Convolutional Neural Networks (CNN)

By Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

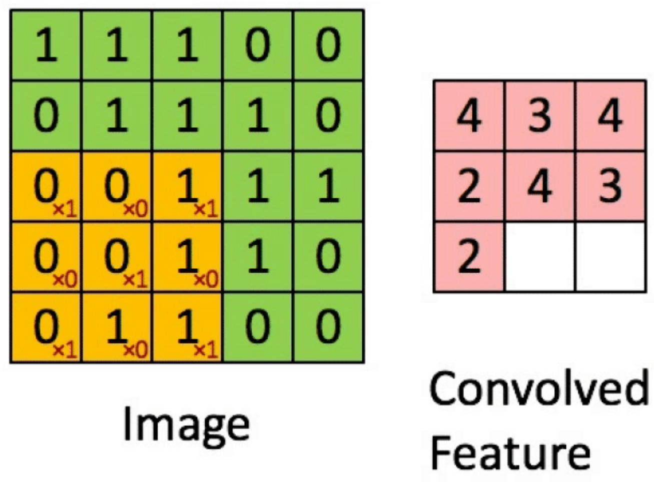
- I. 1. Convolution on Image
- II. 2. Convolutional Neural Networks (CNN)
  - I. 2.1. Convolutional operator
  - II. 2.2. Nonlinear activation function
  - III. 2.3. Pooling
  - IV. 2.4. Inside Convolution Layer
- III. 3. CNN with TensorFlow
  - I. 3.1. Import Library
  - II. 3.2. Load MNIST Data
  - III. 3.3. Build Model
  - IV. 3.4. Define a CNN Shape
  - V. 3.5. Define Weights, Biases and Network
  - VI. 3.6. Define Loss, Initializer and Optimizer
  - VII. 3.7. Summary of Model
  - VIII. 3.8. Define Configuration
  - IX. 3.9. Optimization
  - X. 3.10. Test
- IV. 4. Deep Learning of Things

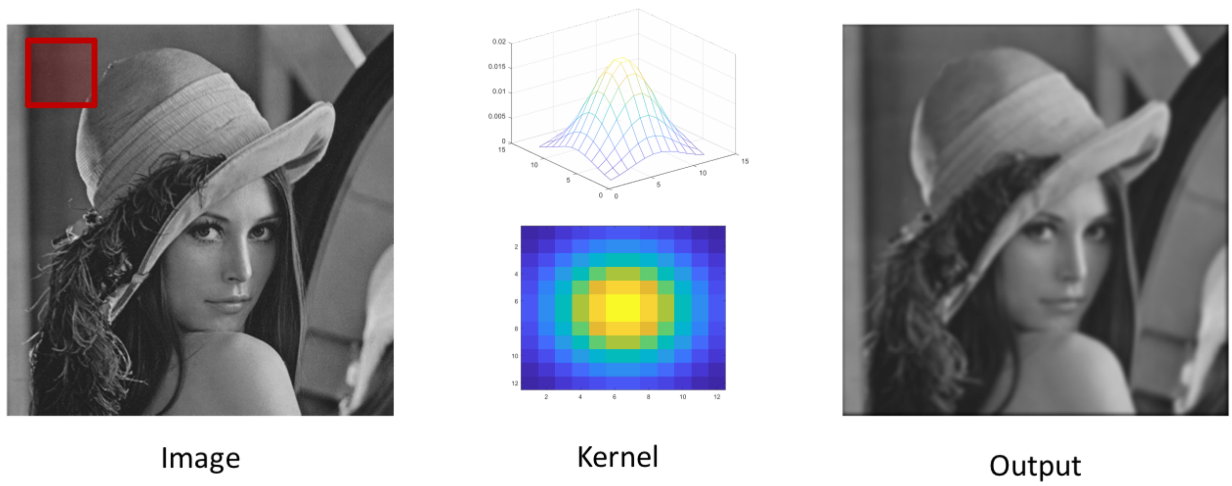
1. Convolution on Image

Filter (or Kernel)

- Modify or enhance an image by filtering
- Filter image to emphasize certain features or remove other features
- Filtering include smoothing, sharpening and edge enhancement

Convolution in 2D





In [20]:

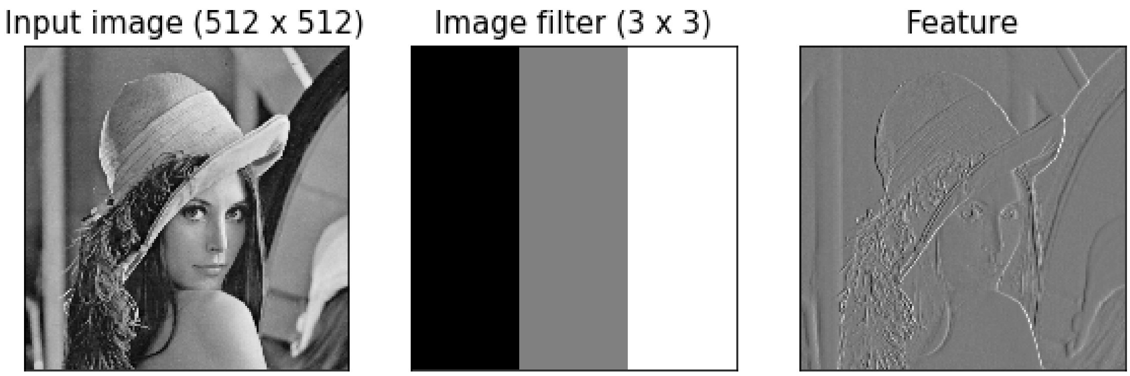
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread, imresize
from scipy.signal import convolve2d
from six.moves import cPickle

