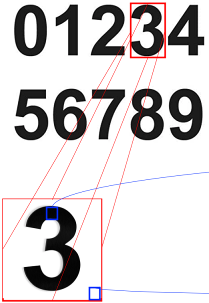
IFI 9000 Analytics Methods
Deep Learning in Computer Vision

by **Houping Xiao**

Spring 2021

Introduction

Computer vision is everywhere



Consider an image as a function:

$f(x, y): [a, b] \times [c, d] \rightarrow [0, 255]$

```
array([[255., 255., 255., 255., 252., 255., 255., 255., 255., 255.],
       [255., 255., 253.,   7.,   4.,   4.,  14., 255., 255., 255.],
       [255., 255., 123.,   7., 250.,   6.,   7., 255., 255., 255.],
       [255., 255., 249.,   6., 255., 252.,  13., 189., 255., 255.],
       [255., 255., 255., 255., 255.,  32.,  16., 255., 255., 255.],
       [255., 255., 255., 255.,  14.,   9., 253., 255., 255., 255.],
       [255., 255., 255., 255., 255., 255.,   7.,   6., 255., 255.],
       [255., 255., 255., 255., 255., 255.,   7.,  11., 255., 255.],
       [255., 255.,   8.,   4., 255., 255.,   7.,   3., 255., 255.],
       [255., 255.,  10.,   5.,   8.,   8.,   6.,  57., 255., 255.],
       [255., 255., 253.,   4.,  11.,  10.,  13., 255., 255., 255.],
       [255., 255., 255., 255., 255., 255., 255., 255., 255., 255.]],
      dtype=float32)
```

Image representation: from gray-scale images to colorful images

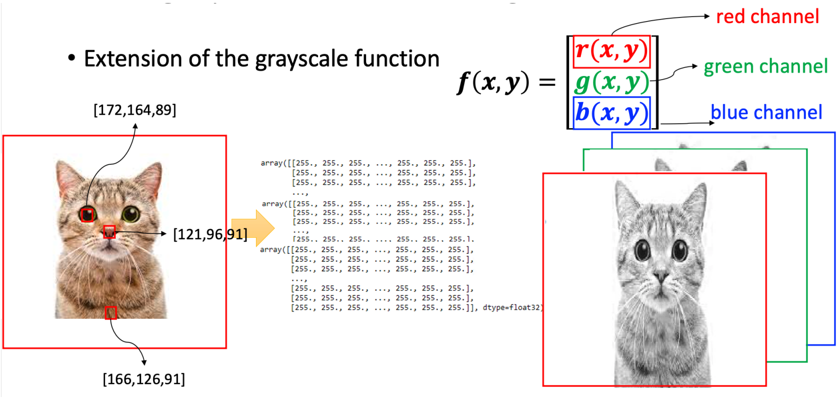


Image representation: a summary

- An image can be represented as a matrix or a tensor (a higher order matrix, usually 3-d in here) of pixel values

```
157, 153, 174, 168, 150, 152, 129, 151, 172, 161, 156, 156
156, 182, 163,  74,  76,  62,  33,  17, 110, 210, 180, 154
180, 180,  50,  14,  34,   6,  10,  33,  48, 106, 159, 181
206, 109,   6, 124, 131, 111, 120, 204, 166,  15,  56, 180
194,  68, 137, 251, 237, 239, 239, 228, 227,  87,  71, 201
172, 106, 207, 233, 233, 214, 220, 239, 228,  98,  74, 206
188,  88, 179, 209, 186, 216, 211, 158, 139,  75,  20, 169
189,  97, 166,  84,  10, 168, 134,  11,  31,  62,  22, 148
199, 168, 191, 193, 158, 227, 178, 143, 182, 106,  36, 190
206, 174, 156, 252, 236, 231, 149, 178, 228,  43,  96, 234
190, 216, 116, 149, 236, 187,  86, 150,  79,  38, 218, 241
190, 224, 147, 108, 227, 210, 127, 102,  36, 101, 256, 224
190, 214, 173,  66, 103, 143,  96,  50,   2, 109, 249, 215
187, 196, 236,  75,   1,  81,  47,   0,   6, 217, 256, 211
183, 202, 237, 145,   0,   0,  12, 108, 200, 138, 243, 236
196, 206, 123, 207, 177, 121, 123, 200, 176,  13,  96, 218
```

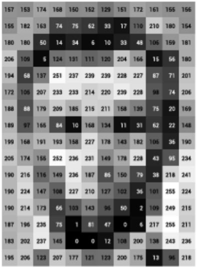


Image processing: image filtering and image warping

- **Image filtering:** change the range, i.e. the pixel values, of an image such that the colors of the image are changed without changing the pixel positions
- **Image warping:** change the domain, i.e. the pixel positions, of an image, where points are mapped to other positions without changing the colors

Image filtering: change the pixel values

- An example using "median filter", replacing each entry with the median of neighboring entries
- used to remove noise from an image or signal
- preserves edges while removing noise

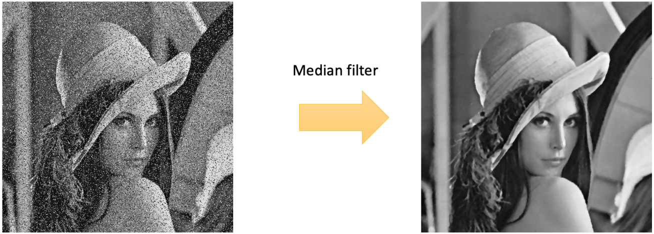


Image filtering: moving average

- A more smooth image with sharp features removed
- replace each pixel with the average pixel value of it and its neighborhood window of adjacent pixels

