Imperial College London

EE462/EE9CS728/EE9SO25

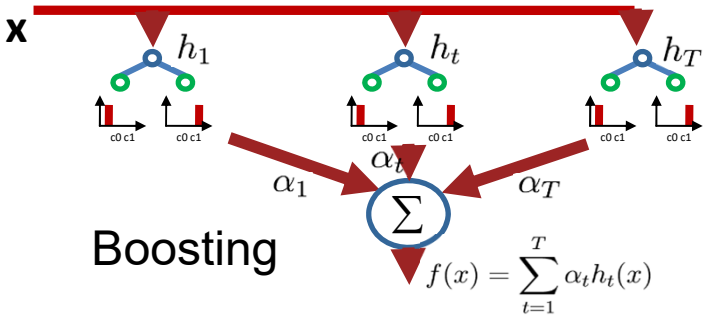
Generative Adversarial Networks (GANs)

Tae-Kyun (T-K) Kim

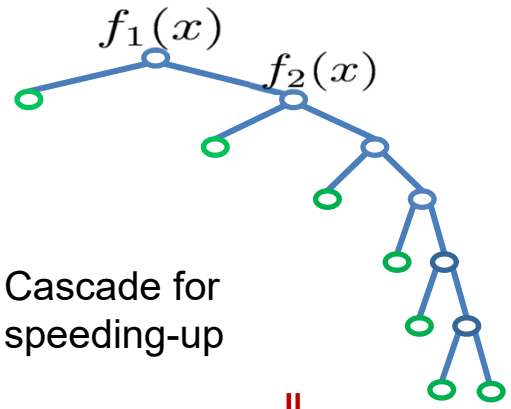
Senior Lecturer

https://labicvl.github.io/

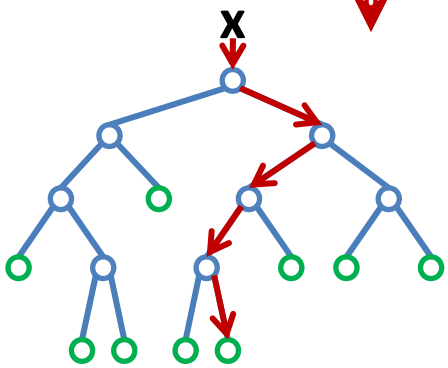
Tree Architecture

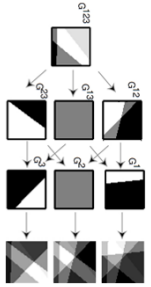


Boosting

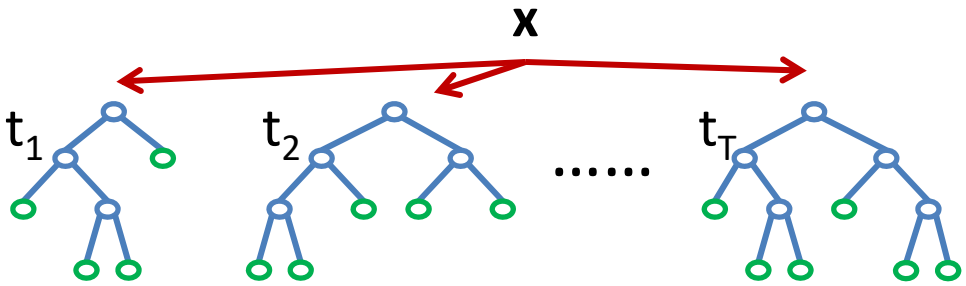


Cascade for speeding-up



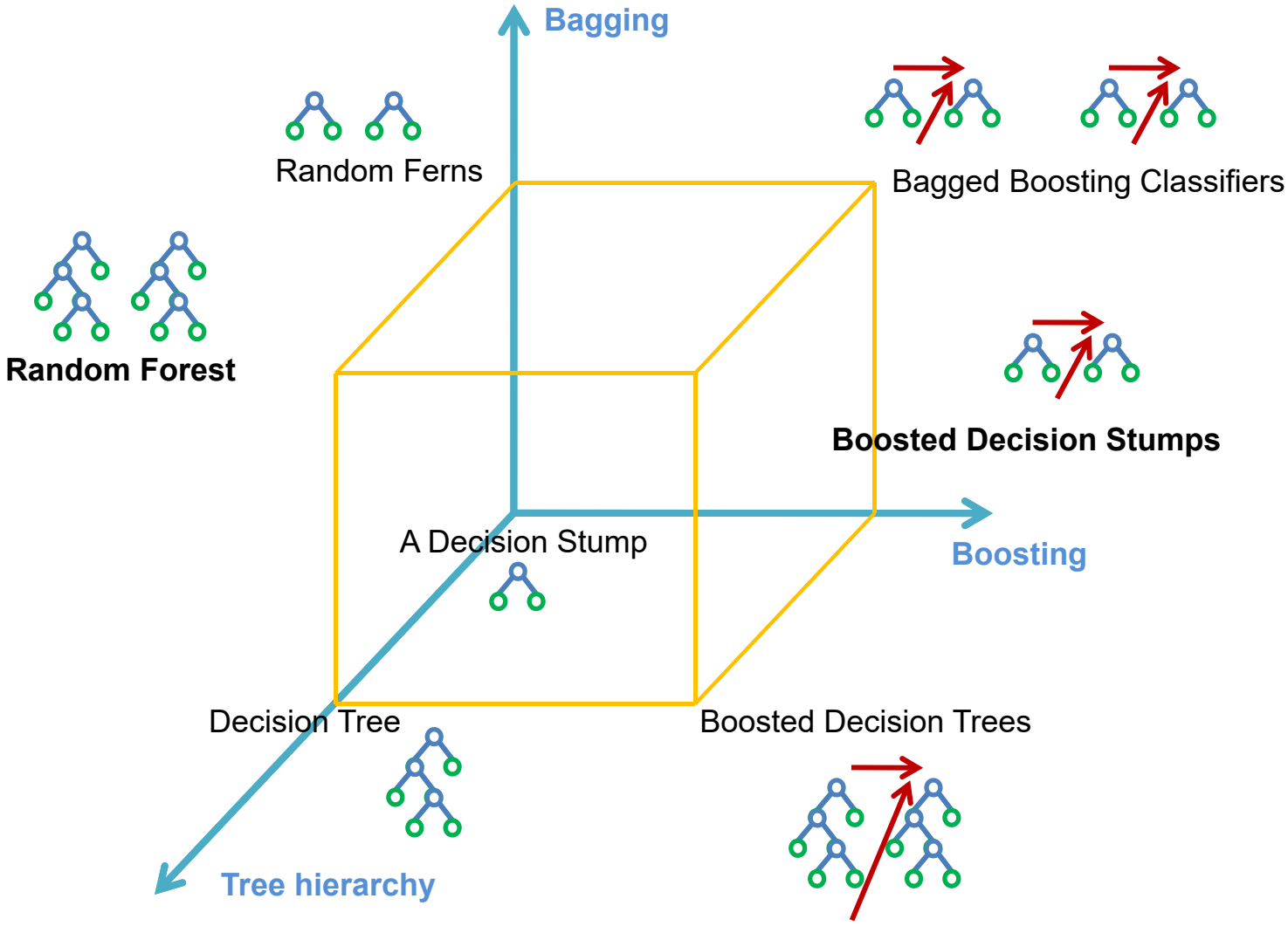


Tree-structure for multi-classes



Randomised Forests/Ferns

Tree Architecture



Deep Convolutional Neural Networks and Decision Forests

Connectivity (memory)

- CNN is fully connected.
- Memory used in decision trees grows exponentially with depth.

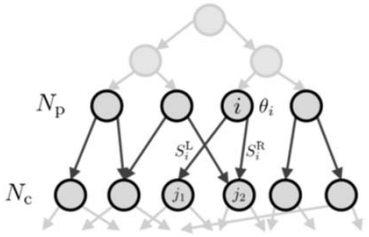
Data representation

- DF requires a set of features manually defined.
- Neural Decision Forests jointly tackles data representation and discriminative tree learning.

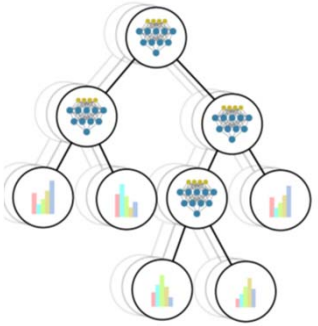
Conditional Convolutional Neural Network

- Learning representation, conditional computing, efficiency
- Cross modality face recognition via deep learning

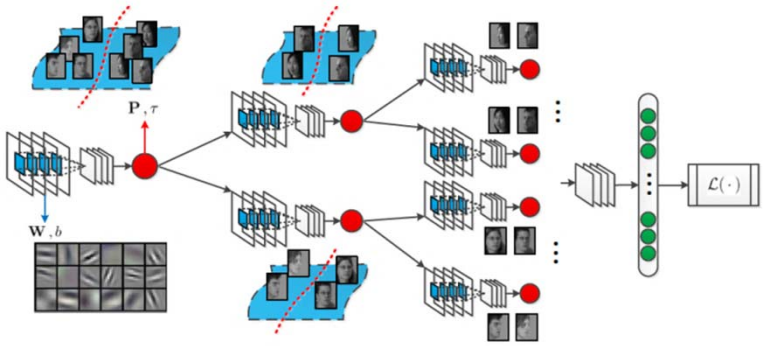
Hierarchically Gated Deep Networks, Qi, CVPR16
Deep Decision Network, Murthy et al, CVPR16
Joint Unsupervised Learning of Deep Representations and Image Clusters, Parikh et al, CVPR16



Decision Jungle
(Shotton et al NIPS13)



NDF (Kontschieder et al, CVPR14, ICCV15)

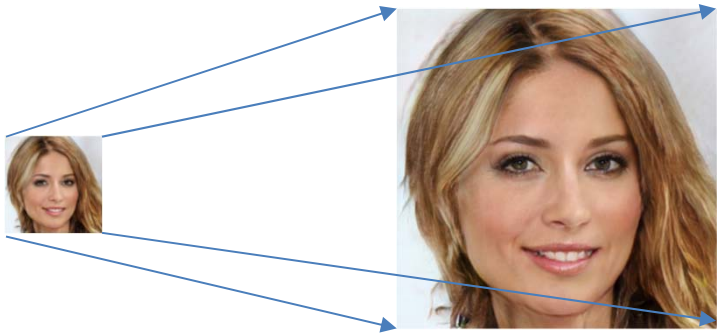


(Xiong, Kim et al ICCV15)

Up-sampling with Transposed Convolution

The Need for Up-sampling

- When we use neural networks to generate images, it usually involves up-sampling from low resolution to high resolution.



A simple way to conduct up-sampling is interpolation:

- Nearest neighbor interpolation, Bi-linear interpolation, Bi-cubic interpolation

Instead, up-sampling can be performed in-network for end-to-end learning by backpropagation from the pixelwise loss.

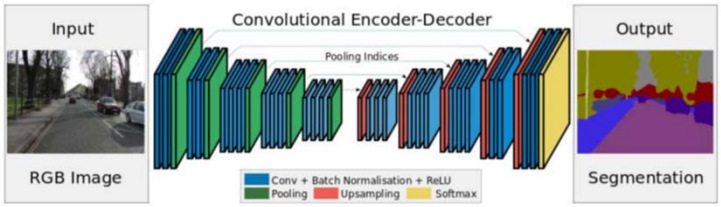
Up-sampling with Transposed Convolution

Why Transposed Convolution?

- The deconvolution filter in such a layer need not be fixed (e.g., to bilinear upsampling), but can be learned.
- A stack of deconvolution layers and activation functions can even learn a nonlinear upsampling.
- In-network upsampling is fast and effective for learning dense prediction.

The transposed convolution is useful for e.g.

- The generator in DCGAN takes randomly sampled values to produce a full-size image.
- The semantic segmentation



Badrinarayanan et al, 2015

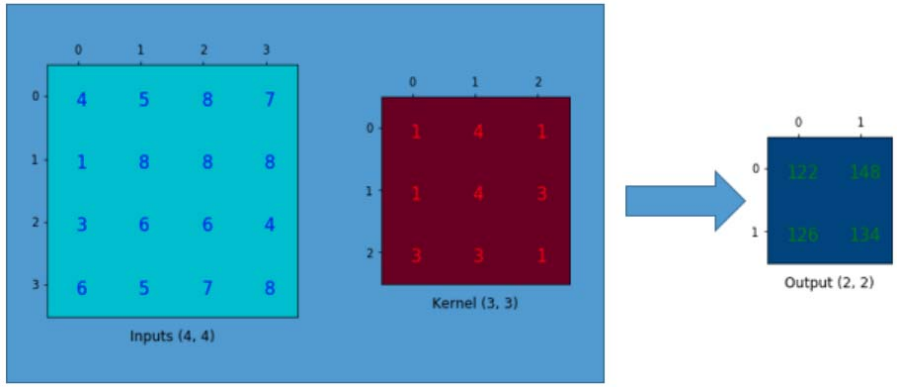
The transposed convolution is also known as:

Deconvolution

Fractionally-strided convolution

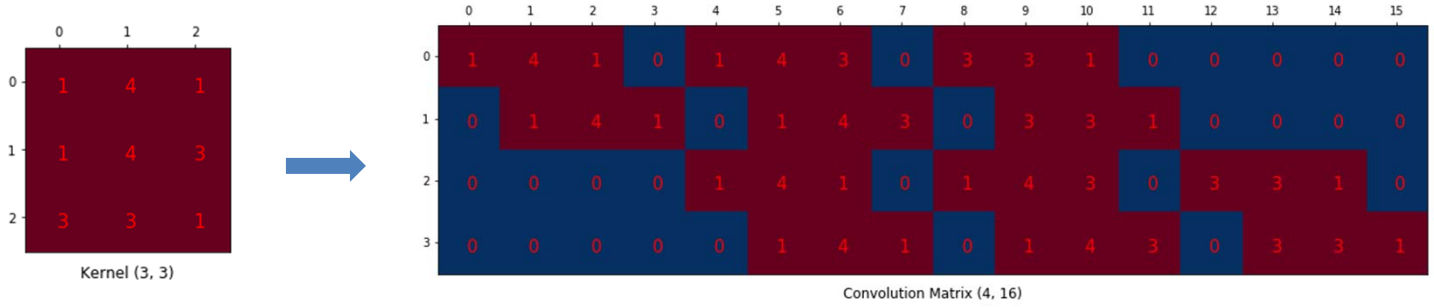
Convolution Operation

– We have a 4x4 matrix and apply a 3x3 convolution kernel, with no padding, and with a stride of 1. The output is a 2x2 matrix.

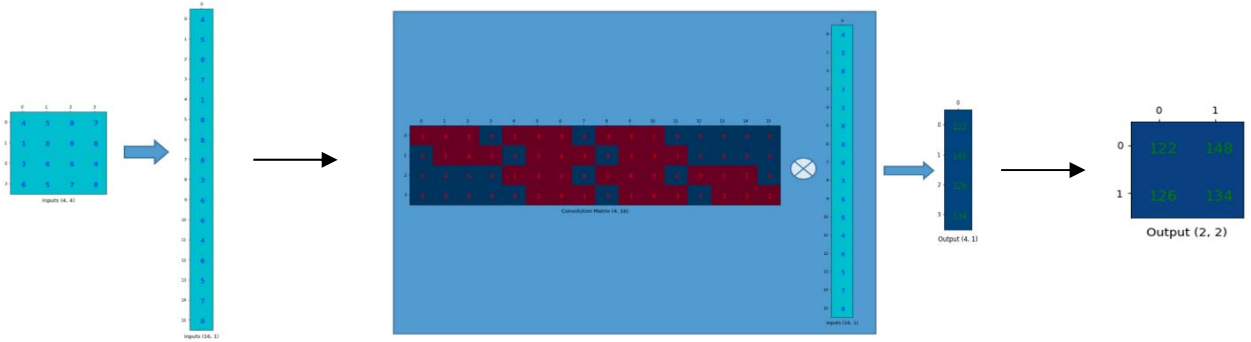


– The convolution operation can be conducted by a matrix multiplication.

– We rearrange the 3x3 kernel into a 4x16 matrix.

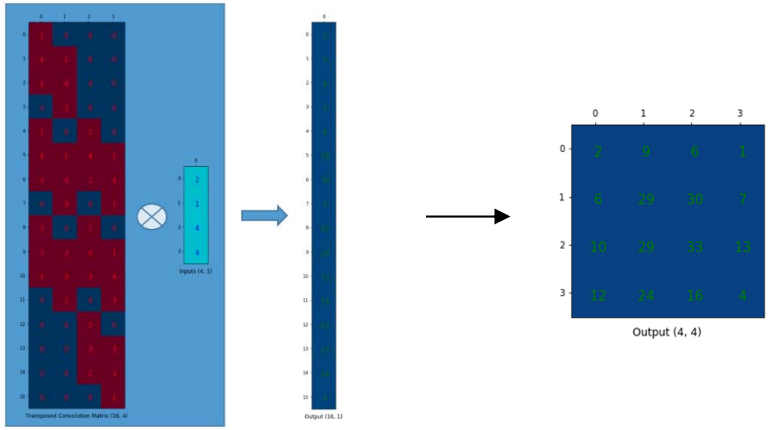


– We flatten the input matrix (4x4) into a column vector (16x1), then matrix-multiply the 4x16 convolution matrix with the 16x1 input matrix.



Going Backward by Transposed Convolution Matrix

– We up-sample a 2x2 matrix to a 4x4 matrix.



