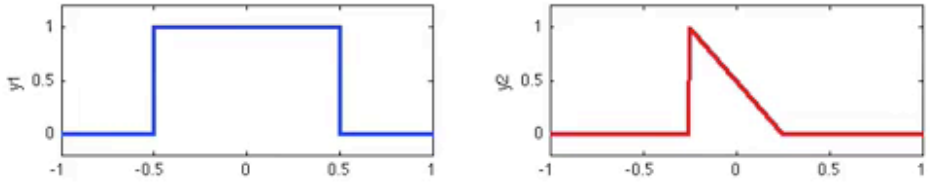
Convolutional Neural Networks (CNN)

Industrial AI Lab.

Convolution on Signal

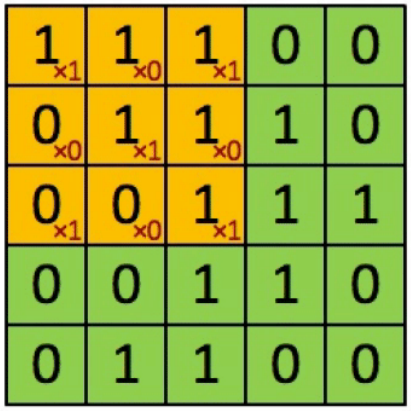
- Convolution in 1D



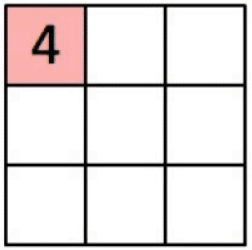
Convolution on Image

- Convolution in 2D

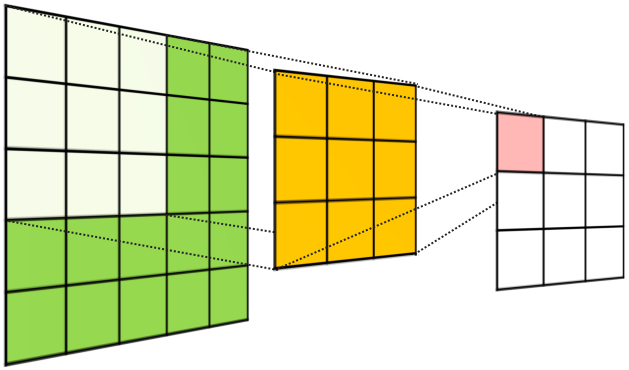
- Filter (or Kernel)
  - Modify or enhance an image by filtering
  - Filter images to emphasize certain features or remove other features
  - Filtering includes smoothing, sharpening and edge enhancement
  - Discrete convolution can be viewed as **element-wise multiplication** by a matrix



Image



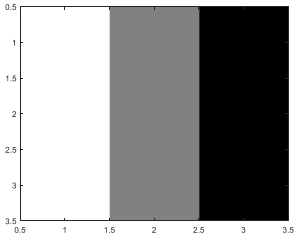
Convolved Feature



Convolution on Image

Image

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



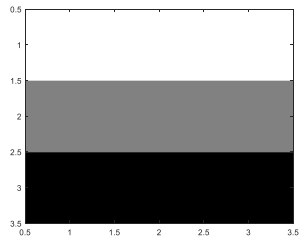
Kernel

Output

Convolution on Image

Image

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



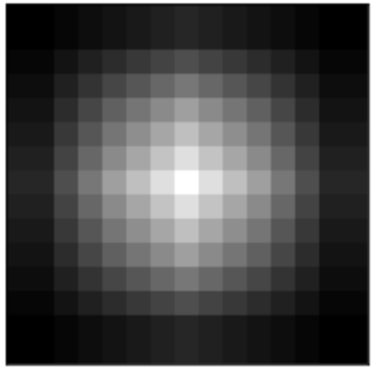
Kernel

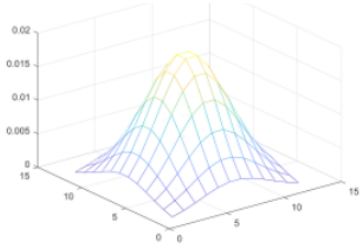
Output

Convolution on Image

Input image (512 x 512)

Image filter (15 x 15)



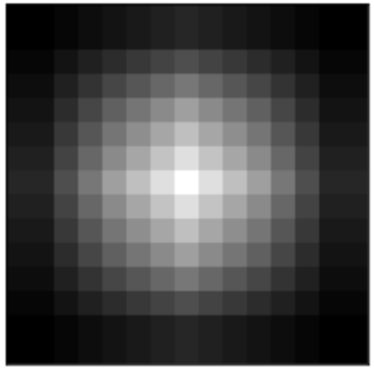


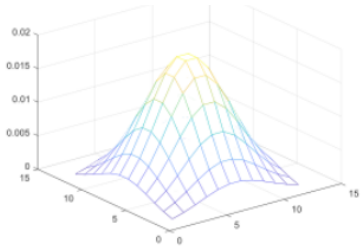
Feature

Convolution on Image

Input image (512 x 512)

Image filter (15 x 15)



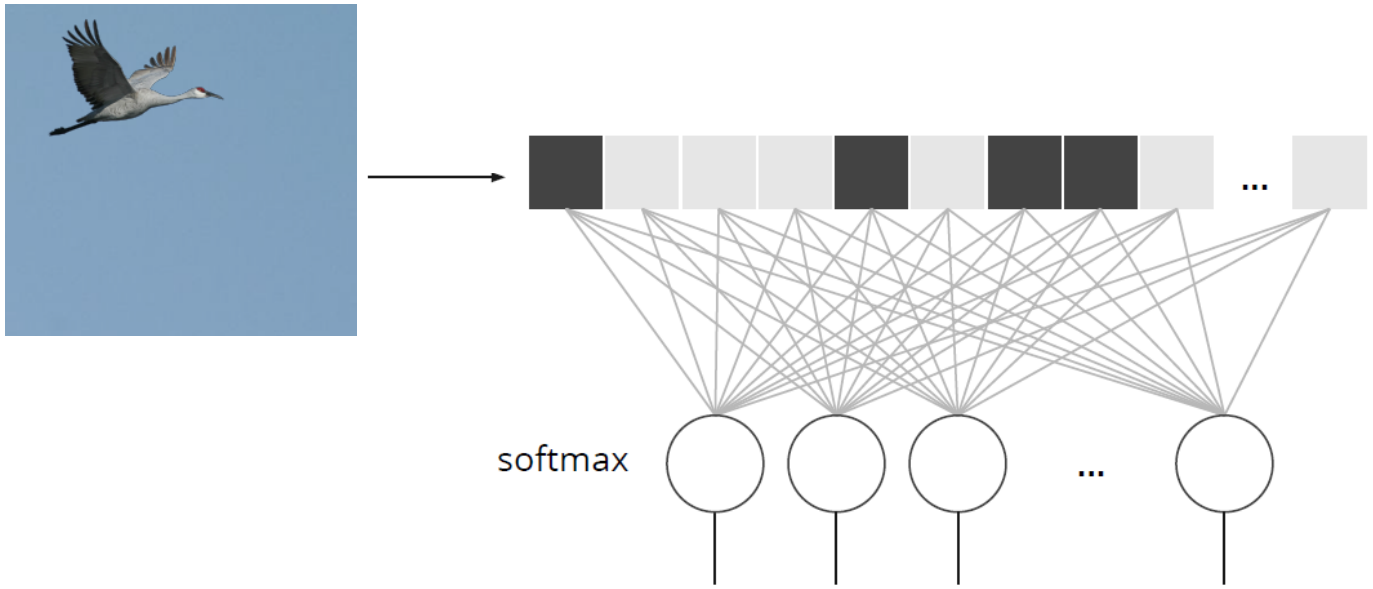


Feature

Convolutional Neural Networks (CNN)

