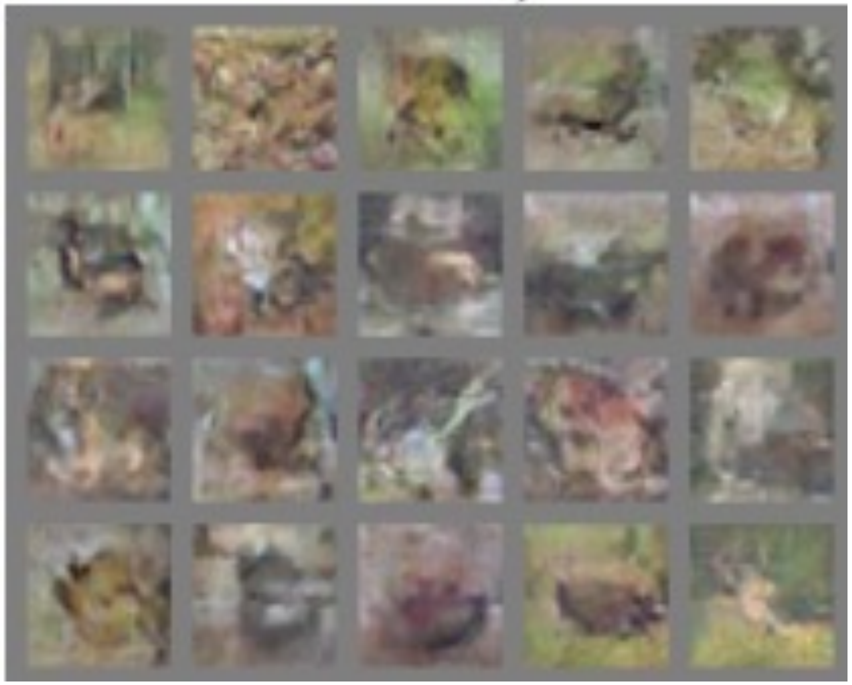
Deep Convolutional Generative Adversarial Networks (DCGAN)

David I. Inouye

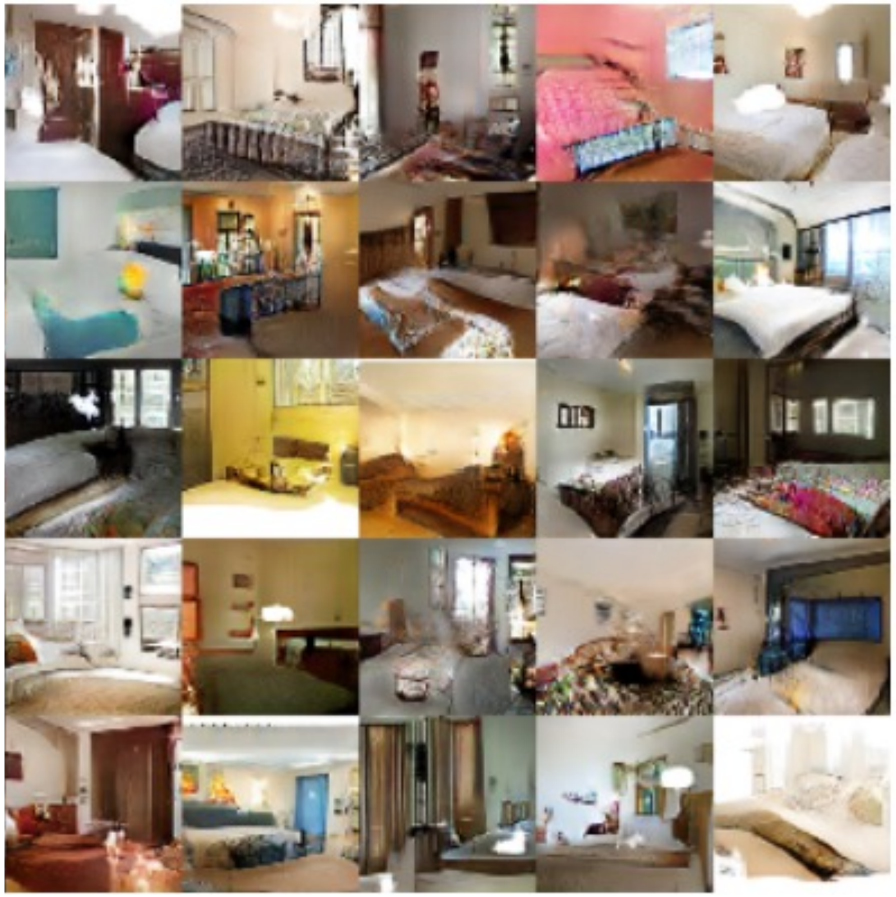
Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

DCGAN: Randomly generated bedrooms show slightly odd but almost realistic bedrooms

Original GAN (CIFAR10)

