Learn Convolutional Neural Network in One Day

Wei Li

National Chiao-Tung University
Computer Games and Intelligence (CGI) Lab

August 22, 2017

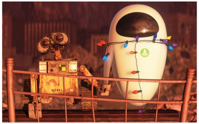
- Part of figures in my slide come from the following links:
  - **CS231n: Convolutional Neural Networks for Visual Recognition**
  - **Hung-yi Lee** - 一日搞懂深度学习
  - **Kai-Ming He** - **Deep residual networks tutorial**
  - **Xiu-Shen Wei** - **Must Know Tips/Tricks in Deep Neural Networks**
- See my **tutorial** for more information.
- I'm a first-year graduate student at National Chiao-Tung University(NCTU).

  Please feel free to contact me via **e-mail**[1] if you have any questions or concerns.

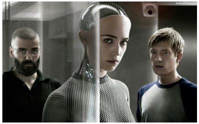
[1] **fm.bigballon[at]gmail.com**

Artifical Intelligence → Just around us


I, robot


WALL-E


The Imitation Game


Ex Machina

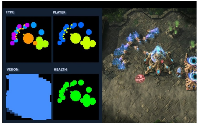

Deep Blue


しょうぎ

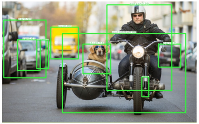
AlphaGo


Libratus Alan

Self-driving



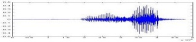
StarCraft2


Object Recognition


Robotic Arm

Machine Learning ≈ Looking for a Function

- Speech Recognition [2]

$f($  $)=$ "How are you"

- Image Recognition

$f($  $)=$ "Cat"

- Playing Go

$f($  $)=$ "5-5"

- Dialogue System

$f($ "Hi" $)=$ "Hello"

[2] Figure from Hung-yi Lee — 一日搞懂深度学习

Image Recognition - NARUTO

$f\Big($  $\Big) = Naruto$

$f\Big($  $\Big) = Naruto$

$f\Big($  $\Big) = Jiraiya$

$f\Big($  $\Big) = Minato$

$f\Big($  $\Big) = Sakura$

$f\Big($  $\Big) =???Wtf$

- Input(Training) data: 『NARUTO -ナルト-』's image
- Then we build a model $f(.)$ to predict an image is 『NARUTO -ナルト-』's character or not.

Image Recognition - NARUTO(cont.)

- We can use lots of methods to train our model
- But we only consider **Convolution Neural Networks**

Training data

Training
~~SVM~~
~~KNN~~
...
...
CNN

f^*

Training Part

f^*

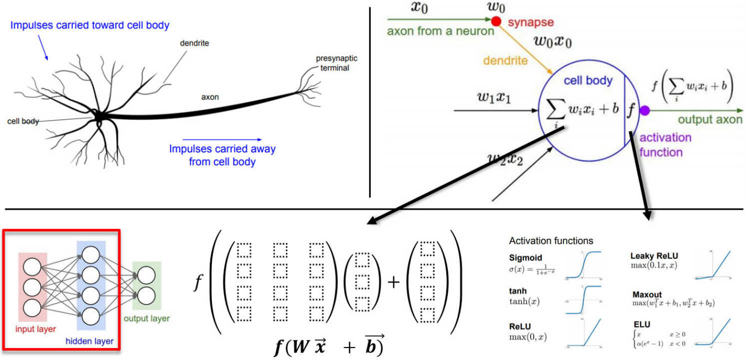
Iruka	90%
Minato	8%
...	..
...	..
Naruto	..

Iruka

Testing Part

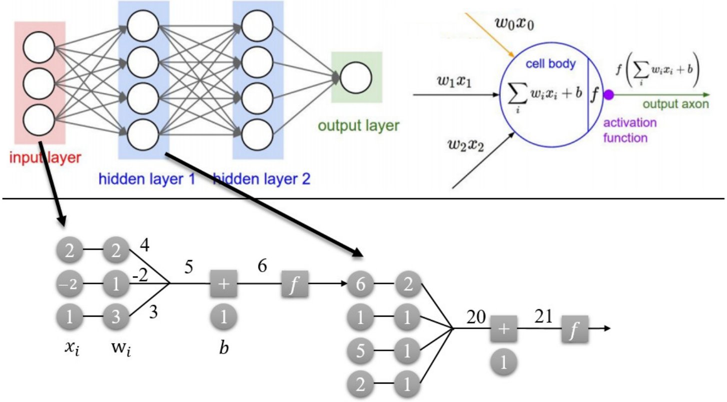
Neural Network

- An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information.



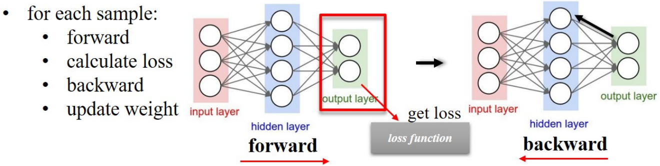
Neural Network(cont.)

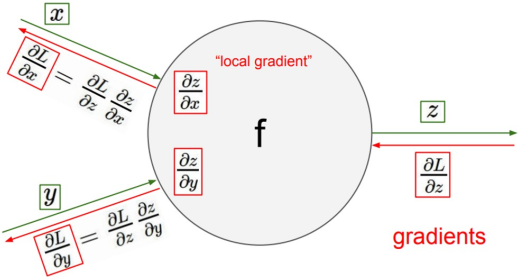
- Fully connected neural network = input layer + hidden layer + output layer



Backpropagation

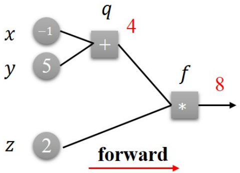
- How to update weights? Back Propagation !



