Neural Networks and Deep Learning Lecture 6

Wei WANG

cs5242@comp.nus.edu.sg

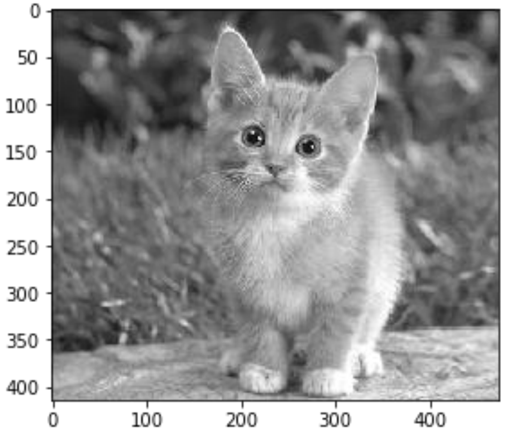
NUS National University of Singapore

School of Computing

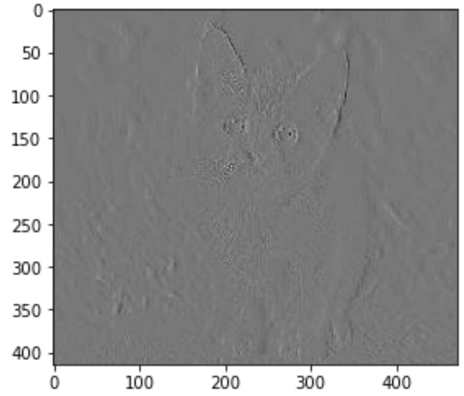
Recap

Convolution

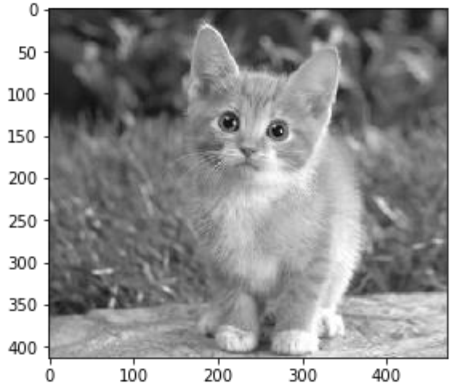
- Convolution operations extract features of the input image (or feature maps) via different filters/kernels.
- The training algorithm tunes the kernel values to learn features that are effective for the tasks of interest.



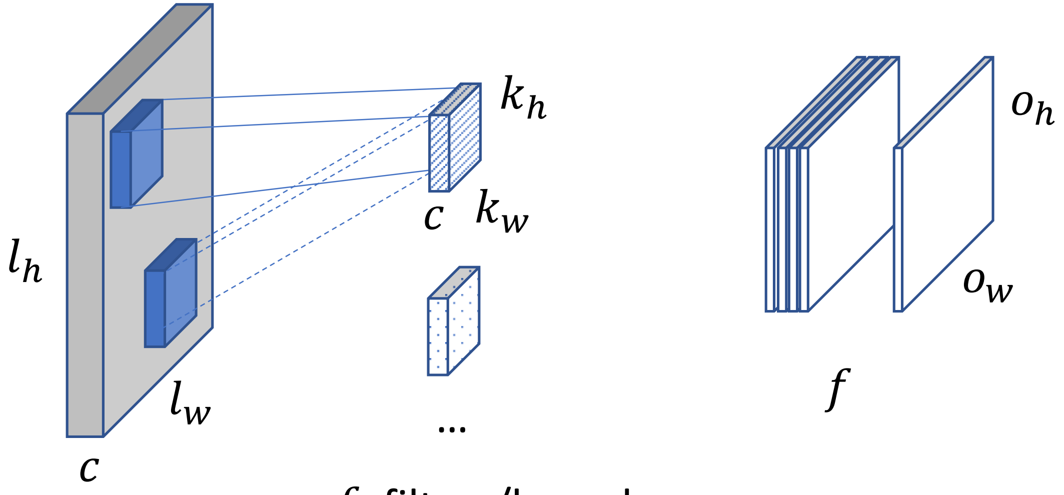
$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \rightarrow$$



$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \rightarrow$$



2D Convolution



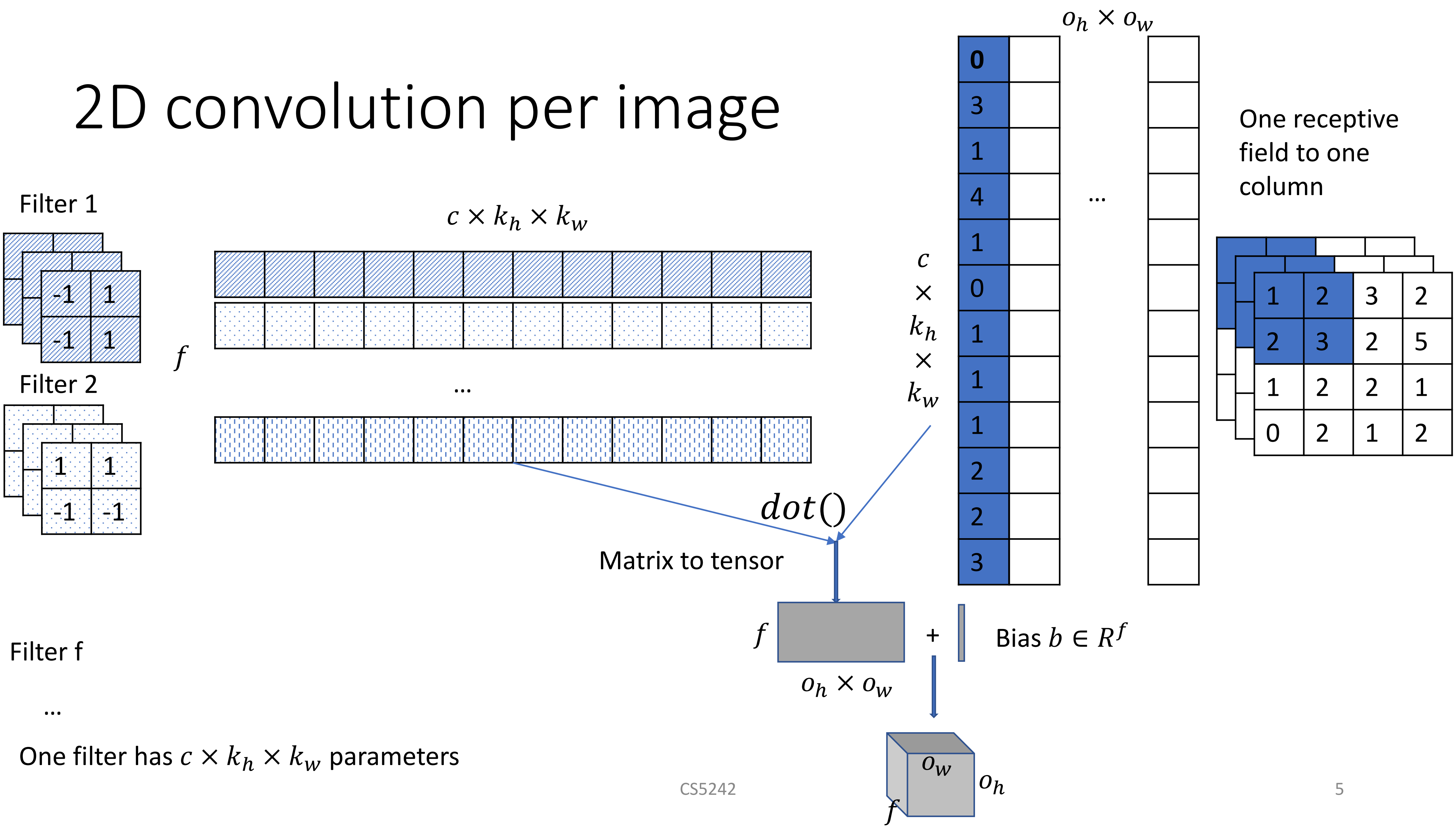
f filters/kernels

http://cs231n.github.io/convolutional-networks/

https://www.youtube.com/watch?v=jajksuQW4mc

$y_{l,i,j} = \sum_{d=0}^{c-1} \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{d,i+a,j+b} \times W_{l,d,a,b} + b_l, l \in [0, f)$

