Deep Generative Models

Saqib Shamsi

SHALA2020.GITHUB.IO

Unsupervised Learning

- No labels are provided during training
- General objective: inferring a function to describe hidden structure from unlabeled data
  - Clustering (discrete labels)
  - Representation/feature learning (continuous vectors)
  - Dimensionality reduction (lower-dimensional representation)
  - Density estimation (continuous probability)

Clustering

- **Goal:** Group objects such that objects in one group (cluster) are likely to be different when compared to objects grouped under another group (cluster).

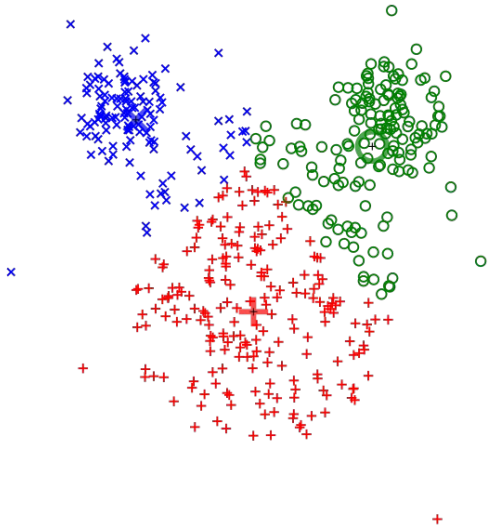


Image Source

Representation Learning

- **Goal:** “Learning representations of the data that make it easier to extract useful information when building classifiers or other predictors”[1]
- Eg. autoencoders, denoising autoencoders, self-supervision, word2vec, BERT

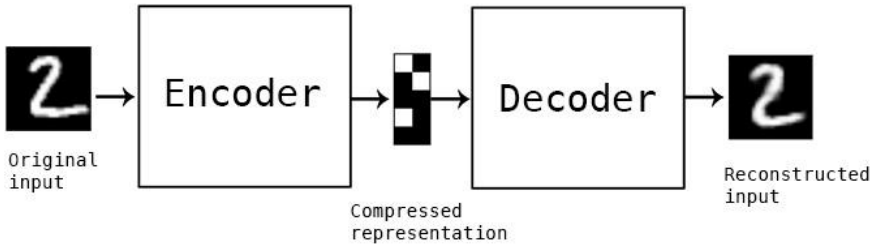


Image Source

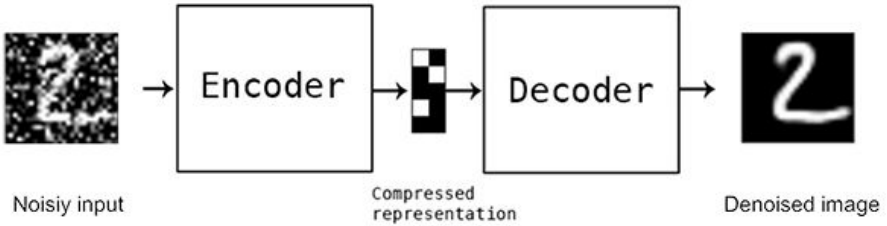
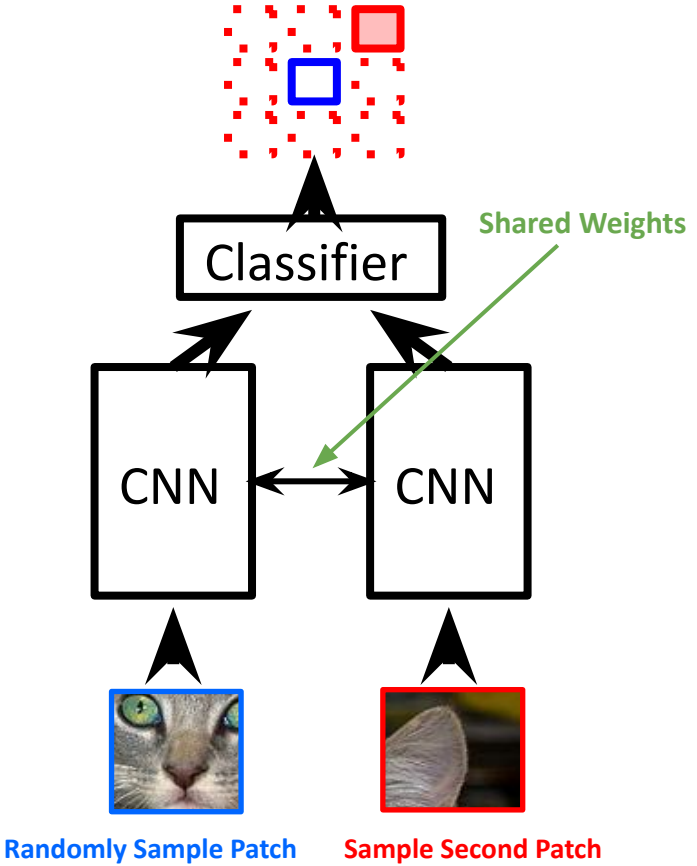


Image Source

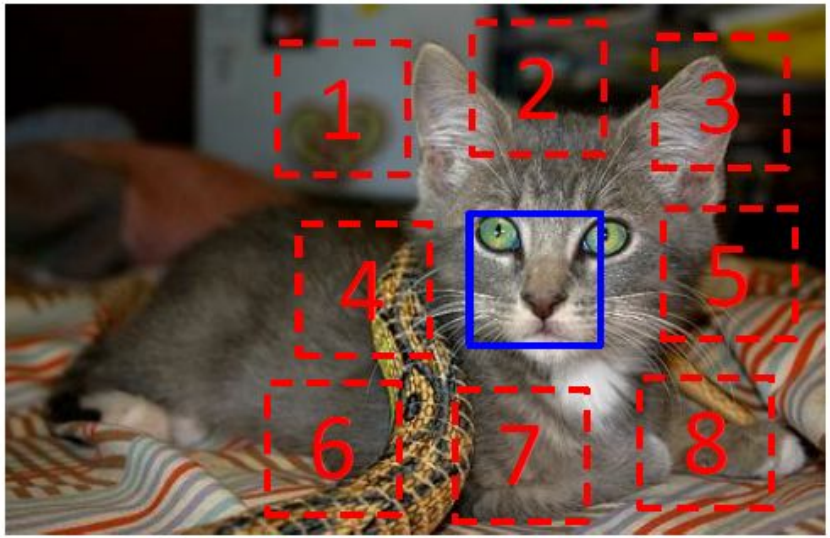
[1] Bengio, Y., Courville, A. & Vincent, P. (2012). Representation Learning: A Review and New Perspectives, arxiv:1206.5538

Representation Learning

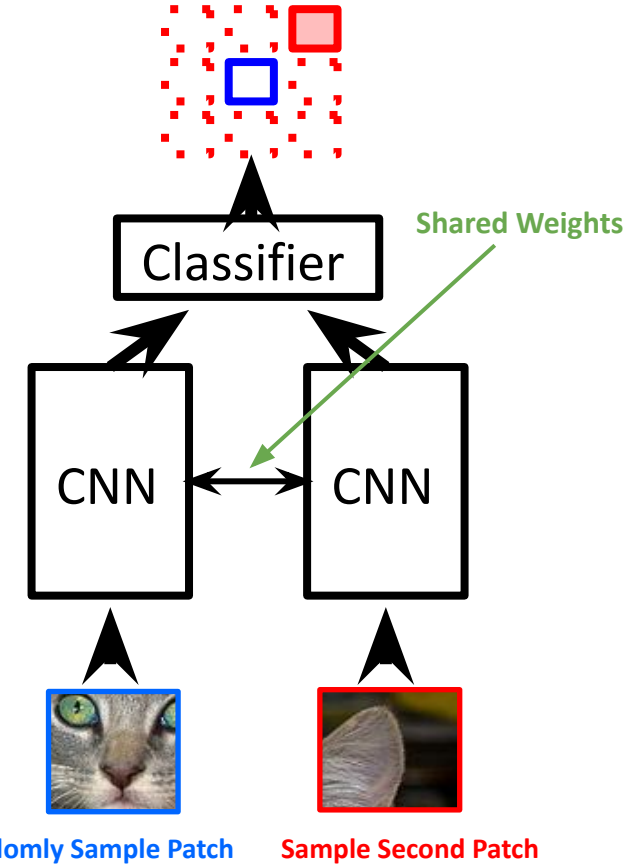


Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised Visual Representation Learning by Context Prediction. ICCV 2015.

Representation Learning



X = (,); Y = 3



Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised Visual Representation Learning by Context Prediction. ICCV 2015.

Density Estimation

- **Goal:** Estimate the probability density function for given data.

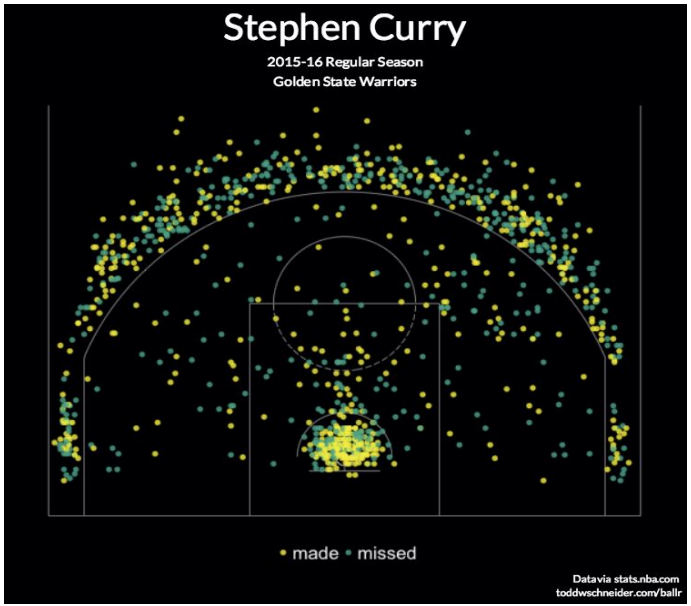


Image Source

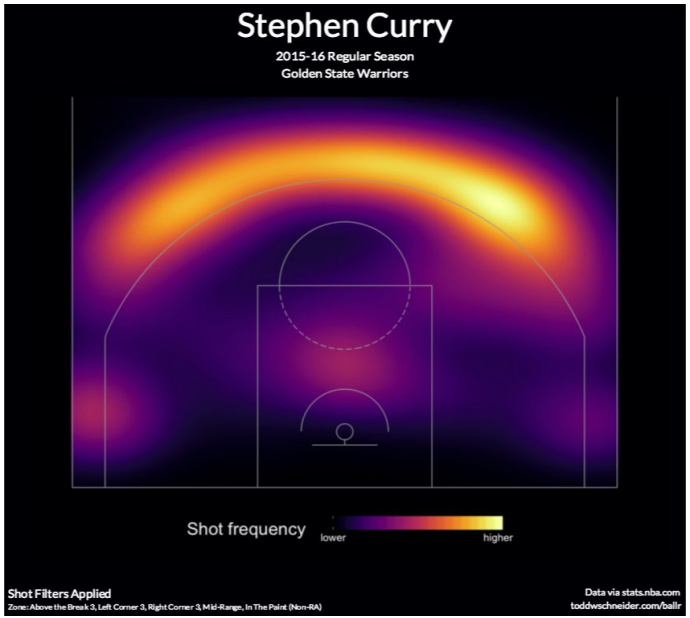


Image Source

Generative Models Motivation

- **Problem:** Given some high dimensional data, generate new samples from the same distribution.

Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Source

What About This Solution?

```
import glob

def augment(image):
    """
    Applies random rotations, brightness, contrast,
    flips, shearing, colour-shift and noise
    """
    # Code for augmenting `image`
    return augmented_image


def generate(given_data_path):
    images = glob.glob(given_data_path + '/*.jpg')
    for image in images:
        yield augment(image)
```

What About This One?

```
def generate_all(num_pixels, intensity_levels):
    """
    Generates all possible images with the number
    of pixels equal to `num_pixels` and of given
    `intensity_levels`
    """
    # Code to generate all possible images
    return all_images


def main():
    DESIRED_SIZE = 100 * 100 * 3
    INTENSITIES = 256
    all_images = generate_all(DESIRED_SIZE, INTENSITIES)
    # Pick whatever images you want from `all_images`
```

What About This One?

```
def generate_all(num_pixels, intensity_levels):
    """
    Generates all possible images with the number
    of pixels equal to `num_pixels` and of given
    `intensity_levels`
    """
    # Code to generate all possible images
    return all_images


def main():
    DESIRED_SIZE = 100 * 100 * 3
    INTENSITIES = 256
    all_images = generate_all(DESIRED_SIZE, INTENSITIES)
    # Pick whatever images you want from `all_images`
```

1.58×10^{72245}

What About This One?

```
def generate_all(num_pixels, intensity_levels):
    """
    Generates all possible images with the number
    of pixels equal to `num_pixels` and of given
    `intensity_levels`
    """
    # Code to generate all possible images
    return all_images


def main():
    DESIRED_SIZE = 100 * 100 * 3
    INTENSITIES = 256
    all_images = generate_all(DESIRED_SIZE, INTENSITIES)
    # Pick whatever images you want from `all_images`
```

Estimated #protons + #electrons in the observable universe: 10^{80}

$$1.58 \times 10^{72247}$$

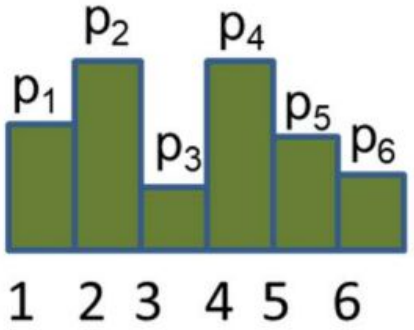
What We Want

- Meaningfully different samples. Not the same data in augmented form
  - Samples that have the same underlying pattern as the training data, but should be different from the training data
  - Eg, digits in a different handwriting, cats in new poses, paintings with different scenery and brushstrokes
- The generation process must be tractable

Generative Model

- A model for probability distribution of given data `X`
- Simple Generative Models:
  - **Multinomial PMF:** For discrete data. Can be modeled as a probability table.
    - Probability of observing a data point can be obtained directly from the table
  - **Gaussian PDF:** For continuous data. Distribution is parameterized by mean μ and standard deviation σ
    - Probability of observing a data point x can be obtained by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



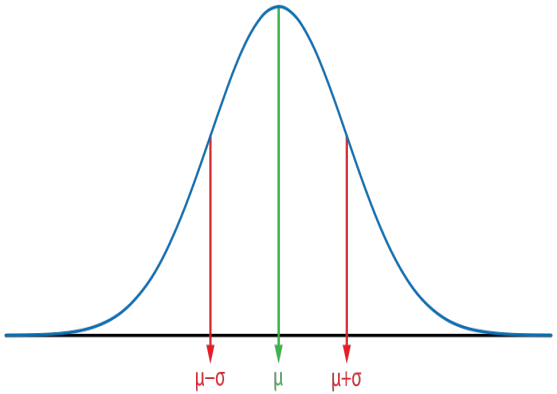


Image Source

Source

Stephen Curry Revisited

- The density function here would be of the form p(x, y), where (x, y) is the coordinate of a position on the court, p(x, y), would give the probability of Curry making a shot from that position