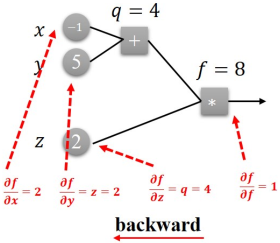


Backpropagation(cont.)

- How to update weights? Back Propagation !
- Keyword: **Chain Rule**



$$f = (x+y)*z \rightarrow \begin{cases} q = x+y \\ f = q*z \end{cases}$$



$$f = q*z \rightarrow \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

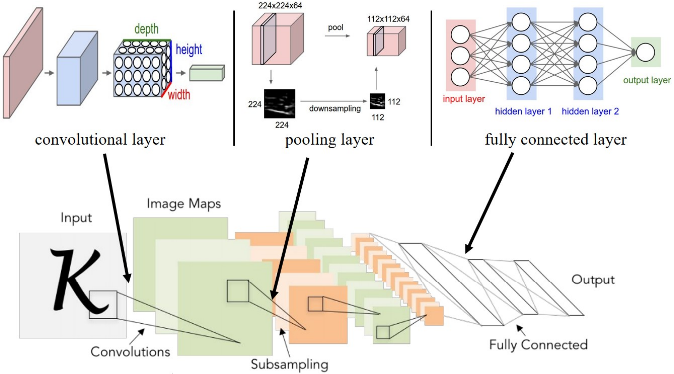
$$q = x+y \rightarrow \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\rightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial x} = z$$

$$f = (x+y)*z \rightarrow \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial y} = z$$

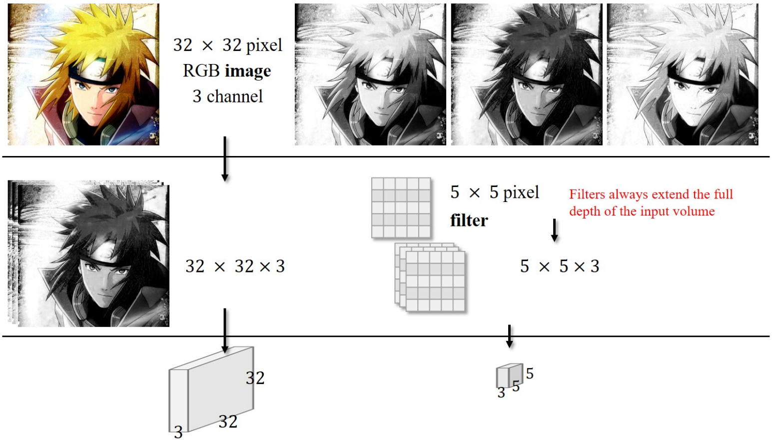
Convolutional Neural Network

- A Convolutional Neural Network (CNN) is comprised of one or more **convolutional layers** (often with a **subsampling step**) and then followed by one or more **fully connected layers** as in a standard multilayer neural network.
  - **Convolutional Layer**
  - **Pooling Layer**
  - **Fully-Connected Layer**(exactly as seen in regular Neural Networks)

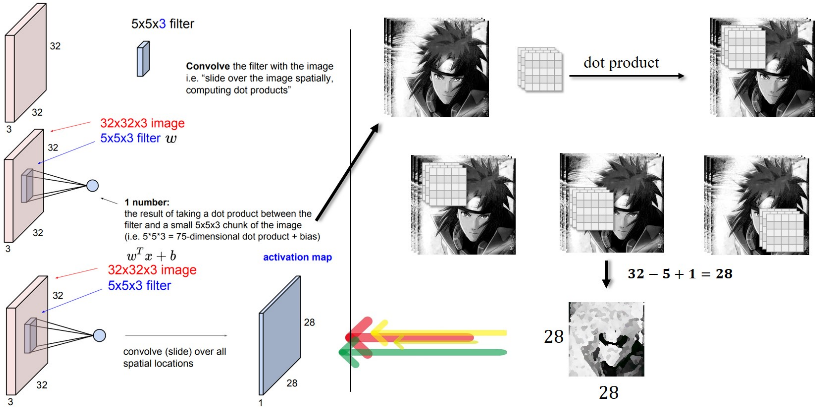


The first successful applications of Convolutional Networks: **LeNet-5**

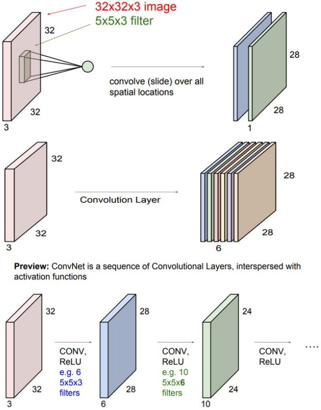
Convolutional Layer



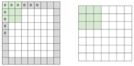
Convolutional Layer(cont.)



