

# Lecture Notes for **Neural Networks and Machine Learning**



Overview of GANs  
and PyTorch



# Logistics and Agenda

- Logistics
  - Student Presentations Today and Next Lecture
- Agenda
  - Review from Last Time,
  - GAN Demo
  - **Student Presentation:** Representational Learning with Deep Convolutional GANs, Radford
  - LS-GAN
  - Practical GANs
  - Wasserstein GAN (Next time)
  - WGAN-GP (Next Time)
  - Big GAN (Next Next Time)



# Last Time

- Simple GANS

- Discriminator:

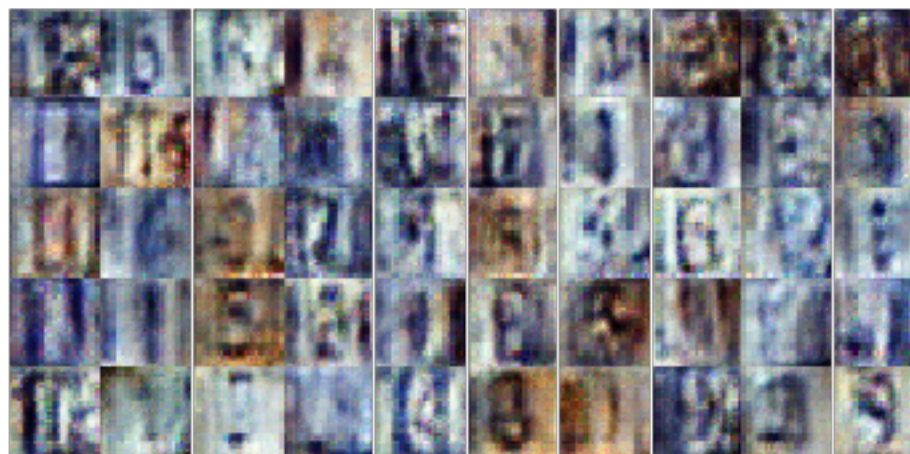
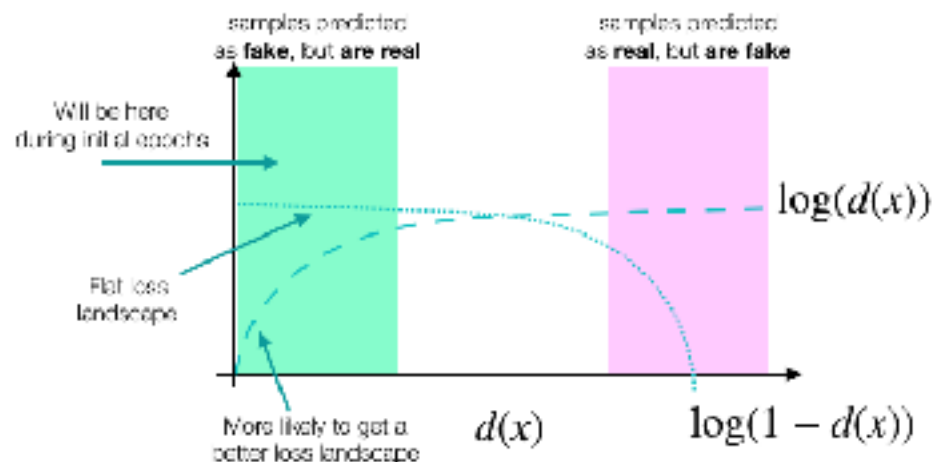
$$\max_d \mathbb{E}_{x \leftarrow p_{data}} [\log d(x)] + \mathbb{E}_{x \leftarrow g(z)} [\log(1 - d(x))]$$

Same as minimizing the binary cross entropy of the model with: (1) **labeled data** and (2) **generated data**!

- Generator:

$$\max_g \mathbb{E}_{x \leftarrow g(z)} [\log(d(x))] \quad \text{Freeze Discriminator Weights}$$

Same as minimizing the binary cross entropy of "misabeled" generated data!



