Simulations (Part2): In Action

Earl Wong

Simulations in Reinforcement Learning

- For reinforcement learning problems involving robots, quad rotors, autonomous vehicles, etc., the cost of collecting real vision / perception based data is high.
- Hence, simulated data is a promising substitute.
- Although simulation data is inexpensive, models trained on simulated data may or may not, perform well on real data.
- Two areas of active investigation include domain randomization and domain adaptation.

Domain Randomization

- Domain randomization is defined as a technique for training models on simulated images that have undergone various randomizations.

- The simulation trained models then need to generalize to real images in the real world.

- Two recent applications include object detection (a high precision task) and collision avoidance (a lower precision task).

- In object detection, randomization occurred via: 1) The number and shape of distractor objects, 2) The position and texture of the objects, 3) The position, orientation and field of view of the camera and 4) The position, orientation, number and specular characteristics of the lights.

- In collision avoidance, randomization occurred via: 1) The generation of random furniture placement, 2) The associated textures in the image and 3) The lighting.

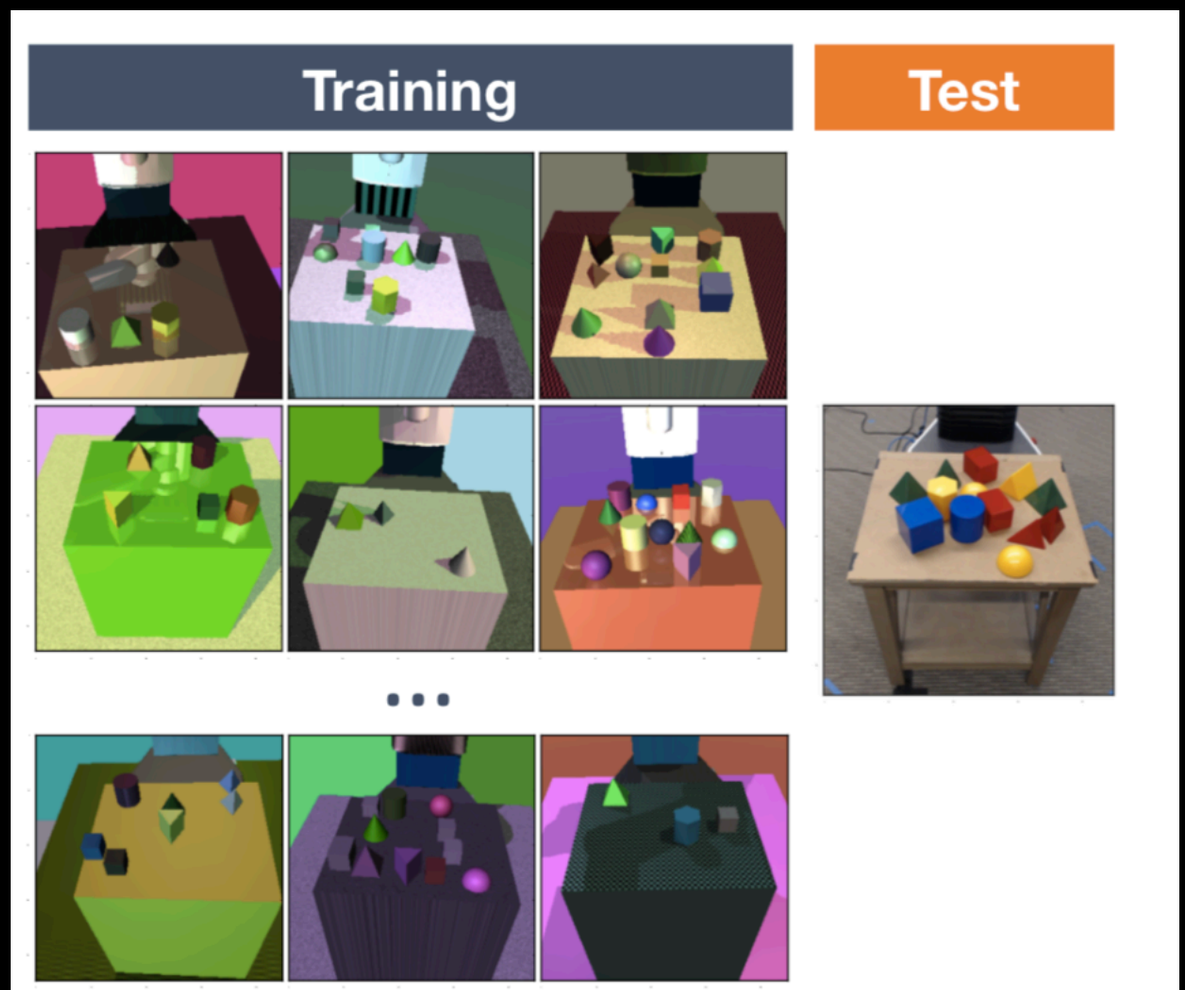
Domain Randomization

- In object detection, a deep neural network was trained using simulated images, to accurately localizing objects that were subject to distractor objects and occlusion.

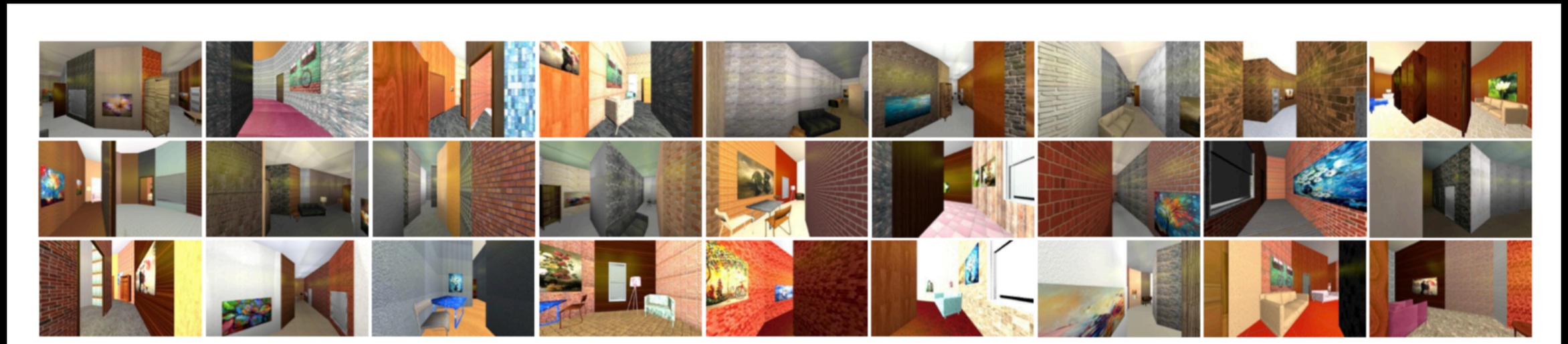
- In collision avoidance, a deep neural network was trained using simulated images, to create a policy that avoided obstacles in indoor hallways.

- Sample simulator based training images for both problems are shown on the next slide.

Object Localization Task



Collision Avoidance Task (Only domain randomized images are shown.)



Domain Adaptation - In Words

Suppose you are given two sets with similar tasks.

<u>Example</u>
Divide ImageNet into two sets - a training set and a test set.

By construction, the tasks for both sets are the same, since the tasks are the object labels.

<u>Domain Adaptation</u>
Keep the tasks for the two sets fixed.

However, modify the images in one set (say the test set) to something “similar”.

For example, the modified images could be edge maps, cartoon images, paintings, etc. of the original images in the test set.

The goal of domain adaptation, is to learn a robust mapping from the modified images to it’s labels, that is also consistent with the training set images and it’s labels.

Domain Adaptation - Mathematically

A domain $\mathcal{D}$ is composed of $\mathcal{X} \subset \mathcal{R}^{\mathrm{d}}$ with marginal probability distribution $\mathrm{P}(\mathbf{X})$.