Generative Adversarial Networks (GAN)

Generative adversarial networks (GAN)

- *Ian Goodfellow, et al. NIPS 2014*

The coolest idea in machine learning in the last twenty years –Yann Lecun

Image generation

→ Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked GANs, 2016

Generative Adversarial Networks (GAN)

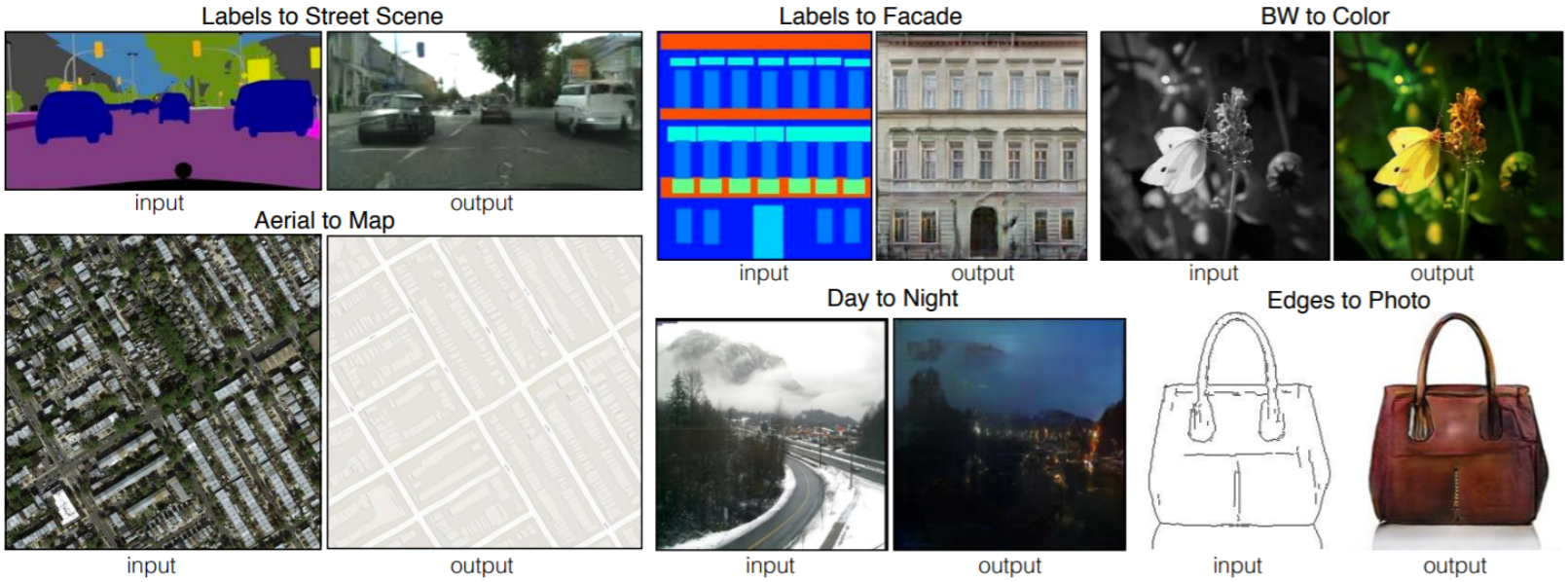
Generative adversarial networks (GAN)

- *Ian Goodfellow, et al. NIPS 2014*

The coolest idea in machine learning in the last twenty years –Yann Lecun

Image translation

→ Isola et al, Image-to-image translation with convolutional adversarial networks, CVPR 2017



Generative adversarial networks (GAN)

3.5 Years of Progress on Faces

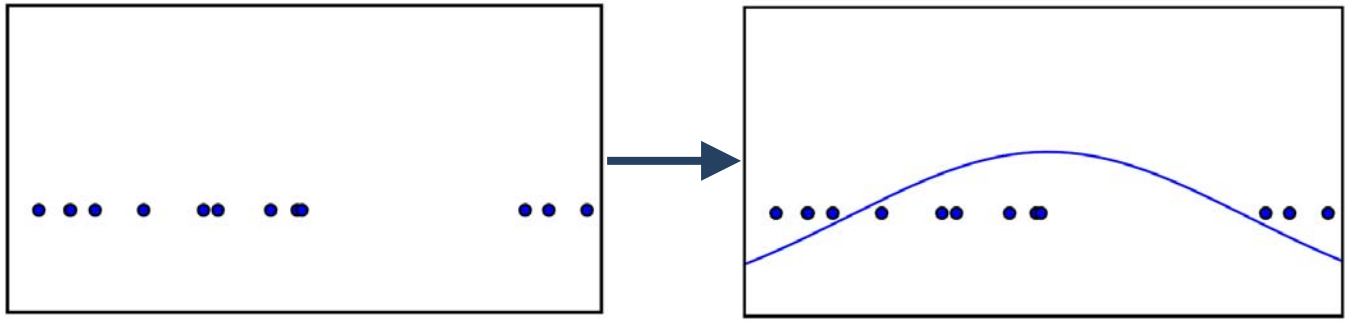
Brundage et al. 2018

Generative Models

- Generative Adversarial Networks (GANs) are example of generative models.

- Generative models take a training set (samples drawn from a data-generating distribution p_{data}), and learn to represent an estimate of that distribution. The result is a probability distribution p_{model}.

- In some cases, the model estimates p_{model} explicitly. Generative model performing density estimation takes training data, which are of an unknown data-generating distribution p_{data}, and return an estimate of that distribution. The estimate p_{model} can be evaluated for a particular value of $\boldsymbol{x}$ to obtain an estimate $p_{model}(\boldsymbol{x})$ of the true density $p_{model}(\boldsymbol{x})$:



Generative Models

- In other cases, the model is only able to generate samples from p_{model}. Some generative models are able to generate samples from the model distribution p_{model}. Ideally, a generative model would be able to train on examples (left), and then create more examples from the same distribution (right):

training examples

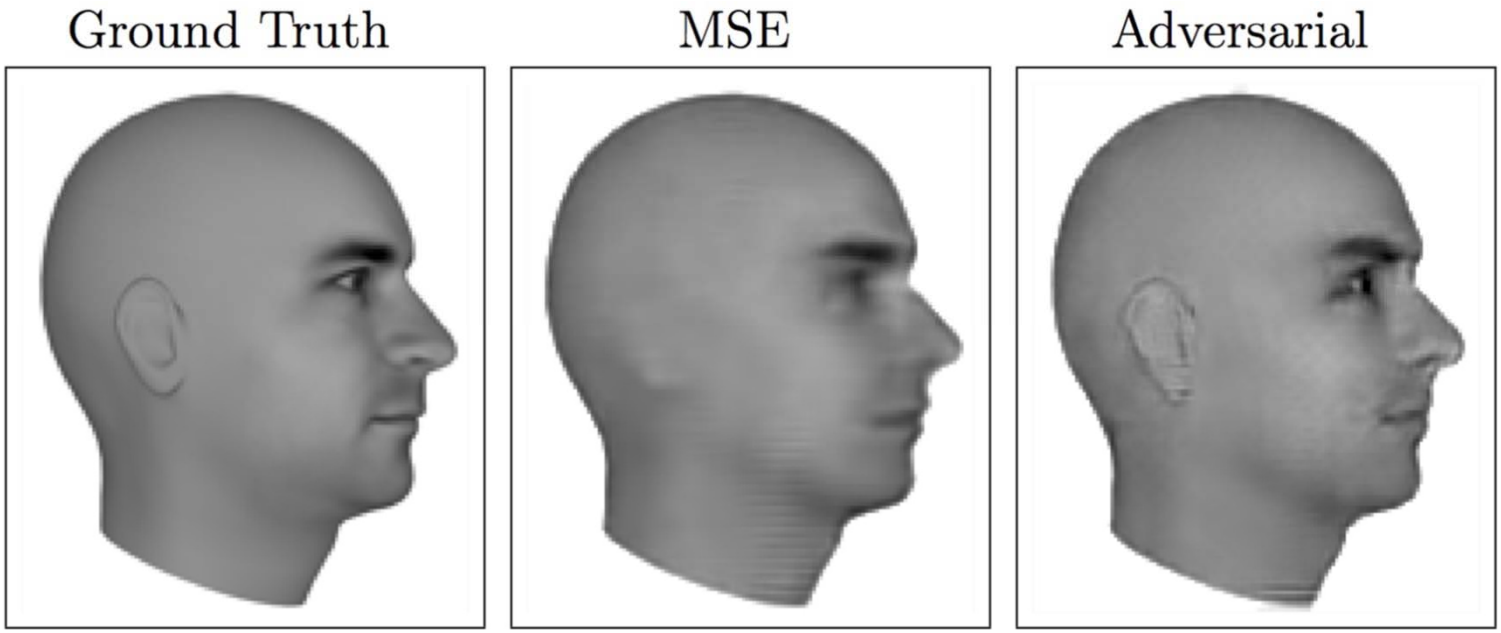
model samples

Generative Modelling: WHY?

- High-dimensional probability distributions are important in engineering and applied math.
- Training and sampling from generative models is an excellent test of our ability to represent and to manipulate high-dimensional probability distributions.
- Generative models can be incorporated into reinforcement learning:
  * for planning,
  * for learning in an imaginary environment,
  * to guide exploration,
  * (especially GANs) can also be used for inverse reinforcement learning.
- Generative models can be trained with missing data and can provide predictions on inputs that are missing data.
- (Particularly GANs) are able to perform semi-supervised learning reasonably well.

Generative Modelling: WHY?

- Generative models (GANs particularly) enable machine learning to work with "multi-modal" outputs.
- Some traditional models, such as minimizing the mean squared error between a desired output and the model's predicted output, are not able to produce multiple different correct answers.



Predicting the next frame in a video sequence

Generative Modelling: WHY?

Single image super-resolution

Art creation

How do generative models work?

Maximum Likelihood Estimation

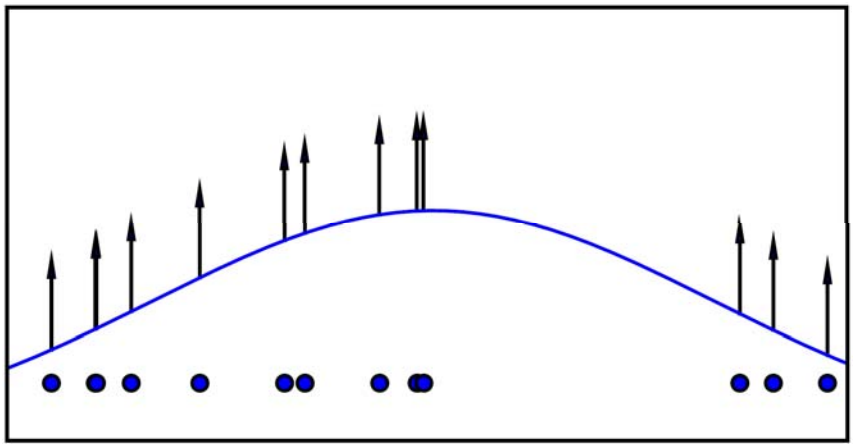
- Most generative models work via the principle of maximum likelihood.
- The basic idea of maximum likelihood is to define a model that provides an estimate of a probability distribution, parameterized by θ.
- Likelihood is then referred as the probability that the model assigns to the training data for a dataset containing m training samples $\boldsymbol{x}^{(i)}$:

$$\theta^* = \operatorname{argmax}_\theta \prod_{i=1}^{m} p_{model}(\boldsymbol{x}^{(i)}; \theta)$$
$$= \operatorname{argmax}_\theta \sum_{i=1}^{m} \log p_{model}(\boldsymbol{x}^{(i)}; \theta)$$

- Maximum likelihood chooses the parameters for the model that maximizes the likelihood of the training data.

How do generative models work?

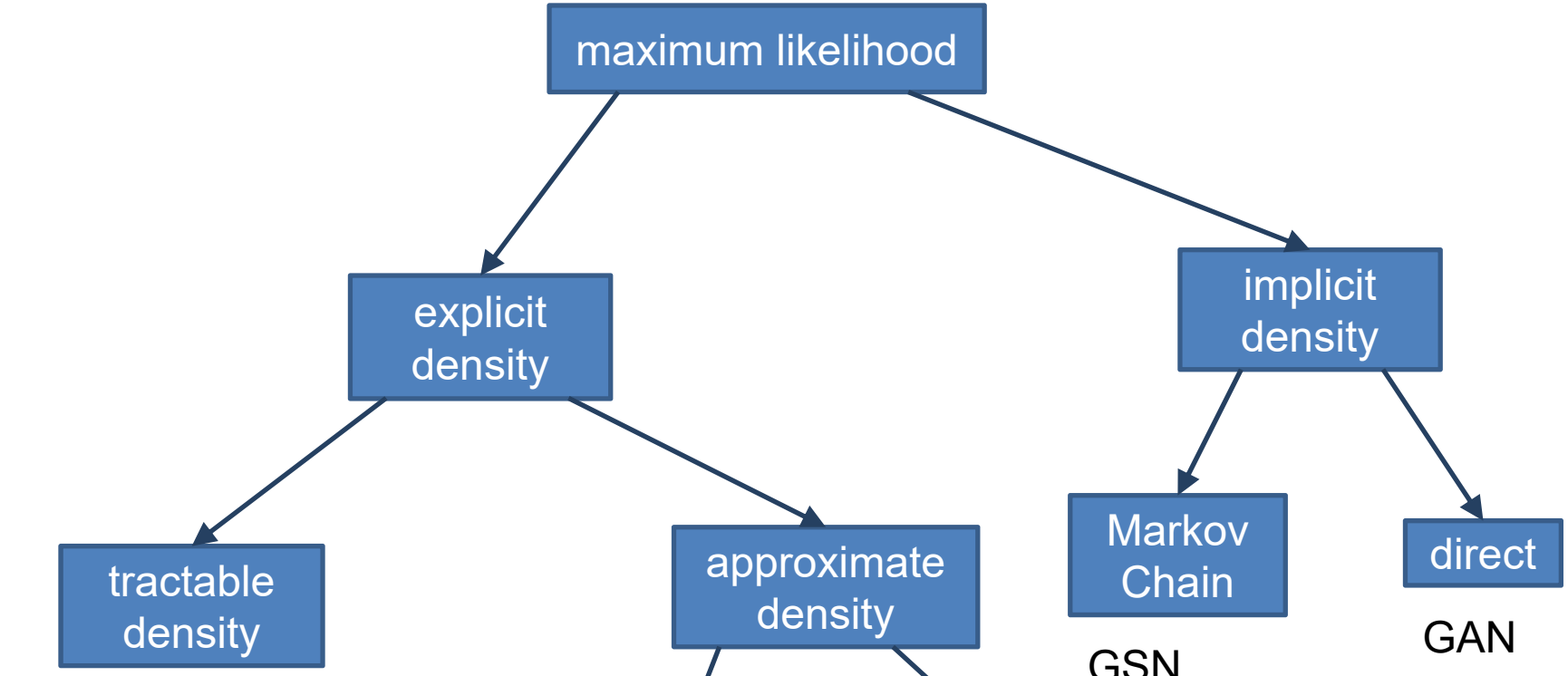
Maximum Likelihood Estimation



$$\theta^* = \arg\max_{\theta} \mathbb{E}_{x \sim p_{data}} \log p_{model}(\mathrm{x} \mid \theta)$$

Gaussian model applied to 1-D data: The maximum likelihood process takes several samples from the data generating distribution to form a training set. Different data points push up on different parts of the density function. One point pushes up in one place and inevitably pulls down in other places, since the density function must sum to 1.

Deep Generative Models: A Taxonomy



- Fully visible belief nets
  - NADE
  - MADE
  - PixelRNN
- Change of variables models (nonlinear ICA)

Deep Generative Models: A Taxonomy

1) Explicit Density Models

— These models define an explicit density function $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$.

— Maximization of the likelihood is straightforward: model' s density function definition is plugged into the expression for the likelihood, and the gradient uphill is followed.

Main Challenge:

— Designing a model that can capture all of the complexity of the data to be generated while still maintaining computational tractability.

— This challenge is confronted by designing:

*(i) tractable explicit models,

*(ii) models that admit tractable approximations to the likelihood and its gradients.

Deep Generative Models: A Taxonomy

1.1) Tractable Explicit Models

Fully Visible Belief Networks (FVBNs)

- These models use the chain rule of probability to decompose a probability distribution over an $n-$dimensional vector $\boldsymbol{x}$ into a product of one-dimensional probability distributions:

$$p_{model}(\boldsymbol{x}) = \prod_{i=1}^{n} p_{model}(x_i | x_1, \ldots, x_{i-1})$$

Main drawbacks:

- Samples must be generated one entry at a time: first x_1, then x_2, …, so the cost of generating a sample is $O(n)$.

- In modern FVBNs (WaveNet, see next slide), the distribution over each x_i is computed by a deep neural network.

