DCGAN: Deep Convolutional Neural Network for Clustering Flowers

Kenneth Kihara

Radford et al. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. *arXiv:1511.06434*, 2016.

About Me

- Computer Engineering, B.S. from UCSD

About Me

- Computer Engineering, B.S. from UCSD
- Advantage gambler for 3 years

About Me

- Computer Engineering, B.S. from UCSD
- Advantage gambler for 3 years
- Galvanize Data Science Immersive graduate and TA

About Me

- Computer Engineering, B.S. from UCSD
- Advantage gambler for 3 years
- Galvanize Data Science Immersive graduate and TA
- Deep learning enthusiast

Motivation

- Neural networks need a lot of images to train, but it is hard to find a large labeled dataset

Motivation

- Neural networks need a lot of images to train, but it is hard to find a large labeled dataset
- Wanted a way to generate my own dataset with labels as well

Motivation

- Neural networks need a lot of images to train, but it is hard to find a large labeled dataset
- Wanted a way to generate my own dataset with labels as well
- Interesting to generate images that look “real”

Motivation

- Neural networks need a lot of images to train, but it is hard to find a large labeled dataset
- Wanted a way to generate my own dataset with labels as well
- Interesting to generate images that look “real”
- Experiment with a challenging model that made good use of tensorflow

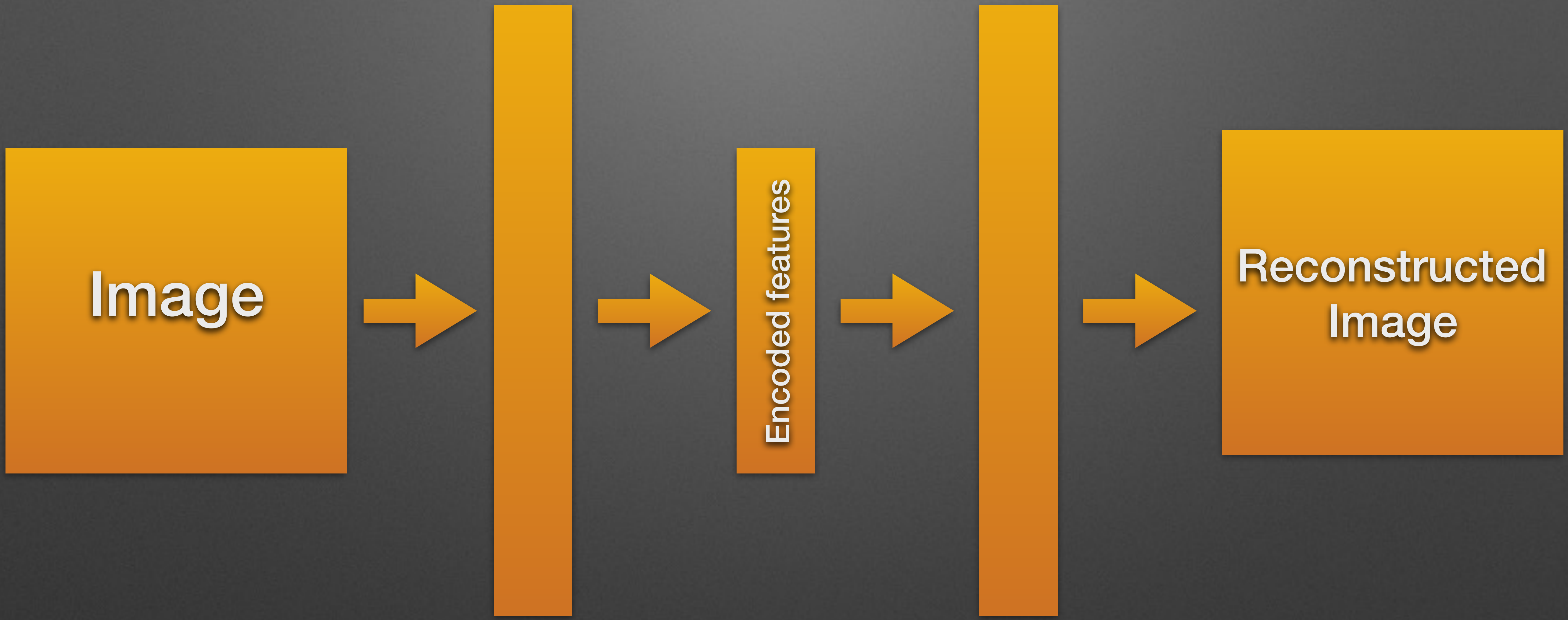
Unsupervised Learning

- Autoencoders
  - Train a set of features that can reconstruct an image

Unsupervised Learning

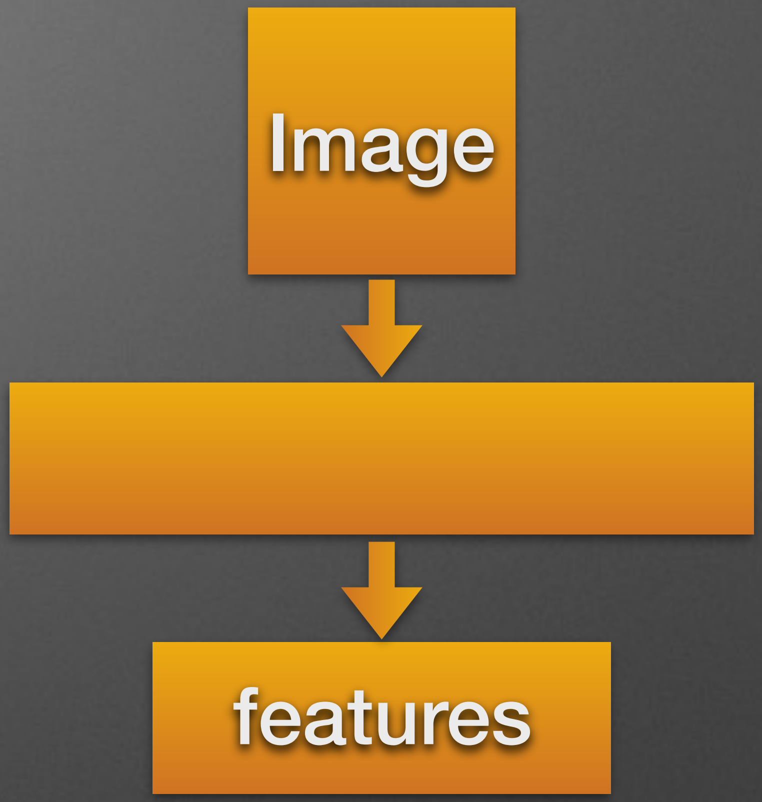
- Autoencoders
    - Train a set of features that can reconstruct an image
- Generative Adversarial Networks (GAN)
    - Train a network to reconstruct an image from noise

Autoencoder



Autoencoder

- Down sample to a set of features



Autoencoder

- Down sample to a set of features
- Up sample to the original image

Autoencoder

- Down sample to a set of features
- Up sample to the original image
- Use L2 loss and backprop to update weights

