CS 466/566
Introduction to Deep Learning

Lecture 5 – Generative Neural Networks

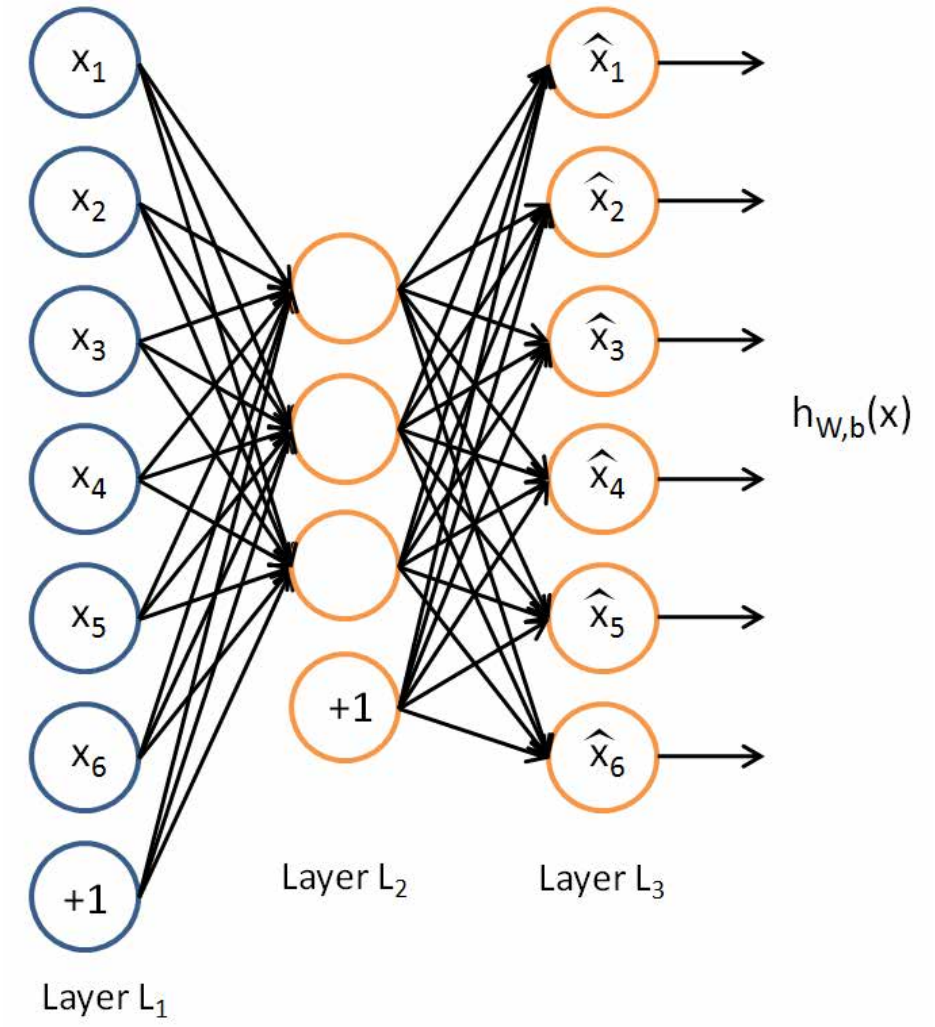
Compiled from various resources

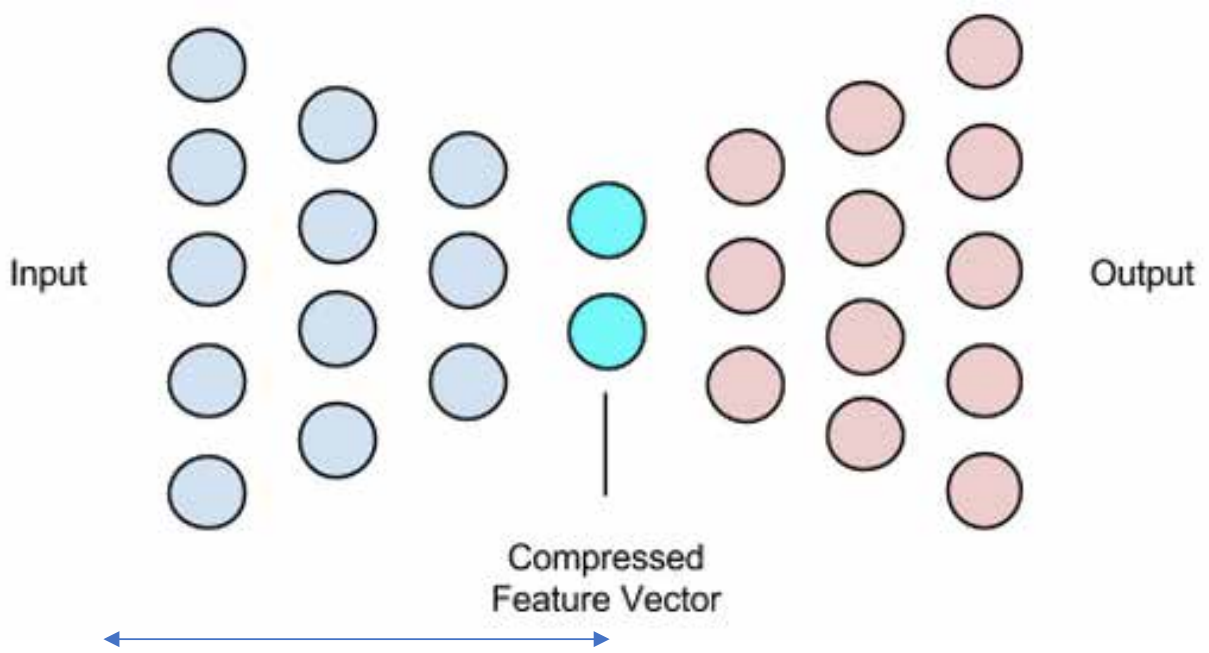
Let’s start with a teaser question first

Color Constancy:
This picture has no red pixels.

Why do we see red in this picture?

Recall: Auto-encoders / Deep Auto-encoders

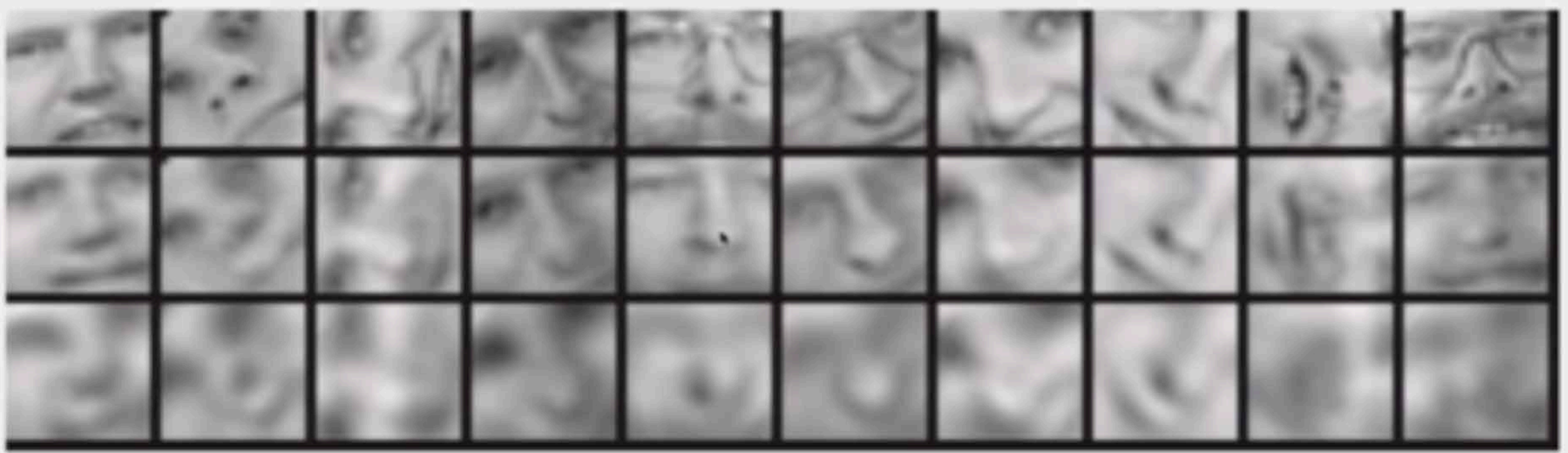




Representation learning

Deep Auto-encoders
25x25 – 2000 – 1000 – 500 – 30 auto-encoder

- 30D feature extractor for Olivetti face patches
  - Top: random samples from data set
  - Middle: reconstructions by the deep auto-encoder
  - Bottom: reconstructions by 30 dimensional PCA



(Stacked) Denoising Auto-encoders

- An auto-encoder is a neural network used for dimensionality reduction; that is, for feature selection and extraction.
  - But auto-encoders with more hidden layers than inputs run the risk of learning the identity function – where the output simply equals the input – thereby becoming useless.
- Denoising auto-encoders are an extension of the basic auto-encoder, and represent a stochastic version of it.

(Stacked) Denoising Auto-encoders

- Denoising auto-encoders attempt to address identity-function risk by randomly corrupting input (i.e. introducing noise) that the auto-encoder must then reconstruct, or denoise.

- Parameters and Corruption level
  - The amount of noise to apply to the input takes the form of a percentage. Typically, 30% is fine
  - if you have very little data, you may want to consider adding more corruption.

