

Generative Adversarial Networks

EECS 6894

Deep Learning for Computer Vision, Speech, and Language

November 27, 2018

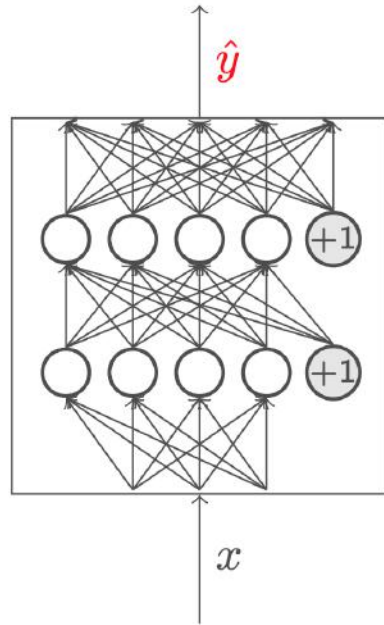
Andrew Kae
andrewkae@oath.com

Outline

- Generative vs Discriminative
- Generative Adversarial Networks
- Applications
 - Realistic Image Generation
 - Domain Adaptation
 - Super resolution
 - Style Transfer
 - Disentangling Factors in Data
 - ...Surprise

Generative vs Discriminative

- Deep learning has mainly focused on discriminative models (e.g. neural nets)



Prediction

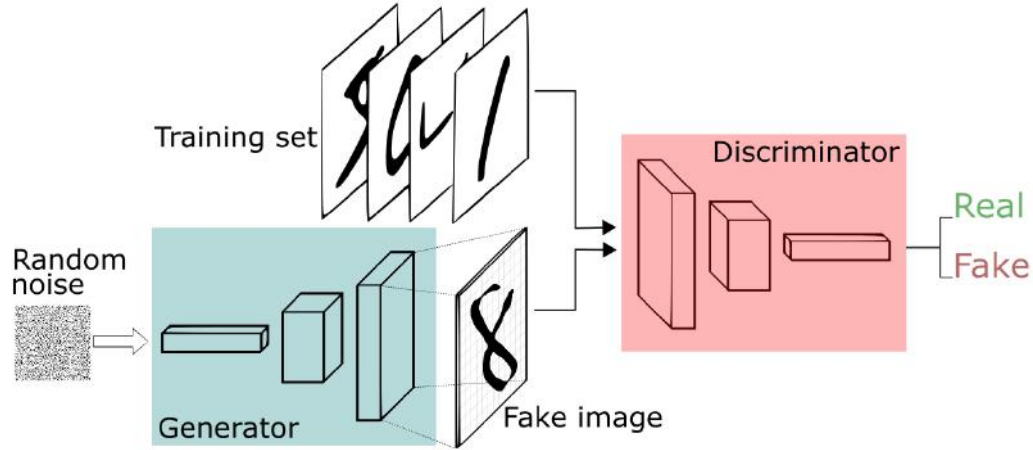
Network with weights θ

Input

Generative vs Discriminative

- Discriminative: Learns posterior distribution $p(y|x)$ directly
 - Only interested in learning the decision boundary
 - Examples: Neural nets, SVMs, Logistic Regression
- Generative: Learns joint distribution $p(x, y)$
 - Learn how the data is generated
 - Can convert to $p(y|x)$ using Bayes Rule
 - Examples: Naive Bayes, HMM, GMM

Generative Adversarial Nets [\[Goodfellow et al 2014\]](#)



Goal: Learn to generate realistic samples from given distribution

Optimization: Minimax game between Generator (G) and Discriminator (D)

Generative Adversarial Nets [\[Goodfellow et al 2014\]](#)

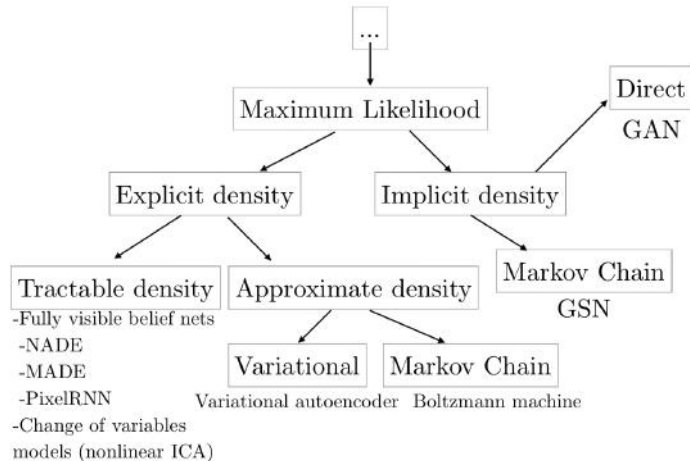
- Define
 - Noise $z \sim p(z)$
 - Generator $G(z)$
 - Samples x_1, \dots, x_n
 - Discriminator $D(x)$
- Optimize: $\min_G \max_D \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))]$
 - D: $D(\text{real}) = 1$ and $D(\text{fake}) = 0$
 - G: $D(\text{fake}) = 1$

GAN Optimization

$$\min_G \max_D \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))]$$

- Ideally after training
 - Data distribution and G converge (Nash equilibrium)
 - D is unable to distinguish between real and fake, i.e. $D(x) = 1/2$
- In practice may be easier to maximize $\log(D(G(z)))$ wrt to G
- Iterative updates between D,G
- In the end
 - Easy to sample from $G(z)$
 - No explicit formulation for $P(x)$

How do GANs relate to other generative models? [\[link\]](#)



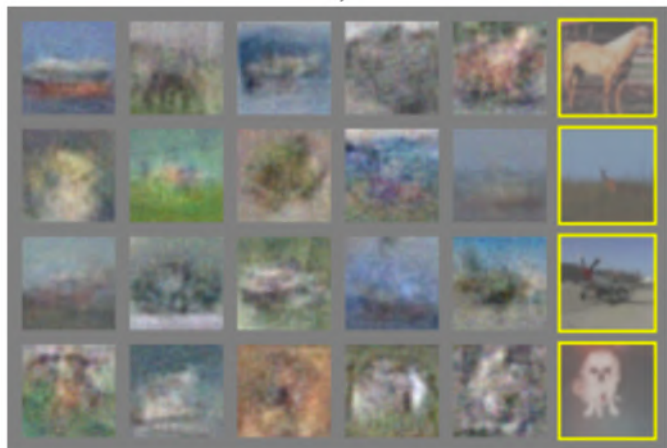
- Some key differences:
 - No explicit representation for $p(x; \theta)$
 - No markov chain needed
 - GANs produce (subjectively) better samples than other models