2D convolution per image



One filter has $c \times k_h \times k_w$ parameters

BP of convolution

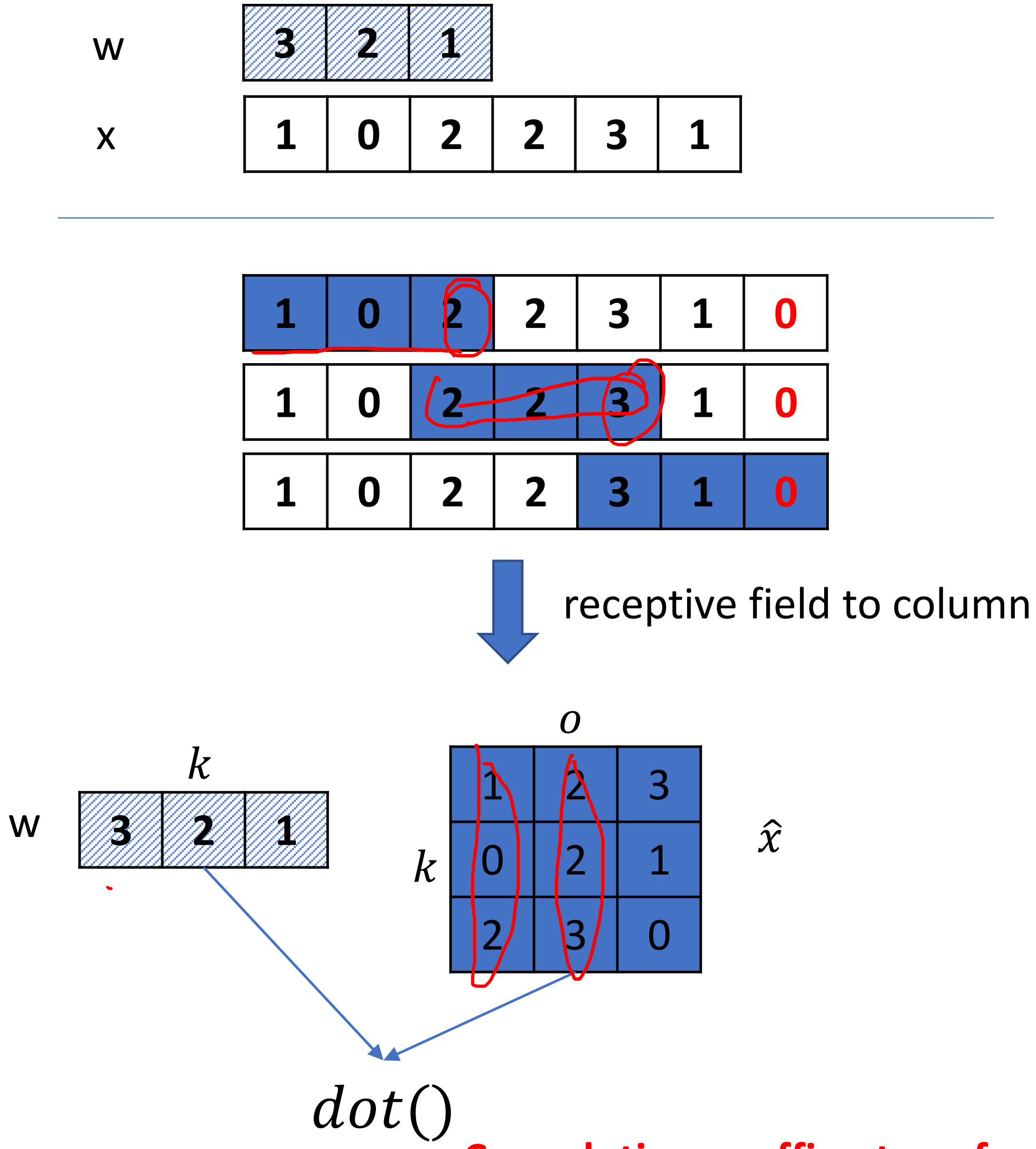
- Create filter matrix
  - W=np.random.rand(f, c$\times$ $k_h \times k_w$) $\times$ 0.01
- Forward(W,$\{X^{(i)}\}_{i=1}^{B}$) (bias is skipped here for the sake of simplicity)
  - For i in range(B):
    - Convert input feature maps $X^{(i)}$ into matrix $\hat{X}^{(i)}$ (img2col)
    - $Y^{(i)} = W\hat{X}^{(i)}$
  - Return the $\{Y^{(i)}\}_{i=1}^{B}$
- Backward($\{\frac{\partial J}{\partial Y^{(i)}}\}_{i=1}^{B}$, $\{X^{(i)}\}_{i=1}^{B}$, W)
  - For i in range(B):
    - $\frac{\partial J}{\partial W}$+= $\frac{\partial L}{\partial Y^{(i)}}\hat{X}^{(i)^T}$, $\frac{\partial J}{\partial \hat{X}^{(i)}}$=$W^T\frac{\partial J}{\partial Y^{(i)}}$
    - Column to receptive field $\frac{\partial J}{\partial \hat{X}^{(i)}} \rightarrow \frac{\partial J}{\partial X^{(i)}}$ (by aggregating the gradients for each neuron)
  - Return $\frac{\partial J}{\partial W}$, and $\{\frac{\partial J}{\partial X^{(i)}}\}_{i=1}^{B}$
  - (note: J is the averaged loss, hence we return $\frac{\partial J}{\partial W}$ instead of $\frac{\partial J}{\partial W}$ / B)
- Apply SGD after getting $\frac{\partial J}{\partial W}$

Implementation (fwd)

- Receptive field to column

```
for t in range(o):
    for i in range(k):
        x_hat[i, t]= x[t*s + i]
y = dot(w, x_hat)
```

s=2 for the example on the right



Convolution -> affine transformation

Implementation (bwd)

```
dw = dot(dy, x_hat.T())
dx_hat = dot(w.T(), dy)
set dx to 0s
for t in range(o):
    for i in range(k):
        dx[t*s+i] += dx_hat[i, t]
```

s=2 for the example on the right

w.T()

3
2
1

$\frac{\partial L}{\partial y}$

1	**-1**	**1**

$\frac{\partial L}{\partial \hat{x}}$

3	-3	3
2	-2	2
1	-1	1

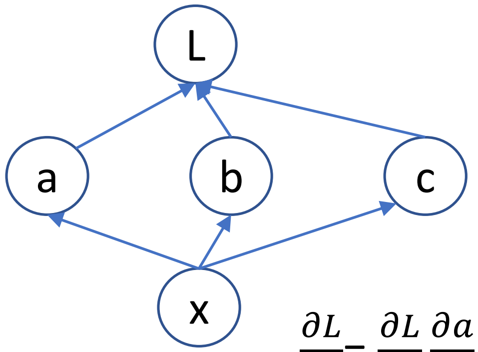
Column to receptive field

3	**2**	**1**	**0**	**0**	**0**	**0**
0	**0**	**-3**	**-2**	**-1**	**0**	**0**
0	**0**	**0**	**0**	**3**	**2**	**1**

+

$\frac{\partial L}{\partial x}$

3	2	**-2**	-2	**1**	2	1



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial x} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial x}$$

Why addition?
x contributes to L via three paths →
its gradient is the sum of the gradients from the three paths