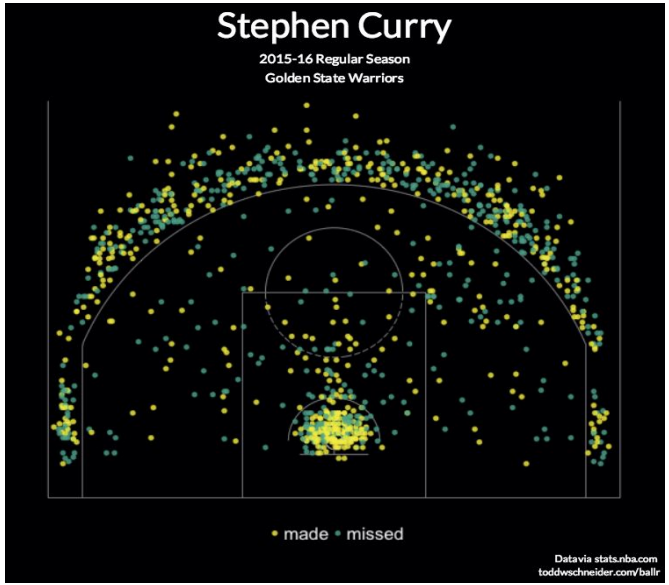


Image Source

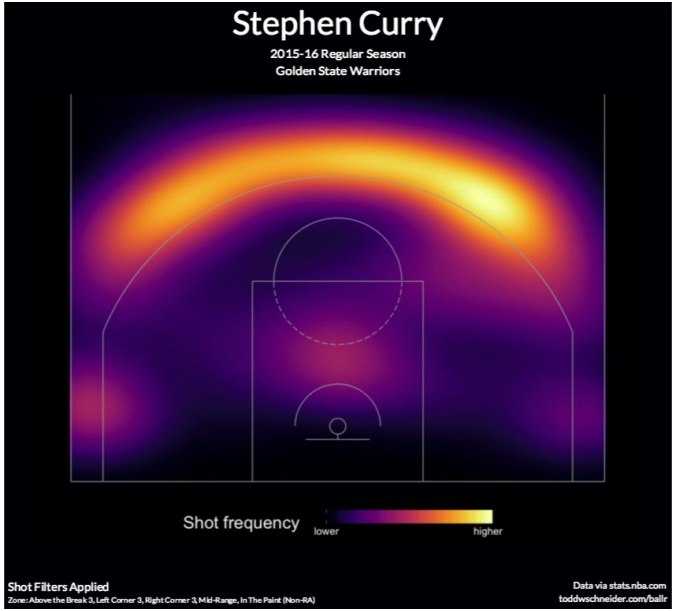


Image Source

Sampling From a Distribution

- Since our goal was to generate new data, we have to sample from the density function.
- Sampling from a multinomial PMF:
  - From the model probabilities $p_1, \ldots, p_k$, compute the cumulative distribution

$$Fi = p_1 + \cdots + p_i \quad \text{for all } i \in \{1, \ldots, k\}$$

  - Draw a uniform random number u ~ [0, 1]
  - Return the smallest i such that $u \leq F_i$
- We have algorithms to sample from a normal distribution given μ and σ
- We have algorithms to sample from uniform and other simple distributions

Source

Generative Models for ‘Simple’ Data

- We are given some data $X = \{x_i\}_{i=1}^{N}$
- We choose a model P(x;w) (that we can sample from later) to model the distribution of X
- We estimate `w` using some method such that P(x;w) best fits the observations X, hoping that it will account for data not in X
- Finally, we sample from P to generate new data such that the new points generated follow the same pattern as X

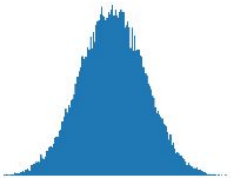
What About High Dimensional Data?

- We are interested in generating high dimensional data like images, audio etc
- We can sample from simple distributions but they do not capture the distribution of high dimensional data well
- Neural networks have the capacity to model the distribution of high dimensional data, but how do we sample from them?
- Can we combine the two above?

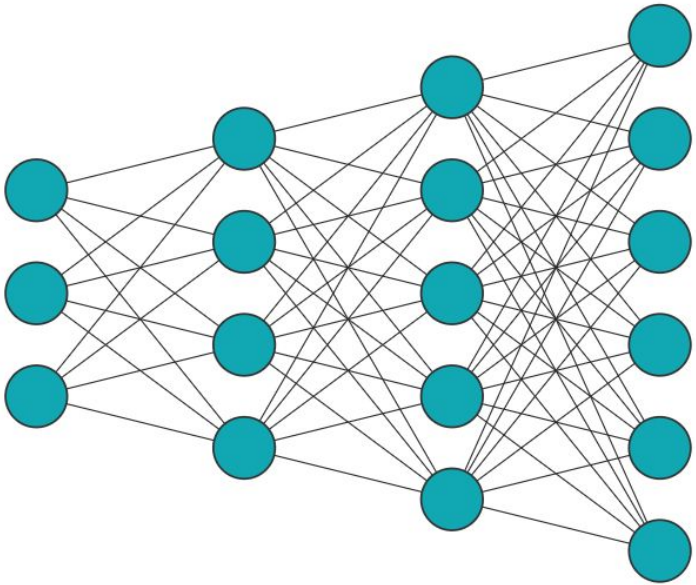
Mapping Distributions

- We create a function `f` such that it maps a simple distribution Z={z} to a high dimensional distribution X={x}. f: z -> x
- `f` would be a neural network which has the capacity to model the distribution of `X`
- When we want to sample new data, we sample a z from Z, provide it as input to f, which gives a data point x, hoping the the generated x is different from the ones in X

Putting it all Together



z~p(z)



f(z; w)

x~p(x)

Learning Network Parameters

- Frameworks to learn the parameters of the generative network `f`:
  - Generative Adversarial Networks (GAN)
  - Variational Autoencoders (VAE)

But First, Some Motivation

Why Study Generative Modeling?

- Create more data
- Manipulate already existing data
- Representation Learning
  - “What I cannot create, I do not understand.” — Richard Feynman

GAN Progress

Link

Generated Face Images

Karras, Tero et al. Analyzing and Improving the Image Quality of StyleGAN. *ArXiv* abs/1912.04958 (2019)

Want More Samples?

Generated Car Images

Karras, Tero et al. Analyzing and Improving the Image Quality of StyleGAN. *ArXiv* abs/1912.04958 (2019)

Generated Cat and Church Images

Karras, Tero et al. Analyzing and Improving the Image Quality of StyleGAN. *ArXiv* abs/1912.04958 (2019)

It's raining cats and cats

Super-Resolution

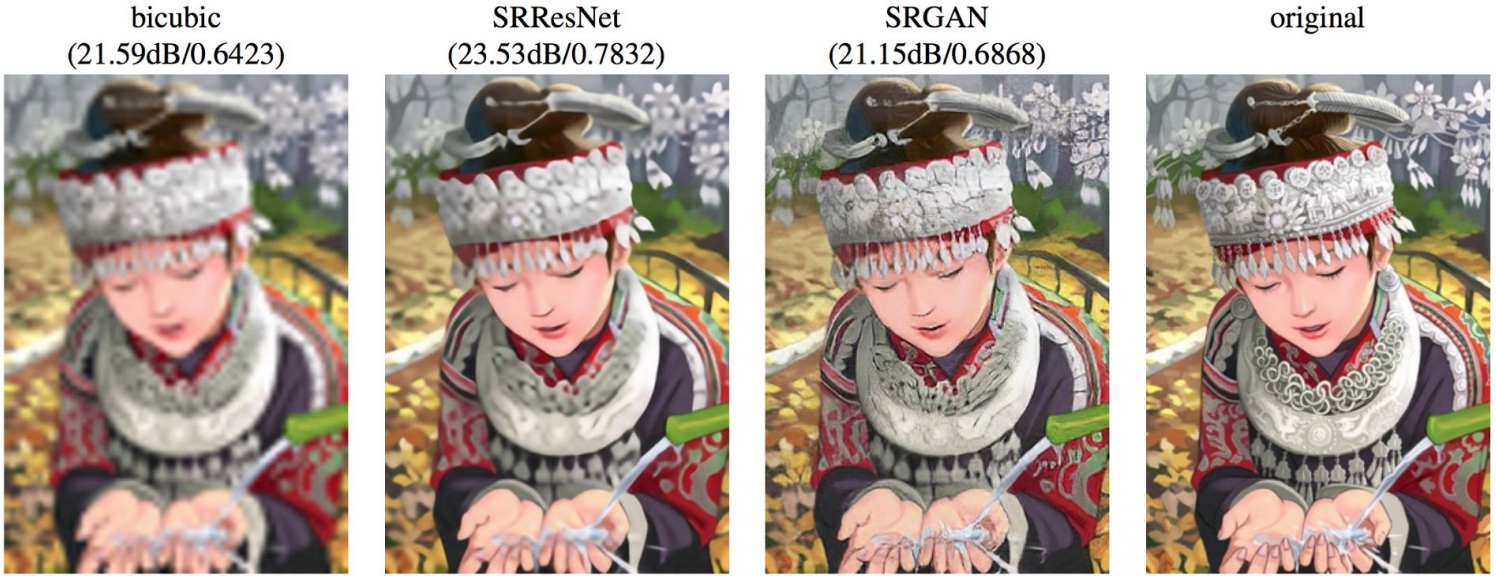
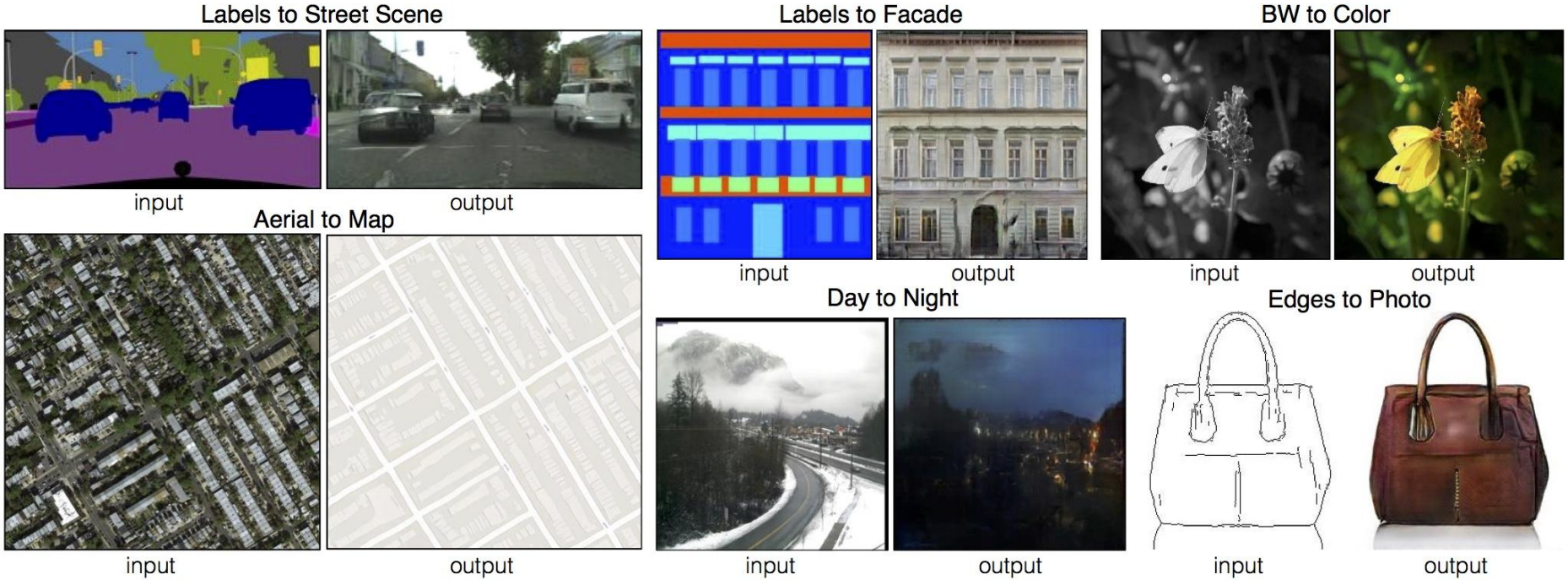


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

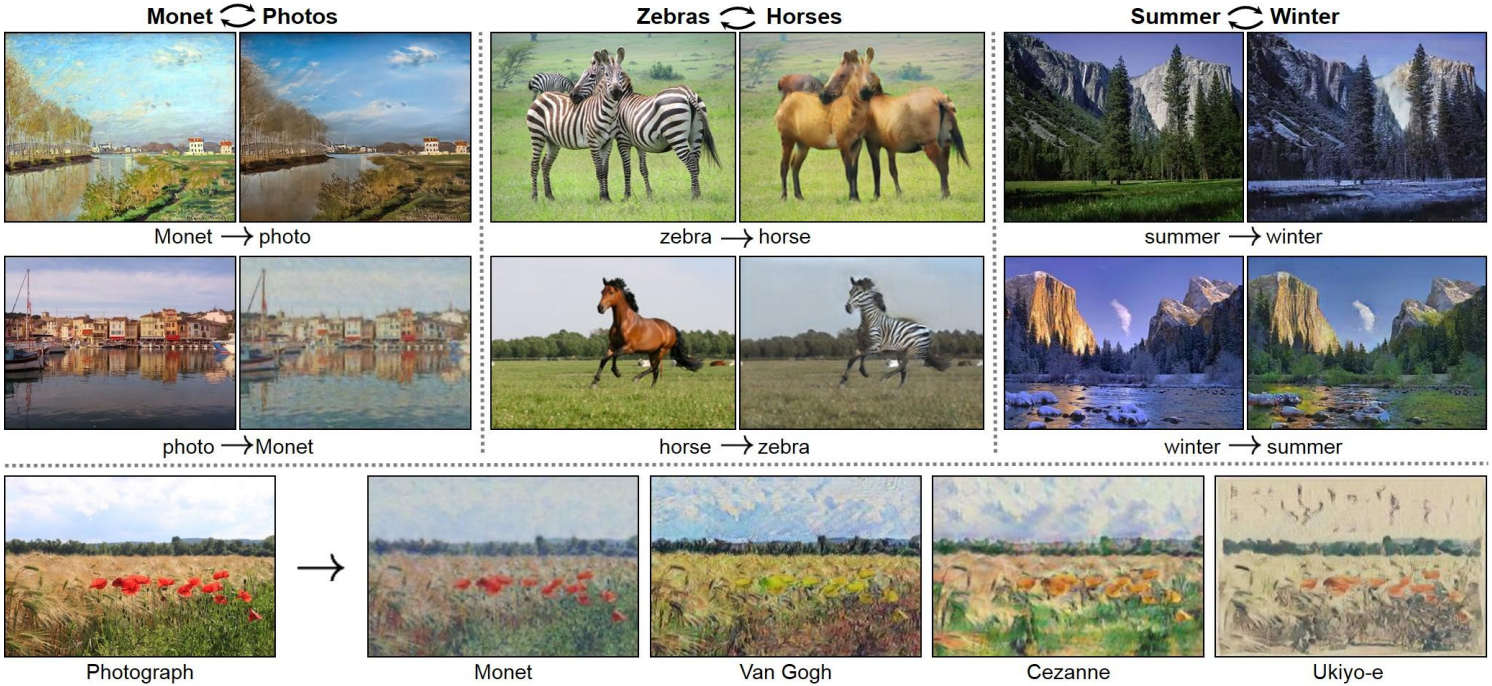
Ledig, Christian et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *CVPR* (2017)

Image-to-Image Translation



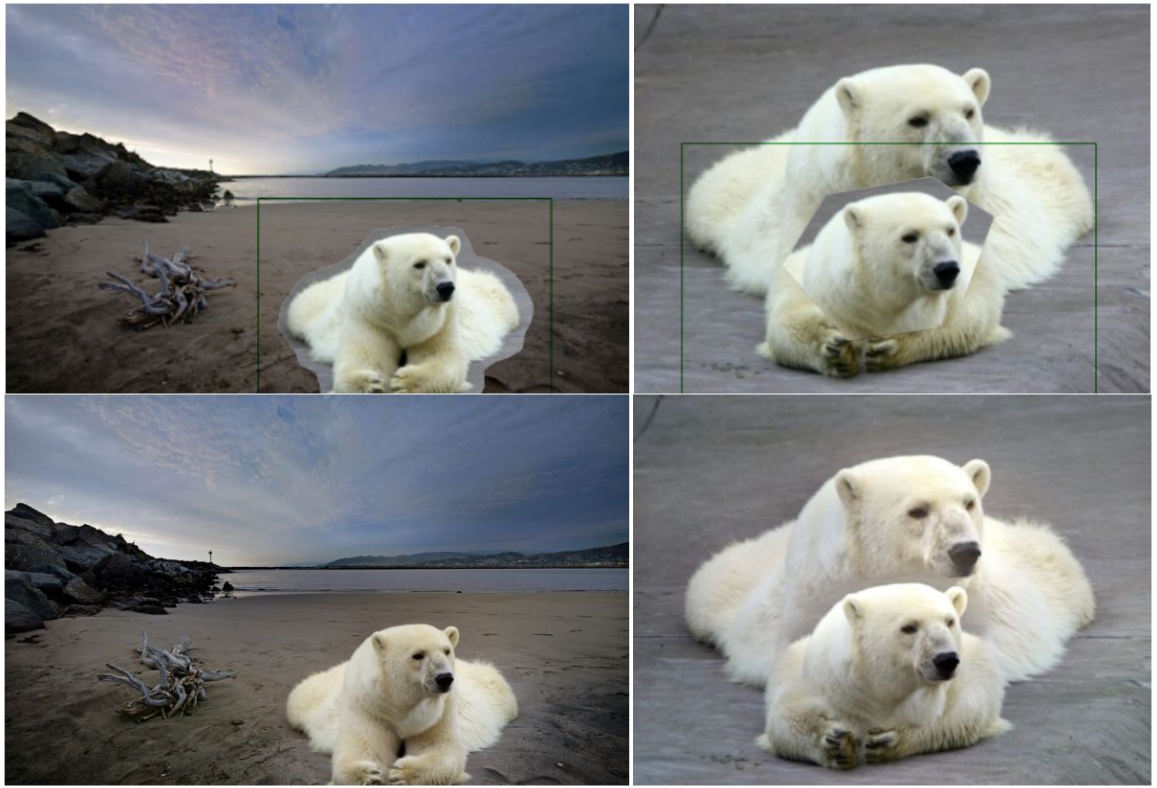
Isola, P., Zhu, J., Zhou, T., & Efros, A.A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. *CVPR (2017)*

Unpaired Image-to-Image Translation



Zhu, J., Park, T., Isola, P., & Efros, A.A. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *ICCV (2017)*

Image Blending



Wu, Huikai et al. GP-GAN: Towards Realistic High-Resolution Image Blending. *International Conference on Multimedia* (2019)

Video Generation

Clark, A., Donahue, J., & Simonyan, K. (2019). Efficient Video Generation on Complex Datasets. *ArXiv, abs/1907.06571*.