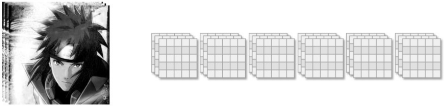
Convolutional Layer(cont.)



stride



padding vs. no padding



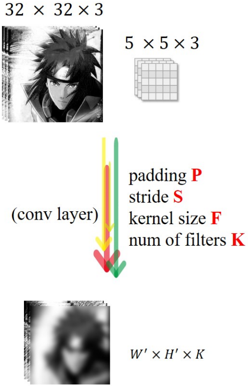
We will get the 28x28 filters because the stride ***S*** **= 1**

If we had 6 5x5 filters, we'll get 6 separate feature maps

The depth of the output **is equal to** the number of filters

Convolutional Layer(cont.)

- Hyperparameters(P,S,F,K)



$W = 32\,, H = 32\,, D = 3\,, F = 5$

Padding: $P = 0$
Stride: $S = 1$

$$W' = \frac{32-5}{1} + 1 = 28$$

28×28

Padding: $P = 0$
Stride: $S = 2$

$$W' = \frac{32-5}{2} + 1 = 14$$

14×14

Padding: $P = 2$
Stride: $S = 1$

$$W' = \frac{32-5+2*2}{1} + 1 = 32$$

32×32

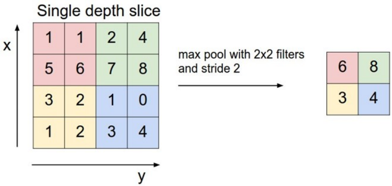
$W \times H \times D$

$$W' = (W - F + 2P)/S + 1$$
$$H' = (H - F + 2P)/S + 1$$
$$D' = K$$

$W' \times H' \times D'$

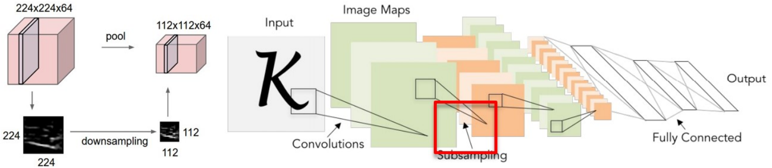
Pooling Layer

- Max pooling
- Avg pooling
- L2-norm pooling
- ...

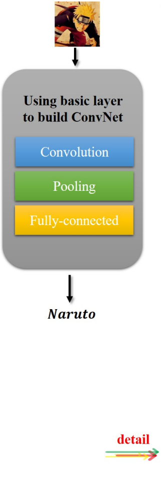


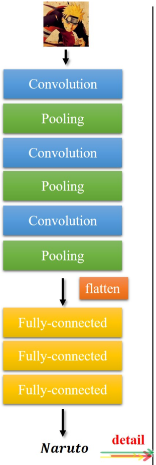
$$W' = (W - F)/S + 1$$
$$H' = (H - F)/S + 1$$
$$D' = D$$

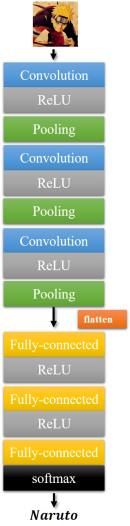
$$W \times H \times D \longrightarrow W' \times H' \times D'$$

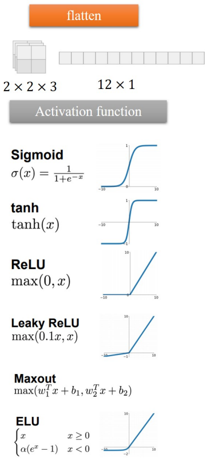


ConvNet


Using basic layer to build ConvNet
Convolution
Pooling
Fully-connected
Naruto
detail


Convolution
Pooling
Convolution
Pooling
Convolution
Pooling
flatten
Fully-connected
Fully-connected
Fully-connected
Naruto
detail


Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
flatten
Fully-connected
ReLU
Fully-connected
ReLU
Fully-connected
softmax
Naruto

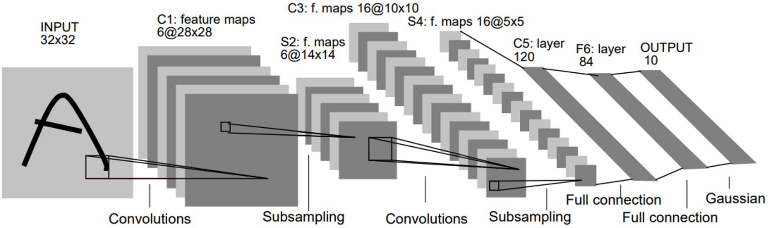

flatten
$2 \times 2 \times 3$
12×1
Activation function
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh
$\tanh(x)$
ReLU
$\max(0, x)$
Leaky ReLU
$\max(0.1x, x)$
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

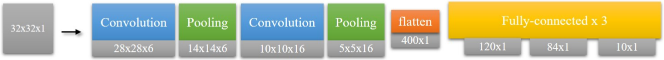
Convolutional Neural Network Architectures

- LeNet-5
- AlexNet
- Network in Network
- VGG Network
- GoogLeNet
- Residual Network
- Wide Residual Network
- ResNeXt Network
- DenseNet
- Dual Path Network

LeNet-5 - Overview

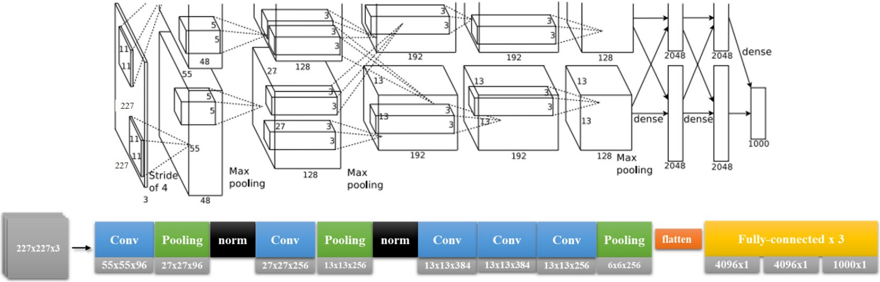
- The **first** successful applications of Convolutional Networks were developed by Yann LeCun in **1990's**.
- Paper: **Gradient-Based Learning Applied to Document Recognition**
- Project page: **lecun-lenet**





