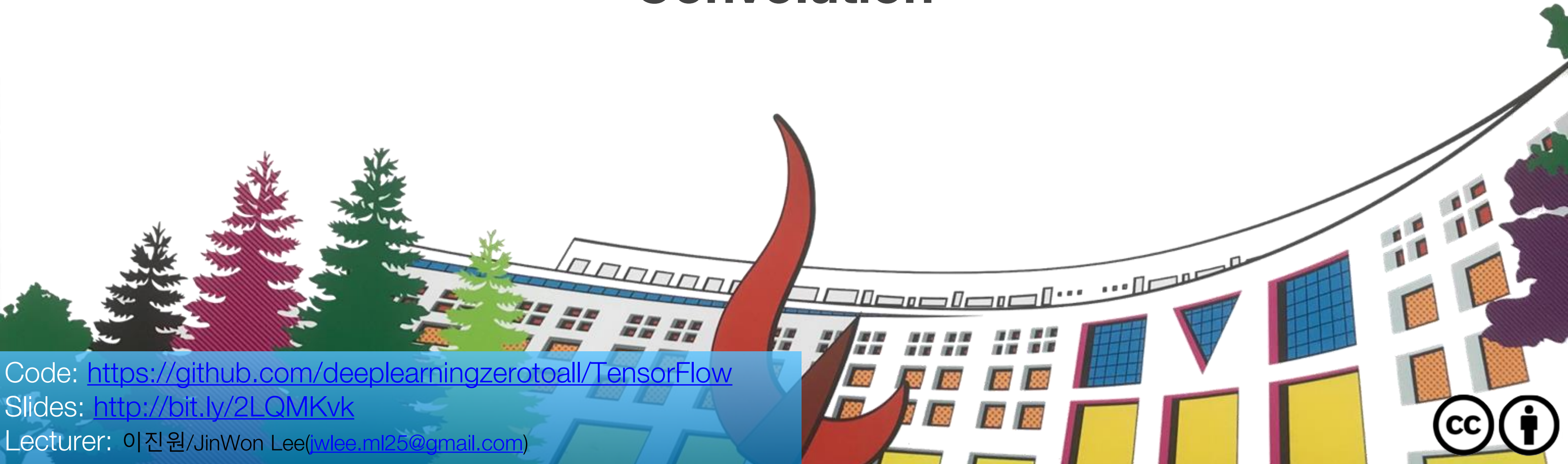
ML/DL for Everyone Season2

with TensorFlow

Lab 11-0 CNN Basics
Convolution



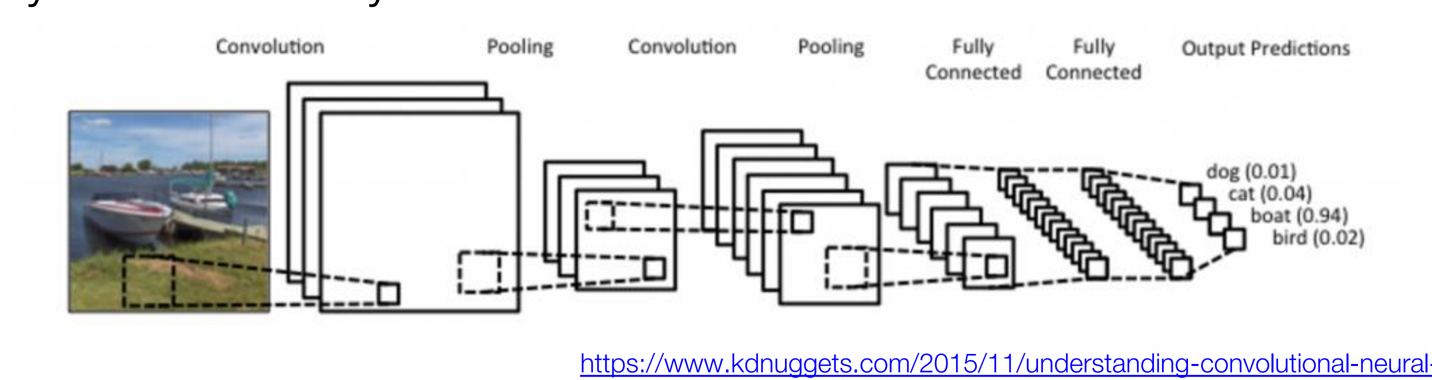
Code: https://github.com/deeplearningzerotoall/TensorFlow

Slides: http://bit.ly/2LQMKvk

Lecturer: 이진원/JinWon Lee(jwlee.ml25@gmail.com)

Convolutional Neural Network

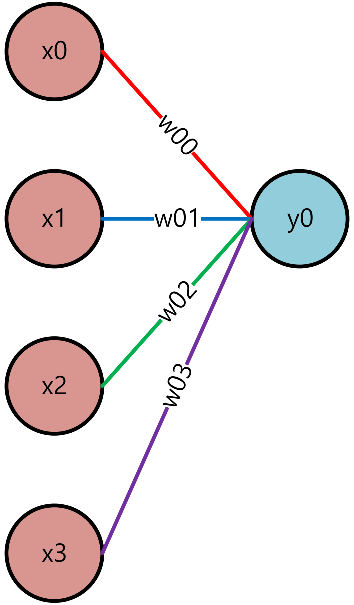
- Most widely used for image classification.
- Generally, it consists of convolution layer, pooling layer and fully-connected layer.
- Weight(parameter, filter, kernel) sharing
- Convolution, Pooling layer – feature extraction
- Fully-connected layer – classification



https://www.kdnuggets.com/2015/11/understanding-convolutional-neural-networks-nlp.html

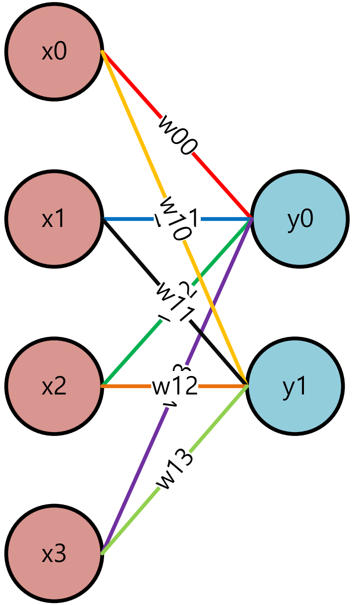
Dense Layer vs 1-D Convolution Layer

- Dense Layer(Fully Connected Layer)
  - $y0 = x0 \cdot w00 + x1 \cdot w01 + x2 \cdot w02 + x3 \cdot w03$



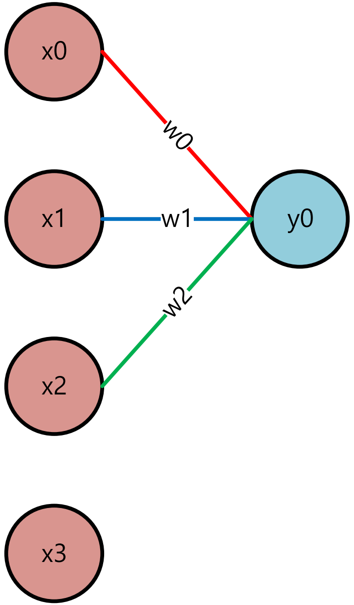
Dense Layer vs 1-D Convolution Layer

- Dense Layer(Fully Connected Layer)
  - $y0 = x0 \cdot w00 + x1 \cdot w01 + x2 \cdot w02 + x3 \cdot w03$
  - $y1 = x0 \cdot w10 + x1 \cdot w11 + x2 \cdot w12 + x3 \cdot w13$



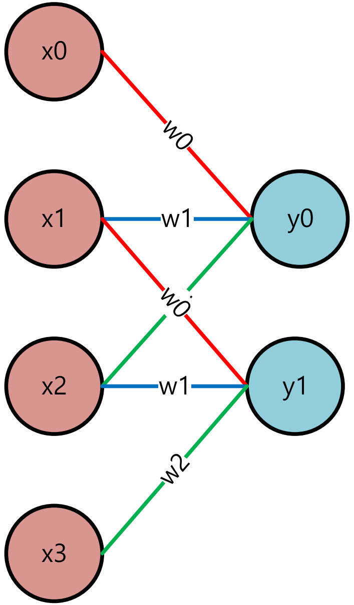
Dense Layer vs 1-D Convolution Layer

- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$



Dense Layer vs 1-D Convolution Layer

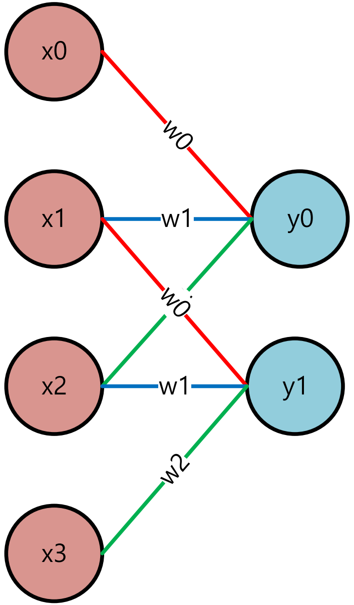
- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$
  - $y0 = x1 \cdot w0 + x2 \cdot w1 + x3 \cdot w2$



