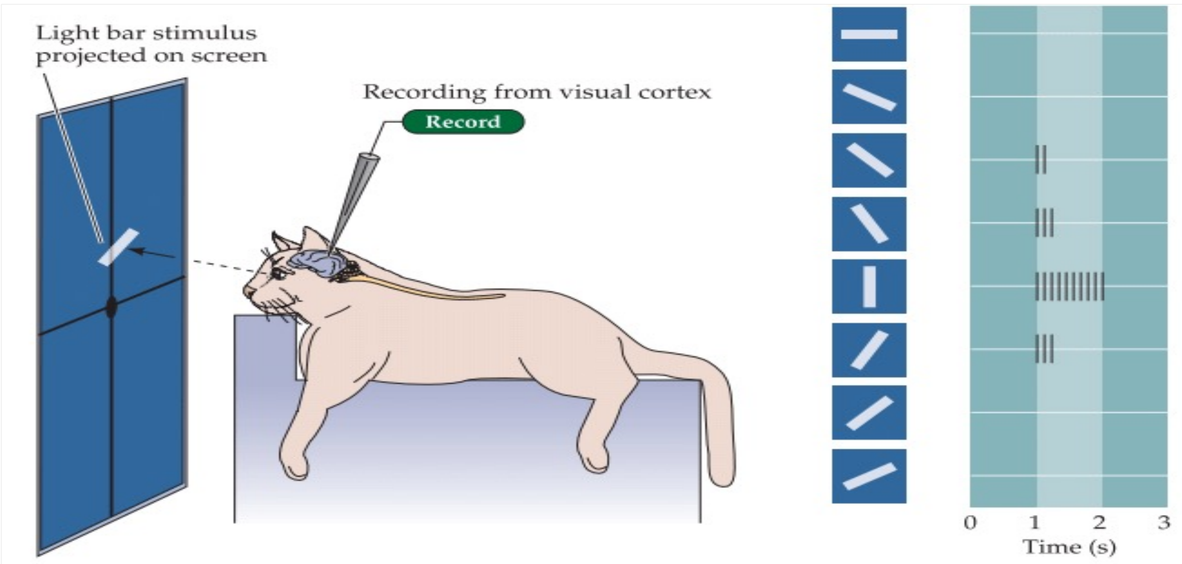
CNN

Convolutional Neural Network

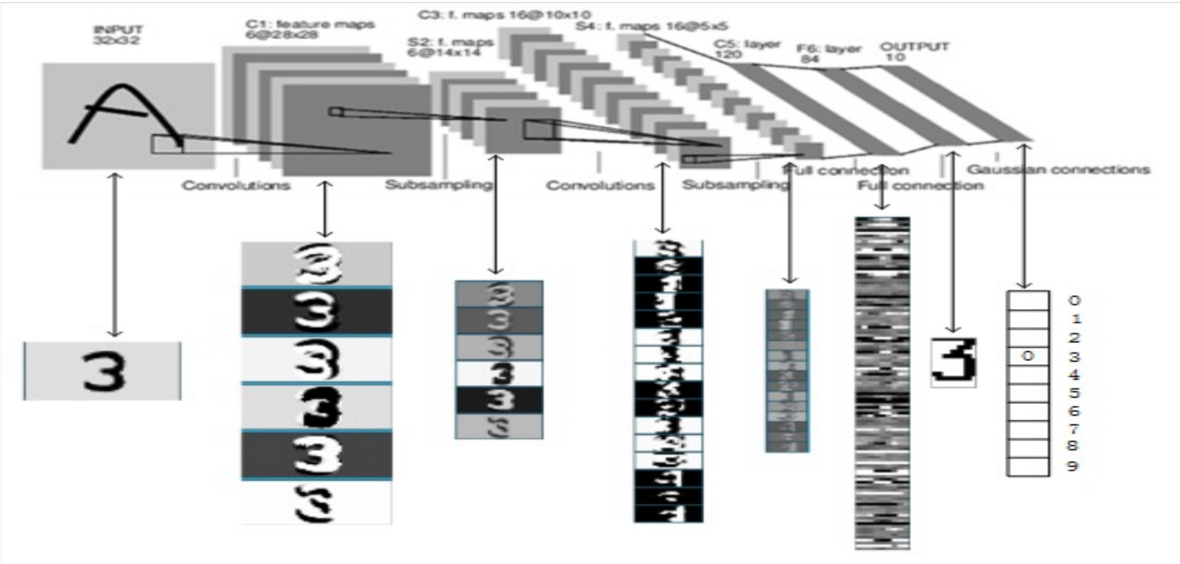
1

Convolutional Neural Networks



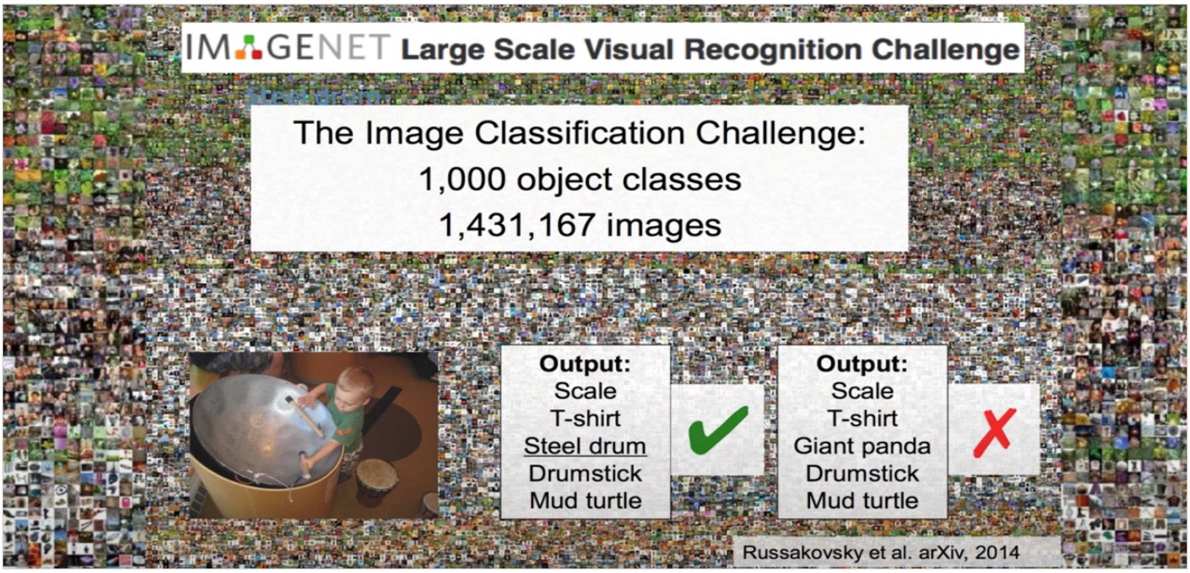
2

Convolutional Neural Networks



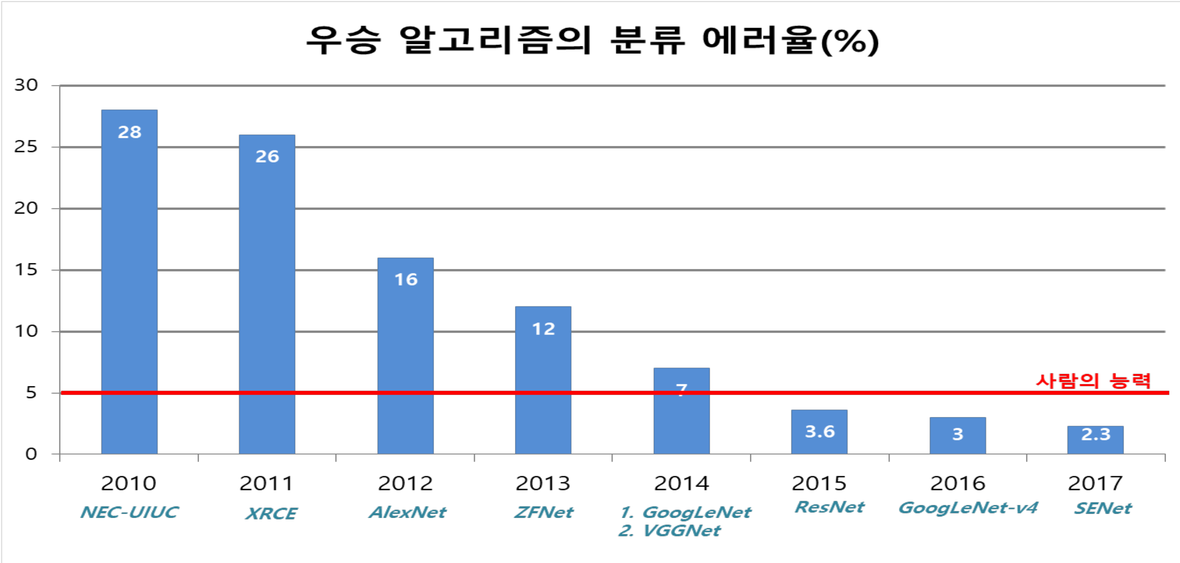
3

IMAGENET



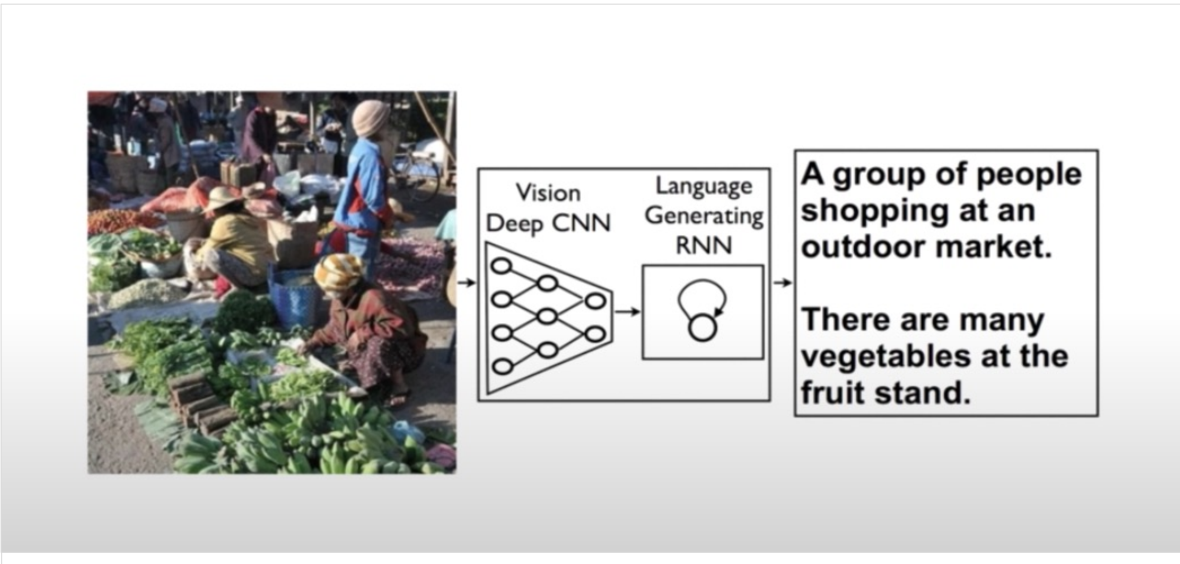
4

ImageNet Classification Error



5

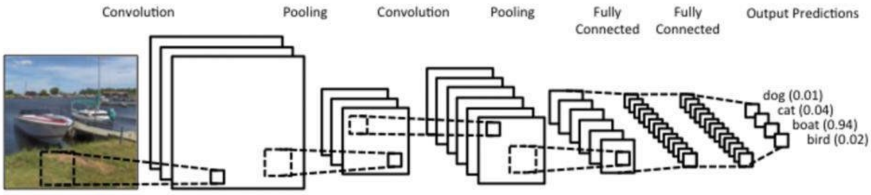
Neural networks that can explain photos



6

Convolutional Neural Network

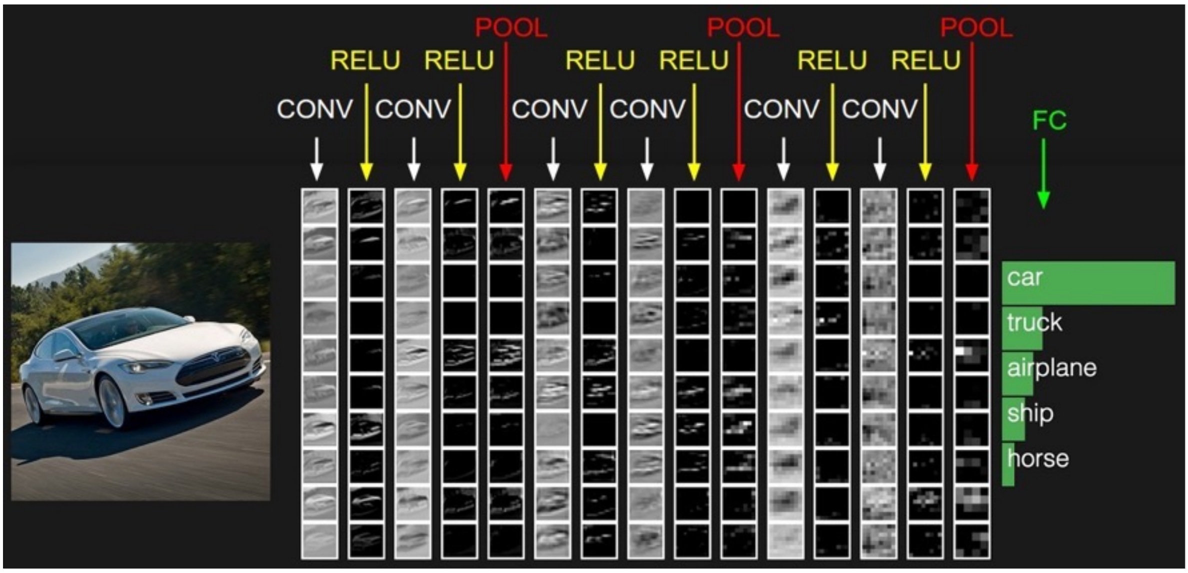
- Most widely used for image classification.
- Generally, it consists of convolution layer, pooling layer and fully-connected layer.
- Weight(parameter, filter, kernel) sharing
- Convolution, Pooling layer - feature extraction
- Fully-connected layer - classification