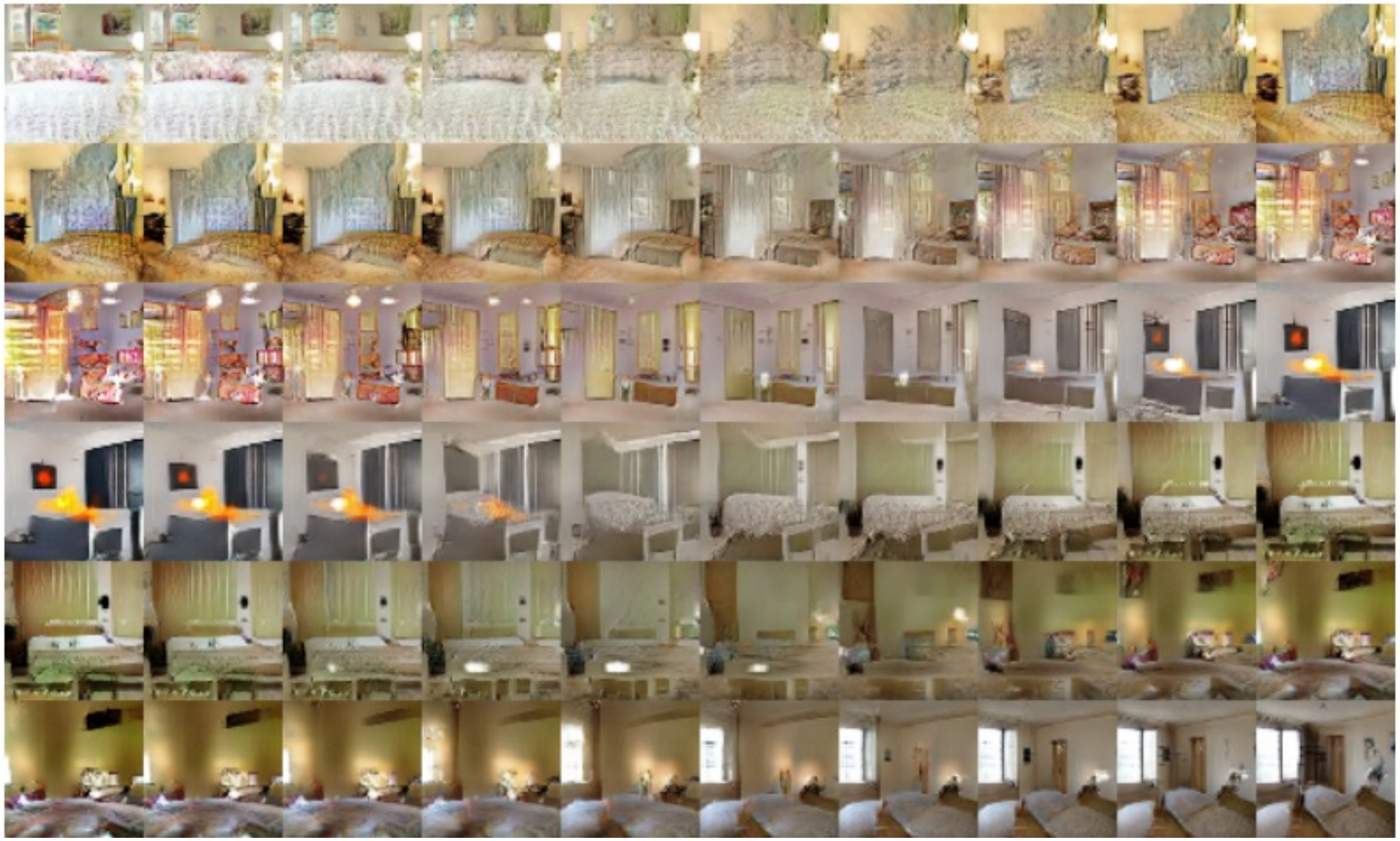


DCGAN (bedrooms)



Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

DCGAN can show interpolation between **imaginary** hotel rooms



Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Removing certain filters can modify the generated images (in this case, a "window" filter)