Dense Layer vs 1-D Convolution Layer

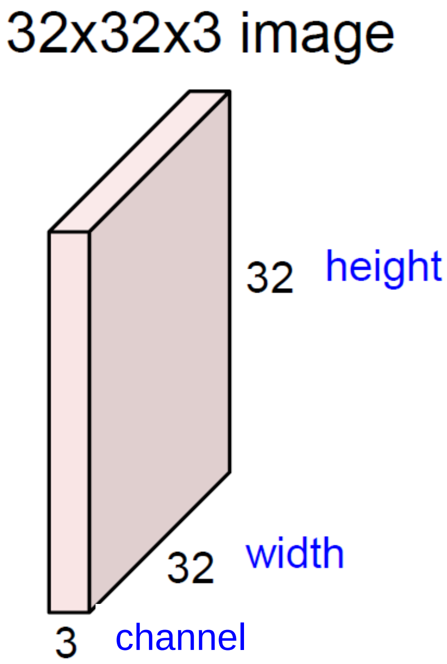
- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$
  - $y0 = x1 \cdot w0 + x2 \cdot w1 + x3 \cdot w2$

Weight sharing
&
Locally connected



2D Convolution Layer

Convolution Layer

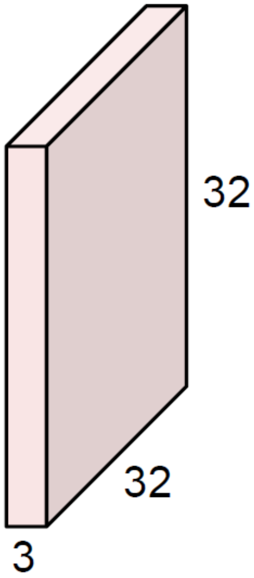


Slide Credit : Stanford CS231n

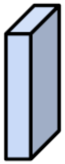
2D Convolution Layer

Convolution Layer

32x32x3 image



5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

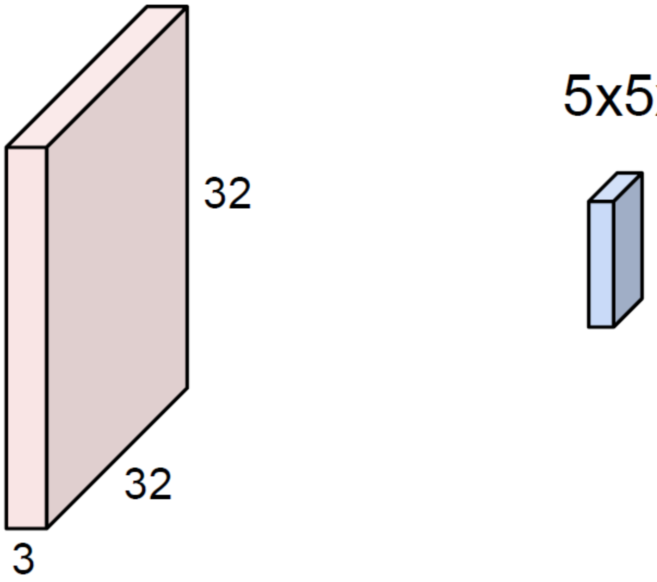
2D Convolution Layer

Convolution Layer

Filters always extend the full channel of the input volume

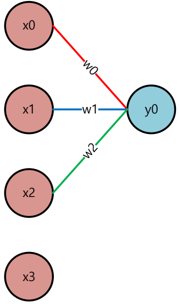
32x32x3 image

5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

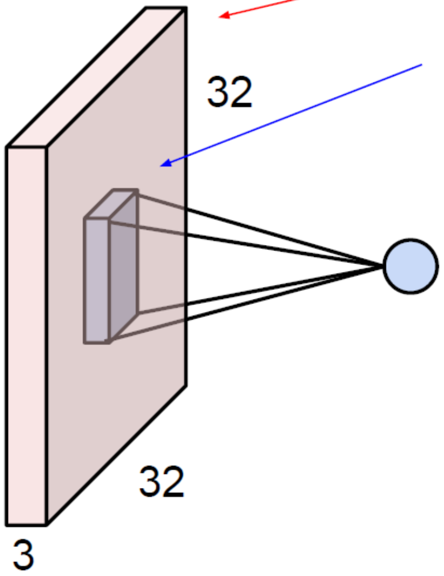
2D Convolution Layer



