Convolutional Neural Networks

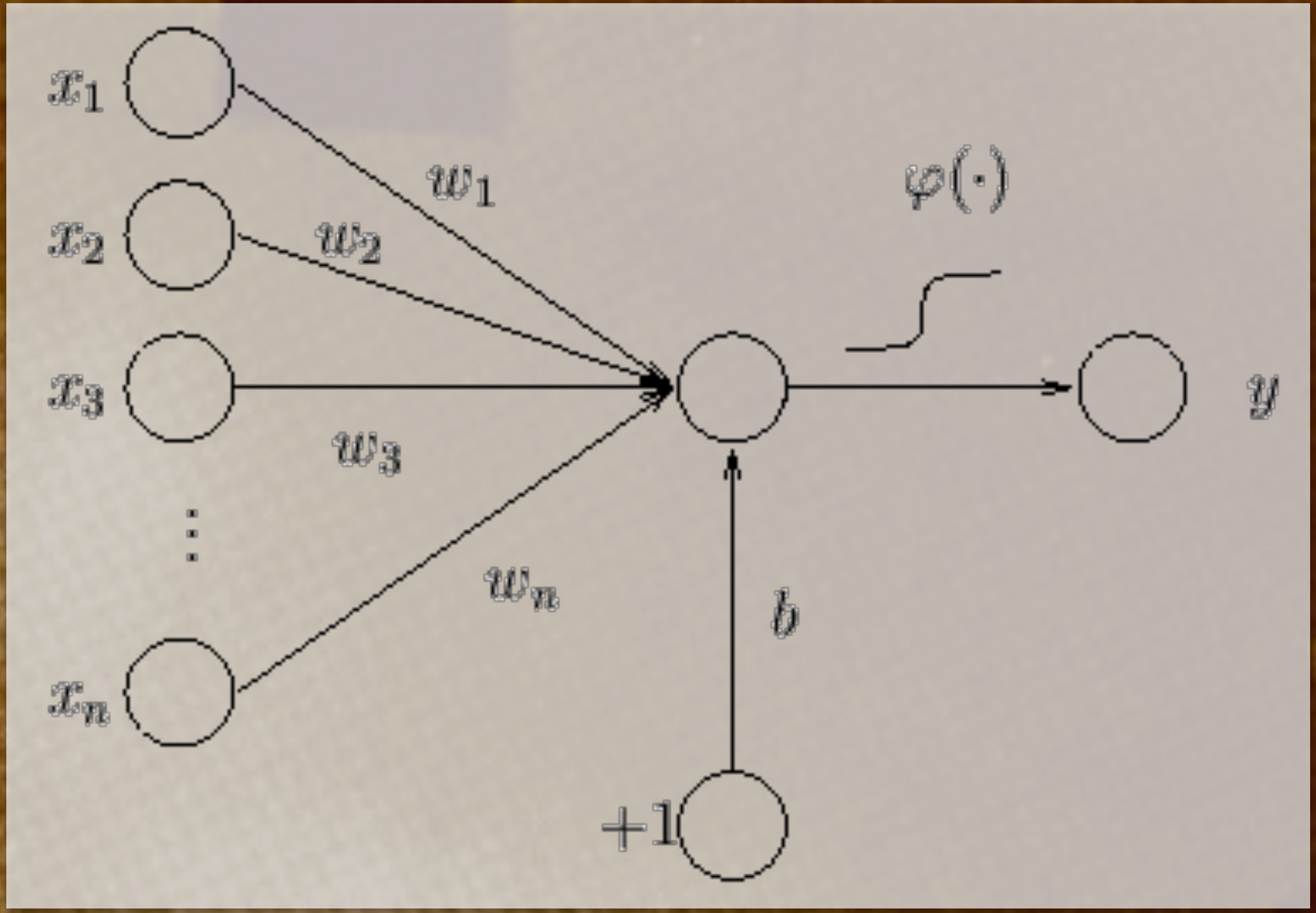
Jonathon Hare

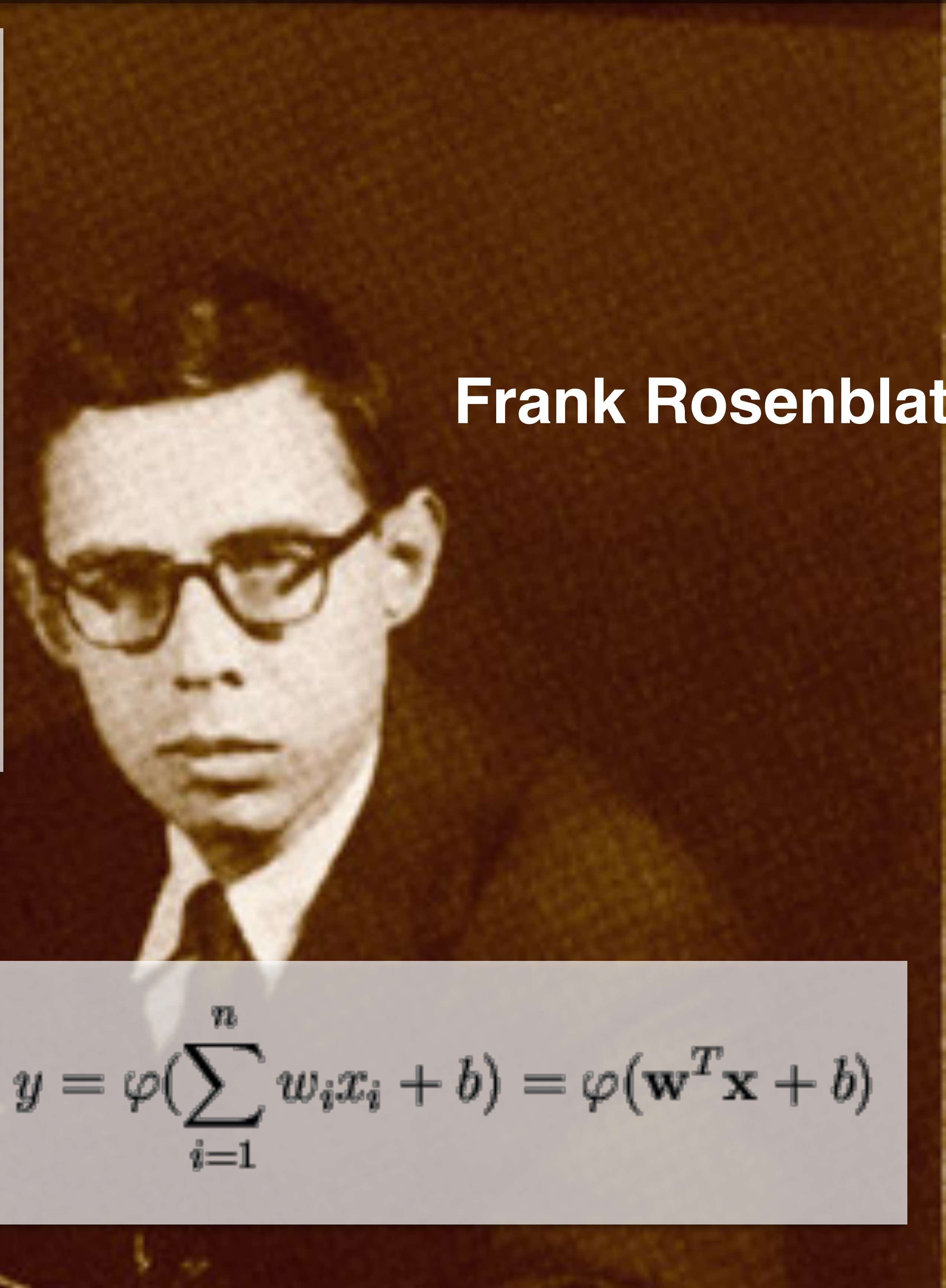
Vision, Learning and Control
University of Southampton

