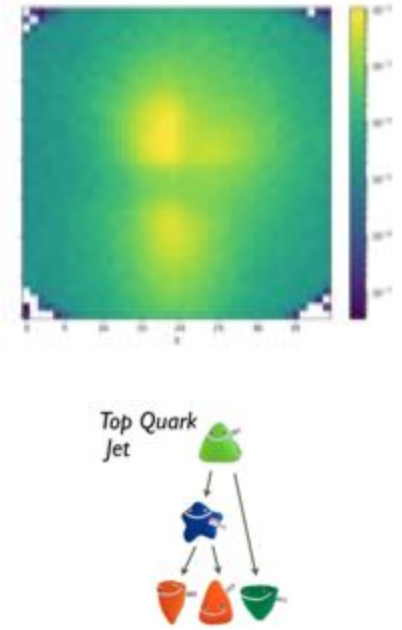
Convolutional Neural Network

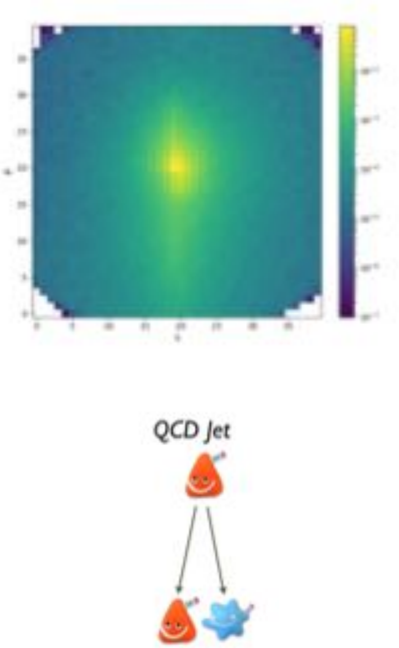
PHYS591000 Spring 2021

Reading:

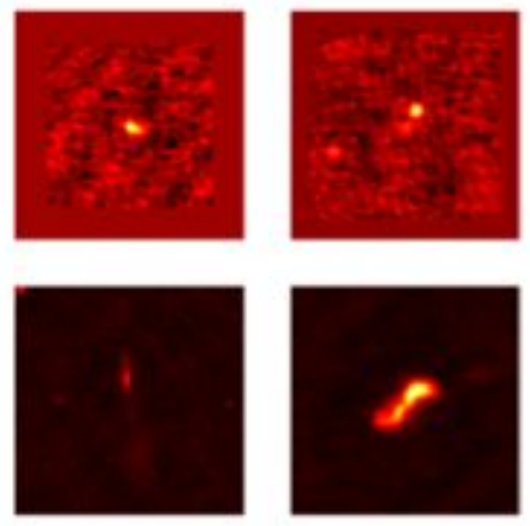
- Shervine's CNN Cheatsheet for CS-230
- Sharma's CNN tutorial blog

Physics with Image Processing





Radio Galaxy

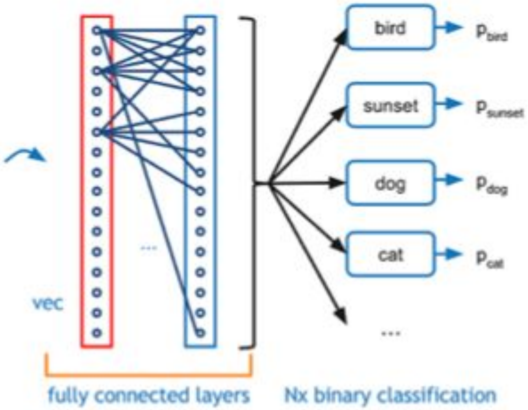


M. Brueggen

Computer Vision

- How can we classify an object when input data get really big?

Size = 32   Color = 3 (Red, Green, Blue)



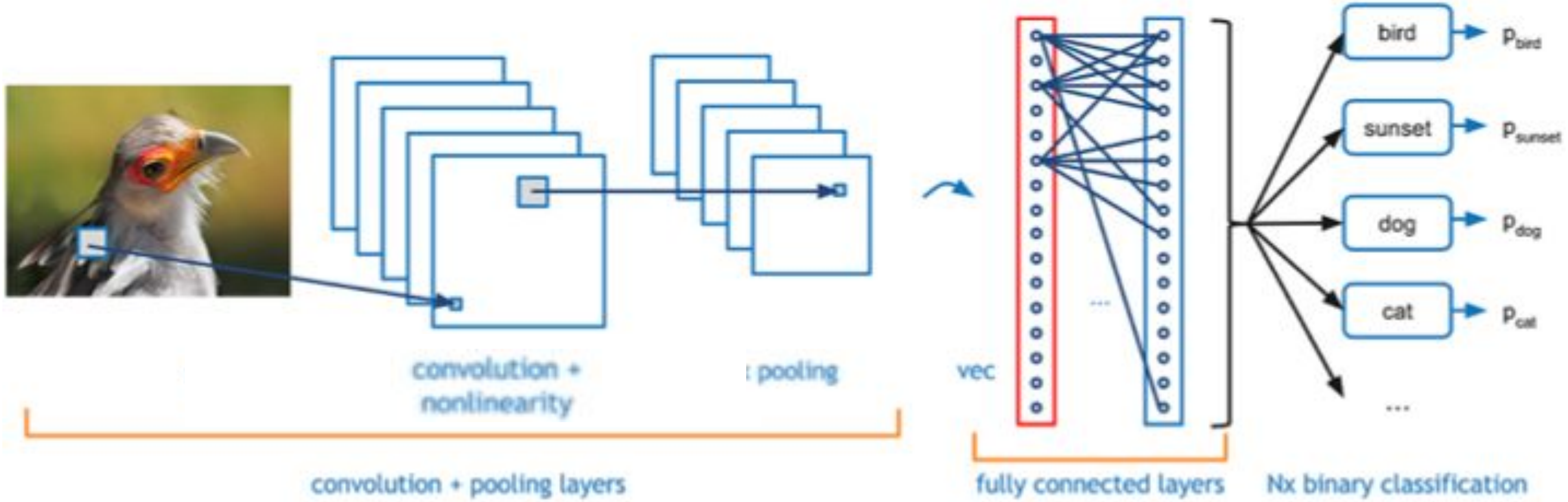
Input N=32 X 32 X 3=3072

N   M (Neurons)

Hyper parameters   P=(N+1)xM = (3072+1)x3072=9.4M

Convolution Nets

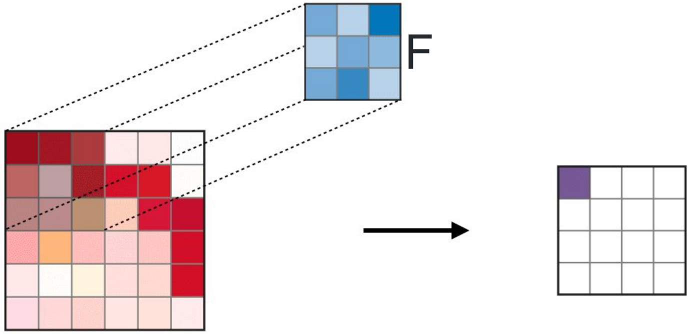
Detect key features with the use of filter (or kernel) to reduce dimensionality.

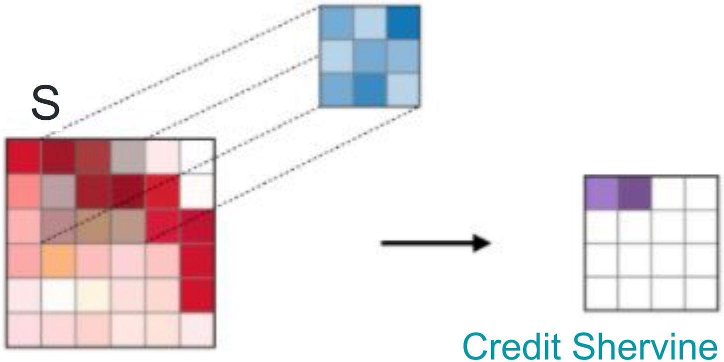


- Learning Goal:
  - How to calculate the tensor size at each stage?
  - How to calculate the total number of parameters in the network?

Convolution Layer (CONV)

- The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input data with respect to its dimensions.
- Parameters
  - Filter size F
  - Padding P
  - Stride S

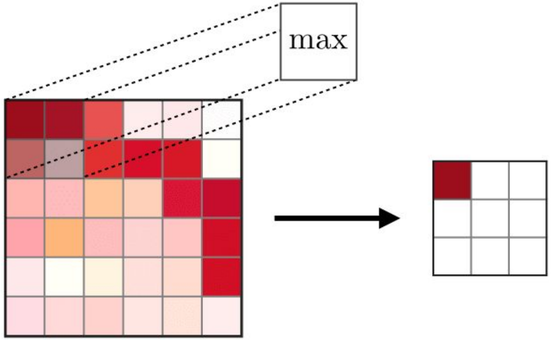


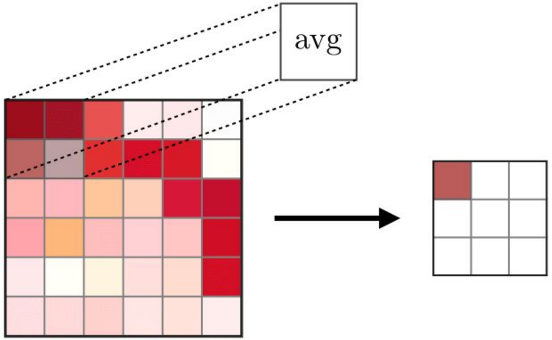


[Credit Shervine CS-230]()

Pooling (POOL)

- The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance.
- Parameters
  - Filter size F
  - Stride S

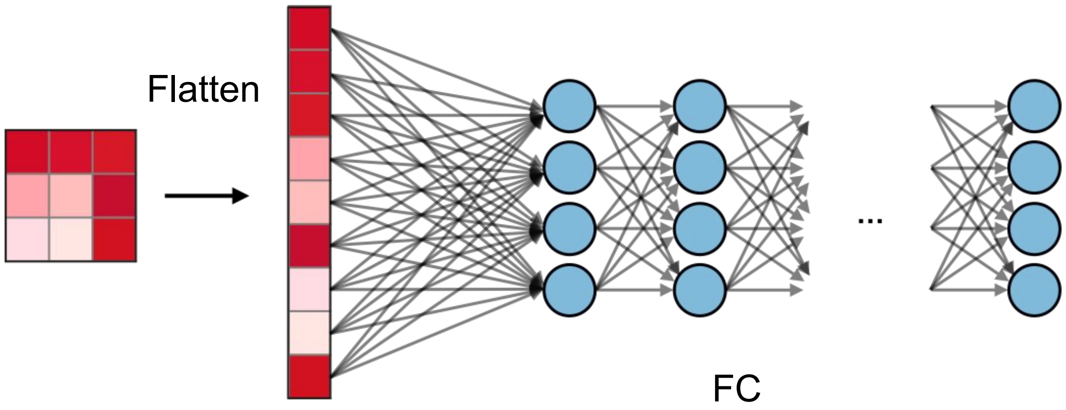




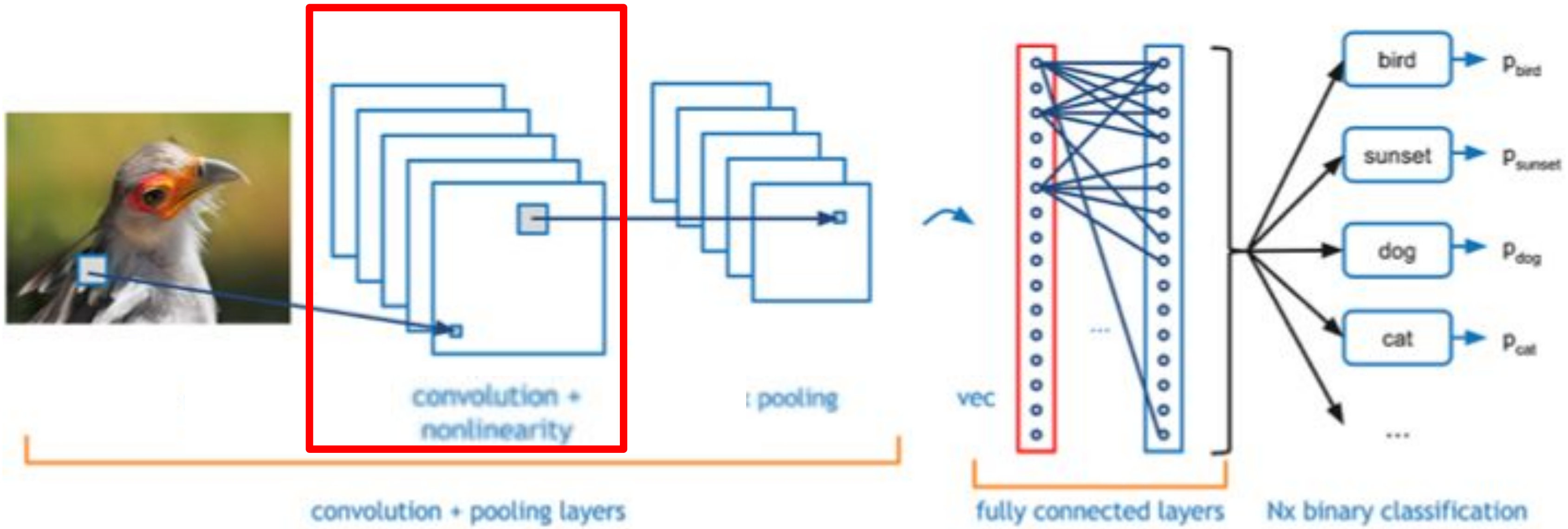
Credit Shervine CS-230

Full Connected (FC)

The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons.



