

# Understanding CNNs and Generative Adversarial Networks (GANs)

- \* Train a baseline model for CIFAR10 classification (~2 hours training time)
- \* Train a discriminator/generator pair on CIFAR10 dataset (~30 hours training time)
- \* Use techniques to create synthetic images maximizing class output scores or particular features as a visualization technique to understand how a CNN is working (<1 minute)

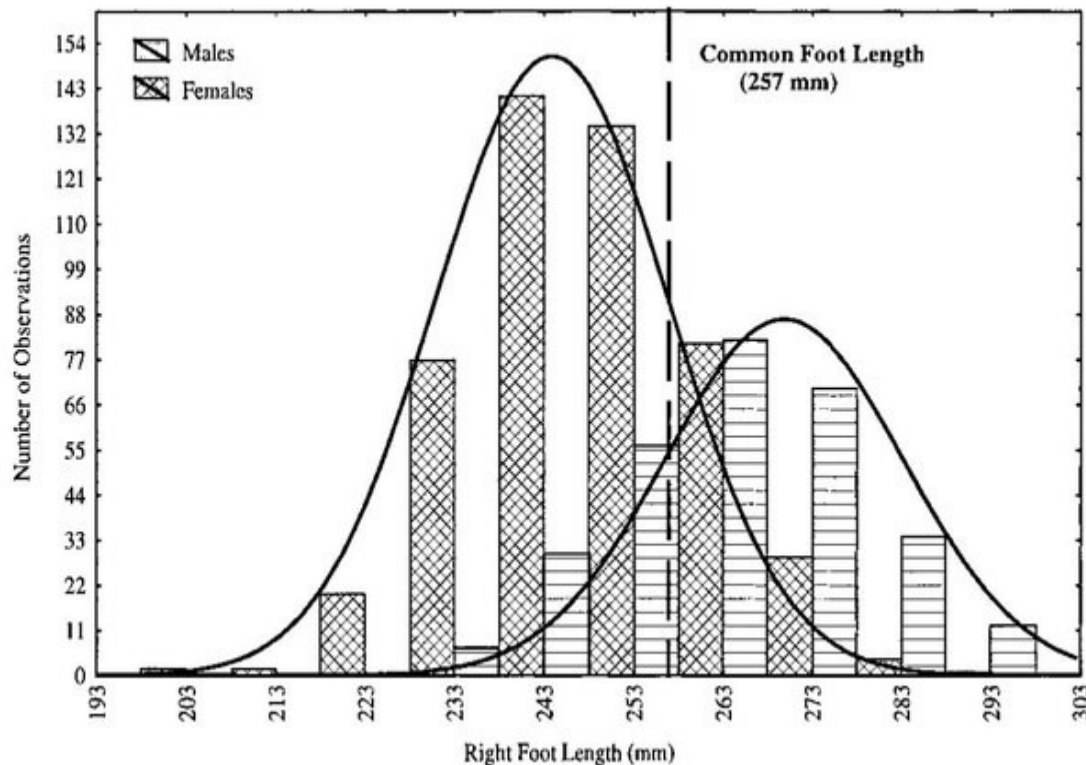
<https://courses.engr.illinois.edu/ie534/fa2018/secure/GAN.html>

Due next Friday (Oct 26)

# Generative vs Discriminative Methods

- Generative methods model the full joint distribution  $P(X,Y)$  (can get  $P(Y|X)$  and  $P(X|Y)$  from Bayes' Rule)
- Typically done by choosing an underlying distribution to model the data
- Neural networks are discriminative models (directly calculates pseudo-posterior probability  $P(Y|X)$ ). Capable of learning highly complex models with many modes
- For classification, we only want  $P(Y|X)$ .
- Can generate data by sampling from  $P(X|Y)$  (can't directly be done with a typical neural network)

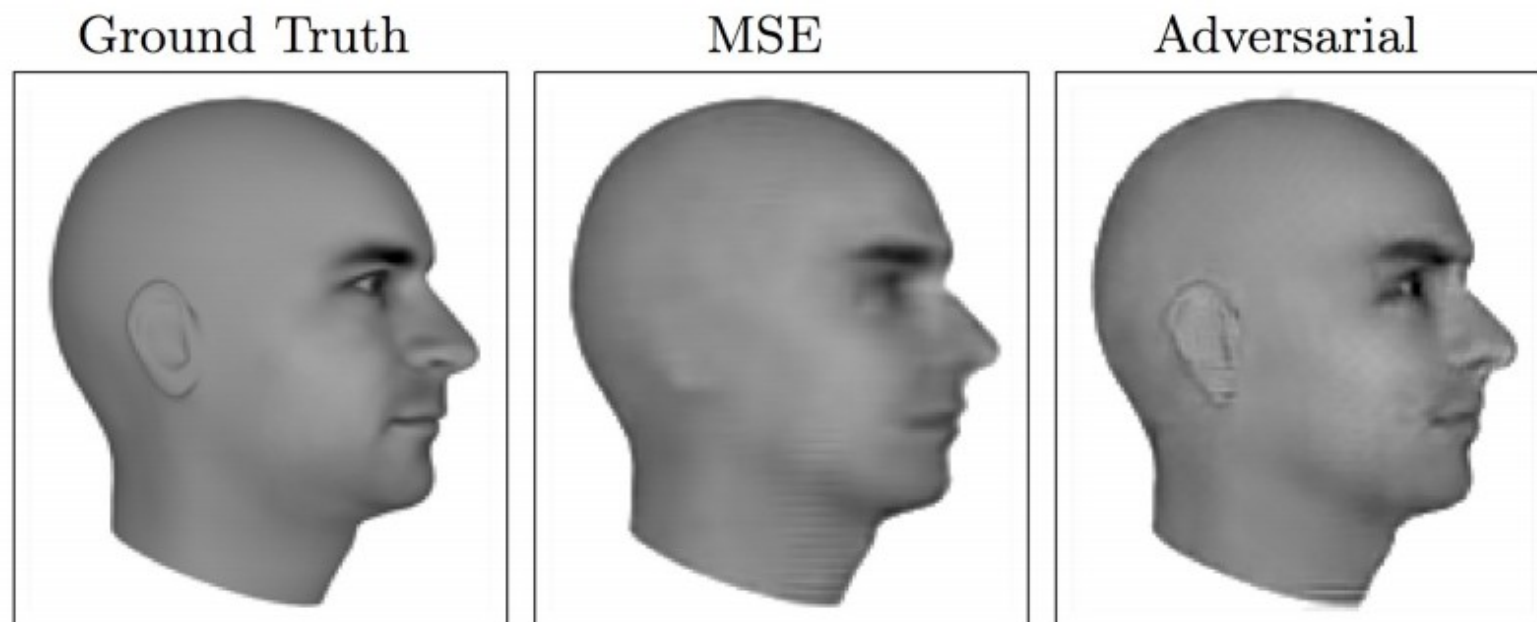
# Simple Example



- Shoe sizes for {male, female} are a multi modal distribution that can be modeled by independent Gaussians
- Given a random shoe size, it is easy to see how to determine most likely class {male, female}
- To generate fake data, simply sample from each of the two Gaussians
- As long as the underlying distribution chosen for modeling the data is correct, the sampled data is indistinguishable from the true data

# Motivation for Generative Models

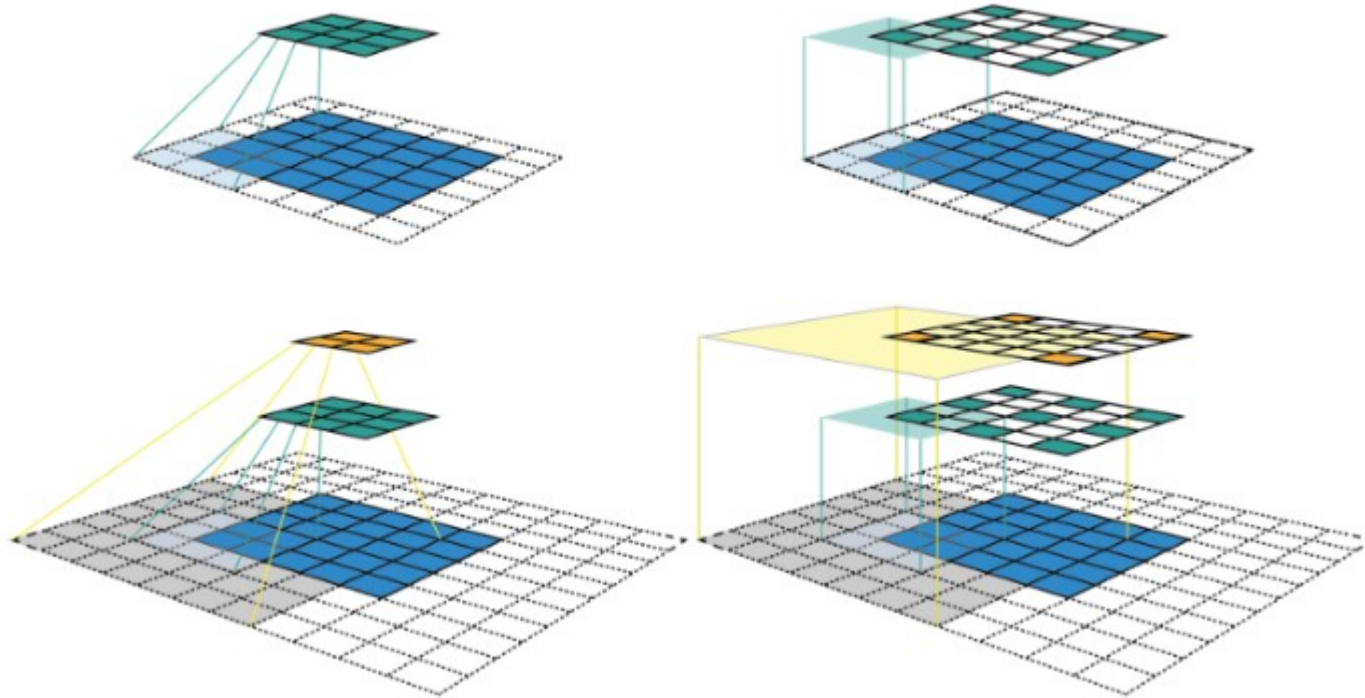
- Represent and manipulate high dimensional probability distributions
- Reinforcement learning applications where the model generates “realistic” experiences to learn from (acquiring data may be costly)
- Unsupervised/Semi-supervised learning (acquiring unlabeled data is typically easier)
- Capable of handling multi-modal models