Generative Adversarial Networks

Why Study Generative Modeling?

- Create more data
- Manipulate already existing data
- Representation Learning
  - “What I cannot create, I do not understand.” — Richard Feynman

Generative Adversarial Networks

- Goodfellow et al, 2014
- Implicit density models: probability density function is not explicitly defined and solved for
- Goal is to model the data distribution
- Trained using a pair of networks playing a minmax game
- They are termed as “adversaries” because they have conflicting loss functions
- New data are sampled by “seeding” the trained generator

GAN Architecture

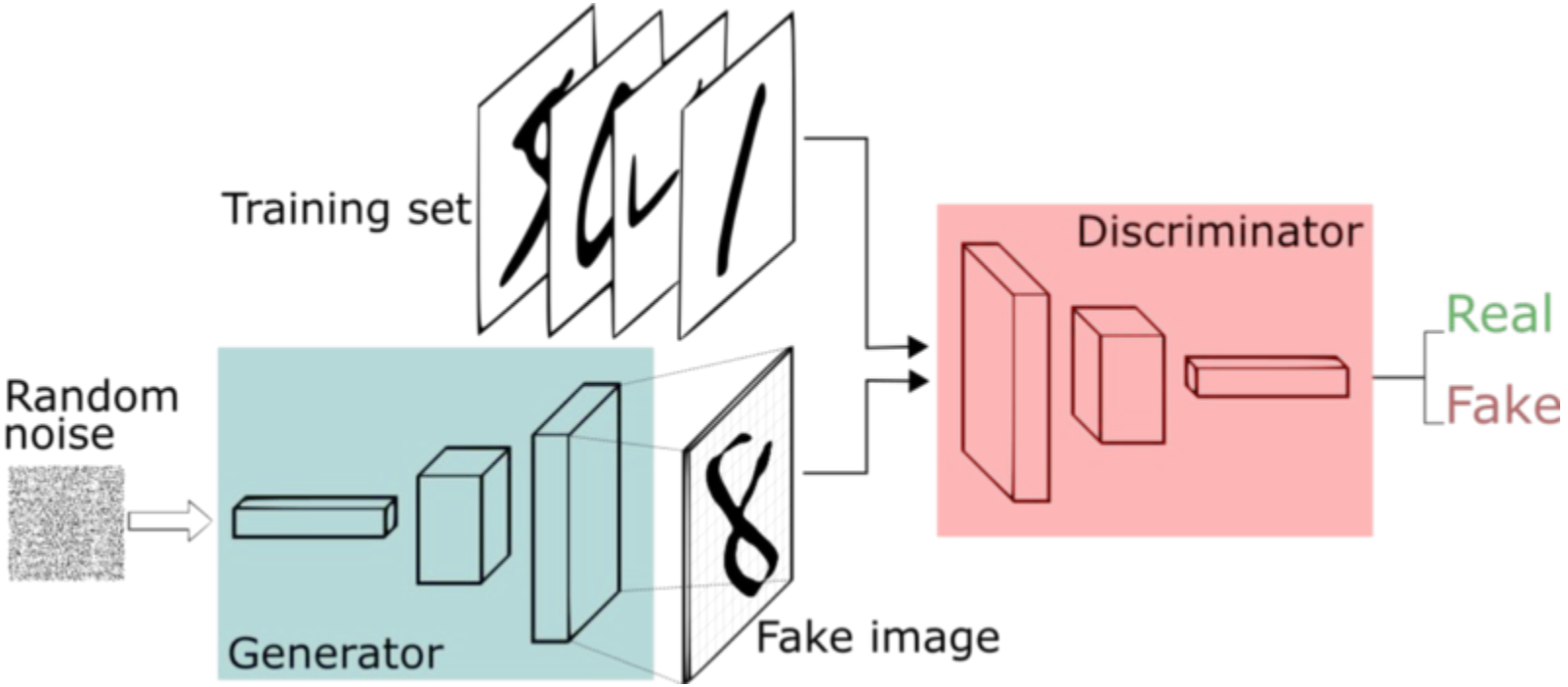


Image Source

Vanilla GAN Loss

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

- Minmax game being played between a generator (G) and a discriminator (D)
- D tries to maximize the log-likelihood for the binary classification problem of real (1) vs fake (0)
- G tries to minimize the log-probability of its samples being classified as fake (0) by D (“fooling” the discriminator)

Goodfellow, Ian J. et al. Generative Adversarial Nets. *NIPS* (2014).

Vanilla GAN Training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

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:

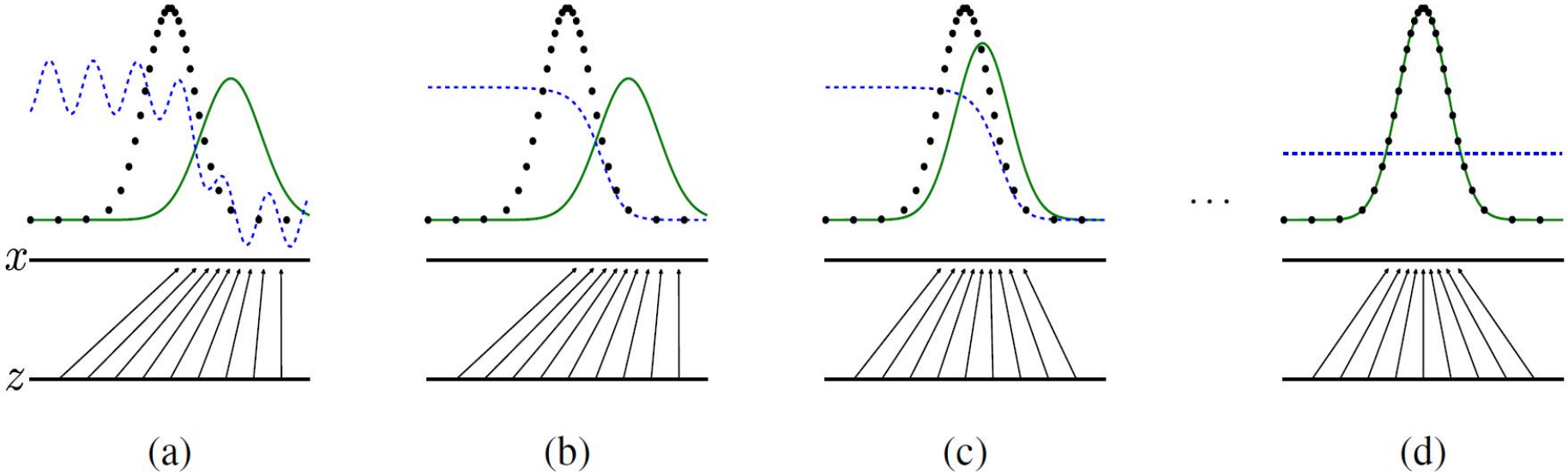
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Goodfellow, Ian J. et al. Generative Adversarial Nets. *NIPS* (2014).

Training Progression



Goodfellow, Ian J. et al. Generative Adversarial Nets. *NIPS* (2014).

Before We Start Training

In practice, equation 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(\boldsymbol{z})))$ saturates. Rather than training G to minimize $\log(1 - D(G(\boldsymbol{z})))$ we can train G to maximize $\log D(G(\boldsymbol{z}))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

- How to do it: flip labels when training generator: real (0), fake (1)

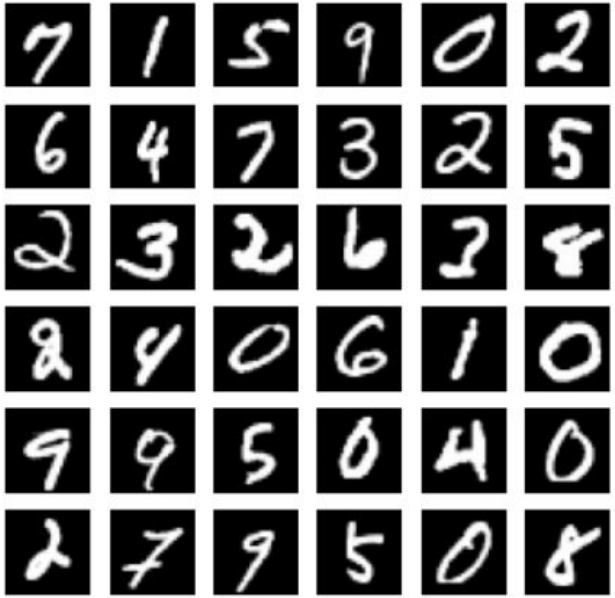
Goodfellow, Ian J. et al. Generative Adversarial Nets. *NIPS* (2014).

Show me the Code

Loading MNIST

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype(np.float32) / 255.0
x_train = 2 * (x_train - 0.5)
print('Max value:', x_train.max())
print('Min value:', x_train.min())
print('Loaded dataset tensor size:', x_train.shape)
dataSize = x_train.shape[0]
dimSize = x_train.shape[1] * x_train.shape[2]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/t
11493376/11490434 [==============================] - 0s 0us/step
Max value: 1.0
Min value: -1.0
Loaded dataset tensor size: (60000, 28, 28)
```



This is a TF2 port of Minimal MNIST GAN originally written in torch7 by Prof Arjun Jain. Slide inspired by Deep Learning (for Computer Vision)

Define Networks

Define Generator Network

```
n_hidden = 128
z_dim = 100

# Generator
model_g = Sequential()
model_g.add(Dense(n_hidden, input_shape=(z_dim,), kernel_initializer='he_uniform'))
model_g.add(LeakyReLU(0.01))
model_g.add(Dense(dimSize, kernel_initializer='he_uniform'))
model_g.add(Activation('tanh'))
```

- LeakyReLUs facilitate better gradient flow by avoiding sparse gradients
- He Uniform Initialization: Default (Glorot Uniform) lead to failed training
- Tanh generates a normalized image, `dimSize` is 28*28

This is a TF2 port of Minimal MNIST GAN originally written in torch7 by Prof Arjun Jain. Slide inspired by Deep Learning (for Computer Vision)

Define Networks

Define Discriminator Network

```
model_d = Sequential()
model_d.add(Dense(n_hidden, input_shape=(dimSize,), kernel_initializer='he_uniform'))
model_d.add(LeakyReLU(0.01))
model_d.add(Dense(2, kernel_initializer='he_uniform'))
```

- LeakyReLU and He Uniform due to the same reasons as before
- Discriminator is a binary classifier
- Adding a Softmax leads to failed training

This is a TF2 port of Minimal MNIST GAN originally written in torch7 by Prof Arjun Jain. Slide inspired by Deep Learning (for Computer Vision)

Setting up the Stage

```
# Hyperparameters
batchSize = 32
batchSizeHalf = batchSize / 2

# Total batches that can be created from the training set
nBatches = math.floor(dataSize / batchSizeHalf)
```

Hyperparameters

```
def get_real_batch(batchData, batchId, shuffle, batchSize):
    start_idx = int(batchId * batchSize)
    end_idx = int((batchId + 1) * batchSize)
    indices = shuffle[start_idx:end_idx]
    num_samples = indices.shape[0]
    return batchData[indices].reshape(num_samples, dimSize)
```

```
def get_fake_batch(generator, latent_dim, batchSize):
    z = np.random.rand(batchSize, latent_dim) - 0.5
    fakes = generator.predict(z)
    return fakes
```

Helper functions to sample data

```
optim_d = Adagrad(0.1)
optim_g = Adagrad(0.1)
criterion = SparseCategoricalCrossentropy(from_logits=True)
maxEpoch = 20
```

Optimizers and Loss function

This is a TF2 port of Minimal MNIST GAN originally written in torch7 by Prof Arjun Jain. Slide inspired by Deep Learning (for Computer Vision)

Let Training Begin!

Real

Fake

Update Discriminator Weights

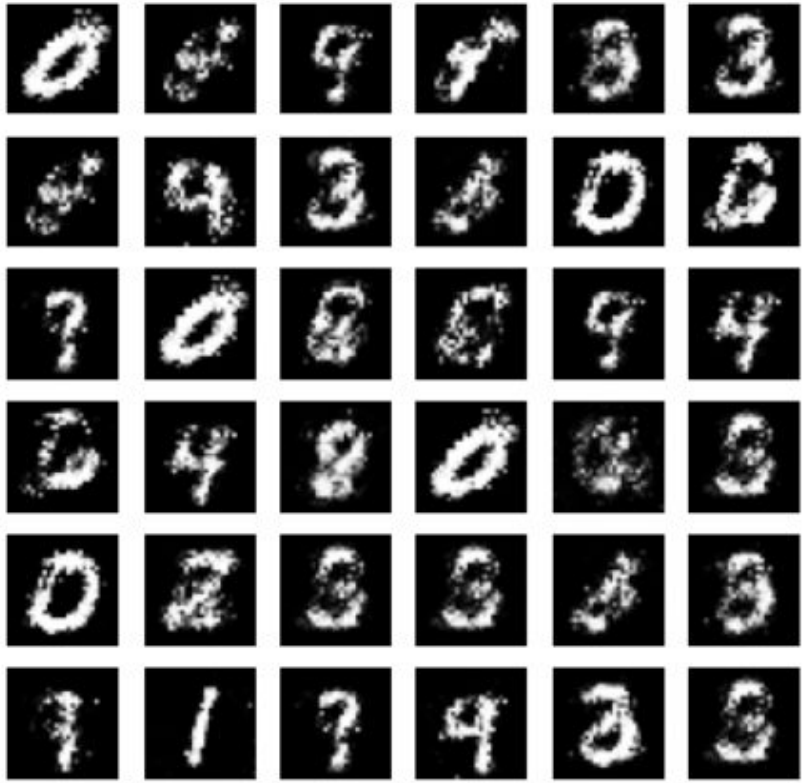
Real (flipped labels)

Update Generator Weights

```
shuffle = np.arange(dataSize)

for epochId in range(maxEpoch):
    np.random.shuffle(shuffle)

    for batch_id in tqdm(range(nBatches)):
        # Train the discriminator
        x_real = get_real_batch(x_train, batch_id, shuffle, batchSizeHalf)
        y_real = np.ones((x_real.shape[0], 1))

        x_fake = get_fake_batch(model_g, z_dim, x_real.shape[0])
        y_fake = np.zeros((x_real.shape[0], 1))

        X, Y = np.vstack((x_real, x_fake)), np.vstack((y_real, y_fake))
        with tf.GradientTape() as tape:
            predictions = model_d(X)
            d_loss = criterion(Y, predictions)
        grads = tape.gradient(d_loss, model_d.trainable_weights)
        optim_d.apply_gradients(zip(grads, model_d.trainable_weights))