https://www.kdnuggets.com/2015/11/understanding-convolutional-neural-networks-nlp.html

7

Convolutional Neural Network

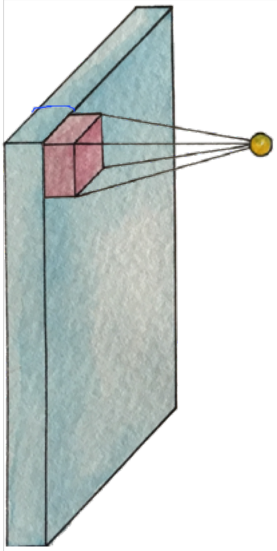


8

Get one number using the filter

'?'에 들어갈 숫자는?

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_?x_? + b$$



One number! $= Wx + b \qquad = ReLU(Wx + b)$

5 x 5 x 3 filter

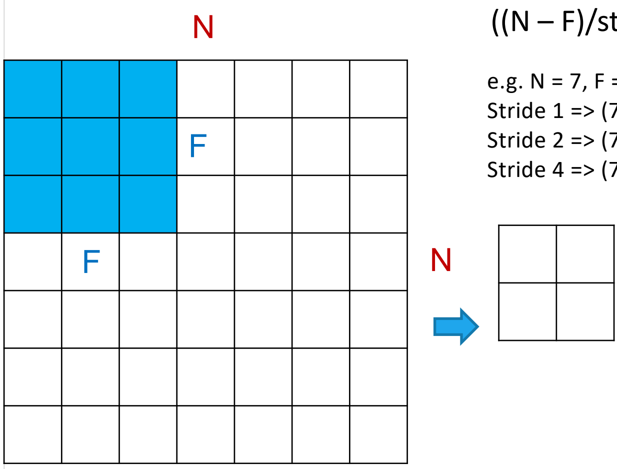
32 x 32 x 3 image

9

A closer look at spatial dimensions

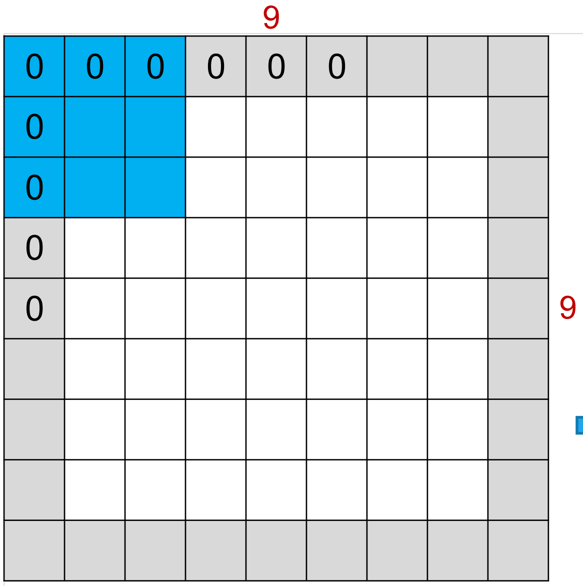
Output size:
((N – F)/stride) + 1

e.g. N = 7, F = 3:
Stride 1 => (7 – 3) / 1 + 1 = 5
Stride 2 => (7 – 3) / 2 + 1 = 3
Stride 4 => (7 – 3) / 4 + 1 = 2



10

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with stride 1

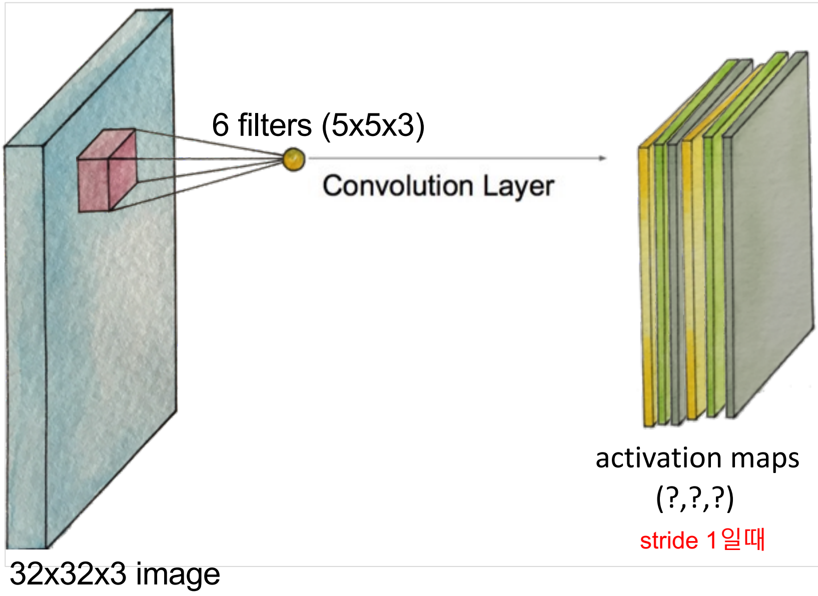
Pad with 1 pixel border => what is the output?