A task $\mathcal{T}$ is defined by a label space $\mathcal{Y}$, and conditional probability distribution $\mathrm{P}(\mathbf{Y}|\mathbf{X})$.

$\mathbf{X}$ and $\mathbf{Y}$ are random variables.

Define the source domain and tasks to be:

$$\mathcal{D}^s = \{\mathcal{X}^s, P(\mathbf{X}^{\mathrm{s}})\}, \quad \mathcal{T}^{\mathrm{s}} = \{\mathcal{Y}^{\mathrm{s}}, \mathrm{P}(\mathbf{Y}^{\mathrm{s}}|\mathbf{X}^{\mathrm{s}})\}$$

Define the target domain and tasks to be:

$$\mathcal{D}^t = \{\mathcal{X}^t, P(\mathbf{X}^{\mathrm{t}})\}, \quad \mathcal{T}^{\mathrm{t}} = \{\mathcal{Y}^{\mathrm{t}}, \mathrm{P}(\mathbf{Y}^{\mathrm{t}}|\mathbf{X}^{\mathrm{t}})\}$$

Domain adaptation:

Let $\mathcal{D}^{\mathrm{s}} \neq \mathcal{D}^{\mathrm{t}}$ and $\mathcal{T}^{\mathrm{s}} = \mathcal{T}^{\mathrm{t}}$.

From $\{\mathcal{D}^{\mathrm{s}}, \mathcal{T}^{\mathrm{s}}\}$, we wish to learn $\mathrm{P}(\mathbf{Y}^{\mathrm{t}}|\mathbf{X}^{\mathrm{t}}\}$.

Domain Adaptation - (DANN) Adversarial Discriminative Models

- An architecture was proposed, where predictions / classifications were made based on features that were discriminative, yet invariant to the changes between the source and target domains.

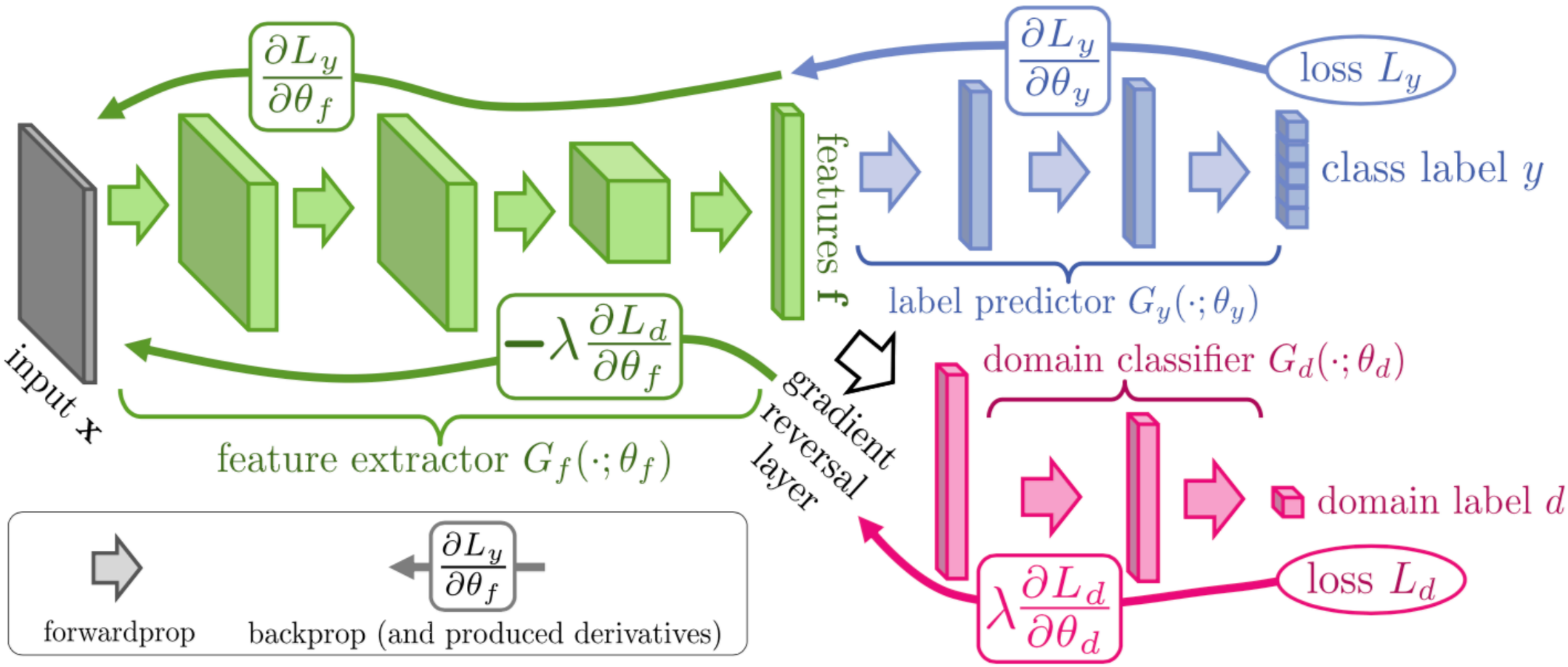
- The network was comprised of 3 components - a feature extractor, a label classifier and a domain classifier.

- The label classifier resembled the vanilla image recognition classifier, and was trained using the source domain.

- The domain classifier used both the source and target domain information, and learned to discriminate between the two during training.

Domain Adaptation - (DANN) Adversarial Discriminative Models

- During domain classifier back propagation, a gradient reversal was introduced, to change the sign of the gradient applied to the feature extractor network.
- i.e. The gradient was multiplied by a negative scalar before being applied in the feature extractor.
- This allowed the domain classifier to act as an adversary to the feature extractor.
- As a result, the feature extractor learned features that were invariant to both the source and target domains.



The green network learned features that were discriminative, yet invariant to the source and target domains.

The blue network learned to perform label classification, using these features.

The red network learned to discriminate between images in the source and target domains, using these features.

The network tried to minimize the loss associated with the class labels and the domain labels, while maximizing the input to the domain classifier.

Loss Function

$$\tilde{E}(\theta_f, \theta_y, \theta_d) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_y \left(G_y(G_f(\mathbf{x}_i; \theta_f); \theta_y), y_i \right) \tag{18}$$

$$- \lambda \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_d \left(G_d(\mathcal{R}(G_f(\mathbf{x}_i; \theta_f)); \theta_d), d_i \right) + \frac{1}{n'} \sum_{i=n+1}^{N} \mathcal{L}_d \left(G_d(\mathcal{R}(G_f(\mathbf{x}_i; \theta_f)); \theta_d), d_i \right) \right).$$

Gf represented the feature extractor network.
Gy represented the label classifier / label predictor network.
Gd represented the domain classifier network.

Ly denoted the label classifier loss while Ld denoted the domain classifier loss.

Ld had two terms, since half of the data was associated with the source domain while the other half of the data was associated with the target domain.

Domain Adaptation - (DSN) Data Reconstruction Methods

- In DANN, a shared representation (that was amenable to BOTH the source and target domains) was learned.