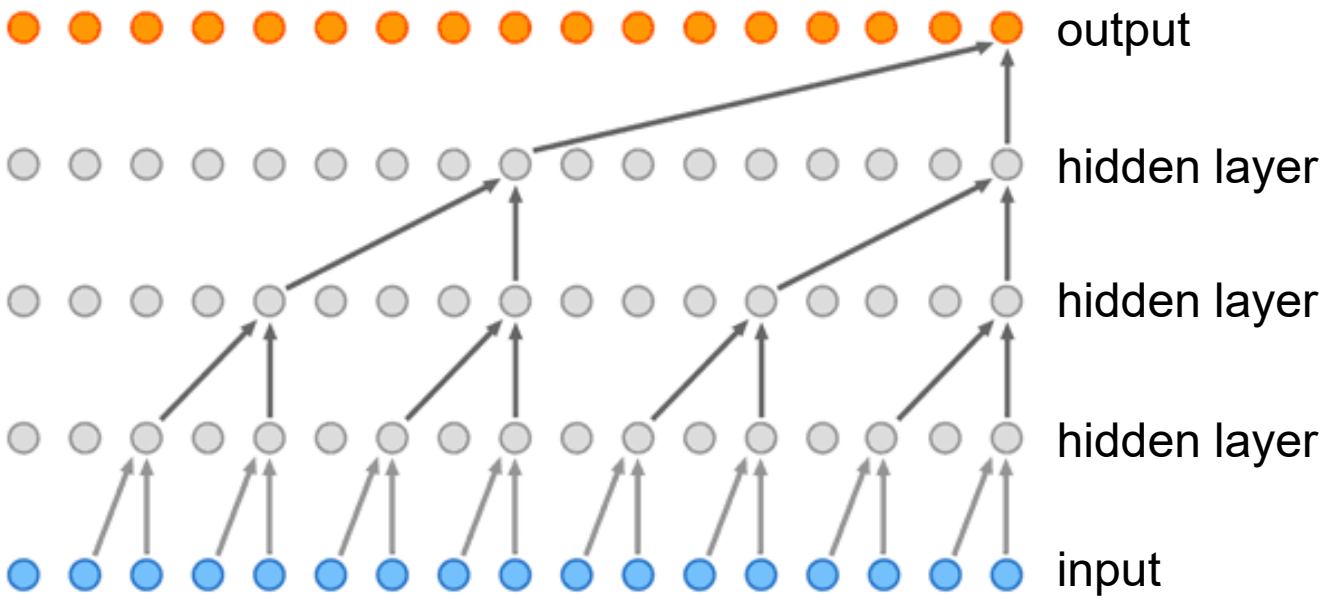
- Each of these n steps involves a nontrivial amount of computation.

- These steps cannot be parallelized.

Deep Generative Models: A Taxonomy

1.1) Tractable Explicit Models

Fully Visible Belief Networks (FVBNs)



WaveNet

- High quality
- Slow sample generation
- One second of audio is synthesized in two minutes

Deep Generative Models: A Taxonomy

1.1) Tractable Explicit Models

Nonlinear Independent Component Analysis (ICA)

- Based on defining continuous, nonlinear transformations between two different spaces.
- Given a vector of latent variables $\boldsymbol{z}$, and a continuous, differentiable, invertible transformation g.
- $g(\boldsymbol{z})$ yields a sample from the model in $\boldsymbol{x}$ space, then:

$$p_x(\boldsymbol{x}) = p_z(g^{-1}(\boldsymbol{x})) \left| det \left(\frac{\partial g^{-1}(x)}{\partial x} \right) \right|.$$

- The density p_x is tractable if the density p_z and the determinant of the Jacobian of g^{-1} is tractable.

Deep Generative Models: A Taxonomy

1.1) Tractable Explicit Models

Nonlinear Independent Component Analysis (ICA)

Samples generated by [Dinh et al., 2016] trained on 64x64 ImageNet images

Main drawbacks:

- Impose restrictions on the choice of g.
- Invertibility requirement means that $\boldsymbol{z}$ and $\boldsymbol{x}$ must have the same dimension.

Deep Generative Models: A Taxonomy

1.2) Explicit Models Requiring Approximation

Variational Autoencoder (VAE)

— Uses deterministic approximation, and defines a lower bond:

$$\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta}) \leq \log p_{model}(\boldsymbol{x};\boldsymbol{\theta})$$

— Maximizing $\mathcal{L}$ guarantees to obtain at least as high a value of the log-likelihood as it does of $\mathcal{L}$. It is possible to define an $\mathcal{L}$ that is computationally tractable even when the log-likelihood not.

Main drawbacks:

— When too weak of an approximate posterior distribution or too weak of a prior distribution is used, the gap between $\mathcal{L}$ and the true likelihood can result in p_{model} learning something other than the true p_{data}.

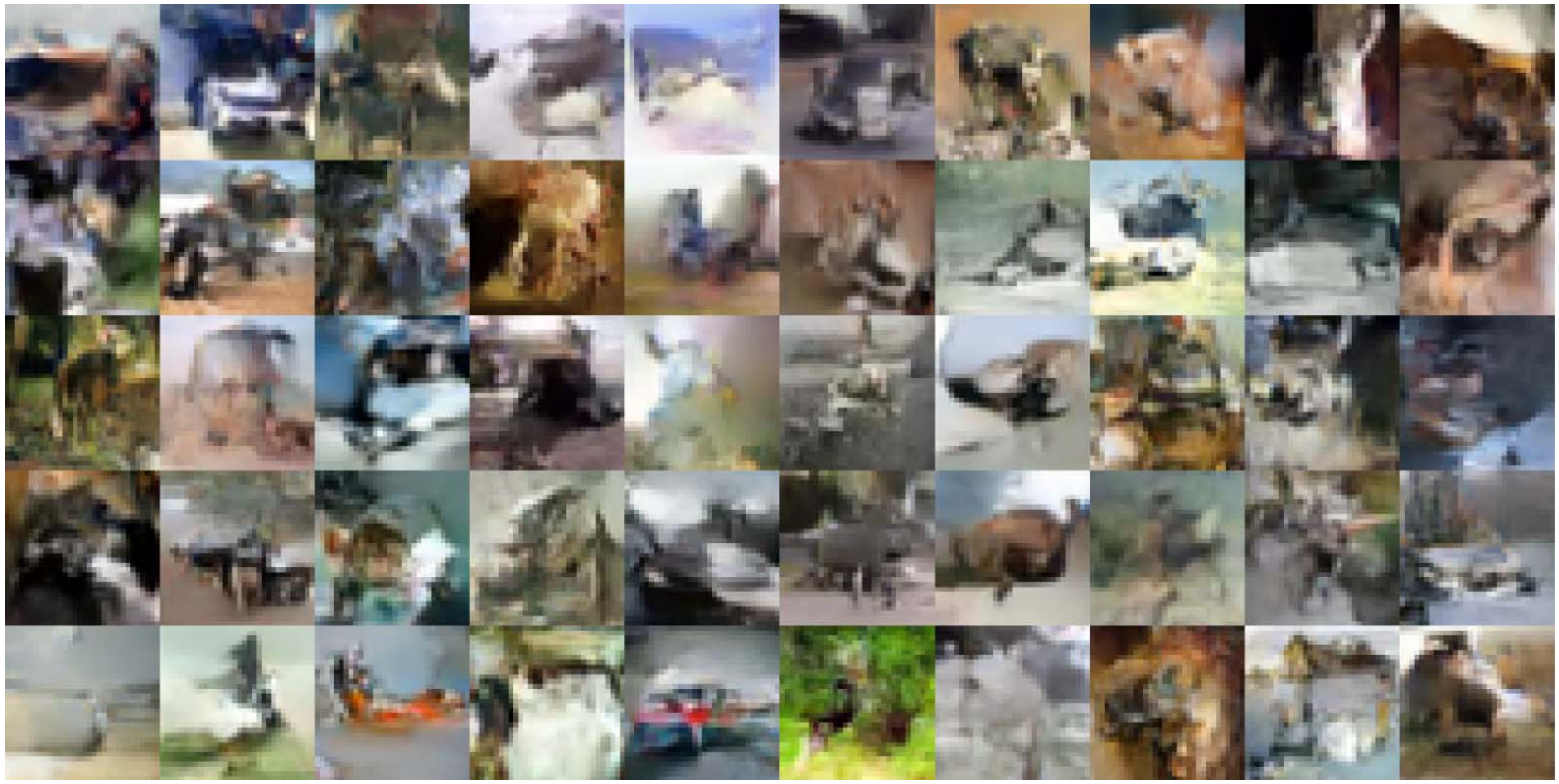
— More difficult to optimize.

— Practically, variational methods obtain good likelihood, but producing lower quality samples (see next slides for examples generated by VAE).

Deep Generative Models: A Taxonomy

1.2) Explicit Models Requiring Approximation

Variational Autoencoder (VAE)



Samples drawn from a VAE trained on the CIFAR-10 dataset.

Deep Generative Models: A Taxonomy

1.2) Explicit Models Requiring Approximation

Boltzmann Machines

- Probability distributions over hidden ($\boldsymbol{h}$) and/or visible vectors ($\boldsymbol{v}$) are defined in terms of the energy function:

$$P(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})}$$

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})}$$

- Some models use both variational and Markov chain approximations. For example, deep Boltzmann machines (see next slide).
- A Markov chain generates samples by repeatedly drawing a sample $\boldsymbol{x}' \sim \boldsymbol{q}(\boldsymbol{x}' | \boldsymbol{x})$, where the transition operator $\boldsymbol{q}$.

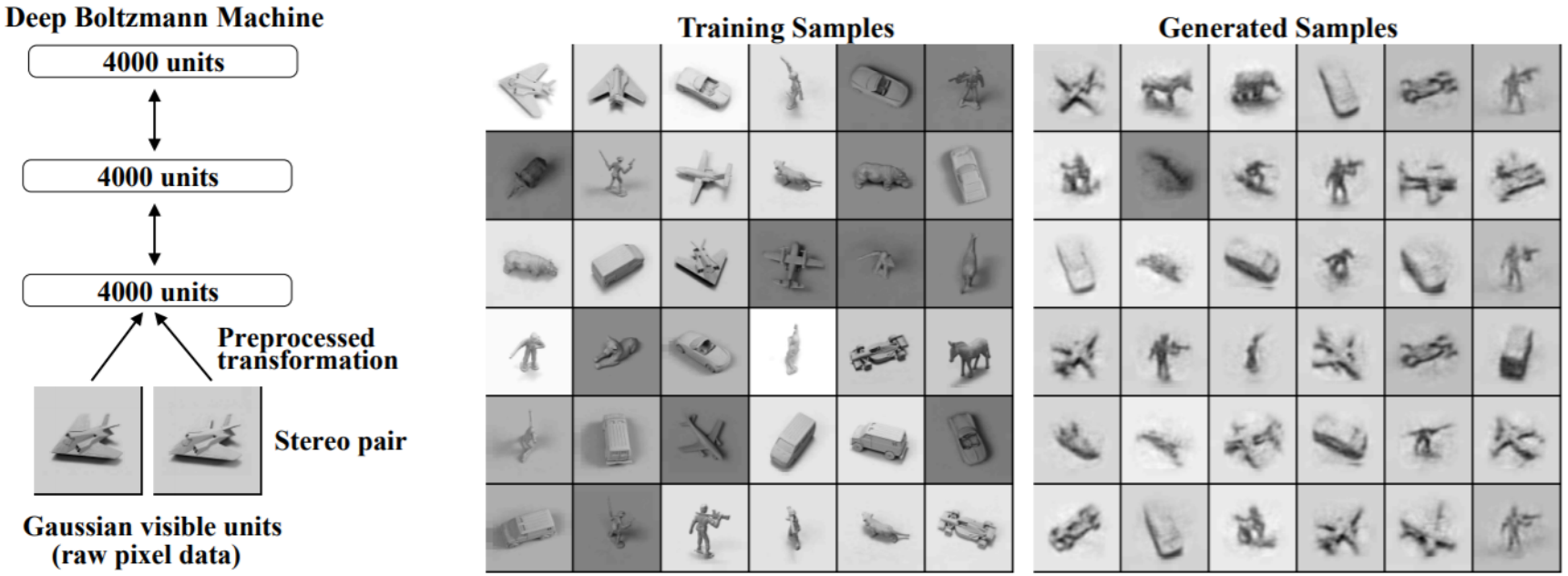
Main drawbacks:

- The underlying Markov chain approximation is not scaled to problems like ImageNet.
- Markov Chain methods, for both training and generating samples.

Deep Generative Models: A Taxonomy

1.2) Explicit Models Requiring Approximation

Boltzmann Machines



(left) The architecture of deep Boltzmann machine. (right) Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

Comparison: GANs vs other generative models

- GANs can generate samples in parallel, instead of using runtime proportional to the dimensionality of $\boldsymbol{x}$ (compared with FVBNs).

- The design of the generator function has very few restrictions:

  *an advantage relative to Boltzmann machines, for which few probability distributions admit tractable Markov chain sampling.

  *an advantage relative to nonlinear ICA, for which the generator must be invertible and the latent code $\boldsymbol{z}$ must have the same dimension as the samples $\boldsymbol{x}$.

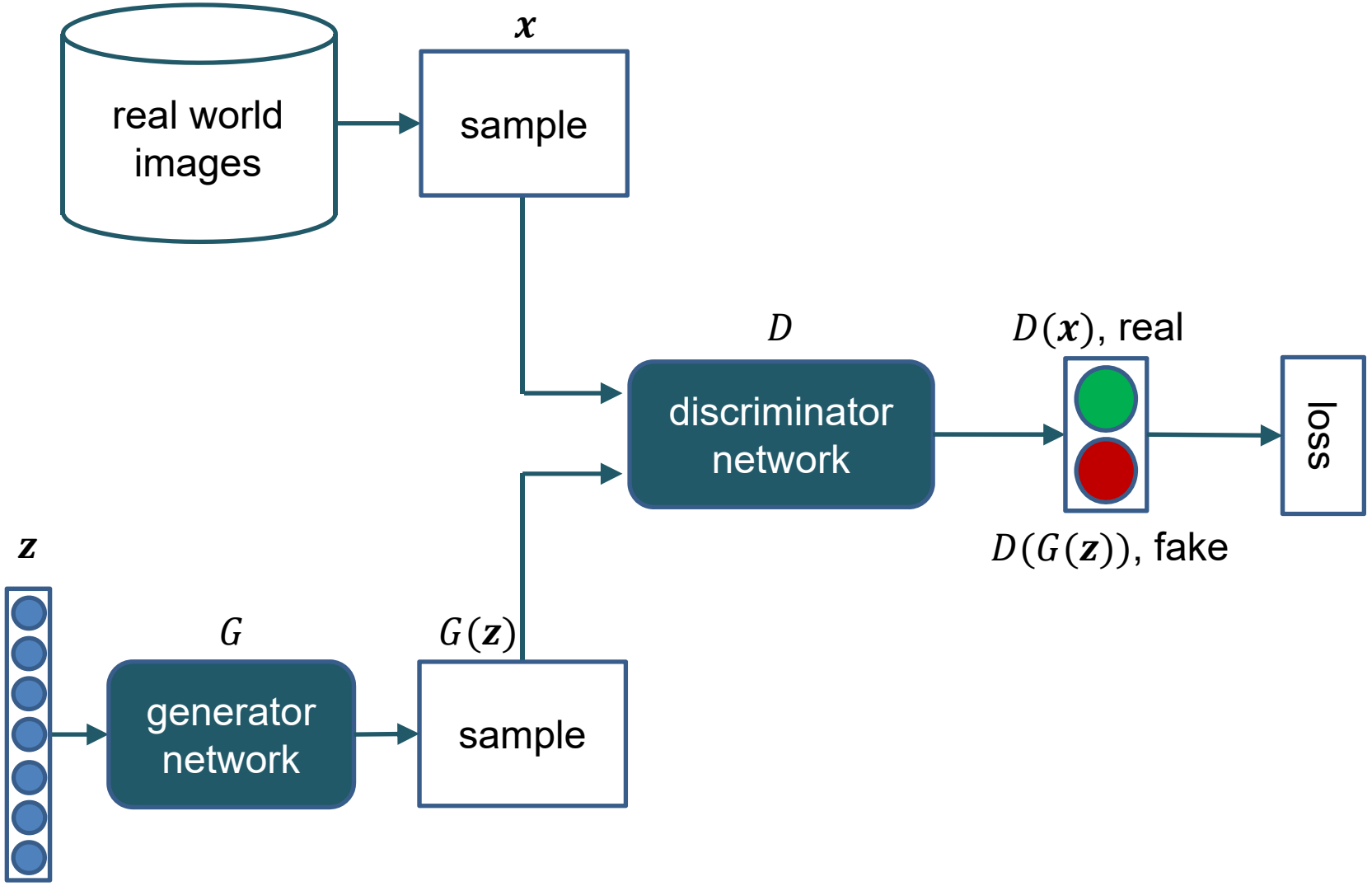
- No Markov chains are needed (compared with Boltzmann machines).

- No variational bound is needed, GANs are known to be asymptotically consistent.