(9 – 3)/1 + 1 = 7

결국 다시 7 x 7이 된다!

11

Swiping the entire image



12

Convolution layers

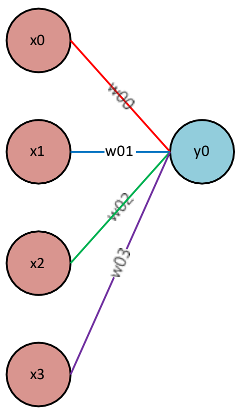
- How many weight variables? How to set them?

13

Dense Layer vs 1-D Convolution Layer

Dense Layer(Fully Connected Layer)

- $y0 = x0 \cdot w00 + x1 \cdot w01 + x2 \cdot w02 + x3 \cdot w03$

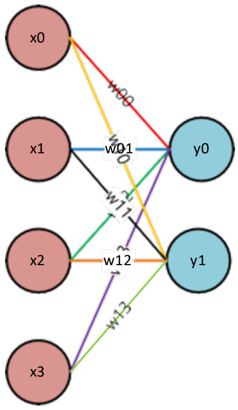


14

Dense Layer vs 1-D Convolution Layer

Dense Layer(Fully Connected Layer)

- $y0 = x0 \cdot w00 + x1 \cdot w01 + x2 \cdot w02 + x3 \cdot w03$
- $y1 = x0 \cdot w10 + x1 \cdot w11 + x2 \cdot w12 + x3 \cdot w13$

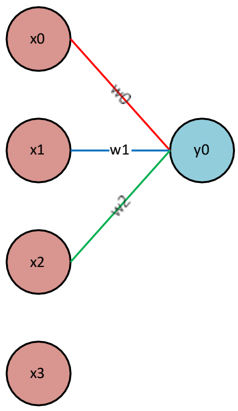


15

Dense Layer vs 1-D Convolution Layer

1-D Convolution Layer

- $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$



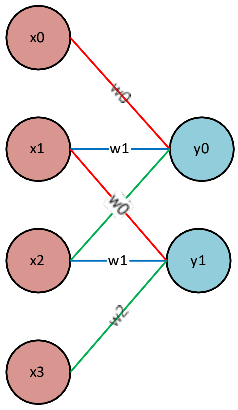
16

Dense Layer vs 1-D Convolution Layer

1-D Convolution Layer

- $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$
- $y0 = x1 \cdot w0 + x2 \cdot w1 + x3 \cdot w2$

