

Deep Reinforcement Learning and Control

# SIM2Real Transfer

Katerina Fragkiadaki



# Paradox

The requirement of large number of samples for model-free RL, **only possible in simulation**, renders model-free RL a **model-based** framework: we can't do without the simulator.

# Choices

We want to learn manipulation and locomotion policies, what do we do?

1. We use a Physics simulator, where Physics rules between objects and/or particles have been hand coded by engineers. We train our policies there with trial-and-error and/or demonstrations in simulation. We then **transfer** them to the real world.
2. We directly learn policies in the real world. Because we cannot afford many samples, model-based RL usually is a better choice than model-free RL.

# Choices

We want to learn manipulation and locomotion policies, what do we do?

1. We use a Physics simulator, where Physics rules between objects and/or particles have been hand coded by engineers. We train our policies there with trial-and-error and/or demonstrations in simulation. We then transfer them in the real world.
2. We directly learn policies in the real world. Because we cannot afford many samples, model-based RL is usually a better choice than model-free RL.
3. We build better simulators to better model the real world, e.g., food: object deformation, fluids, etc.. Then, GOTO 1.

# This lecture

We want to learn manipulation and locomotion policies, what do we do?

1. We use a Physics simulator, where Physics rules between objects and/or particles have been hand coded by engineers. We train our policies there with trial-and-error and/or demonstrations in simulation. We then transfer them to the real world.
2. We go directly in the real world and learn policies there. Because we cannot afford many samples, model-based control is a better choice than model-free RL.
3. We build better simulators to better model the real world, e.g., food: object deformation, fluids, etc.. Then, GOTO 1.

# Pros of Simulation

- We can afford many samples
- Safe: we do not want to deploy partially trained policies in the real world
- Avoids wear and tear of the robot
- We can explore creative robot configurations

# Cons of Simulation

- Under-modeling: It is hard to exactly replicate the real world and its physics and mechanics
- Large engineering effort into building the environment which we care to manipulate
- Wrong parameters. Even if our physical equations were correct, we would need to estimate the right parameters, e.g., inertia, frictions (system identification).
- Systematic discrepancy w.r.t. the real world regarding:
  1. observations
  2. dynamics

Result: Policies learnt in simulation usually do not directly transfer to the real world

# Simulators

Mujoco: rigid body simulator on a CPU

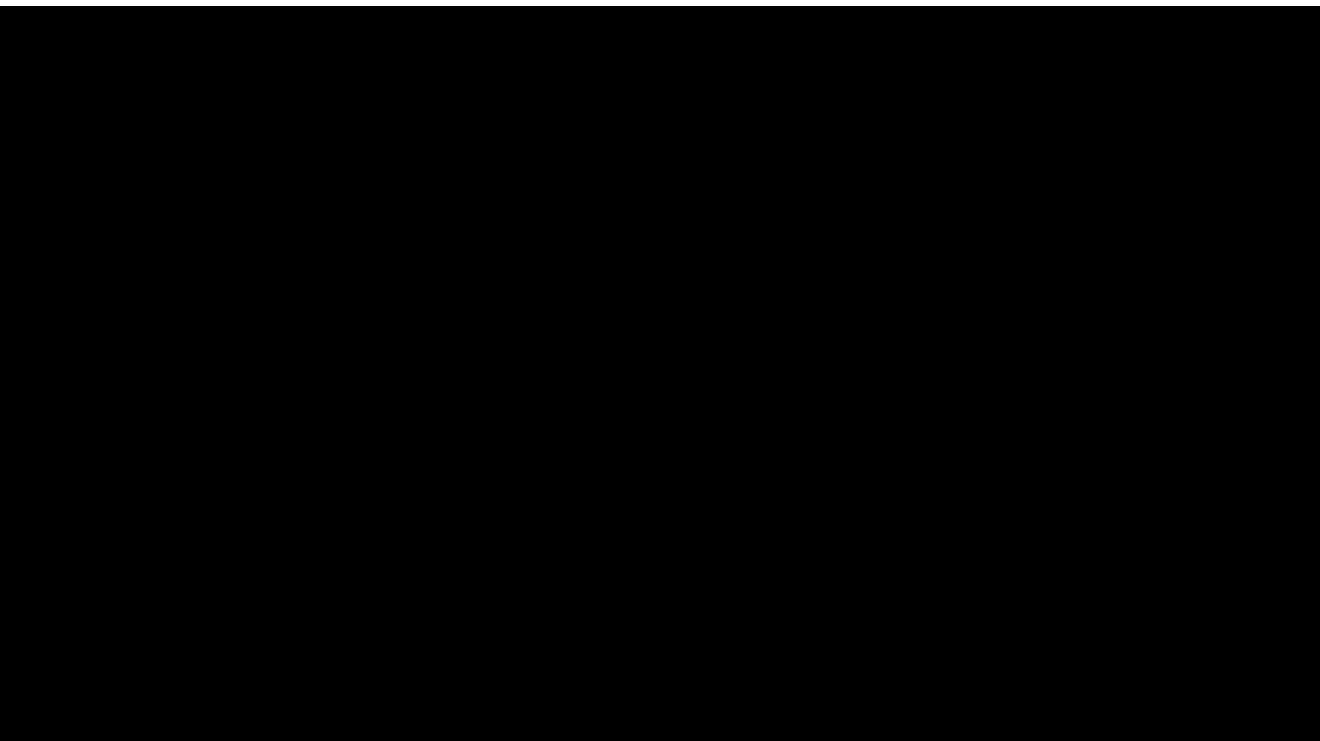
**MuJoCo physics**

**Roboti LLC**

**[www.mujoco.org](http://www.mujoco.org)**

<http://www.mujoco.org/image/home/mujocodemo.mp4>

FLEX: particle based simulator on a GPU for rigid / soft bodies, fluids, gas.



<https://www.youtube.com/watch?v=1o0Nuq71gl4>

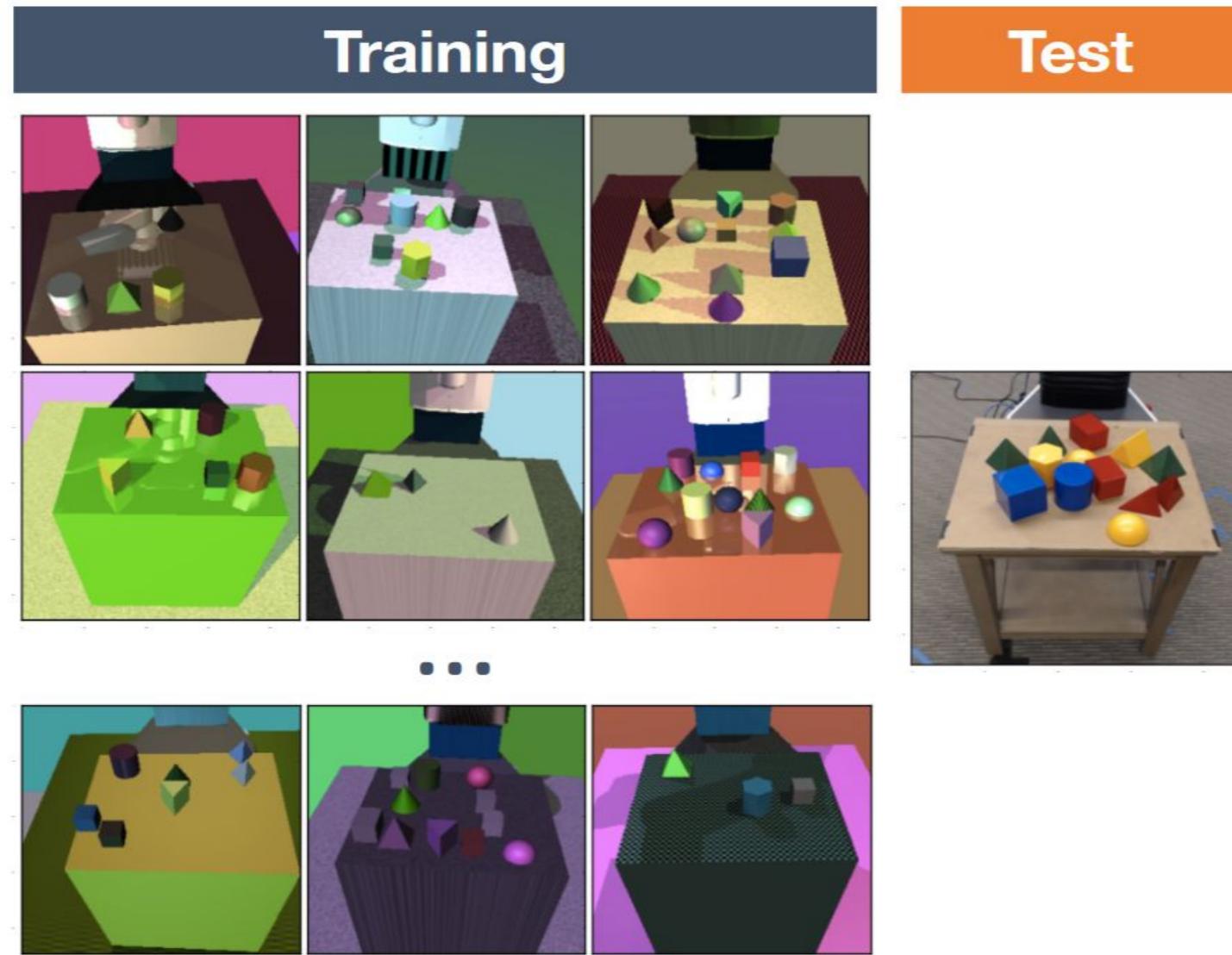
# Sim2Real:What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the simulator to match the real domain
- Learning from label images as opposed to pixel images-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

# What has shown to work

- Domain randomization (**dynamics, visuals**)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the simulator to match the real domain
- Learning from label images as opposed to pixel images-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

# Domain randomization

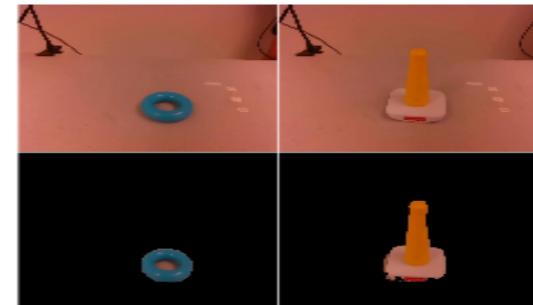


We create (automatically) tons of simulation environments by randomizing textures and camera viewpoints. We use the simulation data to train object detectors

# Data dreaming

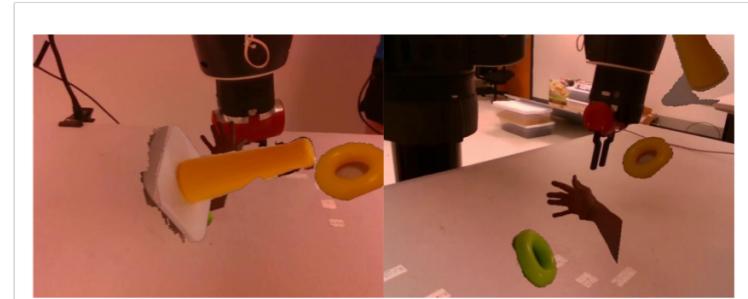
## 1. Obtaining object masks

- background subtraction gives ground truth object masks



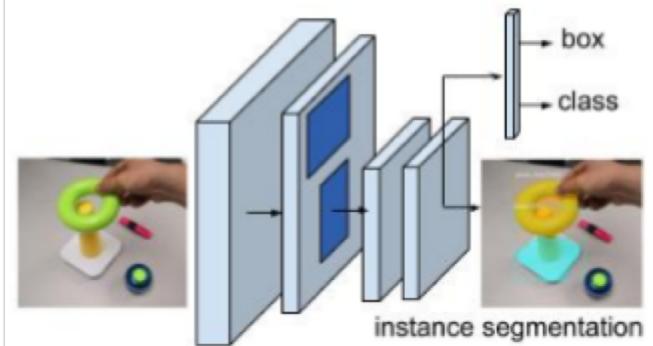
## 2. Creating synthetic labelled data

- Massive augmentation of ground truth masks by random transformations/occlusions and random backgrounds



## 3. Training object detectors

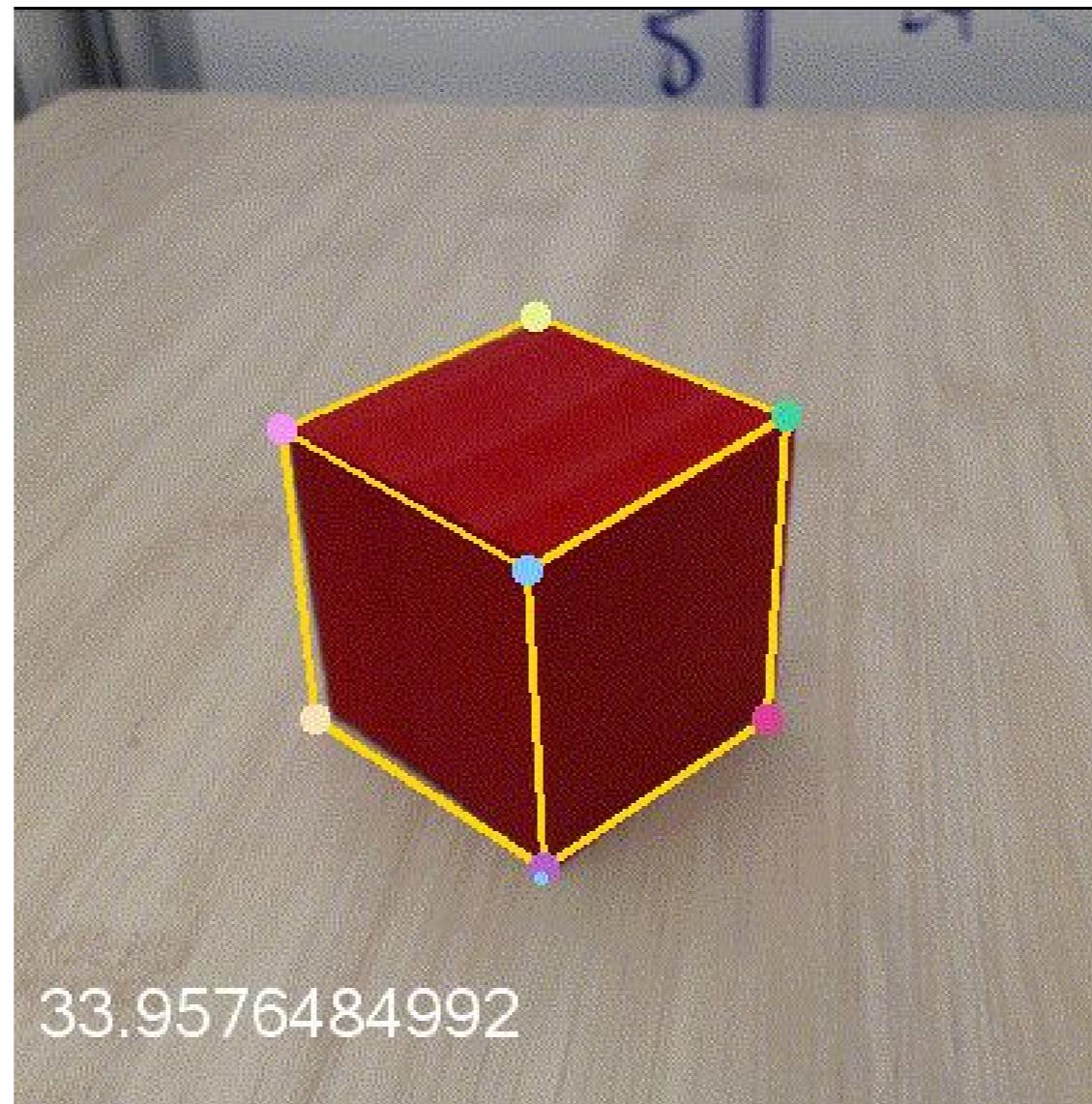
- Mask R-CNN



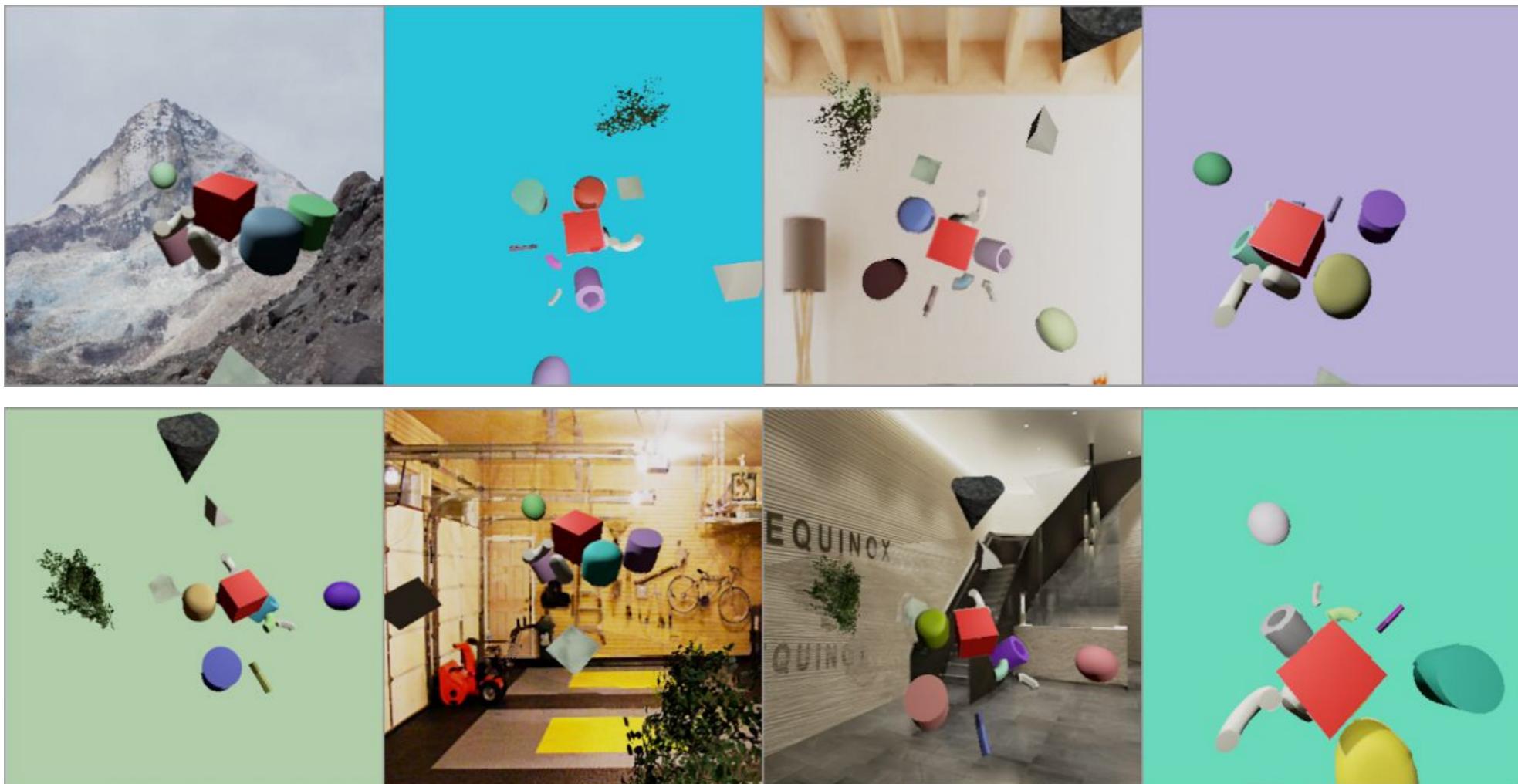
Similar randomization can be used for training object detectors **on-the-fly in the real world** directly.

