3546 – Deep Learning

Module 9: Deep Generative Models

Course Syllabus

Module	Topic	Deliverables
1	Course Intro + Review	Term Project Released
2	Model Tuning	Assignment 1 Released
3	Convolutional Networks	
4	Deep Computer Vision	Assignment 1 Due, A2 Released
5	Recurrent Neural Networks	
6	Natural Language Processing	
7	Deep Models for Text	Assignment 2 Due, A3 Released
8	Representational Learning & Variational Methods	
9	Deep Generative Models	Assignment 3 Due, A4 Released
10	Speech and Music Recognition & Synthesis	
11	Term Project Presentations A	Term Project Due
12	Term Project Presentations B	Assignment 4 Due

Learning Outcomes for this Module

By the end of this module, you will:

- Understand the core concepts behind Generative Adversarial Networks.
- Know common difficulties when training GANs and techniques to mitigate them.
- Be able to implement a basic GAN in Tensorflow.
- Be familiar with more advanced GAN architectures including CycleGAN and StyleGAN

Module 9 - Section 1

Generative Adversarial Networks

Recap

Generative Models

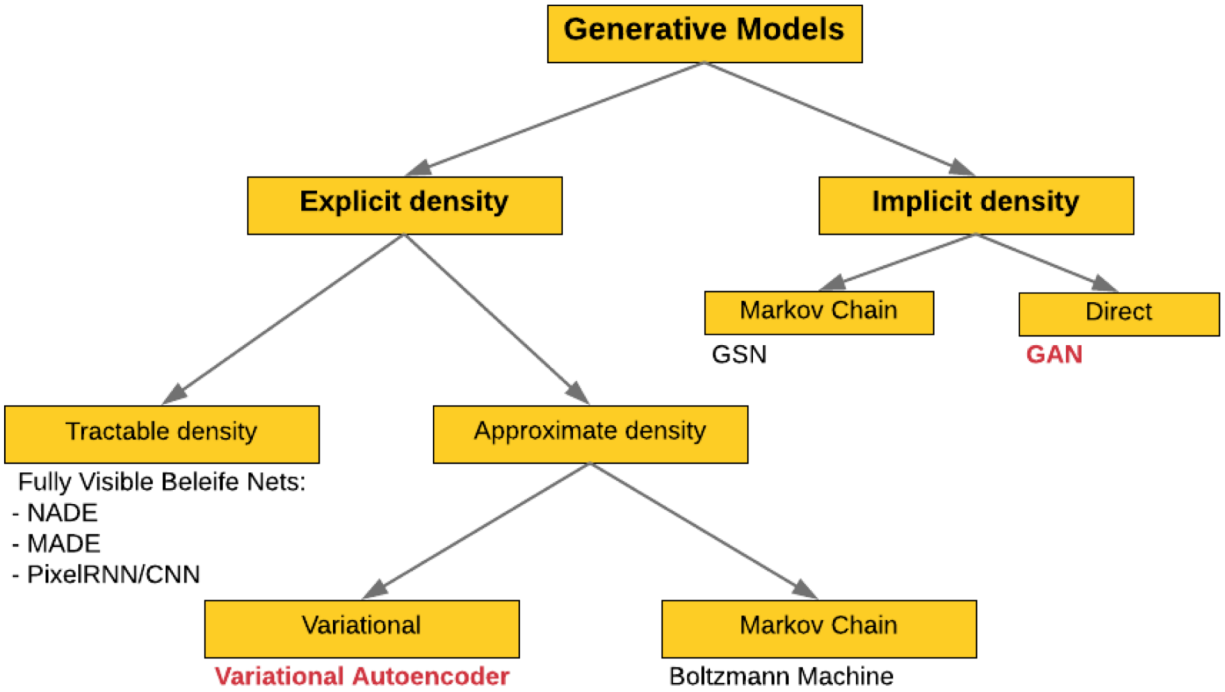
- Given target variable ***Y***, find a probable ***X*** that belongs to it.
- Sampling from conditional probability *P(**X** | **Y**)* lets us synthesize new, never-before-seen samples.

Variational Autoencoders (VAEs)

- Learns a latent-vector representation of data.
- Encoder output for each input is a Gaussian distribution (μ, σ), from which a latent vector is sampled.
- Uses KL-divergence penalty to constrain encoding space to a unit Gaussian over all classes. Makes sampling 'realistic' examples easy.

Image Source: https://www.jeremyjordan.me/variational-autoencoders/

Generative Adversarial Networks Taxonomy

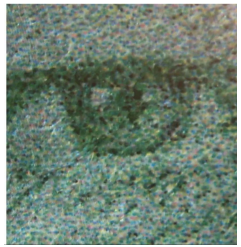


[**Image credit:** Goodfellow, I. (2016), Generative Adversarial Networks - NIPS 2016 tutorial.Retrieved from https://www.youtube.com/watch?v=HGYYEUSm-0Q]

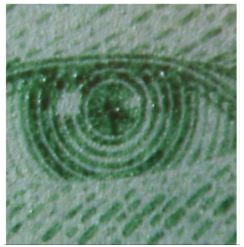
Adversarial Learning

Example: anti-counterfeit measures.
Both parties adapt their methods.

Fake



Real



1969

"Rainbow printing" with multicolour tints.

1986

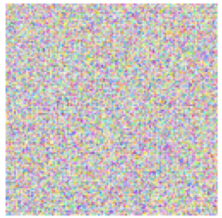
Metallic patch.
Raised ink.
Larger fine-grain portraits.

2011

Multi-layer polymer material.
Holographic insert.

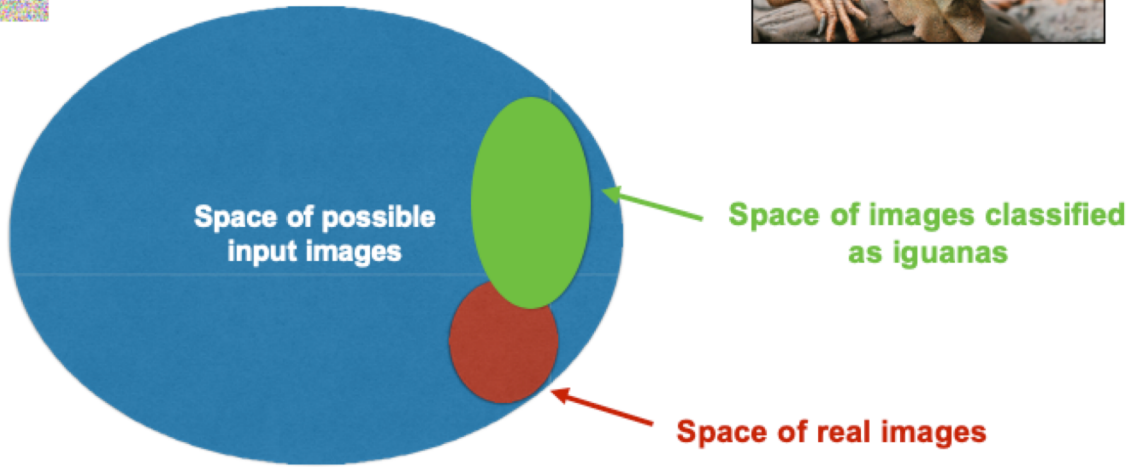
Attacking a network with adversarial examples

Question: Will the forged image **x** look like an **iguana**?



?

$256^{32\times32\times3} \approx 10^{7400}$



Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Goal: Given a network pretrained on ImageNet, find an input image that is a cat but will be classify as an iguana.

Neural network (pretrained on ImageNet)

0.04	"car"
0.85	"iguana"
0.02	"tomato"
0.07	"bike"
0.81	"cat"
⋮	
0.02	"crab"

1. Rephrasing what we want:

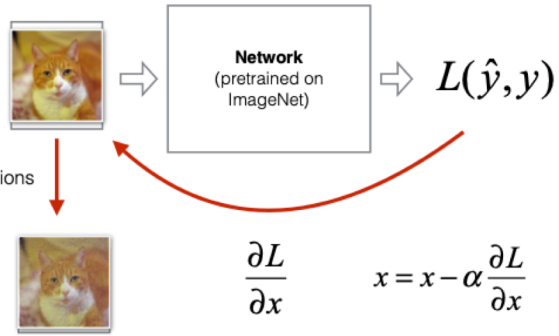
Find x such that: $\hat{y}(x) = y_{iguana} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

And: $x = x_{cat}$

2. Defining the loss function

$$L(\hat{y}, y) = \frac{1}{2}\left\|\hat{y}(W, b, x) - y_{iguana}\right\|_2^2 + \lambda\left\|x - x_{cat}\right\|_2^2$$

3. Optimize the image

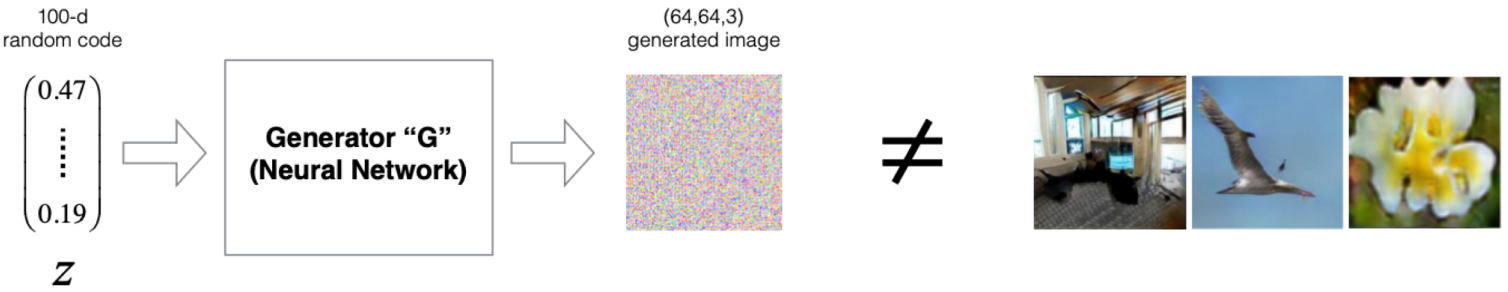


[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

92% Cat

94% Iguana



How can we train G to generate images from the true data distributions?

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas (2017): StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks]

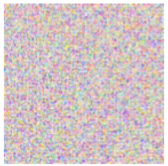
100-d
random code

$$\begin{pmatrix} 0.47 \\ \vdots \\ 0.19 \end{pmatrix}$$

z

Generator "G"
(Neural Network)

(64,64,3)
generated image



Run Adam simultaneously on two minibatches
(true data / generated data)

x

Discriminator "D"
(Neural Network)

Gradients

Binary classification

$$\begin{cases} y = 0 \quad if \quad x = G(z) \\ y = 1 \quad otherwise \end{cases}$$

Real images
(database)

Probability distributions

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Training procedure, we want to minimize:

Labels: $\begin{cases} y_{real} \text{ is always 1} \\ y_{gen} \text{ is always 0} \end{cases}$

- The cost of the discriminator

$$J^{(D)} = \underbrace{-\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} y_{real}^{(i)} . \log(D(x^{(i)}))}_{\text{cross-entropy 1}} \underbrace{- \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1 - y_{gen}^{(i)}) . \log(1 - D(G(z^{(i)})))}_{\text{cross-entropy 2}}$$

cross-entropy 1:
"D should correctly label real data as 1"

cross-entropy 2:
"D should correctly label generated data as 0"

- The cost of the generator

$$J^{(G)} = -J^{(D)} = \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D(G(z^{(i)})))$$

"G should try to fool D: by minimizing the opposite of what D is trying to minimize"

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

GAN Loss Functions

Likelihood of a correct classification.

Discriminator Objective:

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right) \right]$$

Identify real images better

Identify fake images better

Generator Objective:

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$

Minimize discriminator's ability to identify fake images better

Equivalent to binary cross-entropy + merely flipping the labels for fake images from 0 to 1 during Stage 2.

GAN Gradients

Discriminator Objective:

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

Generator Objective:

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

Generative Adversarial Networks (GANs)

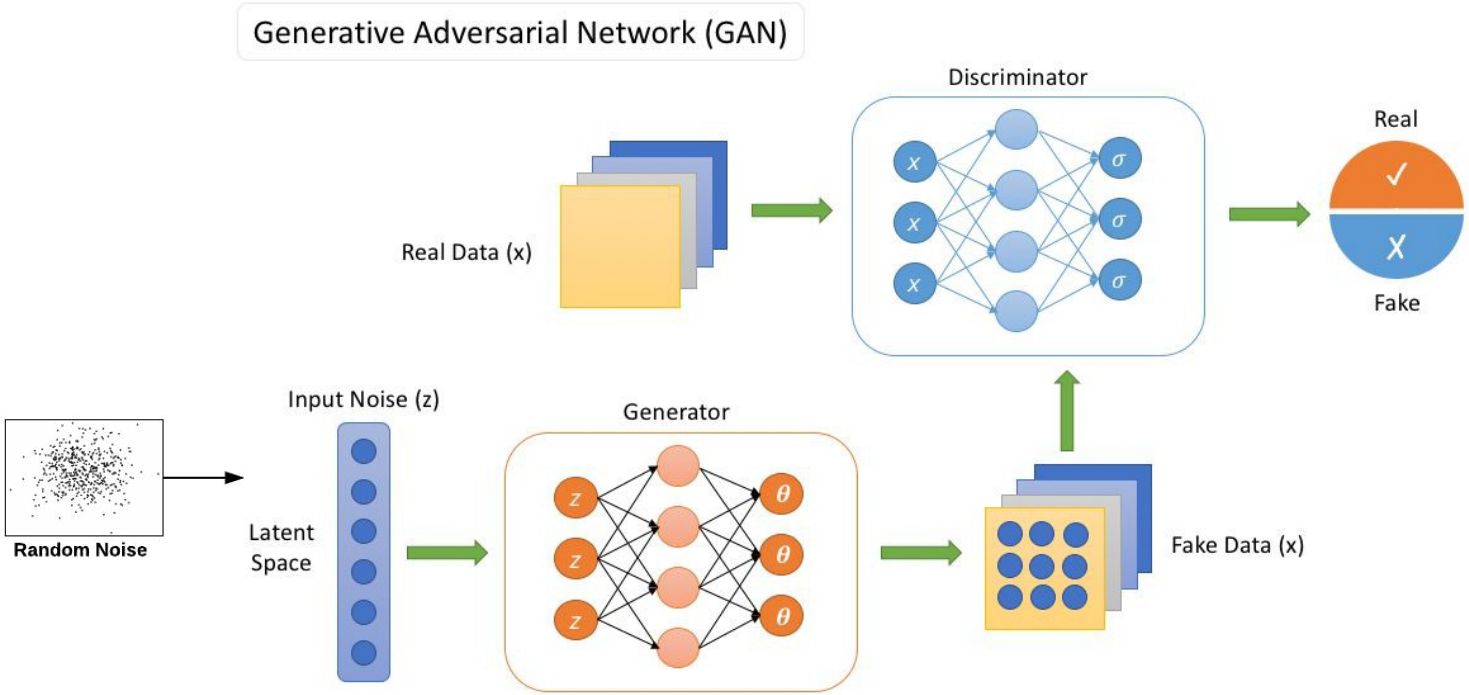
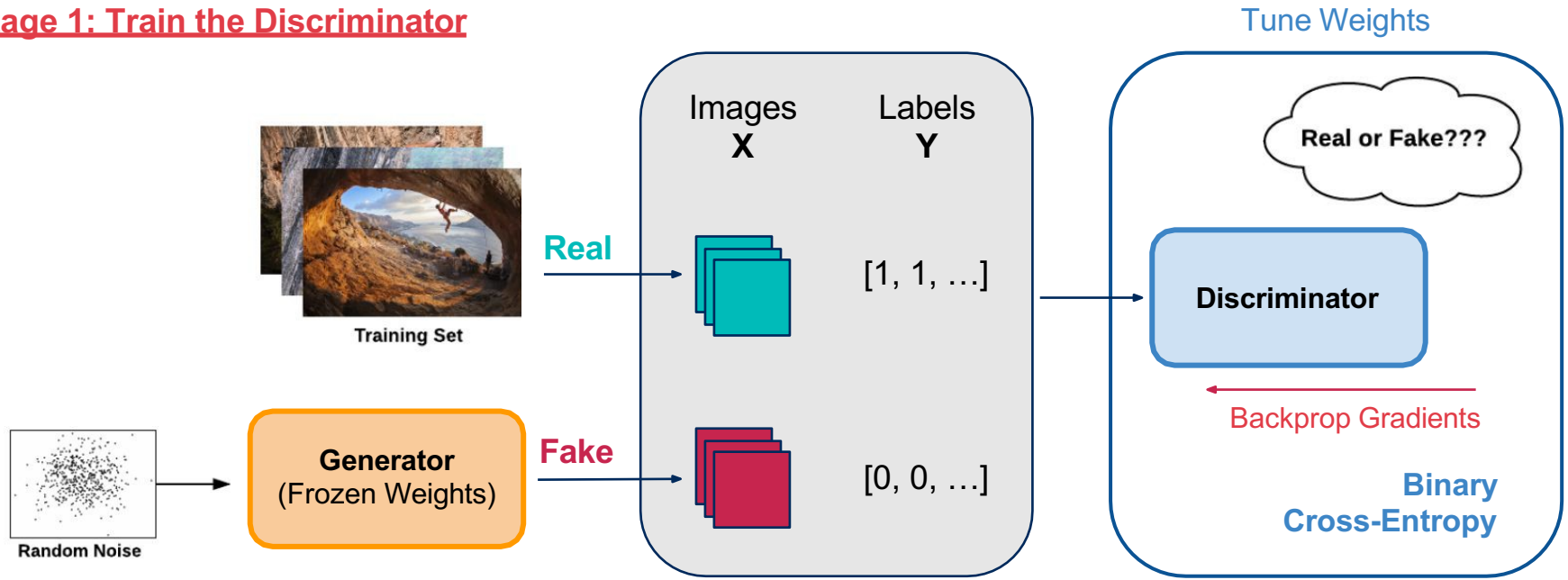


Image Source: https://medium.com/xebia-engineering/generative-adversarial-network-2b063f587bae

Training GANs

A two-step process for each mini-batch.

Stage 1: Train the Discriminator

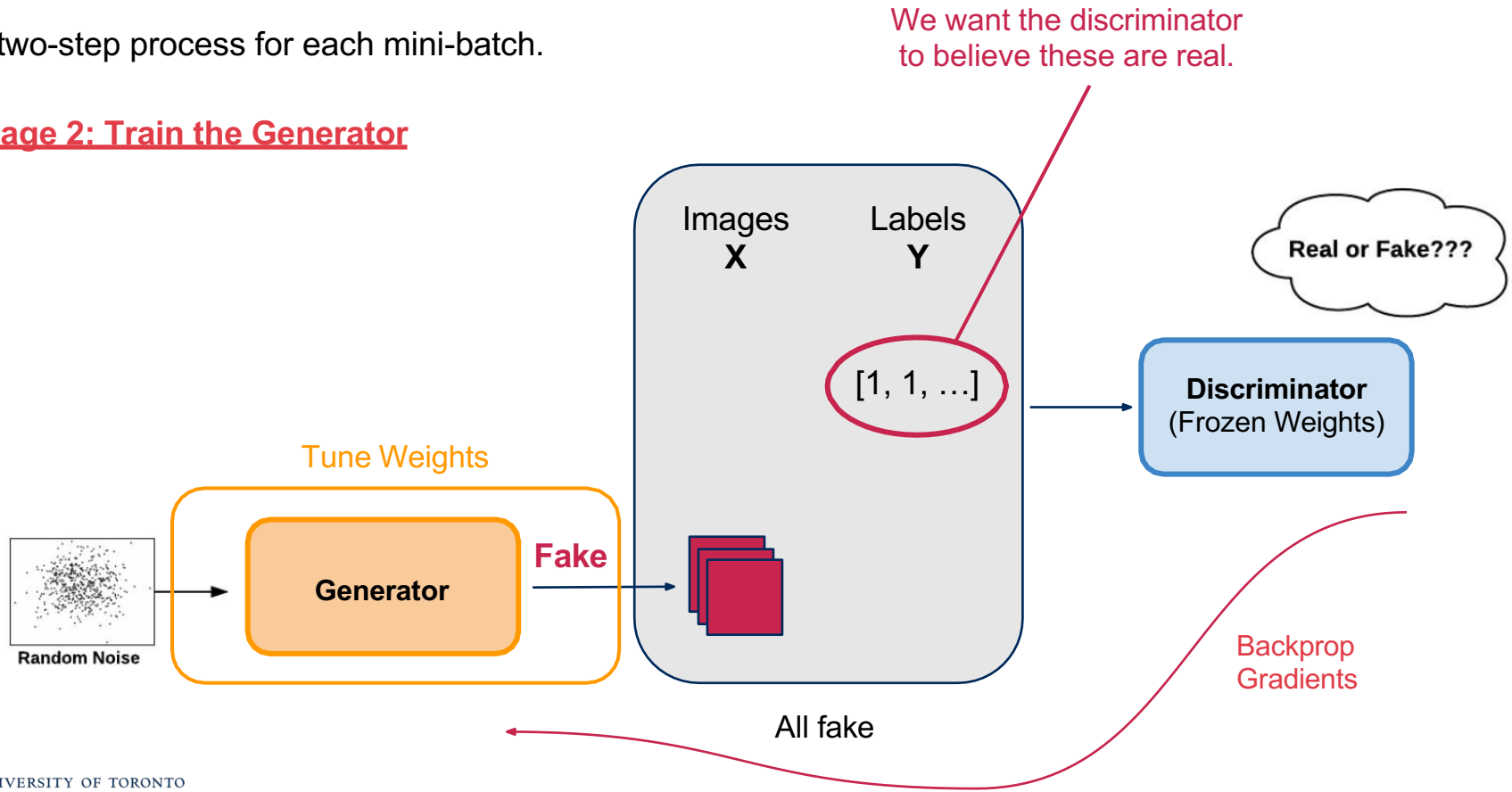


Equal number of fake and real.

Training GANs

A two-step process for each mini-batch.

Stage 2: Train the Generator



Discriminator Guides the Generator

The generator never gets to see the actual, real images. Yet it learns to produce convincing images through the second-hand gradients passed to it by the discriminator.

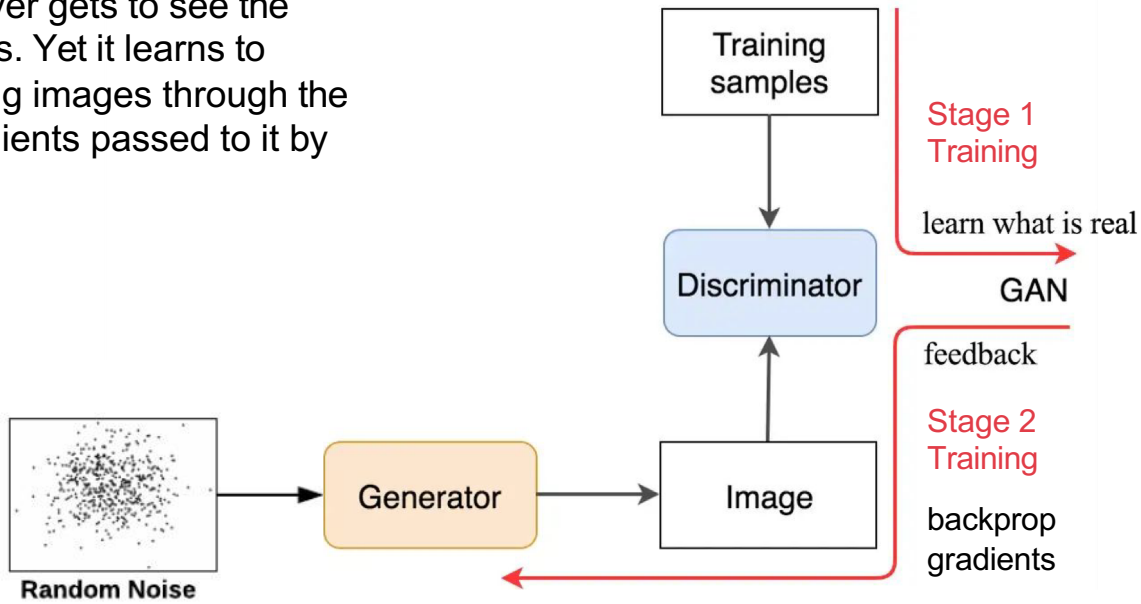
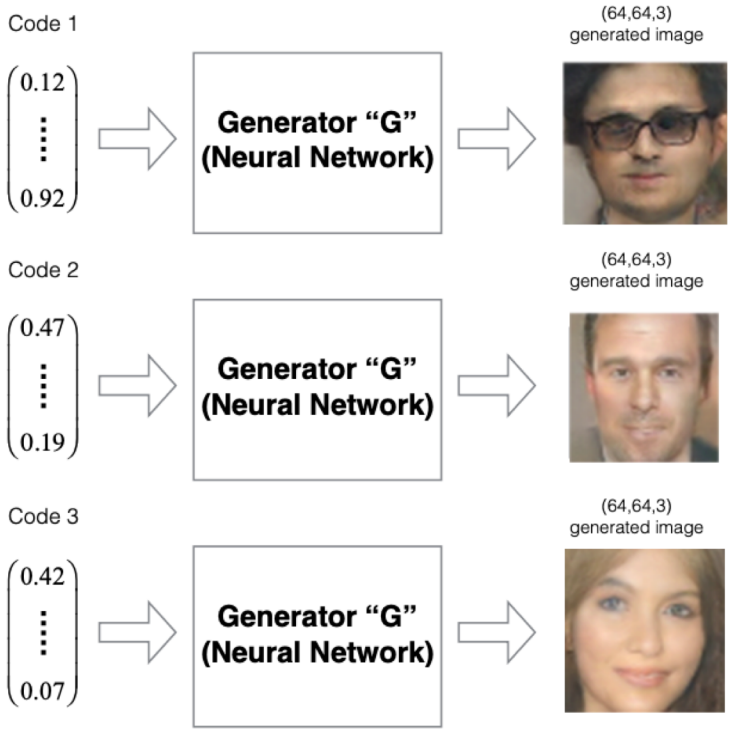
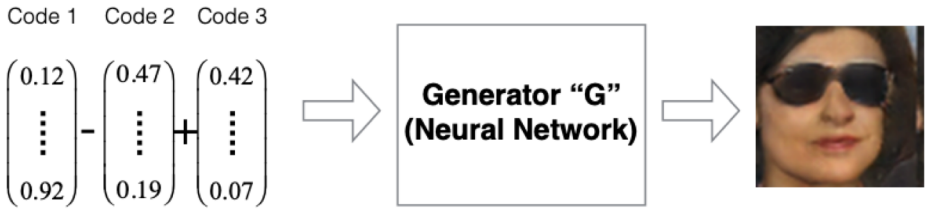


Image Source: https://jonathan-hui.medium.com/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09

Operation on codes





Man with glasses - man + woman = woman with glasses

Module 9 - Section 2

Training GANs Successfully

Challenges when Training GANs

Theoretically, equilibrium is reached when the generator produces perfectly realistic images and the discriminator is forced to guess 50:50. However, several problems can occur:

Slow Initial Training Rate: The generator starts by producing random noise, which is easy for the discriminator to spot. The discriminator has low incentive to improve.

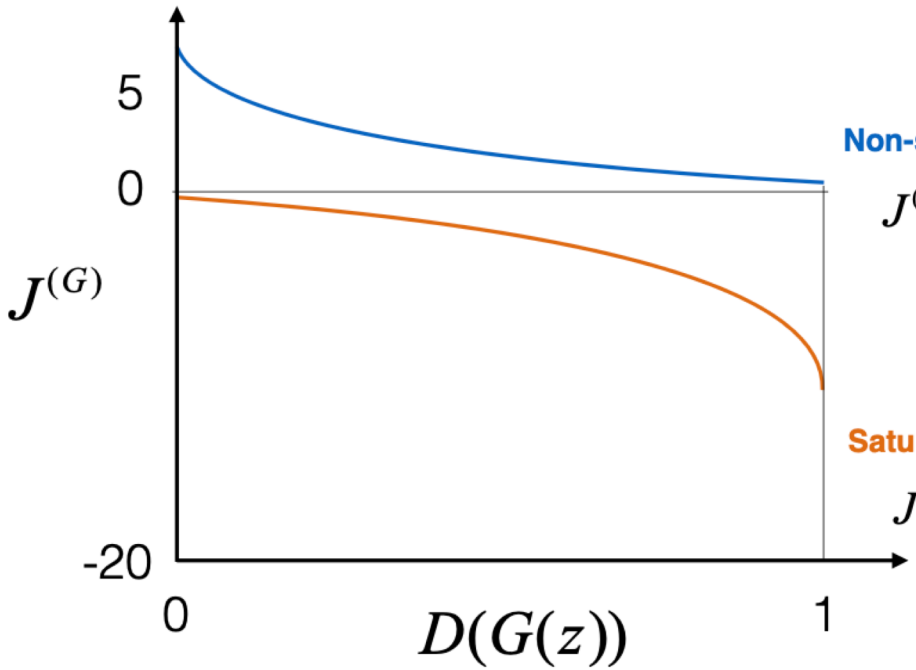
Non-Convergence: Since the generator and discriminator are constantly pushing back against each other, scores for the two 'players' may oscillate, with their trained parameters oscillating and becoming unstable.