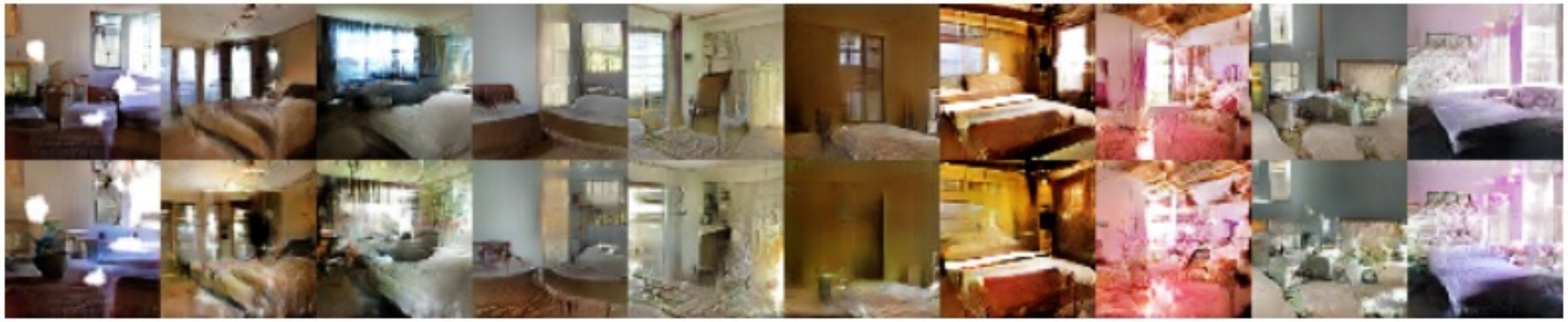
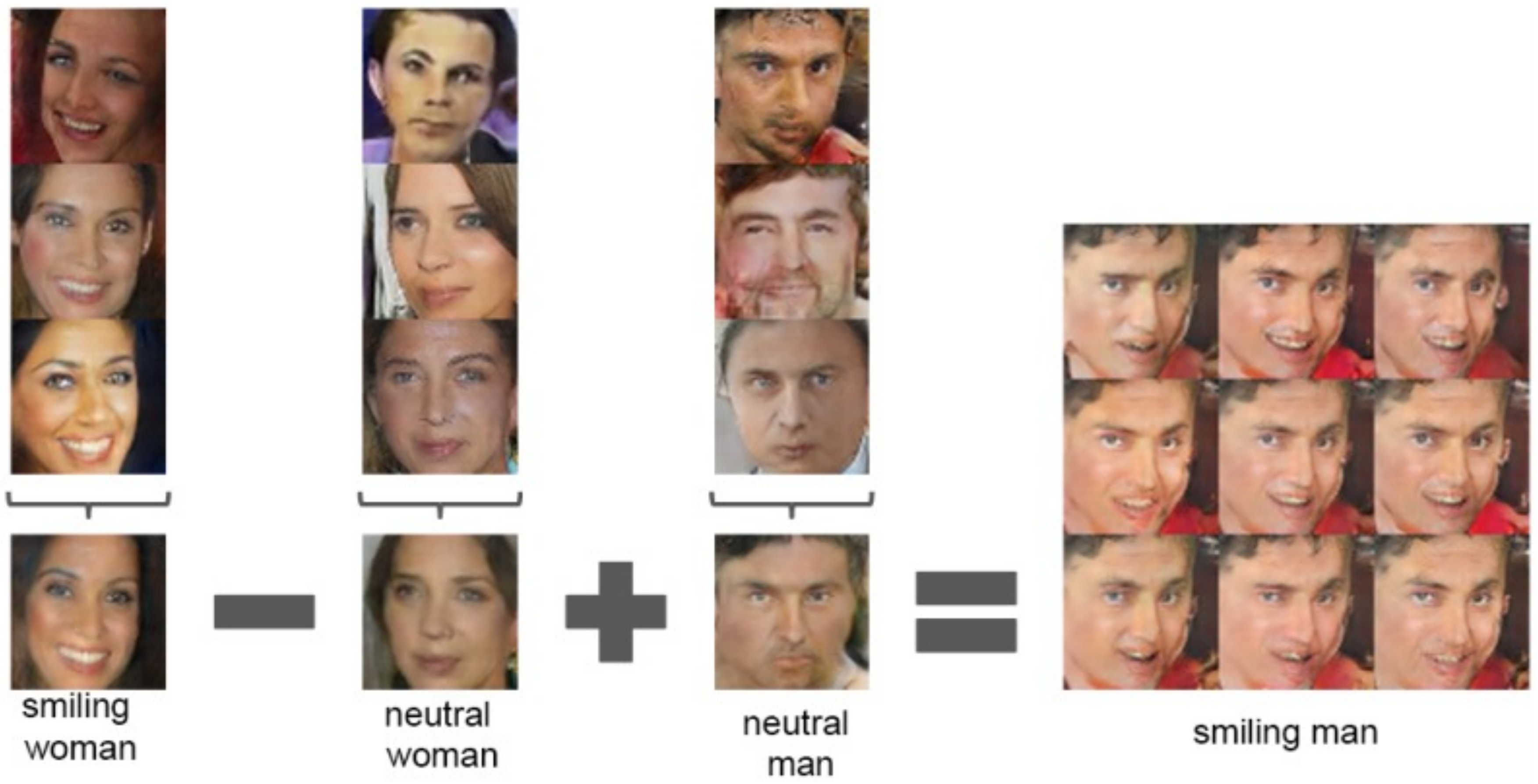


Figure 6: Top row: un-modified samples from model. Bottom row: the same samples generated with dropping out "window" filters. Some windows are removed, others are transformed into objects with similar visual appearance such as doors and mirrors. Although visual quality decreased, overall scene composition stayed similar, suggesting the generator has done a good job disentangling scene representation from object representation. Extended experiments could be done to remove other objects from the image and modify the objects the generator draws.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Simple vector arithmetic in **latent space** of DCGAN can generate new faces