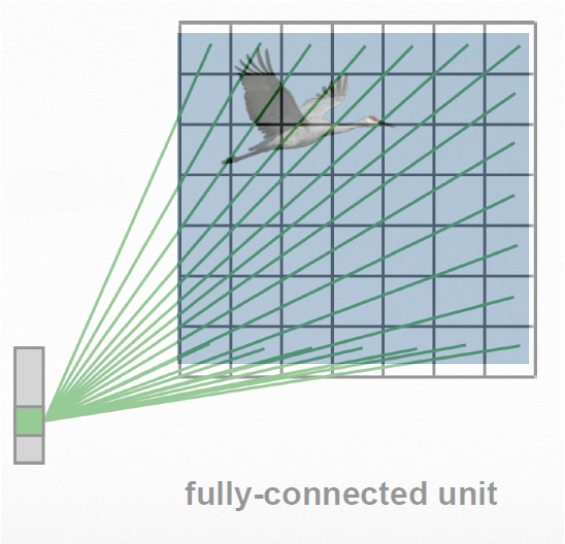
- Motivation
  - The bird occupies a local area and looks the same in different parts of an image. We should construct neural networks which exploit these properties.

Generic Structure of Neural Network



- does not seem the best
- did not make use of the fact that we are dealing with images

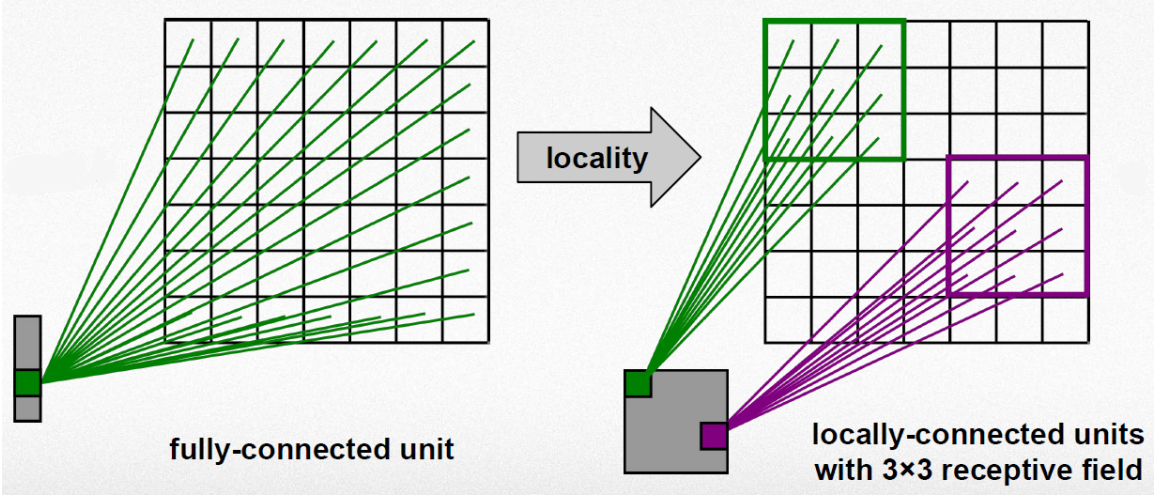
Generic Structure of Neural Network



- does not seem the best
- did not make use of the fact that we are dealing with images

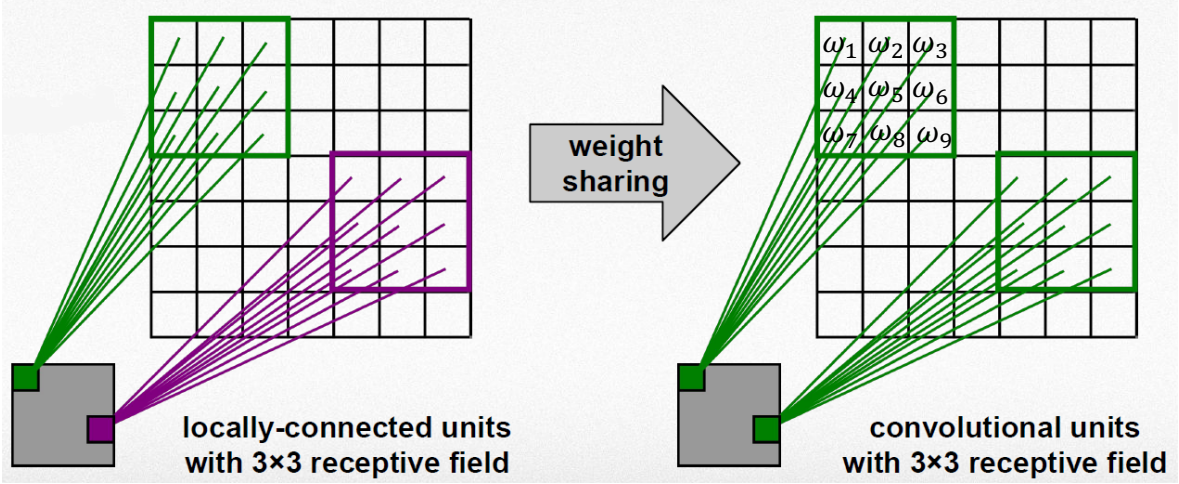
Locality

- Locality: objects tend to have a local spatial support
  - fully-connected layer → locally-connected layer