Many pictures used here are from https://github.com/vdumoulin/conv_arithmetic

a little history of
VISION

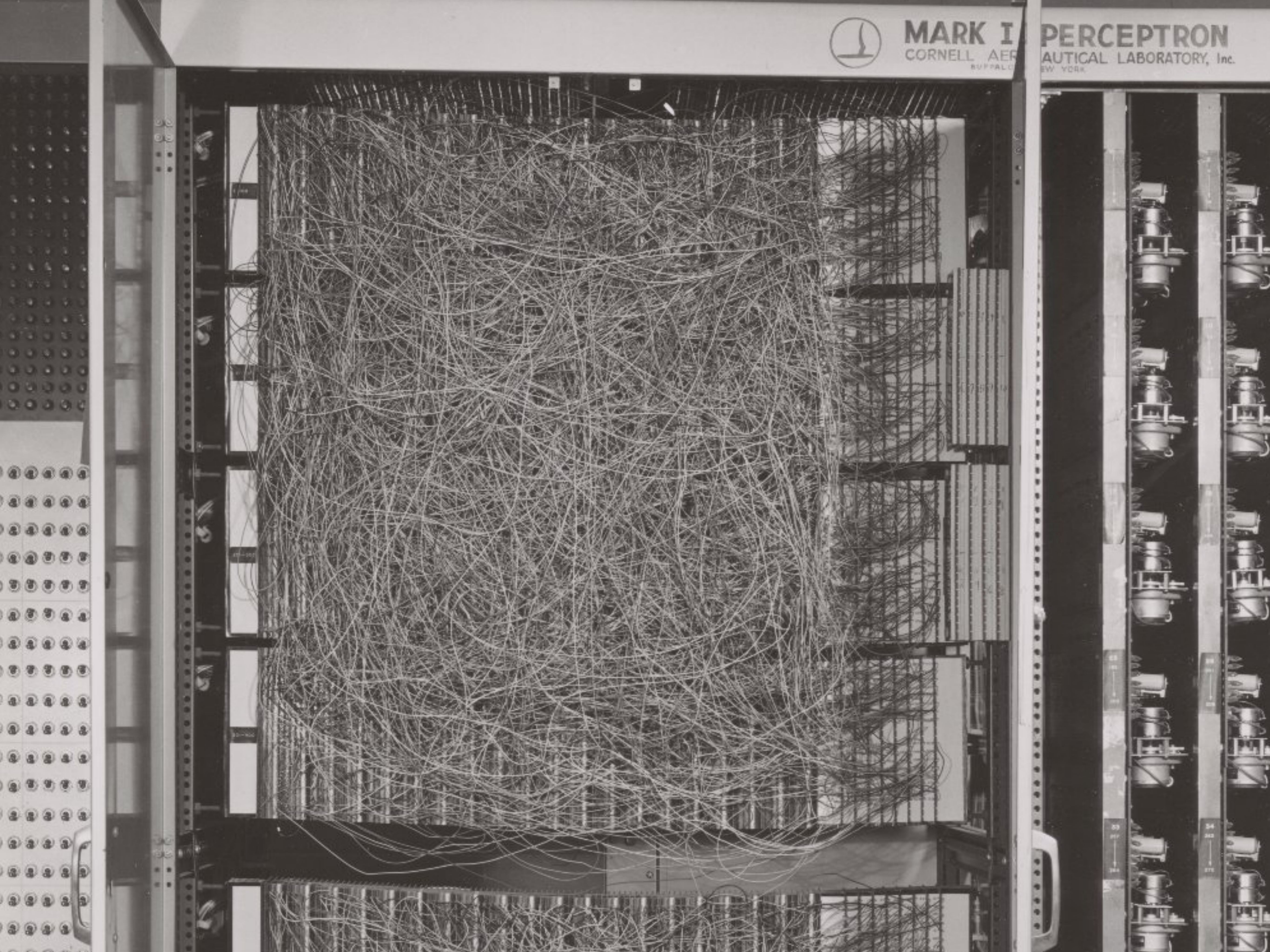
1958
Rosenblatt's Perceptron





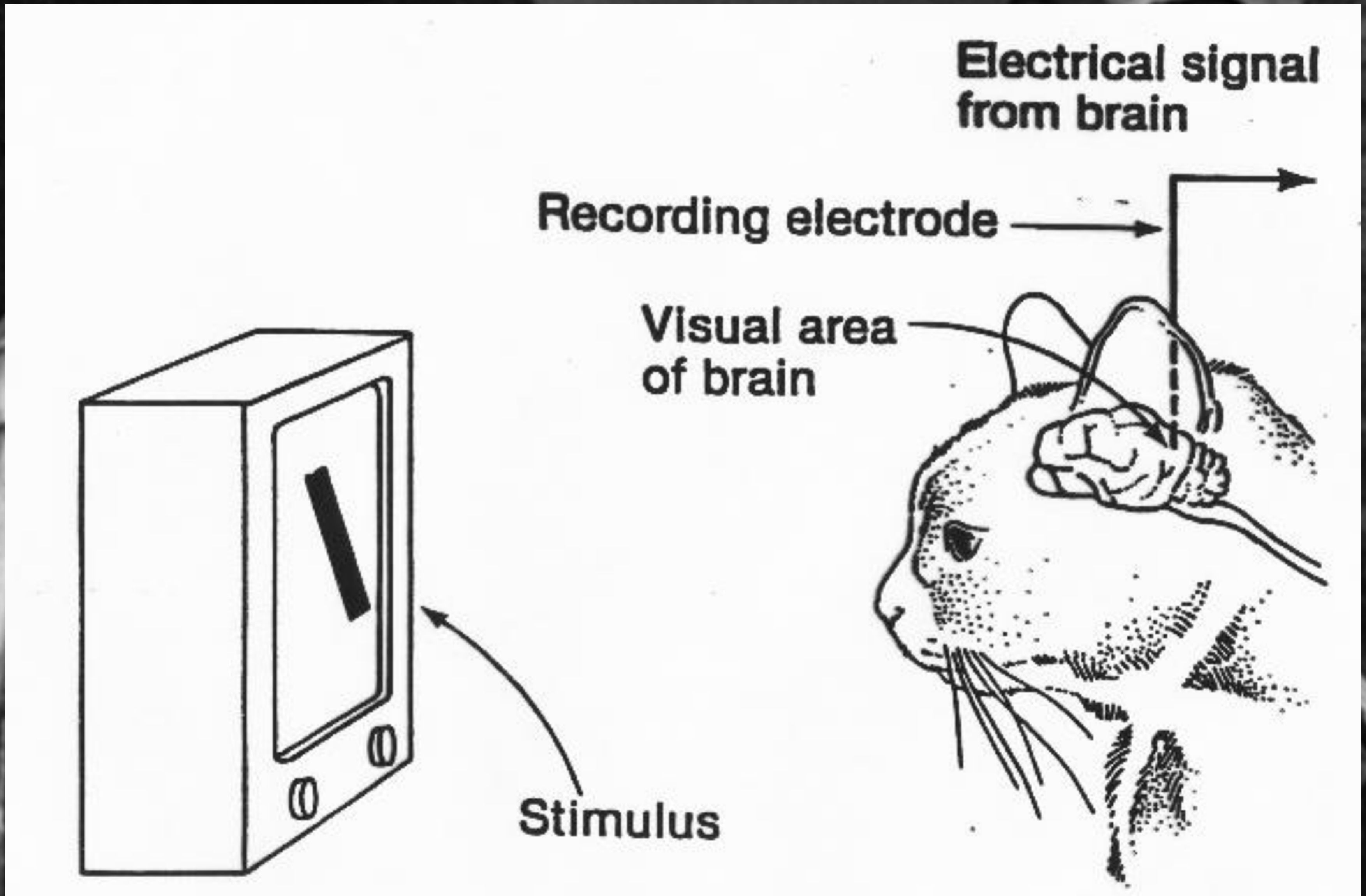
Frank Rosenblat

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi(\mathbf{w}^T \mathbf{x} + b)$$


