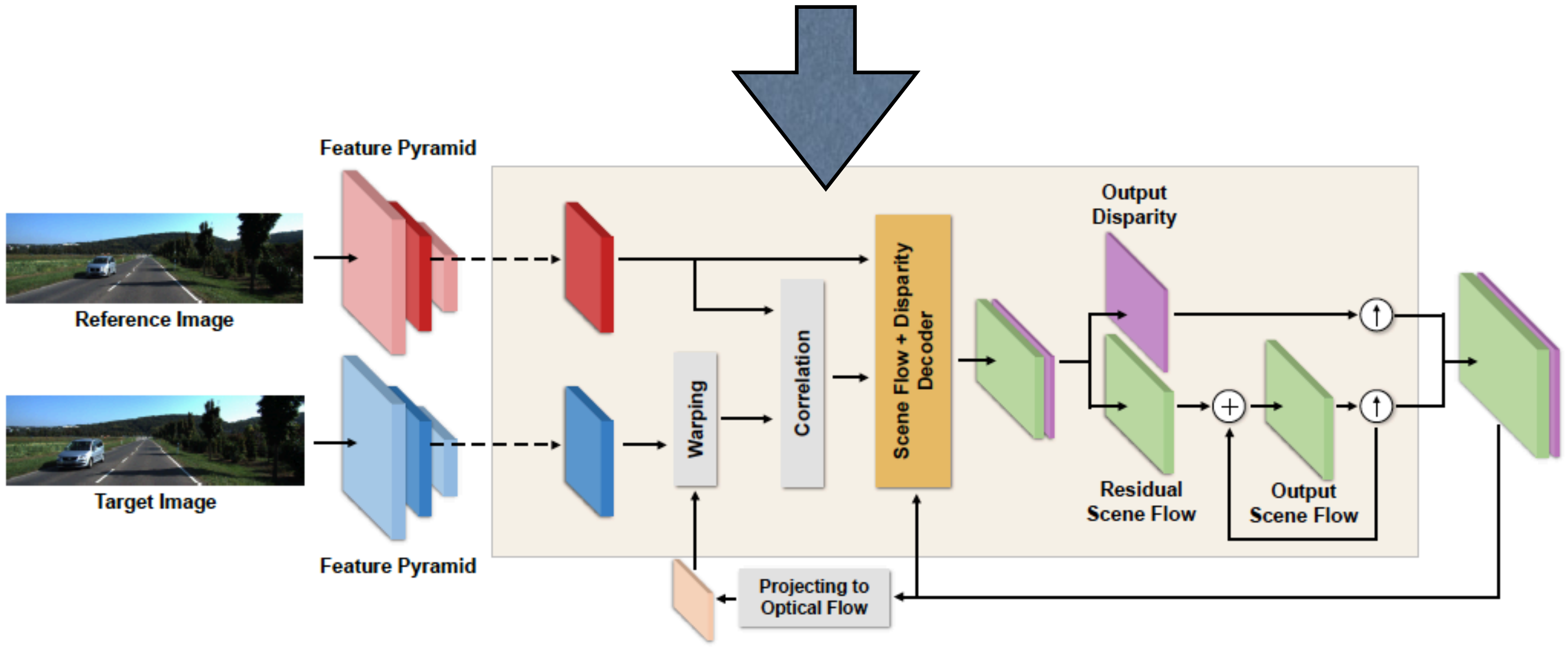
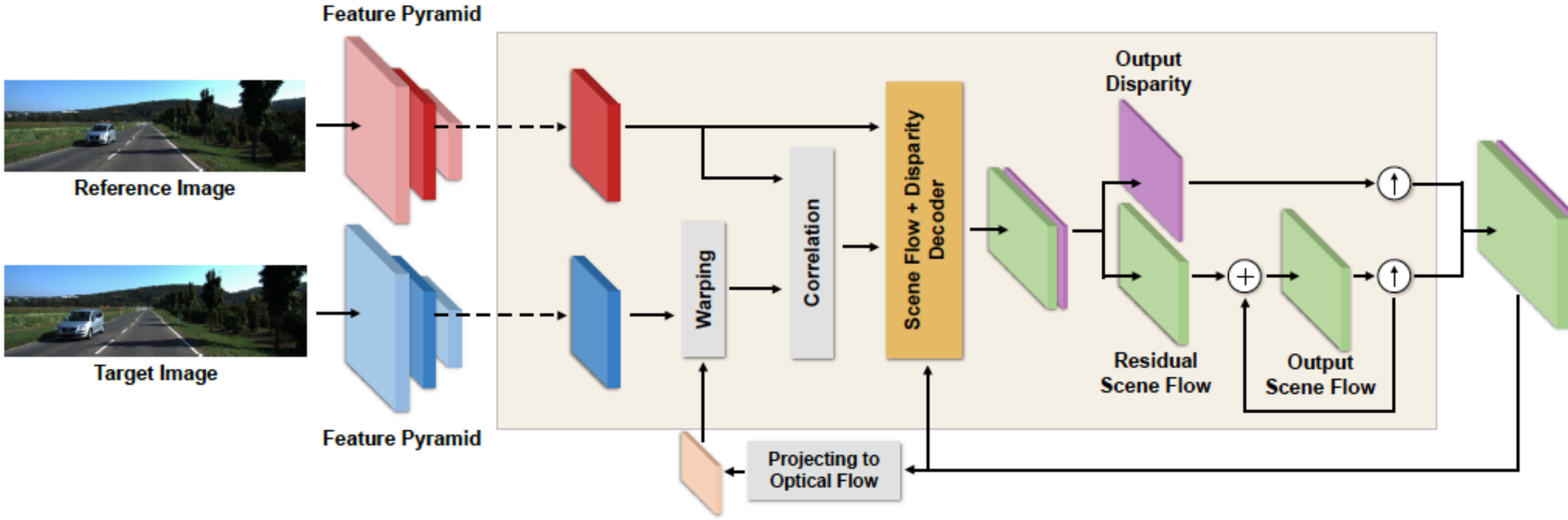
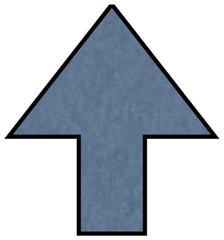


Figure 3. **Our monocular scene flow architecture based on PWC-Net [45]**: while maintaining the overall original structure of PWC-Net, we modify the decoder to output *residual scene flow* and (non-residual) *disparity* together. After the residual update of scene flow, we project the scene flow back to optical flow using depth. Then, the optical flow is used for warping the feature map (only 3 of 7 levels shown for ease of visualization) in the next pyramid level. The light-yellow shaded region shows one forward pass for each pyramid level.

