Deep Convolutional Generative Adversarial Networks (DCGAN)

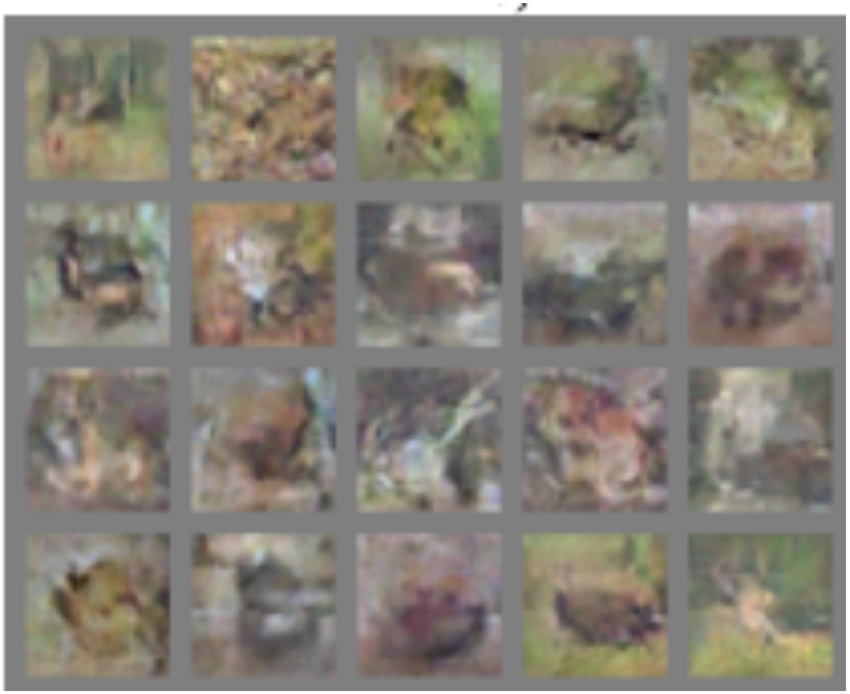
ECE57000: Artificial Intelligence

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