“The Dress”

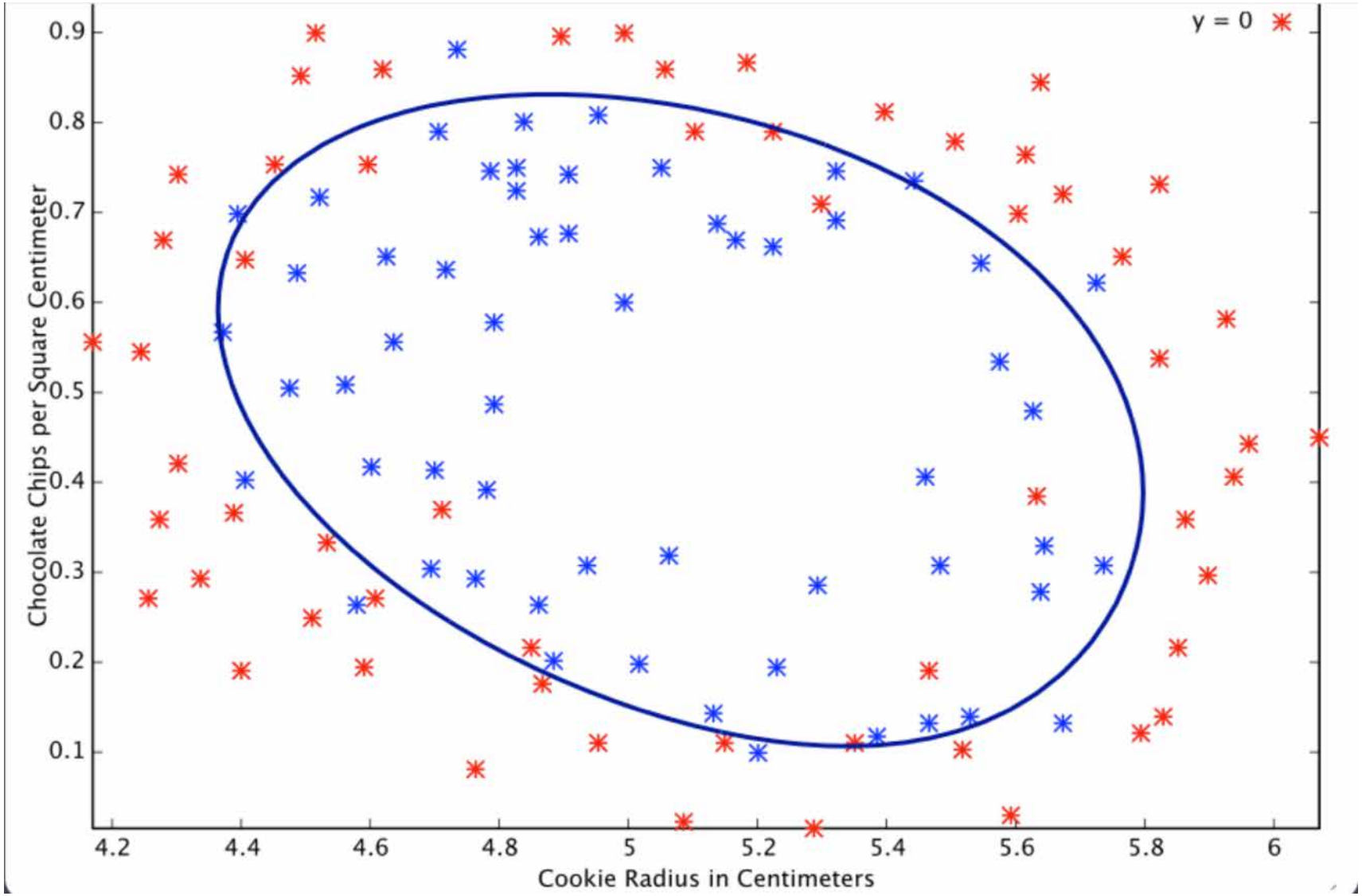
- "**The dress**" is a photo that became a viral Internet picture on 26 February 2015, when viewers disagreed over whether the item of clothing depicted was *“black and blue”* or *“white and gold”*.
- The phenomenon revealed differences in human color perception which have been the subject of ongoing scientific investigation in neuroscience and vision science, with a number of papers published in peer-reviewed science journals.

Recall: Discriminative vs Generative Models

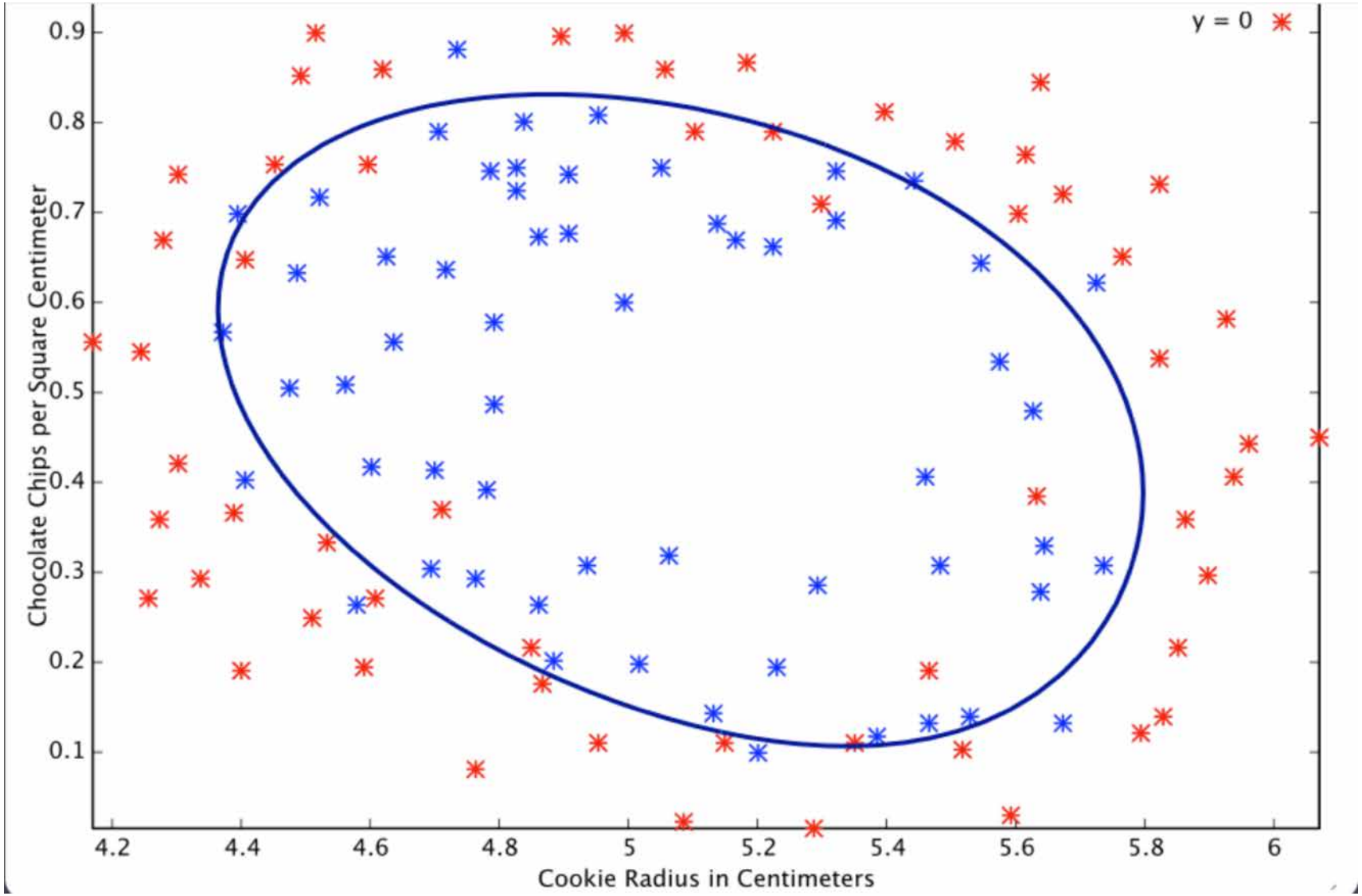
- Discriminative Models
  - **Discriminative models** learn the (hard or soft) **boundary** between classes

- Generative Models
  - providing **a model of how the data is actually generated.**
  - model the **distribution** of individual classes
  - often outperforms discriminative models on smaller datasets because their **generative assumptions place some structure on your model that prevent overfitting.**

Recall: Cookie Example was a discriminative model



How would you represent this with a Generative Model?



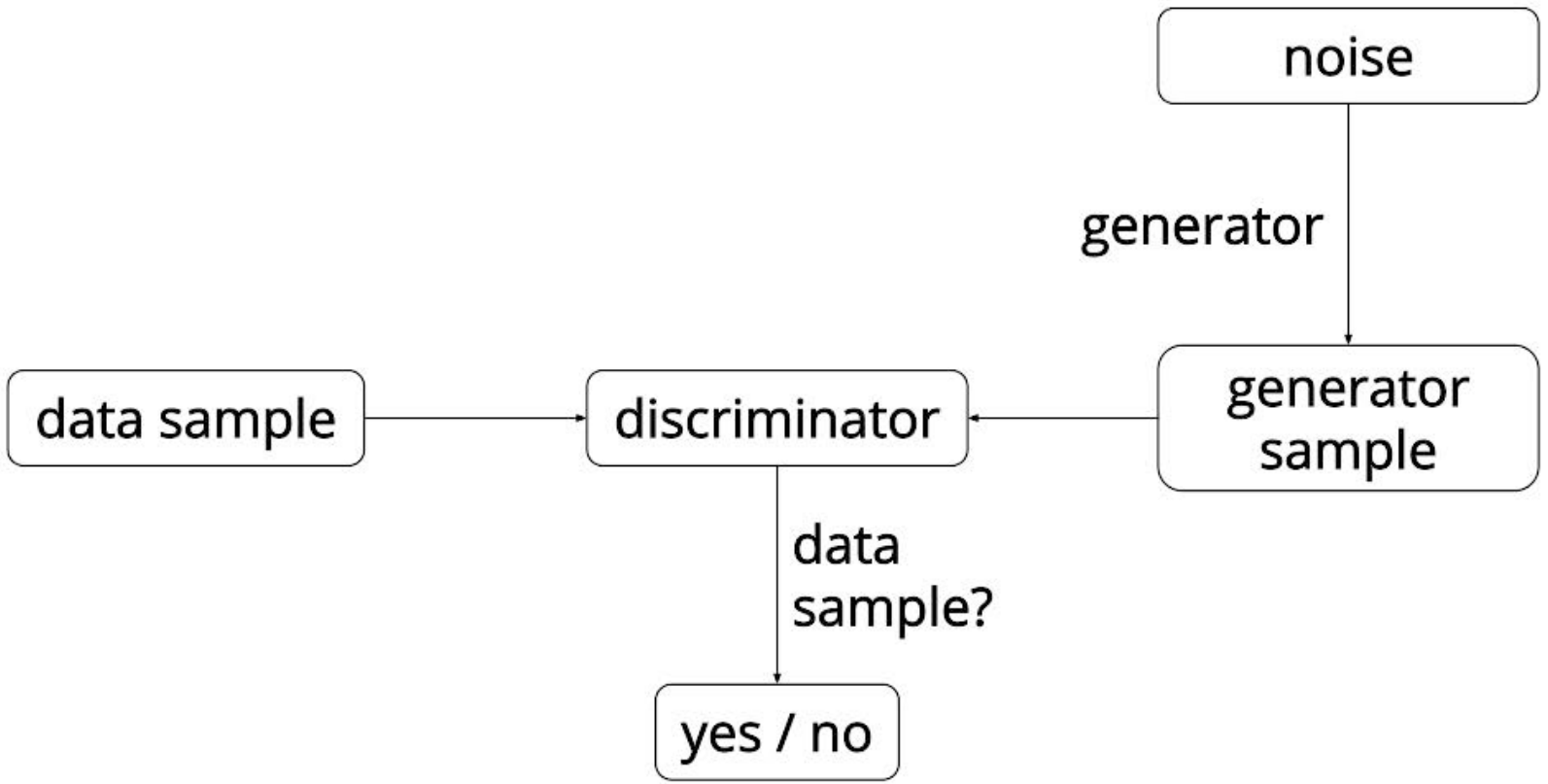
Generative Adversarial Networks (GANs)

two neural networks competing against each other in a zero-sum game framework

Generative Adversarial Networks (GANs)