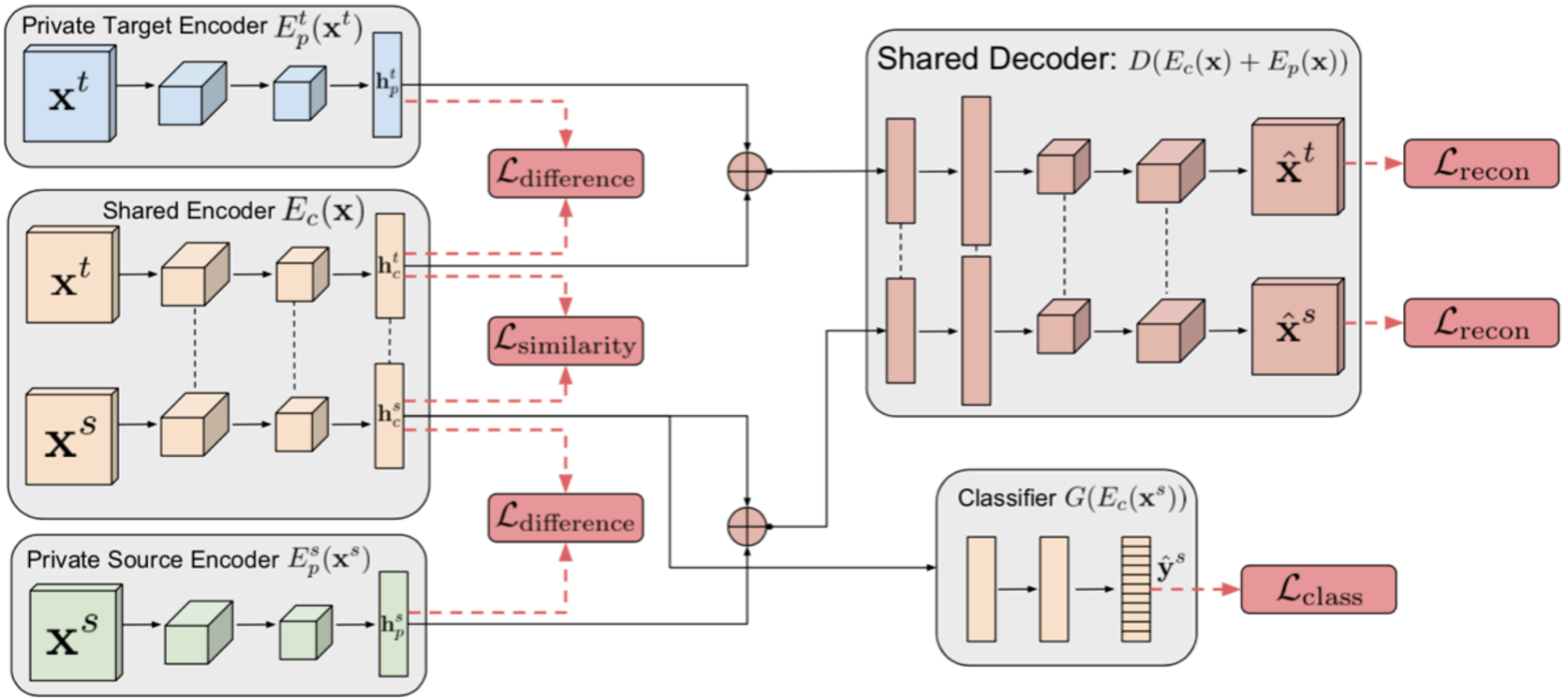
- In Domain Separation Networks (DSN), the underlying goal remained the same.

- However, separate private networks were also learned, to capture additional information that was unique to each domain.

- These separate private networks were then used to learn improved domain invariant features.

Domain Adaptation - (DSN) Data Reconstruction Methods

- Two fundamental assumptions were made, regarding DSN.
- First, it was assumed that the source and target domains differed mainly in their “low level” statistics (= background, noise, resolution, illumination and color).
- Second, it was assumed that the source and target domains had little to no “high level “differences (= 3D position, pose, geometric variations and object types).
- In DSN, the additional private low level subspaces / networks then captured unique “low level” statistics relevant to the specific domain.
- These subspaces / network were then used to find a shared subspace / network (that was orthogonal to the private subspaces) containing the domain invariant features.



The DSN architecture is shown above.

The goal was to learn a shared encoder network than could be used to classify the source and target domains.

The function of the shared decoder network was to ensure that a trivial solution did not result from the training.

The L_difference loss functions extracted the low level source and target domain information, infused this information into the private encoder networks, and then used these private network to further improve the performance of the shared encoder network.

Loss Function

The parameters associated with the various networks were given by:

$$\Theta = \{\theta_c, \theta_p, \theta_d, \theta_g\}$$ **{Shared Encoder, Private Encoders, Shared Decoder, Classifier}**

The loss function consisted of 4 components:

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

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

where $\mathbf{y}_i^s$ is the one–hot encoding of the class label for source input i and $\hat{\mathbf{y}}_i^s$ are the softmax predictions of the model: $\hat{\mathbf{y}}_i^s$

L_recon penalized outputs that were not close to the original inputs, thereby preventing trivial solutions:

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^{N_s} \mathcal{L}_{\text{si\_mse}}(\mathbf{x}_i^s, \hat{\mathbf{x}}_i^s) + \sum_{i=1}^{N_t} \mathcal{L}_{\text{si\_mse}}(\mathbf{x}_i^t, \hat{\mathbf{x}}_i^t)$$

Loss Function

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

where k is the number of pixels in input x, $\mathbf{1}_k$ is a vector of ones of length k; and $\|\cdot\|_2^2$ is the squared L_2-norm.

$$L_{\text{difference}} = \left\|\mathbf{H}_c^{s\top}\mathbf{H}_p^s\right\|_F^2 + \left\|\mathbf{H}_c^{t\top}\mathbf{H}_p^t\right\|_F^2,$$

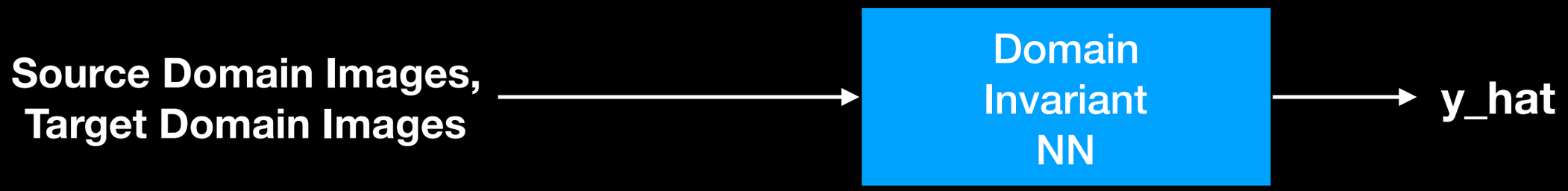
$\|\cdot\|_F^2$ is the squared Frobenius norm

Let $\mathbf{H}_c^s$ and $\mathbf{H}_c^t$ be matrices whose rows are the hidden *shared* representations $\mathbf{h}_c^s = E_c(\mathbf{x}^s)$ and $\mathbf{h}_c^t = E_c(\mathbf{x}^t)$ from samples of source and target data respectively. Similarly, let $\mathbf{H}_p^s$ and $\mathbf{H}_p^t$ be matrices whose rows are the *private* representation $\mathbf{h}_p^s = E_p^s(\mathbf{x}^s)$ and $\mathbf{h}_p^t = E_p^t(\mathbf{x}^t)$ from samples of source and target data respectively.

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

where $d_i \in \{0, 1\}$ is the ground truth domain label for sample i.

Domain Adaptation - (GAN) Adversarial Generative Models



Instead of trying to train a network that contained domain invariant features, a GAN was now trained to specifically map the source domain images into the target domain images.

These GAN generated images were then passed through a task classifier.

Domain Adaptation - (GAN) Adversarial Generative Models

- The input to the GAN now consisted of a stochastic noise vector AND the source image.

