CMPE 258, Deep Learning

Coding for CNN

April 3, 2018

DMH 149A

Taehee Jeong

Ph.D., Data Scientist

SJSU SAN JOSÉ STATE UNIVERSITY

Group Project Proposal

Title submission deadline: April 9th

- Project title
- List of Members
- Preferred presentation day: 4/12 or 4/24

Group Project Proposal

Content during proposal

- Justification for the project
- Background: any relevant previous work
- How to collect data set
- Which algorithms / platform will be used
- What is the role for each team member

Mid-term Exam_2

Start Morning on April 12th .

End the midnight on April 15th

Image classification using CNN

Assignment_5

Due April 8^{th}, 2018

Deadline for re-submitting is April 15^{th}, 2018

Grading policy:

The code is supposed to be executable without any extra effort and produce reasonable result within 50 minutes.

If the code cannot be executable with any error or taking more than 50 minutes, 50 points will be assigned.

If the code can be executable without any error within 50 minutes, score will be assigned as following formula.

Score = (10 – cost) * 10

Re-submitting is available until March 15^{th}, but 10 point will be deducted every re-submitting after March 8^{th}.

If extra effort is needed to get reasonable result (whatever it is), 5 to 10 points will be deducted.

You may use your trained weights and bias (transfer learning). In this case, please make sure to submit the trained weights and bias as one separate file (*para_yourFirstName_LastName.hdf5)*

HDF5

Hierarchical Data Format 5

Create dataset

```
In [38]: import h5py

In [39]: f = h5py.File(path+"weight_rma.hdf5", "w")

In [40]: dataset = f.create_dataset("W2", data=W2)
         dataset = f.create_dataset("b2", data=b2)
         dataset = f.create_dataset("W1", data=W1)
         dataset = f.create_dataset("b1", data=b1)

In [41]: f.close()
```

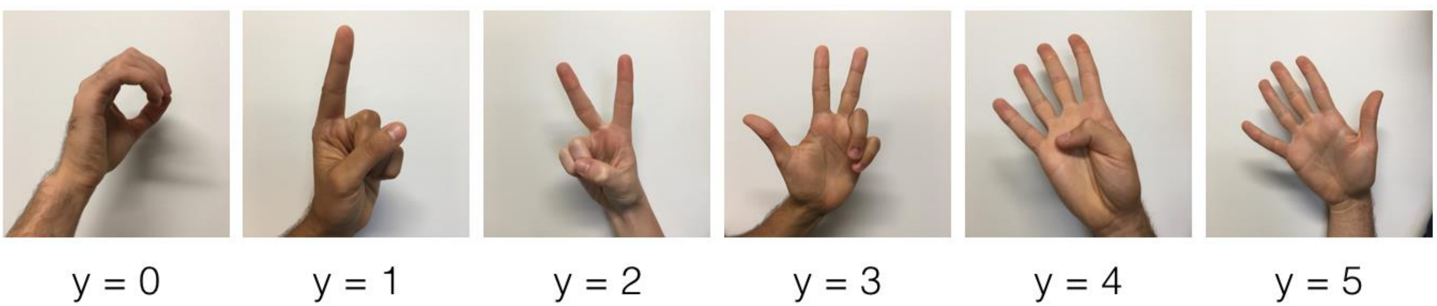
Load dataset

```
In [15]: dataset = h5py.File(path+'weight_rma.hdf5', "r")

In [16]: dataset.keys()

Out[16]: [u'W1', u'W2', u'b1', u'b2']

In [17]: W1 = np.array(dataset["W1"])
         W2 = np.array(dataset["W2"])
         b1 = np.array(dataset["b1"])
         b2 = np.array(dataset["b2"])

In [18]: print (W1.shape)
         print (W2.shape)
         print (b1.shape)
         print (b2.shape)

(40L, 2429L)
(4L, 40L)
(40L, 1L)
(4L, 1L)
```

Assignment_5

Data set :

SIGNS Dataset from Coursera (Deep Learning specialization)



Each picture is a RGB image with 64 by 64 pixels.
1020 pictures.