Convolution Layer

32x32x3 image

5x5x3 filter w

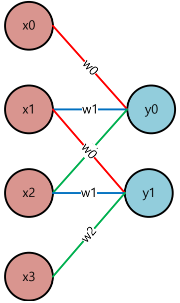


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

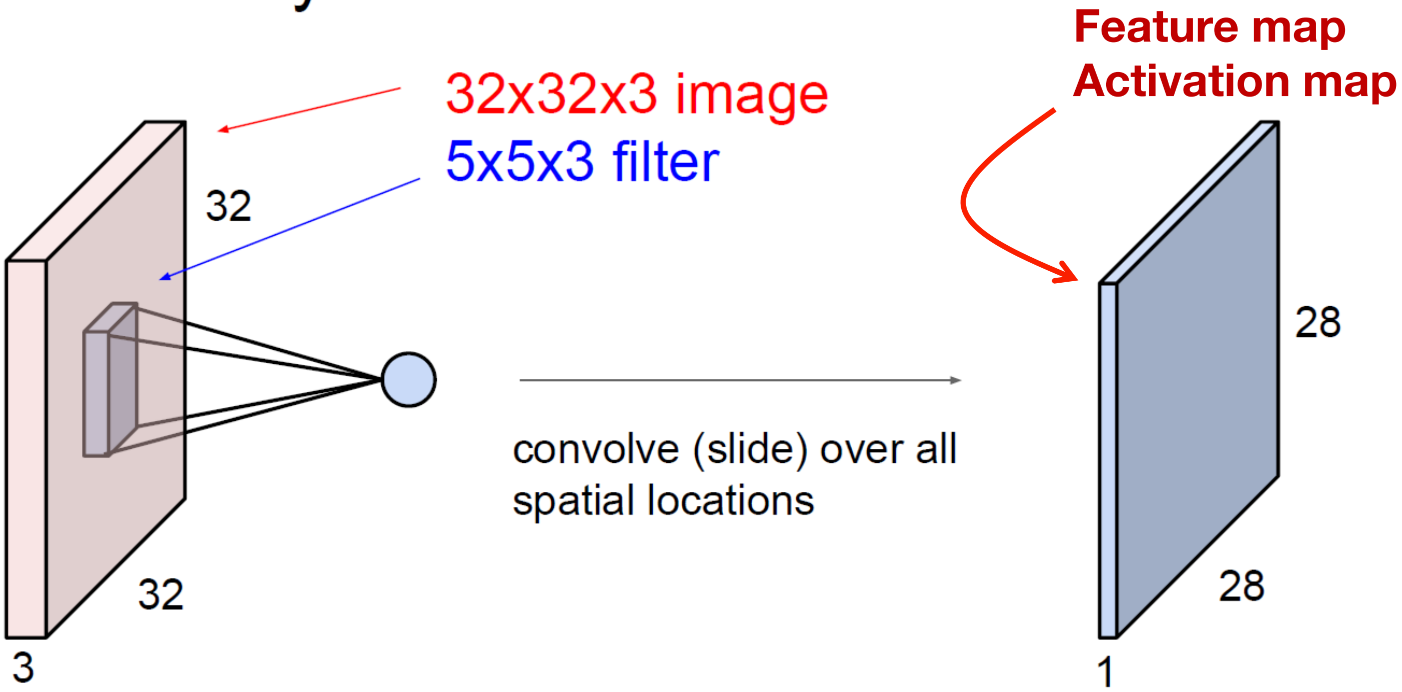
$$w^T x + b$$

Slide Credit : Stanford CS231n

2D Convolution Layer



Convolution Layer

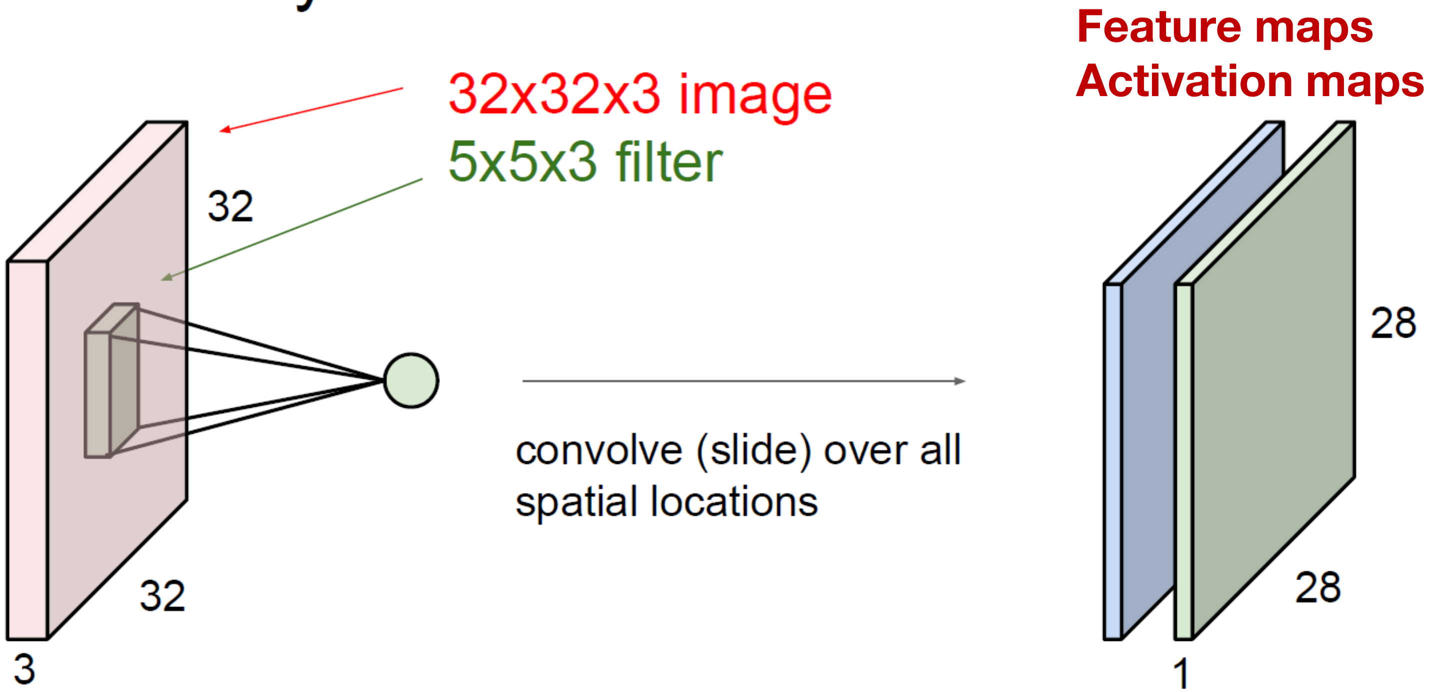


Slide Credit : Stanford CS231n

2D Convolution Layer

Convolution Layer

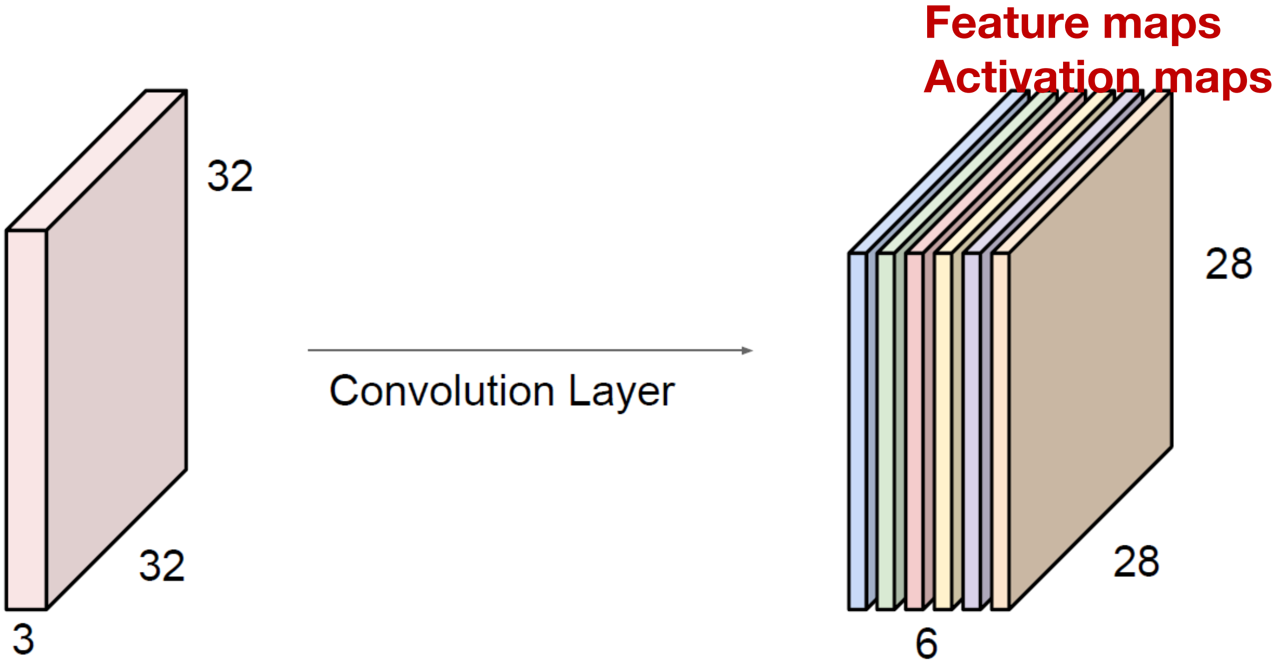
consider a second, green filter



Slide Credit : Stanford CS231n

2D Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

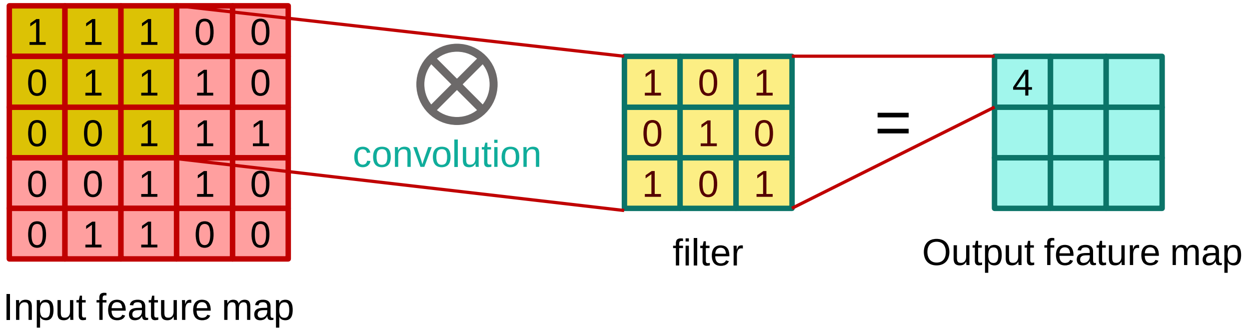


We stack these up to get a "new image" of size 28x28x6!

Slide Credit : Stanford CS231n

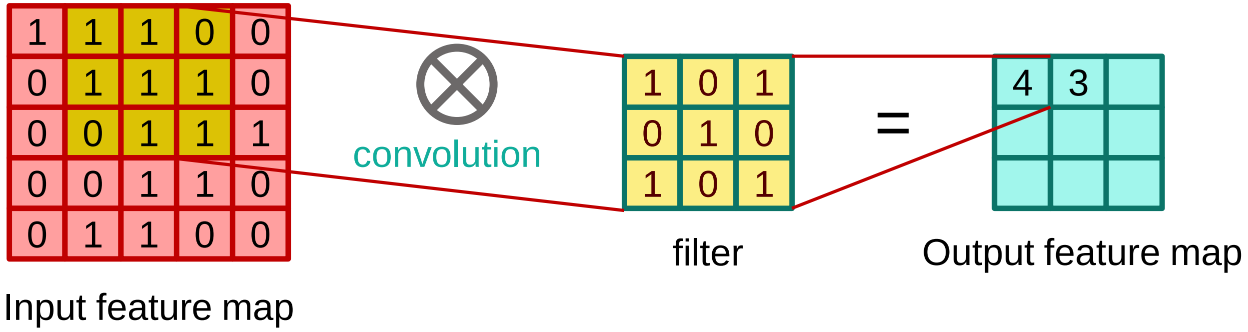
2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 = 4$