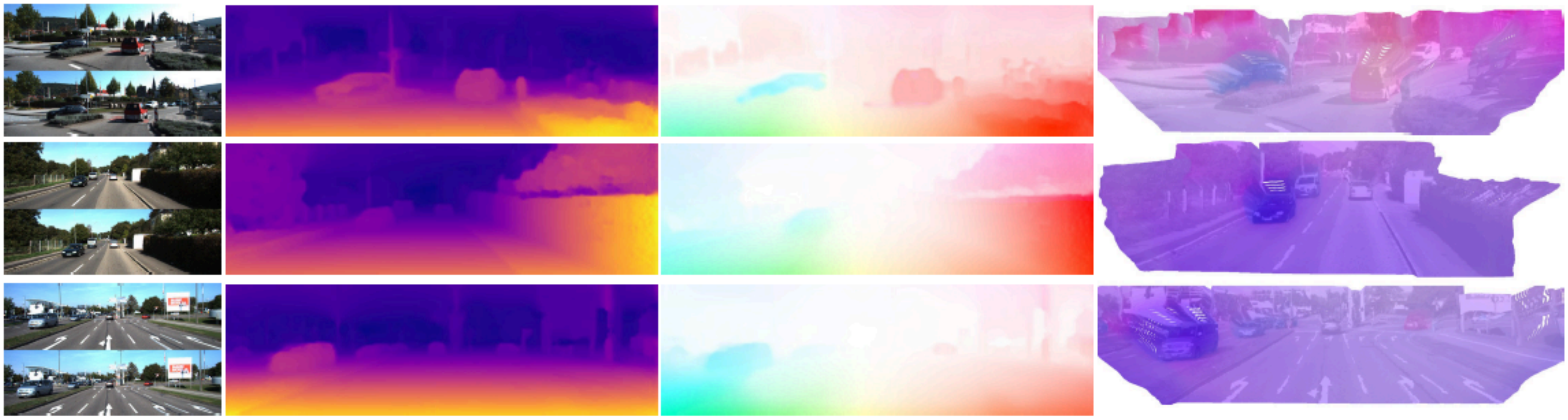




Scene flow predictions warped back to optic flow;
this allows photometric consistency to be imposed
at run time

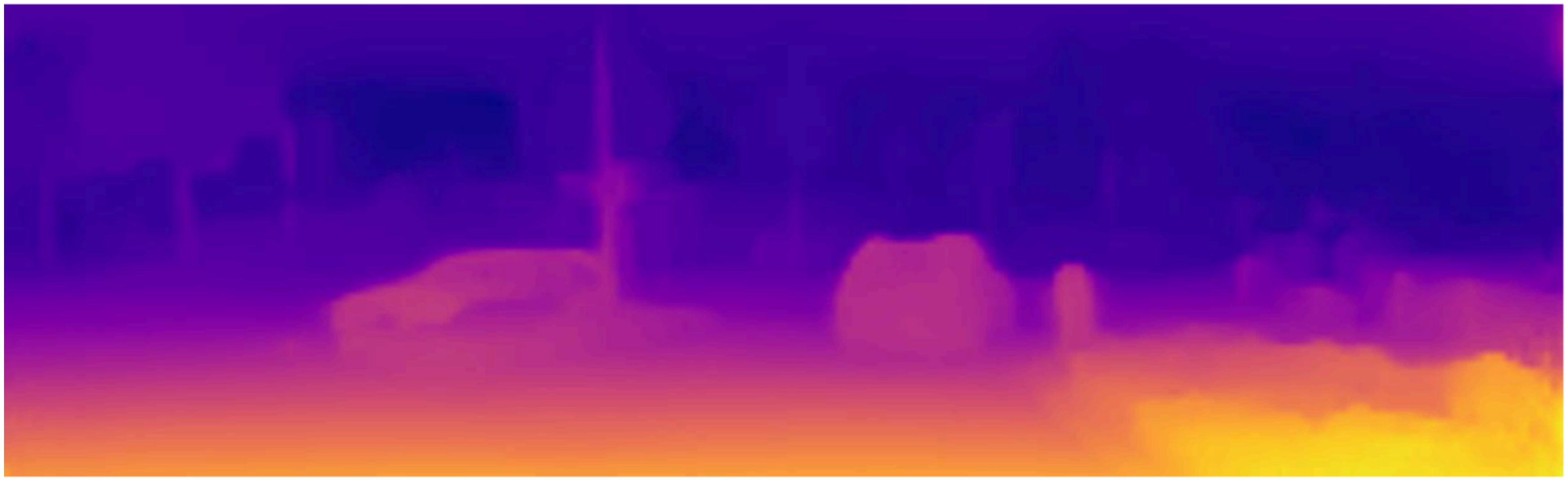
Training losses

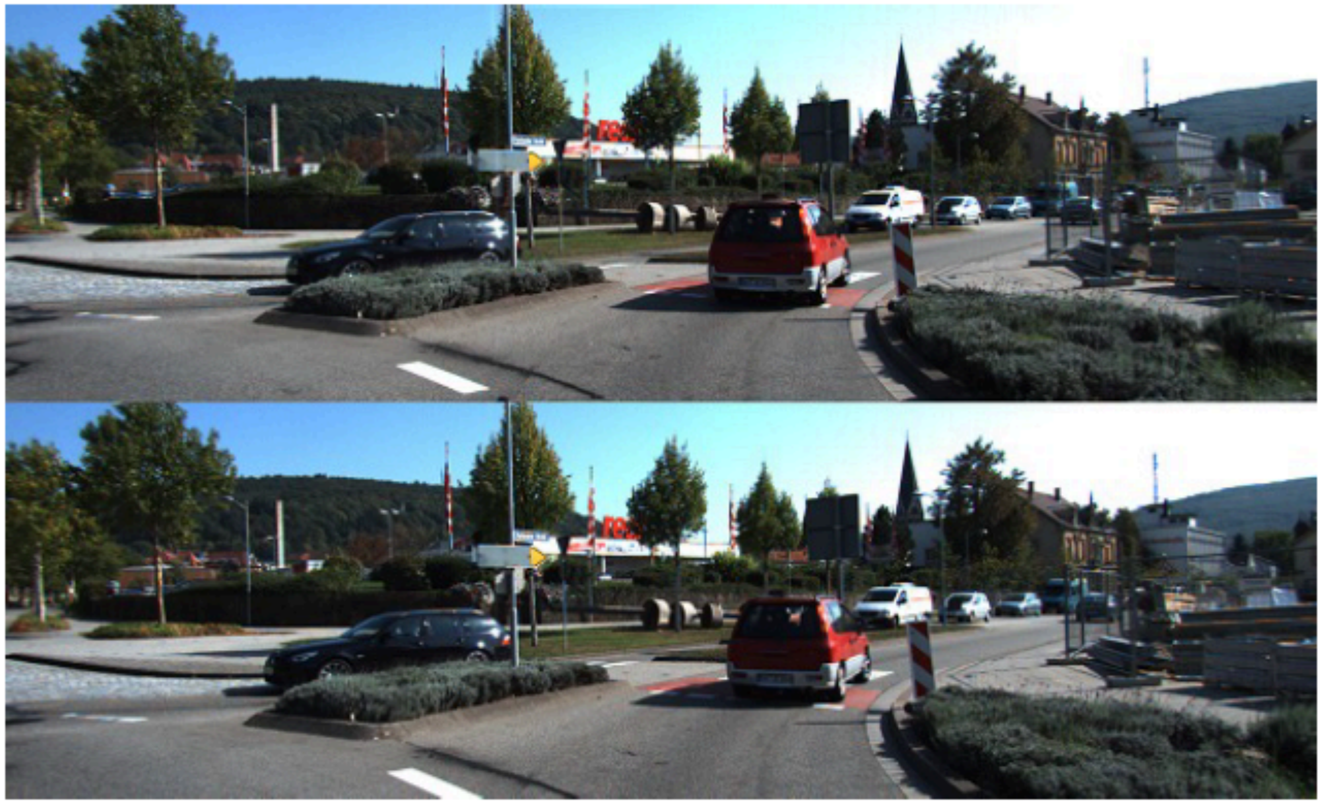
- Disparity predictions should be good
  - train with stereo pairs for this
  - disparity should predict color in other frame (in training)
  - disparity should be smooth
- Photometric consistency
  - scene flow should predict pixel values in next frame
- Point consistency
  - scene flow should predict depth in next frame
- Smoothness
  - scene flow at a point should be similar to neighbors

