TensorFlow Tutorial #03

Convolutional Neural Network

Seoung Wug Oh, Ph.D. Student
Computational Intelligence and Photography Lab.
Yonsei University

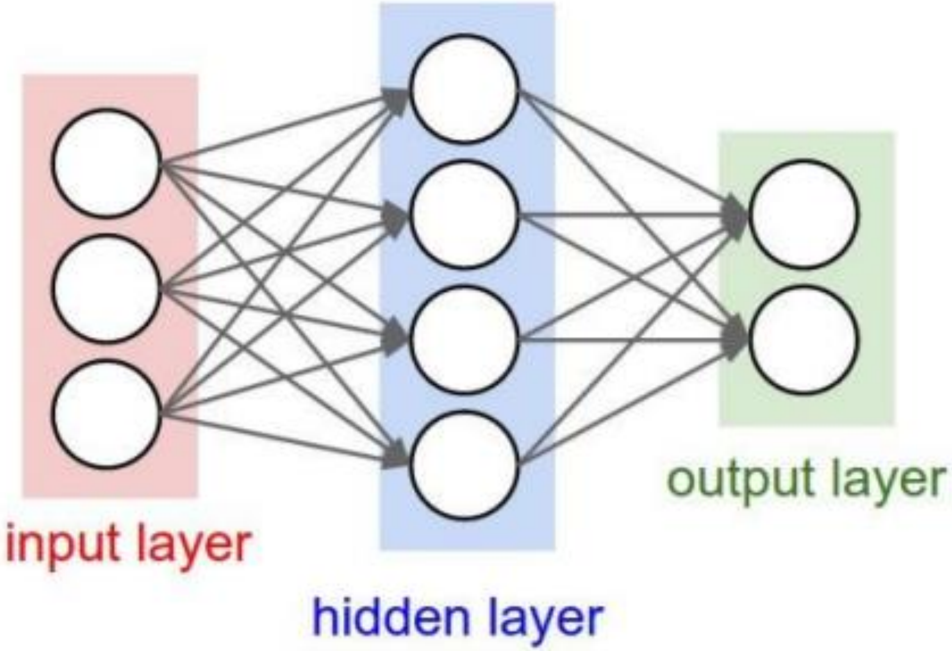
Neural Network

- For now
  - Linear function for logit: $logit = W \times x + b$

Neural Network

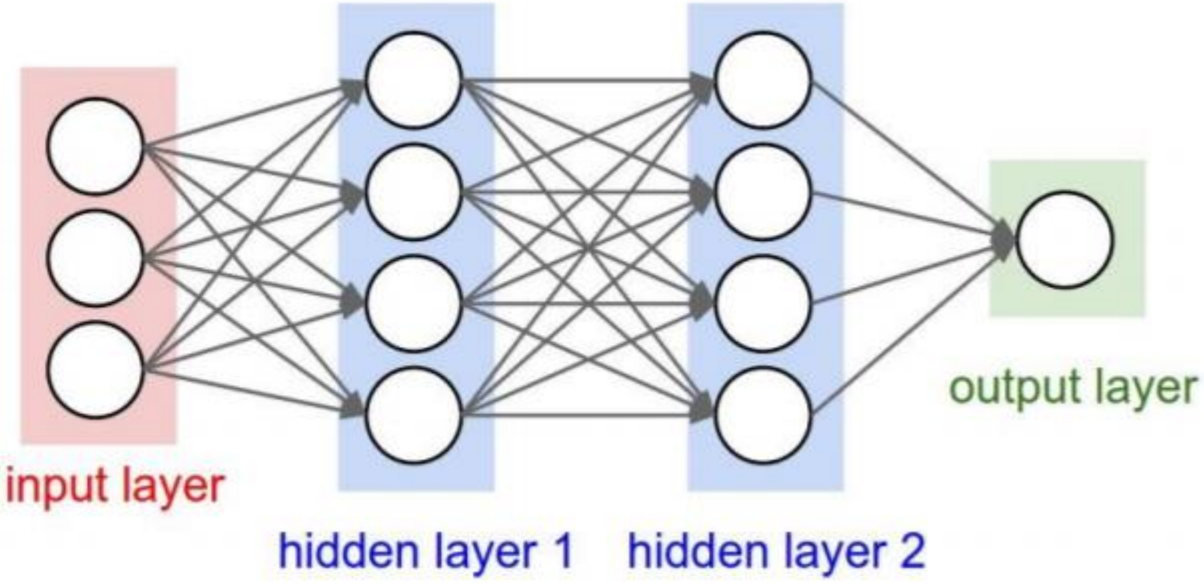
- For now
  - Linear function for logit: $logit = W \times x + b$

- 2-layer Neural Network
  - $logit = W_2 \times \sigma(W_1 \times x + b_1) + b_2$



Multi-Layer Perceptron

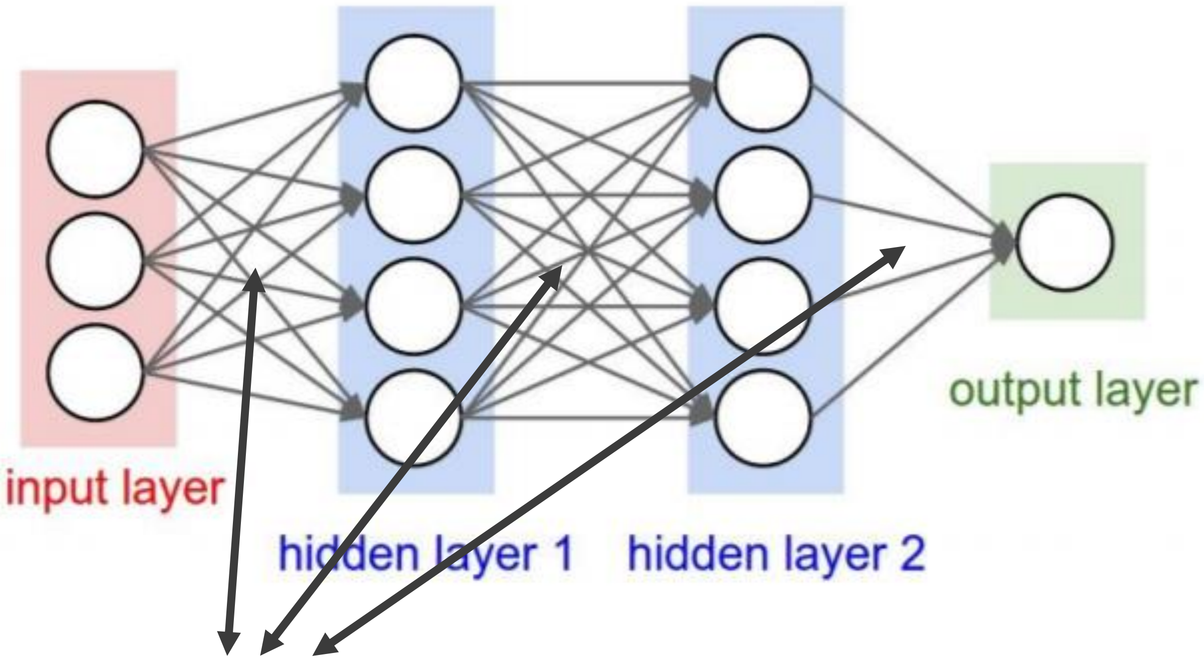
- 3-layer Neural Network
  - $logit = W_3 \times \sigma(W_2 \times \sigma(W_1 \times x + b_1) + b_2) + b_3$



  - Often called Multi-Layer Perceptron (MLP)

Neural Network

- 3-layer Neural Network
  - $logit = W_3 \times \sigma(W_2 \times \sigma(W_1 \times x + b_1) + b_2) + b_3$



Fully-connected or
or Dense 'Layer'

Fully-connected layer in TensorFlow

```python
def Dense(input, weight_shape, name='Dense'):
    '''
    Fully connected layer.
    '''
    with tf.variable_scope(name):
        W = tf.get_variable("W", weight_shape,
              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
        b = tf.get_variable("b", (weight_shape[-1]),
              initializer=tf.constant_initializer(value=0.0))
    return tf.matmul(input, W) + b
```

- **Layer:** weights (variable) + operation
  - This function defines new variables `W, b`
  - And also define operation `tf.matmul`

Fully-connected layer in TensorFlow

```
def Dense(input, weight_shape, name='Dense'):
    '''
    Fully connected layer.
    '''
    with tf.variable_scope(name):
        W = tf.get_variable("W", weight_shape,
              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
        b = tf.get_variable("b", (weight_shape[-1]),
              initializer=tf.constant_initializer(value=0.0))
    return tf.matmul(input, W) + b
```

- `With tf.variable_scope('layer1'):`
    - Set prefix to variable name
    - Variable name: `'W'` `->` `'layer1/W'`

Fully-connected layer in TensorFlow

```
def Dense(input, weight_shape, name='Dense'):
    '''
    Fully connected layer.
    '''
    with tf.variable_scope(name):
        W = tf.get_variable("W", weight_shape,
              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
        b = tf.get_variable("b", (weight_shape[-1]),
              initializer=tf.constant_initializer(value=0.0))
    return tf.matmul(input, W) + b
```

- `tf.get_variable()`:
  - Gets an existing variable with these parameters or create a new one.
  - Support `initializer`

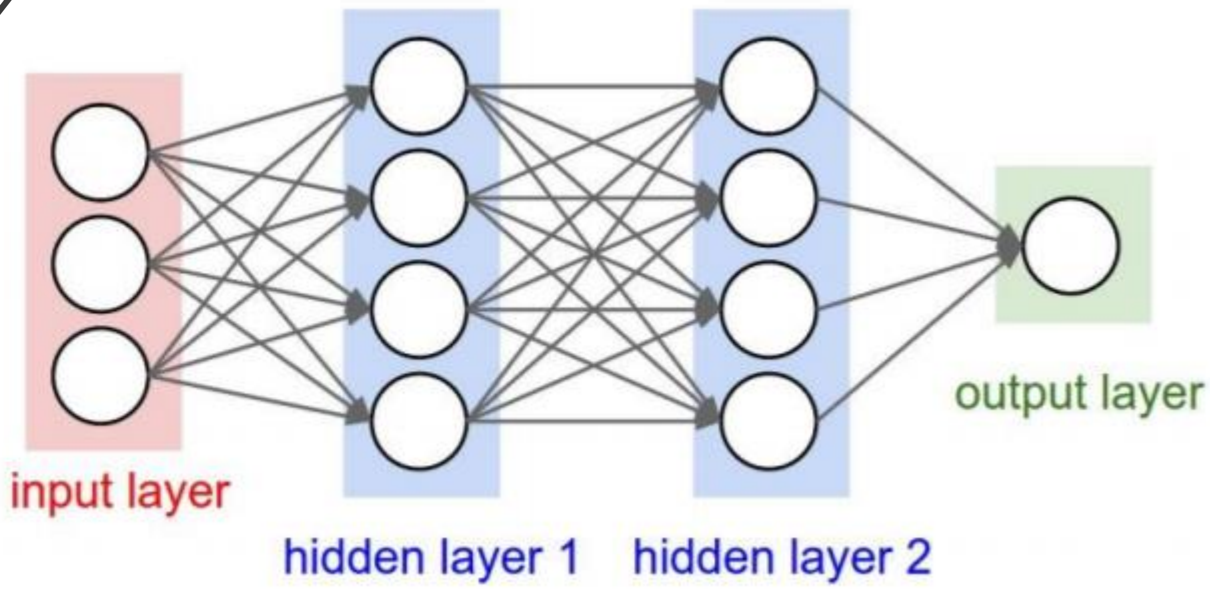
Fully-connected layer in TensorFlow

```
def Dense(input, weight_shape, name='Dense'):
    '''
    Fully connected layer.
    '''
    with tf.variable_scope(name):
        W = tf.get_variable("W", weight_shape,
              initializer=tf.contrib.layers.xavier_initializer(uniform=False))
        b = tf.get_variable("b", (weight_shape[-1]),
              initializer=tf.constant_initializer(value=0.0))
    return tf.matmul(input, W) + b
```

- Weight initialization
  - Xavier initialization
    `tf.contrib.layers.xavier_initializer(uniform=False)`
  - He's initialization
    `tf.contrib.layers.variance_scaling_initializer(factor=2.0, mode='FAN_IN', uniform=False)`

Activation function

- 3-layer Neural Network
  - $logit = W_3 \times \sigma(W_2 \times \sigma(W_1 \times x + b_1) + b_2) + b_3$

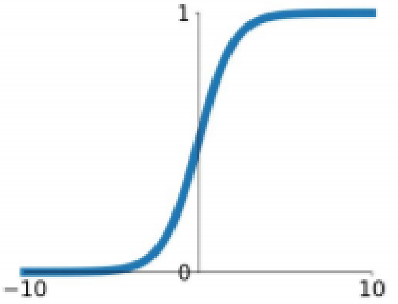


σ:
Activation function
Or non-linearity function

Activation function

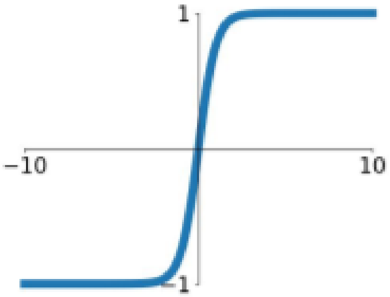
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



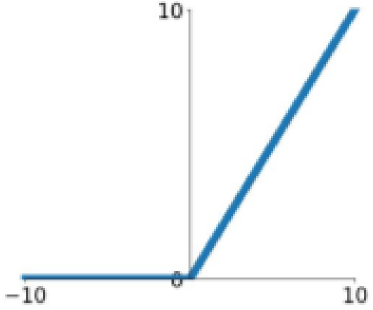
tanh

$\tanh(x)$



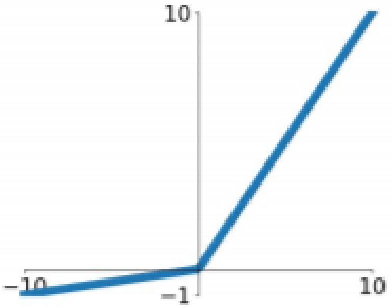
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

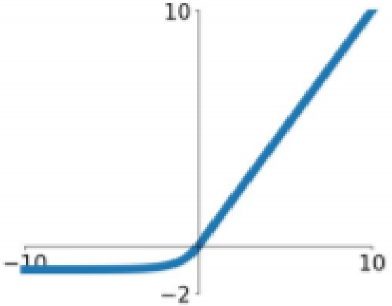


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

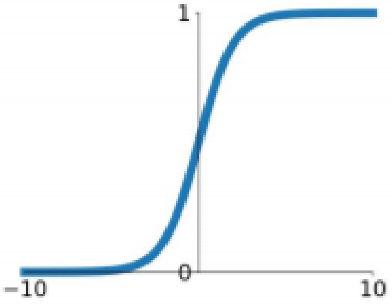
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation function

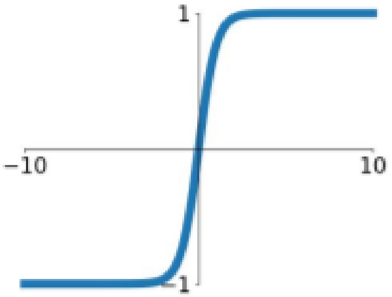
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



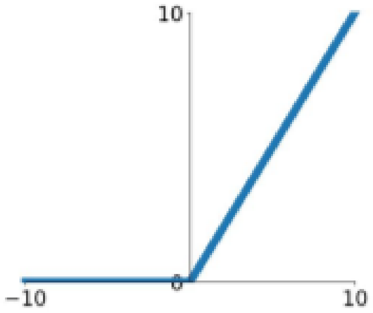
tanh

$\tanh(x)$



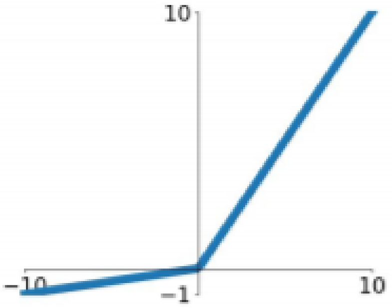
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