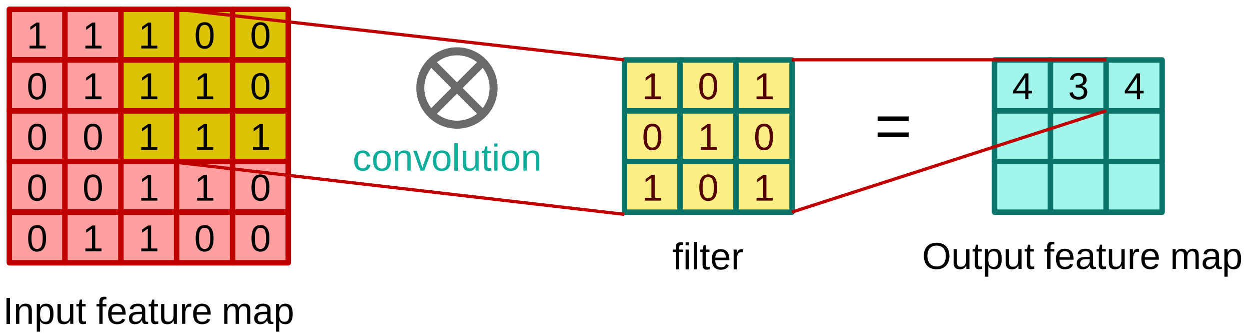
2D Convolution Layer – Computation

- 1x1 + 1x0 + 0x1 + 1x0 + 1x1 + 1x0 + 0x1 + 1x0 + 1x1 = 3



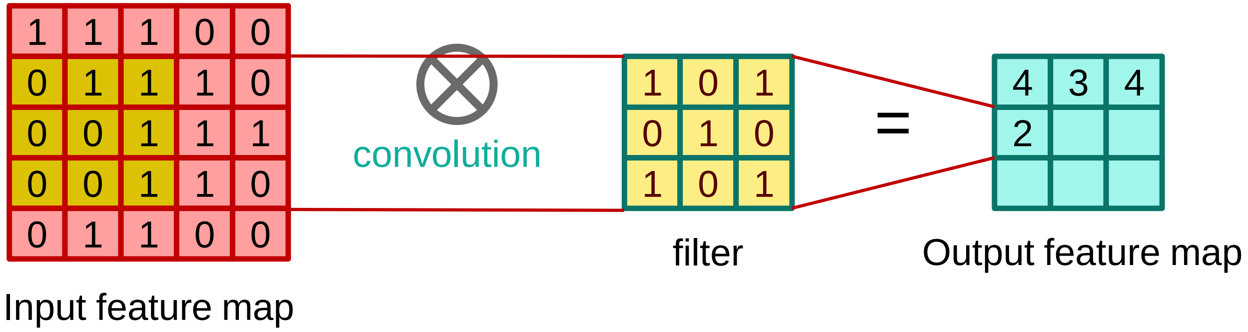
2D Convolution Layer – Computation

- $1 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 = 4$



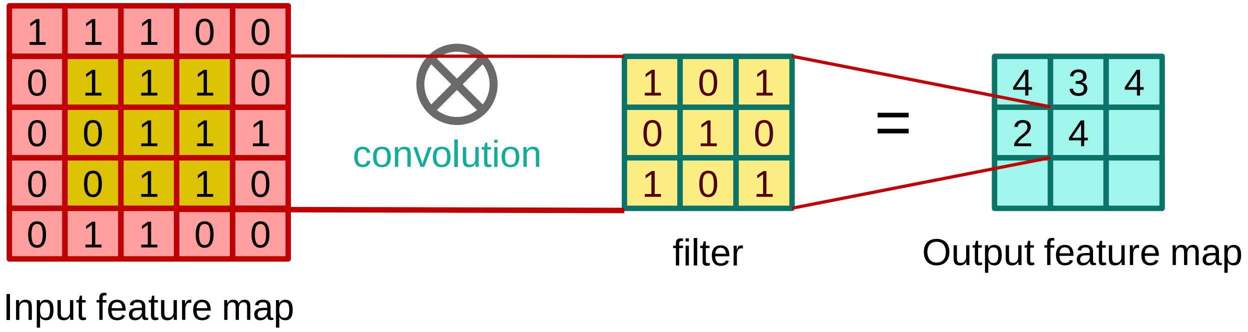
2D Convolution Layer – Computation

- 0x1 + 1x0 + 1x1 + 0x0 + 0x1 + 1x0 + 0x1 + 0x0 + 1x1 = 2



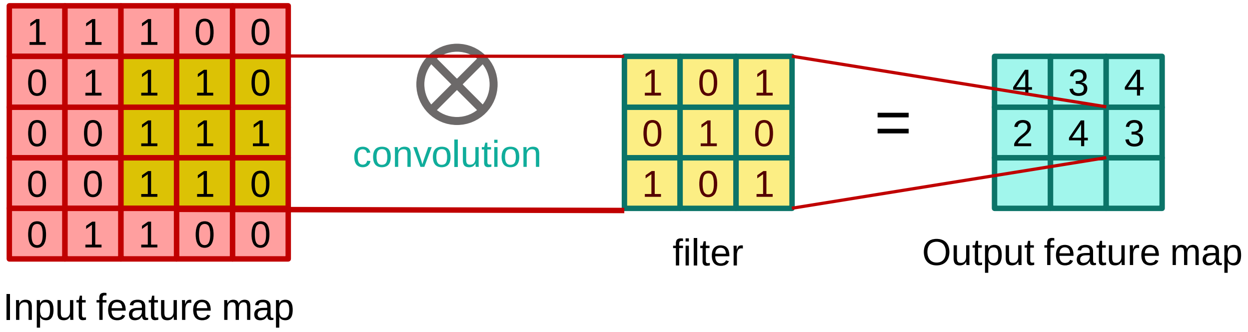
2D Convolution Layer – Computation

- 1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 1x0 + 1x1 = 4



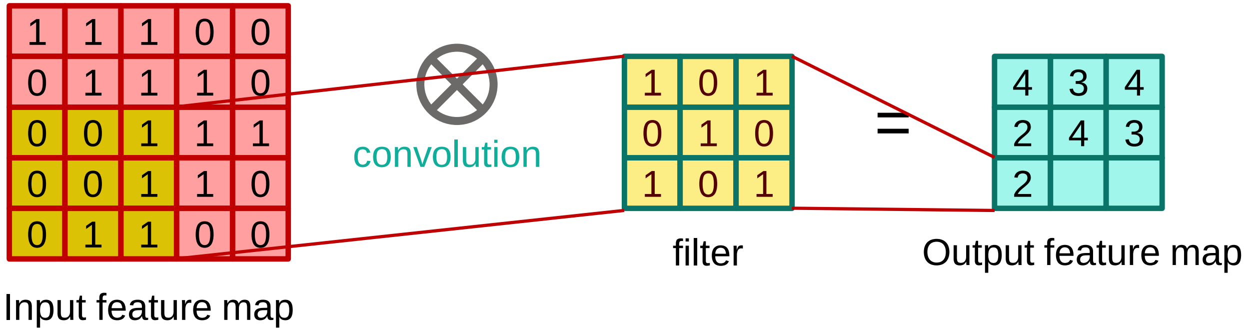
2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 = 3$