GAN Code

```
1 G_solver = optim.Adam(G_params, lr=1e-3)
2 D_solver = optim.Adam(D_params, lr=1e-3)
3
4 ones_label = Variable(torch.ones(mb_size))
5 zeros_label = Variable(torch.zeros(mb_size))
6
7 for it in range(100000):
8     # Sample data
9     z = Variable(torch.randn(mb_size, Z_dim))
10    X, _ = mnist.train.next_batch(mb_size)
11    X = Variable(torch.from_numpy(X))
12
13    # Discriminator forward-loss-backward-update
14    G_sample = G(z)
15    D_real = D(X)
16    D_fake = D(G_sample)
17
18    D_loss_real = nn.binary_cross_entropy(D_real, ones_label)
19    D_loss_fake = nn.binary_cross_entropy(D_fake, zeros_label)
20    D_loss = D_loss_real + D_loss_fake
21
22    D_loss.backward()
23    D_solver.step()
24
25    # Generator forward-loss-backward-update
26    G_loss = nn.binary_cross_entropy(D_fake, ones_label)
27
28    G_loss.backward()
29    G_solver.step()
```

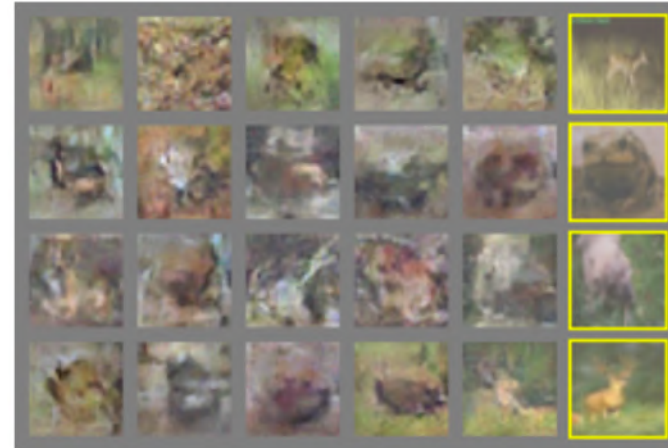
Generated

Real



Generated

Real

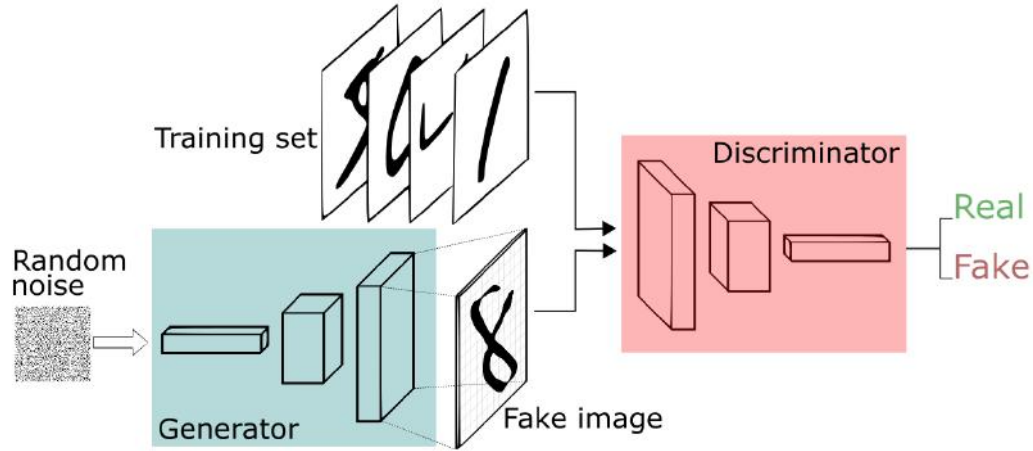


Difficulties in Training GANs

- Model doesn't converge
 - Never reaches Nash equilibrium (i.e. no further changes to G or D)
 - G,D parameters may oscillate
 - Uneven progress between G,D
 - May need to let G train a few iterations and not update D
- Mode collapse (aka Helvetica scenario)
 - G will only generate samples (even one sample) from a single mode
 - Restart training
- Samples may lack global structure
 - i.e. some generated faces will have 3 eyes
- Some of these problems are addressed in Wasserstein GAN ([WGAN](#))

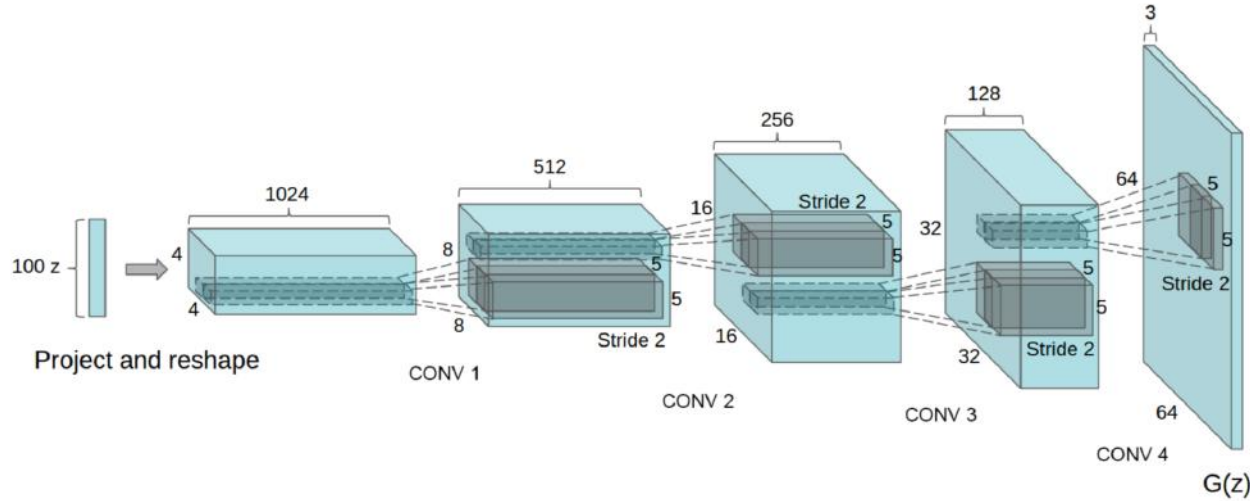
Applications

More realistic generator



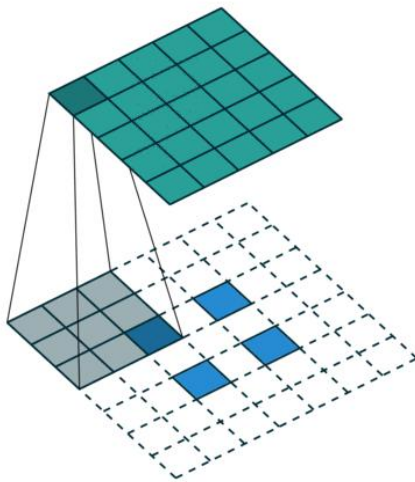
- Original GAN used MLP
 - Generated samples were OK for digits but not so great for real-world images

Deep Convolution GAN (DCGAN) [\[link\]](#)



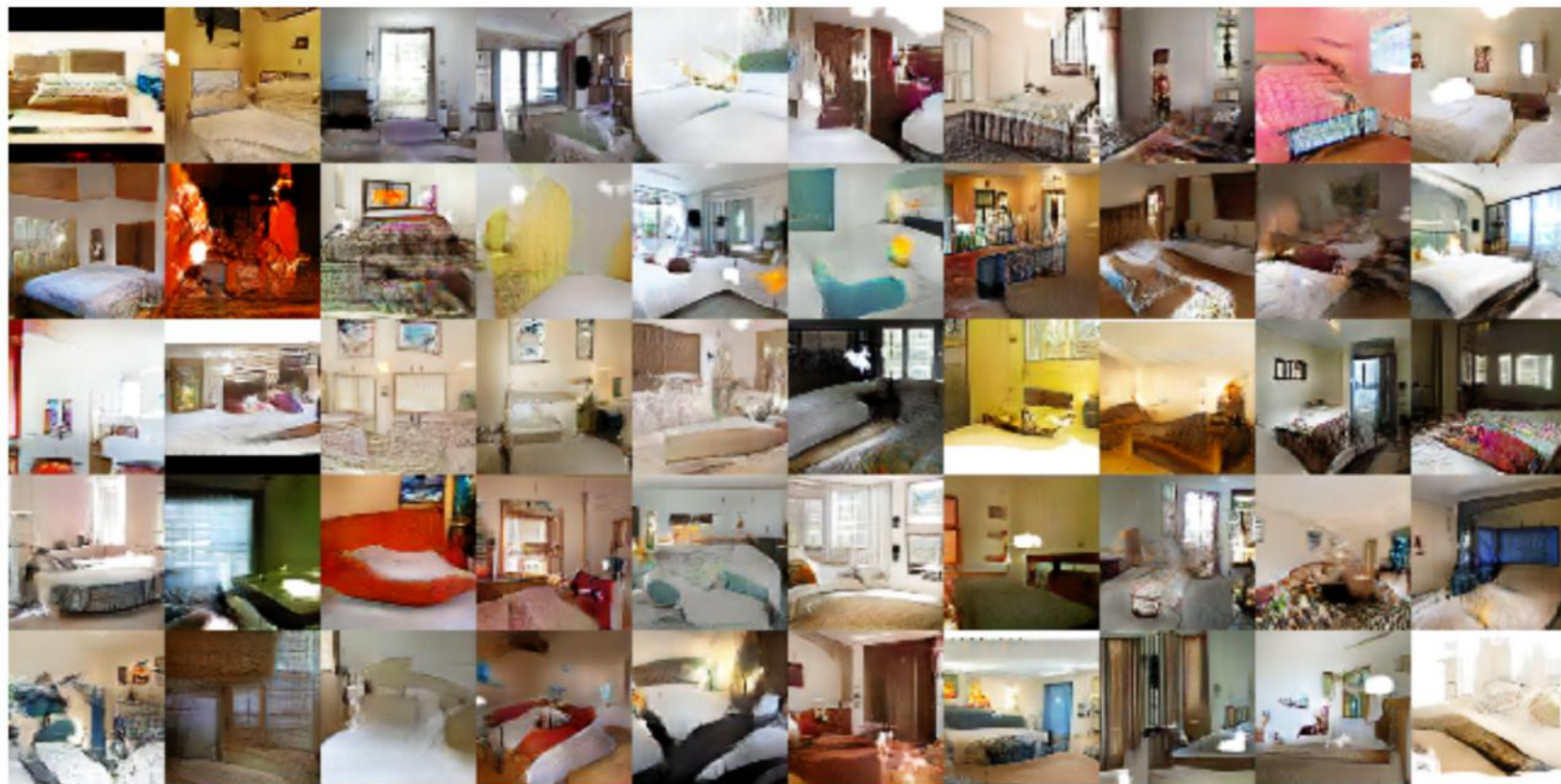
- No FC layers
- Pooling layers replaced w/ fractional stride convolutions
- Batchnorm used throughout G, D
 - Normalize input in each layer to have zero mean and unit variance

Fractional (Transposed) Convolution [\[link\]](#)



- Zero values inserted between input pixels
- Useful for upsampling

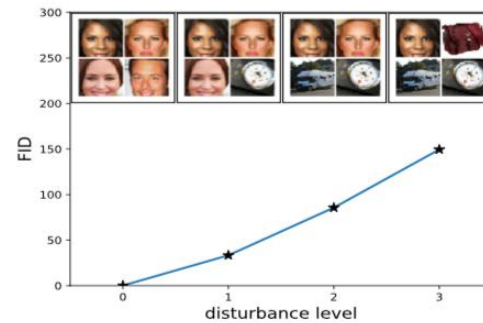
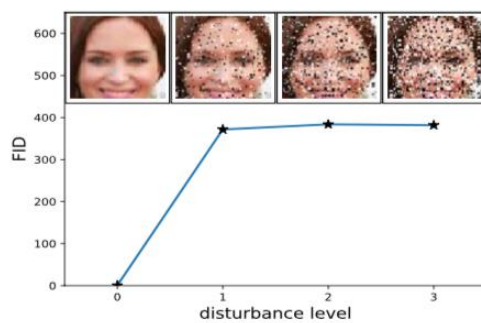
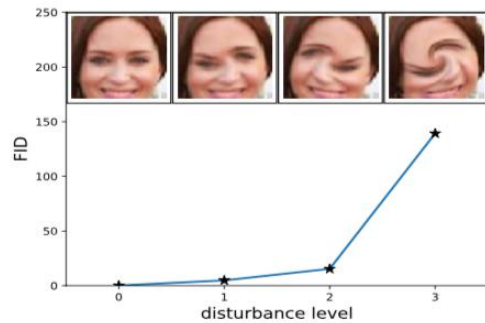
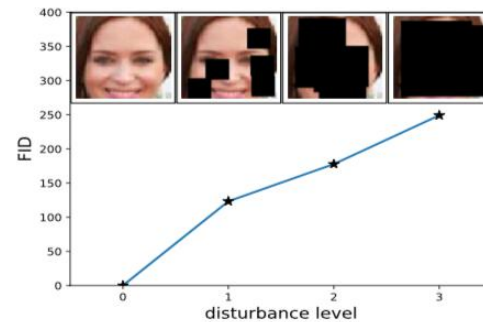
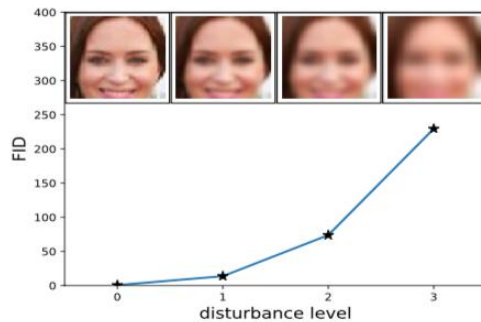
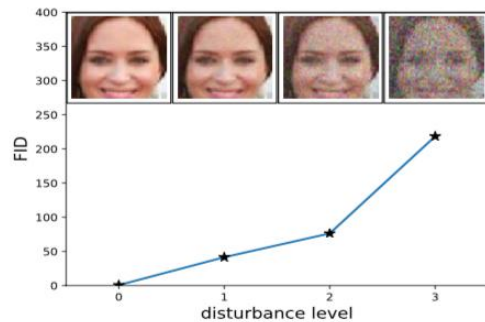
DCGan Output



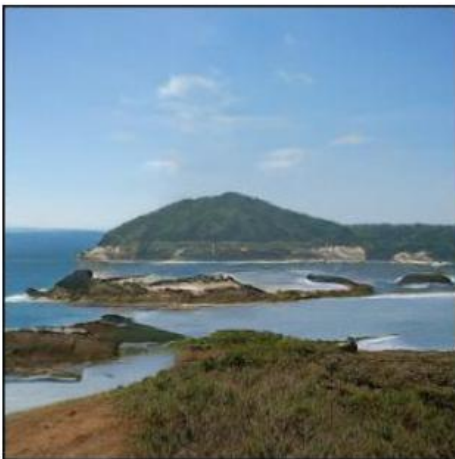
How to evaluate these models?

- Inception score [\[link\]](#)
 - Good quality: $p(y|x)$ should have low entropy (i.e. the image should correspond to a small number of classes)
 - Variation: $p(y) = \int p(y|x = G(z)) dz$ should have high entropy
 - $IS(G) = \exp[\mathbb{E}_x KL(p(\hat{y}|x) || p(y))]$ combines the two factors
 - Higher is better
- “Fréchet Inception Distance [\[link\]](#)
 - Extract Inception features for real and generated images
 - Compute means and covariances $\mu_x, \mu_g, \Sigma_x, \Sigma_g$
 - $FID(x, g) = \|\mu_x - \mu_g\| + tr(\Sigma_x + \Sigma_g + 2(\Sigma_x \Sigma_g)^{1/2})$ computes difference b/w dist
 - Lower is better

FID Score [\[link\]](#)

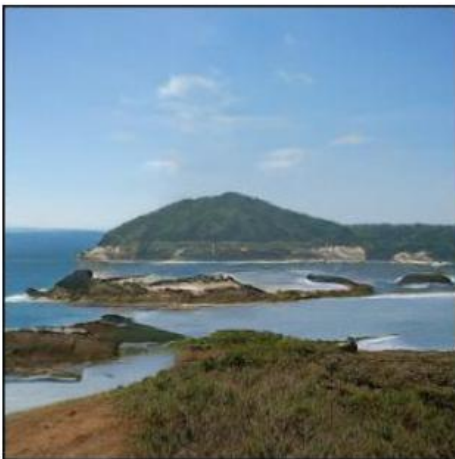


BigGAN [\[link\]](#)



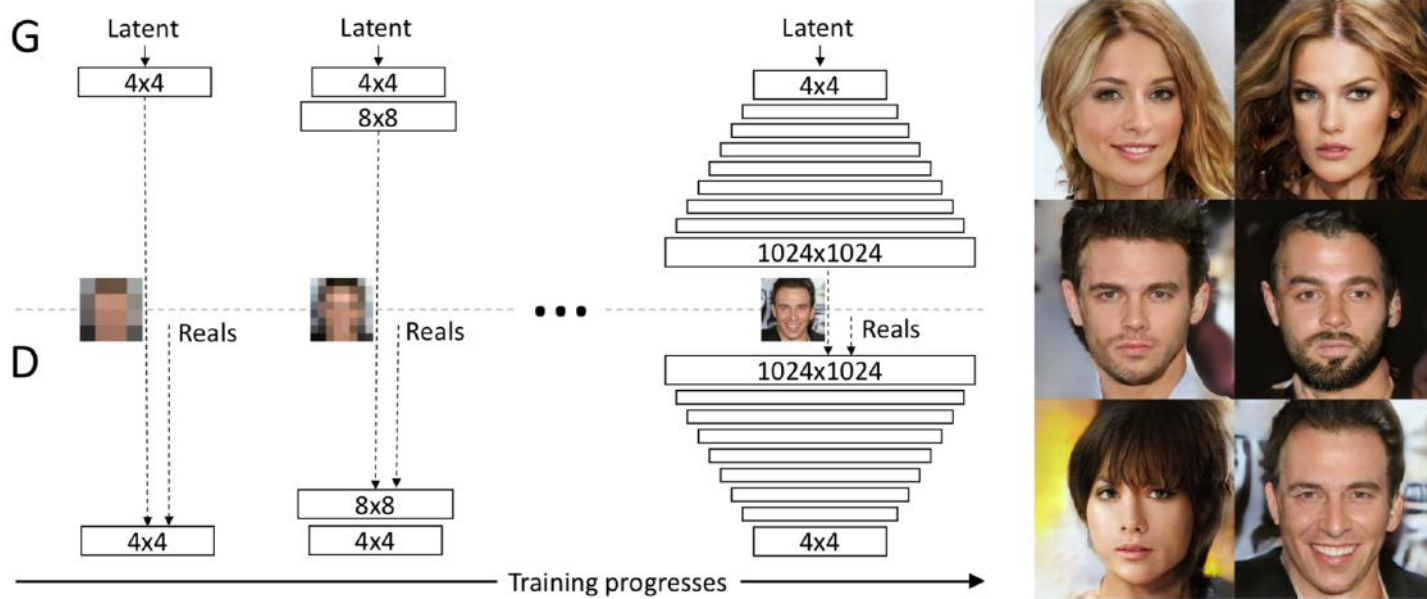
- Can you tell which images are real vs generated?

BigGAN [\[link\]](#)



- They are ALL generated!
- Scale up
 - Batch Size
 - Number of channels (2x overall number parameters)
- But still susceptible to mode collapse

Realistic Face Image Generation [\[link\]](#)



- Start at low spatial resolution and progressively add more, wider layers

Realistic Face Image Generation [\[link\]](#)



- Scales up to 1024 x 1024
- Some artifacts at perimeter

Super-resolution GAN [\[link\]](#)

bicubic
(21.59dB/0.6423)



SRResNet
(23.53dB/0.7832)



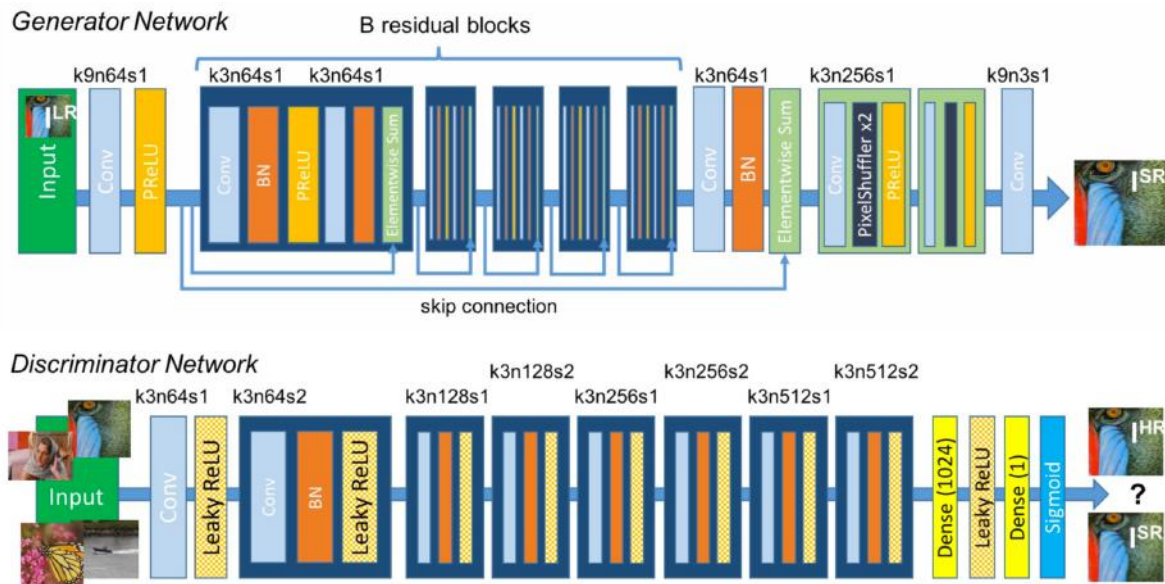
SRGAN
(21.15dB/0.6868)



original



Super-resolution GAN [\[link\]](#)



- Notation: k (kernel size), n (# feature maps), s (stride)
- Discriminator distinguishes between original high-res and generated images

SRGan Perceptual Loss

- Optimize $\min_G \max_D \mathbb{E}_{I^{HR}}[\log D(I^{HR})] + \mathbb{E}_{I^{LR}}[\log(1 - D(G(I^{LR})))]$

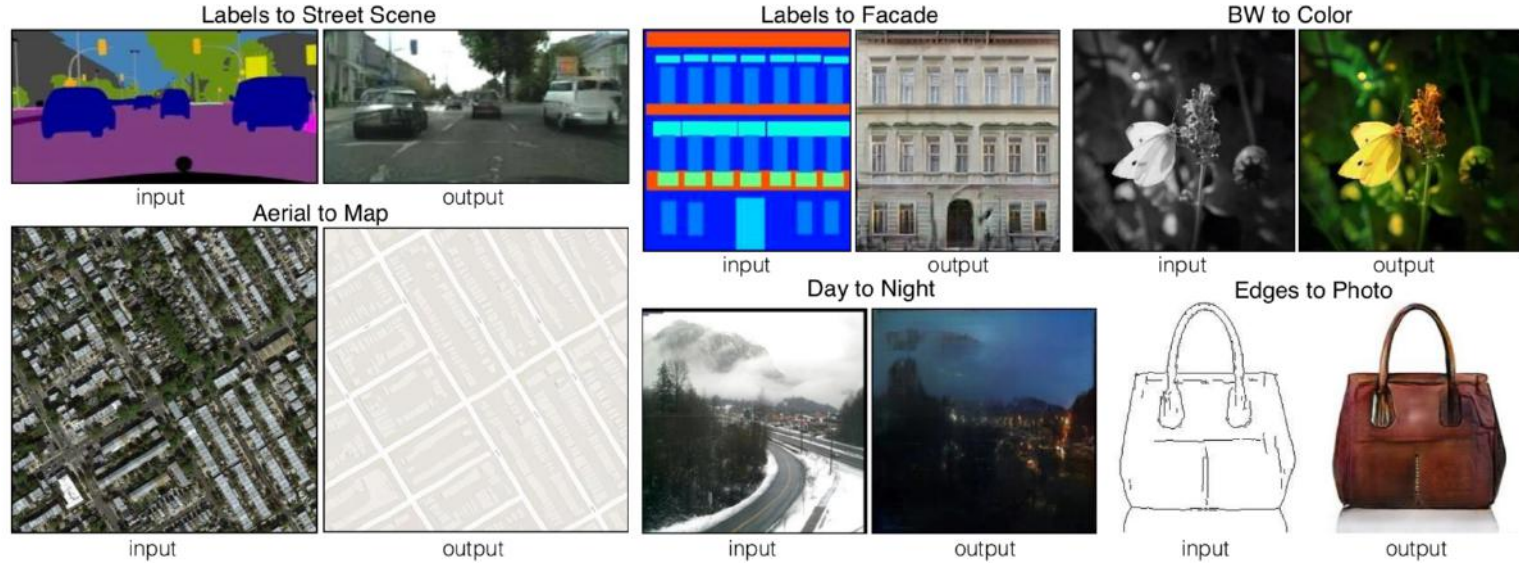
- G is optimized for $\hat{\theta}_G = \underset{\theta_G}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N l^{SR}(G(I_n^{LR}), I_n^{HR})$

- Perceptual loss $l_{SR} = l_X^{SR} + \lambda l_{Gen}^{SR}$

- Content Loss $l_X^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G(I^{LR})_{x,y})^2$

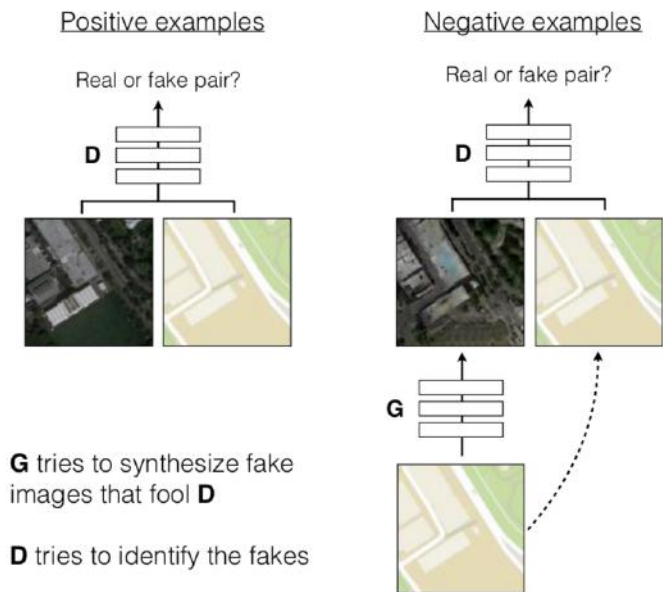
- Adversarial Loss $l_{Gen}^{SR} = \sum_{n=1}^N -\log D(G(I^{LR}))$

Pix2Pix [\[link\]](#)



- Uses conditional adversarial network for image translation

Pix2Pix [\[link\]](#)



- Conditional GAN: $L_{cGAN}(G, D) = \mathbb{E}_{x, y \sim data} [\log D(x, y)] + \mathbb{E}_{x \sim data, z \sim p(z)} [\log(1 - D(G(x, z)))]$
- L1: $L_{L_1}(G) = \mathbb{E}_{x, y \sim data, z \sim p(z)} \|y - G(x, z)\|_1$
- Total: $\min_G \max_D L_{cGAN}(G, D) + \lambda L_{L_1}(G)$

CycleGAN [\[link\]](#)

Monet \leftrightarrow Photos



Monet \rightarrow photo



photo \rightarrow Monet

Zebras \leftrightarrow Horses



zebra \rightarrow horse



horse \rightarrow zebra

Summer \leftrightarrow Winter



summer \rightarrow winter



winter \rightarrow summer



Photograph



Monet



Van Gogh

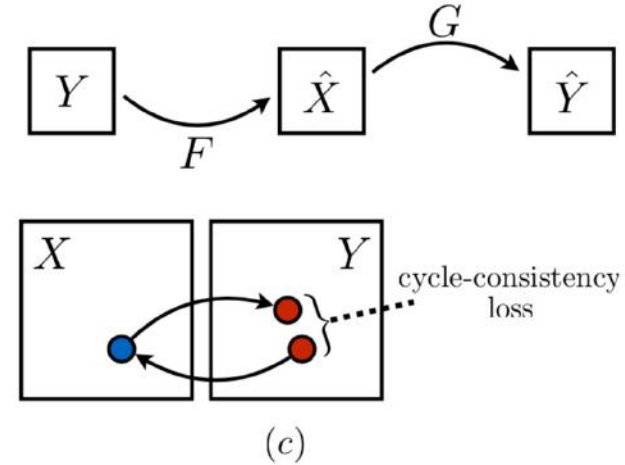
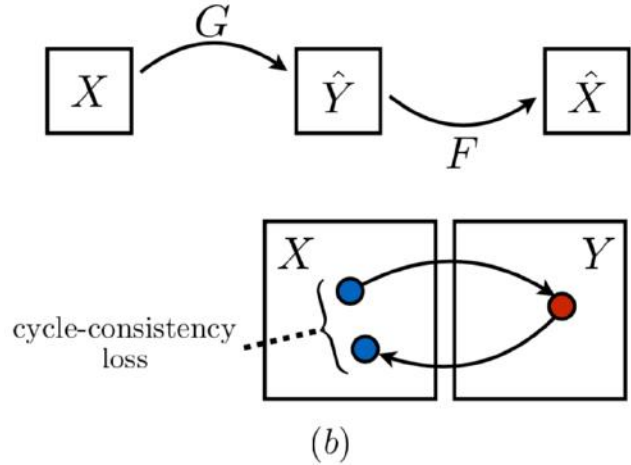
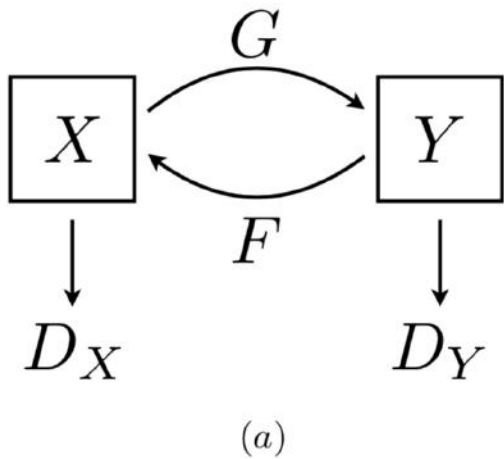


Cezanne



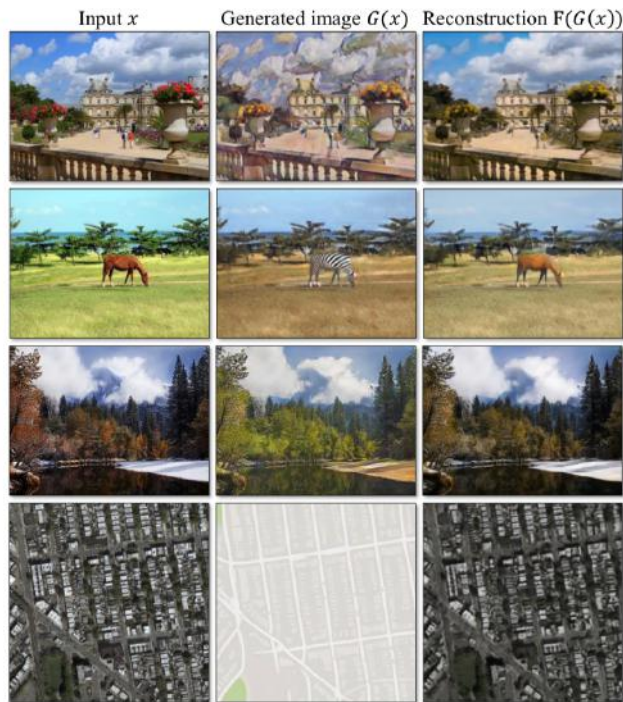
Ukiyo-e

CycleGAN [\[link\]](#)



- Pairs of generators G, F . $G: X \rightarrow Y$ and $F: Y \rightarrow X$
- Pair of discriminators
- Extension from Pix2Pix

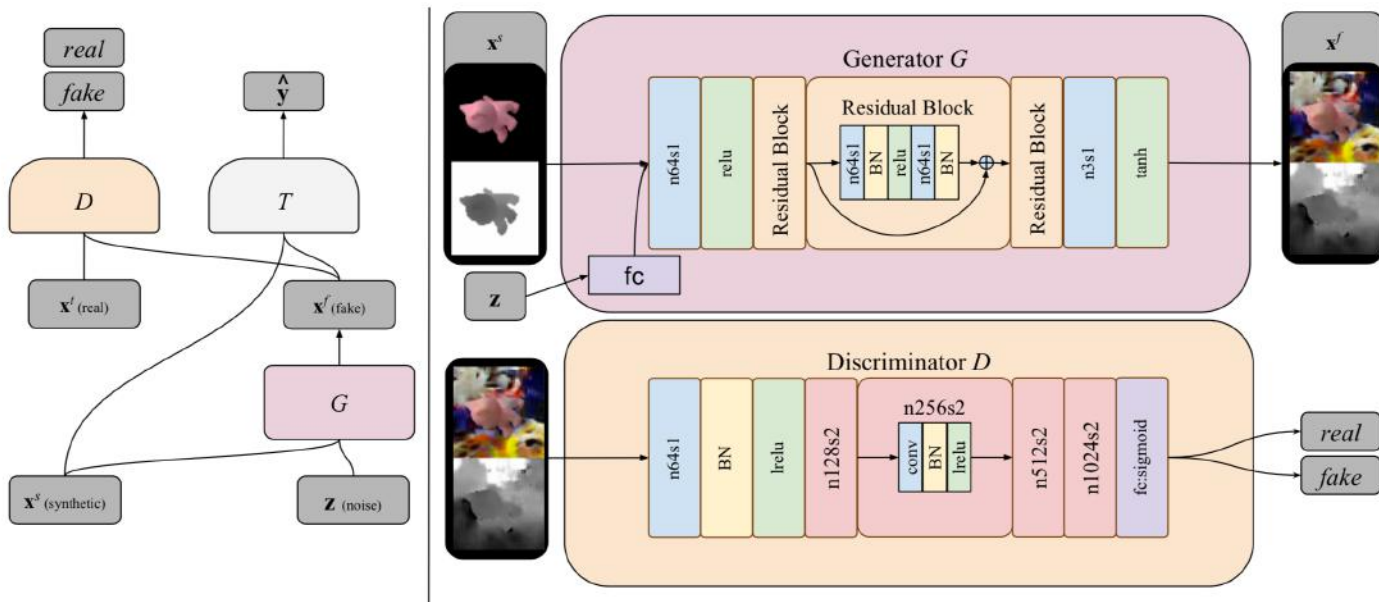
CycleGAN [\[link\]](#)