# Understanding CNNs (What does a CNN learn?)

- Does it learn small or large parts? Collection of small parts? Concept of “whole”? (Think about how you would describe an object to another person, would the inner workings of a CNN be capable of providing a similar description?)
- Lots of questions about how many layers/hidden units/pooling/etc. - no definitive answer. SGD allows a network to learn without user input, but it must be capable of learning the function in the first place
- A network is trained only to “discriminate” between objects (tiger vs. soccerball? vs. basketball? vs. zebra?)
- Layers of convolution start simple and become more complex
- “Receptive Field” gets bigger as more layers are used

# Receptive Field



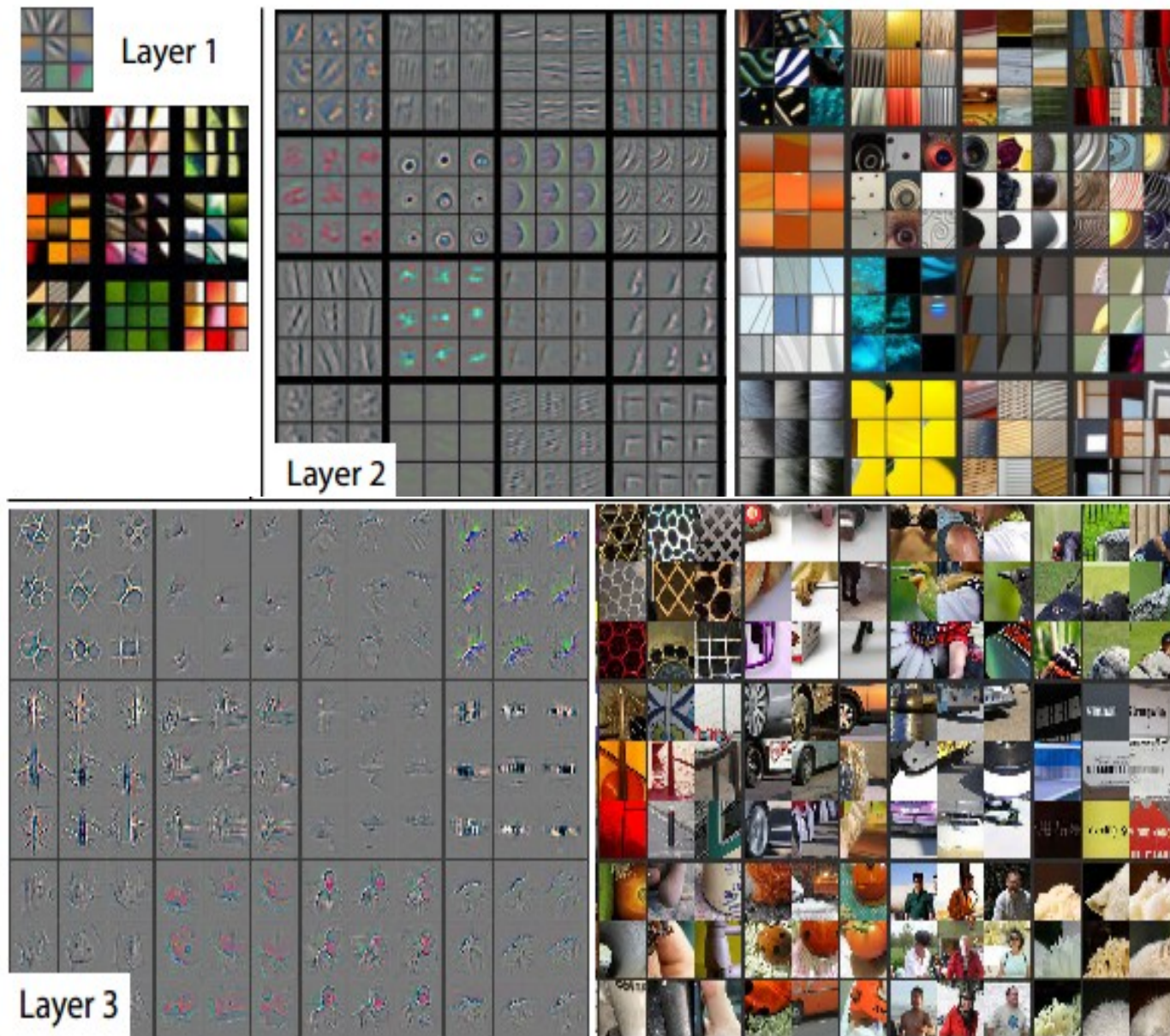
- Region of the input image that went into the calculation of a particular feature
- Convolutions and pooling layers increase the receptive field
- Near the output of the network, the receptive field should be relative to the “size” of the objects of interest within the image

<https://fomoro.com/tools/receptive-fields/#3,1,1,SAME;3,2,1,SAME;3,1,1,SAME;3,2,1,SAME;3,1,1,SAME;3,1,1,SAME;3,1,1,SAME;3,2,1,SAME>

<https://medium.com/@Synced/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-42f33d4378e0>