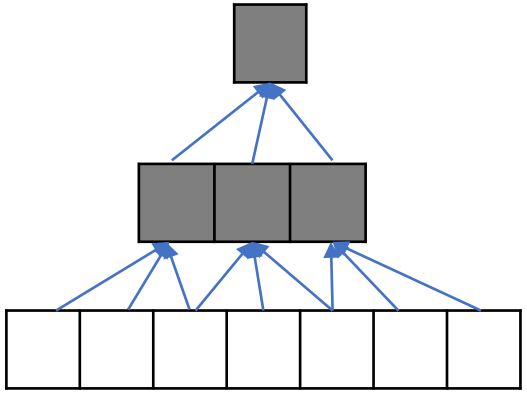
Hyper-parameters of the l-th conv layer

- Filter
  - Number of filters/kernels/channels, $c^{[l]}$; 32, 64, 128, 256, etc.
  - Width $k_w^{[l]}$; and height $k_h^{[l]}$; Odd values, 1x1, 3x3, 5x5, 7x7
    - Why odd values? To get the same number of padding on both sides
- Padding
  - Manual configuration
    - $p^{[l]}$ for each side (top, left, bottom, right)
    - total padding on horizontal/vertical dimension is $2p^{[l]}$
  - Same or valid
    - Compute the number of padding on the left and right respectively
    - Compute the number of padding on the top and bottom respectively

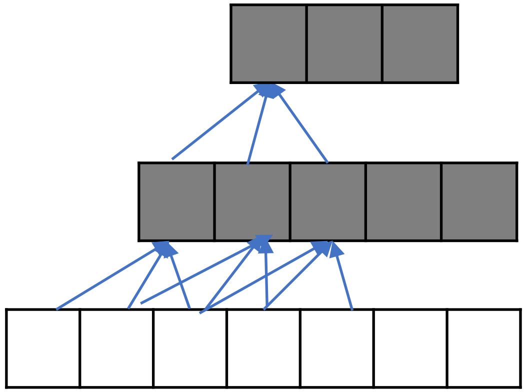
Hyper-parameters of the l-th conv layer

- Stride
  - $s^{[l]}$;
  - 1 for conv; >1 for pooling

The receptive field area of each neuron in the top layer w.r.t the bottom layer is 7



S=2

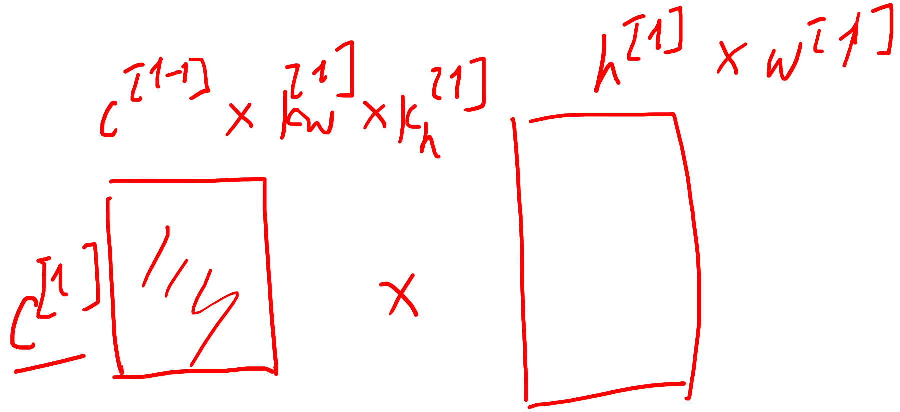


S=1

The receptive field area of each neuron in the top layer w.r.t the bottom layer is 5

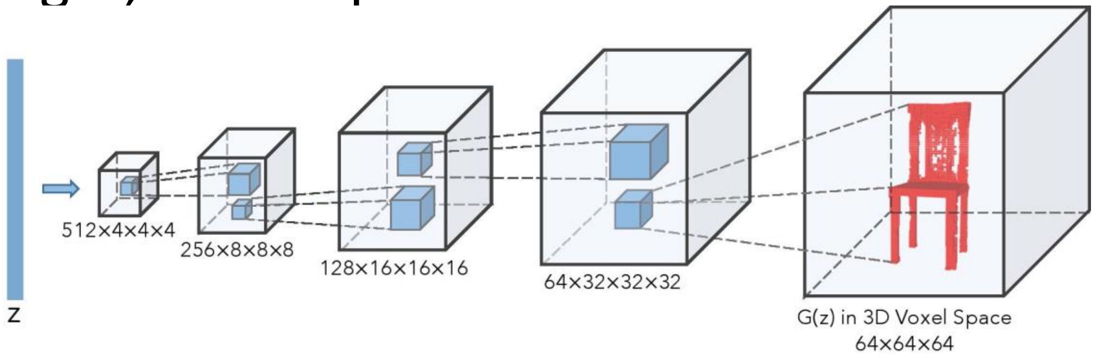
Statistics for the l-th conv layer

- Output shape
  - $(c^{[l]}, h^{[l]}, w^{[l]})$ or $(h^{[l]}, w^{[l]}, c^{[l]})$
- Parameter size
  - Weight matrix, $c^{[l]} \times (c^{[l-1]} \times k_w^{[l]} \times k_h^{[l]}\ \ )$
  - Bias vector, $c^{[l]}$
- Computation cost
  - $O(c^{[l]} \times\ \ c^{[l-1]} \times k_w^{[l]} \times k_h^{[l]}\ \times\ h^{[l]} \times w^{[l]})$ (float multiplication ops)



3D convolution

- 2D convolution
  - Convolution/slide over spatial dimensions, i.e. height and width
  - Slide the window from left to right, from top to bottom to extract receptive fields
- 3D convolution
  - Convolution/slide over h, w, and one more dimension, i.e. depth, d.
  - Slide from left to right, from top to bottom and from front to back to extract receptive fields.



http://3dgan.csail.mit.edu/

ConvNets for Image Classification

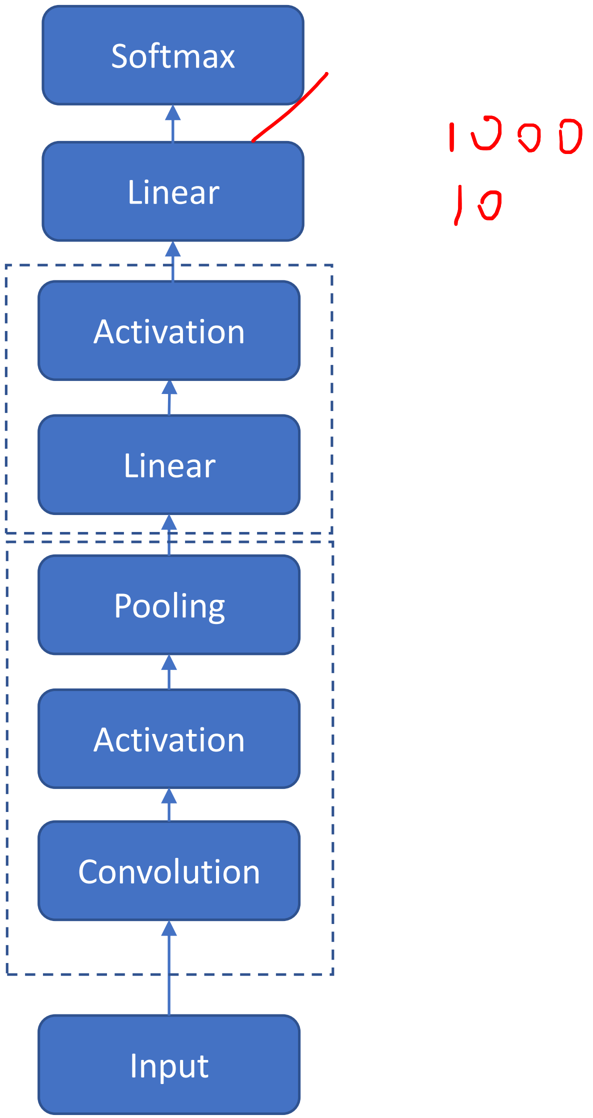
Multi-class image classification

- Given an image, predict its class from a set of candidate classes.
- Dataset
  - Small benchmark dataset
    - CIFAR10
      - 10 classes
      - 60000 RGB images, 50000 training and 10000 images
      - 32x32 RGB image
    - MNIST
      - 10 classes
      - 70000 grayscale images, 60000 training and 10000 test
      - 28x28
  - Big benchmark dataset
    - ImageNet
      - 1000 classes
      - 1.2 M RGB images
      - Variant size

https://pythonic-ocr.herokuapp.com/

Model

- ConvBlock
  - Convolution
  - Activation
  - Pooling
- Linear + Activation



Output layer

- The layer before the loss, e.g. the last linear layer
- From binary classification to multi-class classification
  - Logistic/sigmoid function for binary classification
    - Probability for positive, p
    - Probability for negative, 1-p
  - Softmax for multi-class classification
    - $\tilde{y}_i = \frac{e^{hi}}{\sum_j e^{h_j}}$, h_j is called a logit, $\tilde{y}_i$ ∈(0, 1), sums to 1.
    - Interpreted as the probability of the image from the i-th category