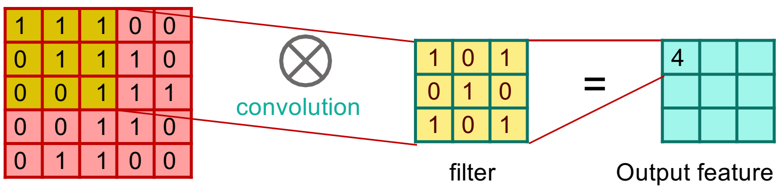
Weight sharing
&
Locally connected



17

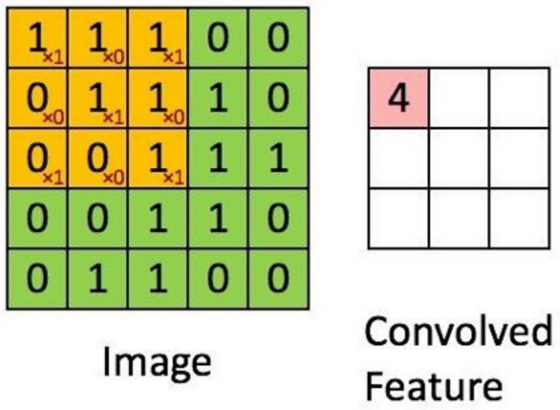
2D Convolution Layer – Computation

- 1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 0x0 + 1x1 =4



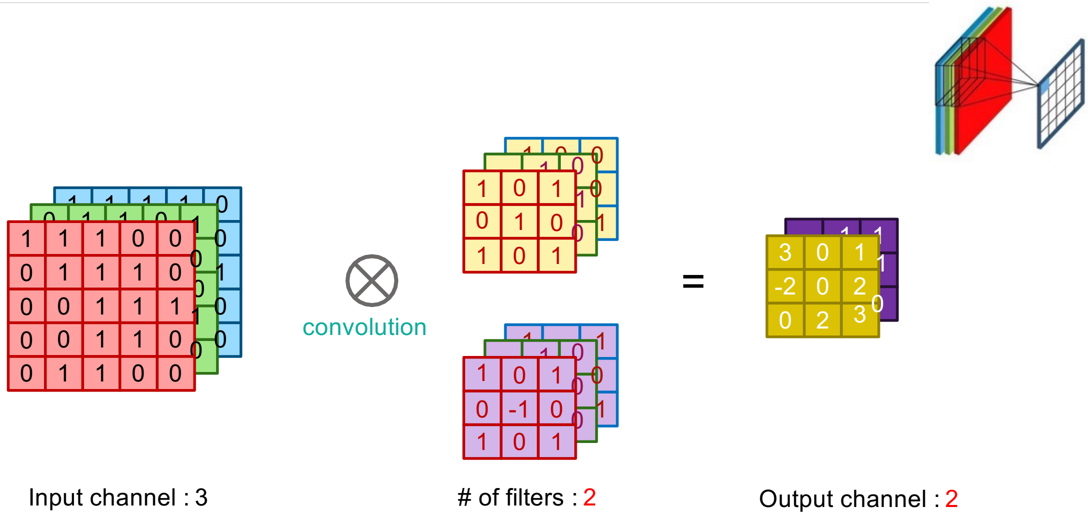
18

2D Convolution Layer – Computation



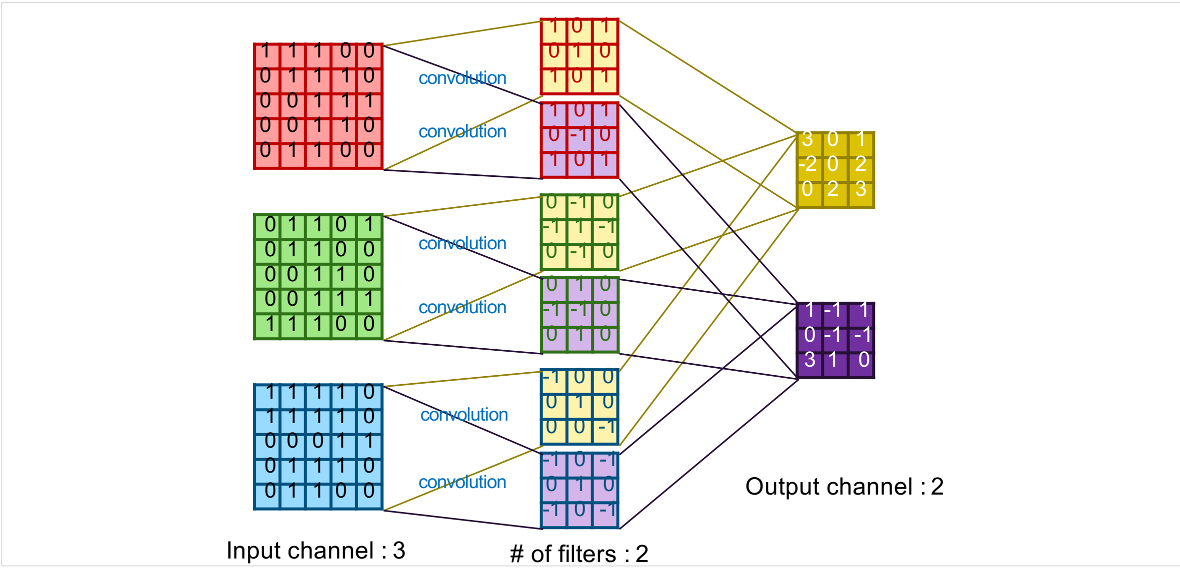
19

2D Convolution Layer – Multi Channel, Many Filters



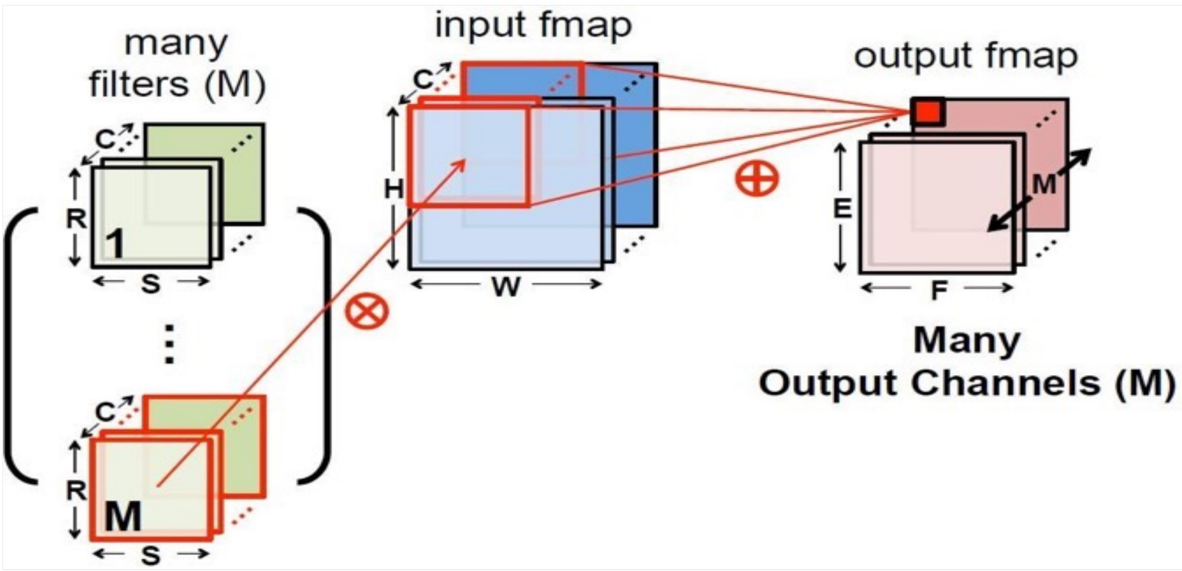
20

2D Convolution Layer – Multi Channel, Many Filters



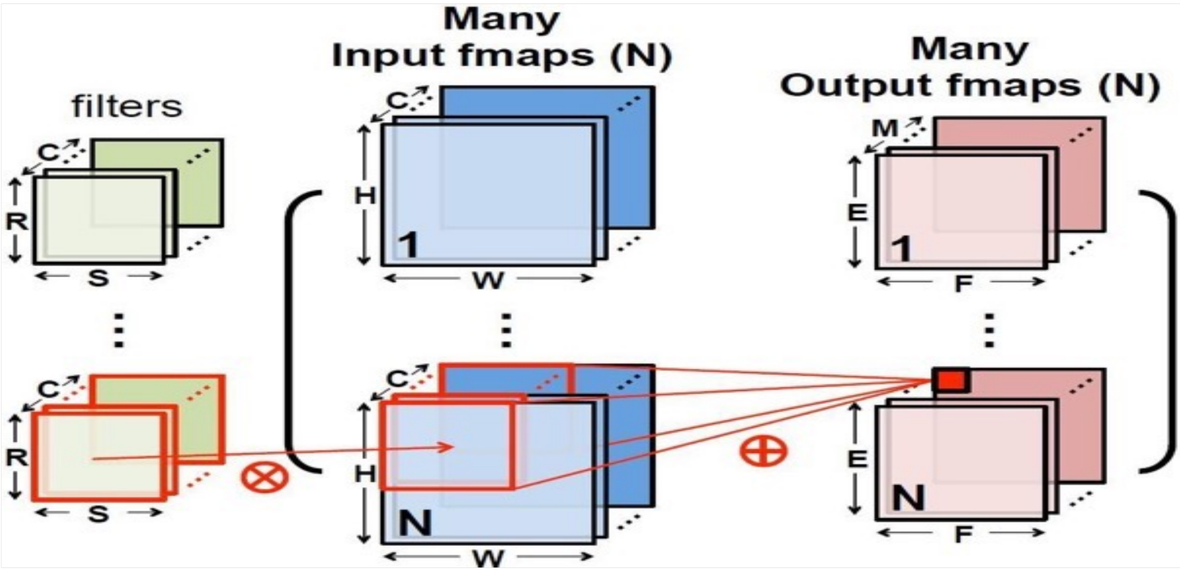
21

2D Convolution Layer



22

2D Convolution Layer – 4D Tensors

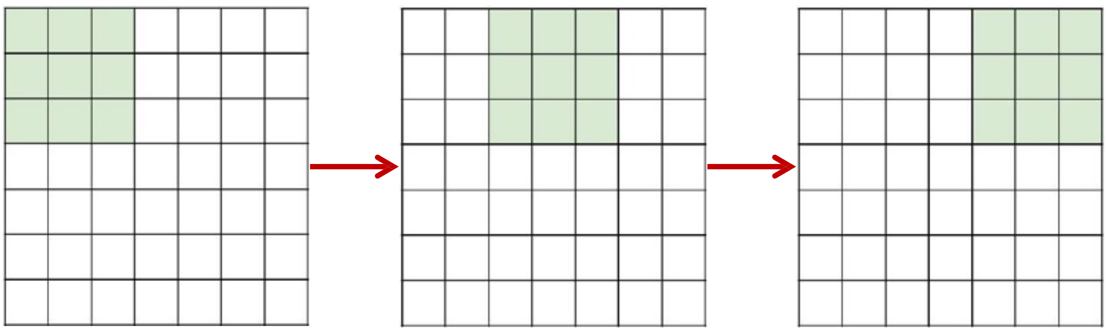


23

Options of Convolution

Stride -How far to go to the right or the bottom to perform the next convolution

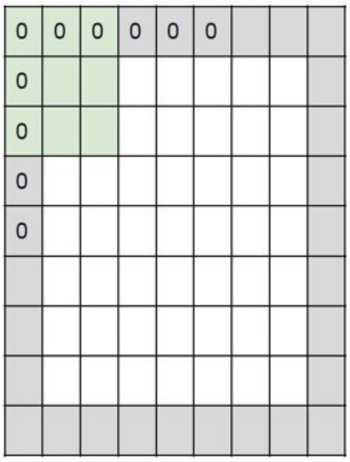
Ex) 7x7 input, 3x3 convolution filter with stride 2 € 3x3 output



24

Options of Convolution

- Zero Padding



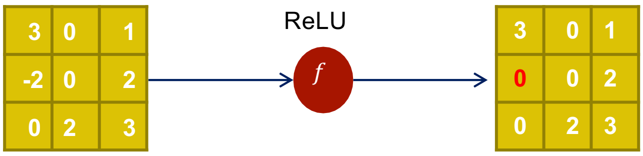
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

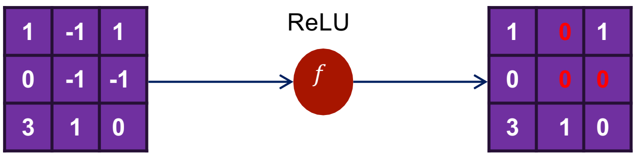
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

25

Activation Function

- ReLU





26

tf.keras.layers.Conv2D

- *tf.keras.layers.Conv2D(*
- *filters,*
- *kernel_size,*
- *strides=(1, 1),*
- *padding="valid",*
- *data_format=None,*
- *dilation_rate=(1, 1),*
- *groups=1,*
- *activation=None,*
- *use_bias=True,*
- *kernel_initializer="glorot_uniform",*
- *bias_initializer="zeros",*
- *kernel_regularizer=None,*
- *bias_regularizer=None,*
- *activity_regularizer=None,*
- *kernel_constraint=None,*
- *bias_constraint=None)*

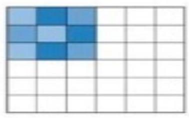
27

tf.keras.layers.Conv2D

- `filters` : Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size` : An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `strides` : An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding` : one of `"valid"` or `"same"` (case-insensitive).
- `data_format` : A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

28

Padding – SAME vs VALID

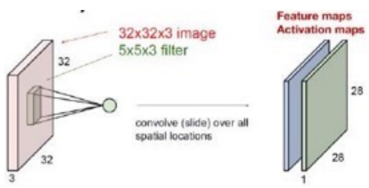
	Valid	Same
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$
Illustration		
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ - Output size is mathematically convenient - Also called 'half' padding

29

tf.keras.layers.Conv2D

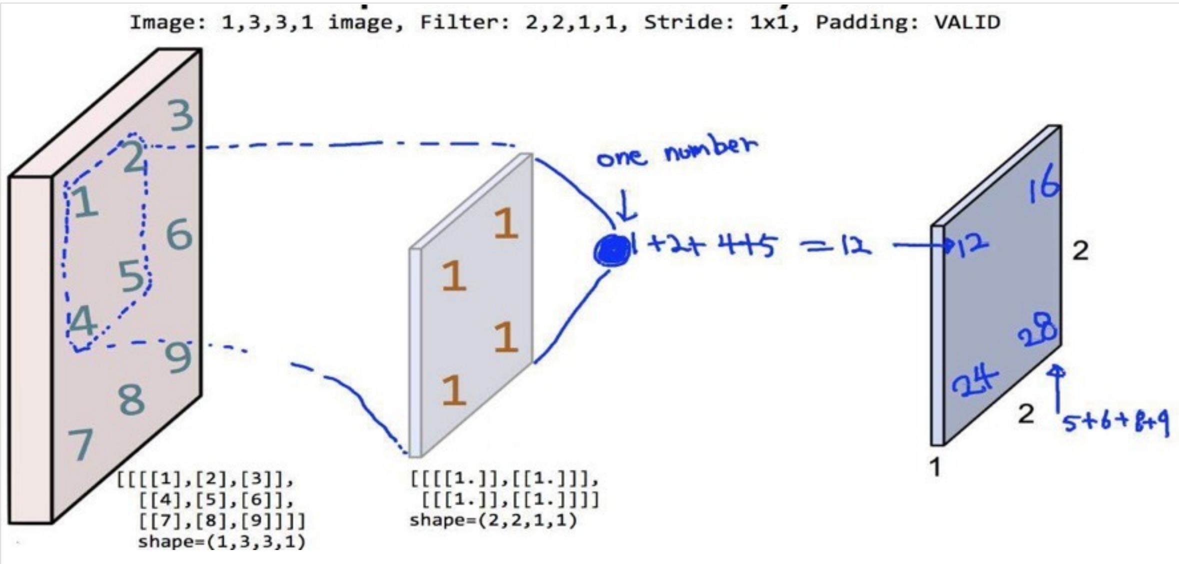
- `activation` : Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias` : Boolean, whether the layer uses a bias vector.
- `kernel_initializer` : Initializer for the `kernel` weights matrix.
- `bias_initializer` : Initializer for the bias vector.
- `kernel_regularizer` : Regularizer function applied to the `kernel` weights matrix.
- `bias_regularizer` : Regularizer function applied to the bias vector.