MARK I PERCEPTRON
CORNELL AER AUTICAL LABORATORY, Inc.
BUFFALO EW YORK

1959

Receptive Fields of Single Neurons in the Cat's Striate Cortex

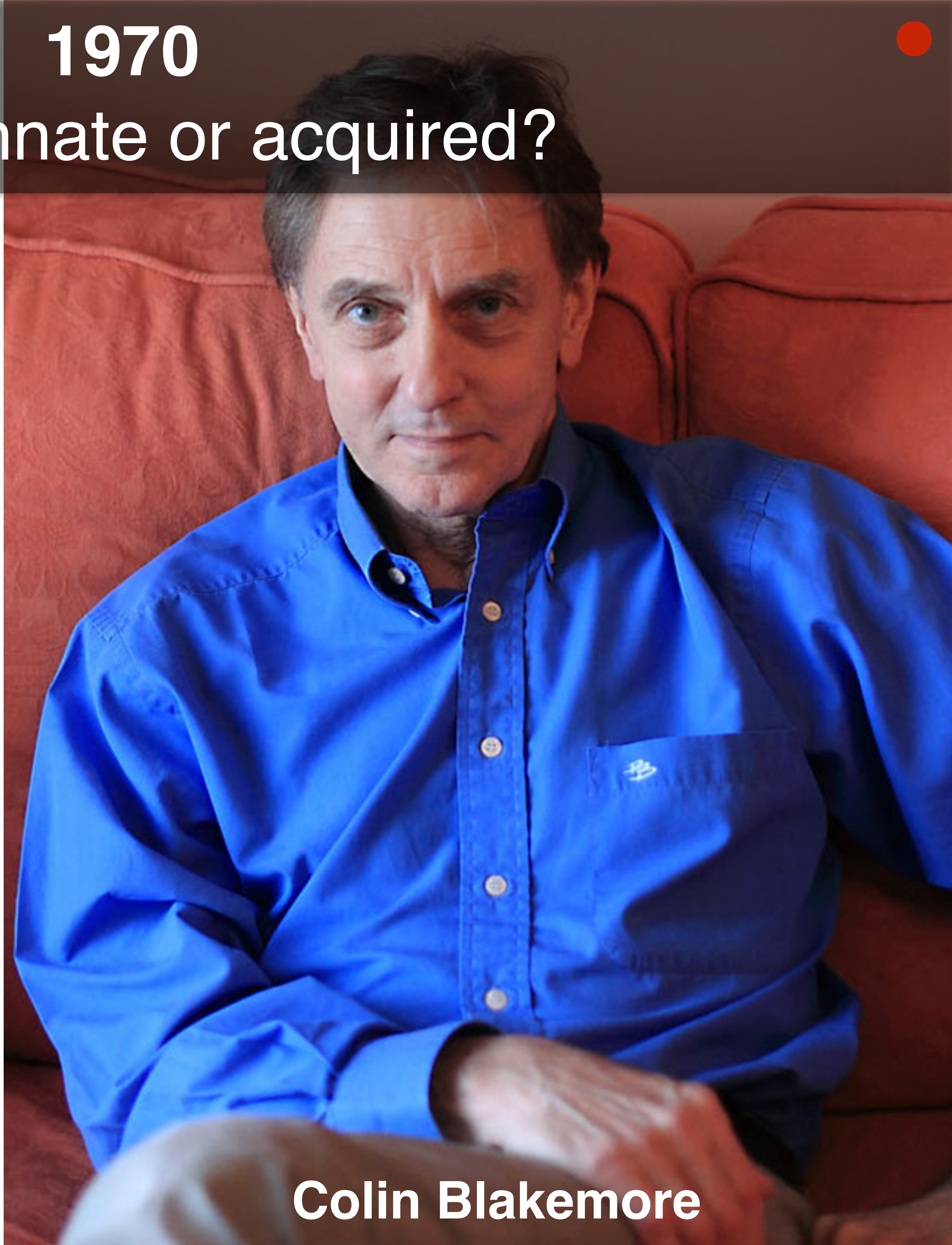


David Hubel

Torsten Wiesel

1970

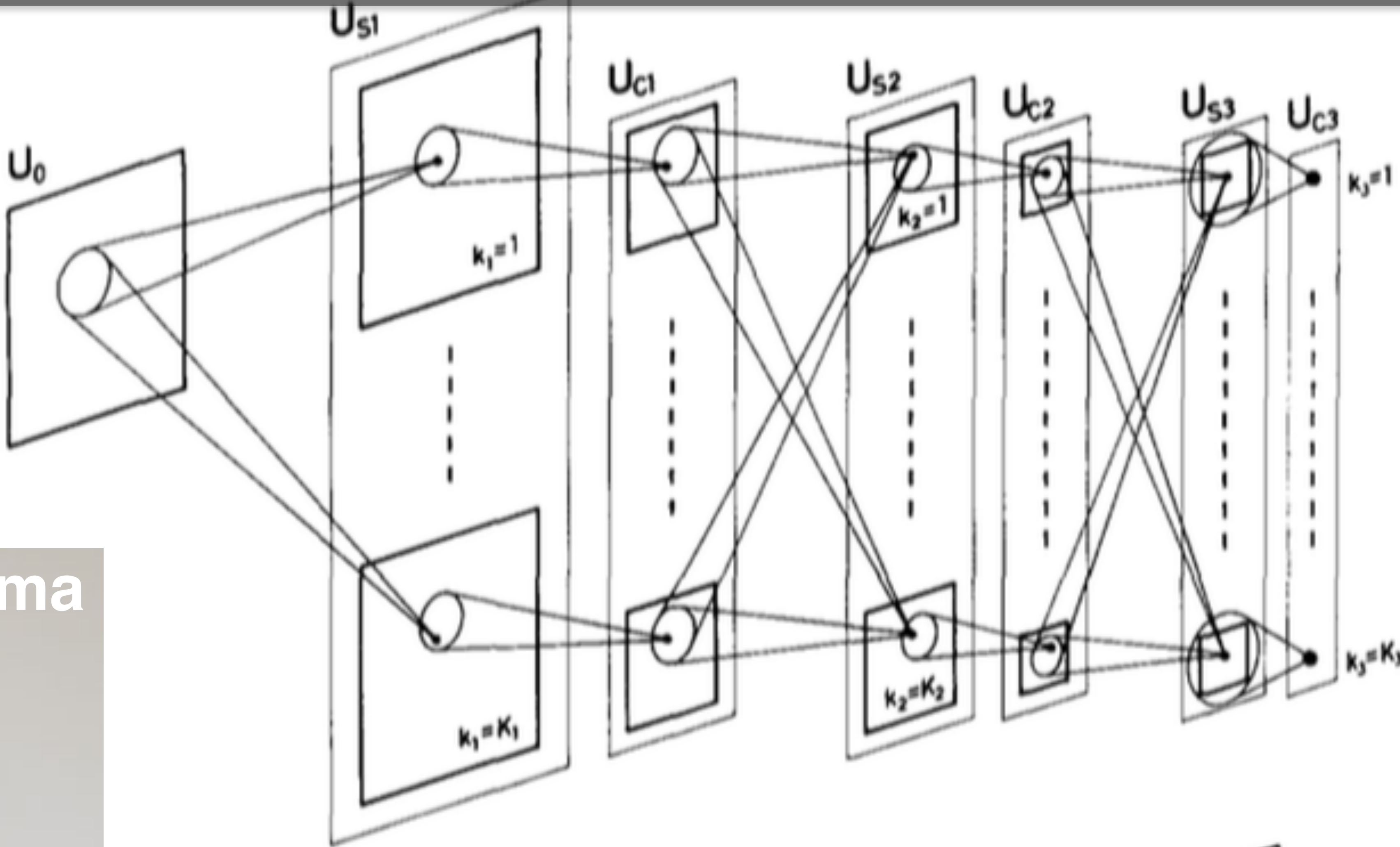
Is vision innate or acquired?