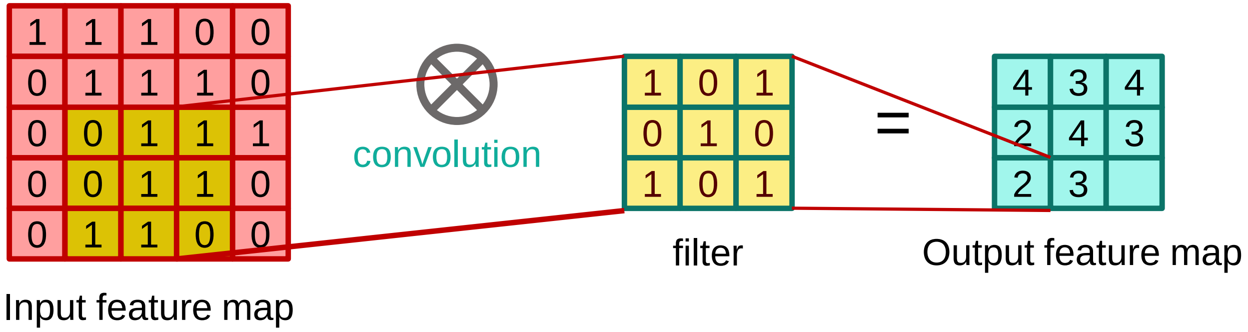
2D Convolution Layer – Computation

- 0x1 + 0x0 + 1x1 + 0x0 + 0x1 + 1x0 + 0x1 + 1x0 + 1x1 = 2



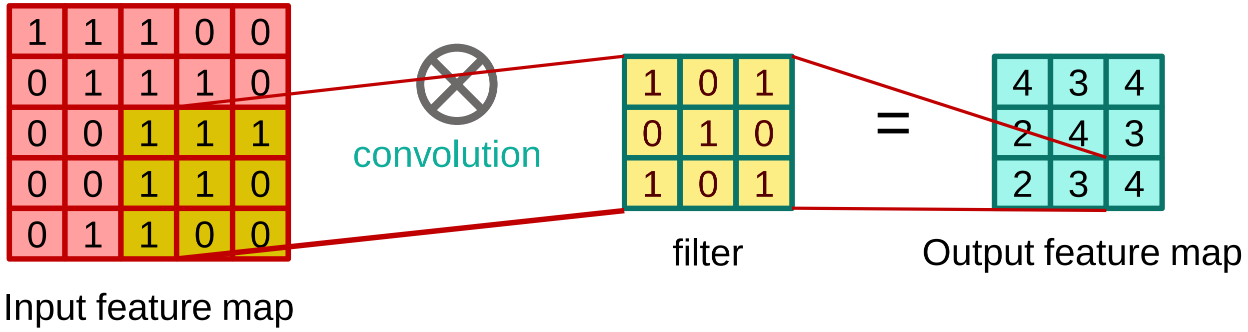
2D Convolution Layer – Computation

- $0 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 = 3$

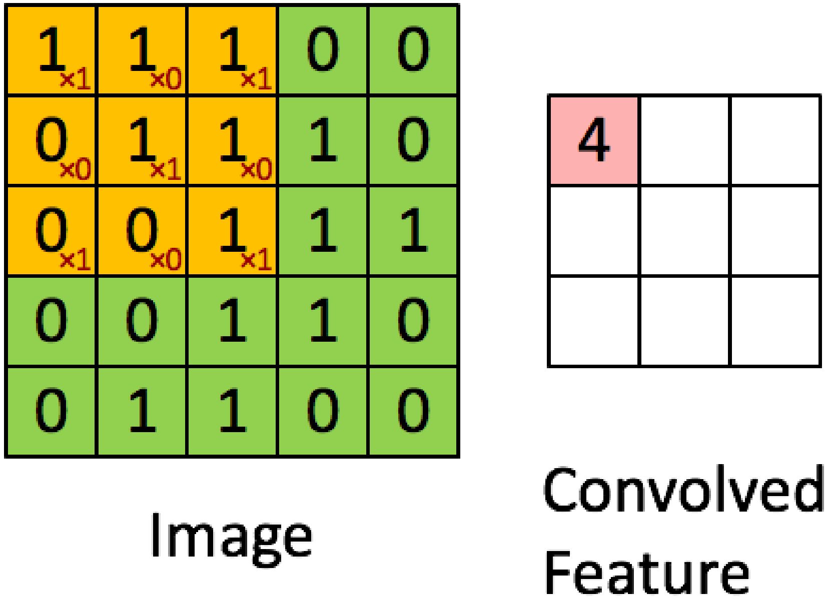


2D Convolution Layer – Computation

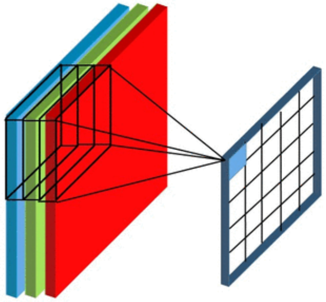
- $1x1 + 1x0 + 1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 0x0 + 0x1 = 4$

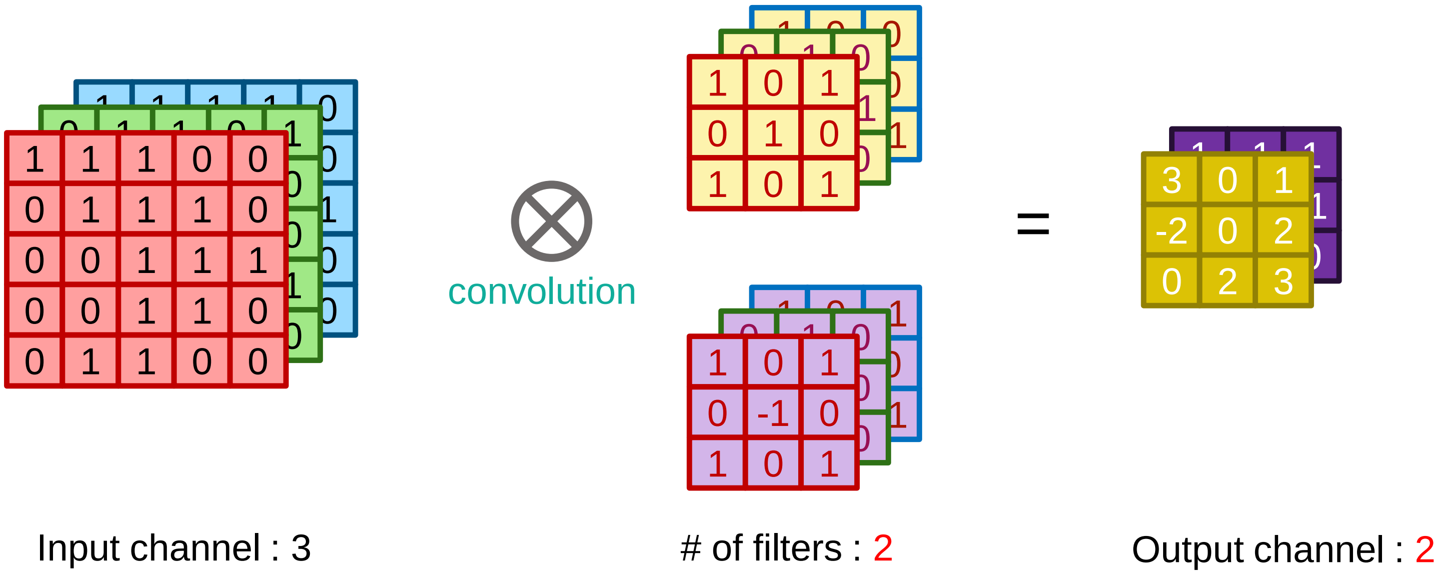


2D Convolution Layer – Computation



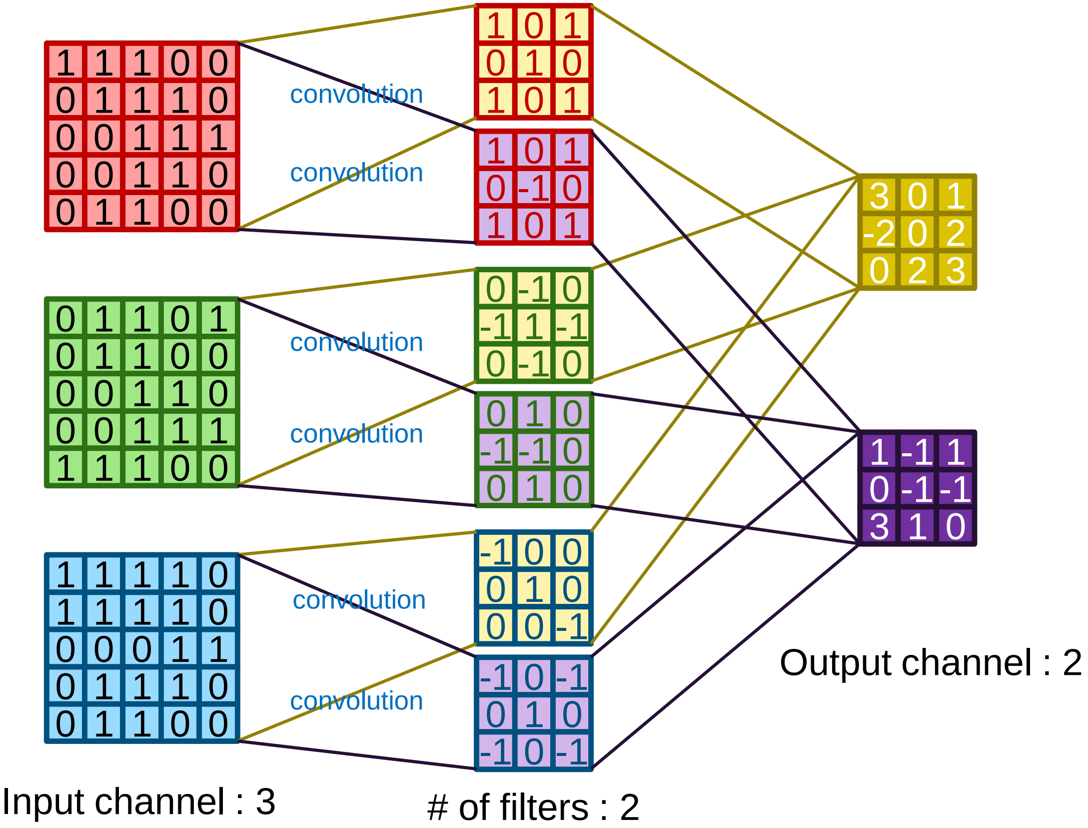
2D Convolution Layer
– Multi Channel, Many Filters



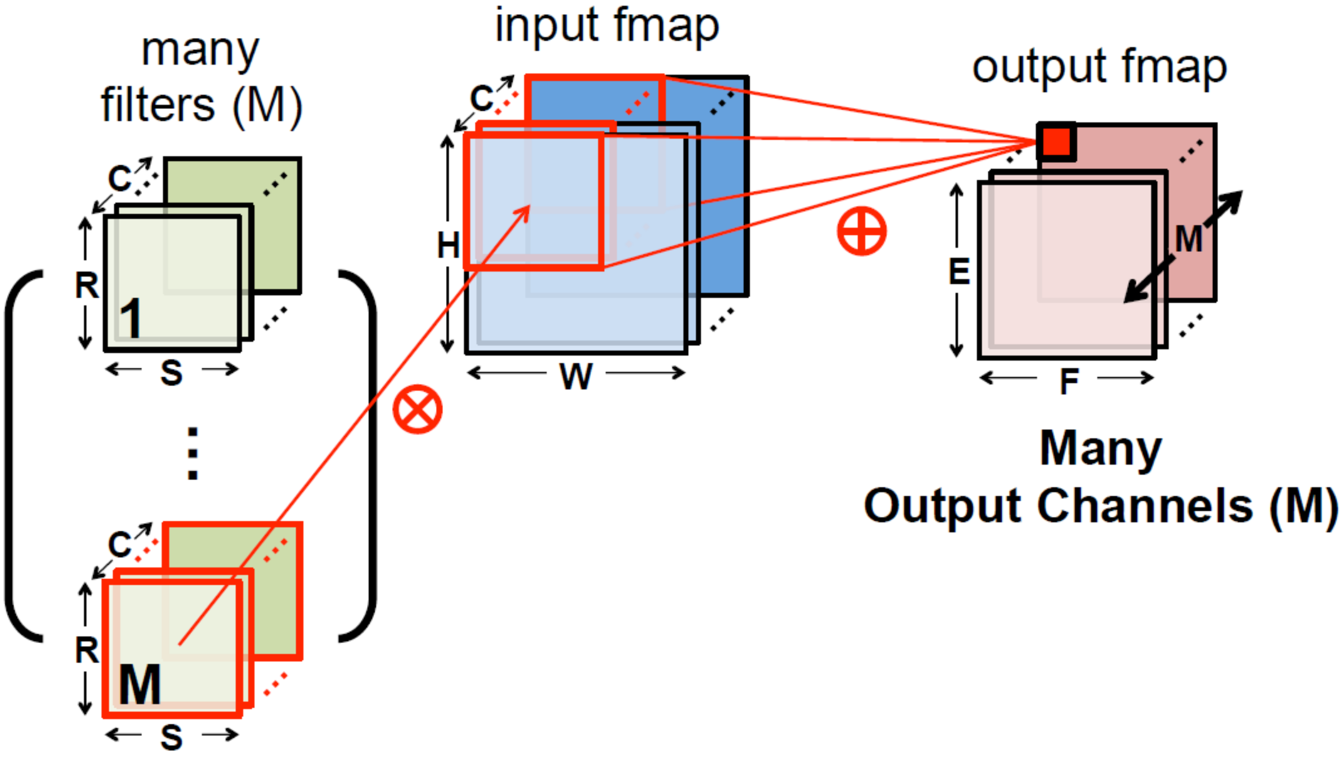


https://predictiveprogrammer.com/famous-convolutional-neural-network-architectures-1/