kernel dimension : {height, width, in_channel, out_channel} Ex) {5, 5, 3, 2}



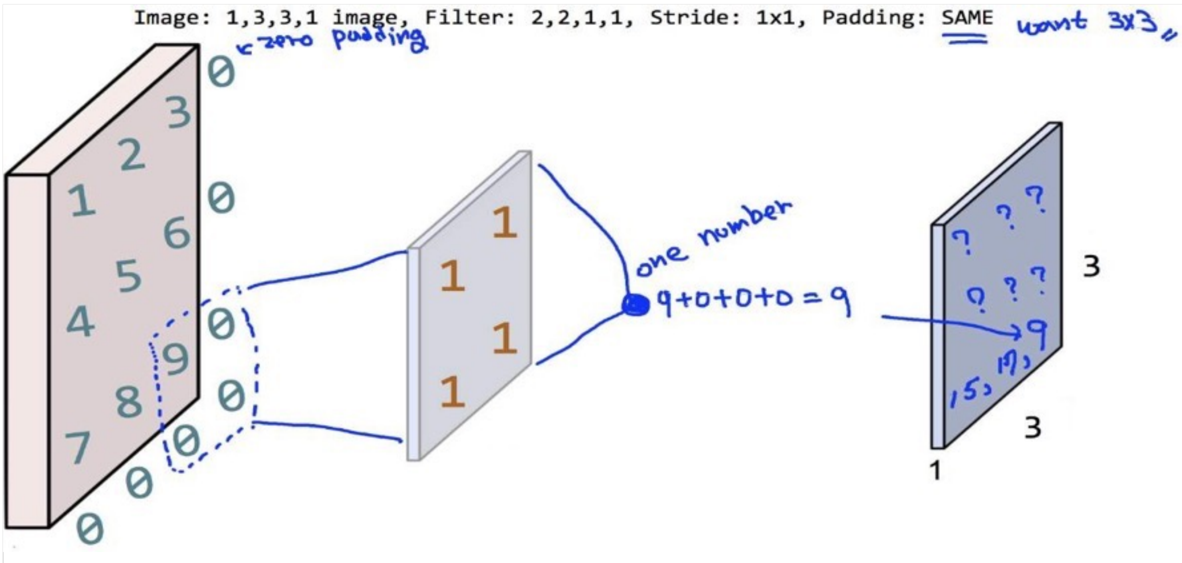
30

Simple Convolution Layer



31

Simple Convolution Layer

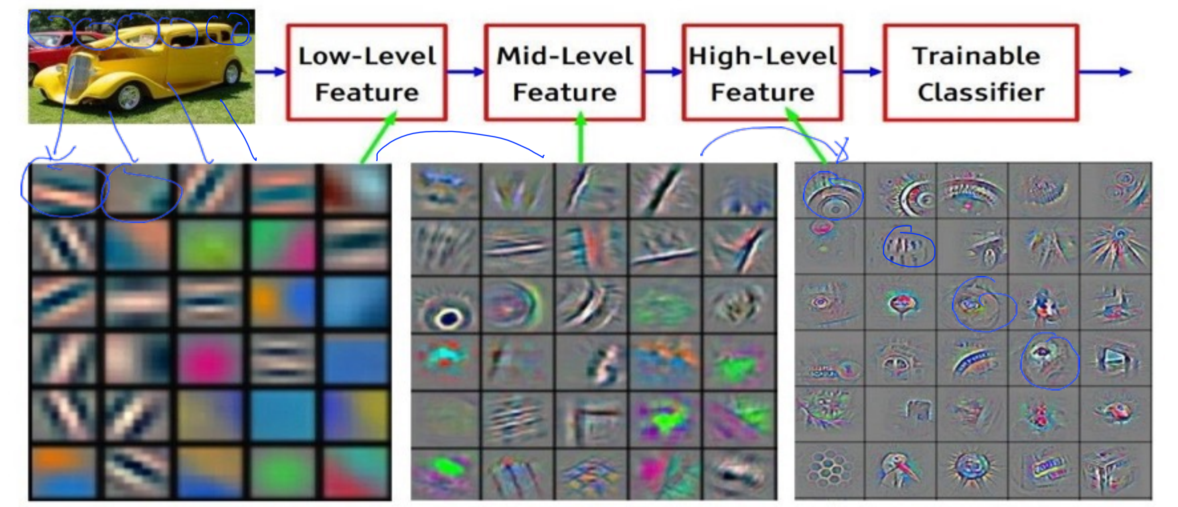


32

Pooling

33

Preview

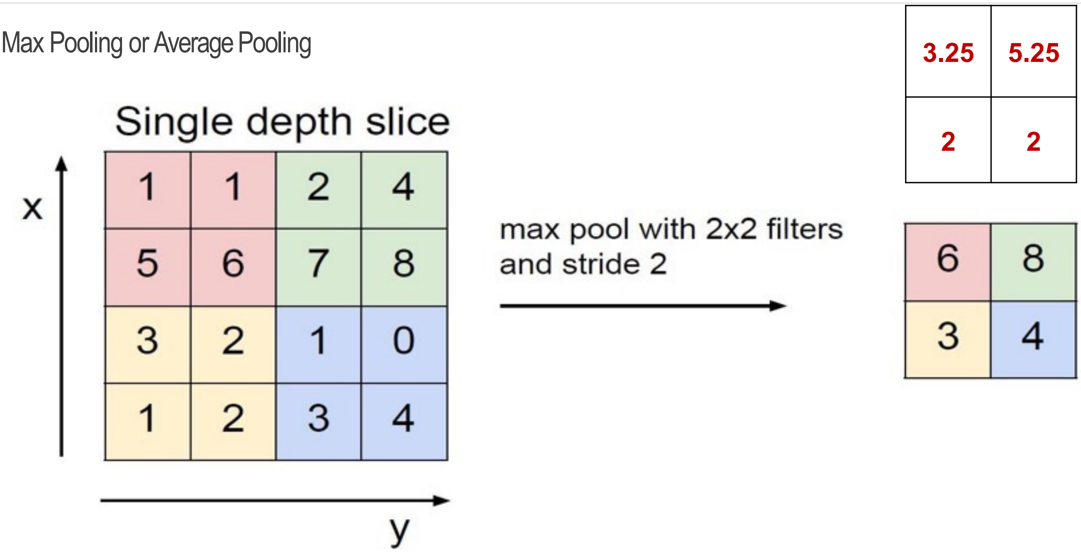


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

34

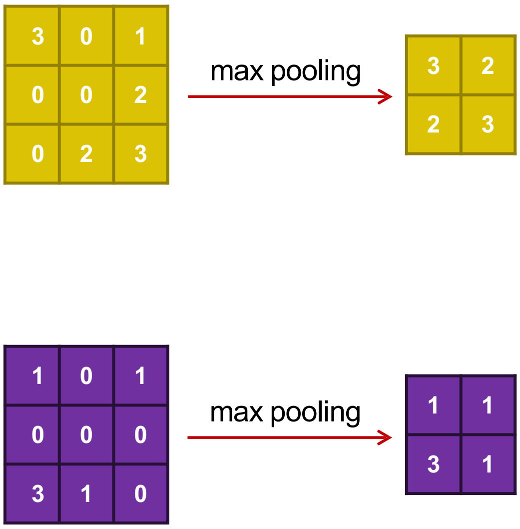
Pooling

- Max Pooling or Average Pooling



35

Pooling (max pooling, 2x2 filter, stride 1)



36

tf.keras.layers.MaxPool2D

```
__init__(
    pool_size=(2, 2),
    strides=None,
    padding='valid',
    data_format=None,
    **kwargs
)
```

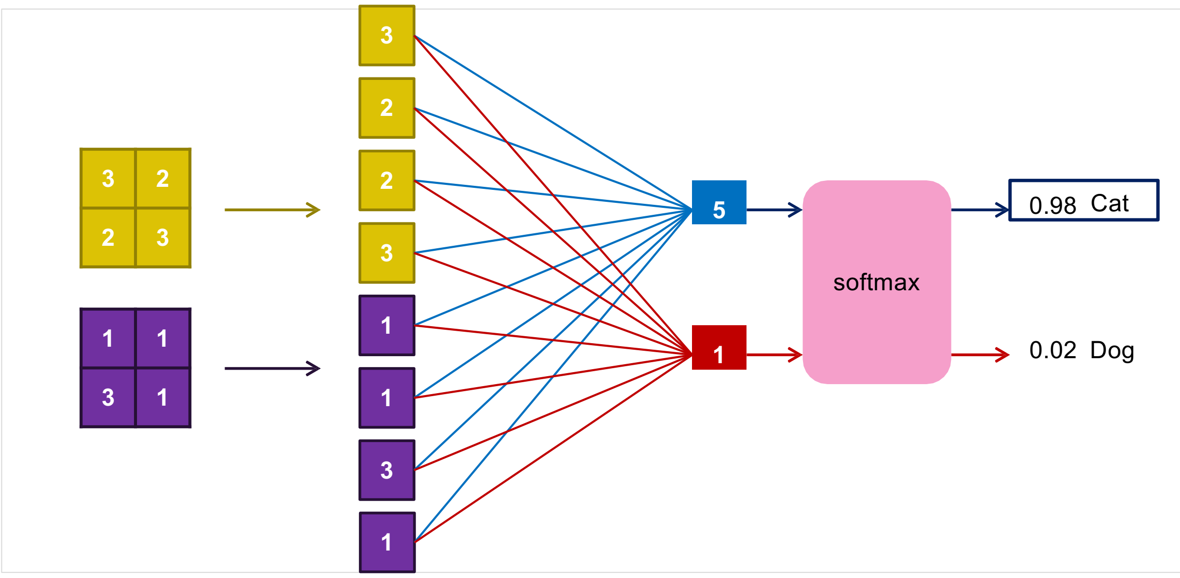
37

tf.keras.layers.MaxPool2D

- `pool_size` : integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
- `strides` : Integer, tuple of 2 integers, or None. Strides values. If None, it will default to `pool_size` .
- `padding` : One of `"valid"` or `"same"` (case-insensitive).
- `data_format` : A string, one of `channels_last` (default) or `channels_first` . The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)` . It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json` . If you never set it, then it will be "channels_last".

