딥러닝 CNN
(Convolutional Neural Net)

CONTENTS

01 CNN 구조

CNN (Convolutional Neural Network)

A BIT OF HISTORY

Hubel & Wiesel,

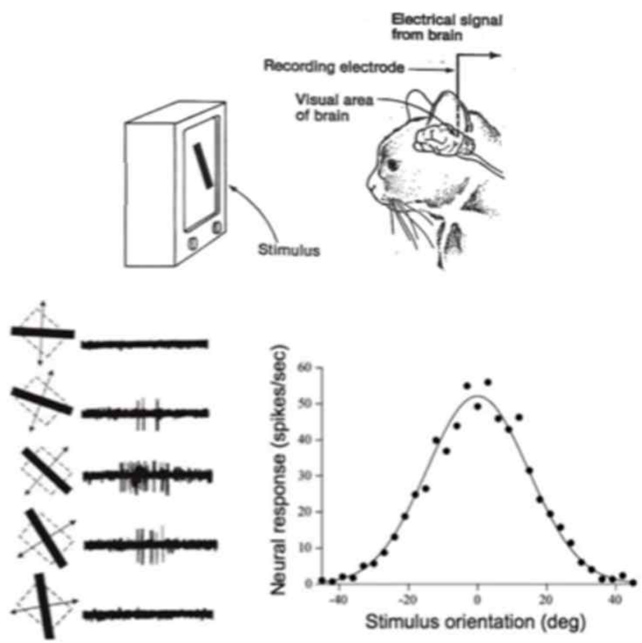
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

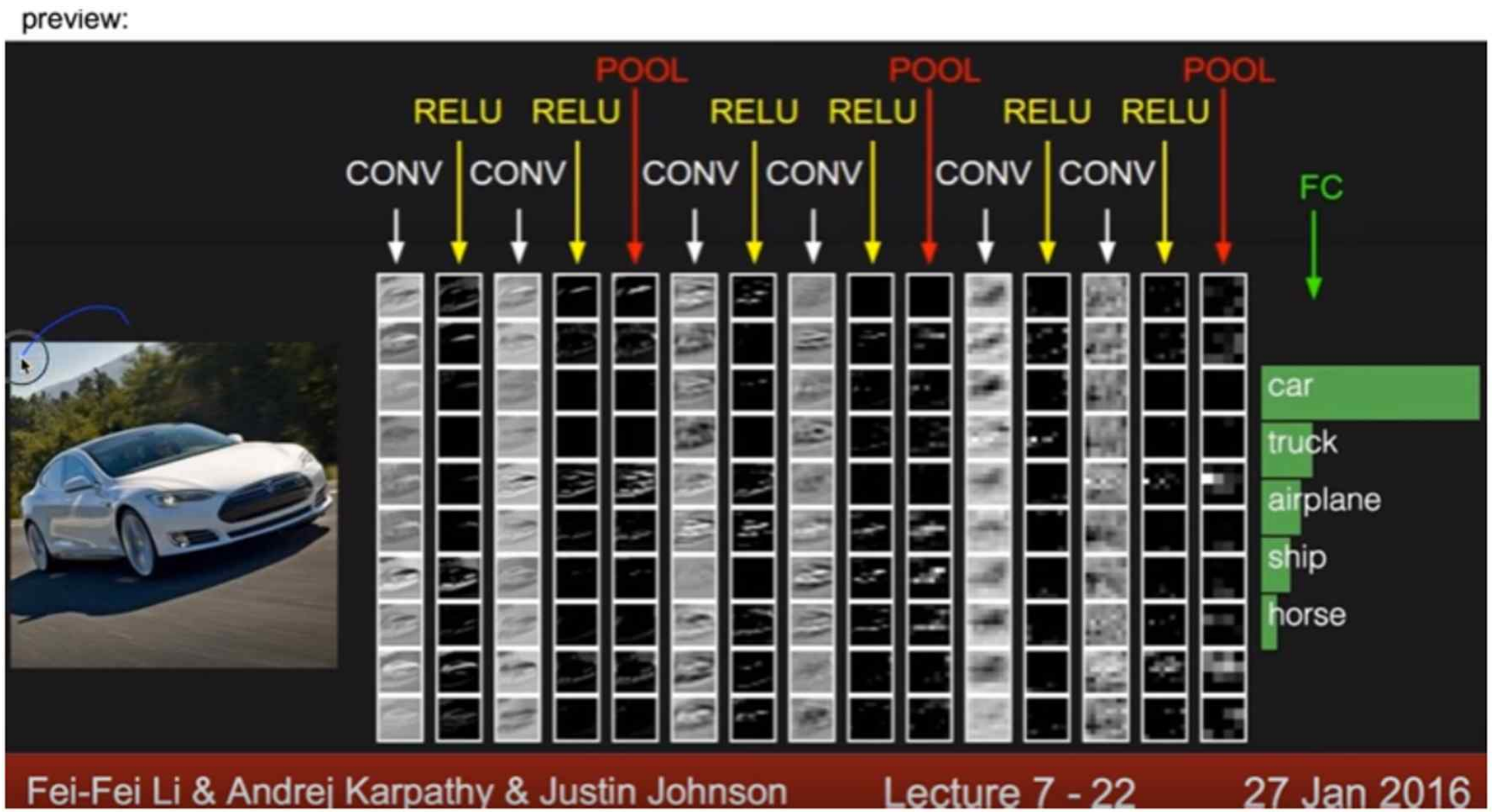
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...



Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 7   27 Jan 2016

CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

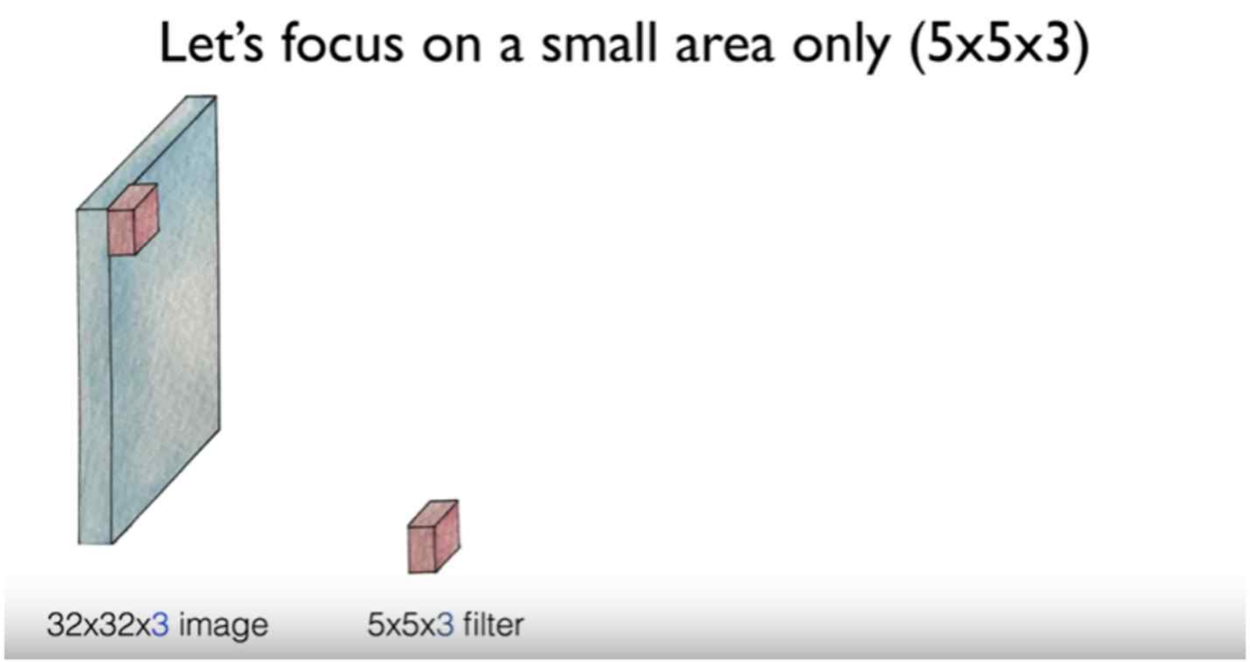
CNN FILTER

Start with an image (width x height x depth)

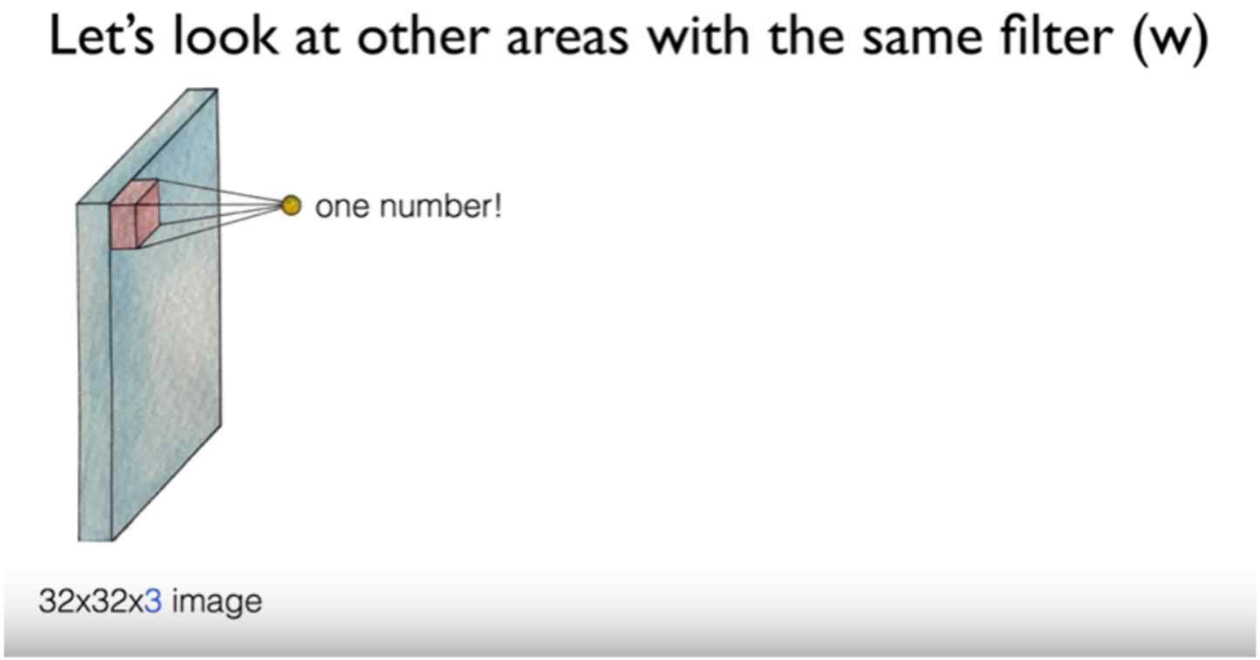
32x32x3 image

CNN (Convolutional Neural Network)