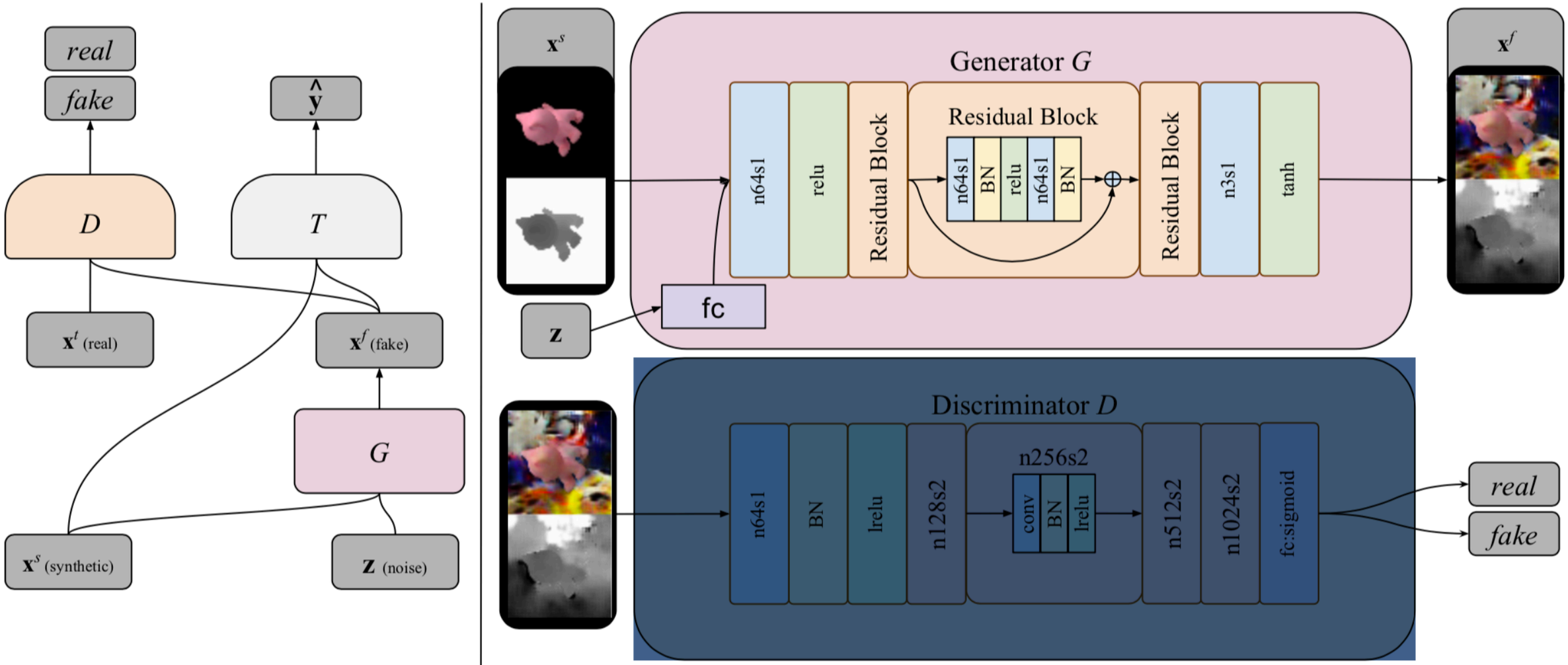
- A discriminator was trained to distinguish between the true target domain images and the images created by the generator.

- In addition, a task specific classifier was introduced, which assigned class specific labels to the generated images and the target images.

- Finally, it was assumed that the difference between the source and target domain was primarily low level (=noise, resolution, illumination, color) versus high level (=object types, geometric variations, etc.).

Domain Adaptation - (GAN) Adversarial Generative Models

- Training consisted of a 2 step process.
- First, the networks associated with the task classifier and the discriminator networks were trained, with the generator network fixed.
- Then the network associated with the generator was trained, with the task classifier and discriminator networks fixed.
- Training utilized three loss function: task loss, discriminator loss and a content loss.
- The content loss encouraged the GAN generated images to have foreground pixel values close to the original source pixel values, or pixel values that changed in a consistent way.



The complete network is shown above.

Like vanilla GANS, there was a generator network (G) and a discriminator network (D).

However, notice how both a noise vector AND a source image are now input into the generator.

Also, notice the addition of a task classifier, denoted by T.

Loss Function

The problem takes the form of a GAN minmax problem (but with an additional task network):

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

The discriminator, task and content losses are given by:

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

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

y^s is a one hot encoded vector for the class label associated with x^s.

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

m denotes the foreground pixel mask.
o denotes the Hadamard product.
k is the number of pixels in x^s.

Which Approach Is Better?

- DANN? DSN? GAN?
- The GAN currently wins, with respect to producing the top benchmark results.
- Recall though, that the GAN “encouraged” the generated output images to have foreground pixel values that were 1) close to the original source image foreground pixel values or 2) that changed in consistent ways, relative to the original source image foreground pixel values.
- This may be one of the reasons why the GAN approach performed better in the benchmark test sets.
- In general, a learning network that captures the invariance between the source and target domains is also a very reasonable line of research / investigation.

Simulations In Reinforcement Learning

- Recall: The end goal was to train a model / policy that operated effectively in the real world.

- In a direct approach, the model / policy was learned in the simulator.

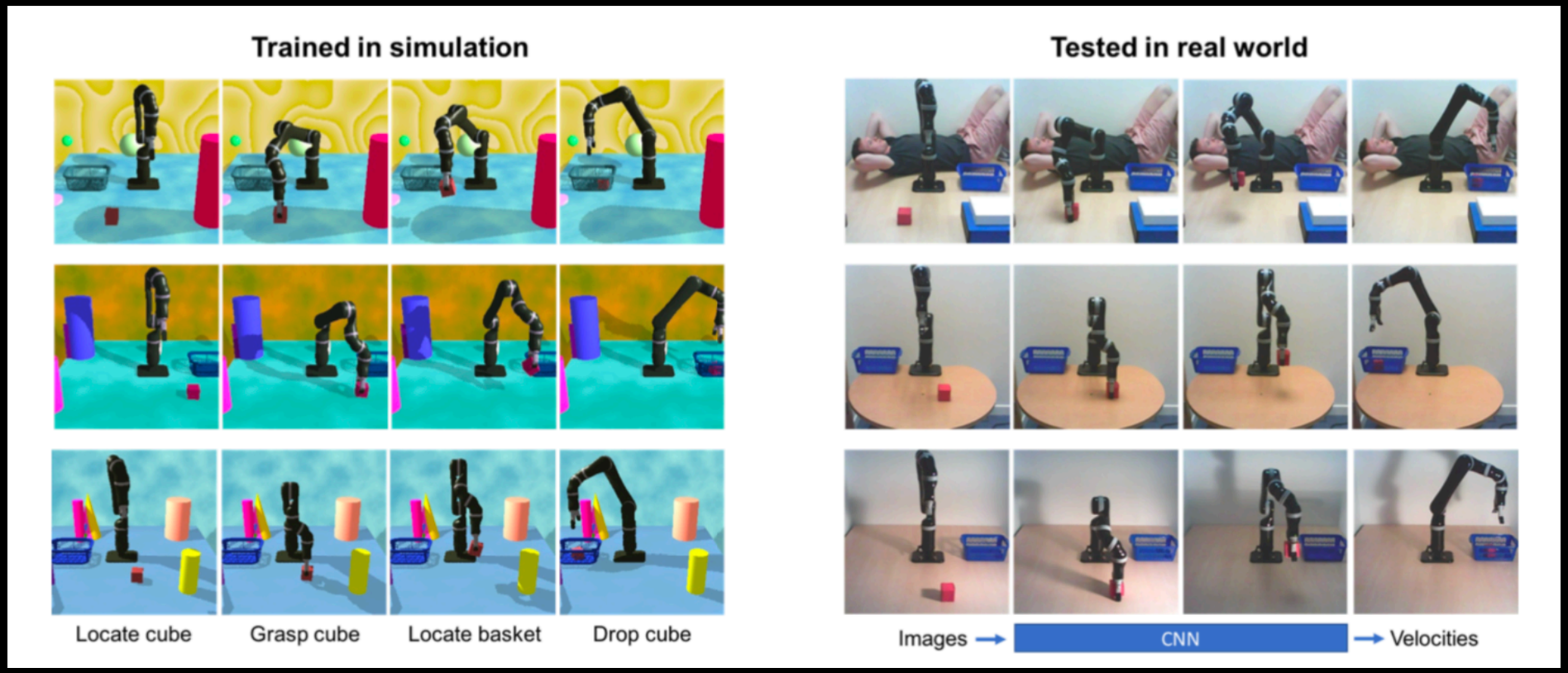
- It was then “hoped”, that the trained model / policy would operate successful in the real world.

