Convolutional Networks

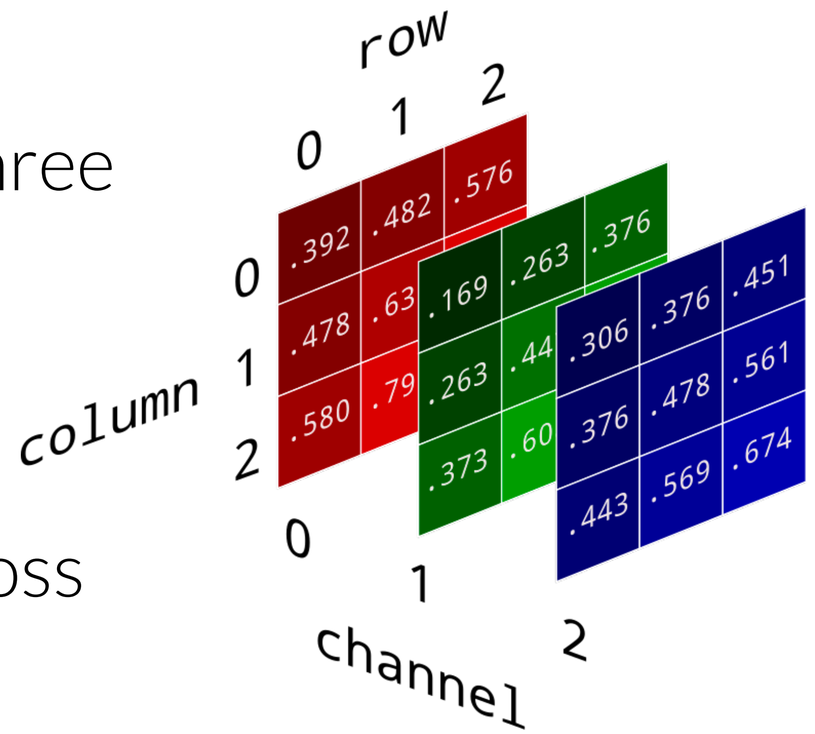
Principles of Deep Learning (AI502/IE408/IE511)

Sungbin Lim (UNIST AIGS & IE)

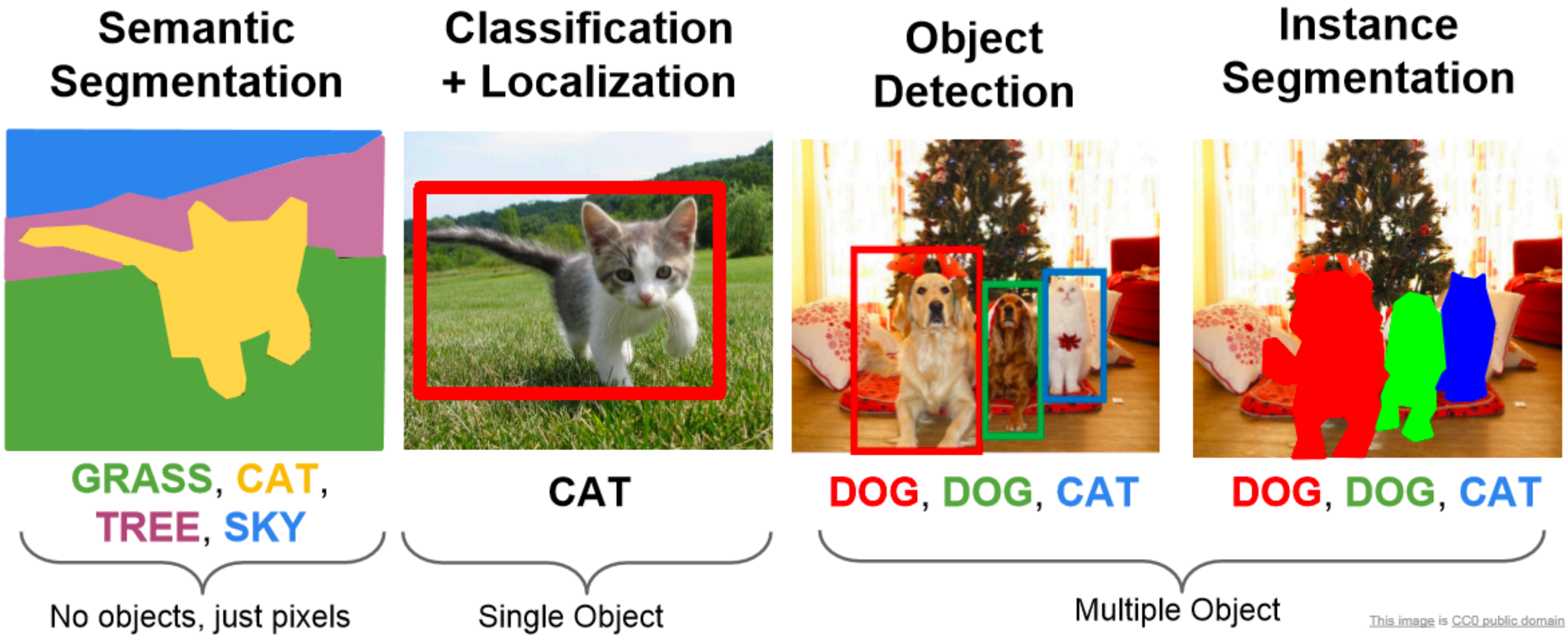
Contact: ai502deeplearning@gmail.com

Understanding Image Data

- Previously, we consider arbitrary type of data point $\mathbf{x} \in \mathbb{R}^d$
- Now we focus on the image data $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$
- Color images are represented as a collection of three two-dimensional `numpy` arrays
  - one each for **red**, **green**, and **blue** channels
  - each pixel has three values
- What will happen if we apply `mean` operation across all three color channels? 🤔



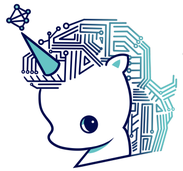
Computer Vision Tasks



Fei-Fei Li (2017)

Invariance in Object Recognition

- **Spatial invariance**
  - Our visual recognition system should not be overly concerned with the precise location of the object in the image
  - ***what object looks like*** does not depend upon ***where the object is located***
- **Locality principle**
  - the contents in distant regions are irrelevant

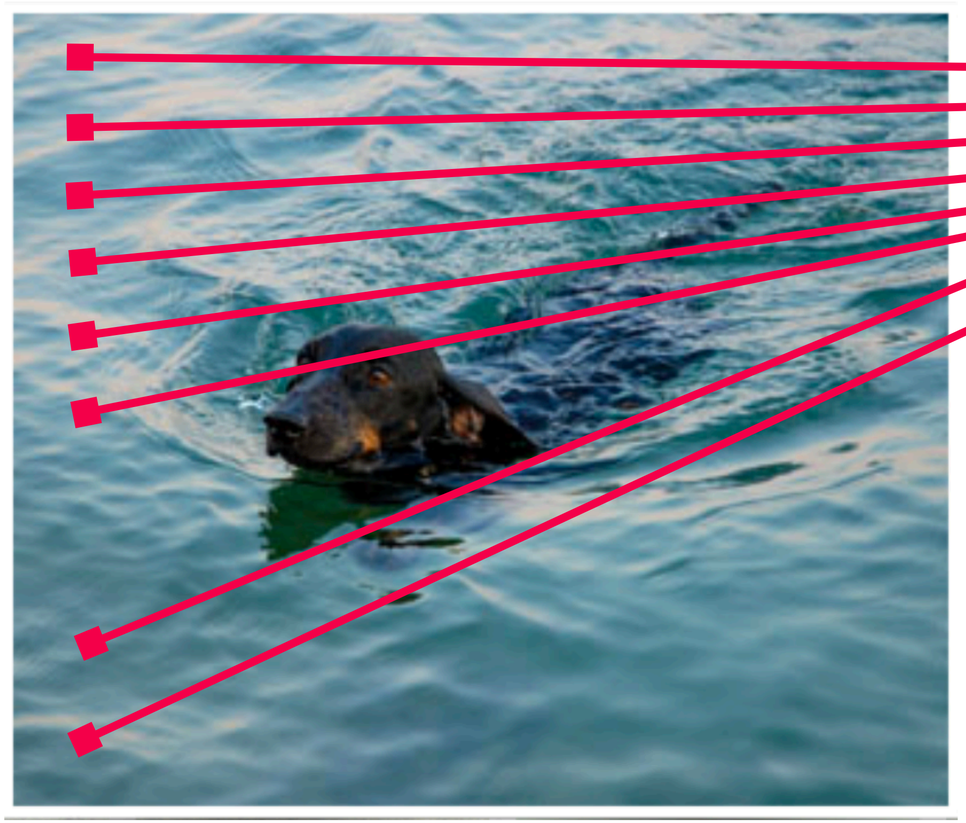


these are key design principles in the convolutional networks

- Hence we can sweep the image with an object detector that could assign a score to each patch, indicating the likelihood that the patch contains object
  - many object detection and segmentation algorithms are based on this

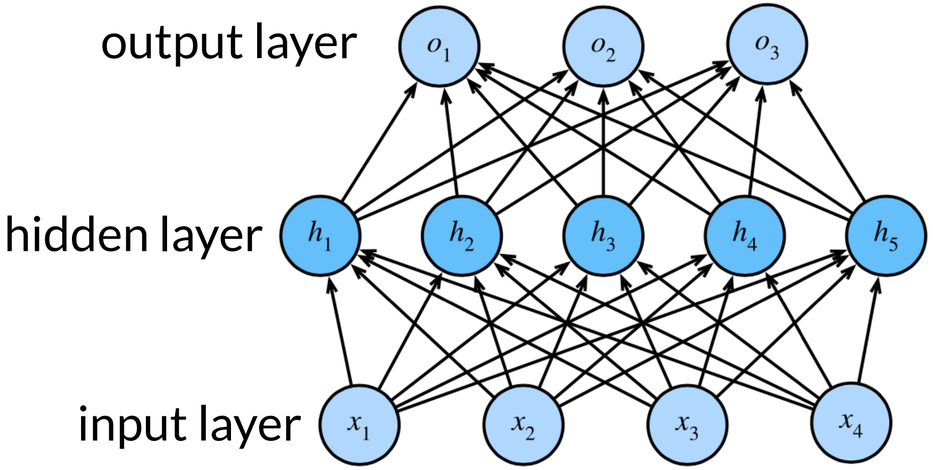
How MLP works?