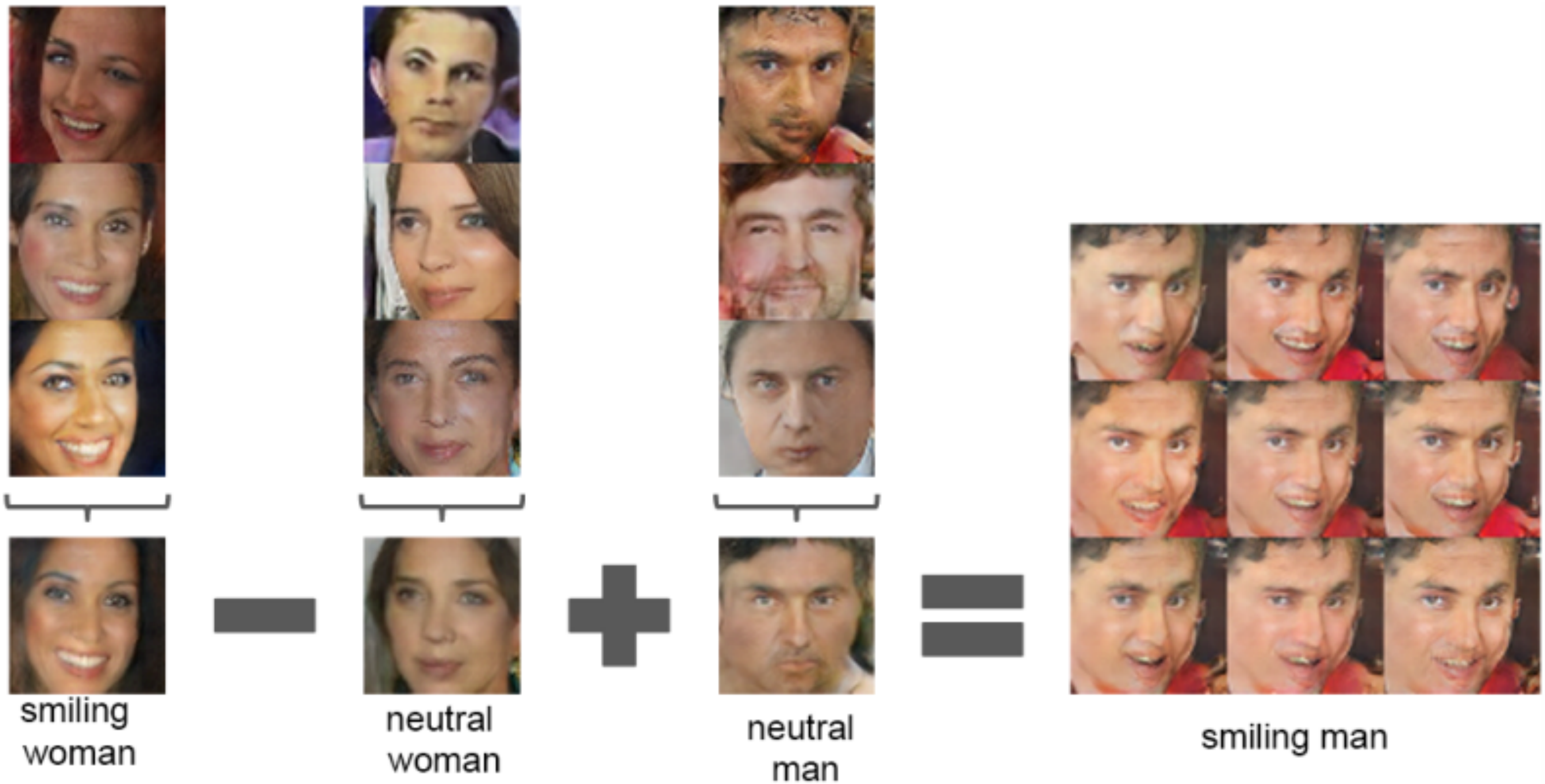
Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

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.

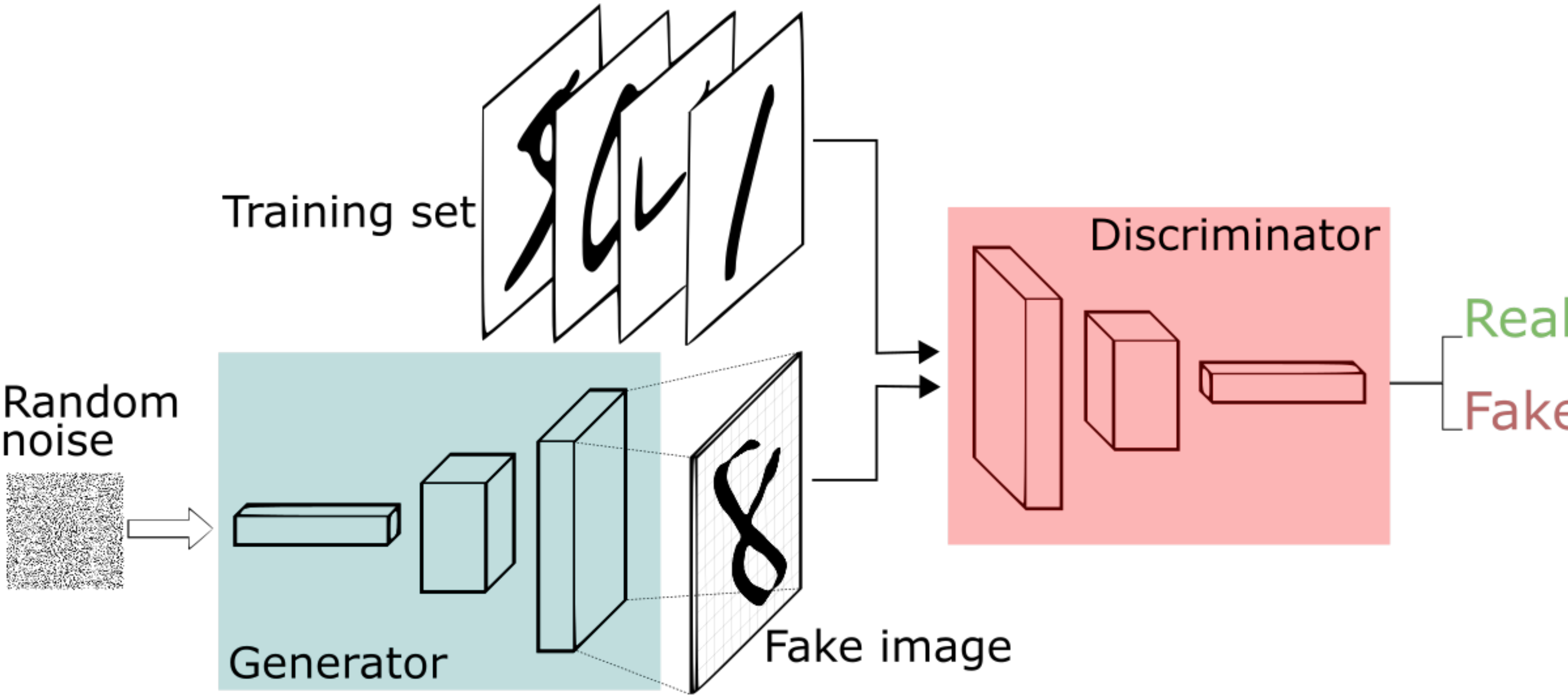
Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

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*.

How do we learn this implicit generative model?
Train two deep networks simultaneously



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

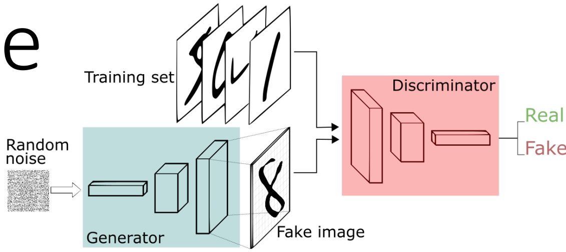
DCGAN requires transposed convolutions, BatchNorm, and a few other training tricks

- Transposed convolutions (for upsampling)

- BatchNorm (for stabilizing training)