- In the indirect approach, the model / policy was still learned in the simulator.

- However, when the model / policy operated in the real world, the real world images were first converted to simulated / canonical images, before being processed by the model / policy.

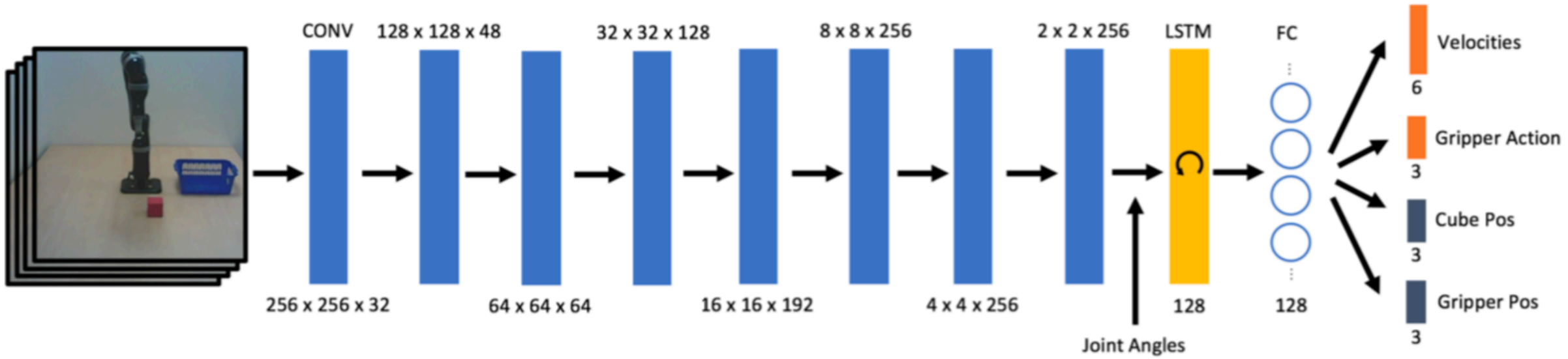
Sim to Real

- Train a model / learn a policy using only a simulator / simulated images.
- The trained model / learned policy should function well on real images.
- This concept was verified for a simple multi stage task, involving locating, reach for and grasping a cube, and then, locating a basket and dropping the cube into the basket.
- A large number of robot trajectories were created (using the simulator) to collect the control velocities associated with different robot joint angles for various test scenarios.
- A CNN was then trained to map simulated images (that had undergone domain randomization) and the joint angles to output control velocities and gripper actions.



Training occurred, using only the simulation images on the left.

Domain randomized simulation images



A CNN captured learned representations from the domain randomized images.

The high level representations and joint angles were then fed into a LSTM, which captured the state information associated with the various stages of the task.

The domain randomized input images enabled the model to generalize to different lighting conditions, distractor objects and moving objects.

The loss function consisted of 4 terms: MSE of the velocities, gripper action, cube position and gripper position.

The authors postulated that their method would not work if the task did not have multiple stages and if the objects required more complicated grasps.