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	90	90	90	90	0	0
0	0	90	0	0	90	0	0
0	0	90	0	0	90	0	0
0	0	90	90	90	90	0	0
0	0	0	0	0	0	0	0

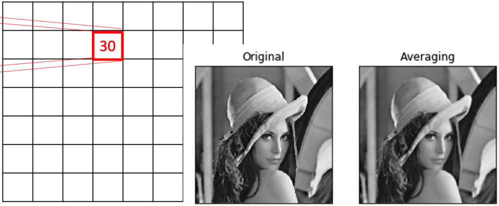


Image filtering: image segmentation filters

- Partition an image into regions where the pixels have similar attributes, so the image is represented in a more simplified way
- identify objects and boundaries more easily

$$f(x,y) = \begin{cases} 255, if\ f(x,y) > 100 \\ 0, otherwise \end{cases}$$

2d convolution filter: works on a input and a kernel image

- Filters can be expressed in a principal manner using 2d convolution, such as smoothing and sharpening images, and detecting edges

Sharpening filter

- Original image - smoothed image = details
- Original image + details = sharpened image

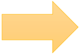
  

An application: edge detection

- Digitally manipulating an image, such as resizing the image (subsampling)
  - any shapes portrayed in the image have been significantly distored

512×512×3

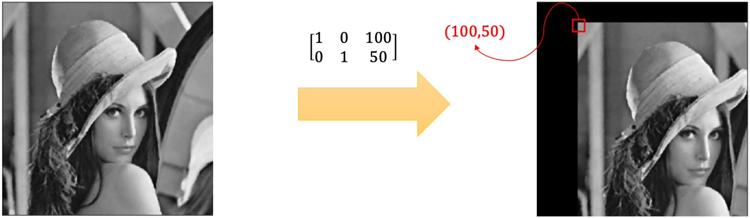
Subsampling/downsampling



256×256×3

Translation

- Shifting of an object location
- transformation matrix: $M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$



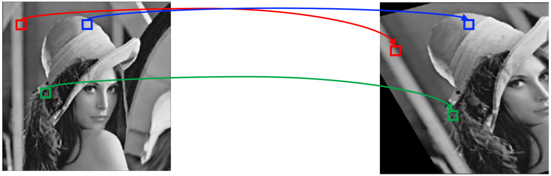
Rotation

- Rotation θ can be achieved by the transofrmation of the form
- $M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ or $M = \begin{bmatrix} \alpha & \beta & (1-\alpha)x - \beta y \\ -\beta & \alpha & \beta x + (1-\alpha)y \end{bmatrix}$, where $\alpha = scale * \cos\theta, \beta = scale * \sin\theta$ and (x, y) is the center

θ=180
Scale = 1
(x,y) is center of image

Affine and perspective transformation

- Affine: similar to rotation, all parallel lines in the original image will still be parallel



- Perspective: zoom out for a specific range defind by four points

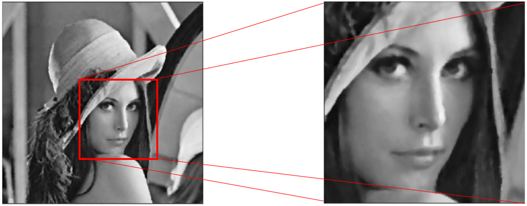
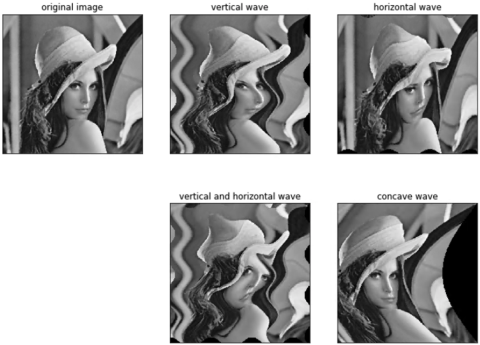
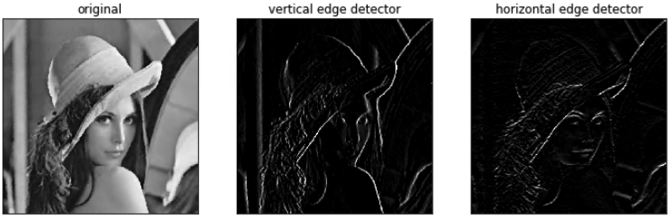


Image warping



Demo on image processing; check the python code!

Filters: a motivating example of edge detection



- vertical edge detection

input 6 x 6

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

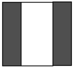
filter 3 x 3

1	0	-1
1	0	-1
1	0	-1

=

output 4 x 4

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



More edge detection filters

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$



More edge detection filters

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

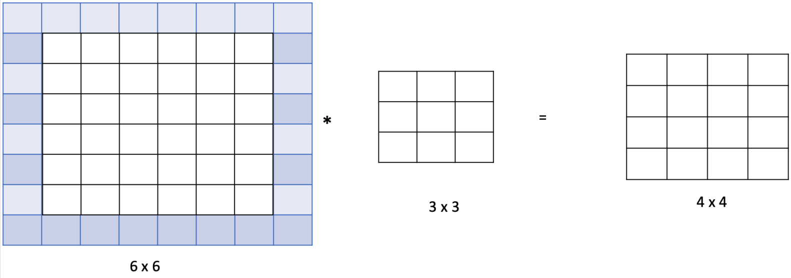
*

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

=

Padding

- Shrinkage output
- through away information from edge



- Options: Valid and Same padding
- Valid: no padding, and output $(n - f + 1) \times (n - f + 1)$
- Same: output feature map stays the same size as the input image (feature map), and output $(n + 2p - f + 1) \times (n + 2p - f + 1)$
  - f usually odd

