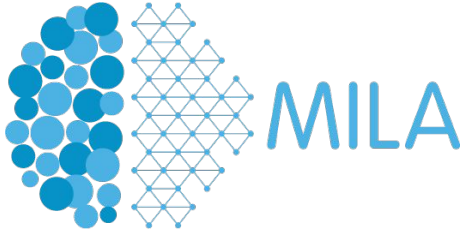
Computer Vision: (Way) Beyond Classification

Dumitru Erhan
Senior Research Scientist @ Google Brain

Transylvanian Machine Learning Summer School 2018
July 17, 2018

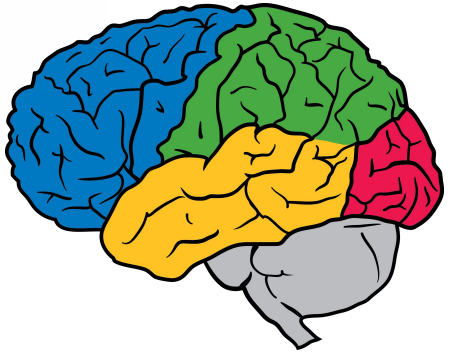
About myself



- PhD @ University of Montreal 2011
  - Pre-training deep nets!
- Scientist at Yahoo! Labs 2011--2012
  - Search ranking
- Google Photos 2012--2015
  - Image understanding, object detection, im2txt
- Google Brain 2015--now:
  - Domain adaptation
  - Video generation
  - Understanding interpretability

YAHOO!
LABS

Google Photos

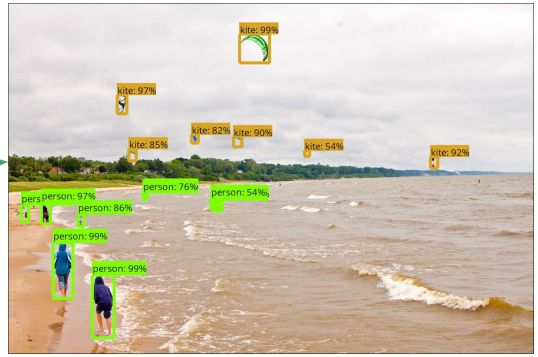


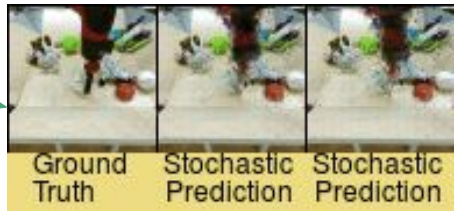
Talk outline

- Object detection
- Visual domain adaptation
- Video prediction
- **Mostly**: a whirlwind overview of challenging research & practical problems in Computer Vision.
- **Hopefully**:
  - It will inspire you to dig deeper.
  - It won't confuse you too much!

Why these 3 topics?

- Object detection:
  - Very common computer vision problem.
  - .... controversial opinion: in a narrow sense, solved.
- Domain adaptation:
  - Hugely important
  - ... we have good solutions
  - ... but not really solved yet.
- Video prediction:
  - **Potentially** hugely important
  - ... very challenging
  - ... current state of the art somewhat terrible.

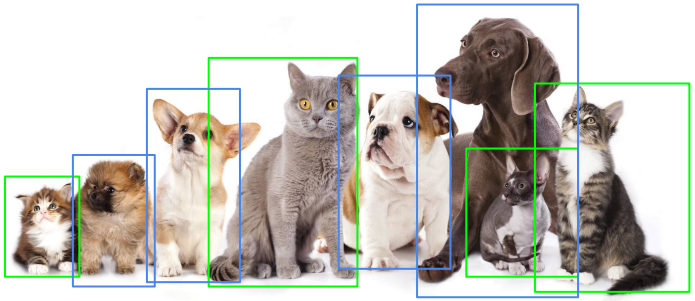




Warning before we go ahead

- Each of these topics deserve their own lecture.
- Aiming to present the some of the
  - **basic intuitions**
  - **a flavor of the methods used to solve the problem.**
  - **current research challenges and open questions.**
  - **Bonus: biased towards worked done by me & colleagues.**
- Deliberately not aiming for
  - **complete coverage**
  - **too many historical notes**
  - **lots details**
- With that being said… let’s start!

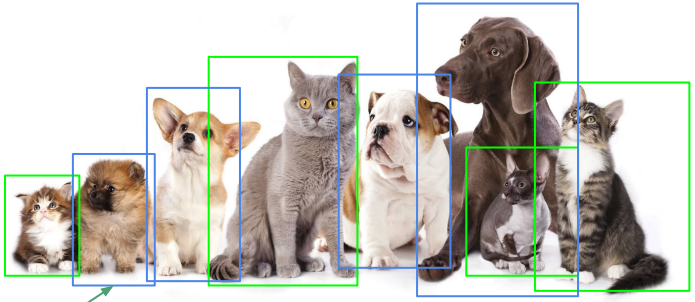
Object detection



- What is object detection?
  - Finding the **location and identity** of all the objects in an image
  - Strictly harder task than image classification
- Why is it useful?
  - **StreetView**: license plates, street signs, faces, house numbers.
  - **Self-driving cars**: person, street sign, cyclist, car detection.
  - **Google Photos**: face, pet etc detection.
  - **Generally:** true image understanding only possible if we know *where* things are and how they relate to each other.
- Main challenges:
  - Sufficient, accurate and *relevant* training data
  - Efficient (in practice) models.

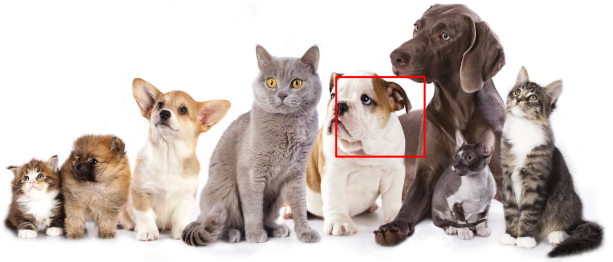
Image: tealswan.com

Why is detection hard(er)?



- Image annotation model typical output:
  - `cat_score: 0.98, dog_score:0.95`
- Object detection model:
  - `[0,0.1,0.2,0.8,cat_score: 0.8],`
  - `[0,1,0.2,0.2,0.8,dog_score: 0.7]` ….
- Typical evaluation: per-class precision-recall curve
  - **Rewards** models for correctly identifying where objects are.
  - **Penalizes** models for predicting objects where there are none.
- Real-world images are more cluttered, contain occlusions, unknown object types, not-really-objects etc.
- No obvious way to deal with variable length groundtruth and predictions.

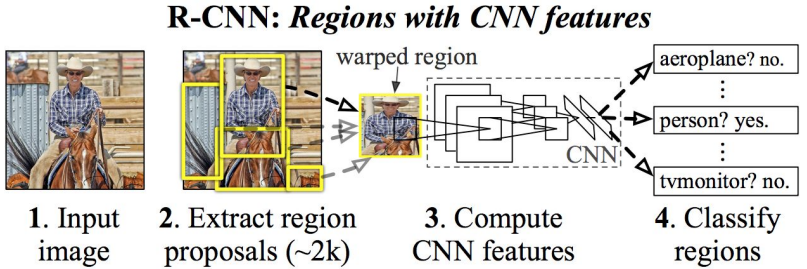
So how do you design an object detector? Idea #1



- Training time:
  - input: fixed-size window of an image
  - output: “how much of a cat is in the input?”
- Test time: Run the model on all possible windows
  - Could be millions!
- Before ~2012 (“classical” object detection):
  - Feature extraction at each window using histograms of oriented gradients (HOG)
  - Support Vector Machine model (SVM) trained on top (0/1 classifier). (eg. Felzenszwalb et al 2010)
  - If careful: can quickly scan through millions of subwindows efficiently.
- After ~2012 (“modern” object detection):
  - Replace HOG with Convolutional Neural Networks

So how do you design an object detector? Idea #2

- Training time: same as idea #1
- Testing time:
  - Testing: run on “interesting parts” of an image
- “Interesting” is a task-agnostic
  - Ideally an cheap-to-run model.
- Advantage:
  - No need to evaluate on potentially millions of useless subwindows
  - Fewer potential false positives
  - Much faster if underlying feature extractor is a huge deep network.
- (Crucial) disadvantage: no reason to thing that task-agnostic regions any good.



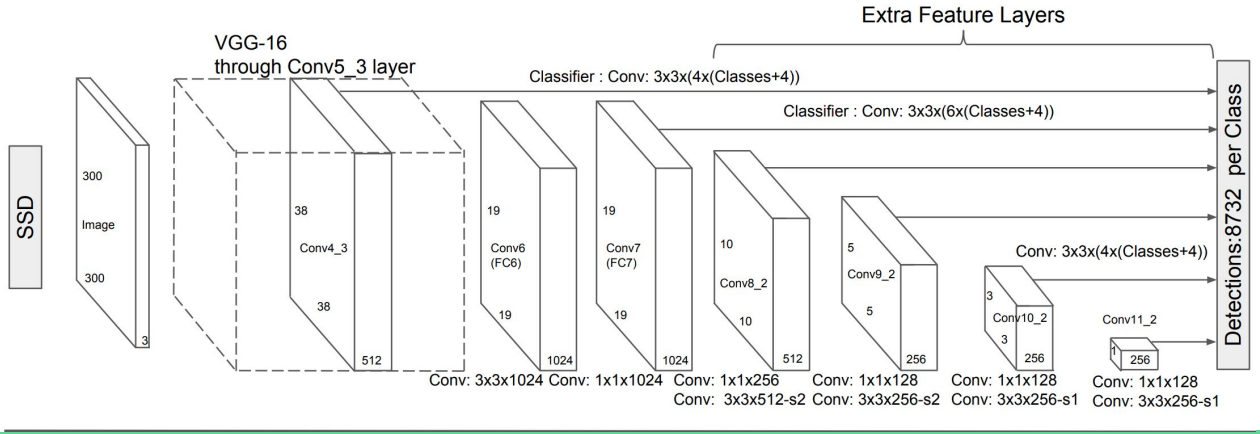
Girshick et al 2013.

So how do you design an object detector? Idea #3

- **Learn** where interesting regions are! But how?
- Have a “proposal” network: its goal is to learn where objects are.
  - Input: image, output: boxes and confidences that something is in there.
- Co-train a region classification network, as before
- How to deal with variable length number of objects?
- Use the multibox trick:
  - N groundtruth bounding boxes, P >> N predictions (boxes **and** confidences)
  - Use Hungarian algorithm to solve for the closest match: assign a P_i box to each groundtruth
  - At convergence, the model will give high scoring confidences only to good predicted boxes.
- Sounds great, but:
  - 2-stage quirky
  - Proposal network lumps classes together, perhaps unnaturally.

So how do you design an object detector? Idea #4

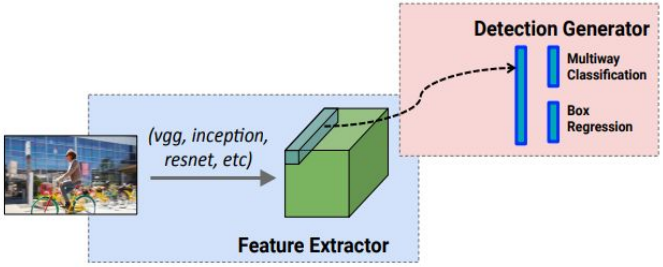
- **Give up on fancy stuff**, just learn everything end-to-end
- This is basically Single-Shot Detector or YOLO.
- From the feature maps at all levels, construct a fully-convolutional predictor
- Crucially, share parameters for each class
- Advantage: all in one architecture.
- Disadvantage: maybe won't work well if truly need to zoom in?



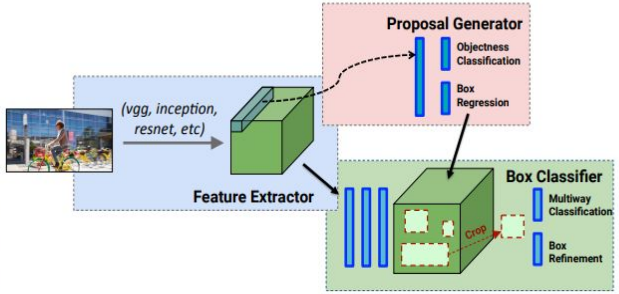
Liu et. al (2015) *Single-shot detector*

Modern object detection

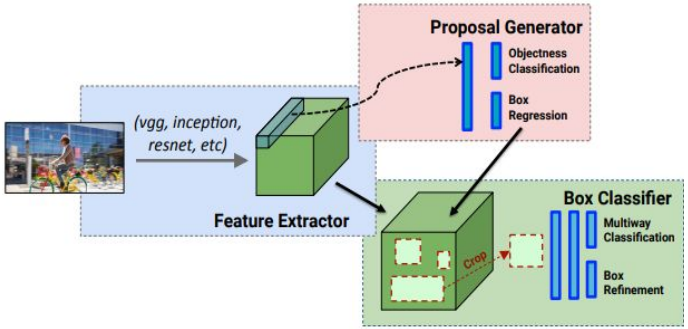
- Faster RCNN, SSD, YOLO : all relatively minor variations of the ideas presented thus far.
- Propose-and-zoom-and-classify vs do everything in one shot
- Fundamentally, we have a collection of good recipes that work well in controlled scenarios



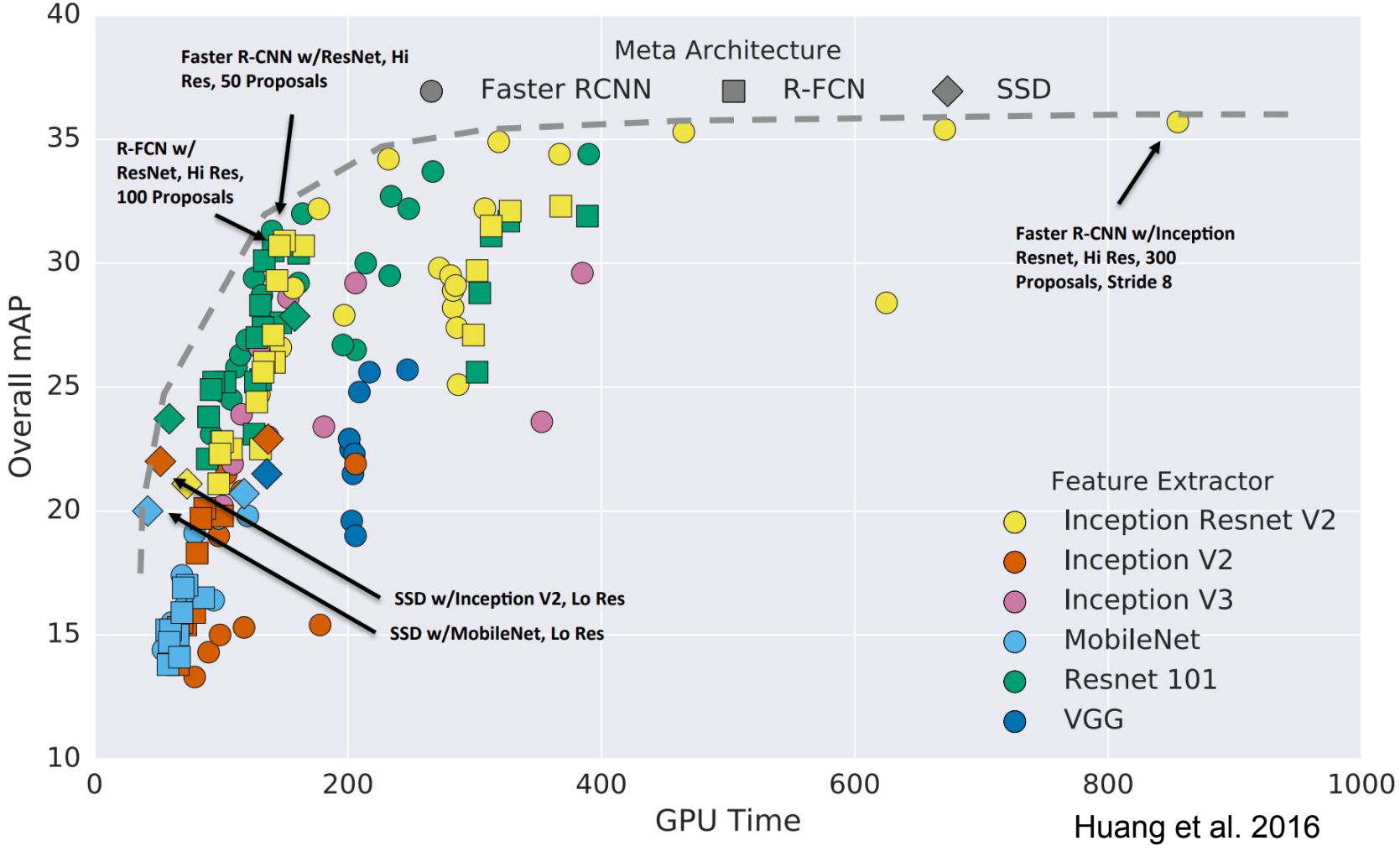
(a) SSD.