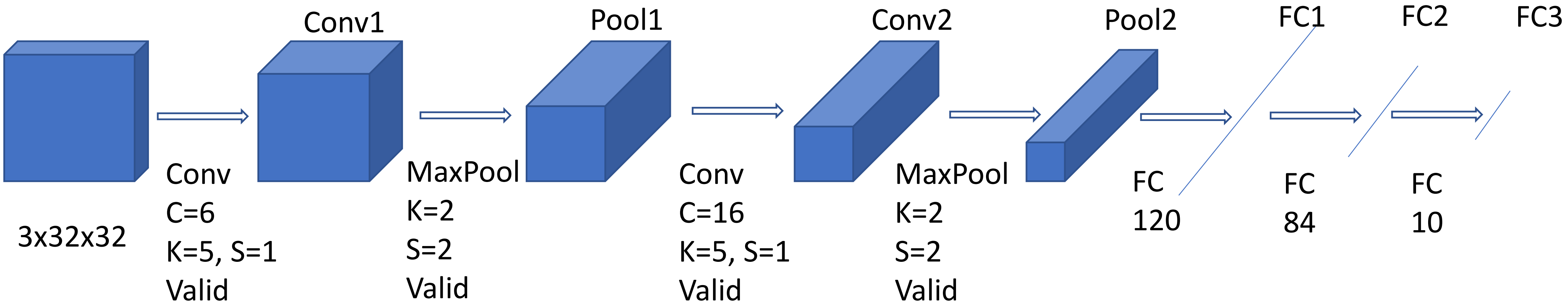
Loss

- Binary cross-entropy (https://www.youtube.com/watch?v=-xautwSTZIY,)
  - $L(x,y) = -\text{y}\log\tilde{y} - (1-y)\log(1-\tilde{y})$
- Cross-entropy
  - https://www.youtube.com/watch?v=ErfnhcEV1O8, https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/, https://www.youtube.com/watch?v=mlaLLQofmR8
  - $L(x,y) = \mathrm{H}(\text{y},\tilde{y}) = -\sum_{i=1}^{C} y_i \log(\tilde{y}_i)$
  - $\qquad\qquad = H(y) + D_{KL}(y||\tilde{y})$
  - C is the total number of classes
  - y_i = 1, if the truth label is the i-th class; 0 otherwise;
- Gradient
  - $\frac{\partial L}{\tilde{y}_i} = \tilde{y}_i - y_i$ (y_i is the ground truth, 0/1)

Give the derivation in Assignment 1

ConvNet architectures

LeNet-5 (1990s)



Layer	Shape	Parameters
Conv1	(6, 28, 28)	6x3x5x5
Pool1	(6,14,14)	0
Conv2	(16, 10, 10)	16x6x5x5
Pool2	(16, 5, 5)	0
FC1	(120,)	120x(16x5x5)
FC2	(84,)	84x120
FC3	(10,)	10x84

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

Design guide of recent ConvNets (2012-)

- Faster
  - Converge faster (Numeric optimization)
    - Learning rate
    - **Gradient**
  - Compute faster
    - Computation **complexity**
- More accurate
  - Large representation capacity -> overfitting
  - **Generalize** better

AlexNet 2012

- http://ethereon.github.io/netscope/#/preset/alexnet

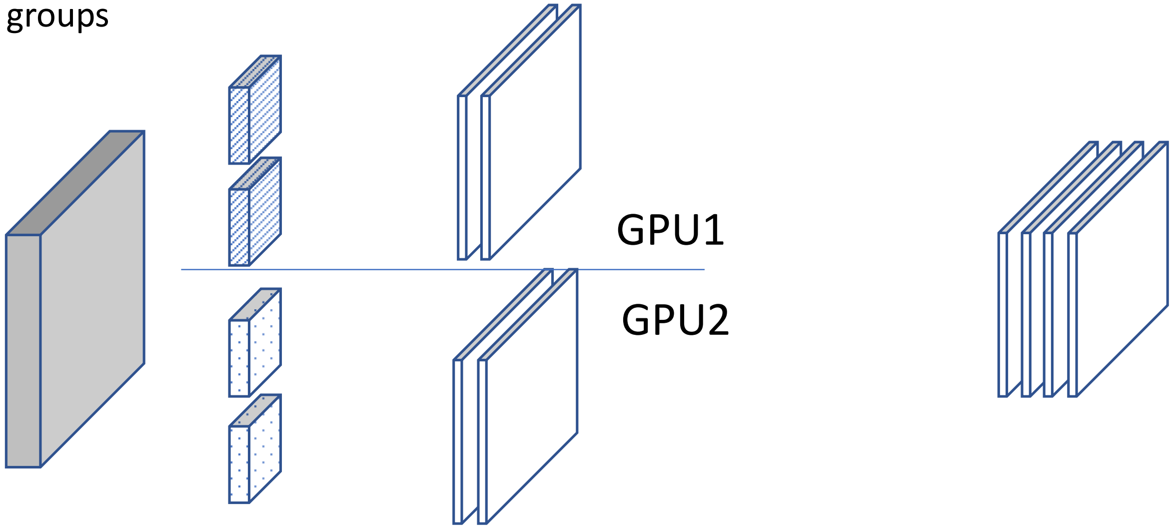
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

Alex, et al. Imagenet classification with deep convolutional neural networks.

From cs231n

AlexNet 2012

- http://ethereon.github.io/netscope/#/preset/alexnet
- AlexNet
  - Original
    - two GPUs, filters of some conv layers are separate into two groups
    - # filters = 96-> 48, 48
  - Caffe implementation
    - One GPU, a single filter group
    - # filters = 96
  - Similar performance
    - 18% error, 7 CNN ensemble: 18.2% -> 15.4%
- Characteristics
  - 5 convolutional layers (Deep architecture)
    - bigger model size (parameters) -> large capacity
    - large number of filters (96, 256, 384, 384, 256)
    - Mixed kernel size, 11x11, 5x5, 3x3; stride size, 4x4, 1x1;
  - Dropout → model ensemble → generalize well → improve accuracy
  - ReLU → avoid gradient vanishing



Alex, et al. Imagenet classification with deep convolutional neural networks.

VGG 2014

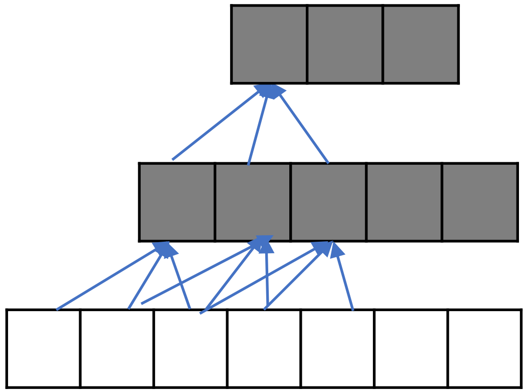
- http://ethereon.github.io/netscope/#/preset/vgg-16
- How to set the kernel size?
- ReLU is after every convolution layer and FC layer

From cs231n

Very Deep Convolutional Networks for Large-Scale Image Recognition. Karen Simonyan, Andrew Zisserman.

