Classifying Images with Deep Convolutional Neural Networks

Hsi-Pin Ma 馬席彬

http://lms.nthu.edu.tw/course/40724

Department of Electrical Engineering
National Tsing Hua University

Outline

- Building Blocks of Convolutional Neural Networks
- Implementing Deep Convolutional Neural Networks in TensorFlow

Features in ML Algorithms

- **Salient (relevant) features** is key to performance of ML algorithms
  - Traditional ML rely on features from ***domain experts*** or ***computational feature extraction techniques***
  - Neural networks (NNs) can ***learn*** the features ***automatically*** from raw data that most useful for a particular task
    - Consider NN as a feature extraction engine, the early layers extract ***low-level features***
- Feature Hierarchy
  - Multilayer NN construct a so-called ***feature hierarchy*** by combining the low-level features in a layer-wise fashion to form high-level features

Convolutional Neural Networks

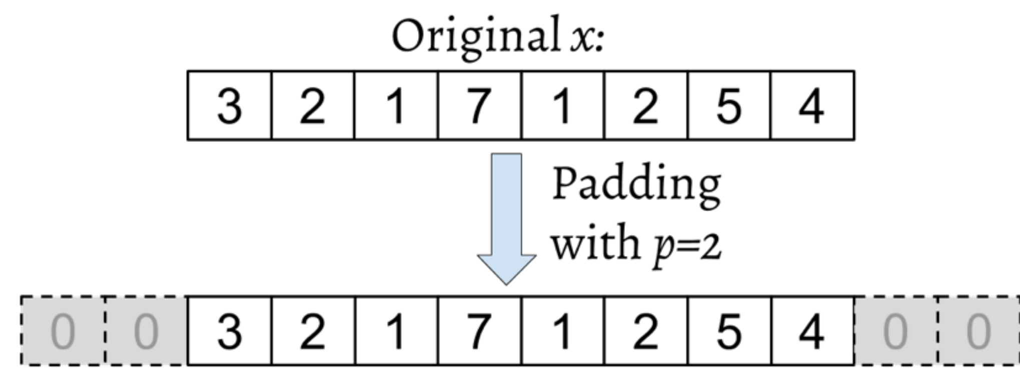
- CNN computes ***feature maps*** from an input image
- CNNs perform very well for image-related tasks
  - Sparse-connectivity: A single element in the feature map is connected to only a small patch of pixels
  - Parameter-sharing: The same weights are used for different patches of the input image
- Components
  - Convolutional layers (conv)
  - Pooling layers (P)
  - Full connected layers (FC)

Building Blocks of Convolutional Neural Networks

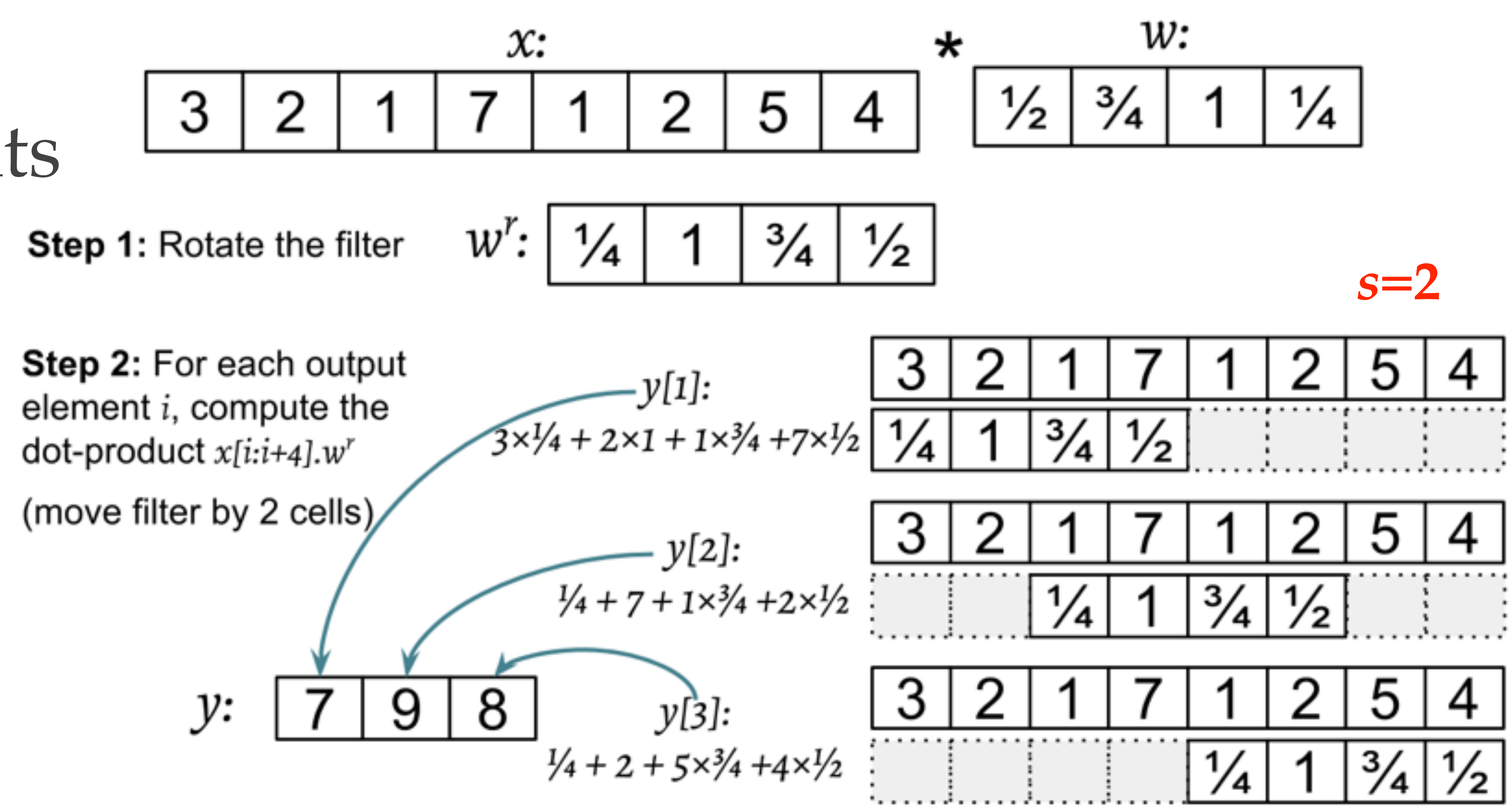
Performing Discrete Convolutions

- A discrete convolution in one dimension
  - *x, w*: one-dimensional vector, *x*: input/signal, *w*: filter/kernel

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{w} \rightarrow \boldsymbol{y}[i] = \sum_{k=-\infty}^{+\infty} \boldsymbol{x}[i-k]\boldsymbol{w}[k]$$



- Padding (zero-padding): p
- Example
  - *x, w*: n, m elements
  - *s*: stride, shift

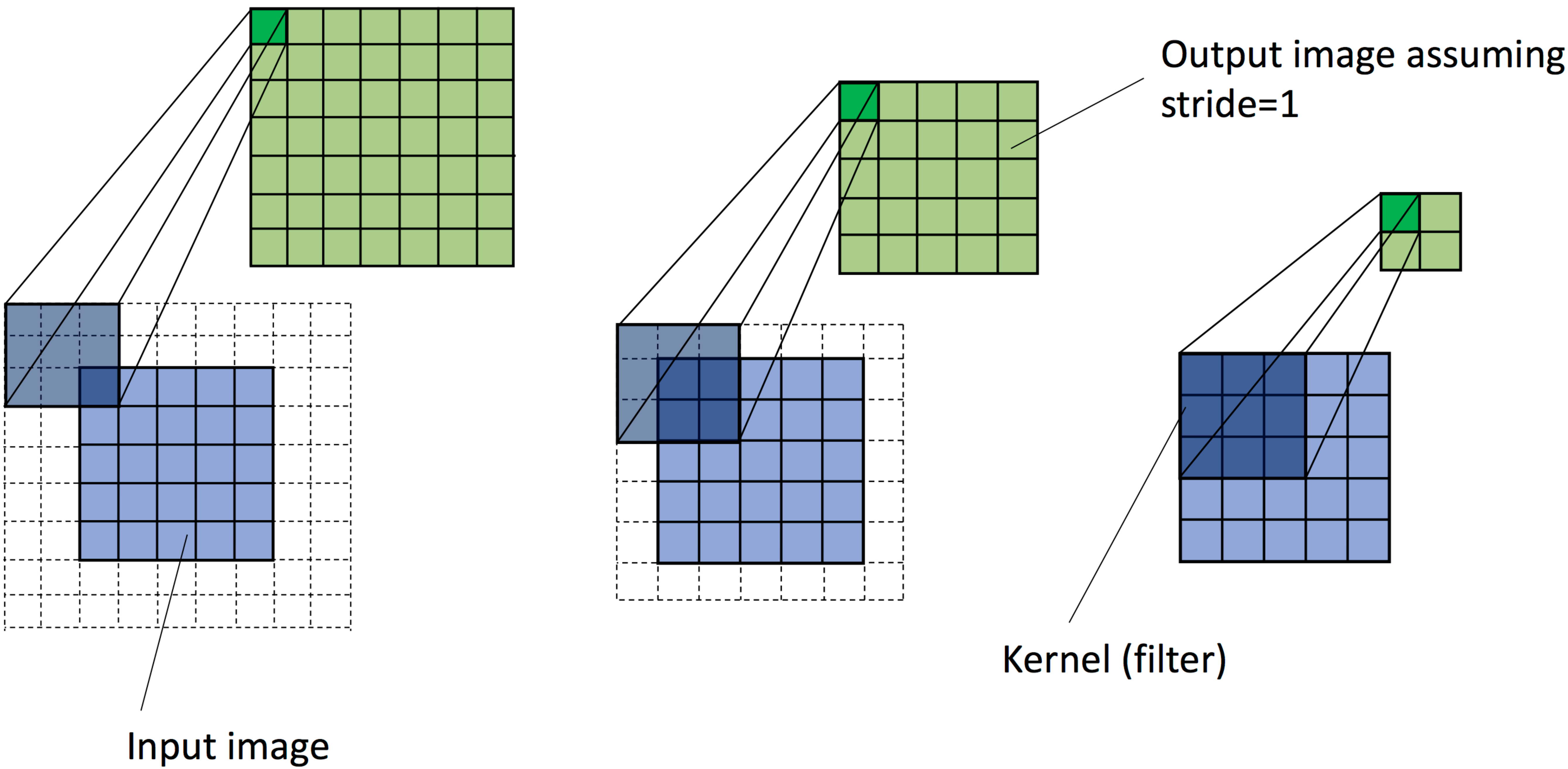


Effect of Zero-Padding in a Convolution

- Three commonly modes
  - Full padding: $p=m-1$. Increase the dimension of the output. Most used in signal processing applications to minimize the boundary effect
  - Same padding: Same size of input and output vectors. Mostly used in CNNs to make a network architecture design more convenient.
  - Valid padding
- In practice, preserve the spatial size using same padding for the convolutional layers and decrease the spatial size via pooling layers

Effect of Zero-Padding in a Convolution

Full padding **Same padding** **Valid padding**



Determining the Size of the Convolutional Output

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1$$

```python
import numpy as np


def conv1d(x, w, p=0, s=1):
    w_rot = np.array(w[::-1])
    x_padded = np.array(x)
    if p > 0:
        zero_pad = np.zeros(shape=p)
        x_padded = np.concatenate([zero_pad, x_padded, zero_pad])
    res = []
    for i in range(0, int(len(x)/s),s):
        res.append(np.sum(x_padded[i:i+w_rot.shape[0]] * w_rot))
    return np.array(res)

## Testing:
x = [1, 3, 2, 4, 5, 6, 1, 3]
w = [1, 0, 3, 1, 2]
print('Conv1d Implementation: ',
      conv1d(x, w, p=2, s=1))
print('Numpy Results:         ',
      np.convolve(x, w, mode='same'))
```

