Introduction to Generative Adversarial Network (GAN)

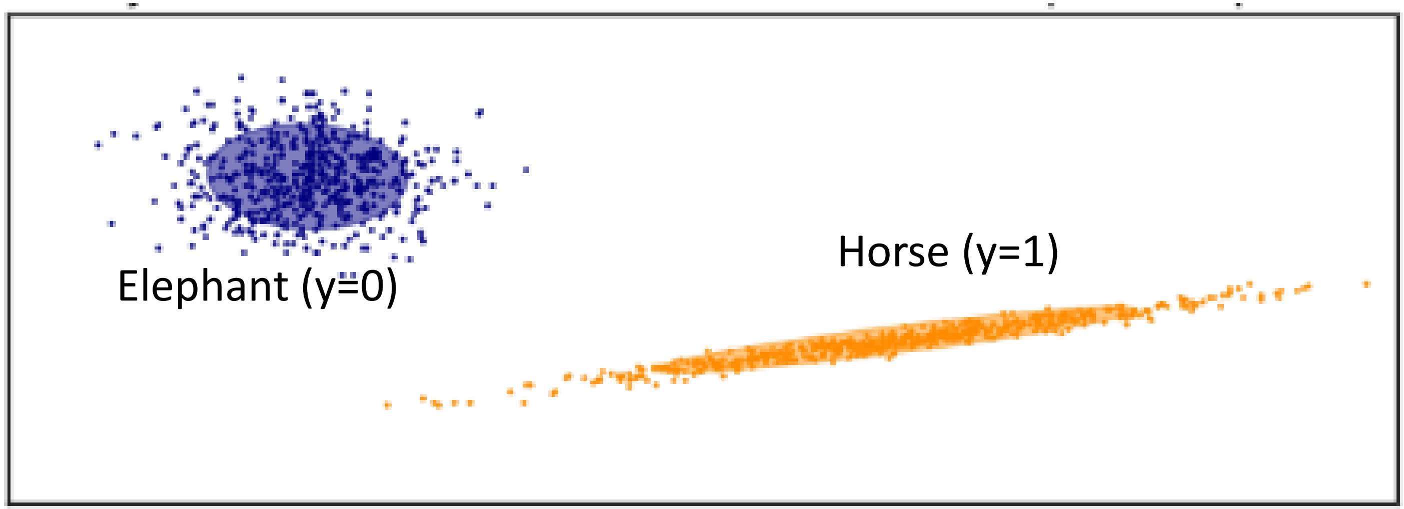
Hongsheng Li
Department of Electronic Engineering
Chinese University of Hong Kong

Adversarial – adj. 對抗的

Generative Models

- Density Estimation
  - Discriminative model: $p(y \mid x)$
    - y=0 for elephant, y=1 for horse
  - Generative model: $p(x \mid y)$

$$p(x \mid y=0) \qquad\qquad p(x \mid y=1)$$



Generative Models

- $$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

$$\begin{aligned} \arg\max_y p(y|x) &= \arg\max_y \frac{p(x|y)p(y)}{p(x)} \\ &= \arg\max_y p(x|y)p(y). \end{aligned}$$

- Sample Generation

Training samples

Model samples

Generative Models

- Sample Generation

Training samples

Model samples

Training samples

Generative Models

- Generative model

p_{data} $\longrightarrow$ Data $\longrightarrow$ p_{model} $\dashrightarrow$ Sample generation

- GAN is a generative model
  - Mainly focuses on sample generation
  - Possible to do both

Why Worth Studying?

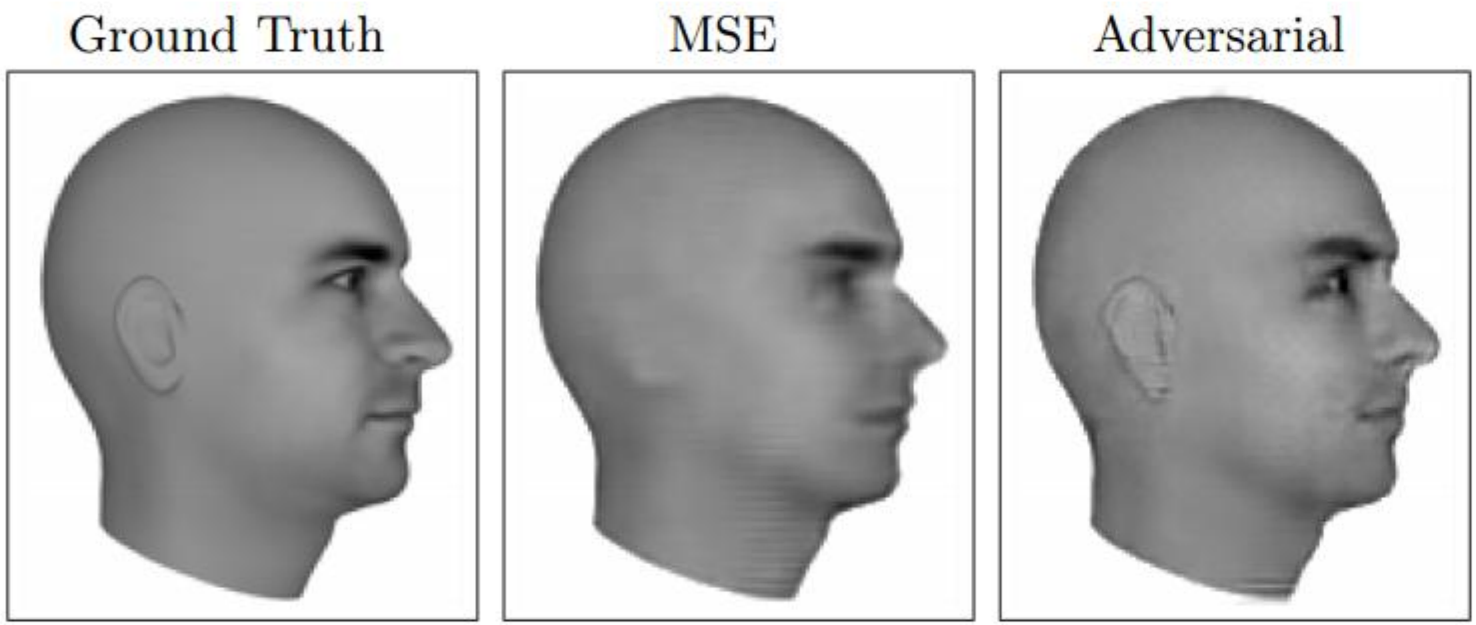
- Excellent test of our ability to use high-dimensional, complicated probability distributions

p_{model} ----→ Sample generation

- Missing data
  – Semi-supervised learning

Why Worth Studying?

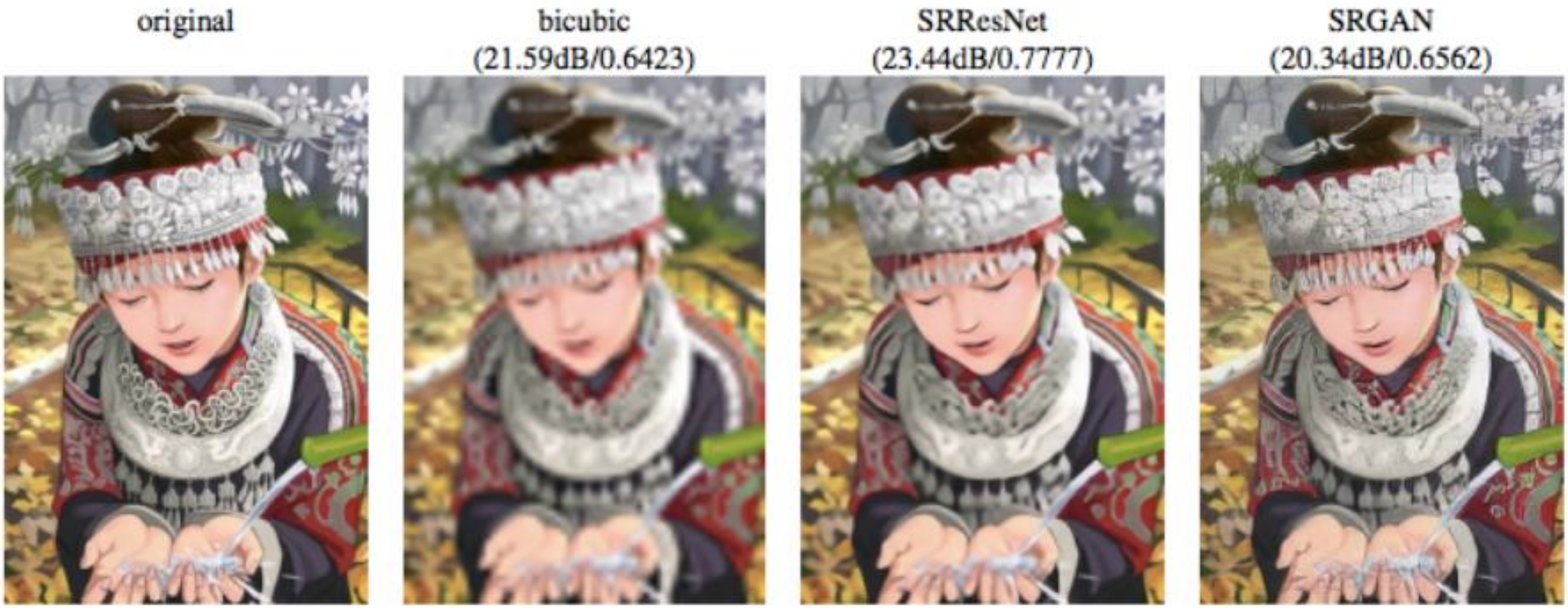
- Multi-modal outputs
  - Example: next frame prediction



Lotter et al. 2015

Why Worth Studying?

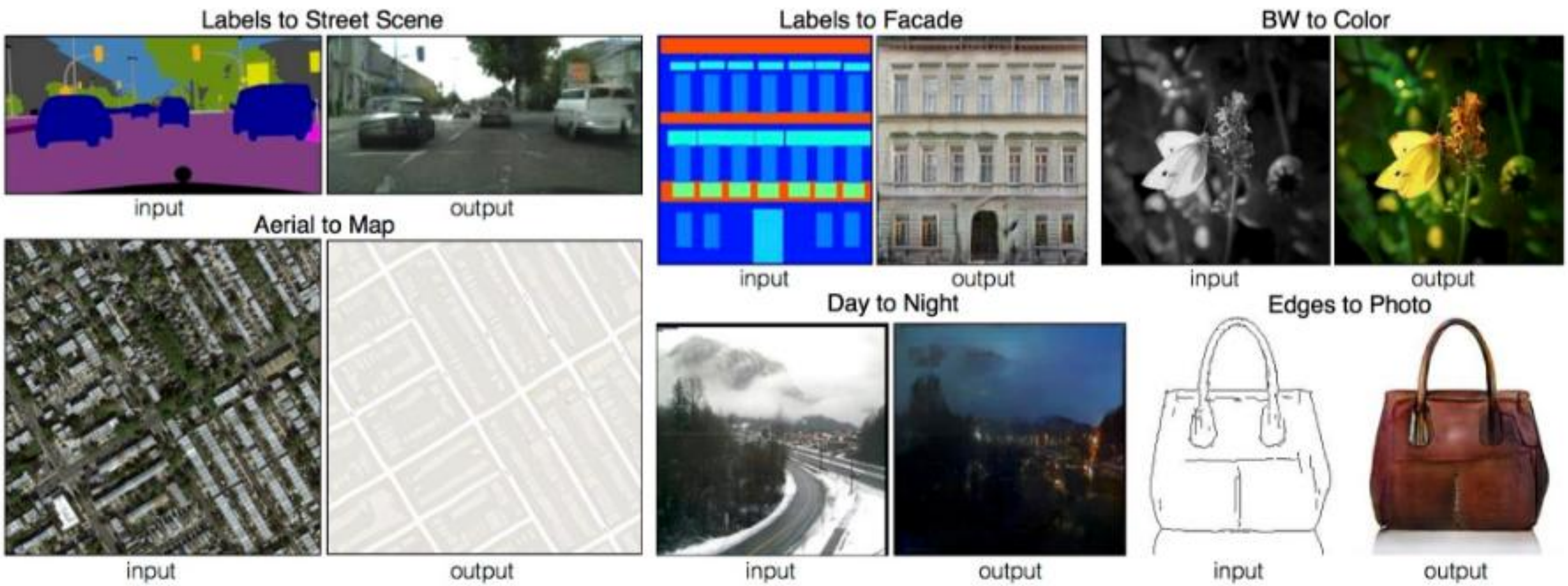
- Image generation tasks
  - Example: single-image super-resolution



Ledig et al 2015

Why Worth Studying?

- Image generation tasks
  - Example: Image-to-Image Translation
  - https://affinelayer.com/pixsrv/



Isola et al 2016

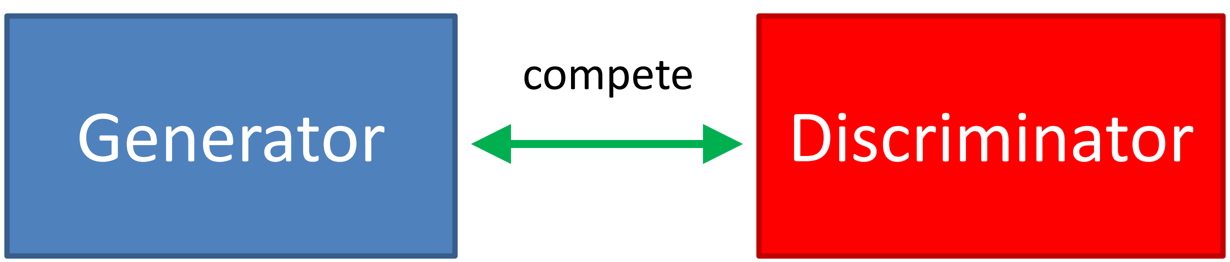
Why Worth Studying?

- Image generation tasks
  - Example: Text-to-Image Generation

Zhang et al 2016

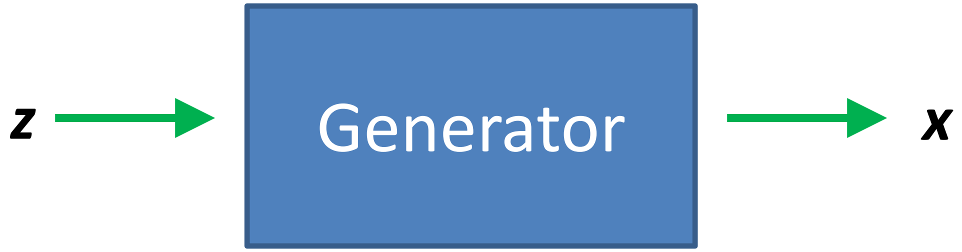
How does GAN Work?

- Adversarial – adj. 對抗的
- Two networks:
  - Generator *G*: creates (fake) samples that the discriminator cannot distinguish
  - Discriminator *D*: determine whether samples are fake or real

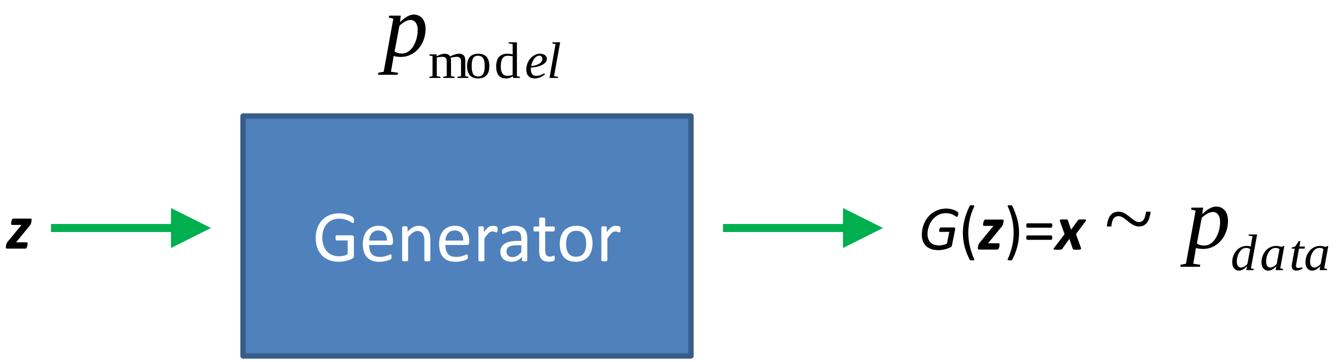


The Generator

- *G*: a differentiable function
  - modeled as a neural network
- Input:
  - $\boldsymbol{z}$: random noise vector from some simple prior distribution
- Output:
  - $\boldsymbol{x} = G(\boldsymbol{z})$: generated samples



The Generator

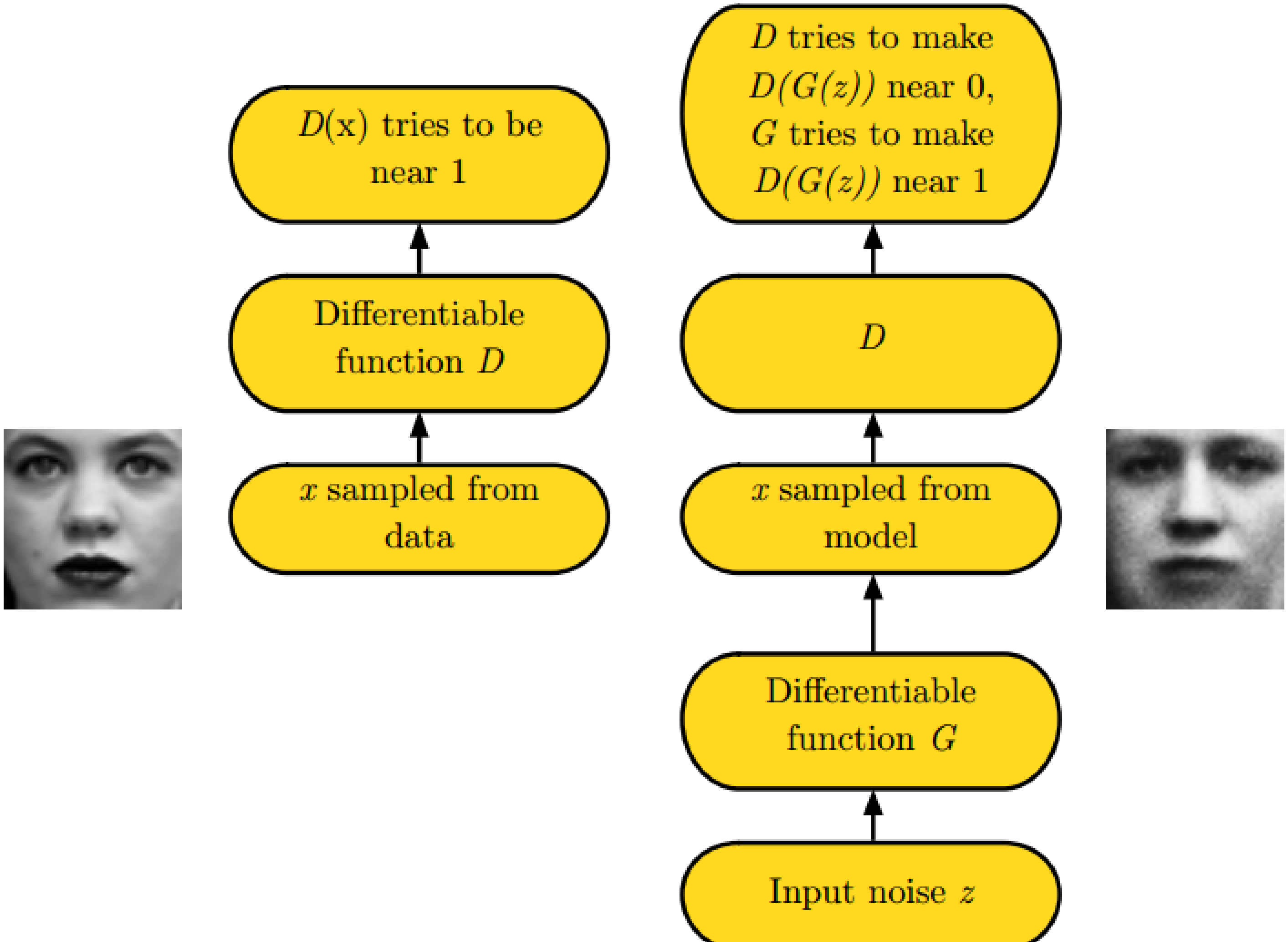


- The dimension of ***z*** should be at least as large as that of ***x***

The Discriminator

- *D*: modeled as a neural network
- Input:
  - Real sample
  - Generated sample ***x***
- Output:
  - 1 for real samples
  - 0 for fake samples

Generative Adversarial Networks



Cost Functions

- The discriminator outputs a value *D(x)* indicating the chance that *x* is a real image
- For real images, their ground-truth labels are 1. For generated images their labels are 0.
- Our objective is to maximize the chance to recognize real images as real and generated images as fake
- The objective for generator can be defined as

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{recognize real images better}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{recognize generated images better}}$$

Cost Functions

- For the generator *G*, its objective function wants the model to generate images with the highest possible value of *D(x)* to fool the discriminator
- The cost function is

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

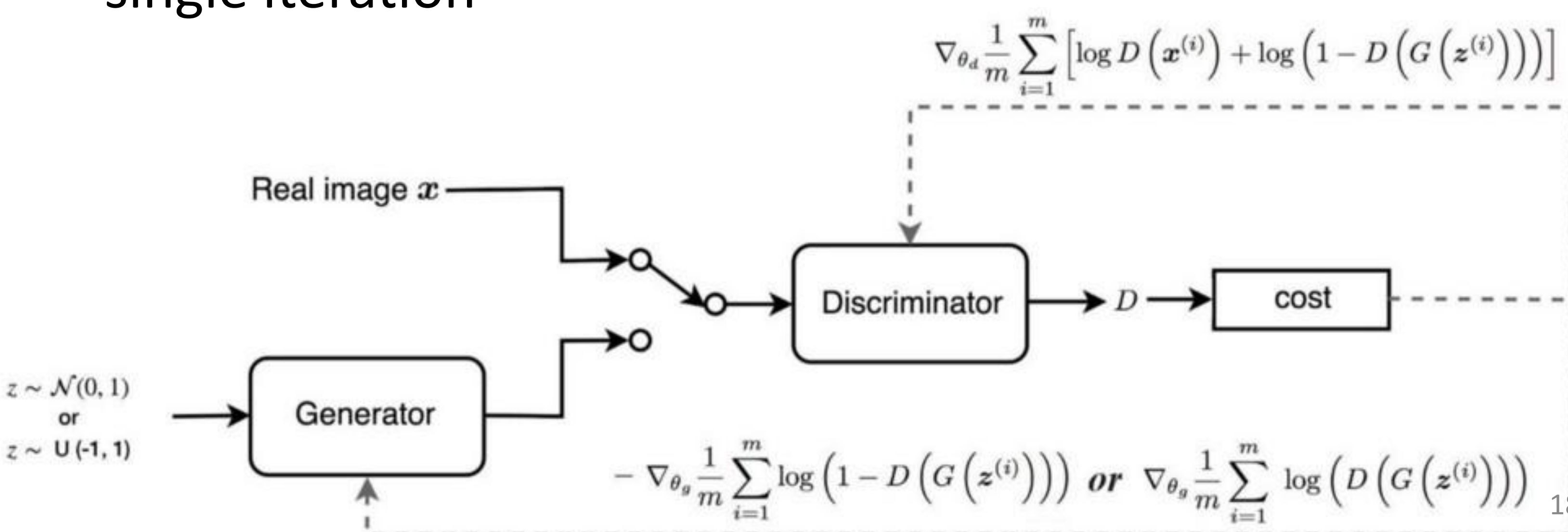
Optimize G that can fool the discriminator the most.