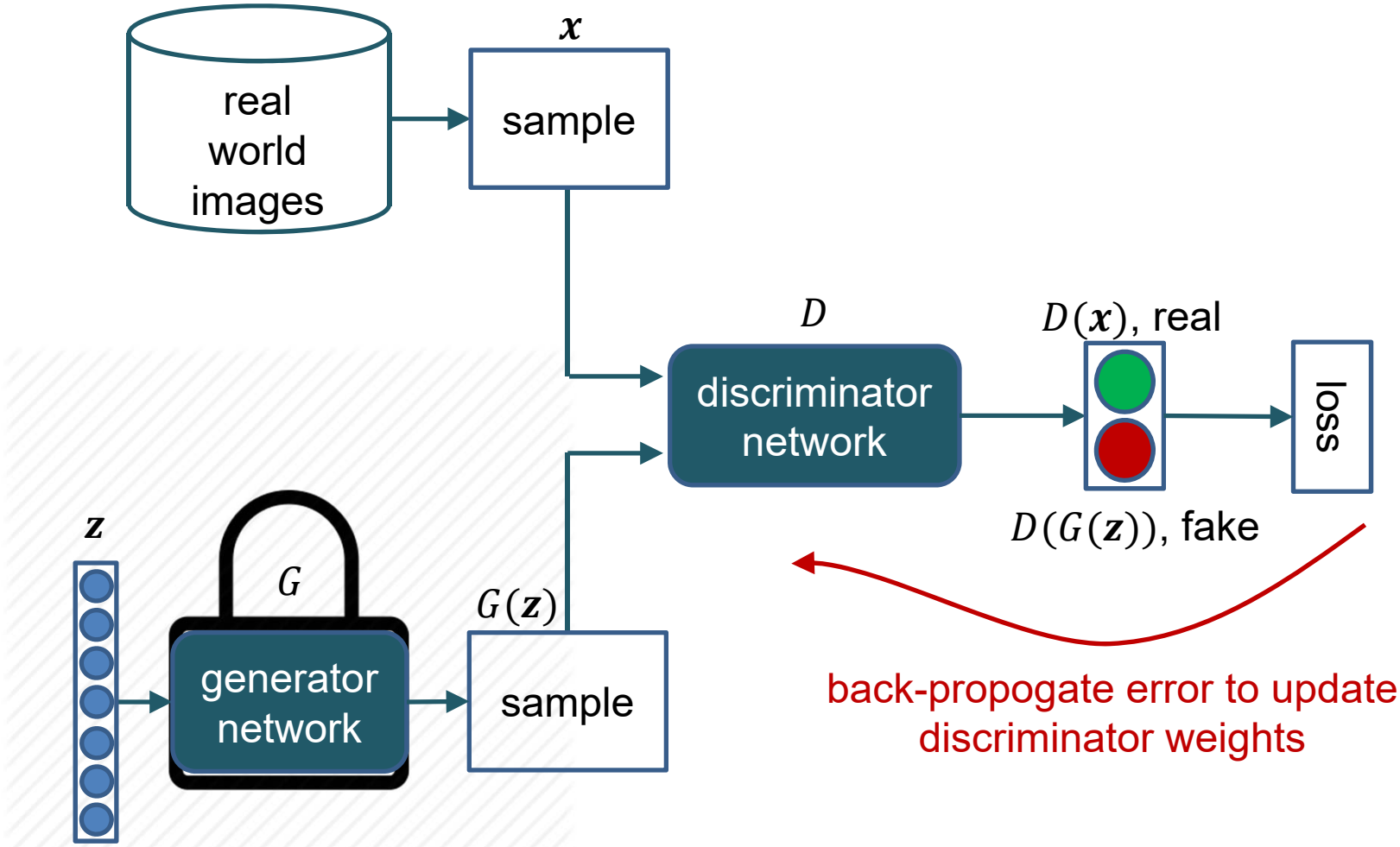
- GANs are regarded as producing better samples than other methods.

GAN Architecture

- GAN composes of two networks: the **generator** and the **discriminator**.

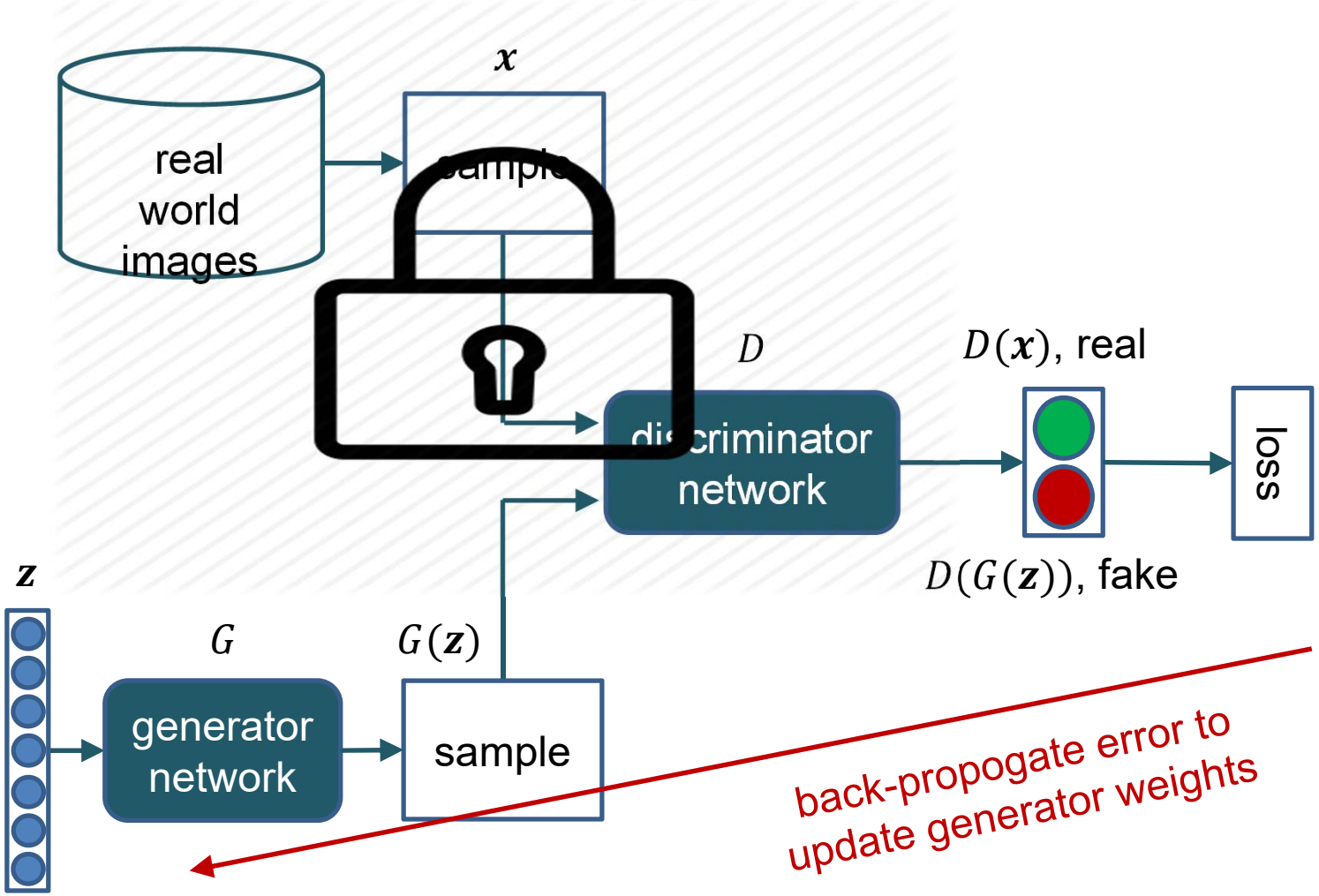


GAN Training (Discriminator)



The training process of the “Discriminator Network”. Error is back-propagated over the discriminator network only, in order to update discriminator weights, while the Generator Network is locked.

GAN Training (Generator)



The training process of the “Generator Network”. Error is back-propagated over the generator network only, in order to update generator weights, while the Discriminator Network is locked.

GAN Training

- Once objective functions (discriminator + generator) are defined, they are jointly learnt by alternating gradient descent.
- We fix the generator model' s parameters and perform a single iteration of gradient descent on the discriminator using the real and the generated images.
- Then we switch sides. Fix the discriminator and train the generator for another iteration.
- We train both networks in alternating steps until the generator produces good quality images.

- We define GAN as a 'minimax game' which G wants to minimize V, while D wants to maximize it:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

GAN Training (Discriminator)

Discriminator (D) Training: The discriminator outputs a value $D(\boldsymbol{x})$ indicating the chance that $\boldsymbol{x}$ is a real image.

- Our objective is to maximize the chance to recognize real images as real and generated images as fake, i.e., the max. likelihood of the observed data (to measure the loss, we use cross-entropy as in most Deep Learning: $p \log q$).
- The objective:

recognize real images better

recognize generated images better

$$\max_D V(D) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator (G) Training: The generator wants the model to generate images with the highest possible value of $D(\boldsymbol{x})$ to fool the discriminator.

- The objective:

optimize G that can fool the discriminator the most.

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

GAN Training

- We define GAN as a 'minimax game' which G wants to minimize V, while D wants to maximize it:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- The Nash Equilibrium of this particular game is achieved at:

$$P_{data}(x) = P_{gen}(x), \forall x$$

$$D(x) = \frac{1}{2}, \forall x$$

Generative Adversarial Networks (GANs)

Generator (G):

- Given a certain *label*, tries to predict *features*.
- For instance: given an email is marked as spam, predicts (generates) the text of the email.

- Let's now look into how a generator creates images: First, we sample some noise $\boldsymbol{z}$ using a normal or uniform distribution. With $\boldsymbol{z}$ as an input, we use a generator G to create an image $\boldsymbol{x}$ ($\boldsymbol{x} = G(\boldsymbol{z})$).

$\boldsymbol{z} \sim \mathcal{N}(0,1)$

$$\boldsymbol{z} = \begin{bmatrix} -0.7 \\ 0.1 \\ -0.3 \\ 0.2 \\ 0.4 \\ . \\ . \end{bmatrix}$$

$G(\boldsymbol{z})$

$\boldsymbol{z} \sim U(0,1)$

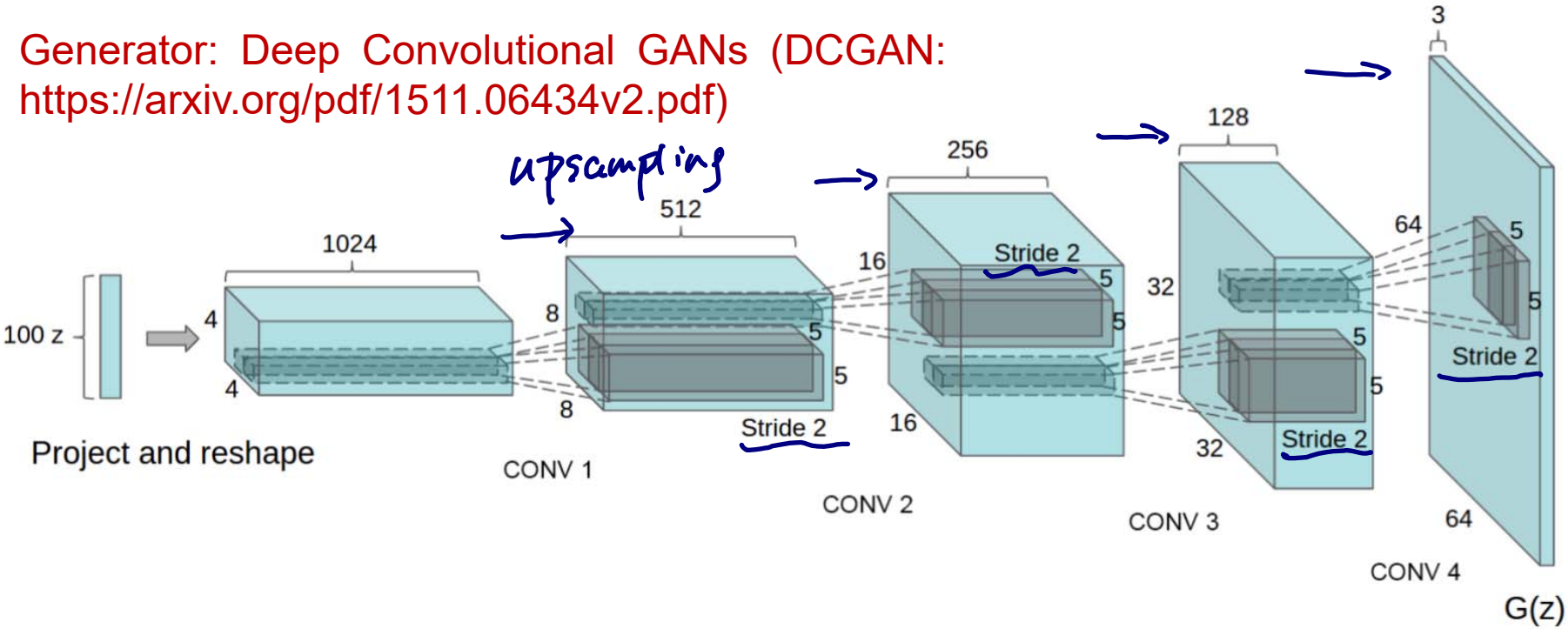
$$\boldsymbol{z} = \begin{bmatrix} -0.2 \\ 0.5 \\ -0.8 \\ 0.1 \\ 0.3 \\ . \\ . \end{bmatrix}$$

$G(\boldsymbol{z})$

- $\boldsymbol{z}$ represents the latent features of the images generated, for example, the colour and the shape.

Generative Adversarial Networks (GANs)

Generator: Deep Convolutional GANs (DCGAN: https://arxiv.org/pdf/1511.06434v2.pdf)



- DCGAN generator performs multiple transposed convolutions to up-sample $\boldsymbol{z}$ to generate the image $\boldsymbol{x}$. A 100 dimensional uniform distribution $\boldsymbol{z}$ is projected to a small spatial extent convolutional representation with many feature maps.

- A series of four fractionally-strided convolutions then convert this high level representation into a 64 × 64 pixel image. Notably, no fully connected or pooling layers are used.

Generative Adversarial Networks (GANs)

- However, a generator alone just creates random noise. Architecture-wise, the discriminator in GANs provides guidance to the generator on what images to create.

Discriminator (D):

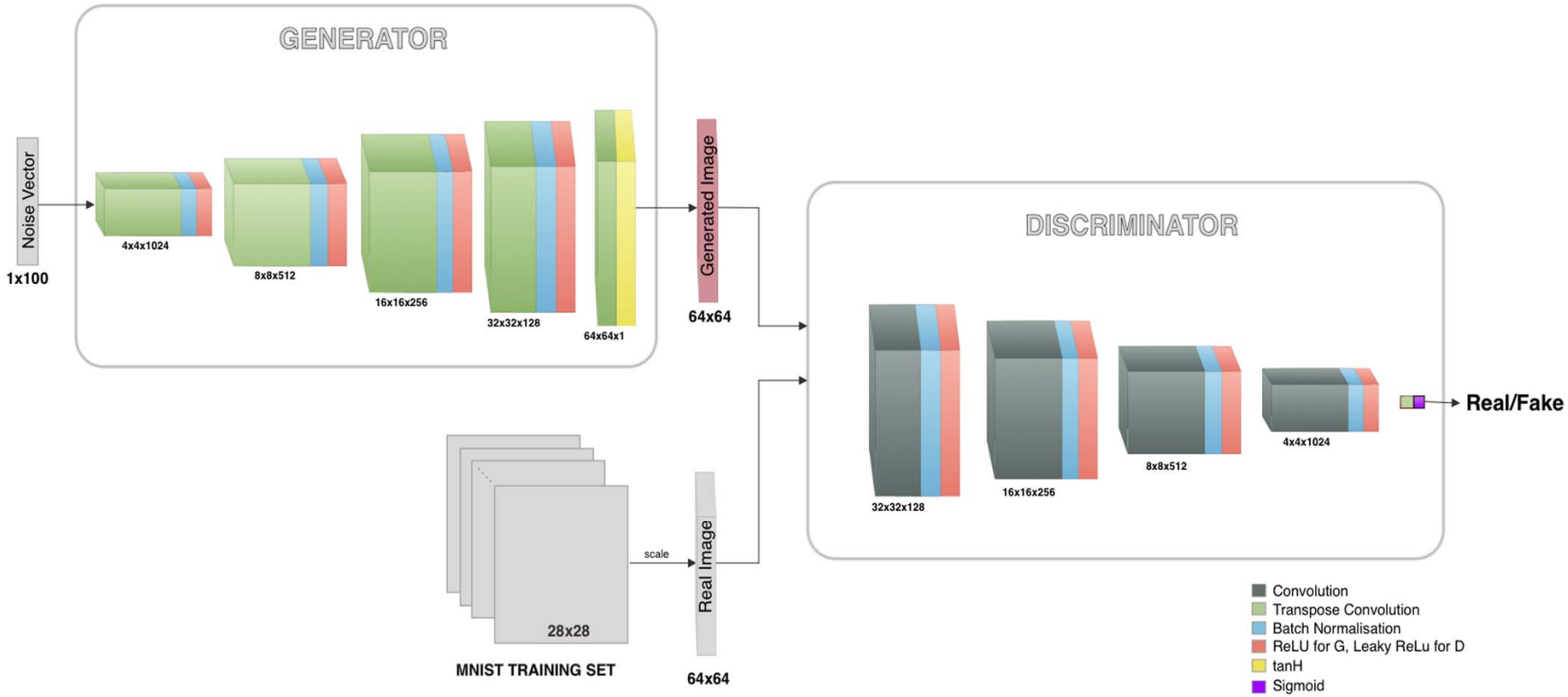
- Given the *features*, tries to predict the *label*.
- For instance: given the text of an email, predicts (discriminates) whether spam or not-spam.

- Discriminative models learn the boundary between classes.