- Adversarial Loss: $L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim data}[\log D_Y(y)] + \mathbb{E}_{x \sim data}[\log(1 - D_Y(G(x)))]$
- Cycle Consistency Loss: $L_{cyc}(G, F) = \mathbb{E}_{x \sim data}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim data}[\|G(F(y)) - y\|_1]$
- Total: $L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F)$

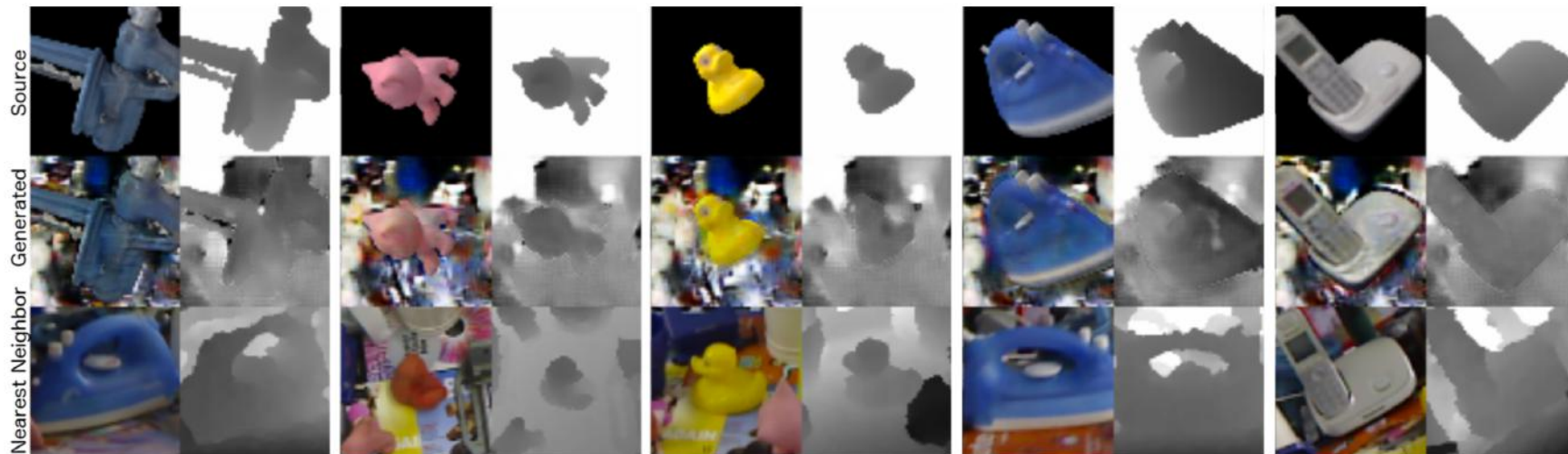
Domain Adaptation ([PixelDA](#))

- Unsupervised DA
- Use G to adapt synthetic images into target domain



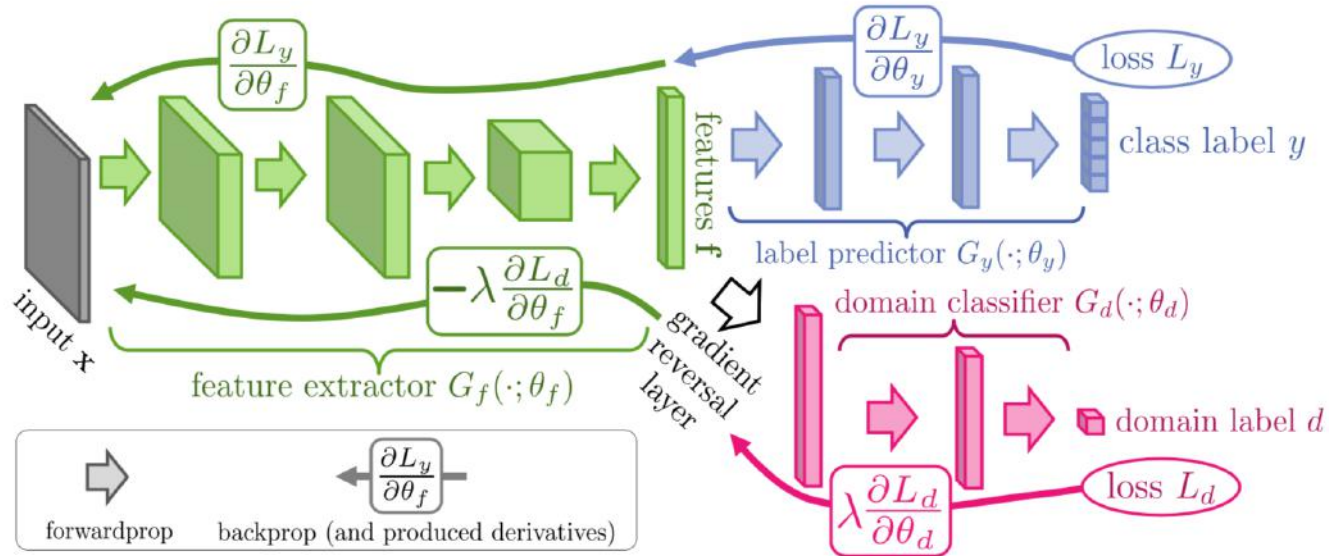
Domain Adaptation ([PixelDA](#))

- Sample output



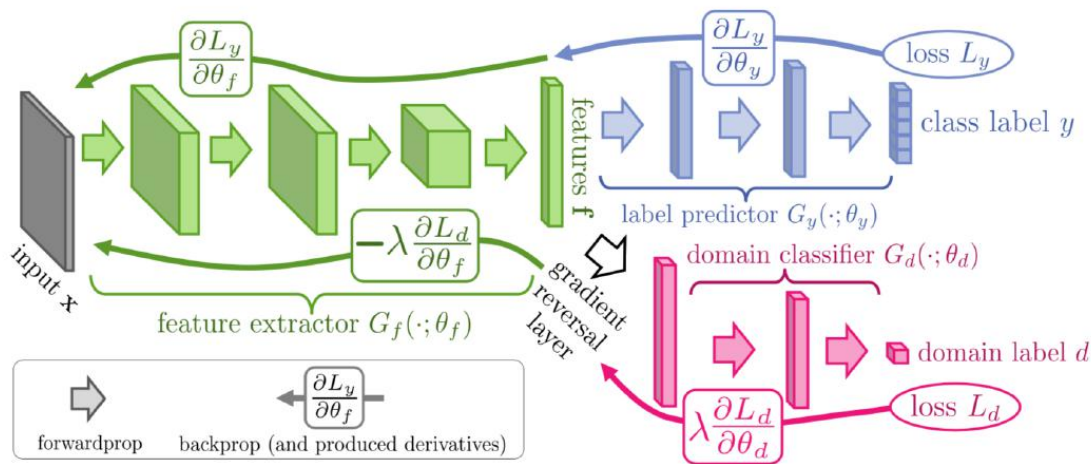
Domain Adaptation

- Domain Adversarial Neural Network (DANN)
 - Uses a gradient reversal layer that multiplies gradient from domain classifier by a negative constant during training