AlexNet - Overview

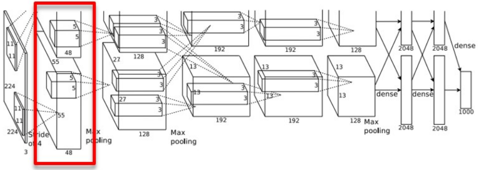
- The first work that popularized Convolutional Networks in Computer Vision
- ImageNet 2012 winner
- Paper: **ImageNet Classification with Deep Convolutional Neural Networks**



AlexNet - Detail

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

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

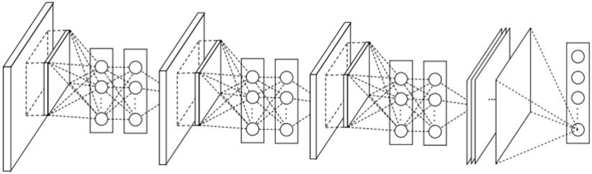


Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

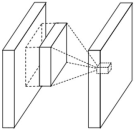
$$[55 \times 55 \times 48] \times 2$$

Network in Network - Mlpconv

- Paper: **Network In Network**

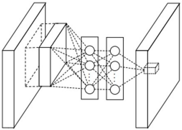


- **Mlpconv layer**



$$f_{i,j,k} = \max(w_k^T x_{i,j}, 0).$$

(a) Linear convolution layer

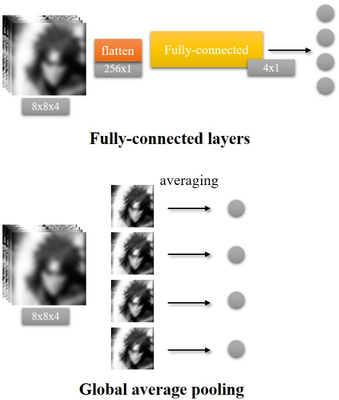


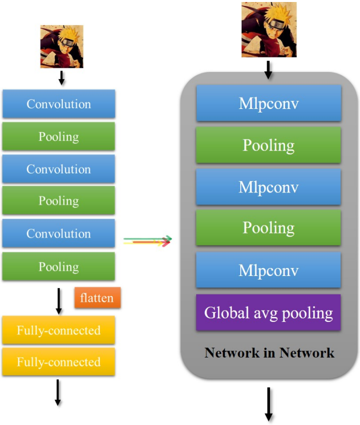
$$\begin{aligned} f^1_{i,j,k_1} &= \max({w^1_{k_1}}^T x_{i,j} + b_{k_1}, 0). \\ &\vdots \\ f^n_{i,j,k_n} &= \max({w^n_{k_n}}^T f^{n-1}_{i,j} + b_{k_n}, 0). \end{aligned}$$

(b) Mlpconv layer

Network in Network - Global average pooling

- **Global average pooling**





VGG Network - Overview

- Paper: **Very Deep Convolutional Networks for Large-Scale Image Recognition**
- Project page: **Visual Geometry Group**

8 layers (AlexNet) → 19 layers (VGGNet)

11x11(5x5,3x3) conv → 3x3 conv

11.7% top 5 error in ILSVRC'13(ZFNet)
→ 7.3% top 5 error in ILSVRC'14

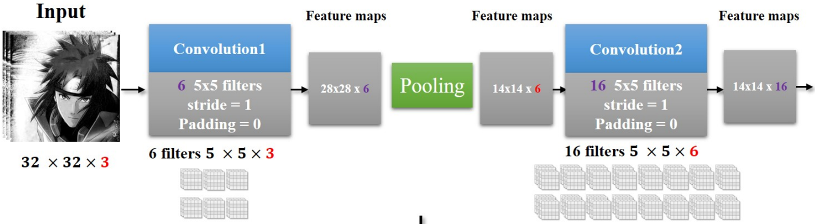
Why use smaller filters? (3x3 conv)

1. **Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer**
2. **But the network is deeper and more non-linearity**
3. **And fewer parameter:**

$$3 * (3^2C^2) \text{ vs. } (7^2C^2)$$
$$27C^2 \text{ vs. } 49C^2$$

VGG Network - Calculate params

- How to cal the params?