CNN FILTER

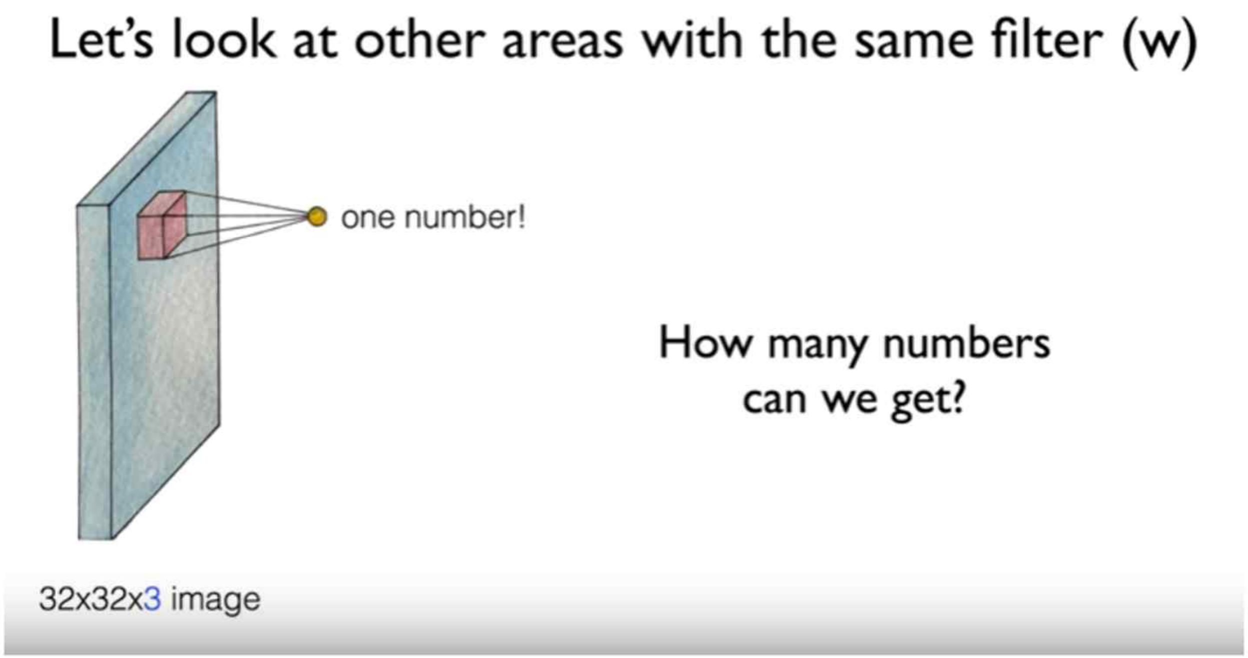


CNN FILTER


Let's look at other areas with the same filter (w)
one number!
32x32x3 image

CNN (Convolutional Neural Network)

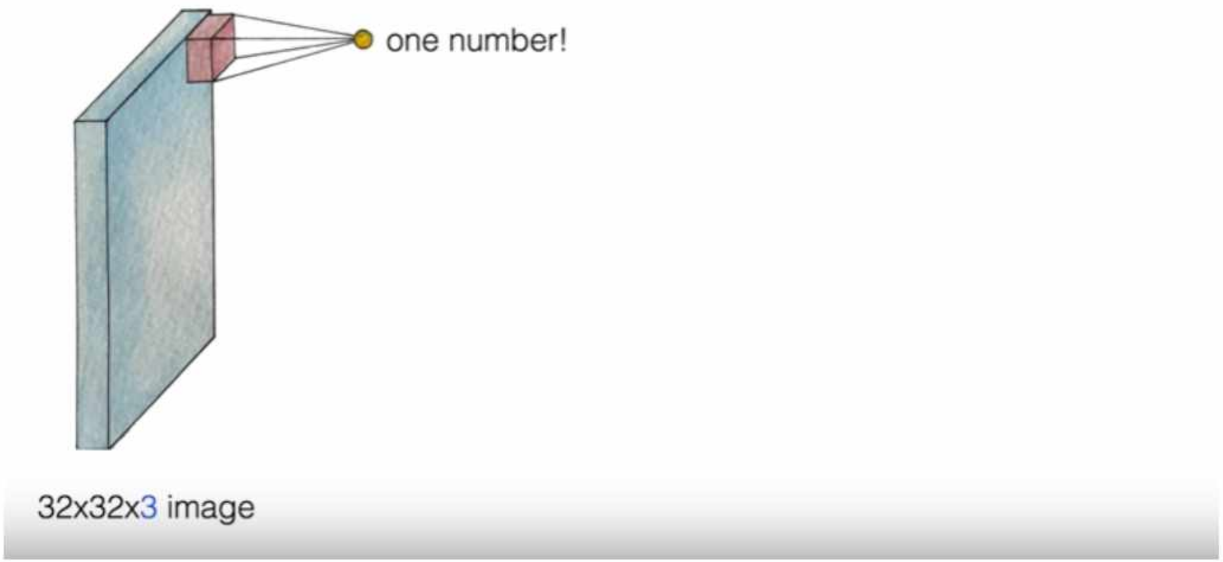
CNN FILTER



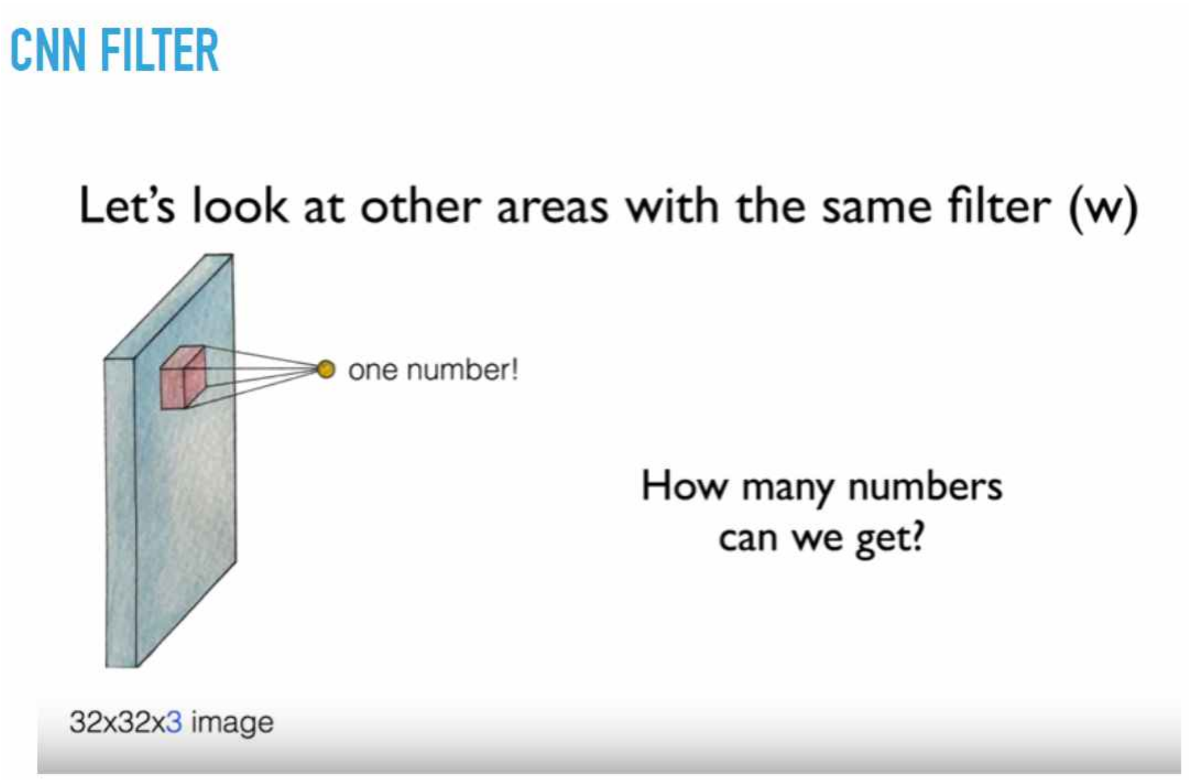
CNN (Convolutional Neural Network)

CNN FILTER

Let's look at other areas with the same filter (w)



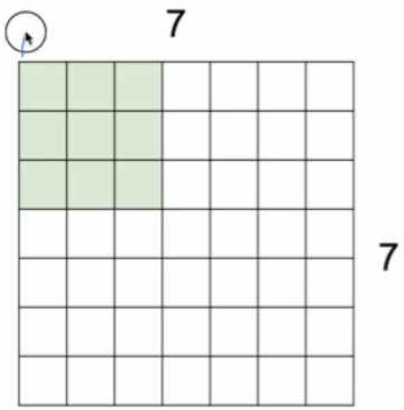
CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

MOVING FILTER

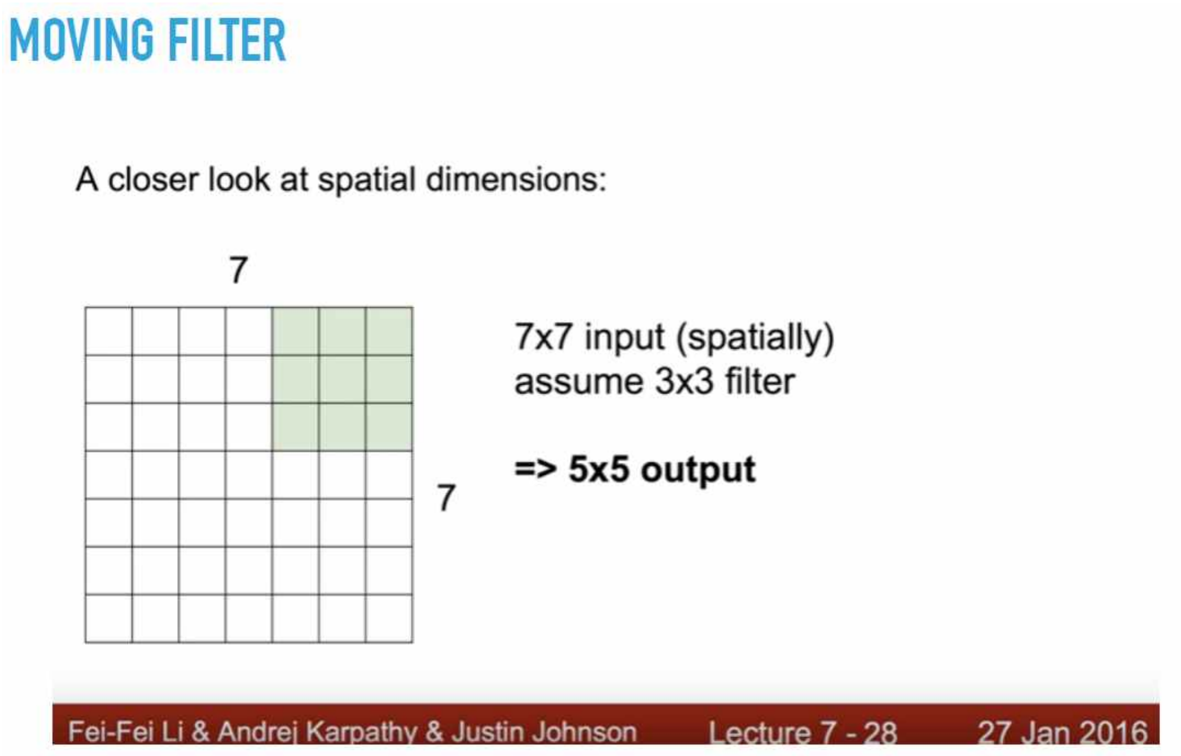
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 24   27 Jan 2016

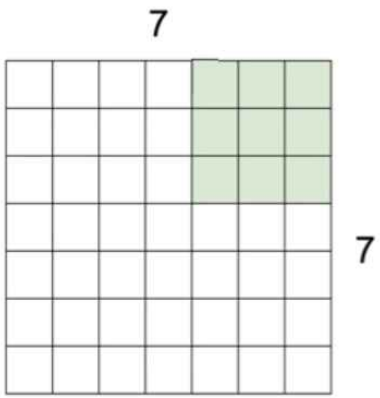
CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

MOVING FILTER

A closer look at spatial dimensions:

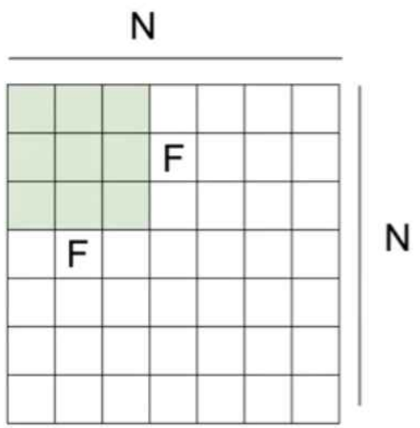


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 31 27 Jan 2016

CNN (Convolutional Neural Network)