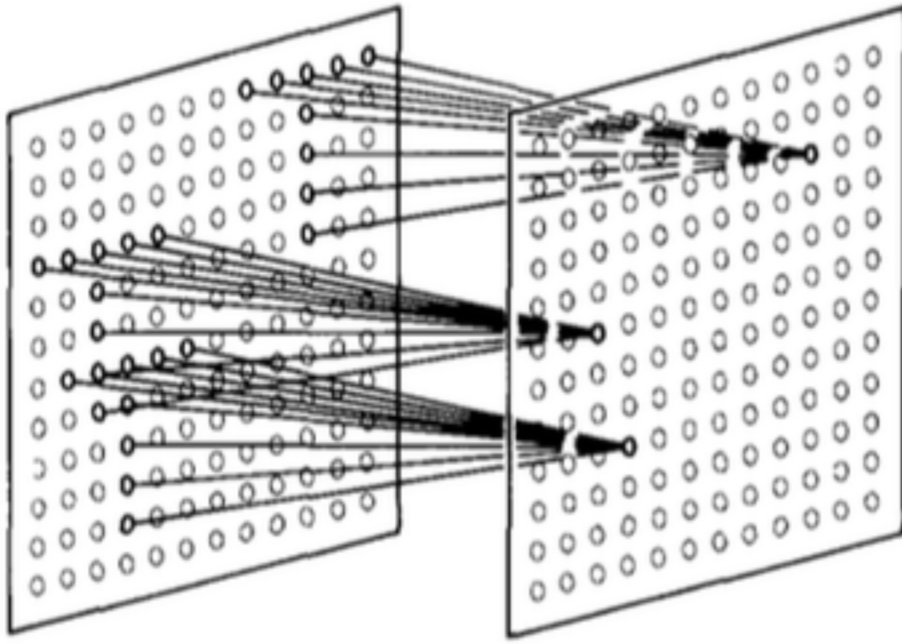


Colin Blakemore

1979

Neocognitron

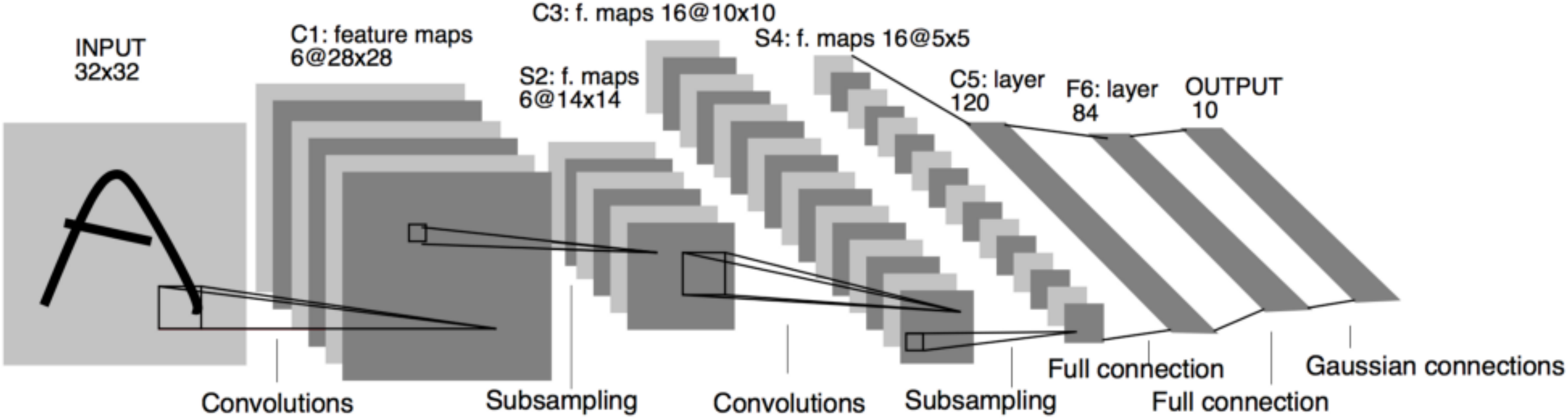




Kunihiko Fukushima

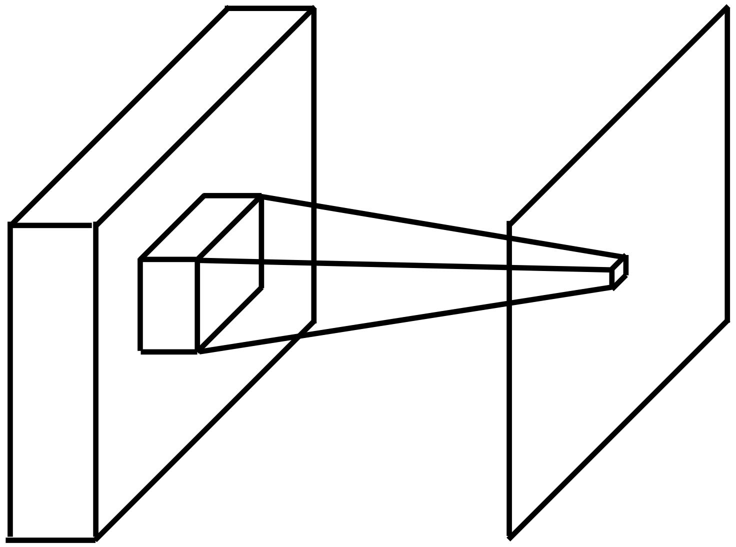
1998

LeNet-5: Convolutional Neural Networks



Important Concept 1: Receptive Fields

- Parts of the visual system consist spatially local connections being fed into the neurons
    - In such a scenario, we can think about the Receptive Field (RF) of a neuron



Important Concept 2: Equivariance

- A function $f(x)$ is **equivariant** to a function g if $f(g(x)) = g(f(x))$
  - If the input changes, the output changes the **same** way

Important Concept 3: Translational Equivariance

- Consider what would happen if you had grids of neurons with their own receptive fields, but with **shared weights**.
  - Each neuron would respond in the same way to a given stimulus within its RF
  - If an input stimulus were moved over the grid, then the outputs of the neurons would *move* in the same way
    - This is **translational equivariance** and this is the key property of a ‘Convolutional Layer’ in a network

Signal Processing: Convolution and Cross-Correlation

- Convolution is an element-wise multiplication in the Fourier domain (*c.f. Convolution Theorem*)
  - *f* ∗ *g* = ifft(fft(*f*) . fft(*g*))
  - Whilst *f* and *g* might only contain real numbers, the FFTs are complex (*real* + *imag*j)
    - Need to do **complex multiplication**!

$$\boxed{(x + yi)(u + vi) = (xu - yv) + (xv + yu)i}$$

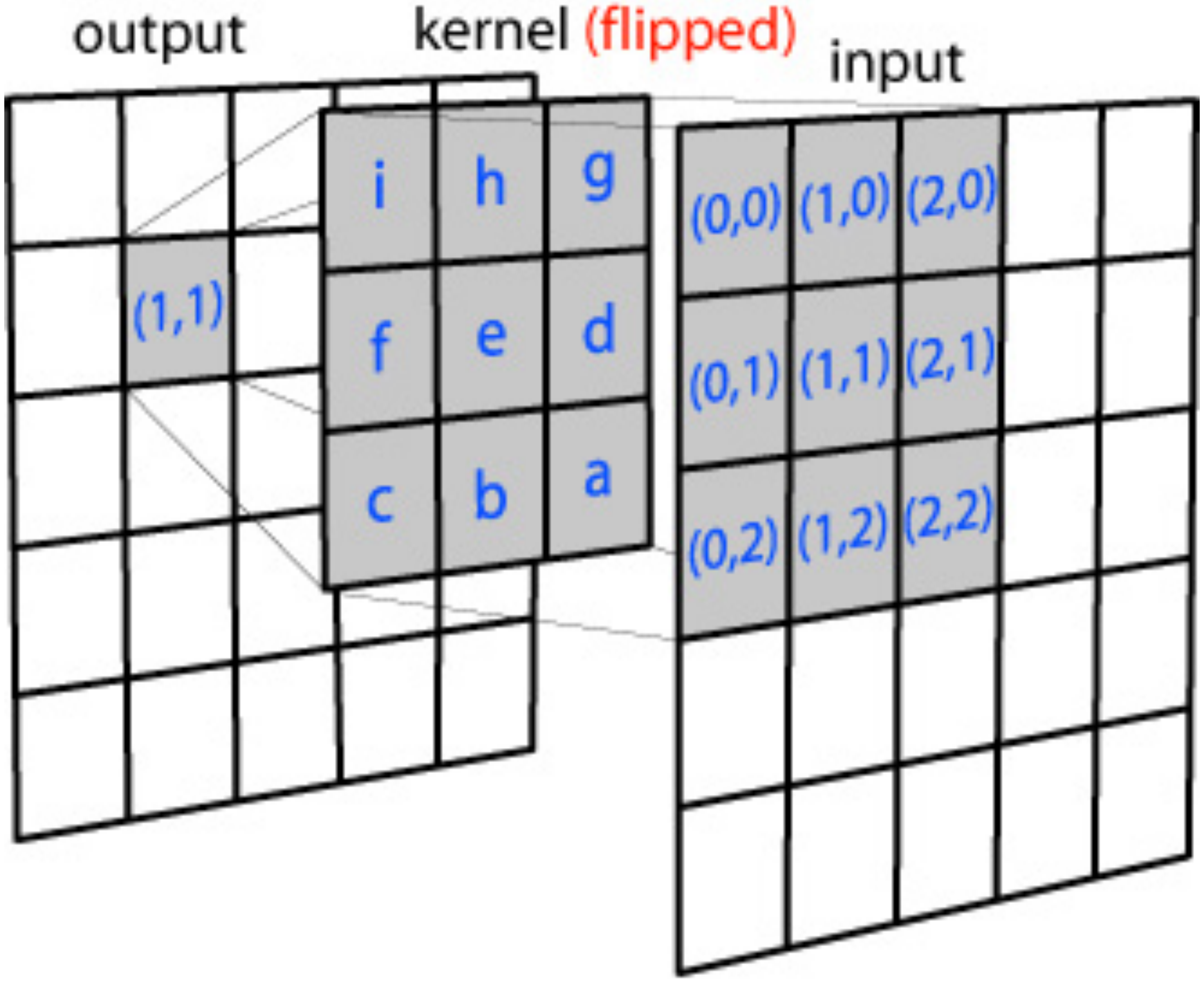
Template Convolution

- In the time domain, convolution is:

$$
\begin{aligned}
(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau \\
&= \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau.
\end{aligned}
$$

- **Notice that the image or kernel is “flipped” in time**
  - Also notice that the is no normalisation or similar

Template Convolution



What if you don’t flip the kernel?