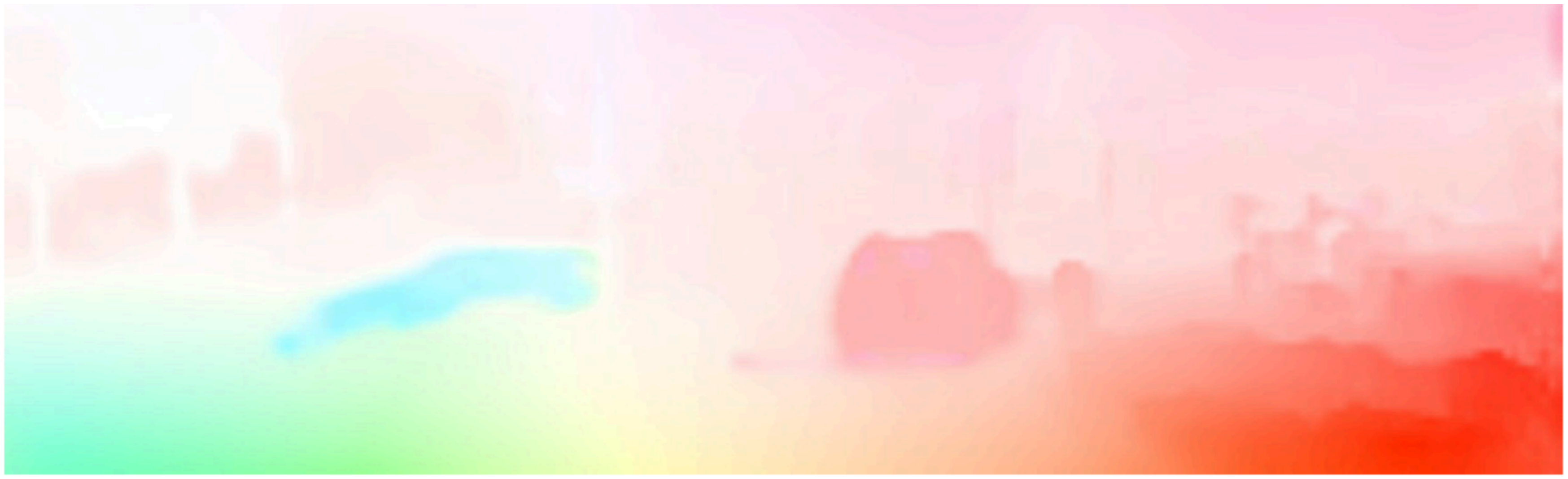


(a) Input images (b) Monocular depth (c) Optical flow (d) 3D visualization of scene flow

Figure 5. **Qualitative results of our monocular scene flow results (Self-Mono-SF-ft) on KITTI 2015 Scene Flow Test**: each scene shows *(a)* two input images, *(b)* monocular depth, *(c)* optical flow, and *(d)* a 3D visualization of estimated depth, overlayed with the reference image, and colored with the (x, z)-coordinates of the 3D scene flow using the standard optical flow color coding.







Notice

- Straightforward consistency losses are very powerful
- Minimal use of labelled data
  - (augmentation by stereo pairs, but no labelling)
- Some form of photometric consistency loss for labels
  - eg
    - predict layout map 1
    - move forward
    - predict layout map 2
    - they should register
    - things that have the same label (tar, paint, junction, etc.)
      - should look similar

Appearance Consistency and Clustering

- Map image into some feature space so that
  - patches that “look similar” are “close”
  - without markup
- Why?
  - because doing so would help produce a layout map eg
    - attach labels to clusters using current maps
    - improve maps using labels

Deep Embedding Clustering

- Compute embedding that
  - autoencodes
  - clusters well

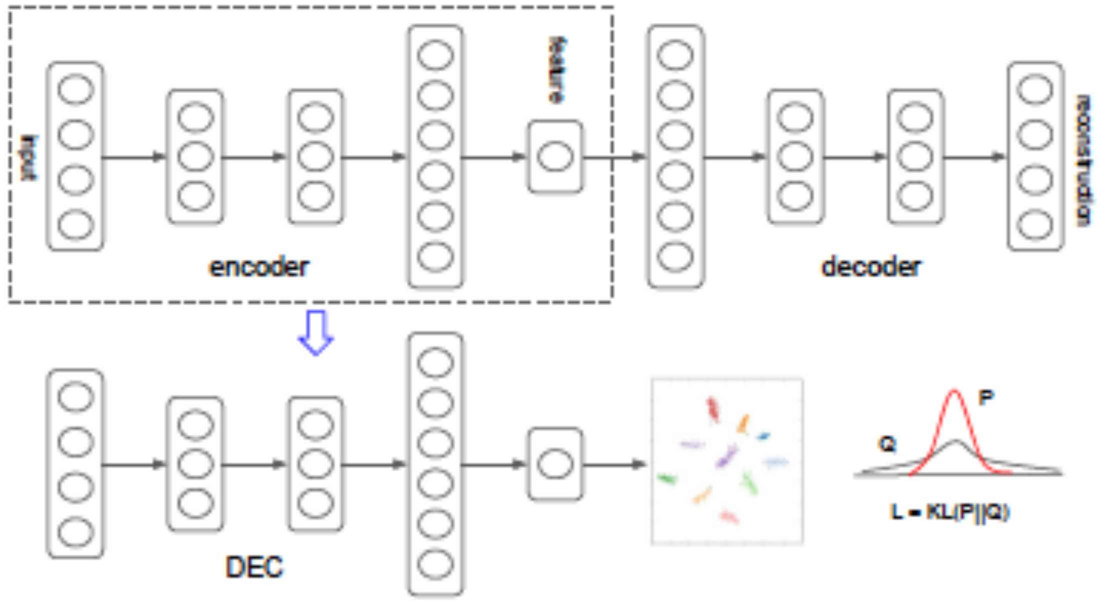


Figure 1. Network structure

Xie et al 15

Clustering

- Cluster centers mu_j must be estimated
  - form membership weights as in TSNE (alpha=1) ->

Following van der Maaten & Hinton (2008) we use the Student's t-distribution as a kernel to measure the similarity between embedded point z_i and centroid μ_j:

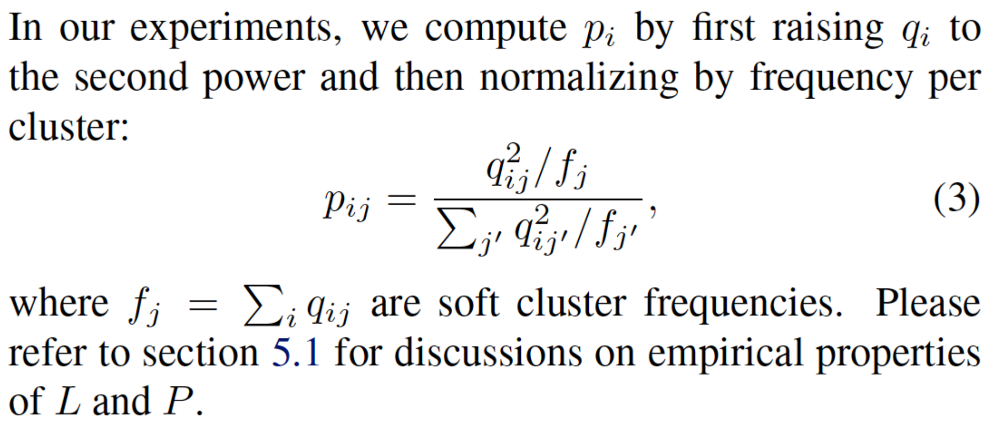
$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}}, \tag{1}$$

