UNSUPERVISED LEARNING /GENERATIVE MODELS

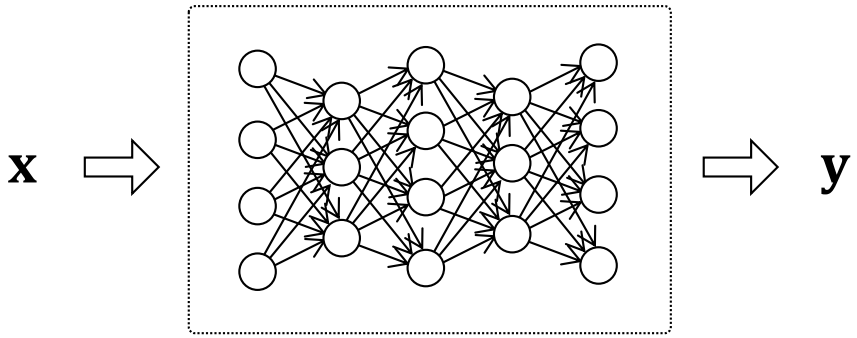
Unsupervised learning / Generative models

- Unsupervised learning (general concepts)
- Autoencoders + Anomaly detection
- Generative Adversarial Networks (GANs)
- Variational Autoencoders

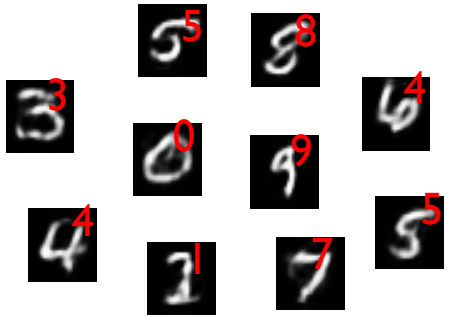
Supervised learning & unsupervised learning

Supervised

- Pairs of input *x* & output *y* are given
- To learn a mapping from *x* to *y so* that *y* can be predicted for a given *x*

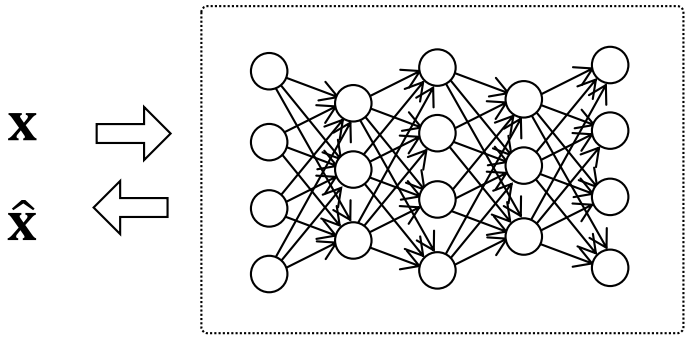


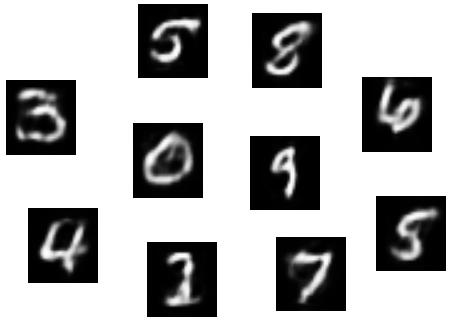
$$\mathbf{y} = f(\mathbf{x}; \theta)$$



Unsupervised

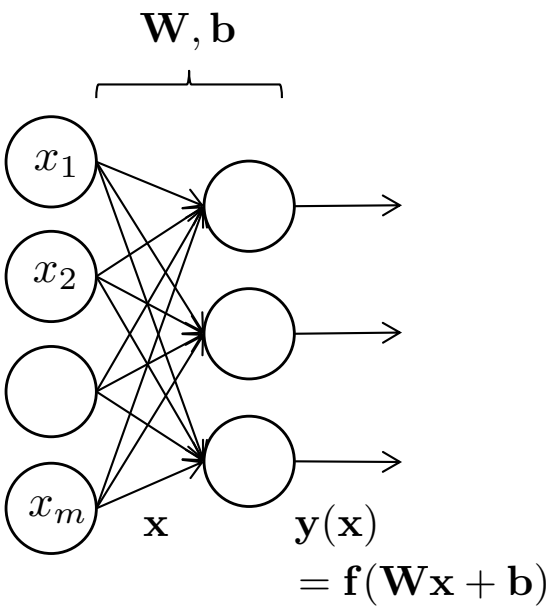
- Only inputs *x*'s are given
- To learn the distribution of *x* so that
  - Anomaly can be detection
  - *x* can be generated

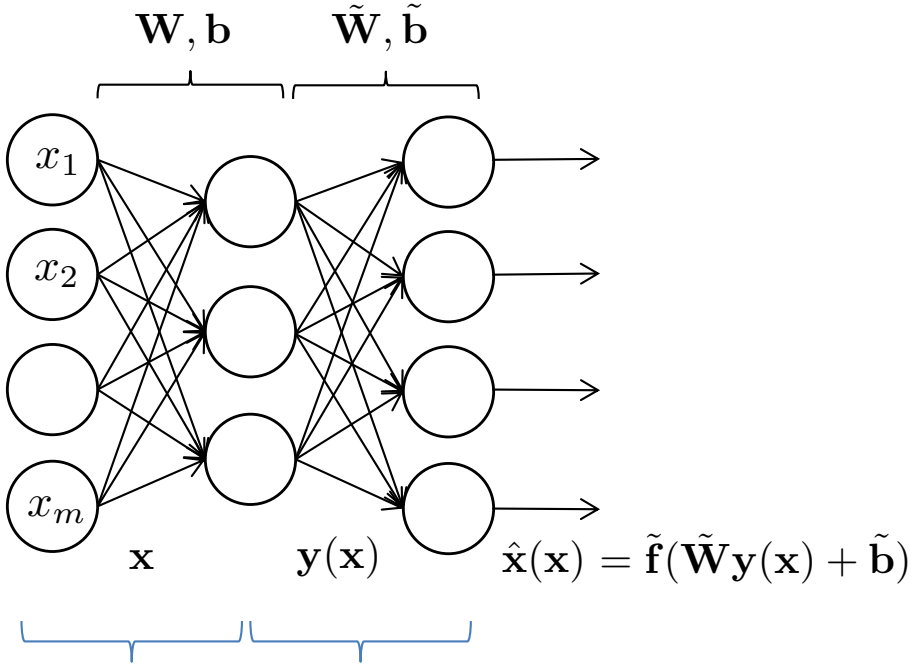




Autoencoders: Structure

- Typically has a structure of folding an FF net w/ *N*(≧1) layers
- The output will be maximally close to an input





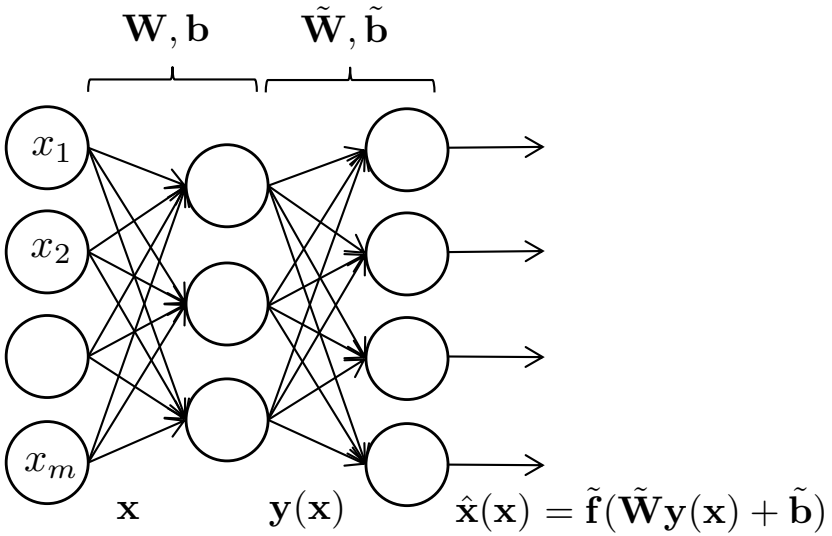
$$\mathbf{y}(\mathbf{x}) = \mathbf{f}(\mathbf{W}\mathbf{x} + \mathbf{b}) \qquad \hat{\mathbf{x}} = \tilde{\mathbf{f}}(\tilde{\mathbf{W}}\mathbf{y} + \tilde{\mathbf{b}})$$

Encoding Decoding

Autoencoders: Training

- If *x* is a continuous vector
  - Activation function of the output layer is an identity mapping
  - Squared loss is employed

$$E(\mathbf{w}) = \sum_{n=1}^{N} \|\mathbf{x}_n - \hat{\mathbf{x}}(\mathbf{x}_n)\|^2$$