- It turns out that CNNs can be turned around and applied to image generation as well.
- If we've got a bunch of images, how can we generate more like them?
- A recent method, Generative Adversial Networks, attempts to train an image generator by simultaneously training a discriminator to challenge it to improve.

Generative Adversarial Networks (GANs)



GAN – An intuitive example

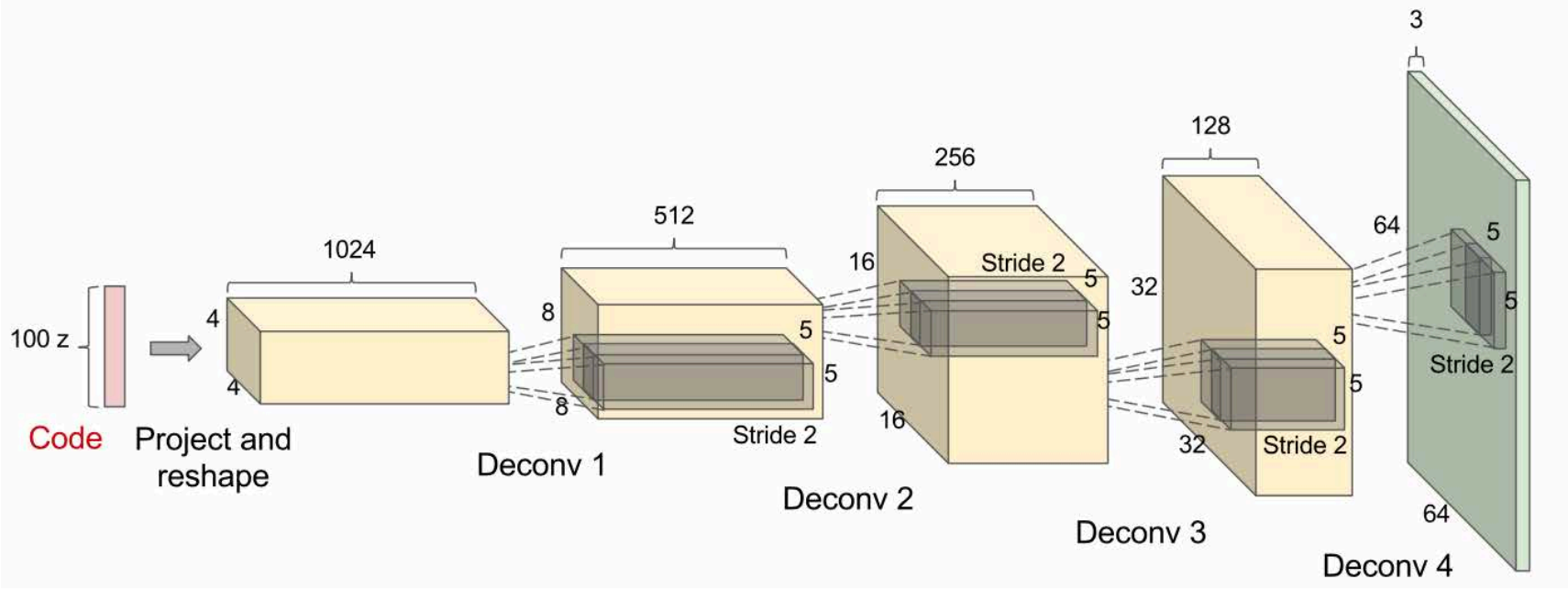
- Think of a back-and-forth situation between a bank and a money counterfeiter.
- At the beginning, the fakes are easy to spot.
- As the counterfeiter keeps trying different kinds of techniques, some may get past the check.
- The counterfeiter then can improve his fakes towards the areas that got past the bank's security checks.
- But the bank doesn't give up. It also keeps learning how to tell the fakes apart from real money.
- After a long period of back-and-forth, the competition has led the money counterfeiter to create perfect replicas.

GAN – An intuitive example

- Now, let the money forger have a spy in the bank that reports back how the bank is telling fakes apart from real money.
- Every time the bank comes up with a new strategy to tell apart fakes, such as using ultraviolet light, the counterfeiter knows exactly what to do to bypass it, such as replacing the material with ultraviolet marked cloth.
- This is exactly what a **GAN** does.

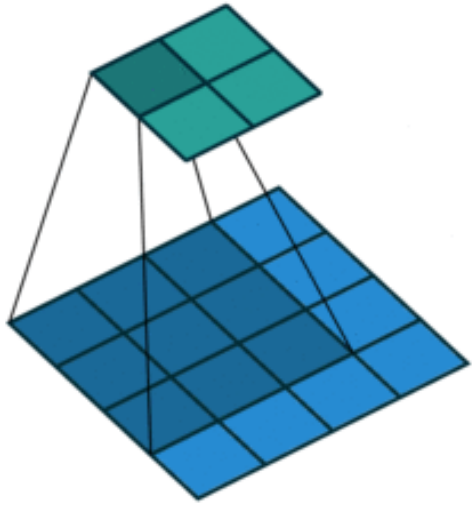
GAN – An intuitive example

- The bank is known as a **discriminator network**
  - in the case of images, it is a **CNN** which assigns a probability if an image is real and not fake.
- The counterfeiter is known as the **generative network**
  - It's a special kind of convolutional network that uses transpose convolutions, sometimes known as a **deconvolutional network**.

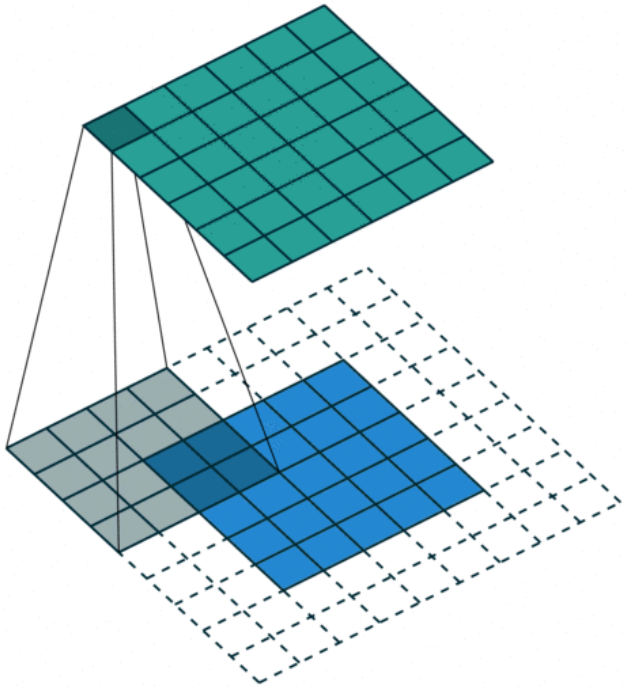


This generative network takes in some, say, 100 parameters of noise (sometimes known as the *code*), and outputs an image accordingly.

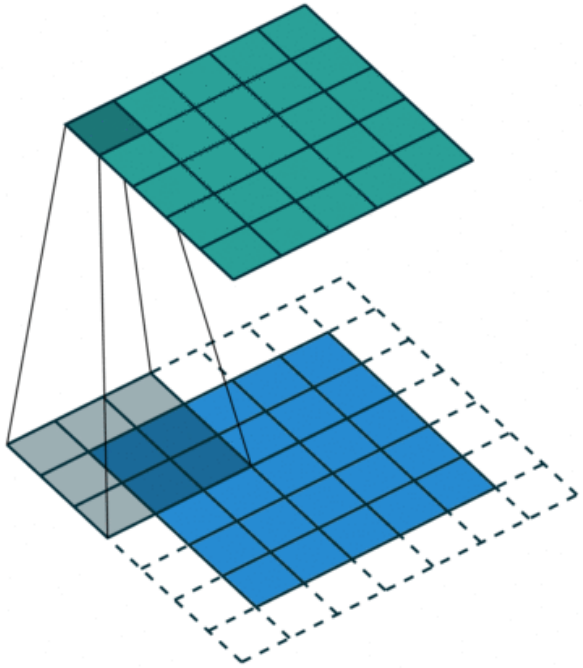
Convolution Animations (from bottom to top)



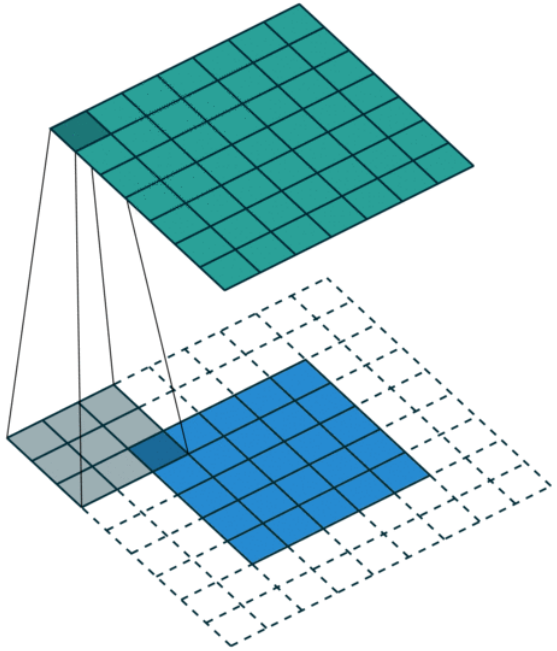
No padding, no strides



Arbitrary padding, no strides

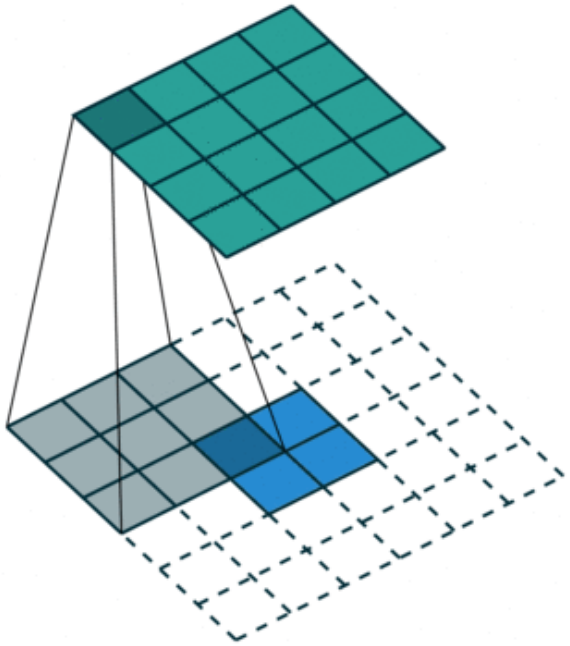


Half padding, no strides

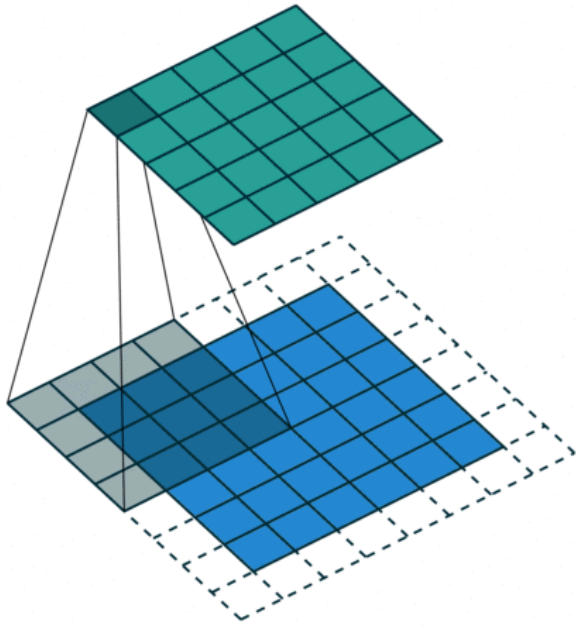


Full padding, no strides

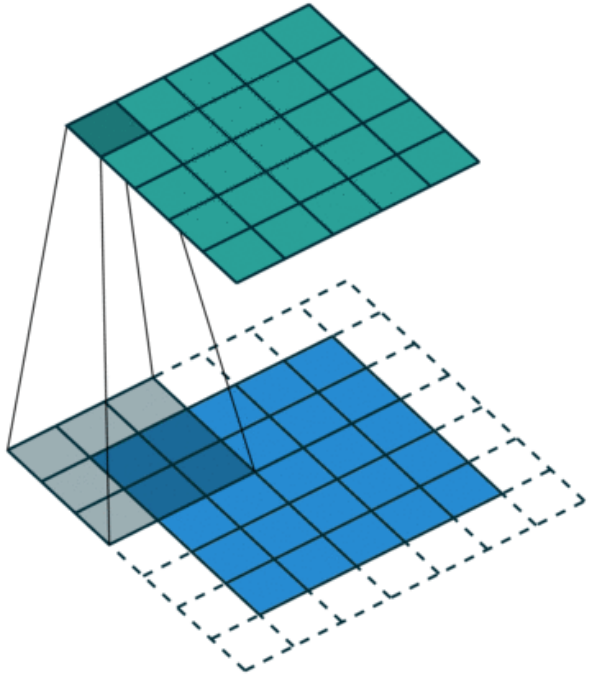
Deconvolution(!) Animations – Transposed Convolution (from top to bottom)



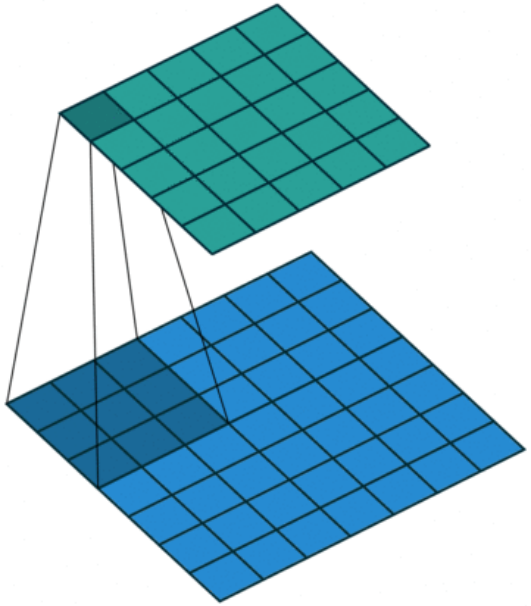
No padding, no strides, transposed



Arbitrary padding, no strides, transposed



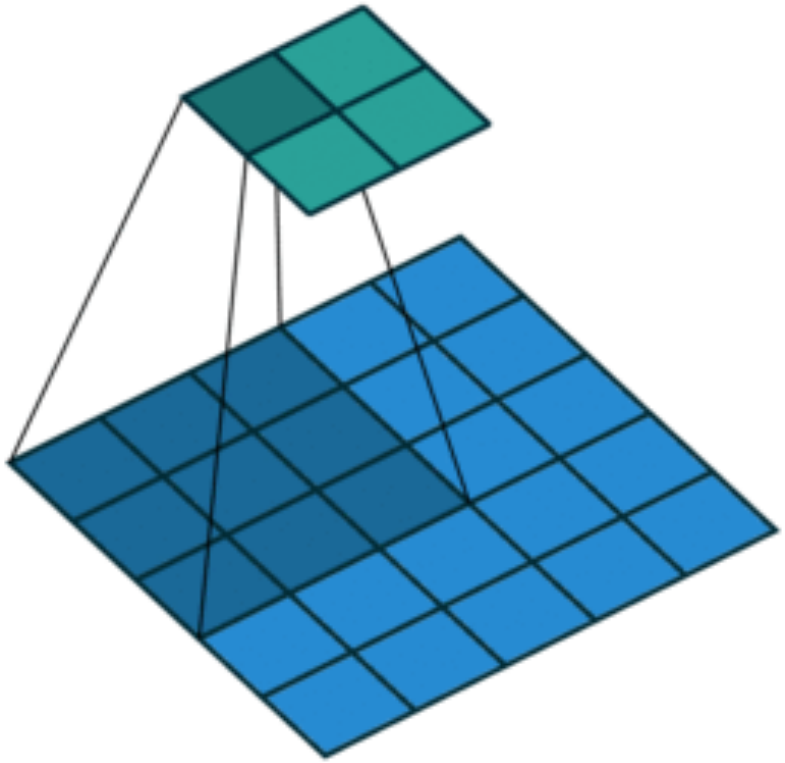
Half padding, no strides, transposed



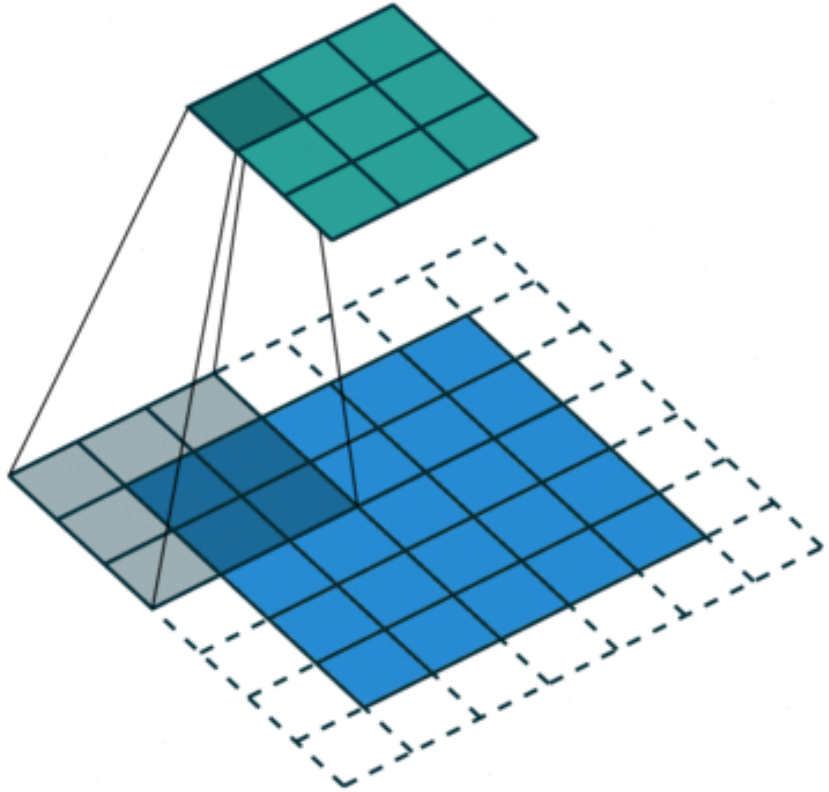
Full padding, no strides, transposed

https://github.com/vdumoulin/conv_arithmetic

Convolution with Strides (from bottom to top)



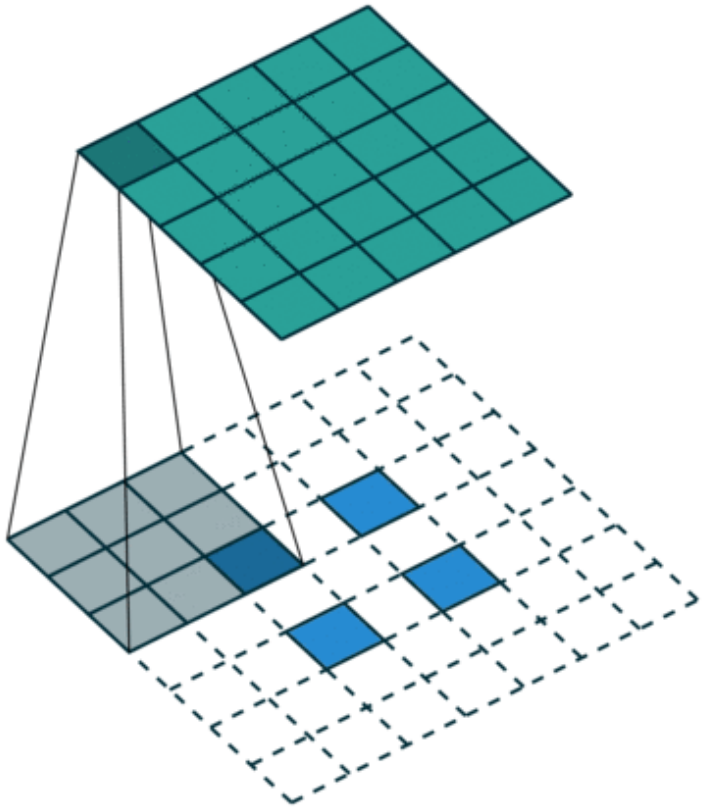
No padding, strides



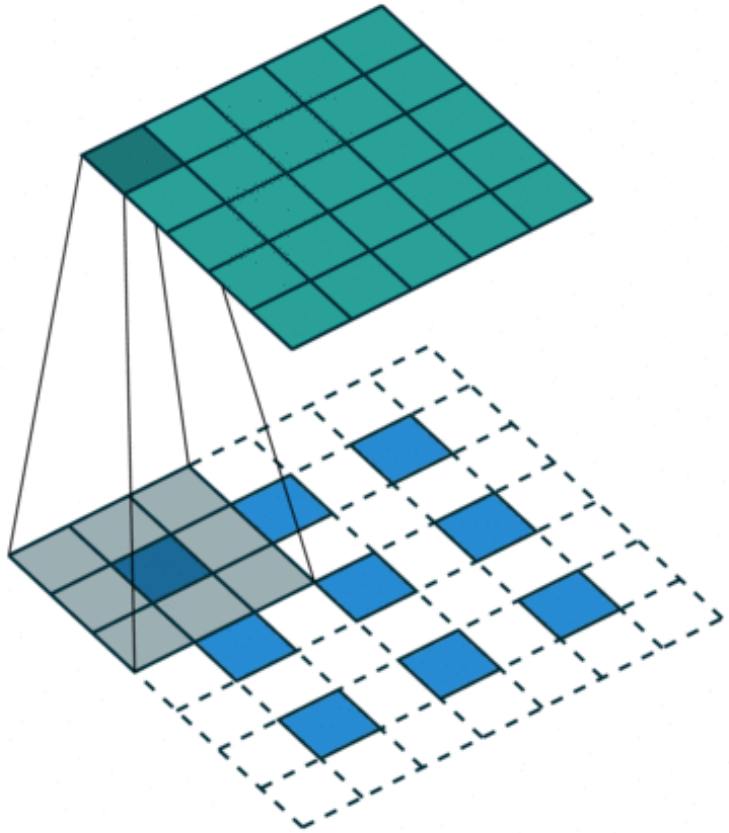
padding, strides

https://github.com/vdumoulin/conv_arithmetic

Transposed Convolution with Strides (from top to bottom)



No padding, strides, transposed



padding, strides, transposed

https://github.com/vdumoulin/conv_arithmetic

Generative Adversarial Networks (GANs)

- Which part was the spy?
- Since the discriminator was just a convolutional neural network, we can back-propogate to find the gradients of the input image.
- This tells us what parts of the image to change in order to yield a greater probability of being real in the eyes of the discriminator network.
- All that's left is to update the weights of our generative network with respect to these gradients, so the generative network outputs images that are more "real" than before.

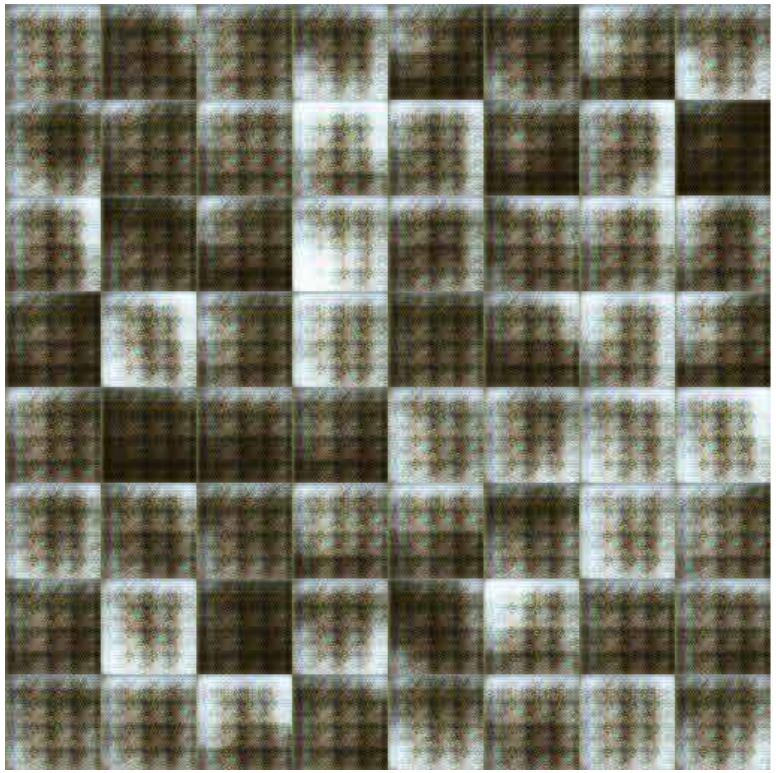
Generative Adversarial Networks (GANs)

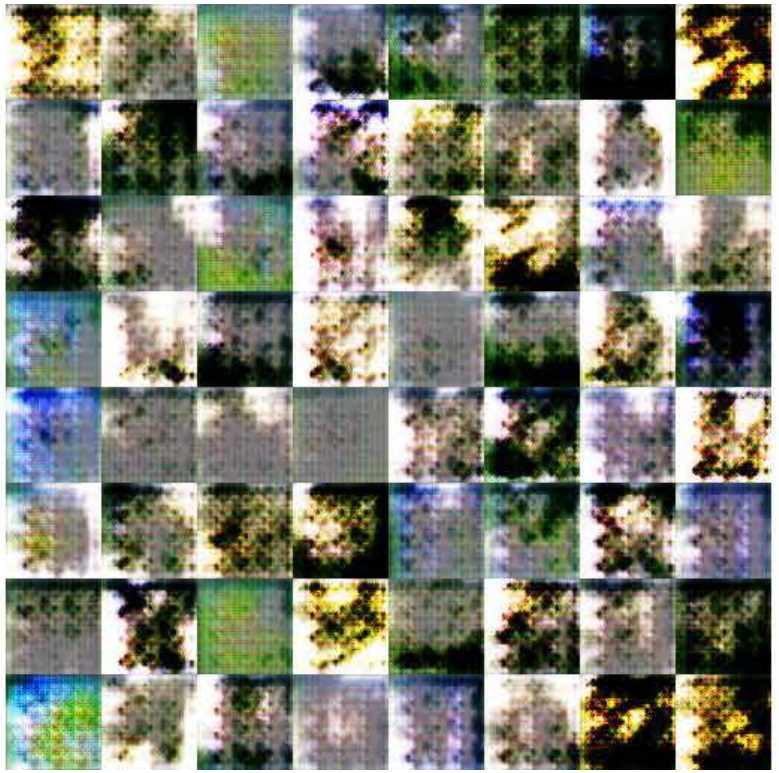
- The two networks are now locked in a competition.
- The discriminative network is constantly trying to find differences between the fake images and real images, and the generative network keeps getting closer and closer to the real deal.
- In the end, we've trained the generative network to produce images that can't be differentiated from real images.

GAN – CIFAR-10 – 32x32
(iterations ~= 300, 900, 6000)