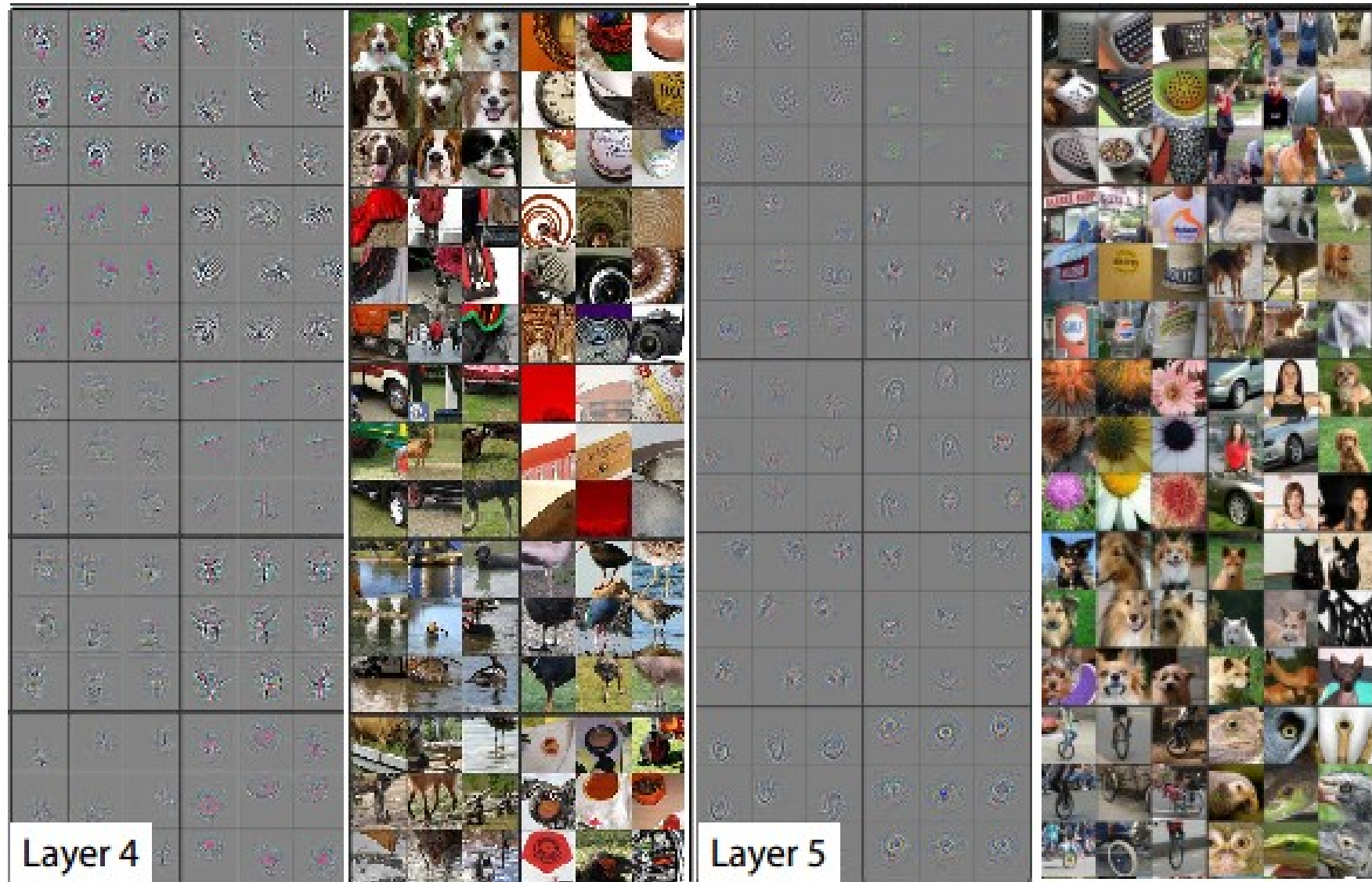


# What Do Features Look Like?



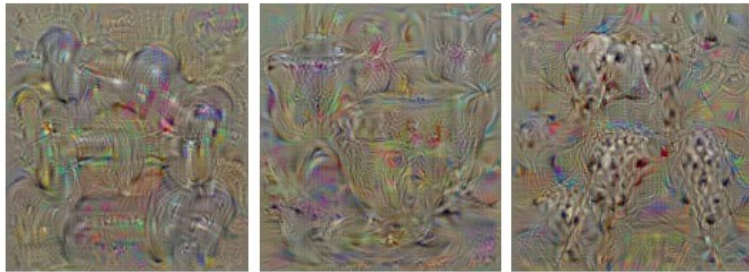
- Features become more interesting and complex as network deepens
- Need lots of training data to “see” all possible types
- Overtraining – particular feature very apparent in dataset yet not necessarily representative of underlying distribution
- Detecting tanks (not sure if real story but demonstrates point well) – it technically “learned” but the data distribution did not match the true targeted distribution

# What Do Features Look Like?





# Synthetic Images with Large Class Scores



dumbbell



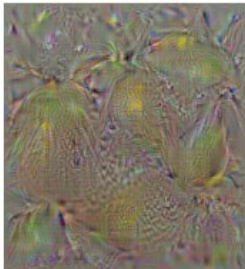
cup



dalmatian



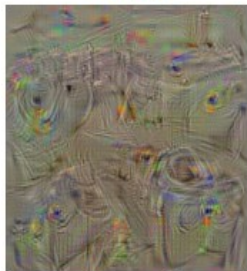
bell pepper



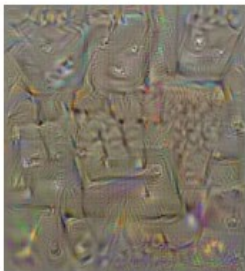
lemon



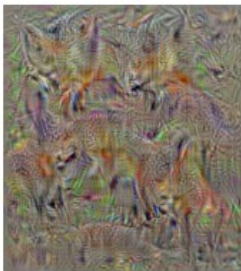
husky



washing machine



computer keyboard



kit fox



goose



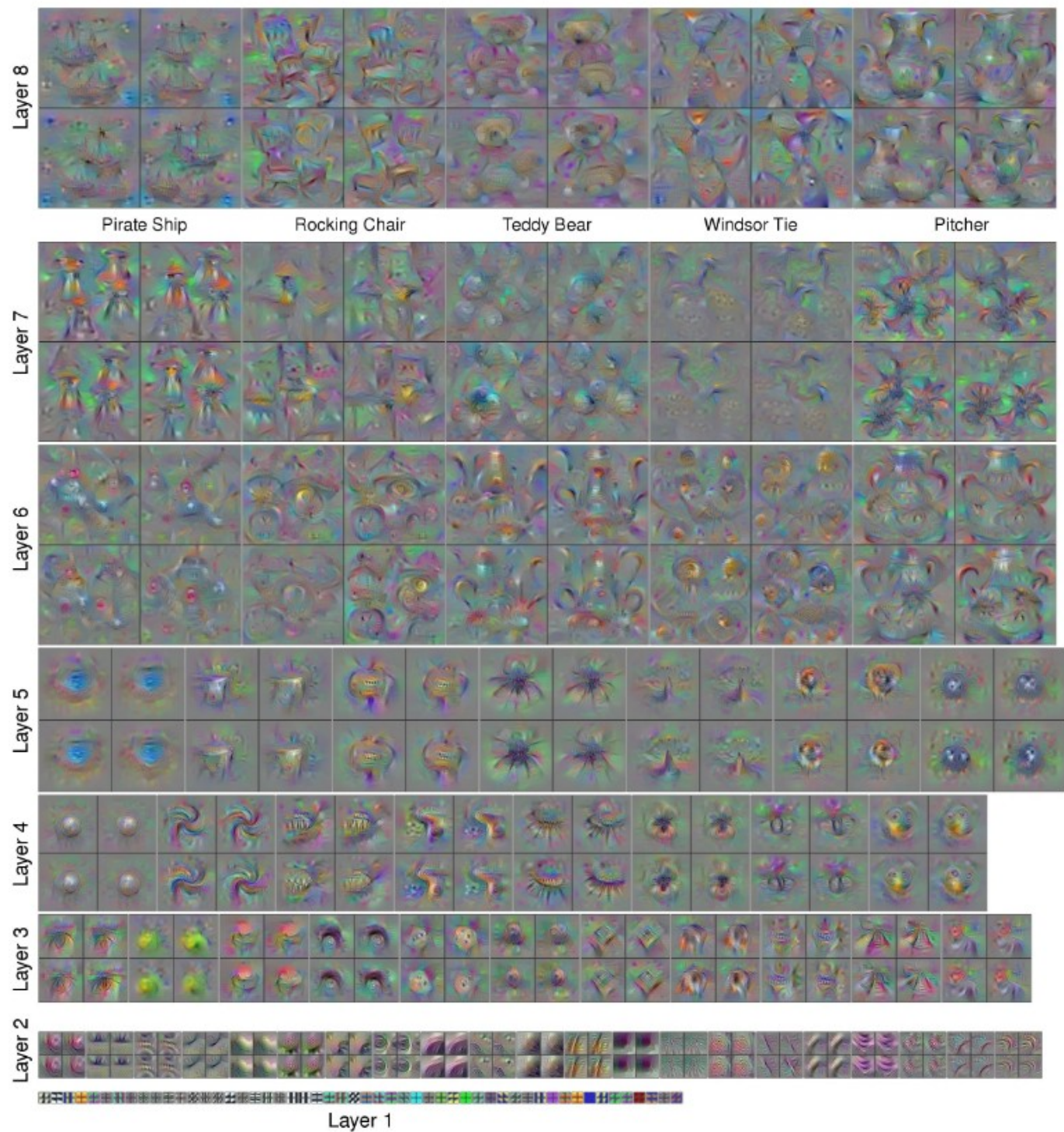
ostrich



limousine

- Backpropagate error to noisy initialized image until network outputs large value for desired class
- Lots of random points/colors/contours that cause high activation but not what we would think is important
- Images don't look real but distinctive features – multiple objects, some patterns (dalmatians), edges (computer keyboard), object size (limousine)
- 

Deep Inside Convolutional Networks: Visualizing Image Classification Models and Saliency Maps - <https://arxiv.org/pdf/1312.6034.pdf>



Layer 1

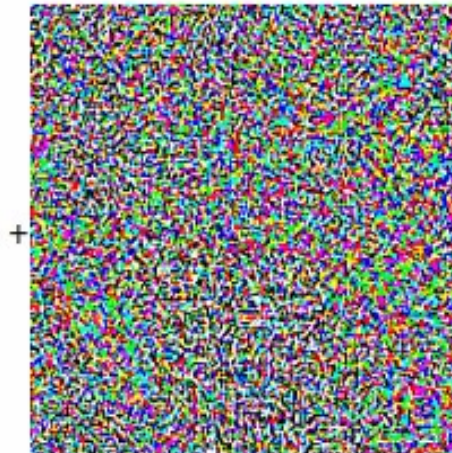


# Tricking Networks

- 8-bit images (0-255) rescaled between -1.0 and 1.0 (resolution of 0.0078)
- Backpropagate error from alternative class to real image, use sign of the gradient (-1 or 1) and multiply by 0.0078



Original image classified as a panda with 60% confidence.



Tiny adversarial perturbation.



Imperceptibly modified image, classified as a gibbon with 99% confidence.