Stride

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

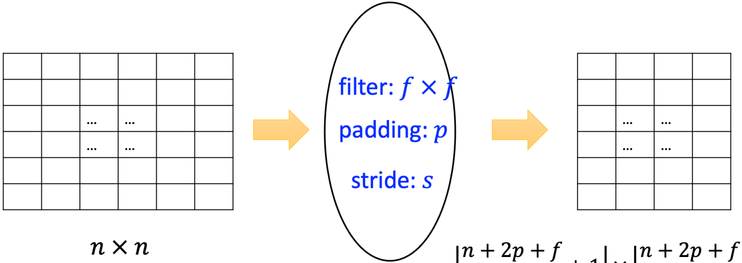
*

3	4	5
1	0	2
-1	0	3

=

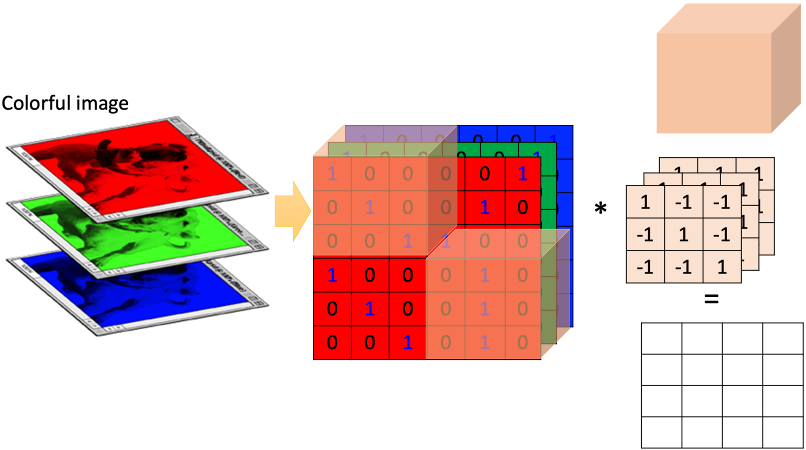
91	100	83
69	91	127
44	72	74

- output: $(\frac{n+2p-f}{s} + 1) \times (\frac{n+2p-f}{s} + 1)$

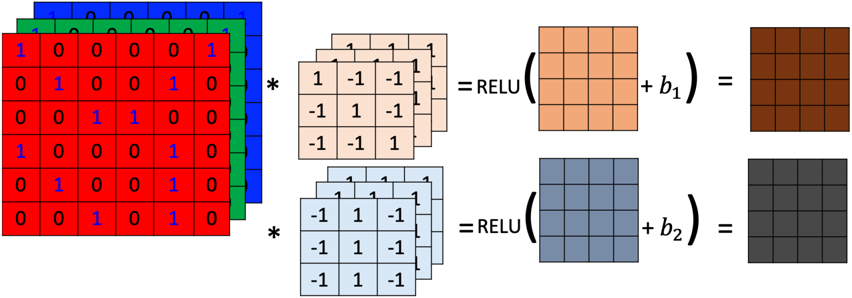


$$\left\lfloor \frac{n + 2p + f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p + f}{s} + 1 \right\rfloor$$

Convolutions on RGB (colorful) images



Convolutions on RGB (colorful) images



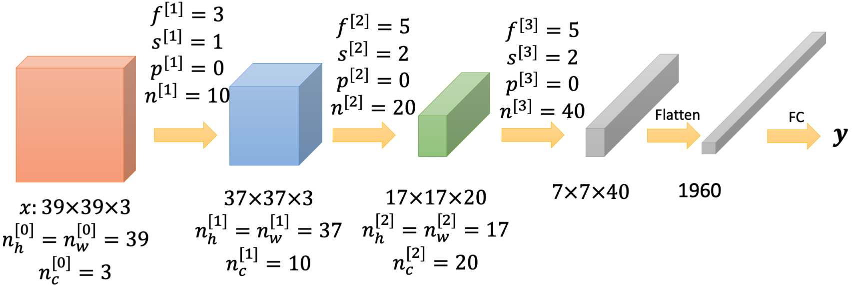
- Number of parameters in one layer
  - consider one convolutional layer with 10 filters that are 333, how many parameters we need to train?

Notations

- If layer l is a conv layer:
- $f^{[l]}$ = filter size
- $p^{[l]}$ = padding
- $s^{[l]}$ = stride
- $n_c^{[l]}$ = number of filters
- Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
- Activations: $\mathrm{a}^{[l]} \rightarrow n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$
- Weights: $\mathrm{f}^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
- Bias: $n_c^{[l]}$

- Input (RGB images): $n_h^{[l-1]} \times n_w^{[l-1]}$
- Output: $n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$
- $n_{h\backslash \mathrm{w}}^{[l]} = \left\lfloor \frac{n_{h\backslash \mathrm{w}}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

An example of ConvNet



- Convolution (ConV), complicated
- Pooling (Pool), easy
- Fully Connected (FC), easy

Pooling layers (Pool)

- reduce the size of image representation, speed up the computation, and robust feature detection