        # Train the generator
        z = np.random.rand(batchSize, z_dim) - 0.5
        labels = np.ones((batchSize, 1))

        with tf.GradientTape() as tape:
            predictions = model_d(model_g(z))
            g_loss = criterion(labels, predictions)
        grads = tape.gradient(g_loss, model_g.trainable_weights)
        optim_g.apply_gradients(zip(grads, model_g.trainable_weights))
```

This is a TF2 port of Minimal MNIST GAN originally written in torch7 by Prof Arjun Jain. Slide inspired by Deep Learning (for Computer Vision)

After Training for 20 Epochs



Source

Problems with Training Vanilla GANs

- Non-convergence
- A balancing act
  - If the discriminator gets too strong too fast, generator does not learn
  - If the discriminator is weak generator does not receive reliable feedback
- Too sensitive to the choice of hyperparameters. Require “hacks” to behave well
- Mode Collapse: generated samples lack diversity

Non-Convergence

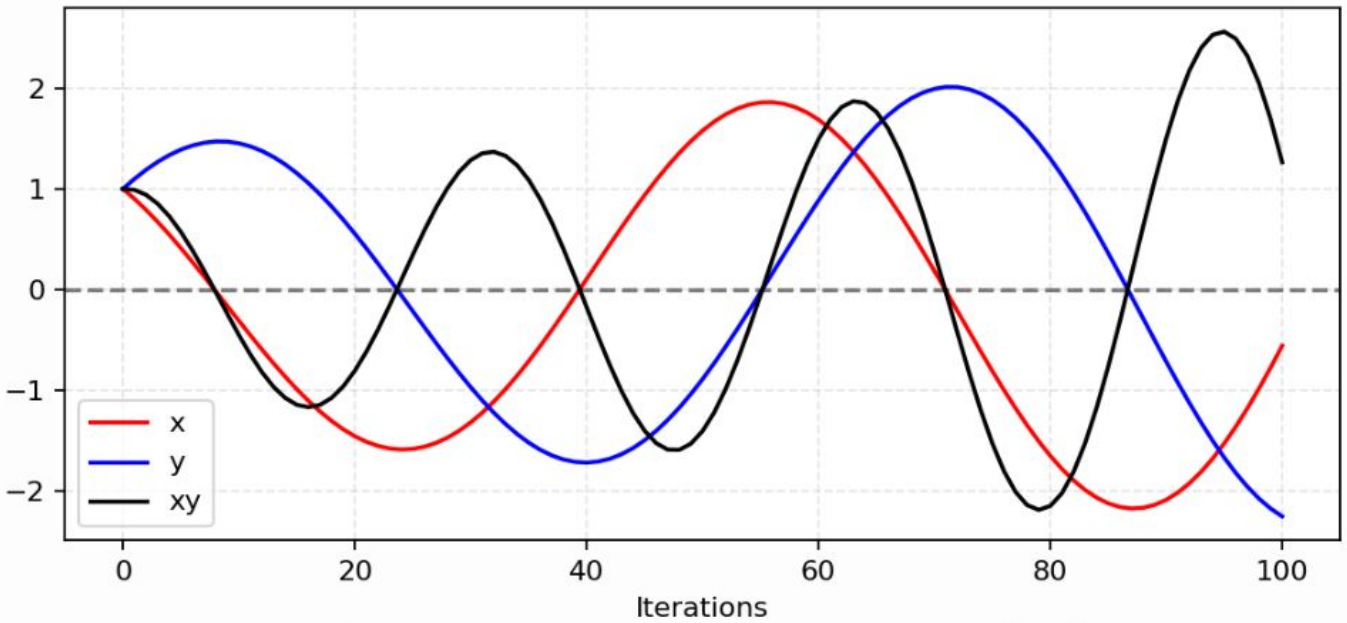
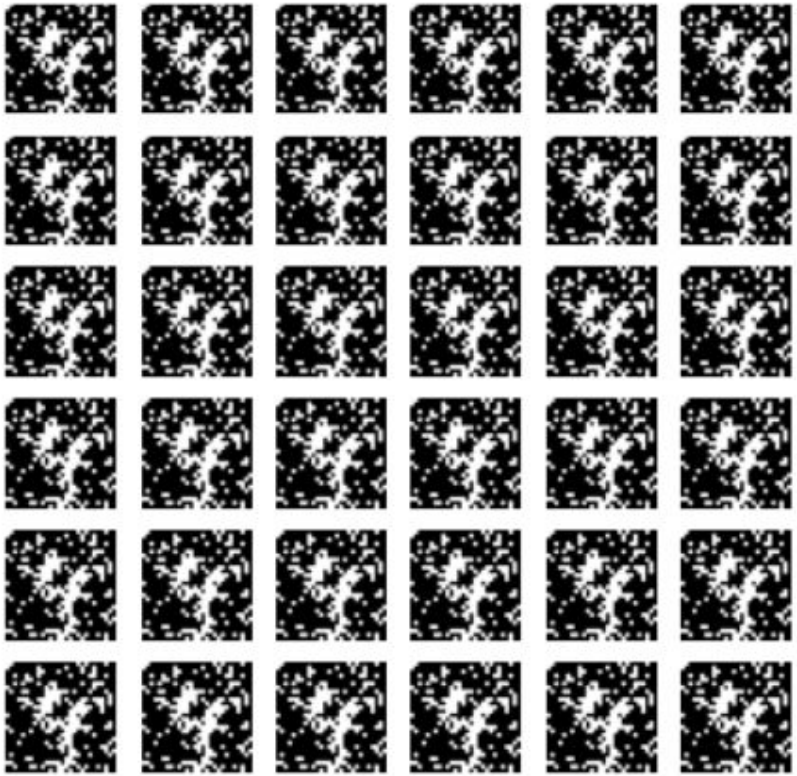


Fig. 3. A simulation of our example for updating x to minimize xy and updating y to minimize $-xy$. The learning rate $\eta = 0.1$. With more iterations, the oscillation grows more and more unstable.

Weng, Lilian. From GAN to WGAN. *ArXiv* abs/1904.08994 (2019)

Mode Collapse

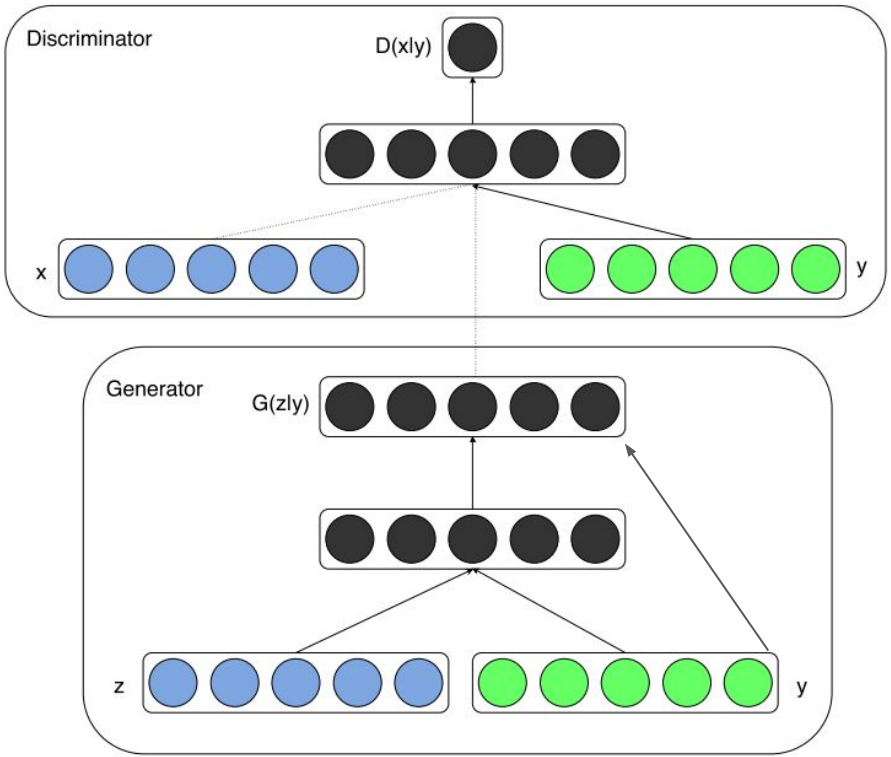


Source

Conditional GANs

- In an unconditional GAN, there is no control over the characteristics of the data being generated
- A cGAN generates fake samples that satisfy some condition (a class label, base image, tags, text description, etc), rather than generating generic samples