VGG 2014

- Characteristics
  - Deeper structure (16, 19 convolution layers)
    - More model parameters compared with AlexNet
      - 130+ Million VS 60 Million
    - More non-linear transformation; Large capacity
  - Unified kernel size
    - 3x3 ?
    - Stacking 2 (or 3) 3x3 conv layers == 5x5 (7x7) in terms of receptive field size
    - Computationally cheaper
      - $O(c^{[l]} \times \ c^{[l-1]} \times k_w^{[l]} \times k_h^{[l]} \times \ h^{[l]} \times w^{[l]})$, k=3,5,7
      - Run faster
  - 2nd of 2014 classification task,
  - Widely used for other applications, via transfer learning



The receptive field are of each neuron in the top layer is 5x5 wr.t the bottom layer

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Conv layers fewer parameters, more computation
FC layer most parameters, less computation

From cs231n

InceptionNet

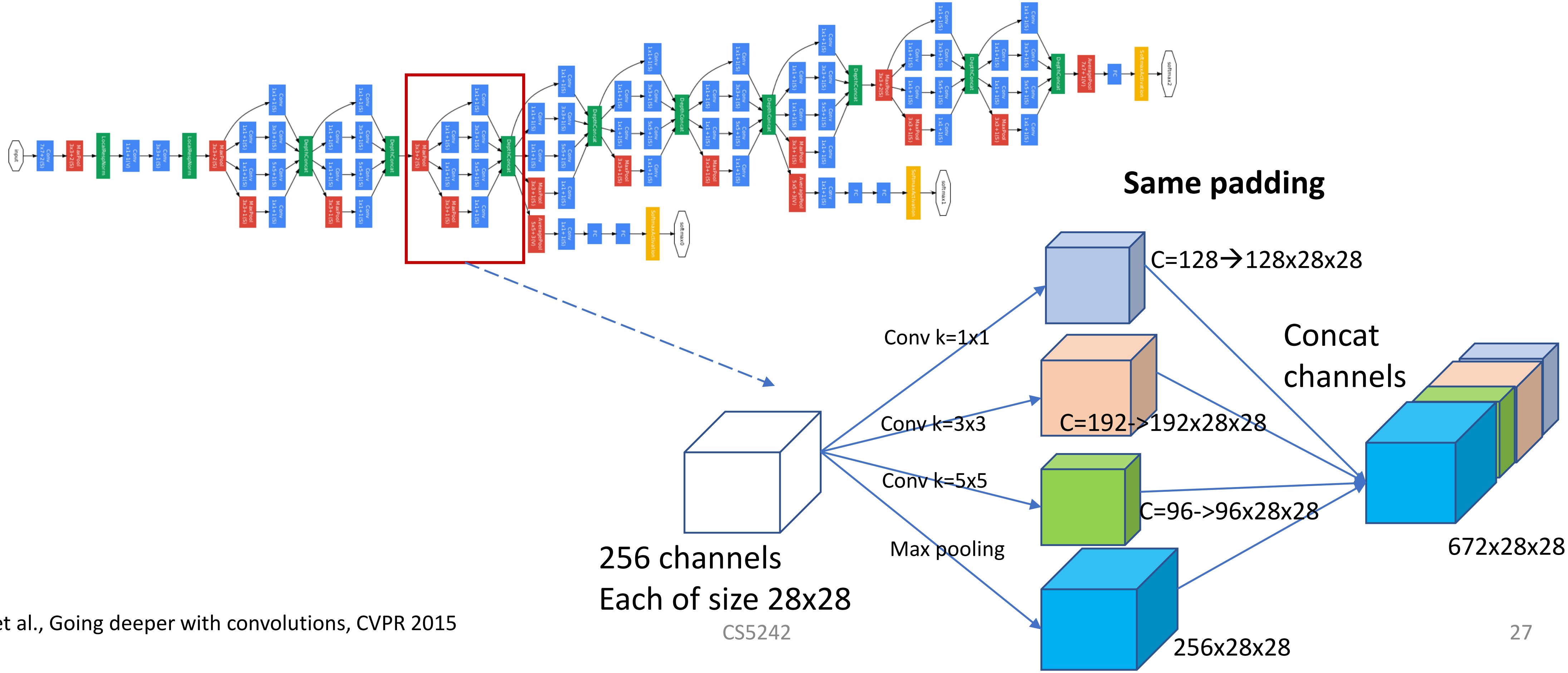
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

Source: jamiis.me/submodules/presentation-convolutional-neural-nets/img/deeper-meme.jpg

https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

InceptionNet

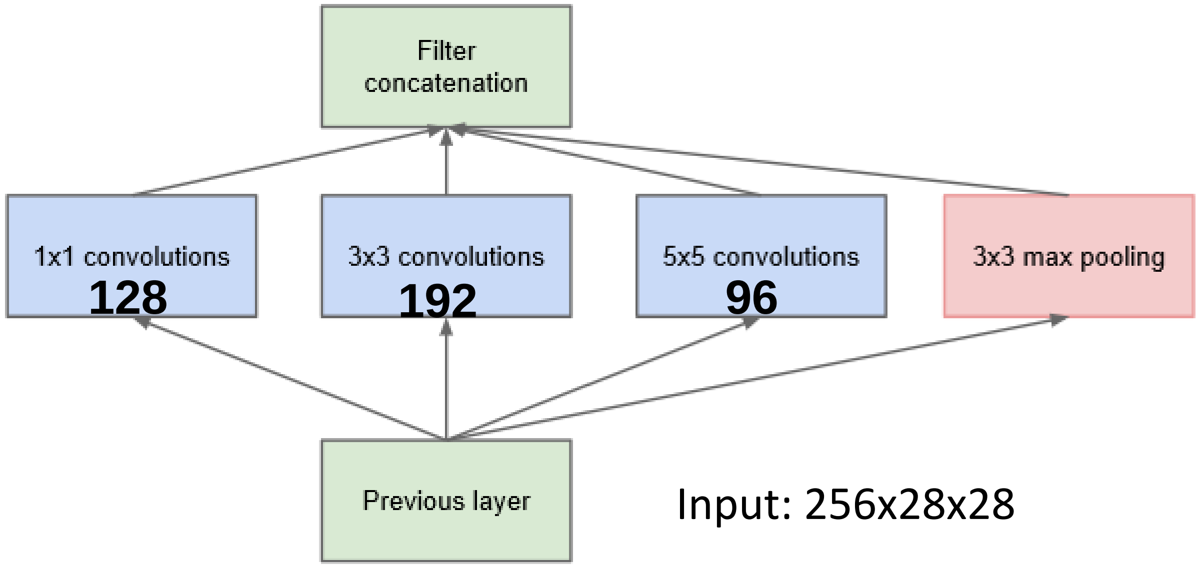
- http://ethereon.github.io/netscope/#/preset/googlenet