% matplotlib inline
```

In [21]:

```
# Import image
input_image = cPickle.load(open('./image_files/lena.pkl', 'rb'))

# Edge filter
image_filter = np.array([[-1, 0, 1]
                        ,[-1, 0, 1]
                        ,[-1, 0, 1]])

# Compute feature
feature = convolve2d(input_image, image_filter, boundary='symm', mode='same')
```

In [22]:

```
# Plot
fig = plt.figure(figsize=(10, 6))
ax1 = fig.add_subplot(1, 3, 1)
ax1.imshow(input_image, 'gray')
ax1.set_title('Input image (512 x 512)', fontsize=15)
ax1.set_xticks([])
ax1.set_yticks([])

ax2 = fig.add_subplot(1, 3, 2)
ax2.imshow(image_filter, 'gray')
ax2.set_title('Image filter (3 x 3)', fontsize=15)
ax2.set_xticks([])
ax2.set_yticks([])

ax3 = fig.add_subplot(1, 3, 3)
ax3.imshow(feature, 'gray')
ax3.set_title('Feature', fontsize=15)
ax3.set_xticks([])
ax3.set_yticks([])
plt.show()
```

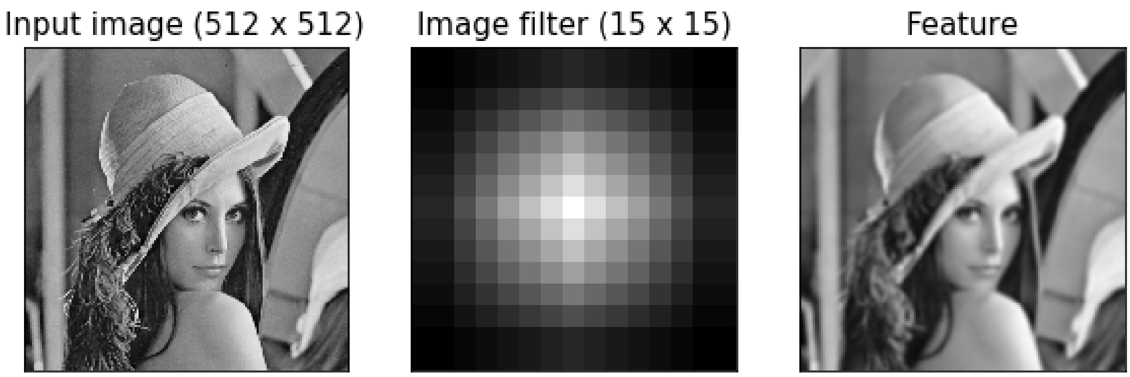


In [23]:

```
# Import image
input_image = cPickle.load(open('./image_files/lena.pkl', 'rb'))

# Gaussian filter
image_filter = 1/273*np.array([[1,  4,  7,  4, 1]
                              ,[4, 16, 26, 16, 4]
                              ,[7, 26, 41, 26, 7]
                              ,[4, 16, 26, 16, 4]
                              ,[1,  4,  7,  4, 1]])
image_filter = imresize(image_filter, [15, 15])

# Compute feature
feature = convolve2d(input_image, image_filter, boundary='symm', mode='same')
```

In [24]:

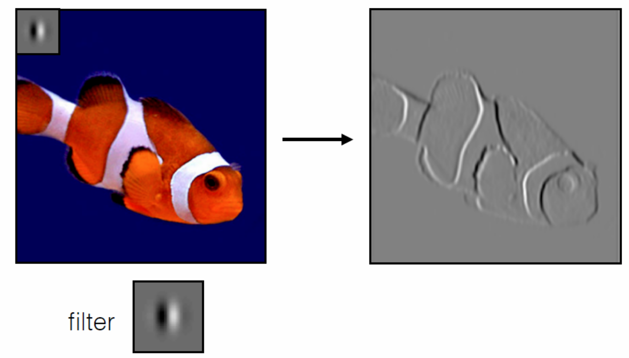
```
# Plot
fig = plt.figure(figsize=(10, 6))
ax1 = fig.add_subplot(1, 3, 1)
ax1.imshow(input_image, 'gray')
ax1.set_title('Input image (512 x 512)', fontsize=15)
ax1.set_xticks([])
ax1.set_yticks([])

ax2 = fig.add_subplot(1, 3, 2)
ax2.imshow(image_filter, 'gray')
ax2.set_title('Image filter (15 x 15)', fontsize=15)
ax2.set_xticks([])
ax2.set_yticks([])