(c) R-FCN.



(b) Faster RCNN.

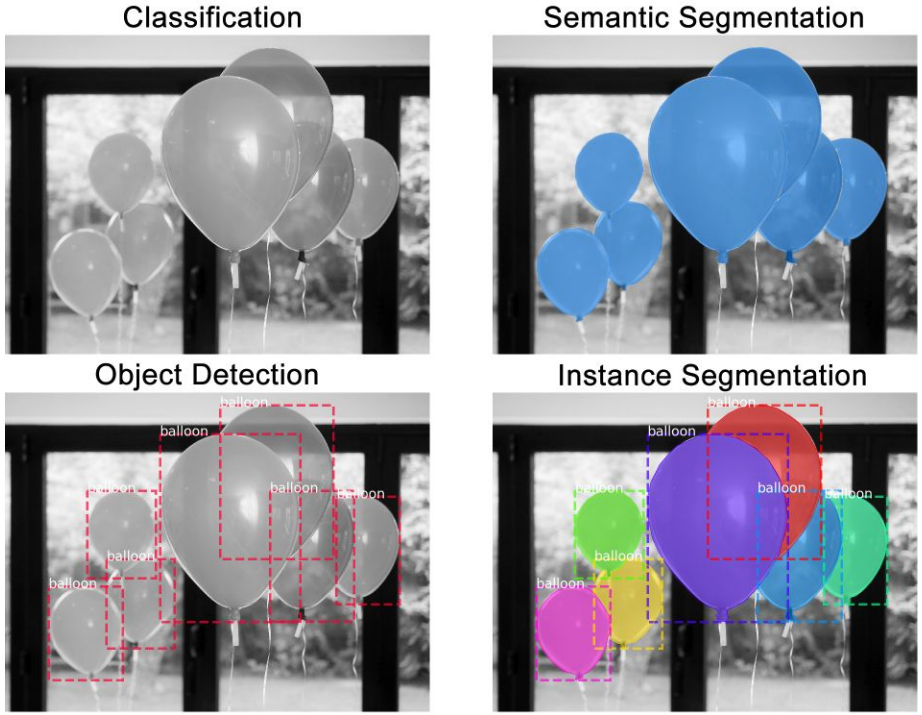


Huang et al. 2016

Object detection: conclusion I

- Surprising result: despite **pooling over spatial regions**, CNNs can localize well.
- Regressing to bounding boxes directly works surprisingly well.
- SSD etc are scalable methods for doing object detection
- No single method is **very fast and very good** at the same time.

Object detection conclusion II



https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow

Object detection: conclusion II

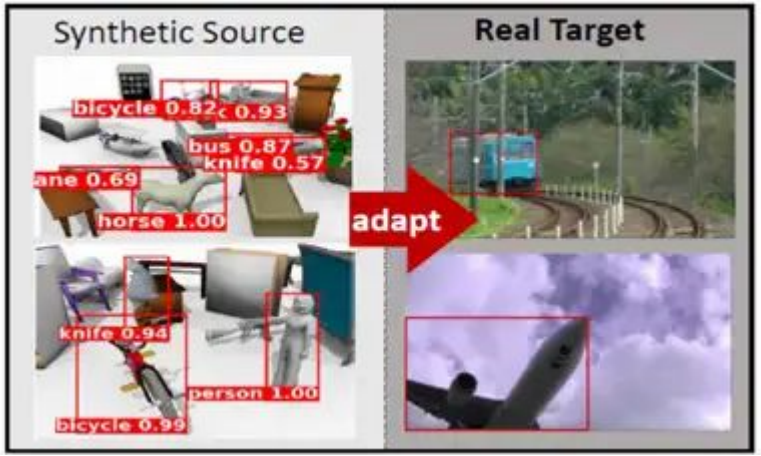
- Open issues:
  a. Real-world shapes rarely rectangular
  b. Certain things (sky, grass) cannot be easily described by a contiguous geometric groundtruth.
  c. Segmentation (semantic vs instance) is one way to solve this problem.
  d. Pixel-level annotation of images at scale is hard (and sometimes impractically expensive).
- One big issue: what if your training set is not representative of your real life?
  a. Train on clean web images, apply to video from phone camera, user's photos etc.
  b. No theoretical guarantees that it'll work, and in practice it may or may not.

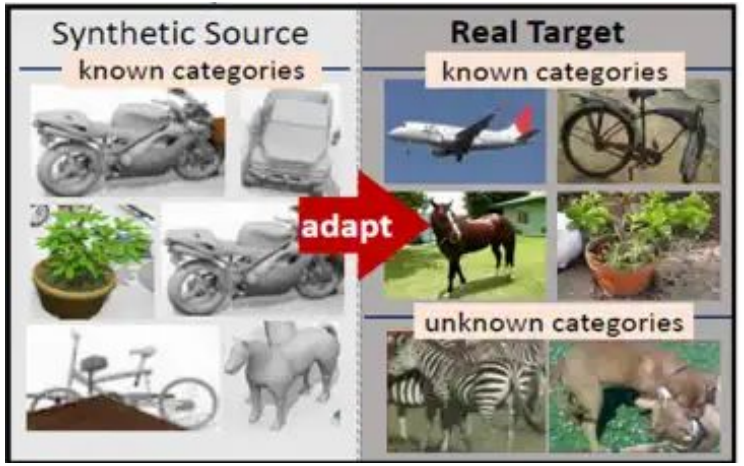
Can we do something about it?

Domain adaptation

- Domain adaptation is the desirable property of a model (agent)
- We would like the trained model to be resilient to all sorts of changes
  - Notably: changes in the data statistics (aka “covariate shift”)
- I will argue that being able to adapt to such changes is a **must** for any practically-deployed machine learning algorithm.
  - Assuming that you will get IID samples from the same distribution as your training set is… courageous.
  - Ads, news recommendation, self-driving cars: the environment changes **all the time**.

Visual domain adaptation





Images from ViSDA challenge website.

What is Domain Adaptation? Let’s be more concrete

The Goal:

- Train on a **source** dataset
- Apply on a **target** dataset generated from a different distribution
- Assumption for now: same label space

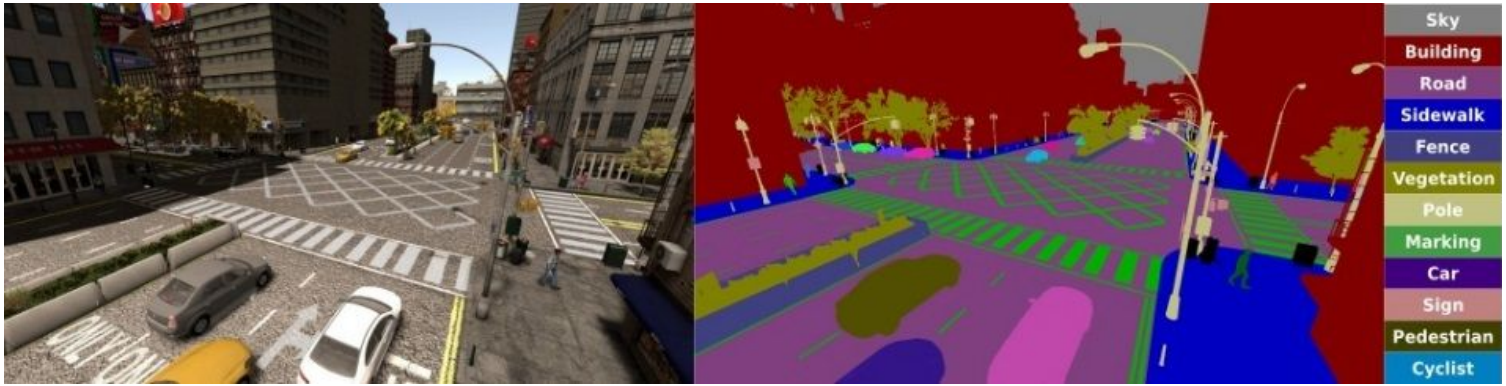
Unsupervised Domain Adaptation: **no labels** in the target domain
Semi-supervised Domain Adaptation: **fewer labels** in the target domain than in the source domain

Big focus on: Unsupervised domain adaptation

Why do we care?

- Creating high quality datasets is very costly and time-consuming!
- Certain annotation tasks are prohibitively expensive
  - 3D pose labels
  - Dense pixel-level labeling, e.g., instance/semantic segmentation

One solution: synthetic data and unsupervised domain adaptation to the real domain



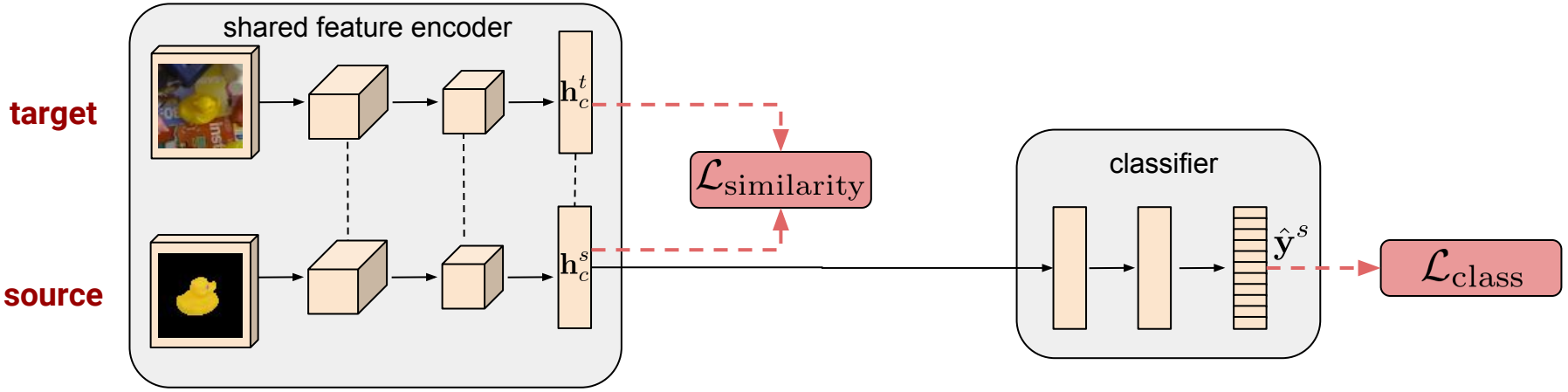
A bunch of slides from Konstantinos Bousmalis

Unsupervised Domain Adaptation for ConvNets

Three main types of methods:

- **Mapping** source domain features to target domain features
  - e.g. *Correlation Alignment* (CORAL)
- End-to-end learning of **domain-invariant features** with a similarity loss
  - Features *statistics matching* methods
    - A similarity loss to match the feature statistics at some level of the CNN
  - *Domain-adversarial* methods
    - A similarity loss that maximizes the confusion of a domain classifier
- **Pixel-to-pixel** methods
  - Create a model that can *transform* an image from a source to domain to have the same characteristics as an image from a target domain.

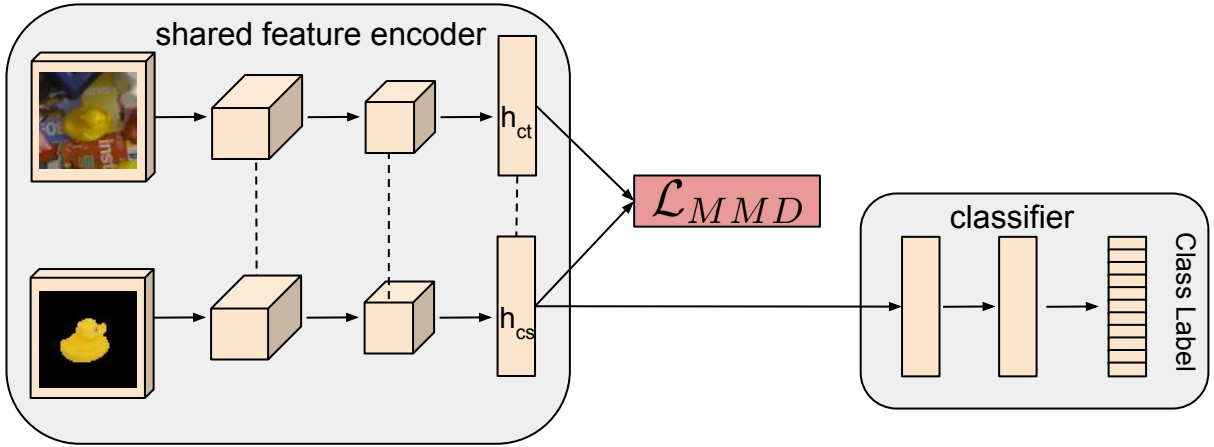
Domain-Invariant feature learning in a nutshell



$$\mathcal{L} = \mathcal{L}_{\text{class}} + \gamma\, \mathcal{L}_{\text{similarity}}$$

Maximum Mean Discrepancy Regularization Networks

Ghifary et al. 2014 *Domain Adaptive Neural Networks for Object Recognition*

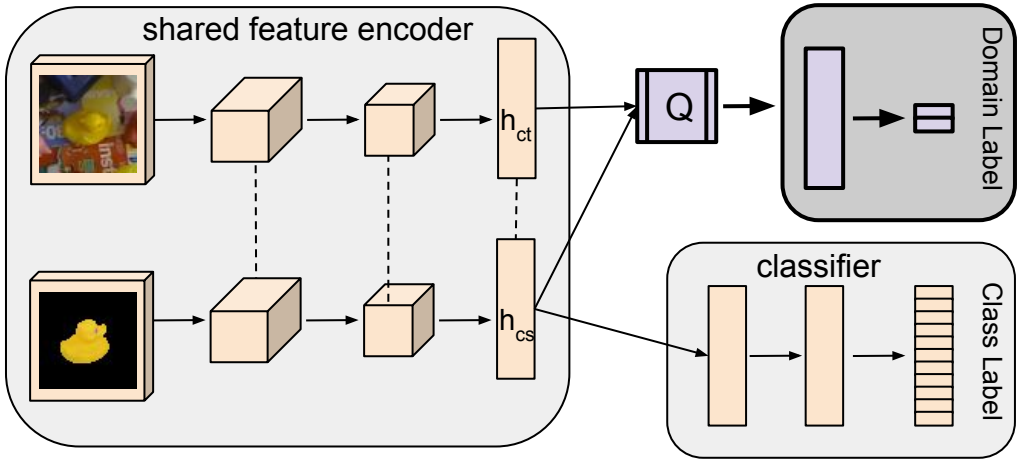


$\mathcal{L}_{\text{MMD}}$ a **kernel two-sample test** used as a similarity loss that pushes the domain representations to be similar.