- The overall GAN training is therefore a min-max game

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

Training Procedure

- The generator and the discriminator are learned jointly by the alternating gradient descent
  - Fix the generator's parameters and perform a single iteration of gradient descent on the discriminator using the real and the generated images
  - Fix the discriminator and train the generator for another single iteration



The Algorithm

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

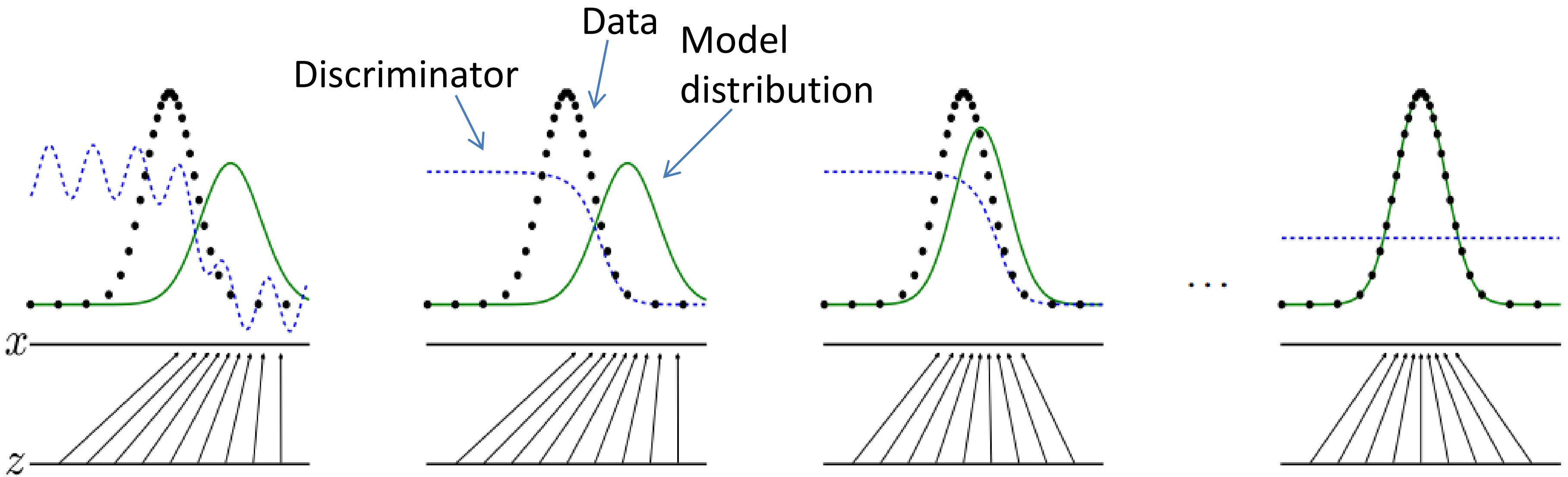
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Illustration of the Learning

- Generative adversarial learning aims to learn a model distribution that matches the actual data distribution



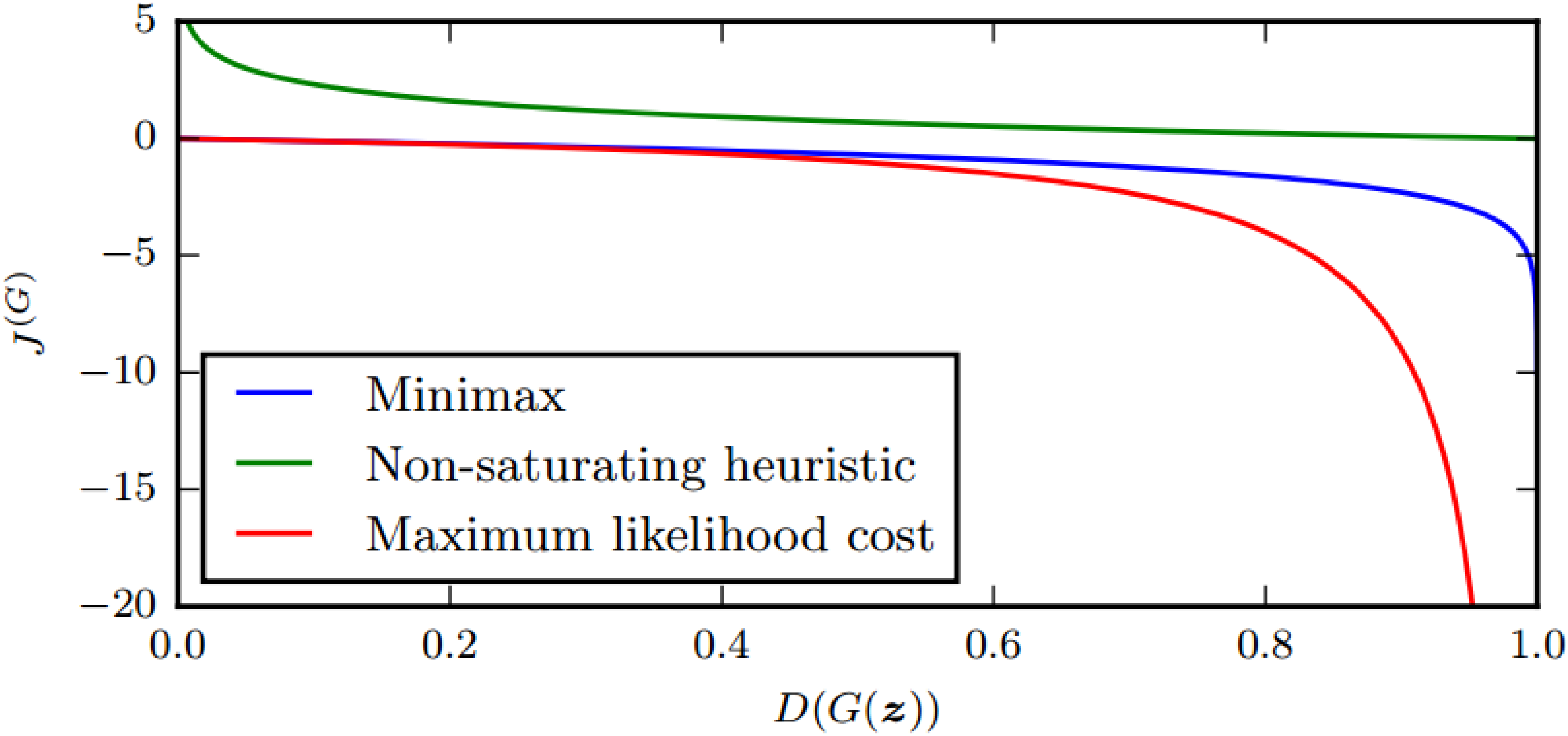
Generator diminished gradient

- However, we encounter a gradient diminishing problem for the generator. The discriminator usually wins early against the generator
- It is always easier to distinguish the generated images from real images in early training. That makes *cost function* approaches 0. i.e. *-log(1 -D(G(z)))* → *0*
- The gradient for the generator will also vanish which makes the gradient descent optimization very slow
- To improve that, the GAN provides an alternative function to backpropagate the gradient to the generator

minimize ***maximize***

$$-\nabla_{\theta_g} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \rightarrow 0 \quad \textit{change to} \quad \nabla_{\theta_g} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$$

Comparison between Two Losses



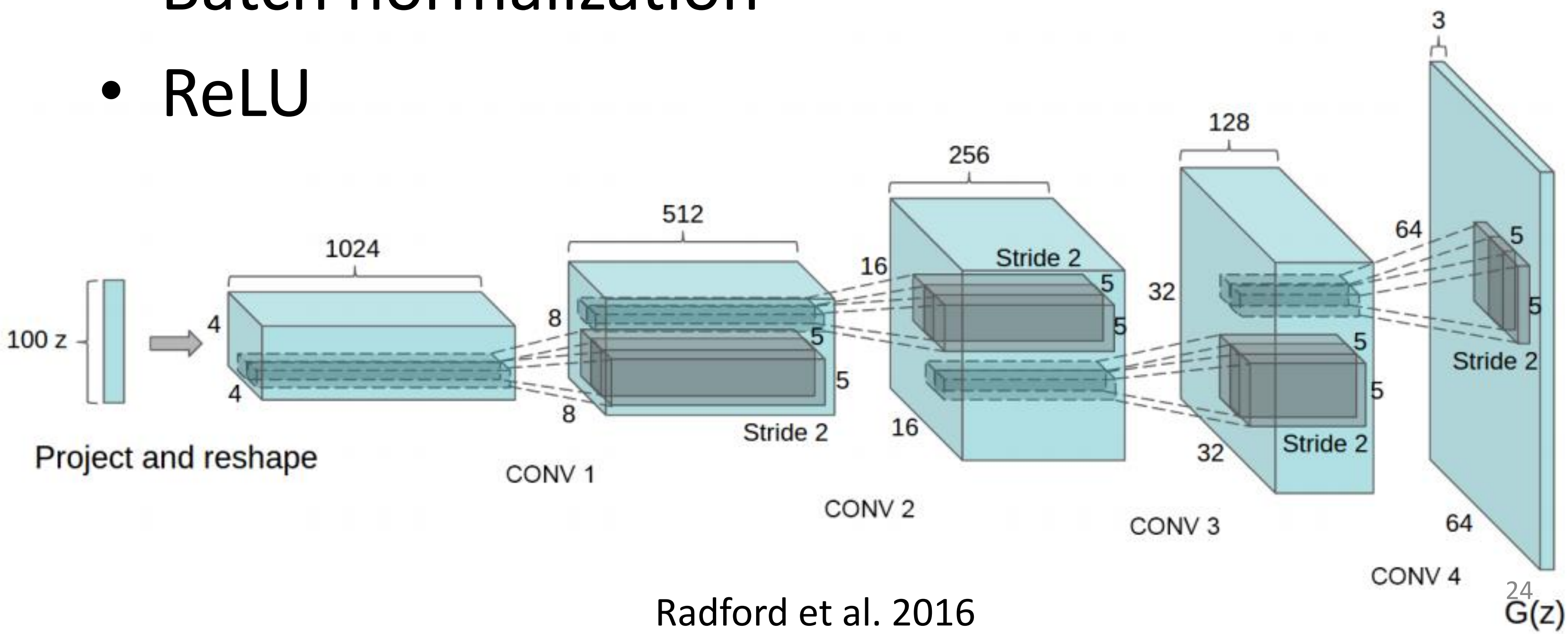
Non-Saturating Game

- $J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log \left(1 - D\left(G(\boldsymbol{z})\right)\right)$
- $J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$

- In the min-max game, the generator maximizes the same cross-entropy
- Now, generator maximizes the log-probability of the discriminator being mistaken
- Heuristically motivated; generator can still learn even when discriminator successfully rejects all generator samples

Deep Convolutional Generative Adversarial Networks (DCGAN)

- All convolutional nets
- No global average pooling
- Batch normalization
- ReLU



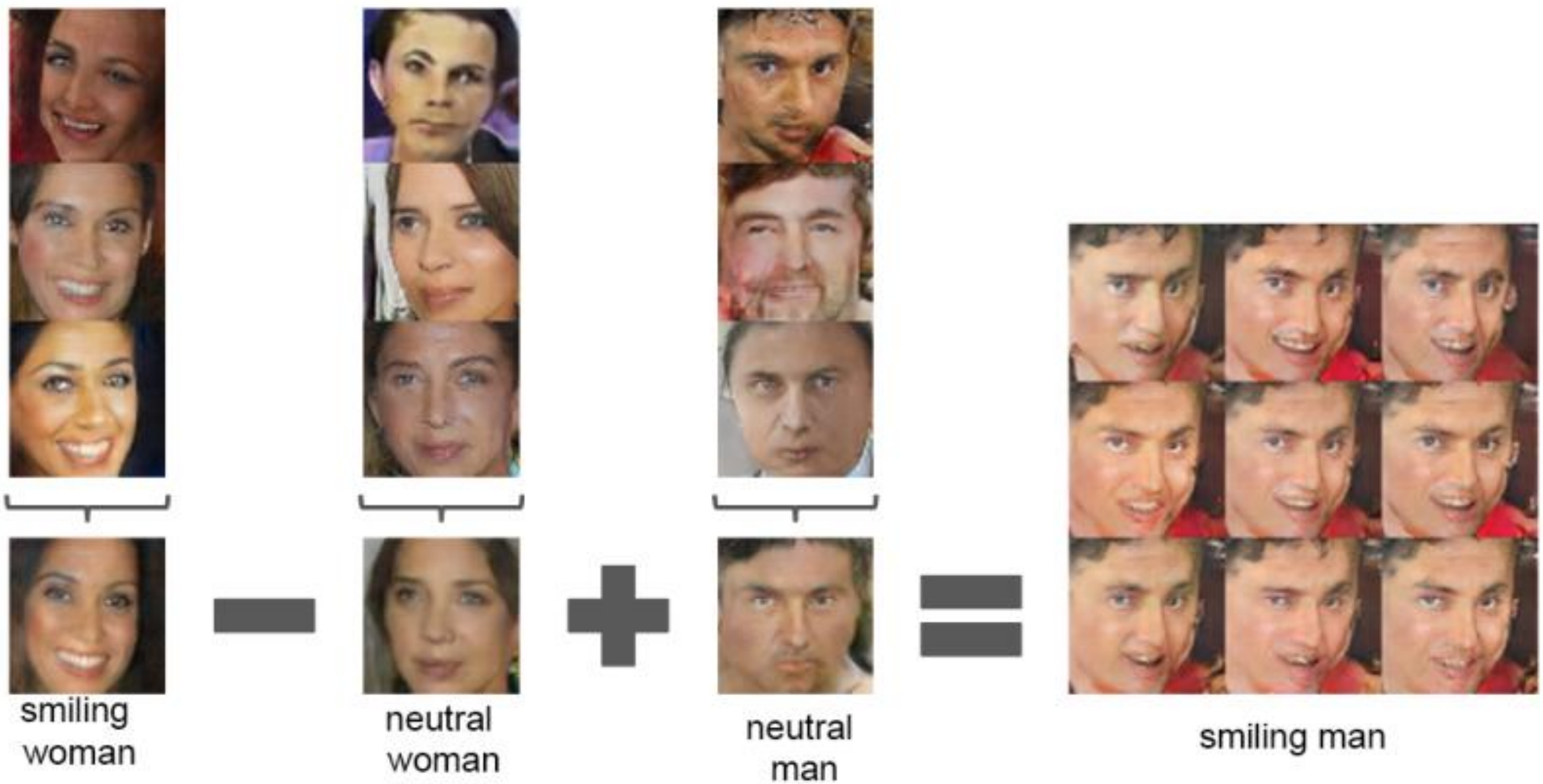
Radford et al. 2016

Deep Convolutional Generative Adversarial Networks (DCGAN)

- LSUN bedroom (about 3m training images)

Radford et al. 2016

Manipulating Learned *z*



Manipulating Learned *z*

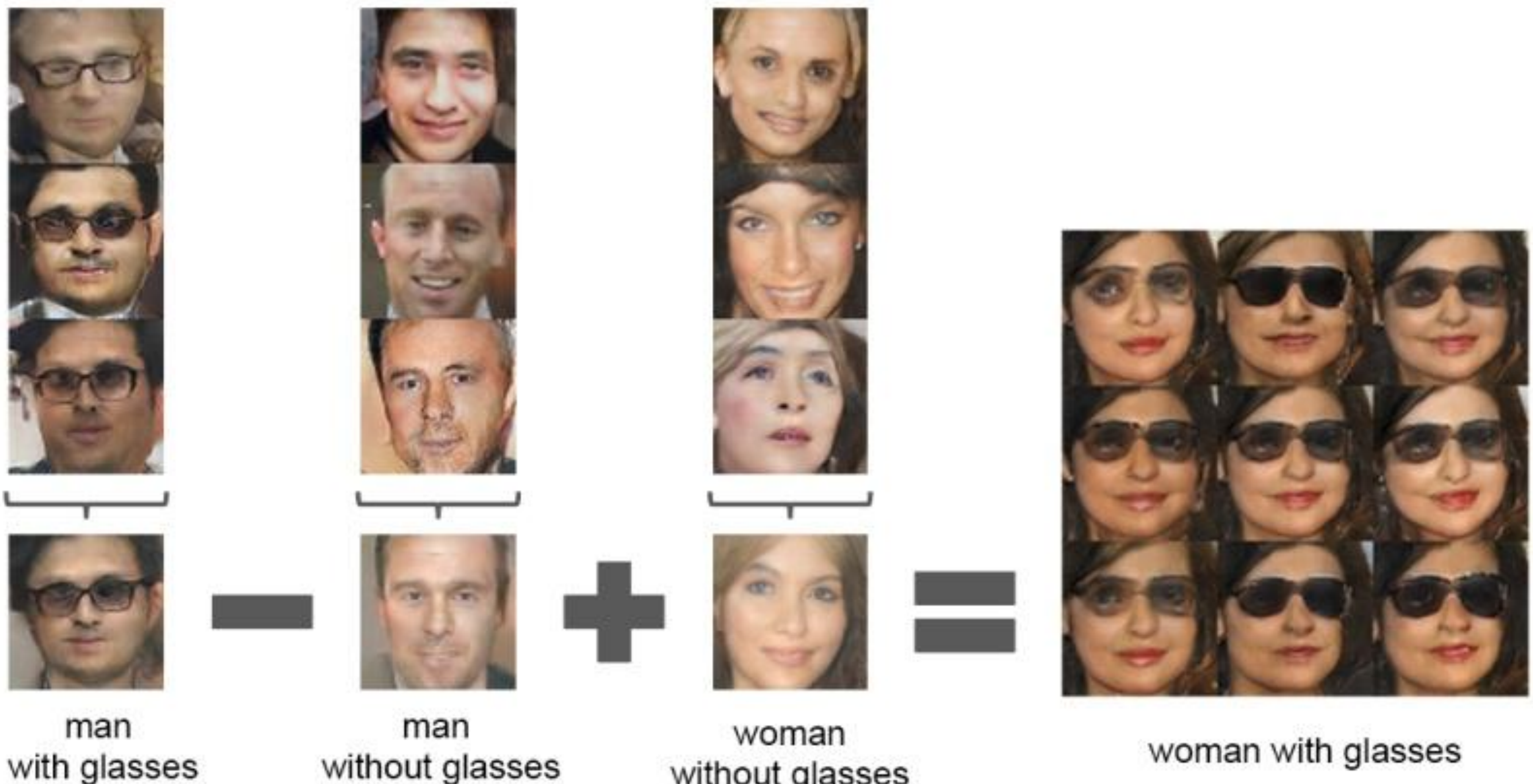
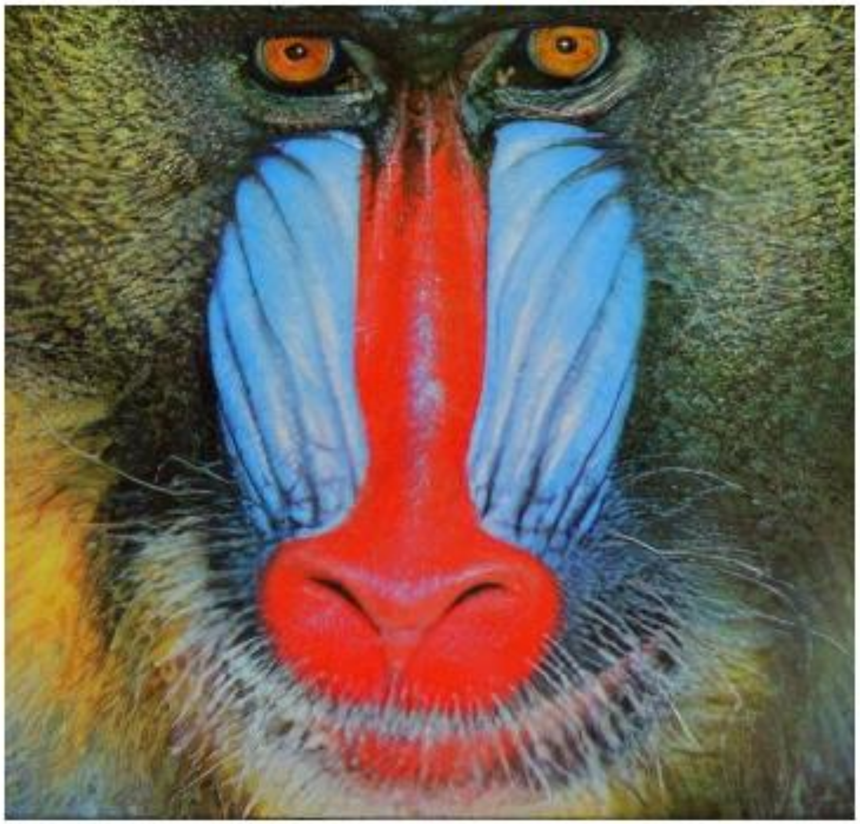
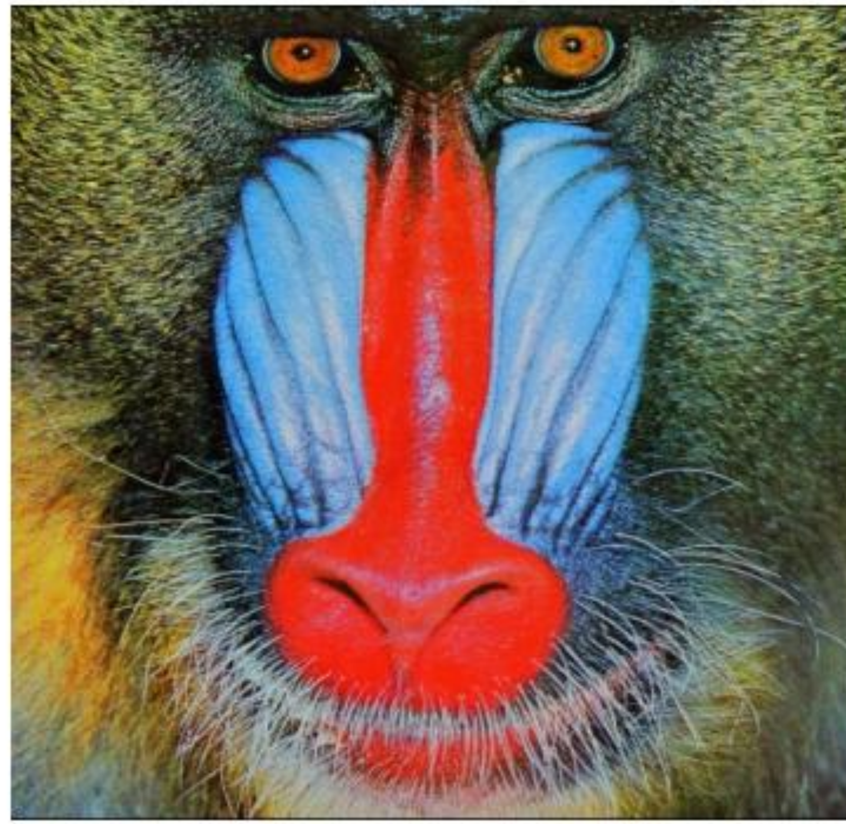