- A few tricks

DCGAN generator upsamples the size of the image in multiple stages



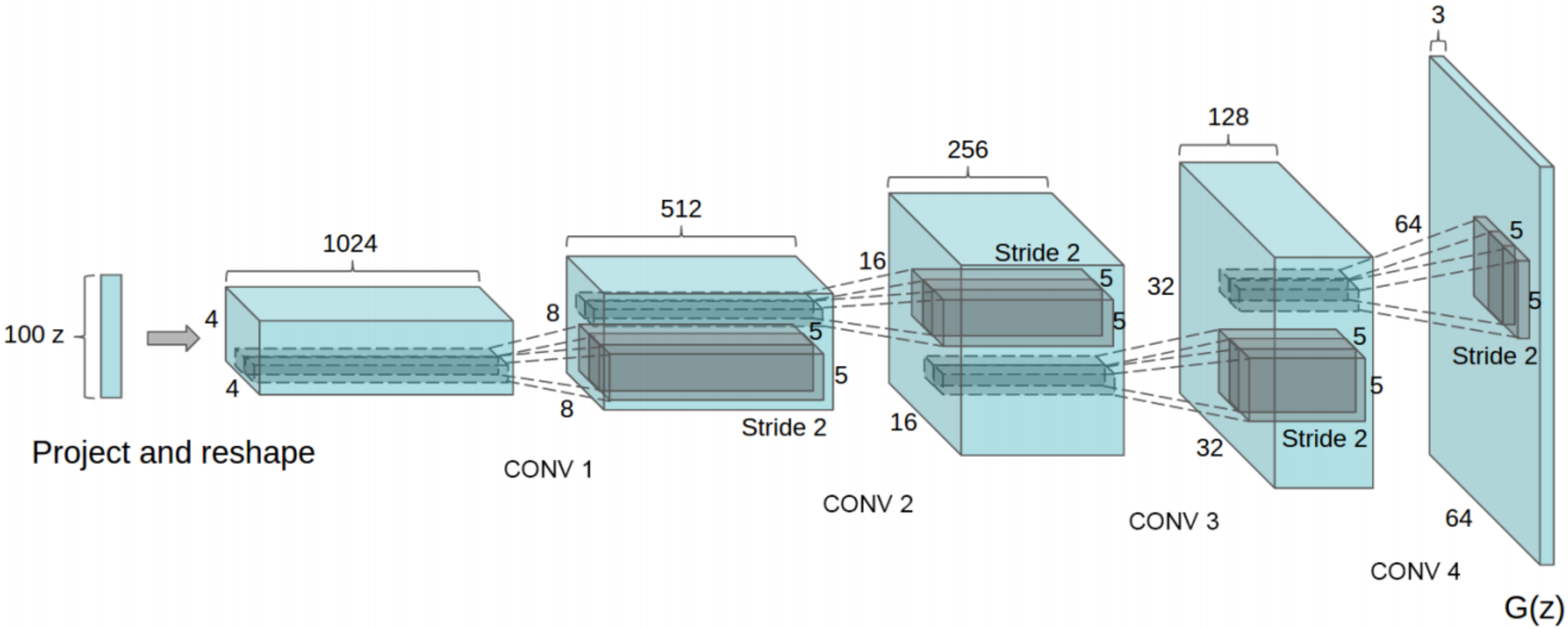


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

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

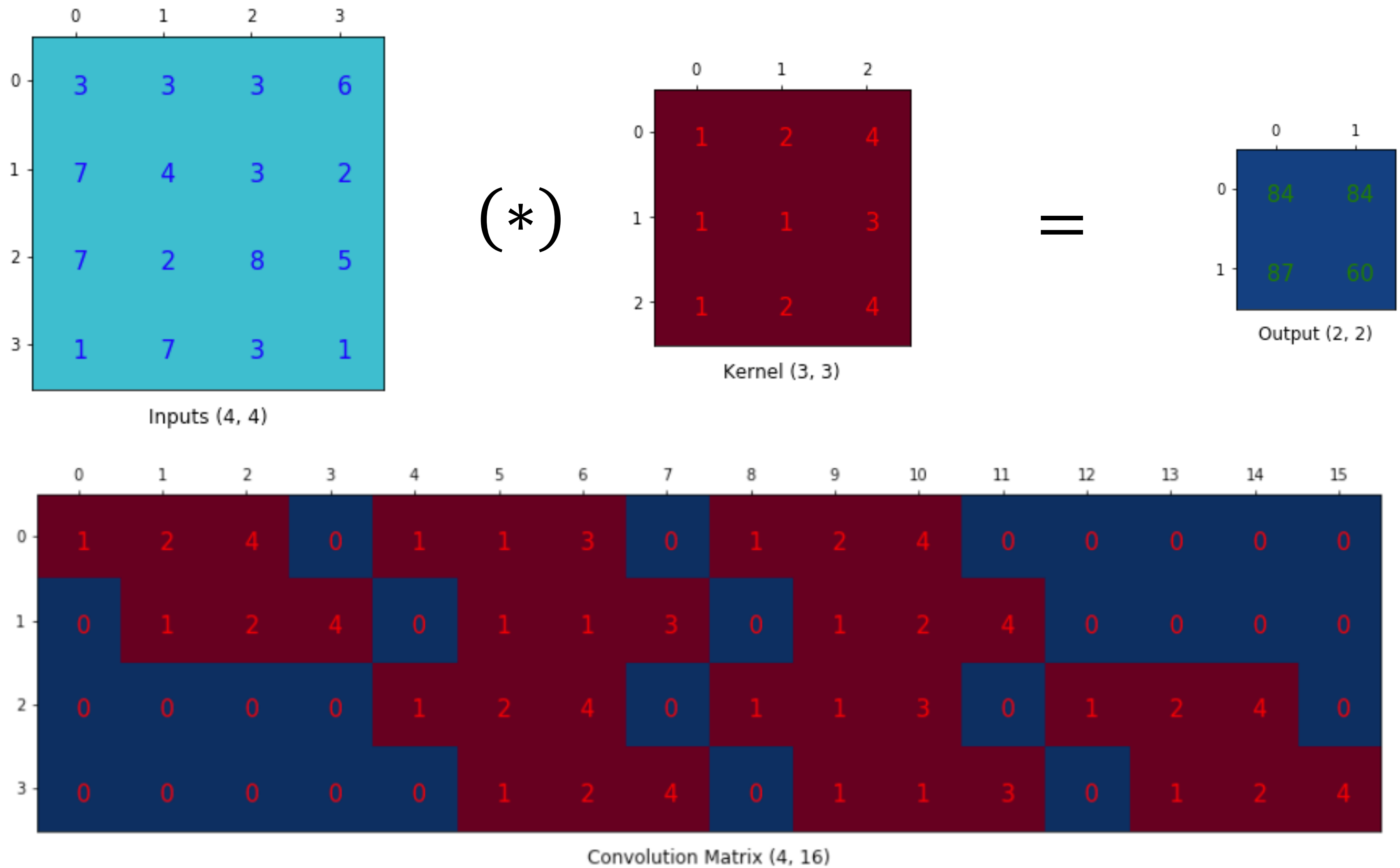
- Also known as:
  - Fractionally-strided convolution
  - Improperly, deconvolution
- Remember: Convolution is like matrix multiplication

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

- Transpose convolution is the transpose of A_f:

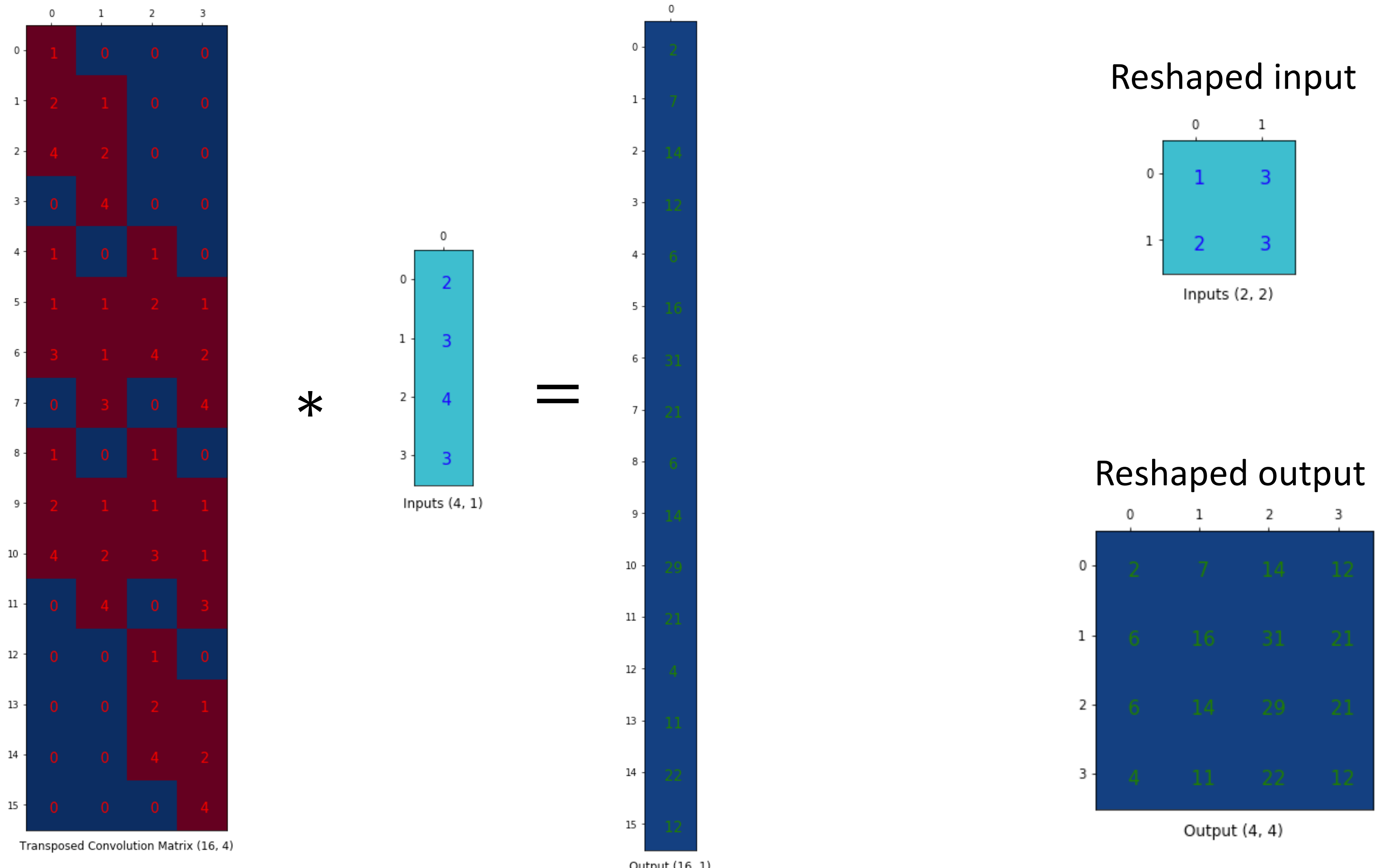
$$\mathrm{vec}(y) = A_f^T \mathrm{vec}(x)$$