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

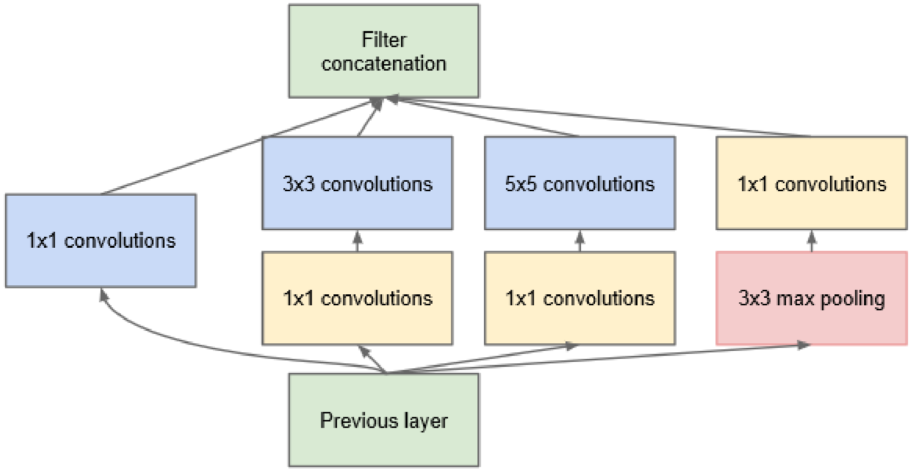
InceptionNet

- Each inception block
  - 1x1, 3x3, 5x5 kernels
    - avoid co-adaption as all branches have different kernel sizes
  - Wide block



(a) Inception module, naïve version

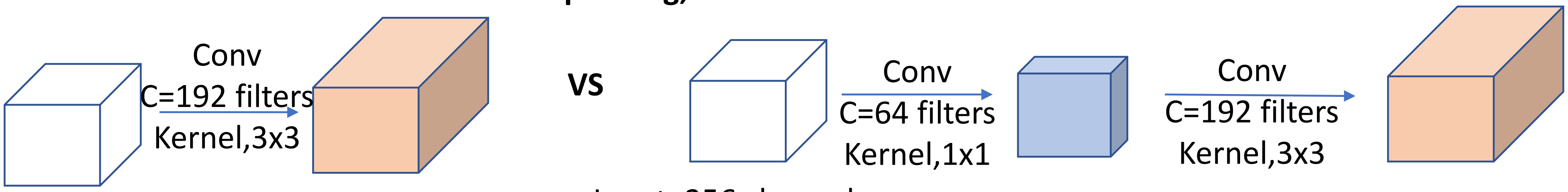
InceptionNet



(b) Inception module with dimension reductions

- 1x1 convolution
  - Fusion of feature maps
    - dimension reduction;
    - remove redundant features;
    - cross channel information learning [16]
  - Cost saving?
    - Write down the cost of the left and right networks

Same padding, stride 1



Input: 256 channels, each of 28x28

Input: 256 channels, each of 28x28

InceptionNet

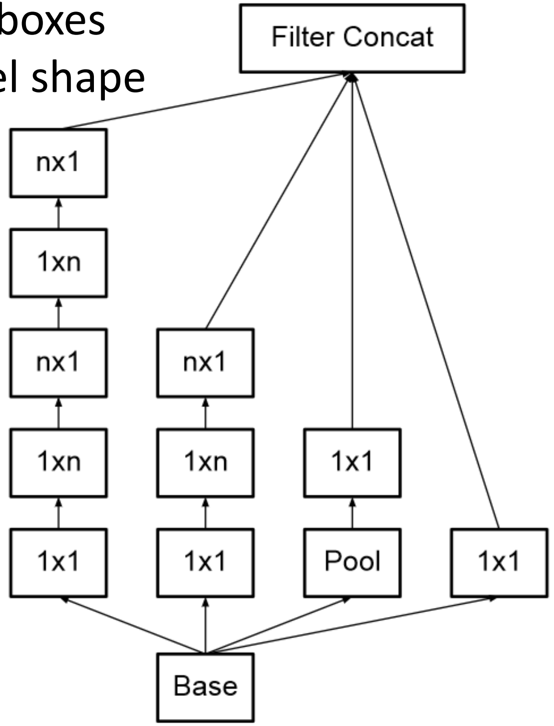
Numbers in the boxes are for the kernel shape

- Factorization of the kernel
  - 7x7 == 1x7 and 7x1 (receptive field)?
  - Reduce computation cost → train faster

nxn

nxn

1x1



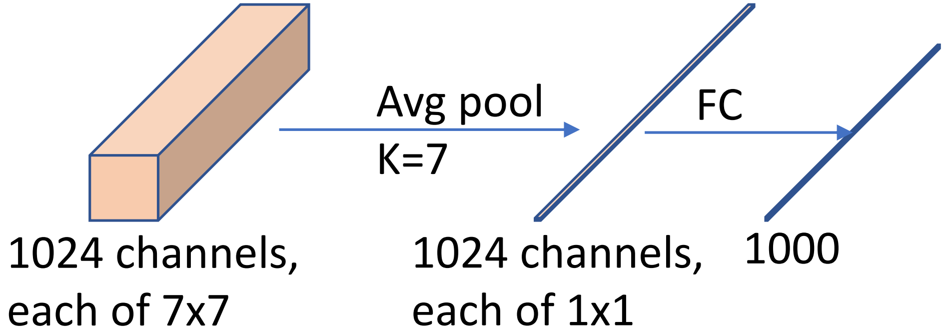
$k_h \times k_w$

① $n \times n$

② $n \times 1 + 1 \times n = 2n$

InceptionNet

- Characteristics
  - Inception block
    - More complex structure compared with AlexNet and VGG (single path)
    - Ensemble multiple paths
    - Avoid co-adaption
  - Average pooling
    - Receptive filed size = feature map size
      - Output feature map size = 1x1
    - Remove fully connected layers
      - Reduce model size → less overfitting
      - Reduce time complexity
  - 5 million parameters, 1st 2014 classification task, 6.7% error (top-5)
  - 22 layers



ResNet

- Can we go much deeper?
  - Overfitting?
    - No
  - Optimization?
    - Difficult for deep net