Image Super-resolution with GAN

4× SRGAN (proposed)



original



Ledig et al. 2016

Image Super-resolution with GAN

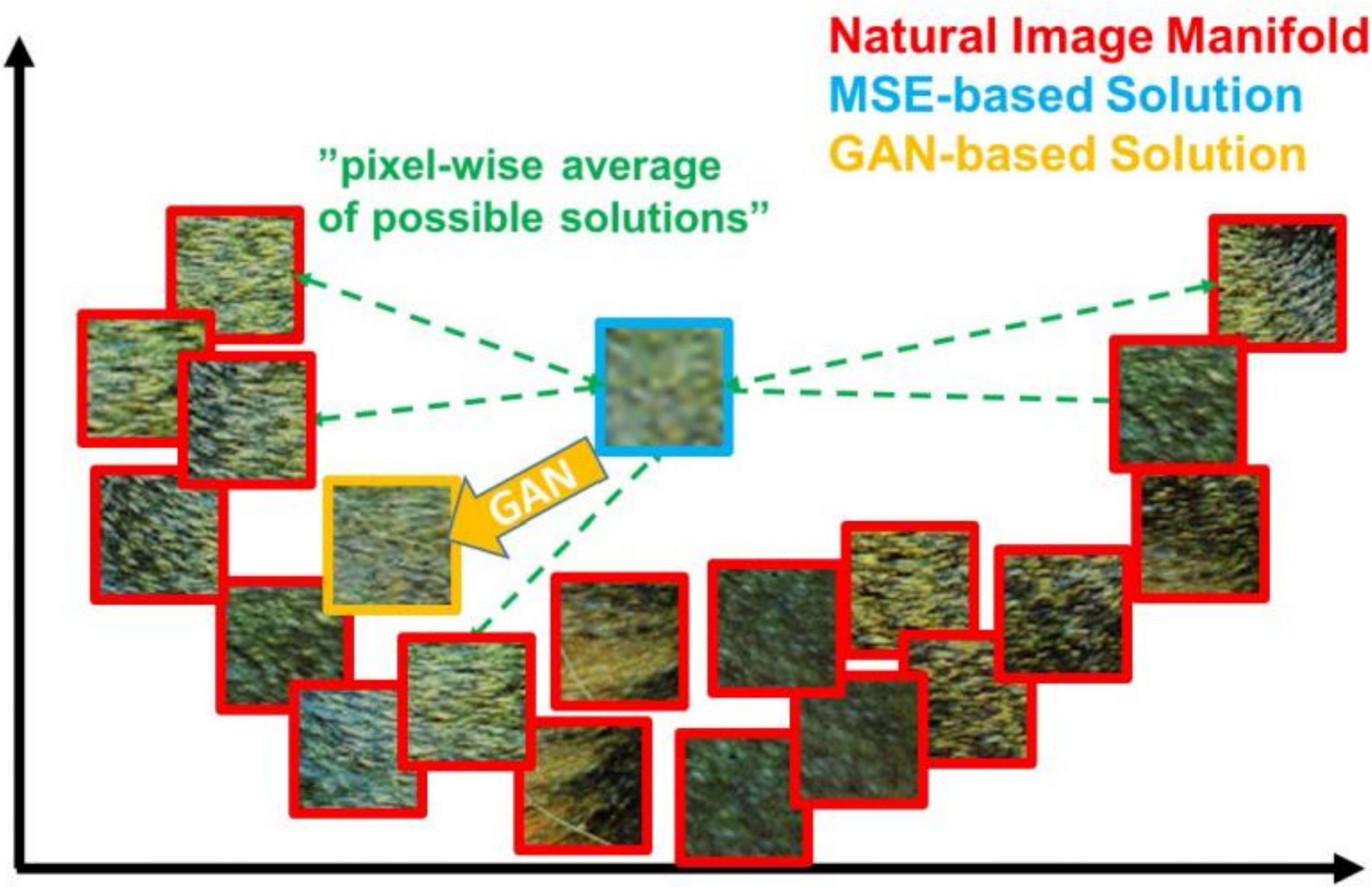


Image Super-resolution with GAN

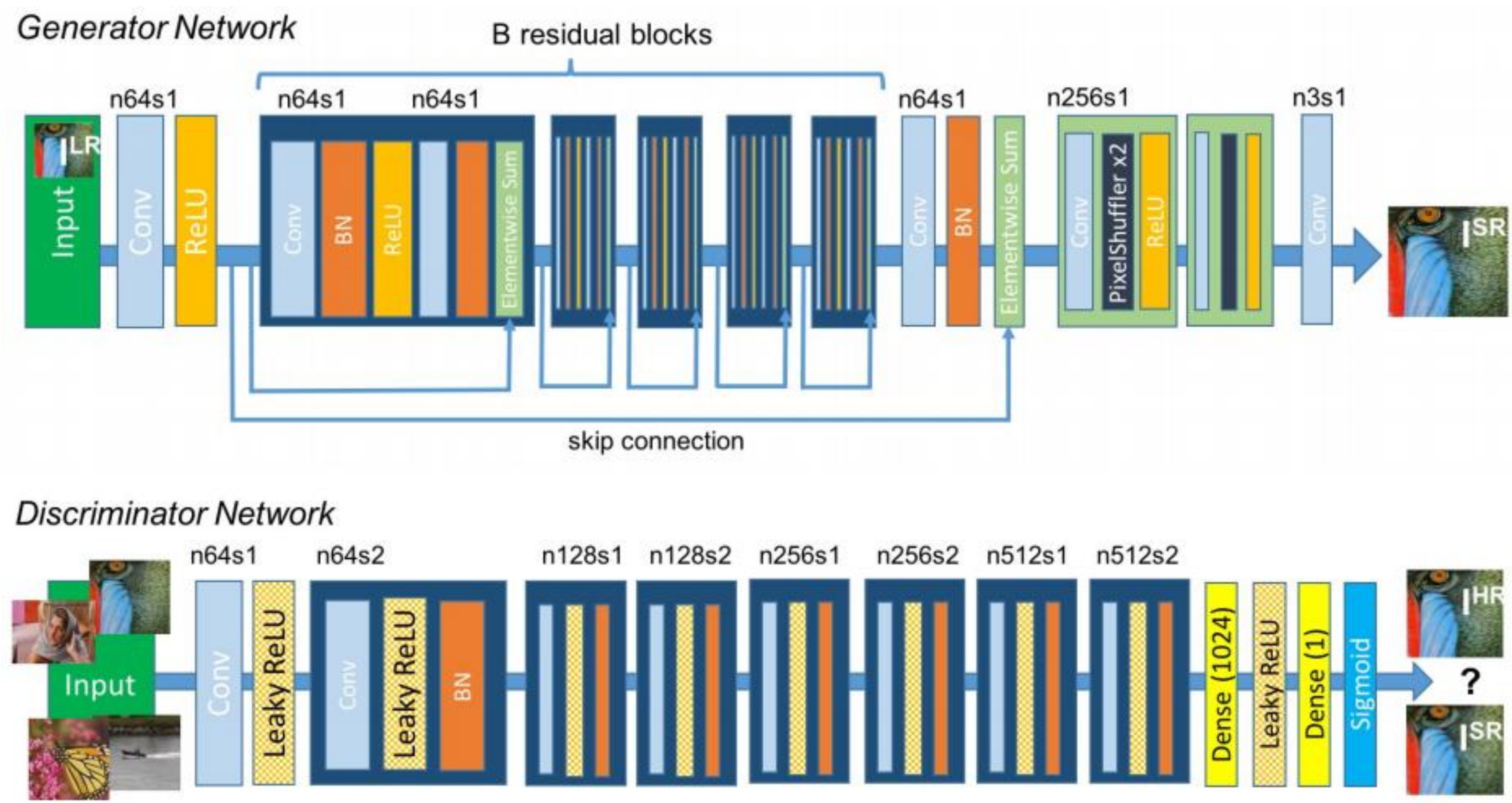


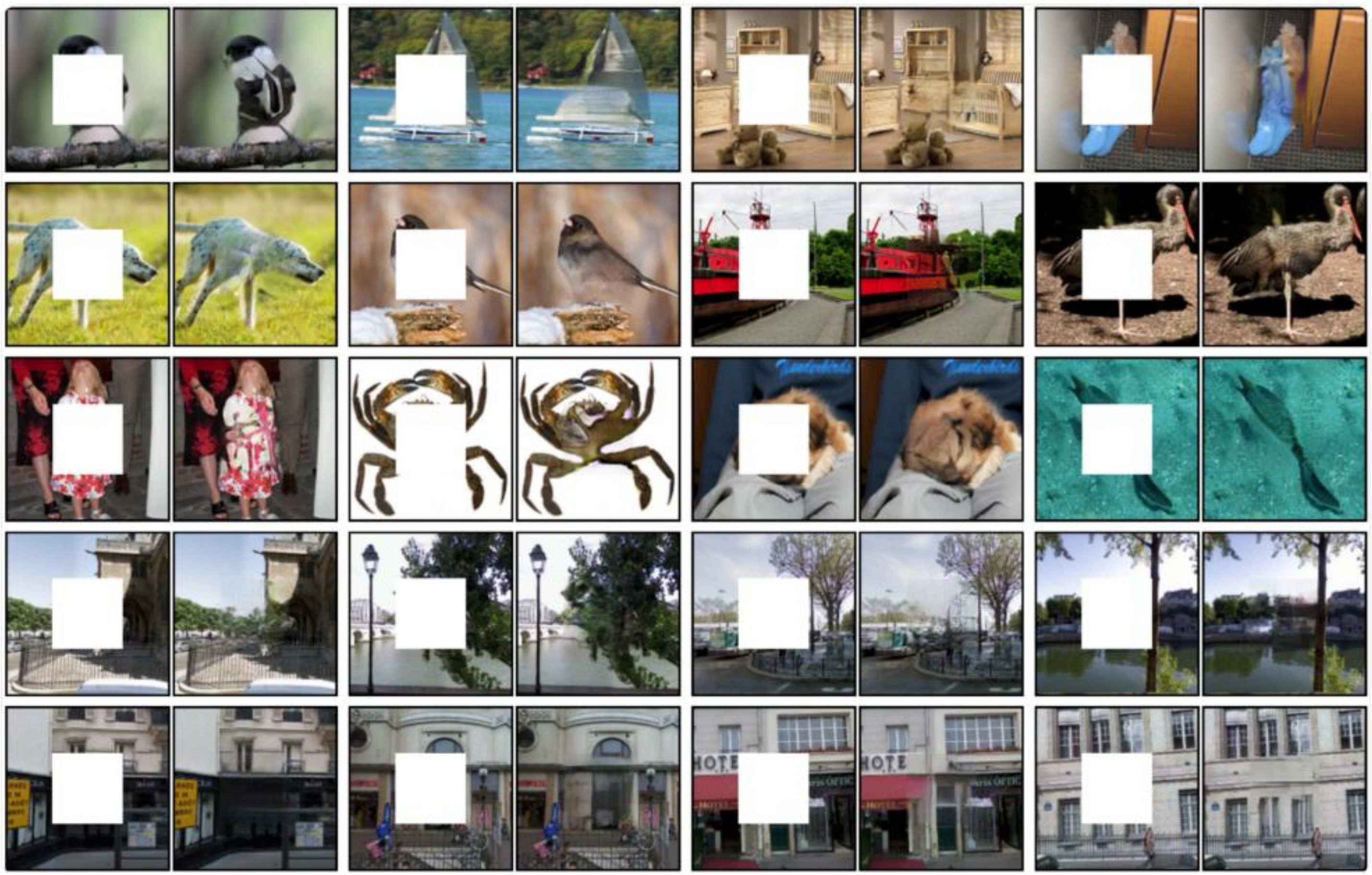
Image Super-resolution with GAN

$$l^{SR} = \underbrace{l^{SR}_{\mathrm{X}}}_{\text{content loss}} + \underbrace{10^{-3} l^{SR}_{Gen}}_{\text{adversarial loss}}$$

bicubic | SRResNet | SRGAN | original

Context-Encoder for Image Inpainting

- For a pre-defined region, synthesize the image contents



Pathak et al 2016

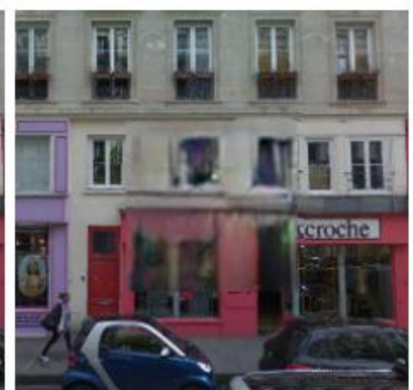
Context-Encoder for Image Inpainting

- For a pre-defined region, synthesize the image contents

(a) Input context

(b) Human artist

(c) Context Encoder
($L2$ loss)

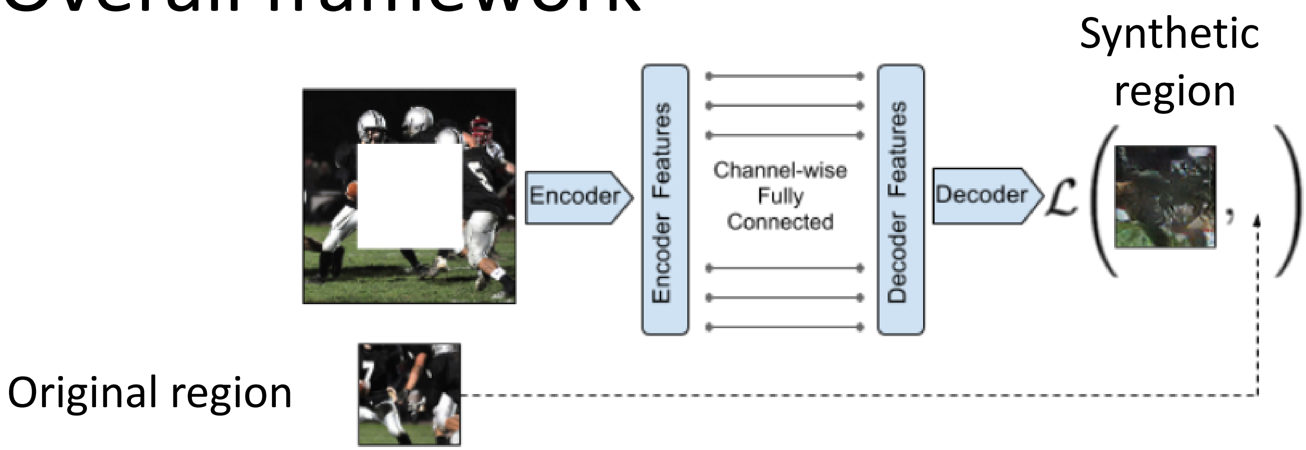


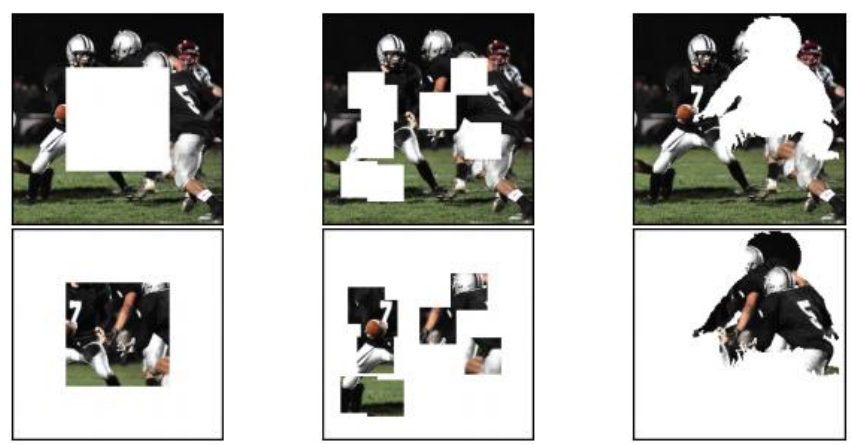
(d) Context Encoder
($L2$ + Adversarial loss)

Pathak et al 2016

Context-Encoder for Image Inpainting

- Overall framework

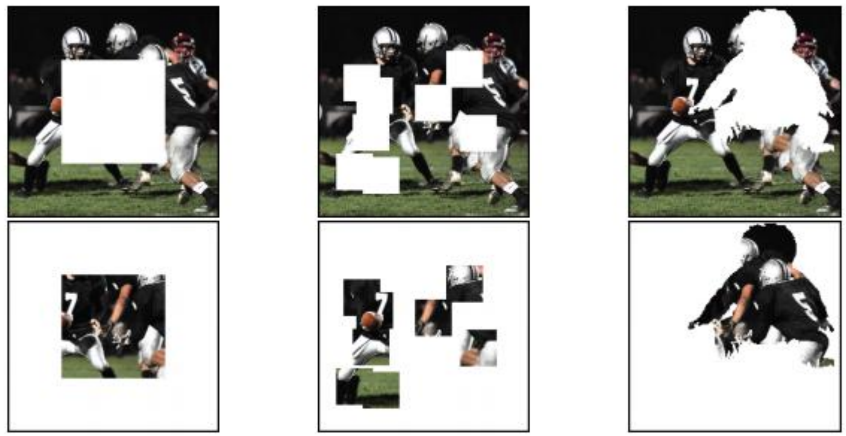




(a) Central region (b) Random block (c) Random region

Context-Encoder for Image Inpainting

- The objective



(a) Central region (b) Random block (c) Random region

$$\mathcal{L}_{rec}(x) = \|\hat{M} \odot (x - F((1 - \hat{M}) \odot x))\|_2^2,$$

$$\begin{aligned}\mathcal{L}_{adv} = \max_{D} \quad & \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) \\ & + \log(1 - D(F((1 - \hat{M}) \odot x)))],\end{aligned}$$

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}.$$

Context-Encoder for Image Inpainting

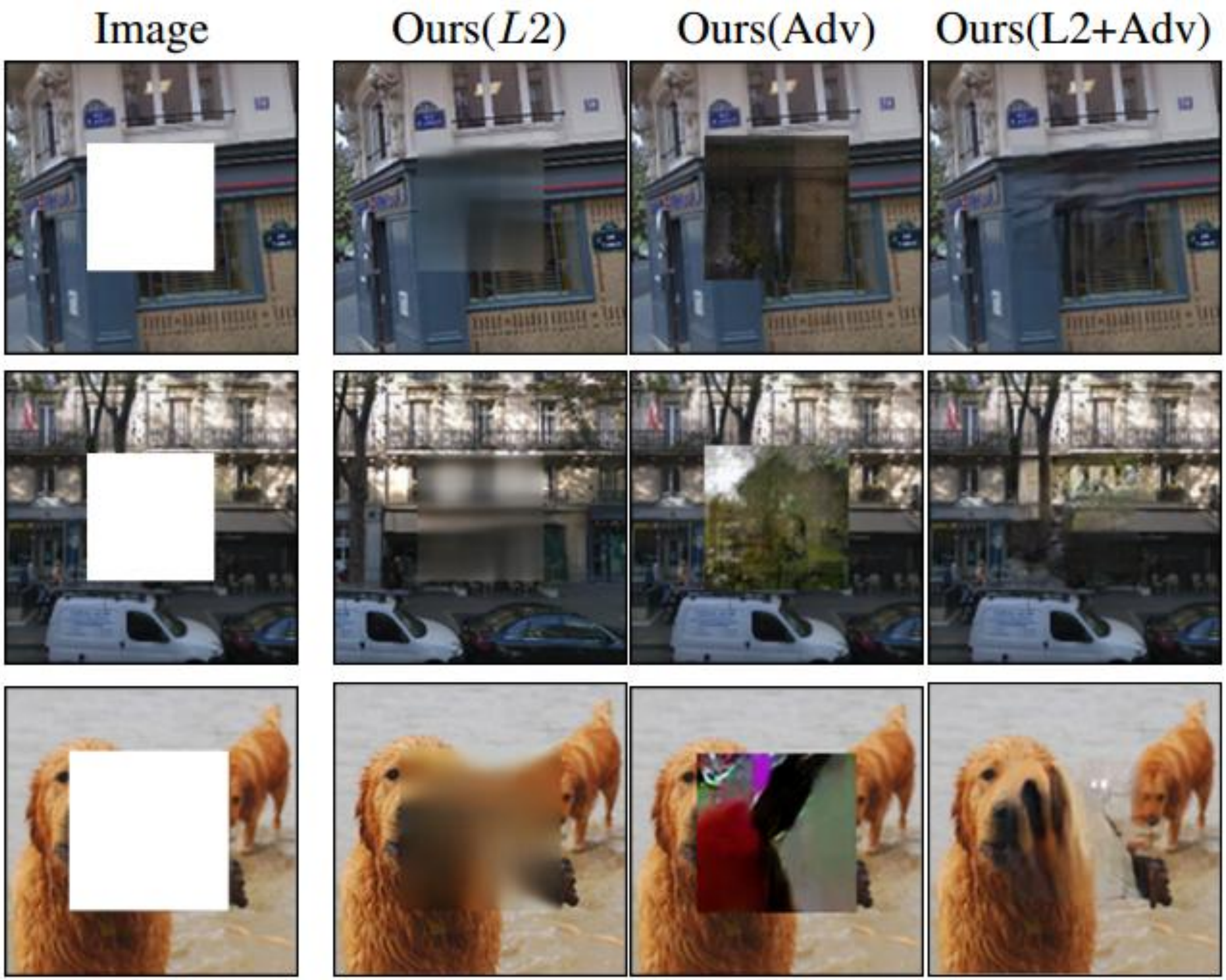


Image Inpainting with Partial Convolution

- Partial convolution for handling missing data
- L1 loss: minimizing the pixel differences between the generated image and their ground-truth images
- Perceptual loss: minimizing the VGG features of the generated images and their ground-truth images
- Style loss (Gram matrix): minimizing the gram matrices of the generated images and their ground-truth images