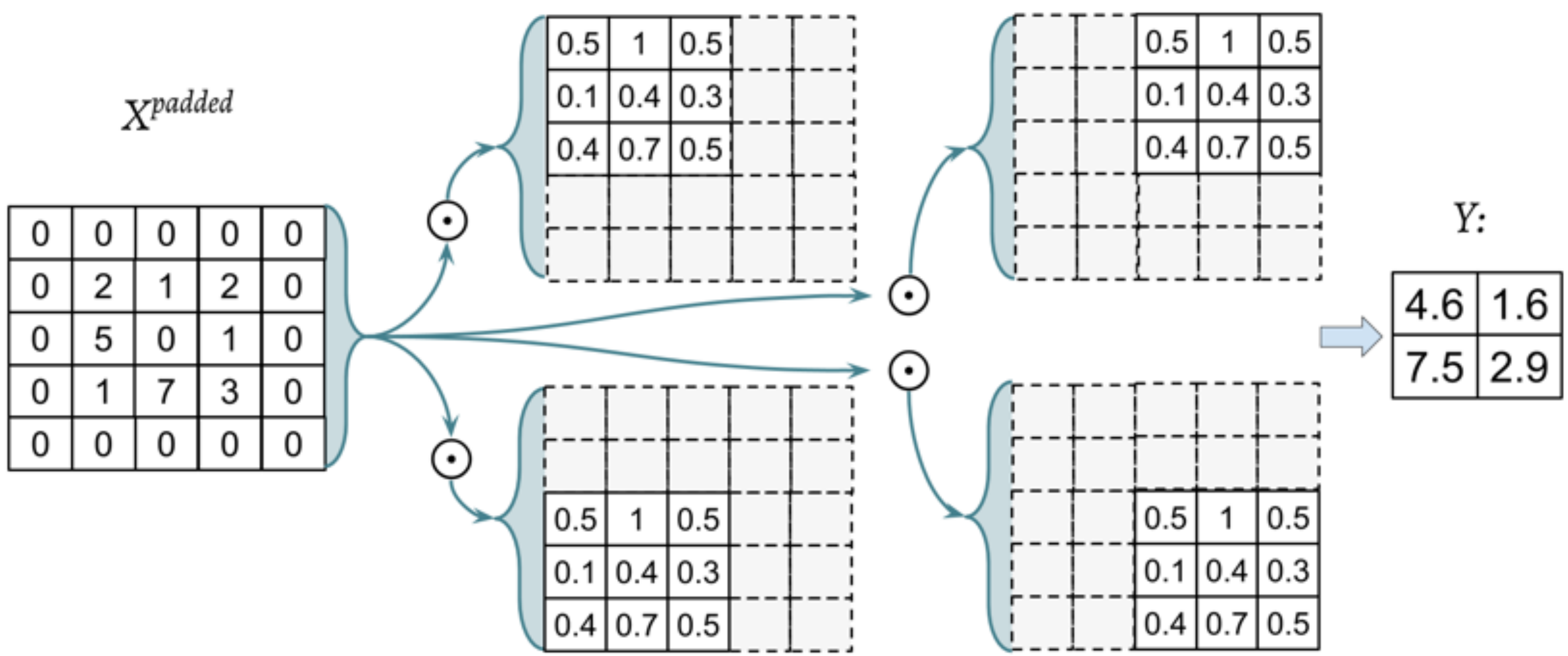
```
Conv1d Implementation:  [  5.  14.  16.  26.  24.  34.  19.  22.]
Numpy Results:          [ 5 14 16 26 24 34 19 22]
```

Performing a Discrete Convolution in 2D

$$\boldsymbol{Y} = \boldsymbol{X} * \boldsymbol{W} \rightarrow \boldsymbol{Y}[i,j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} \boldsymbol{X}[i-k_1, j-k_2] \boldsymbol{W}[k_1, k_2]$$

$$\underset{X}{\begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 2 & 1 & 2 & 0 \\ \hline 0 & 5 & 0 & 1 & 0 \\ \hline 0 & 1 & 7 & 3 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}} \; * \; \underset{W}{\begin{array}{|c|c|c|} \hline 0.5 & 0.7 & 0.4 \\ \hline 0.3 & 0.4 & 0.1 \\ \hline 0.5 & 1 & 0.5 \\ \hline \end{array}}$$

$$\boldsymbol{W}^r = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.1 & 0.4 & 0.3 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$



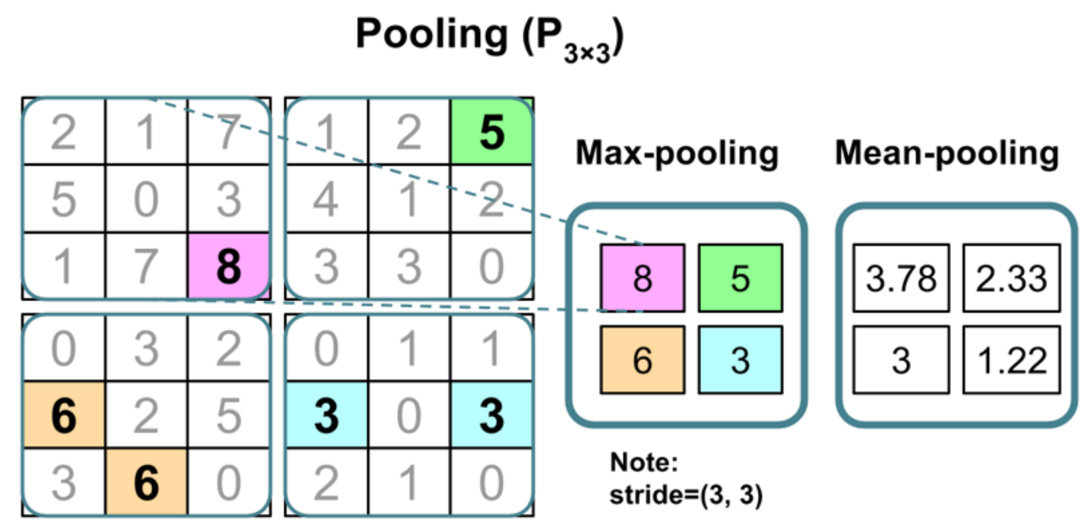
2D Convolution

```
import numpy as np
import scipy.signal


def conv2d(X, W, p=(0,0), s=(1,1)):
    W_rot = np.array(W)[::-1,::-1]
    X_orig = np.array(X)
    n1 = X_orig.shape[0] + 2*p[0]
    n2 = X_orig.shape[1] + 2*p[1]
    X_padded = np.zeros(shape=(n1,n2))
    X_padded[p[0]:p[0] + X_orig.shape[0],
             p[1]:p[1] + X_orig.shape[1]] = X_orig

    res = []
    for i in range(0, int((X_padded.shape[0] -
                          W_rot.shape[0])/s[0])+1, s[0]):
        res.append([])
        for j in range(0, int((X_padded.shape[1] -
                              W_rot.shape[1])/s[1])+1, s[1]):
            X_sub = X_padded[i:i+W_rot.shape[0], j:j+W_rot.shape[1]]
            res[-1].append(np.sum(X_sub * W_rot))
    return(np.array(res))

X = [[1, 3, 2, 4], [5, 6, 1, 3], [1 , 2,0, 2], [3, 4, 3, 2]]
W = [[1, 0, 3], [1, 2, 1], [0, 1, 1]]
print('Conv2d Implementation: \n',
      conv2d(X, W, p=(1,1), s=(1,1)))

print('Scipy Results:         \n',
      scipy.signal.convolve2d(X, W, mode='same'))
```

```
Conv2d Implementation:
 [[ 11.  25.  32.  13.]
 [ 19.  25.  24.  13.]
 [ 13.  28.  25.  17.]
 [ 11.  17.  14.   9.]]
Scipy Results:
 [[11 25 32 13]
 [19 25 24 13]
 [13 28 25 17]
 [11 17 14  9]]
```

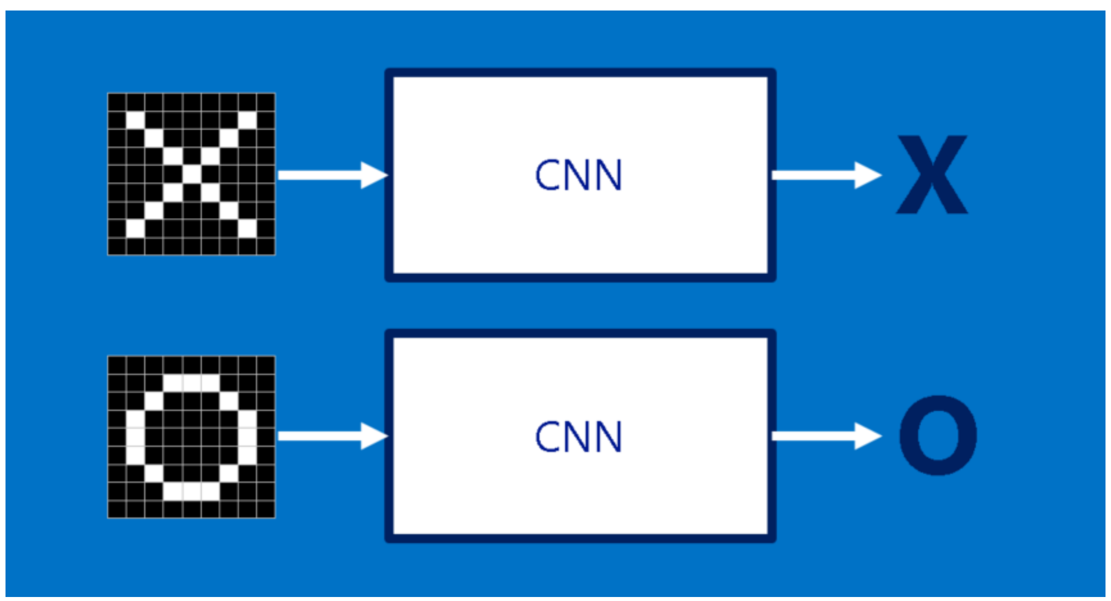
Subsampling

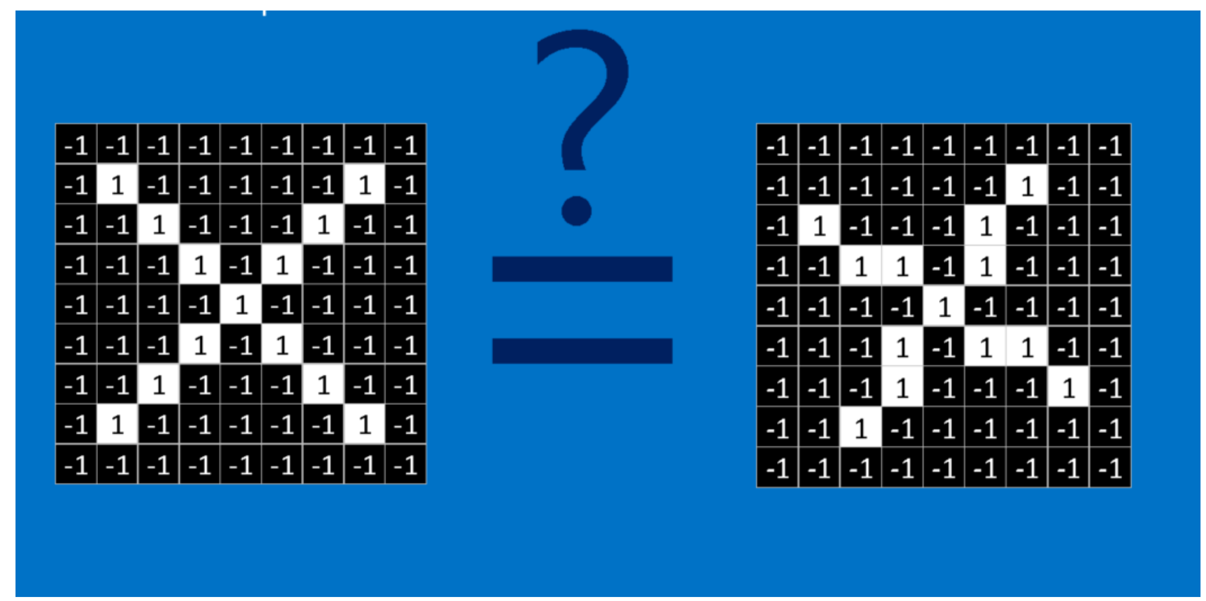
- Two forms of subsampling
  - max-pooling
  - mean-pooling (average-pooling)
- Advantages of pooling
  - Introducing some sort of local invariance to generate features that are more robust to noise in the input data
  - Decrease the size of features and result in high computation efficiency and also can reduce the degree of overfitting