2D Convolution Layer
– Multi Channel, Many Filters

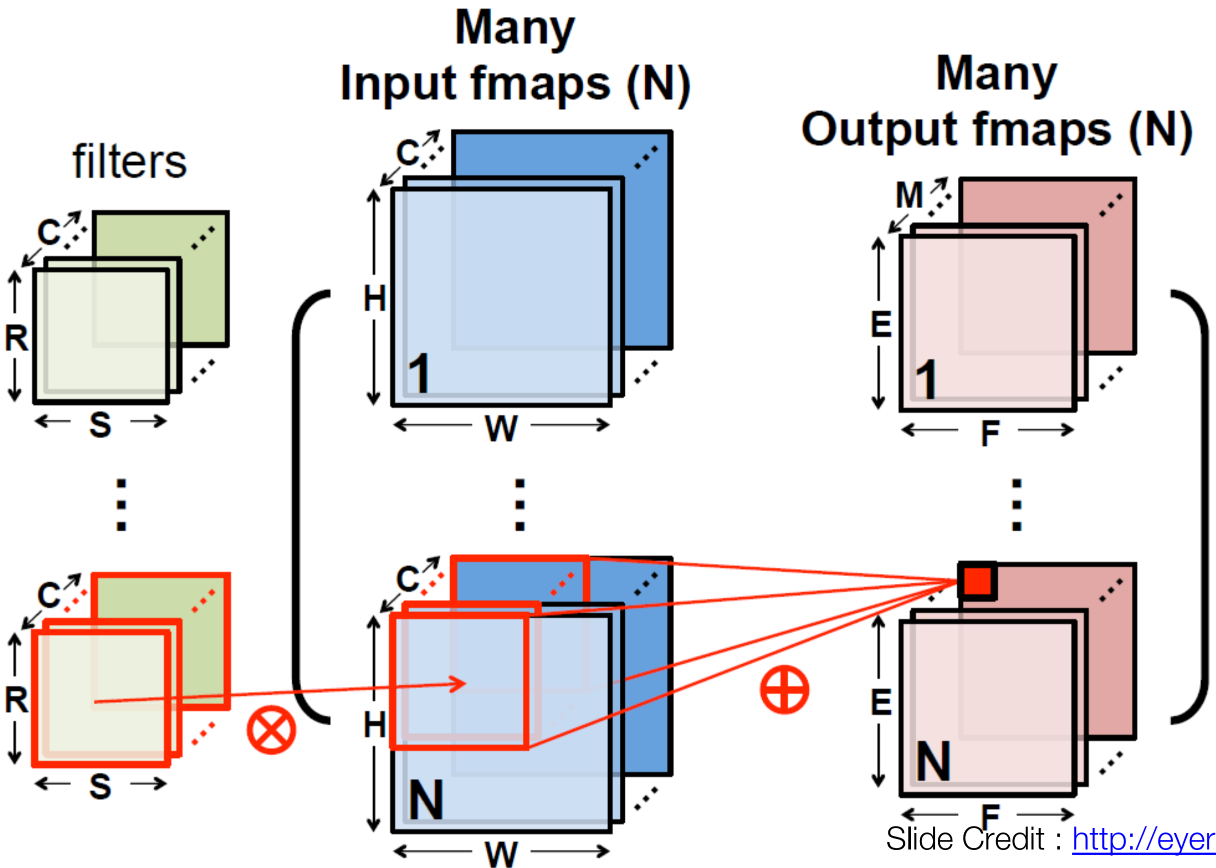


2D Convolution Layer



Slide Credit : http://eyeriss.mit.edu/tutorial.html

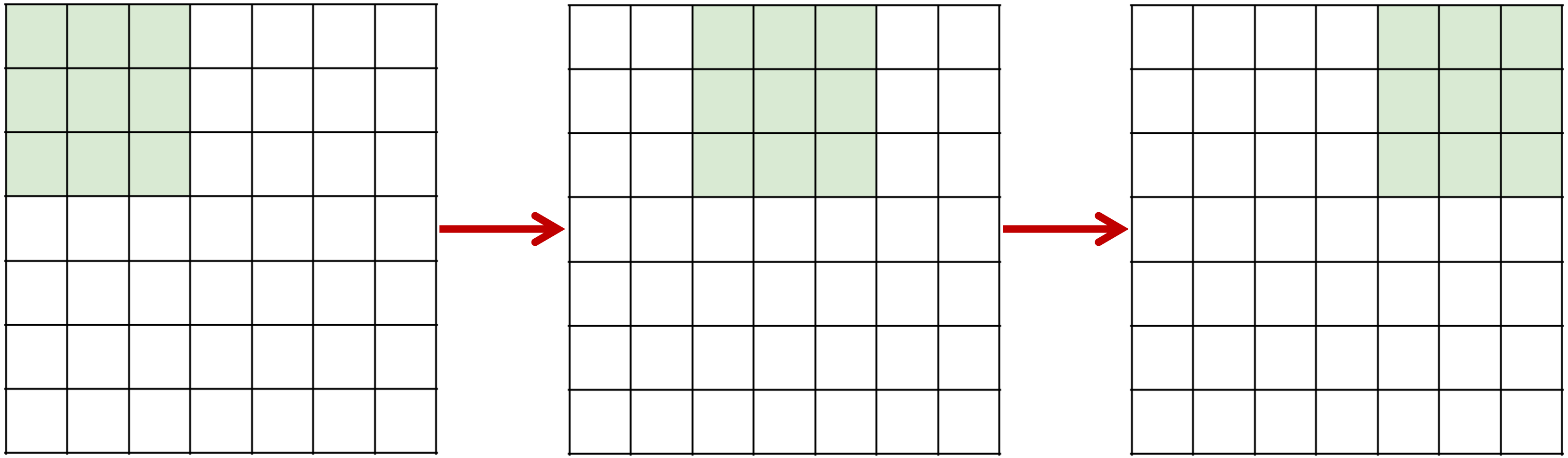
2D Convolution Layer – 4D Tensors



Slide Credit : http://eyeriss.mit.edu/tutorial.html

Options of Convolution

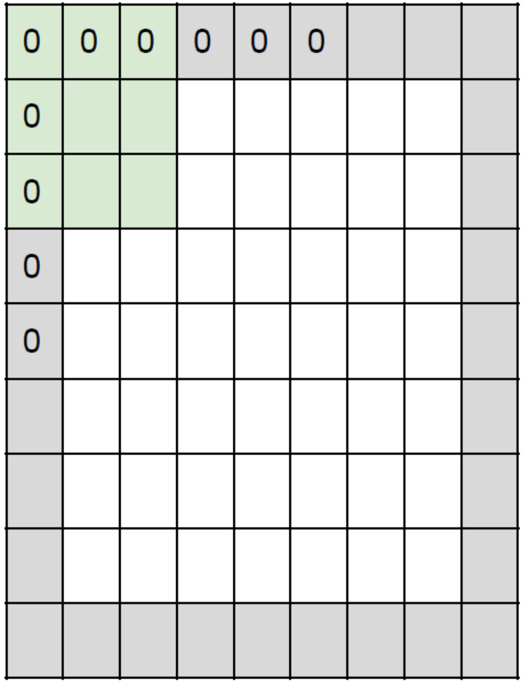
- Stride – How far to go to the right or the bottom to perform the next convolution
  - Ex) 7x7 input, 3x3 convolution filter with stride 2 → 3x3 output



Slide Credit : Stanford CS231n

Options of Convolution

- Zero Padding

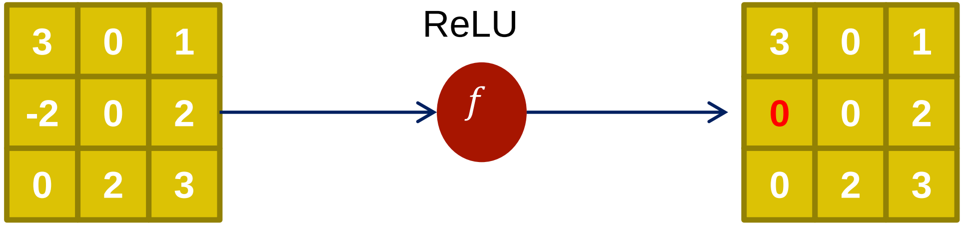


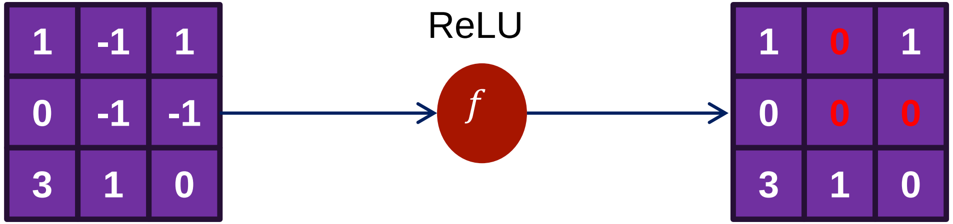
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Activation Function

- ReLU





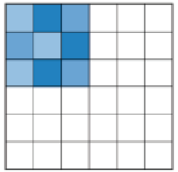
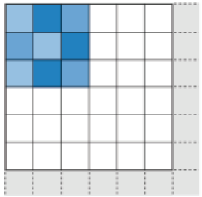
tf.keras.layers.Conv2D

```
__init__(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

tf.keras.layers.Conv2D

- `filters`: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size`: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `strides`: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding`: one of `"valid"` or `"same"` (case-insensitive).
- `data_format`: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Padding – SAME vs VALID

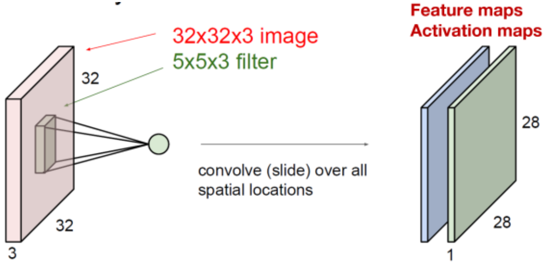
	Valid	Same
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$
Illustration		
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ - Output size is mathematically convenient - Also called 'half' padding

tf.keras.layers.Conv2D

- `activation`: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias`: Boolean, whether the layer uses a bias vector.
- `kernel_initializer`: Initializer for the `kernel` weights matrix.
- `bias_initializer`: Initializer for the bias vector.
- `kernel_regularizer`: Regularizer function applied to the `kernel` weights matrix.
- `bias_regularizer`: Regularizer function applied to the bias vector.

kernel dimension : {height, width, in_channel, out_channel}

Ex) {5, 5, 3, 2}



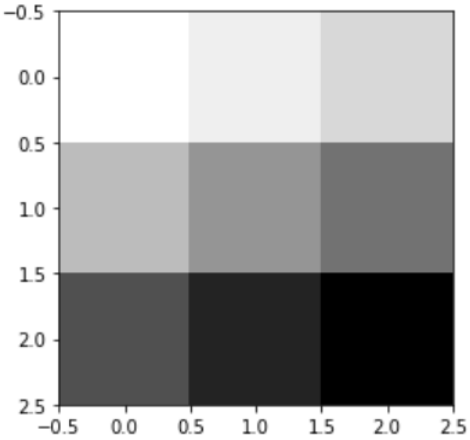
Importing Libraries & Enable Eager Mode

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt

print(tf.__version__)
print(keras.__version__)

tf.enable_eager_execution()
```

Toy Image

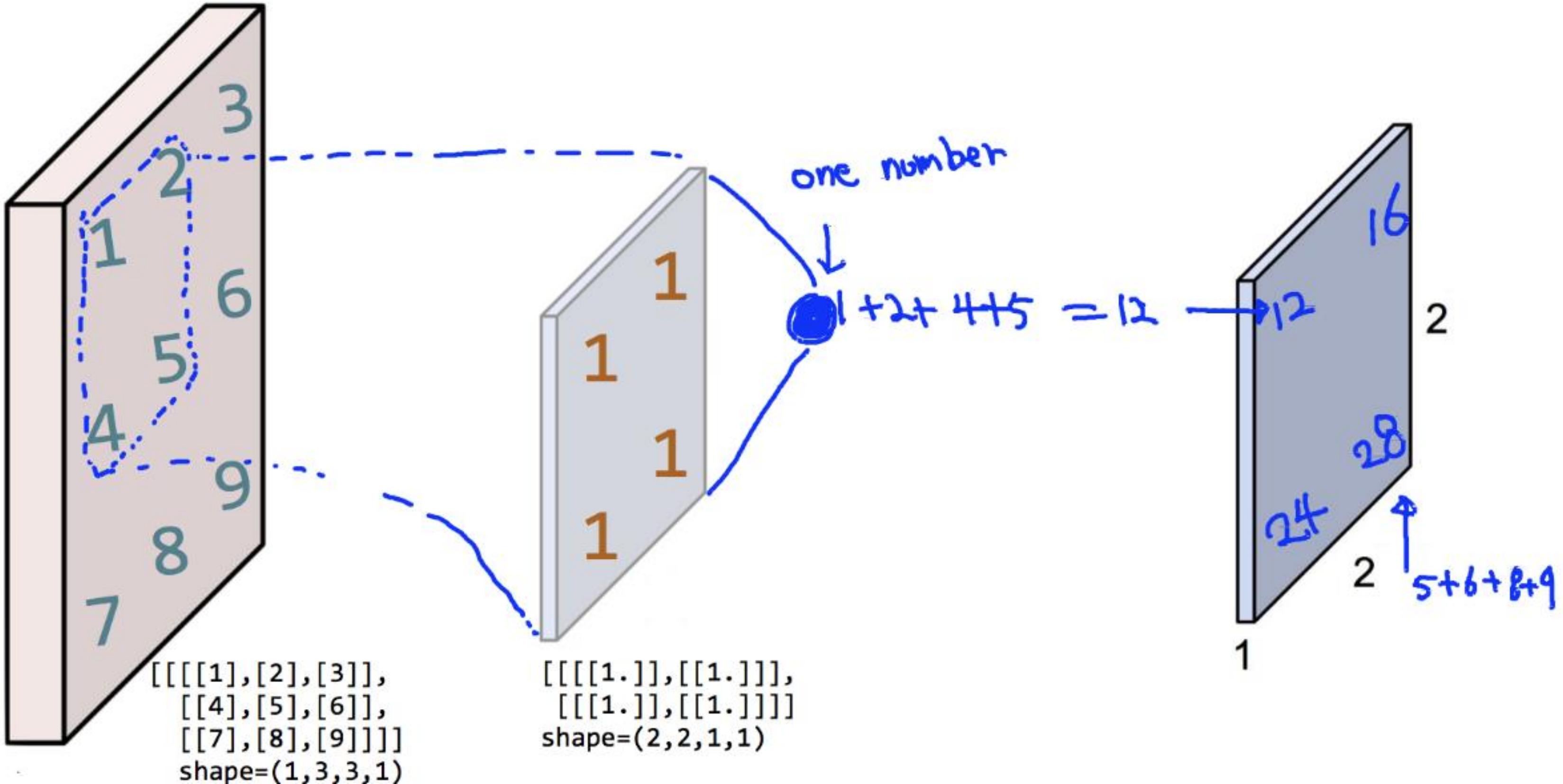
```
image = tf.constant([[[[1],[2],[3]],
                      [[4],[5],[6]],
                      [[7],[8],[9]]]], dtype=np.float32)
print(image.shape)
plt.imshow(image.numpy().reshape(3,3), cmap='Greys')
plt.show()
```

```
(1, 3, 3, 1)
```