MOVING FILTER

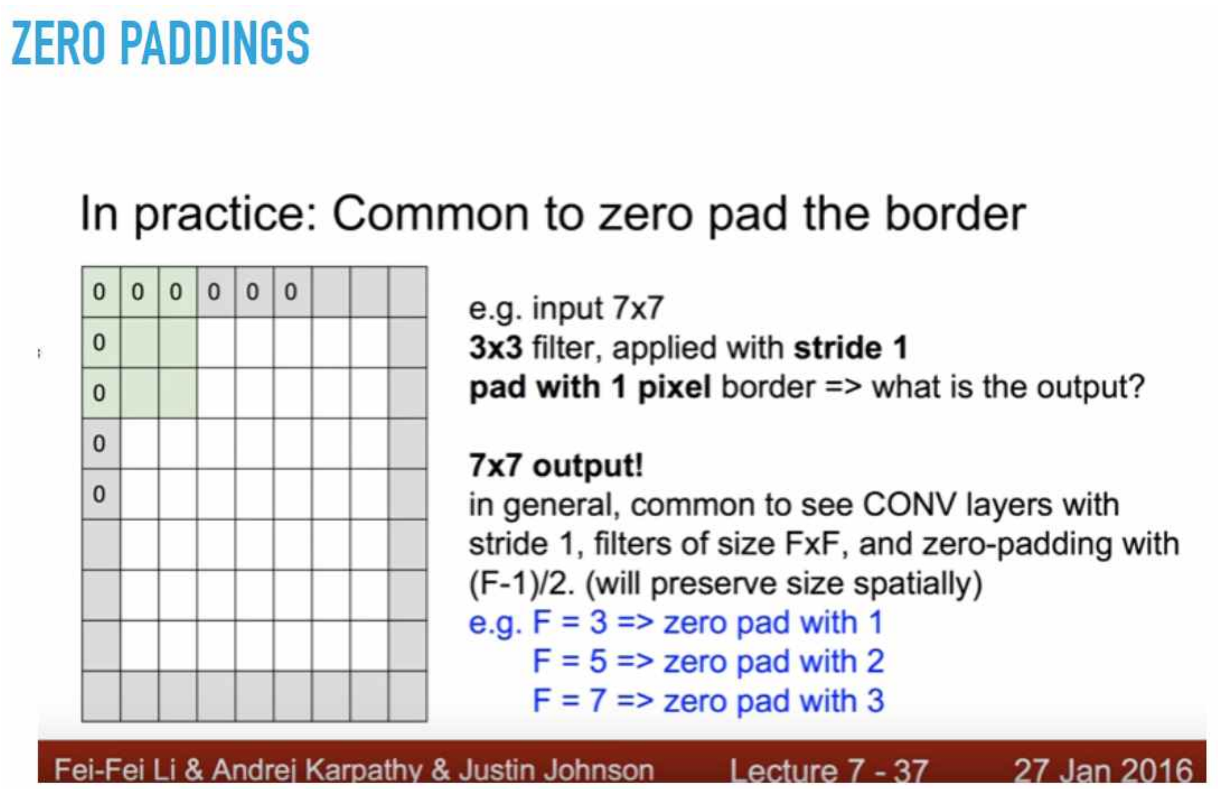


Output size:
(N - F) / stride + 1

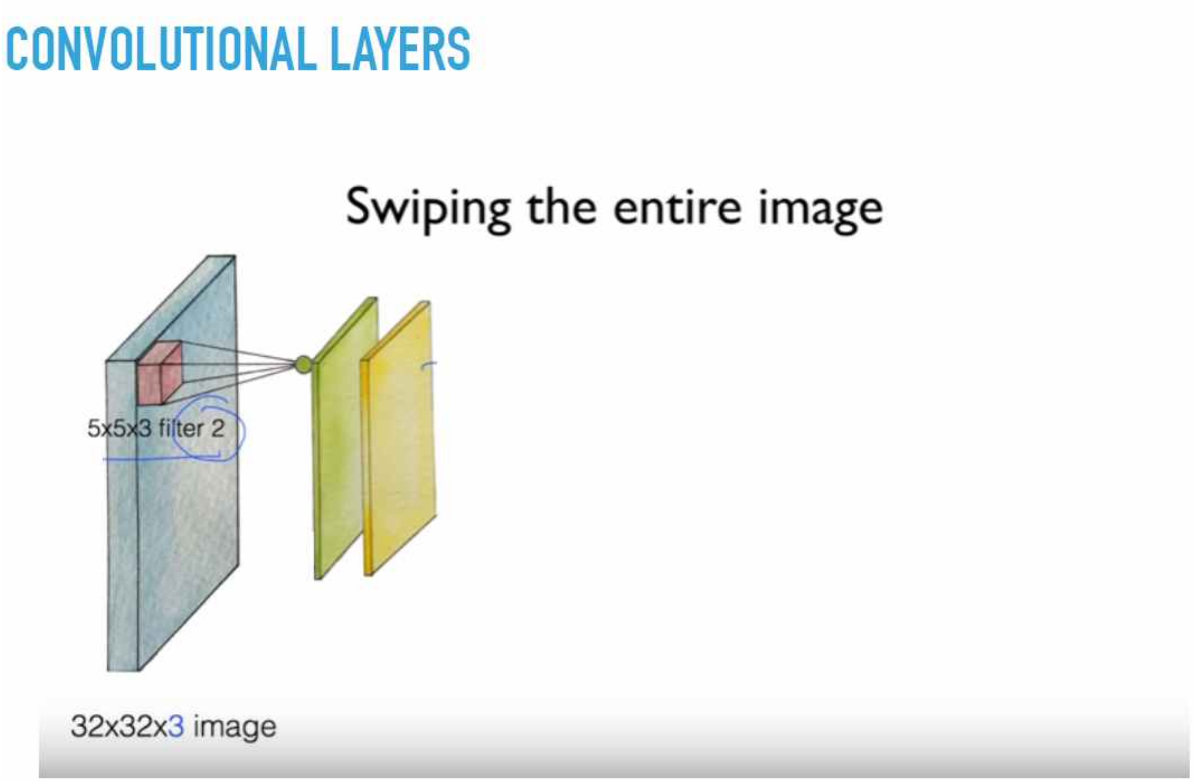
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 34 27 Jan 2016

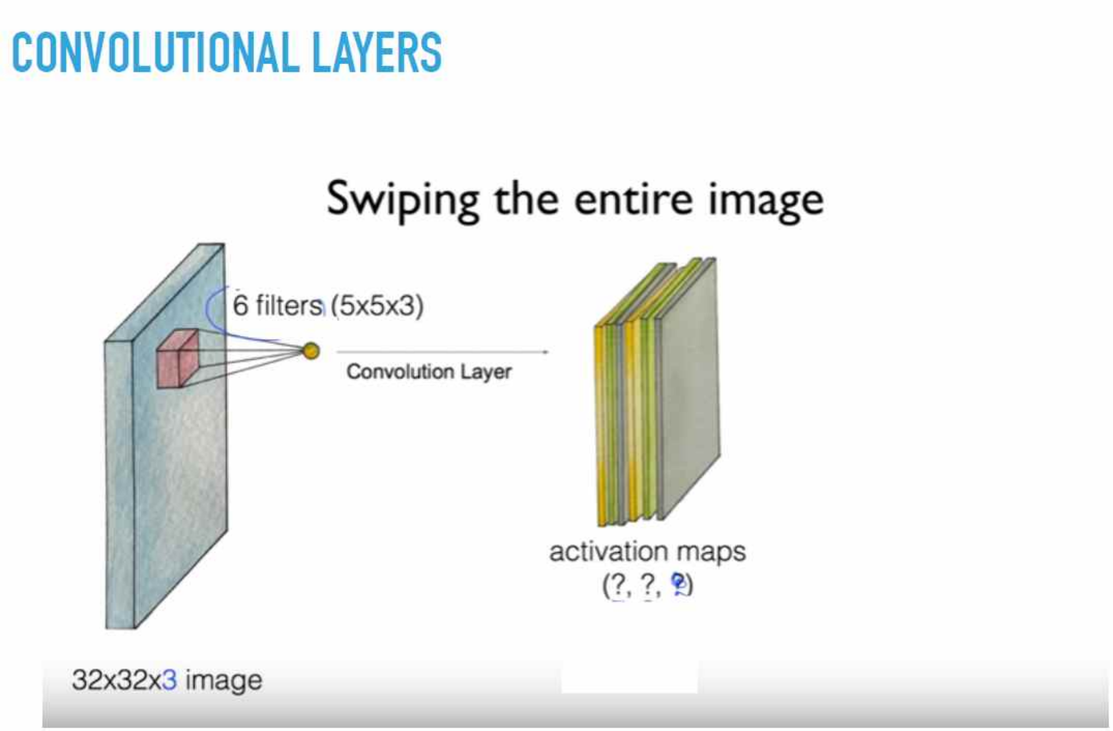
CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

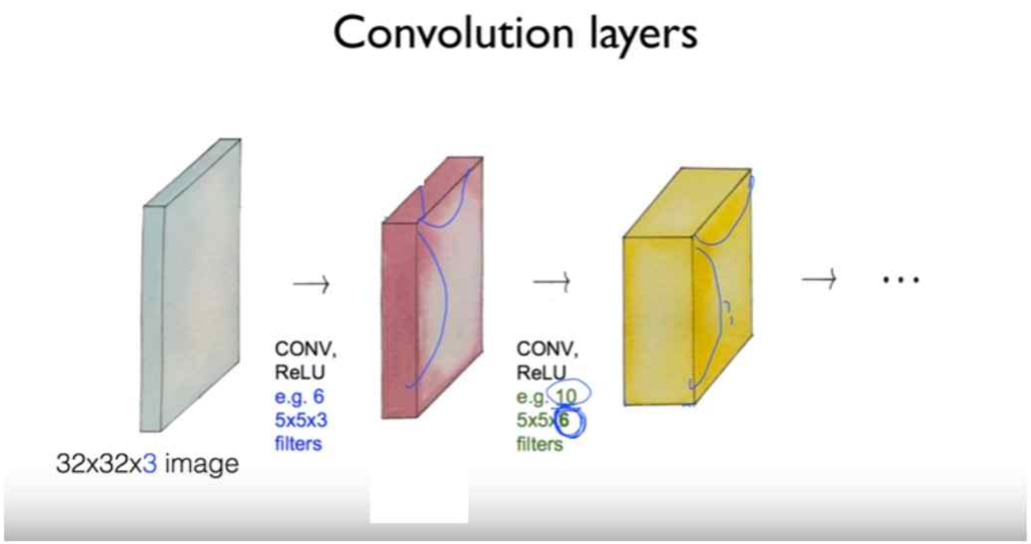


CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

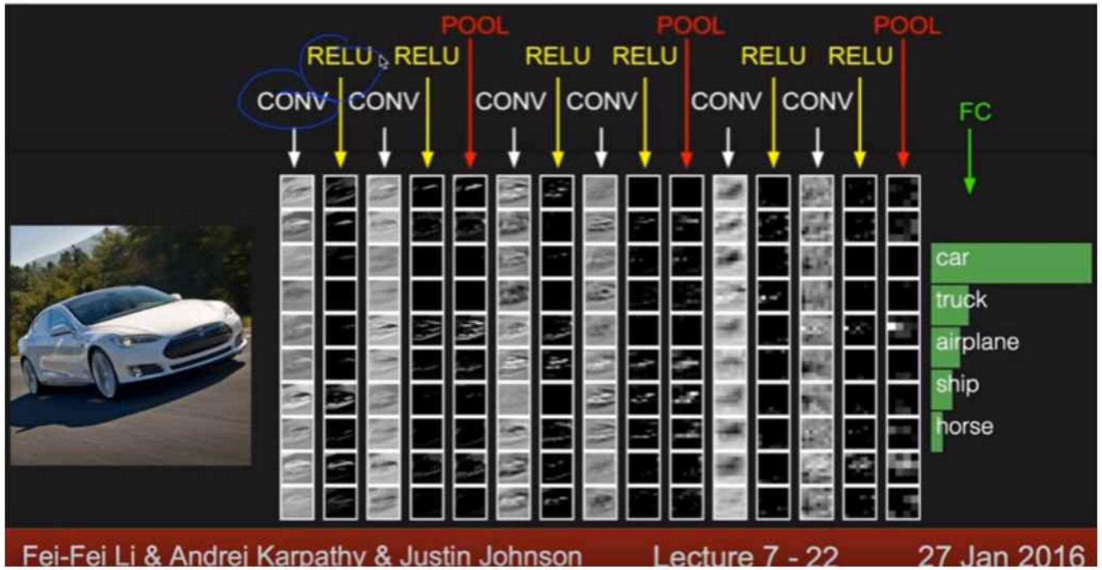
CONVOLUTIONAL LAYERS



CNN (Convolutional Neural Network)

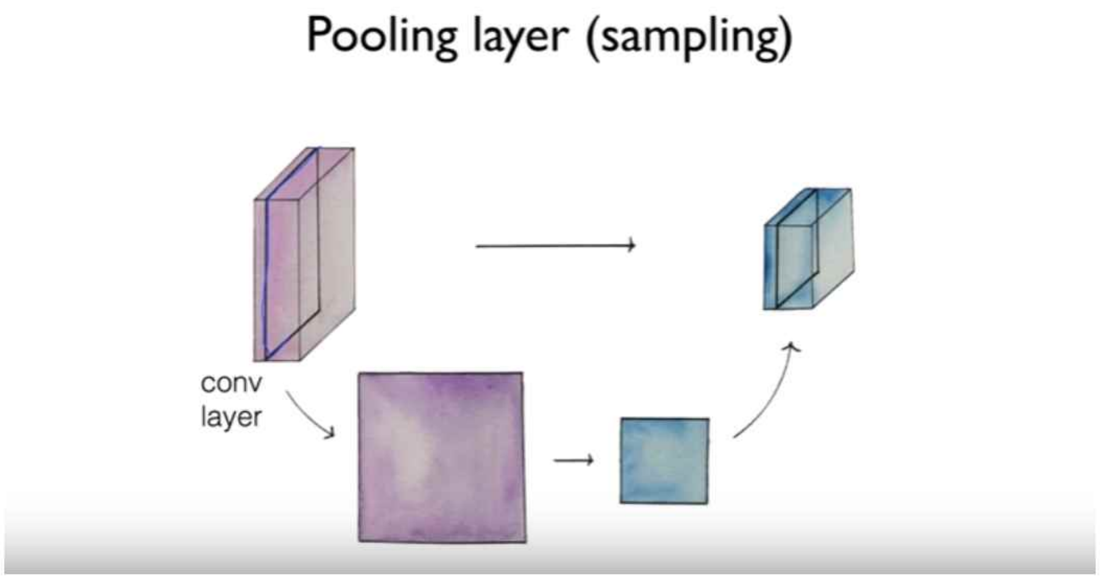
POOLING LAYER

preview:



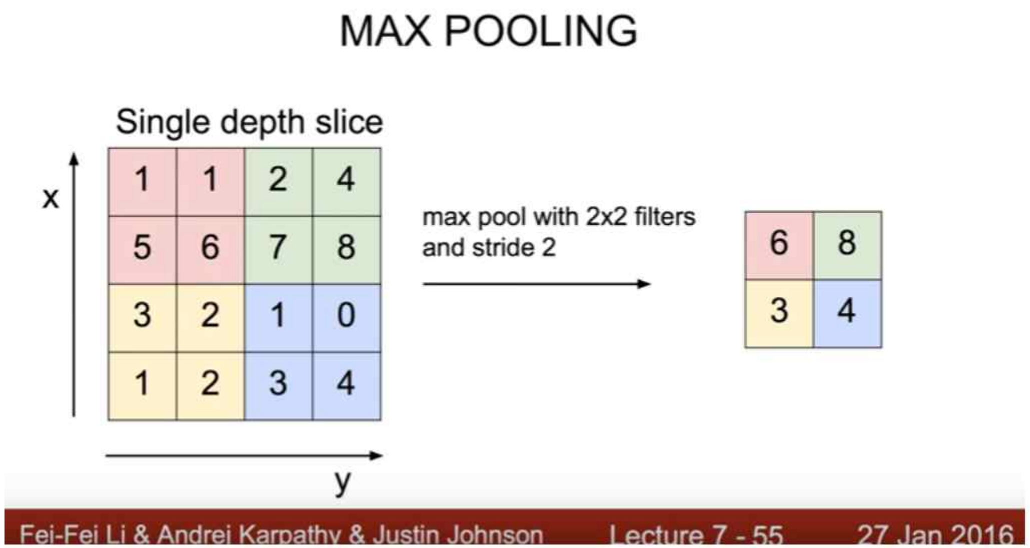
CNN (Convolutional Neural Network)

POOLING LAYER



CNN (Convolutional Neural Network)

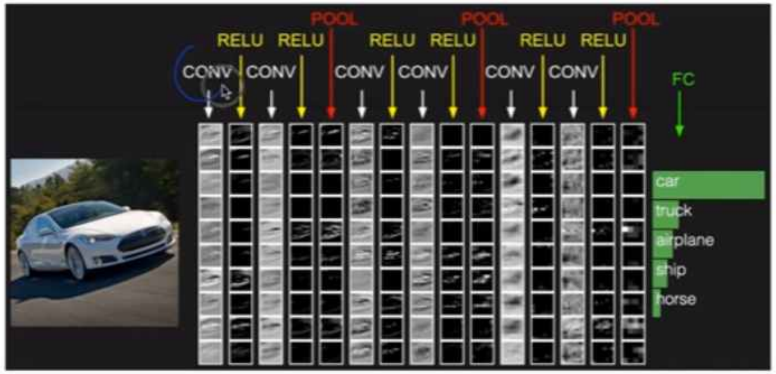
POOLING LAYER



CNN (Convolutional Neural Network)

FULLY CONNECTED LAYER

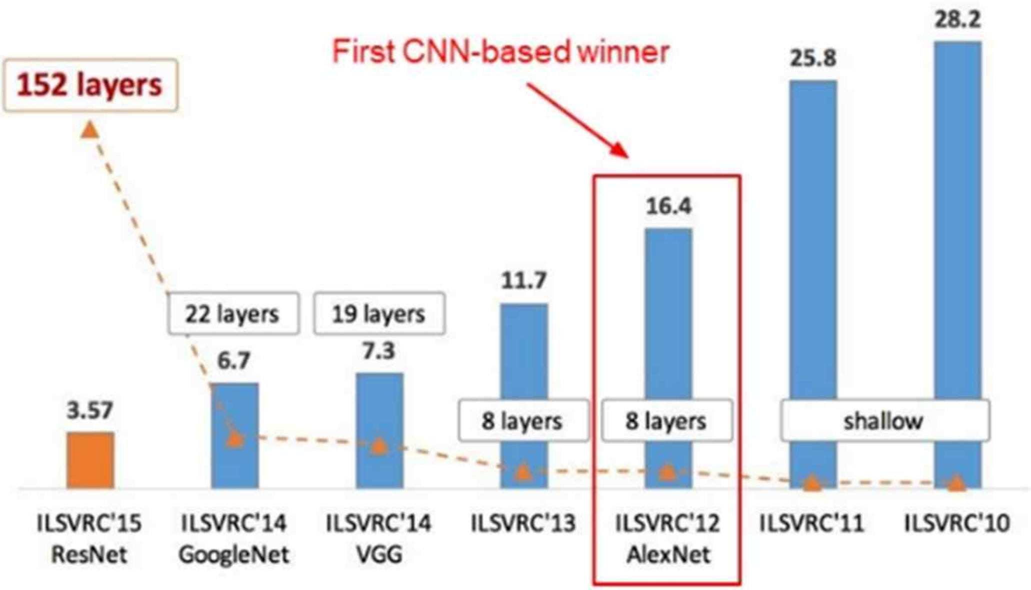
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 58 27 Jan 2016

CNN (Convolutional Neural Network) History

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



http://inha-kim.tistory.com/41

CNN (Convolutional Neural Network) History

- ILSVRC 2010: SuperVision (에러율 28.2%)
- ILSVRC 2011: SuperVision (에러율 25.8%)
- ILSVRC 2012: AlexNet (에러율 15.3%)
- ILSVRC 2013: ZFNet (에러율 11.2%)
- ILSVRC 2014: GoogLeNet (에러율 6.7%)
- ILSVRC 2015: ResNet (에러율 3.6%)
- ILSVRC 2016: Ensemble of Multiple Models (에러율 2.99%)

https://tensorflow.blog/2016/09/27/imagenet-ilsvrc-2016-results-out/

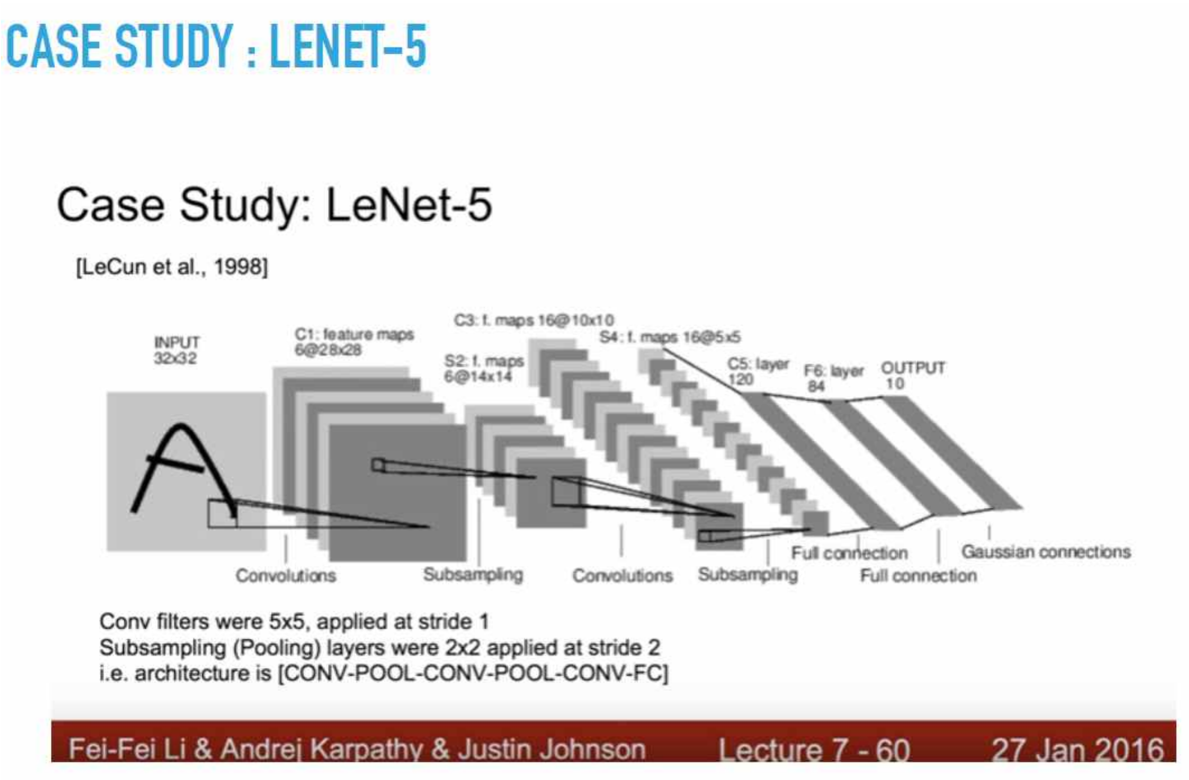
- ILSVRC 2017: SENet (에러율 2.25%)

https://deep-learning-study.tistory.com/539

- ILSVRC 2018: SENet (에러율 2.13%)
- ILSVRC 2019: EfficientNet (에러율1.5%)

https://greeksharifa.github.io/computer%20vision/2022/03/01/EfficientNet/

CNN (Convolutional Neural Network)

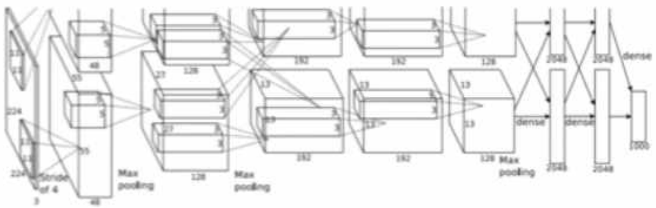


CNN (Convolutional Neural Network)

CASE STUDY : ALEXNET

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

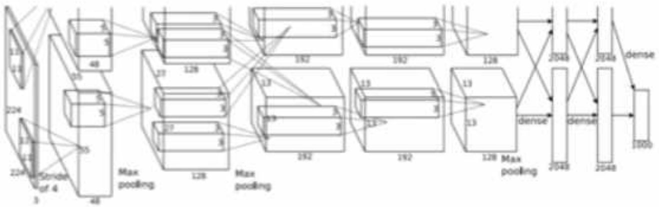
Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 63   27 Jan 2016

CNN (Convolutional Neural Network)

CASE STUDY : ALEXNET

Case Study: AlexNet

[Krizhevsky et al. 2012]

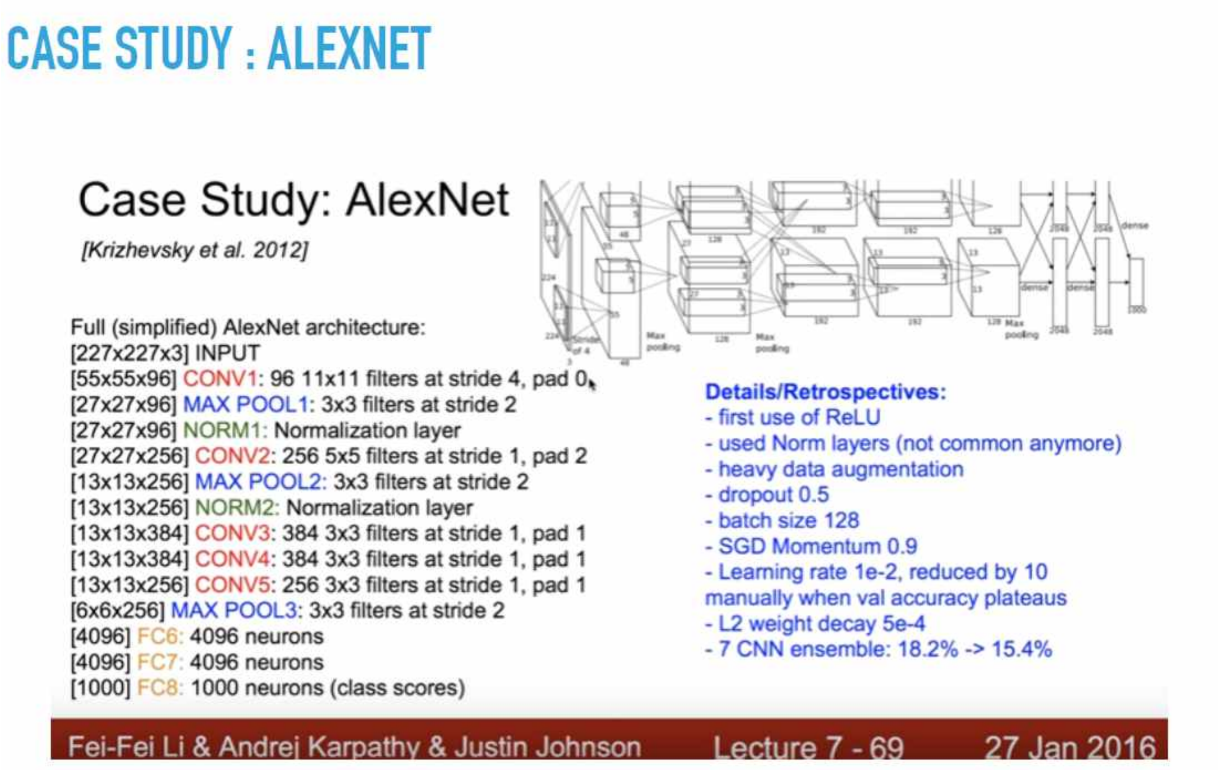


Input: 227x227x3 images
After CONV1: 55x55x96

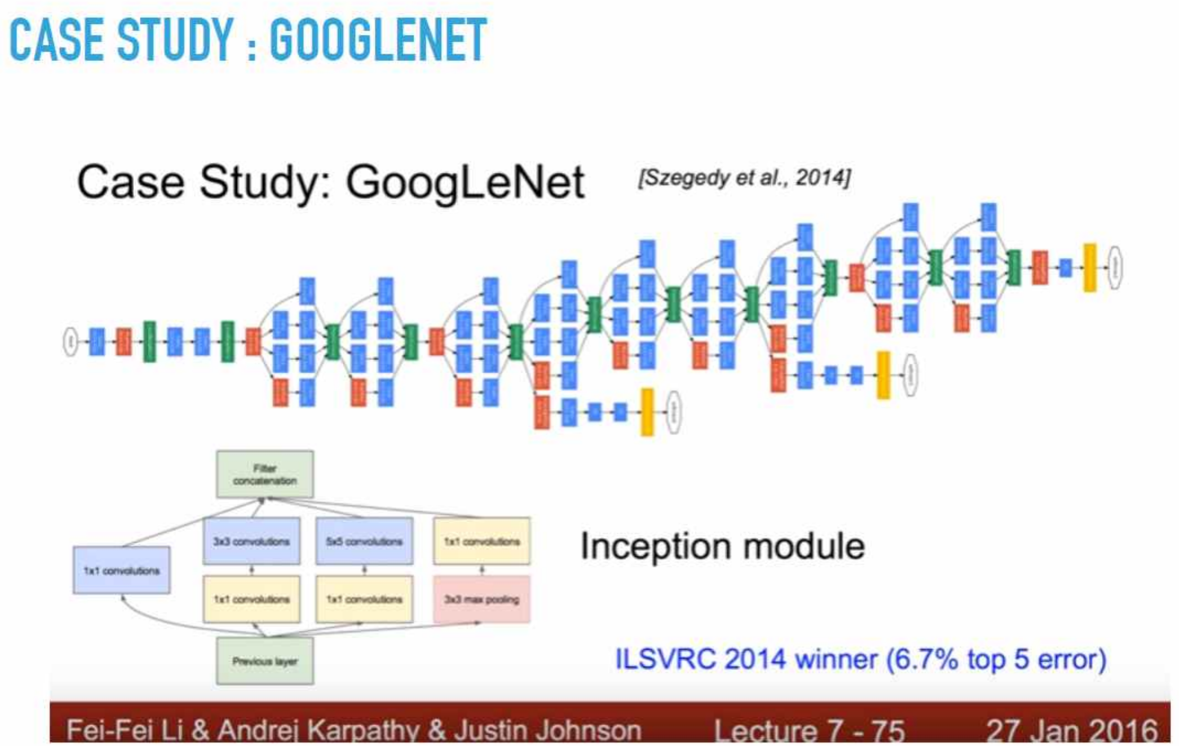
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 66 27 Jan 2016

CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

CASE STUDY : RESNET

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)

Research

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

*improvements are relative numbers

ICCV15

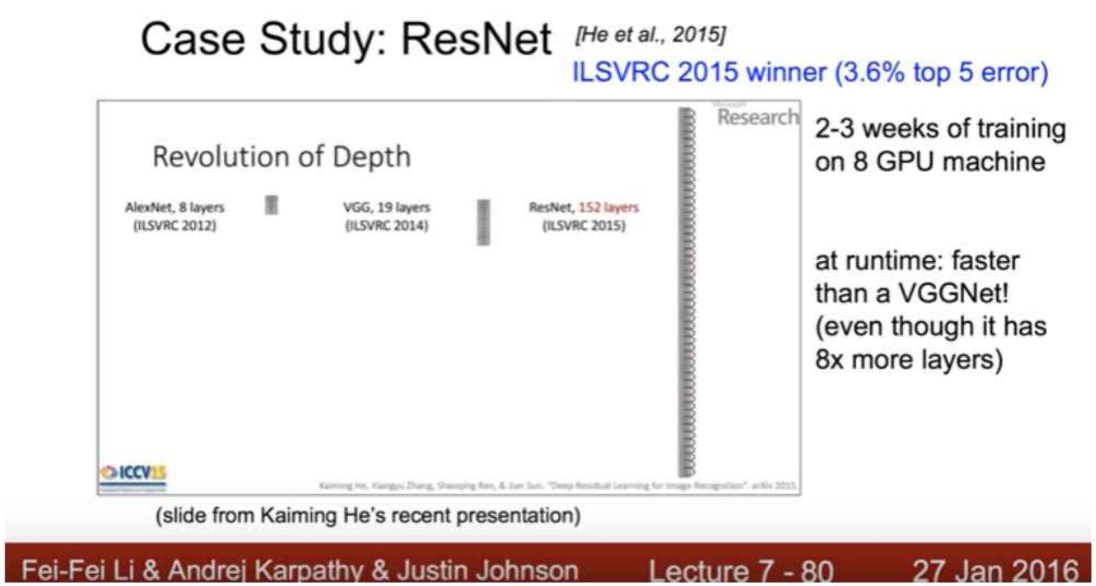
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 77 27 Jan 2016

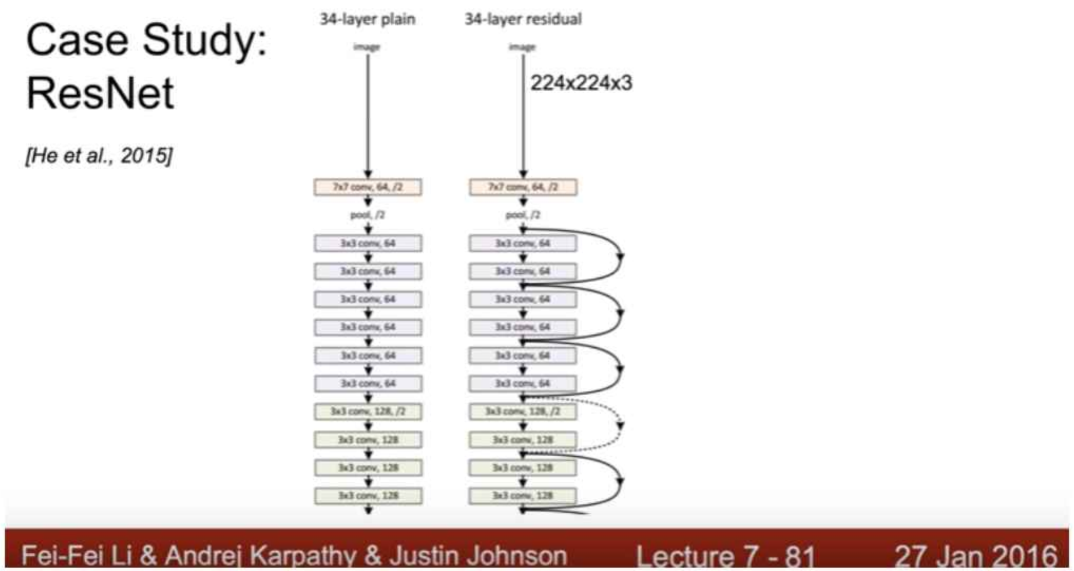
CNN (Convolutional Neural Network)