A Simple Walk Through

- To classify X’s and O’s from images

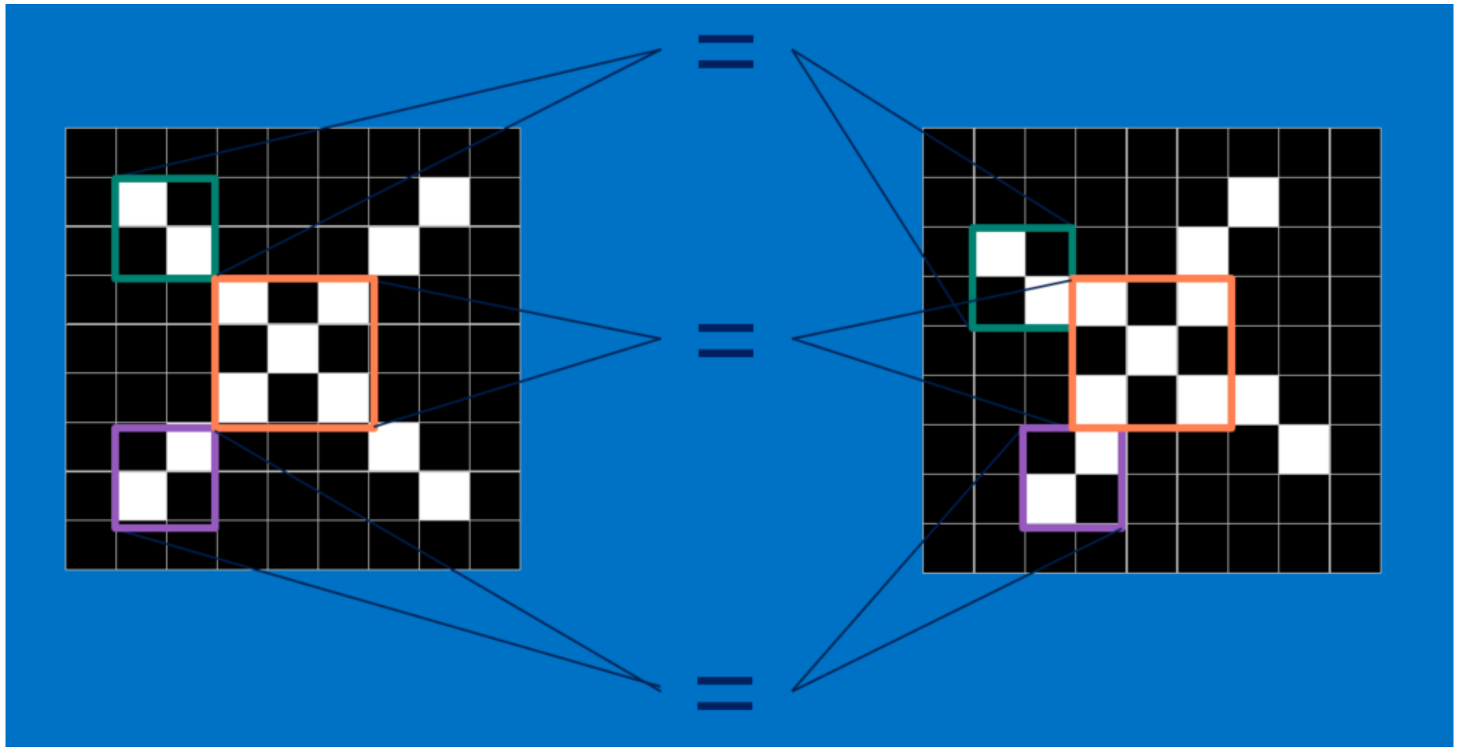


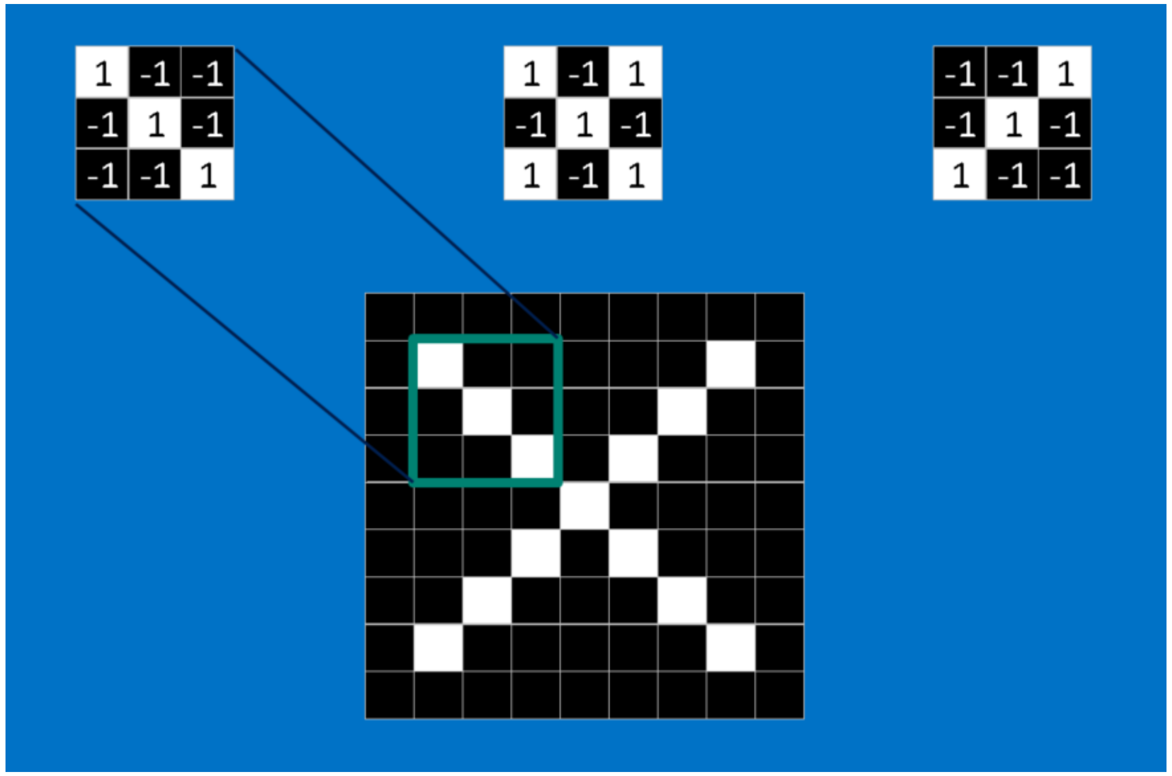


[Brandon, https://brohrer.github.io/how_convolutional_neural_networks_work.html]

Features

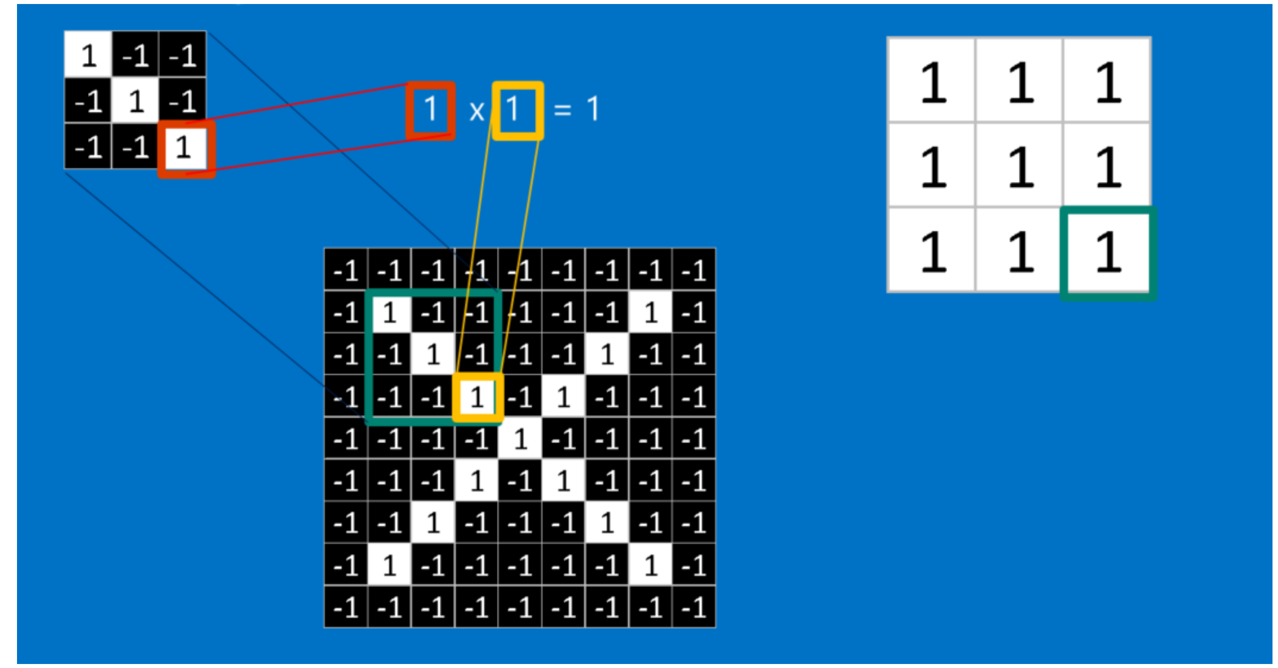
- CNN compare images piece by piece (features)

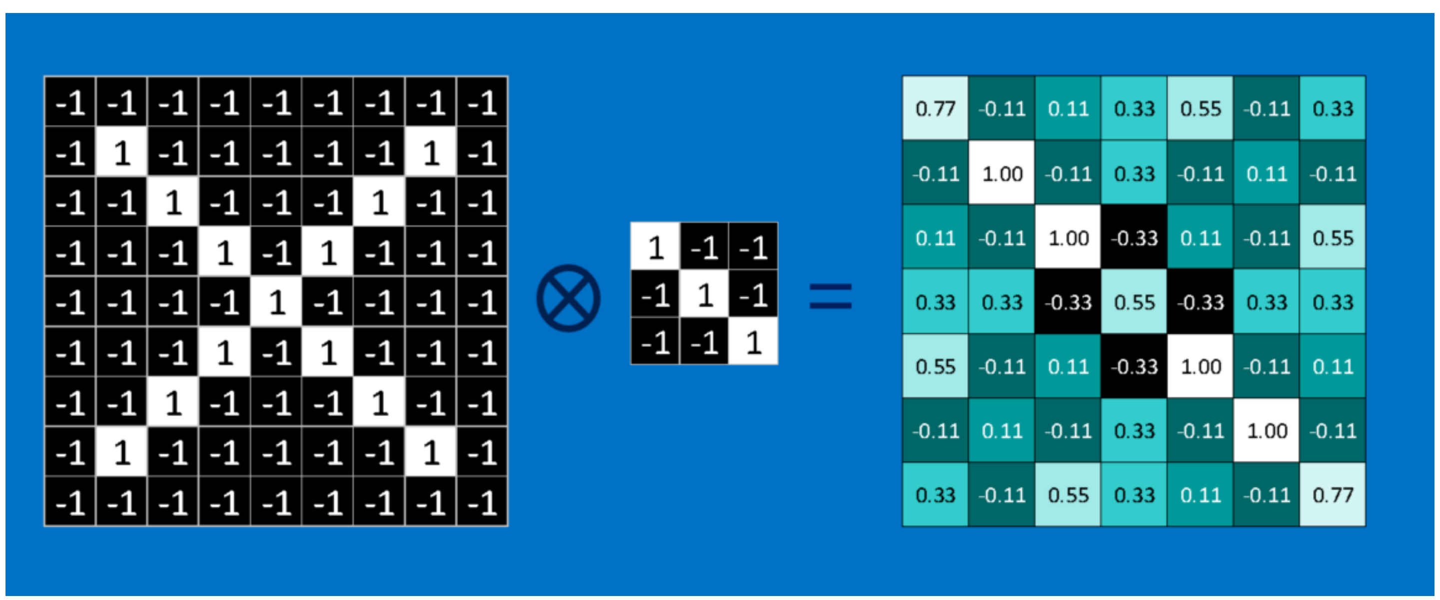


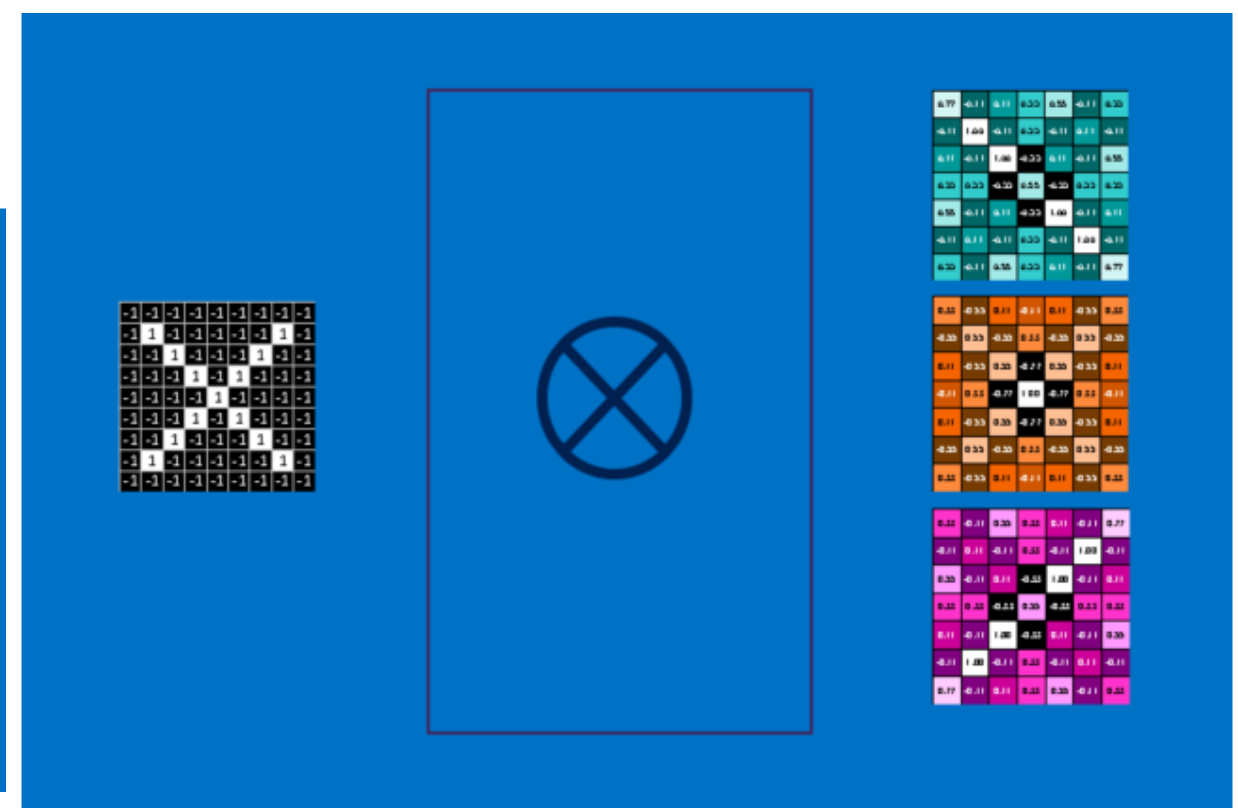


Convolution

- Define white 1, black -1
  - For the convolution results
    - close to 1: strong matches
    - close to -1: strong matches to the photographic negative of the feature
    - near 0: no match



