Machine Intelligence & Deep Learning Workshop

Raymond Ptucha, Majid Rabbani, Mark Smith

The Kate Gleason COLLEGE OF ENGINEERING

Convolution Neural Networks

Raymond Ptucha
June 27-29, 2018
Rochester Institute of Technology
www.rit.edu/kgcoe/cqas/machinelearning

Fair Use Agreement

This agreement covers the use of all slides in this document, please read carefully.

- You may freely use these slides for personal use, if:
  - My name (R. Ptucha) appears on each slide.
- You may freely use these slides externally, if:
  - You send me an email telling me the conference/venue/company name in advance, and which slides you wish to use.
  - You receive a positive confirmation email back from me.
  - My name (R. Ptucha) appears on each slide you use.

(c) Raymond Ptucha, rwpeec@rit.edu

Agenda

- Wed, June 27
  - 9-10:30am — Regression and Classification
  - 10:30-10:45pm — Break
  - 10:45-12:15pm — Boosting and SVM
  - 12:15-1:30pm — Lunch
  - 1:30-3:30pm — Neural Networks and Dimensionality Reduction
  - 3:30-5pm — Hands-on Python and Machine Learning
- Thur, June 28
  - 9-10:30am — Introduction to deep learning
  - 10:30-10:45pm — Break
  - 10:45-12:15pm — **Convolutional Neural Networks**
  - 12:15-1:30pm — Lunch
  - 1:30-3:30pm — Region and pixel-level convolutions
  - 3:30-5pm — Hands-on CNNs
- Fri, June 29
  - 9-10:30am — Recurrent neural networks
  - 10:30-10:45pm — Break
  - 10:45-12:15pm — Language and Vision
  - 12:15-1:30pm — Lunch
  - 1:30-3:30pm — Graph convolutional neural networks; Generative adversarial networks
  - 3:30-5pm — Hands-on regional CNNs, RNNs

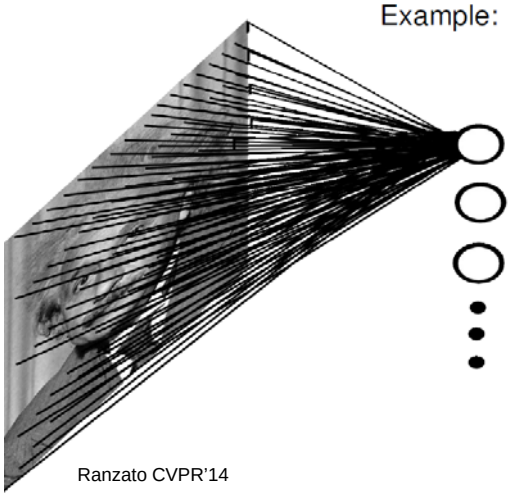
Two Most Important Deep Learning Fields

- Convolutional Neural Networks (CNN)
  - Examine high dimensional input, learn features and classifier simultaneously
- Recurrent Neural Networks (RNN)
  - Learn temporal signals, remember both short and long sequences

Two Most Important Deep Learning Fields

- Convolutional Neural Networks (CNN)
  - Examine high dimensional input, learn features and classifier simultaneously
- Recurrent Neural Networks (RNN)
  - Learn temporal signals, remember both short and long sequences

Fully Connected Layers?

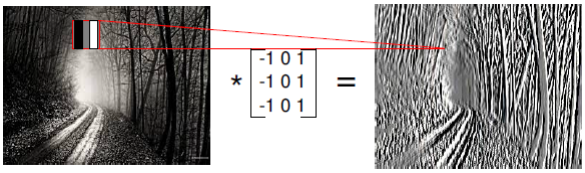


Ranzato CVPR'14

Example:

- 200×200 pixel image.
- 40K input fully connected to 40K hidden (or output) layer.
- 1.6 billion weights!
- Generally don't have enough training samples to learn that many weights.

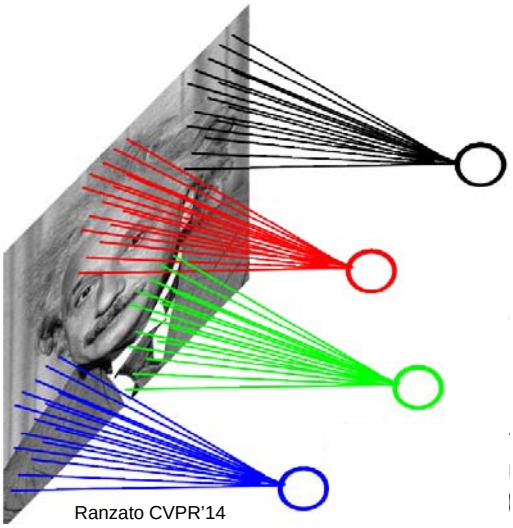
Convolution Filter



Ranzato CVPR'14

- Convolution filters apply a transform to an image.
- The above filter detects vertical edges.

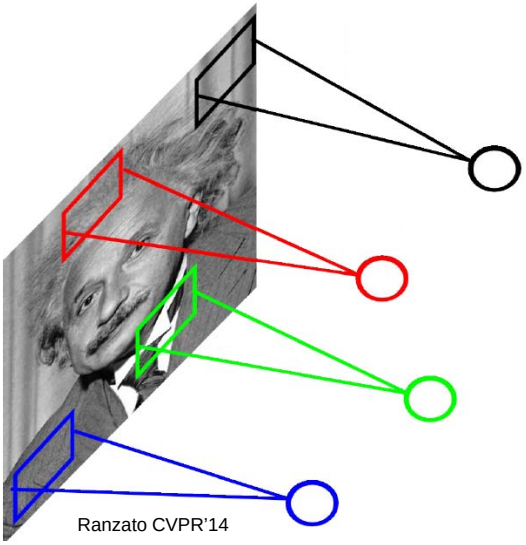
Locally Connected Layer



Ranzato CVPR'14

- 200×200 pixel image.
- 40K input.
- Four 10×10 filters, each fully connected
- 40K×10×10×4=16M weights….getting better!

Locally Connected Layer

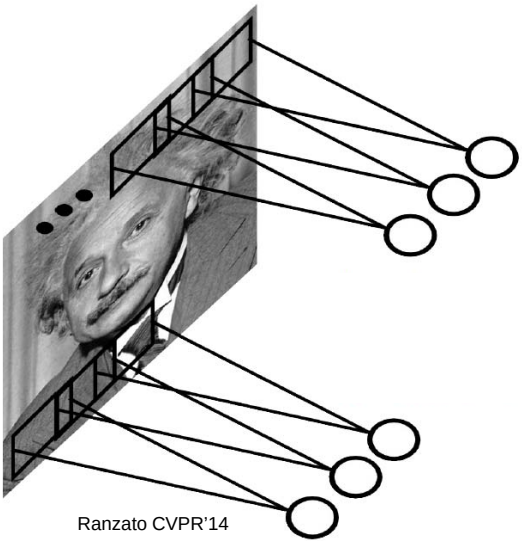


Ranzato CVPR'14

- 200×200 pixel image.
- 40K input.
- Four 10×10 filters, each fully connected
- 40K×10×10×4=16M weights….getting better!

- Can we formulate so each filter has similar statistics across all locations?

Convolution Layer

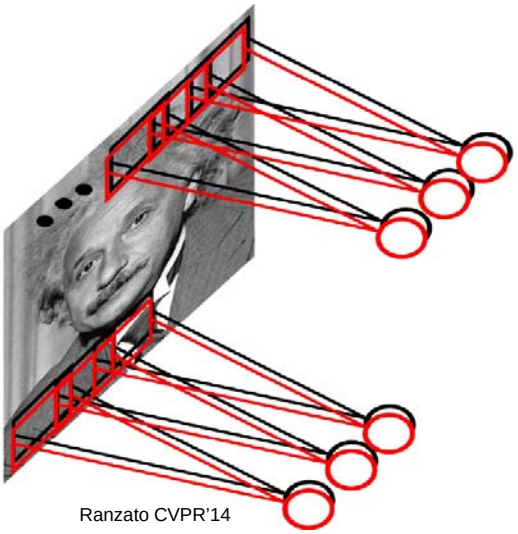


Ranzato CVPR'14

- 200×200 pixel image.
- 40K input.
- Four 10×10 filters, each fully connected
- 40K×10×10×4=16M weights….getting better!

- Require each filter has same statistics across all locations.
- Learn filters.

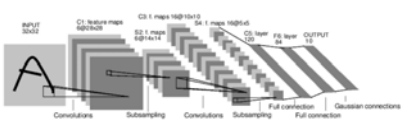
Convolution Layer



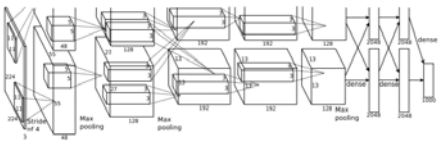
Ranzato CVPR'14

- 200×200 pixel image.
- 40K input.
- Four 10×10 filters, each fully connected
- 40K×10×10×4=16M weights….getting better!
- Require each filter has same statistics across all locations.
- Learn filters.
- To learn four filters we have 4×10×10=400 parameters- great!