# Tricking Networks

- Models behave “too linearly”, extrapolate far from the training data and become overconfident

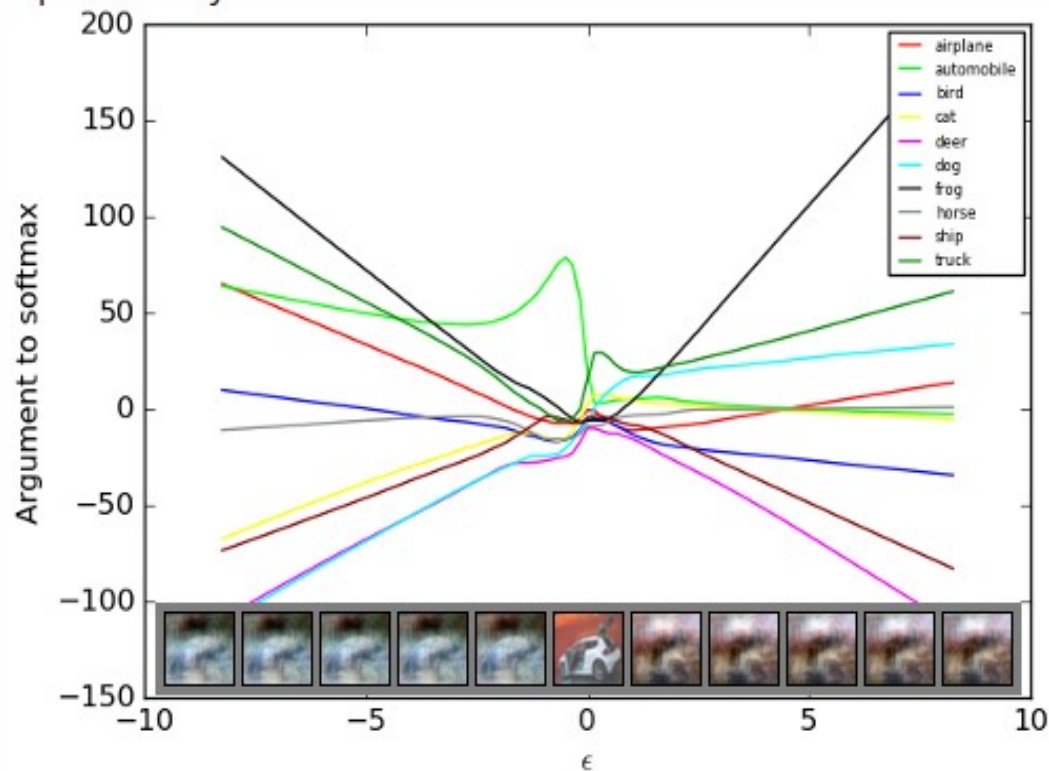


Fig 2. We can trace out a linear path in input space by adding an adversarial perturbation scaled by differing amounts to a clean image of a car. Here we follow the linear path from a scaling factor of negative 10 to positive 10. We see that the logits output by the network behave linearly far from the data. This causes the network's predictions to become extreme, resulting in rubbish class inputs being classified as real classes with high confidence.

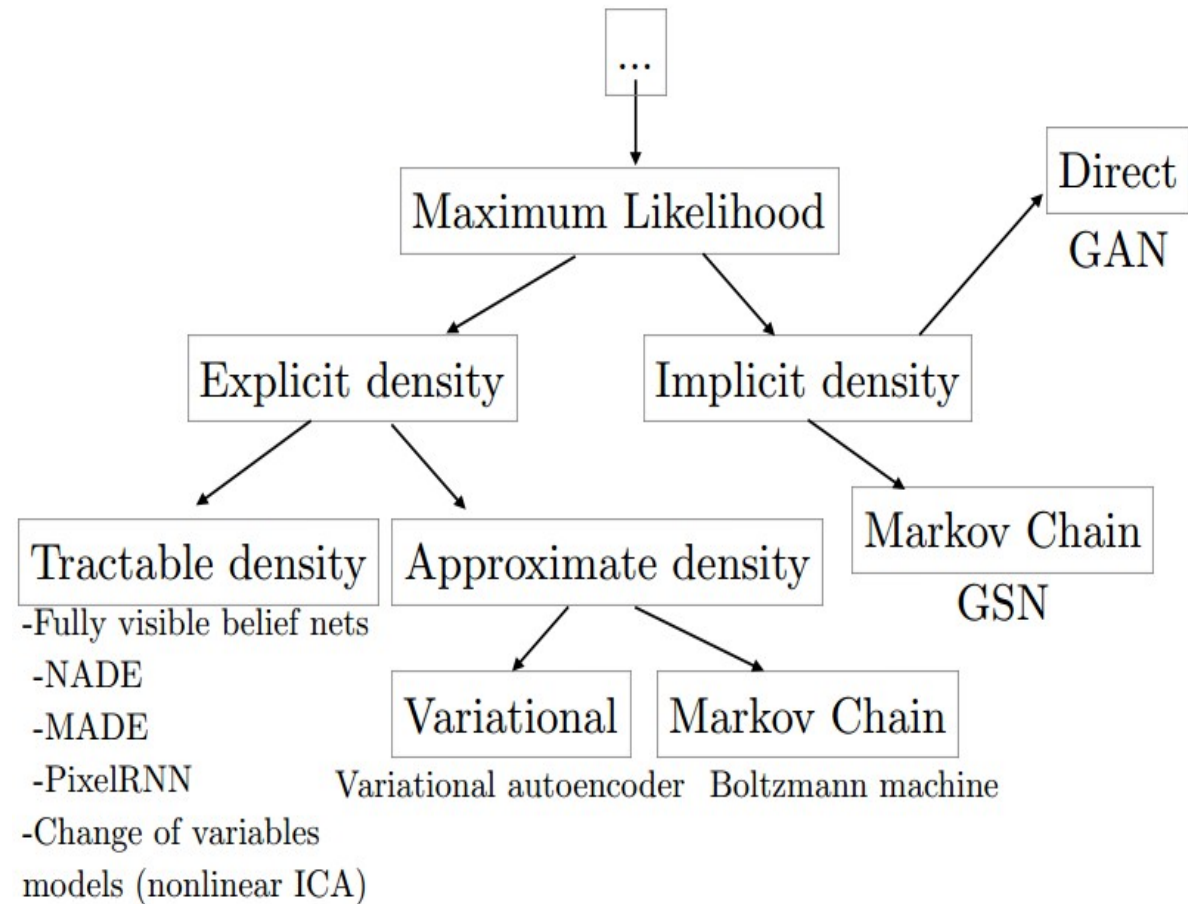
# Generative Adversarial Networks (GANs)

NIPS 2016 Tutorial: Generative Adversarial Networks -  
<https://arxiv.org/pdf/1701.00160.pdf>

- Lecture slides 33-56 of lecture 10 cover most of the math behind GANs and Wasserstein GANs. I will additionally go over the extras for what you will implement in the homework
- Read the tutorial above before, far more comprehensive

# GANs

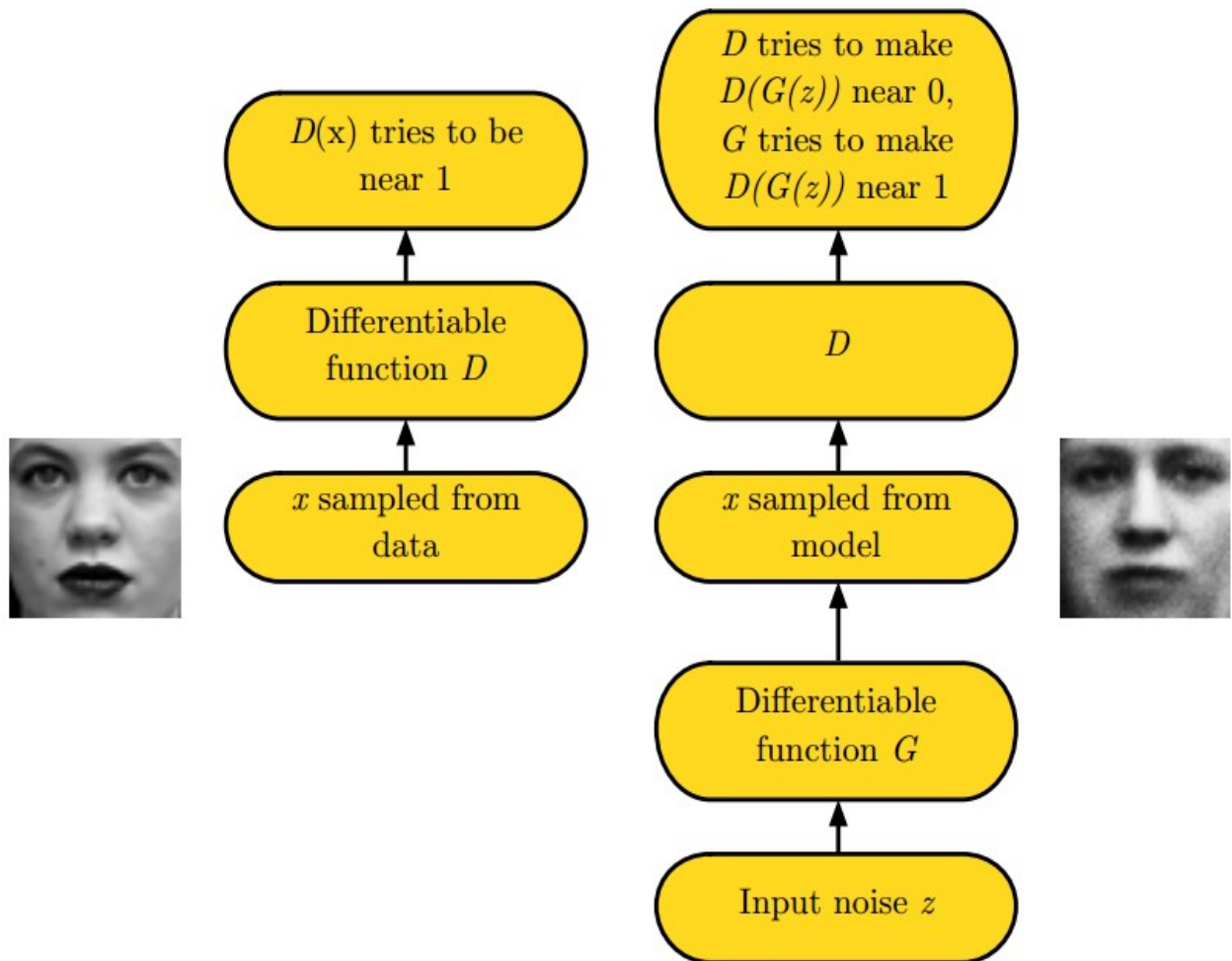
- Lots of generative models have been studied in the past
- GANs are very flexible: as long as you can back propagate some error, no need to explicitly model distributions
- Originally very difficult to train, Wasserstein GANs with a few other recent improvements seem to help