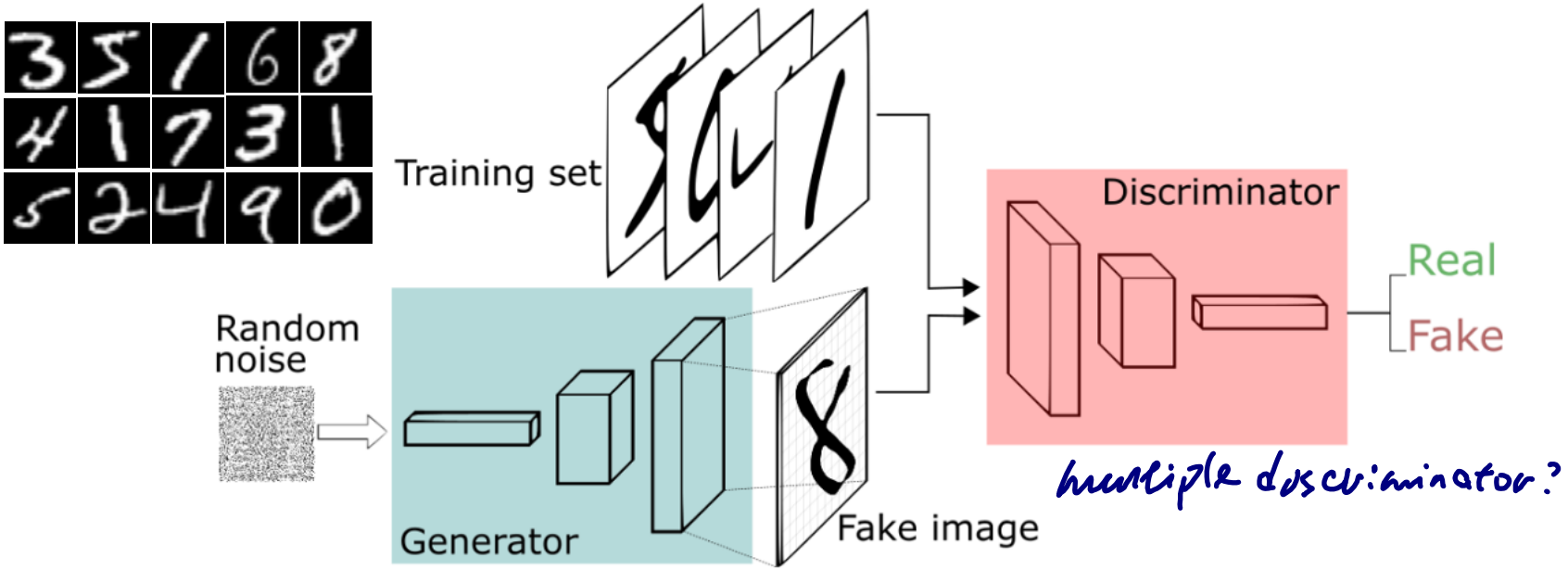
- In practice, a known dataset serves as the initial training data for the discriminator. Training the discriminator involves presenting it with samples from the dataset, until it reaches some level of accuracy. Samples synthesized by the generator are evaluated by the discriminator.

- Back-propagation is applied in both networks so that the generator produces better images, while the discriminator becomes more skilled at flagging synthetic images.

Detailed architecture of DCGAN



An Application: MNIST Dataset



- The generator creates new images that it passes to the discriminator.
- The goal of the Generator is to generate hand-written digits, which will be judged by the discriminator as real (authentic), even though they are fake.
- The goal of the Discriminator is to classify images coming from the generator as fake.

GAN Training (Complete Algorithm)

for numbers of training iterations **do**

for k steps **do**

*sample minibatch of m noise samples $\{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$ from noise prior $p_z(\boldsymbol{z})$.

*sample minibatch of m examples $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.

*update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D\left(x^{(i)}\right) + \log(1 - D\left(G(z^{(i)})\right))]$$

end for

*sample minibatch of m noise samples $\{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$ from noise prior $p_z(\boldsymbol{z})$.

*update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D\left(G(z^{(i)})\right))]$$

end for

GAN Training (Vanishing Gradient Problem)

- Gradient diminishing problem is encountered for the generator. The discriminator usually wins early against the generator. Distinguishing generated images from real images is easier at the early stages of training.

- This phenomena makes V approaches 0, i.e., $\log(1 - D(G(\boldsymbol{z}))) \to 0$. The gradient for the generator will also vanish which makes the gradient descent optimization very slow.

$$\nabla_{\theta_G} V(D, G) = \nabla_{\theta_G} \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

$$\nabla_a \log(1 - \sigma(a)) = \frac{-\nabla_a \sigma(a)}{1 - \sigma(a)} = \frac{-\sigma(a)(1 - \sigma(a))}{1 - \sigma(a)} = -\sigma(a)$$

where $\sigma(a)$ is a sigmoid function.

- Hence, gradient goes to 0 if D is confident, i.e., $D(G(\boldsymbol{z})) \to 0$.

- To improve that, the GAN provides an alternative function to backpropagate the gradient to the generator:

  Minimize $-\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(D(G(\boldsymbol{z})))]$ for the **generator** instead, keeping discriminator as it is.

$$\nabla_a -\log(\sigma(a)) = \frac{-\nabla_a \sigma(a)}{\sigma(a)} = \frac{-\sigma(a)(1 - \sigma(a))}{\sigma(a)} = -(1 - \sigma(a))$$

The Performance of GANs

How to improve the performance of GANs?

- Train with labels: Using labels results in an improvement in the subjective quality of the samples generated by the model. First observed in class-conditional CCGANs.

- One-sided label smoothing: shown to reduce the vulnerability of neural networks to adversarial examples.

- Virtual batch normalization: The main purpose of the model is to improve the optimization of the model.

- Balancing G and D: Somehow we need to balance G and D to prevent one from overpowering the other.

- Mode collapse: It could reach a equilibrium if the generator becomes so good at generating a single sample that it's indistinguishable from the real data. In a standard GAN, there's no incentive for a generator network to generate a wide distribution of samples.

Train with Labels

Conditional Generative Adversarial Nets (CGAN)

— CGAN is an extension of GAN, where both D and G receive an additional vector of information l as input. This input can contain information about the class of the training example $\boldsymbol{x}$. The loss thus is:

$$\min_G \max_D V(D,G)$$

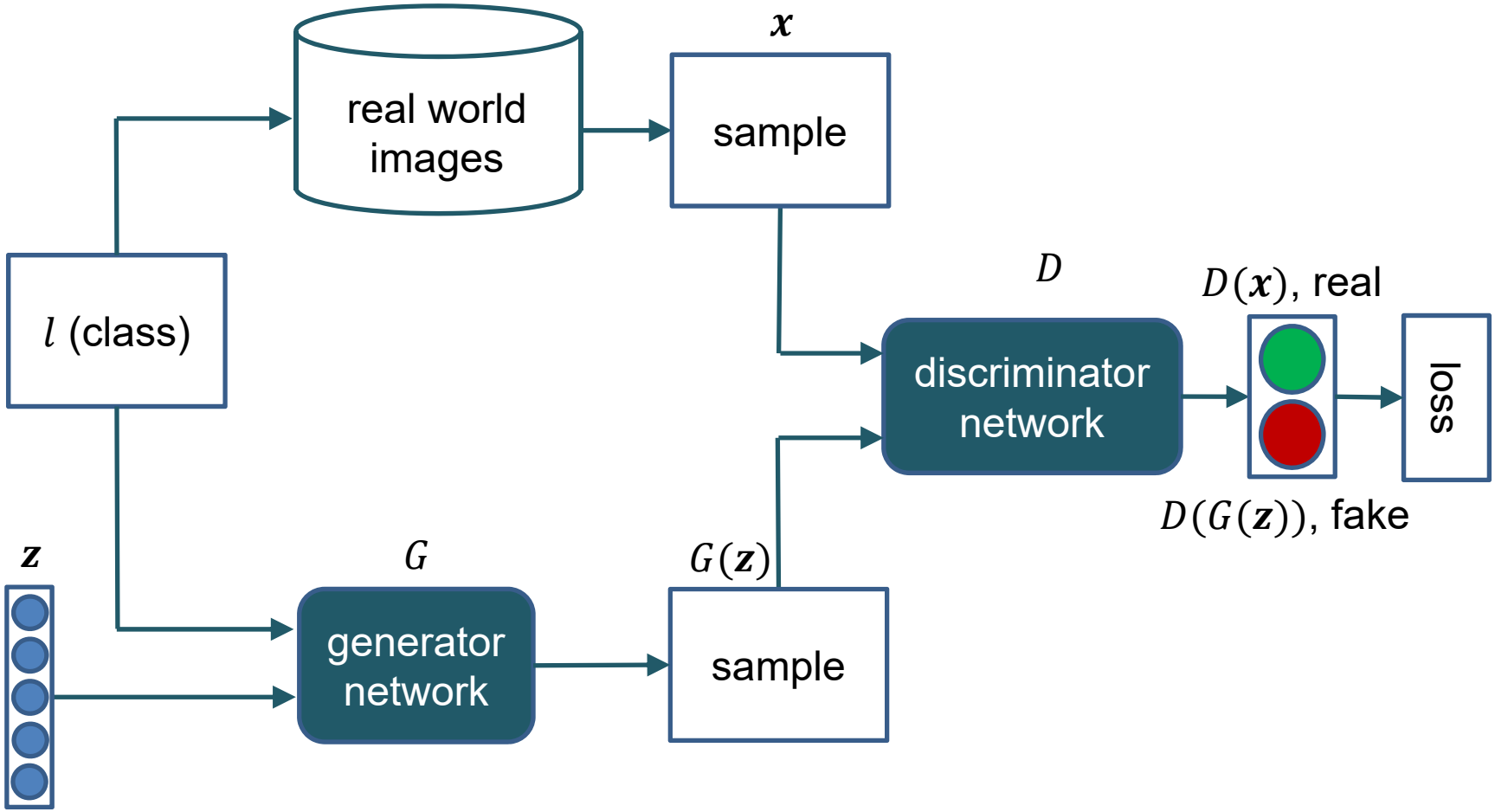
$$= \mathbb{E}_{\boldsymbol{x},l\sim p_{data}(\boldsymbol{x},l)}[\log D(\boldsymbol{x},l)] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z}),l\sim p_l(l)}[\log(1-D(G(\boldsymbol{z},l),l))]$$

where, $p_l(l)$ is the prior distribution over classes. The output of this model is controlled by the conditioning variable l.

Sample images generated by CGANs

Train with Labels

Conditional Generative Adversarial Nets (CGAN)



Train with Labels

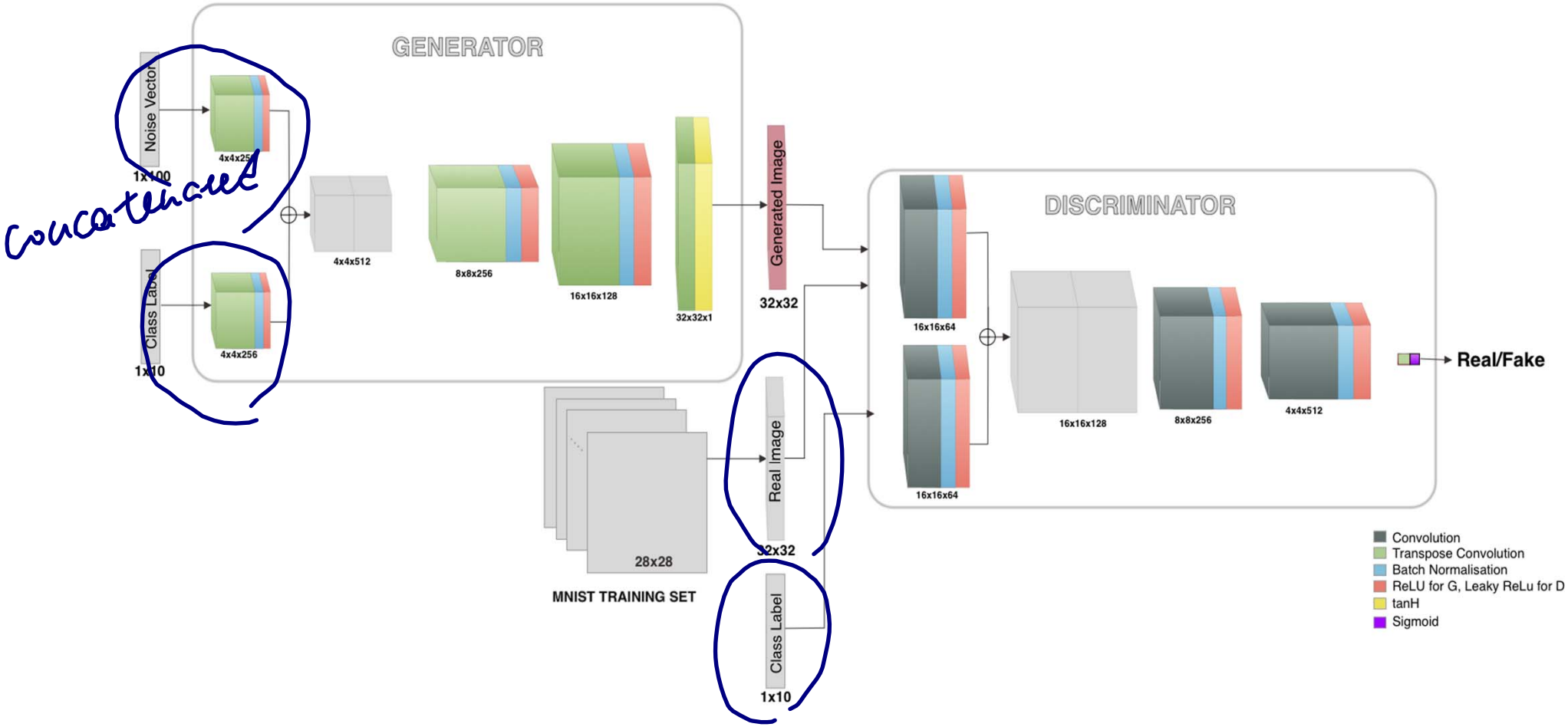
Class-Conditional Generative Adversarial Nets (CGAN)

— CCGAN utilizes the class label l, adding a 1-hot vector, as a conditioning variable for G and D.

— HOW is c integrated into G and D?

* c is passed through a layer.
* the output of the layer is reshaped into feature maps.
* the maps are then concatenated with the 1st layer maps.

Detailed architecture of CGAN



One-sided Label Smoothing

- Label smoothing replaces the 0 and 1 targets for a classifier with smoothed values, like 0.1 or 0.9.

- Replacing positive classification targets with α and negative targets with β, the optimal discriminator becomes:

α small, $\beta = 0$ avoid large gradient

$$D(\boldsymbol{x}) = \frac{\alpha p_{data}(\boldsymbol{x}) + \beta p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}$$

- When $\beta \neq 0$, erroneous samples from p_{model} have no incentive to move nearer to the data i.e. it reinforces current generator behaviour.

- Hence, we smooth only the positive labels to α, leaving negative labels set to 0.

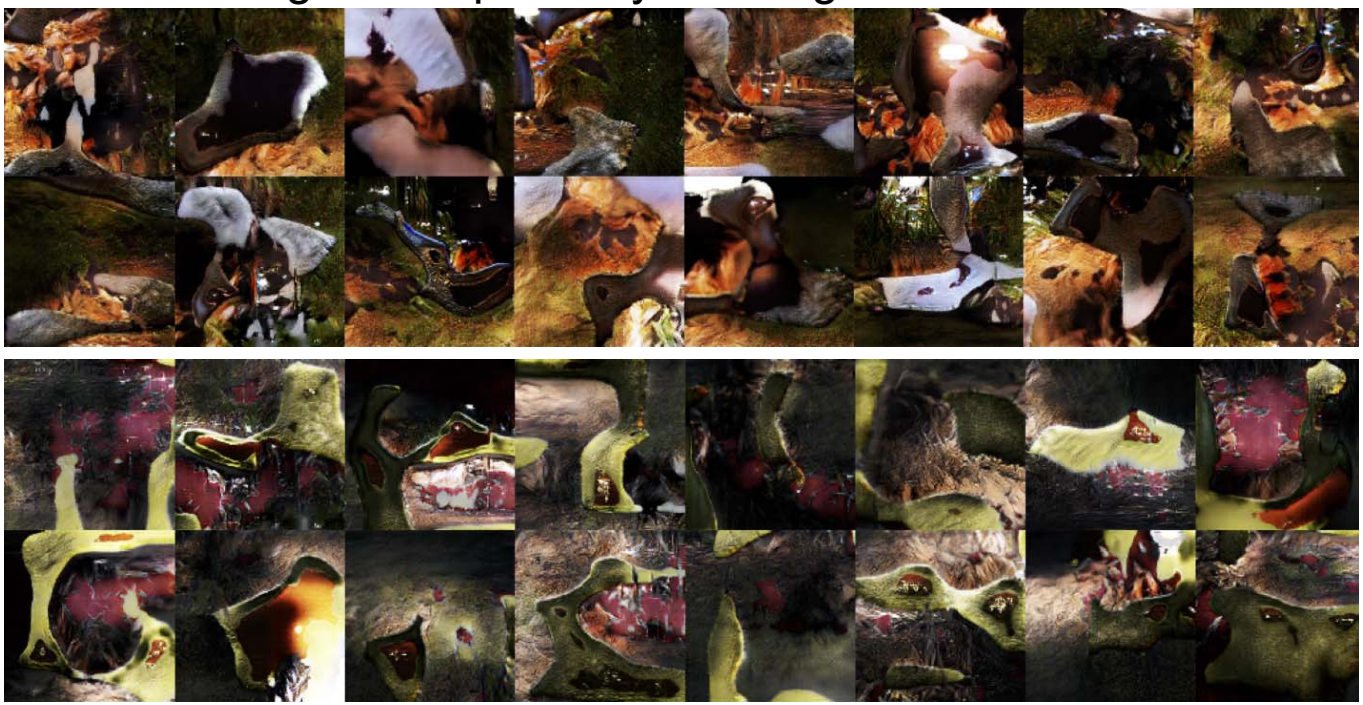
- This prevents discriminator from giving very large gradient signal to generator.