# Let's try a more fine grained task

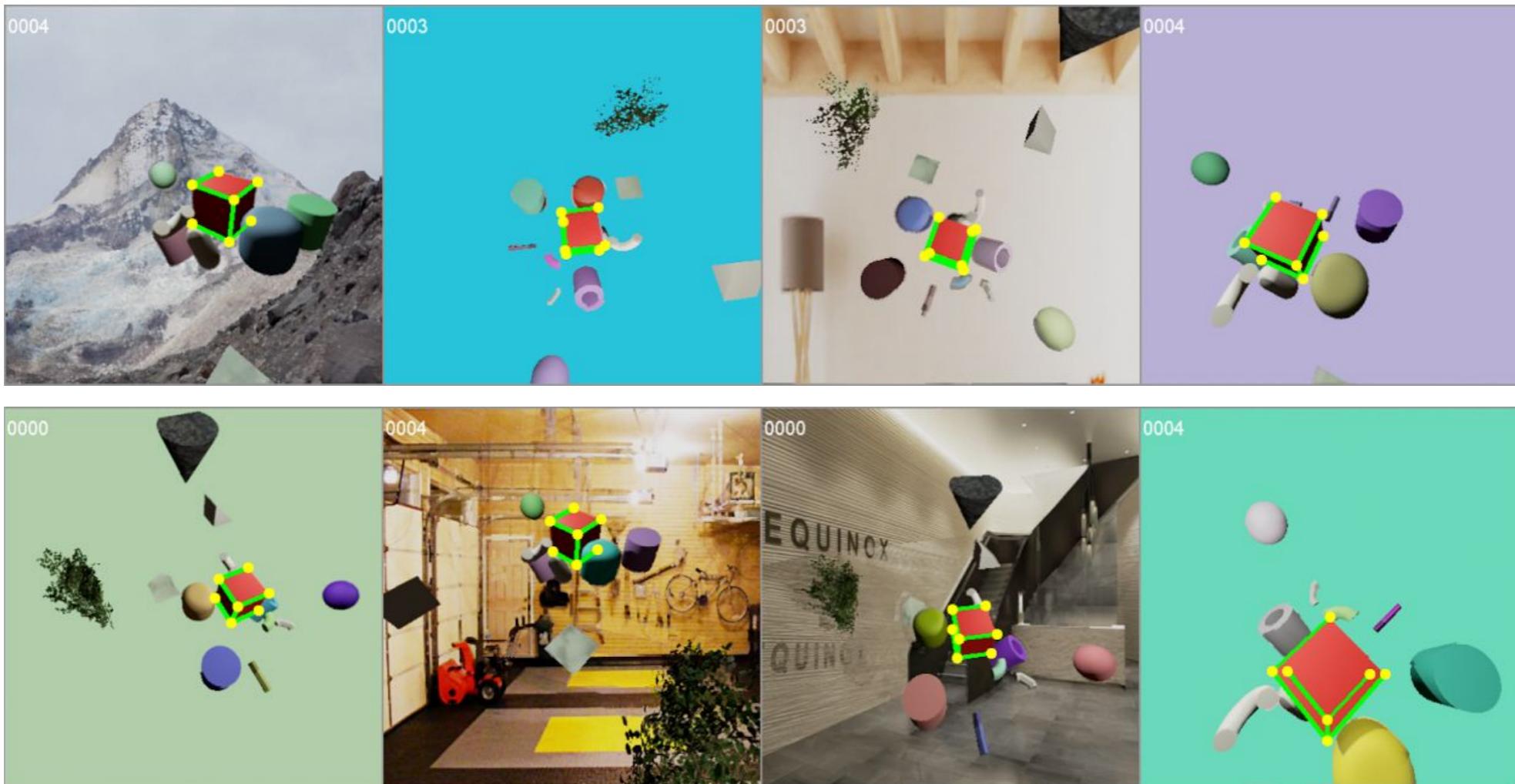
## Cuboid Pose Estimation



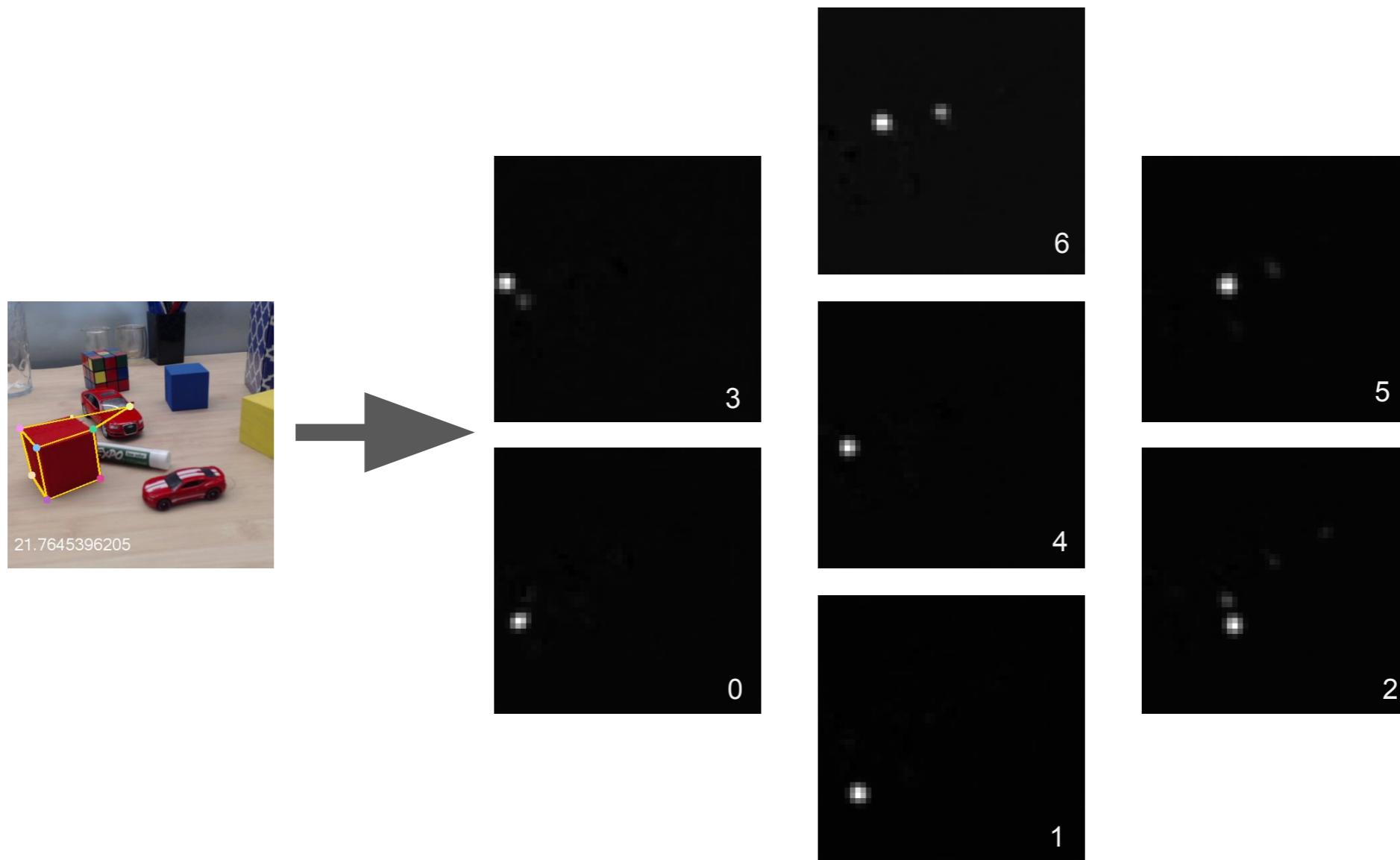
# Synthetic data generation



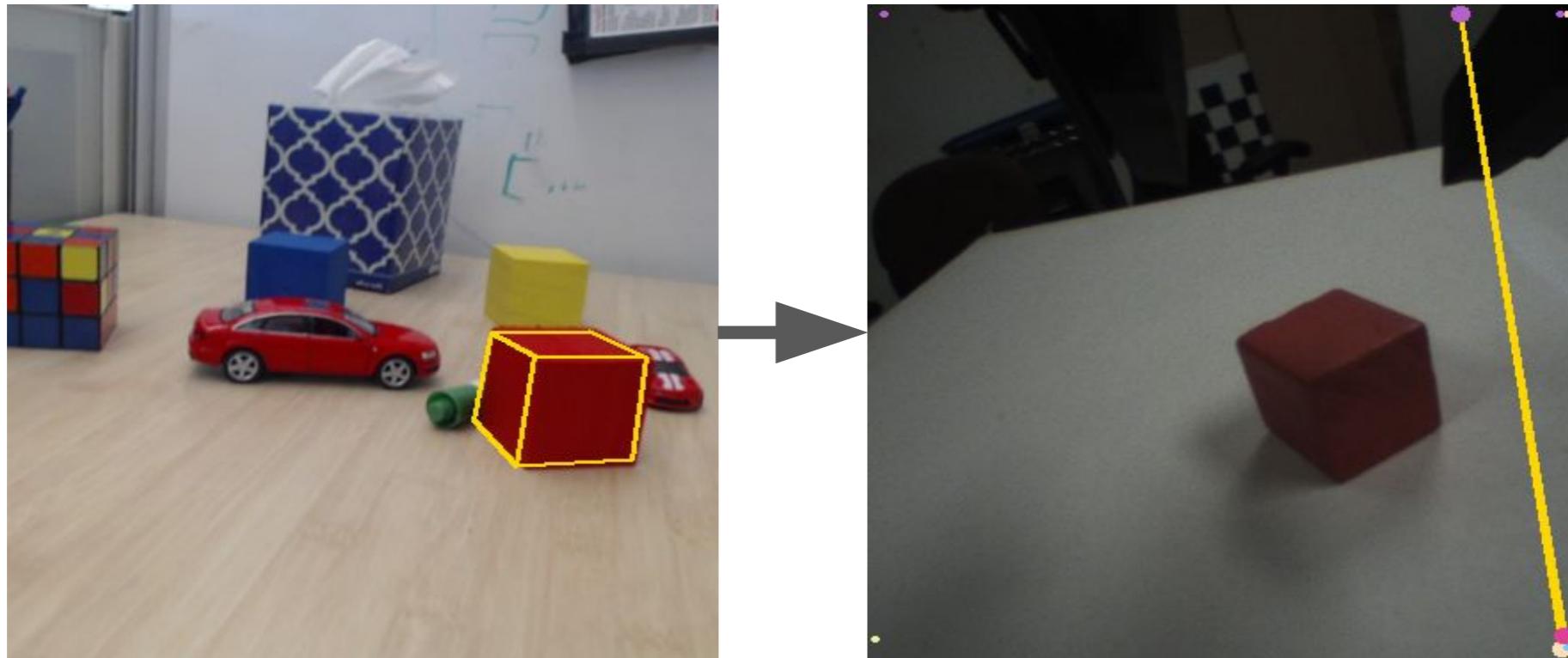
# Synthetic data generation



# Predicting vertex heatmaps



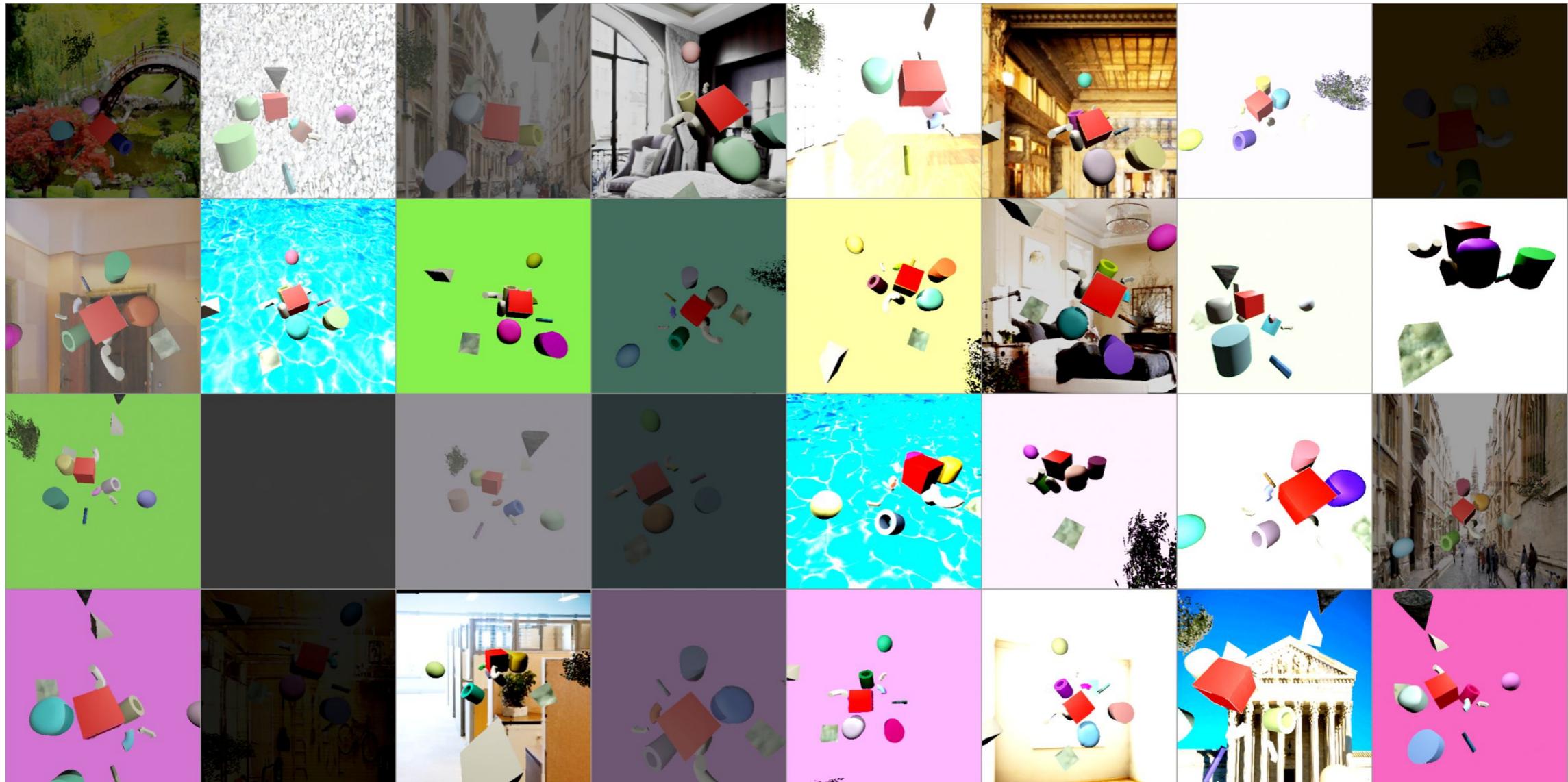
# SIM2REAL



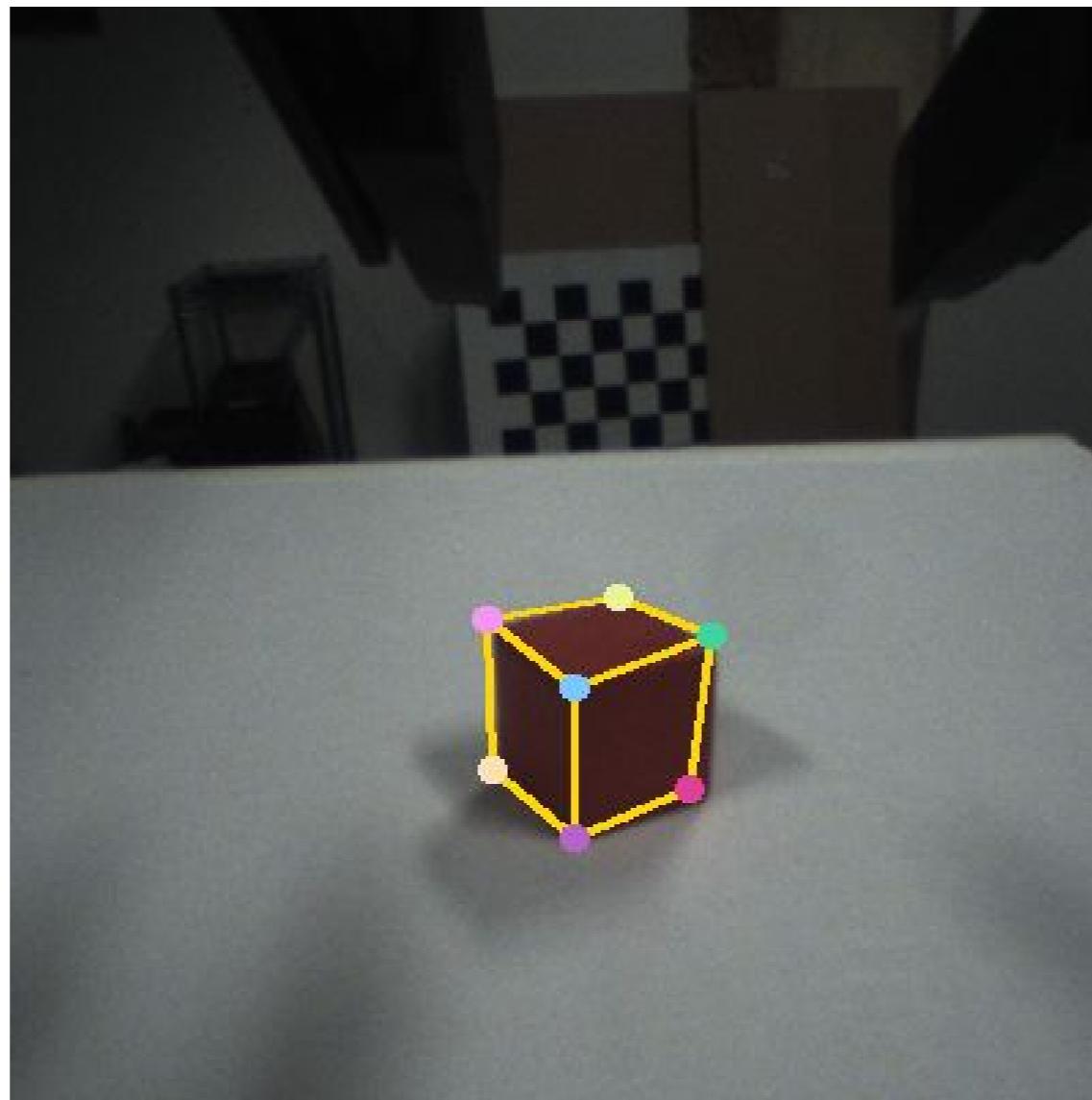
- Pose detector fails when the brightness of the image changes. Solution?
- Randomize also the brightness

# Synthetic data generation

## Data - Contrast and Brightness



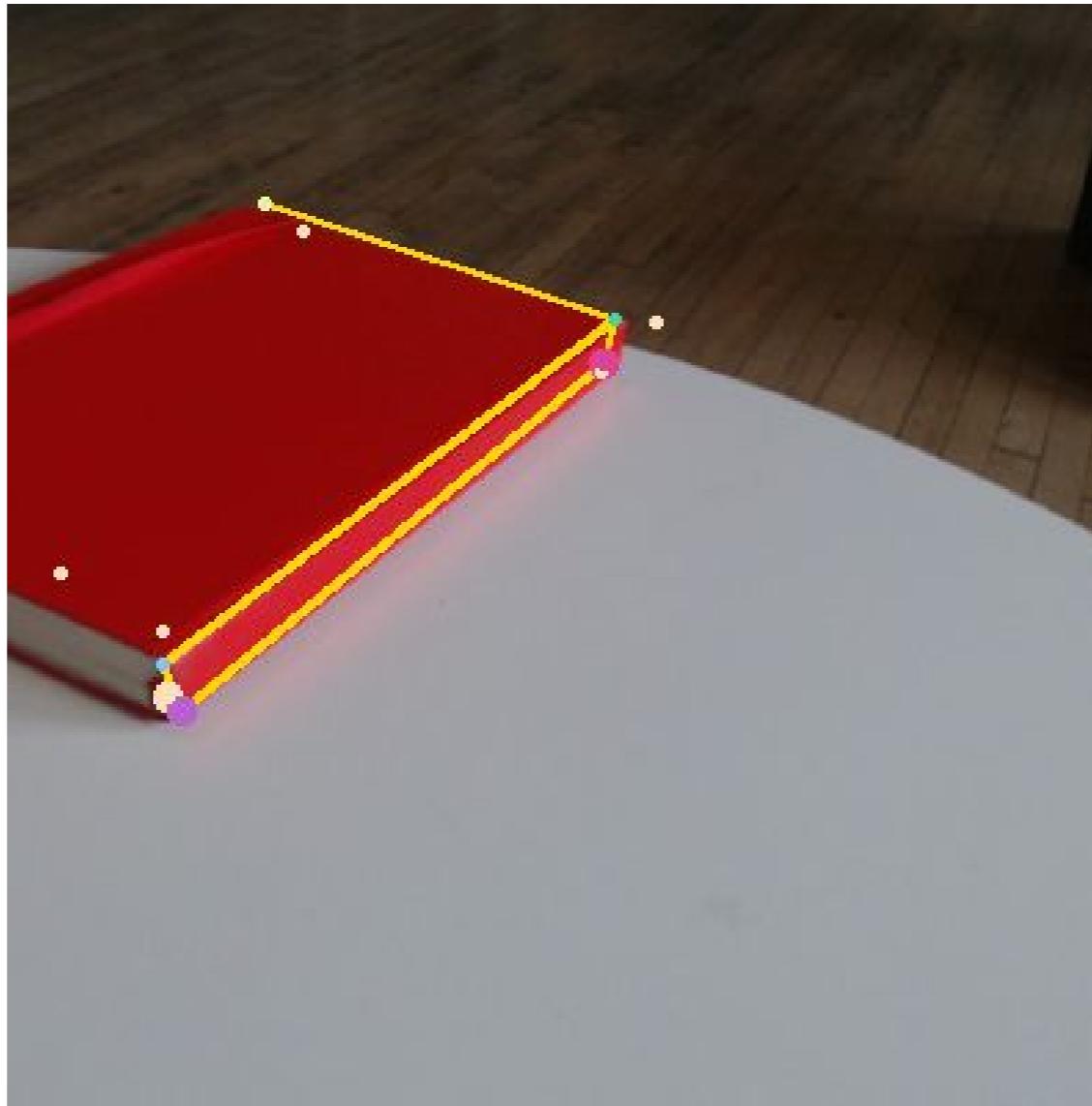
# SIM2REAL



- Now it works..

# SIM2REAL

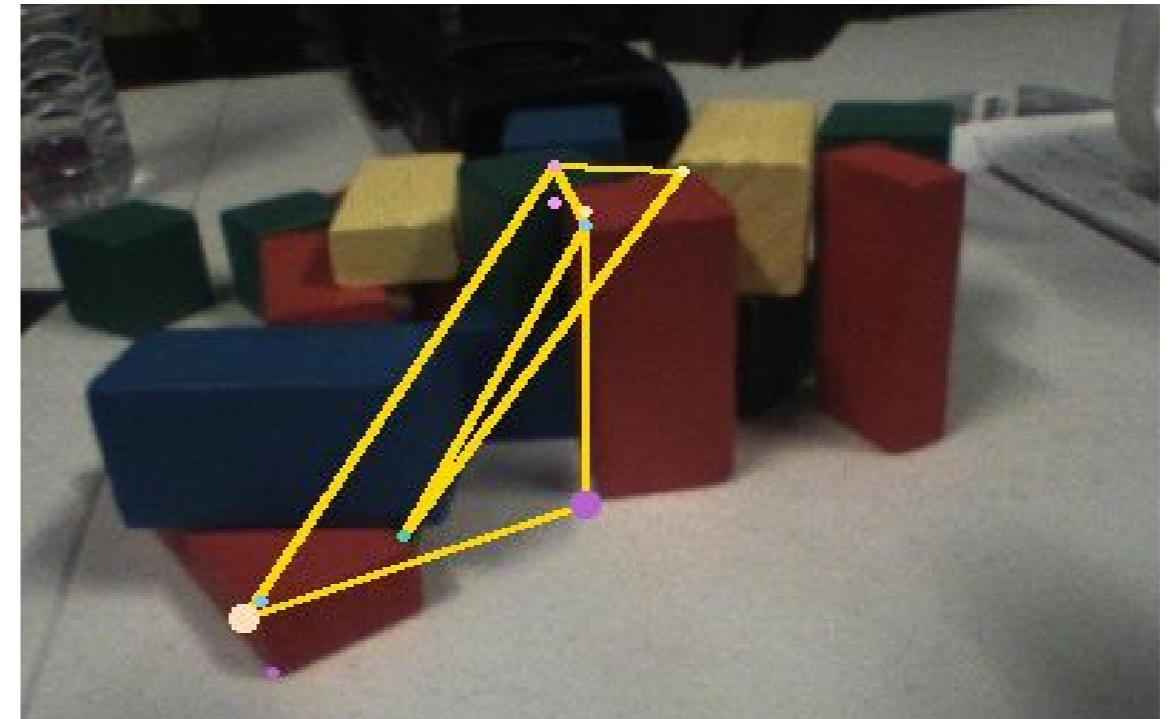
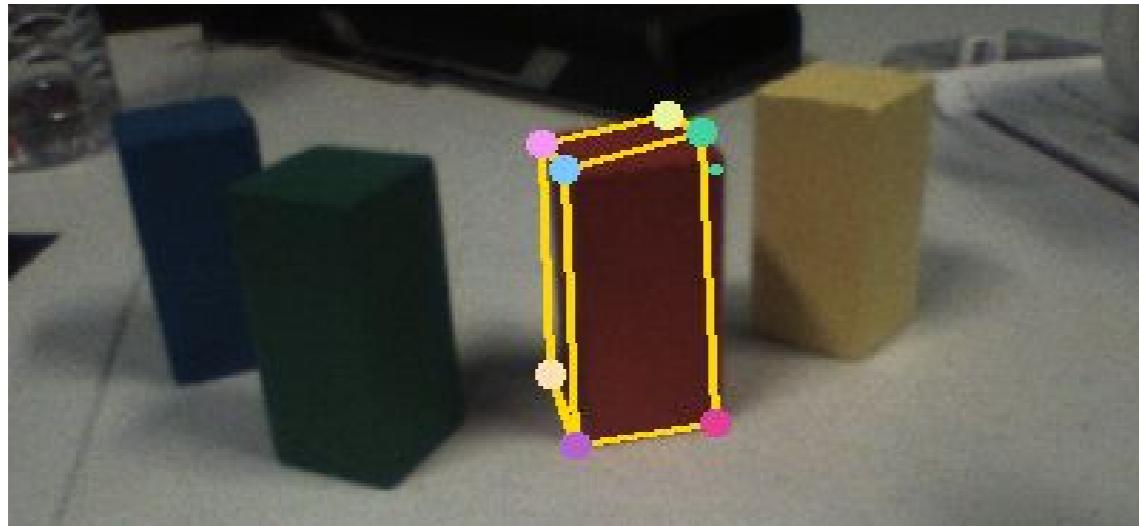
## Surprising Result



- Even for non cube objects sometimes

# SIM2REAL

## Baxter's camera



- It can fail under clutter.
- Solution: use an architecture from computer vision research: combine object detection with vertex heatmap prediction, do not vertex heatmaps with the whole image as input

# Car detection

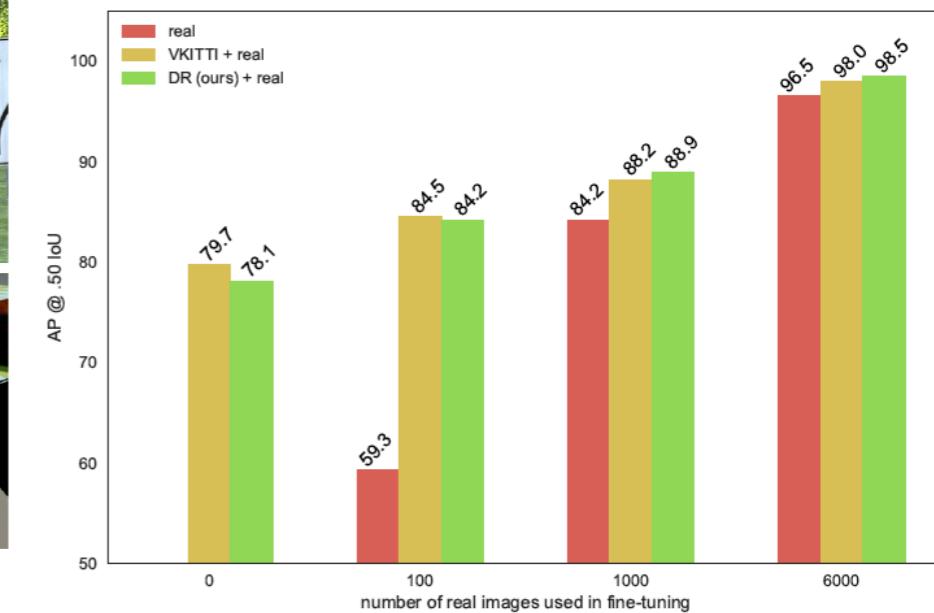
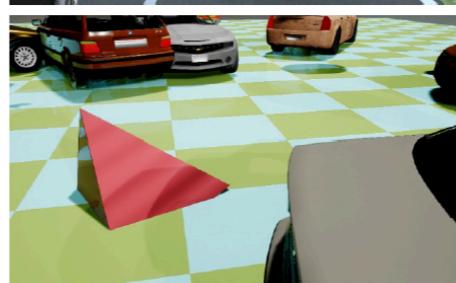
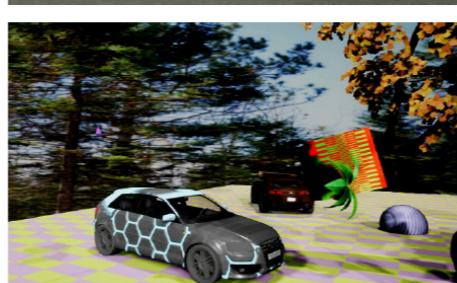
VKITTI (Virtual KITTI): a carefully designed simulation dataset to mimic real driving conditions (large engineering effort)

DR: an automatically created simulation dataset with non-realistic visuals and content (small engineering effort)

VKITTI



DR



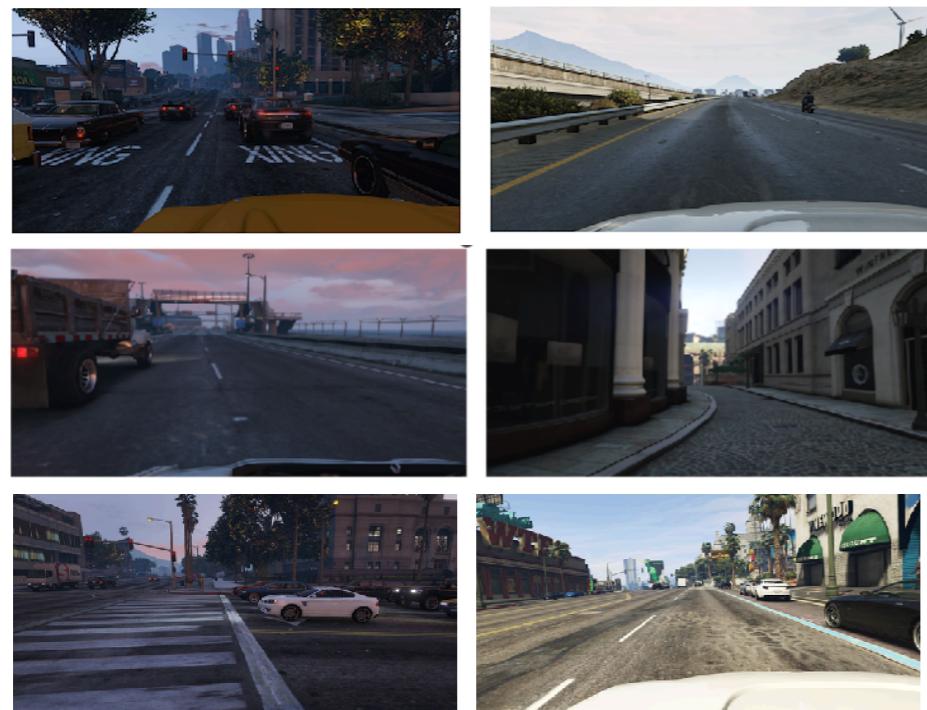
The fewer the real labelled data, the larger the gain from synthetic data

# What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the dymulator to match the real domain
- Learning from label (as opposed to pixel) maps-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

# Domain adaptation for visual observations

**GTA:** synthetic data of urban scenes from a camera mounted on a car



19 object classes to be detected: people, cars, stop signs, poles, etc.