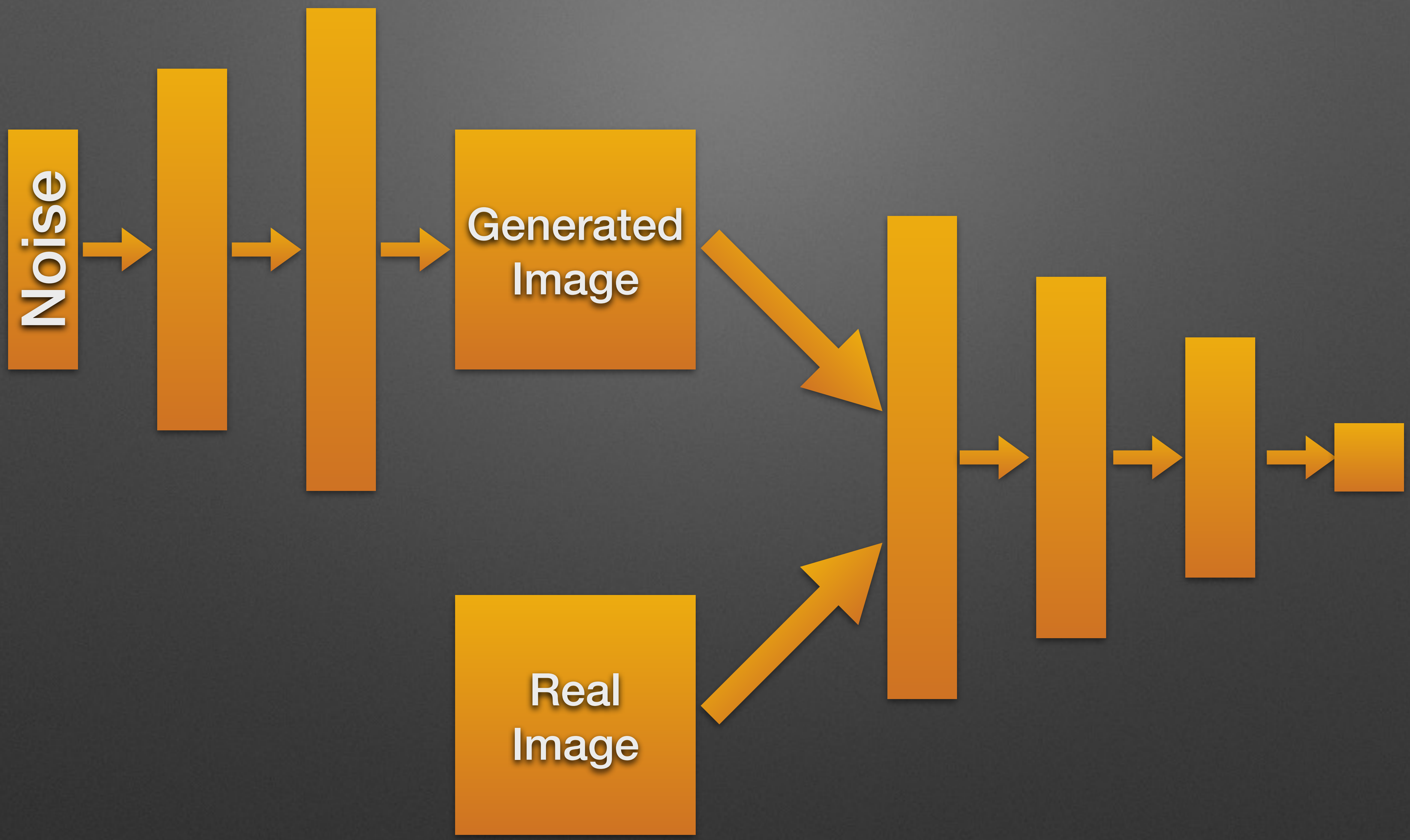
Autoencoder

- Down sample to a set of features
- Up sample to the original image
- Use L2 loss and backprop to update weights
- Cons:
  - Can't generate new images
  - Doesn't work as well in practice