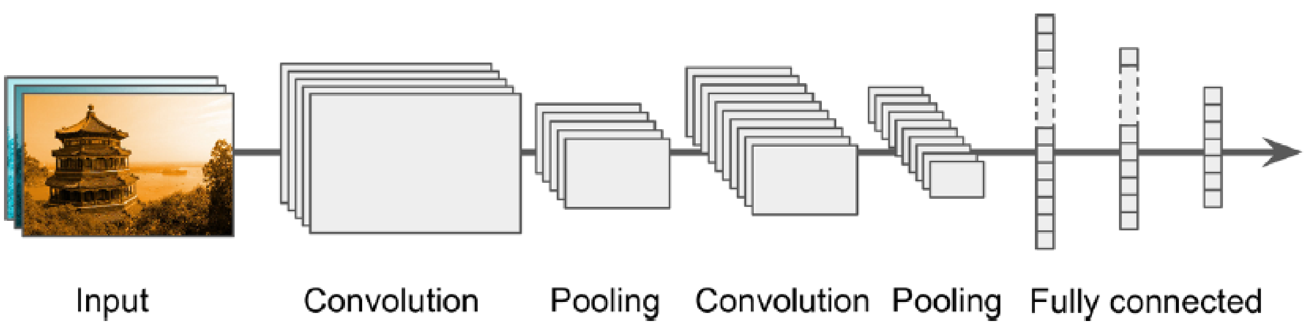
Image classification using CNN

CNN architecture

2 Convolution layers

2 Pooling layers

2 Fully connected layers

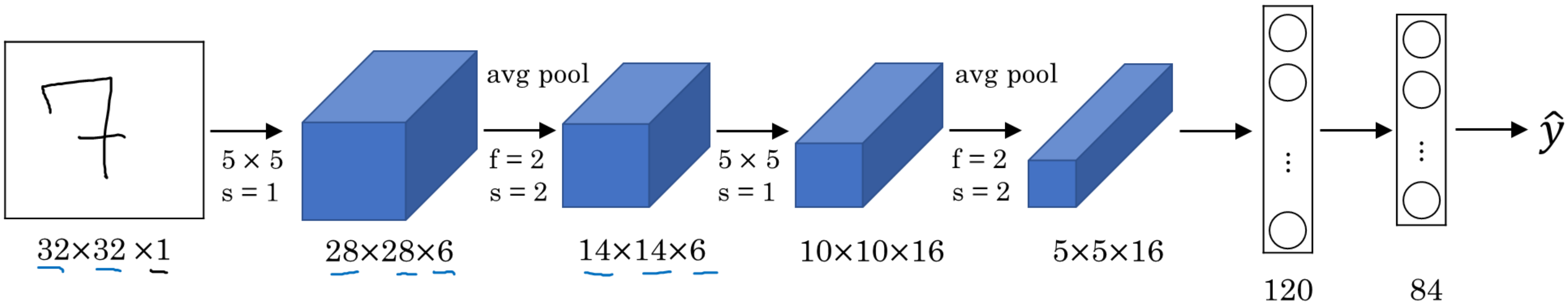


<Hands-on ML, Aurelien Geron>

One example of CNN model

Layer	Type	Size	Channels	Kernel size	Stride	Padding	Function
0	Input	64 x 64	3				
1	Convolution (C1)	32 x 32	8	4 x 4	2	1	ReLU
1	Pooling (P1)	28 x 28	8	5 x 5	1	0	max
2	Convolution (C2)	13 x 13	16	4 x 4	2	0	ReLU
2	Pooling (P2)	9 x 9	16	5 x 5	1	0	Avg
3	Flatten (F3)	1296					
4	Fully connected (F4)	108					ReLU
5	Fully connected (F5)	6					Sigmoid