CASE STUDY : RESNET

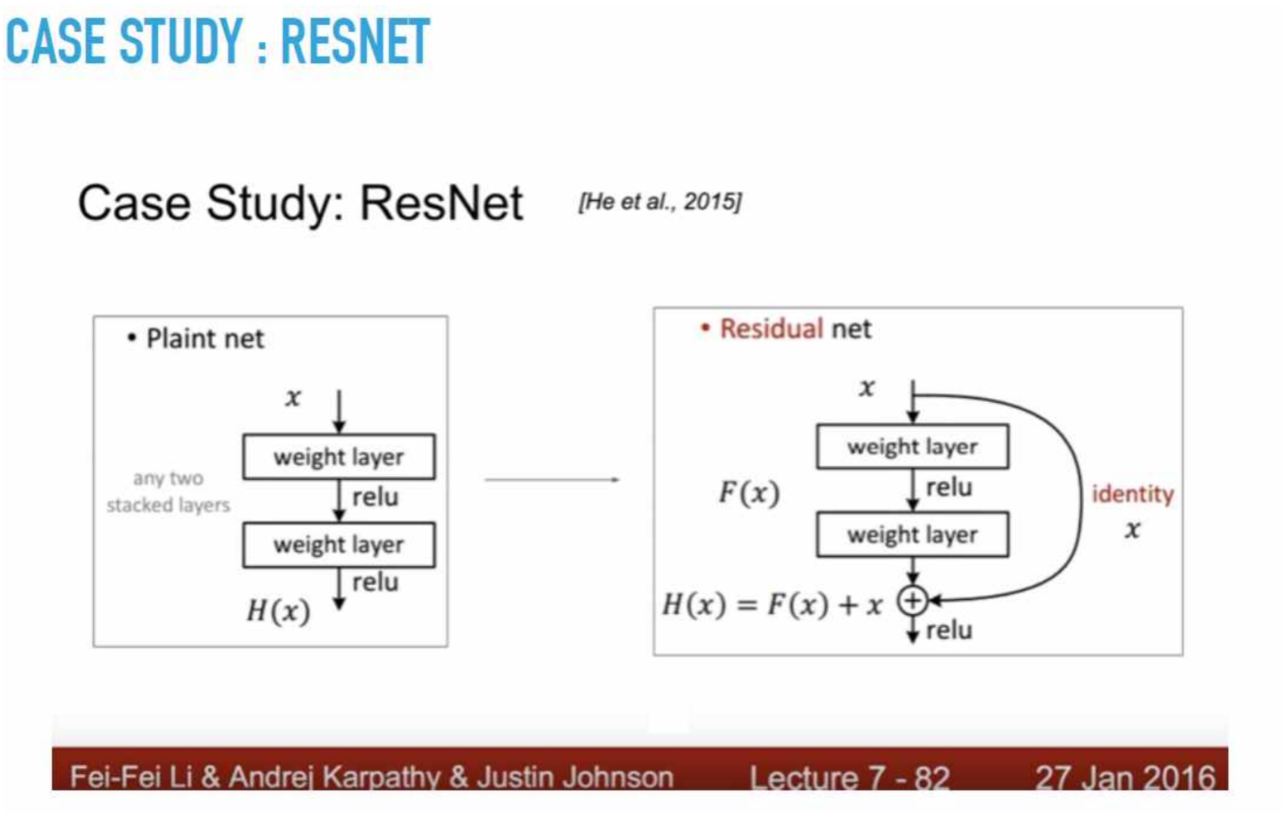


CNN (Convolutional Neural Network)

CASE STUDY : RESNET

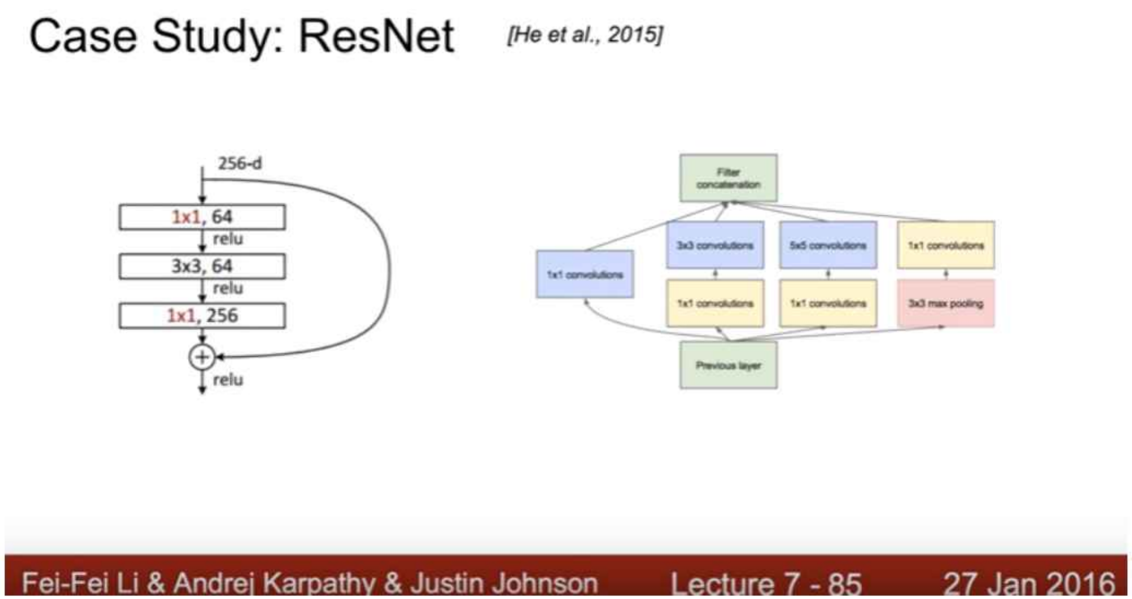


CNN (Convolutional Neural Network)



CNN (Convolutional Neural Network)

CASE STUDY : RESNET



CNN (Convolutional Neural Network)

CASE STUDY : SENTENCE CLASSIFICATION

Convolutional Neural Networks for Sentence Classification
[Yoon Kim, 2014]

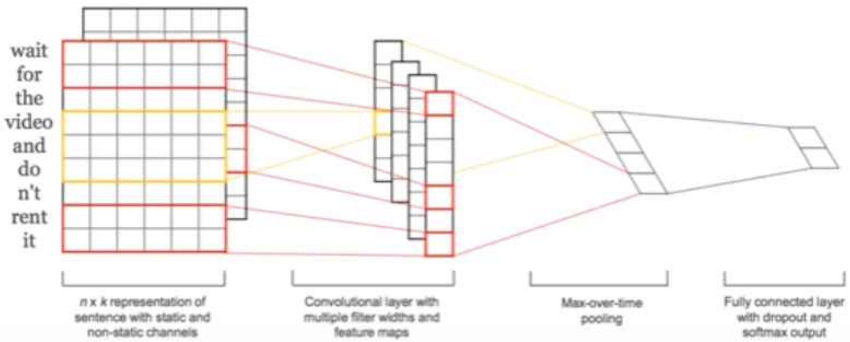
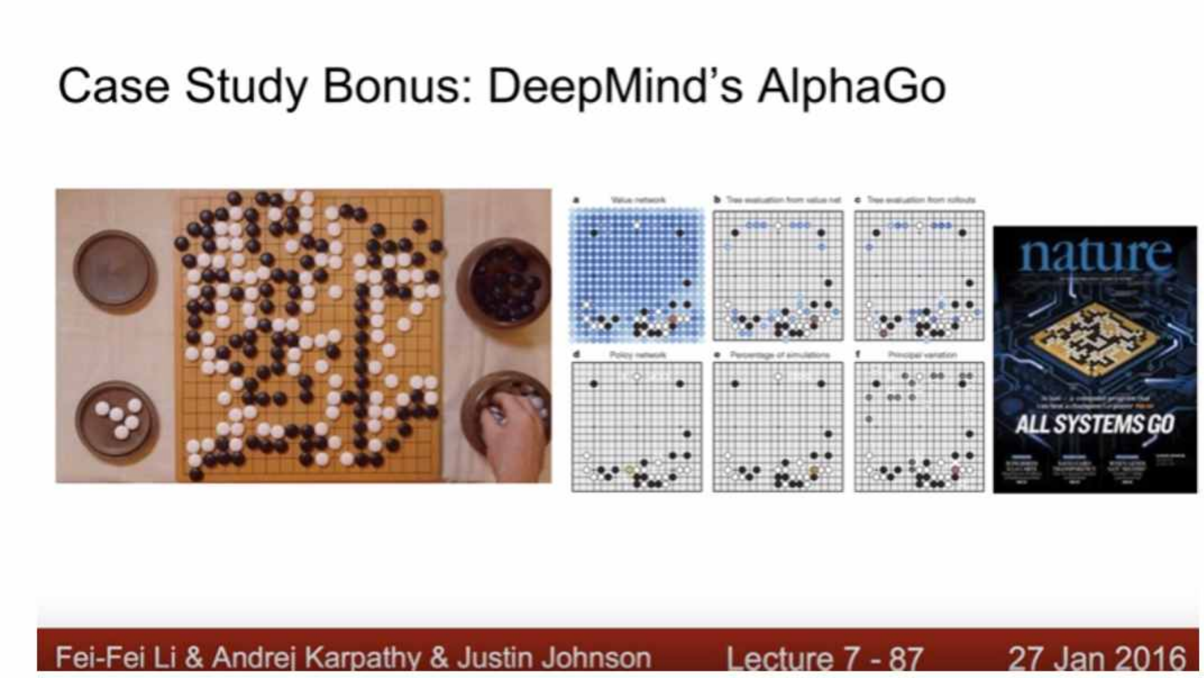


Figure 1: Model architecture with two channels for an example sentence.

CNN (Convolutional Neural Network)

CASE STUDY : DEEPMIND'S ALPHAGO



CNN (Convolutional Neural Network)

CASE STUDY : DEEPMIND'S ALPHAGO

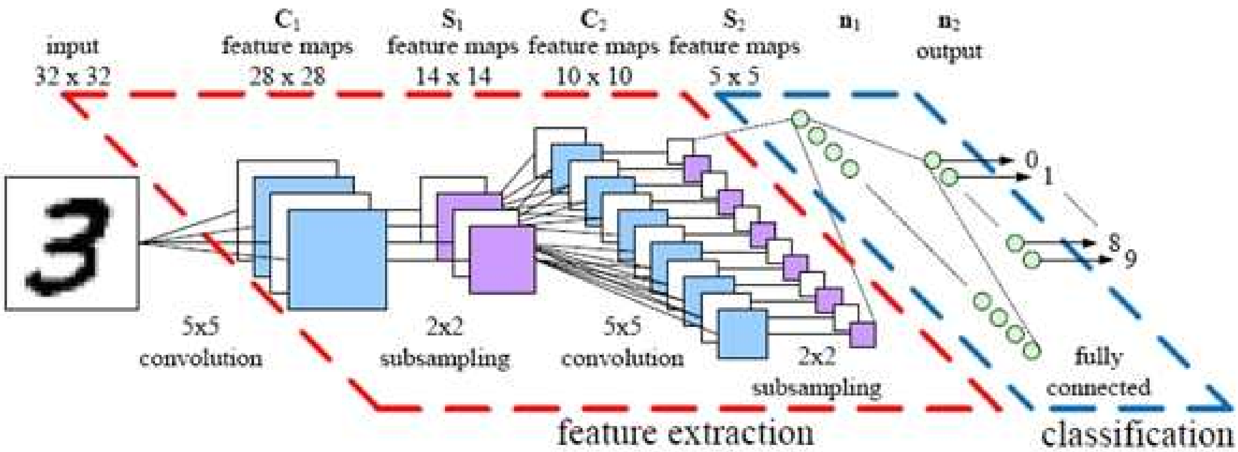
The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