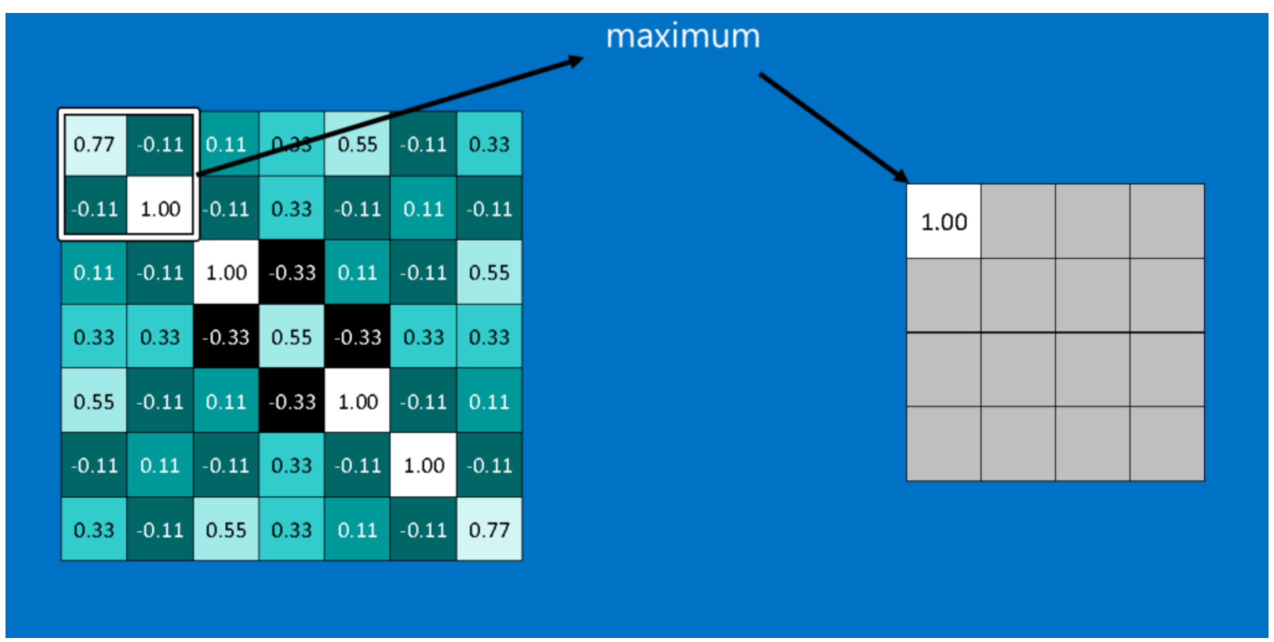


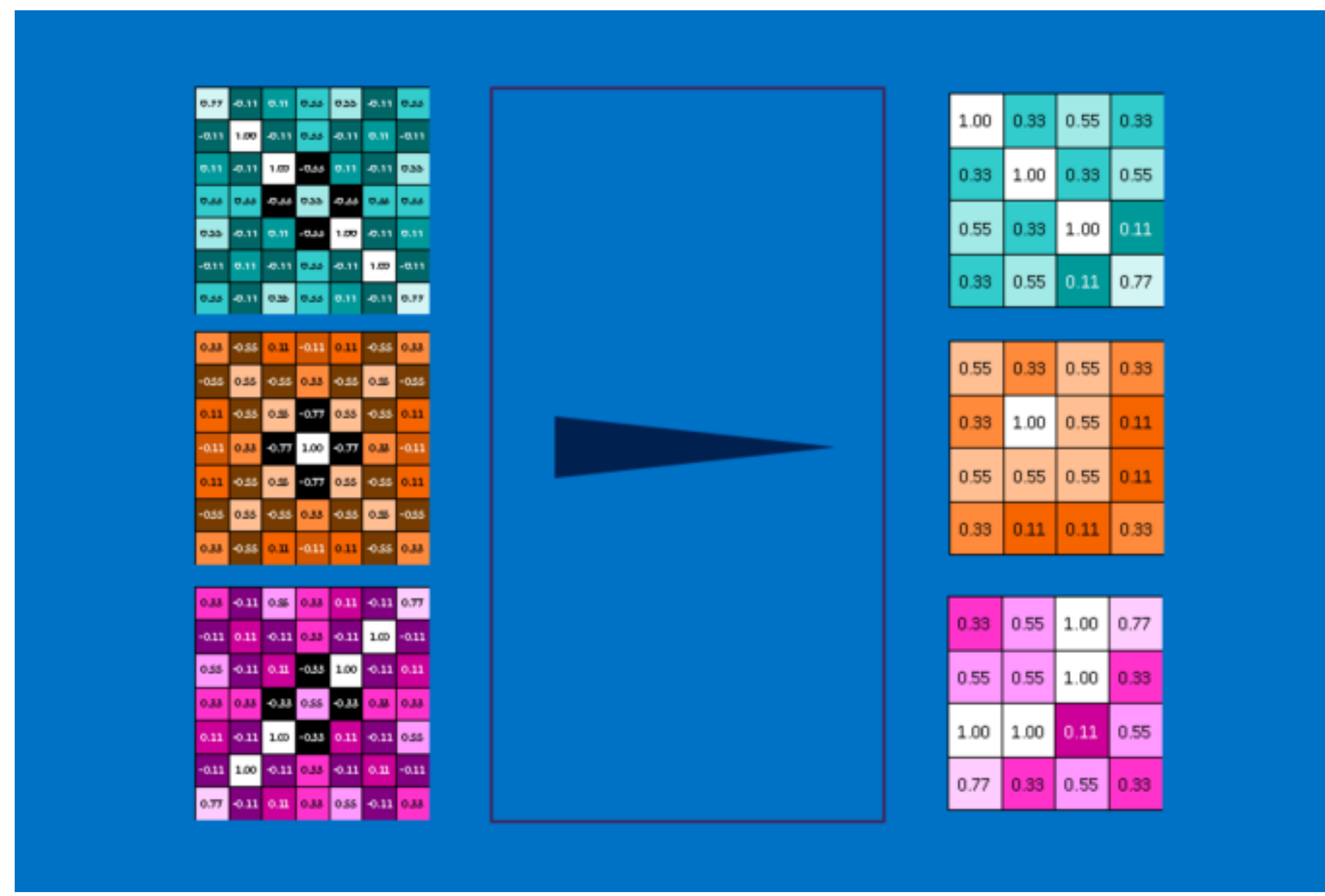


A map of where in the image the feature is found

Pooling

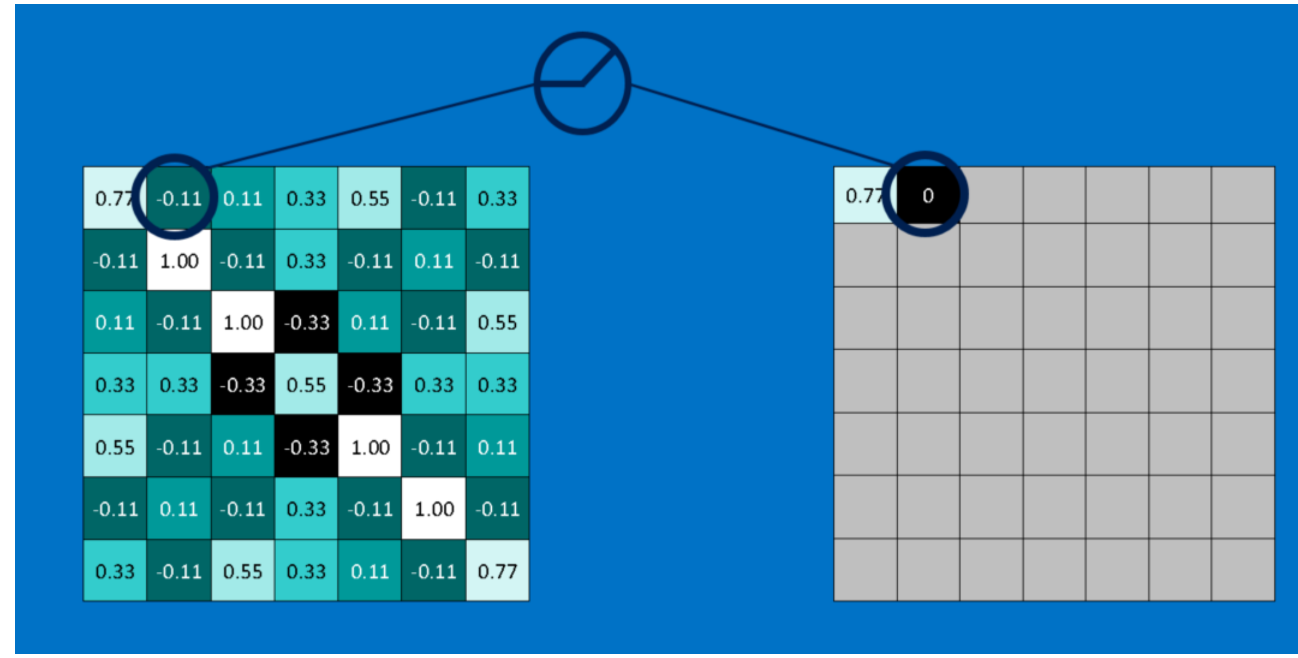
- Take large images to shrink them down while preserving the most important information in them

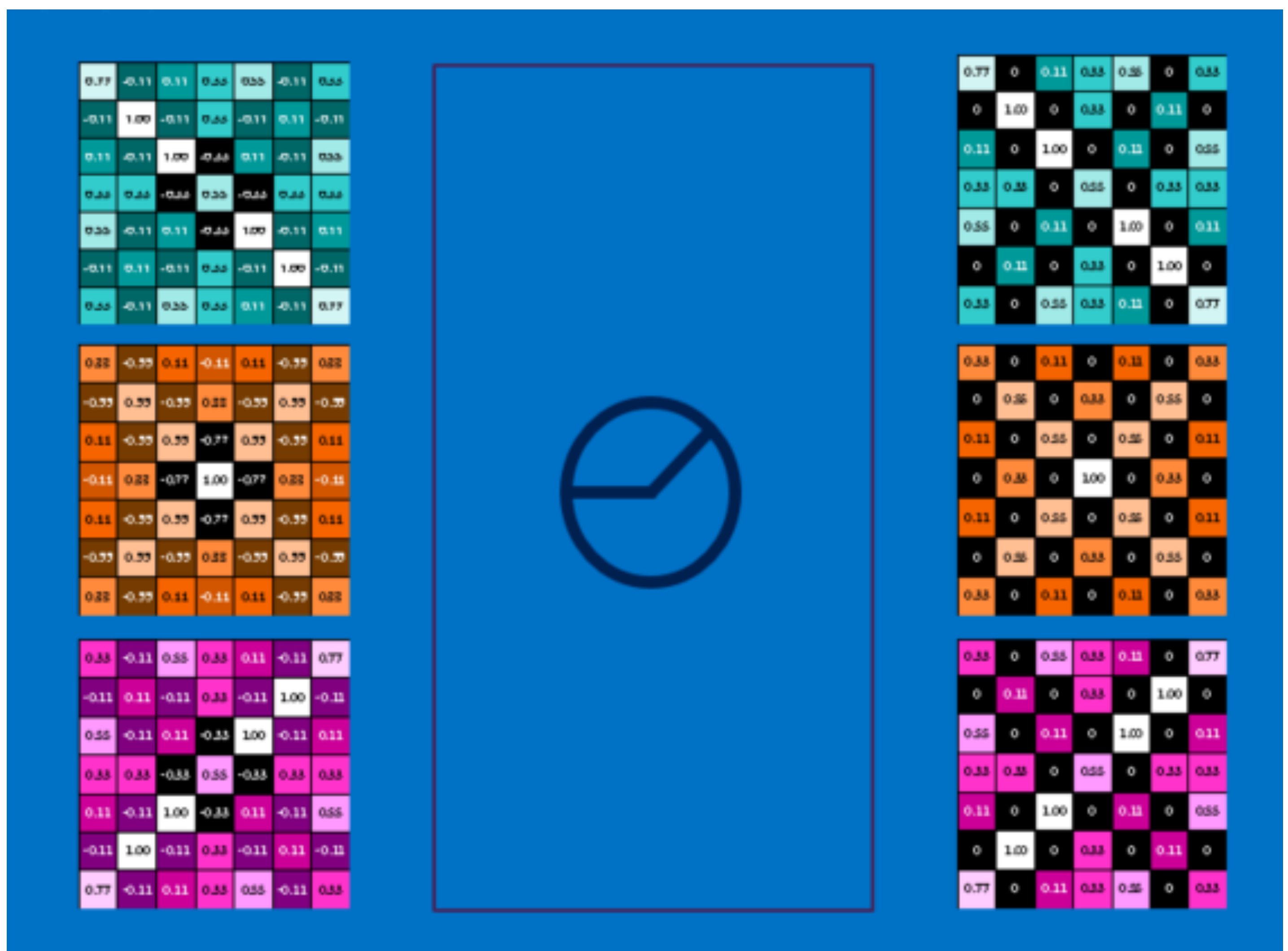




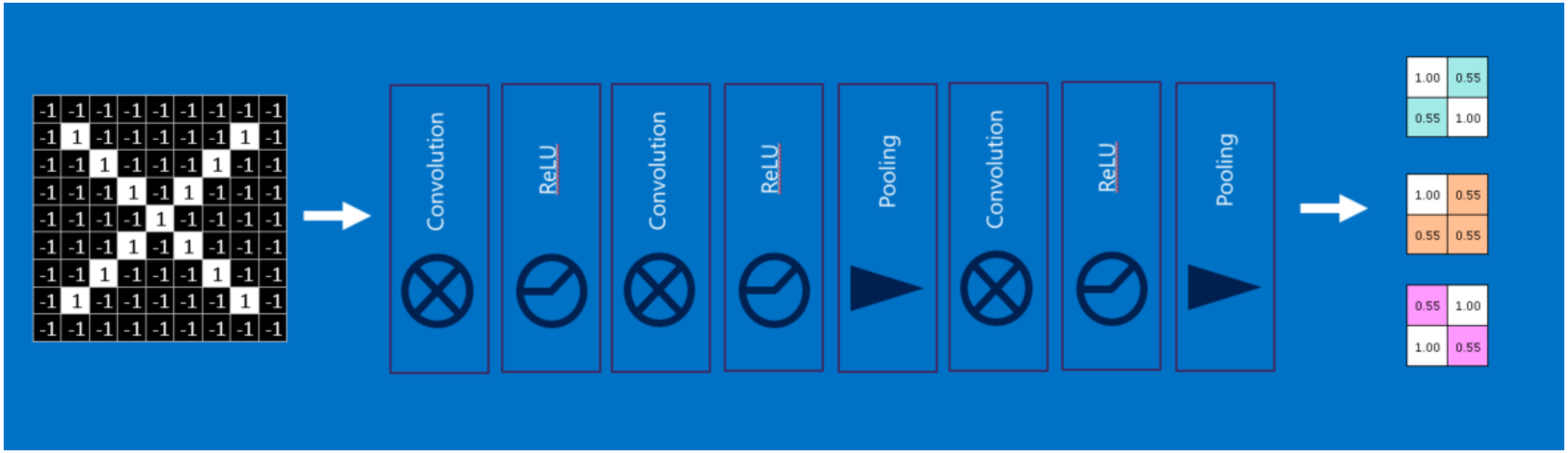
Rectified Linear Units (ReLU)