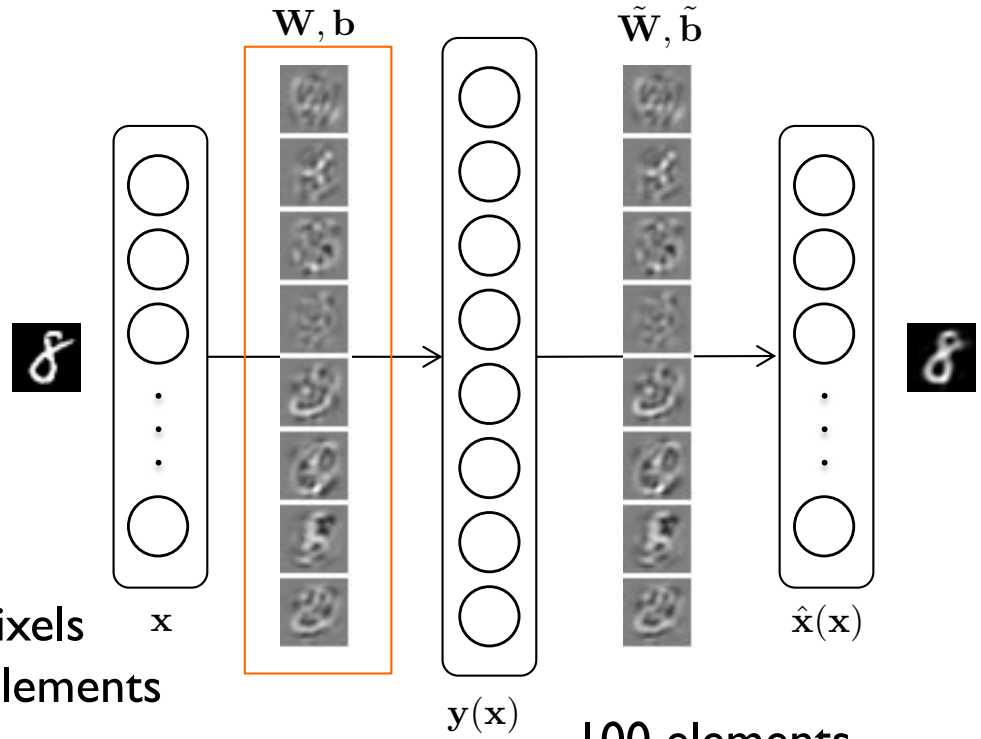
- If *x* is a binary vector
  - ActFunc: logistic function

$$\tilde{f}_i(u) = 1/(1 - \exp(-u))$$

  - Cross-entropy loss is used

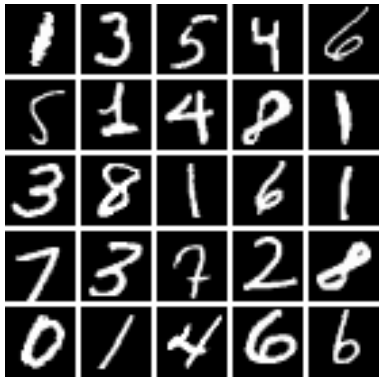
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{i=1}^{D} x_{ni} \log \hat{x_{ni}}(x_{ni}) + (1 - x_{ni}) \log(1 - \hat{x_{ni}}(x_{ni}))$$

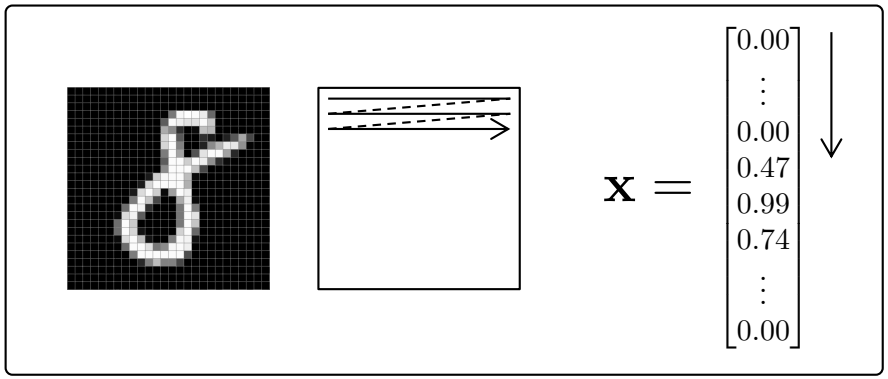
Example: Unsupervised learning of MNIST



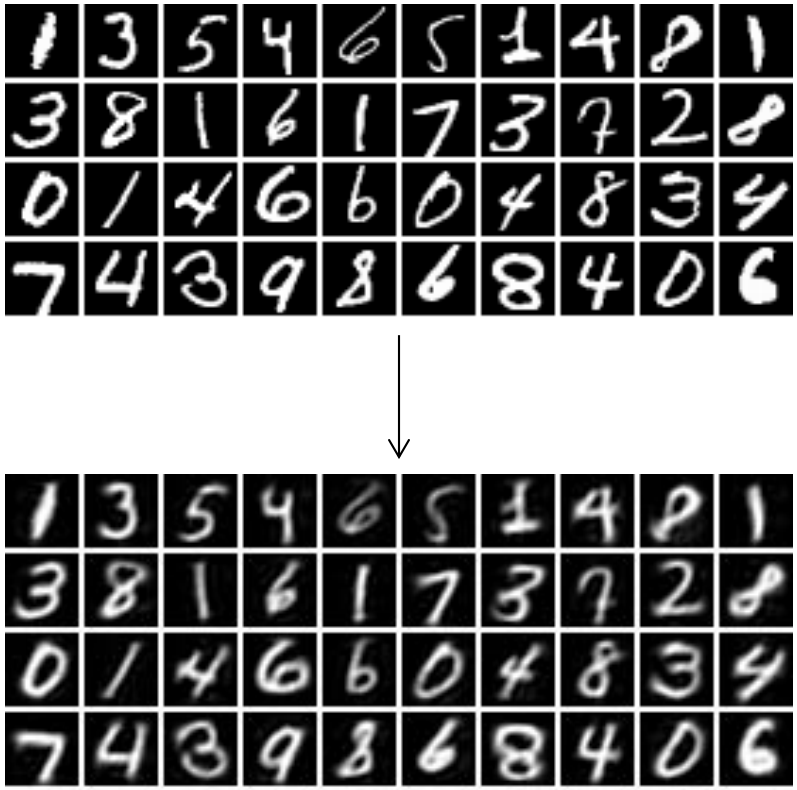
26x26 pixels
= 676 elements

100 elements

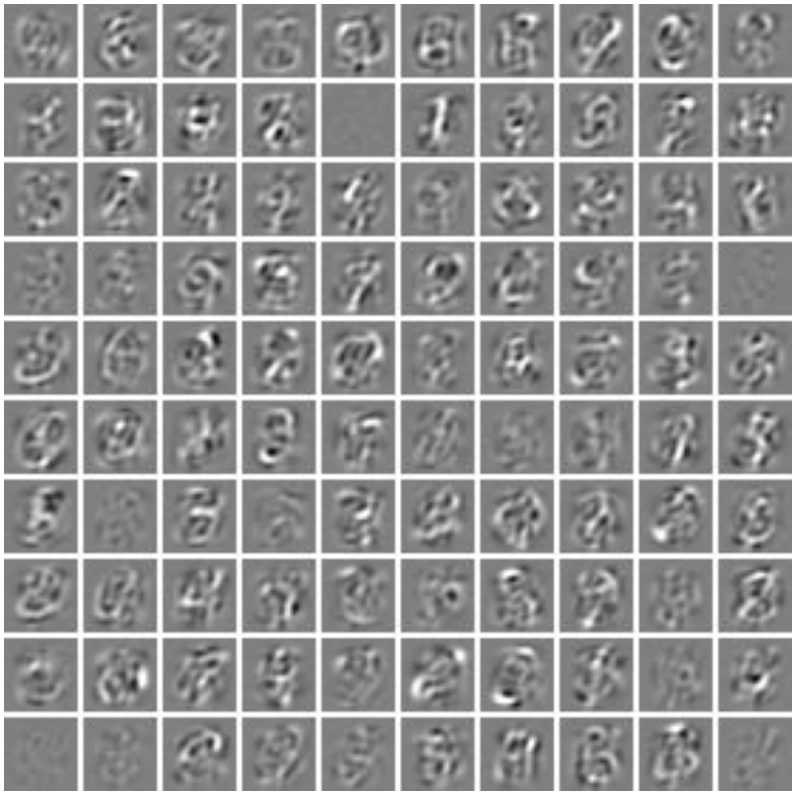




Example: Unsupervised learning of MNIST



Inputs & outputs

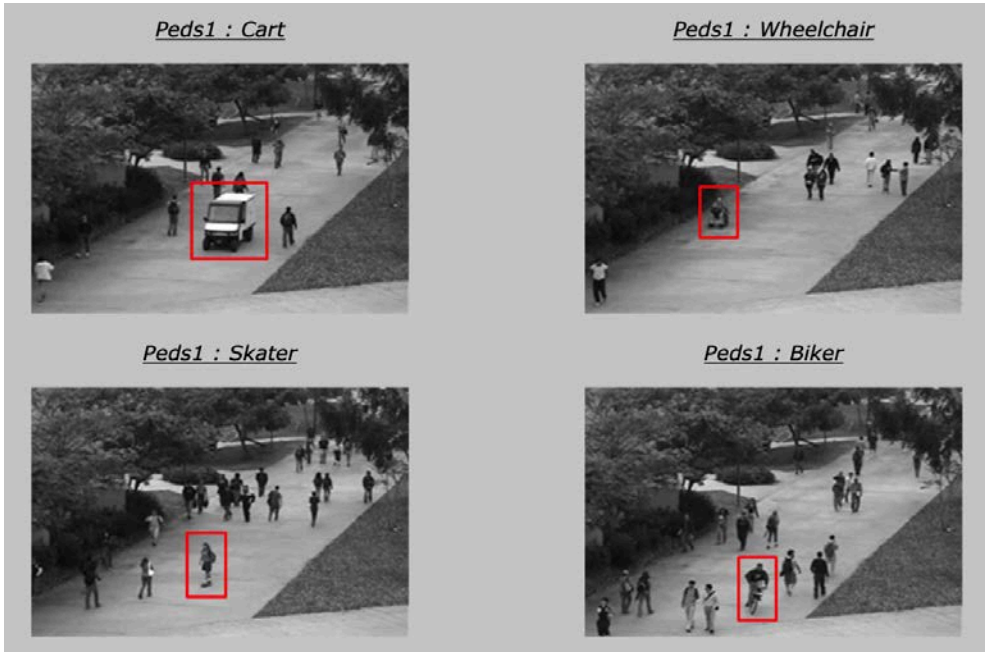


Learned weights (features)

