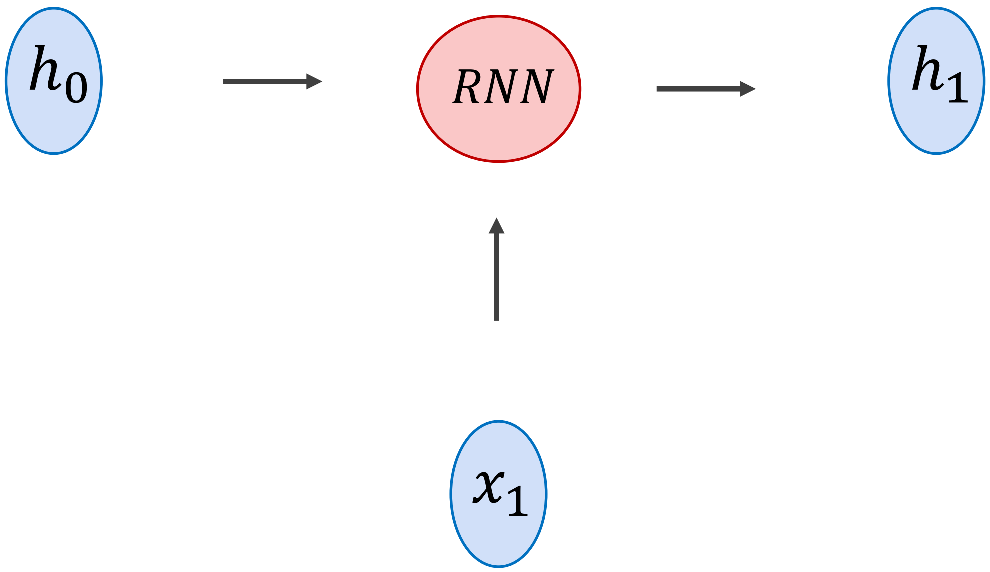
Deep Learning for Vision & Language

Generative Adversarial Networks, Text-to-Scene Introduction

RICE UNIVERSITY

Recap RNNs for your Assignment

$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$



RNN in Pytorch

Recurrent layers

class `torch.nn.RNN`(**args, **kwargs*) [source]

Applies a multi-layer Elman RNN with tanh or ReLU non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(w_{ih} * x_t + b_{ih} + w_{hh} * h_{(t-1)} + b_{hh})$$

where h_t is the hidden state at time *t*, and x_t is the hidden state of the previous layer at time *t* or $input_t$ for the first layer. If nonlinearity='relu', then *ReLU* is used instead of *tanh*.

Parameters:

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers.
- **nonlinearity** – The non-linearity to use ['tanh'|'relu']. Default: 'tanh'
- **bias** – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If True, becomes a bidirectional RNN. Default: False

LSTM Cell (Long Short-Term Memory)

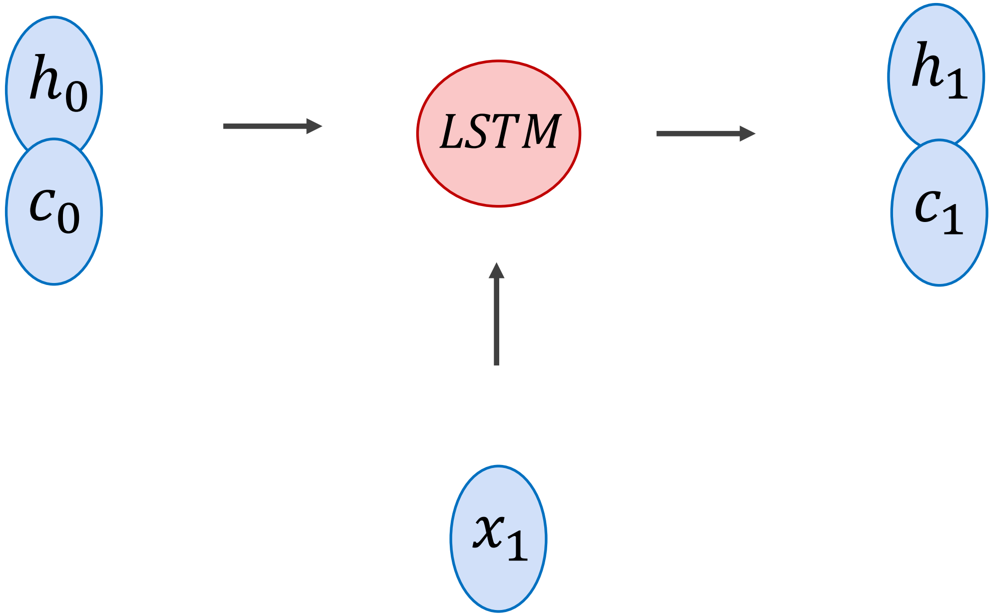
$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \tag{7}$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \tag{8}$$

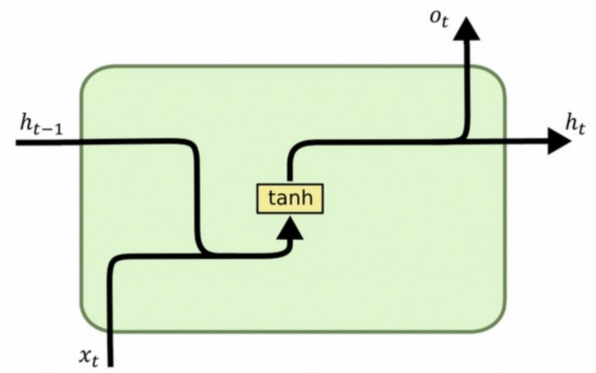
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \tag{9}$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \tag{10}$$

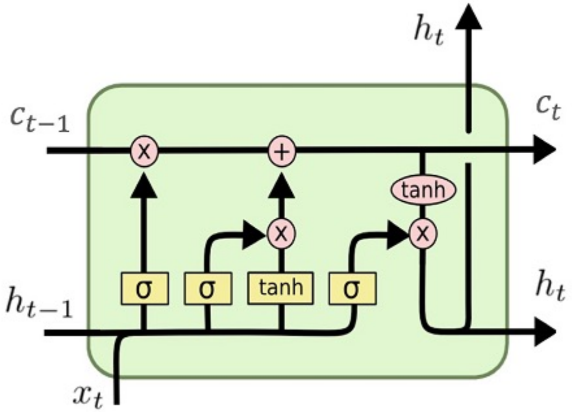
$$h_t = o_t \tanh(c_t) \tag{11}$$



https://ieeexplore.ieee.org/abstract/document/6795963



RNN



LSTM
(Long-Short Term Memory)

LSTM in Pytorch

class `torch.nn.LSTM`(**args, **kwargs*) [source]

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
i_t &= \mathrm{sigmoid}(W_{ii} x_t + b_{ii} + W_{hi} h_{(t-1)} + b_{hi}) \\
f_t &= \mathrm{sigmoid}(W_{if} x_t + b_{if} + W_{hf} h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig} x_t + b_{ig} + W_{hc} h_{(t-1)} + b_{hg}) \\
o_t &= \mathrm{sigmoid}(W_{io} x_t + b_{io} + W_{ho} h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

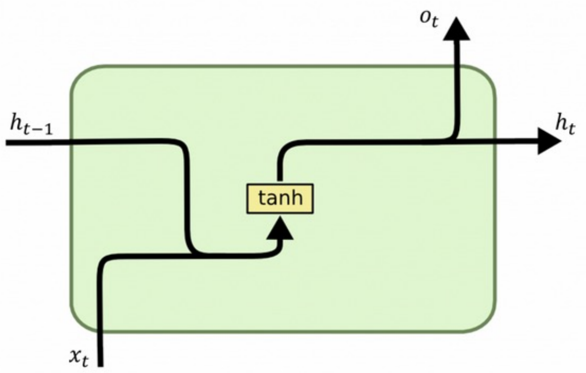
where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the hidden state of the previous layer at time t or $input_t$ for the first layer, and i_t, f_t, g_t, o_t are the input, forget, cell, and out gates, respectively.

Parameters:

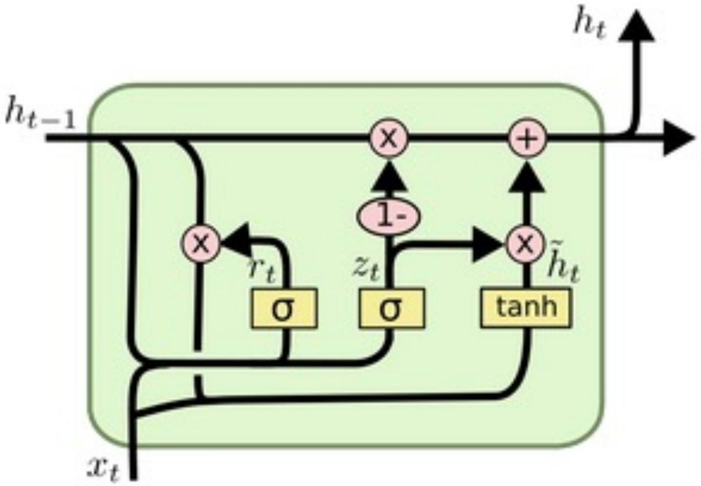
- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers.
- **bias** – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If True, becomes a bidirectional RNN. Default: False

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU in Pytorch



RNN



class `torch.nn.GRU`(**args, **kwargs*) [source]

Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned} r_t &= \mathrm{sigmoid}(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\ z_t &= \mathrm{sigmoid}(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\ h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)} \end{aligned}$$

where h_t is the hidden state at time t, x_t is the hidden state of the previous layer at time t or $input_t$ for the first layer, and r_t, z_t, n_t are the reset, input, and new gates, respectively.