1	3	2	1
2	9	1	2
1	3	3	2
6	5	2	1

Max pool
Hyperparameters:
$f = 2$
$s = 2$

9	2
6	3

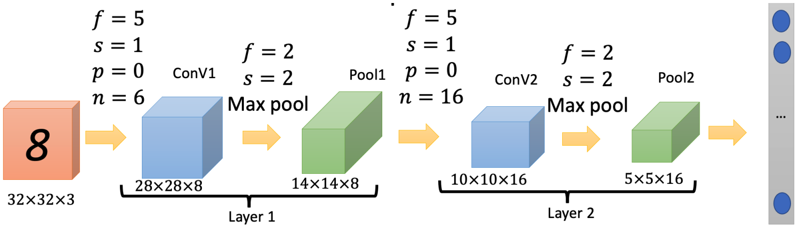
1	3	2	1
2	9	1	2
1	3	3	2
6	5	2	1

Average pool
Hyperparameters:
$f = 2$
$s = 2$

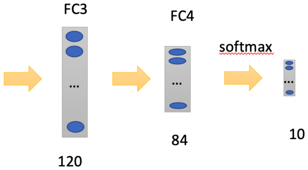
3.75	1.25
4	2

- No parameters to learn, output $\lfloor \frac{n+2p-f}{s} + 1 \rfloor \times \lfloor \frac{n+2p-f}{s} + 1 \rfloor$

An example

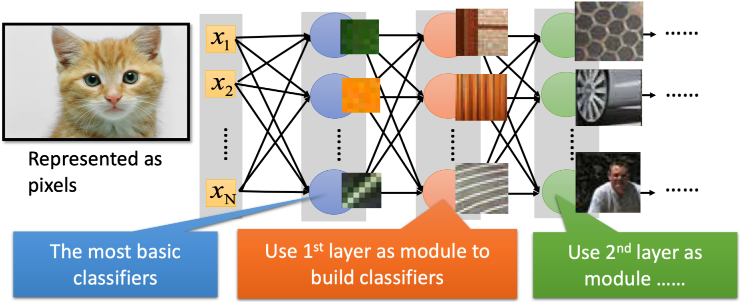


400



	Activation shape	Activation size	# parameters
Input	(32,32,3)	3,072	0
CONV1(f=5,s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2(f=5,s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

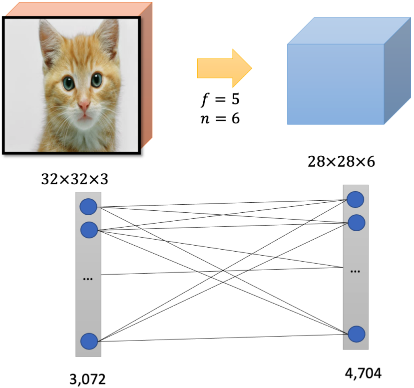
Why convolutions



Can the network be simplified by considering the properties of images?

[Zeiler, M. D., *ECCV 2014*]

Why convolutions



Only 156 parameters

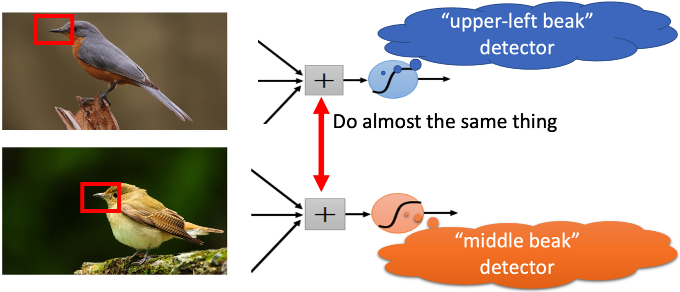
3,072 * 4,704 parameters

Convolutions enable parameter sharing

- A feature detector (such as the vertical edge detector) that's useful in one part of the image is probably useful in another part of the image

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

- The same patterns appear in different regions

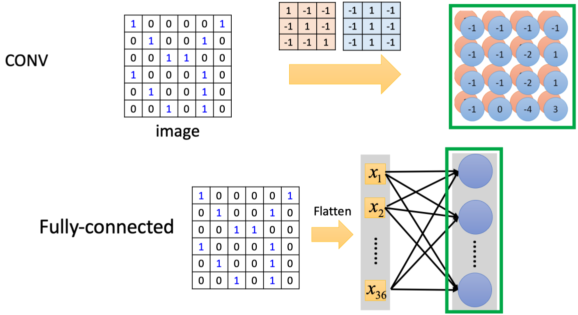


Convolutions enable sparsity of connections

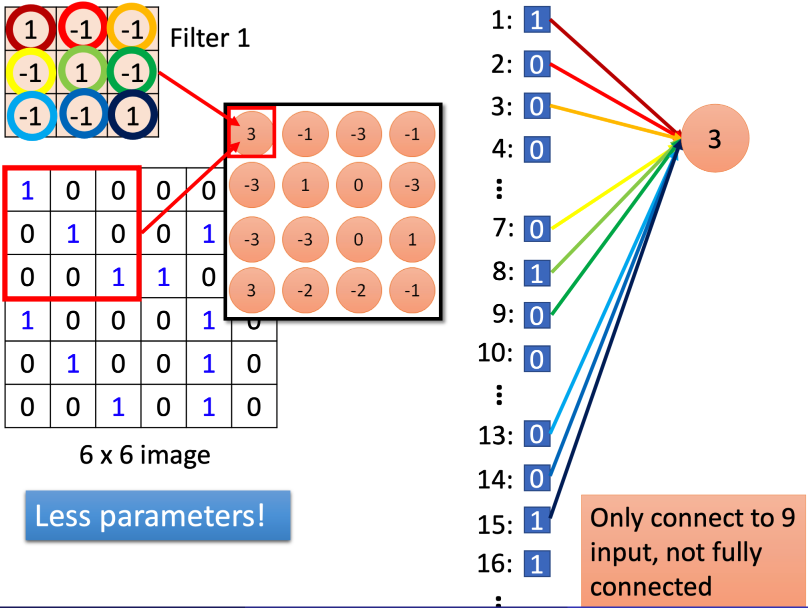
- in each layer, each output value depends only on a small number of inputs

$$
\begin{bmatrix}
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0
\end{bmatrix}
$$