Convolution operator with corresponding matrix



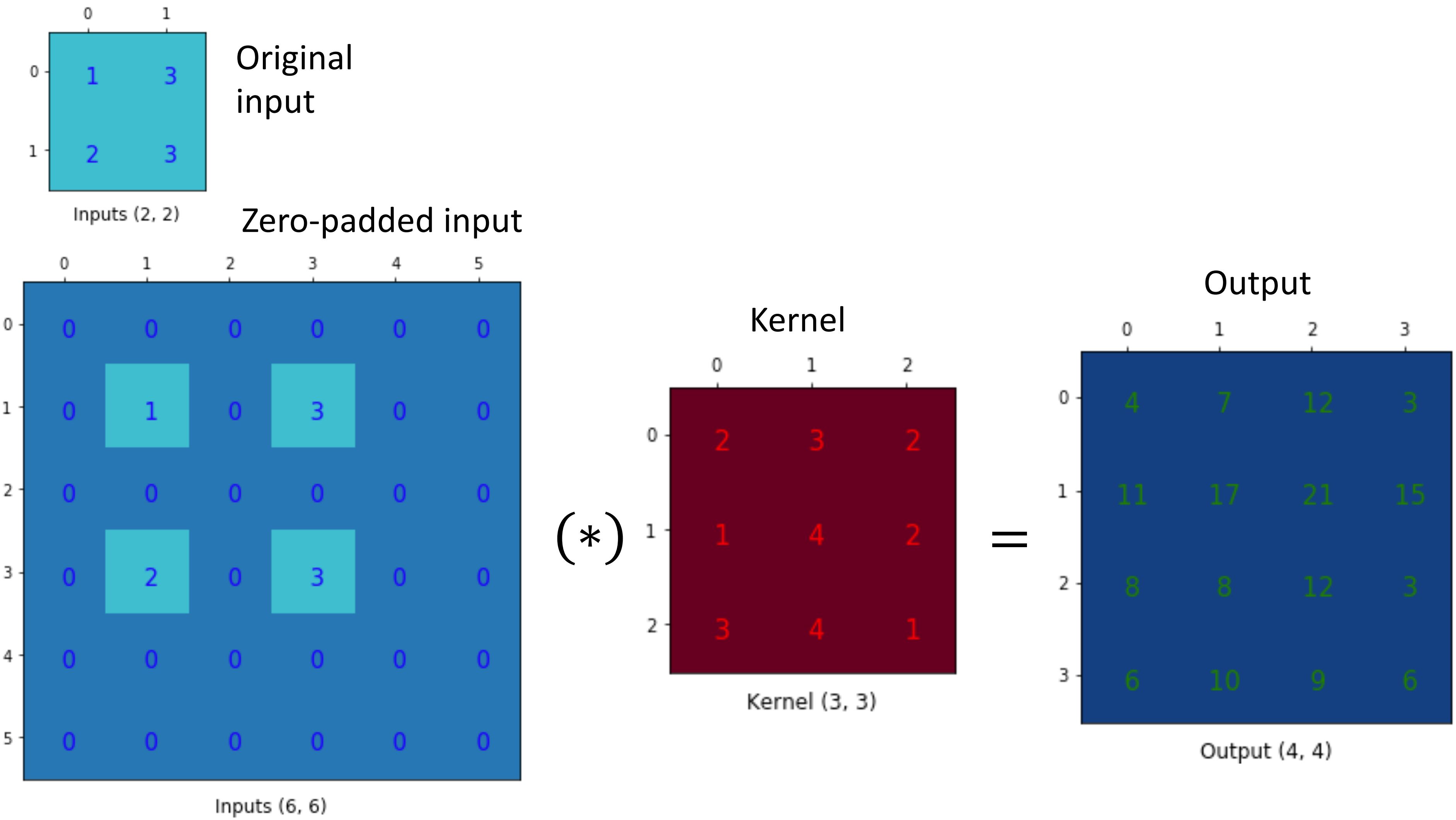
https://github.com/naokishibuya/deep-learning/blob/master/python/transposed_convolution.ipynb

Transposed convolution operator with corresponding matrix



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)



BatchNorm dynamically normalizes each feature to have zero mean and unit variance

- Normalize input batch to each layer after **every update**
  1. Input is minibatch of data X^t at iteration t
  2. Compute mean and standard deviation for every feature

$$\mu_j^t, \sigma_j^t \; \forall j \in \{1, \cdots, d\}$$

  3. Normalize each feature (note **different for every batch**)

$$\tilde{x}_j^t = \frac{\left(x_j^t - \mu_j^t\right)}{\sigma_j^t}$$

  4. Output $\tilde{X}^t$

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems* (pp. 2483-2493).

<u>BatchNorm dynamically</u> normalizes each feature to have zero mean and unit variance

- Only normalize batches during training (`model.train()`)
- **<u>Turn off</u>** after training (`model.eval()`)
  - Use running average of mean and variance

$$\mu_{run}^{t} = \lambda\mu_{run}^{t-1} + (1-\lambda)\mu_{batch}^{t}$$

$$\sigma^{2}{}_{run}^{t} = \lambda\sigma^{2}{}_{run}^{t-1} + (1-\lambda)\sigma^{2}{}_{batch}^{t}$$

- Surprisingly effective at stabilizing training, reducing training time, and producing better models
- Not fully understood why it works

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems* (pp. 2483-2493).

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