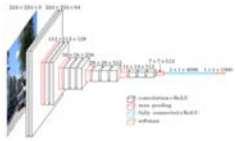
Many Flavors of CNNs…



LeNet-5, LeCun 1989



AlexNet, Krizhevsky 2012



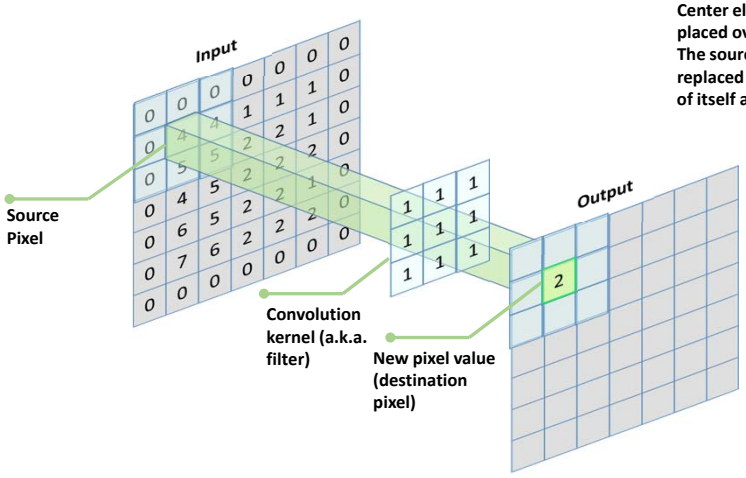
VGGNet, Simonyan 2014

GoogLeNet (Inception), Szegedy 2014

ResNet, He 2015

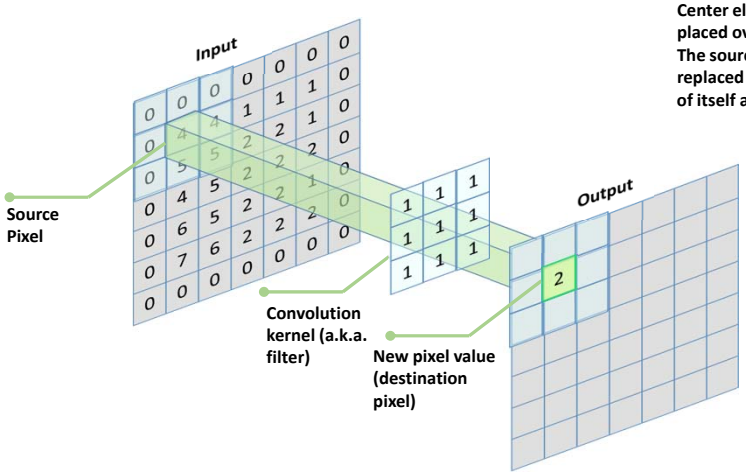
DenseNet, Huang 2017

Convolution



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

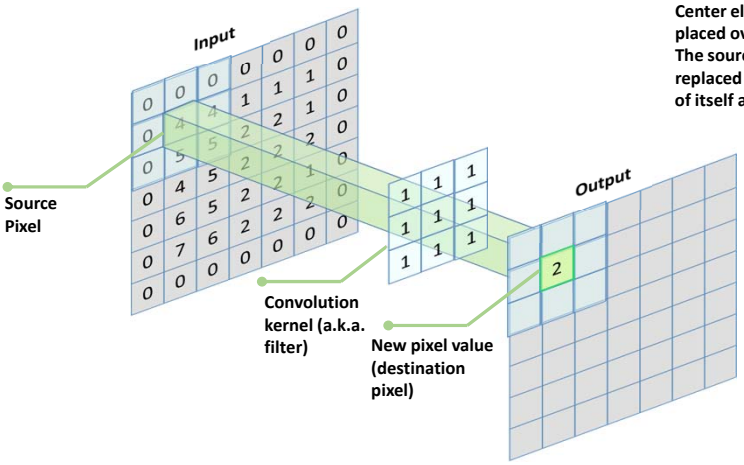
Convolution



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+1*0+1*4+1*4+1*0+1*5+1*5)=2$$

Convolution

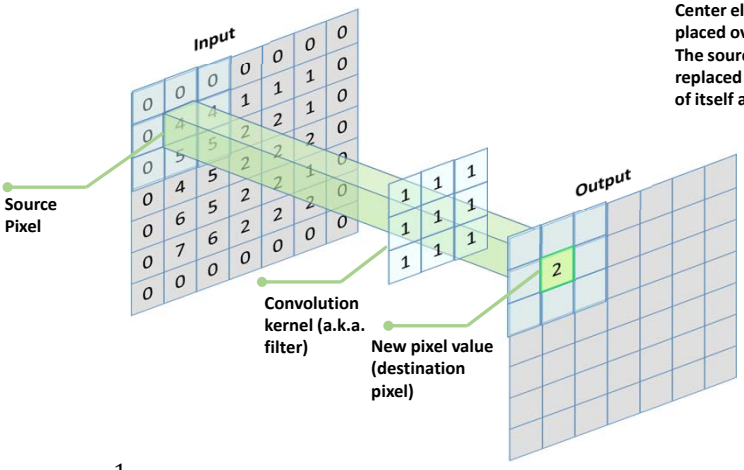


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+\underbrace{1*0+1*4+1*4}_{\text{Middle}}+1*0+1*5+1*5)=2$$

15

Convolution

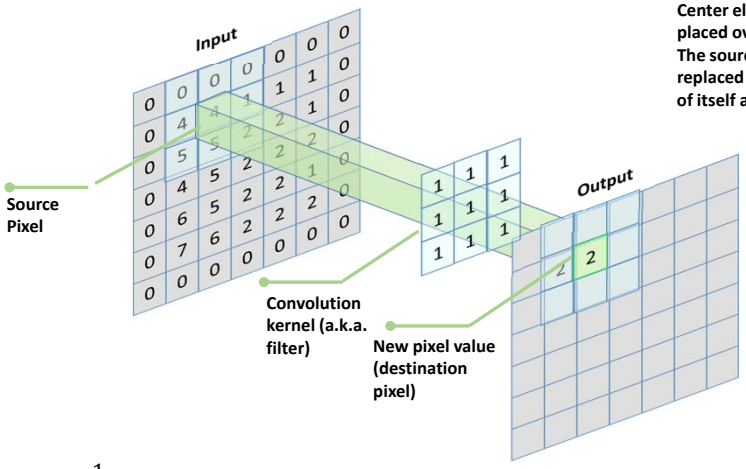


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+\underbrace{1*0+1*4+1*4}_{\text{Middle}}+\underbrace{1*0+1*5+1*5}_{\text{Bottom}})=2$$

16

Convolution

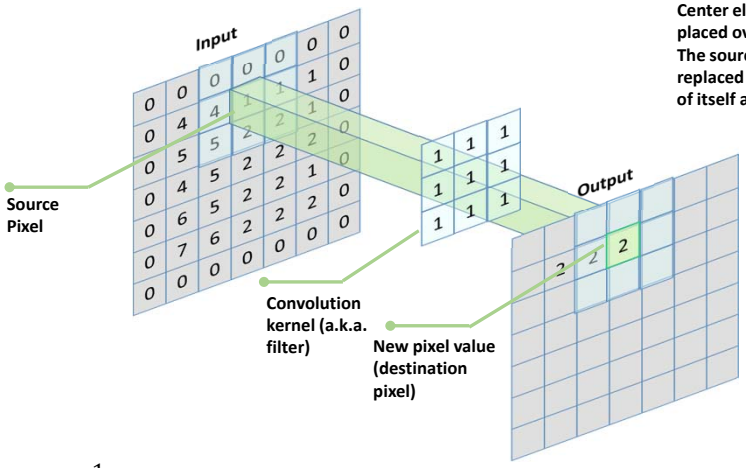


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+\underbrace{1*4+1*4+1*1}_{\text{Middle}}+\underbrace{1*5+1*5+1*2}_{\text{Bottom}})=2.3\approx 2$$

17

Convolution

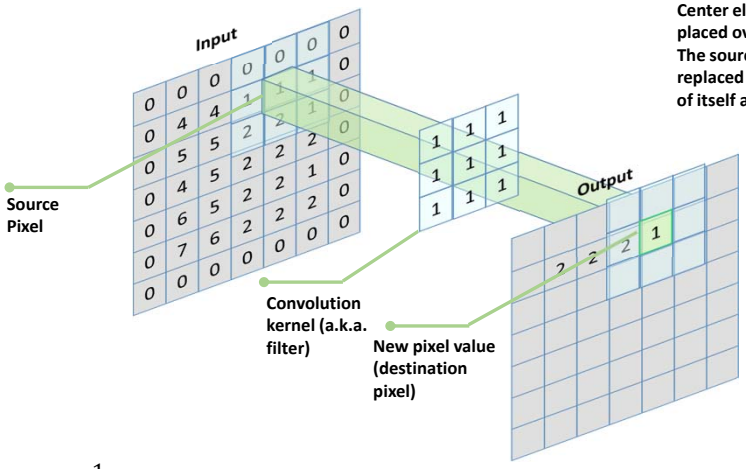


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+\underbrace{1*4+1*1+1*1}_{\text{Middle}}+\underbrace{1*5+1*2+1*2}_{\text{Bottom}})=1.7\approx 2$$

18

Convolution

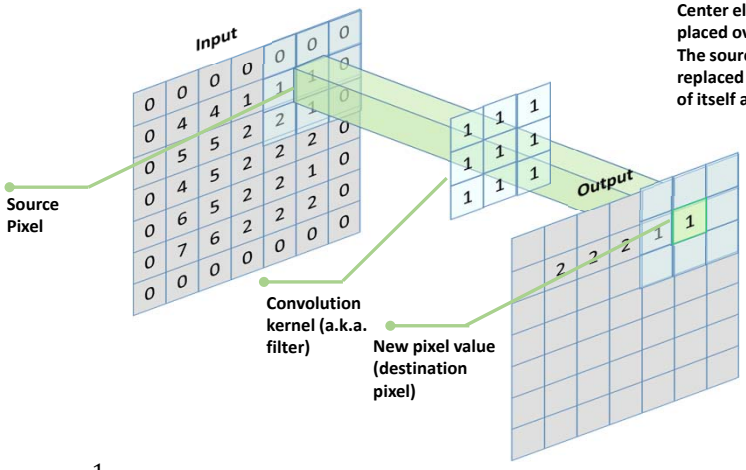


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+\underbrace{1*1+1*1+1*1}_{\text{Middle}}+\underbrace{1*2+1*2+1*1}_{\text{Bottom}})=0.9\approx 1$$

19

Convolution

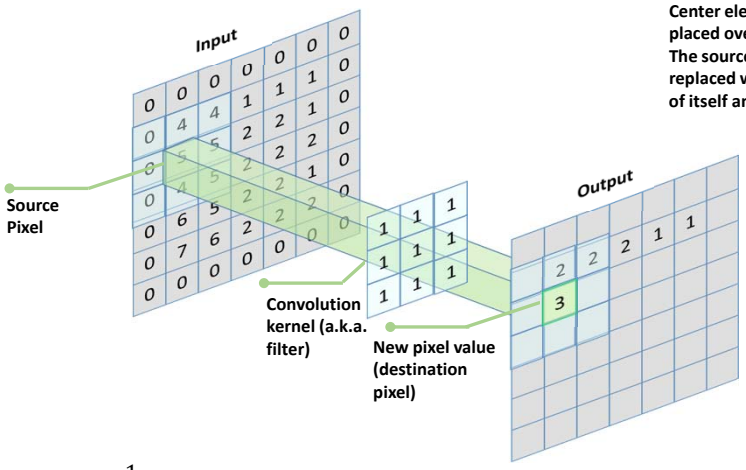


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*0+1*0}_{\text{Top}}+\underbrace{1*1+1*1+1*0}_{\text{Middle}}+\underbrace{1*2+1*1+1*0}_{\text{Bottom}})=0.6\approx 1$$

20

Convolution

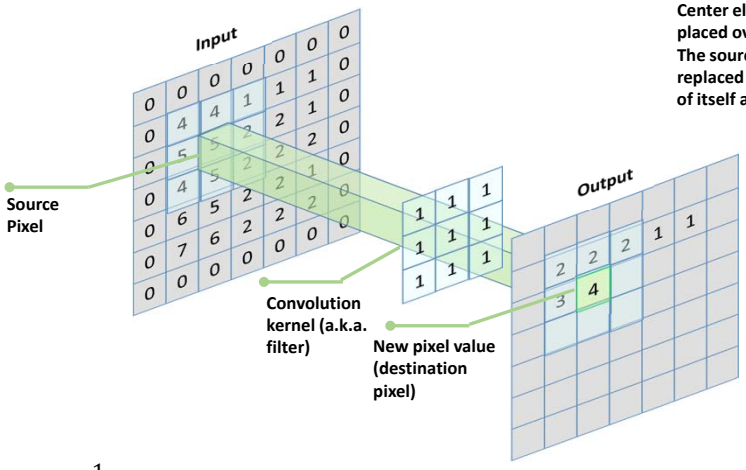


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*0+1*4+1*4}_{\text{Top}}+\underbrace{1*0+1*5+1*5}_{\text{Middle}}+\underbrace{1*0+1*4+1*5}_{\text{Bottom}})=3$$

21

Convolution

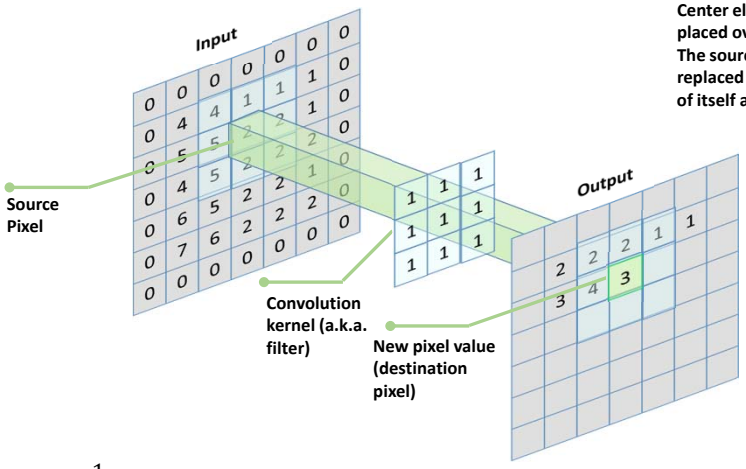


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*4+1*4+1*1}_{\text{Top}}+\underbrace{1*5+1*5+1*2}_{\text{Middle}}+\underbrace{1*4+1*5+1*2}_{\text{Bottom}})=3.6\approx 4$$

22

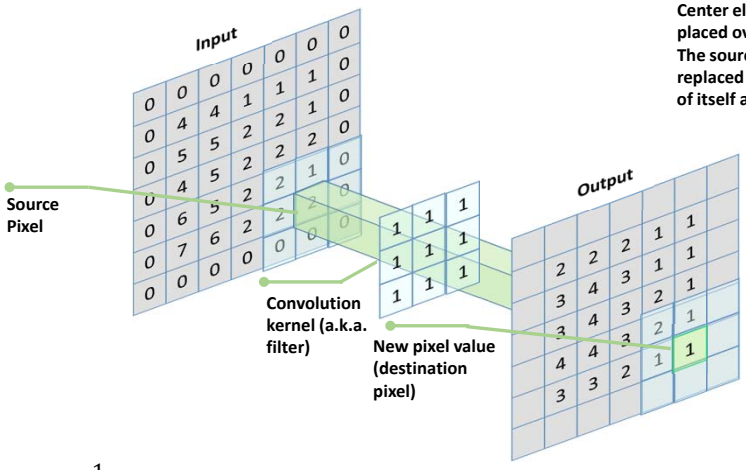
Convolution



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*4+1*1+1*1}_{\text{Top}}+\underbrace{1*5+1*2+1*2}_{\text{Middle}}+\underbrace{1*5+1*2+1*2}_{\text{Bottom}})=2.7\approx 3$$

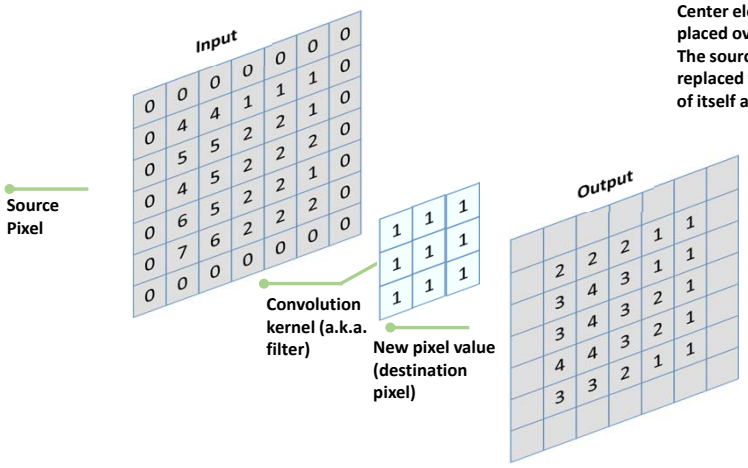
Convolution



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\frac{1}{9}(\underbrace{1*2+1*1+1*0}_{\text{Top}}+\underbrace{1*2+1*2+1*0}_{\text{Middle}}+\underbrace{1*0+1*0+1*0}_{\text{Bottom}})=0.8\approx 1$$