Virtual Batch Normalization

Batch Normalization:

- Given inputs $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$; compute mean and standard deviation of features of X, normalize features (subtract mean, divide by standard dev.).

- Normalization operation is part of the model;
  *Back-propagation computes the gradient through the normalization.
  *This avoids wasting time repeatedly learning to undo the normalization.



Batch normalization in G can cause strong intra-batch correlation

Virtual Batch Normalization

Reference Batch Normalization:

- Fix a reference batch $R = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_m\}$, and given new input $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, compute mean and standard deviation of features of R, normalize features of X using the mean and standard deviation from R (Note that though R does not change, the feature values change when the parameters change).

- Every $\boldsymbol{x}_i$ is always treated the same, regardless of which other examples appear in the minibatch.

Virtual Batch Normalization:

- Reference batch normalization can overfit to the reference batch. A partial solution is virtual batch normalization. Fix a reference batch $R = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_m\}$, and given new inputs $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;

- For each $\boldsymbol{x}_i$ in X;
  - *Construct a virtual batch VB containing both $\boldsymbol{x}_i$ and all of R.
  - *Compute mean and standard deviation of features of VB.
  - *Normalize the features of $\boldsymbol{x}_i$ using the mean and standard deviation from VB.

Balancing G and D

- Usually the discriminator overpowers the generator.
- G' s gradient can vanish when D is too accurate. (solve by) using a non-saturating cost.
- G' s gradient can become very large if D becomes too confident. (solve by) using one-sided label smoothing.
- One can also balance G and D by choosing model size: usually D is bigger and deeper than G.
- Sometimes run D more often than G. Mixed results.

How to Evaluate

- Train a classification network with softmax, using MNIST (we will do this in the next computer lab).

- Generate N (e.g. 100 images per class) images and measure the Inception score.

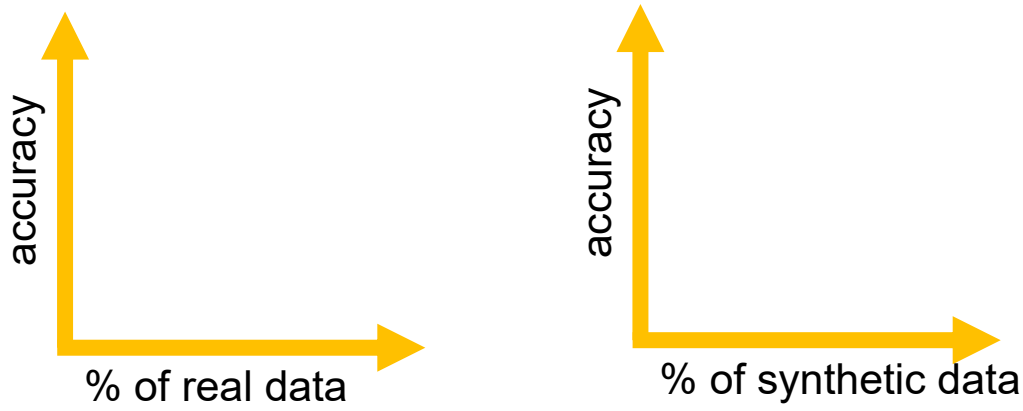
Inception Score (IS): measures the quality of generated images, and their diversity.

In GAN, we want the conditional probability *P(y|x)* to be highly predictable (low entropy). Given an image, we should know the object class easily. So we use an Inception network (i.e. a classification network independently trained using a real dataset) to classify the generated images and predict *P(y|x)*—where *y* is the label and *x* is the generated data. This reflects the quality of the images.

In CGAN, we use the class labels of images to generate and compare them with the predicted labels, measuring the success rate.

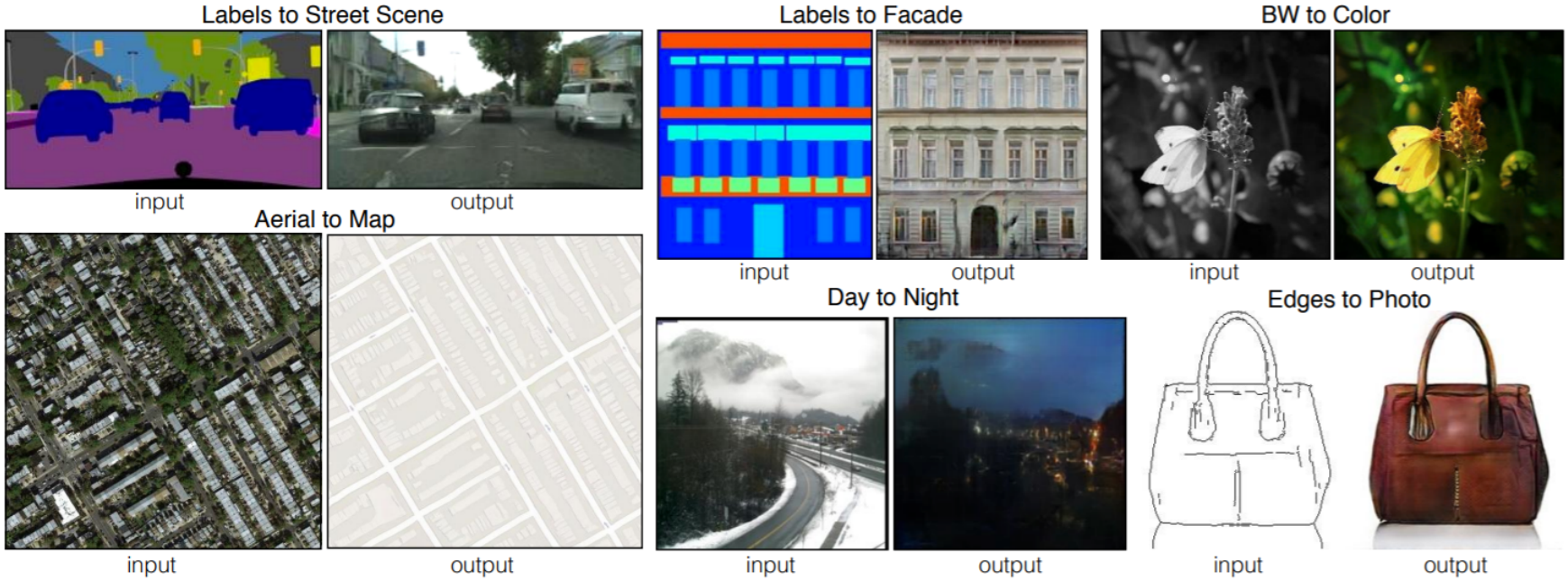
How to Evaluate

– Re-train the classifier with generated samples, and the full real training data or its small subset (e.g. 10% or even few samples per class), see if the validation accuracy is improved.



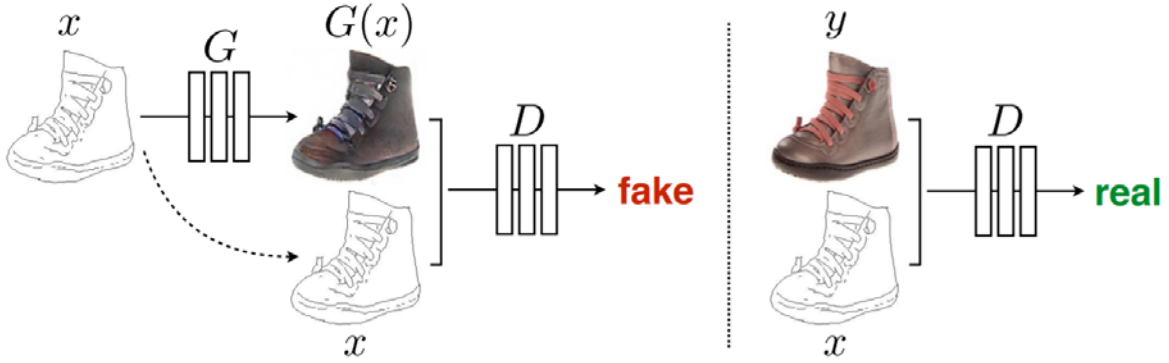
Applications

Image-to-Image Translation (Isola et al, CVPR17)

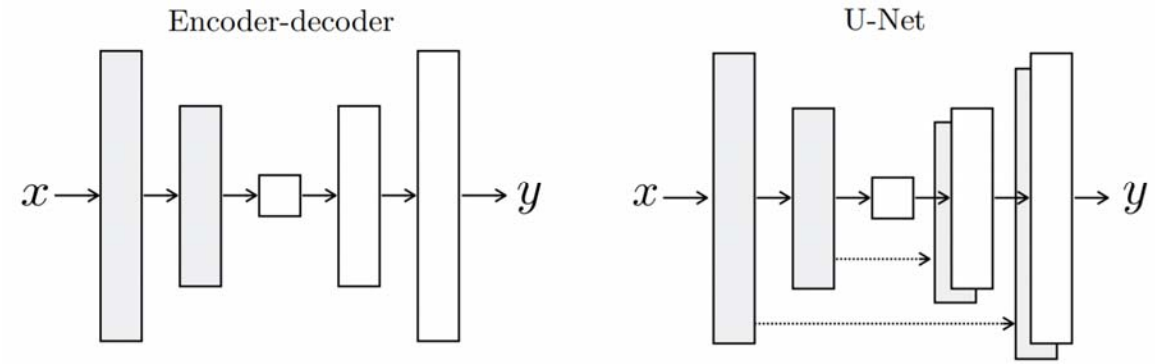


Many problems in image processing, graphics, and vision involve translating an input image into a corresponding output image.
Conditional adversarial nets are a general-purpose solution that appears to work well on a wide variety of these problems.
In each case the same architecture and objective are used, and are simply trained on different data.

Image-to-Image Translation (Isola et al, CVPR17)

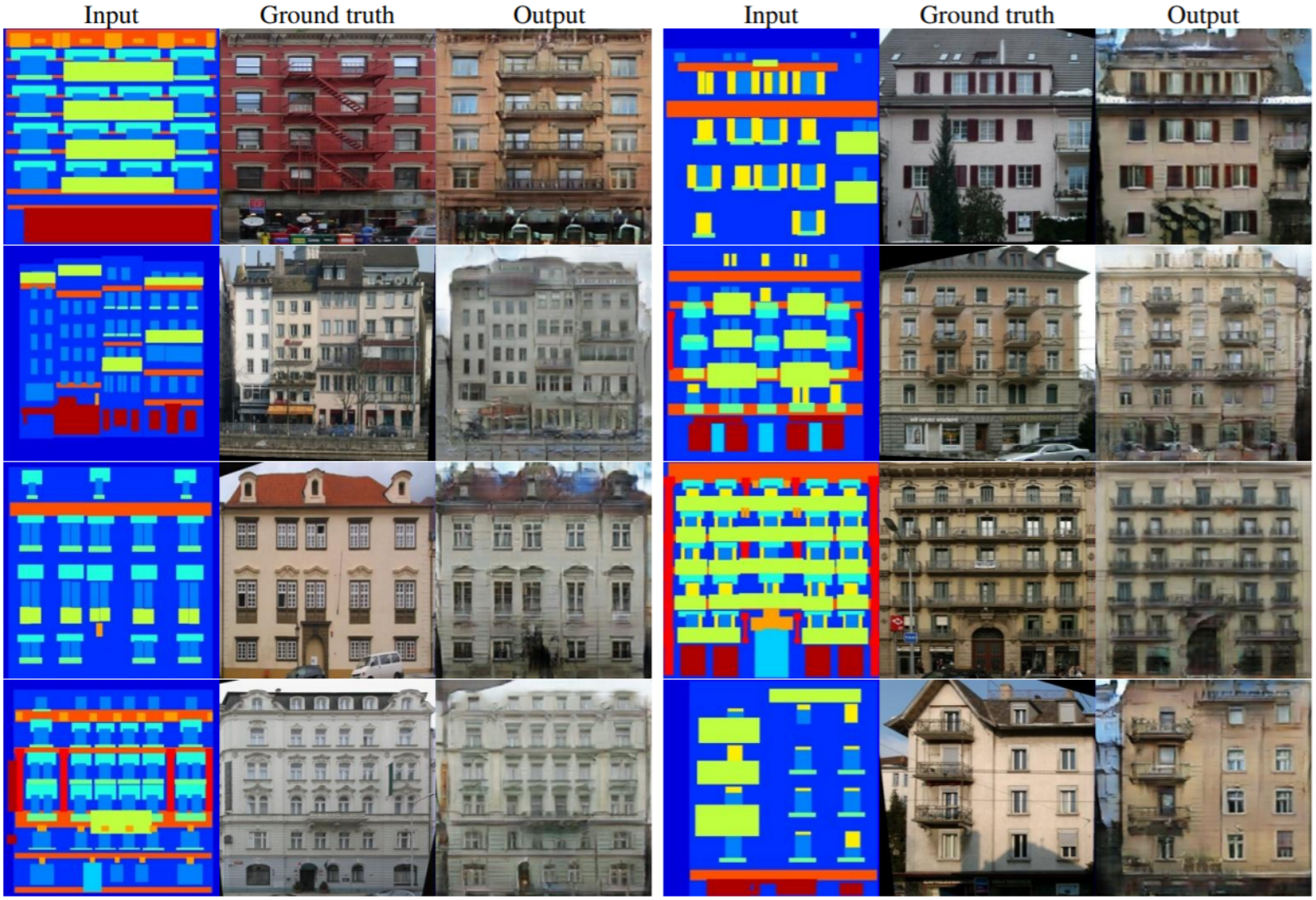


Training a conditional GAN to map edges→photo. The discriminator D learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator G learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.



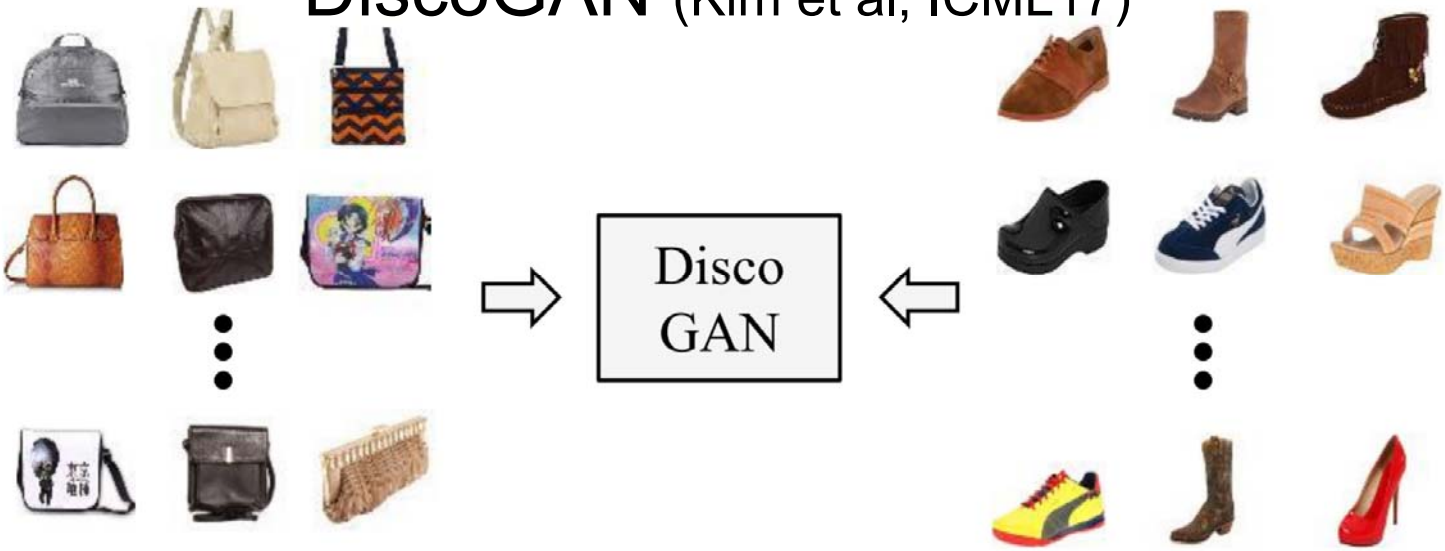
Two choices for the architecture of the generator. The "U-Net" is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

Image-to-Image Translation (Isola et al, CVPR17)



Example results of the method on facades labels→photo, compared to ground truth.