# How to Train your (dra) GAN

deeplearning.ai presents  
Heroes of Deep Learning

**Ian Goodfellow**

Research Scientist at ~~Google~~ Brain

Apple



Every time Ian Goodfellow has a tutorial, It should be called:

“GAN-splaining with Ian”

—Geoffrey Hinton, Probably



# Simple Training Approach

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

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

**end for**

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

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

**end for**

Does this work?!

# Still No! Why?



# A bag of tricks from F. Chollet

The process of training GANs and tuning GAN implementations is notoriously difficult. There are a number of known tricks you should keep in mind. Like most things in deep learning, it's **more alchemy than science**: **these tricks are heuristics, not theory-backed guidelines**. They're supported by a level of intuitive understanding of the phenomenon at hand, and they're known to work well empirically, although not necessarily in every context.

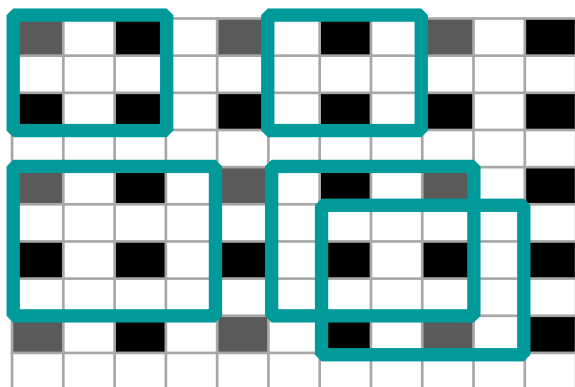
—Francois Chollet, DL with Python

- Use tanh for generator output, not a sigmoid
  - We typically normalize inputs to a NN to be from  $[-1, 1]$ , generator needs to mirror this squashing
- Sample from Normal Distribution
  - Everyone else is doing it (practically whatever we do here should help to create a latent space easy to sample from)
- Random is more robust in optimizer and labels
  - GANs get stuck a lot
- Sparse gradients are not your friend here
  - No max pooling, no ReLU 🤨
- Make decoder upsampling multiple of stride...
  - Unequal pixel coverage (checkerboards)



# What do you mean checkerboard patterns?

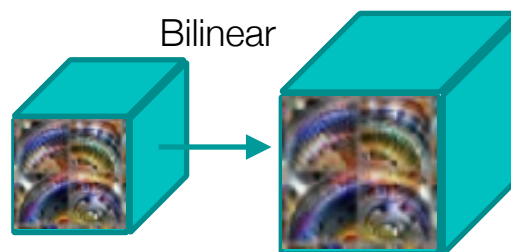
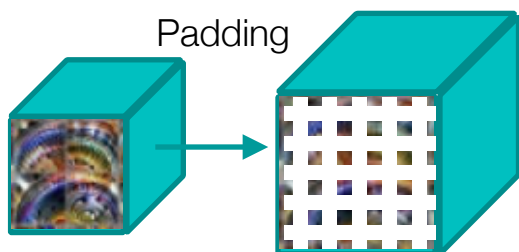
- Option 1: padding, Kernel size should be a symmetric multiple of the stride



Bias needs to account for both when 2 pixels and four pixels are in the kernel space

Multiple of stride ensures that same number of active pixels are there.

- Option 2: Bilinear resizing of activations, rather than zero insertion or Transpose convolution





# Some Results of Generation

Goodfellow et al. "Generative Adversarial Networks" (NeurIPS 2014)