- We want these weights to match a target distribution
  - p_ij=target for j'th cluster on i'th point
  - KL divergence (as in TSNE)

$$L = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2}$$

Clustering-II

- But what are p?
  - notice we have some form of reestimation going on here



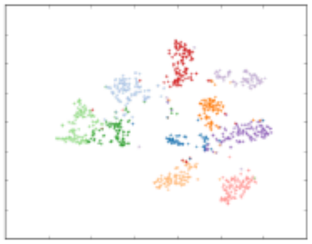
In our experiments, we compute p_i by first raising q_i to the second power and then normalizing by frequency per cluster:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}}, \qquad (3)$$

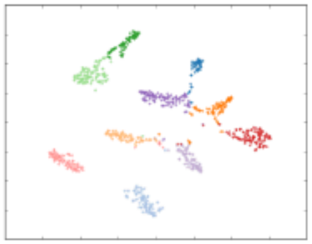
where $f_j = \sum_i q_{ij}$ are soft cluster frequencies. Please refer to section 5.1 for discussions on empirical properties of L and P.

- After that, just descend
  - note autoencoder initialization would probably be done differently now

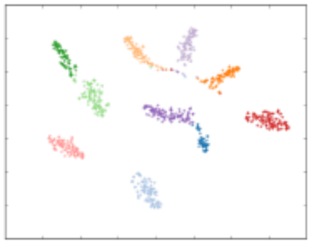
Clustering



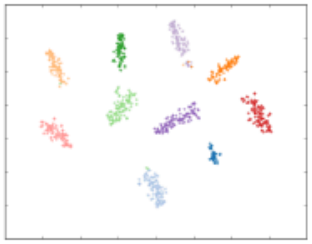
(a) Epoch 0



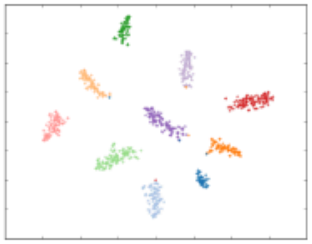
(b) Epoch 3



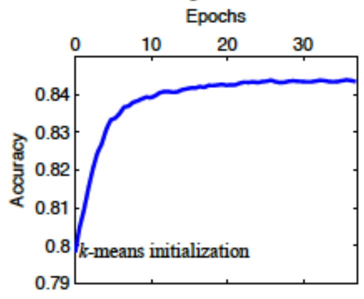
(c) Epoch 6



(d) Epoch 9



(e) Epoch 12



(f) Accuracy vs. epochs

Xie et al 15

Clustering

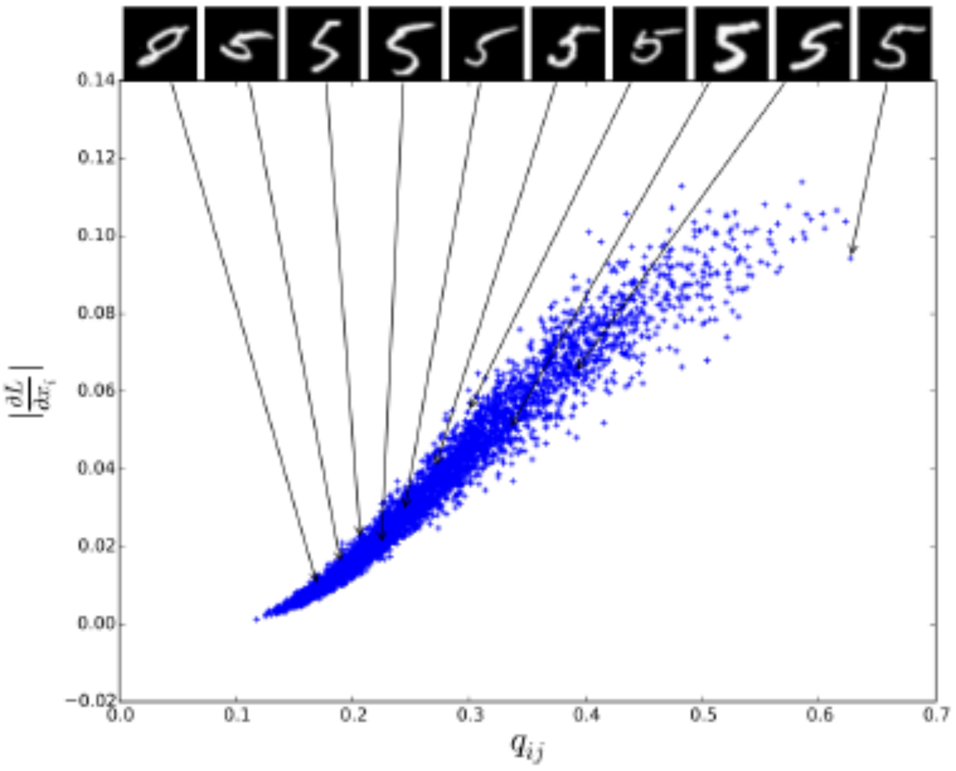
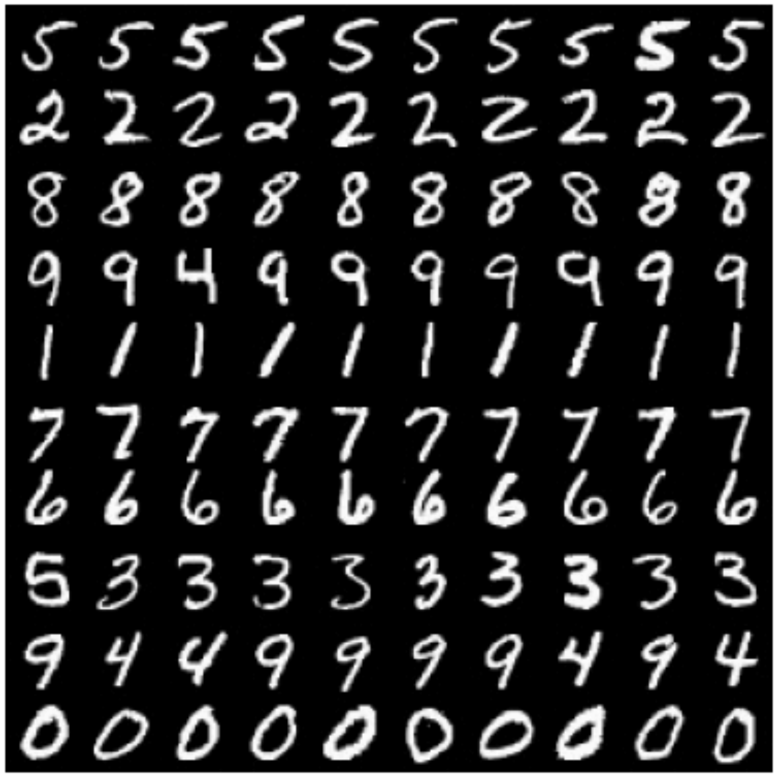


Figure 4. Gradient visualization at the start of KL divergence minimization. This plot shows the magnitude of the gradient of the loss L vs. the cluster soft assignment probability q_{ij}. See text for discussion.