- Obviously if the kernel is symmetric there is no difference

- However, you’re actually not computing convolution, but another operation called cross-correlation

$$(f \star g)(\tau) \overset{\mathrm{def}}{=} \int_{-\infty}^{\infty} f^*(t)\ g(t+\tau)\,dt,$$

- * represents the complex conjugate

- (you can compute this with the multiplication of the FFTs just like convolution: iFFT(FFT(f)* . FFT(g))

“Convolution” in Neural Networks

- **Important Concept 4:** “Convolution” in the neural network literature almost always refers to an operation akin **cross-correlation**
  - An element-wise multiplication of learned weights across a receptive field, which is repeated at various positions across the input.
  - Normally, we also add an additional *bias term*
    - Most often a single one (for each *kernel*), but could be one for each spatial position.
  - There are also other parameters of these “convolutions”…

Convolutional Layers

- In a convolutional layer, we have multiple kernels or filters which are learnt (plus the biases)
    - Each filter produces a single “Response Map” or “Feature Map” which are stacked together as “channels” of the resultant output tensor

Efficient Computation of Convolutions

- Classical theory would suggest that the most efficient way to compute convolution (or cross-correlation) is via the Fourier transform if the kernels are larger
  - Or via direct spatial-domain implementation for small kernels
- In neural networks we need to be able to compute many convolutions on a single input as quickly as possible
  - We have specialised multi-core hardware and efficient GEneral Matrix Multiply (GEMM) in BLAS to help though…

Convolution as a Matrix Multiplication

- The convolution operation can be expressed as a matrix multiplication if either the kernel or the signal is manipulated into a form known as a Toeplitz matrix:

$$y = h * x = \begin{bmatrix} h_1 & 0 & \dots & 0 & 0 \\ h_2 & h_1 & \dots & \vdots & \vdots \\ h_3 & h_2 & \dots & 0 & 0 \\ \vdots & h_3 & \dots & h_1 & 0 \\ h_{m-1} & \vdots & \dots & h_2 & h_1 \\ h_m & h_{m-1} & \vdots & \vdots & h_2 \\ 0 & h_m & \dots & h_{m-2} & \vdots \\ 0 & 0 & \dots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & 0 & \dots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

- For 2D convolution one would use a “doubly block circulant matrix”

- **Important Concept 5: convolution is a linear operator**

N-d Tensor Convolution

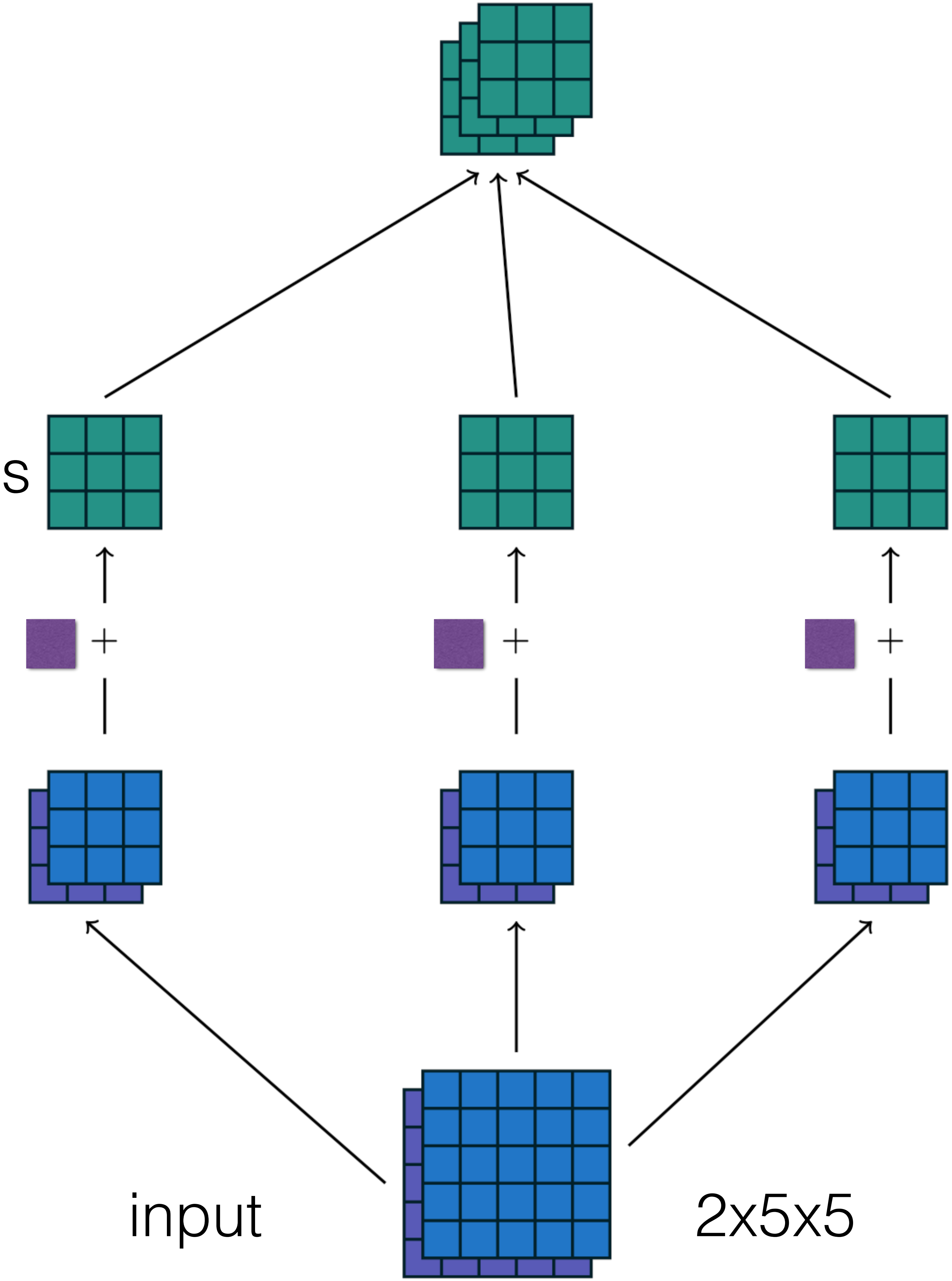
- In neural networks we want to expand our use of convolutions to work with tensors of any number of dimensions

    - If the input is say C x H x W, where C is the “channels” dimension and H & W are the spatial dimensions, we would define a convolutional kernel of size C x K x L

N-d Tensor Convolution

- We also don't typically want a single kernel, but rather many
  - Each one acting as a feature detector producing a feature map
  - We can just add another dimension to the kernel tensor to incorporate convolution with all kernels in one operation:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$



Feature maps 3x3x3

Biases 3

Kernel/ weights 3x2x3x3

input 2x5x5

Data Types

- Convolutions are applied to many dimensionalities and types of data - for example:

	Single Channel	**Multichannel**
1-D	Audio	Multiple sensor data over time
2-D	Audio data preprocessed into a spectrogram; greyscale images	Colour image data (e.g. RGB)
3-D	Volumetric data, e.g. CT scans	Colour video data

Convolutional Layer Parameters

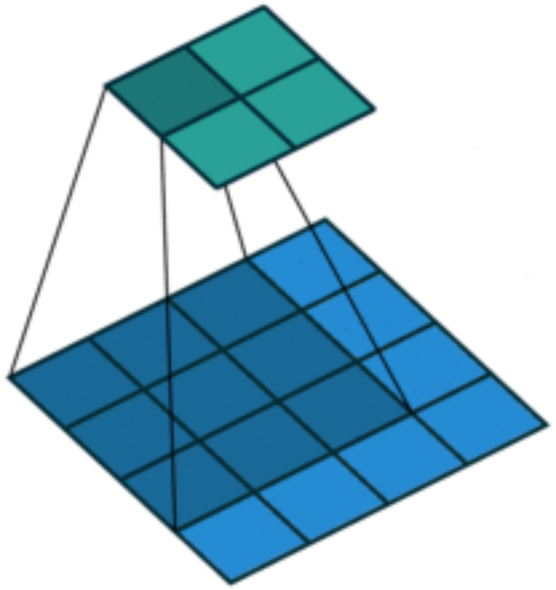
- The core parameters of a convolution are:
  - The dimensionality (is it 1-D, 2-D, 3-D in the spatial sense?)
  - The spatial extent of the kernel(s)
  - The number of kernels (or output channels)

2d convolutions, kernel size=(1,1)

- 1x1 convolutions are a common place operation, but might seem non-sensical at first
  - They do not capture any local spatial information
  - They are used to change the number of channels without affecting the spatial resolution

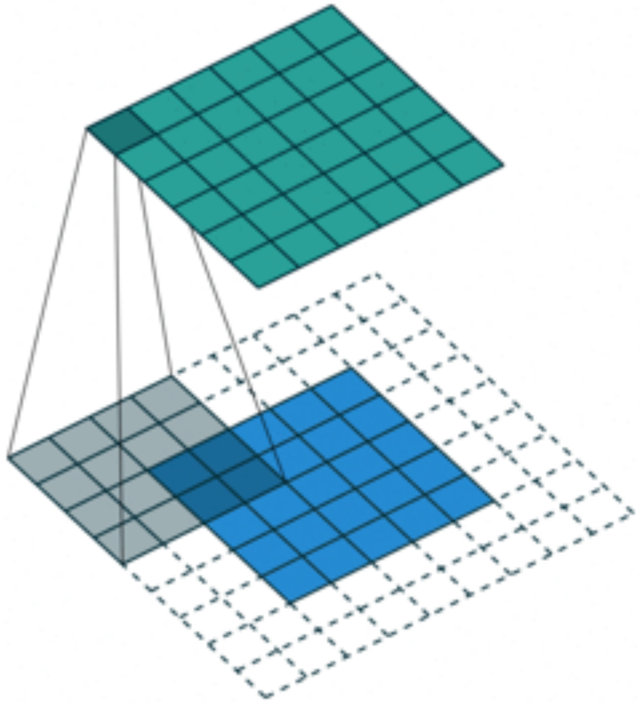
Padding

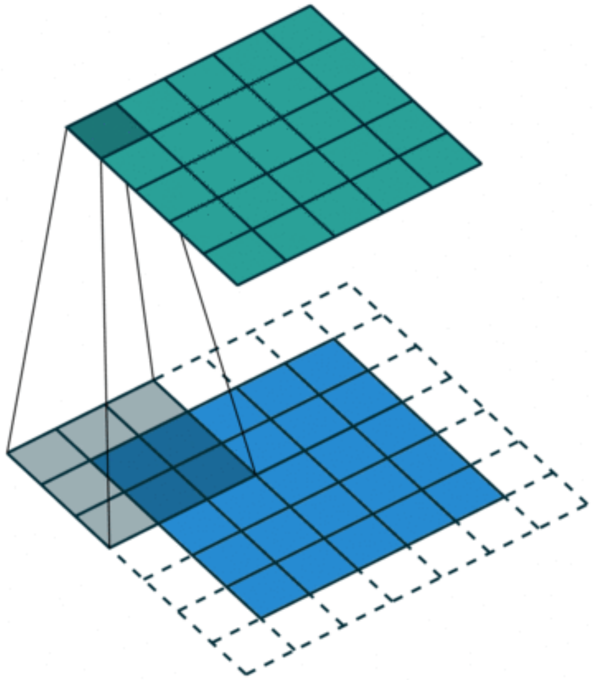
- What happens to a convolution at the edges of its spatial extent?
  - In signal processing, using the Fourier transform the “image” wraps around, so the output is the same size as the input
  - In spatial convolution if we do nothing, the output will be smaller…
    - So, we often use zero-padding to retain the size



No padding

Arbitrary padding

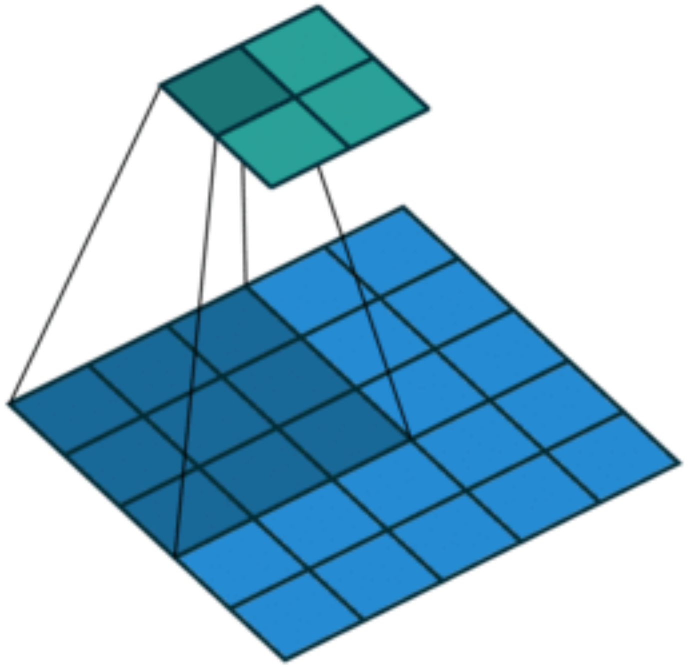




“same” padding

Striding

- Convolution is expensive… could we make it cheaper by skipping over positions?

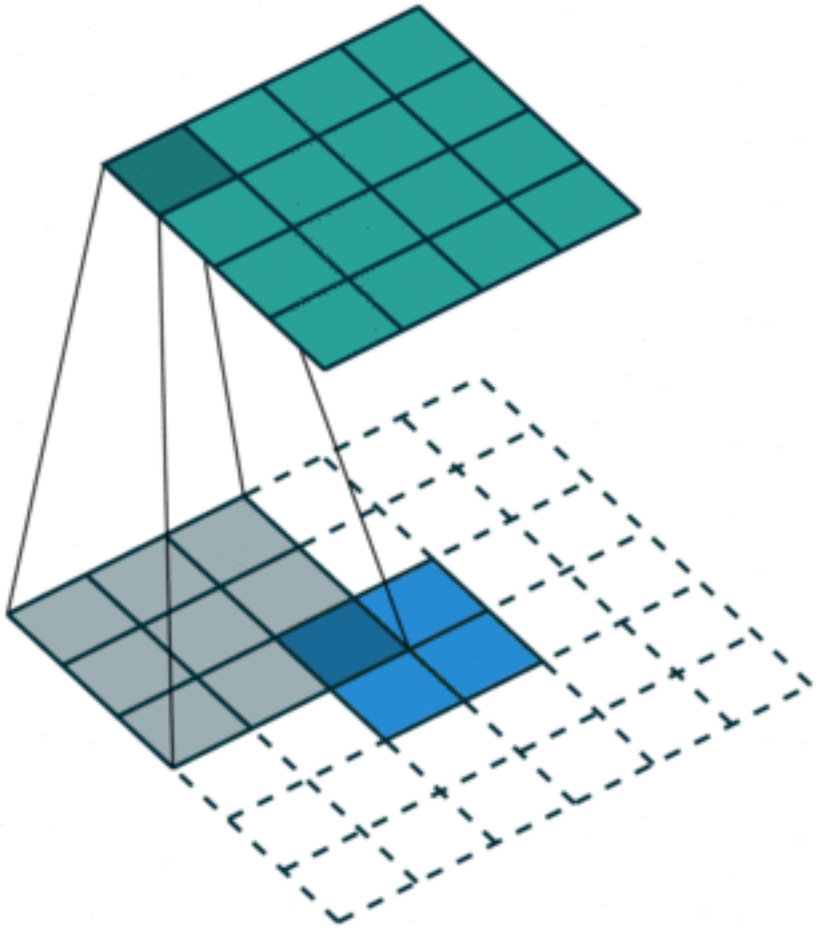


Stride=(2,2)