Credit Shervine CS-230



convolution +
nonlinearity
pooling
vec
bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
convolution + pooling layers
fully connected layers
Nx binary classification

Convolution Filter

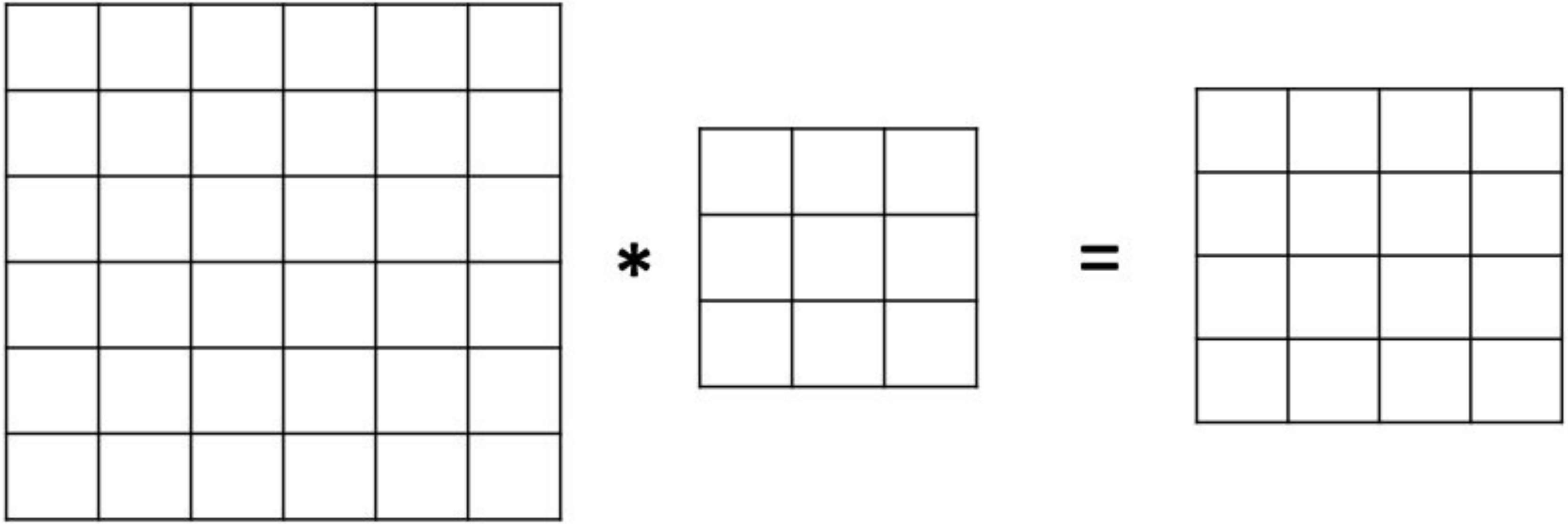


Image
Input

Filter
Kernel

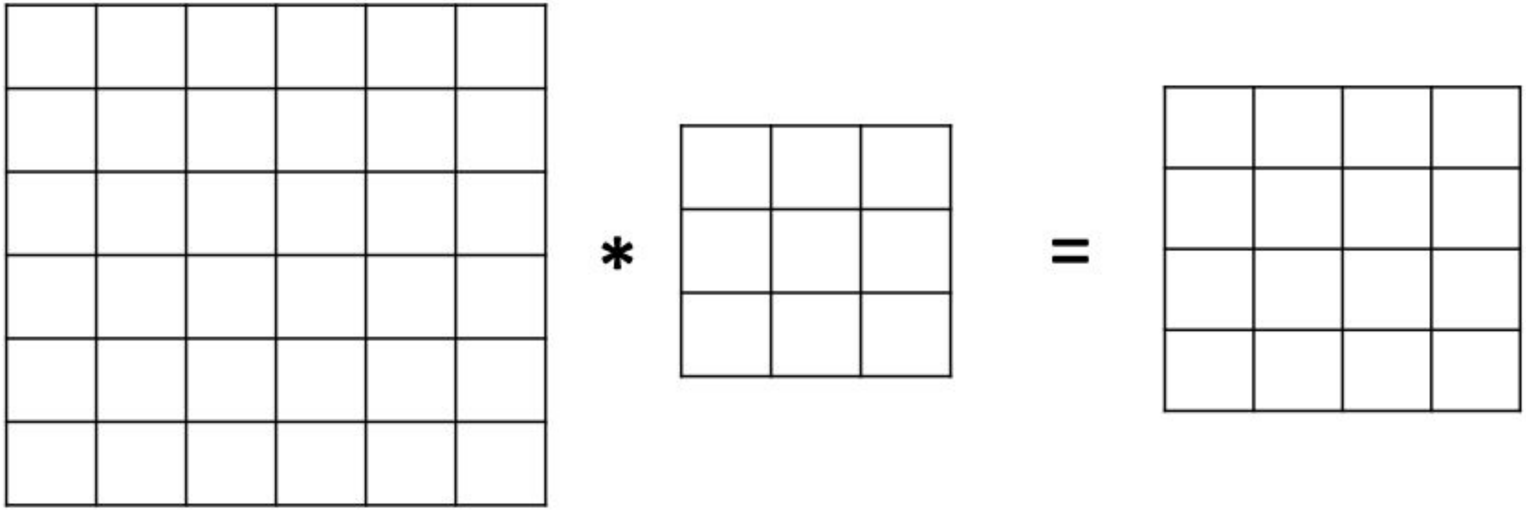
Feature map
Activation map
Output

Feature map size

Input size: I

Filter size: F

Output size: O



I x I

F x F

O x O

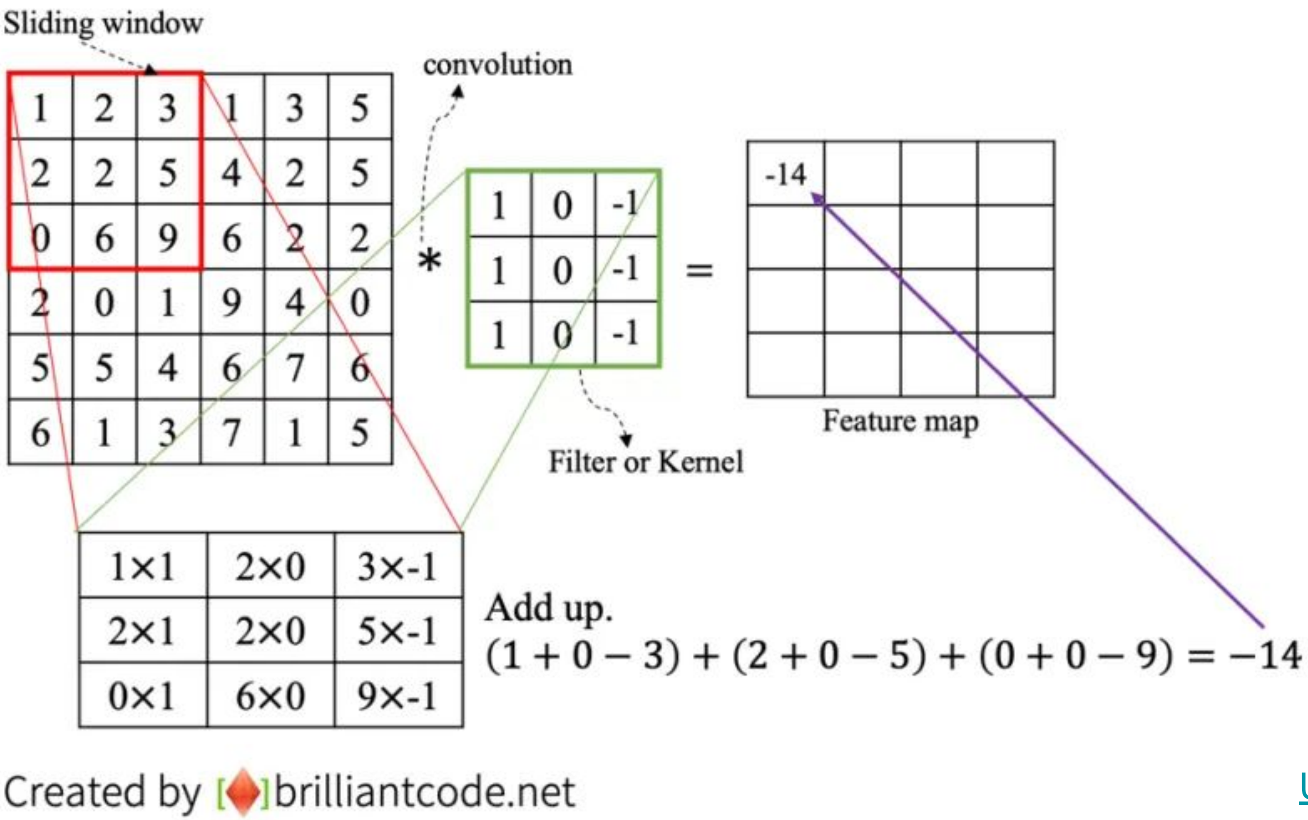
O= I - F+1

Convolution Filter as Edge Detector

$$\begin{bmatrix} 1 & 2 & 3 & 1 & 3 & 5 \\ 2 & 2 & 5 & 4 & 2 & 5 \\ 0 & 6 & 9 & 6 & 2 & 2 \\ 2 & 0 & 1 & 9 & 4 & 0 \\ 5 & 5 & 4 & 6 & 7 & 6 \\ 6 & 1 & 3 & 7 & 1 & 5 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}$$