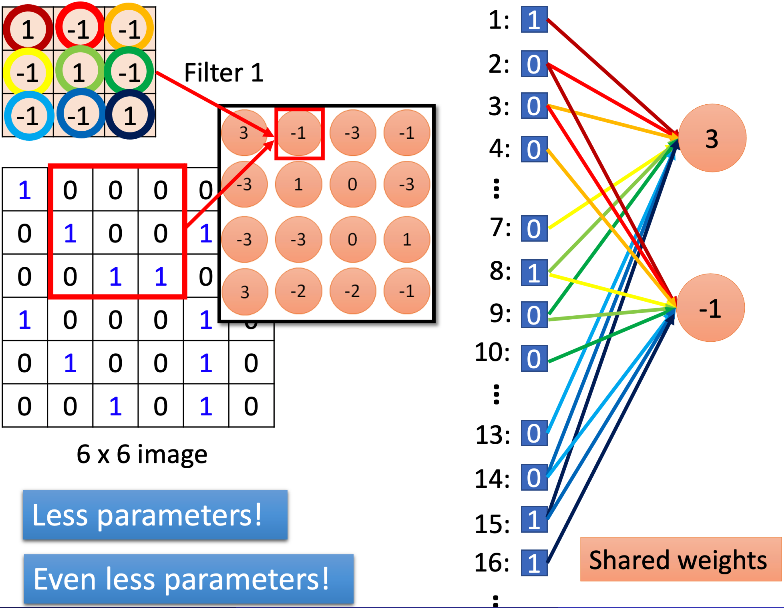
- Sparsity of connections: ConV v.s. FC



Benefits of using ConV



Benefits of using ConV



Less parameters!

Even less parameters!

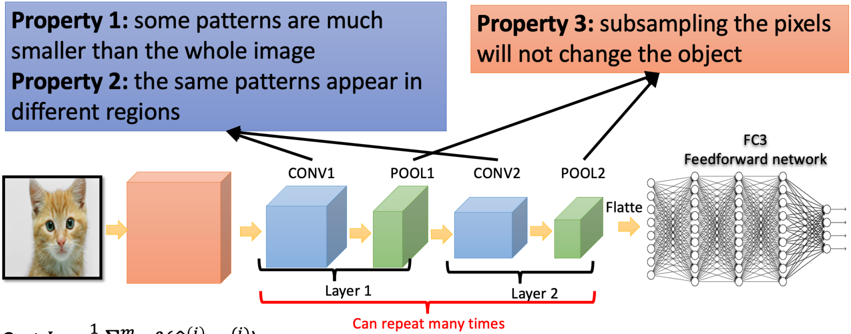
Why pooling?

- Subsampling the pixels will not change the object

We can subsample the pixels to make image smaller

Less parameters for the network to process the image

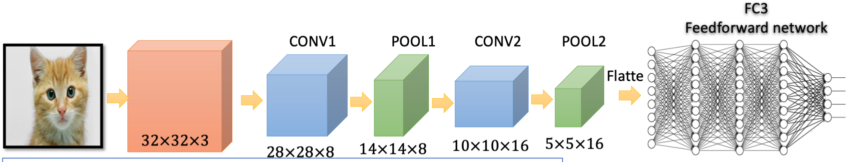
The whole CNN architecture



Cost $J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Use gradient descent to optimize parameters to reduce J

The whole CNN architecture use Keras



model.add(Convolution2D(8, 5, 5), input_shape=(3, 32, 32))

Number of filter

Filter size

Number of channel: 1 for black; 3 for RGB

32 by 32 pixels

model.add(MaxPooling2D((2,2))

model.add(Convolution2D(16, 5, 5))

model.add(MaxPooling2D((2,2))

model.add(Dense(output_dim = 100))

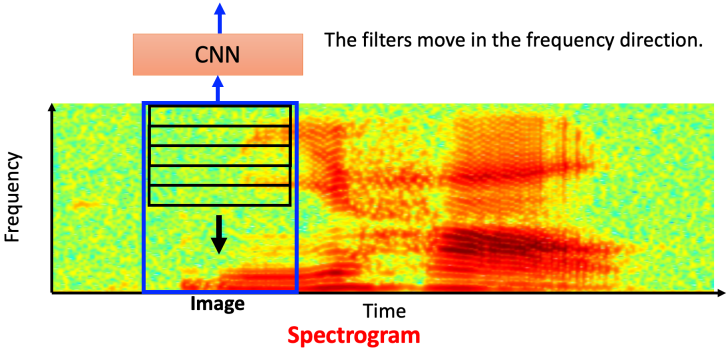
model.add(Activation('relu'))

model.add(Dense(output_dim=10)

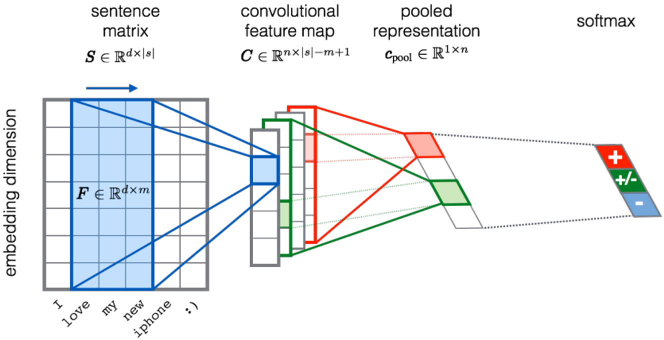
model.add(Activation('softmax'))

model.add(Flatten())

CNN for speech



CNN for text

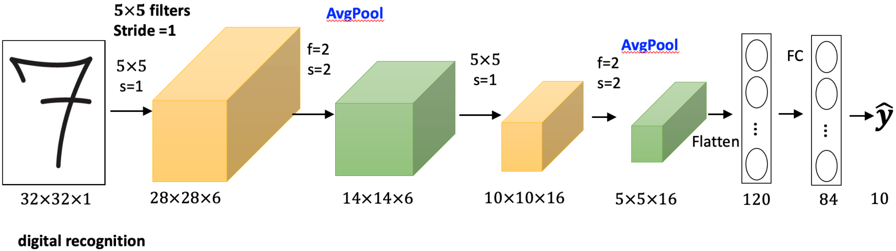


Source of image:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

Classic Networks

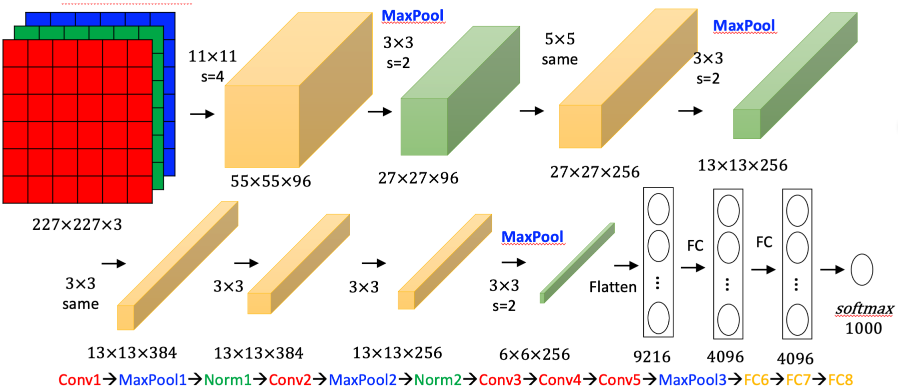
LeNet-5, AlexNet, VGG, GoogleNet, ResNet

LeNet-5



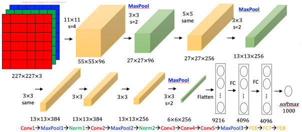
Conv1→AvgPool1→Conv2→AvgPool2→FC3→FC4→Softmax

AlexNet



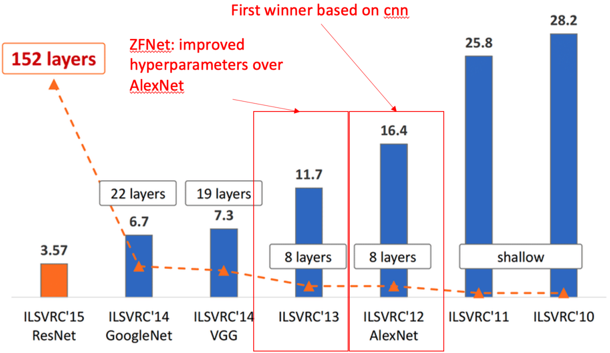
- First large scale CNN to do well in image classification!

AlexNet



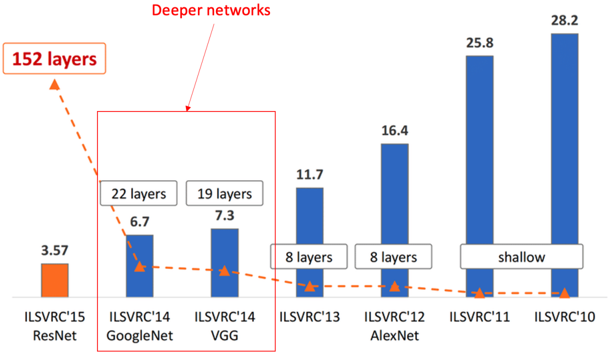
- Input: 227×227×3 images; First layer (Conv1): 96:11×11 filters with stride = 4
  - Q: what is the output volume size? 55×55×96
  - Q: how many parameters? (11×11×3)×96
- Second layer (MaxPool1): 3×3 filters with stride = 2
  - Q: what is the output volume size? 27×27×96
  - Q: how many parameters? 0
- Details:
  - 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 0.01, reduced by 10 manually when val accuracy plateaus; Le weight decay 0.0005; 7 cnn ensemble, accuracy improved by around 3

ImageNet large scale visual recognition challenge winners



- ZFNet has the same structure with AlexNet, but
  - Conv1: 11×11 filers with stride 4 $\rightarrow$ 7×7 filers with stride 2
  - Conv3, 4, 5: number of filers 384, 384, 256 $\rightarrow$ 512, 1024, 512
- accuracy improved by 4.7%

ImageNet large scale visual recognition challenge winners



VGG-16

Small filters, but deeper networks

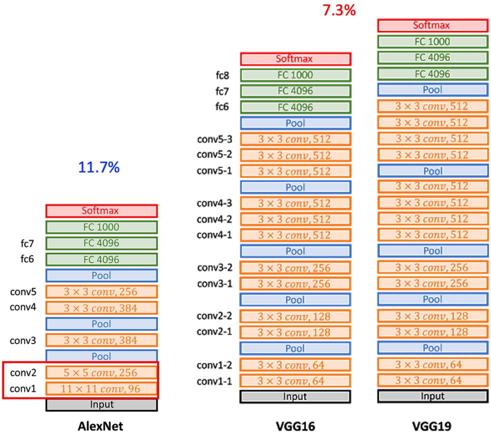
Q: why use smaller filters?

Stack of three 3x3 Conv (stride 1) layers has same effective receptive field as one 7x7 Conv layer

Q: what is the effective receptive field of three 3x3 Conv (stride 1) layers? 7x7

Pros:

1. more non-linearities
2. fewer parameter: $3\times(3^2\times C)$ vs $7^2\times C$ where C is the number of channels



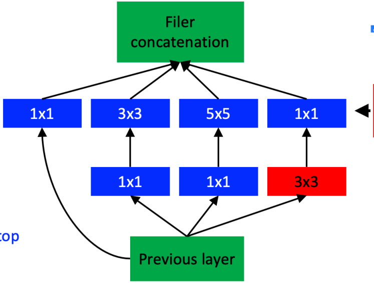
GoogleNet

GoogleNet

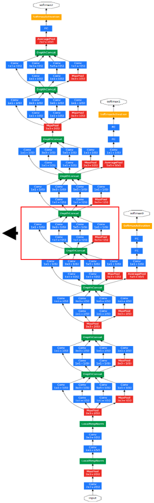
Deeper networks with computational efficiency

- 22 layers
- Efficient inception module
- No fully connect layers
- Only 5m parameters (much less than alexnet)
- ILSVRC'14 classification winner (6.7%)

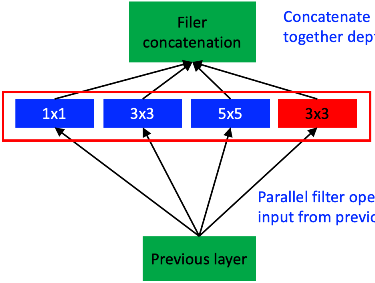
Inception module: a good local network topology and then stack these modules on top of each other



Inception module



Novelty of GoogleNet: Inception Module



Concatenate all filter outputs together depth-wise

Parallel filter operations on the input from previous layer

Naïve Inception module

[Szegedy et al., 2014]

Q: what is the problem with this naïve inception module?
Computational complexity

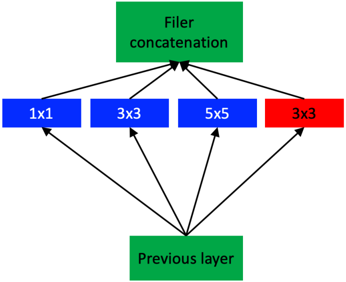
Input: 28x28x256
126 1x1 Conv same padding
192 3x3 Conv same padding
96 5x5 Conv same padding
3x3 Pool

What's the output size?
28x28x672

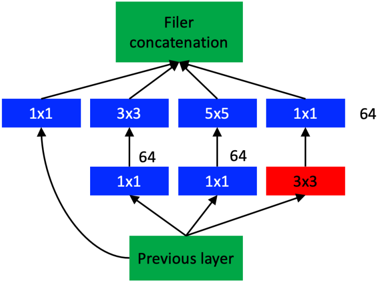
Pooling layer also an issue. Why?

Google solution: use bottleneck layers that use 1x1 Conv to reduce feature depth

Novelty of GoogleNet: Inception Module



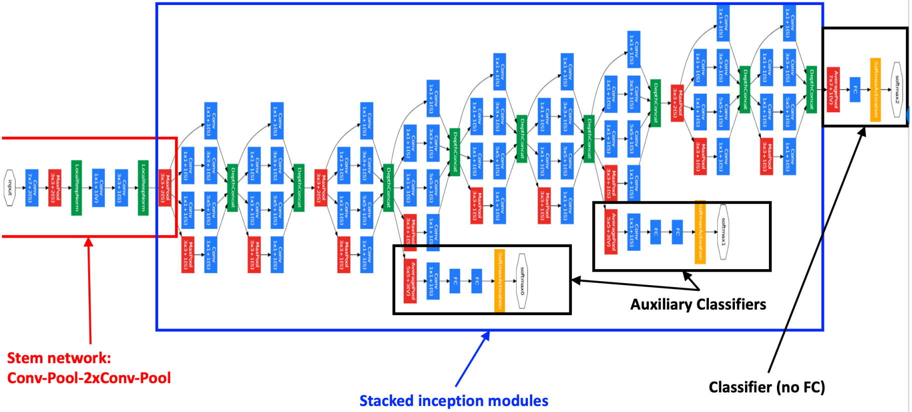
Naïve Inception module



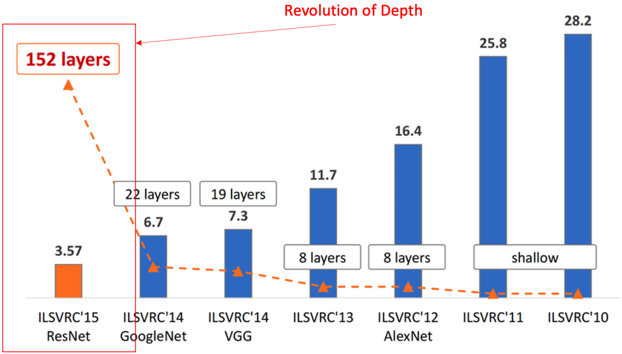
Inception module: dimension reduction

[Szegedy et al., 2014]

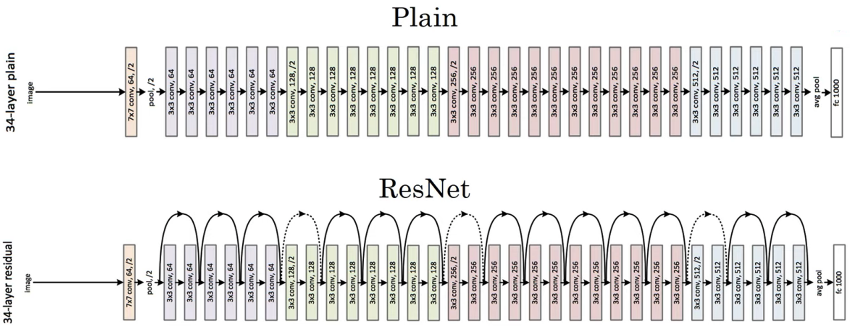
GoogleNet


Auxiliary Classifiers
Stem network:
Conv-Pool-2xConv-Pool
Stacked inception modules
Classifier (no FC)