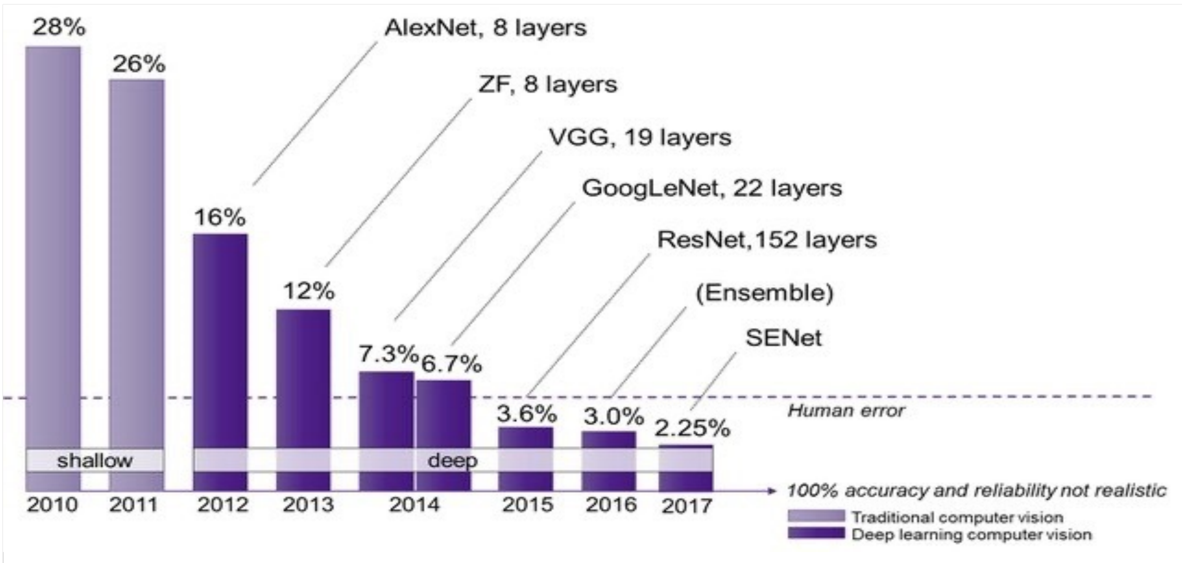
38

Fully Connected(Dense) Layer



39

IMAGENET

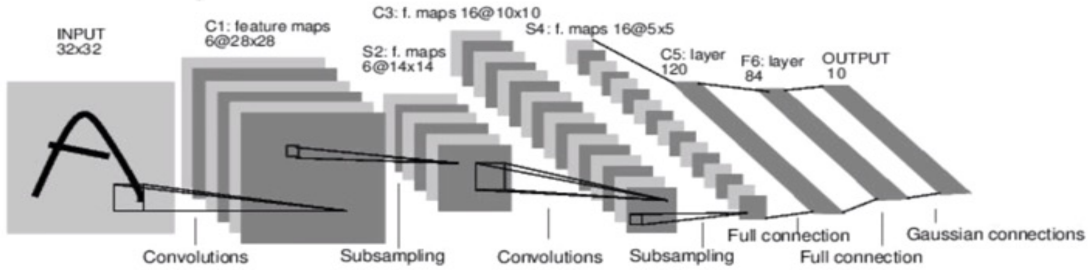


40

CNN Case study

Case Study: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

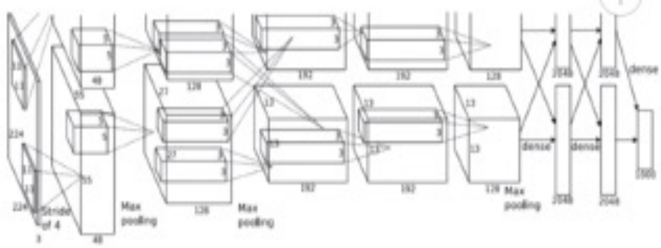
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 60 27 Jan 2016

41

CNN Case study

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

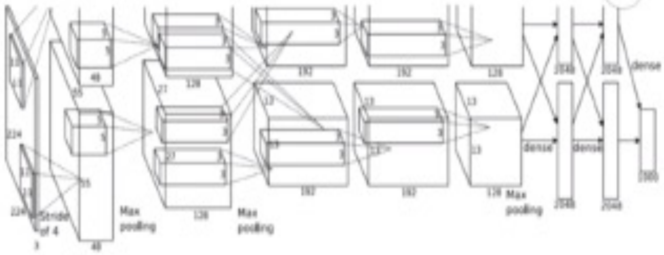
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

42

CNN Case study

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

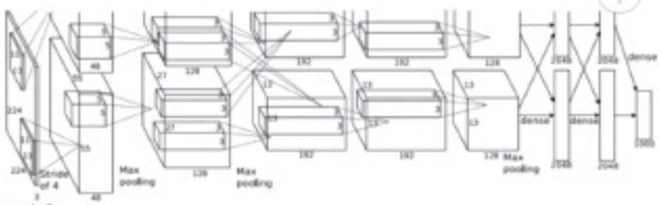
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

43

CNN Case study

Case Study: AlexNet

[Krizhevsky et al. 2012]

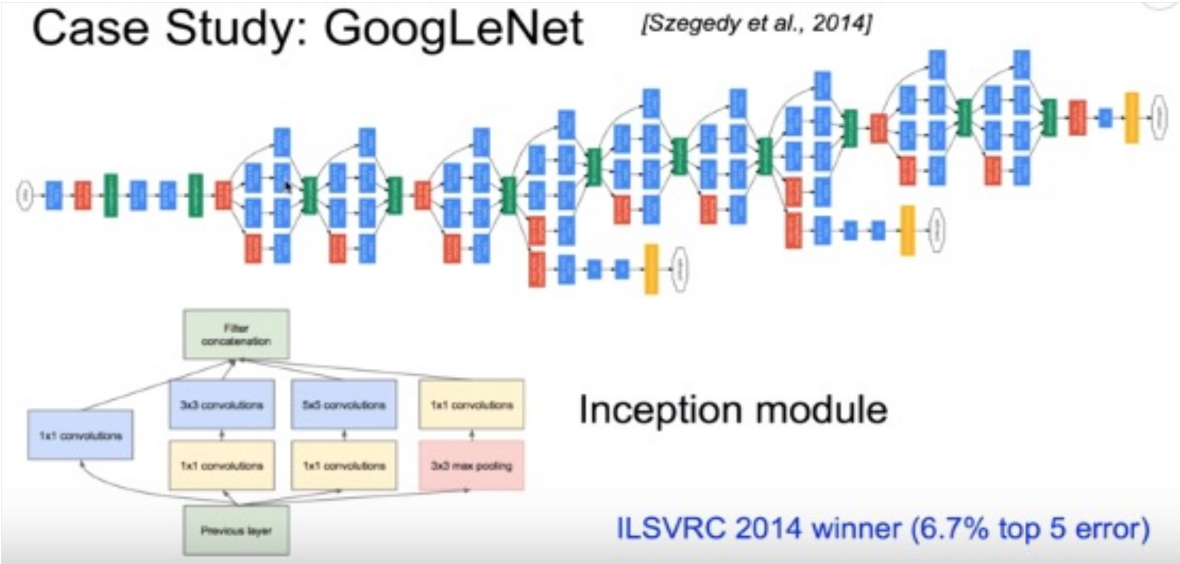


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

44

CNN Case study



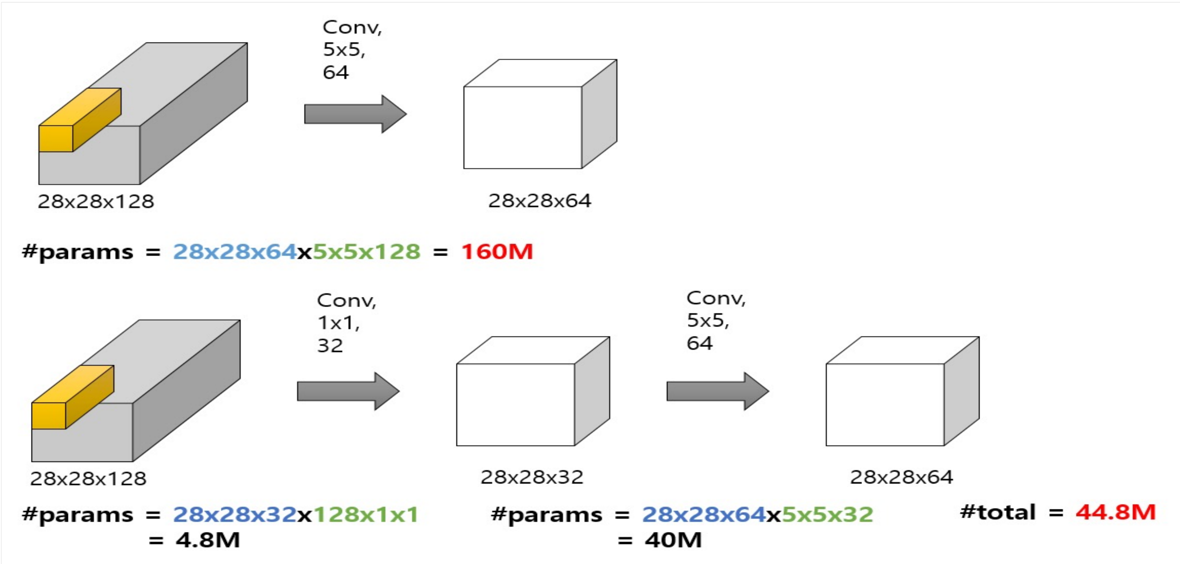
45

CNN Case study

- 1x1 Convolution 장점 세 가지
    1) Channel 수 조절
    2) 연산량 감소 (Efficient)
    3) 비선형성 (Non-linearity)