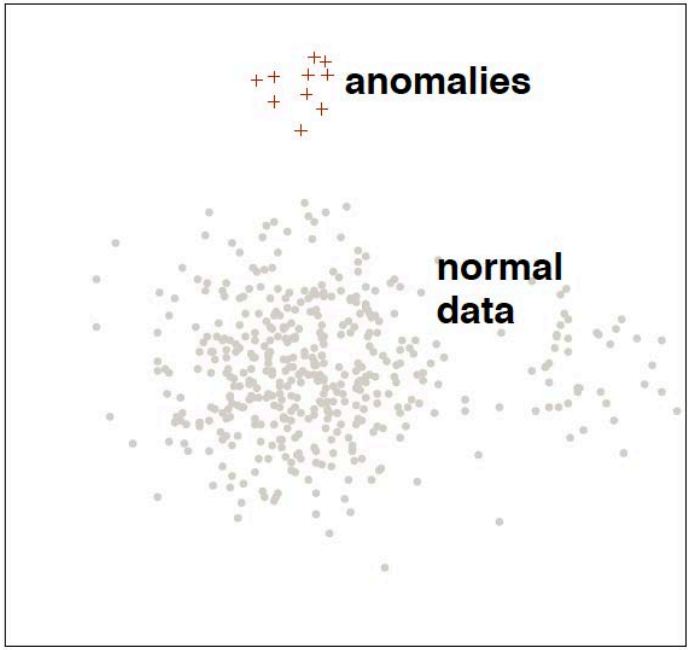
Application to anomaly detection

- Anomaly detection: we want to detect anomalies(=outliers)
  - We have only normal data, e.g., videos from surveillance camera
  - We want to get a good *feature space* like the one below

E.g., anomalous events in videos



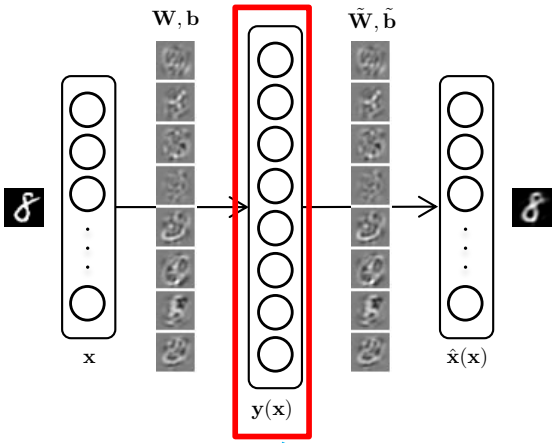
UCSD Ped1 dataset



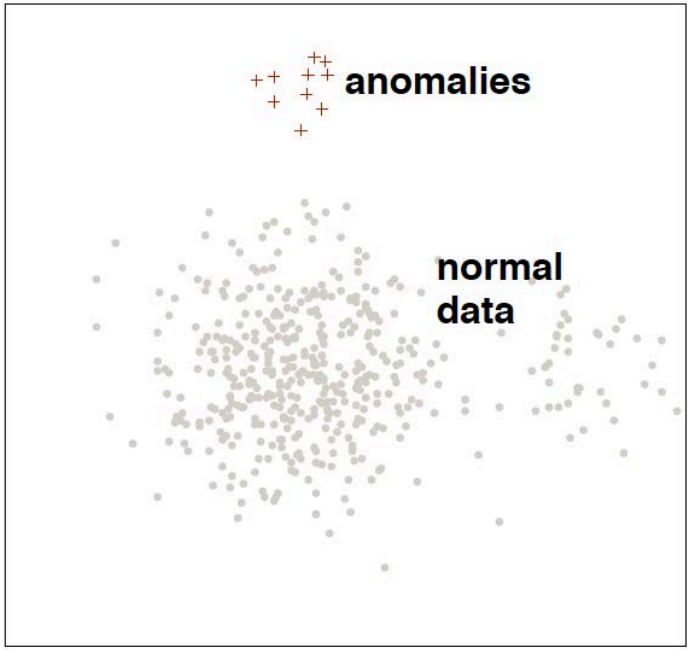
Feature space

Application to anomaly detection

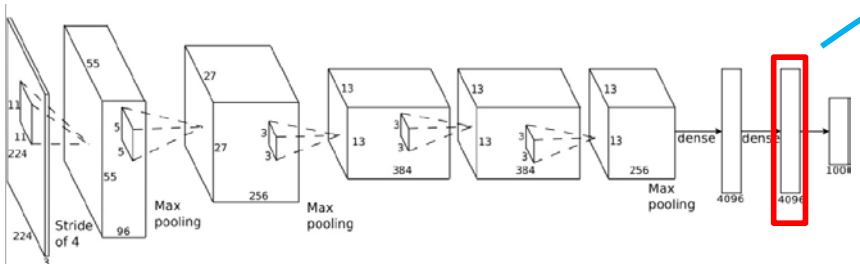
- Anomaly detection: we want to detect anomalies(=outliers)
  - We have only normal data, e.g., videos from surveillance camera
  - We want to get a good *feature space* like the one below



A (compact) representation for an input



Feature space



Application to anomaly detection

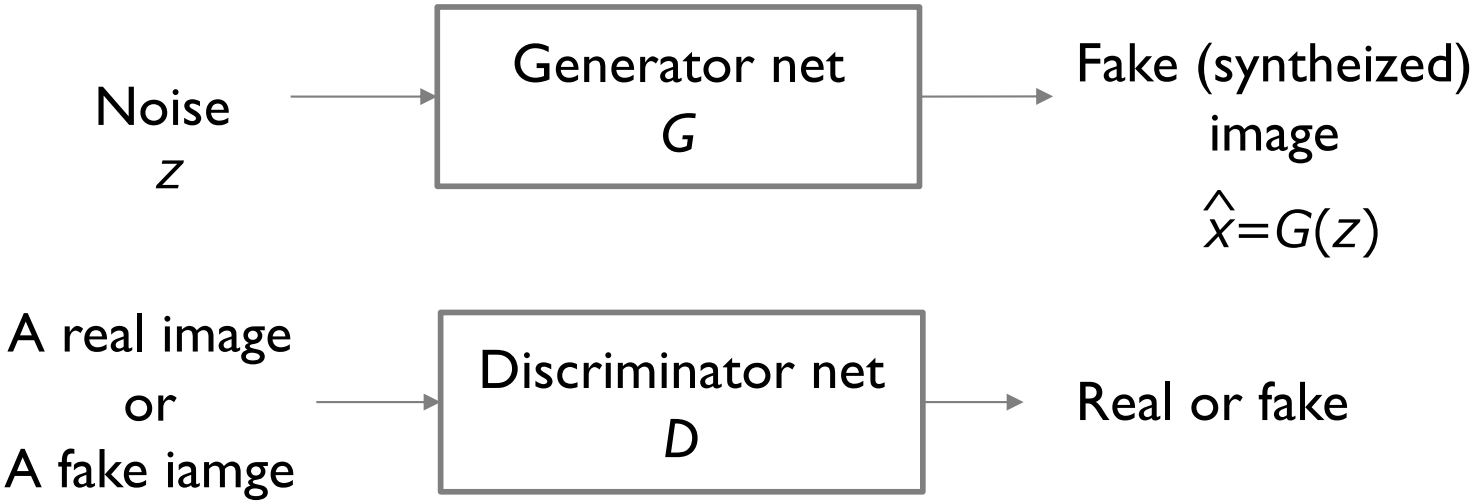
Input

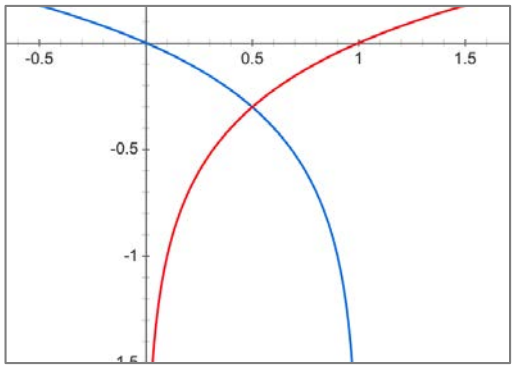
Output

Ravanbakhsh+, Training Adversarial Discriminators for Cross-channel Abnormal Event Detection in Crowds, arXiv2017

GANs (Generative Adversarial Networks): Basics

- Train two networks in an adversarial fashion; Contradictive goals
  - Generator *G* is trained to generate realistic images; to fool D
  - Discriminator *D* is trained to accurately distinguish real and fake; not to be fooled





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

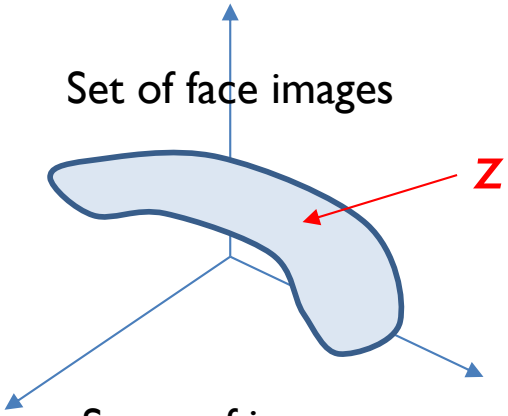
$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]$$

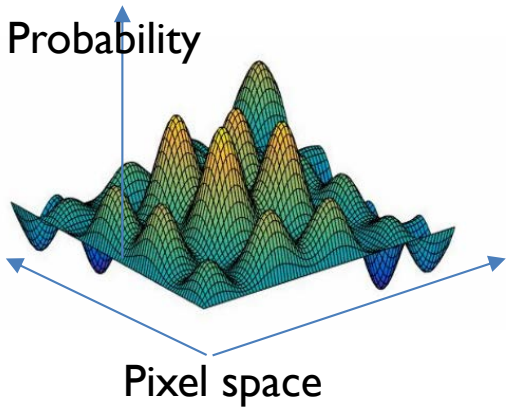
A goal of GANs

- To learn the distribution of data

Real

Fake(Synth)

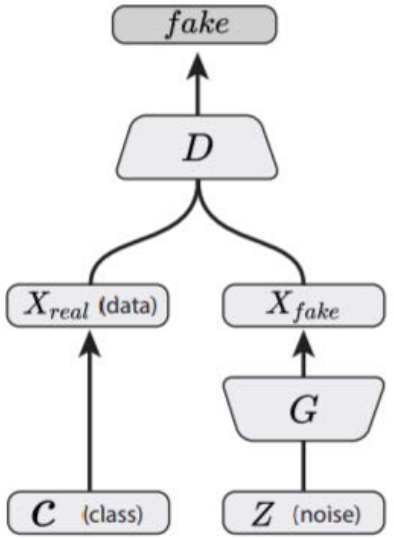




https://drive.google.com/open?id=1tvhZeRGidszuDXCHfj1YmstWGJQebCY3

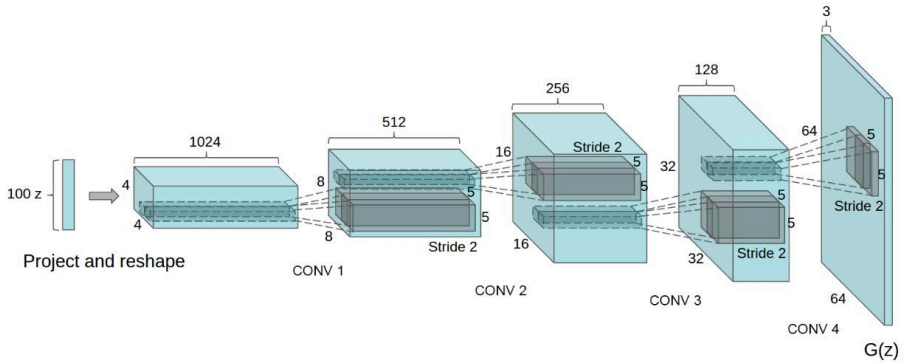
GANs for image generation

- The most fundamental GANs
  - Generates various images from random noise *z*
  - *z* is regarded as a `code' for the generated image; also called a latent variable



$$\min_{\theta} \max_{\phi} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim P_r(\boldsymbol{x})}[\log(D_\phi(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\boldsymbol{z}\sim P(\boldsymbol{z})}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))]$$

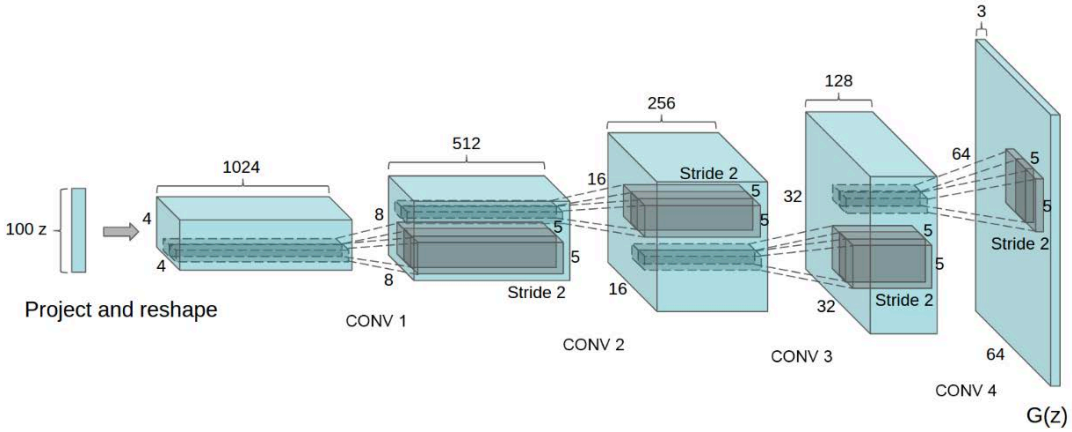
- DCGAN: Deep Convolutional GAN [Radford+2016]
  - *G* is a CNN

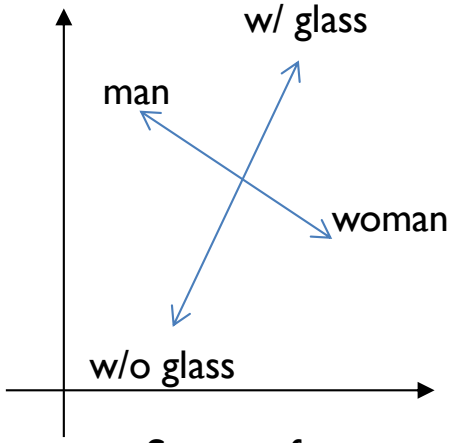


Space of the latent variable *z*

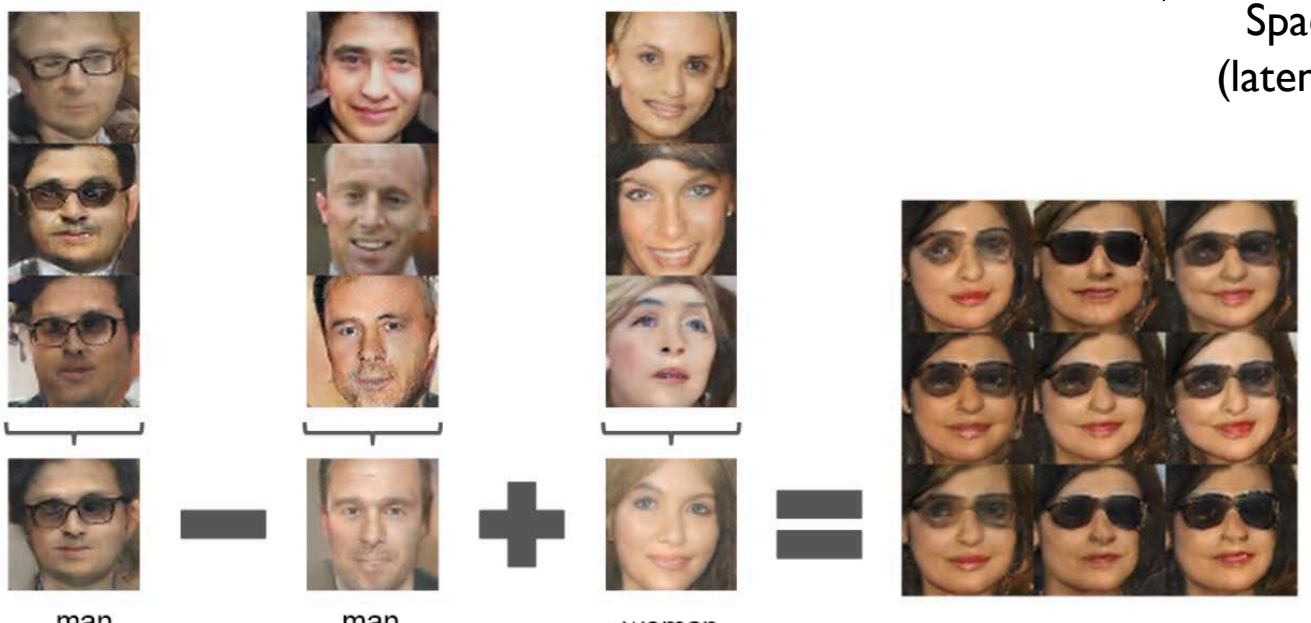
Radford+, UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS, ICLR2016

- *z* retains the semantic info about the image





Space of *z*
(latent space)



Example: GAN modeling 3D shape

Wu+, Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling, NIPS2016

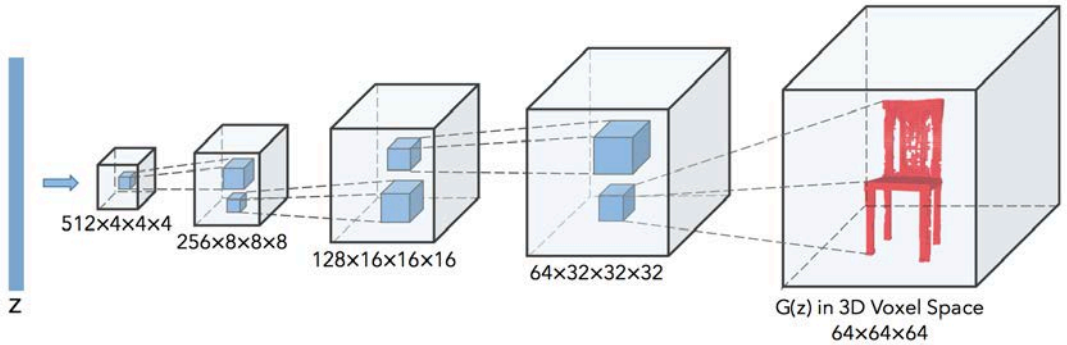
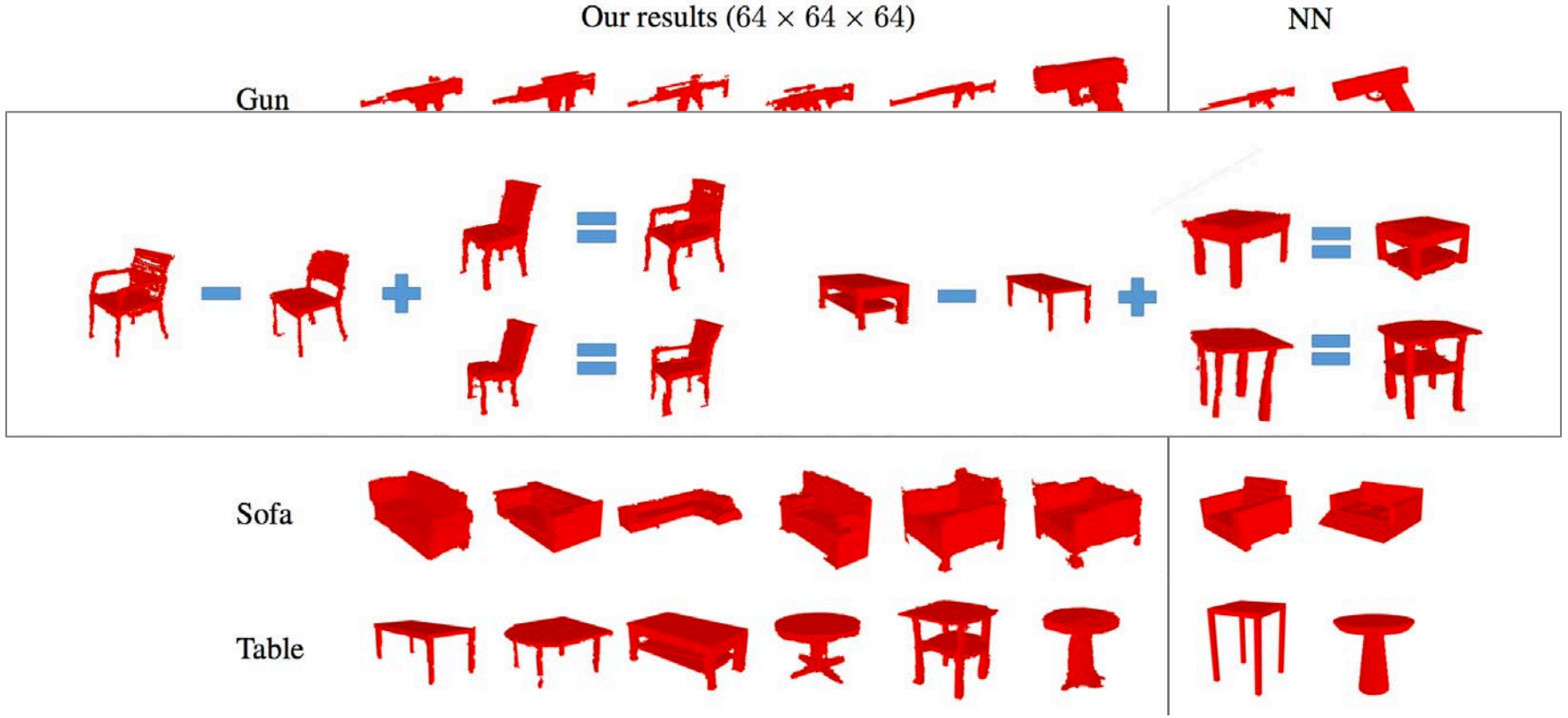


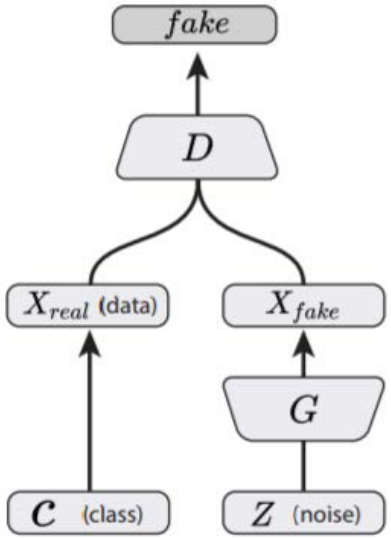
Figure 1: The generator in 3D-GAN. The discriminator mostly mirrors the generator.



Conditional GAN

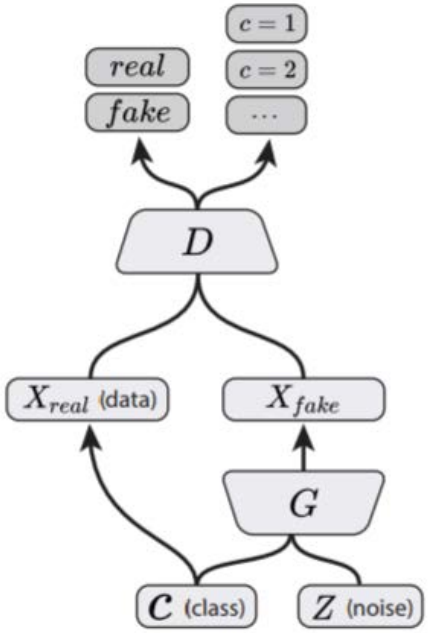
Unconditional GAN

(Goodfellow, *et. al.* 2014)



(Class-)Conditional GAN

(Odena, *et. al.* 2016)

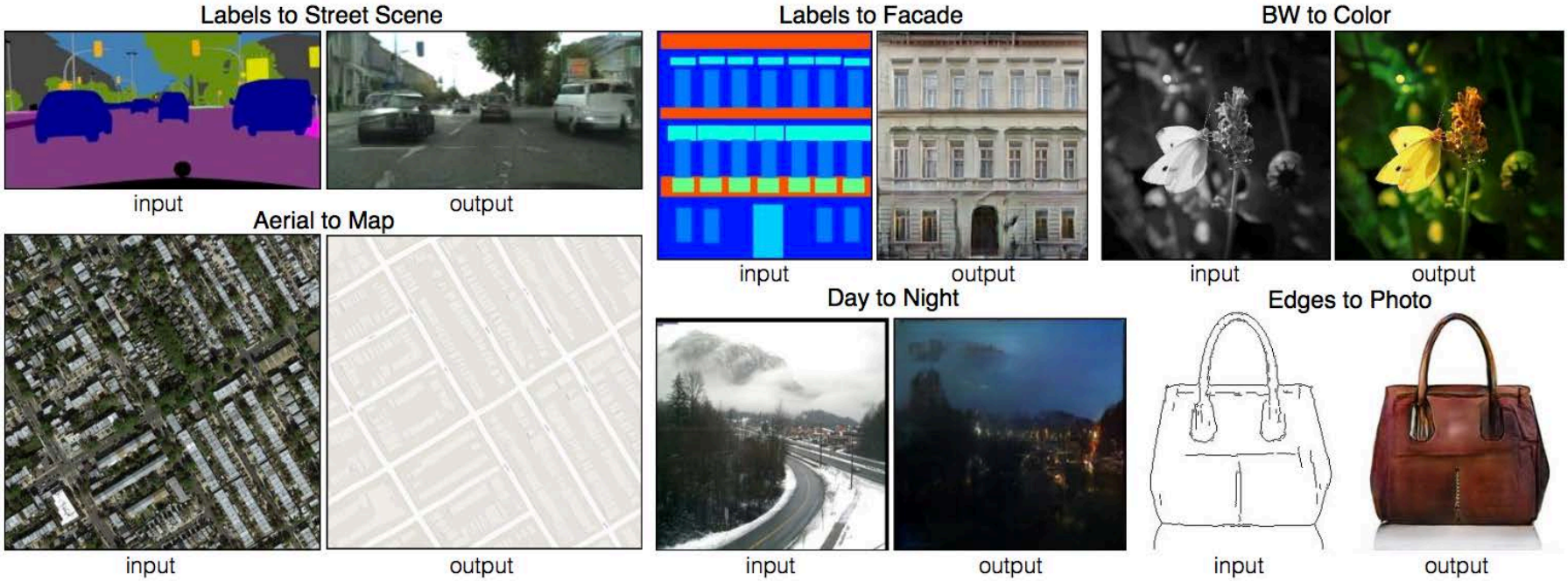


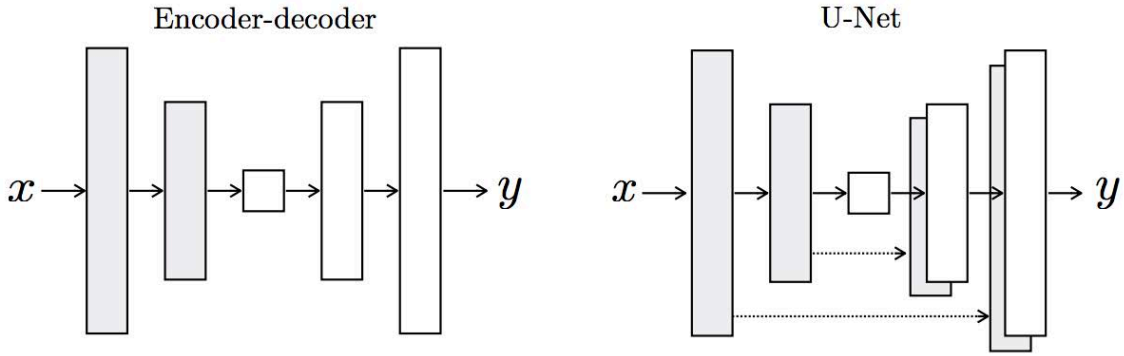
$$\min_{\theta} \max_{\phi} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim P_r(\boldsymbol{x})}[\log(D_\phi(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\boldsymbol{z}\sim P(\boldsymbol{z})}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))] \qquad -\frac{1}{n}\sum_{k=1}^{K} \mathbb{1}(y = k)\log\frac{e^{\boldsymbol{w}^{(k)}\boldsymbol{x}}}{\sum_{i=1}^{K} e^{\boldsymbol{w}^{(i)}\boldsymbol{x}}},$$

Conditional GAN：pix2pix

Isola+, Image-to-Image Translation with Conditional Adversarial Networks, CVPR2017

- Image-to-image translation

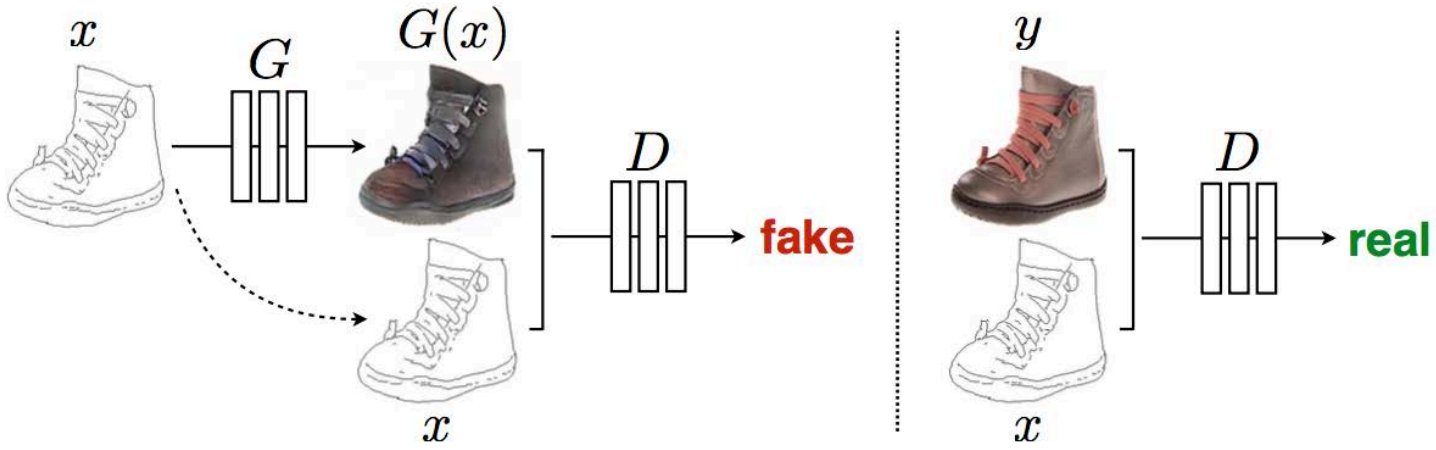




Conditional GAN：pix2pix

Isola+, Image-to-Image Translation with Conditional Adversarial Networks, CVPR2017

- *x* is input not only to *G* but to *D*



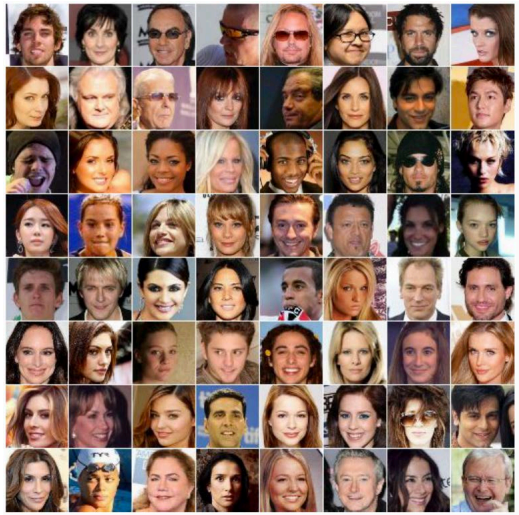
$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]$$

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

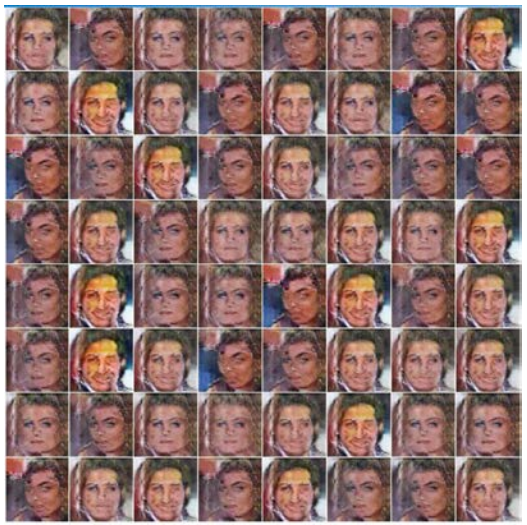
Training GANs

- Training GANs is notoriously hard
  - Learning speed of *G* and *D* should be similar during training
  - A huge number of tricks (that are claimed to be effective) have been published
  - Hard to evaluate the success/failure of the training
    - Only humans can judge the goodness of the outcomes in many cases



Data (real)

"mode collapse"



Synthesized

Many trikcs --- From Denton et al., How to Train a GAN, ICCV2017 Tutorial

#1: Normalize the inputs

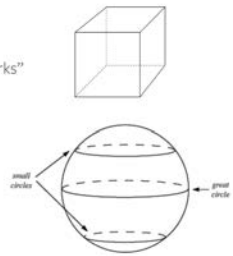
- normalize the images between -1 and 1
- Tanh as the last layer of the generator output
  - or some kind of bounds normalization

#2: Modified loss function (classic GAN)

- In papers people write min (log 1-D), but in practice folks practically use max log D
  - because the first formulation has vanishing gradients early on
  - Goodfellow et. al (20**LOT OF NEW LOSS FORMULATIONS**
- In practice:
  - Flip labels when training generator: real = fake, fake = real

#3: Use spherical z

- interpolation via great circle
- Tom White "Sampling Generative Networks"
  - https://arxiv.org/abs/1609.04468



#4: BatchNorm

- different batches for real and fake
- when batchnorm is not an option use instance norm

#5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers
- LeakyReLU (both G and D)
- Downsampling: Average Pooling, Conv2d + stride
- Upsampling: PixelShuffle, ConvTranspose2d + stride
  - PixelShuffle: https://arxiv.org/abs/1609.05158

#6: Soft and Noisy Labels

- Label Smoothing
- making the labels the noisy a bit for the discriminator, sometimes
  - Salimans et. al. 2016

#7: Architectures: DCGANs / Hybrids

- DCGAN when you can
- if you cant use DCGANs and no model is stable,
- use a hybrid model : KL + GAN or VAE + GAN
- ResNets from WGAN-gp also work pretty well (but are very slow)
  - https://github.com/igul222/improved_wgan_training
- Width matters more than Depth

#9: Optimizer: ADAM

- optim.Adam rules!
  - See Radford et. al. 2015
- [MMathieu] Use SGD for discriminator and ADAM for generator

#10: Use Gradient Penalty

- Regularize the norm of the gradients
  - multiple theories on why this is useful (WGAN-GP, DRAGAN, Stabilizing GANs by Regularization etc.)

#12: If you have labels, use them

- if you have labels available, training the discriminator to also classify the samples: auxillary GANs

#13: Add noise to inputs, decay over time

- Add some artificial noise to inputs to D (Arjovsky et. al., Huszar, 2016)
  - http://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/
  - https://openreview.net/forum?id=Hk4_qw5xe
- adding gaussian noise to every layer of generator (Zhao et. al. EBGAN)
- Improved GANs: OpenAI code also has it (commented out)

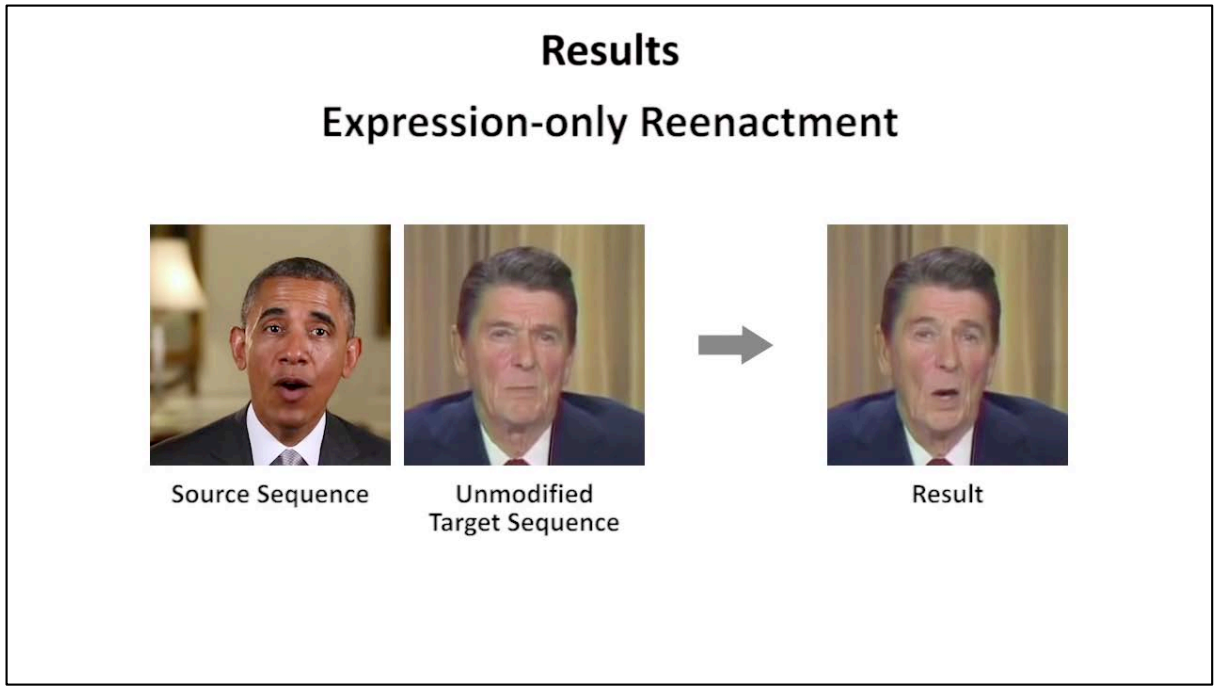
#14: Train discriminator more

- especially when you have noise
- hard to find a schedule of number of D iterations vs G iterations
- WGAN/WGAN-gp papers suggest 5x D iterations per G iteration

Well-established? tricks

- Spectral normalization
  - Miyato+, Spectral Normalization for Generative Adversarial Networks, ICLR2018
- Two-timescale update rule
  - Heusel+, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, NIPS2017
- Gradient penalty
  - Gulrajani+, Improved Training of Wasserstein GANs, NIPS2017
  - Fedus+, MANY PATHS TO EQUILIBRIUM: GANS DO NOT NEED TO DECREASE A DIVERGENCE AT EVERY STEP, ICLR2018

GANs can now generate very natural images



Kim+, Deep Video Portrait, SIGGRAPH2018

and a specific static, but otherwise general scene background. Our rendering-to-video translation network follows an encoder–decoder architecture and is trained in an adversarial manner based on a discriminator that is jointly trained. In the following, we explain

T: rendering-to-video translation net

$$\mathbf{T}^* = \arg\min_{\mathbf{T}} \max_{\mathbf{D}} E_{\text{cGAN}}(\mathbf{T}, \mathbf{D}) + \lambda E_{\ell_1}(\mathbf{T})$$

$$E_{\text{GAN}}(\mathbf{T}, \mathbf{D}) = \mathbb{E}_{\mathbf{X},\mathbf{Y}}\left[\log \mathbf{D}(\mathbf{X}, \mathbf{Y})\right] + \mathbb{E}_{\mathbf{X}}\left[\log\left(1 - \mathbf{D}(\mathbf{X}, \mathbf{T}(\mathbf{X}))\right)\right]$$

[Karras+@nVIDIA arXiv Feb.2019]

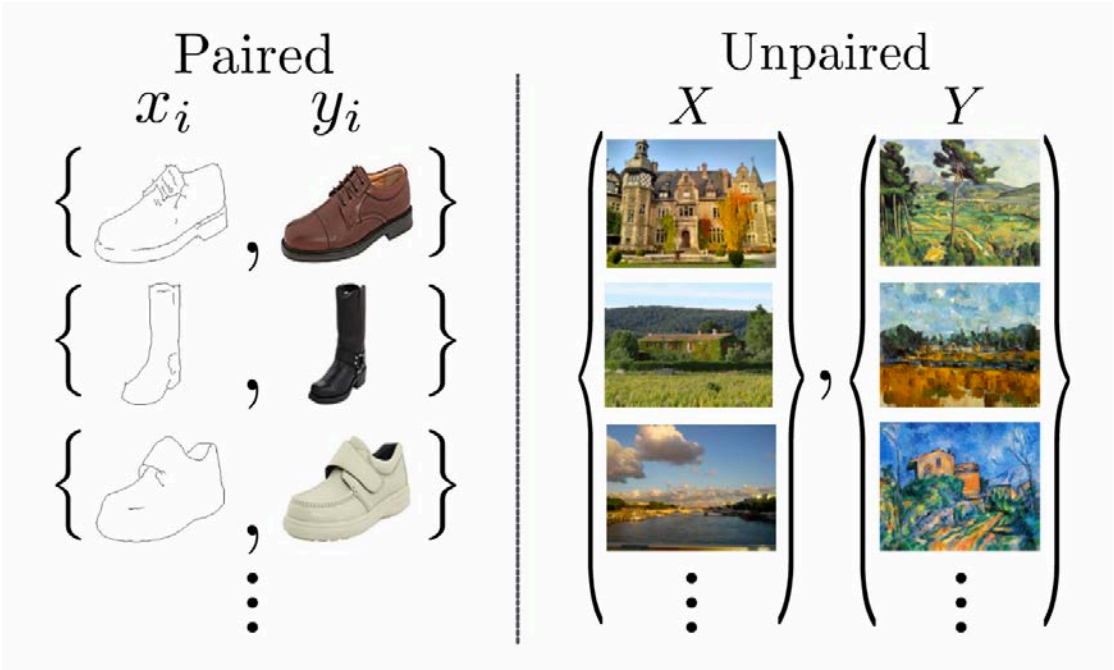
Image inpainting

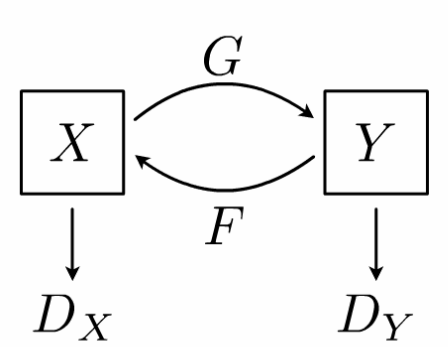
Iizuka+, Globally and Locally Consistent Image Completion, SIGGRAPH2017

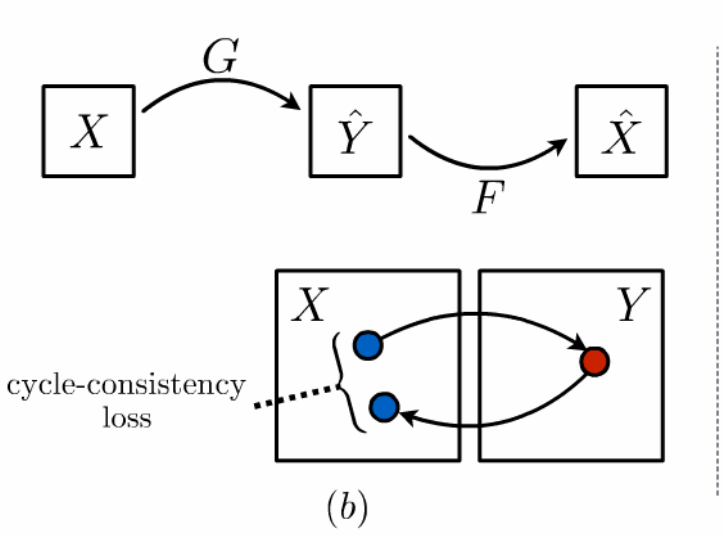
CycleGAN

Zhu+, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

- pix2pix: needs paired images
- CycleGAN: need only unpaired images

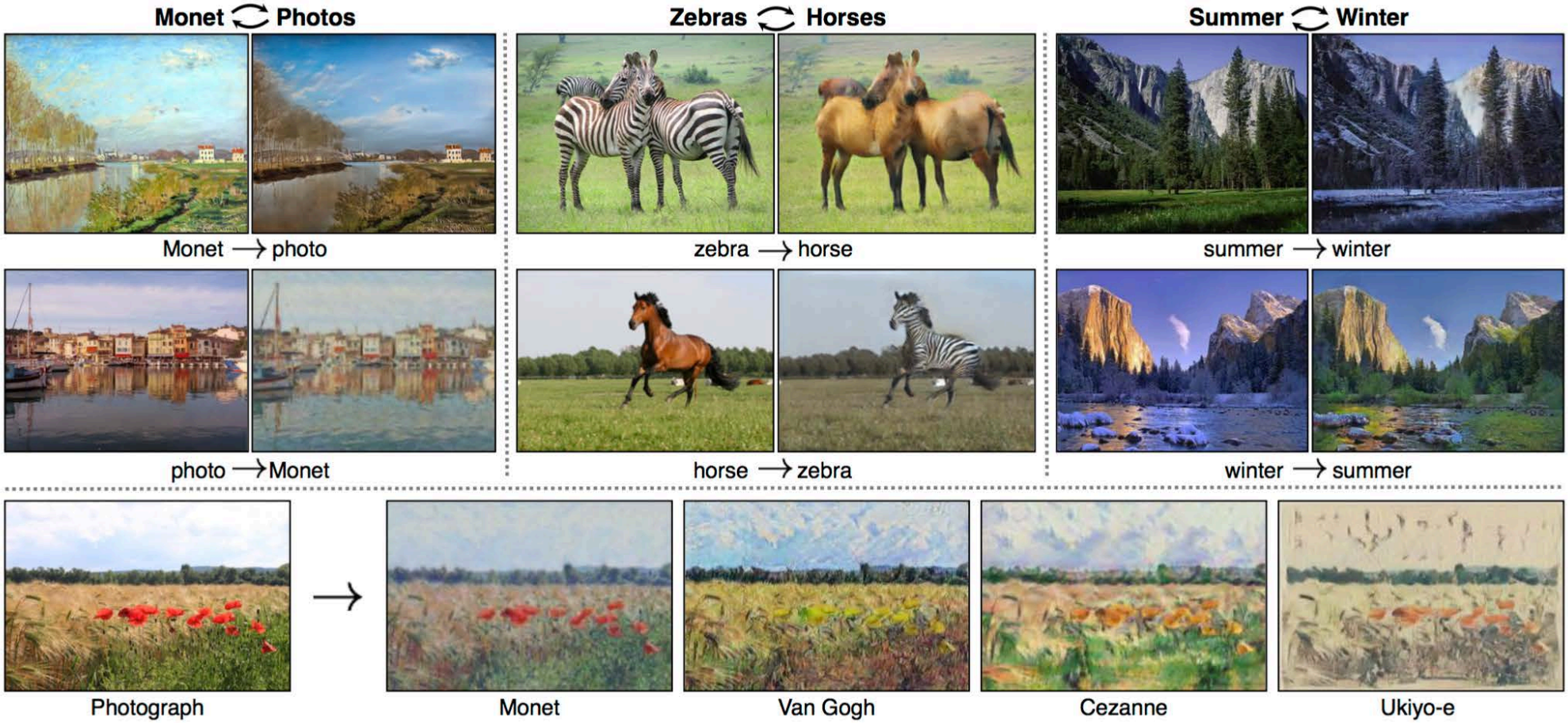






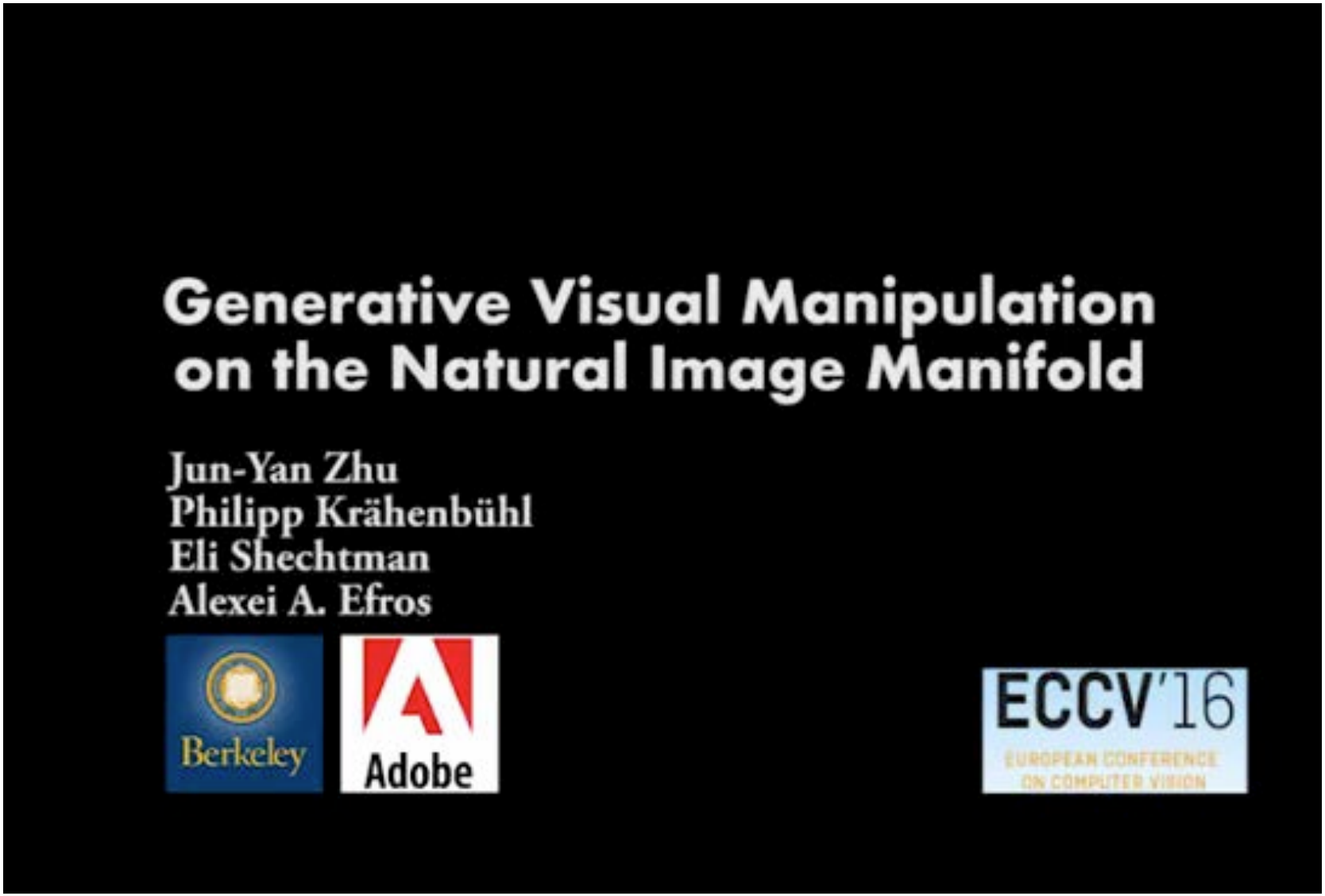
CycleGAN

Zhu+, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017



Manipulation of images

Zhu+, Generative Visual Manipulation on the Natural Image Manifold, ECCV2016



Assignments 3

- Mission: Analyze the latent space learned by GANs on MNIST dataset
  - Send your submission to okatani@vision.is.tohoku.ac.jp by Nov. 25

- Minimum requirements: https://drive.google.com/open?id=1tvhZeRGidszuDXCHfj1YmstWGJQebCY3
  - Train GANs using the provided sample code
  - Find two latent variables z_1 and z_2 generating two different digits; they can be any digits if they are different, e.g., 0 & 1, 3 & 9, etc.
  - Generate *intermediate* images of the two digits by specifying z's interpolating z_1 and z_2.
    - $z = a\, z_1 + (1-a)\, z_2$, where $a = 0, 0.1, 0.2, \ldots, 0.9, 1.0$

- Optional (5% additional score will be given if you accomplish this):
  - Visualize how each digit class occupies the learned latent space
    1. Sample noise vectors z's ~ 1,000 - 10,000
    2. Generate digits images from them using G
    3. Classify each of them into the 10 classes using a classifier you trained previously
    4. Use a dimensionality reduction method (e.g., tSNE) and plot z's with different colors corresponding to the classes

https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

Variational autoencoder (VAE)

- AE w/ a stochastic behavior

- Minimize $L(\boldsymbol{\Theta}_E, \boldsymbol{\Theta}_D; \mathbf{x}, \boldsymbol{\epsilon}) = L_D(\boldsymbol{\Theta}_D) + \lambda L_E(\boldsymbol{\Theta}_E)$

$$L_D \equiv \|\mathbf{x} - \mathbf{f}\|_2^2 \qquad L_E \equiv \mathrm{KL}\left[N(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \middle| N(\mathbf{0}, \mathbf{I})\right] = \frac{1}{2}\sum_{k=1}^{K}\left(\mu_k^2 + \sigma_k^2 - 1 - \log(\sigma_k^2)\right)$$