DiscoGAN (Kim et al, ICML17)



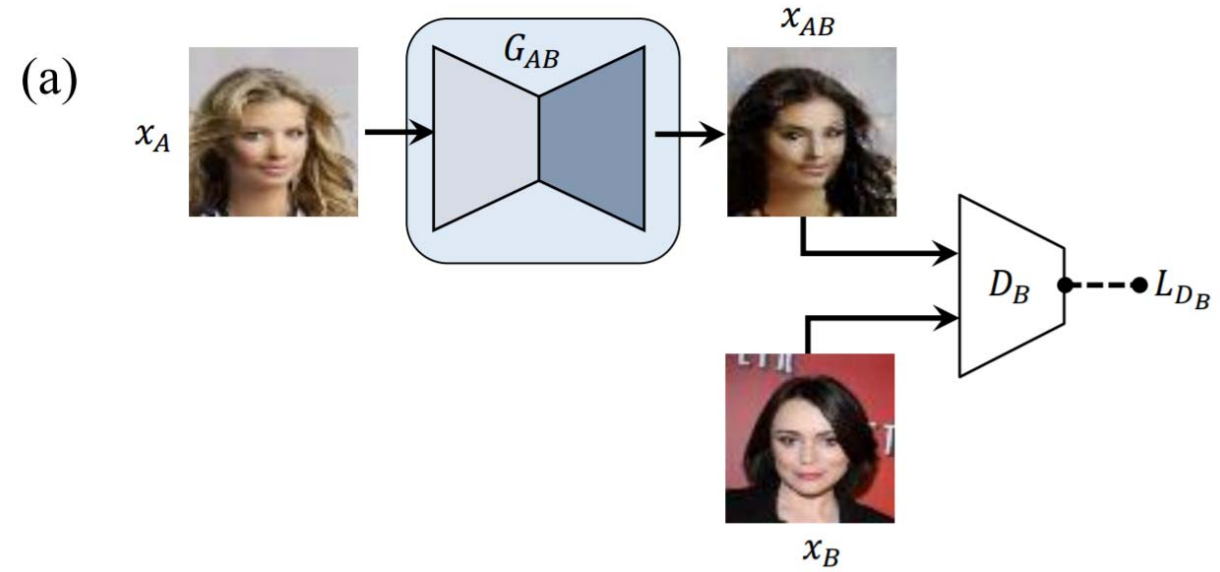
(a) Learning cross-domain relations without any extra label

(b) Handbag images (input) & Generated shoe images (output)

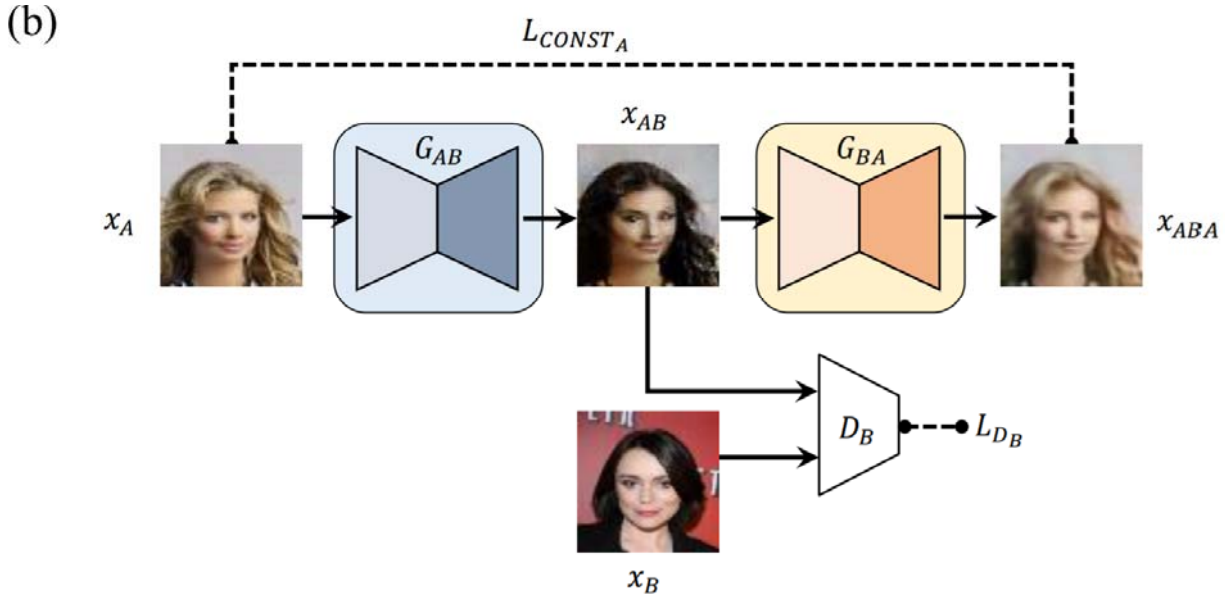
(c) Shoe images (input) & Generated handbag images (output)

DiscoGAN trains with two independently collected sets of images and learns how to map two domains without any extra label

DiscoGAN (Kim et al, ICML17)

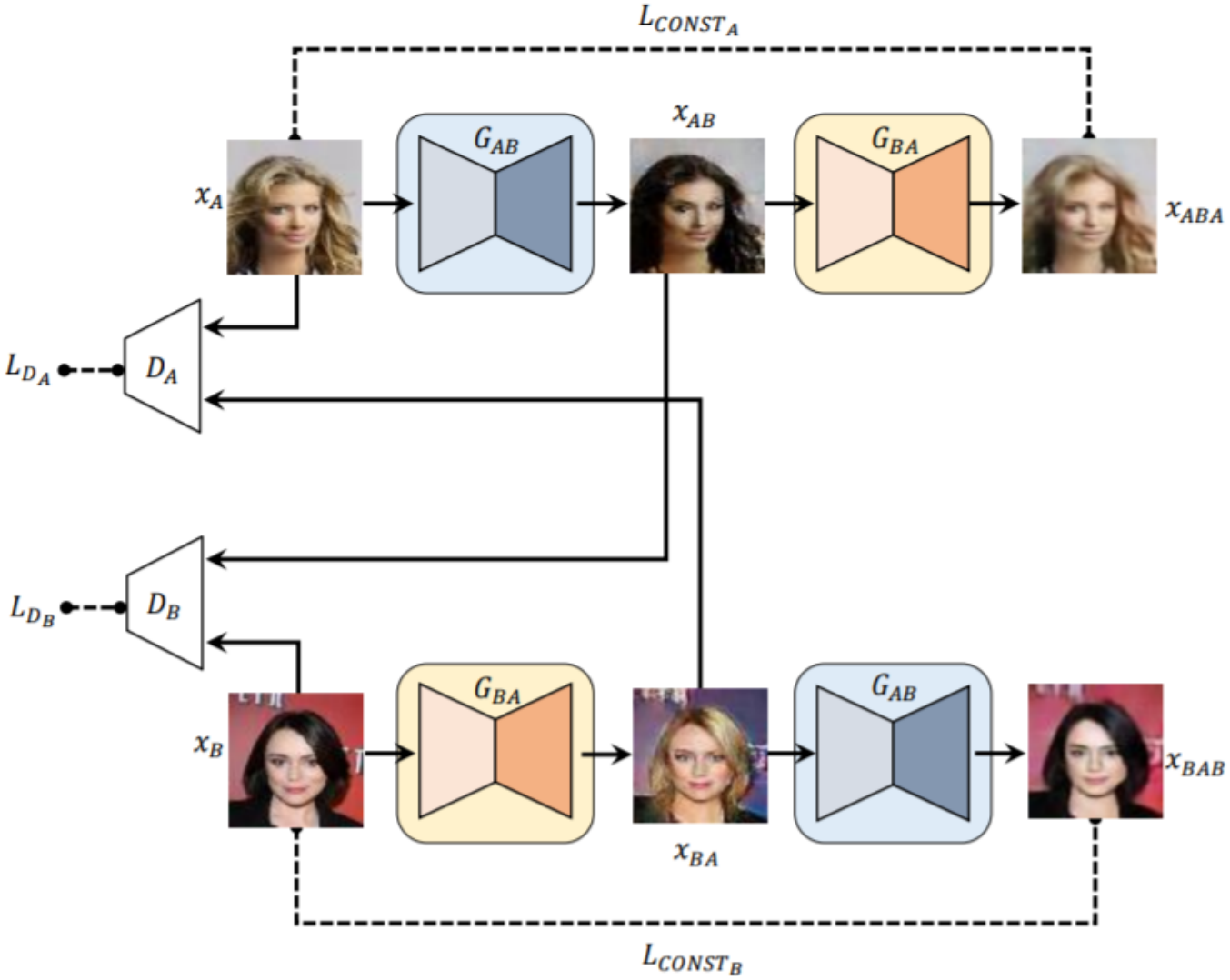


(a) standard GAN



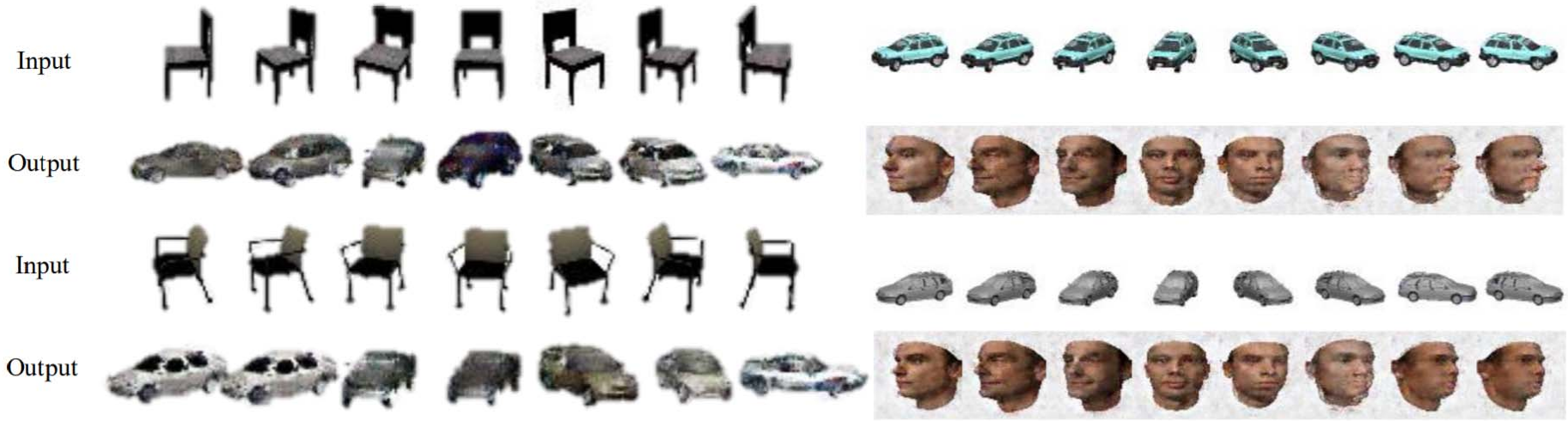
(b) GAN with a reconstruction loss

DiscoGAN (Kim et al, ICML17)



DiscoGAN discovers relations between two unpaired, unlabeled datasets.

DiscoGAN (Kim et al, ICML17)

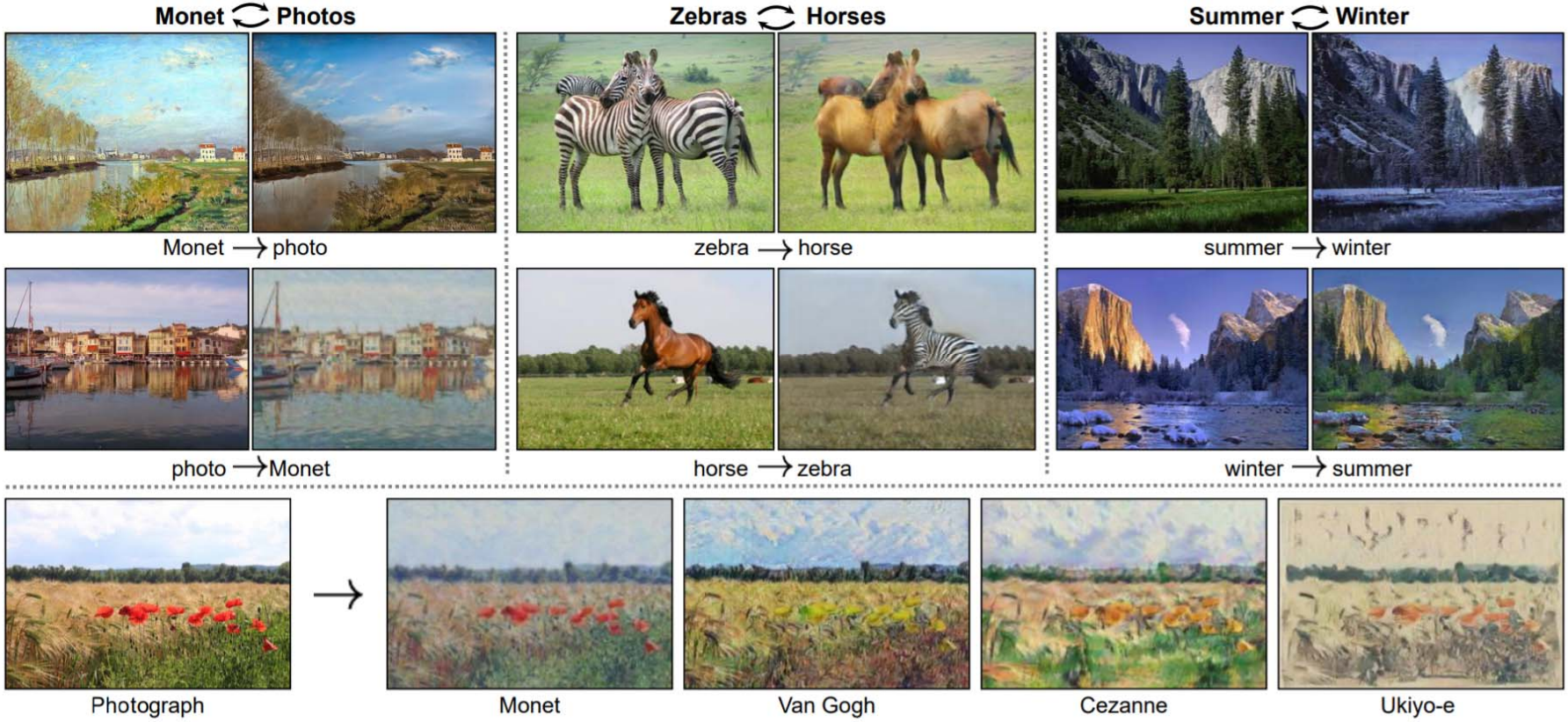


(a) Chair to Car

(b) Car to Face

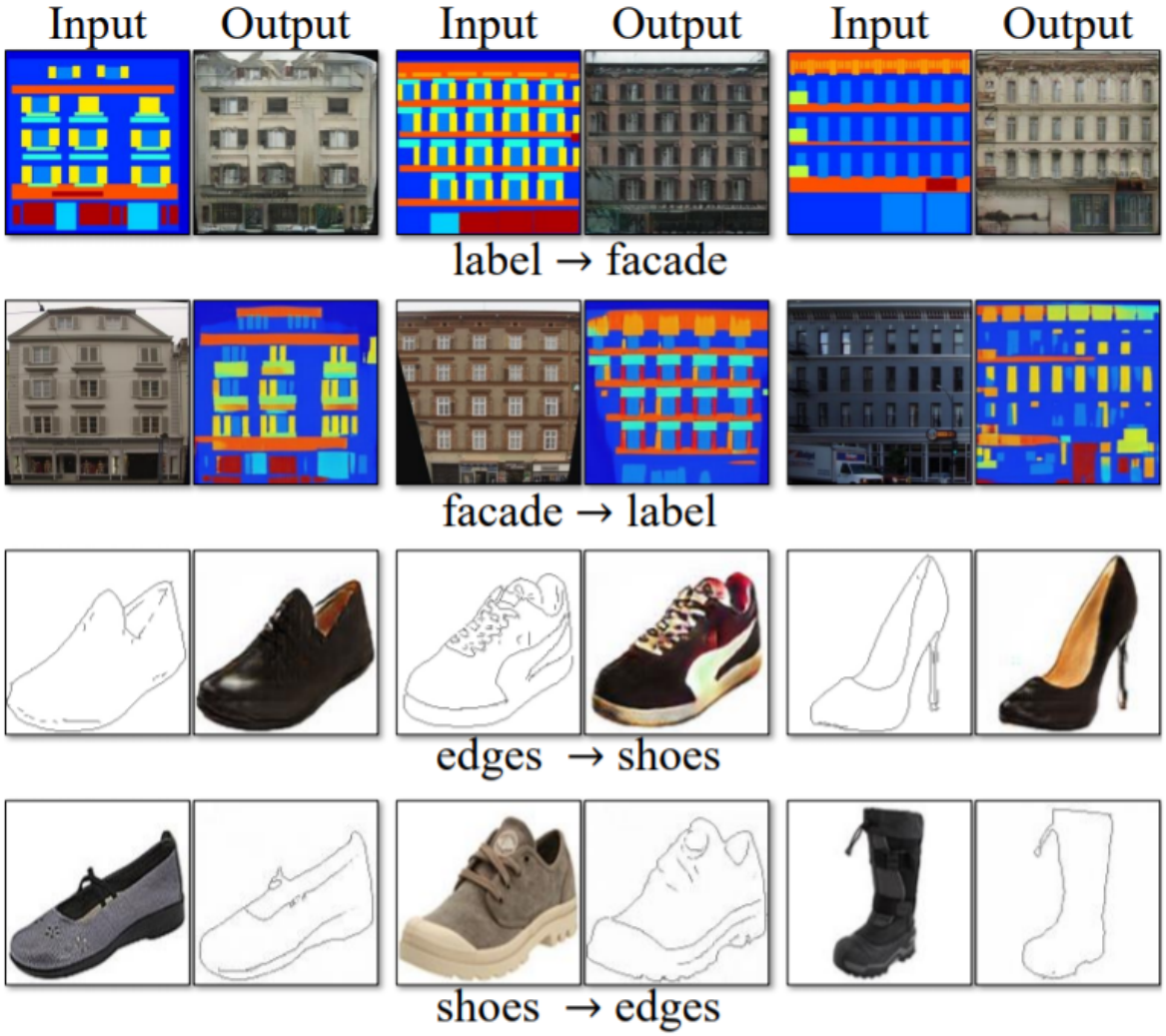
Discovering relations of images from visually very different object classes. (a) chair to car translation. DiscoGAN is trained on chair and car images (b) car to face translation. DiscoGAN is trained on car and face images. Our model successfully pairs images with similar orientation.