Up/Down Sampling

$$WX + b = z$$

GAN

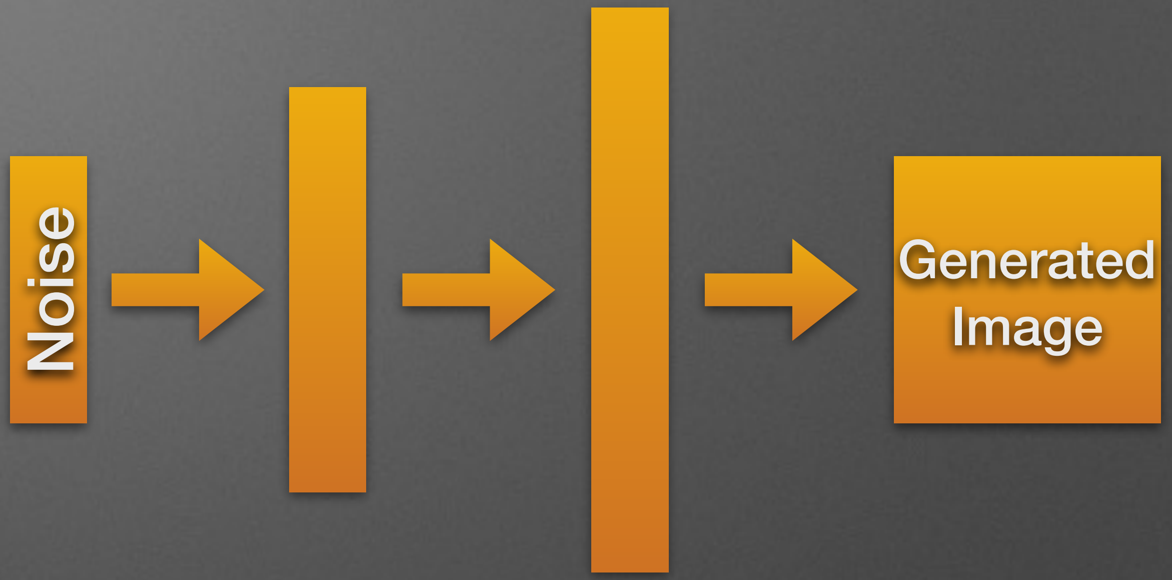


GAN

- Generator: create a real looking image
  - Input a noise vector

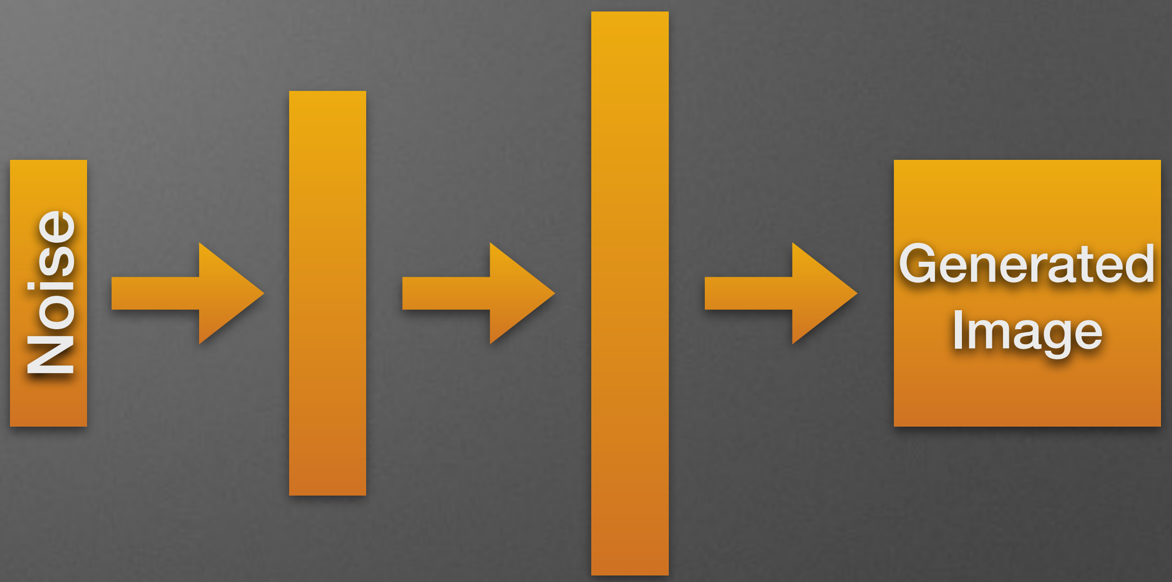
GAN

- Generator: create a real looking image
  - Input a noise vector
  - Up sample to an image



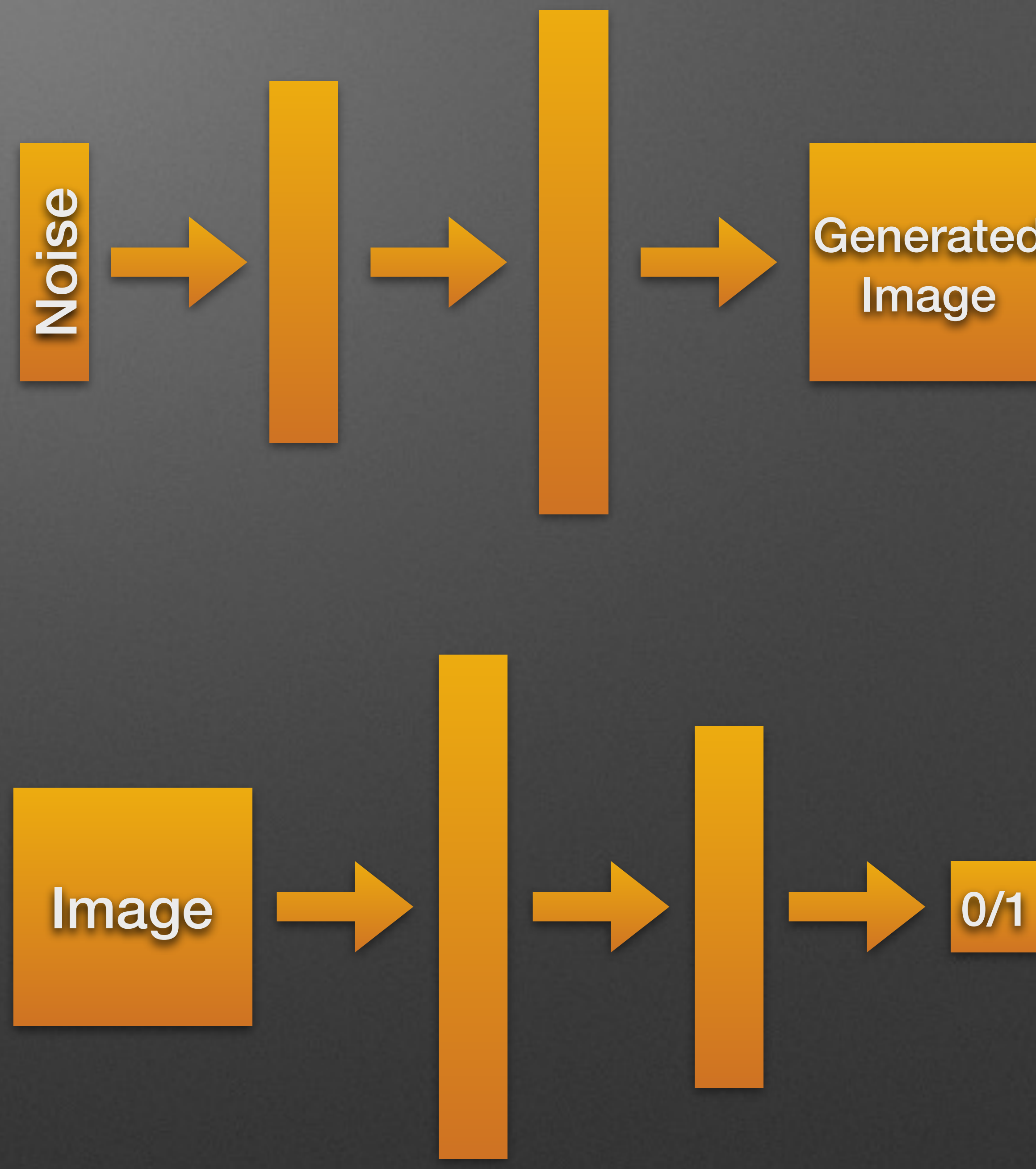
GAN

- Generator: create a real looking image
  - Input a noise vector
  - Up sample to an image
- Discriminator: discriminate between real and fake images
  - Input an image (real or fake)



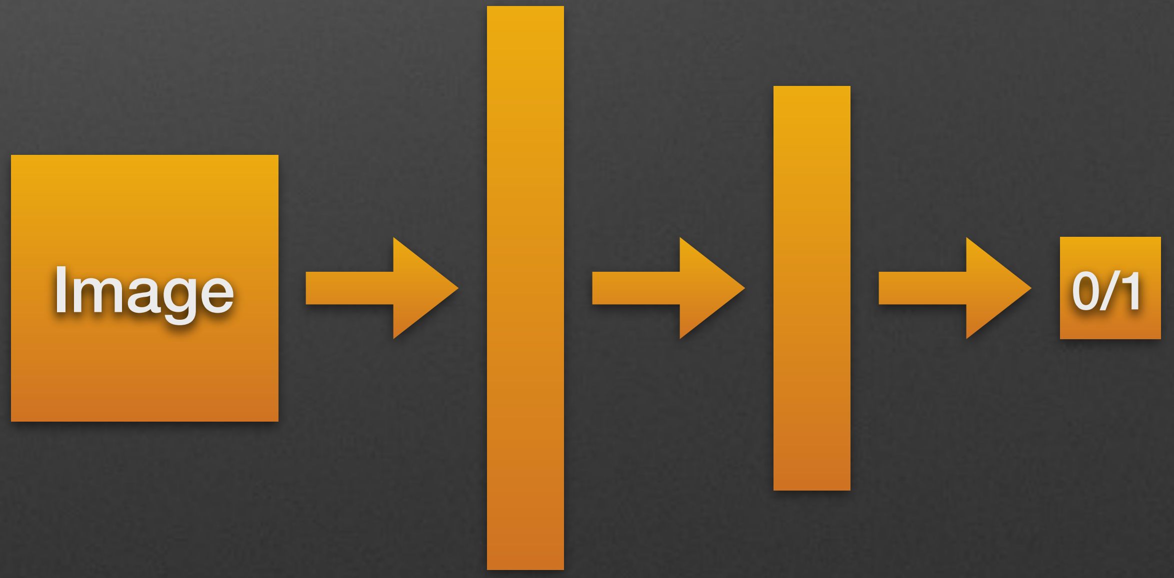
GAN

- Generator: create a real looking image
  - Input a noise vector
  - Up sample to an image
- Discriminator: discriminate between real and fake images
  - Input an image (real or fake)
  - Down sample to a probability, real or fake