- memory inefficient
- hard to optimize
- prone to overfitting

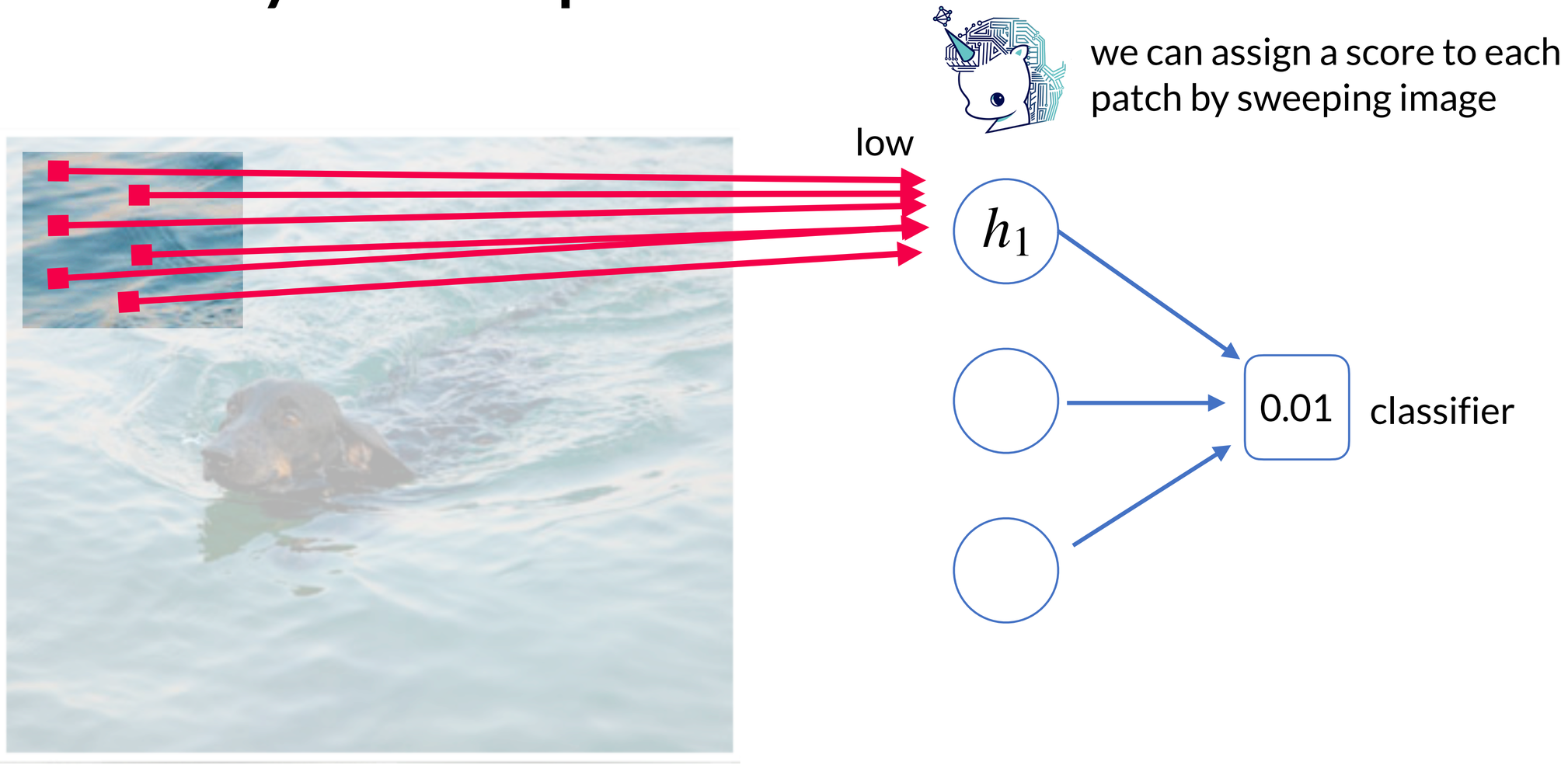


$$h_i = \sum_j w_{ij} x_j$$

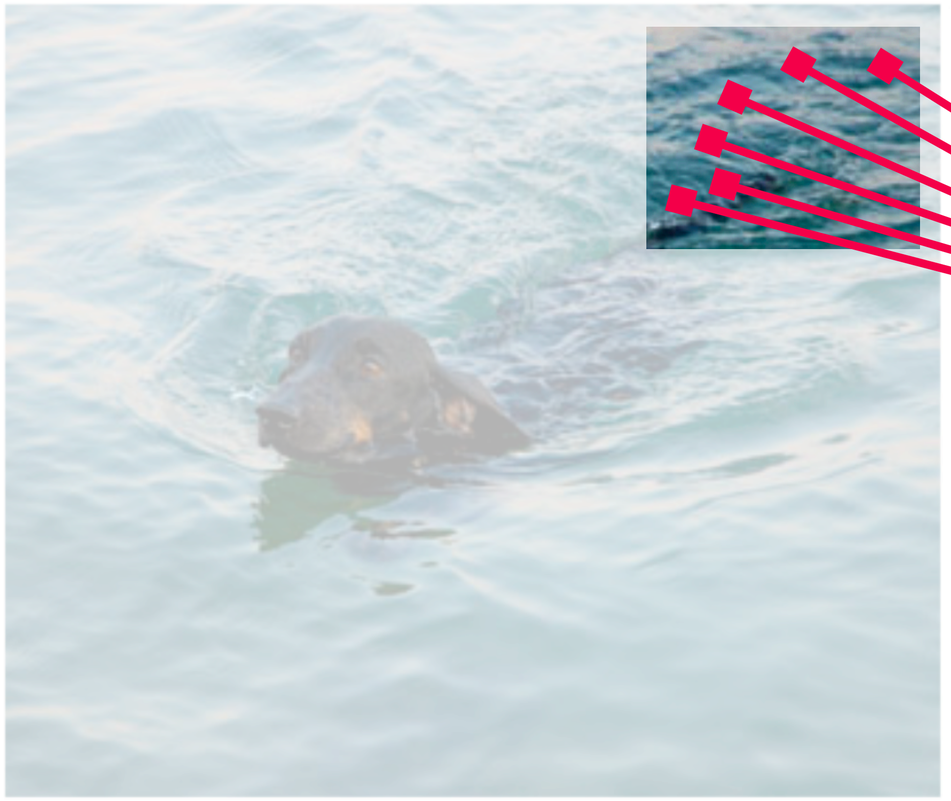
In MLP, every pixel has own parameters



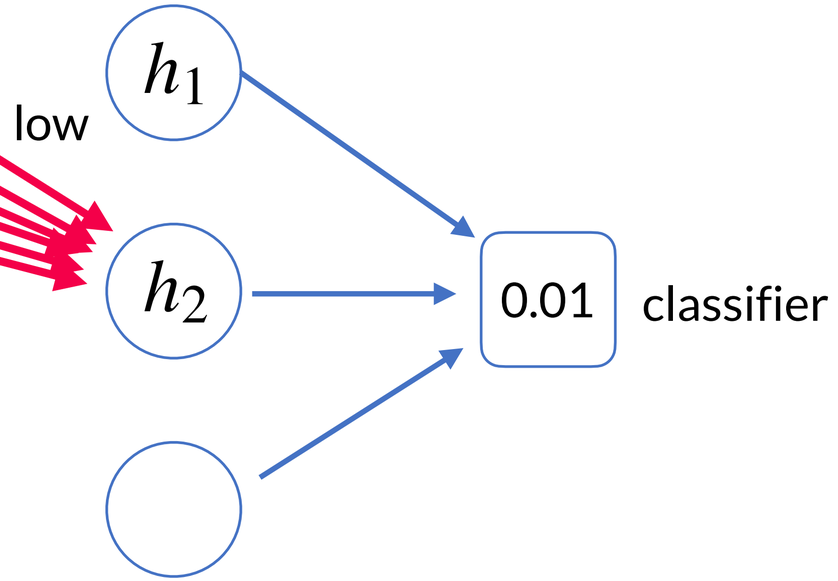
Locality Principle



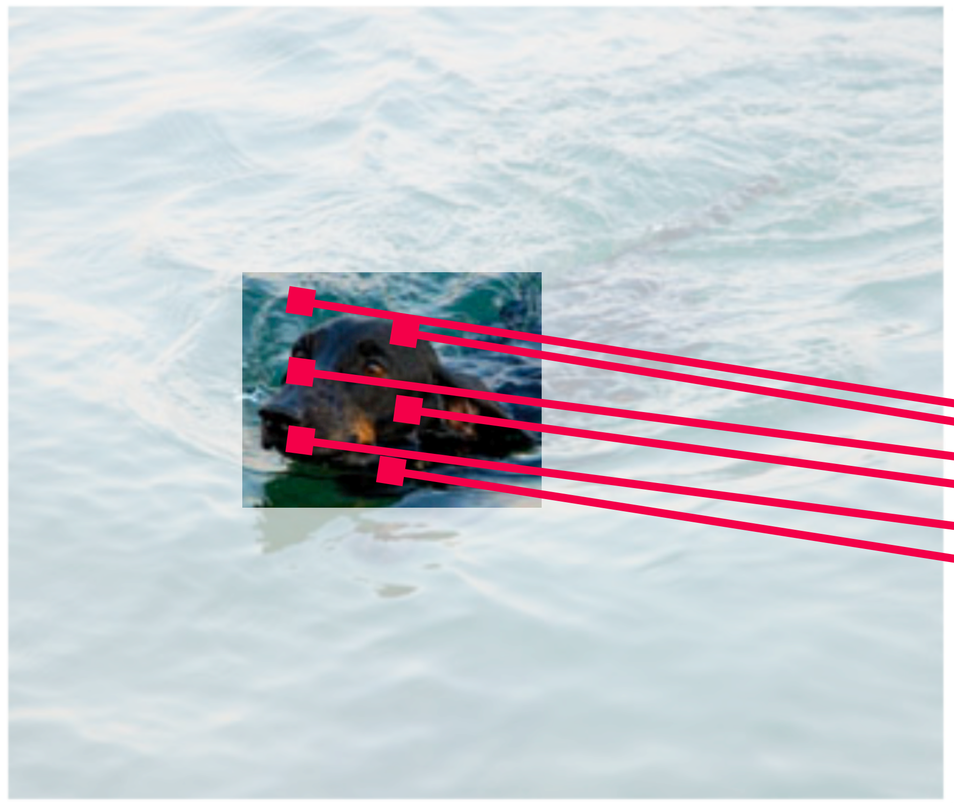
Locality Principle



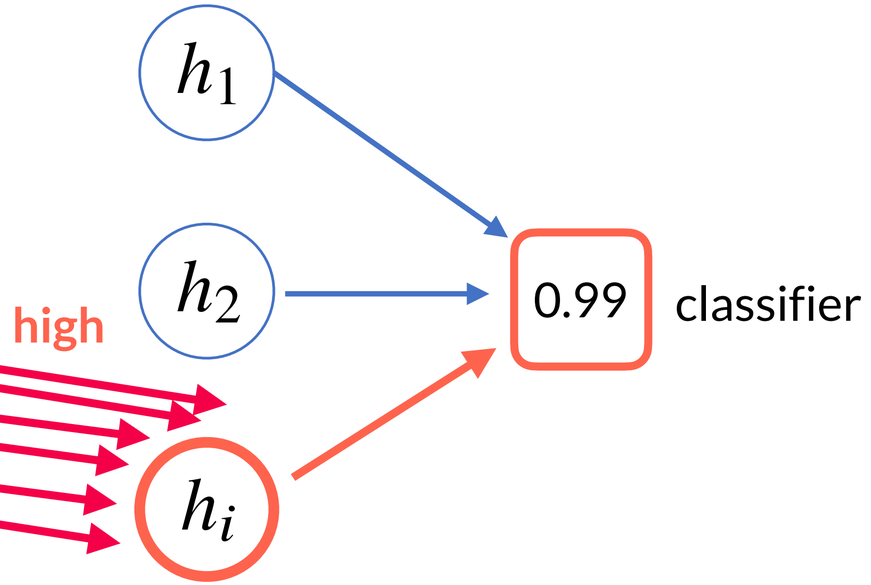
low score indicates there is no object in the corresponding region



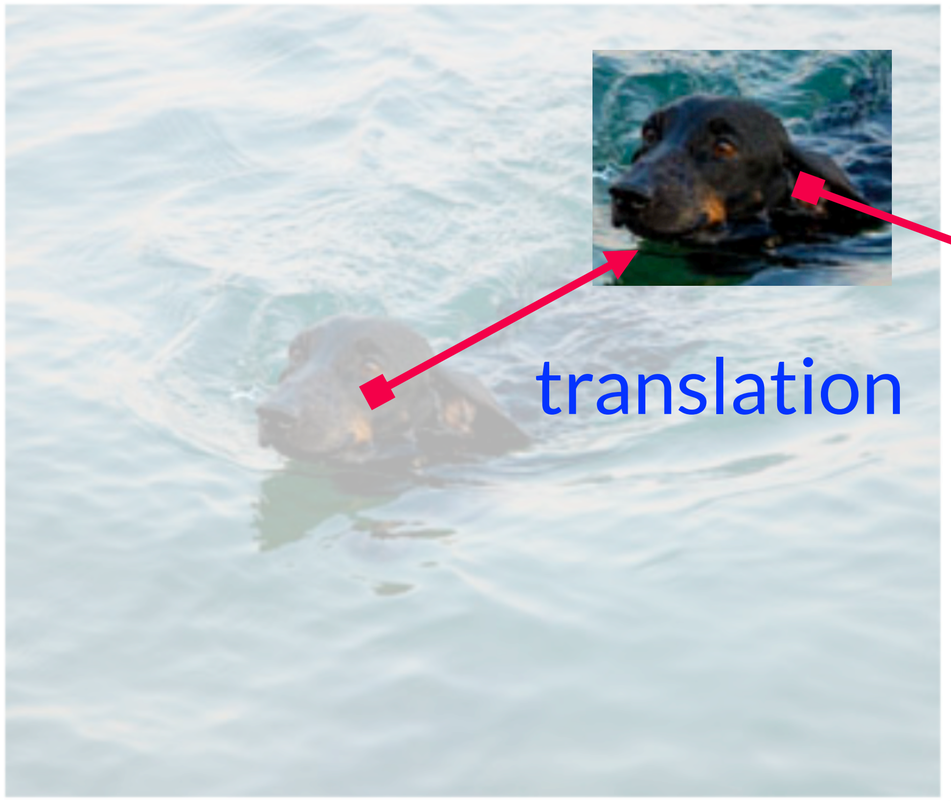
Locality Principle



object detector will assign high score
when the patch includes the target object



Spatial Invariance



if the object is translated, then the detector catches the information in the translated patch

high

h_1

h_2

h_i

0.99

classifier

Key Idea: Convolution Operation

- Fully connected layer

$$h(i) = \sum_{\ell} W_i(\ell) x(\ell)$$

for each location i, there exists the corresponding weight matrix W_i

Key Idea: Convolution Operation

- Fully connected layer

$$V_i(p) := W_i(i+p)$$

$$h(i) = \sum_{\ell} W_i(\ell)x(\ell) = \sum_{p} V_i(p)x(i+p)$$

we need to train $V_i(\cdot)$ for each i

Key Idea: Convolution Operation

- Fully connected layer

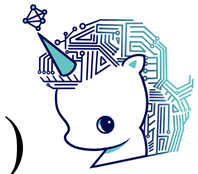
$$V_i(p) := W_i(i+p)$$

$$h(i) = \sum_{\ell} W_i(\ell)x(\ell) = \sum_{p} V_i(p)x(i+p)$$

- Now let's define **convolution**

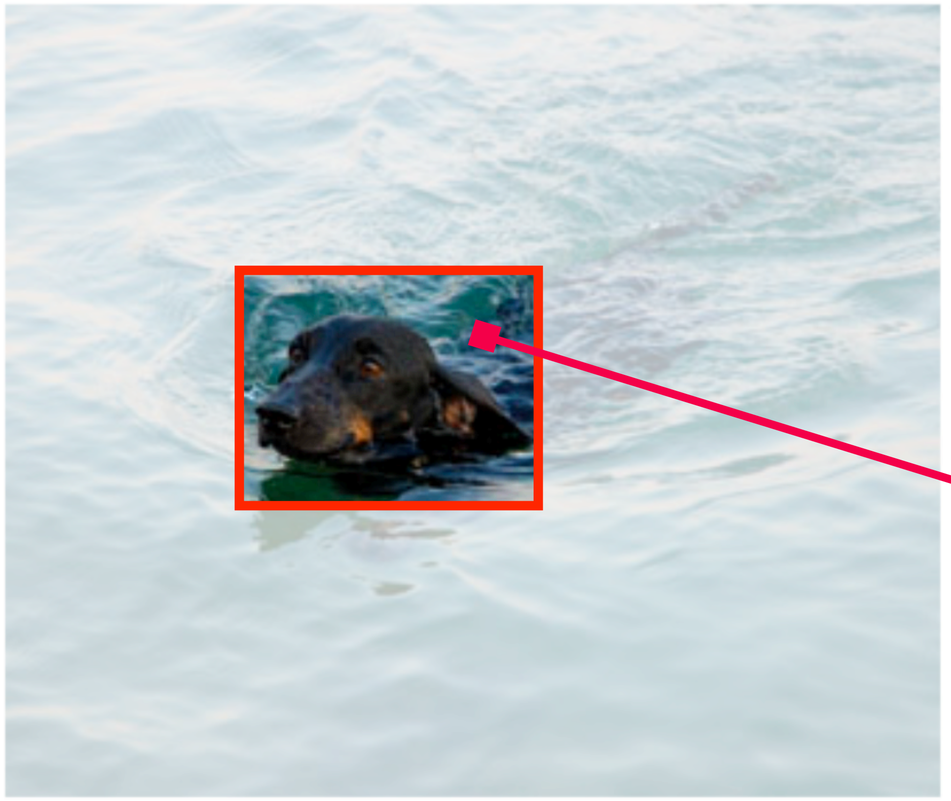
$$h(i) = \sum_{p} V(p)x(i+p)$$

$|p| \leq \Delta$ patch size



convolution shares parameters $V(\cdot)$ along pixel locations i

Convolution for Locality Principle



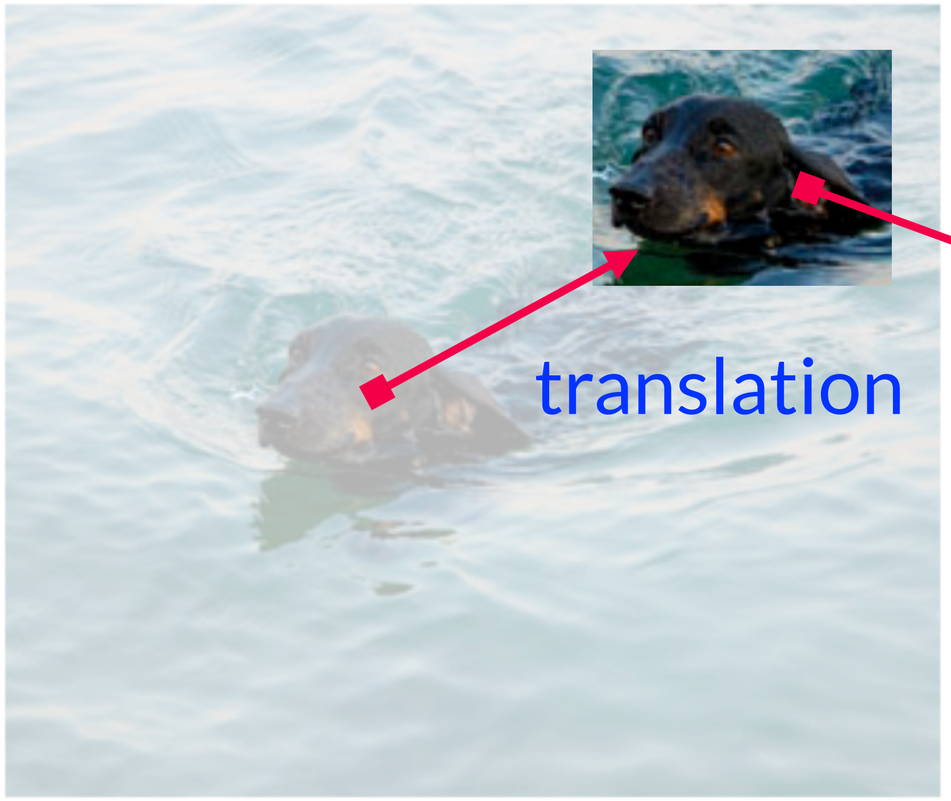
$$h_1 = \sum_p w_p x_{1+p} \qquad x_{1+p} \in \text{1st Patch}$$

$$h_2 = \sum_p w_p x_{2+p} \qquad x_{2+p} \in \text{2nd Patch}$$

$$h_i = \sum_p w_p x_{i+p} \qquad x_{i+p} \in i\text{-th Patch}$$

we process the visual info through local patch

Convolution for Spatial Invariance



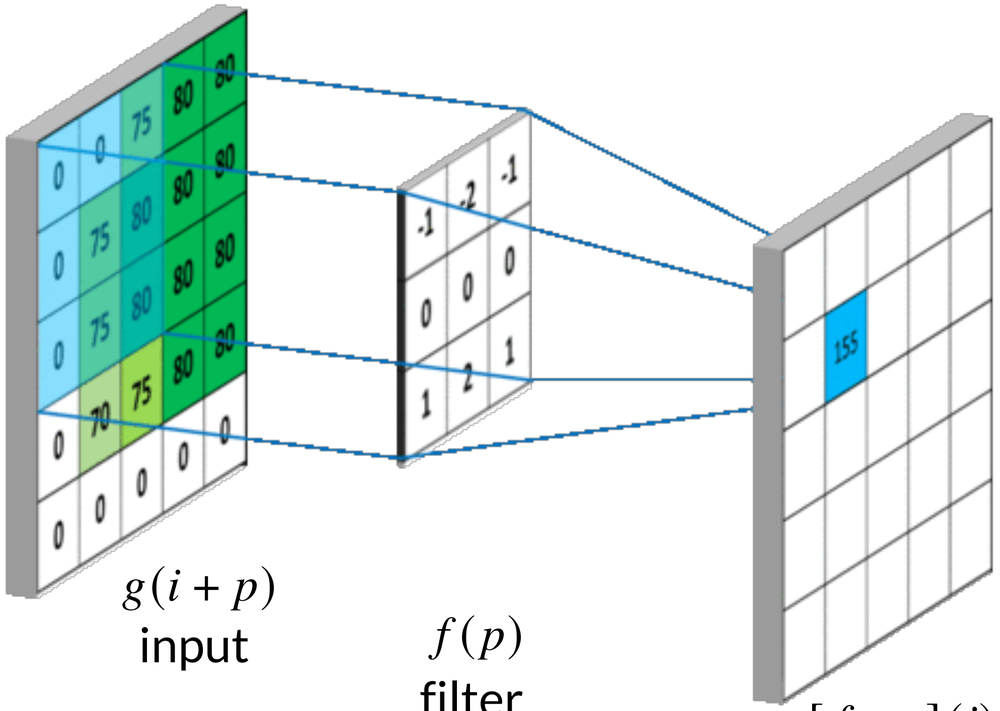
here we use **same** weight across different patches!