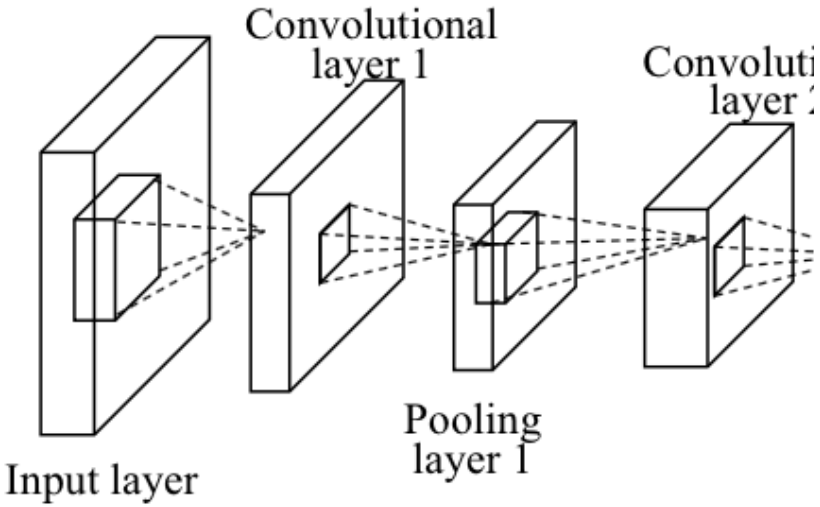


Translation Invariance

- Translation invariance: object appearance is independent of location
  - Weight sharing: units connected to different locations have the same weights

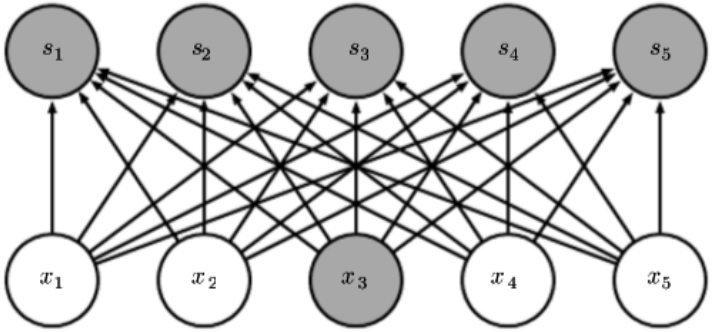


Object Size?

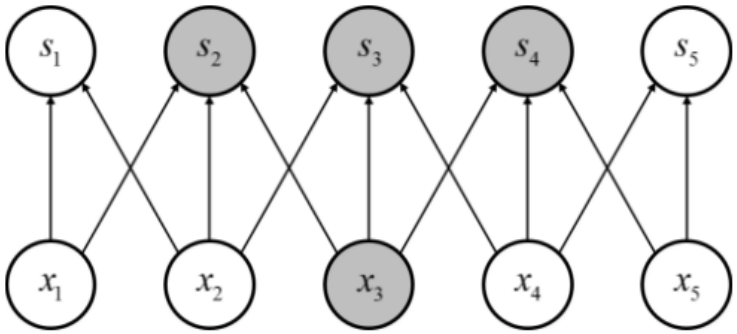

Convolutional
layer 1
Convoluti
layer
Pooling
layer 1
Input layer

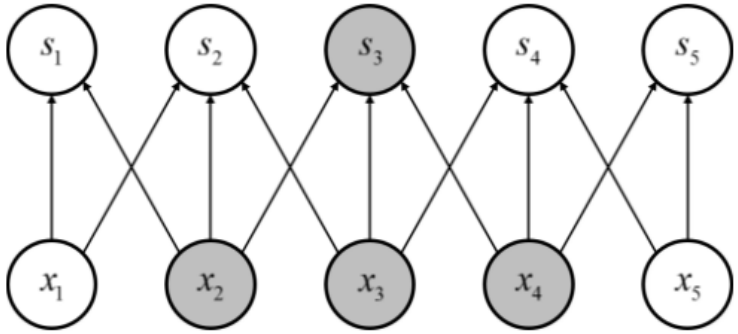
Convolutional Neural Networks (CNN)

- Matrix multiplication
  - Every output unit interacts with every interacts unit



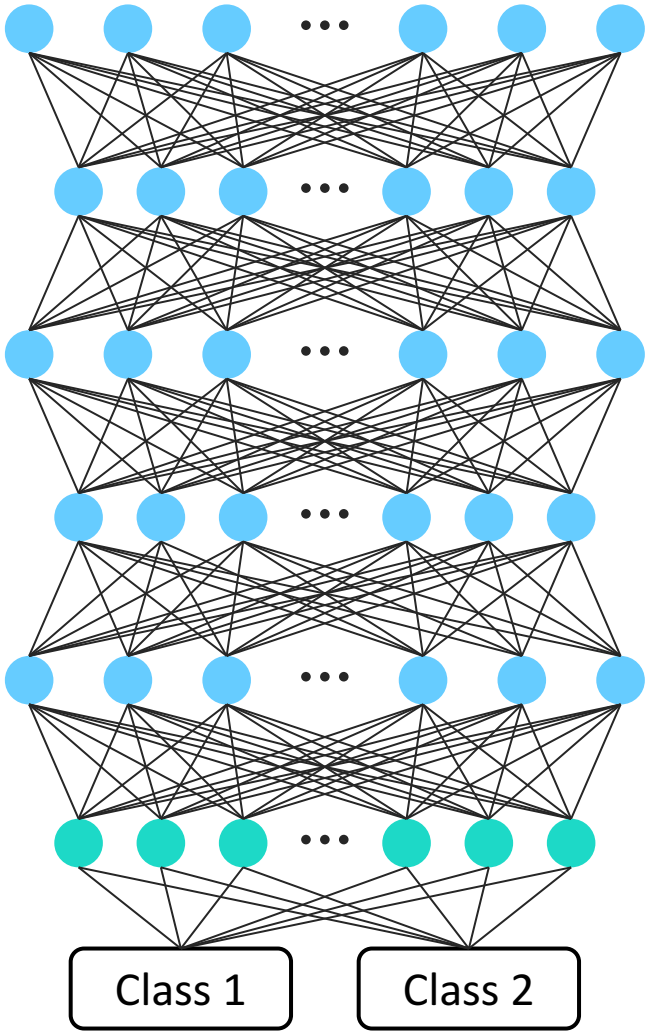
- Convolution
  - Typically have sparse interactions
  - Local connectivity