Vanilla cGANs



Mirza, Mehdi and Simon Osindero. Conditional Generative Adversarial Nets. *ArXiv* abs/1411.1784 (2014)

BigGAN

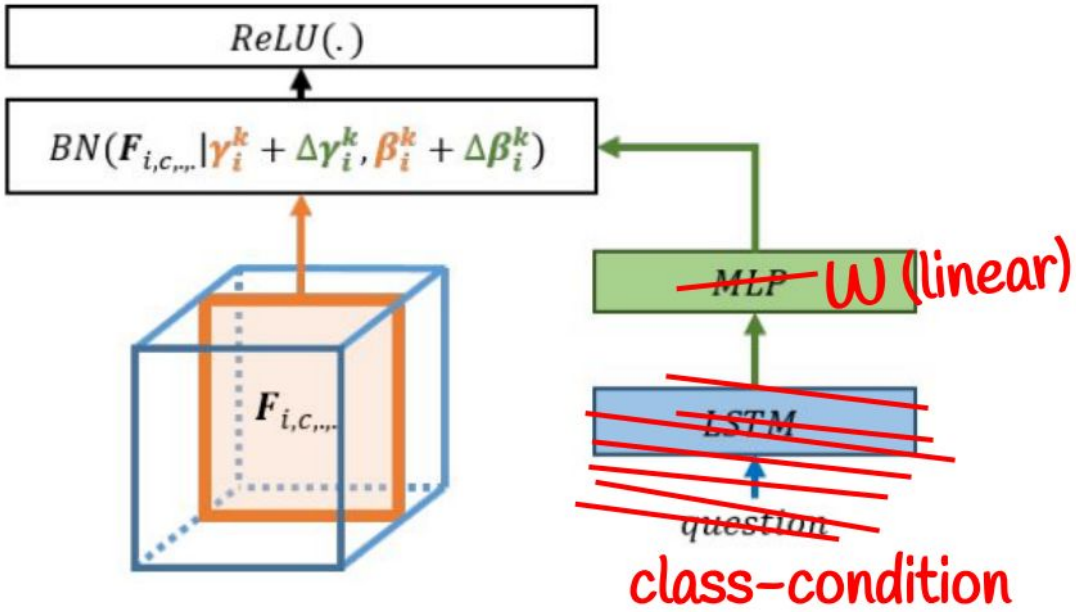


Image Source

- Among other things, use class-conditional batchnorm layers

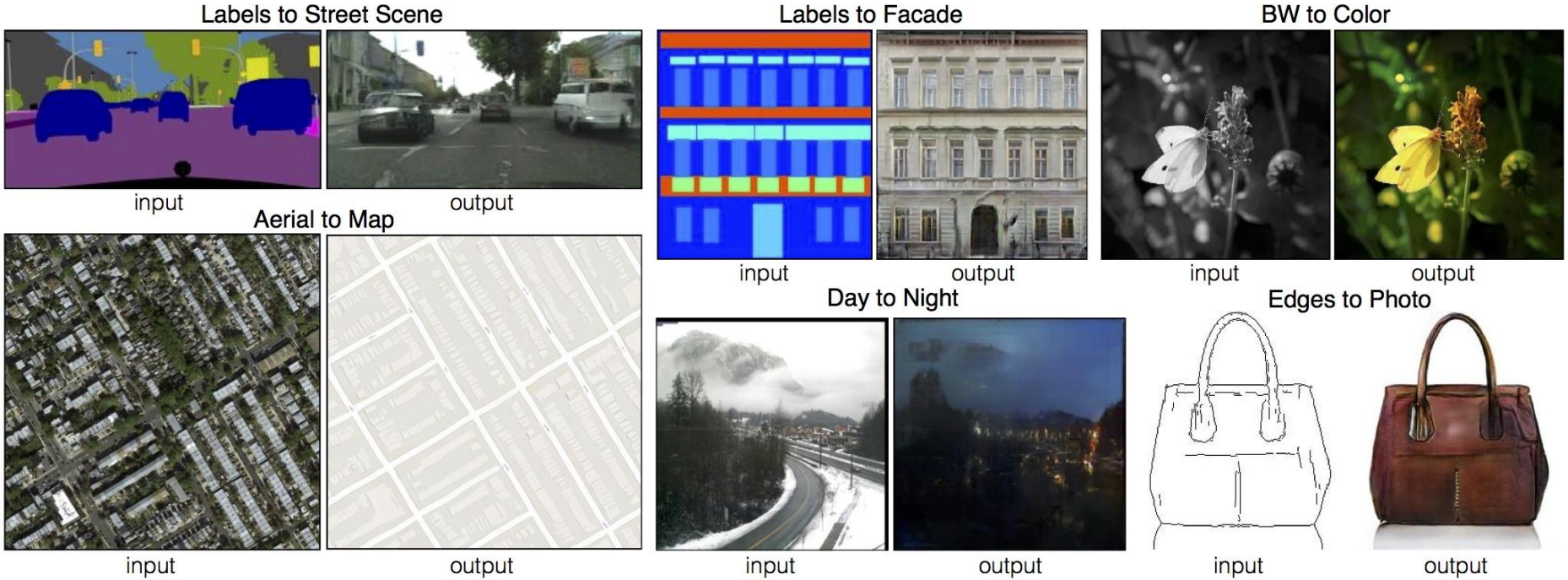
BigGAN

Brock, Andrew et al. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *ArXiv* abs/1809.11096 (2018)

Why Study Generative Modeling?

- Create more data
- Manipulate already existing data
- Representation Learning
  - “What I cannot create, I do not understand.” — Richard Feynman

Paired Image-to-Image Translation



Isola, P., Zhu, J., Zhou, T., & Efros, A.A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. *CVPR (2017)*

Paired Image-to-Image Translation

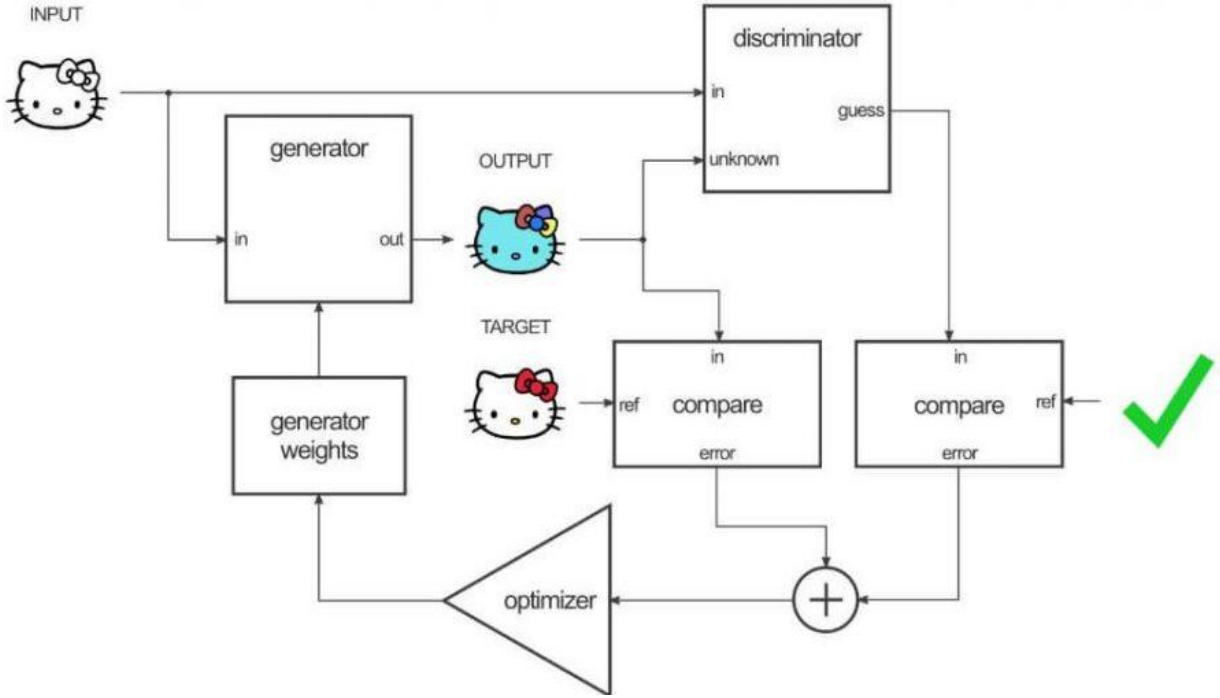
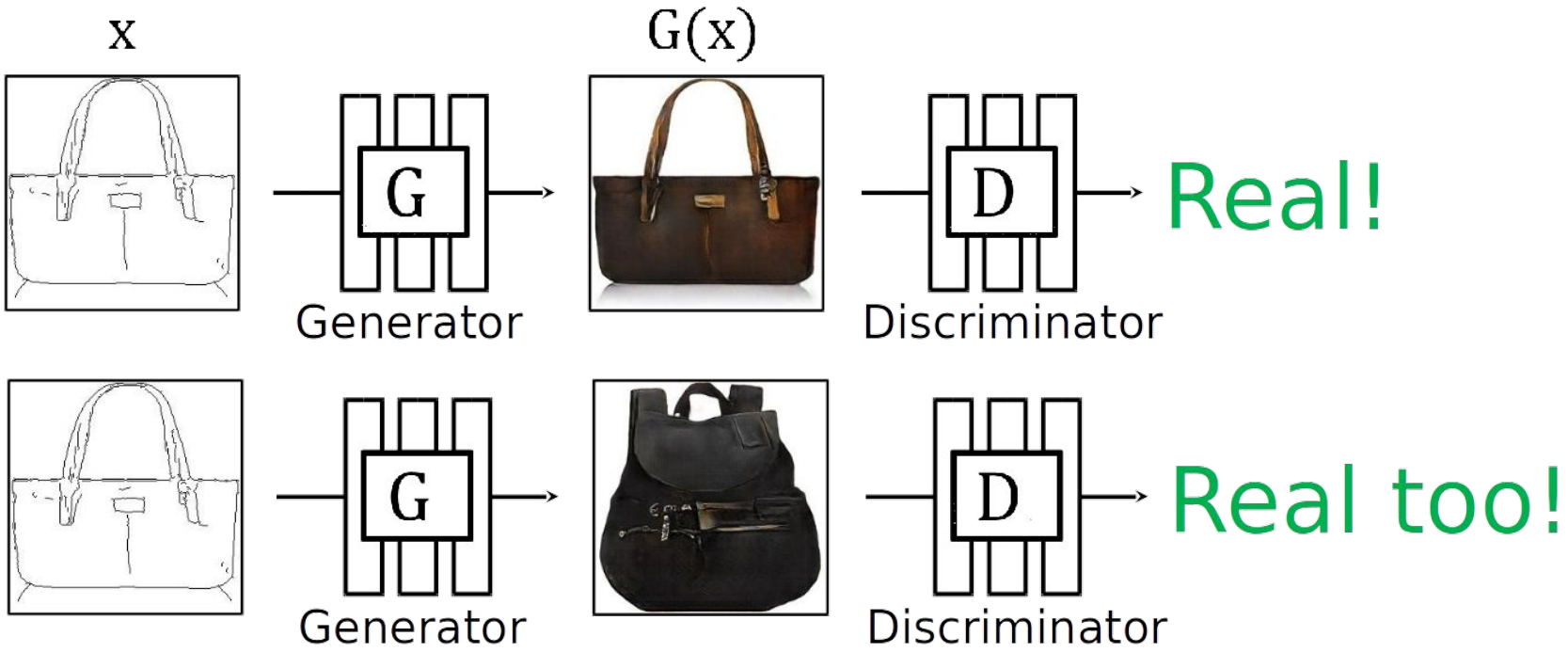


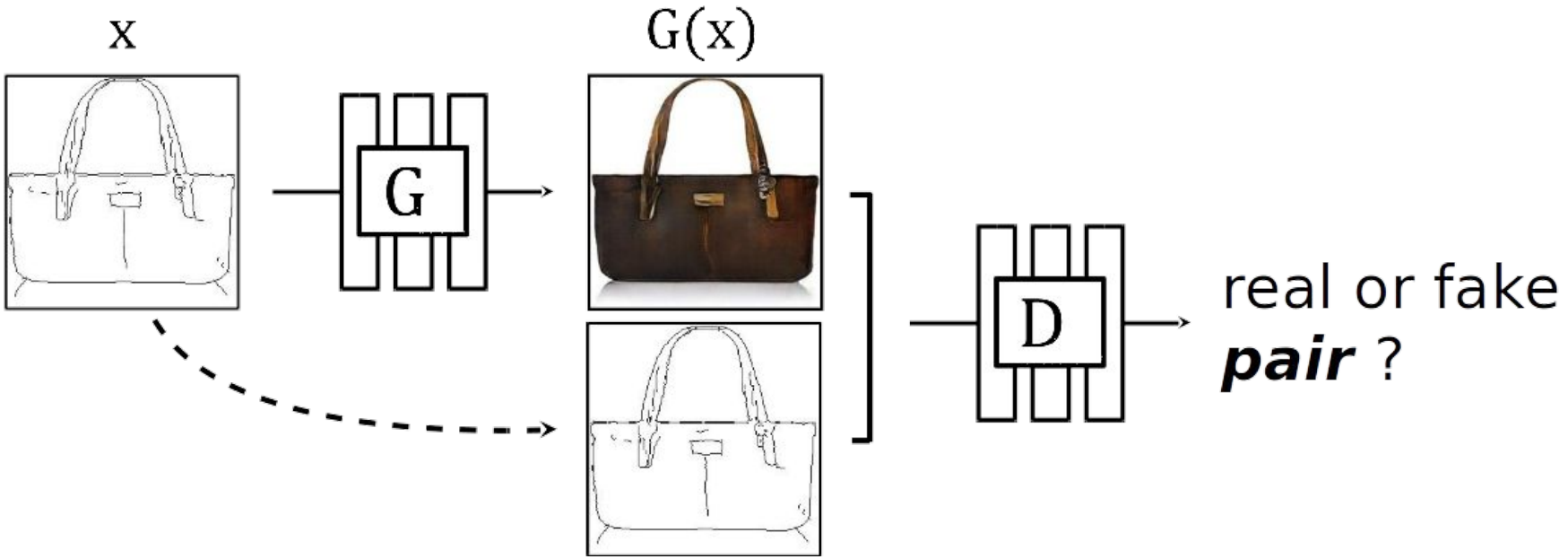
Image Source

Why cGAN?



Credits: Jun-Yan Zhu

Why cGAN?



Credits: Jun-Yan Zhu

Effect of Different Losses

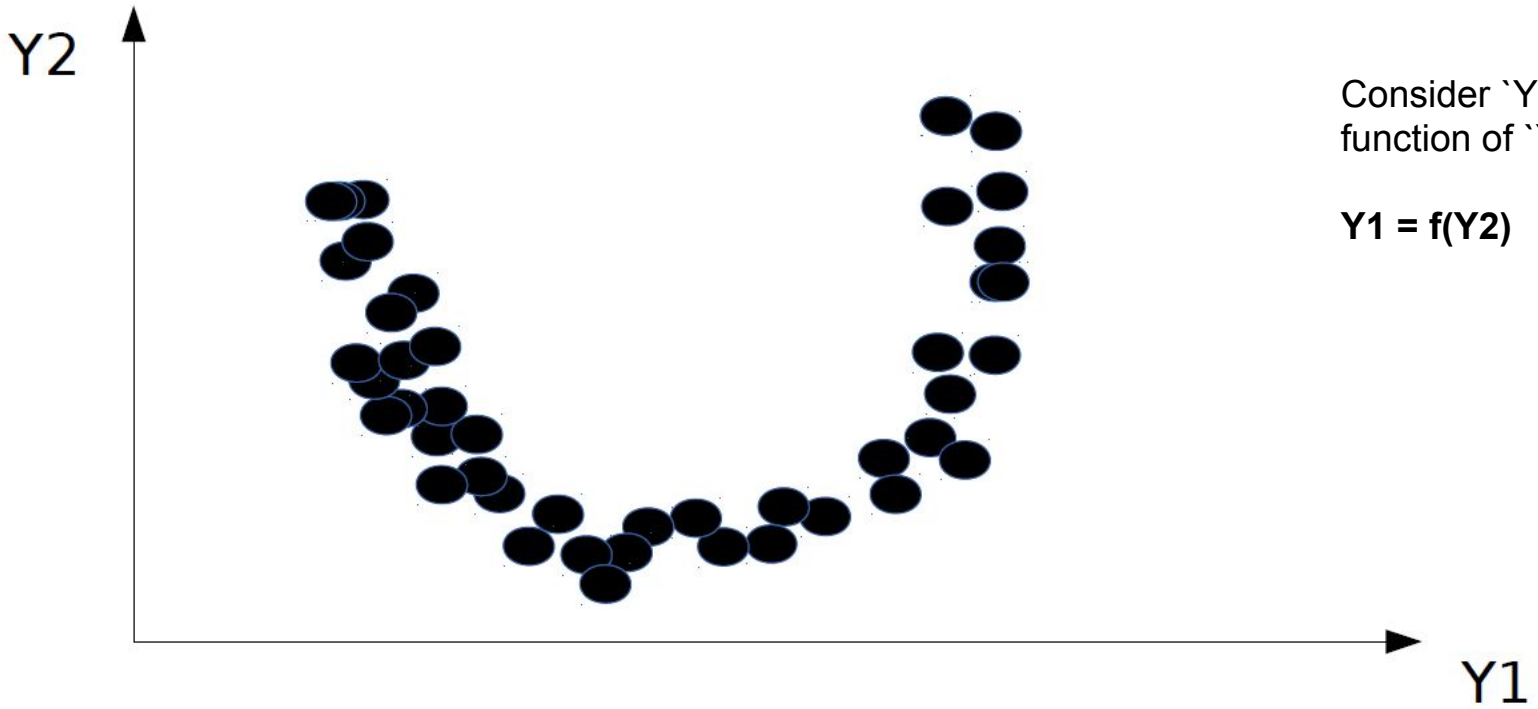
Isola, P., Zhu, J., Zhou, T., & Efros, A.A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. *CVPR (2017)*

Notion of Adversarial Loss

Understanding Blurry Results Due to L1/L2

Image Source

Understanding Blurry Results Due to L1/L2

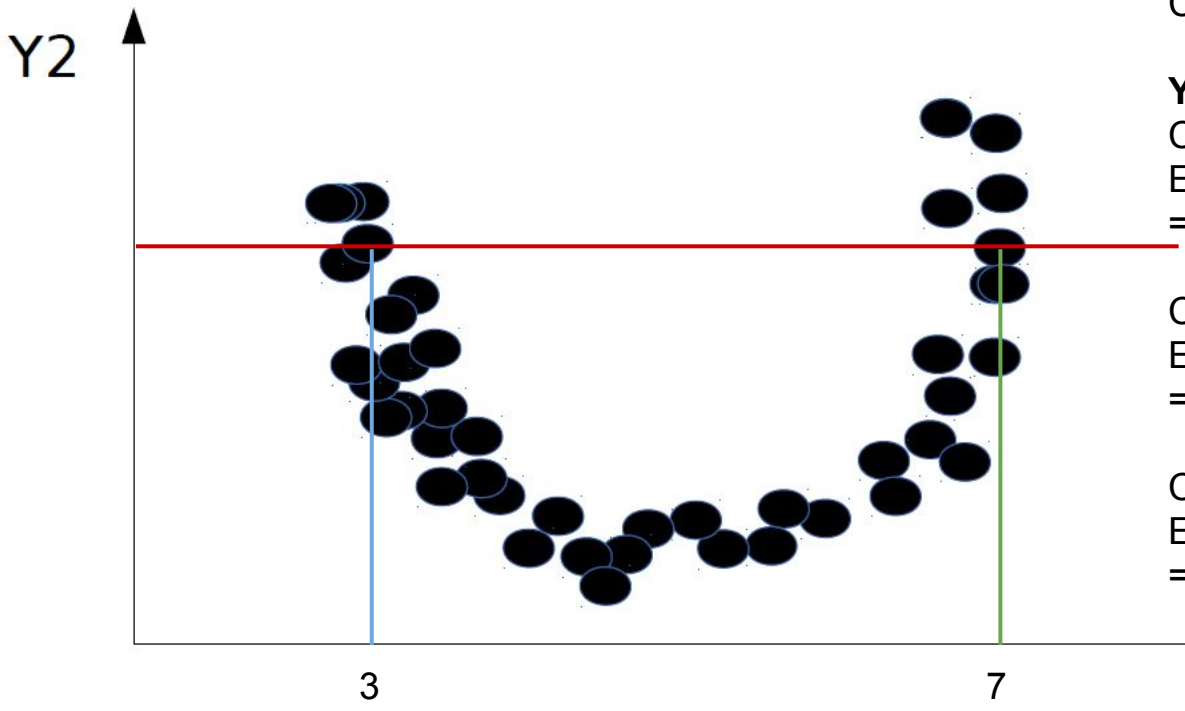


Consider `Y1` as a function of `Y2`

Y1 = f(Y2)

Image Credits: Prof Yann LeCun

Understanding Blurry Results Due to L1/L2



Consider `Y1` as a function of `Y2`

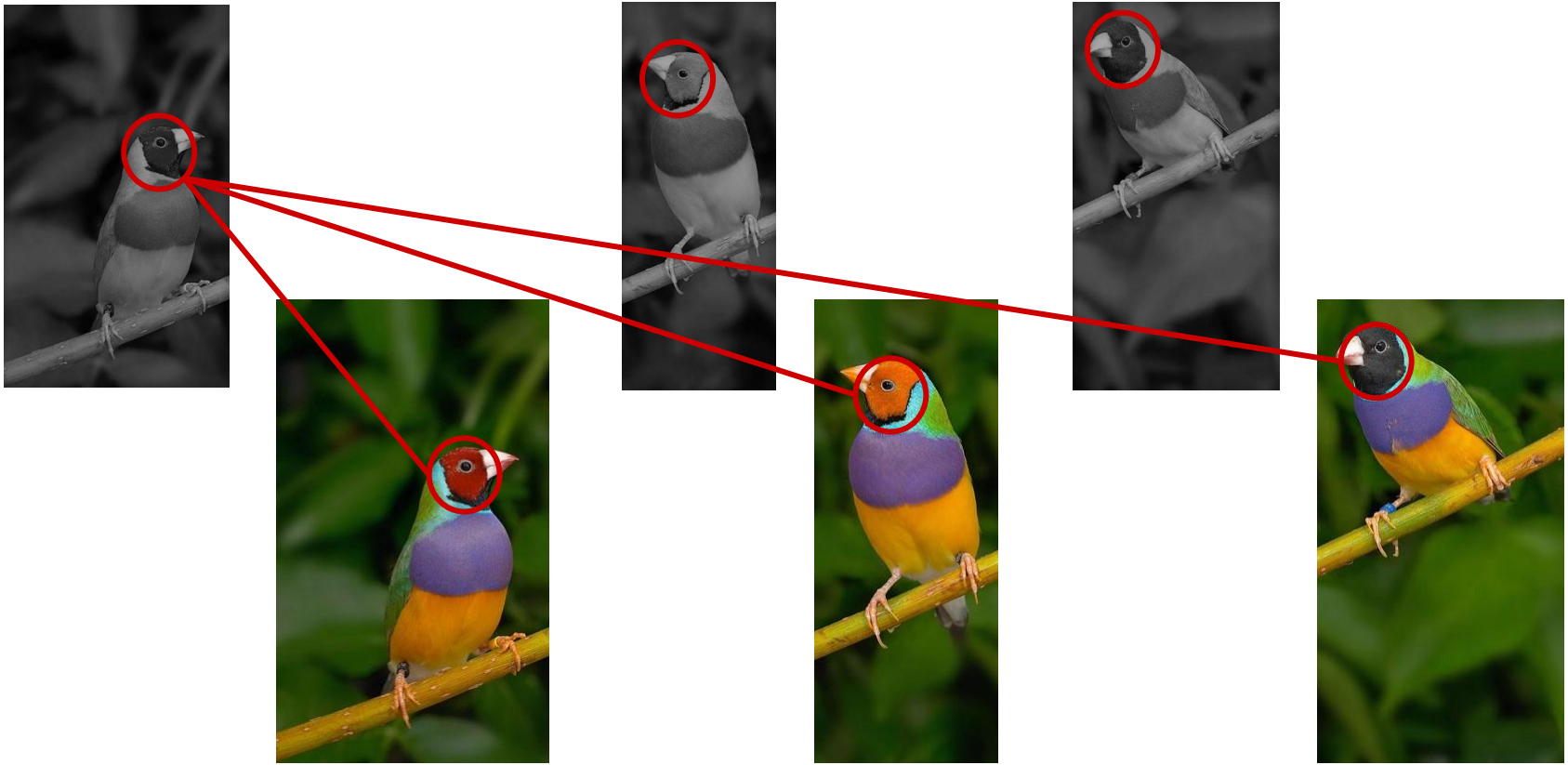
Y1 = f(Y2)
Choose blue
Error = 0.5 * (error(b, b) + error(b, g))
= 0.5 * (0 + 16) = 8

Choose green
Error = 0.5 * (error(g, b) + error(g, g))