GAN

- Generator: create a real looking image
  - Input a noise vector
  - Up sample to an image
- Discriminator: discriminate between real and fake images
  - Input an image (real or fake)
  - Down sample to a probability, real or fake
- Use a minimax cost function to update weights



GAN

- Minimax cost function

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

GAN

- Minimax cost function

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

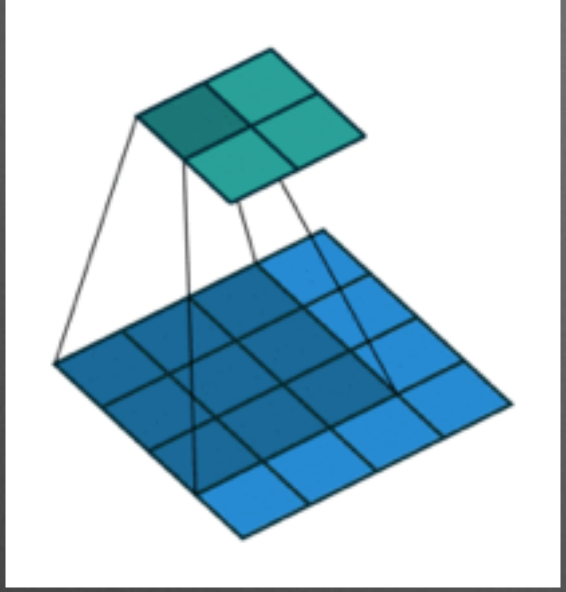
- My implementation

```
For number of training steps
    Sample n training images, X
    Compute n generated images, G

    Compute discriminator probabilities for X and G, D(X) and D(G)
    Label training images 1 and generated images 0
        Cost = (1/n)sum[log(D(X)) + log(1 - D(G))]
    Update discriminator weights, hold generator weights constant

    Label generated images 1
        Cost = (1/n)sum[log(D(G))]
    Update generator weights, hold discriminator weights constant
```

DCGAN

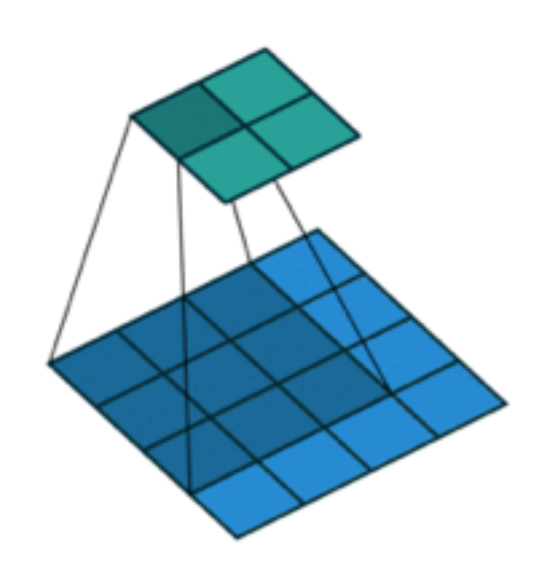
- “Deep Convolutional” Generative Adversarial Networks (DCGAN) use convolution and deconvolution layers

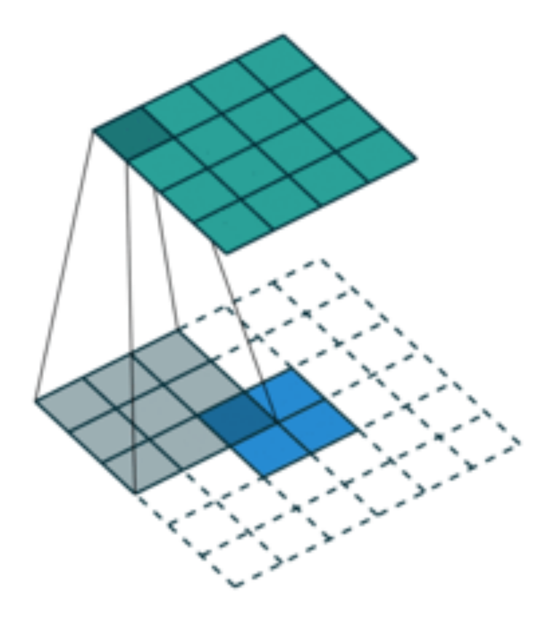
DCGAN



- “Deep Convolutional” Generative Adversarial Networks (DCGAN) use convolution and deconvolution layers
- Convolution: down sampling layer