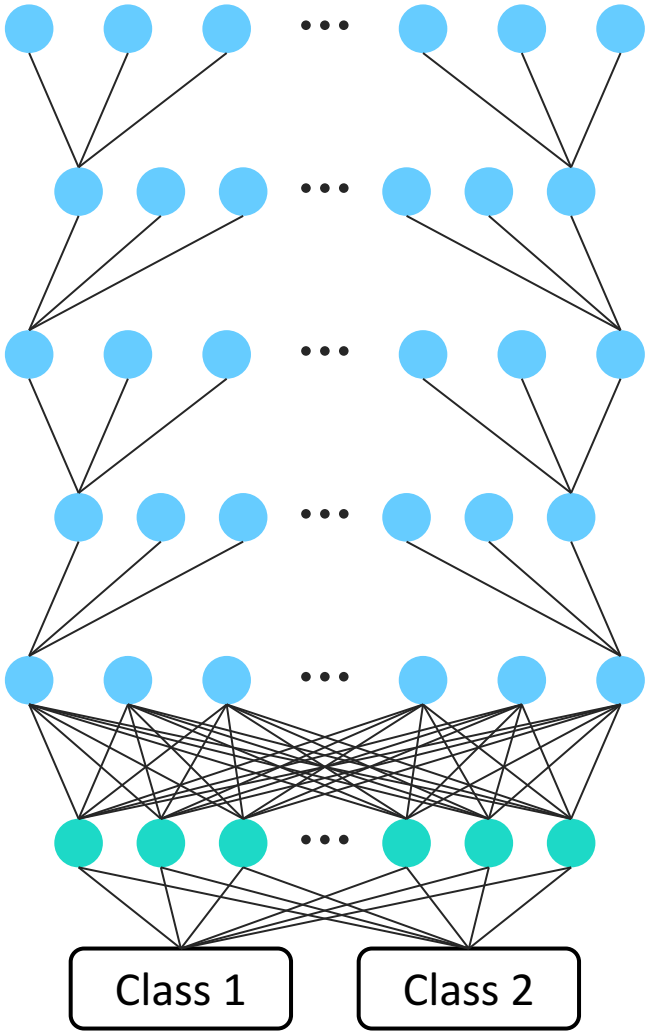
Artificial Neural Networks

- Complex function approximator
  - Simple nonlinear neurons
  - Linear connected networks
- Hidden layers
  - Autonomous feature learning

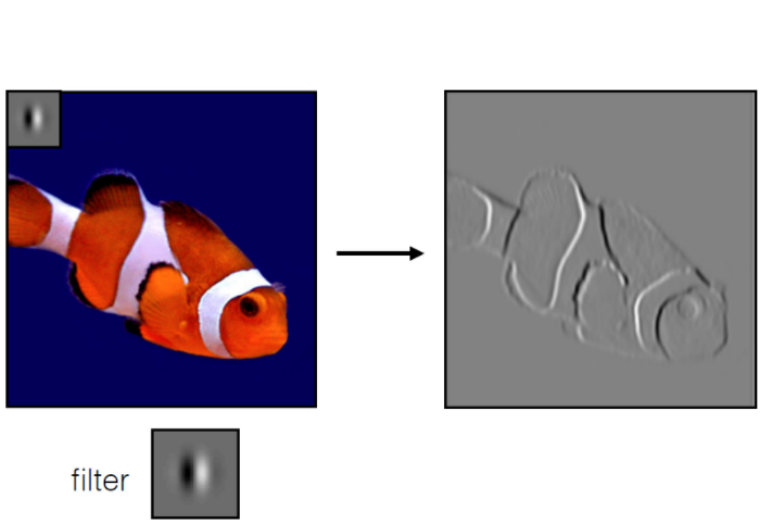


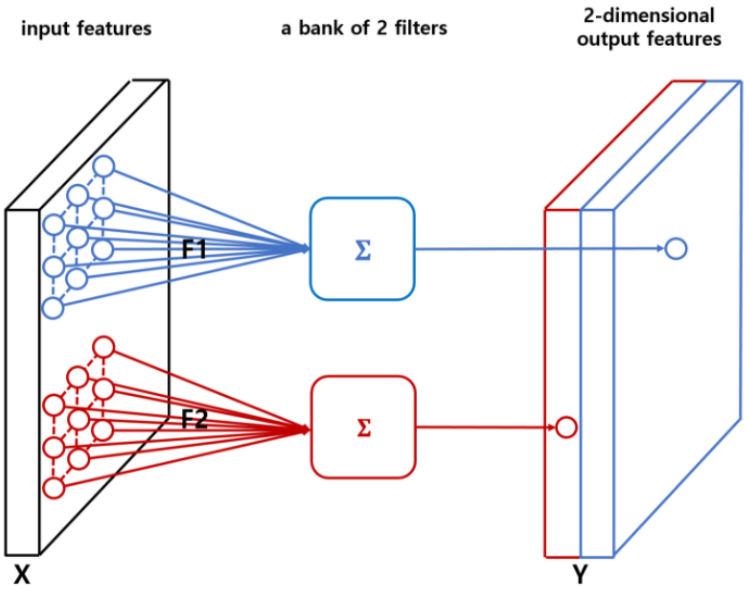
Convolutional Neural Networks

- Structure
  - Weight sharing
  - Local connectivity
- Optimization
  - Smaller searching space