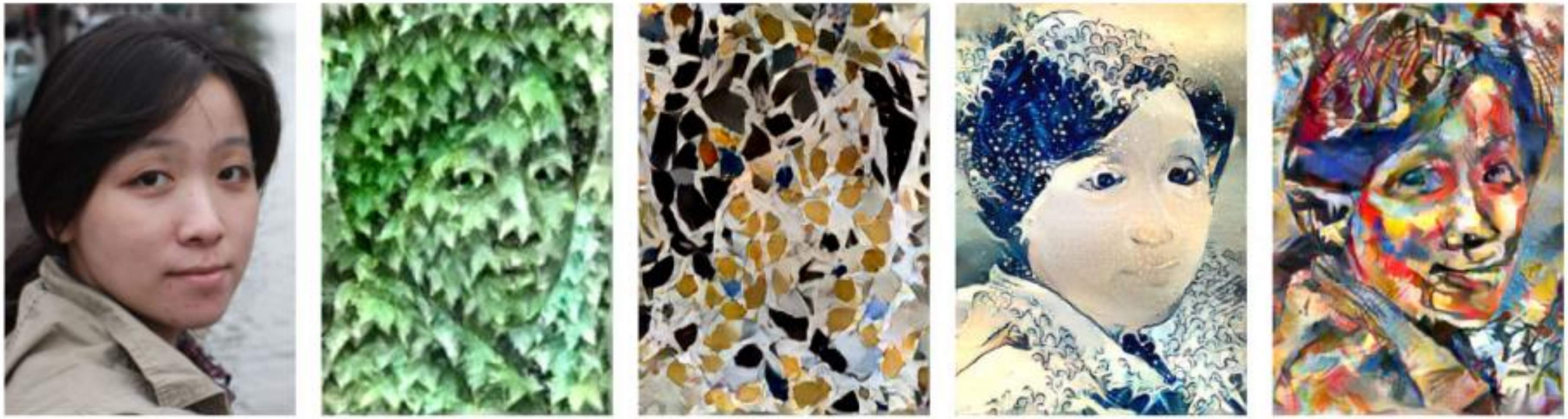
Liu 2016

Image Inpainting with Partial Convolution: Results

Liu 2016

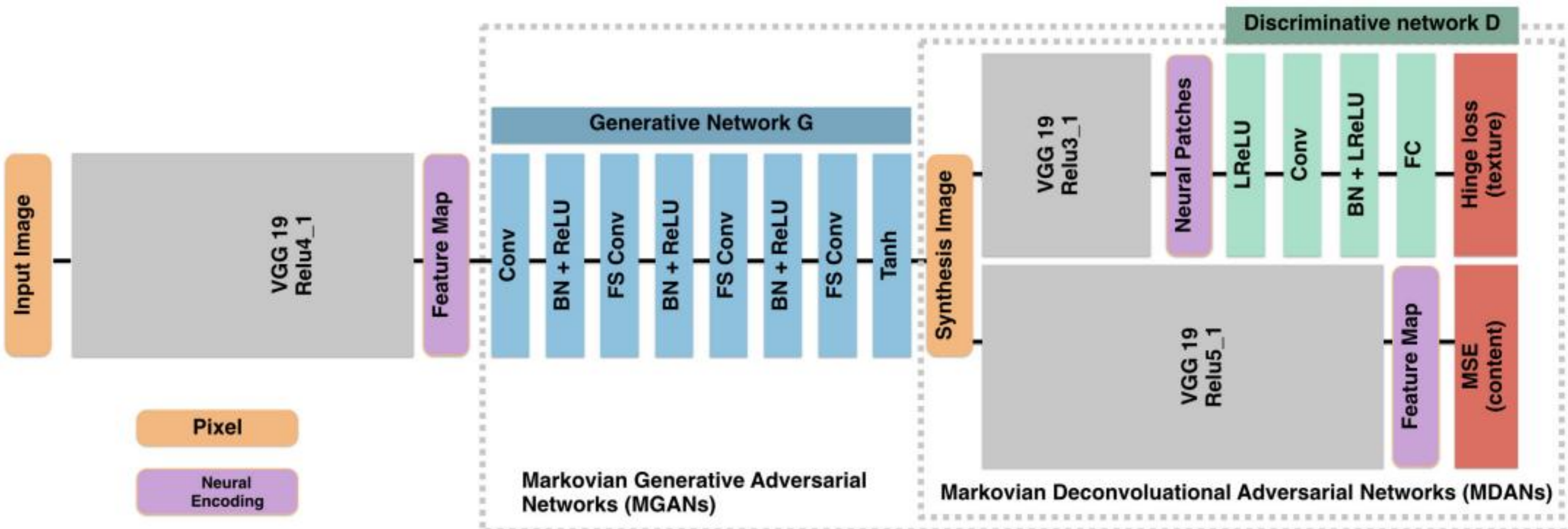
Texture Synthesis with Patch-based GAN

- Synthesize textures for input images



Liu et al. 2018

Texture Synthesis with Patch-based GAN



$$\mathbf{x} = \arg\min E_t(\Phi(\mathbf{x}), \Phi(\mathbf{x}_t)) + \alpha_1 E_c(\Phi(\mathbf{x}), \Phi(\mathbf{x}_c)) + \alpha_2 \Upsilon(\mathbf{x})$$

MSE Loss Adv loss

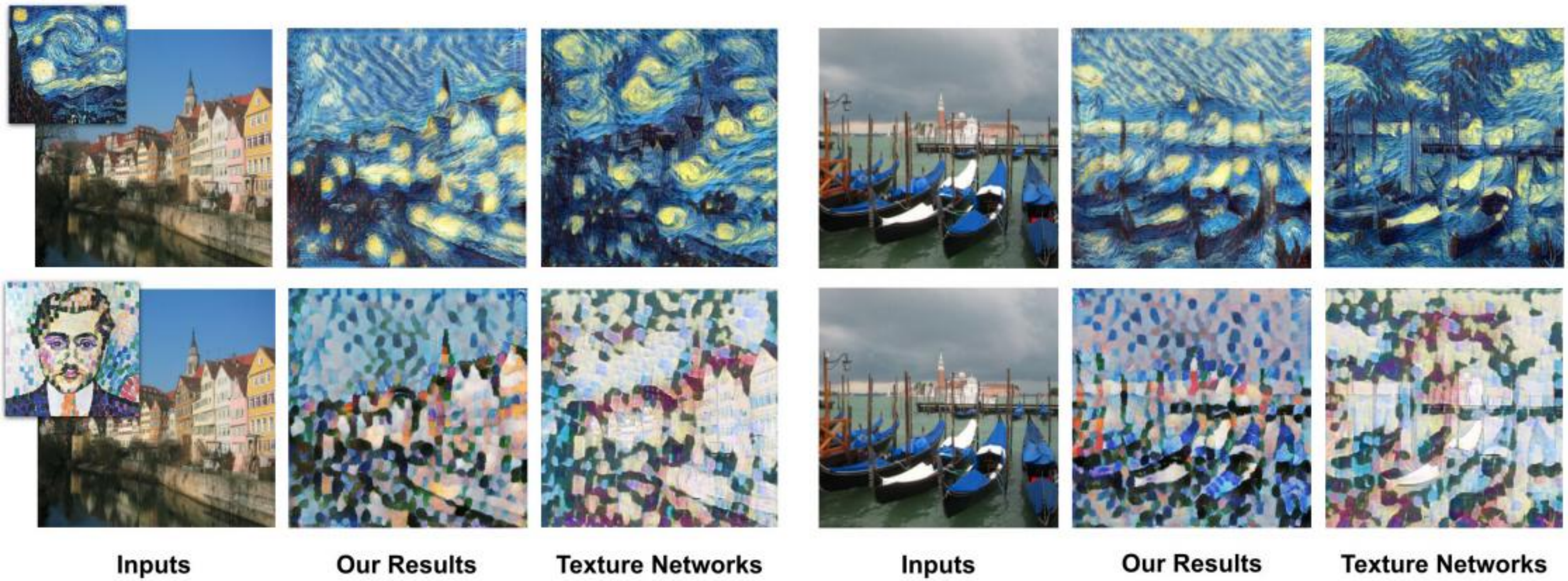
Li and Wand 2016

Texture Synthesis with Patch-based GAN

$$\mathbf{x} = \arg\min E_t(\Phi(\mathbf{x}), \Phi(\mathbf{x}_t)) + \boxed{\alpha_1 E_c(\Phi(\mathbf{x}), \Phi(\mathbf{x}_c)) + \alpha_2 \Upsilon(\mathbf{x})}$$

Li and Wand 2016

Texture Synthesis with Patch-based GAN

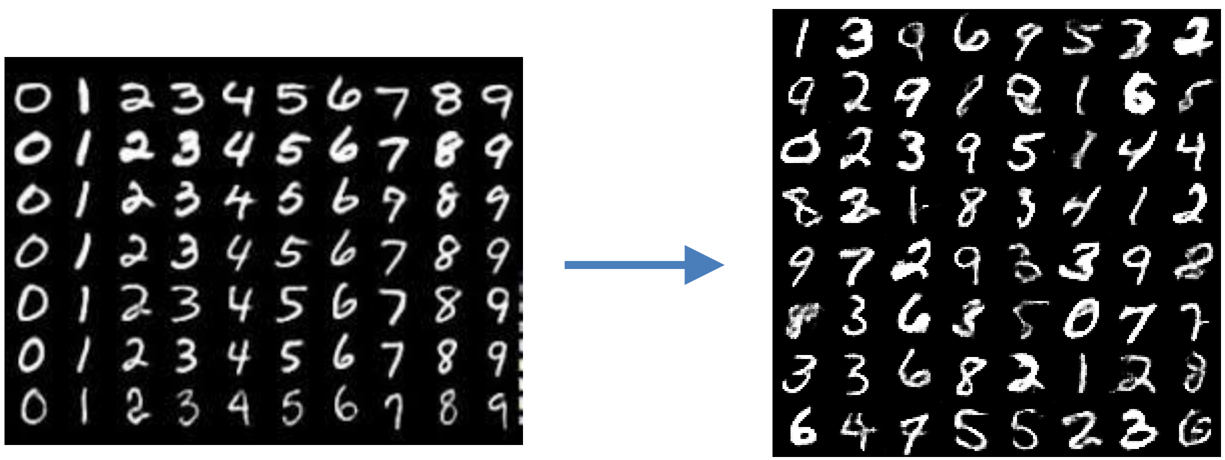


Li and Wand 2016

Conditional GAN

- GAN is too free. How to add some constraints?
- Add conditional variables ***y*** into the generator

Training samples



Model samples

Mirza and Osindero 2016

Conditional GAN

- GAN is too free. How to add some constraints?
- Add conditional variables ***y*** into *G* and *D*

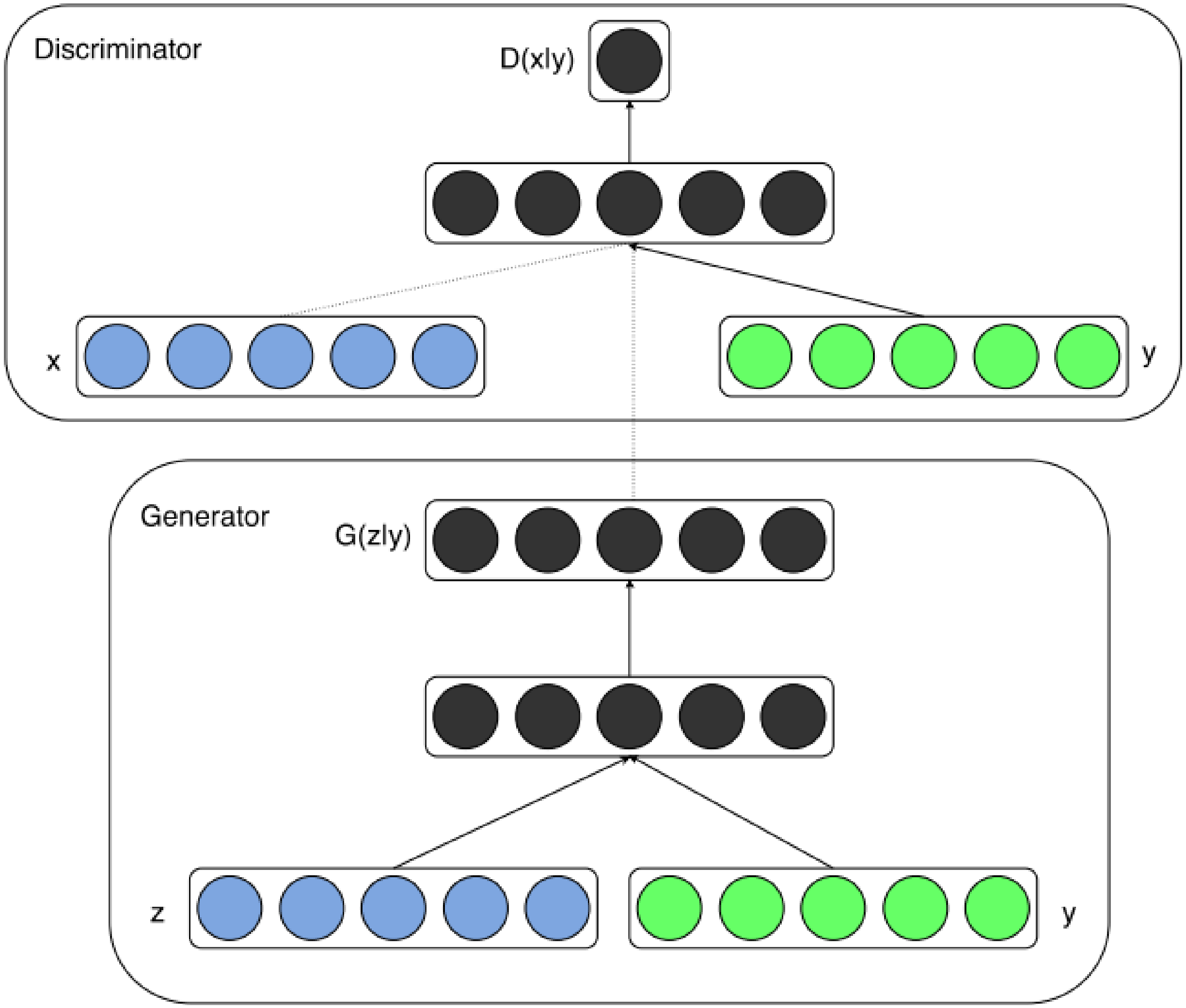
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

$$\downarrow$$

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

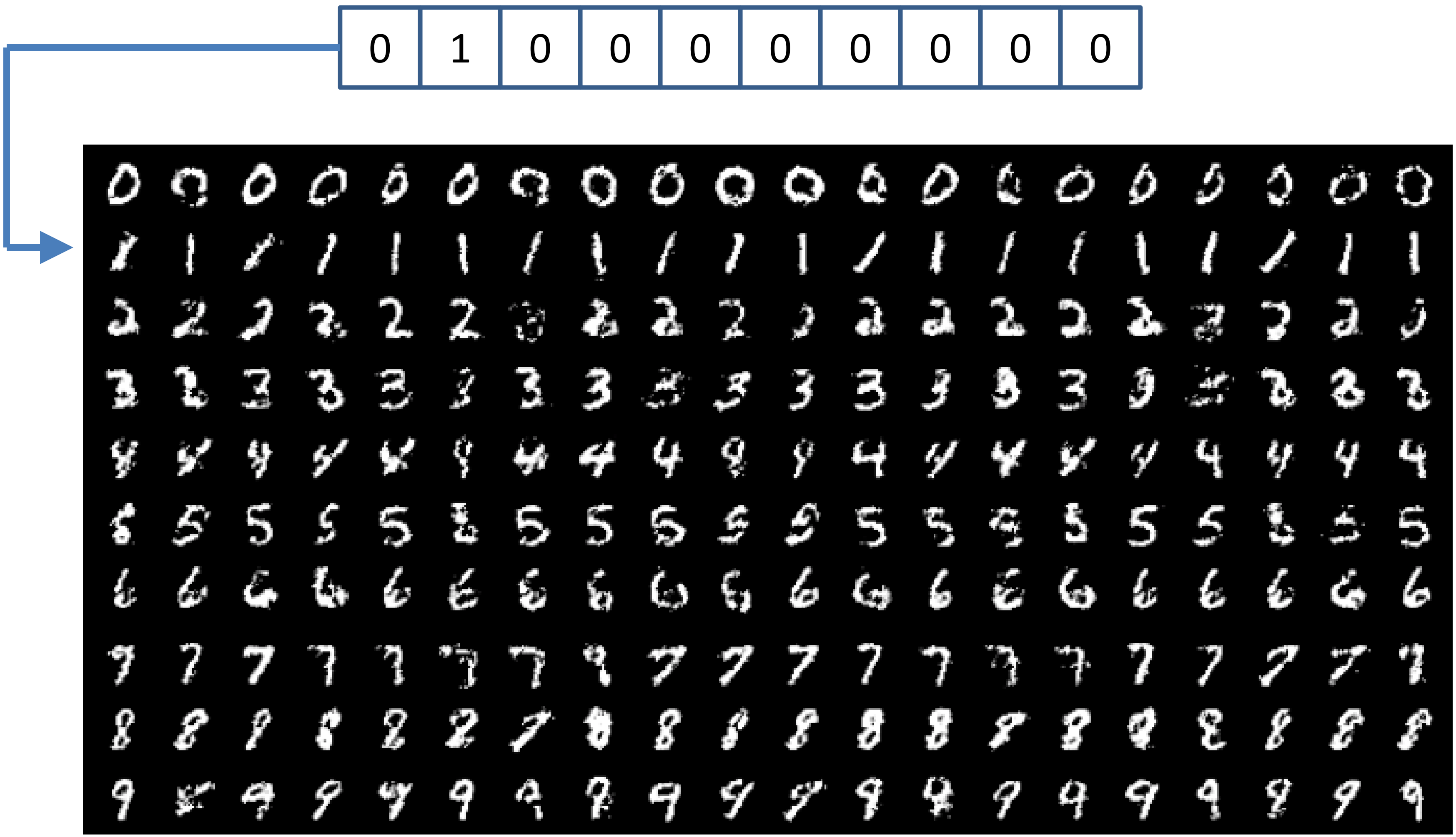
Mirza and Osindero 2016

Conditional GAN



Mirza and Osindero 2016

Conditional GAN



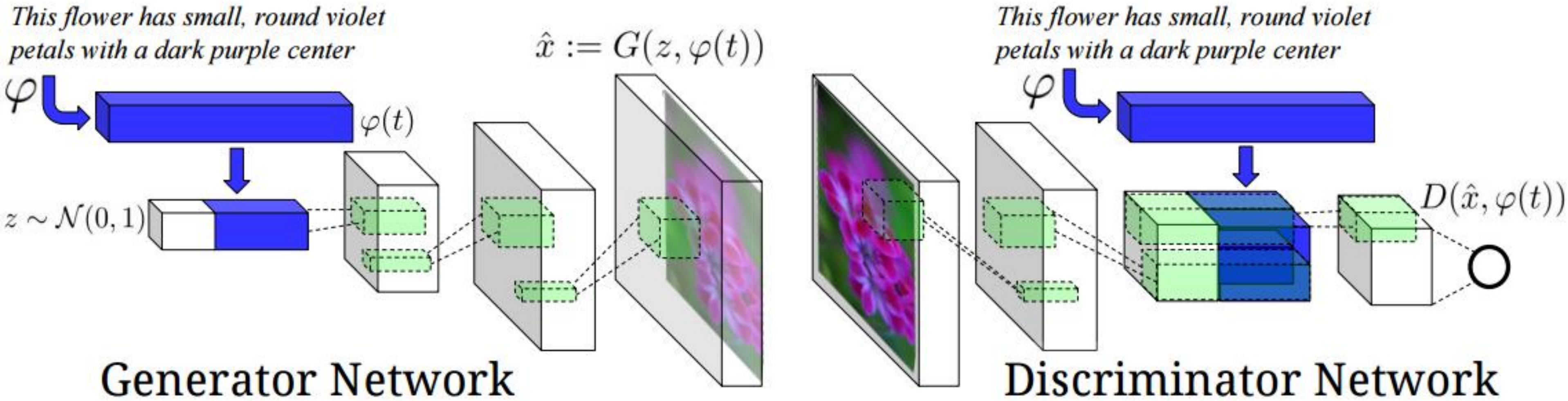
Mirza and Osindero 2016

Conditional GAN

- Positive samples for *D*
  - True data + corresponding conditioning variable
- Negative samples for *D*
  - Synthetic data + corresponding conditioning variable
  - True data + non-corresponding conditioning variable

Mirza and Osindero 2016

Text-to-Image Synthesis



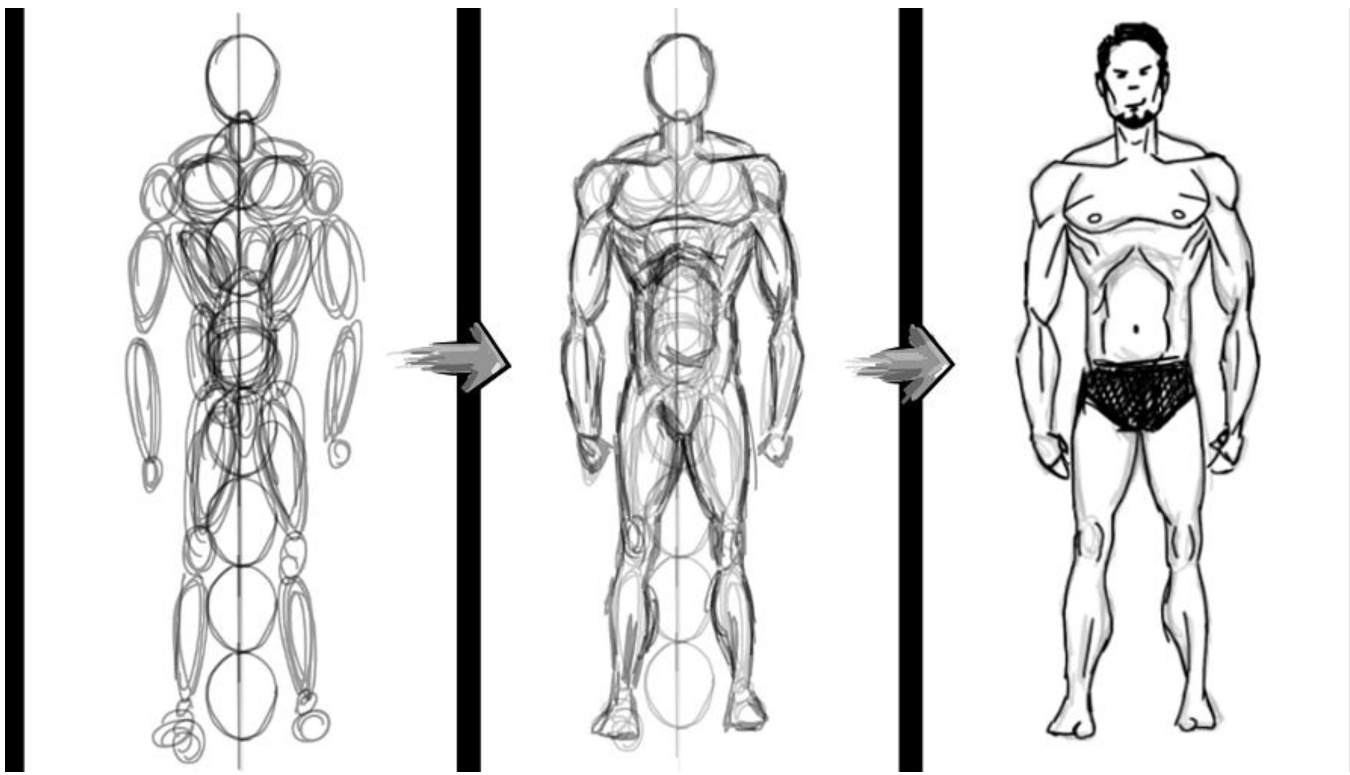
this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.