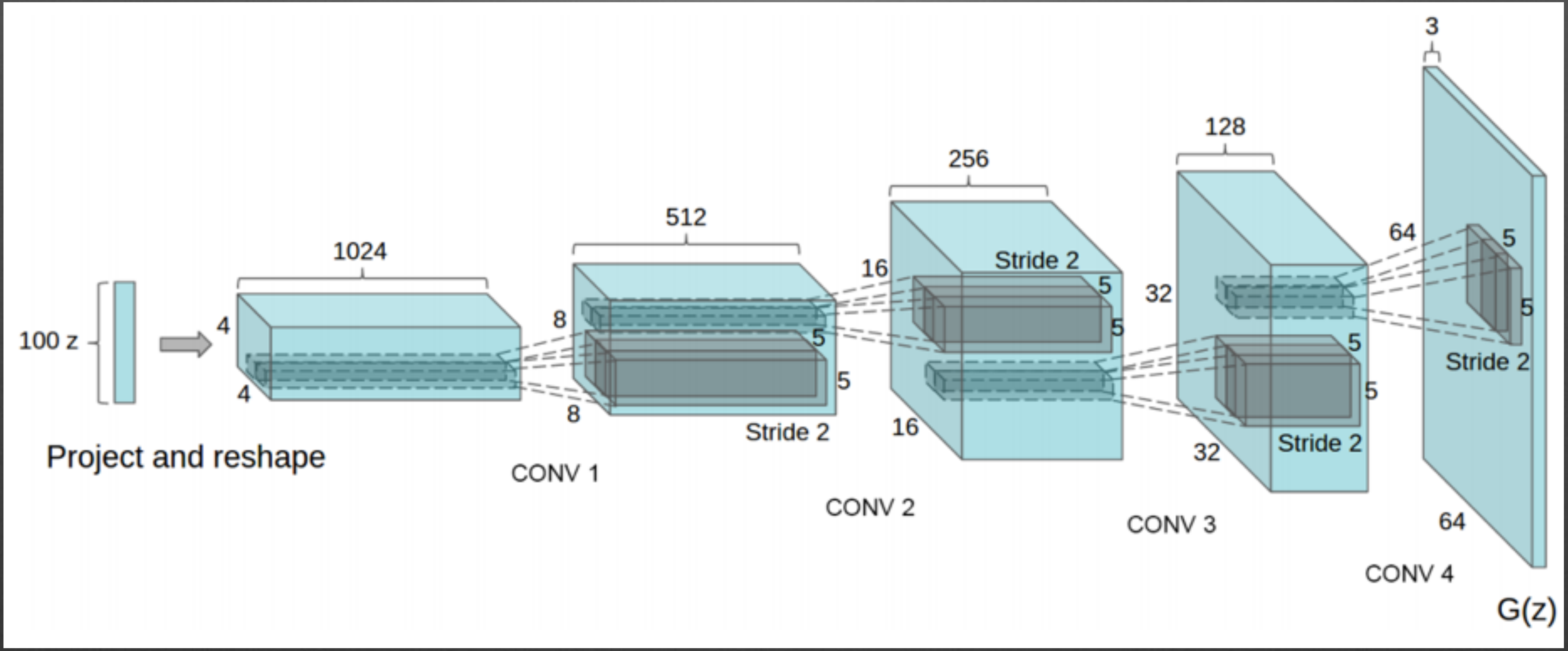
DCGAN





- “Deep Convolutional” Generative Adversarial Networks (DCGAN) use convolution and deconvolution layers
- Convolution: down sampling layer
- Deconvolution: up sampling layer

DCGAN



Radford et al. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. *arXiv:1511.06434*, 2016.

DCGAN

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. *arXiv:1511.06434*, 2016.

Results

- Autoencoder
  - ~2 x 10^8 parameters
  - 1500 steps
  - 128 batch size
  - 0.0001 learning rate
  - 8 hours

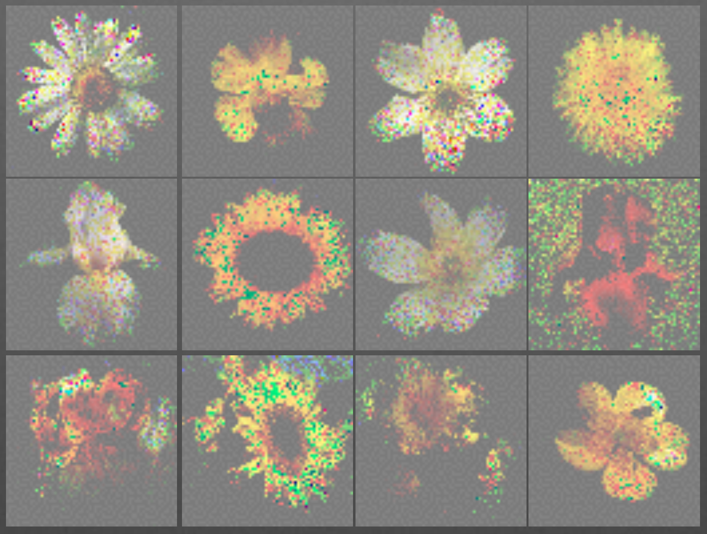
Results

- Autoencoder
  - ~2 x 10^8 parameters
  - 1500 steps
  - 128 batch size
  - 0.0001 learning rate
  - 8 hours
- DCGAN
  - ~2 x 10^6 parameters
  - 650k steps
  - 128 batch size
  - 0.00002 learning rate
  - 72 hours