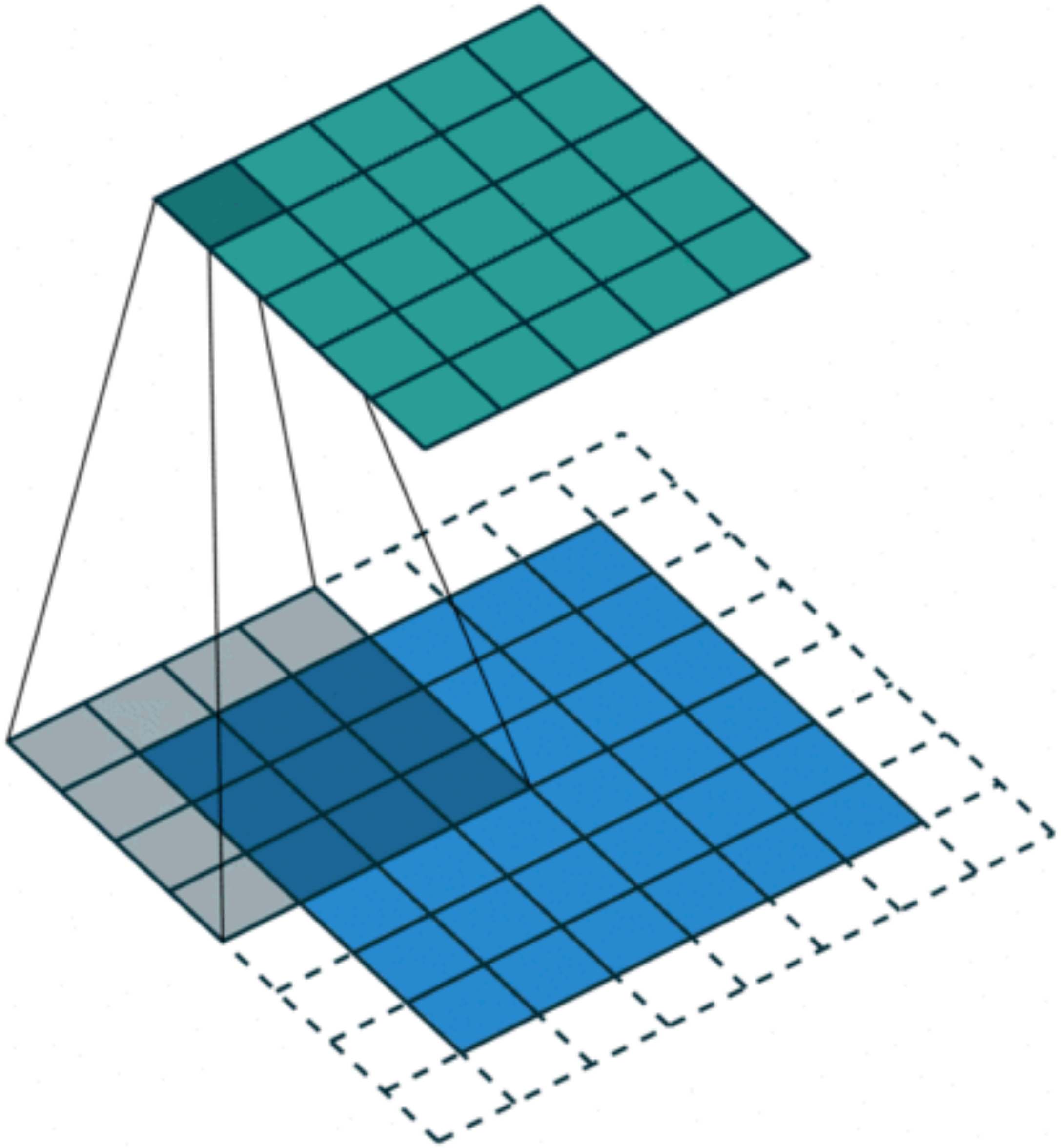
Fractional Striding/ Transpose Convolution

- What if we consider *fractional* strides between 0 and 1?
  - Intuitively, if bigger strides subsample, then fractional strides should upsample
  - This is equivalent to “expanding” the input by padding and performing convolution
    - And potentially also striding by adding zeros around all the values