ImageNet large scale visual recognition challenge winners

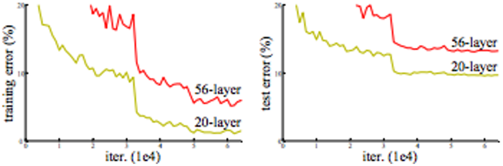


ResNets



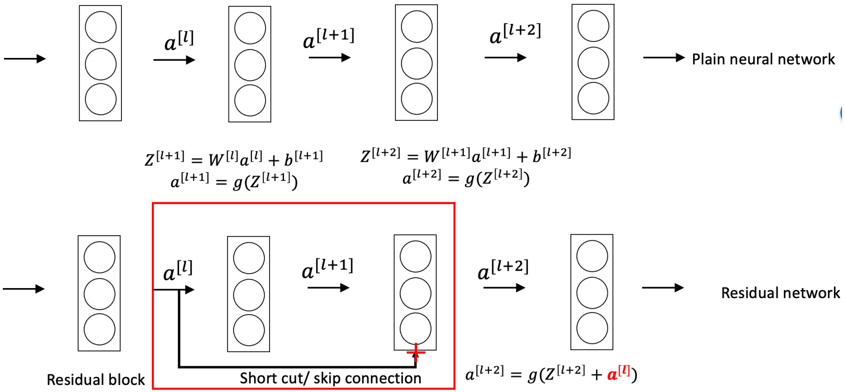
Plain
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
ResNet
34-layer residual

- Very deep neural network is difficult to train because of vanishing and exploding gradients
- ResNet is able to train very deep neural network



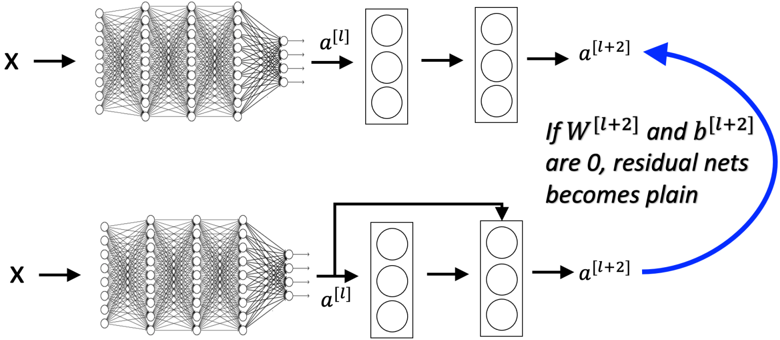
- 56-layer model performs worse on both training and test error
- The deeper model performs worse, but it's not caused by overfitting, more because of optimization issue

Residual block

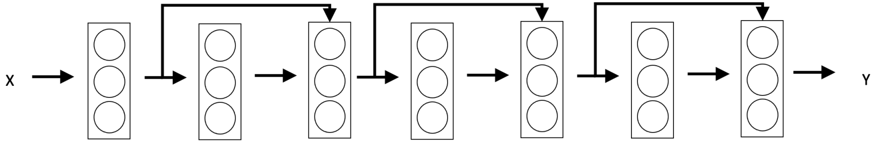


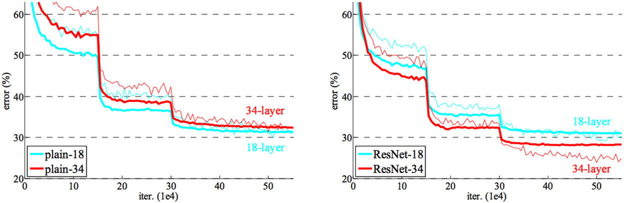
[He et al., 2015, deep residual networks for image recognition]

Why ResNets work?



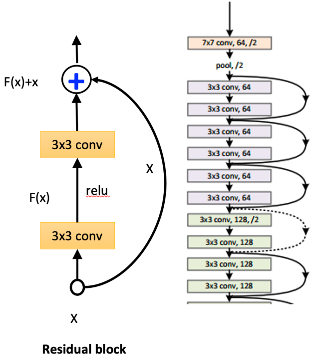
Why ResNets work?





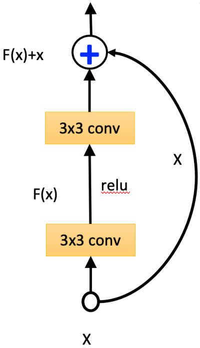
[He et al., 2015, deep residual networks for image recognition]

ResNets



- stack residual blocks; every residual block has 2: 3x3 ConV layers
- periodically, docule number of filters and downsample spatially using stride 2
- additional ConV layer at the beginning and no FC layers at the end

ResNets

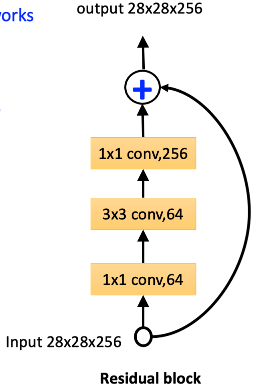


Similar to GoogLeNet, for deeper networks (more than 50 layers), ResNet use bottleneck layer to improve efficiency

1x1 conv, 256 filters to project back to to 28x28x256

3x3 conv (onlyt 64 feature maps)

1x1 conv, 64 filters to project to 28x28x64



The End