$$h_1 = \sum_p w_p x_{1+p} \qquad x_{1+p} \in \text{1st Patch}$$

$$h_2 = \sum_p w_p x_{2+p} \qquad x_{2+p} \in \text{2nd Patch}$$

$$h_i = \sum_p w_p x_{i+p} \qquad x_{i+p} \in i\text{-th Patch}$$

Dive into Convolution



1D Conv $$[f * g](i) = \sum_{p=1}^{d} f(p)g(i+p)$$

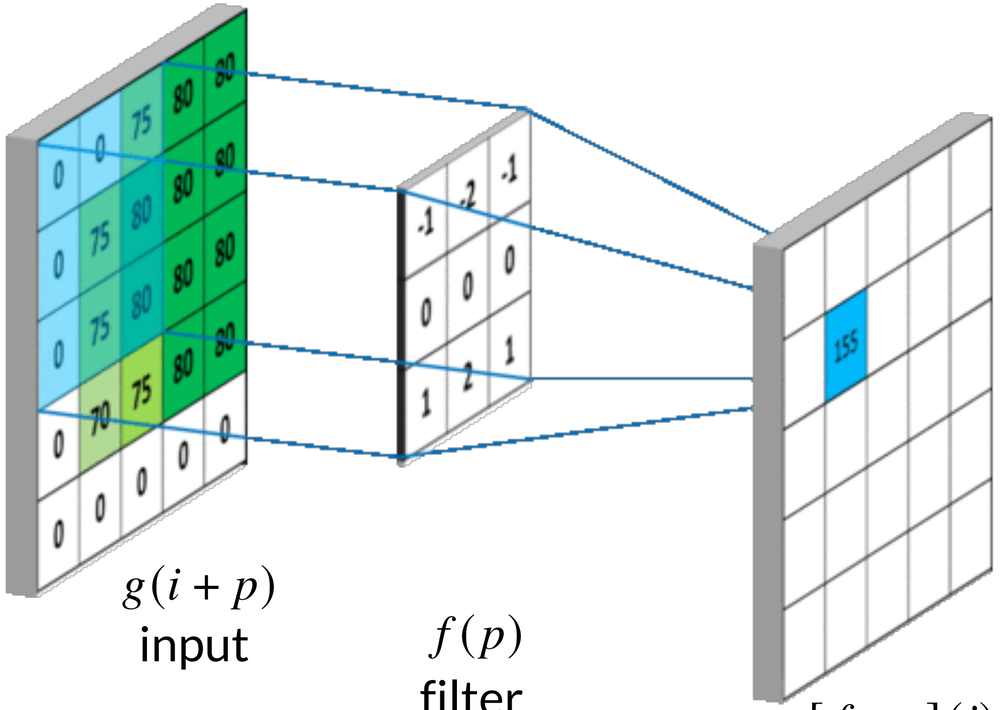
2D Conv $$[f * g](i,j) = \sum_{p,q} f(p,q)g(i+p, j+q)$$

3D Conv $$[f * g](i,j,k) = \sum_{p,q,r} f(p,q,r)g(i+p, j+q, k+r)$$

Dive into Convolution

filter is invariant w.r.t
the position of the input

1D Conv $$[f * g](i) = \sum_{p=1}^{d} f(p)\, g(i+p)$$



$g(i+p)$
input

$f(p)$
filter

$[f * g](i)$
output

2D Conv $$[f * g](i,j) = \sum_{p,q} f(p,q)\, g(i+p, j+q)$$

3D Conv $$[f * g](i,j,k) = \sum_{p,q,r} f(p,q,r)\, g(i+p, j+q, k+r)$$

What exactly filters do?

0.06	0.12	0.06
0.12	0.25	0.12
0.06	0.12	0.06

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

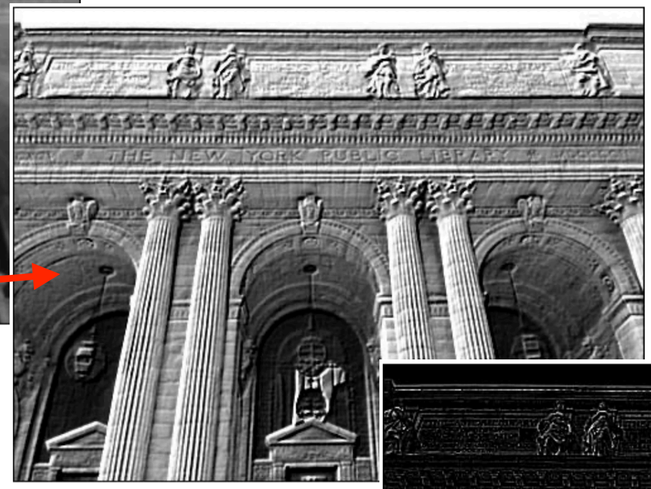
-1	-1	-1
-1	8	-1
-1	-1	-1

http://setosa.io/ev/image-kernels/

What exactly filters do?

Blur

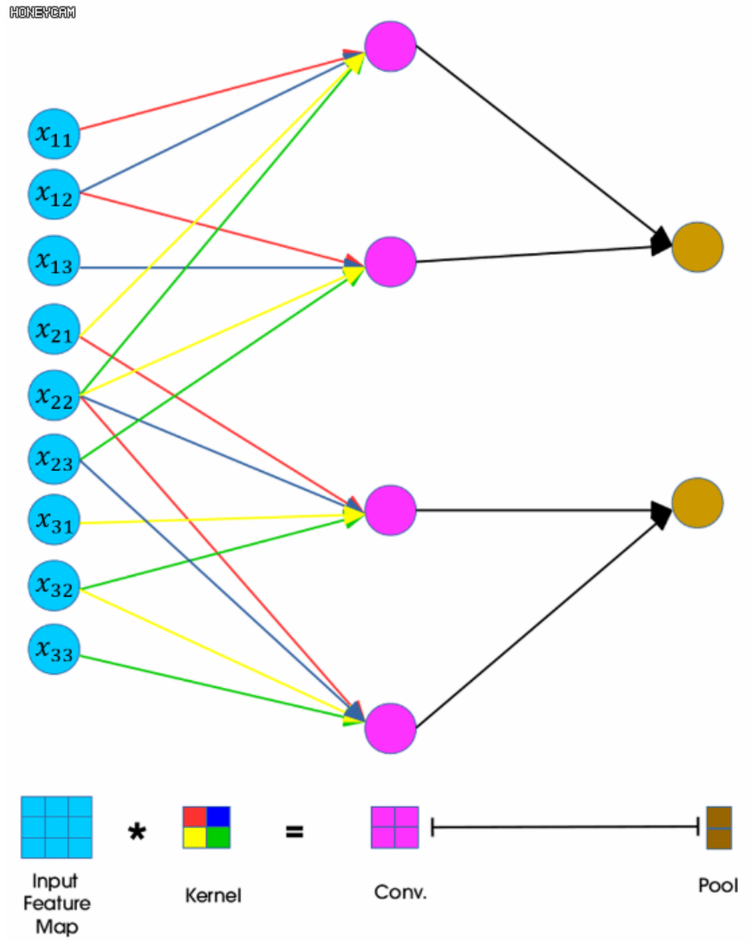
Emboss

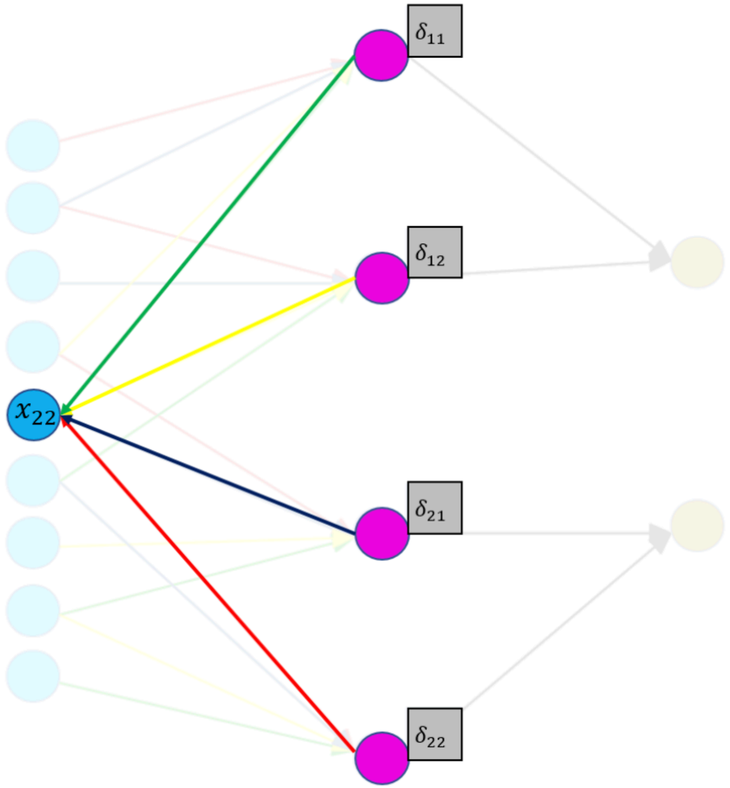


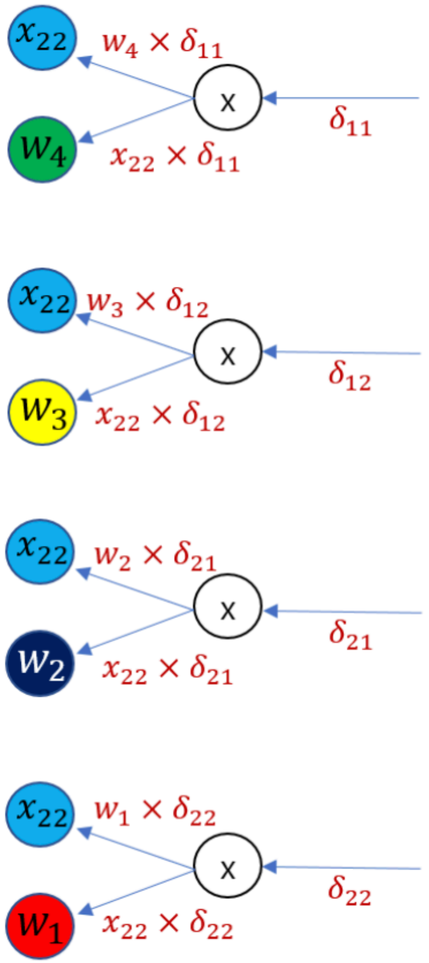
Outline

http://setosa.io/ev/image-kernels/

What about backpropagation?





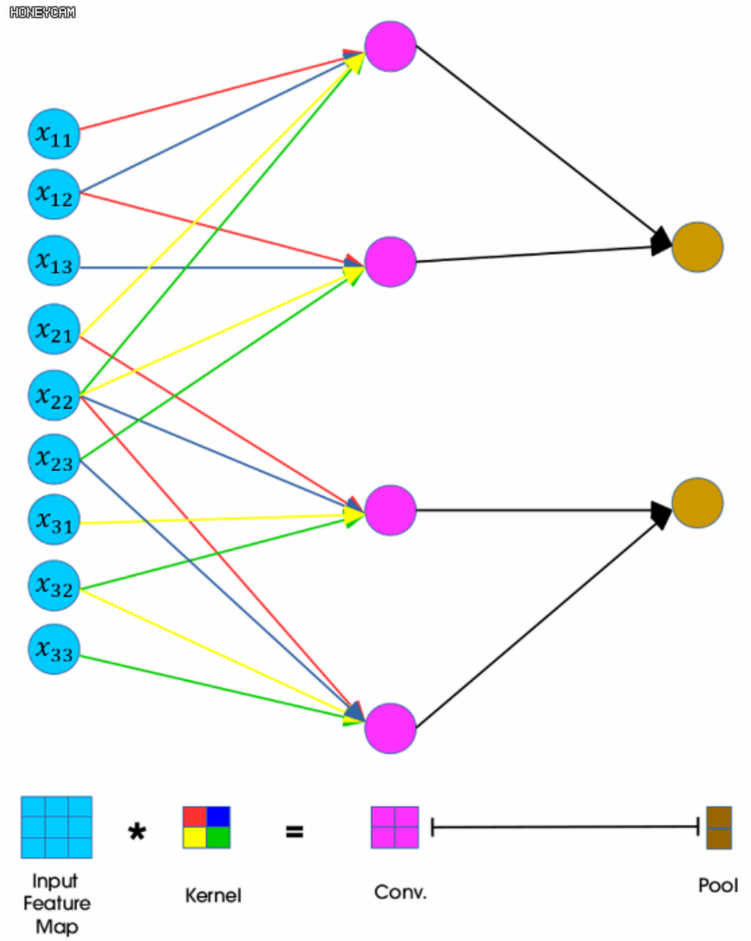


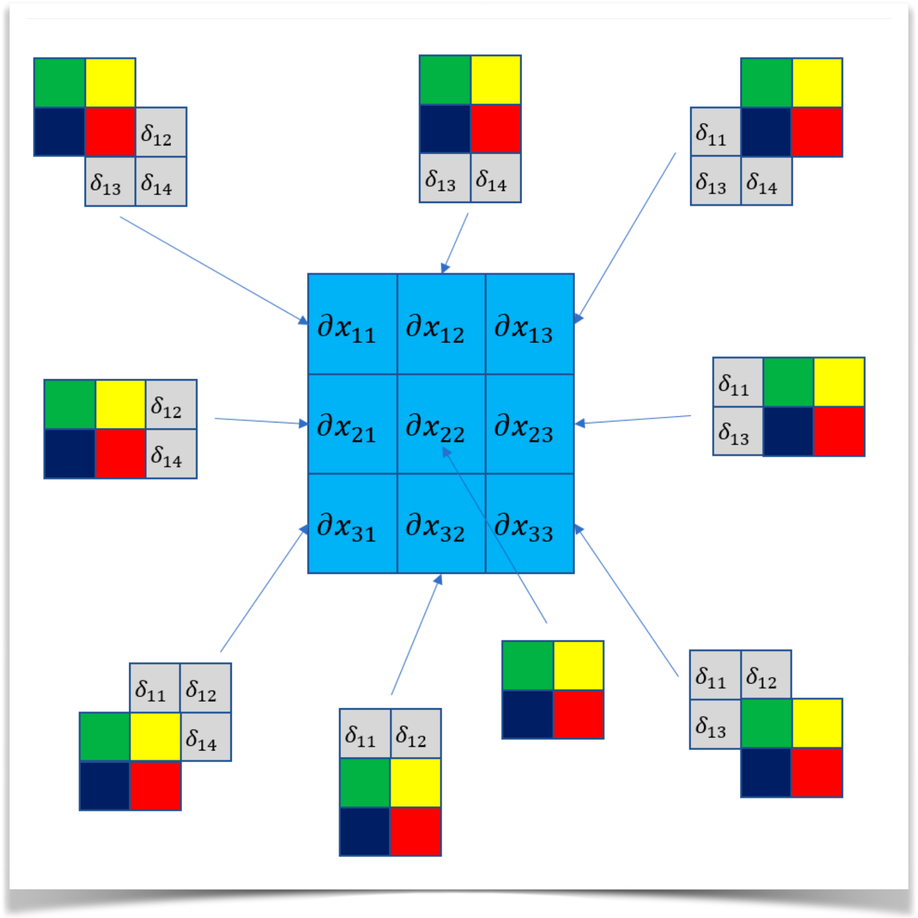
[ratsgo's blog (korean)](), [DeepGrid (english)]()

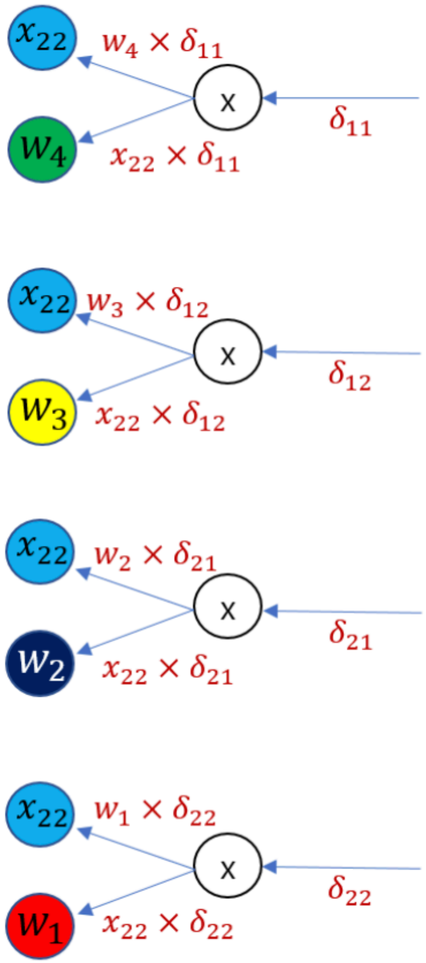
backward

computation graph

What about backpropagation?







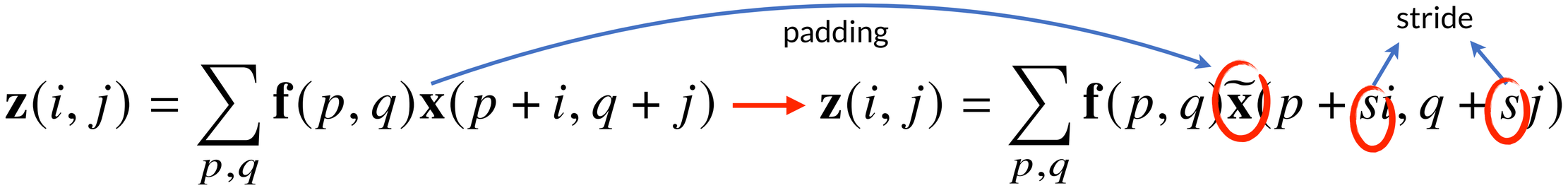
[ratsgo's blog (korean)](), [DeepGrid (english)]()

backward

computation graph

Stride & Padding

- Filters (kernels) generally have width and height greater than 1
  - applying successive convolutions, we tend to wind up with outputs that are considerably smaller than our input
  - *padding* is the most popular tool for handling this issue
    - if we want to reduce the dimension faster, *stride* is useful technique



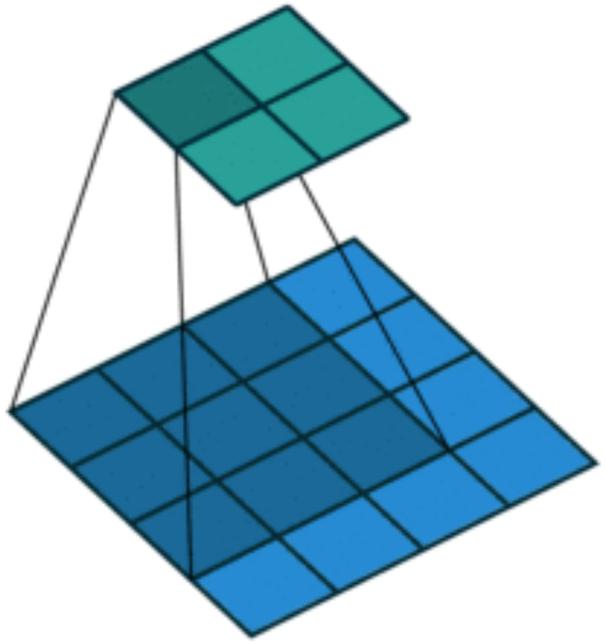
Stride & Padding

$$(h_{\text{out}}, w_{\text{out}}) = \left(\left\lfloor \frac{h_{\text{in}} - k_h + 2p_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{w_{\text{in}} - k_w + 2p_w}{s_w} \right\rfloor + 1 \right)$$

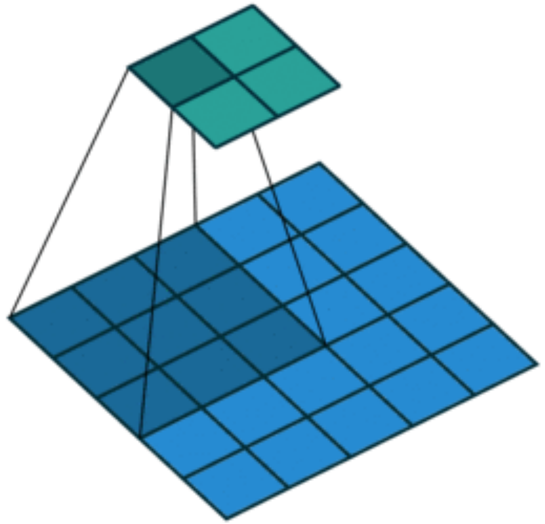
filter size

padding size

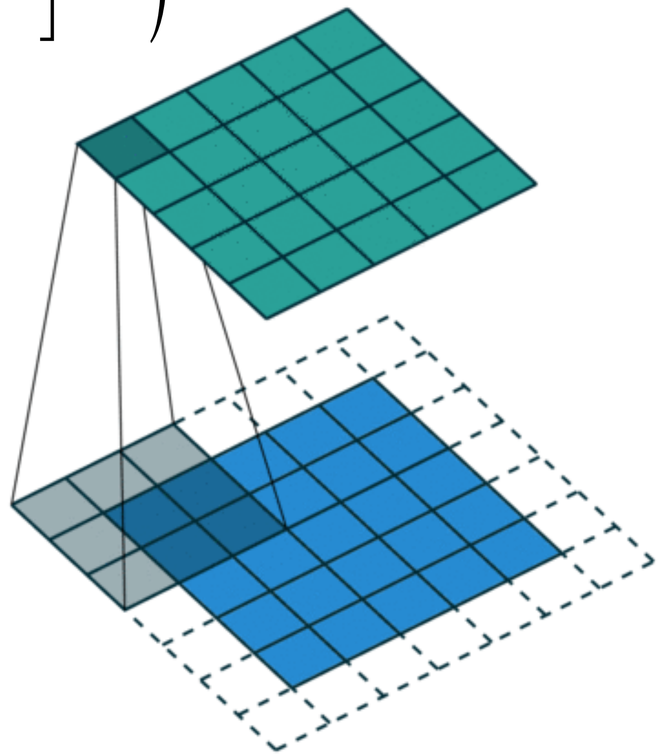
stride size



Vanilla convolution



convolution + stride



convolution + padding

Stride & Padding

- Example: vanilla 3x3 convolusion

input * filter

output

0	1	2
2	2	0
0	1	2

filter

Stride & Padding

$$(h_{\text{out}}, w_{\text{out}}) = \left(\left\lfloor \frac{h_{\text{in}} - k_h + 2p_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{w_{\text{in}} - k_w + 2p_w}{s_w} \right\rfloor + 1 \right)$$

- Example: convolution operation (3x3) + zero padding (1x1) + stride (2x2)