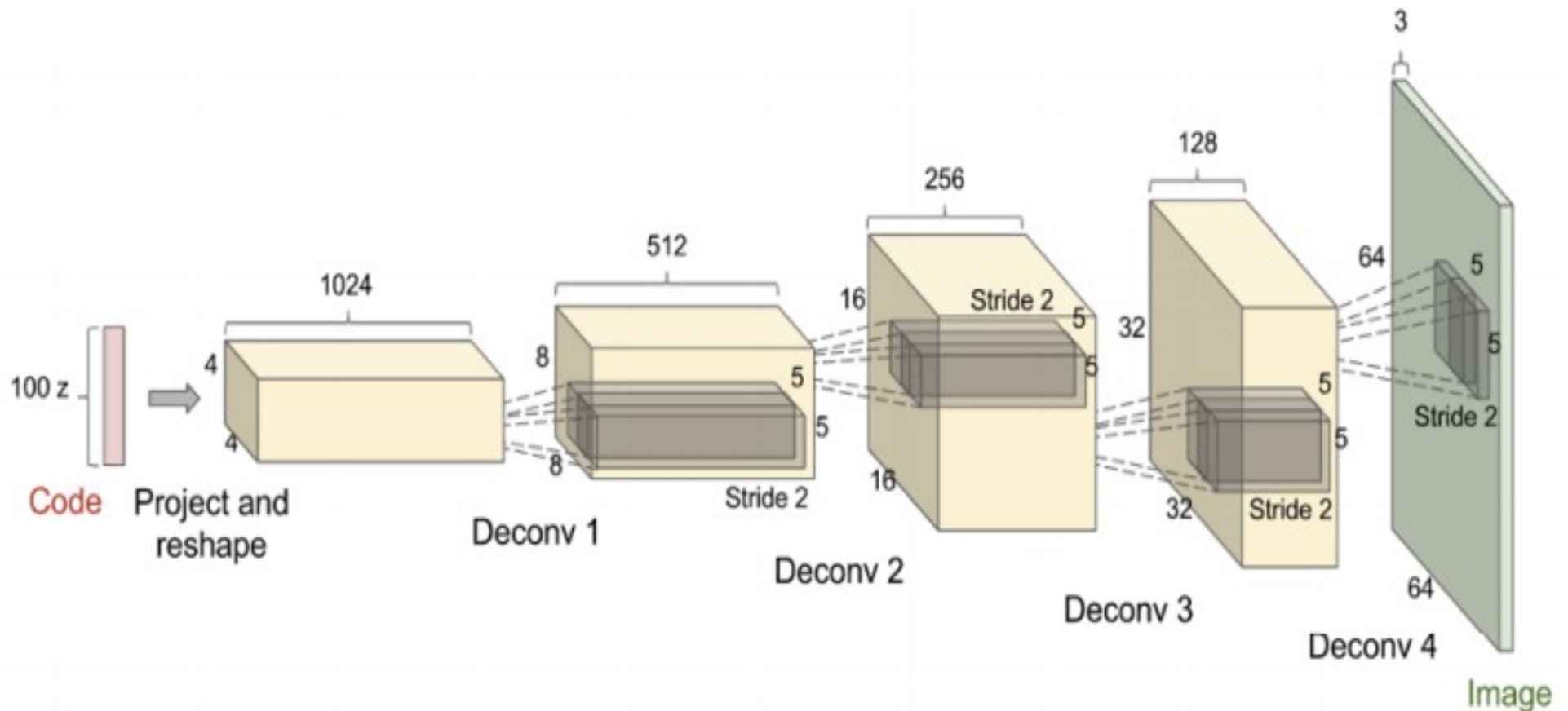
# GANs Setup

- Consists of a generator network and a discriminator network
- Generator will sample a vector from a random distribution and manipulate it with transposed convolutions into a fake image
- Discriminator will be a basic convolutional network
- Doesn't require labeled data (although labeled data always helps)



# Generator

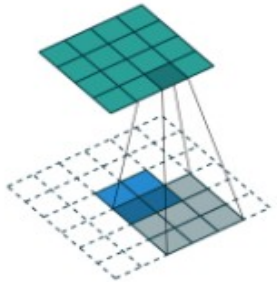
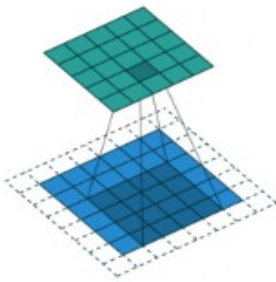
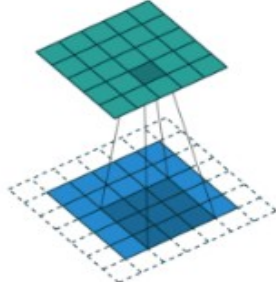
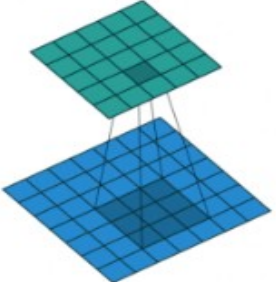
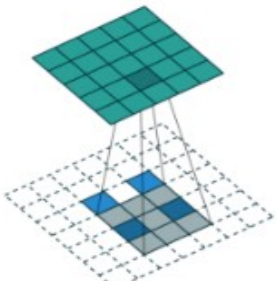
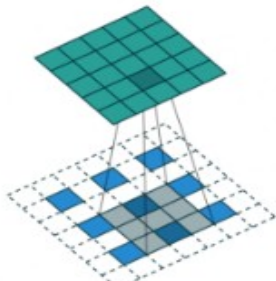
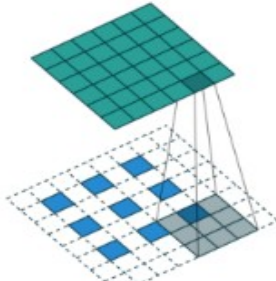
- Transposed convolutions are sometimes referred to as deconvolutions (although deconvolution is not the correct term)
- Manipulates a low dimensional random distribution into the shape of an image



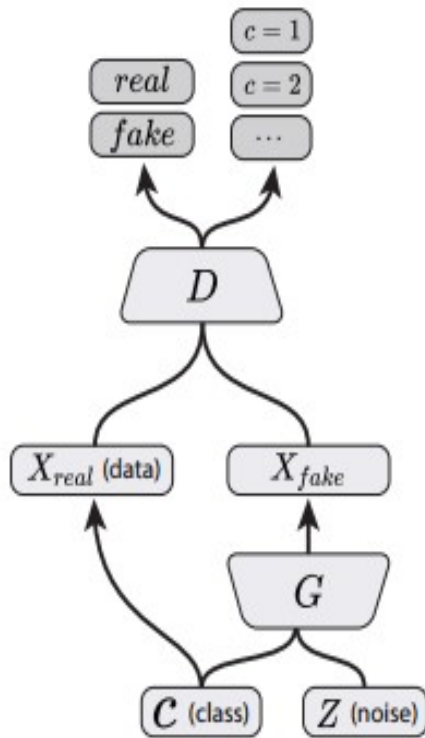
# Transposed Convolutions

- Output image increases in size when stride>1
- There is a built in PyTorch module

*N.B.: Blue maps are inputs, and cyan maps are outputs.*

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

# Conditional Image Synthesis with Auxiliary Classifier GANs (ACGAN)



AC-GAN  
(Present Work)

