SDU Summer School

Deep Learning

Summer 2022

Welcome to the Summer School

Dude, what about the cool stuff?

- **Style Transfer**
- **Autoencoders**
- **GANs**
- **Modern Architectures**

We have covered the basics

- We have looked at the types of networks available
- Our output was normally a classification
- But, how to do the awesome stuff?
  - How to create images?
  - How to create art with networks?
  - …

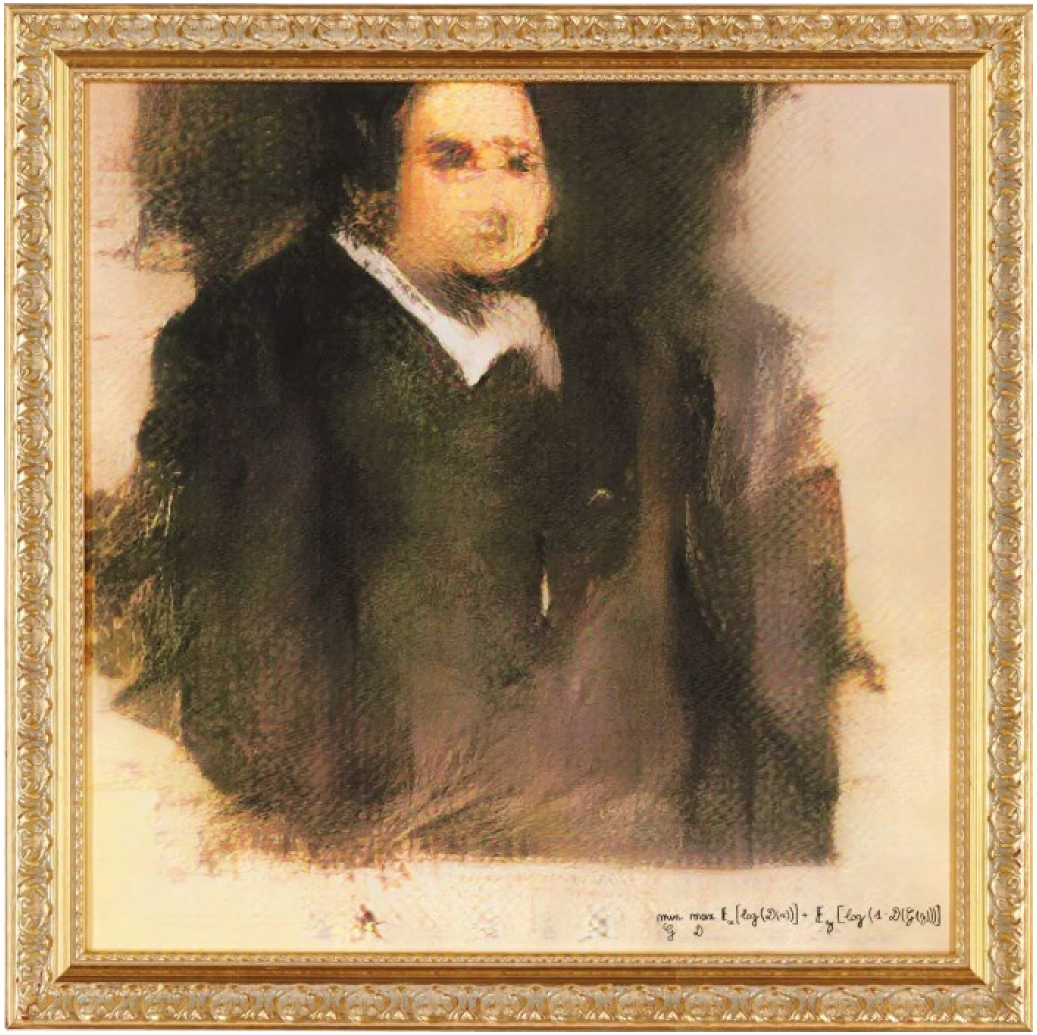
Cool Examples

Which Images are Real?

Which Images are Real? - None

Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

Machine Learning Generated Artwork Auctions Off for $ 432,500



Dude, what about the cool stuff?

- Style Transfer
- Autoencoders
- GANs
- Modern Architectures

Good news: We know most of the stuff

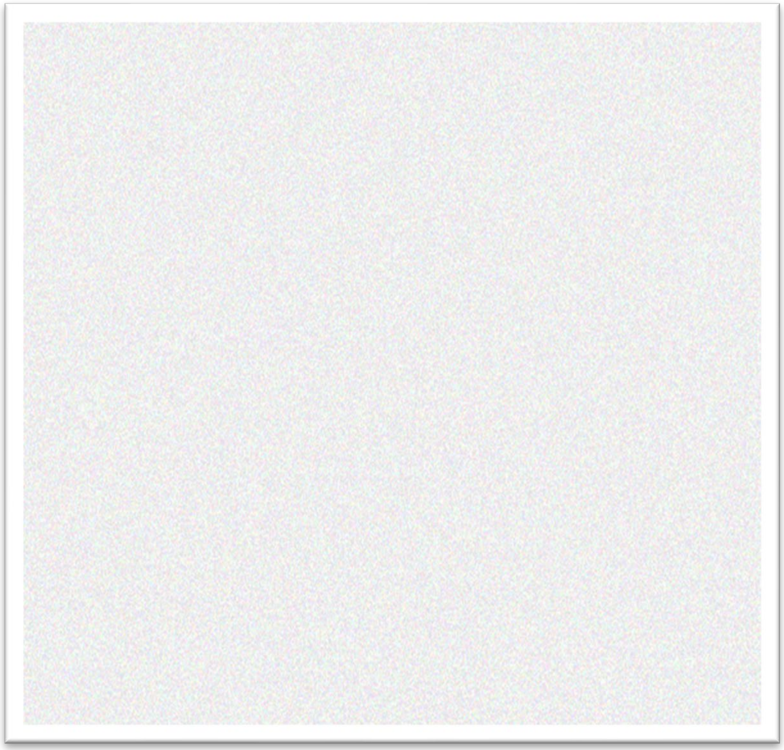
- First of all, good news:
  - We know most of the stuff
  - We understand whats going on under the hood

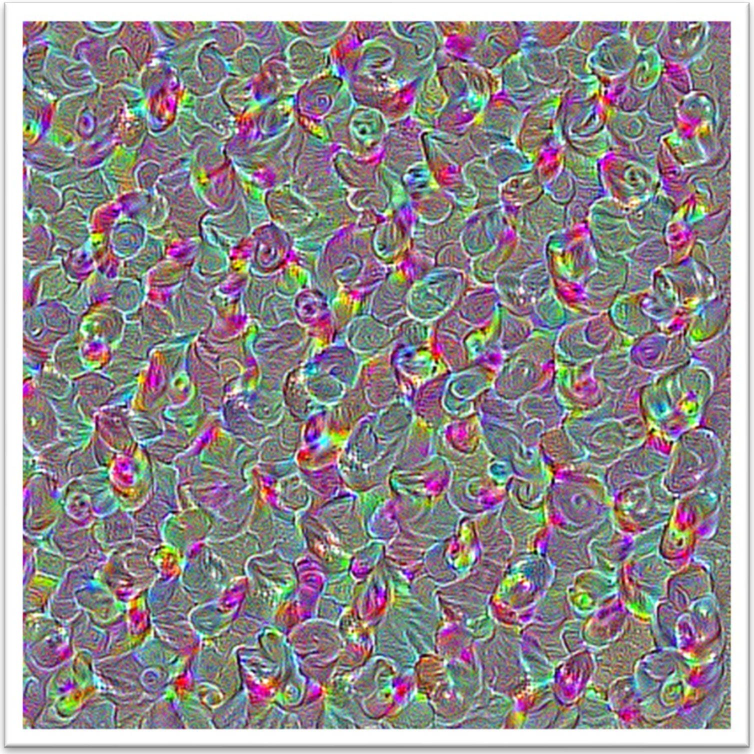
- Many of these awesome applications are standard networks with a creative combination of loss functions and optimization

- Require not only long time for training, but are also highly individual solutions

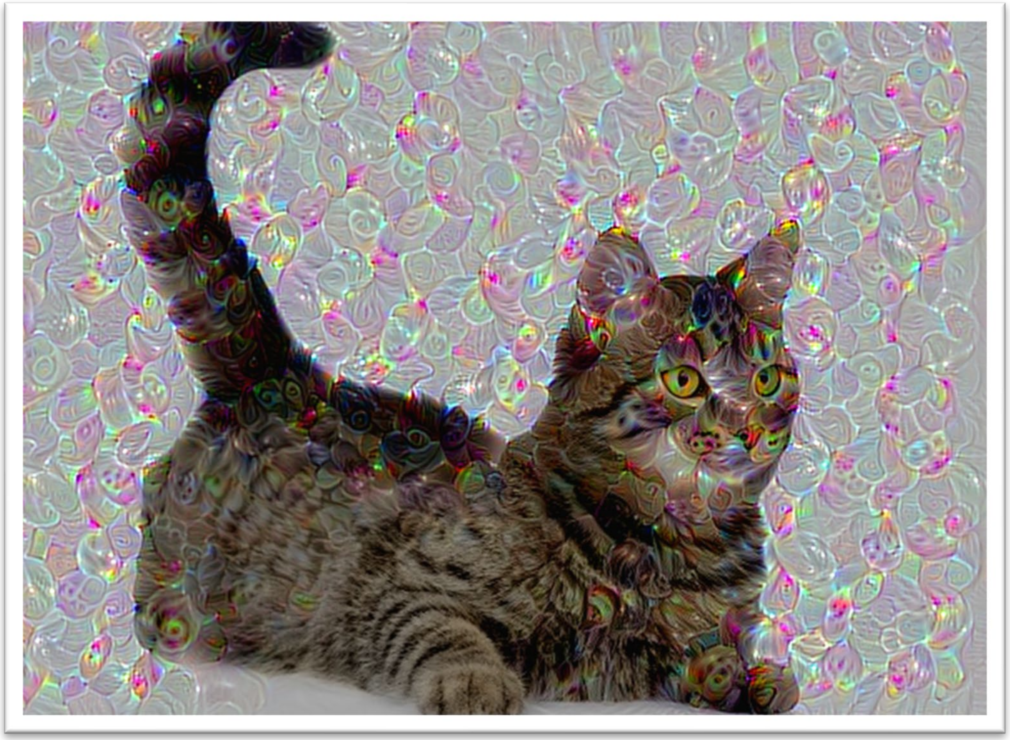
- But we are perfectly able to understand these tricks

Style Transfer: We already did it (almost)!