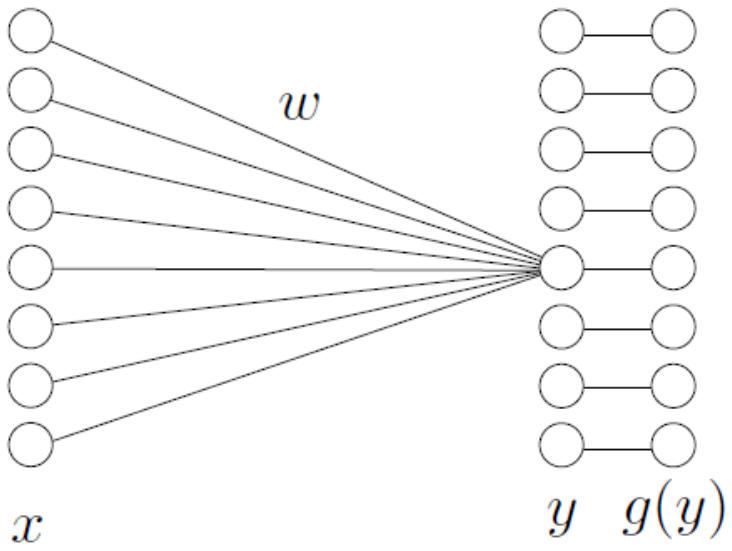


Channels

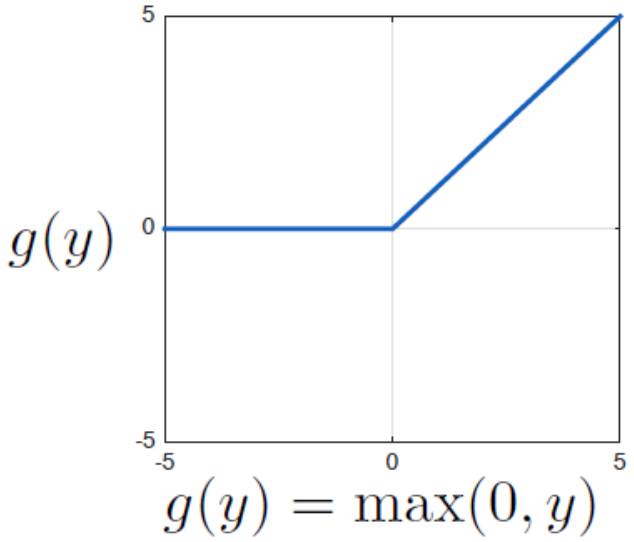

filter


input features
a bank of 2 filters
2-dimensional output features
F1
F2
Σ
Σ
X
Y

Nonlinear Activation Function

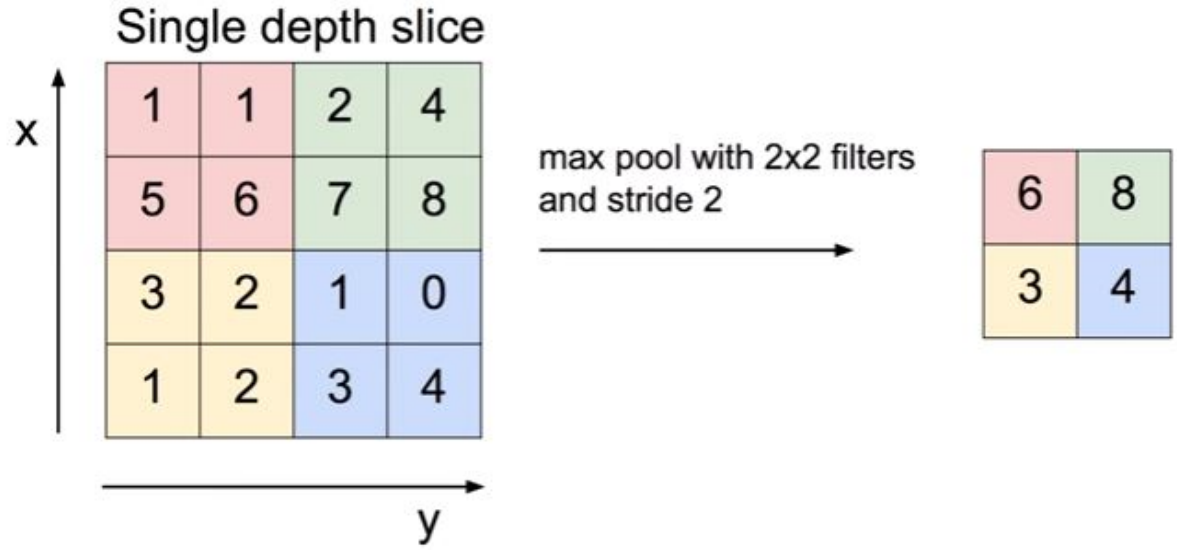


Rectified linear unit (ReLU)

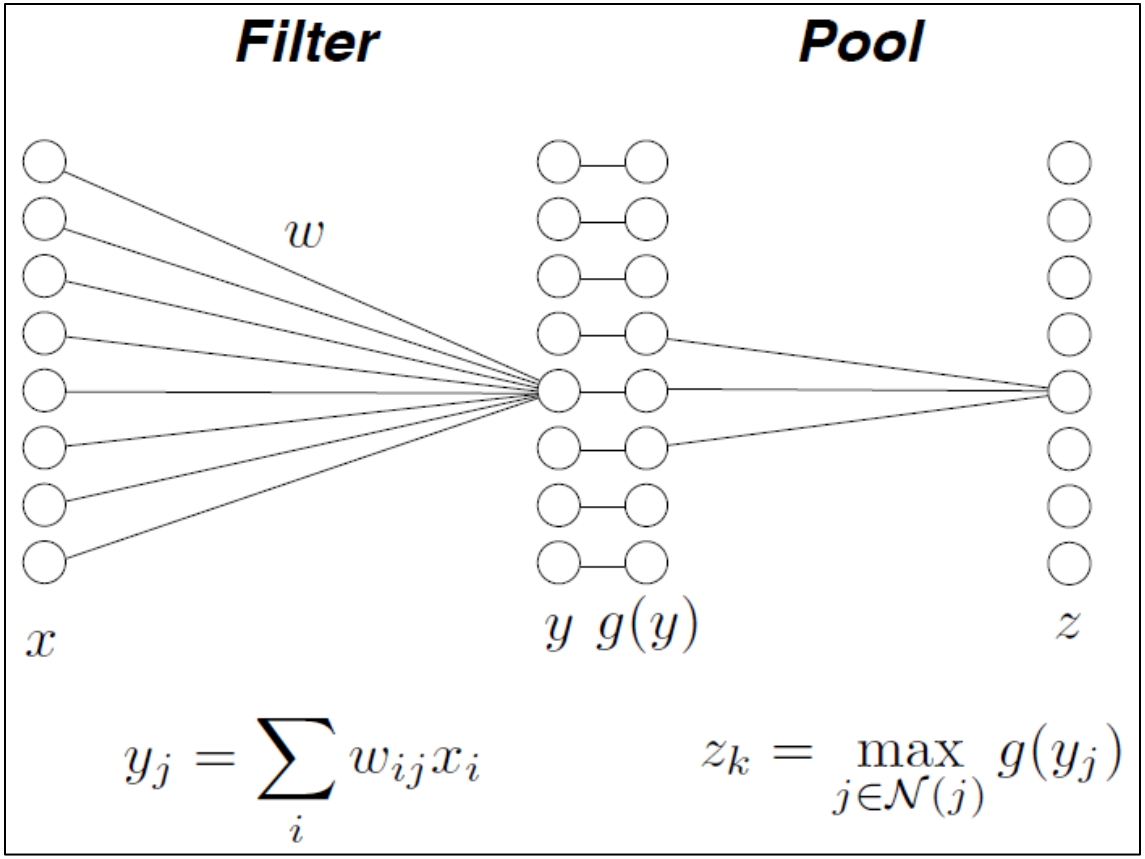


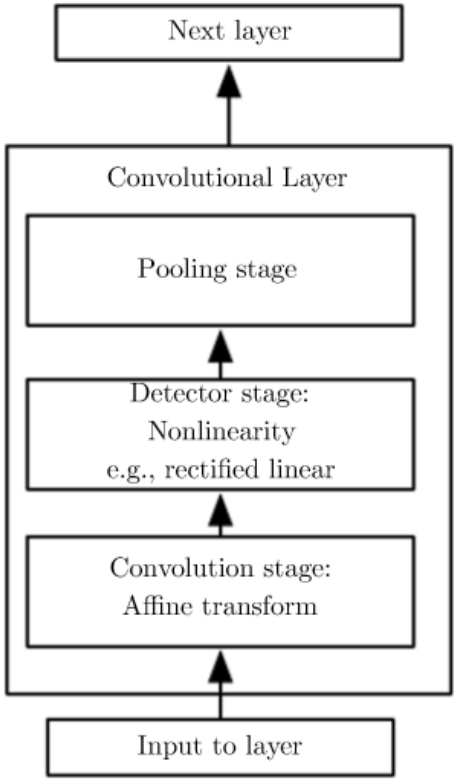
Pooling

- Compute a maximum value in a sliding window (max pooling)
- Reduce spatial resolution for faster computation
- Achieve invariance to local translation
- Max pooling introduces invariances
  - Pooling size : 2×2



