Generative Models

IEOR 135 / 290: Applied Data Science with Venture Applications

Alexander Fred-Ojala
Research Director, Data Lab
SCET, UC Berkeley
afo@berkeley.edu

Berkeley
UNIVERSITY OF CALIFORNIA

Pantas and Ting
Sutardja Center
for Entrepreneurship & Technology
Berkeley Engineering

Outline

- Generative models
- Deep Dream
- Neural Style Transfer
- PixelRNNs / PixelCNNs
- Variational Autoencoders (+ Autoencoders)
- Generative Adverserial Networks (GANs)
- Examples + Notebook

Let the computer be an artist!

Supervised vs. Unsupervised

Discriminative vs. Generative

Supervised vs. Unsupervised learning

Supervised learning

Learn function to map input x to output y.

Unsupervised learning

No labels - utilize more data!

Goal: Learn underlying representation of the data.

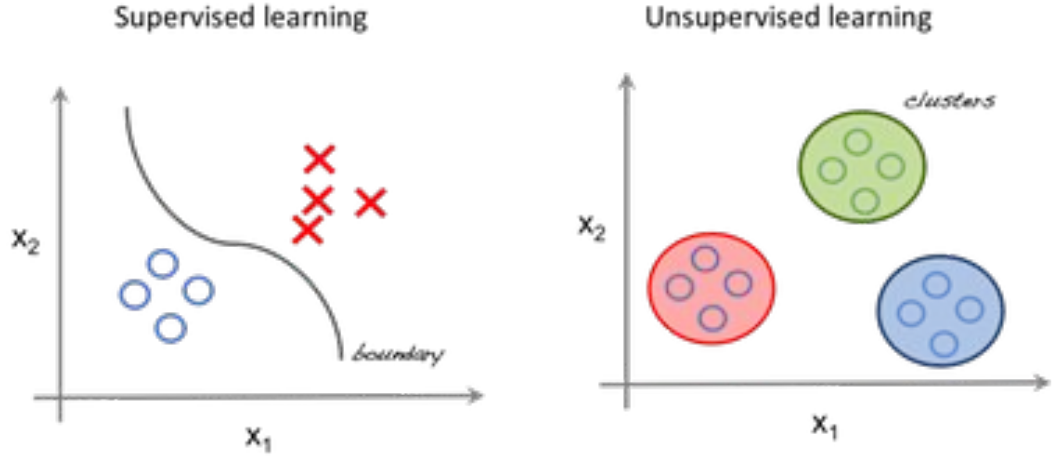


Image source: https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03

Supervised vs. Unsupervised learning

Supervised learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

→ Cat

Classification

Source: Serena Yeung, CS231n, Stanford

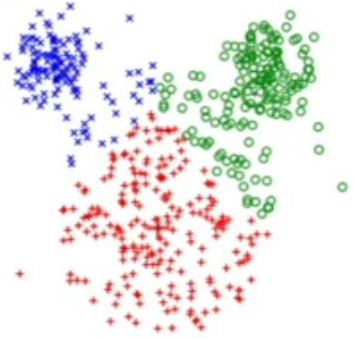
Supervised vs. Unsupervised learning

Unsupervised learning

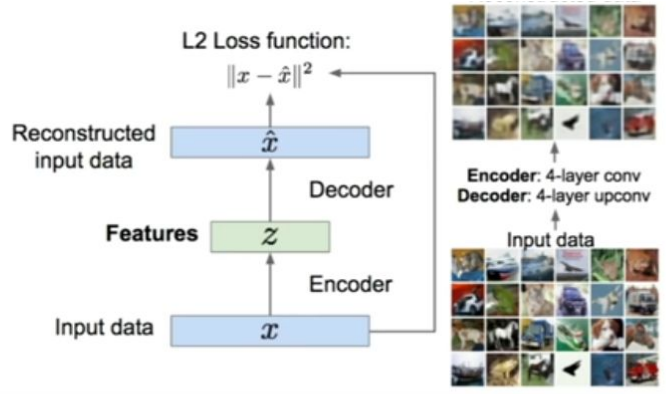
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



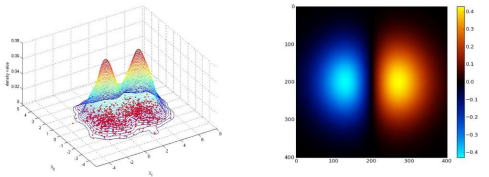
K-means clustering



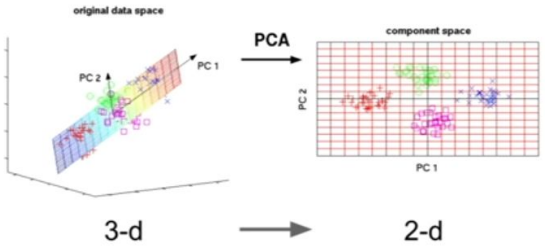
Autoencoders

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation



Principal Component Analysis

Source: Serena Yeung, CS231n, Stanford

Generative vs. Discriminative Algorithms

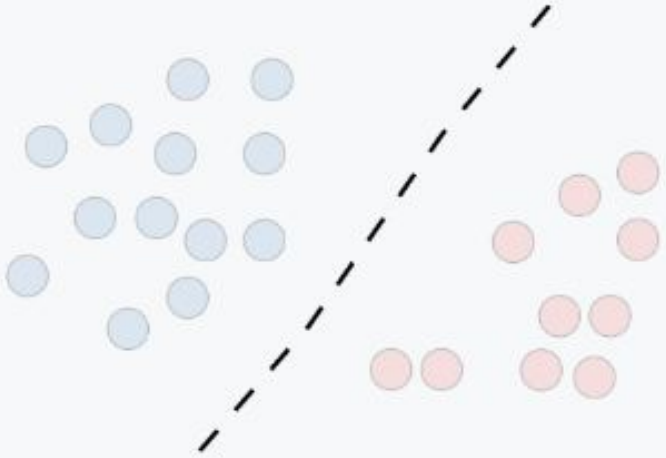
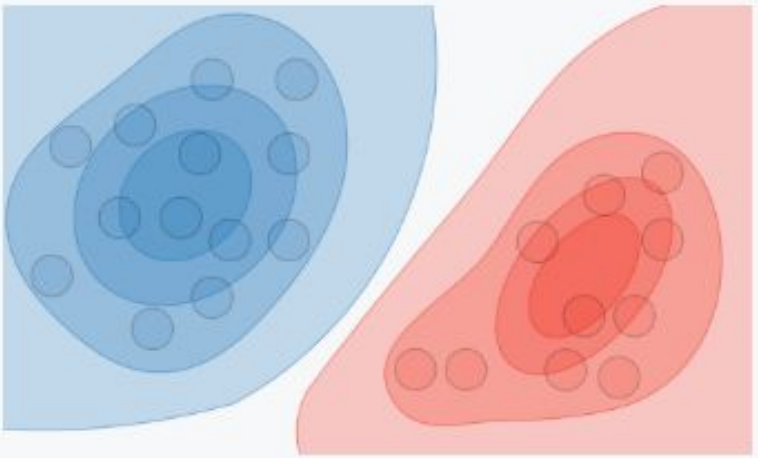
Discriminative algorithms classify input data / learn the boundary between classes. Given input x they predict output y. Model for classifying emails (x) as spam / not spam (y):

p(y|x) *"the probability that an email is spam given the words it contains."*

Generative algorithms model the distribution of individual classes / attempt to predict features given a certain label. Modeling the probability of x given y:

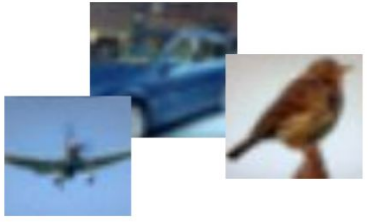
p(x|y) *"assuming this email is spam, how likely are these words? "*

Generative vs. Discriminative Algorithms

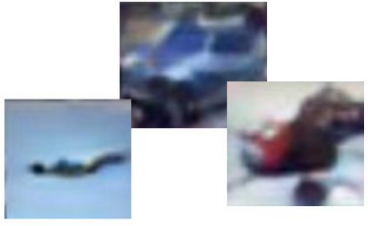
	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		

Generative Models

Given training data, generate new samples from same distribution



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



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

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

Addresses density estimation, a core problem in unsupervised learning

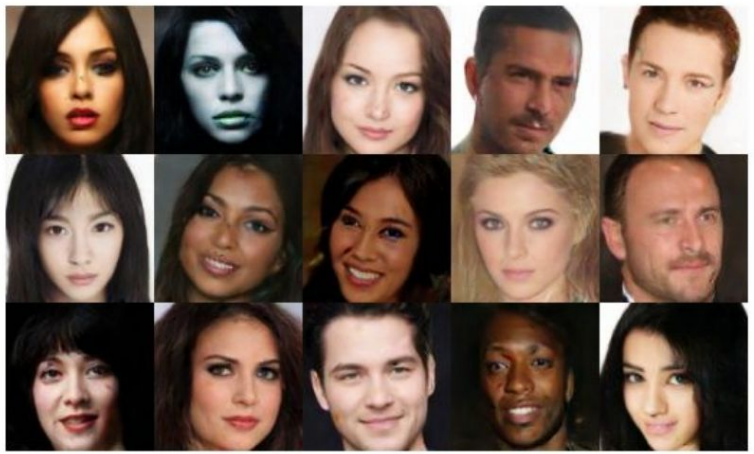
Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

Source: Serena Yeung, CS231n, Stanford

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)

Source: Serena Yeung, CS231n, Stanford

Generative Models

Models that are able to produce / generate new content:

- Generate text
- Generate images
- Generate audio
- Generate number
- Generate distributions
- Generate music notes
- Generate movement

.....

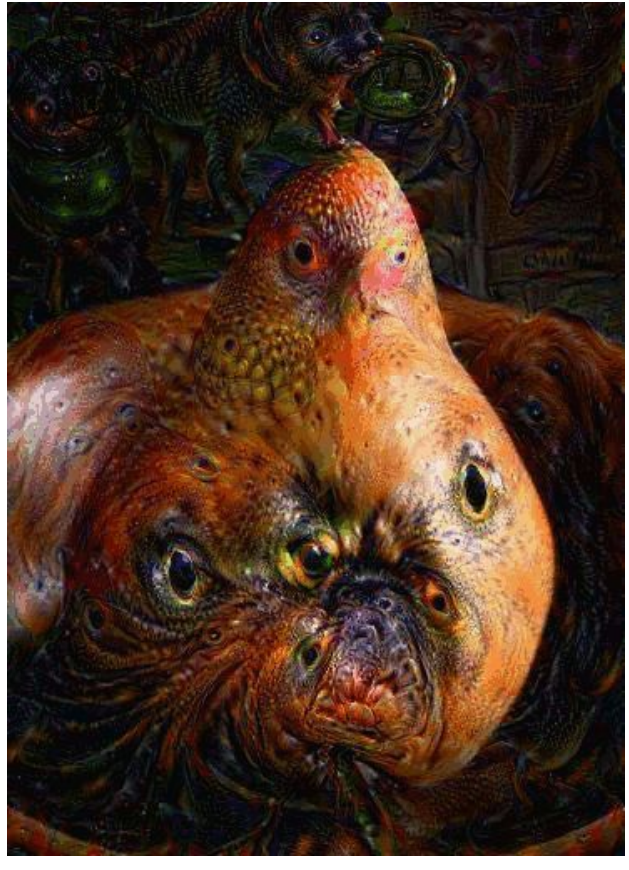
SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Example from Open-AIs Transformer model GPT2

Deep Dream

Deep Dream (July 2015)

Uses a CNN to find and enhance patterns in images to create dream-like images

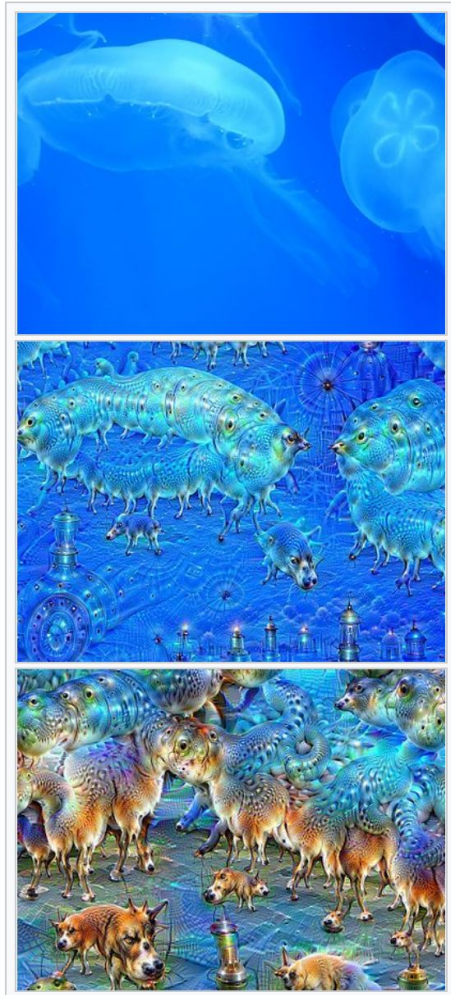


Deep Dream

How it works

- Take a pretrained CNN (e.g. VGG16 trained on ImageNet)
- Setup a gradient ascent function, maximizing the value of (arbitrary) activation maps in the bottom layer
- Run the network in reverse, adjusting pixel values in the input image every iteration
- After several iterations psychedelic and surreal images are generated algorithmically.