Parameters:

Convolution1 : $F^2 * D * K = (5 \times 5 \times 3) * 6 = 450$

Convolution2 : $F^2 * D * K = (5 \times 5 \times 6) * 16 = 2400$

Pooling layer: 0

W	Width of feature map
H	Height of feature map
D	Depth of feature map
F	Size of the filters
K	Number of the filters

VGG Network - Detail

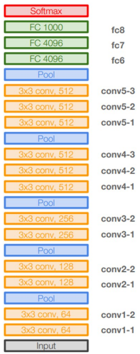
(not counting biases)

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

Most memory is in early CONV

Most params are in late FC

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

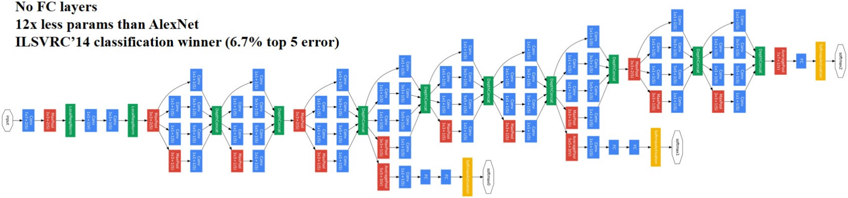


VGG16

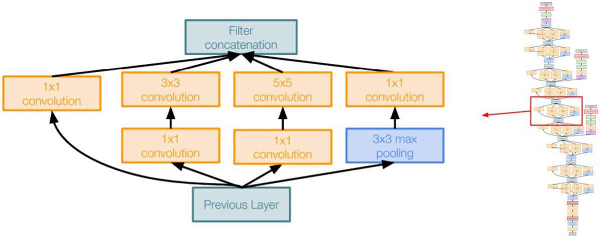
GoogLeNet - Overview

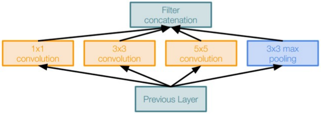
- Papers
  - [V1]: Going Deeper with Convolutions
  - [V2]: Accelerating Deep Network Training by Reducing Internal Covariate Shift
  - [V3]: Rethinking the Inception Architecture for Computer Vision
  - [V4]: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

22 layers
Efficient "Inception" module
No FC layers
12x less params than AlexNet
ILSVRC'14 classification winner (6.7% top 5 error)

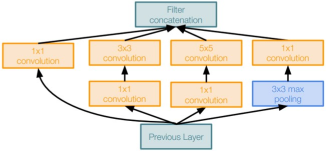


GoogLeNet - Inceptions



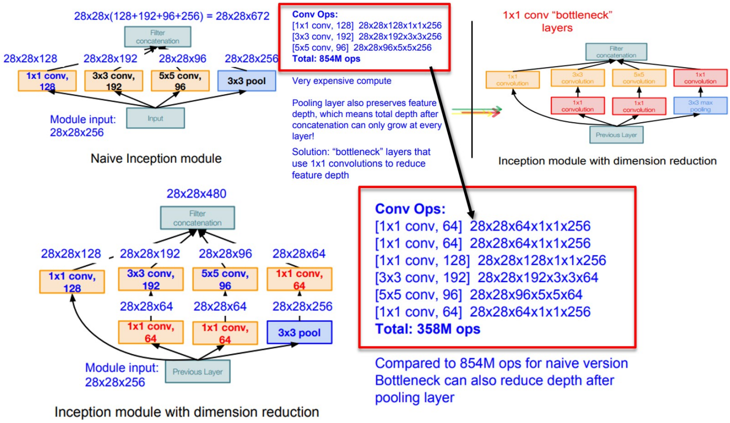


Naive Inception module



Inception module with dimension reduction

GoogLeNet - Detail



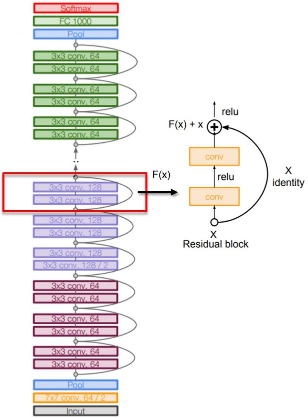
Residual Network - Overview

- Papers
  - **Deep Residual Learning for Image Recognition**
  - **Identity Mappings in Deep Residual Networks**

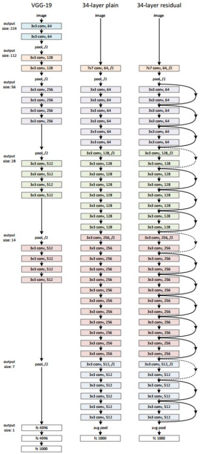
- **MSRA - Kaiming He**
- **152-layer model for ImageNet**
- **ILSVRC'15 classification winner (3.57% top 5 error)**
- **Swept all classification and detection competitions in ILSVRC'15 and COCO'15**

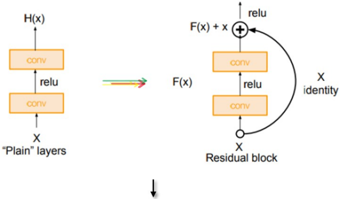
Full ResNet Architecture:

- **Stack residual blocks**
- **Every residual block has two 3x3 conv layers**
- **Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)**
- **Additional conv layer at the beginning**
- **No FC layers at the end (only FC 1000 to outputclasses)**



Residual Network - Residual Block





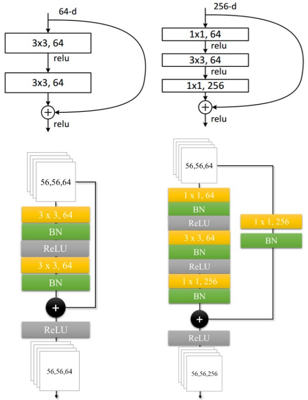
assume $x = 2.9$,
after two conv layers $H_1(x) = 3.0, F(x) = 0.1$
after two conv layers $H_2(x) = 3.1, F(x) = 0.2$