**source**

**Cityscapes:** real data of urban scenes from a camera mounted on a car



**target**

**Our goal:** Train detectors and pixel labelers on GTA that generalize to Cityscapes

# Baseline

**Train a classifier on source and test it on the target, and hope it generalizes**

1. Pick a network architecture, e.g. ResNet101 or VGG
2. Download a **pretrained** neural network, e.g., trained for image classification on Imagenet dataset
3. **Finetune** it on the source domain (GTA)
4. Apply it on the target domain (Cityscapes)

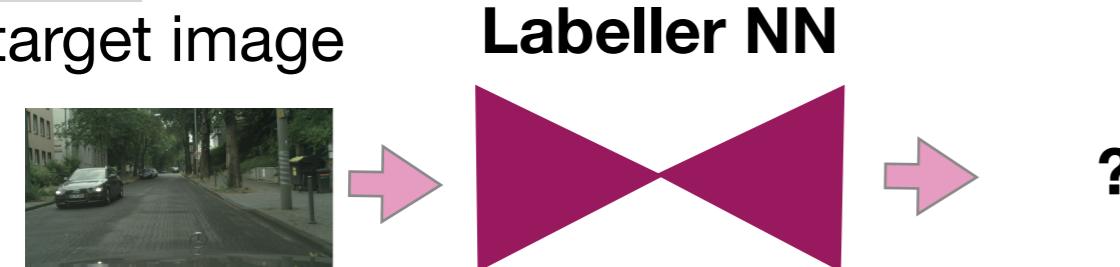
## Training



Image classification in Imagenet



## Testing



# Baseline

**Train a classifier on source and test it on the target, and hope it generalizes**

1. Download a **pretrained** neural network, e.g., trained for image classification on Imagenet dataset
2. **Finetune** it on the source domain (GTA)
3. Apply it on the target domain (Cityscapes)

Pretrain on Imagenet -> finetune in GTA->test in GTA: 53% meanIoU

Pretrain on Imagenet -> finetune in GTA->test in Cityscapes: 28% meanIoU

Pretrain on PASCAL -> finetune in GTA->test in GTA: 58.84% meanIoU

Pretrain on PASCAL -> finetune in GTA->test in Cityscapes: **32%** meanIoU

Pretrain on PASCAL -> **cotrain** in GTA/PASCAL->test in Cityscapes: **39%** meanIoU

# Baseline

Train a classifier on source and test it on the target, and hope it generalizes

1. Download a **pertained** neural network, e.g., trained for image classification on Imagenet, or trained for pixel labelling in PASCAL
2. **Finetune** it on the source domain (GTA)
- 3.

## Catastrophic forgetting:

- During fine-tuning, the network forgets the general and nicely transferable PASCAL features!
- **Finetuning** a neural net on a very **limited** domain is a

Other solutions for catastrophic forgetting?

# Baseline

**Train a classifier on source and test it on the target, and hope it generalizes**

1. Download a **pretrained** neural network, e.g., trained for image classification on Imagenet dataset
2. **Finetune it on the source domain (GTA):**
  1. Adapt only the top layers and keep the earlier frozen.
  2. Cotrain it using both the old task and the new task in the smaller dataset
3. Apply it on the target domain (Cityscapes)

Pretrain on Imagenet -> finetune in GTA->test in GTA: 53% meanIoU

Pretrain on Imagenet -> finetune in GTA->test in Cityscapes: 28% meanIoU

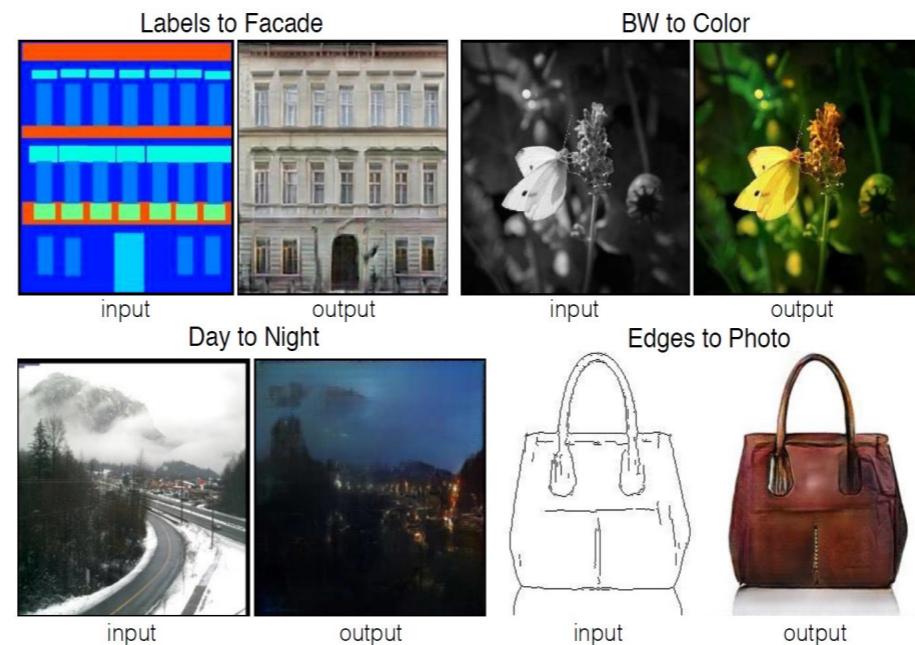
Pretrain on PASCAL -> finetune in GTA->test in GTA: 58.84% meanIoU

Pretrain on PASCAL -> finetune in GTA->test in Cityscapes: 32% meanIoU

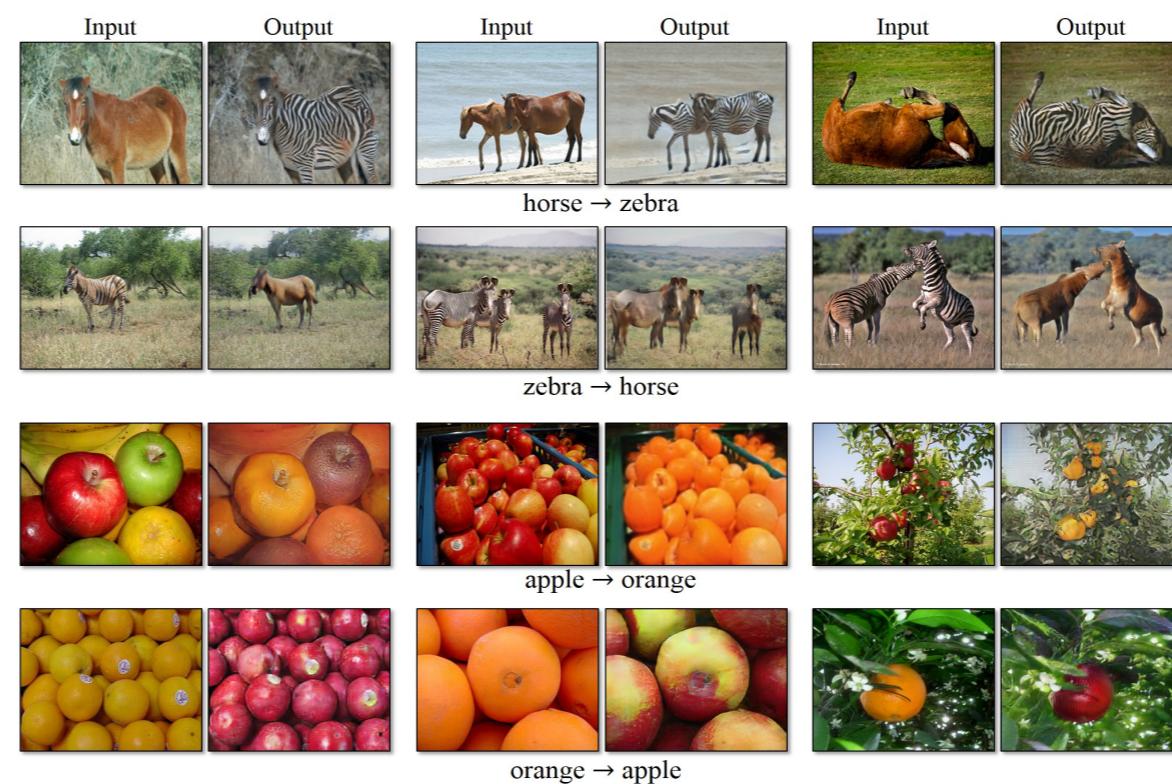
Pretrain on PASCAL -> **cotrain** in GTA/PASCAL->test in Cityscapes: 39% meanIoU

# Learning to translate images across domains

- Paired

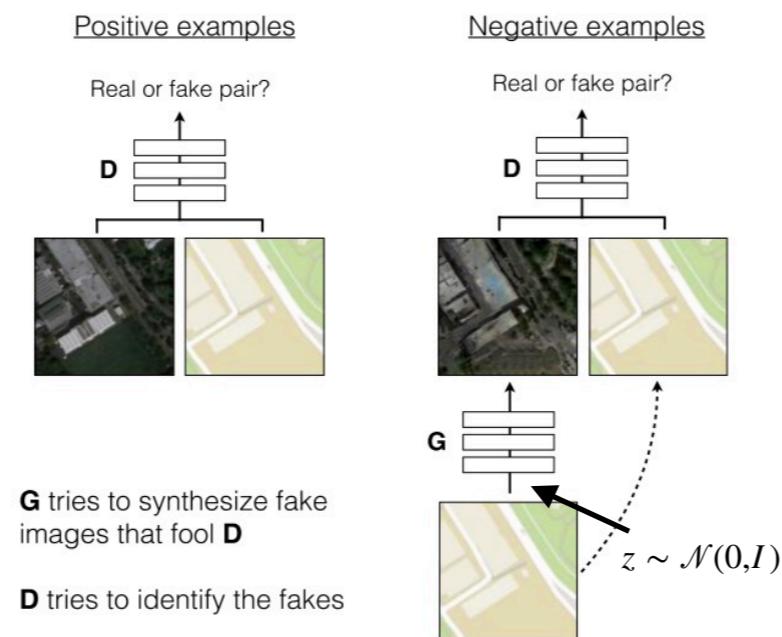


- Unpaired (this is our case)

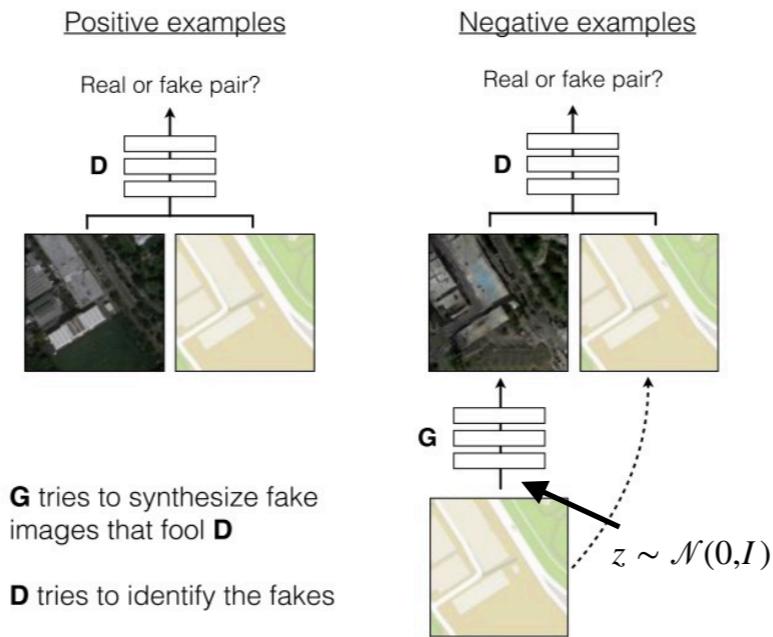


# Paired case

- The generator takes the (source) image as input and tries to output the corresponding target image
- Minimize
  - image reconstruction loss
  - adversarial loss: pairs of source-target images as input to discriminator



# Paired case

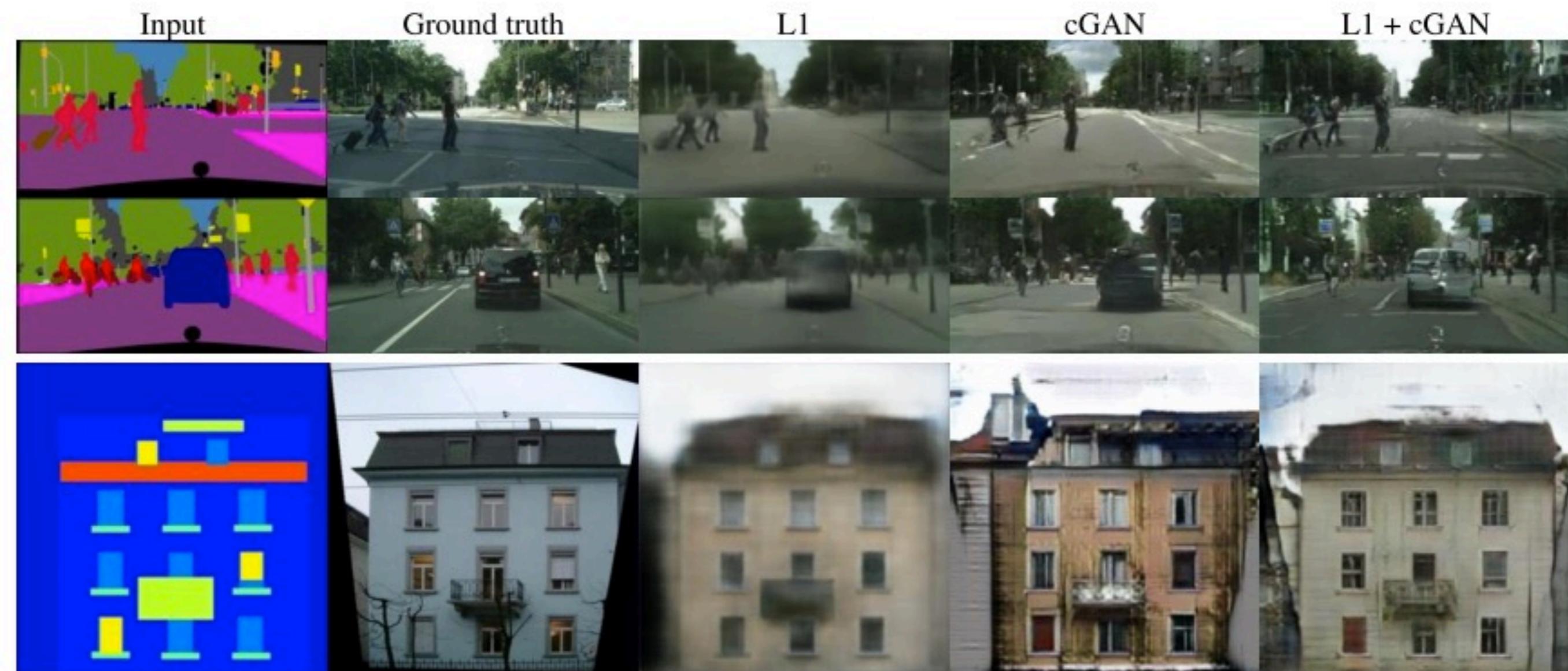


$x$ : source image,  $y$ : target image,  $z$ : noise

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y \sim p_{data}(x,y)} [\log D(x, y)] + \mathbb{E}_{x \sim p_{data}(x), z \sim p_z(z)} [\log(1 - D(x, G(x, z)))]$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y \sim p_{data}(x,y), z \sim p_z(z)} [||y - G(x, z)||_1]$$

# Paired case



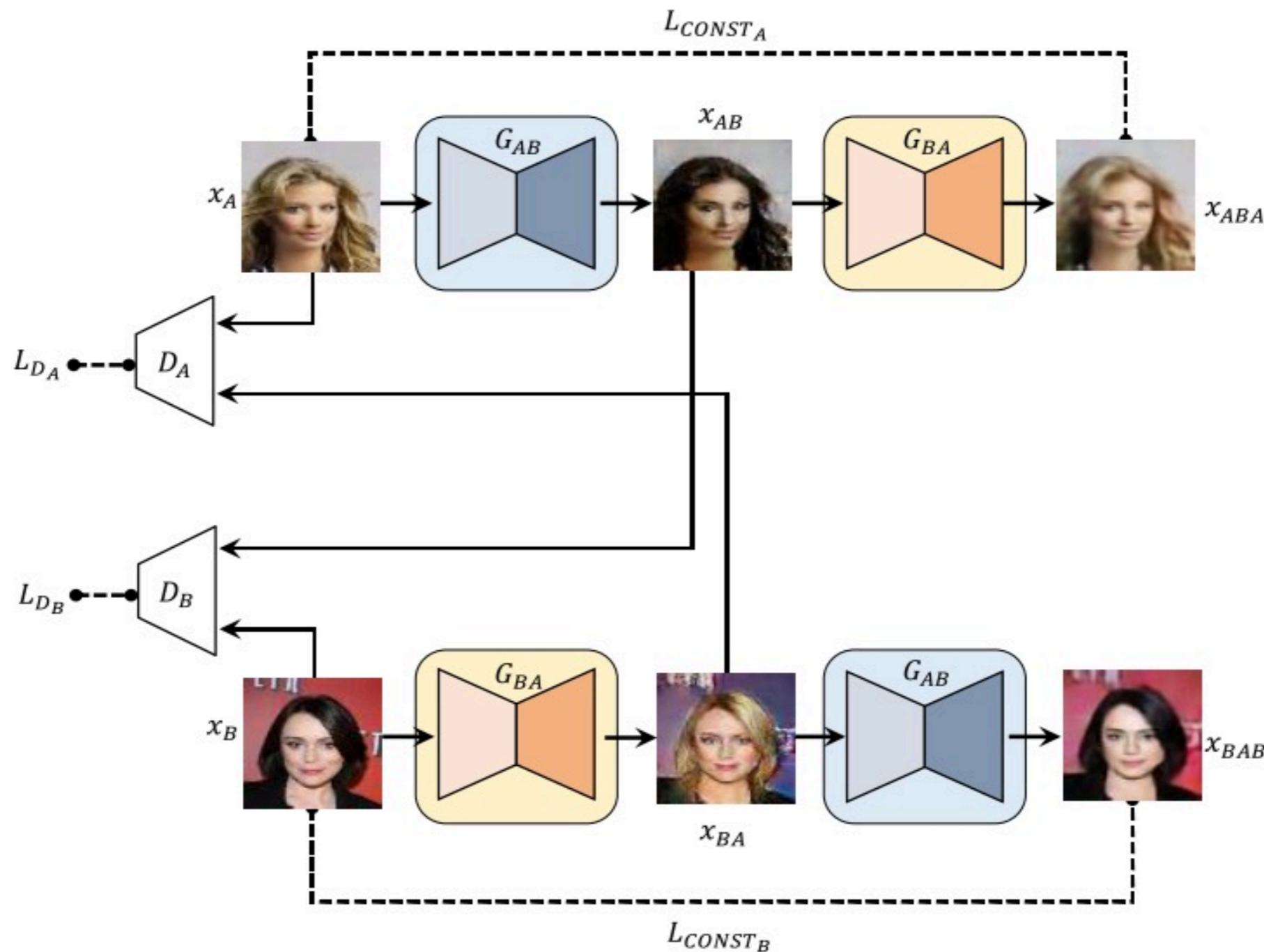
# Paired case



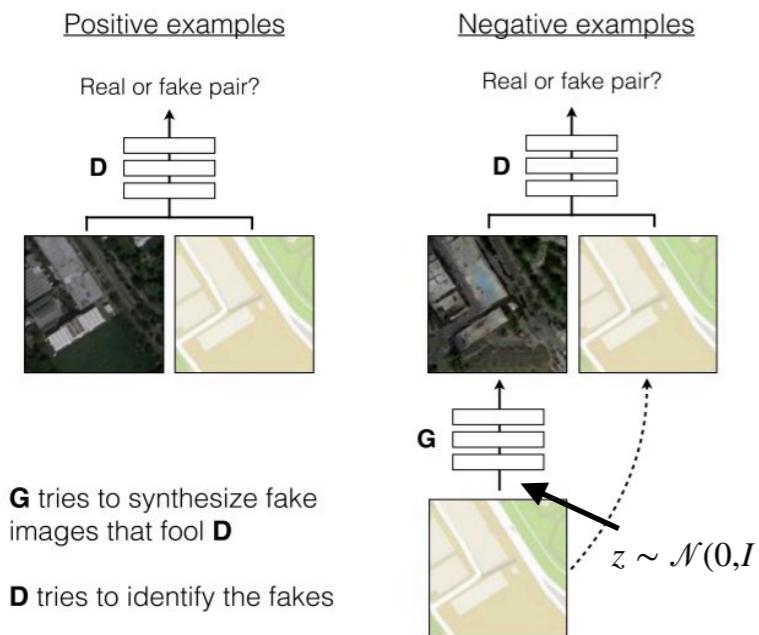
# Paired case



# Unpaired case



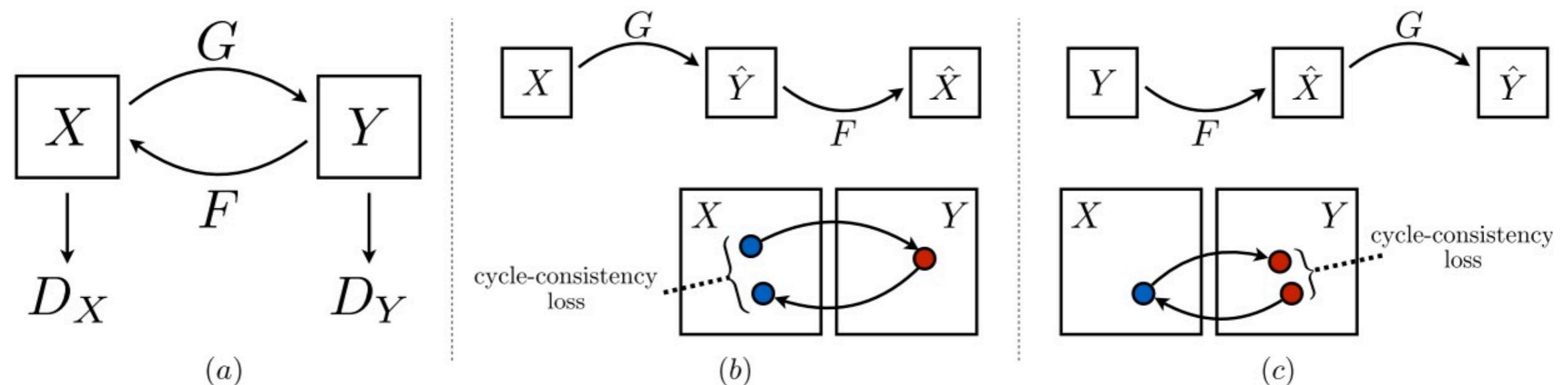
# Paired case



This works despite adding noise experiences mode collapse and cannot output diverse transferred images. For handling this in the paired case check: <https://junyanz.github.io/BicycleGAN/>



# Unpaired case: Cycle GAN / DISCO GAN



# Cycle GAN / DISCO GAN

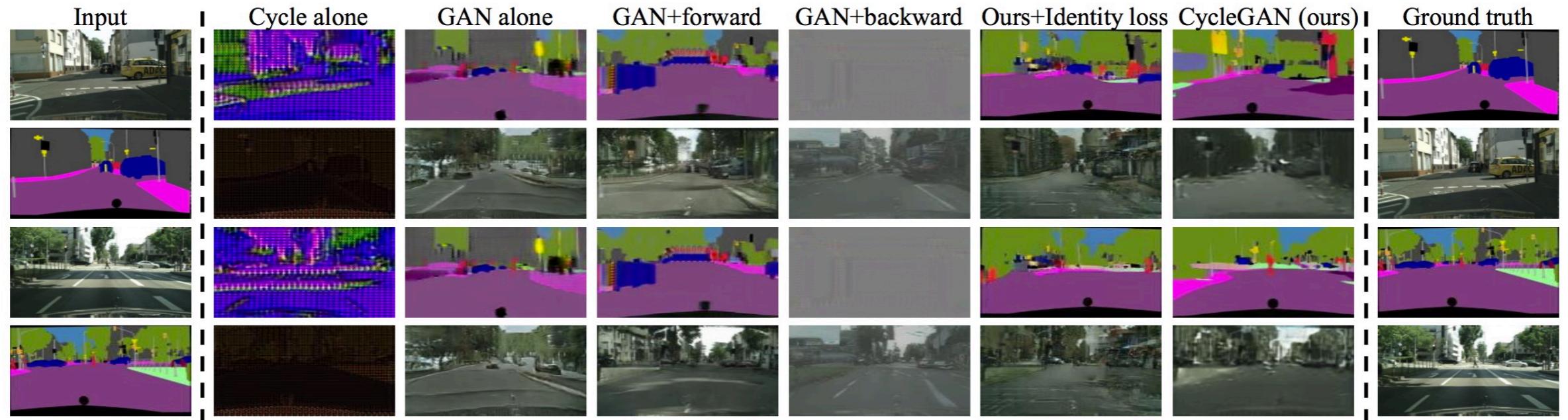
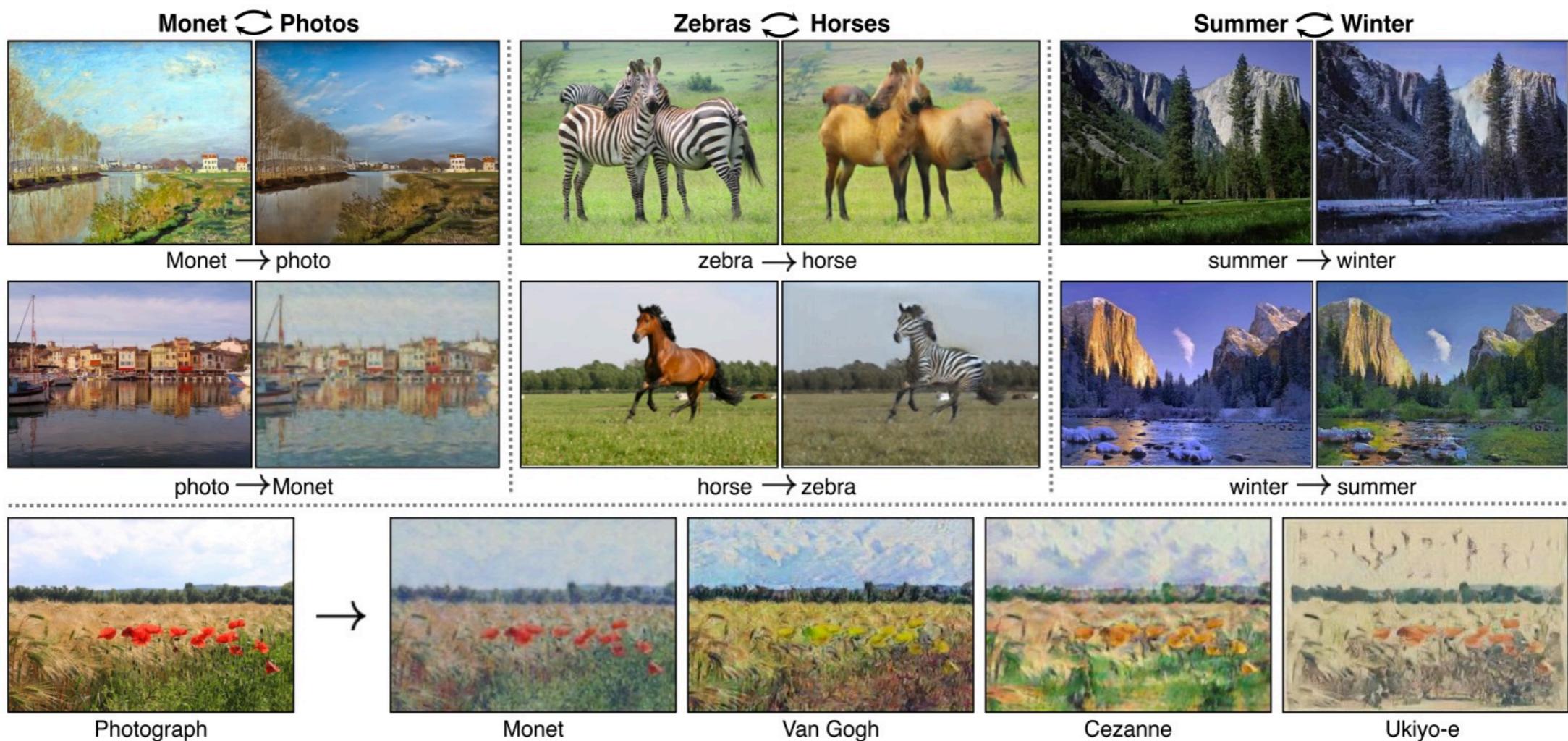
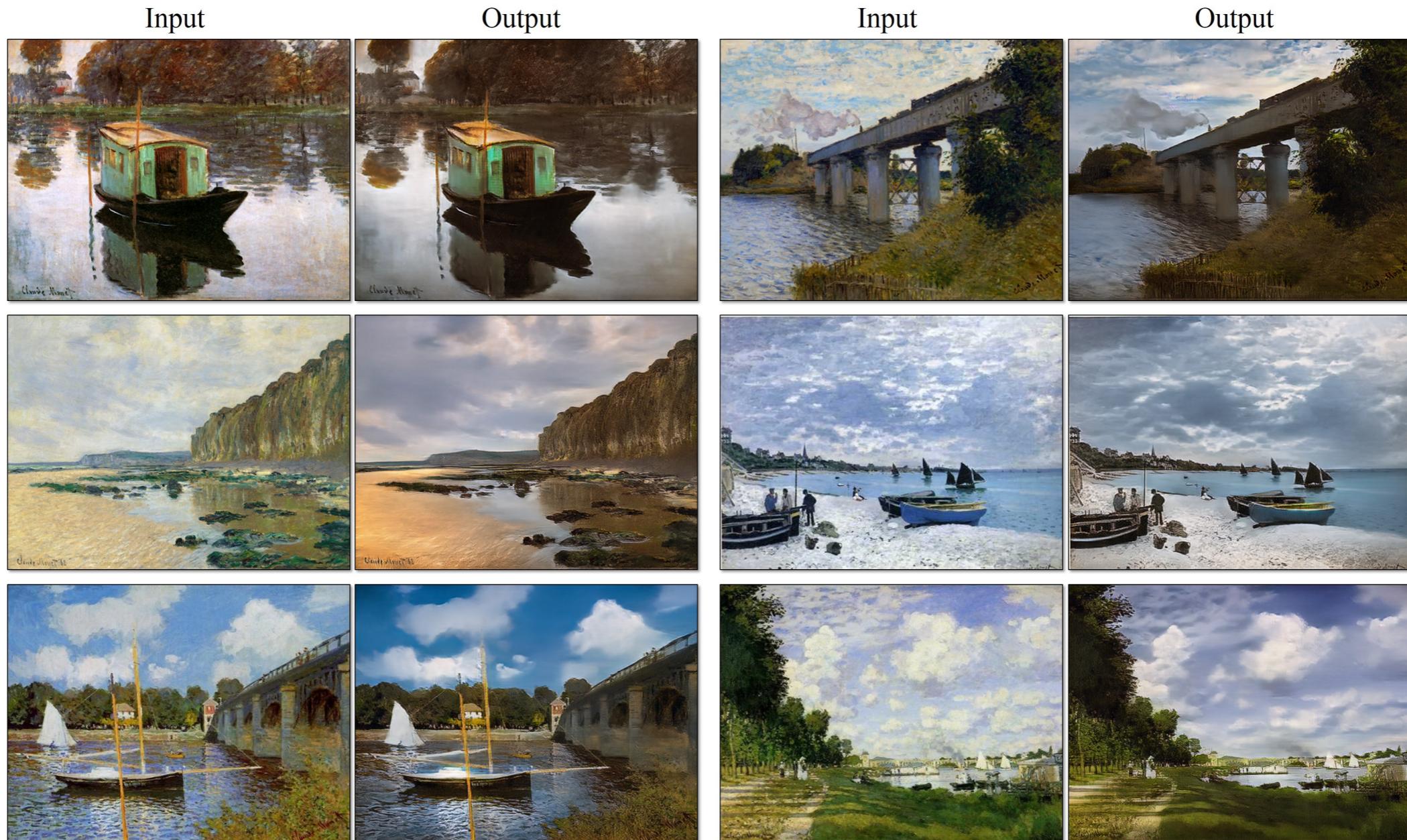


Figure 7: Different variants of our method for mapping labels $\leftrightarrow$ photos trained on cityscapes. From left to right: input, cycle-consistency loss alone, adversarial loss alone, GAN + forward cycle-consistency loss ( $F(G(x)) \approx x$ ), GAN + backward cycle-consistency loss ( $G(F(y)) \approx y$ ), CycleGAN (our full method), and ground truth. Both *Cycle alone* and *GAN + backward* fail to produce images similar to the target domain. *GAN alone* and *GAN + forward* suffer from mode collapse, producing identical label maps regardless of the input photo.

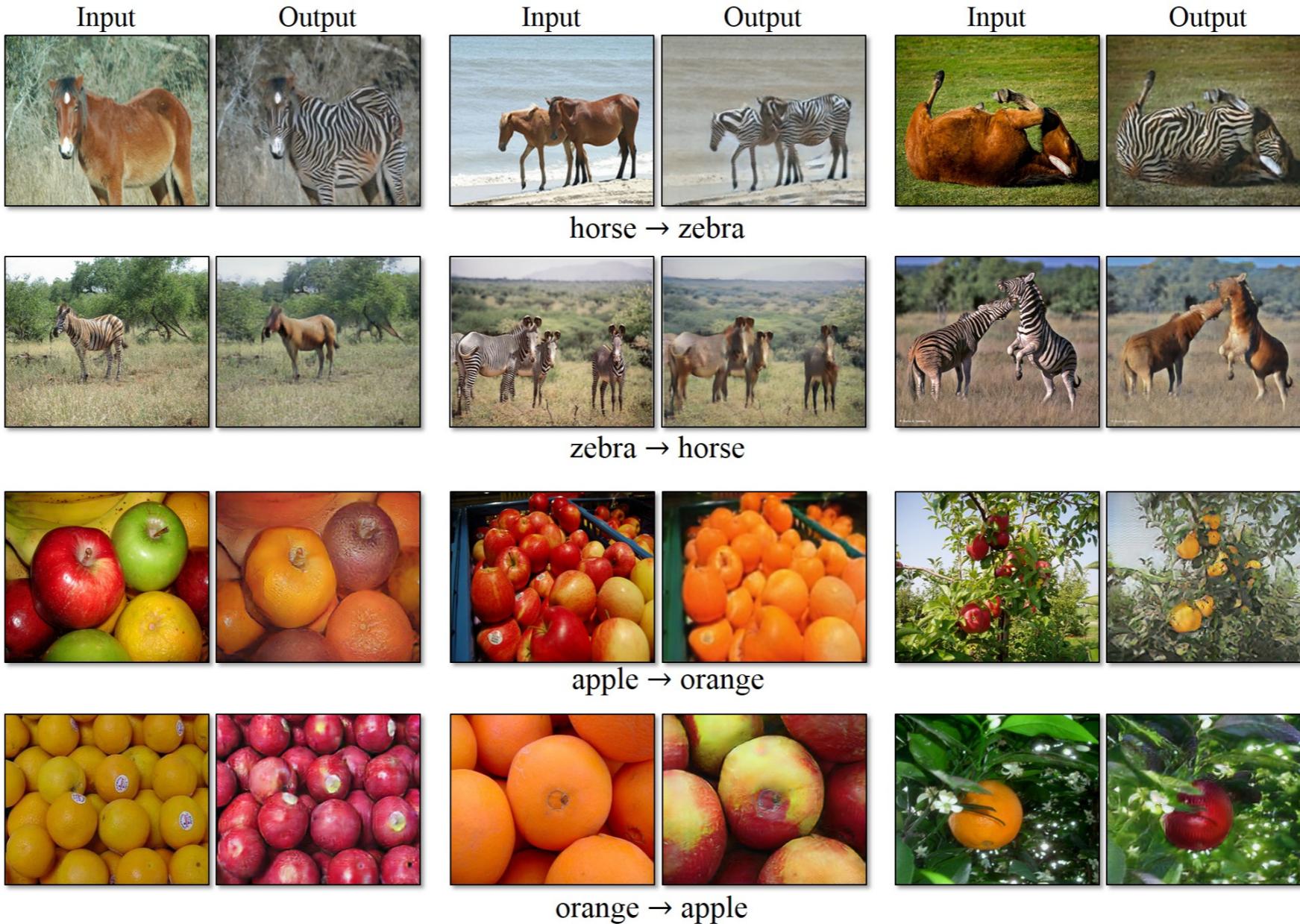
# Unpaired case



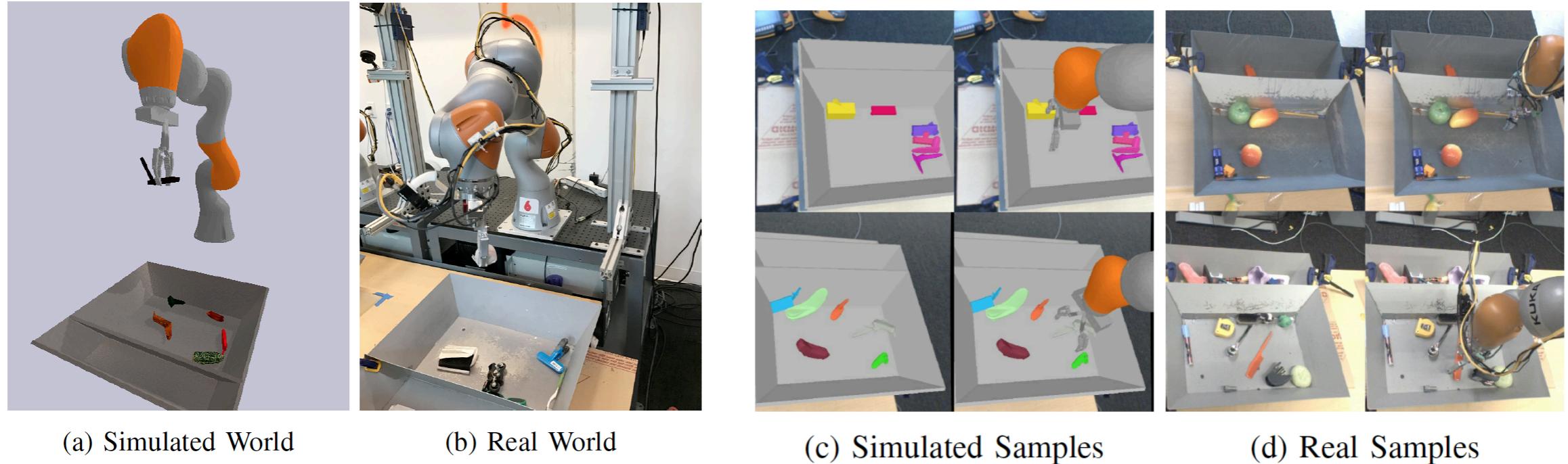
# Unpaired case



# Unpaired case

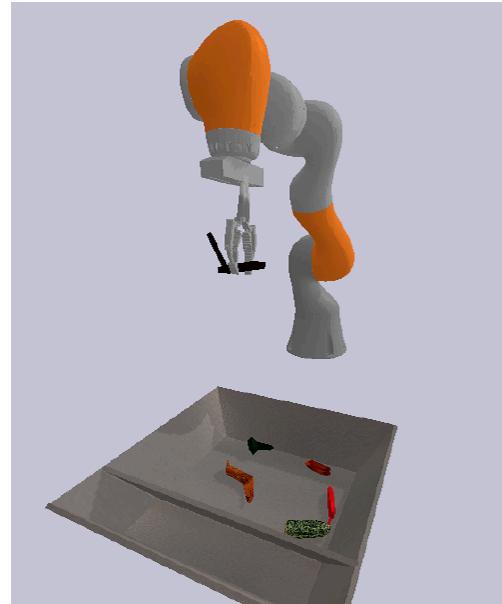


# SIM2real for learning to grasp

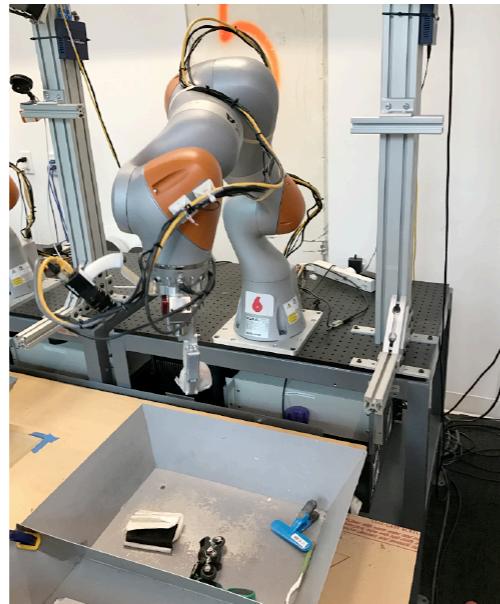


- I want to learn the function  $\text{Grasp}(I, v; \theta)$  : given image  $I$  and end-effector motion  $v$ , will I successfully grasp the object?
- $\text{Grasp}(I, v; \theta)$  can be trained with supervised learning. I want to use a simulated environment to quickly collect lots of samples. I want it to generalize to the real world.

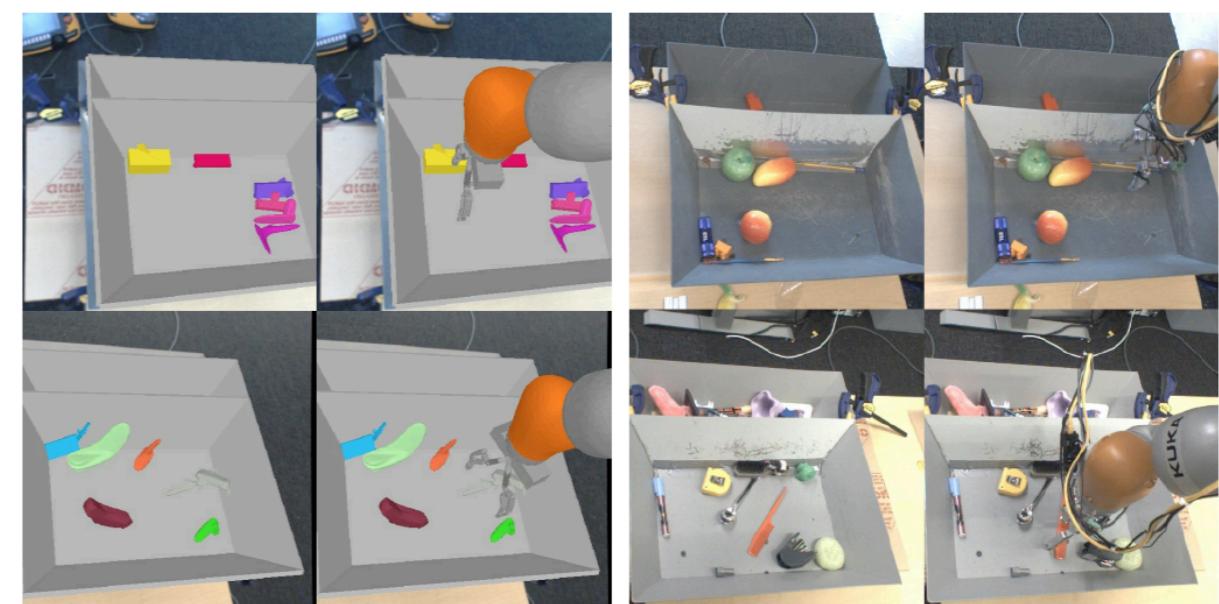
# SIM2real for learning to grasp



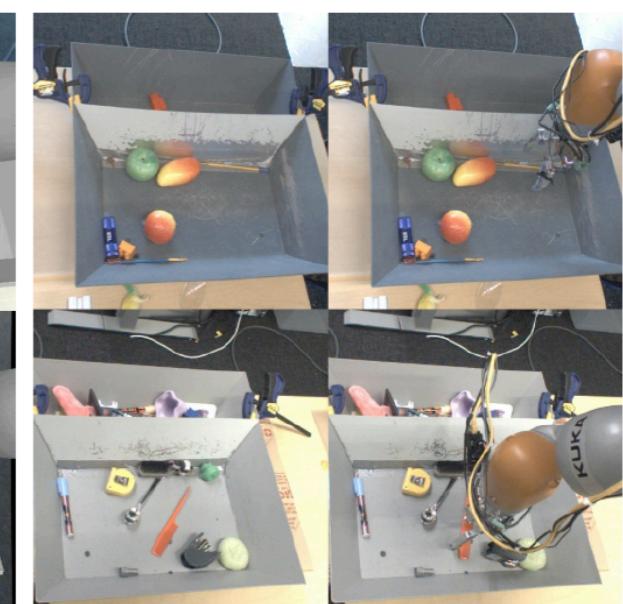
(a) Simulated World



(b) Real World



(c) Simulated Samples



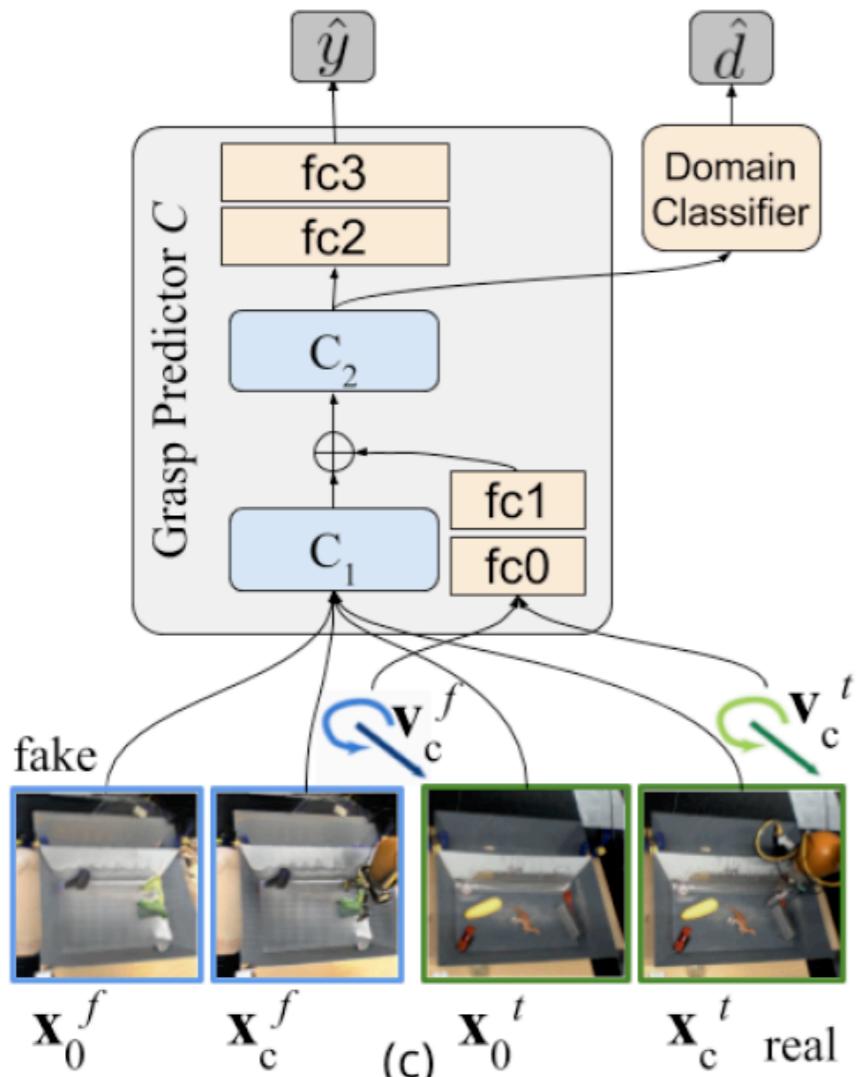
(d) Real Samples

- Use Bullet simulator to emulate the Kuka hardware setup. Camera is mounted over the Kuka shoulder
- 51300 ShapeNet 3D models
- Use progressively better grasping models to collect data
- Randomization: both visuals and dynamics were randomized in simulation: the background image, object masses, textures, coefficients of friction.

# Feature adaptation

Two losses: domain confusion loss and grasping prediction loss

Grasping prediction (task loss)



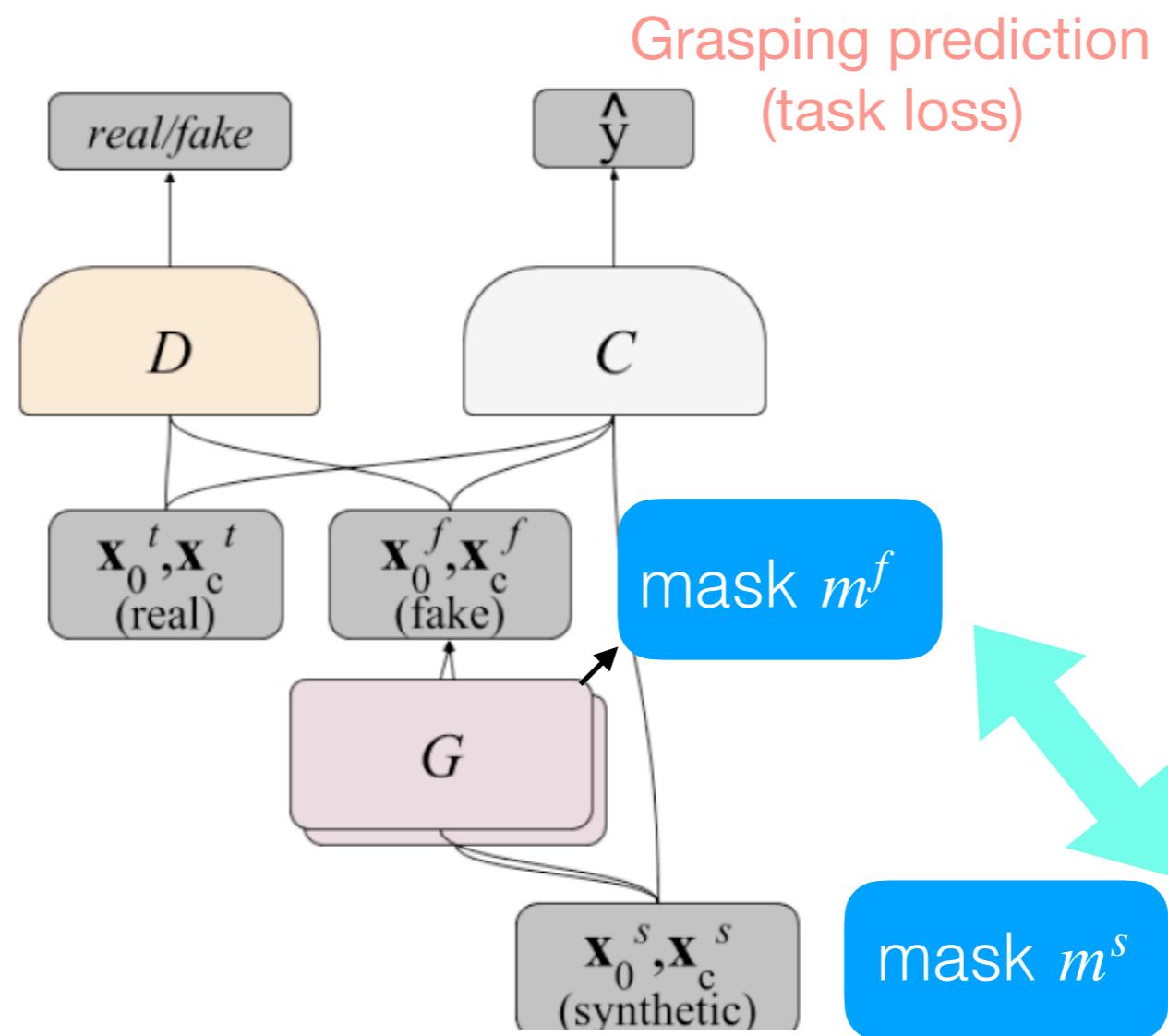
We add a domain classifier, that attempts to classify the domain the features come from

$$\mathcal{L}_{\text{DANN}} = \sum_{i=0}^{N_s+N_t} \{ d_i \log \hat{d}_i + (1 - d_i) \log (1 - \hat{d}_i) \}$$

The shared features  $C_1$ ,  $C_2$  attempt to confuse the domain classifier (maximize its loss), while the domain classifier features attempts to min. its loss.

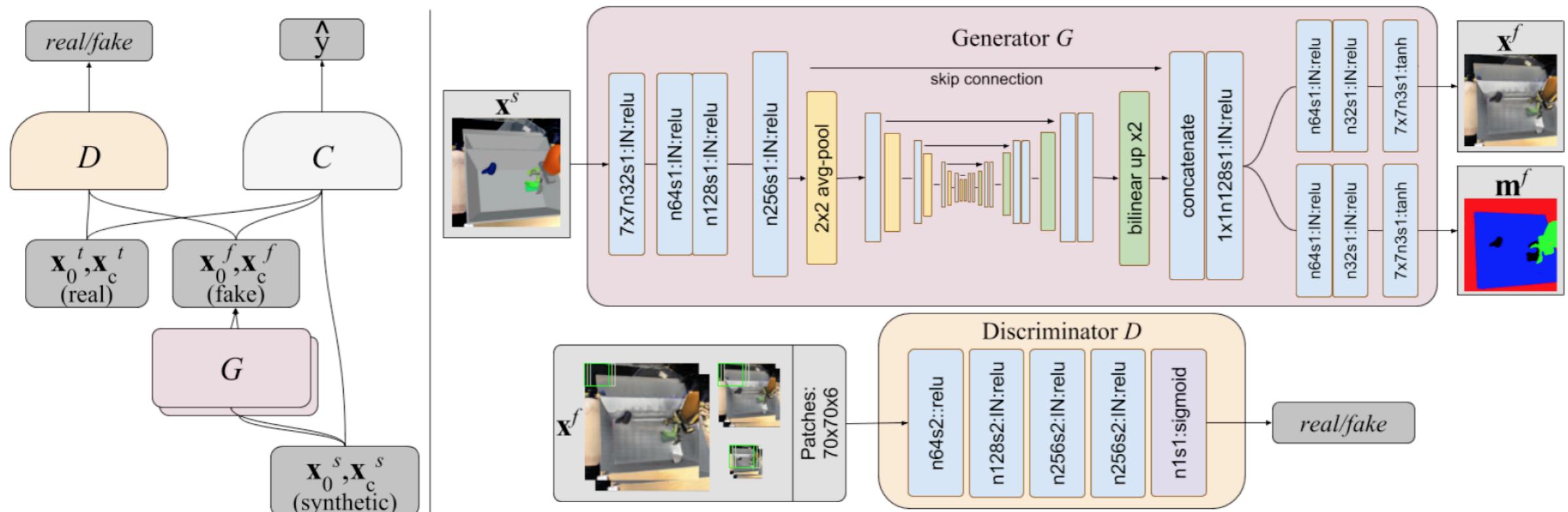
# Pixel Adaptation

Three losses: grasping prediction loss, semantic labelling loss, adversarial loss



# Pixel Adaptation

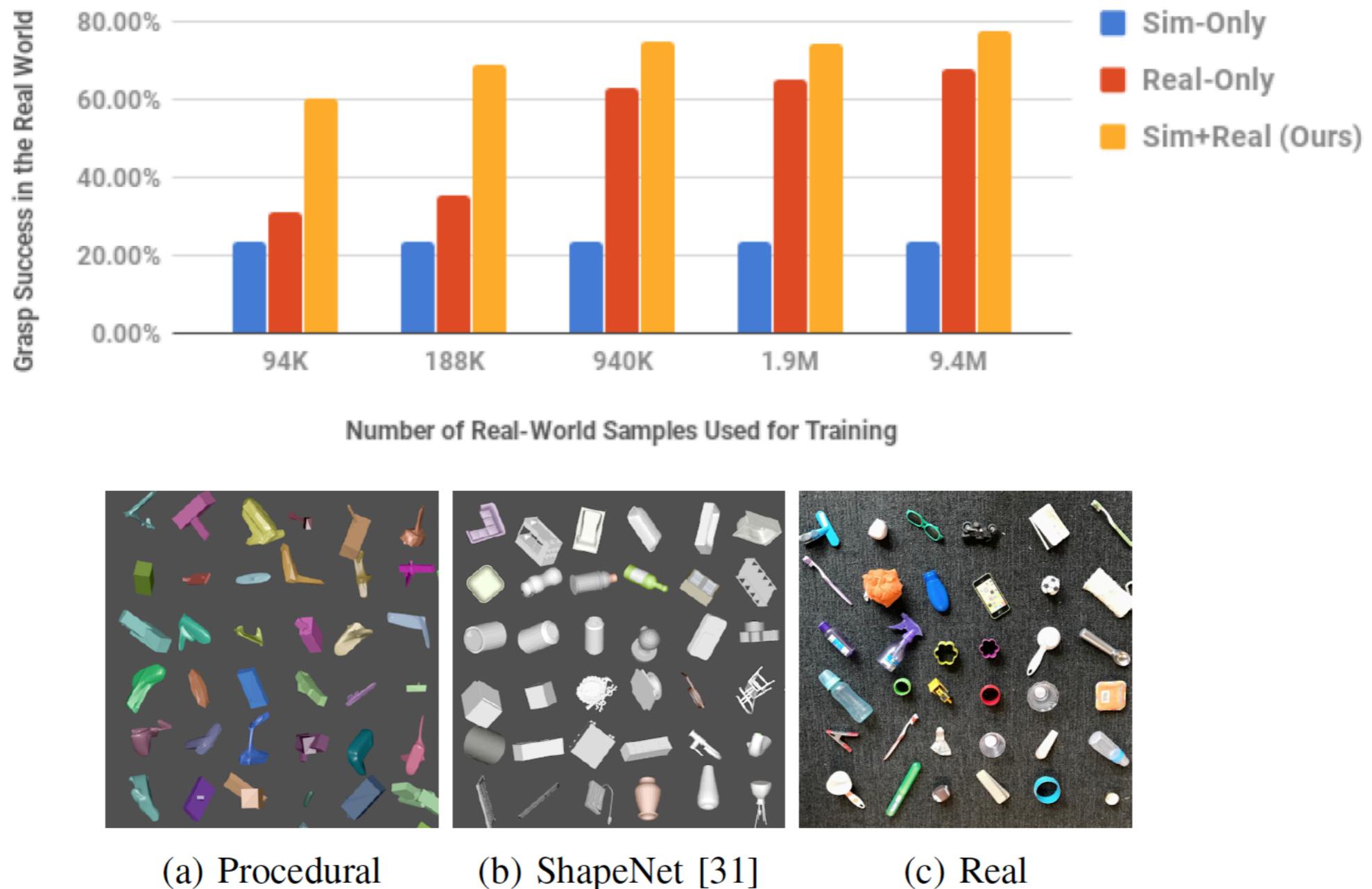
Three losses: grasping prediction loss, semantic labelling loss, adversarial loss



Goal: we want our generator to **translate simulated images** so that:

1. they do well in the task loss (grasping),
2. look real
3. retain the same semantics as their simulated counterparts

# Results



# What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the dymulator to match the real domain
- Learning not from pixels but rather from label maps-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

# EPOPT: LEARNING ROBUST NEURAL NETWORK POLICIES USING MODEL ENSEMBLES

**Aravind Rajeswaran<sup>1</sup>, Sarvjeet Ghotra<sup>2</sup>, Balaraman Ravindran<sup>3</sup>, Sergey Levine<sup>4</sup>**

aravraj@cs.washington.edu, sarvjeet.13it236@nitk.edu.in,  
ravi@cse.iitm.ac.in, svlevine@eecs.berkeley.edu

<sup>1</sup> University of Washington Seattle

<sup>2</sup> NITK Surathkal

<sup>3</sup> Indian Institute of Technology Madras

<sup>4</sup> University of California Berkeley

Ideas:

- Consider a **distribution over simulation models** instead of a single one for learning policies robust to modeling errors that work well under many ``worlds''. Hard model mining.
- Progressively **bring the simulation model distribution closer to the real world**.

# Policy Search under model distribution

Learn a policy that performs best in expectation over MDPs in the source domain distribution:

$$\mathbb{E}_{p \sim \mathcal{P}} \left[ \mathbb{E}_{\hat{\tau}} \left[ \sum_{t=0}^{T-1} \gamma^t r_t(s_t, a_t) \mid p \right] \right]$$

p: simulator parameters

I consider a distribution over simulation parameters, as opposed to a single set

# Policy Search under model distribution

Learn a policy that performs best in expectation over MDPs in the source domain distribution:

$$\mathbb{E}_{p \sim \mathcal{P}} \left[ \mathbb{E}_{\hat{\tau}} \left[ \sum_{t=0}^{T-1} \gamma^t r_t(s_t, a_t) \mid p \right] \right]$$

p: simulator parameters

## Hard world model mining

Learn a policy that performs best in expectation over **the worst** \epsilon- percentile of MDPs in the source domain distribution

$$\max_{\theta, y} \quad \int_{\mathcal{F}(\theta)} \eta_{\mathcal{M}}(\theta, p) \mathcal{P}(p) dp \quad \quad s.t. \quad \mathbb{P}(\eta_{\mathcal{M}}(\theta, P) \leq y) = \epsilon$$

# Hard model mining

---

**Algorithm 1:** EPOpt- $\epsilon$  for Robust Policy Search

---

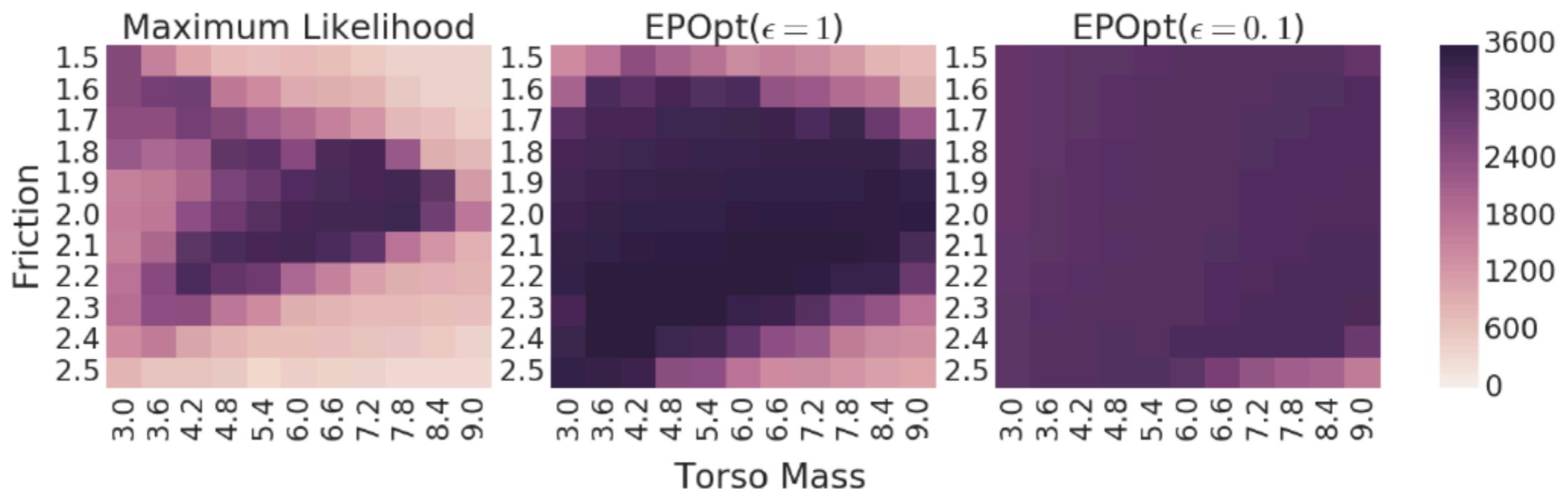
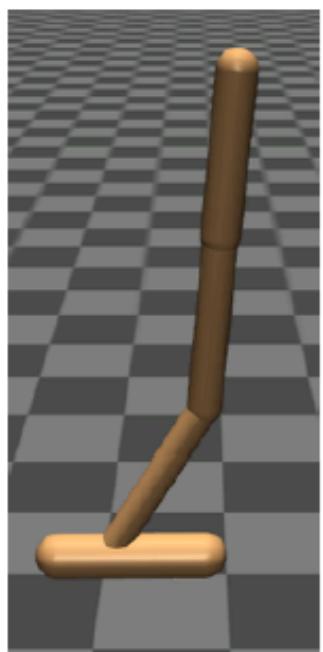
```
1 Input:  $\psi, \theta_0, niter, N, \epsilon$ 
2 for iteration  $i = 0, 1, 2, \dots niter$  do
3   for  $k = 1, 2, \dots N$  do
4     sample model parameters  $p_k \sim \mathcal{P}_\psi$ 
5     sample a trajectory  $\tau_k = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$  from  $\mathcal{M}(p_k)$  using policy  $\pi(\theta_i)$ 
6   end
7   compute  $Q_\epsilon = \epsilon$  percentile of  $\{R(\tau_k)\}_{k=1}^N$ 
8   select sub-set  $\mathbb{T} = \{\tau_k : R(\tau_k) \leq Q_\epsilon\}$ 
9   Update policy:  $\theta_{i+1} = \text{BatchPolOpt}(\theta_i, \mathbb{T})$ 
10 end
```

---

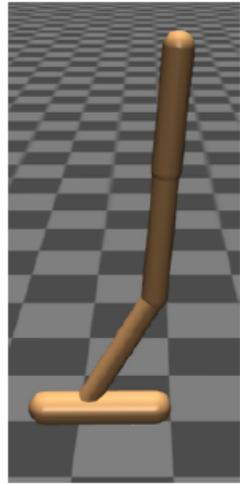
Select the simulation parameters where the current policy fails

# Hard model mining results

Hard world mining results in policies with high reward over wider range of parameters

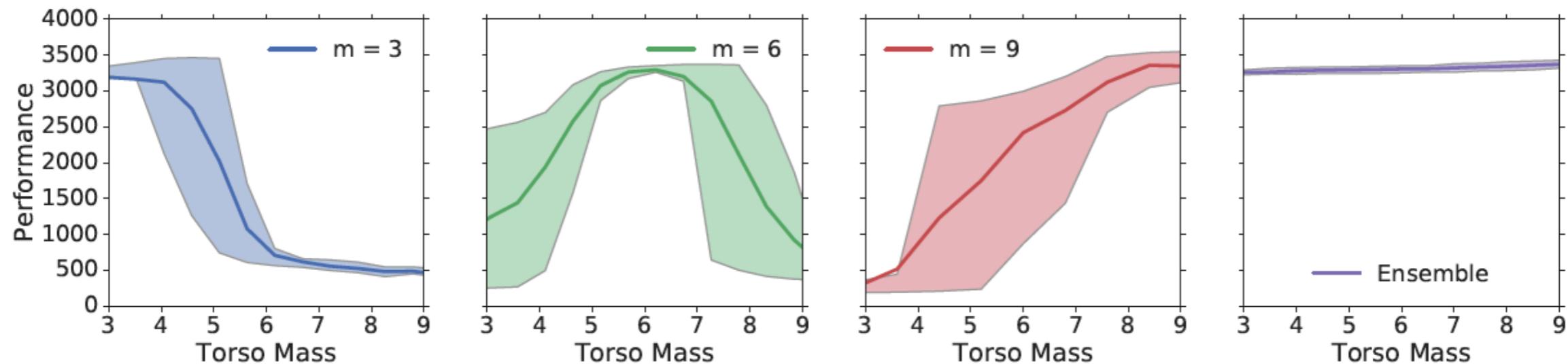


# Performance on hopper policies



trained on single source domains

trained on Gaussian distribution of mean mass 6 and standard deviation 1.5



# What can go wrong with dynamics randomization?

- Overly conservative policies
- Same instances of the problem may not have solution and hinder policy search
- Instead: try to bring the simulation dynamics closer to the real world dynamics

# What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- **Learning to adapt the dynamics of the dymulator to match the real domain**
- Learning not from pixels but rather from label maps-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

# Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience

Yevgen Chebotar<sup>1,2</sup> Ankur Handa<sup>1</sup> Viktor Makoviychuk<sup>1</sup>  
Miles Macklin<sup>1,3</sup> Jan Issac<sup>1</sup> Nathan Ratliff<sup>1</sup> Dieter Fox<sup>1,4</sup>

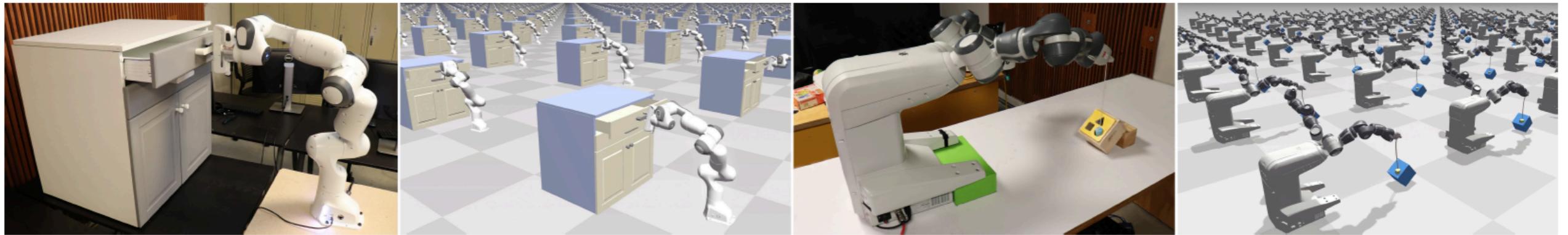


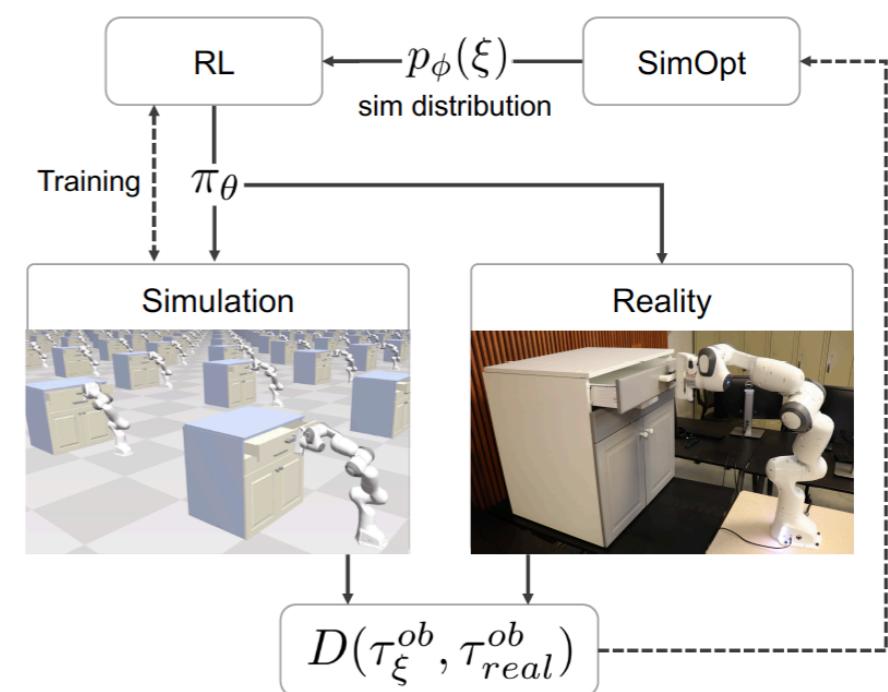
Fig. 1. Policies for opening a cabinet drawer and swing-peg-in-hole tasks trained by alternatively performing reinforcement learning with multiple agents in simulation and updating simulation parameter distribution using a few real world policy executions.

- Main idea: dynamics randomization can fail: policies may be impossible from some parameter ranges
- Instead of randomizing dynamics, attempt to adapt dynamics (simulation parameters) to bring them closer to reality

# Adapting simulation to the real world

## Algorithm 1 SimOpt framework

```
1:  $p_{\phi_0} \leftarrow$  Initial simulation parameter distribution
2:  $\epsilon \leftarrow$  KL-divergence step for updating  $p_{\phi}$ 
3: for iteration  $i \in \{0, \dots, N\}$  do
4:   env  $\leftarrow$  Simulation( $p_{\phi_i}$ )
5:    $\pi_{\theta, p_{\phi_i}} \leftarrow$  RL(env)
6:    $\tau_{real}^{ob} \sim$  RealRollout( $\pi_{\theta, p_{\phi_i}}$ )
7:    $\xi \sim$  Sample( $p_{\phi_i}$ )
8:    $\tau_{\xi}^{ob} \sim$  SimRollout( $\pi_{\theta, p_{\phi_i}}, \xi$ )
9:    $c(\xi) \leftarrow D(\tau_{\xi}^{ob}, \tau_{real}^{ob})$ 
10:   $p_{\phi_{i+1}} \leftarrow$  UpdateDistribution( $p_{\phi_i}, \xi, c(\xi), \epsilon$ )
```



$$\begin{aligned} & \min_{\phi_{i+1}} \mathbb{E}_{P_{\xi_{i+1}} \sim p_{\phi_{i+1}}} \left[ \mathbb{E}_{\pi_{\theta, p_{\phi_i}}} \left[ D(\tau_{\xi_{i+1}}^{ob}, \tau_{real}^{ob}) \right] \right] \\ \text{s.t. } & D_{KL} (p_{\phi_{i+1}} \| p_{\phi_i}) \leq \epsilon, \end{aligned}$$

# What has shown to work

- Domain randomization (dynamics, visuals)
- Learning to adapt the textures of the simulator to match the real domain
- Learning to adapt the dynamics of the dymulator to match the real domain
- Learning policies not from pixels but rather from label maps-> semantic maps between simulation and real world are closer than textures
- Learning higher level policies, not low-level controllers, as the low level dynamics are very different between Sim and REAL

# Driving Policy Transfer via Modularity and Abstraction

**Matthias Müller**  
Visual Computing Center  
KAUST, Saudi Arabia

**Alexey Dosovitskiy**  
Intelligent Systems Lab  
Intel Labs, Germany

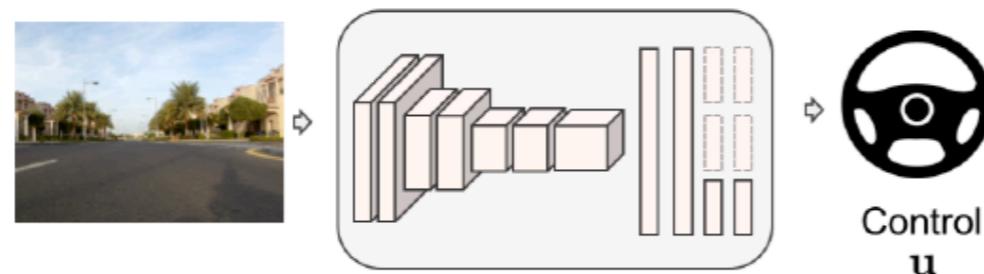
**Bernard Ghanem**  
Visual Computing Center  
KAUST, Saudi Arabia

**Vladlen Koltun**  
Intelligent Systems Lab  
Intel Labs, USA

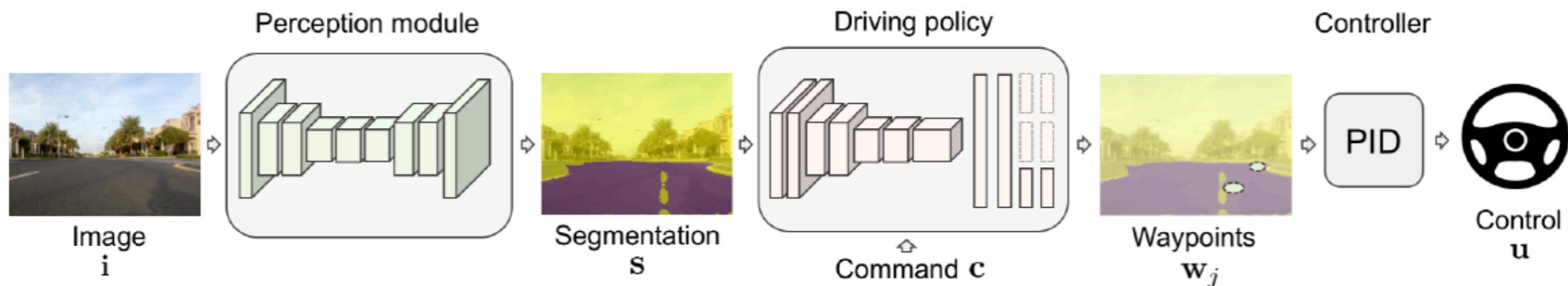
**Idea:** the driving policy is not directly exposed to raw perceptual input or low-level vehicle dynamics.

# Main idea

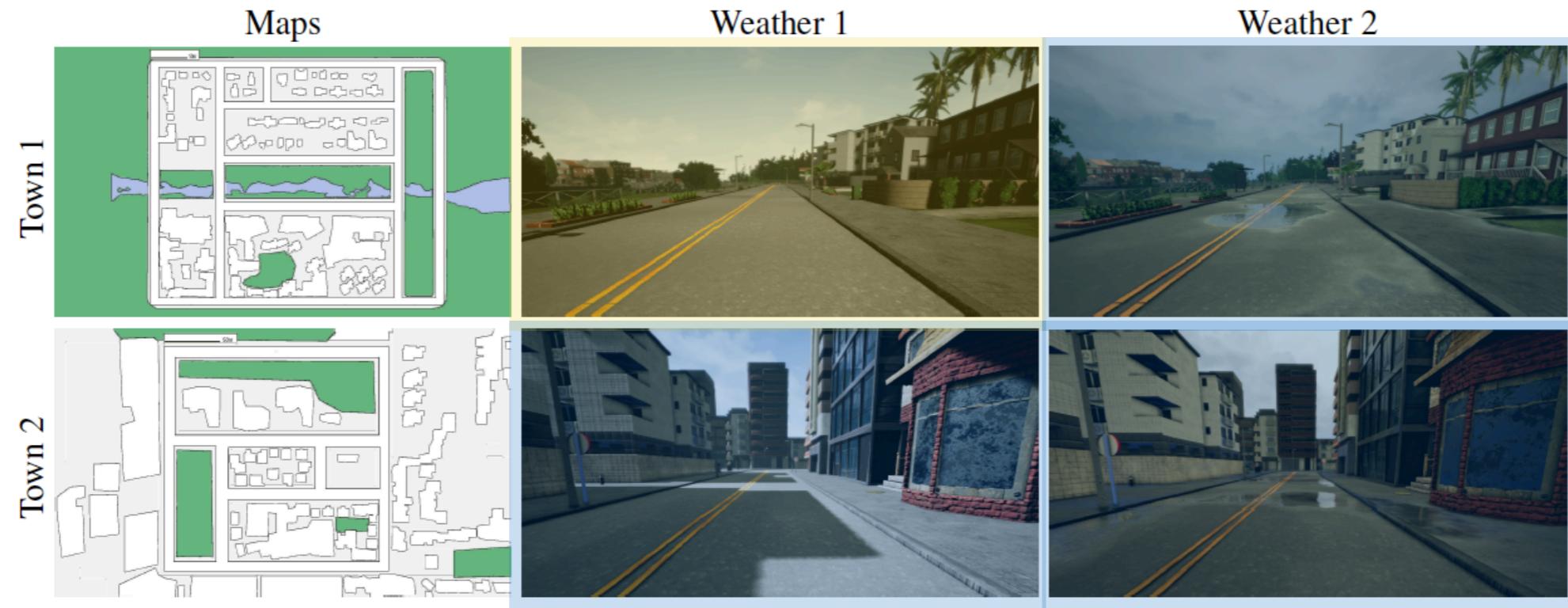
Pixels to steering wheel mapping is not SIM2REAL transferable: image textures and car dynamics mismatch



Instead: **label maps to waypoint** mapping is better SIM2REAL transferable:  
label maps and waypoints are similar across SIM and REAL. A low-level  
controller will take the car from waypoint to waypoint in the real world



# Train/Test



We train policies via behaviour cloning (standard regression loss) in Town1/ Weather1 dataset, and evaluate them on all four.

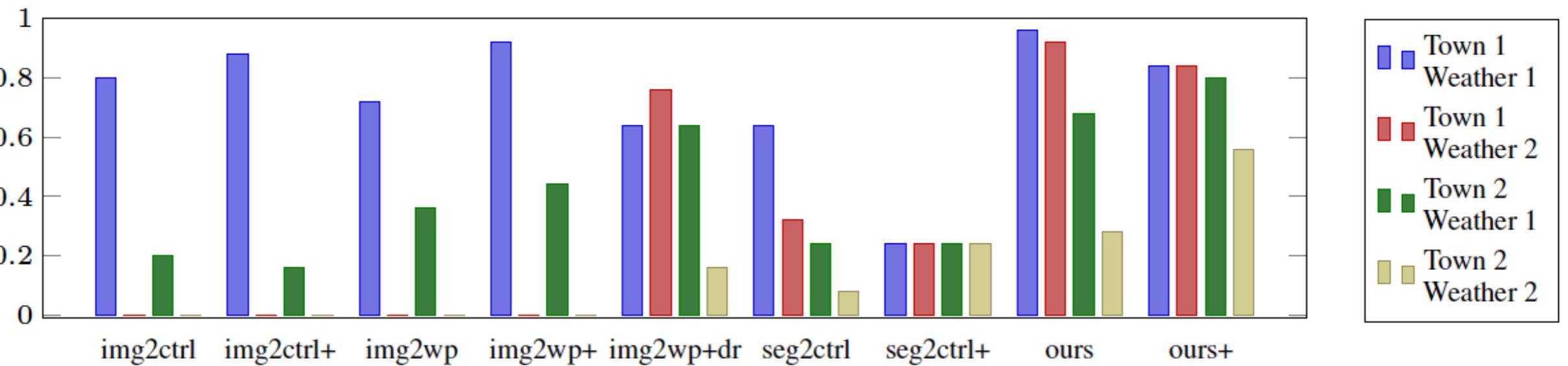


Figure 4: Quantitative evaluation of goal-directed navigation in simulation. We report the success rate over 25 navigation trials in four town-weather combinations. The models have been trained in Town 1 and Weather 1. The evaluated models are: *img2ctrl* – predicting low-level control from color images; *img2wp* – predicting waypoints from color images; *seg2ctrl* – predicting low-level control from the segmentation produced by the perception module; *ours* – predicting waypoints from the segmentation produced by the perception module. Suffix ‘+’ denotes models trained with data augmentation, and ‘+dr’ denotes the model trained with domain randomization.