It's like backprop, but we don't adjust the network weights. Instead the weights are held fixed and the input is adjusted.



The original image (top) after applying ten (middle) and fifty (bottom) iterations DeepDream, the network having been trained to perceive dogs

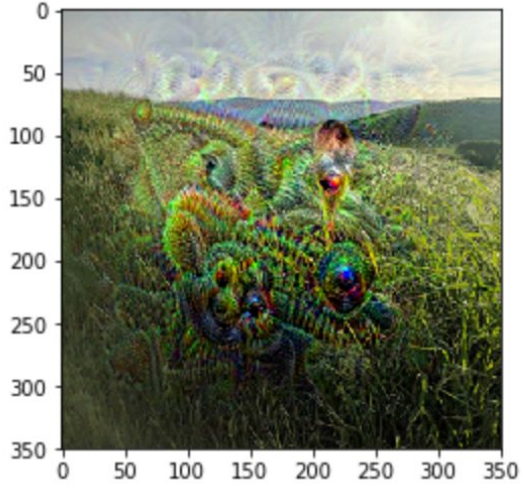
Deep Dream

Example of generating deep dream images:

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.2-deep-dream.ipynb

```
from matplotlib import pyplot as plt

plt.imshow(deprocess_image(np.copy(img)))
plt.show()
```



Neural Style Transfer

Neural Style Transfer (Aug 2015)

Use Deep Learning to manipulate digital images / videos to adopt the style of another image.

Common uses:

Transfer one image style onto another.

Neural Style Transfer

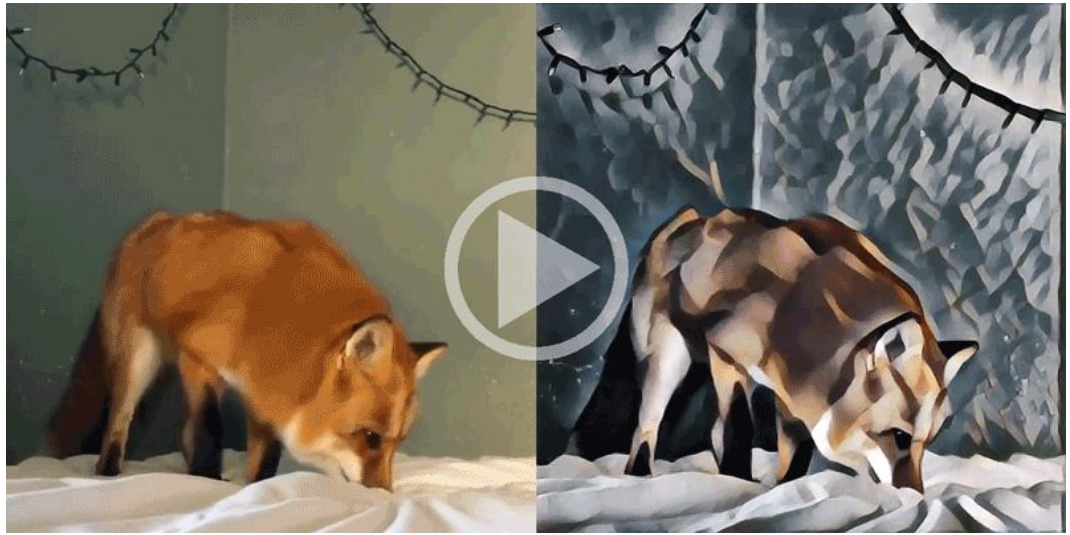
How it works

- Take a pretrained CNN (e.g. VGG19 trained on ImageNet)
- Setup a loss function that can capture the content loss (from original image) and the style loss (from style image). Distance can be L2 norm.

```
loss = distance(style(reference_image) - style(generated_image))
 +     distance(content(original_image) - content(generated_image))
```

- Content aims at preserving high-level layer activations between the original image and the generated image.
- Style maintains similar correlations within activations for both low-level layers and high-level layers, between style image and original image.

Neural Style Transfer



Neural Style Transfer

Example of using Neural Style Transfer with Tensorflow 2 to create a Kadinsky turtle:

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/r2/tutorials/generative/style_transfer.ipynb

Image of Green Sea Turtle -By P.Lindgren CC BY-SA 3.0, from Wikimedia Common

Now how would it look like if Kandinsky decided to paint the picture of this Turtle excl

Take aways

- Style transfer consists in creating a new image that preserves the "contents" of a target image while also capturing the "style" of a reference image.
- "Content" can be captured by the high-level activations of a convnet.
- "Style" can be captured by the internal correlations of the activations of different layers of a convnet.
- Hence deep learning allows style transfer to be formulated as an optimization process using a loss defined with a pre-trained convnet.
- Starting from this basic idea, many variants and refinements are possible!

PixelRNN / PixelCNN

Fully visible belief network

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data

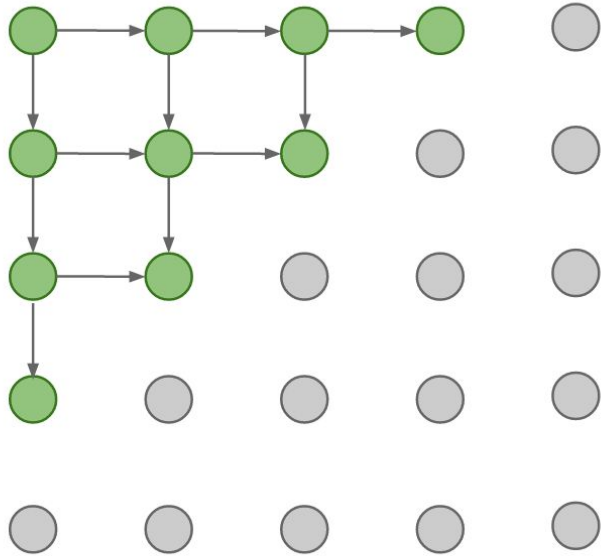
Source: Serena Yeung, CS231n, Stanford

Pixel RNNs

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!



Source: Serena Yeung, CS231n, Stanford

Pixel CNNs

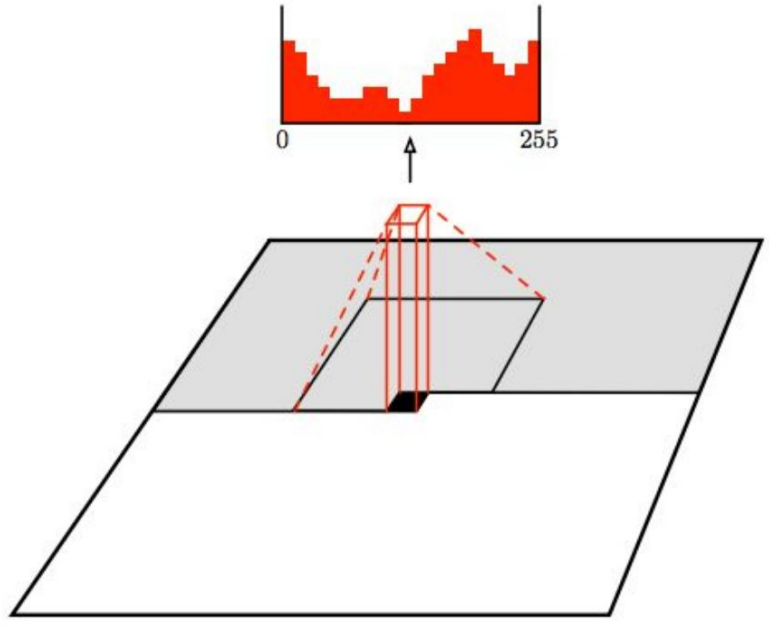
Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$$

Softmax loss at each pixel



Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially
=> still slow

Source: Serena Yeung, CS231n, Stanford

Autoencoders

Autoencoders

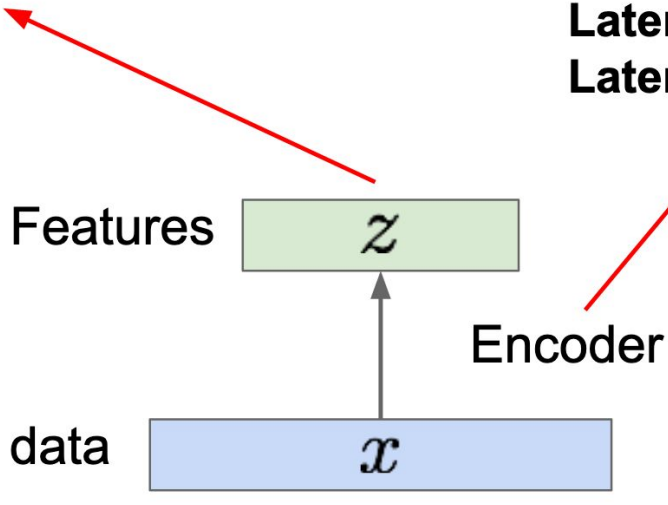
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

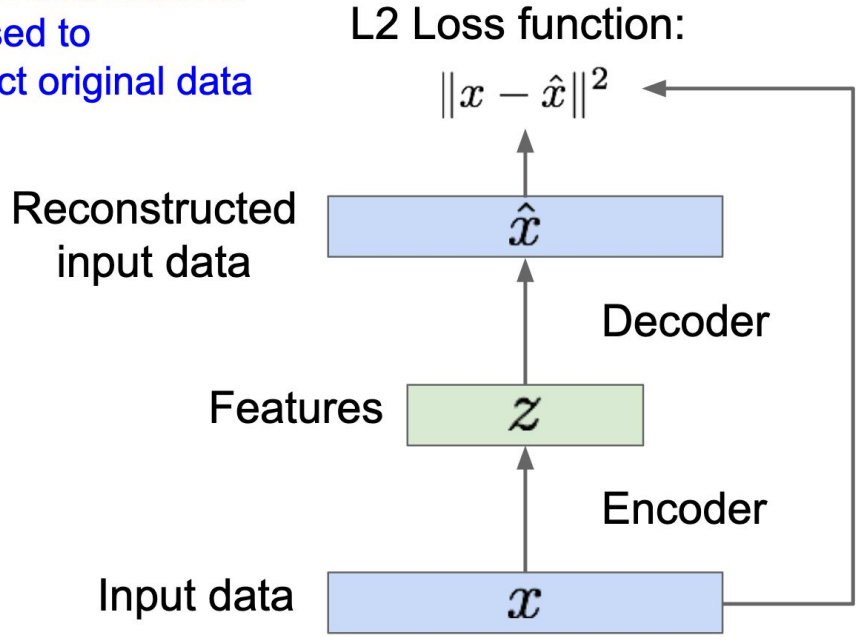
Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

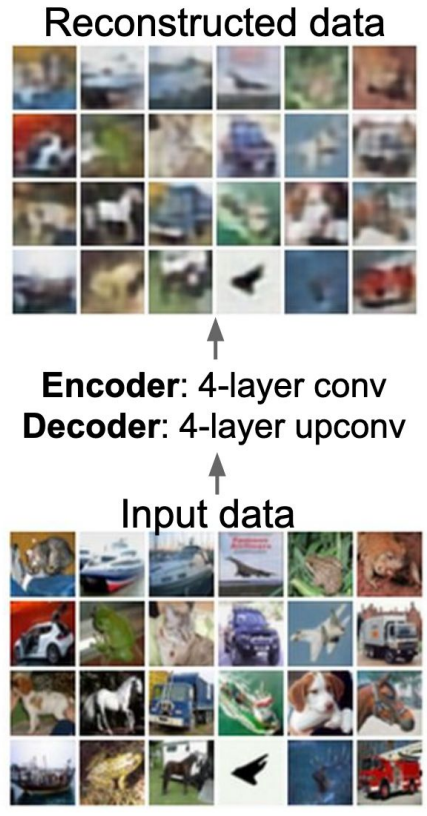


Source: Serena Yeung, CS231n, Stanford

Autoencoders

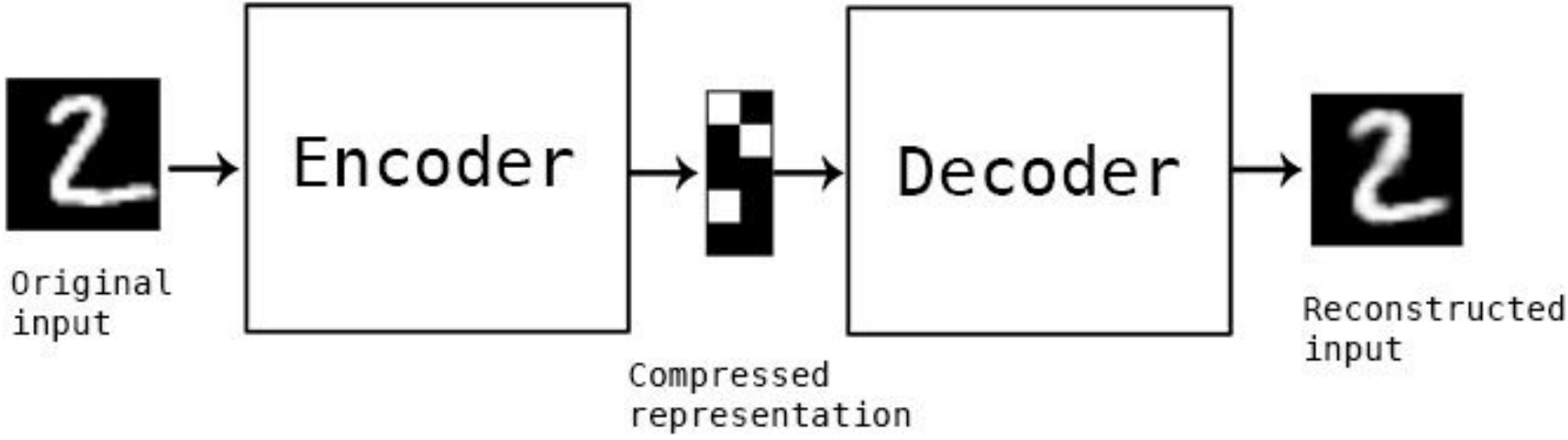
Train such that features can be used to reconstruct original data





Source: Serena Yeung, CS231n, Stanford

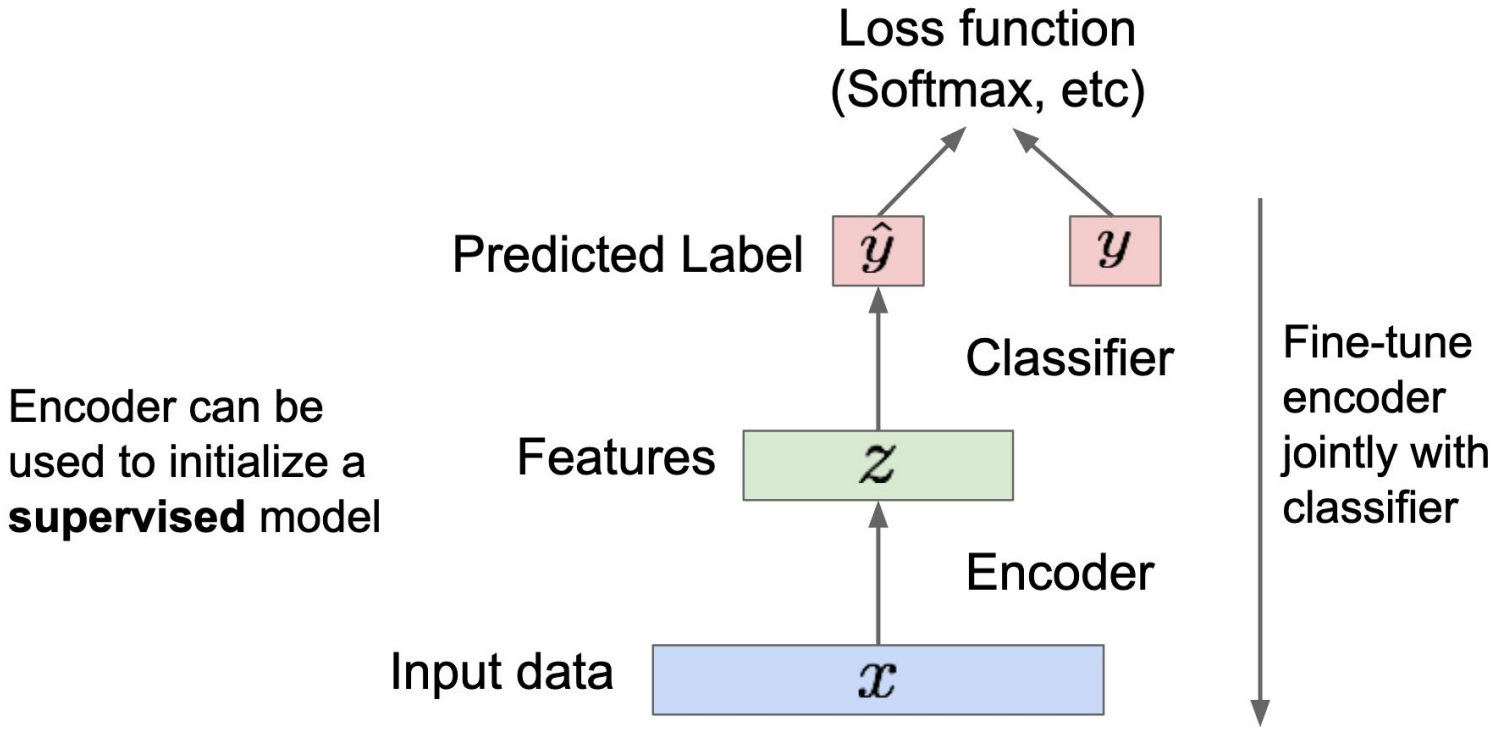
Autoencoders



Keras.io

Autoencoders

After training throw away decoder and instead initialize supervised model



Source: Serena Yeung, CS231n, Stanford

Variational Autoencoders

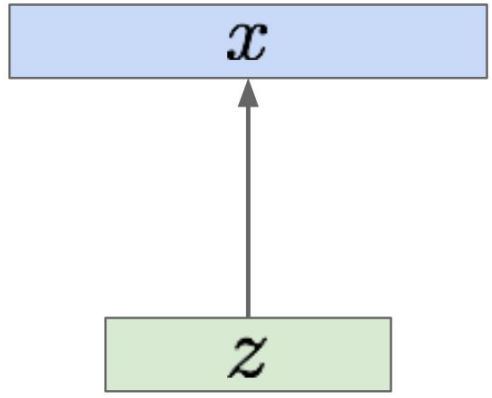
Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$



Intuition (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

Source: Serena Yeung, CS231n, Stanford

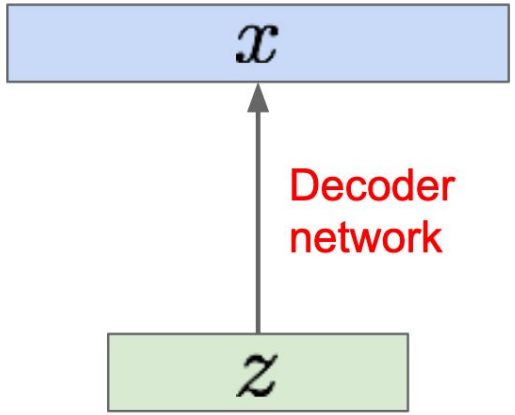
Variational Autoencoders

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

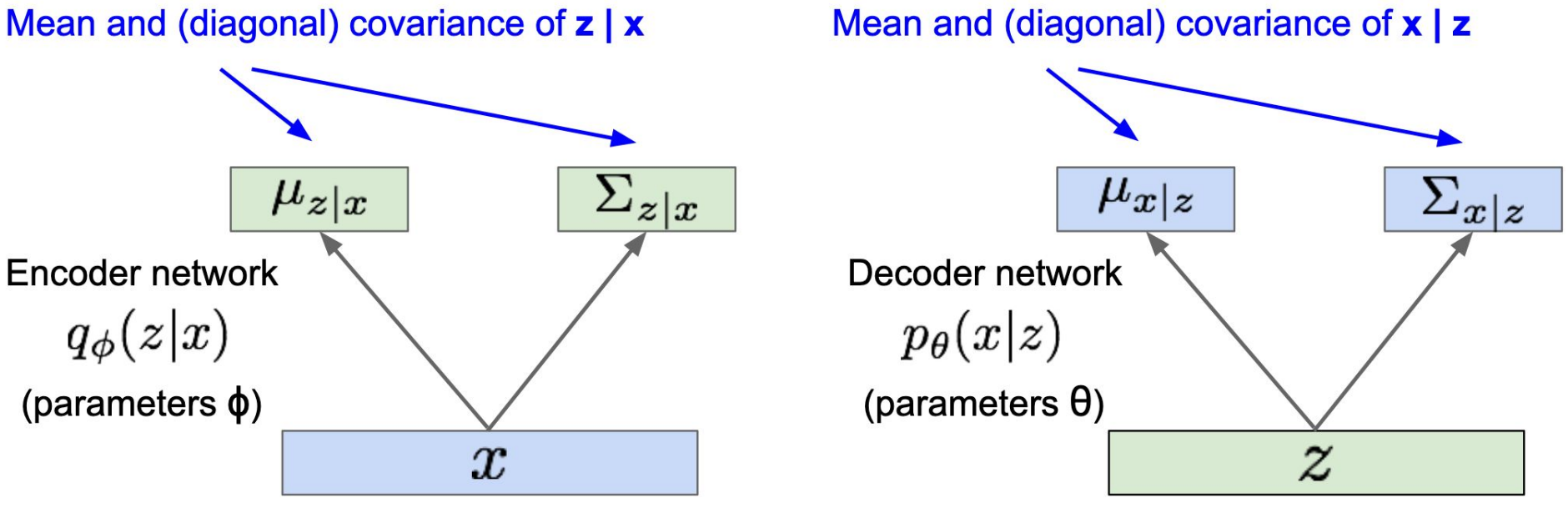
Choose prior p(z) to be simple, e.g. Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network

Source: Serena Yeung, CS231n, Stanford

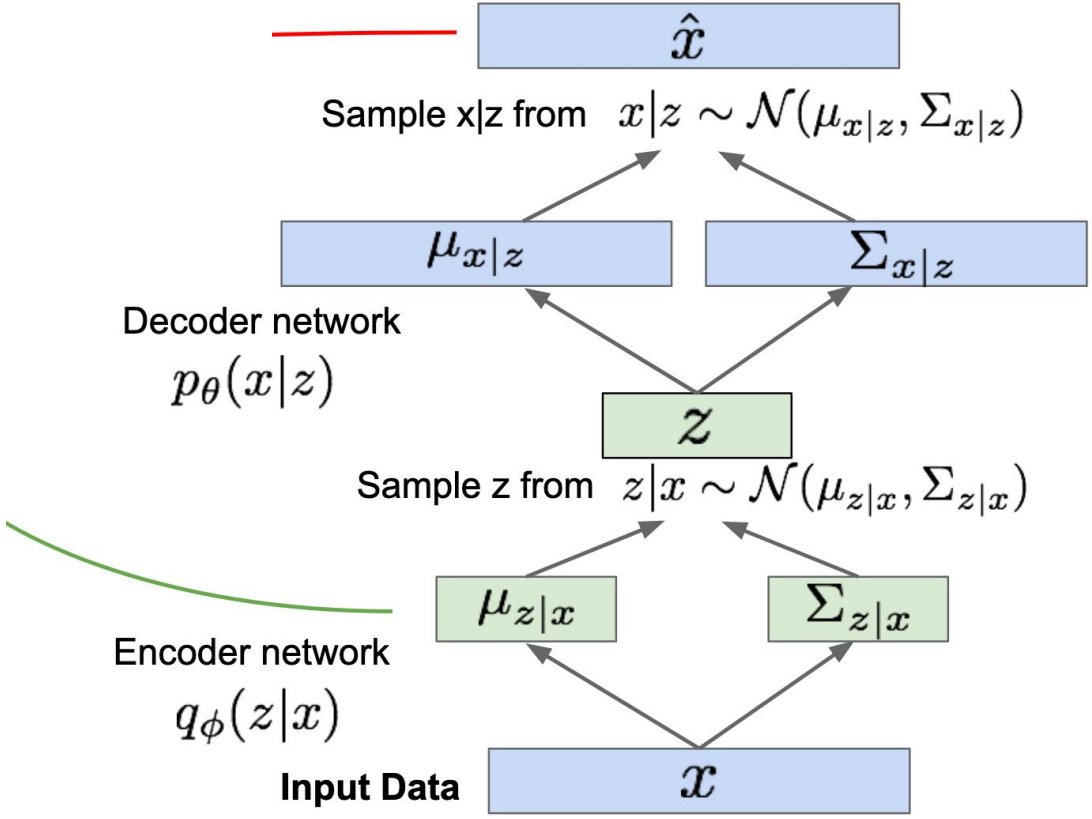
Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



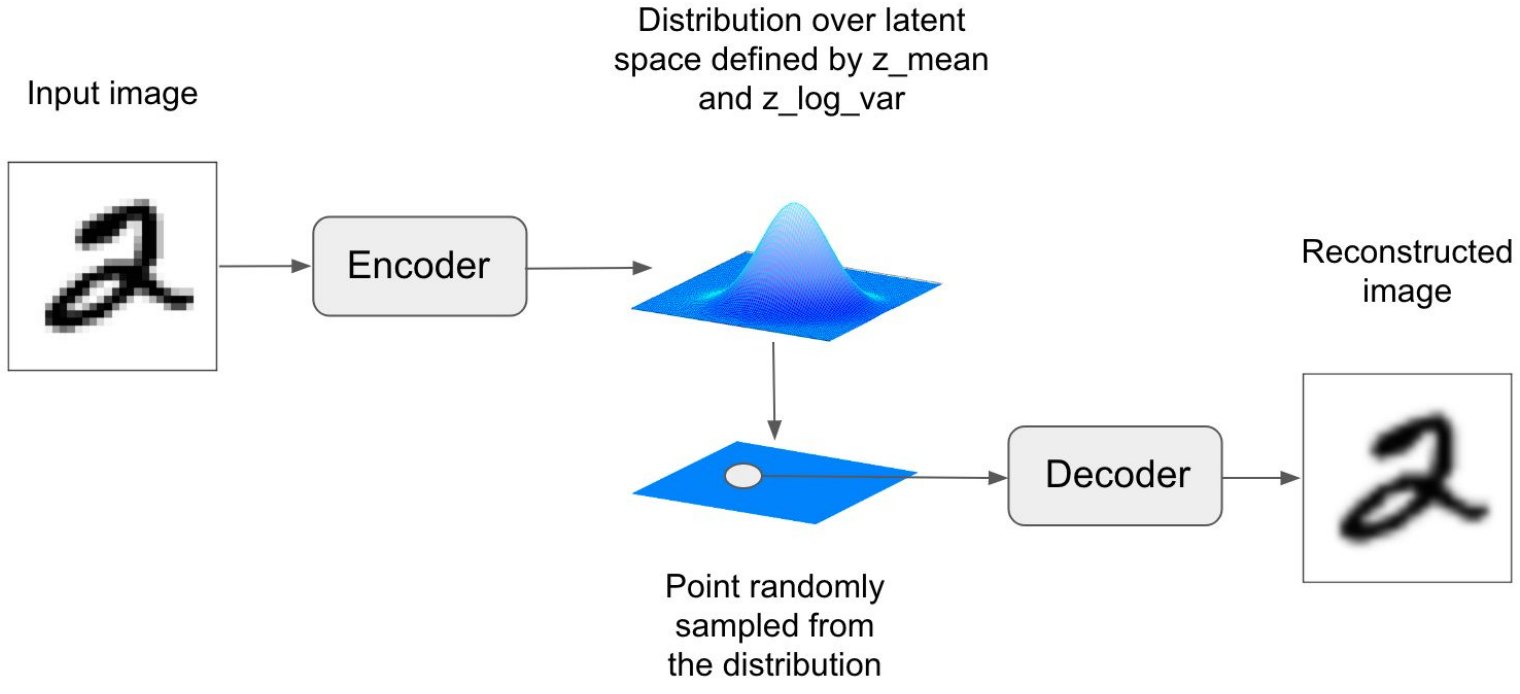
Source: Serena Yeung, CS231n, Stanford

Variational Autoencoders



Source: Serena Yeung, CS231n, Stanford

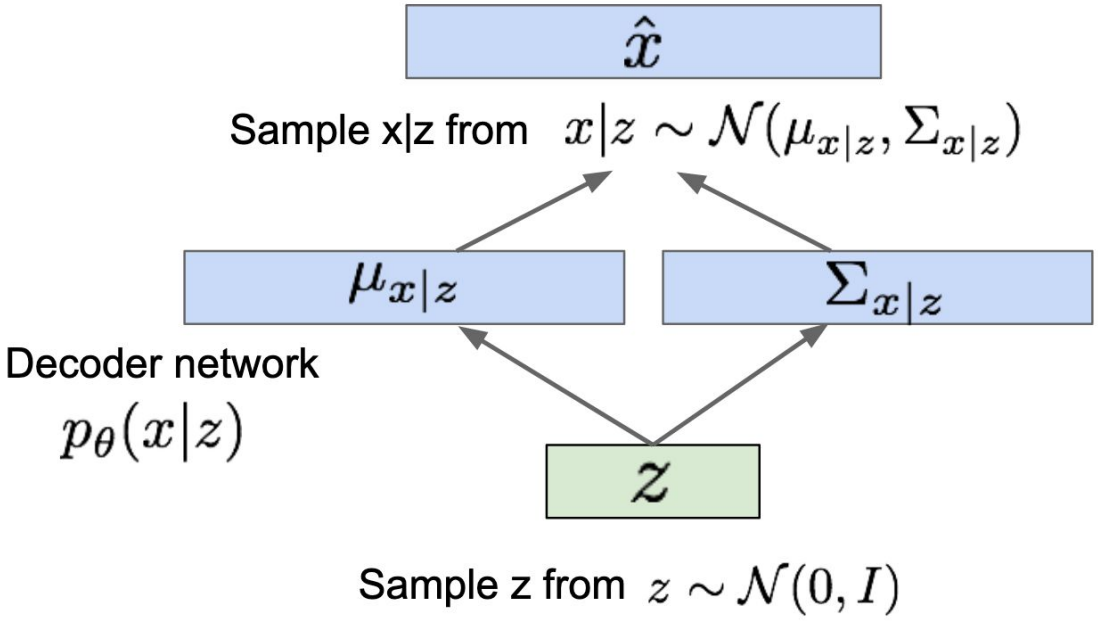
Variational Autoencoders



Source: keras.io

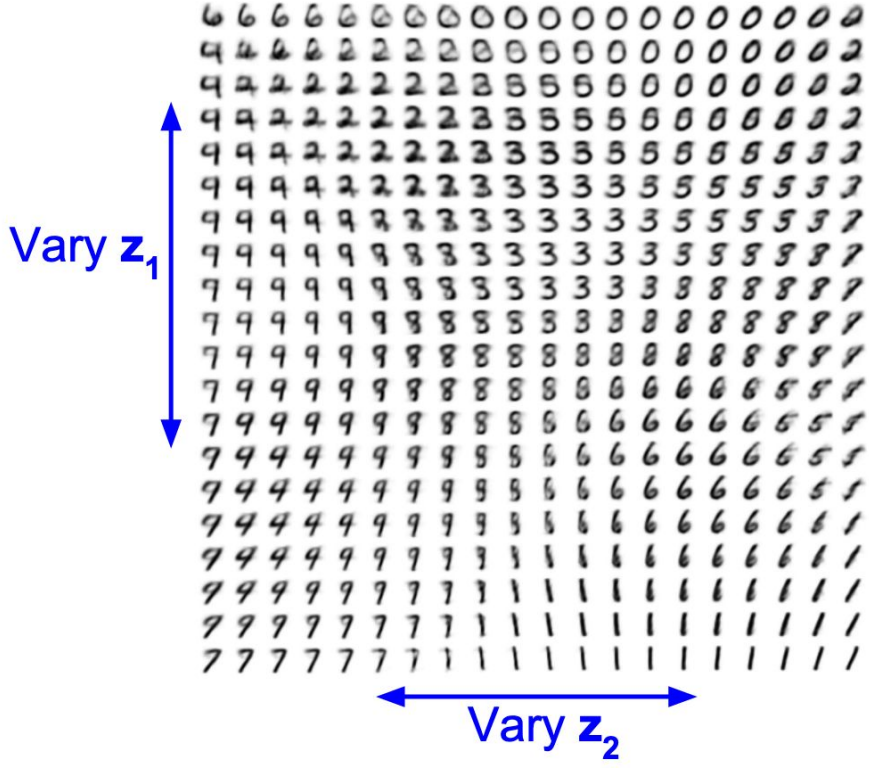
Variational Autoencoders

Use decoder network. Now sample z from prior!



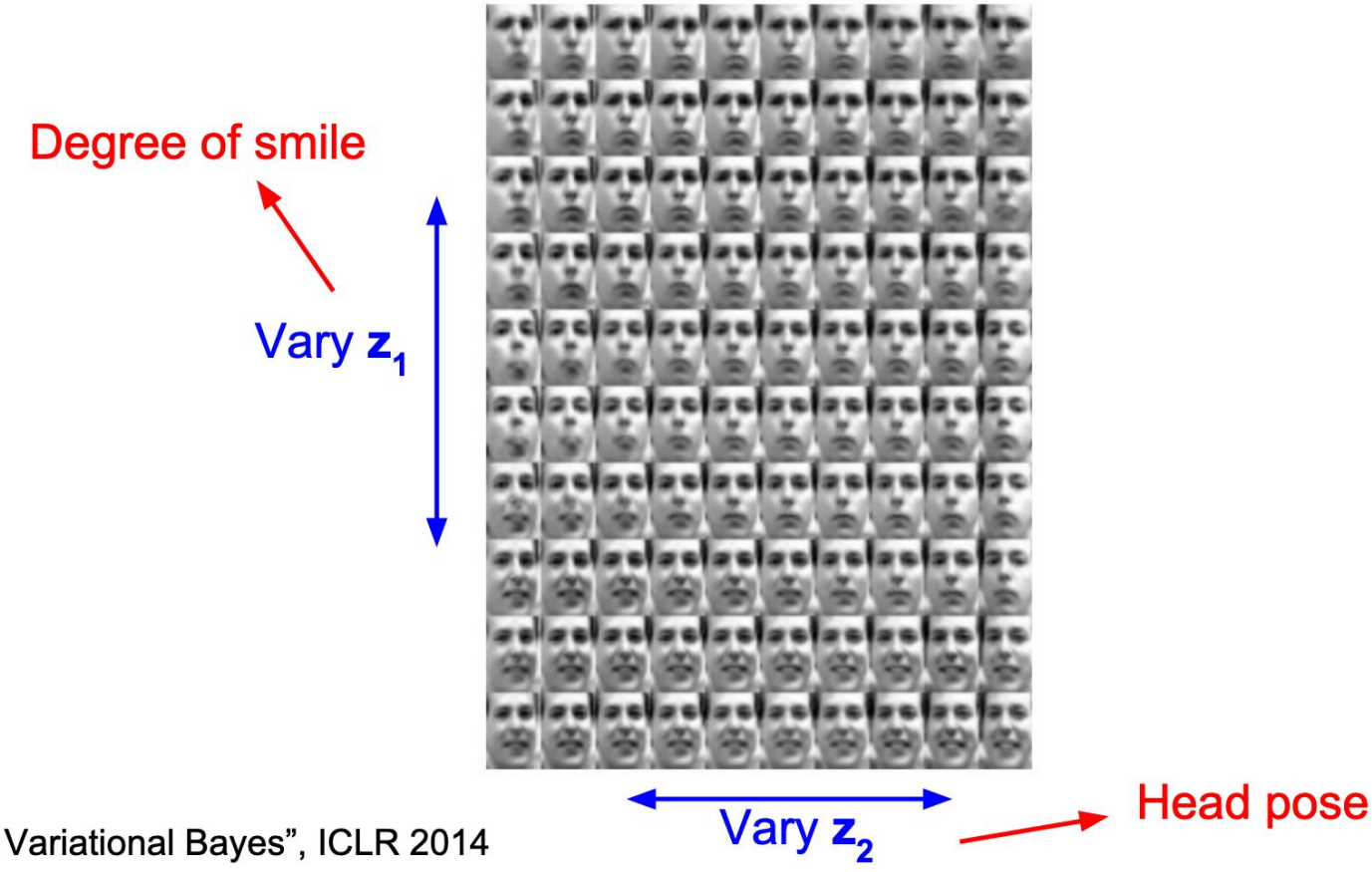
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Data manifold for 2-d **z**



Variational Autoencoders

Different dimensions of **z** encode interpretable factors of variation

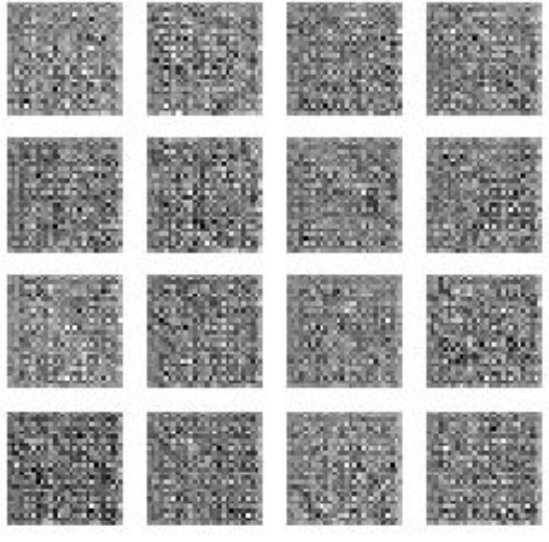


Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Convolutional Variational Autoencoder

Generate MNIST images with VAEs in Tensorflow 2.0

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/r2/tutorials/generative/cvae.ipynb



GANs

GANs History

Generative Adversarial Networks (GANs) were developed in 2014 by by Ian Goodfellow, Yoshua Bengio et al.

"The most interesting idea in the last 10 years in Machine Learning"

- Yann LeCun

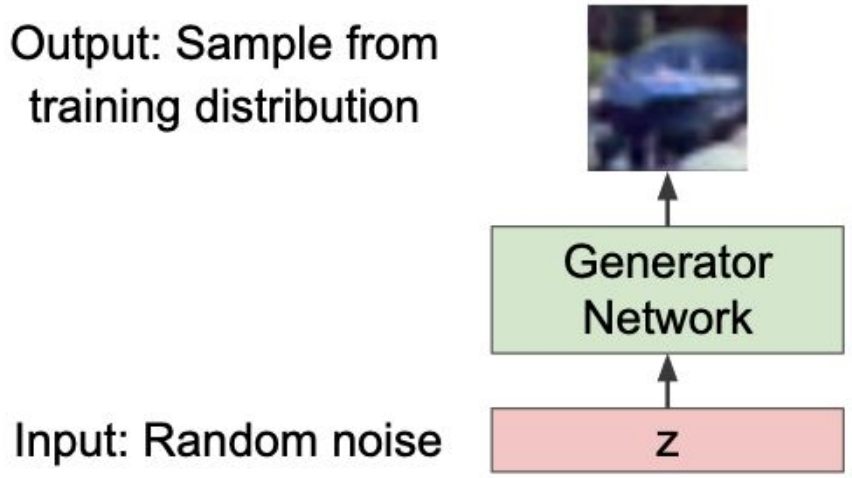
GANs

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!



Source: Serena Yeung, CS231n, Stanford

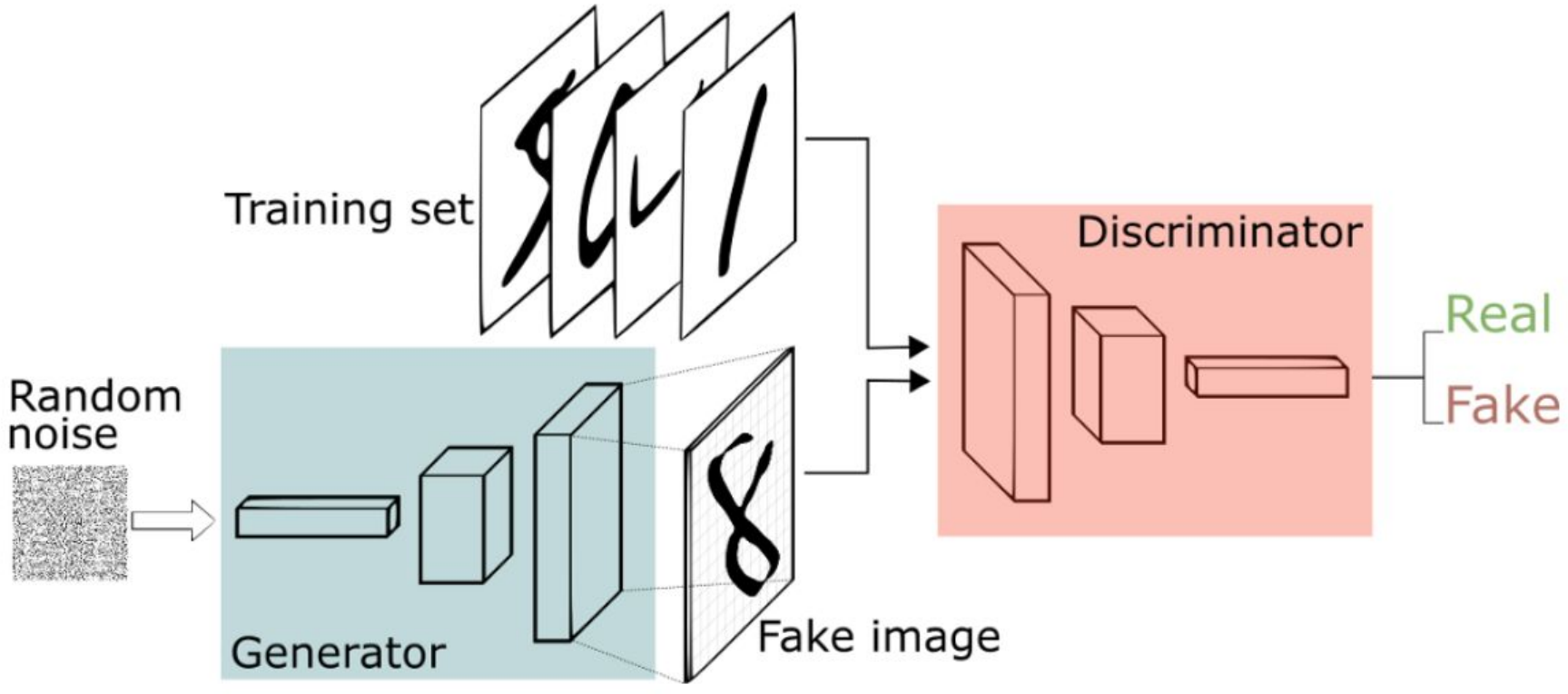
Consists of Two Functions

*The main idea is to have two separate Neural Networks, a **generator** and a **discriminator**, locked in a competition with each other.*

- **Generator:** Creates output similar to the one in the dataset
- **Discriminator:** Tries to understand if they are real or not

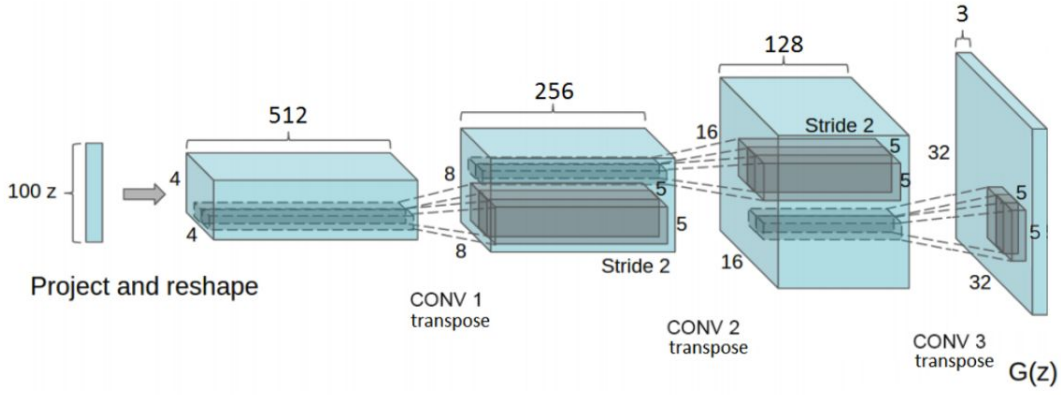
Metaphor: The generator function is a money launderer and the discriminator is a police. Cycle of feedback, until the counterfeiter is better than the police.

GAN Systemic Architecture



Generative Adversarial Network framework.

GAN Convolutional Generator



Adapted from the DCGAN paper. The Generator network implemented here. Note the non-

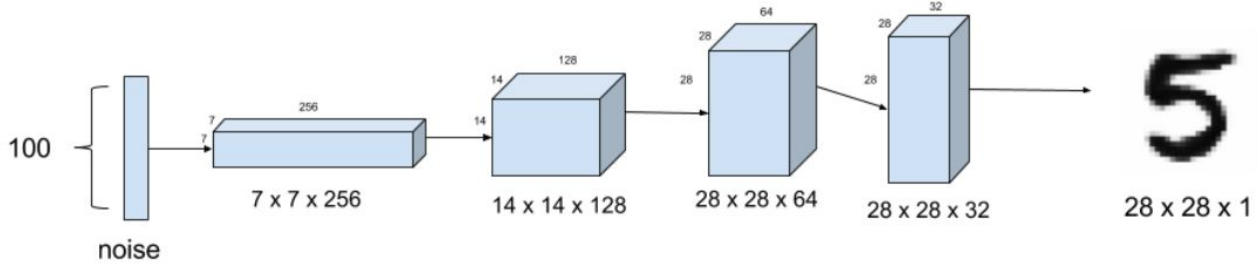
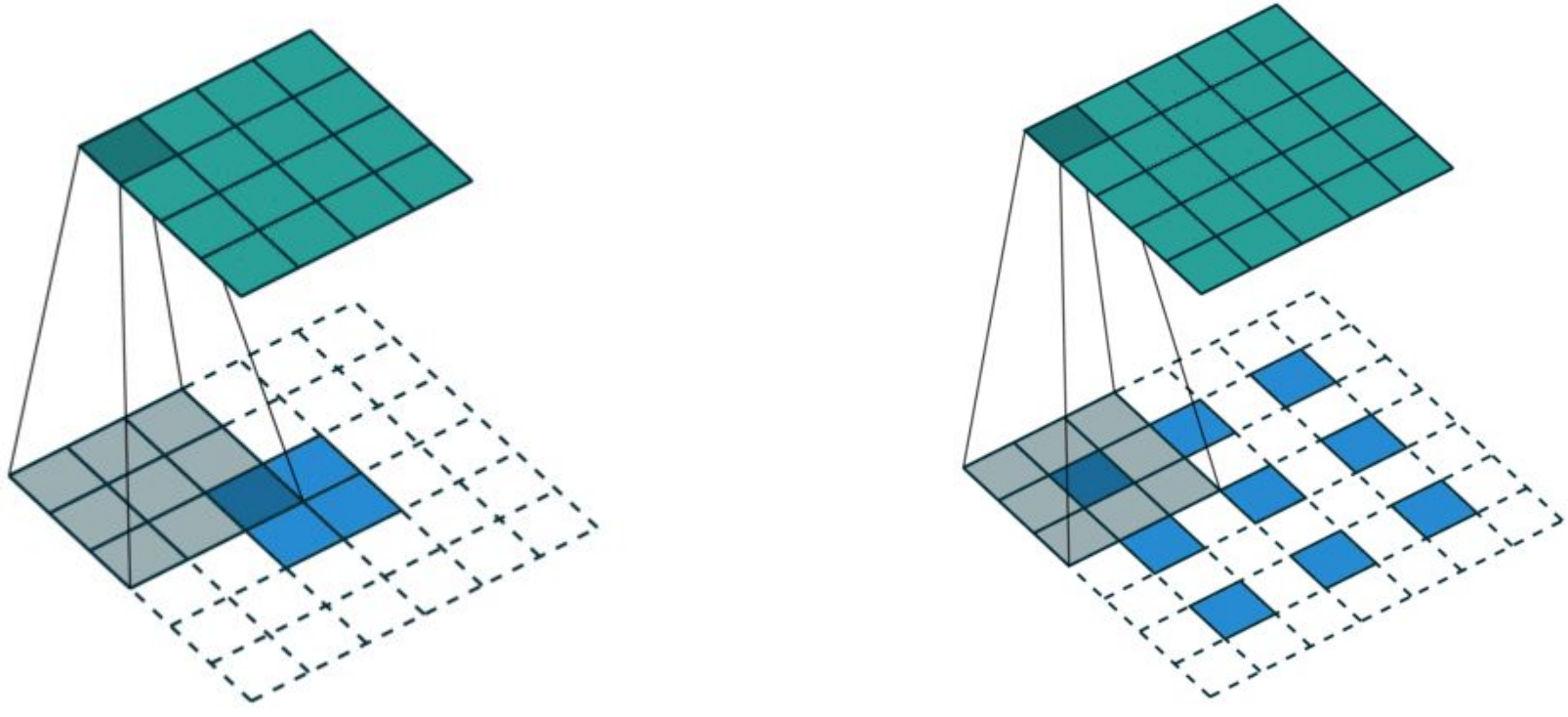


Figure 2. Generator model synthesizes fake MNIST images from noise. Upsampling is used instead of fractionally-strided transposed convolution.

Source: 1. *https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394*
2. https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

GAN: Upsampling (transposed convolutions)



Source: 1. *https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394*
2. https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

GAN: Discriminator

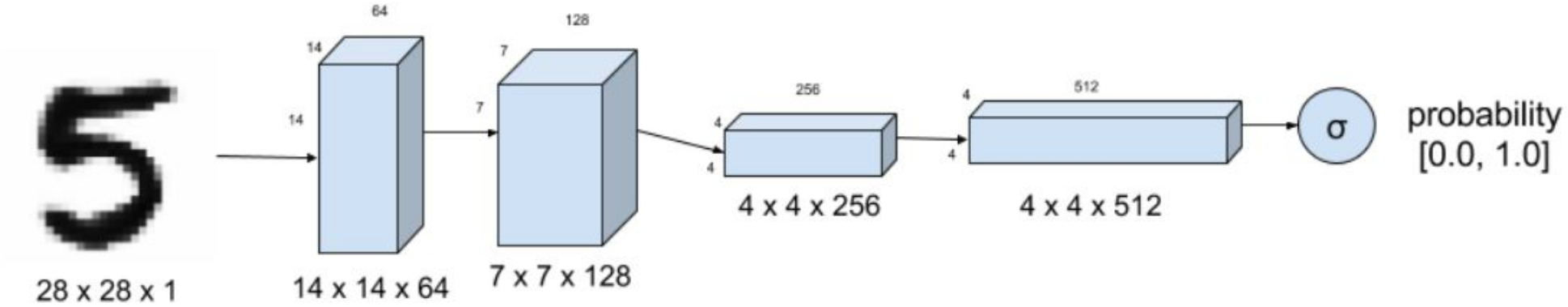


Figure 1. Discriminator of DCGAN tells how real an input image of a digit is. MNIST Dataset is used as ground truth for real images. Strided convolution instead of max-pooling down samples the image.

https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

GAN Training

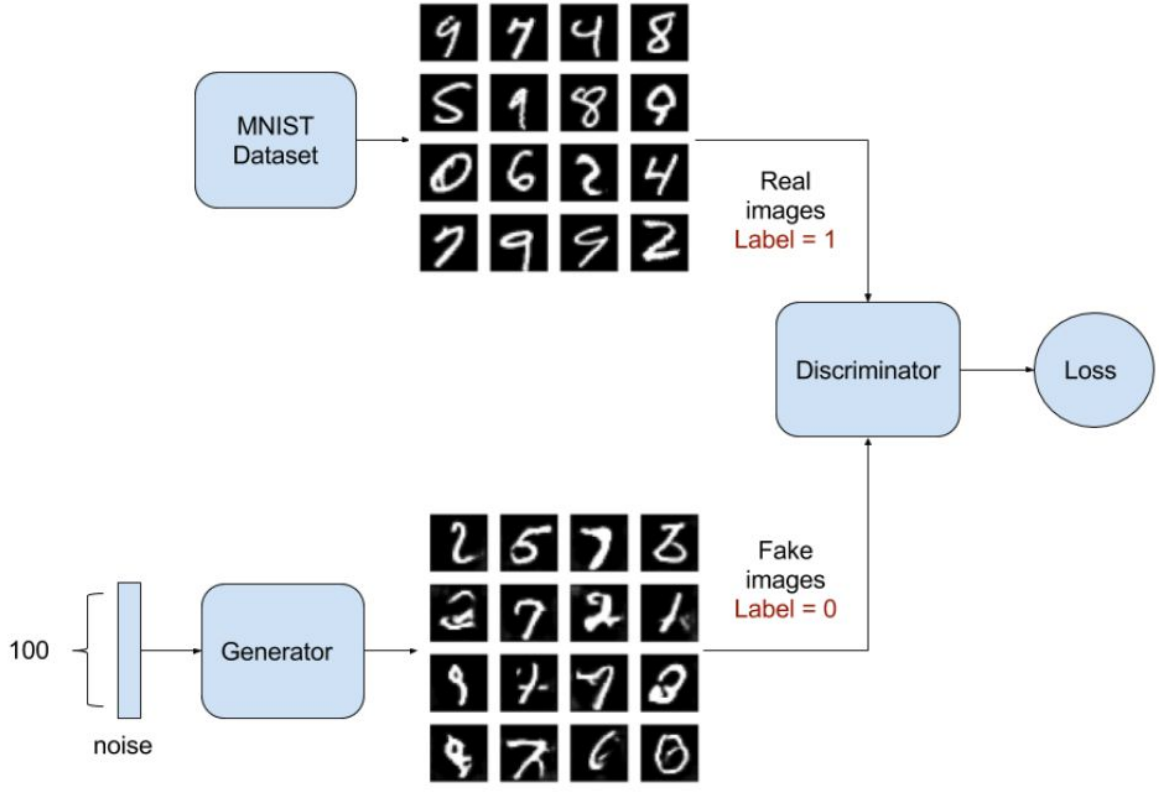


Figure 4. Discriminator model is trained to distinguish real from fake handwritten images.

Source: *https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0*

GAN Training Pseudo code

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

Source: *Diego Gomez Mosquera*

The Adversarial Model

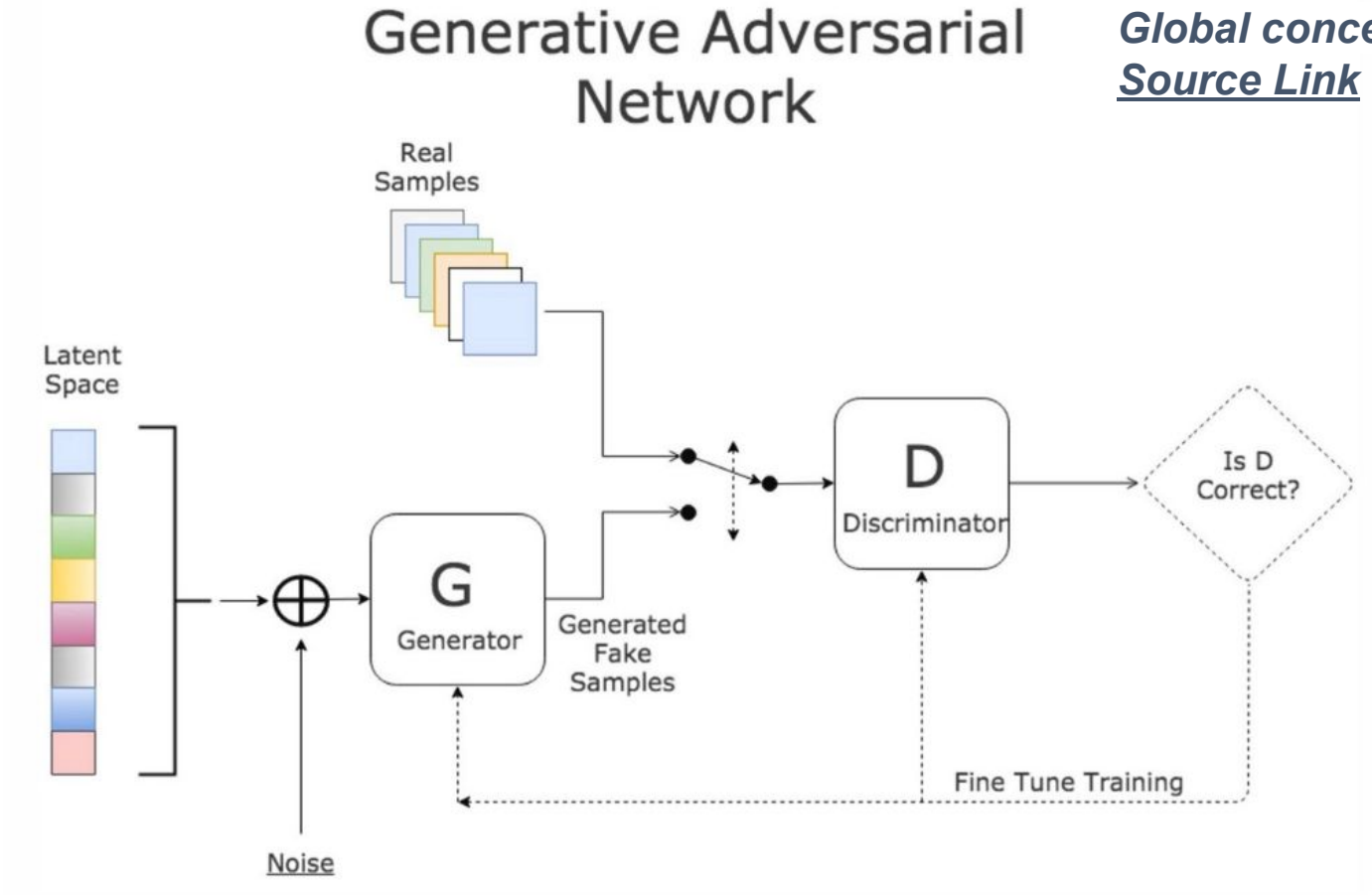
Figure 3. The Adversarial model is simply generator with its output connected to the input of the discriminator. Also shown is the training process wherein the Generator labels its fake image output with 1.0 trying to fool the Discriminator.

GAN: The Magic

Train the generator through the adversarial model:

The Generator is trained indirectly through the adversarial model. Pass noise to the adversarial model and mislabel everything as if they were images taken from the database (even though they are generated). The discriminator, which is trained beforehand on the real images, will fail to label the synthetic images as real, and there will be a high error overall. Then we use backpropogation and the parameter of the discriminator are frozen so only parameters in the generator will be affected. So minimising the error function of the adversarial model really means make the generated images as similar as possible to something, the discriminator will recognise as real.

GAN Architecture



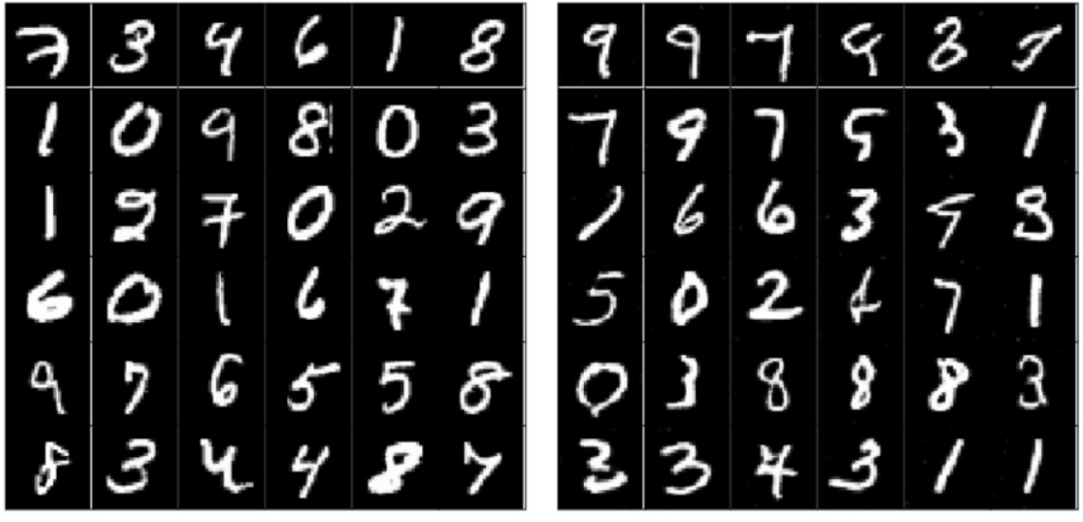
Global concept of a GAN.
Source Link

Source: *https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f*

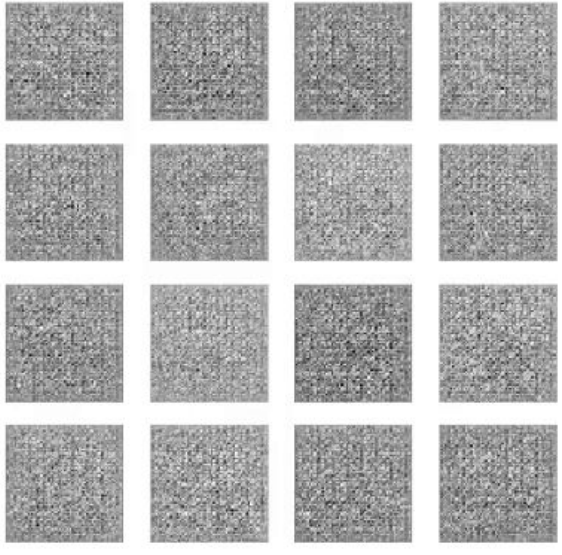
Result interpretation

*At the **end of training** you want the **loss of the adversarial model to be small, and the error of the discriminator to be high** -- it is no longer able to tell the difference.*

GAN: Result Examples

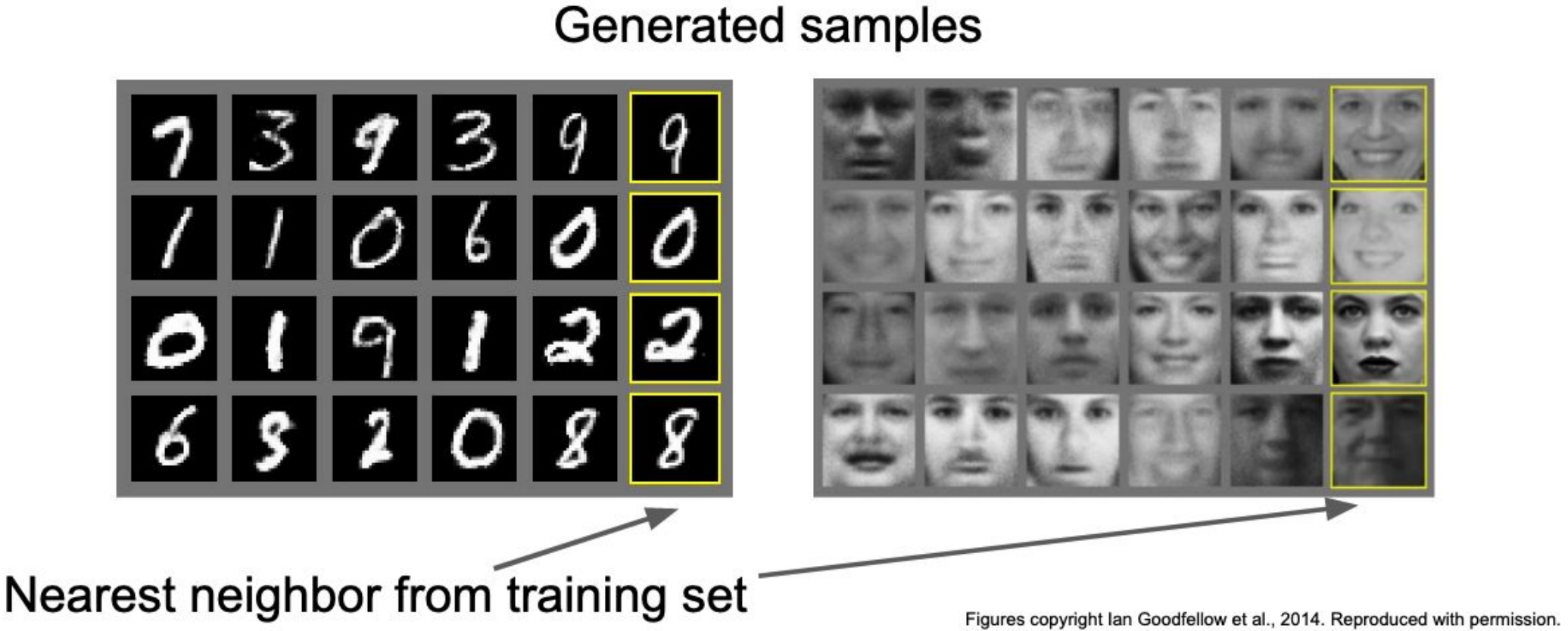


Comparing real (left) and generated (right) MNIST sample images. Because MNIST images have a simpler data structure, the model was able to produce more realistic samples when compared to the SVHNs.



Output of a GAN through time, learning to Create Hand-written digits. We'll code this example!
https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f

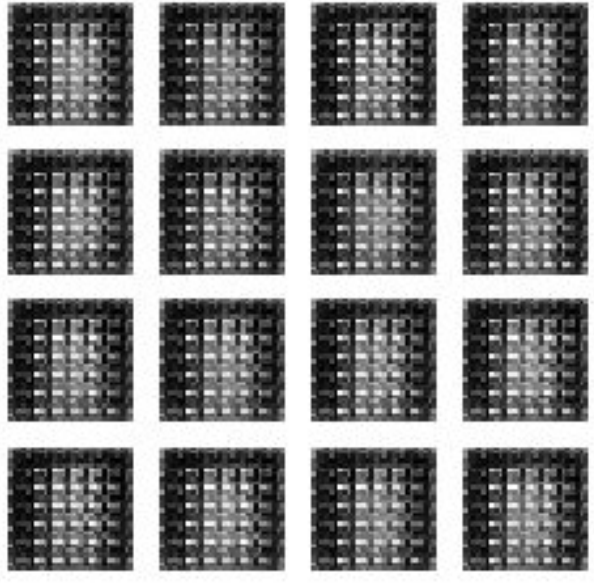
GAN: Result Examples



GAN notebook

Example of using DCGANS with Tensorflow 2 to generate MNIST data.

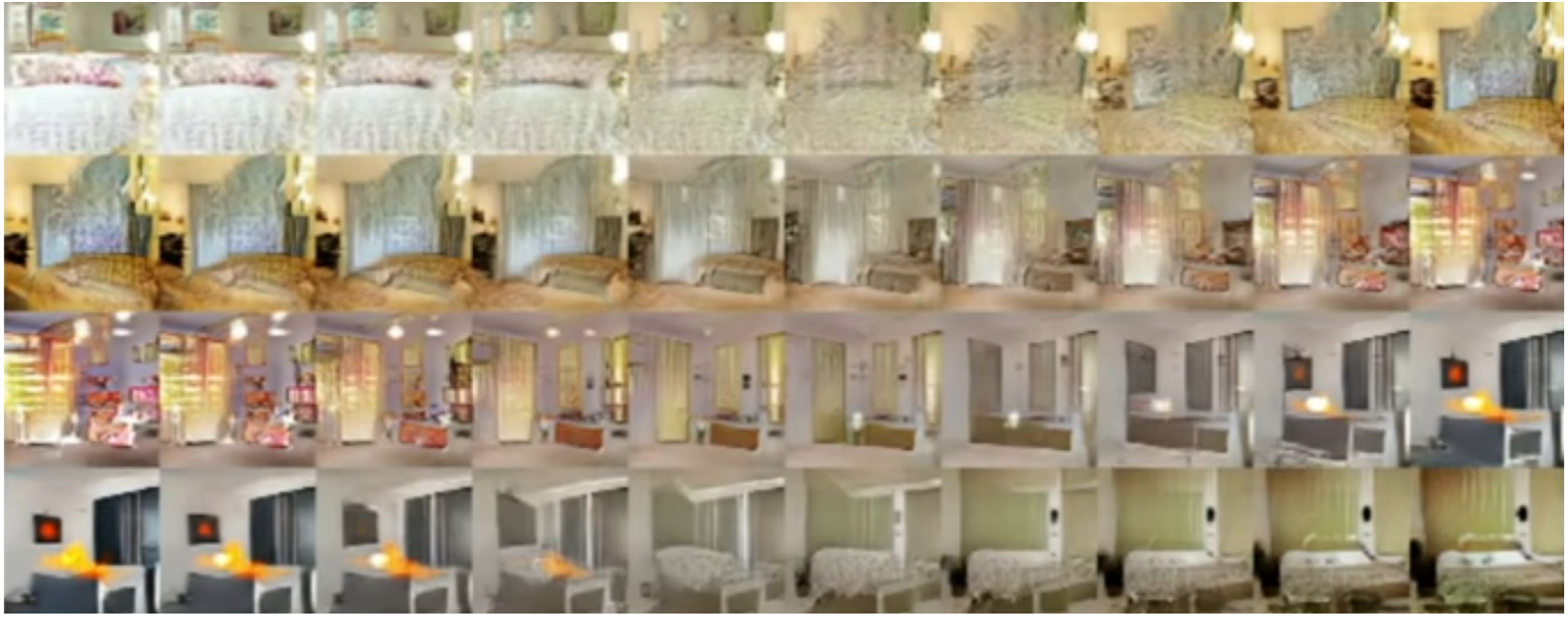
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/r2/tutorials/generative/dcgan.ipynb



GAN Applications

Tuning the latent trait vector z

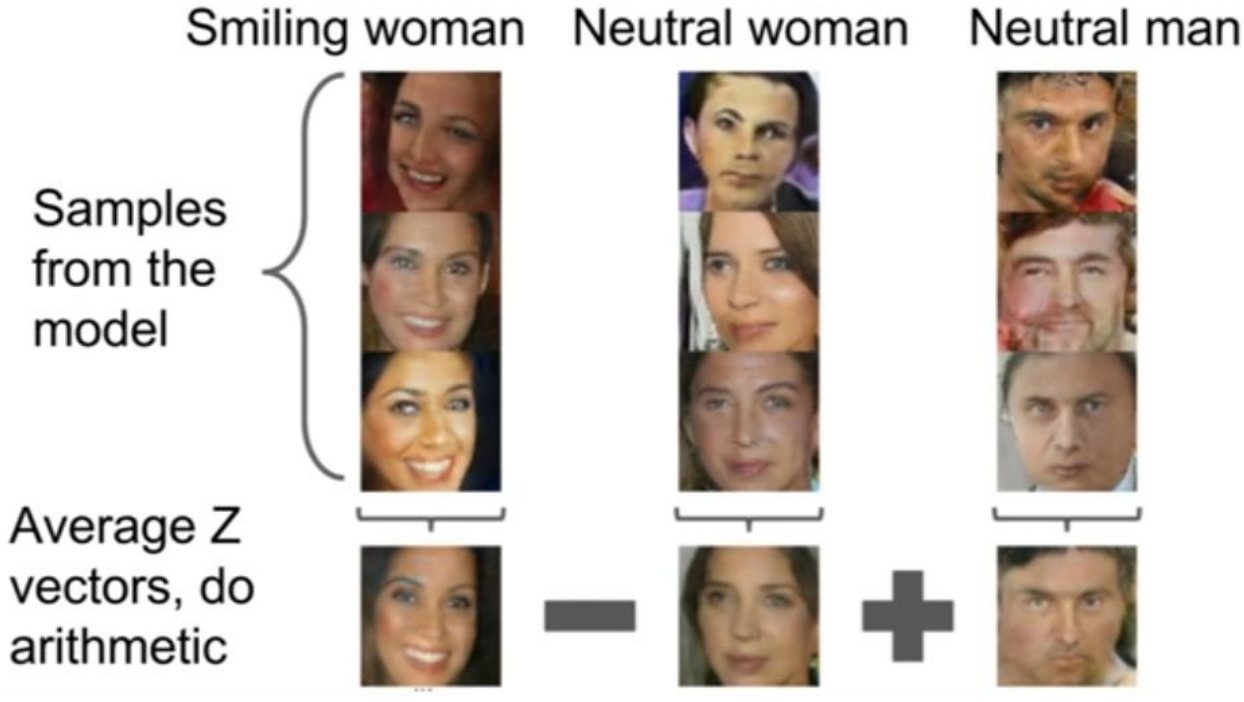
Interpolating between random point (vectors) in the latent space, shows a smooth transition between states.



Radford et a IXLR l 2016

Interpret GAN generation dimensions

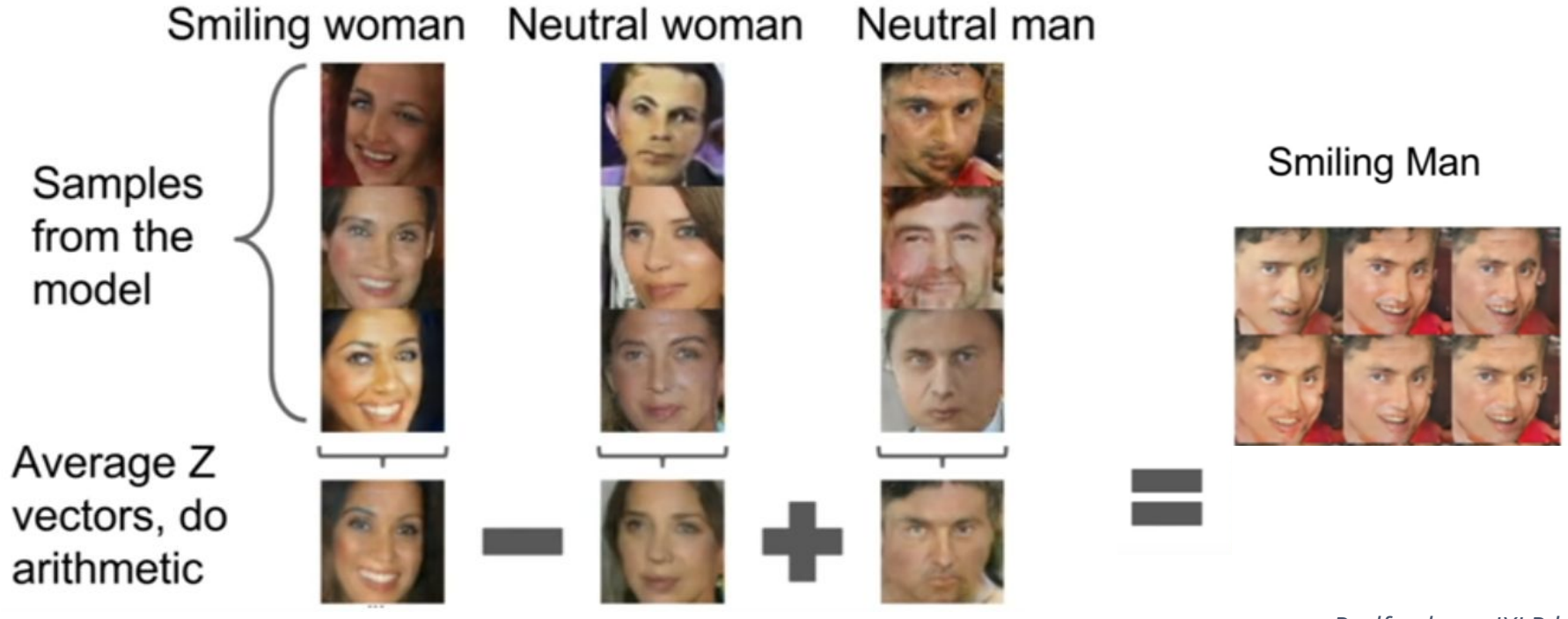
Finding the latent space dimensions that correspond to certain features in the generated output images. We can try to do vector mathematics of different samples



Radford et a IXLR l 2016

Interpret GAN generation dimensions

Finding the latent space dimensions that correspond to certain features in the generated output images. We can try to do vector mathematics of different samples



Radford et a IXLR l 2016

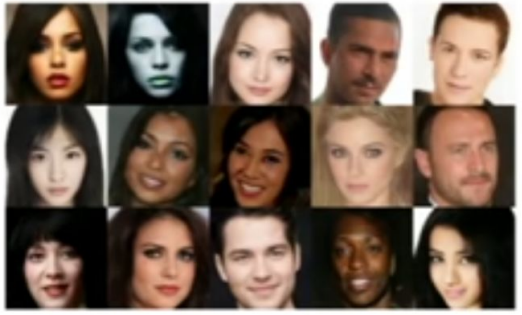
Interpret GAN generation dimensions

Radford et a IXLR l 2016

GAN Examples

Better training and generation

LSGAN. Mao et al. 2017.



BEGAN. Bertholet et al. 2017.

Source->Target domain transfer

CycleGAN. Zhu et al. 2017.

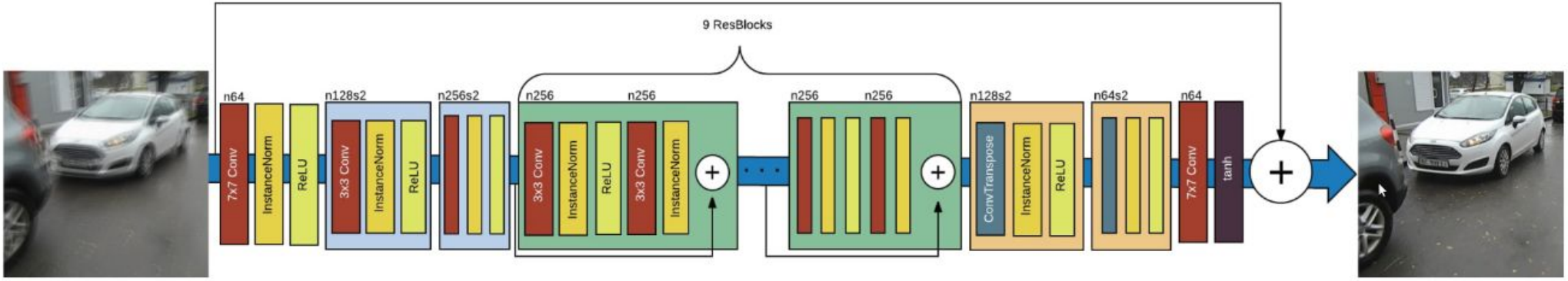
Text -> Image Synthesis

Akata et al. 2017.