CycleGAN (Zhu et al, ICCV17)



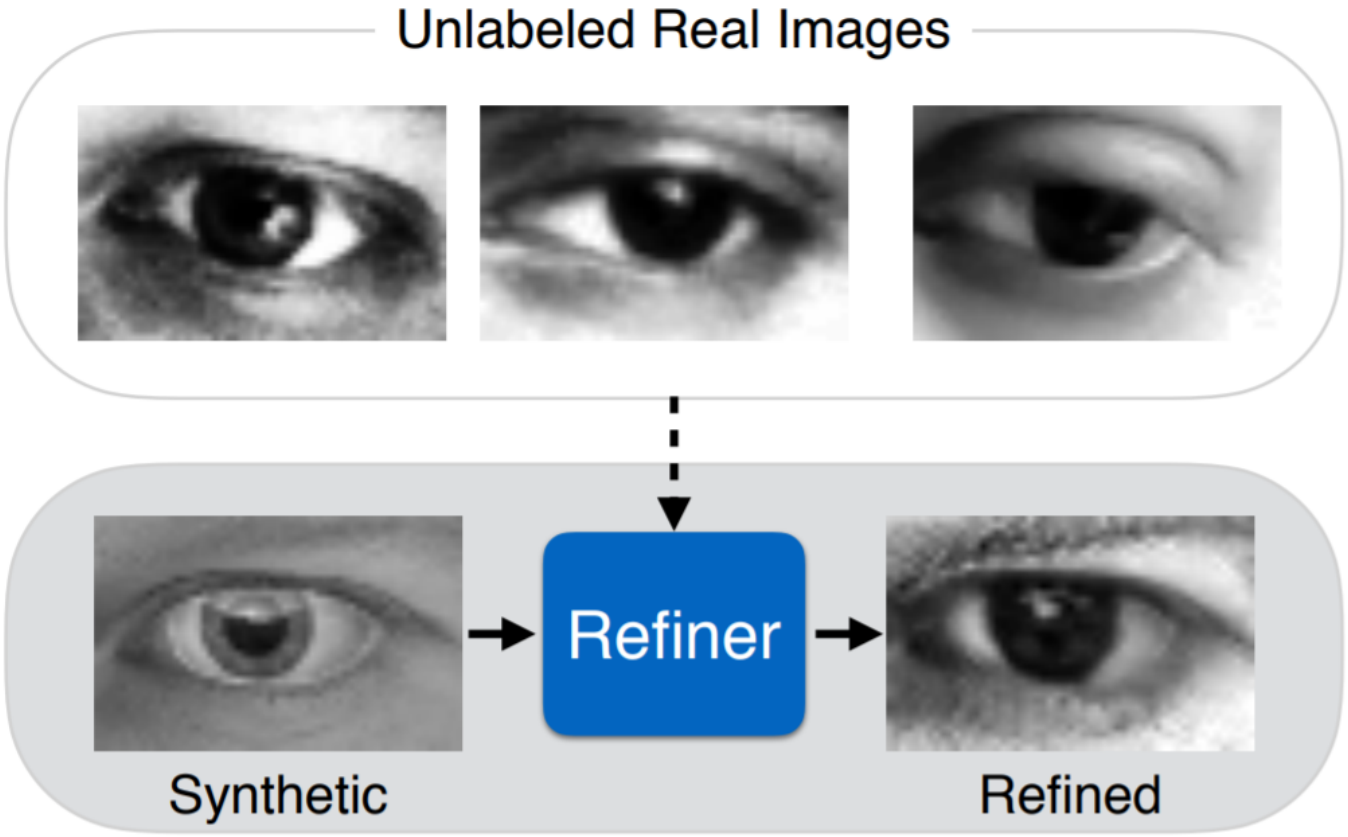
Given any two unordered image collections, CycleGAN learns to automatically "translate" an image from one into the other and vice versa: (left) Monet paintings and landscape photos from Flickr; (center) zebras and horses from ImageNet; (right) summer and winter Yosemite photos from Flickr. Example application (bottom): using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

CycleGAN (Zhu et al, ICCV17)



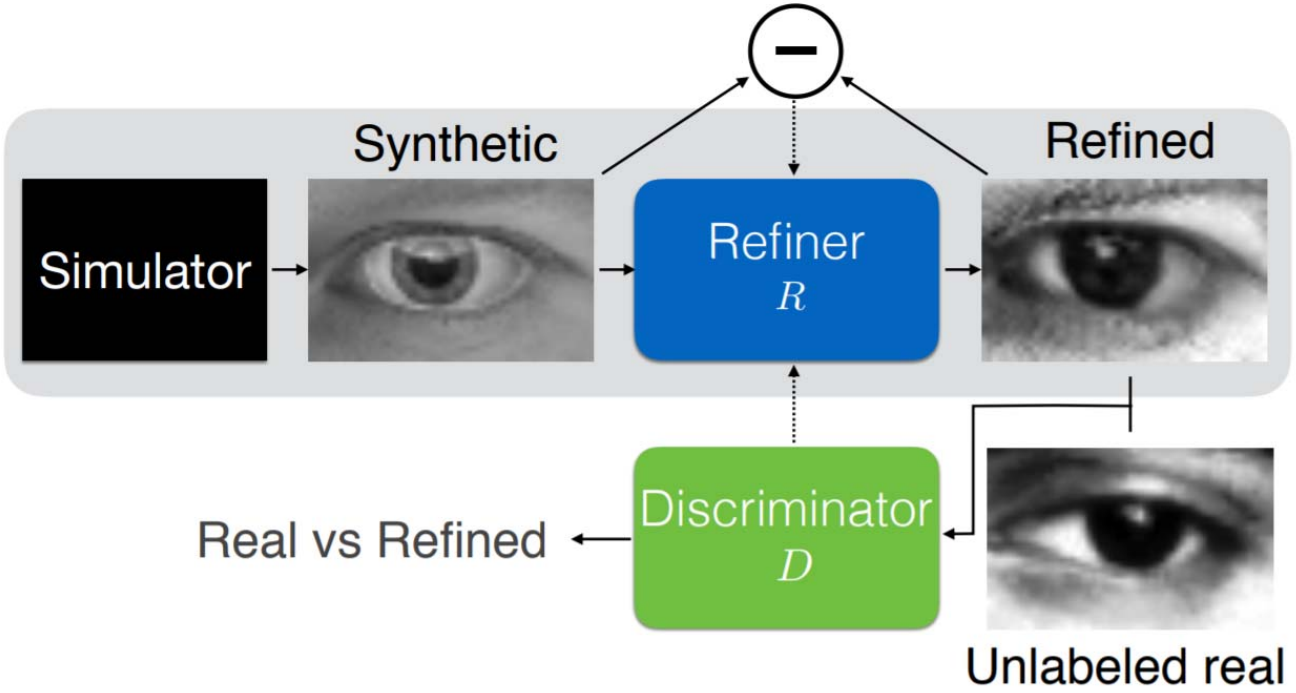
Example results of CycleGAN on paired datasets, such as architectural labels↔photos and edges↔shoes.

SimGAN (Shrivastava et al, CVPR17)



Simulated+Unsupervised (S+U) learning. The task is to learn a model that improves the realism of synthetic images from a simulator using unlabeled real data, while preserving the annotation information.

SimGAN (Shrivastava et al, CVPR17)



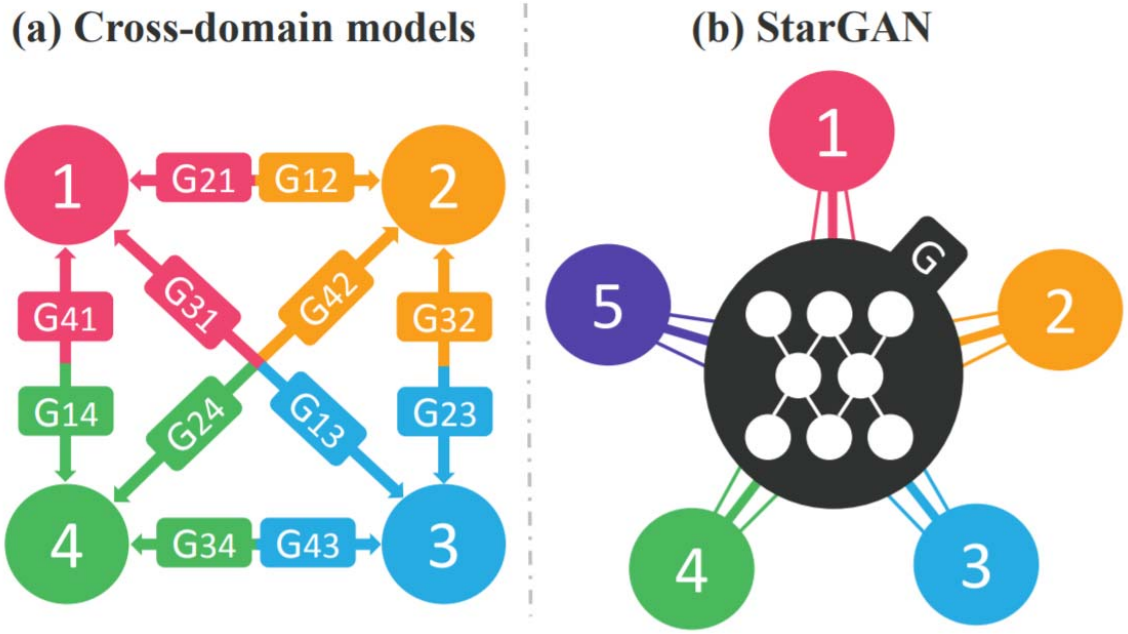
Overview of SimGAN: The output of the simulator is refined with a refiner neural network, R, that minimizes the combination of a local adversarial loss and a 'self-regularization' term. The adversarial loss 'fools' a discriminator network, D, that classifies an image as real or refined. The self-regularization term minimizes the image difference between the synthetic and the refined images. The refiner network and the discriminator network are updated alternately.

SimGAN (Shrivastava et al, CVPR17)

Example output of SimGAN for the UnityEyes gaze estimation dataset. (Left) real images from MPIIGaze. The refiner network does not use any label information from MPIIGaze dataset at training time. (Right) refinement results on UnityEye.

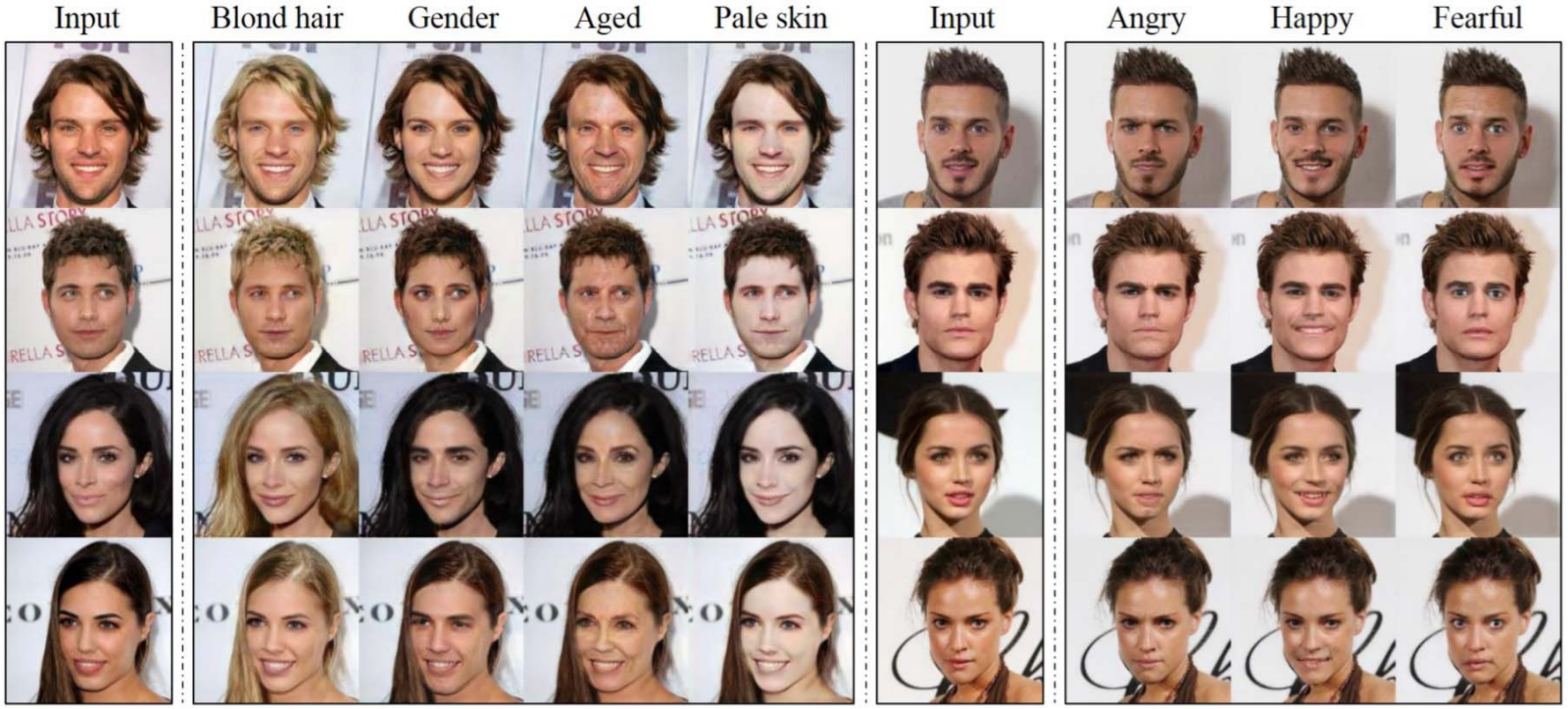
Using a history of refined images for updating the discriminator. (Left) synthetic images; (middle) result of using the history of refined images; (right) result without using a history of refined images.

StarGAN (Choi et al, CVPR18)



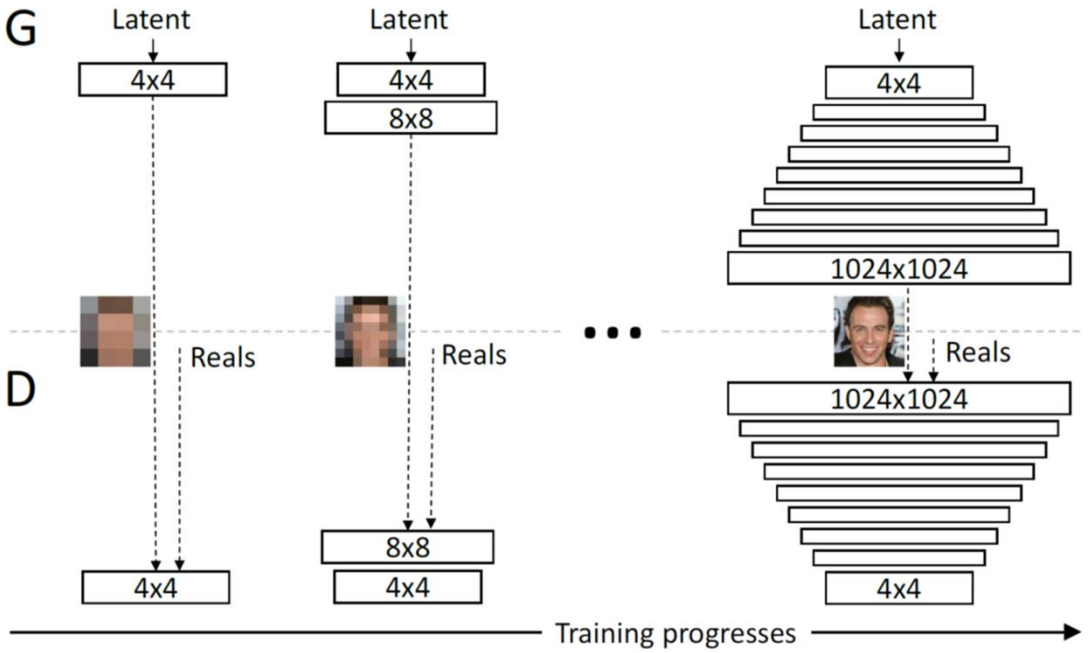
Comparison between cross-domain models and StarGAN. (a) To handle multiple domains, cross-domain models should be built for every pair of image domains. (b) StarGAN is capable of learning mappings among multiple domains using a single generator. The figure represents a star topology connecting multi-domains.

StarGAN (Choi et al, CVPR18)



Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

Progressive Growing of GANS (Karras et al, ICLR18)



Our training starts with both the generator (G) and discriminator (D) having a low resolution of 4×4 pixels.

As the training advances, we incrementally add layers, increasing the spatial resolution.

All existing layers remain trainable throughout the process.

Here N × N refers to convolutional layers operating on N × N spatial resolution.

This allows stable synthesis in high resolutions and also speeds up training.

Progressive Growing of GANS (Karras et al, ICLR18)

We show example images generated using progressive growing at 1024 × 1024.

Progressive Growing of GANS (Karras et al, ICLR18)

Top: Our CELEBA-HQ results. Next five rows: Nearest neighbors from the training data, based on feature distance. We used five VGG layers on the cropped area.