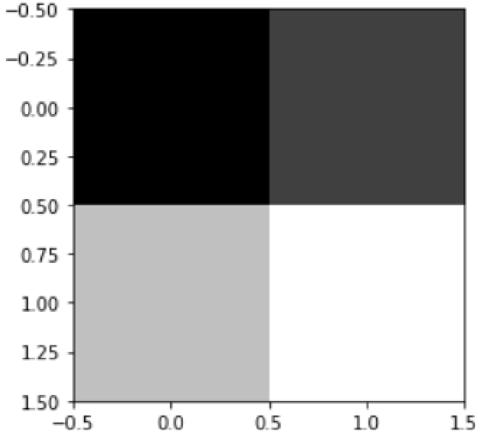
Simple Convolution Layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID



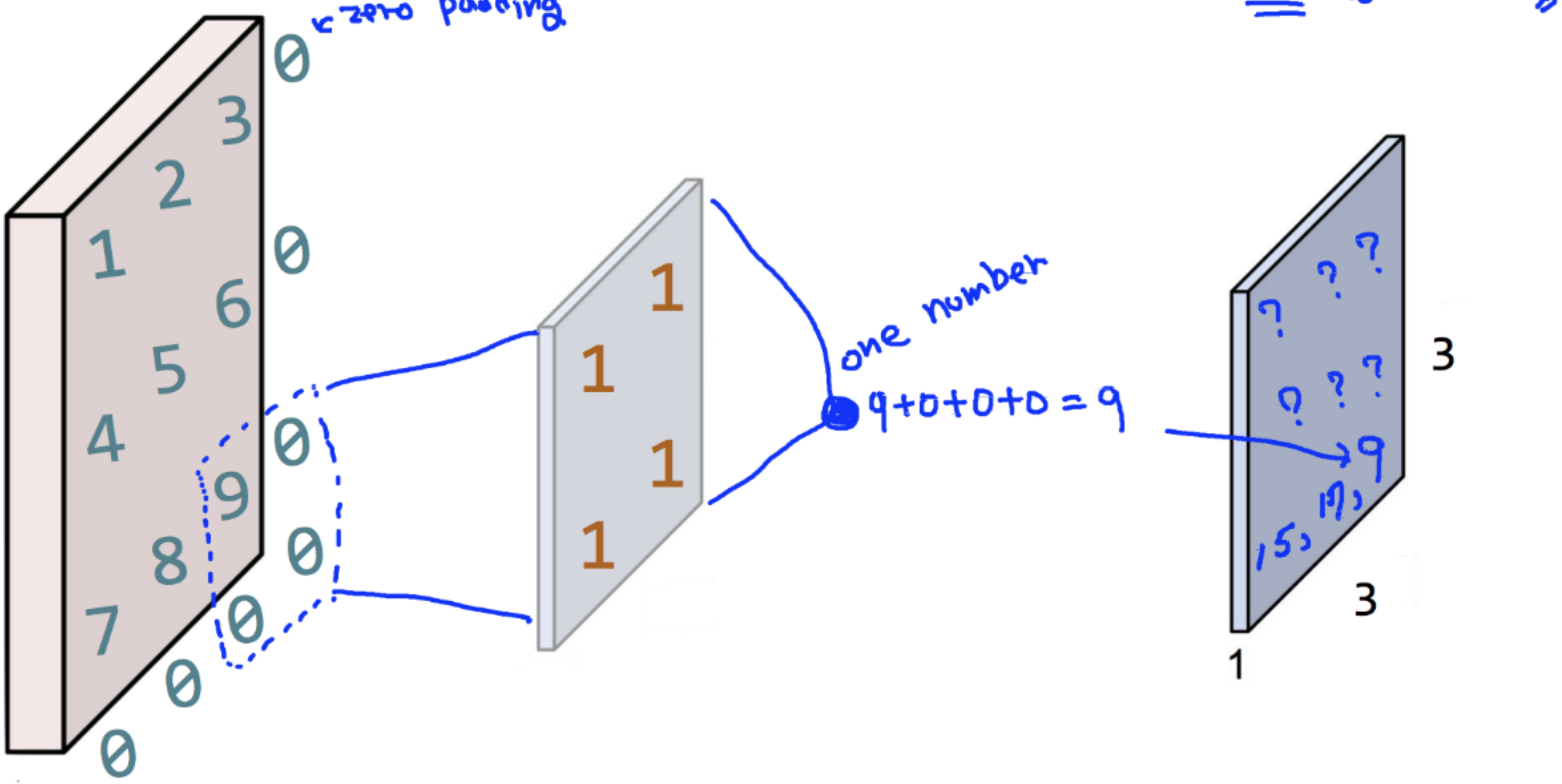
```
print("image.shape", image.shape)
weight = np.array([[[[1.]],[[1.]]],
                   [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
weight_init = tf.constant_initializer(weight)
conv2d = keras.layers.Conv2D(filters=1, kernel_size=2, padding='VALID',
                             kernel_initializer=weight_init)(image)
print("conv2d.shape", conv2d.shape)
print(conv2d.numpy().reshape(2,2))
plt.imshow(conv2d.numpy().reshape(2,2), cmap='gray')
plt.show()
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d.shape (1, 2, 2, 1)
[[12. 16.]
 [24. 28.]]
```



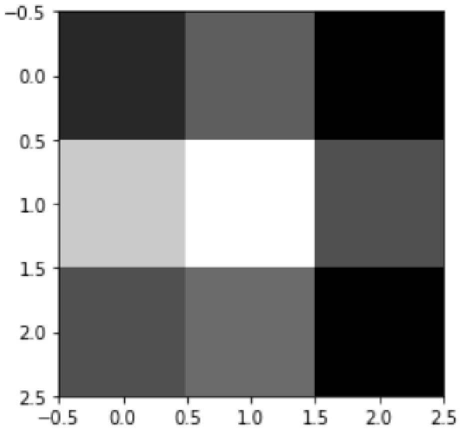
Simple Convolution Layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: SAME want 3x3



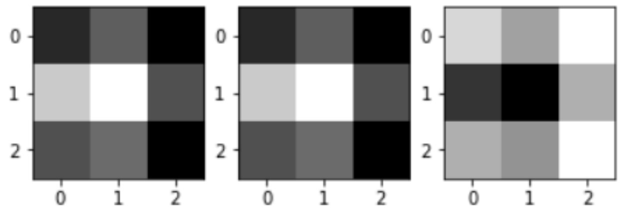
```python
print("image.shape", image.shape)
weight = np.array([[[[1.]],[[1.]]],
                   [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
weight_init = tf.constant_initializer(weight)
conv2d = keras.layers.Conv2D(filters=1, kernel_size=2, padding='SAME',
                             kernel_initializer=weight_init)(image)
print("conv2d.shape", conv2d.shape)
print(conv2d.numpy().reshape(3,3))
plt.imshow(conv2d.numpy().reshape(3,3), cmap='gray')
plt.show()
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d.shape (1, 3, 3, 1)
[[12. 16.  9.]
 [24. 28. 15.]
 [15. 17.  9.]]
```



3 Filters (2, 2, 1, 3)

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
conv2d.shape (1, 3, 3, 3)
[[12. 16.  9.]
 [24. 28. 15.]
 [15. 17.  9.]]
[[120. 160.  90.]
 [240. 280. 150.]
 [150. 170.  90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```



```python
print("image.shape", image.shape)

weight = np.array([[[[1.,10.,-1.]],[[1.,10.,-1.]]],
                   [[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
weight_init = tf.constant_initializer(weight)
conv2d = keras.layers.Conv2D(filters=3, kernel_size=2, padding='SAME',
                             kernel_initializer=weight_init)(image)
print("conv2d.shape", conv2d.shape)
feature_maps = np.swapaxes(conv2d, 0, 3)
for i, feature_map in enumerate(feature_maps):
   print(feature_map.reshape(3,3))
   plt.subplot(1,3,i+1), plt.imshow(feature_map.reshape(3,3), cmap='gray')
plt.show()
```

What's Next?

- CNN Basics – Pooling