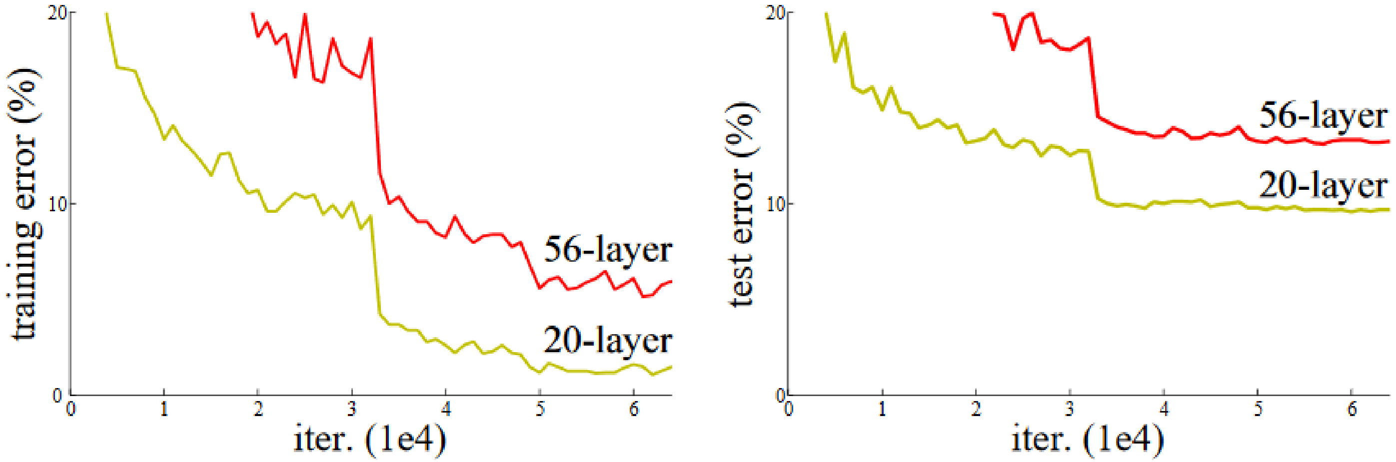
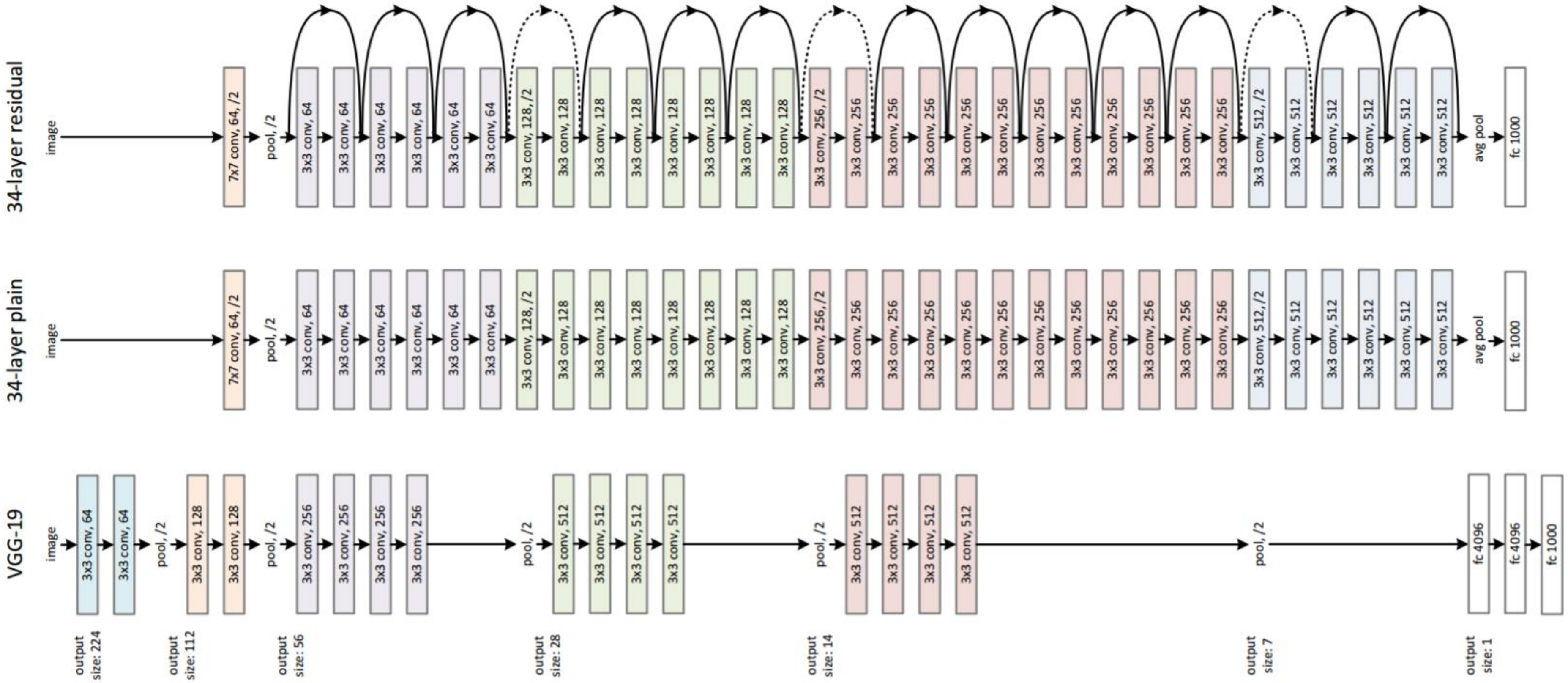


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

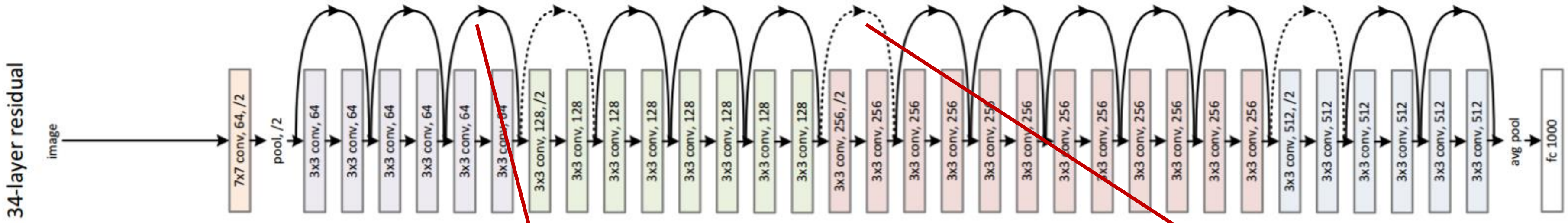
Kaiming He, Xiangyu Zhang, Shaoqing Ren , and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016

ResNet

- http://ethereon.github.io/netscope/#/gist/d38f3e6091952b45198b

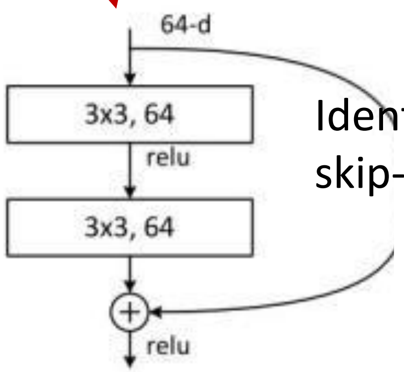


ResNet



Conv, 64 filters, same padding, stride 1

Conv, 64 filters, same padding, stride 1



Identity / skip-connection

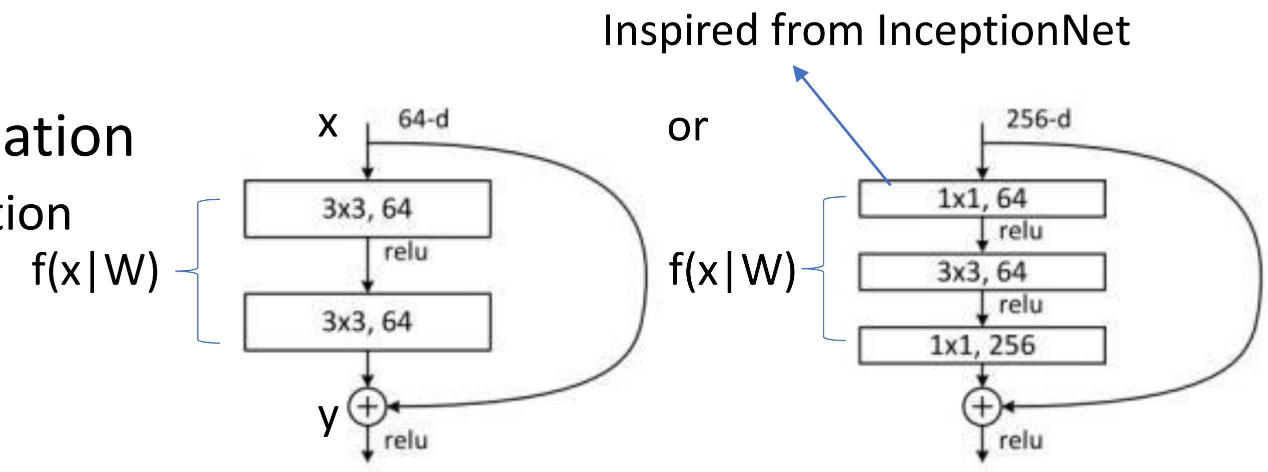
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

One convolution layer has stride=2 → feature map area halved → replace the identity connection with a linear transformation to make the feature maps of the same shape as those from the convolution layers

Why ResNet works?