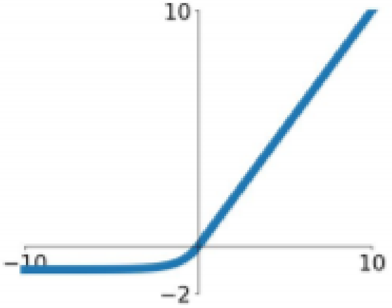


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

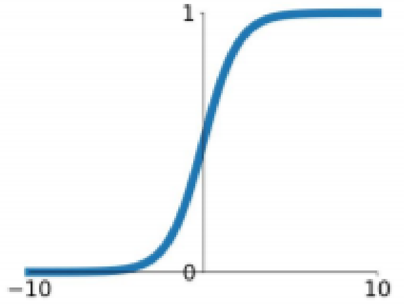
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation function

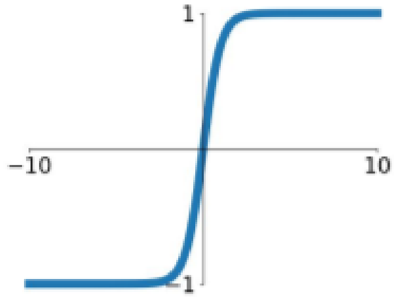
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



tanh

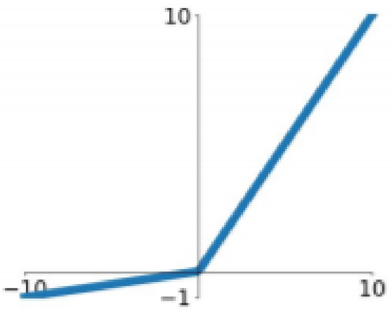
$\tanh(x)$




```
tf.nn.relu(x)
```

Leaky ReLU

$\max(0.1x, x)$

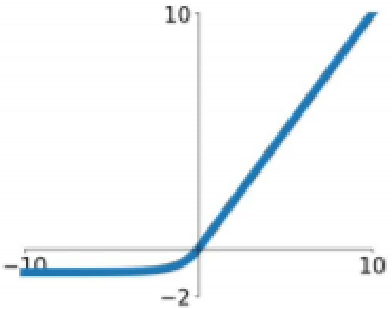


Maxout

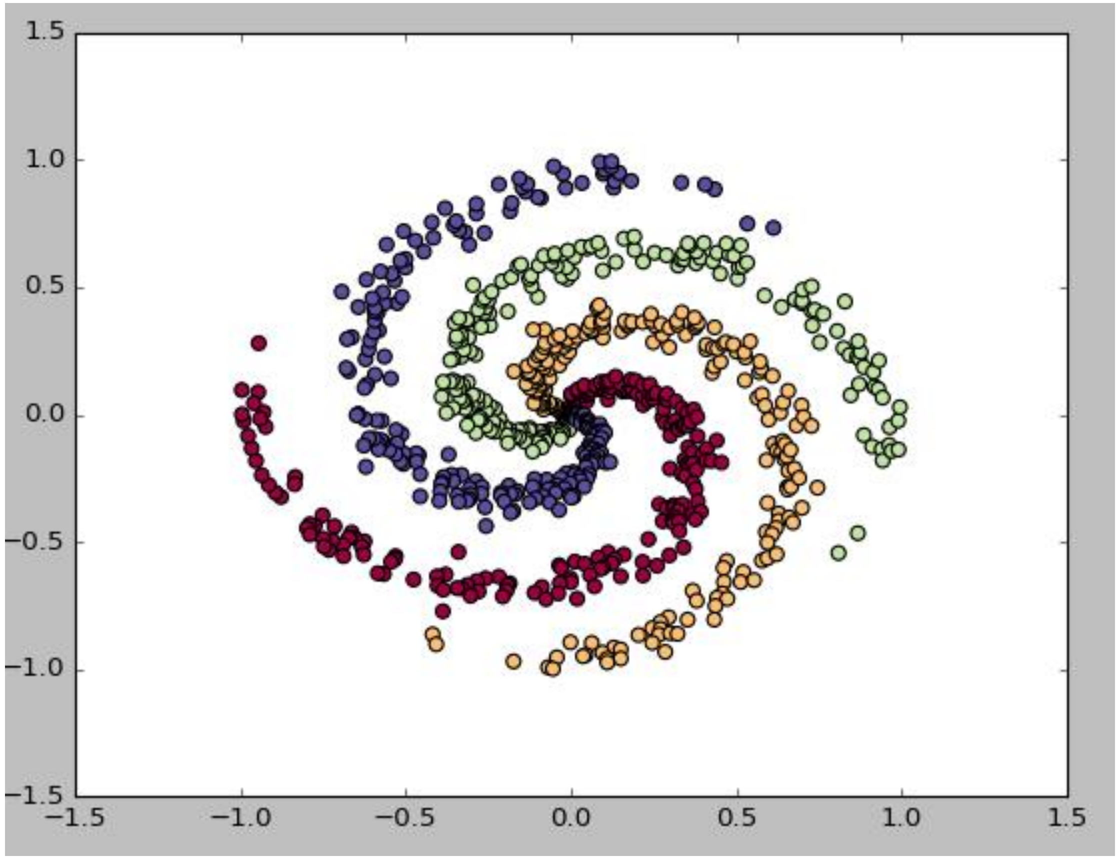
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Example: Back to 2D spiral data



Example: Back to 2D spiral data

```
###### Build graph ######################################################

# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 4]) # 4 classes
```

Example: Back to 2D spiral data

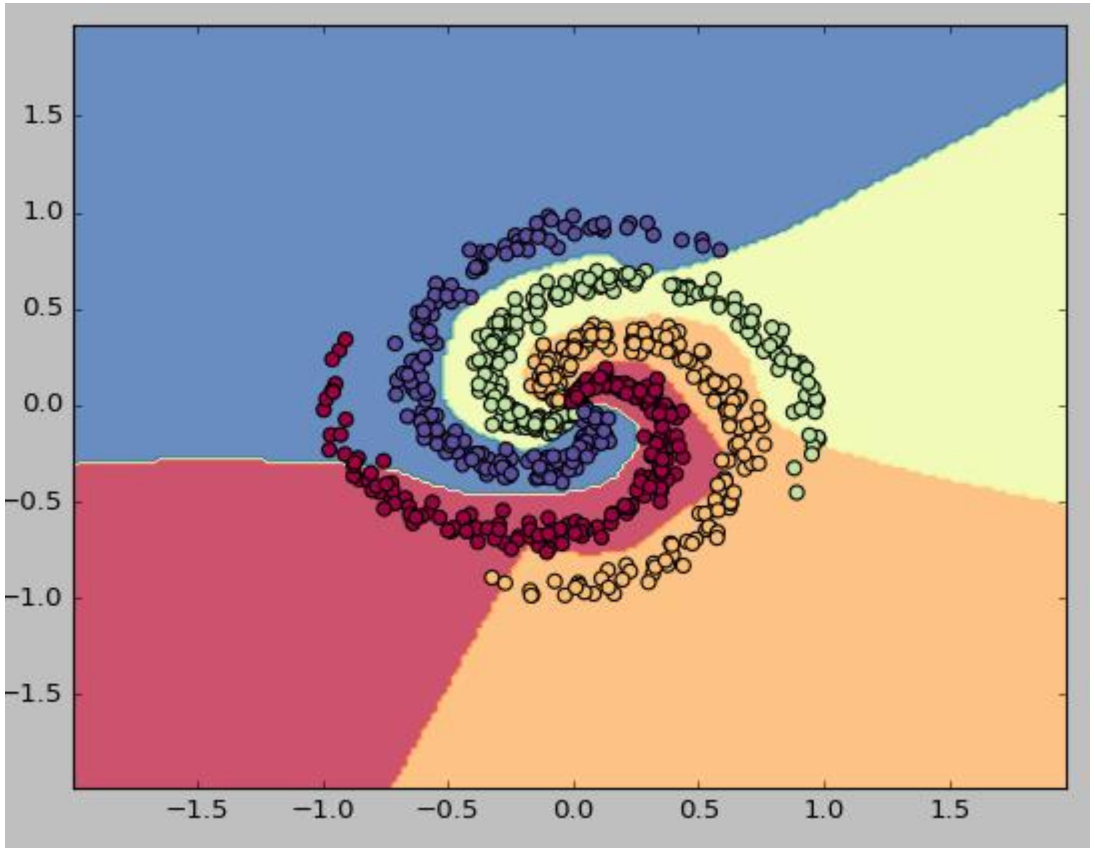
```
###### Build graph ######################################################

# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 4]) # 4 classes

# Construct MLP with two hidden layer
h = Dense(x, [2,64], 'ih')
h = tf.nn.relu(h)
h = Dense(h, [64,64], 'hh')
h = tf.nn.relu(h)
logit = Dense(h, [64,4], 'hl')

pred = tf.nn.softmax(logit) # Softmax
```