Plain layer: $\Delta = \frac{3.1 - 3.0}{3.0} = 3.3\%$

Residual block: $\Delta = \frac{0.2-0.1}{0.1} = 100\%$

"We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping"——authors

Residual Network - Bottleneck



Residual block: **2 layers** **(3-3)**

Params:

$$(3 \times 3 \times 64) * 64$$
$$+(3 \times 3 \times 64) * 64$$
$$= 73K$$

Bottleneck: **3 layers** **(1-3-1)**

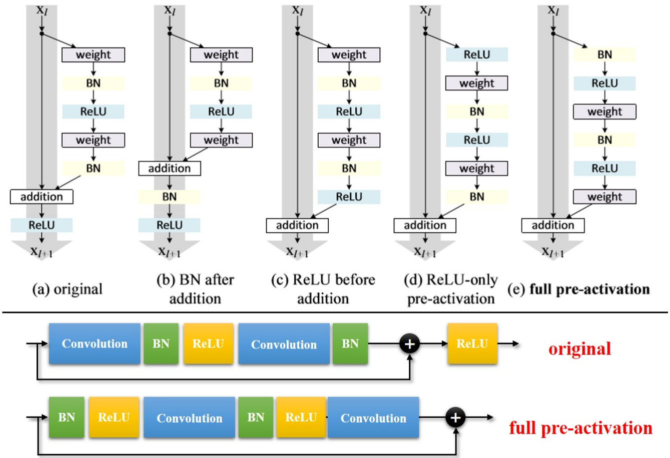
Params:

$$(1 \times 1 \times 64) * 64$$
$$+(3 \times 3 \times 64) * 64$$
$$+(1 \times 1 \times 64) * 256$$
$$+(1 \times 1 \times 64) * 256$$
$$= 73K$$

	34-layer	50-layer
	7×7, 64, stride 2	
	3×3 max pool, stride 2	
	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
2	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$
2	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$
2	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	average pool, 1000-d fc, sof	
	3.6×10^9	3.8×10^9

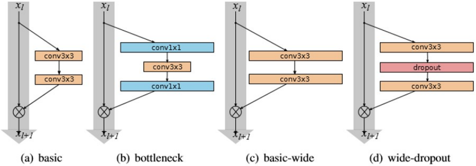
both designs have similar time complexity

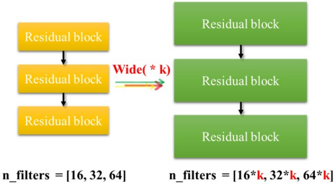
Residual Network - Identity mapping



Wide Residual Network - Overview

- Paper: **Wide Residual Networks**



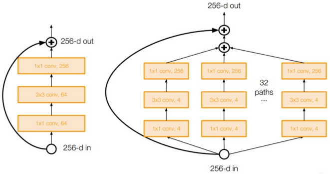


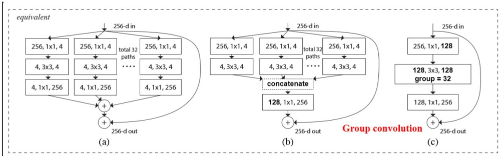
	depth-k	# params	CIFAR-10	CIFAR-100
pre-act-ResNet[13]	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.9M	4.53	21.18
	16-8	11.0M	4.27	20.43
	28-10	36.5M	**4.00**	**19.25**

depth	k	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.00	19.25	-
28	10	✓	**3.89**	**18.85**	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

ResNeXt Network - Overview

- Paper: **Aggregated Residual Transformations for Deep Neural Networks**



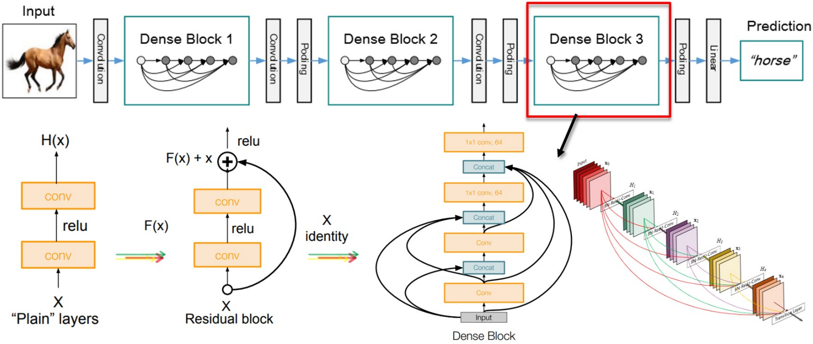


	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [14]	22.0	6.0	-	-
ResNet-200 [15]	21.7	5.8	20.1	4.8
Inception-v3 [39]	-	-	21.2	5.6
Inception-v4 [37]	-	-	20.0	5.0
Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (**64** × **4d**)	20.4	5.3	**19.1**	**4.4**

	# params	CIFAR-10	CIFAR-100
Wide ResNet [43]	36.5M	4.17	20.50
ResNeXt-29, 8×64d	34.4M	3.65	17.77
ResNeXt-29, 16×64d	68.1M	**3.58**	**17.31**