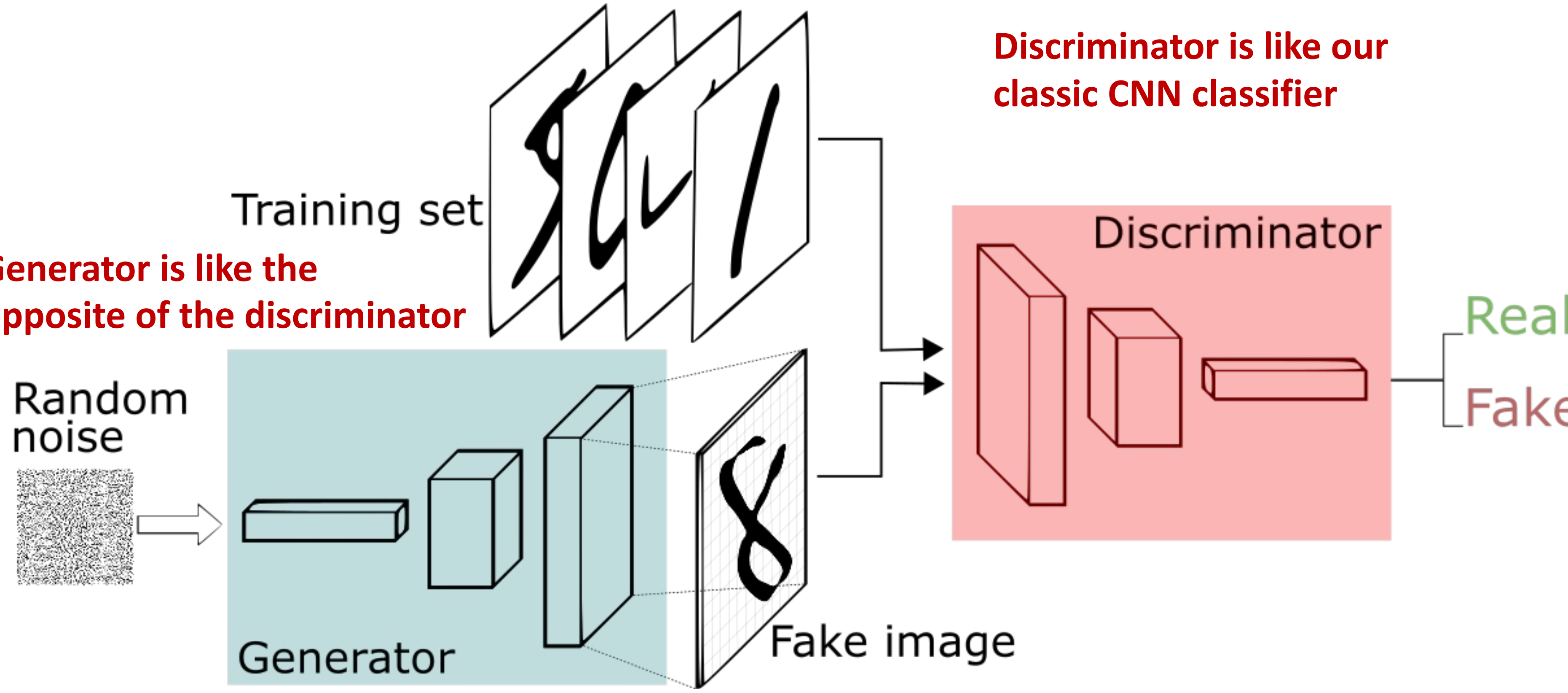


Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

DCGAN Architecture:
Two deep networks that look like opposites

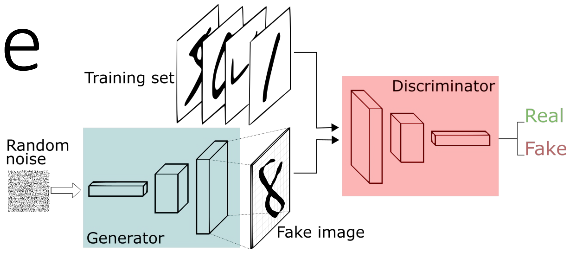
Discriminator is like our classic CNN classifier