ax3 = fig.add_subplot(1, 3, 3)
ax3.imshow(feature, 'gray')
ax3.set_title('Feature', fontsize=15)
ax3.set_xticks([])
ax3.set_yticks([])
plt.show()
```

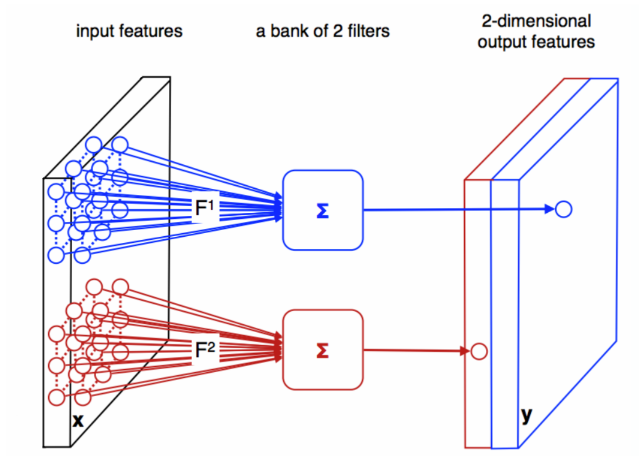


2. Convolutional Neural Networks (CNN)

Convolutional Networks

- Simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers
- Convolution can be interpreted as matrix multiplication

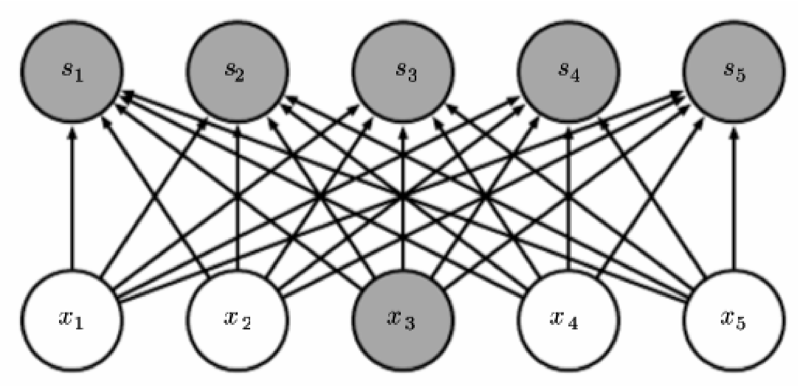




2.1. Convolutional operator

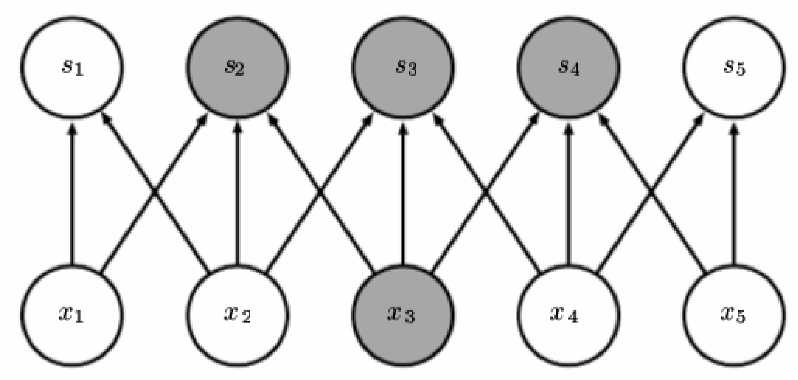
Matrix multiplication

