http://inst.eecs.berkeley.edu/~ee290-2/sp21/

# Hardware for Machine Learning

## Lecture 5: DNN Kernel Computation


Sophia Shao


**Ian Buck**

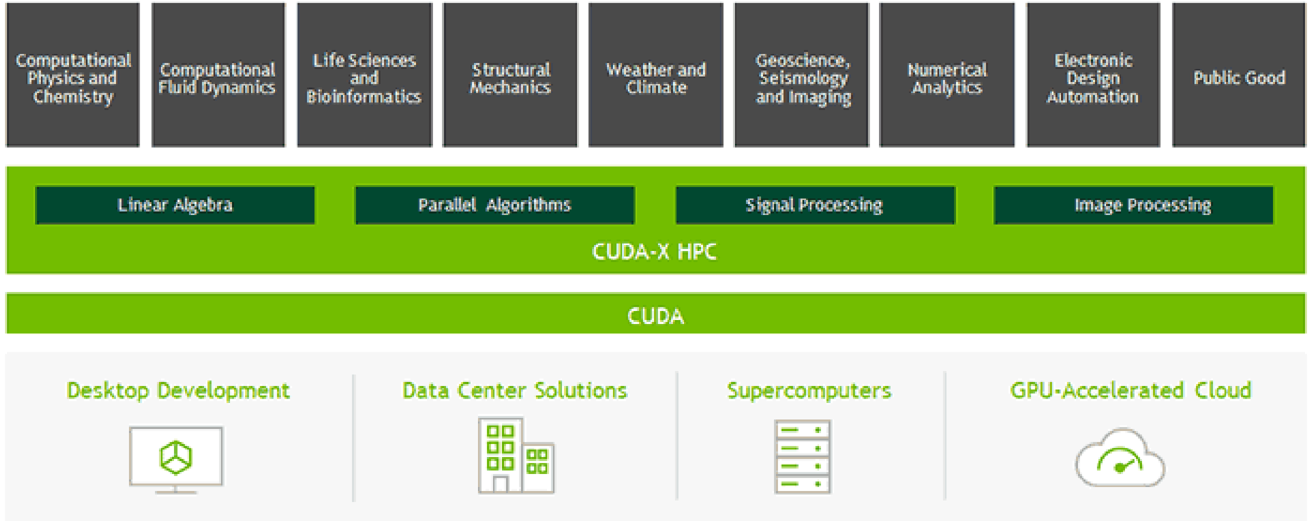
**I have completed my Ph.D. at Stanford and currently work at NVIDIA**
**LinedIn Profile: http://www.linkedin.com/pub/ian-buck/15/13/192**
**Contact Info:**

**408-486-2000**
**2701 San Tomas Expressway**
**Santa Clara, CA 95050**
ibuck@nvidia.com

**BrookGPU**

**BrookGPU is a compiler and runtime implementation of the Brook stream programming language which provides an easy, C-like programming environment for today's GPU. As the programmability and performance of modern GPUs continues to increase, many researchers are looking to graphics hardware to solve problems previously performed on general purpose CPUs. In many cases, performing general purpose computation on graphics hardware can provide a significant advantage over implementations on traditional CPUs.**

## CUDA (Compute Unified Device Architecture)

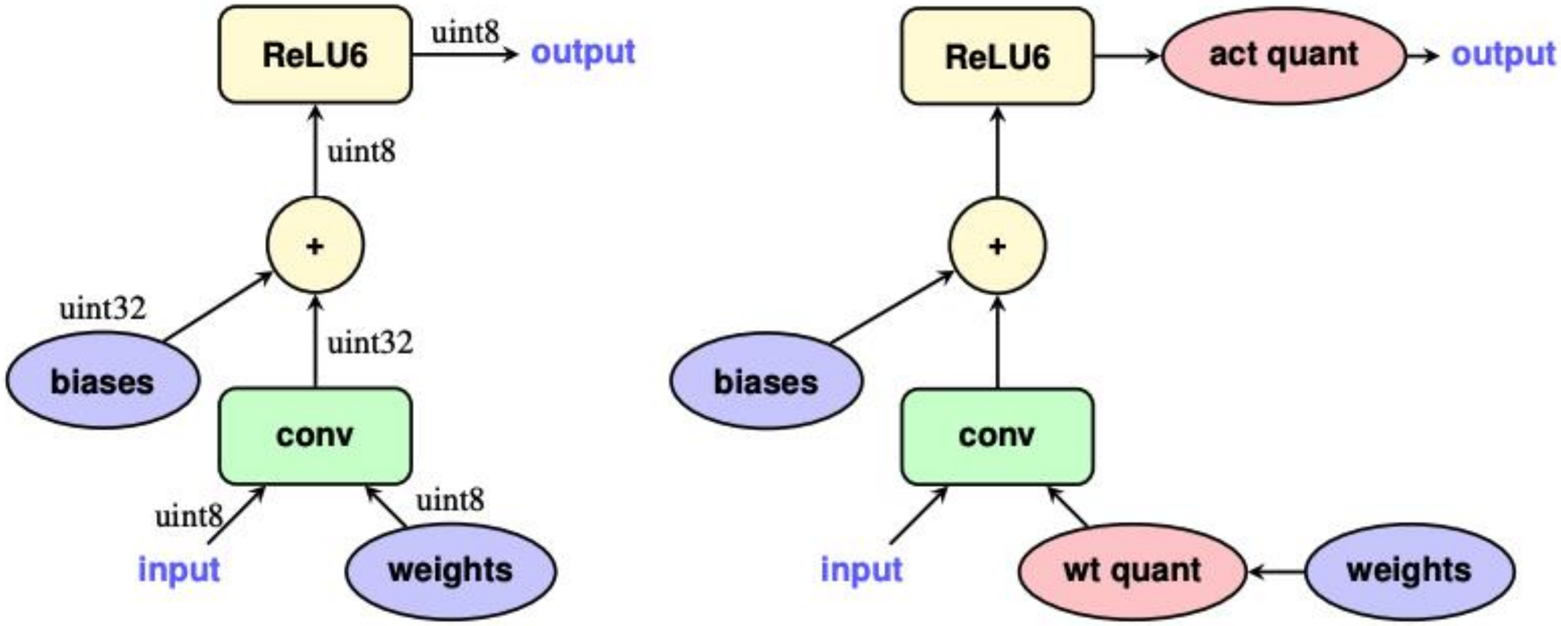


# Review

- AlexNet's cost function and optimization function
- Floating-point and fixed-point representations
- Hardware implications:
    - Fewer # of bits -> Energy/storage efficiency
- DNN Quantization
    - Using the "slope and bias" of fixed-point representation: $y = s*x + z$
        - Scaling factor
            - How to scale? How to choose threshold value?
        - Zero point
    - Post-training quantization vs Quantization-aware training
    - State-of-the-art hardware support for low-precision DNNs

# Quantization-Aware Training

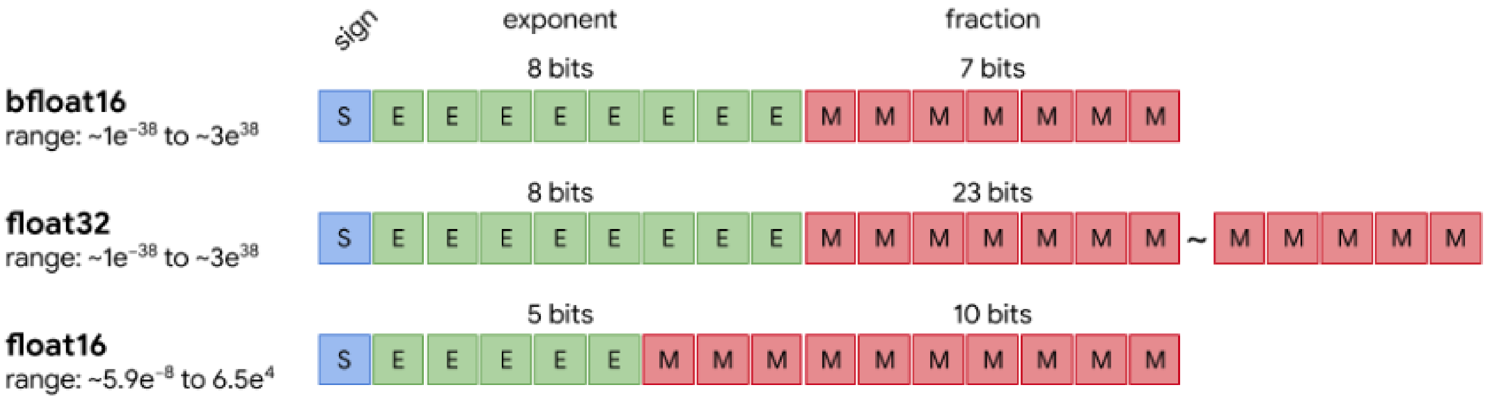
- Typically performs better than post-training quantization
- "Simulate" quantization effects in the forward pass
- Weights and biases are updated in floating point during backpropagation so that they can be nudged by small amounts.




(a) Integer-arithmetic-only inference (b) Training with simulated quantization

# Bfloat16 for Google's Tensor Processing Unit

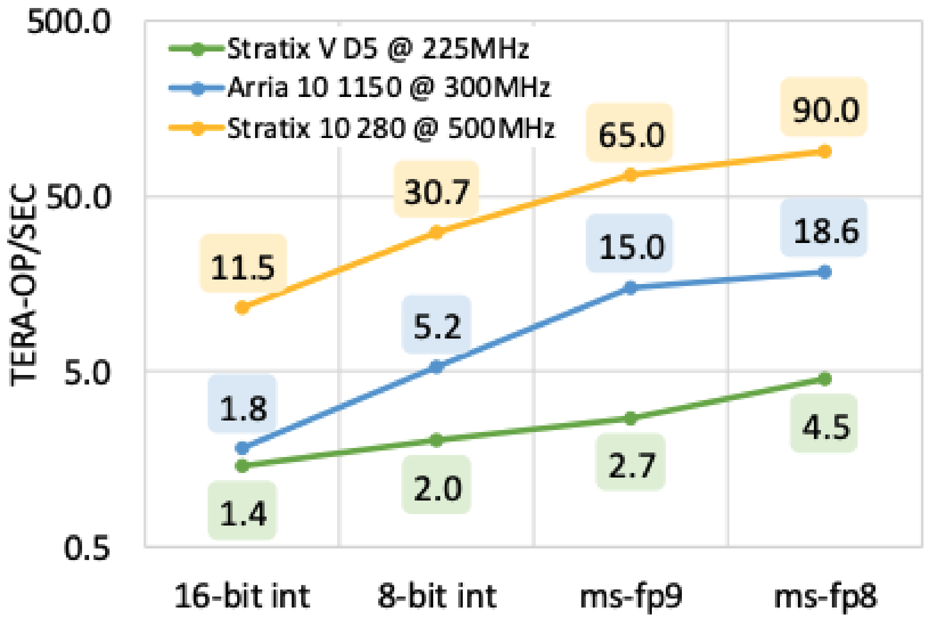
- fp32 - IEEE single-precision floating-point
- fp16 - IEEE half-precision floating point
- bfloat16 - 16-bit *brain floating point*



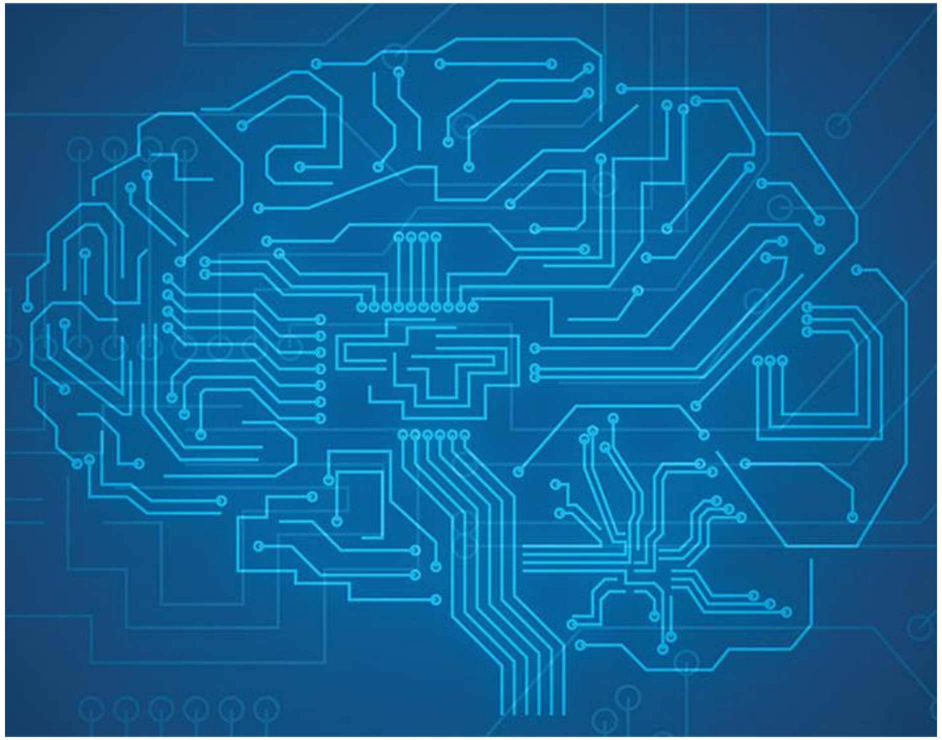

https://cloud.google.com/tpu/docs/bfloat16

# MS-FP in Brainwave FPGA @ Microsoft

- " 'neural'-optimized data formats based on 8- and 9-bit floating point, where mantissas are trimmed to 2 or 3 bits. "
- " These formats, referred to as ms-fp8 and ms-fp9, exploit efficient packing into reconfigurable resources and are comparable in FPGA area"




Serving DNNs in Real Time at Datacenter Scale with Project Brainwave



# DNN Kernels

- Overview
- Convolution
  - Basics
  - Transformation
- Pooling
- BatchNorm

# AlexNet Model

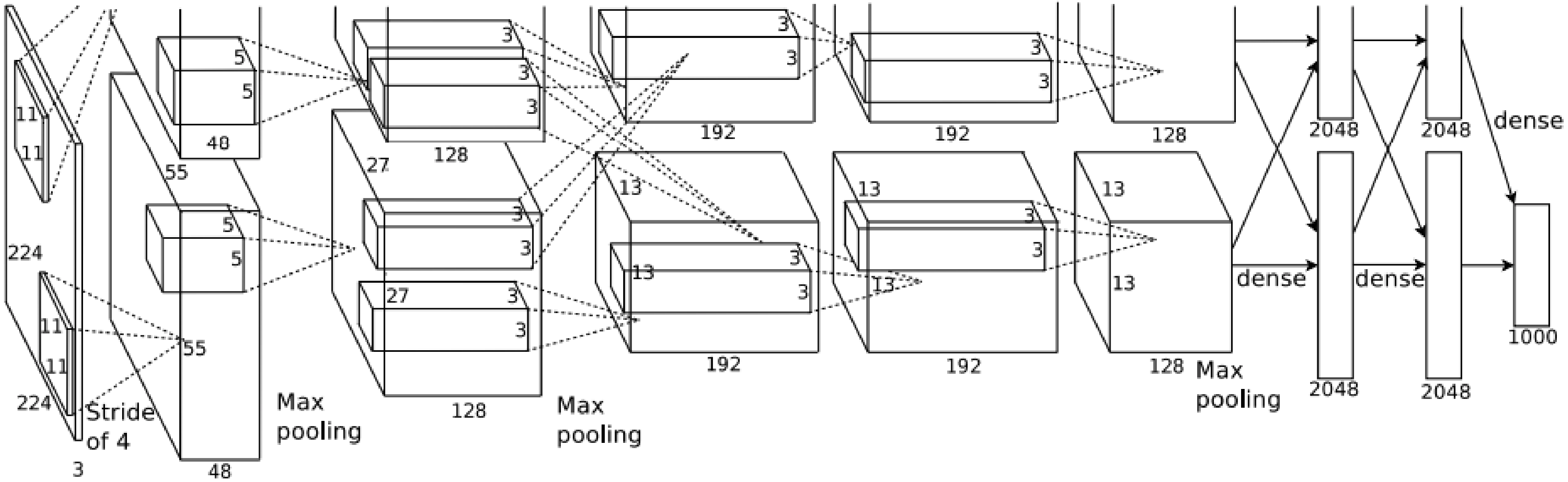



Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

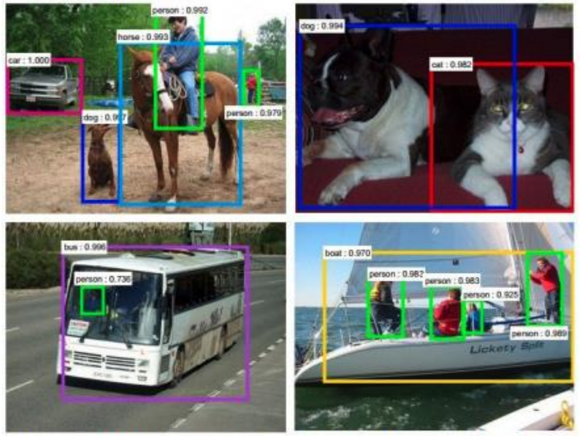
# Convolutional Neural Networks Everywhere

Classification


Retrieval

Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

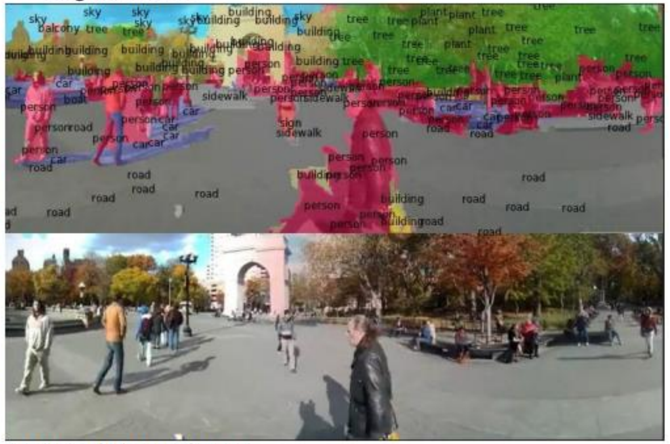
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

No errors | Minor errors | Somewhat related

*A white teddy bear sitting in the grass*

*A man in a baseball uniform throwing a ball*

*A woman is holding a cat in her hand*

*A man riding a wave on top of a surfboard*

*A cat sitting on a suitcase on the floor*

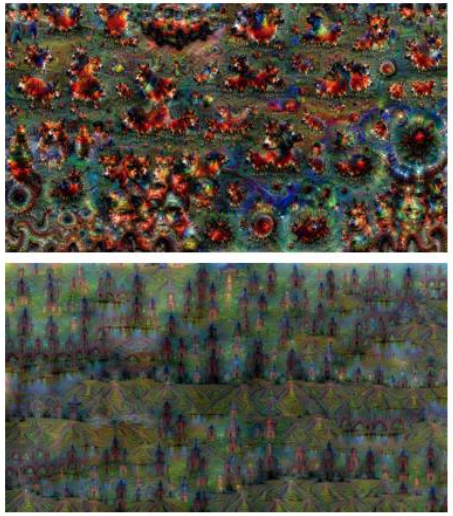
*A woman standing on a beach holding a surfboard*

Image Captioning

*[Vinyals et al., 2015]*
*[Karpathy and Fei-Fei, 2015]*

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

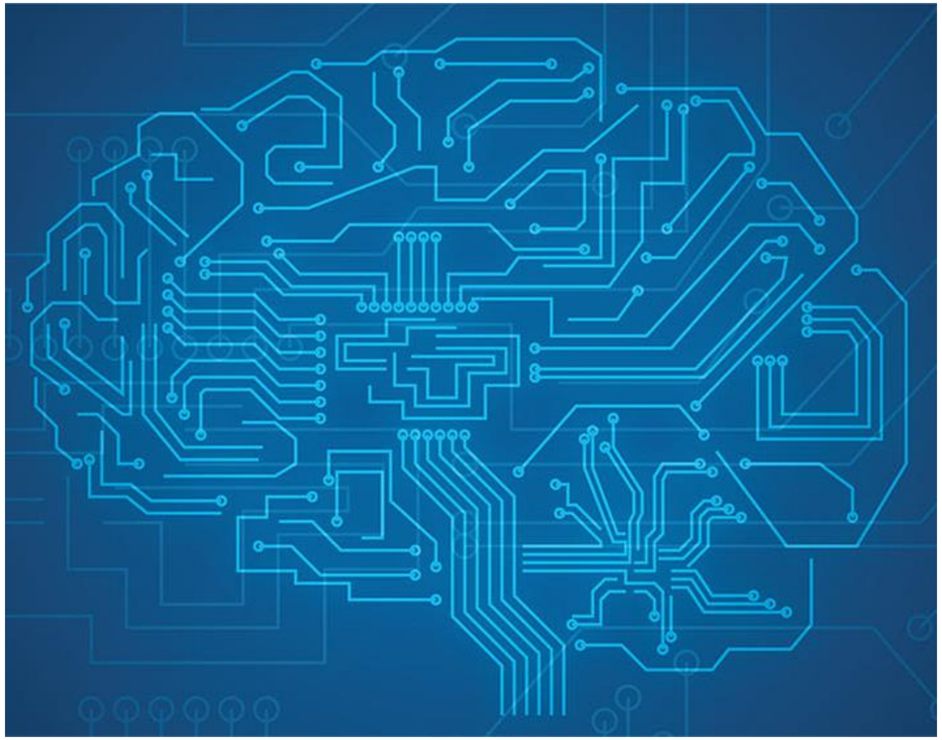
Captions generated by Justin Johnson using Neuraltalk2



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inception approach from a blog post by Google Research.

Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017; reproduced with permission

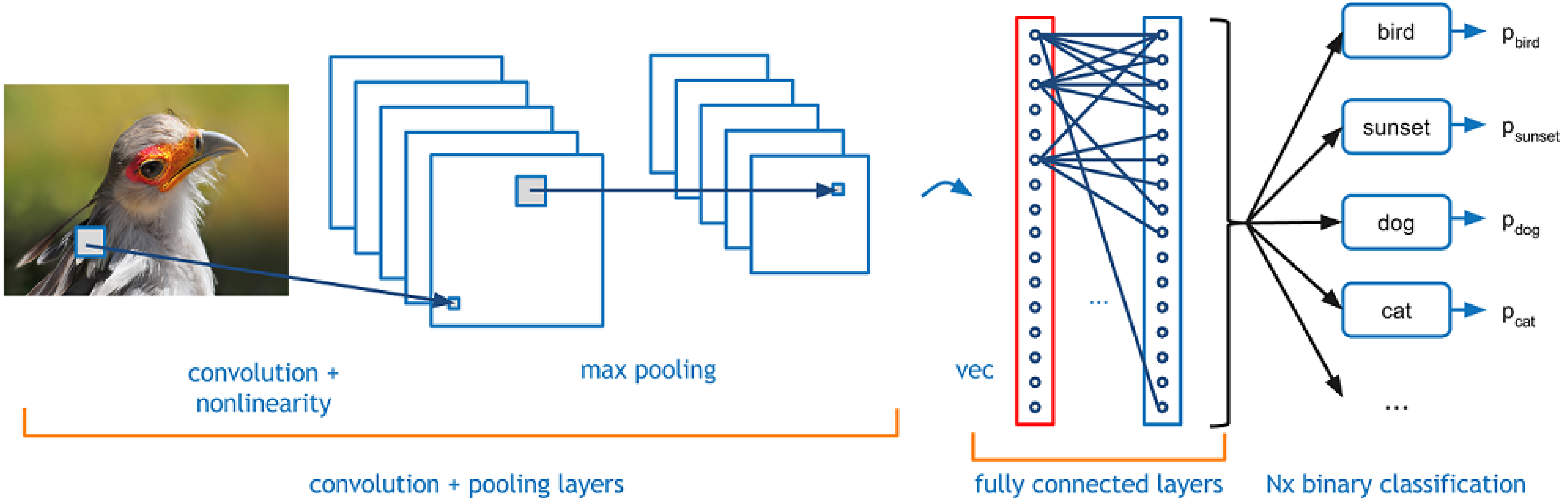
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017



# DNN Kernels

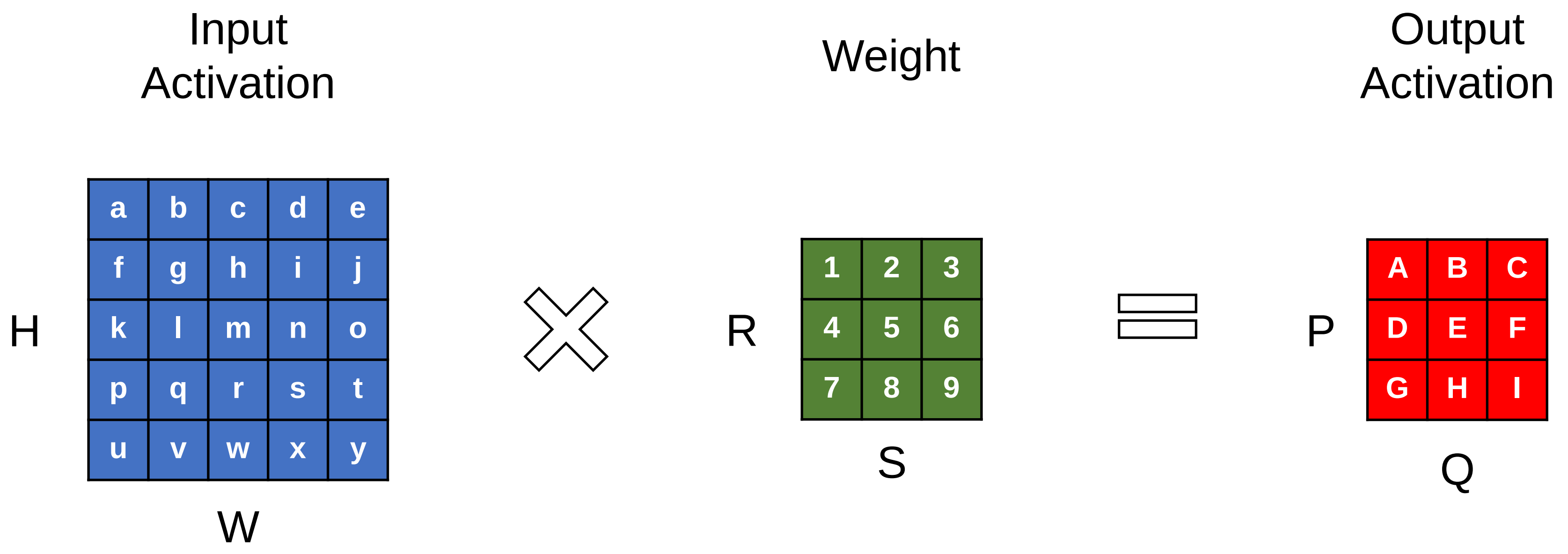
- Overview
- Convolution
  - Basics
  - Transformation
- Pooling
- BatchNorm

# Convolutional Neural Nets



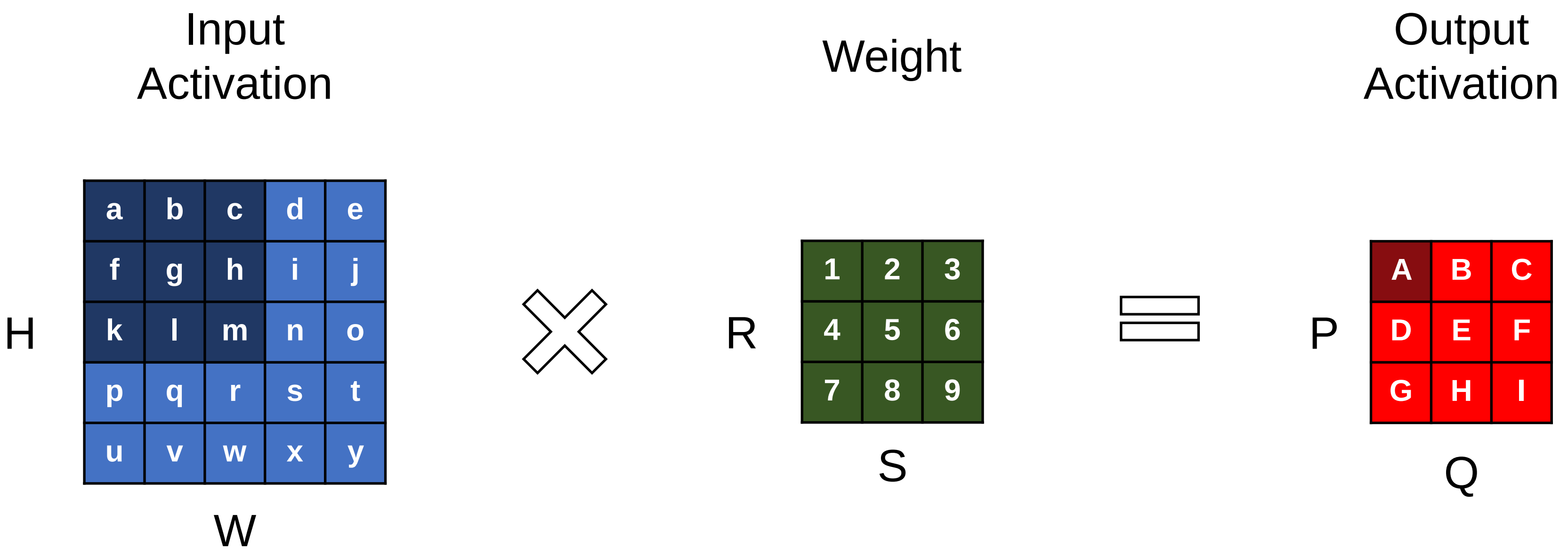

https://github.com/vdumoulin/conv_arithmetic

# 2-D Convolution




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation

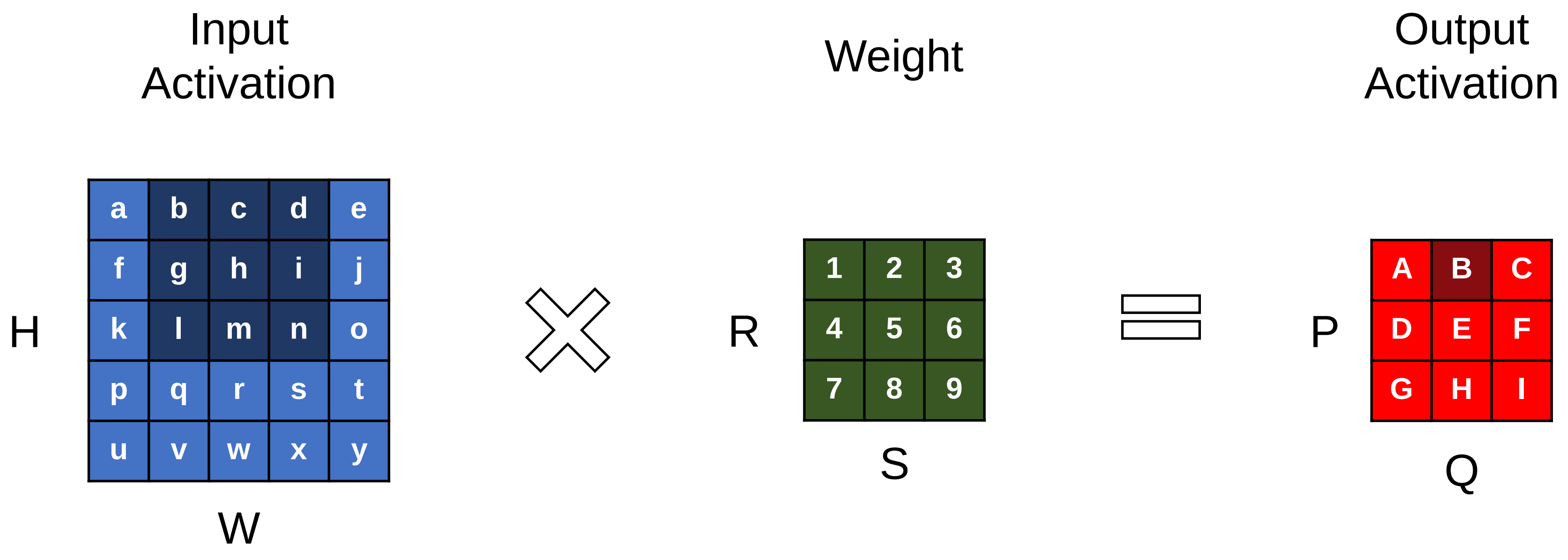
# 2-D Convolution




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation

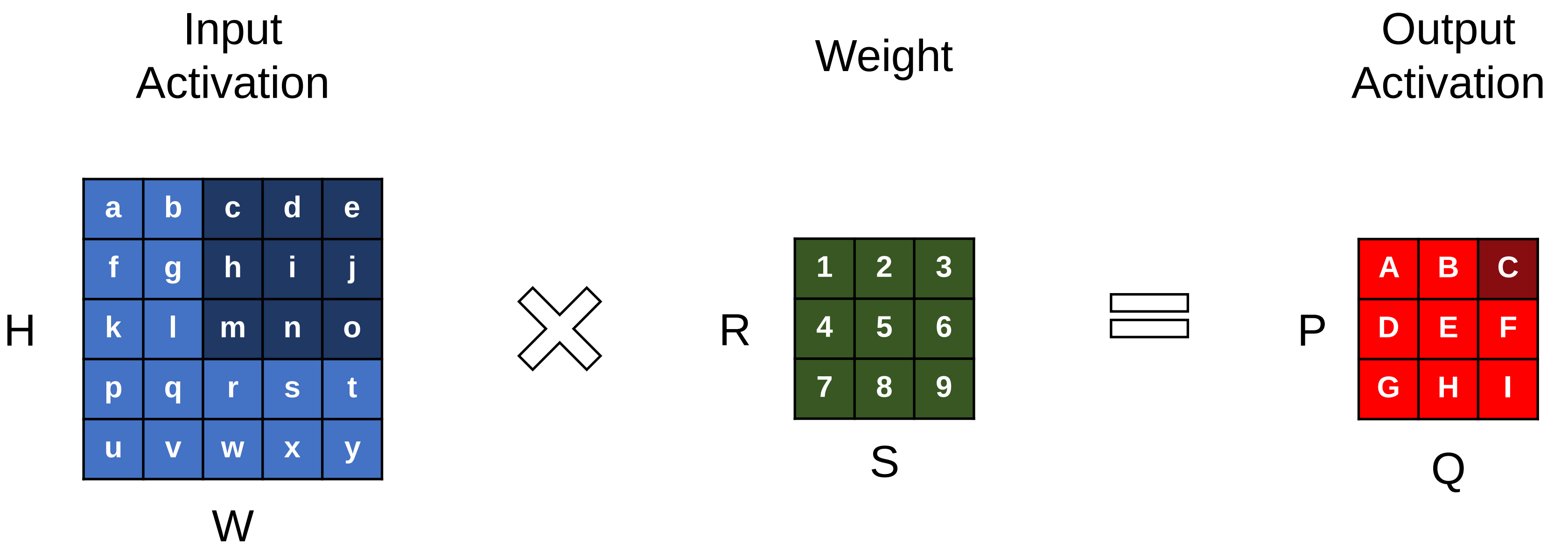
$$
\begin{aligned}
A = a*1 + b*2 + c*3 \\
+ f*4 + g*5 + h*6 \\
+ k*7 + l*8 + m*9
\end{aligned}
$$

# 2-D Convolution (stride = 1)



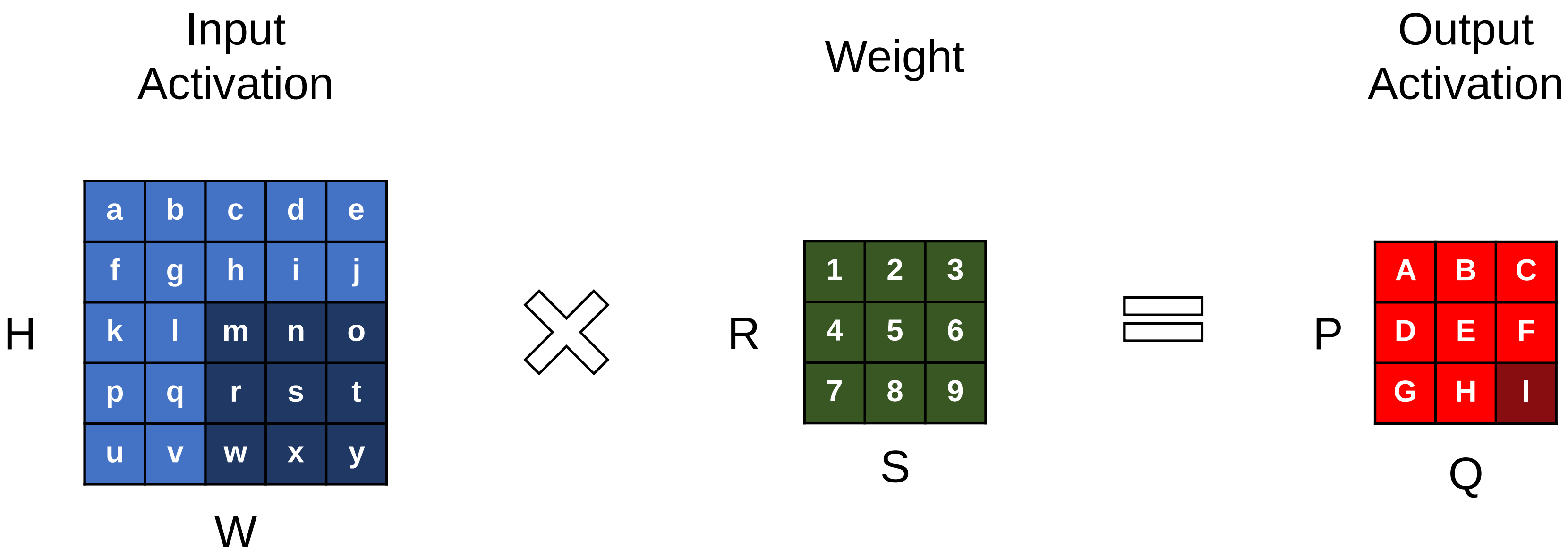

**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step

# 2-D Convolution (stride = 1)




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step

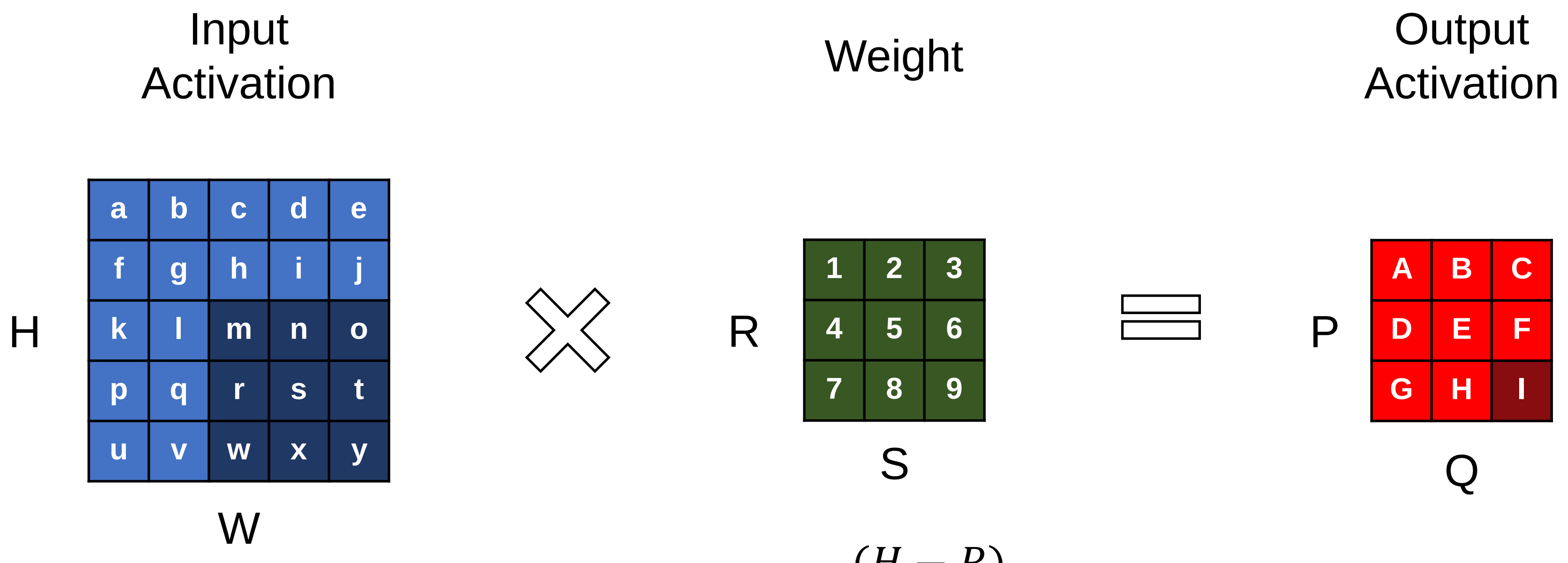
# 2-D Convolution (stride = 1)




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step

$$
\begin{aligned}
I = m*1 + n*2 + o*3 \\
+r*4 + s*5 + t*6 \\
+w*7 + x*8 + y*9
\end{aligned}
$$

# 2-D Convolution (stride = 1, valid conv.)

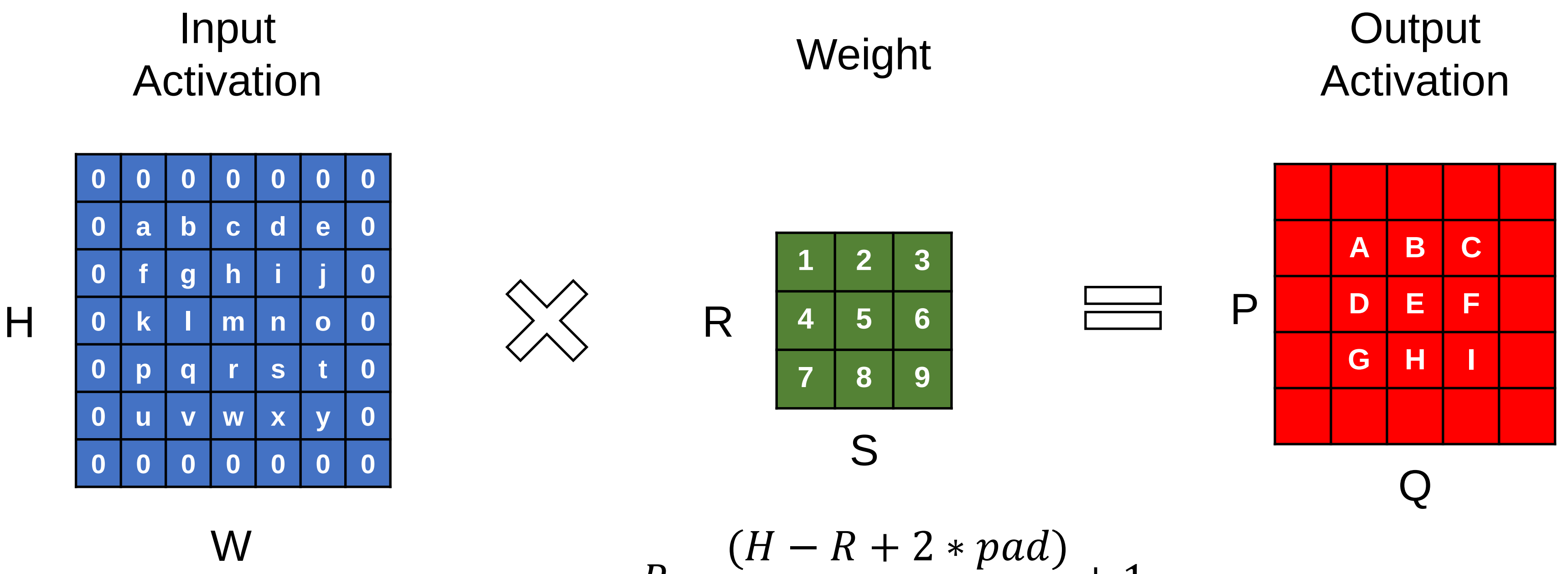



$$P = \frac{(H - R)}{stride} + 1$$

$$Q = \frac{(W - S)}{stride} + 1$$

**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step

# 2-D Convolution (stride = 1, padding = 1)

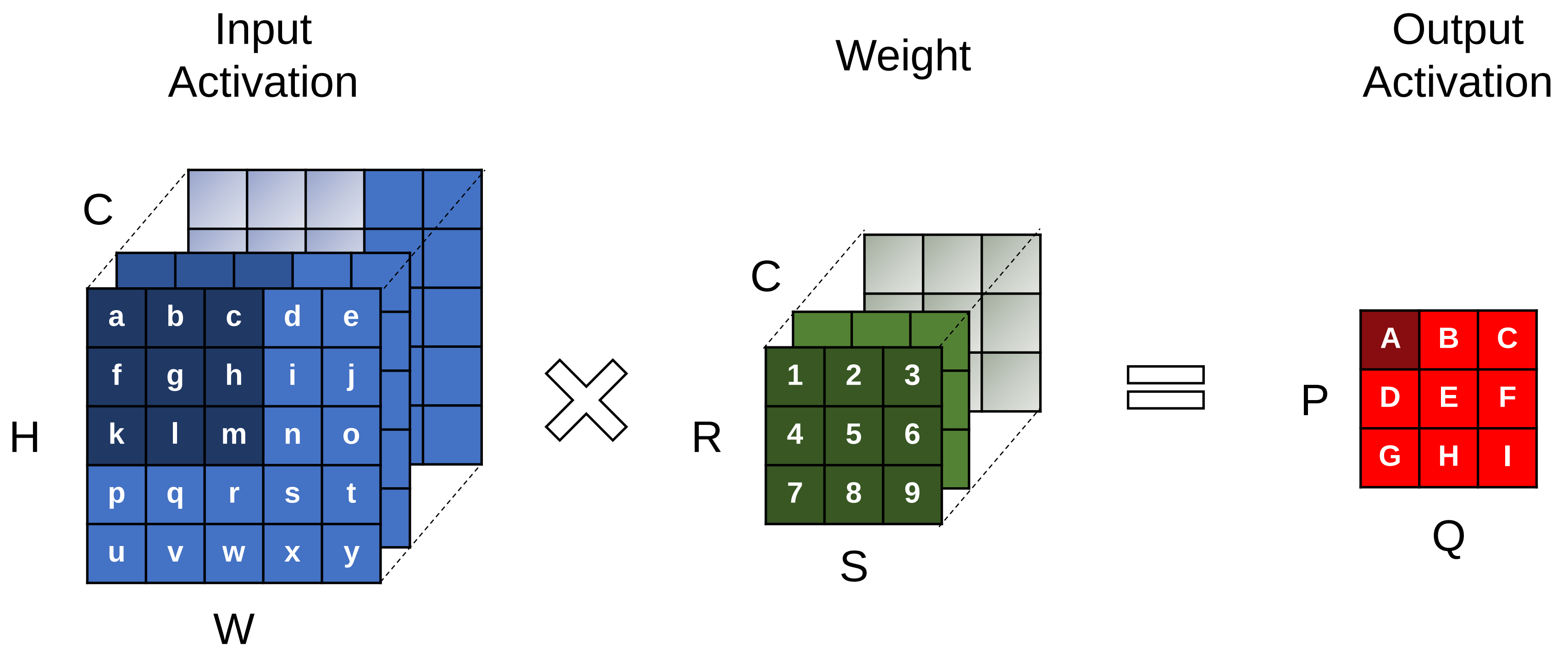



$$P = \frac{(H - R + 2 * pad)}{stride} + 1$$

$$Q = \frac{(W - S + 2 * pad)}{stride} + 1$$

**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step
**padding:** # of zero rows/columns added

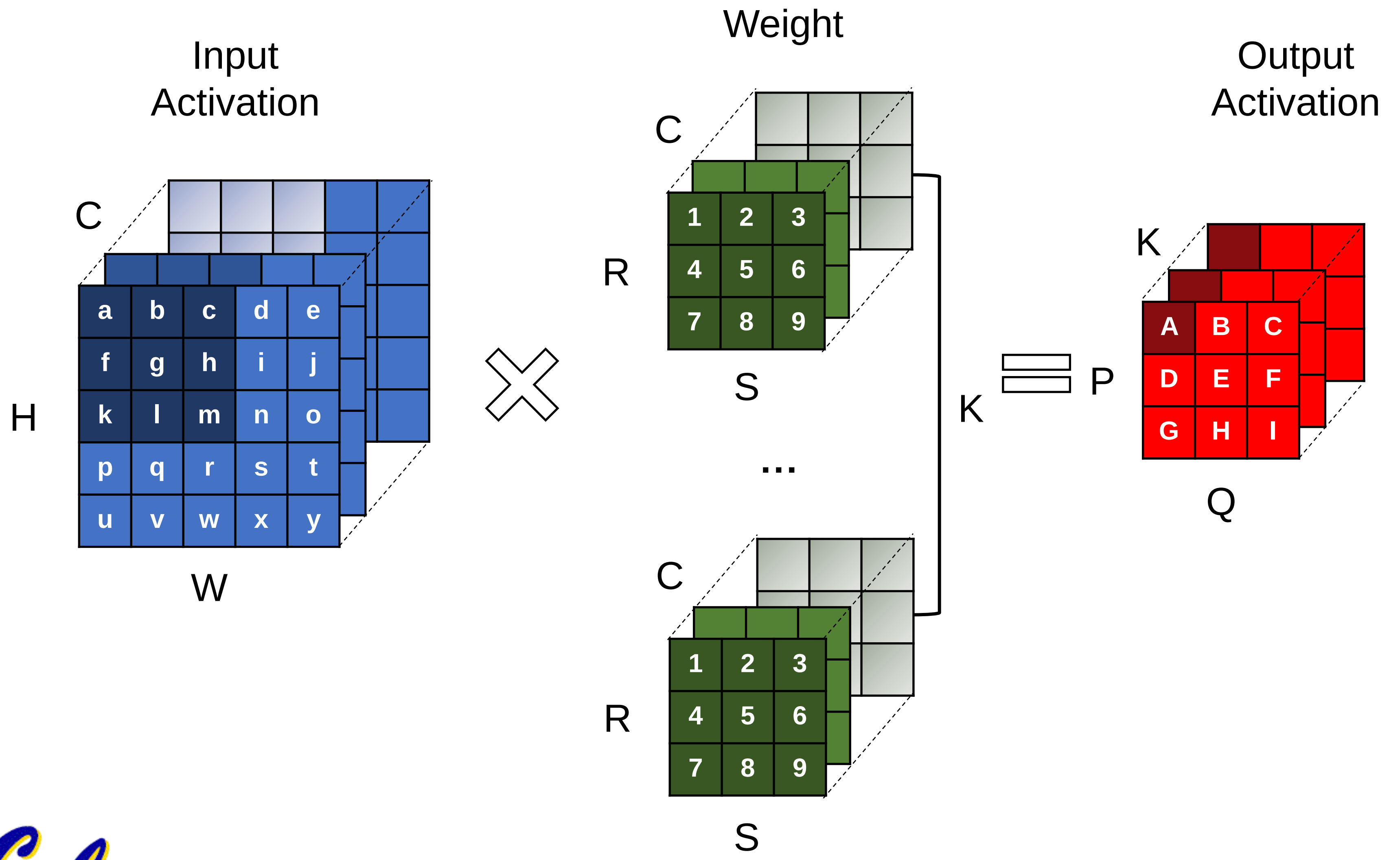
# 3-D Convolution




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step
**padding:** # of zero rows/columns added

**C:** # of Input Channels

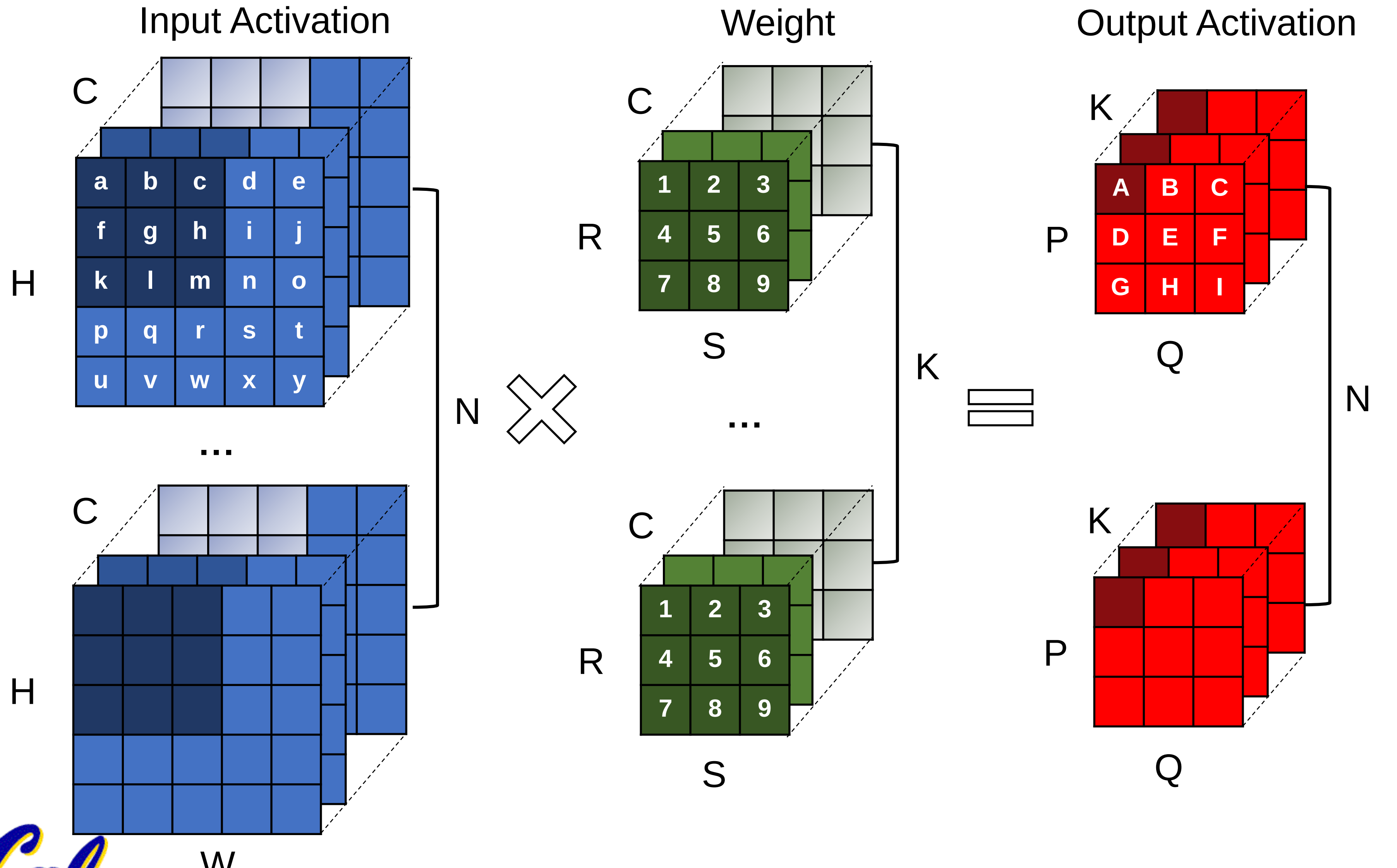
# 3-D Convolution




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step
**padding:** # of zero rows/columns added

**C:** # of Input Channels
**K:** # of Output Channels

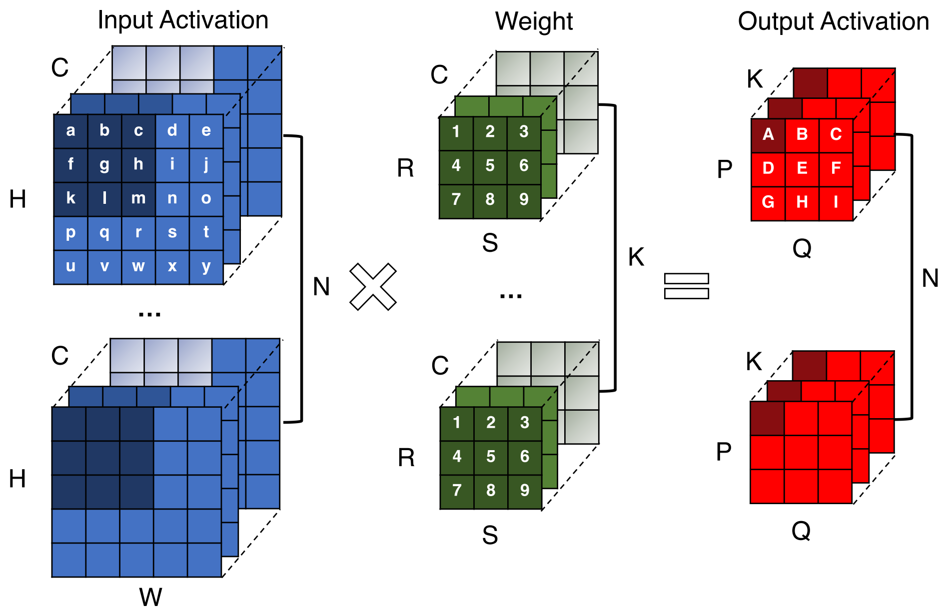
# 3-D Convolution




**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step
**padding:** # of zero rows/columns added

**C:** # of Input Channels
**K:** # of Output Channels
**N:** Batch size

# Convolution Loop Nest

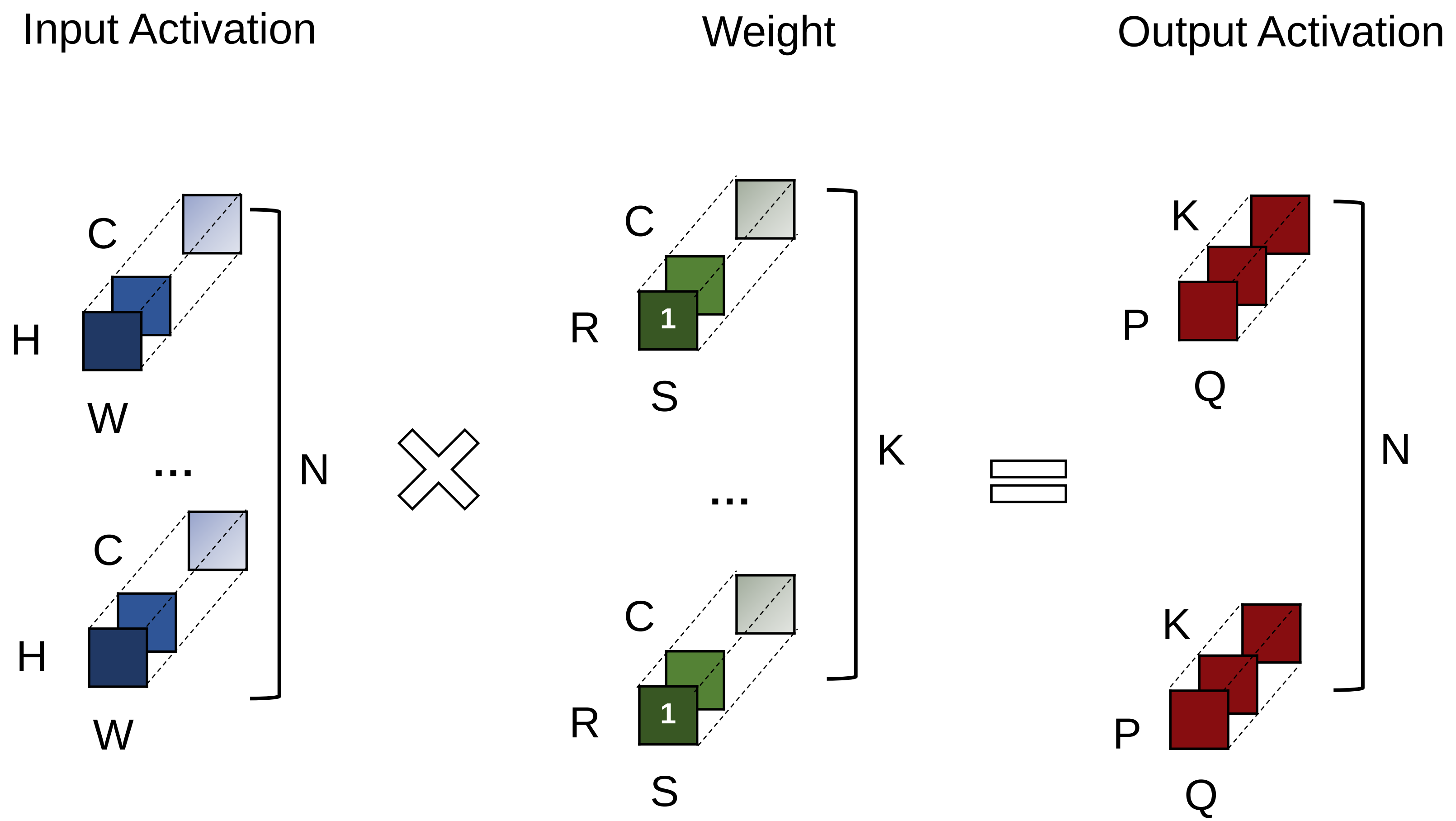



```
for (n=0; n<N; n++) {
  for (k=0; k<K; k++) {
    for (p=0; p<P; p++) {
      for (q=0; q<Q; q++) {
        OA[n][k][p][q]= 0;
        for (r=0; r<R; r++) {
          for (s=0; s<S; s++) {
            for (c=0; c<C; c++) {
              h = p * stride - pad + r;
              w = q * stride - pad + s;
              OA[n][k][p][q] +=
                            IA[n][c][h][w]
                            * W[k][c][r][s];
            }
          }
        }
        OA[n][k][p][q]= Activation(OA[n][k][p][q]);
      }
    }
  }
}
```

for each output activation

convolution window

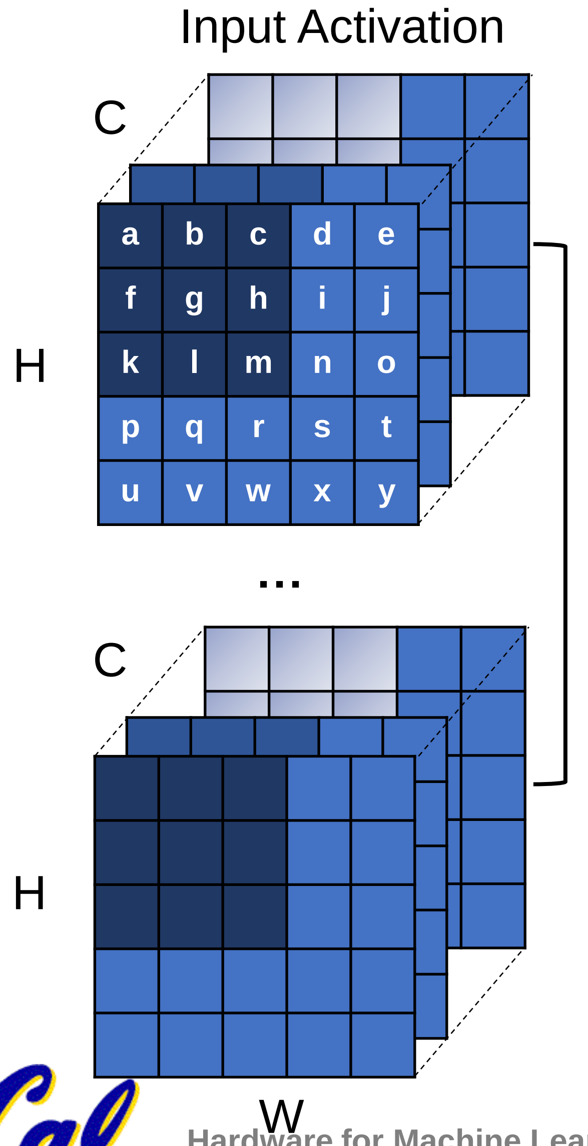
# Fully-Connected Layer

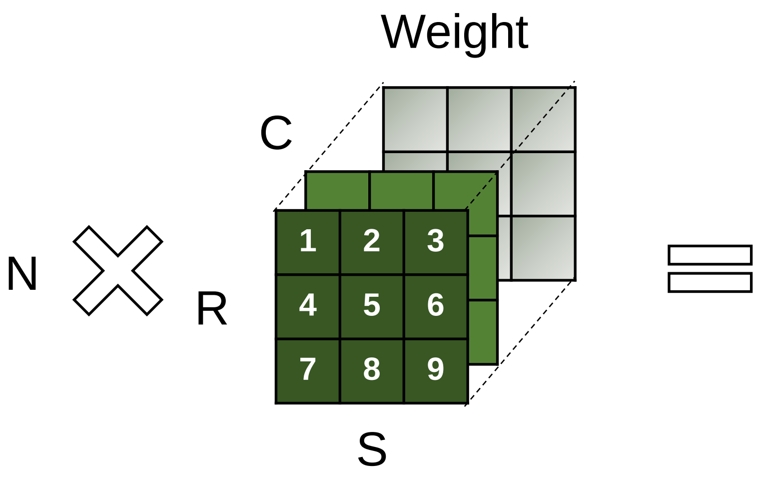


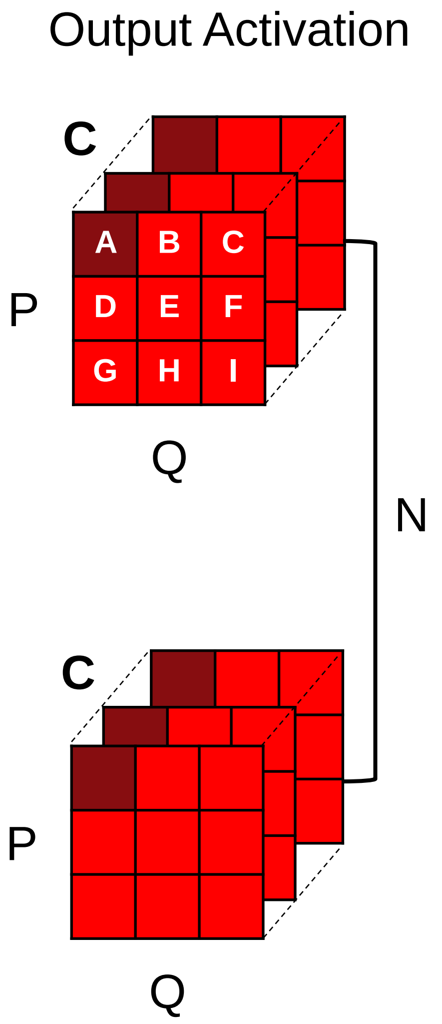

**H = 1**
**W = 1**
**R = 1**
**S = 1**
**P = 1**
**Q = 1**
**stride = 1**
**padding= 0**

**C:** # of Input Channels
**K:** # of Output Channels
**N:** Batch size

# Depth-wise Convolution



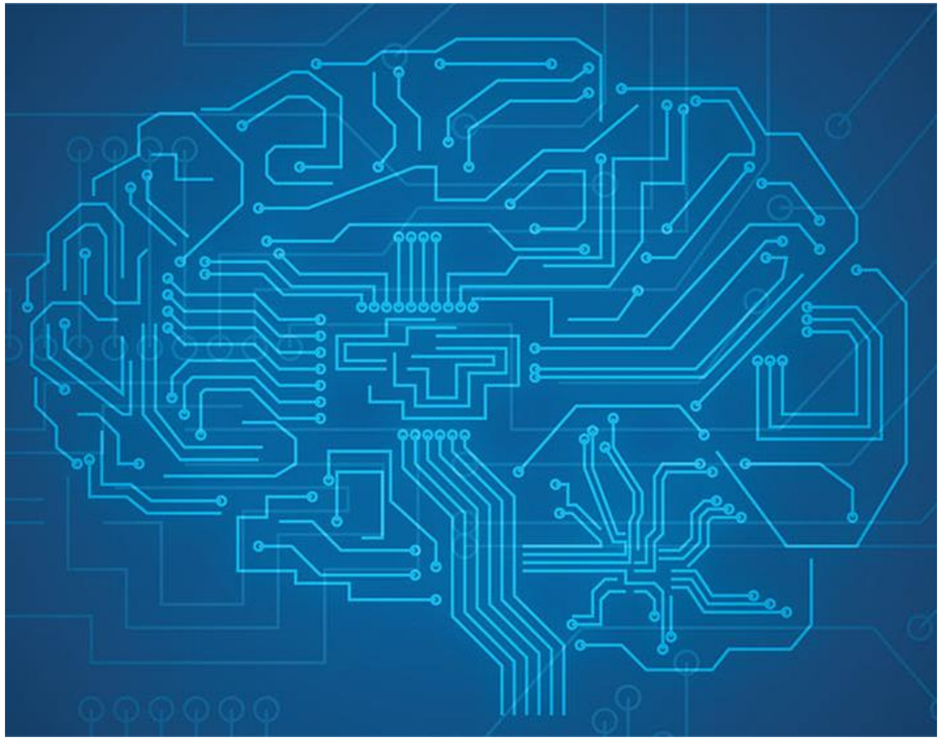

- Reduce weight size (KRSC -> RSC)
- Reduce # of Ops (RSC mul./out -> RS mul./out)

**H:** Height of Input Activation
**W:** Width of Input Activation
**R:** Height of Weight
**S:** Width of Weight
**P:** Height of Output Activation
**Q:** Width of Output Activation
**stride:** # of rows/columns traversed per step
**padding:** # of zero rows/columns added

**C:** # of Input Channels
**K:** # of Output Channels
**N:** Batch size

# Administrivia

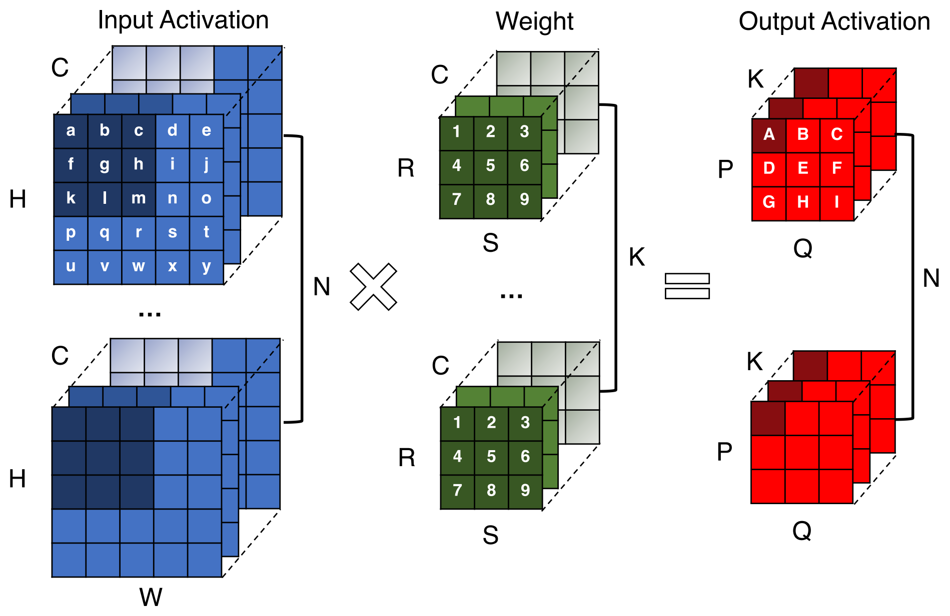
- Lab 1 due this Friday.
- Lab 2 will release next Monday.
- Week 3 reading is posted.



# DNN Kernels

- Overview
- Convolution
  - Basics
  - Transformation
- Pooling
- BatchNorm

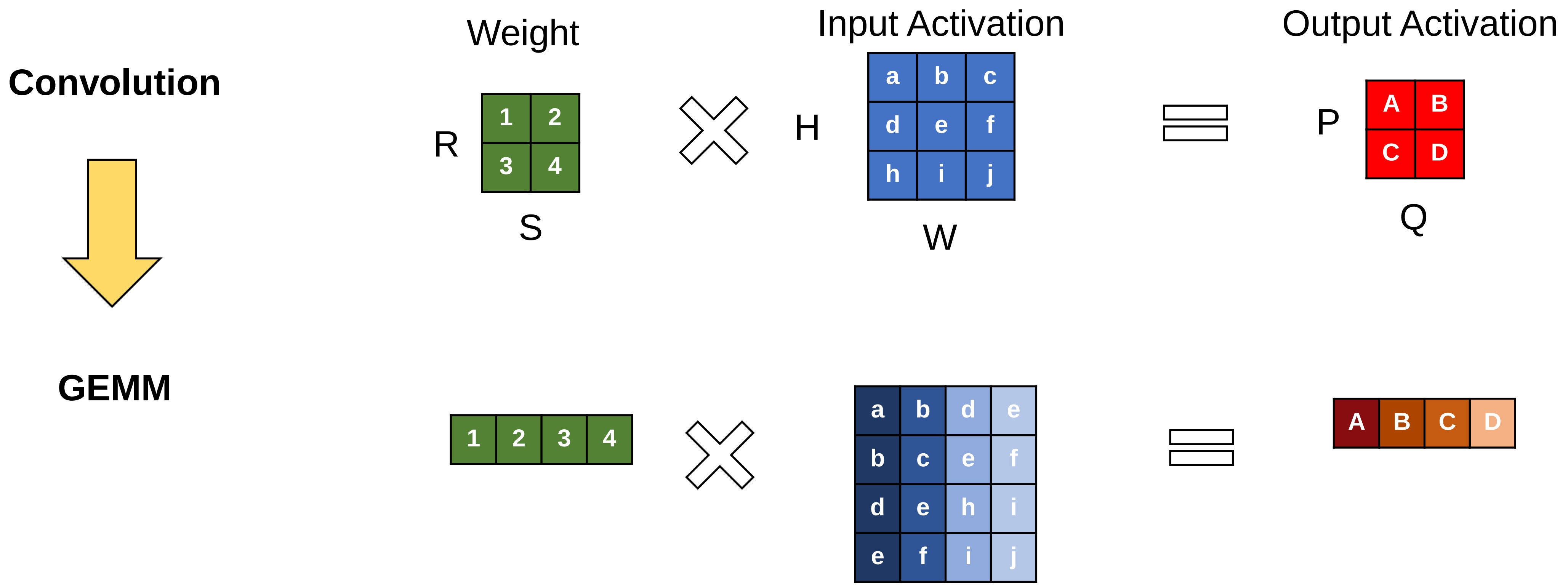
# Option 1: Direct Convolution




```
for (n=0; n<N; n++) {
  for (k=0; k<K; k++) {
    for (p=0; p<P; p++) {
      for (q=0; q<Q; q++) {
        OA[n][k][p][q]= 0;
        for (r=0; r<R; r++) {
          for (s=0; s<S; s++) {
            for (c=0; c<C; c++) {
              h = p * stride - pad + r;
              w = q * stride - pad + s;
              OA[n][k][p][q] +=
                             IA[n][c][h][w]
                             * W[k][c][r][s];
            }
          }
        }
        OA[n][k][p][q]= Activation(OA[n][k][p][q]);
      }
    }
  }
}
```

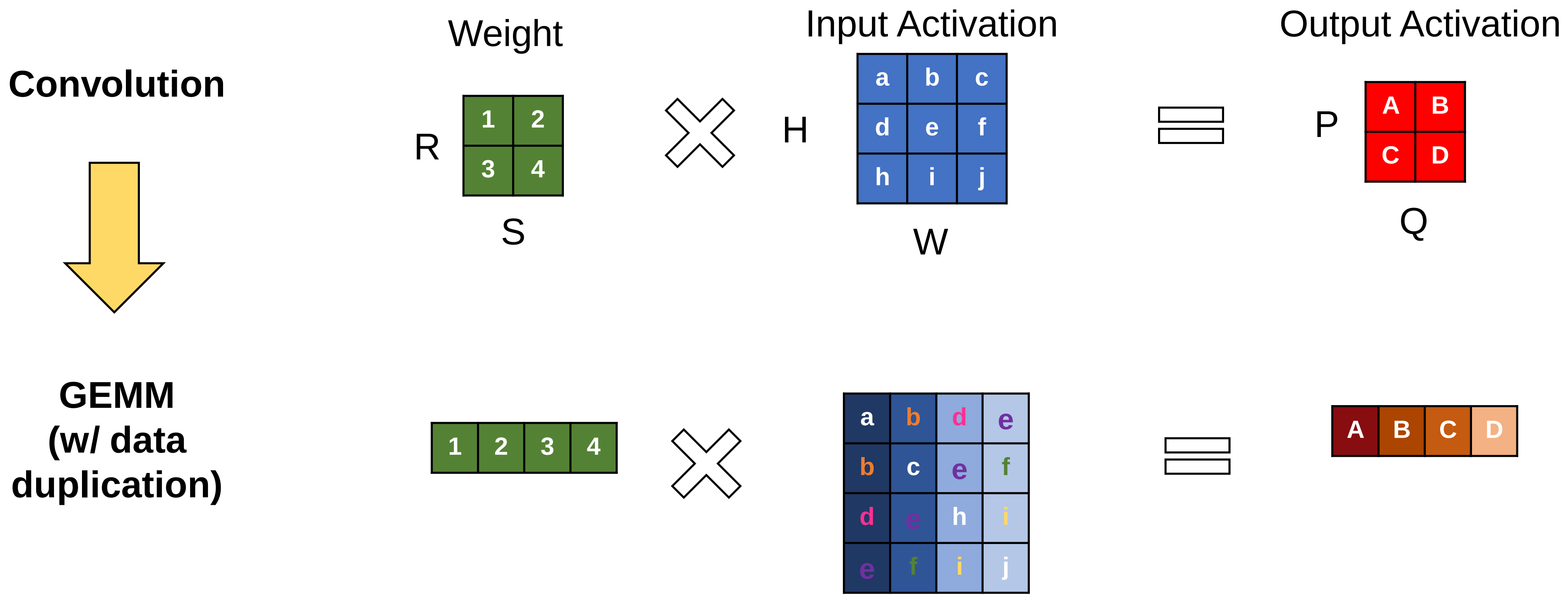
# Option 2: GEMM

- Converting convolution to GEMM via **im2col**



# Option 2: GEMM

- Converting convolution to GEMM via **im2col**



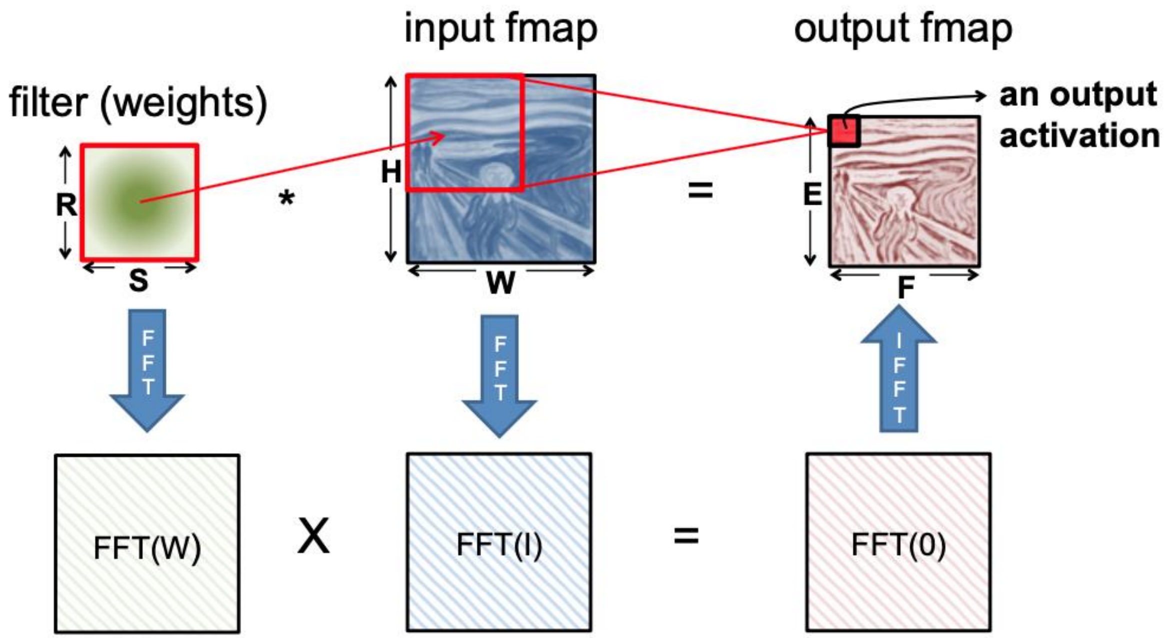
# Option 3: FFT-based Convolution

- **Convolution theorem**: convolution in the time domain is equivalent to point-wise multiply in the frequency domain.

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

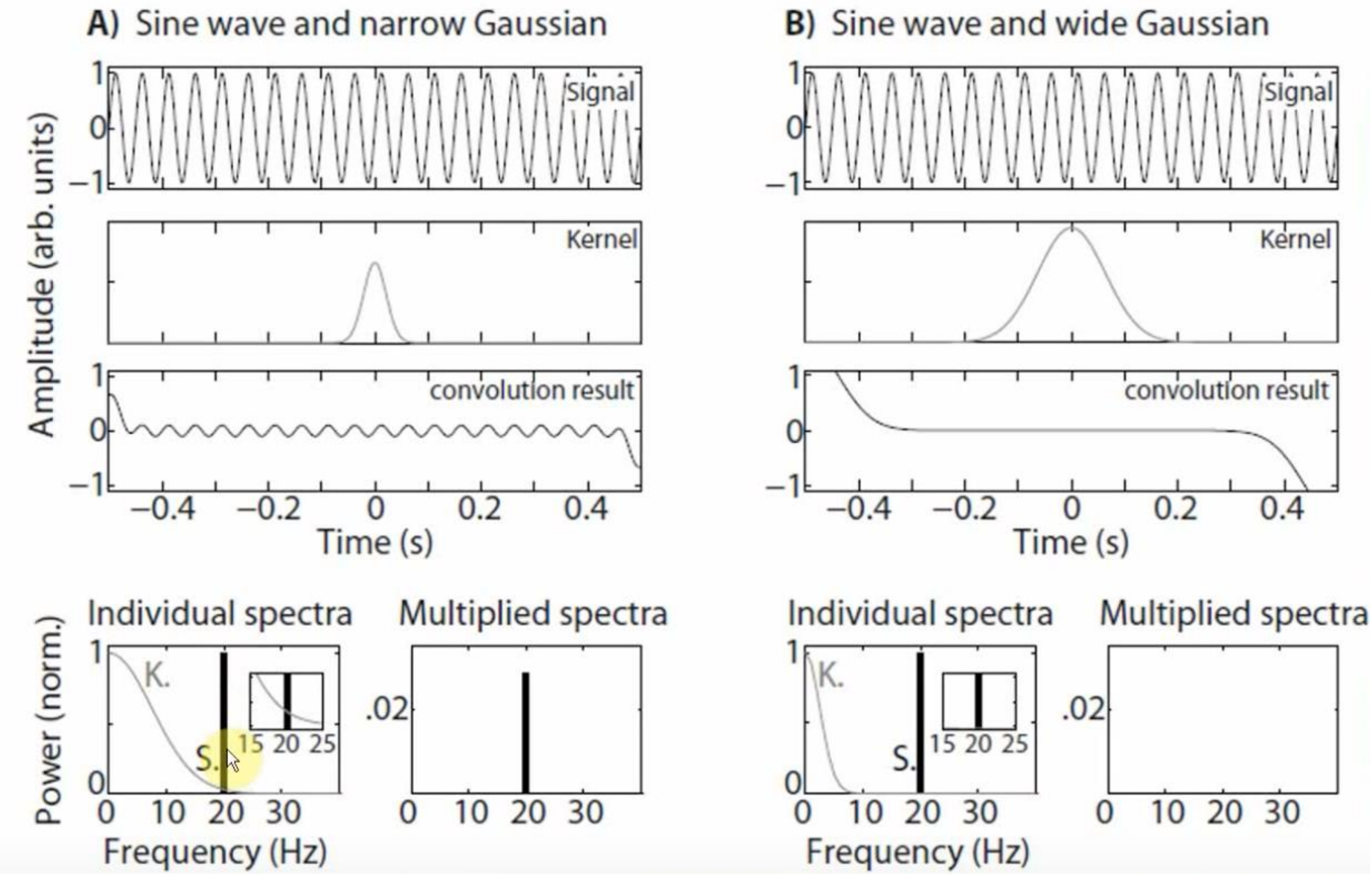
$\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$ are the Fourier transforms of $f$ and $g$

The asterisk denotes convolution, not multiplication.




Eyeriss tutorial

# Option 3: FFT-based Convolution



# Option 3: FFT-based Convolution

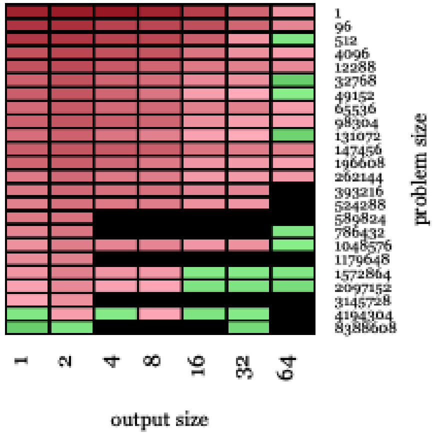


Figure 1: 3 × 3 kernel (K40m)

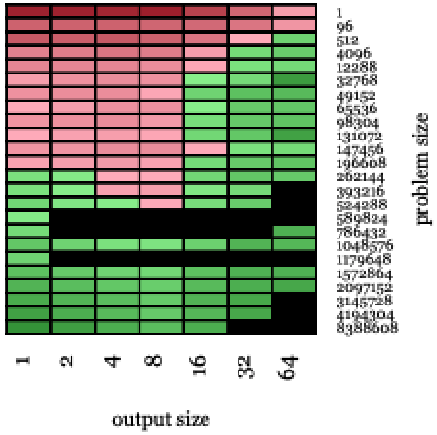


Figure 2: 5 × 5 kernel (K40m)

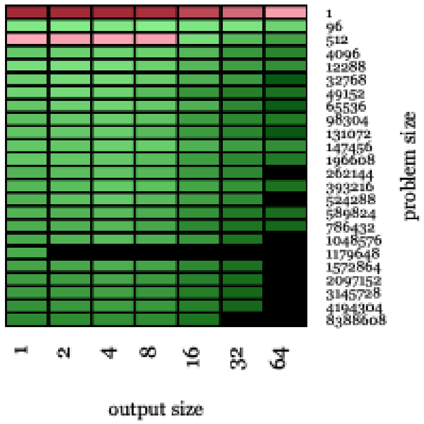


Figure 5: 11 × 11 kernel (K40m)

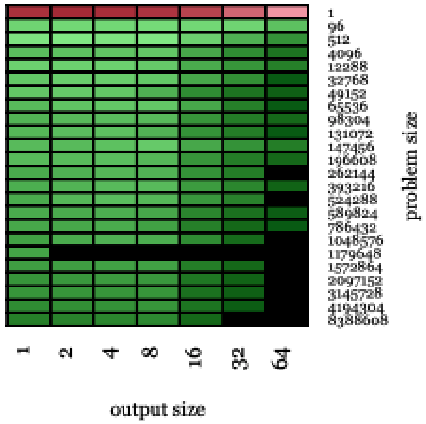


Figure 6: 13 × 13 kernel (K40m)

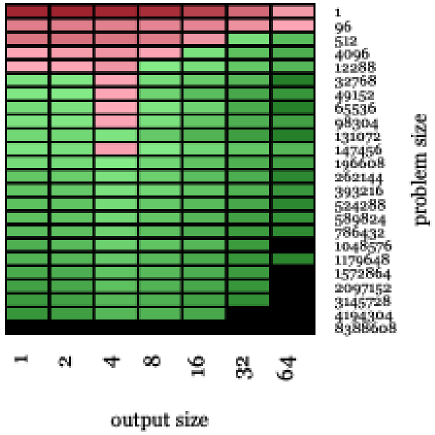


Figure 3: 7 × 7 kernel (K40m)

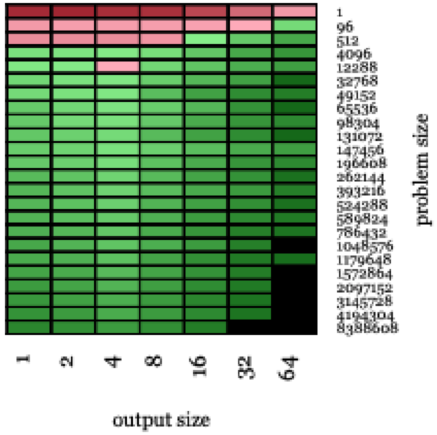
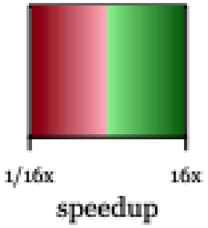


Figure 4: 9 × 9 kernel (K40m)




*Fast Convolutional Nets with fbfft: A GPU Performance Evaluation*

# Option 4: Winograd Transform

- Re-association of intermediate values to reduce # of multiplications.
- Works well for 3x3 convolution.

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \tag{5}$$

where

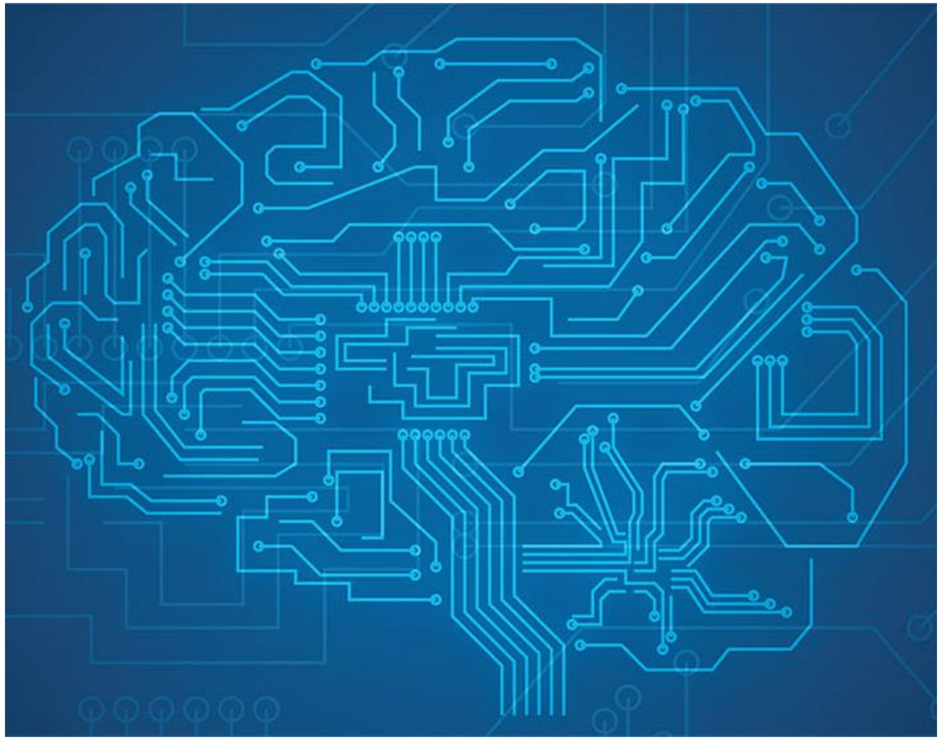
$$m_1 = (d_0 - d_2)g_0 \qquad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$
$$m_4 = (d_1 - d_3)g_2 \qquad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

Before: 6 MULs, 4 ADDs
After:
- IA (d): 4 ADDs
- W (g): 3 ADDs, 2 MULs
- OA (m): 4 MULs, 4 ADDs

$$Y = A^T\left[(Gg) \odot (B^T d)\right] \tag{6}$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$
$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$
$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$
$$g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T$$
$$d = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}^T$$

*Fast Algorithms for Convolutional Neural Networks*

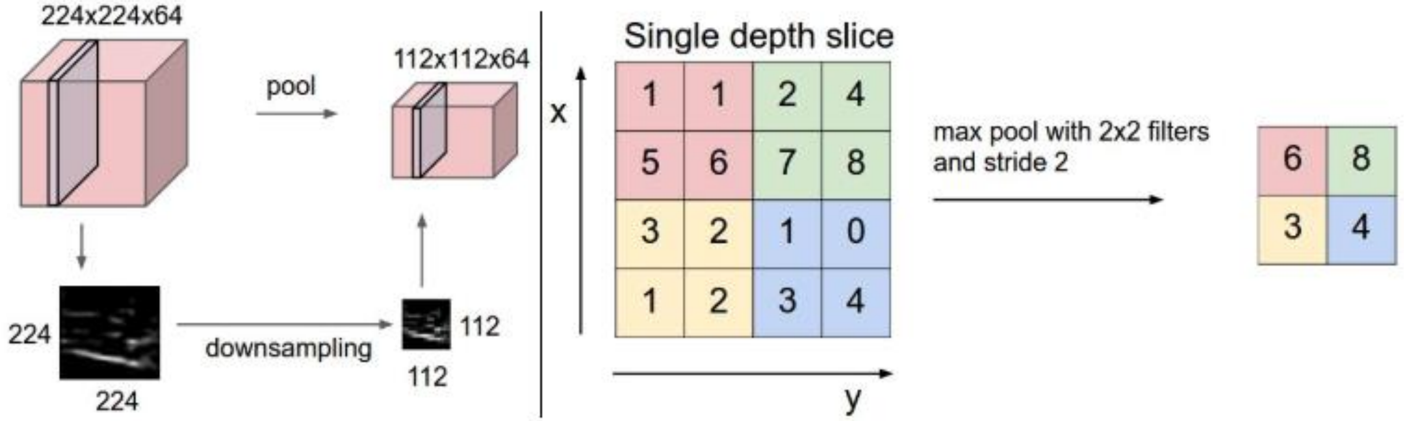


# DNN Kernels

- Overview
- Convolution
  - Basics
  - Transformation
- **Pooling**
- BatchNorm

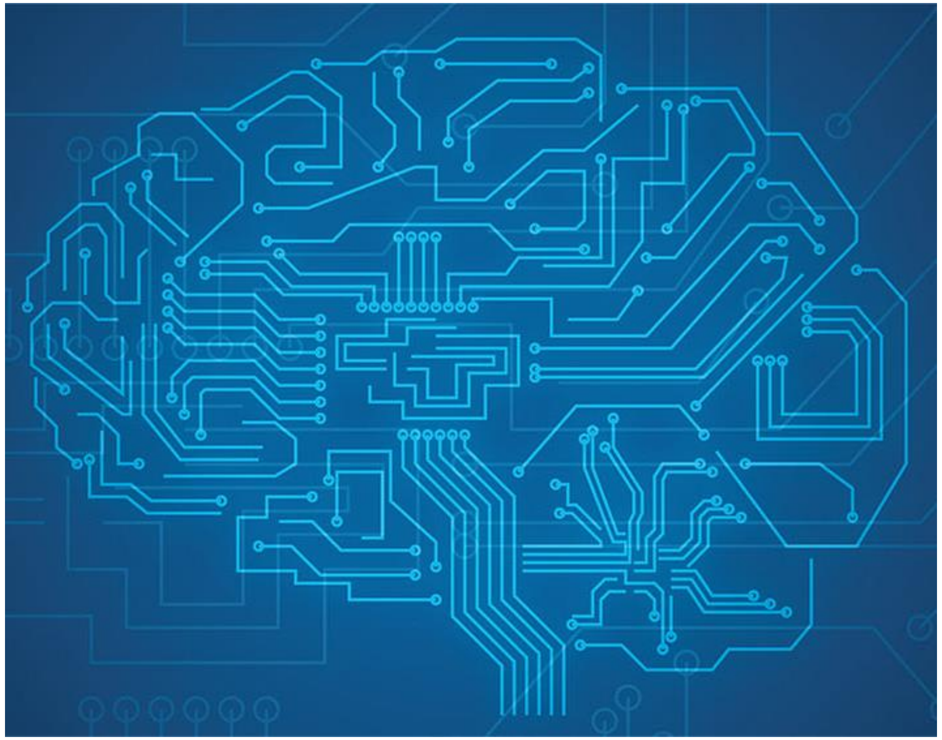
# Pooling Layer

- Progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- Parameters:
  - Type: MAX and AVG
  - Pooling kernel size
  - Pooling stride




Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

*cs231n notes*

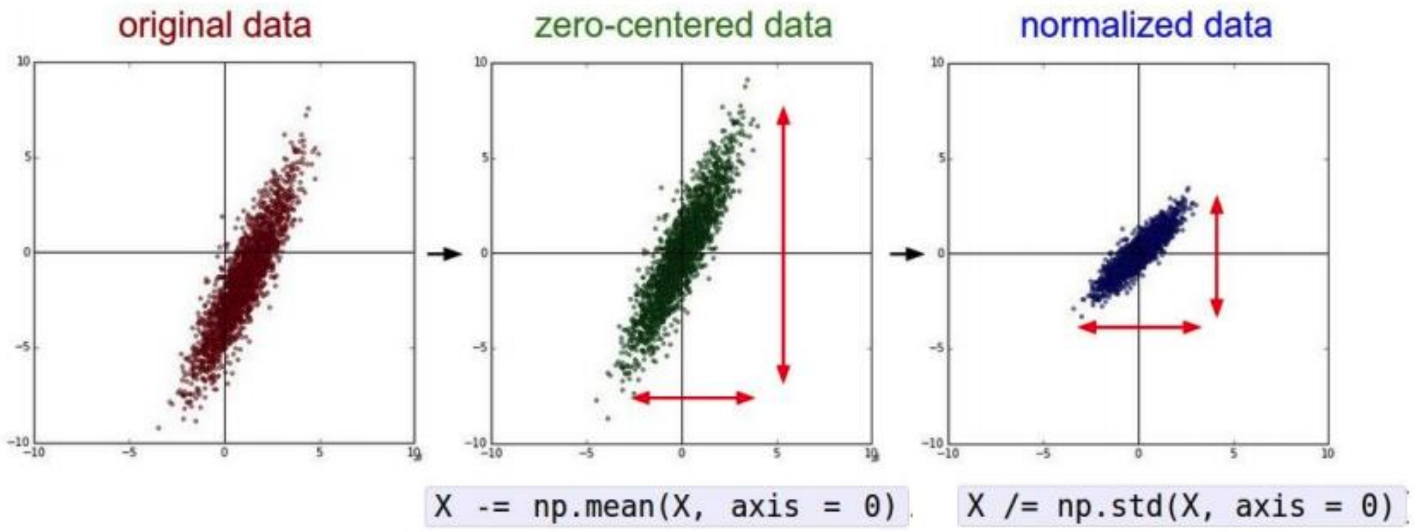


# DNN Kernels

- Overview
- Convolution
  - Basics
  - Transformation
- Pooling
- **BatchNorm**

# BatchNorm Layer

- Goal: make it easier to train by providing zero-mean, unit-variance activations.
  - The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.




*Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*

# BatchNorm Layer

**Training Time**

**Input:** $x : N \times D$

**Learnable scale and shift parameters:**

$\gamma, \beta : D$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

**Test Time**

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

# Review

- Deep neural networks typically have a sequence of convolutional, fully-connected, pooling, batch normalization, and activation layers.
- Convolution is one of the fundamental kernel in DNNs.
  - 2-D convolution
  - Stride and padding
  - 3-D convolution with input/output channels
  - Batch size
- Convolution can be calculated in different ways.
  - Direct, GEMM, FFT-based, Winograd-based.
- Pooling and Batch Normalization layers.