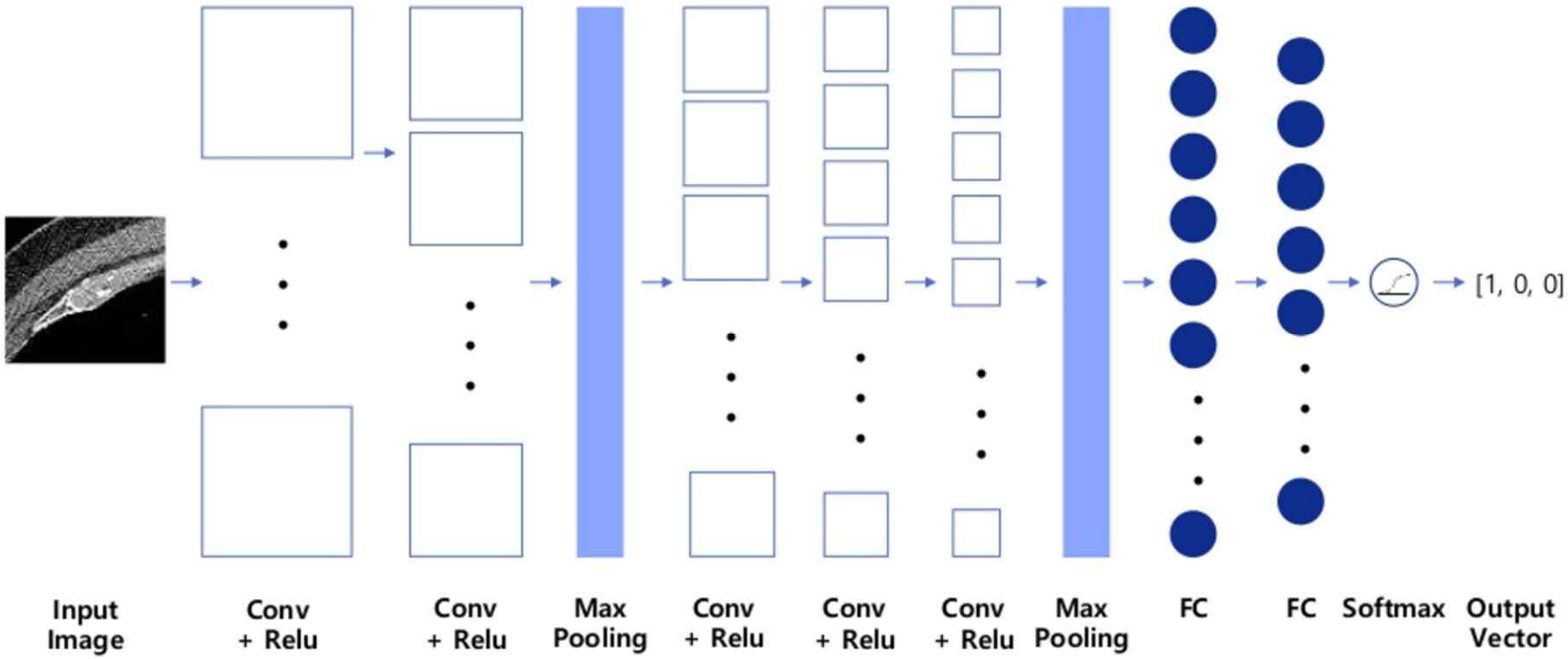
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 88 27 Jan 2016

02 CNN 구현 설명
합성곱 연산

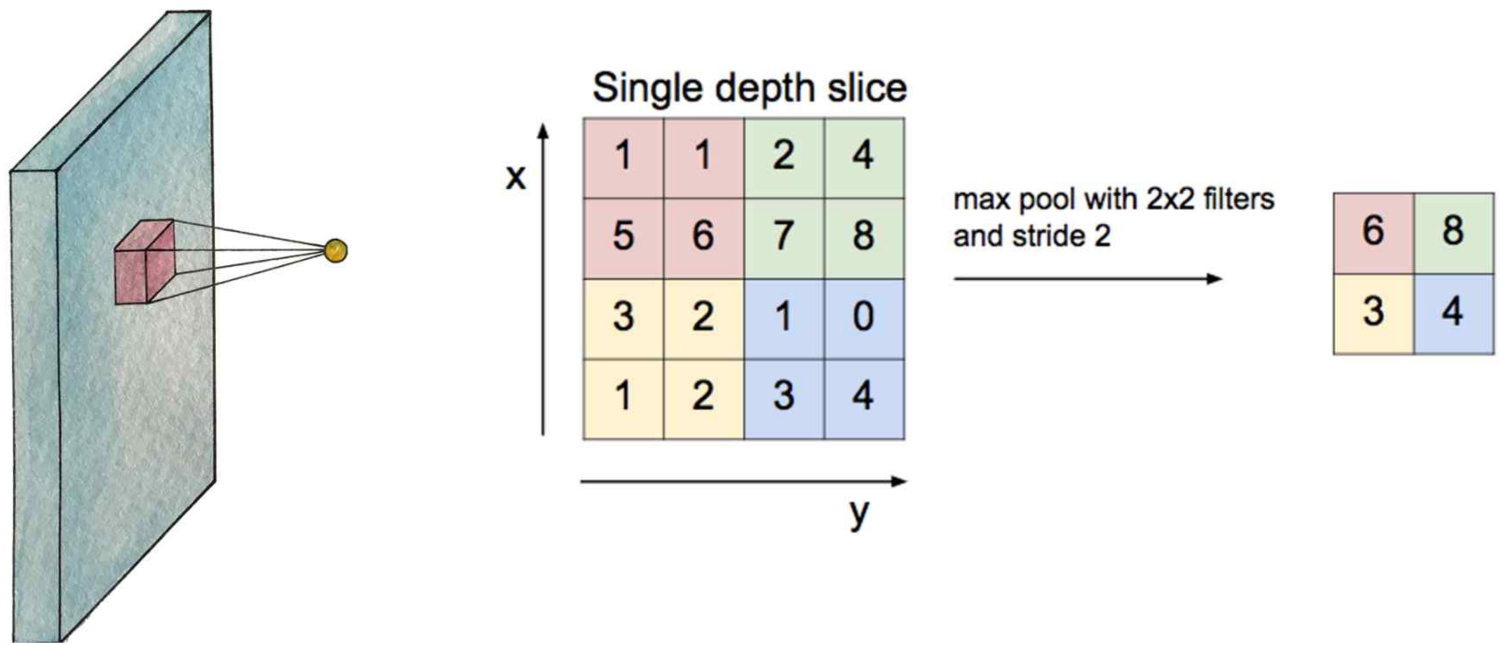
CNN


input
32 x 32
C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification

CNN for CT images

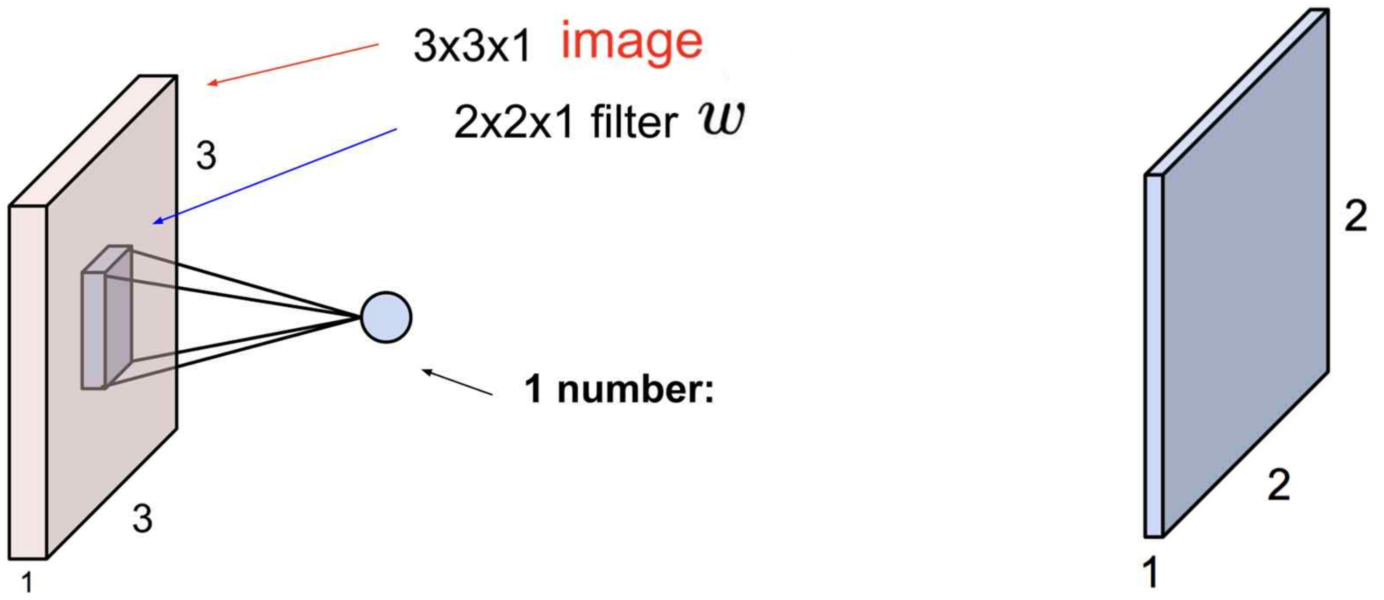


Convolution layer and max pooling



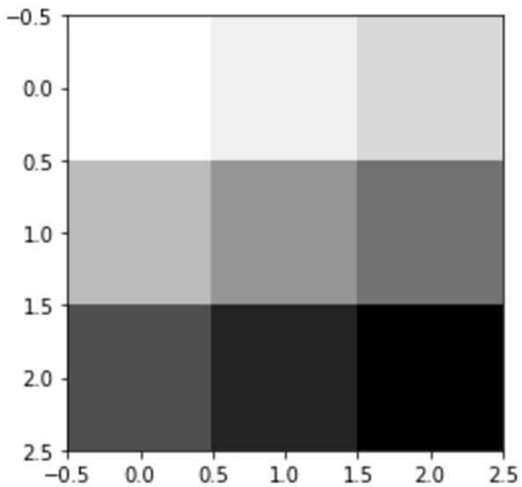
Simple convolution layer

Stride: 1x1



Simple convolution layer

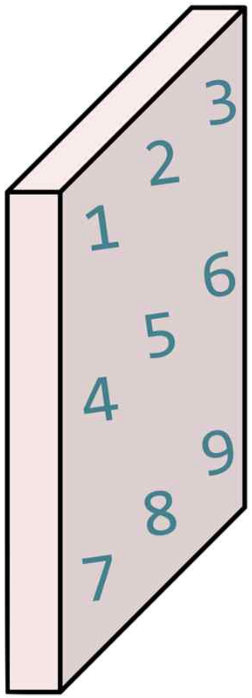
```
In [2]: sess = tf.InteractiveSession()
        image = np.array([[[[1],[2],[3]],
                           [[4],[5],[6]],
                           [[7],[8],[9]]]], dtype=np.float32)
        print(image.shape)
        plt.imshow(image.reshape(3,3), cmap='Greys')

        (1, 3, 3, 1)

Out[2]: <matplotlib.image.AxesImage at 0x10db67dd8>
```

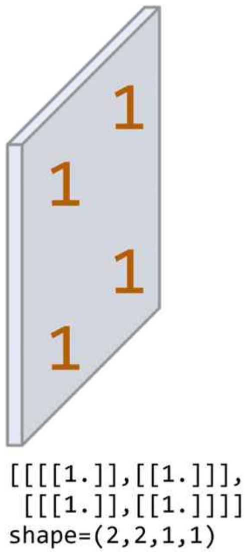


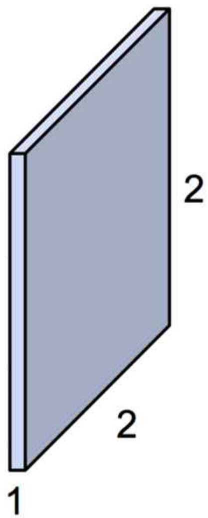
Toy image

Simple convolution layer


```
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID
```





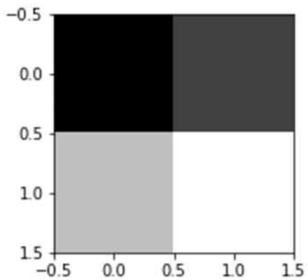


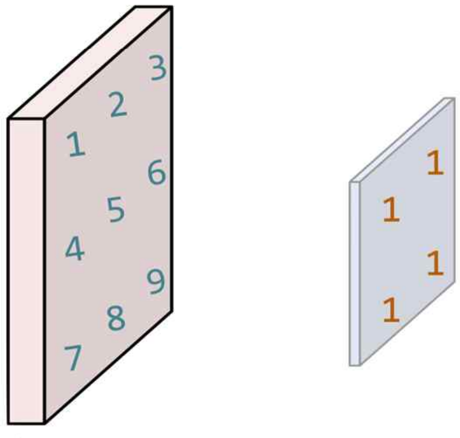
Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID

```python
# print("imag:\n", image)
print("image.shape", image.shape)
weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')
```