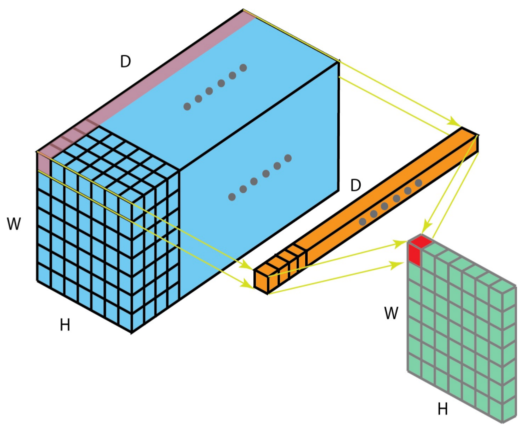
46

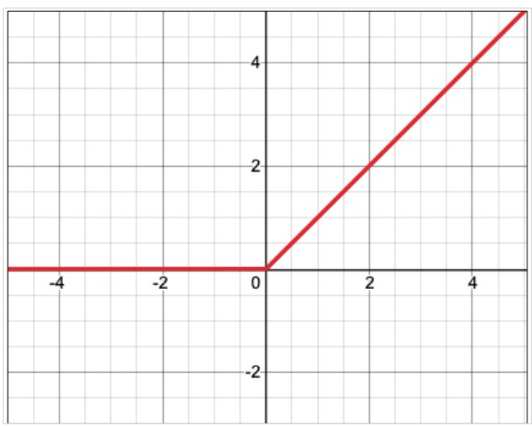
2) 연산량 감소



47

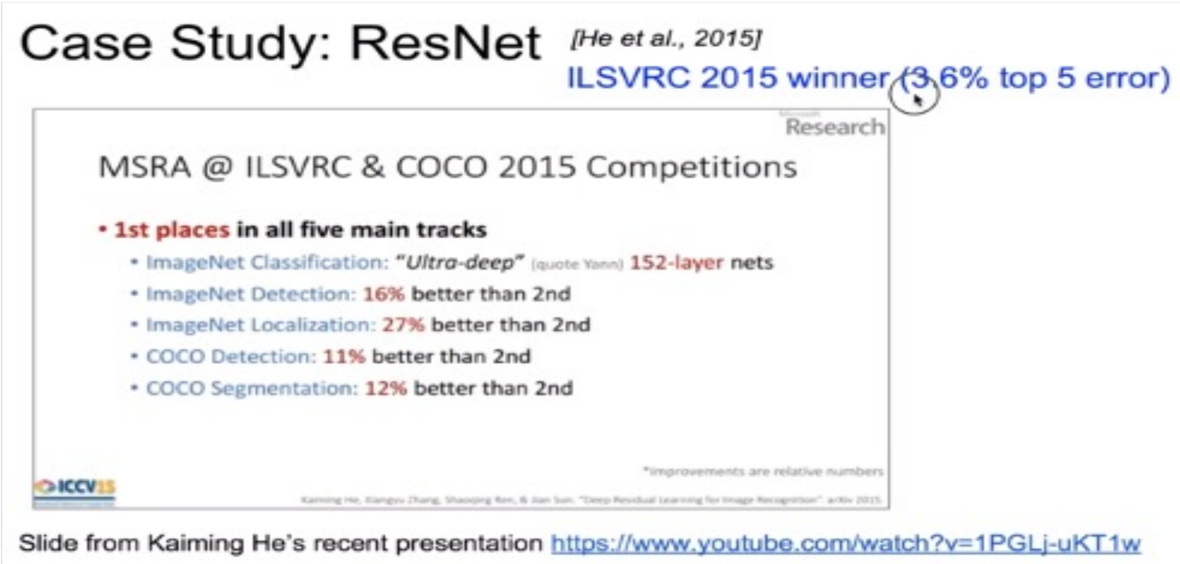
3) 비선형성





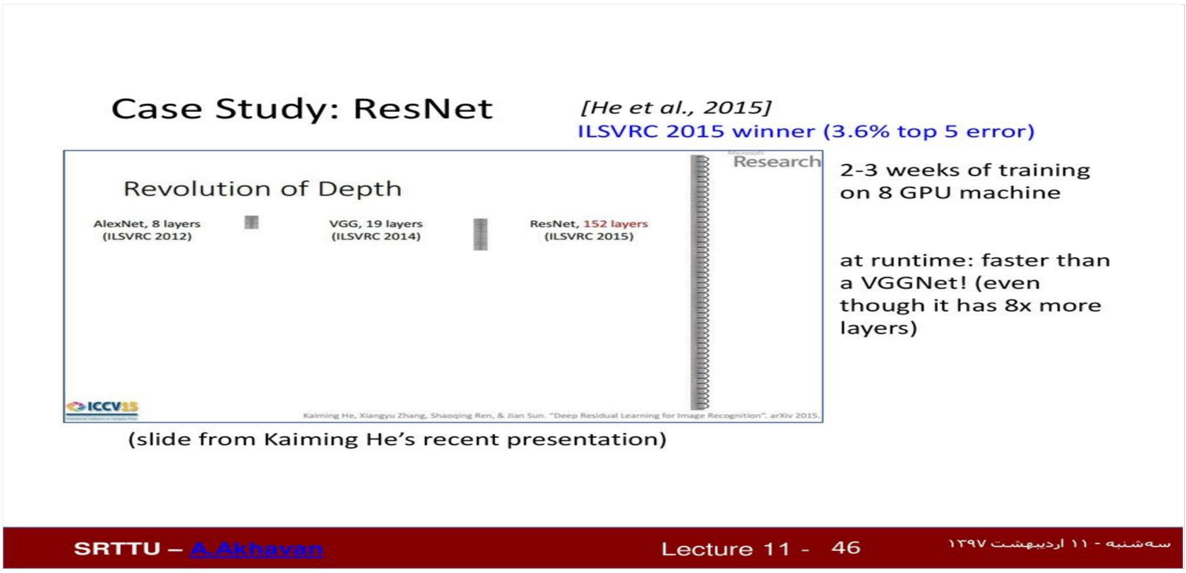
48

CNN Case study



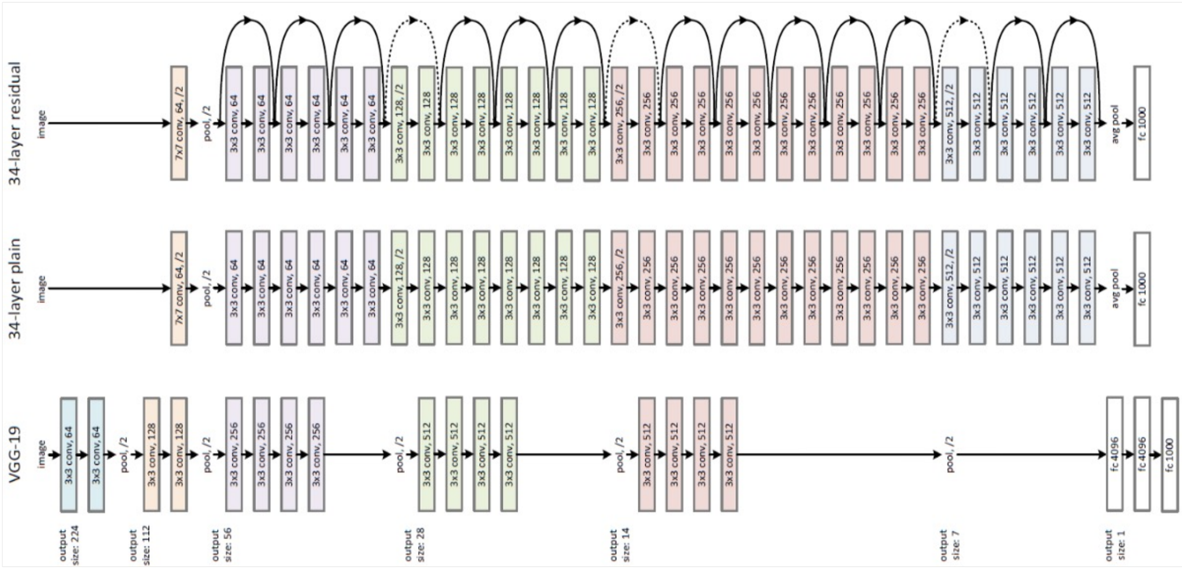
49

CNN Case study



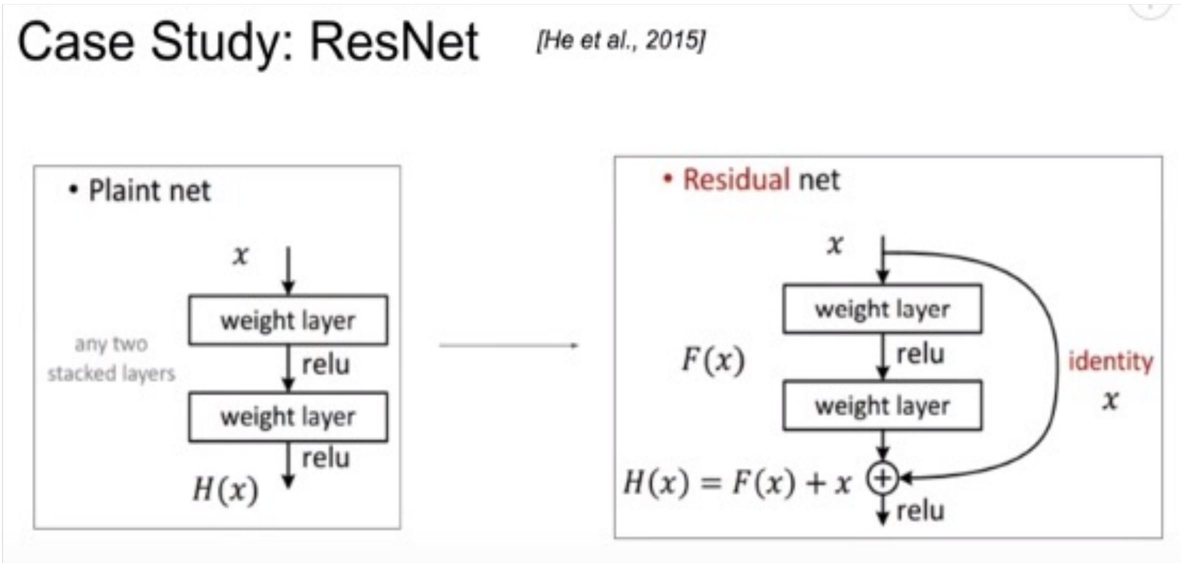
50

CNN Case study



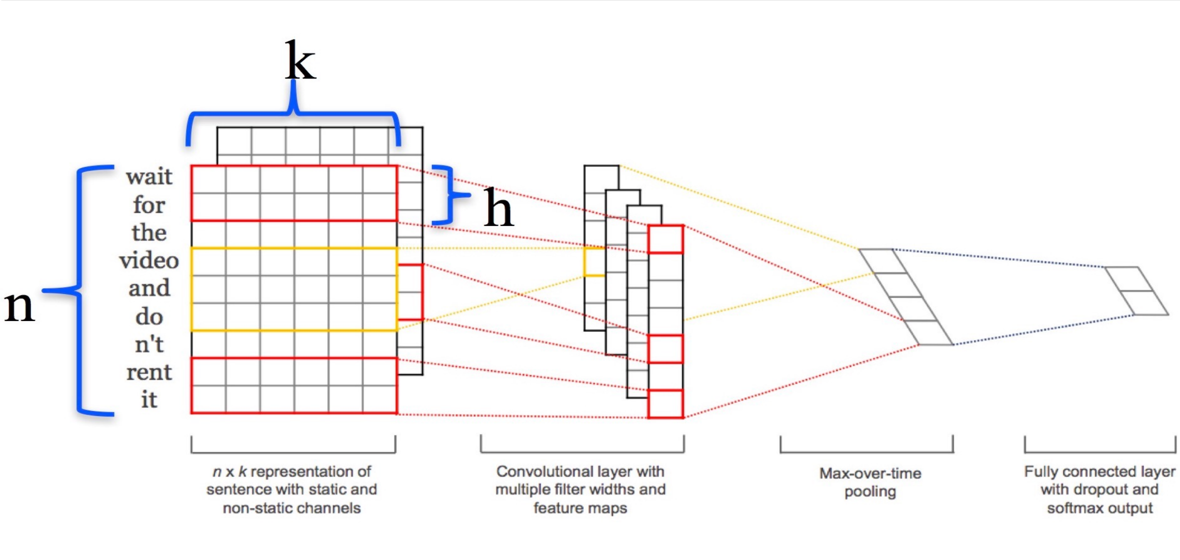