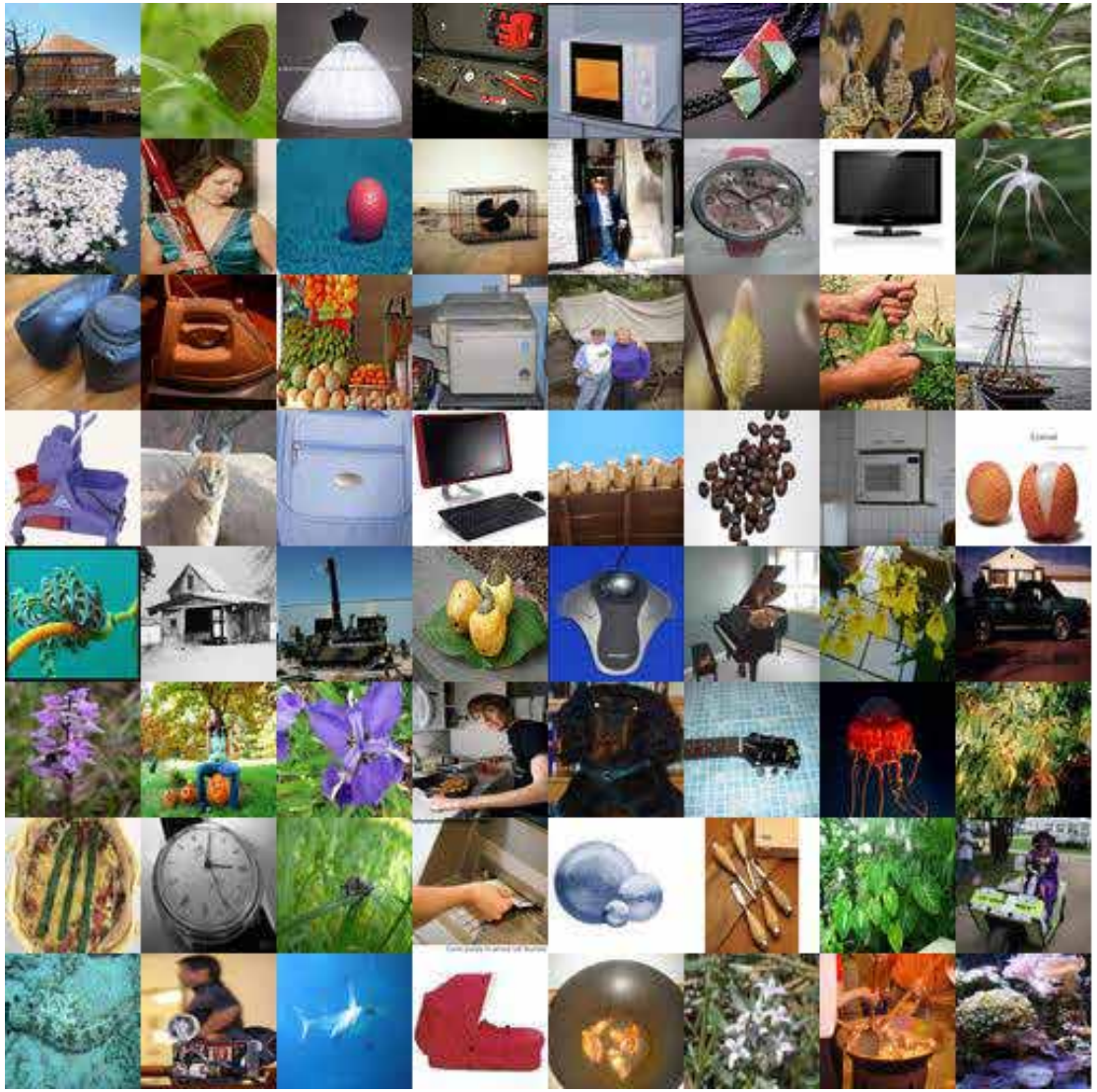
GAN – CIFAR-10 – 32x32
Real Images (left), Generated Images (right)

GAN – Sub-sampled ImageNet (64x64) (iterations ~= 300, 900, 6000)





GAN – Sub-sampled ImageNet (64x64)
Real Images (left), Generated Images (right)



GAN – Sub-sampled ImageNet (64x64)

- When generating from CIFAR or ImageNet, there are no concept of classes in the GAN.
- The network is not learning to make images of specific classes, it is learning to make images that look real in general.
- The problem is, this results in some image that may have some combination of features from all sorts of objects
  - like the **outline of a car** but the **coloring of a cow**.
- For the simplicity of the simple GAN, it does a pretty good job in creating images that look real, at least from a distance.

GAN - Summary

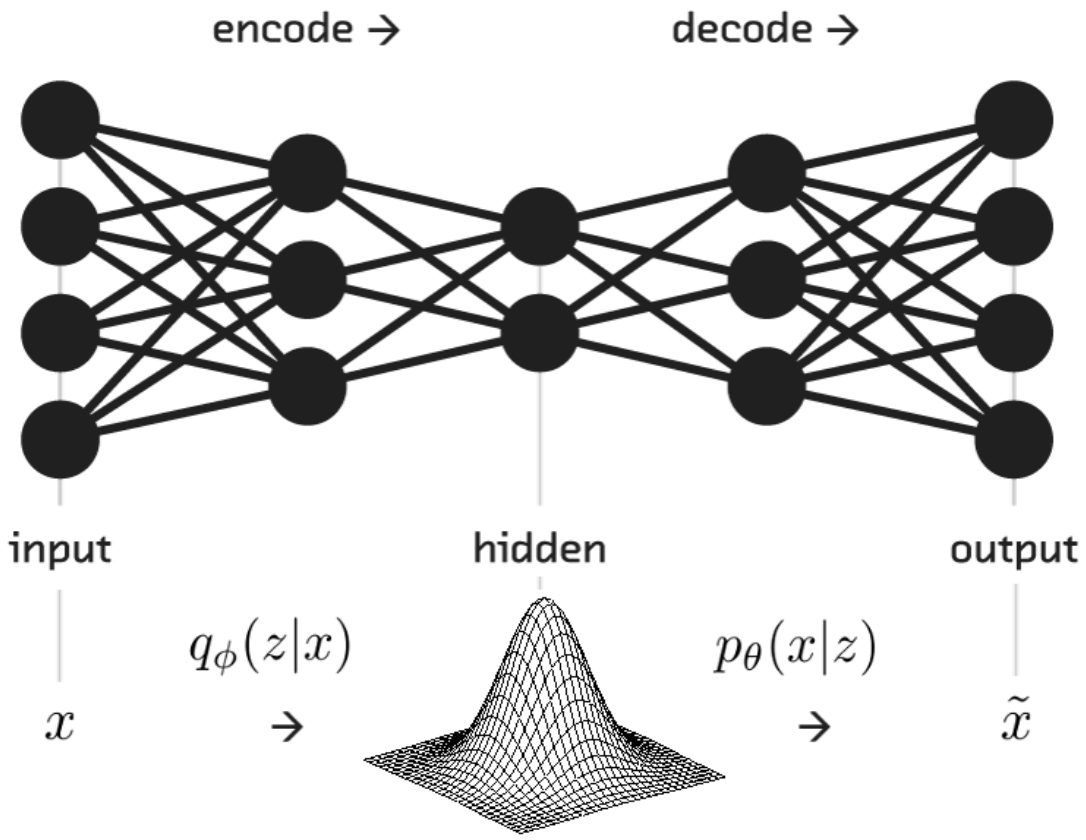
- The **generative network**'s training objective is to increase the error rate of the **discriminative network**.
- In practice, a particular dataset serves as the training data for the discriminator.
- Training the discriminator involves presenting the discriminator with samples from the dataset and samples synthesized by the generator, and doing logistic regression.
- In order to produce a sample, the generator is seeded with a randomized input that is sampled from a predefined latent space (e.g., a multivariate normal distribution).
- Training the generator involves back-propagating the negation of the binary classification loss of the discriminator.
- The generator adjusts its parameters so that the training data and generated data cannot be distinguished by the discriminator model.
- The goal is to find a setting of parameters that makes generated data look like the training data to the discriminator network.
- **In practice, the generator is typically a deconvolutional neural network, and the discriminator is a convolutional neural network.**

Generative Adversarial Networks (GANs)

- They have recently been used
  - to model very basic patterns of motion in video.
  - to reconstruct 3D models of objects from images.
  - to improve astronomical images.
- Downsides:
  - the images are generated off some arbitrary noise. If you wanted to generate a picture with specific features, there's no way of determining which initial noise values would produce that picture, other than searching over the entire distribution.
  - It only discriminates between "real" and "fake" images. There's no constraints that an image of a cat has to look like a cat. This leads to results where there's no actual object in a generated image, but the style just looks like picture.
- How to solve these problems?

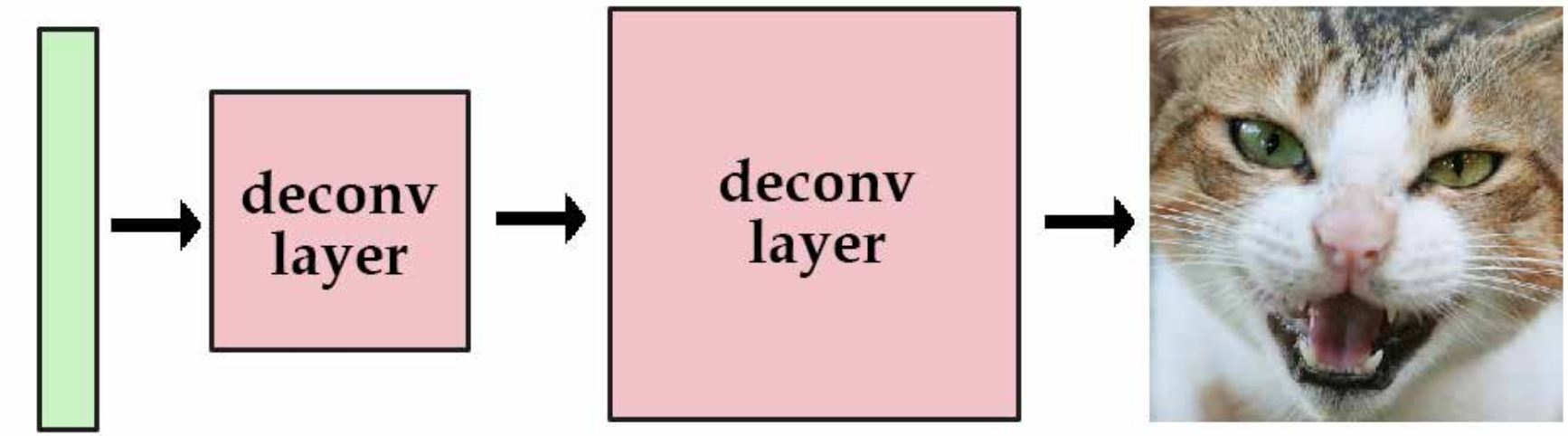
Variational Auto-encoders (VAEs)

the lovechild of Bayesian inference and unsupervised deep learning



Variational Auto-encoders (VAEs)

- Let's say we had a network comprised of a few deconvolution layers.
- We set the input to always be a **vector of ones**.
- we can train the network to reduce the mean squared error between itself and one target image.

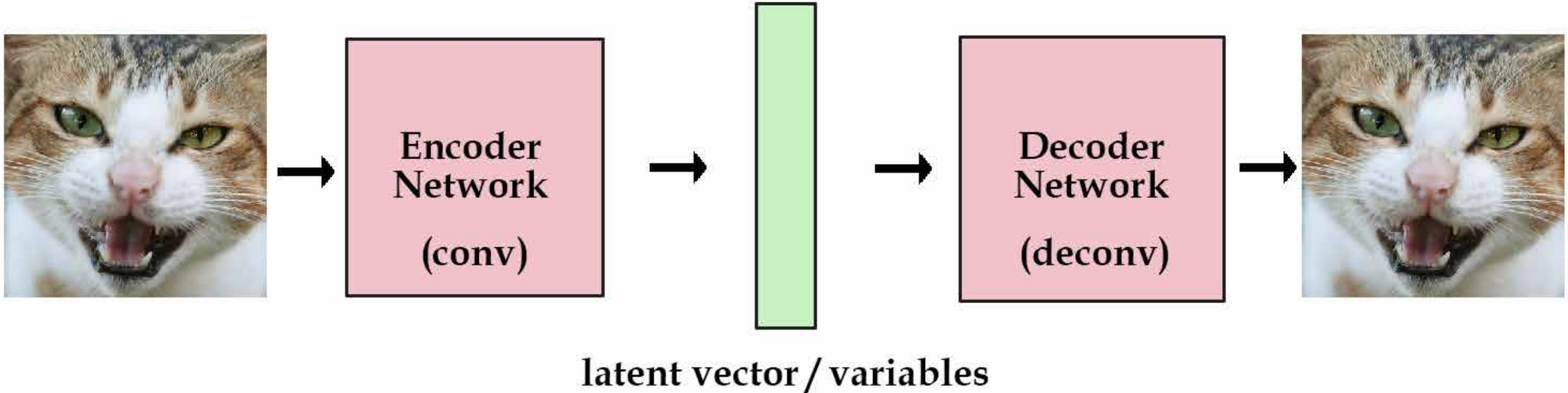


Variational Auto-encoders (VAEs)

- let's try it for multiple images.
- Instead of a vector of ones input, use a one-hot vector (1 of k encoding).
  - [1, 0, 0, 0] could mean a cat image, while [0, 1, 0, 0] could mean a dog.
  - This works, but we can only store up to 4 images.
  - Using a longer vector means adding in more and more parameters so the network can memorize the different images.
- To fix this, we use a vector of real numbers instead of a one-hot vector.
  - We can think of this as a code for an image, which is where the terms encode/decode come from.
  - For example, [3.3, 4.5, 2.1, 9.8] could represent the cat image, while [3.4, 2.1, 6.7, 4.2] could represent the dog.
- This initial vector is known as our **latent variables**.

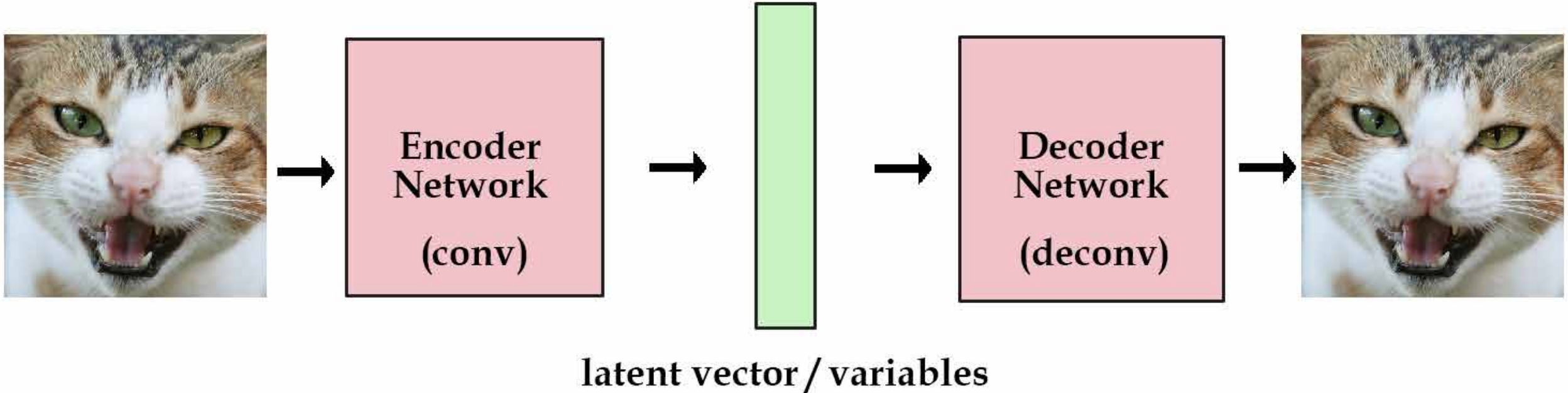
Variational Auto-encoders (VAEs)

- To make the auto-encoder complete let’s add its encoder part:
- The deconvolutional layers then "decode" the vectors back to the original images.
- We've finally reached a stage where our model has some hint of a practical use.
  - We can train our network on as many images as we want.
  - If we save the encoded vector of an image, we can reconstruct it later by passing it into the decoder portion.
  - What we have is the standard auto-encoder.



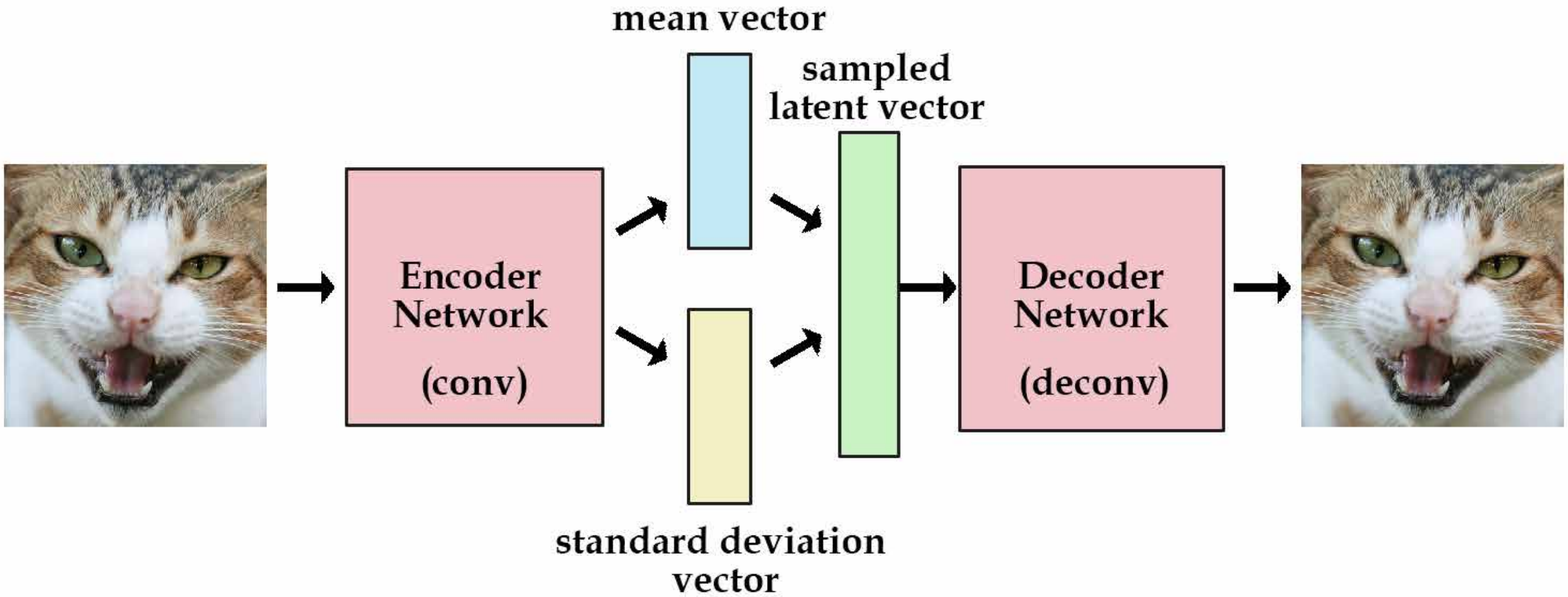
Variational Auto-encoders (VAEs)

- We're trying to build a generative model here, not just a fuzzy data structure that can "memorize" images.
- We can't generate anything yet, since we don't know how to create latent vectors other than encoding them from images.
- For that, we add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a unit Gaussian distribution.
- It is this constraint that separates a variational autoencoder from a standard one.



Variational Auto-encoders (VAEs)