Minimization of the MMD loss

Easy optimization goal, but choosing the **kernel parameters** is not trivial.

Domain-Adversarial Neural Networks



$$Q\left(f(u)\right) = f(u)$$

$$\frac{d}{du}Q(f(u)) = -\frac{d}{du}f(u)$$

- **Minimize** domain classification loss wrt the domain classifier parameters
- **Maximize** domain classification loss wrt to the encoder parameters
  - Possible with a **gradient reversal layer** *Q*
- Difficult **minimax optimization** goal

Ganin et al. (2016) *Domain-Adversarial Training of Neural Networks*

Domain Separation Networks (DSNs)

Previous: The produced features are contaminated by domain-private information.

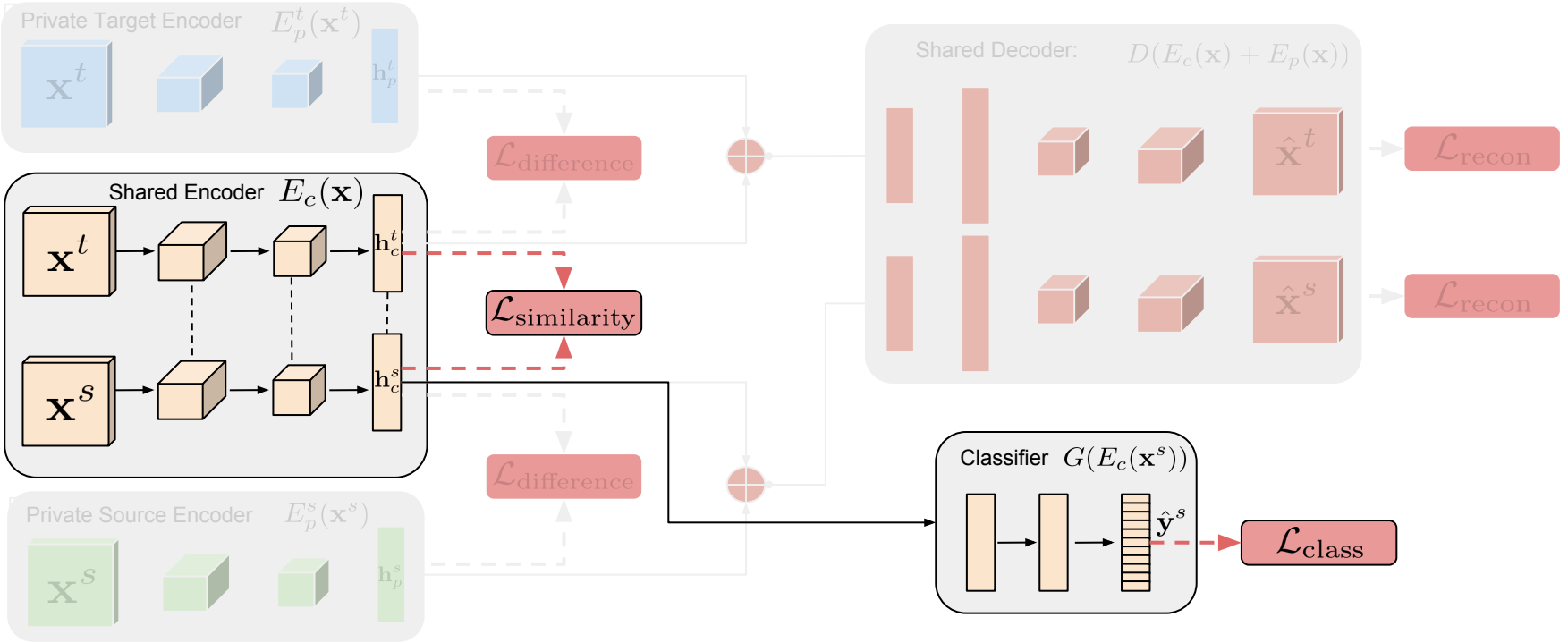
The domain adaptation process is **not interpretable**

DSN: By explicitly and jointly modeling **private and shared components** of the domain representation, we create shared **representations uncontaminated by private information**.

Domain adaptation should work better!

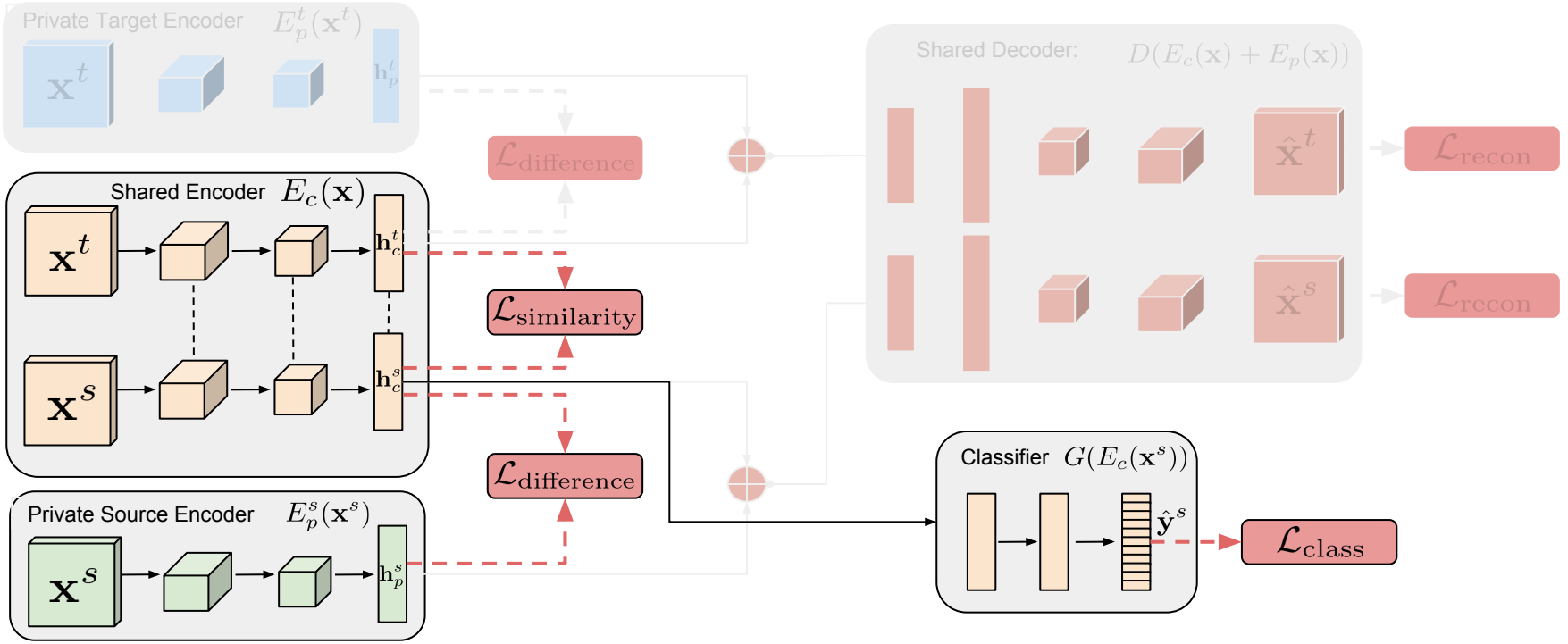
In the process, we provide interpretability to domain adaptation

Domain Separation Networks (DSNs) explained

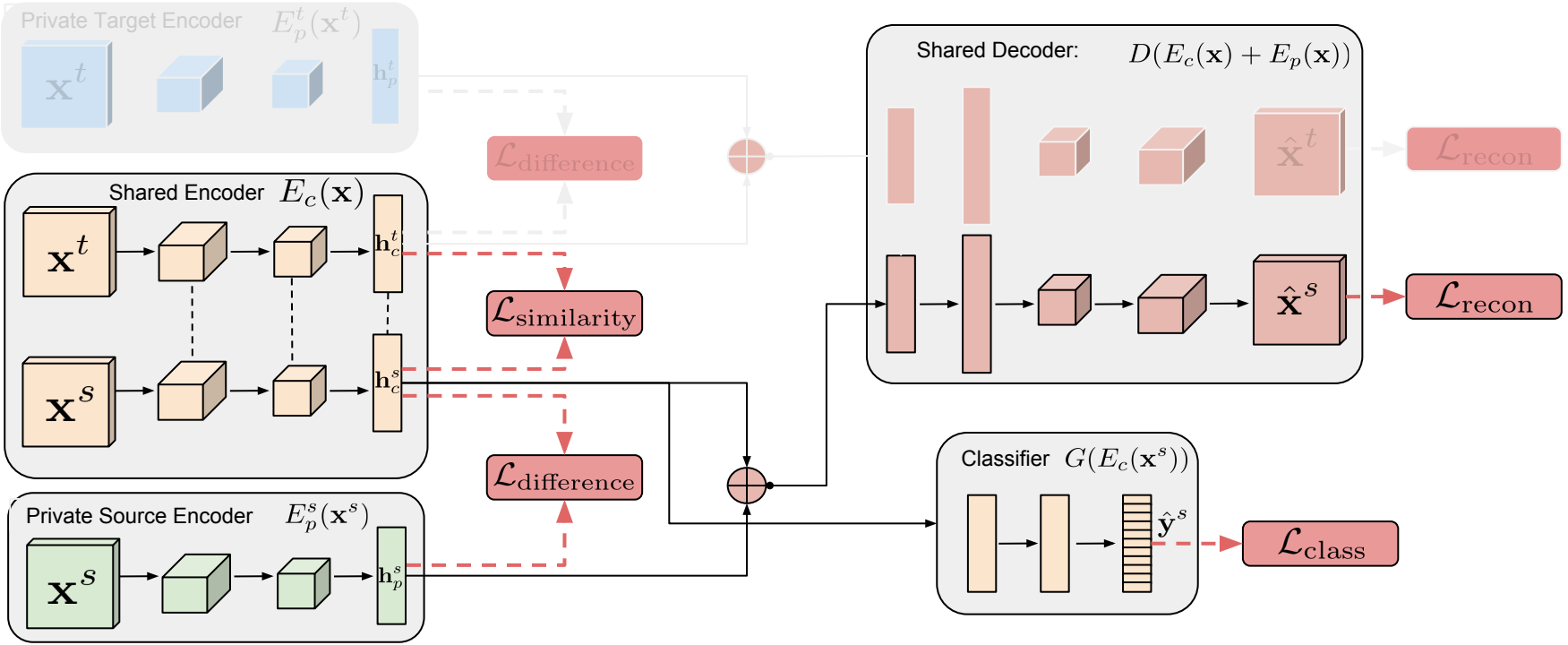


Bousmalis et al. (2016) *Domain Separation Networks.*

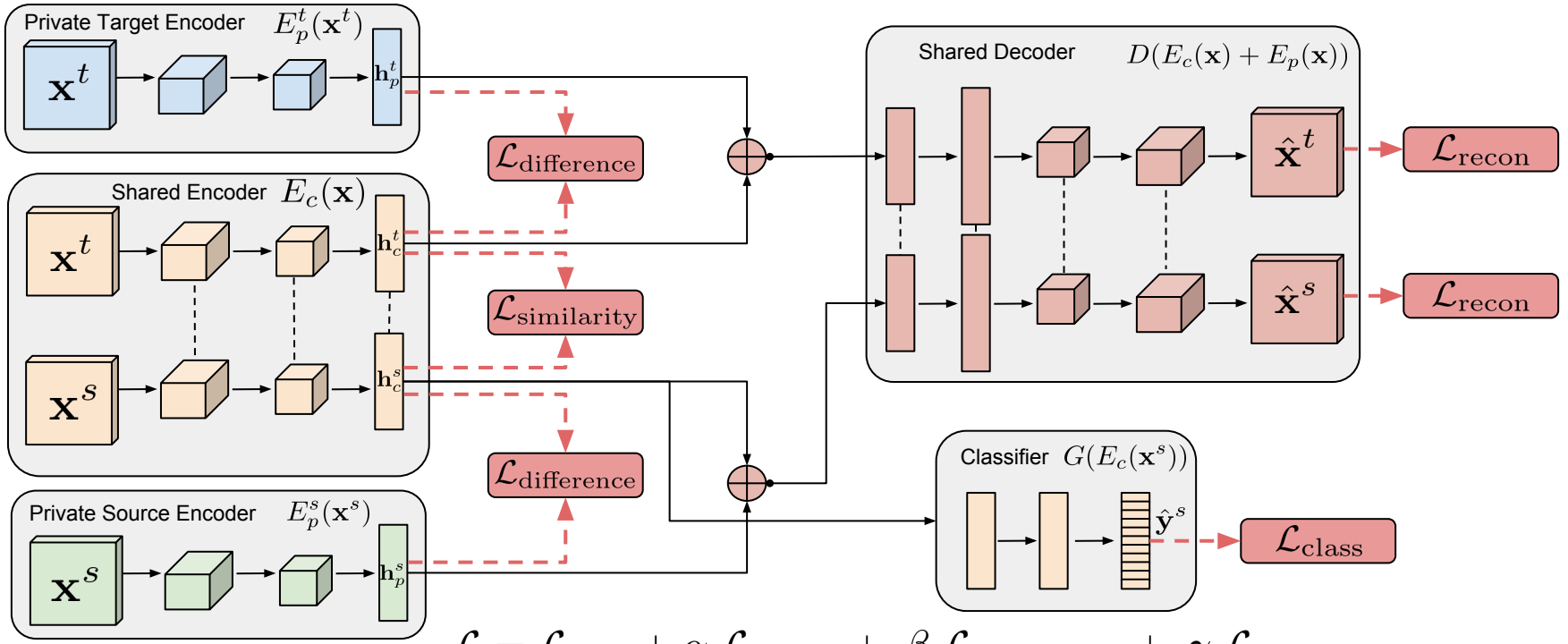
Domain Separation Networks (DSNs) explained



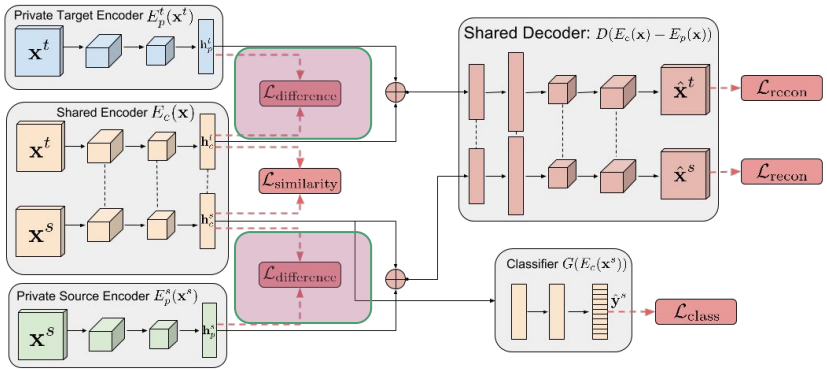
Domain Separation Networks (DSNs) explained



Domain Separation Networks (DSNs) explained



$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha\, \mathcal{L}_{\text{recon}} + \beta\, \mathcal{L}_{\text{difference}} + \gamma\, \mathcal{L}_{\text{similarity}}$$