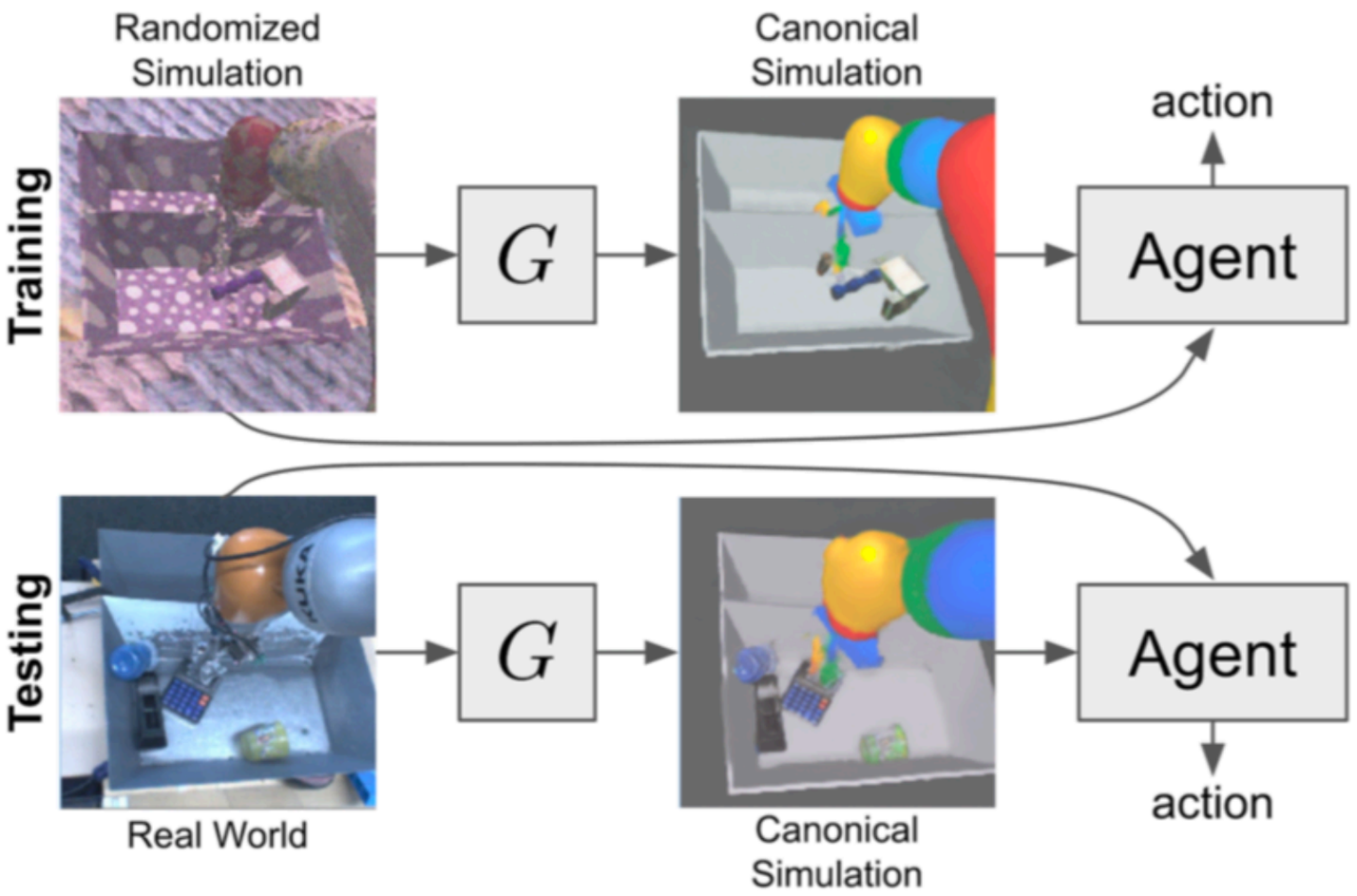
Sim to Real via Sim to Sim

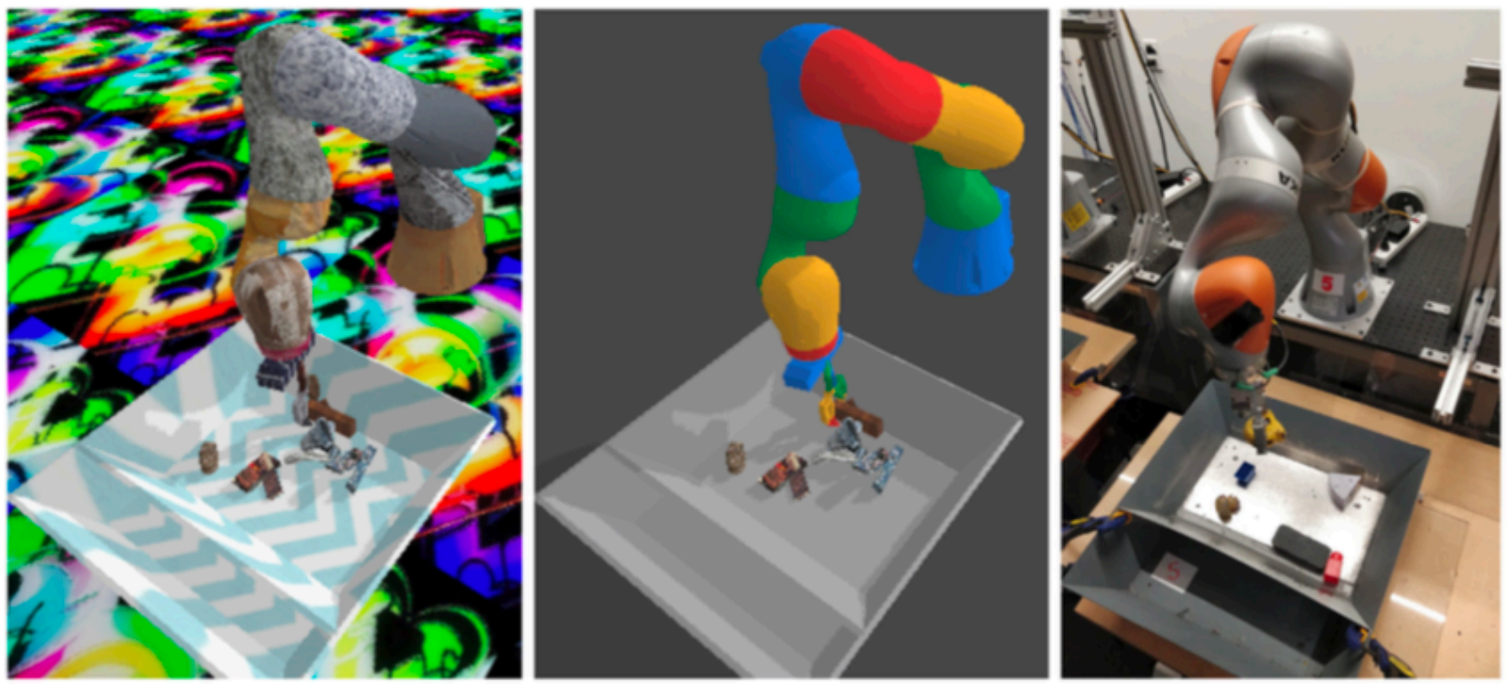
- Train a model / learn a policy using simulated / canonical images that were generated from domain randomized images (sim to sim).

- In real world operations, the real images were first converted into simulated / canonical images.

- These simulated / canonical images were then processed by the trained model / learned policy.

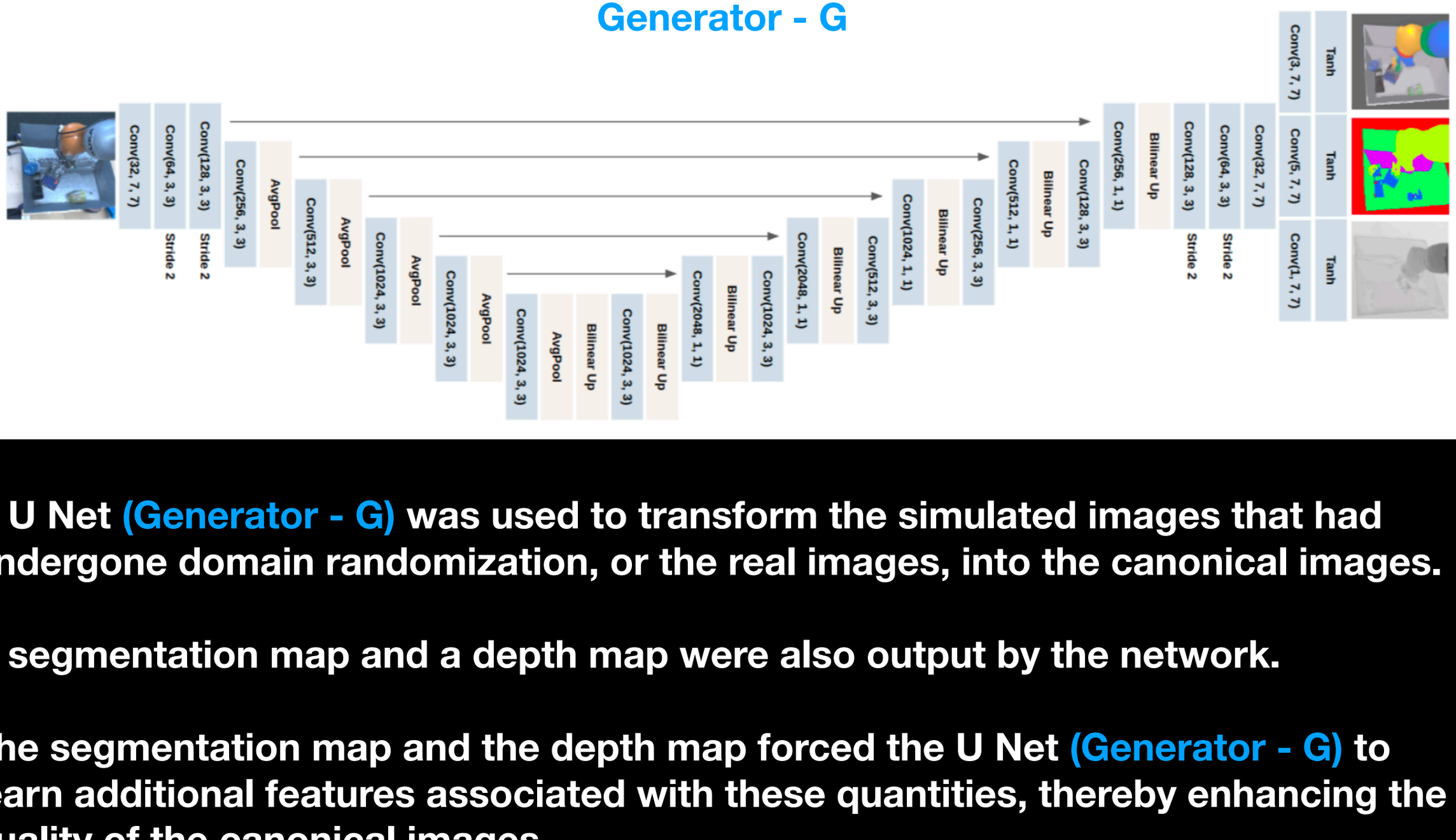
- i.e. Generate canonical images that were “close”, regardless of whether the inputs were simulated images that had undergone domain randomization, or, real images.