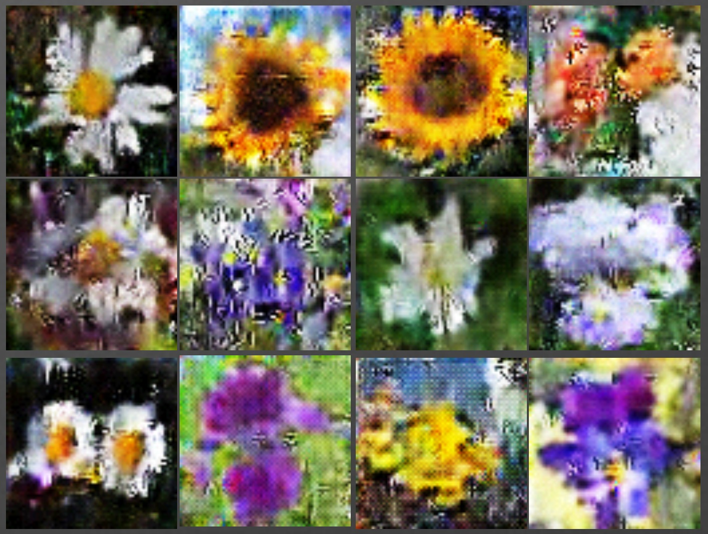
Results

Real

Autoencoder

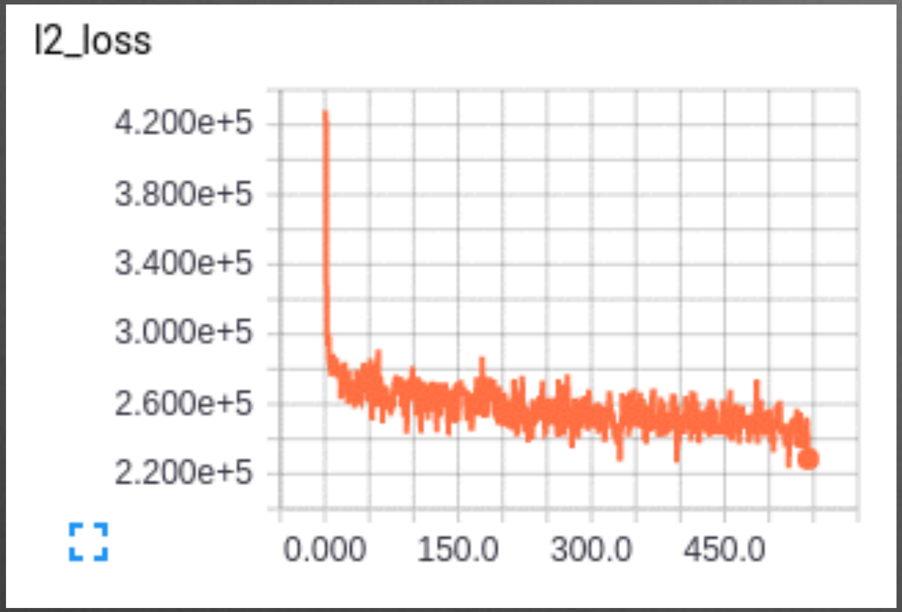


DCGAN

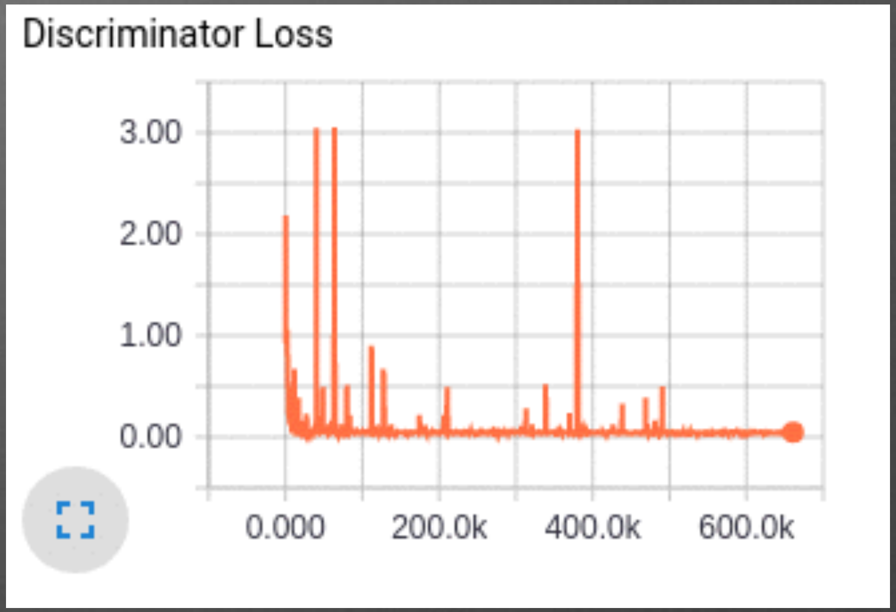


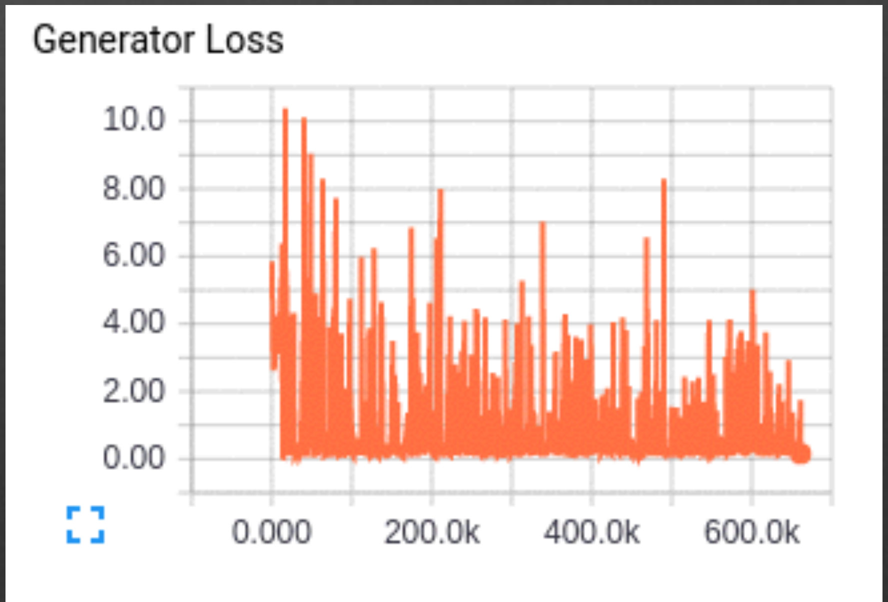
Results

Autoencoder



DCGAN

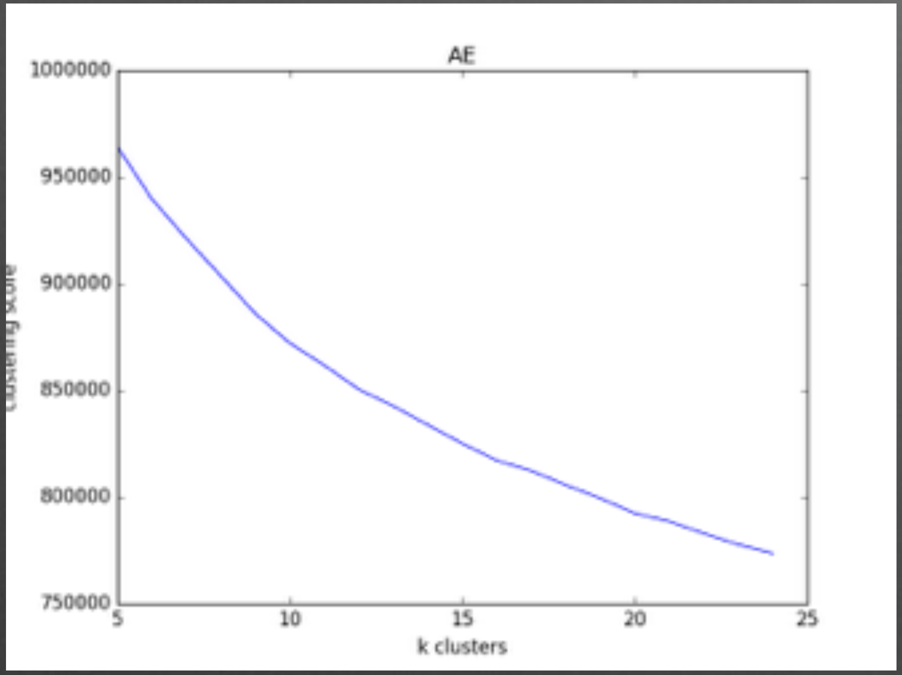


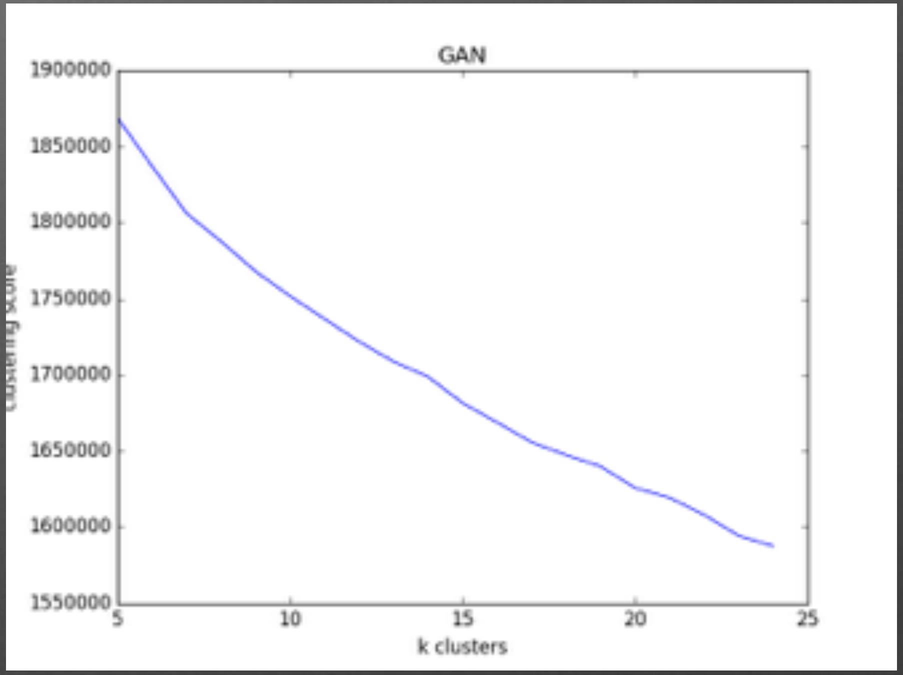


Results

Autoencoder

DCGAN





Results

Autoencoder

Cluster #

Label #

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0.05	0	0	0.13	0.01	0.10	0.01	0.26	0.03	0.01	0.02	0.16	0	0.08	0.08	0.01
1	0.03	0.11	0.03	0.08	0	0.08	0	0	0	0.17	0.03	0.03	0	0.03	0.11	0.12	0.11
2	0.02	0.10	0.03	0.05	0.01	0.06	0	0	0	0.21	0.02	0.23	0.01	0.01	0.06	0.05	0.10
3	0.02	0	0.01	0.37	0	0.05	0	0	0	0.27	0.01	0.02	0	0.11	0.01	0.01	0.08
4	0.01	0.03	0	0.18	0.11	0.07	0	0	0.01	0.11	0.07	0	0	0.11	0.13	0.05	0.07
5	0.07	0	0.25	0.05	0.01	0.01	0.03	0	0.05	0.03	0	0.01	0.1	0.2	0	0	0.16
6	0	0.03	0	0.06	0.20	0.15	0	0.02	0.05	0.10	0.01	0.02	0.01	0.05	0.10	0.06	0.11
7	0.03	0	0	0.05	0.10	0.02	0.06	0	0.02	0.17	0.02	0.10	0.01	0.01	0.05	0.12	0.20
8	0	0.01	0.02	0.11	0.01	0.06	0.01	0	0.02	0.16	0.07	0.01	0	0.17	0.11	0.05	0.15
9	0	0	0	0	0.02	0.05	0.02	0.61	0.03	0	0.01	0	0.05	0.13	0	0.01	0.03
10	0.28	0.06	0.31	0.02	0.03	0	0.05	0	0.02	0.03	0.06	0	0	0.02	0.06	0	0.01
11	0	0.03	0	0	0.02	0.02	0.10	0.02	0.12	0	0.10	0	0.30	0.01	0.05	0.05	0.15
12	0	0.03	0	0	0.11	0	0.17	0	0.16	0.01	0.12	0	0.31	0	0.01	0.03	0.01