Example: Back to 2D spiral data

```
###### Build graph ######################################################

# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 4]) # 4 classes

# Construct MLP with two hidden layer
h = Dense(x, [2,64], 'ih')
h = tf.nn.relu(h)
h = Dense(h, [64,64], 'hh')
h = tf.nn.relu(h)
logit = Dense(h, [64,4], 'hl')

pred = tf.nn.softmax(logit) # Softmax

# Directly compute loss from logit (to ensure stability and avoid overflow)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logit, labels=y))

# Define optimizer and train_op
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Result

- Non-linear decision boundary



Convolutional Neural Networks

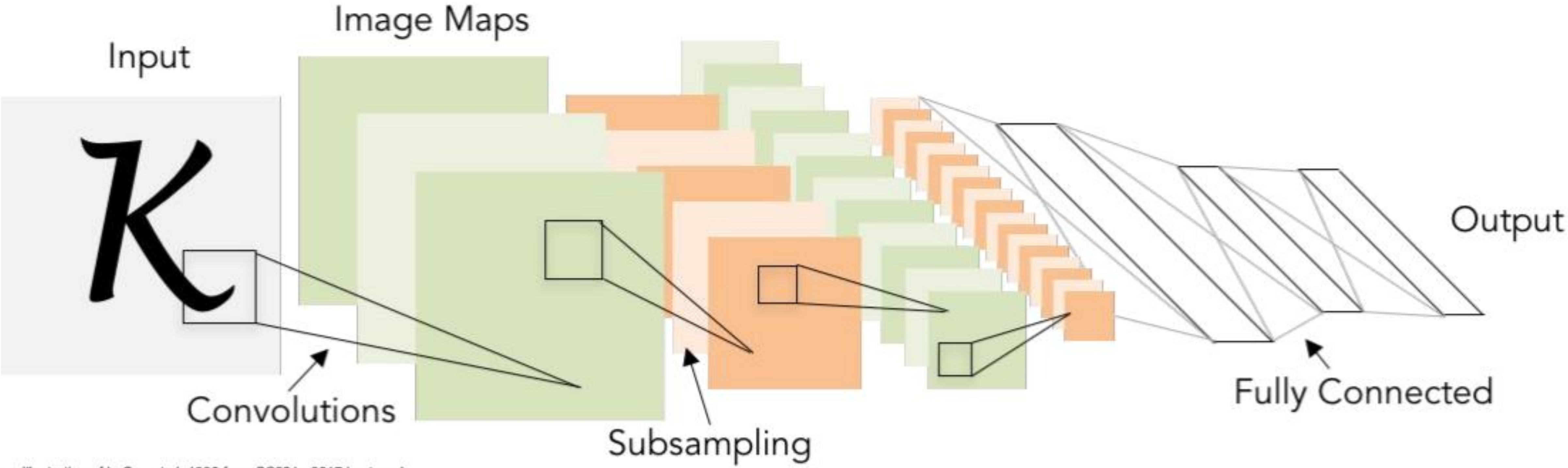
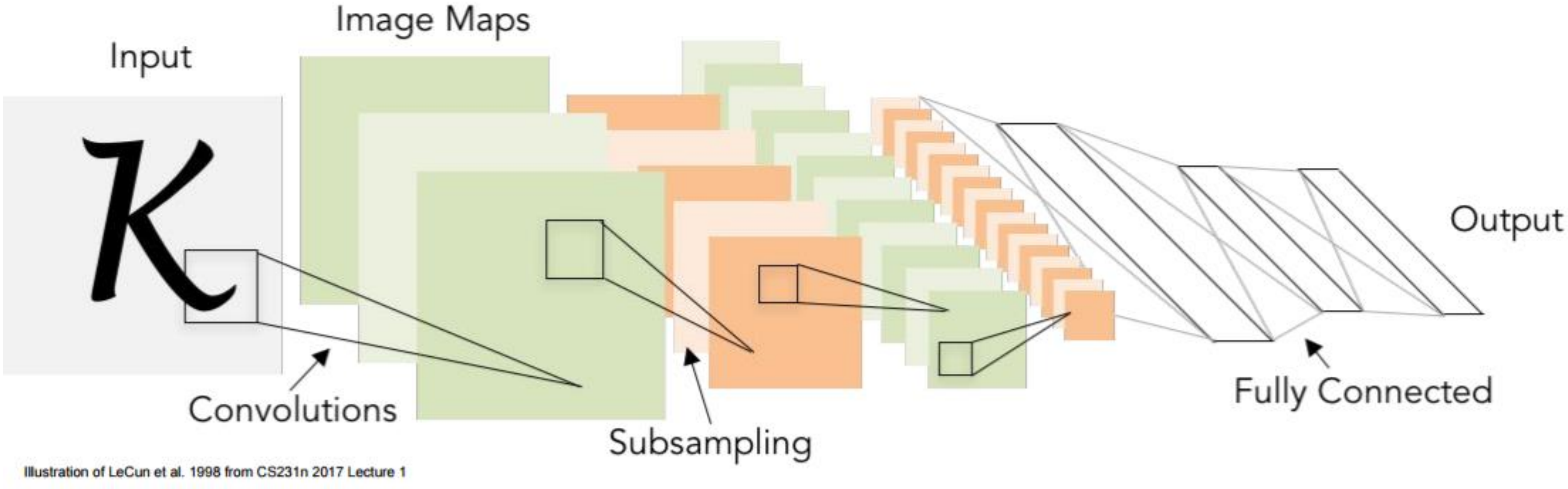


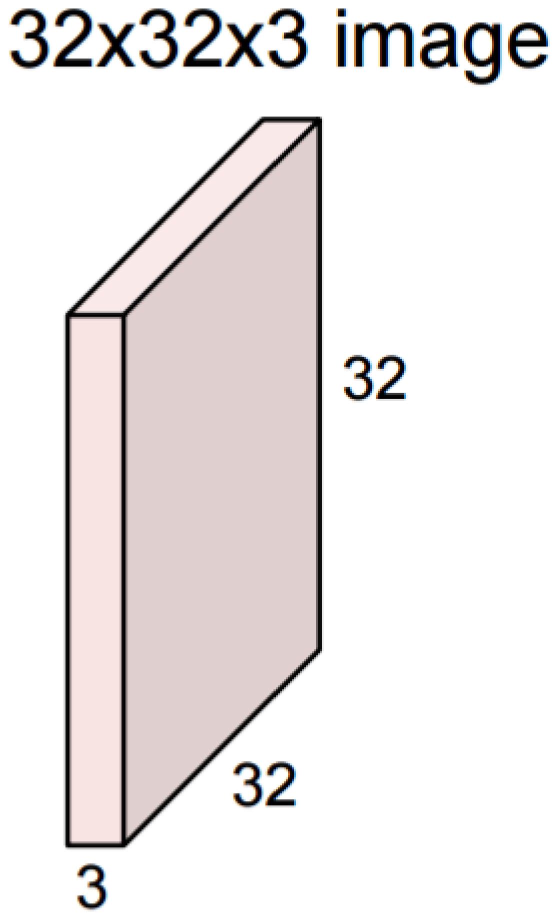
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

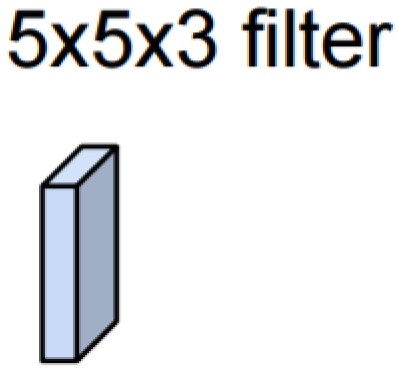
Convolutional Neural Networks



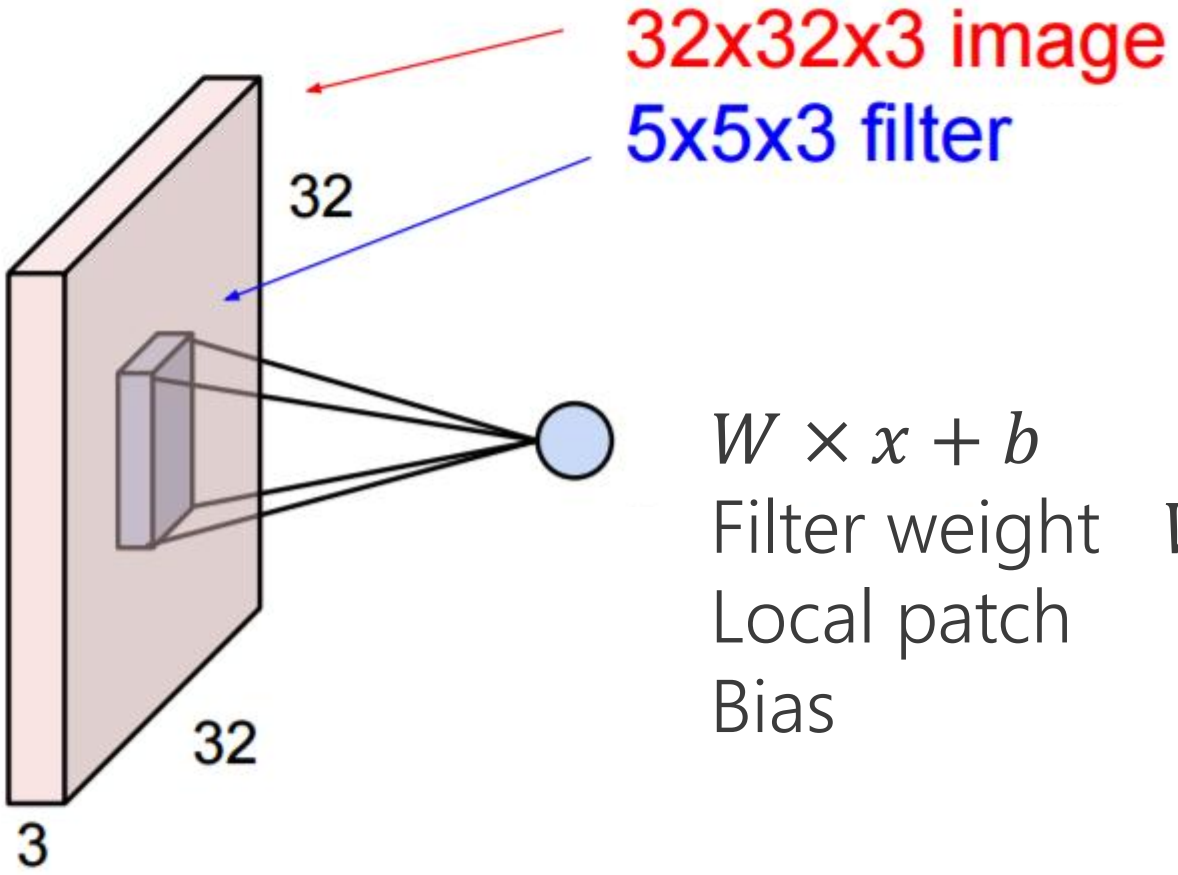
Convolutional Layer   Pooling Layer   Flatten   Fully connected Layer

2D Convolution Layer





2D Convolution Layer



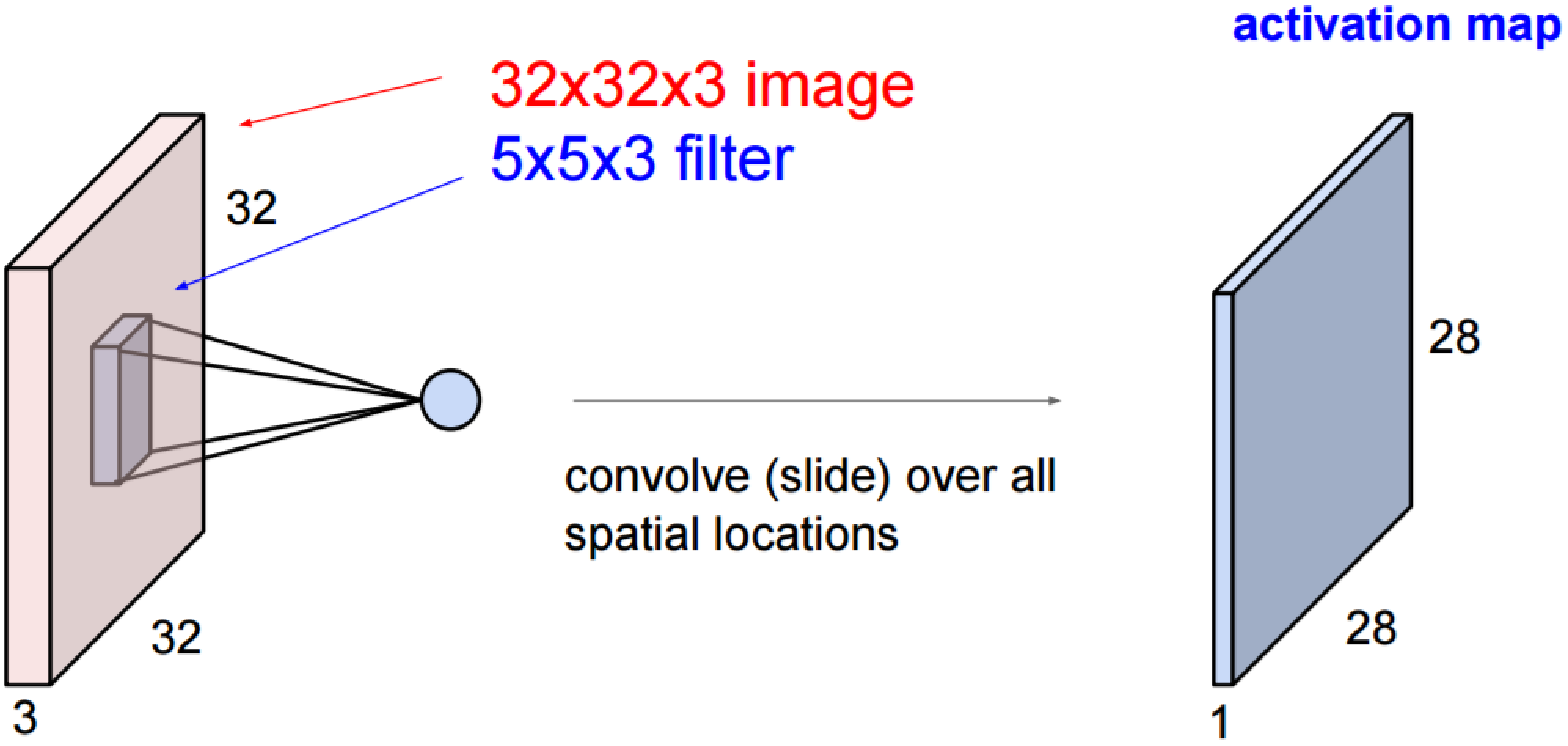
$W \times x + b$

Filter weight $W\colon\ 5 \times 5 \times 3 = 75$

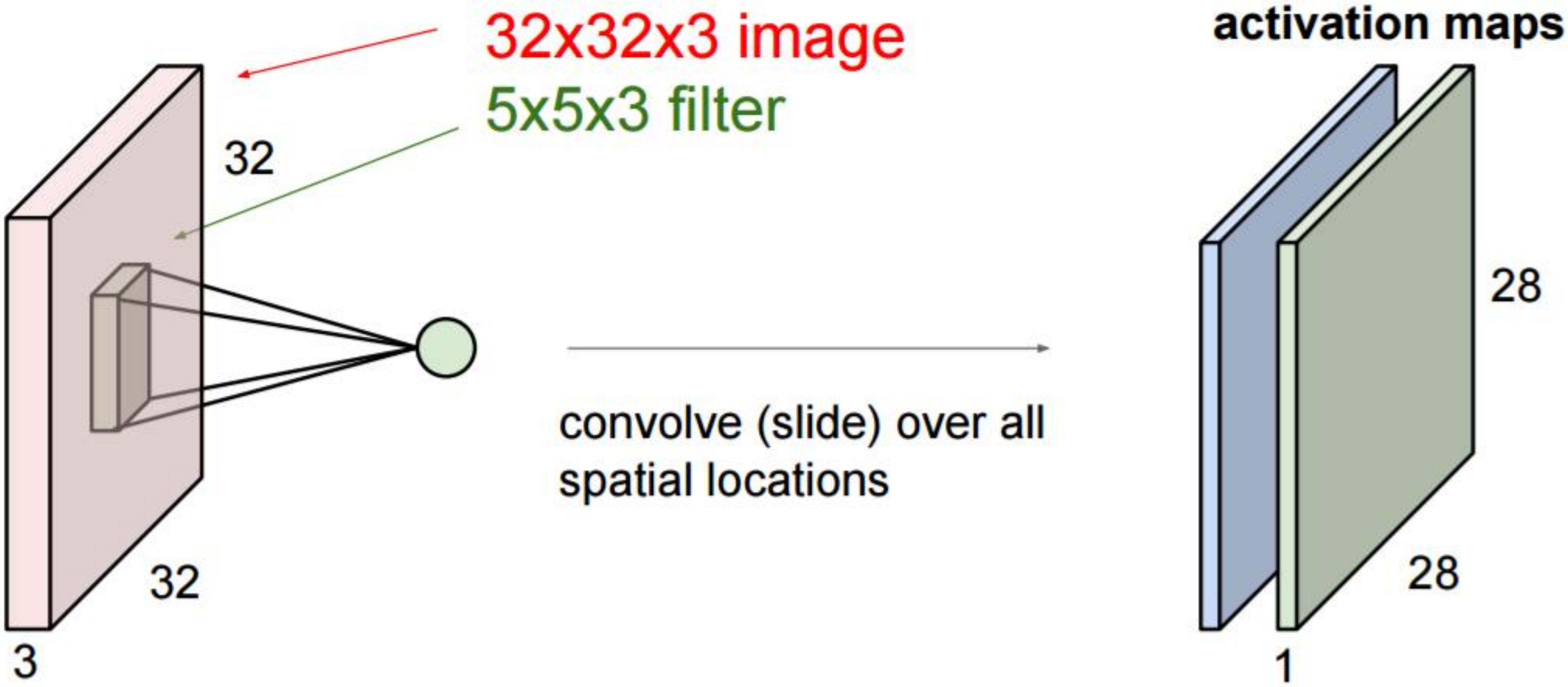
Local patch $x\colon\ 5 \times 5 \times 3 = 75$

Bias b: $scalar$

2D Convolution Layer

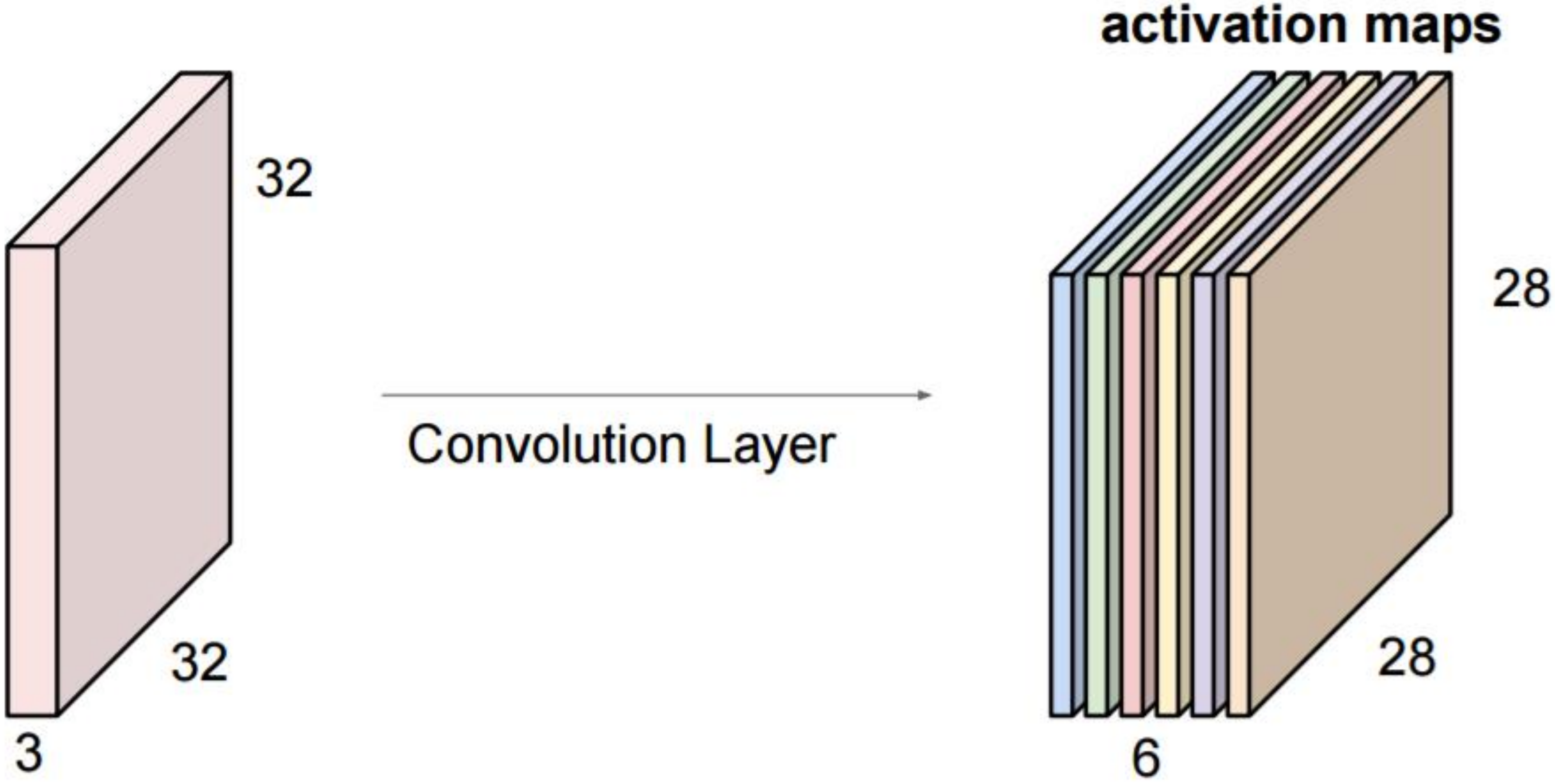


2D Convolution Layer



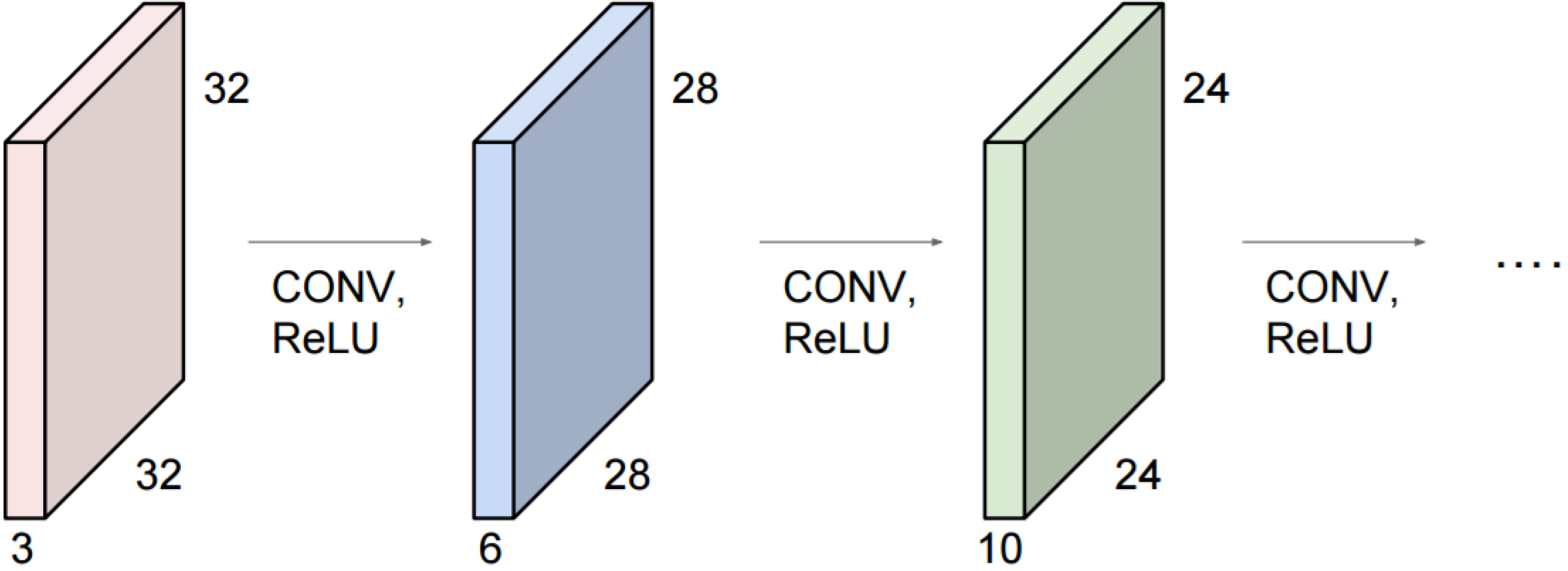
2D Convolution Layer

- If we have six 5x5x3 filters.



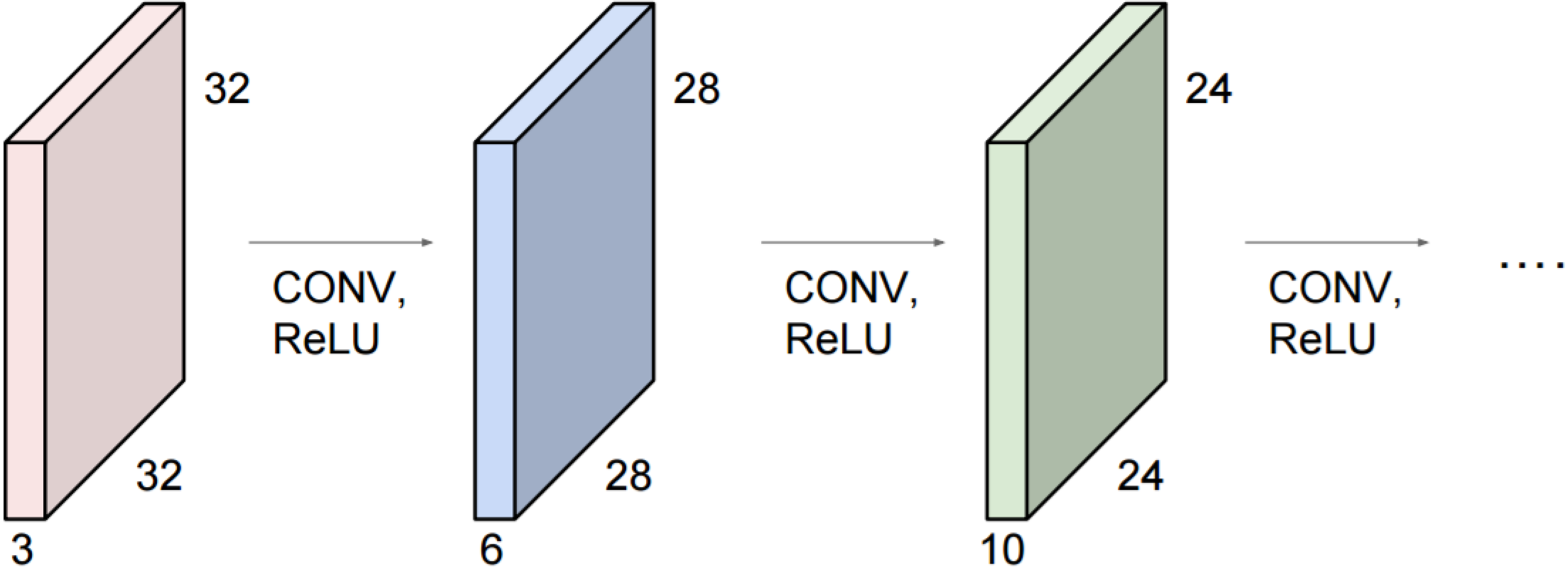