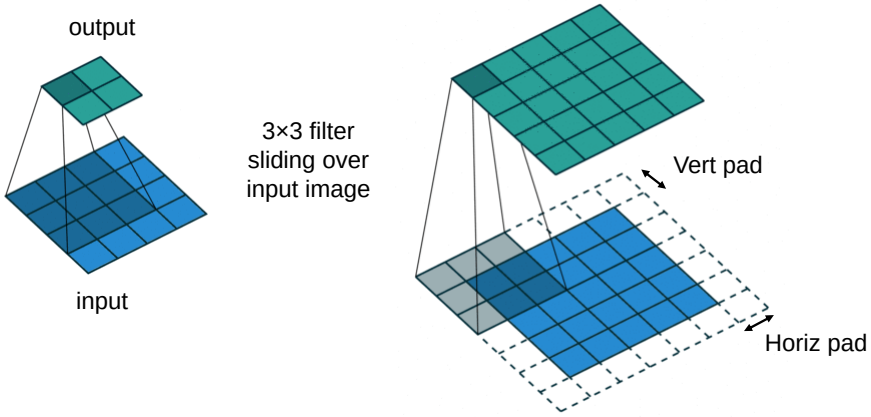
Convolution



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

Image Convolution

By padding (filterWidth-1)/2, output image size matches input image size



https://github.com/vdumoulin/conv_arithmetic

Filtering

Image is $I_w \times I_h$
Filter is $F_w \times F_h$

$$OutputImageSize = \frac{(InputImageSize - FilterSize)}{stride} + 1$$

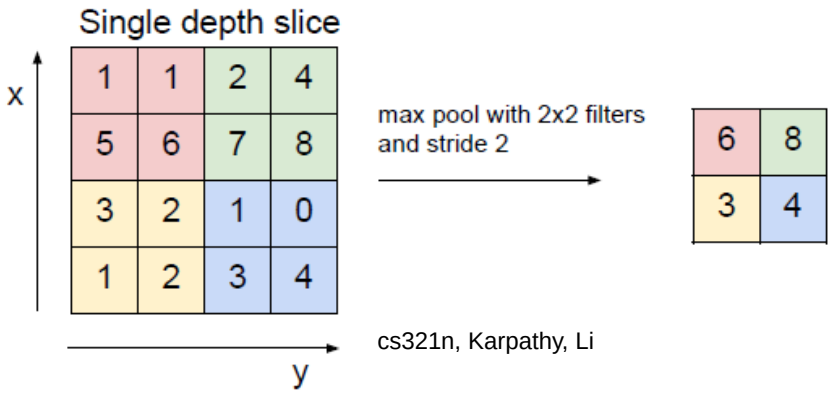
e.g. InputImageSize=32, FilterSize=7
Stride 1 → (32-7)/1 + 1 = 26
Stride 5 → (32-7)/5 + 1 = 6

Very common to use square images and filters, but certainly don't have to.

Stride 2 → (32-7)/2 + 1 = ??

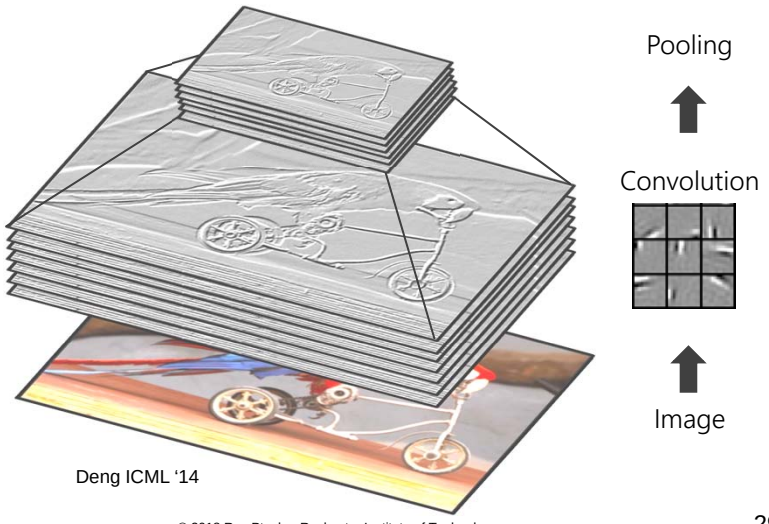
FilterSize=5, Stride 3 → (32-5)/3 + 1 = 10

Max Pooling- Reducing the Size of an Image



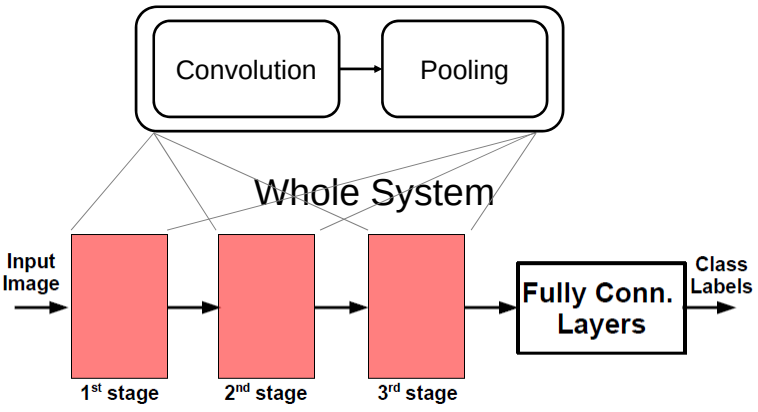
cs321n, Karpathy, Li

Convolution Neural Network (CNN) Building Block



29

Putting it All Together

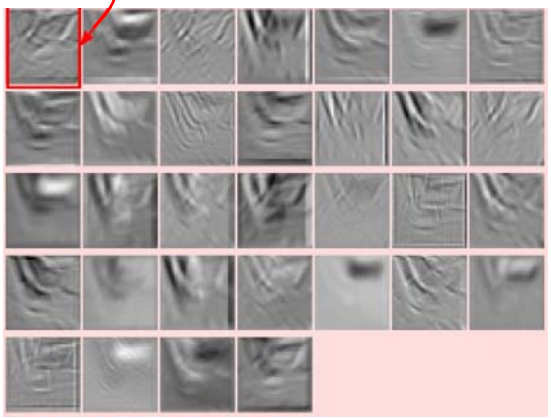


30

Learning Filters

32 Learned Filters, each 5×5

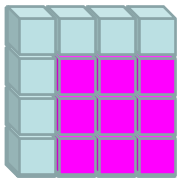
32 Filtered images, each is 28×28



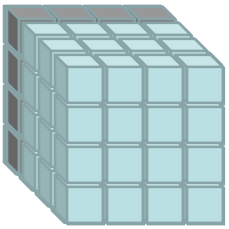
Input image
28×28

Use zero padding

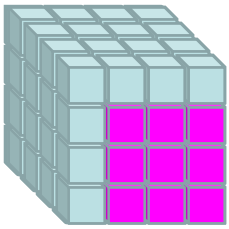
Filters



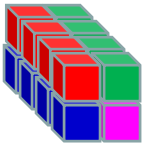
3×3 filter



3×3 filter



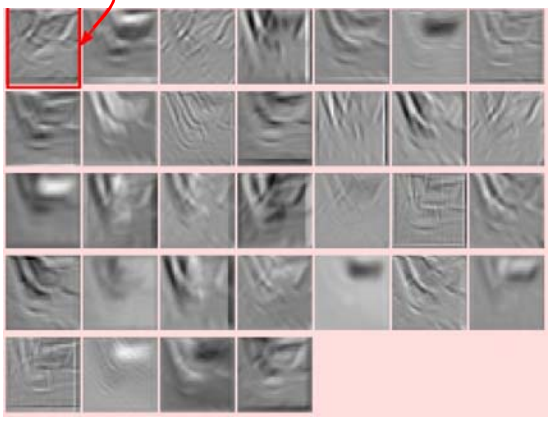
3×3×4 filter



Learning Filters

32 Learned Filters, each 5×5×3

32 Filtered images, each is 28×28×1



Input image
28×28×3

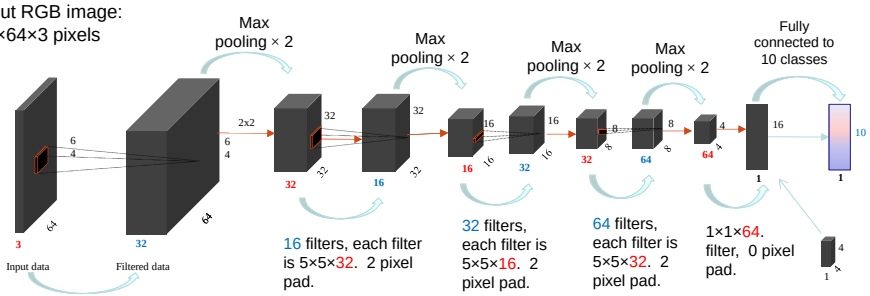
Use zero padding

CNN Architecture

(Not so) Toy Example

Output:
prediction of 1 of 10 categories

Input RGB image:
64×64×3 pixels



32 filters, each filter is 5×5×3. 2 pixel pad added to top/bot/left/right so filtered image is same dimension as input image.