DenseNet - Overview

- Paper: **Densely Connected Convolutional Networks**



DenseNet - Detail

- DenseNet architectures for ImageNet

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

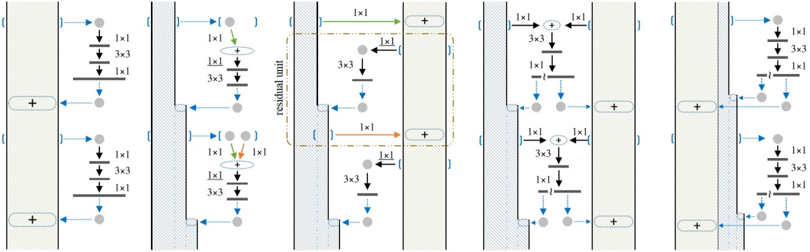
DenseNet - Results

- Error rates on CIFAR and SVHN datasets

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [31]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [33]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [41]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Dual Path Networks - Overview

- Paper: **Dual Path Networks**
- Github: **cypw/DPNs**
- ResNet + DenseNet $\rightrightarrows$ Dual Path Networks



(a) Residual Network (b) Densely Connected Network (c) Densely Connected Network (with shared connections) (d) Dual Path Architecture (e) DPN

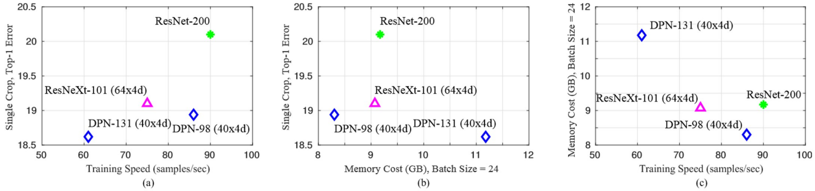
Dual Path Networks - Detail

stage	output	DenseNet-161 (k=48)	ResNeXt-101 (32×4d)	ResNeXt-101 (64×4d)	DPN-92 (32×3d)	DPN-98 (40×4d)
conv1	112x112	7 × 7, 96, stride 2	7 × 7, 64, stride 2	7 × 7, 64, stride 2	7 × 7, 64, stride 2	7 × 7, 96, stride 2
conv2	56x56	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 192 \\ 3\times3, 48 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, G{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, G{=}64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 96 \\ 3\times3, 96, G{=}32 \\ 1\times1, 256\ (+16) \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 160 \\ 3\times3, 160, G{=}40 \\ 1\times1, 256\ (+16) \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 192 \\ 3\times3, 48 \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, G{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, G{=}64 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 192 \\ 3\times3, 192, G{=}32 \\ 1\times1, 512\ (+32) \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 320 \\ 3\times3, 320, G{=}40 \\ 1\times1, 512\ (+32) \end{bmatrix} \times 6$
conv4	14×14	$\begin{bmatrix} 1\times1, 192 \\ 3\times3, 48 \end{bmatrix} \times 36$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, G{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, G{=}64 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 384 \\ 3\times3, 384, G{=}32 \\ 1\times1, 1024\ (+24) \end{bmatrix} \times 20$	$\begin{bmatrix} 1\times1, 640 \\ 3\times3, 640, G{=}40 \\ 1\times1, 1024\ (+32) \end{bmatrix} \times 20$
conv5	7×7	$\begin{bmatrix} 1\times1, 192 \\ 3\times3, 48 \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, G{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 2048 \\ 3\times3, 2048, G{=}64 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 768 \\ 3\times3, 768, G{=}32 \\ 1\times1, 2048\ (+128) \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1280 \\ 3\times3, 1280, G{=}40 \\ 1\times1, 2048\ (+128) \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params		$\mathbf{28.9} \times 10^6$	$\mathbf{44.3} \times 10^6$	$\mathbf{83.7} \times 10^6$	$\mathbf{37.8} \times 10^6$	$\mathbf{61.7} \times 10^6$
FLOPs		$\mathbf{7.7} \times 10^9$	$\mathbf{8.0} \times 10^9$	$\mathbf{15.5} \times 10^9$	$\mathbf{6.5} \times 10^9$	$\mathbf{11.7} \times 10^9$

Dual Path Networks - Results

Method	Model Size	GFLOPs	x224 top-1	x224 top-5	x320 / x299 top-1	x320 / x299 top-5
DenseNet-161(k=48) [8]	111 MB	7.7	22.2	–	–	–
ResNet-101* [5]	170 MB	7.8	22.0	6.0	–	–
ResNeXt-101 (32 × 4d) [21]	170 MB	8.0	21.2	5.6	–	–
DPN-92 (32 × 3d)	**145 MB**	**6.5**	**20.7**	**5.4**	**19.3**	**4.7**
ResNet-200 [6]	247 MB	15.0	21.7	5.8	20.1	4.8
Inception-resnet-v2 [20]	227 MB	–	–	–	19.9	4.9
ResNeXt-101 (64 × 4d) [21]	320 MB	15.5	20.4	5.3	19.1	4.4
DPN-98 (40 × 4d)	**236 MB**	**11.7**	**20.2**	**5.2**	**18.9**	**4.4**
Very deep Inception-resnet-v2 [23]	531 MB	–	–	–	19.10	4.48
Very Deep PolyNet [23]	365 MB	–	–	–	18.71	4.25
DPN-131 (40 × 4d)	304 MB	16.0	19.93	5.12	18.62	4.23
DPN-131 (40 × 4d) †	**304 MB**	**16.0**	**19.93**	**5.12**	**18.55**	**4.16**

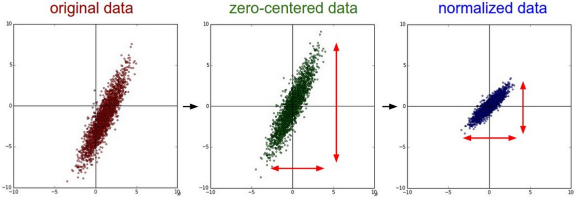
Method	Model Size	top-1 acc.	top-5 acc.
AlexNet [24]	223 MB	53.17	82.89
GoogLeNet [24]	44 MB	53.63	83.88
VGG-16 [24]	518 MB	55.24	84.91
ResNet-152 [24]	226 MB	54.74	85.08
ResNeXt-101 [3]	165 MB	56.21	86.25
CRU-Net-116 [3]	163 MB	56.60	86.55
DPN-92 (32 × 3d)	**138 MB**	**56.84**	**86.69**



Training Tricks

- Pre-Processing
- Data Augmentation
- Initializations
- Regularizations
- Fine-tune

Pre-Processing

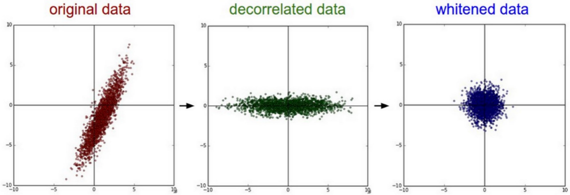



```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

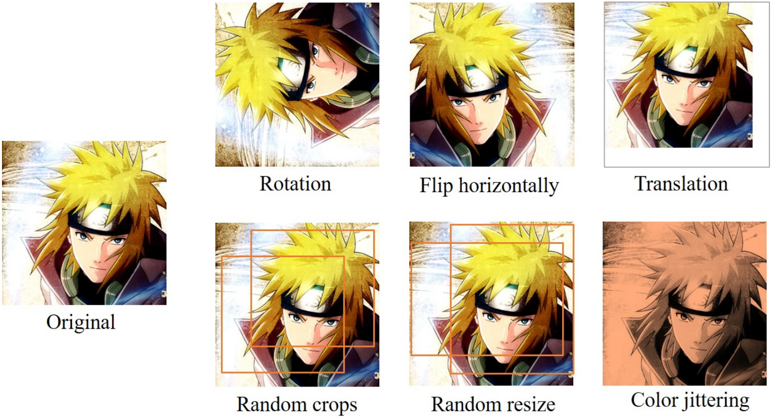
- Mean / Std

PCA Whitening




```
Xwhite = Xrot / np.sqrt(S + 1e-5)
```

Data Augmentation

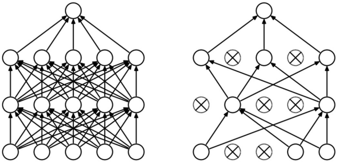


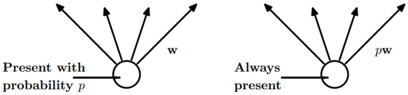
Initializations

- Random Normal
- Random Uniform
- Lecun Uniform
- Glorot Normal
  - It draws samples from a uniform distribution within [-limit, limit] where limit is sqrt(6 / (fan_in + fan_out)) where fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor.
- He Normal - Current Recommendation
  - It draws samples from a truncated normal distribution centered on 0 with stddev = sqrt(2 / fan_in) where fan_in is the number of input units in the weight tensor.

Regularizations

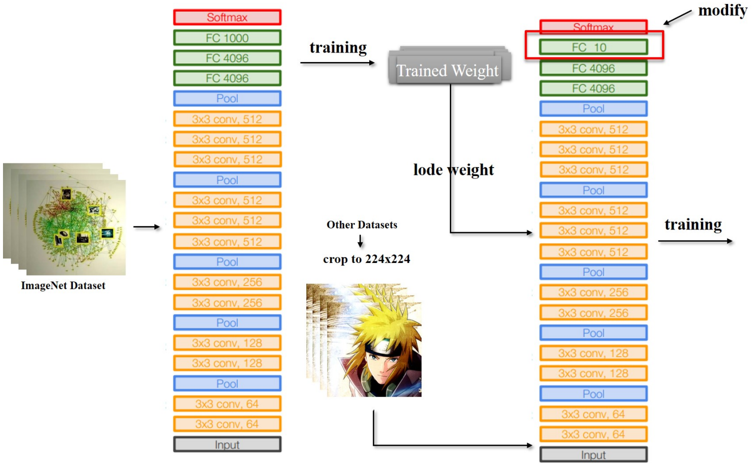
- L1 regularization
  - $C = C_0 + \frac{\lambda}{n} \sum_w |w|$
- L2 regularization(Weight Decay)
  - $C = C_0 + \frac{\lambda}{2n} \sum_w w^2$
- Dropout


(a) Standard Neural Net (b) After applying dropout.

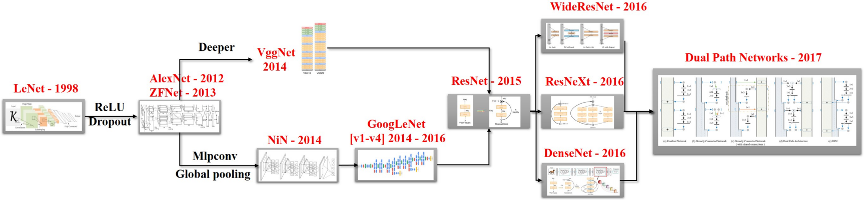


(a) At training time (b) At test time

Fine-tune


Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
ImageNet Dataset
training
Trained Weight
lode weight
Other Datasets
crop to 224x224
modify
Softmax
FC 10
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
training

Summary of CNN



CNN model

What is the next model?