re-parameterization trick

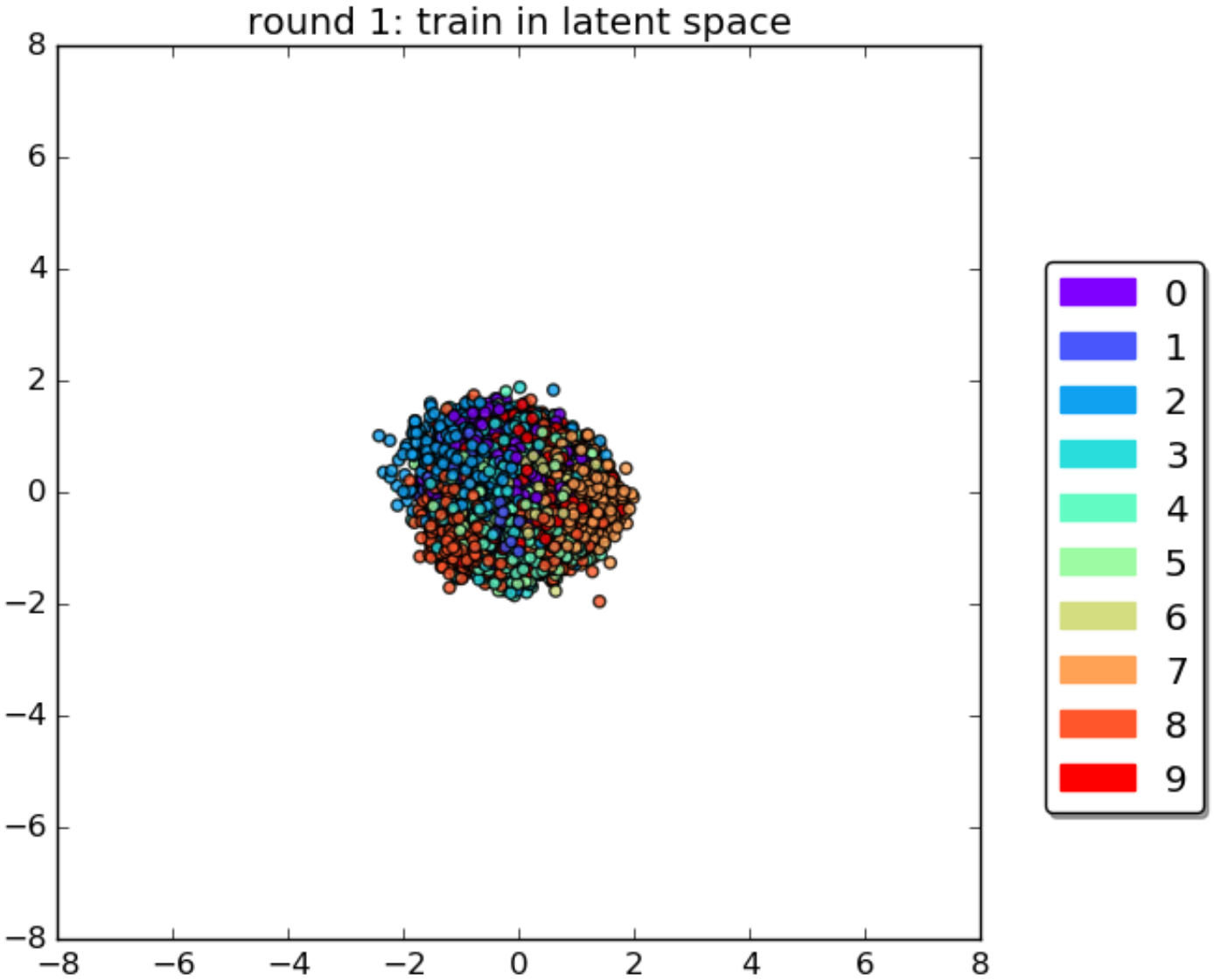


Variational Auto-encoders (VAEs)

- Generating new images is now easy:
  - all we need to do is sample a latent vector from the unit Gaussian and pass it into the decoder.
- In practice, there's a tradeoff between how accurate our network can be and how close its latent variables can match the unit Gaussian distribution.
  - We let the network decide this itself.
- For our loss term, we sum up two separate losses:
  - the generative loss, which is a mean squared error that measures how accurately the network reconstructed the images, and,
  - a latent loss, which is the **KL divergence** that measures how closely the latent variables match a unit Gaussian.

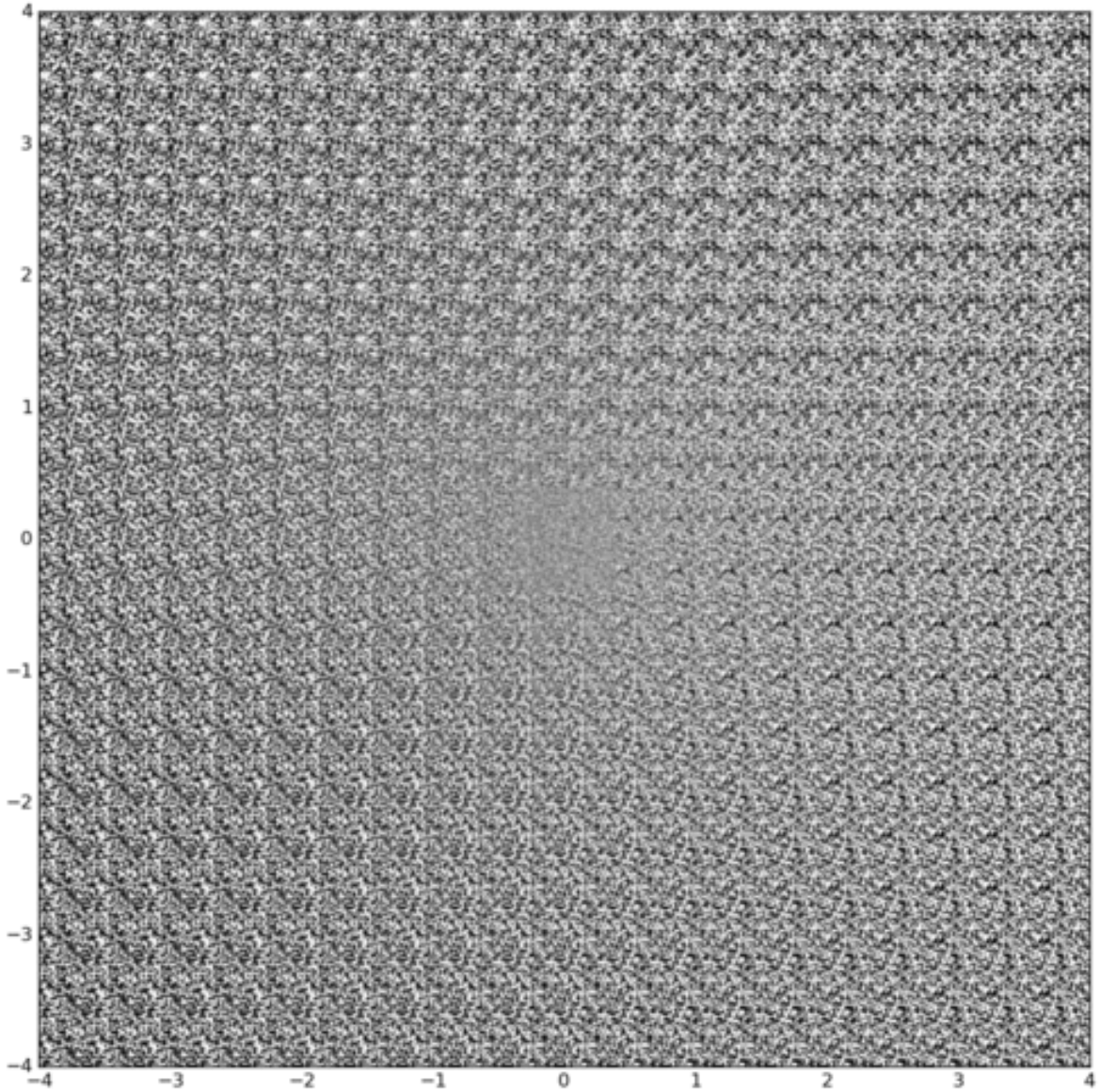
Variational Auto-encoders (VAEs) – MNIST 2D embedding

how the encoder/inference network learns to map the training set from the input data space to the latent space



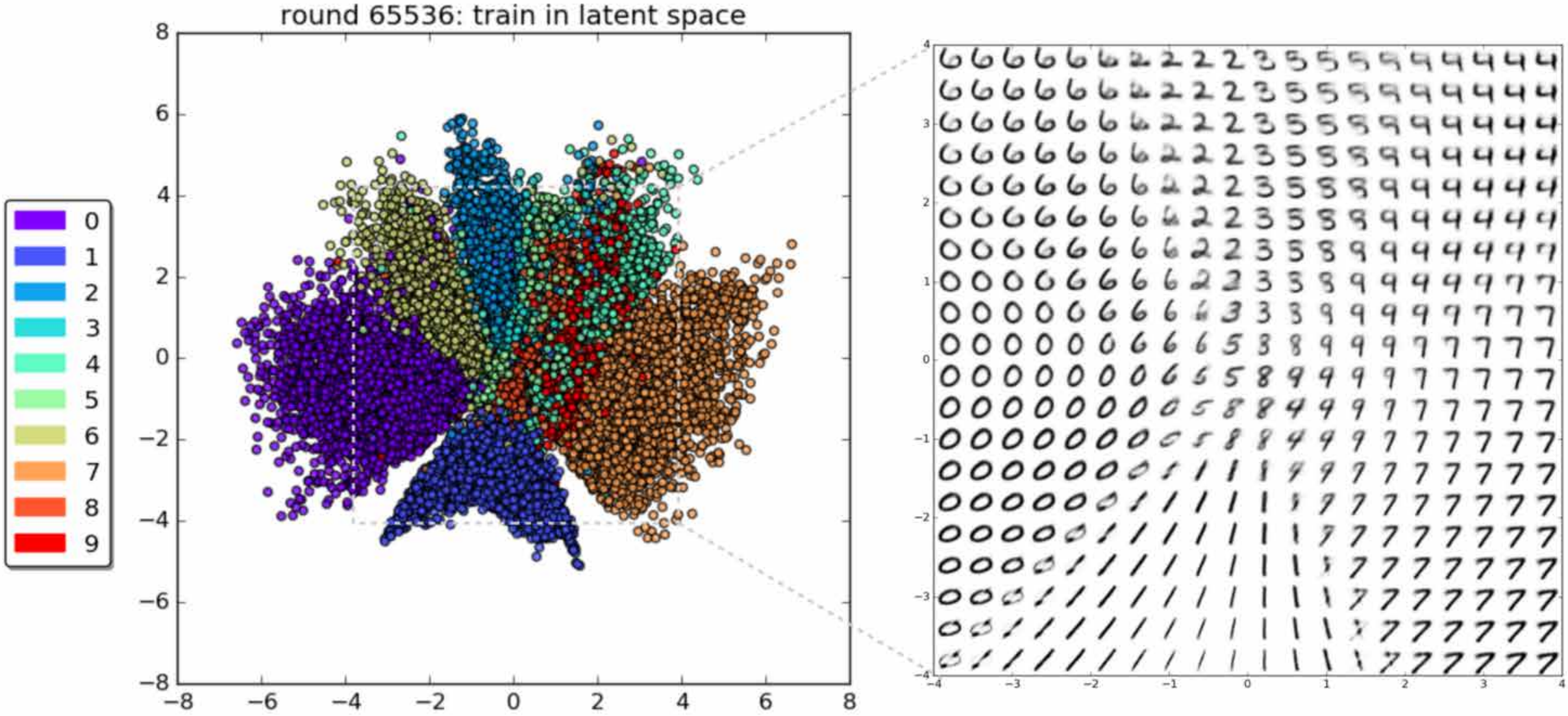
Variational Auto-encoders (VAEs) - MNIST 2D embedding

how the decoder/generative network learns to map latent coordinates into reconstructions of the original data space



Variational Auto-encoders (VAEs) - MNIST 2D embedding

how optimizing the encoder and decoder in tandem enables efficient pairing of inference and generation



Demos

- Demos of MNIST and SGVB data sets
  - http://dpkingma.com/sgvb_mnist_demo/demo.html

- Demo of human face morphing
  - http://vdumoulin.github.io/morphing_faces/online_demo.html