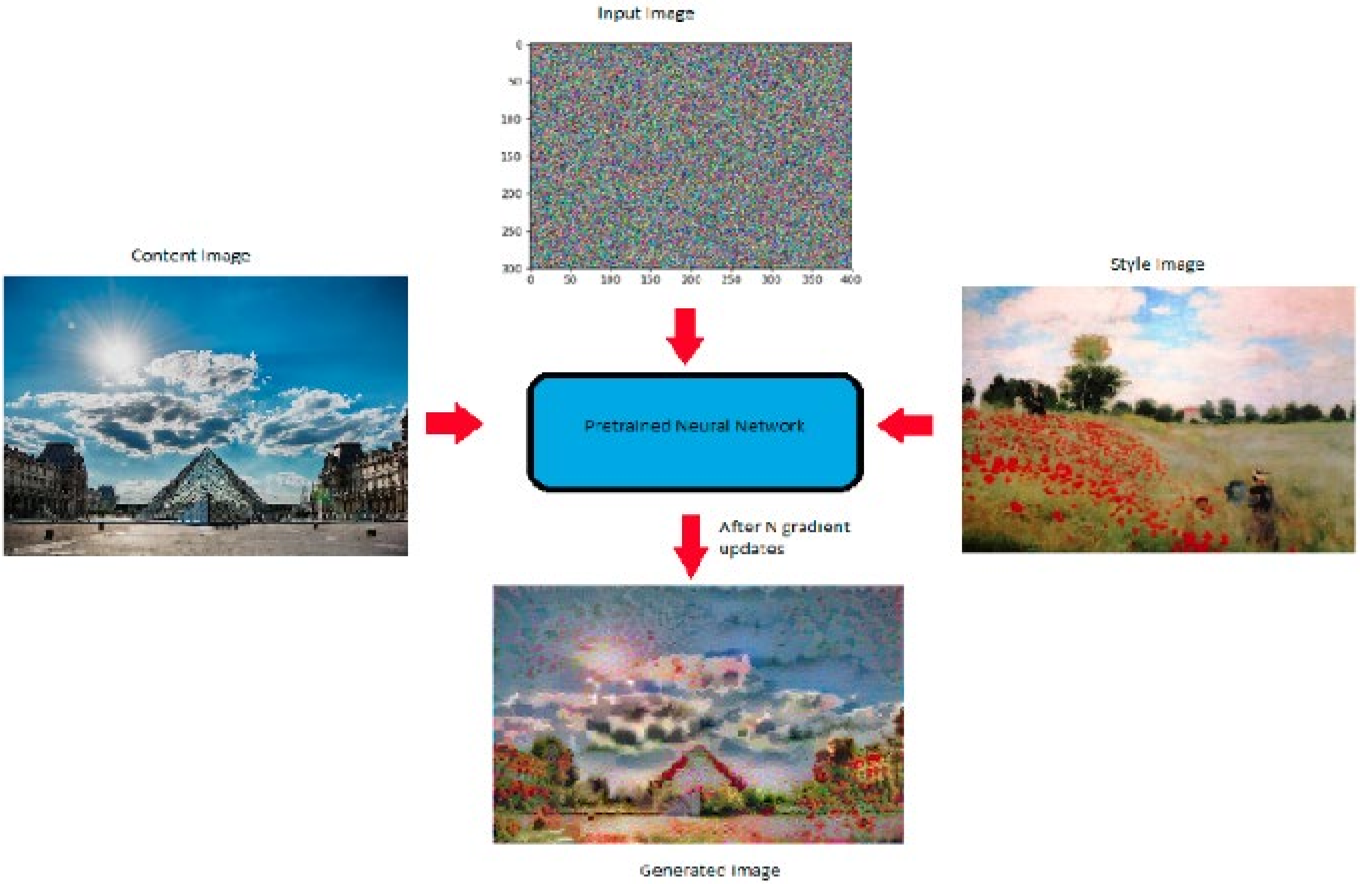
Replace Noise with an Actual Image



(Almost) Style Transfer

- What have we done?
- We have modified an input image and superposed the style of the wanted filter over it
- But what if we do not want to get the filter transferred but the style of an arbitrary image?
- Works almost the same!

How Style Transfer works



Overall Procedure

- We again use a pre-trained network, like vgg16
- We pass the style image through and look at the activation maps of this style image
- We pass the content image through and also look at the activations
- Now we gradient-learn the noise image according to a two-part loss function:
  - One part tries to capture and maintain the structure of the content image
  - The other part tries to capture the style

The Loss Function

- First part:
  - How close is our noise image to the content?
  - We extract the image at a layer (normally low layer which maintains geometric order of the image, e.g., "block2_conv2")

$$L_{content} = \frac{1}{2}\sum_{i,j,l}\left(F_{ij}^{l} - P_{ij}^{l}\right)^2$$

- Second part:
  - Works very similar, but we consider different activation maps which are more representative of the style ([block1_conv2, block2_conv2, block3_conv3, block4_conv3, block5_conv3])
  - Further, we do not minimize the difference between the activations, but increase the correlation of the activation maps using a gram-matrix

The Result

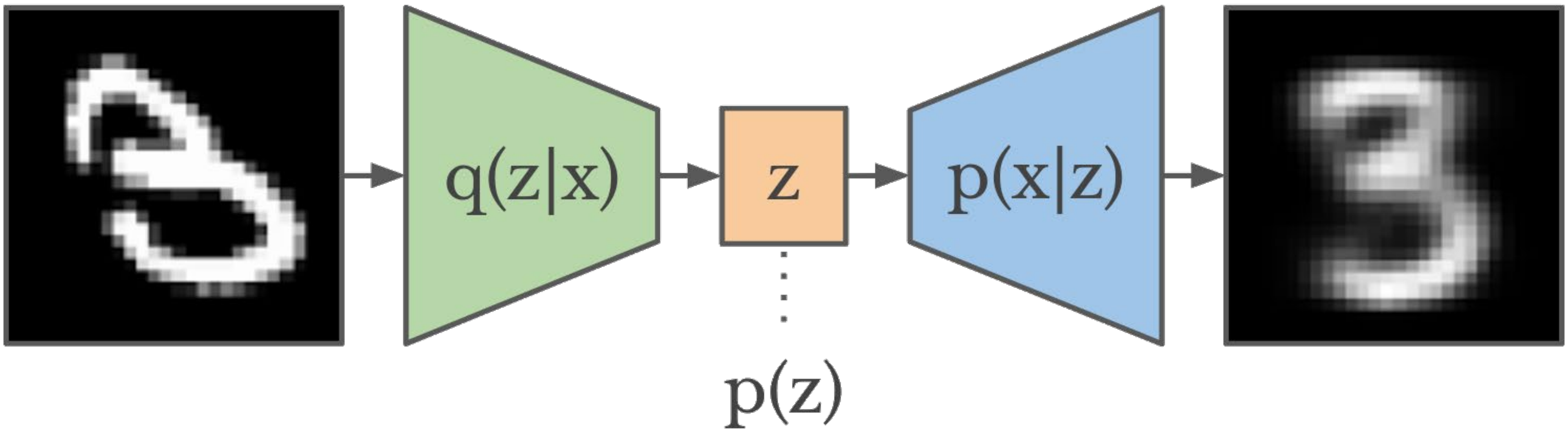
➢ https://towardsdatascience.com/style-transfer-styling-images-with-convolutional-neural-networks-7d215b58f461

Dude, what about the cool stuff?

- Style Transfer
- **Autoencoders**
- GANs
- Modern Architectures

What is an Autoencoder?

- **An autoencoder is a data compression algorithm**
  - Typically an artificial neural network trained to copy its input to its output



- Normally two stages:
  1. Compress input to (normally) smaller internal representation
  2. Reconstruct original input as closely as possible

Not to Mistake with General Data Compression

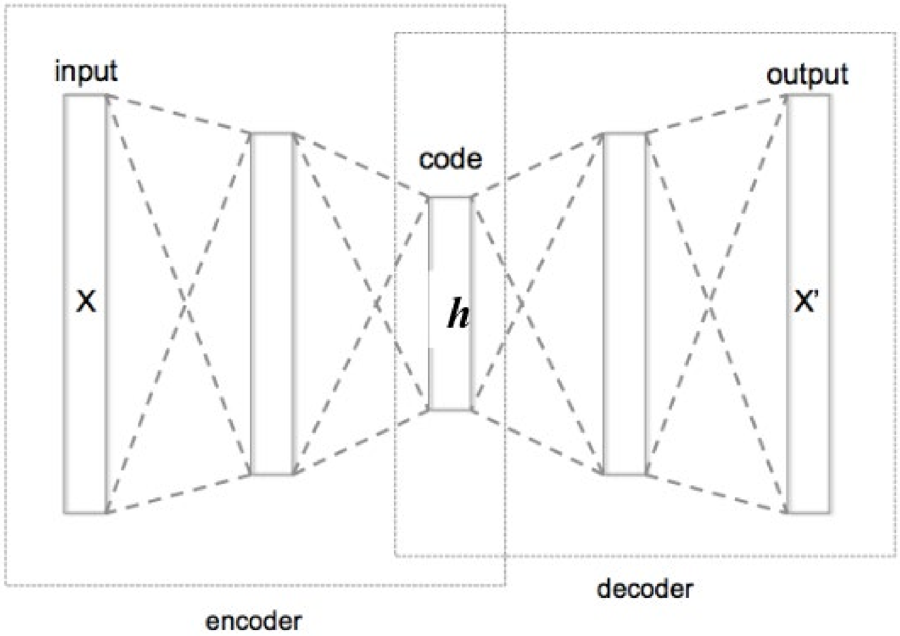
- Autoencoders are data-specific
  - i.e., only able to compress data similar to what they have been trained on
- This is different from, say, MP3 or JPEG compression algorithm
  - Which make general assumptions about "sound/images”, but not about specific types of sounds/images
  - Autoencoder for pictures of cats would do poorly in compressing pictures of trees
- Autoencoders are lossy
  - i.e., exact reconstruction is normally not possible
- **Autoencoders are learnt**

Rationale of an Autoencoder

- An autoencoder that simply learns to set $g\big(f(x)\big) = x$ everywhere is not really useful
- Autoencoders are designed to be unable to copy perfectly
  - They are restricted in ways to copy only approximately
  - Copy only input that resembles training data
- Because model is forced to prioritize which aspects of input should be copied, it often learns useful properties of the data
- Modern autoencoders have generalized the idea of encoder and decoder beyond deterministic functions to stochastic mappings $p_{\text{encoder}}(h|x)$ and $p_{\text{decoder}}(x|h)$

Training and Loss Function

- Encoder f and decoder g
  - $f: X \rightarrow h$
  - $g: h \rightarrow X$
  - $\mathrm{argmin}_{\mathrm{f,g}} \|X - (f \circ g)X\|^2$



- One hidden layer
  - Non-linear encoder
  - Takes input $\boldsymbol{x} \in \mathbb{R}^d$
  - Maps into output $\boldsymbol{h} \in \mathbb{R}^p$ (σ being any activation function)

$$\boldsymbol{h} = \sigma_1(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \qquad \boldsymbol{x}' = \sigma_2(\boldsymbol{W}'\boldsymbol{h} + \boldsymbol{b}')$$

- Minimize reconstructions error

$$L(x, x') = \|x - x'\|^2 = \left\|x - \sigma_2\left(\boldsymbol{W}'\left(\sigma_1(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})\right) + \boldsymbol{b}'\right)\right\|^2$$

Undercomplete Autoencoder

- Copying input to output sounds useless
- But we are not interested in the output of the decoder
- We hope that training the autoencoder to perform copying task will result in h taking on useful properties
- To obtain useful features, constrain $\boldsymbol{h}$ to have lower dimension than $\boldsymbol{x}$
  - Such an autoencoder is called undercomplete
  - Learning the undercomplete representation forces the autoencoder to capture most salient features of training data

Encoder/Decoder Capacity

- If encoder f and decoder g are allowed too much capacity
  - autoencoder can learn to perform the copying task without learning any useful information about distribution of data

- Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map $x^{(i)}$ to code i.

- The decoder can learn to map these integer indices back to the values of specific training examples

- Autoencoder trained for copying task fails to learn anything useful if f/g capacity is too great

Use Regularization

- Ideally, choose code size (dimension of h) small and capacity of encoder f and decoder g based on complexity of distribution modeled

- Regularized autoencoders provide the ability to do so
  - Rather than limiting model capacity by keeping encoder/decoder shallow and code size small
  - They use a loss function that encourages the model to have properties other than copy its input to output

Regularized Autoencoder Properties

- Regularized AEs have properties beyond copying input to output:
  - **Sparsity of representation**
  - **Robustness to noise**
  - **Robustness to missing inputs**
  - **Smallness of the derivative of the representation**

- Regularized autoencoder can be nonlinear and overcomplete
  - But still learn something useful about the data distribution even if model capacity is great enough to learn trivial identity function

Sparse Autoencoders

- A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the code layer $\boldsymbol{h}$, in addition to the reconstruction error:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(h)$$

- They are often used to learn good features for other tasks, like classification

- The hidden layers then can be interpreted as latent variables of a generative model
  - An autoencoder that has been trained to be sparse must respond to unique statistical features of the dataset rather than simply perform copying

Denoising Autoencoders (DAE)

- Traditional autoencoders minimize

$$L\left(\boldsymbol{x}, g(f(\boldsymbol{x}))\right)$$

- A DAE minimizes

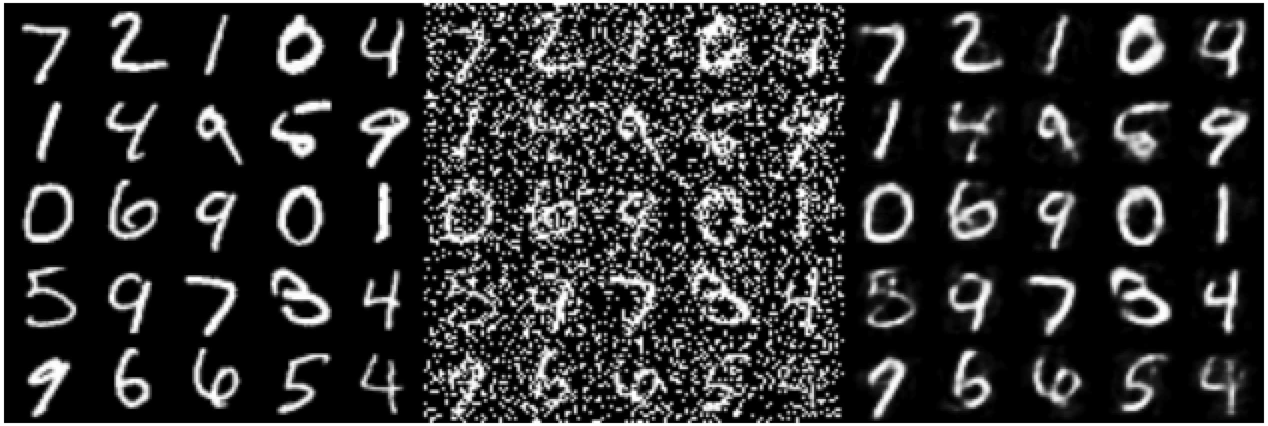
$$L\left(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))\right)$$

  - where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ that has been corrupted by some form of noise
  - The autoencoder must undo this corruption rather than simply copying the input

- Denoising training forces f and g to implicitly learn the structure of $p_{\text{data}}(x)$

Example

- An autoencoder with high capacity can end up learning an identity function (also called null function) where input=output
- A DAE can solve this problem by corrupting the data input
- Corrupt input nodes by setting 30-50% of random input nodes to zero



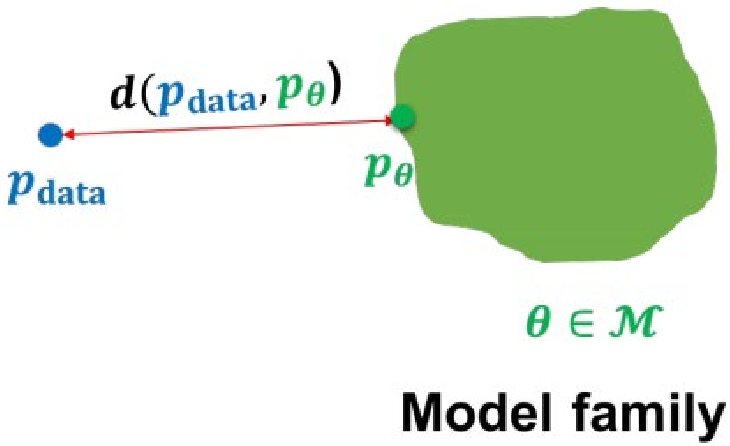
Original input, corrupted data and reconstructed data. Copyright by opendeep.org.

➢ https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2

Dude, what about the cool stuff?

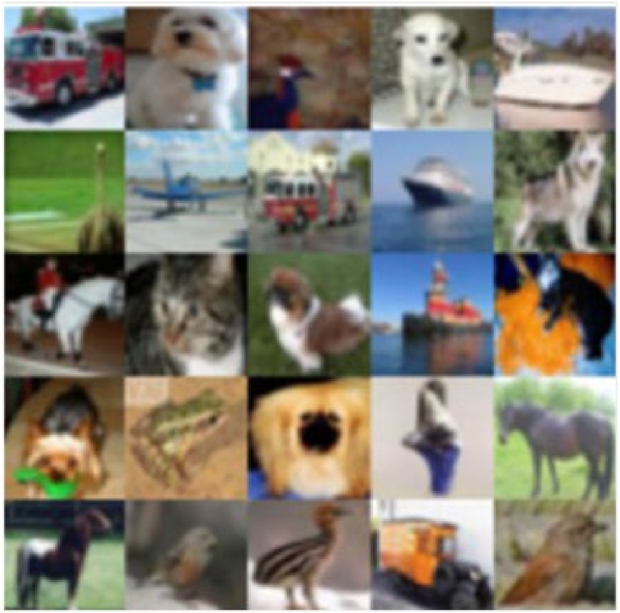
- Style Transfer
- Autoencoders
- **GANs**
- Modern Architectures

Remember Back: Generative Models



- We have a real data distribution p_{data}
- We want to approximate this distribution as closely as possible
- Required a lot of likelihood calculations, etc.
- To see how close we are, we would need to know p_{data}

Comparing Distributions via Samples



vs.

$S_1 = \{\mathbf{x} \sim P\}$ $S_2 = \{\mathbf{x} \sim Q\}$

- Given a finite set of samples from two distributions $S_1 = \{x \sim P\}$ and $S_2 = \{x \sim Q\}$, how can we tell if these samples are from the same distribution? (i.e., $P = Q$?)

Two-sample Tests

- Given $S_1 = \{x \sim P\}$ and $S_2 = \{x \sim Q\}$, a two-sample test considers the following hypotheses
  - Null hypothesis H_0: $P = Q$
  - Alternate hypothesis H_1: $P \neq Q$

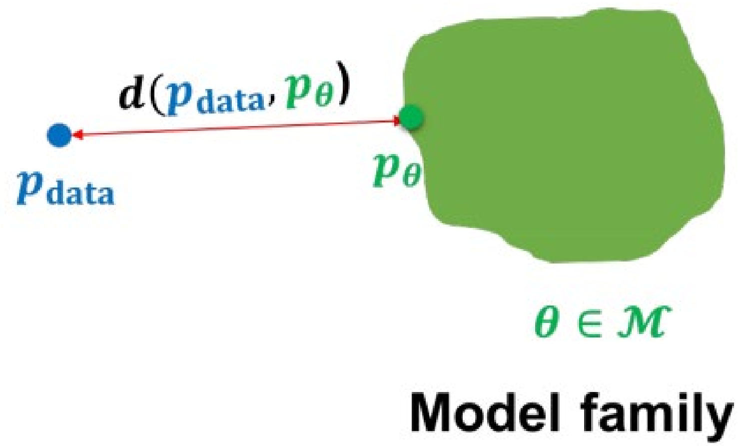
- Test statistic T compares S_1 and S_2 e.g., difference in means, variances of the two sets of samples

- If T is less than a threshold α, then accept H_0 else reject it

- **Key observation**: Test statistic is **likelihood-free** since it does not involve P or Q (only samples)

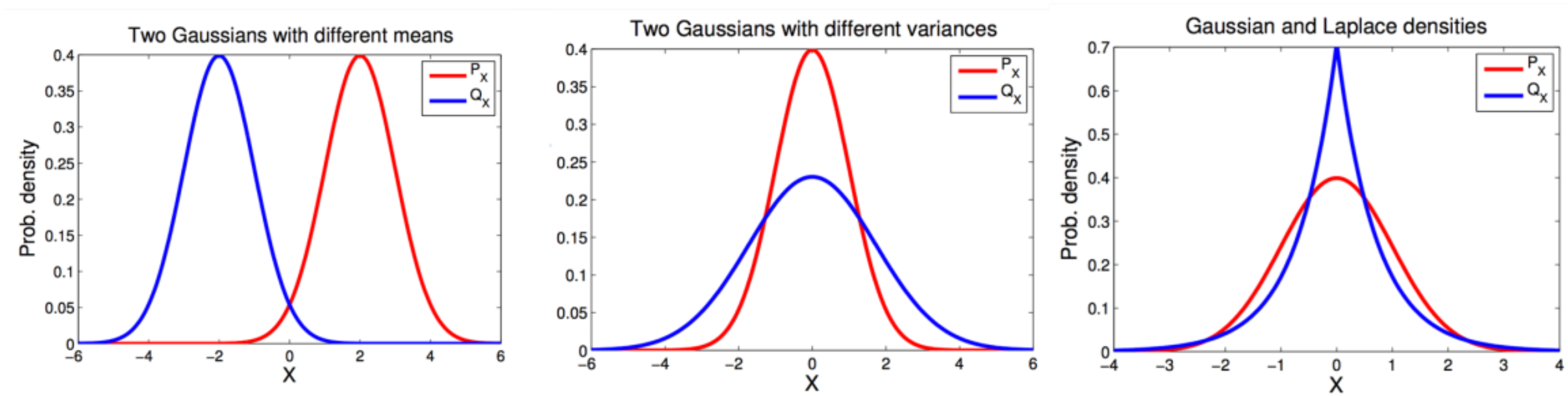
Generative Modeling and Two-Sample Tests

$\mathbf{x}^{(j)} \sim p_{\text{data}}$
$j = 1, 2, \ldots, |\mathcal{D}|$



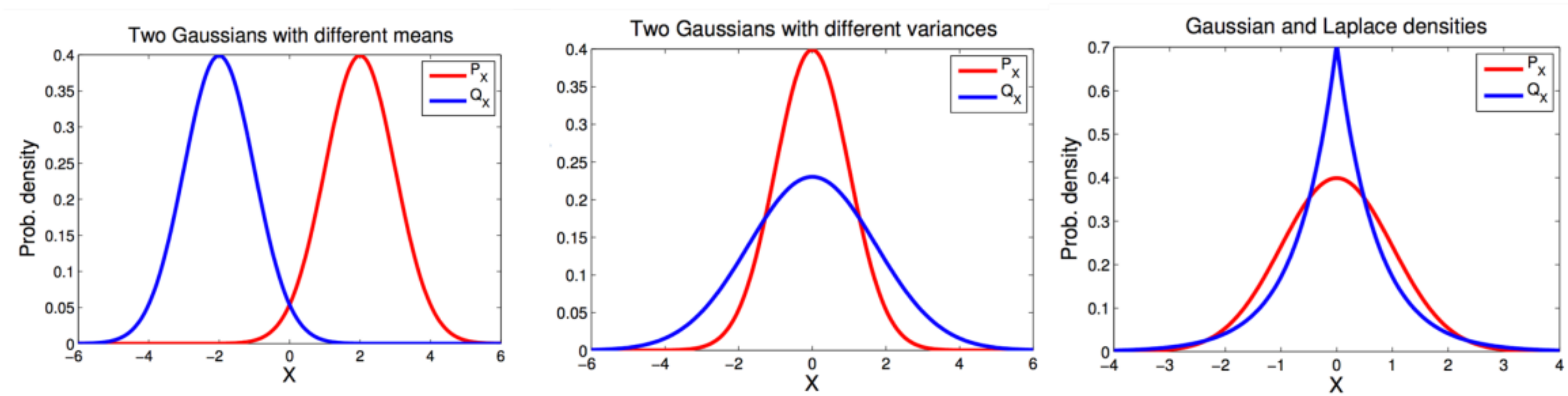
- Apriori we assume direct access to $S_1 = D = \{\boldsymbol{x} \sim p_{\text{data}}\}$
- In addition, we have a model distribution p_θ
  - Assume that the model distribution permits efficient sampling (e.g., directed models). Let $S_2 = \{x \sim p_\theta\}$
- Alternate notion of distance between distributions: Train the generative model to minimize a two-sample test objective between S_1 and S_2

Two-Sample Test via a Discriminator



- Many different test statistics exist to distinguish distributions or samples statistics of them
- **But:** Finding a two-sample test objective in high dimensions is hard
- How should we find such a statistic for function we cannot fully assess?

Two-Sample Test via a Discriminator

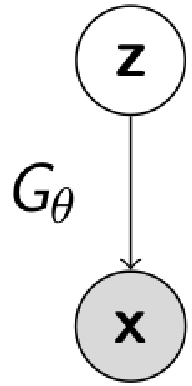


- Many different test statistics exist to distinguish distributions or sa
- Bu hard
- Ho ly assess?

Key idea: Learn a statistic that maximizes a suitable notion of distance between the two sets of samples S_1 and S_2

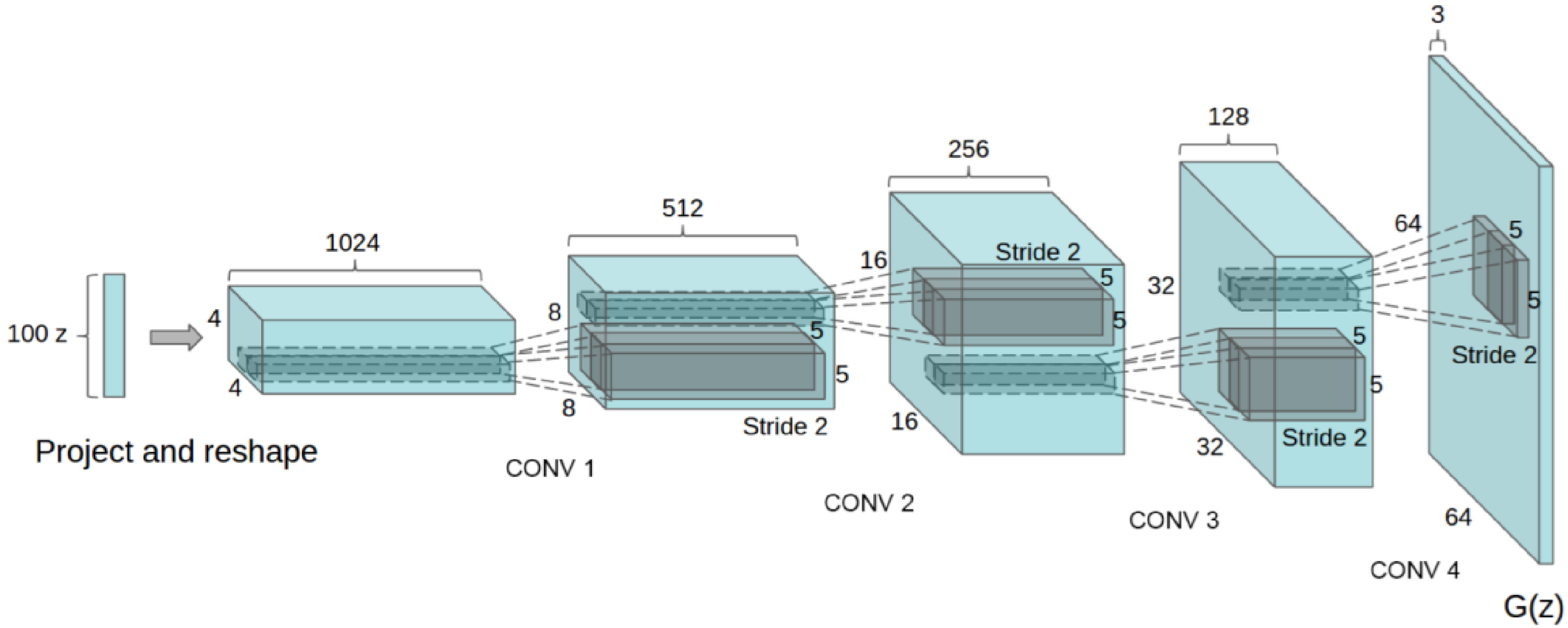
Generative Adversarial Networks

- A two player min-max game between a **generator** and a **discriminator**



- **Generator**
  - Directed, latent variable model with a deterministic mapping between $\boldsymbol{z}$ and $\boldsymbol{x}$ given by G_θ
  - Minimizes a two-sample test objective (in support of the null hypothesis $p_{\text{data}} = p_\theta$)

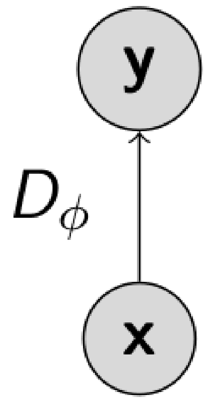
Typical Generator



- Unit Gaussian distribution on z, typically 10-100 dim.
- Up-convolutional deep network (reverse recognition CNN)

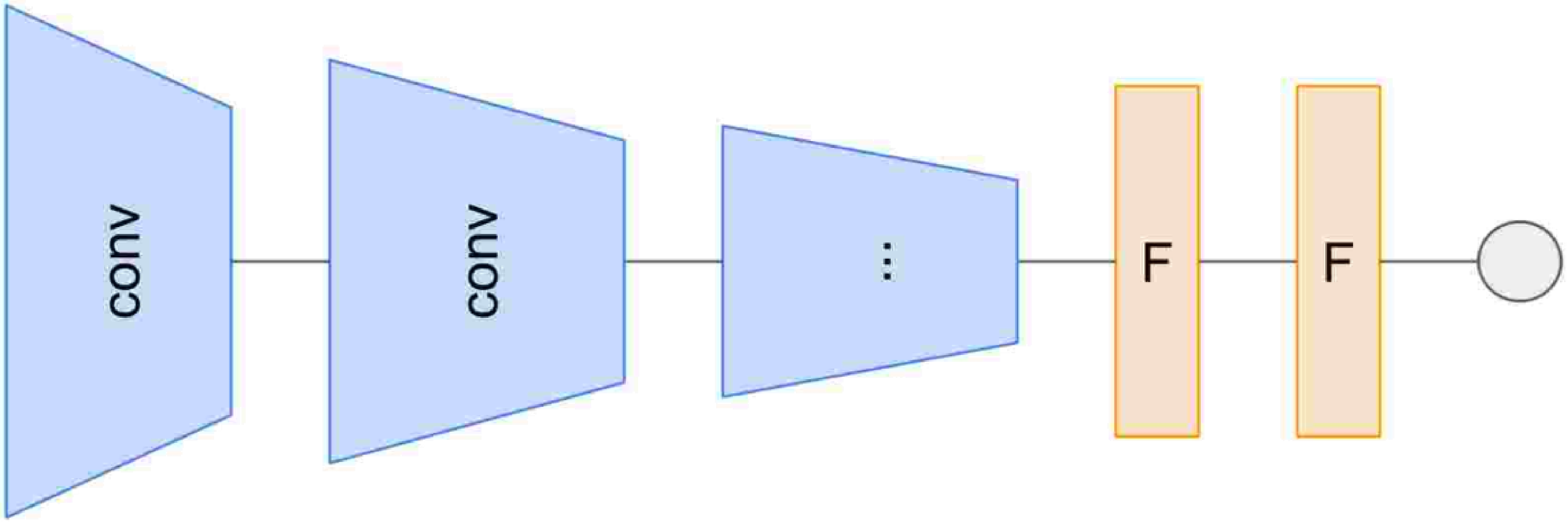
Generative Adversarial Networks

- A two player min-max game between a **generator** and a **discriminator**



- **Discriminator**
  - Any function (e.g., neural network) which tries to distinguish "real" samples from the dataset and "fake" samples generated from the model
  - Maximizes the two-sample test objective (in support of the alternate hypothesis $p_{\text{data}} \neq p_\theta$)

Typical Discriminator



Example of GAN objective

- **Training objective for discriminator**:

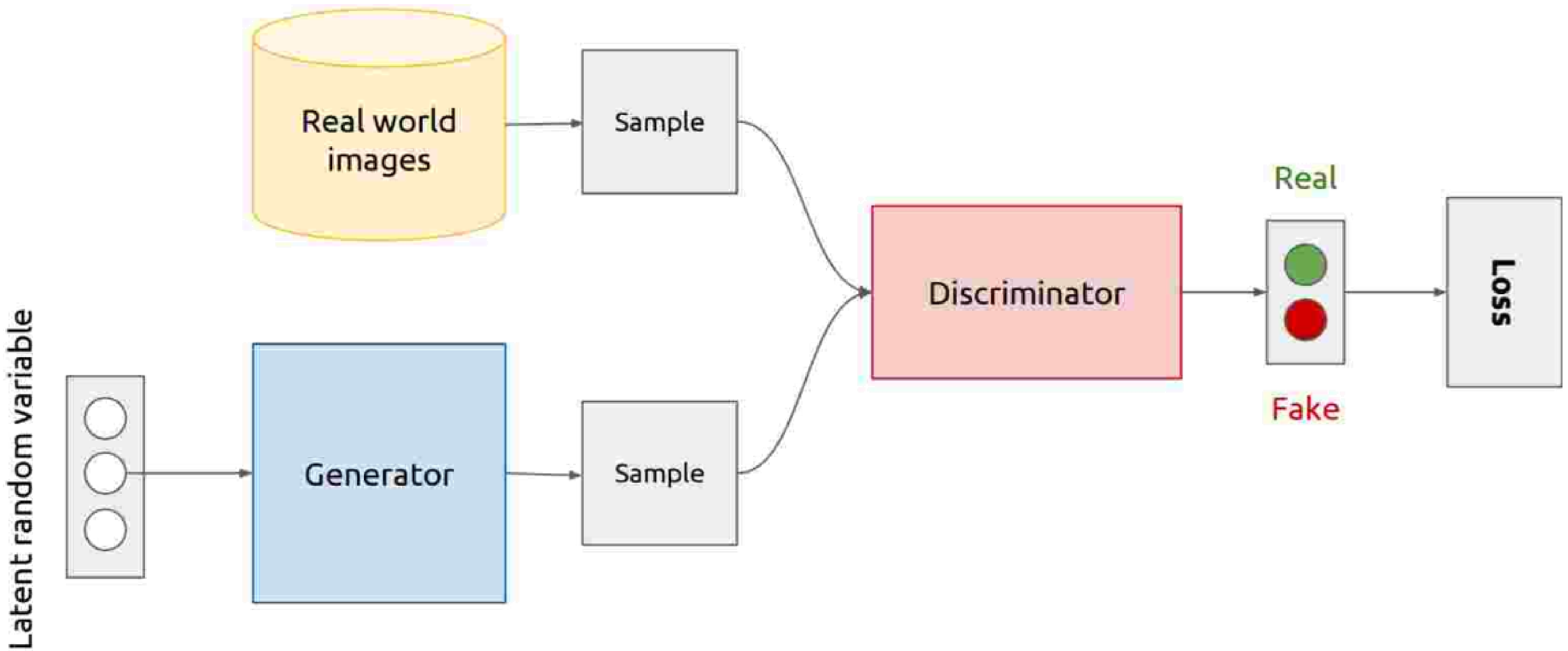
$$\max_D V(G,D) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x\sim p_G}[\log(1 - D(x))]$$

- For a fixed generator G, the discriminator is performing binary classification with the cross entropy objective
  - Assign probability 1 to true data points $x \sim p_{\text{data}}$
  - Assign probability 0 to fake samples $x \sim p_G$

- **Training objective for generator**:

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

Schematic Setup of Adversarial Training



The GAN training algorithm

- Sample minibatch of m training points $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}$ from D
- Sample minibatch of m noise vectors $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(m)}$ from p_z
- Update the generator parameters θ by stochastic gradient **descent**

$$\nabla_\theta V(G_\theta, D_\Phi) = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \log\left(1 - D_\Phi\left(G_\theta\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

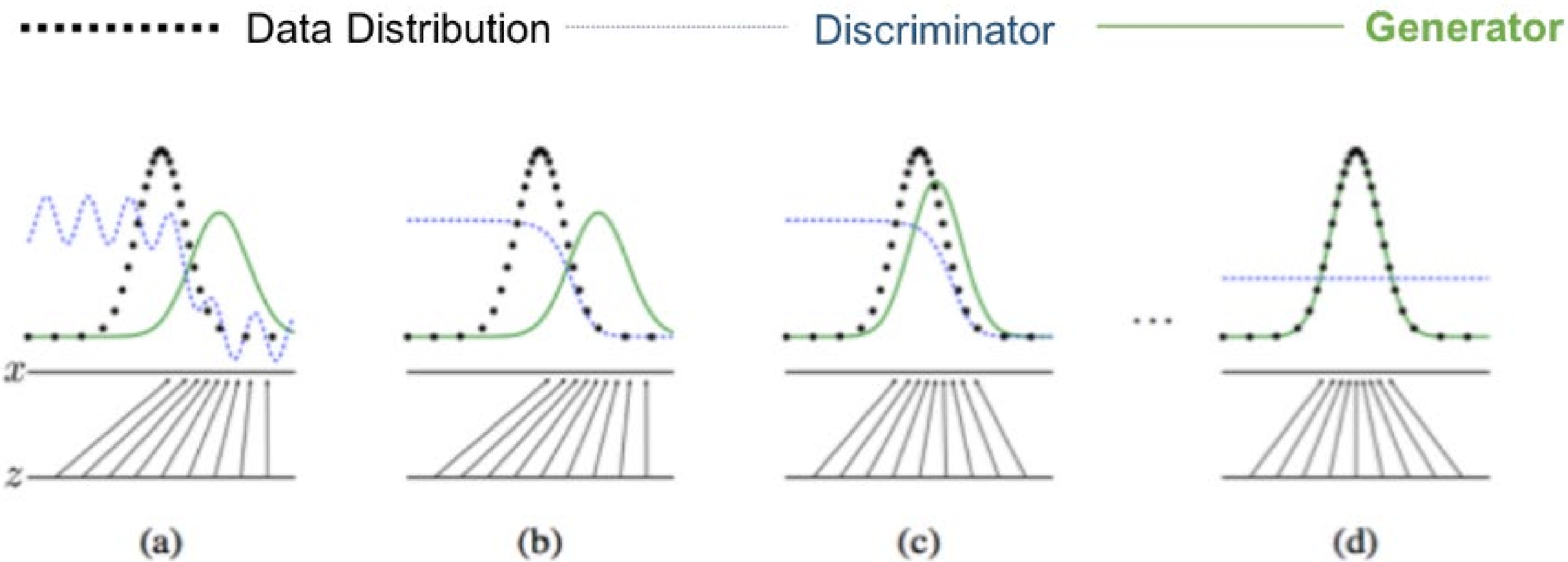
- Update the discriminator parameters Φ by stochastic gradient **ascent**

$$\nabla_\Phi V(G_\theta, D_\theta) = \frac{1}{m} \nabla_\Phi \sum_{i=1}^{m} \left[\log D_\Phi\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D_\Phi\left(G_\theta\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

- Repeat for fixed number of epochs

Alternating optimization in GANs

$$\min_{\theta} \max_{\Phi} V(G_\theta, D_\Phi) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D_\Phi(\boldsymbol{x})] + \mathbb{E}_{z\sim p(z)}\left[\log\left(1 - D_\Phi\left(G_\theta(z)\right)\right)\right]$$



Current state | Update discriminator | Update generator | Convergence

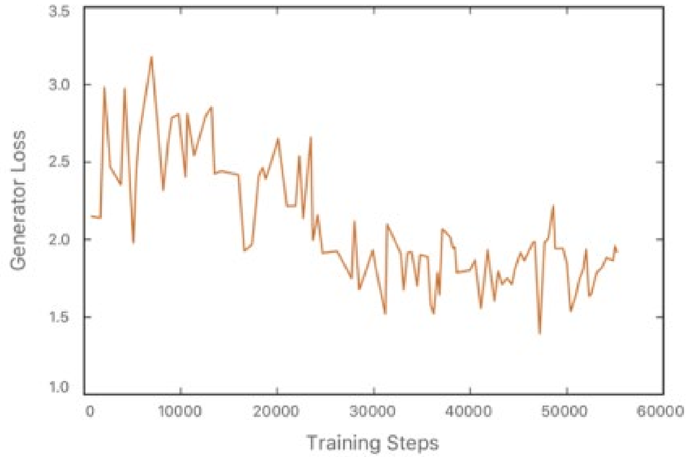
Goodfellow et al., 2014

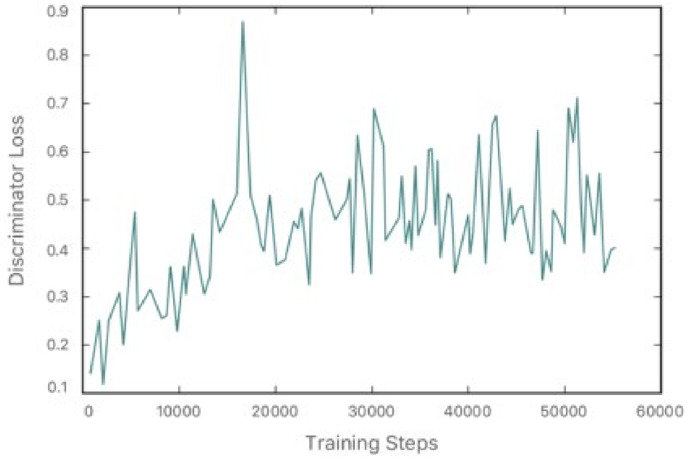
Both images are generated via GANs!

Source: Karras et al., 2018; The New York Times

Optimization challenges

- Theorem: If the generator updates are made in function space and discriminator is optimal at every step, then the generator is guaranteed to converge to the data distribution
- **Unrealistic assumptions!** In practice, the generator and discriminator loss keeps oscillating during GAN training





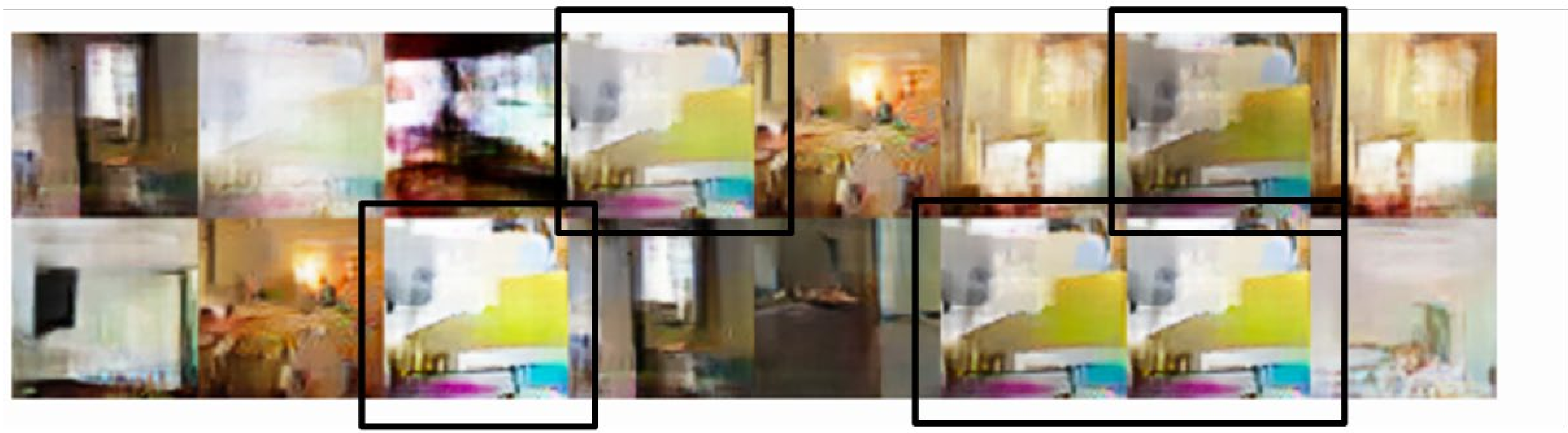
- No robust stopping criteria in practice!

Issues in Practice

- GANs are known to be very difficult to train in practice
- Formulated as min-max objective between two networks
- Optimization can oscillate between solutions (Mode Collapse)
- Generator can collapse to represent part of the training data, and miss another part
- Hard to pick “compatible” architectures between generator and discriminator

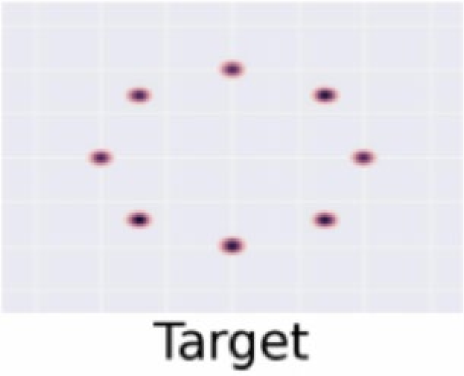
Mode Collapse

- GANs are notorious for suffering from mode collapse
- Intuitively, this refers to the phenomena where the generator of a GAN collapses to one or few samples (dubbed as “modes”)

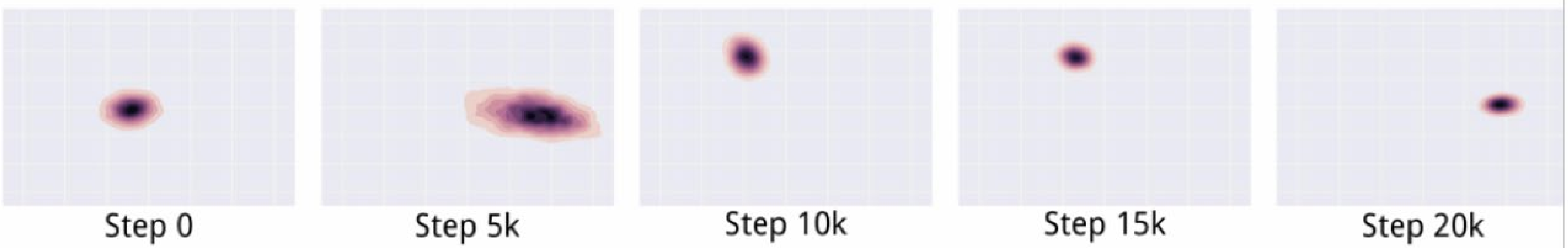


Arjovsky et al., 2017

Mode Collapse



- True distribution is a mixture of Gaussians



Source: Metz et al., 2017

- The generator distribution keeps oscillating between different modes

Mode Collapse

Source: Metz et al., 2017

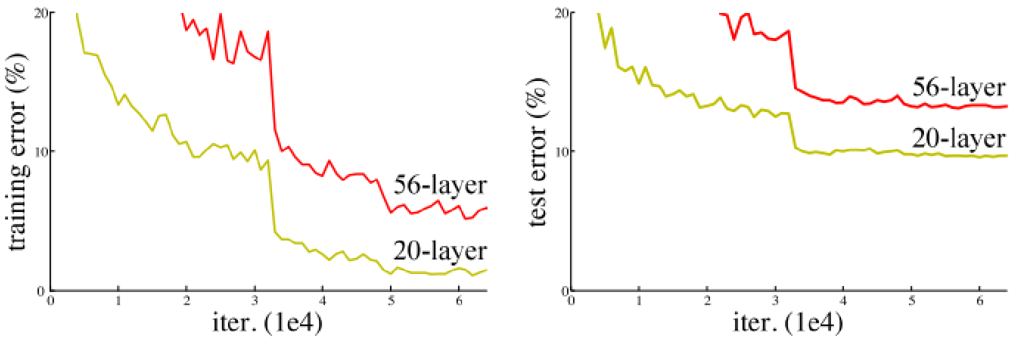
- Fixes to mode collapse are mostly empirically driven: alternate architectures, adding regularization terms, injecting small noise perturbations etc.
- https://github.com/soumith/ganhacks
  How to Train a GAN? Tips and tricks to make GANs work by Soumith Chintala

Dude, what about the cool stuff?

- Style Transfer
- Autoencoders
- GANs
- Modern Architectures

Problems with Deep Networks

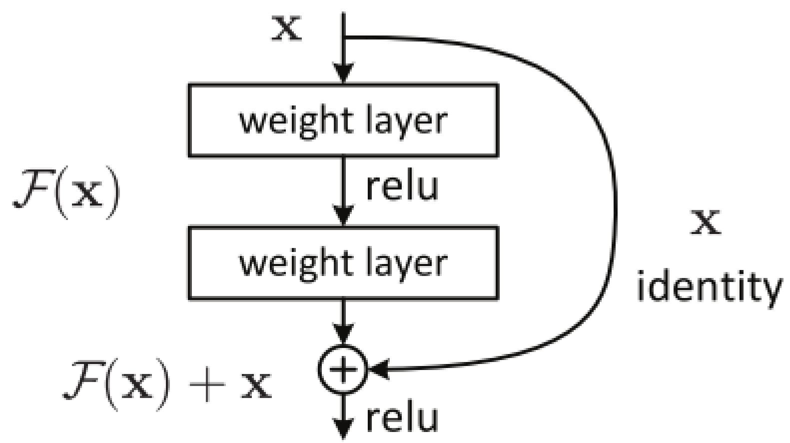
- Generally observed:
  - Deeper networks become increasingly hard to train
  - Gradients vanish or explode
  - Information is required to travel through a long distance
    - Compare to the long term memory problem



- Solution:
  - Ease training of deep network with more direct influence onto the gradient
  - Batch Normalization was an example for that
  - Or auxiliary classifiers

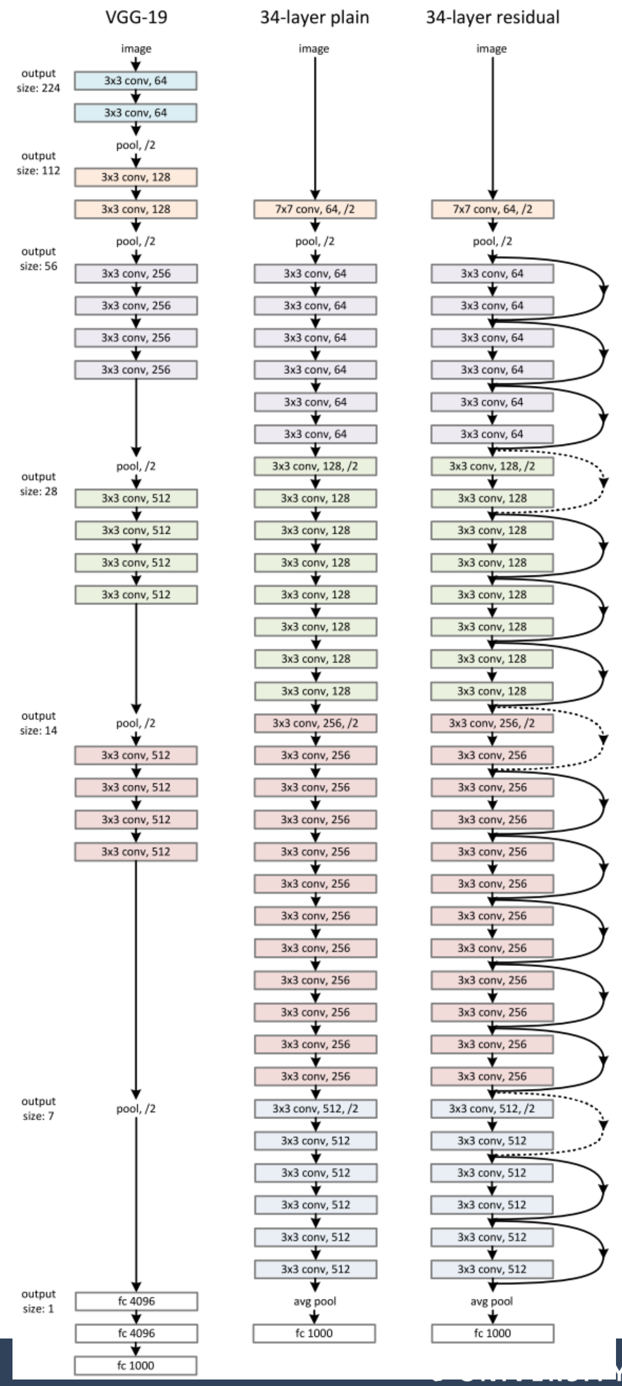
Idea of ResNet

- The network is constructed in blocks
- Introduces the identity shortcut connections

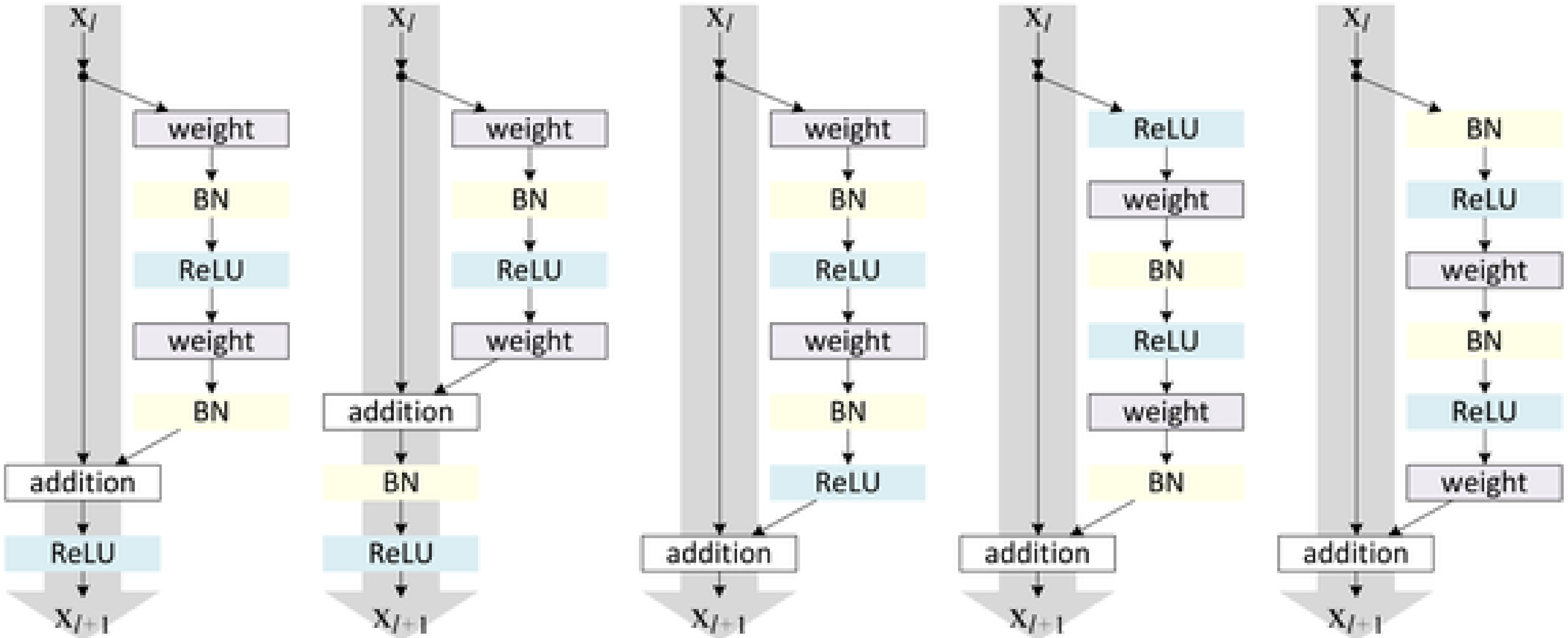


- Observe the similarity to the LSTM

Architecture

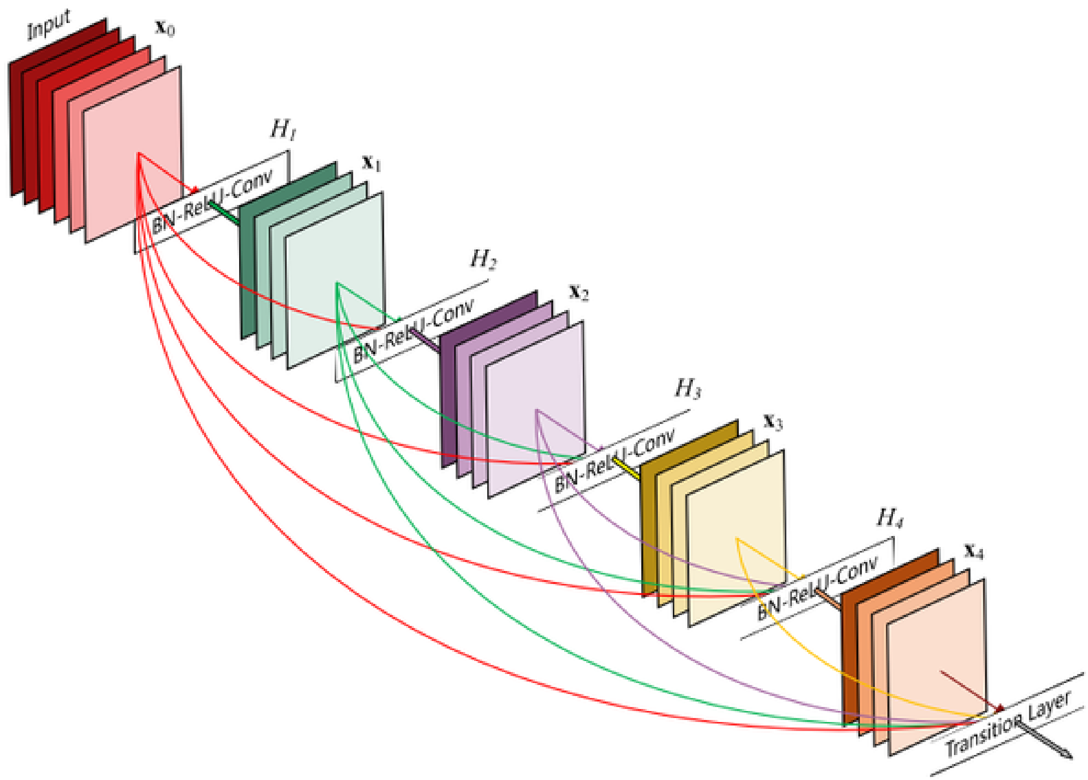


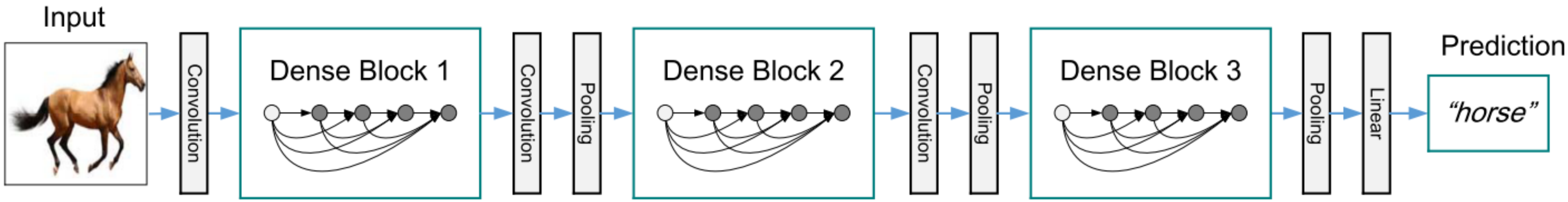
Different Flavors



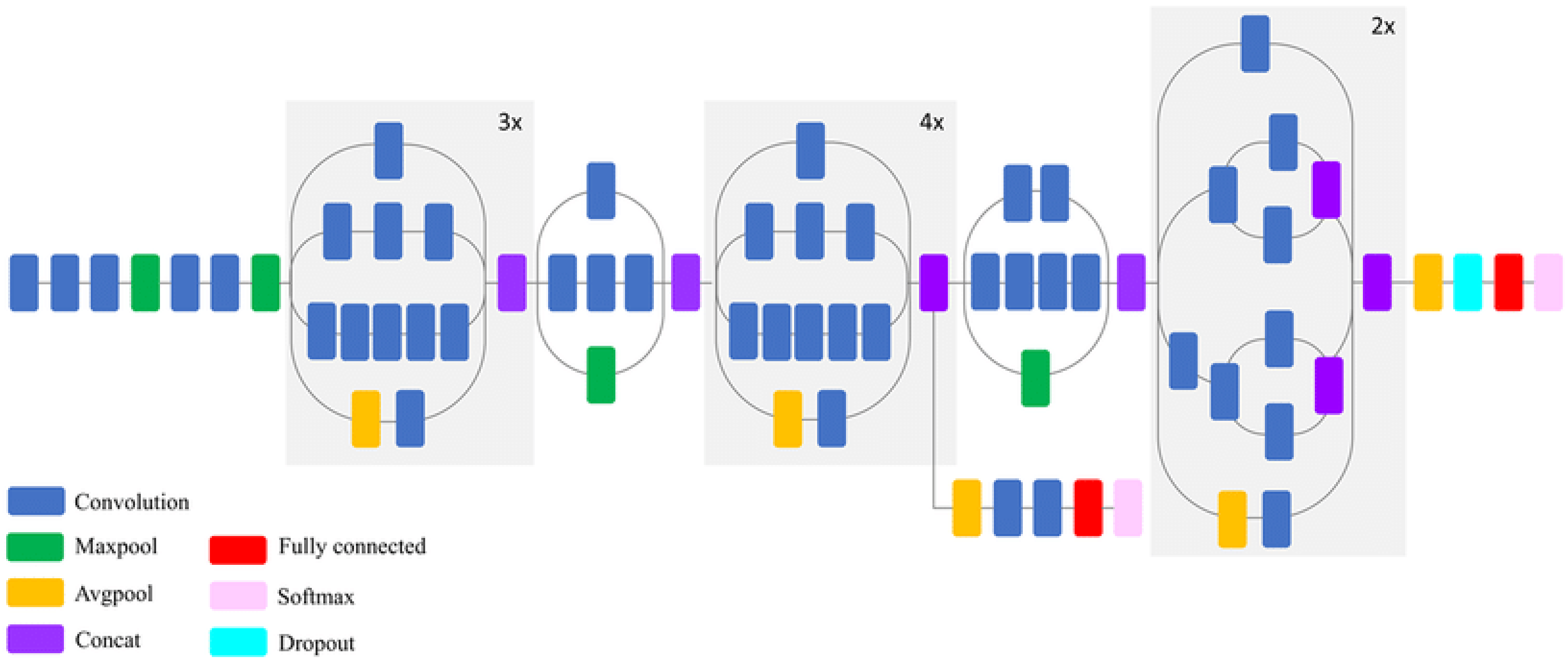
(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

DenseNet


Input
$\mathbf{x}_0$
H_1
BN-ReLU-Conv
$\mathbf{x}_1$
H_2
BN-ReLU-Conv
$\mathbf{x}_2$
H_3
BN-ReLU-Conv
$\mathbf{x}_3$
H_4
BN-ReLU-Conv
$\mathbf{x}_4$
Transition Layer


Input
Convolution
Dense Block 1
Convolution
Pooling
Dense Block 2
Convolution
Pooling
Dense Block 3
Pooling
Linear
Prediction
"horse"

Inception v3



Beauty lies in the eyes of the discriminator