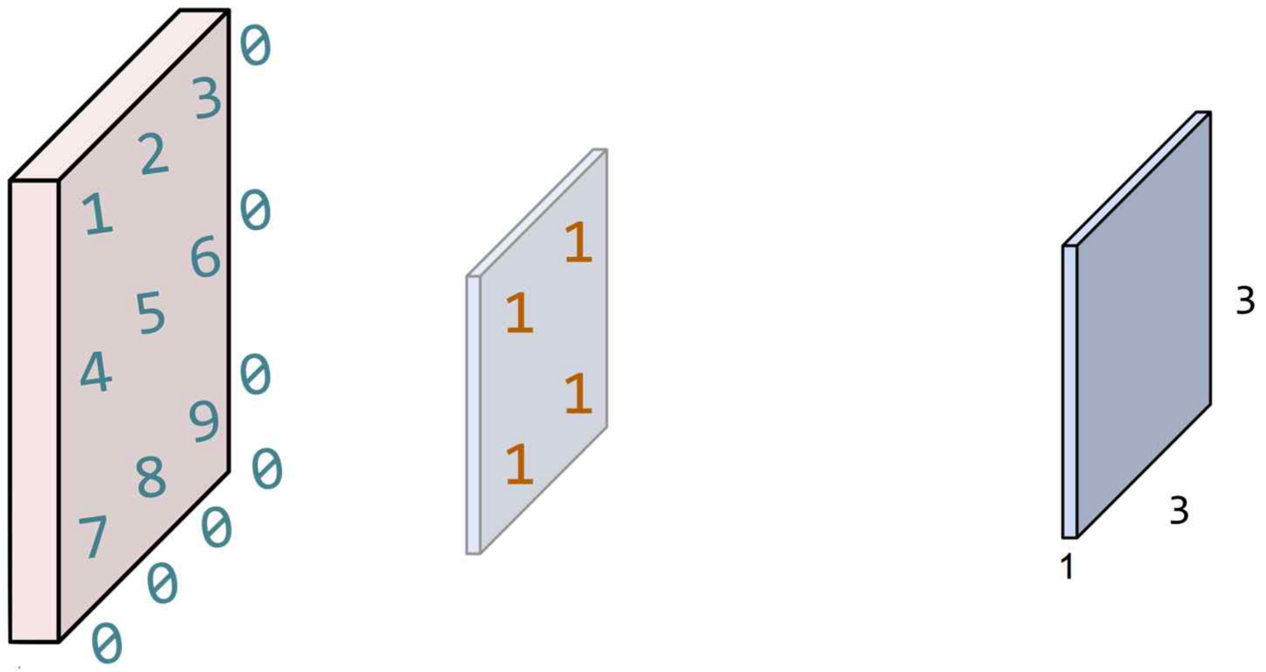
```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 2, 2, 1)
[[ 12.  16.]
 [ 24.  28.]]
```





Simple convolution layer

`Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding:` **SAME**

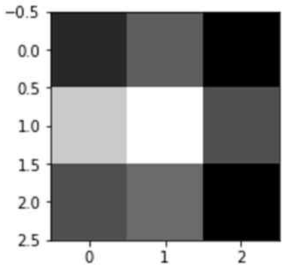


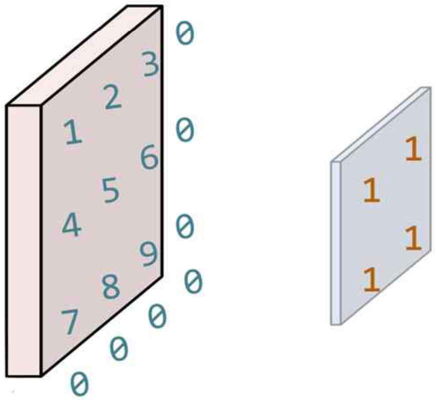
Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: SAME

```
# print("imag:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 3, 3, 1)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
```



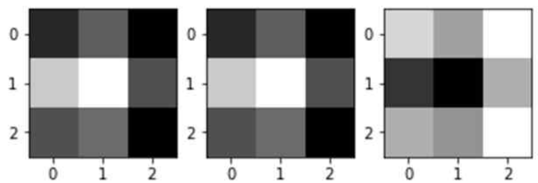


Simple convolution layer

3 filters (2,2,1,3)

```
# print("imag:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.,10.,-1.]],[[1.,10.,-1.]]],
                      [[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
conv2d_img.shape (1, 3, 3, 3)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
[[ 120.  160.   90.]
 [ 240.  280.  150.]
 [ 150.  170.   90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```



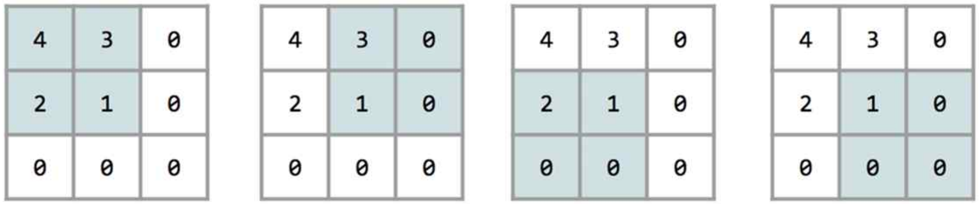
Max Pooling

4	3
2	1

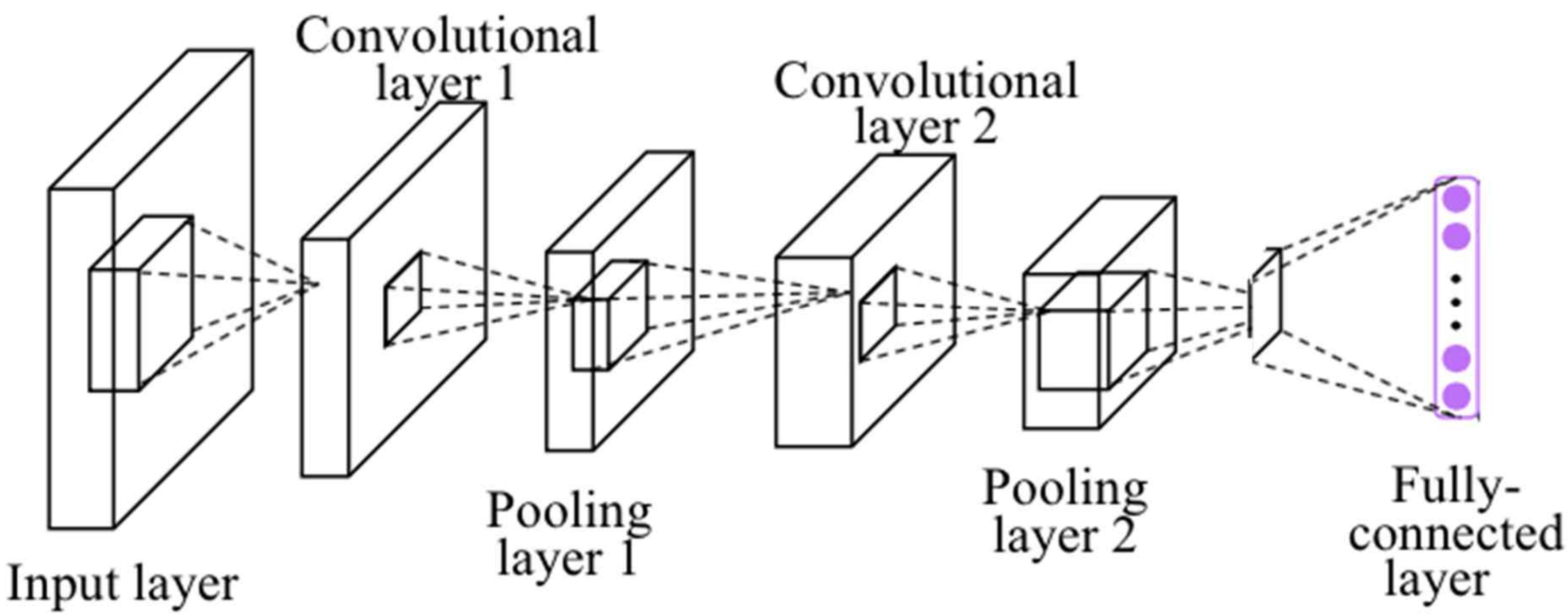
```
In [19]: image = np.array([[[[4],[3]],
                           [[2],[1]]]], dtype=np.float32)
         pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
                               strides=[1, 1, 1, 1], padding='SAME')
         print(pool.shape)
         print(pool.eval())
```

```
(1, 2, 2, 1)
[[[[ 4.]
   [ 3.]]

  [[ 2.]
   [ 1.]]]]
```

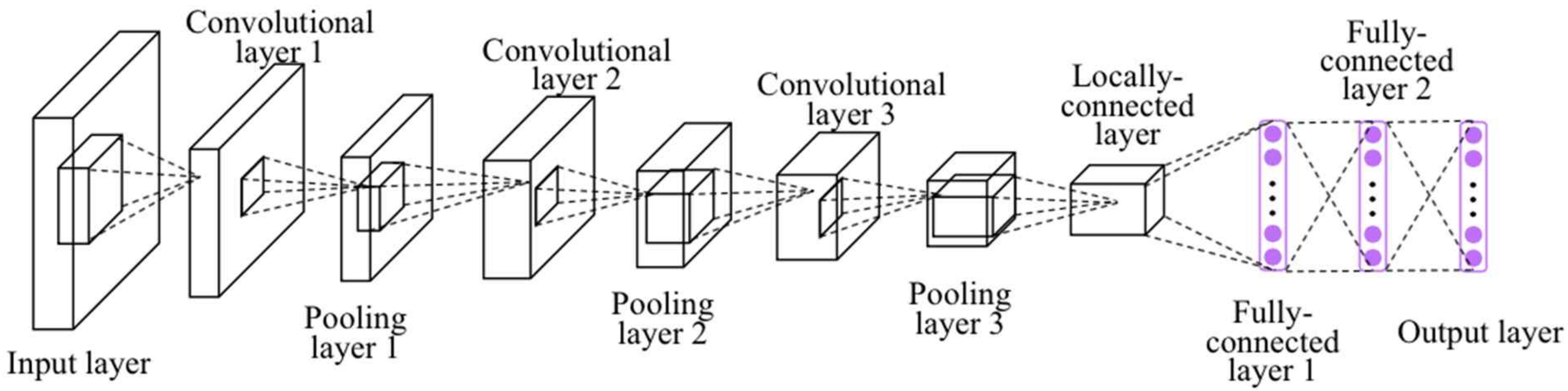
SAME: Zero paddings



Simple CNN


Convolutional
layer 1
Convolutional
layer 2
Input layer
Pooling
layer 1
Pooling
layer 2
Fully-
connected
layer

Deep CNN



Convolutional layer 1
Convolutional layer 2
Convolutional layer 3
Locally-connected layer
Fully-connected layer 2
Input layer
Pooling layer 1
Pooling layer 2
Pooling layer 3
Fully-connected layer 1
Output layer

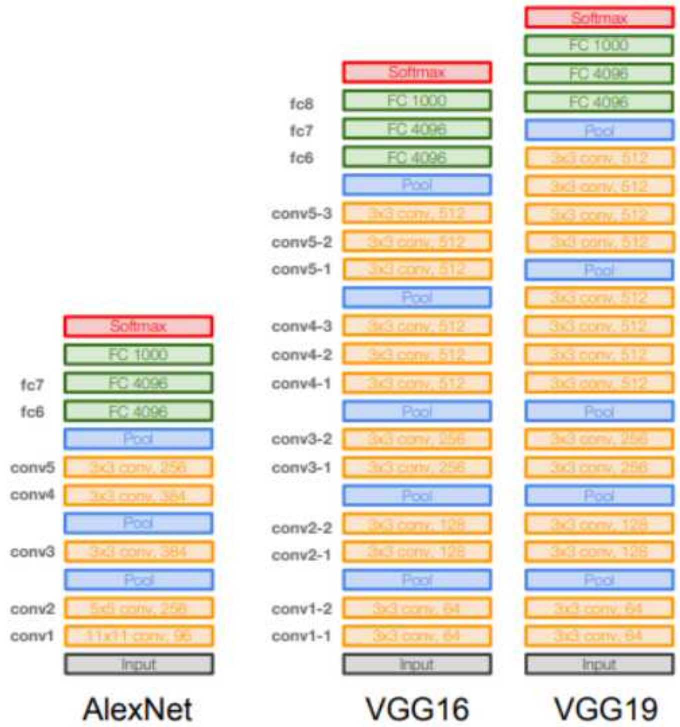
VGGNet

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



VGG : 2014 2nd winner, Visual Geometry Group in Oxford

`tf.keras.applications.VGG19()`