51

CNN Case study



52

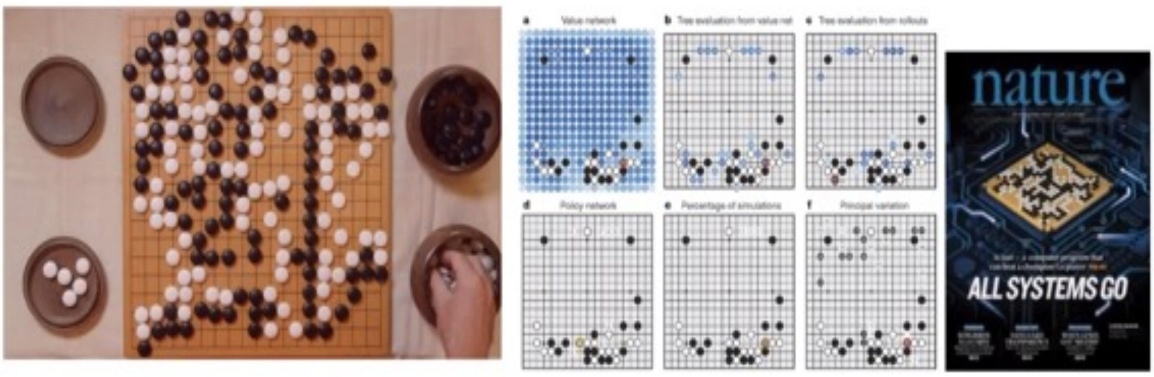
CNN Case study



53

CNN Case study

Case Study Bonus: DeepMind's AlphaGo



54

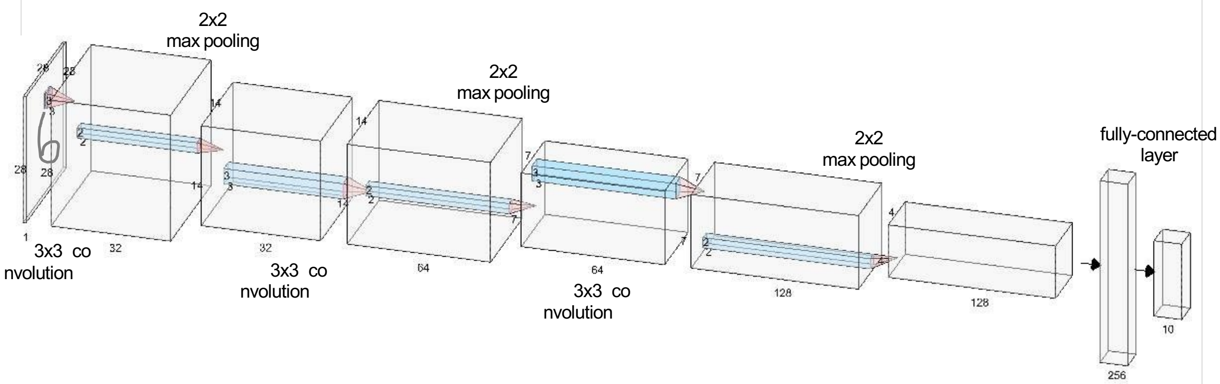
CNN Case study

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)*

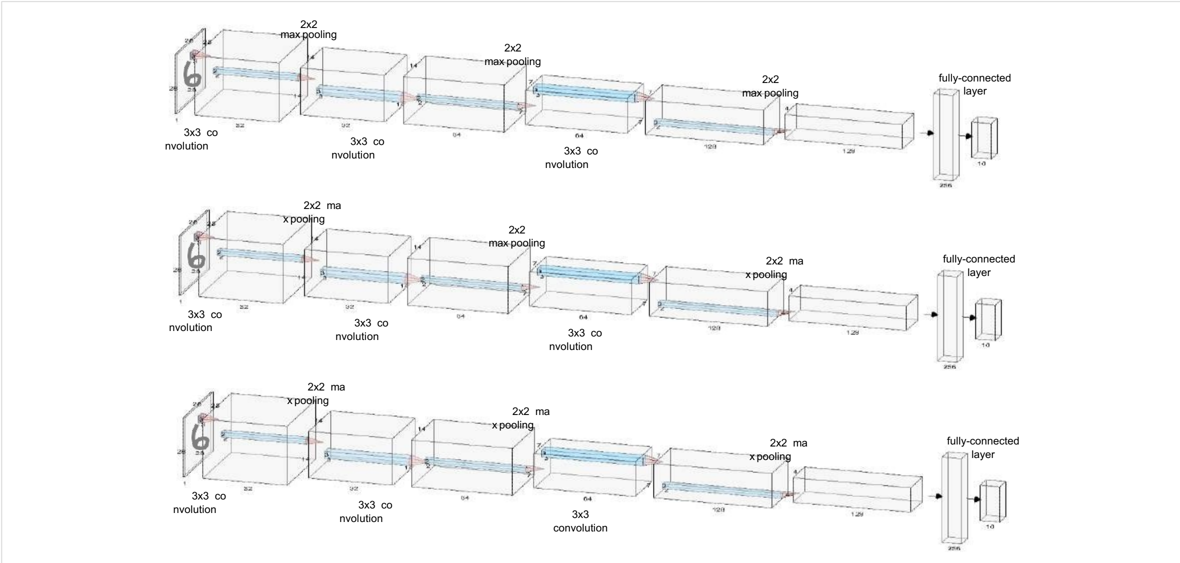
55

CNN with MNIST Data



56

CNN Model Ensemble with MNIST Data



57