16 filters, each filter is 5×5×32. 2 pixel pad.

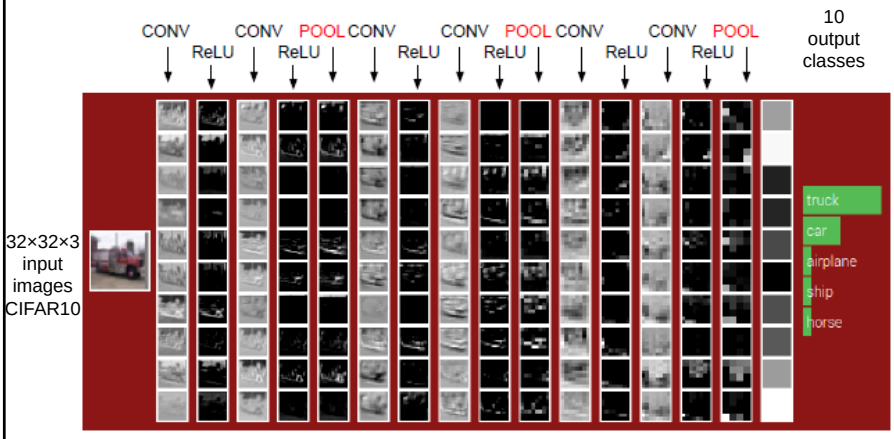
32 filters, each filter is 5×5×16. 2 pixel pad.

64 filters, each filter is 5×5×32. 2 pixel pad.

1×1×64. filter, 0 pixel pad.

4×4 converted to 16 element vector

CNN Example

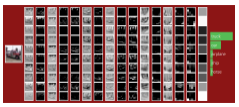


cs231n, Li, Karpathy, Johnson

Note: Two Conv's between each pool…

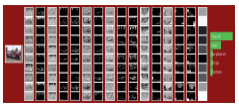
CNN Example

- Input [32x32x3]
- CONV with ten 3x3 filters, stride 1, pad 1:
  - Parameters: (3*3*3)*10 + 10 = 280
  - Memory: 32*32*10
- CONV with ten 3x3 filters, stride 1, pad 1:
  - Parameters: (3*3*10)*10 + 10 = 910
  - Memory: 32*32*10
- Pool with 2x2 filters, stride 2:
  - Parameters: 0
  - Memory: 16*16*10



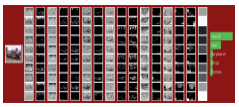
CNN Example

- CONV with ten 3x3 filters, stride 1, pad 1:
  - Parameters: (3*3*10)*10 + 10 = 910
  - Memory: 16*16*10
- CONV with twenty 3x3 filters, stride 1, pad 1:
  - Parameters: (3*3*10)*20 + 20 = 1820
  - Memory: 16*16*20
- Pool with 2x2 filters, stride 2:
  - Parameters: 0
  - Memory: 8*8*20



CNN Example

- CONV with ten 3x3 filters, stride 1, pad 1:
  - Parameters: (3*3*20)*10 + 10 = 1810
  - Memory: 8*8*10
- CONV with twenty 3x3 filters, stride 1, pad 1:
  - Parameters: (3*3*10)*20 + 20 = 1820
  - Memory: 8*8*20
- Pool with 2x2 filters, stride 2:
  - Parameters: 0
  - Memory: 4*4*20



CNN Example

- Fully connect (FC) 4x4x20 to 10 output classes
  - Parameters: (4*4*20)*10 + 10 = 3210
  - Memory: 10
  - Done!

Case Study

Case study: VGGNet / OxfordNet
(runner-up winner of ILSVRC 2014)
[Simonyan and Zisserman]

best model

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

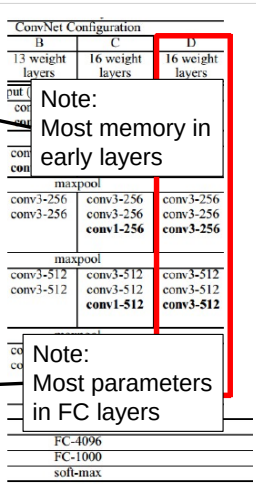
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

cs321n, Karpathy, Li

Case Study

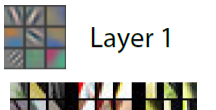
INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] 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

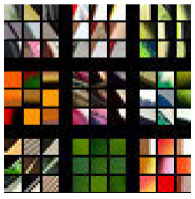


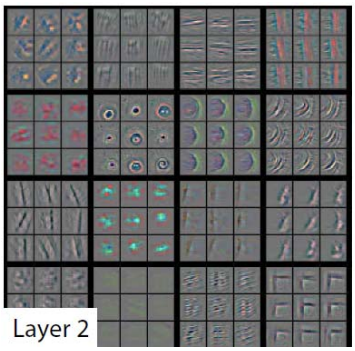
Note:
Most memory in early layers

Note:
Most parameters in FC layers

CNN Visualization

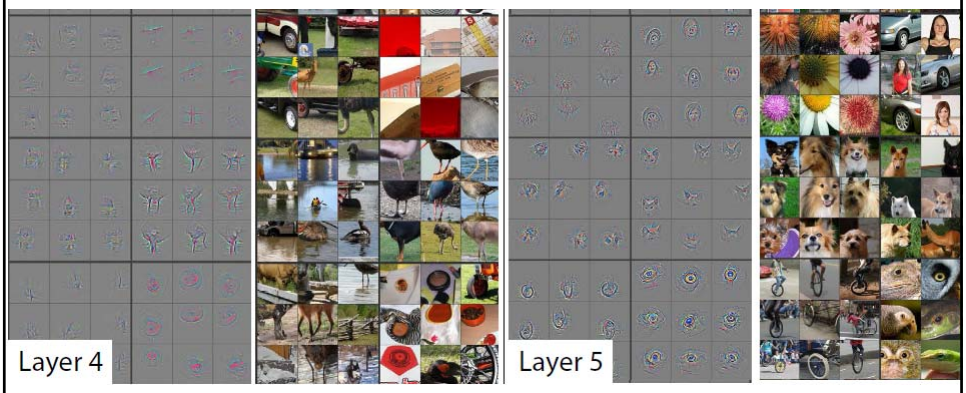

Layer 1




Layer 2

Zeiler, Fergus, 2014

CNN Visualization

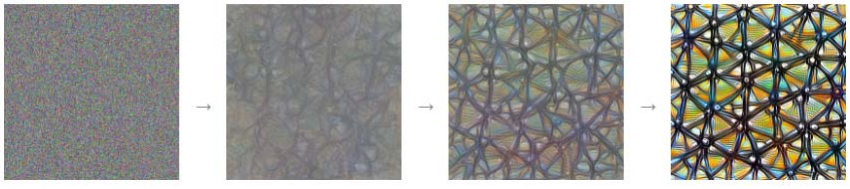


Zeiler, Fergus, 2014

CNN Visualization

- We can compute the partial derivative of input pixels with respect to a cost.
- Start with random noise, pretrained network, then iteratively tweak the input as we minimize our cost.

Google Inception v1: layer mixed4a, unit 11



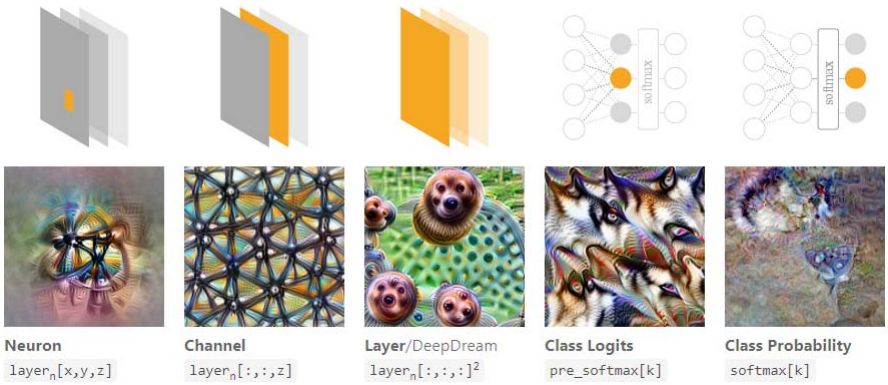
Initial → 4 iterations → 48 iterations → 2048 iterations

Olah, et al., "Feature Visualization", Distill, 2017.

CNN Visualization

- Can modify the objective to get different types of insight to what the CNN is responding to.

n layer index
x,y spatial position
z channel index
k class index



Olah, et al., "Feature Visualization", Distill, 2017.