- Every output unit interacts with every interacts unit

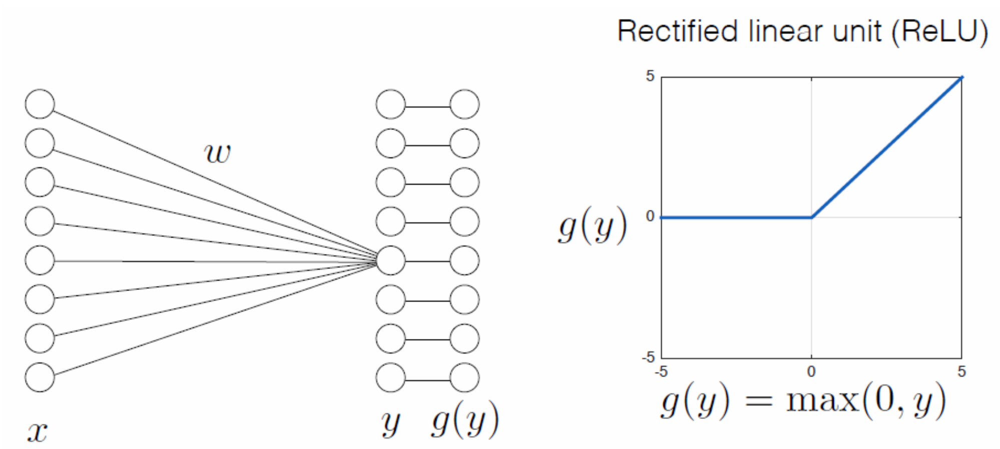


Convolution

- Local connectivity
- Weight sharing
- Typically have sparse interactions
- Accomplished by making the filter smaller than input (sparse interations)

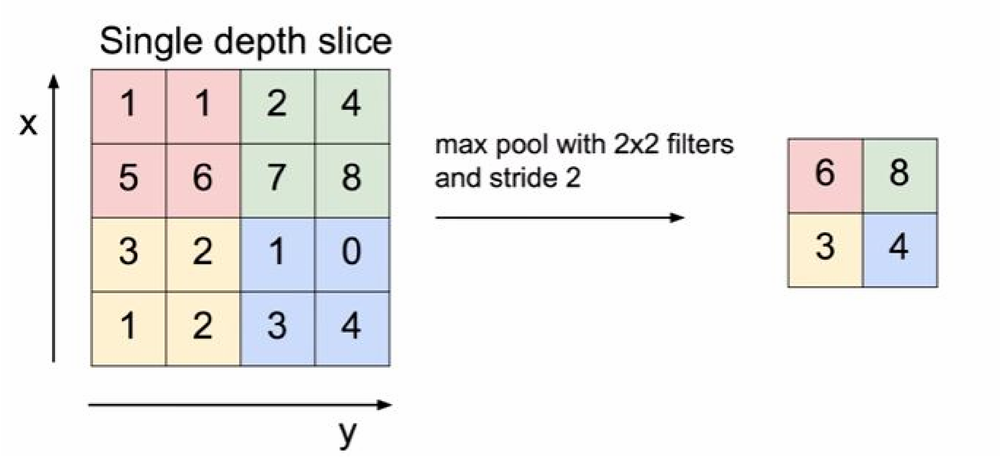


2.2. Nonlinear activation function

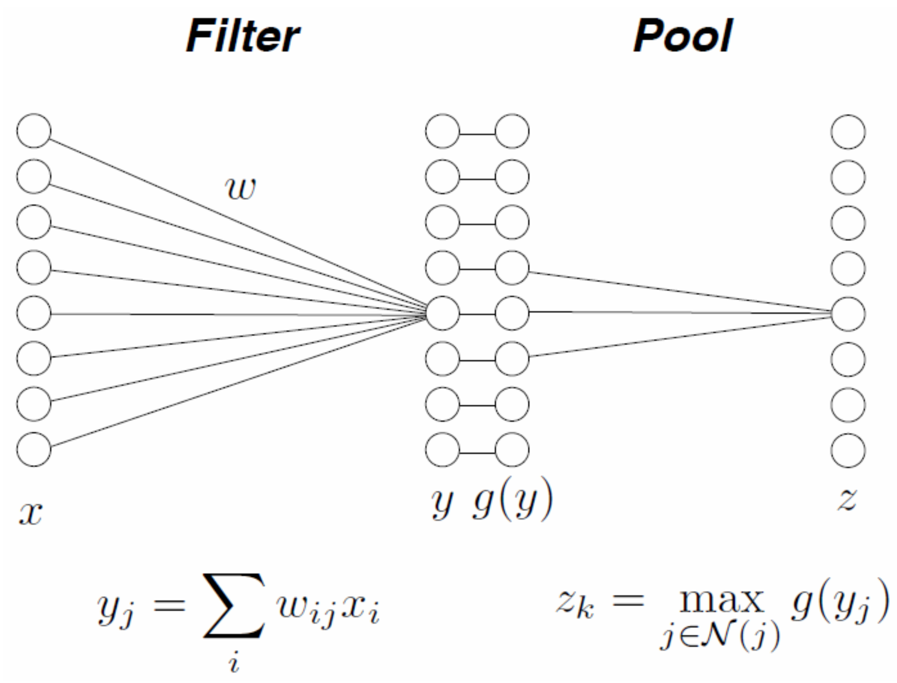


2.3. Pooling

- Compute a maximum value in a sliding window (max pooling)
- Pooling size : 2×2