Parameters:

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers.
- **bias** – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If True, becomes a bidirectional RNN. Default: False

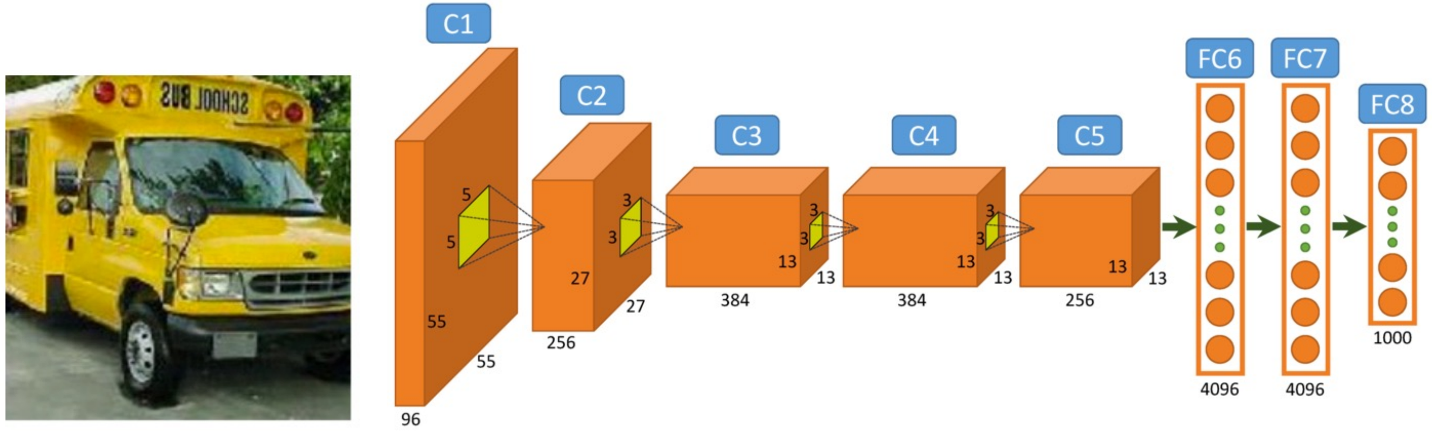
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Today’s Class

- Adversarial Examples – Input Optimization
- Generative Adversarial Networks (GANs)
- Conditional GANs

What we have been doing: Optimize weights in the network to predict bus (correct class).

I $\quad\quad y = f(I; w) \quad\quad L(y, bus)$

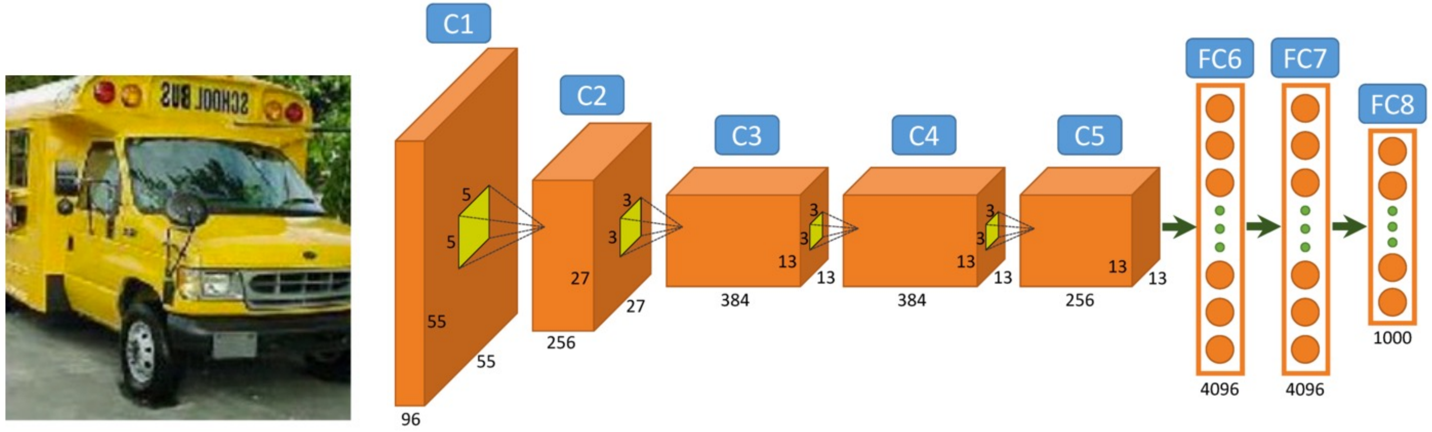


bus

$$w = w - \lambda \frac{\partial L}{\partial w}$$

New Idea: Create Adversarial Inputs by optimizing the input image to predict ostrich (wrong class).

I $\qquad y = f(I;w) \qquad L(y, ostrich)$

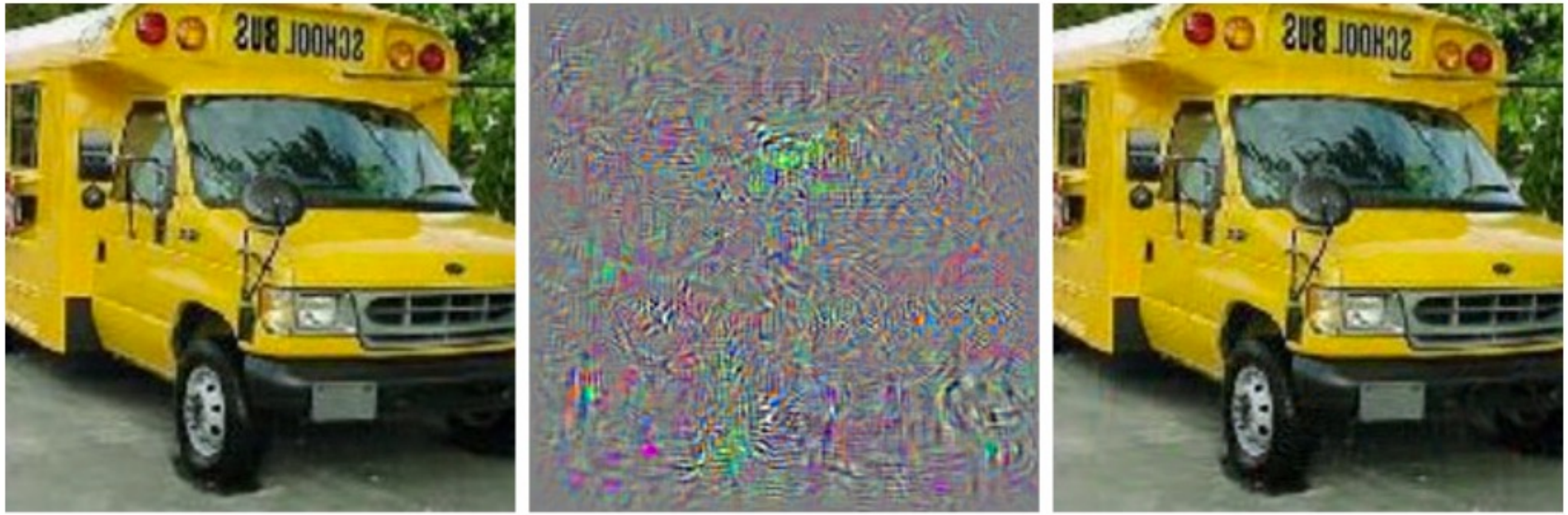


ostrich

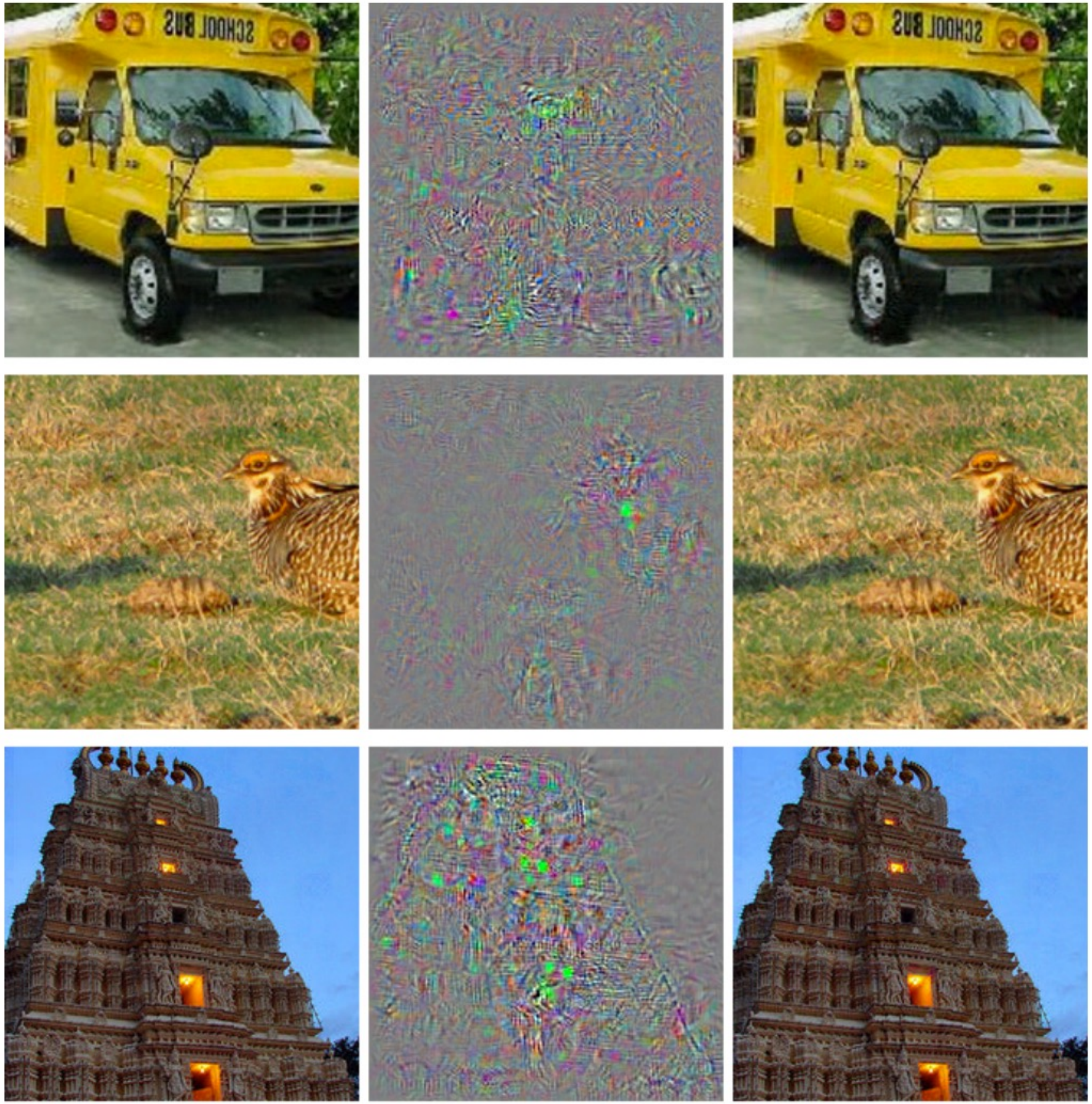
$$I = I - \lambda \frac{\partial L}{\partial I}$$

Work on Adversarial examples by Goodfellow et al. , Szegedy et. al., etc.

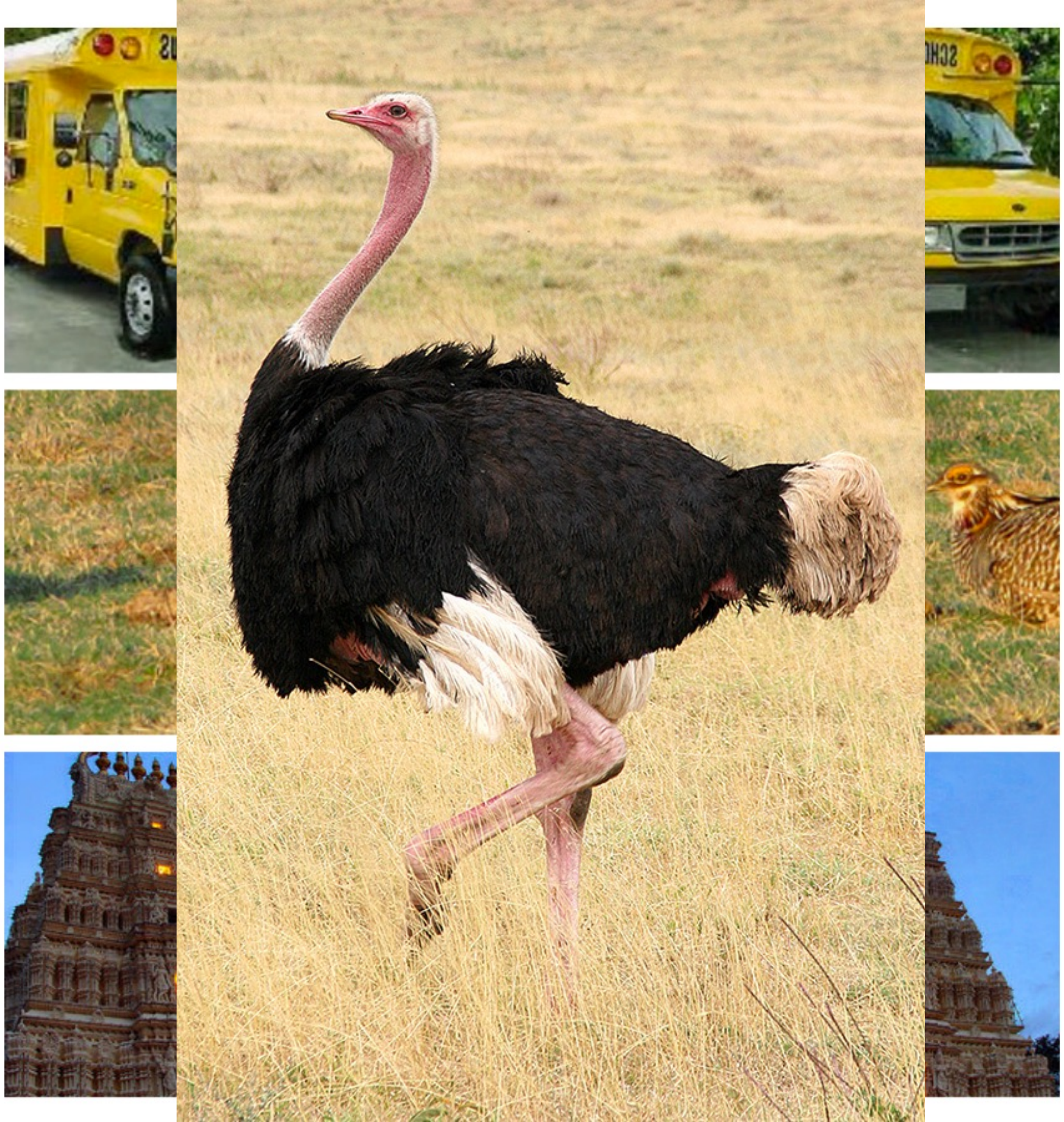
Convnets (optimize input to predict ostrich)



Work on Adversarial examples by Goodfellow et al. , Szegedy et. al., etc.



All get predicted as ostrich



Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images

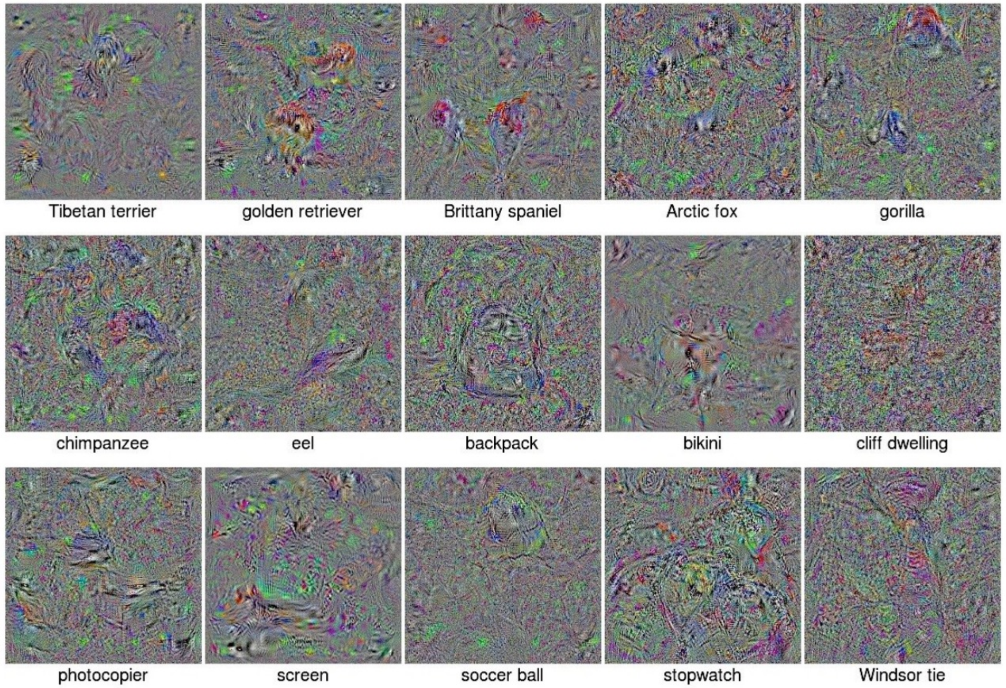
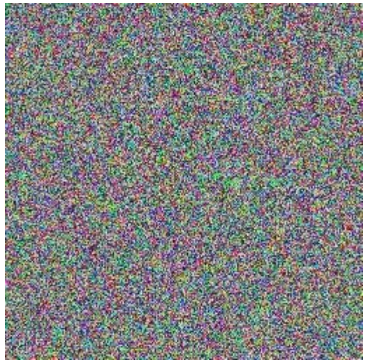


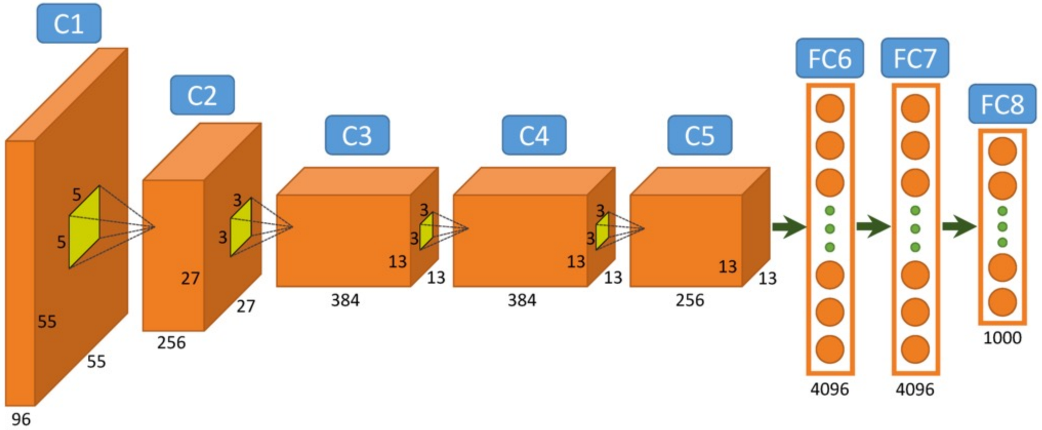
Figure 13. Images found by maximizing the softmax output for classes via gradient ascent [11, 26]. Optimization begins at the ImageNet mean (plus small Gaussian noise to break symmetry) and continues until the DNN confidence for the target class reaches 99.99%. Images are shown with the mean subtracted. Adding regularization makes images more recognizable but results in slightly lower confidence scores (see supplementary material).

Anh Nguyen, Jason Yosinski, Jeff Clune, 2014

New Idea: Create Adversarial Inputs by optimizing the input image to predict ostrich (wrong class).

I $\qquad$ $y = f(I; w)$ $\qquad$ $L(y, parking\ meter)$





parking meter

$$I = I - \lambda \frac{\partial L}{\partial I}$$

Work on Adversarial examples by Goodfellow et al. , Szegedy et. al., etc.

parking meter: 0.999679

Total Variation Regularization

A second richer regulariser is *total variation* (TV) $\mathcal{R}_{V^\beta}(\mathbf{x})$, encouraging images to consist of piece-wise constant patches. For continuous functions (or distributions) $f : \mathbb{R}^{H\times W} \supset \Omega \to \mathbb{R}$, the TV norm is given by:

$$\mathcal{R}_{V^\beta}(f) = \int_\Omega \left(\left(\frac{\partial f}{\partial u}(u,v) \right)^2 + \left(\frac{\partial f}{\partial v}(u,v) \right)^2 \right)^{\frac{\beta}{2}} du\, dv$$

where $\beta = 1$. Here images are discrete ($\mathbf{x} \in \mathbb{R}^{H\times W}$) and the TV norm is replaced by the finite-difference approximation:

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}.$$

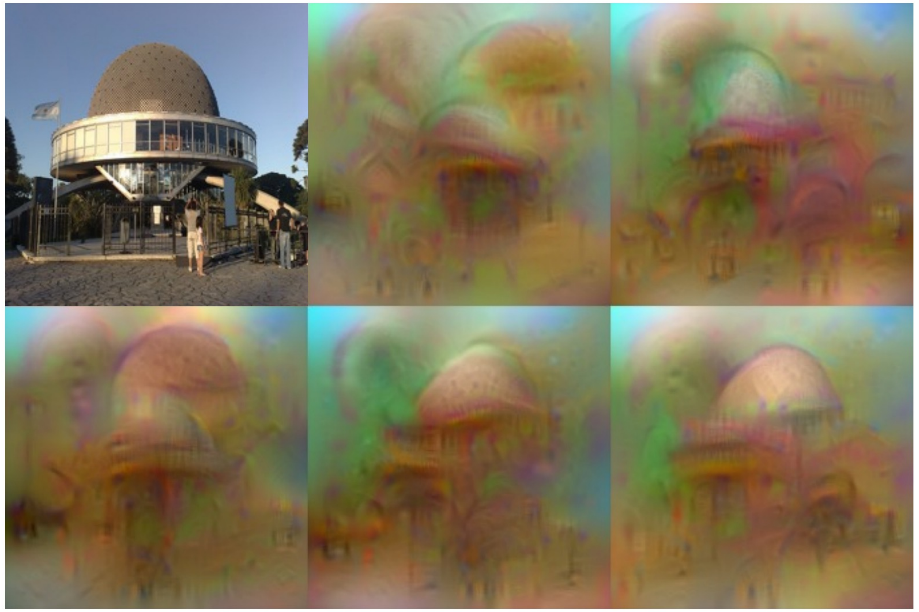
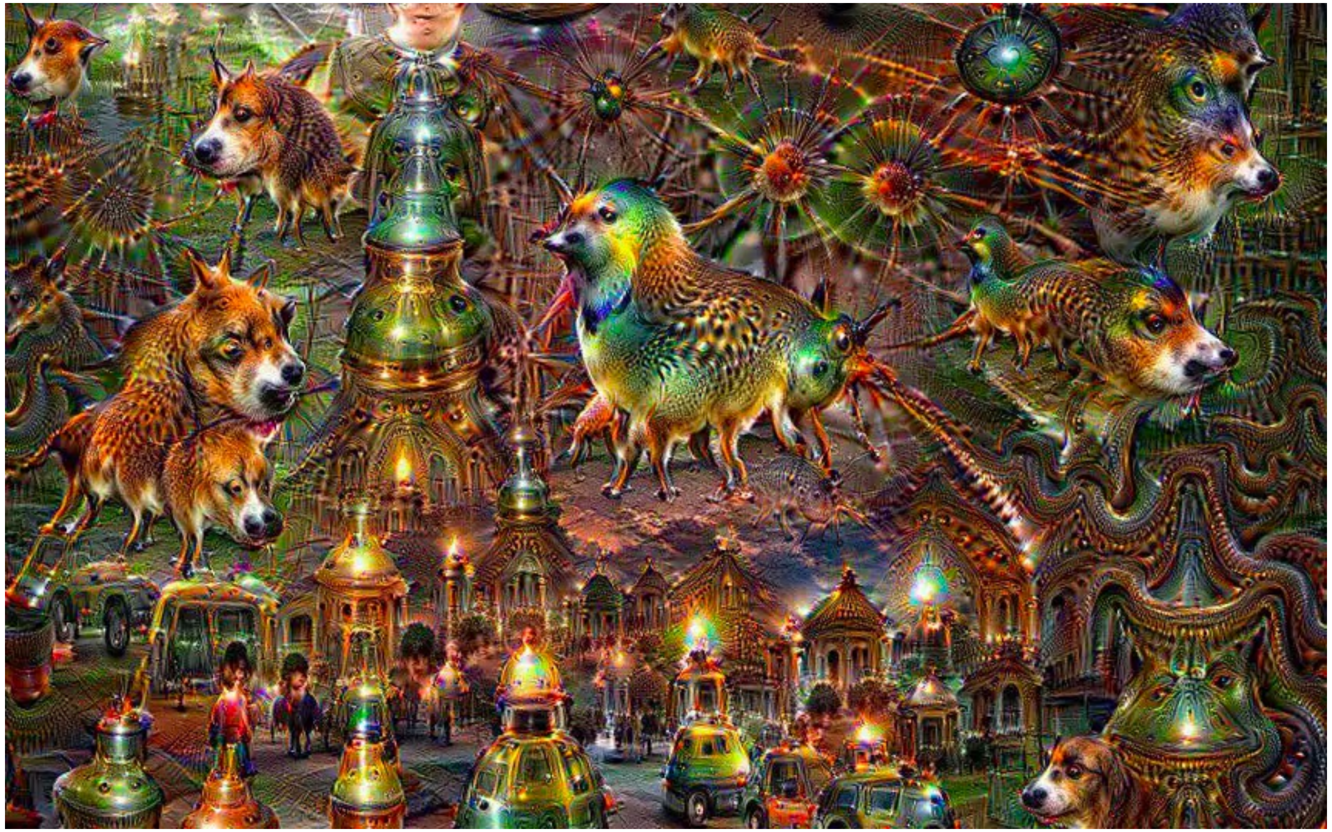


Figure 1. **What is encoded by a CNN?** The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference CNN[13] (before the softmax is applied) trained on the ImageNet data. From the viewpoint of the model, all these images are practically equivalent. This image is best viewed in color/screen.

Mahendran and Vedaldi, Understanding Deep Image Representations by Inverting Them, 2014

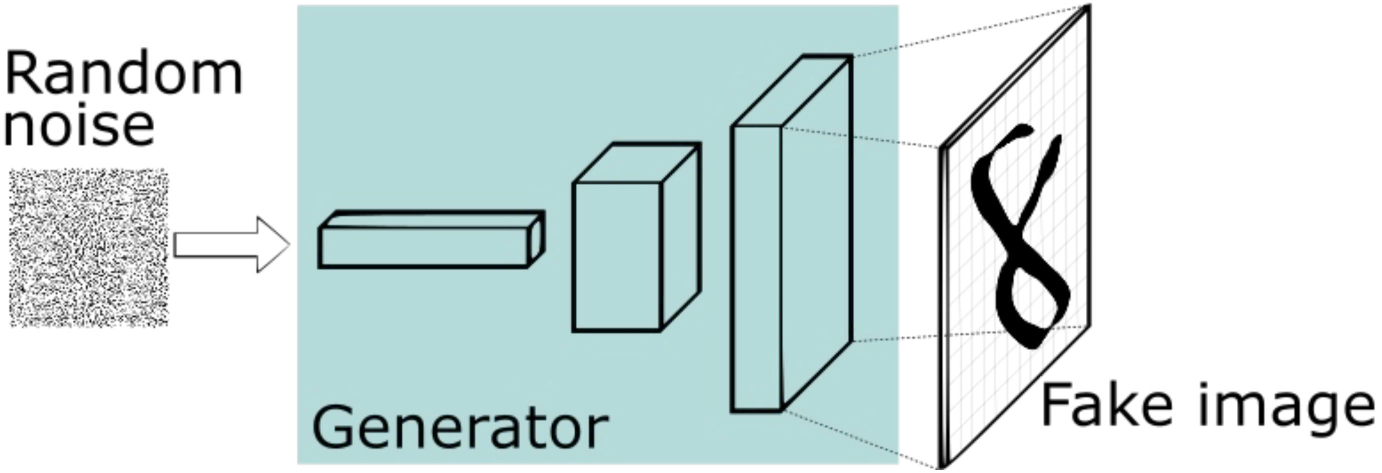
Taking the idea to the extreme: Google's DeepDream



https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

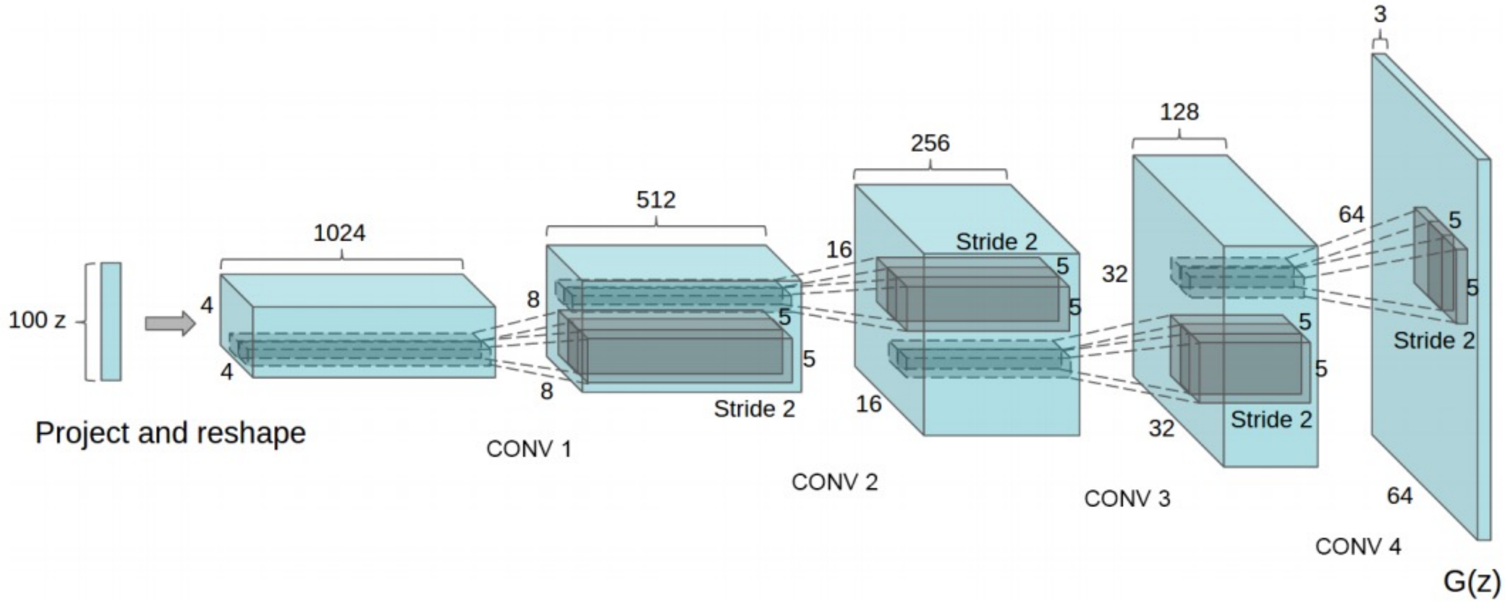
Generate your own in Pytorch: https://github.com/XavierLinNow/deepdream_pytorch

Generative Adversarial Networks (GAN) [Goodfellow et al 2014]



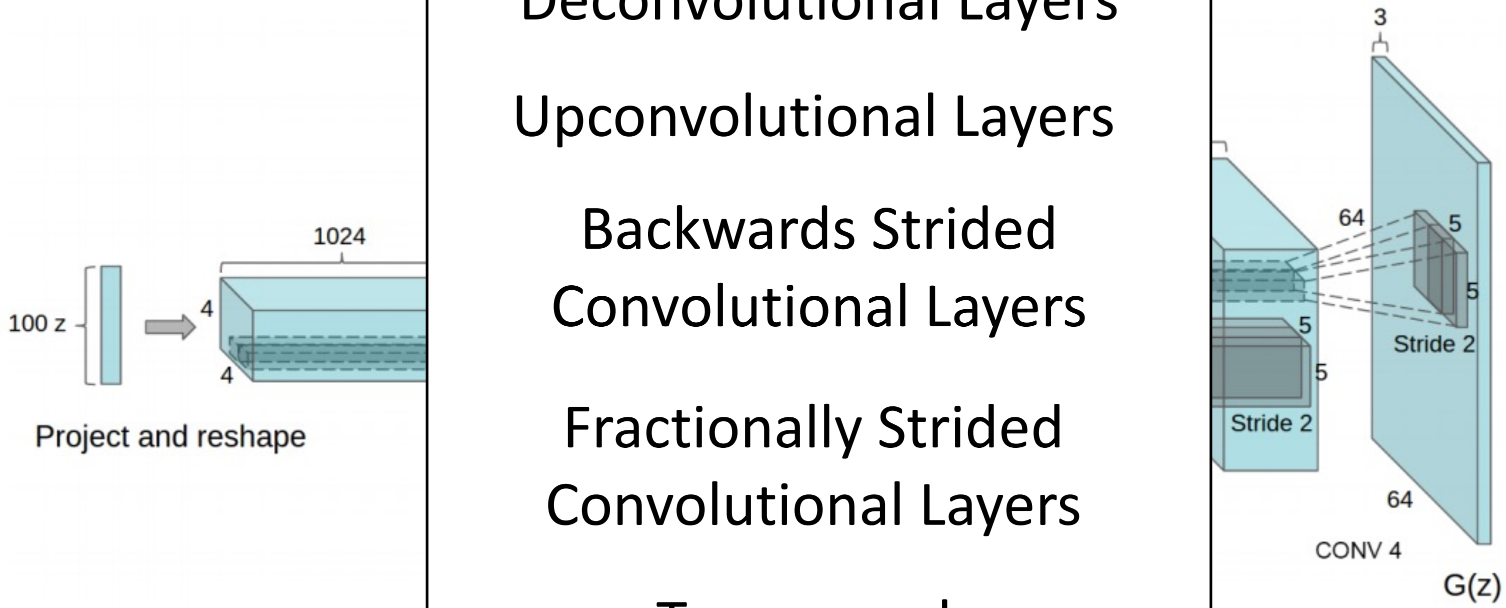
https://deeplearning4j.org/generative-adversarial-network

Generative Network (closer look)



Radford et. al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. ICLR 2016

Generative Network (closer look)



Deconvolutional Layers

Upconvolutional Layers

Backwards Strided Convolutional Layers

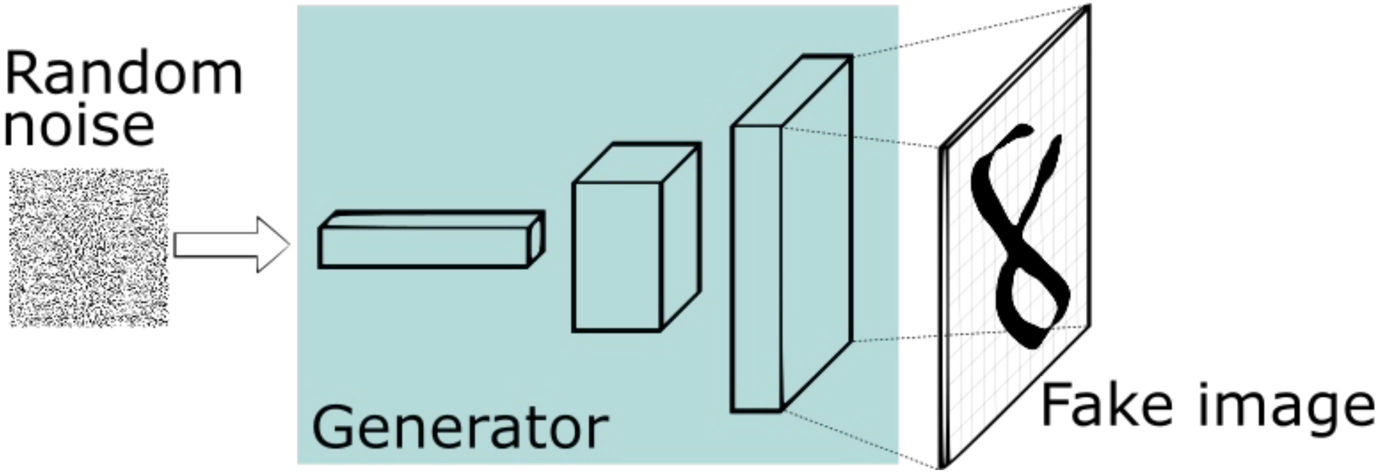
Fractionally Strided Convolutional Layers

Transposed Convolutional Layers

Spatial Full Convolutional Layers

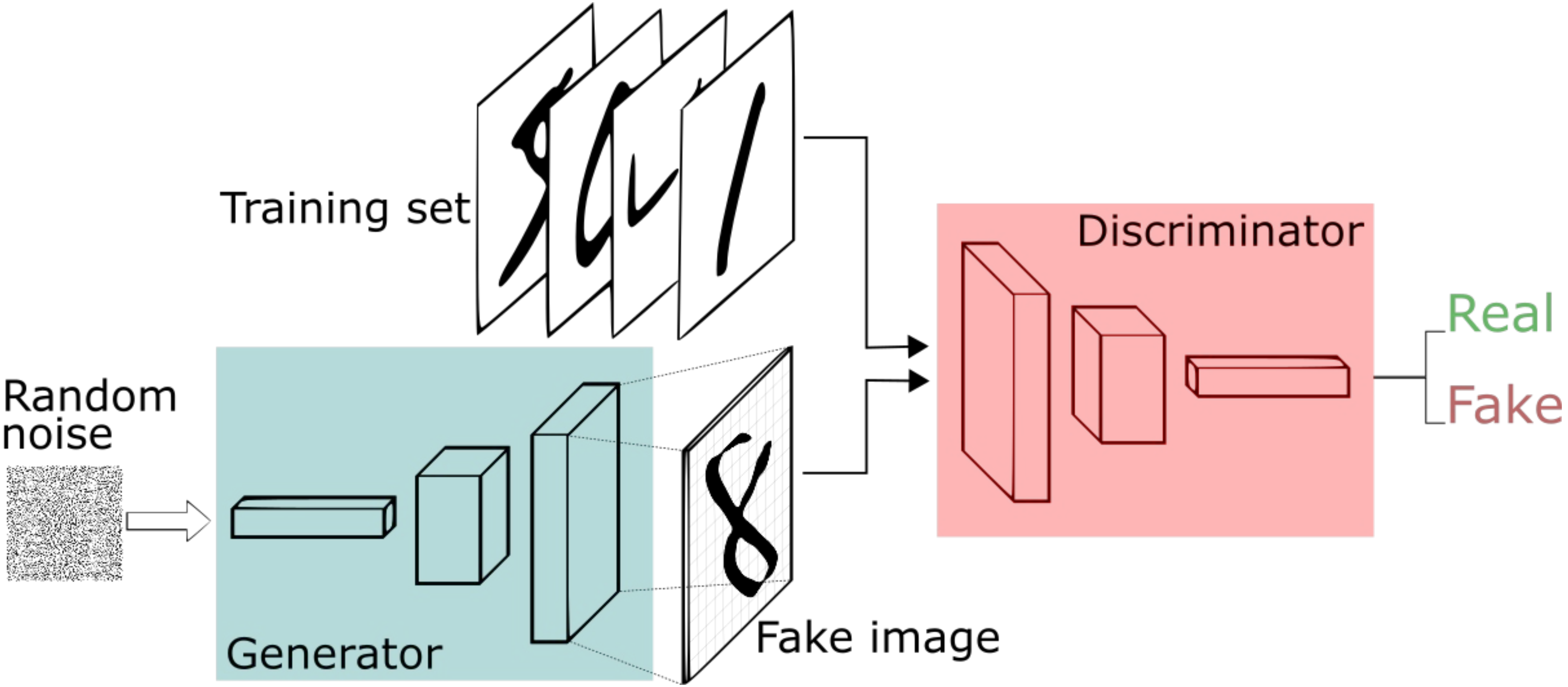
Radford et. al. … ion
Learning with … ative
Adversarial Networks. ICLR 2016

Generative Adversarial Networks (GAN) [Goodfellow et al.]



https://deeplearning4j.org/generative-adversarial-network

Generative Adversarial Networks (GAN) [Goodfellow et al.]



https://deeplearning4j.org/generative-adversarial-network

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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.

Goodfellow et al. NeurIPS 2014

Update Discriminator D

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.

Goodfellow et al. NeurIPS 2014

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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

Update Generator G

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

end for

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

Goodfellow et al. NeurIPS 2014

Until Desirable Results are Achieved?

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:

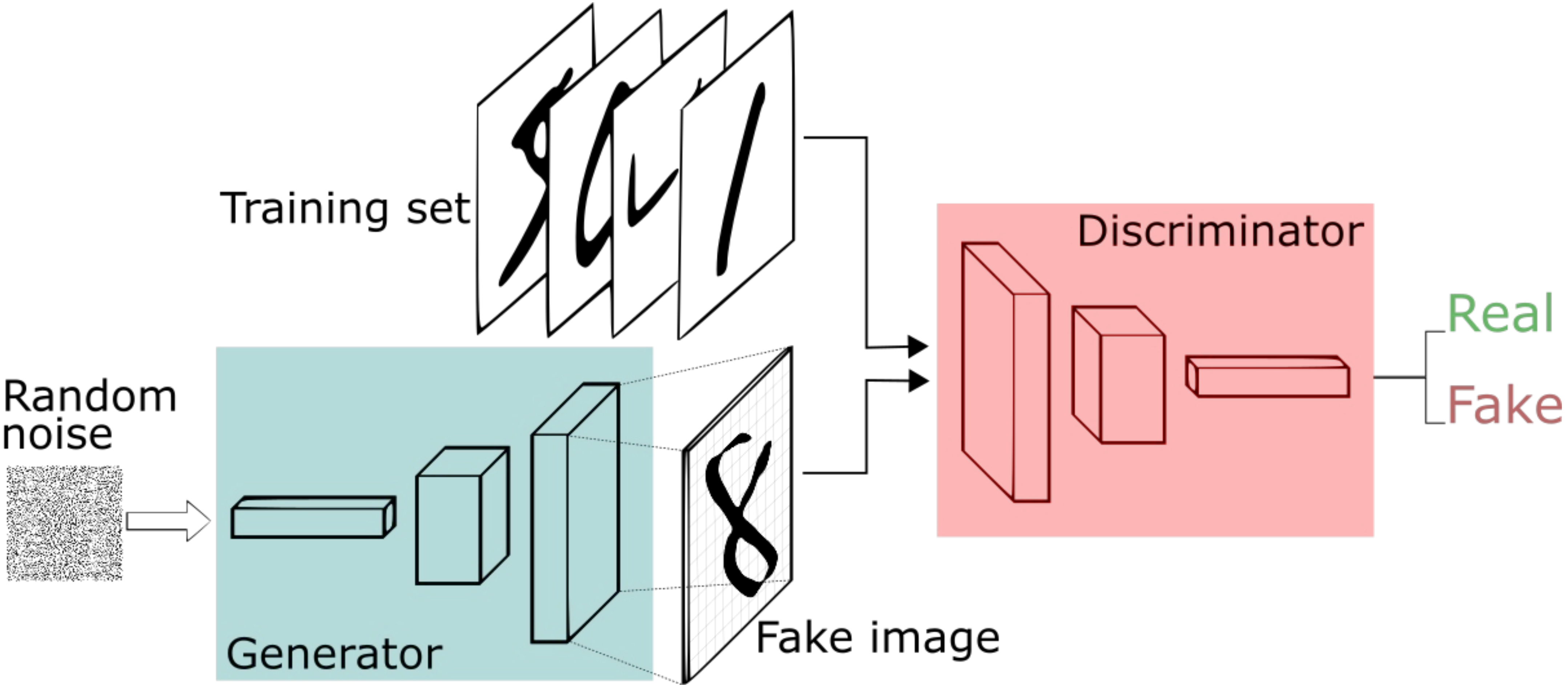
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Goodfellow et al. NeurIPS 2014

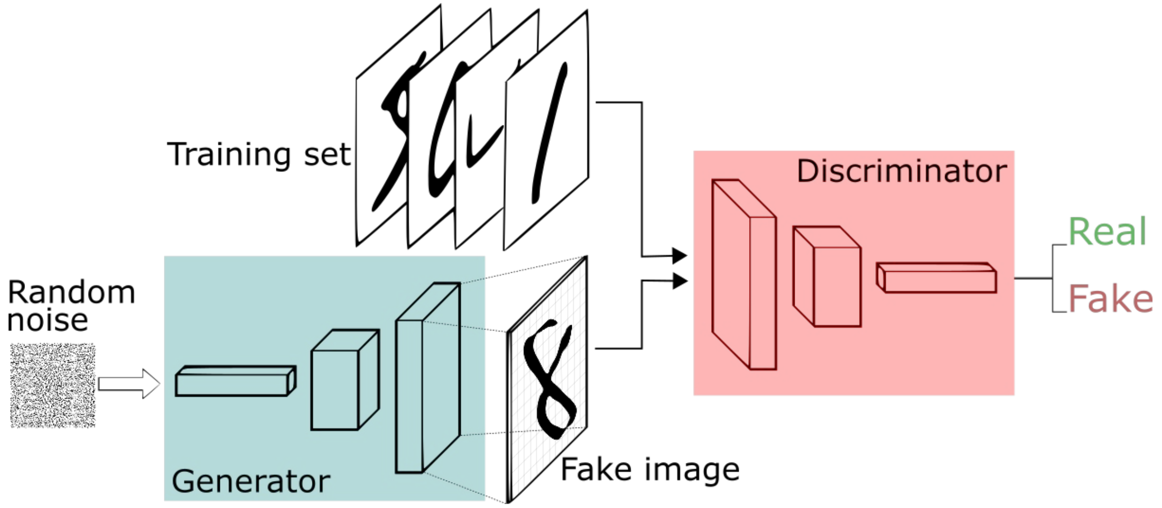
Generative Adversarial Networks (GAN) [Goodfellow et al.]



https://deeplearning4j.org/generative-adversarial-network

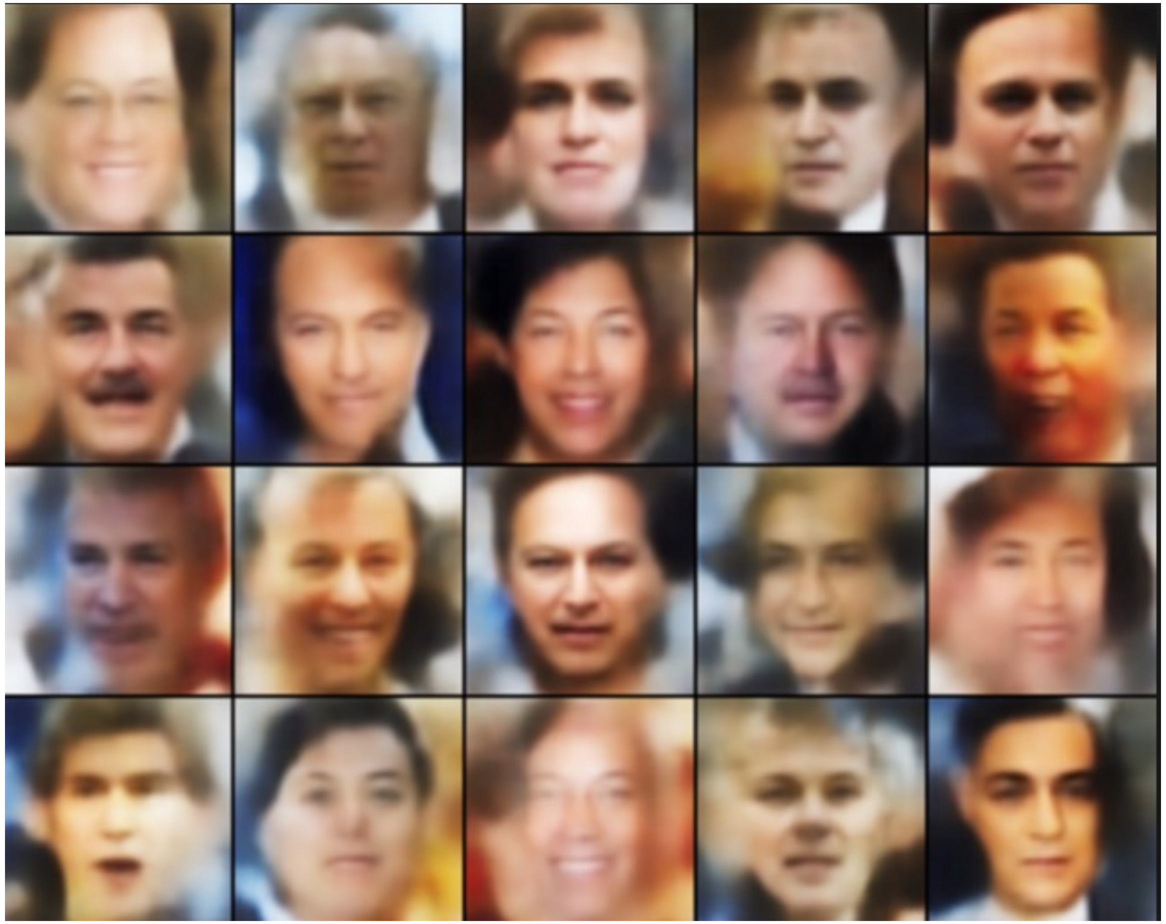
Generative Adversarial Networks (GAN) [Goodfellow et al.]

- GANs are hard to train, loss for the discriminator and generator might fluctuate.
- There are many choices for loss, and other auxiliary signals.
- Training of these models is even less well understood than for other deep models.



https://deeplearning4j.org/generative-adversarial-network

Basic GAN Results (Example implementation is provided in Pytorch's examples)



http://torch.ch/blog/2015/11/13/gan.html

NVidia’s progressive GANs ICLR 2018

Google’s BigGAN

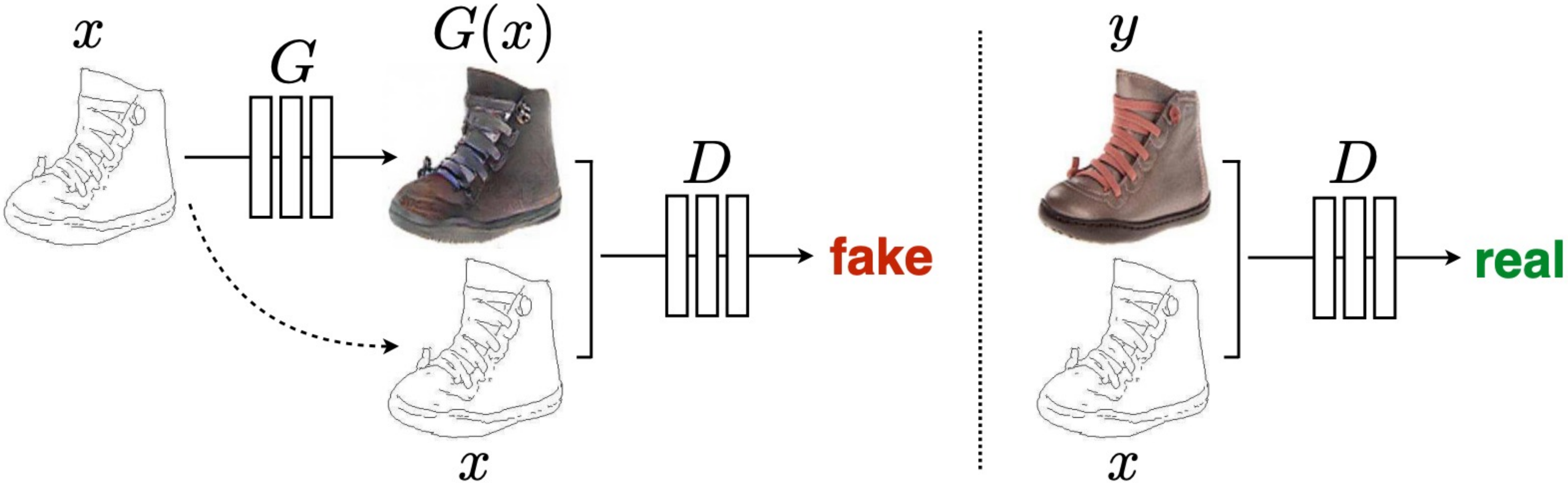
Google's BigGAN

Teddy Bear

Microphone

http://aiweirdness.com/post/179626595787/the-creepiest-images-generated-by-biggan

Conditional GANs: Input is not just Noise

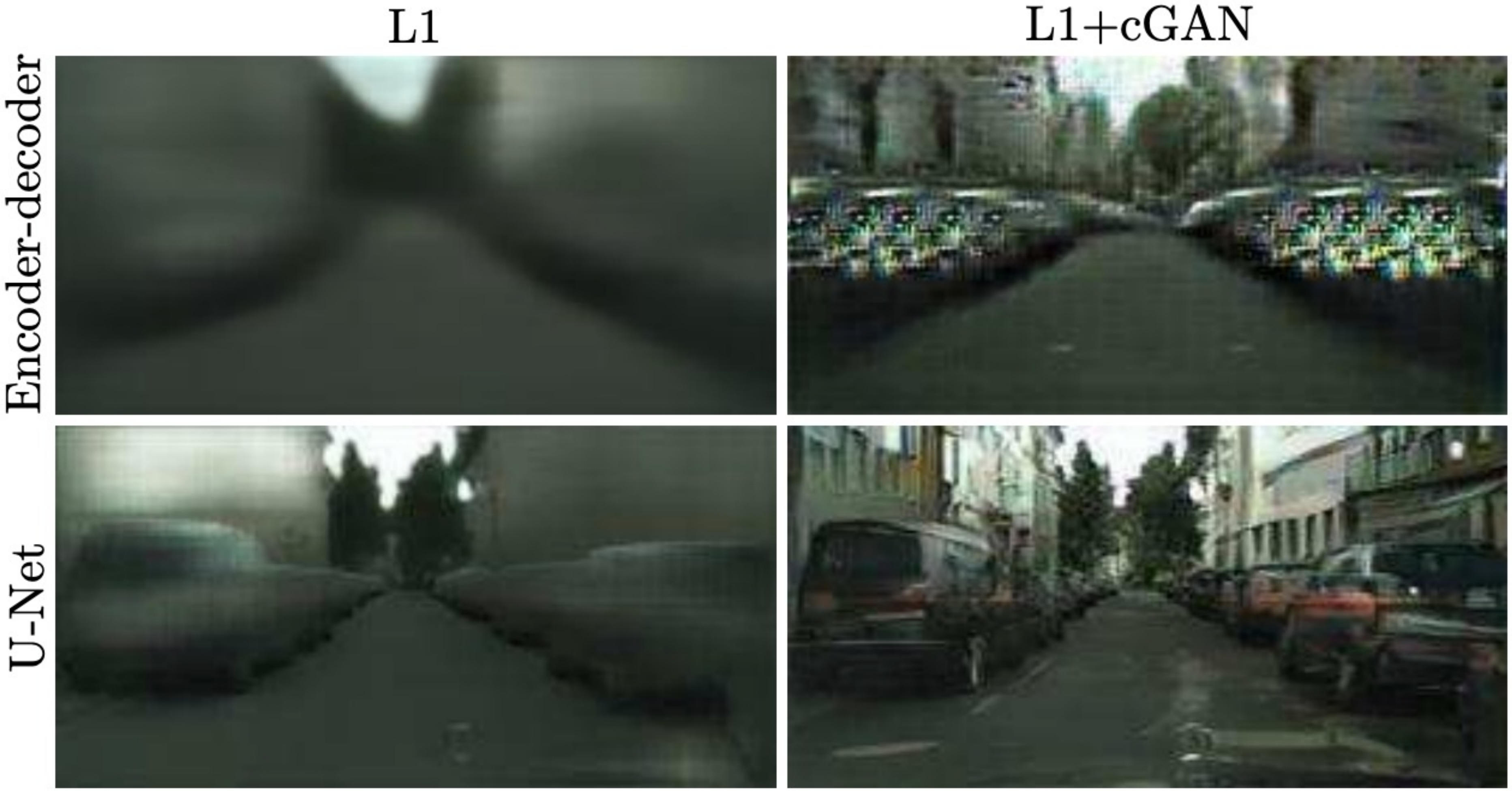


Isola et al. CVPR 2017: Image-to-Image Translation with Conditional Adversarial Networks

Conditional GANs: Also Hard to Train

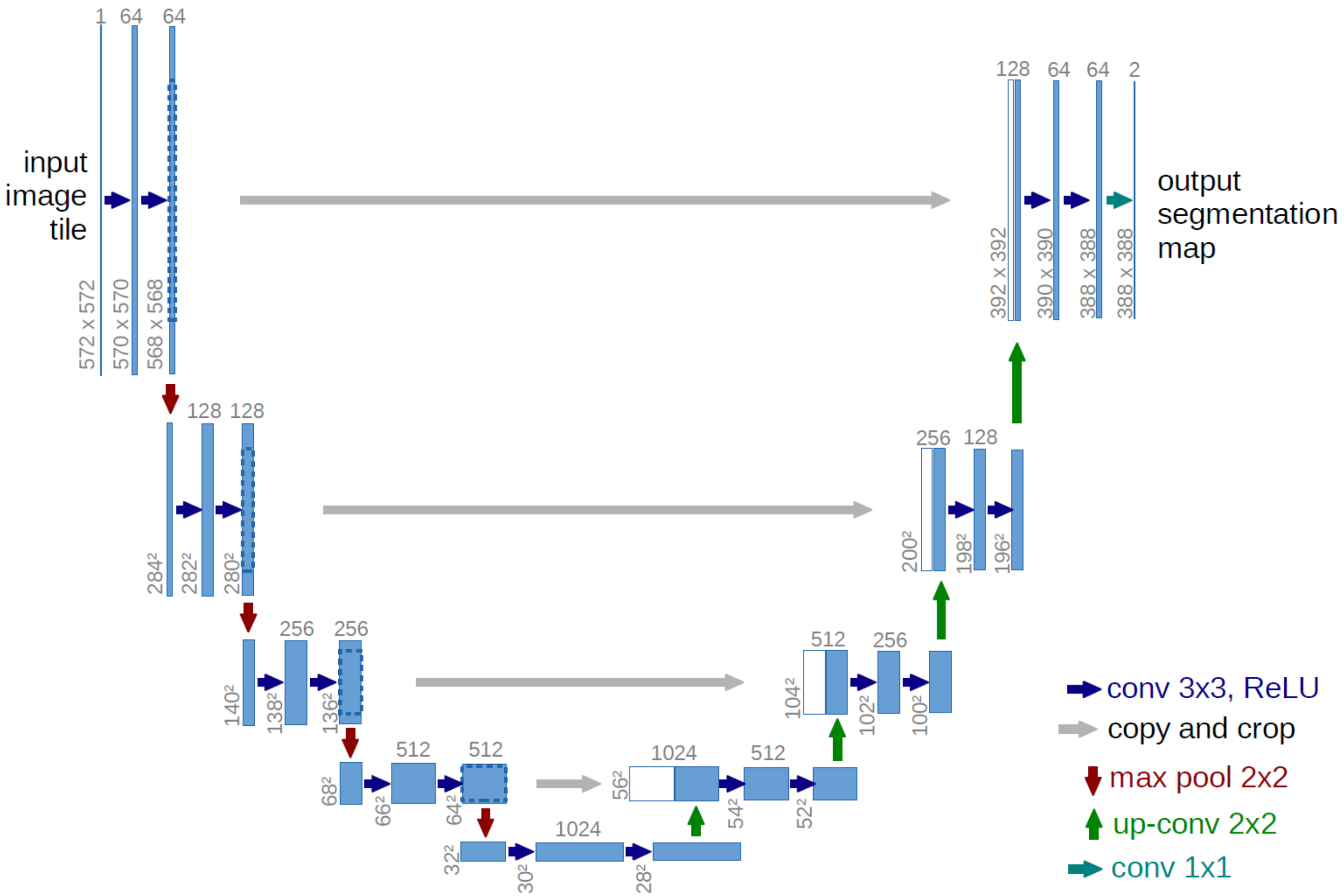
Result they obtained with a regular Fully Convolutional Network

Result they obtained with a U-Net network (with skip-connections)



Isola et al. CVPR 2017: Image-to-Image Translation with Conditional Adversarial Networks

Conditional GANs: Also Hard to Train



Ronneberger et al. MICCAI 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation

Conditional GANs / Text-conditioned

AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks

Tao Xu*[1], Pengchuan Zhang[2], Qiuyuan Huang[2],
Han Zhang[3], Zhe Gan[4], Xiaolei Huang[1], Xiaodong He[2]

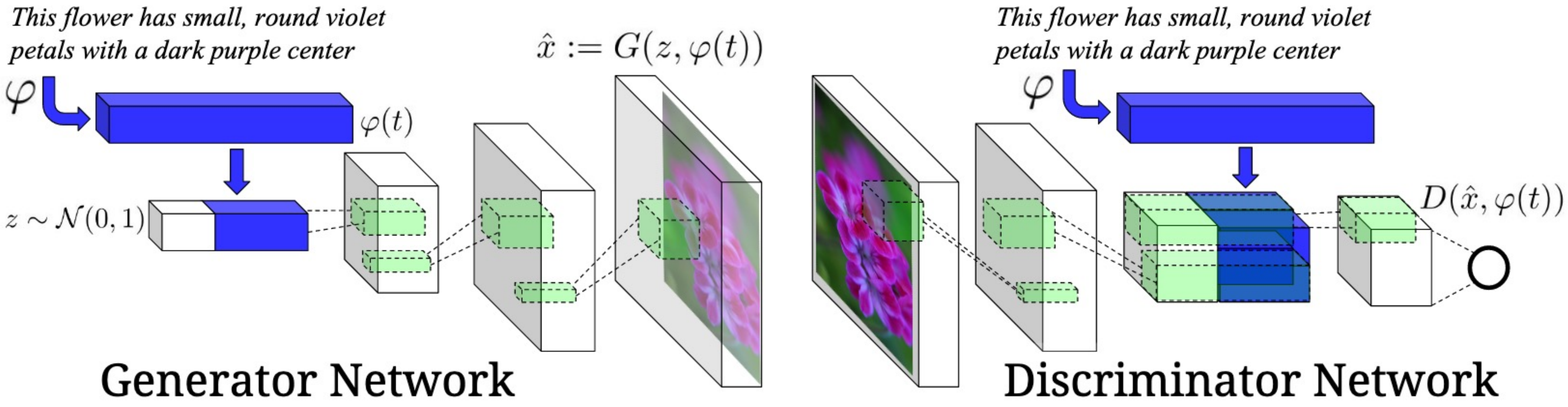
[1]Lehigh University [2]Microsoft Research [3]Rutgers University [4]Duke University
{tax313, xih206}@lehigh.edu, {penzhan, qihua, xiaohe}@microsoft.com
han.zhang@cs.rutgers.edu, zhe.gan@duke.edu

Conditional GANs / Text-conditioned

Generative Adversarial Text to Image Synthesis

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran REEDSCOT[1], AKATA[2], XCYAN[1], LLAJAN[1]
Bernt Schiele, Honglak Lee SCHIELE[2],HONGLAK[1]
[1] University of Michigan, Ann Arbor, MI, USA (UMICH.EDU)
[2] Max Planck Institute for Informatics, Saarbrücken, Germany (MPI-INF.MPG.DE)

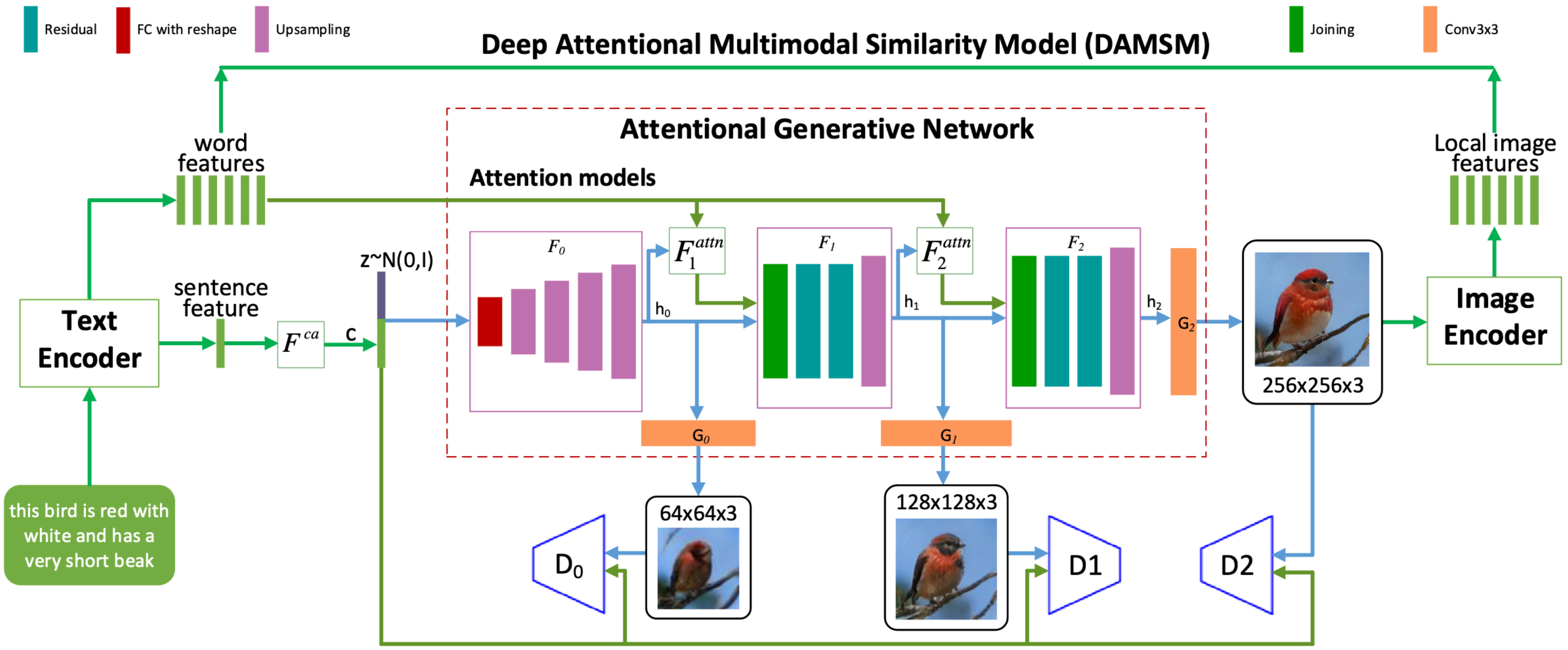
Conditional GANs / Text-conditioned



Conditional GANs / Text-conditioned

this small bird has a pink breast and crown, and black primaries and secondaries.

Conditional GANs / Text-conditioned



Conditional GANs / Text-conditioned

Questions