13	0	0.07	0	0.01	0.12	0.07	0	0	0.08	0.15	0.02	0.06	0.06	0.02	0.10	0.05	0.15
14	0.01	0.03	0	0	0.05	0	0.20	0.01	0.15	0.05	0.02	0.06	0.23	0.02	0.02	0.08	0.02
15	0.30	0.07	0.22	0.02	0.05	0	0.03	0	0.02	0	0.05	0.02	0.01	0	0.11	0.03	0.02
16	0.07	0.02	0.05	0.43	0	0	0	0	0.01	0.07	0.05	0	0.02	0.06	0.02	0.11	0.05

Results

DCGAN

Cluster #

Label #

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0.17	0.06	0.56	0	0	0.01	0	0	0.02	0	0	0	0.13	0.02	0	0	0
1	0	0.32	0	0.02	0.01	0.01	0.01	0	0.07	0	0	0.23	0.30	0	0	0	0
2	0	0.08	0	0.01	0.01	0	0	0	0.57	0	0	0.07	0.21	0	0.02	0	0
3	0	0.01	0	0	0	0.01	0.02	0.20	0.11	0	0	0.01	0.61	0	0	0	0.01
4	0	0.12	0.02	0	0.01	0	0.02	0.13	0.01	0	0	0.05	0.60	0	0	0	0.01
5	0.05	0.02	0.05	0.40	0.06	0.06	0.07	0.05	0	0	0	0	0.11	0	0	0.02	0.08
6	0	0.02	0.03	0.01	0	0.02	0.01	0	0.06	0	0.08	0.01	0.25	0.01	0	0.46	0
7	0.01	0.01	0.37	0	0	0	0	0	0.03	0	0	0.01	0.53	0	0.01	0	0
8	0	0.05	0	0.45	0	0.01	0	0	0.01	0	0	0.03	0.42	0	0	0.01	0
9	0.4	0	0.10	0.01	0	0.01	0	0	0	0	0	0	0.05	0.26	0	0.16	0
10	0	0.18	0.01	0	0.12	0.01	0	0	0.01	0.05	0	0.06	0.05	0.01	0	0	0.47
11	0.35	0.01	0.36	0	0	0	0	0	0	0	0	0	0.25	0.01	0	0.01	0
12	0.42	0	0.46	0	0	0	0	0	0.01	0	0	0	0.08	0	0	0.01	0
13	0	0.02	0.20	0	0	0	0	0	0.10	0	0	0	0.67	0	0	0	0
14	0.42	0.02	0.33	0	0	0.01	0	0	0.06	0	0	0	0.08	0.05	0	0	0
15	0	0.20	0	0	0.21	0	0	0	0.05	0	0	0.08	0.06	0	0	0	0.38
16	0.01	0.08	0.07	0.02	0.03	0.02	0	0.52	0.02	0	0	0	0.17	0	0	0	0.01