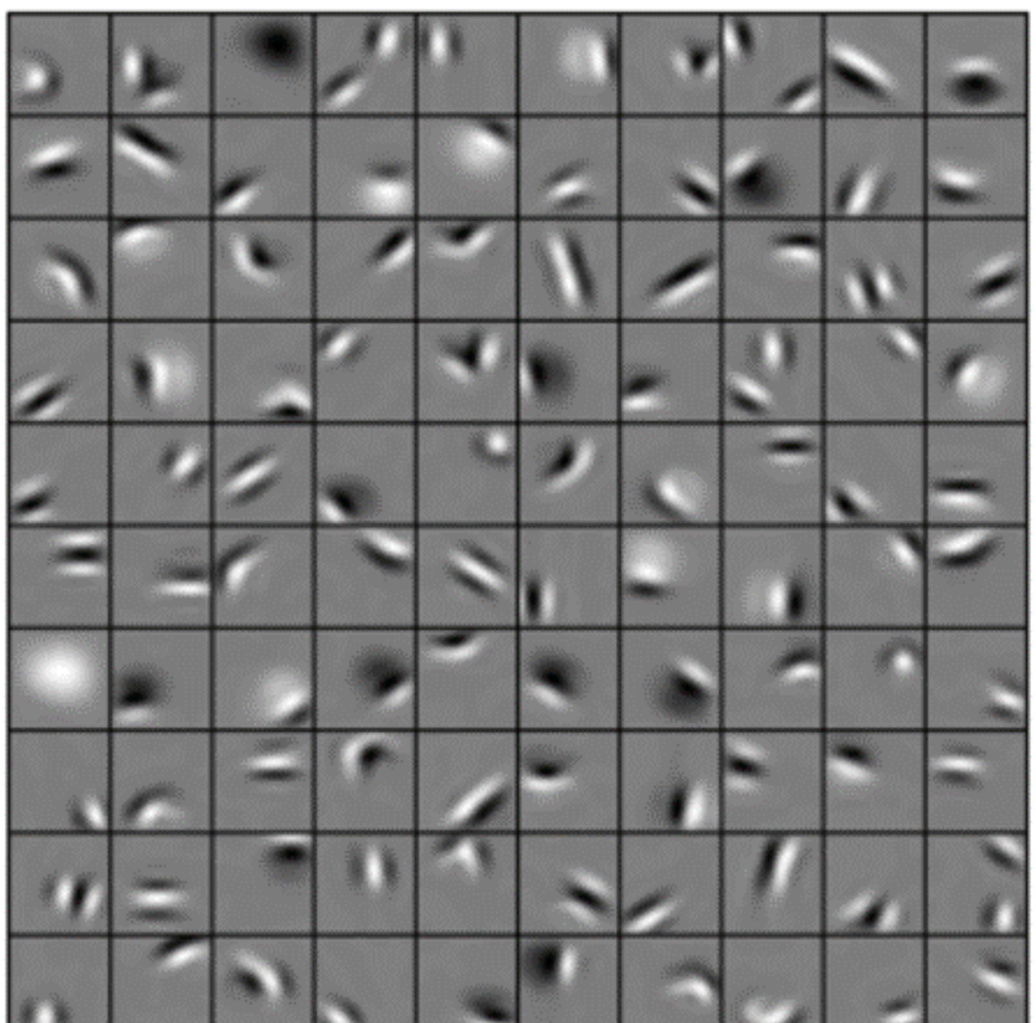
- Whenever a negative number occurs, swap it out for a 0.

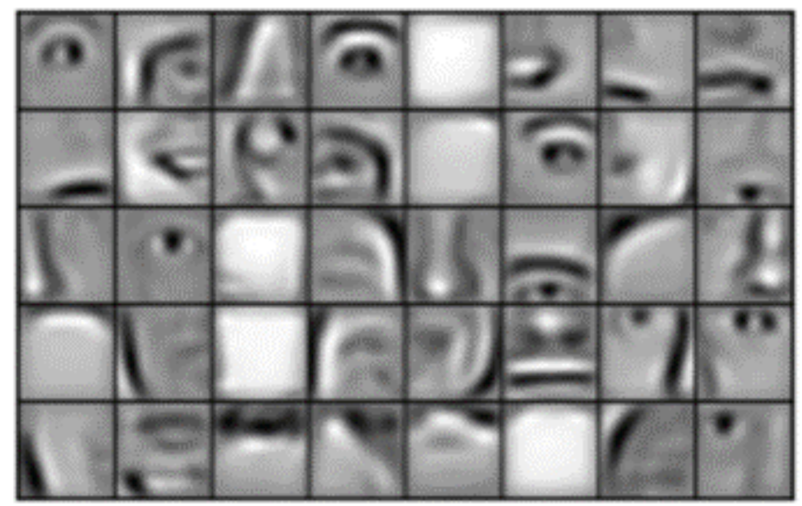


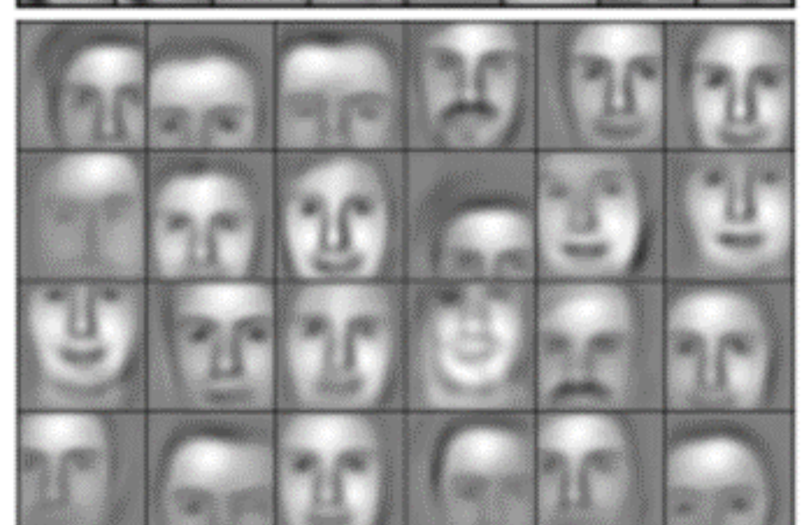


Deep Learning

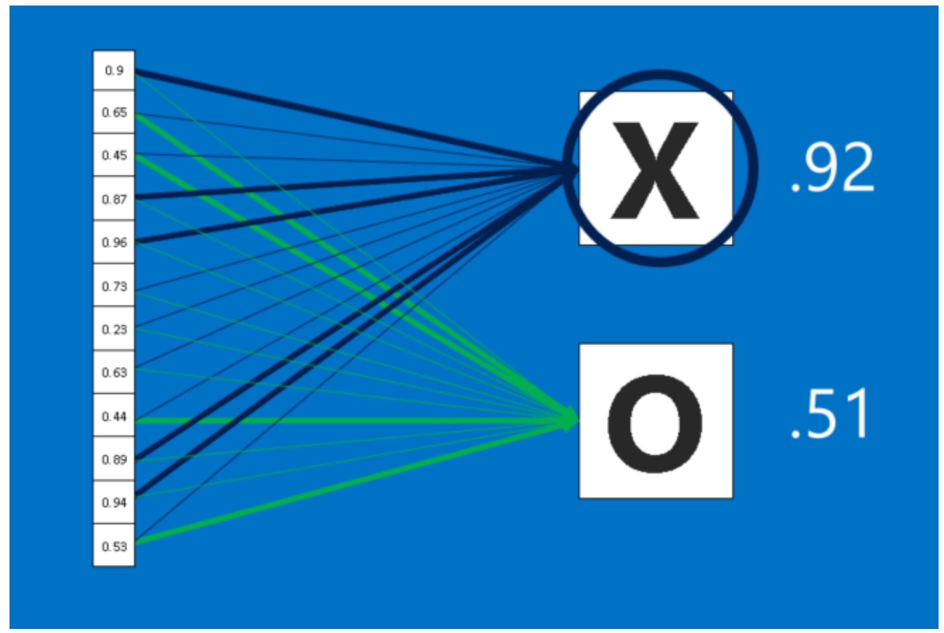


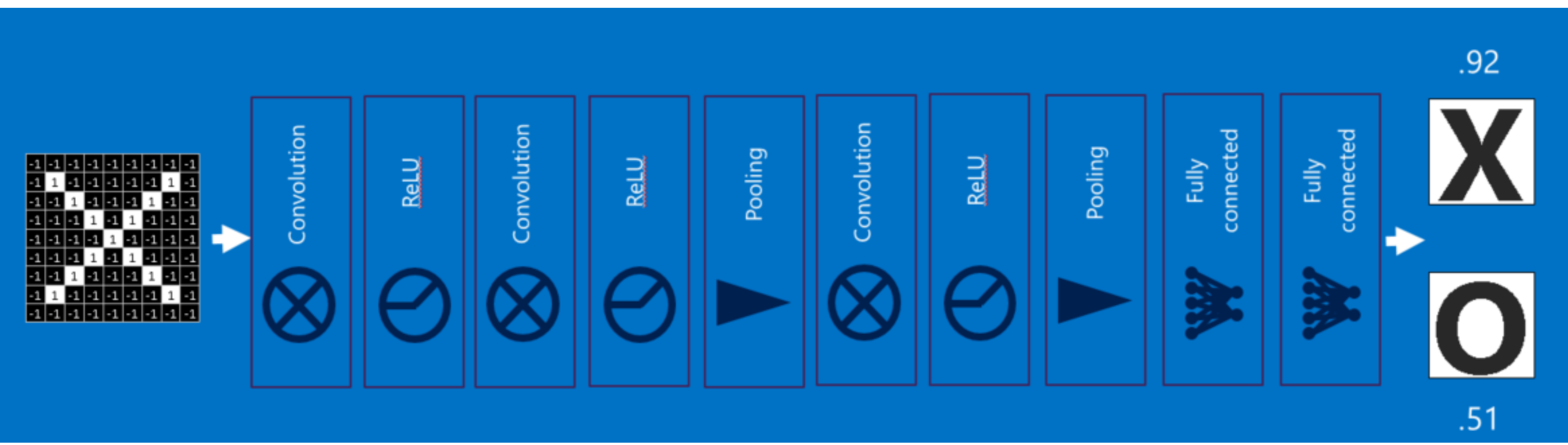






Fully Connected Layers





Working with Multiple Input or Color Channels

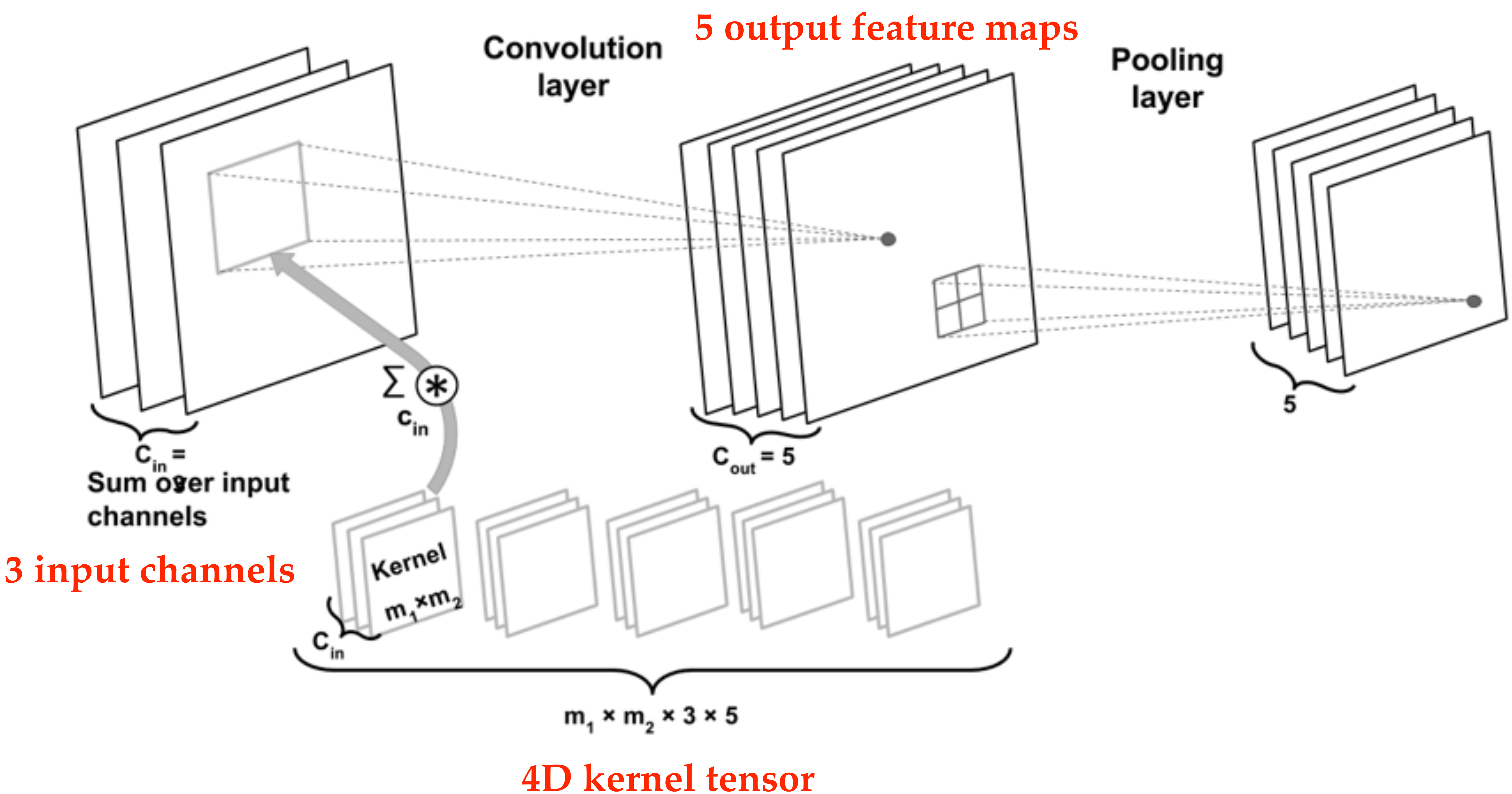
- Using multiple channels as input to a convolutional layer requires to use a rank-3 tensor or a 3D array $\boldsymbol{X}_{N_1 \times N_2 \times C_{in}}$