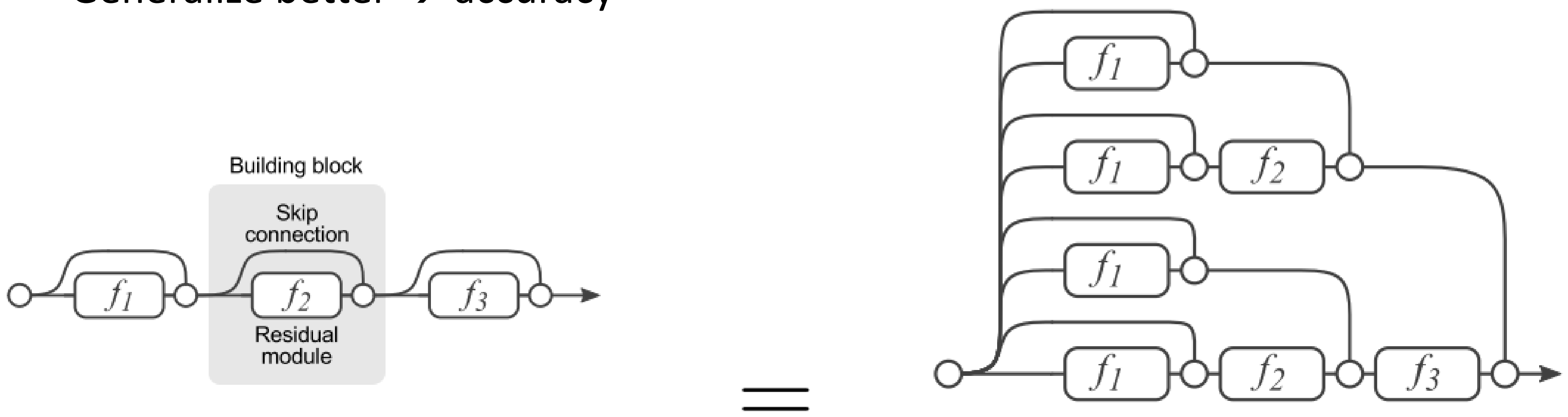
- http://ethereon.github.io/netscope/#/gist/d38f3e6091952b45198b
- 152-layers, 3.57% top 5 error, winner for classification, detection, etc.
- Skip/residual connection
  - $y = x + f(x|W)$
  - Gradient flows without degradation
    - Extreme case is identity connection
    - $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} + \frac{\partial L}{\partial y}\frac{\partial f}{\partial x}$
  - Easy to optimize
    - Converge faster



https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Why ResNet works?

- Unravel
  - K skip-connections → 2^k paths
  - Ensemble modelling
    - Generalize better → accuracy



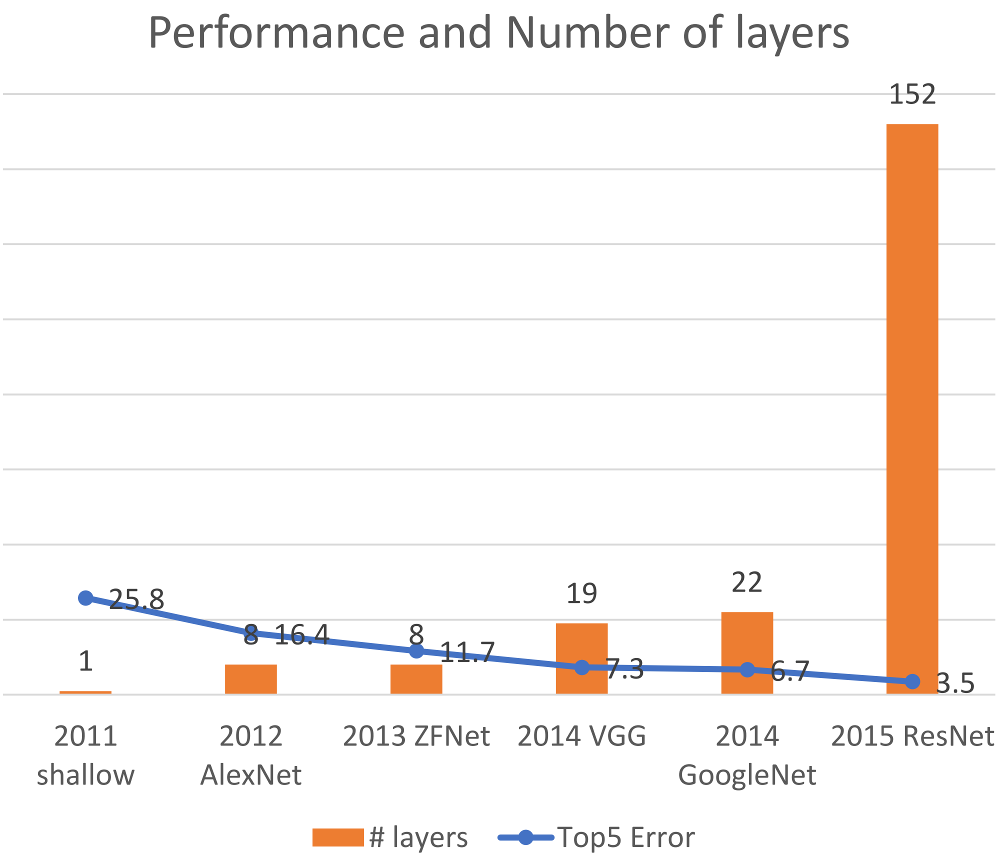
(a) Conventional 3-block residual network

(b) Unraveled view of (a)

A. Veit, M. Wilber and S. Belongie. Residual Networks Behave Like Ensembles of Relatively Shallow Networks. arXiv:1605.06431v2,2016

Summary

https://towardsdatascience.com/neural-network-architectures-156e5bad51ba



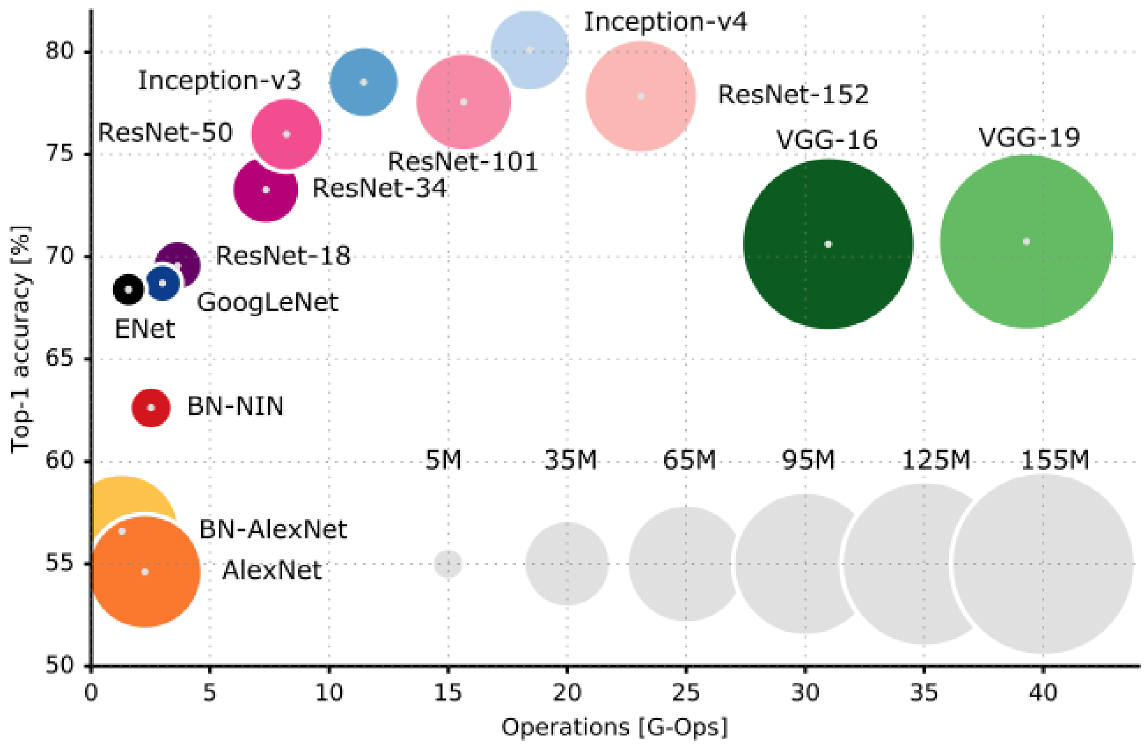


Image source: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba