UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

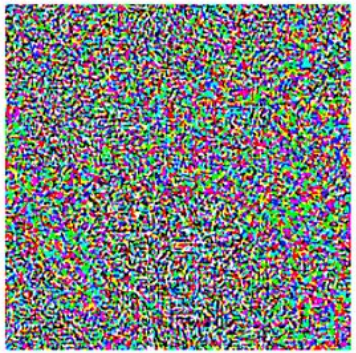
Deep Learning

Week 5 : Generative Adversarial Networks + Image Pyramids (Processing Images Part 1)

TRANSFORMATEC

Generative Adversarial Network ≠ Adversarial Images

$+ .007 \times$



$=$

“panda”

57.7% confidence

noise

“gibbon”

99.3% confidence

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie,[*] Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,[†] Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Intriguing properties of neural networks

Christian Szegedy Google Inc.
Wojciech Zaremba New York University
Ilya Sutskever Google Inc.
Joan Bruna New York University
Dumitru Erhan Google Inc.
Ian Goodfellow University of Montreal
Rob Fergus New York University, Facebook Inc.

Abstract

Deep neural networks are highly expressive models that have recently achieved state of the art performance on speech and visual recognition tasks. While their expressiveness is the reason they succeed, it also causes them to learn uninterpretable solutions that could have counter-intuitive properties. In this paper we report two such properties.

First, we find that there is no distinction between individual high level units and random linear combinations of high level units, according to various methods of unit analysis. It suggests that it is the space, rather than the individual units, that contains of the semantic information in the high layers of neural networks.

Second, we find that deep neural networks learn input-output mappings that are fairly discontinuous to a significant extend. Specifically, we find that we can cause the network to misclassify an image by applying a certain imperceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the

Under review as a conference paper at ICLR 2015

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

ABSTRACT

Several machine learning models, including neural networks, consistently misclassify *adversarial examples*—inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. Early attempts at explaining this phenomenon focused on nonlinearity and overfitting. We argue instead that the primary cause of neural networks' vulnerability to adversarial perturbation is their linear nature. This explanation is supported by new quantitative results while giving the first explanation of the most intriguing fact about them: their generalization across architectures and training sets. Moreover, this view yields a simple and fast method of generating adversarial examples. Using this approach to provide examples for adversarial training, we reduce the test set error of a maxout network on the MNIST dataset.

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie,* Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,† Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Intriguing properties of neural networks

Christian Szegedy Google Inc.
Wojciech Zaremba New York University
Ilya Sutskever Google Inc.
Joan Bruna New York University
Dumitru Erhan Google Inc.
Ian Goodfellow University of Montreal
Rob Fergus New York University, Facebook Inc.

Abstract

Deep neural networks are highly expressive models that have recently achieved state of the art performance on speech and visual recognition tasks. While their expressiveness is the reason they succeed, it also causes them to learn uninterpretable solutions that could have counter-intuitive properties. In this paper we report two such properties.

First, we find that there is no distinction between individual high level units and random linear combinations of high level units, according to various methods of unit analysis. It suggests that it is the space, rather than the individual units, that contains of the semantic information in the high layers of neural networks.

Second, we find that deep neural networks learn input-output mappings that are fairly discontinuous to a significant extend. Specifically, we find that we can cause the network to misclassify an image by applying a certain imperceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the

Under review as a conference paper at ICLR 2015

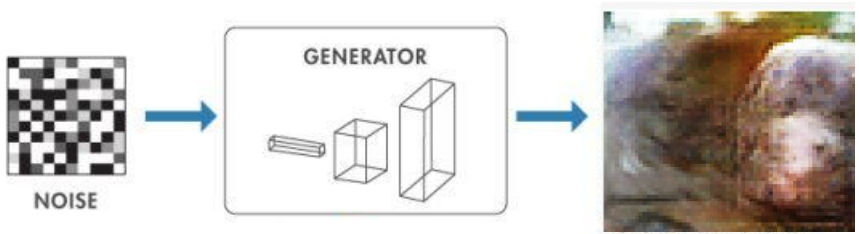
EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

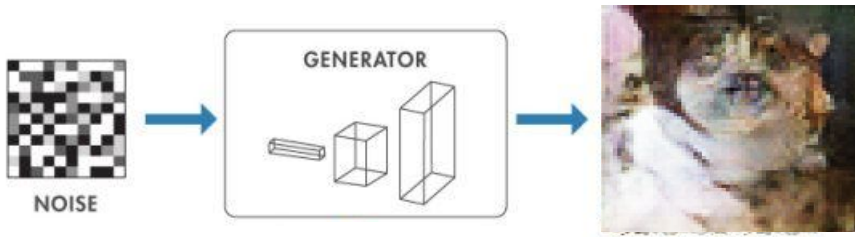
ABSTRACT

Several machine learning models, including neural networks, consistently misclassify *adversarial examples*—inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. Early attempts at explaining this phenomenon focused on nonlinearity and overfitting. We argue instead that the primary cause of neural networks' vulnerability to adversarial perturbation is their linear nature. This explanation is supported by new quantitative results while giving the first explanation of the most intriguing fact about them: their generalization across architectures and training sets. Moreover, this view yields a simple and fast method of generating adversarial examples. Using this approach to provide examples for adversarial training, we reduce the test set error of a maxout network on the MNIST dataset.

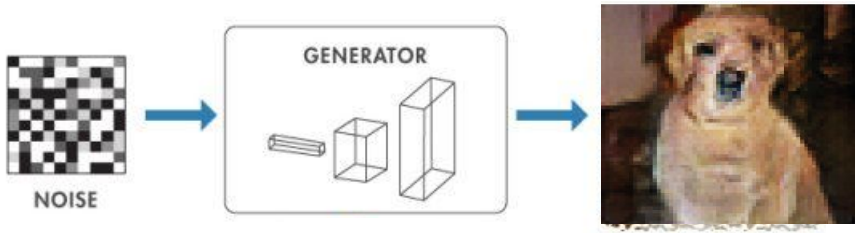
What is the Motivation of GANs?



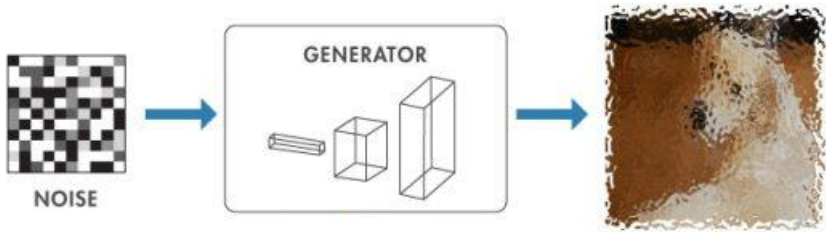
What is the Motivation of GANs?



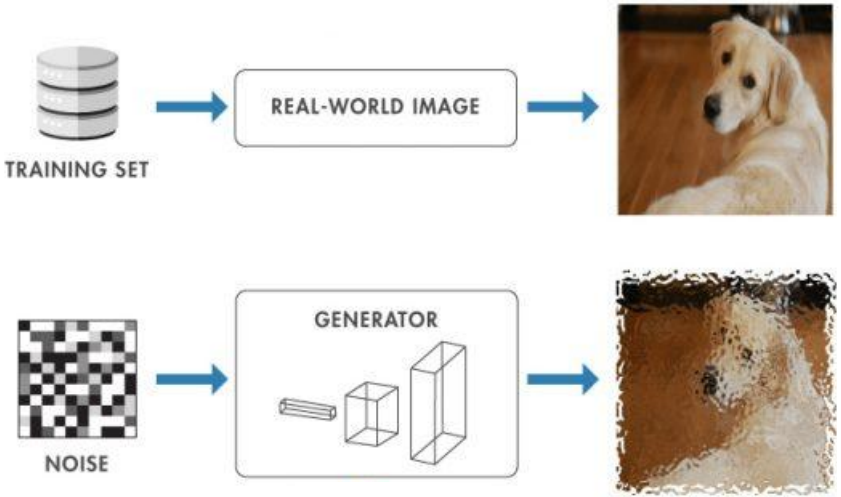
What is the Motivation of GANs?



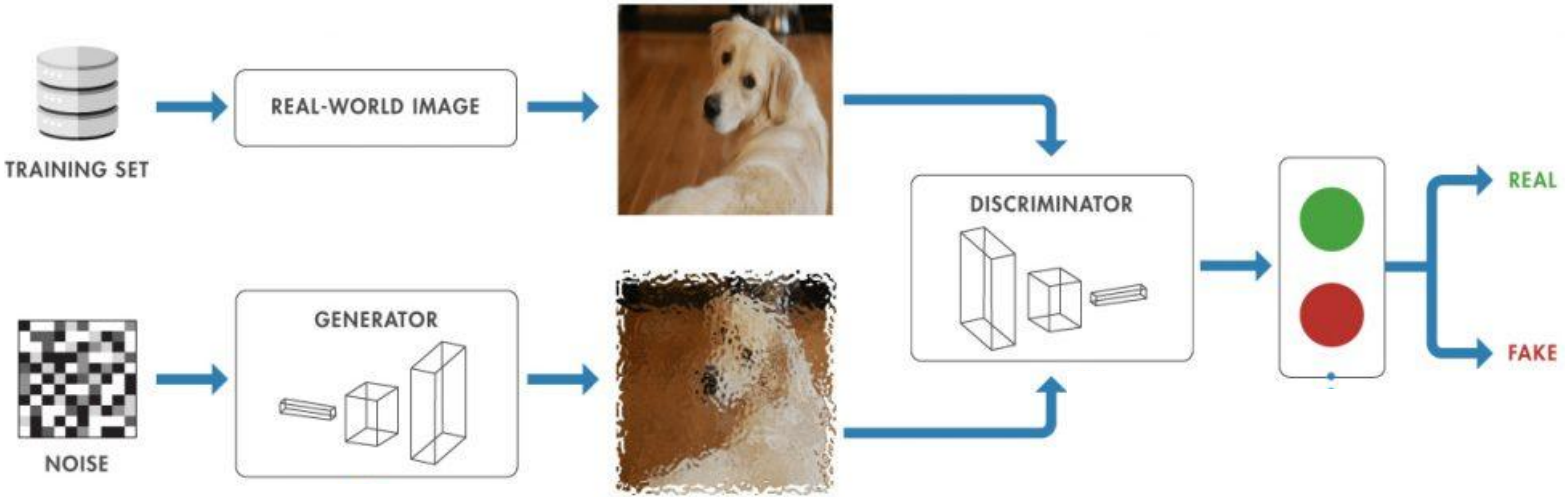
What is the Motivation of GANs?



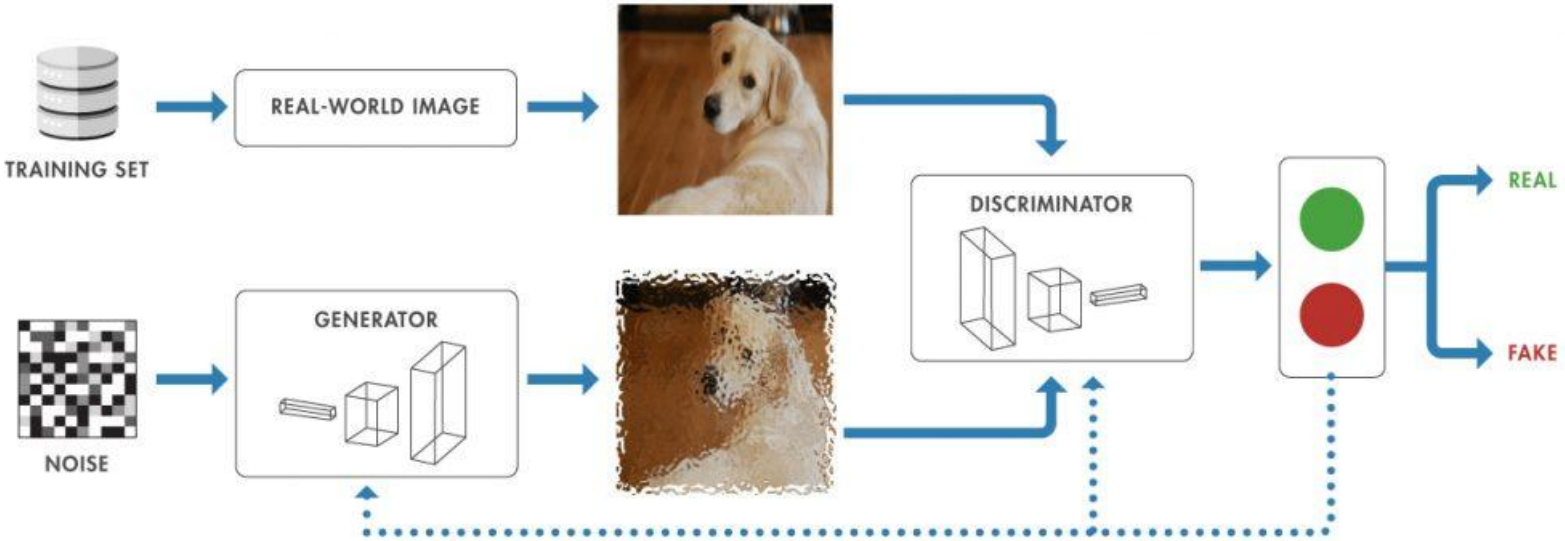
What is the Motivation of GANs?



What is the Motivation of GANs?



What is the Motivation of GANs?



Let us take a step back to analyze (a small sub-set) of different Deep Learning Models:

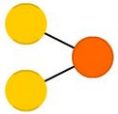
A mostly complete chart of

Neural Networks

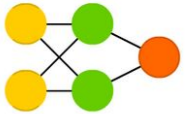
 asimovinstitute.org

- Input Cell
- Backfed Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Gated Memory Cell
- Kernel
- Convolution or Pool

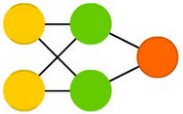
Perceptron (P)



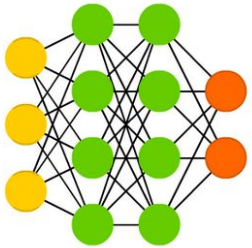
Feed Forward (FF)



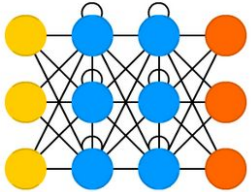
Radial Basis Network (RBF)



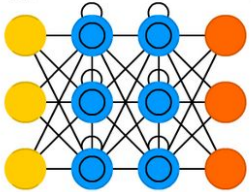
Deep Feed Forward (DFF)



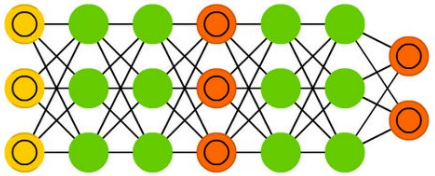
Recurrent Neural Network (RNN)



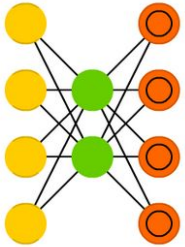
Long / Short Term Memory (LSTM



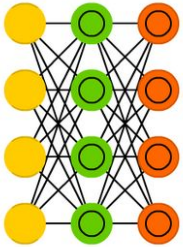
Generative Adversarial Network (GAN)



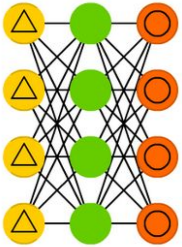
Auto Encoder (AE)



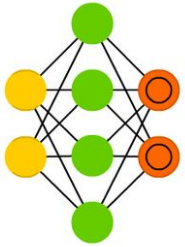
Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



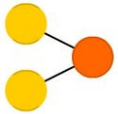
A mostly complete chart of

Neural Networks

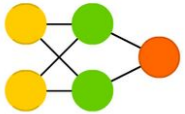
 asimovinstitute.org

- Input Cell
- Backfed Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Gated Memory Cell
- Kernel
- Convolution or Pool

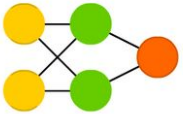
Perceptron (P)



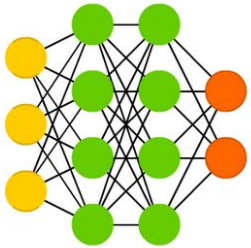
Feed Forward (FF)



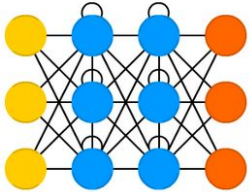
Radial Basis Network (RBF)



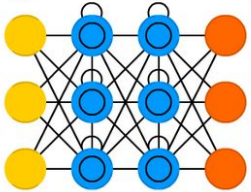
Deep Feed Forward (DFF)



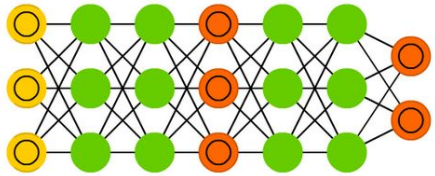
Recurrent Neural Network (RNN)



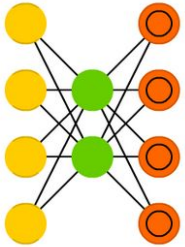
Long / Short Term Memory (LSTM



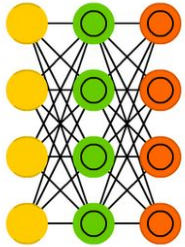
Generative Adversarial Network (GAN)



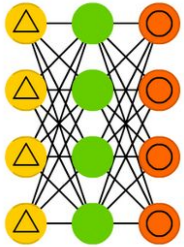
Auto Encoder (AE)



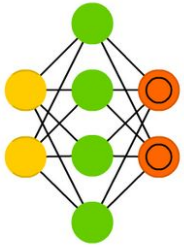
Variational AE (VAE)



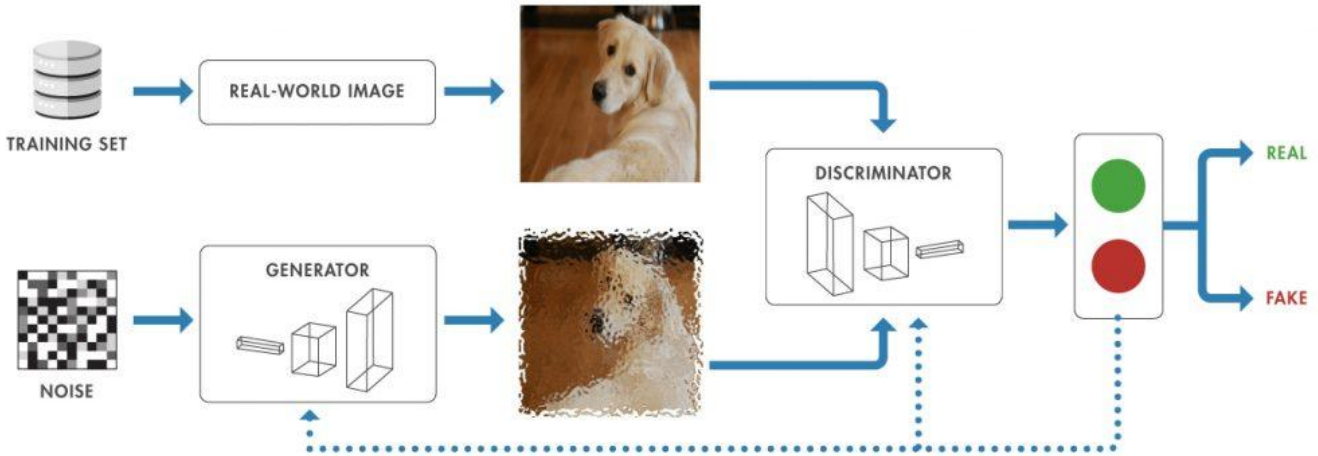
Denoising AE (DAE)



Sparse AE (SAE)

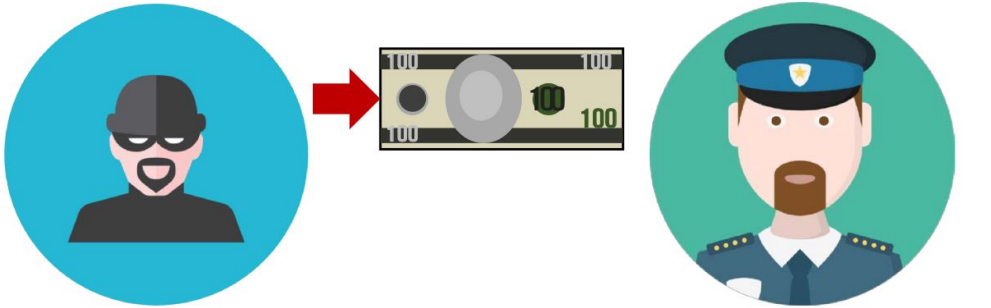


Generative Adversarial Networks



Model architecture

Generative Adversarial Network



Generator

Tries to generates

more real-like fake bills

Discriminator

Tries to catch fake bills

Penalty if failure

Source: Bumsoo Kim (Korea University)

Model architecture

Generative Adversarial Network

Generator

Tries to generates more real-like fake bills

Discriminator

Tries to catch fake bills

Penalty if failure

Source: Bumsoo Kim (Korea University)

Model architecture

Generative Adversarial Network

Counterfeiter

Generator

Tries to generates

more real-like fake bills

Police

Discriminator

Tries to catch fake bills

Penalty if failure

Source: Bumsoo Kim (Korea University)

Model architecture

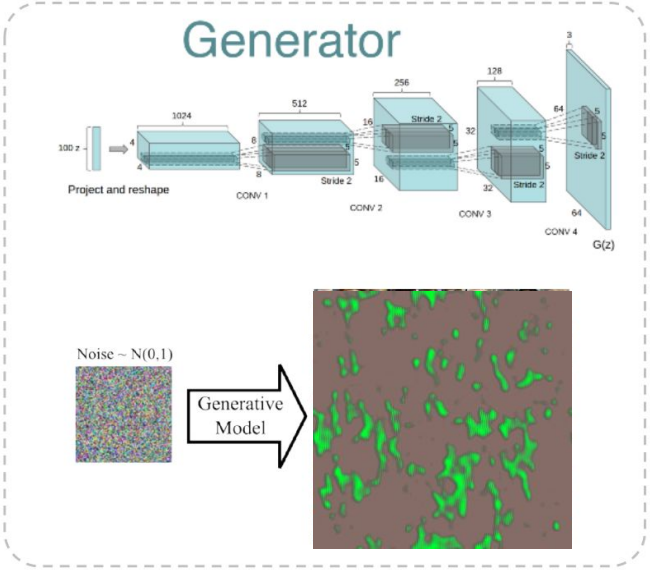
Generative Adversarial Network

Counterfeiter

Generator

Tries to generates

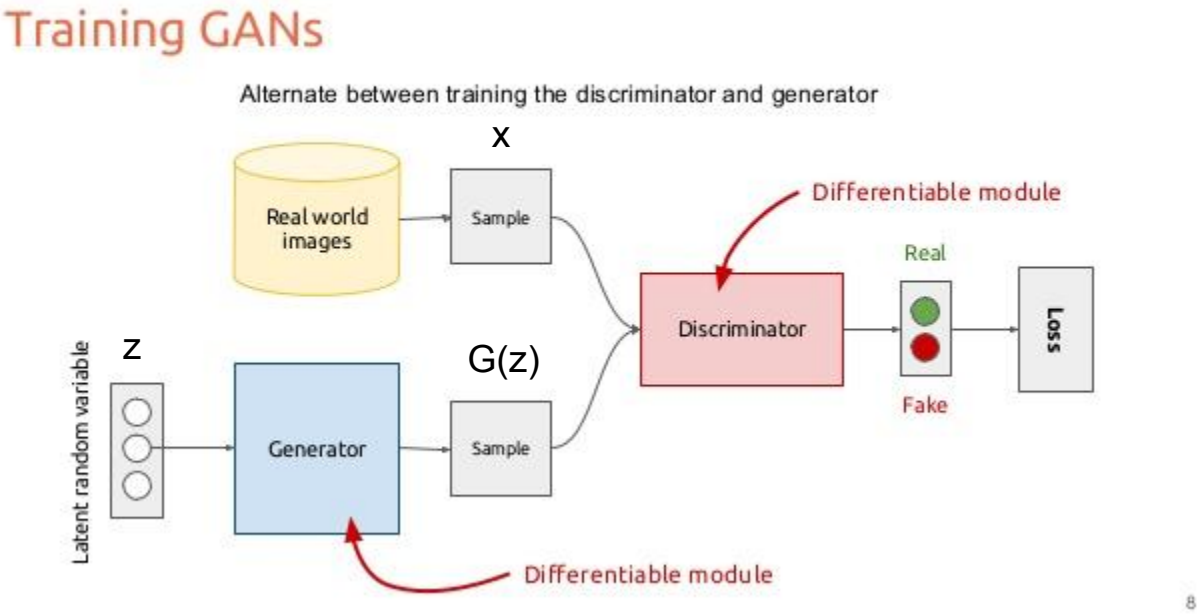
more real-like fake bills



Discriminator:
Real or Fake?

Source: Bumsoo Kim (Korea University)

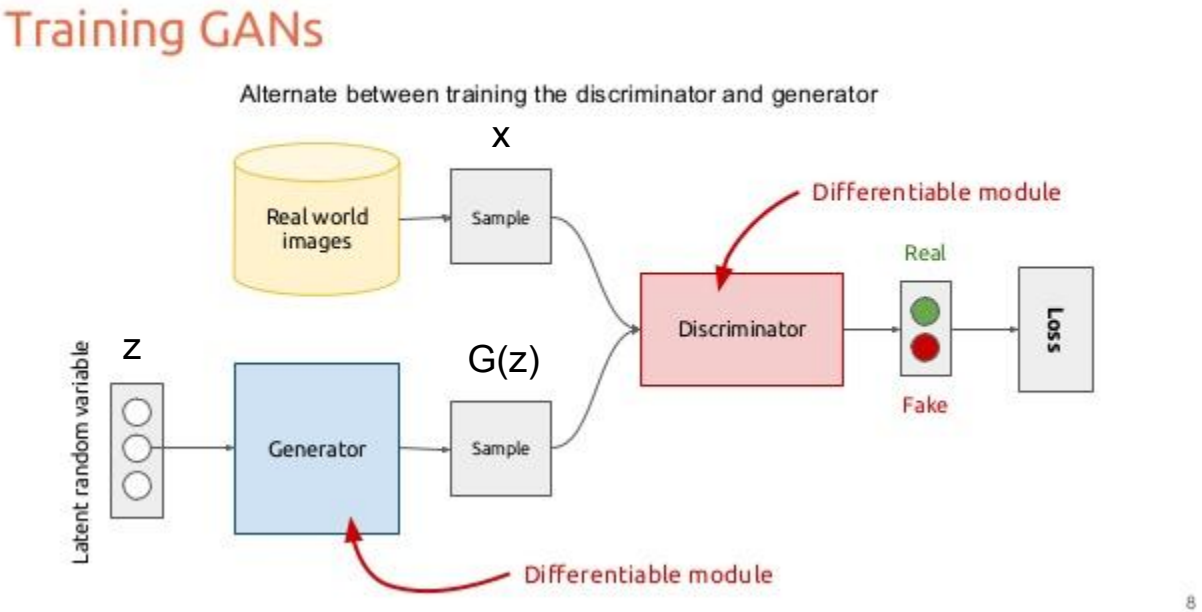
Image Source: Kevin McGuiness (Dublin City University)



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

Goodfellow et al. 2014

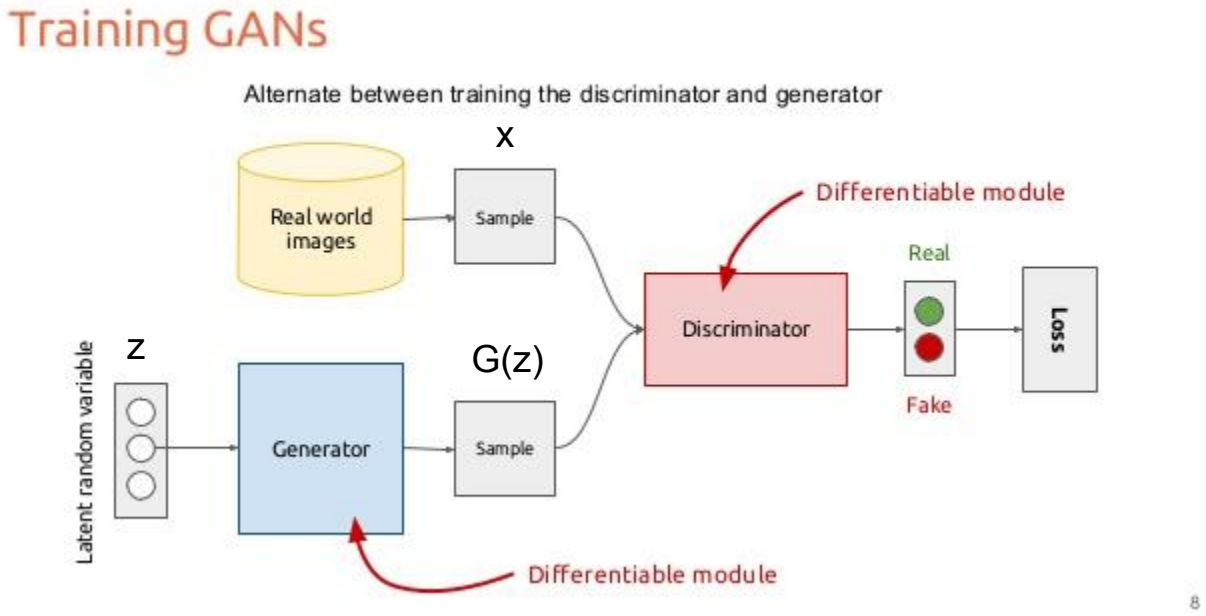
Image Source: Kevin McGuiness (Dublin City University)



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

Goodfellow et al. 2014

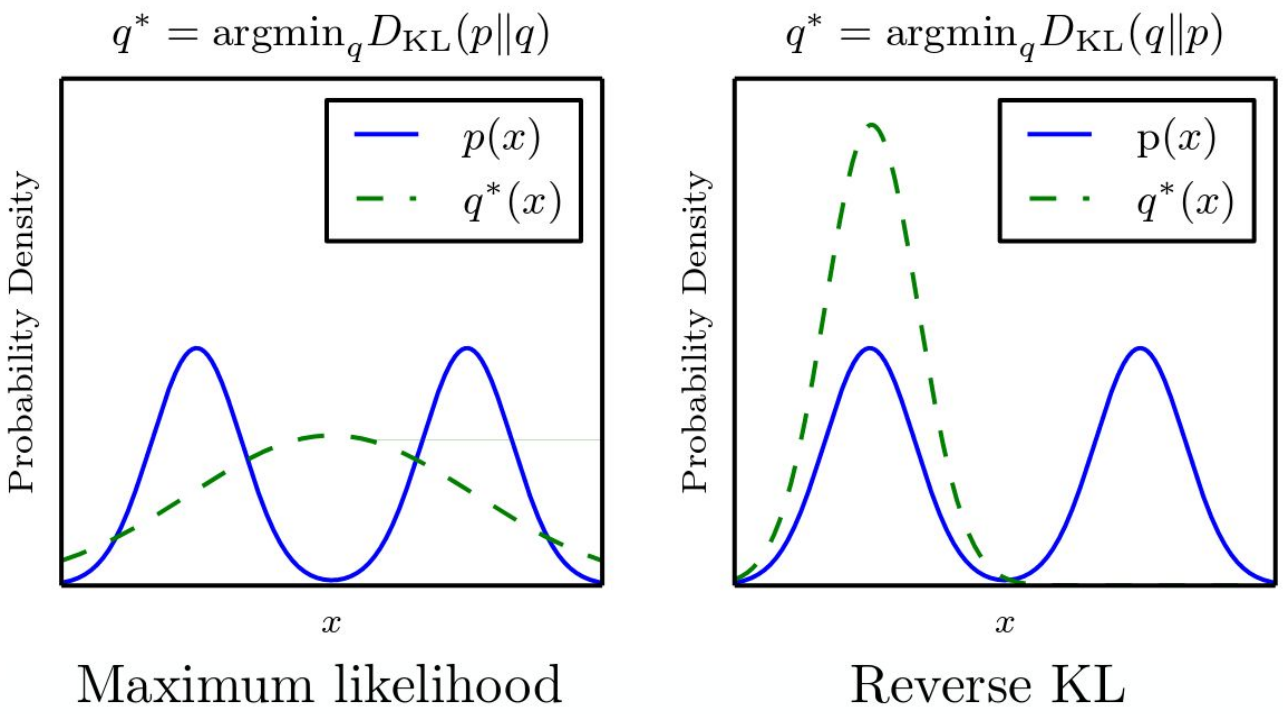
Image Source: Kevin McGuiness (Dublin City University)



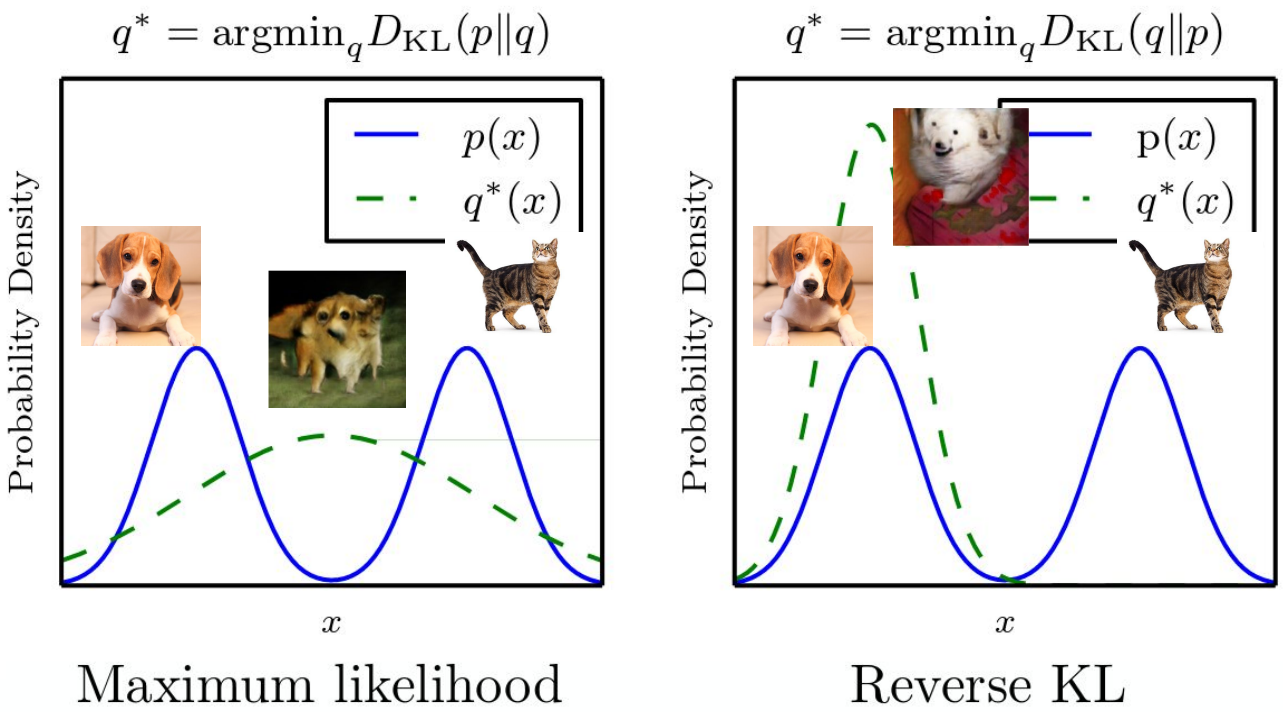
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Goodfellow et al. 2014

Consider *p(x)* being the real data distribution,
and *q*(x)* being the generated data distribution

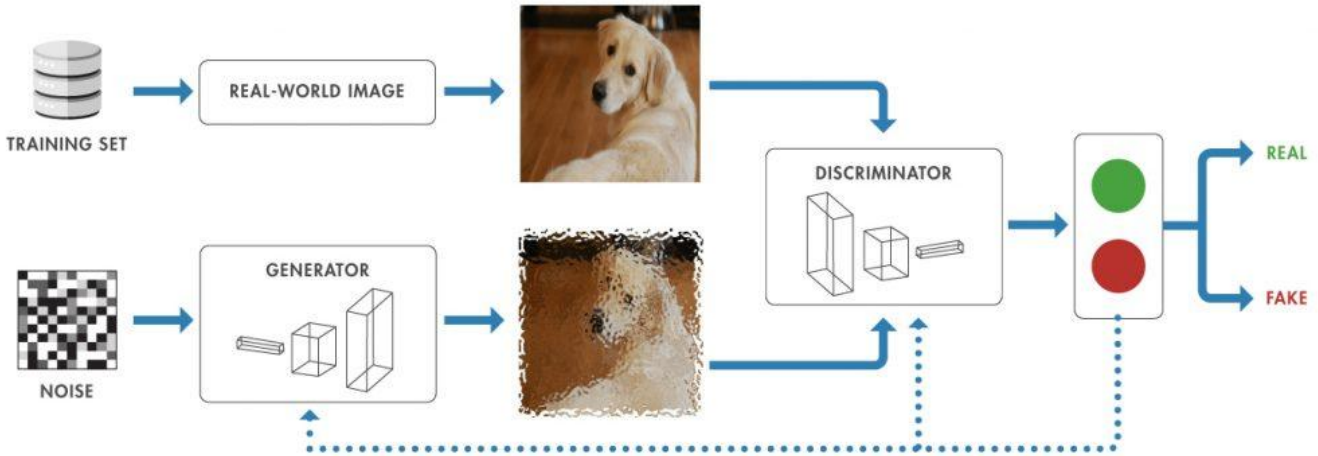


Consider *p(x)* being the real data distribution,
and *q*(x)* being the generated data distribution



"Mode Dropping"

Generative Adversarial Networks



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

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

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

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

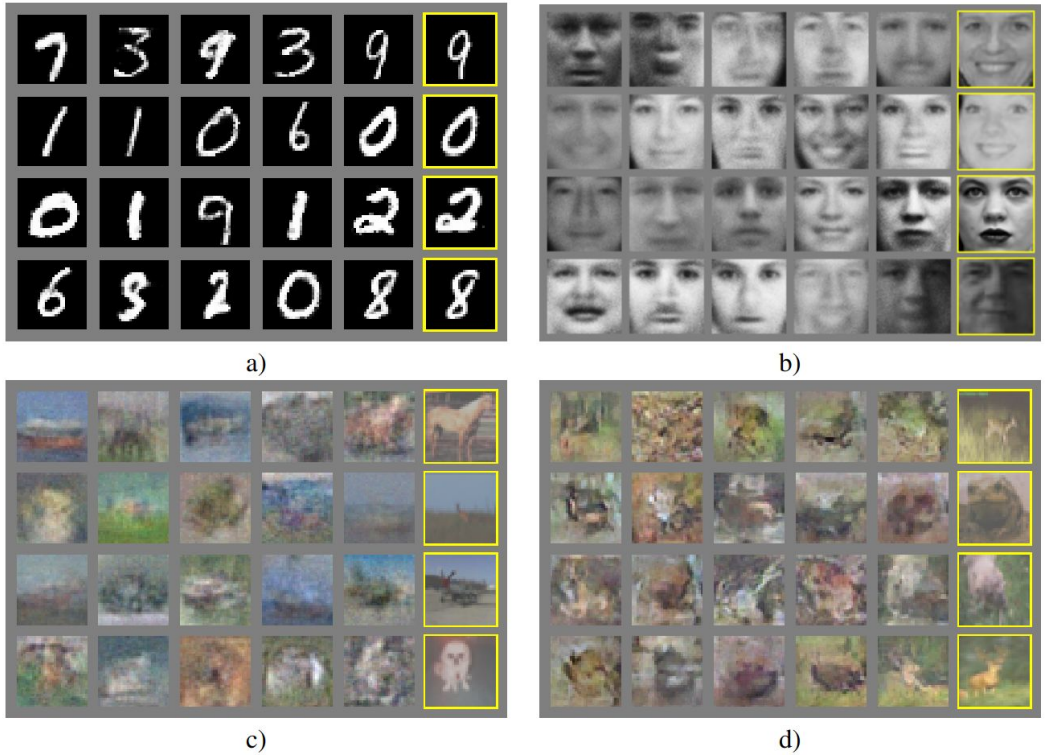


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

Style-GAN

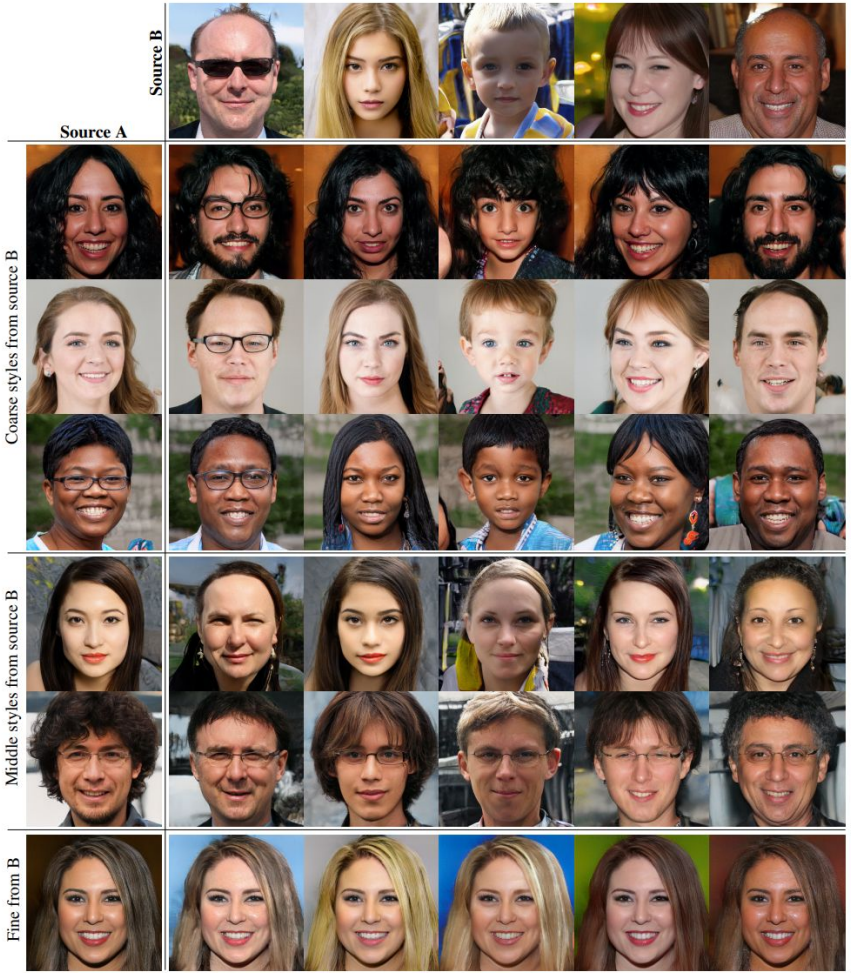
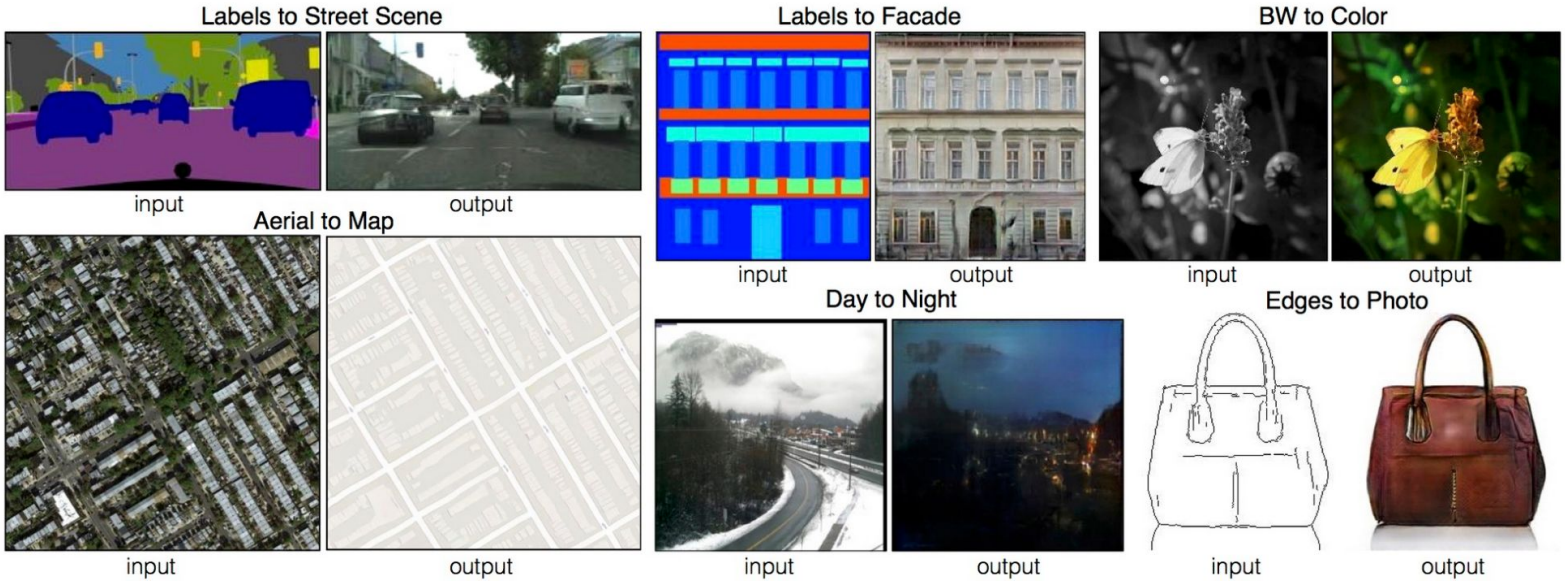

Source B
Source A
Coarse styles from source B
Middle styles from source B
Fine from B

Image-to-Image Translation with Conditional Adversarial Nets

Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros

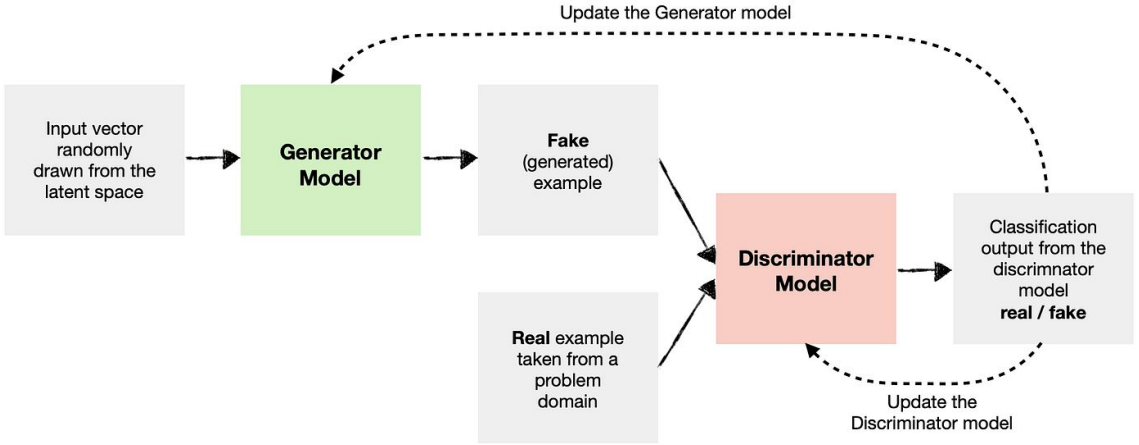
University of California, Berkeley
In CVPR 2017

[Paper] [GitHub]

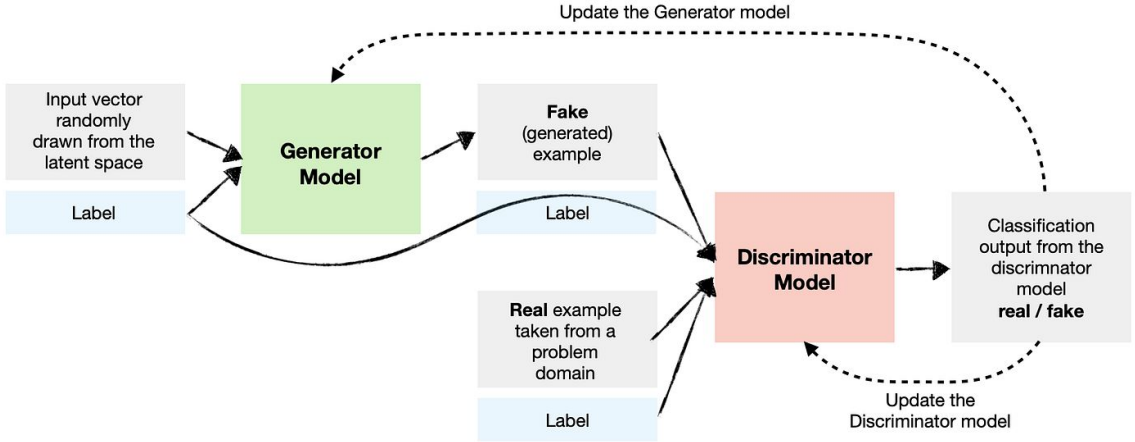


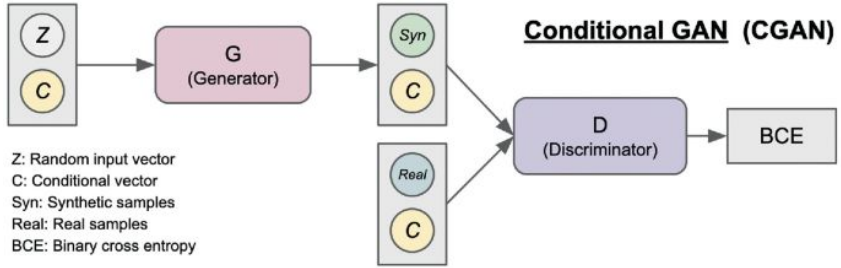
Example results on several image-to-image translation problems. In each case we use the same architecture and objective, simply training on different data.

Generative Adversarial Network (GAN)



conditional Generative Adversarial Network (GAN)





Note : C, the conditional vector does not need to be a label [!]

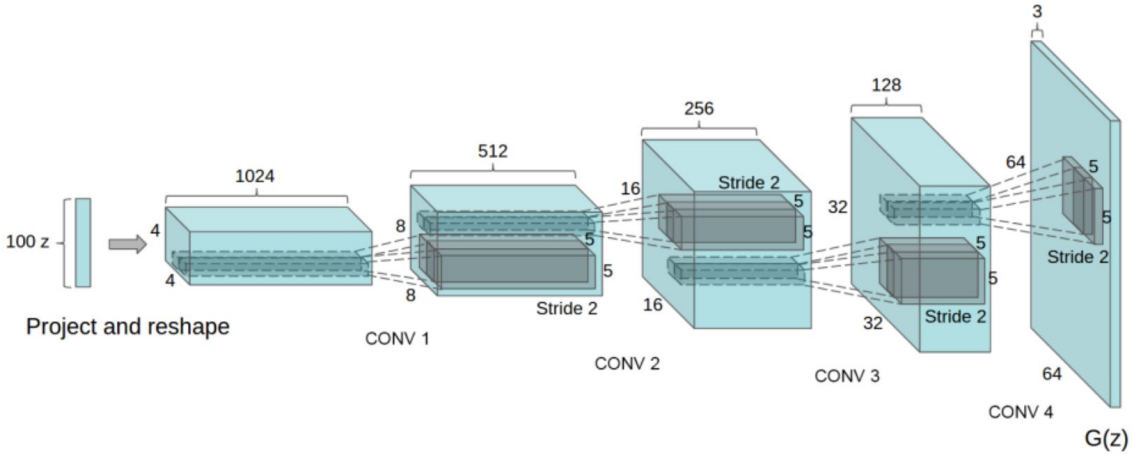
UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

[DCGAN]

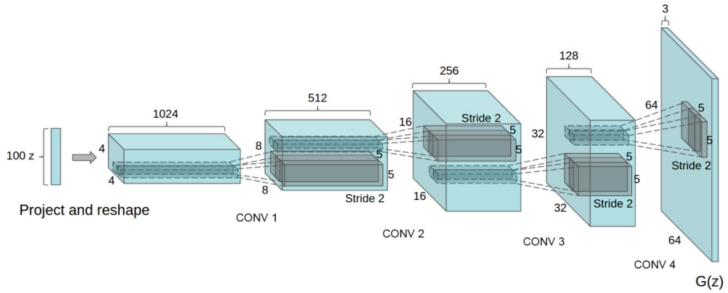
Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

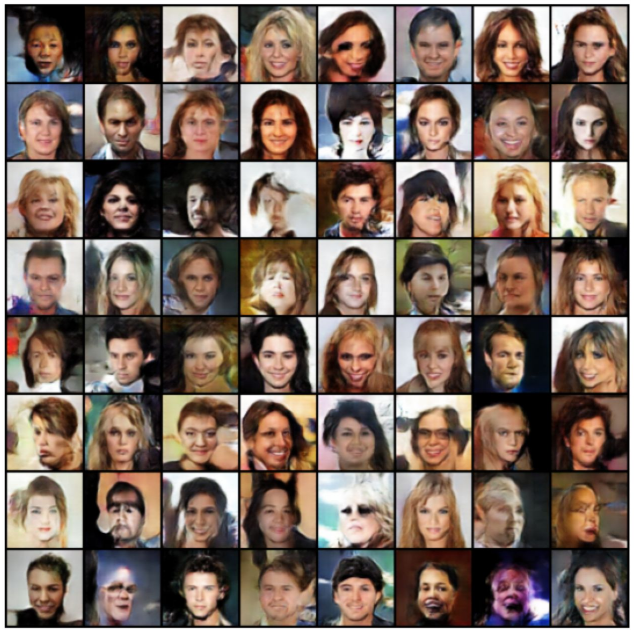
Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

Jan 2016



```python
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. ``(ngf*8) x 4 x 4``
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. ``(ngf*4) x 8 x 8``
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. ``(ngf*2) x 16 x 16``
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. ``(ngf) x 32 x 32``
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. ``(nc) x 64 x 64``
        )
```





Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

Emily Denton[*]
Dept. of Computer Science
Courant Institute
New York University

Soumith Chintala[*] **Arthur Szlam** **Rob Fergus**
Facebook AI Research
New York

Abstract

In this paper we introduce a generative parametric model capable of producing high quality samples of natural images. Our approach uses a cascade of convolutional networks within a Laplacian pyramid framework to generate images in a coarse-to-fine fashion. At each level of the pyramid, a separate generative convnet model is trained using the Generative Adversarial Nets (GAN) approach [10]. Samples drawn from our model are of significantly higher quality than alternate approaches. In a quantitative assessment by human evaluators, our CIFAR10 samples were mistaken for real images around 40% of the time, compared to 10% for samples drawn from a GAN baseline model. We also show samples from models trained on the higher resolution images of the LSUN scene dataset.

2015 [!]

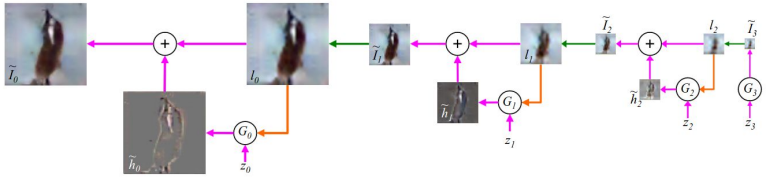


Figure 1: The sampling procedure for our LAPGAN model. We start with a noise sample z_3 (right side) and use a generative model G_3 to generate $\tilde{I}_3$. This is upsampled (green arrow) and then used as the conditioning variable (orange arrow) l_2 for the generative model at the next level, G_2. Together with another noise sample z_2, G_2 generates a difference image $\tilde{h}_2$ which is added to l_2 to create $\tilde{I}_2$. This process repeats across two subsequent levels to yield a final full resolution sample I_0.

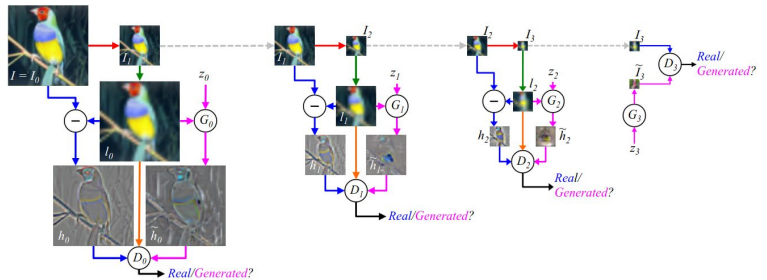


Figure 2: The training procedure for our LAPGAN model. Starting with a 64x64 input image I from our training set (top left): (i) we take $I_0 = I$ and blur and downsample it by a factor of two (red arrow) to produce I_1; (ii) we upsample I_1 by a factor of two (green arrow), giving a low-pass version l_0 of I_0; (iii) with equal probability we use l_0 to create *either* a real *or* a generated example for the discriminative model D_0. In the real case (blue arrows), we compute high-pass $h_0 = I_0 - l_0$ which is input to D_0 that computes the probability of it being real vs generated. In the generated case (magenta arrows), the generative network G_0 receives as input a random noise vector z_0 and l_0. It outputs a generated high-pass image $\tilde{h}_0 = G_0(z_0, l_0)$, which is input to D_0. In both the real/generated cases, D_0 also receives l_0 (orange arrow). Optimizing Eqn. 2, G_0 thus learns to generate realistic high-frequency structure $\tilde{h}_0$ consistent with the low-pass image l_0. The same procedure is repeated at scales 1 and 2, using I_1 and I_2. Note that the models at each level are trained independently. At level 3, I_3 is an 8×8 image, simple enough to be modeled directly with a standard GANs G_3 & D_3.

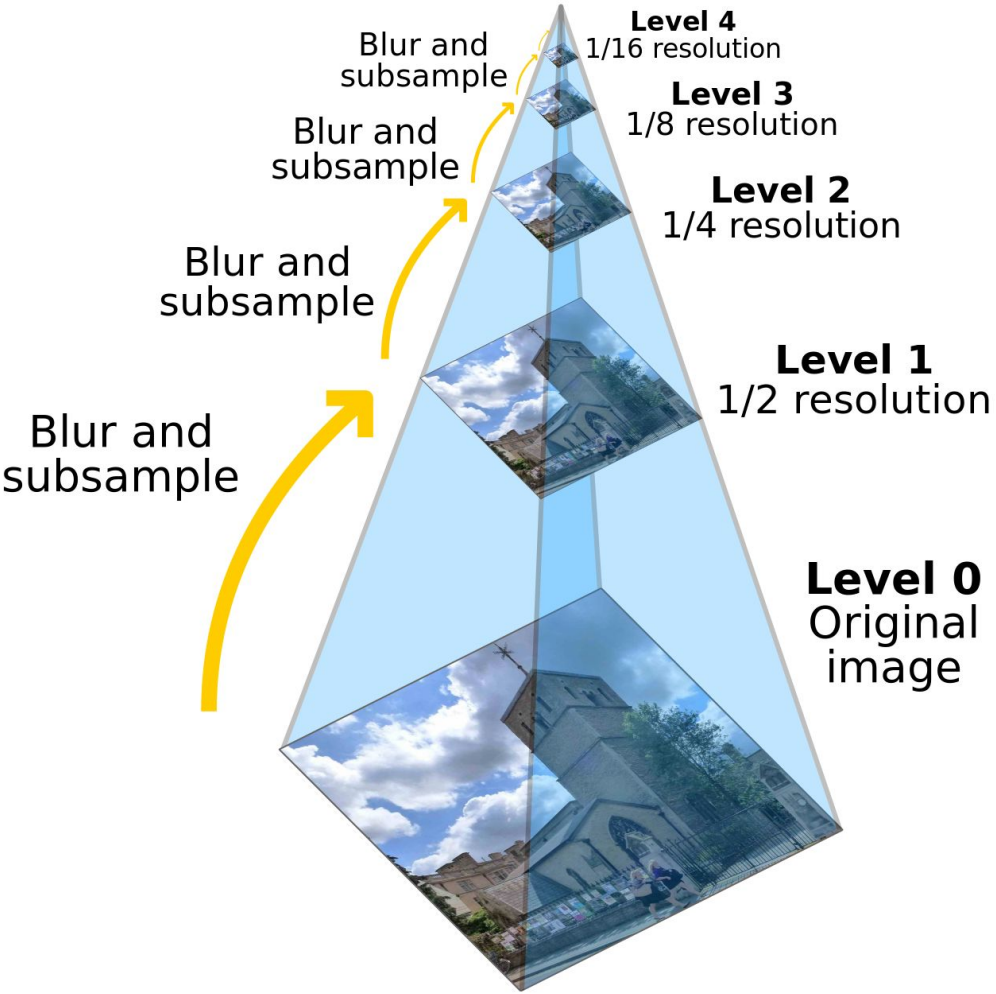


Image Pyramids

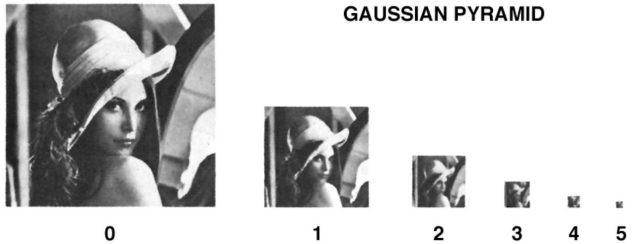


Fig. 4. First six levels of the Gaussian pyramid for the "Lady" image The original image, level 0, meusures 257 by 257 pixels and each higher level array is roughly half the dimensdons of its predecessor. Thus, level 5 measures just 9 by 9 pixels.