$$\begin{array}{l}\text{Given a sample } \mathbf{X}_{\mathrm{n}_1 \times \mathrm{n}_2 \times c_{\mathrm{in}}}{}', \\ \text{a kernel matrix } \mathbf{W}_{\mathrm{m}_1 \times \mathrm{m}_2 \times c_{\mathrm{in}}}{}', \\ \text{and bias value } b\end{array} \Rightarrow \begin{cases} \boldsymbol{Y}^{Conv} = \sum_{c=1}^{C_{in}} \boldsymbol{W}[:,:,c] * \boldsymbol{X}[:,:,c] \\ \text{pre-activation:} \quad \mathbf{A} = \mathbf{Y}^{\mathrm{Conv}} + \mathrm{b} \\ \text{Feature map:} \quad \boldsymbol{H} = \phi(\boldsymbol{A}) \end{cases}$$

Consider the number of output feature maps

$$\begin{array}{l}\text{Given a sample } \boldsymbol{X}_{n_1 \times n_2 \times C_{in}} \\ \text{kernel matrix } \boldsymbol{W}_{m_1 \times m_2 \times C_{in} \times C_{out}} \\ \text{and bias vector } \boldsymbol{b}_{C_{out}}\end{array} \Rightarrow \begin{cases} \boldsymbol{Y}^{Conv}[:,:,k] = \sum_{c=1}^{C_{in}} \boldsymbol{W}[:,:,c,k] * \boldsymbol{X}[:,:,c] \\ \boldsymbol{A}[:,:,k] = \boldsymbol{Y}^{Conv}[:,:,k] + \boldsymbol{b}[k] \\ \boldsymbol{H}[:,:,k] = \phi(\boldsymbol{A}[:,:,k]) \end{cases}$$

An Example

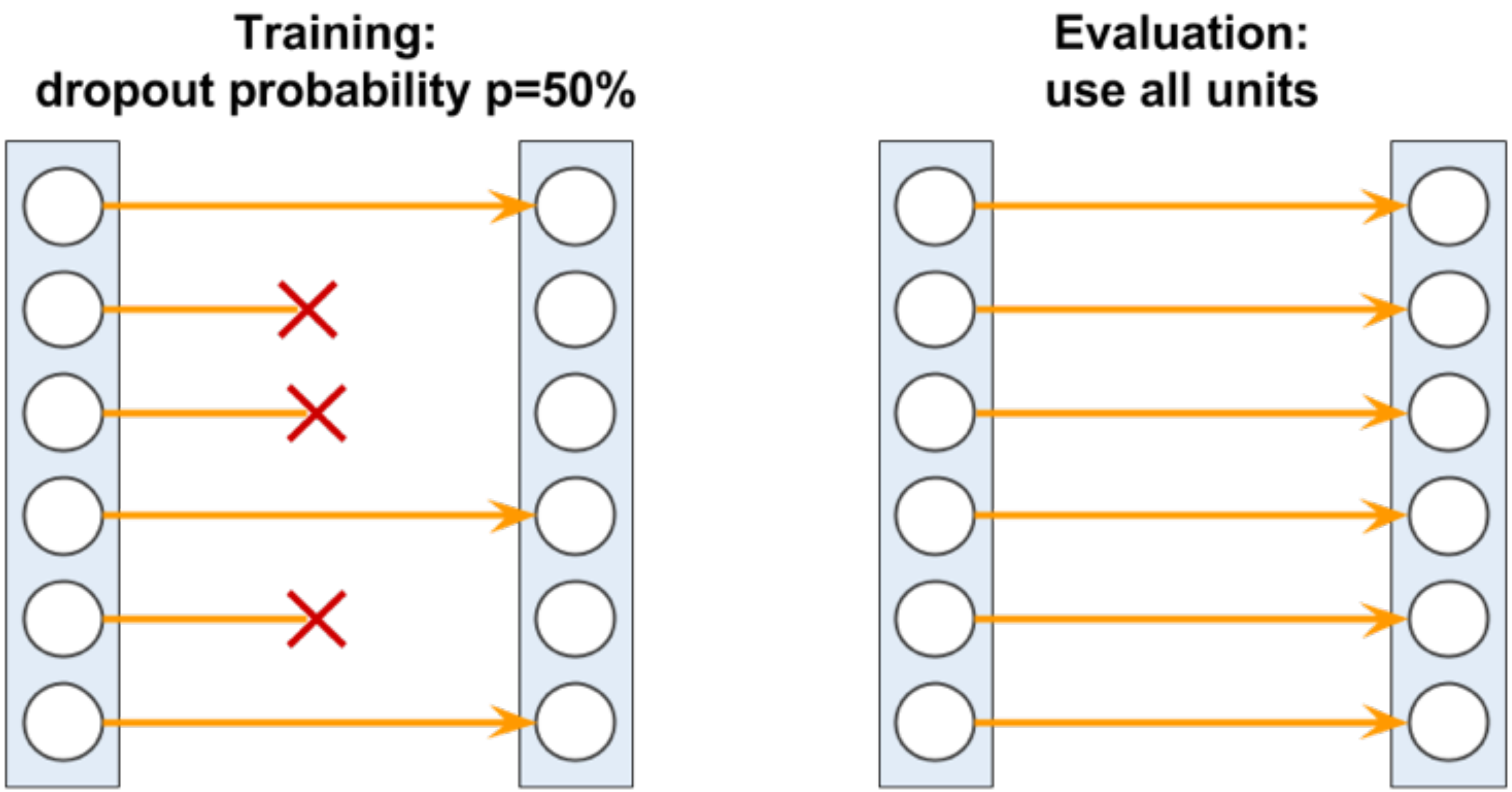

5 output feature maps
Convolution layer
Pooling layer
Σ
c_in
C_in =
Sum over input channels
C_out = 5
5
3 input channels
Kernel
m1×m2
C_in
m1 × m2 × 3 × 5
4D kernel tensor

Regularizing a Neural Network with Dropout

- Choosing the size of a network has always been a challenging problem
  - The *capacity* of a network refers to the level of complexity of the function that it can learn. underfit vs. overfit issue
- Ways to address the problem
  - Build a network with relatively large capacity with L2 regularization
  - Dropout
    - Can be considered as the consensus (averaging) of an ensemble of models

Dropout

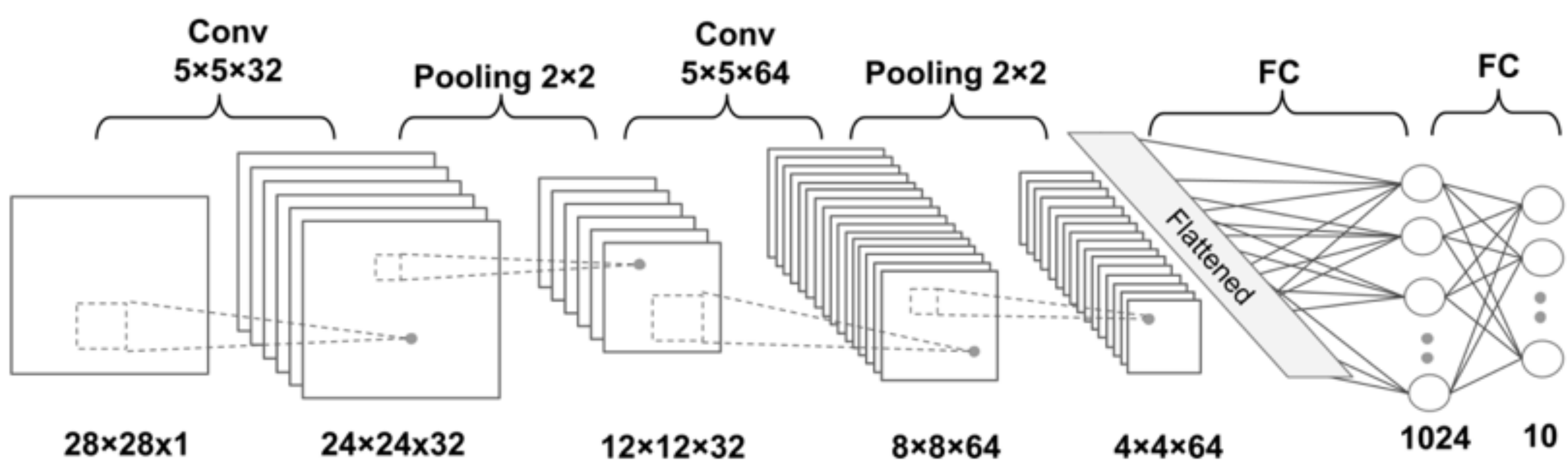
- Dropout is usually applied to the hidden units of higher layers with probability P_{drop}
- Random dropout at training and evaluate with all units



Implementing Deep Convolutional Neural Networks in TensorFlow

Use a CNN to Classify Handwritten Digits

- Input tensor: 28x28x1 (28x28 greyscale images)
- kernel size: 5x5
- 1st convolutional output 32 feature maps and 2nd output 64 feature maps
- Each convolution layer is followed by a subsampling layer in the form of a max-pooling



Load and Preprocess the Data (1/3)

```
## unzips mnist

import sys
import gzip
import shutil
import os


if (sys.version_info > (3, 0)):
    writemode = 'wb'
else:
    writemode = 'w'

zipped_mnist = [f for f in os.listdir('./')
                if f.endswith('ubyte.gz')]
for z in zipped_mnist:
    with gzip.GzipFile(z, mode='rb') as decompressed, open(z[:-3], writemode) as outfile:
        outfile.write(decompressed.read())
```

Load and Preprocess the Data (2/3)

```
import struct
import numpy as np


def load_mnist(path, kind='train'):
    """Load MNIST data from `path`"""
    labels_path = os.path.join(path,
                               '%s-labels-idx1-ubyte'
                                % kind)
    images_path = os.path.join(path,
                               '%s-images-idx3-ubyte'
                               % kind)

    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II',
                                 lbpath.read(8))
        labels = np.fromfile(lbpath,
                             dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack(">IIII",
                                               imgpath.read(16))
        images = np.fromfile(imgpath,
                             dtype=np.uint8).reshape(len(labels), 784)

    return images, labels
```

Load and Preprocess the Data (3/3)

```
X_data, y_data = load_mnist('./', kind='train')
print('Rows: %d,  Columns: %d' % (X_data.shape[0], X_data.shape[1]))
X_test, y_test = load_mnist('./', kind='t10k')
print('Rows: %d,  Columns: %d' % (X_test.shape[0], X_test.shape[1]))

X_train, y_train = X_data[:50000,:], y_data[:50000]
X_valid, y_valid = X_data[50000:,:], y_data[50000:]

print('Training:    ', X_train.shape, y_train.shape)
print('Validation: ', X_valid.shape, y_valid.shape)
print('Test Set:    ', X_test.shape, y_test.shape)
```

```
Rows: 60000,  Columns: 784
Rows: 10000,  Columns: 784
Training:    (50000, 784) (50000,)
Validation:  (10000, 784) (10000,)
Test Set:    (10000, 784) (10000,)
```

Generate the Mini-batches

```
def batch_generator(X, y, batch_size=64,
                    shuffle=False, random_seed=None):

    idx = np.arange(y.shape[0])

    if shuffle:
        rng = np.random.RandomState(random_seed)
        rng.shuffle(idx)
        X = X[idx]
        y = y[idx]

    for i in range(0, X.shape[0], batch_size):
        yield (X[i:i+batch_size, :], y[i:i+batch_size])
```

```
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)

X_train_centered = (X_train - mean_vals)/std_val
X_valid_centered = X_valid - mean_vals
X_test_centered = (X_test - mean_vals)/std_val

del X_data, y_data, X_train, X_valid, X_test
```

Implementing a CNN in the TensorFlow Low-Level API

Convolutional Layer (1/2)

```python
import tensorflow as tf
import numpy as np


## wrapper functions

def conv_layer(input_tensor, name,
               kernel_size, n_output_channels,
               padding_mode='SAME', strides=(1, 1, 1, 1)):
    with tf.variable_scope(name):
        ## get n_input_channels:
        ##   input tensor shape:
        ##   [batch x width x height x channels_in]
        input_shape = input_tensor.get_shape().as_list()
        n_input_channels = input_shape[-1]

        weights_shape = (list(kernel_size) +
                         [n_input_channels, n_output_channels])

        weights = tf.get_variable(name='_weights',
                                  shape=weights_shape)
        print(weights)
        biases = tf.get_variable(name='_biases',
                                 initializer=tf.zeros(
                                     shape=[n_output_channels]))
        print(biases)
        conv = tf.nn.conv2d(input=input_tensor,
                            filter=weights,
                            strides=strides,
                            padding=padding_mode)
        print(conv)
```

Convolutional Layer (2/2)

```
        conv = tf.nn.bias_add(conv, biases,
                              name='net_pre-activation')
        print(conv)
        conv = tf.nn.relu(conv, name='activation')
        print(conv)

        return conv


## testing
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(tf.float32, shape=[None, 28, 28, 1])
    conv_layer(x, name='convtest', kernel_size=(3, 3), n_output_channels=32)

del g, x
```

```
<tf.Variable 'convtest/_weights:0' shape=(3, 3, 1, 32) dtype=float32_ref>
<tf.Variable 'convtest/_biases:0' shape=(32,) dtype=float32_ref>
Tensor("convtest/Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("convtest/net_pre-activation:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("convtest/activation:0", shape=(?, 28, 28, 32), dtype=float32)
```

Fully Connected Layer (1/2)

```
def fc_layer(input_tensor, name,
             n_output_units, activation_fn=None):
    with tf.variable_scope(name):
        input_shape = input_tensor.get_shape().as_list()[1:]
        n_input_units = np.prod(input_shape)
        if len(input_shape) > 1:
            input_tensor = tf.reshape(input_tensor,
                                      shape=(-1, n_input_units))

        weights_shape = [n_input_units, n_output_units]

        weights = tf.get_variable(name='_weights',
                                  shape=weights_shape)
        print(weights)
        biases = tf.get_variable(name='_biases',
                                 initializer=tf.zeros(
                                     shape=[n_output_units]))
        print(biases)
        layer = tf.matmul(input_tensor, weights)
        print(layer)
        layer = tf.nn.bias_add(layer, biases,
                               name='net_pre-activation')
        print(layer)
        if activation_fn is None:
            return layer

        layer = activation_fn(layer, name='activation')
        print(layer)
        return layer
```

Fully Connected Layer (1/2)

```python
## testing:
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(tf.float32,
                       shape=[None, 28, 28, 1])
    fc_layer(x, name='fctest', n_output_units=32,
             activation_fn=tf.nn.relu)

del g, x
```

```
<tf.Variable 'fctest/_weights:0' shape=(784, 32) dtype=float32_ref>
<tf.Variable 'fctest/_biases:0' shape=(32,) dtype=float32_ref>
Tensor("fctest/MatMul:0", shape=(?, 32), dtype=float32)
Tensor("fctest/net_pre-activation:0", shape=(?, 32), dtype=float32)
Tensor("fctest/activation:0", shape=(?, 32), dtype=float32)
```

Build CNN (1/6)

```
def build_cnn():
    ## Placeholders for X and y:
    tf_x = tf.placeholder(tf.float32, shape=[None, 784],
                          name='tf_x')
    tf_y = tf.placeholder(tf.int32, shape=[None],
                          name='tf_y')

    # reshape x to a 4D tensor:
    # [batchsize, width, height, 1]
    tf_x_image = tf.reshape(tf_x, shape=[-1, 28, 28, 1],
                            name='tf_x_reshaped')
    ## One-hot encoding:
    tf_y_onehot = tf.one_hot(indices=tf_y, depth=10,
                             dtype=tf.float32,
                             name='tf_y_onehot')

    ## 1st layer: Conv_1
    print('\nBuilding 1st layer: ')
    h1 = conv_layer(tf_x_image, name='conv_1',
                    kernel_size=(5, 5),
                    padding_mode='VALID',
                    n_output_channels=32)
    ## MaxPooling
    h1_pool = tf.nn.max_pool(h1,
                             ksize=[1, 2, 2, 1],
                             strides=[1, 2, 2, 1],
                             padding='SAME')
```

Build CNN (2/6)

```python
## 2n layer: Conv_2
print('\nBuilding 2nd layer: ')
h2 = conv_layer(h1_pool, name='conv_2',
                kernel_size=(5,5),
                padding_mode='VALID',
                n_output_channels=64)
## MaxPooling
h2_pool = tf.nn.max_pool(h2,
                         ksize=[1, 2, 2, 1],
                         strides=[1, 2, 2, 1],
                         padding='SAME')

## 3rd layer: Fully Connected
print('\nBuilding 3rd layer:')
h3 = fc_layer(h2_pool, name='fc_3',
              n_output_units=1024,
              activation_fn=tf.nn.relu)

## Dropout
keep_prob = tf.placeholder(tf.float32, name='fc_keep_prob')
h3_drop = tf.nn.dropout(h3, keep_prob=keep_prob,
                        name='dropout_layer')

## 4th layer: Fully Connected (linear activation)
print('\nBuilding 4th layer:')
h4 = fc_layer(h3_drop, name='fc_4',
              n_output_units=10,
              activation_fn=None)
```

Build CNN (3/6)

```
## Prediction
predictions = {
    'probabilities' : tf.nn.softmax(h4, name='probabilities'),
    'labels' : tf.cast(tf.argmax(h4, axis=1), tf.int32,
                       name='labels')
}

## Visualize the graph with TensorBoard:

## Loss Function and Optimization
cross_entropy_loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(
        logits=h4, labels=tf_y_onehot),
    name='cross_entropy_loss')

## Optimizer:
optimizer = tf.train.AdamOptimizer(learning_rate)
optimizer = optimizer.minimize(cross_entropy_loss,
                               name='train_op')

## Computing the prediction accuracy
correct_predictions = tf.equal(
    predictions['labels'],
    tf_y, name='correct_preds')

accuracy = tf.reduce_mean(
    tf.cast(correct_predictions, tf.float32),
    name='accuracy')
```

Build CNN (4/6)

```python
def save(saver, sess, epoch, path='./model/'):
    if not os.path.isdir(path):
        os.makedirs(path)
    print('Saving model in %s' % path)
    saver.save(sess, os.path.join(path,'cnn-model.ckpt'),
               global_step=epoch)


def load(saver, sess, path, epoch):
    print('Loading model from %s' % path)
    saver.restore(sess, os.path.join(
        path, 'cnn-model.ckpt-%d' % epoch))
```

Build CNN (5/6)

```
def train(sess, training_set, validation_set=None,
          initialize=True, epochs=20, shuffle=True,
          dropout=0.5, random_seed=None):

    X_data = np.array(training_set[0])
    y_data = np.array(training_set[1])
    training_loss = []

    ## initialize variables
    if initialize:
        sess.run(tf.global_variables_initializer())

    np.random.seed(random_seed) # for shuflling in batch_generator
    for epoch in range(1, epochs+1):
        batch_gen = batch_generator(
                        X_data, y_data,
                        shuffle=shuffle)
        avg_loss = 0.0
        for i,(batch_x,batch_y) in enumerate(batch_gen):
            feed = {'tf_x:0': batch_x,
                    'tf_y:0': batch_y,
                    'fc_keep_prob:0': dropout}
            loss, _ = sess.run(
                    ['cross_entropy_loss:0', 'train_op'],
                    feed_dict=feed)
            avg_loss += loss
        training_loss.append(avg_loss / (i+1))
        print('Epoch %02d Training Avg. Loss: %7.3f' % (
            epoch, avg_loss), end=' ')
        if validation_set is not None:
            feed = {'tf_x:0': validation_set[0],
                    'tf_y:0': validation_set[1],
                    'fc_keep_prob:0':1.0}
            valid_acc = sess.run('accuracy:0', feed_dict=feed)
            print(' Validation Acc: %7.3f' % valid_acc)
        else:
            print()
```

Build CNN (6/6)

```
def predict(sess, X_test, return_proba=False):
    feed = {'tf_x:0': X_test,
            'fc_keep_prob:0': 1.0}
    if return_proba:
        return sess.run('probabilities:0', feed_dict=feed)
    else:
        return sess.run('labels:0', feed_dict=feed)
```

Build TensorFlow Graph Object (1/2)

```
import tensorflow as tf
import numpy as np

## Define hyperparameters
learning_rate = 1e-4
random_seed = 123

np.random.seed(random_seed)


## create a graph
g = tf.Graph()
with g.as_default():
    tf.set_random_seed(random_seed)
    ## build the graph
    build_cnn()

    ## saver:
    saver = tf.train.Saver()
```

Build TensorFlow Graph Object (2/2)

```
Building 1st layer:
<tf.Variable 'conv_1/_weights:0' shape=(5, 5, 1, 32) dtype=float32_ref>
<tf.Variable 'conv_1/_biases:0' shape=(32,) dtype=float32_ref>
Tensor("conv_1/Conv2D:0", shape=(?, 24, 24, 32), dtype=float32)
Tensor("conv_1/net_pre-activation:0", shape=(?, 24, 24, 32), dtype=float32)
Tensor("conv_1/activation:0", shape=(?, 24, 24, 32), dtype=float32)

Building 2nd layer:
<tf.Variable 'conv_2/_weights:0' shape=(5, 5, 32, 64) dtype=float32_ref>
<tf.Variable 'conv_2/_biases:0' shape=(64,) dtype=float32_ref>
Tensor("conv_2/Conv2D:0", shape=(?, 8, 8, 64), dtype=float32)
Tensor("conv_2/net_pre-activation:0", shape=(?, 8, 8, 64), dtype=float32)
Tensor("conv_2/activation:0", shape=(?, 8, 8, 64), dtype=float32)

Building 3rd layer:
<tf.Variable 'fc_3/_weights:0' shape=(1024, 1024) dtype=float32_ref>
<tf.Variable 'fc_3/_biases:0' shape=(1024,) dtype=float32_ref>
Tensor("fc_3/MatMul:0", shape=(?, 1024), dtype=float32)
Tensor("fc_3/net_pre-activation:0", shape=(?, 1024), dtype=float32)
Tensor("fc_3/activation:0", shape=(?, 1024), dtype=float32)

Building 4th layer:
<tf.Variable 'fc_4/_weights:0' shape=(1024, 10) dtype=float32_ref>
<tf.Variable 'fc_4/_biases:0' shape=(10,) dtype=float32_ref>
Tensor("fc_4/MatMul:0", shape=(?, 10), dtype=float32)
Tensor("fc_4/net_pre-activation:0", shape=(?, 10), dtype=float32)
```

Training the CNN Model

```
## crearte a TF session
## and train the CNN model

with tf.Session(graph=g) as sess:
    train(sess,
          training_set=(X_train_centered, y_train),
          validation_set=(X_valid_centered, y_valid),
          initialize=True,
          random_seed=123)
    save(saver, sess, epoch=20)
```

```
Epoch 01 Training Avg. Loss: 274.884  Validation Acc:   0.973
Epoch 02 Training Avg. Loss:  76.537  Validation Acc:   0.981
Epoch 03 Training Avg. Loss:  51.816  Validation Acc:   0.984
Epoch 04 Training Avg. Loss:  38.888  Validation Acc:   0.986
Epoch 05 Training Avg. Loss:  33.064  Validation Acc:   0.987
Epoch 06 Training Avg. Loss:  27.396  Validation Acc:   0.990
Epoch 07 Training Avg. Loss:  23.094  Validation Acc:   0.987
Epoch 08 Training Avg. Loss:  20.075  Validation Acc:   0.989
Epoch 09 Training Avg. Loss:  16.844  Validation Acc:   0.991
Epoch 10 Training Avg. Loss:  15.895  Validation Acc:   0.990
Epoch 11 Training Avg. Loss:  13.291  Validation Acc:   0.989
Epoch 12 Training Avg. Loss:  10.677  Validation Acc:   0.991
Epoch 13 Training Avg. Loss:  10.241  Validation Acc:   0.992
Epoch 14 Training Avg. Loss:   9.578  Validation Acc:   0.990
Epoch 15 Training Avg. Loss:   7.571  Validation Acc:   0.992
Epoch 16 Training Avg. Loss:   6.860  Validation Acc:   0.991
Epoch 17 Training Avg. Loss:   6.921  Validation Acc:   0.990
Epoch 18 Training Avg. Loss:   5.492  Validation Acc:   0.991
Epoch 19 Training Avg. Loss:   4.859  Validation Acc:   0.991
Epoch 20 Training Avg. Loss:   4.572  Validation Acc:   0.992
```