CNN as Vector Representation

Typical CNN Architecture

Input Image

2D Plot of fc8 Feature Vector

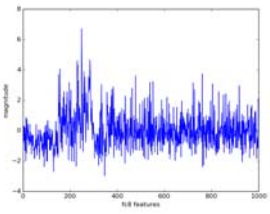
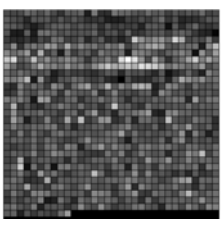


Image of fc8 Feature Vector

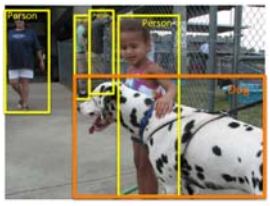


CNN as Vector Representation

- As it turns out, these fully connected layers are excellent descriptors of the input image!
- For example, you can pass images through a pre-trained CNN, then take the output from a FC layer as input to a SVM classifier. (image2vec)
- Images in this vector space generally have the property that similar images are close in this latent representation.

ImageNet

- Amazon Turk did bulk of labeling
- 14M labeled images
- 20K classes



Russakovsky et al., 2015

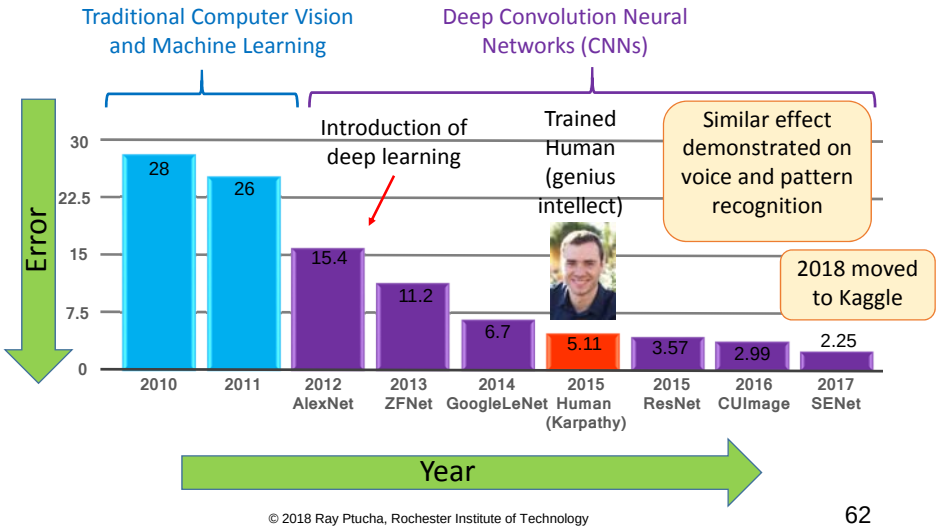
IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

- 1.2M images, 1000 categories
- Image classification, object localization, video detection

ImageNet: Examples of Hammer

Deep Learning- Surpassing The Visual Cortex's Object Detection and Recognition Capability

Top-5 error on ImageNet



GoogLeNet

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

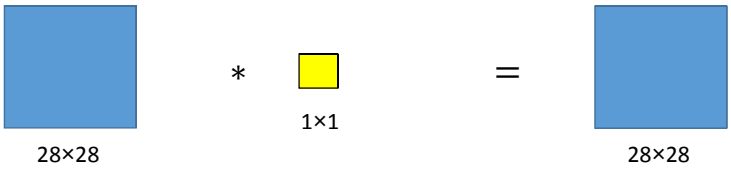
http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf

GoogLeNet, also called Inception model

- ImageNet 2014 winner
- Smaller number of parameters, but deeper net with better results!
  - Uses 12× fewer parameters than AlexNet
  - 16.4% for AlexNet, 6.67% for GoogLeNet
- Note: Hebbian principle is the tying of neurons together over time:
  - “Cells that fire together, wire together”
  - Any two cells or systems of cells that are repeatedly active at the same time will tend to become associated, so that activity in one facilitates activity in the other.

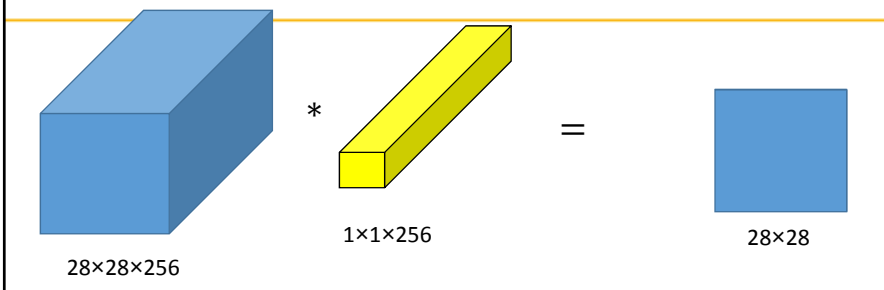
Using 1×1 Convolutions

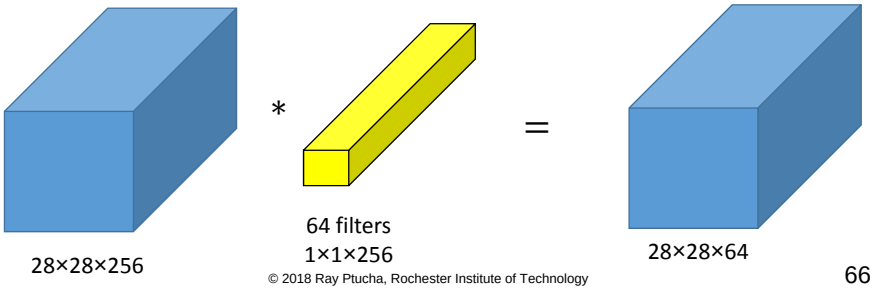
[Len et al. '13 Network in Network]



65

Using 1×1 Convolutions

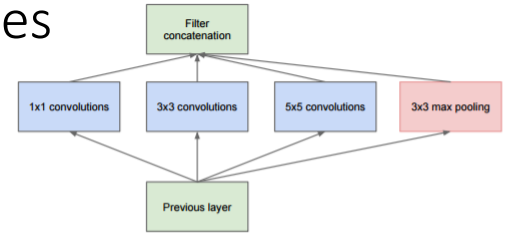




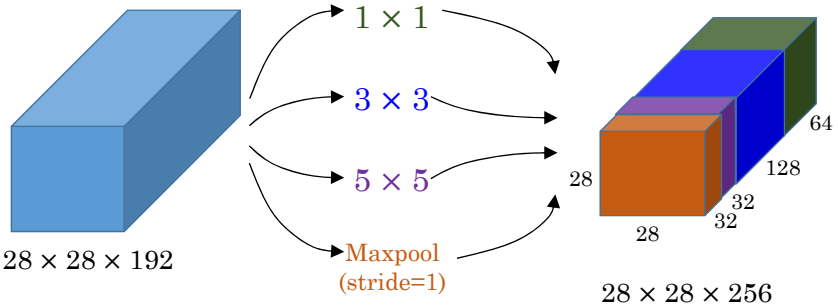
66

Inception Modules

- Basic inception building block.
- Let network figure out what filter size to use.

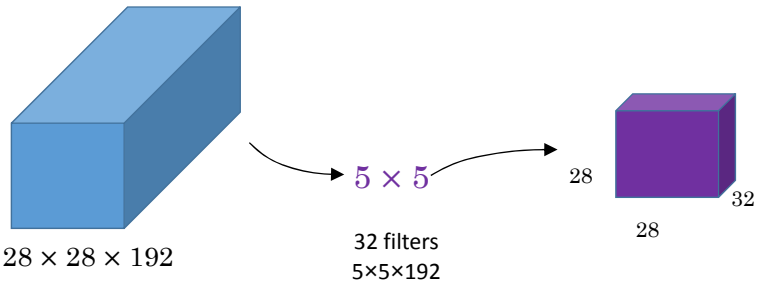


(a) Inception module, naïve version



Computational Cost

- For the 5x5 contribution, we have 32 filters, each 5×5×192.
- Parameters to be learned = (32)(5)(5)(192)=153,600
- Each pixel position in output has 5×5×192 mults and adds
- In total: (28×28×32)(5×5×192)=120M mults and adds!



Computational Cost with 1×1 bottleneck filter

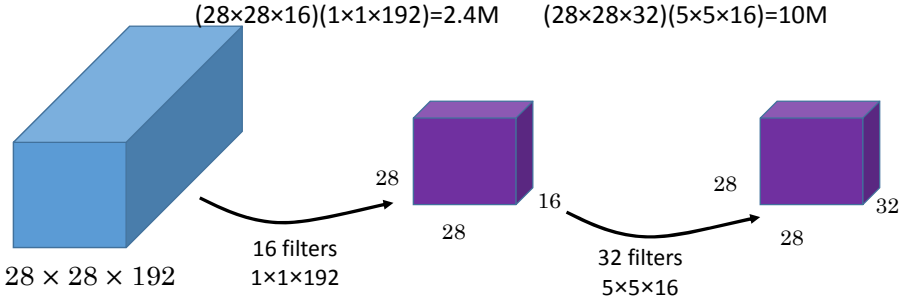
Total parameters: 15,872 (was 153,600)
Mults and adds: 12.4M (was 120M)

Parameters to be learned:
(16)(1)(1)(192)=3,072

Mults and adds:
(28×28×16)(1×1×192)=2.4M

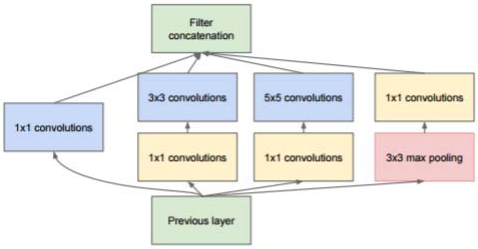
Parameters to be learned:
(32)(5)(5)(16)=12,800

Mults and adds:
(28×28×32)(5×5×16)=10M



Inception Modules

- 1×1 convolutions are used to compute reductions before expensive 3×3 or 5×5 convolutions.



(b) Inception module with dimensionality reduction

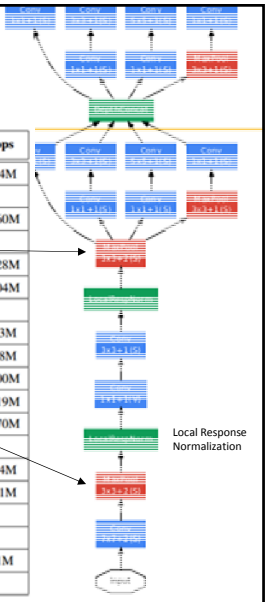
- Note: 1×1 convolution after 1×1 max pooling is to reduce the number of channels output from max pooling.
  - For example, the number of activation maps from 1×1, 3×3, 5×5, and max pooling branches might be 64, 128, 32, and 32, which can be concatenated to 256 channels.

GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

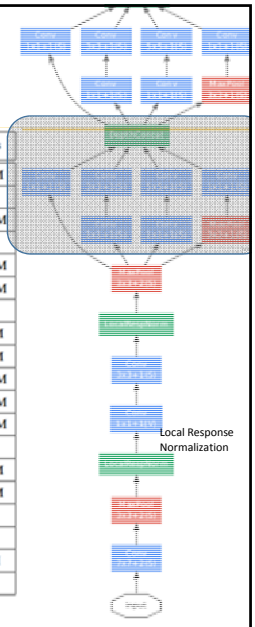
GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



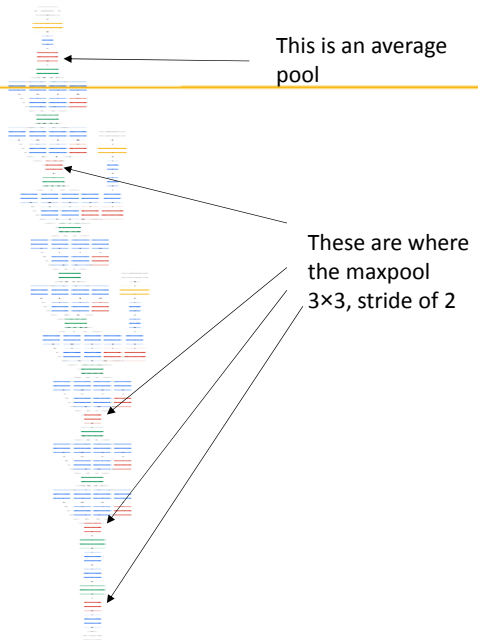
GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

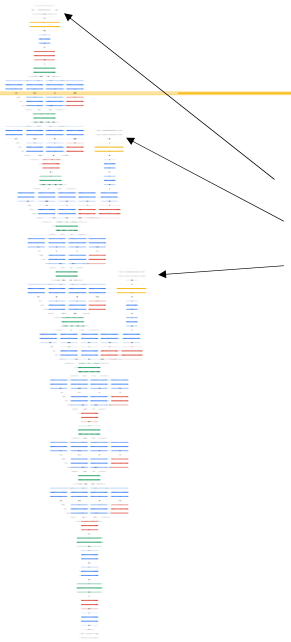


256 = 64+128+32+32

GoogLeNet



GoogLeNet



Because backprop has problems with deep nets, inception folks added intermediate softmax, so that a small net can be trained, then larger, then final

Testing Results

Ensemble of seven models, with 144 crops of each test image best results

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

ResNet

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

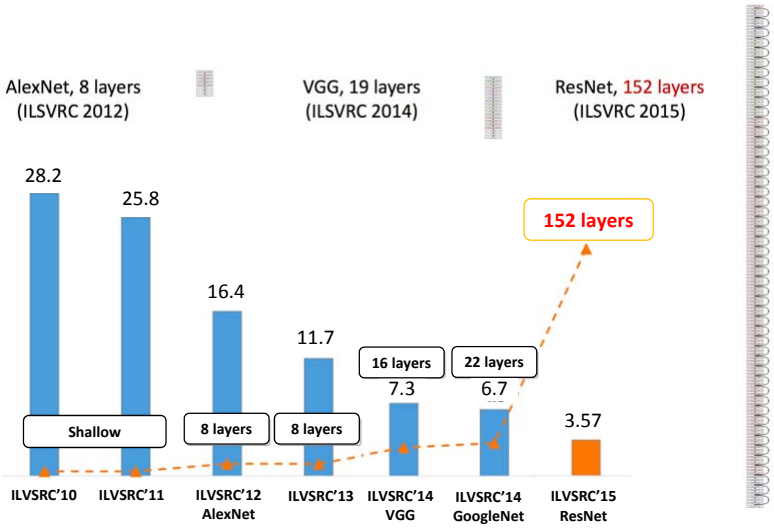
https://arxiv.org/pdf/1512.03385.pdf

ResNets @ ILSVRC & COCO 2015 Competitions

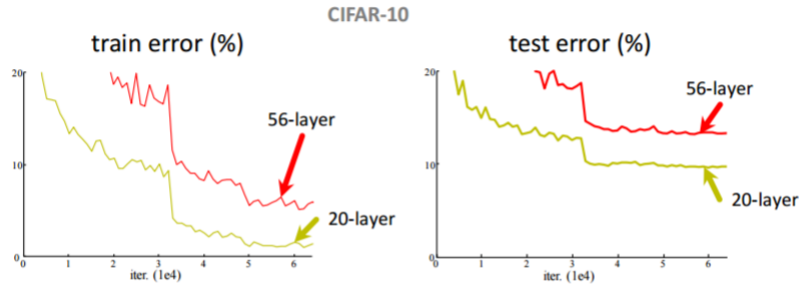
- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

Revolution of Depth

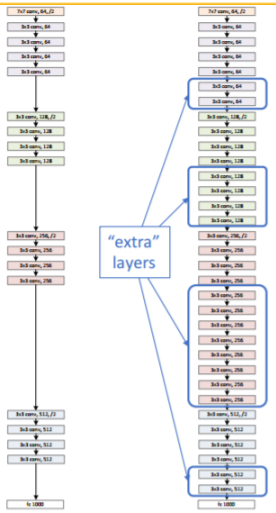


Simply stacking layers?



- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net

a shallower model (18 layers)



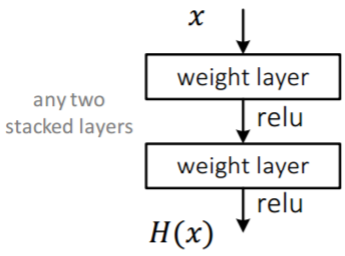
a deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

$H(x)$ is any desired mapping,

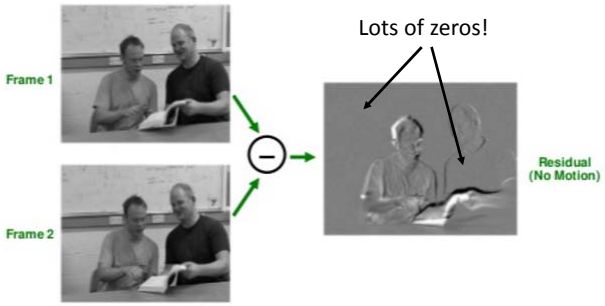
hope the 2 weight layers fit $H(x)$

- Plaint net



Residuals

- Well known in for example, image compression it is easier to compress residuals, than an original image



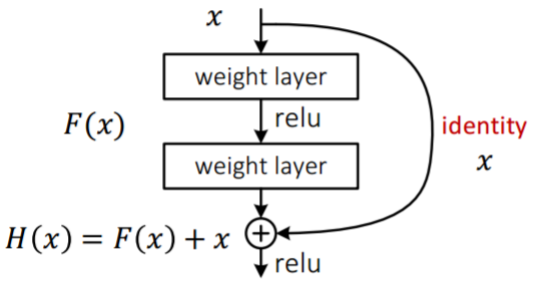
$H(x)$ is any desired mapping,

~~hope the 2 weight layers fit $H(x)$~~

hope the 2 weight layers fit $F(x)$

let $H(x) = F(x) + x$

- Residual net



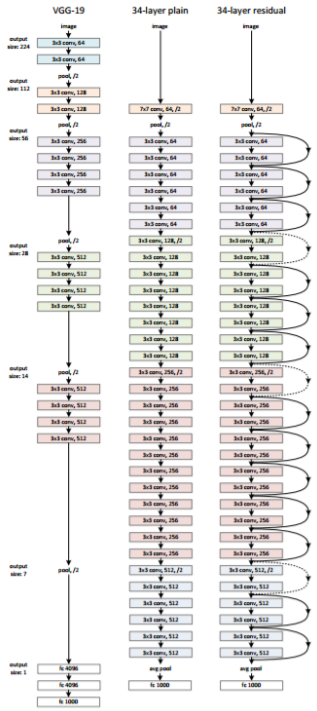
If optimal mapping closer to identity, easier to fit small fluctuations than entire function.