(Burt & Adelson, 1983)

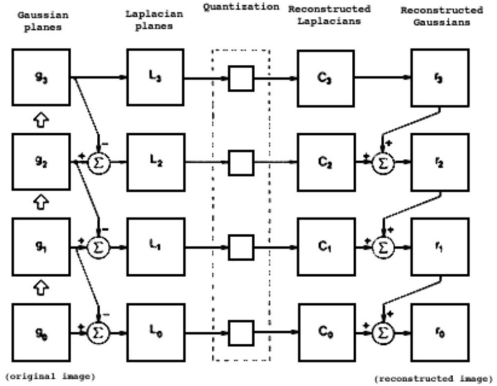
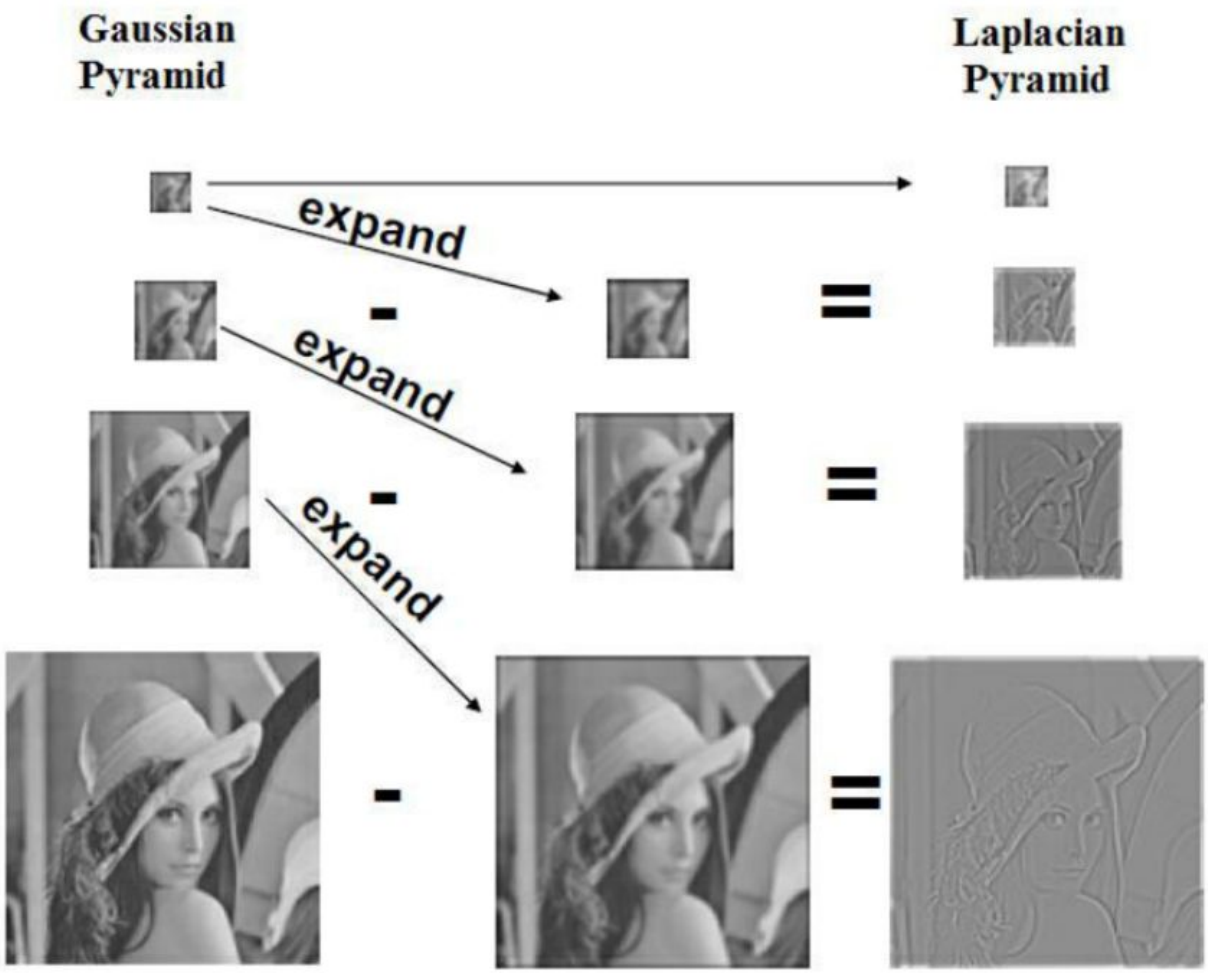
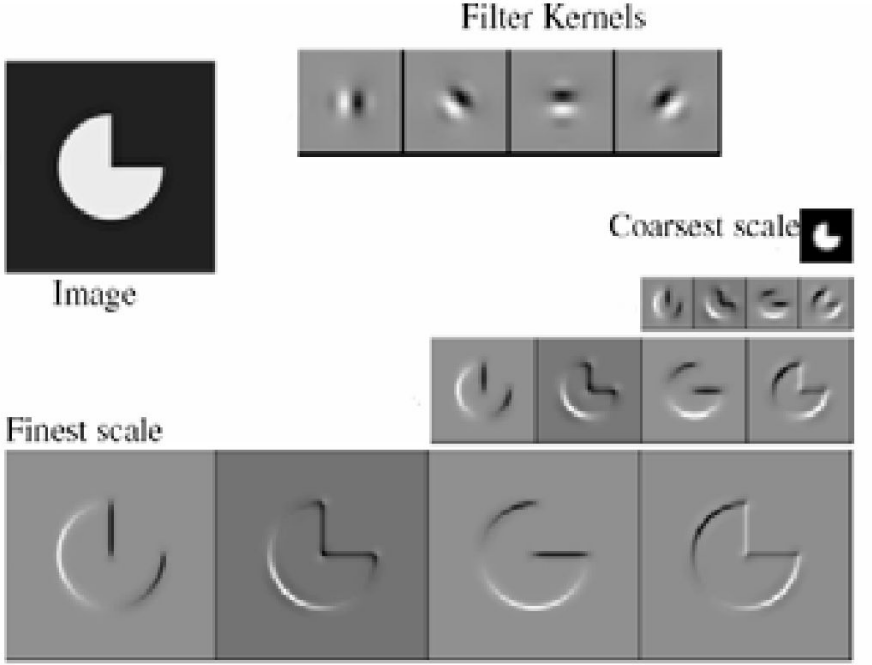


Fig. 10. A summary of the steps in Laplacian pyramid coding and decoding. First, the original image g_0 (lower left) is used to generate Gaussian pyramid levels g_1, g_2, … through repeated local averaging. Levels of the Laplacian pyramid L_0, L_1, … are then computed as the differences between adjacent Gaussian levels. Laplacian pyramid elements are quantized to yield the Laplacian pyramid code C_0, C_1, C_2, …. Finally, a reconstructed image r_0 is generated by summing levels of the code pyramid.

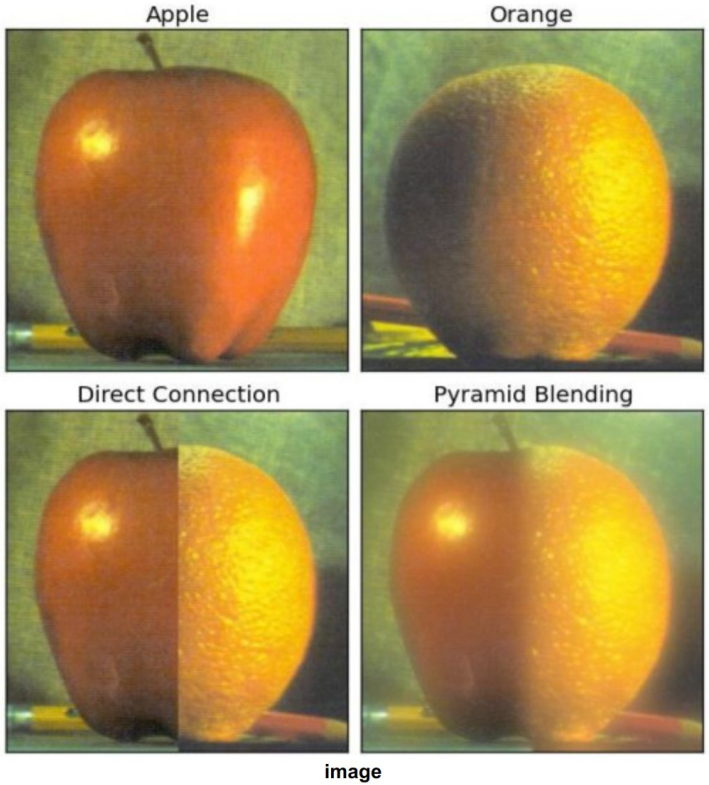

Gaussian Pyramid
Laplacian Pyramid
expand
-
=
expand
-
=
expand
-
=

Steerable Pyramid



Reprinted from “Shiftable MultiScale Transforms,” by Simoncelli et al., IEEE Transactions on Information Theory, 1992, copyright 1992, IEEE

Applications of Pyramids : Blending + Compression



Some form of Interpolation is likely used at every scale of the Pyramid [!]