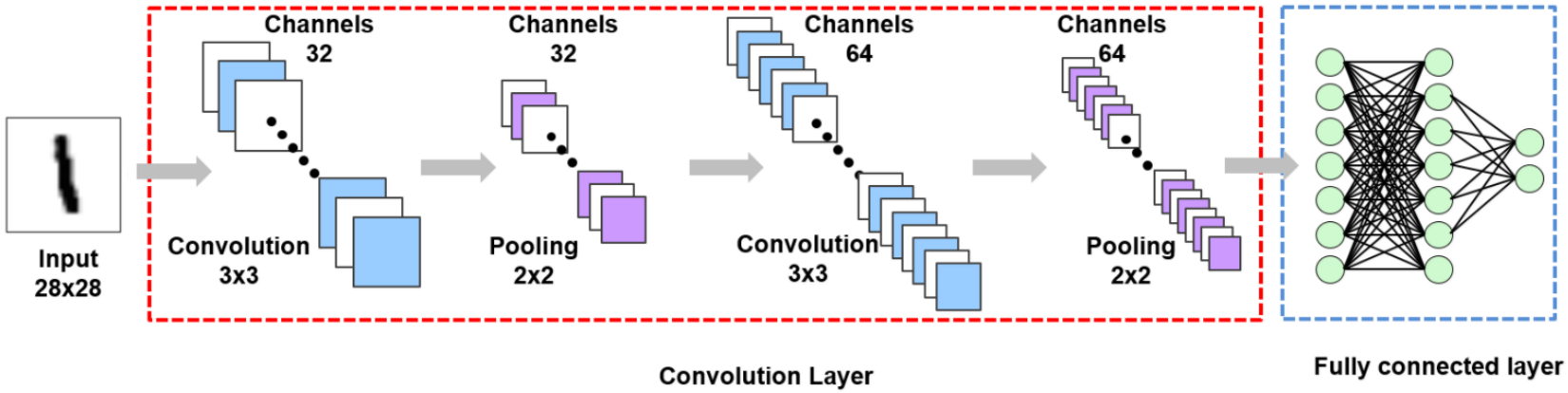
Inside a Convolution Layer





Lab: CNN with TensorFlow

- MNIST example
- To classify handwritten digits



Lab: CNN with TensorFlow

- Import Library

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

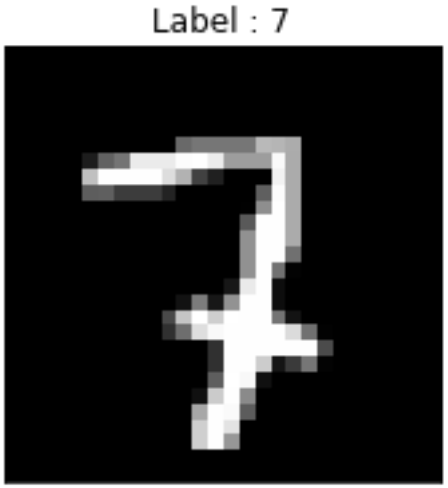
- Load MNIST Data
  - Download MNIST data from the TensorFlow tutorial example

```
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
```