- This convolution layer have weight shape = [5,5,3,6].

2D Convolution Layer



weight shape = [5,5,3,6]   weight shape = ?

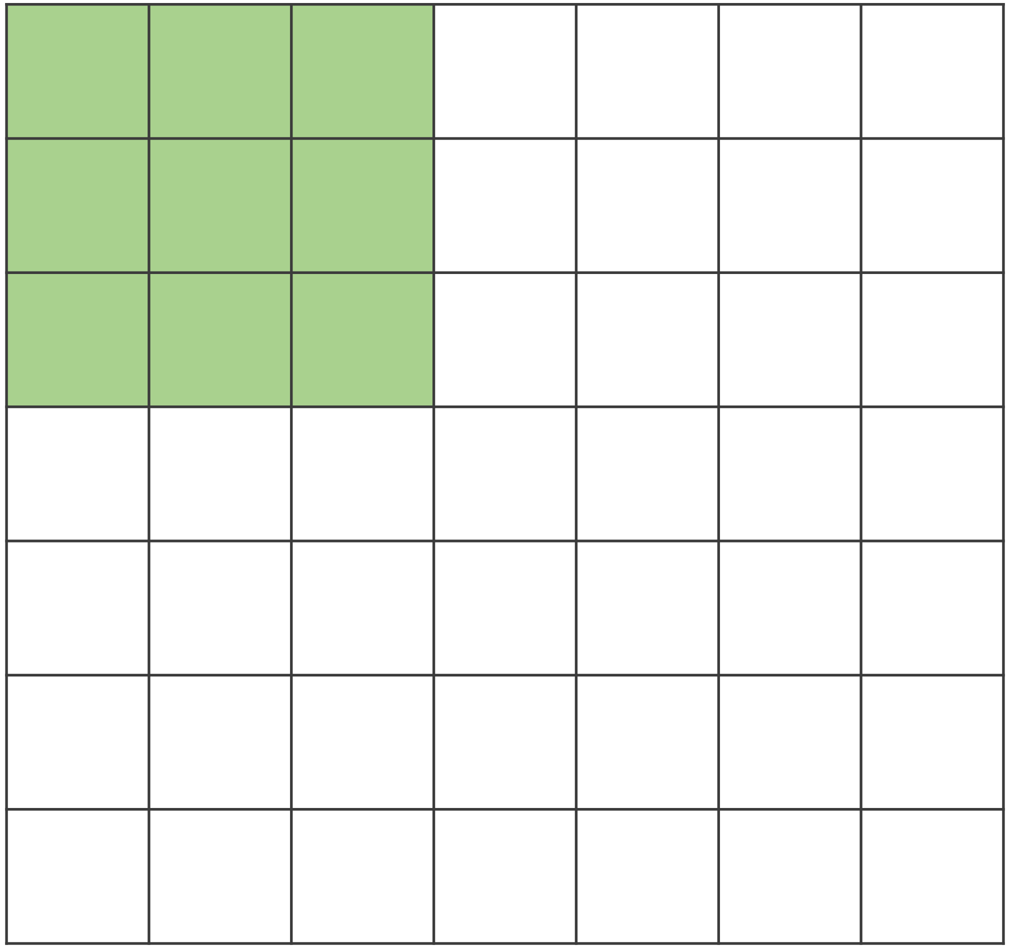
2D Convolution Layer



weight shape = [5,5,3,6]   weight shape = [5,5,6,10]

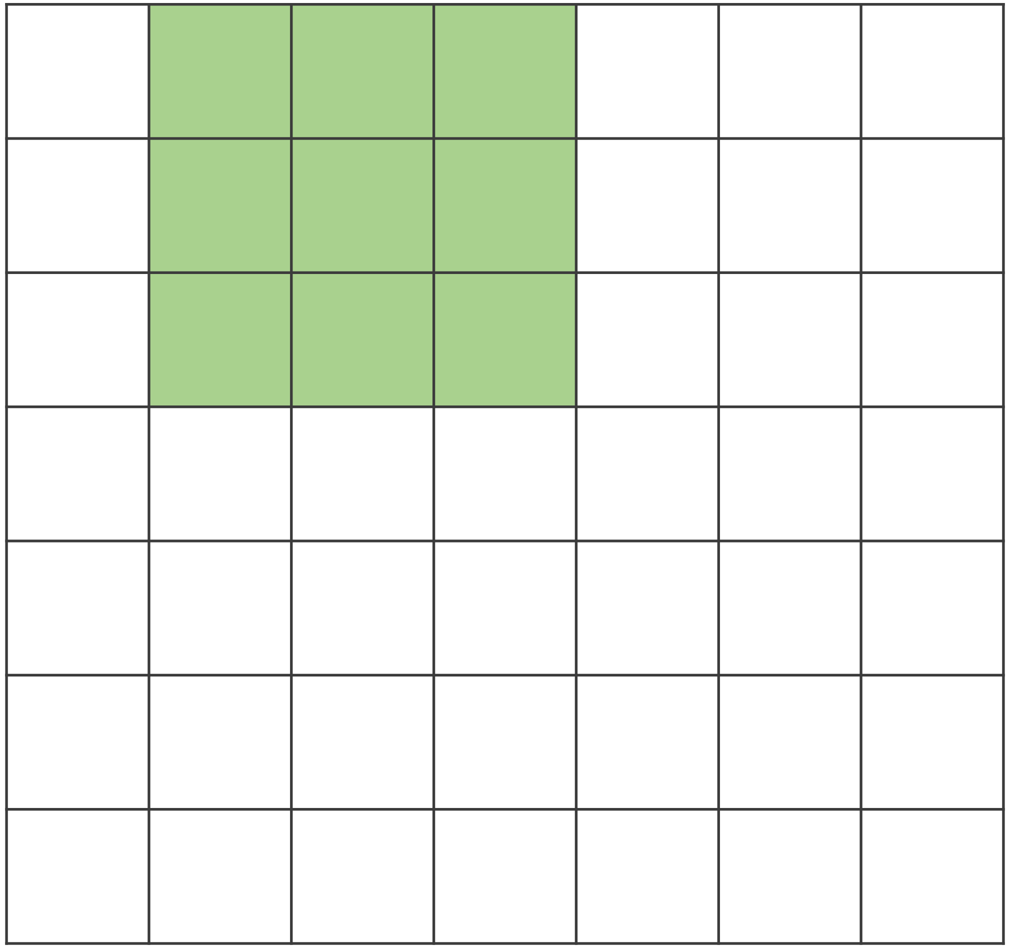
2D Convolution Layer

Stride = 1



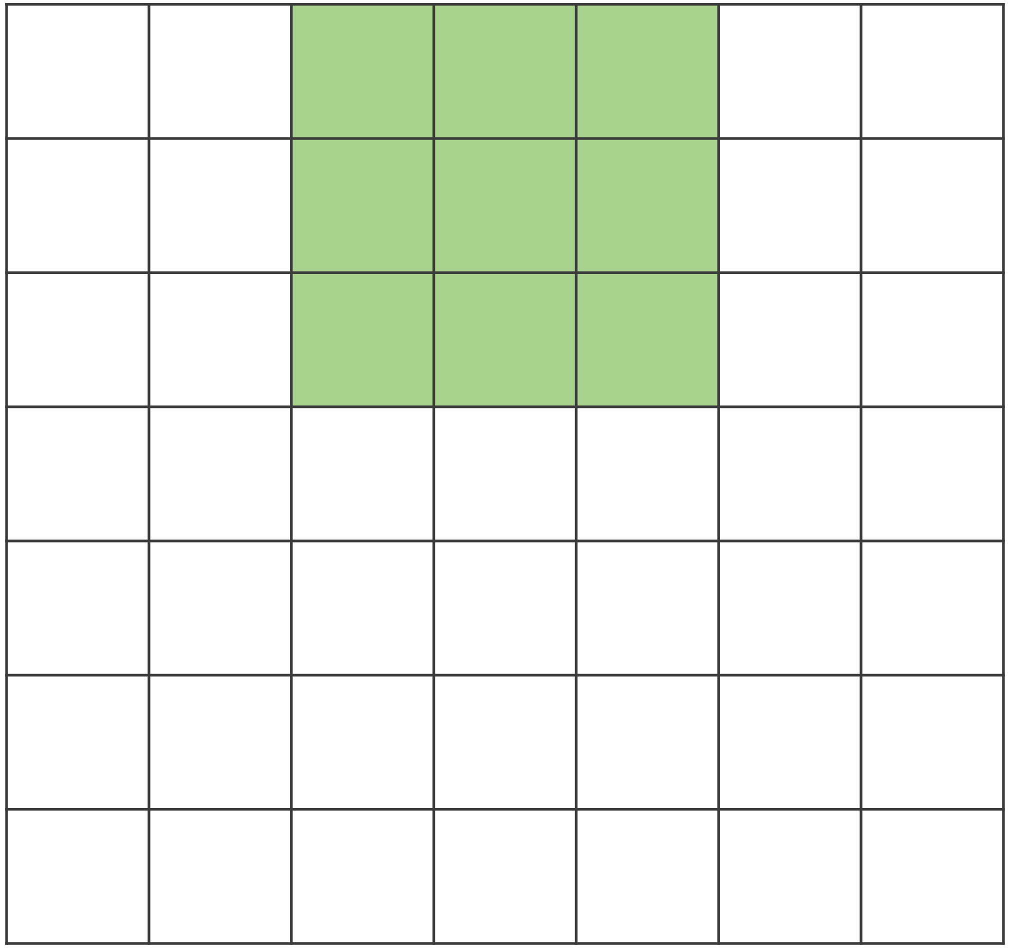
2D Convolution Layer

Stride = 1



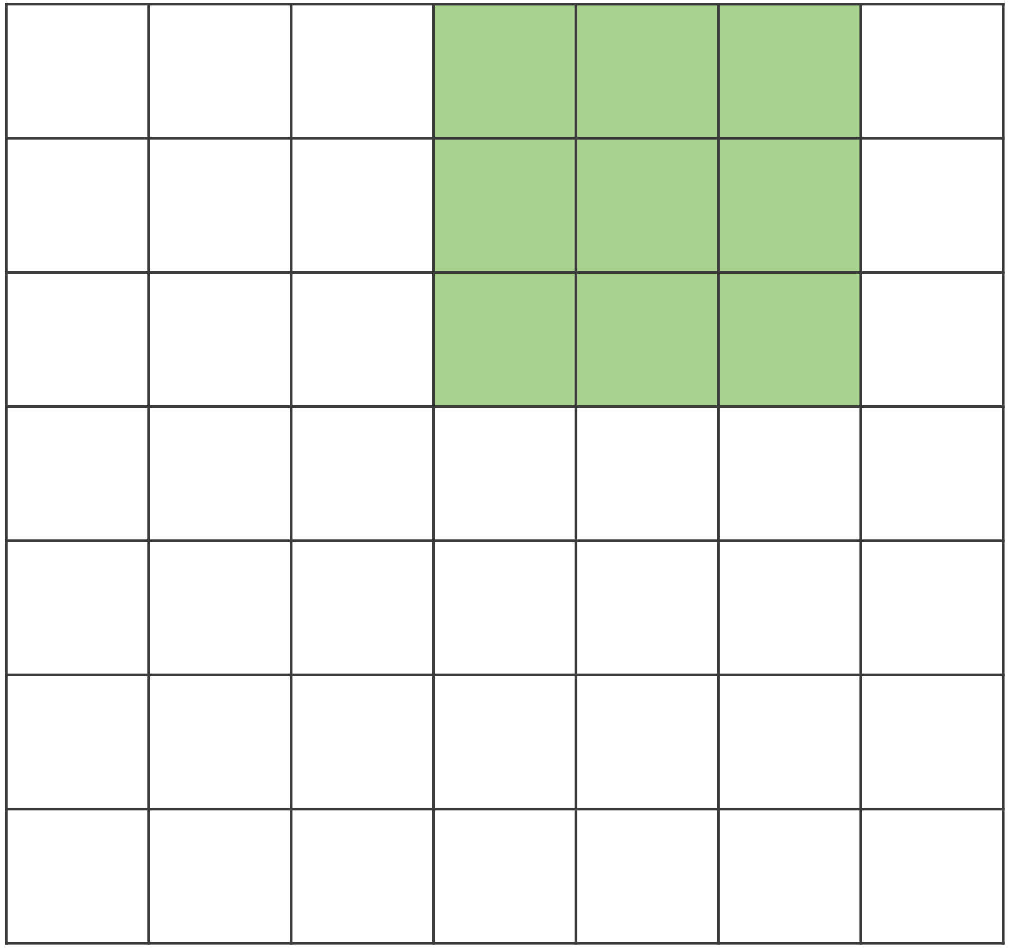
2D Convolution Layer

Stride = 1



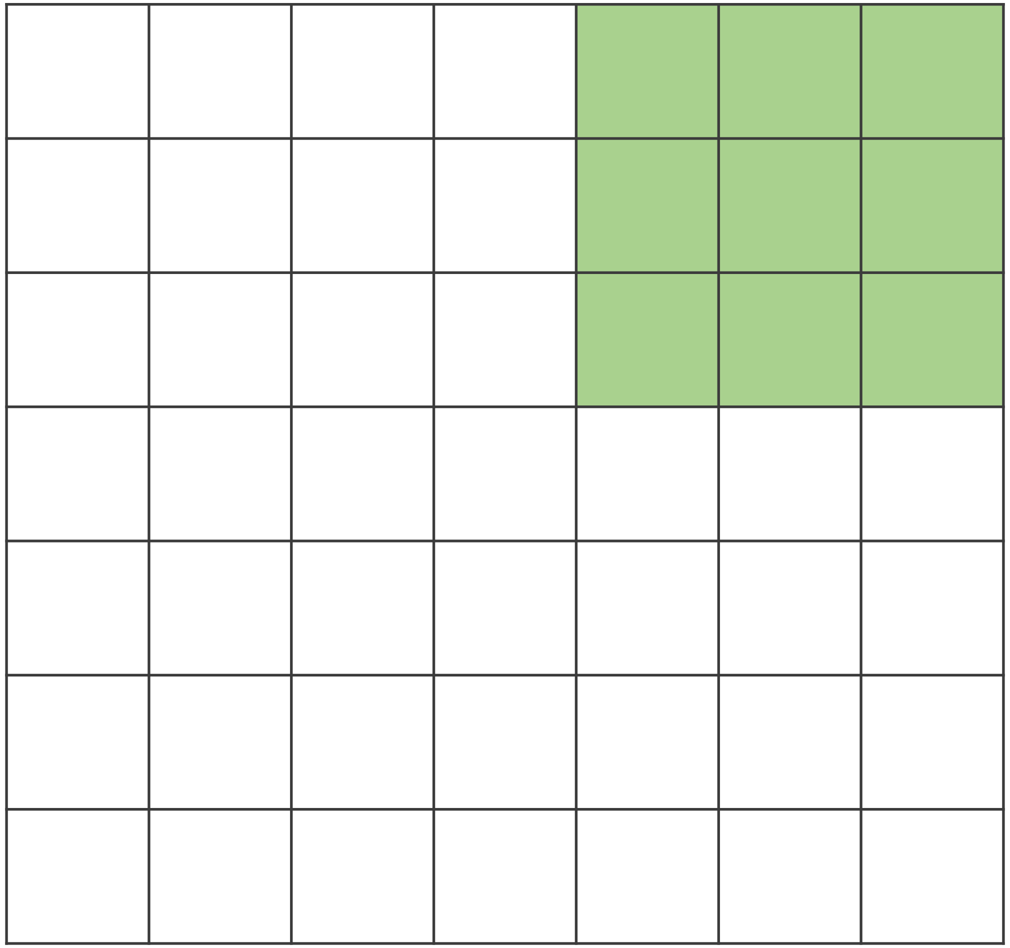
2D Convolution Layer

Stride = 1



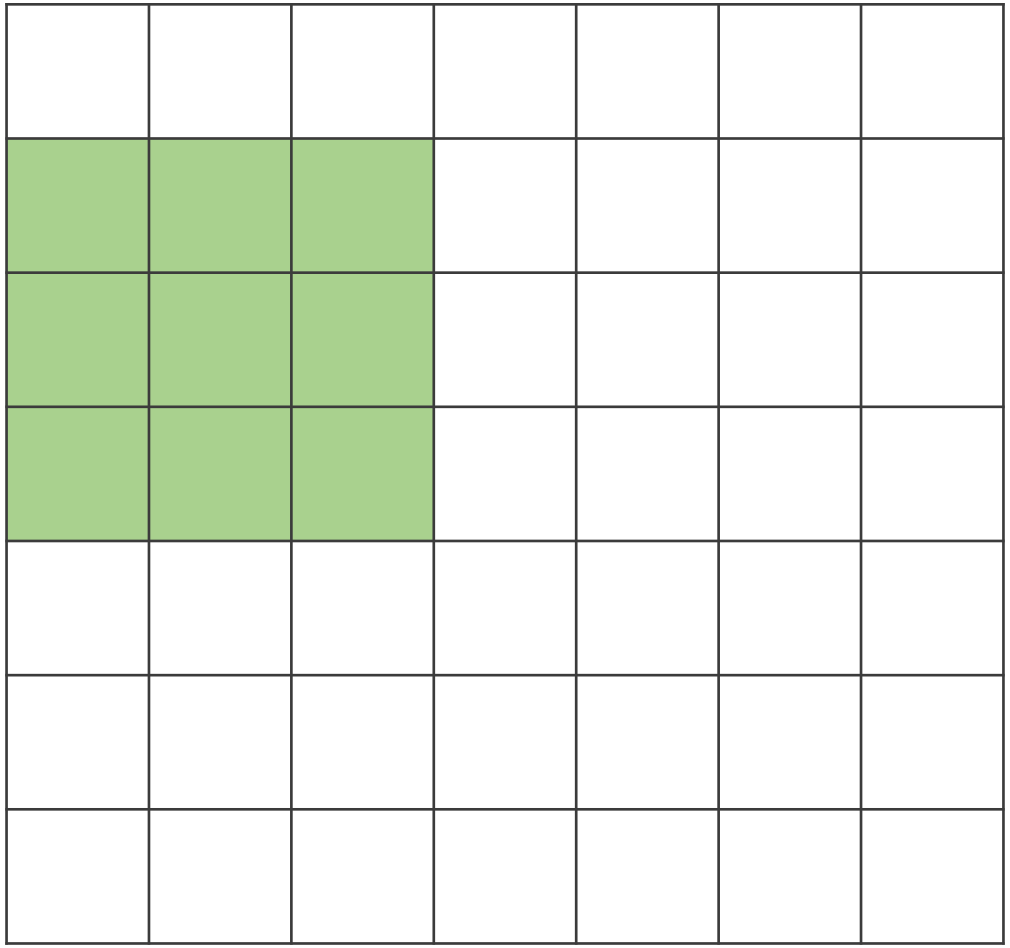
2D Convolution Layer

Stride = 1



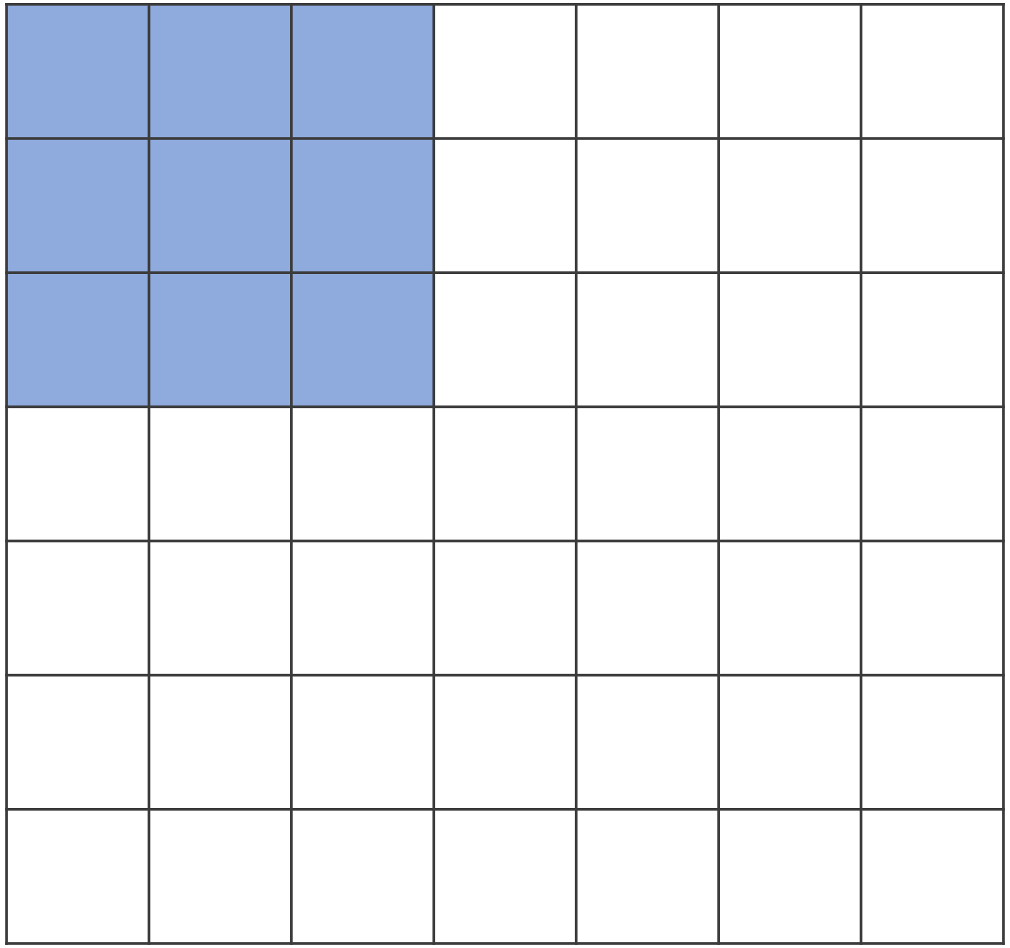
2D Convolution Layer

Stride = 1



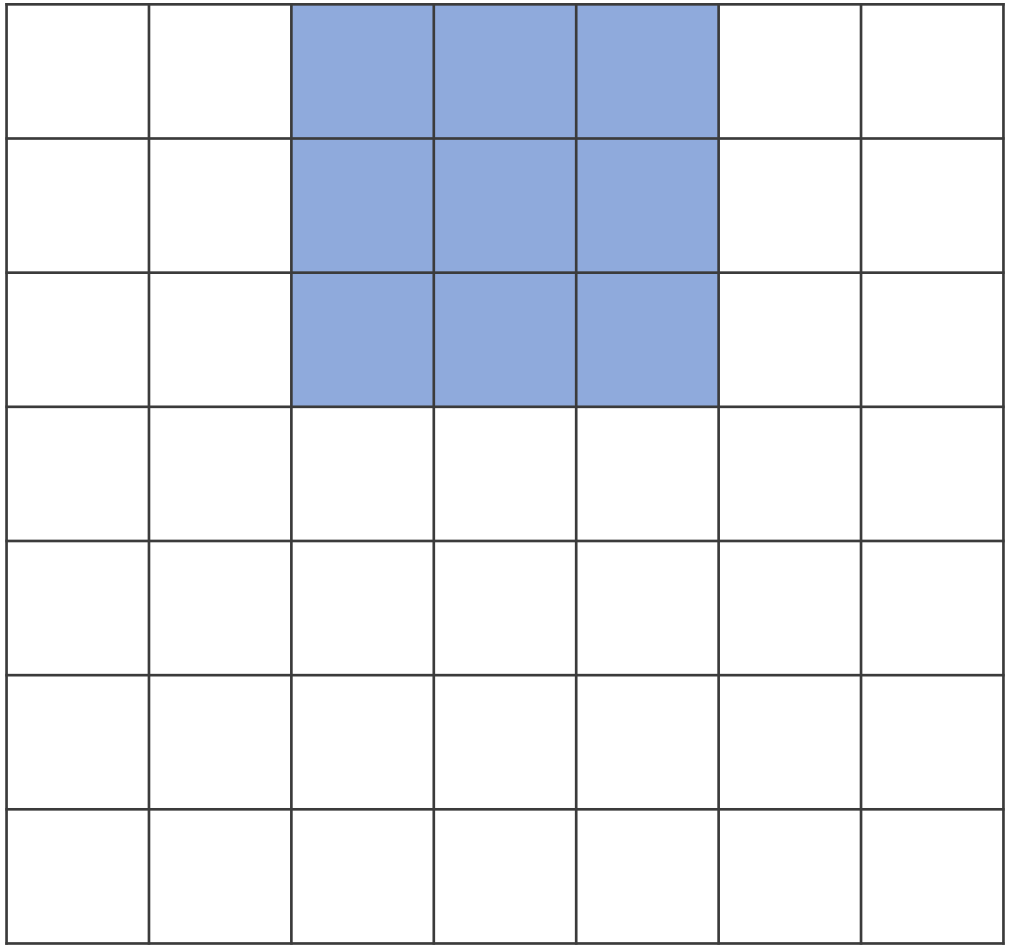
2D Convolution Layer

Stride = 2



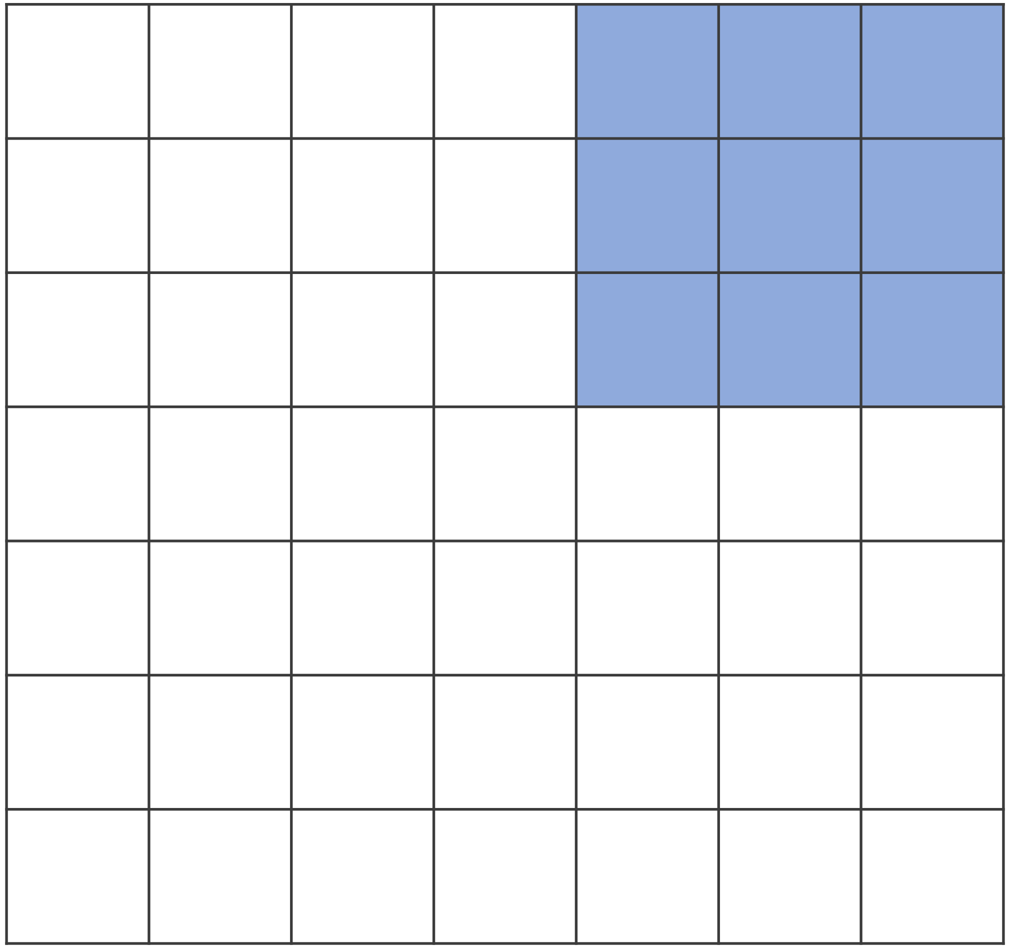
2D Convolution Layer

Stride = 2



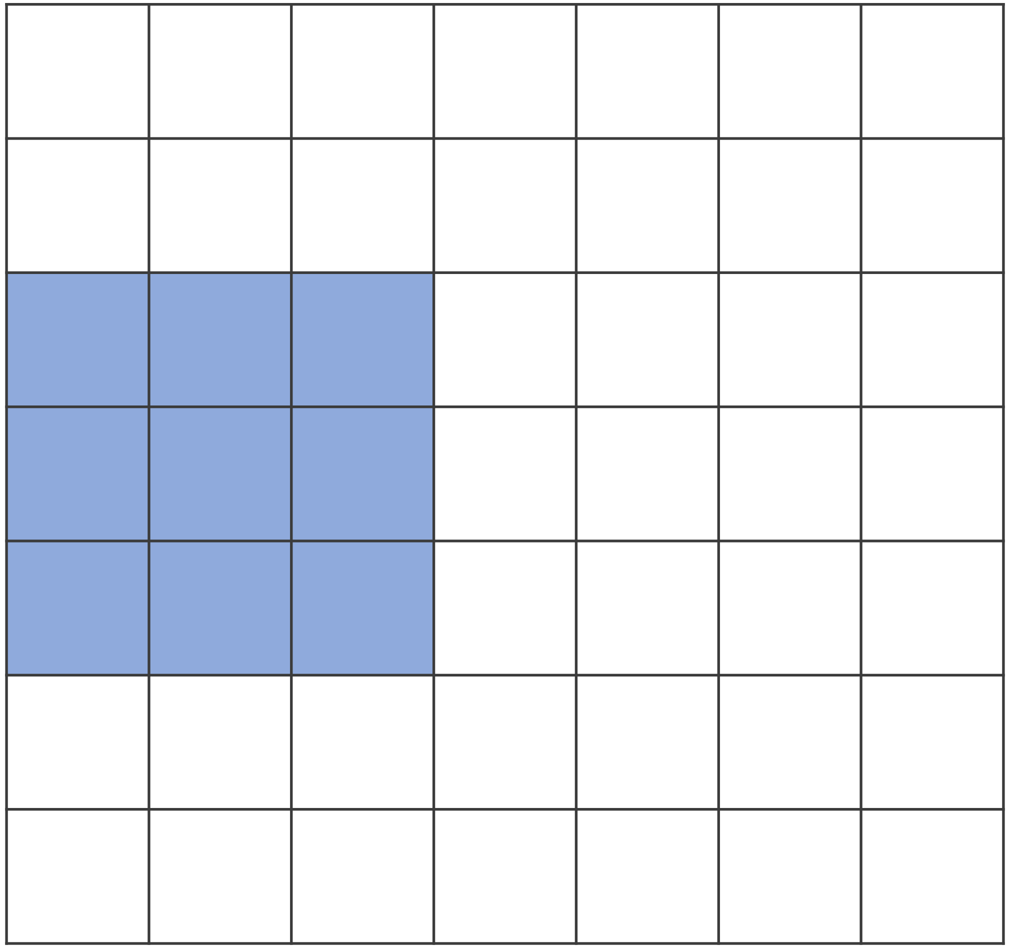
2D Convolution Layer

Stride = 2



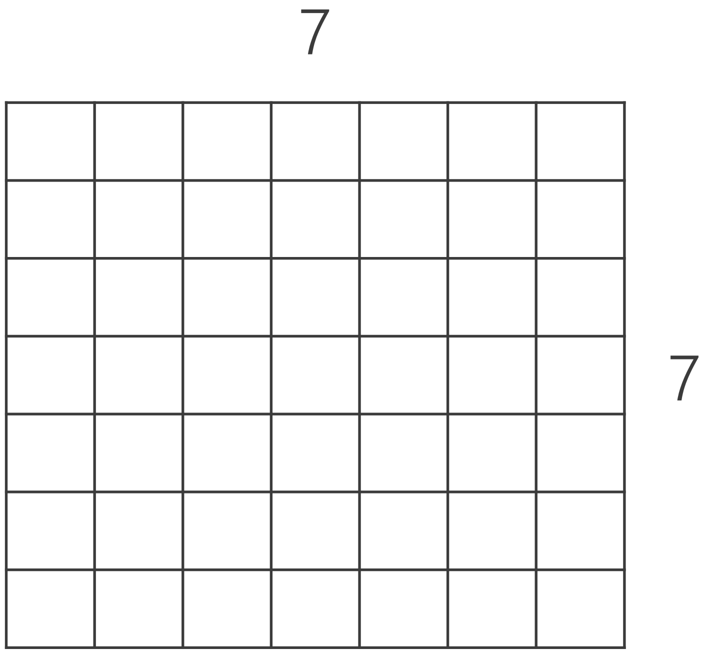
2D Convolution Layer

Stride = 2



2D Convolution Layer

Padding = ‘SAME’
Zero padding to make the output size same as the input



2D Convolution Layer

Padding = ‘SAME’

Zero padding to make the output size same as the input

9

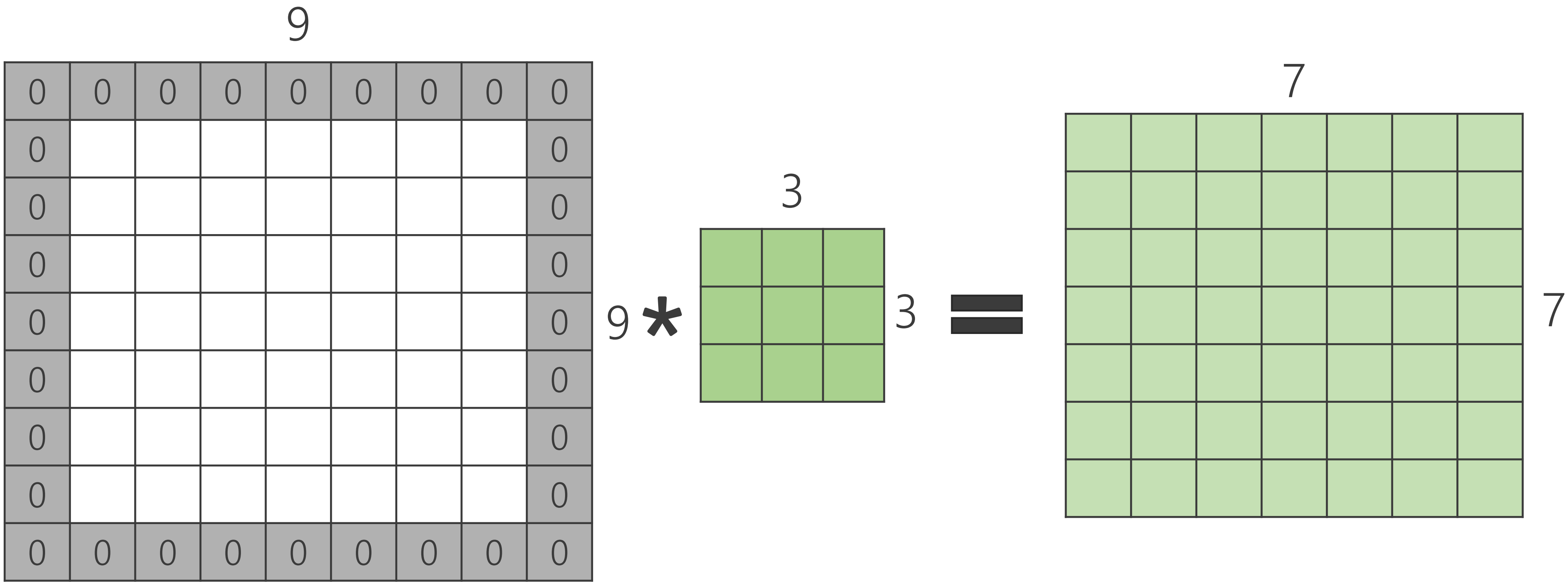
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

9

2D Convolution Layer

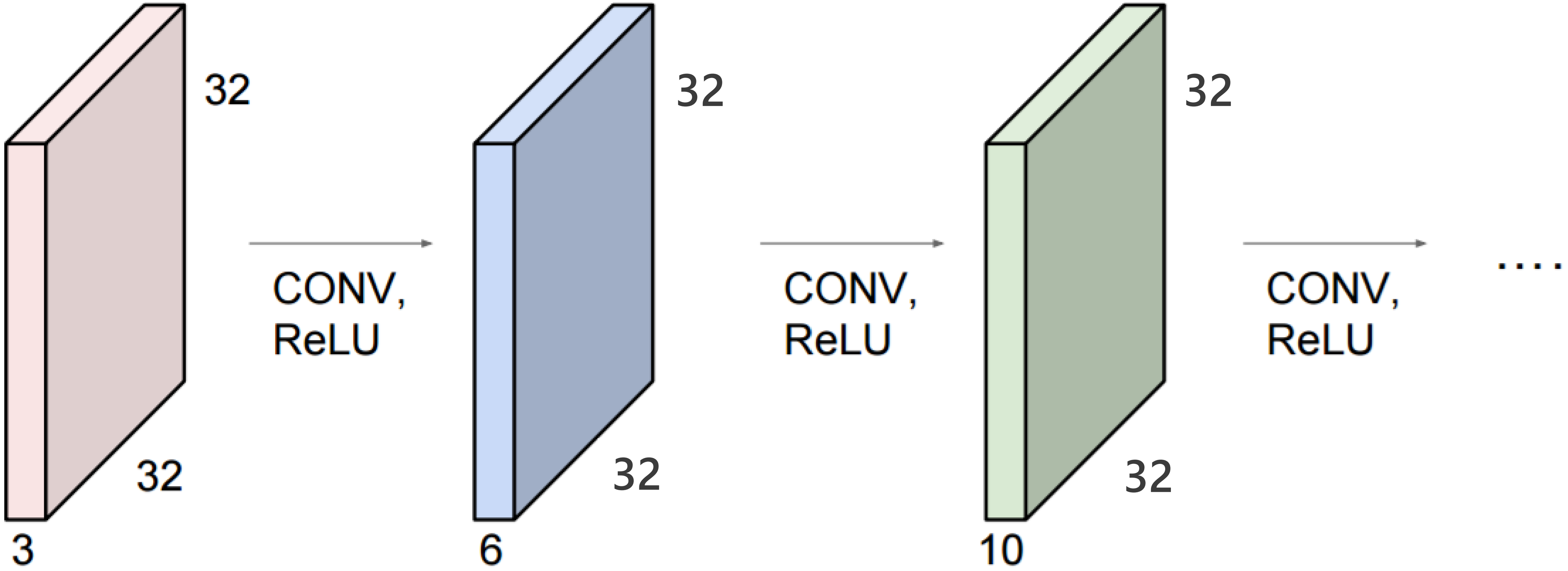
Padding = ‘SAME’

Zero padding to make the output size same as the input



2D Convolution Layer

With padding, we don't need to care about spatial size.



2D Convolution Layer in TF

- Convolution operation

tf.nn.conv2d