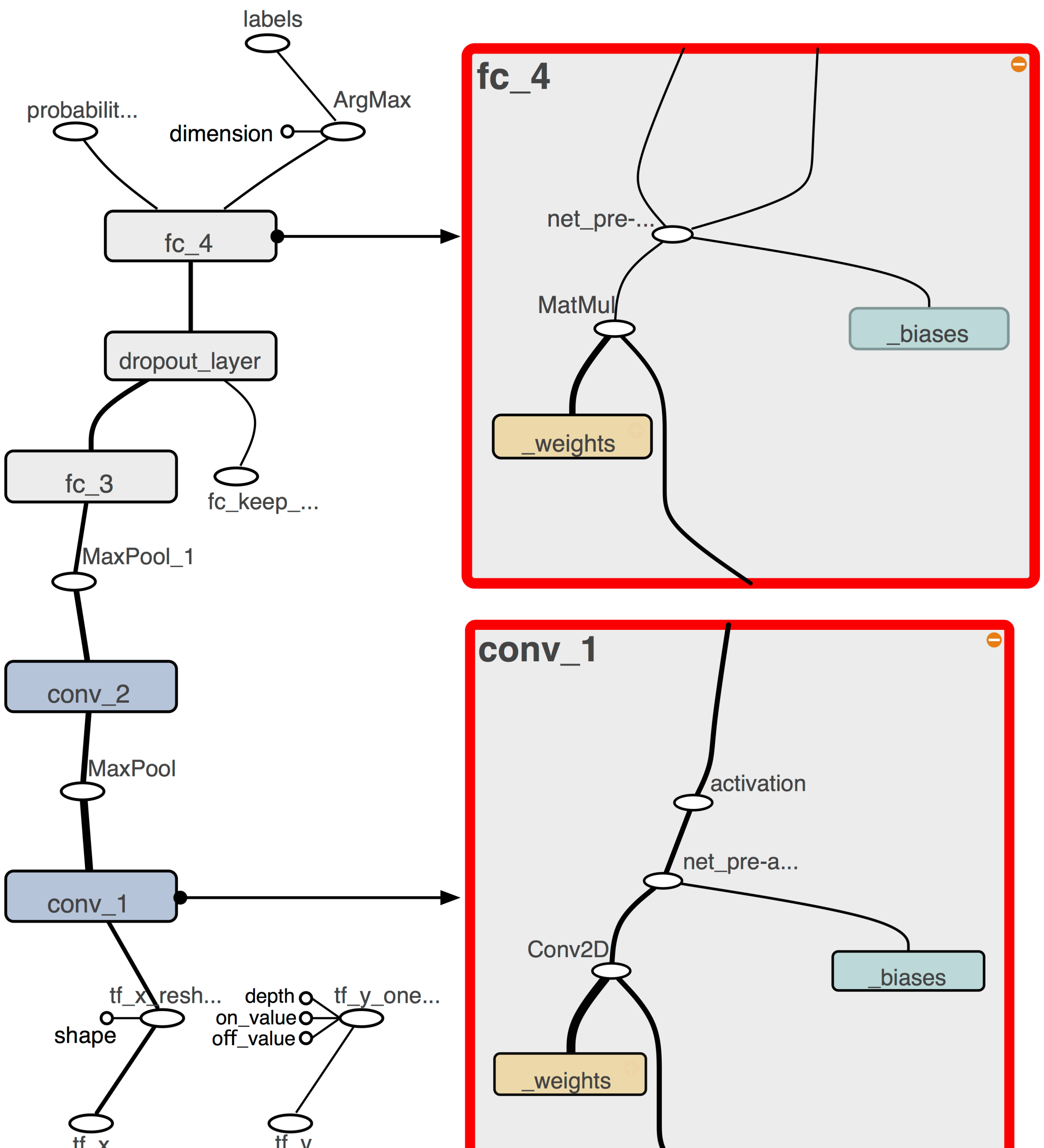
Prediction Accuracy (1/2)

```
### Calculate prediction accuracy
### on test set
### restoring the saved model

del g

## create a new graph
## and build the model
g2 = tf.Graph()
with g2.as_default():
    tf.set_random_seed(random_seed)
    ## build the graph
    build_cnn()

    ## saver:
    saver = tf.train.Saver()

## create a new session
## and restore the model
with tf.Session(graph=g2) as sess:
    load(saver, sess,
         epoch=20, path='./model/')

    preds = predict(sess, X_test_centered,
                    return_proba=False)

    print('Test Accuracy: %.3f%%' % (100*
                np.sum(preds == y_test)/len(y_test)))
```

Prediction Accuracy (2/2)

```
Building 1st layer:
<tf.Variable 'conv_1/_weights:0' shape=(5, 5, 1, 32) dtype=float32_ref>
<tf.Variable 'conv_1/_biases:0' shape=(32,) dtype=float32_ref>
Tensor("conv_1/Conv2D:0", shape=(?, 24, 24, 32), dtype=float32)
Tensor("conv_1/net_pre-activation:0", shape=(?, 24, 24, 32), dtype=float32)
Tensor("conv_1/activation:0", shape=(?, 24, 24, 32), dtype=float32)

Building 2nd layer:
<tf.Variable 'conv_2/_weights:0' shape=(5, 5, 32, 64) dtype=float32_ref>
<tf.Variable 'conv_2/_biases:0' shape=(64,) dtype=float32_ref>
Tensor("conv_2/Conv2D:0", shape=(?, 8, 8, 64), dtype=float32)
Tensor("conv_2/net_pre-activation:0", shape=(?, 8, 8, 64), dtype=float32)
Tensor("conv_2/activation:0", shape=(?, 8, 8, 64), dtype=float32)

Building 3rd layer:
<tf.Variable 'fc_3/_weights:0' shape=(1024, 1024) dtype=float32_ref>
<tf.Variable 'fc_3/_biases:0' shape=(1024,) dtype=float32_ref>
Tensor("fc_3/MatMul:0", shape=(?, 1024), dtype=float32)
Tensor("fc_3/net_pre-activation:0", shape=(?, 1024), dtype=float32)
Tensor("fc_3/activation:0", shape=(?, 1024), dtype=float32)

Building 4th layer:
<tf.Variable 'fc_4/_weights:0' shape=(1024, 10) dtype=float32_ref>
<tf.Variable 'fc_4/_biases:0' shape=(10,) dtype=float32_ref>
Tensor("fc_4/MatMul:0", shape=(?, 10), dtype=float32)
Tensor("fc_4/net_pre-activation:0", shape=(?, 10), dtype=float32)
Loading model from ./model/
INFO:tensorflow:Restoring parameters from ./model/cnn-model.ckpt-20
Test Accuracy: 99.310%
```

Visualize the Graph with TensorBoard



Implementing a CNN in the TensorFlow Layers API

Class Definition

```python
import tensorflow as tf
import numpy as np
class ConvNN(object):
    def __init__(self, batchsize=64,
                 epochs=20, learning_rate=1e-4,
                 dropout_rate=0.5,
                 shuffle=True, random_seed=None):
        np.random.seed(random_seed)
        self.batchsize = batchsize
        self.epochs = epochs
        self.learning_rate = learning_rate
        self.dropout_rate = dropout_rate
        self.shuffle = shuffle

        g = tf.Graph()
        with g.as_default():
            ## set random-seed:
            tf.set_random_seed(random_seed)

            ## build the network:
            self.build()

            ## initializer
            self.init_op = \
                tf.global_variables_initializer()

            ## saver
            self.saver = tf.train.Saver()

        ## create a session
        self.sess = tf.Session(graph=g)
```

Build the Model (1/3)

```python
def build(self):

    ## Placeholders for X and y:
    tf_x = tf.placeholder(tf.float32,
                          shape=[None, 784],
                          name='tf_x')
    tf_y = tf.placeholder(tf.int32,
                          shape=[None],
                          name='tf_y')
    is_train = tf.placeholder(tf.bool,
                          shape=(),
                          name='is_train')

    ## reshape x to a 4D tensor:
    ##  [batchsize, width, height, 1]
    tf_x_image = tf.reshape(tf_x, shape=[-1, 28, 28, 1],
                          name='input_x_2dimages')
    ## One-hot encoding:
    tf_y_onehot = tf.one_hot(indices=tf_y, depth=10,
                          dtype=tf.float32,
                          name='input_y_onehot')

    ## 1st layer: Conv_1
    h1 = tf.layers.conv2d(tf_x_image,
                          kernel_size=(5, 5),
                          filters=32,
                          activation=tf.nn.relu)
    ## MaxPooling
    h1_pool = tf.layers.max_pooling2d(h1,
                          pool_size=(2, 2),
                          strides=(2, 2))
```

Build the Model (2/3)

```
## 2n layer: Conv_2
h2 = tf.layers.conv2d(h1_pool, kernel_size=(5,5),
                      filters=64,
                      activation=tf.nn.relu)
## MaxPooling
h2_pool = tf.layers.max_pooling2d(h2,
                      pool_size=(2, 2),
                      strides=(2, 2))

## 3rd layer: Fully Connected
input_shape = h2_pool.get_shape().as_list()
n_input_units = np.prod(input_shape[1:])
h2_pool_flat = tf.reshape(h2_pool,
                      shape=[-1, n_input_units])
h3 = tf.layers.dense(h2_pool_flat, 1024,
                      activation=tf.nn.relu)

## Dropout
h3_drop = tf.layers.dropout(h3,
                      rate=self.dropout_rate,
                      training=is_train)

## 4th layer: Fully Connected (linear activation)
h4 = tf.layers.dense(h3_drop, 10,
                      activation=None)
```

Build the Model (3/3)

```
## Prediction
predictions = {
    'probabilities': tf.nn.softmax(h4,
                         name='probabilities'),
    'labels': tf.cast(tf.argmax(h4, axis=1),
                         tf.int32, name='labels')}

## Loss Function and Optimization
cross_entropy_loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(
        logits=h4, labels=tf_y_onehot),
    name='cross_entropy_loss')

## Optimizer:
optimizer = tf.train.AdamOptimizer(self.learning_rate)
optimizer = optimizer.minimize(cross_entropy_loss,
                         name='train_op')

## Finding accuracy
correct_predictions = tf.equal(
    predictions['labels'],
    tf_y, name='correct_preds')

accuracy = tf.reduce_mean(
    tf.cast(correct_predictions, tf.float32),
    name='accuracy')
```

Load and Save the Model

```
def save(self, epoch, path='./tflayers-model/'):
    if not os.path.isdir(path):
        os.makedirs(path)
    print('Saving model in %s' % path)
    self.saver.save(self.sess,
                    os.path.join(path, 'model.ckpt'),
                    global_step=epoch)

def load(self, epoch, path):
    print('Loading model from %s' % path)
    self.saver.restore(self.sess,
         os.path.join(path, 'model.ckpt-%d' % epoch))
```

Model Training (1/2)

```
    def train(self, training_set,
              validation_set=None,
              initialize=True):
        ## initialize variables
        if initialize:
            self.sess.run(self.init_op)

        self.train_cost_ = []
        X_data = np.array(training_set[0])
        y_data = np.array(training_set[1])

        for epoch in range(1, self.epochs + 1):
            batch_gen = \
                batch_generator(X_data, y_data,
                                 shuffle=self.shuffle)
            avg_loss = 0.0
            for i, (batch_x,batch_y) in \
                enumerate(batch_gen):
                feed = {'tf_x:0': batch_x,
                        'tf_y:0': batch_y,
                        'is_train:0': True} ## for dropout
                loss, _ = self.sess.run(
                        ['cross_entropy_loss:0', 'train_op'],
                        feed_dict=feed)
                avg_loss += loss
```

Model Training (2/2)

```
print('Epoch %02d: Training Avg. Loss: '
      '%7.3f' % (epoch, avg_loss), end=' ')
if validation_set is not None:
    feed = {'tf_x:0': batch_x,
            'tf_y:0': batch_y,
            'is_train:0': False} ## for dropout
    valid_acc = self.sess.run('accuracy:0',
                              feed_dict=feed)
    print('Validation Acc: %7.3f' % valid_acc)
else:
    print()
```

Prediction

```
def predict(self, X_test, return_proba = False):
    feed = {'tf_x:0': X_test,
            'is_train:0': False} ## for dropout
    if return_proba:
        return self.sess.run('probabilities:0',
                             feed_dict=feed)
    else:
        return self.sess.run('labels:0',
                             feed_dict=feed)
```

Build and Train the Model

```
cnn = ConvNN(random_seed=123)
```

```
cnn.train(training_set=(X_train_centered, y_train),
          validation_set=(X_valid_centered, y_valid))

cnn.save(epoch=20)
```

```
Epoch 01: Training Avg. Loss: 262.962 Validation Acc:   1.000
Epoch 02: Training Avg. Loss:  73.309 Validation Acc:   1.000
Epoch 03: Training Avg. Loss:  50.763 Validation Acc:   1.000
Epoch 04: Training Avg. Loss:  39.567 Validation Acc:   1.000
Epoch 05: Training Avg. Loss:  32.161 Validation Acc:   1.000
Epoch 06: Training Avg. Loss:  26.815 Validation Acc:   0.938
Epoch 07: Training Avg. Loss:  23.939 Validation Acc:   0.938
Epoch 08: Training Avg. Loss:  19.429 Validation Acc:   1.000
Epoch 09: Training Avg. Loss:  17.655 Validation Acc:   1.000
Epoch 10: Training Avg. Loss:  14.999 Validation Acc:   1.000
Epoch 11: Training Avg. Loss:  13.101 Validation Acc:   1.000
Epoch 12: Training Avg. Loss:  11.254 Validation Acc:   1.000
Epoch 13: Training Avg. Loss:   9.751 Validation Acc:   1.000
Epoch 14: Training Avg. Loss:   9.099 Validation Acc:   1.000
Epoch 15: Training Avg. Loss:   8.616 Validation Acc:   1.000
Epoch 16: Training Avg. Loss:   7.116 Validation Acc:   1.000
Epoch 17: Training Avg. Loss:   6.629 Validation Acc:   1.000
Epoch 18: Training Avg. Loss:   5.702 Validation Acc:   1.000
Epoch 19: Training Avg. Loss:   5.537 Validation Acc:   1.000
Epoch 20: Training Avg. Loss:   4.726 Validation Acc:   1.000
Saving model in ./tflayers-model/
```

Test the Model

```
del cnn

cnn2 = ConvNN(random_seed=123)

cnn2.load(epoch=20, path='./tflayers-model/')

print(cnn2.predict(X_test_centered[:10,:]))
```

```
Loading model from ./tflayers-model/
INFO:tensorflow:Restoring parameters from ./tflayers-model/model.ckpt-20
[7 2 1 0 4 1 4 9 5 9]
```

```
preds = cnn2.predict(X_test_centered)

print('Test Accuracy: %.2f%%' % (100*
      np.sum(y_test == preds)/len(y_test)))
```

```
Test Accuracy: 99.37%
```