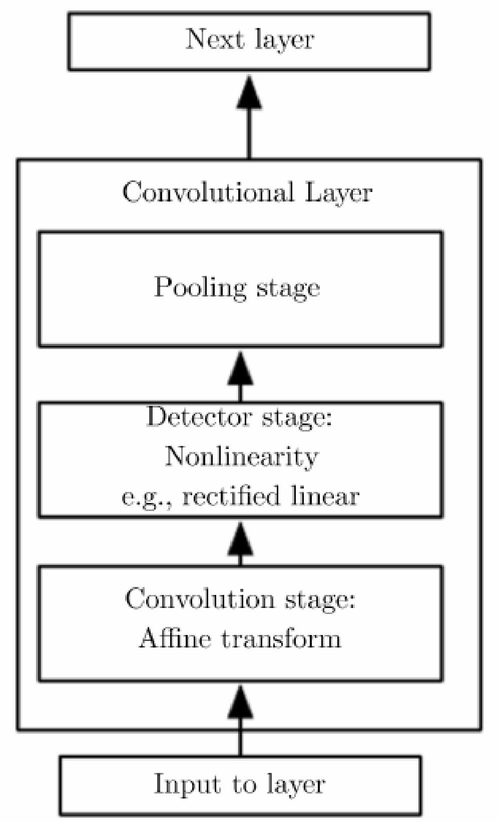


- Max pooling introduces invariances



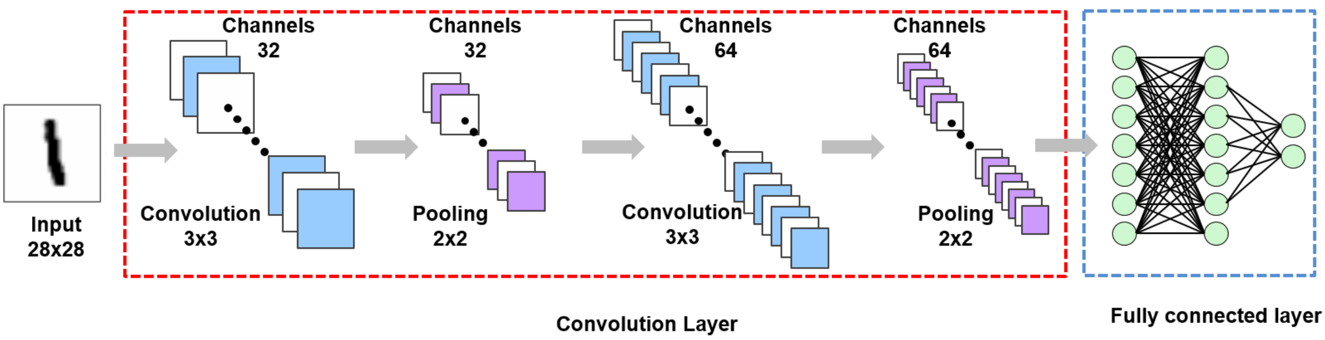
2.4. Inside Convolution Layer

- First, the layer performs several convolutions to produce a set of linear activations
- Second, each linear activation is run through a nonlinear activation function
- Third, use pooling function to modify the output of the layer further



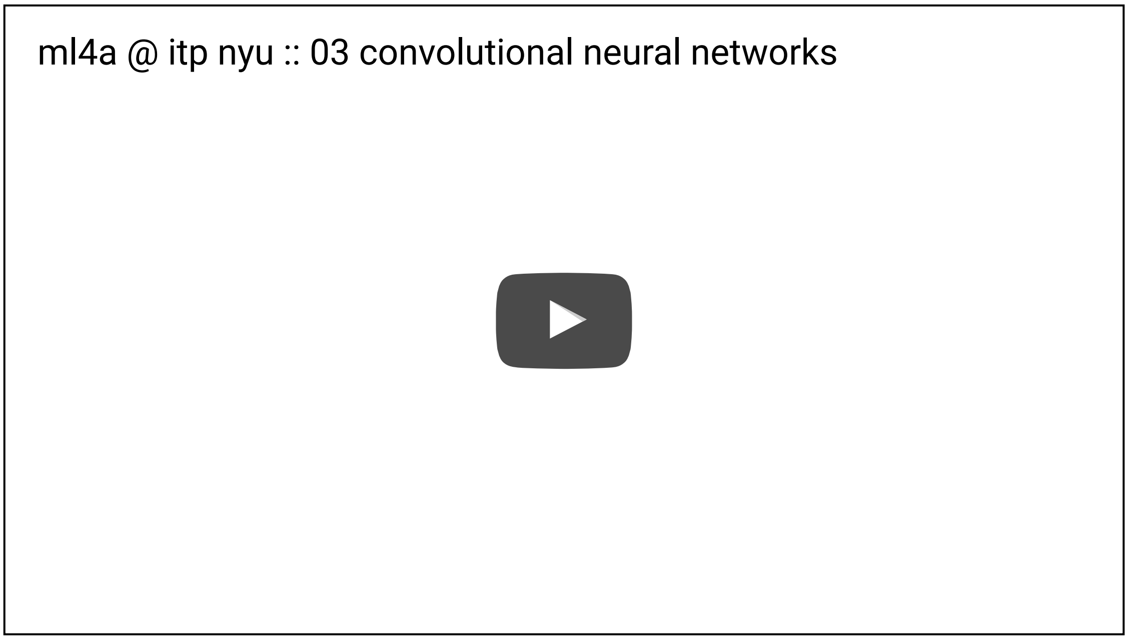
3. CNN with TensorFlow

- MNIST example
- Classifying hand written digits



In [25]:

```
%%html
<center><iframe src="https://www.youtube.com/embed/z6k_RMKExlQ?start=5150&end=6132" width="560" height="315" frameborder="0" allowfullscreen></iframe></center>
```



3.1. Import Library

In [26]:

```
# Import Library
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

3.2. Load MNIST Data

- Download MNIST data from tensorflow tutorial example

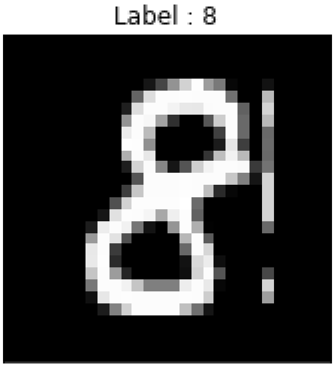
In [27]:

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

In [28]:

```
# Check data
train_x, train_y = mnist.train.next_batch(10)
img = train_x[9,:].reshape(28, 28)

plt.figure(figsize=(5, 3))
plt.imshow(img,'gray')
plt.title("Label : {}".format(np.argmax(train_y[9])))
plt.xticks([])
plt.yticks([])
plt.show()
```



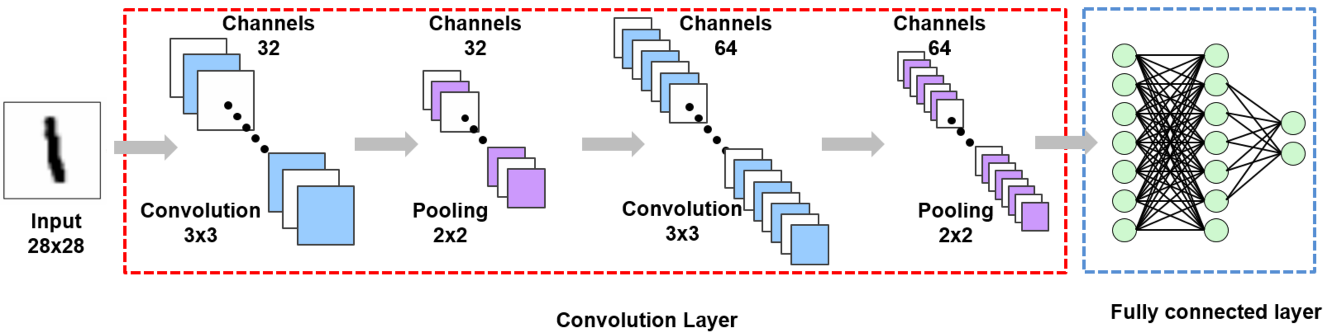
3.3. Build Model

Convolution layers

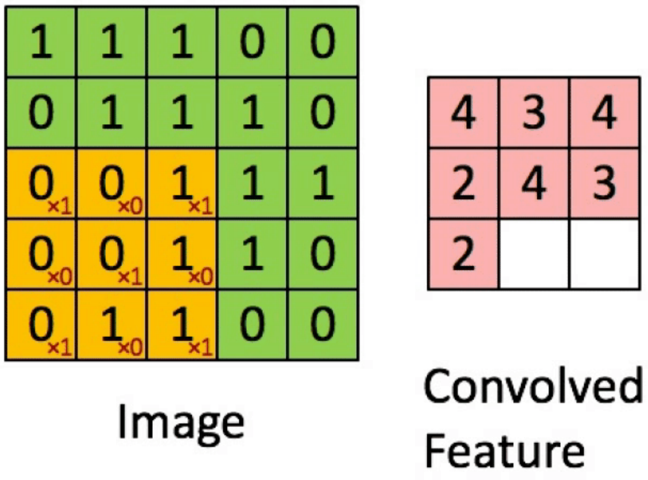
- First, the layer performs several convolutions to produce a set of linear activations
- Second, each linear activation is run through a nonlinear activation function
- Third, use pooling function to modify the output of the layer further

Fully connected layers

- Simple multi layer perceptrons



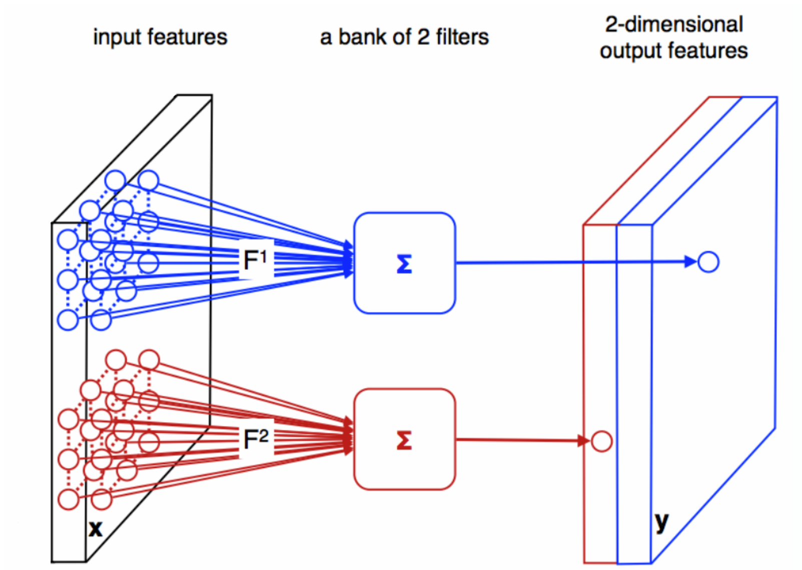
First, the layer performs several convolutions to produce a set of linear activations