Generator is like the opposite of the discriminator



https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/

DCGAN generator upsamples the size of the image in multiple stages



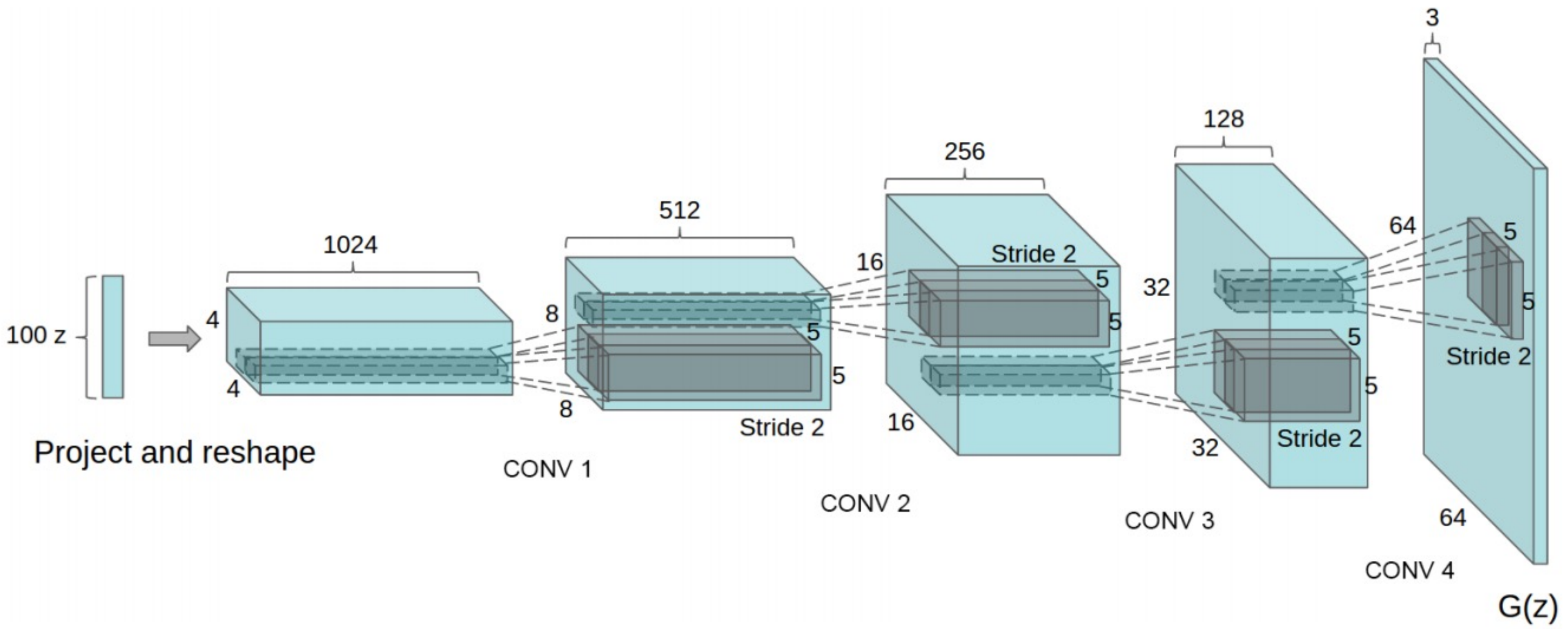
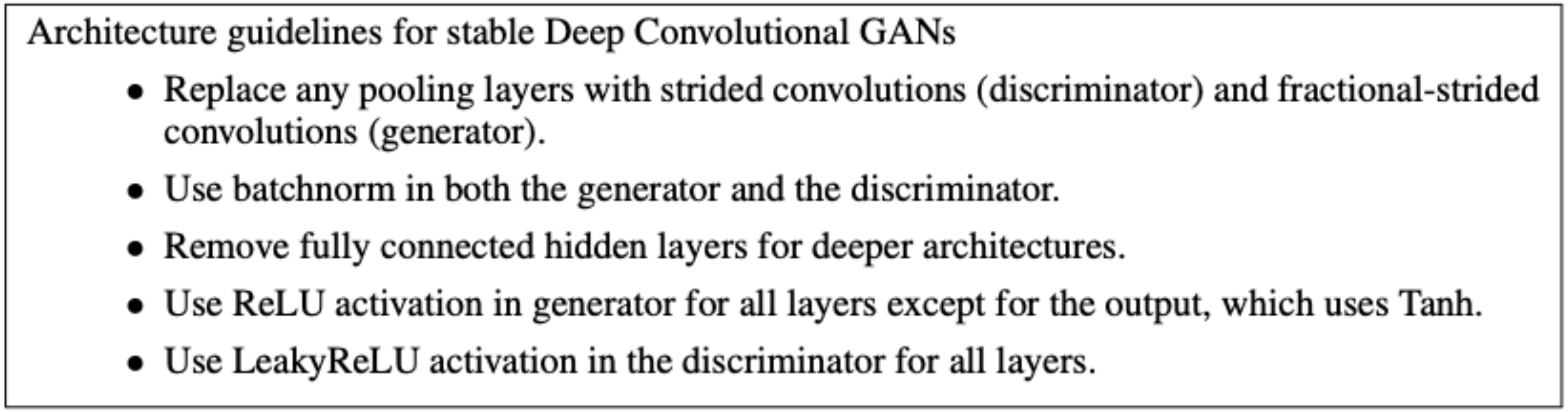


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

DCGAN requires transposed convolutions, BatchNorm, and a few other training tricks

- Transposed convolutions (for upsampling)
- BatchNorm (for stabilizing training)
- A few tricks


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.

Transposed convolution can be used to **upsample** an tensor/image to have higher dimensions

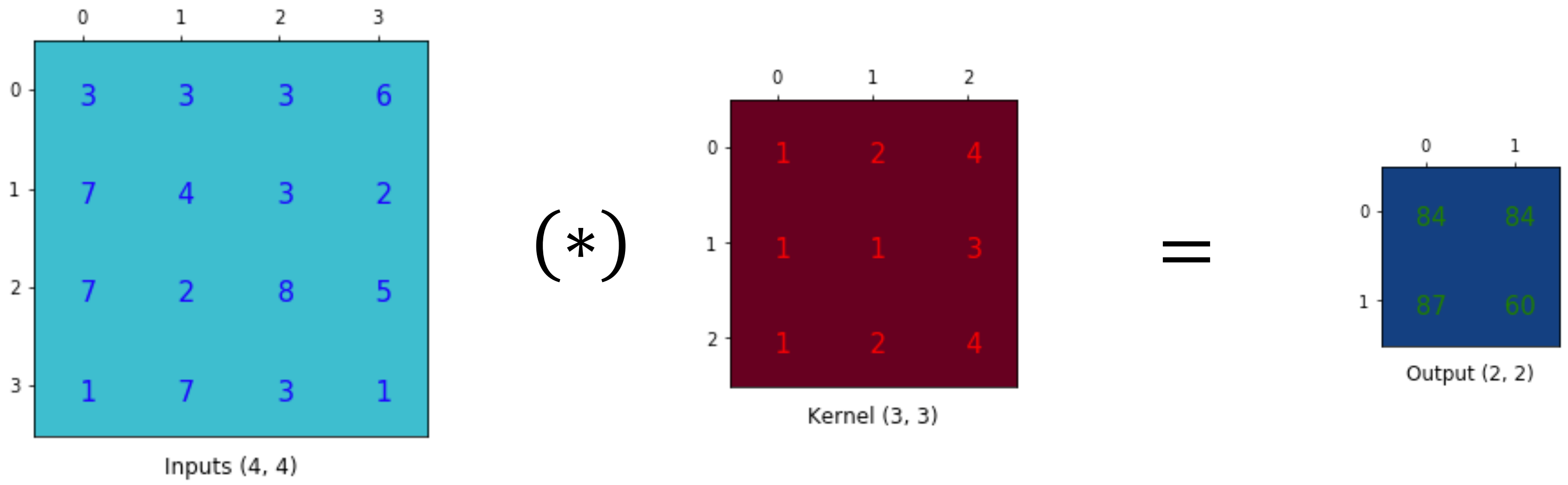
- Also known as:
  - Fractionally-strided convolution
  - Improperly, deconvolution
- Remember: Convolution is like matrix multiplication

$$y = x\ (*)\ f \Leftrightarrow \mathrm{vec}(y) = A_f \mathrm{vec}(x)$$

- Transpose convolution is the transpose of A_f:

$$\mathrm{vec}(y) = A_f^T \mathrm{vec}(x)$$

Convolution operator with corresponding matrix

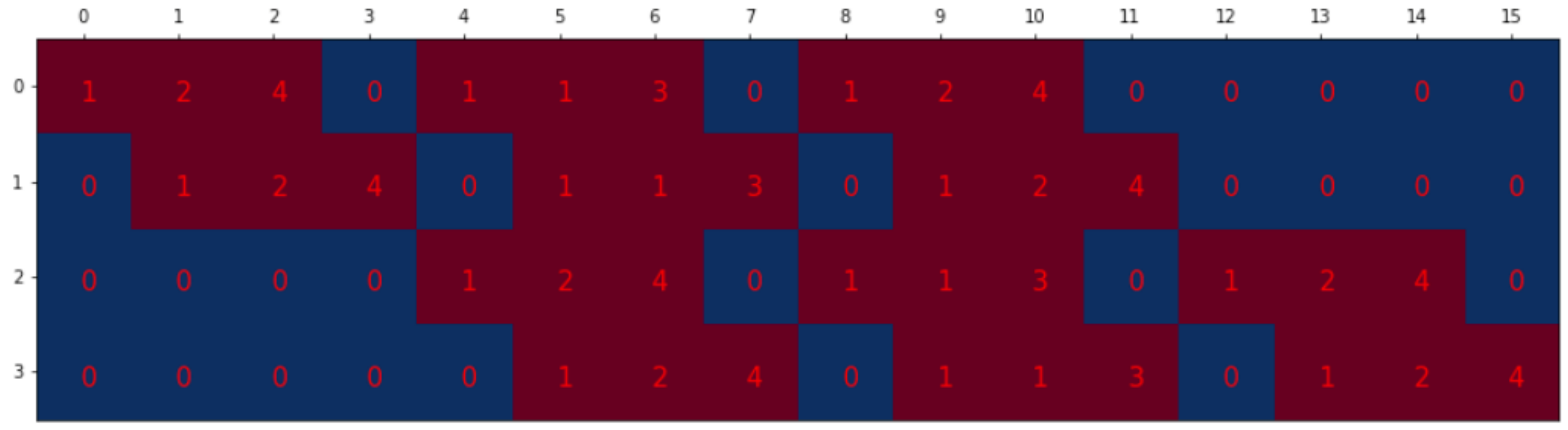


$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 3 & 3 & 3 & 6 \\ 1 & 7 & 4 & 3 & 2 \\ 2 & 7 & 2 & 8 & 5 \\ 3 & 1 & 7 & 3 & 1 \end{array} \circledast \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 4 \\ 1 & 1 & 1 & 3 \\ 2 & 1 & 2 & 4 \end{array} = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 84 & 84 \\ 1 & 87 & 60 \end{array}$$

Inputs (4, 4)

Kernel (3, 3)

Output (2, 2)

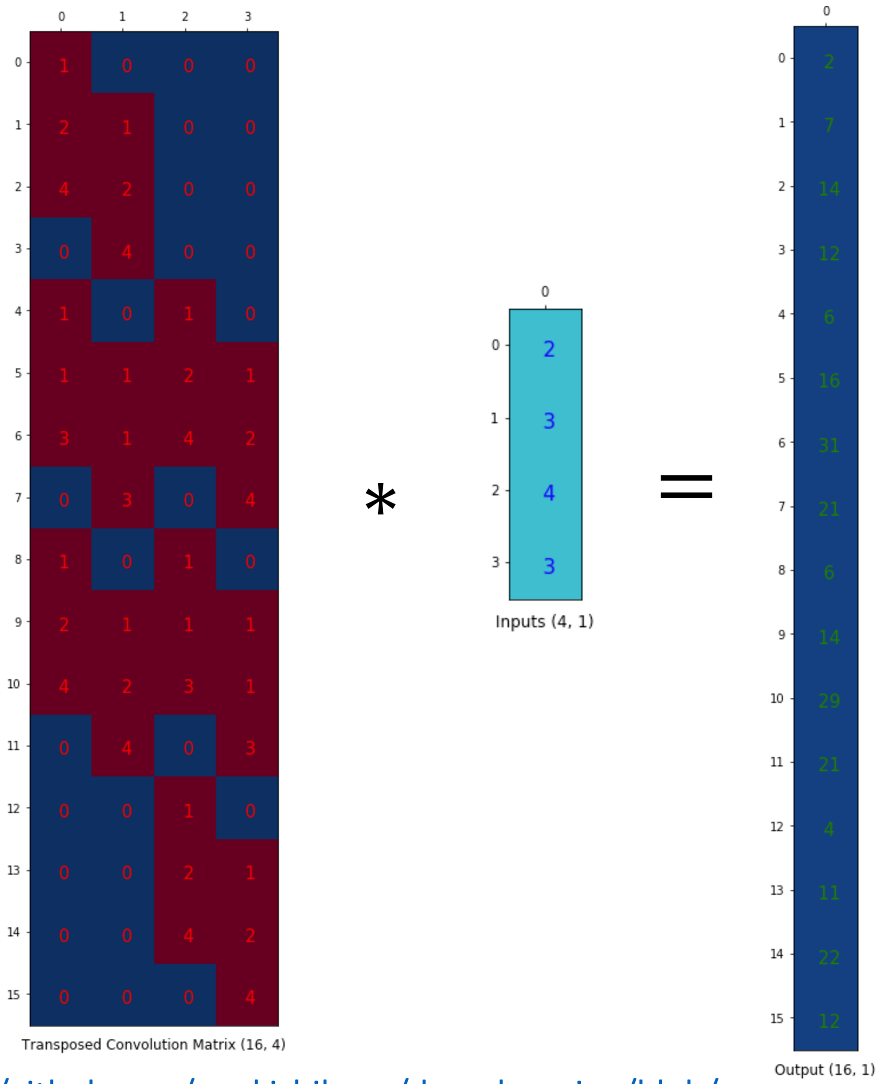


$$\begin{array}{c|cccccccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \hline 0 & 1 & 2 & 4 & 0 & 1 & 1 & 3 & 0 & 1 & 2 & 4 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 4 & 0 & 1 & 1 & 3 & 0 & 1 & 2 & 4 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 0 & 1 & 1 & 3 & 0 & 1 & 2 & 4 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 0 & 1 & 1 & 3 & 0 & 1 & 2 & 4 \end{array}$$

Convolution Matrix (4, 16)

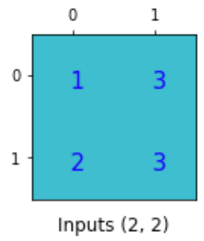
https://github.com/naokishibuya/deep-learning/blob/master/python/transposed_convolution.ipynb

Transposed convolution operator with corresponding matrix



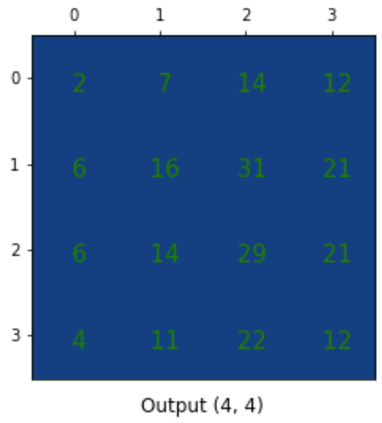
$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 1 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 0 \\
2 & 4 & 2 & 0 & 0 \\
3 & 0 & 4 & 0 & 0 \\
4 & 1 & 0 & 1 & 0 \\
5 & 1 & 1 & 2 & 1 \\
6 & 3 & 1 & 4 & 2 \\
7 & 0 & 3 & 0 & 4 \\
8 & 1 & 0 & 1 & 0 \\
9 & 2 & 1 & 1 & 1 \\
10 & 4 & 2 & 3 & 1 \\
11 & 0 & 4 & 0 & 3 \\
12 & 0 & 0 & 1 & 0 \\
13 & 0 & 0 & 2 & 1 \\
14 & 0 & 0 & 4 & 2 \\
15 & 0 & 0 & 0 & 4
\end{array}
*
\begin{array}{c|c}
 & 0 \\
\hline
0 & 2 \\
1 & 3 \\
2 & 4 \\
3 & 3
\end{array}
=
\begin{array}{c|c}
 & 0 \\
\hline
0 & 2 \\
1 & 7 \\
2 & 14 \\
3 & 12 \\
4 & 6 \\
5 & 16 \\
6 & 31 \\
7 & 21 \\
8 & 6 \\
9 & 14 \\
10 & 29 \\
11 & 21 \\
12 & 4 \\
13 & 11 \\
14 & 22 \\
15 & 12
\end{array}
$$

Reshaped input



$$
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
0 & 1 & 3 \\
1 & 2 & 3
\end{array}
$$

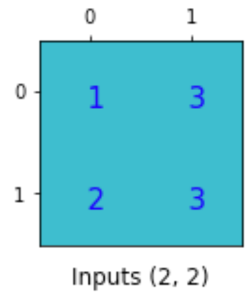
Reshaped output



$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 2 & 7 & 14 & 12 \\
1 & 6 & 16 & 31 & 21 \\
2 & 6 & 14 & 29 & 21 \\
3 & 4 & 11 & 22 & 12
\end{array}
$$

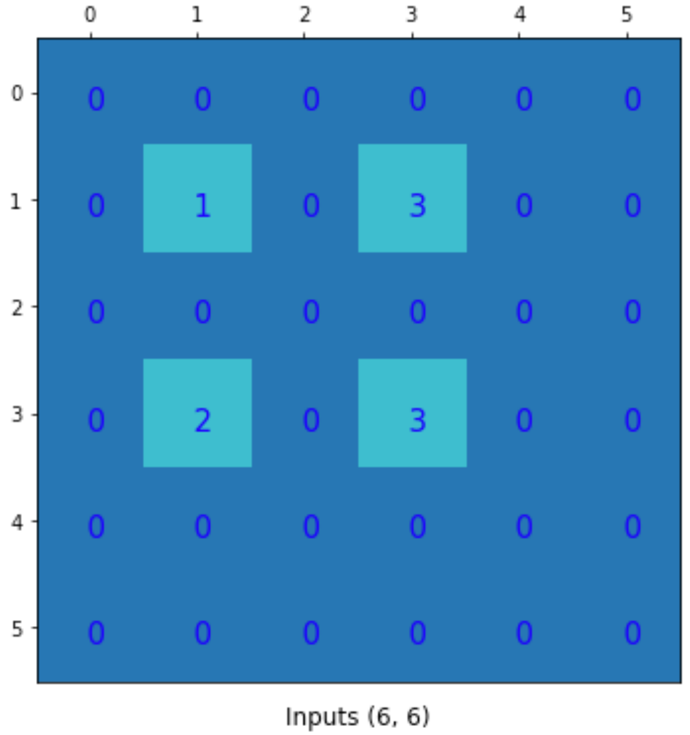
https://github.com/naokishibuya/deep-learning/blob/master/python/transposed_convolution.ipynb

Transposed convolution can be **equivalent** to a simple convolution with zero rows/columns added (added zeros simulate fractional strides)



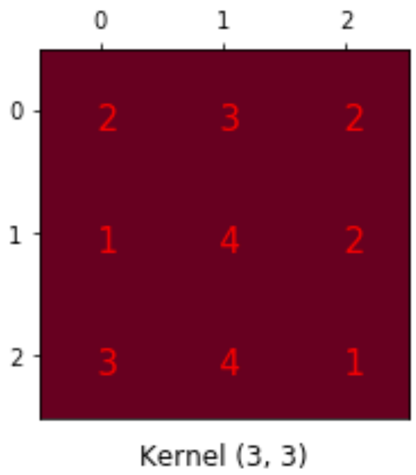
Original input

Zero-padded input

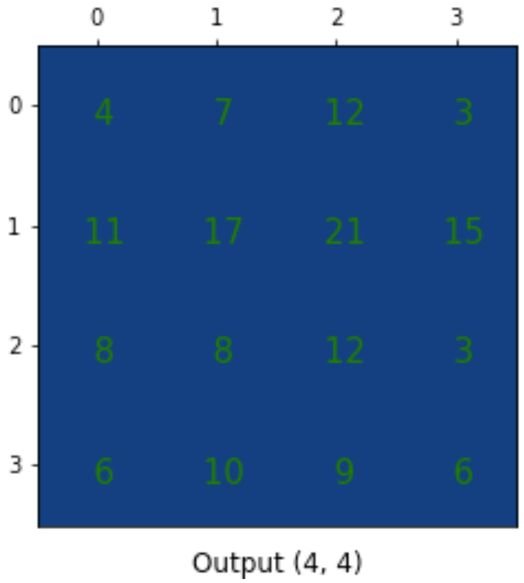


$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 3 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 2 & 0 & 3 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (*) \quad \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 2 & 3 & 2 \\ 1 & 1 & 4 & 2 \\ 2 & 3 & 4 & 1 \end{array} \quad = \quad \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 4 & 7 & 12 & 3 \\ 1 & 11 & 17 & 21 & 15 \\ 2 & 8 & 8 & 12 & 3 \\ 3 & 6 & 10 & 9 & 6 \end{array}$$

Kernel



Output



Two GAN training stability tricks

1. Do backwards (gradient calc) separately for fake and real batches because of BatchNorm layers
   $$V(D,G) = \mathbb{E}_{p_{data}}[\log D(x)] + \mathbb{E}_{\epsilon}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$
   - Compute $\nabla_D \mathbb{E}_{p_{data}}[\log D(x)]$ and $\nabla_D \mathbb{E}_{\epsilon}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$ **separately**
2. Use modified objective for generator training from original GAN paper
   $$\nabla_G \mathbb{E}_{\epsilon}\left[-\log D\big(G(z)\big)\right] \text{ instead of } \nabla_G \mathbb{E}_{\epsilon}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

- See DCGAN demo on MNIST

Resources for GANs

- DCGAN Tutorial
  https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
- GAN training tips/hacks
  - https://github.com/soumith/ganhacks
- GAN common problems
  - https://developers.google.com/machine-learning/gan/problems