(a) Randomized (b) Canonical (c) Real

Generator - G

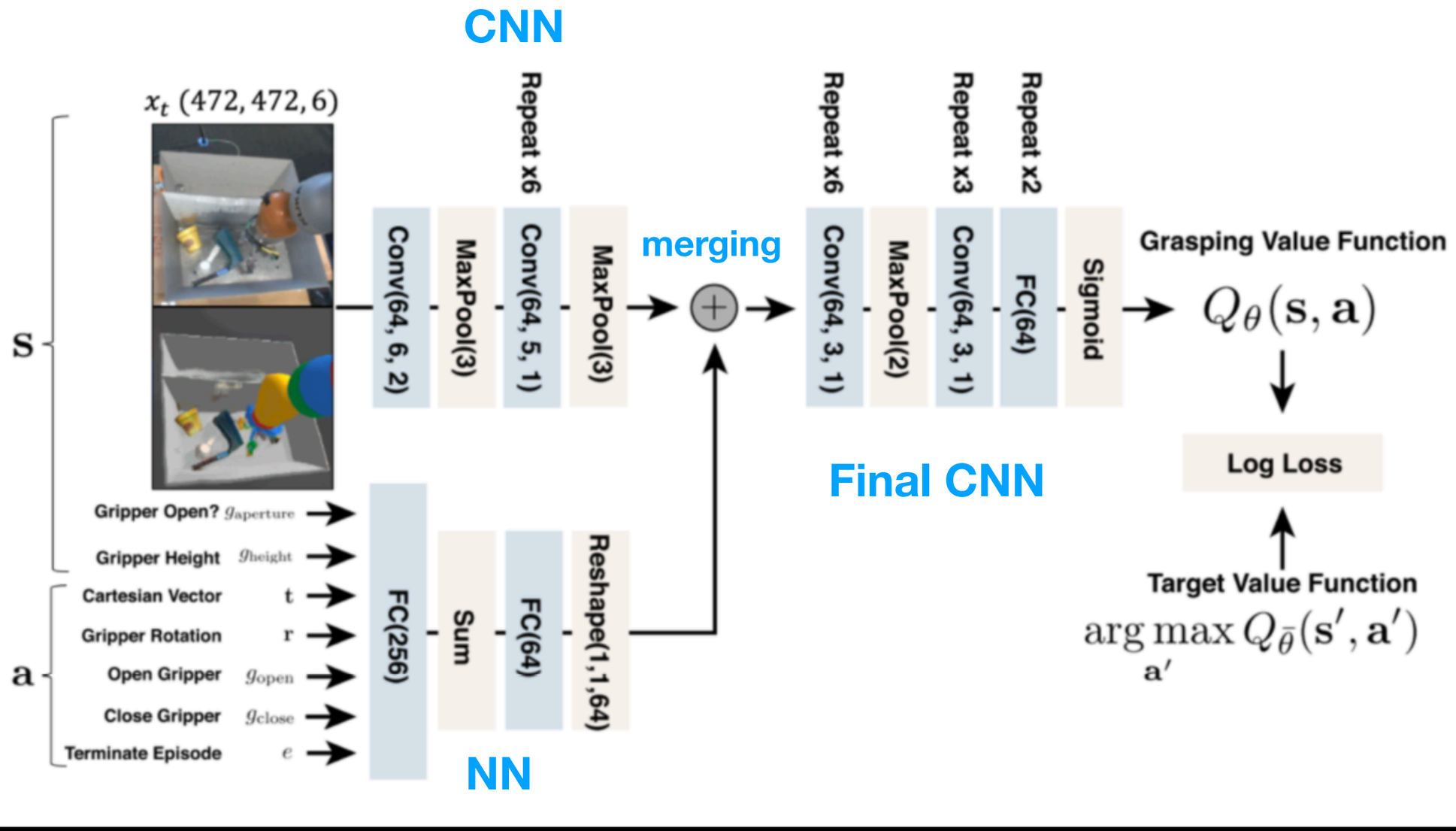


A U Net (Generator - G) was used to transform the simulated images that had undergone domain randomization, or the real images, into the canonical images.

A segmentation map and a depth map were also output by the network.

The segmentation map and the depth map forced the U Net (Generator - G) to learn additional features associated with these quantities, thereby enhancing the quality of the canonical images.

In addition, a sigmoid cross entropy GAN objective was used to encourage the creation of sharp canonical images.



Next, the input image and the generated canonical image (from the U Net) were fed into a CNN.

In addition, the state and action information was fed into a separate NN.

The outputs from the two networks were then merged through element wise addition.

This result was then fed into a final CNN, which produced the “Grasping” Value function.

Real To Sim

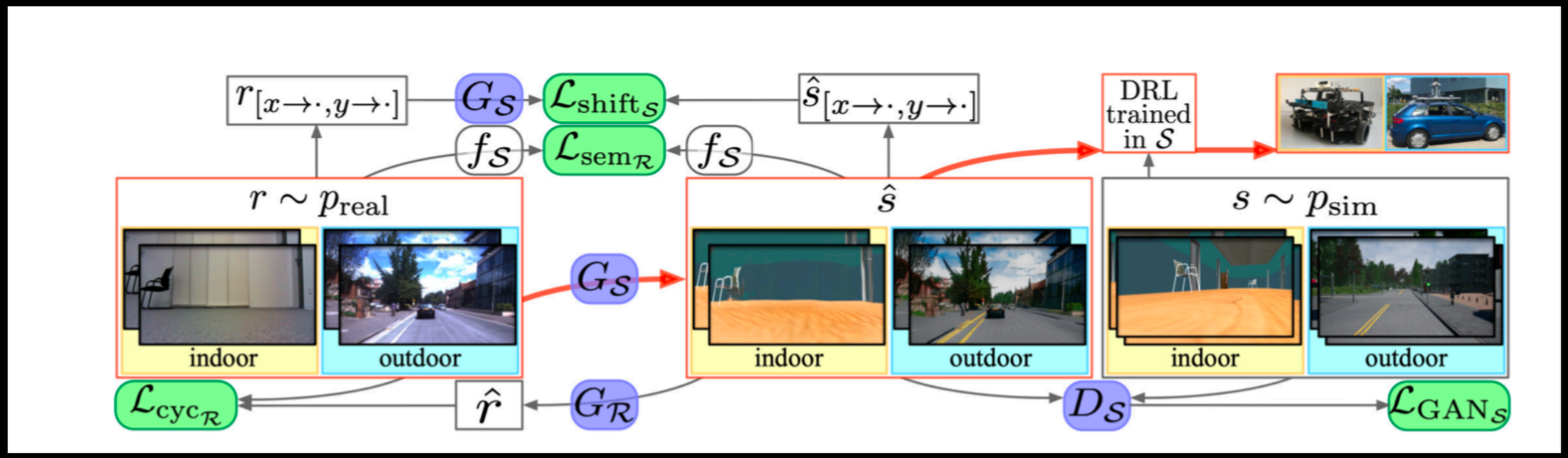
- This approach shared similarities with the previous approach, since the model / policy was learned using canonical images.

- However, a cycle GAN was employed to enhance the quality of the canonical images.

- In addition, a consistency constraint was also incorporated using a semantic loss function.

- Finally, a shift constraint was also added, to increase the temporal consistency between neighboring frames.

- Unlike the previous approach, the target application was navigation.



This block diagram was provided by the authors, and actually omits the counterpart losses: L_cycle_s, Lshift_r, L_sem_s and L_GAN_r.

The approach consisted of 2 main components: cycleGAN and shift consistency

In addition to the vanilla cycleGAN, the authors also added a semantic loss to further improve the quality of the cycleGAN output.

In order to address temporal consistency without employing optical flow, the authors introduced the above mentioned semantic loss and a shift loss.

To obtain a fuller understanding of the entire process, the optimization objective and the loss function are included on the next slide.