Lab: CNN with TensorFlow

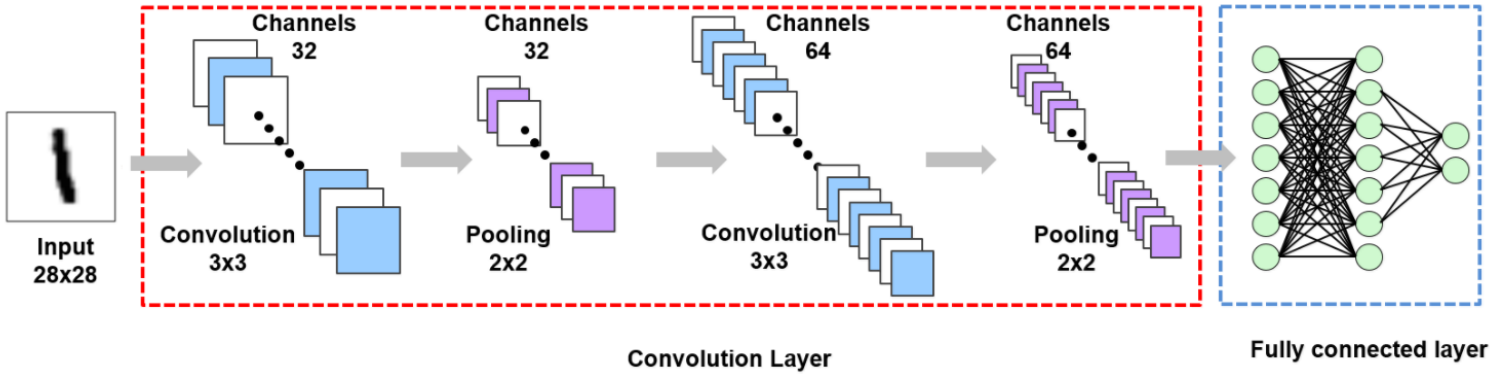
```
# Check data
train_x, train_y = mnist.train.next_batch(10)
img = train_x[9,:].reshape(28, 28)

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



Build a Model

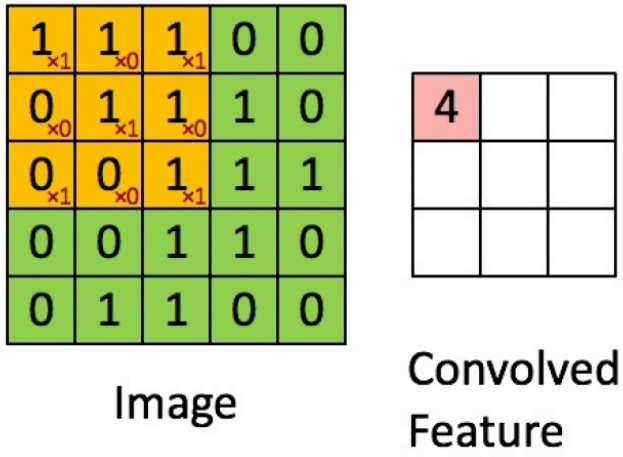
- Convolution layers
  1) The layer performs several convolutions to produce a set of linear activations
  2) Each linear activation is running through a nonlinear activation function
  3) Use pooling to modify the output of the layer further

- Fully connected layers
  – Simple multi-layer perceptron



Convolution

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

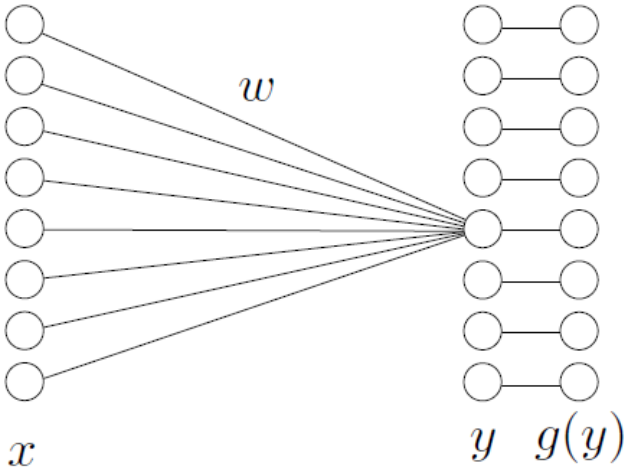


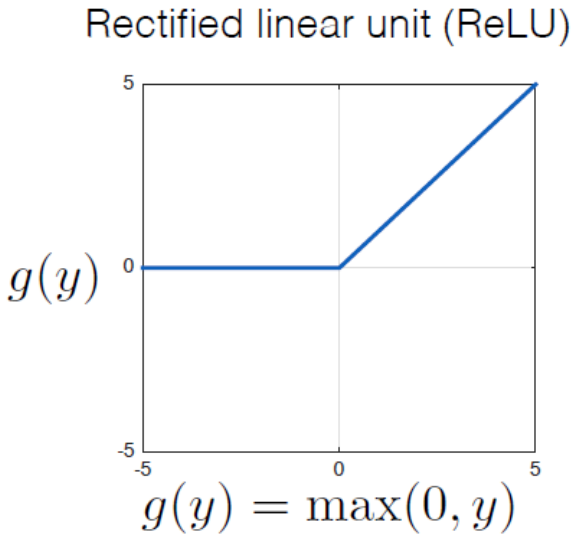
  - Filter size : 3×3
  - Stride : The stride of the sliding window for each dimension of input
  - Padding : Allow us to control the kernel width and the size of the output independently
    - 'SAME' : zero padding
    - 'VALID' : No padding

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

Nonlinear Activation

- Second, each linear activation is running through a nonlinear activation function

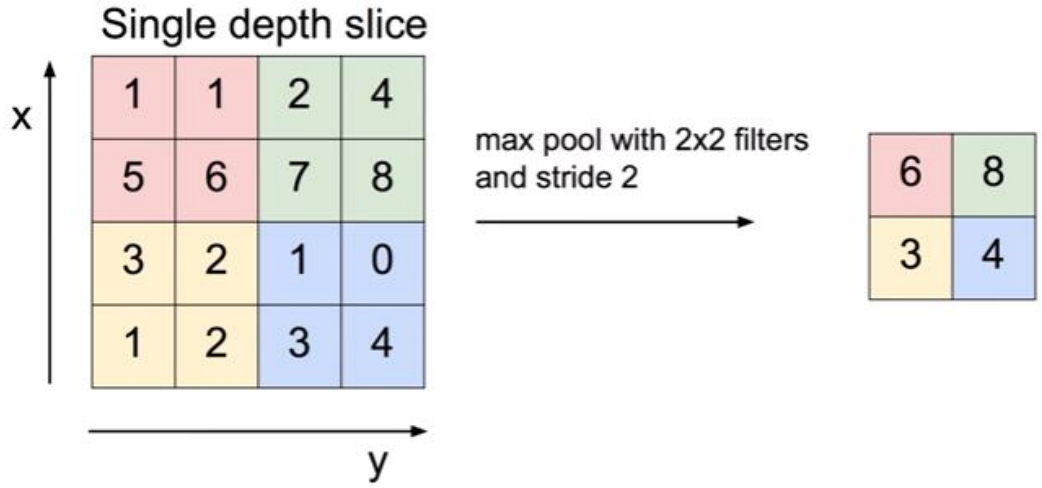





```
conv1 = tf.nn.relu(tf.add(conv1, biases['conv1']))
```

Pooling

- Third, use a pooling to modify the output of the layer further
  - Compute a maximum value in a sliding window (max pooling)

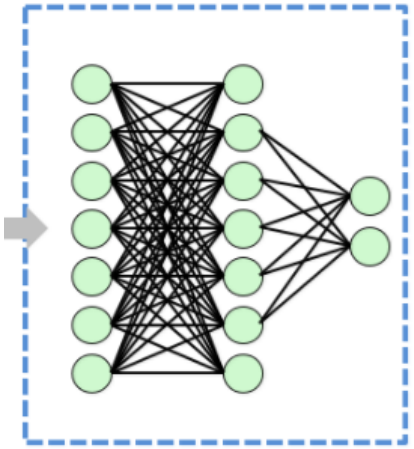


  - Pooling size : 2×2

```
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

- Fully connected layer
  - Input is typically in a form of flattened features
  - Then, apply softmax to multiclass classification problems
  - The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

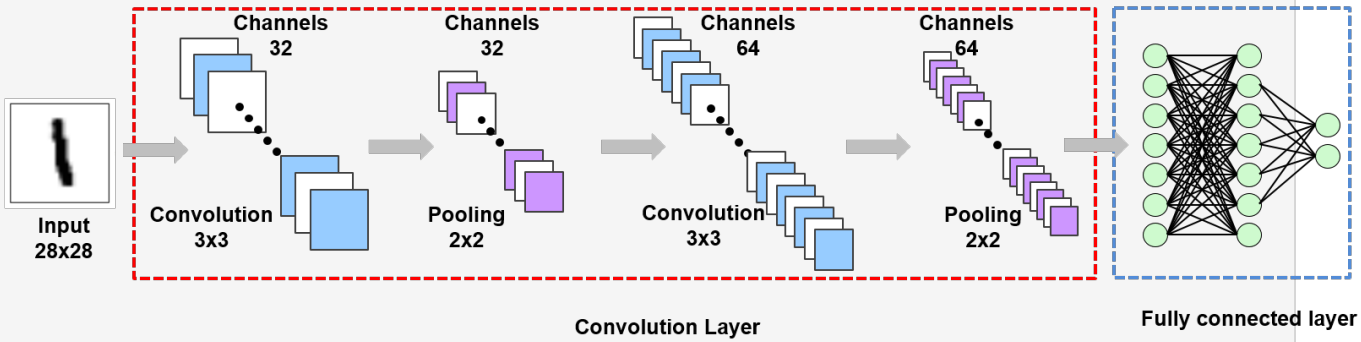


Fully connected layer

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

CNN Structure

```
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
```



Weights, Biases and Placeholder

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

```
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])
```