Reed et al 2015

StackGAN: Text to Photo-realistic Images

- How humans draw a figure?
  - A coarse-to-fine manner



Zhang et al. 2016

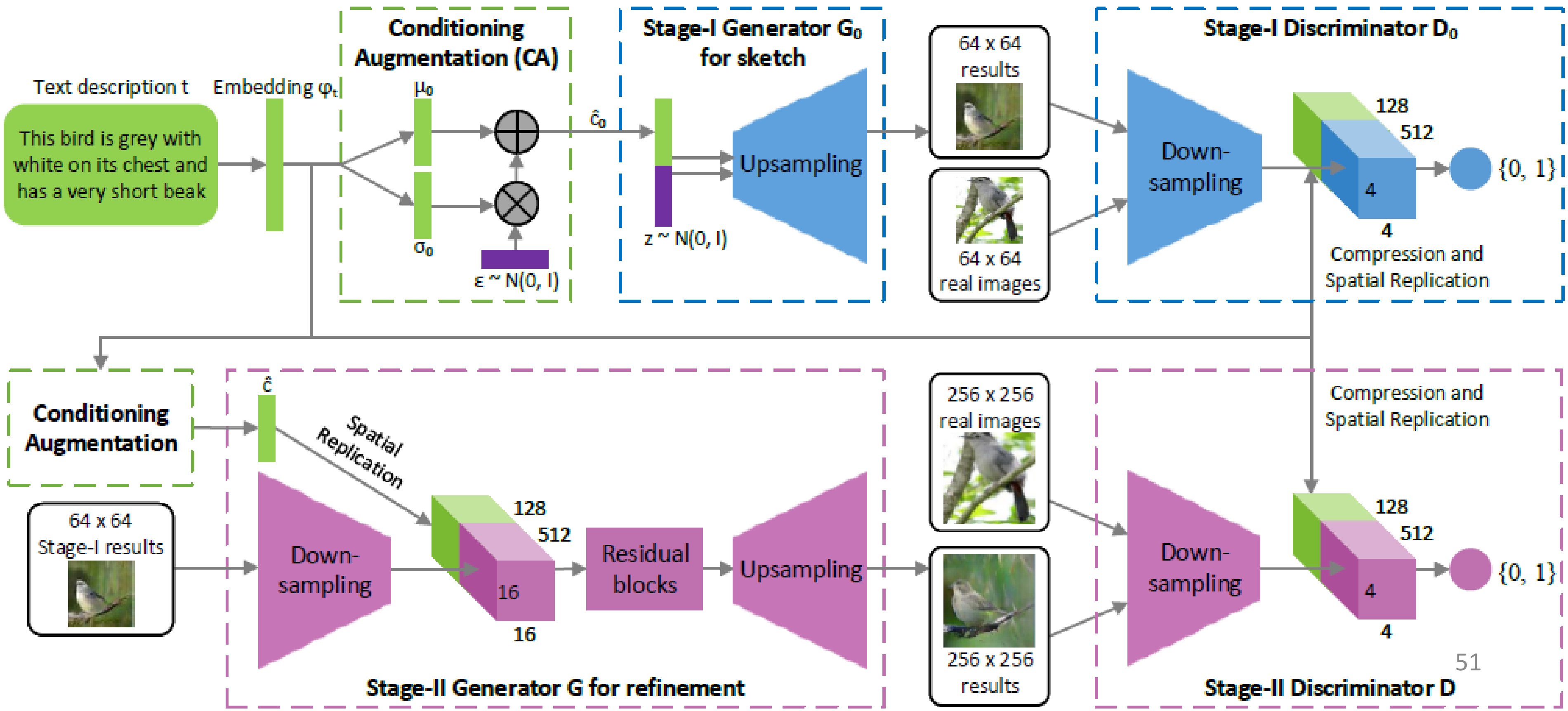
StackGAN: Text to Photo-realistic Images

- Use stacked GAN structure for text-to-image synthesis

Zhang et al. 2016

StackGAN: Text to Photo-realistic Images

- Use stacked GAN structure for text-to-image synthesis

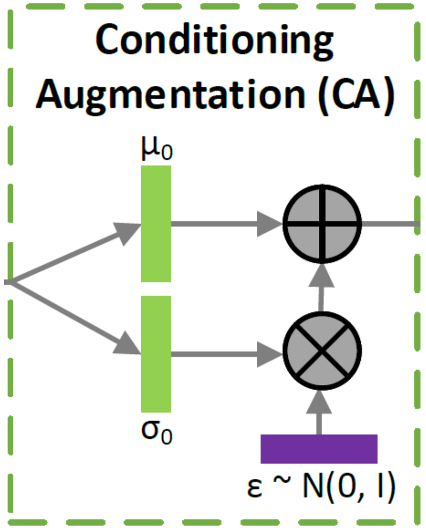


StackGAN: Text to Photo-realistic Images

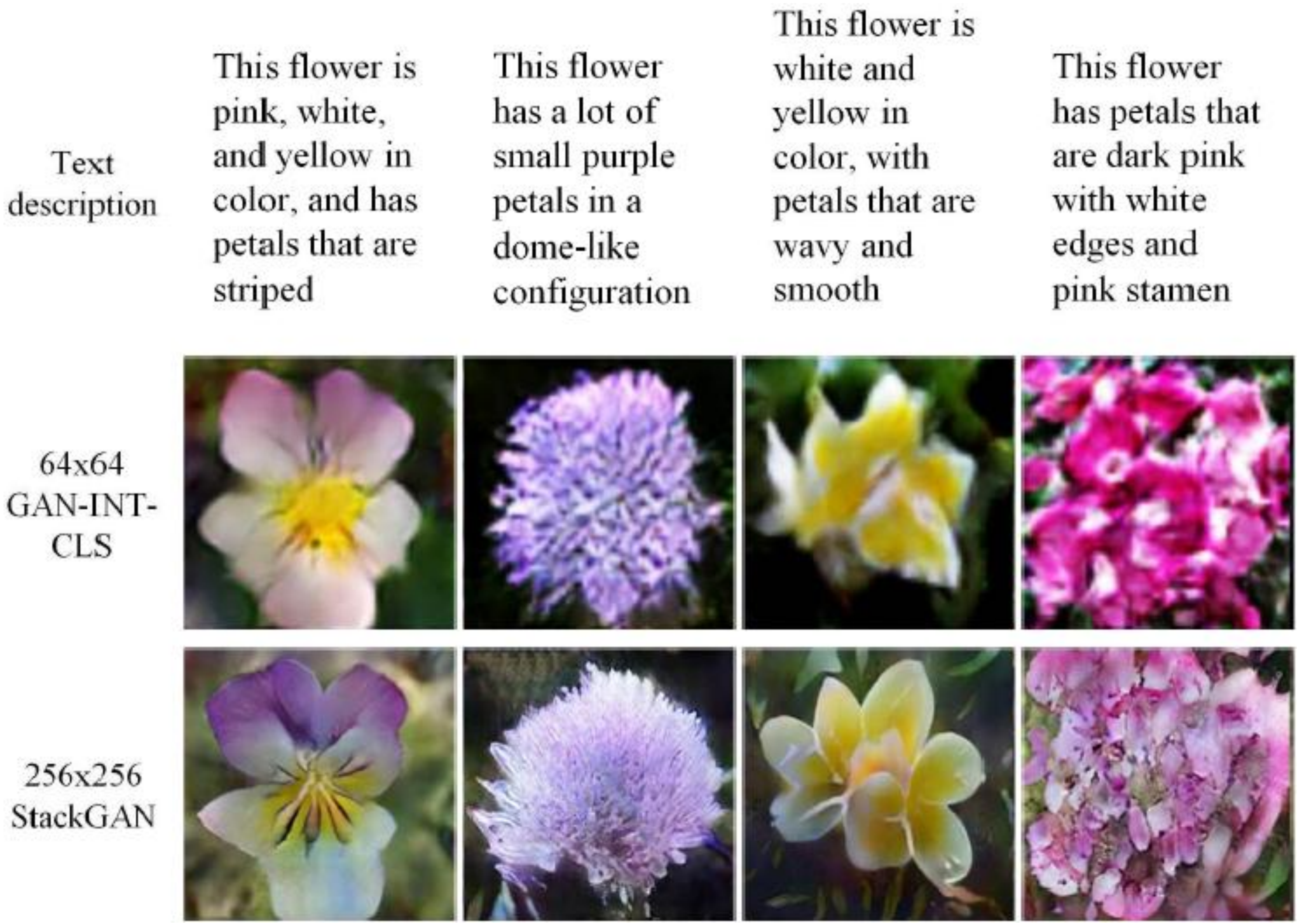
- Conditioning augmentation
- No random noise vector ***z*** for Stage-2
- Conditioning both stages on text help achieve better results
- Spatial replication for the text conditional variable
- Negative samples for *D*
  – True images + non-corresponding texts
  – Synthetic images + corresponding texts

Conditioning Augmentation

- How train parameters like the mean and variance of a Gaussian distribution
- $N(\mu_0, \sigma_0)$
- Sample from standard Normal distribution $N(0,1)$
- Multiple with σ_0 and then add with μ_0
- The re-parameterization trick



More StackGAN Results on Flower



More StackGAN Results on COCO

A picture of a very clean living room

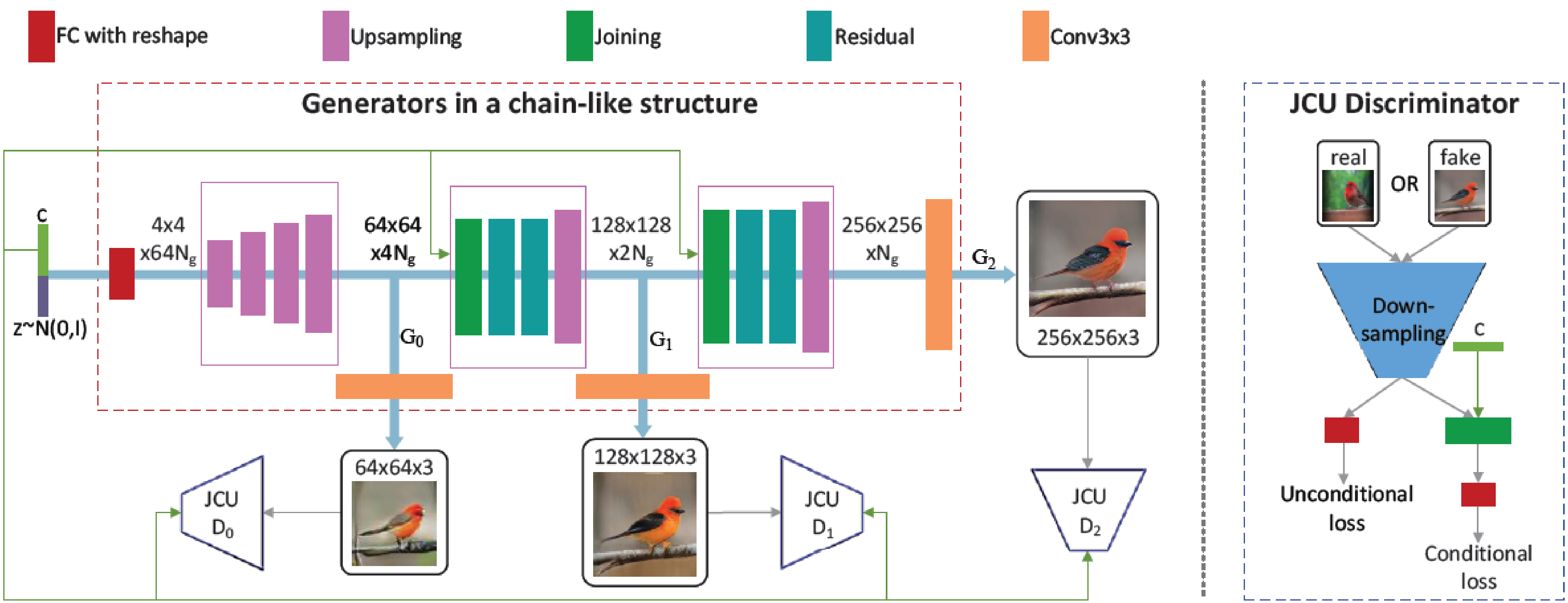
Eggs fruit candy nuts and meat served on white dish

A street sign on a stoplight pole in the middle of a day

A group of people on skis stand in the snow

StackGAN-v2: Architecture

- Approximate multi-scale image distributions jointly
- Approximate conditional and unconditional image distributions jointly



StackGAN-v2: Results

256×256 samples by our StackGAN-v2 on LSUN bedroom dataset

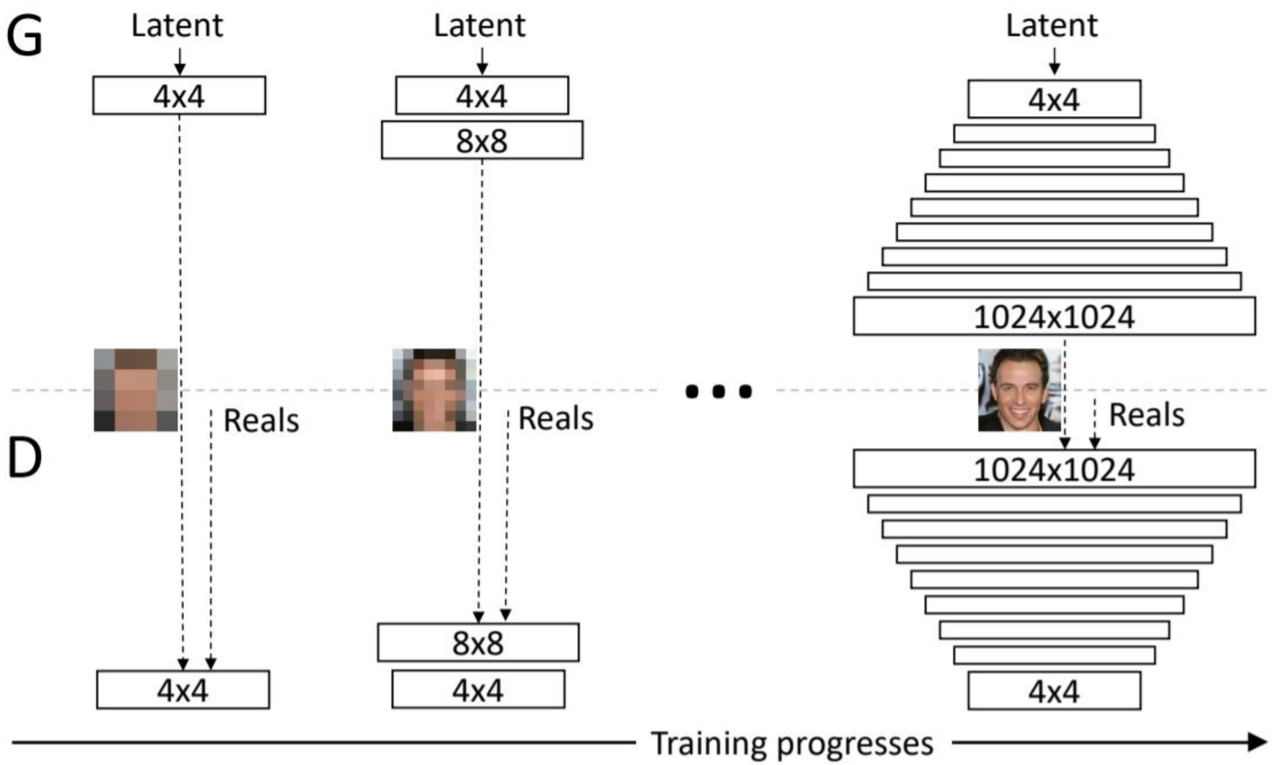
256×256 samples by our StackGAN-v2 on LSUN church dataset

256×256 samples by StackGAN-v2 on ImageNet dog dataset

256×256 samples by StackGAN-v2 on ImageNet cat dataset

Progressive Growing of GAN

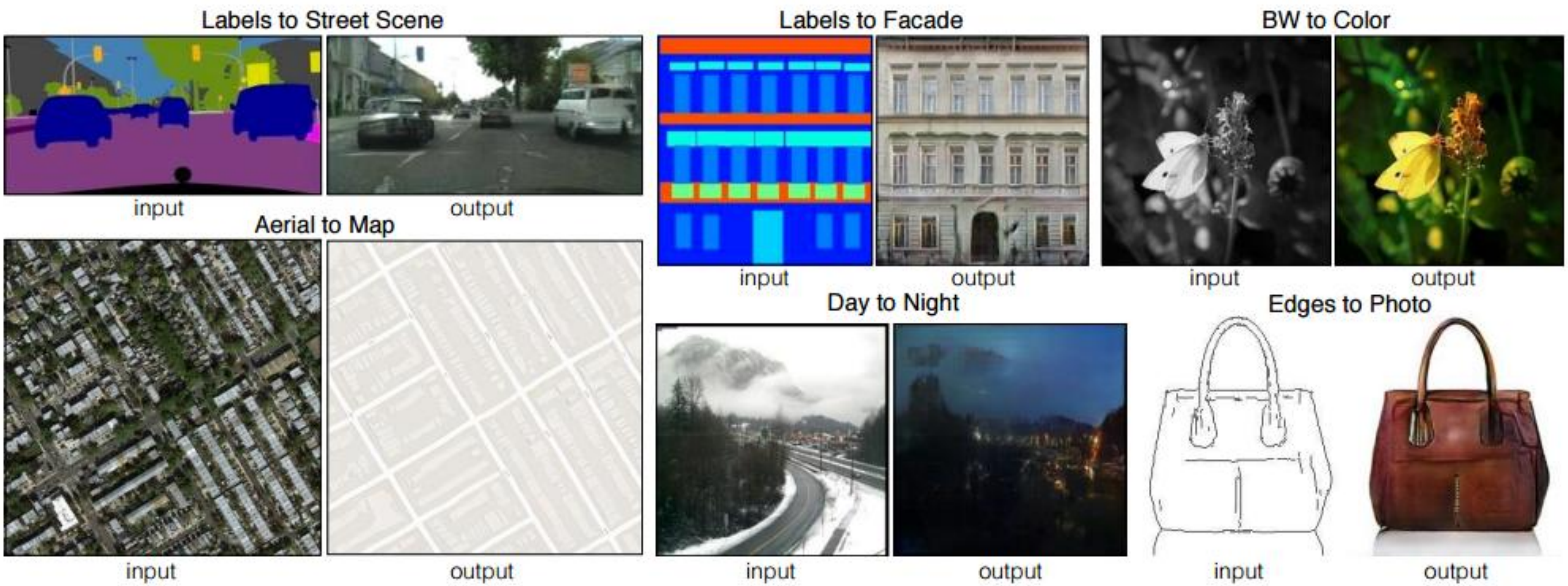
- Share the similar spirit with StackGAN-v1/-v2 but use a different training strategy



Progressive Growing of GAN

- Impressively realistic face images

Image-to-Image Translation with Conditional GAN



Isola et al. 2016

Image-to-Image Translation with Conditional GAN

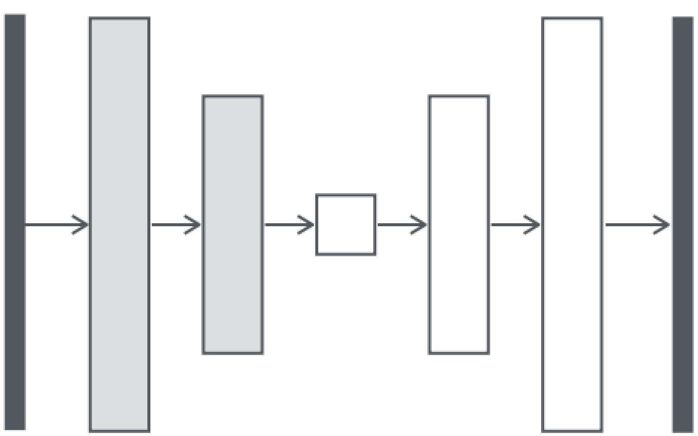
- Incorporate L1 loss into the objective function

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

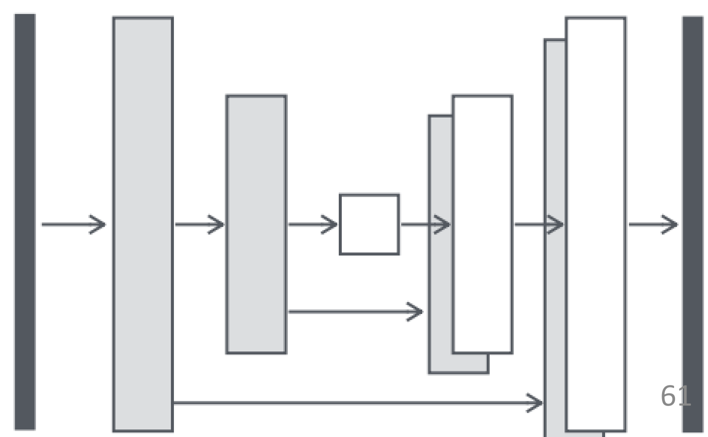
$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda\mathcal{L}_{L1}(G).$$

- Adopt the U-net structure for the generator

Encoder-decoder

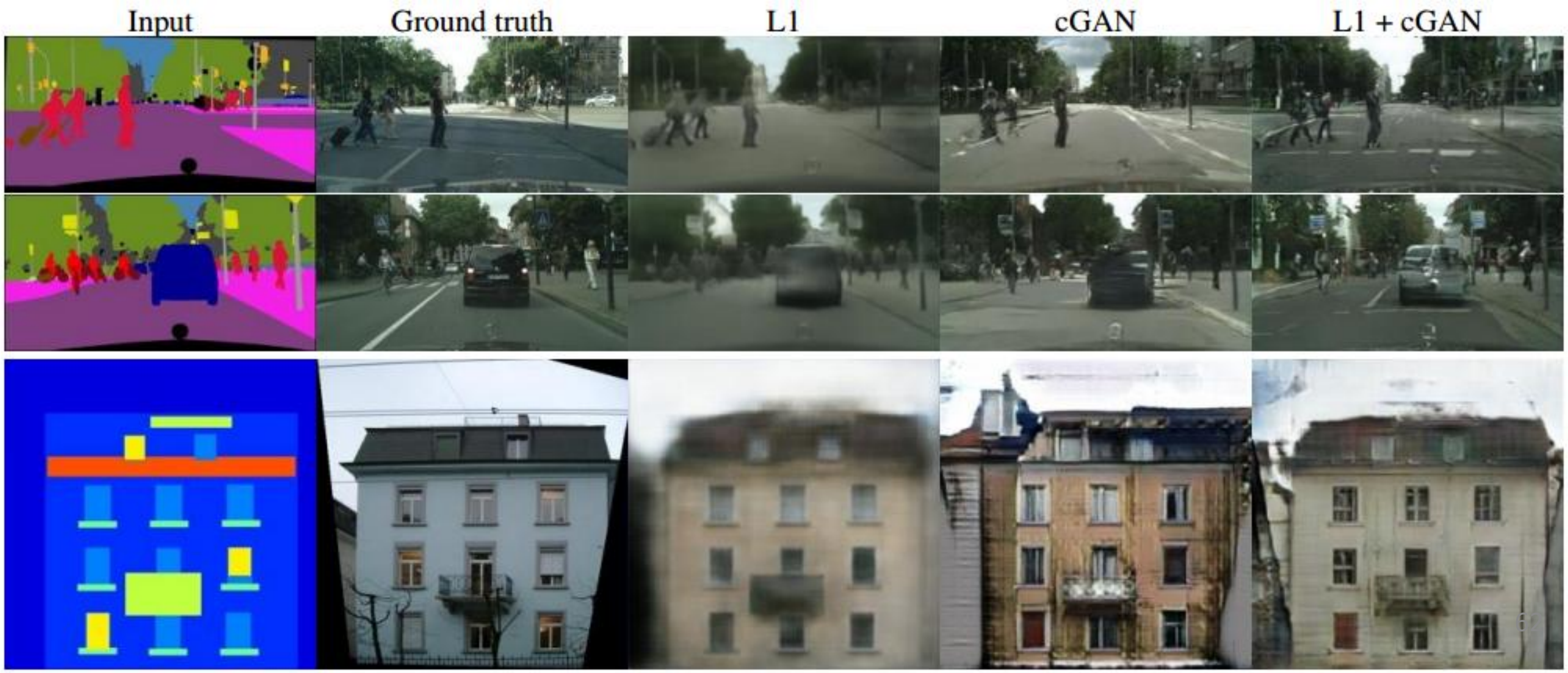


Encoder-decoder with skips



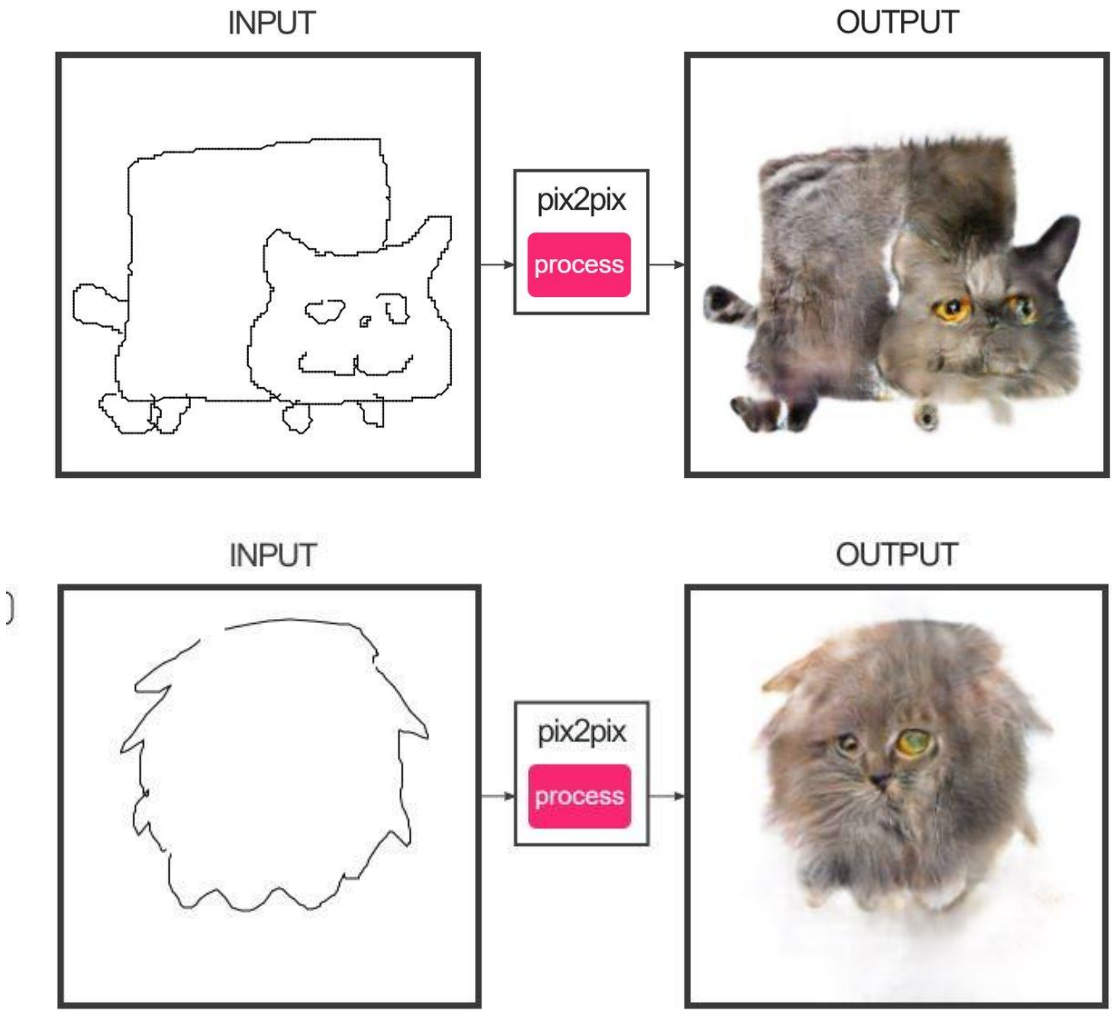
Patch-based Discriminator

- Separate each image into *N* x *N* patches
- Instead of distinguish whether the whole image is real or fake, train a patch-based discriminator



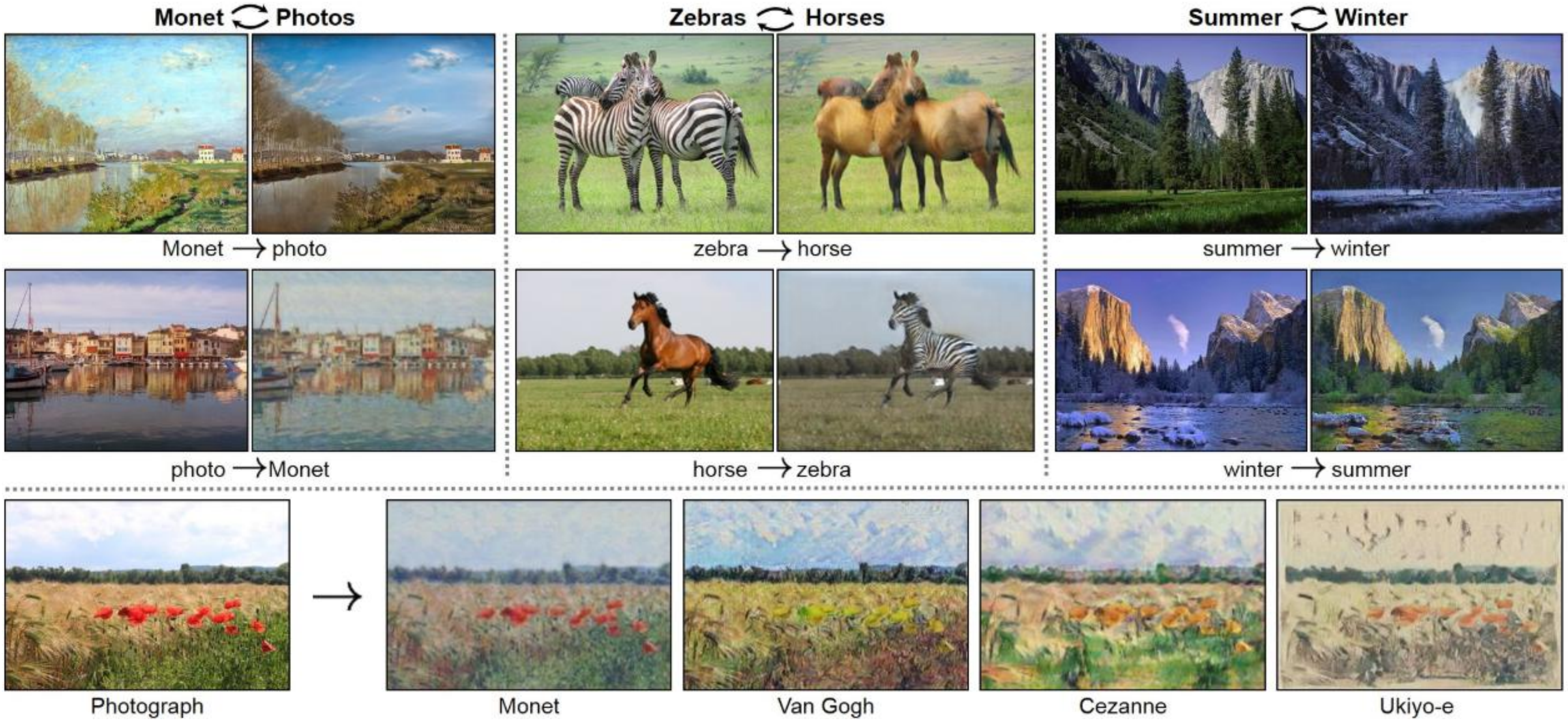
More Results

More Results



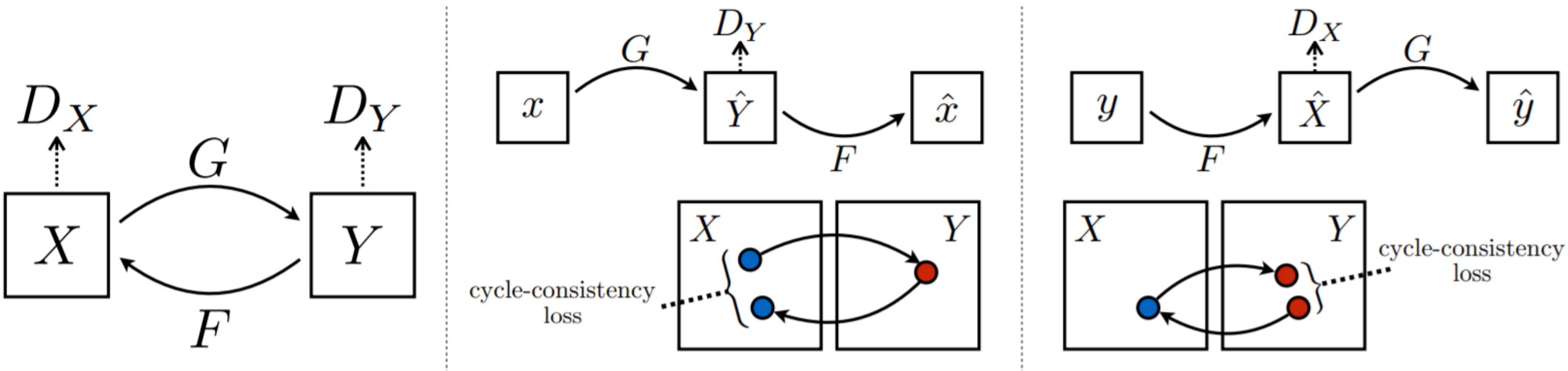
CycleGAN

- All previous methods require to have paired training data, i.e., exact input-output pairs, which can be extremely difficult to obtain in practice



CycleGAN

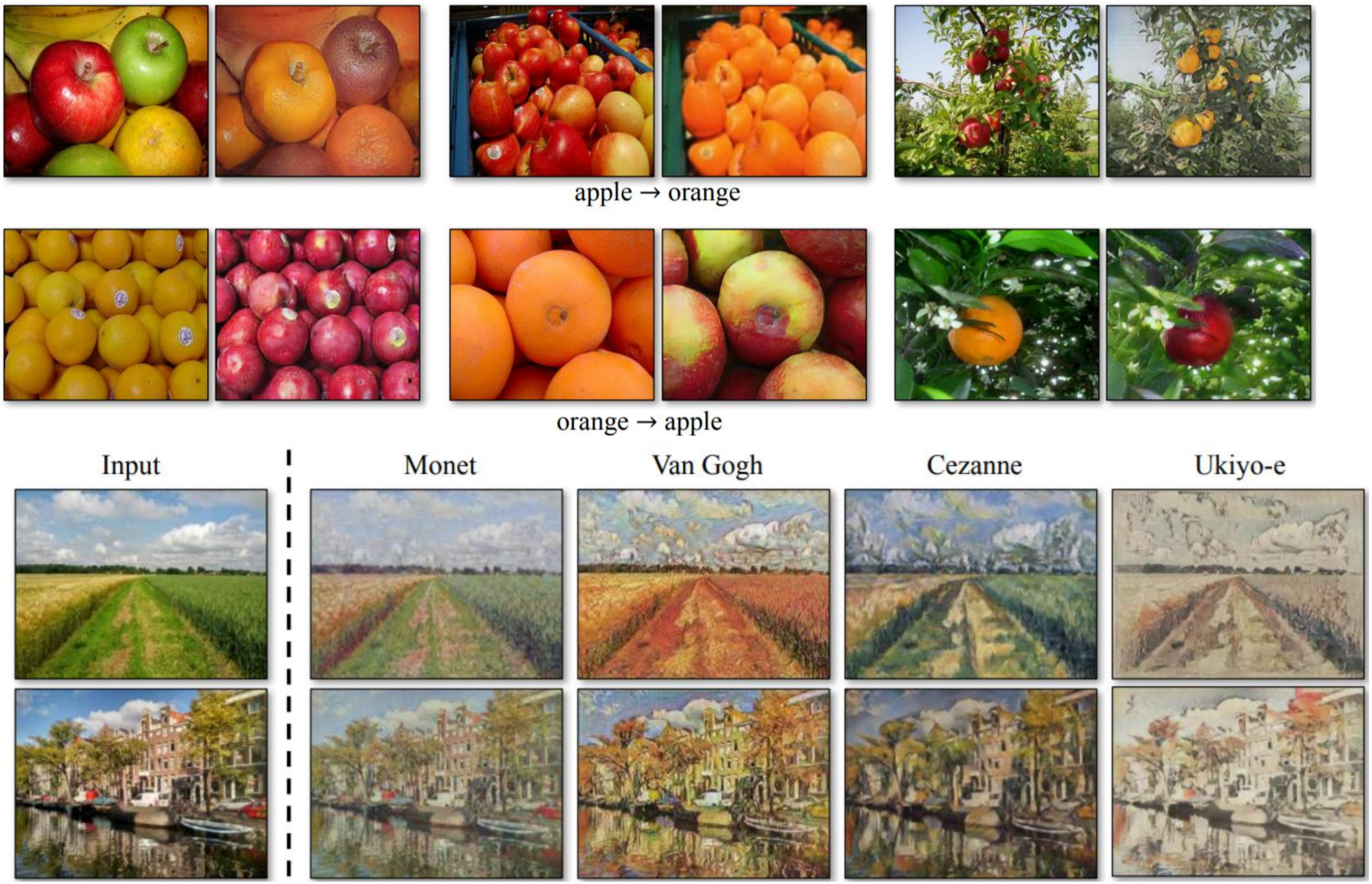
- The framework learns two mapping functions (generators) *G : X → Y* and *F : Y → X* with two domain discriminators D_X and D_Y



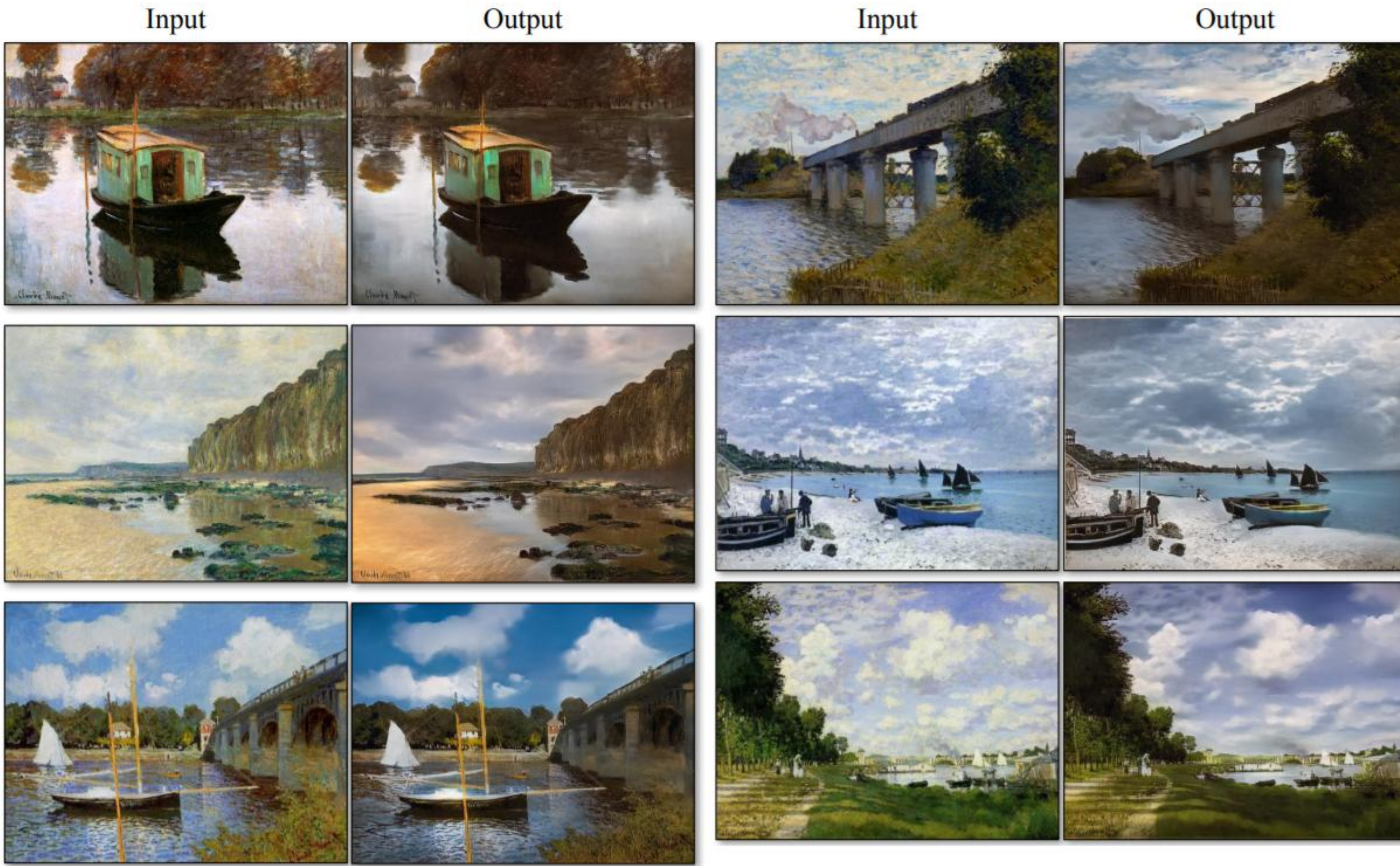
$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))],$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

CycleGAN: Results

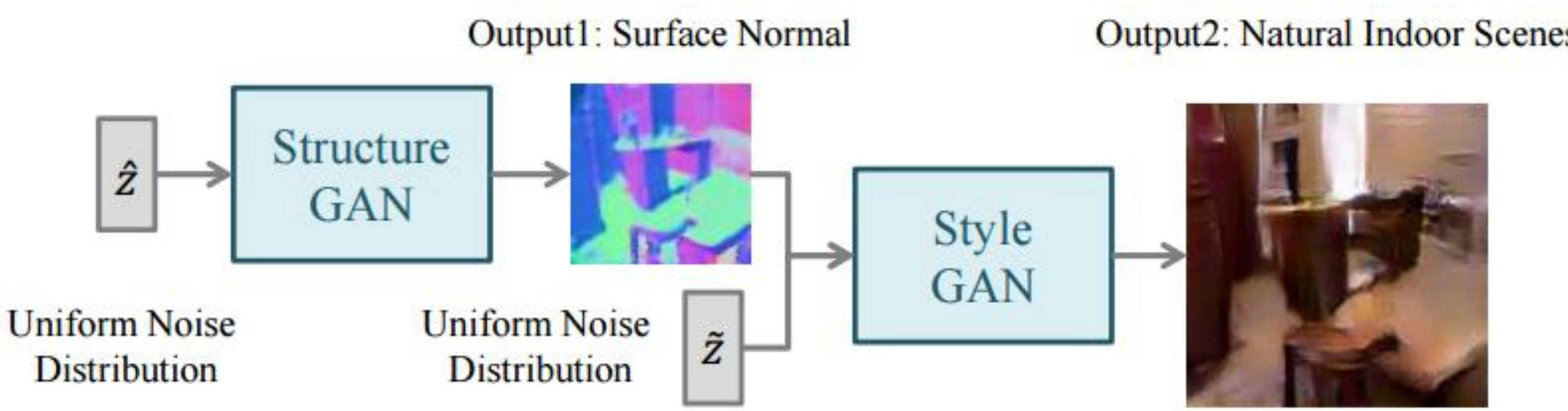


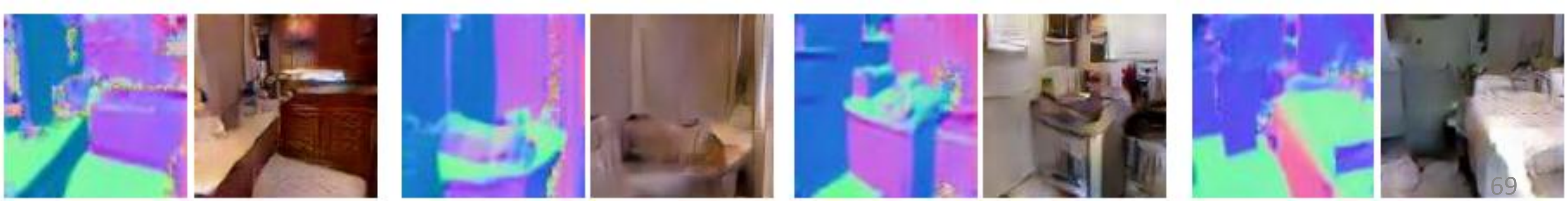
CycleGAN: Results



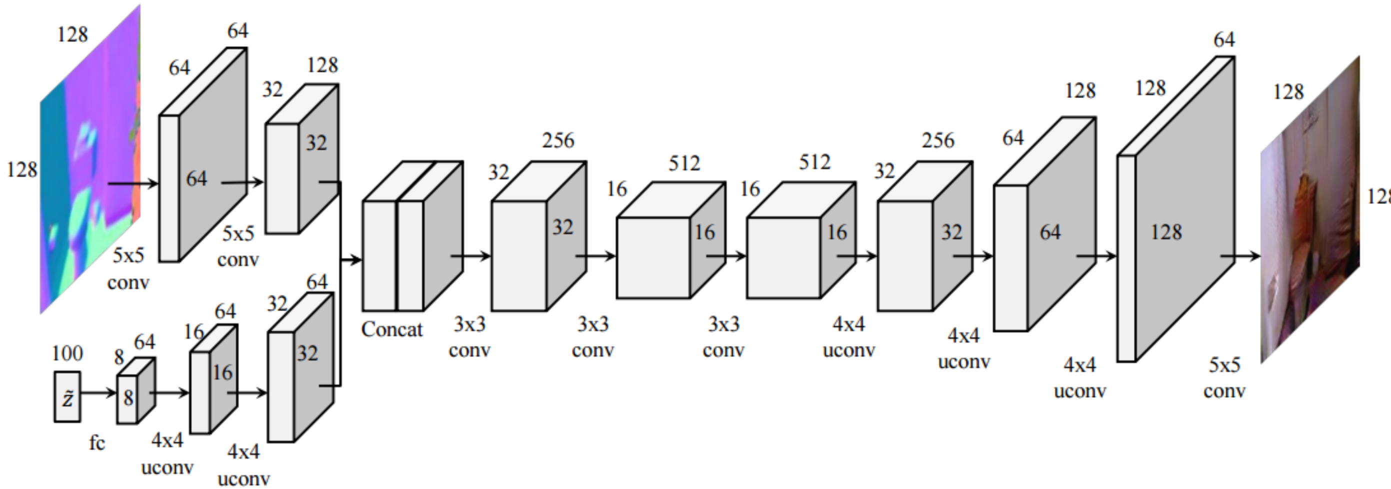
S^2-GAN: Decomposing difficult problems into subproblems

- Generating indoor images
- Generating surface normal map + surface style map

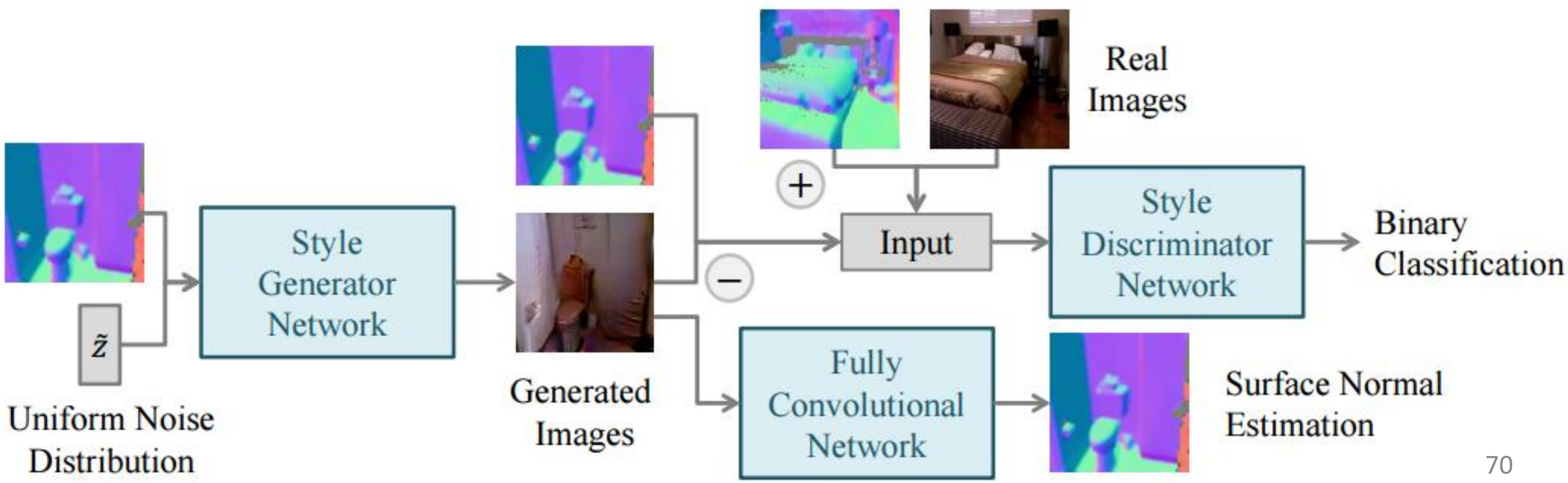




Style-GAN

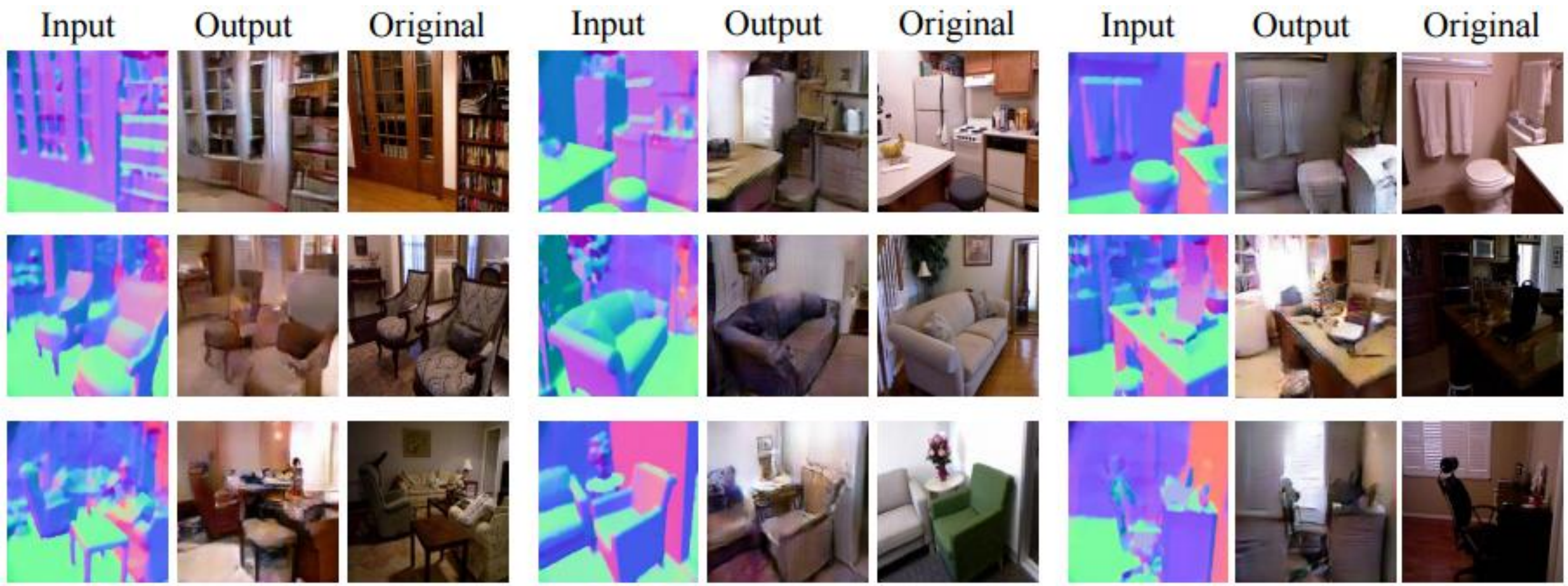


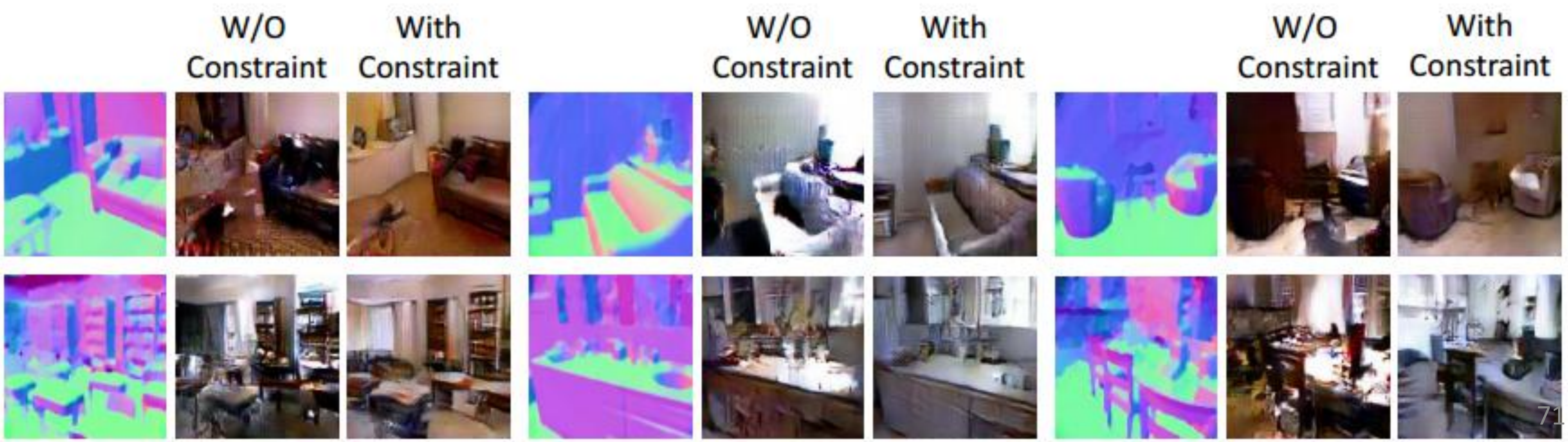
128
128
64
64
64
32
128
32
5x5
conv
5x5
conv
100
$\tilde{z}$
8
8
fc
64
16
16
64
4x4
uconv
32
32
64
4x4
uconv
Concat
32
256
32
3x3
conv
16
512
16
3x3
conv
16
512
16
3x3
conv
32
256
32
4x4
uconv
64
128
64
4x4
uconv
128
64
128
4x4
uconv
5x5
conv
128
128



$\tilde{z}$
Uniform Noise
Distribution
Style
Generator
Network
Generated
Images
Real
Images
+
−
Input
Style
Discriminator
Network
Binary
Classification
Fully
Convolutional
Network
Surface Normal
Estimation

S²-GAN Results





Insights

- Some insights
  - Decomposing the problems into easier problems
  - Spatially well-aligned conditioning variables are generally better

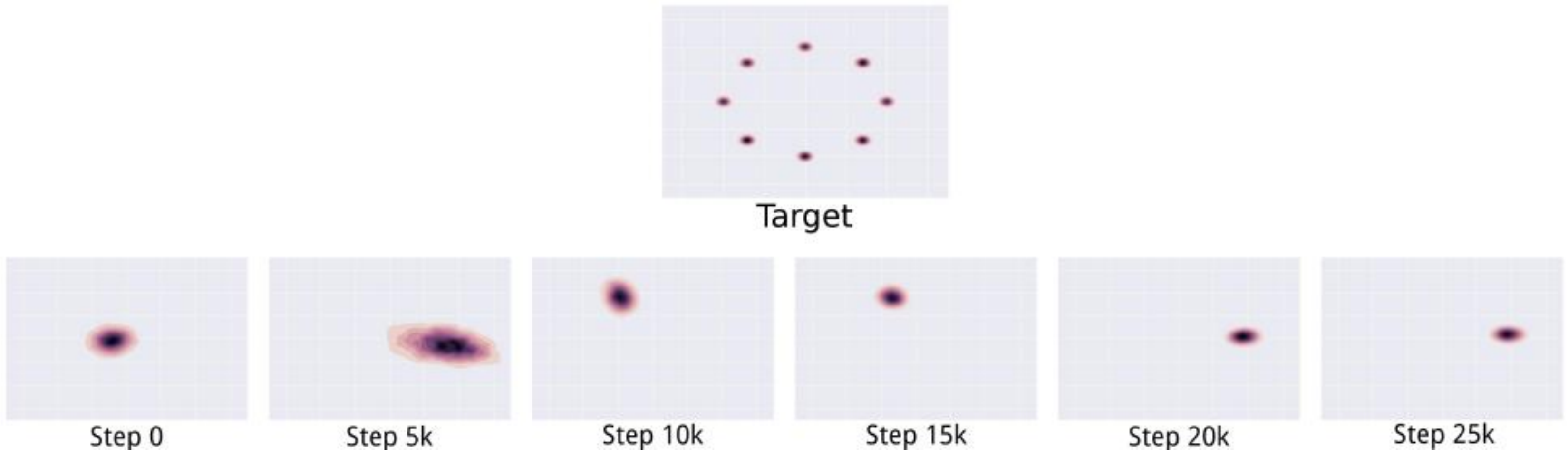
Non-convergence in GANs

- Finding equilibrium is a game of two players
- Exploiting convexity in function space, GAN training is theoretically guaranteed to converge if we can modify the density functions directly, but:
  - Instead, we modify G (sample generation function) and D (density ratio), not densities
  - We represent *G* and *D* as highly non-convex parametric functions
- "Oscillation": can train for a very long time, generating very many different categories of samples, without clearly generating better samples
- Mode collapse: most severe form of non-convergence

Mode Collapse

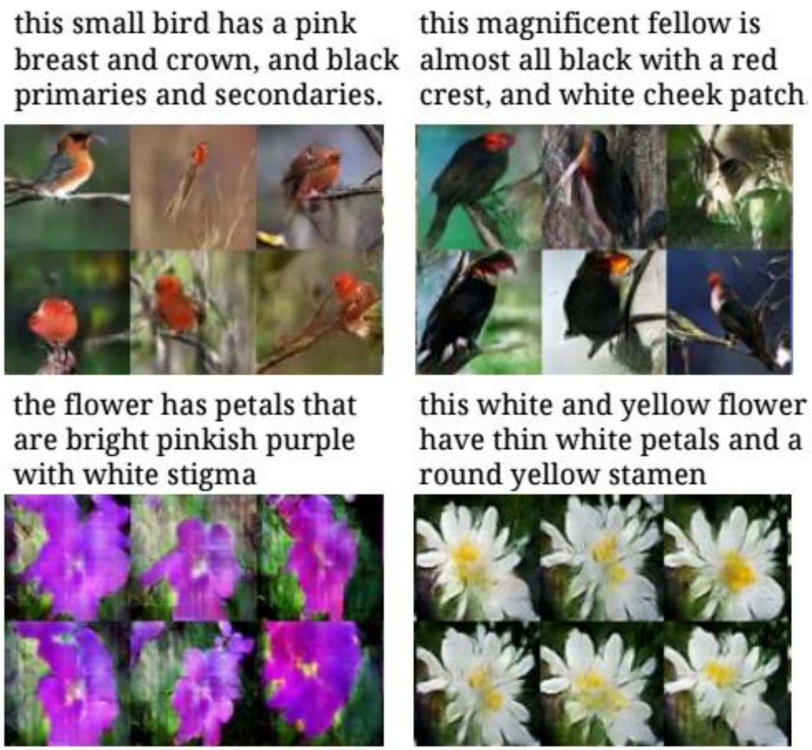
$$\min_G \max_D V(G, D) \neq \max_D \min_G V(G, D)$$

- *D* in inner loop: convergence to correct distribution
- *G* in inner loop: place all mass on most likely point



Metz et al 2016

Mode Collapse Causes Low Output Diversity



(Reed et al 2016)

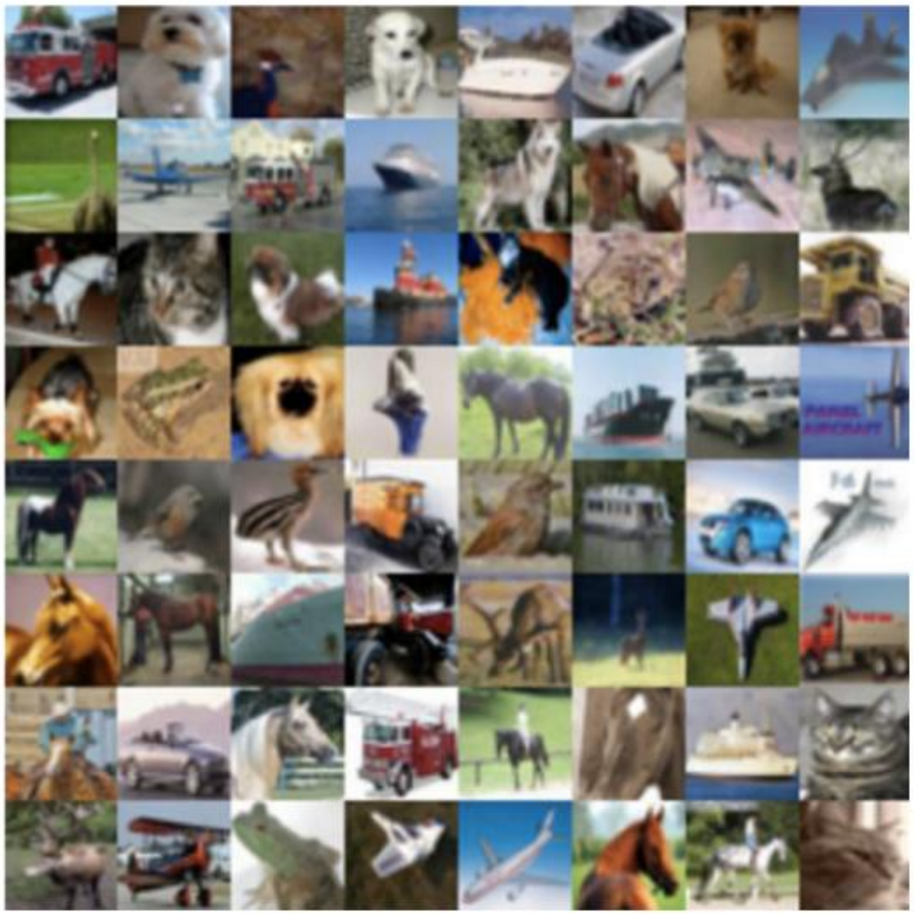
(Reed et al, submitted to ICLR 2017)

Conditioning Augmentation

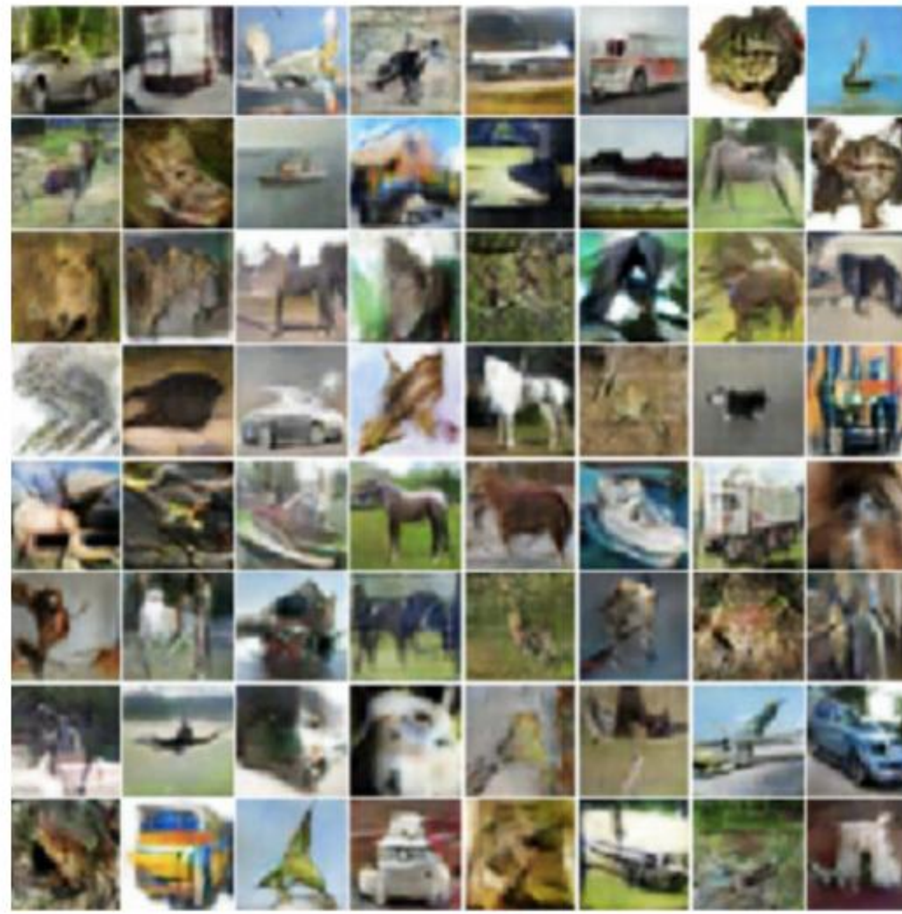
Minibatch Features

- Add minibatch features that classify each example by comparing it to other members of the minibatch (Salimans et al 2016)
- Nearest-neighbor style features detect if a minibatch contains samples that are too similar to each other

Minibatch GAN on CIFAR

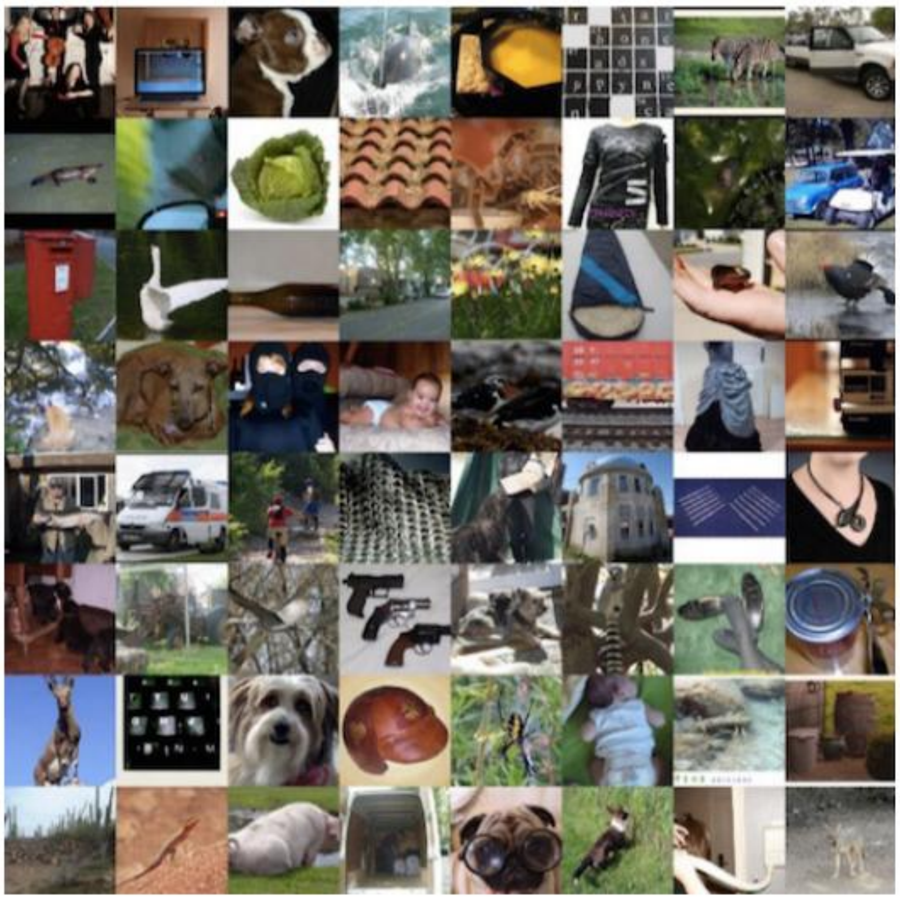


Training Data



Samples

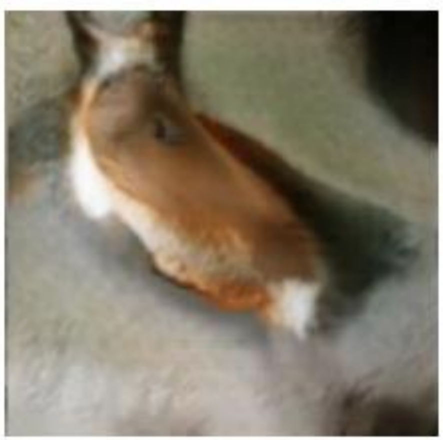
Minibatch GAN on ImageNet



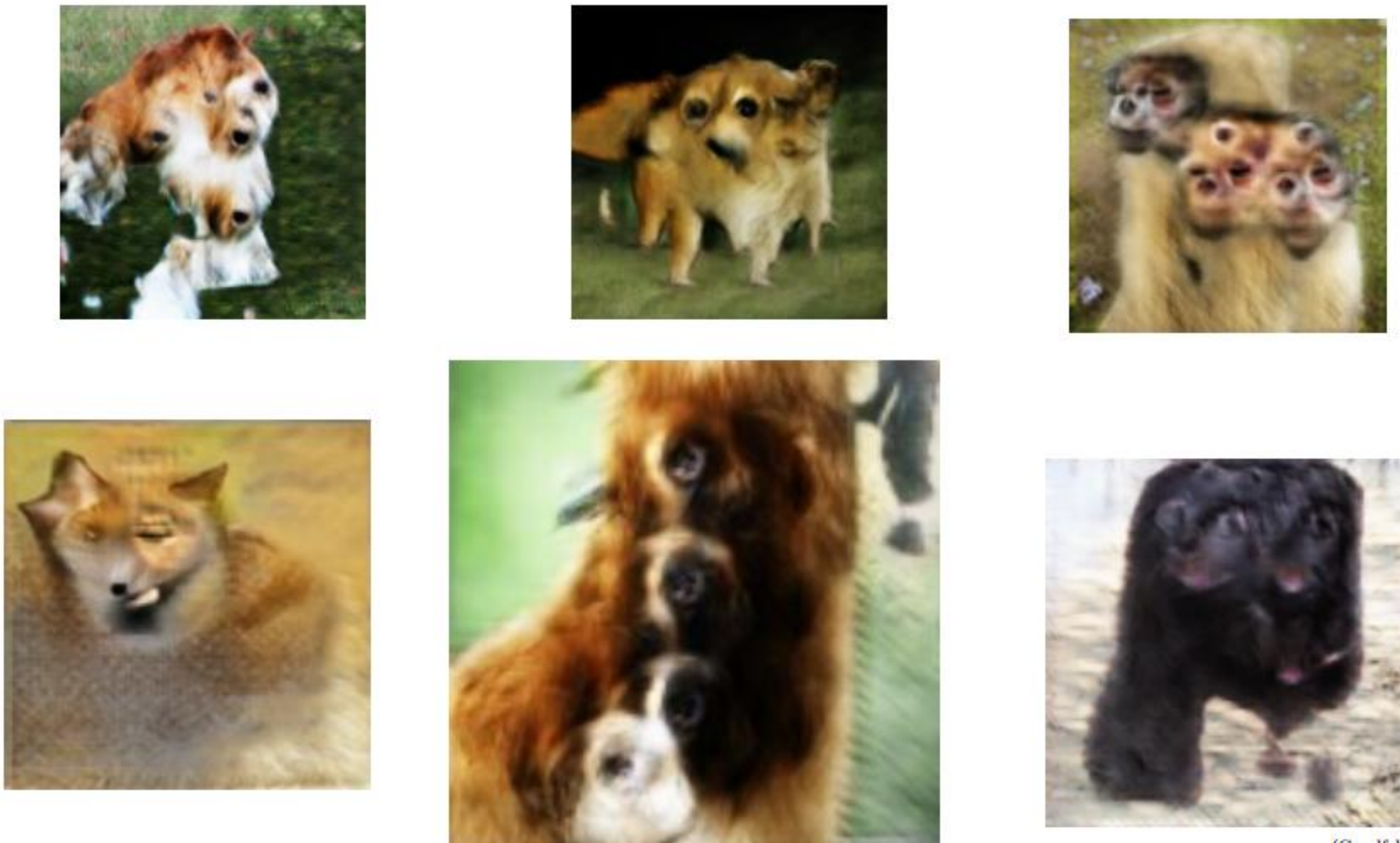
Cherry-picked Results







Problems with Counting



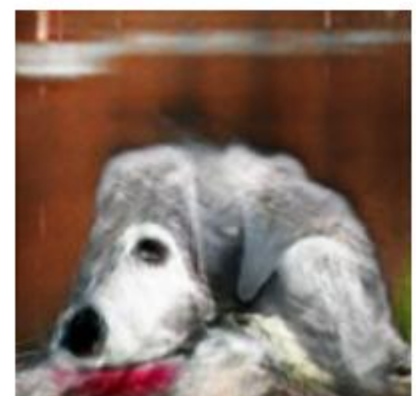
(Goodfellow 2016)

Problems with Perspective



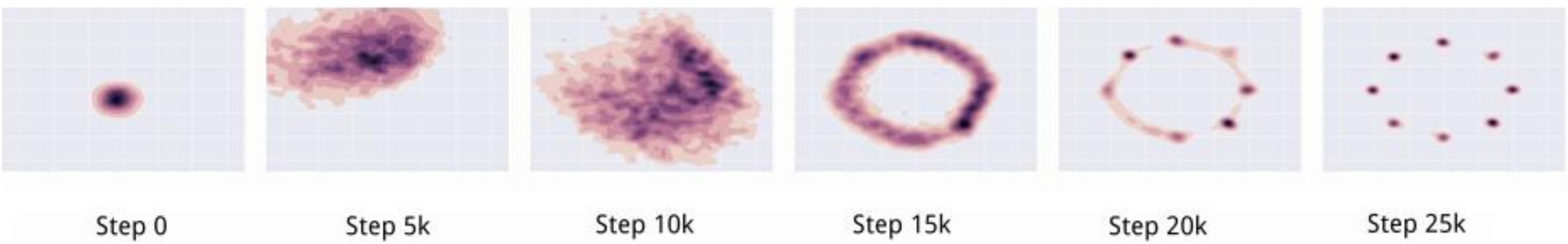






Unrolled GANs

- A toy example



Metz et al 2016

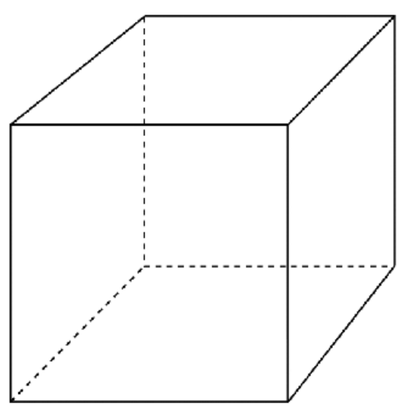
Tips and Tricks for training GAN

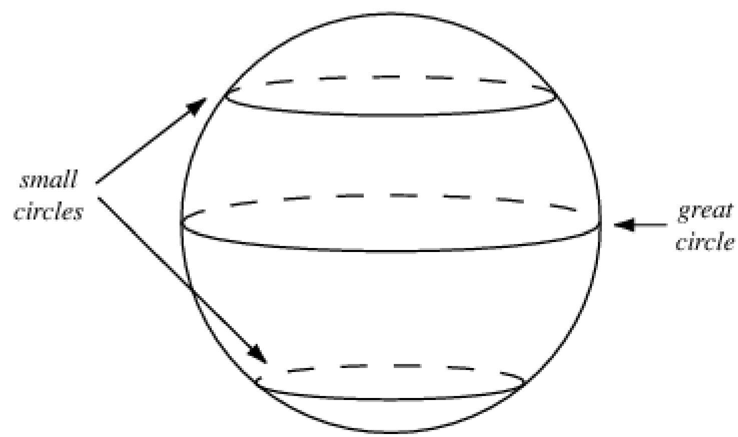
- **Normalize the inputs**
  - normalize the images between -1 and 1
  - Tanh as the last layer of the generator output
- **Modified loss function**
  - Because of the vanishing gradients (Goodfellow et al 2014). Use $J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$
  - Flip labels when training generator: real = fake, fake = real

Soumith et al 2016

Tips and Tricks for training GAN

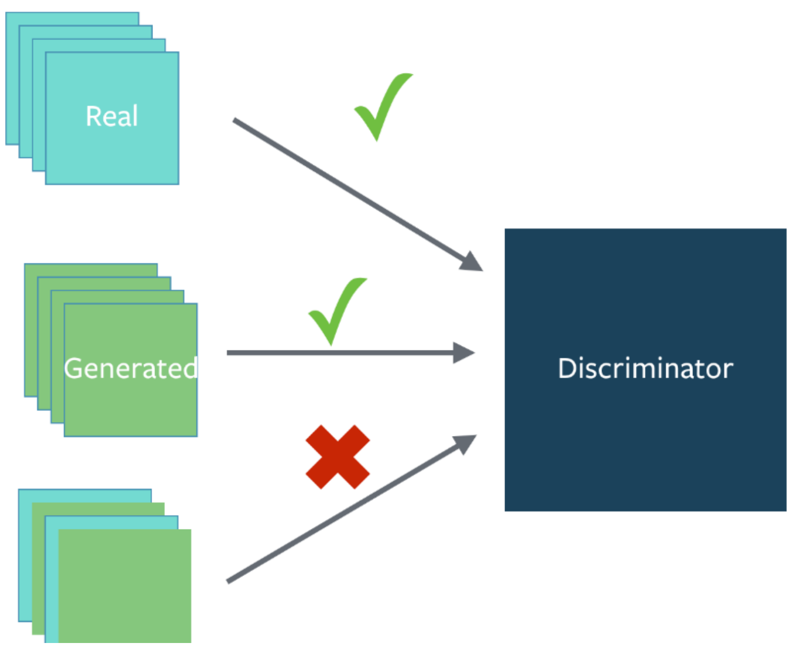
- **Use a spherical Z**
  - Don't sample from a Uniform distribution
  - Sample from a Gaussian distribution
  - When doing interpolations, do the interpolation via a great circle, rather than a straight line (White et al 2016)





Tips and Tricks for training GAN

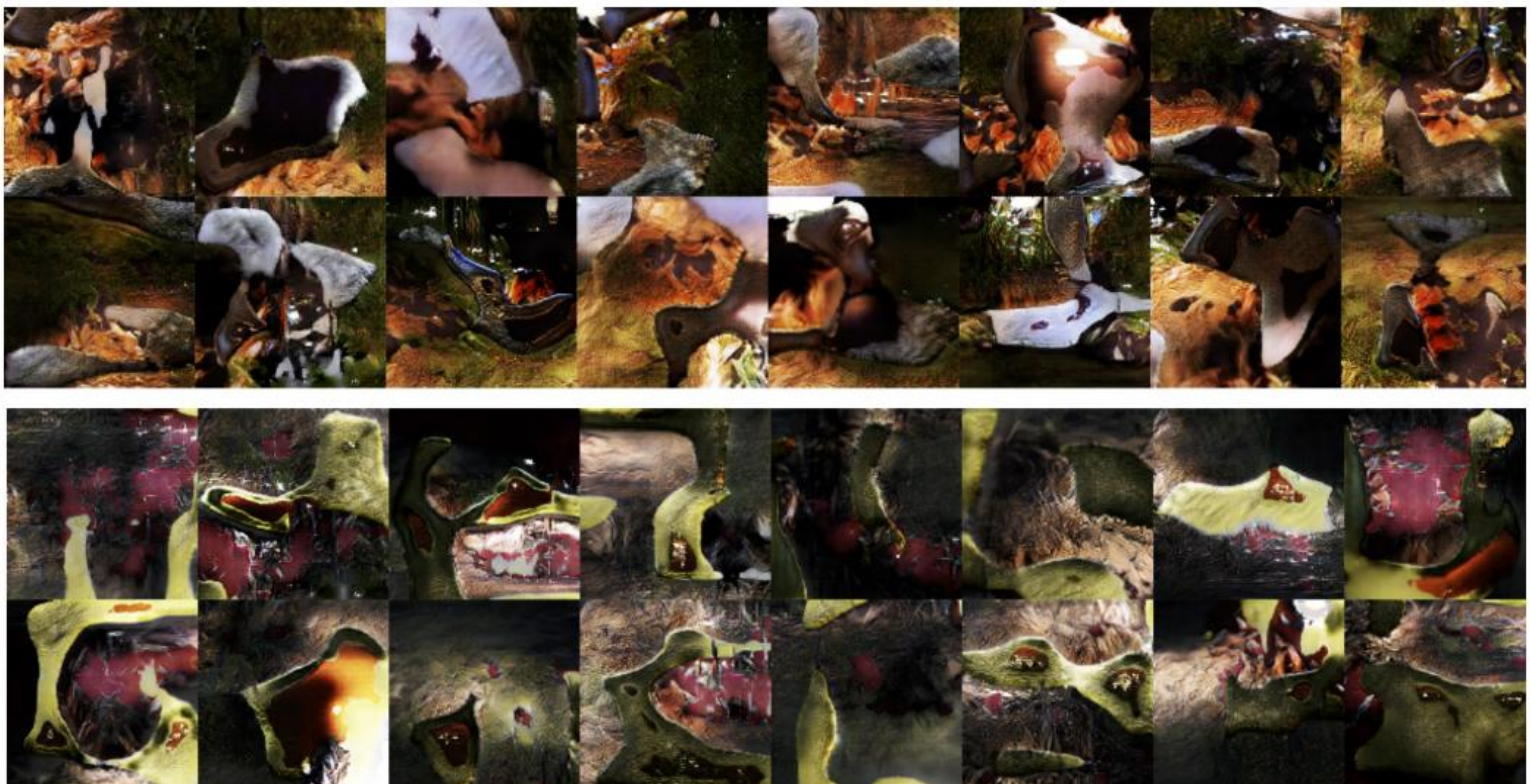
- **Batch normalization**
  - Compute mean and standard deviation of features
  - Normalize features (subtract mean, divide by standard deviation)



Soumith et al 2016

Tips and Tricks for training GAN

- **Batch normalization in *G***



Soumith et al 2016

Tips and Tricks for training GAN

- **Reference Batch Normalization**
  - Fix a reference batch $R=\{r^{(1)}, r^{(2)}, .., r^{(m)}\}$
  - Given new inputs $X=\{x^{(1)}, x^{(2)}, .., x^{(m)}\}$
  - Normalize the features of *X* using the mean and standard deviation from *R*
  - Every $x^{(i)}$ is always treated the same, regardless of which other examples appear in the minibatch

Salimens et al 2016

Tips and Tricks for training GAN

- **Virtual Batch Normalization**
  - Reference batch norm can overfit to the reference batch. A partial solution is virtual batch norm
  - Fix a reference batch $R = \{r^{(1)}, r^{(2)}, .., r^{(m)}\}$
  - Given new inputs $X = \{x^{(1)}, x^{(2)}, .., x^{(m)}\}$
  - For each $x^{(i)}$
    - Construct a minibatch containing $x^{(i)}$ and all *R*
    - Compute mean and standard deviation of *V*
    - Normalize the features of $x^{(i)}$ using the mean and standard deviation

Salimens et al 2016

Tips and Tricks for training GAN

- **Use Adam optimizer**
  - Use SGD for discriminator & Adam for generator

- **Avoid Sparse Gradients: ReLU, MaxPool**
  - the stability of the GAN game suffers if you have sparse gradients
  - LeakyReLU = good (in both *G* and *D*)
  - For Downsampling, use: Average Pooling, Conv2d + stride
  - For Upsampling, use: Bilinear Interpolation, PixelShuffle

Tips and Tricks for training GAN

- **Use Soft and Noisy Labels**
  - Default cost

```
   cross_entropy(1., discriminator(data))
 + cross_entropy(0., discriminator(samples))
```

  - Label Smoothing (Salimans et al. 2016)
  - For real ones (label=1), replace it with 0.9; For fake ones (label=0), keep it to 0.

```
   cross_entropy(.9, discriminator(data))
 + cross_entropy(0., discriminator(samples))
```

Tips and Tricks for training GAN

- **Use Soft and Noisy Labels**

```
cross_entropy(1.-alpha, discriminator(data))
+ cross_entropy(beta, discriminator(samples))
```

$$D(\boldsymbol{x}) = \frac{(1-\alpha)p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

  – make the labels noisy for the discriminator: occasionally flip the labels when training the discriminator

Tips and Tricks for training GAN

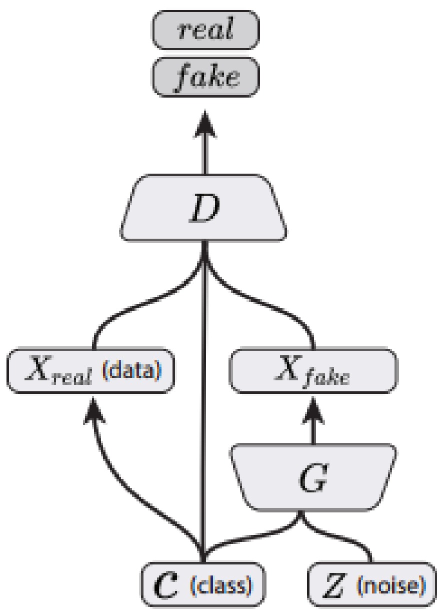
- **Track failures early**
  - D loss goes to 0: failure mode
  - Check norms of gradients: > 100 is bad
  - When training well, *D* loss has low variance and is going down. Otherwise, *D* loss is spiky
  - If loss of *G* steadily decreases, then it's fooling D with garbage

Tips and Tricks for training GAN

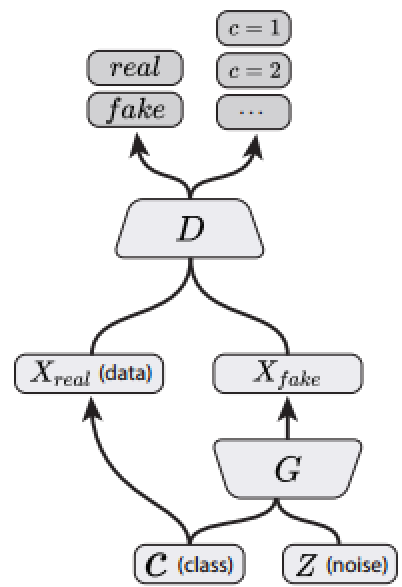
- **Discrete variables in Conditional GANs**
  - Use an Embedding layer
  - Add as additional channels to images
  - Keep embedding dimensionality low and upsample to match image channel size

Tips and Tricks for training GAN

- **Use label information when possible**
  - Used as conditioning variable
  - Auxiliary classifier GAN



Conditional GAN



AC-GAN

Tips and Tricks for training GAN

- **Balancing *G* and *D***
  – Usually the discriminator “wins”
  – Good thing: theoretical justification are based on assuming *D* is perfect
  – Usually *D* is bigger and deeper than *G*
  – Sometimes run *D* more often than *G*. Mixed results
  – Do not try to limit *D* to avoid making it “too smart”
    - Use non-saturating cost
    - Use label smoothing

Research Directions

- Research direction of GANs
  - Better network structures
  - Better objective functions
  - Novel problem setups
  - Use of adversarial losses in other CV/ML applications
  - Theories on GAN