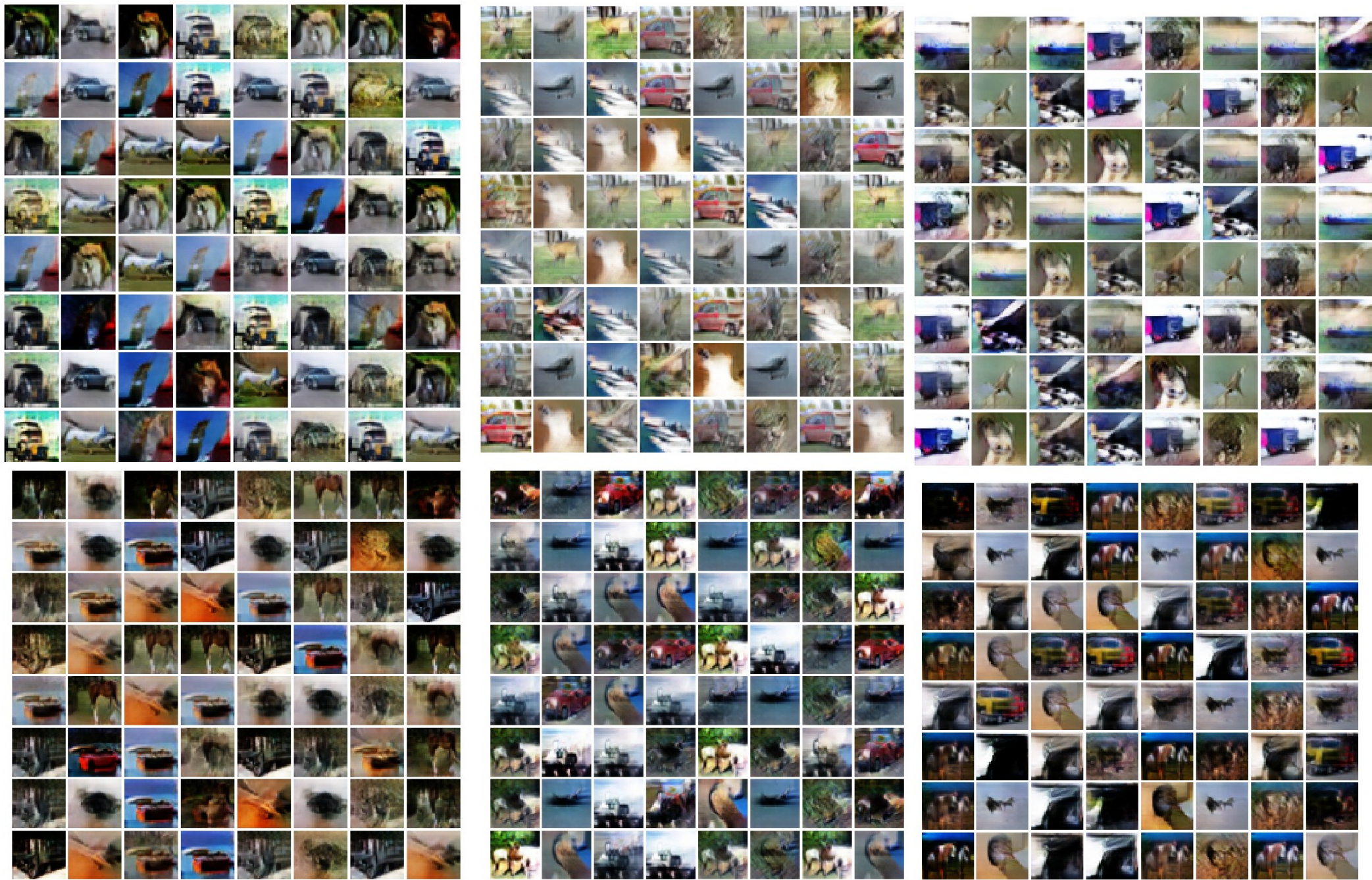
- Two primary distinctions
- 1 – generator is provided with some information about class label. Uses first  $n$  dimensions of  $d$  dimensional random vector for a 1-hot representation
- 2 – Two output layers in the discriminator. The first is for real/fake. The second is the auxiliary classifier for class label
- Allows the generator to generate class specific samples

# Wasserstein GANs

- GANs were very unstable to train and frequently experienced mode collapse. No successful training of large models and very sensitive to hyperparameters/architecture
- Wasserstein GAN optimizes a different loss function that doesn't vanish to 0 if the models become unbalanced
- Implementation is easy. Remove the sigmoid operation from the final layer and introduce weight clipping to satisfy the Lipschitz constraint.
- Discriminator:  $\text{loss} = \text{output\_fake} - \text{output\_real}$
- Generator:  $\text{loss} = -\text{output\_fake}$
- Discriminator wants large values when the image is real and small values when the image is fake. Generator wants to generate fake samples with a high discriminator output



# Mode Collapse





# Improved Training of Wasserstein GANs (2017)

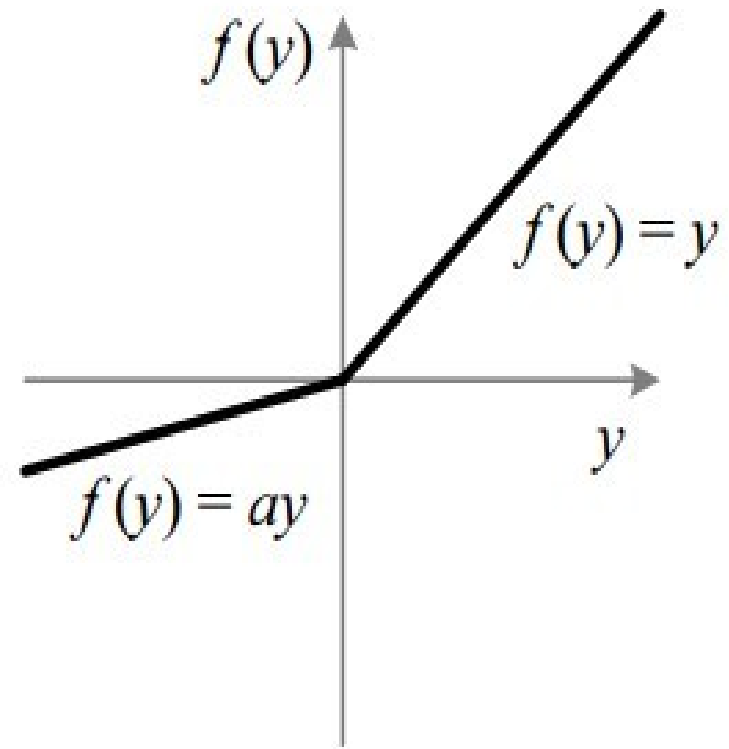
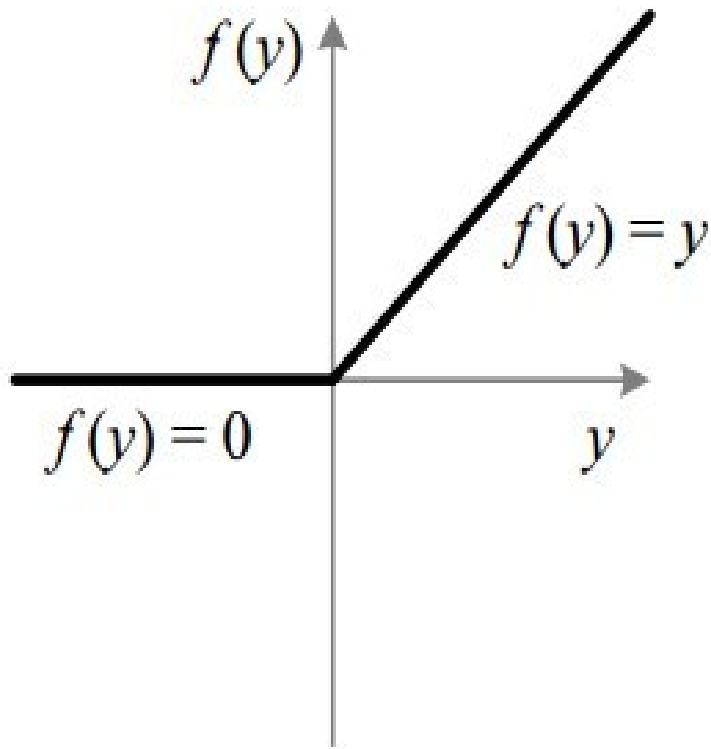
- Found that weight clipping to enforce the Lipschitz constraint still led to undesired behavior. They penalize the norm of the gradient of the critic (discriminator) with respect to its input to enforce the constraint instead
- Significantly easier to train. ResNet101 generator and discriminator models. Far more robust in terms of hyperparameters
- Doesn't work with Batch Normalization. Uses layer normalization instead

# Layer Normalization

- Layer normalization attempted to address certain issues related to applying batch normalization to recurrent neural networks and to remove batch dependencies between the input and the output
- This batch dependency makes the gradient penalty invalid
- Let  $X$  be a data matrix of size (b by d).  $b$ =batch\_size,  $d$ =num\_of\_features
- Batch normalization normalizes with respect to the columns while layer normalization normalizes with respect to the row
- The features of a single sample are normalized based on all of the other features in that sample without any information about other samples in the batch
- Output from layer normalization is the same during training and testing

# Leaky ReLU

- Typically easier to train GANs in practice with the leaky relu. It still allows gradient flow even if the output is negative



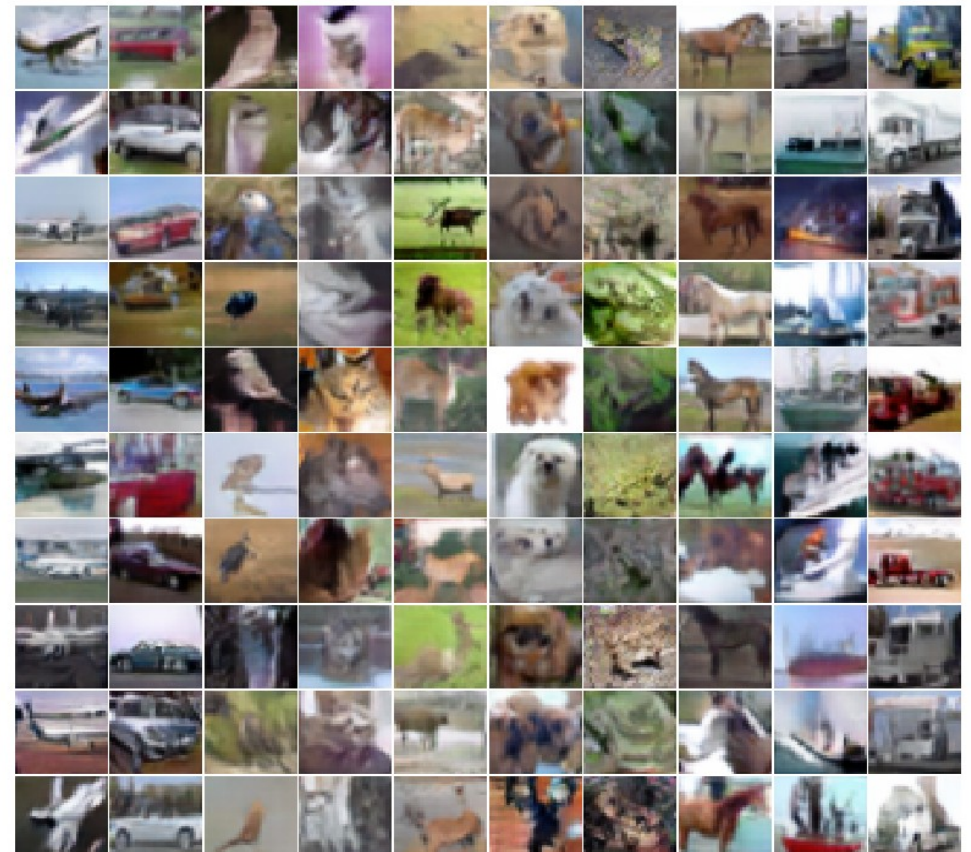
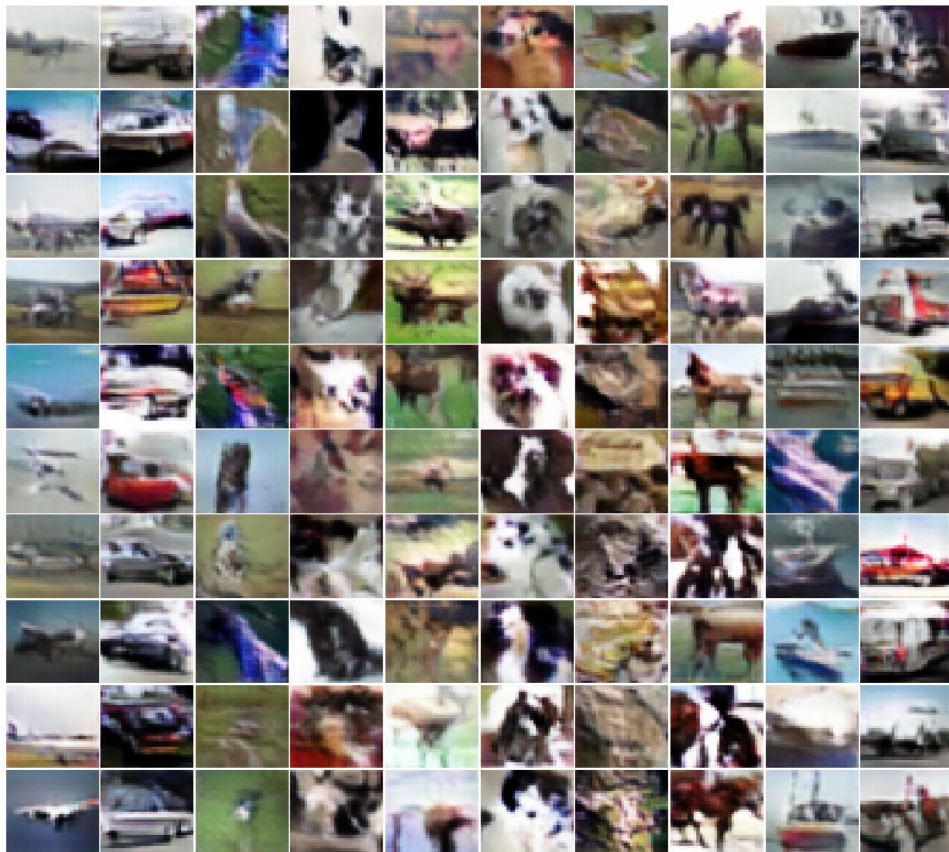
# Assignment – Part 1 Training a GAN

name	type	input_channels	output_channels	ksize	padding	stride	input	output
conv1	conv2d	3	196	3	1	1	32x32	32x32
conv2	conv2d	196	196	3	1	2	32x32	16x16
conv3	conv2d	196	196	3	1	1	16x16	16x16
conv4	conv2d	196	196	3	1	2	16x16	8x8
conv5	conv2d	196	196	3	1	1	8x8	8x8
conv6	conv2d	196	196	3	1	1	8x8	8x8
conv7	conv2d	196	196	3	1	1	8x8	8x8
conv8	conv2d	196	196	3	1	2	8x8	4x4
pool	max2d			4	0	4	4x4	1x1
fc1	linear	196	1					
fc10	linear	196	10					

name	type	input_channels	output_channels	ksize	padding	stride	input	output
fc1	linear	100 (noise)	196*4*4					
conv1	convtranspose2d	196	196	4	0	2	4x4	8x8
conv2	conv2d	196	196	3	1	1	8x8	8x8
conv3	conv2d	196	196	3	1	1	8x8	8x8
conv4	conv2d	196	196	3	1	1	8x8	8x8
conv5	convtranspose2d	196	196	4	0	2	8x8	16x16
conv6	conv2d	196	196	3	1	1	16x16	16x16
conv7	convtranspose2d	196	196	4	0	2	16x16	32x32
conv8	conv2d	196	3	3	1	1	32x32	32x32

# Assignment – Part 1

- Will train one discriminator without the generator and another with the generator
- The assignment on the website steps you through the process of how to do this and incorporate everything from the slides (ACGAN, Wasserstein GAN, Gradient Penalty)





# Assignment - Part 2 Feature Visualization

- Will recreate the slides from the beginning on generating synthetic images maximizing activations in order to visualize what the network learns
- Compare learned features between the two discriminator trained models





# Takeaway

- Help build your intuition of CNNs to better design networks based on your data as opposed to treating it as a black box
- Help realize various problems and creative solutions neural networks can tackle as opposed to being limited by a simple classification structure
- Please ask questions and let me know if anything is unclear

# GAN Applications

- Text-to-image synthesis

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



this magnificent fellow is almost all black with a red crest, and white cheek patch.



the flower has petals that are bright pinkish purple with white stigma



this white and yellow flower have thin white petals and a round yellow stamen



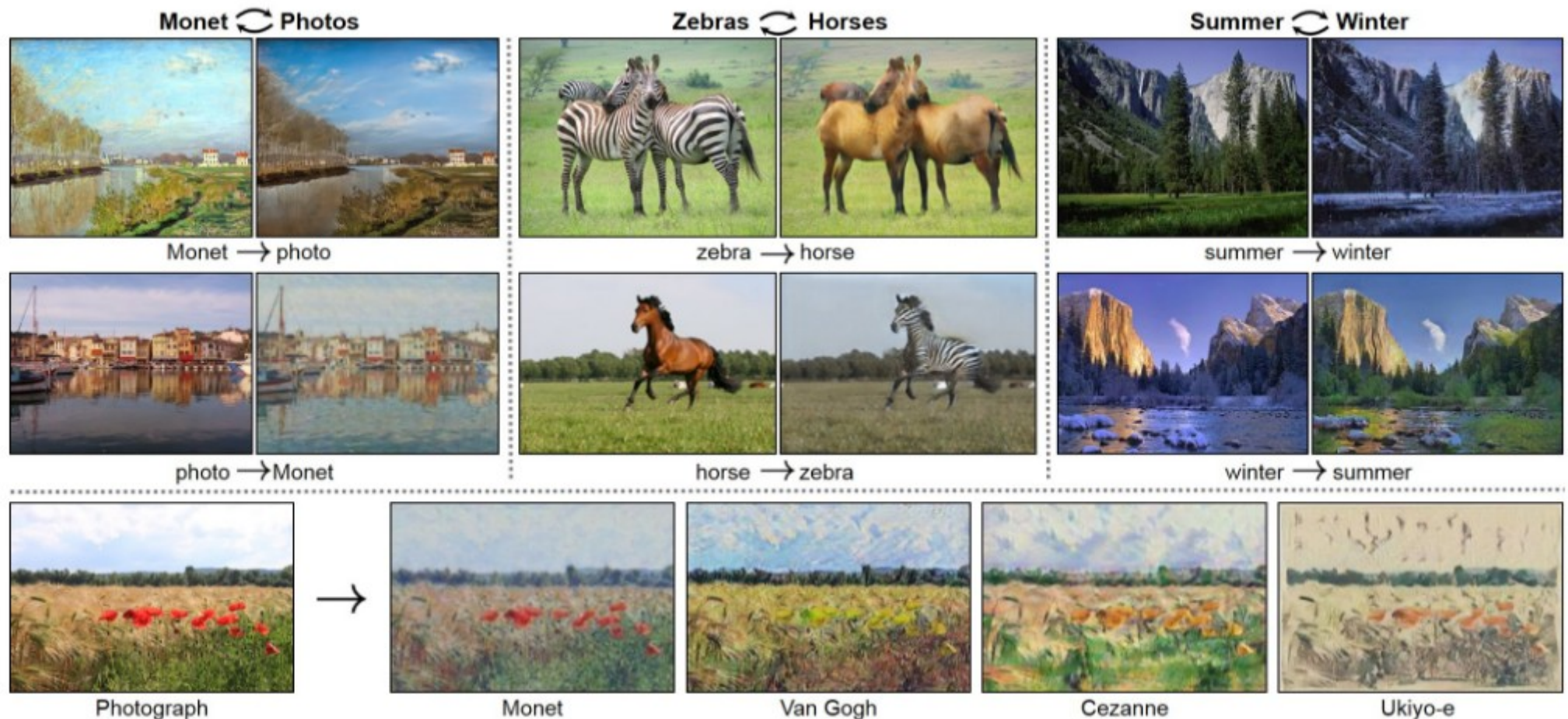
# Examples – More image-to-image translation





# GAN Applications

- Art (style transfer) and Image-to-Image Translation



Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks -  
<https://junyanz.github.io/CycleGAN/>

# Progressively Growing GANs

[https://research.nvidia.com/publication/2017-10\\_Progressive-Growing-of](https://research.nvidia.com/publication/2017-10_Progressive-Growing-of)

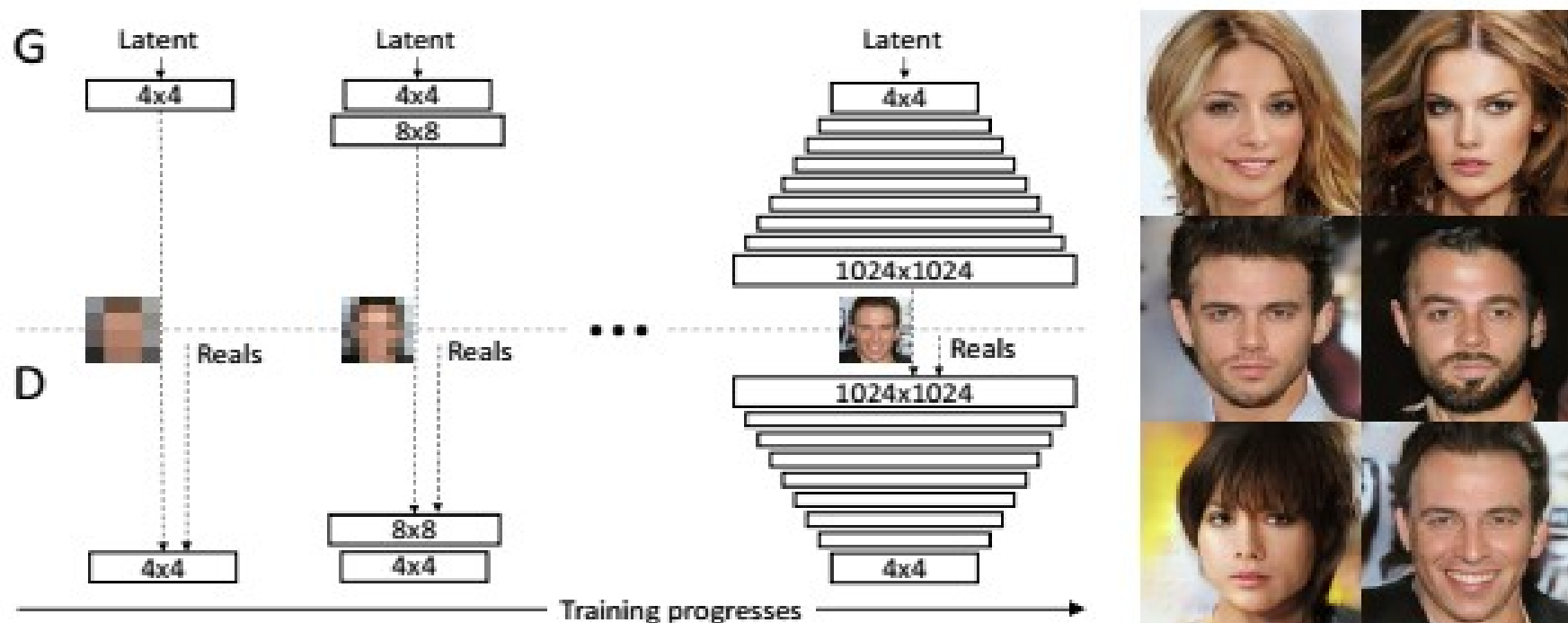
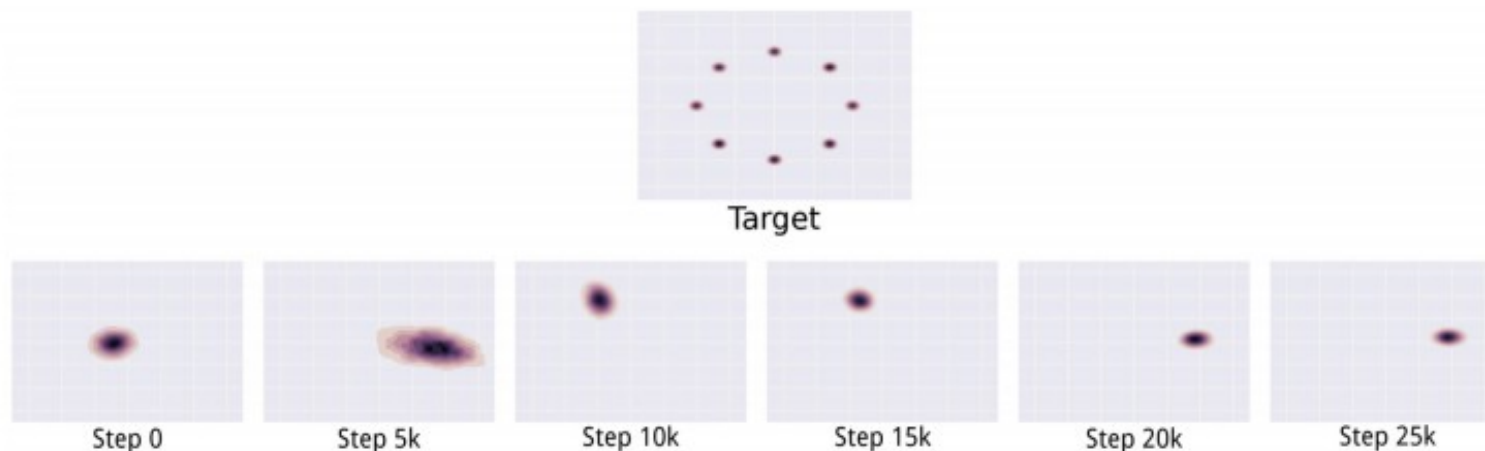


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of  $4 \times 4$  pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here  $N \times N$  refers to convolutional layers operating on  $N \times N$  spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. On the right we show six example images generated using progressive growing at  $1024 \times 1024$ .

# Problems with GAN

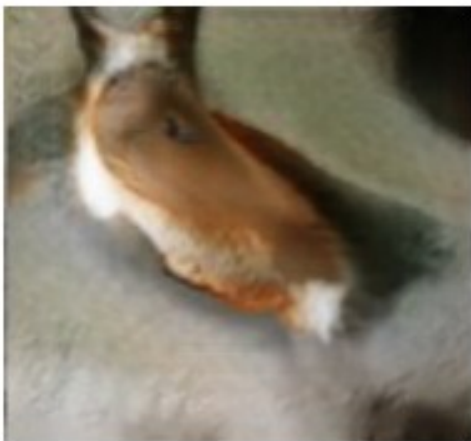
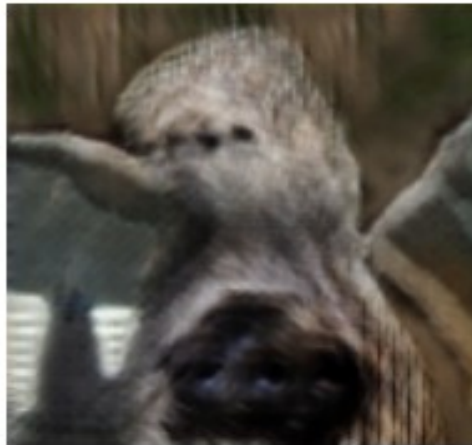
- Non-convergence: Requires finding an equilibrium between two players (G and D) in a game. Sometimes can repeatedly undo progress (D focuses too much on G without learning worthwhile features, D becomes too good at recognizing fake images and G gets stuck)
- Mode Collapse – several different input values  $\mathbf{z}$  mapped to the same output, the generator then oscillates between modes without further progress





# Examples - Minibatch GANs

- Method includes features based on the distance between samples in a mini-batch to encourage them to be different



# Examples – Trouble with Counting

- Discriminator can output a very high score because of the small part (face) without considering the object as a whole



# Examples – Face Generation



<https://github.com/carpedm20/DCGAN-tensorflow>