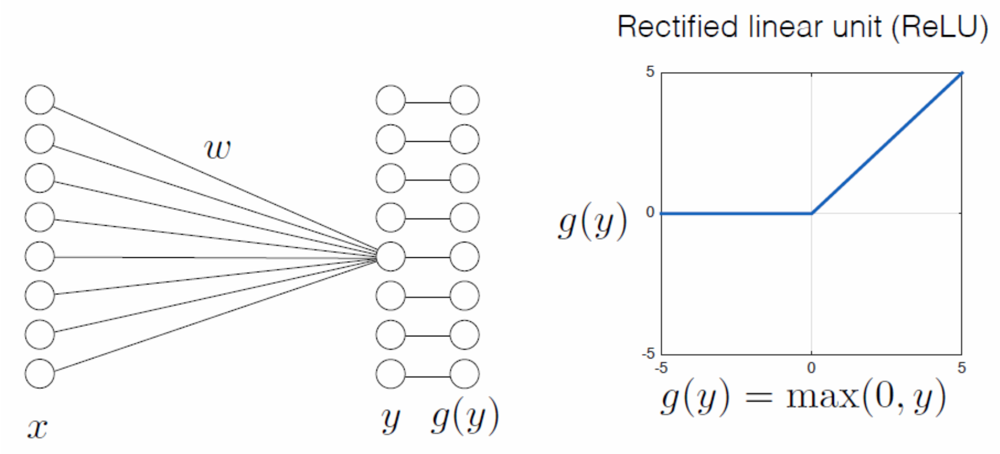
- Filter size : 3×3
- Stride : The stride of the sliding window for each dimension of input
- Padding : Allow us to control the kernel width and the size of the output independently
  - 'SAME' : zero padding
  - 'VALID' : No padding

```
conv1 = tf.nn.conv2d(x, weights['conv1'], strides= [1,1,1,1], padding = 'SAME')
```

- The number of channels: 2



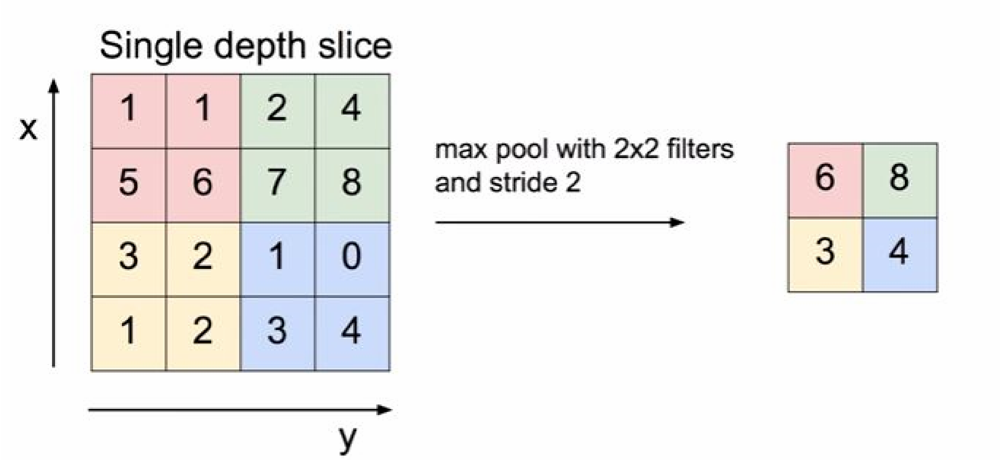
Second, each linear activation is run through a nonlinear activation function



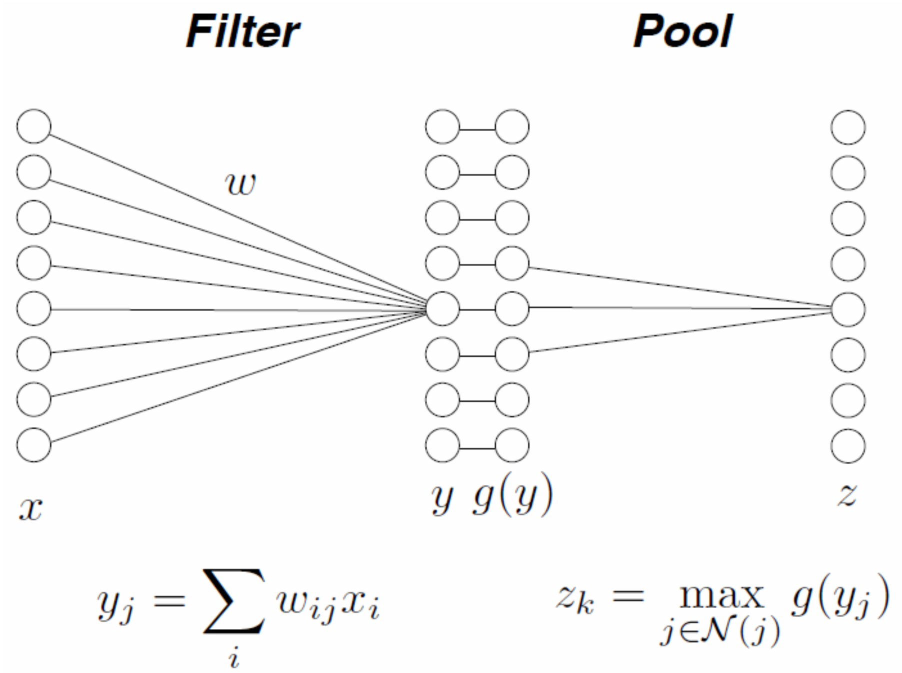

```
conv1 = tf.nn.relu(tf.add(conv1, biases['conv1']))
```