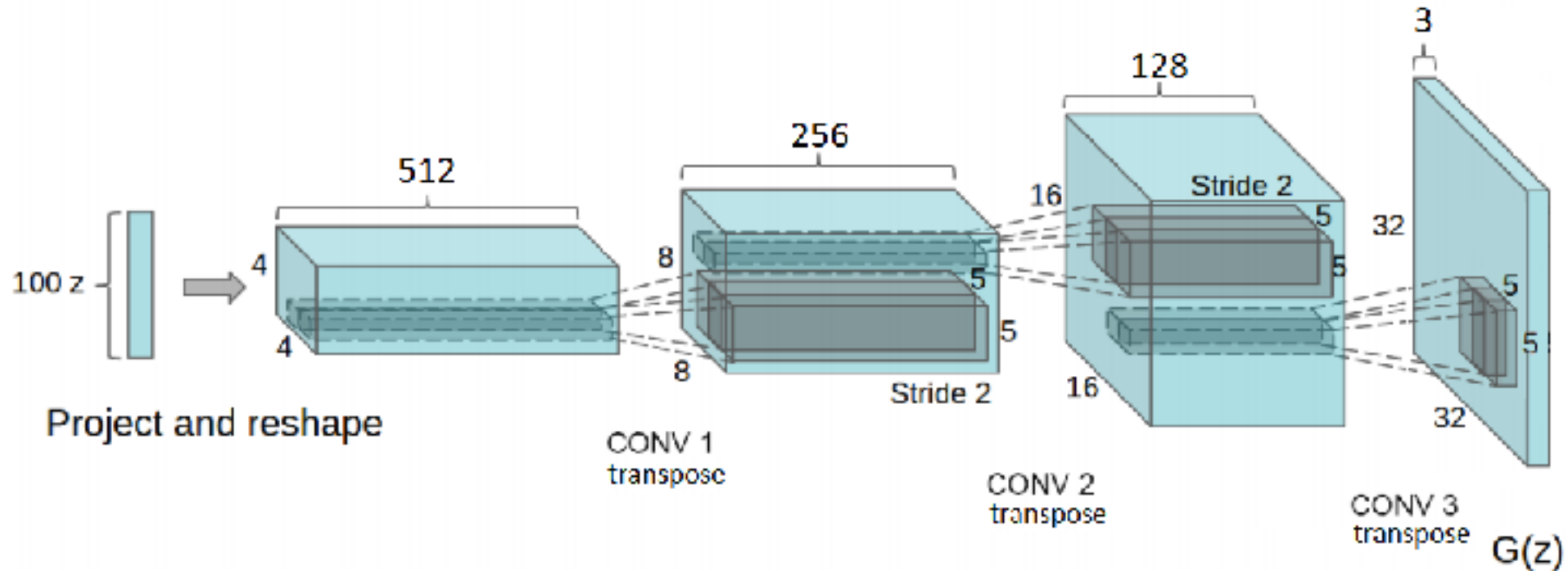
**Highlight:** Nearest Training Sample



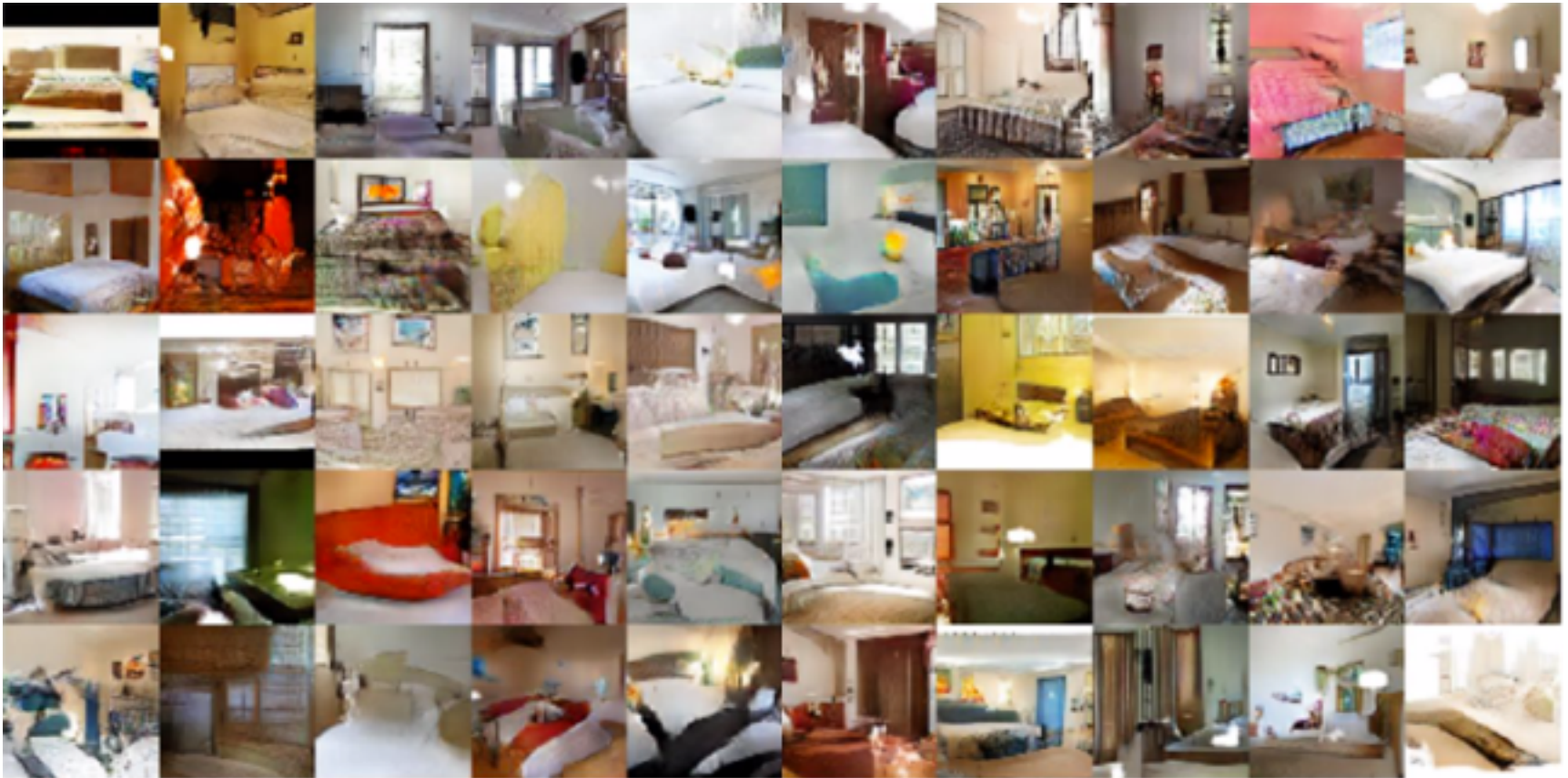


# Deep Convolutional GANs

- Just need to reshape and use upsampling



# Some Results of Generation



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

63





# GANs in PyTorch

Master Repository:

[07c GANsWithTorch.ipynb](#)

*wait we are gone use PyTorch?  
we need an intro to that...*



## GANs in Keras

Implementation from Book on  
“Frogs from CIFAR”



**Demo by Francois Chollet**

<https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.5-introduction-to-gans.ipynb>



# Basics of PyTorch

**When you're the only one of your friends  
who uses PyTorch instead of TensorFlow**



**Chip Huyen** @chipro · 17h

Sometimes I feel like R. Nobody's favorite but functional and pretty good with data.

9




23

237



# Wait, why are we switching to PyTorch?

- Well, its good to know more than just Tensorflow
- Pytorch has some distinct advantages:
  - No need to setup a static computation graph—graph can be dynamic (eager execution)
    - ♦ Lazy computations still happen on dynamic graph
  - Integration with numpy code on the fly is much easier and faster (compared to TF)
    - ♦ Can tradeoff computations with numpy easily, though not necessarily with autograd
- Also, all the RL book examples are in Pytorch... so this is nice for following along with the examples

	Keras 	TensorFlow 	PyTorch 
Level of API	high-level API <sup>1</sup>	Both high & low level APIs	Lower-level API <sup>2</sup>
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex <sup>3</sup>
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook <sup>4</sup>
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs <sup>5</sup>





# PyTorch General Flow Training Flow

- Inherit from `torch.nn.Module`
- Define `__init__` and `forward`
- Run epochs in a loop with explicit calls to:
  - Loss creation (for batch)
  - Backward calculation of gradient for batch
  - Step of optimizer for batch
  - Gives a great deal of flexibility to design and use optimization processes
- Lots of different pythonic ways to carry this out
  - Your RL book likes to setup steps of model through iterators (**yield** the batch, loss, etc.)



# A Simple Definition (much like Keras!)

```
1 import torch
2 import torch.nn as nn
3
4 class OurModule(nn.Module):
5     def __init__(self, num_inputs, num_classes, dropout_prob=0.3):
6         super(OurModule, self).__init__()
7         self.pipe = nn.Sequential(
8             nn.Linear(num_inputs, 5),
9             nn.ReLU(),
10            nn.Linear(5, 20),
11            nn.ReLU(),
12            nn.Linear(20, num_classes),
13            nn.Dropout(p=dropout_prob),
14            nn.Softmax(dim=1)
15        )
16
17    def forward(self, x):
18        return self.pipe(x)
19
20 if __name__ == "__main__":
21     net = OurModule(num_inputs=2, num_classes=3)
22     print(net)
23     v = torch.FloatTensor([[2, 3]])
24     out = net(v)
25     print(out)
26     print("Cuda's availability is %s" % torch.cuda.is_available())
27     if torch.cuda.is_available():
28         print("Data from cuda: %s" % out.to('cuda'))
```

**Sequential  
Definitions**

**Common Functions**





# The MNIST Example (not so like Keras)

## Functional Definitions

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

## Definitions

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size,
                 stride=1, padding=0, dilation=1, groups=1, bias=True)
```

```
view(*shape) → Tensor
```

reshape without copy

## Training One Epoch

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

negative log likelihood, cross entropy  
calculate gradients  
use gradients

## Training Multiple Epochs

```
model = Net().to("cpu")
optimizer = optim.SGD(model.parameters())

for epoch in range(1, args.epochs + 1):
    train(args, model, "cpu", train_loader, optimizer, epoch)
```

## Utils

```
from torchvision import datasets, transforms
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args.batch_size, shuffle=True, **kwargs)
```

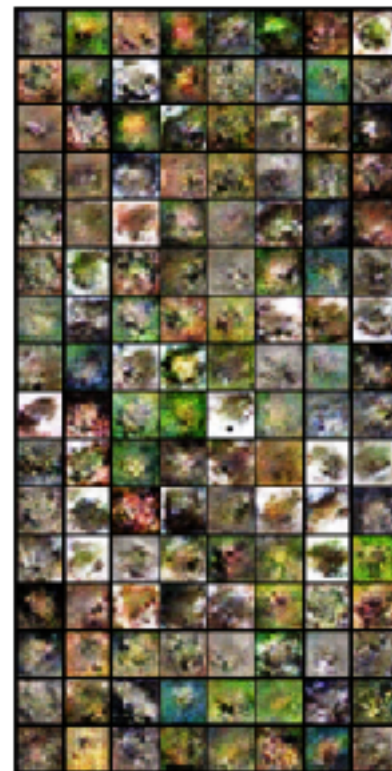
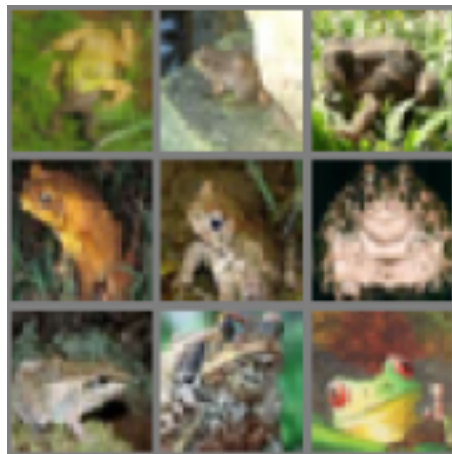




# GANs in PyTorch

Master Repository:

`07c GANsWithTorch.ipynb`



## GANs in Keras

Implementation from Book on  
“Frogs from CIFAR”



**Demo by Francois Chollet**

<https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.5-introduction-to-gans.ipynb>



# Paper Presentation

## UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

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

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

### ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.



# Lecture Notes for **Neural Networks and Machine Learning**

GANs

**Next Time:**  
Practical GANs and LSGAN  
**Reading:** Chollet CH8