Many GAN applications

Pix2pix.

GAN Examples: Deblur / Depixel Images



The Architecture of the DeblurGAN generator network—Source

Source: https://blog.sicara.com/keras-generative-adversarial-networks-image-deblurring-45e3ab6977b5

Cycle GAN: Style Translation

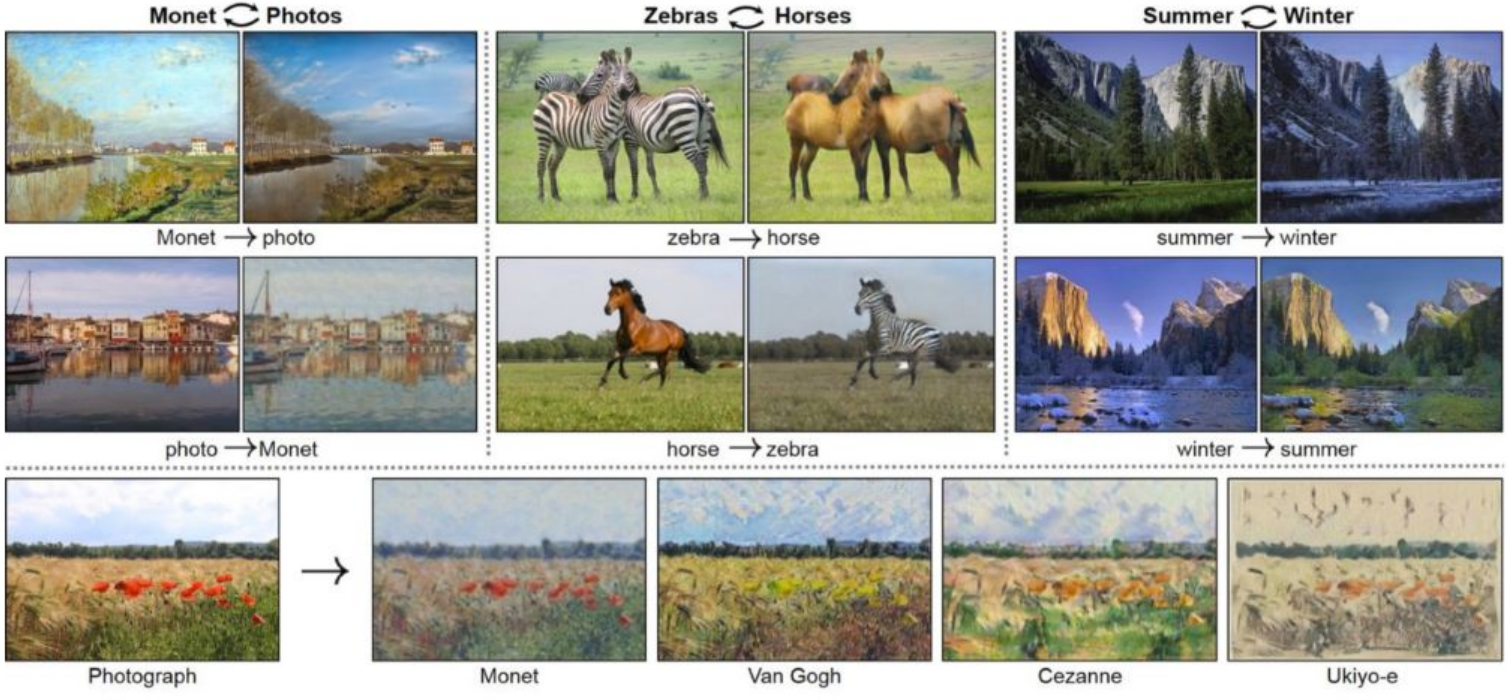


Image-to-Image Translation using GANs.

Source:
https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f

Art generation

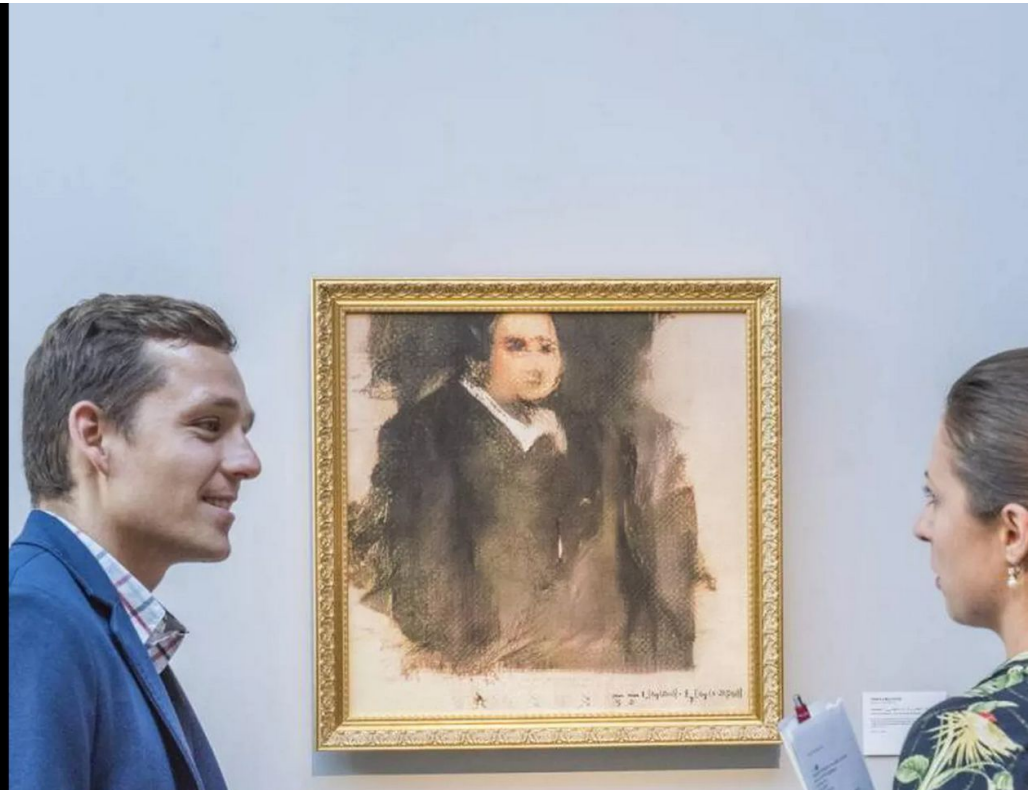
Christie's sold a portrait for $432,000 that had been generated by a GAN, based on open-source code written by Robbie Barrat of Stanford. (He didn't see any of the money, it went to the French company)

THE VERGE

REPORT

HOW THREE FRENCH STUDENTS USED BORROWED CODE TO PUT THE FIRST AI PORTRAIT IN CHRISTIE'S

By James Vincent | Oct 23, 2018, 9:34am EDT



(Recycle-)GAN: Fake video generation

GAN Examples:
Photo Generation

thispersondoesnotexist.com

thisxdoesnotexist.com

GAN: Application Examples

- AI Generates Fake Celebrity Faces (Paper)
- AI Learns Fashion Sense (Paper)
- Image to Image Translation using Cycle-Consistent Adversarial Neural Networks
- AI Creates Modern Art (Paper)
- This Deep Learning AI Generated Thousands of Creepy Cat Pictures
- MIT is using AI to create pure horror
- Amazon's new algorithm designs clothing by analyzing a bunch of pictures
- AI creates Photo-realistic Images (Paper)

End

End

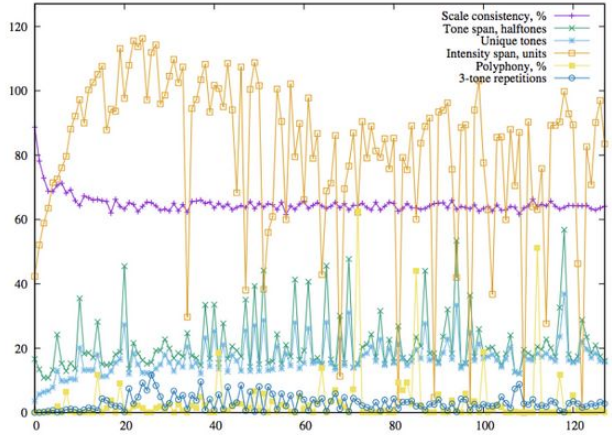
GAN Examples: Music Generation

C-RNN-GAN

C-RNN_GAN is a recurrent neural network with adversarial training. The adversaries are two different deep recurrent neural models, a generator (G) and a discriminator (D).

The generator is trained to generate data that is indistinguishable from real data, while the discriminator is trained to identify the generated data.

The training becomes a zero-sum game for which the Nash equilibrium is when the generator produces data that the discriminator cannot tell from real data.



(a) **C-RNN-GAN** using feature matching.

Source: https://blog.sicara.com/keras-generative-adversarial-networks-image-deblurring-45e3ab6977b5

Neural Style Transfer

In order to obtain a description of an image's content, the image is fed forward through the CNN and network activations sampled at a late convolution layer of the VGG-19 architecture. Let use denote C (p) {\displaystyle C(p)} C(p) to indicate the content of the input image p {\displaystyle p} p.

In order to obtain a description of an image's style, the image is fed forward through the CNN and network activations sampled at early to middle layers of the VGG-19 architecture. These activations are encoded into a Gramian matrix representation. Let us denote S (a) {\displaystyle S(a)} S(a) to describe the style of the example style image a {\displaystyle a} a.

Image x {\displaystyle x} x is initially approximated by adding a small amount of white noise to input image p {\displaystyle p} p. An iterative optimization then proceeds, gradually updating x {\displaystyle x} x to minimize the error (loss function):

L (x) = | C (x) − C (p) | + k | S (x) − S (a) | {\displaystyle {\mathcal {L(x)}}=|C(x)-C(p)|+k|S(x)-S(a)|} {\displaystyle {\mathcal {L(x)}}=|C(x)-C(p)|+k|S(x)-S(a)|},

GAN Basics: Generator Example Code

```
def generator(self):

model = Sequential()
        model.add(Dense(256, input_shape=(100,)))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(512))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(1024))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(self.WIDTH * self.HEIGHT *
self.CHANNELS, activation='tanh'))

model.add(Reshape((self.WIDTH, self.HEIGHT, self.CHANNELS)))

        return model
```

https://medium.com/@mattiaspinelli/simple-generative-adversarial-network-gans-with-keras-1fe578e44a87

GAN Basics: Discriminator Example Code

```
def discriminator(self):

    model = Sequential()
    model.add(Flatten(input_shape=self.SHAPE))
    model.add(Dense((self.WIDTH * self.HEIGHT *
self.CHANNELS), input_shape=self.SHAPE))

    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense((self.WIDTH * self.HEIGHT *
self.CHANNELS)/2))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Dense(1, activation='sigmoid'))
    model.summary()

    return model
```

https://medium.com/@mattiaspinelli/simple-generative-adversarial-network-gans-with-keras-1fe578e44a87

Create the instances

```
def __init__(self, width = 28, height= 28, channels = 1):

    self.WIDTH = width
    self.HEIGHT = height
    self.CHANNELS = channels

    self.SHAPE = (self.WIDTH, self.HEIGHT, self.CHANNELS)

    self.OPTIMIZER = Adam(lr=0.0002, decay=8e-9)

    self.noise_gen = np.random.normal(0,1,(100,))

    self.G = self.generator()
    self.G.compile(loss='binary_crossentropy', \
        optimizer=self.OPTIMIZER)

    self.D = self.discriminator()
    self.D.compile(loss='binary_crossentropy', \
        optimizer=self.OPTIMIZER, metrics=['accuracy']
```

https://medium.com/@mattiaspinelli/simple-generative-adversarial-network-gans-with-keras-1fe578e44a87

The Adversarial Model

Just a Generator followed by a discriminator.

IMPORTANT: We froze the Discriminator weights.

```
def stacked_G_D(self):
    self.D.trainable = False ## this freezes the weight, keep
                             ## reading to understand why this is
                             ## necessary

    model = Sequential()
    model.add(self.G)
    model.add(self.D)

    return model

### I need to compile the stacked model, so that a new instance of
### the discriminator is created with the frozen parameters.

self.stacked_G_D = self.stacked_G_D()
self.stacked_G_D.compile(loss='binary_crossentropy', \
 optimizer=self.OPTIMIZER)
```

Start Training (just batch training of discriminator)

```
def train(self, X_train, epochs=20000, batch = 32, save_interval =
200):

  for cnt in range(epochs):

    ## train discriminator

    random_index =  np.random.randint(0, len(X_train) - batch/2)

    legit_images = X_train[random_index : random_index +
batch/2].reshape(batch/2, self.WIDTH, self.HEIGHT, self.CHANNELS)

    gen_noise = np.random.normal(0, 1, (batch/2,100))
    syntetic_images = self.G.predict(gen_noise)

    x_combined_batch = np.concatenate((legit_images,
syntetic_images))

    y_combined_batch = np.concatenate((np.ones((batch/2, 1)),
np.zeros((batch/2, 1))))

    d_loss = self.D.train_on_batch(x_combined_batch,
y_combined_batch)
```

https://medium.com/@mattiaspinelli/simple-generative-adversarial-network-gans-with-keras-1fe578e44a87