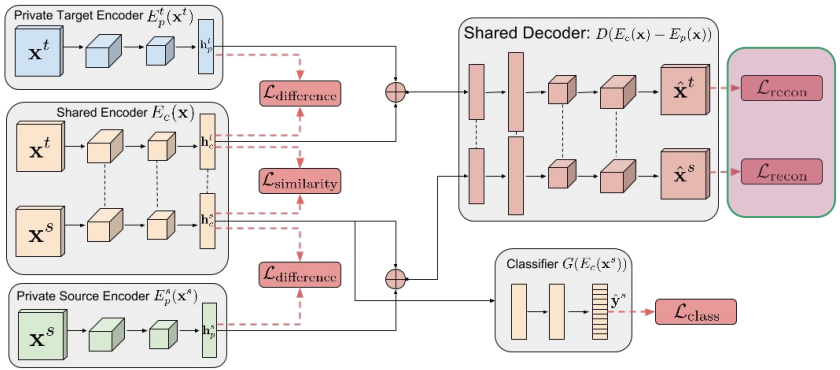


$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha\, \mathcal{L}_{\text{recon}} \; + \; \beta\, \mathcal{L}_{\text{difference}} \; + \; \gamma\, \mathcal{L}_{\text{similarity}}$$

Difference Loss

- Soft orthogonality constraints to push the private and shared subspaces to be complementary components

$$L_{\text{difference}} = \| \underbrace{H_c^{s\top}}_{\text{shared}} \overbrace{H_p^s}^{\text{private}} \|_F^2 + \left\| H_c^{t\top} H_p^t \right\|_F^2$$



$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha\, \mathcal{L}_{\text{recon}} + \beta\, \mathcal{L}_{\text{difference}} + \gamma\, \mathcal{L}_{\text{similarity}}$$

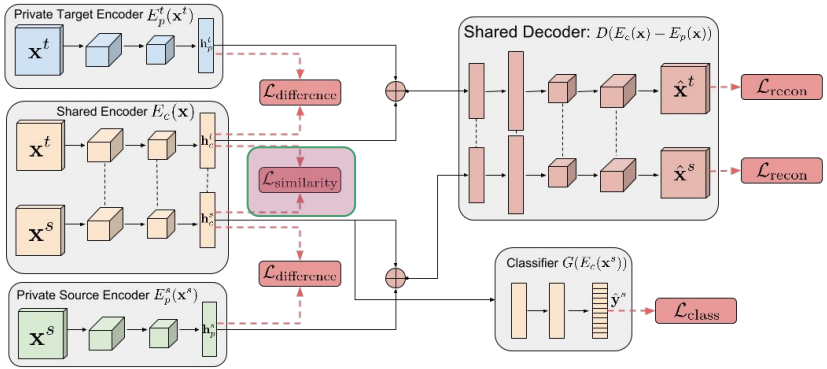
Reconstruction Loss

- We used a scale-invariant mean squared error

$$\mathcal{L}_{\text{recon}}(x, \hat{x}) = \frac{1}{k}\|x - \hat{x}\|_2^2 - \frac{1}{k^2}([x - \hat{x}] \cdot 1_k)^2$$

- Allows model to learn to reproduce the overall shape of objects without expending modeling power on absolute color or intensity.

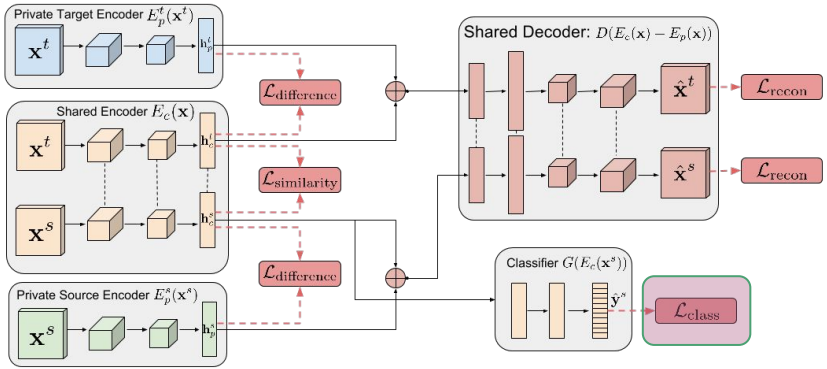
- We validated this was better than MSE experimentally



$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha\,\mathcal{L}_{\text{recon}} + \beta\,\mathcal{L}_{\text{difference}} + \gamma\,\mathcal{L}_{\text{similarity}}$$

Similarity Loss

- MMD/DANN
- Encourages the hidden representations from the shared encoder to be as similar as possible irrespective of the domain



$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha\,\mathcal{L}_{\text{recon}} \;+\; \beta\,\mathcal{L}_{\text{difference}} \;+\; \gamma\,\mathcal{L}_{\text{similarity}}$$

Task Loss

$$\mathcal{L}_{\text{task}} = \mathcal{L}_{\text{class}} = -\sum_{i=0}^{N_s} y_i^s \cdot \log \hat{y}_i^s$$

Unsupervised Domain Adaptation Experiments

Classification

MNIST to MNIST-M

SVHN to MNIST

Synthetic Digits to SVHN

Synthetic Signs to GTSRB

Classification & Pose Estimation

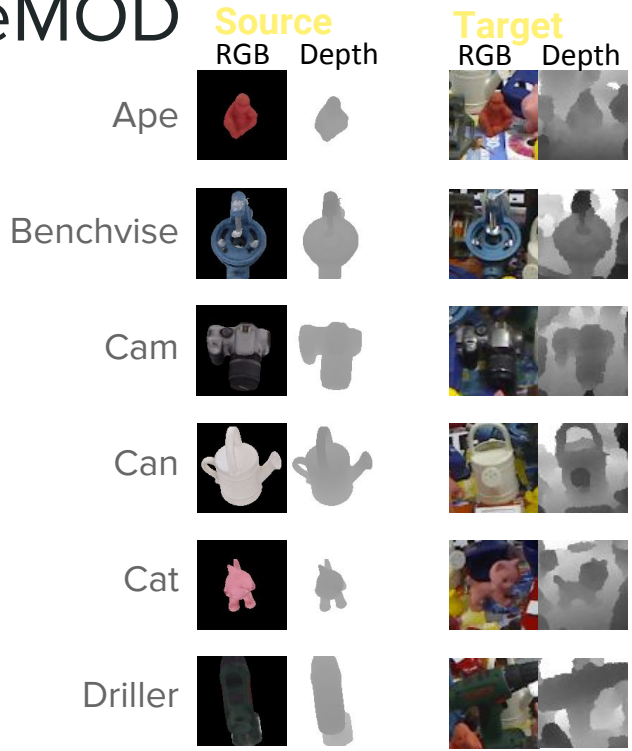
Synthetic Objects to LineMOD

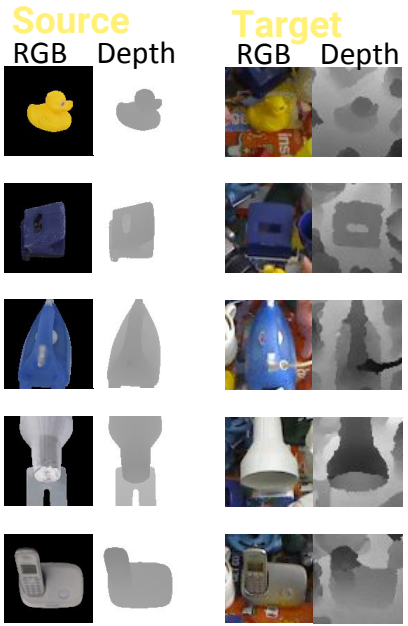
Classification Experiments

Model	MNIST to MNIST-M	Synth Digits to SVHN	SVHN to MNIST	Synth Signs to GTSRB
Source-only	56.6	86.7	59.2	85.1
CORAL	57.7	85.2	63.1	86.9
MMD	76.9	88.0	71.1	91.1
DANN	77.4	90.3	70.7	92.9
DSN w/ MMD (ours)	80.5	88.5	72.2	92.6
DSN w/ DANN (ours)	**83.2**	**91.2**	**82.7**	**93.1**
Target-only	98.7	92.4	99.5	99.8

Mean classification accuracy (%)

Synthetic Objects to LineMOD





Classification and Pose Estimation

$$\mathrm{L}_{\mathrm{task}} = \sum_{i=0}^{N_s} \{ -y_i^s \cdot \log \hat{y}_i^s + \xi \log(1 - |q^s \cdot \hat{q}^s|) \}$$

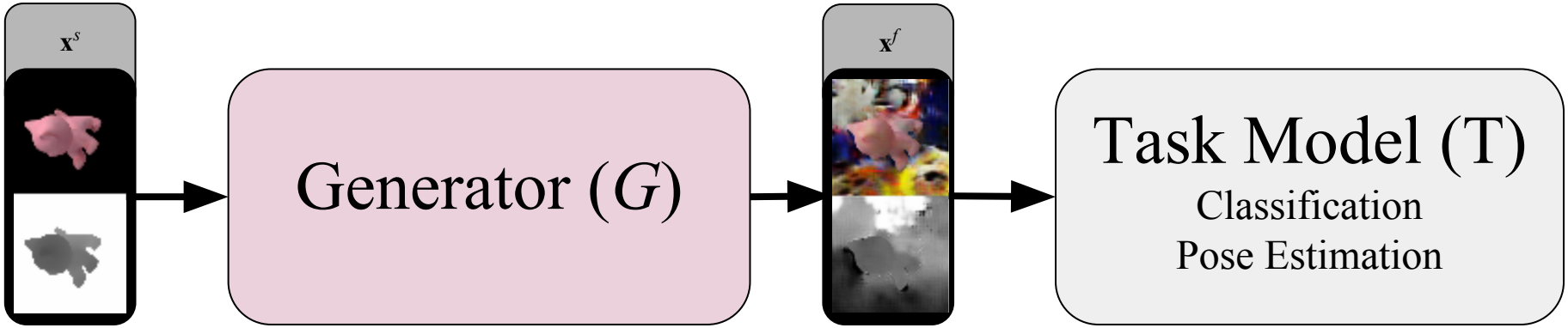
Classification loss

Pose estimation loss

Method	Classification Accuracy	Mean Angle Error
Source-only	47.33%	89.2°
MMD	72.35%	70.62°
DANN	99.90%	56.58°
DSN w/ MMD (ours)	99.72%	66.49°
DSN w/ DANN (ours)	**100.00%**	**53.27°**
Target-only	100.00%	6.47°

Pixel-level instead of **feature-level** adaptation

Image generator that, given a source image, generates a target-like image



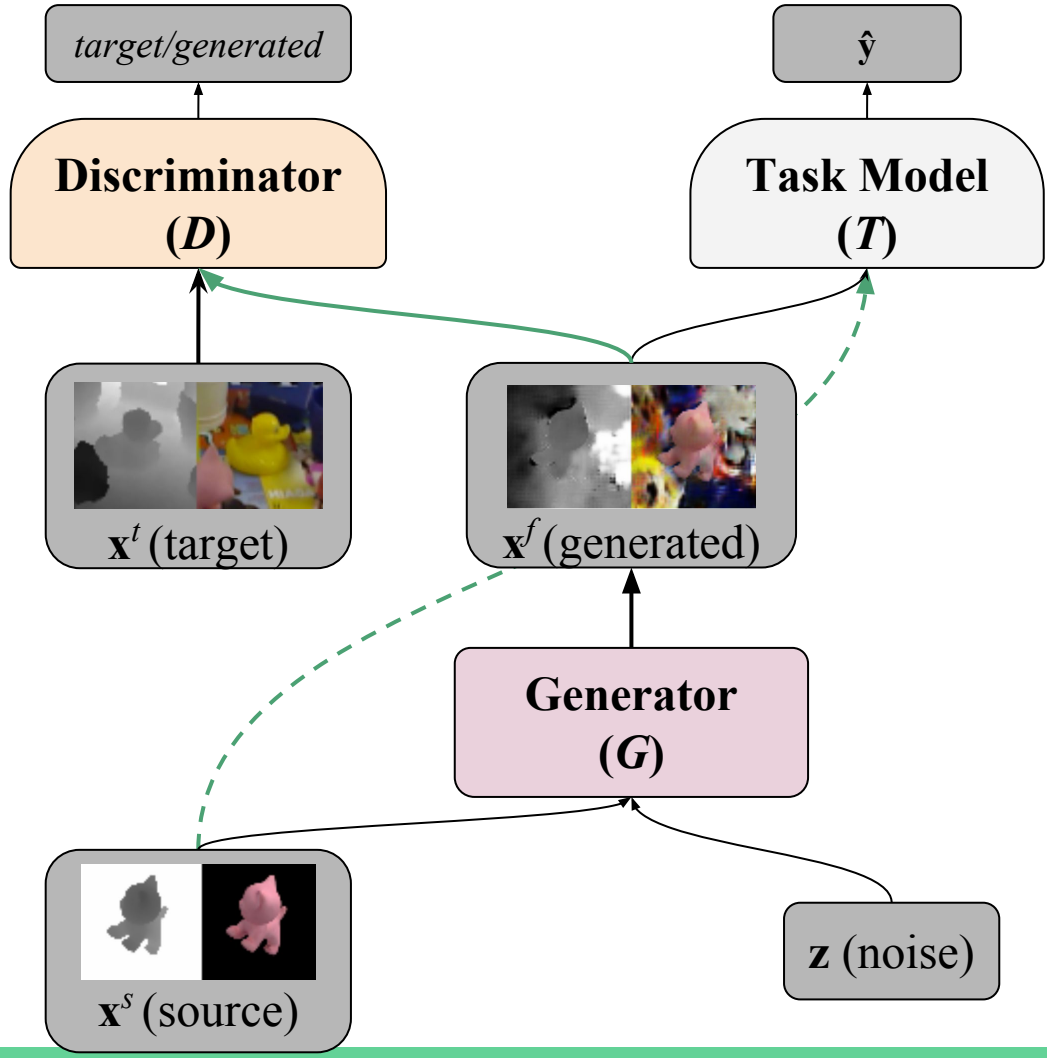
Generated images must be useful for training downstream task model

Bousmalis et al (2017) *Unsupervised pixel-level domain adaptation with generative adversarial networks*

PixelDA Model

We learn these **pixel-level transformations** from source to target domain by:

- Employing a **GAN** conditioned on both an image and a noise vector
- Keeping the **task model in the training loop** and training it on both source & generated images
- **Content-similarity** loss between source and generated images
- Both stabilize training and avoid mode shift



Why Pixel-Level Adaptation?

- Training **Stability**
  - Adaptation processes that rely on adversarial training are sensitive to random initialization
  - By adding a task-specific loss and a content-similarity loss we regularize the process, avoid mode shift, and we show that we improve repeatability
- Data **Augmentation**
  - By conditioning our model to a stochastic vector as well as a source image, we are able to generate a large number of stochastic samples that appear similar to the target domain
- **Potential decoupling** from the Task-Specific Architecture
  - In most unsupervised domain-adaptation methods one cannot switch the task model without relearning the entire domain adaptation process

Results

Digit Classification (MNIST to MNIST-M)

MNIST

Transferred

MNIST-M Neighbors

Model	MNIST to MNIST-M
Source Only	63.6
CORAL [41]	57.7
MMD [45, 31]	76.9
DANN [14]	77.4
DSN [5]	83.2
CoGAN [30]	62.0
Our model	**98.2**
Target-only	96.4

- Our generator learns not only to insert backgrounds, but also the inversion process.
- Our generator does not memorize images from the target domain.
- PixelDA outperforms not only other feature-level models, but also the “Target-only” model!