Third, use a pooling function to modify the output of the layer further

- Compute a maximum value in a sliding window (max pooling)



- Pooling size : 2×2
- Max pooling introduces invariances




```
maxp1 = tf.nn.max_pool(conv1,
                       ksize = [1, p1_h, p1_w, 1],
                       strides = [1, p1_h, p1_w, 1],
                       padding ='VALID')
```

Fully connected layer

- Input is typically flattened features


```
output = tf.add(tf.matmul(hidden1, weights['output']), biases['output'])
```

3.4. Define a CNN Shape

In [29]:

```
input_h = 28 # Input height
input_w = 28 # Input width
input_ch = 1 # Input channel : Gray scale
# (None, 28, 28, 1)

## First convolution layer
# Filter size
k1_h = 3
k1_w = 3
# the number of channels
k1_ch = 32
# Pooling size
p1_h = 2
p1_w = 2
# (None, 14, 14 ,32)

## Second convolution layer
# Filter size
k2_h = 3
k2_w = 3
# the number of channels
k2_ch = 64
# Pooling size
p2_h = 2
p2_w = 2
# (None, 7, 7 ,64)

## Fully connected
# Flatten the features
# -> (None, 7*7*64)
conv_result_size = int((28/(2*2)) * (28/(2*2)) * k2_ch)
n_hidden1 = 100
n_output = 10
```

3.5. Define Weights, Biases and Network

- Define parameters based on predefined layer size
- Initialize with normal distribution with $\mu = 0$ and $\sigma = 0.1$

In [30]:

```
weights = {
    'conv1' : tf.Variable(tf.random_normal([k1_h, k1_w, input_ch, k1_ch],stddev = 
0.1)),
    'conv2' : tf.Variable(tf.random_normal([k2_h, k2_w, k1_ch, k2_ch],stddev = 0.1)),
    'hidden1' : tf.Variable(tf.random_normal([conv_result_size, n_hidden1], stddev = 0.
1)),
    'output' : tf.Variable(tf.random_normal([n_hidden1, n_output], stddev = 0.1))
}

biases = {
    'conv1' : tf.Variable(tf.random_normal([k1_ch], stddev = 0.1)),
    'conv2' : tf.Variable(tf.random_normal([k2_ch], stddev = 0.1)),
    'hidden1' : tf.Variable(tf.random_normal([n_hidden1], stddev = 0.1)),
    'output' : tf.Variable(tf.random_normal([n_output], stddev = 0.1))
}

x = tf.placeholder(tf.float32, [None, input_h, input_w, input_ch])
y = tf.placeholder(tf.float32, [None, n_output])
```

In [31]:

```
# Define Network
def net(x, weights, biases):
    ## First convolution layer
    conv1 = tf.nn.conv2d(x, weights['conv1'], 
                         strides= [1, 1, 1, 1], 
                         padding = 'SAME')
    conv1 = tf.nn.relu(tf.add(conv1, biases['conv1']))
    maxp1 = tf.nn.max_pool(conv1, 
                           ksize = [1, p1_h, p1_w, 1], 
                           strides = [1, p1_h, p1_w, 1], 
                           padding = 'VALID'
                           )
    
    ## Second convolution layer
    conv2 = tf.nn.conv2d(maxp1, weights['conv2'], 
                         strides= [1, 1, 1, 1], 
                         padding = 'SAME')
    conv2 = tf.nn.relu(tf.add(conv2, biases['conv2']))
    maxp2 = tf.nn.max_pool(conv2, 
                           ksize = [1, p2_h, p2_w, 1], 
                           strides = [1, p2_h, p2_w, 1], 
                           padding = 'VALID')

    # shape = conv2.get_shape().as_list()
    # maxp2_re = tf.reshape(conv2, [-1, shape[1]*shape[2]*shape[3]])
    maxp2_re = tf.reshape(maxp2, [-1, conv_result_size])
    
    ### Fully connected
    hidden1 = tf.add(tf.matmul(maxp2_re, weights['hidden1']), biases['hidden1'])
    hidden1 = tf.nn.relu(hidden1)
    output = tf.add(tf.matmul(hidden1, weights['output']), biases['output'])
    return output
```

3.6. Define Loss, Initializer and Optimizer

Loss

- Classification: Cross entropy
  - Equivalent to apply logistic regression

$$-\frac{1}{N}\sum_{i=1}^{N} y^{(i)}\log(h_\theta\left(x^{(i)}\right)) + (1-y^{(i)})\log(1-h_\theta\left(x^{(i)}\right))$$

Initializer

- Initialize all the empty variables

Optimizer

- GradientDescentOptimizer
- AdamOptimizer: the most popular optimizer

In [32]:

```
LR = 0.0001

pred = net(x, weights, biases)
loss = tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=pred)
loss = tf.reduce_mean(loss)

# optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
optm = tf.train.AdamOptimizer(LR).minimize(loss)

init = tf.global_variables_initializer()
```