Mode Collapse: The generator starts to focus on classes that it's particularly good at faking. Its outputs become less diverse. It forgets how to fake the other classes. The discriminator in turn specializes in spotting fakes for this class, and forgets about the other classes. Eventually the generator might switch to another class to focus on, and this process repeats itself.

→ **GANs are highly sensitive to their hyperparameters!**

Saturating cost for the generator:

$$\min\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D(G(z^{(i)})))\right] \Leftrightarrow \max\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right] \Leftrightarrow \min\left[-\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right]$$



Non-saturating cost

$$J^{(G)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))$$

Saturating cost

$$J^{(G)} = \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D(G(z^{(i)})))$$

[Ian Goodfellow (2014): NIPS Tutorial: GANs]

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Note that: $$\min\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D(G(z^{(i)})))\right] \Leftrightarrow \max\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right] \Leftrightarrow \min\left[-\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right]$$

New training procedure, we want to minimize:

$$J^{(D)} = \underbrace{-\frac{1}{m_{real}}\sum_{i=1}^{m_{real}} y_{real}^{(i)}.\log(D(x^{(i)}))}_{\text{cross-entropy 1:}} \underbrace{-\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}(1-y_{gen}^{(i)}).\log(1-D(G(z^{(i)})))}_{\text{cross-entropy 2:}}$$

cross-entropy 1:
"D should correctly label real data as 1"

cross-entropy 2:
"D should correctly label generated data as 0"

$$J^{(G)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))$$

"G should try to fool D: by minimizing this"

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Table 1: Generator and discriminator loss functions. The main difference whether the discriminator outputs a probability (MM GAN, NS GAN, DRAGAN) or its output is unbounded (WGAN, WGAN GP, LS GAN, BEGAN), whether the gradient penalty is present (WGAN GP, DRAGAN) and where is it evaluated. We chose those models based on their popularity.

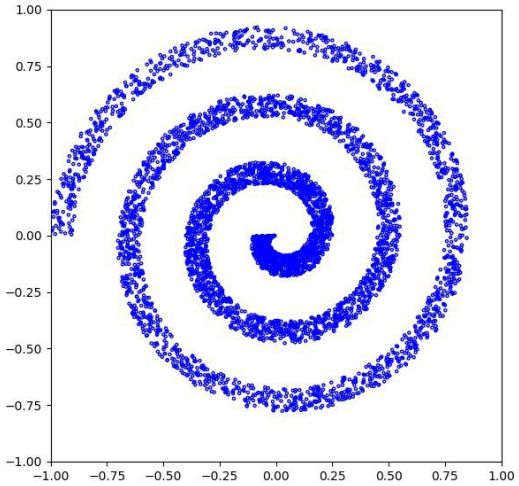
GAN	DISCRIMINATOR LOSS	GENERATOR LOSS
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$

[Lucic, Kurach et al. (2018): Are GANs Created Equal? A Large-Scale Study]

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

Visualizing Mode Collapse

Example: generate samples of (x, y) coordinates. In the 'real' dataset (below), these are only along some spiral. The generator doesn't know that.



Accuracies and losses.

Samples produced by the generator each mini-batch.

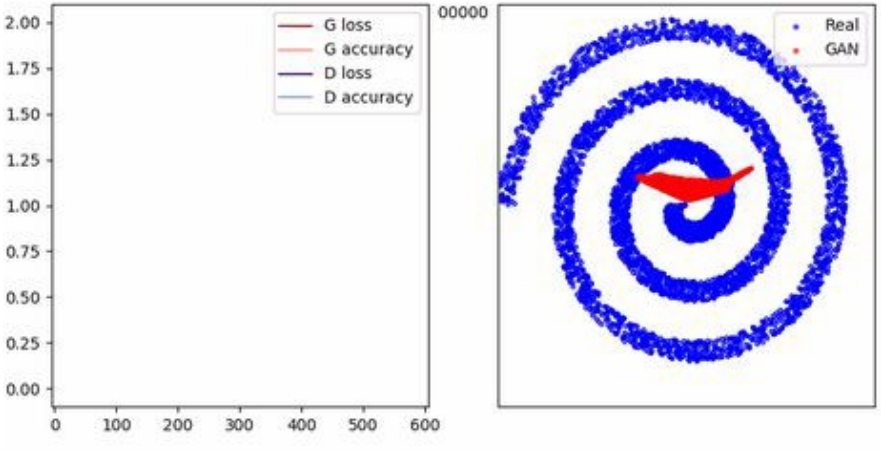


Image Source: https://towardsdatascience.com/this-will-change-the-way-you-look-at-gans-9992af250454

Avoiding Mode Collapse

Experience Replay: For each iteration, store the images produced by the generator in a buffer (gradually dropping older ones), and draw some of the fake images from this buffer. This reduces the chance that the discriminator will overfit the generator's last outputs.

Minibatch Discrimination: Measures how similar images are across a minibatch, and provides this statistic to the discriminator, so it can easily reject a whole batch of fake images that lake diversity. This encourages the generator to produce a wider variety of images.

Trial and Error: for some networks, empirically there seems to be some hyperparameter sweet-spots, such as the guidelines put forward by the DCGAN paper.

Visualizing Convergence

The same network, but using hyperparameters and some ‘tricks’ to avoid mode collapse.

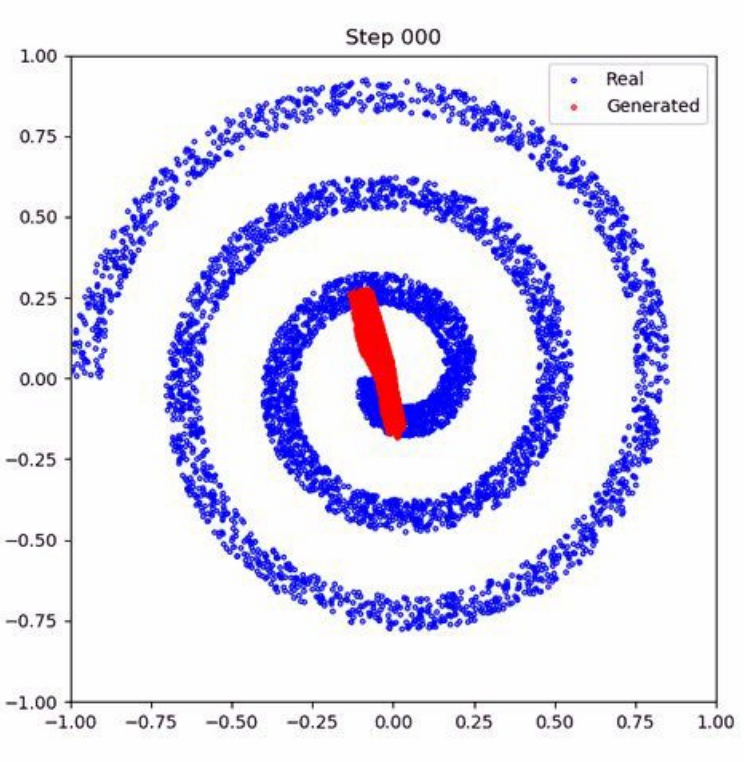


Image Source: https://towardsdatascience.com/this-will-change-the-way-you-look-at-gans-9992af250454

Global Inconsistencies

DCGANs can produce locally-convincing images, but these can have global inconsistencies. How can we do better?

Progressive Growing of GANs

Generate small images at the start of training, then gradually add convolutional layers to produce larger and larger images. Resembles greedy layer-wise training.

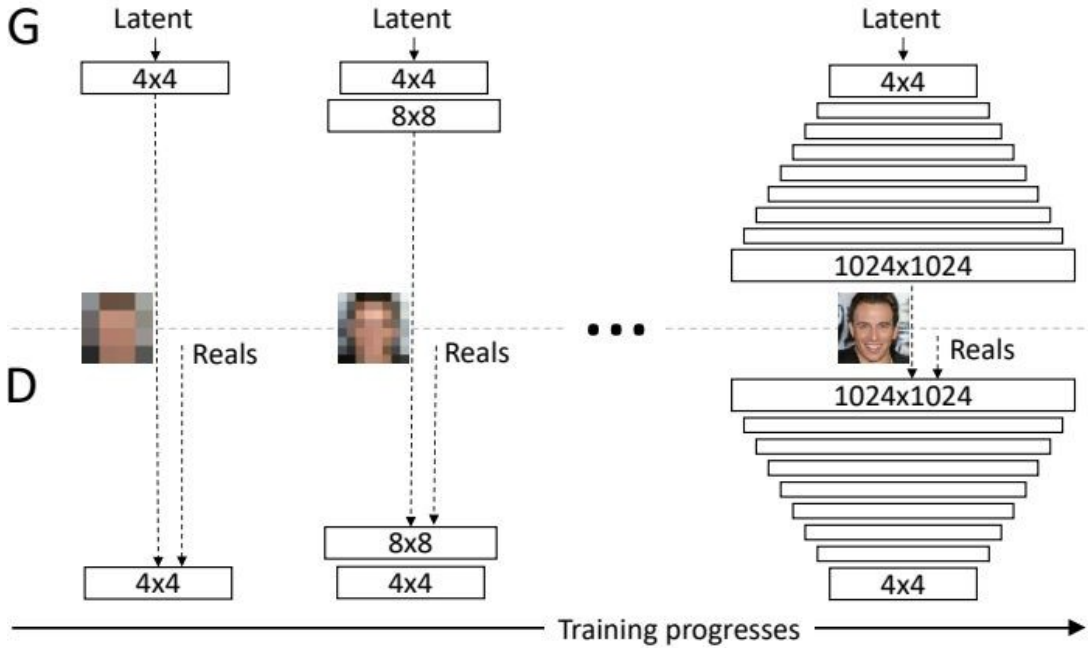


Image source: https://towardsdatascience.com/progressively-growing-gans-9cb795caebee

Progressive Growing of GANs

The original outputs are mixed with the new outputs, while gradually ramping up α from 0 to 1.

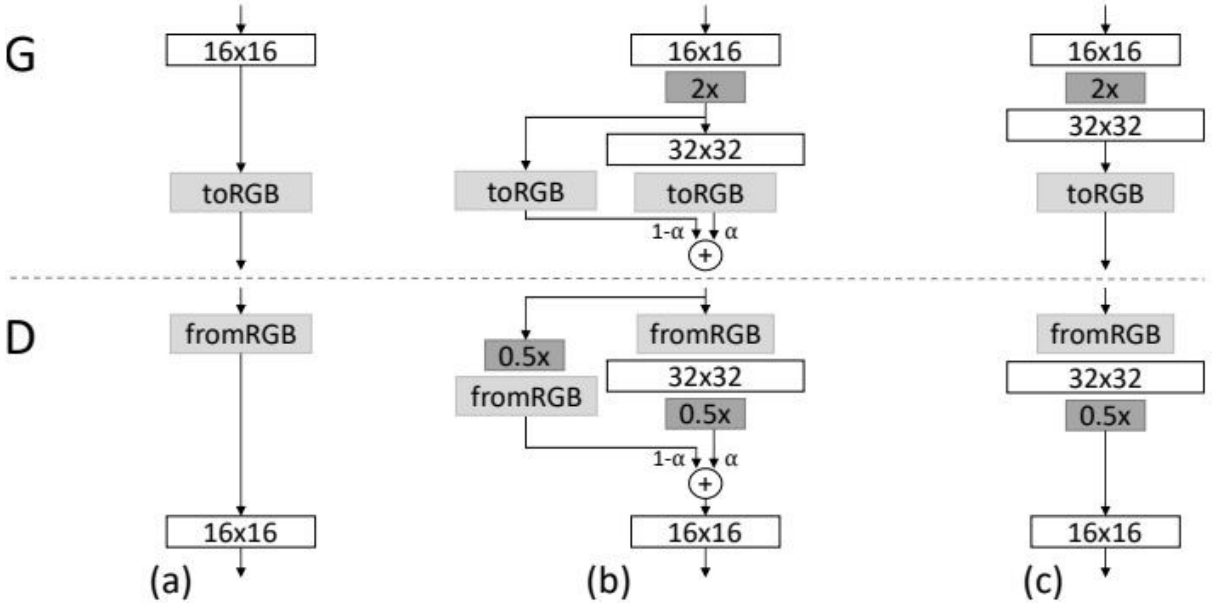
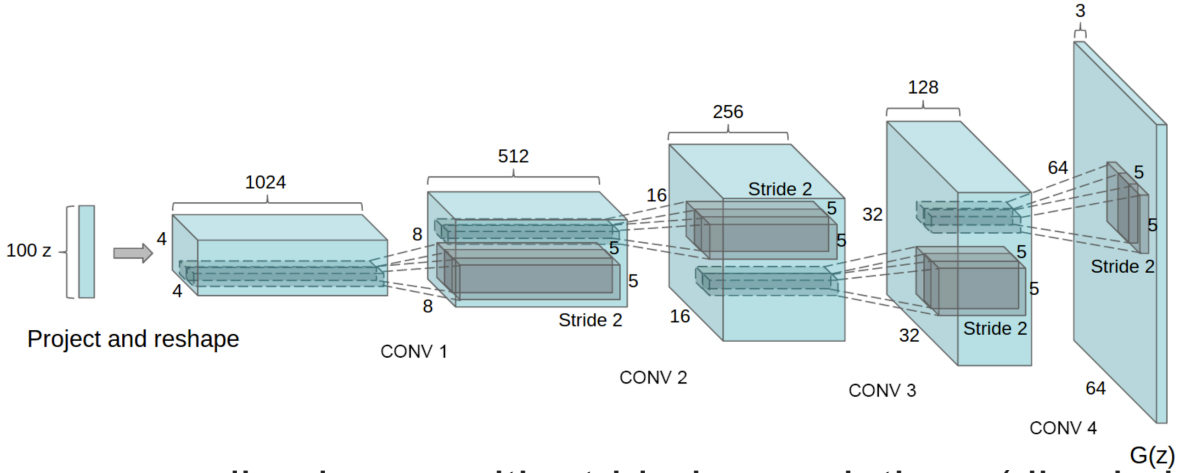


Image source: https://towardsdatascience.com/progressively-growing-gans-9cb795caebee

Module 9 - Section 3

Improved GAN Architectures

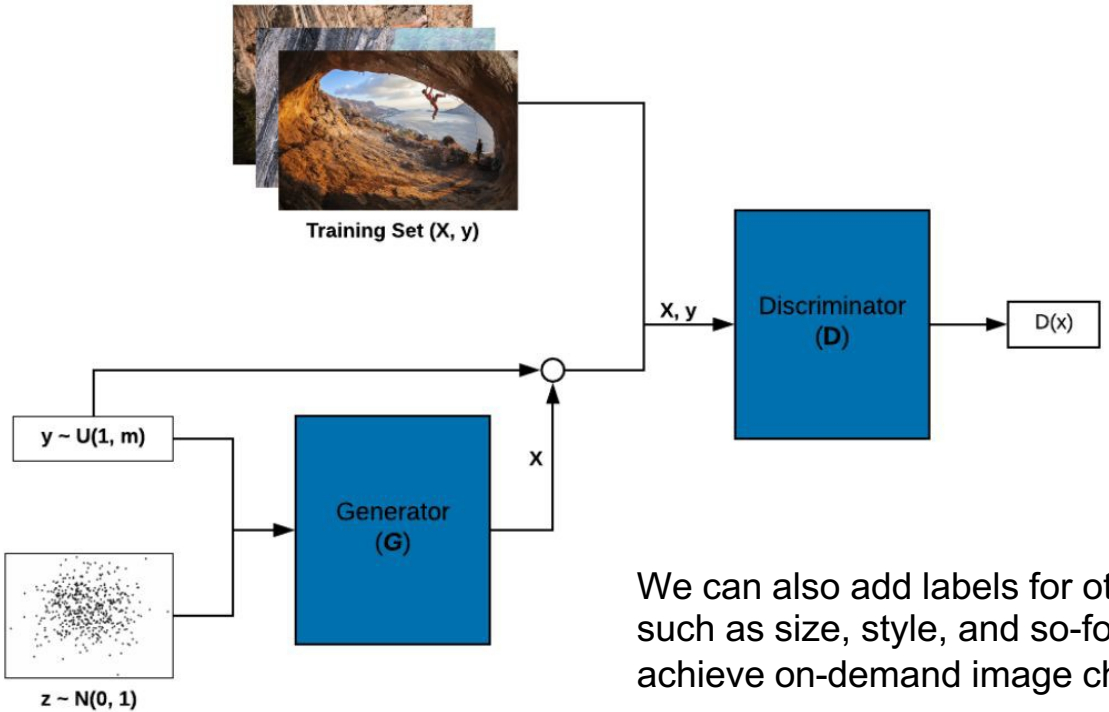
Deep convolutional generative adversarial networks (DCGAN)



- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers

Conditional GANs

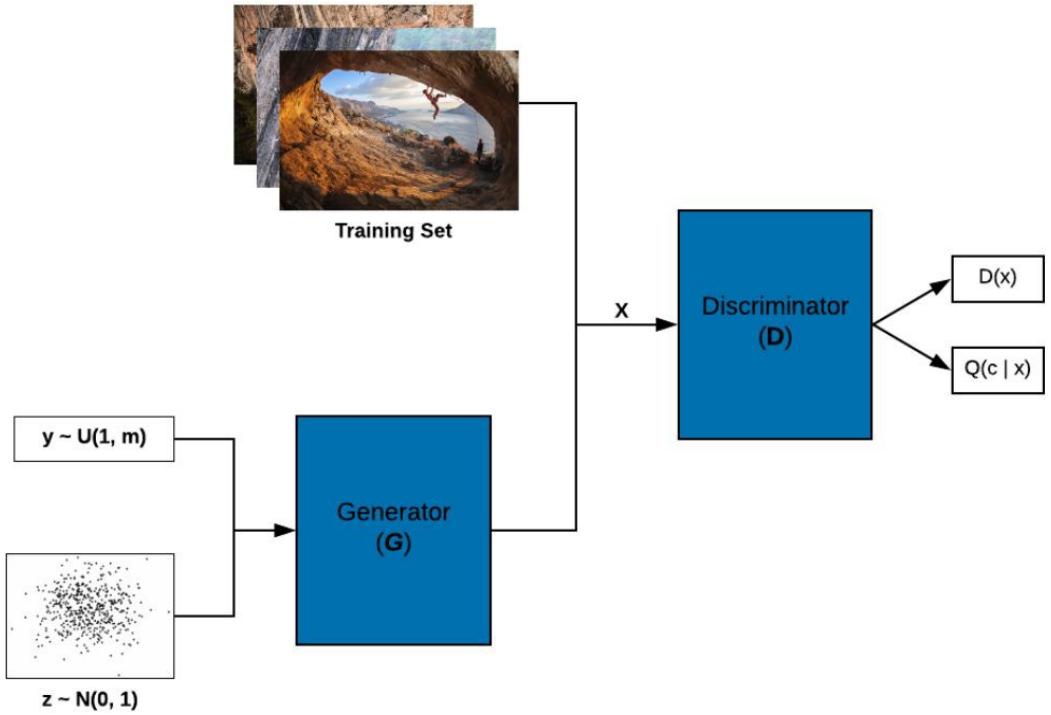
One-hot encoded class labels are passed to **both** the discriminator and the generator. The discriminator gets ‘help’ from the label. The generator must produce any label on demand.



We can also add labels for other attributes, such as size, style, and so-forth. Helps us achieve on-demand image characteristics.

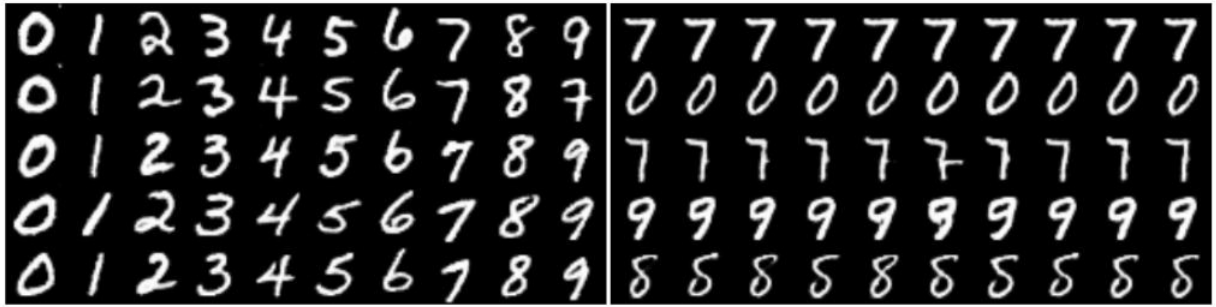
InfoGANs

If we don’t have a labelled training set, we can ask the discriminator to extract labels for us. The generator creates an image with an arbitrary label, and the discriminator must now guess what the label was, in addition to classifying the image as real or fake.



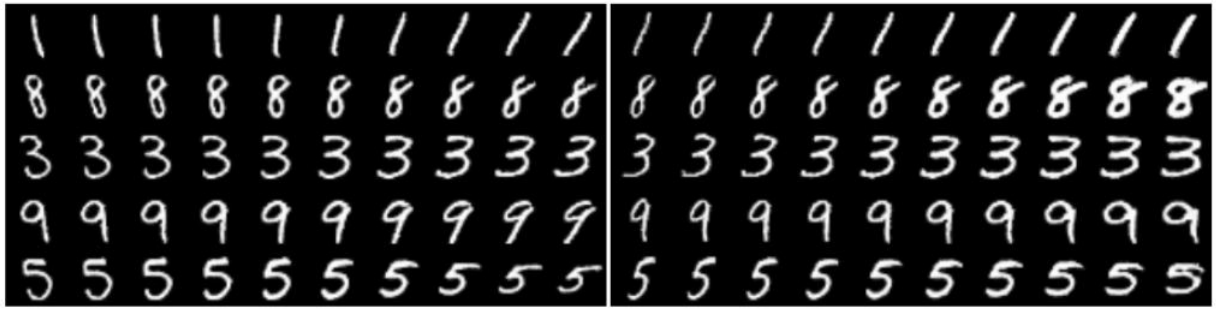
InfoGANs

The labels learned can correspond to class, rotation, width, etc. The trained generator will then be able to generate images that match these labels on-demand.



(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)



(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

Image credit: Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In Advances in neural information processing systems (pp. 2172-2180).]

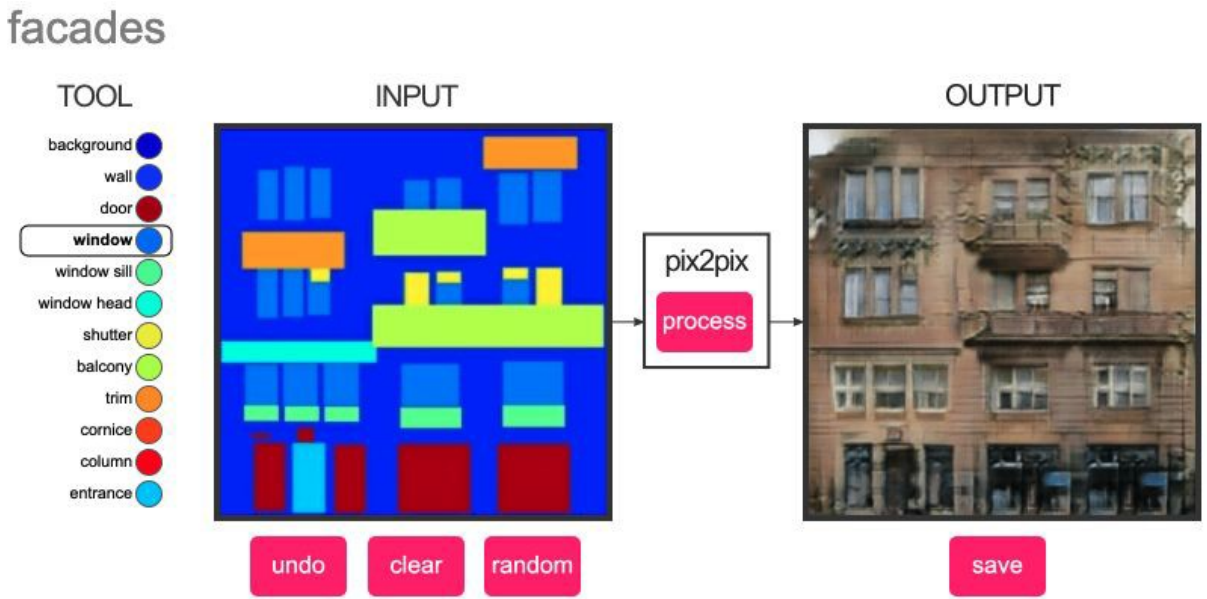
Image-to-Image Translation

Maps image in “Domain A” to images in “Domain B”.
Example: Horse → Zebra
Below: frame-by-frame mapping, applied to video, produced by “CycleGAN”.

Image source: https://junyanz.github.io/CycleGAN/

Image Translation: Pix2Pix Demo

The Pix2Pix model is a precursor to CycleGANs, based on a Conditional GAN. It has access to paired training images (same image presented in both domains A and B).



Try it here: https://affinelayer.com/pixsrv/

CycleGAN

Image-to-image translation *without* labeled data. Unlike Pix2Pix, it can map in *both* directions.

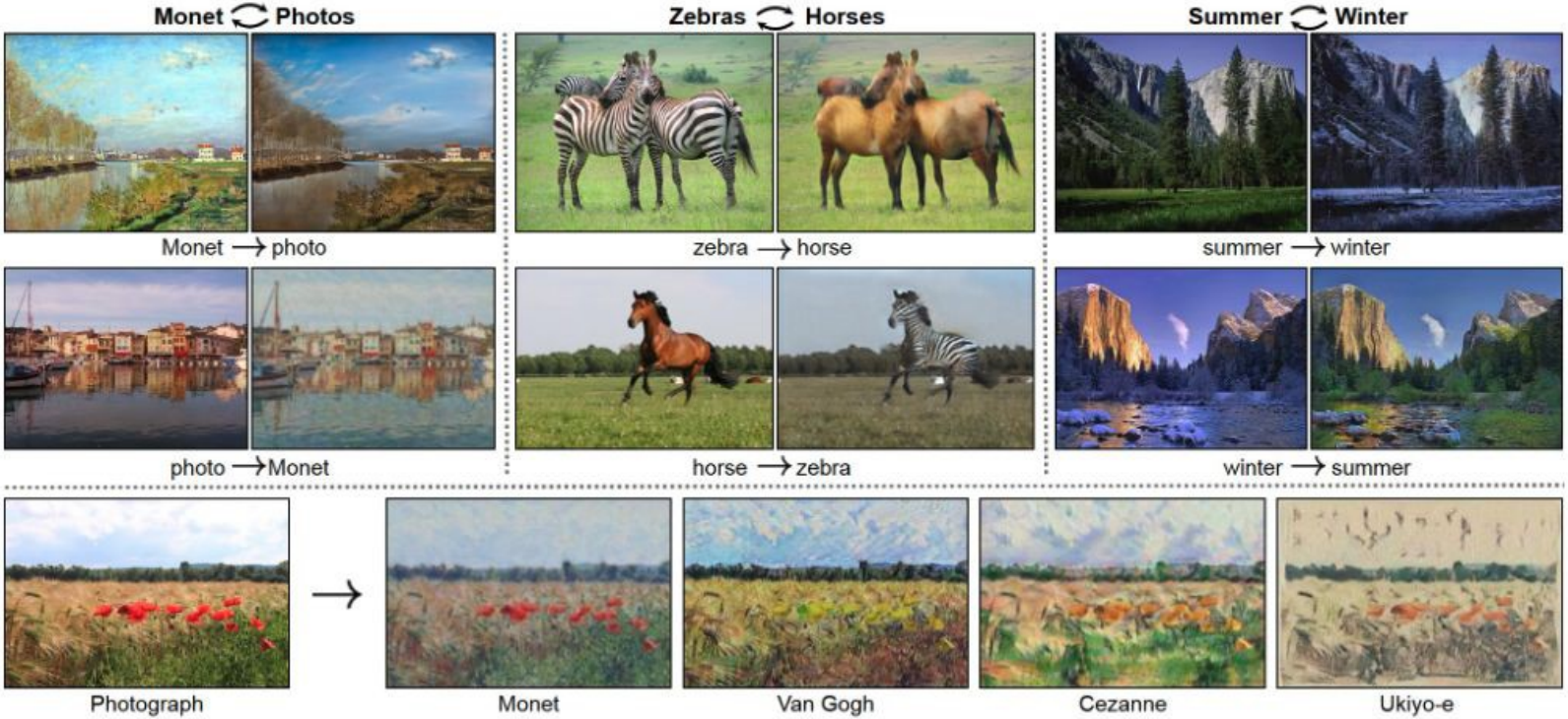
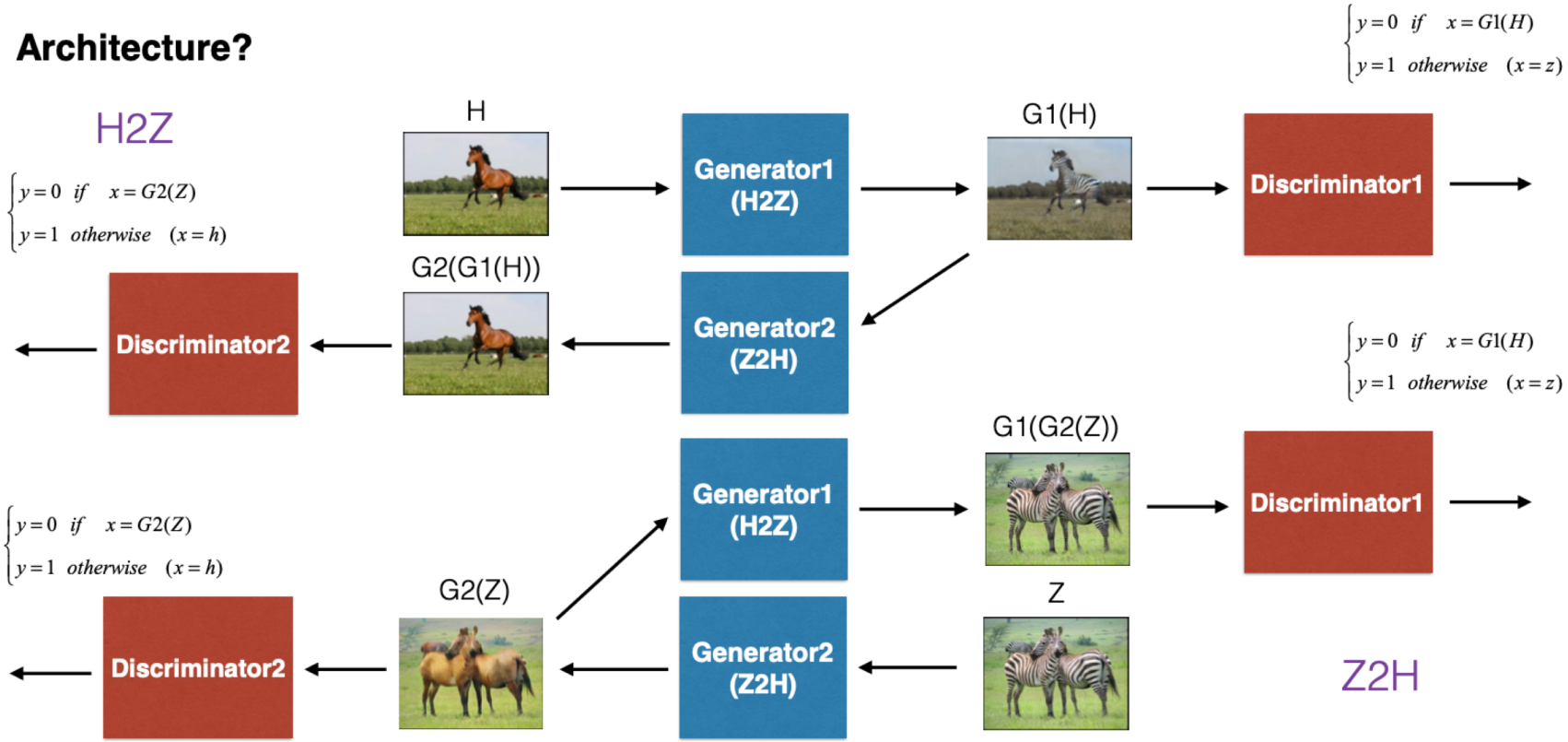


Image source: https://junyanz.github.io/CycleGAN/

Architecture?



[Zhu, Park et al. (2017): Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

Loss to minimize?

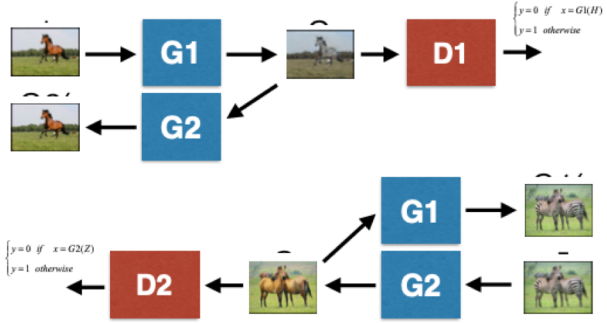
$$J^{(D1)} = -\frac{1}{m_{real}}\sum_{i=1}^{m_{real}}\log(D1(z^{(i)})) - \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1 - D1(G1(H^{(i)})))$$

$$J^{(G1)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D1(G1(H^{(i)})))$$

$$J^{(D2)} = -\frac{1}{m_{real}}\sum_{i=1}^{m_{real}}\log(D2(h^{(i)})) - \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1 - D2(G2(Z^{(i)})))$$

$$J^{(G2)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D2(G2(Z^{(i)})))$$

$$J^{cycle} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\| G2(G1(H^{(i)}) - H^{(i)} \|_1 - \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\| G1(G2(Z^{(i)}) - Z^{(i)} \|_1$$



$$J = J^{(D1)} + J^{(G1)} + J^{(D2)} + J^{(G2)} + \lambda J^{cycle}$$

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

CycleGAN Training

Does not require paired training data. Two sets of generators / discriminators, one for each domain.
Uses **discriminative loss** and **image reconstruction error** (also called “cycle consistency loss”).

Example:
satellite images ←→ route maps

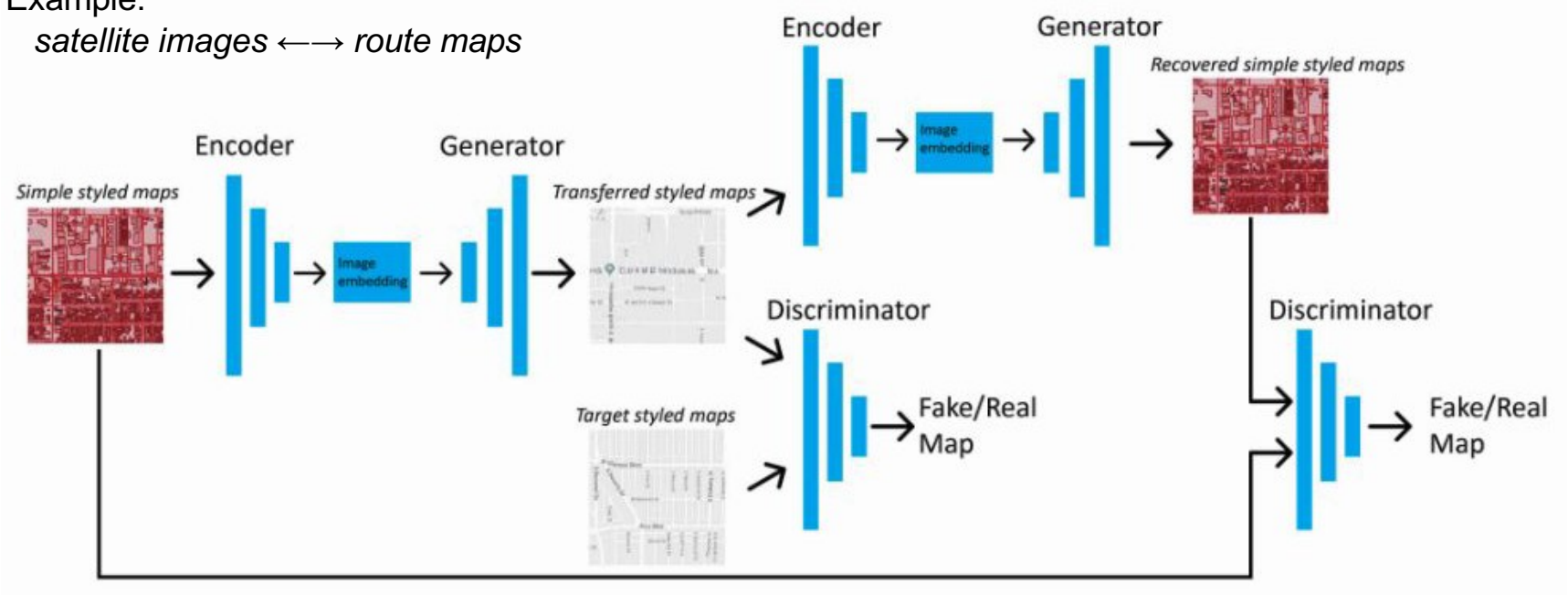


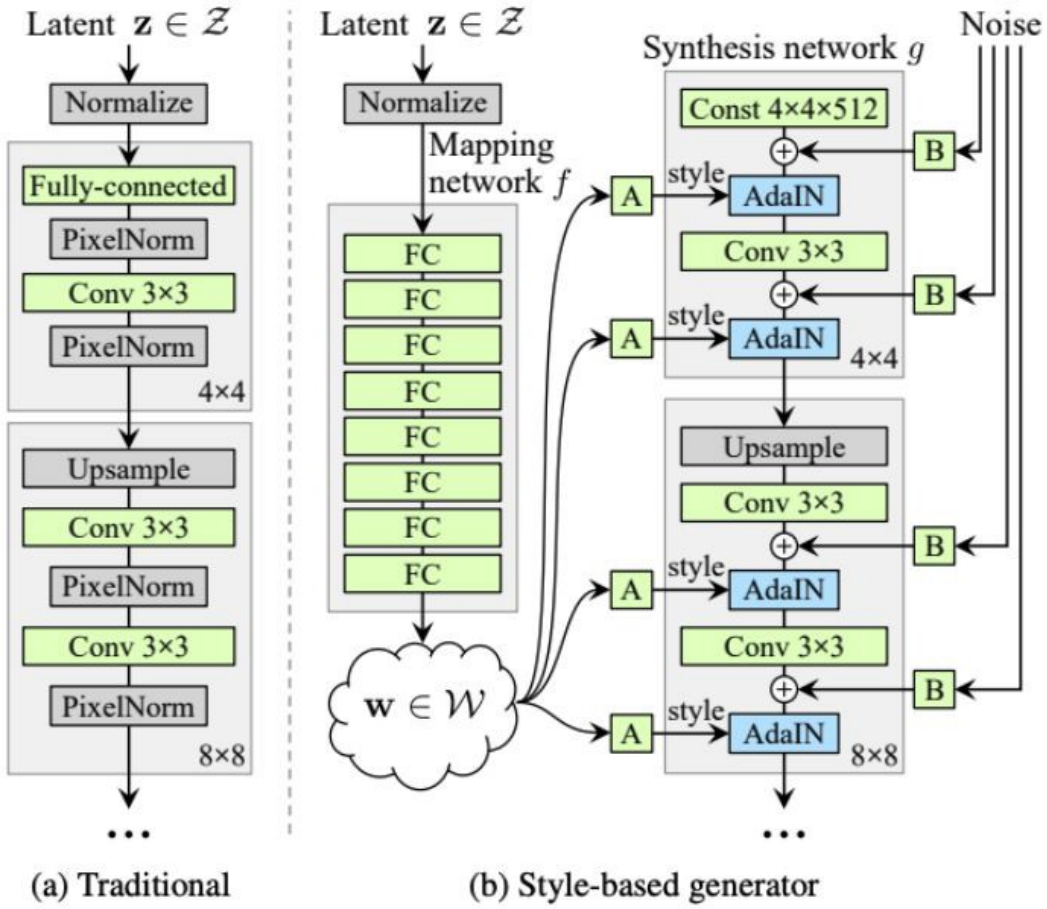
Image source: https://developers.arcgis.com/python/guide/how-cyclegan-works/

StyleGANs

State-of-the-art circa 2018.

Uses style transfer techniques in the generator to ensure the output images have the same local structure as the training images at every scale.

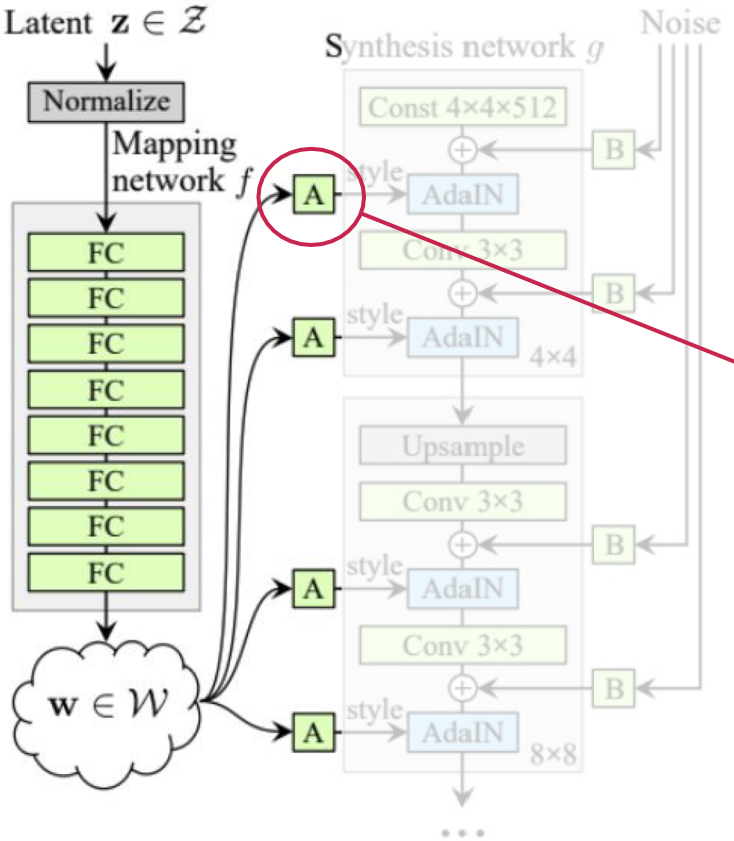
Discriminator and loss are the same as before. Only the generator differs.



StyleGANs

Mapping Network, built from dense layers, turns latent vectors into multiple style vectors.

First produces a global style vector "**w**", which is then injected into each generator layer through an affine transformation "A".



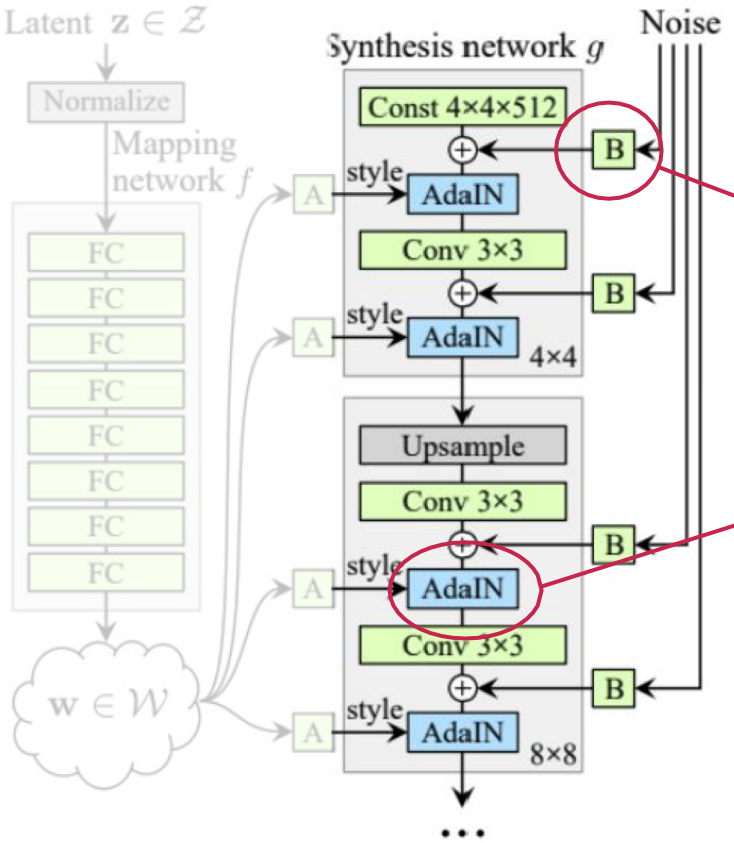
The affine transformation "A" is just a rotation and scaling of **w**, achieved using a dense layer without nonlinearity.

For each feature map in the layer, the transformed style vector will contribute two terms: a scaling term and an offset term.

StyleGANs

Synthesis Network, starts with a constant initial state (learned during training), followed by convolution and upscaling layers.

After each conv layer, we have a noise injection layer followed by an Adaptive Instance Normalization (AdaIN) layer.



Each noise input is a single feature map full of Gaussian noise, scaled using per-feature scaling factors "**B**", and added to all feature maps at that level.

AdaIN layers standardize each feature map independently, and then use the style vector to determine the scale and offset of each feature map.

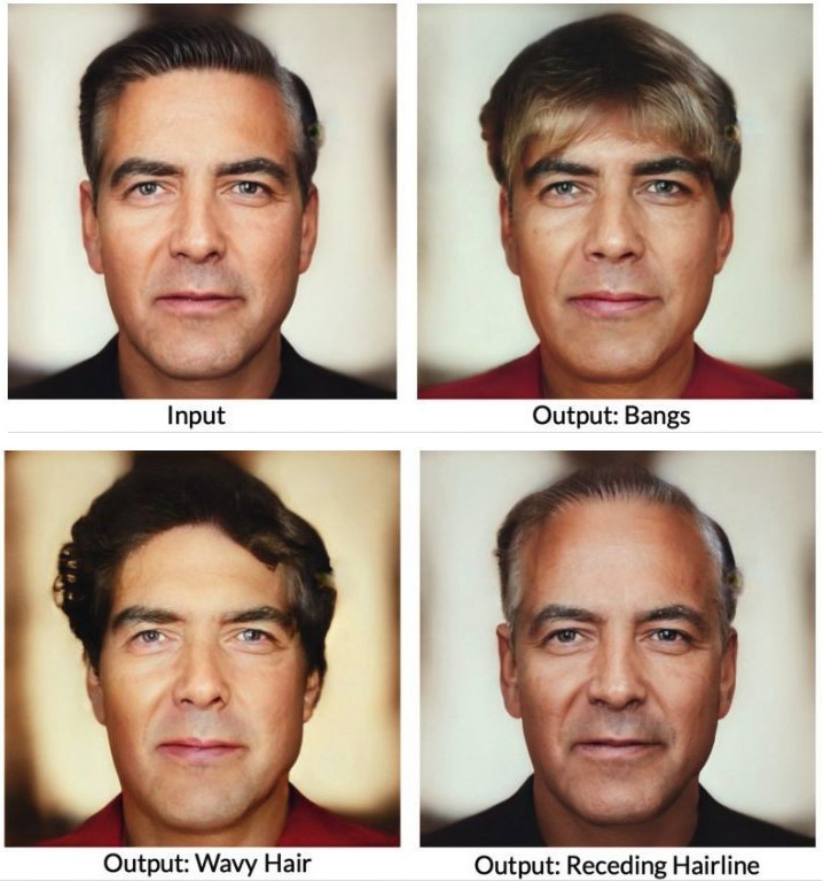
\+ "Mixing regularization"

StyleGAN Demos

Semantic Editing of Hairstyles:
https://www.sfu.ca/~ysa195/projects/CMPT743Project/

Semantic Editing of Faces:
https://hanlab.mit.edu/projects/anycost-gan/

Semantic Editing of Videos:
https://stitch-time.github.io/



Module 9

Resources and Wrap-up

Homework

- Review Module 9 notebook.
- Focus on Final Project; due in two weeks.
  - You must submit by start of Module 11, even if your presentation is scheduled for Module 12.
- Start Assignment 4; due in three weeks.
  - Due by start of Module 12. Since it's due later, you might choose to work on it *after* you submit the Final Project.
-

Next Class

- Speech & Music Recognition and Synthesis
- Preview of notebook will be posted in advance.

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies