MACHINE
LEARNING
TOKYO

Deep Learning Workshop Series

CNN Architectures

Plan A

CONTENT

- ★ CNN Architectures (with interactive implementation)
  - VGG-Net: 3x3 vs 11x11 Convolution
  - Inception-Net: “1x1 convolution” vs “Fully Connected”
  - MobileNet: Depthwise (Separable) Convolutions for Training Light Models
  - ShuffleNet(not implemented)
  - SqueezeNet: Distributed Training of Networks
  - ResNet: Residuals in Convolution Operations
  - DenseNet: Dense Connections in Convolution Operations
- ★ Extras (short summary only)
  - Feature Pyramid Networks (FPNs)
  - Neural ODEs

Plan A

PROGRAM FLOW

10:00 - 10:30	10:30 - 11:10	11:10 - 12:00	12:00 - 12:45	12:45 - 13:05	13:05 - 13:50
3x3 vs 11x11	"1x1 conv" vs "FC"	Depthwise (separable) conv	BREAK	Channel shuffling	SqueezeNet
10min 15min	20min 25min	30min 20min		20min	20min 25min

13:50 - 14:20	14:20 - 14:30	14:30 - 15:00	15:00 - 15:30	15:30 - 15:45	16:00 -
Residuals & Dense Connections in ConvNets	BREAK	Implementation of Resnet	Implementation of DenseNe	Extras	Q&A
30min		30min	30min	15min	

Plan B

CONTENT

- ★ Part 1 : A Brief Intro to Convolution Operations
- ★ Part 2 : Popular CNN Architectures (interactive implementation)
  - ○ VGG-Net: 3x3 vs 11x11 Convolution
  - ○ Inception-Net: “1x1 convolution” vs “Fully Connected”
  - ○ MobileNet: Depthwise (Separable) Convolutions for Training Light Models
  - ○ SqueezeNet: Distributed Training of Networks
  - ○ ResNet: Residuals in Convolution Operations
  - ○ DenseNet: Dense Connections in Convolution Operations
- ★ Extras (short summary only)
  - ○ Feature Pyramid Networks (FPNs)
  - ○ Neural ODEs

Plan B

PROGRAM FLOW

10:00 - 10:30	10:30 - 11:00	11:00 - 12:45	12:45 - 13:30	13:30 - 14:20	14:20 - 15:00
Part 1 : A Historical Review on Deep CNNs	3x3 vs 11x11	“1x1 conv” vs “FC”	BREAK	Depthwise (separable) conv	SqueezeNet
15min 15min	10min 15min	20min 25min		30min 20min	20min 25min

15:00 - 15:10	15:10 - 15:30	15:30 - 16:00	16:00 - 16:30	16:30 -	
BREAK	Residuals & Dense Connections in ConvNets	Implementation of Resnet	Implementation of DenseNe	CLOSING	Q&A
	20min	30min	30min		

Part 1 : A Brief Intro to Convolution Operations

Computer Vision

Computer vision is a field deals with how to gain high-level understanding from images or videos

Goal :

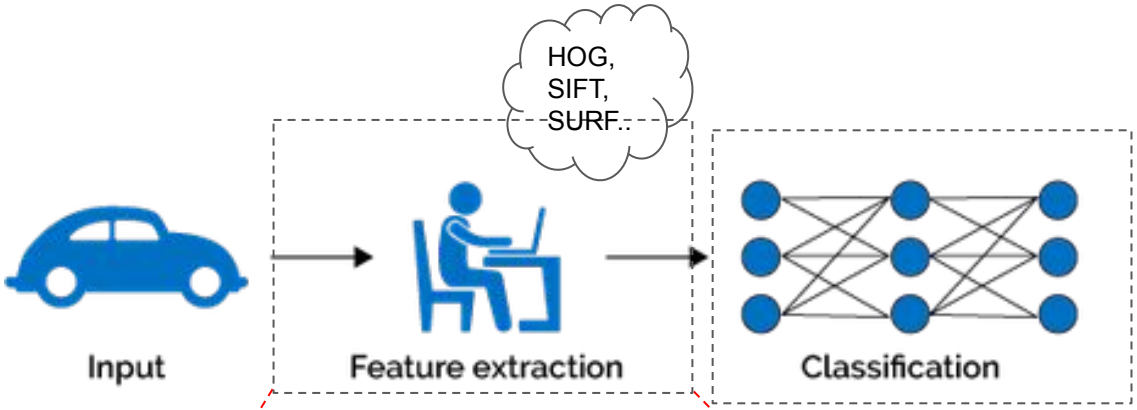
Extracting meaningful features from an image

edges, corners, colors, shapes..

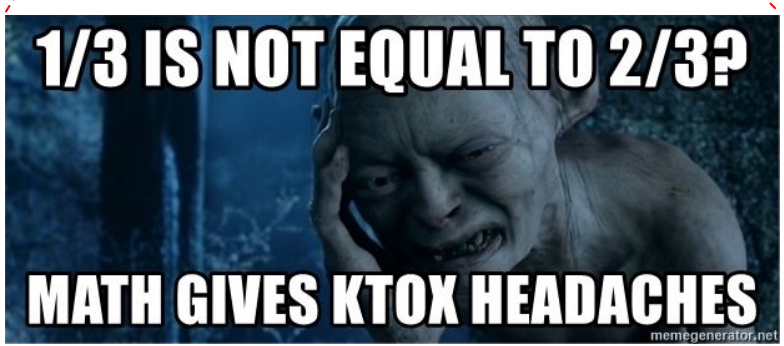
Tool:

Algorithms

- gray-scaling, thresholding,
- complex descriptors (HOG, SIFT, SURF etc.)



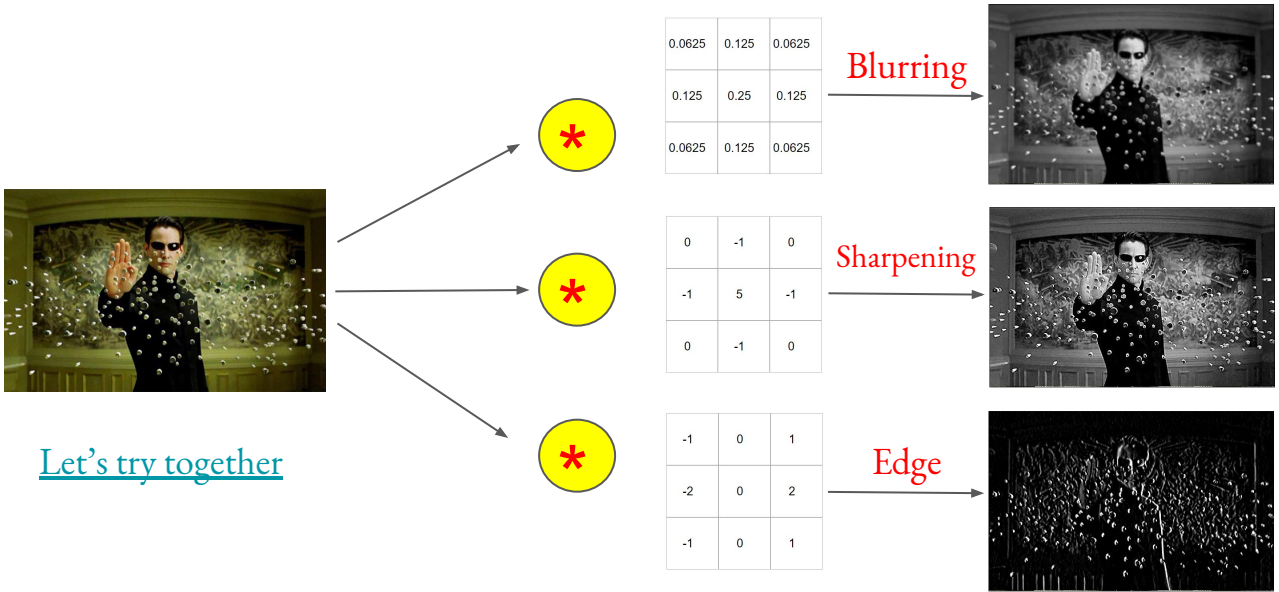
full-of hand-engineering



..well, then

..by generalizing all these techniques with applying
convolution operations along with neural nets

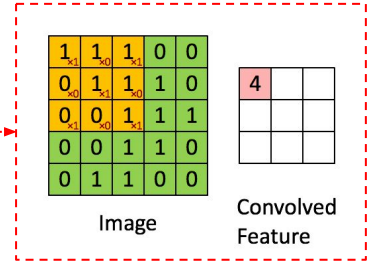
- Convolutions are filtering operations
- Different filter (kernel) sizes **unveil** different image feature information
- Commonly in two steps:
    1. **Slide** *the same fixed* kernel across the image
    2. Calculate the **dot product** between kernel and image



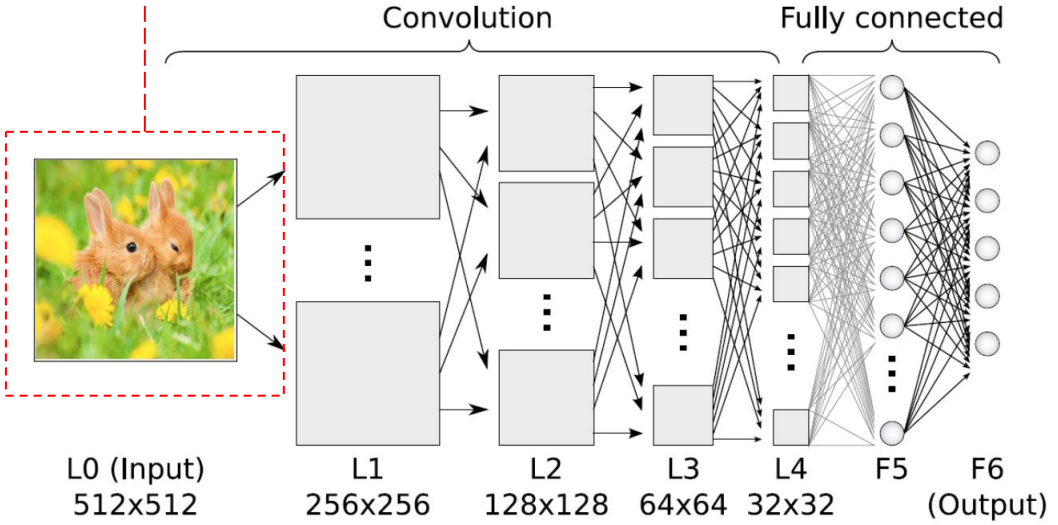
Let's try together

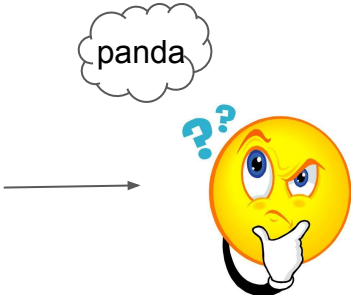
Extracting Image Features via ConvOps

Extracting useful information from image



Sliding windows (kernels or filters) are used to convolve an input image





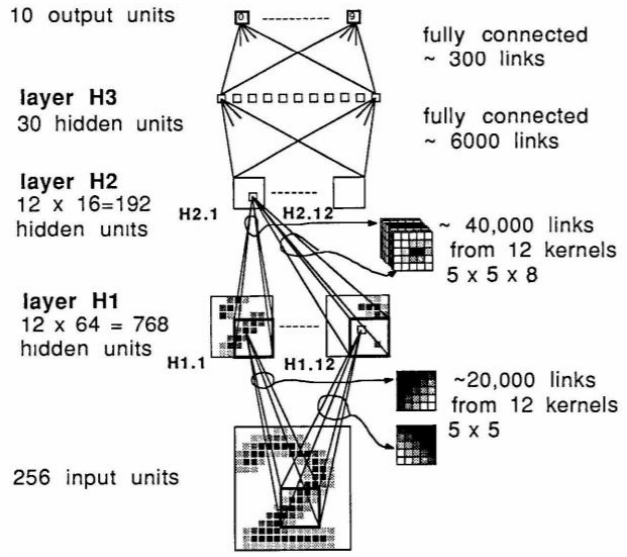
A brief history...

Convolution operations are first introduced into Machine Learning by Yann LeCun at AT&T Laboratories (Y. LeCun et. al. 1989, Fukushima 1980, A.Weibel 1987)

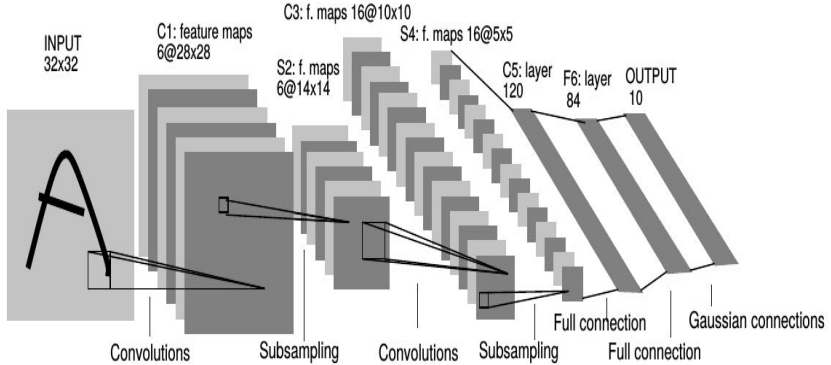
Backpropagation first applied

Improved by Gradient-descent

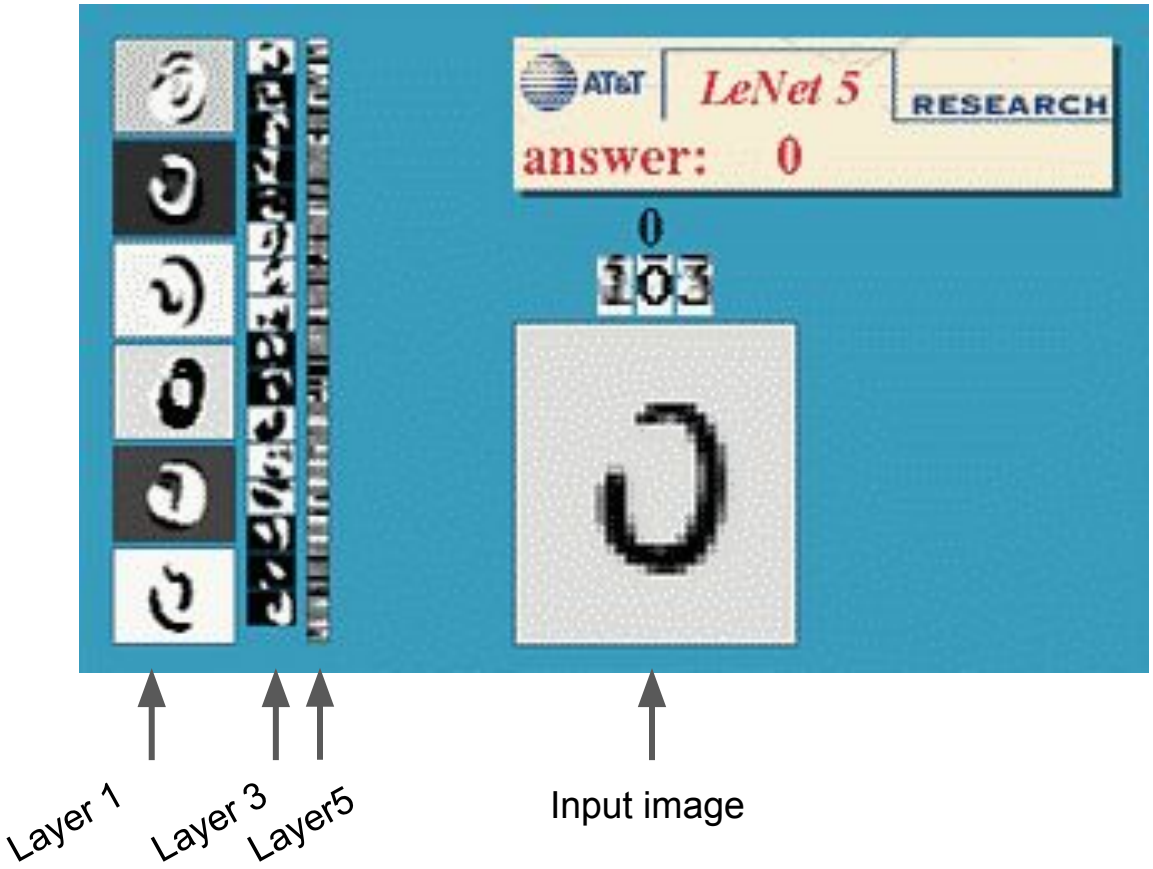
LeCun et al 1989 -



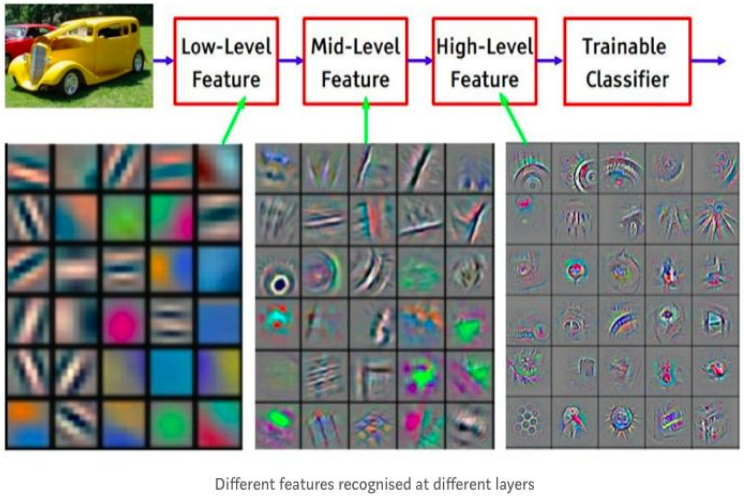
LeNet Architecture (1998)



LeNet5 by Lecun et al 1998



Convolutions in CNN Architectures



Different features recognised at different layers

features

simple complex

Building blocks of common CNN architectures:

Feature Extraction:

Conv

- Convolution operation
- Activation function

Pool

- Max (mean, global, etc) pooling

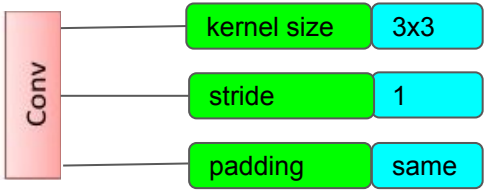
Classification/Regression:

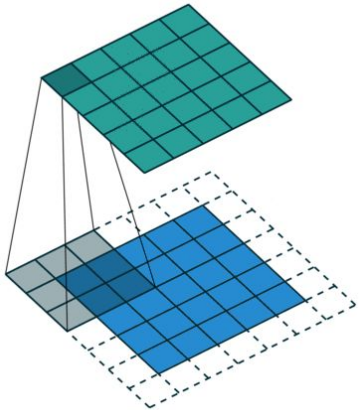
FC

Conv

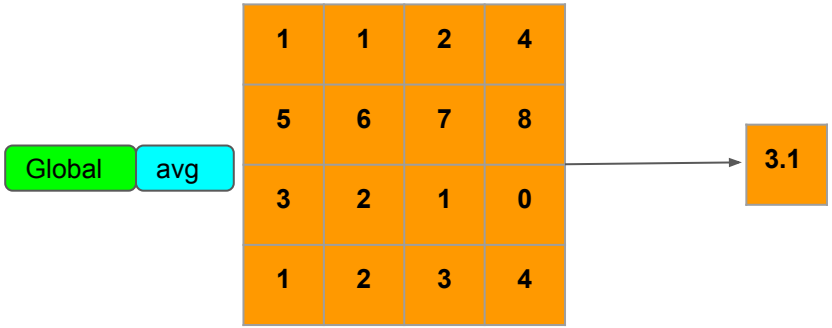
- kernel size: spatial dimensions of filter, usually odd numbers
- stride: controls how the filter convolves around input tensor
- Padding: controls the spatial dimensions of output tensor

Convolution Layer Hyperparameters





[Convolution Visualizer]



REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, **Backpropagation Applied to Handwritten Zip Code Recognition**; AT&T Bell Laboratories

[2] LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "**Gradient-based learning applied to document recognition**" (PDF). *Proceedings of the IEEE*. 86 (11): 2278–2324. doi:10.1109/5.726791. Retrieved October 7, 2016.

[3] **The History of Neural Networks**
by Eugenio Culurciello
https://dataconomy.com/2017/04/history-neural-networks/

[4] **Convolutions by AI Shack**
Utkarsh Sinha
http://aishack.in/tutorials/image-convolution-examples/

[5] **The History of Neural Networks**
Andrew Fogg
https://www.import.io/post/history-of-deep-learning/

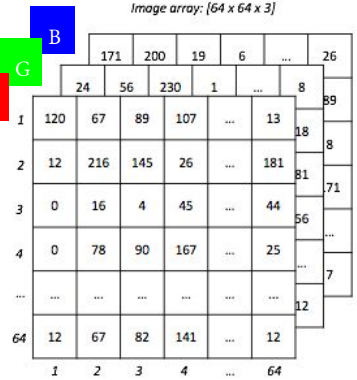
[6] **Overview of Convolutional Neural Networks for Image Classification**
Intel Academy
https://software.intel.com/en-us/articles/hands-on-ai-part-15-overview-of-convolutional-neural-networks-for-image-classification

[7] **Convolution Arithmetic**
https://github.com/vdumoulin/conv_arithmetic

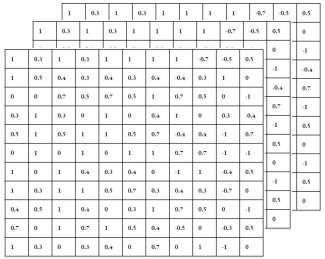
Snippet Implementation

Part 2 : Convolutions in Deep Architectures

3x3 vs 11x11



Filter size: 11x11

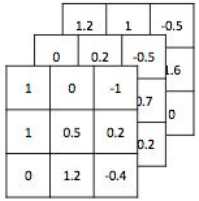


Bigger filter size, more global information captured

Output: (64 x 64)

	1	2	3	4	...	64
1	1.2	2.5	0.6	0.7	...	1.3
2	1.2	-1.7	-1.6	0.2	...	-1
3	0	-6.1	-3	4.5	...	0.4
4	0	1.1	3.2	1.7	...	-2.5
...	...	...	...	...	...	...
64	-0.8	0.7	3	1.6	...	0.2

Filter size: 3x3



Smaller filter size more local information captured

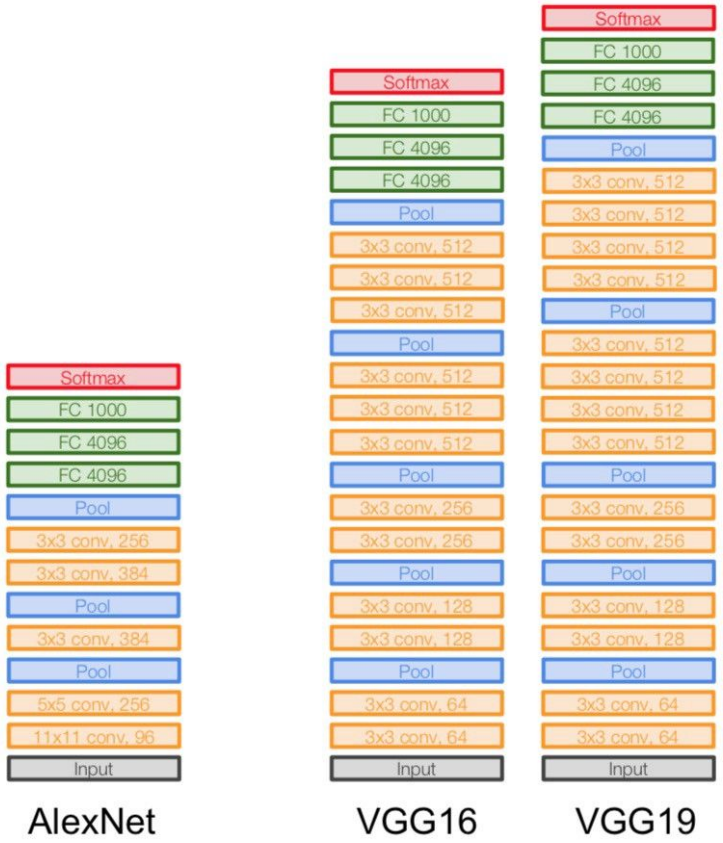
Output: (64 x 64)

	1	2	3	4	...	64
1	1.2	2.5	0.6	0.7	...	1.3
2	1.2	-1.7	-1.6	0.2	...	-1
3	0	-6.1	-3	4.5	...	0.4
4	0	1.1	3.2	1.7	...	-2.5
...	...	...	...	...	...	...
64	-0.8	0.7	3	1.6	...	0.2

AlexNet vs VGG

- Convolution filter sizes:
  - Alexnet : 11x11 , 3x3 , 5x5
  - VGGNet : 3x3

- How deep networks:
  - AlexNet : 7 Layer
  - VGGNet : 16 Layer



Efficiency & Computation

	AlexNet	VGG-16
# of Convolution Layers	5	13
Convolution Parameters	3.8M	15M
# of FC Layers	3	3
FC Layer Parameters	59M	59M
Total Params	**62M**	**138M**
ImageNet Error	**17%**	**7.3%**

REFERENCES

[1] **Different Kinds of Convolutional Filters**
Soham Chatterjee
https://www.saama.com/blog/different-kinds-convolutional-filters/

[2] **Image recognition by Deep Learning on mobile**
https://qiita.com/negi111111/items/c46635b5d70058ebae93

[3] **Paper Explanation : VGGNet**
Mohit Jain
https://mohitjain.me/2018/06/07/vggnet/

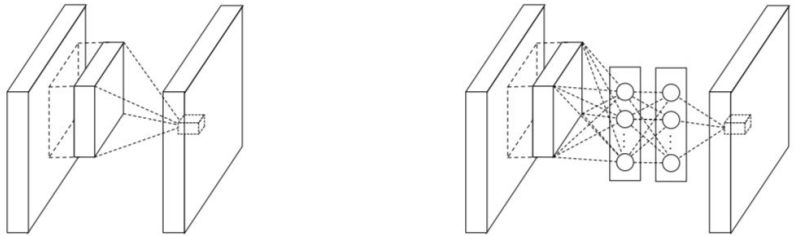
[4] **CNN Architectures - VGGNet**
Gary Chang
https://medium.com/deep-learning-g/cnn-architectures-vggnet-e09d7fe79c45

Interactive Implementation

1x1 (pointwise) conv

Proposed in “Network-in-network” by Min.Lin et.al (2013)

- Micro network - mlpconv
  - uses multilayer perceptron, FC layers

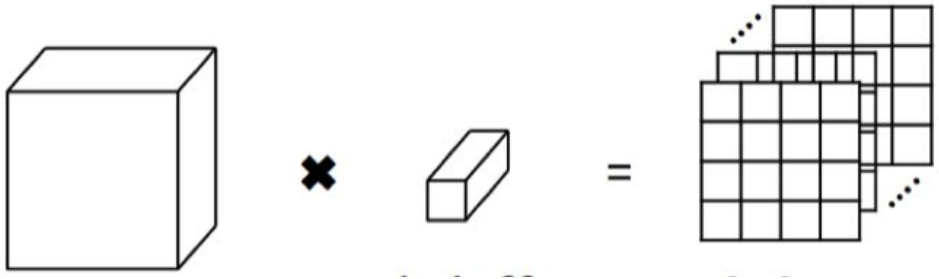


(a) Linear convolution layer (b) Mlpconv layer

- Better discriminate the model for local patches
- Cross channel pooling
  - Weighted linear combination of feature maps → ReLU
  - Complex and learnable interactions of cross channel information

1x1 (pointwise) conv

- adds non-linearity
- feature pooling: shrinks the # of channels
- doesn't care about spatial information
- can replace fully-connected layers



6x6x32 1x1x32x16 6x6x16

❑ Decreases the computations (NxN conv → 1x1 conv)

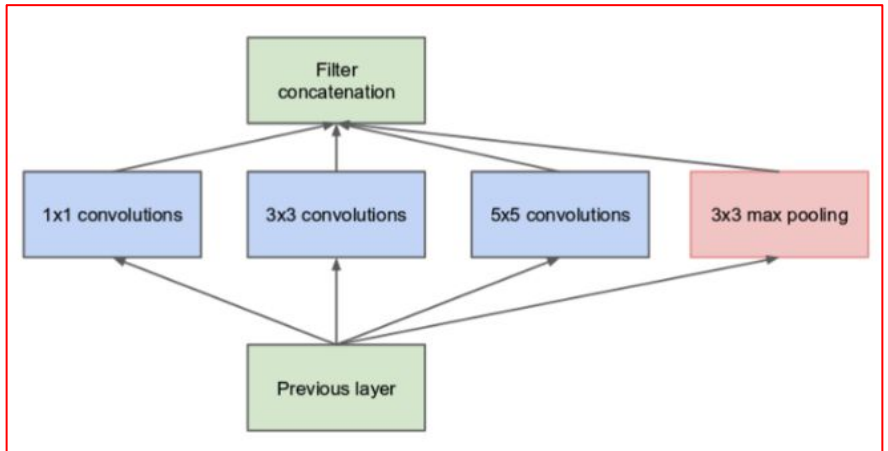
❑ Decreases the parameters (FC → 1x1 conv)

❑ Get more valuable combination of filters: represent "M" features with "N" features

Inception Module

- CNN design has a lot of parameters;
  - Conv:
    - 3x3?
    - 5x5?
    - 1x1?
  - Pooling:
    - 3x3?

- Do them all (at once)!!!
- Let the network learn;
  - Whatever parameter,
  - Whatever the combination of these filter sizes it wants to learn
- Inception Layer



Inception → GoogLeNet

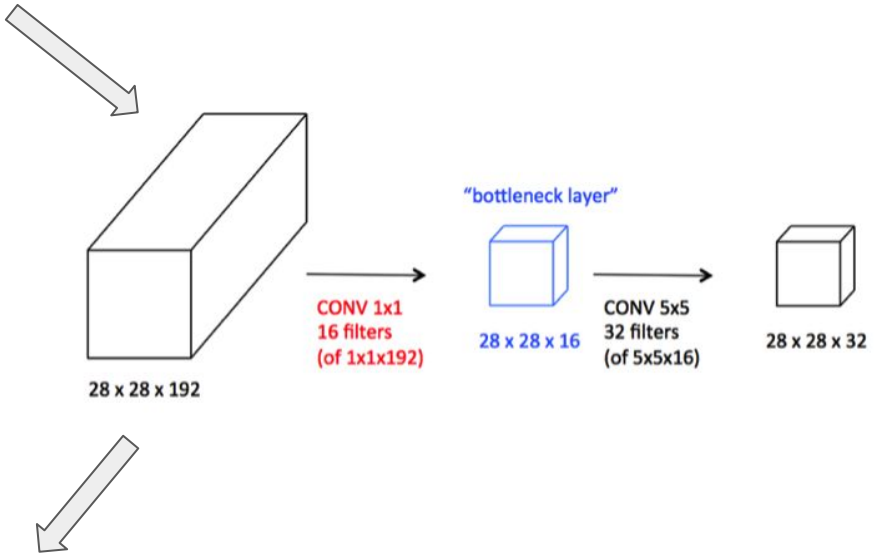
Computation (convolution) per inception layer

- 5x5 conv ⟹ (28x28)x(5x5x192)x(32) ≈ 120M
- 3x3 conv ⟹ (28x28)x(3x3x192)x(128) ≈ 170M
- 1x1 conv ⟹ (28x28)x(1x1x192)x(64) ≈ 10M
- In total; ≈ 300M computations

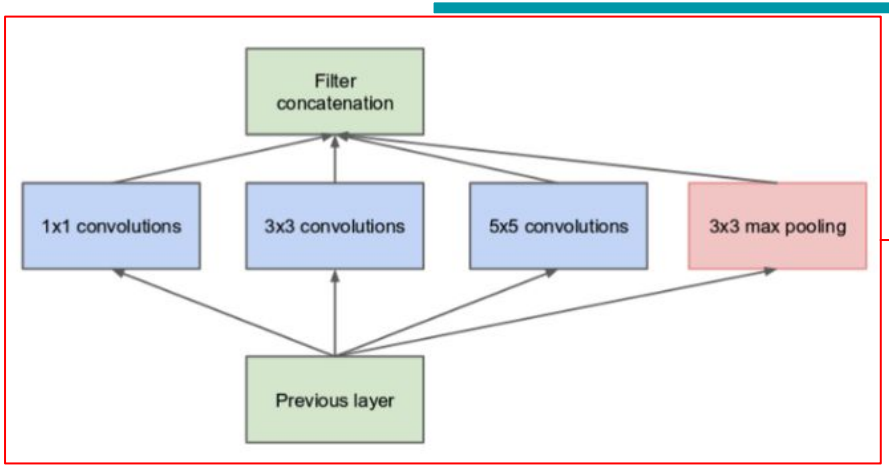
Input tensor (spatial) One Kernel Output tensor

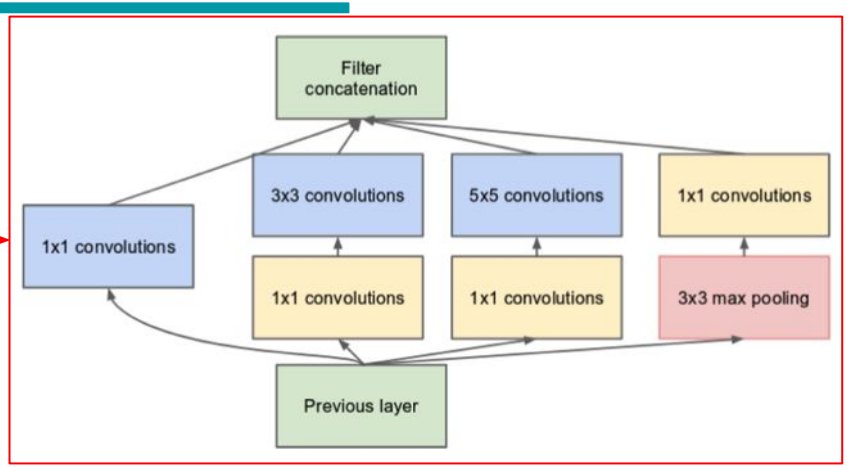
"Bottleneck layer":

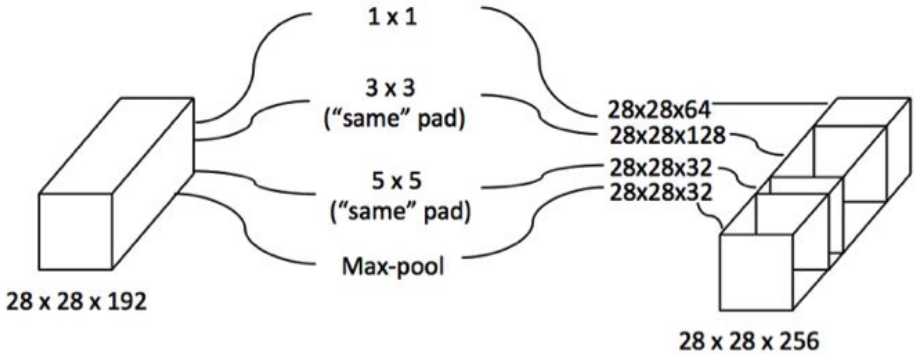
- 1x1 conv ⟹ 5x5 conv
  - (28x28) x (1x1x192)x(32) ≈ 2.4M
  - +
  - (28x28) x (5x5x16)x(32) ≈ 10M
  - ≈ 12.4M
- 10x less computation!



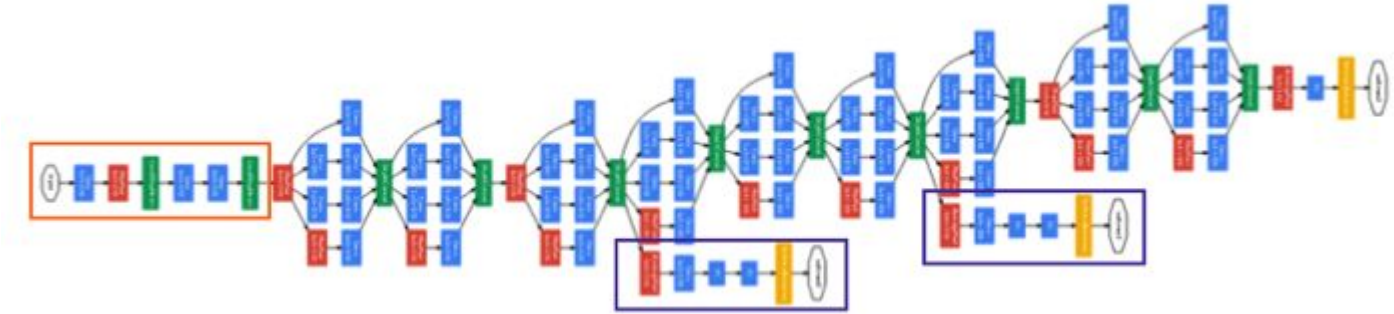
Inception → GoogLeNet







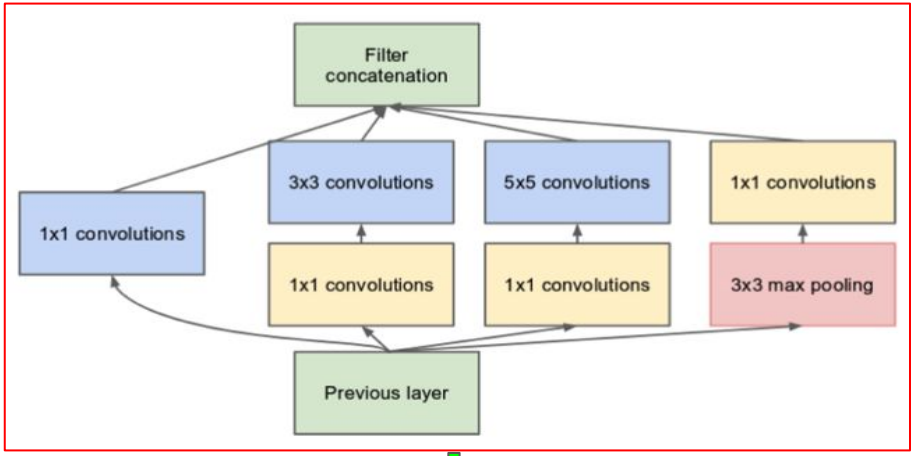
GoogLeNet (Inception v1)

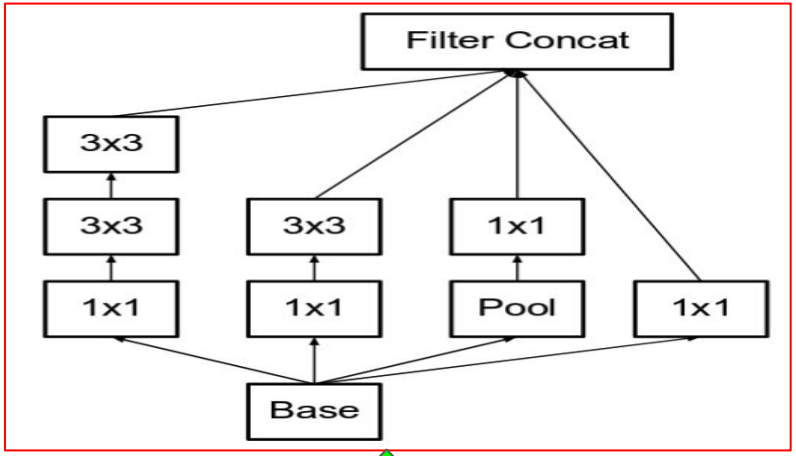


Vanishing gradient problem?

- Add two auxiliary classifiers
  - Prevent from “dying out” of middle part of network
  - Regularization effect: prevent from overfitting
- Total loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2

Inception v2





The premise:

- Reduce “representational bottleneck” - loss of information by reducing the dimensions too much
- Smart factorization methods

The solution:

- Factorize 5x5 conv to two 3x3 conv.
  - To improve computational speed
  - 5x5 conv is 2.78x more expensive than a 3x3 conv.

REFERENCES

[1] Network In Network
Min Lin, Qiang Chen, Shuicheng Yan
https://arxiv.org/pdf/1312.4400v3.pdf

[2] Network in Networks and 1x1 Convolutions
Andrew Ng
https://www.coursera.org/lecture/convolutional-neural-networks/networks-in-networks-and-1x1-convolutions-ZTb8x

[3] One by One [1x1] Convolution - counter-intuitively useful
Aaditya Prakash
https://iamaaditya.github.io/2016/03/one-by-one-convolution/

[4] Deep Learning series: Convolutional Neural Networks
Mike Cavaioni
https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

[5] Going Deeper with Convolutions
Christian Szegedy et.al.
http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

[6] A Simple Guide to the Versions of the Inception Network
Bharath Raj
https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Interactive Implementation

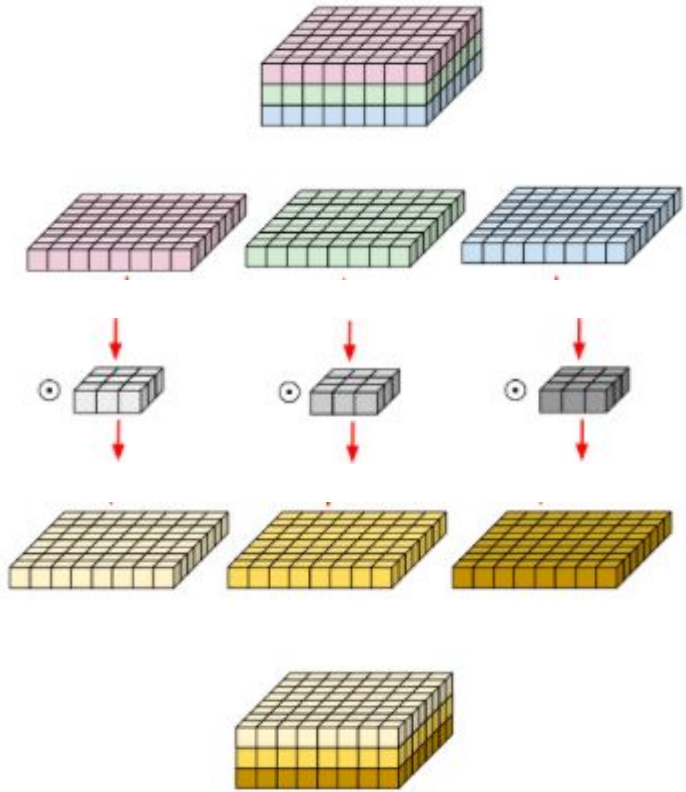
Depthwise convolution

Difference:

- So far;
  - 2D convolutions performed over all input channels
  - Lets us to mix channels
- Depthwise convolution;
  - Each channel kept separate

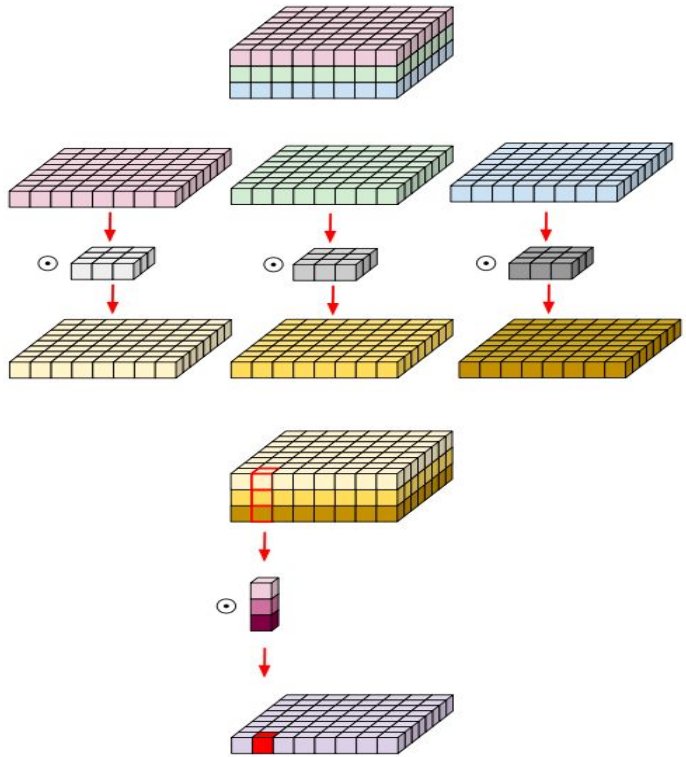
Approach:

- Split the input tensor into channels & split the kernel into channels
- For each channels, convolve the input with the corresponding filter → 2D tensor
- Stack the output (2D) tensors back together



Depthwise separable conv

- Depthwise convolution is commonly used in combination with an additional step → depthwise separation convolution
  - 1. Filtering
  - 2. Combining

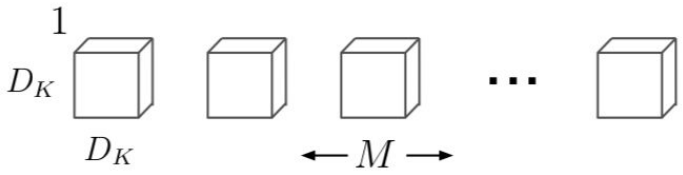


Input tensor (spatial) One Kernel Output tensor

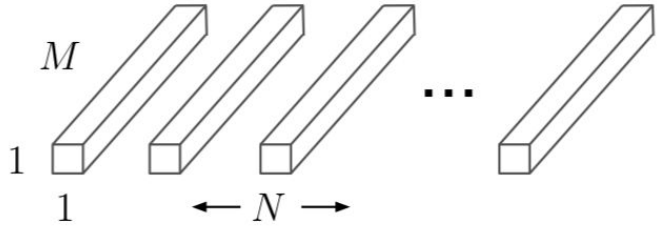
Depthwise separation convolution:

- Depthwise convolution → 1x1 convolution across channels
- Much less operations
  - Input: 8x8x3, output: 8x8x256
  - Original conv:
    - (8x8) x (5x5x3) x (256) → 1,228,800
  - Depthwise separable conv:
    - (8x8) x (5x5x1) x (3) → 3800
    - (8x8) x (1x1x3) x (256) → 49,152
    - Total: 53,952
  - 1,228,800 / 53,952 ≈ 23x less multiplication

MobileNet



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

- Depthwise separable convolution
- Shrinking hyperparameters:
  - Width multiplier (α): adjusts the channel numbers
  - Resolution multiplier (ρ): adjusts input image and feature map spatial dimensions

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

REFERENCES

[1] Depthwise separable convolutions for machine learning
Eli Bendersky
https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/

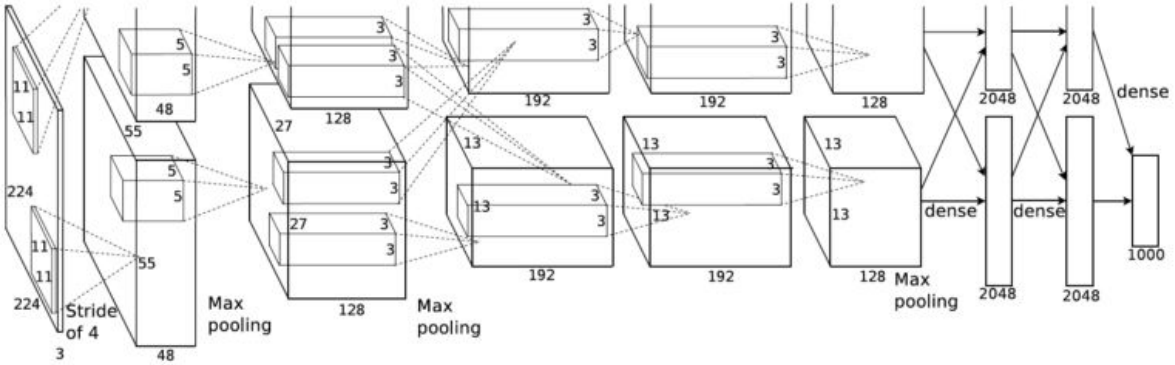
[2] MobileNets: **Efficient Convolutional Neural Networks for Mobile Vision Applications**
Andrew G.Howard, et. al. (Google Inc.)
https://arxiv.org/pdf/1704.04861.pdf

[3] Xception: **Deep Learning with Depthwise Separable Convolutions**
Francois Chollet
https://arxiv.org/pdf/1610.02357.pdf

[4] A Basic Introduction to Separable Convolutions
Chi-Feng Wang
https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

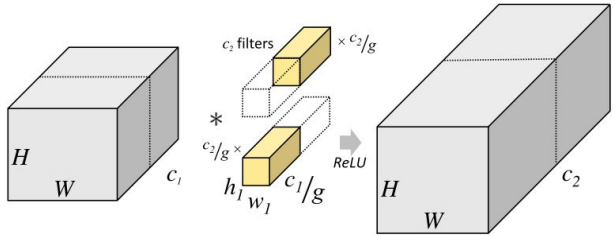
Interactive Implementation

Group Convolutions



Grouped Convolutions:

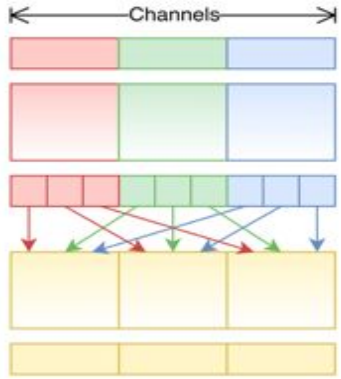
- Proposed first in AlexNet
  - Memory constraints
- Decreases number of operations
  - 2 groups → 2x less operation
- (+) Learns better representations
  - Feature relationships are sparse
- (-) outputs from a certain channel are only derived from a small fraction of input channels



Channel shuffling

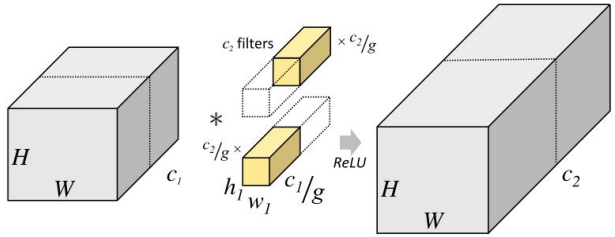
Depthwise separable convolution:

- Eliminate main side effect of grouped convolutions:
  - Outputs from a certain channel are only derived from a small fraction of input channels
- Solution:
  - Conv1_output → channel shuffling → conv2_input
- Applies group convolutions on 1x1 layer also
- (!) channel shuffling is also differentiable

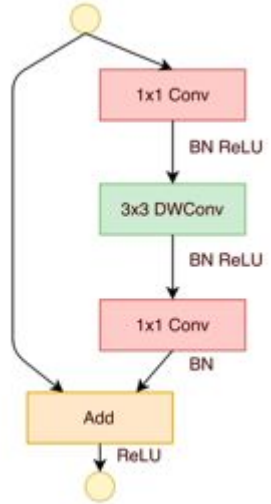


Grouped Convolutions:

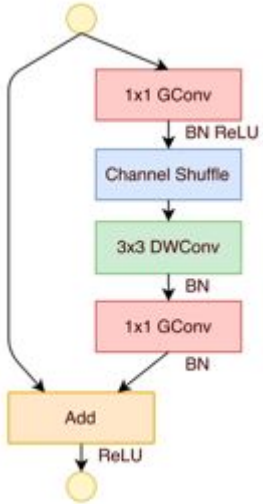
- First proposed in AlexNet
  - Memory constraints
- Decreases number of operations
  - 2 groups → 2x less operation
- (+) Learns better representations
  - Feature relationships are sparse
- (-) outputs from a certain channel are only derived from a small fraction of input channels



ShuffleNet



Bottleneck unit with depthwise convolution



ShuffleNet unit with pointwise group convolution

Channel shuffling:

- Applies group convolutions on 1x1 layer also
  - By grouping filters, computation decreases significantly
- (!) remind the side effect of grouped convolutions
  - Channel shuffling addresses this issue
- (!) channel shuffling is also differentiable

REFERENCES

[1] Convolutions Type
Illarion Khlestov
https://ikhlestov.github.io/pages/machine-learning/convolutions-types/#depthwise-separable-convolutions-separable-convolutions

[2] A Tutorial on Filter Groups（Grouped Convolution）
Yani Ioannou
https://blog.yani.io/filter-group-tutorial/

[3] ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices
Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun
https://arxiv.org/abs/1707.01083

SqueezeNet

is;

- Smart and small architecture which proposes:
- → AlexNet level accuracy (on ImageNet) with
  - 50x fewer parameters
  - 500x fewer parameters after compression
- → 3 times faster
- → Fully Convolutional Network (FCN), i.e. no FC Layer

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

SqueezeNet

SqueezeNet uses multiple tricks:

- Replace 3x3 filters with 1x1 (pointwise) filters
- Uses 1x1 filters as a bottleneck layer
- Use 3x3 filters in fire module
- Late downsampling
- Network compression
- 1x1 vs. 3x3 rate analysis
- Bypass connections

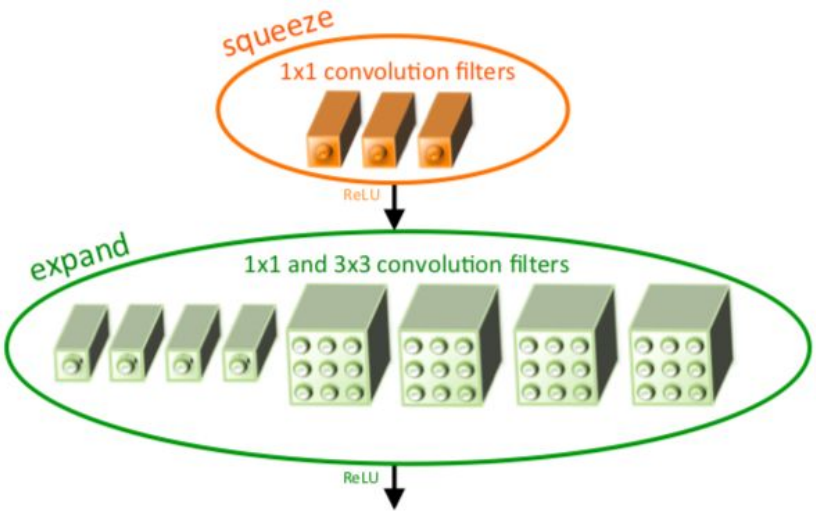
Gains OR analysis outcome:

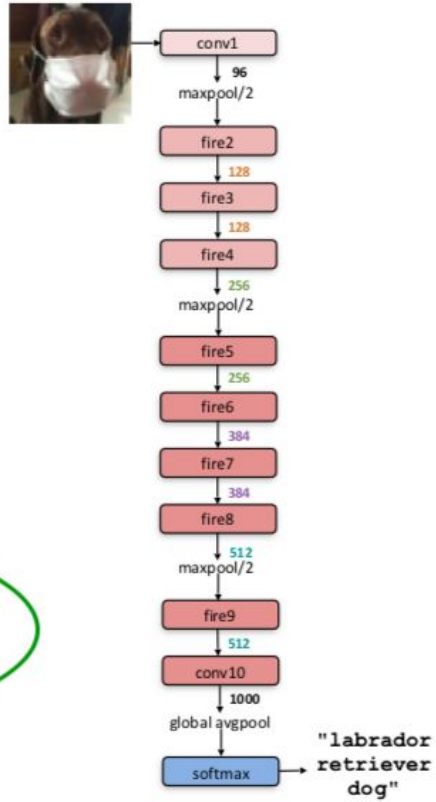
- reduce the computation by 1/9
- reduce depth → reduce
- Affects final accuracy
- Preserve feature map spatial dimensions
- Smaller networks also can be compressed
- Accuracy of trade-off
- Helps to alleviate representational bottleneck which affected by squeeze layer (in fire module)

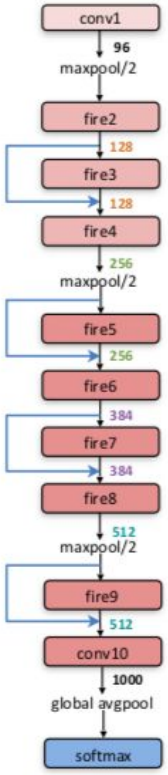
SqueezeNet

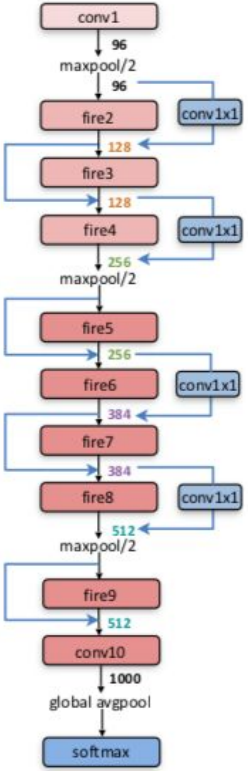
Fire module:

- Squeeze layer: only 1x1 filters (bottleneck)
- Expand layer: 1x1 and 3x3 filters









REFERENCES

[1] Notes on SqueezeNet
Hao Gao
https://medium.com/@smallfishbigsea/notes-of-squeezenet-4137d51feef4

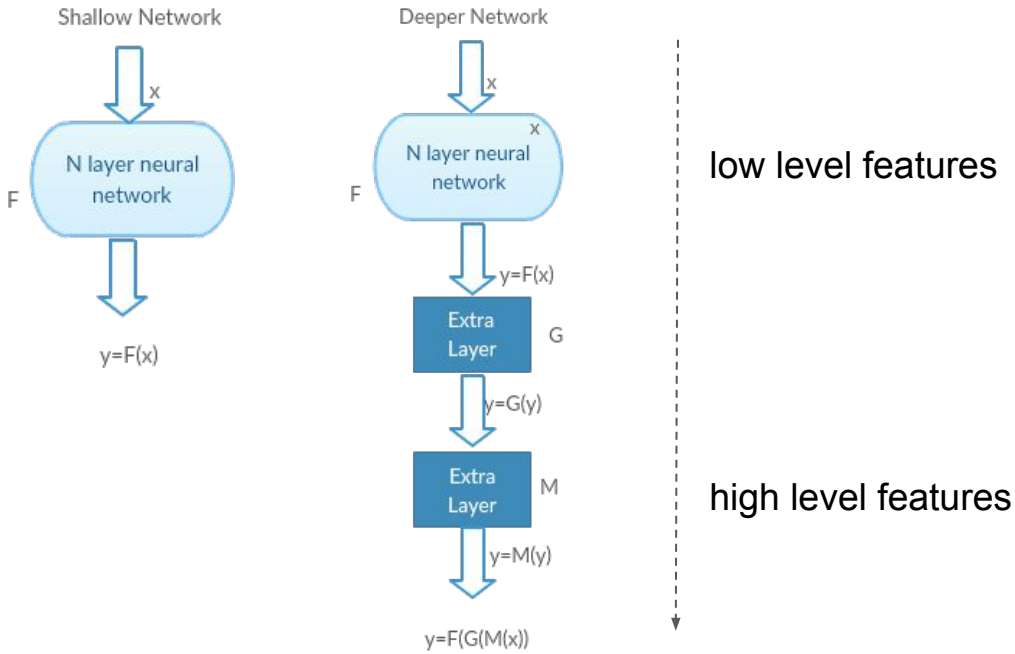
[2] Review: **SqueezeNet**（**Image Classification**）
Sik-Ho Tsang
https://towardsdatascience.com/review-squeezenet-image-classification-e7414825581a

[3] SqueezeNet: **AlexNet-level accuracy with** 50**x fewer parameters and** <0.5**MB model size**
Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer
https://arxiv.org/abs/1602.07360

Interactive Implementation

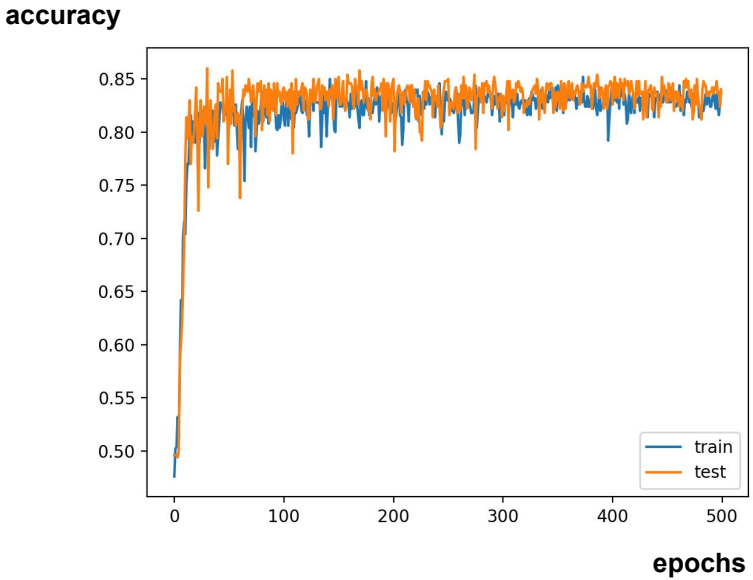
Residuals in ConvNets

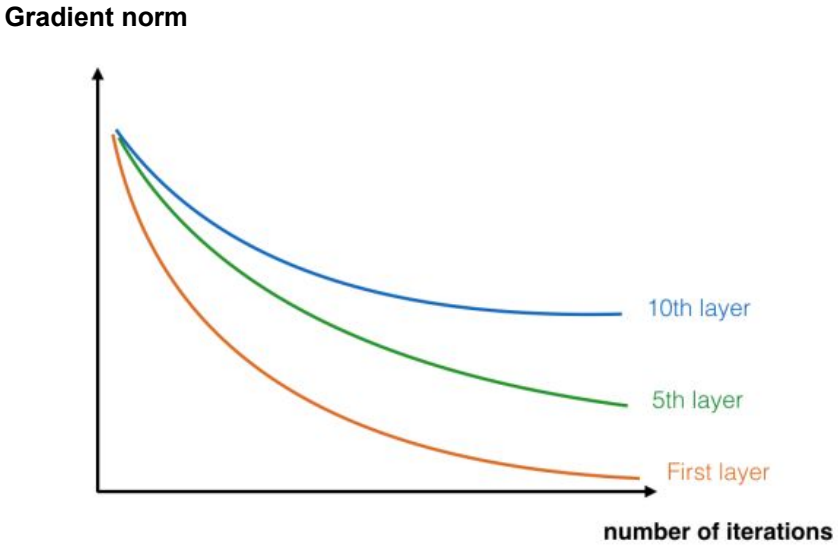
Power of going deeper = Richer contextual information



is there any limitation to having more depth?

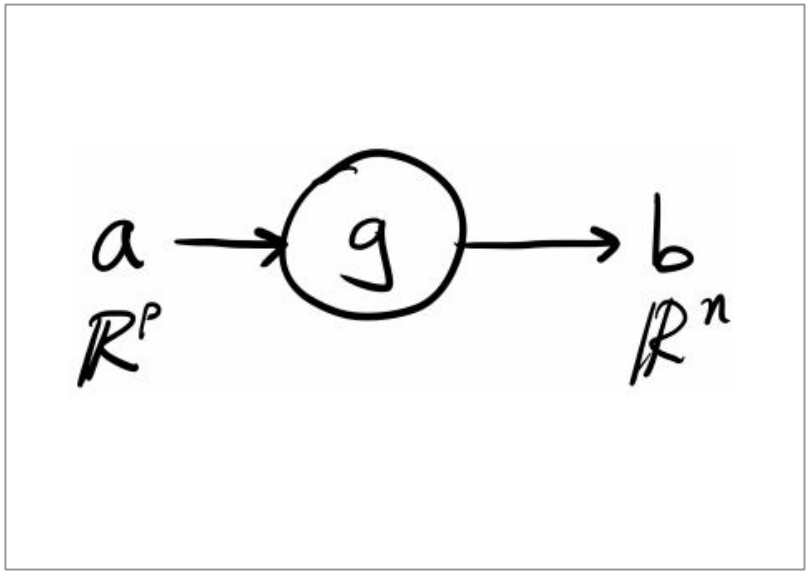
Gradients vanishing ⇨ after repeated layer operations.

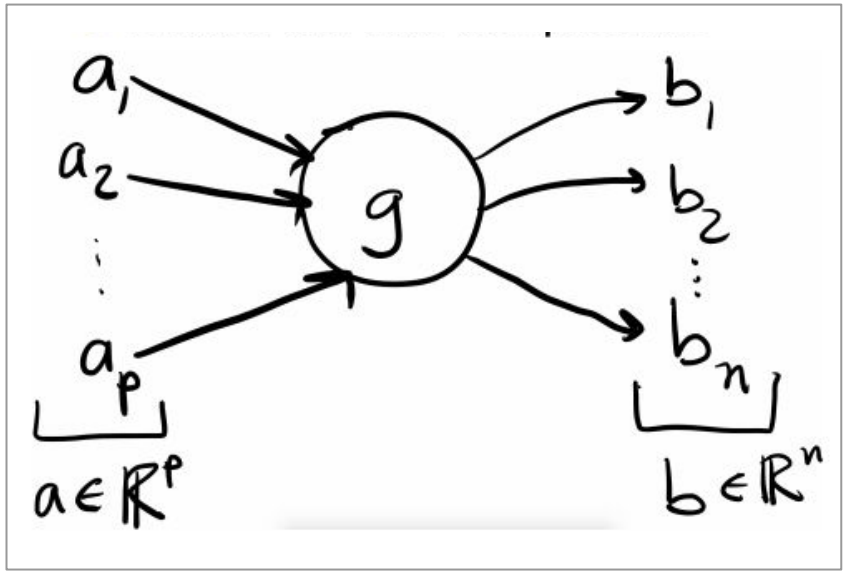




Recap : Backpropagation

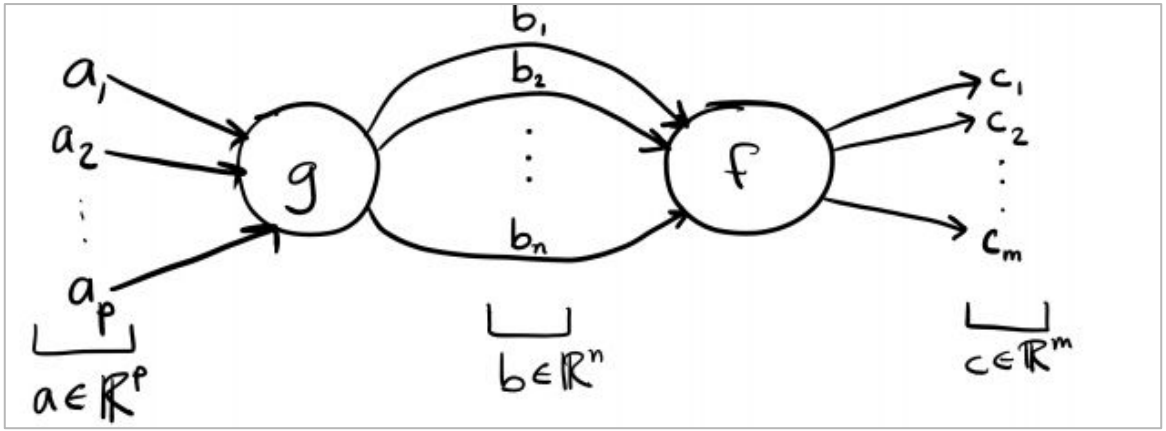
$g : \mathbf{R}^p \rightarrow \mathbf{R}^n$





Recap : Backpropagation

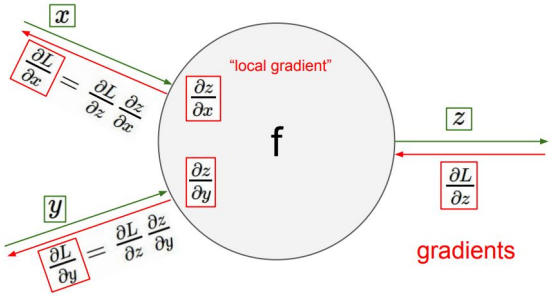
relation of outputs to inputs



$$\frac{\partial c_i}{\partial a_j} = \sum_{k=1}^{n} \frac{\partial c_i}{\partial b_k} \frac{\partial b_k}{\partial a_j}.$$

Recap : Backpropagation

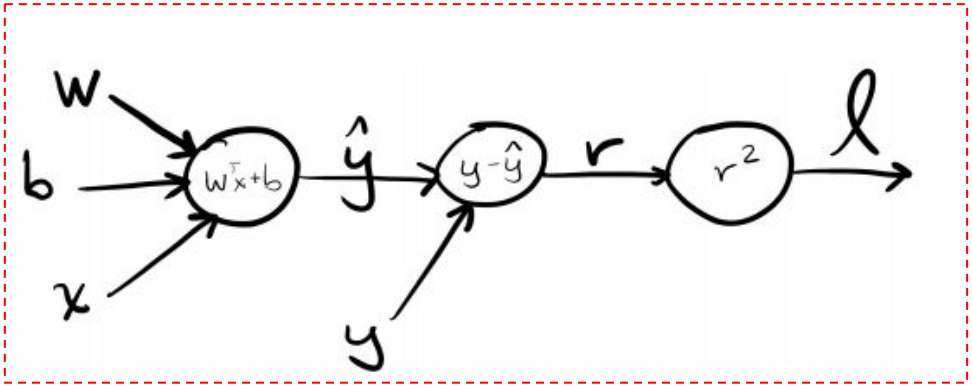
gradients of a single node..



imagine all calculations..

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial h} \frac{\partial h}{\partial z_1} \frac{\partial z_1}{\partial x}$$

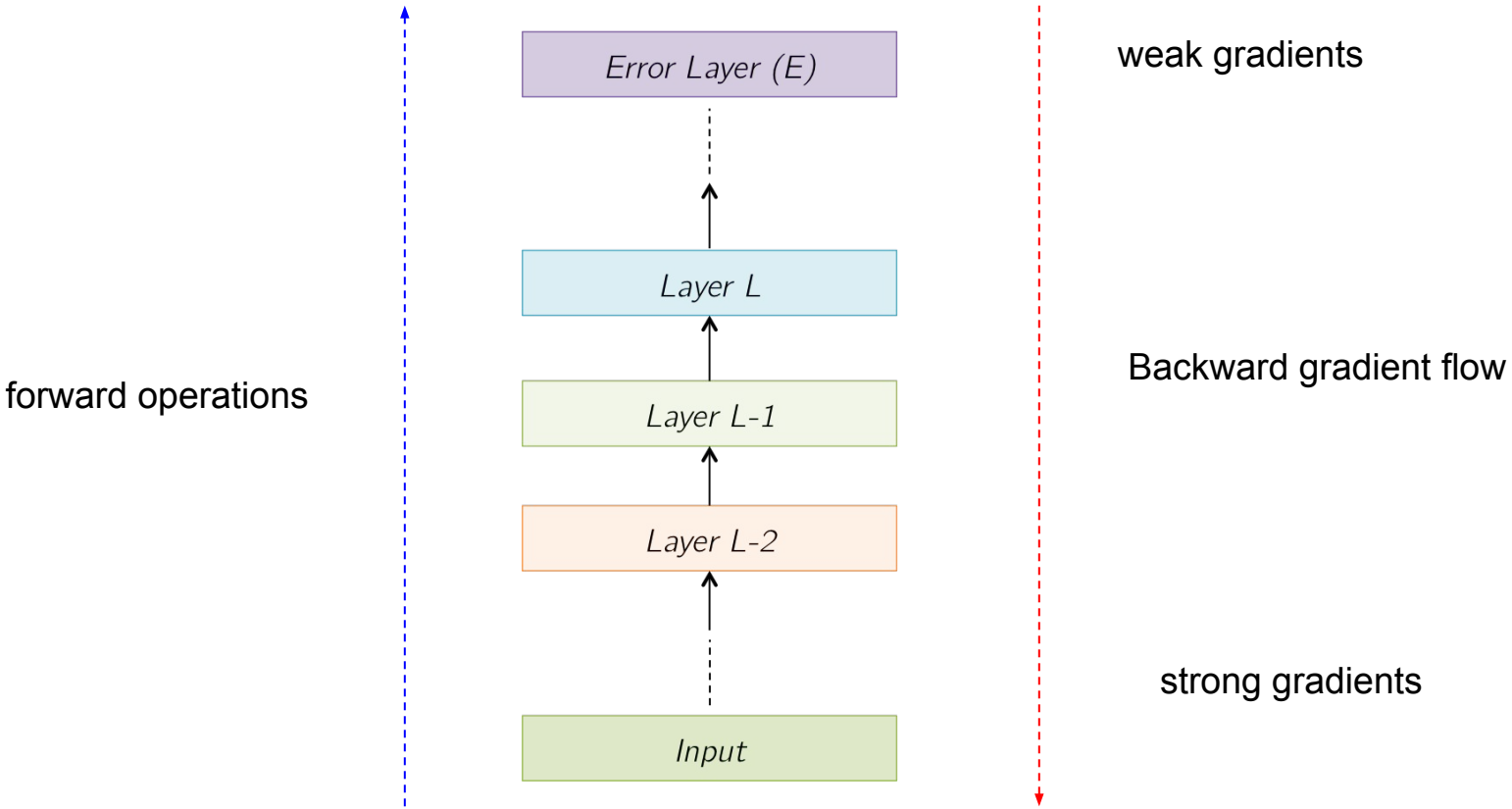
Recap : Backpropagation



$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

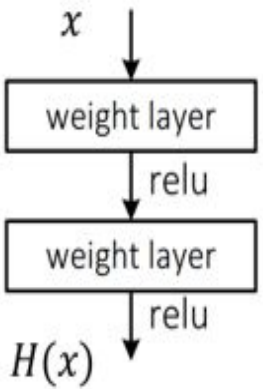
All we want a good parameter update!

Recap : Backpropagation

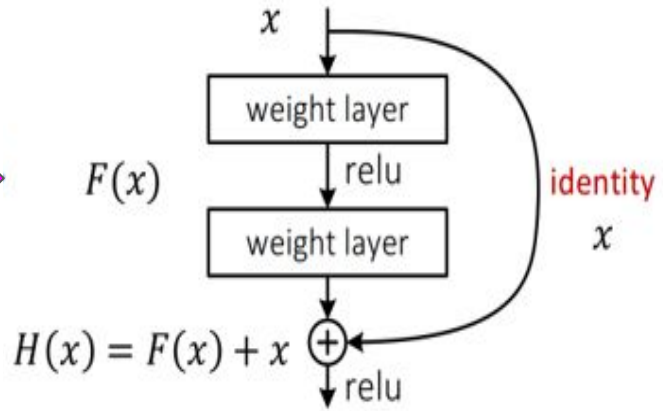


The core idea of ResNet is introducing a "identity shortcut (residual) connection"

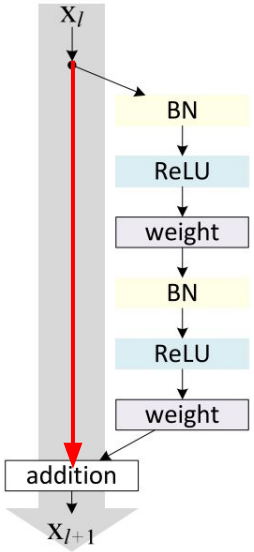
Standard connection



Skips one or more layers



Easy gradient flow via shortcuts



Plain

Input →

ResNet

Input →

ResNeXT

Input →

REFERENCES

[1] **DenResNet: Ensembling Dense Networks and Residual Networks**
Victor Cheung
http://cs231n.stanford.edu/reports/2017/pdfs/933.pdf

[2] **An Overview of ResNet and its Variants**
Vincent Fung
https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[3] **The Efficiency of Densenet**
Hao Gao
https://medium.com/@smallfishbigsea/densenet-2b0889854a92

[4] **Understanding and Implementing Architectures of ResNet and ResNeXt**
Prakash Jay
https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cc5d0adf648e

[5] **Hand-Gesture Classification using Deep Convolution and Residual Neural Network**
Sandipan Dey
https://sandipanweb.wordpress.com/2018/01/20/hand-gesture-classification-using-deep-convolution-and-residual-neural-network-with-tensorflow-keras-in-python/

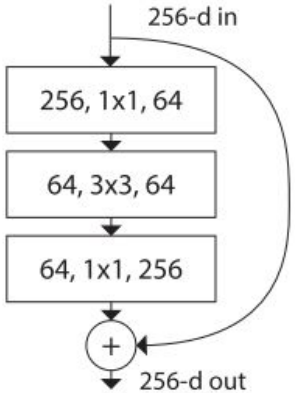
Extras:

ResNet vs ResNexT vs Inception-Resnet

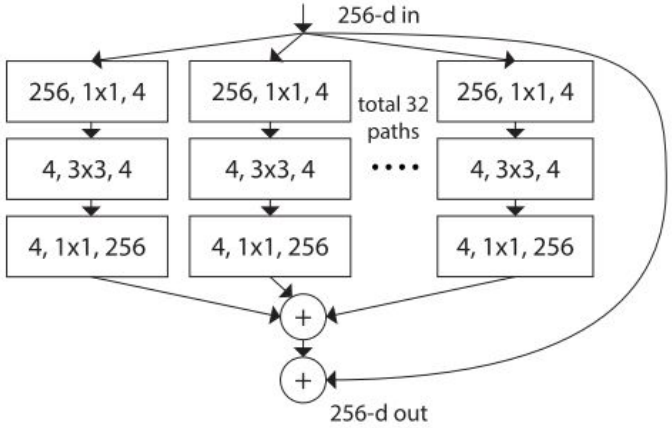
- Trend of split-transform-merge
- Minor changes on ResNet

- Inception style in ResNet
- Similar convolution topology.

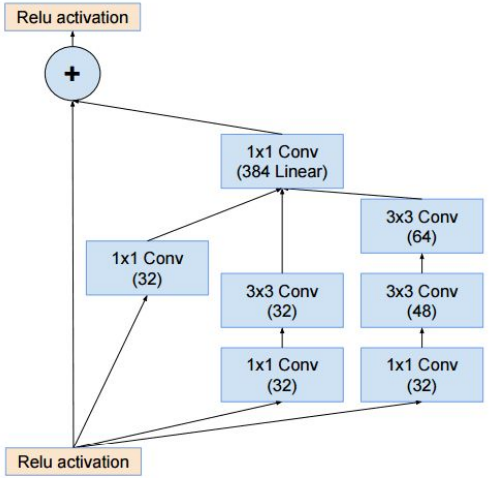
ResNet



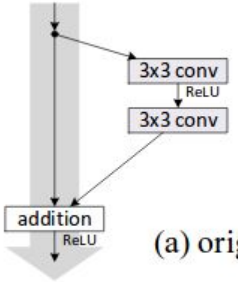
ResNeXT



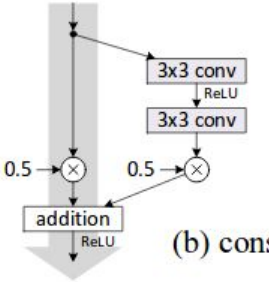
Inception-Resnet



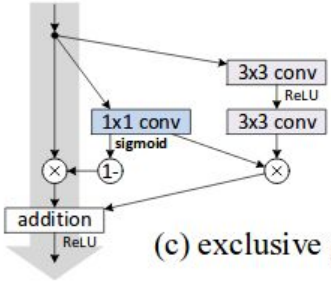
Extras:



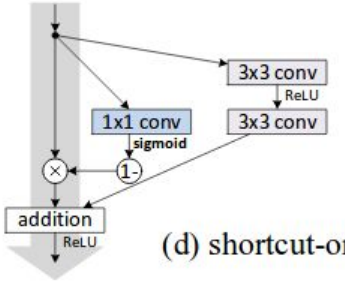
(a) original



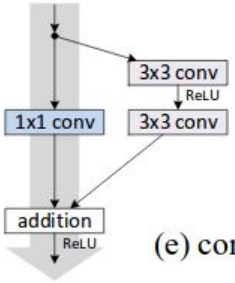
(b) constant scali



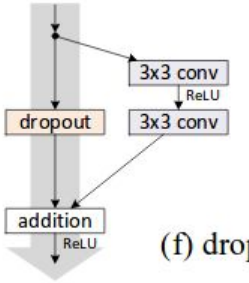
(c) exclusive gating



(d) shortcut-only



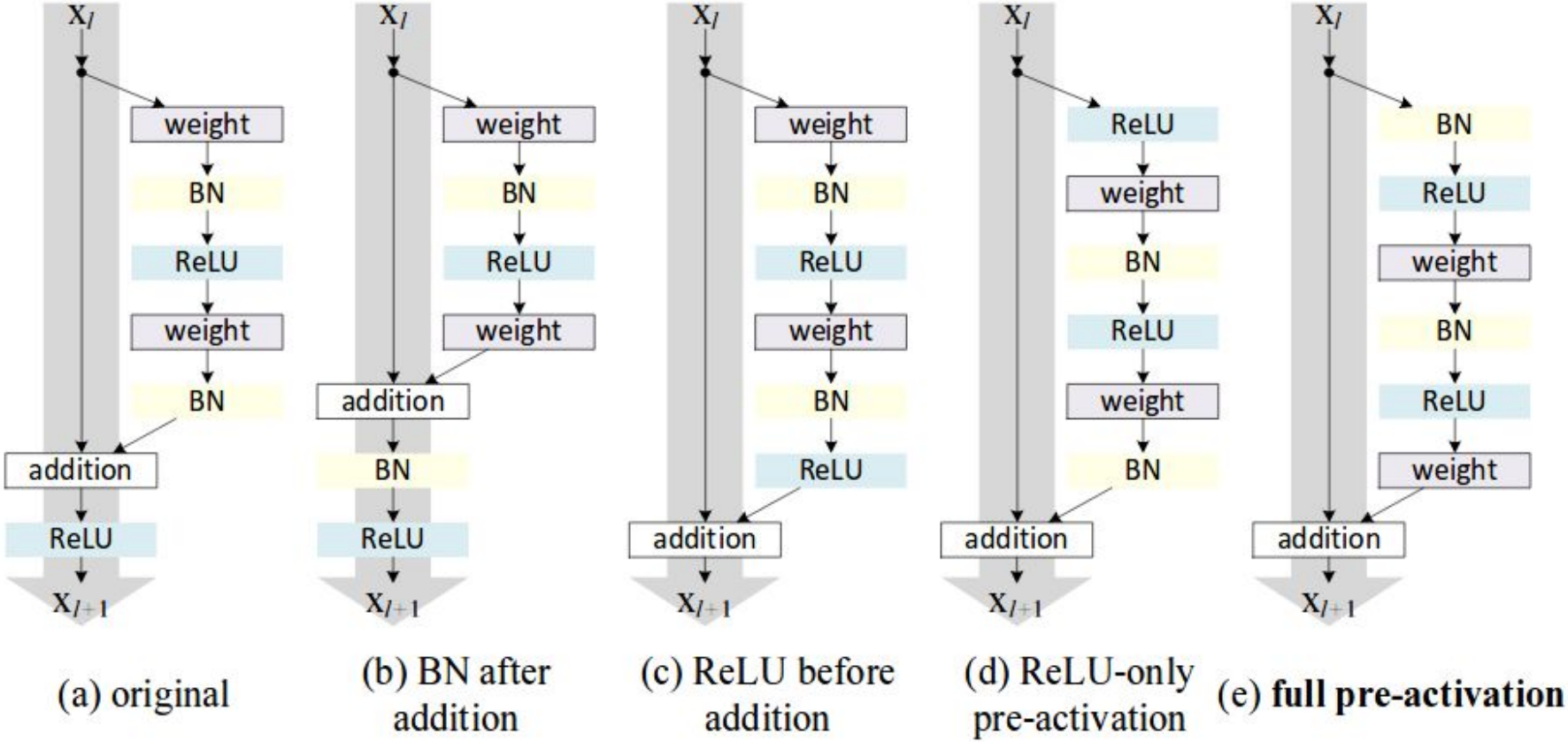
(e) conv shortcut



(f) dropout shortcut

case	Fig.	on shortcut	on $\mathcal{F}$	error (%)	remark
original [1]	Fig. 2(a)	1	1	**6.61**	
constant scaling	Fig. 2(b)	0	1	fail	This is a plain net
		0.5	1	fail	
		0.5	0.5	12.35	frozen gating
exclusive gating	Fig. 2(c)	$1-g(\mathbf{x})$	$g(\mathbf{x})$	fail	init b_g=0 to −5
		$1-g(\mathbf{x})$	$g(\mathbf{x})$	8.70	init b_g=-6
		$1-g(\mathbf{x})$	$g(\mathbf{x})$	9.81	init b_g=-7
shortcut-only gating	Fig. 2(d)	$1-g(\mathbf{x})$	1	12.86	init b_g=0
		$1-g(\mathbf{x})$	1	6.91	init b_g=-6
1×1 conv shortcut	Fig. 2(e)	1×1 conv	1	12.22	
dropout shortcut	Fig. 2(f)	dropout 0.5	1	fail	

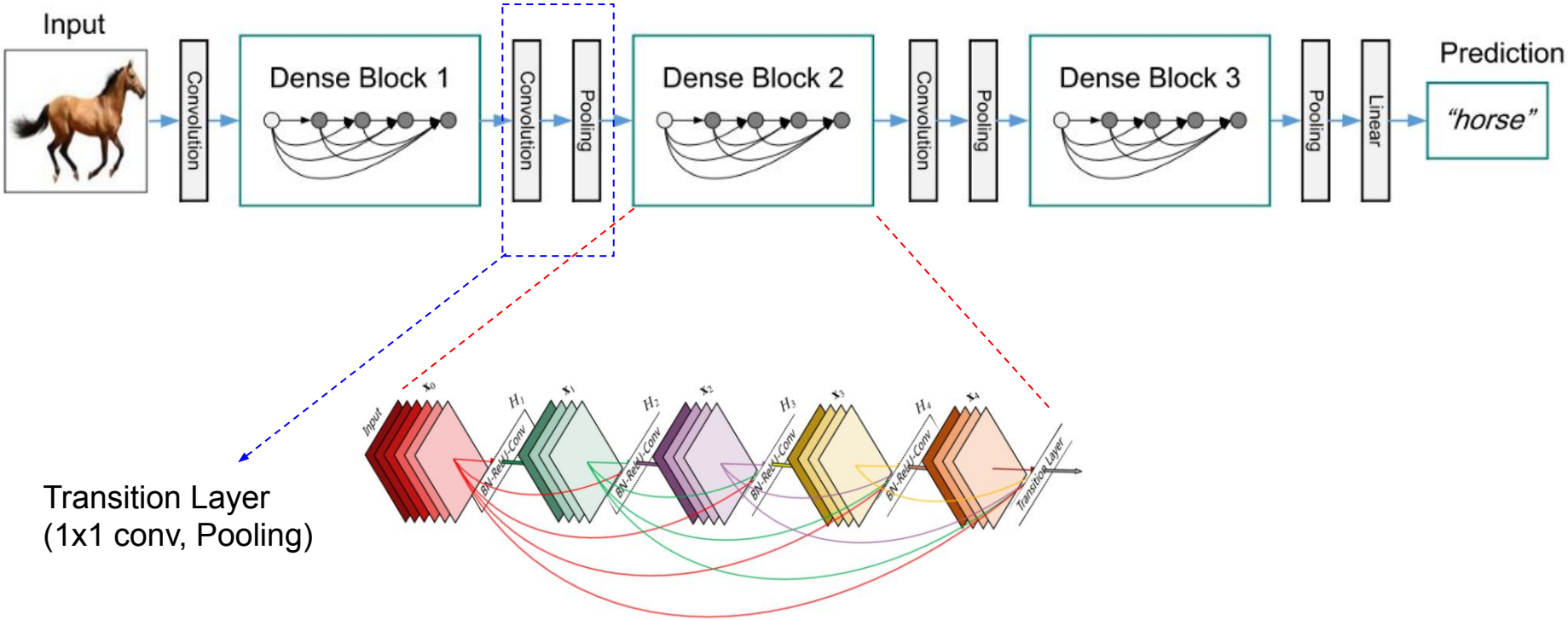
Extras:


x_l
weight
BN
ReLU
weight
BN
addition
ReLU
x_{l+1}
(a) original
x_l
weight
BN
ReLU
weight
addition
BN
ReLU
x_{l+1}
(b) BN after addition
x_l
weight
BN
ReLU
weight
BN
ReLU
addition
x_{l+1}
(c) ReLU before addition
x_l
ReLU
weight
BN
ReLU
weight
BN
addition
x_{l+1}
(d) ReLU-only pre-activation
x_l
BN
ReLU
weight
BN
ReLU
weight
addition
x_{l+1}
(e) full pre-activation

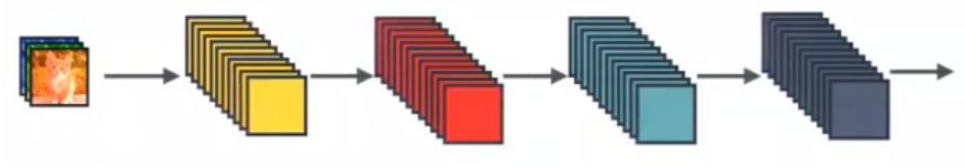
Interactive Implementation

Connections in ConvNets

connect every layer to one another



Standard Connectivity

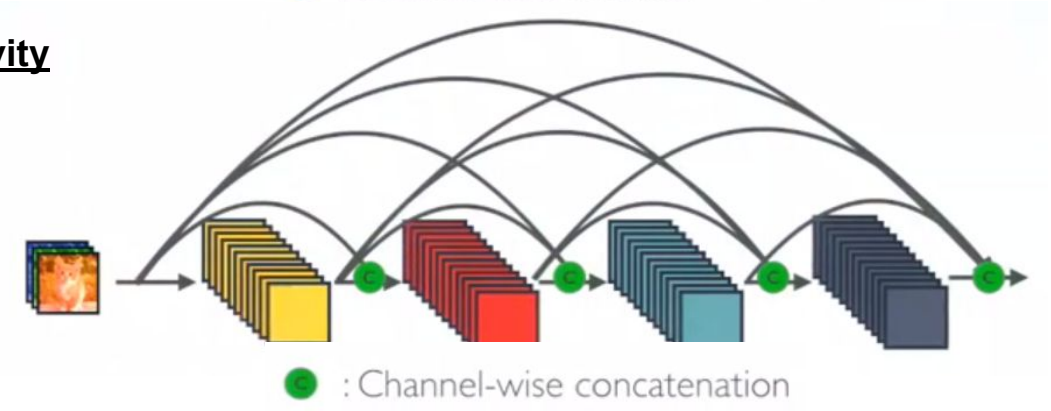


Successive convolutions

Resnet Connectivity

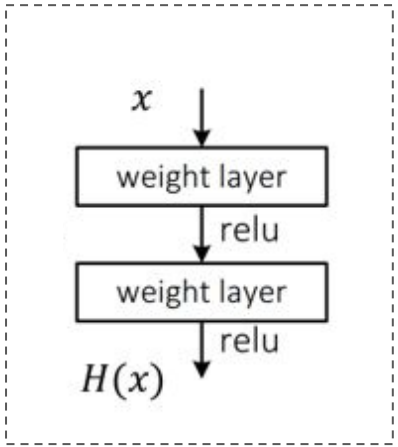
Element-wise feature summation

DenseNet Connectivity



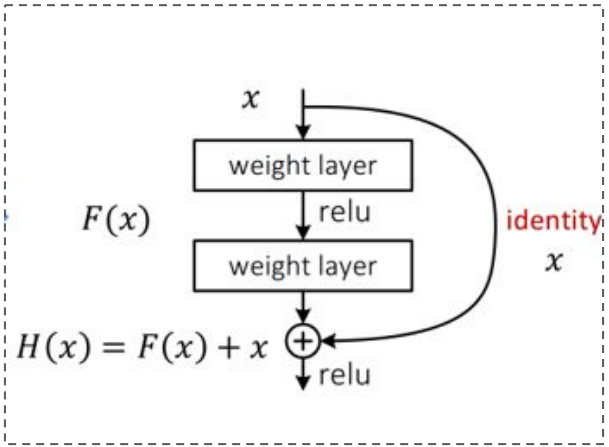
Feature concatenation

Standard Connectivity



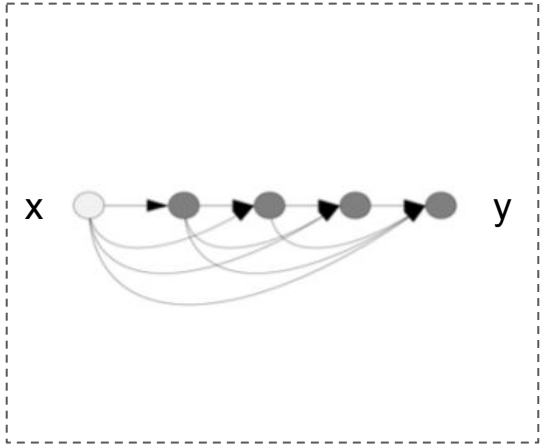
$$y = f(x)$$

ResNet Connectivity



$$y = f(x) + x$$

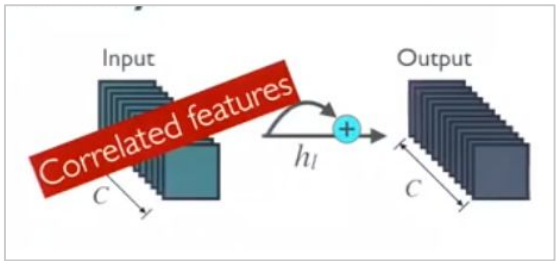
DenseNet Connectivity



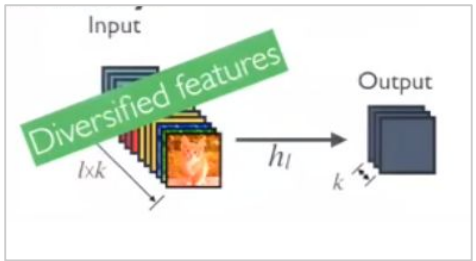
$$y = f(x,x\text{-}1,x\text{-}2\ldots x\text{-}n)$$

- power of feature reuse

ResNet

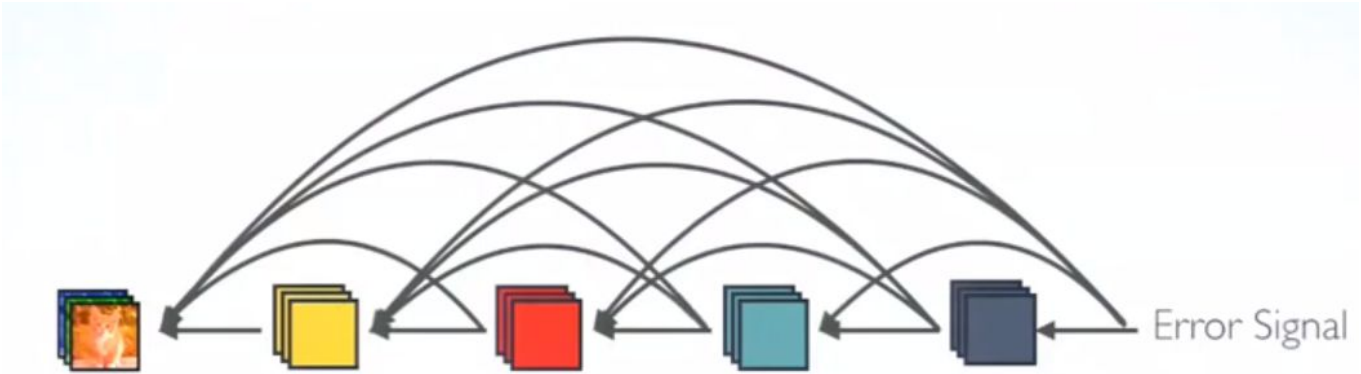


DenseNet



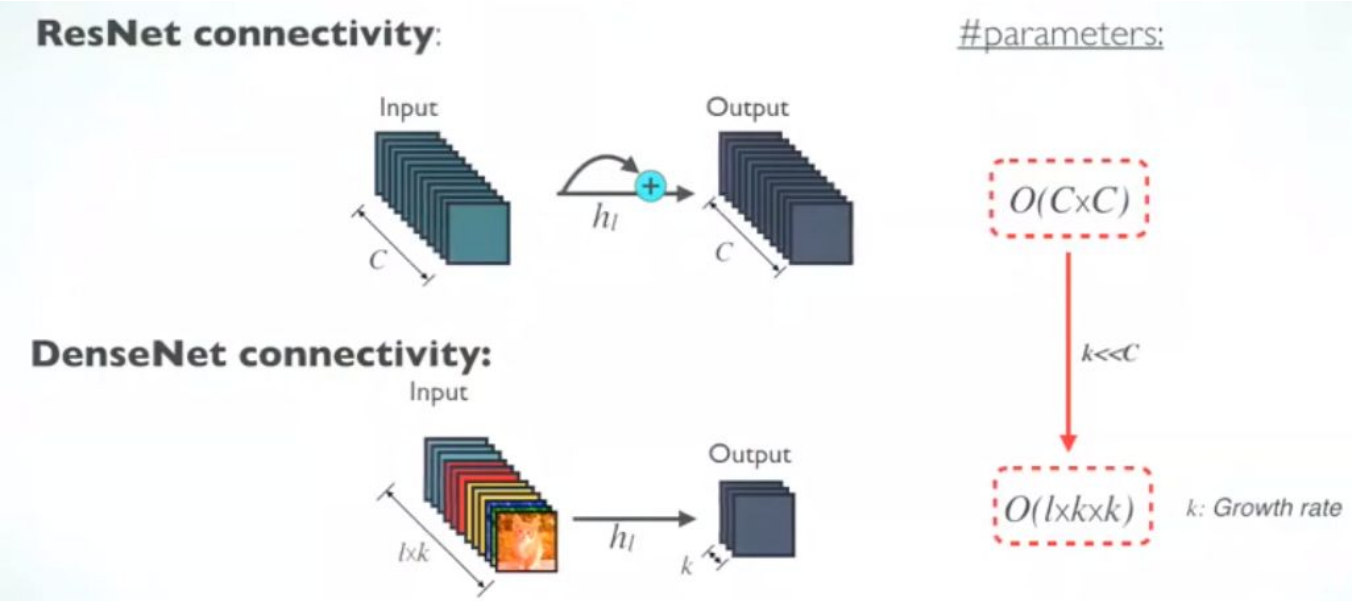
Maintains low complexity..

- More shortcut connections, better gradient flow

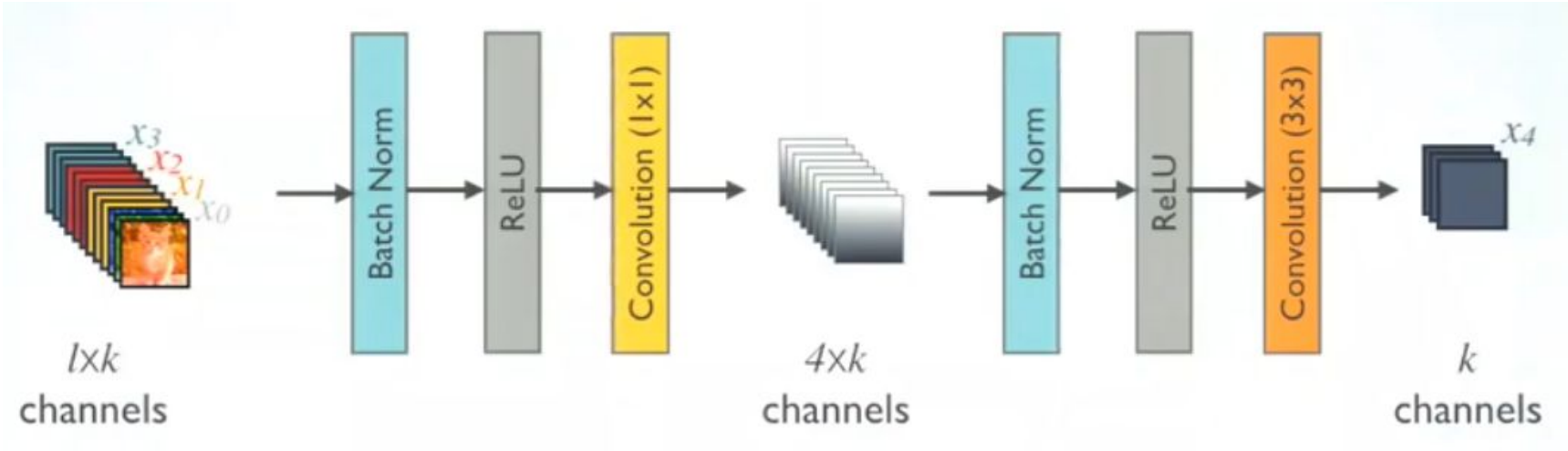


Supervision to gradients

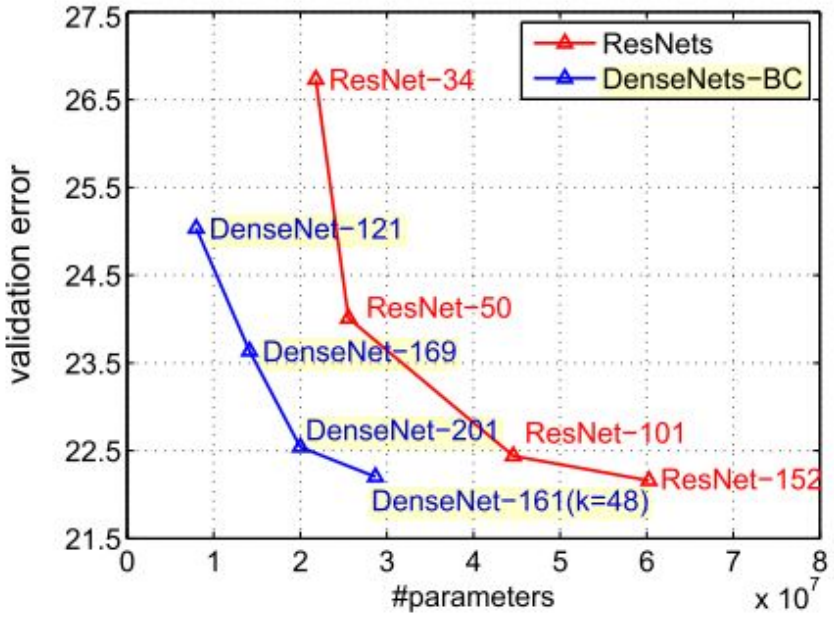
- Less parameters , computationally efficient

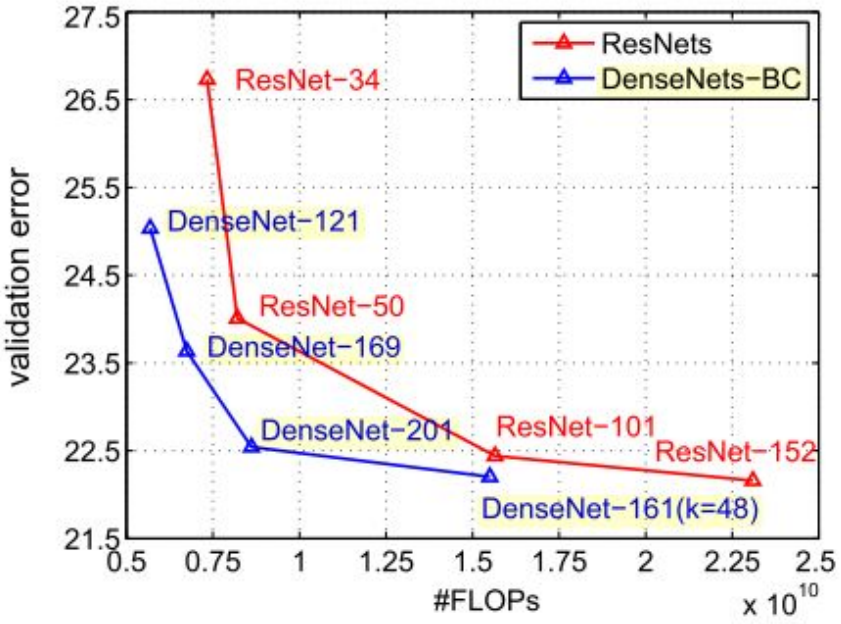


★ Bottleneck Layer



- Error vs parameters & computation





REFERENCES

[1] **DenResNet: Ensembling Dense Networks and Residual Networks**
Victor Cheung
http://cs231n.stanford.edu/reports/2017/pdfs/933.pdf

[2] **An Overview of ResNet and its Variants**
Vincent Fung
https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[3] **The Efficiency of Densenet**
Hao Gao
https://medium.com/@smallfishbigsea/densenet-2b0889854a92

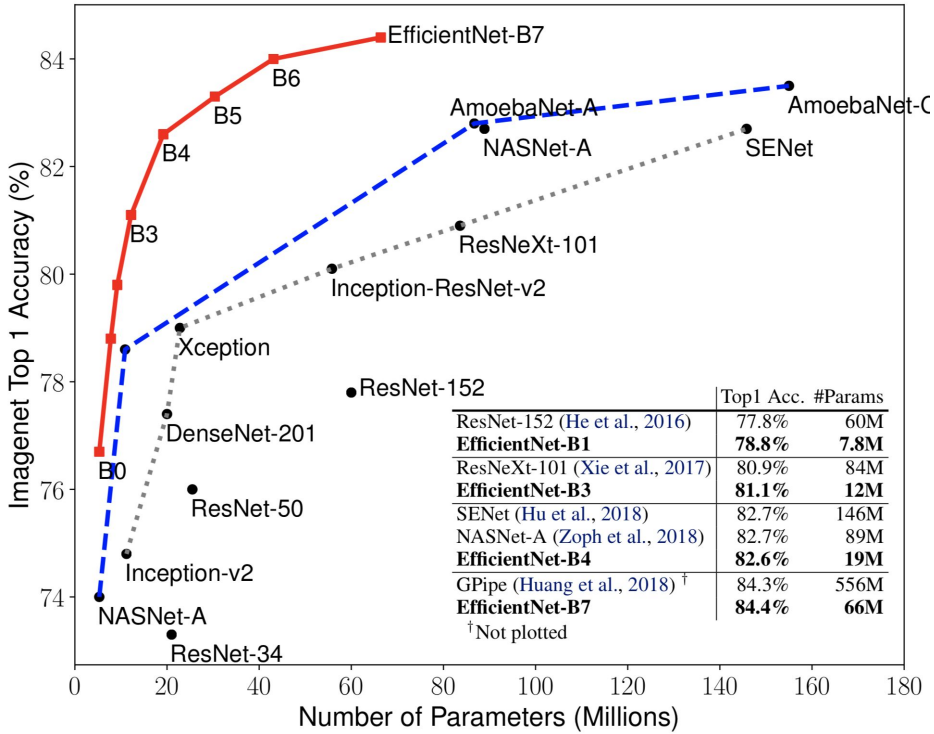
[4] **Understanding and Implementing Architectures of ResNet and ResNeXt**
Prakash Jay
https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cc5d0adf648e

[5] **Hand-Gesture Classification using Deep Convolution and Residual Neural Network**
Sandipan Dey
https://sandipanweb.wordpress.com/2018/01/20/hand-gesture-classification-using-deep-convolution-and-residual-neural-network-with-tensorflow-keras-in-python/

Part 3 : Extras

State-of-the-art:

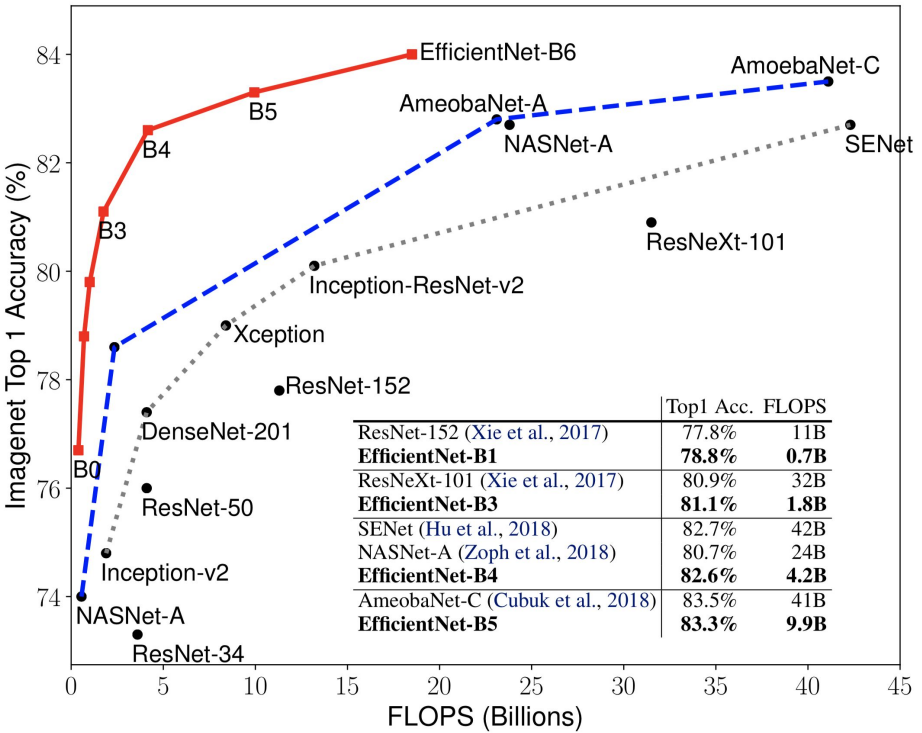
of parameters



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

of flops



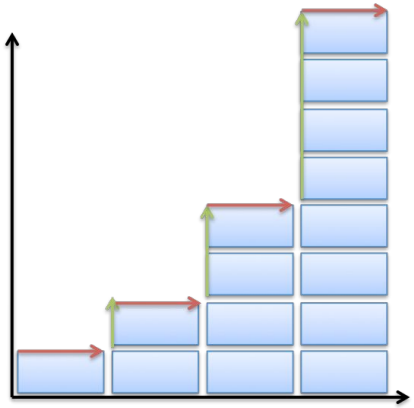
	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**78.8%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.1%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.6%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2018)	83.5%	41B
EfficientNet-B5	**83.3%**	**9.9B**

Source: https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet

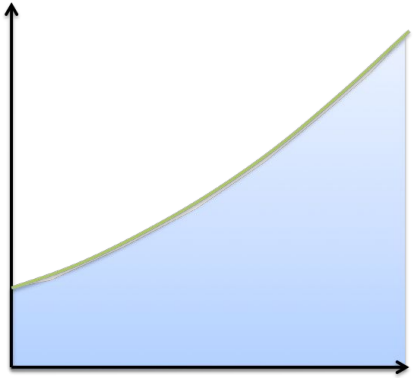
Neural ODEs

- So far we were using the layer (discrete) approach in neural networks,
- It worked well to differentiate classes
- but lacks when it comes to continuous events
- such as health records taken at random times..

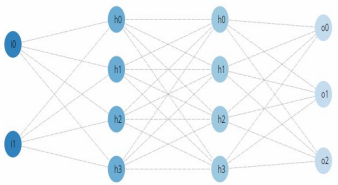
Discrete Growth (2^n)



Continuous Growth (e^x)



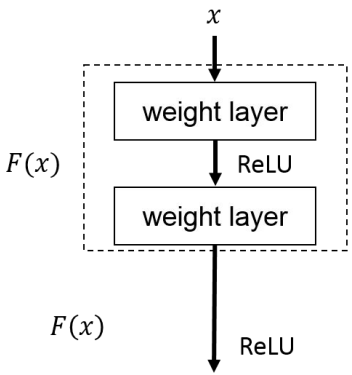
A Neural Net



Is possible to achieve continuity?

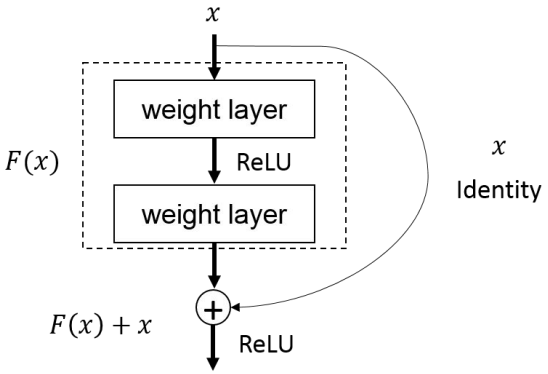
Let’s look a bit closer then;

A Plain Network




```
h1 = f1(x)
h2 = f2(h1)
h3 = f3(h2)
h4 = f3(h3)
y  = f5(h4)
```

A Residual Network




```
h1 = f1(x)  + x
h2 = f2(h1) + h1
h3 = f3(h2) + h2
h4 = f4(h3) + h3
y  = f5(h4) + h4
```

Very similar to an Euler-equation

Re-parameterizing continuous dynamics of hidden states by an ODE

$$
\begin{aligned}
h_{t+1} &= h_t + f(h_t) \\
&= h_t + \frac{\Delta t}{\Delta t} f(h_t) \\
&= h_t + \Delta t \, G(h_t)
\end{aligned}
$$

ResNet

```
def G(h, t, θ):
  return nnet(h, θ[t])

def resnet(h):
  for t in [1:T]:
    h = h + G(h, t, θ)
  return h
```

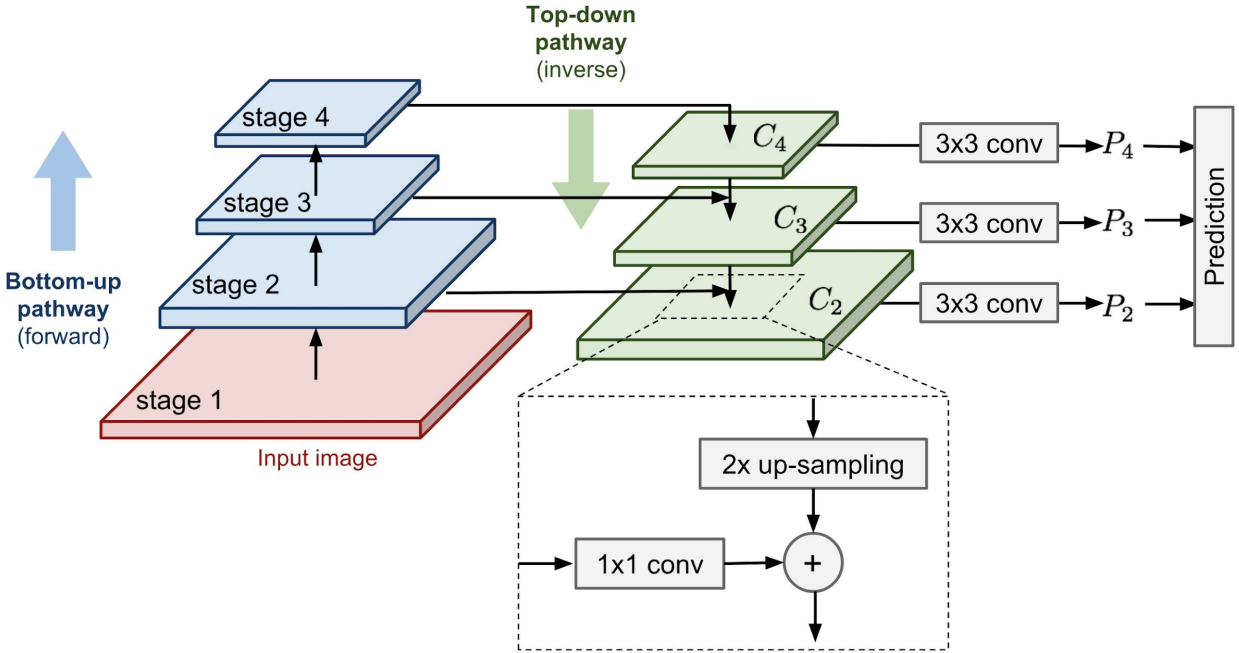
ODE-Net

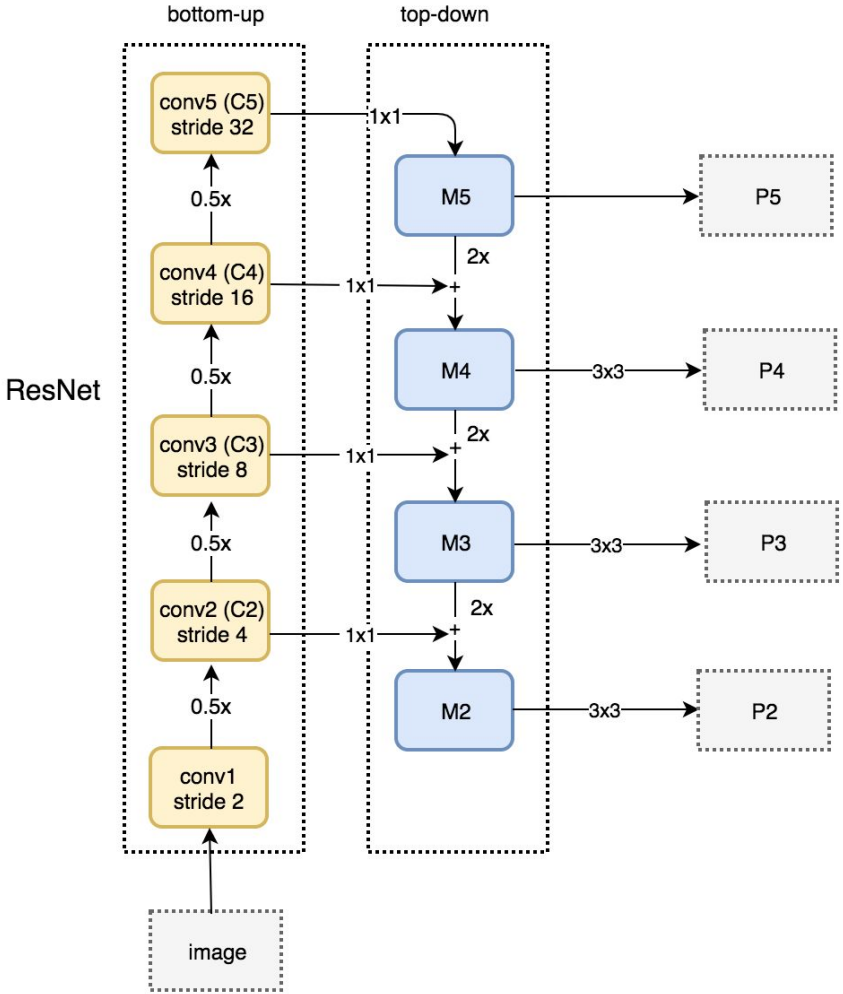
```
def G(h, t, θ):
  return nnet([h,t], θ)

def ODEnet(h, θ):
  return ODESolve(G, h, 0, 1, θ)
```

Feature Pyramid Networks

Top-down pathway restores resolution with rich semantic information



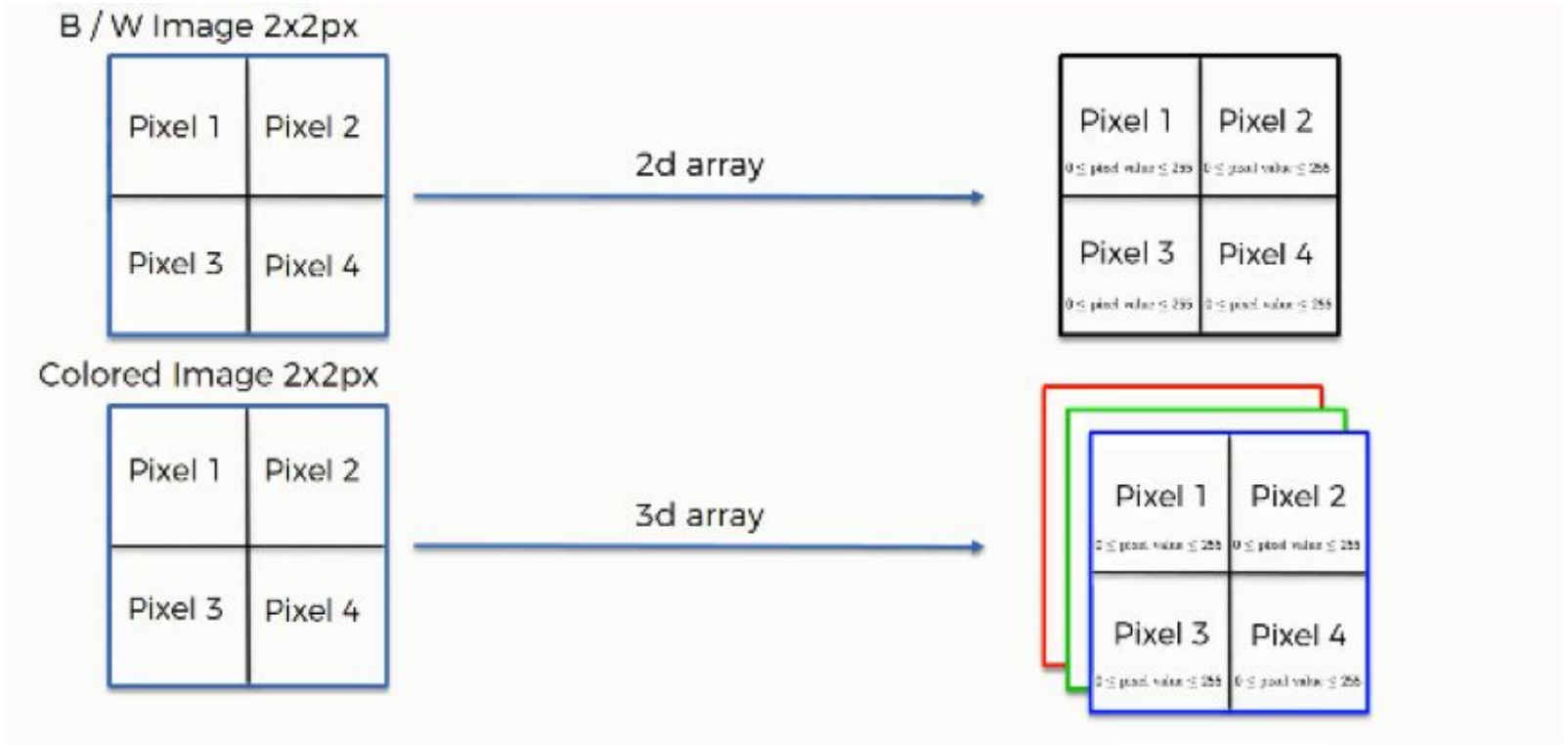


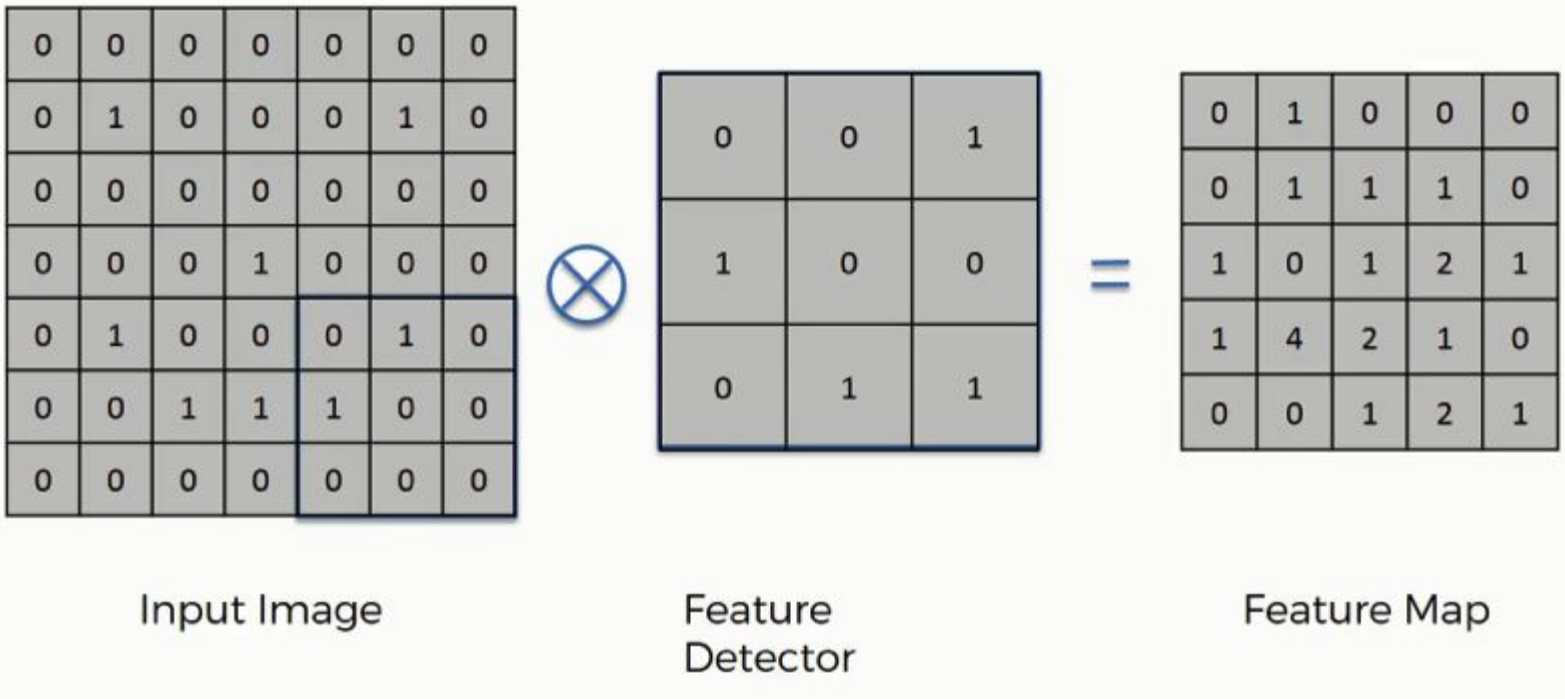
Bottom-Up Pathway:
applies ResNet to downscale by convolutions

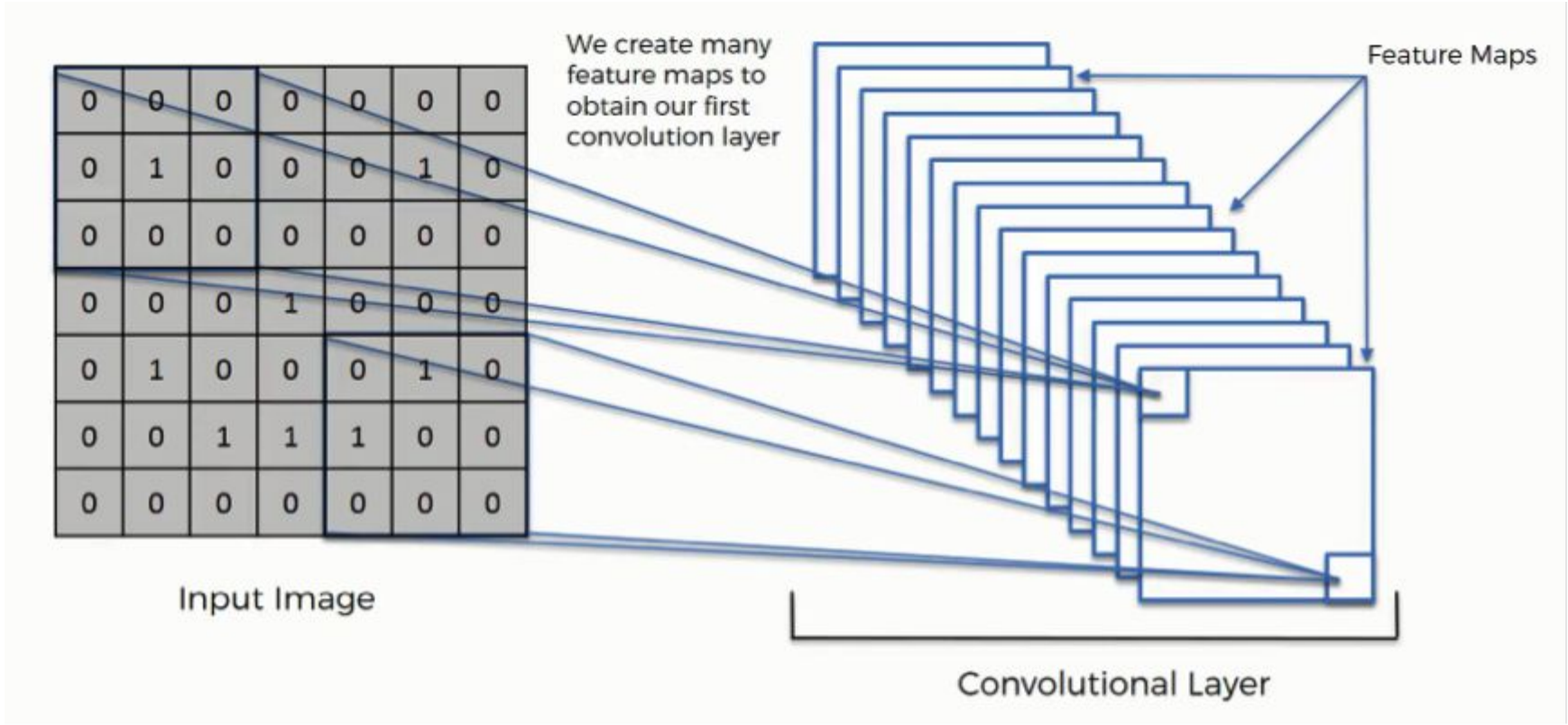
Top-Down Pathway:
applies 1x1 convolutions and neares neighbour to downscale and element-wise addition of feature maps

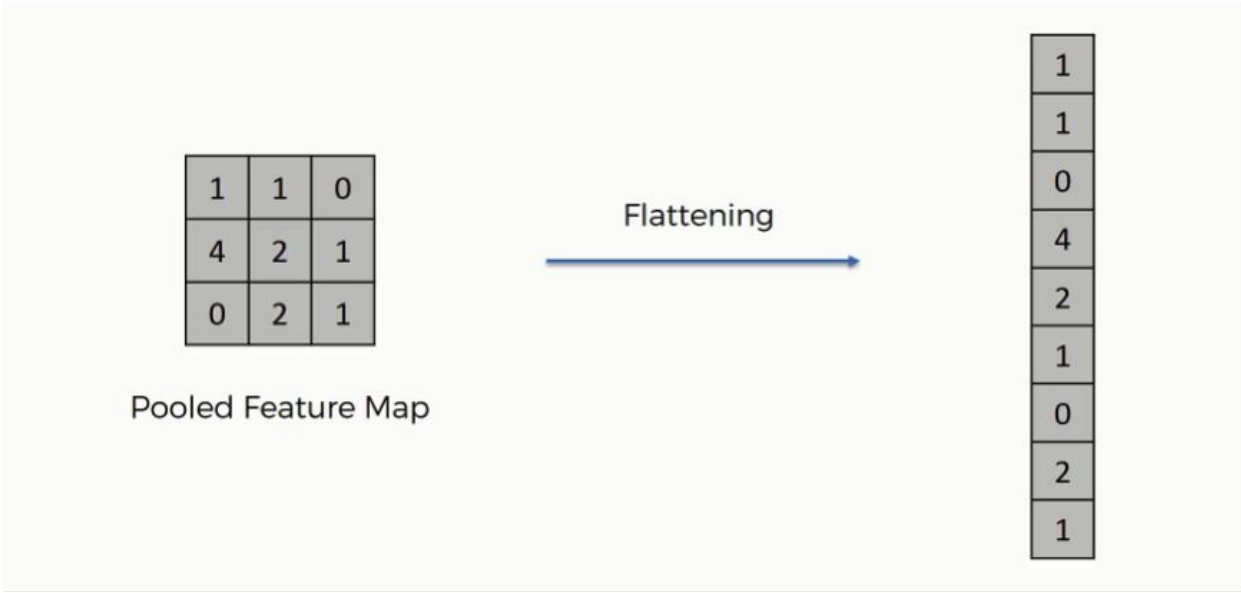
Interactive Implementation

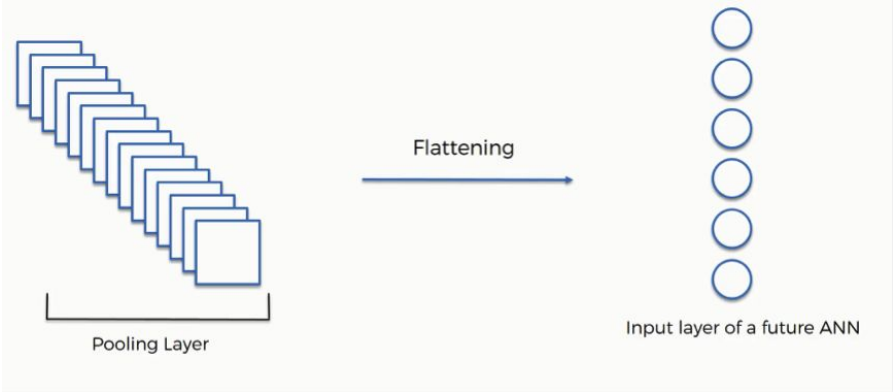
Appendix : State-of-the-art

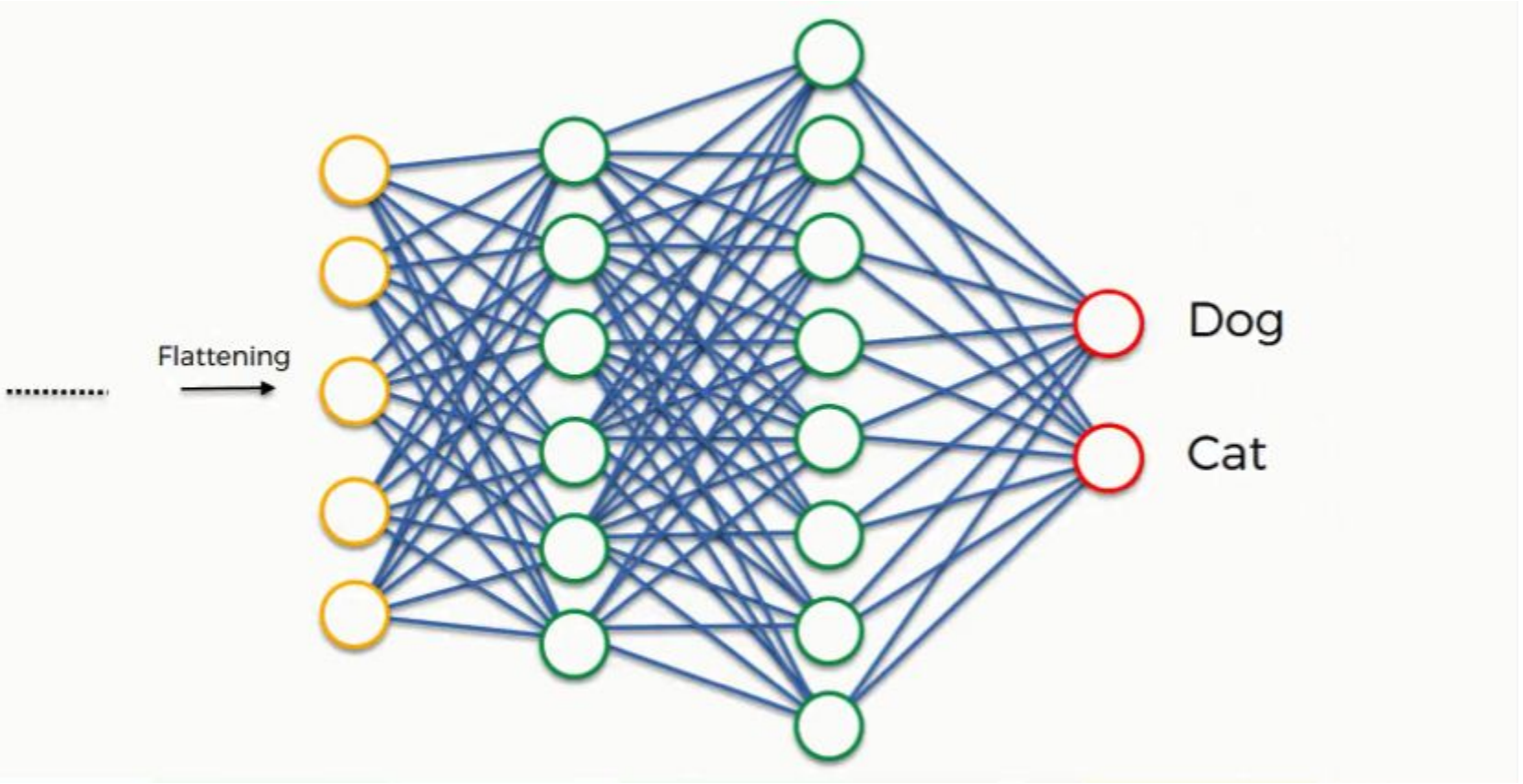

B / W Image 2x2px
Pixel 1
Pixel 2
Pixel 3
Pixel 4
2d array
Pixel 1
Pixel 2
Pixel 3
Pixel 4
Colored Image 2x2px
Pixel 1
Pixel 2
Pixel 3
Pixel 4
3d array
Pixel 1
Pixel 2
Pixel 3
Pixel 4

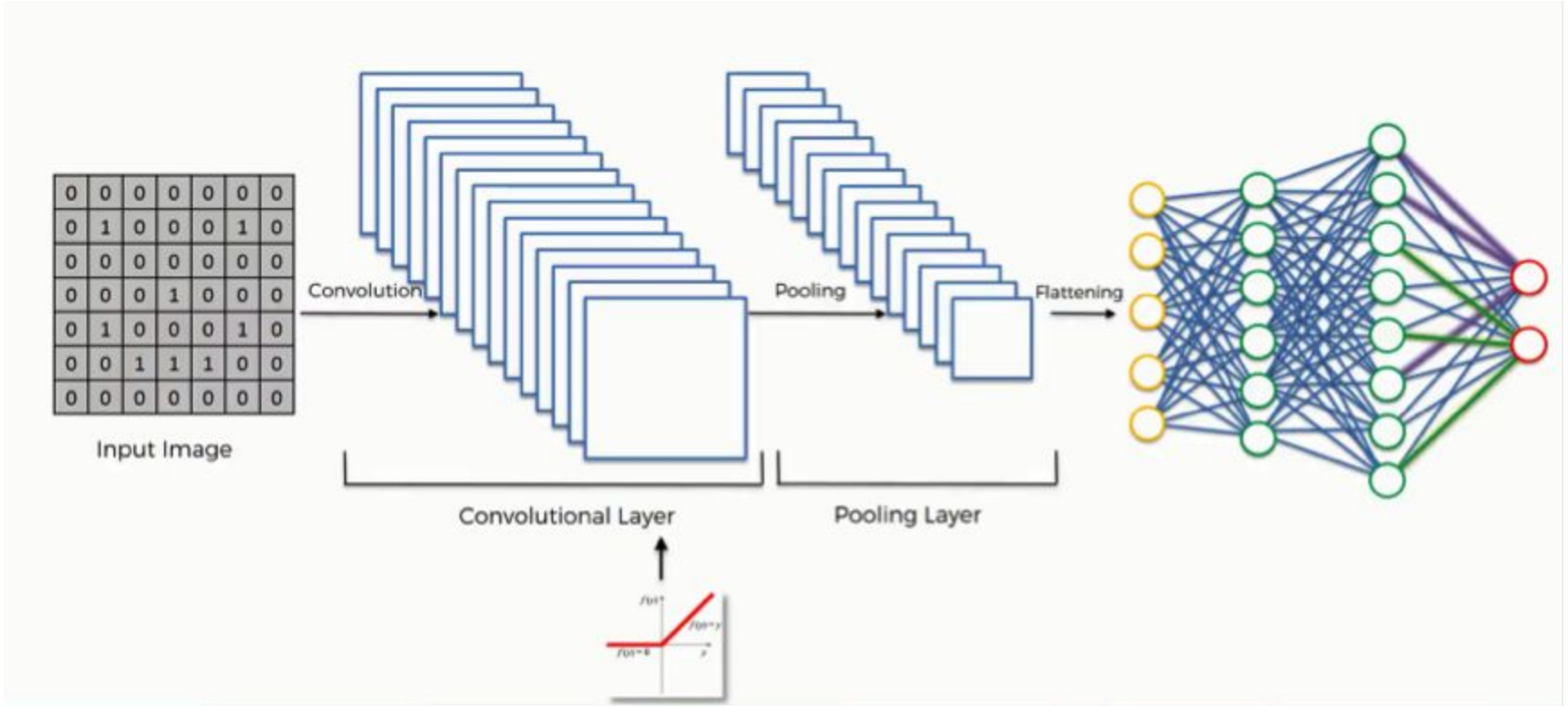

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Input Image
Feature Detector
Feature Map


We create many feature maps to obtain our first convolution layer
Feature Maps
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolutional Layer

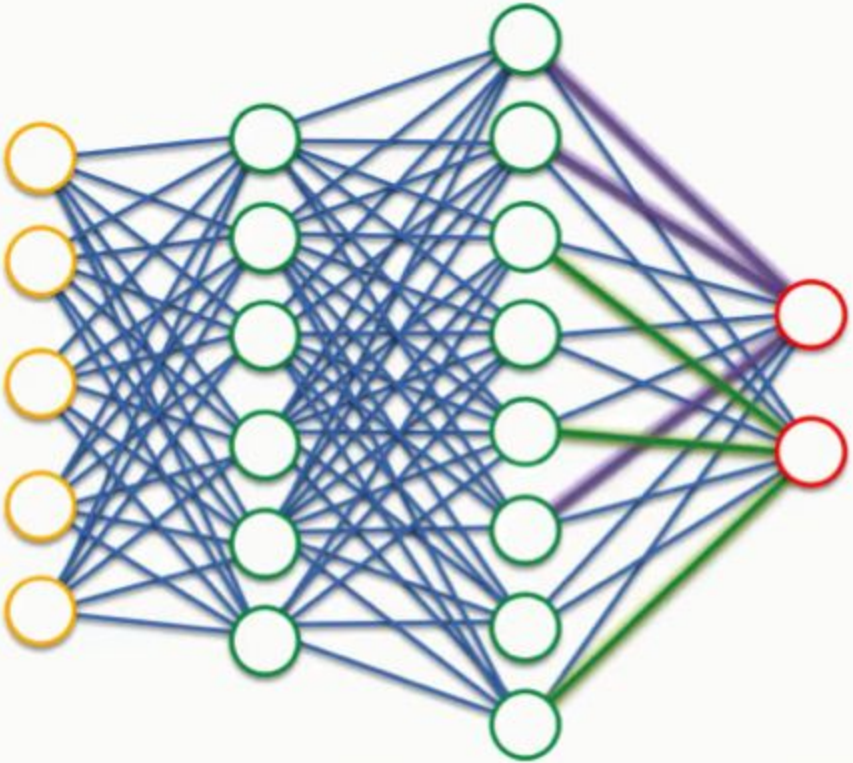

1 1 0
4 2 1
0 2 1
Pooled Feature Map
Flattening
1
1
0
4
2
1
0
2
1


Pooling Layer
Flattening
Input layer of a future ANN


Flattening
Dog
Cat


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Pooling
Flattening
Convolutional Layer
Pooling Layer

Flattening



Dog 0.95

Cat 0.05

Dog 0.9

Cat 0.1

$$H(p,q) = -\sum_{x} p(x)\ \log q(x)$$

1
0

NN1

Row	Dog^	Cat^	Dog	Cat
#1	0.9	0.1	1	0
#2	0.1	0.9	0	1
#3	0.4	0.6	1	0

NN2

Row	Dog^	Cat^	Dog	Cat
#1	0.6	0.4	1	0
#2	0.3	0.7	0	1
#3	0.1	0.9	1	0

Classification Error

1/3 = 0.33

1/3 = 0.33

Mean Squared Error

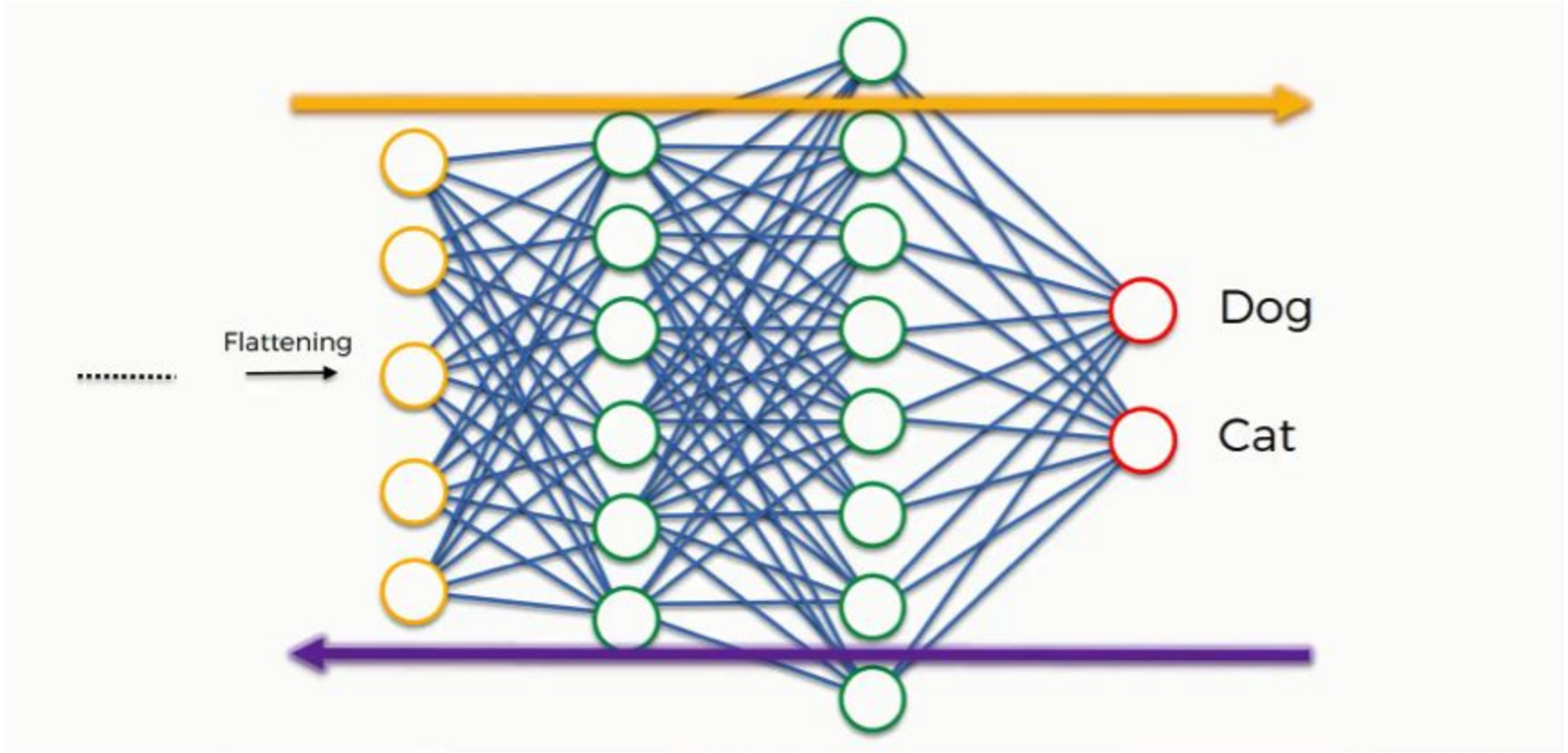
0.25

0.71

Cross-Entropy

0.38

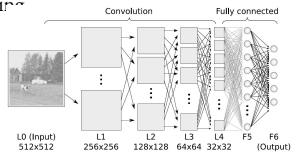
1.06


Flattening
Dog
Cat

Cheat Sheet

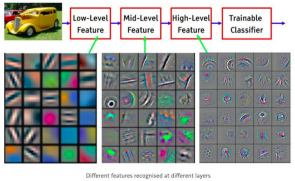
Convolution Operations

>Convolutions are basically a filtering operation used in CV world
>Extracting useful information from images
> Sliding windows (kernels or filter are used to convolve an input image



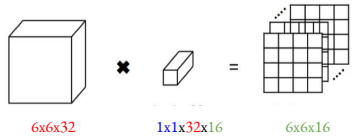
Convolution in CNN Architectures

> convolution: filtering
> stride: sliding step size
> padding: control output size
> pooling: downsampling



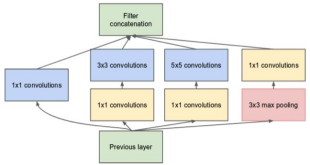
1x1 conv

> feature pooling
> decreases parameter
> decreases computation
> adds nonlinearity



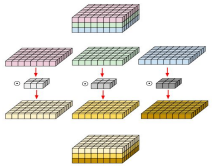
Inception/GoogLeNet

> use bottleneck layer
> decreases computation (10x)
> auxiliary loss layers
> factorize bigger conv layers
> regularization



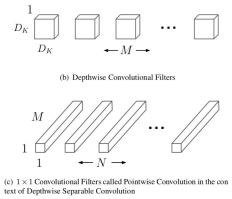
Depthwise Convolution

> each kernel is kept separately
> split input & kernels into channels
> convolve each input channel with corresponding filter channel
> stack the output (2D) tensors back together



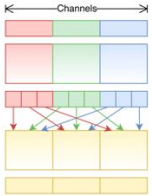
MobileNet

> depthwise separable convolutions
> shrinking hyperparameters
> width multiplier: adjusts # of channels
> resolution multiplier: adjusts input image and feature map resolutions



Channel Shuffling-ShuffleNet

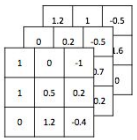
> eliminate main **side effect** of grouped convs
> **side effect:** outputs are derived only from certain channels, shuffles the channels after grouped convolutions
> apply group convolutions also on 1x1 layers
> note: channel shuffling is also differentiable!



11x11 vs 3x3

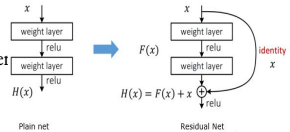
Filter size: 3x3

> Bigger filters capture, more global information
> Smaller filters captures more local information
> AlexNet uses 11x11, 55x55 and 3x3
> VGGNet uses only 3x3 filters
> By VGGNet, effectivity of going deeper proved



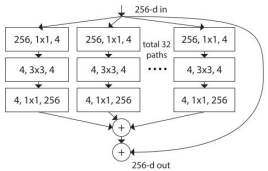
Residual Nets

> Identity shortcut (residual) connections
> Helps for gradient flow
> Skipping one or more layers
> Deeper architectures works better



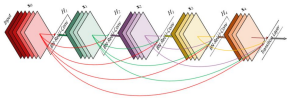
ResNeXt

> Inception style in ResNet
>Depth concatenation, same convolution topology
>Having high cardinality helps in decreasing validation error
>New hyper-parameter : cardinality → width size



DenseNet

>Connecting all layers to the other layers
>Strong gradient flow
>More diversified features
> Allowing feature re-use
>More memory hungry,
> computationally more efficient



THANK YOU FOR JOINING US TODAY!

Machine Learning Tokyo