Transpose convolution, stride=1

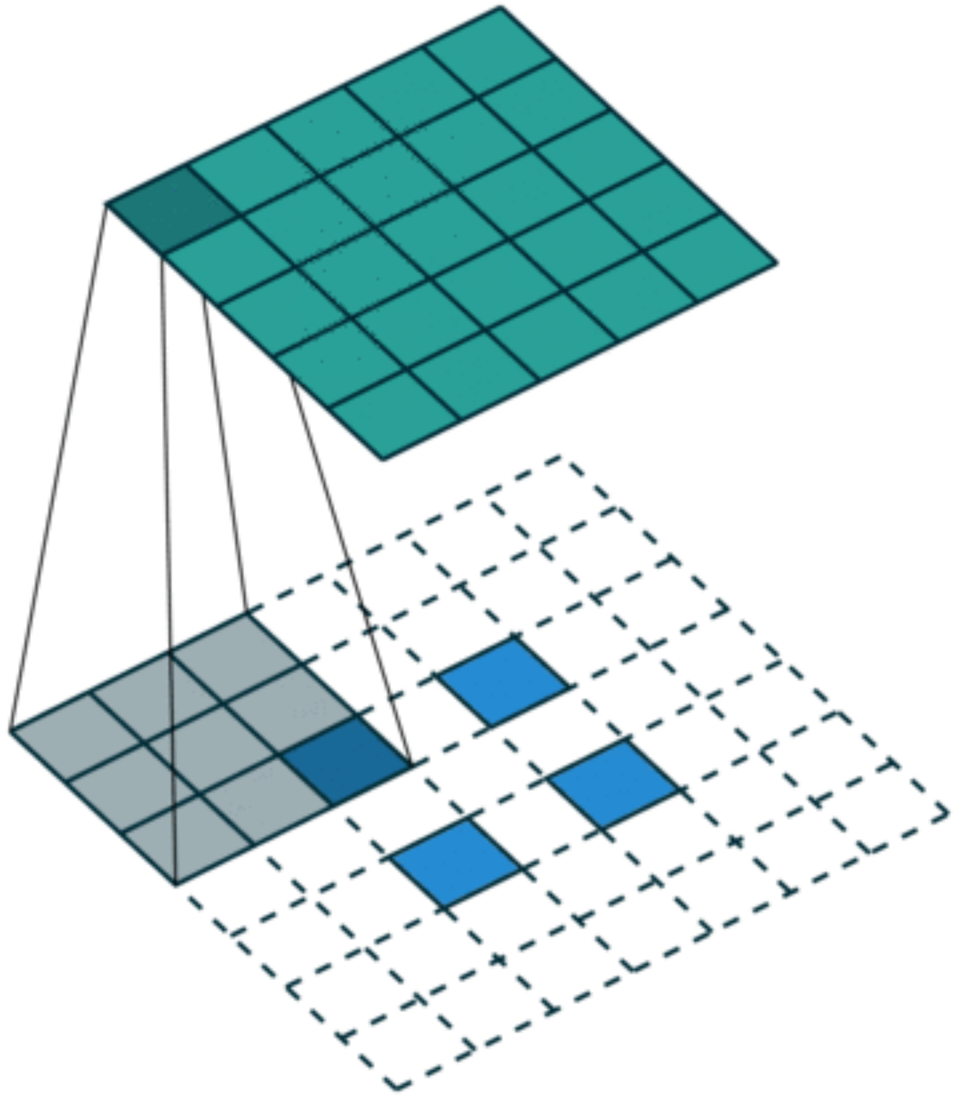


No padding

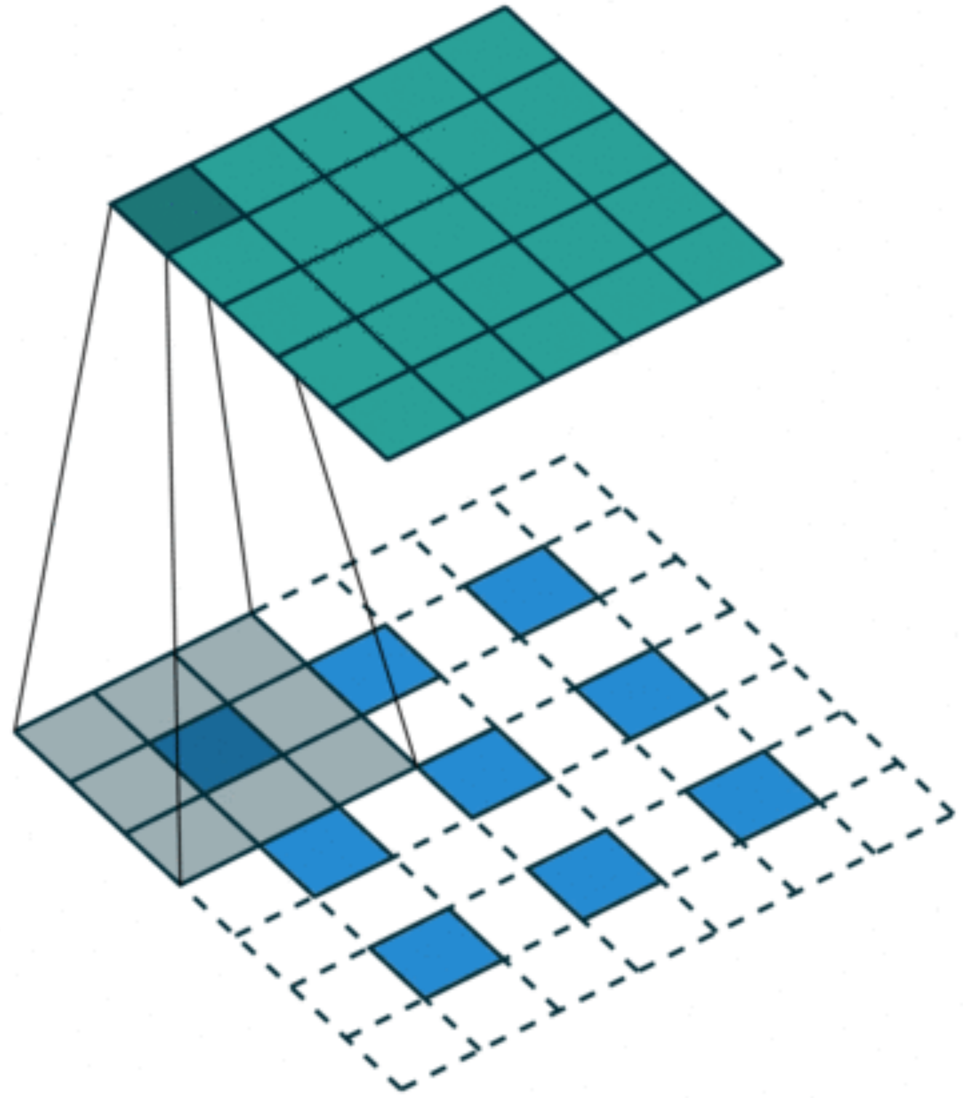


Arbitrary padding

Transpose convolution, stride=2



No padding



Padding

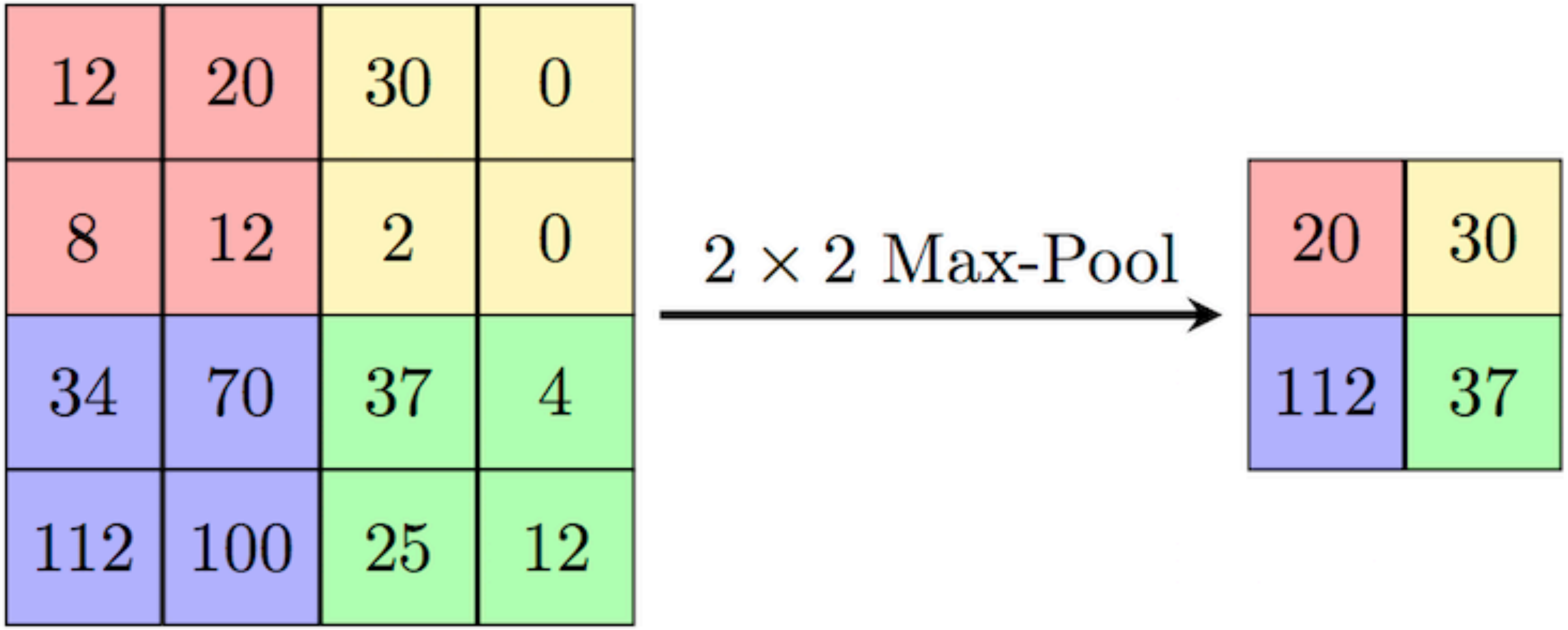
- You'll often find fractionally stride convolutions described as "transposed convolutions"
    - That's because they can be implemented by transposing the kernel's Toeplitz matrix before the multiply
- Some old literature also refers to this as "deconvolution"
    - *Please don't do that!!*

- Also note that this might not be the best way of upsampling (see https://distill.pub/2016/deconv-checkerboard/)

Pooling

- Striding is a popular way to reduce spatial dimensionality in modern networks
    - before striding was devised, **pooling**, was the defacto way of reducing dimensionality

Max Pooling, 2x2, stride=2



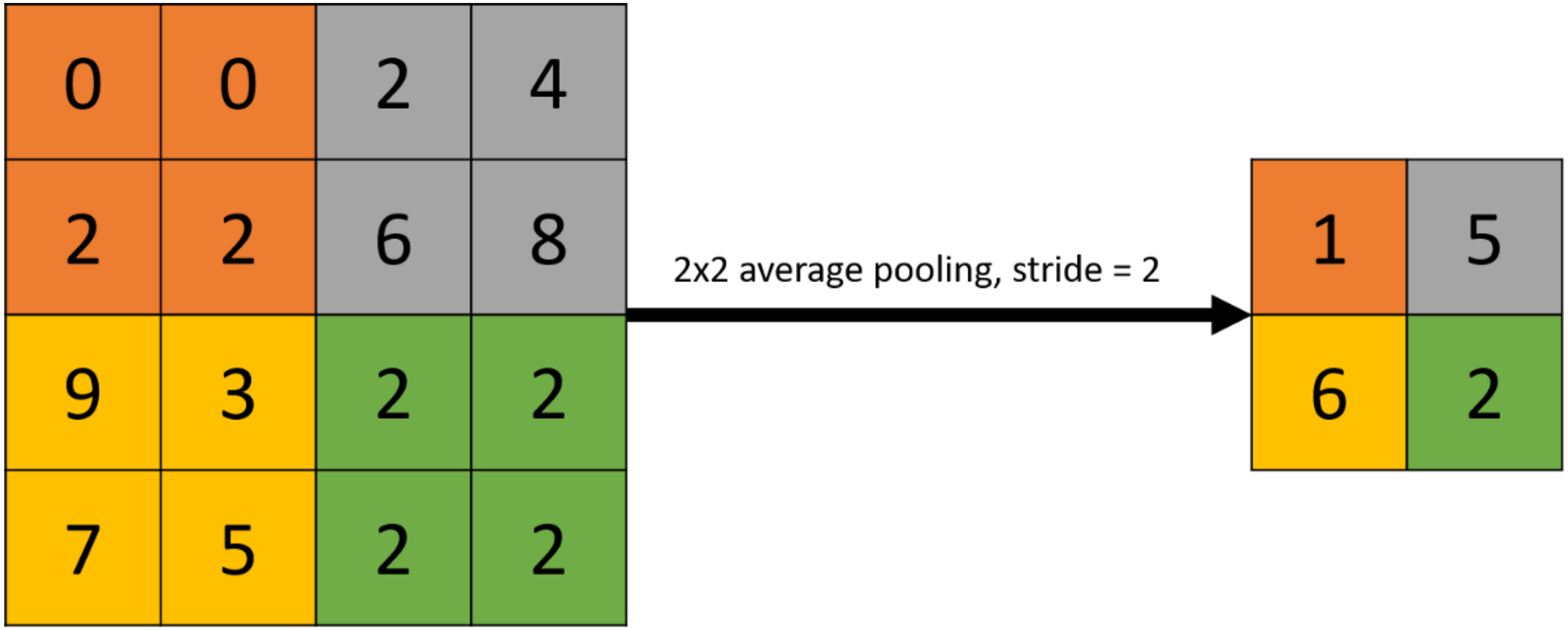
12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

Max Pooling Gradients

- The gradient of the max pooling operation is 1 everywhere a max value was selected, and zero elsewhere
    - This means that implementations not only need to record the max values in the forward-pass, but also keep track of the positions of those maximums for the backward pass

Average Pooling



0	0	2	4
2	2	6	8
9	3	2	2
7	5	2	2

2x2 average pooling, stride = 2

1	5
6	2

Local Versus Global Pooling

- The pooling operations on the previous slides are local
  - They result in a feature map reducing in spatial size
- **Global pooling** reduces a feature map to a scalar
  - So a tensor of many feature maps would be reduced to a single feature **vector**
  - Often used near the end of networks to flatten feature maps into feature vectors that can be fed into an MLP

Dilated Convolutions

- Sometimes we want to have larger receptive fields in our networks
  - We can increase the kernel size to achieve this, but this introduces more weights
  - We can downsample/pool the input, but this decreases spatial resolution
  - Or we could 'pad' the kernel with zeros throughout to increase the effective size without increasing the number of parameters