```
conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=None,
    data_format=None,
    name=None
)
```

Input shape = [batch, in_height, in_width, in_channels].

filter shape = [filter_height, filter_width, in_channels, out_channels]

Strides = [1, vertical_stride, horizontal_stride, 1].

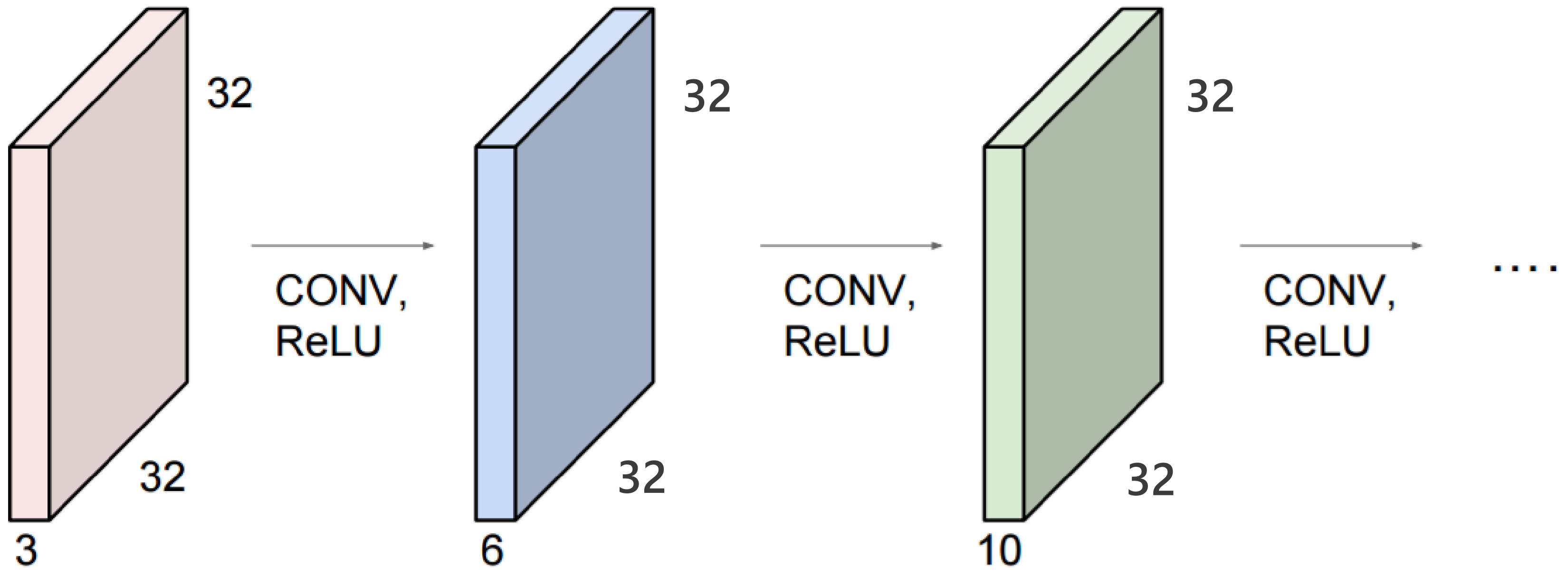
Padding = {'SAME', 'VALID'}

2D Convolution Layer in TF