input * filter

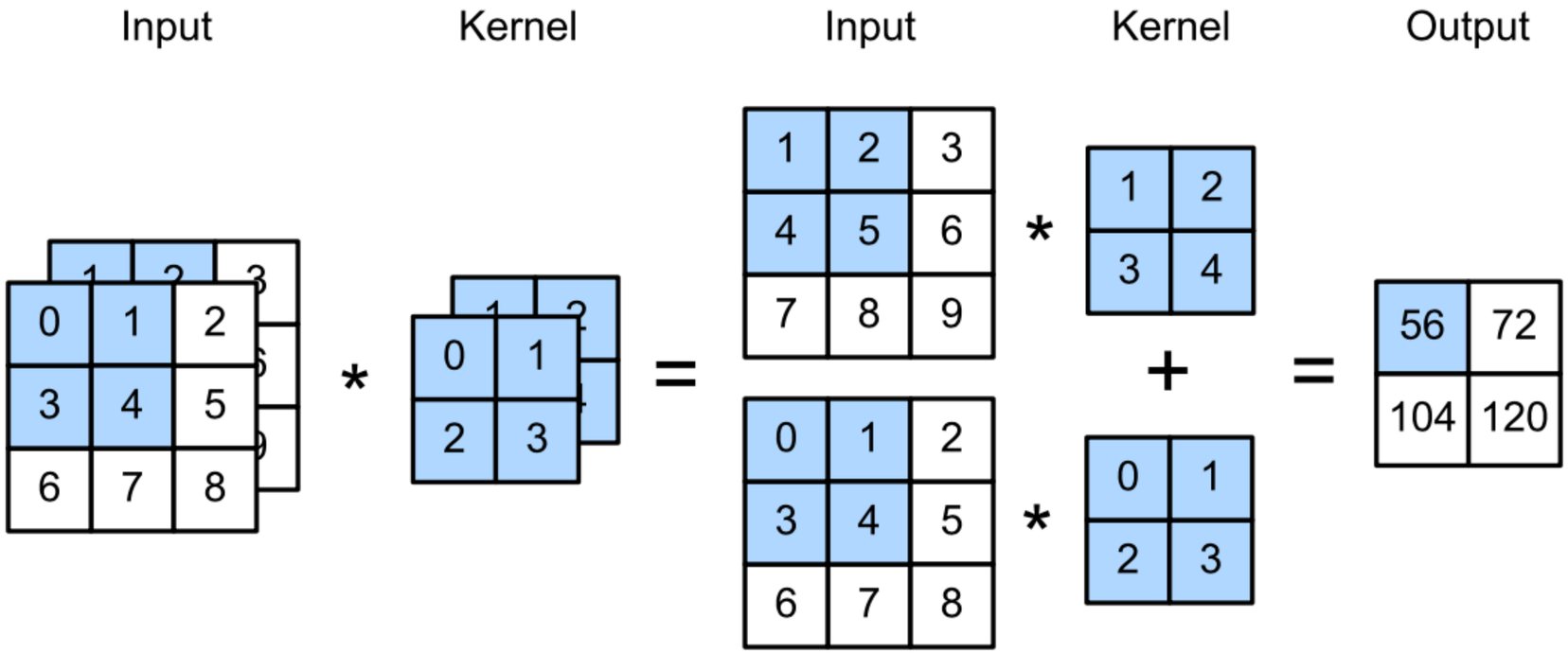
output

2	0	0
1	0	0
0	1	1

filter

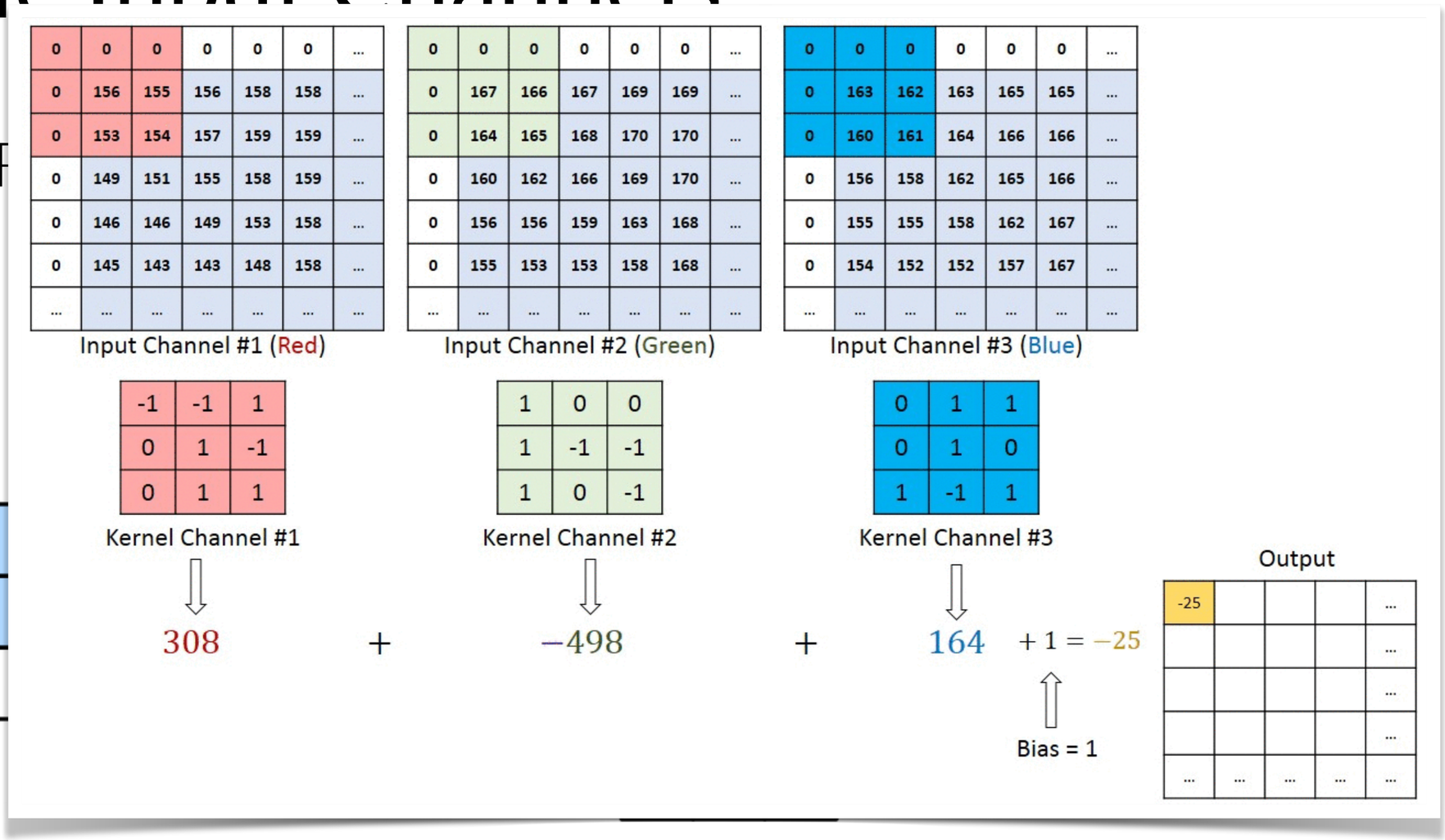
Multiple Input Channels

- Recall an RGB image has 3 input channels; **red**, **green**, and **blue**



Multiple Input Channels

- Recall an F

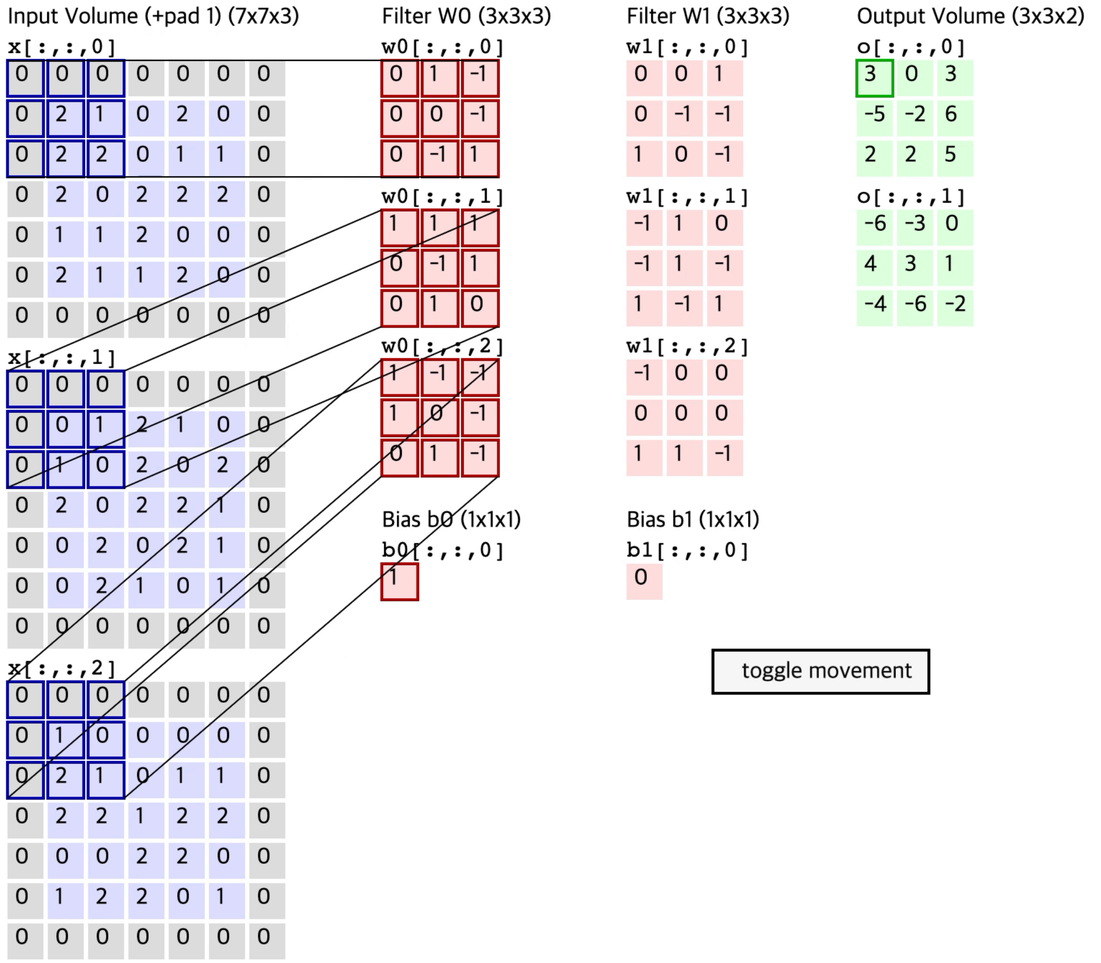


https://blog.e-kursy.it/deeplearning4j-cnn/video/html/#33.0

Multiple Output Channels

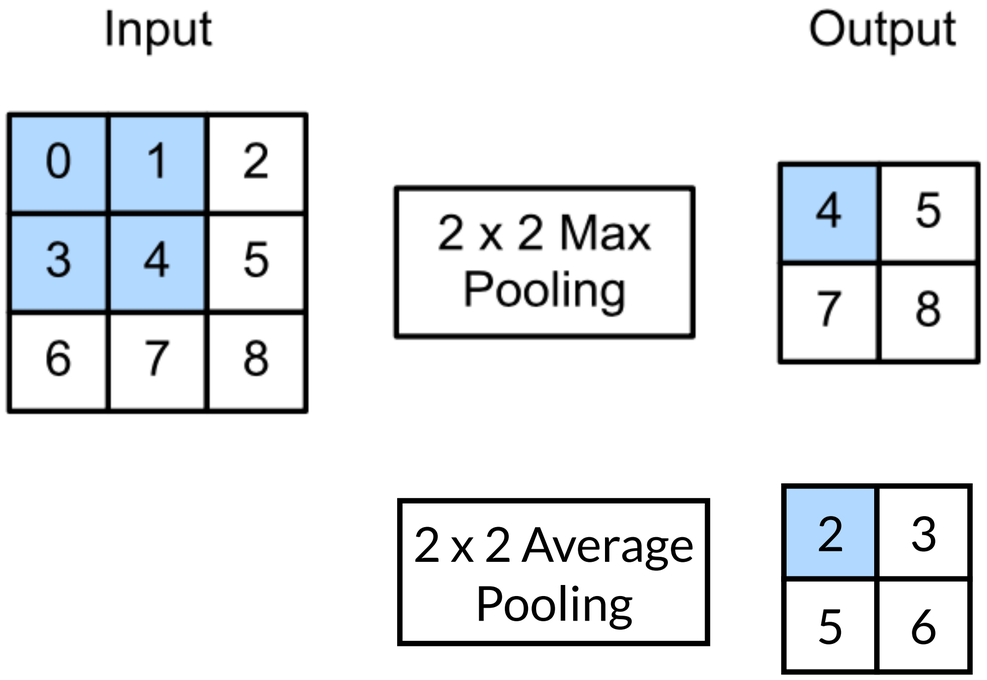
- c_{in} : number of input channels
- c_{out} : number of output channels
- k_h, k_w : height and width of filter
- Filter shape: $c_{\text{out}} \times c_{\text{in}} \times k_h \times k_w$

(FAQ) What is the dimension of output when we apply convolution with the shape $c_{\text{out}} \times c_{\text{in}} \times k_h \times k_w$, padding of (p_h, p_w), stride of (s_h, s_w)?



Pooling

- Gradually reduce the spatial resolution
  - aggregate information
  - downsampling
  - parameter-free operator
  - no learning
- Max pooling
- Average pooling



Modern Convolutional Nets

LeNet, AlexNet, VGG, Inception, ResNet, DenseNet

LeNet

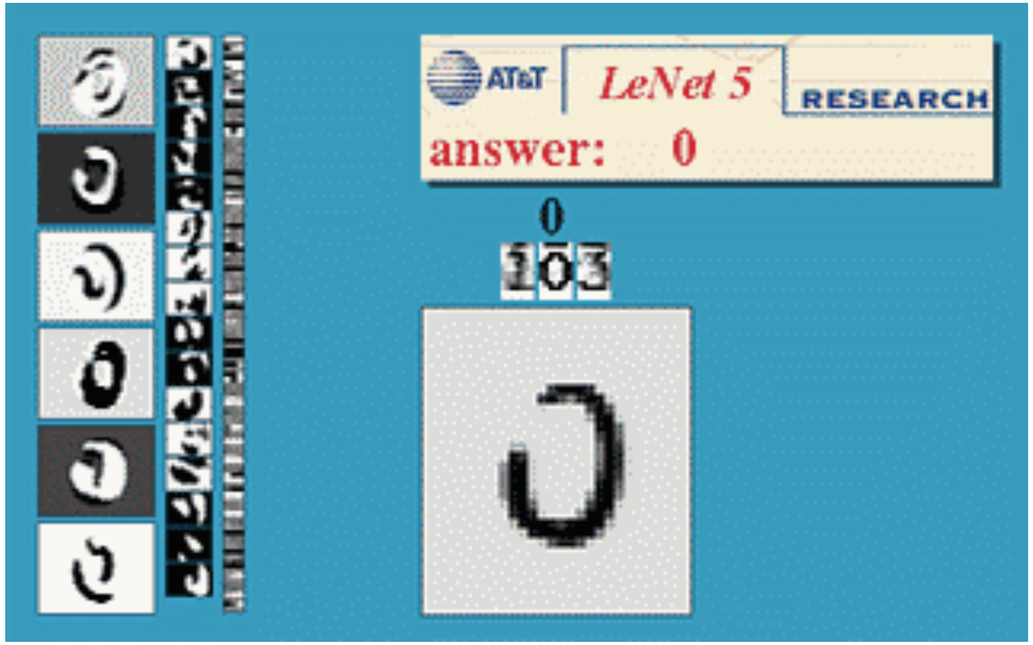
- One of the first published convolutional neural networks
  - savings in the number of parameters
  - keep the image in grid
  - train CNN with backprop
- Adopted to recognize digits for processing deposits in ATM


Yann LeCun


Léon Bottou


Yoshua Bengio


Patrick Haffner

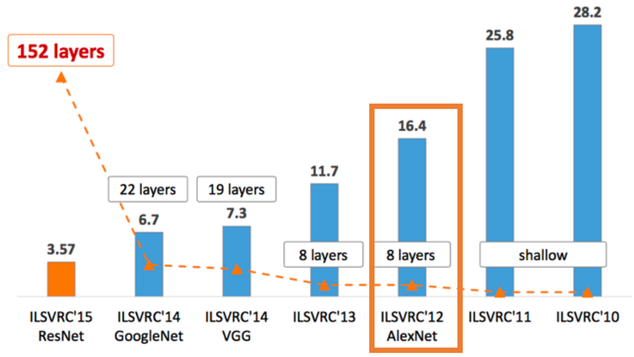


Gradient-Based Learning Applied to Document Recognition, Yann Lecun et al., ***Proc. of the IEEE*** (1998)

Missing Ingredients

- Data
  - deep learning models require large amounts of data
  - ImageNet data was released at 2009 (by Fei-Fei Li)
  - ImageNet competition pushed computer vision and machine learning research forward
- Hardware
  - take hundreds of epochs and computationally heavy linear algebra operations → needs many GPUs!

AlexNet



Alex Krizhevsky

Ilya Sutskever

Geoffrey E. Hinton

- Until 2012, the representation was calculated mechanically
  - SIFT, SURF, HOG, …
- Another groups believed features ought to be learned
- Interestingly, AlexNet learned feature extractors that resembled traditional filters

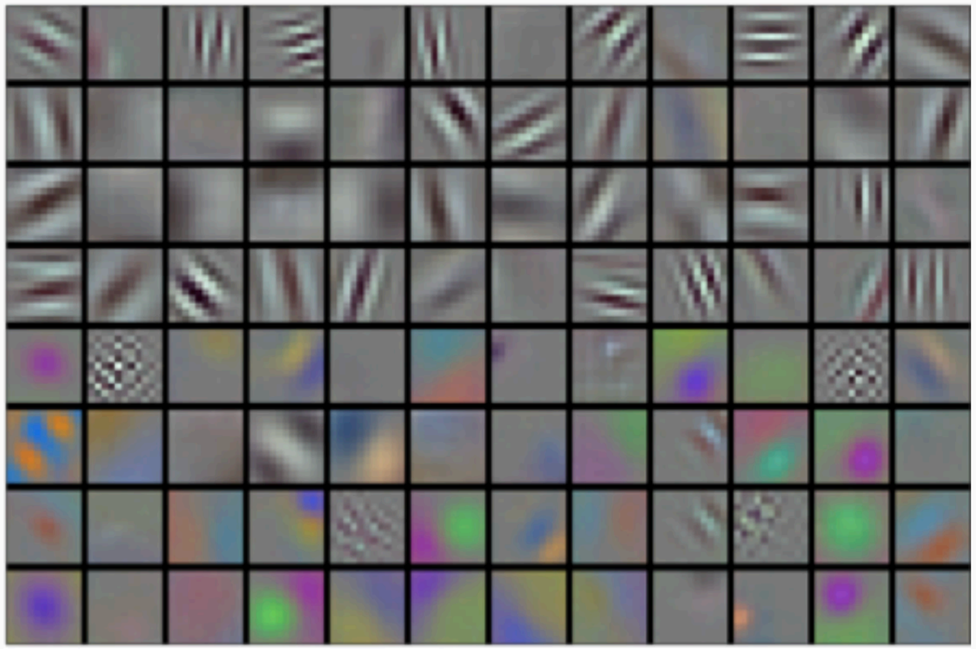


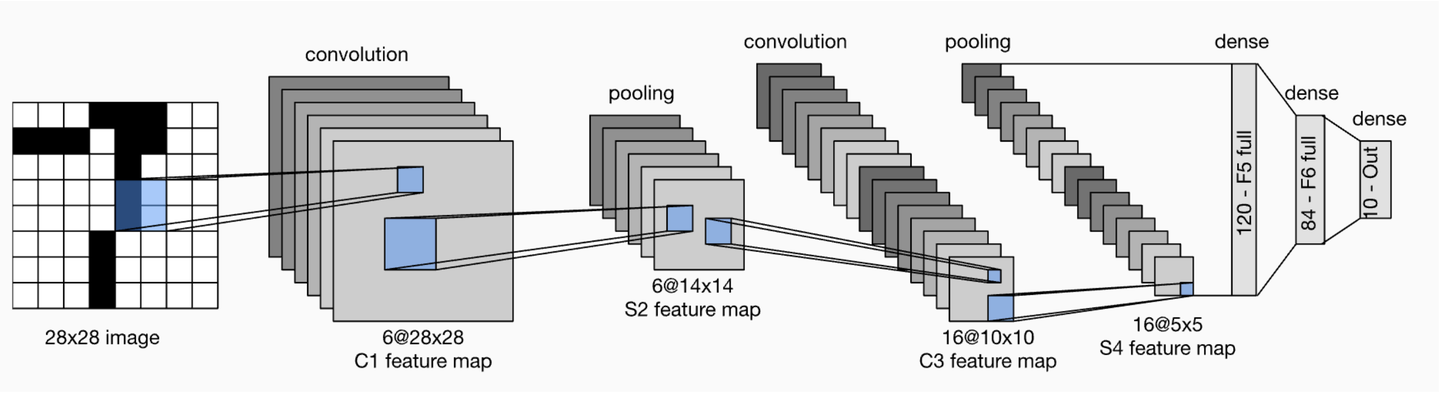
Image filters learned by the first layer of AlexNet

ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky et al., ***NIPS*** (2012)

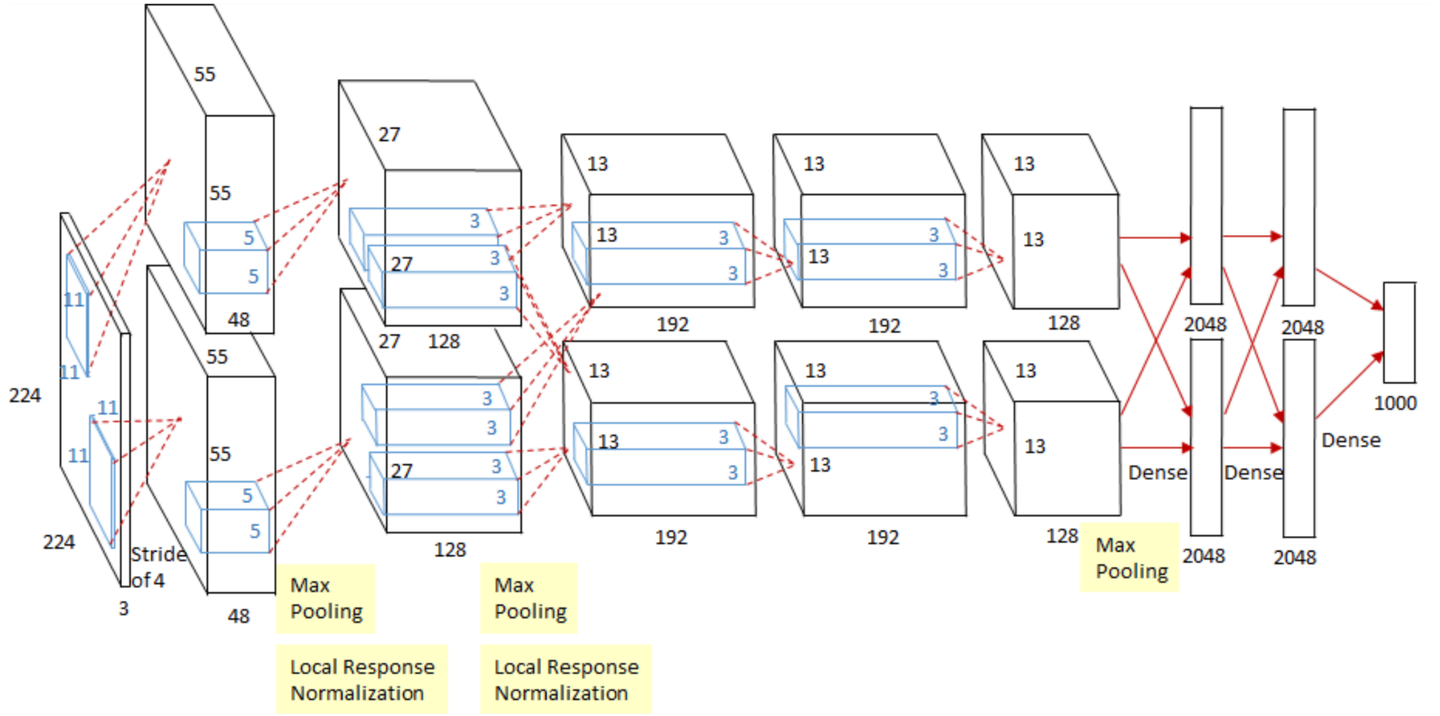
AlexNet

- LeNet
  - sigmoid
  - average pooling
- AlexNet (+ 2GPUs)
  - ReLU
  - max pooling
  - Dropout
  - Augmentation

LeNet



AlexNet

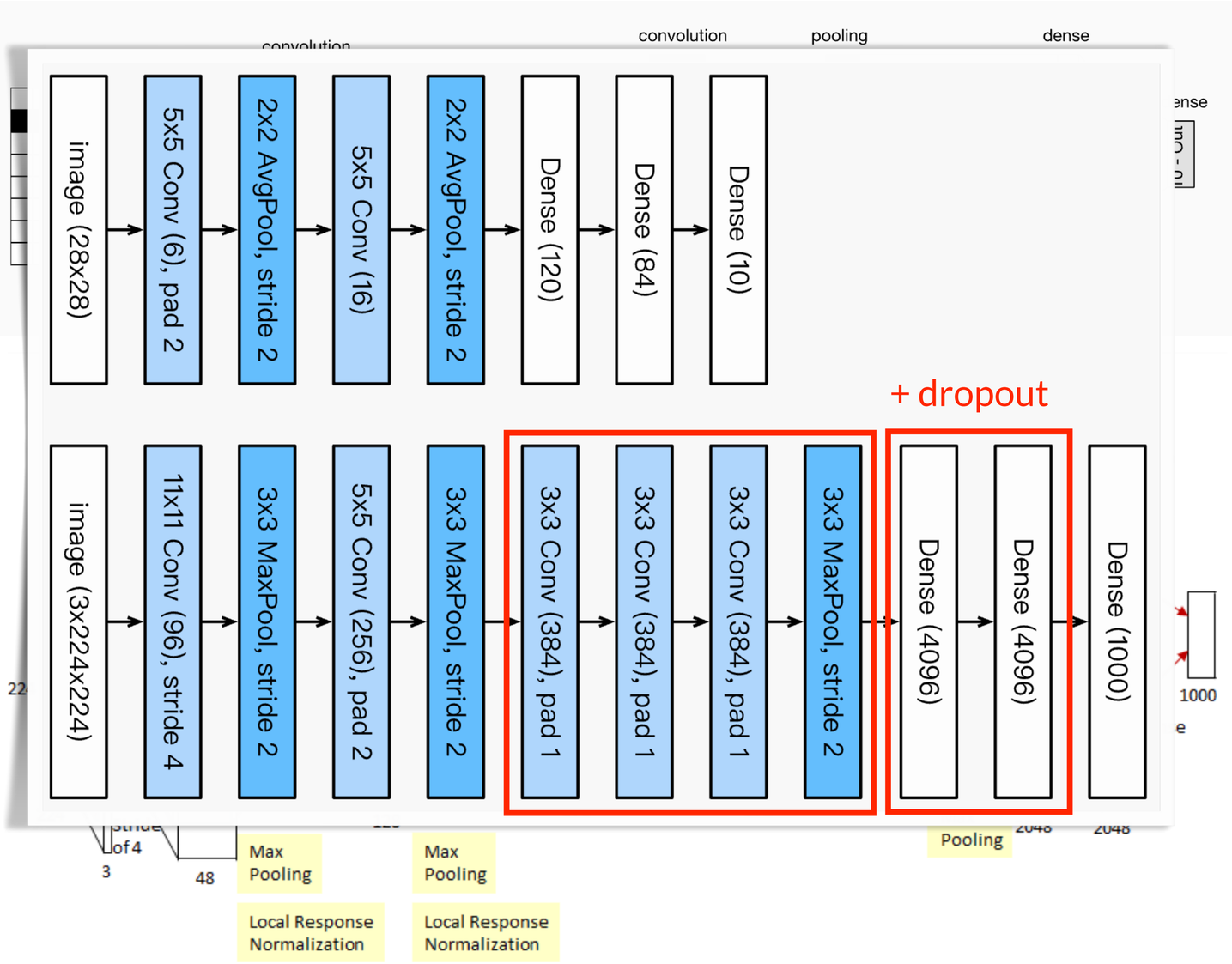


AlexNet

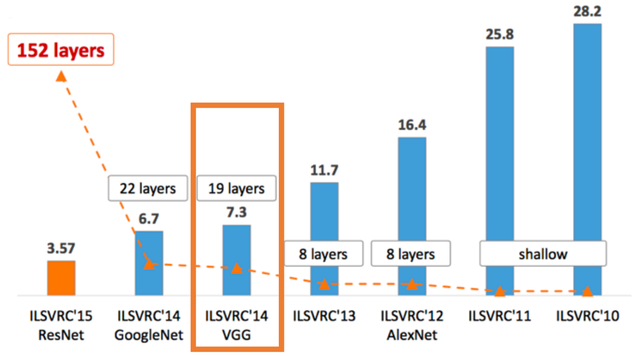
- LeNet
  - sigmoid
  - average pooling
- AlexNet (+ 2GPUs)
  - ReLU
  - max pooling
  - Dropout
  - Augmentation

LeNet

AlexNet



VGG

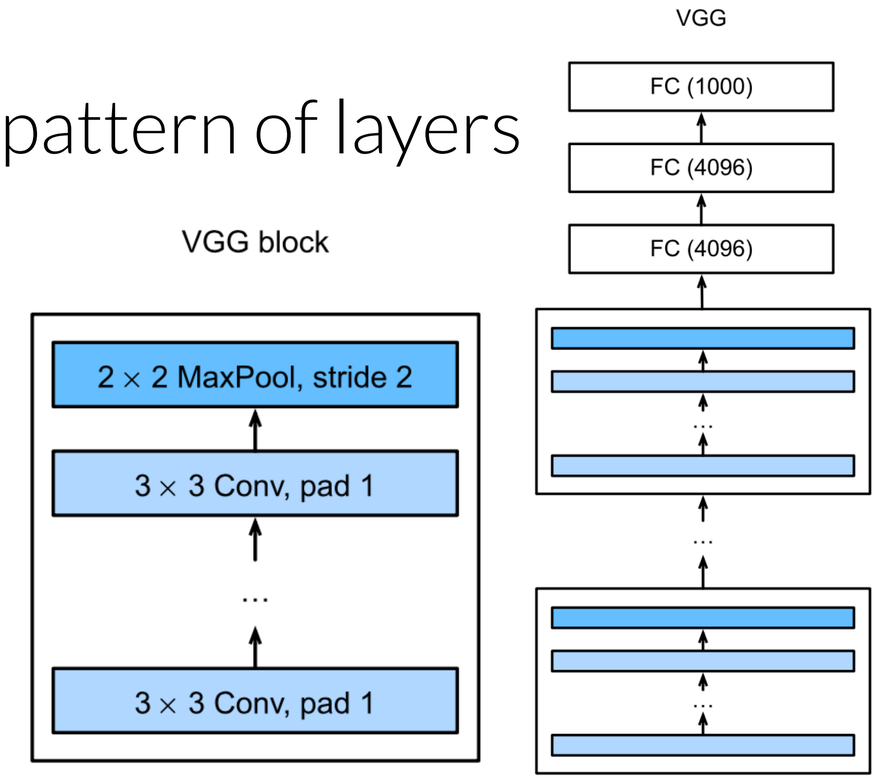


Karen Simonyan

Andrew Zisserman

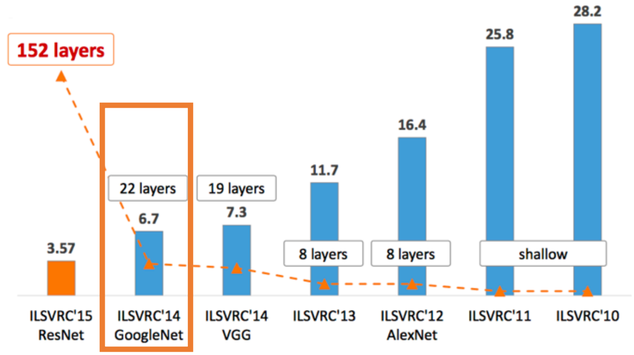
- **V**isual **G**eometry **G**roup of Oxford Univ. suggested the idea of using ***blocks***, repeating pattern of layers
- Basic building block of convolutional layer
  - `Conv2D() + ReLU() + Pooling()`
  - `vgg_block` consists of a sequence of convolutional layers

```
[Conv2D(out,in,3,padding=1), ReLU()] x N
+ MaxPool2D(3,stride=2)
```



Very Deep Convolutional Networks for Large-Scale Image Recognition, Simonyan & Zisserman, (2014)

Inception

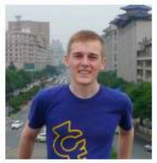


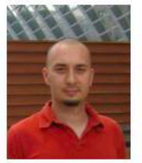

Christian Szegedy, Google


Wei Liu, UNC


Yangqing Jia, Google


Pierre Sermanet, Google


Scott Reed, University of Michigan

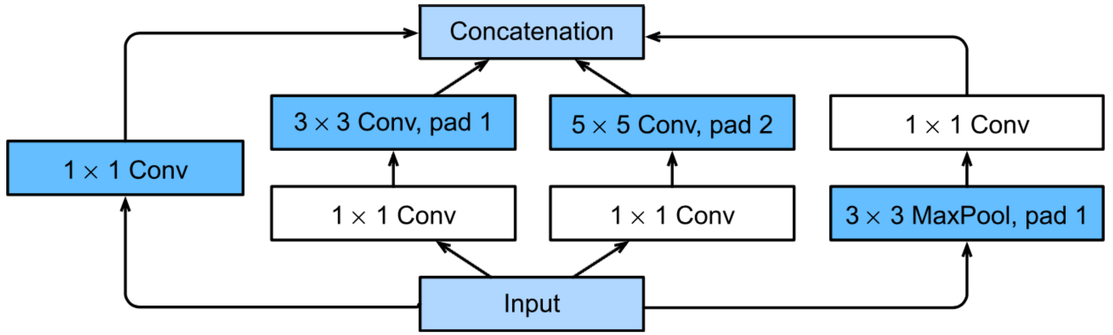

Dragomir Anguelov, Google


Dumitru Erhan, Google


Vincent Vanhoucke, Google


Andrew Rabinovich, Google

- Google won the ImageNet challenge at 2014
  - combined NiN and repeated blocks paradigms
  - advantageous to employ a combination of various filters
- Inception Blocks
  - consists of four parallel paths
  - convolution with multiple filter sizes

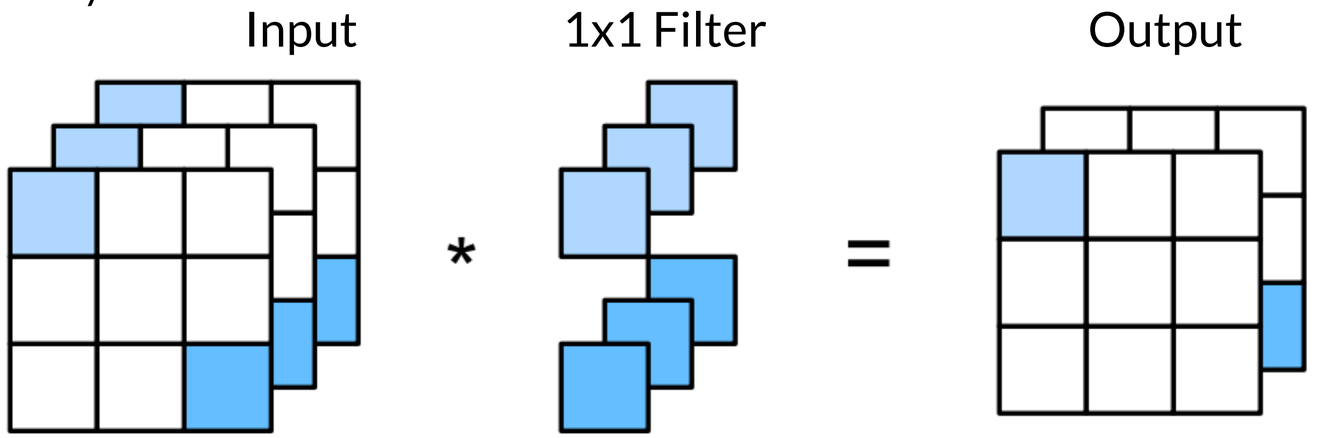


Going deeper with convolutions, Szegedy et al., ***CVPR*** (2015)

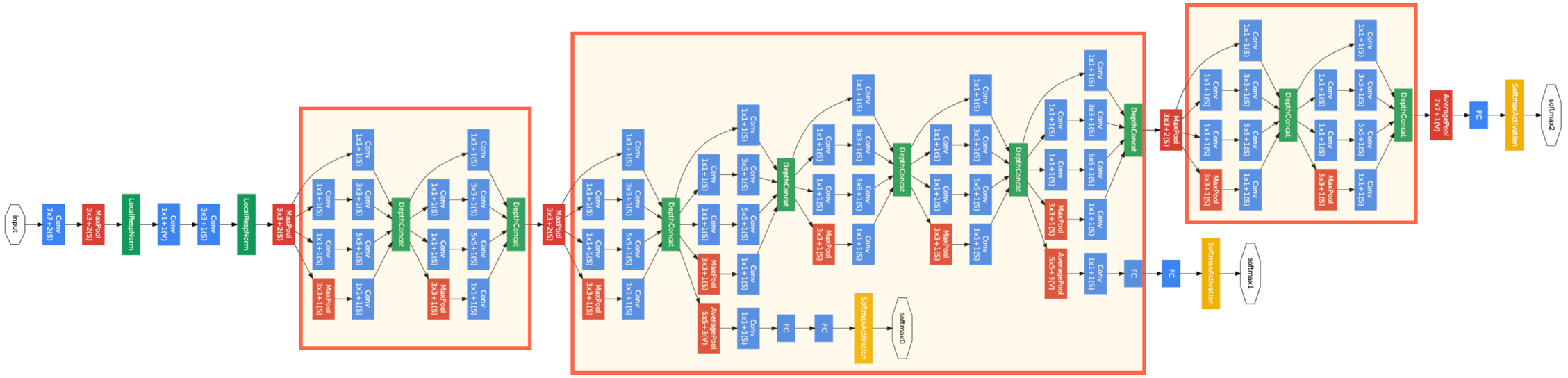
1x1 Convolution Layer

- 1x1 Conv layer is used to adjust number of channels
  - they are popular operations in the designs of deep networks
  - each element of output is derived from a linear combination of elements at the *same position* in the input image
  - this is similar to fully connected layer
  - requires $c_{\text{out}} \times c_{\text{in}}$ weights

we often use 1x1 Conv for dimension reduction

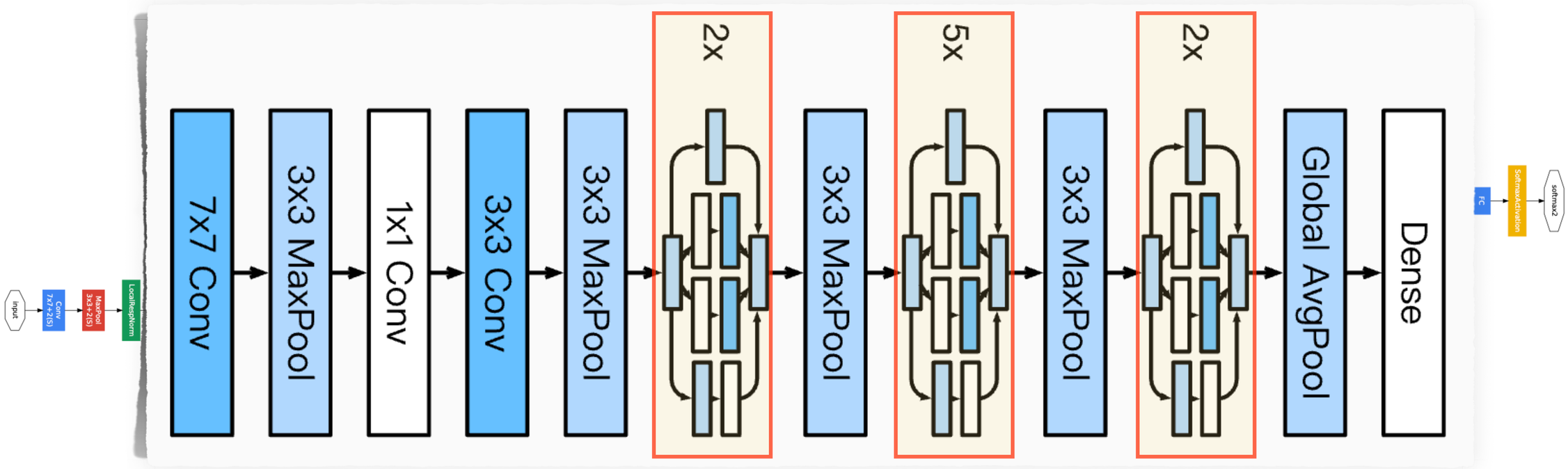


Inception Architecture



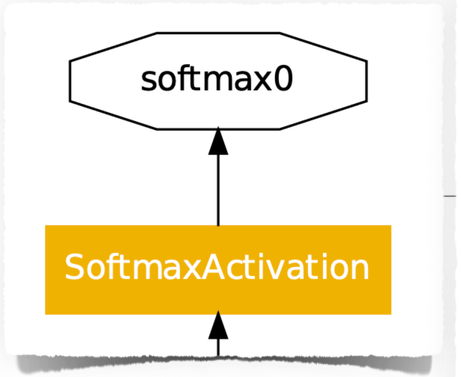
22 layers! Now the era of deep learning begins!

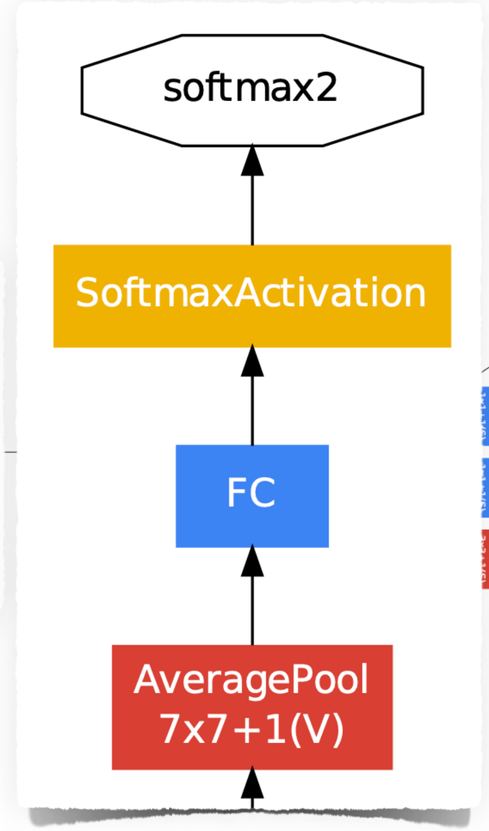
Inception Architecture



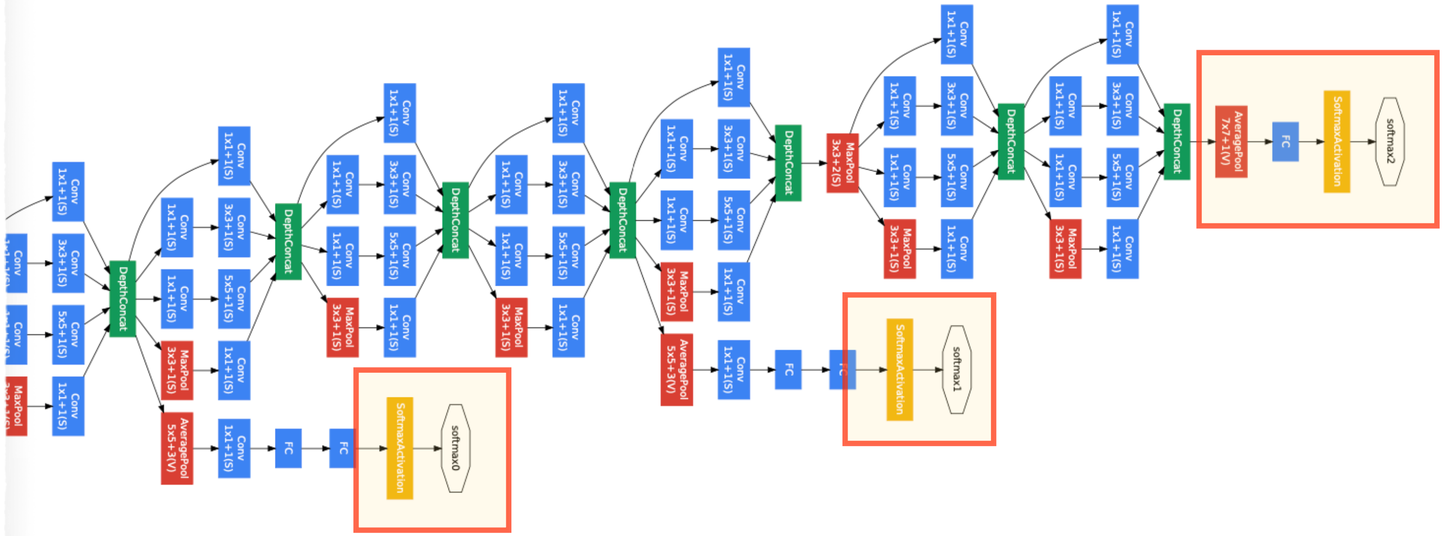
22 layers! Now the era of deep learning begins!

Inception Architecture





7 x 7 x 1024



3 classifiers + Global Average Pooling (GAP)

Pop Quiz

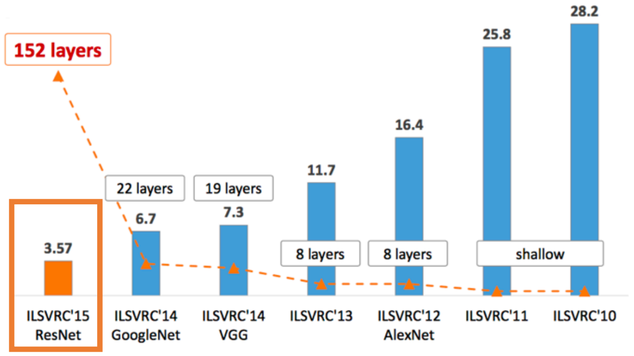
- Which Convolutional Network has **less** parameters?
    1. AlexNet (8-layers)
    2. VGGNet (19-layers)
    3. GoogLeNet (22-layers)

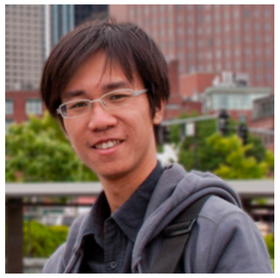
Pop Quiz

- Which Convolutional Network has **less** parameters?
  1. AlexNet (8-layers) → 60M
  2. VGGNet (19-layers) → 138M
  3. **GoogLeNet (22-layers) → 4M**

more layers do not imply more parameters!

ResNet

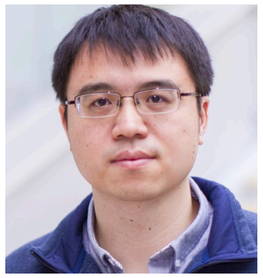




Kaiming He

Xiangyu Zhang

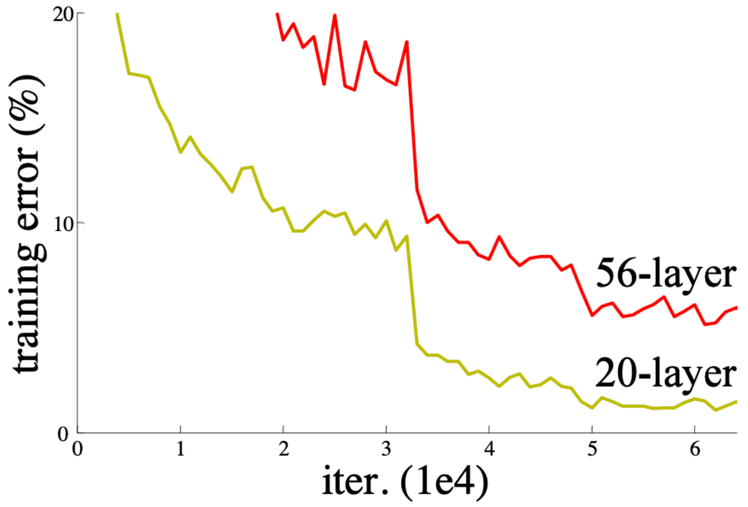
Shaoqing Ren

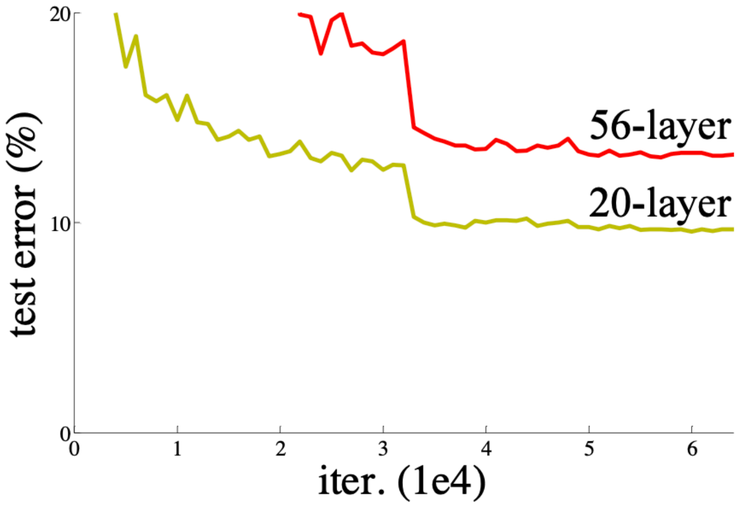


Jian Sun

- Deeper neural networks are more difficult to train
  - not caused by *overfitting* → hard to optimize

deeper network does not mean overfitting



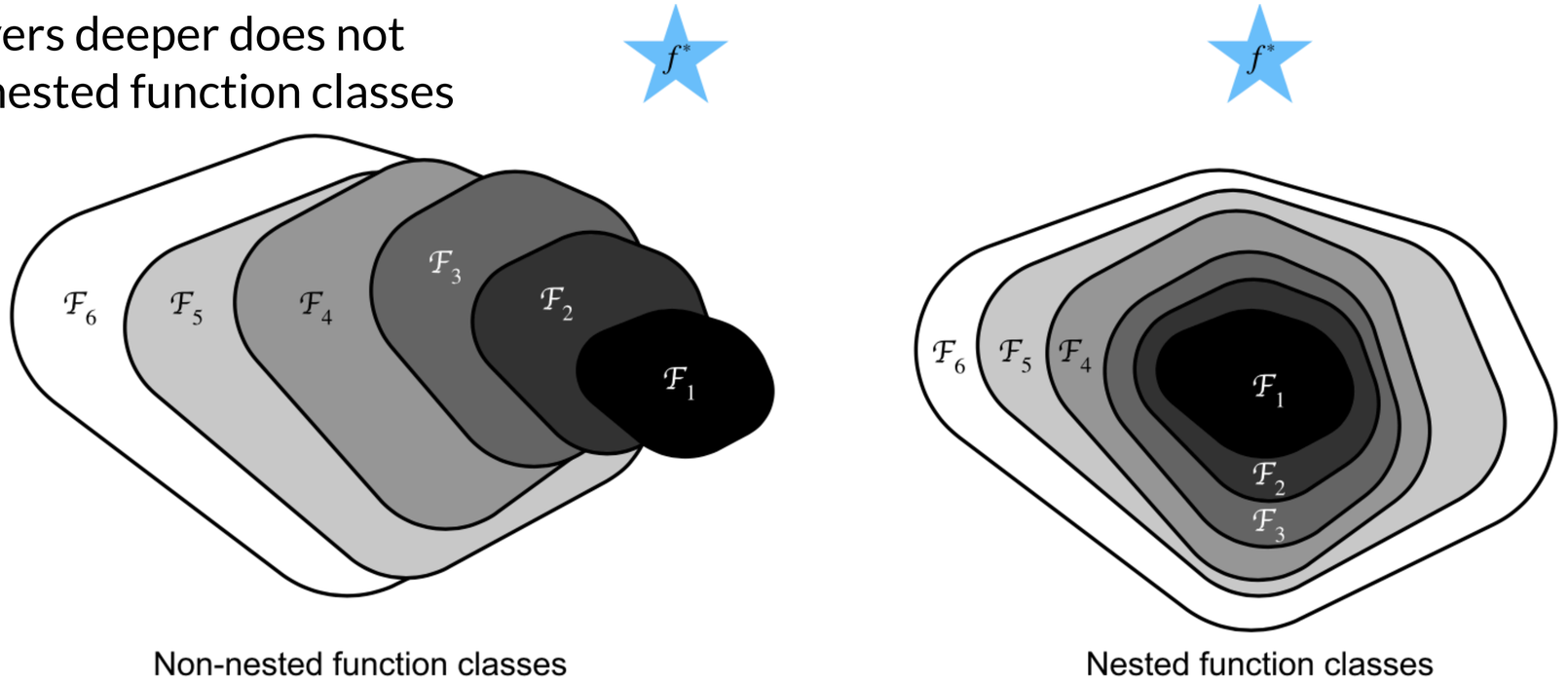


Deep Residual Learning for Image Recognition, He et al., ***CVPR*** (2016)

Towards Nested Function Classes

- If $\mathcal{F}_{\text{shallow}} \not\subset \mathcal{F}_{\text{deep}}$, we cannot expect deeper models work better
  - how to make our neural nets families to be nested?

stacking layers deeper does not guarantee nested function classes

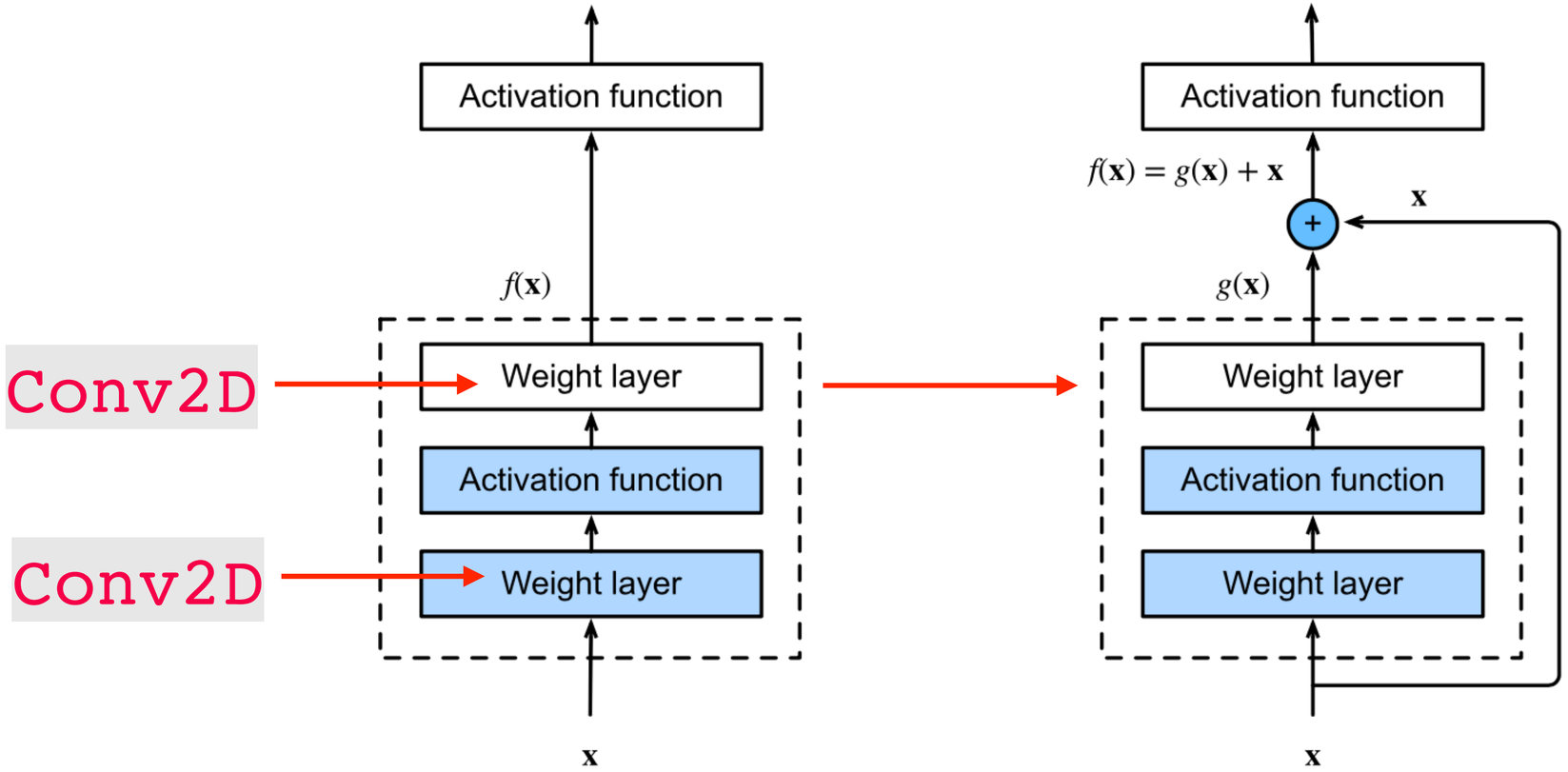


Non-nested function classes

Nested function classes

Residual Blocks

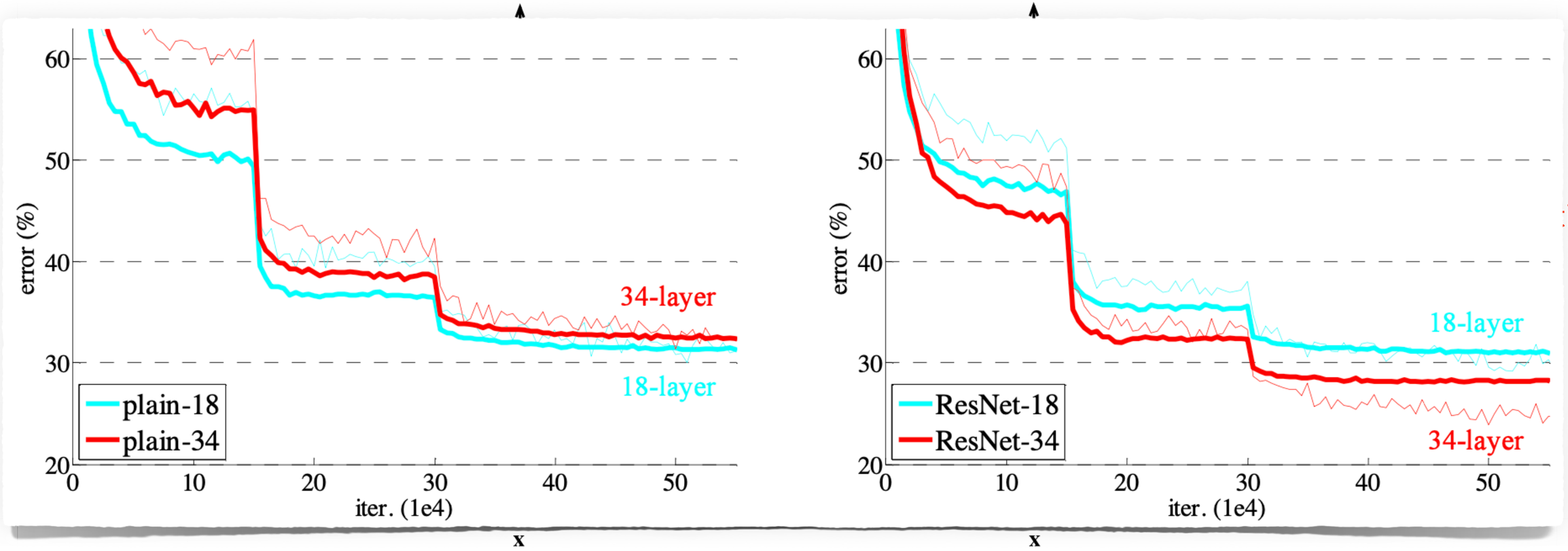
- Add identity map after nonlinear function: $f(x) \rightarrow x + f(x)$



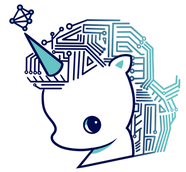
this is called
skip connection

Residual Blocks

- Add identity map after nonlinear function: $f(x) \rightarrow x + f(x)$

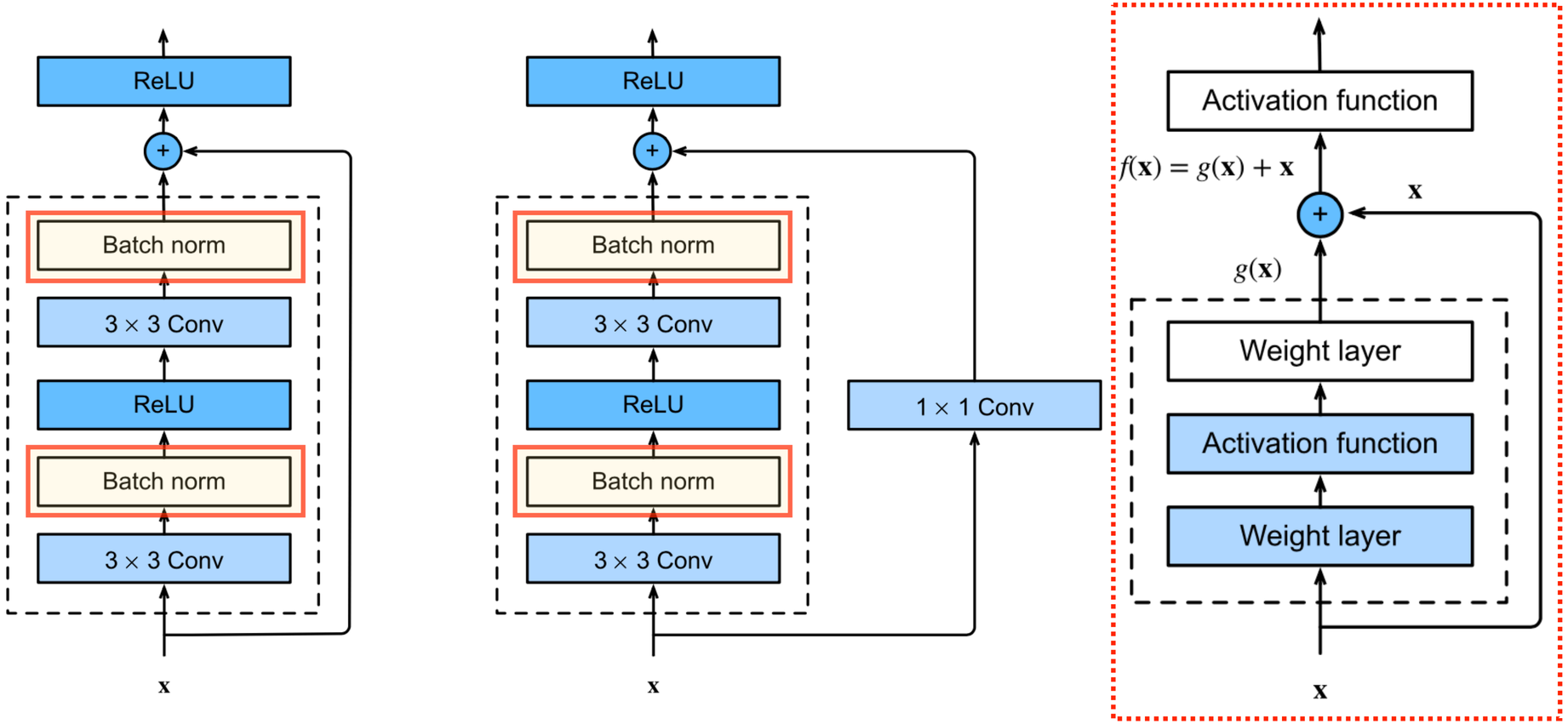


Residual Blocks



we have two types of residual blocks

- Add identity map after nonlinear function: $f(x) \rightarrow x + f(x)$



Batch Normalization

- Ioffe & Szegedy poposed BatchNorm, a popular and effective technique that accelerates the convergence of deep nets
  - they did not intend *regularization*
  - with residual blocks, BatchNorm has made it possible for engineers to train very deep networks (over 100 layers)

Tip: do not use BatchNorm when batch size is small

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}$$

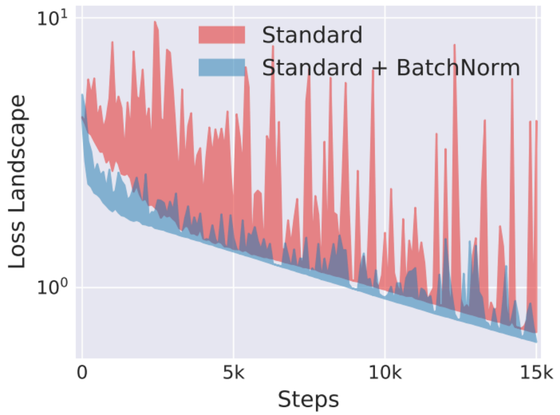
- $\hat{\boldsymbol{\mu}}_{\mathcal{B}}$: minibatch sample mean
- $\boldsymbol{\beta}$: offsets
- $\boldsymbol{\gamma}$: scaling coefficients
- $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}$: minibatch sample variance

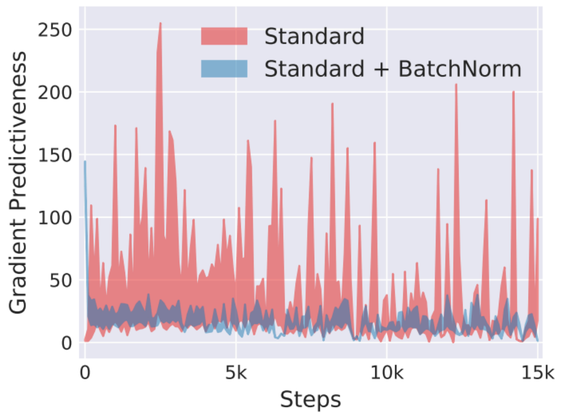
Batch Normalization, Ioffe & Szegedy, ***ICML*** (2016)

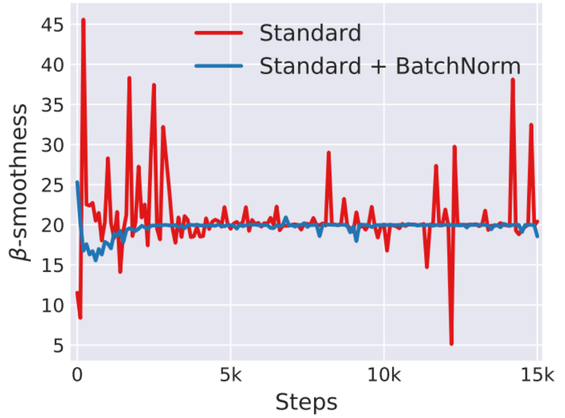
How BatchNorm works?

we don't know exactly why
BatchNorm works theoretically

- Original paper offered an explanation internal covariate shift
  - but BatchNorm rather *increases* internal covariate shift
- Santurkar et al. claimed BatchNorm (and other normalization techniques) works since it makes the loss landscape smoother → not proved!

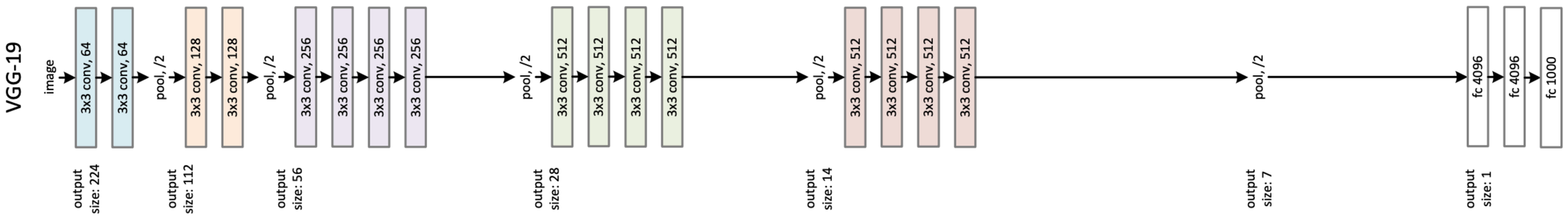




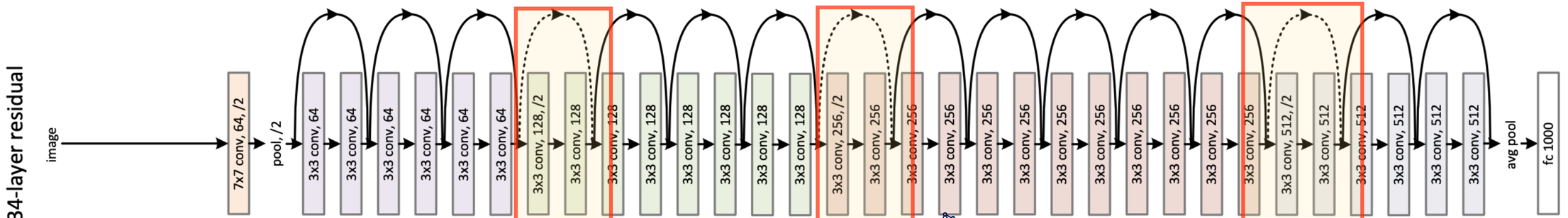


How Does Batch Normalization Help Optimization?, Santurkar et al., ***NeuIPS*** (2018)

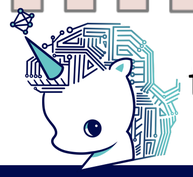
VGGNet vs ResNet



VGG-19

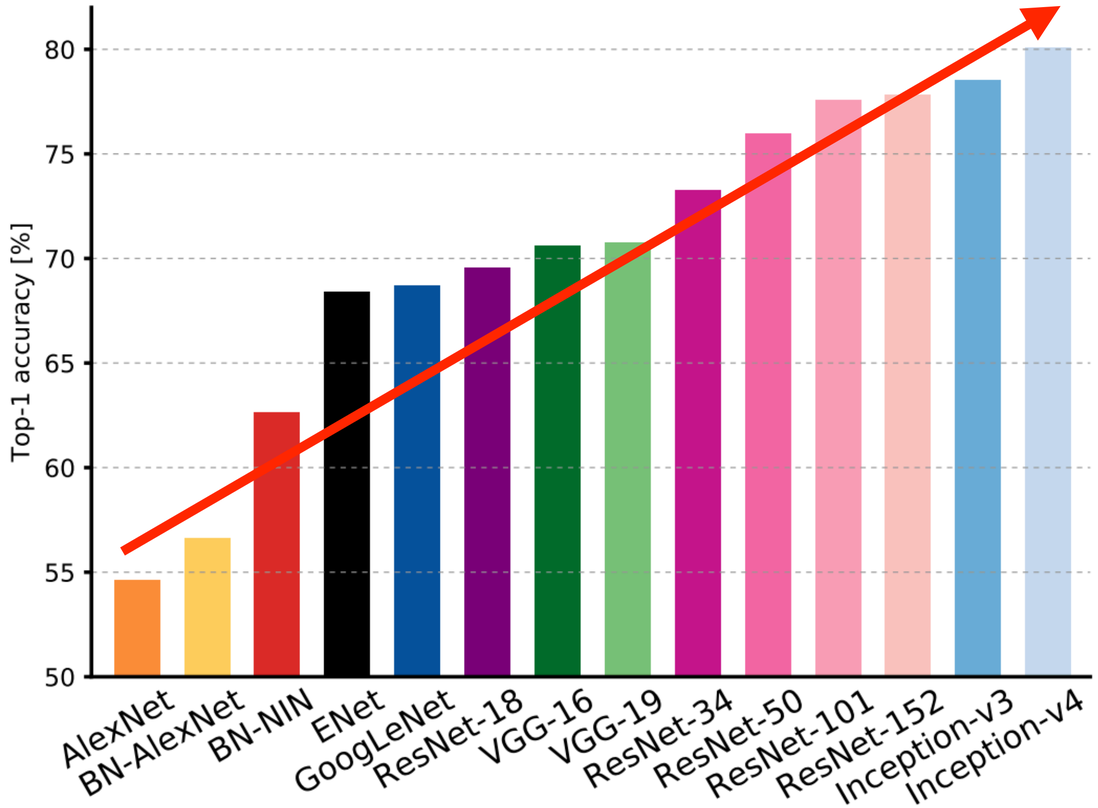


ResNet-34

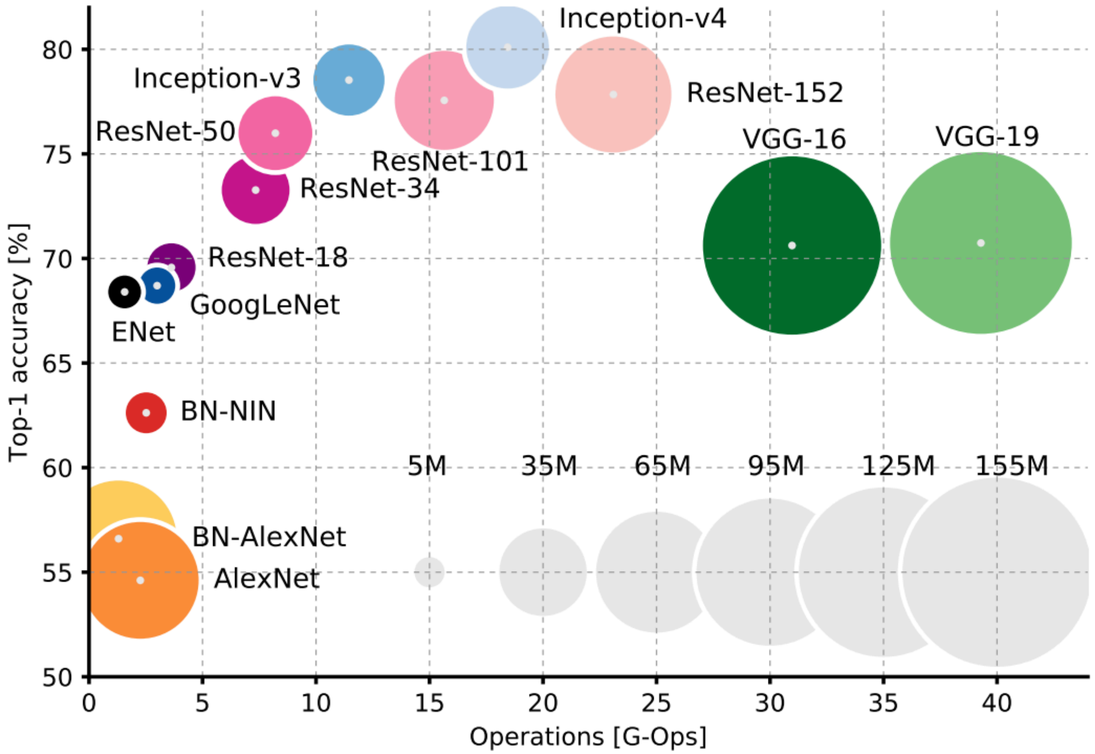


these blocks use projection shortcuts

Network Comparison



Top1 vs. Networks



Top1 vs. Parameter Sizes

An Analysis of Deep Neural Network Models for Practical Applications, Canziani et al., (2016)

Q&A