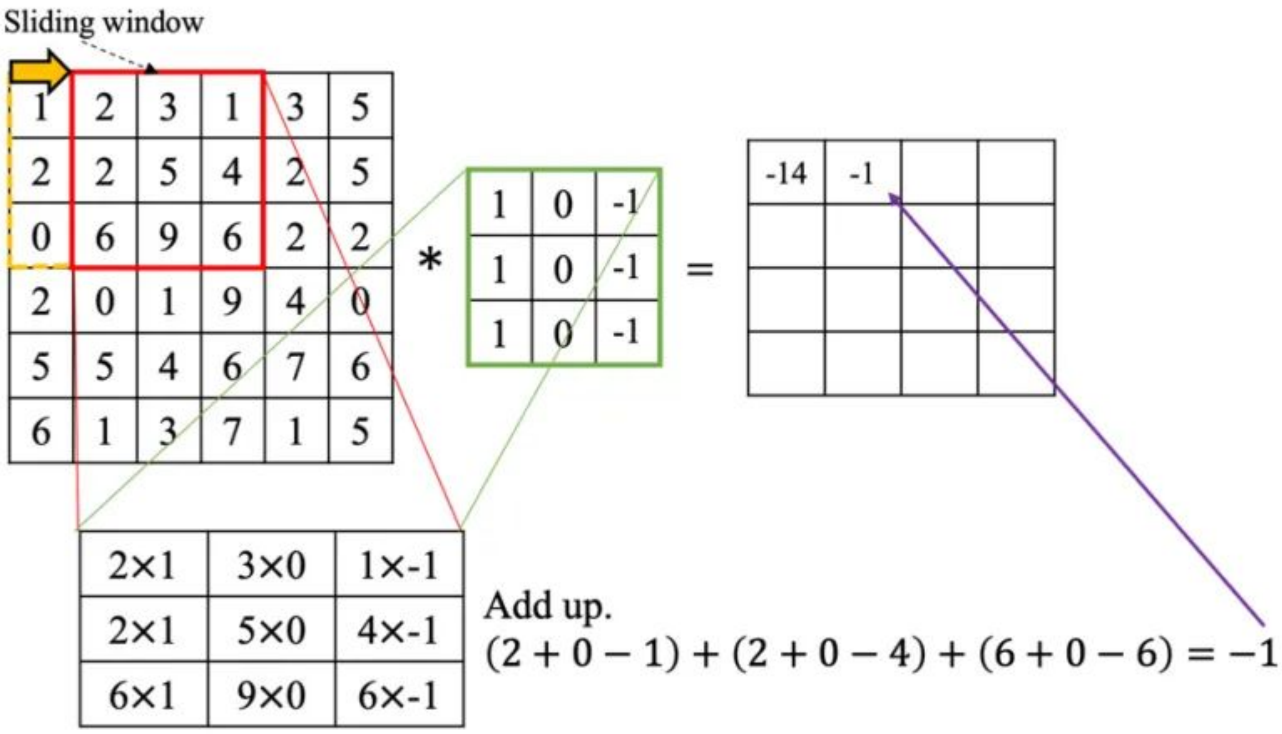
O= I - F + 1
= 6 - 3 +1 = 4

Convolution Calculation



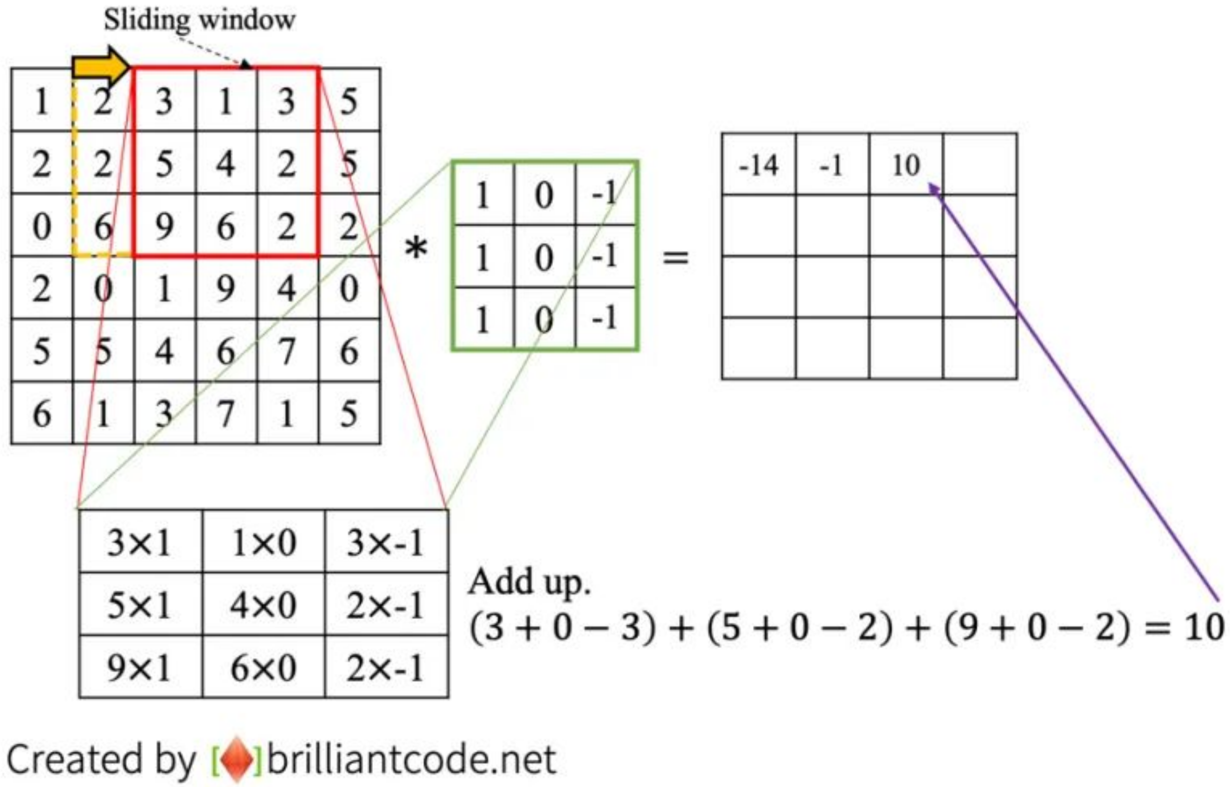
URL

Convolution Calculation

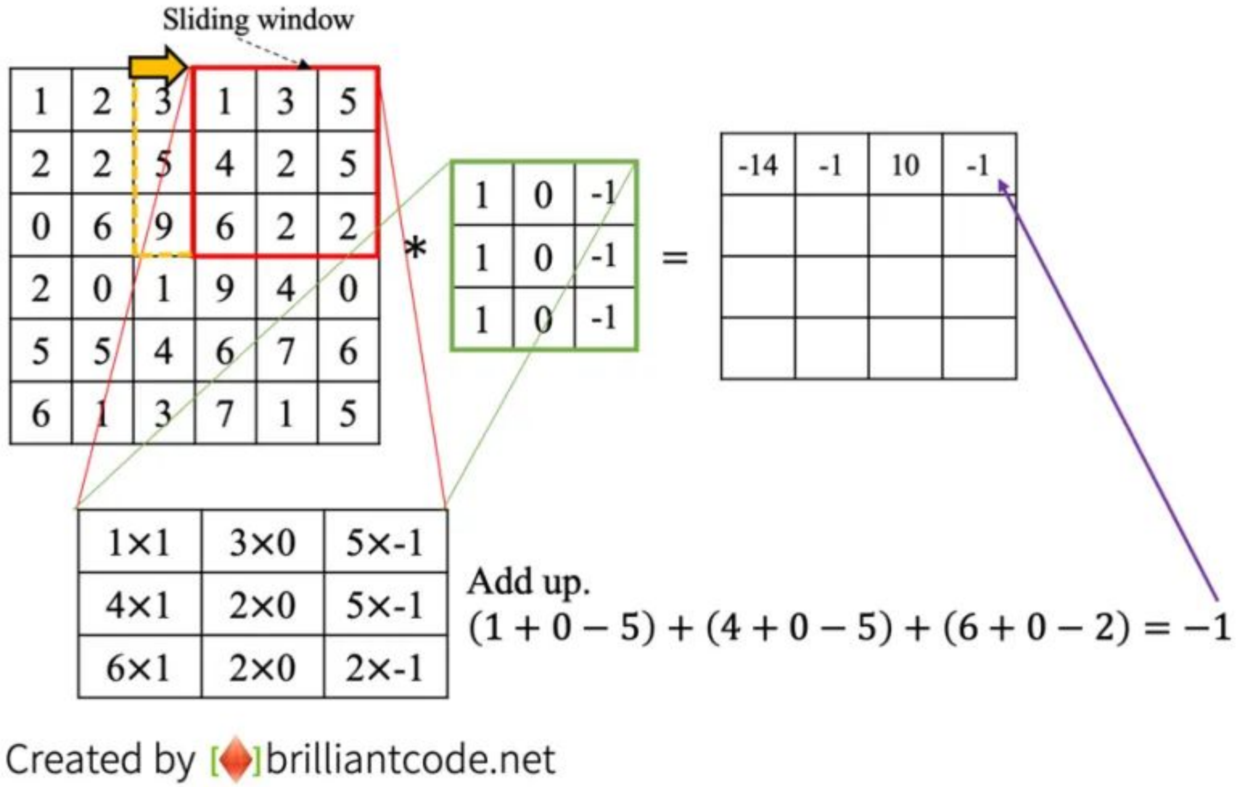


Created by [◆]brilliantcode.net

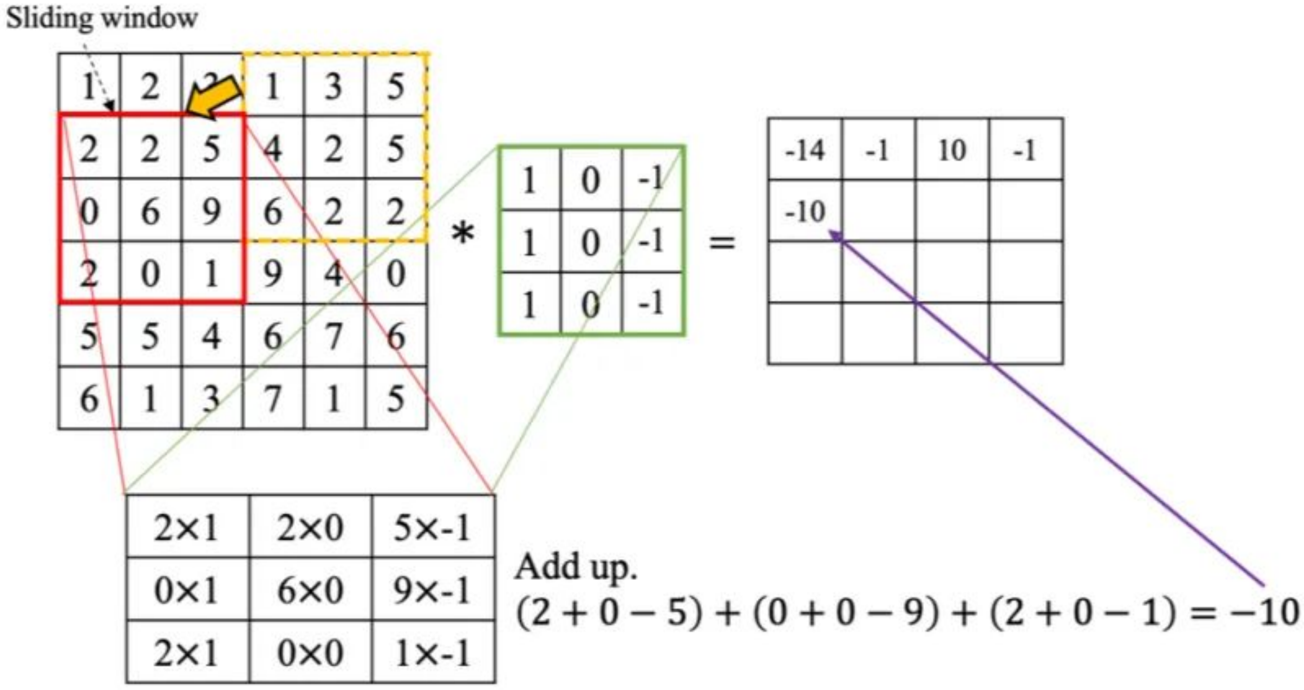
Convolution Calculation



Convolution Calculation

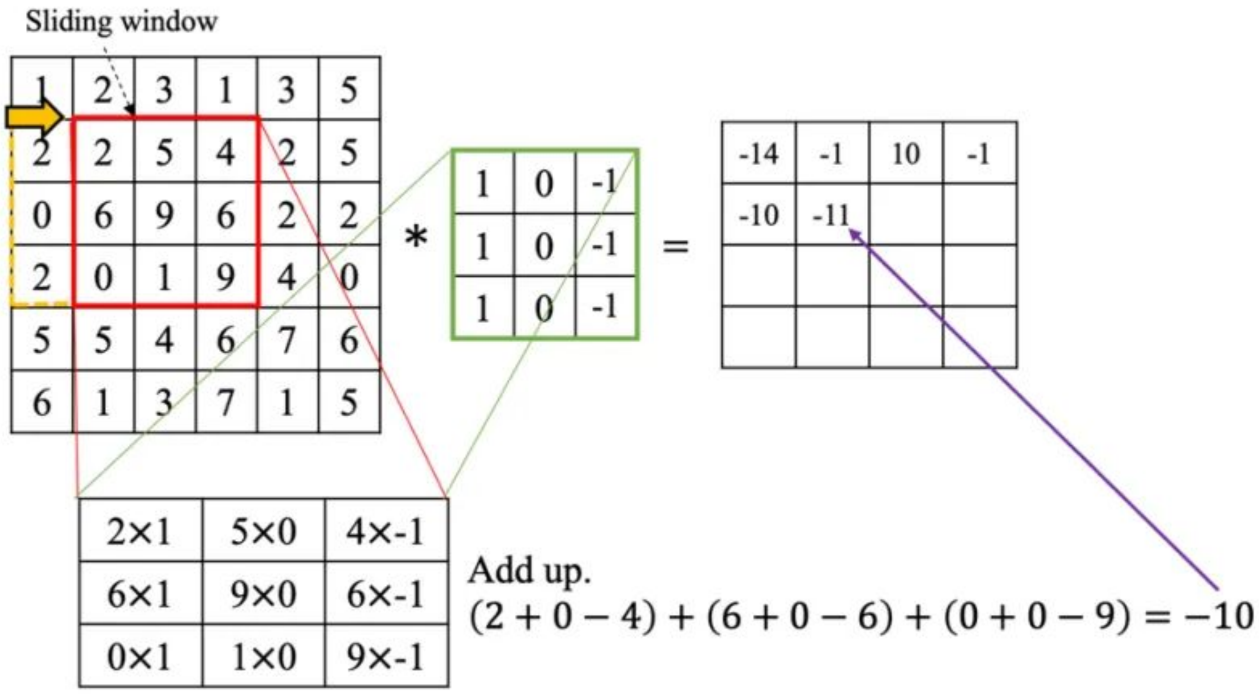


Convolution Calculation



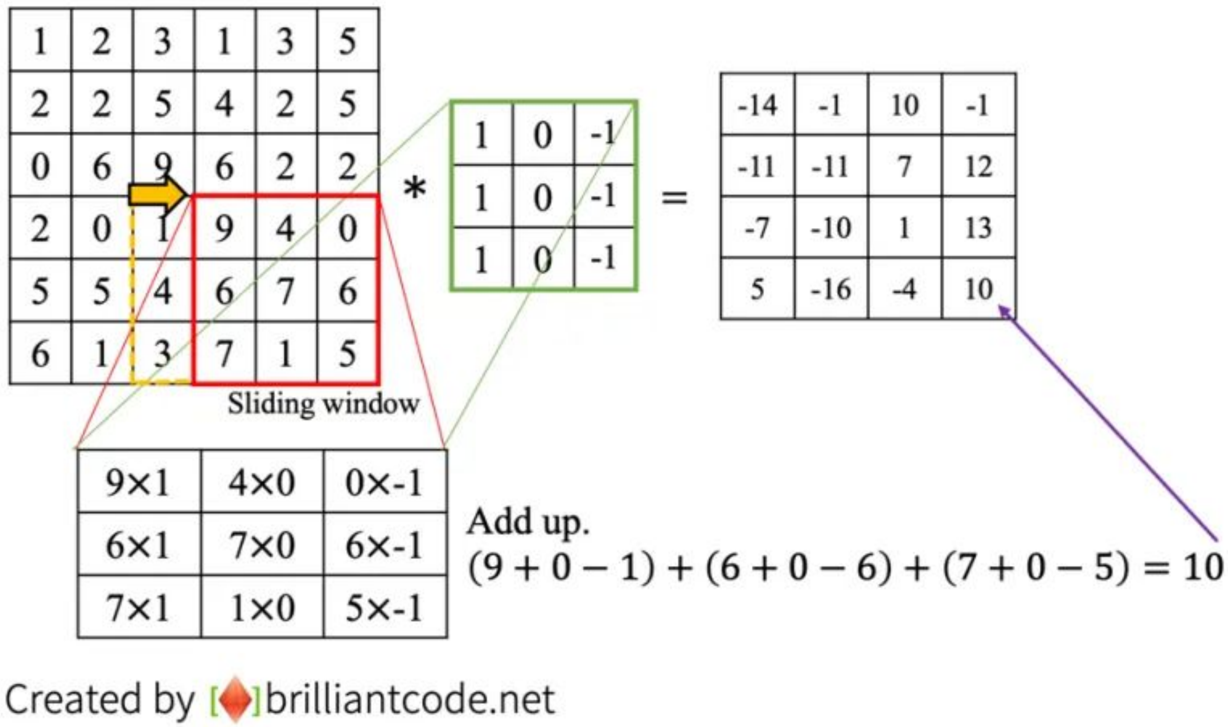
Created by brilliantcode.net

Convolution Calculation



Created by brilliantcode.net

Convolution Calculation



Convolution Filter as Edge Detector

1	2	3	1	3	5
2	2	5	4	2	5
0	6	9	6	2	2
2	0	1	9	4	0
5	5	4	6	7	6
6	1	3	7	1	5

*

1	0	-1
1	0	-1
1	0	-1

=

-14	-1	10	-1
-11	-11	7	12
-7	-10	1	13
5	-16	-4	10

Image
Input

Filter
Kernel

Feature map
Activation map
Output

Filters

Original

Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Edge

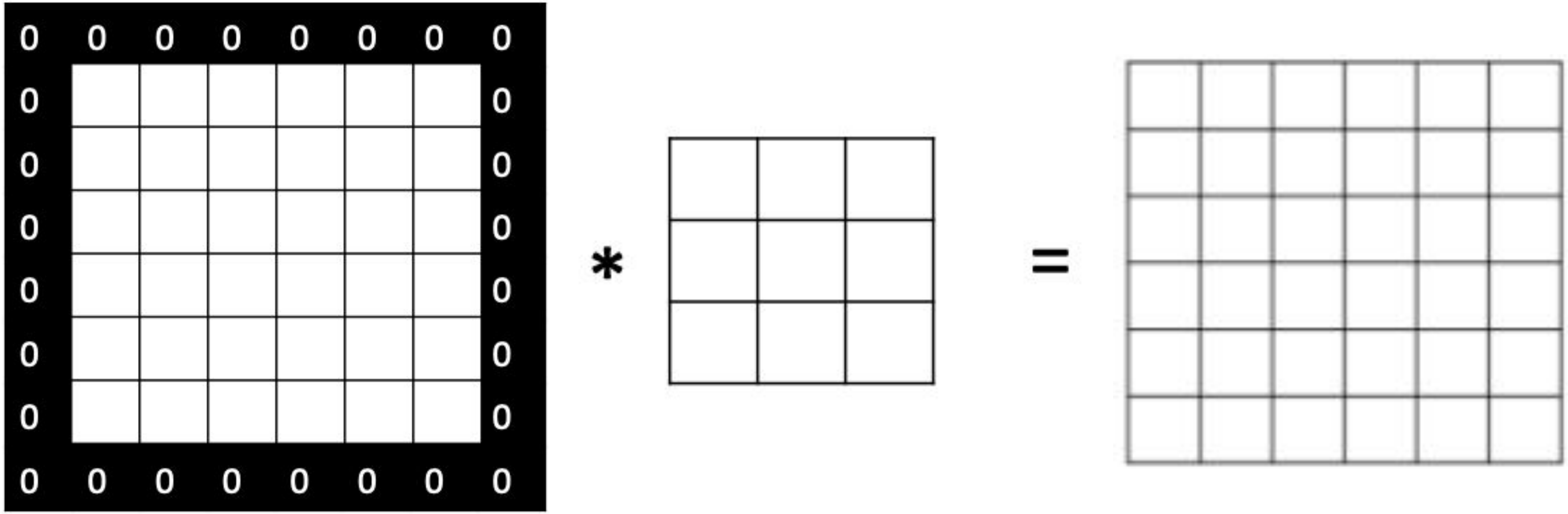
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Gaussian blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Padding

Padding size: P



(I+2P) x (I+2P)

F x F

O x O

O= (I + 2P- F+1)

Same Image size condition: P = (F-1)/2

Padding Example

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 1 & 3 & 5 & 0 \\
0 & 2 & 2 & 5 & 4 & 2 & 5 & 0 \\
0 & 0 & 6 & 9 & 6 & 2 & 2 & 0 \\
0 & 2 & 0 & 1 & 9 & 4 & 0 & 0 \\
0 & 5 & 5 & 4 & 6 & 7 & 6 & 0 \\
0 & 6 & 1 & 3 & 7 & 1 & 5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
\boxed{-4} & -5 & -1 & 3 & -5 & 5 \\
-10 & -14 & -1 & 10 & -1 & 7 \\
-8 & -11 & -11 & 7 & 12 & 8 \\
11 & -7 & -10 & 1 & 13 & 13 \\
-6 & 5 & -16 & -4 & 10 & 12 \\
-6 & 4 & -7 & -1 & 2 & 8
\end{bmatrix}
$$

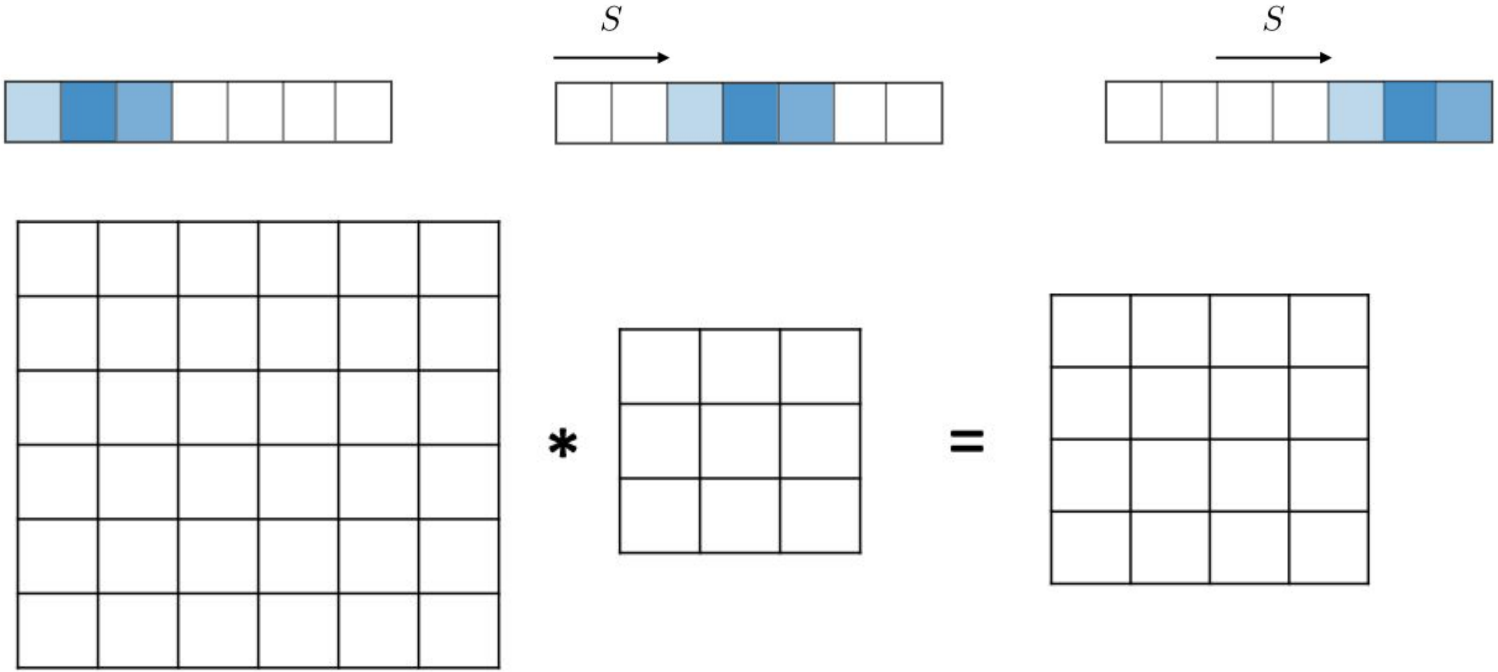
$$
\begin{aligned}
O &= (I + 2P - F + 1) \\
&= (6 + 2 \times 1 - 3 + 1) = 6
\end{aligned}
$$

Strides

the stride S denotes the number of pixels by which the window moves after each operation.

Strides

the stride S denotes the number of pixels by which the window moves after each operation.

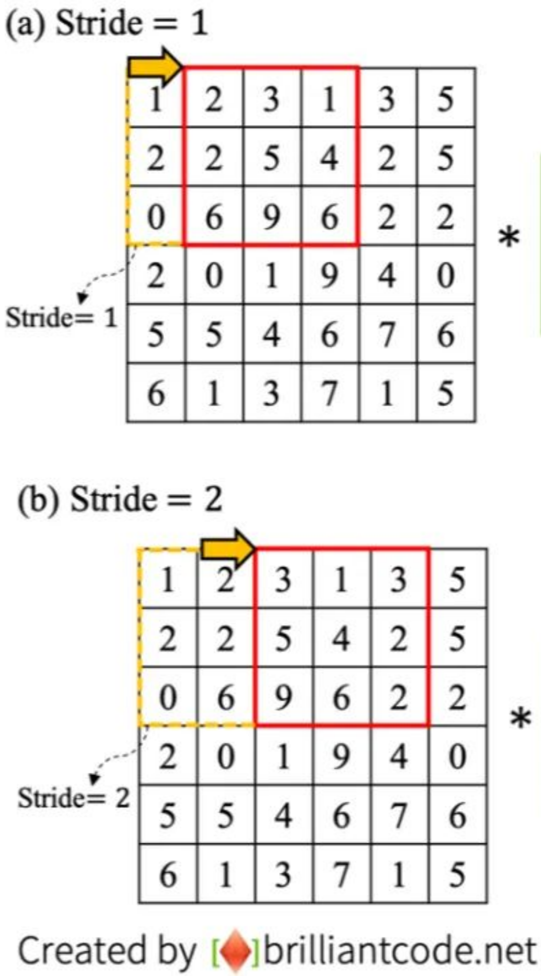


I x I

F x F

O = floor [(I-F+2P)/S+1]

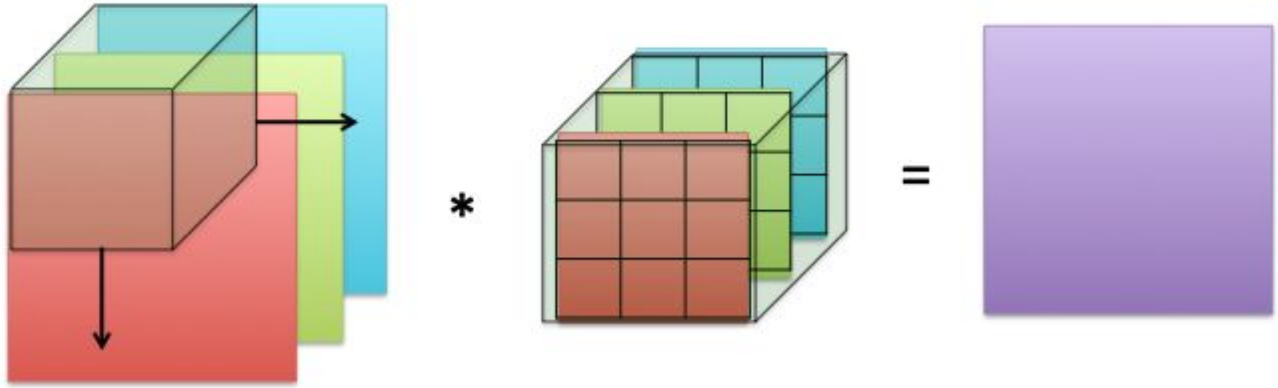
Strides



O = floor [(I-F+2P)/S+1]
= floor [(6-3+2x0)/2+1]
= floor [2.5] =2

Volume Convolutions with multiple Channels

Channel number: C



I x I x C

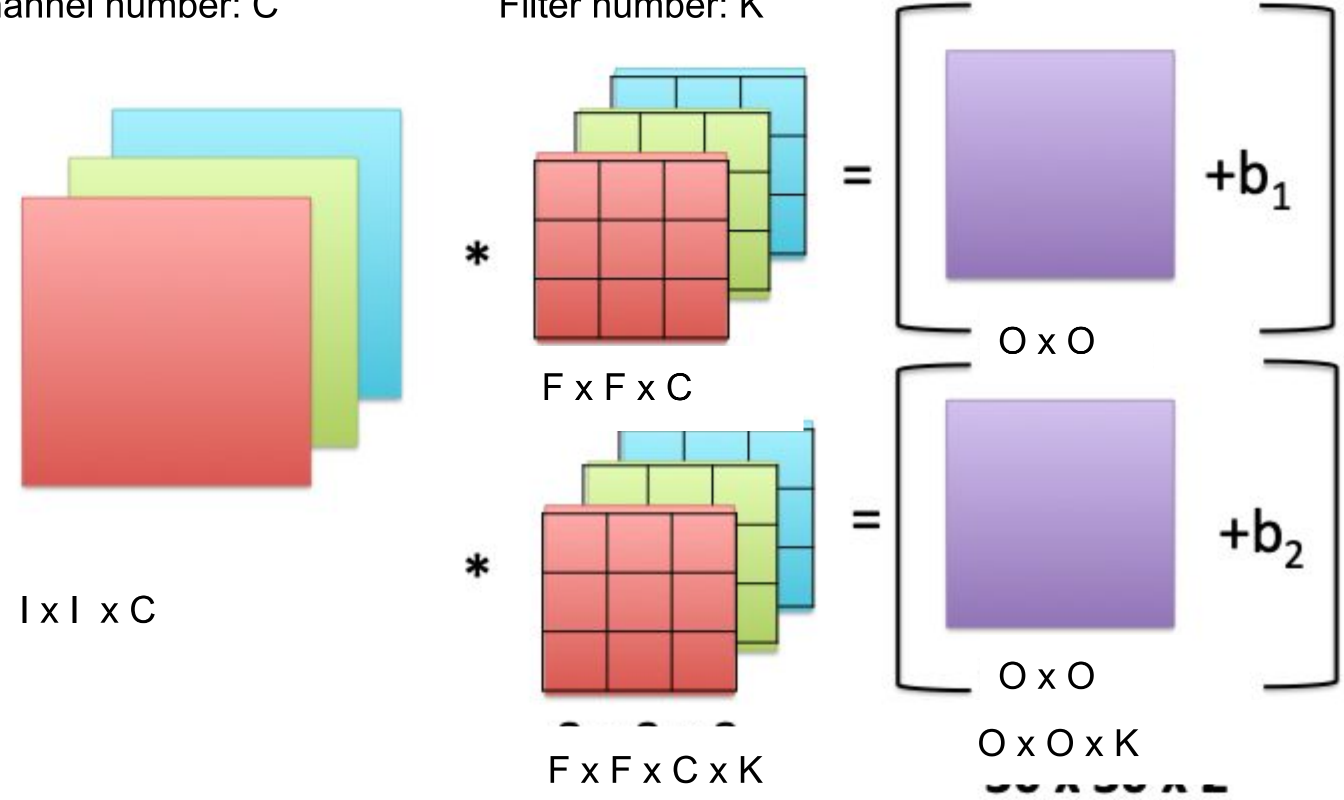
F x F x C

Convolution Layers with Multiple Filters

One bias parameter per filter

Channel number: C

Filter number: K



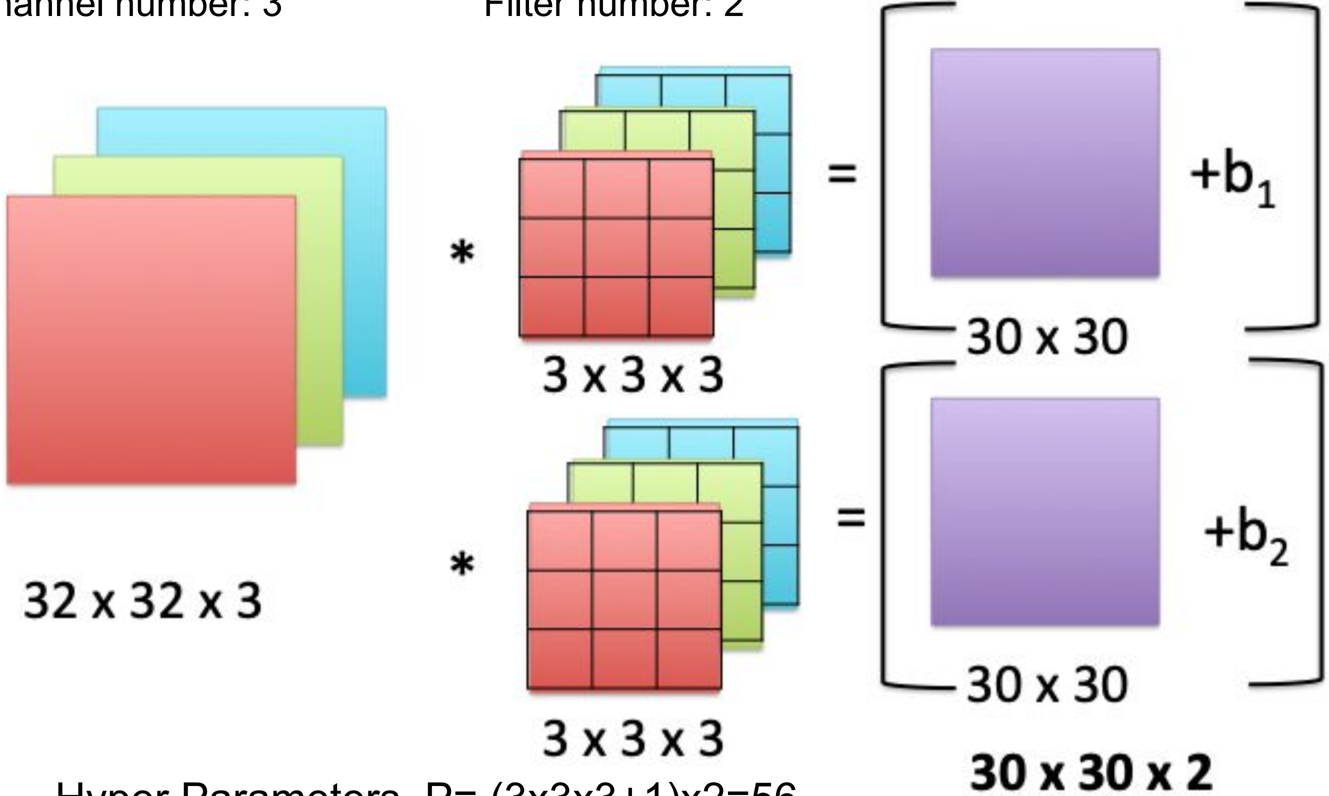
Number of Parameters: (FxFxC+1) x K

Convolution Layer Example

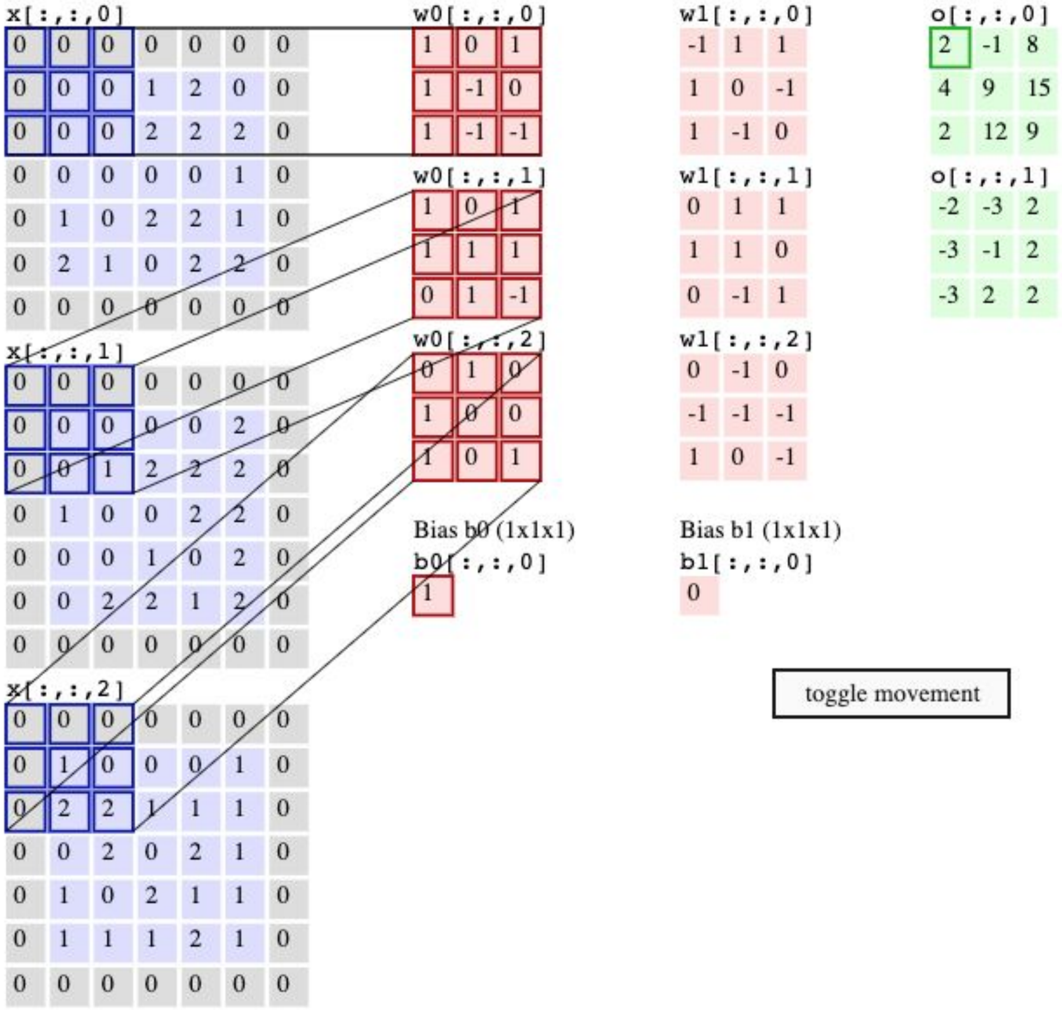
Output size O= 32-3+1 = 30

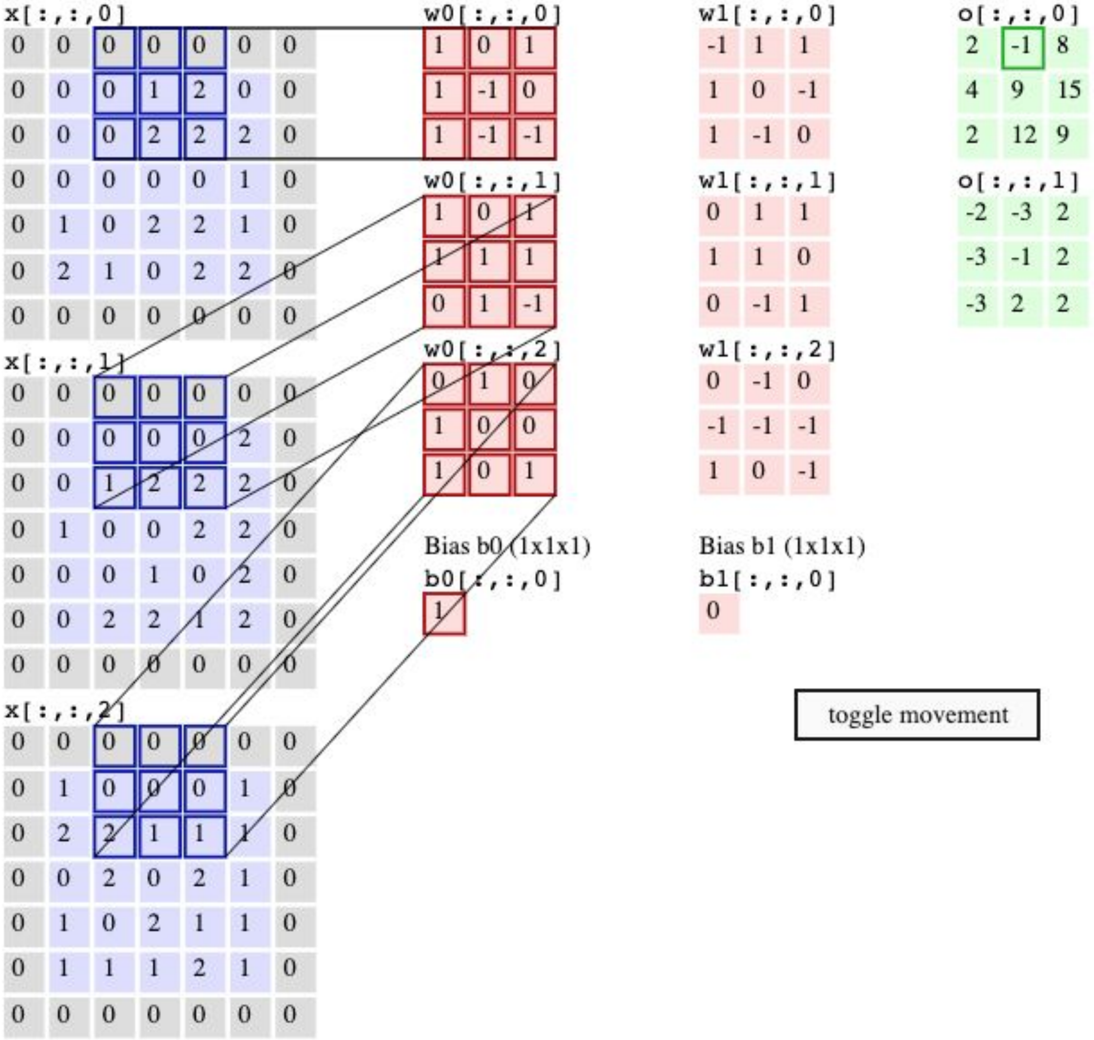
Channel number: 3

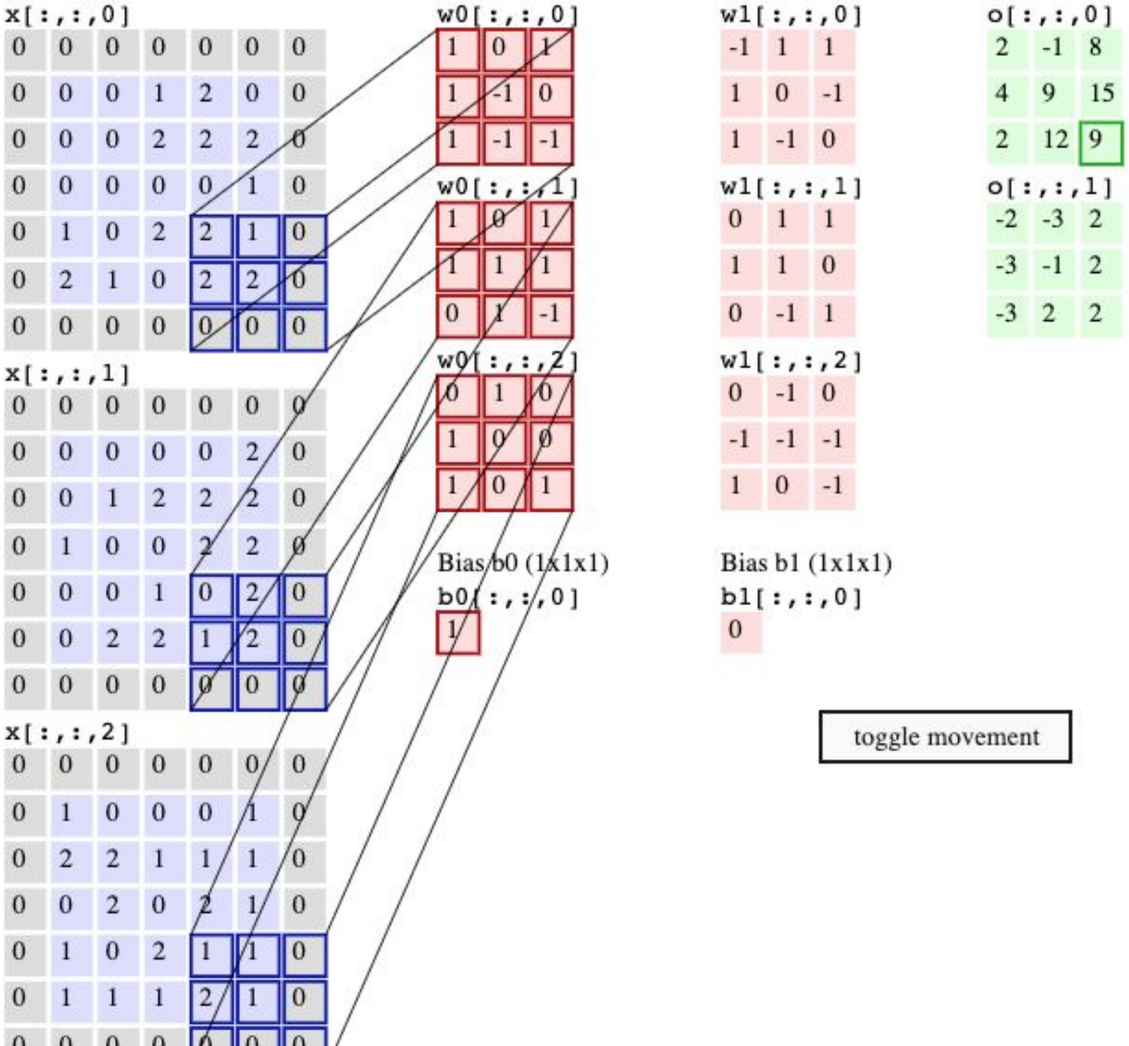
Filter number: 2

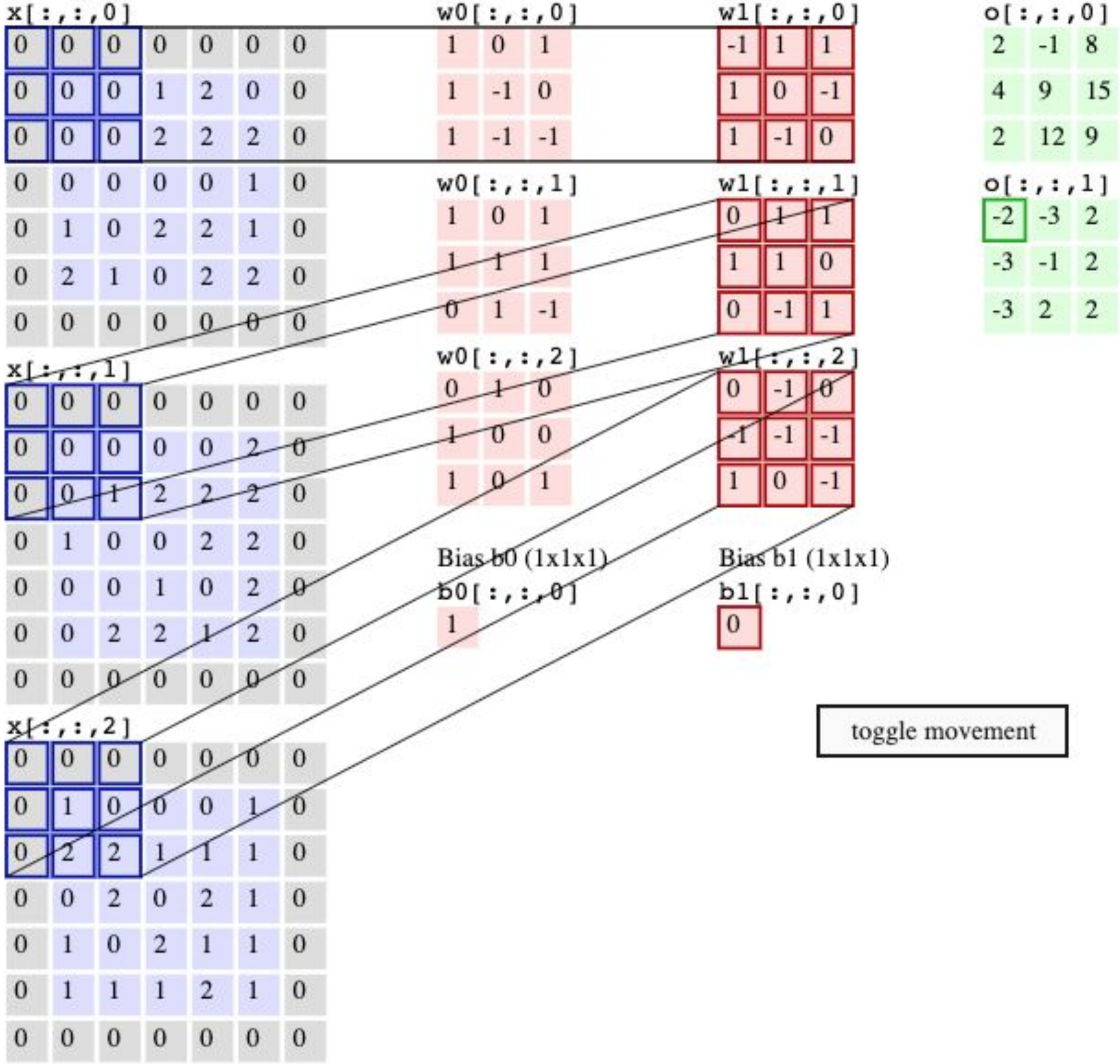


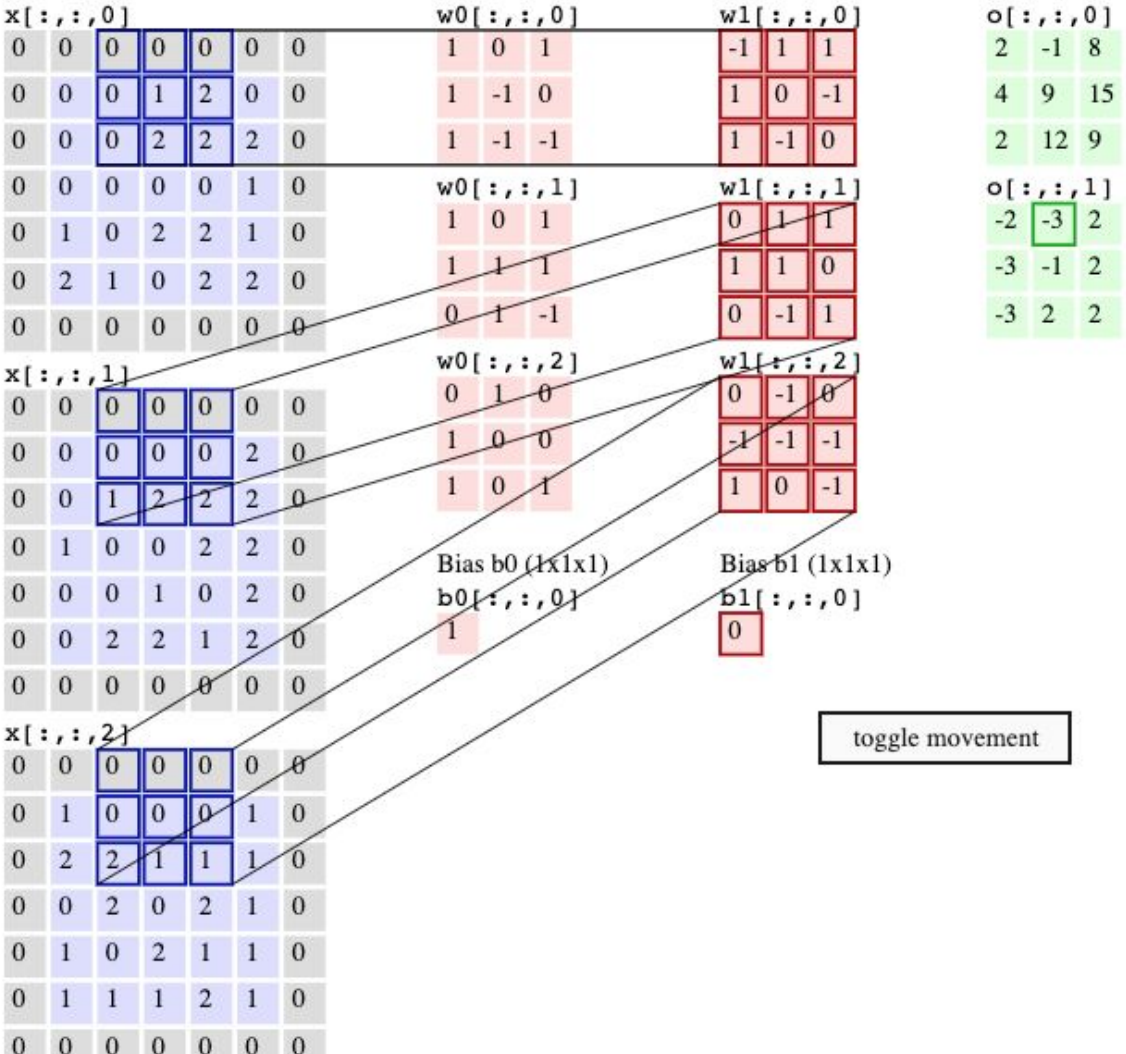
Hyper Parameters P= (3x3x3+1)x2=56


x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 2 0 0
0 0 0 2 2 2 0
0 0 0 0 0 1 0
0 1 0 2 2 1 0
0 2 1 0 2 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 0 0 2 0
0 0 1 2 2 2 0
0 1 0 0 2 2 0
0 0 0 1 0 2 0
0 0 2 2 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 2 2 1 1 1 0
0 0 2 0 2 1 0
0 1 0 2 1 1 0
0 1 1 1 2 1 0
0 0 0 0 0 0 0
w0[:,:,0]
1 0 1
1 -1 0
1 -1 -1
w0[:,:,1]
1 0 1
1 1 1
0 1 -1
w0[:,:,2]
0 1 0
1 0 0
1 0 1
Bias b0 (1x1x1)
b0[:,:,0]
1
w1[:,:,0]
-1 1 1
1 0 -1
1 -1 0
w1[:,:,1]
0 1 1
1 1 0
0 -1 1
w1[:,:,2]
0 -1 0
-1 -1 -1
1 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
o[:,:,0]
2 -1 8
4 9 15
2 12 9
o[:,:,1]
-2 -3 2
-3 -1 2
-3 2 2
toggle movement

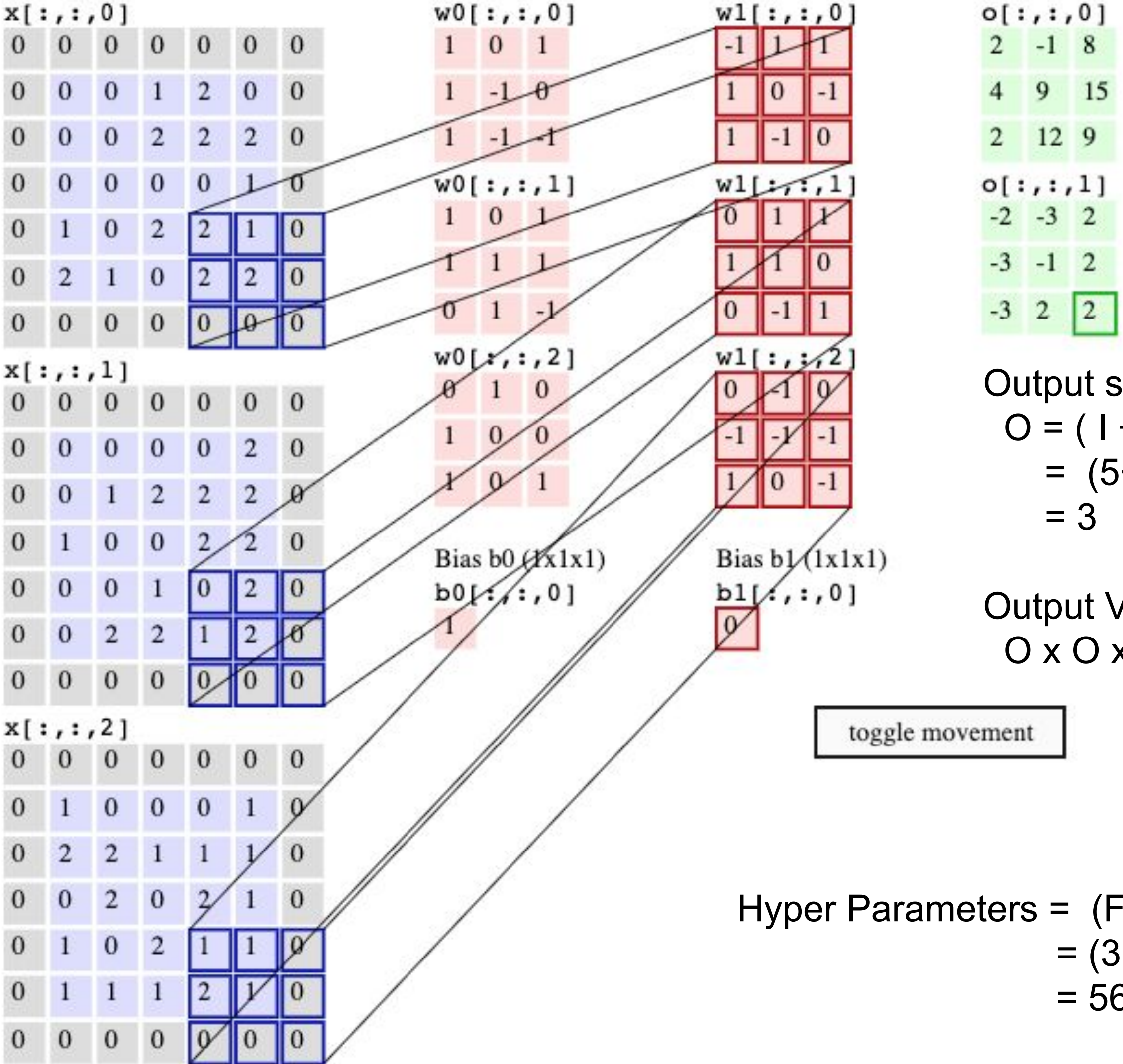

x[:,:,0]
x[:,:,1]
x[:,:,2]
w0[:,:,0]
w0[:,:,1]
w0[:,:,2]
w1[:,:,0]
w1[:,:,1]
w1[:,:,2]
o[:,:,0]
o[:,:,1]
Bias b0 (1x1x1)
b0[:,:,0]
Bias b1 (1x1x1)
b1[:,:,0]
toggle movement


x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 2 0 0
0 0 0 2 2 2 0
0 0 0 0 0 1 0
0 1 0 2 2 1 0
0 2 1 0 2 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 0 0 2 0
0 0 1 2 2 2 0
0 1 0 0 2 2 0
0 0 0 1 0 2 0
0 0 2 2 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 2 2 1 1 1 0
0 0 2 0 2 1 0
0 1 0 2 1 1 0
0 1 1 1 2 1 0
0 0 0 0 0 0 0
w0[:,:,0]
1 0 1
1 -1 0
1 -1 -1
w0[:,:,1]
1 0 1
1 1 1
0 1 -1
w0[:,:,2]
0 1 0
1 0 0
1 0 1
Bias b0 (1x1x1)
b0[:,:,0]
1
w1[:,:,0]
-1 1 1
1 0 -1
1 -1 0
w1[:,:,1]
0 1 1
1 1 0
0 -1 1
w1[:,:,2]
0 -1 0
-1 -1 -1
1 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
o[:,:,0]
2 -1 8
4 9 15
2 12 9
o[:,:,1]
-2 -3 2
-3 -1 2
-3 2 2
toggle movement


x[:,:,0]
x[:,:,1]
x[:,:,2]
w0[:,:,0]
w0[:,:,1]
w0[:,:,2]
w1[:,:,0]
w1[:,:,1]
w1[:,:,2]
Bias b0 (1x1x1)
b0[:,:,0]
Bias b1 (1x1x1)
b1[:,:,0]
o[:,:,0]
o[:,:,1]
toggle movement


x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 2 0 0
0 0 0 2 2 2 0
0 0 0 0 0 1 0
0 1 0 2 2 1 0
0 2 1 0 2 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 0 0 2 0
0 0 1 2 2 2 0
0 1 0 0 2 2 0
0 0 0 1 0 2 0
0 0 2 2 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 2 2 1 1 1 0
0 0 2 0 2 1 0
0 1 0 2 1 1 0
0 1 1 1 2 1 0
0 0 0 0 0 0 0
w0[:,:,0]
1 0 1
1 -1 0
1 -1 -1
w0[:,:,1]
1 0 1
1 1 1
0 1 -1
w0[:,:,2]
0 1 0
1 0 0
1 0 1
Bias b0 (1x1x1)
b0[:,:,0]
1
w1[:,:,0]
-1 1 1
1 0 -1
1 -1 0
w1[:,:,1]
0 1 1
1 1 0
0 -1 1
w1[:,:,2]
0 -1 0
-1 -1 -1
1 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
o[:,:,0]
2 -1 8
4 9 15
2 12 9
o[:,:,1]
-2 -3 2
-3 -1 2
-3 2 2
toggle movement

Input Volume
I x I x C = 5 x 5 x 3

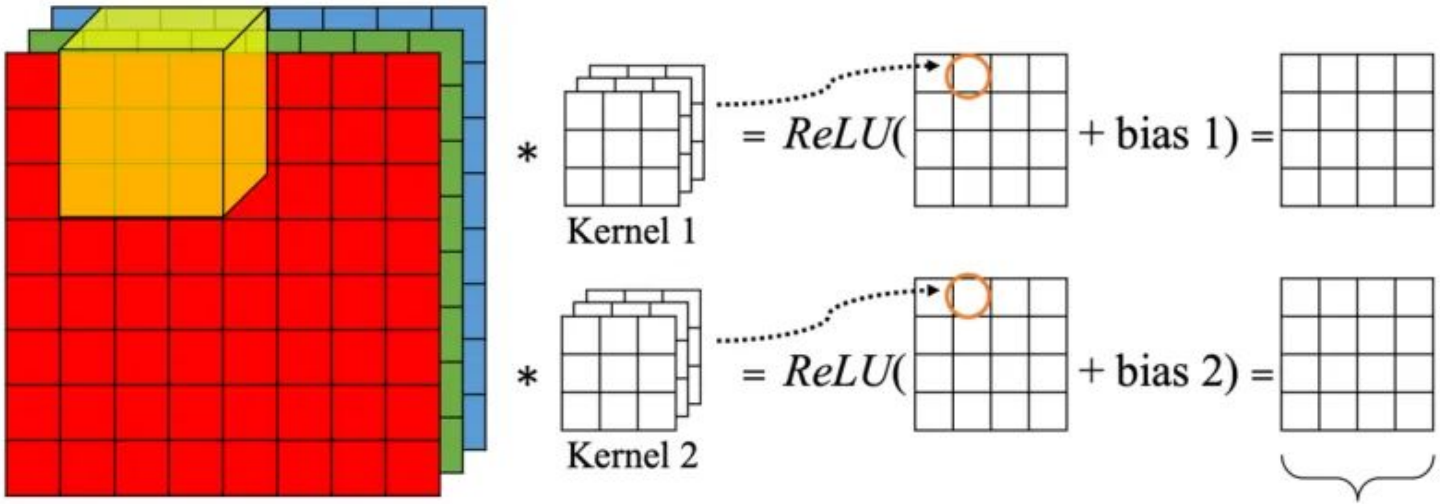


Output size
O = (I + 2P - F)/S+1
= (5+2x1-3)/2+1
= 3

Output Volume
O x O x K = 3 x 3 x 2

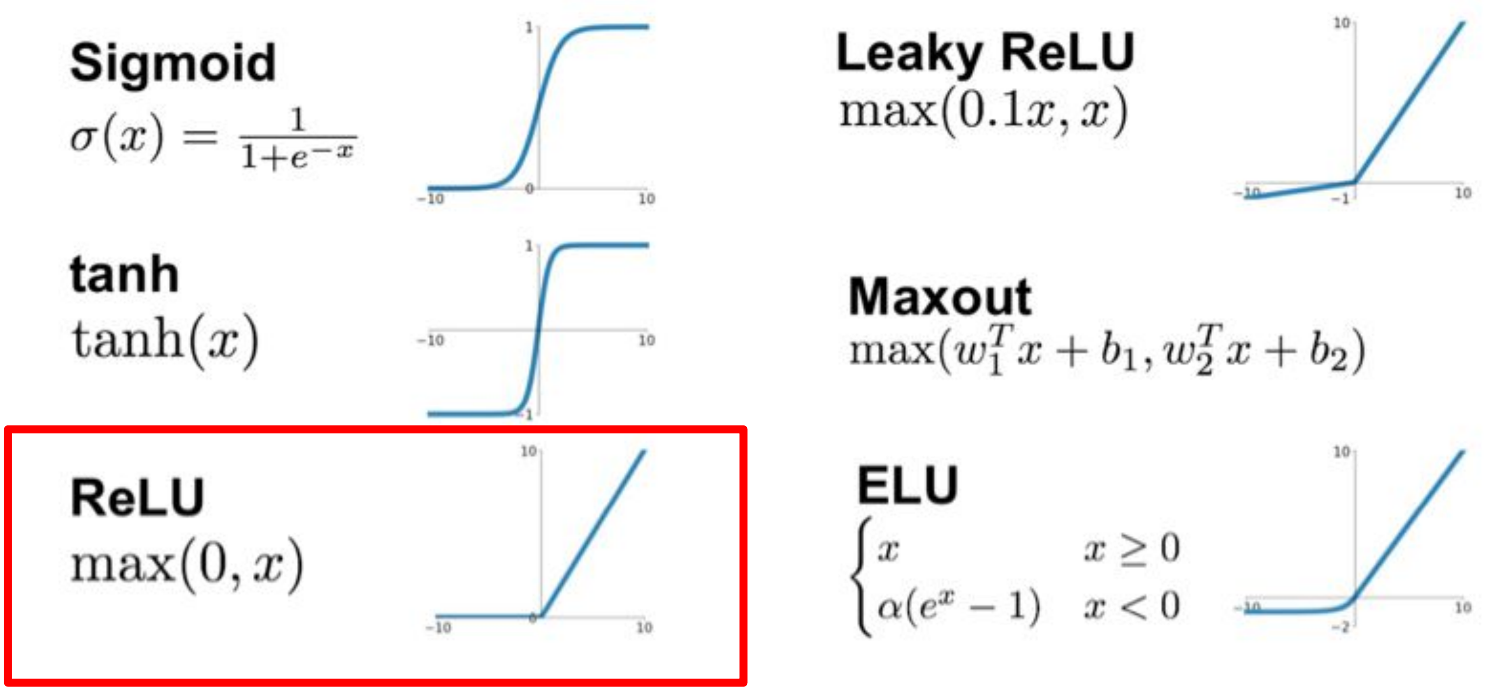
Hyper Parameters = (F x F x C +1) x K
= (3 x 3 x 3 +1) x 2
= 56

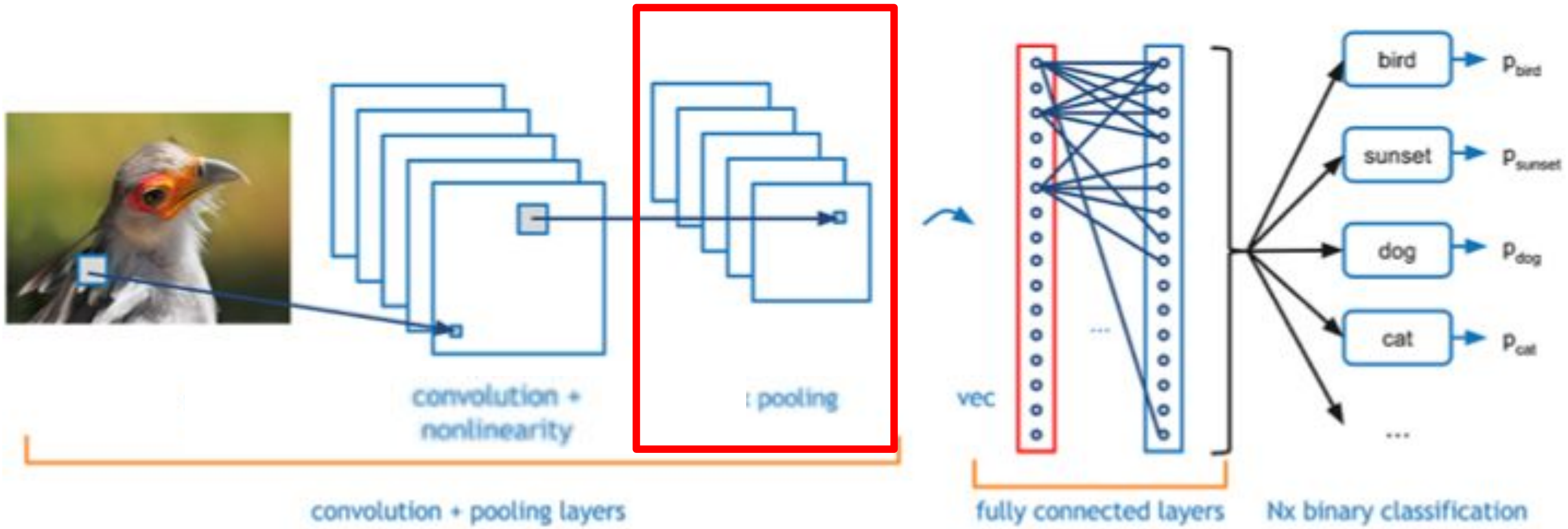
Non-linearity



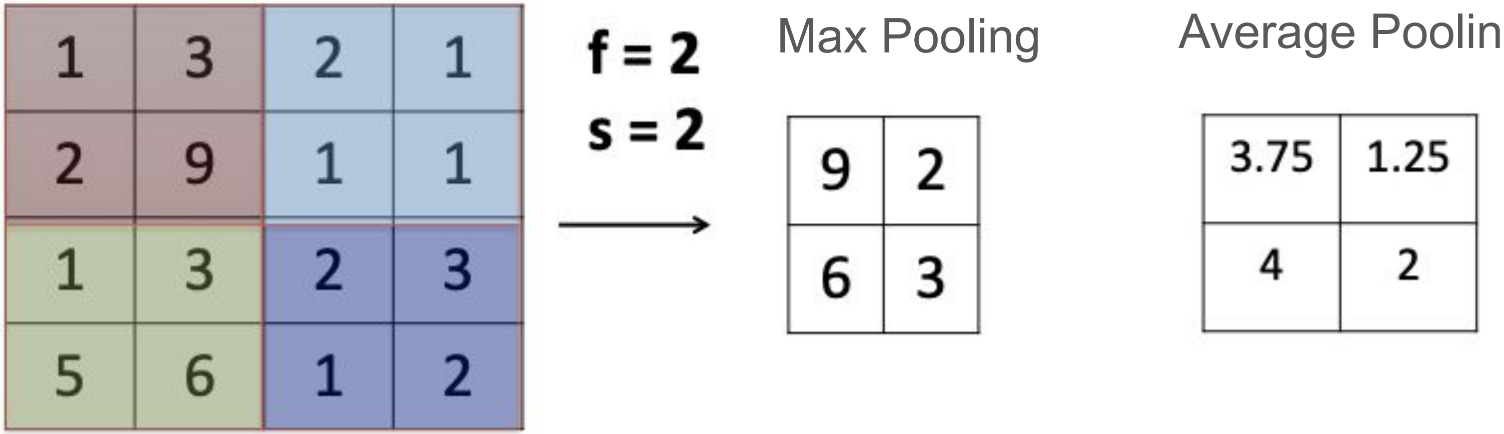
Commonly Used Activation Functions

Rectified Linear Unit (ReLU) introduces non-linearities to the network.




convolution +
nonlinearity
: pooling
vec
bird
sunset
dog
cat
...
convolution + pooling layers
fully connected layers
Nx binary classification

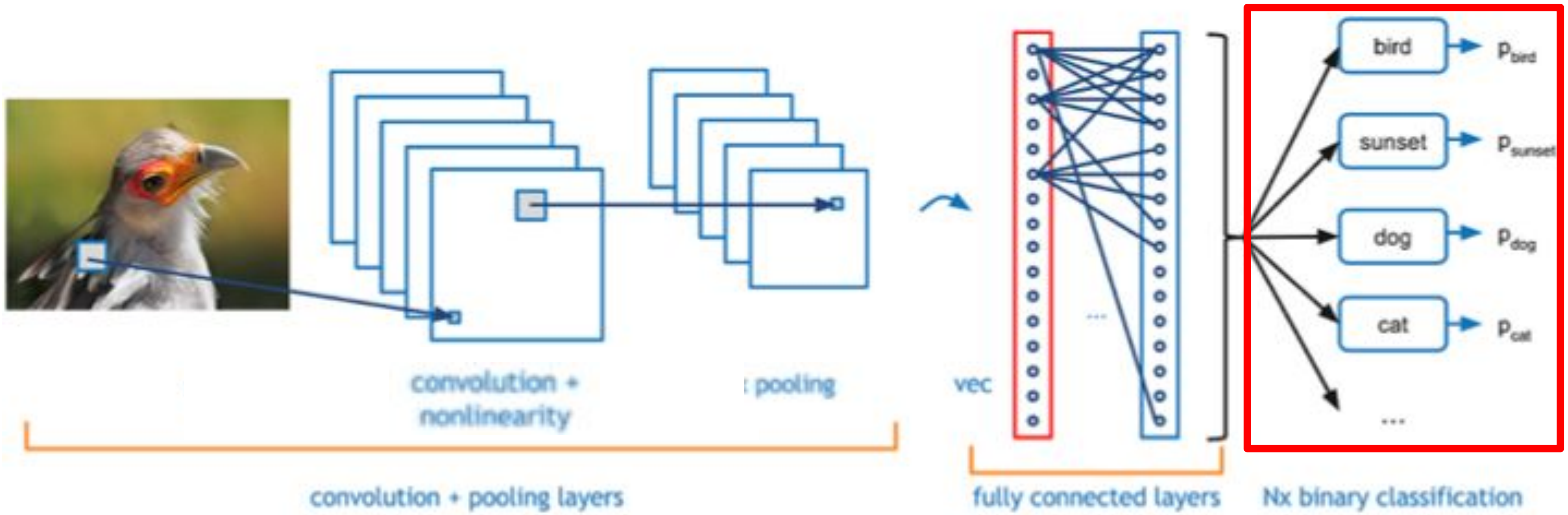
Pooling



I x I   FxF   O=floor [(I+2P-F)/S+1]

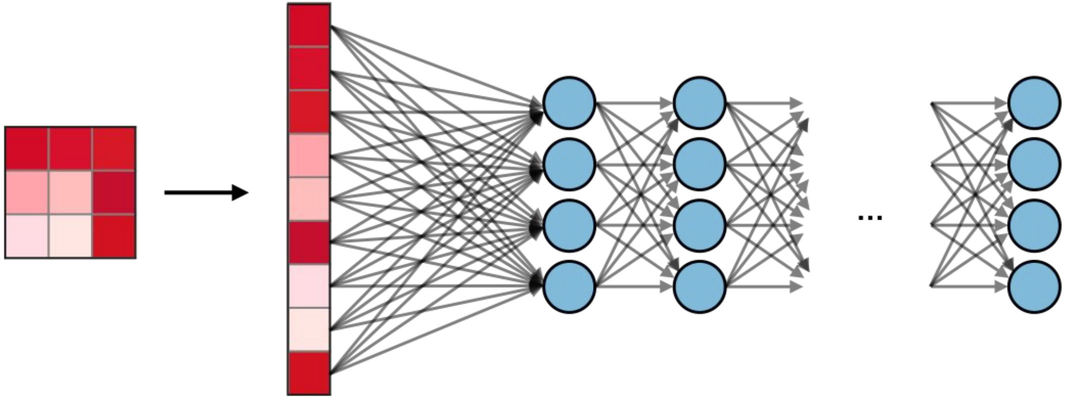
In most cases, Max Pooling is used and S = F

Number of parameters for Pooling is **0**



bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
convolution +
nonlinearity
: pooling
vec
convolution + pooling layers
fully connected layers
Nx binary classification

Softmax

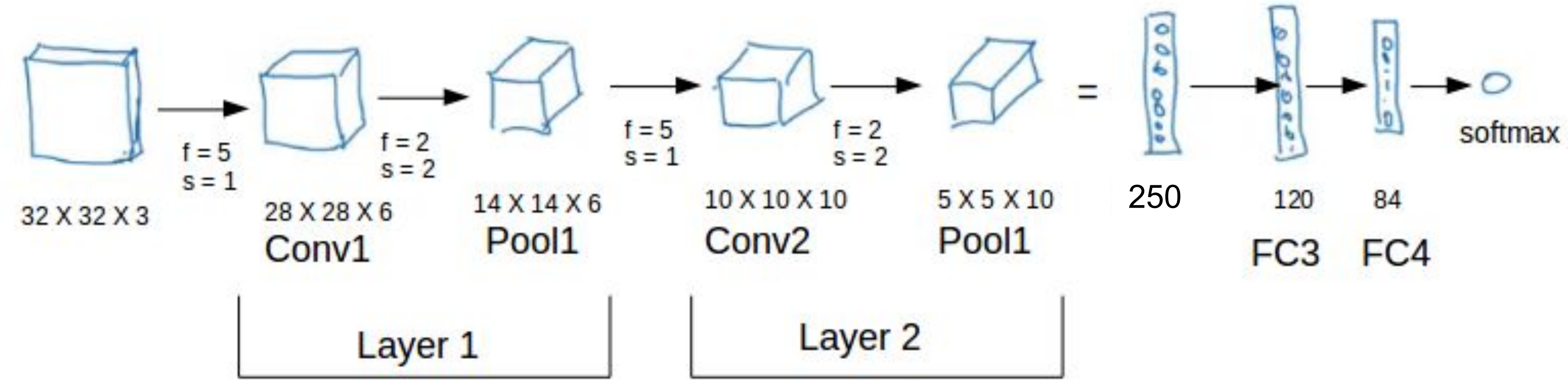


For final classification layer

$$\sigma : \mathbb{R}^K \to (0,1)^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

A full CNN example



Credit: Analytic Vidhya

Convolution Benefit

- Parameter Sharing
  - Filter can be used in different part of inputs
- Sparsity of Connections
  - In each layer each output value depends only on small number of inputs (local)
  - Translation invariance

TensorFlow Tutorial - CNN

https://www.tensorflow.org/tutorials/images/cnn

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

conv2d:

O= (I-F)+1=(32-3+1)=30

P= (F*F*C+1)*K=(3x3x3+1)x32=896

max_pooling2d:

O= (I-F)/S+1=(30-2)/2+1=15

P=0

conv2d_1:

O= (I-F)+1=(15-3+1)=13

P= (F*F*C+1)*K=(3x3x32+1)x64=18496

```python
model.add(layers.Flatten())
model.add(layers.Dense(64,
activation='relu'))
model.add(layers.Dense(10))

model.summary()
```

```
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)         36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)             0
_________________________________________________________________
dense (Dense)                (None, 64)               65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)               650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

<< 9M parameters (FC case)

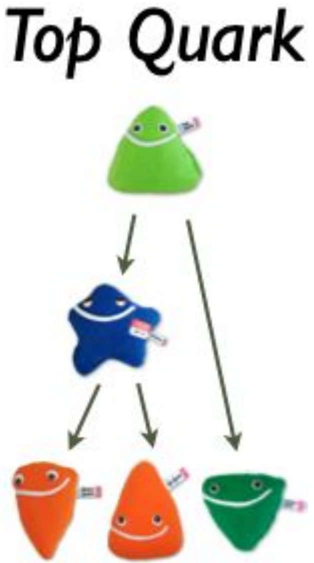
Flatten =4x4x64 = 1024

dense:
P=(I+1)xN=(1024+1)x64=65600

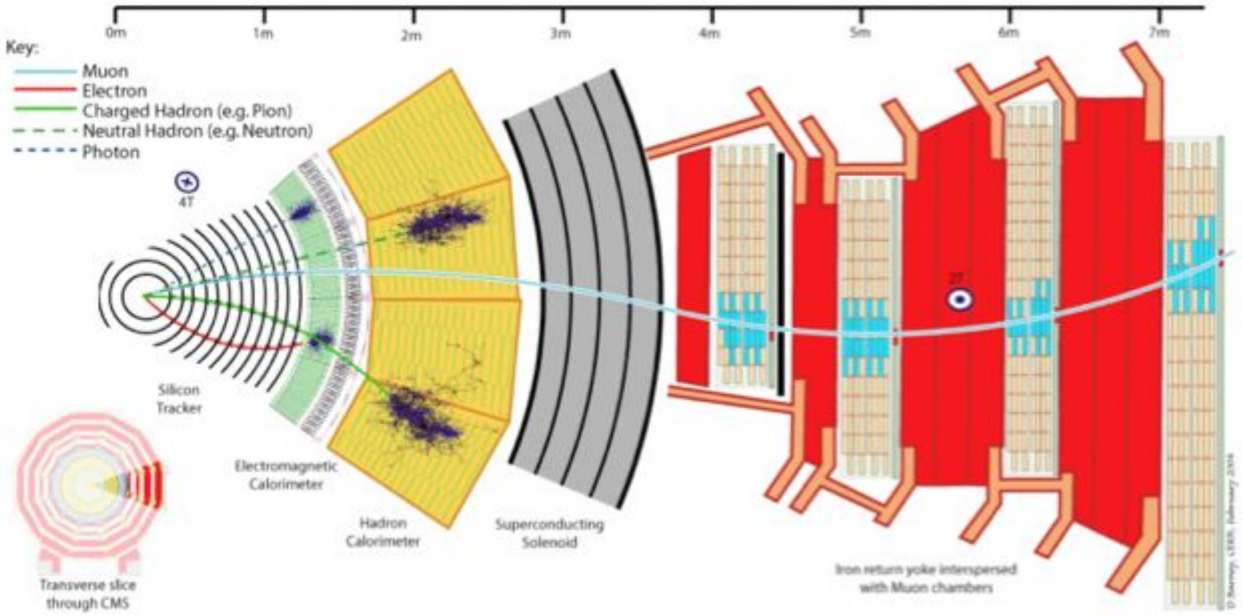
dense_1:
P=(I+1)xN=(64+1)x10=650

Jet Images for in-class Quiz and Lab

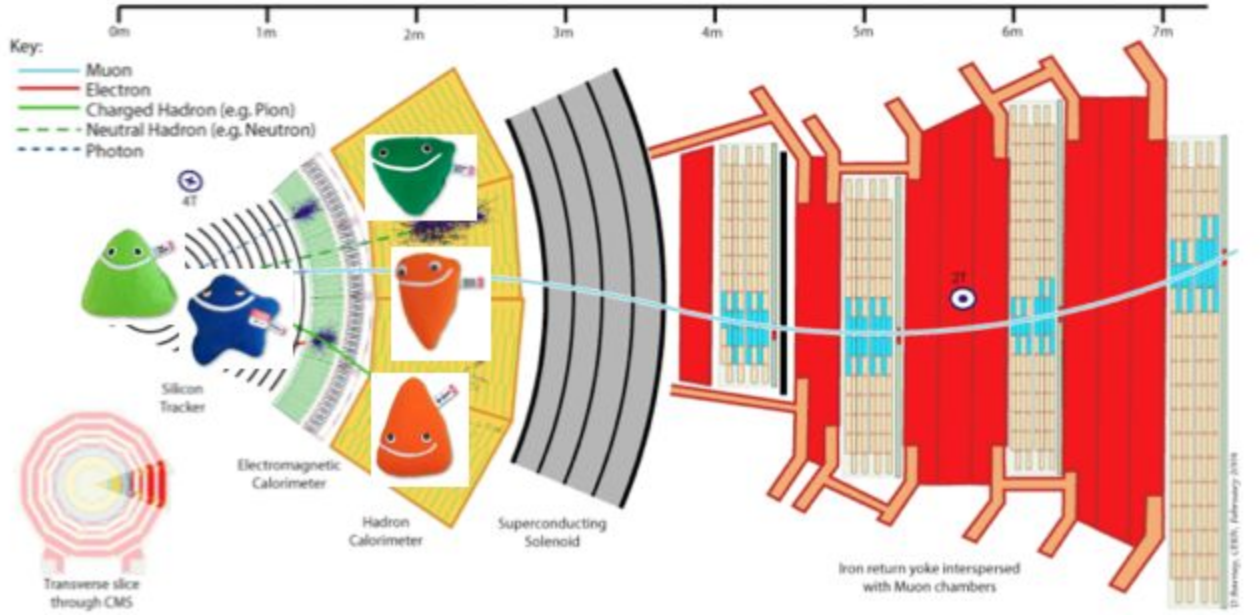
LHC Jet Physics



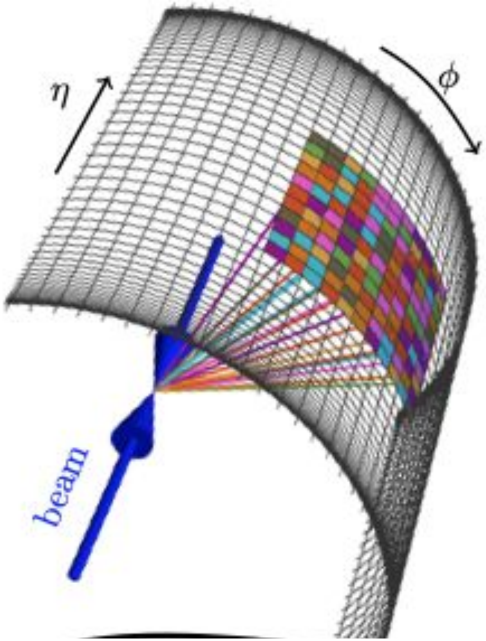
+



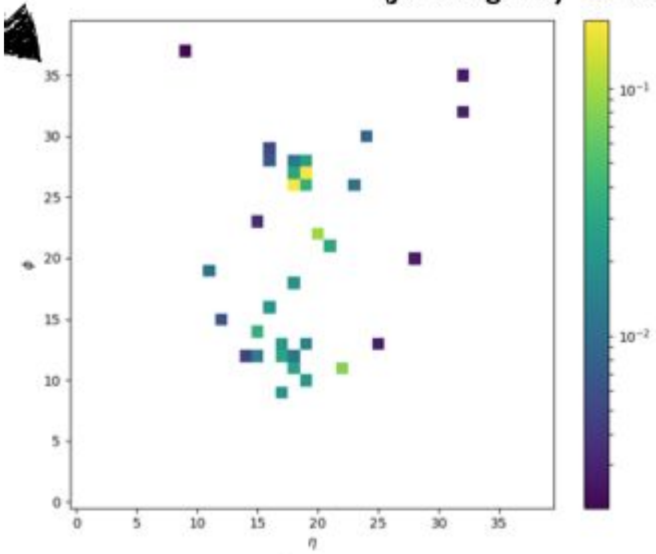
Top Quark Jets



Jet as Image

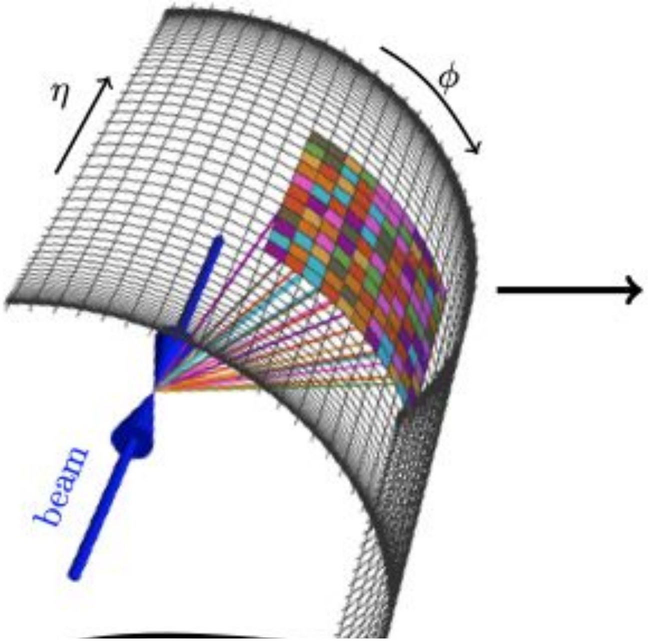


(jet images by C Daza)

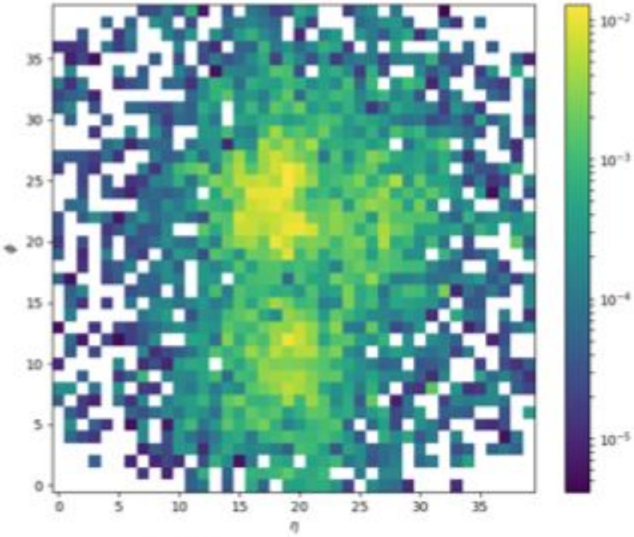


I image

[JHEP 01 (2017) 110](#)

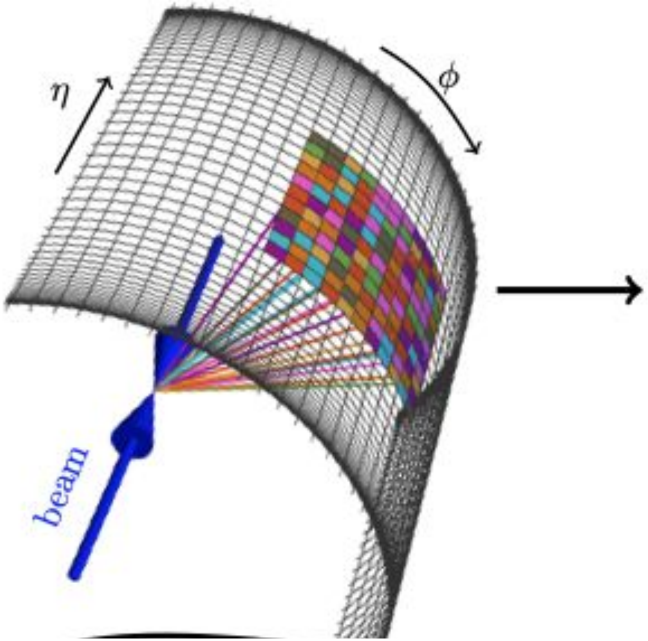


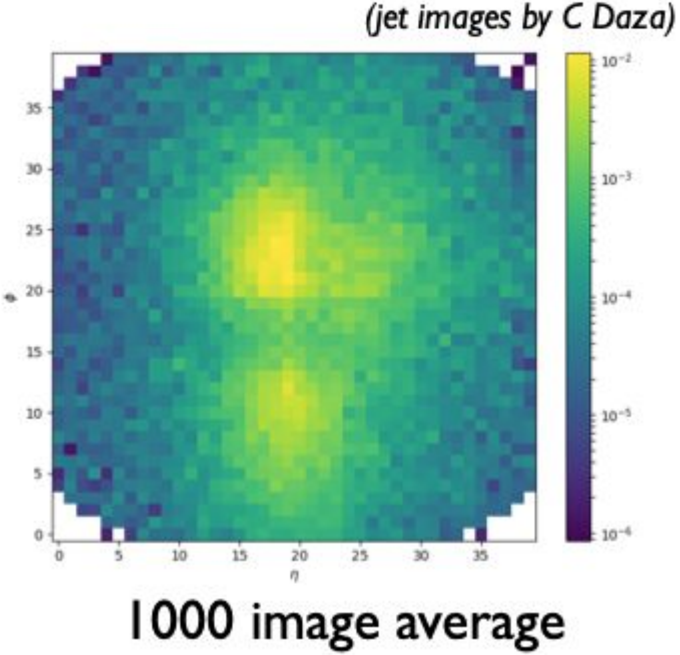
(jet images by C Daza)



100 image average

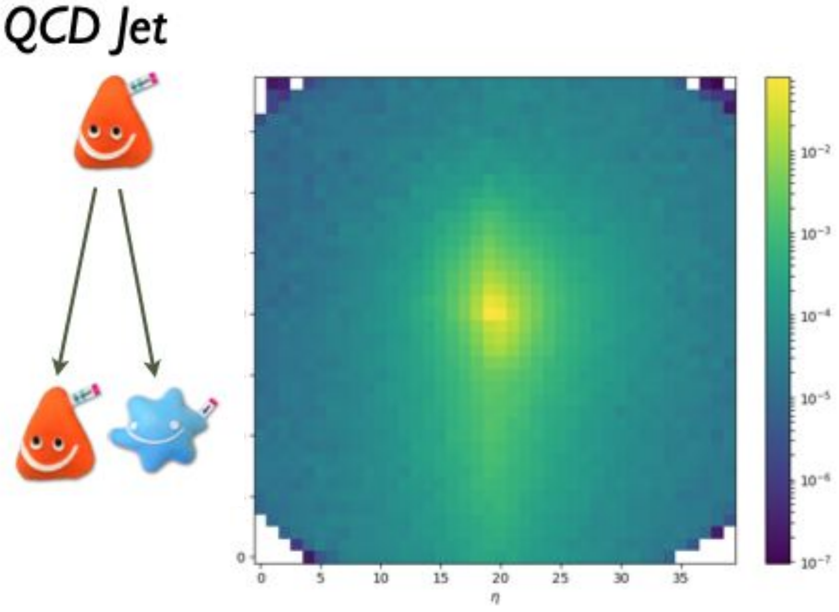
JHEP 01 (2017) 110

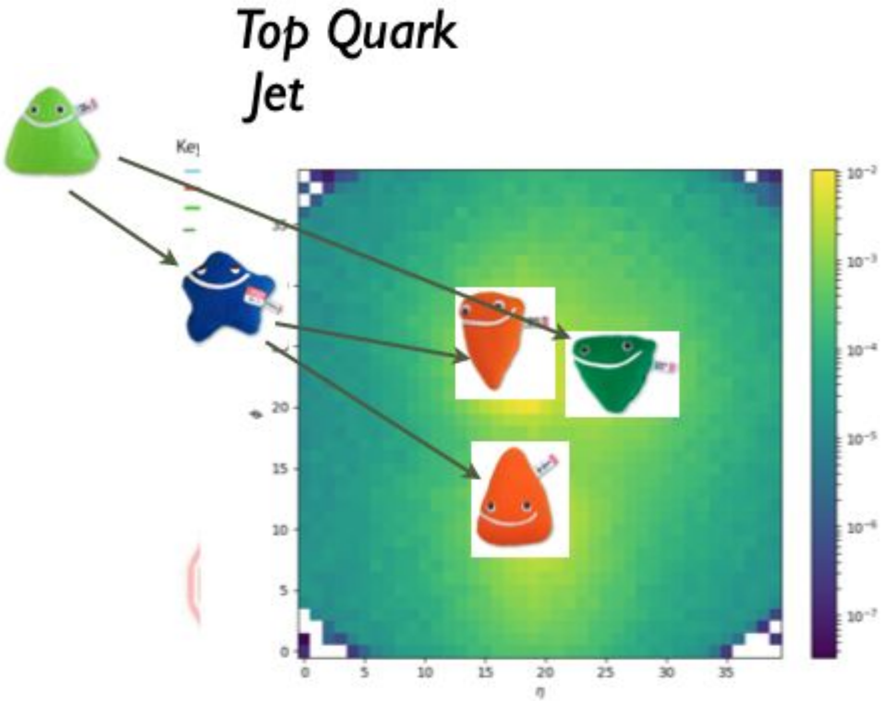




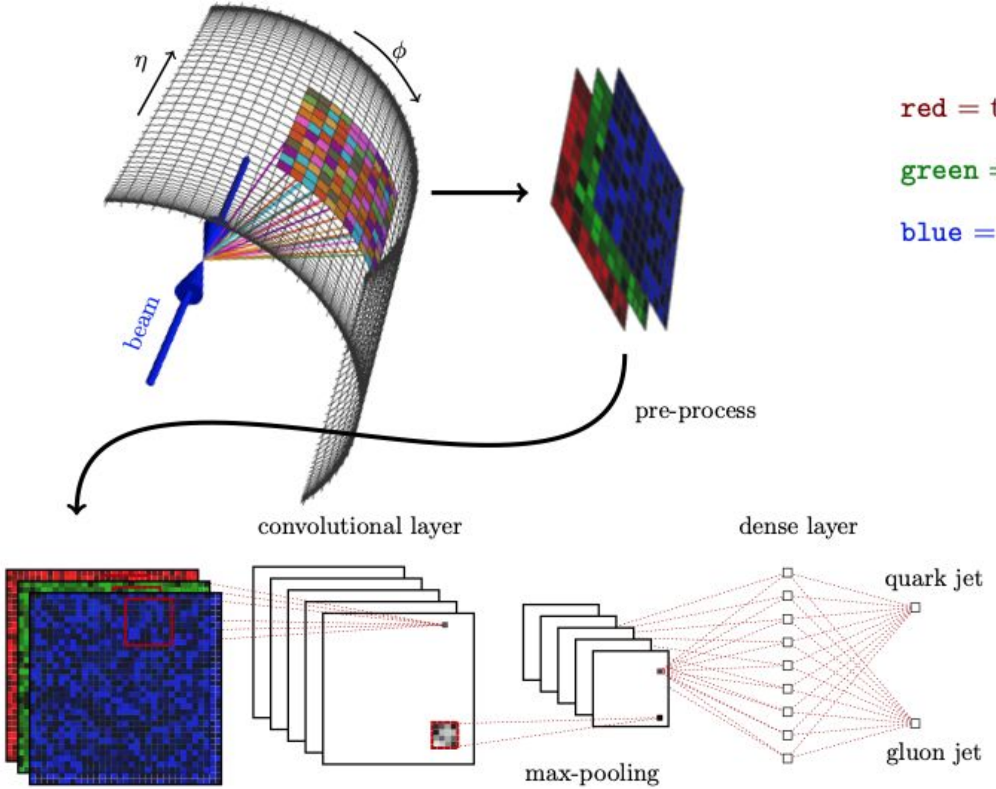
JHEP 01 (2017) 110

QCD Jet vs Top Quark Jet





Jet Image Classification with CNN



red = transverse momenta of charged particles

green = the transverse momenta of neutral particles

blue = charged particle multiplicity

JHEP 01 (2017) 110

References

- Stanford CS 230 https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
- P. Sharma blog https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
- Gregor Kasieczka, “ML4HEP - miniCourse” [URL]
- P. Komiske, et.al. “Deep Learning in Color: towards automated quark/gluon jet discrimination” JHEP 01 (2017) 110