Results

Overlapping Clusters

Cluster 2

Daffodil

Colts' Foot

Dandelion

Cluster 12

Bluebell

Cocus

Tulip

Cowslip

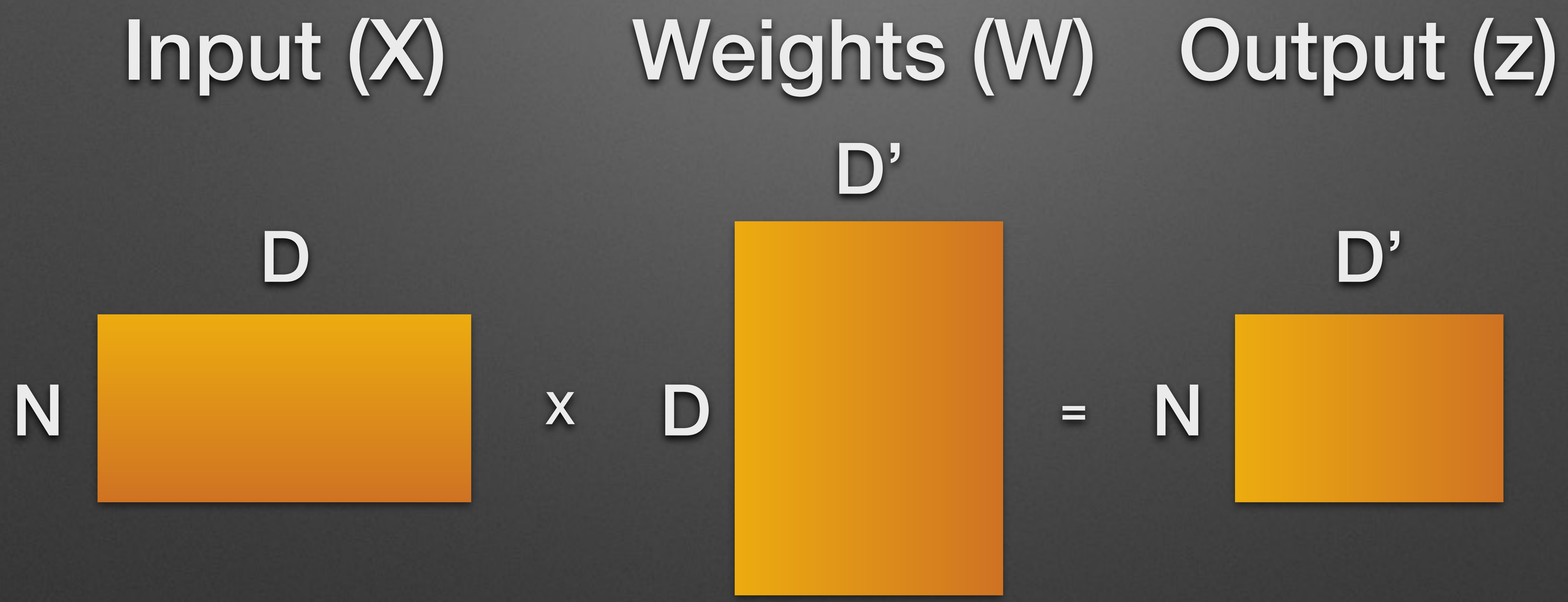
Future Work

- Experiment more with the DCGAN model
  - What happens when including pooling layers?
  - Why LeakyRELU for the discriminator?
- Try other clustering methods

Appendix
Class Names

Label #	Class Name
0	Daffodil
1	Snowdrop
2	Lily Valley
3	Bluebell
4	Cocus
5	Iris
6	Tigerlily
7	Tulip
8	Fritillary
9	Sunflower
10	Daisy
11	Colts' Foot
12	Dandelion
13	Cowslip
14	Buttercup
15	Windflower
16	Pansy

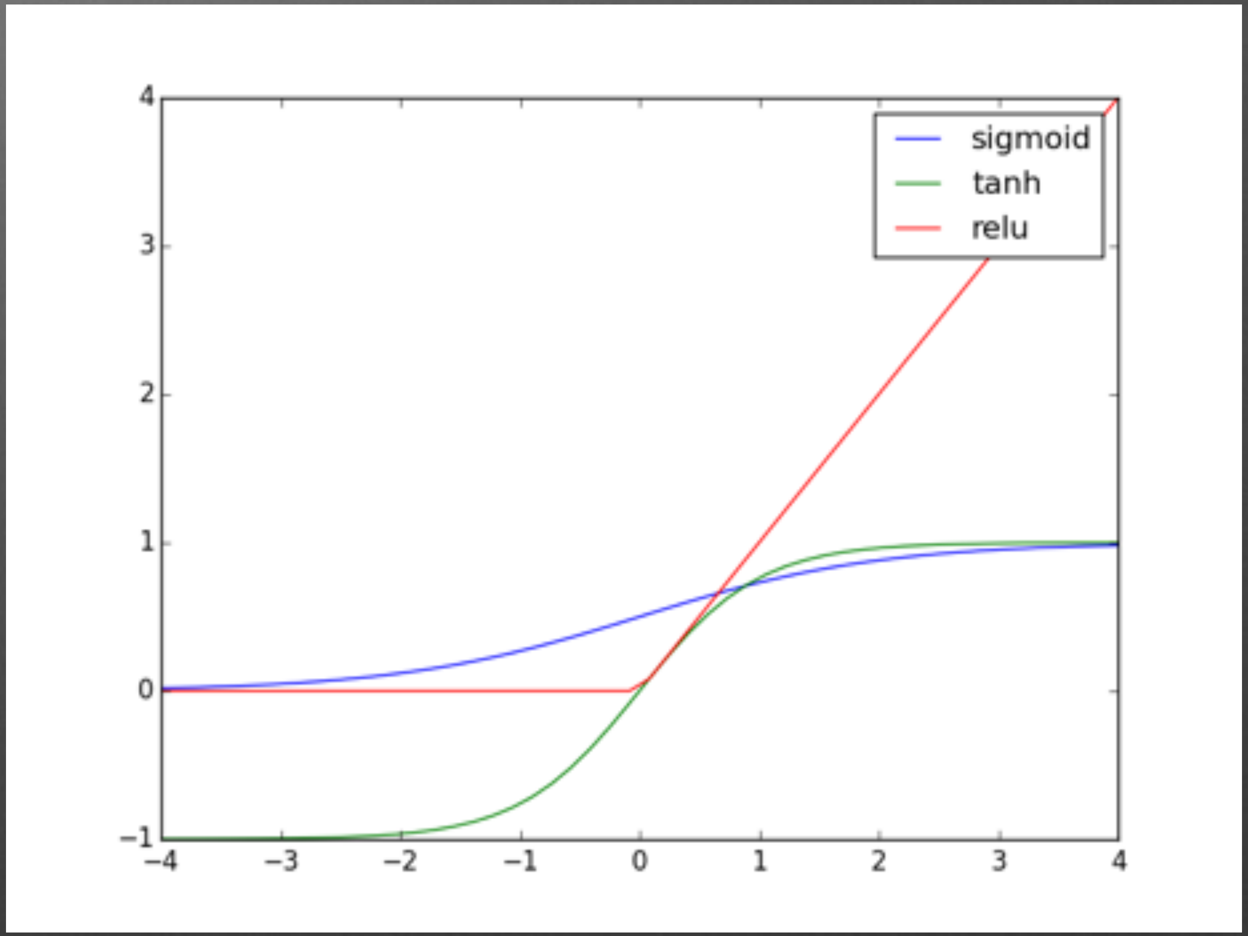
Appendix
Fully Connected Layer



$$WX + b = z$$

Appendix
Activation Functions

- Map to another dimension
  - z = WX + b
- Typically apply element-wise activation function
  - a = f(z)
  - sigmoid $f(x) = \frac{1}{1+e^{-x}}$
  - tanh $f(x) = tanh(x)$
  - relu $f(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ if } x \leq 0 \end{cases}$



Appendix
Gradients

Function	Equation	Derivative
sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$\frac{df}{dx} = f(x)(1 - f(x))$
tanh	$f(x) = tanh(x)$	$\frac{df}{dx} = 1 - f(x)^2$
relu	$f(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ if } x \leq 0 \end{cases}$	$\frac{df}{dx} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x \leq 0 \end{cases}$

Appendix
Batch normalization

1. Scale and shift batch to N(0, 1)

$$\hat{x} = \frac{x - \bar{x}}{\sigma}$$

2. Update running mean and variance for testing

$$\bar{x}_{\text{test}} = m\bar{x}_{\text{test}} + (1 - m)\bar{x}_{\text{batch}}$$

$$\sigma_{\text{test}} = m\sigma_{\text{test}} + (1 - m)\sigma_{\text{batch}}$$

3. Apply learnable scale and shift parameters beta and gamma

$$y = \gamma\hat{x} - \beta$$