(a) MNIST

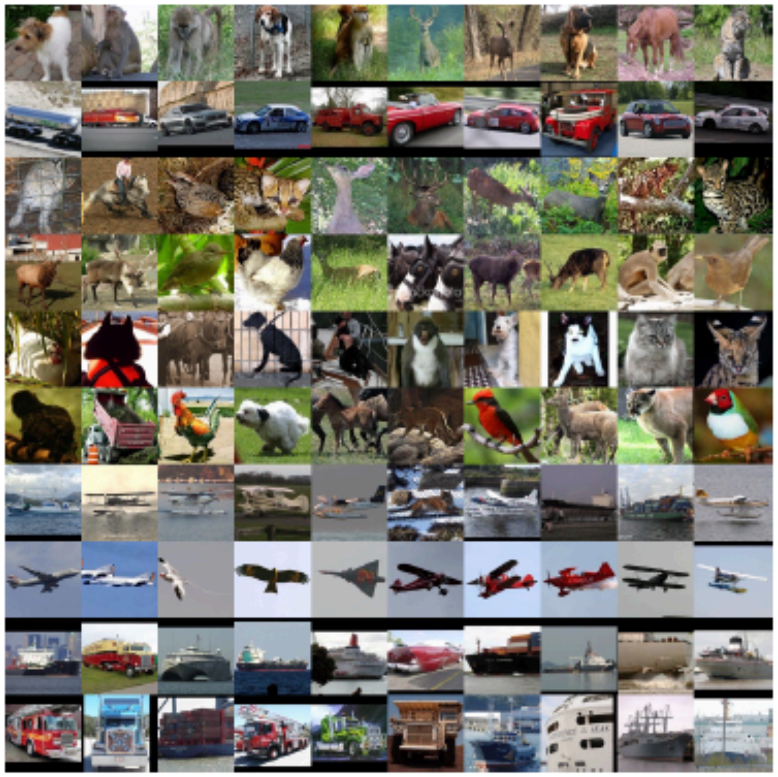

(b) STL-10

Figure 3. Each row contains the top 10 scoring elements from one cluster.

Xie et al 15

Attribute discovery

- We have:
  - a set of images labelled with class, but not attribute
  - a feature construction (now very old fashioned)
- We want:
  - to associate each image with a bit vector
    - attribute present/absent
  - where
    - bits are “easily predicted”
    - bits are “informative”
    - bit vectors within a category cluster

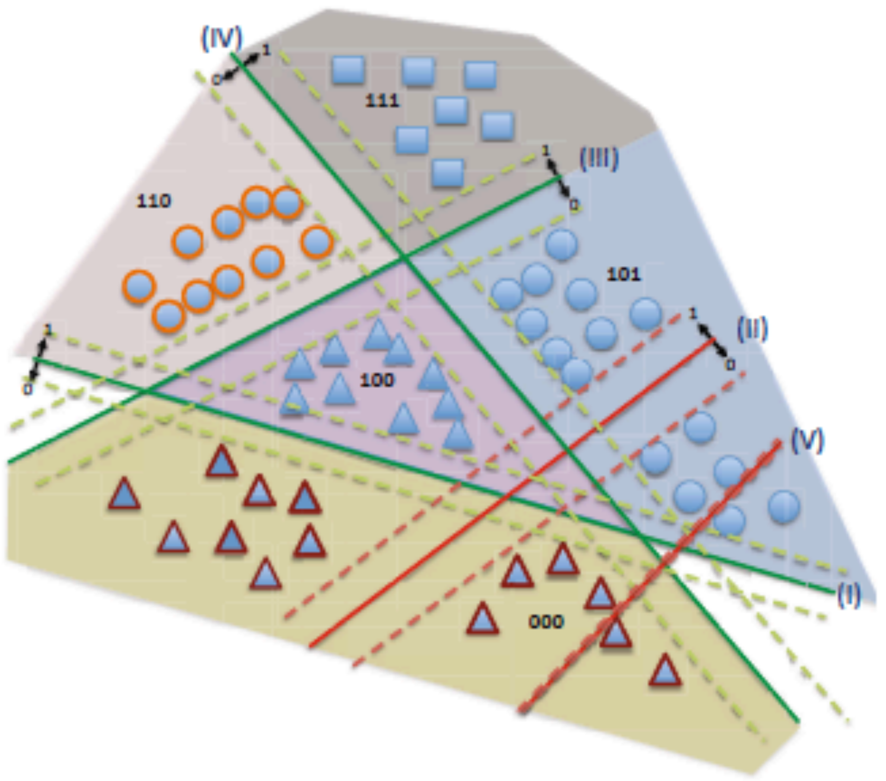


Fig. 1. Each bit in the code can be thought of as a hyperplane in the feature space. We learn arrangements of hyperplanes in a way that the resulting bit codes are discriminative and also all hyperplanes can be reliably predicted (enough margin). For example, the red hyperplanes (II,V) are not desirable because II is not informative(discriminative) and IV is not predictable (no margin). Our method allows the green hypeplanes (good ones) to sacrifice discrimination for predictability and vice versa. For example, our method allows the green hyperplane (I) to go through the triangle class because it has strong evidence that some of the triangles are very similar to circles.

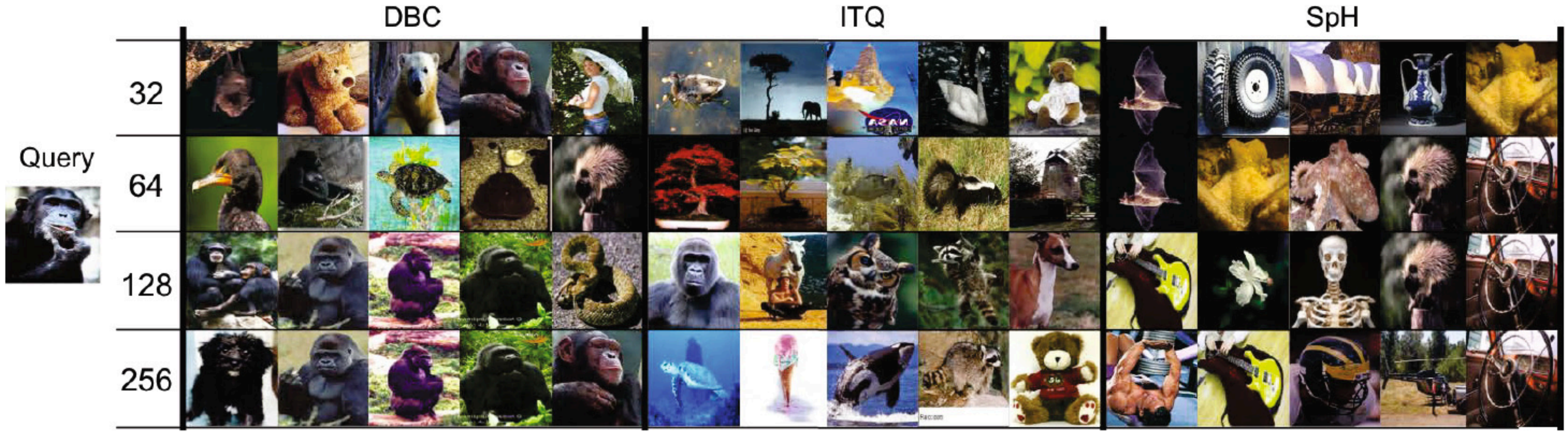


Fig. 6. This figure qualitatively compares the quality of retrieved images by our method comparing to that of ITQ and SpH. Each row corresponds to the top five images returned by three different methods: ours, ITQ and spectral hashing. This retrieval is done by first projecting the query image to the space of binary codes and then running KNN in that space. Notice how, even with relatively short codes(32 bits), our method recovers relevant objects. This menas that the discriminative training of the code has forced our code learning to focus on distinctive shared properties of categories. Our method consistently becomre more accurate as we increase the code size.