CNN Network

```
# 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
```

Loss, Initializer and Optimizer

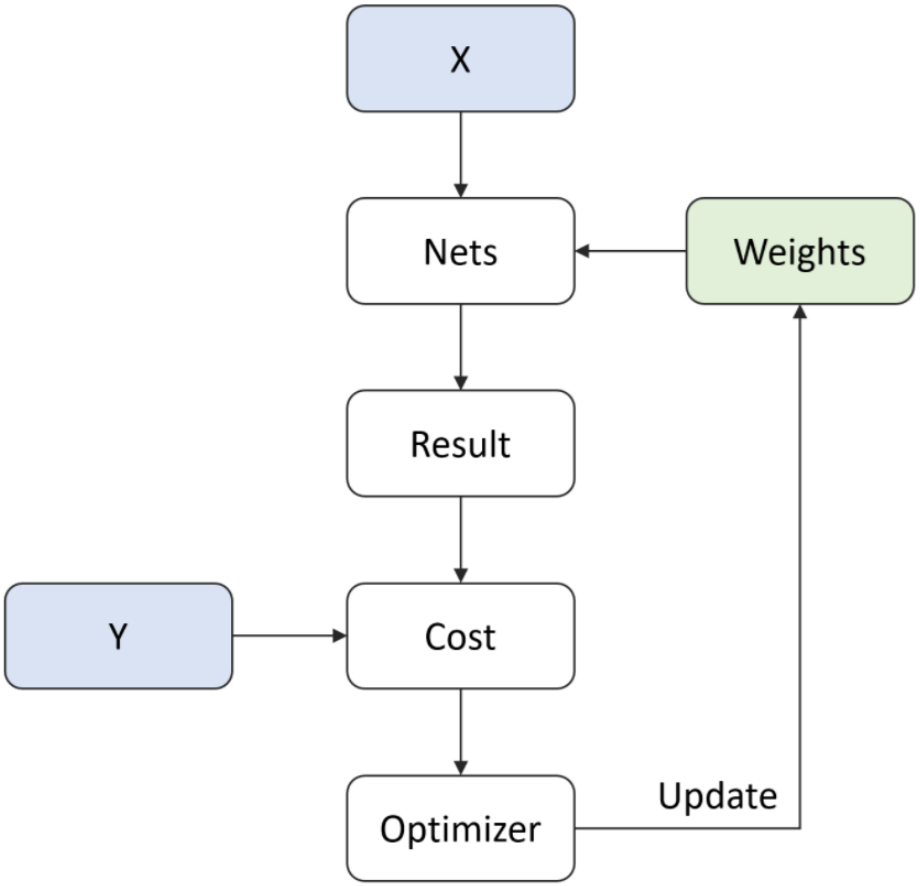
- Loss
  - Classification: Cross entropy
  - Equivalent to apply logistic regression

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

- Initializer
  - Initialize all the empty variables
- Optimizer
  - GradientDescentOptimizer
  - AdamOptimizer: the most popular optimizer

```
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()
```

Summary of Iterative Optimization Flow



Iterative Optimization

- n_batch : batch size for stochastic gradient descent
- n_iter : the number of training steps
- n_prt : check loss for every n_prt iteration

```
n_batch = 50
n_iter = 2500
n_prt = 250
```

Optimization

```
# Run initialize
# config = tf.ConfigProto(allow_soft_placement=True)  # GPU Allocating policy
# sess = tf.Session(config=config)
sess = tf.Session()
sess.run(init)

# Training cycle
for epoch in range(n_iter):
    train_x, train_y = mnist.train.next_batch(n_batch)
    train_x = np.reshape(train_x, [-1, input_h, input_w, input_ch])
    sess.run(optm, feed_dict={x: train_x,  y: train_y})

    if epoch % n_prt == 0:
        c = sess.run(loss, feed_dict={x: train_x, y: train_y})
        print ("Iter : {}".format(epoch))
        print ("Cost : {}".format(c))
```

```
Iter : 0
Cost : 2.483252763748169
Iter : 250
Cost : 0.6449698805809021
Iter : 500
Cost : 0.41933631896972656
Iter : 750
Cost : 0.19014181196689606
Iter : 1000
Cost : 0.15989668667316437
```

Test or Evaluation

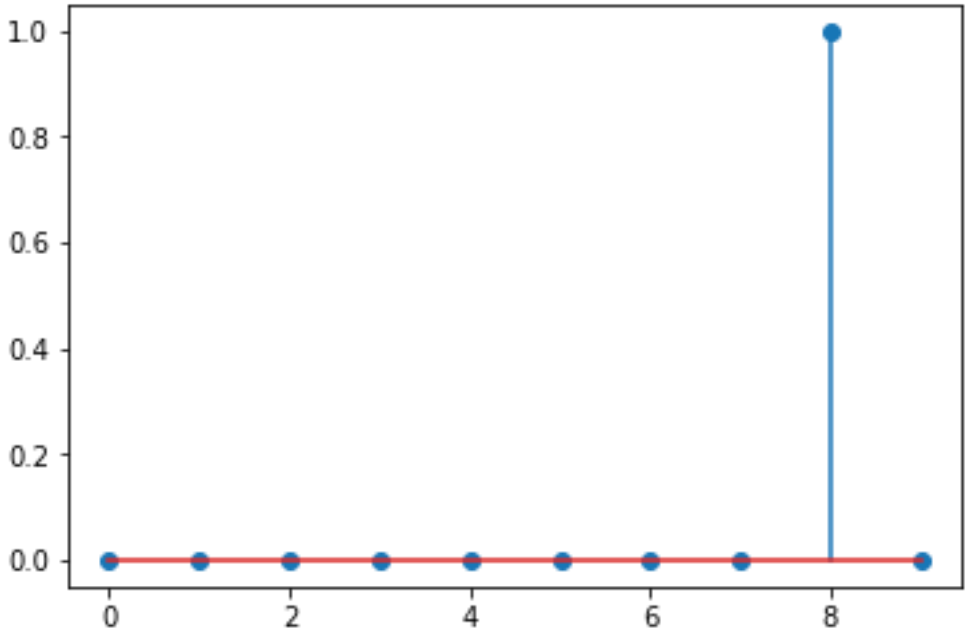
```
test_x, test_y = mnist.test.next_batch(100)

my_pred = sess.run(pred, feed_dict={x : test_x.reshape(-1, 28, 28, 1)})
my_pred = np.argmax(my_pred, axis=1)

labels = np.argmax(test_y, axis=1)

accr = np.mean(np.equal(my_pred, labels))
print("Accuracy : {}%".format(accr*100))
```

```
Accuracy : 99.0%
```

Test or Evaluation

```
test_x, test_y = mnist.test.next_batch(1)
logits = sess.run(tf.nn.softmax(pred), feed_dict={x : test_x.reshape(-1, 28, 28, 1)})
predict = np.argmax(logits)

plt.imshow(test_x.reshape(28, 28), 'gray')
plt.xticks([])
plt.yticks([])
plt.show()

print('Prediction : {}'.format(predict))

plt.stem(logits.ravel())
plt.show()

np.set_printoptions(precision=2, suppress=True)
print('Probability : {}'.format(logits.ravel()))
```

Test or Evaluation




```
Probability : [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

Deep Learning of Things

- CNN implemented in an Embedded System