= 0.5 * (16 + 0) = 8

Choose Mid-point
Error = 0.5 * ((error(m, b) + error(m, g))
= 0.5 * (4 + 4) = 4

Image Credits: Prof Yann LeCun

Understanding Blurry Results Due to L1/L2



SRGAN

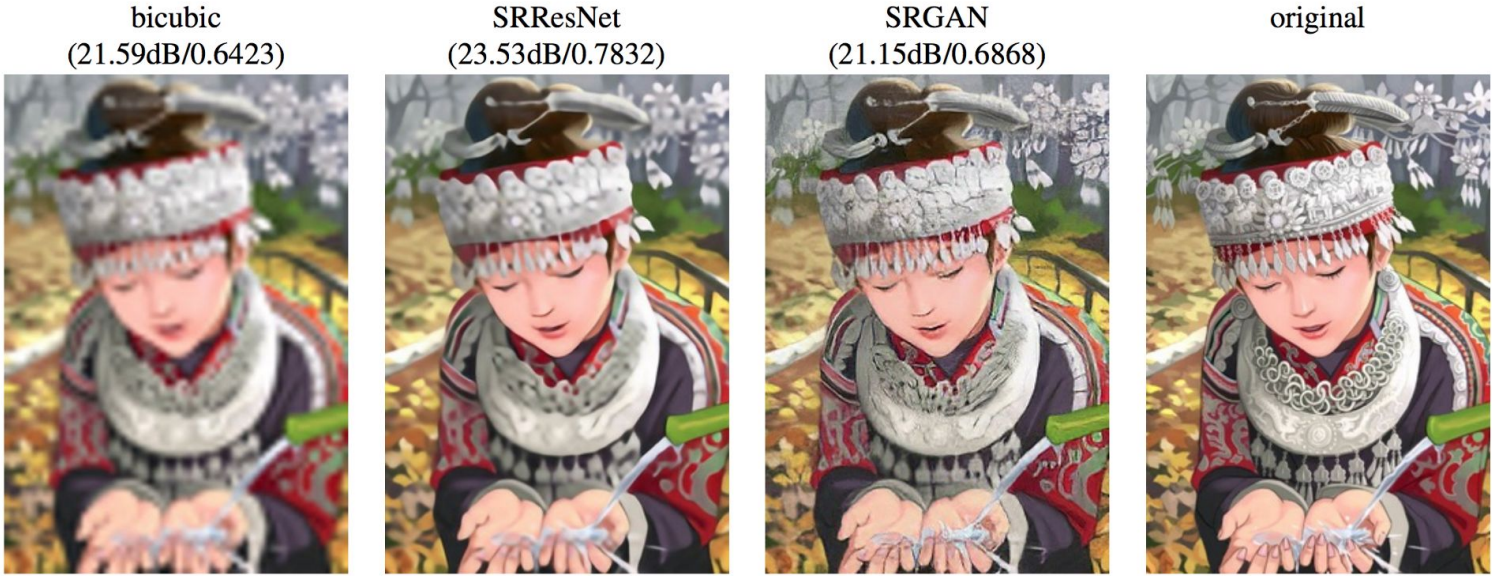


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Ledig, Christian et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *CVPR* (2017)

SRGAN Architecture

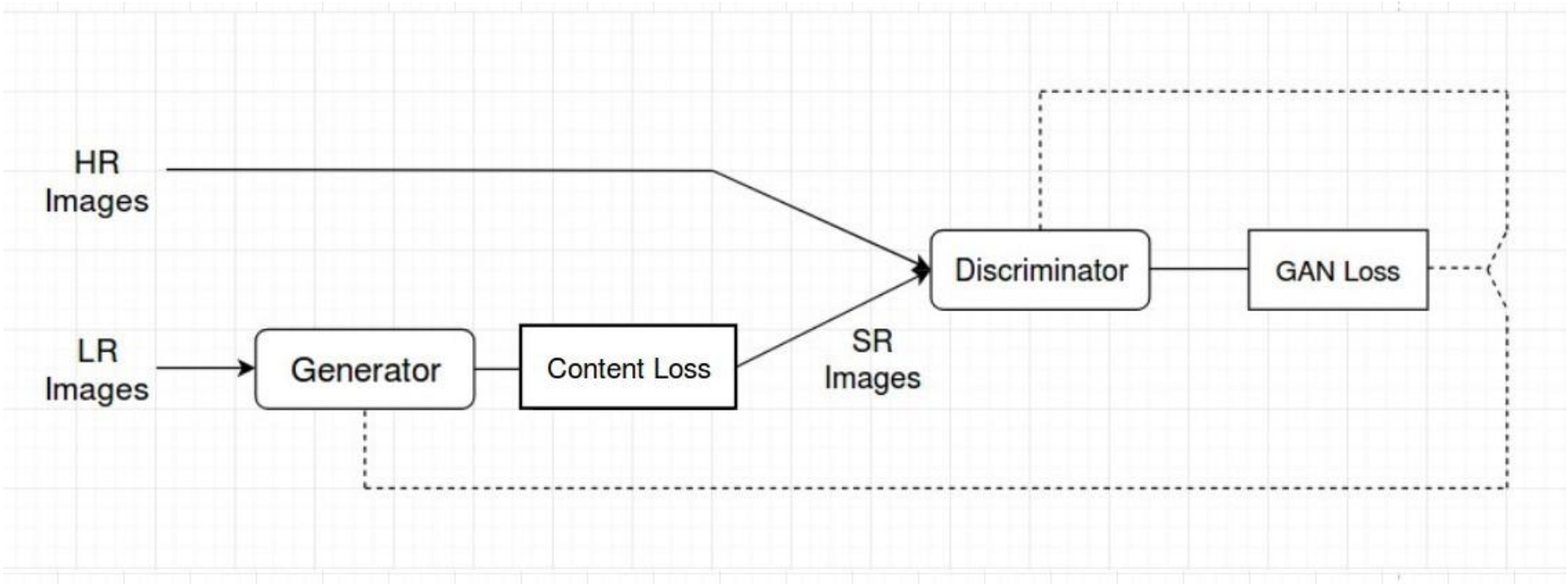


Image Source

Paired Image-to-Image Translation

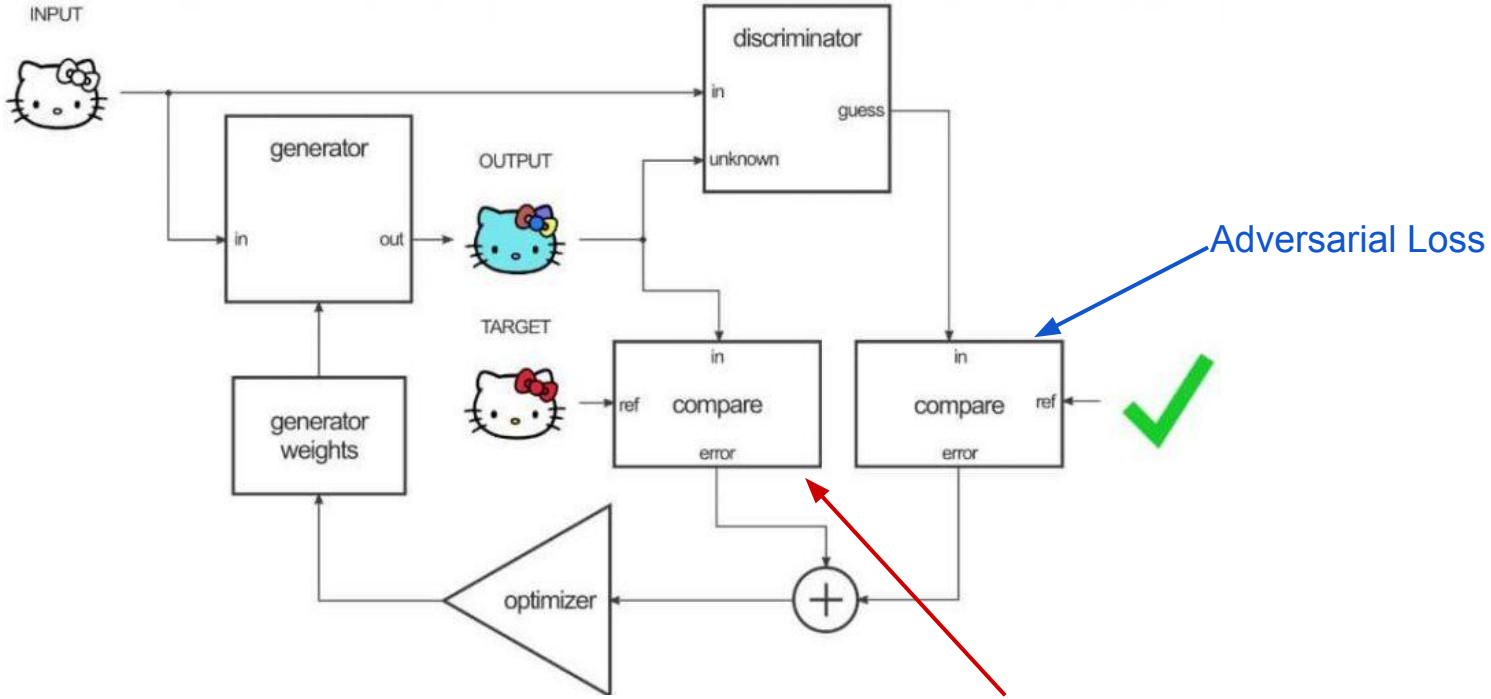
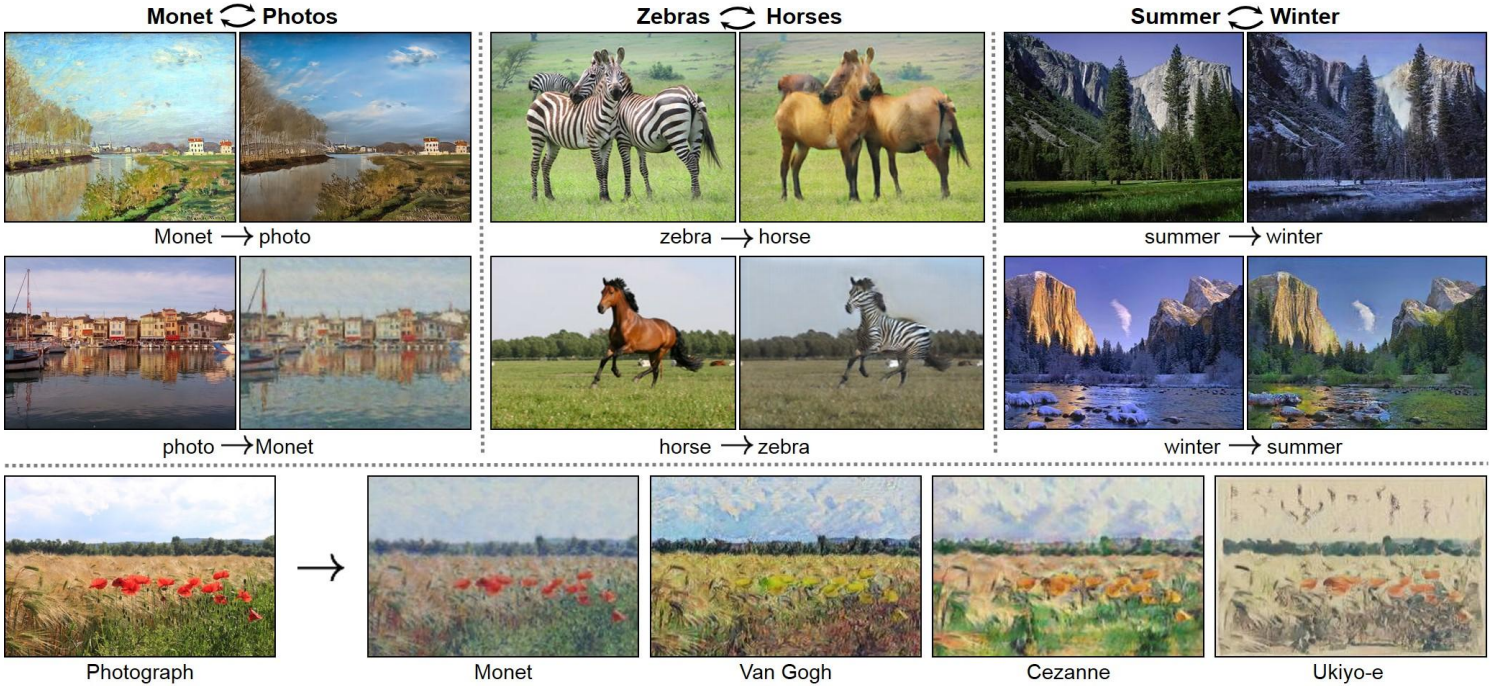


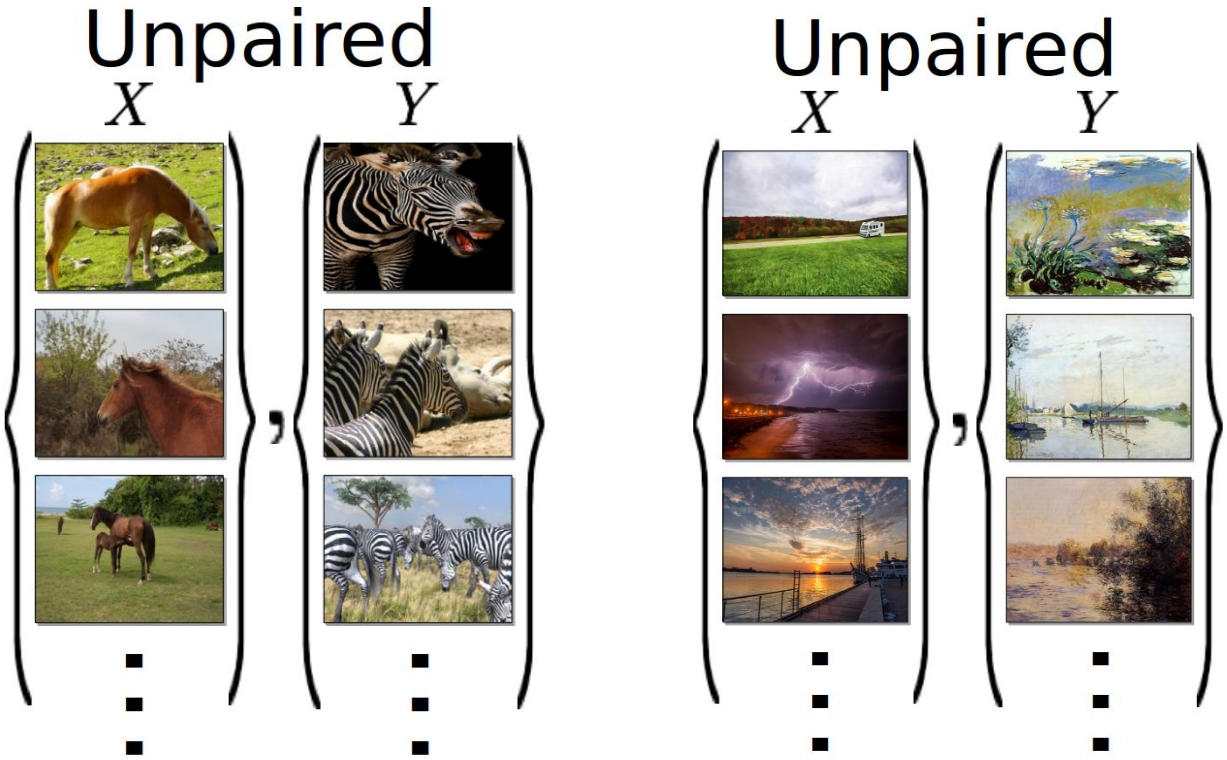
Image Source

Unpaired Image-to-Image Translation



Zhu, J., Park, T., Isola, P., & Efros, A.A. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *ICCV (2017)*

Snapshot of Data

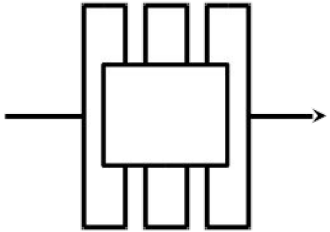


Credits: Jun-Yan Zhu

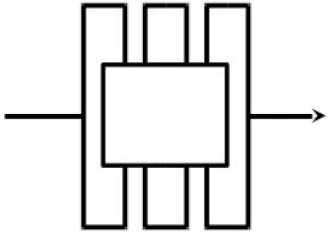
What About This Method?

- Train the generator to generate something from horses
- Sample a generated batch, and a batch of zebras
- Ask the discriminator to differentiate between them
- Train like a normal GAN

Problem

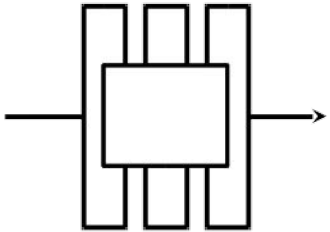


Generator

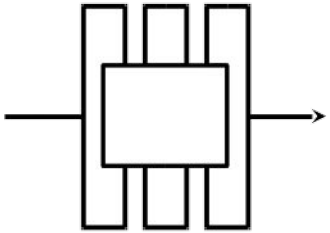


Discriminator

Real!



Generator

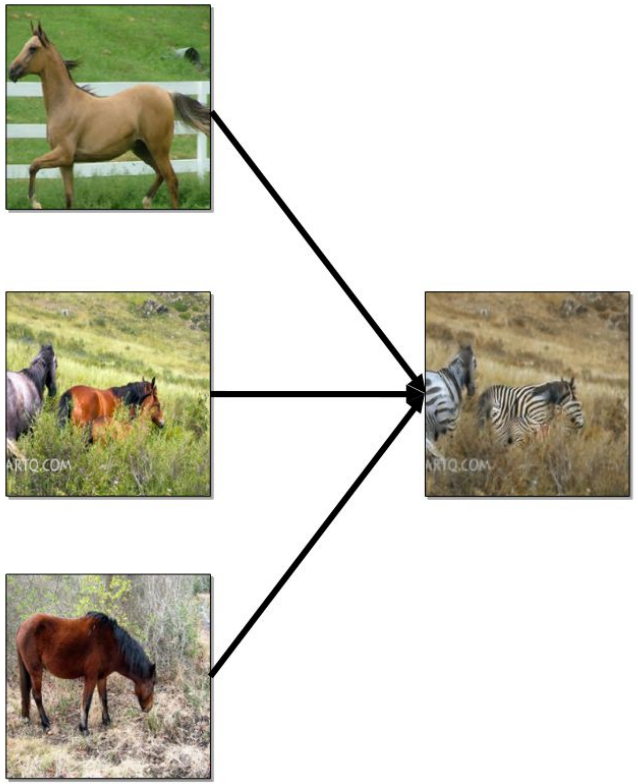


Discriminator

Real too!

Credits: Jun-Yan Zhu

Moreover



mode collapse!

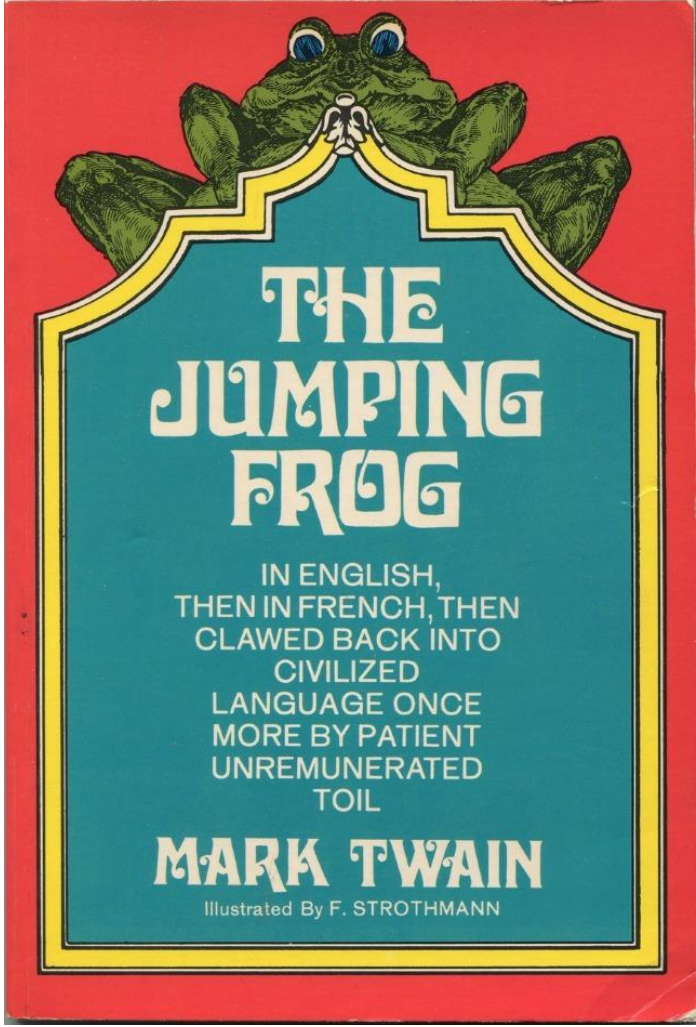
Credits: Jun-Yan Zhu

Cycle Consistency

Original: *“There was a feller here once by the name of Jim Smiley, in the winter of ’49 or maybe it was the spring of ’50 I don’t recollect exactly”*