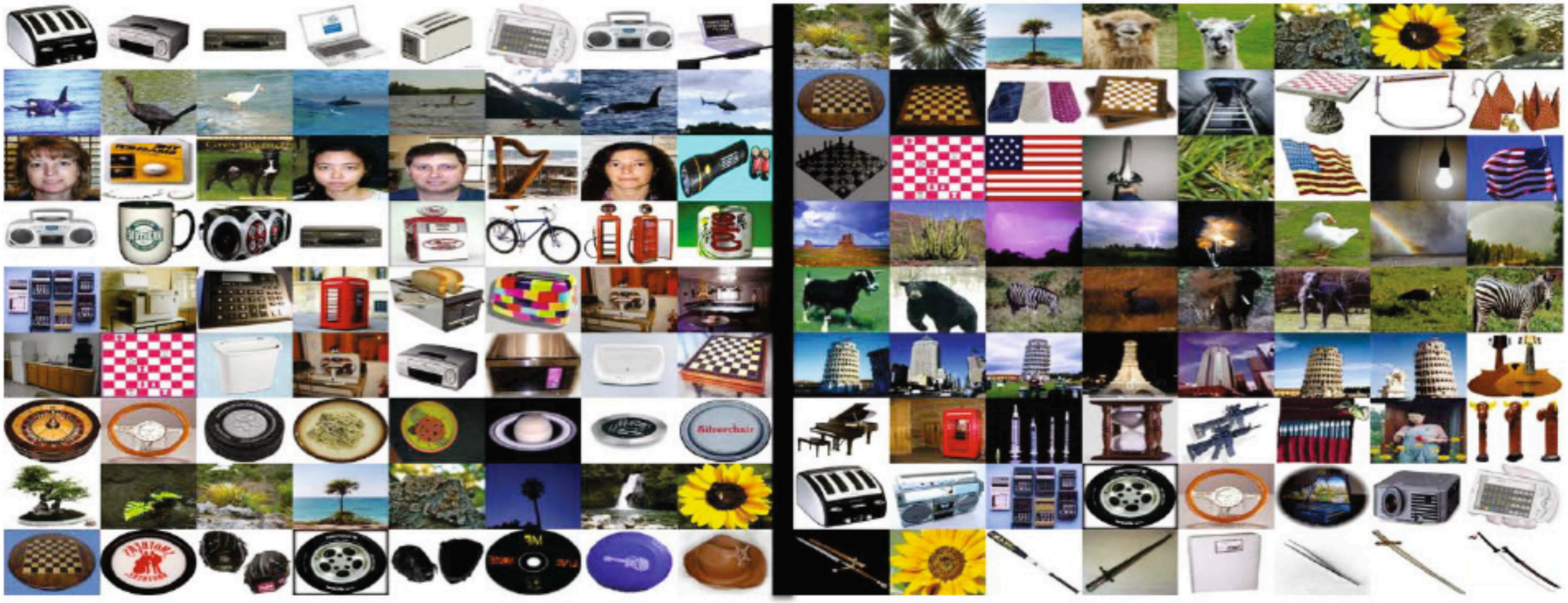


Fig. 8. Discovering attributes: Each bit corresponds to a hyperplane that group the data according to unknown notions of similarity. It is interesting to show what our bits have discovered. On two sides of the black bar we show 8 most confident images for 5 different hyperplanes/bits (Each row). Note that one can easily provide names for these attributes. For example, the bottom row corresponds to all round objects versus objects with straight vertical lines. The top row has silver, metalic and boxy objects on one side and natural images on the other side, the second row has water animals versus objects with checkerboard patterns. Discovered attributes are in the form of contrast: both sides have its own meaning. These attributes are compact representations of standard attributes that only explain one property. For more examples of discovered attributes please see supplementary material.

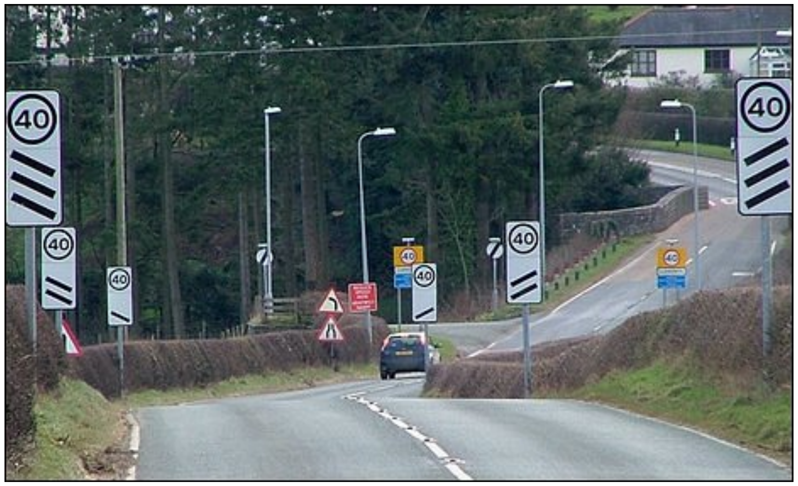
Why do we care?

- Each imputes labels by
    - compelling the label space to have strong properties
        - variant clustering
- DEC suggests that this is enough to learn features
    - DBC has fixed feature stack (but this is discriminative)
- Idea:
    - a feature stack that is discriminative
        - and perhaps has autoencoding properties
    - likely clusters appearance in a useful way
        - so you can impose labels by just compelling them to have spatial structure

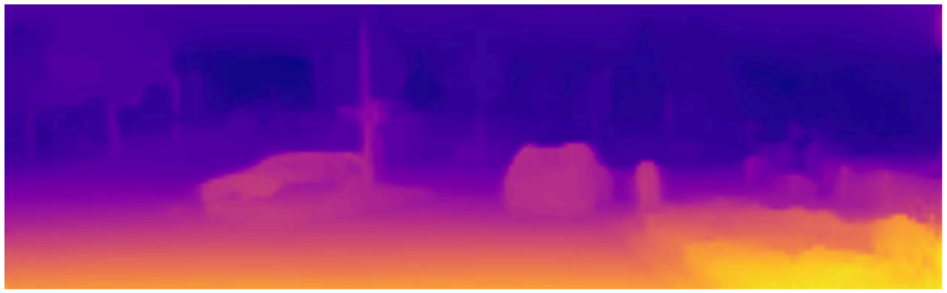
How do we deal with relief?

- Surely some form of height field
  - estimated by consistency
  - changing slowly
- Horizon estimation gets complicated in tilted planes
  - you might get distracted by distant horizon
  - Local horizon estimator has problems

Nasty geometries

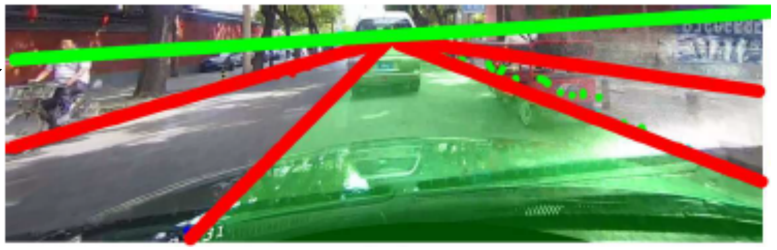


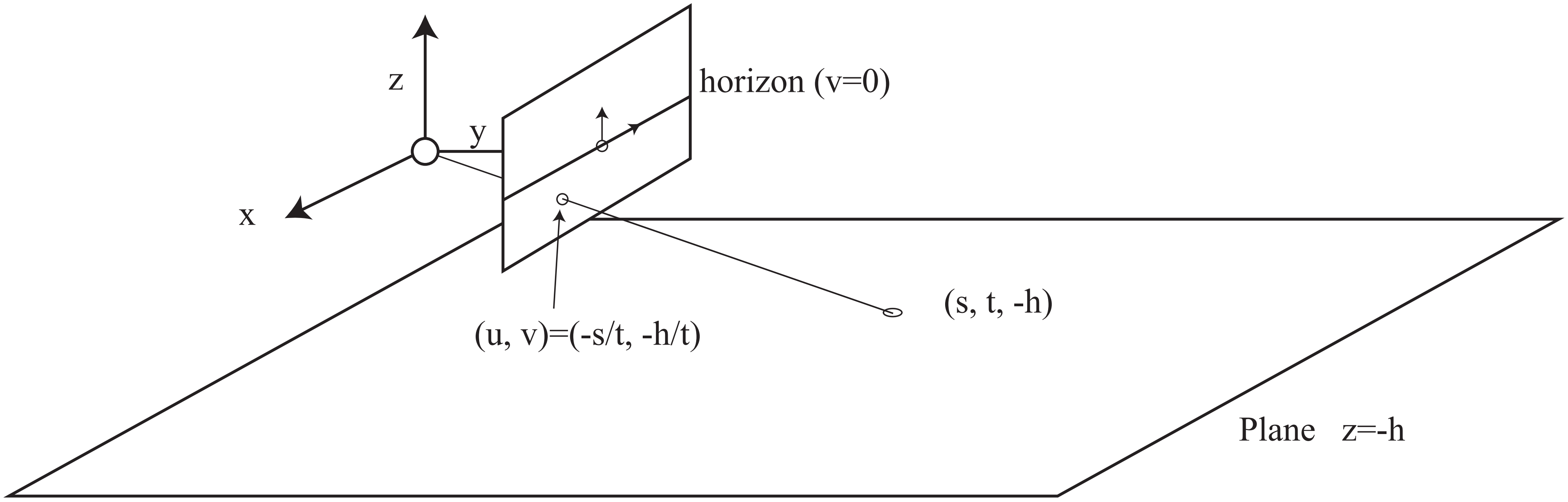
- Single image depth prediction likely doesn't work here
  - weird relief and dip in road
- Ground plane estimates likely don't work here either



Estimating the camera

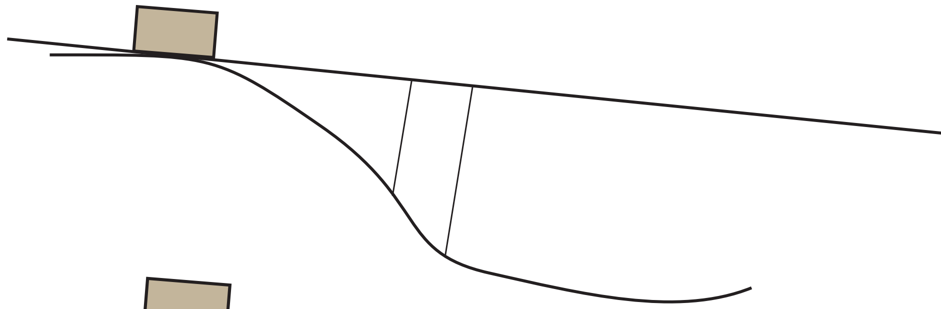
- Height
  - from car (calibrated and known)
- Roll and pitch
  - from horizon
    - roll is why horizon isn't parallel to image plane
    - pitch is why it isn't centerline



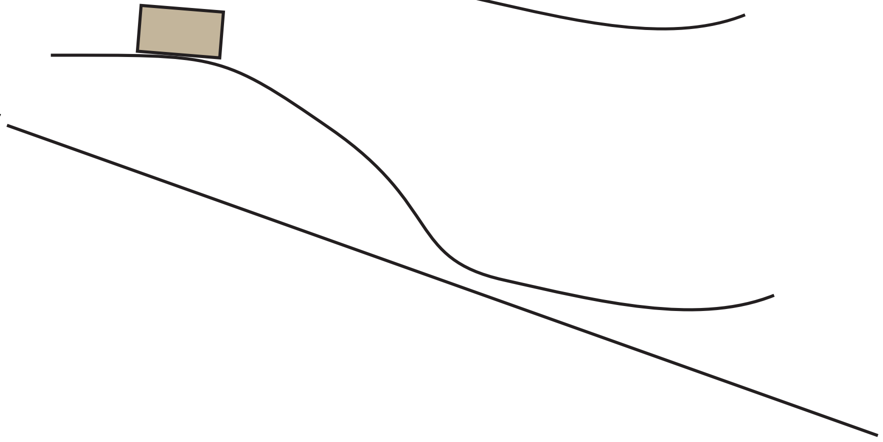


Sources of variation in the label map

- Foreshortening

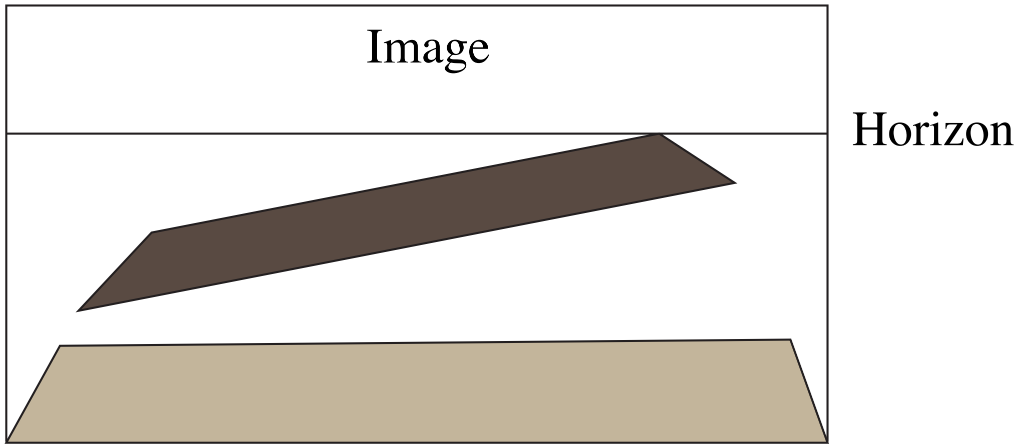


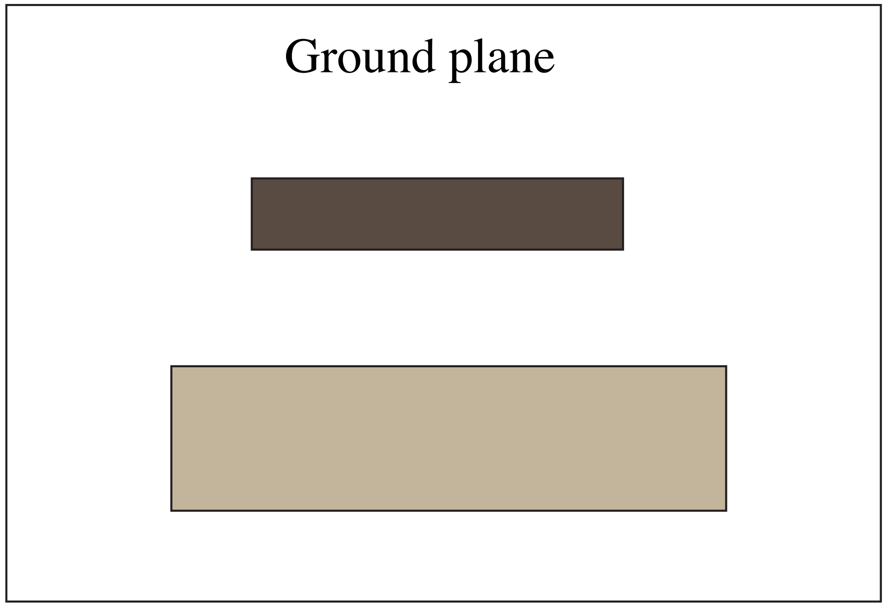
- Wrong ground plane estimate



Sources of variation in the label map

- Torsion





Horizon estimation

- Khan et al - vanishing points from road lines + fudge
- Workman et al - mark up dataset, classify

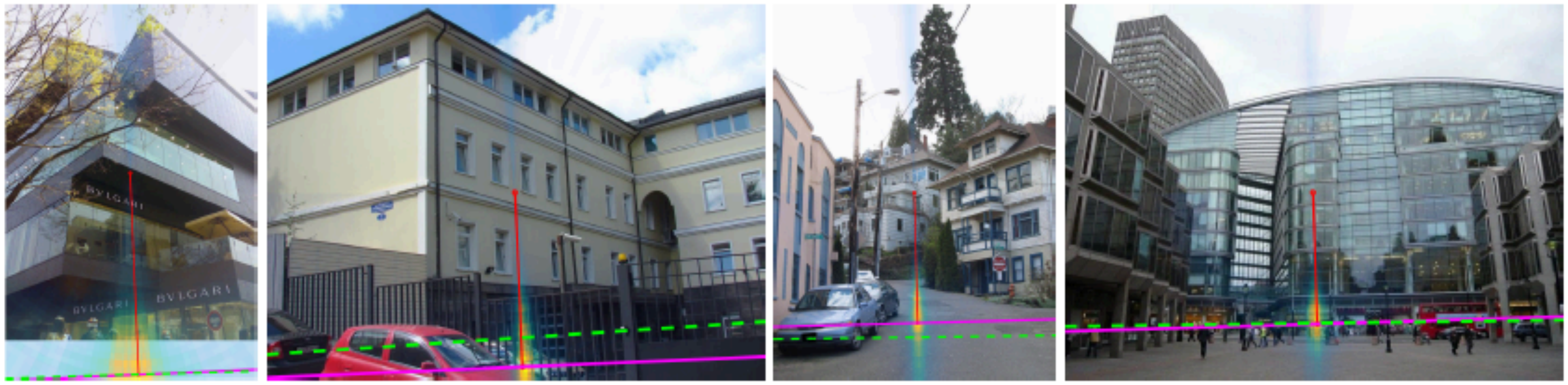
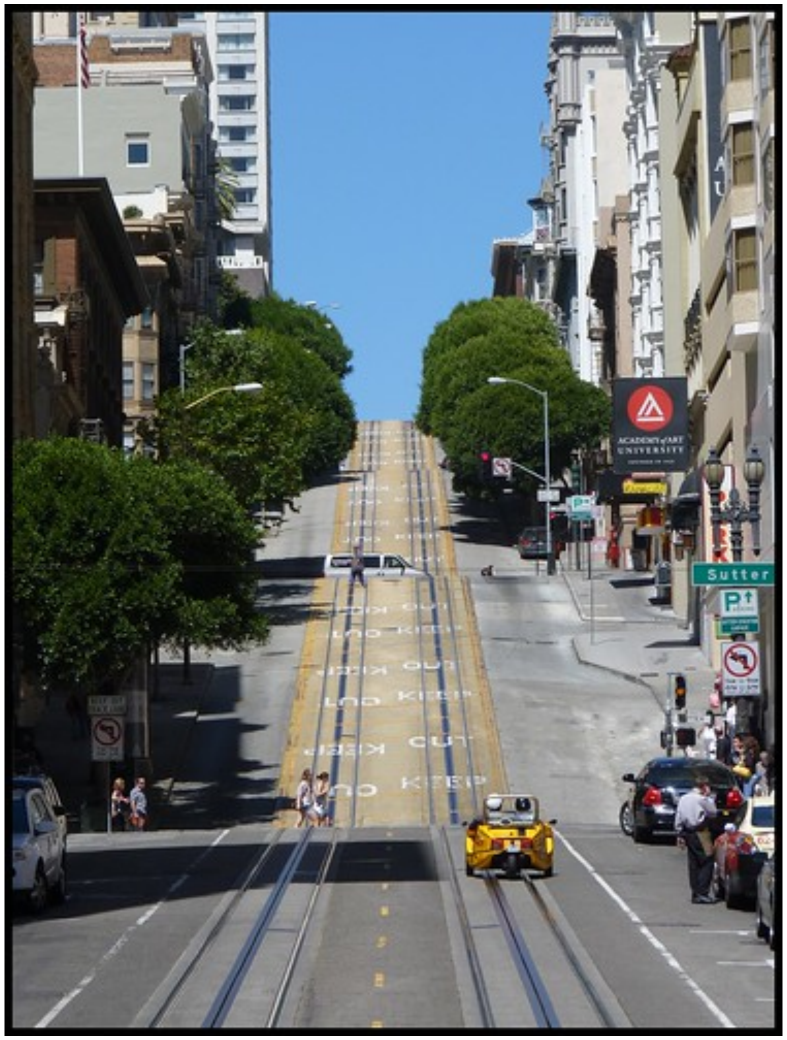


Figure 5: Example results showing the estimated distribution over horizon lines. For each image, the ground truth horizon line (dash green) and the predicted horizon line (magenta) are shown. A false-color overlay (red = more likely, transparent = less likely) shows the estimated distribution over the point on the horizon line closest to the image center.

Horizons

- Horizon estimation gets complicated in tilted planes
    - you might get distracted by distant horizon (picture)



Horizons

- Horizon estimation gets complicated in tilted planes
  - local cues are a problem

What to do?

- (Likely)
  - build sources of variance into simulated label fields
  - work on best available ground plane
    - (possibly) estimate several planes to rectify label fields
  - train without labelled images, as above
    - note this is a clusterer