```
def Conv2D(input, kernel_shape, strides, padding, name='Conv2d'):
    '''
    Convolutional layer.
    '''
    with tf.variable_scope(name):
        W = tf.get_variable("W", kernel_shape,
            initializer=tf.contrib.layers.xavier_initializer(uniform=False))
        b = tf.get_variable("b", (kernel_shape[-1]),
            initializer=tf.constant_initializer(value=0.0))
    return tf.nn.conv2d(input, W, strides, padding) + b
```

- Convolutional Layer
  - Create filter weight `W` and bias `b`
  - Define `tf.nn.conv2d` operation between input and weights

2D Convolution Layer in TF



- Usage

```
h = Conv2D(x, [5,5,3,6], [1,1,1,1], 'SAME', 'conv1')
h = tf.nn.relu(h)
h = Conv2D(h, [5,5,6,10], [1,1,1,1], 'SAME', 'conv2')
h = tf.nn.relu(h)
…
```

2D Pooling Layer

- Spatially subsample feature (activation)
  - Max pooling
    - Kernel size = 2, Stride = 2, padding='SAME'

7	2	-4	2
6	-7	9	3
3	-3	-4	2
-5	5	6	10

7	

2D Pooling Layer

- Spatially subsample feature (activation)
  - Max pooling
    - Kernel size =2, Stride = 2, padding='SAME'

7	2	-4	2
6	-7	9	3
3	-3	-4	2
-5	5	6	10

7	9

2D Pooling Layer

- Spatially subsample feature (activation)
  - Max pooling
    - Kernel size =2, Stride = 2, padding='SAME'

7	2	-4	2
6	-7	9	3
3	-3	-4	2
-5	5	6	10

7	9
5	

2D Pooling Layer

- Spatially subsample feature (activation)
  - Max pooling
    - Kernel size =2, Stride = 2, padding='SAME'

7	2	-4	2
6	-7	9	3
3	-3	-4	2
-5	5	6	10

7	9
5	10

2D Pooling Layer

- Spatially subsample feature (activation)
  - Max pooling
  - Average pooling
  - Stride convolution

- Pooing in TensorFlow
  - `tf.nn.max_pool`
  - `tf.nn.avg_pool`
  - Strided convolution
    - `Conv2D(x, [5,5,3,6], [1,2,2,1], 'SAME', 'conv1')`

2D Pooling Layer

tf.nn.max_pool