Loss Function

This problem took the form of a GAN minmax problem - only now, a cycleGAN Was used (= 2 Generator networks and 2 Discriminator networks).

$$G_{\mathcal{R}}^*, G_{\mathcal{S}}^* = \arg \min_{G_{\mathcal{R}}, G_{\mathcal{S}}} \max_{D_{\mathcal{R}}, D_{\mathcal{S}}} \mathcal{L}(G_{\mathcal{R}}, G_{\mathcal{S}}, D_{\mathcal{R}}, D_{\mathcal{S}})$$

$$\begin{aligned}
&\mathcal{L}(G_{\mathcal{R}}, G_{\mathcal{S}}, D_{\mathcal{R}}, D_{\mathcal{S}}; \mathcal{S}, \mathcal{R}, f_{\mathcal{S}}) \\
&\quad = \mathcal{L}_{\text{GAN}_{\mathcal{R}}}(G_{\mathcal{R}}, D_{\mathcal{R}}; \mathcal{S}, \mathcal{R}) + \mathcal{L}_{\text{GAN}_{\mathcal{S}}}(G_{\mathcal{S}}, D_{\mathcal{S}}; \mathcal{R}, \mathcal{S}) \\
&\quad + \lambda_{\text{cyc}} \left(\mathcal{L}_{\text{cyc}_{\mathcal{R}}}(G_{\mathcal{S}}, G_{\mathcal{R}}; \mathcal{R}) + \mathcal{L}_{\text{cyc}_{\mathcal{S}}}(G_{\mathcal{R}}, G_{\mathcal{S}}; \mathcal{S}) \right) \\
&\quad + \lambda_{\text{sem}} \left(\mathcal{L}_{\text{sem}_{\mathcal{R}}}(G_{\mathcal{S}}; \mathcal{R}, f_{\mathcal{R}}) + \mathcal{L}_{\text{sem}_{\mathcal{S}}}(G_{\mathcal{R}}; \mathcal{S}, f_{\mathcal{S}}) \right) \\
&\quad + \lambda_{\text{shift}} \left(\mathcal{L}_{\text{shift}_{\mathcal{R}}}(G_{\mathcal{R}}; \mathcal{S}) + \mathcal{L}_{\text{shift}_{\mathcal{S}}}(G_{\mathcal{S}}; \mathcal{R}) \right).
\end{aligned}$$

The GAN and cycle losses were given by:

$$\begin{aligned}
\mathcal{L}_{\text{GAN}_{\mathcal{R}}}(G_{\mathcal{R}}, D_{\mathcal{R}}; \mathcal{S}, \mathcal{R}) =& \mathbb{E}_{p_{\text{real}}} [\log D_{\mathcal{R}}(r)] + \\
& \mathbb{E}_{p_{\text{sim}}} [\log(1 - D_{\mathcal{R}}(G_{\mathcal{R}}(s)))] \\
\mathcal{L}_{\text{GAN}_{\mathcal{S}}}(G_{\mathcal{S}}, D_{\mathcal{S}}; \mathcal{R}, \mathcal{S}) =& \mathbb{E}_{p_{\text{sim}}} [\log D_{\mathcal{S}}(s)] + \\
& \mathbb{E}_{p_{\text{real}}} [\log(1 - D_{\mathcal{S}}(G_{\mathcal{S}}(r)))]
\end{aligned}$$

Enhanced discussion of cycleGANS can be found in the UnsupervisedLearning_ImplicitModels_GANS pdf.

$$\begin{aligned}
\mathcal{L}_{\text{cyc}_{\mathcal{R}}}(G_{\mathcal{S}}, G_{\mathcal{R}}; \mathcal{R}) &= \mathbb{E}_{p_{\text{real}}} [||G_{\mathcal{R}}(G_{\mathcal{S}}(r)) - r||_1] \\
\mathcal{L}_{\text{cyc}_{\mathcal{S}}}(G_{\mathcal{R}}, G_{\mathcal{S}}; \mathcal{S}) &= \mathbb{E}_{p_{\text{sim}}} [||G_{\mathcal{S}}(G_{\mathcal{R}}(s)) - s||_1]
\end{aligned}$$

Loss Function

The semantic loss functions were given by:

$$\mathcal{L}_{\text{sem}_\mathcal{R}}(G_\mathcal{S}; \mathcal{R}, f_\mathcal{S}) = \mathbb{E}_{p_{\text{real}}}[\text{CrossEnt}(f_\mathcal{S}(r), f_\mathcal{S}(G_\mathcal{S}(r)))]$$
$$\mathcal{L}_{\text{sem}_\mathcal{S}}(G_\mathcal{R}; \mathcal{S}, f_\mathcal{S}) = \mathbb{E}_{p_{\text{sim}}}[\text{CrossEnt}(f_\mathcal{S}(s), f_\mathcal{S}(G_\mathcal{R}(s)))]$$

Enhanced discussion of semantic loss can be found in the SupervisedLearning_Segmentation pdf.

Finally, a new shift loss was introduced:

$$\mathcal{L}_{\text{shift}_\mathcal{R}}(G_\mathcal{R}; \mathcal{S}) = \mathbb{E}_{p_{\text{sim}},\ i,j\sim u(1,K-1)} \left[\left|\left| G_\mathcal{R}(s)_{[x\to i, y\to j]} - G_\mathcal{R}(s_{[x\to i, y\to j]}) \right|\right|_2^2 \right]$$
$$\mathcal{L}_{\text{shift}_\mathcal{S}}(G_\mathcal{S}; \mathcal{R}) = \mathbb{E}_{p_{\text{real}},\ i,j\sim u(1,K-1)} \left[\left|\left| G_\mathcal{S}(r)_{[x\to i, y\to j]} - G_\mathcal{S}(r_{[x\to i, y\to j]}) \right|\right|_2^2 \right]$$

The shift loss encouraged temporal consistency in the following way:

1) **Sample i and j shift quantities from a uniform distribution.**
2) **Pass a shifted version of the simulated image (real image) through the Real Generator network (Simulated Generator network).**
3) **Pass a non shifted version of the simulated image (real image) through the Real Generator network (Simulated Generator network), and then apply the shift.**
4) **Encourage the network to minimize the difference resulting from 2) and 3).**

Summary

- In Simulations (Part1), the human visual system (HVS) was examined.
- Different elements in the HVS were discussed, relating to color, exposure, focus, contrast, edge / line detection, texture, depth and shadows.
- This understanding provided a nice segue into the application of simulations for vision based reinforcement learning problems.
- Here, the objective is to use simulated images to train a model / learn a policy that would be effective in the real world, without the significant cost associated with acquiring real world training data.

References

- CAD2RL: Real Single Image Flight Without a Single Real Image, Sadhegi and Levine, arXiv June 2017
- Domain Randomization for Transferring Deep Neural Networks from Simulation to Real World, Tobin, et. al., March 2017
- Domain Adaptation for Visual Applications: A Complete Survey, Csurka, arXiv March 2017
- Learning Transferrable Features with Deep Adaptation Networks, Long, et. al., ICML 2015
- Domain Adversarial Training of Neural Networks (DANN), Ganin, et. al., JMLR, April 2016
- Domain Separation Networks (DSN), Bousmalis, et. al., NIPS 2016
- Unsupervised Pixel Level Domain Adaptation with GANS (GAN), Bousmalis, et. al., arXiv August 2017
- Adapting Deep Visuomotor Representations with Weak Pairwise Constraints, Tzeng, et. al., arXiv May 2017
- Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robot Grasping, Bousmalis, et. al., arXiv September 2017
- Transferring End to End Visuomotor Control from Simulation to Real World for a MultiStage Task (Sim to Real), Davison, et. al., arXiv October 2017
- Sim to Real Robot Learning from Pixels with Progressive Nets, Rusu, et. al., arXiv May 2018
- Sim to Real via Sim to Sim, James, et. al., arXiv July 2019
- QT-Opt: Scalable Deep RL for Vision Based Robotic Manipulation, Kalashnikov, et. al., arXiv November 2018
- VR Goggles for Robots: Real to Sim Domain Adaptation for Visual Control, Zhang, et. al., arXiv February 2018