Results

Classification and 3D Pose Estimation (Synthetic to Real Cropped LineMod)

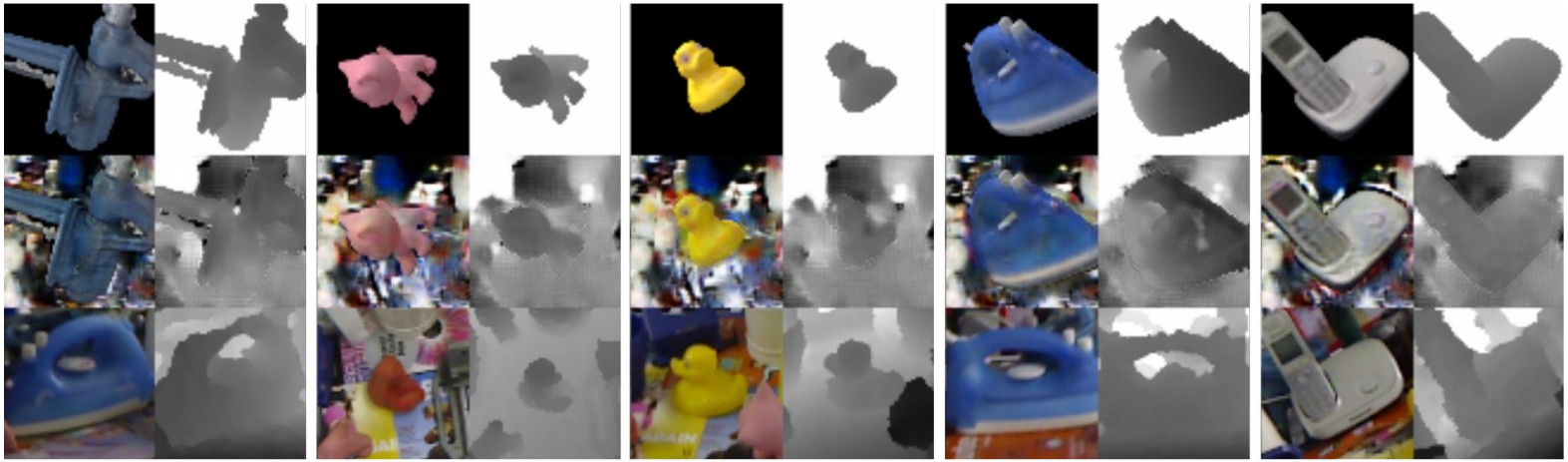
- We measure performance with classification accuracy and mean angle error
- Mean angle error is the angle the object would need to rotate around a fixed axis to move from the predicted to the ground truth pose.

Model	Classification Accuracy	Mean Angle Error
Source-only	47.33%	89.2°
MMD [45, 31]	72.35%	70.62°
DANN [14]	99.90%	56.58°
DSN [5]	**100.00%**	53.27°
Our model	99.98%	**23.5°**
Target-only	100.00%	6.47°

Synthetic

Transferred

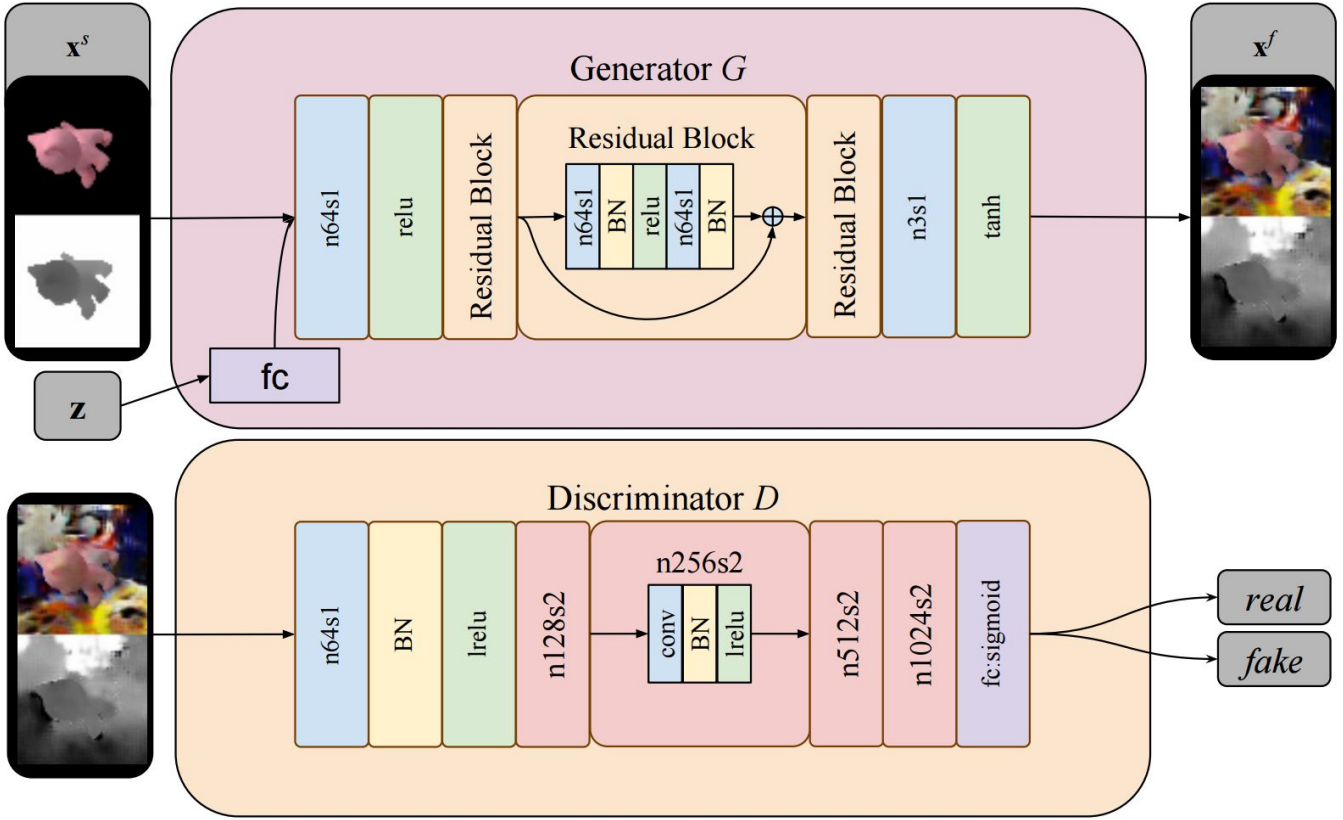
Nearest Neighbors



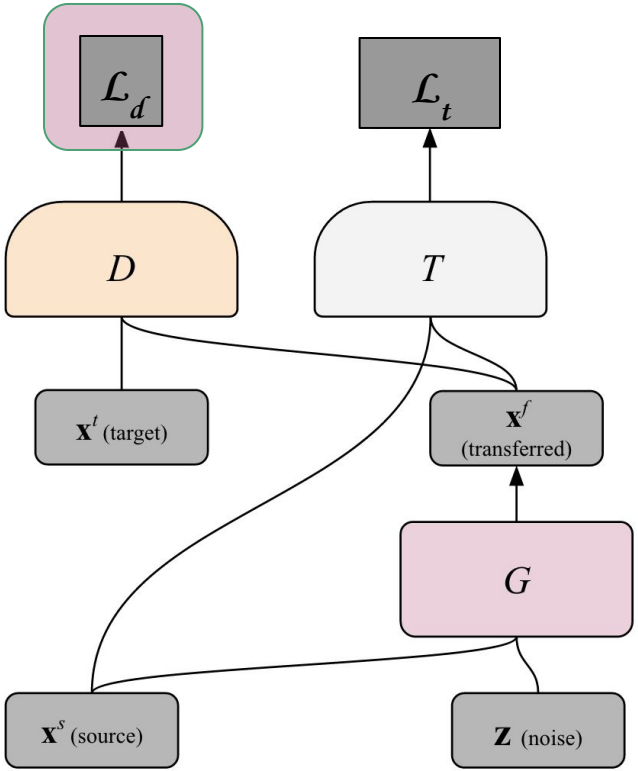
Concurrent Related Work:

- Style transfer: Gatys et. al., 2015, Johnson et. al. 2016
  - Uses back-propagation to create new image with the “style” on one and the “content” of another
  - Deals with pairs of images rather than a corpus
- GANs conditioned on images: Yoo et. al., 2016, Ledig et. al, 2016
  - Generate new images by conditioning on input images
  - Do not condition on noise to allow generation of many samples
  - Do not apply to unsupervised domain adaptation
- CoGAN: Liu and Tuzel, 2016
  - Pair of GANs, for domain adaptation; one GAN for each domain
  - Conditioned only on noise: hard to train to generate good samples in both domains
- SimGAN: Shrivastava et. al, Apple, 2016
  - Concurrent similar work

Architecture Expanded View



Losses

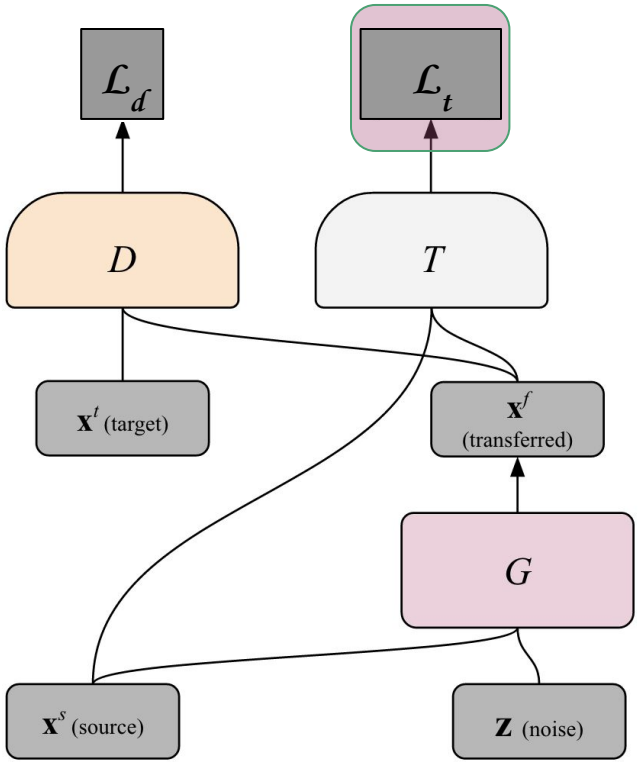


- **Domain loss**: “Standard” GAN loss
- G conditioned on source image *and* noise

$$\mathcal{L}_d(D, G) = \mathbb{E}_{\mathbf{x}^t}[\log D(\mathbf{x}^t; \boldsymbol{\theta}_D)] +$$

$$\mathbb{E}_{\mathbf{x}^s, \mathbf{z}}[\log(1 - D(G(\mathbf{x}^s, \mathbf{z}; \boldsymbol{\theta}_G); \boldsymbol{\theta}_D))]$$

Losses

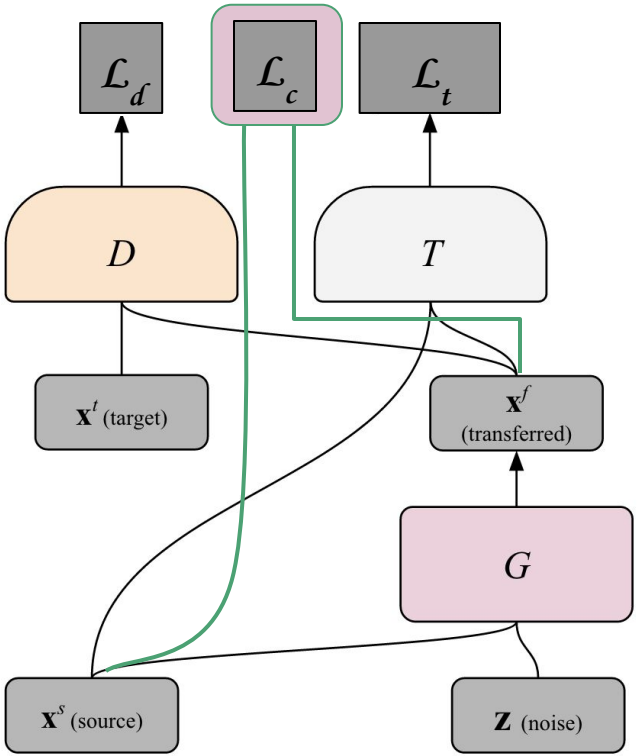


- For classification, cross-entropy loss:

$$\mathcal{L}_t(G,T) = \mathbb{E}_{\mathbf{x}^s,\mathbf{y}^s,\mathbf{z}}\big[-\mathbf{y}^{\mathbf{s}\top}\log T\left(G(\mathbf{x}^s,\mathbf{z};\boldsymbol{\theta}_G);\boldsymbol{\theta}_T\right) - \mathbf{y}^{\mathbf{s}\top}\log T(\mathbf{x}^s);\boldsymbol{\theta}_T\big]$$

  - Uses source and transferred images, source labels
  - Greatly improves repeatability (reducing performance variance across runs)
  - Parameters $\boldsymbol{\theta}_T$ updated with fixed $\boldsymbol{\theta}_G$

Losses



- **Content Similarity Loss** (optional):

$$\mathcal{L}_c(G) = \mathbb{E}_{\mathbf{x}^s,\mathbf{z}}\Big[\frac{1}{k}\left\|(\mathbf{x}^s - G(\mathbf{x}^s,\mathbf{z};\boldsymbol{\theta}_G))\circ\mathbf{m}\right\|_2^2 - \frac{1}{k^2}\left((\mathbf{x}^s - G(\mathbf{x}^s,\mathbf{z};\boldsymbol{\theta}_G))^\top\mathbf{m}\right)^2\Big]$$

  - Pairwise Mean Squared Error
  - Specific loss to help with pixel adaptation process
  - Assumption: foreground should change little and consistently
  - Foreground mask (available in rendered data)

Loss Function

$$\min_{\boldsymbol{\theta}_G, \boldsymbol{\theta}_T} \max_{\boldsymbol{\theta}_D} \alpha \mathcal{L}_d(D, G) + \beta \mathcal{L}_t(T, G) + \gamma \mathcal{L}_c(G)$$

Domain Loss (GAN)

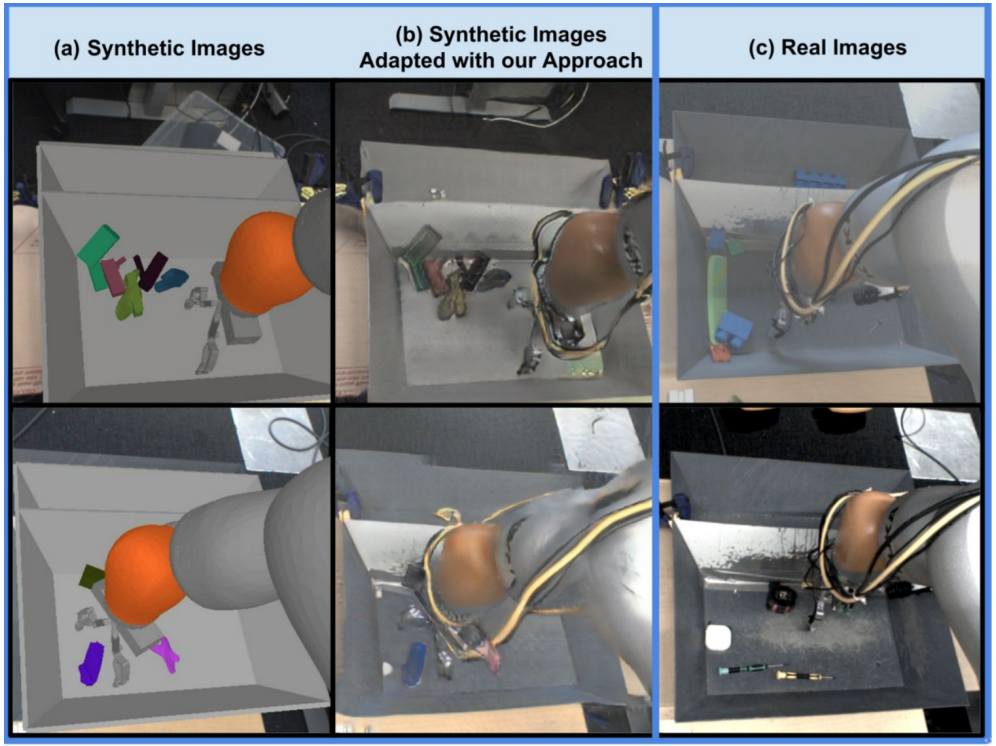
Task-specific

Content Similarity

- **Domain loss:** adversarial loss that trains generator/discriminator pair
- **Task-specific loss:** Uses source labels to avoid mode shift and stabilizes GAN training
- **Content similarity loss:** Penalizes changes in foreground pixels more than changes in background

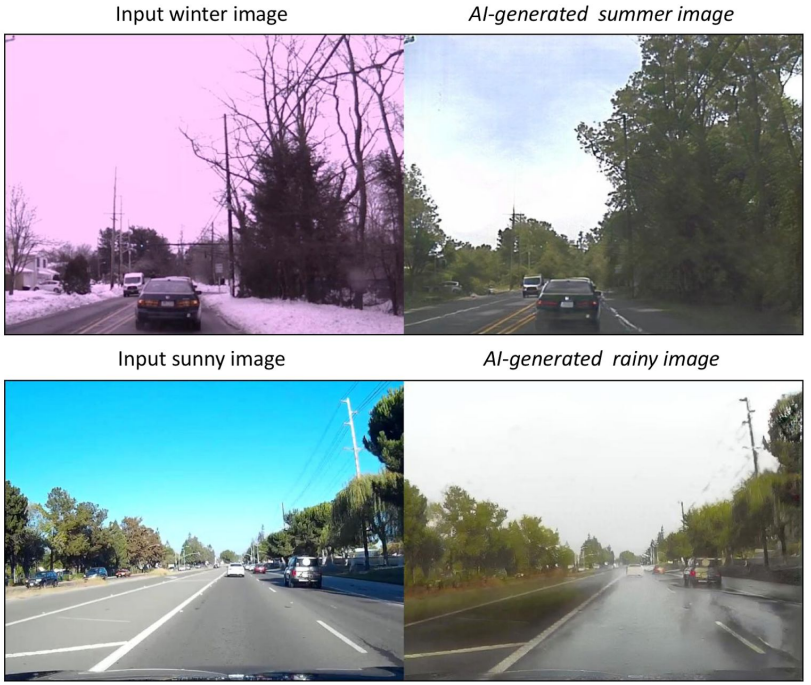
GraspGAN (Bousmalis et al. 2017)

- Sim2real taken to the next level
- Train to grasp objects in simulation
- Learn to grasp them IRL **50x faster** due to domain adaptation



Summary

- Many ways for tackling unsupervised domain adaptation
- Pixel-level the “coolest” looking (CycleGAN et al)
- Potentially very useful for the case of synthetic to real domain adaptation,
  where low-level changes are needed.
- But not quite as “solved” in the same way that detection is!



Domain adaptation: food for thought

- Using extensive simulation data for domain adaptation is clearly a clever idea.
- PixelDA, CycleGAN, GraspGAN et al. are (surprisingly!) good at transfer.
- But is using a simulator the only way to learn useful policies for robots?
  - It's not always possible to build a good simulator.
- We know from ArmFarm that having the arm just repeatedly interact hundreds of thousands of times "works".
- Could we
  - dispense of the simulator altogether
  - learn a **model** of the that allows answering "what will happen if I take this action?"
  - use that model to learn that solves the task ("grasping")?
- We hope the answer is "yes" :)

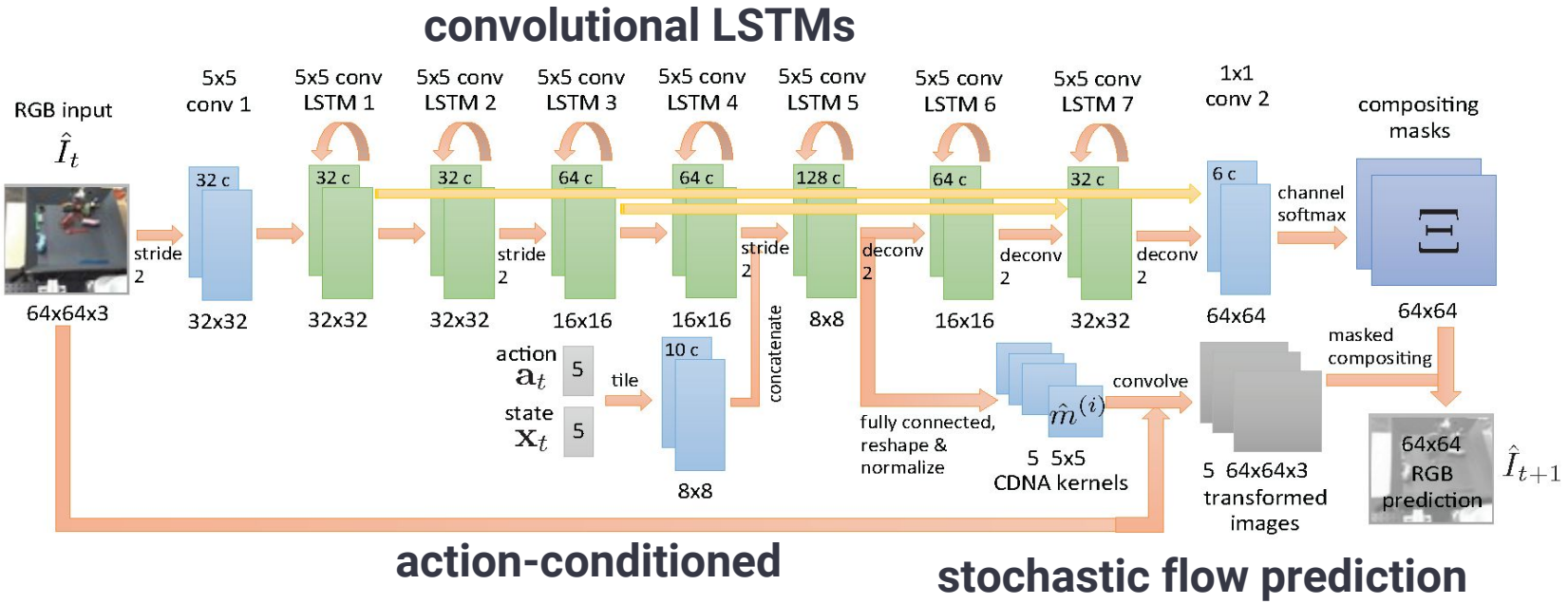
Predictive learning

- Yann LeCun calls this “predictive learning”
  - Also “self-supervised learning”
- (Supposedly) humans and other animals do this:
  - They predict the effect of their actions
  - e.g. “try it in your head” before acting.
- What if we had a way to do that directly? Let’s say we observe:
  - the present: a sequence of events (frames), call it **state**
  - the **action** that the agent takes (“move arm to coordinate (x,y,z)”)
  - the **future state** once the action complets.
- We could learn a model that tries to infer $\mathbf{s}_{t+1}$ from $[\mathbf{s}_t, \mathbf{a}_t]$
- In general, this is hard: state can be a video, action can be continuous multi-dimensional, future is non-determistic etc.

Predictive learning

- The state being a video is a good thing as well!
  - Plenty of training data
  - The state at t+1 is a rich, dense predictive target.
- Assuming we can learn such a model, multi-task learning becomes feasible
- We'll talk about:
  - how to learn it, how to deal with non-determinism, hierarchical versions, Visual MPC.
- So how can we learn such a model? Cast it a supervised learning problem of course!
- A popular approach: convolutional LSTM
  - A recurrent model tailored for sequences of image frames.

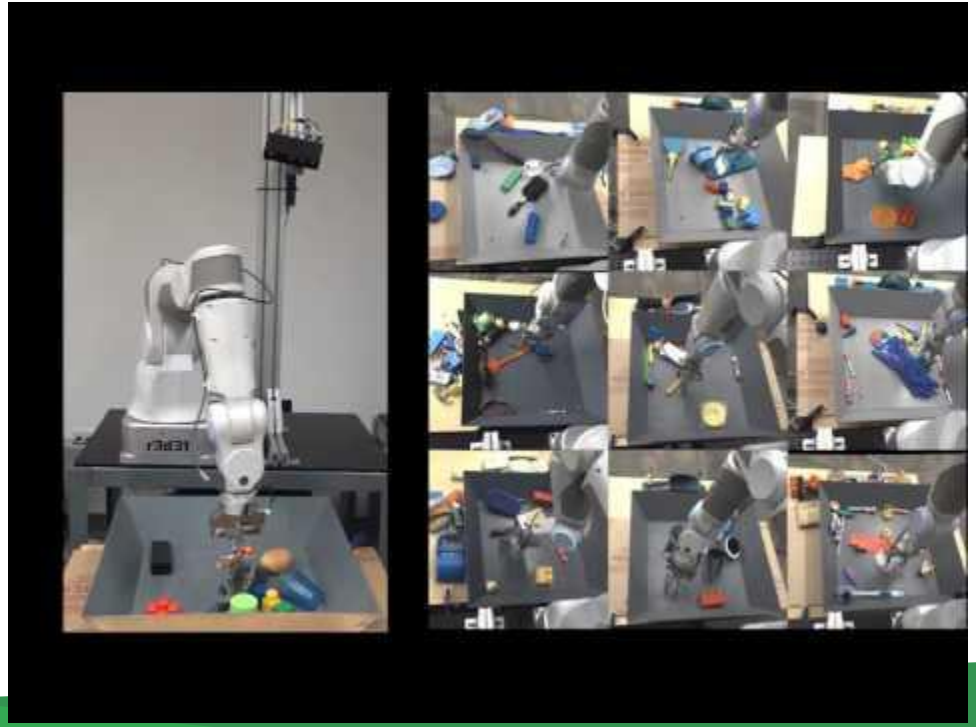
Finn et al. (2016) model

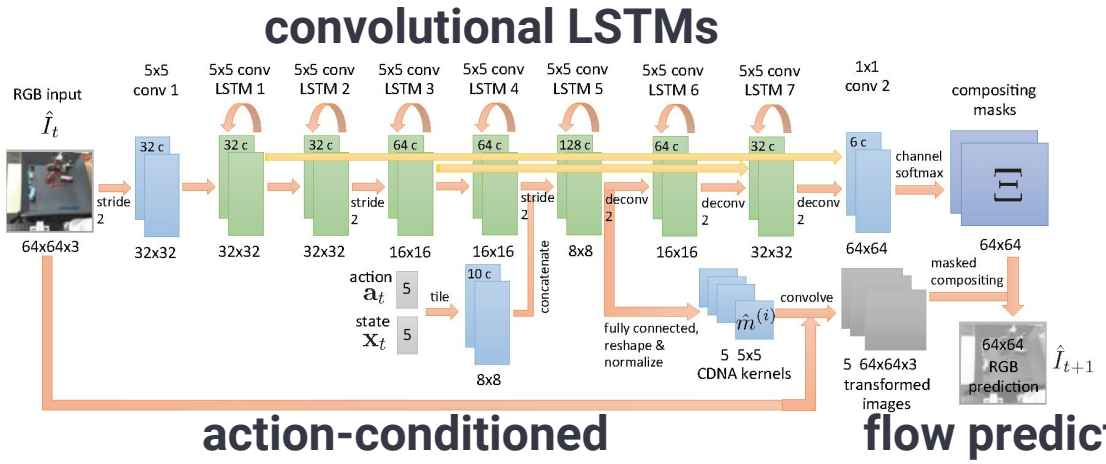


Unsupervised Learning for Physical Interaction through Video Prediction by Finn et al. (2016)

Data Collection

Program & record random pushing motions:





One big problem: model makes mostly deterministic predictions!

Stochastic Video Prediction

Given Frames

f_0

f_1

Video Prediction

Given Frames

Future Frames

f_0 f_1 $\hat{f}_2$ $\hat{f}_3$ $\hat{f}_4$ $\hat{f}_5$ $\hat{f}_6$ $\hat{f}_7$ $\hat{f}_8$ $\hat{f}_{\ldots}$

Video Prediction (Action Conditioned)

Given Frames

Future Frames

f_0 f_1 $\hat{f}_2$ $\hat{f}_3$ $\hat{f}_4$ $\hat{f}_5$ $\hat{f}_6$ $\hat{f}_7$ $\hat{f}_8$ $\hat{f}_{...}$

a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8

Problem: Stochasticity

Given Frames

f_0 f_1

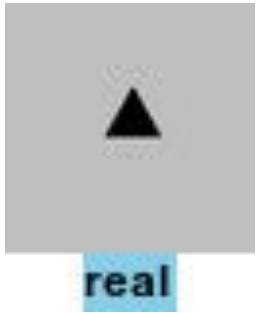
Future Frames

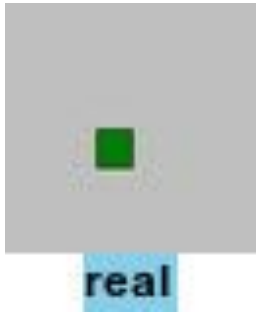
Possibility 1

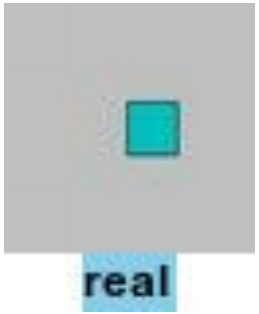
Possibility 2

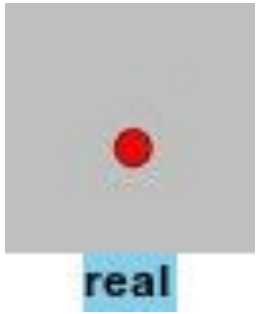
Possibility N

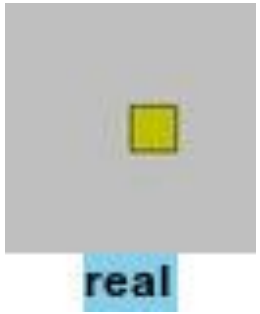
Stochastic Shapes Dataset





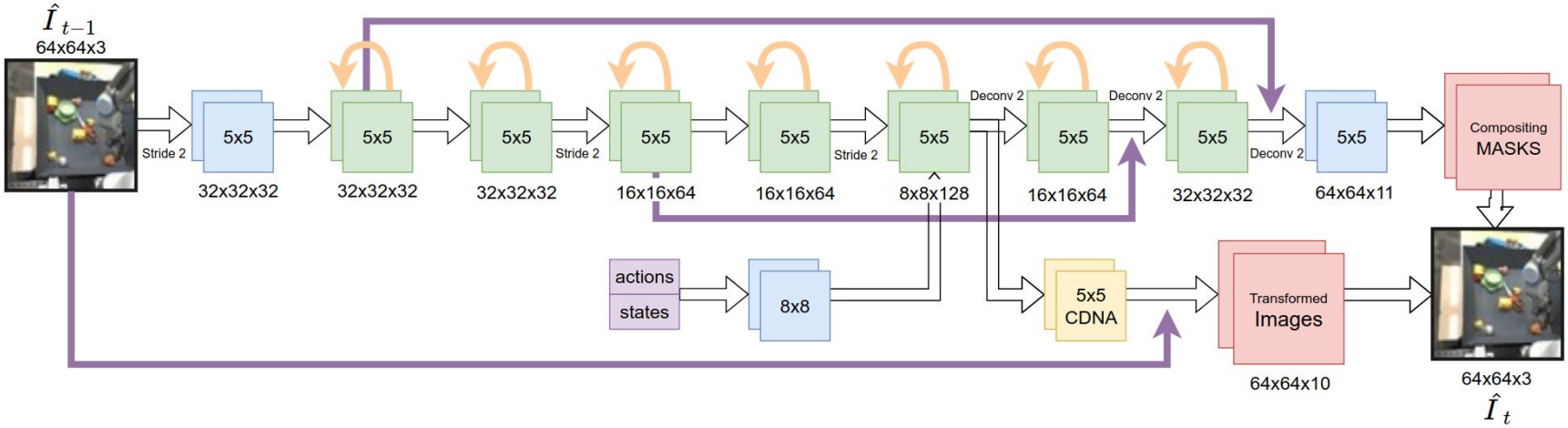




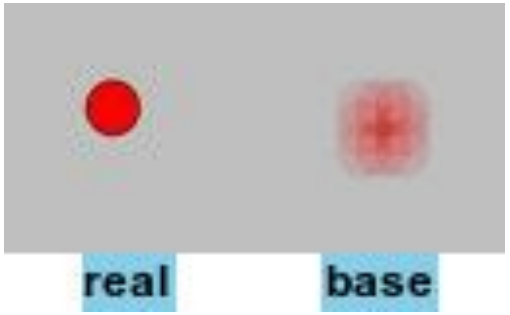


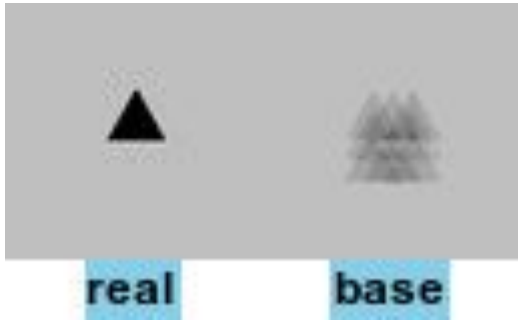
Base Model (Deterministic)