Domain Adaptation

- Domain Adversarial Neural Network (DANN)



$$\theta_f \leftarrow \theta_f - \mu \left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f} \right),$$

$$\theta_y \leftarrow \theta_y - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_y},$$

$$\theta_d \leftarrow \theta_d - \mu \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_d},$$

- **Green** and **blue** regions correspond to standard CNN
- **Red** region corresponds to discriminator

InfoGAN [\[link\]](#)

- Learn disentangled representations of data
- Instead of a single noise variable z , also include latent code c
 - e.g. c can correspond to 0-9 when generating digits
- Enforce high mutual information $I(c; G(z, c))$
- Optimize: $\min_G \max_D \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))] - \lambda I(c; G(z, c))$
 - Add the MI term

InfoGAN [\[link\]](#)



(a) Azimuth (pose)

(b) Elevation

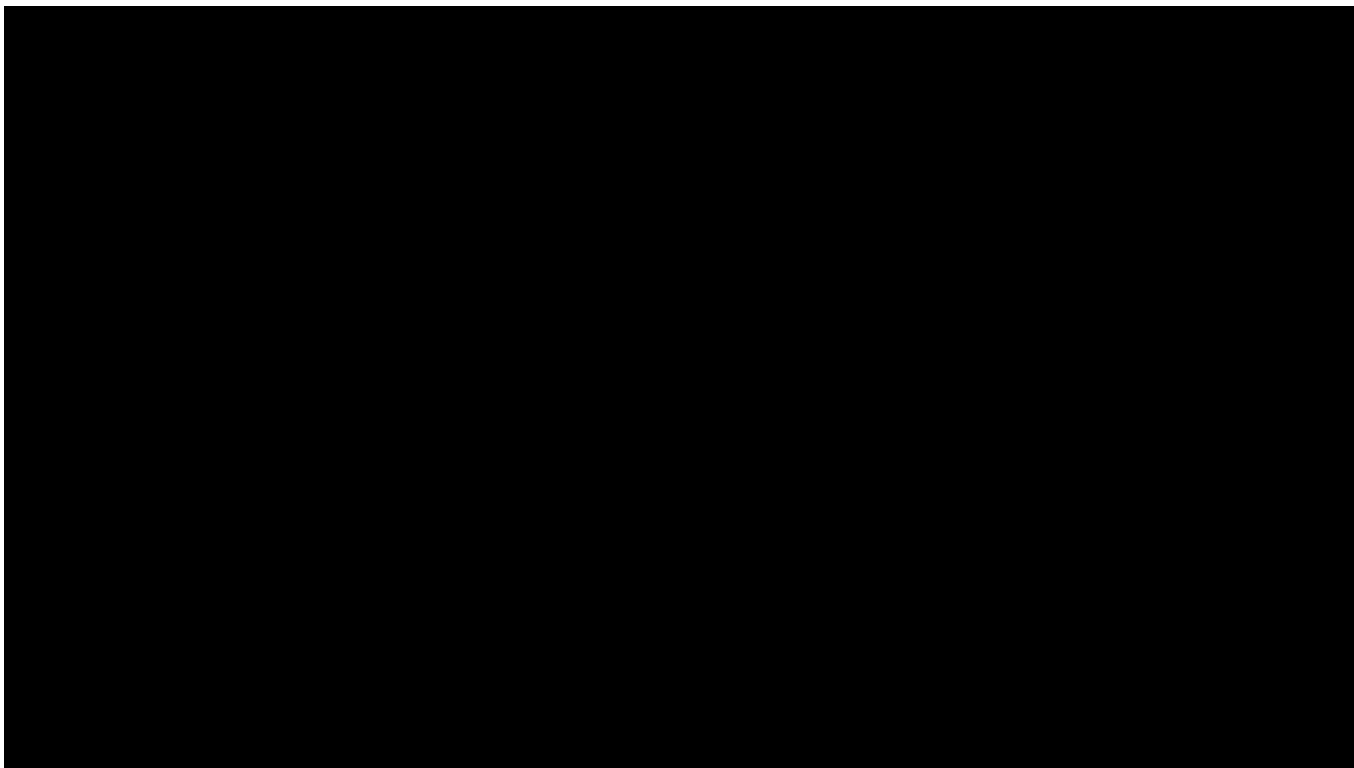


(a) Rotation

(b) Width

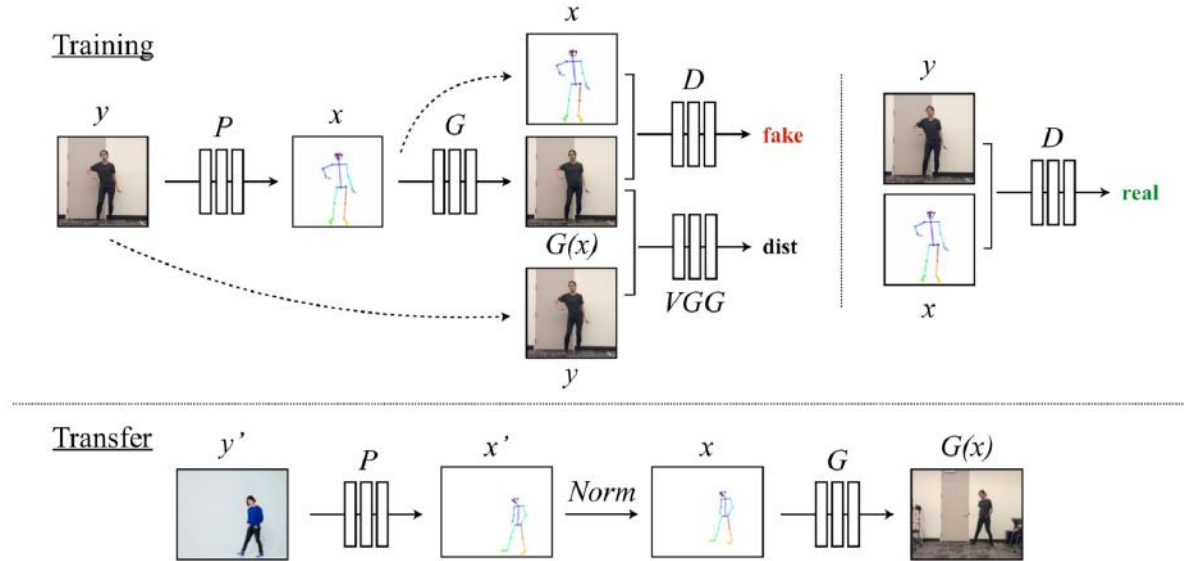
- Latent coded correspond to attributes

Everybody Dance Now [\[link\]](#)



- <https://www.youtube.com/watch?v=PCBTZh41Ris>

Everybody Dance Now [\[link\]](#)



- G maps from pose “stick figure” to synthesized target
- D distinguishes (x, y) from $(x, G(x))$

Links to papers

- [GAN](#)
 - [WGAN](#)
- Applications
 - [DCGan](#)
 - [Face Generation](#)
 - [BigGAN](#)
 - [Pix2Pix](#)
 - [CycleGAN](#)
 - [SRGan](#)
 - [DANN](#)
 - [InfoGAN](#)
 - [Dancing GAN](#)

Thank you!

Any questions?