Easier to fit the residual, or difference *F(x)* between *x* and *H(x)*.

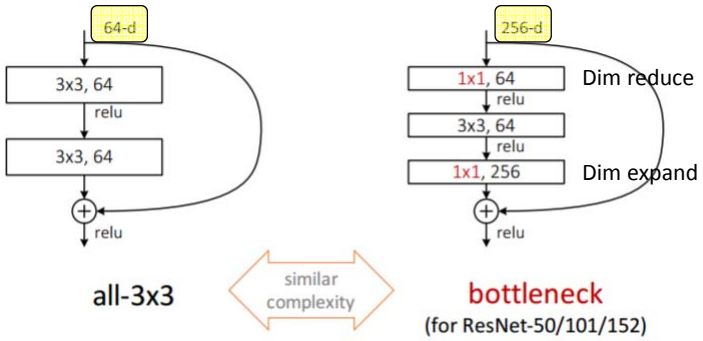
If Identity were optimal, all weights in F(x)=0.

Network Design

- Simple, VGG-style, mostly 3×3 filters
- No fully connected layers at end
- No dropout, but use batch normalization everywhere.
- Trained from scratch
- Augmentation of samples



Similar Complexity

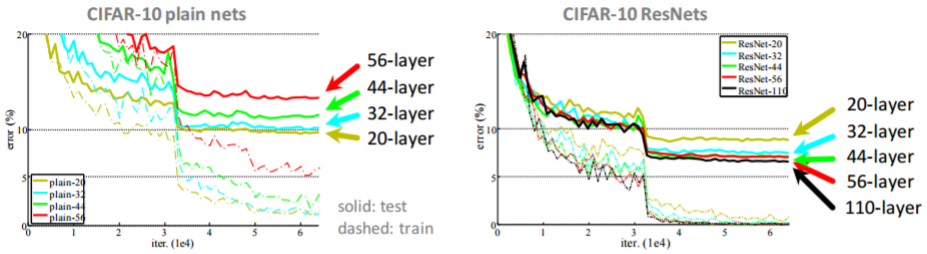


64x3x3x64 +
64x3x3x64 = 73.7K params

64x1x1x256 +
64x3x3x64 +
256x1x1x64 = 69.6K params

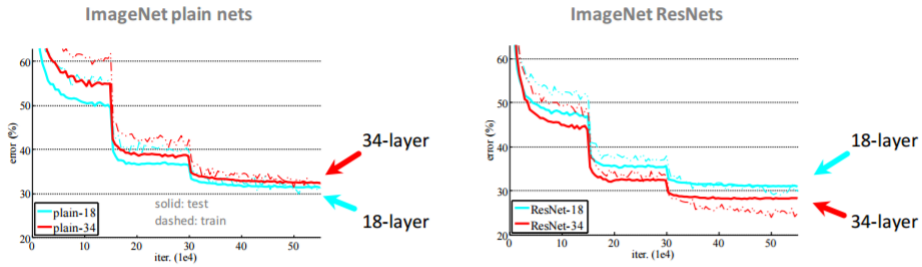
86

CIFAR-10 experiments



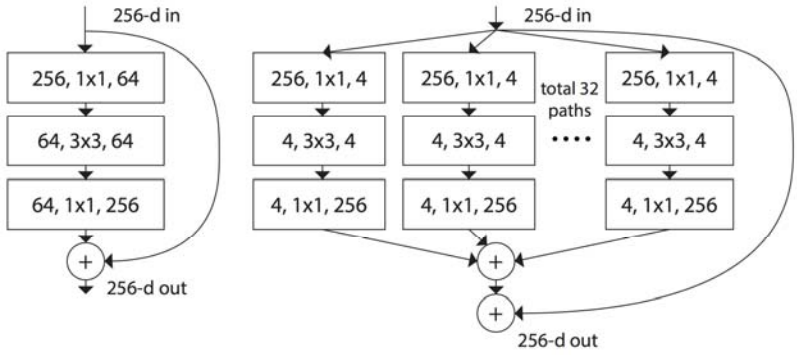
- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

ImageNet experiments



- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

ResNext

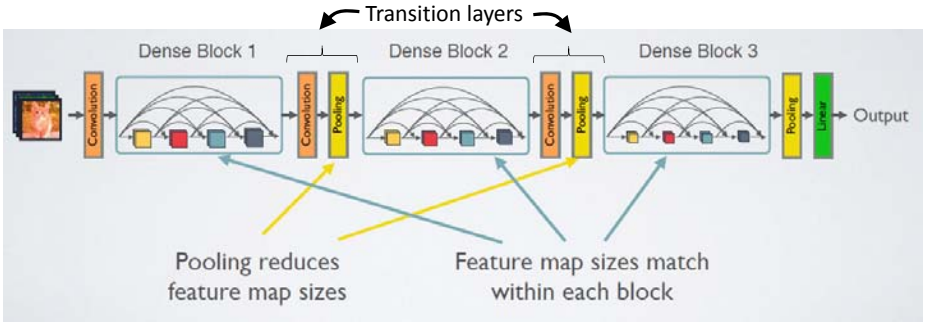


ResNet ResNeXt

Saining Xie, Ross Girshick, Piotr Dollár, Zhouwen Tu, Kaiming He. "Aggregated Residual Transforms for Deep Neural Networks". CVPR 2017.

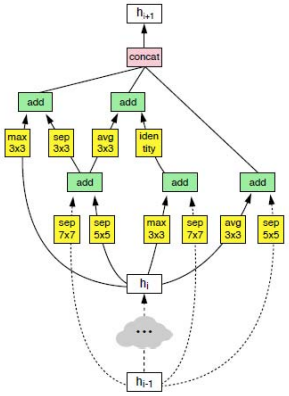
DenseNet

- Dense connectivity in each block
- Allows each layer to be thinner and more compact

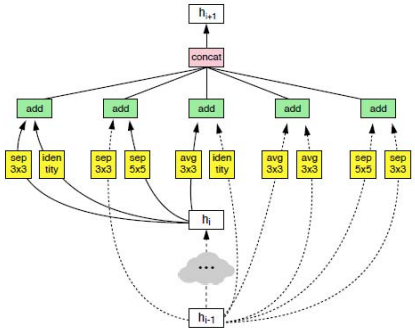


http://www.cs.cornell.edu/~gaohuang/papers/DenseNet-CVPR-Slides.pdf

Learning Better Architectures



reduction cell



normal cell

Learning Transferable Architectures for Scalable Image Recognition
B Zoph, V Vasudevan, J Shlens, Q Le (2017)

deeplearning.ai | coursera

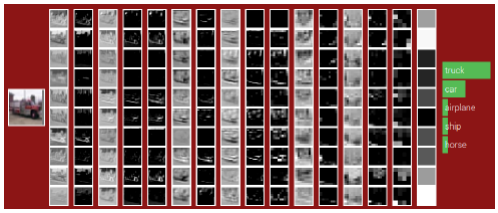
Andrew Ng, 2017

https://www.deeplearning.ai/

Deep Learning Specialization, Five courses:

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks
3. Structured Machine Learning Projects
4. Convolutional Neural Networks
5. Sequence Models

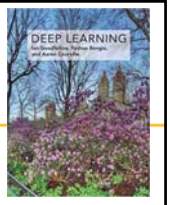
CS231n: Convolutional Neural Networks for Visual Recognition



Li, Johnson, Yeung 2017

http://cs231n.stanford.edu/ l

Books on Deep Learning



- **Grokking Deep Learning** by Andrew Trask. Use Udacity discount code **traskud17** for 40% off. This provides a very gentle introduction to Deep Learning and covers the intuition more than the theory.
  - https://www.manning.com/books/grokking-deep-learning ($49)
- **Neural Networks And Deep Learning** by Michael Nielsen. This book is more rigorous than Grokking Deep Learning and includes a lot of fun, interactive visualizations to play with.
  - http://neuralnetworksanddeeplearning.com/ (free!)
- **The Deep Learning Textbook** from Ian Goodfellow, Yoshua Bengio, and Aaron Courville. This online book contains a lot of material and is the most rigorous of the three books suggested.
  - http://www.deeplearningbook.org/ (html- free!)
  - https://www.amazon.com/Deep-Learning-Adaptive-Computation-Machine/dp/0262035618/ (hard copy $50)

Thank you!!

Ray Ptucha
rwpeec@rit.edu

https://www.rit.edu/mil