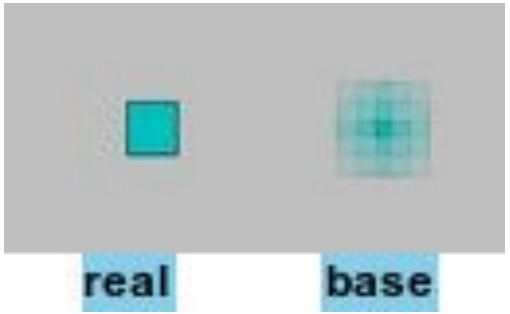
Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction." NIPS 2016

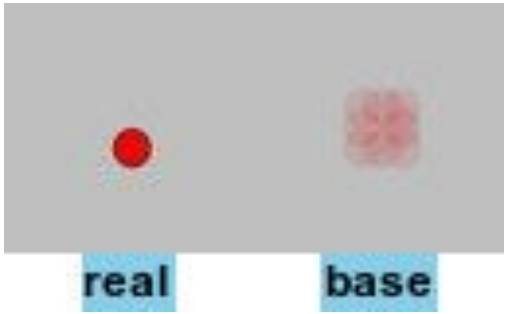


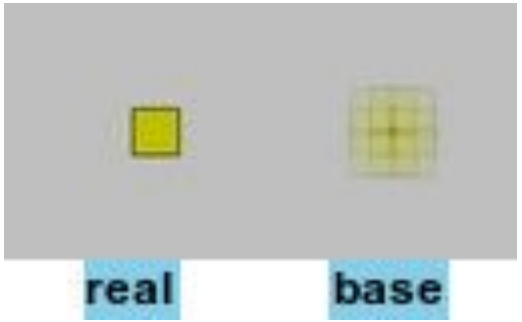
Oops: probably not very useful







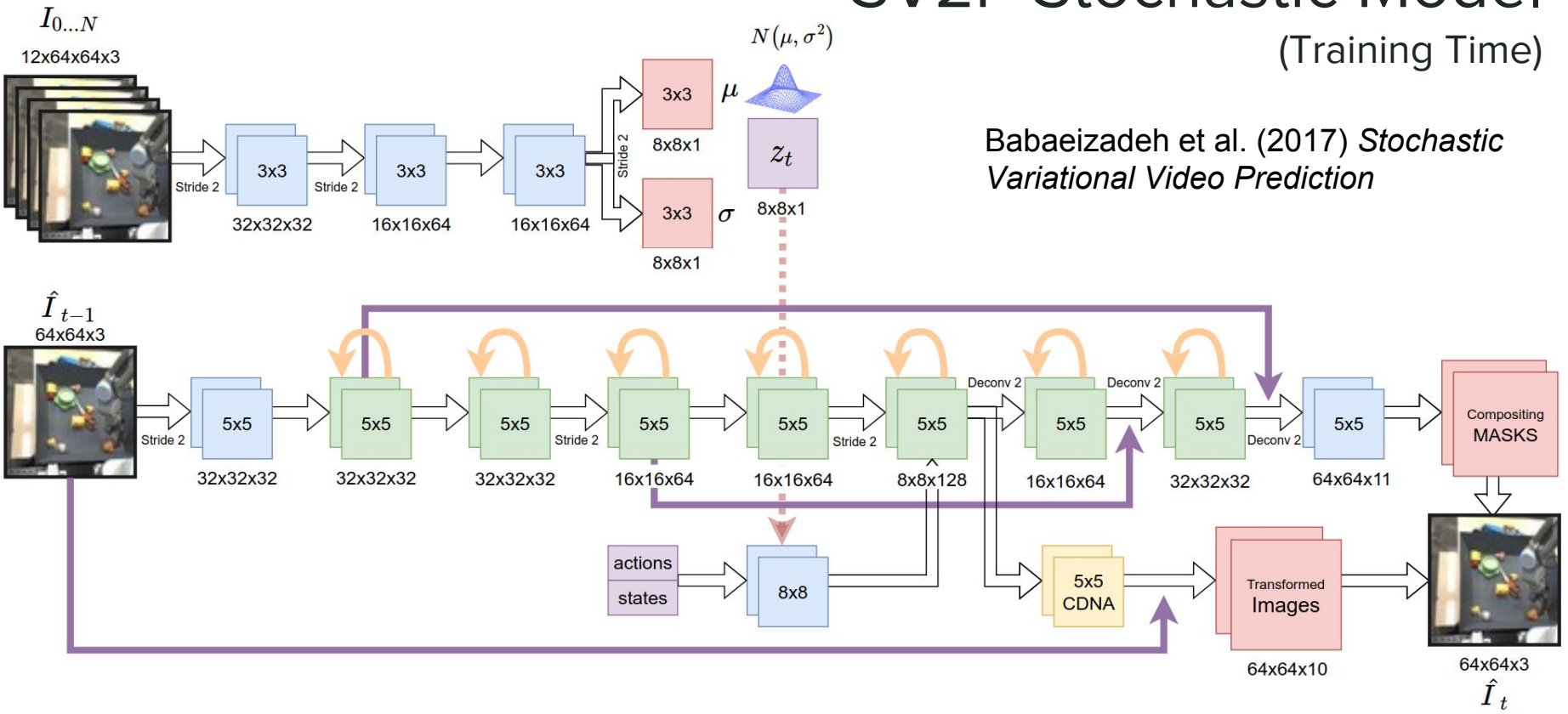




SV2P Stochastic Model

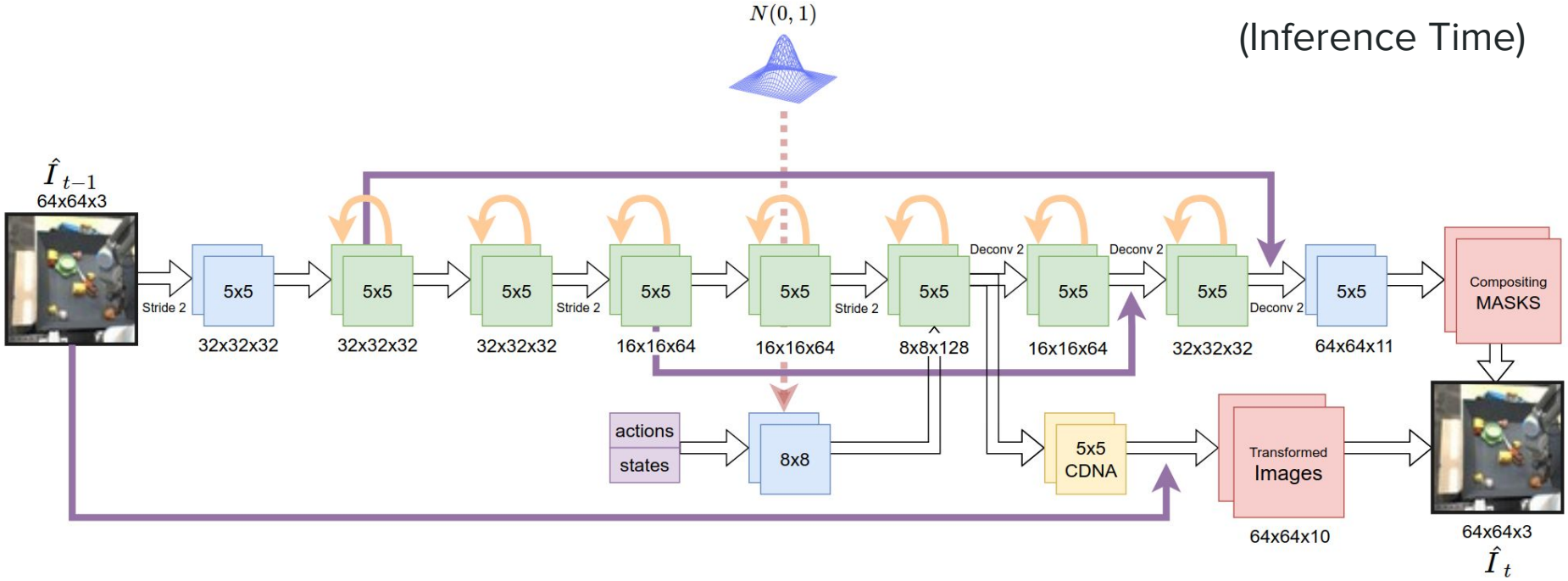
(Training Time)

Babaeizadeh et al. (2017) *Stochastic Variational Video Prediction*

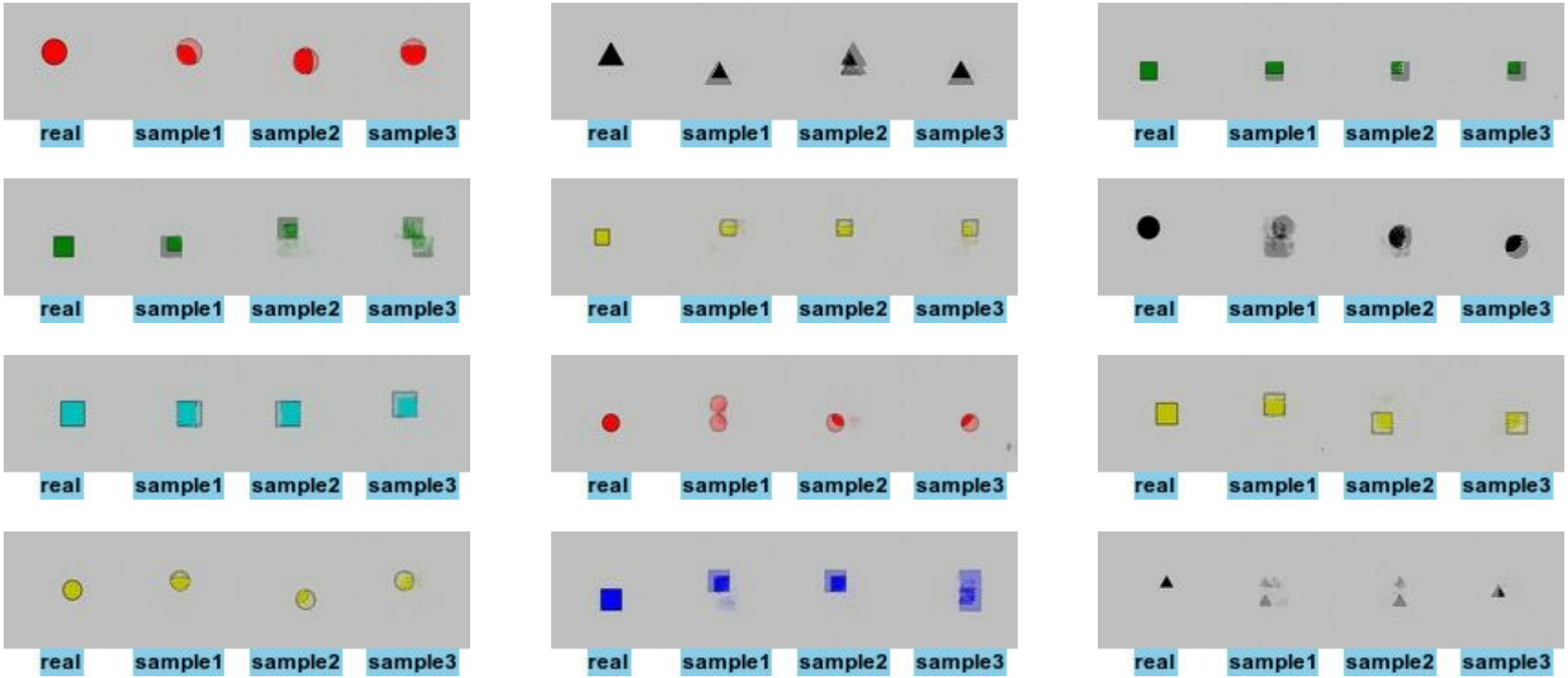


SV2P Stochastic Model

(Inference Time)