```
max_pool(
    value,
    ksize,
    strides,
    padding,
    data_format='NHWC',
    name=None
)
```

value shape = [batch, height, width, channels]

ksize (kernel size) shape = [1, kernel_height, kernel_width, 1]

strides = [1, vertical_stride, vertical_stride, 1]

padding: {'VALID', 'SAME'}

- Usage

```
h = tf.nn.max_pool(h, [1,2,2,1], [1,2,2,1], 'SAME')
```

Example: Digit recognition

- Problem
  - Given a 2D image, recognize digit

- Dataset: MNIST hand written digit dataset
  - $x =$ 2D image with shape [28, 28, 1]
  - $y =$ label

Data Loader for MNIST

```
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./data/", one_hot=True)
```

- TensorFlow provide data loader for MNIST ☺
  - Automatically download and extract MNIST data
  - one_hot vector: like [0,0,0,0,1,0,0,0,0,0]

Build TF graph (CNN)

```
###### Build graph ######################################################

# Place holders
x = tf.placeholder(tf.float32, [None,28,28,1]) # mnist data image of shape [28,28,1]
y = tf.placeholder(tf.float32, [None,10]) # 0-9 digits recognition => 10 classes
```

- Input is now 4D tensor
  - Image with shape 28x28x1
  - The number of channels is 1 (grayscale image)

Build TF graph (CNN)

```
###### Build graph ######################################################

# Place holders
x = tf.placeholder(tf.float32, [None,28,28,1]) # mnist data image of shape [28,28,1]
y = tf.placeholder(tf.float32, [None,10]) # 0-9 digits recognition => 10 classes

# Construct CNN
h = Conv2D(x, [3,3,1,4], [1,1,1,1], 'SAME', 'conv1')  # shape: [Batch,28,28,4]
h = tf.nn.relu(h)
h = tf.nn.max_pool(h, [1,2,2,1], [1,2,2,1], 'SAME')   # shape: [Batch,14,14,4]

h = Conv2D(h, [3,3,4,8], [1,1,1,1], 'SAME', 'conv2')  # shape: [Batch,14,14,8]
h = tf.nn.relu(h)
h = tf.nn.max_pool(h, [1,2,2,1], [1,2,2,1], 'SAME')   # shape: [Batch,7,7,8]
```

- Stacking some convolutional layers
  - Conv -> ReLU -> Pooling
  - Consider change of tensor shape !

Build TF graph (CNN)

```
###### Build graph ######################################################

# Place holders
x = tf.placeholder(tf.float32, [None,28,28,1]) # mnist data image of shape [28,28,1]
y = tf.placeholder(tf.float32, [None,10]) # 0-9 digits recognition => 10 classes

# Construct CNN
h = Conv2D(x, [3,3,1,4], [1,1,1,1], 'SAME', 'conv1')  # shape: [Batch,28,28,4]
h = tf.nn.relu(h)
h = tf.nn.max_pool(h, [1,2,2,1], [1,2,2,1], 'SAME')   # shape: [Batch,14,14,4]

h = Conv2D(h, [3,3,4,8], [1,1,1,1], 'SAME', 'conv2')  # shape: [Batch,14,14,8]
h = tf.nn.relu(h)
h = tf.nn.max_pool(h, [1,2,2,1], [1,2,2,1], 'SAME')   # shape: [Batch,7,7,8]

h = tf.reshape(h, [-1,7*7*8]) # flatten [Batch,7,7,8] -> [Batch,7*7*8]
logit = Dense(h, [7*7*8,10], 'fc1')
```

- Place fully-connected layer
  - Flatten 4D tensor to 2D Matrix using `tf.reshape` operation
  - Connect fully-connected (Dense) layer

Build TF graph (CNN)

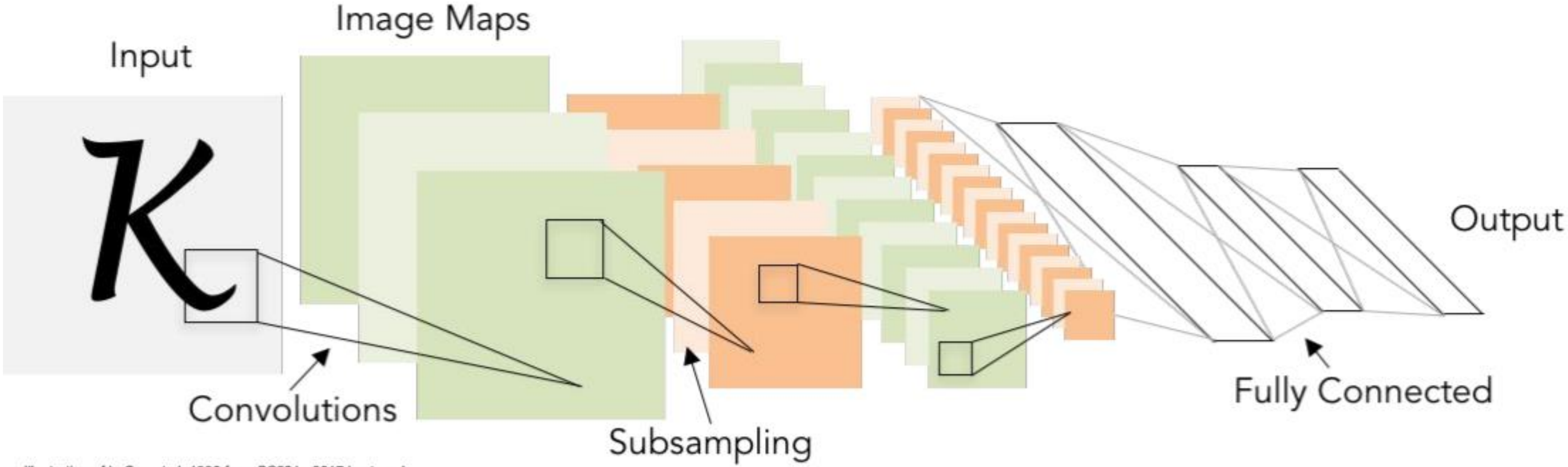


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Build TF graph (CNN)

```
…

h = tf.reshape(h, [-1,7*7*8]) # flatten [Batch,7,7,8] -> [Batch,7*7*8]
logit = Dense(h, [7*7*8,10], 'fc1')

pred = tf.nn.softmax(logit) # Softmax

# Directly compute loss from logit (to ensure stability and avoid overflow)
cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=logit, labels=y))

# Define optimizer and train_op
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

- Remaining parts are same as softmax regression

Training Loop

```
# Open a Session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):

            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_xs = np.reshape(batch_xs, [batch_size,28,28,1])

            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, cost], feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))
```

Training Loop

```python
# Open a Session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):

            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_xs = np.reshape(batch_xs, [batch_size,28,28,1])

            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, cost], feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))
```

Testing

```
# Test model
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy:", accuracy.eval({x: np.reshape(mnist.test.images, [-1,28,28,1]),
                                   y: mnist.test.labels}))
```

- Argmax for prediction
  - choose one class with maximum probability
- Accuracy is the percentage of prediction that is correct

Result

```
Epoch: 0001 cost= 0.210149454
Epoch: 0002 cost= 0.088126932
Epoch: 0003 cost= 0.072253242
Epoch: 0004 cost= 0.061968127
Epoch: 0005 cost= 0.054706751
Optimization Finished!
Accuracy: 0.9415
```

Advanced, Recent, and Practical techniques for Deep Convolutional Neural Network:

Train deeper CNN faster

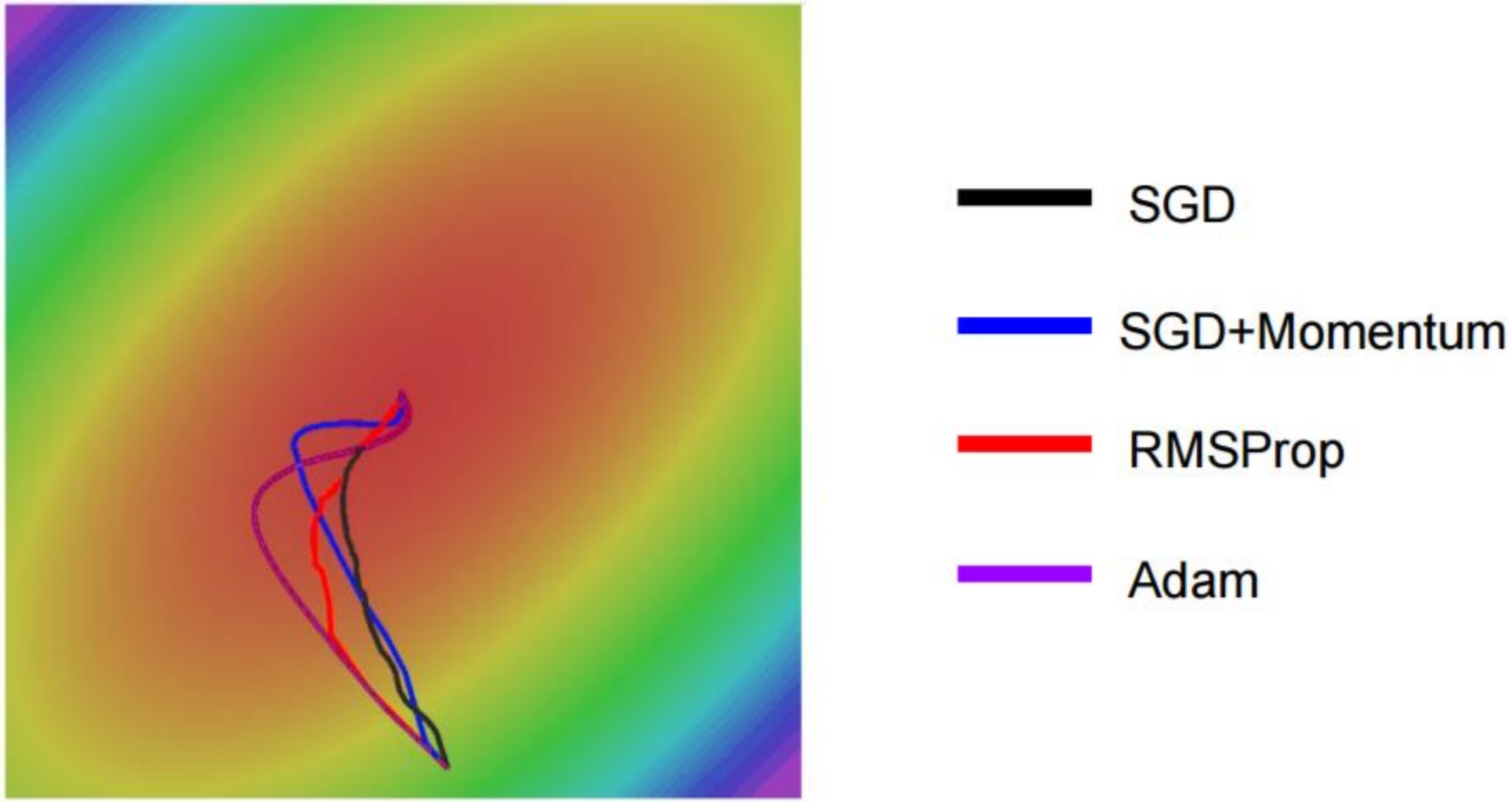
- Use more efficient Optimizer
  - Adam
- Do Data preprocessing
- Batch normalization
- Skip connection
  - Residual Network

Use more efficient Optimizer

- There are a lot of optimization methods that is better than simple SGD
- To use other optimizer, simply replace `train_op` with
  - SGD + momentum
    ```
    tf.train.MomentumOptimizer(learning_rate, momentum).minimize(cost)
    ```
  - RMSProp [Hinton]
    ```
    tf.train.RMSPropOptimizer(learning_rate).minimize(cost)
    ```
  - Adam [Kingma, 2014]
    ```
    tf.train.AdamOptimizer(learning_rate).minimize(cost)
    ```

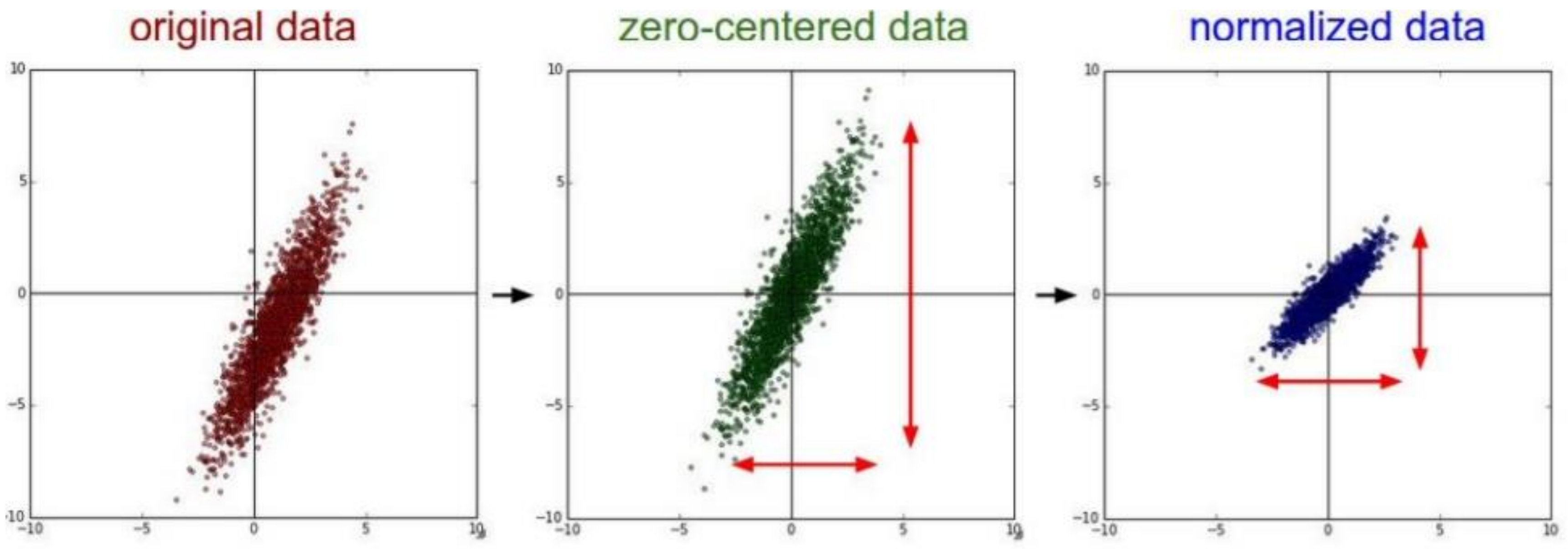
Use more efficient Optimizer

- Adam is a good default choice in most cases



Do data preprocessing

- Especially, the model is simple (e.g. linear)

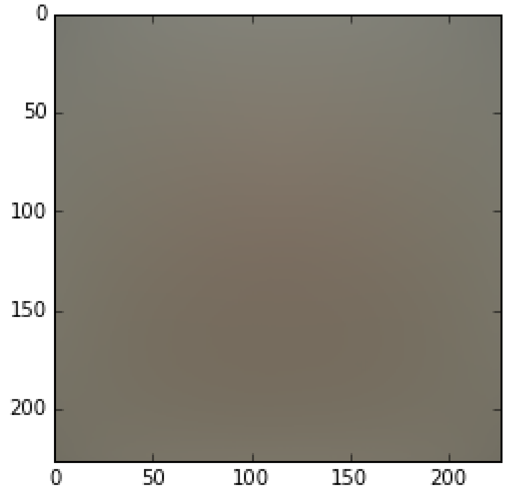



```
# Normalize features (zero mean, unit variance)
X_train = (X_train - np.mean(X_train, axis=0)) / np.std(X_train, axis=0)
X_test = (X_test - np.mean(X_test, axis=0)) / np.std(X_test, axis=0)
```

Do data preprocessing

- For image data, zero center the image
  - Option 1)
    Subtract mean image
  - Option 2)
    Subtract mean RGB values

Mean image for ImageNet data

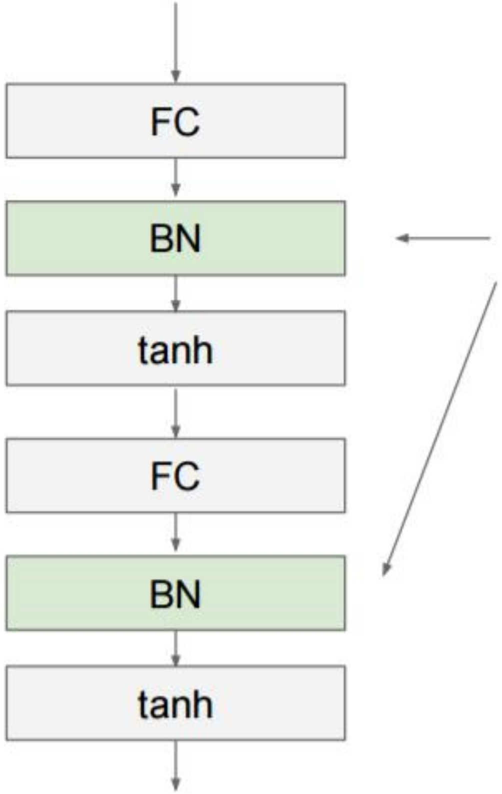


Mean RGB value for ImageNet data

= [123.68, 116.78, 0103.94]

Batch Normalization [Ioffe and Szegedy, 2015]

- Normalize activation map using batch statistics (mean, variance)



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

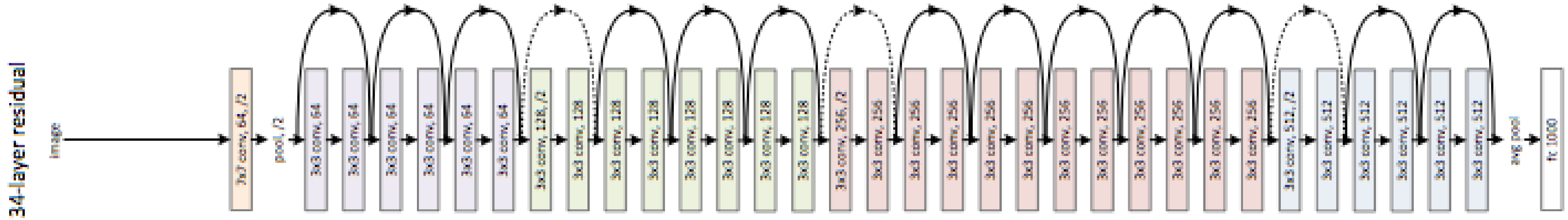
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization [Ioffe and Szegedy, 2015]

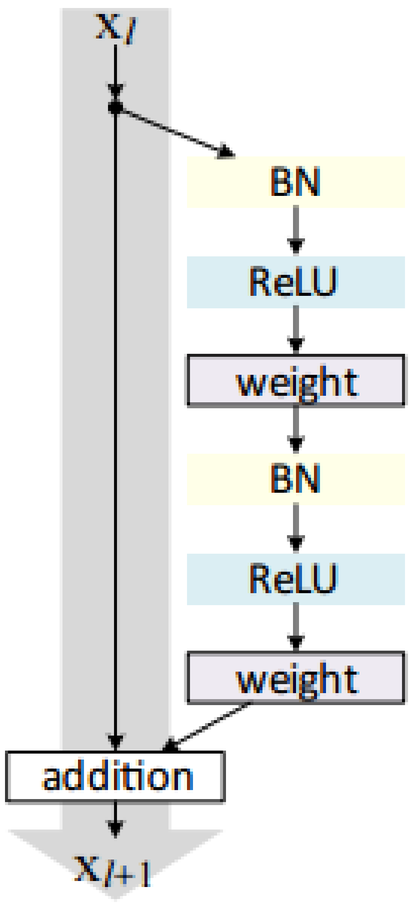
```
def BatchNorm(input, is_train, decay=0.999, name='BatchNorm'):
    from tensorflow.python.training import moving_averages
    from tensorflow.python.ops import control_flow_ops

    axis = list(range(len(input.get_shape()) - 1))
    fdim = input.get_shape()[-1:]

    with tf.variable_scope(name):
        beta = tf.get_variable('beta', fdim, initializer=tf.constant_initializer(value=0.0))
        gamma = tf.get_variable('gamma', fdim, initializer=tf.constant_initializer(value=1.0))
        moving_mean = tf.get_variable('moving_mean', fdim, initializer=tf.constant_initializer(value=0.0), trainable=False)
        moving_variance = tf.get_variable('moving_variance', fdim, initializer=tf.constant_initializer(value=0.0), trainable=False)

        def mean_var_with_update():
            batch_mean, batch_variance = tf.nn.moments(input, axis)
            update_moving_mean = moving_averages.assign_moving_average(moving_mean, batch_mean, decay, zero_debias=True)
            update_moving_variance = moving_averages.assign_moving_average(moving_variance, batch_variance, decay, zero_debias=True)
            with tf.control_dependencies([update_moving_mean, update_moving_variance]):
                return tf.identity(batch_mean), tf.identity(batch_variance)

            mean, variance = control_flow_ops.cond(is_train, mean_var_with_update, lambda: (moving_mean, moving_variance))

        return tf.nn.batch_normalization(input, mean, variance, beta, gamma, 1e-3)
```

Skip Connection

Residual Network [He, 2016]



Skip Connection



- Residual Block in TensorFlow

```
s = x
x = BatchNorm(x, is_train, name='bn1’)
x = tf.nn.relu(x)
x = Conv2D(x, [3,3,256,256], [1,1,1,1], ‘SAME', name='conv1‘)
x = BatchNorm(x, is_train, name='bn2')
x = tf.nn.relu(x)
x = Conv2D(x, [3,3,256,256], [1,1,1,1], ‘SAME', name='conv2')
x += s
```

Practice: MNIST digit recognition

We've explained and provided TF codes
Now, build your own model for MNIST digit recognition.

Tune hyper parameters for the best model

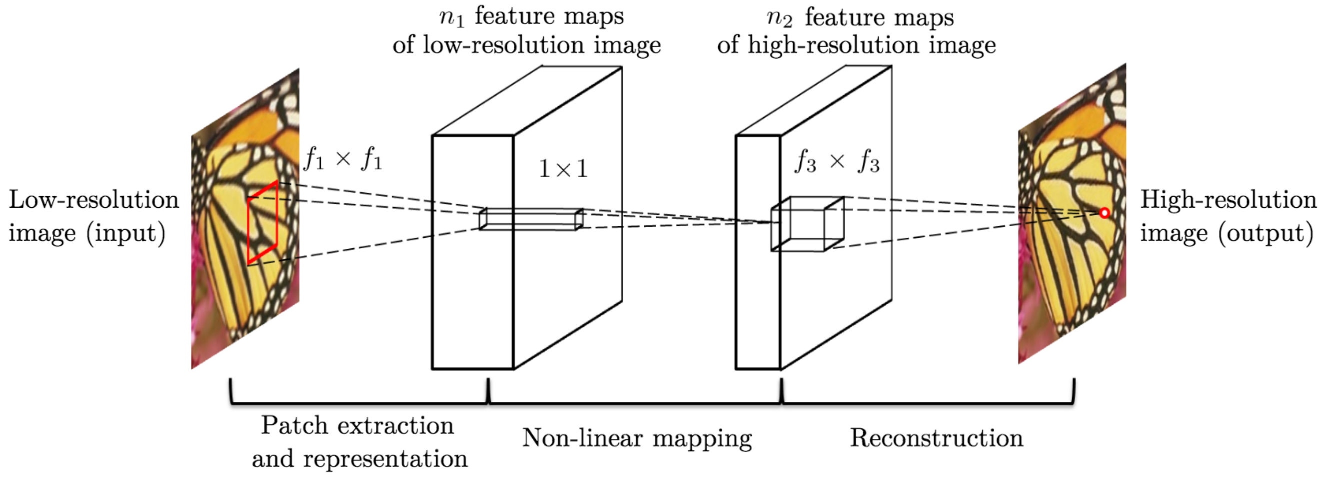
- Network structure
- Optimizer / Learning rate
- Filter size / Filter number
- Etc..

- **Consider training deep model using CPU can takes forever..**
    - **Design efficient network**

Example: Image Super Resolution (SRCNN)

- Problem
  - Given a low-res image, recover a high-res image
- Dataset
  - MNIST, again.

- SRCNN [Dong, 2014]



Example: Image Super Resolution (SRCNN)

```
# Place holders
H = tf.placeholder(tf.float32, [None,28,28,1]) # Input image
L = tf.image.resize_bicubic(H, [7,7]) # Downsample input inside graph
L = tf.image.resize_nearest_neighbor(L, [28,28]) # and upsample back
```

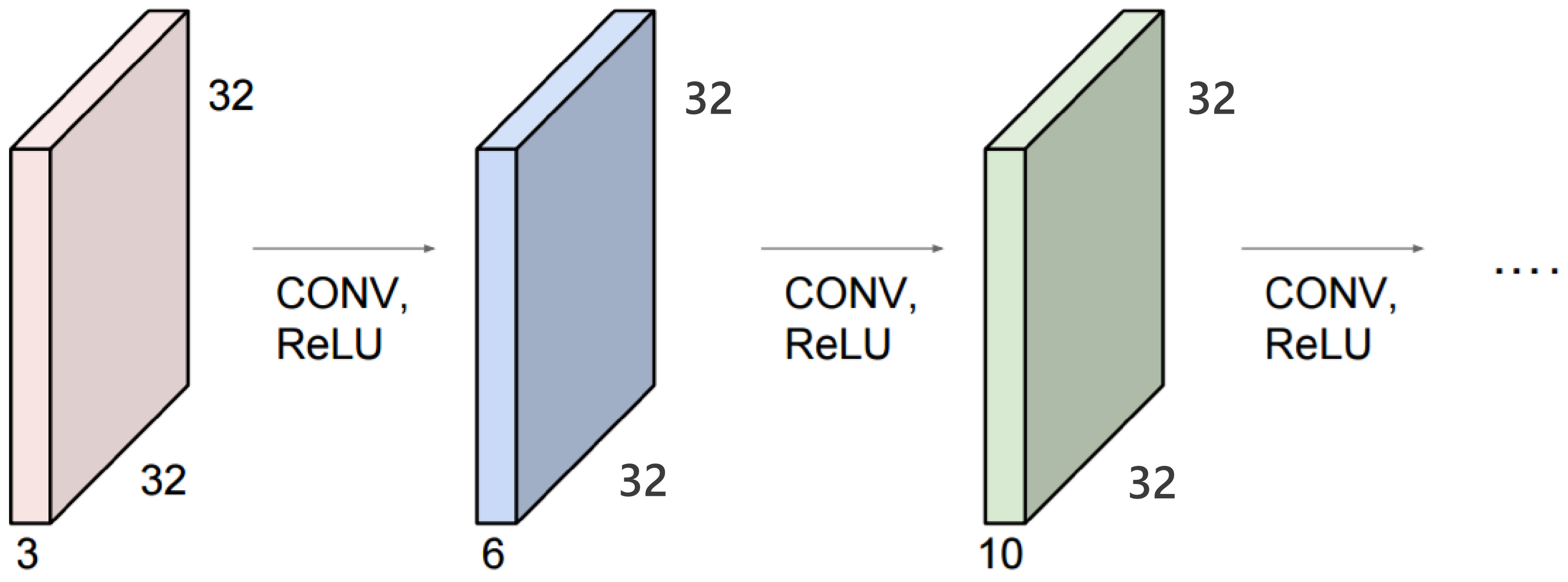
- Place holder for high-resolution input
  - No need for digit labels
  - High resolution image is labels
- Generate low-resolution image within TF graph
  - `tf.image.resize` functions

Example: Image Super Resolution (SRCNN)

```
# Place holders
H = tf.placeholder(tf.float32, [None,28,28,1]) # Input image
L = tf.image.resize_bicubic(H, [7,7]) # Downsample input inside graph
L = tf.image.resize_nearest_neighbor(L, [28,28]) # and upsample back

# Construct CNN
h = Conv2D(L, [3,3,1,8], [1,1,1,1], 'SAME', 'conv1') # shape: [Batch,28,28,16]
h = tf.nn.relu(h)

h = Conv2D(h, [3,3,8,8], [1,1,1,1], 'SAME', 'conv2') # shape: [Batch,28,28,16]
h = tf.nn.relu(h)

pred = Conv2D(h, [3,3,8,1], [1,1,1,1], 'SAME', 'conv3') # shape: [Batch,28,28,1]
```

- Stack some convolution layers
  - Prediction is also an image [28,28,1]

Example: Image Super Resolution (SRCNN)



Example: Image Super Resolution (SRCNN)

```
# Place holders
H = tf.placeholder(tf.float32, [None,28,28,1]) # Input image
L = tf.image.resize_bicubic(H, [7,7]) # Downsample input inside graph
L = tf.image.resize_nearest_neighbor(L, [28,28]) # and upsample back

# Construct CNN
h = Conv2D(L, [3,3,1,8], [1,1,1,1], 'SAME', 'conv1') # shape: [Batch,28,28,16]
h = tf.nn.relu(h)

h = Conv2D(h, [3,3,8,8], [1,1,1,1], 'SAME', 'conv2') # shape: [Batch,28,28,16]
h = tf.nn.relu(h)

pred = Conv2D(h, [3,3,8,1], [1,1,1,1], 'SAME', 'conv3') # shape: [Batch,28,28,1]

# L2 (Euclidean) distance as cost function
cost = tf.reduce_mean((pred - H)**2)

# Define optimizer and train_op
train_op = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

Results

High-res
Low-res
Pred

High-res
Low-res
Pred

Acknowledgement

- Stanford CS231n
  - http://cs231n.stanford.edu/
  - http://cs231n.github.io/

- Andrew Ng's ML course
  - https://www.coursera.org/learn/machine-learning

- 모두를 위한 머신러닝/딥러닝 강의
  - https://hunkim.github.io/ml/