Back Translation: *“It there was one time here an individual known under the name of Jim Smiley; it was in the winter ’49, possibly well at the spring of ’50, I no me recollect not exactly.”*

— Mark Twain



Slide Inspired by Prof Alexei Efros’ Lecture: Self-supervision, Meta-supervision, Curiosity: Making Computers Study Harder

Image Source

Cycle Consistency

- **G(F(x)) = x**, and
- **F(G(y)) = y**
- F and G are inverses of each other
- F and G are neural networks
- We will learn a function and its inverse together

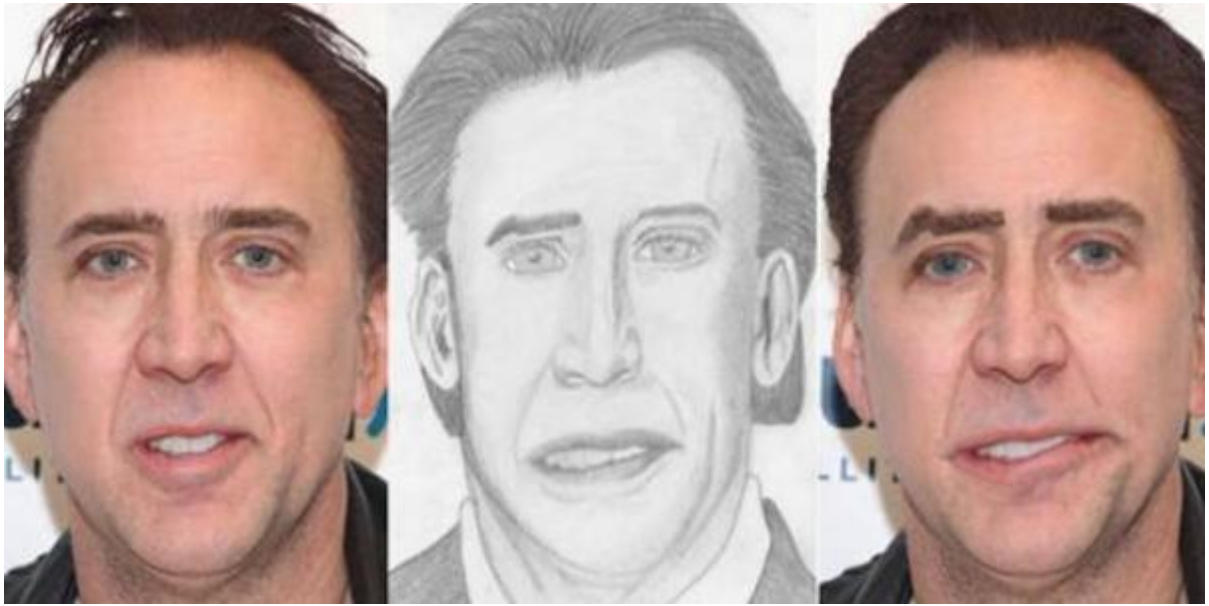
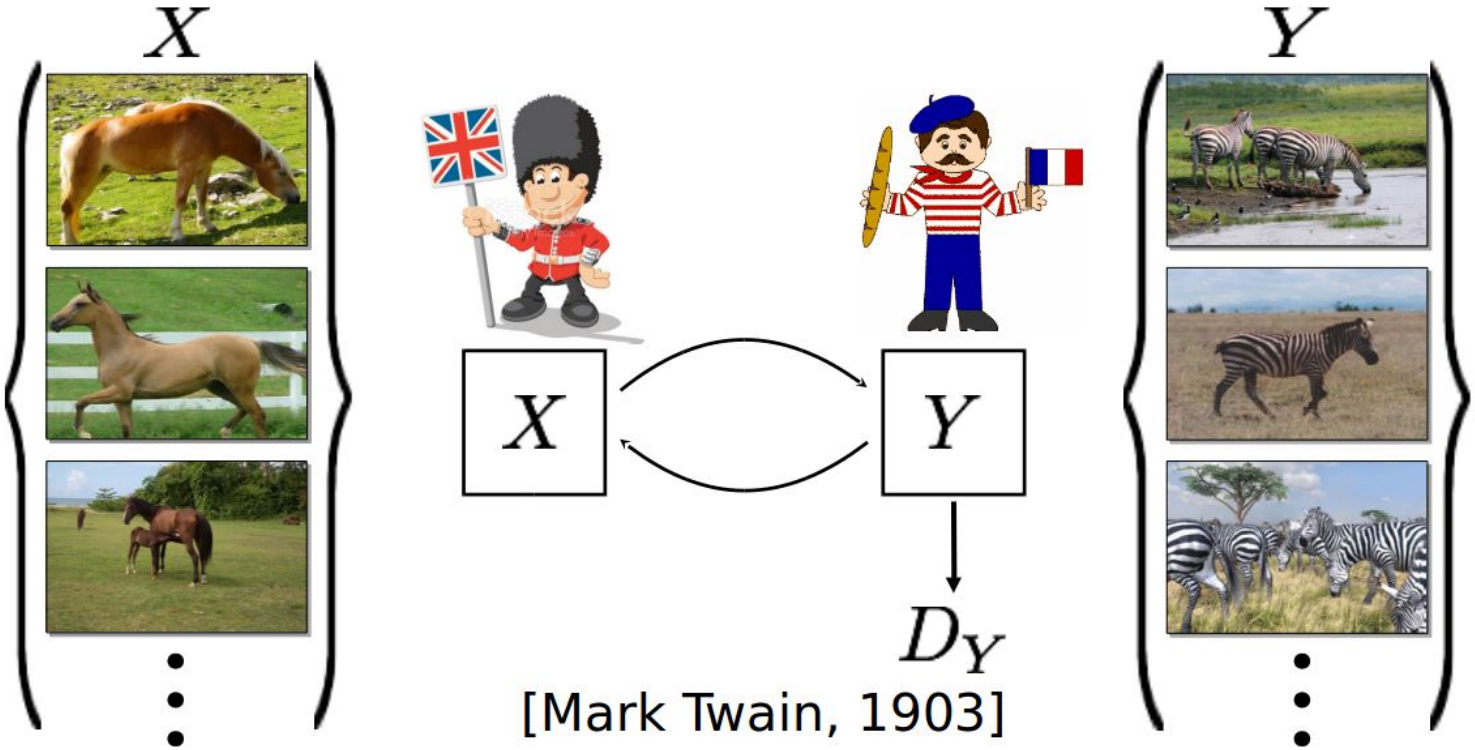


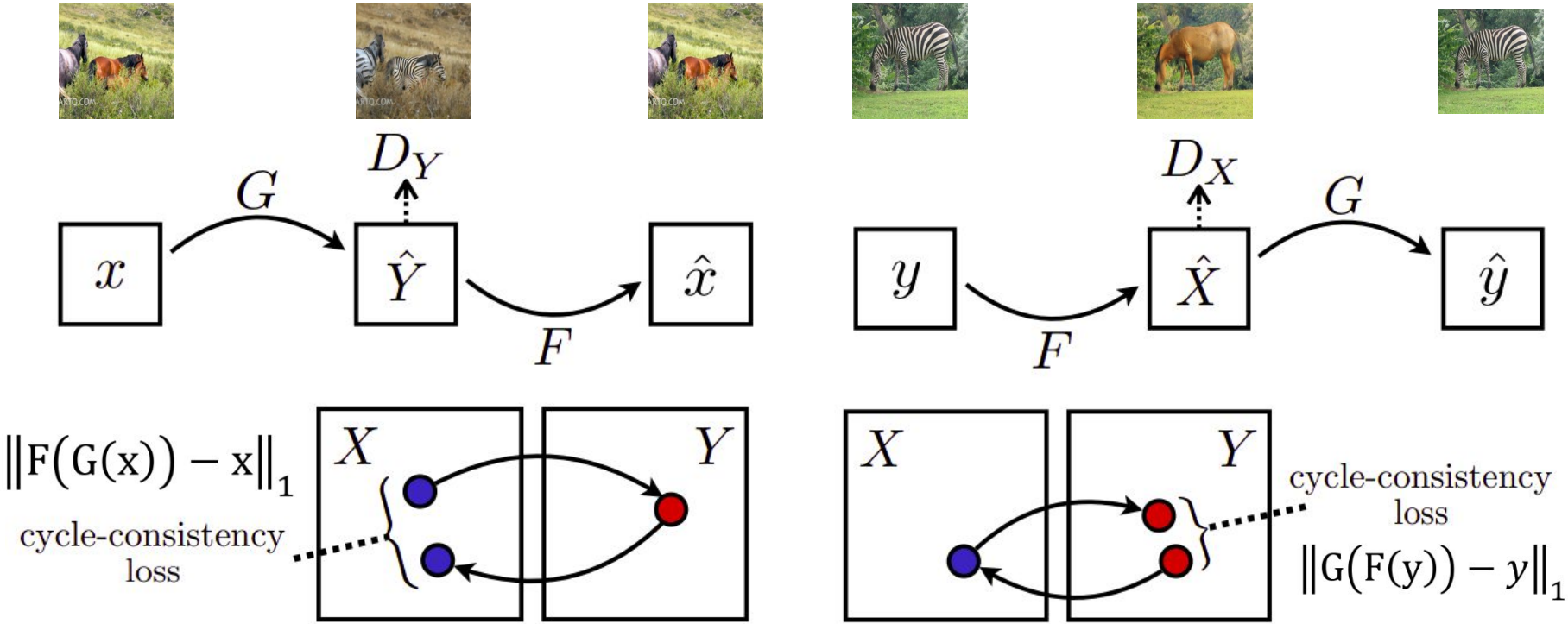
Image Source

Cycle Consistency



Credits: Jun-Yan Zhu

Cycle Consistency Loss

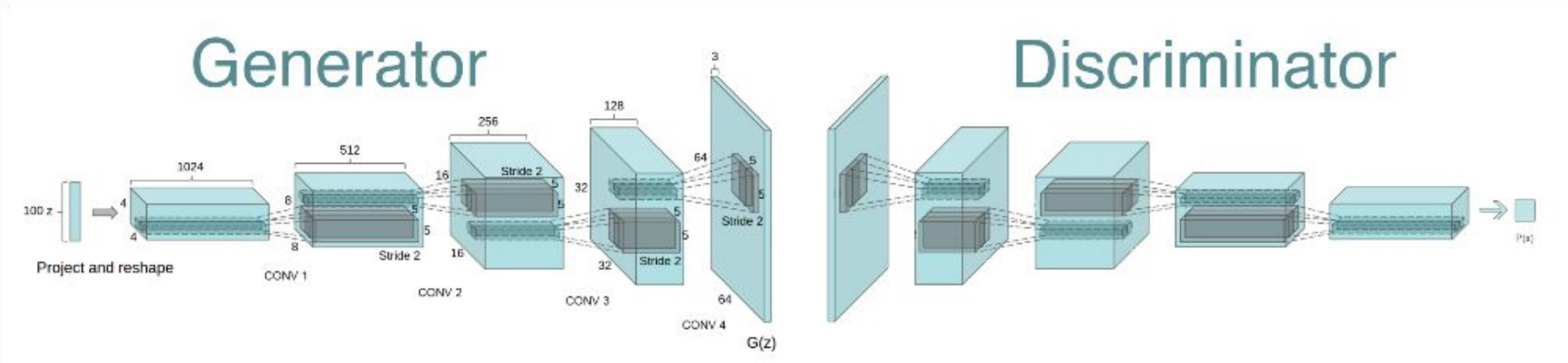


Zhu, J., Park, T., Isola, P., & Efros, A.A. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *ICCV (2017)*

Why Study Generative Modeling?

- Create more data
- Manipulate already existing data
- Representation Learning
  - “What I cannot create, I do not understand.” — Richard Feynman

DCGAN



Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR, abs/1511.06434*.

Using Learned Representations

- Trained a DCGAN on dataset S_1
- Extracted Discriminator's features from all layers for images in another dataset S_2
- Maxpooled each feature map to reduce spatial size to 4x4
- Concatenated and flattened into a single vector
- Trained an SVM on the vector to classify images in S_2

Model	Accuracy
1 Layer K-means	80.6%
3 Layer K-means Learned RF	82.0%
View Invariant K-means	81.9%
Exemplar CNN	84.3%
DCGAN (ours) + L2-SVM	82.8%

S_1: ImageNet-1K, S_2: CIFAR-10

Model	error rate
KNN	77.93%
TSVM	66.55%
M1+KNN	65.63%
M1+TSVM	54.33%
M1+M2	36.02%
SWWAE without dropout	27.83%
SWWAE with dropout	23.56%
DCGAN (ours) + L2-SVM	22.48%
Supervised CNN with the same architecture	28.87% (validation)

S_1: ImageNet-1K, S_2: SVHN

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *abs/1511.06434*.

Variational Autoencoders

Autoencoder

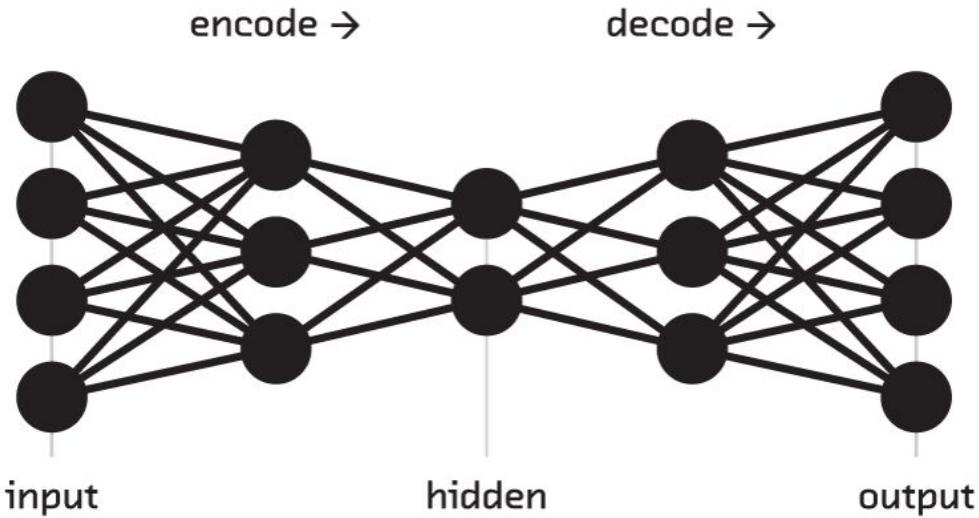
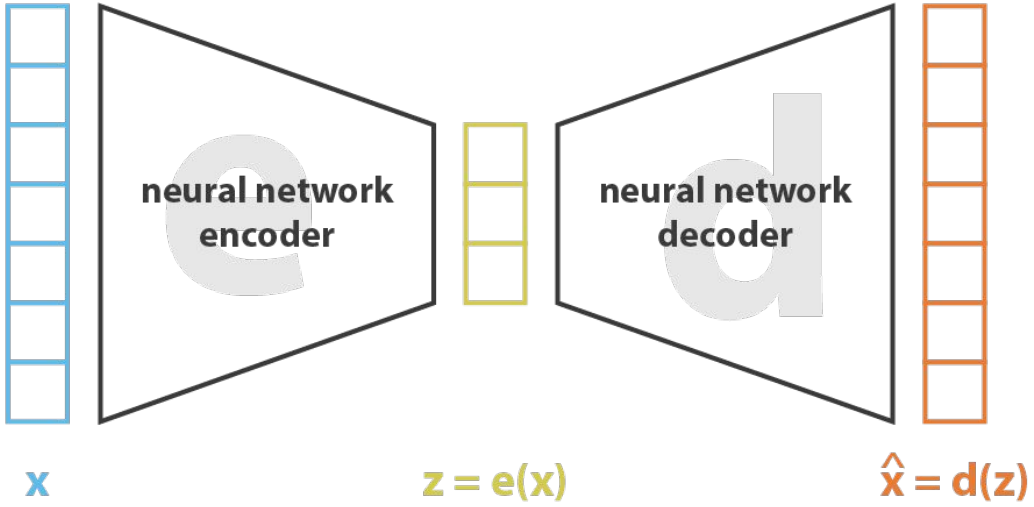


Image Source

Autoencoder Loss



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

Image Source

Variational Autoencoder

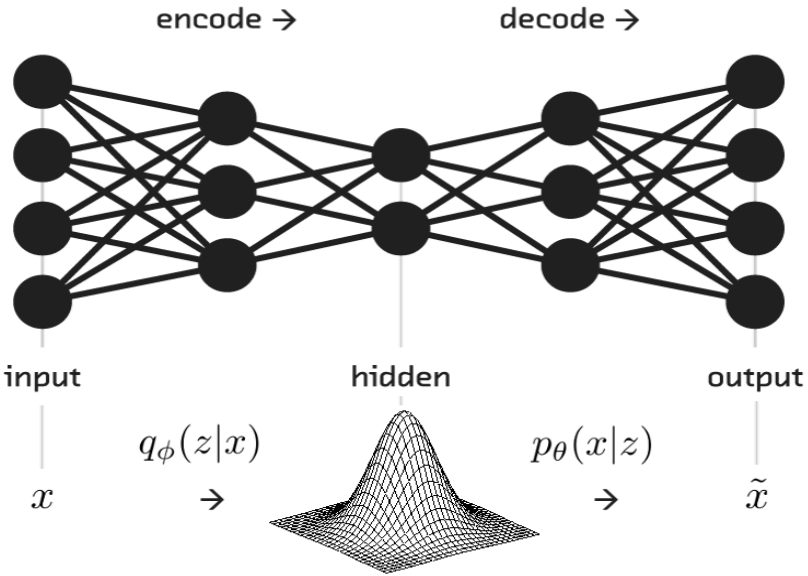
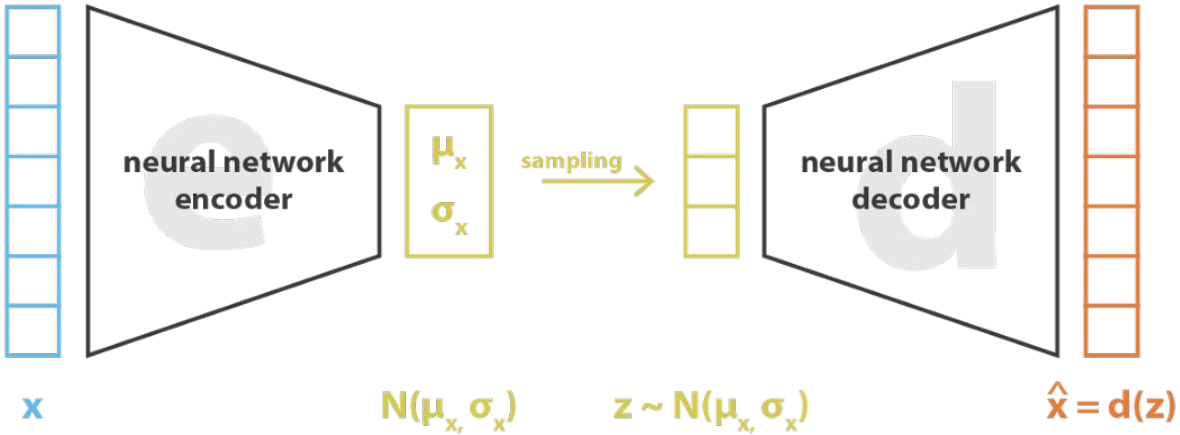


Image Source

Variational Autoencoder Loss



$$\text{loss} = ||x - \hat{x}||^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Image Source

$$-\frac{1}{2}\sum(1 + log(\sigma_x^2) - \mu_x^2 - \sigma_x^2)$$

Reparameterization Trick

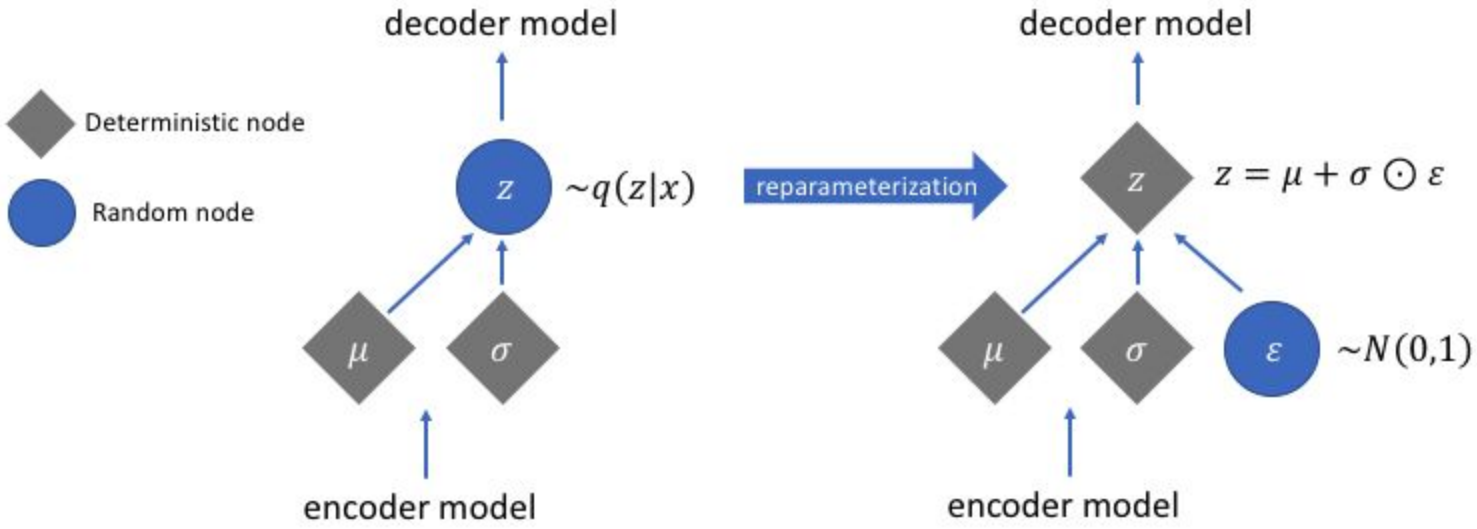
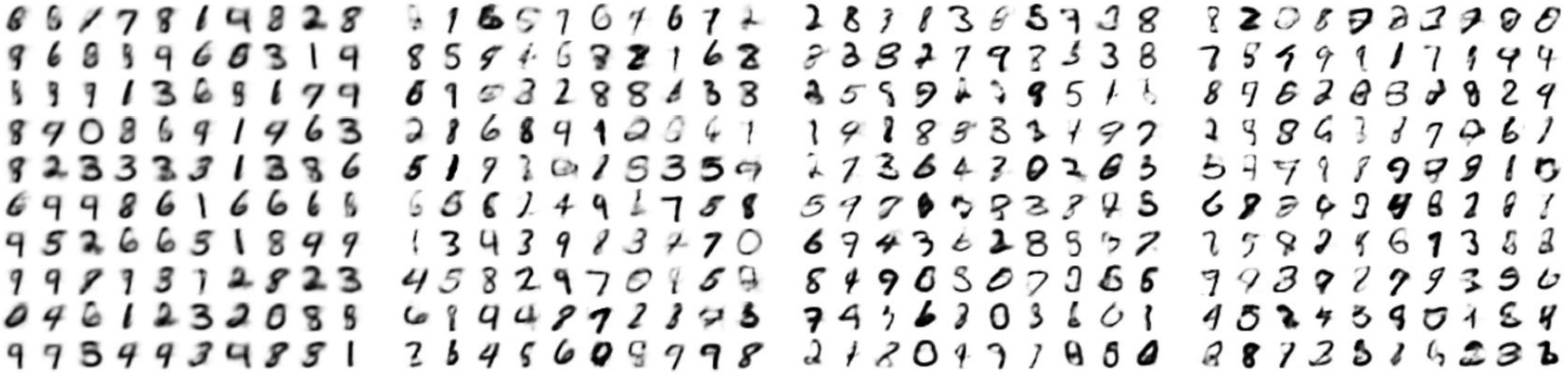


Image Source

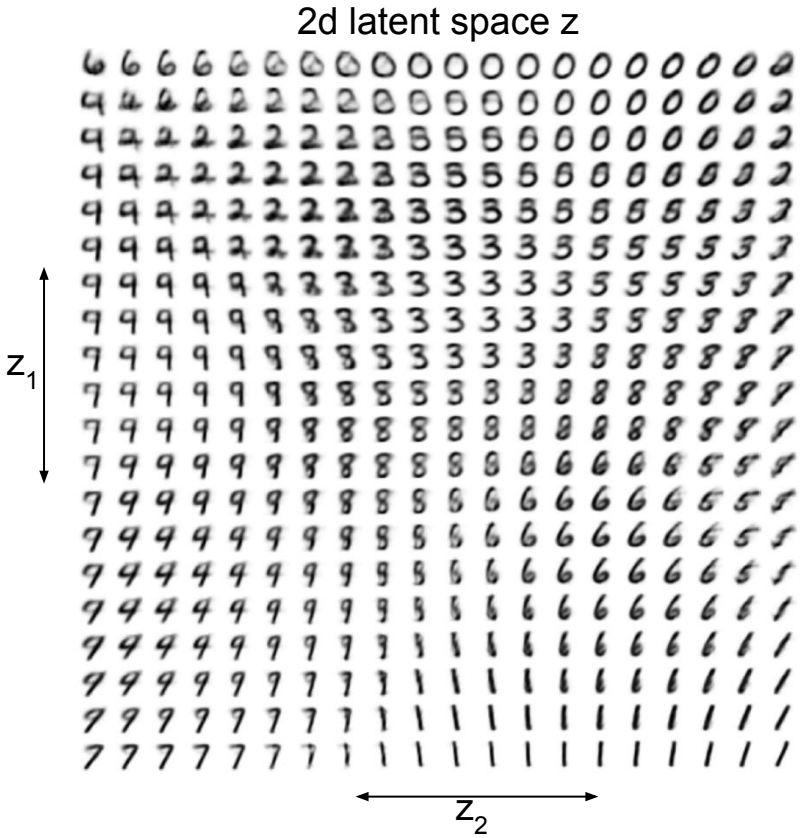
Data Generation



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Kingma, D.P., & Welling, M. Auto-Encoding Variational Bayes. ICLR (2014)

Data Generation



Kingma, D.P., & Welling, M. Auto-Encoding Variational Bayes. ICLR (2014)

References

- Deep Learning (for Computer Vision): Arjun Jain
- Deep Generative Models: Shenlong Wang
- Variational Autoencoders: Bhiksha Raj
- Generative Adversarial Networks: Pieter Abbeel, Xi (Peter) Chen, Jonathan Ho, Aravind Srinivas, Alex Li, Wilson Yan
- Likelihood Models: Pieter Abbeel, Xi (Peter) Chen, Jonathan Ho, Aravind Srinivas, Alex Li, Wilson Yan
- Conditional GAN: Mehdi Mirza, Simon Osindero