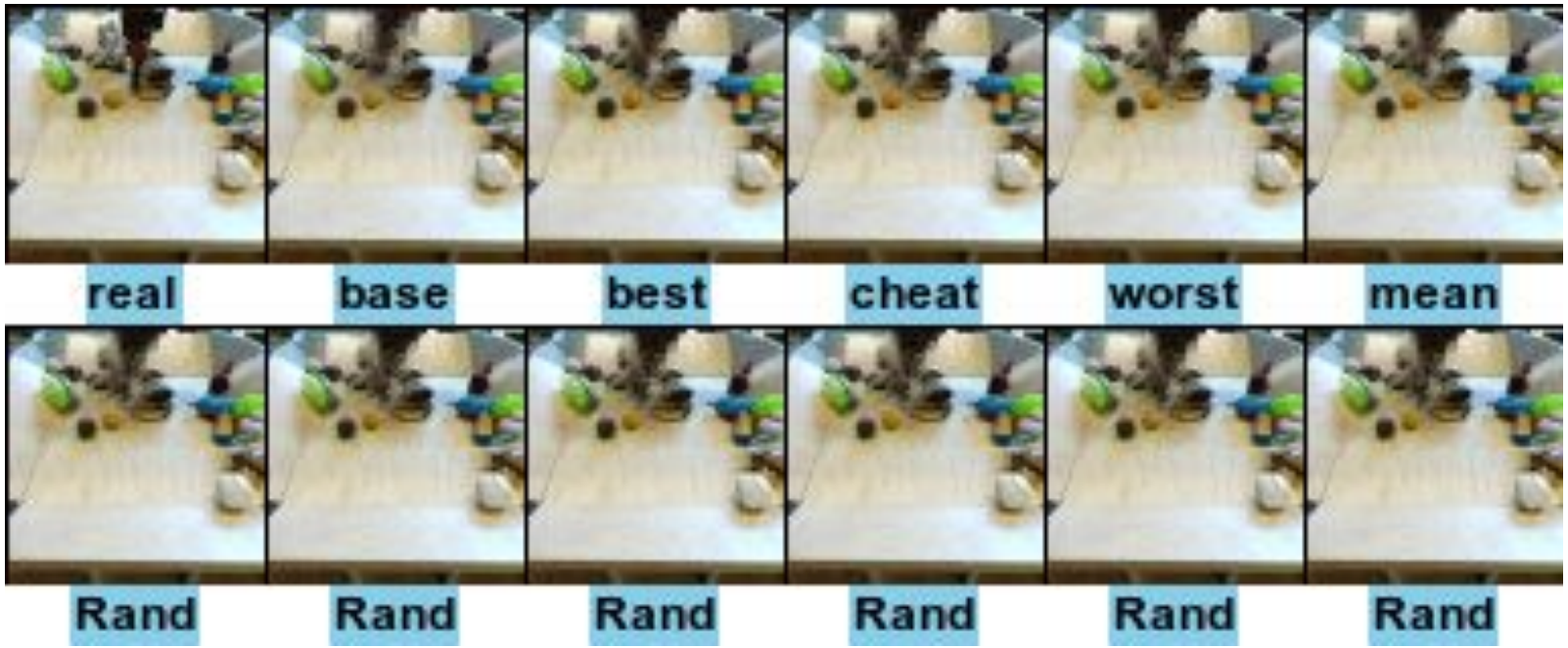


Random Samples



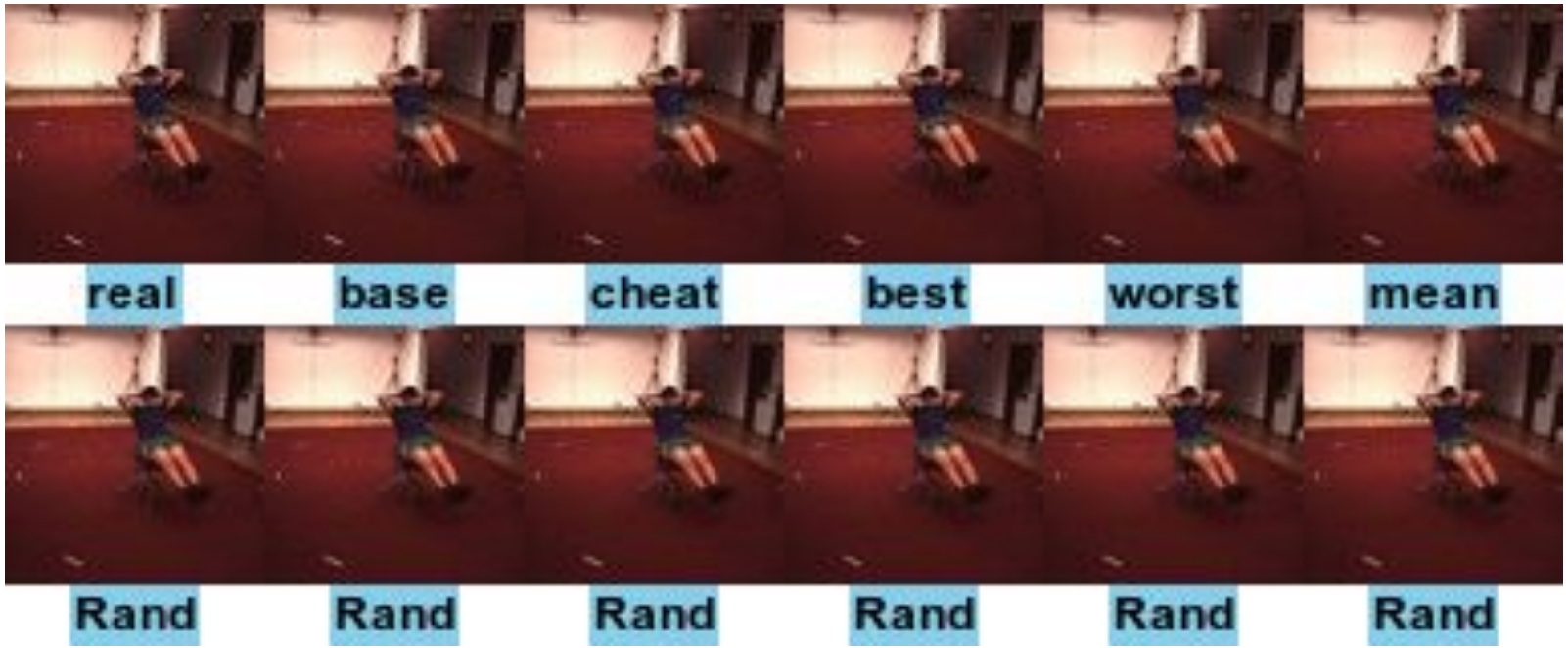
Berkeley Dataset (action conditioned)

Berkeley Dataset (no action)

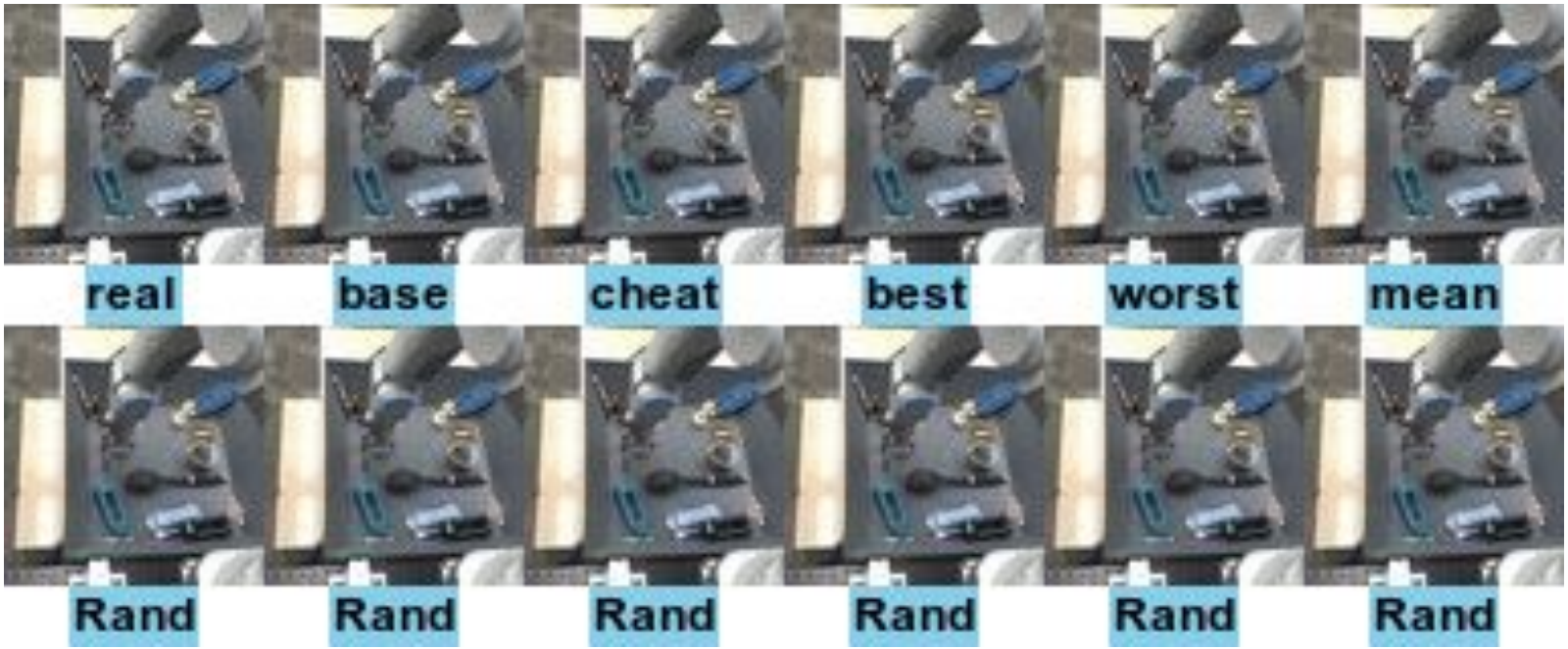


Human3.6M (no action)

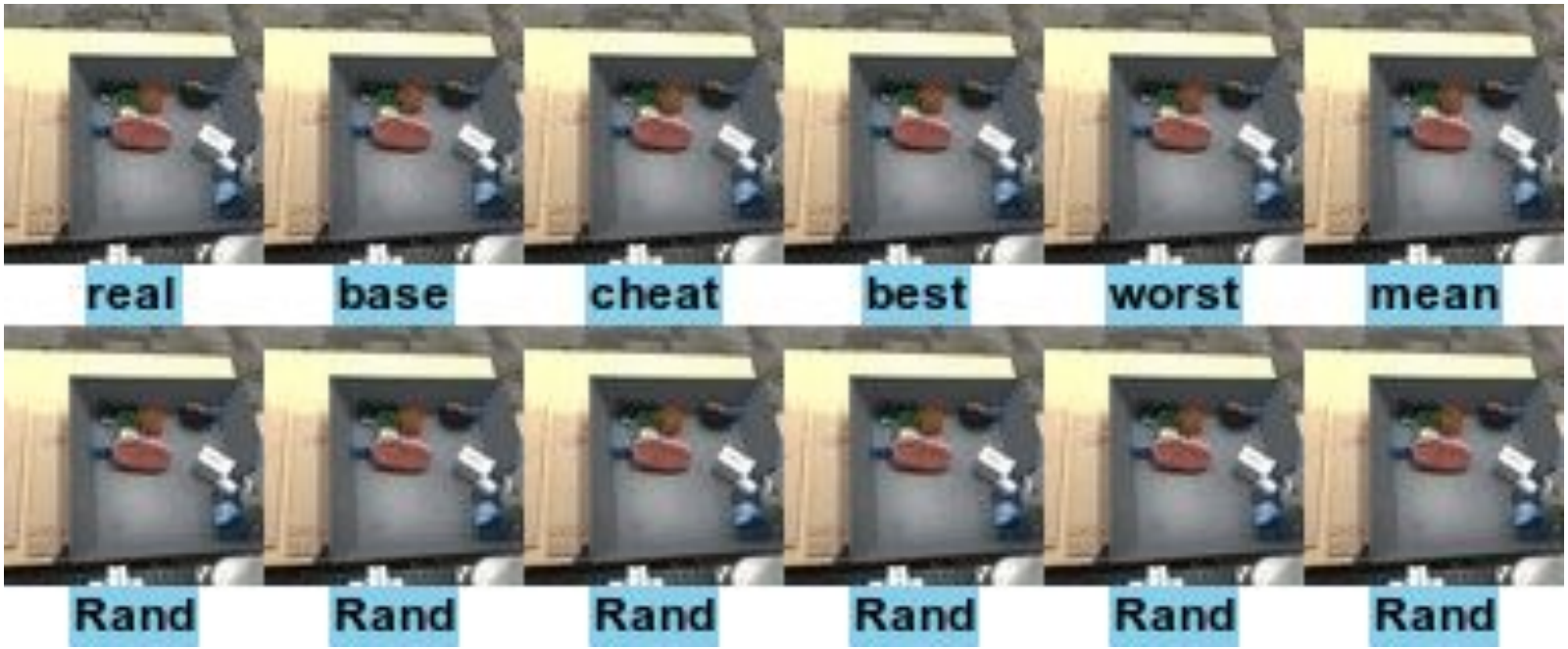
Human3.6M (no action)

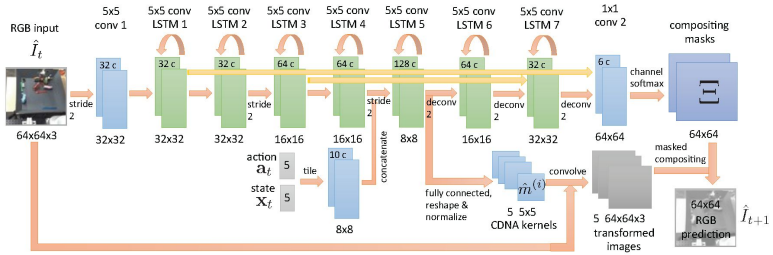


Pushing Dataset (action conditioned)



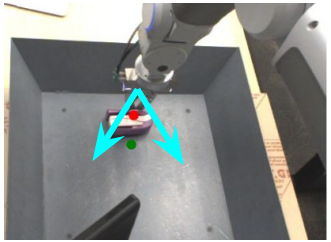
Pushing Dataset (action conditioned)





output is the mean of a probability distribution over pixel motion predictions

How to make video prediction useful?



We can **predict how pixels will move based on the robot's actions**.

Next few slides from Chelsea Finn
Deep Visual Foresight for Planning Robot Motion

Planning Actions with Visual Foresight

1. Sample N possible action sequences.
2. Predict the future for each action sequence
3. Pick best future & execute corresponding action
4. Repeat 1--3 to replan in real time.

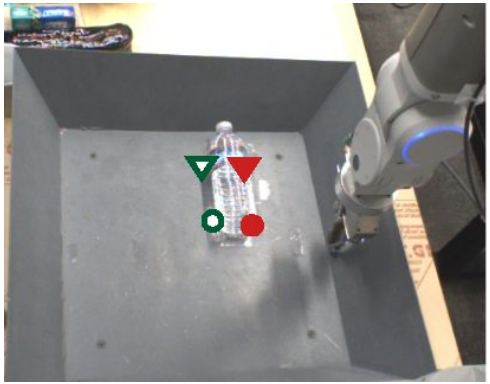
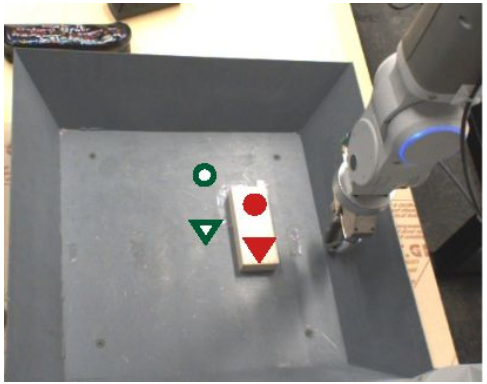
Roboticists call this “model-predictive control” (MPC).

current state

2 predicted futures

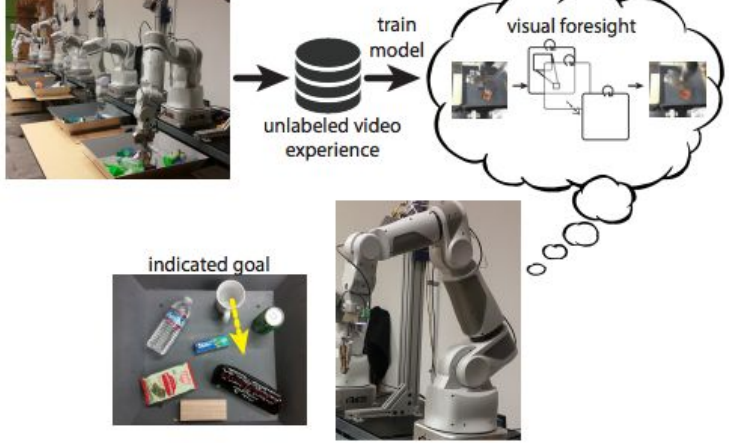
Which future is the best one?

- specify goal by selecting where pixels should move to


- select future with maximal probability of pixels reaching their respective goals

Takeaways of Visual MPC



Benefits of this approach

- Learn for a **wide variety of tasks**
- **Scalable** -- requires minimal human involvement (no cost function, labels, image of goal, instrumented training set-up)
- A good way to **evaluate video prediction models**

Limitations

- Can’t [yet] learn complex policies, like model-free methods
- Compute-intensive at test-time.
- Some planning methods susceptible to **adversarial examples**

Future directions

This is just the beginning...

Can we design the **right** model?

- stochastic?
- longer sequences?
- hierarchical?
- deeper?

Can we handle **long-term** planning?

Summary

- Despite amazing progress & hype, computer vision not really “solved”.
- Anything that deviates from ImageNet template is hard.
- Intelligent agents resilient to domain shifts: crucial
- **Sim2real** is one potential key to “robots that learn”
- Another key is building **predictive agents**
- Open questions:
  - Can we deal with uncertainty?
  - How to do efficient planning under uncertainty?

Thanks!

- Many awesome collaborators/interns/residents: Konstantinos Bousmalis, Sergey Levine, Chelsea Finn, Mohammad Babaeizadeh, Drago Anguelov, Christian Szegedy and many many more at Google Brain/DeepMind/Berkeley.

- Questions? Now, in person or at **dumitru@google.com**