LeNet-5



<Deep Learning, Andrew Ng>

LeCun et al., 1998. Gradient-based learning applied to document recognition

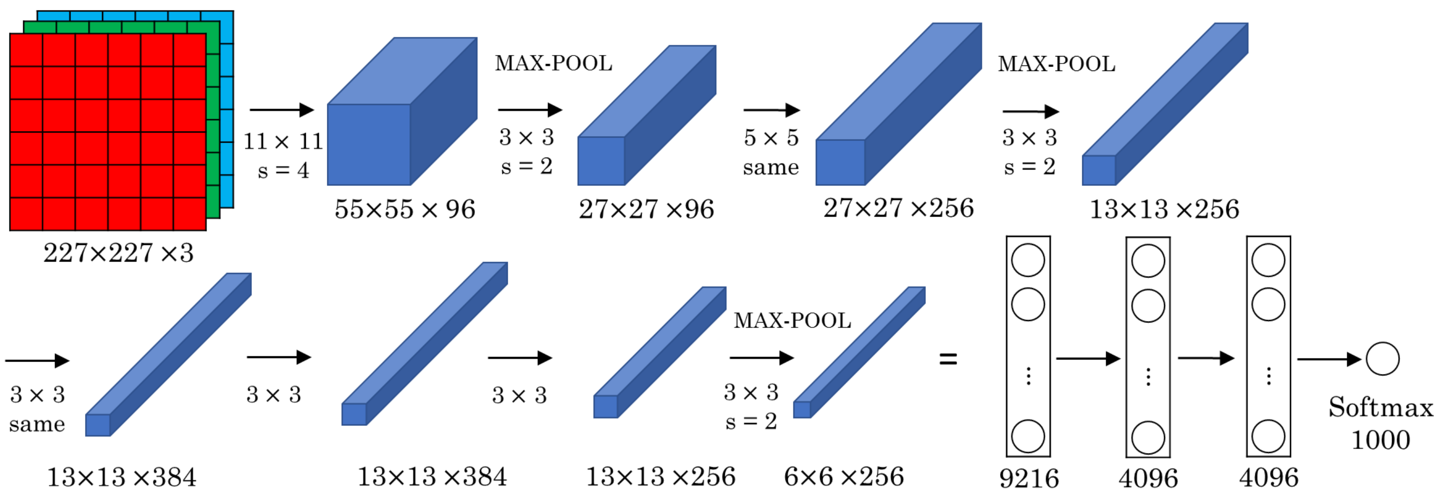
LeNet-5

created by Yann LeCun in 1998 and widely used for handwritten digit recognition (MNIST)

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

<Hands-on ML, Aurelien Geron>

AlexNet



<Deep Learning, Andrew Ng>

AlexNet

won the 2012 ImageNet ILSVRC challenge with 83% accuracy.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

<Hands-on ML, Aurelien Geron>

“ImageNet Classification with Deep Convolutional Neural Networks,” A. Krizhevsky et al. (2012)

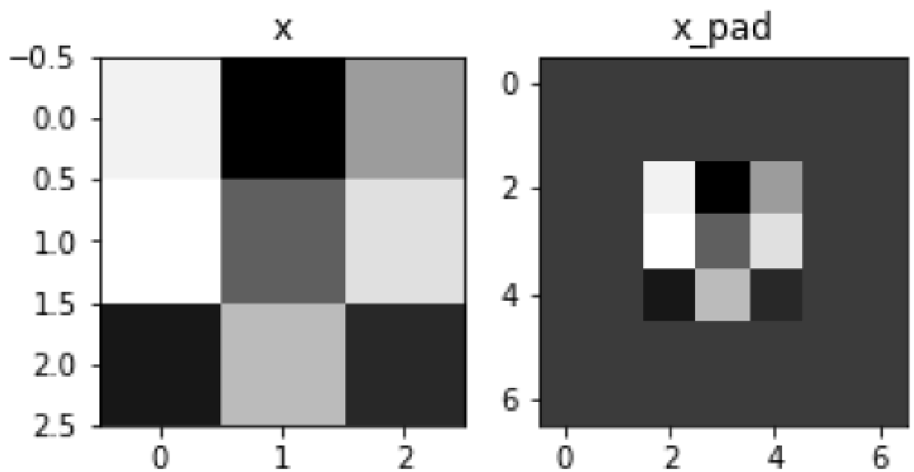
Functions for CNN

- Zero pad
- Convolution with single step
- Convolution forward (for all data)
- Pooling forward (max, average)
- Convolution backward (for all data)
- Pooling backward (max, average)

Zero pad

Input: (m, n_H, n_W, n_C)

Output: (m, n_H + 2*pad, n_W + 2*pad, n_C)



<Deep Learning, Andrew Ng>

Zero pad

np.pad

```
In [4]: a

Out[4]: array([[0, 1, 2],
               [3, 4, 5]])
```

```
In [7]: np.pad(a, (1,1), 'constant', constant_values=(0))

Out[7]: array([[0, 0, 0, 0, 0],
               [0, 0, 1, 2, 0],
               [0, 3, 4, 5, 0],
               [0, 0, 0, 0, 0]])
```

```
In [8]: np.pad(a, (2,2), 'constant', constant_values=(0))

Out[8]: array([[0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 1, 2, 0, 0],
               [0, 0, 3, 4, 5, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0]])
```

Convolution with single step

Input:

a_slice : slice of input data of shape (f, f, n_c)

W : Weight parameters contained in a window - matrix of shape (f, f, n_c)

b : Bias parameters contained in a window - matrix of shape (1, 1, 1)

Output

Z : a scalar value

Convolution with single step

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

W

1	0	0
0	-1	0
0	0	1

Size: 3 x 3

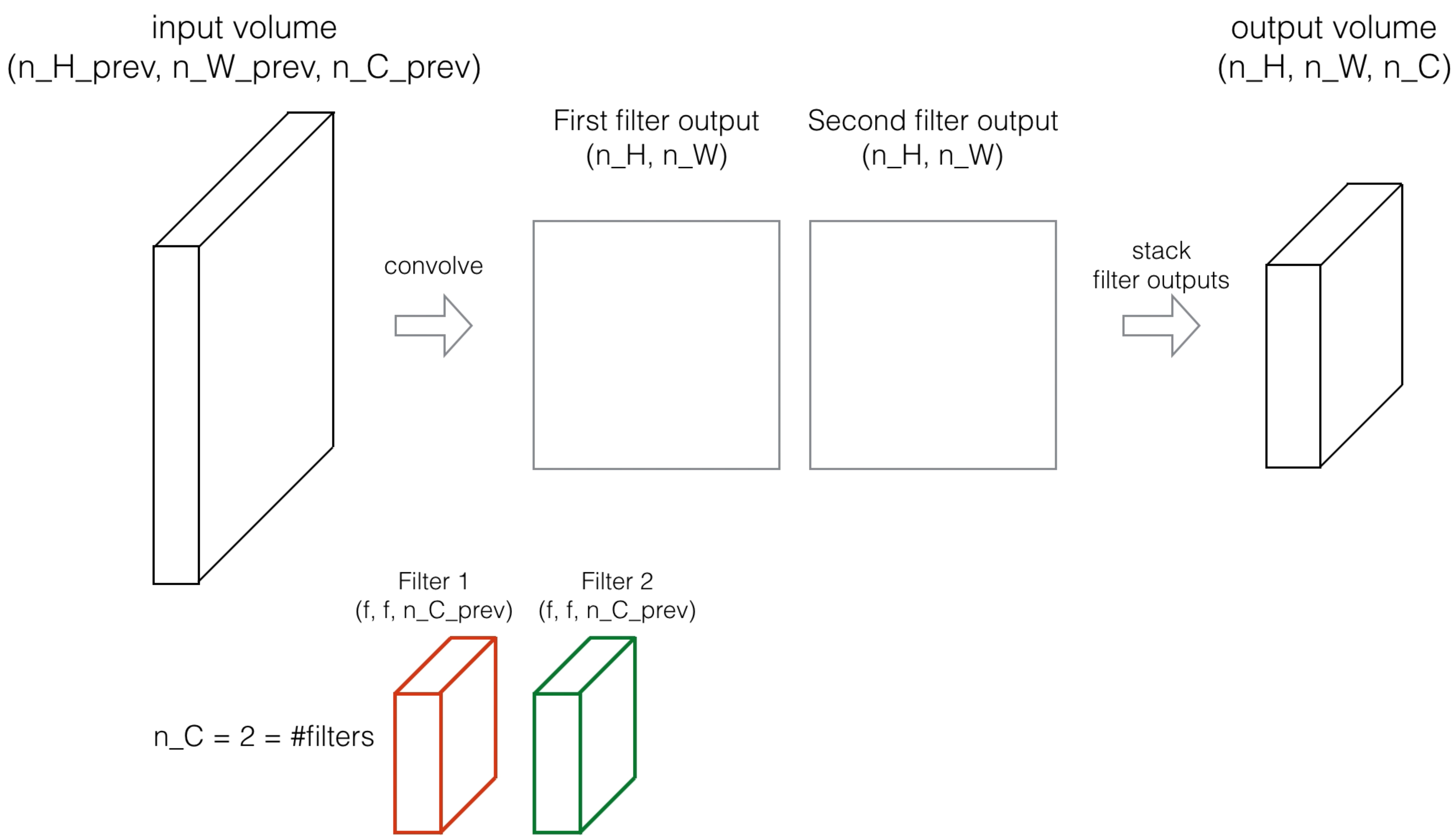
Z = W*a(Red)+W*a(Green)+W*a(Blue)+b

(0-155+157)+(0-166+168)+(0-162+164)+5

= 1

<image Convolution>
Machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

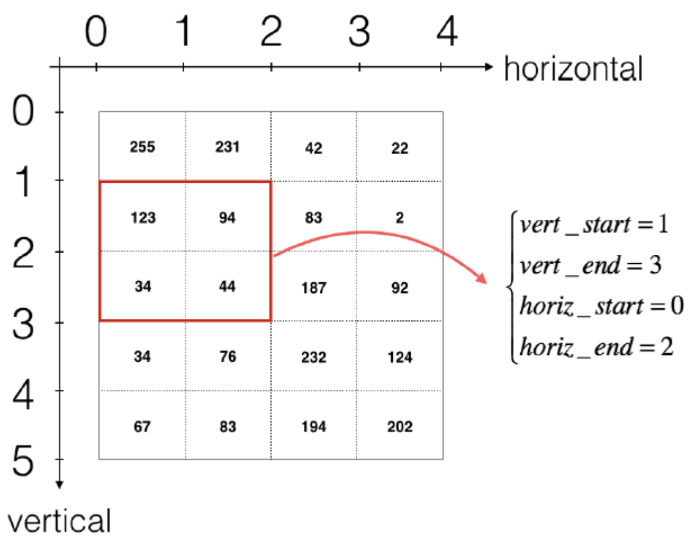
How do convolutions work?



<deep learning, Andrew Ng>

Convolution forward

```
a_slice = a [1:3,0:2,:]
```



<Deep Learning, Andrew Ng>

Summary of convolutions

- n x n image
- f x f filter
- padding p
- stride s

Output size:

$$\left\lfloor\frac{n + 2p - f}{s} + 1\right\rfloor \times \left\lfloor\frac{n + 2p - f}{s} + 1\right\rfloor$$

Summary of convolution

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

Input size: $n_H^{[l-1]} \times n_W^{[l-1]} \times \boxed{n_c^{[l-1]}}$

output size: $n_H^{[l]} \times n_W^{[l]} \times \boxed{n_c^{[l]}}$

$$n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

$$n_W^{[l]} = \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

Weights: $f^{[l]} \times f^{[l]} \times \boxed{n_c^{[l-1]}} \times \boxed{n_c^{[l]}}$

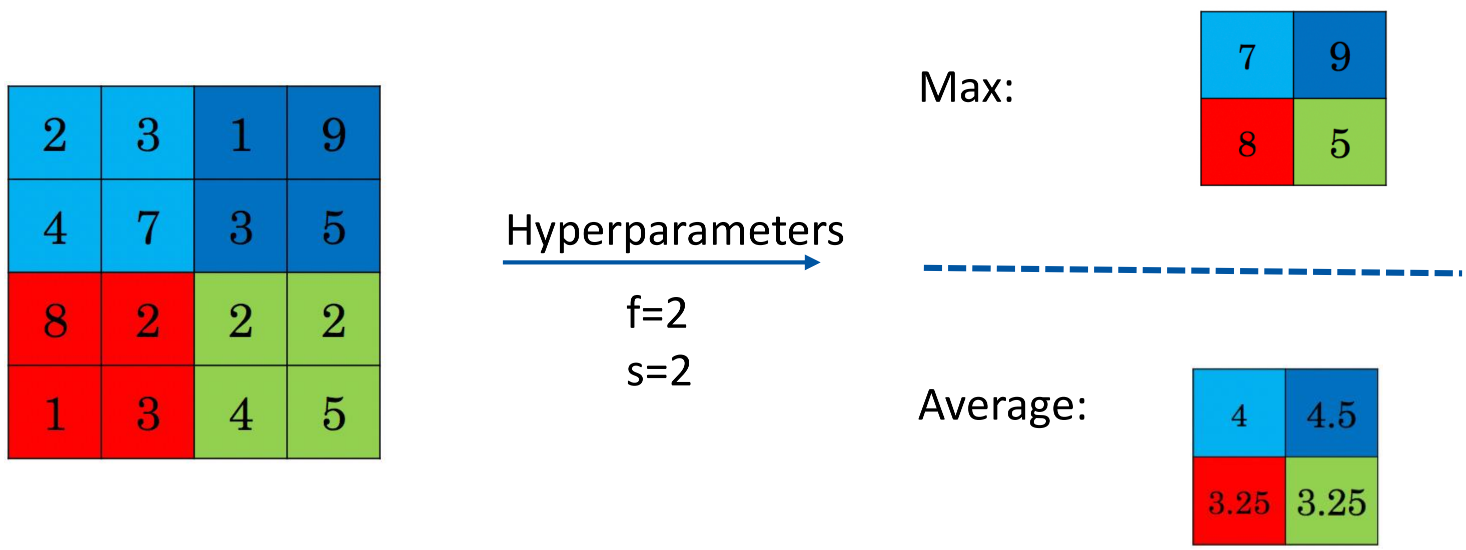
bias: $1 \times 1 \times 1 \times n_c^{[l]}$

Convolution forward

```
for i in range(m):                               # loop over the batch of training examples
    a_prev_pad = A_prev_pad[i,:]                 # Select ith training example's padded activation
    for h in range(n_H):                         # loop over vertical axis of the output volume
        for w in range(n_W):                     # loop over horizontal axis of the output volume
            for c in range(n_C):                 # loop over channels (= #filters) of the output volume

                # Find the corners of the current "slice"
                vert_start = h*stride
                vert_end = h*stride + f
                horiz_start = w*stride
                horiz_end = w*stride + f

                # Use the corners to define the (3D) slice of a_prev_pad (See Hint above the cell).
                a_slice_prev = a_prev_pad[vert_start:vert_end, horiz_start:horiz_end,:]

                # Convolve the (3D) slice with the correct filter W and bias b, to get back one output neuron.
                Z[i, h, w, c] = conv_single_step(a_slice_prev, W[:,:,:,c], b[:,:,:,c])
```

<Deep Learning, Andrew Ng>

Pooling



<Deep Learning, Andrew Ng>

Size of Pooling

- Hyperparameters
  - f: filter size
  - s : stride
  - p : padding (usually p=0)

- Max or average pooling
- No parameters to learn

Input size: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output size: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \frac{n_H^{[l-1]} - f^{[l]}}{s^{[l]}} + 1$$

$$n_W^{[l]} = \frac{n_W^{[l-1]} - f^{[l]}}{s^{[l]}} + 1$$

Pooling forward

```
for i in range(m):                         # loop over the training examples
    for h in range(n_H):                   # loop on the vertical axis of the output volume
        for w in range(n_W):               # loop on the horizontal axis of the output volume
            for c in range (n_C):          # loop over the channels of the output volume

                # Find the corners of the current "slice" (≈4 lines)
                vert_start = h*stride
                vert_end = h*stride + f
                horiz_start = w*stride
                horiz_end = w*stride + f

                # Use the corners to define the current slice on the ith training example of A_prev, channel c.
                a_prev_slice = A_prev[i,vert_start:vert_end, horiz_start:horiz_end,c]

                # Compute the pooling operation on the slice. Use an if statment to differentiate the modes.
                if mode == "max":
                    A[i, h, w, c] = np.max(a_prev_slice)
                elif mode == "avg":
                    A[i, h, w, c] = np.mean(a_prev_slice)
```

<Deep Learning, Andrew Ng>

Forward propagation step

$$X \xrightarrow[W1,\ b1]{\text{convolution}} Z1 \xrightarrow{\text{activation}} A1 \xrightarrow{\text{pooling}} P1 \xrightarrow[W2,\ b2]{\text{convolution}} Z2 \xrightarrow{\text{activation}} A2 \xrightarrow{\text{pooling}} P2$$

$$\xrightarrow{\text{flatten}} A3 \xrightarrow[W4,\ b4]{\text{linear}} Z4 \xrightarrow{\text{activation}} A4 \xrightarrow[W5,\ b5]{\text{linear}} Z5 \xrightarrow{\text{activation}} A5$$

Backward propagation step

	convolution		activation		pooling		convolution		activation		pooling	
X	←	dZ1	←	dA1	←	dP1	←	dZ2	←	dA2	←	dP2
dW1 db1						dW2 db2						

flatten		linear		activation		linear		activation	
←	dA3	←	dZ4	←	dA4	←	dZ5	←	dA5
	dW4 db4				dW5 db5				

Pooling backward

average

$\boxed{dA := dP \,/\, (distribute\_value) * np.ones \text{ of the shape}}$ <Deep Learning, Andrew Ng>

dP = 1 (scalar)

shape : (n_H, n_W) of the output matrix

distribute_value = n_H*n_W

```
In [15]: dP = 1

In [24]: n_H, n_W = (2,2)

In [25]: distribute_value = n_H*n_W

In [26]: dA = dP*np.ones((n_H,n_W))/float(distribute_value)

In [27]: dA

Out[27]: array([[ 0.25,  0.25],
                [ 0.25,  0.25]])
```

Pooling backward

max : $\boxed{\text{dA := mask*dP}}$ <Deep Learning, Andrew Ng>

```
In [28]: A = np.arange(4).reshape((2, 2))
         A

Out[28]: array([[0, 1],
                [2, 3]])
```

Pooling forward →

```
In [31]: P = np.max(A)
         P

Out[31]: 3
```

```
In [29]: mask = (A==np.max(A))
         mask

Out[29]: array([[False, False],
                [False,  True]], dtype=bool)
```

dP=1 Pooling backward →

```
In [33]: dA = mask*dP
         dA

Out[33]: array([[0, 0],
                [0, 1]])
```

Convolution layer backward

dA

<Deep Learning, Andrew Ng>

$$dA += \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} W_c \times dZ_{hw}$$

```
da [vert_start:vert_end, horiz_start:horiz_end, :] += W[:,:,:,c] * dZ[i, h, w, c]
```

dW

$$dW_c += \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} a_{slice} \times dZ_{hw}$$

```
dW[:,:,:,c] += a_slice * dZ[i, h, w, c]
```

Convolution layer backward

db

<Deep Learning, Andrew Ng>

$$db = \sum_h \sum_w dZ_{hw}$$

```
db[:,:,:,c] += dZ[i, h, w, c]
```

Convolution backward

```
for h in range(n_H):                    # loop over vertical axis of the output volume
    for w in range(n_W):                # loop over horizontal axis of the output volume
        for c in range(n_C):            # loop over the channels of the output volume

            # Find the corners of the current "slice"
            vert_start = h*stride
            vert_end = h*stride + f
            horiz_start = w*stride
            horiz_end = w*stride + f

            # Use the corners to define the slice from a_prev_pad
            a_slice = a_prev_pad[vert_start:vert_end,horiz_start:horiz_end,:]

            # Update gradients for the window and the filter's parameters
            da_prev_pad[vert_start:vert_end, horiz_start:horiz_end, :] += W[:,:,:,c] * dZ[i,h,w,c]
            dW[:,:,:,c] += a_slice * dZ[i,h,w,c]
            db[:,:,:,c] += dZ[i,h,w,c]
```

<Deep Learning, Andrew Ng>

Today's lesson

- Zero pad
- Convolution with single step
- Convolution forward (for all data)
- Pooling forward (max, average)
- Pooling backward (max, average)
- Convolution backward (for all data)