CSCI 566: Deep Learning and its Applications

Yue Zhao
Thomas Lord Department of Computer Science
University of Southern California

Credits to previous versions of USC CSCI566, CMU 10601/701, Stanford CS 229, 231n

Logistics

Week 3 Jan 26	1. Classical ML (continued) Neural Network Basics - Perceptron Revisited - Gradient Descent - Forward Propagation 2. Project idea discussion		S24-block3-NeuralNet.pdf
Week 4 Feb 2	1. Neural Network Basics - Backpropogation Propagation - Vanishing Gradient 3. Different types of Neural Networks: - Convolutional Neural Networks	Quiz 1	Course Project Teams Formed; Pre-proposal DUE Google Cloud Platform - 2024.pdf

Active Assignments	Released	Due (PST)	Submissions	% Graded	Published	Regrades
Project Pre-proposal	JAN 24, 2024 12:00 AM	FEB 2, 2024 11:59 PM Late Due Date: FEB 3, 2024 8:59 AM	1	0%		ON

Logistics

Week 5 Feb 9	1. Different types of Neural Networks: - Convolutional Neural Networks 2. Deep Learning Software Tutorial (maybe)	Assignment 1 OUT	S24-block4-CNN.pdf

It will be on Gradescope and due in three weeks (March 1st)

- We do not use late days since it is hard to manage
- But we will curve people's grades later – so no panic
- Give your best shot, and **try** to finish everything – you will be fine ☺

Logistics

Week 7 Feb 23	MIDTERM EXAM

It will be in-person, open-book but no electronic devices

You will have at least two hours for it

Mostly multiple-choice questions

Again, we have a curve by the end of the semester.

Cloud Credits

First, **Google** has $300 for all new signups -- https://cloud.google.com/free?hl=en

- Also, we have applied for the Google Cloud credit ($50/student) -- considering pooling them for the project.

I have also applied for **USC advanced computing credit** -- https://www.carc.usc.edu/ to request the usage of it, please add your information

- https://docs.google.com/spreadsheets/d/1KWPlHyMVZ-2o47IhpFC9gxDYusbchPuvX971HhtUULI/edit?usp=sharing

Cloud Credits

Google Cloud Credit:

https://vector.my.salesforce-sites.com/GCPEDU?cid=U4cYmFRYRNMCY3QWOi3mu7niNVKxs%2Fq76pJeSiD9VVD23kqHSaeE71KE2OJOYly6/

USC Credit: https://docs.google.com/spreadsheets/d/1KWPlHyMVZ-2o47IhpFC9gxDYusbchPuvX971HhtUULI/edit#gid=0

See details: https://piazza.com/class/loendg1jxkryr/post/21

Project and Allocation Management

Log In To User Portal

The **CARC User Portal** is how users manage their projects and allocations on CARC systems. Projects and allocations on both the general use Discovery cluster and the Endeavour condo cluster are managed in the CARC User Portal.

For information on accessing CARC's systems (including the User Portal) and the types of allocations available, see the Accounts and Allocations page.

The CARC User Portal uses ColdFront, an open source resource allocation management tool developed by the University at Buffalo's Center for Computational Research. ColdFront supports the management of resources and user allocations to those resources. The tool allows users to request and manage the access they and their students or collaborators have to the resources in CARC's data center.

The user portal handles the management of 3 components: **projects**, **allocations**, and **resources**. The graphic below shows how these components interact on CARC systems:

Cloud Credits

Allocations 2

Increase allocation size? Create a new Account category support ticket: include project id, allocation size, and reason.
*PIs will get an annual credit of $400 (10TB cost) for project storage adjusted at the end of the charge cycle.

+ Request Resource Allocation

Resource Name	Resource Type	Information	Quantity	Quota	Annual Cost*	Status	End Date	Actions
Discovery	Cluster Partition	slurm_account: yzhao010_1246	class (10k hours)	0.00%	0	Active	Jul. 01, 2024	
ProjectStorage for USC classes	Storage	/project/yzhao010_1246	1	0.00%	0	Active	Jul. 01, 2024	

For Discovery nodes, use the chart below to determine which GPU type to specify:

GPU type	GPU model	Partitions	Max number of GPUs per node
a100	NVIDIA Tesla A100	gpu	2
a40	NVIDIA Tesla A40	gpu	2
v100	NVIDIA Tesla V100	gpu	2
p100	NVIDIA Tesla P100	gpu, debug	2
k40	NVIDIA Tesla K40	main, debug	2

Also note that some A100 GPUs have 40 GB of GPU memory and some have 80 GB of GPU memory. To request a specific A100 model, add one of the following options:

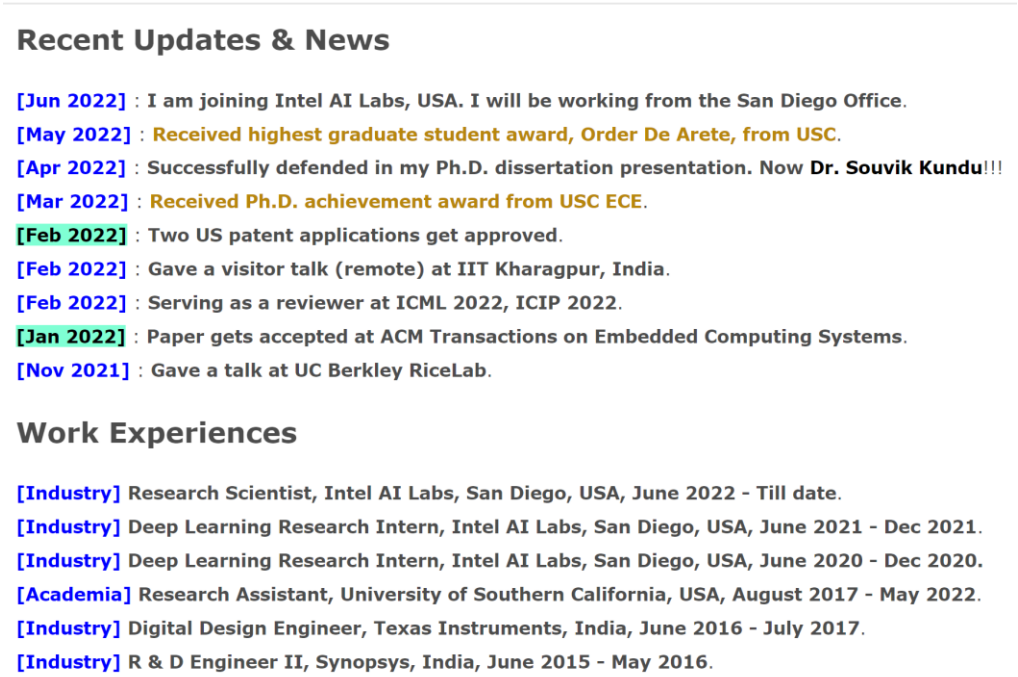
https://www.carc.usc.edu/user-information/user-guides/software-and-programming/using-gpus

Invited Talk 4: Dr. Souvik Kundu

Guest lecture on March 28th

Souvik Kundu is a Research Scientist at Intel AI Labs. Prior to joining Intel, I did my Ph.D. in Electrical & Computer Engineering from USC.

He is still based in LA ☺



Recent Updates & News

[Jun 2022] : I am joining Intel AI Labs, USA. I will be working from the San Diego Office.
[May 2022] : Received highest graduate student award, Order De Arete, from USC.
[Apr 2022] : Successfully defended in my Ph.D. dissertation presentation. Now **Dr. Souvik Kundu**!!!
[Mar 2022] : Received Ph.D. achievement award from USC ECE.
[Feb 2022] : Two US patent applications get approved.
[Feb 2022] : Gave a visitor talk (remote) at IIT Kharagpur, India.
[Feb 2022] : Serving as a reviewer at ICML 2022, ICIP 2022.
[Jan 2022] : Paper gets accepted at ACM Transactions on Embedded Computing Systems.
[Nov 2021] : Gave a talk at UC Berkley RiceLab.

Work Experiences

[Industry] Research Scientist, Intel AI Labs, San Diego, USA, June 2022 - Till date.
[Industry] Deep Learning Research Intern, Intel AI Labs, San Diego, USA, June 2021 - Dec 2021.
[Industry] Deep Learning Research Intern, Intel AI Labs, San Diego, USA, June 2020 - Dec 2020.
[Academia] Research Assistant, University of Southern California, USA, August 2017 - May 2022.
[Industry] Digital Design Engineer, Texas Instruments, India, June 2016 - July 2017.
[Industry] R & D Engineer II, Synopsys, India, June 2015 - May 2016.

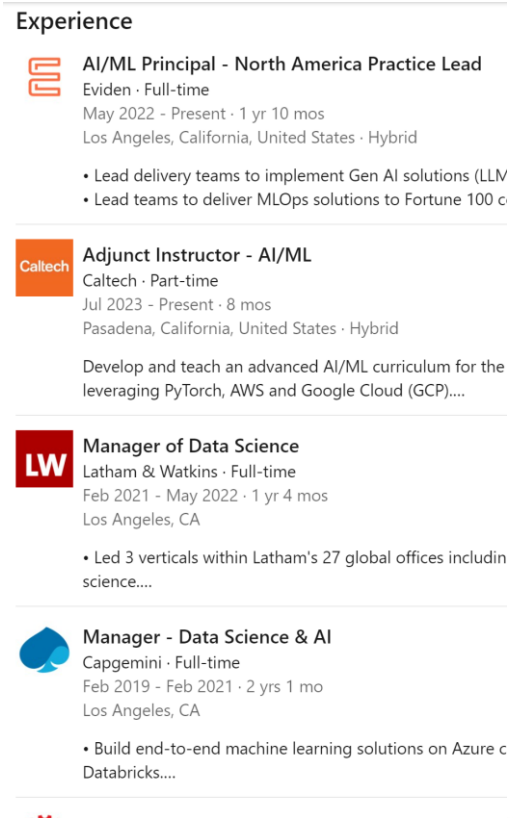
Invited Talk 5: Nicholas Beaudoin

Guest lecture on March 28th

AI/ML Principal @ Eviden | Instructor @ Caltech

“We will most likely be looking around April at hiring so I don't have anything open right now”

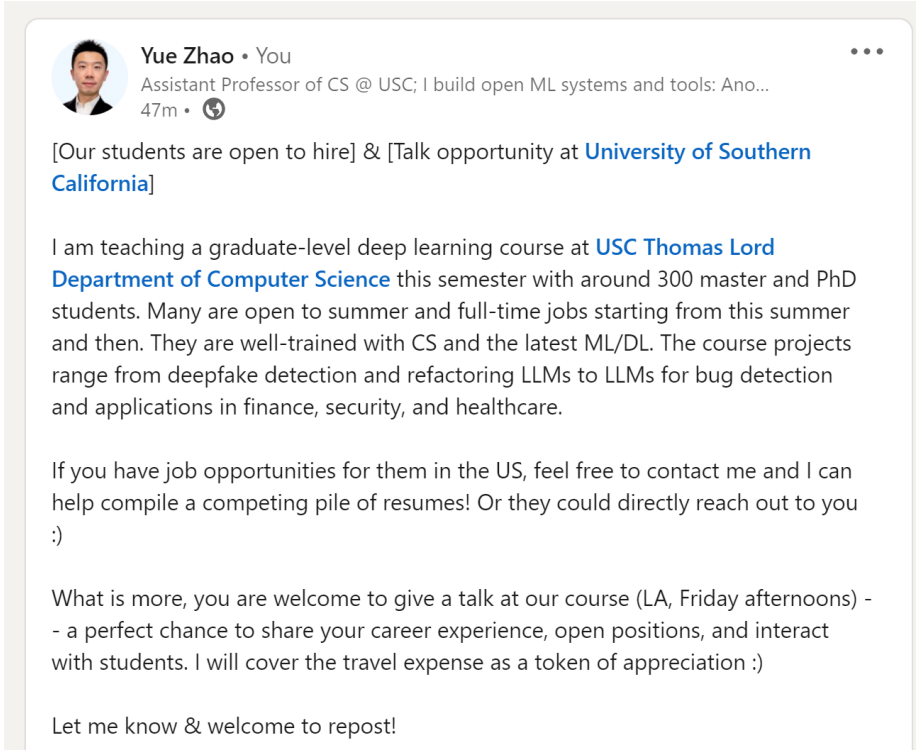
Nicholas is a seasoned professional, with expertise at the intersection of data science and strategic decision-making. With a hybrid background in both public and private organizations, he has honed his skills throughout the entire ML lifecycle.



Referral

I receive emails to refer them for open positions:

- That is indeed a good action, but.
- It is not scalable on my end
- I just sent some posts online to solicit helps



Yue Zhao • You
Assistant Professor of CS @ USC; I build open ML systems and tools: Ano...
47m •

[Our students are open to hire] & [Talk opportunity at University of Southern California]

I am teaching a graduate-level deep learning course at USC Thomas Lord Department of Computer Science this semester with around 300 master and PhD students. Many are open to summer and full-time jobs starting from this summer and then. They are well-trained with CS and the latest ML/DL. The course projects range from deepfake detection and refactoring LLMs to LLMs for bug detection and applications in finance, security, and healthcare.

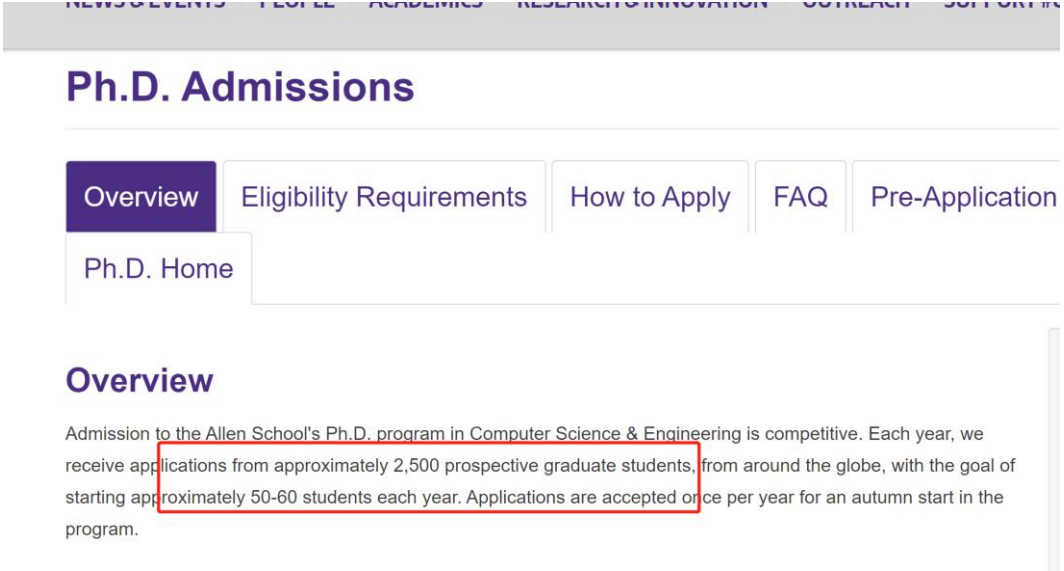
If you have job opportunities for them in the US, feel free to contact me and I can help compile a competing pile of resumes! Or they could directly reach out to you :)

What is more, you are welcome to give a talk at our course (LA, Friday afternoons) -- a perfect chance to share your career experience, open positions, and interact with students. I will cover the travel expense as a token of appreciation :)

Let me know & welcome to repost!

Current Ph.D. Application Situation

For top universities, e.g., top 50 in the U.S., the number of applications increases like crazy – anything ranging from 2%~10% (like us)



Current Ph.D. Application Situation

However, many decent universities face the issue of not getting enough applicants – which is good opportunity if you are looking into industry jobs after the Ph.D. time.

- Top 100-200
- Not in the huge cities but in less popular locations

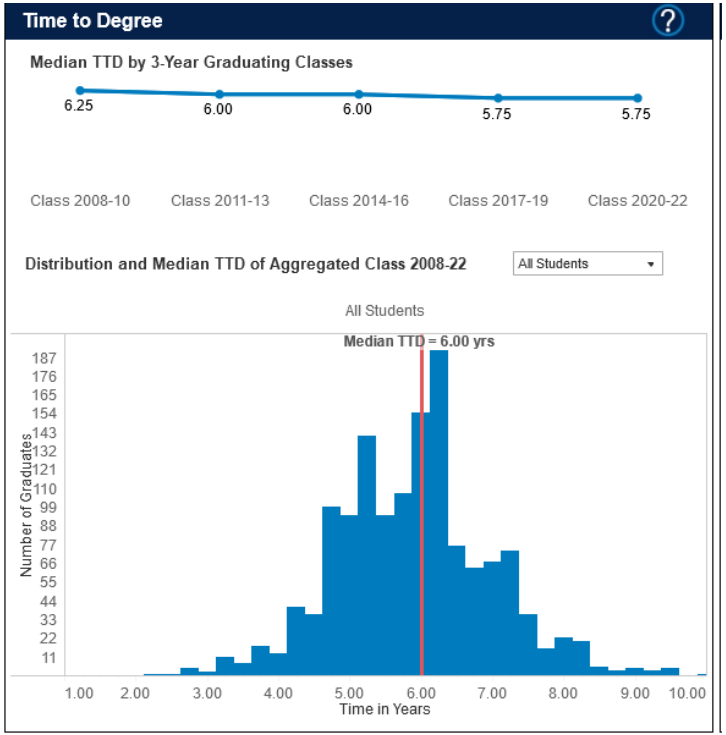
Should I do it?

Pros:

- An alternative when industry goes bad
- Better leverage/title in a few years

Cons:

- Waste of time
- May not generate any impact
- Even may not get it (~49% in the U.S.)
- Have to find something between June to next September…



USC Viterbi School of Engineering

How to secure a position?

For top universities, we value:

- Research potentials – via letters from top researchers in the field
- Publications – the track record to show you will excel in the future
    - Top venues only
- Nothing else (many of time, connection is all u need ☹)

For decent universities, we value:

- Good undergraduate school (top ones from each country) with good GPAs
- Maybe some publications to show interest (optional)

How to prepare for it?

What is more urgent?

- Consider whether you want to do it
- Understand and find the topic to work on
- Identify potential research mentors at USC to accumulate research experience – PhD students are most appropriate to yield publications
    - Note: course recommendation letters mean NOTHING
    - Unfortunately, I have too much on my plate
- Reach out (cold email) to potential Ph.D. advisors by research interest and shared background

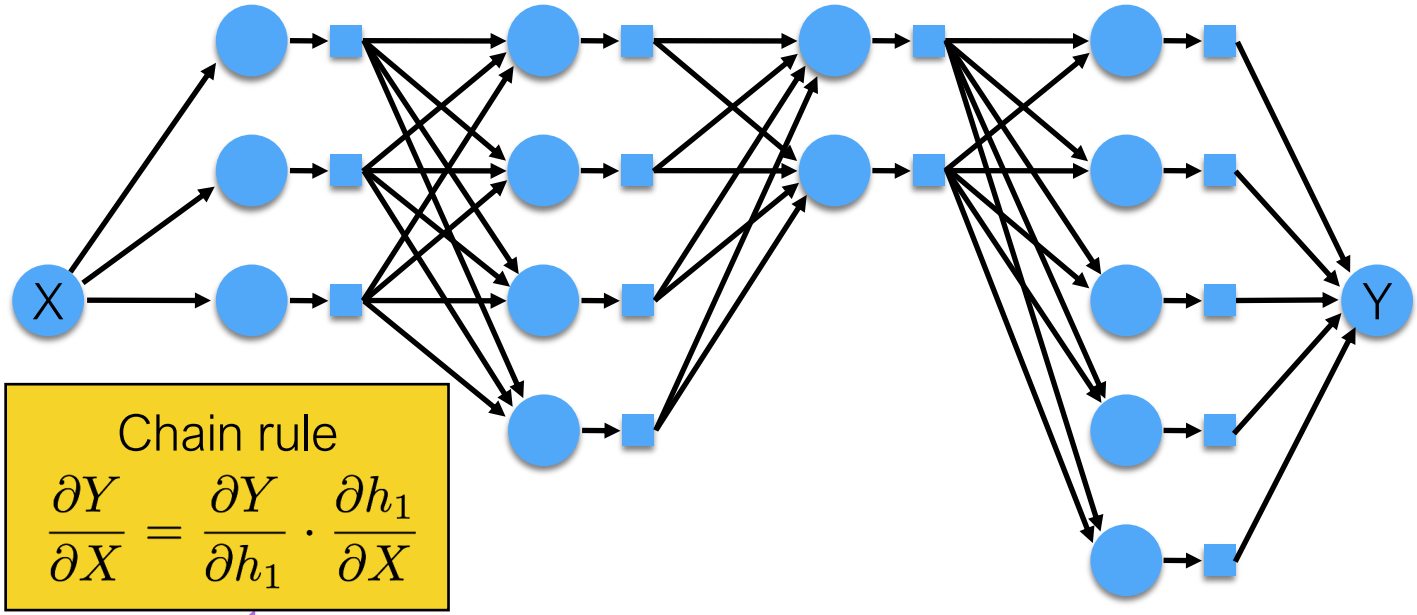
Convolutional Neural Networks (CNN)

Recap on NN

Different NN models

- Feedforward networks (CNNs, …)
- Recurrent networks (RNNs, …)
- Memory networks
- …

Recap: Backward Propagation



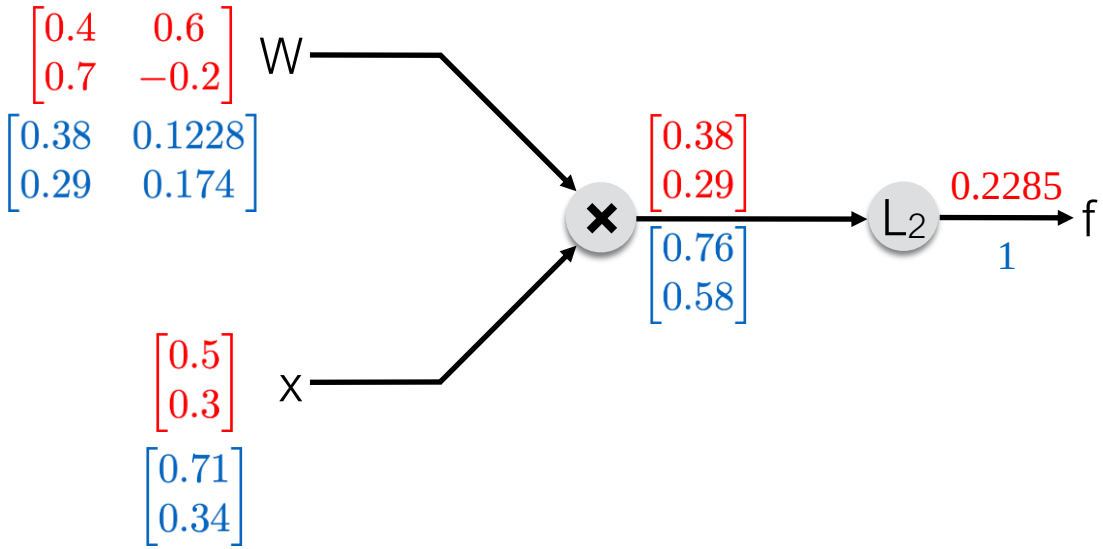
Chain rule

$$\frac{\partial Y}{\partial X} = \frac{\partial Y}{\partial h_1} \cdot \frac{\partial h_1}{\partial X}$$

Backward propagation

Recap: Computational Graph

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \Sigma_{i=1}^{n}(W \cdot x)_i^2$



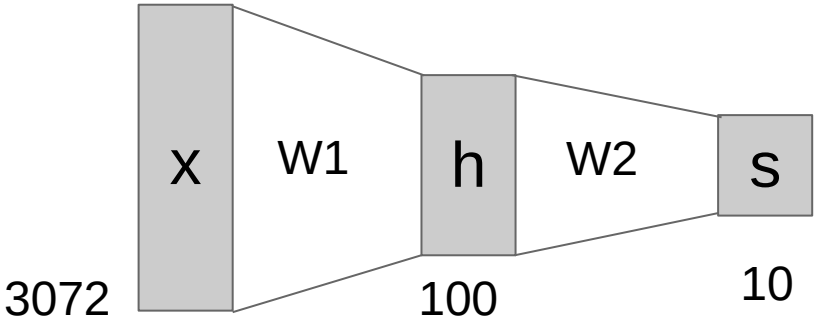
Recap : Neural Networks

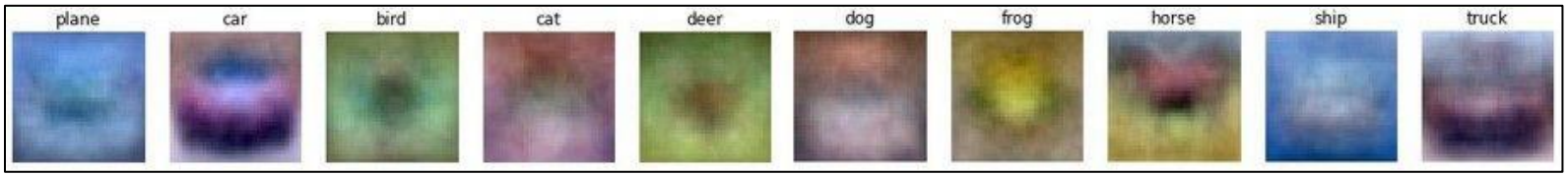
Linear score function:

$$f = Wx$$

2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$





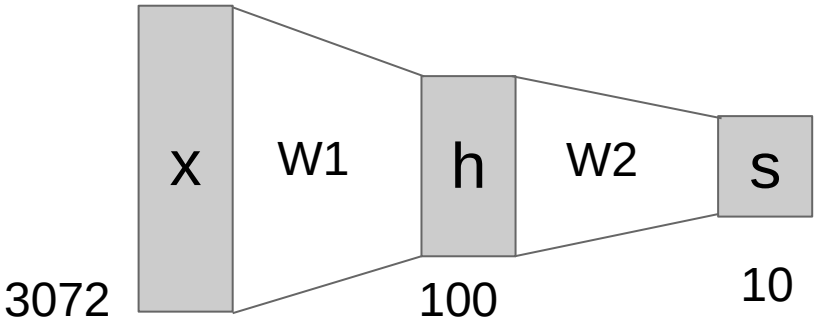
Recap : Neural Networks

Linear score function:

$$f = Wx$$

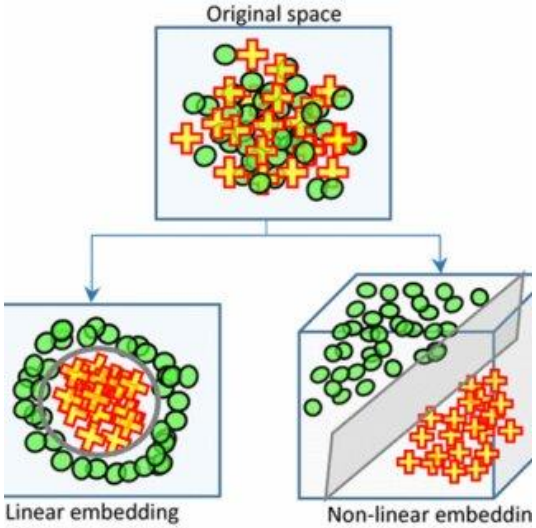
2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$



That is why NN is often called "feature extractors" – learn a good space

Recap : Neural Networks



That is why NN is often called “feature extractors” – learn a good space

Next: Convolutional Neural Networks

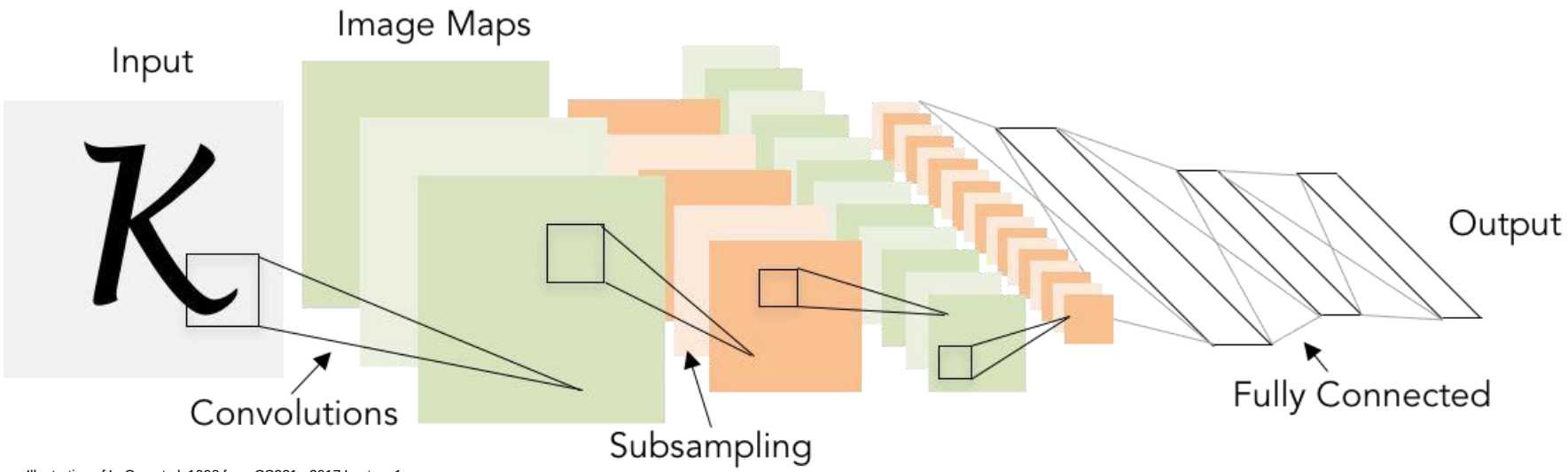


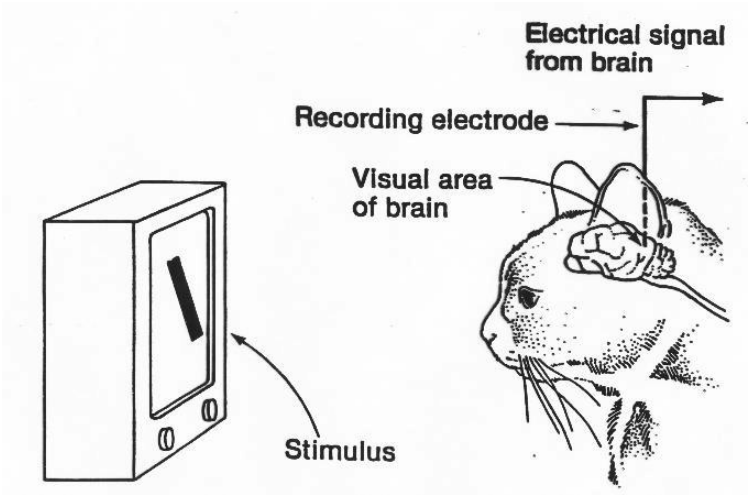
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

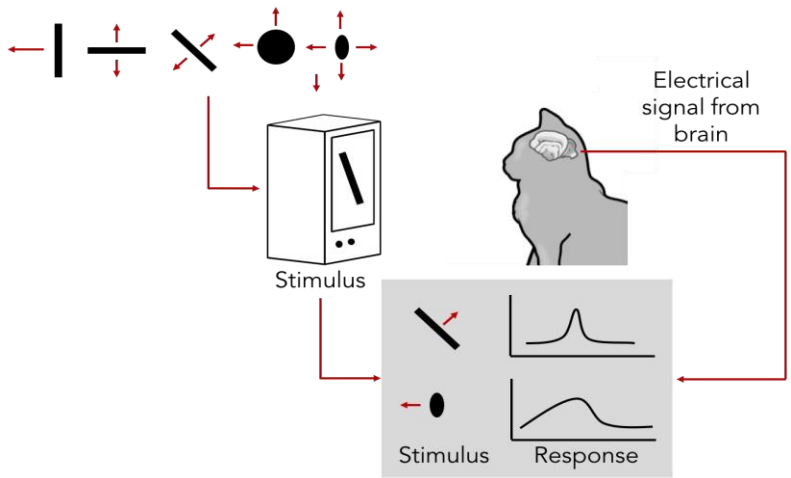
Convolutional Neural Networks (CNN)

History

History

Hubel & Wiesel 1959



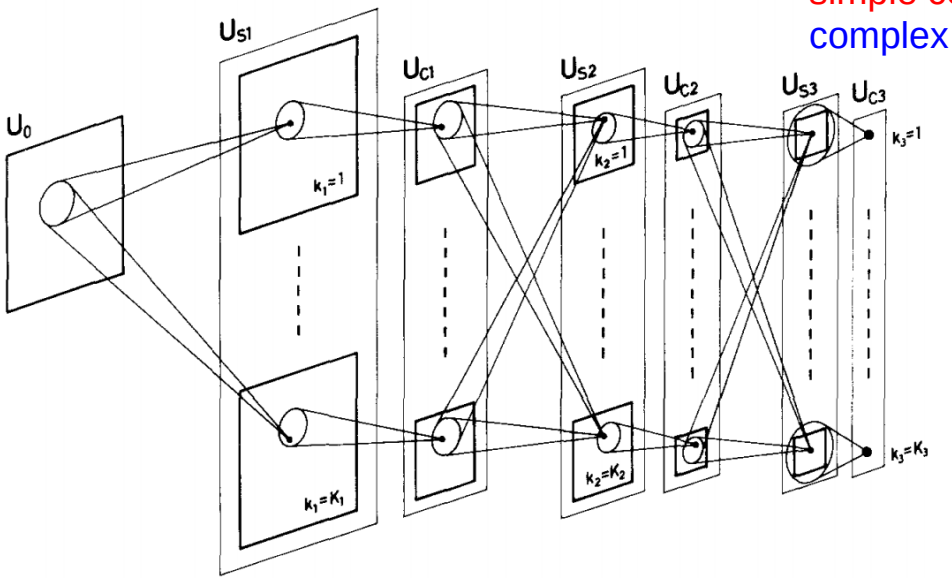


Understanding how the system constructs complex representation from simple stimulus features.

History

Neocognitron 1980

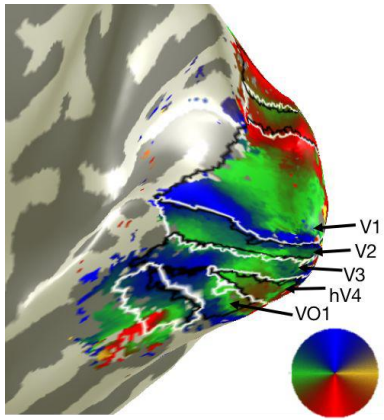
"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling



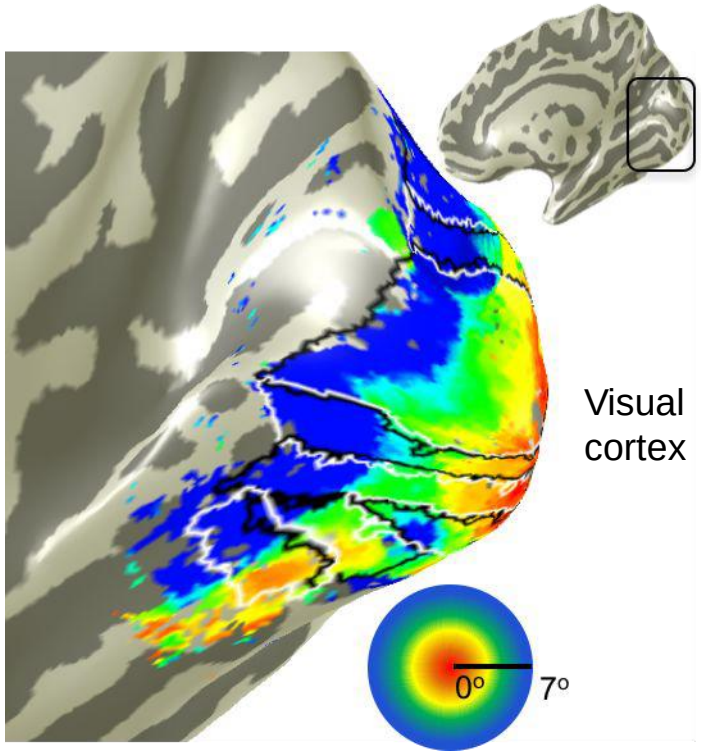
Fukushima, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. Competition and cooperation in neural nets 1982.

History

Topographical mapping in the cortex:
nearby cells in cortex represent
nearby regions in the visual field



There is a direct relationship between the spatial layout of the retina and the spatial layout of the visual cortex.



Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

USC Viterbi School of Engineering

Hierarchical organization

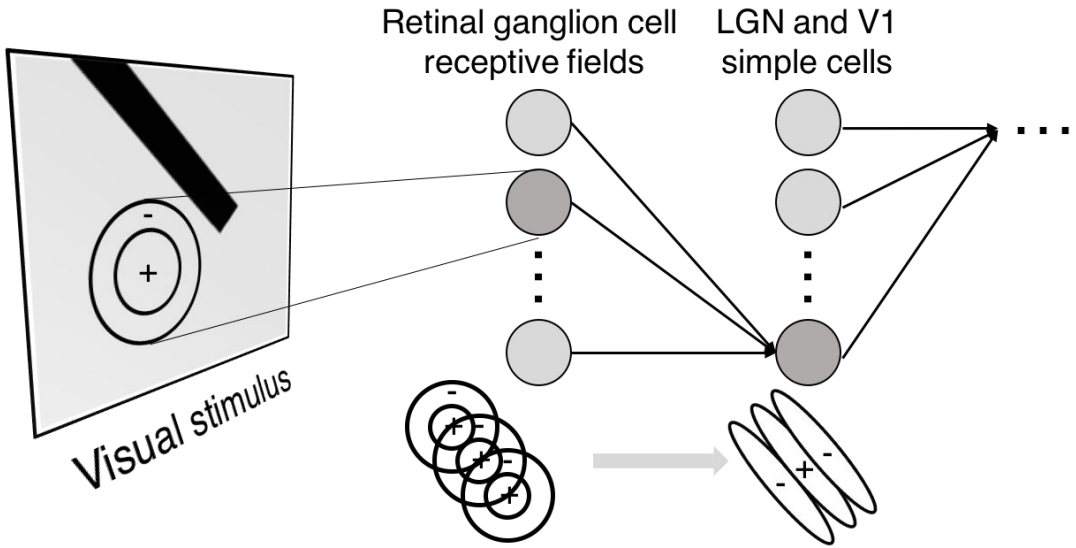
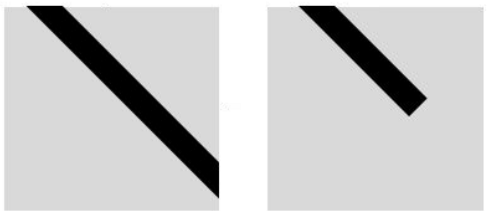


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

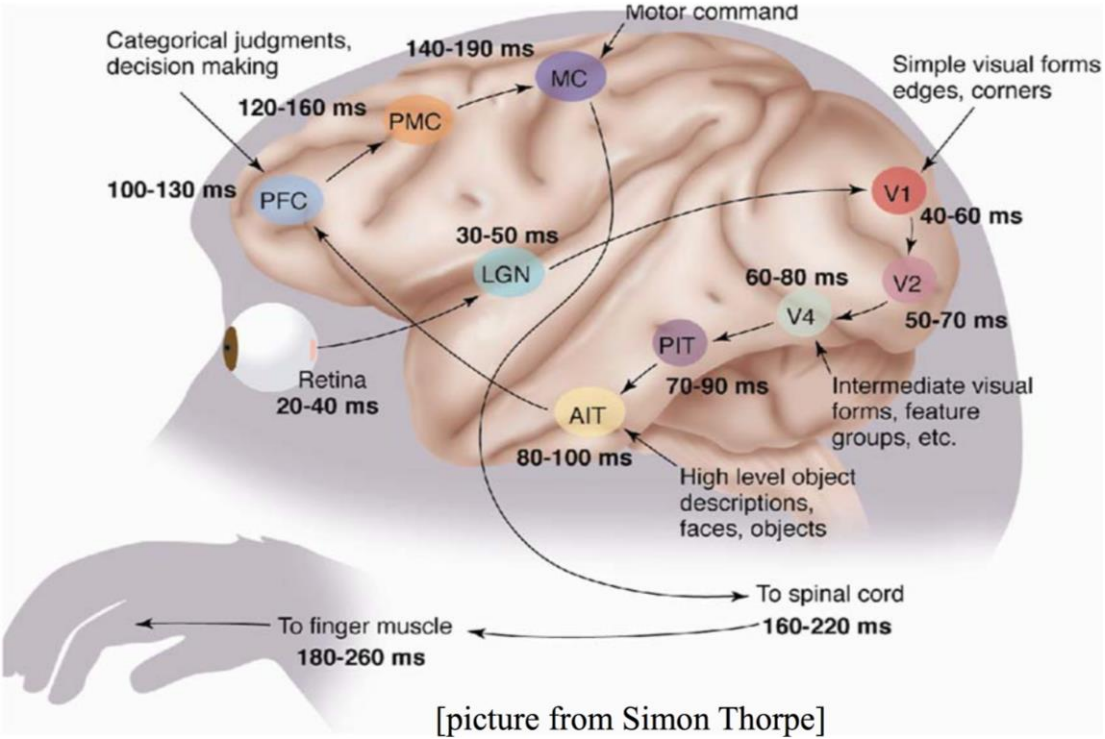
Hypercomplex cells:
response to movement with an end point



No response

Response (end point)

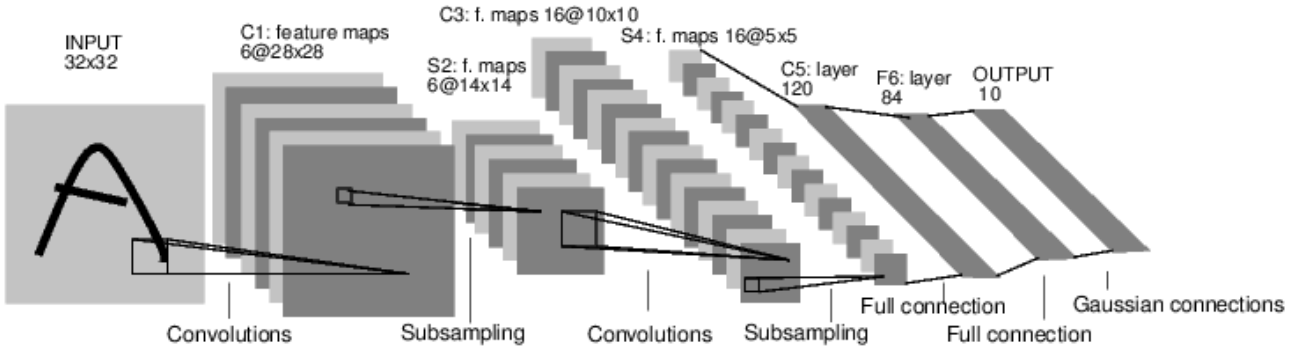
Deep Learning may be Motivated by Human Brain



[picture from Simon Thorpe]

History

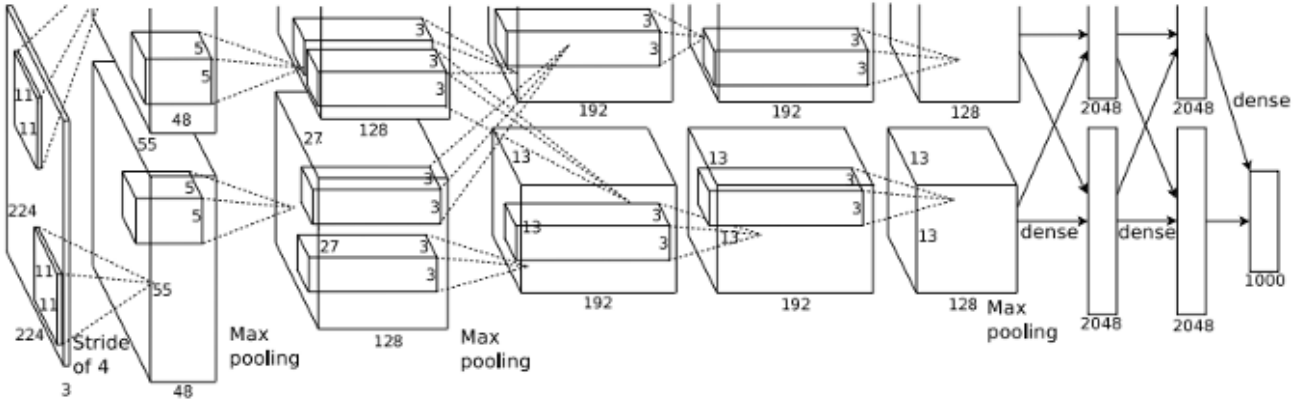
LeNet



Y. LeCun, et. al. Handwritten digit recognition with a back-propagation network. NIPS 1989.

History

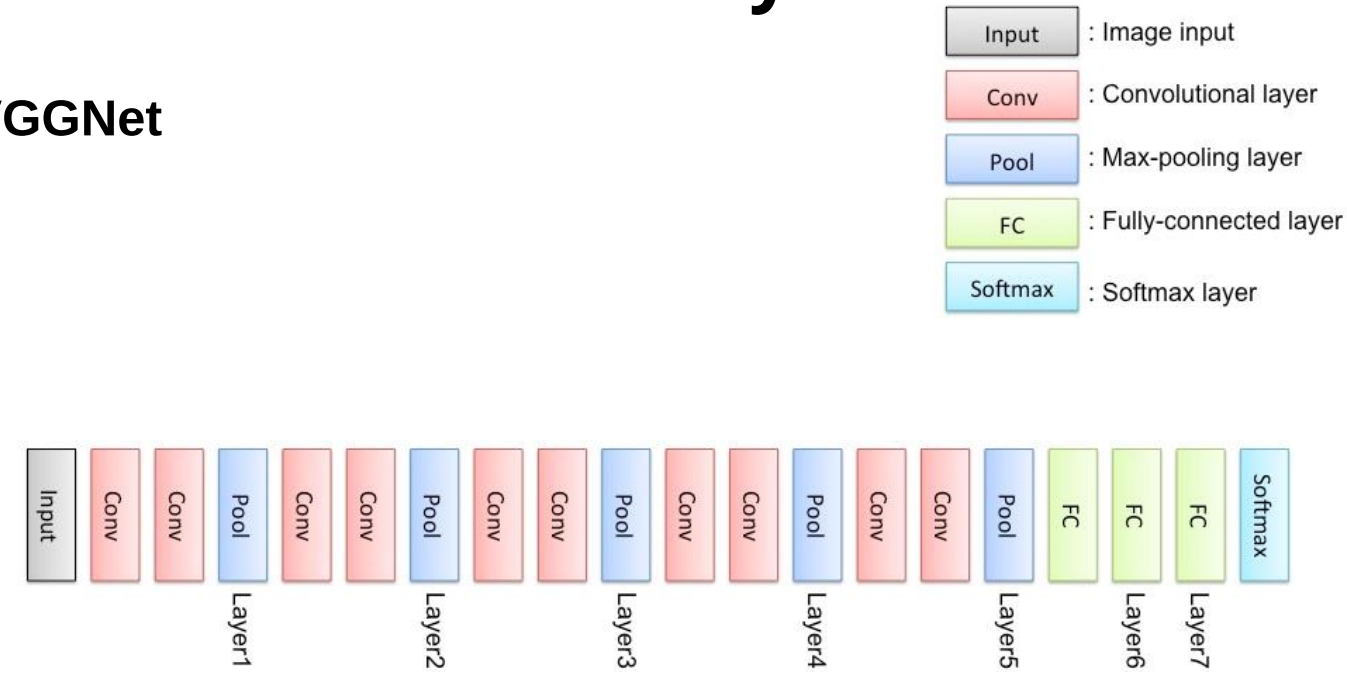
AlexNet



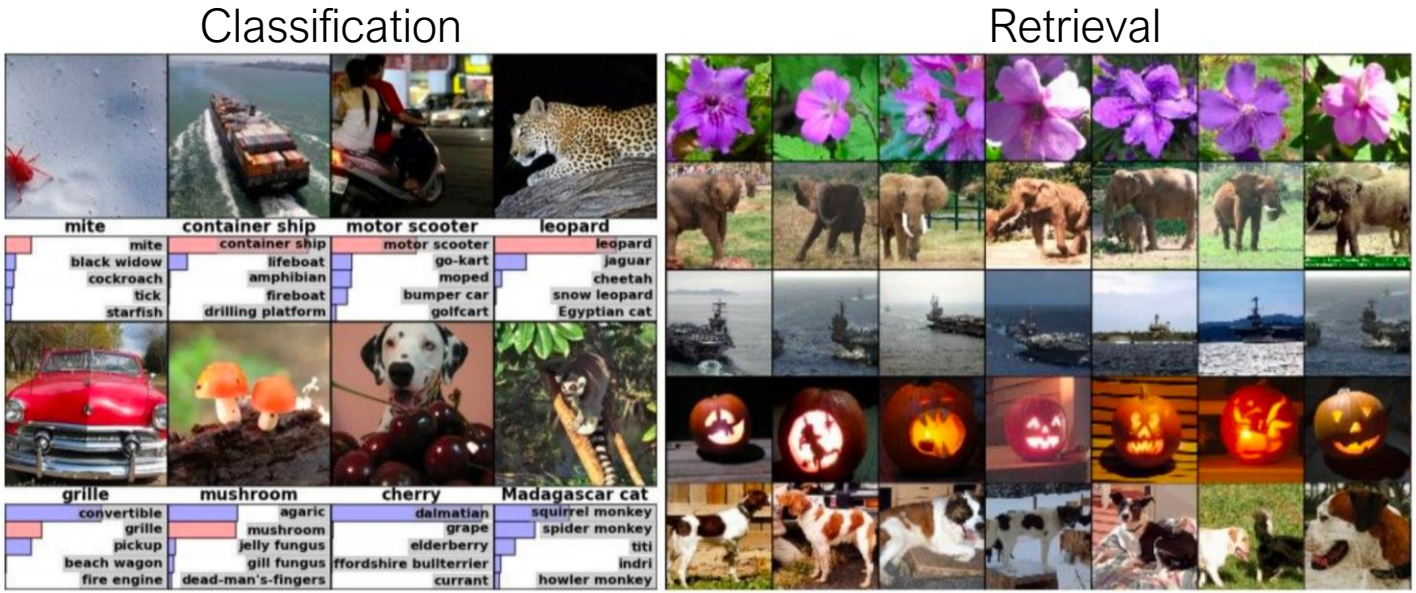
AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever (OpenAI thing in 2023 ☺) and Geoffrey Hinton.

History

VGGNet

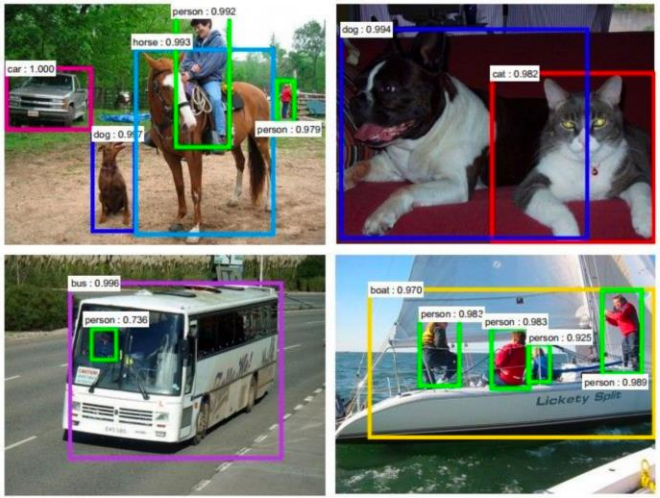


Convolutional NNs



Convolutional NNs

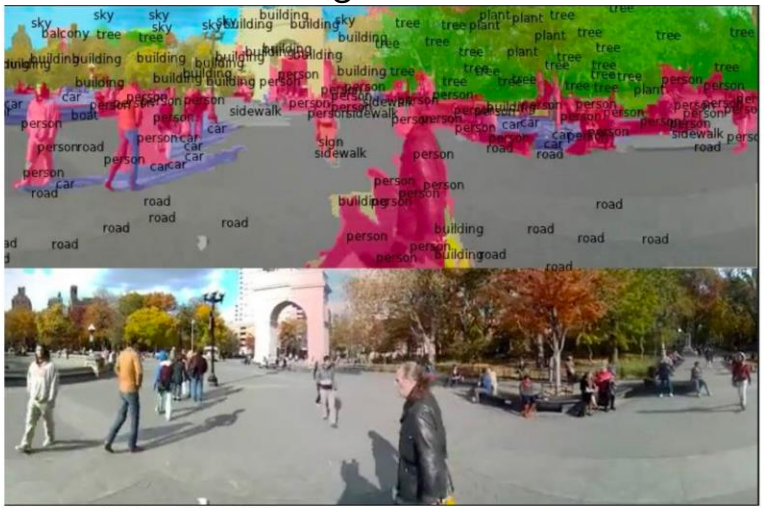
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

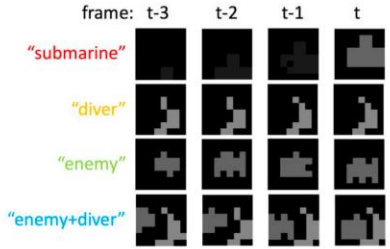


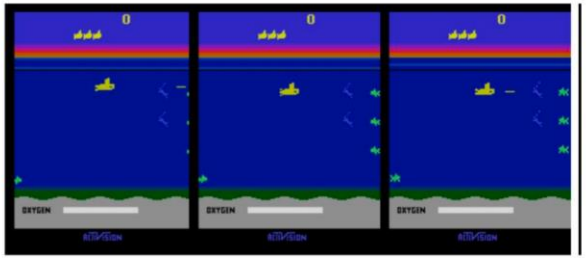
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Convolutional NNs

Game AI





[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Arts

Slide credit: CS 231N

Convolutional NNs

Image Captioning

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

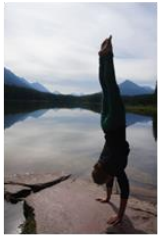
Somewhat related

A woman is holding a cat in her hand

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

A man riding a wave on top of a surfboard

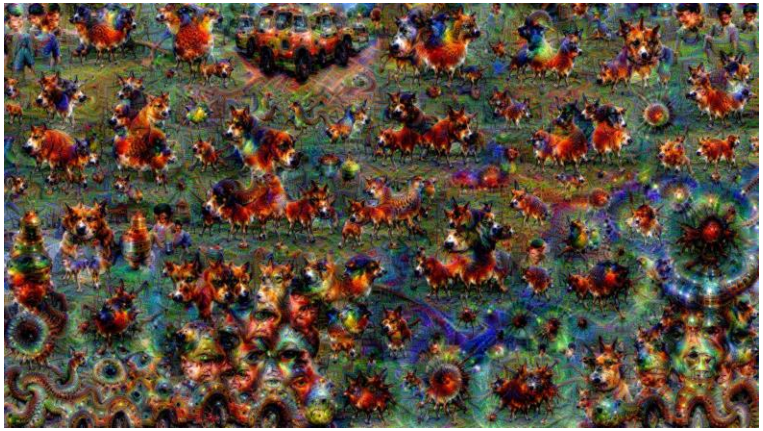
A cat sitting on a suitcase on the floor

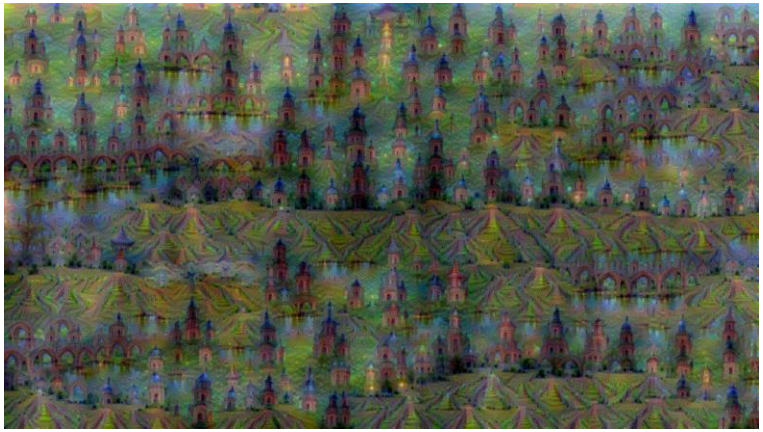


A woman standing on a beach holding a surfboard

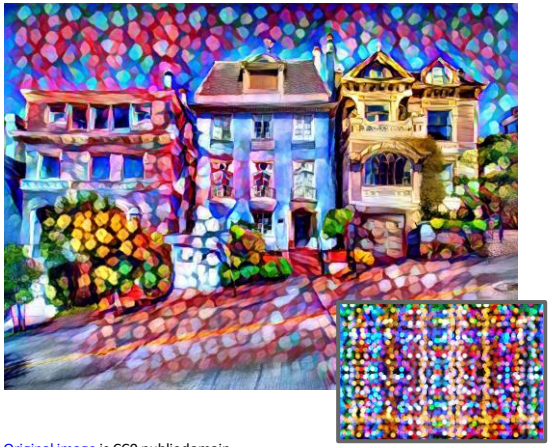
All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Slide credit: CS 231N

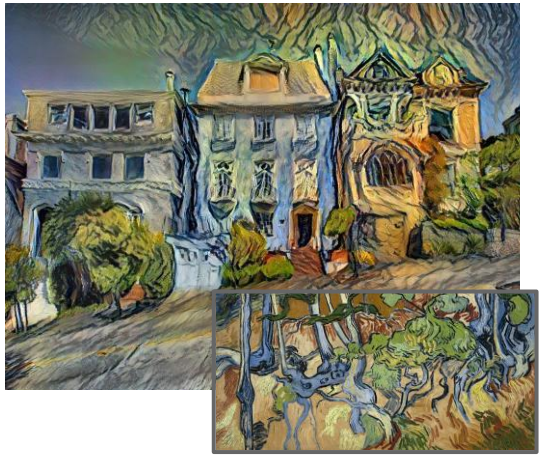




Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



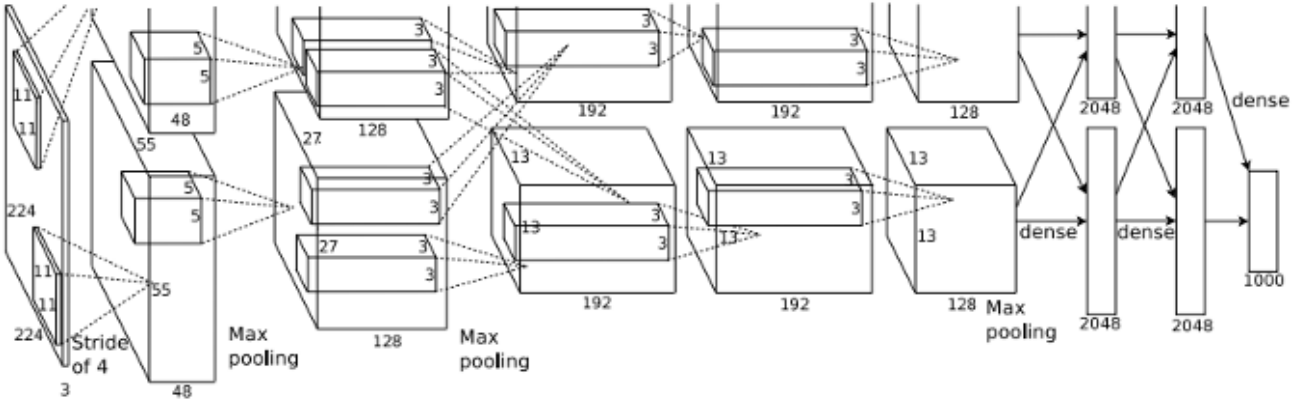
Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission



Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Tr

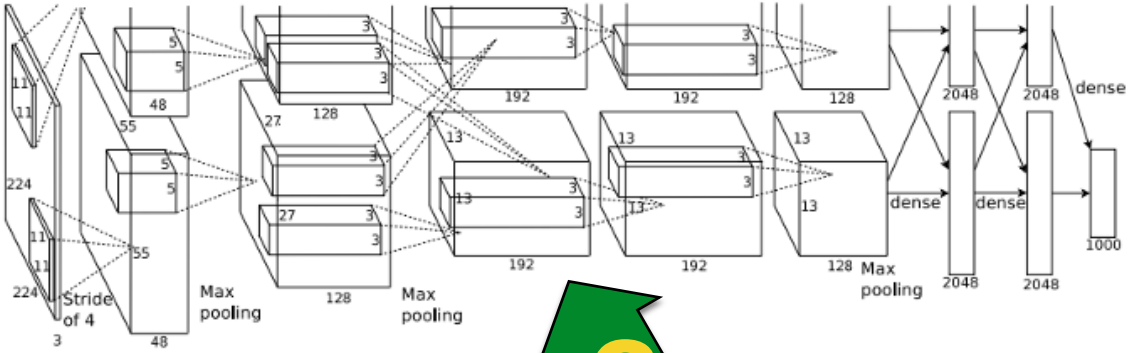
Slide credit: CS 231N

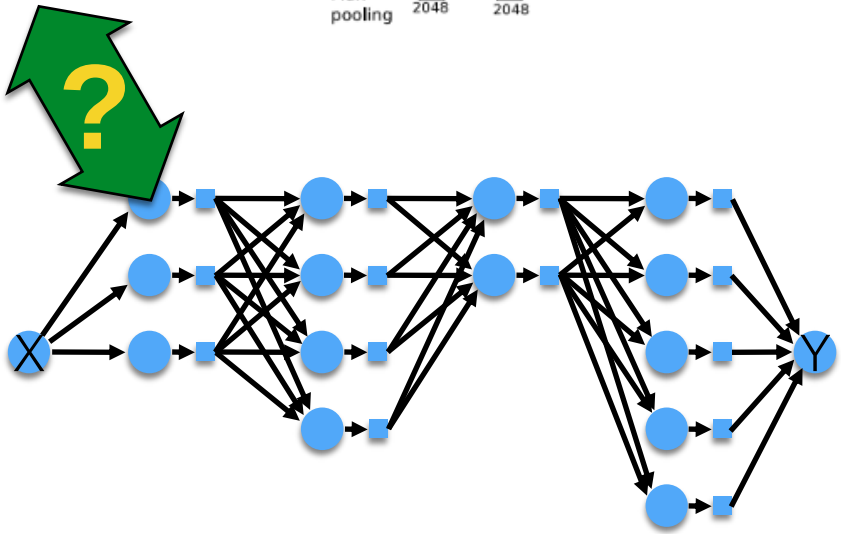
AlexNet in details



A Krizhevsky, et. al. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012.

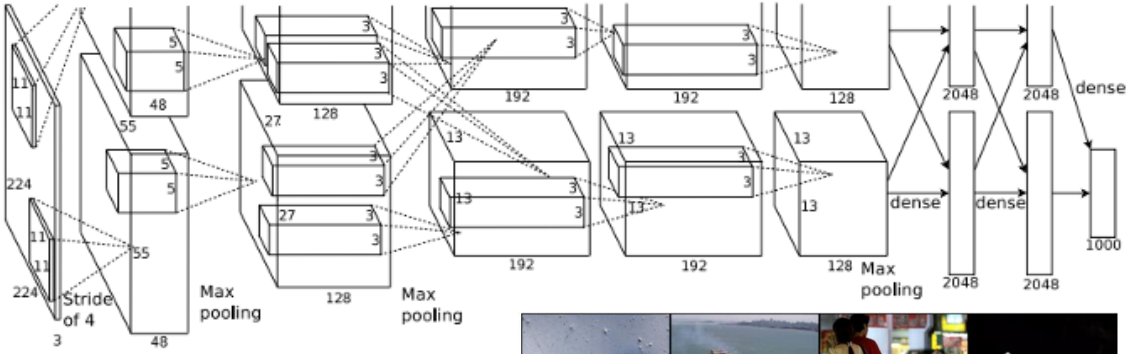
AlexNet in details

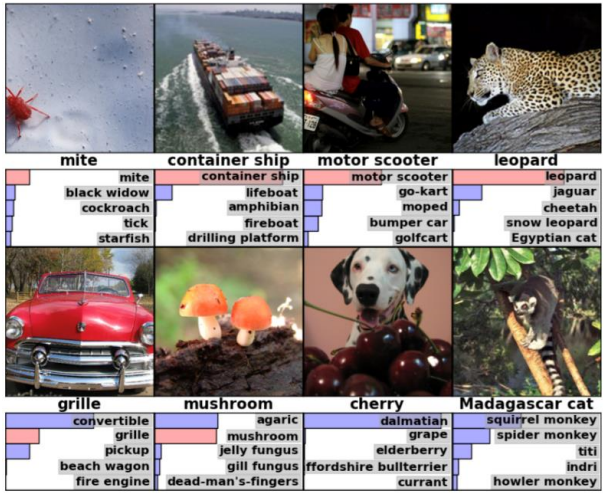




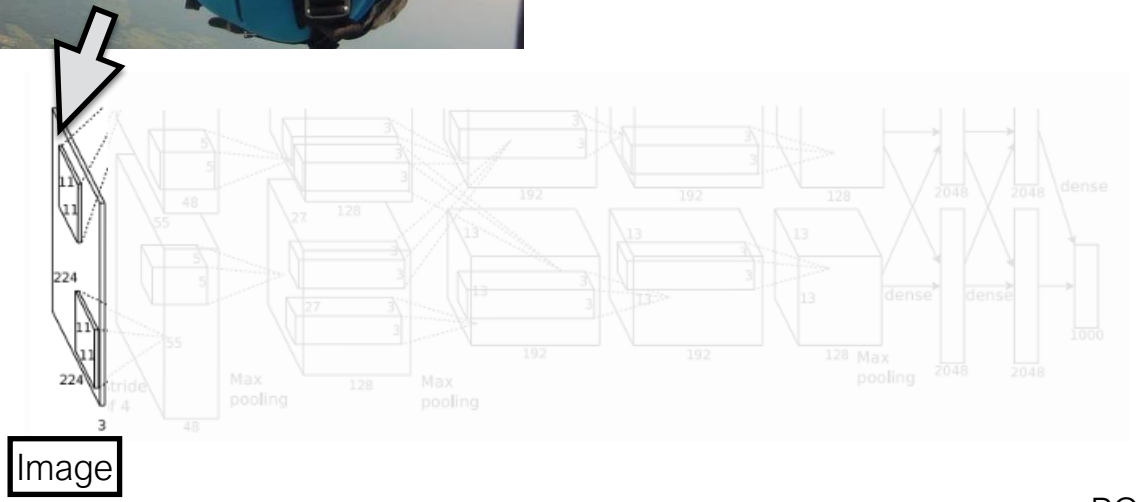
USC Viterbi School of Engineering

AlexNet in details



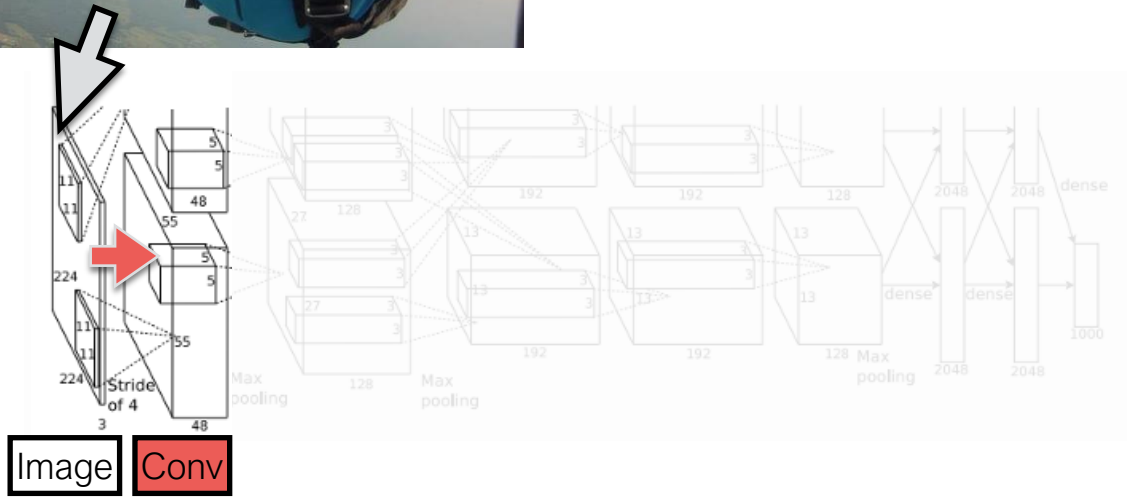


AlexNet in details

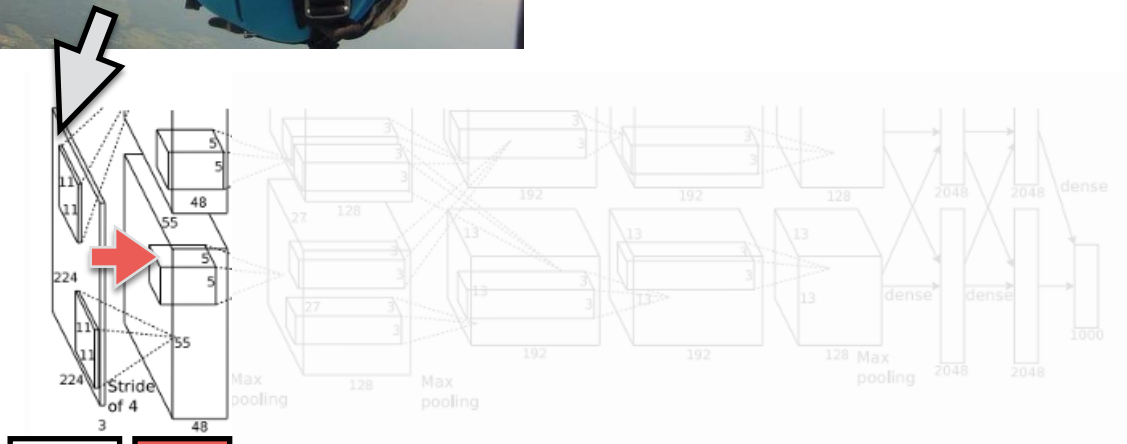


PC: Daniel Yang

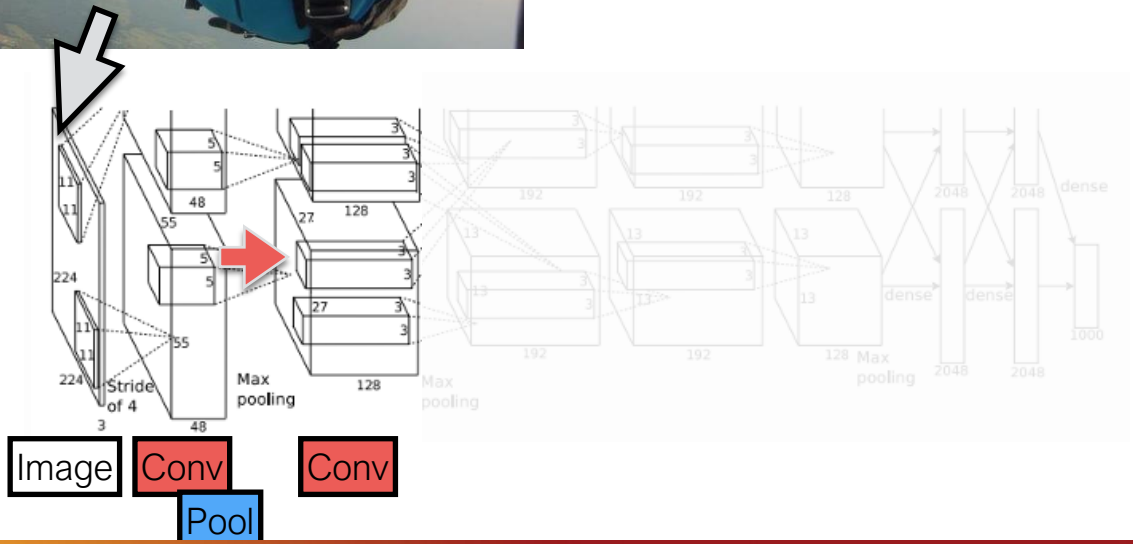
AlexNet in details



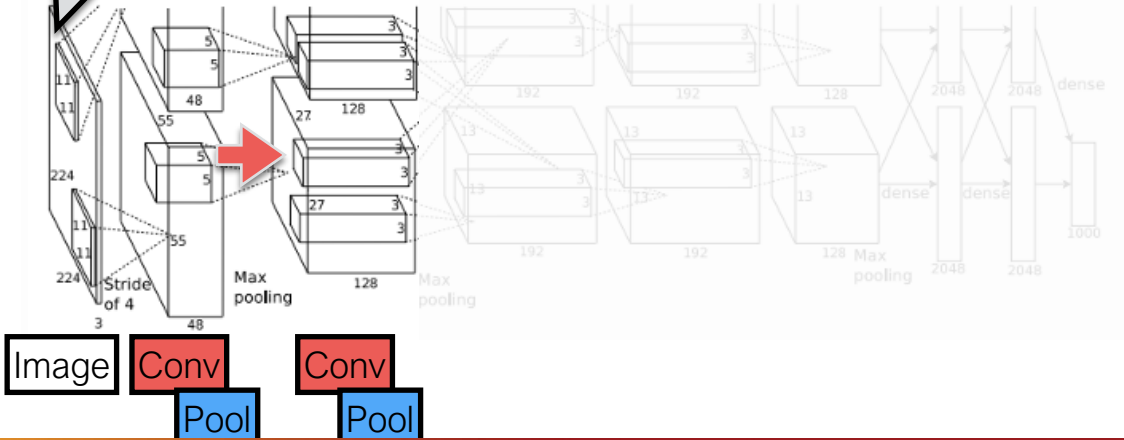
AlexNet in details



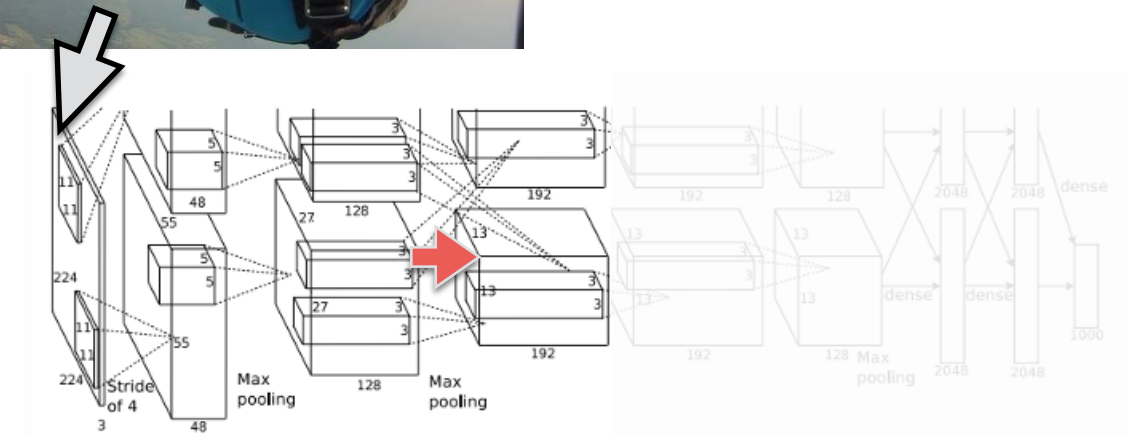
AlexNet in details



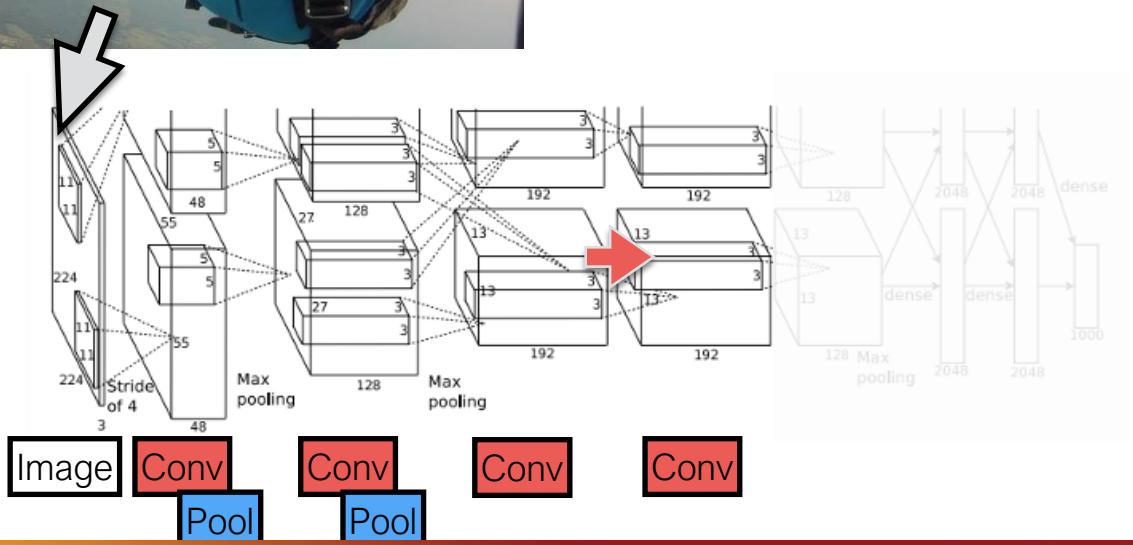
AlexNet in details



AlexNet in details

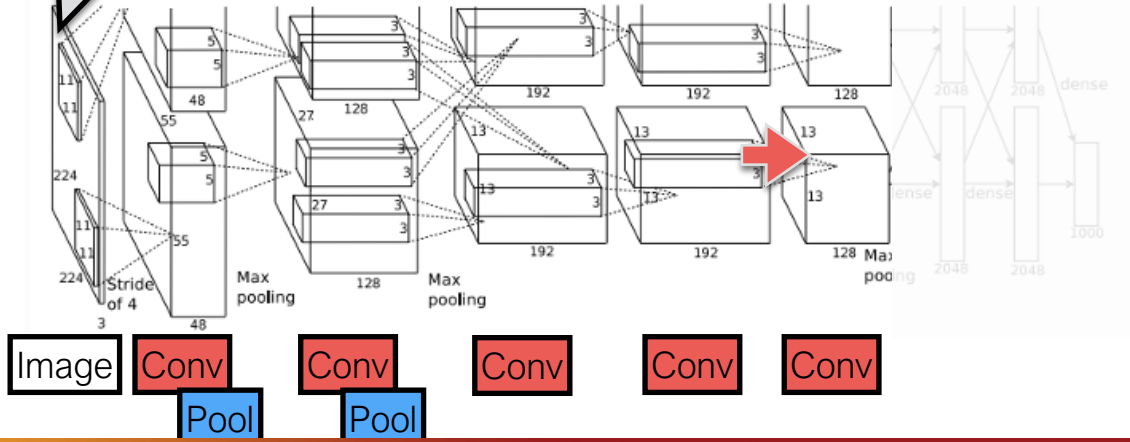


AlexNet in details

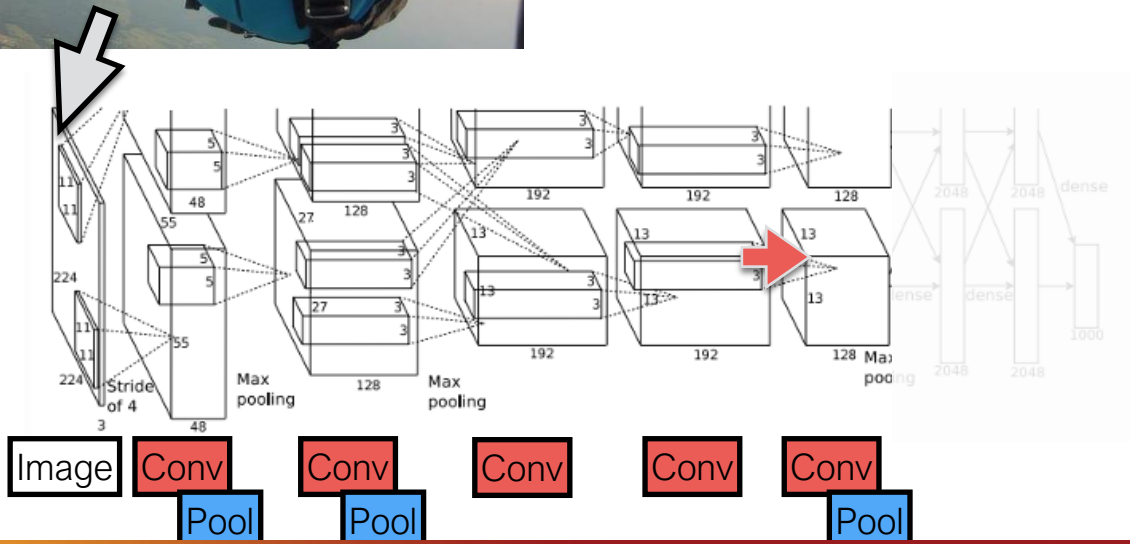


USC Viterbi School of Engineering

AlexNet in details

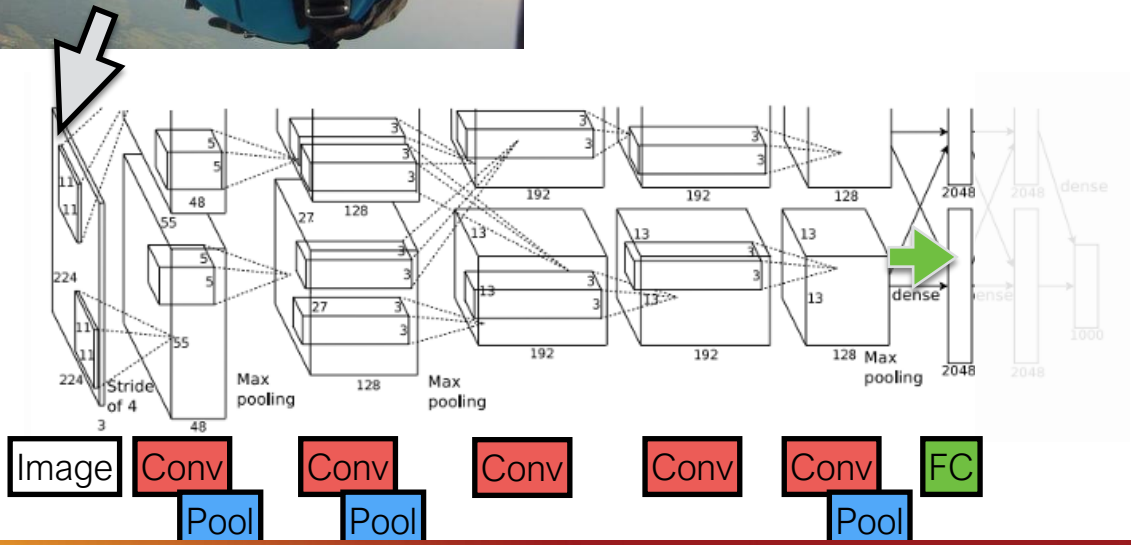


AlexNet in details

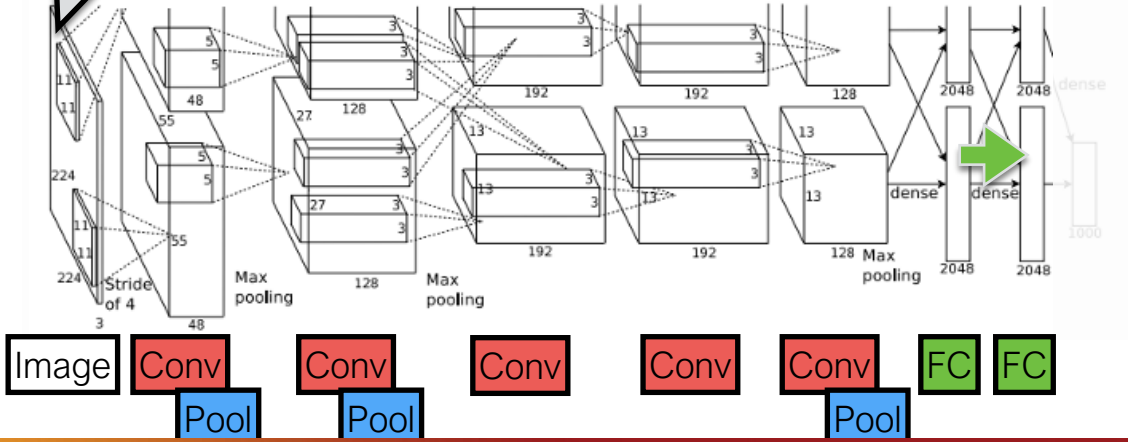


USC Viterbi School of Engineering

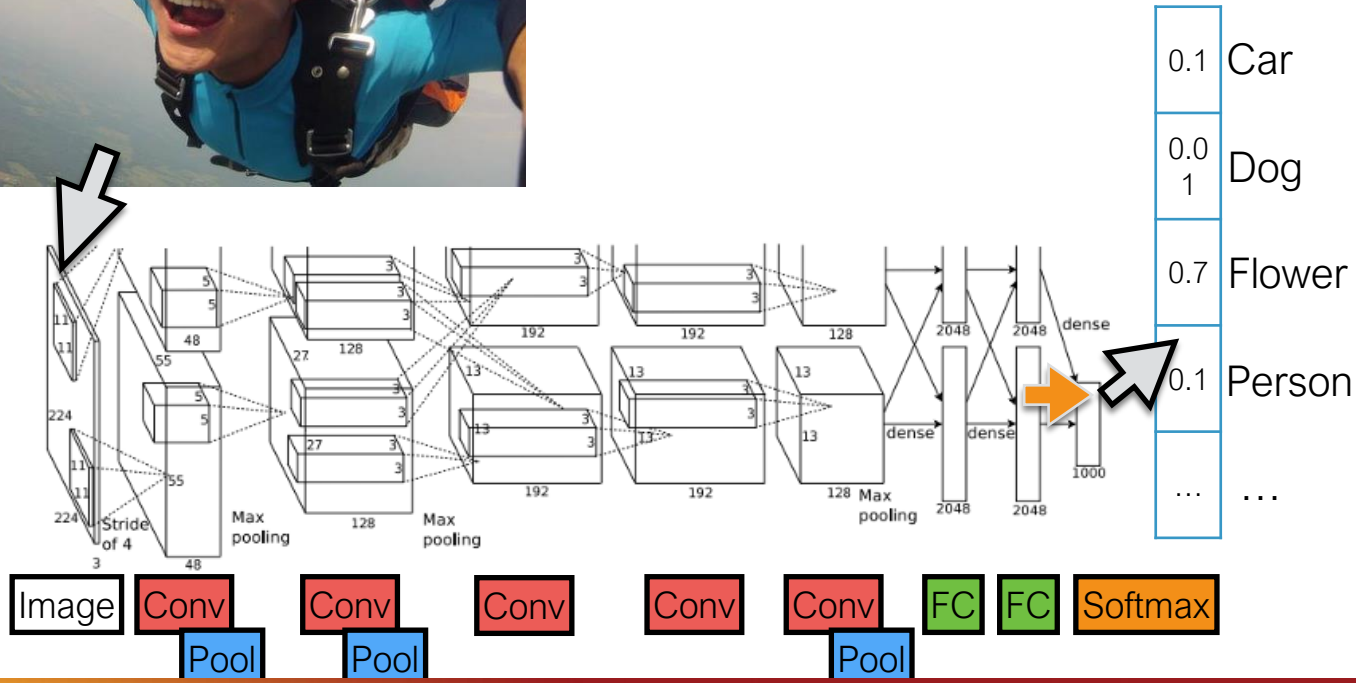
AlexNet in details



AlexNet in details



AlexNet in details



AlexNet in details

Convolutional Neural Networks (CNN)

Layers

What are these layers?

- Convolutional layer 

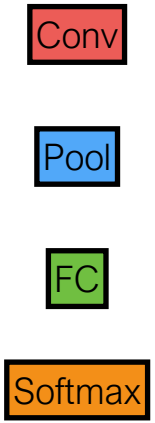
- Pool layer 

- Fully connected layer 

- Softmax layer 

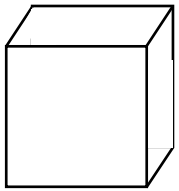
What are these layers?

- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



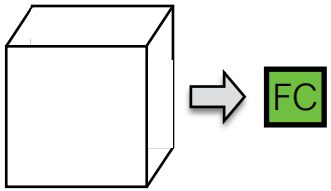
Layer = Function

Fully connected layer (FC)



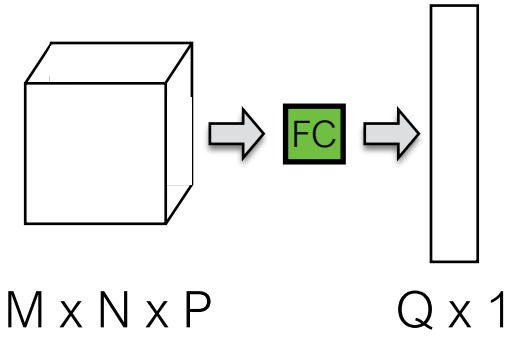
M x N x P

Fully connected layer (FC)

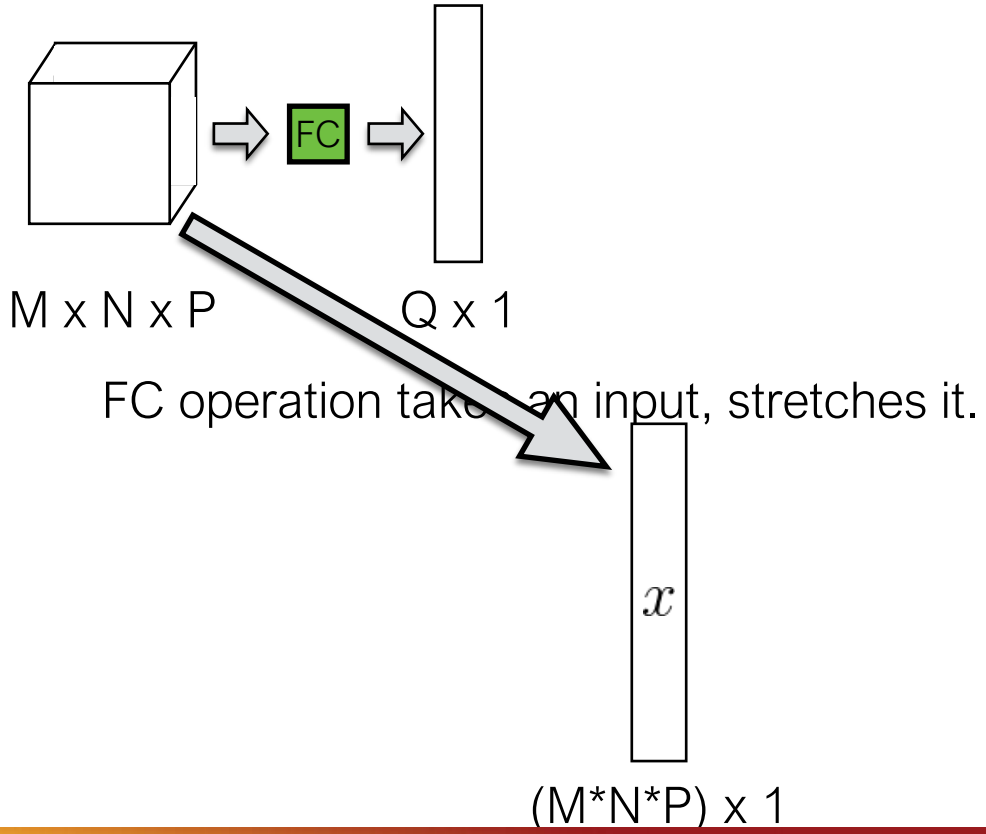


M x N x P

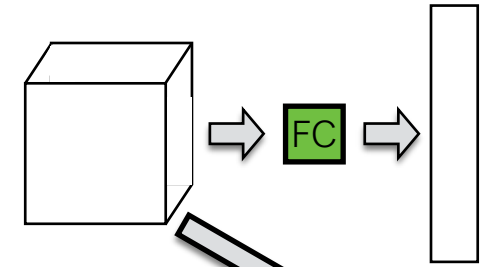
Fully connected layer (FC)



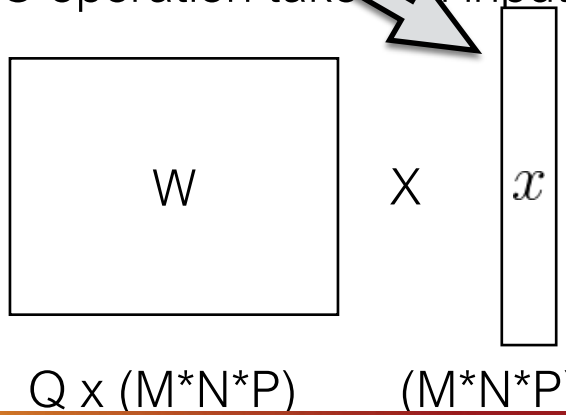
Fully connected layer (FC)



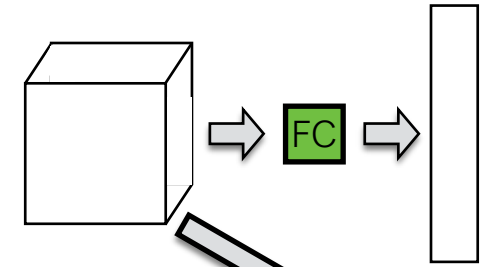
Fully connected layer (FC)



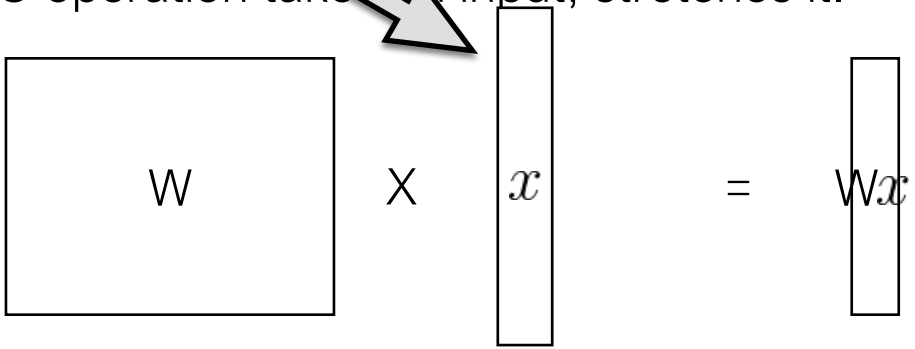
FC operation takes an input, stretches it.



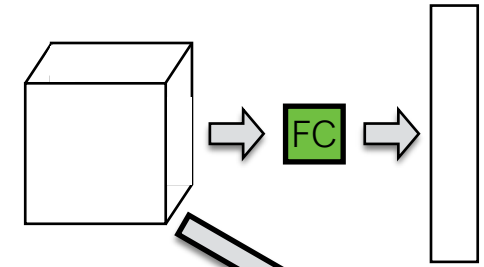
Fully connected layer (FC)



FC operation takes an input, stretches it.



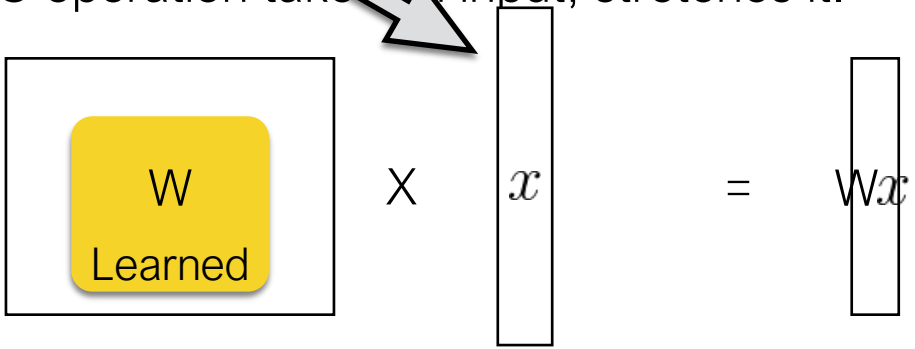
Fully connected layer (FC)



M x N x P

Q x 1

FC operation takes an input, stretches it.

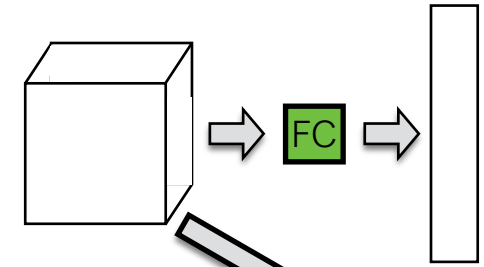


Q x (M*N*P)

(M*N*P) x 1

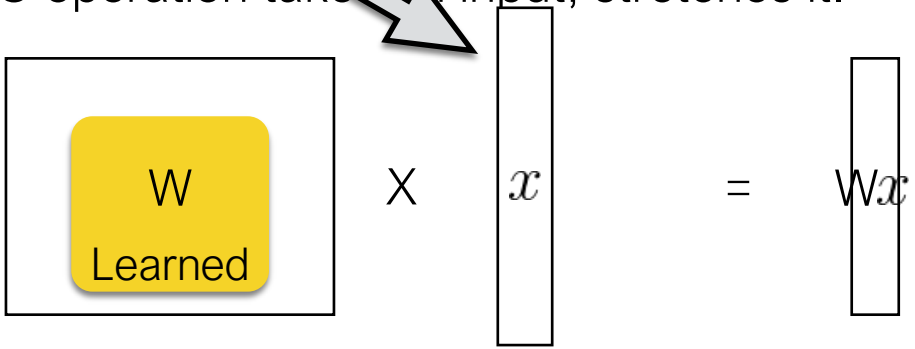
Q x 1

Fully connected layer (FC)

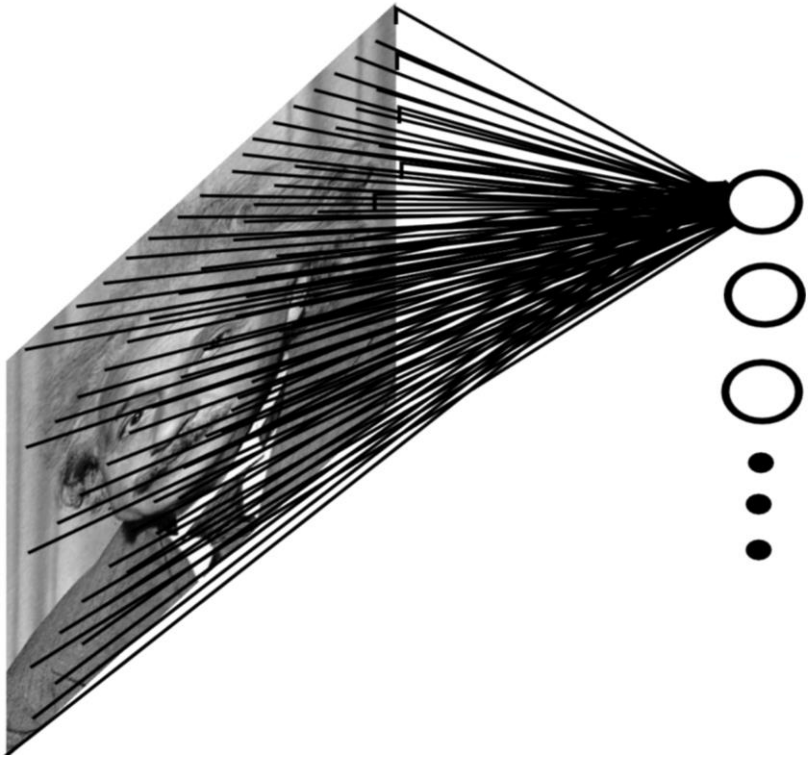


Why is this called FC?

FC operation takes an input, stretches it.



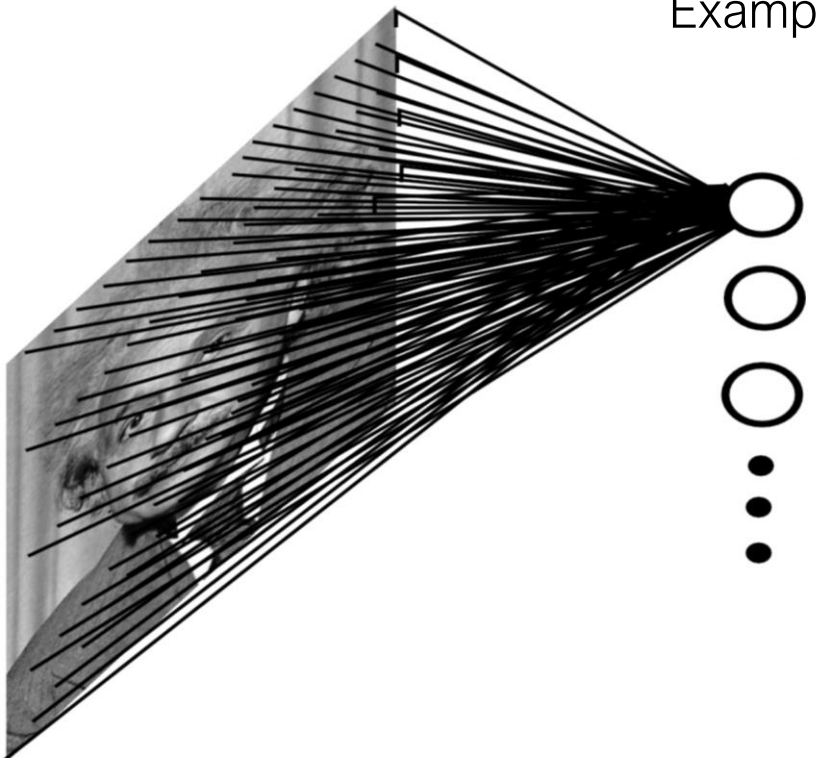
Fully connected layer (FC)



Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

Example: 200x200 image

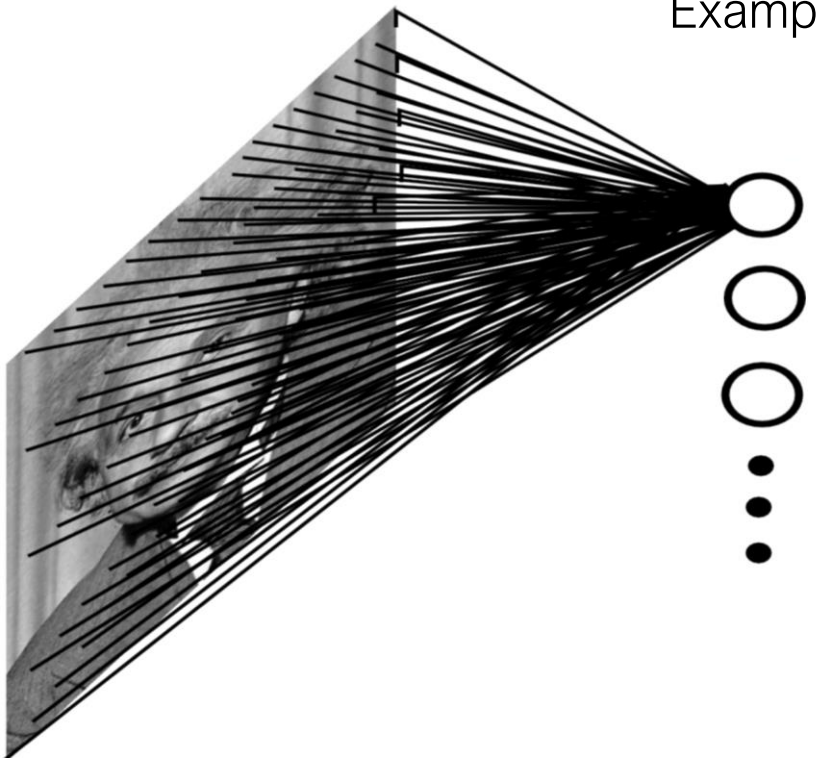


Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)

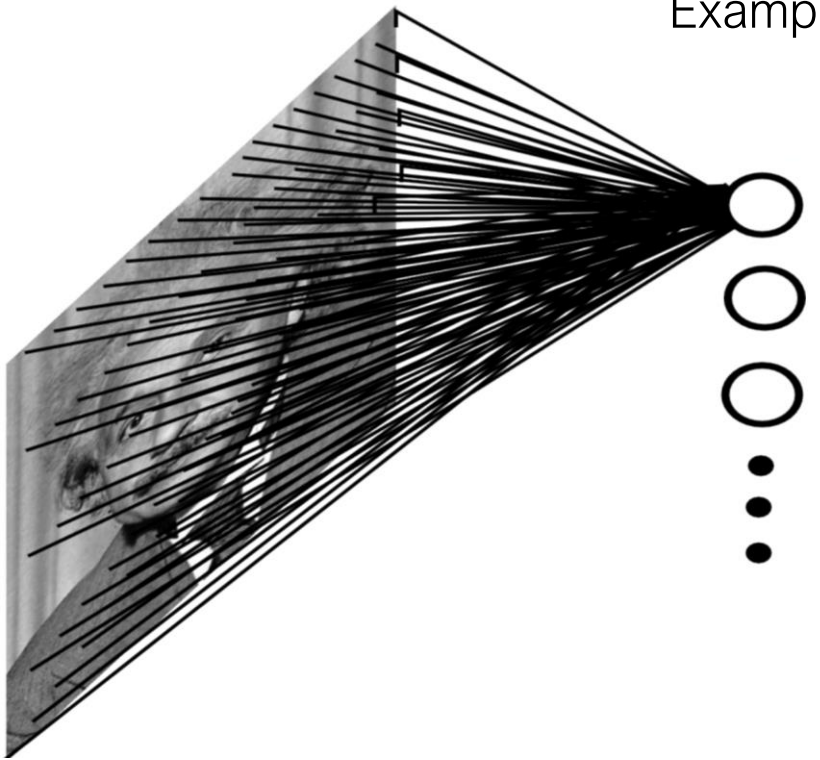
Example: 200x200 image

40k hidden units



Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)



Example: 200x200 image

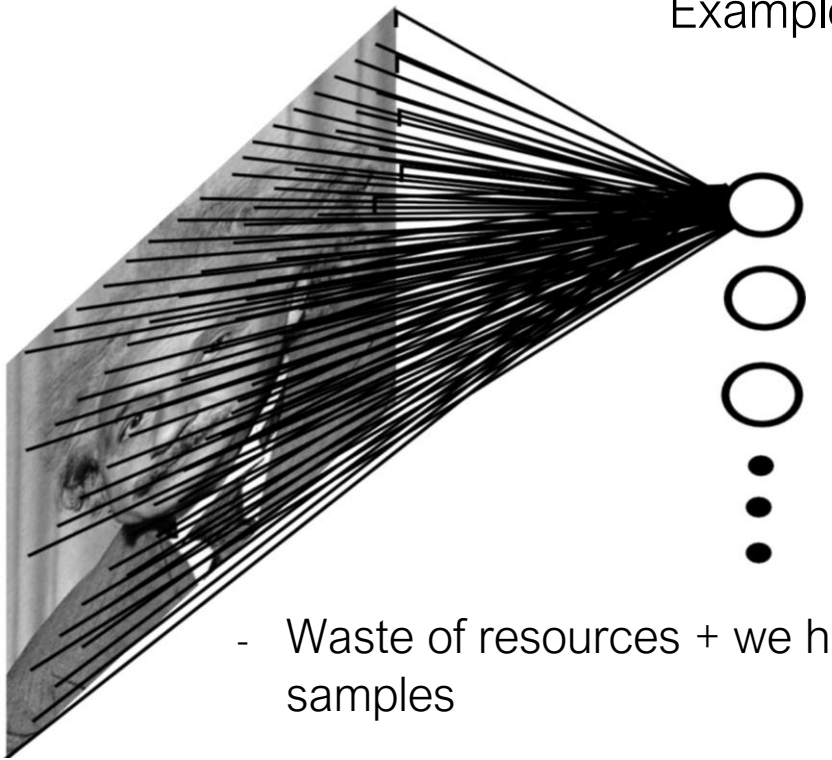
40k hidden units

~2B parameters!!!

Why?

Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)



Example: 200x200 image

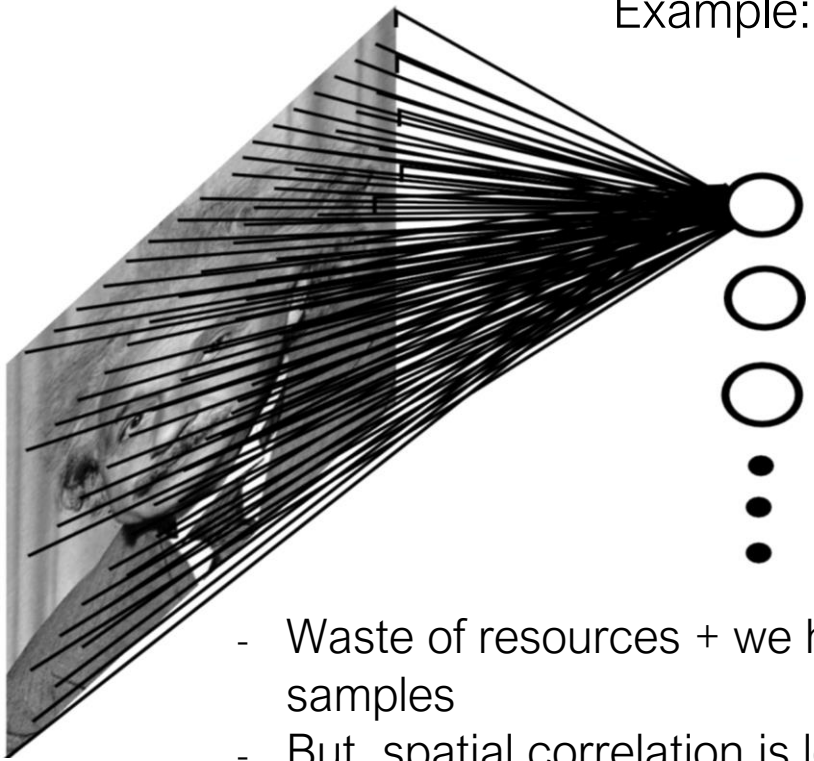
40k hidden units

~2B parameters!!!

- Waste of resources + we have not enough training samples

Slide credit: Marc'Aurelio Ranzato

Fully connected layer (FC)



Example: 200x200 pixel image

40k hidden units

~2B parameters!!!

- Waste of resources + we have not enough training samples
- But, spatial correlation is local!

Slide credit: Marc'Aurelio Ranzato

Recap: Neural Networks

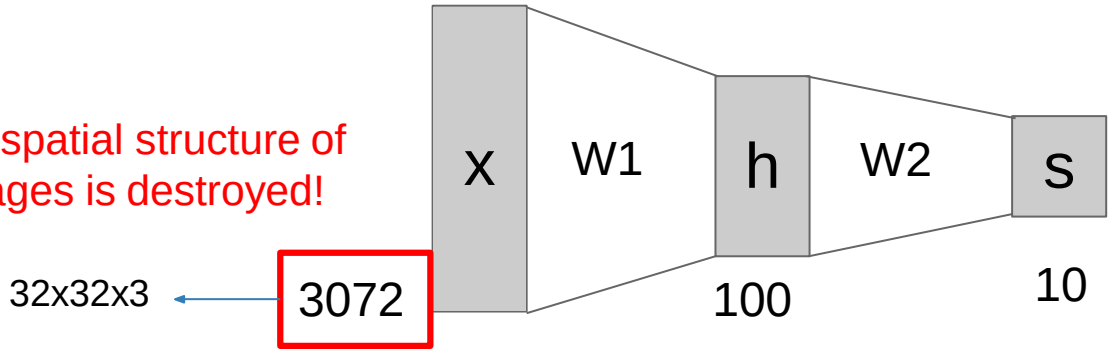
Linear score function:

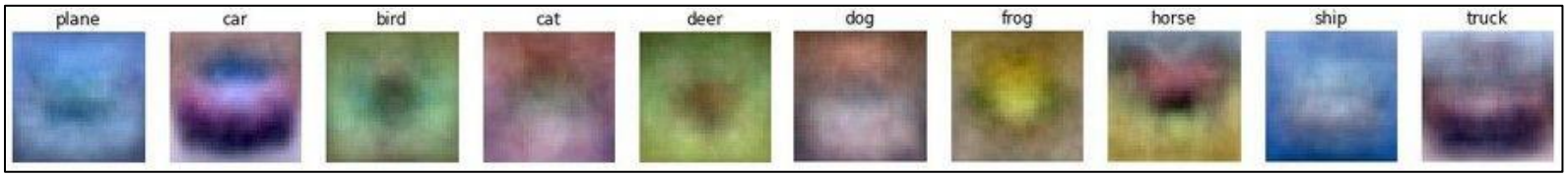
$$f = Wx$$

2-layer Neural Network

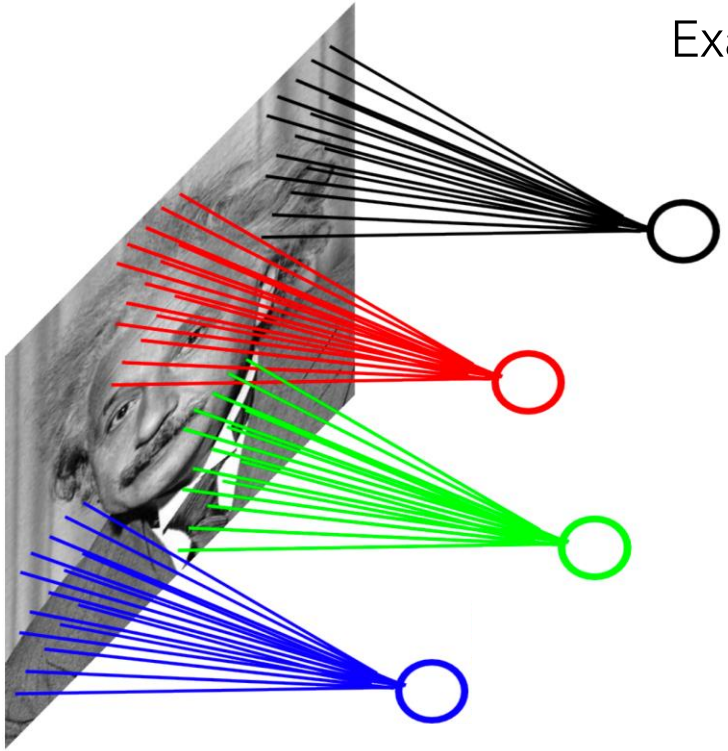
$$f = W_2 \max(0, W_1 x)$$

The spatial structure of images is destroyed!





Locally connected layer



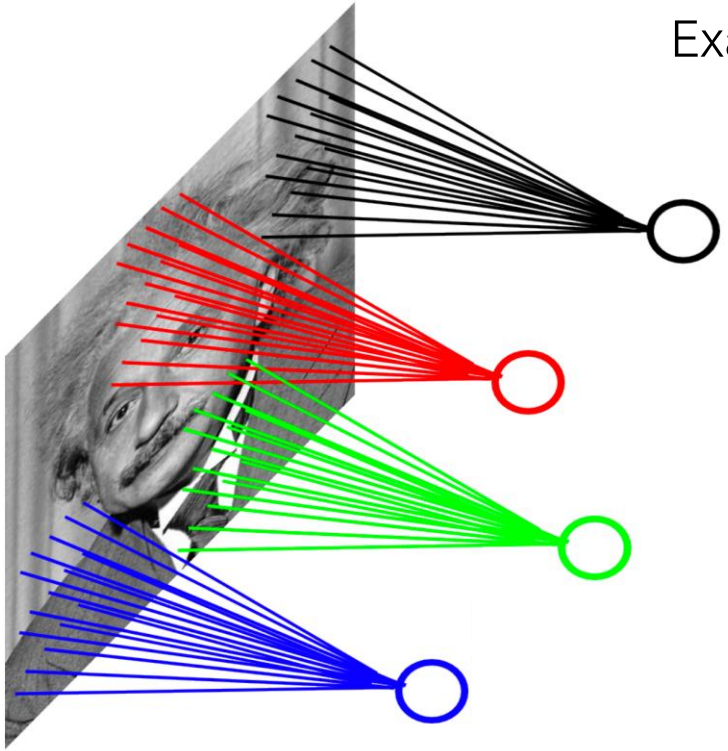
Example: 200x200 image

40k hidden units

Filter size: 10x10

Slide credit: Marc'Aurelio Ranzato

Locally connected layer



Example: 200x200 image

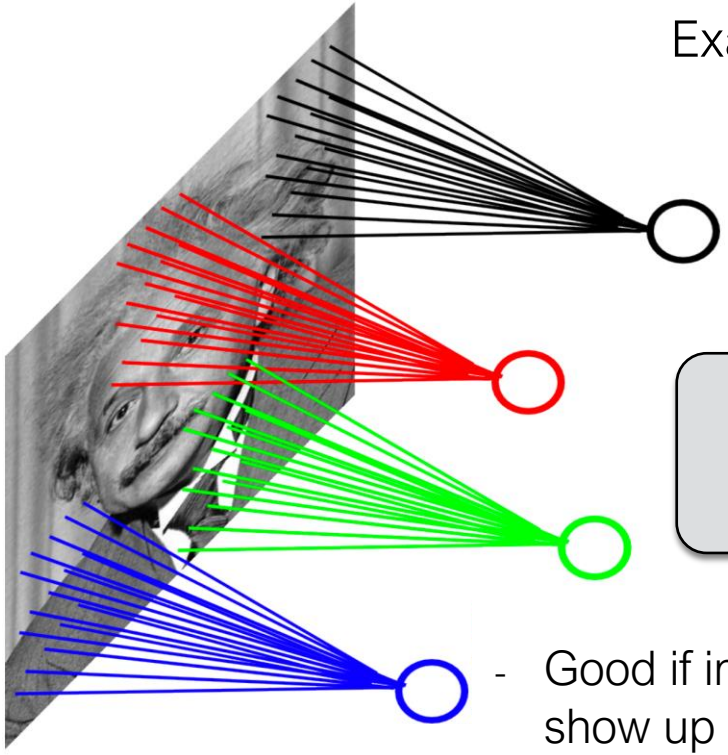
40k hidden units

Filter size: 10x10

4M parameters

Slide credit: Marc'Aurelio Ranzato

Locally connected layer



Example: 200x200 image

40k hidden units

Filter size: 10x10 pixels

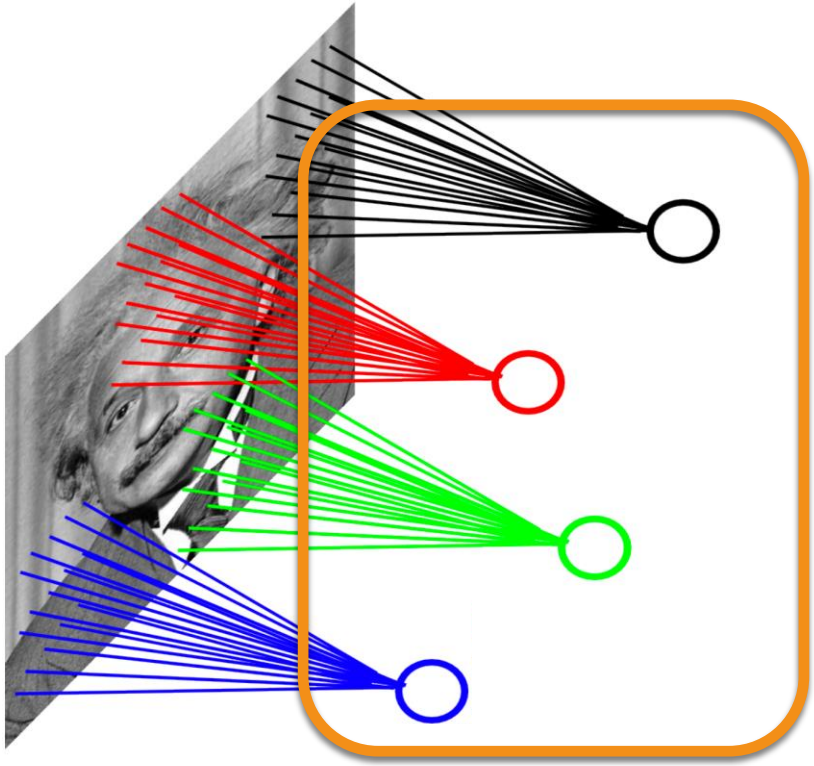
4M parameters (40k x 100)

What can we do?

- Good if images are registered (i.e. parts show up at the same locations)

Slide credit: Marc'Aurelio Ranzato

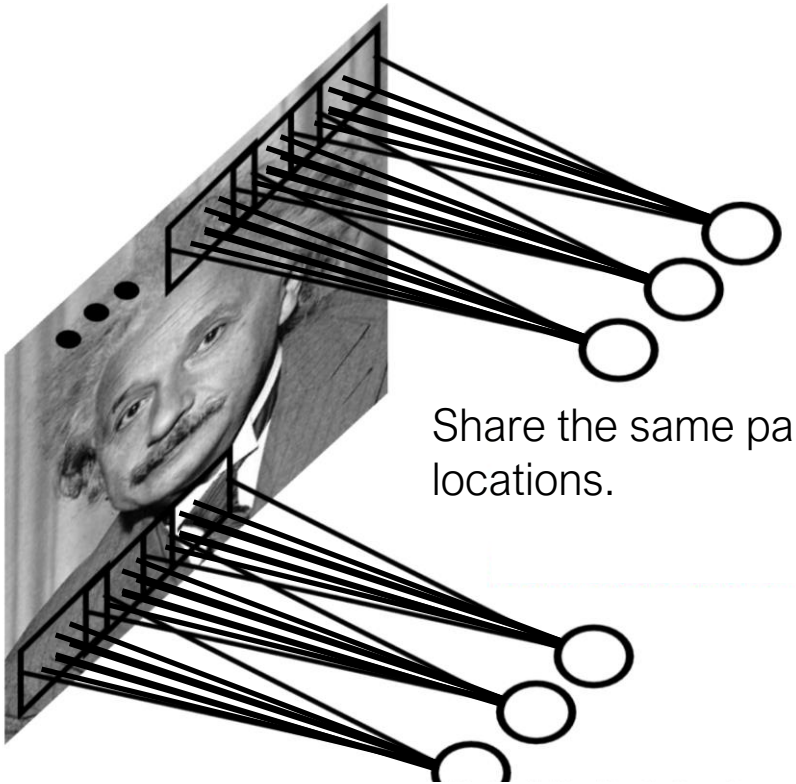
Locally connected layer



Share all local weights

Slide credit: Marc'Aurelio Ranzato

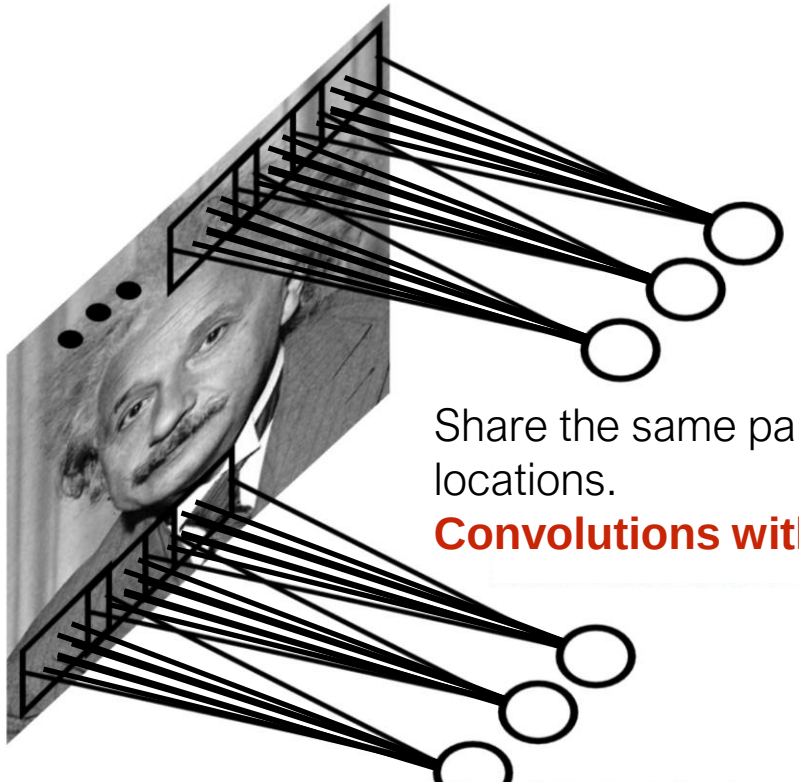
Convolutional layer



Share the same parameters across different locations.

Slide credit: Marc'Aurelio Ranzato

Convolutional layer



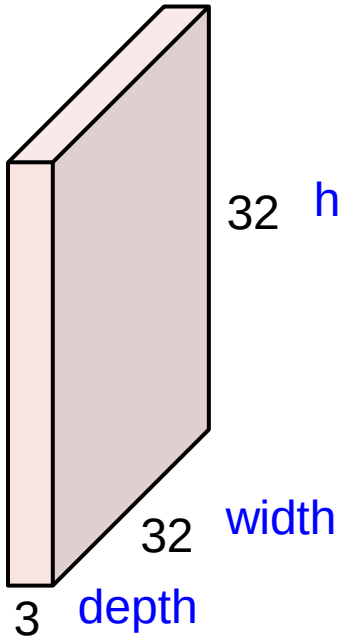
Share the same parameters across different locations.

Convolutions with learned kernels (weights)

Slide credit: Marc'Aurelio Ranzato

Convolutional layer

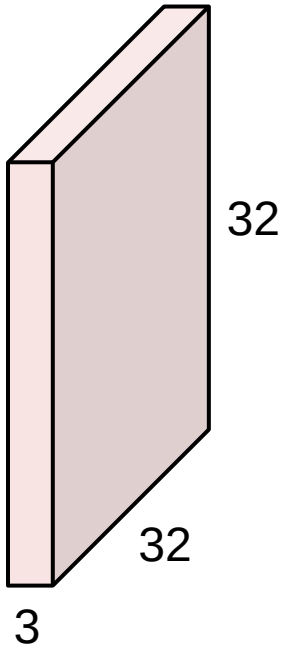
32x32x3 image -> preserve spatial structure



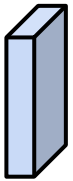
Slide credit: CS 231N

Convolutional layer

32x32x3 image



5x5x3 filter



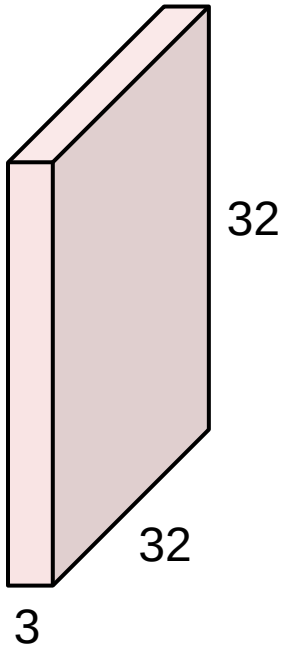
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS 231N

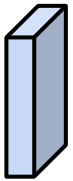
Convolutional layer

Filters always extend the full depth of the input volume

32x32x3 image



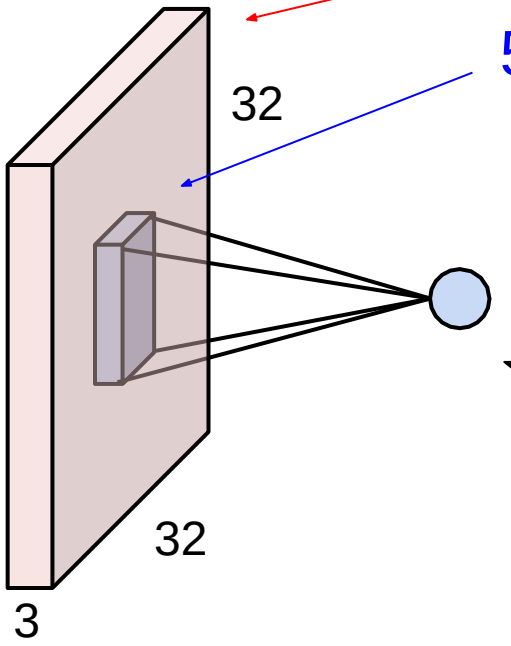
5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS 231N

Convolutional layer



32x32x3 image

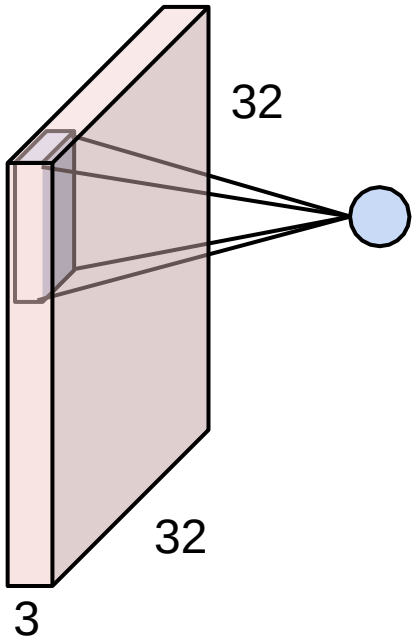
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

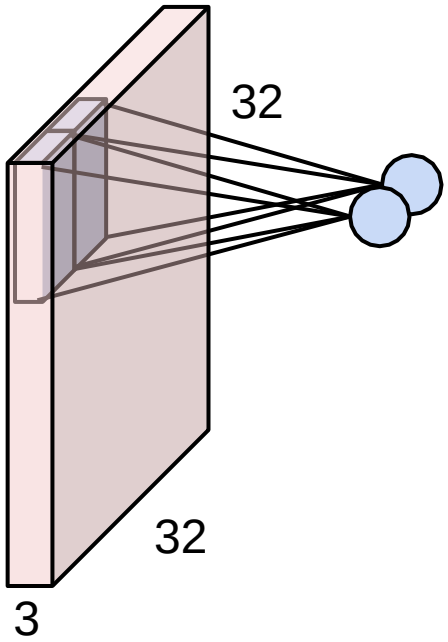
Slide credit: CS 231N

Convolutional layer



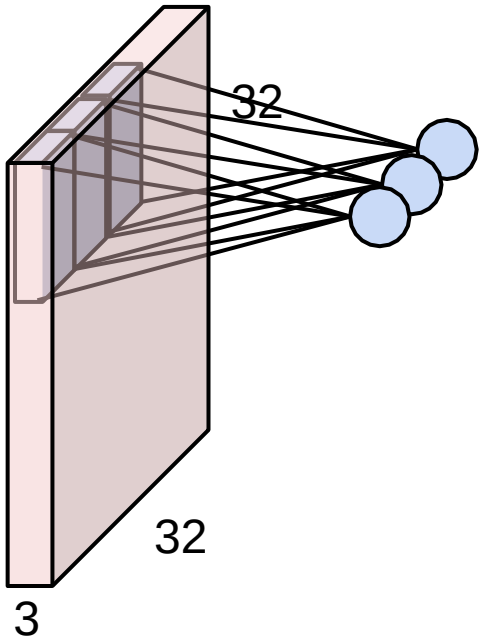
Slide credit: CS 231N

Convolutional layer



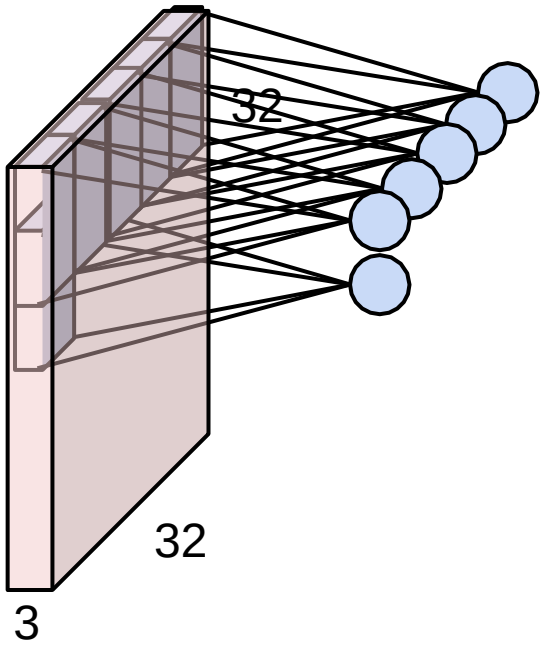
Slide credit: CS 231N

Convolutional layer



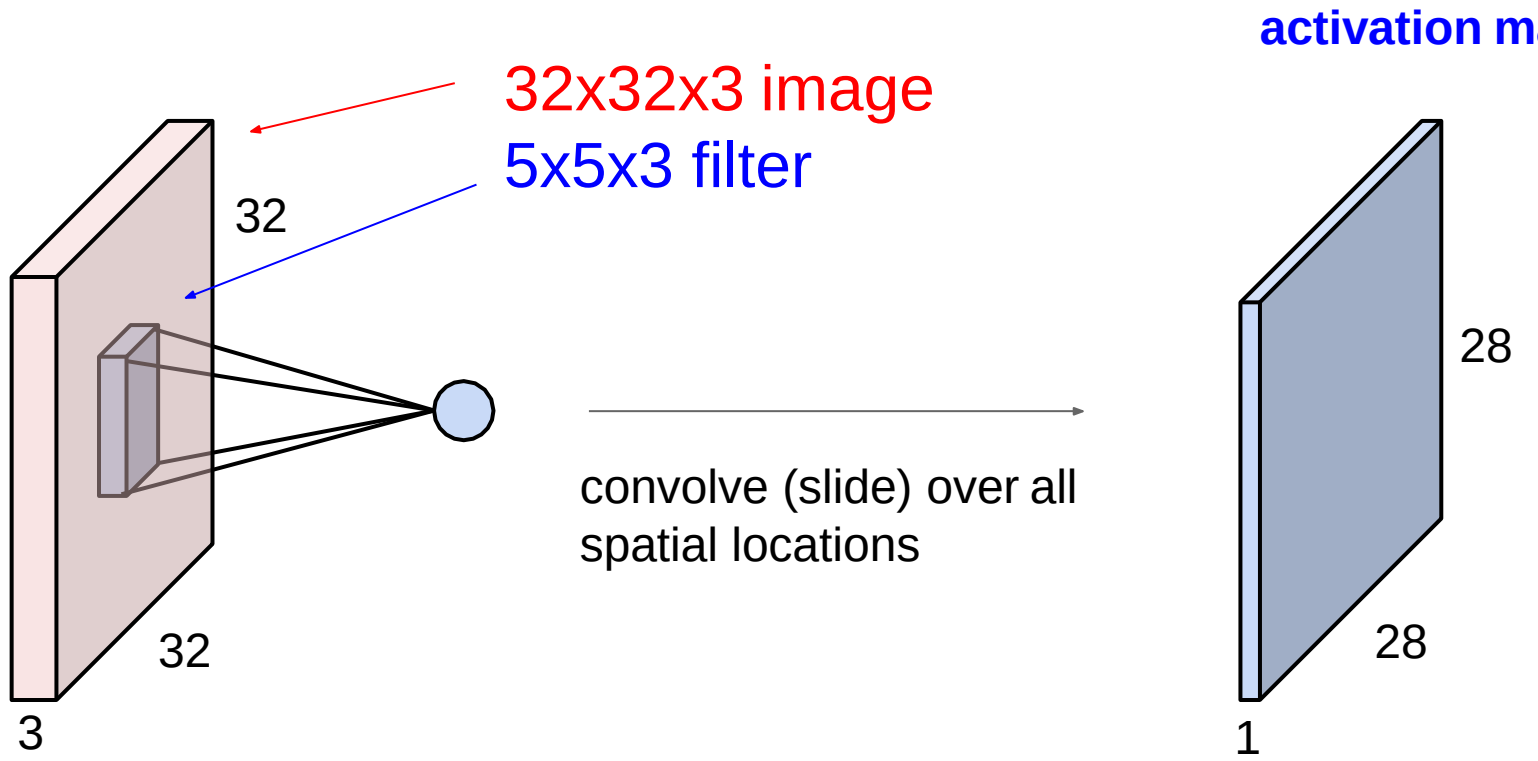
Slide credit: CS 231N

Convolutional layer



Slide credit: CS 231N

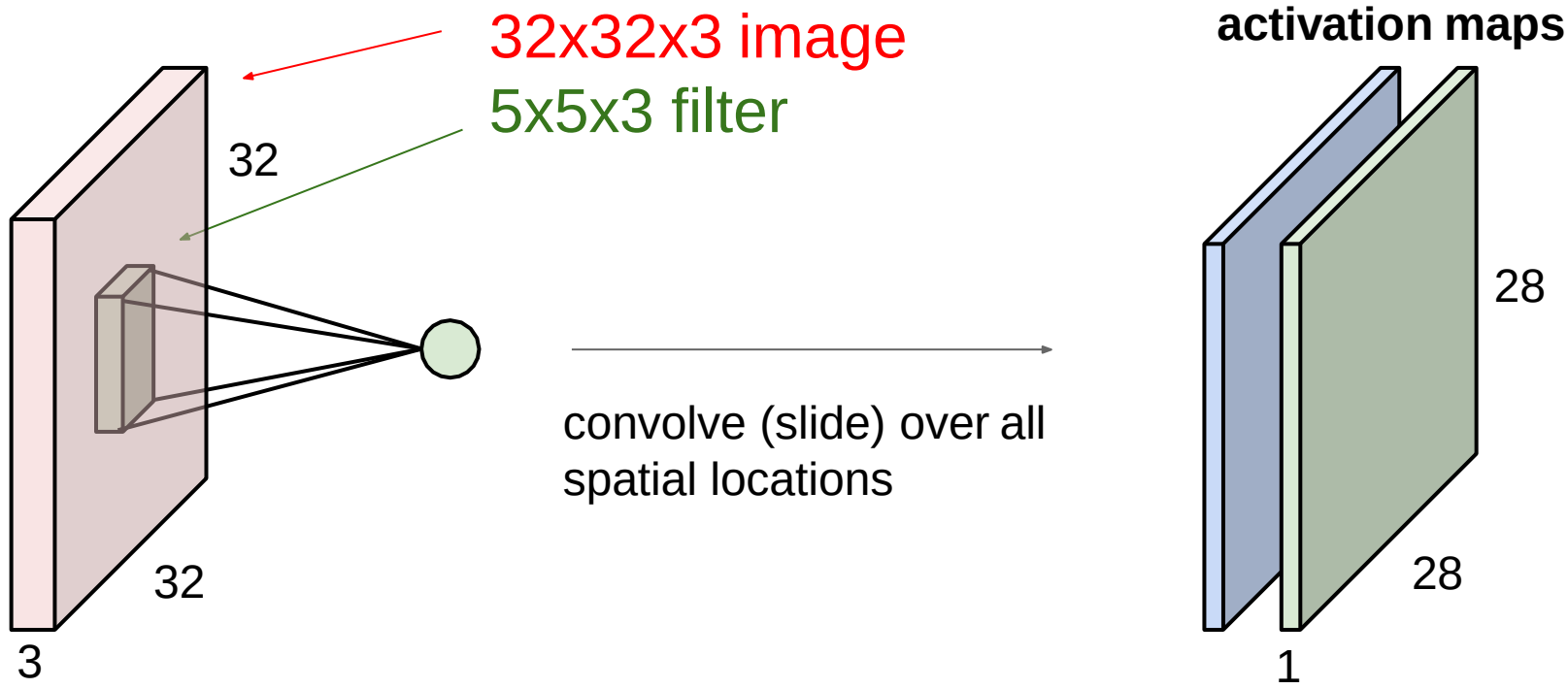
Convolutional layer



Slide credit: CS 231N

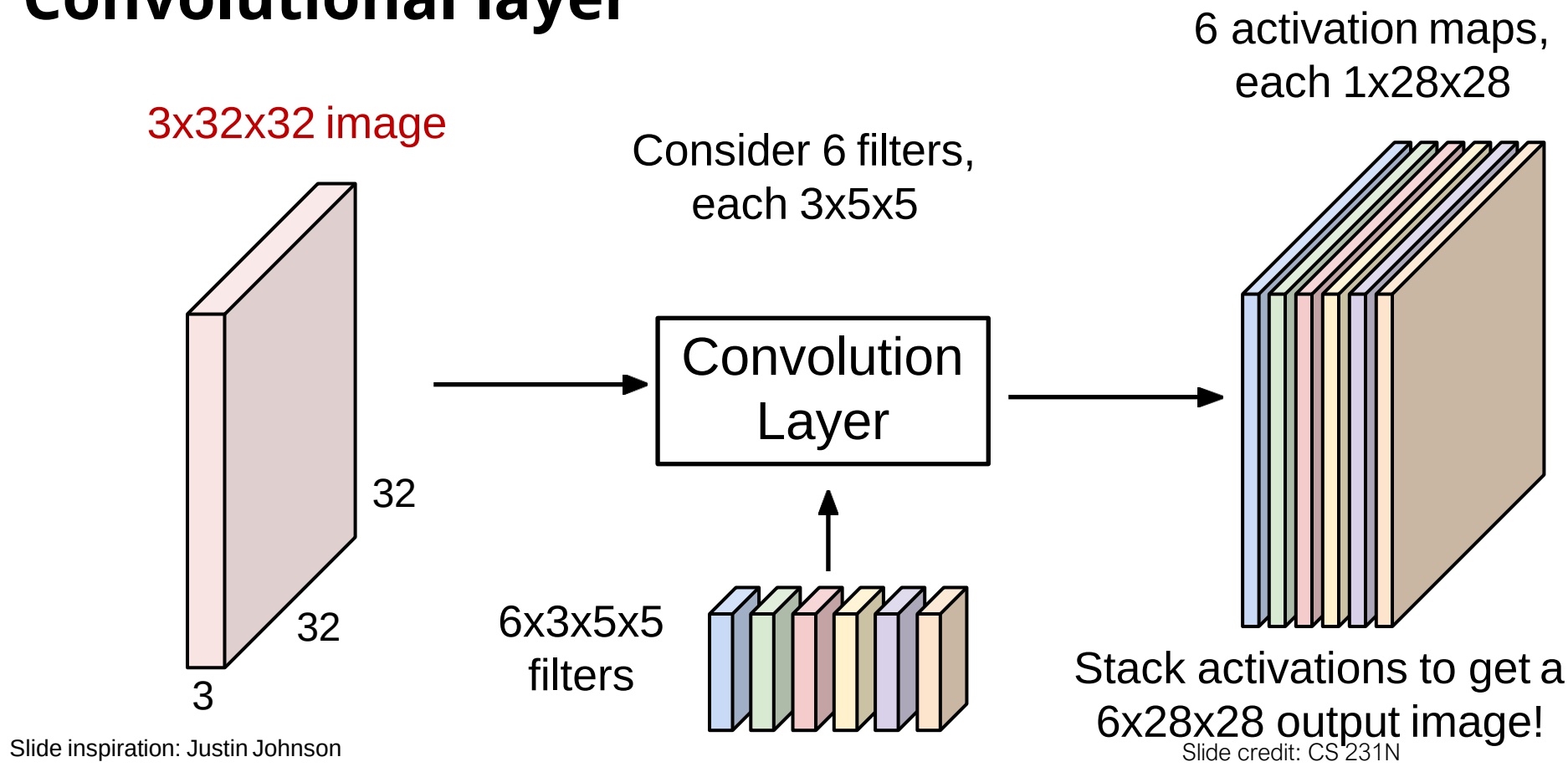
Convolutional layer

consider a second, green filter



Slide credit: CS 231N

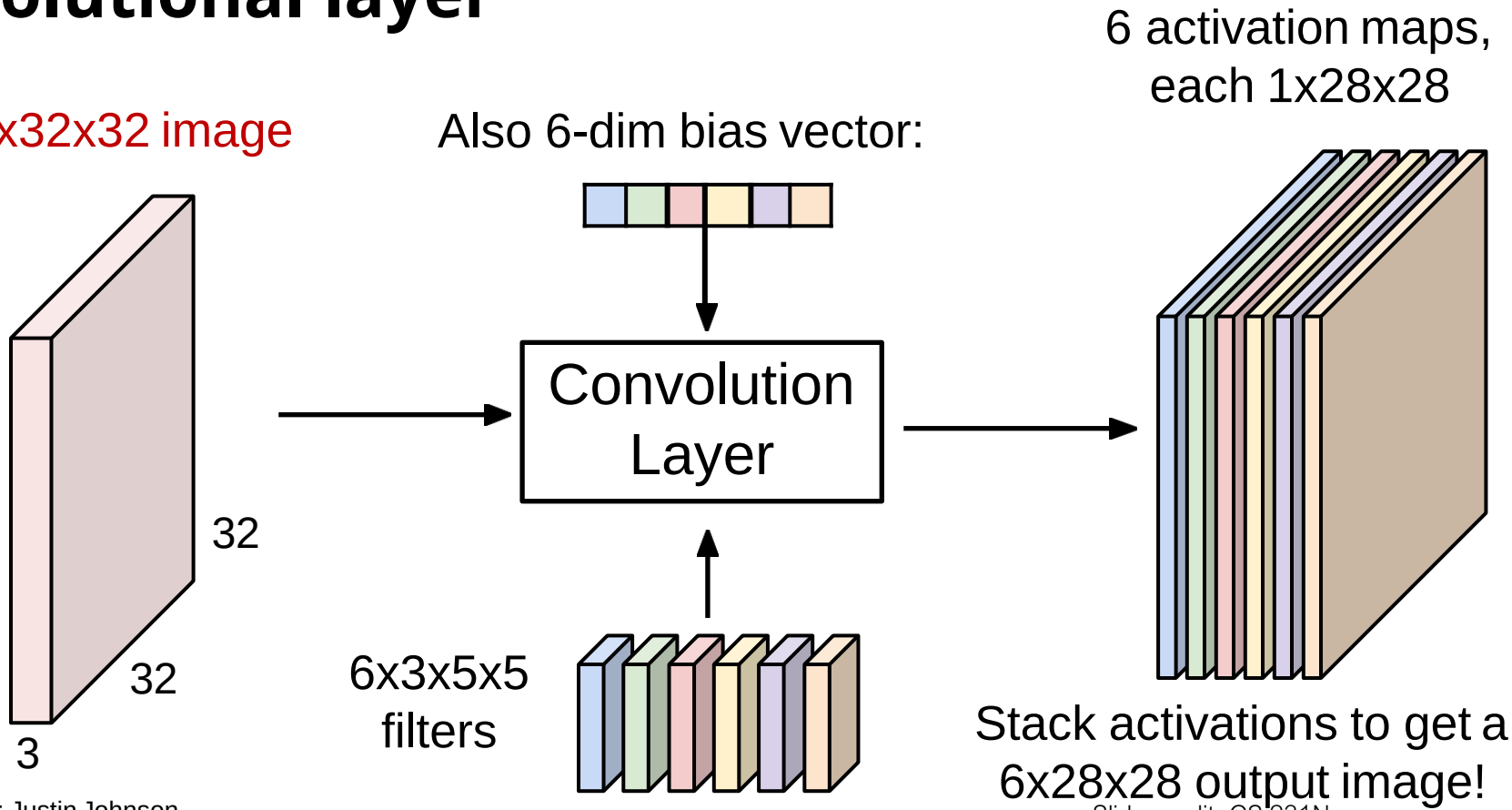
Convolutional layer



Slide inspiration: Justin Johnson

Slide credit: CS 231N

Convolutional layer



Slide inspiration: Justin Johnson

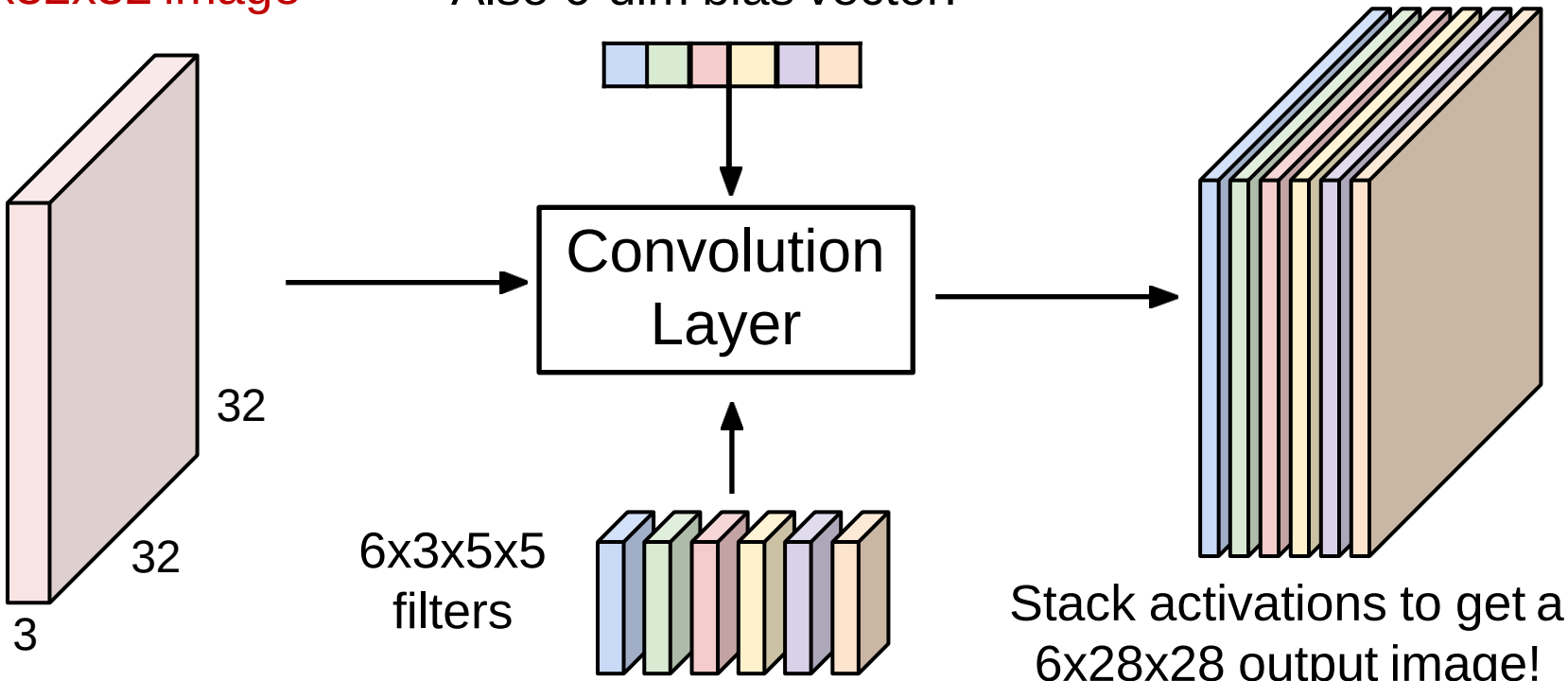
Slide credit: CS 231N

Convolutional layer

28x28 grid, at each point a 6-dim vector

3x32x32 image

Also 6-dim bias vector:

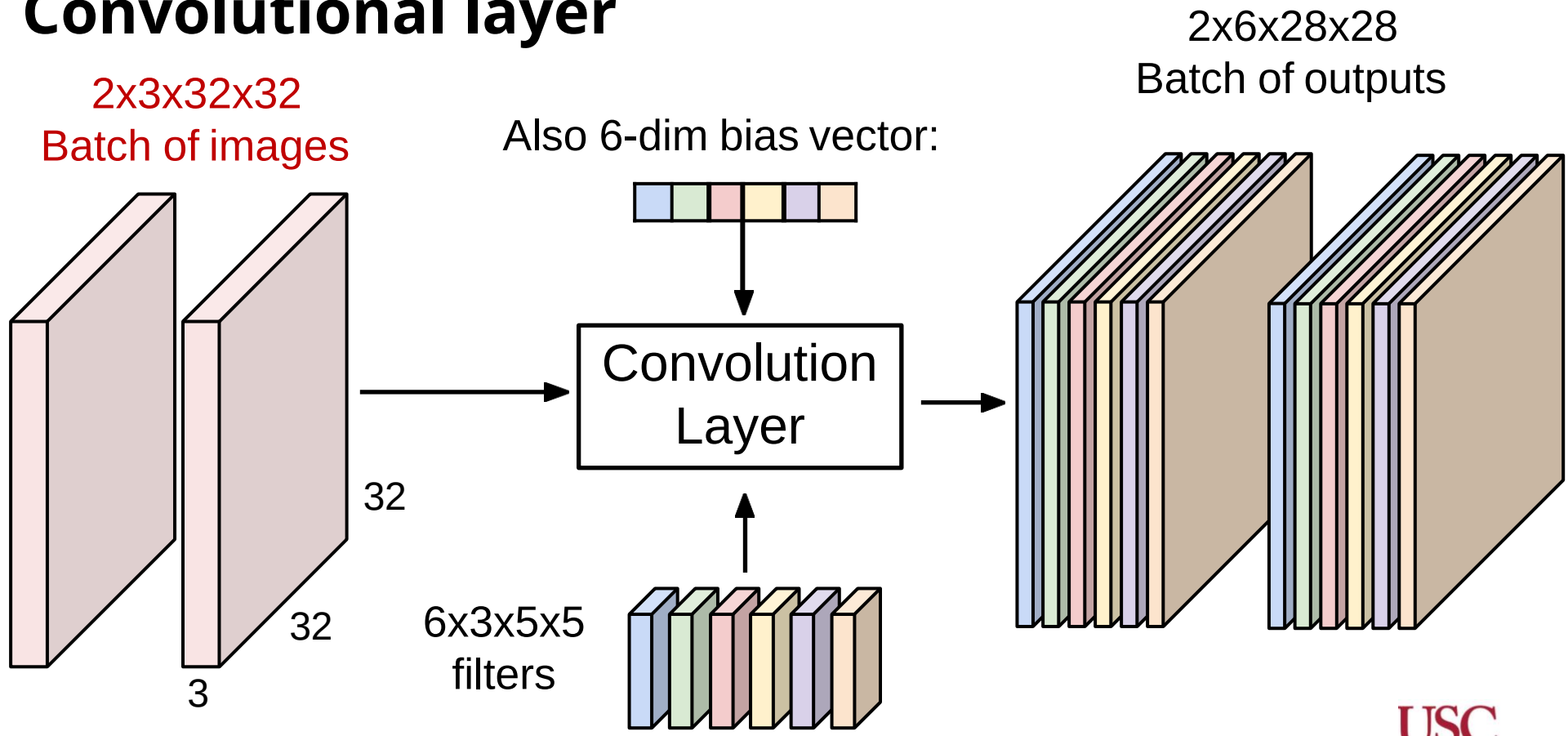


Stack activations to get a 6x28x28 output image!

Slide inspiration: Justin Johnson

Slide credit: CS 231N

Convolutional layer

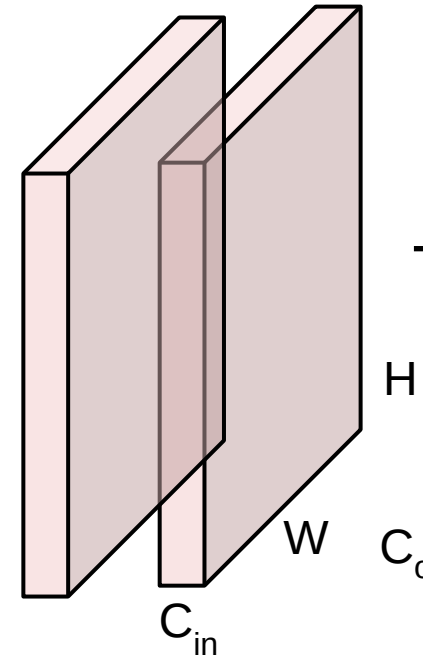


Slide inspiration: Justin Johnson

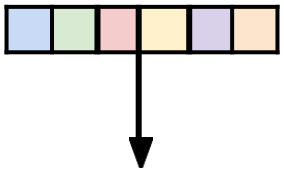
Slide credit: CS 231N

Convolutional layer

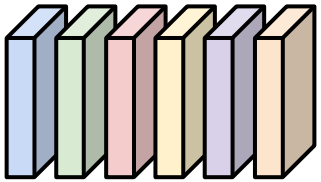
$N \times C_{in} \times H \times W$
Batch of images



Also C_{out}-dim bias vector:

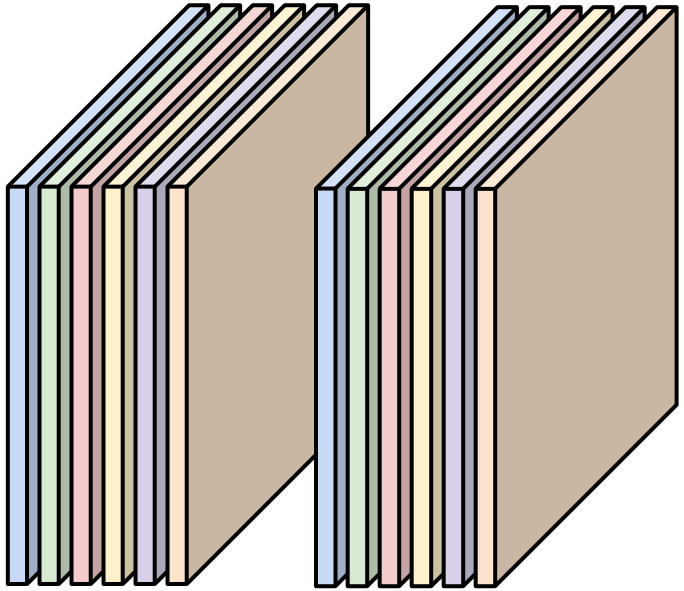


Convolution Layer



$C_{out} \times C_{in} \times K_w \times K_h$
filters

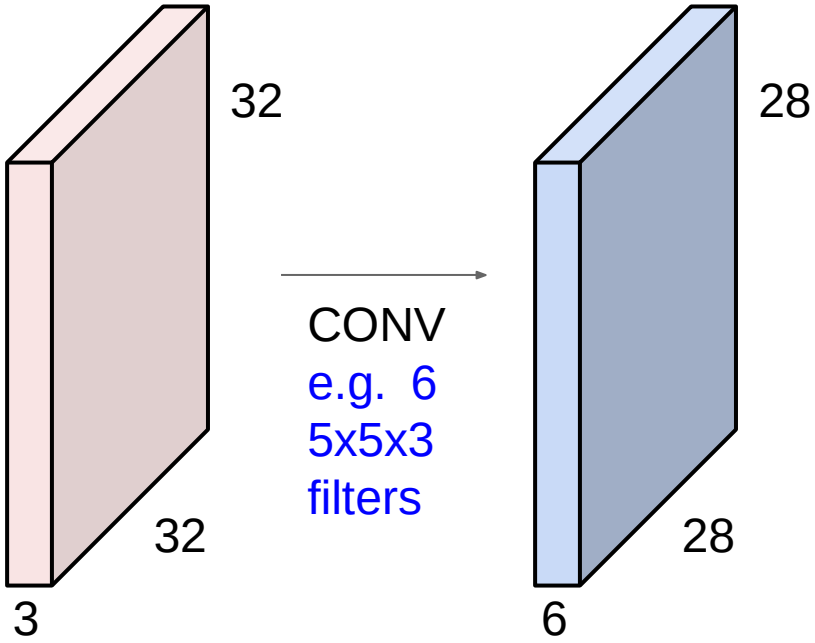
$N \times C_{out} \times H' \times W'$
Batch of outputs



Slide inspiration: Justin Johnson

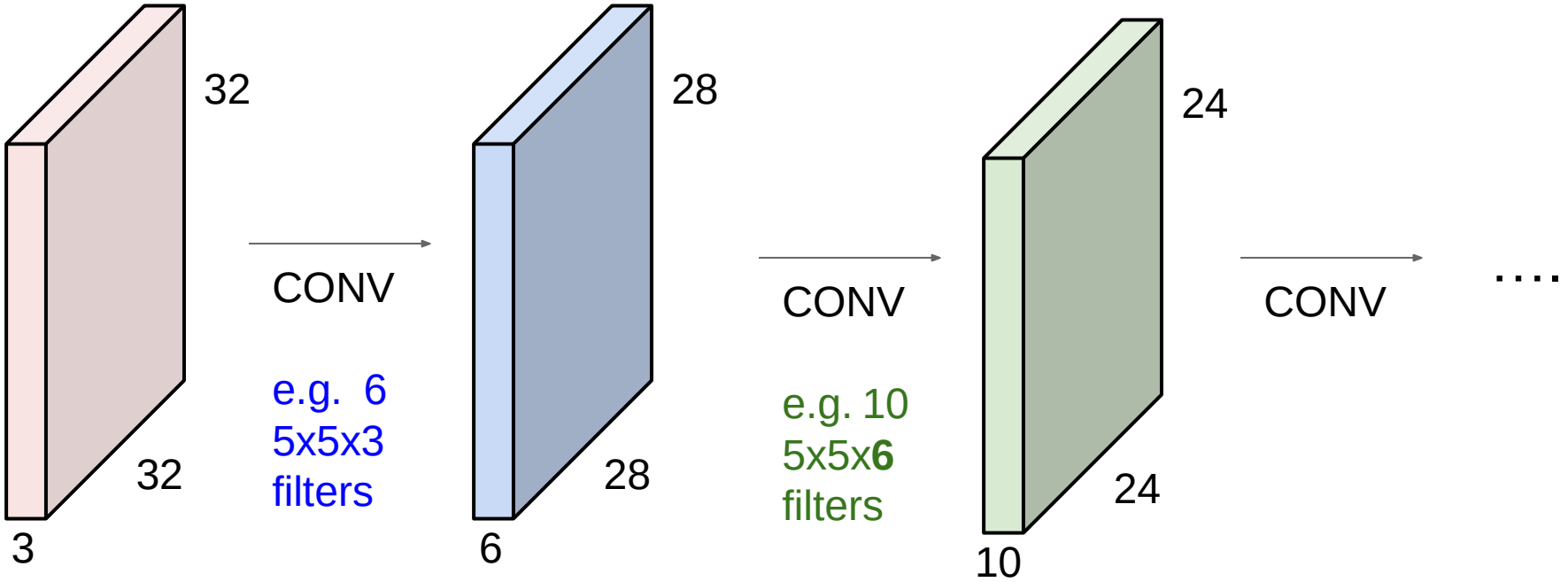
Slide credit: CS 231N

Preview: ConvNet is a sequence of Convolution Layers



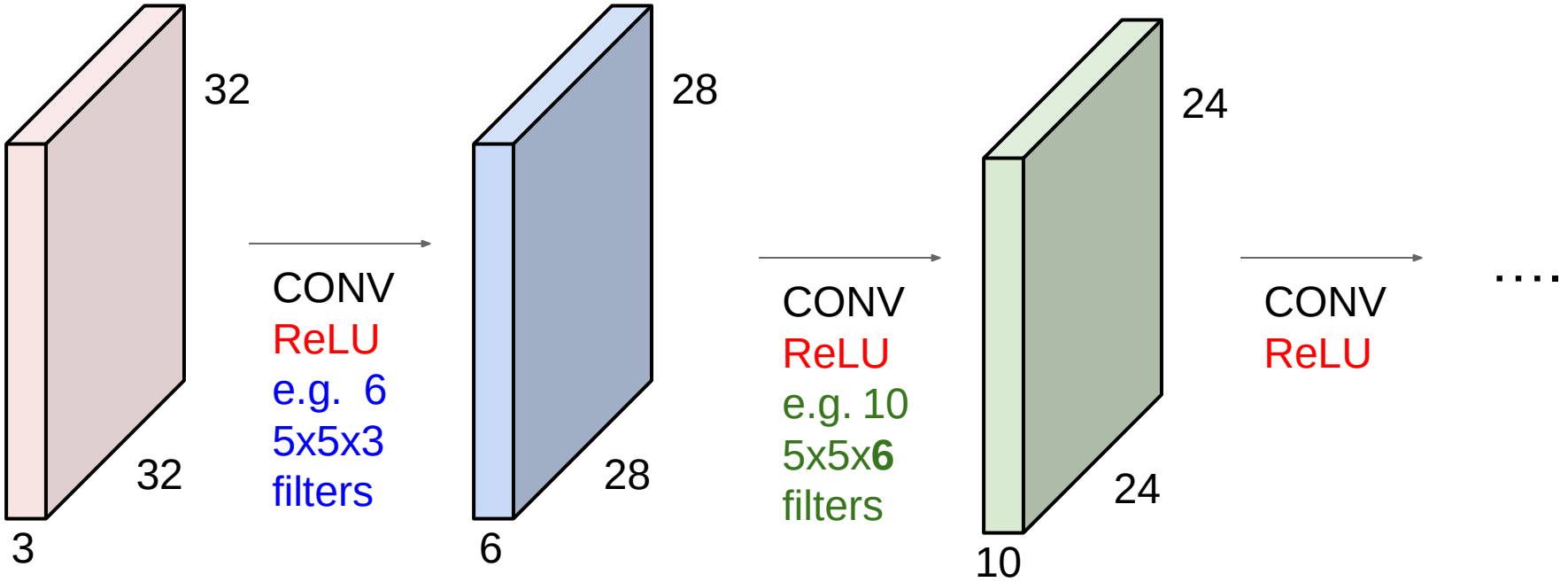
Slide credit: CS 231N

Preview: ConvNet is a sequence of Convolution Layers



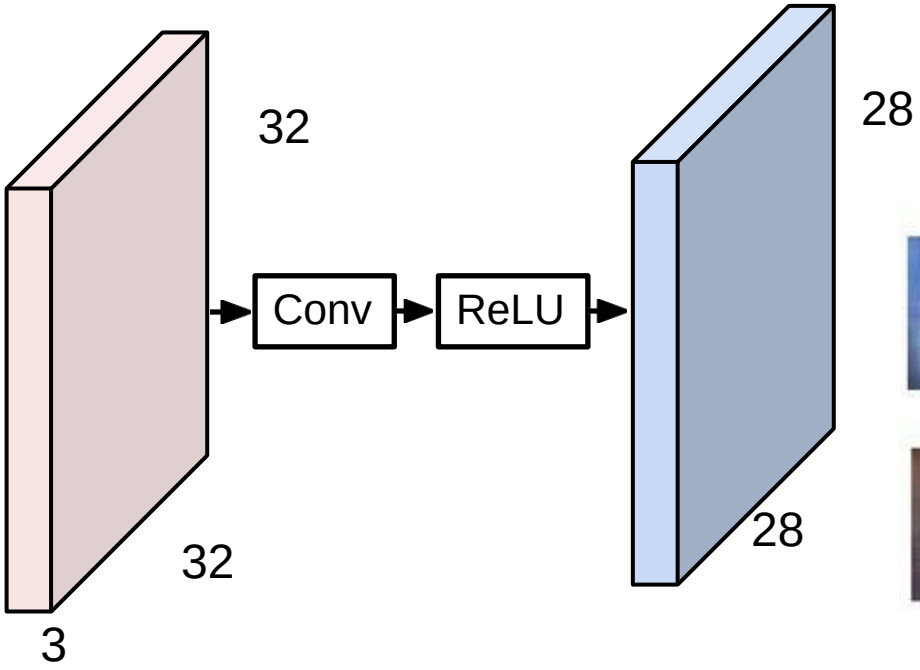
Slide credit: CS 231N

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Slide credit: CS 231N

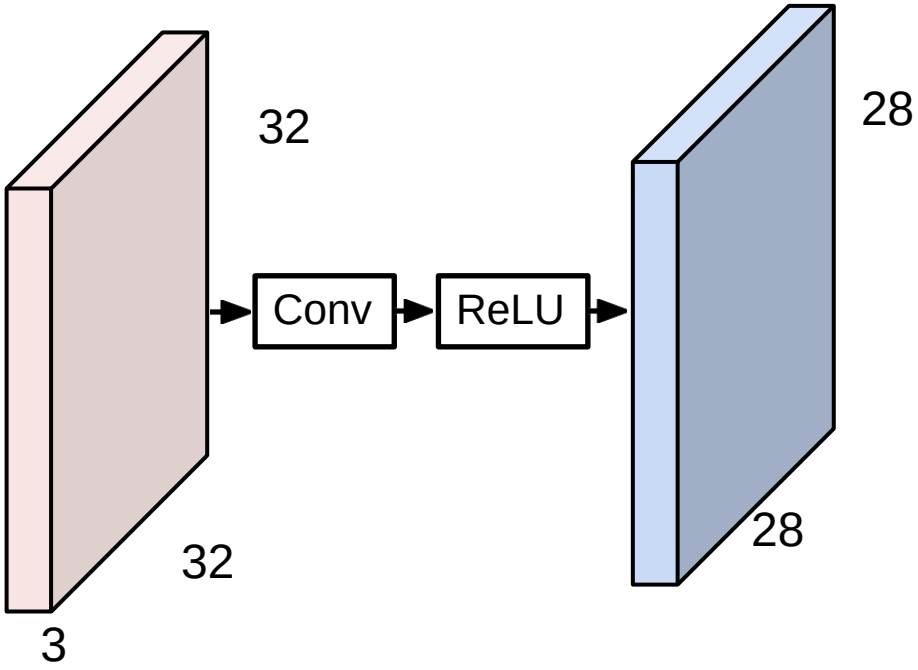
Preview: What do convolutional filters learn?



Linear classifier: One template per class

Slide credit: CS 231N

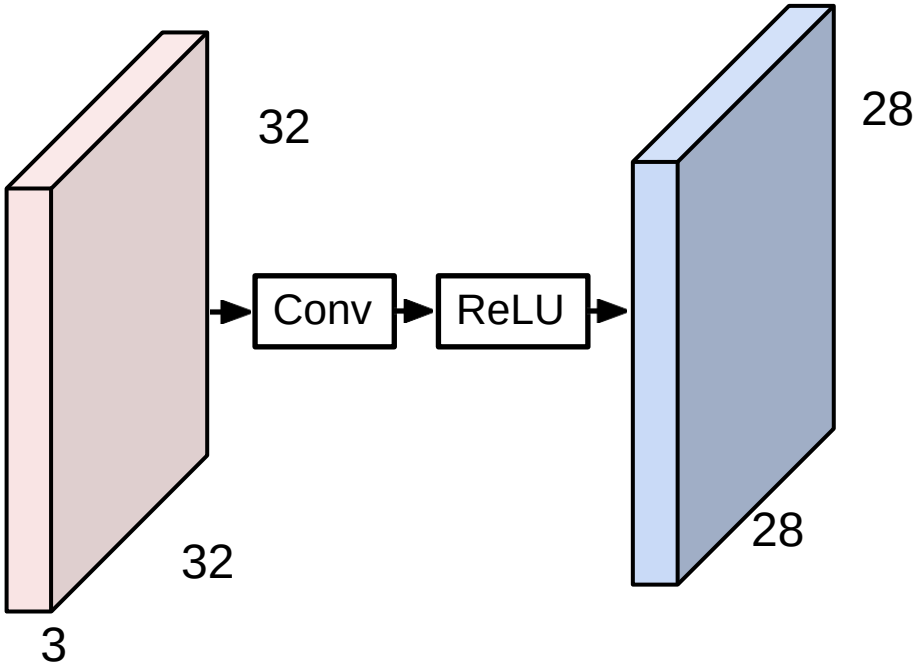
Preview: What do convolutional filters learn?



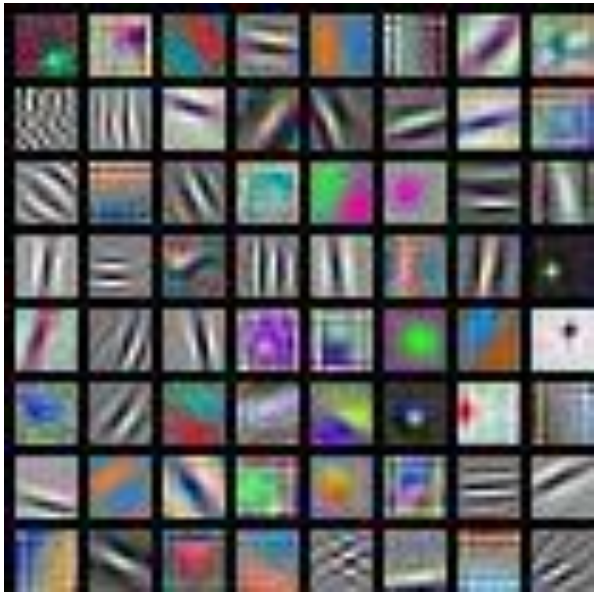
MLP: Bank of whole-image templates

Slide credit: CS 231N

Preview: What do convolutional filters learn?



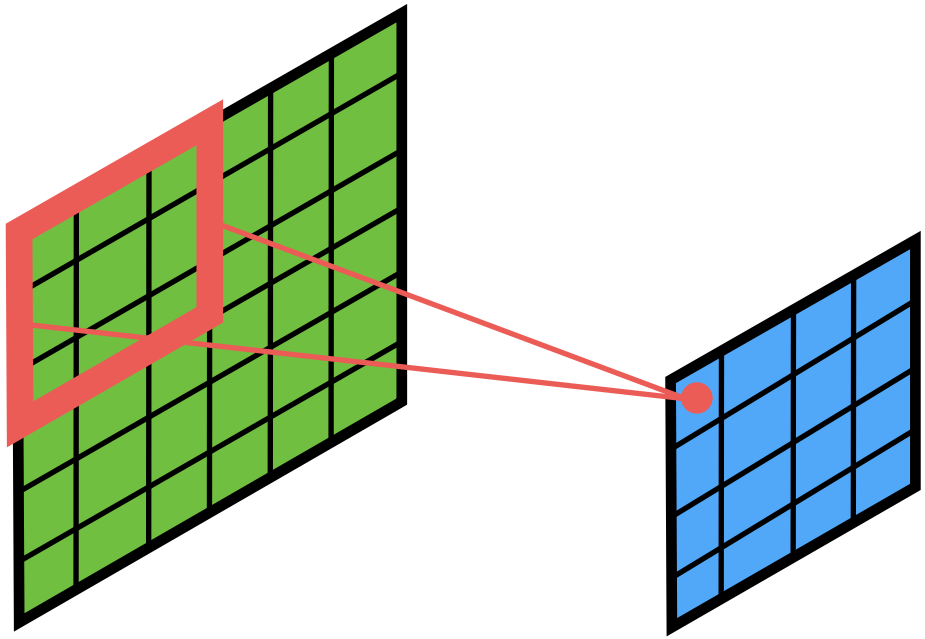
First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11

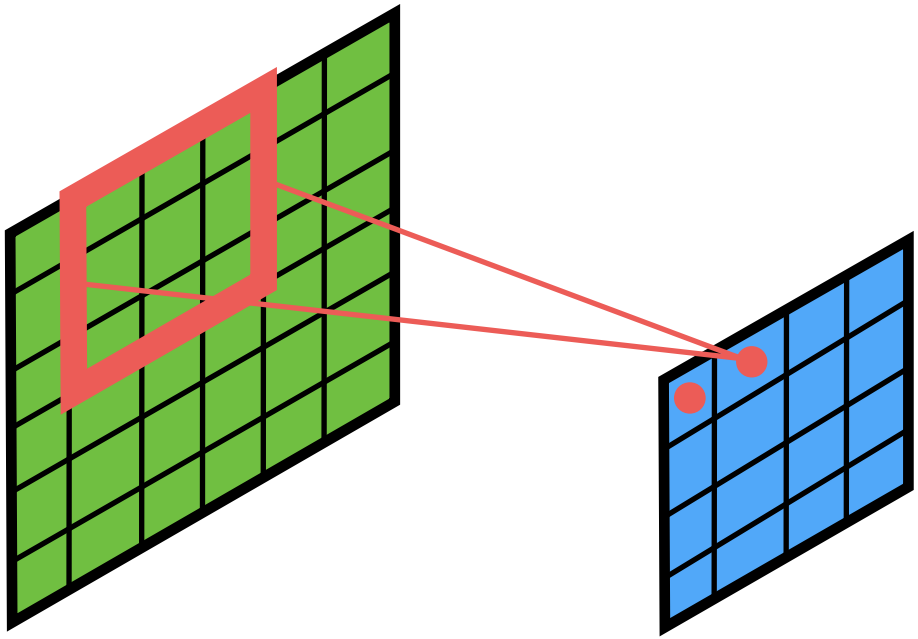
Slide credit: CS 231N

Convolutional layer



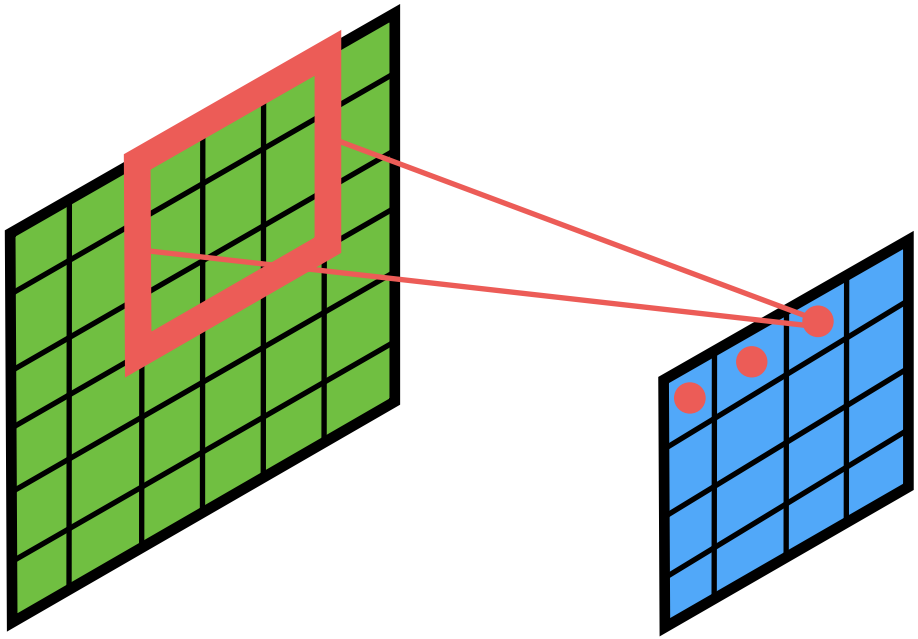
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



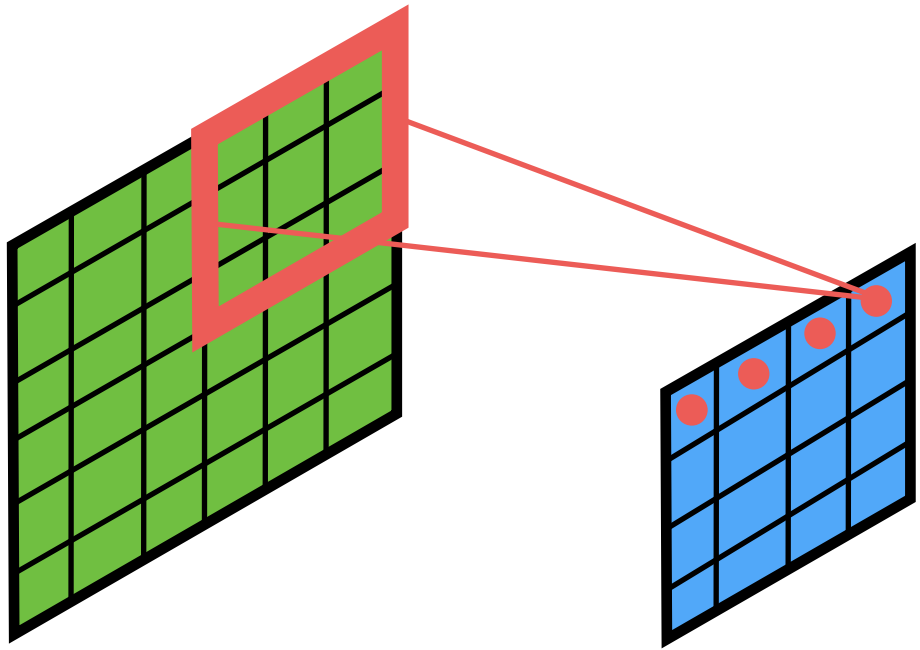
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



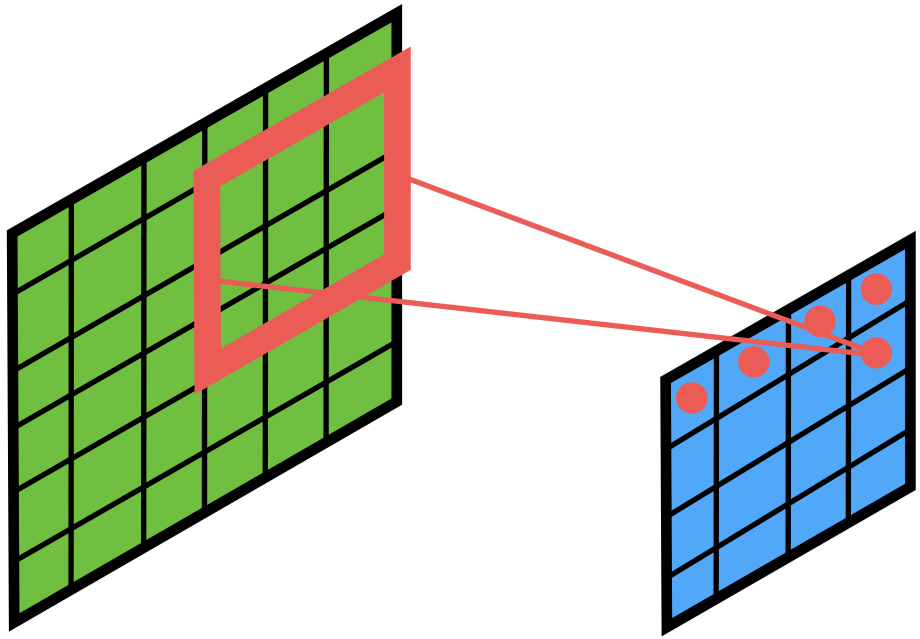
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



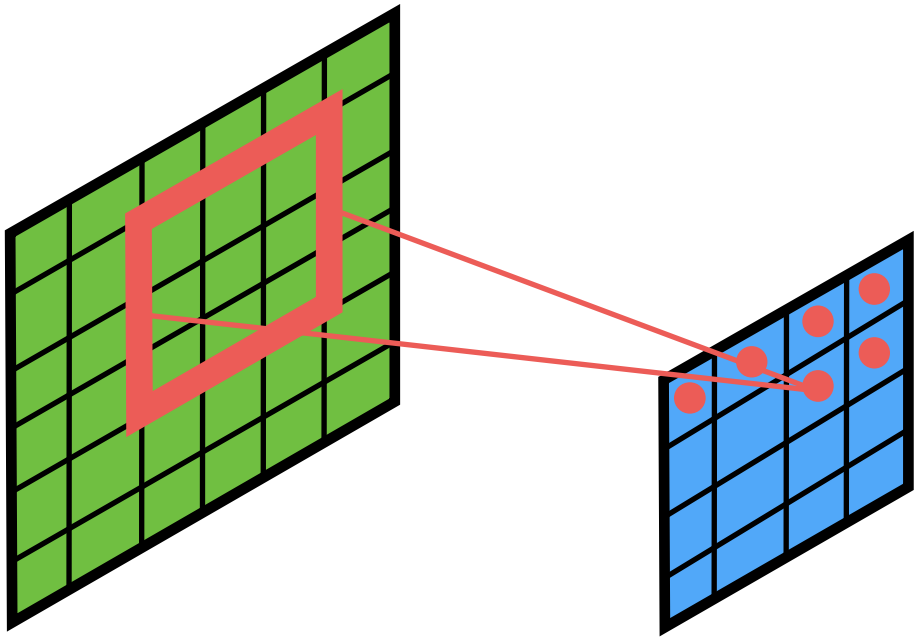
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



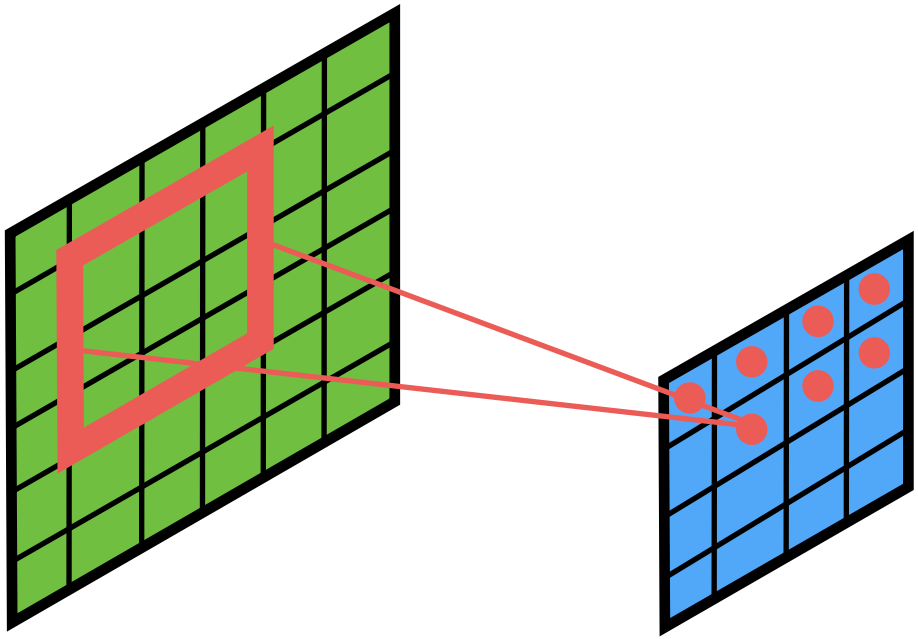
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



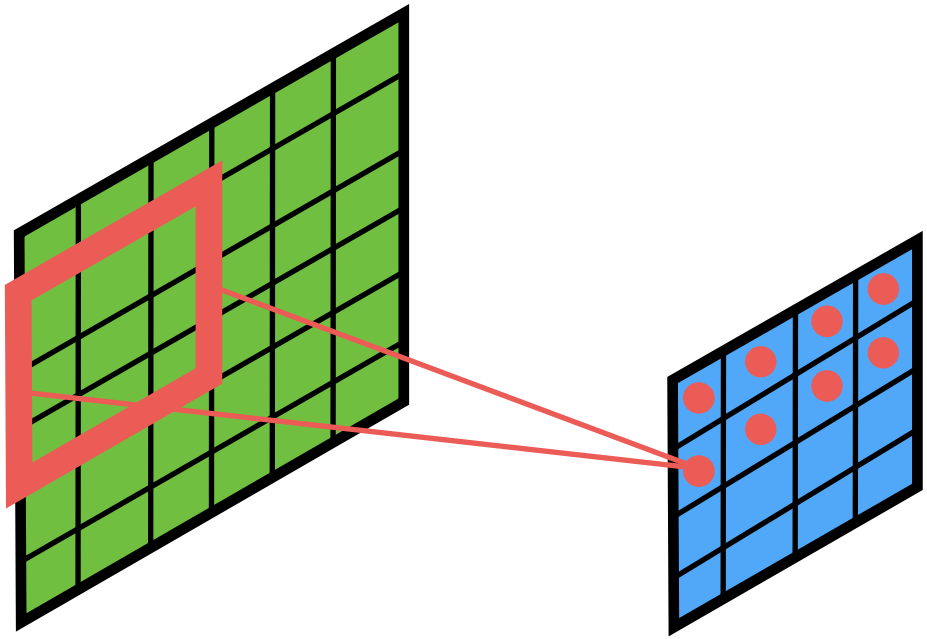
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



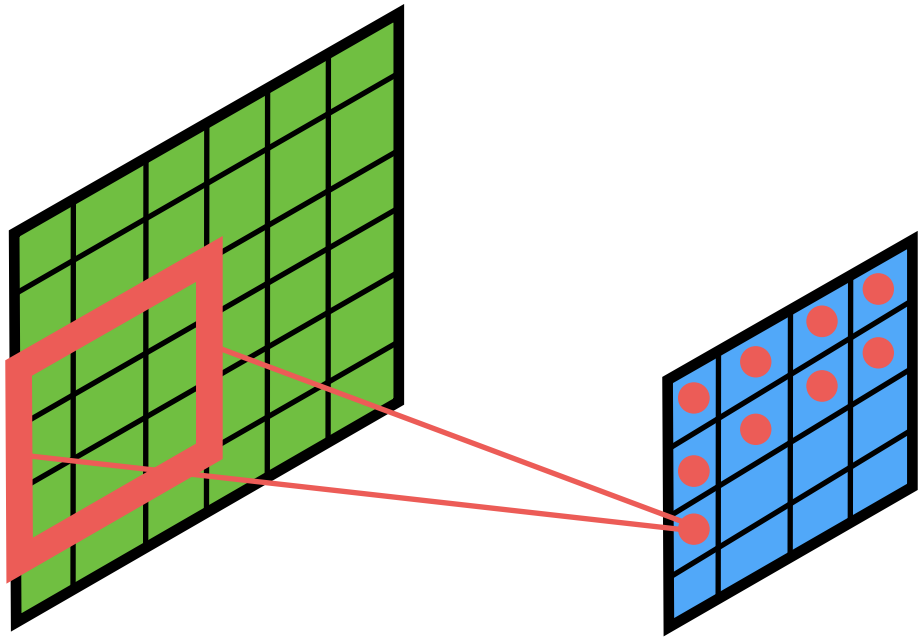
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



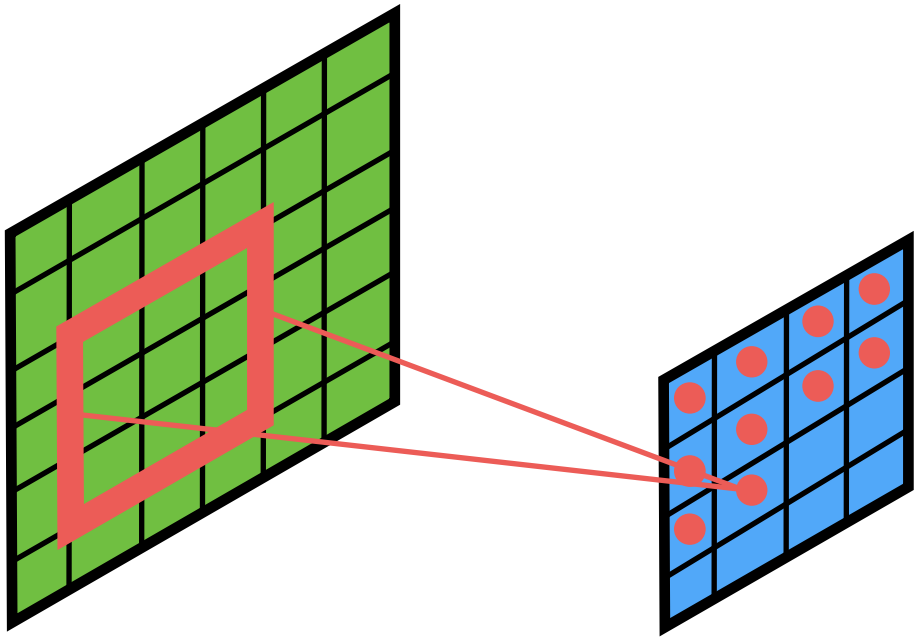
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



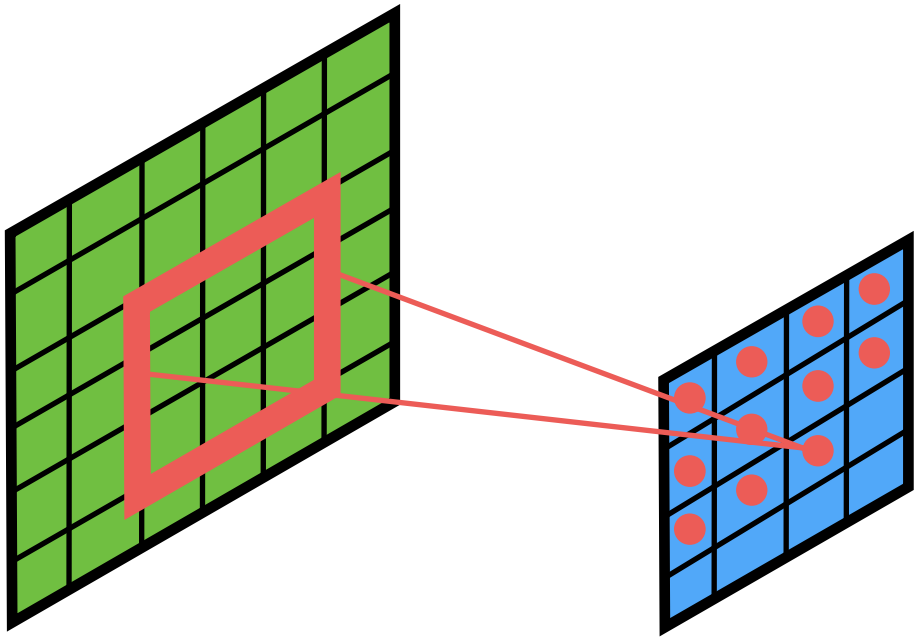
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



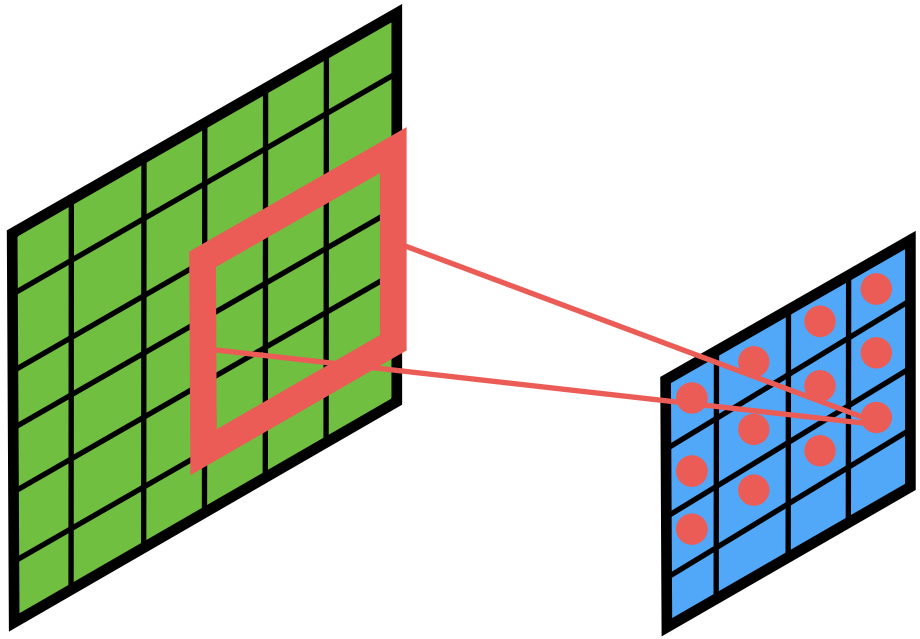
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



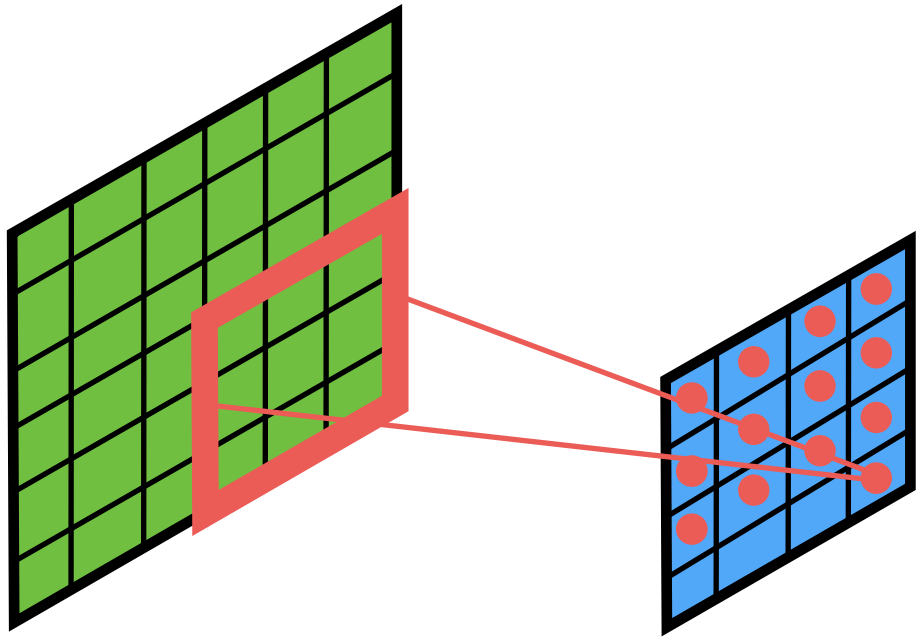
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



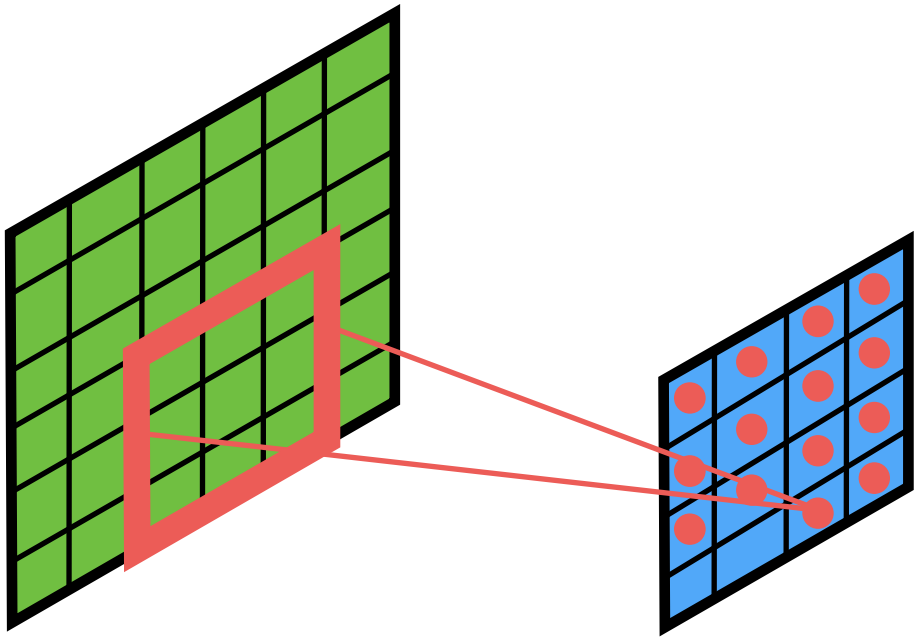
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



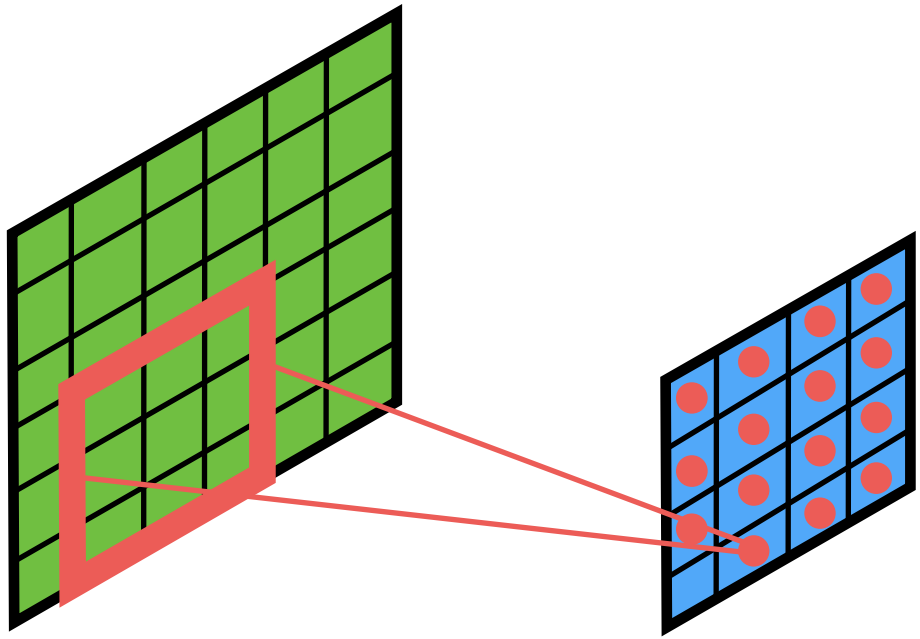
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



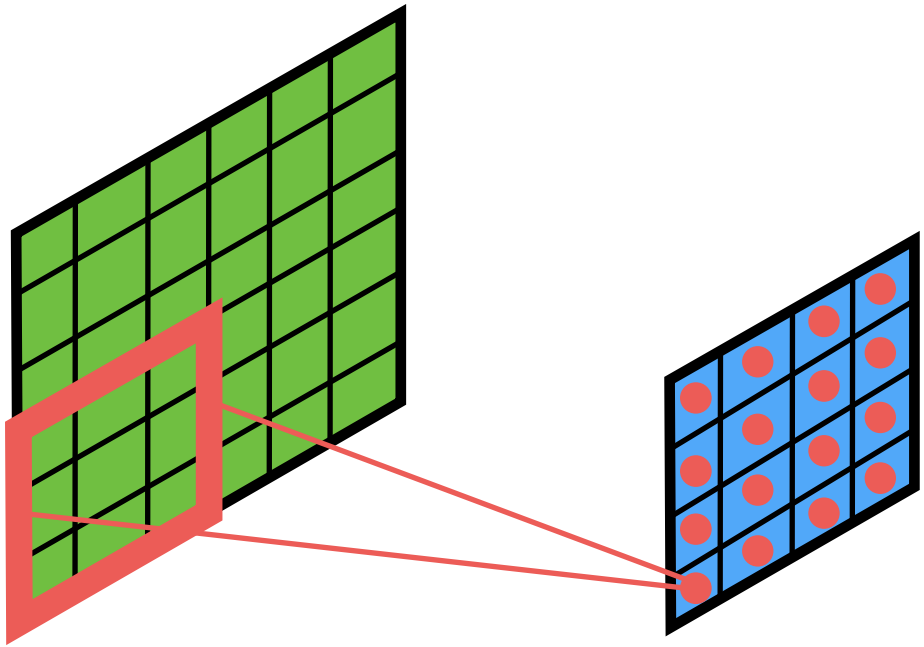
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer



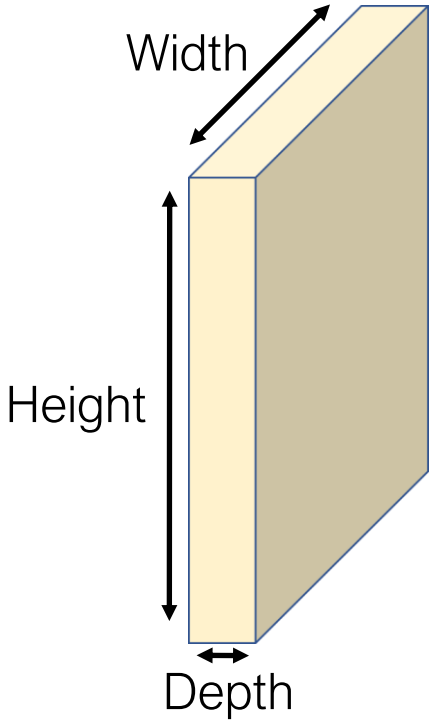
Inspired by Marc'Aurelio Ranzato's slides

Convolutional layer

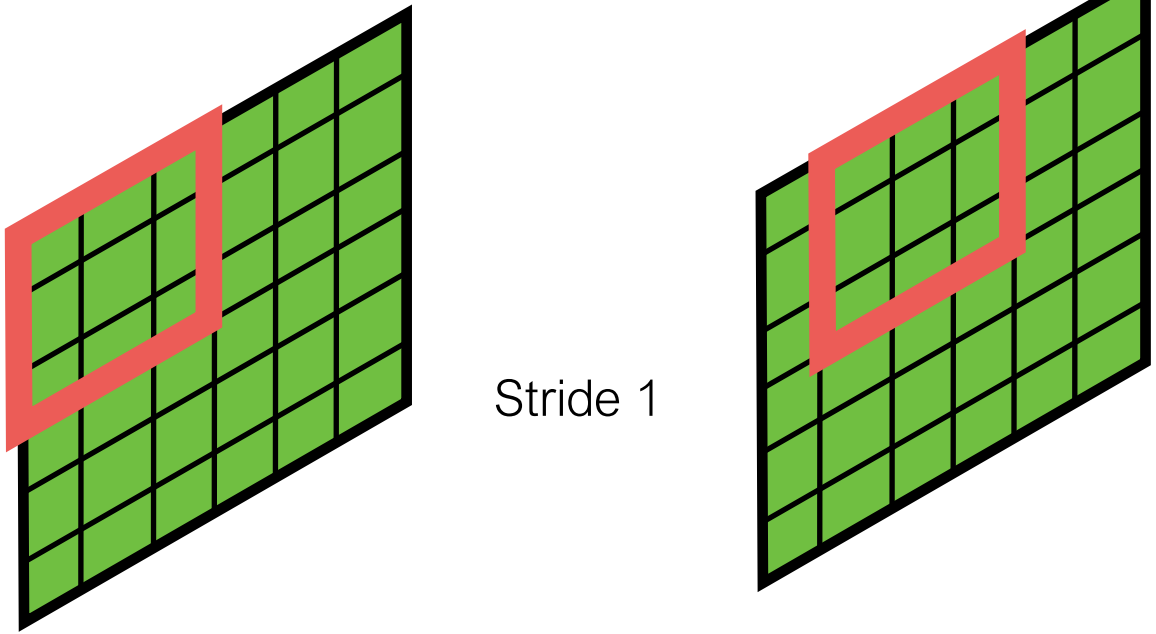


Inspired by Marc'Aurelio Ranzato's slides

Terminologies

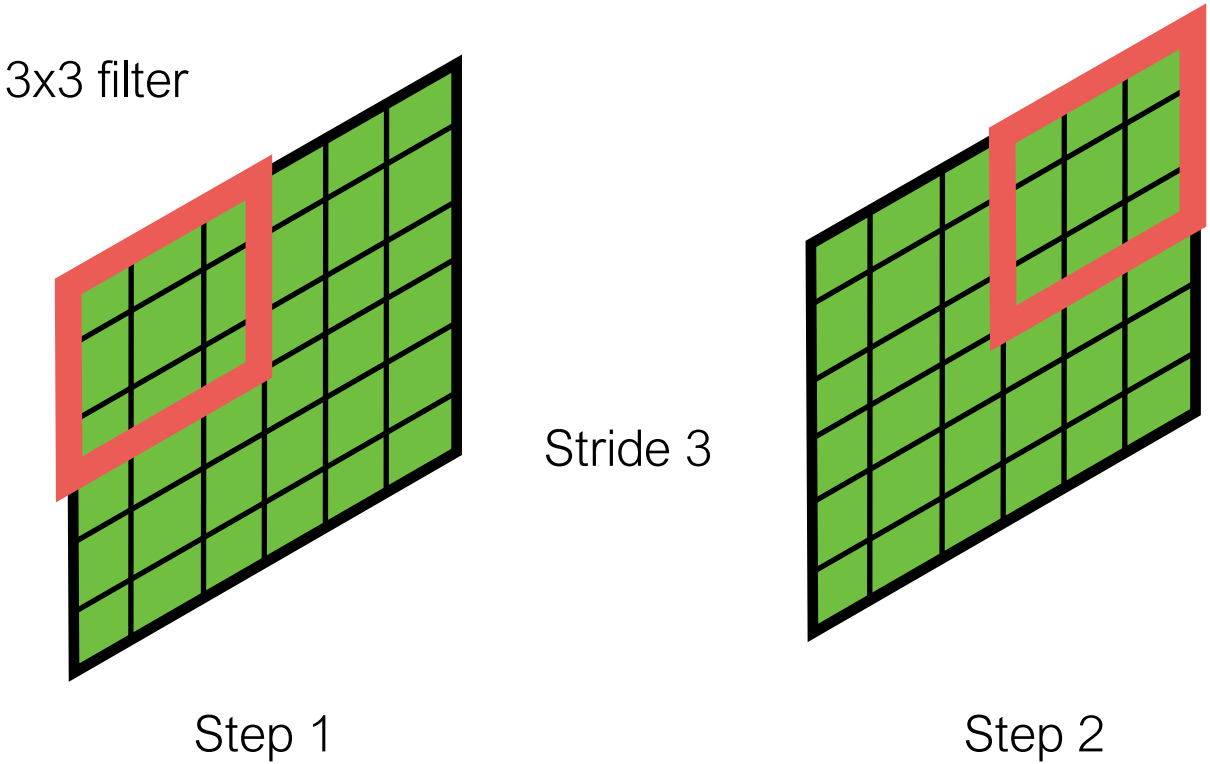


Terminologies



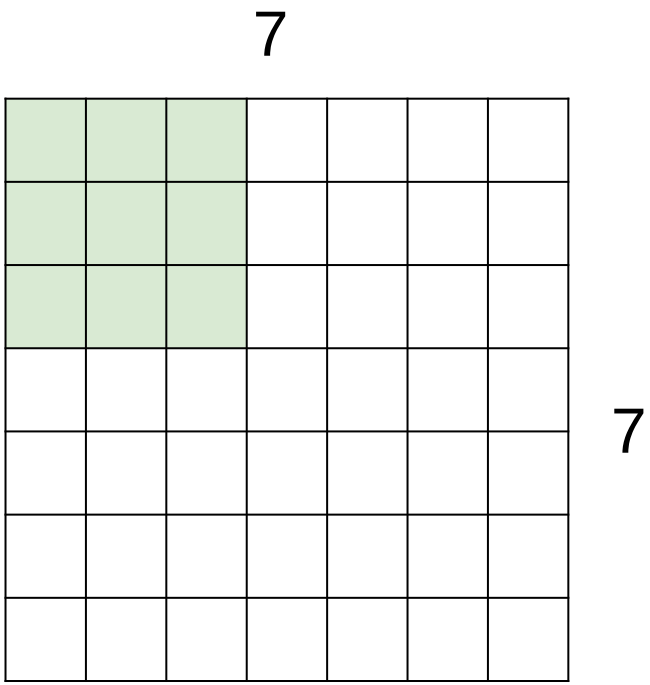
Inspired by Marc'Aurelio Ranzato's slides

Terminologies



Inspired by Marc'Aurelio Ranzato's slides

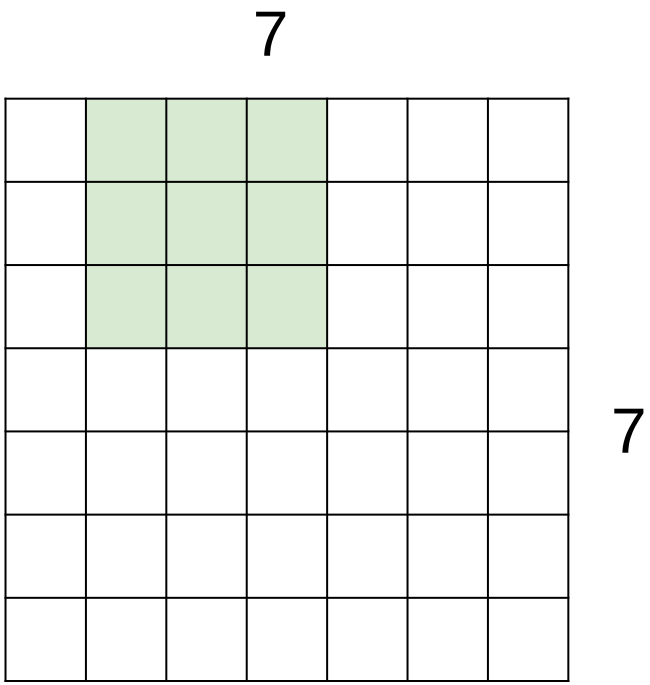
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Slide credit: CS 231N

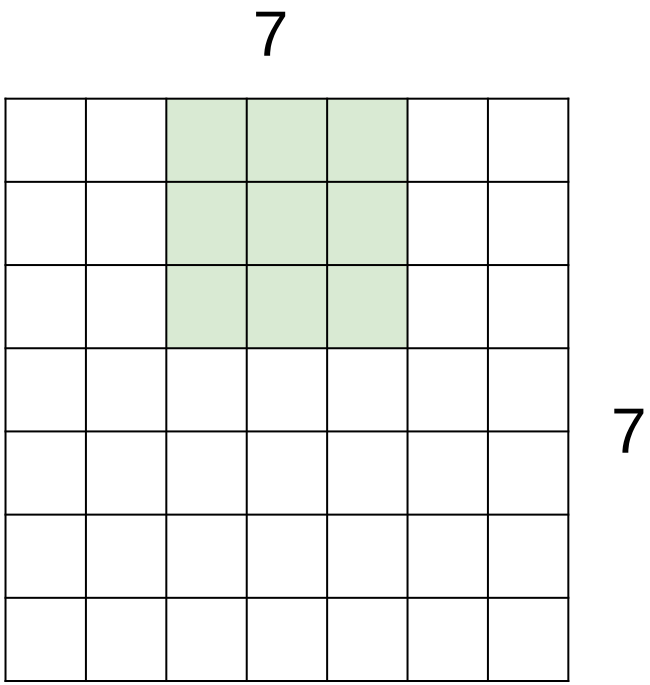
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

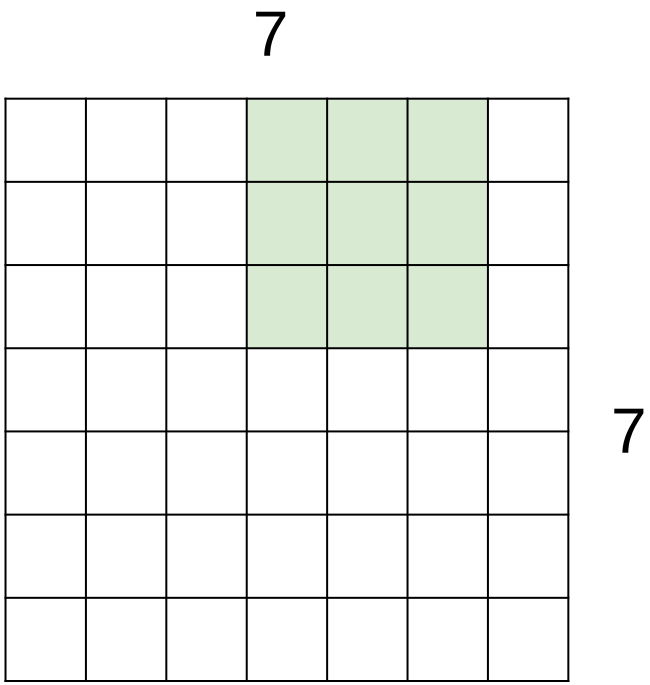
Slide credit: CS 231N

A closer look at spatial dimensions:



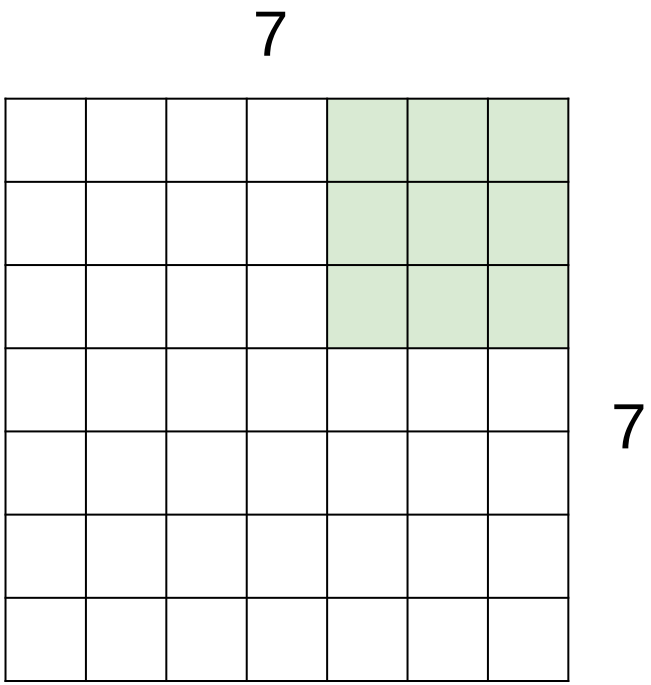
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

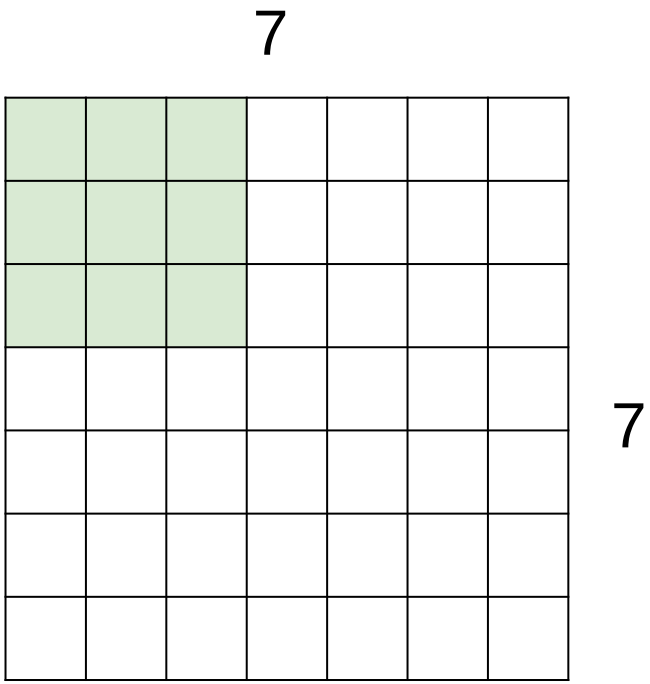


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Slide credit: CS 231N

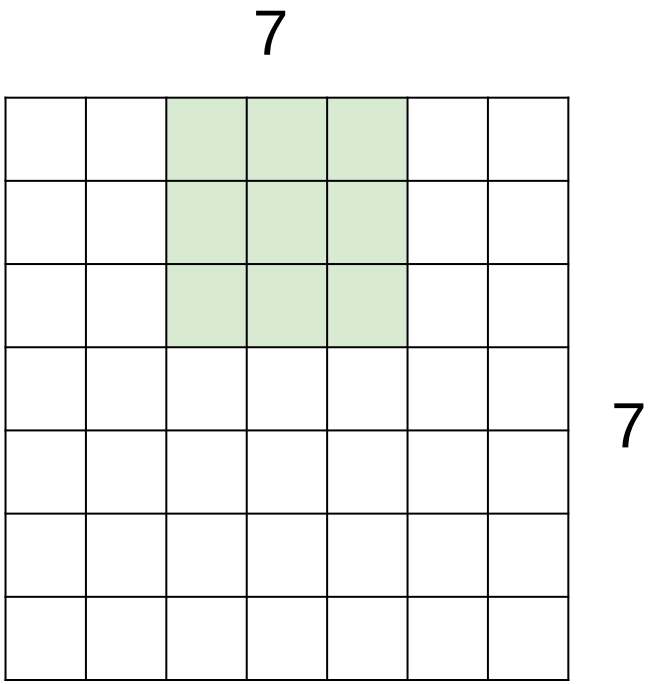
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

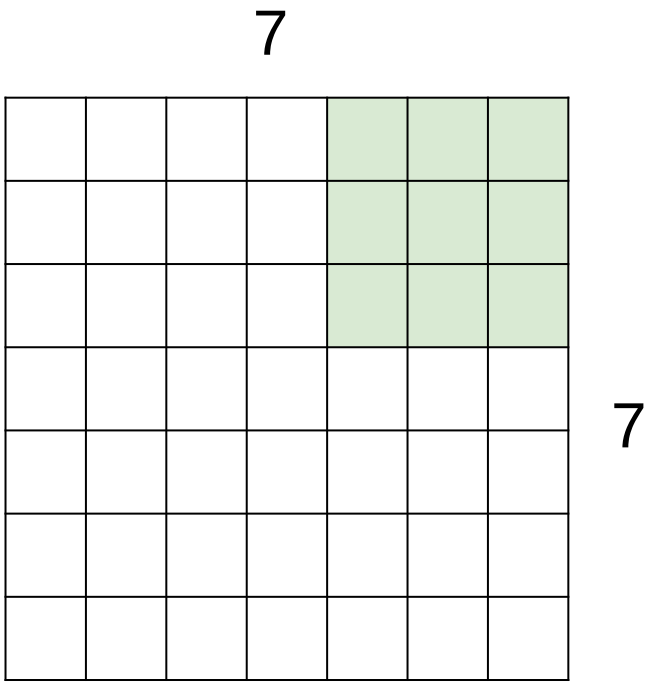
Slide credit: CS 231N

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

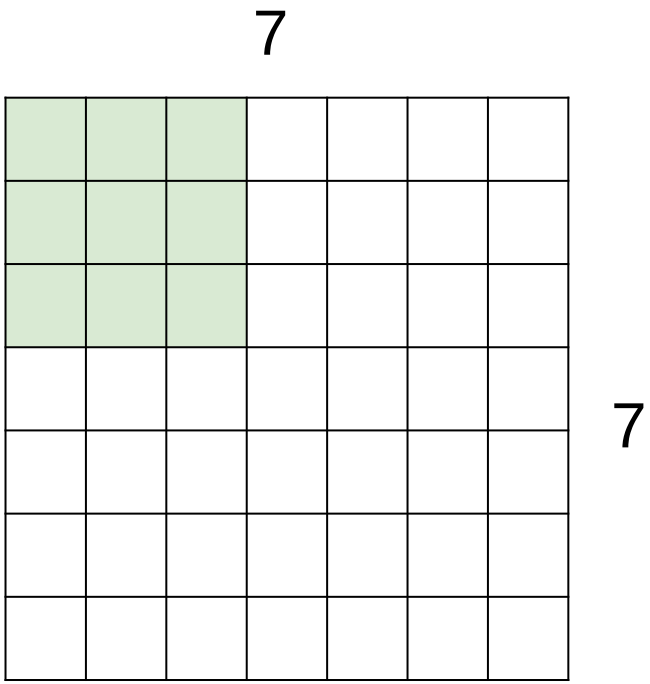
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Slide credit: CS 231N

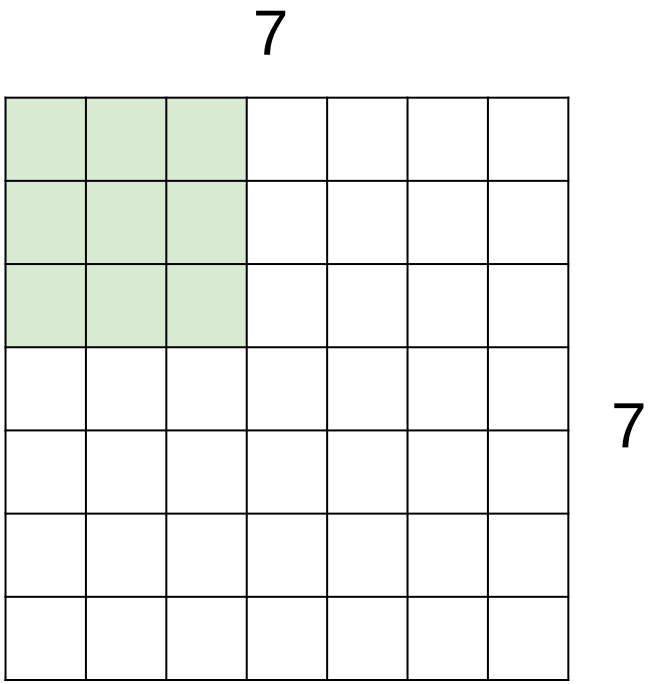
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Slide credit: CS 231N

A closer look at spatial dimensions:

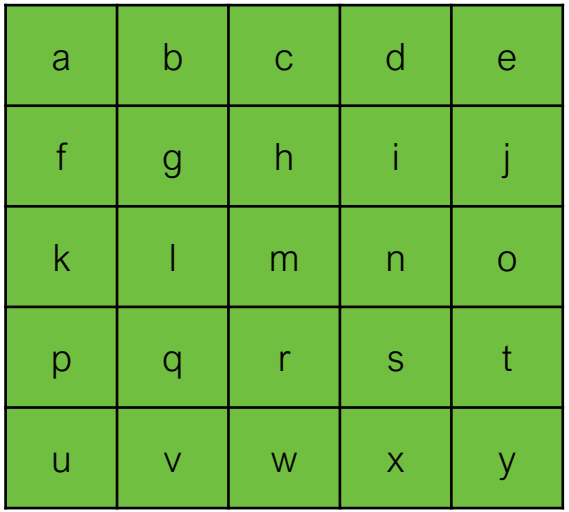


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Slide credit: CS 231N

Terminologies



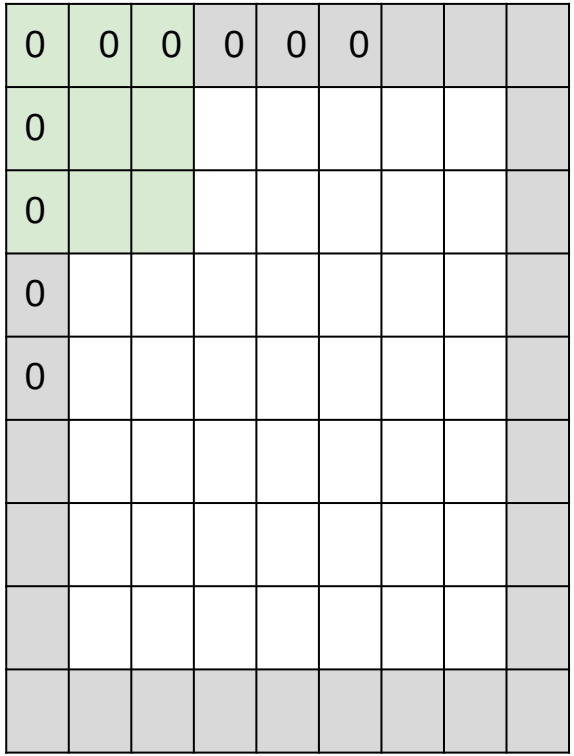
Terminologies

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	a	b	c	d	e	0	0
0	0	f	g	h	i	j	0	0
0	0	k	l	m	n	o	0	0
0	0	p	q	r	s	t	0	0
0	0	u	v	w	x	y	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Zero-padding: 2

USC Viterbi School of Engineering

In practice: Common to zero pad the border

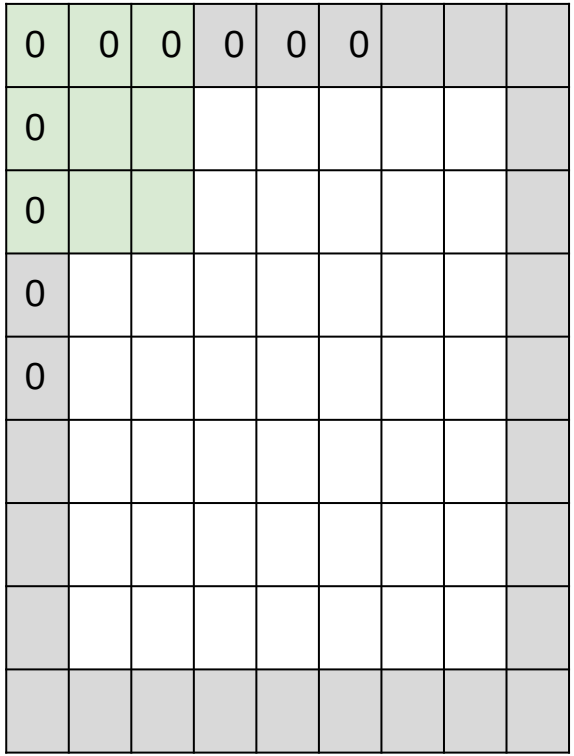


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Slide credit: CS 231N

In practice: Common to zero pad the border



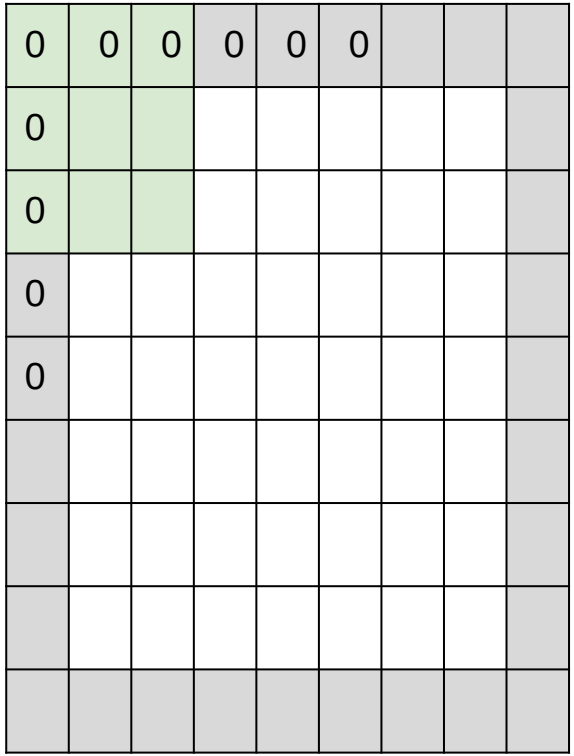
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N + 2P - F) / stride + 1

Slide credit: CS 231N

In practice: Common to zero pad the border

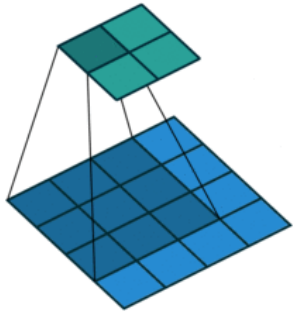


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

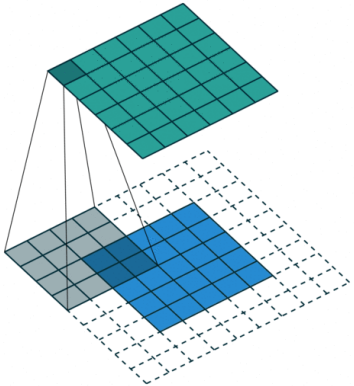
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Slide credit: CS 231N

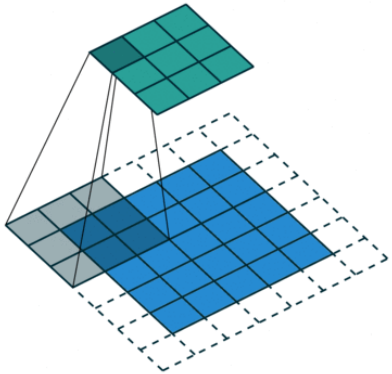
Convolutional layer



padding 0, stride 1



padding 2, stride 1



padding 1, stride 2

Note: it can have **depth** (3rd dimension).

Image credit: vdumoulin (github): https://github.com/vdumoulin/conv_arithmetic

Convolution layer: summary

Let's assume input is W_1 x H_1 x C Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of W_2 x H_2 x K where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Slide credit: CS 231N

Convolution layer: summary

Let's assume input is W_1 x H_1 x C Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

This will produce an output of W_2 x H_2 x K where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Slide credit: CS 231N

Fully connected vs Convolutional

- Fully connected layer
- Locally connected layer
- Convolutional layer

- Which layer is the best? Why?

Fully connected vs Convolutional

Fully connected layer

Every neuron in a fully connected layer is connected to every neuron in the previous layer. This type of layer is often used toward **the end of neural network** architectures to make decisions based on the features extracted by previous layers.

It is **parameter-heavy** and does **NOT** account for the **spatial hierarchy** in the input data, making it less suitable for tasks like image recognition, where spatial relationships are key.

Fully connected vs Convolutional

Locally connected layer

Each neuron is connected only to **a subset of the input data**, maintaining spatial relationships similar to convolutional layers.

However, unlike convolutional layers, **the weights are not shared across different spatial locations**. This allows the model to learn different features at different locations, but it can lead to **a large number of parameters**, making the network prone to overfitting and computationally expensive.

Fully connected vs Convolutional

Convolutional layer

These layers apply a convolution operation to the input, effectively sliding a number of filters across the input data to produce feature maps.

This method **respects the spatial hierarchy of the data**, is **parameter-efficient** due to weight sharing, and is translation invariant. Convolutional layers are typically used in tasks involving images or time series data where such properties are advantageous.

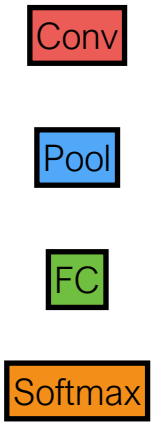
Fully connected vs Convolutional

Which layer is “best” cannot be universally answered:

- For image-related tasks, convolutional layers are often the best because they can efficiently learn and represent spatial hierarchies in the data.
- For problems where the precise location of features within the input data is important, and the input data is consistently structured, locally connected layers might be preferable.
- For tasks that require holistic understanding of the input data, where spatial relationships are not as important, fully connected layers can be effective, especially as a part of the network's output layer.

What are these layers?

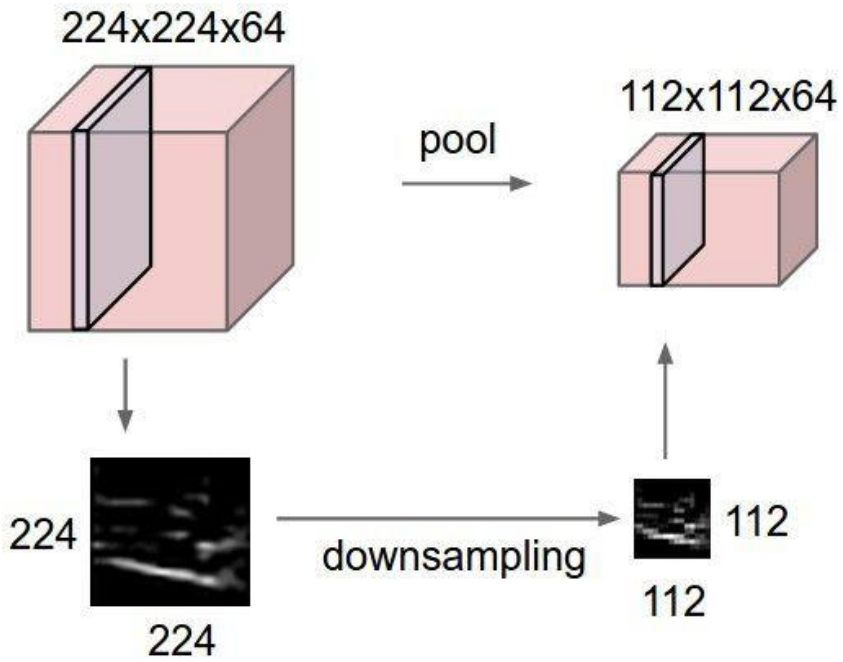
- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



Layer = Function

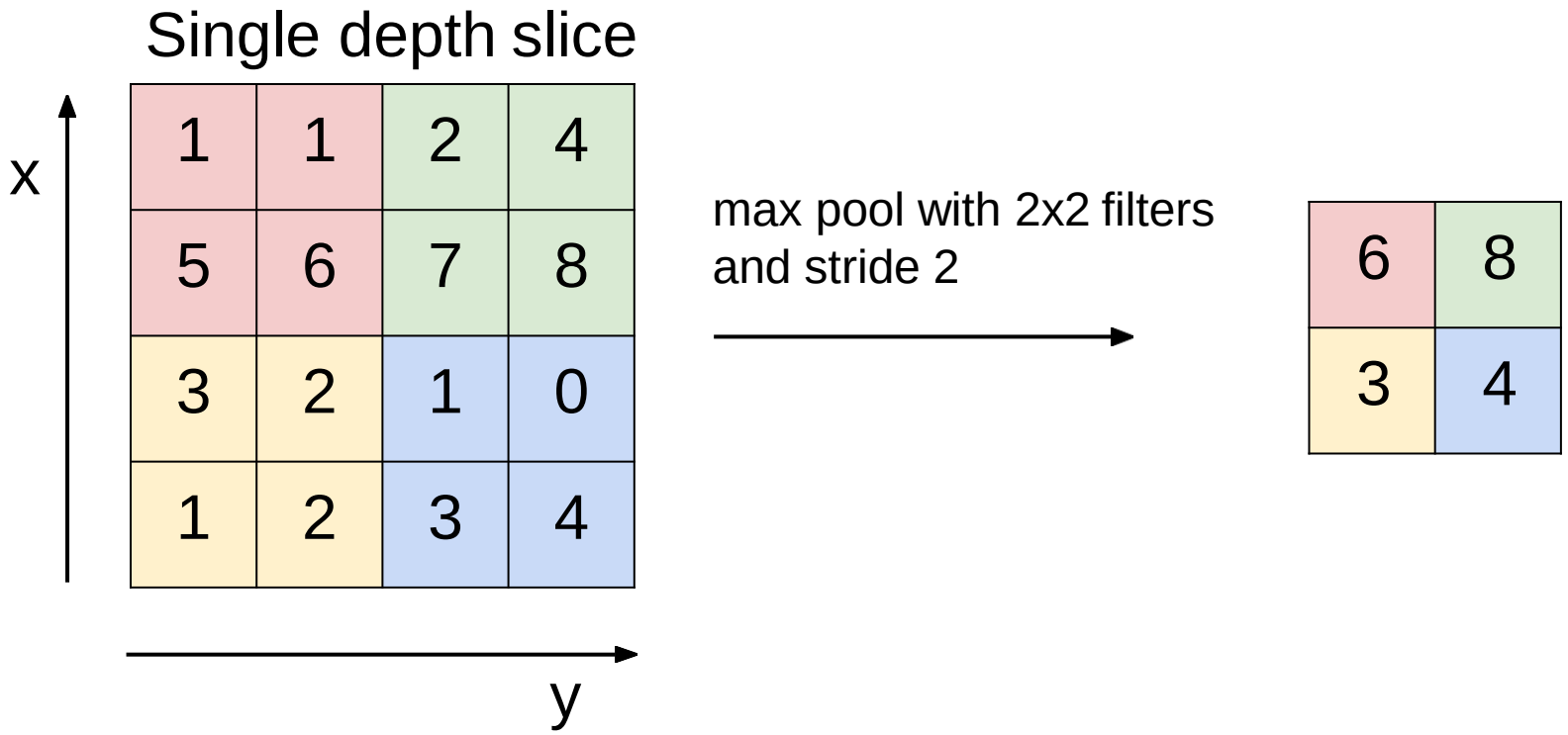
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently



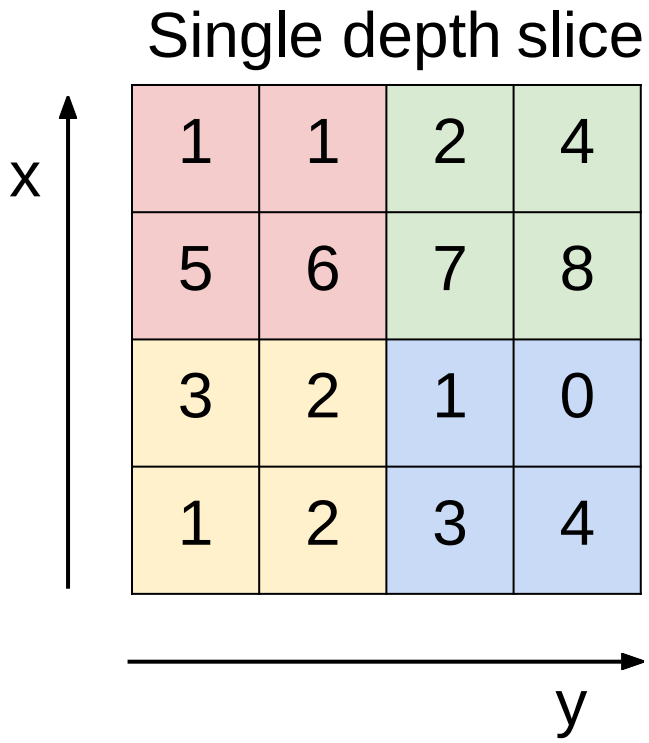
Slide credit: CS 231N

Max Pooling



Slide credit: CS 231N

Max Pooling



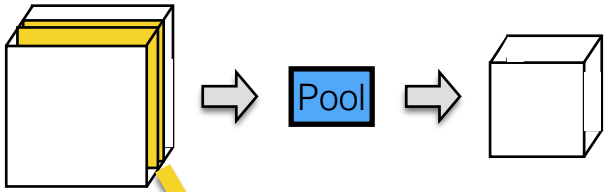
max pool with 2x2 filters
and stride 2

6	8
3	4

- No learnable parameters
- Introduces spatial **invariance**

Slide credit: CS 231N

Max Pooling

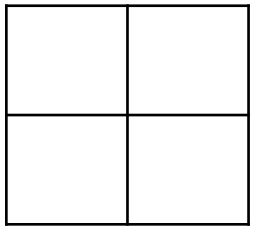


M x N x P

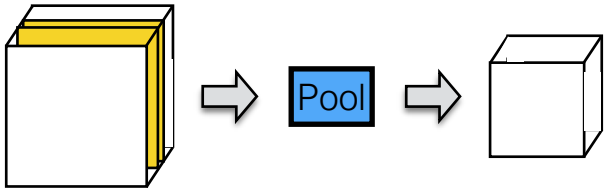
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2



Max pooling layer



M x N x P

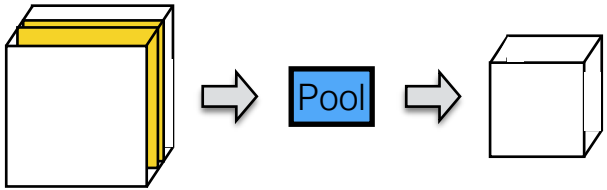
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	

Max pooling layer



M x N x P

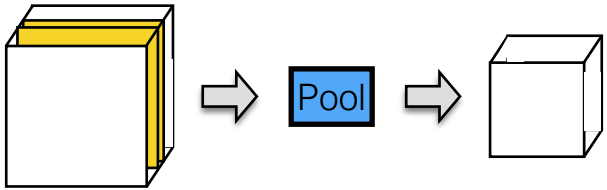
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	7

Max pooling layer



M x N x P

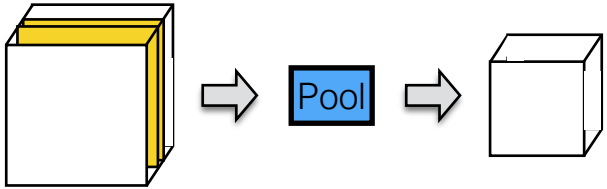
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	7
8	

Max pooling layer



M x N x P

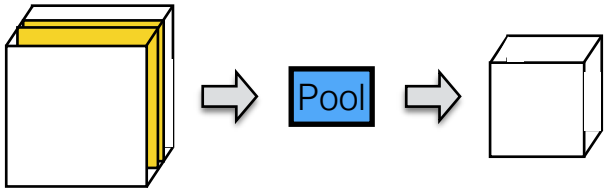
m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	7
8	9

Max pooling layer



M x N x P

m x n x P

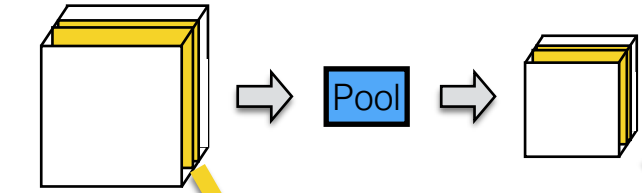
1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

→

8	7
8	9

Max pooling layer



M x N x P

m x n x P

1	3	2	7	2
5	1	3	0	5
4	8	5	0	6
3	2	6	0	8
0	7	4	6	9

max pool with
3x3 filters & stride 2

8	7
8	9

Pooling layers

- **Max pooling** (most common)
- Average pooling
- L2-norm pooling
- …

Why pooling layers?

- Reduces # of parameters in the following layers

Why pooling layers?

- Reduces # of parameters in the following layers
  - Hence, less overfitting!—reducing model complexity

Pooling layer: summary

Let's assume input is W_1 x H_1 x C Pooling layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

This will produce an output of W_2 x H_2 x C where:

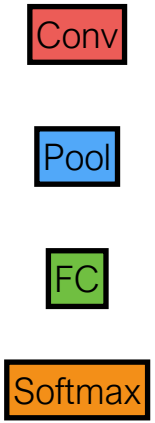
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0 -> nothing to be learned

Slide credit: CS 231N

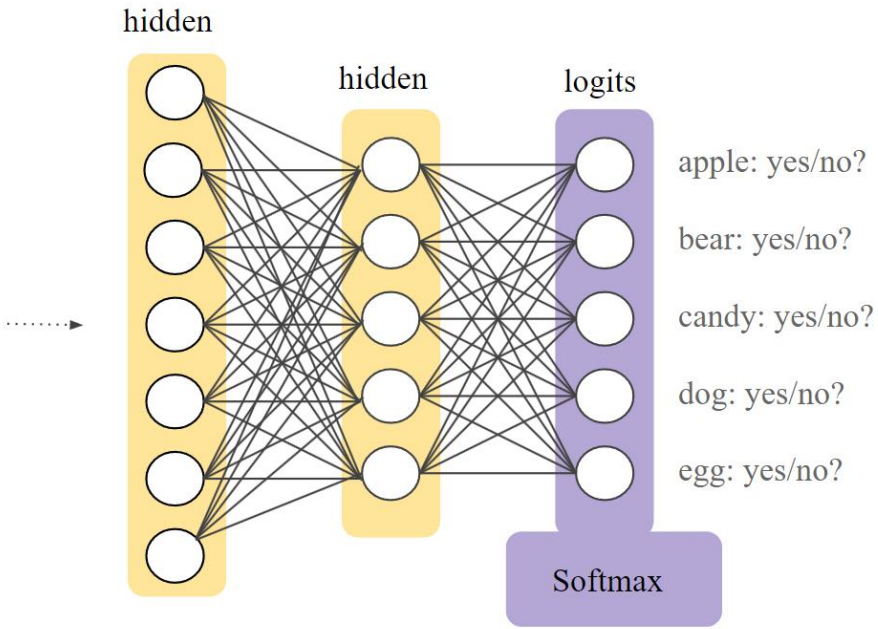
What are these layers?

- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



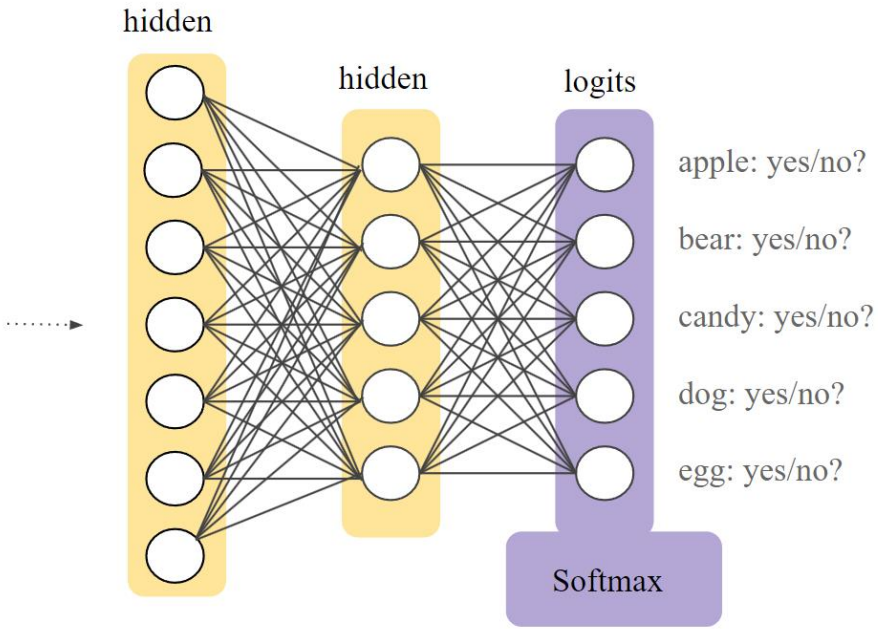
Layer = Function

Softmax Layer



Softmax is implemented through a neural network layer just ***before*** the output layer. The Softmax layer must have the same number of nodes as the output layer.

Softmax Layer



The goal is to map the non-normalized output of a network to a **probability distribution** over predicted output classes.

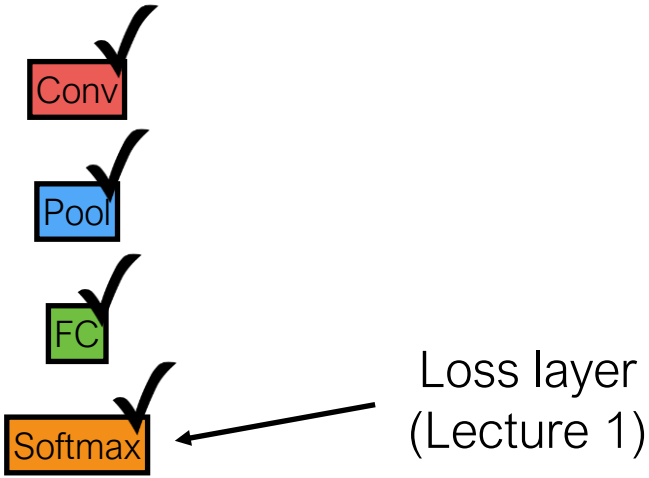
a given class *I* in a classification problem with K classes is:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

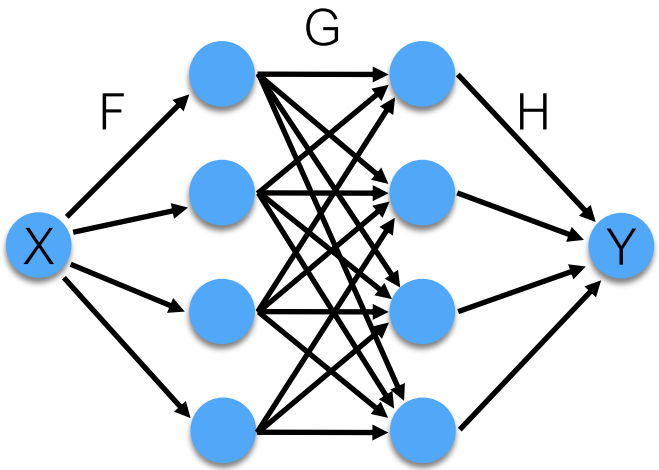
- *z* is the input vector to the softmax function, containing the raw class scores from the network
- $\sigma(z)_i$ is the output probability of the *i*-th class.

All these are linear!!!

- Convolutional layer
- Pool layer
- Fully connected layer
- Softmax layer



Activation functions

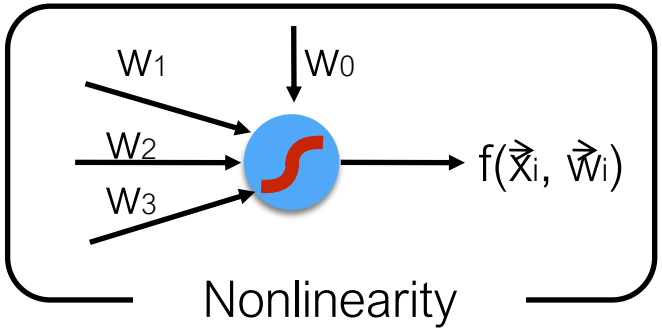


Y=H(G(F(X)))

Y=(HGF)(X)

Without nonlinear activation,
multiple layers are reduced to one matrix multiplication.

Activation functions

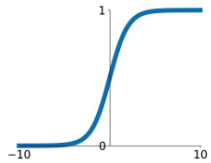


Nonlinearity makes a neural network deeper and more expressive

Recap on Activation functions

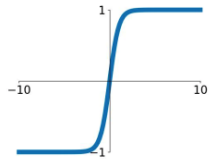
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



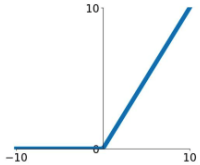
tanh

$$\tanh(x)$$



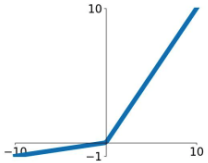
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

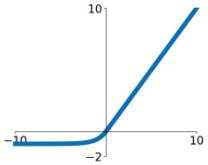
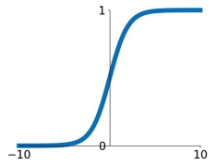


Figure from Stanford cs231n lecture slides

Recap on Activation functions

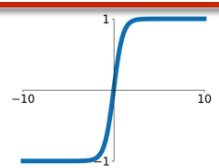
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



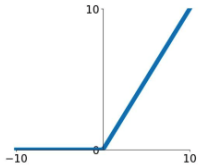
tanh

$$\tanh(x)$$



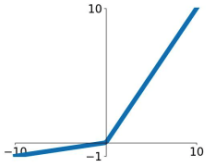
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

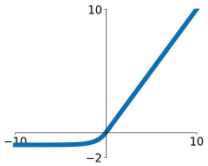
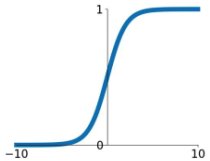


Figure from Stanford cs231n lecture slides

Recap on Activation functions

Sigmoid

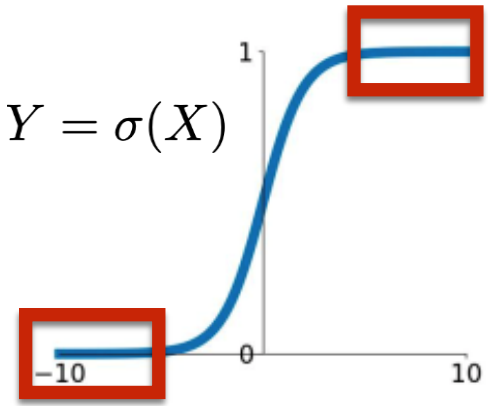
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- Similar to step functions, but differentiable
- (-) easily saturated (no gradients)

Figure from Stanford cs231n lecture slides

Recap on Gradient vanishing problem



Chain rule

$$\frac{\partial Y}{\partial X} = \frac{\partial Y}{\partial \sigma} \frac{\partial \sigma}{\partial X}$$

is almost 0 for most of X

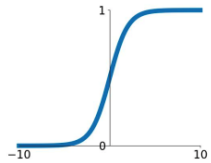
Thus, $\frac{\partial Y}{\partial X}$ also goes to 0.

Figure from Stanford cs231n lecture slides

Recap on Activation functions

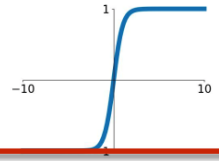
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



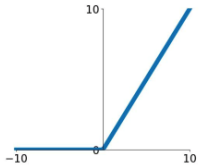
tanh

$$\tanh(x)$$



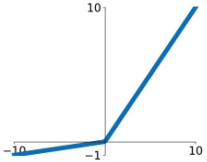
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

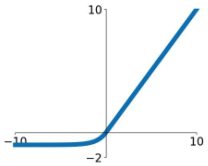
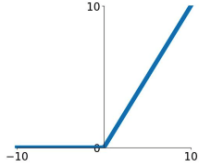


Figure from Stanford cs231n lecture slides

Recap on Activation functions

ReLU

$$\max(0, x)$$



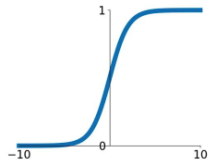
- Prevent gradient vanishing when x > 0
- Computationally efficient
- Biologically plausible
- (-) Still loose gradient when x < 0

Figure from Stanford cs231n lecture slides

Recap on Activation functions

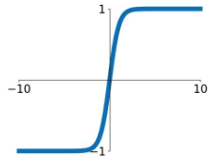
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



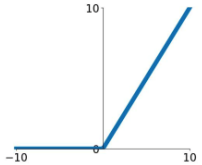
tanh

$$\tanh(x)$$



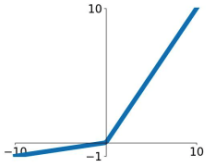
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

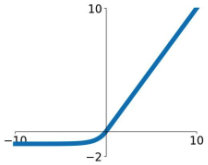
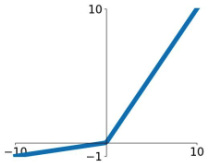


Figure from Stanford cs231n lecture slides

Leaky ReLU/ELU

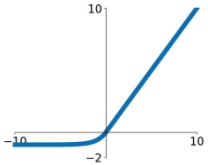
Leaky ReLU

$$\max(\alpha x, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

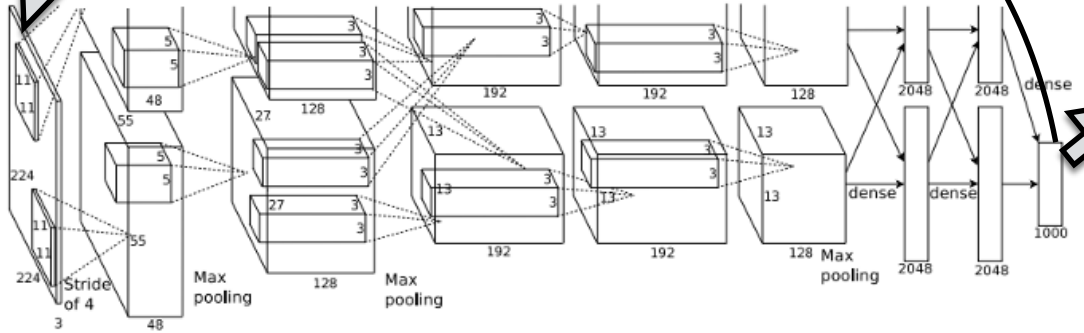


- No gradient vanishing problem
- Can be used instead of ReLU (e.g. Leaky ReLU for deconvolution)

Figure from Stanford cs231n lecture slides

AlexNet in details

Flower



0.1	Car
0.01	Dog
0.7	Flower
0.1	Person
…	…

Image → Conv → Pool → ReLU → Conv → Pool → ReLU → Conv → ReLU → Conv → ReLU → Conv → Pool → ReLU → FC → ReLU → FC → ReLU → Softmax

USC Viterbi School of Engineering

AlexNet in details

Flower

0.1 Car

0.1 Person

… …

Typically omit ReLU for the visualization

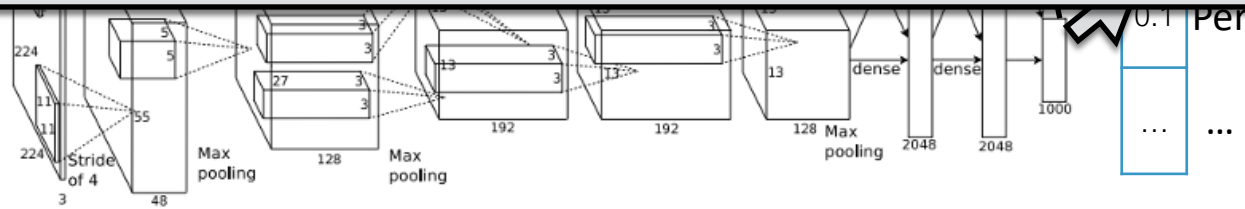


Image | Conv, Pool, ReLU | Conv, Pool, ReLU | Conv, ReLU | Conv, ReLU | Conv, Pool, ReLU | FC, ReLU | FC, ReLU | Softmax

USC Viterbi School of Engineering

AlexNet in details

Flower

0.1 Car

0.1 Person

… …

Typically omit ReLU for the visualization

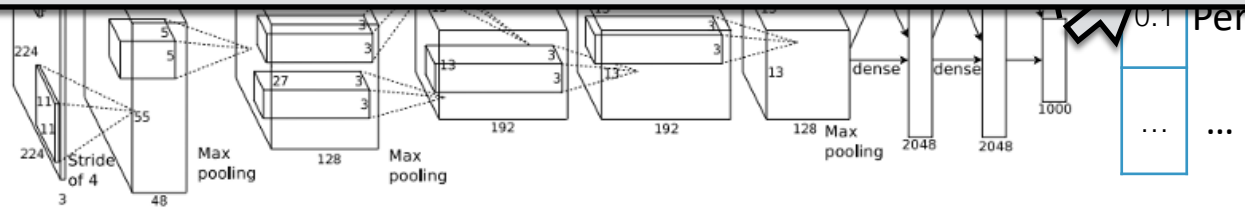
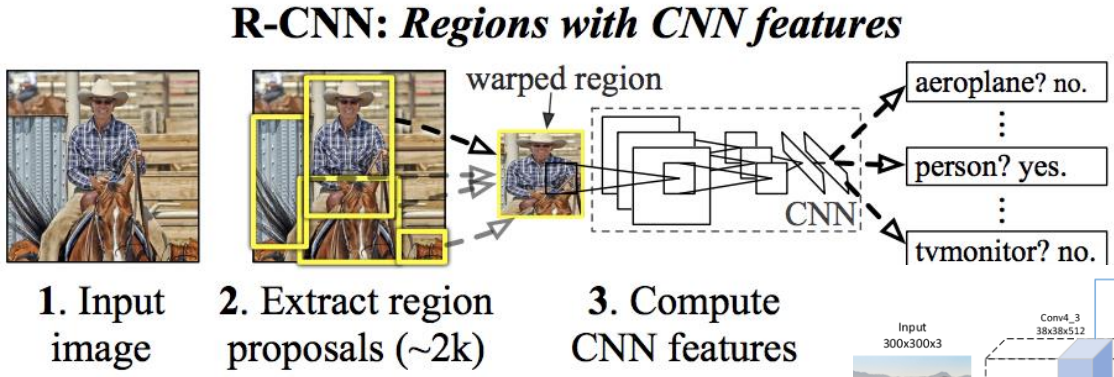


Image | Conv | Pool | Conv | Pool | Conv | Conv | Conv | Pool | FC | FC | Softmax

More CNNs



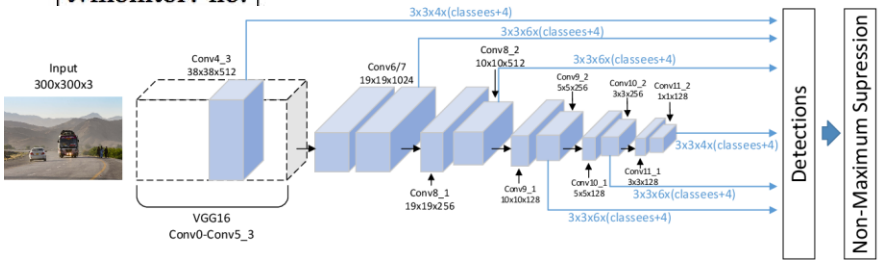
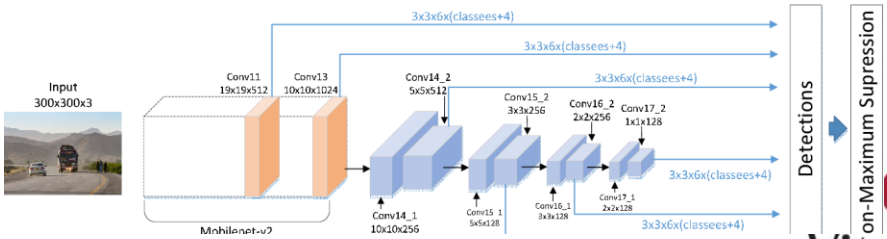


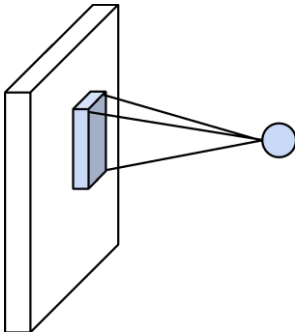
Fig. 1. Network architecture of VGG16-SSD.

RCNN, MobileNet, and many more

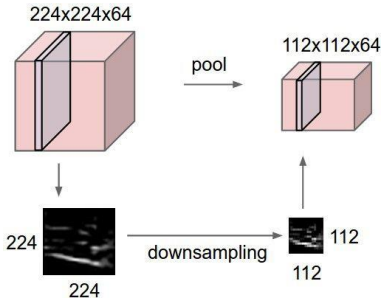


Components of CNNs

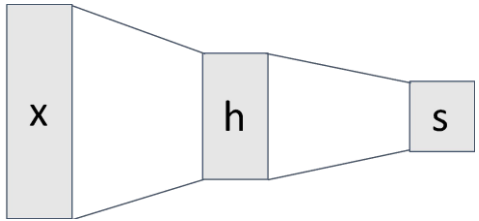
Convolution Layers



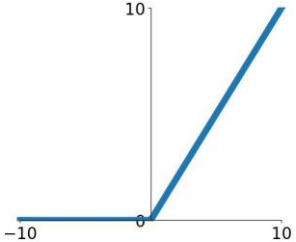
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Figure from Stanford cs231n lecture slides

Batch Normalization

Consider a single layer $y = Wx$

The following could lead to tough optimization:

- Inputs x are not *centered around zero* (need large bias)
- Inputs x have different scaling per-element (entries in W will need to vary a lot)

Idea: force inputs to be “nicely scaled” at each layer!

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want zero-mean unit-variance activations? just make them so."

consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

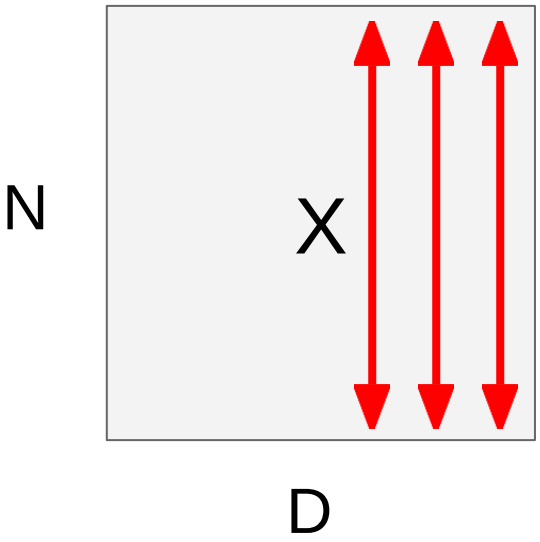
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

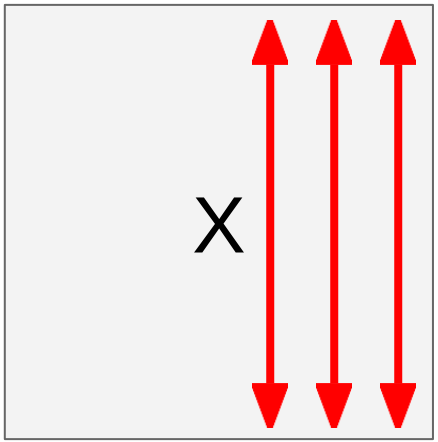
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

During testing batchnorm becomes a linear operator!
Can be fused with the previous fully-connected or conv layer

$$\mu_j = \text{(Running) average of values seen during training}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \text{(Running) average of values seen during training}$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

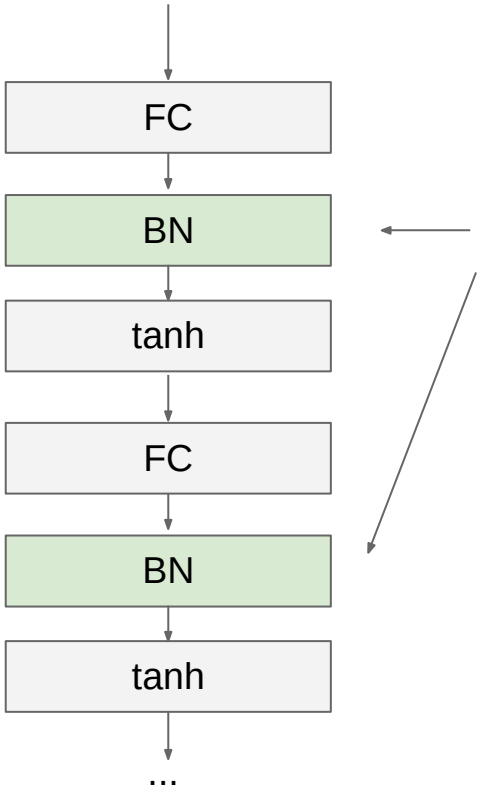
$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Figure from Stanford cs231n lecture slides

Batch Normalization

[Ioffe and Szegedy, 2015]

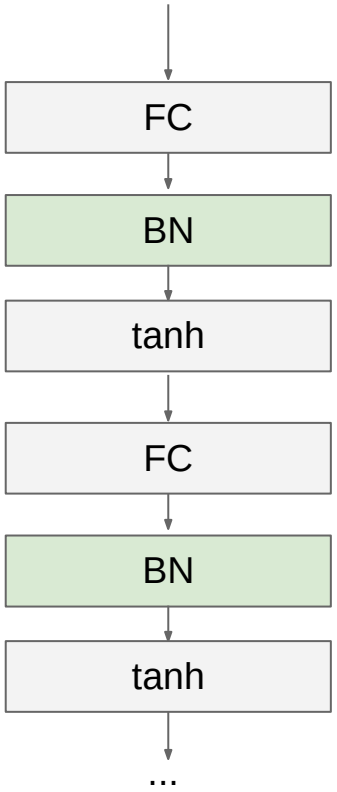


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization

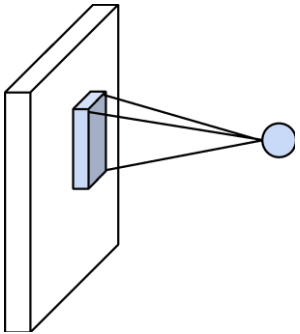
[Ioffe and Szegedy, 2015]



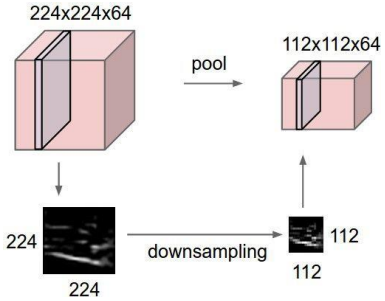
- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Components of CNNs

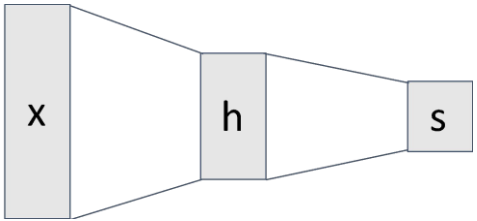
Convolution Layers



Pooling Layers

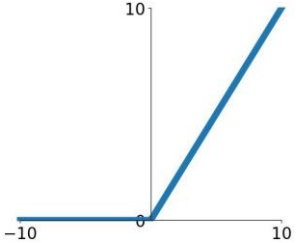


Fully-Connected Layers



Question: How should we put them together?

Activation Function



Normalization

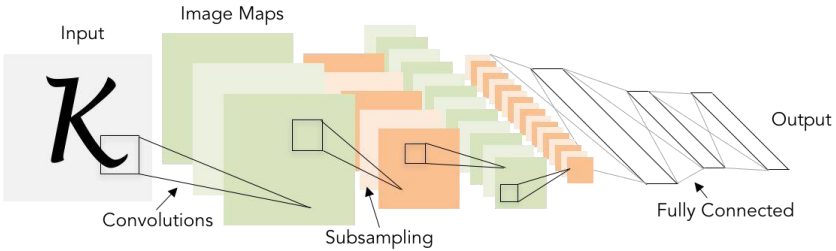
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

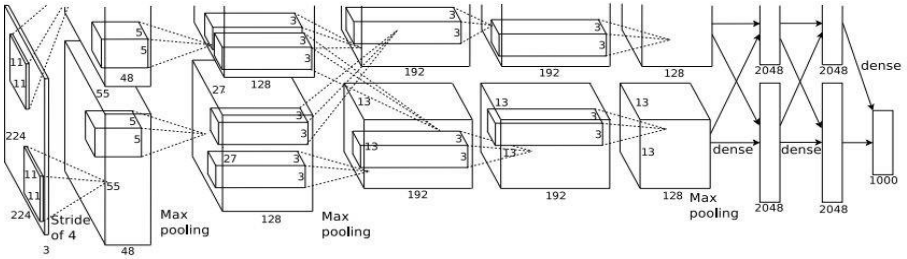
Figure from Stanford cs231n lecture slides

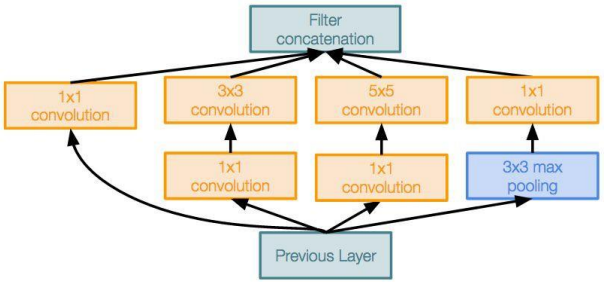
Convolutional Neural Networks (CNN)

CNN Architectures & Applications

Today: CNN Architectures







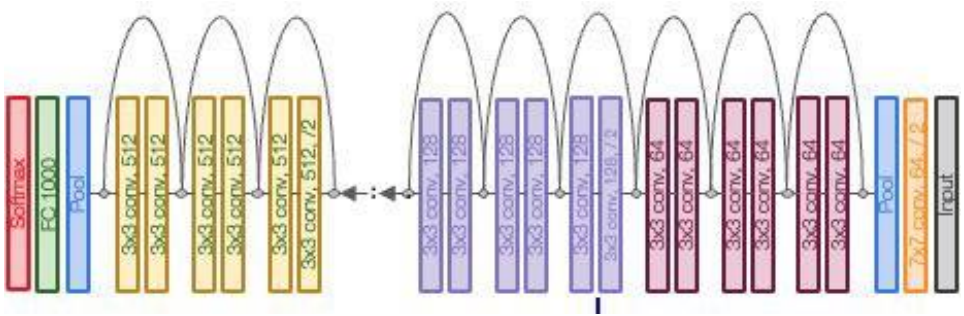


Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

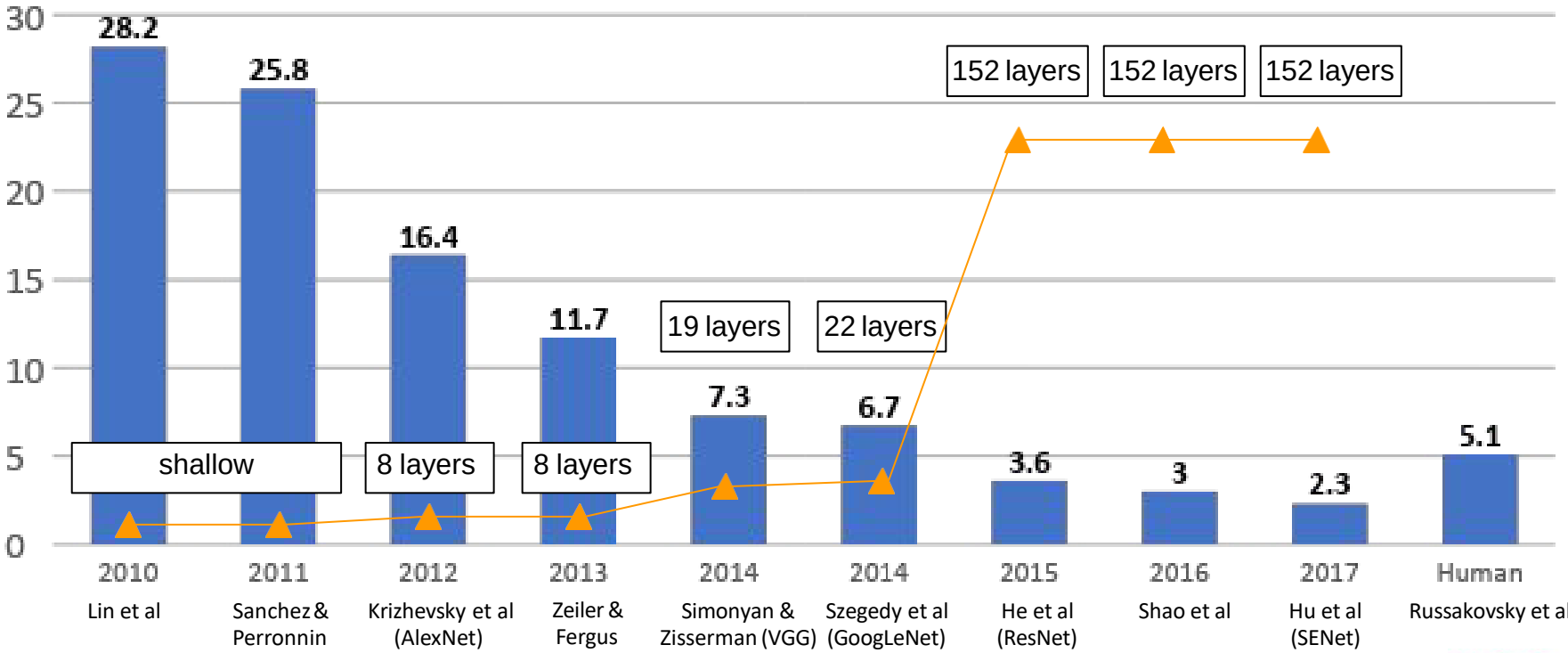


Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

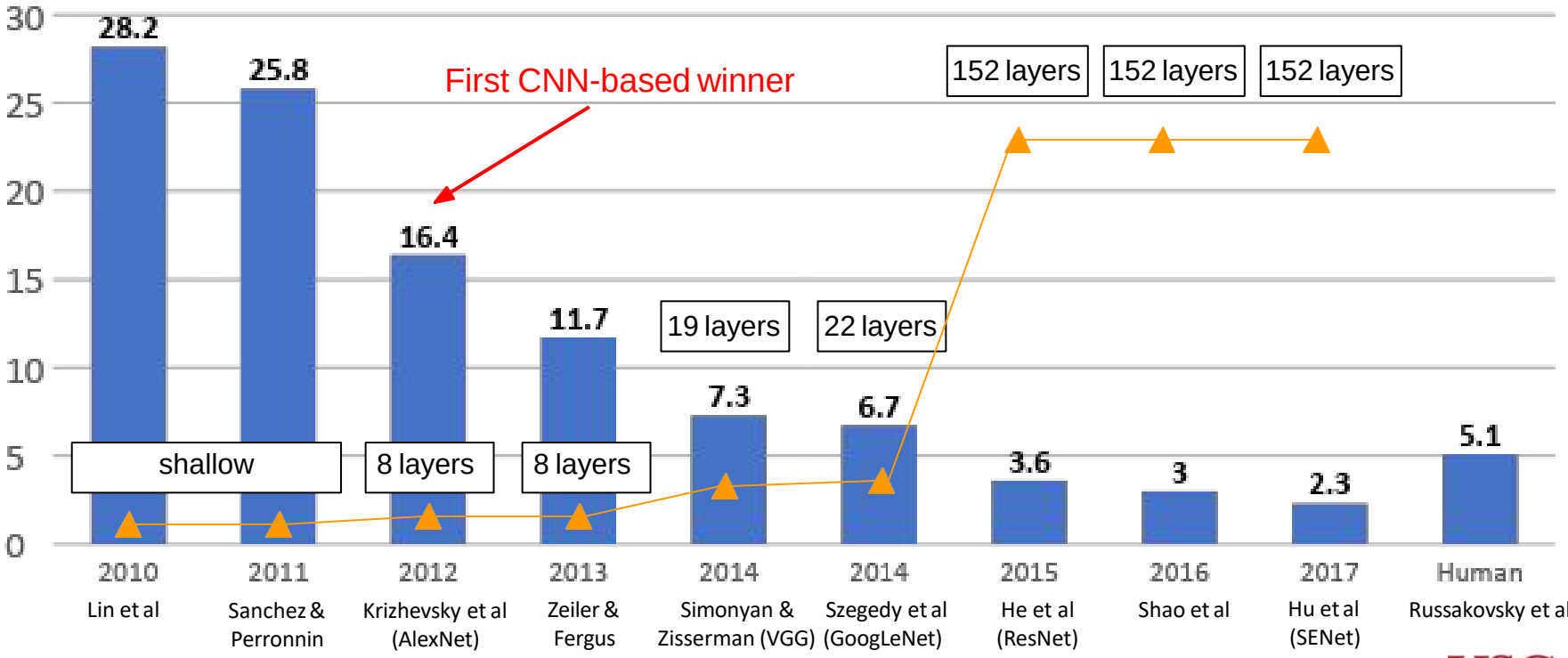


Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

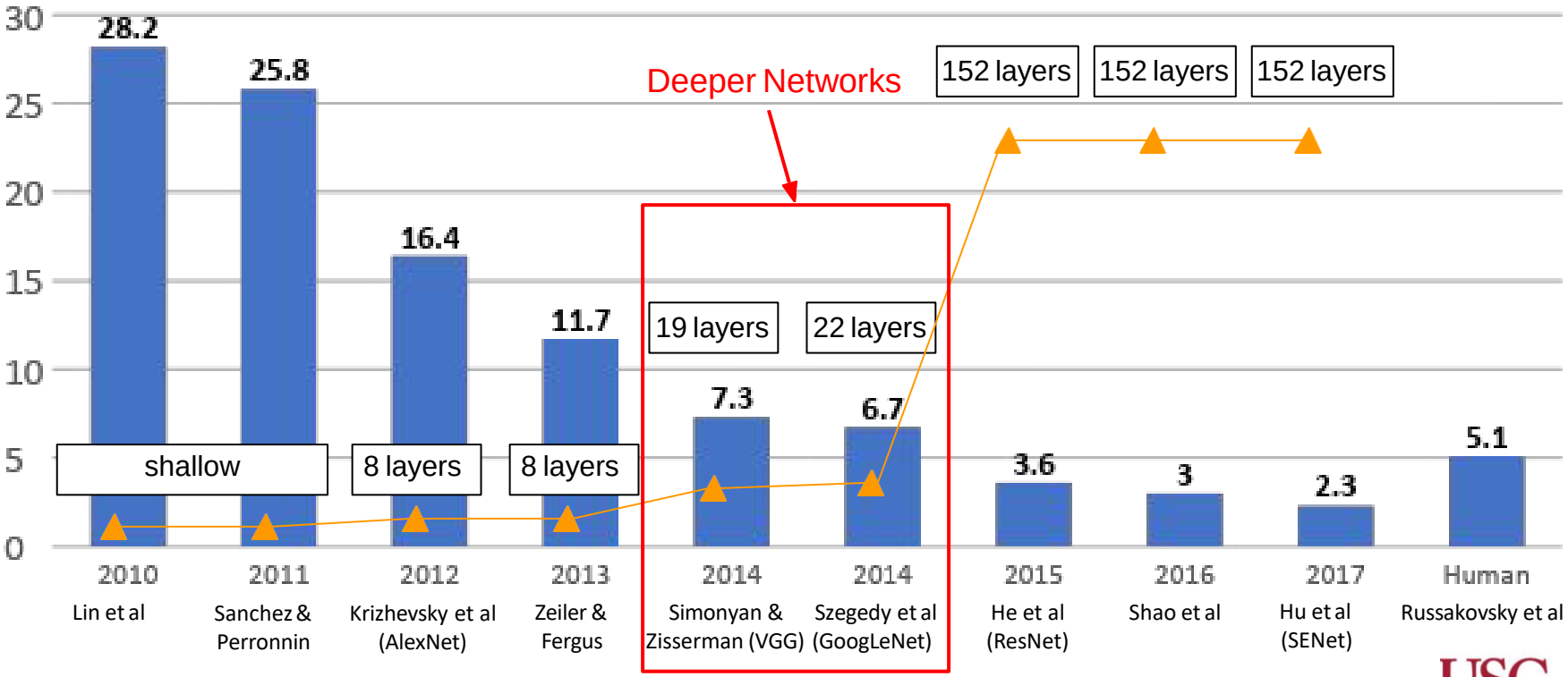


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

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

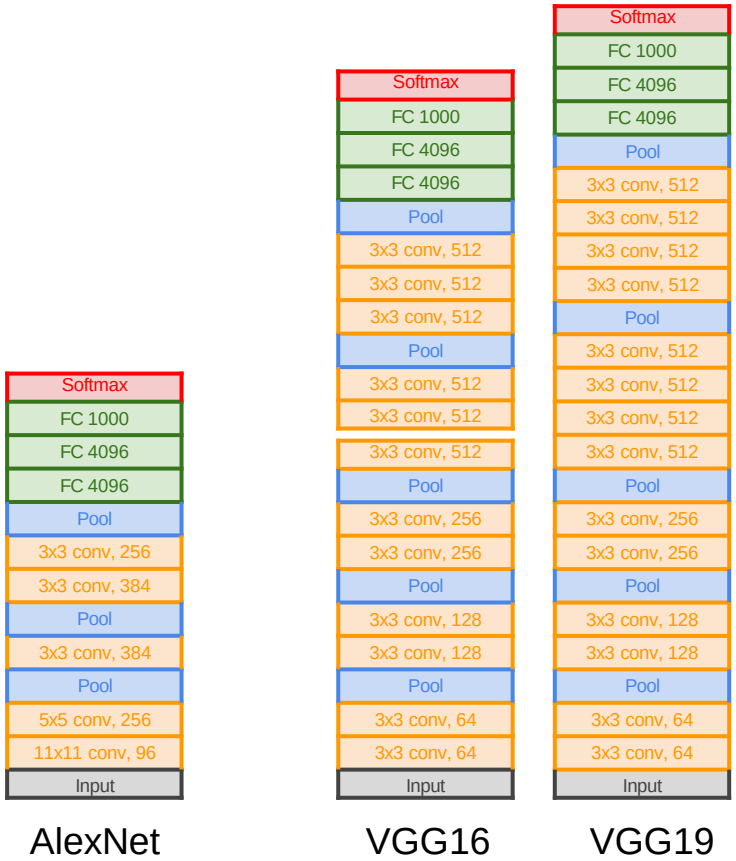


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

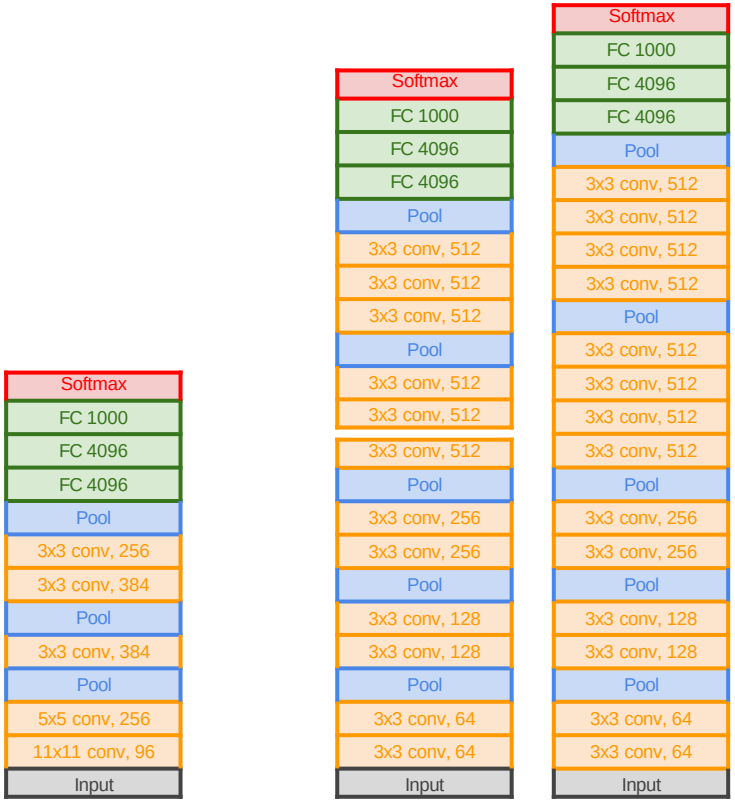


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

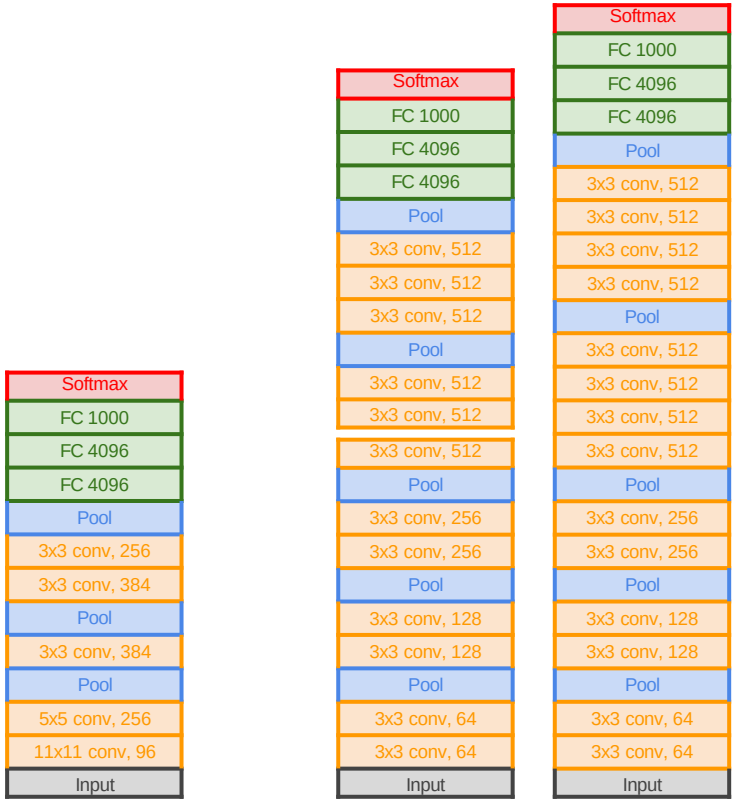
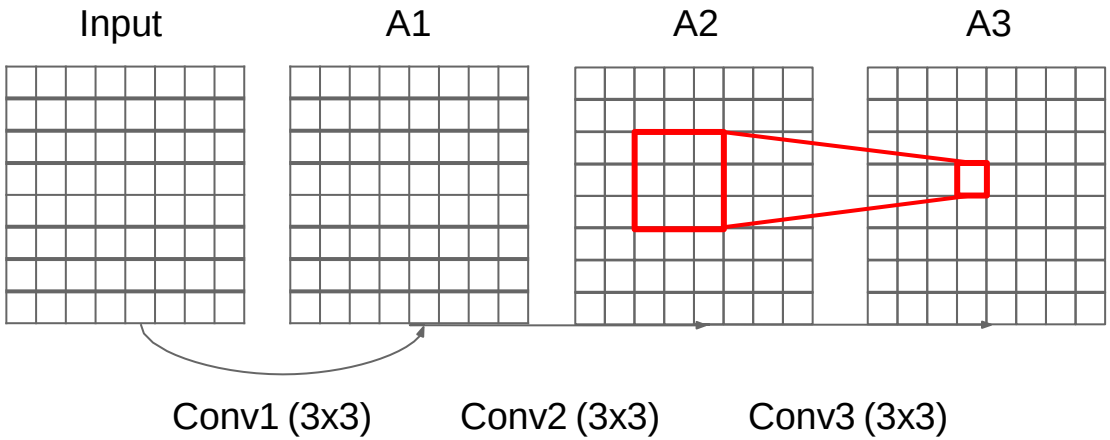


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



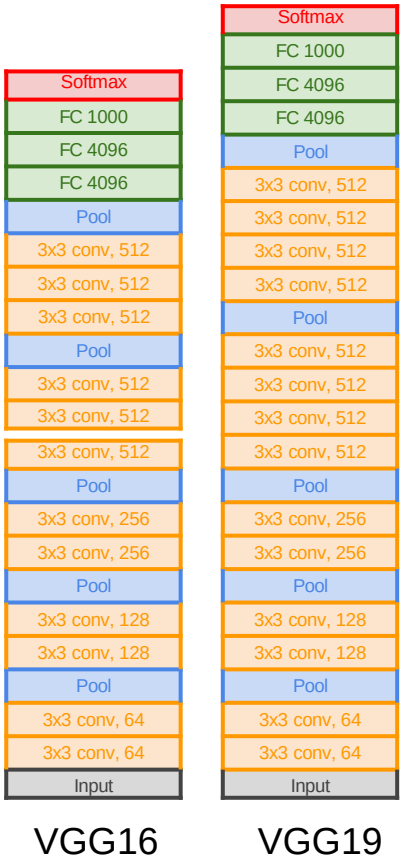
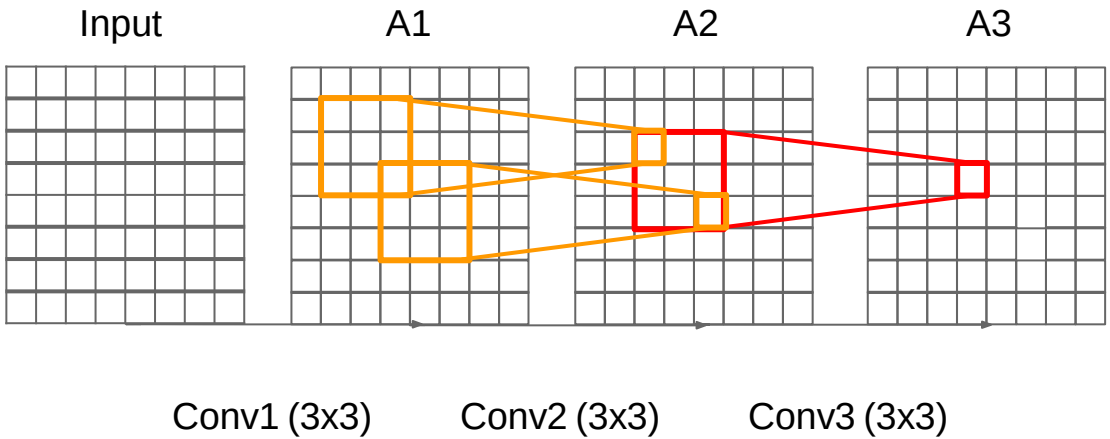


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



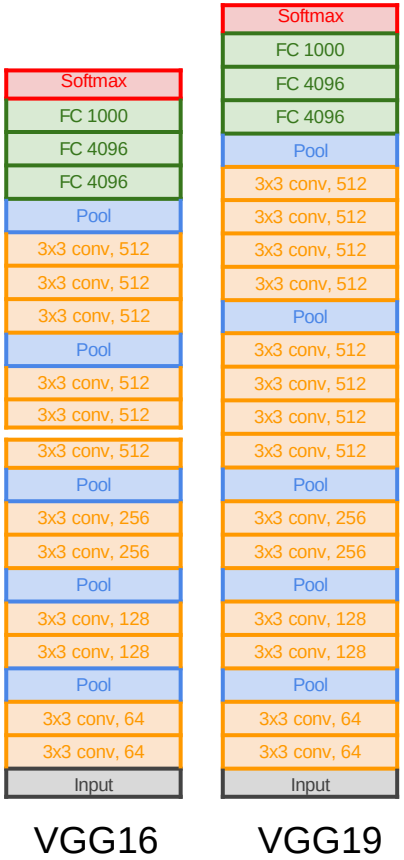
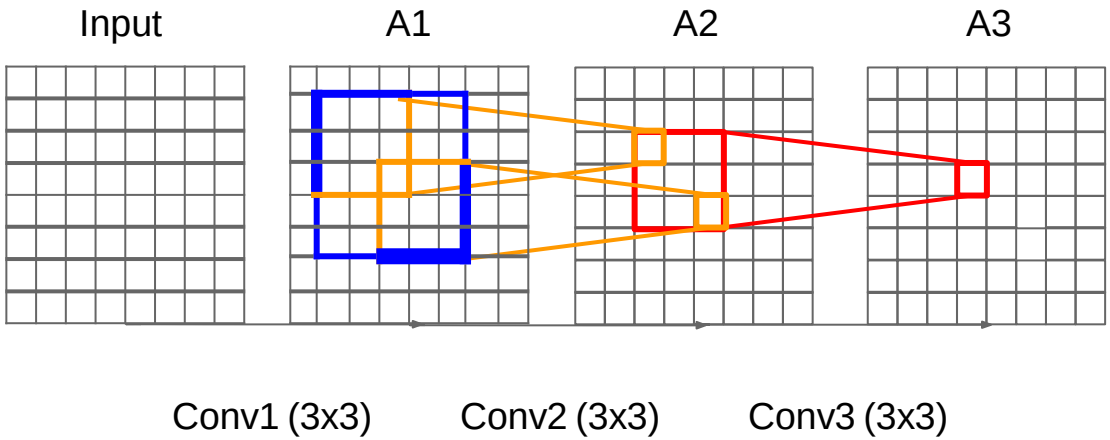


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



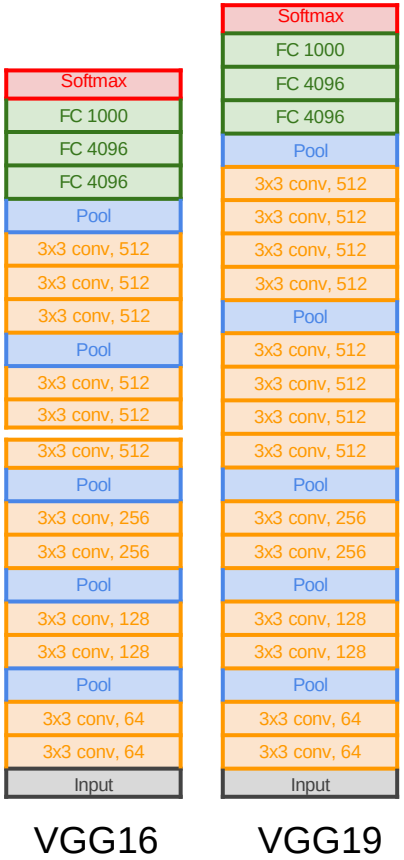
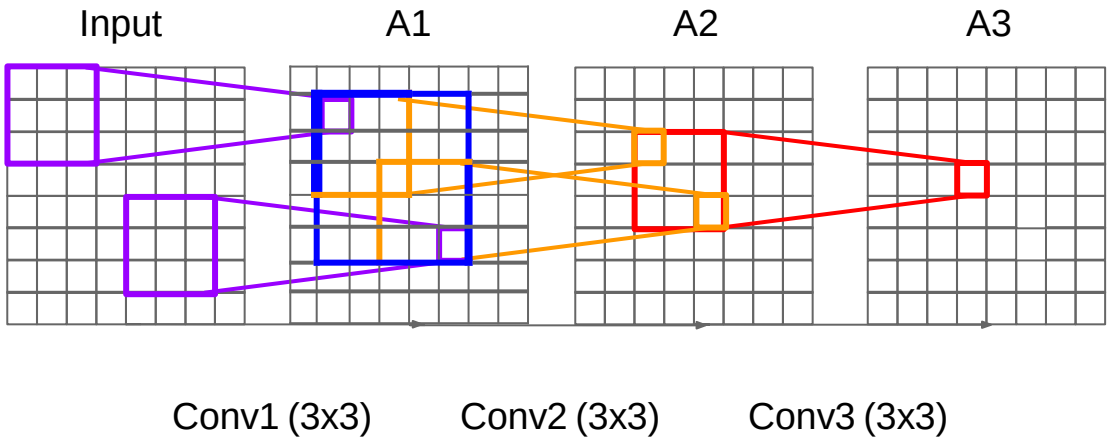


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



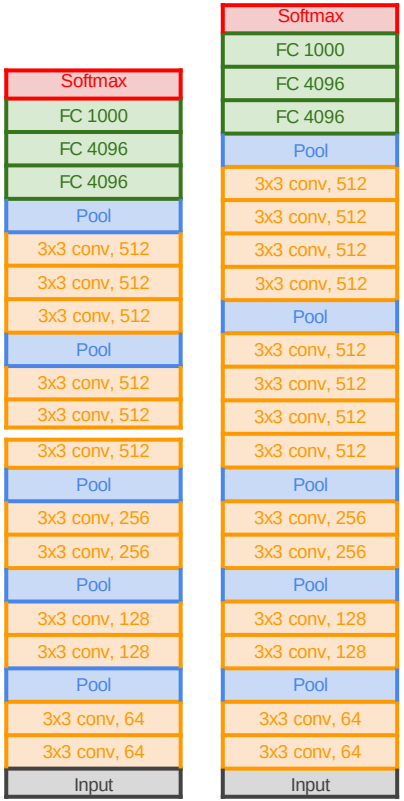
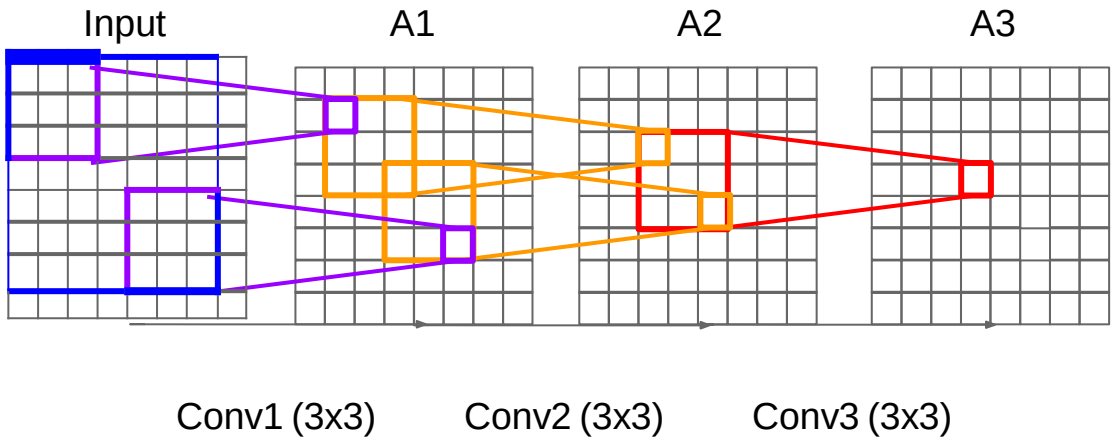


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



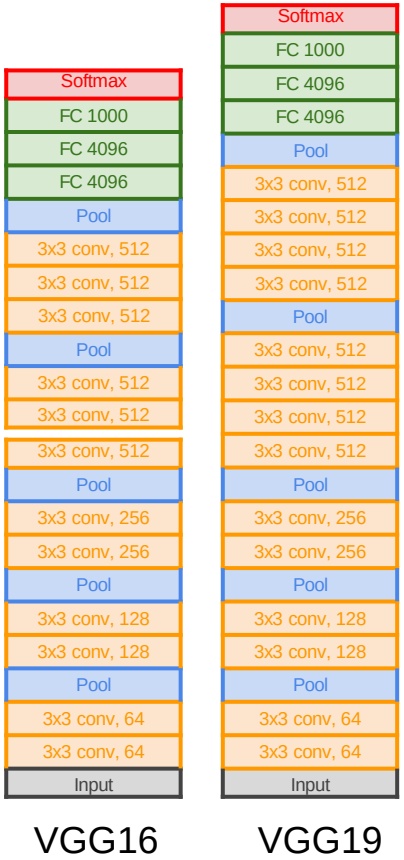


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

[7x7]

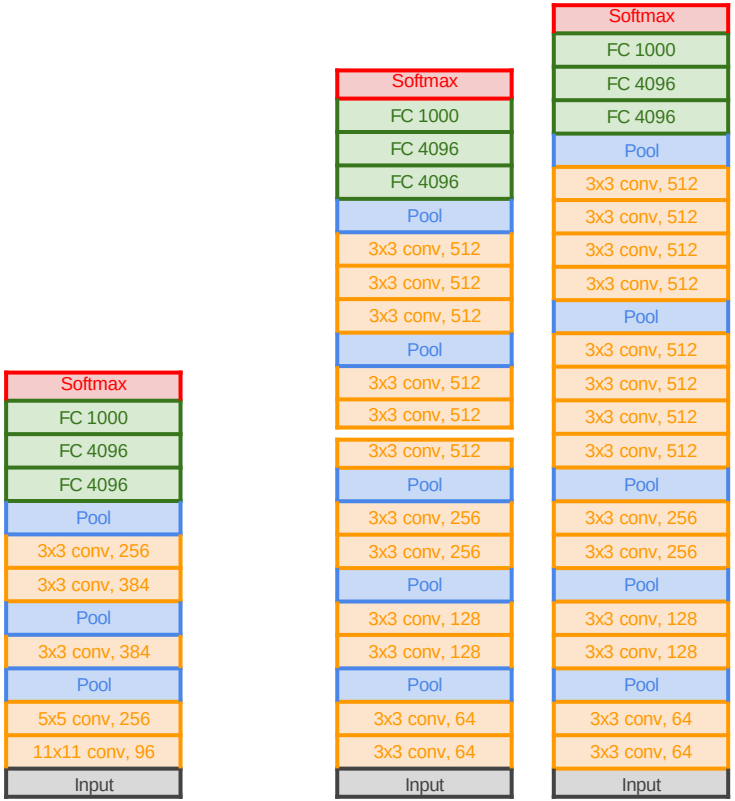


Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

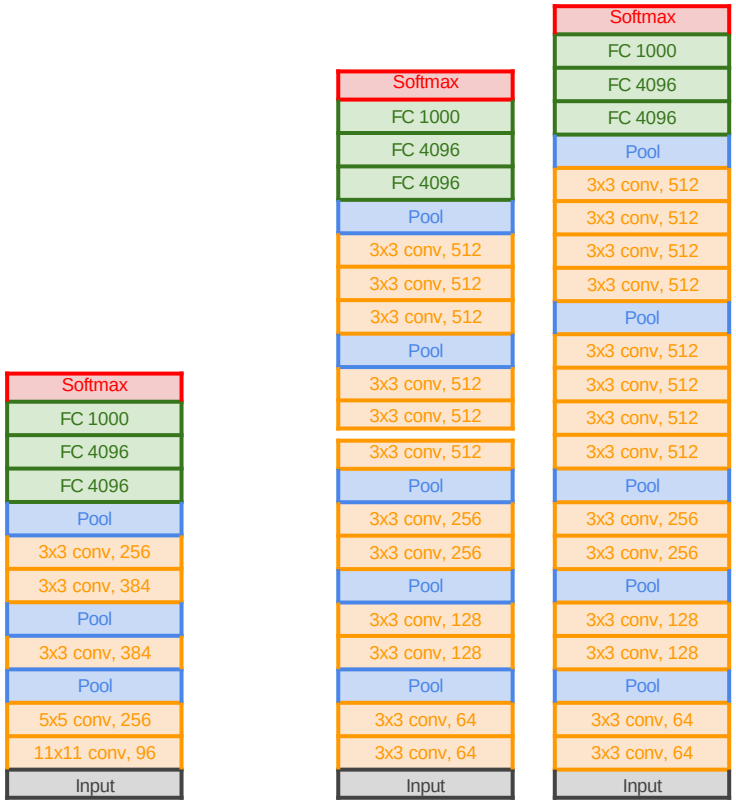
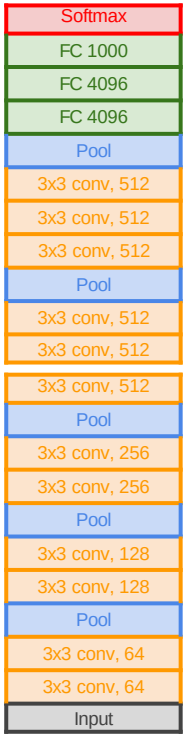


Figure from Stanford cs231n lecture slides

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

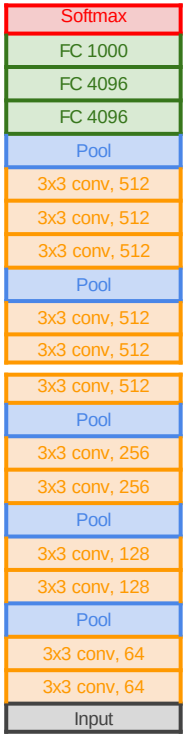


VGG16

Figure from Stanford cs231n lecture slides

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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters



VGG16

Figure from Stanford cs231n lecture slides

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 is in early CONV

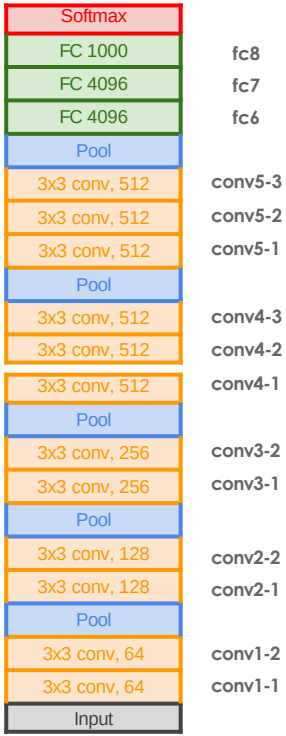
Most params are in late FC

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

Figure from Stanford cs231n lecture slides

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

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



VGG16

Common names

Figure from Stanford cs231n lecture slides

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

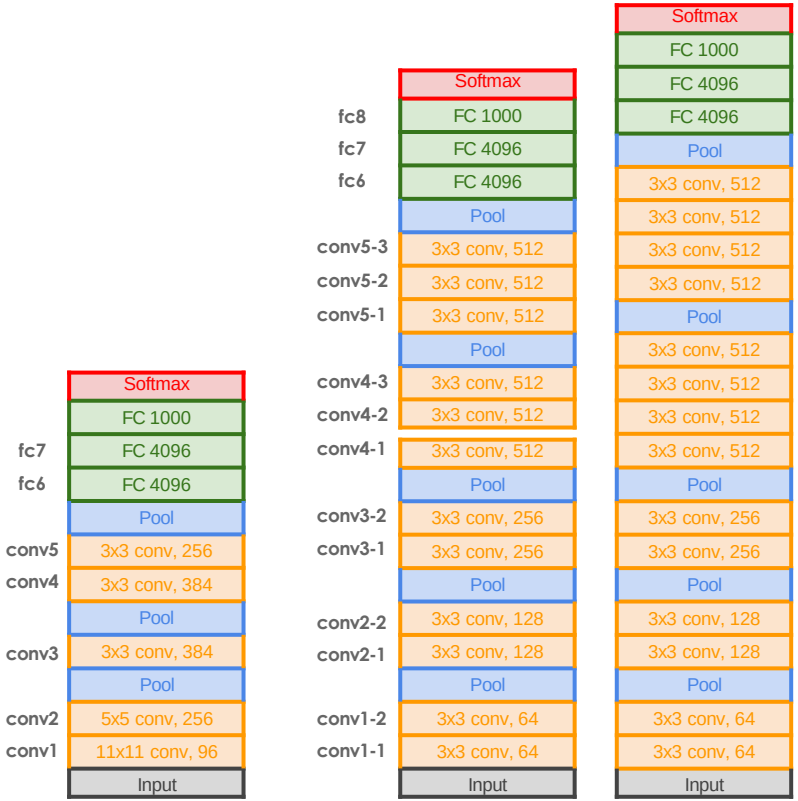


Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

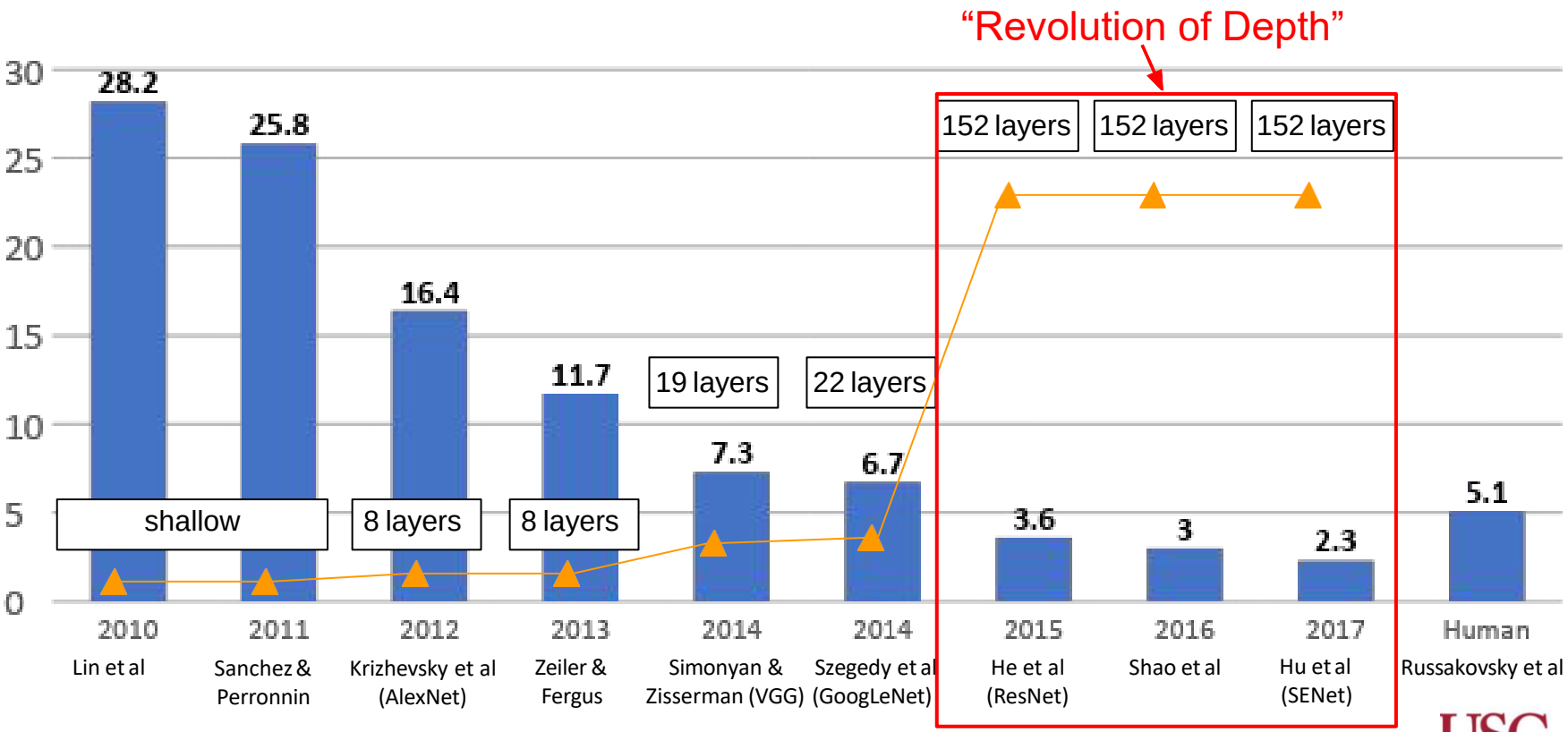


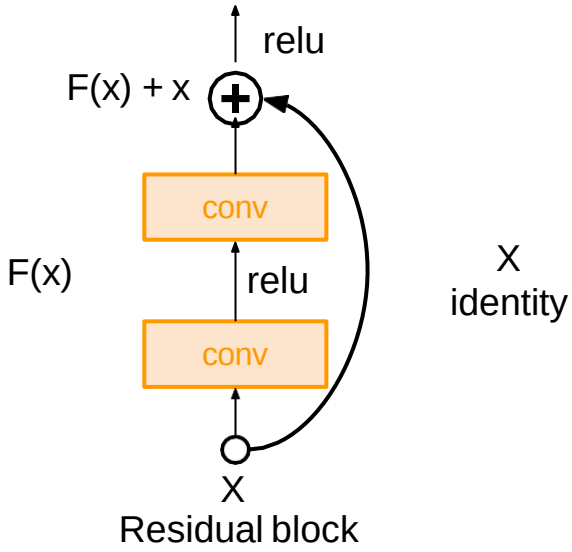
Figure from Stanford cs231n lecture slides

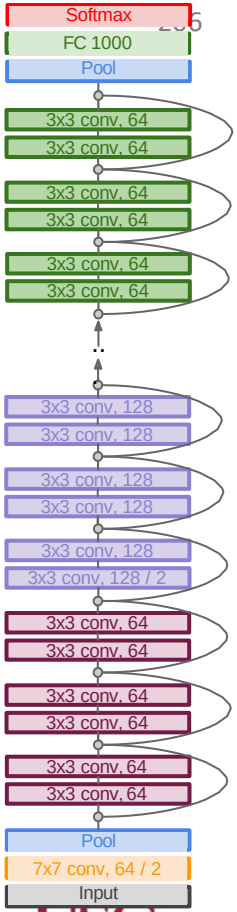
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 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!





USC Viterbi School of Engineering

Case Study: ResNet

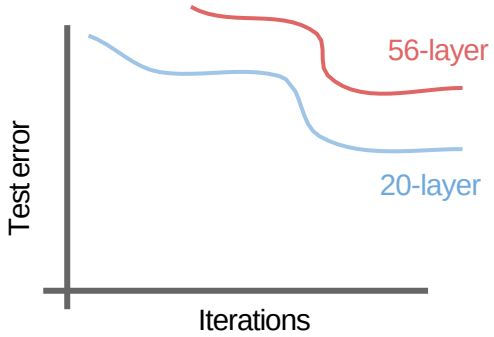
[He et al., 2015]

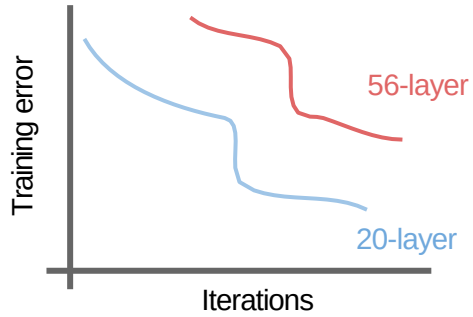
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

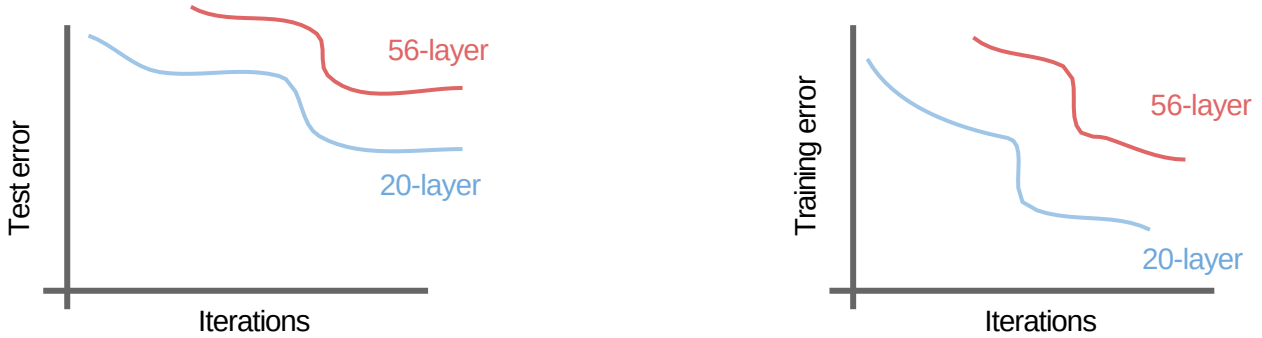




Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, **deeper models are harder to optimize**

Case Study: ResNet

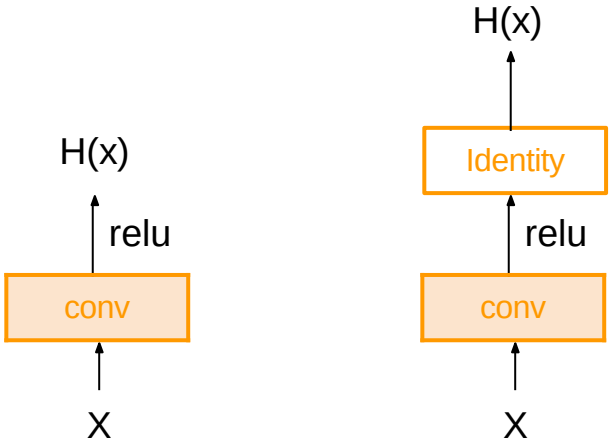
[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

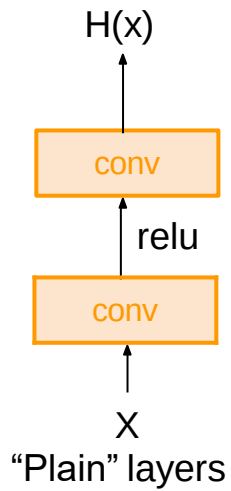
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



Case Study: ResNet

[He et al., 2015]

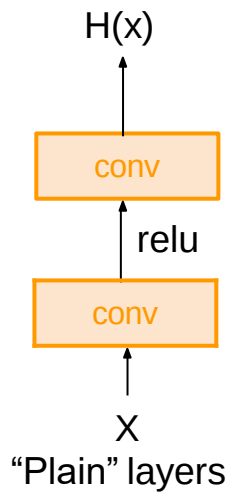
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

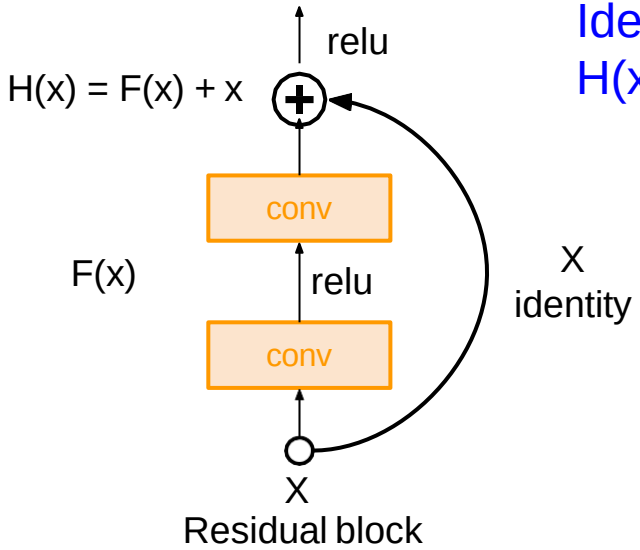


Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



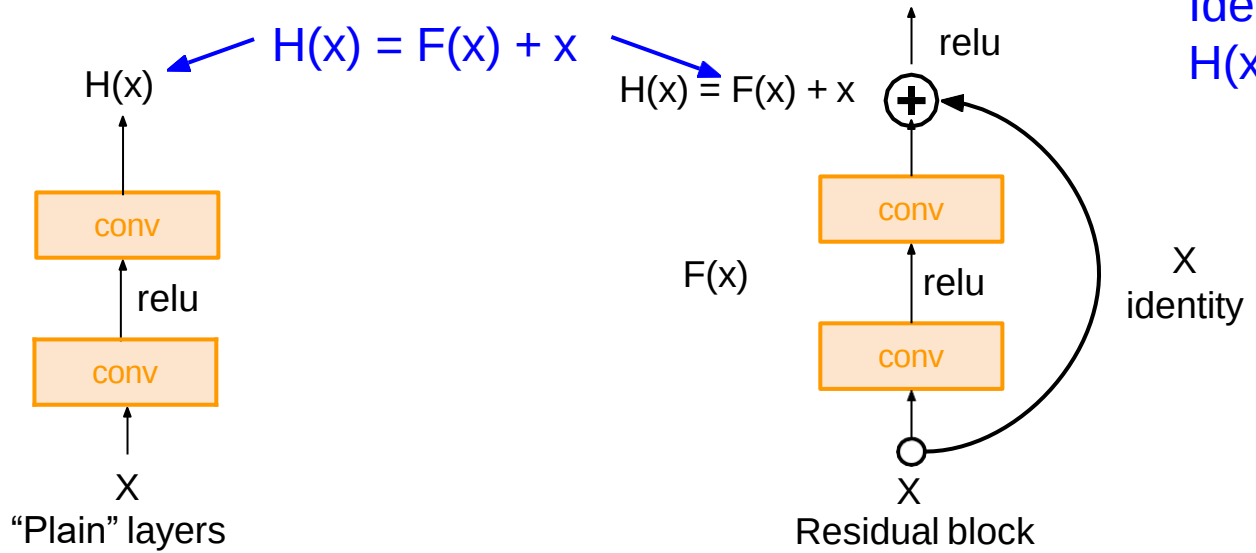


Identity mapping:
H(x) = x if F(x) = 0

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

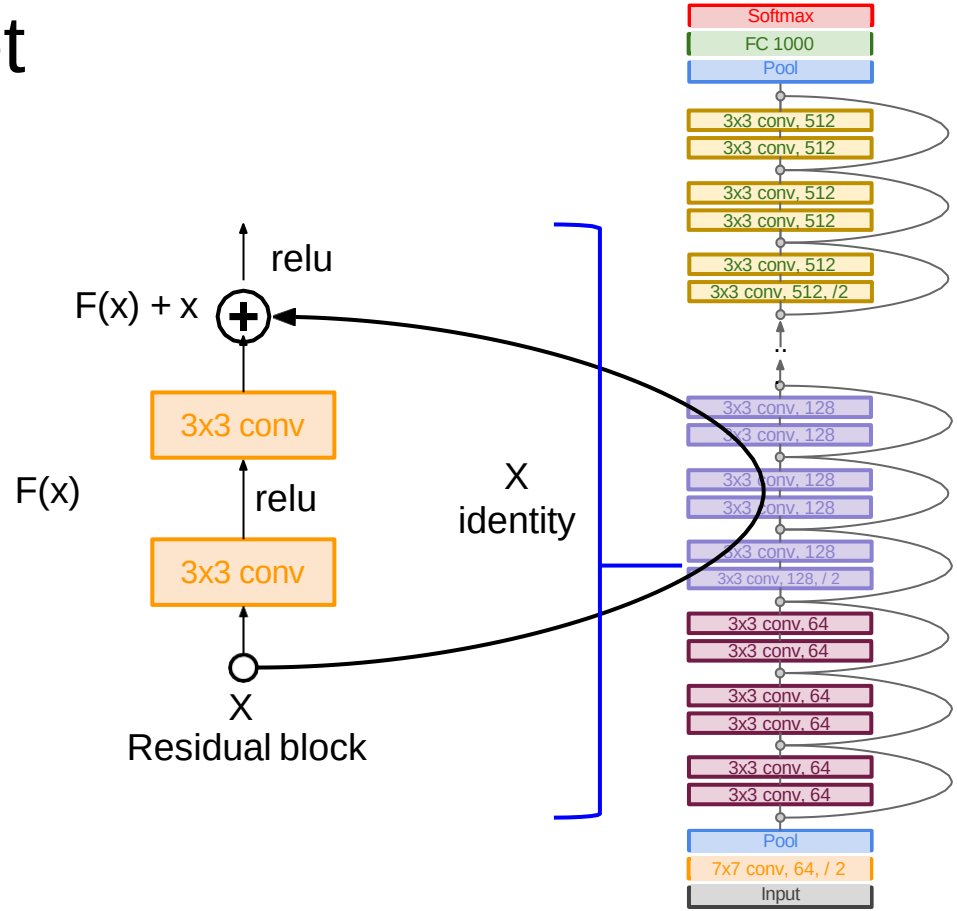
Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

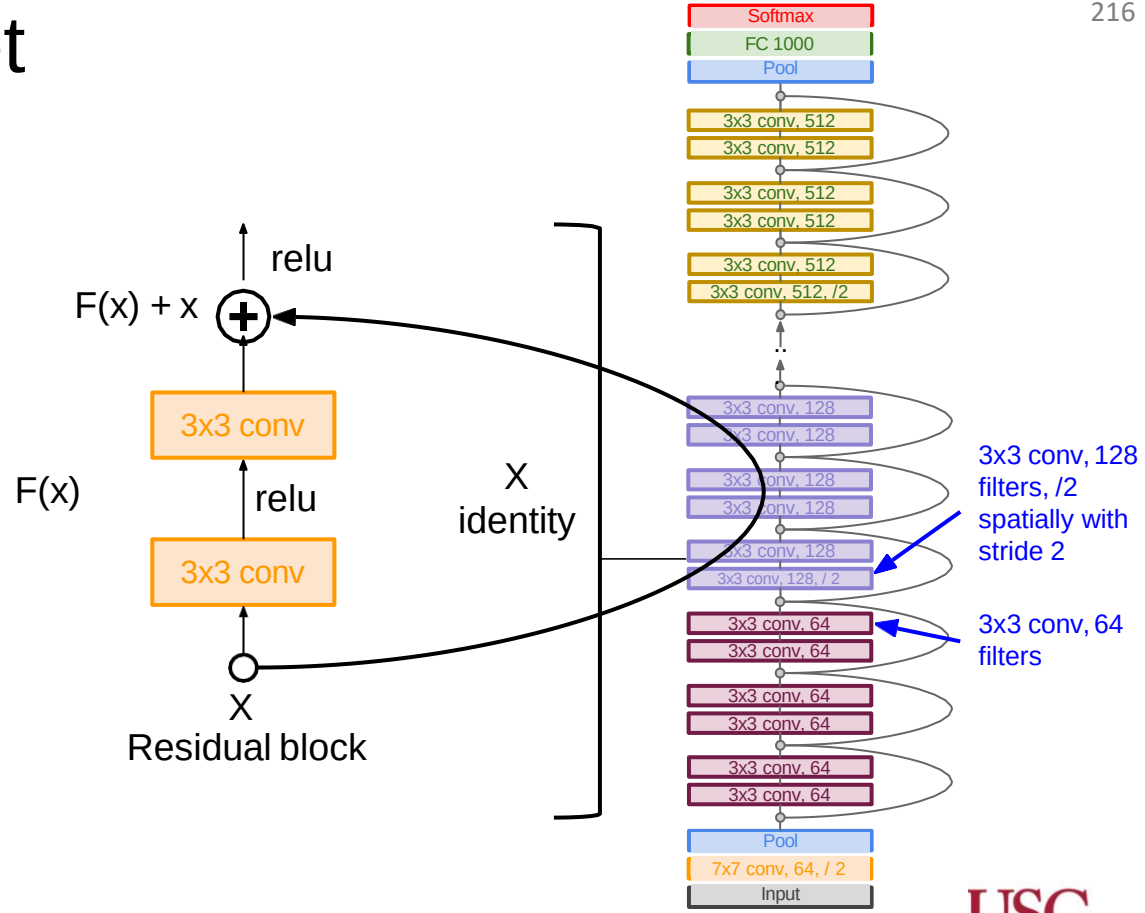


Case Study: ResNet

[He et al., 2015]

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) Reduce the activation volume by half.

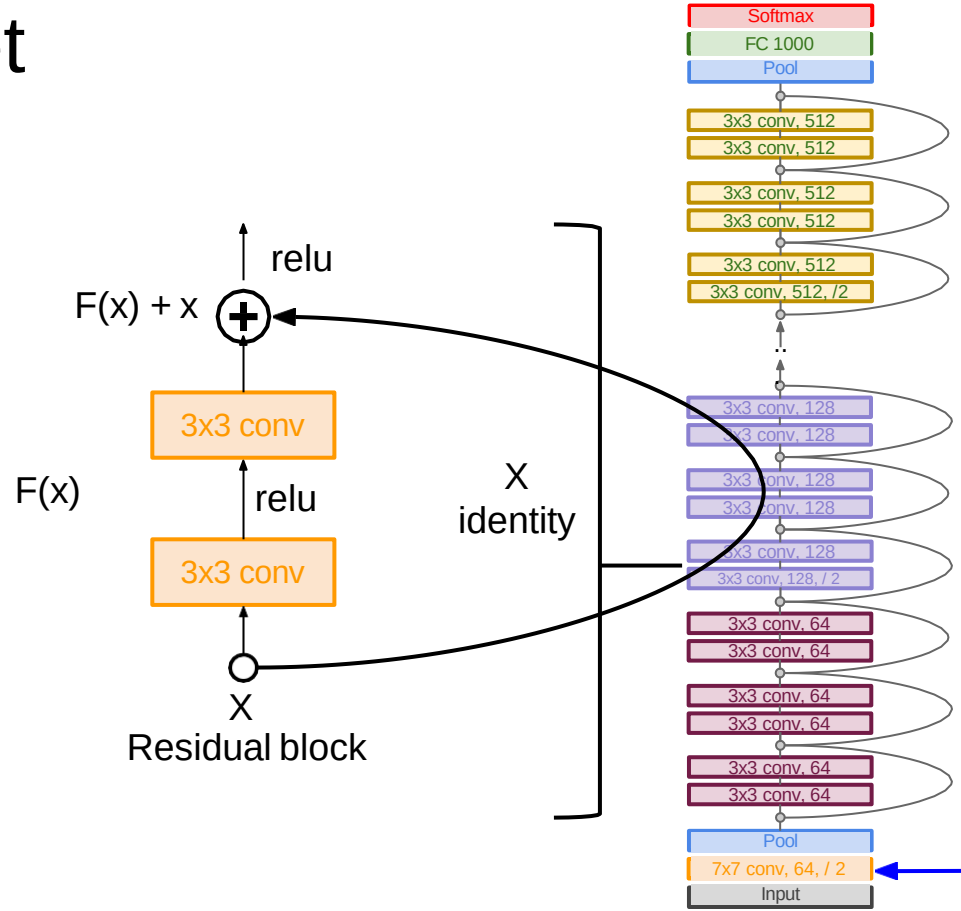


Case Study: ResNet

[He et al., 2015]

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 (stem)

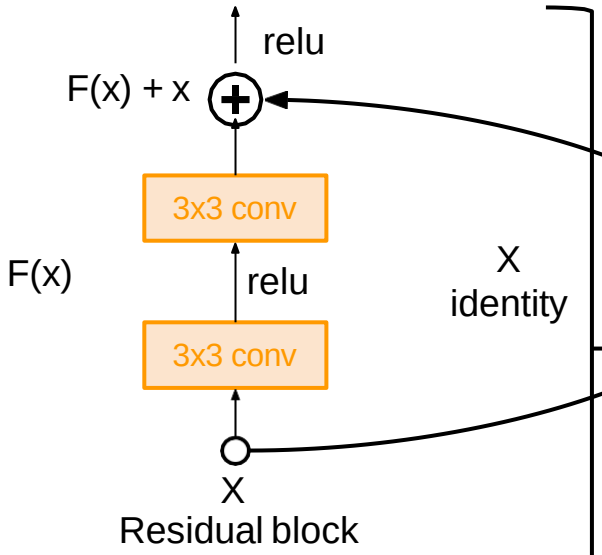


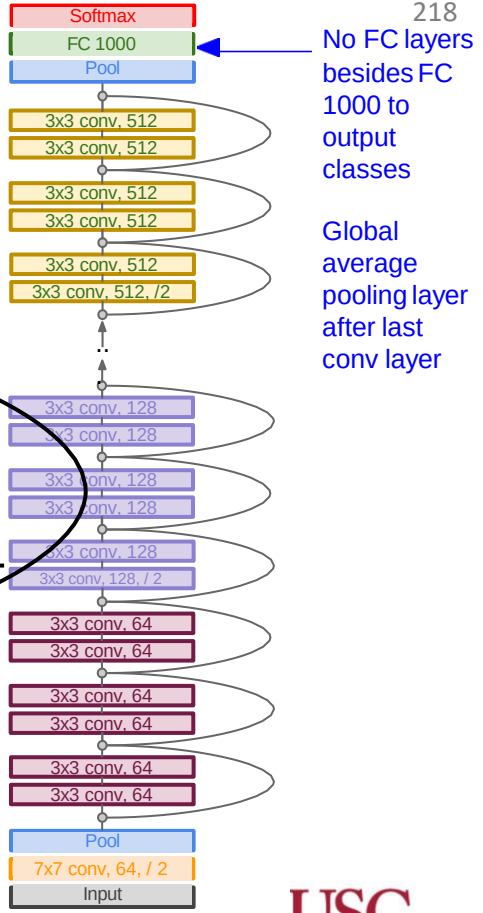
Case Study: ResNet

[He et al., 2015]

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 (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)

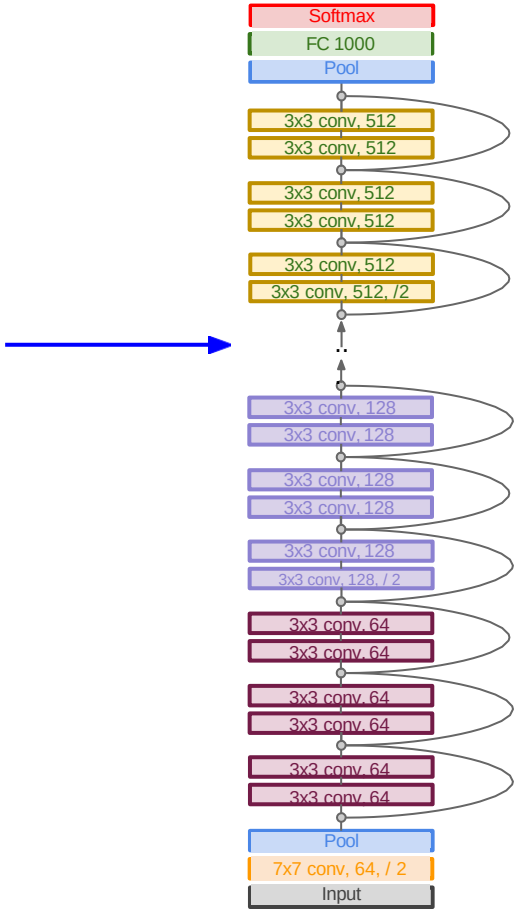




Case Study: ResNet

[He et al., 2015]

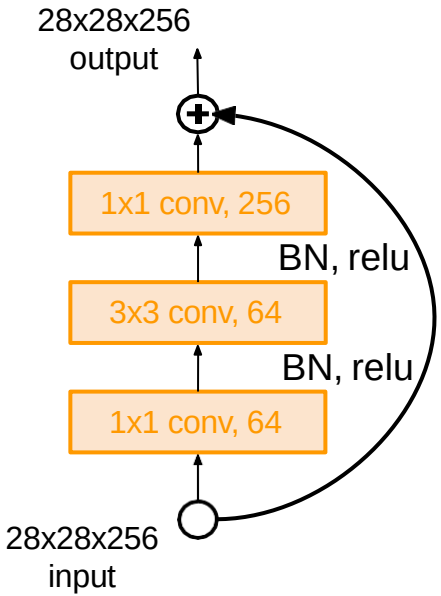
Total depths of 18, 34, 50, 101, or 152 layers for ImageNet



Case Study: ResNet

[He et al., 2015]

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)



Case Study: ResNet

[He et al., 2015]

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

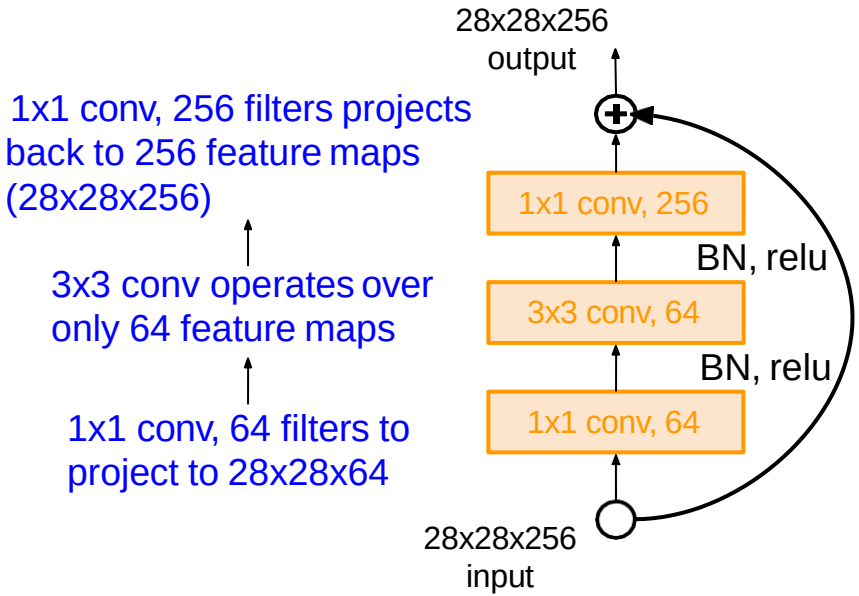


Figure from Stanford cs231n lecture slides

Case Study: ResNet

[He et al., 2015]

Training ResNet in practice (we will cover how to train later!):

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Case Study: ResNet

[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

Figure from Stanford cs231n lecture slides

Case Study: ResNet

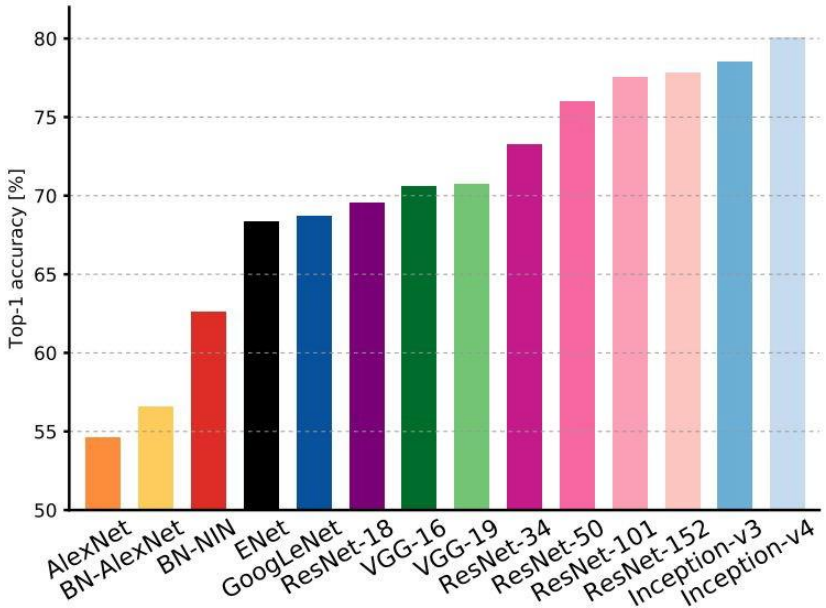
[He et al., 2015]

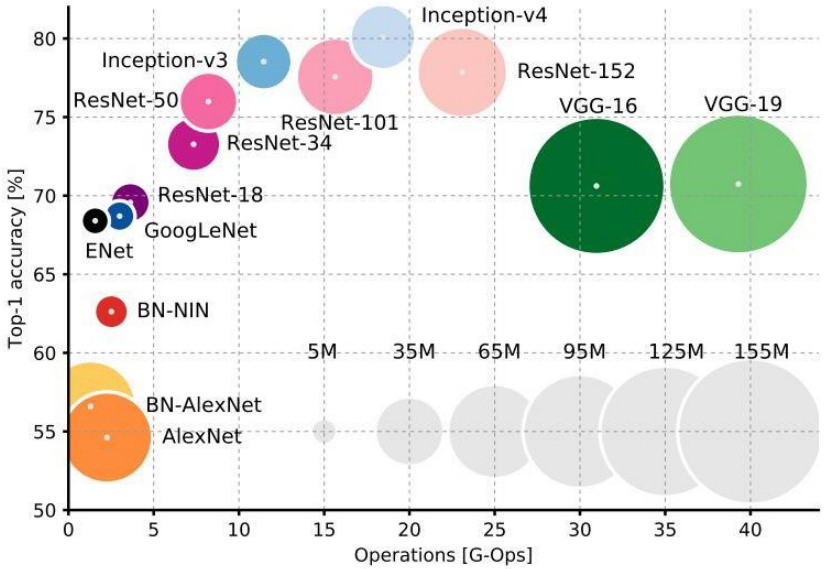
Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

Comparing complexity...

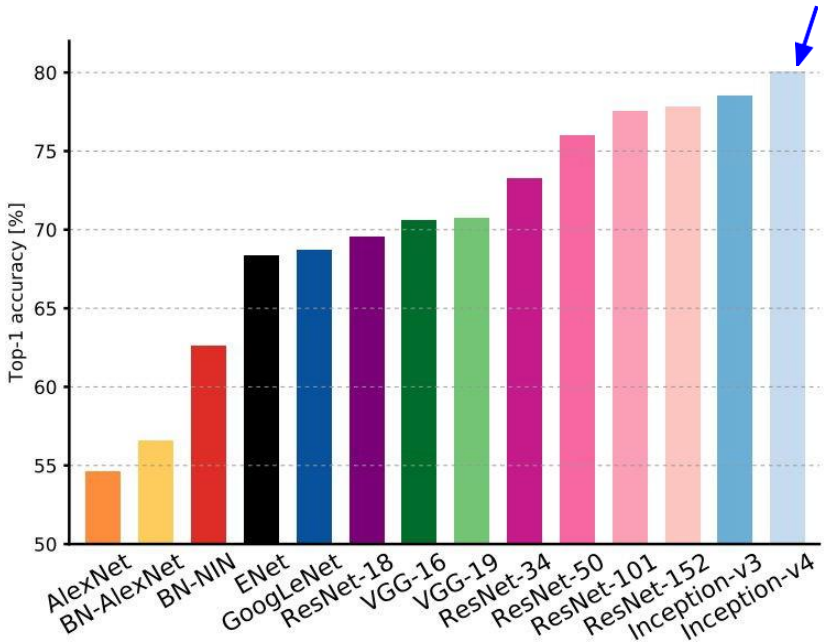


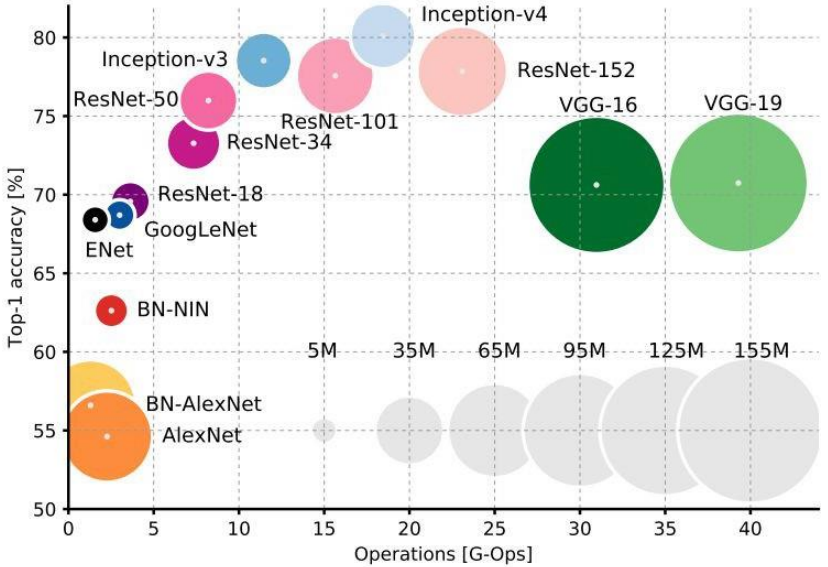


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

Inception-v4: Resnet + Inception!

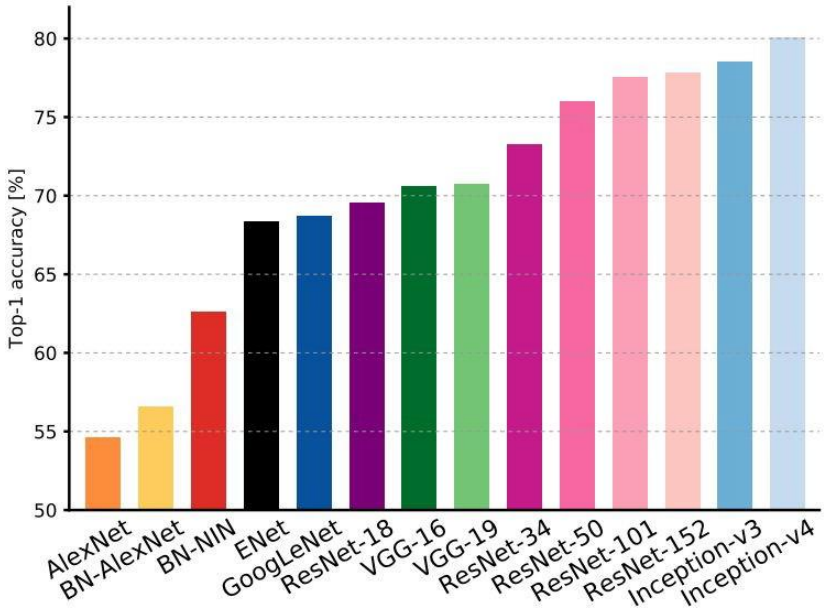


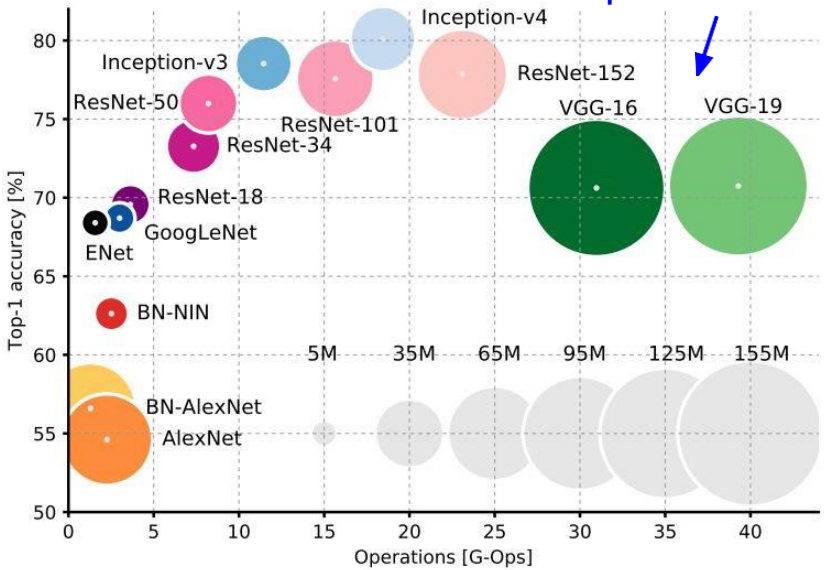


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

VGG: most parameters, most operations

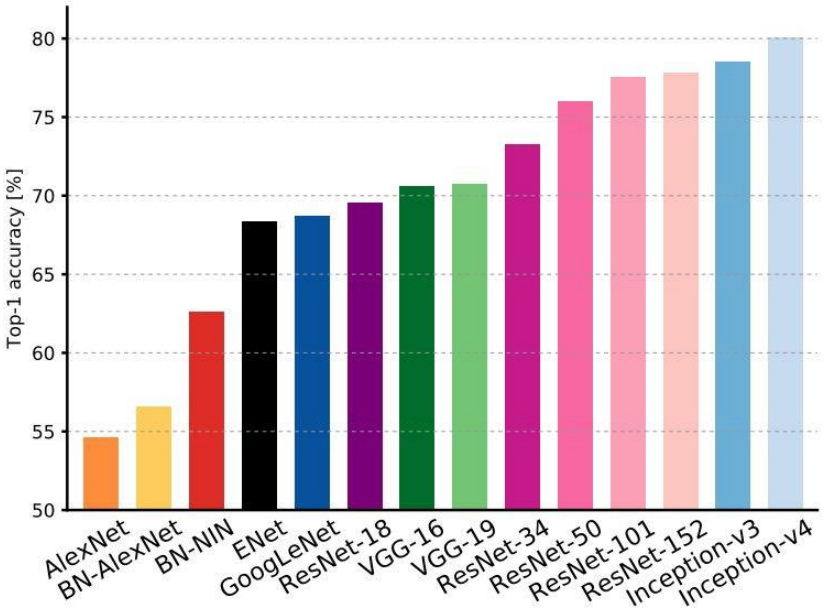


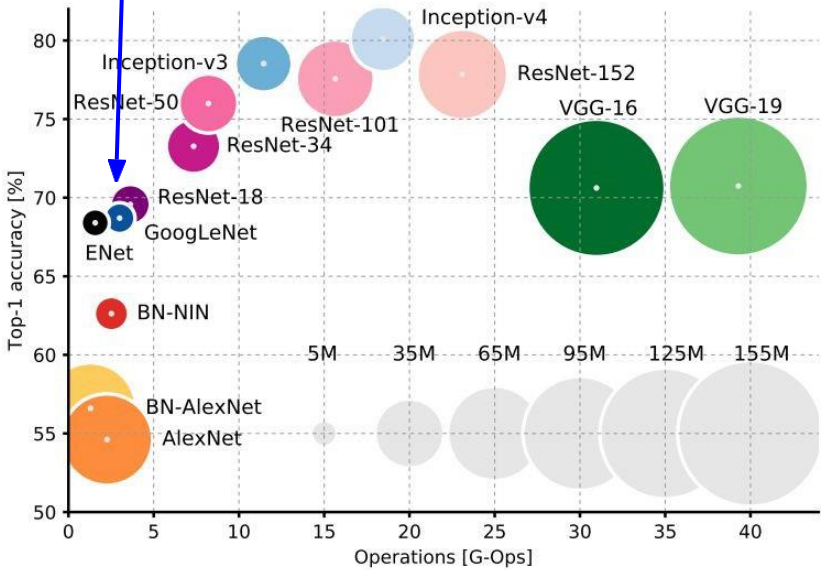


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

GoogLeNet:
most efficient



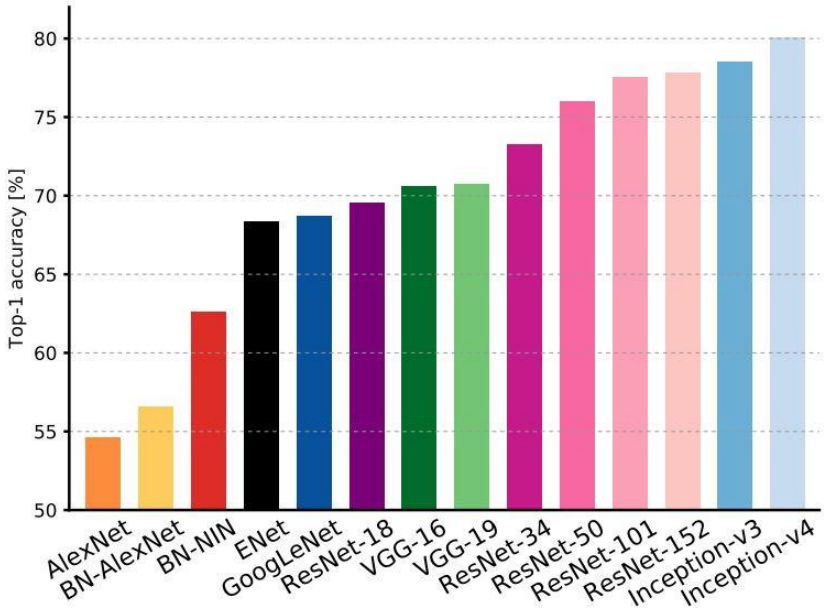


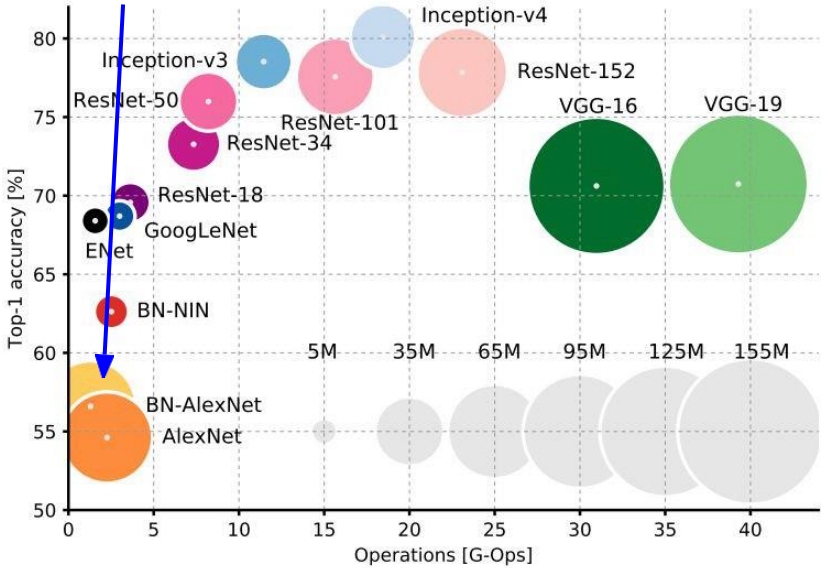
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figure from Stanford cs231n lecture slides

Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy



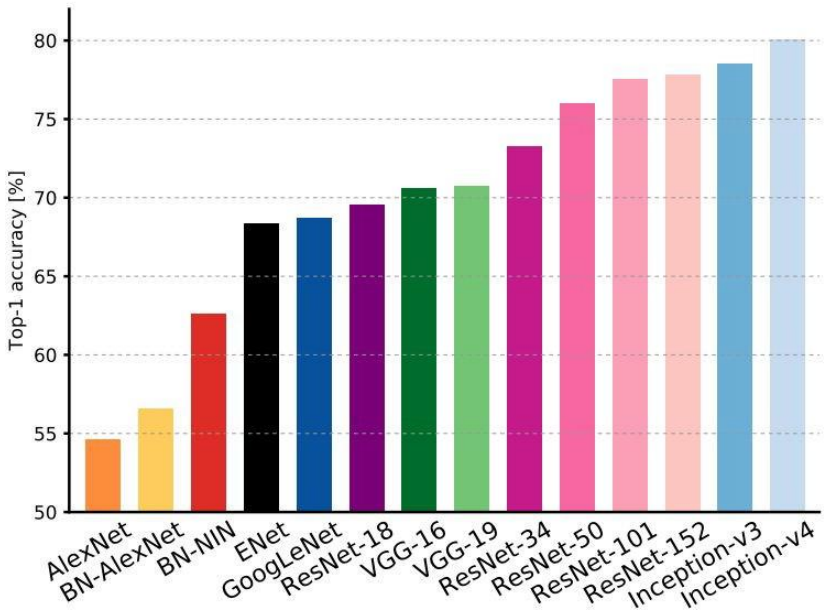


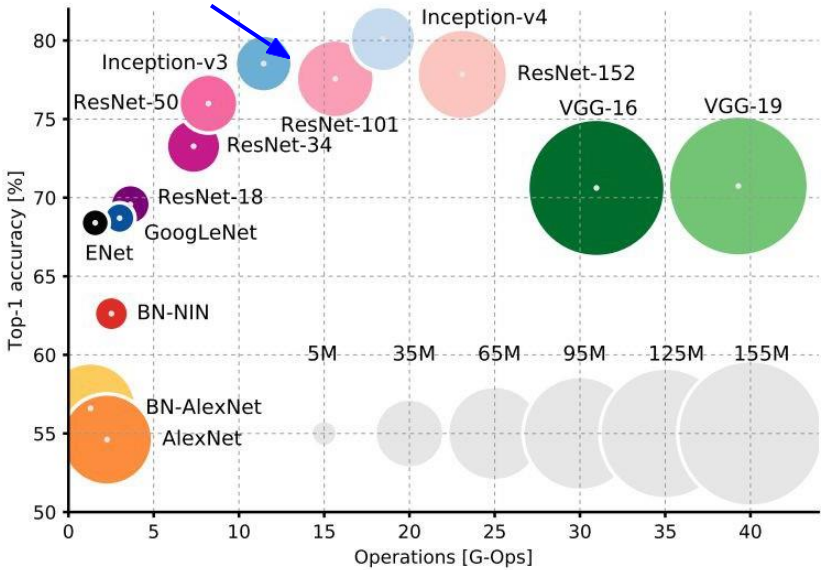
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figure from Stanford cs231n lecture slides

Comparing complexity...

ResNet:
Moderate efficiency depending on model, highest accuracy





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

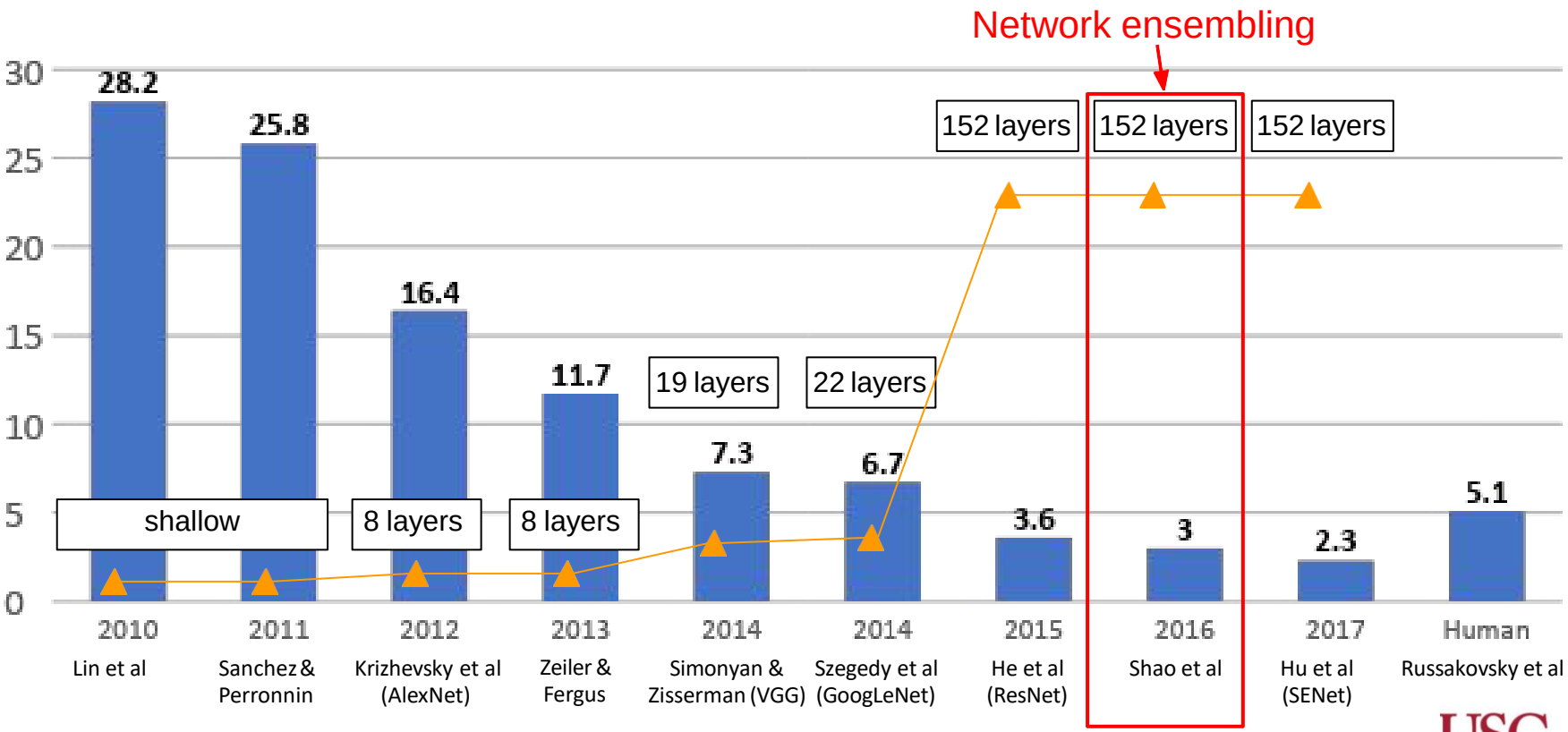


Figure from Stanford cs231n lecture slides

Improving ResNets...

“Good Practices for Deep Feature Fusion”

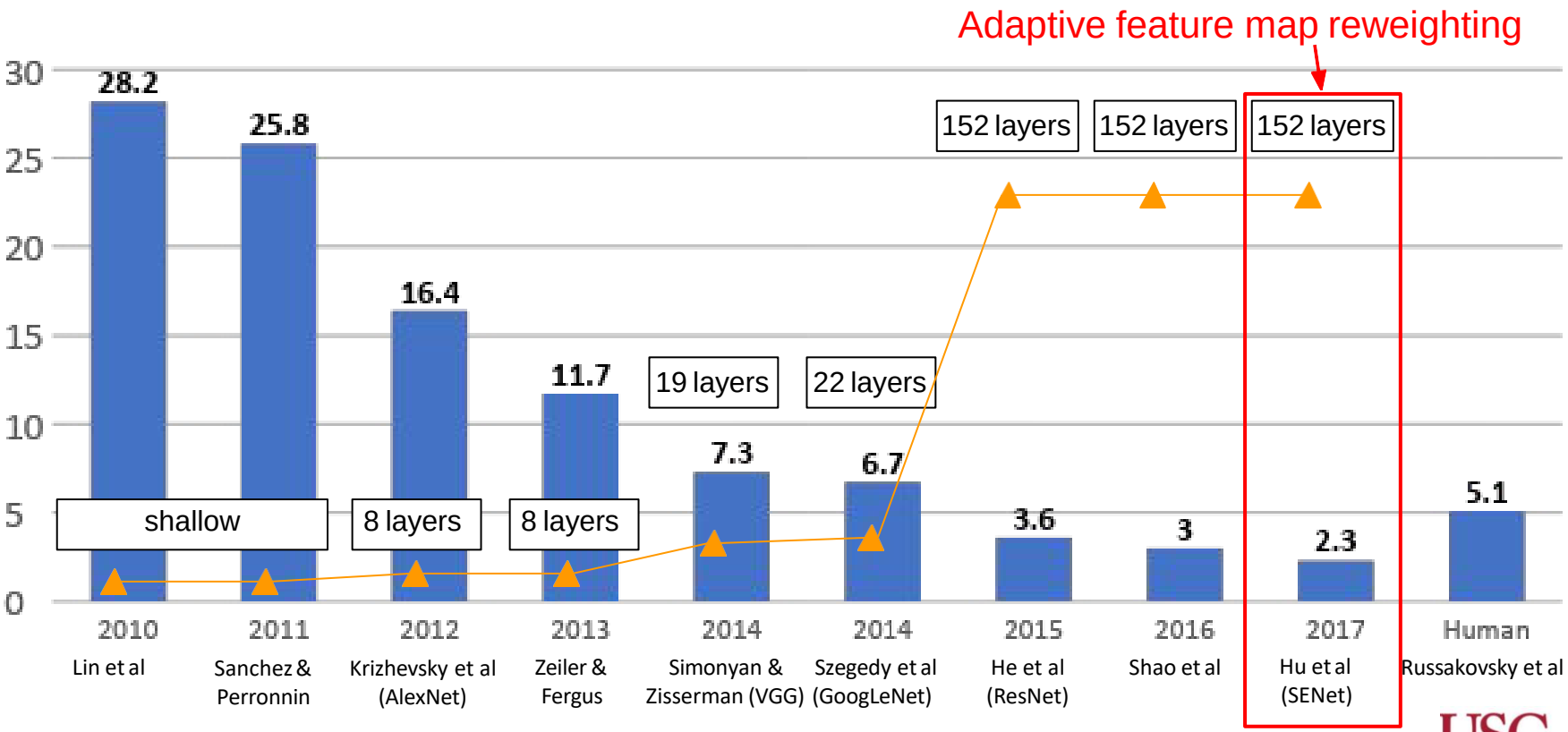
[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

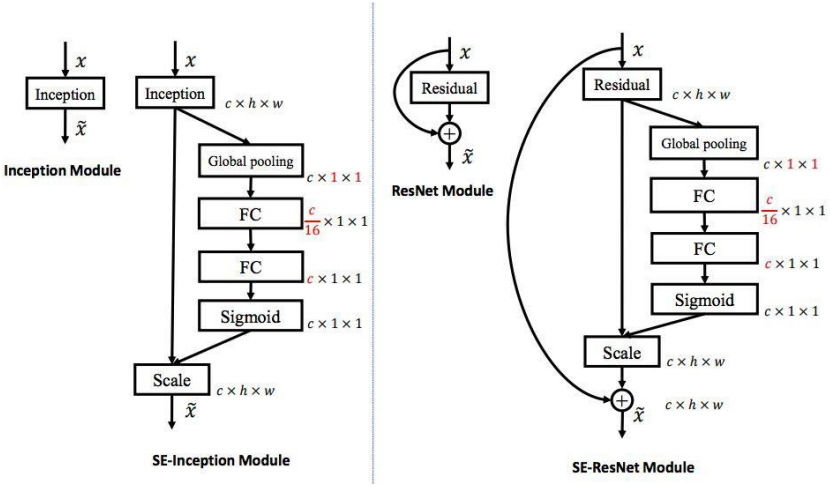


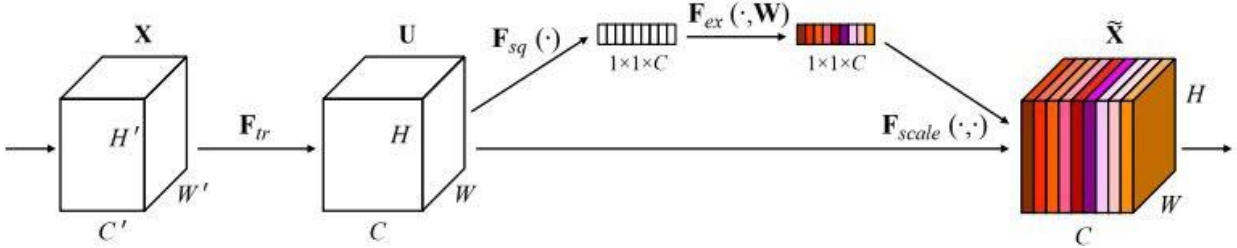
Improving ResNets...

Squeeze-and-Excitation Networks (SENet)

[Hu et al. 2017]

- Add a "feature recalibration" module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)





ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

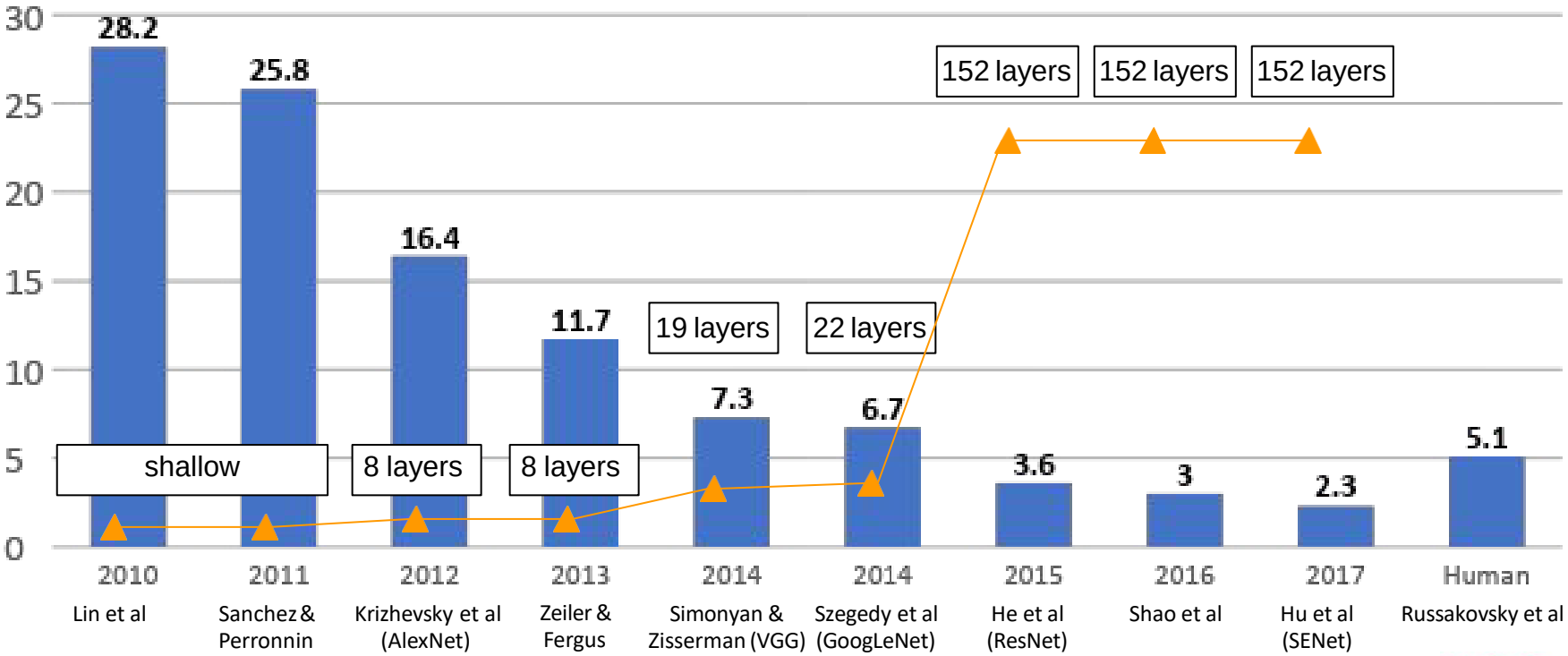
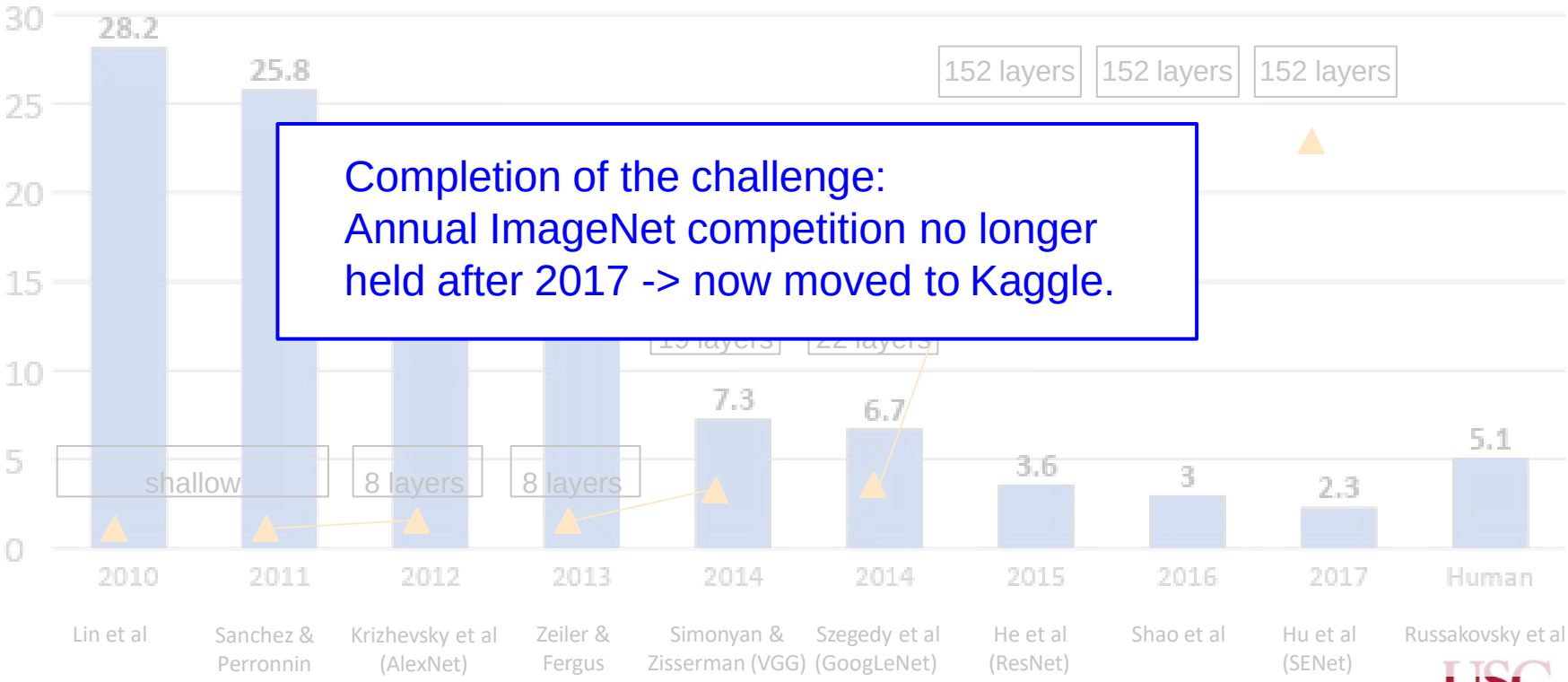


Figure from Stanford cs231n lecture slides

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Completion of the challenge:
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.

Figure from Stanford cs231n lecture slides

Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network
- Gives better performance

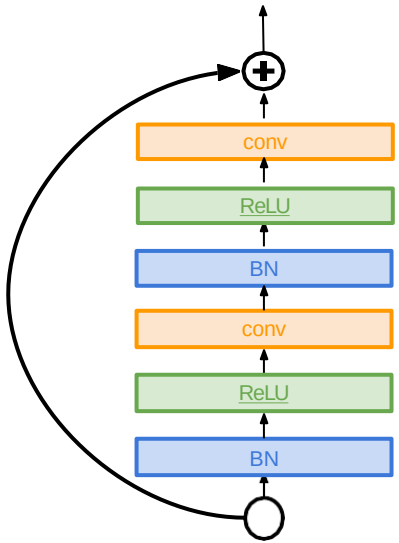
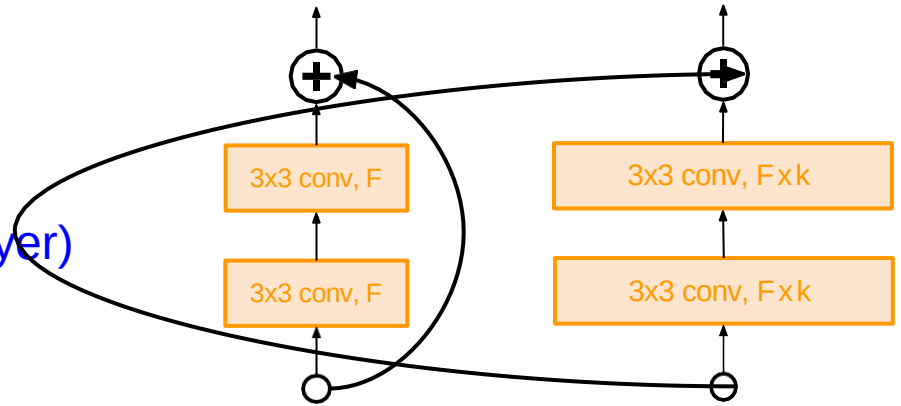


Figure from Stanford cs231n lecture slides

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)

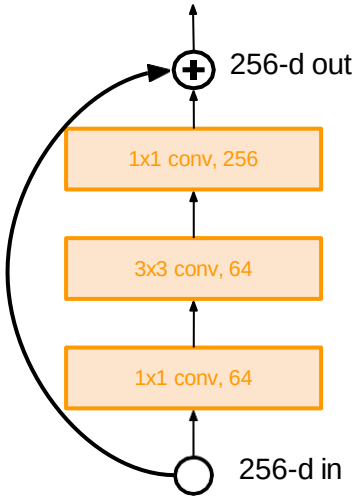


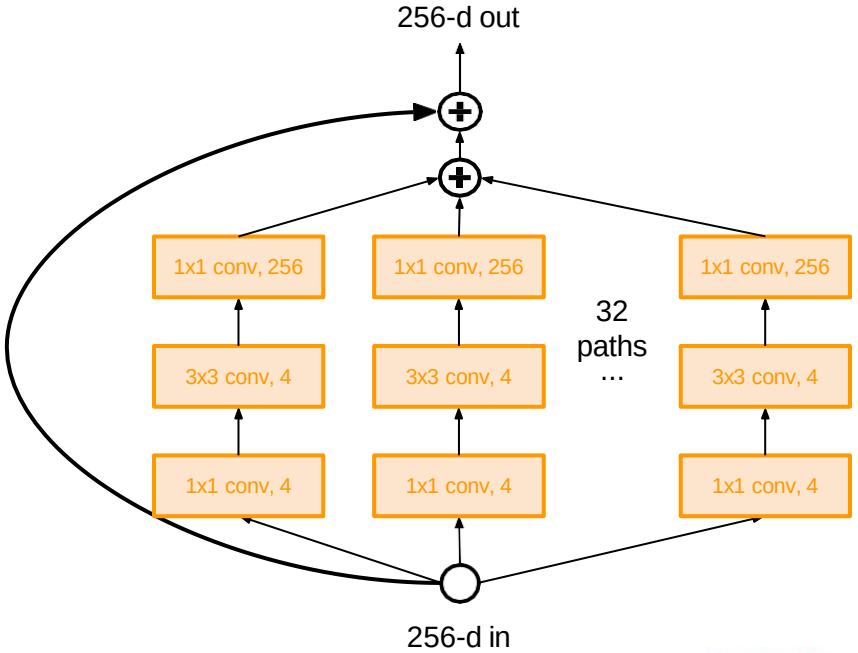
Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



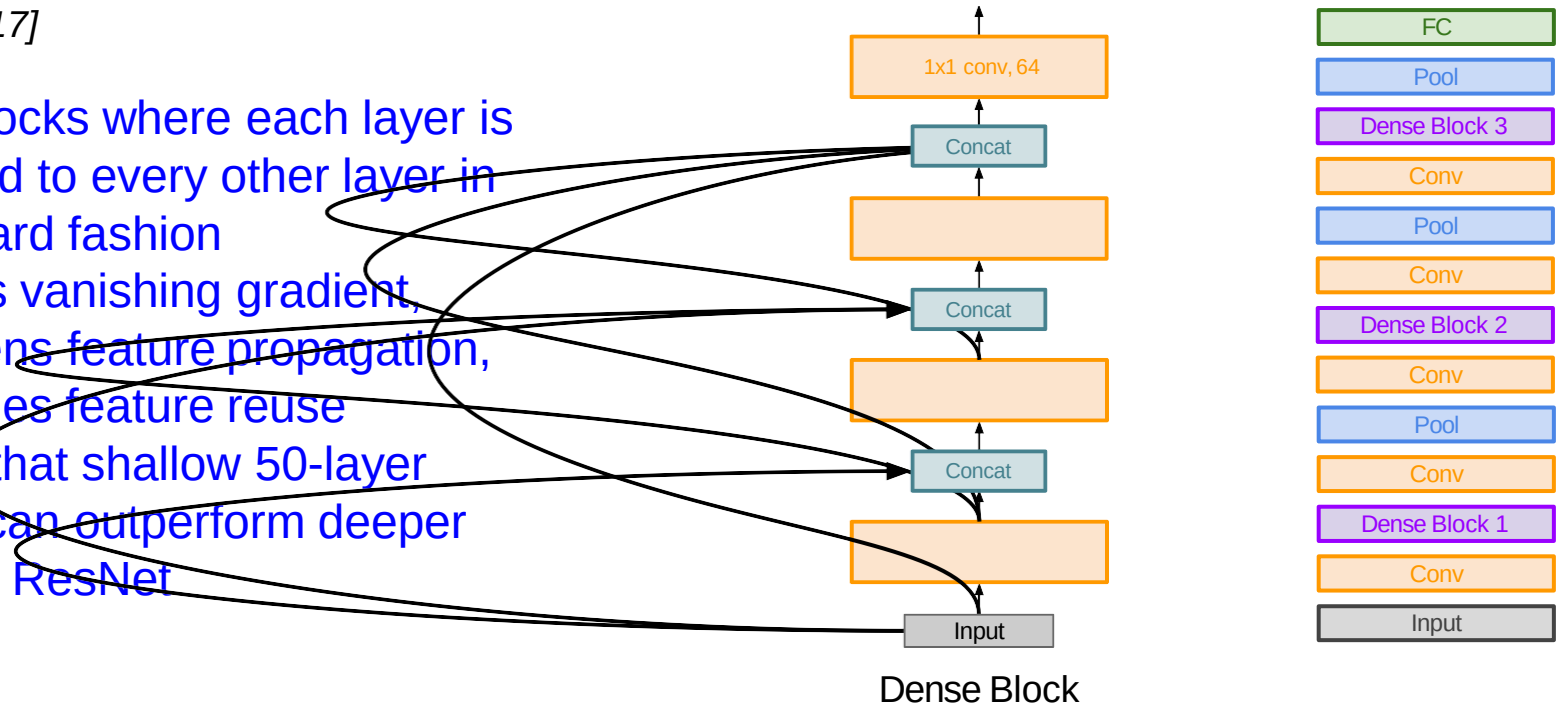


Other ideas...

Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

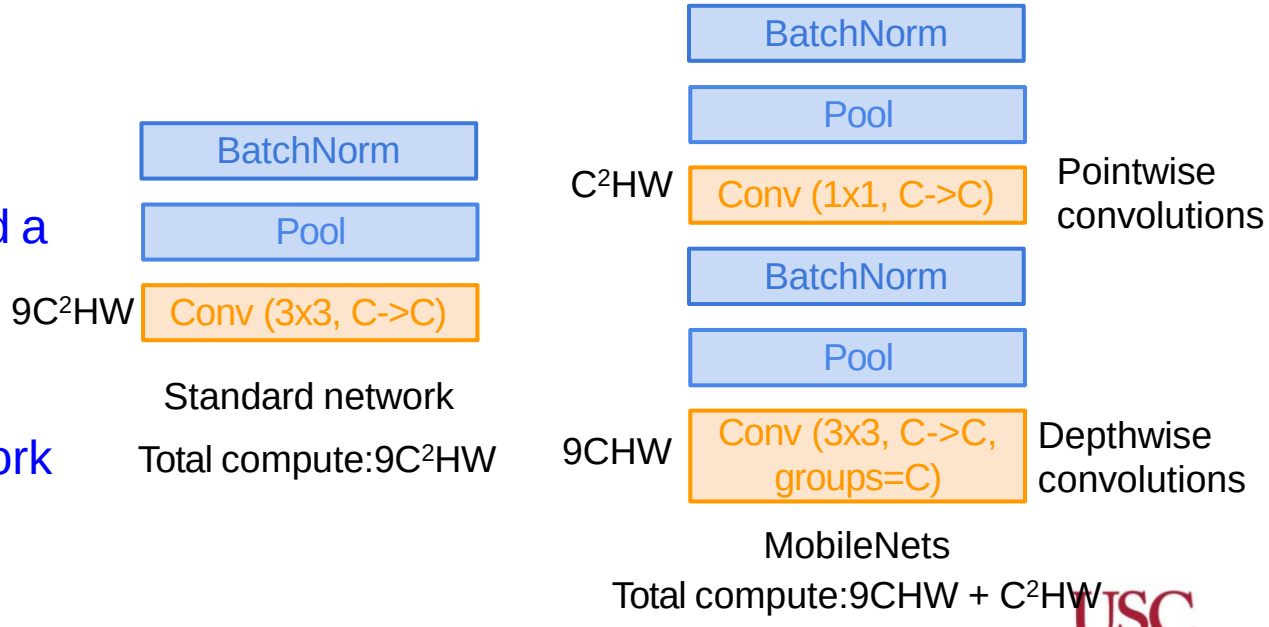
- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



Efficient networks...

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications *[Howard et al. 2017]*

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018



Learning to search for network architectures...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)

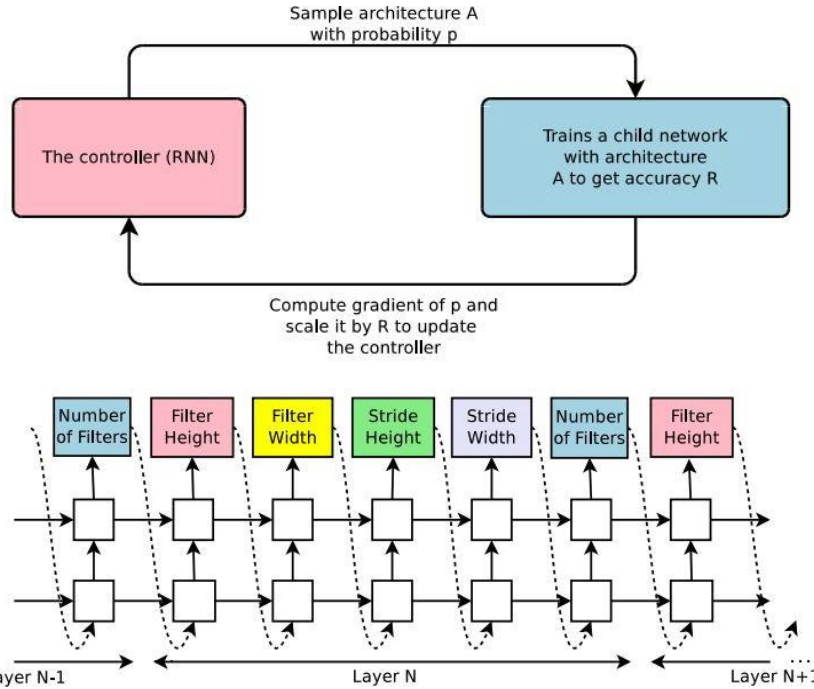


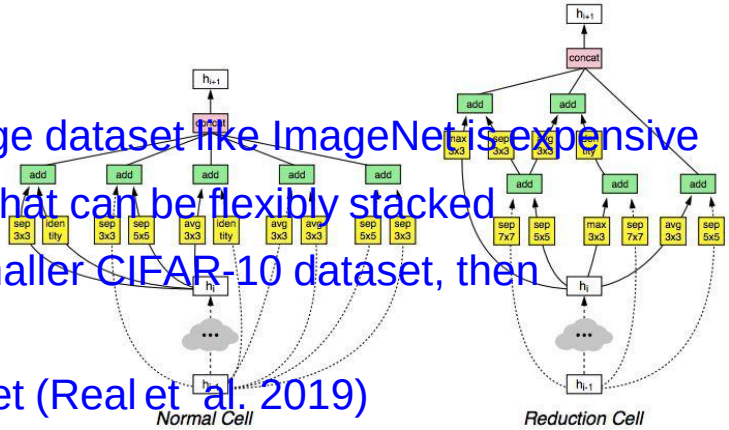
Figure from Stanford cs231n lecture slides

Learning to search for network architectures...

Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks ("cells") that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet
- Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)



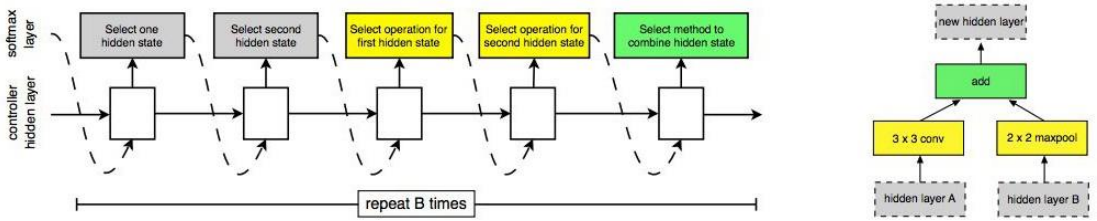


Figure from Stanford cs231n lecture slides

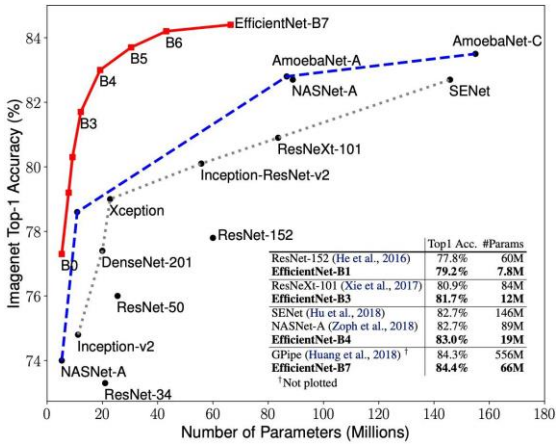
But sometimes smart heuristic is better than NAS ...

EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



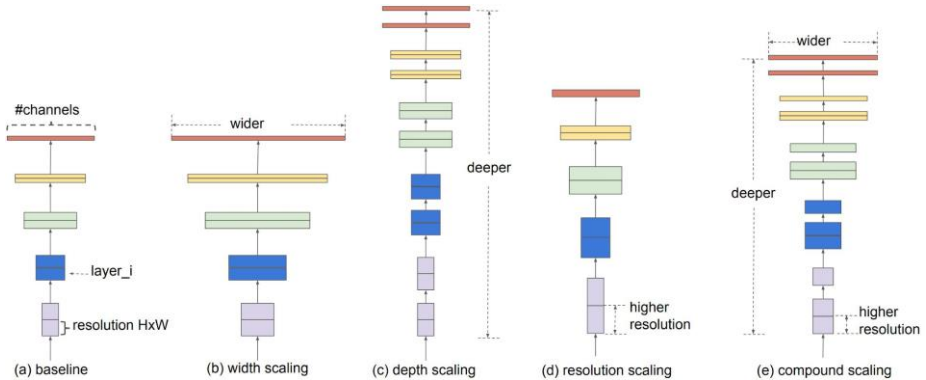


Figure from Stanford cs231n lecture slides

Efficient networks.

https://openai.com/blog/ai-and-efficiency/

Summary: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet

Main takeaways

AlexNet showed that you can use CNNs to train Computer Vision models.
ZFNet, **VGG** shows that bigger networks work better
GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
ResNet showed us how to train extremely deep networks

- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to **Efficient networks**:

- Lots of tiny networks aimed at mobile devices: **MobileNet**, **ShuffleNet**

Neural Architecture Search can now automate architecture design (covered in the later part of the course).

Summary: CNN Architectures

- Many popular architectures are available in model zoos.
- ResNets are currently good defaults to use.
- Networks have gotten increasingly deep over time.
- Many other aspects of network architectures are also continuously being investigated and improved.

MODEL ZOO

This page lists model archives that are pre-trained and pre-packaged, ready to be served for inference with TorchServe. To propose a model for inclusion, please submit a pull request.

Special thanks to the PyTorch community whose Model Zoo and Model Examples were used in generating these model archives.

Model	Type	Dataset	Size	Download	Sample Input	Model mode
AlexNet	Image Classification	ImageNet	216 MB	.mar	kitten.jpg	Eager
Densenet161	Image Classification	ImageNet	106 MB	.mar	kitten.jpg	Eager
Resnet18	Image Classification	ImageNet	41 MB	.mar	kitten.jpg	Eager
VGG16	Image Classification	ImageNet	489 MB	.mar	kitten.jpg	Eager

CNN applications

- Image classification
- Object detection
- Object segmentation
- Image resolution
- Human pose detection
- Action classification
- Image captioning

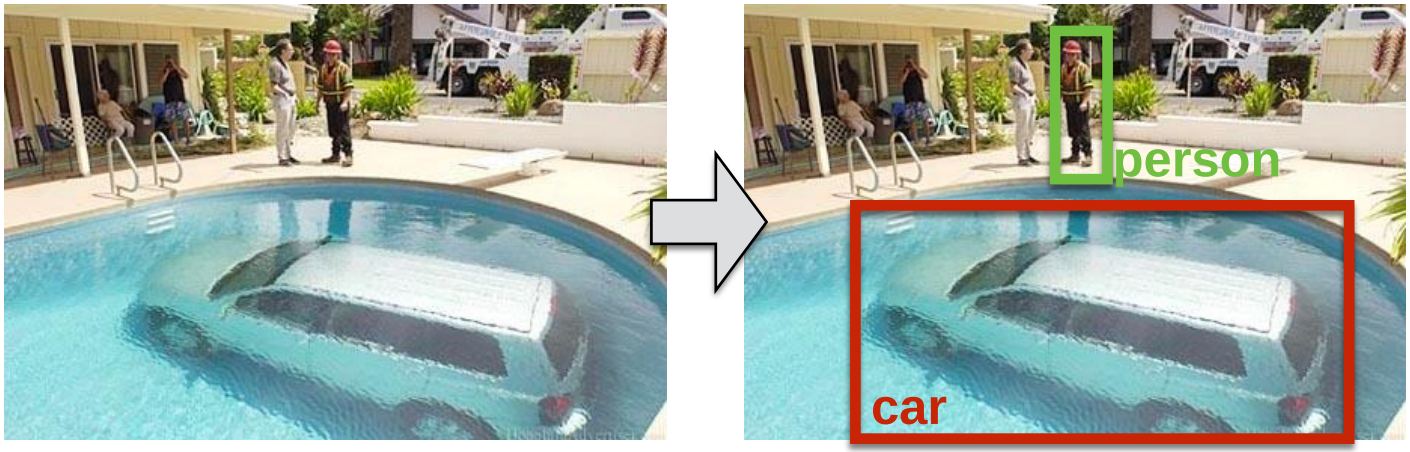
Image Classification

Car

Image Classification

Object Detection

Object Detection



Object Detection

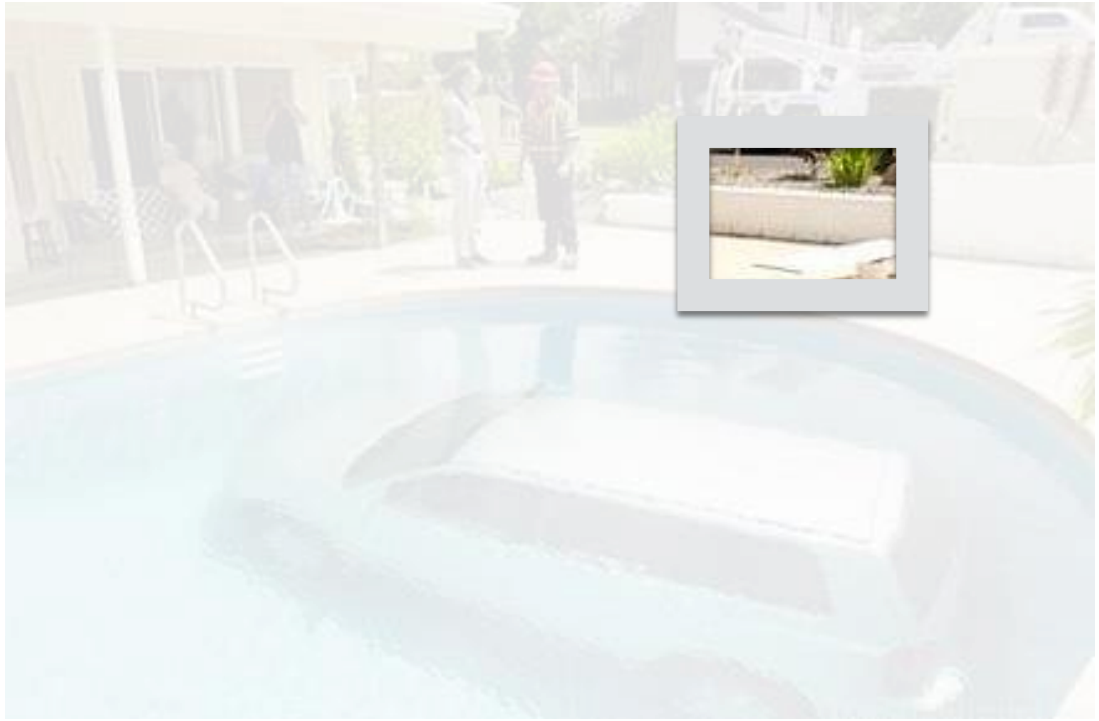
Object Detection

1. Object proposal

Object Detection

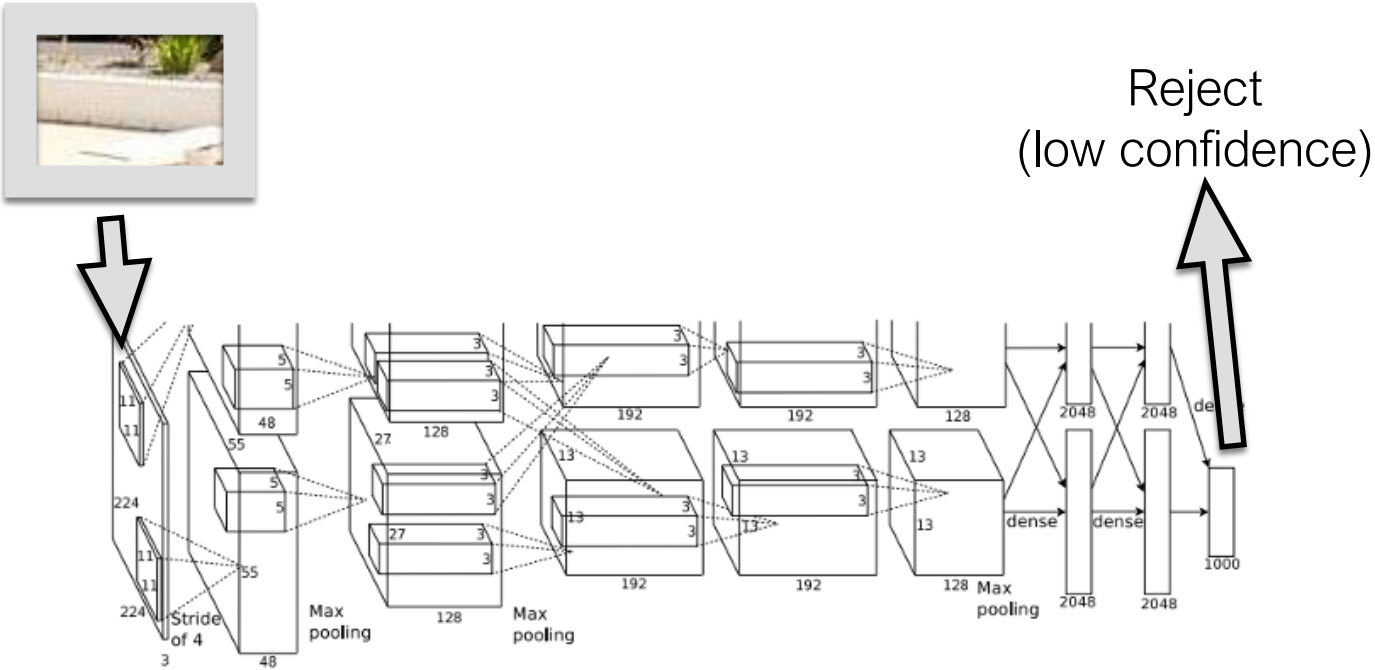
2. Evaluate each proposal

Object Detection



2. Evaluate each proposal

Object Detection

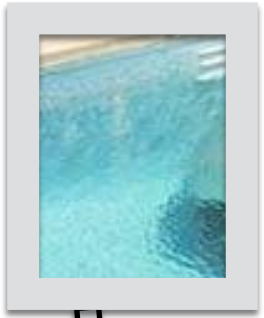


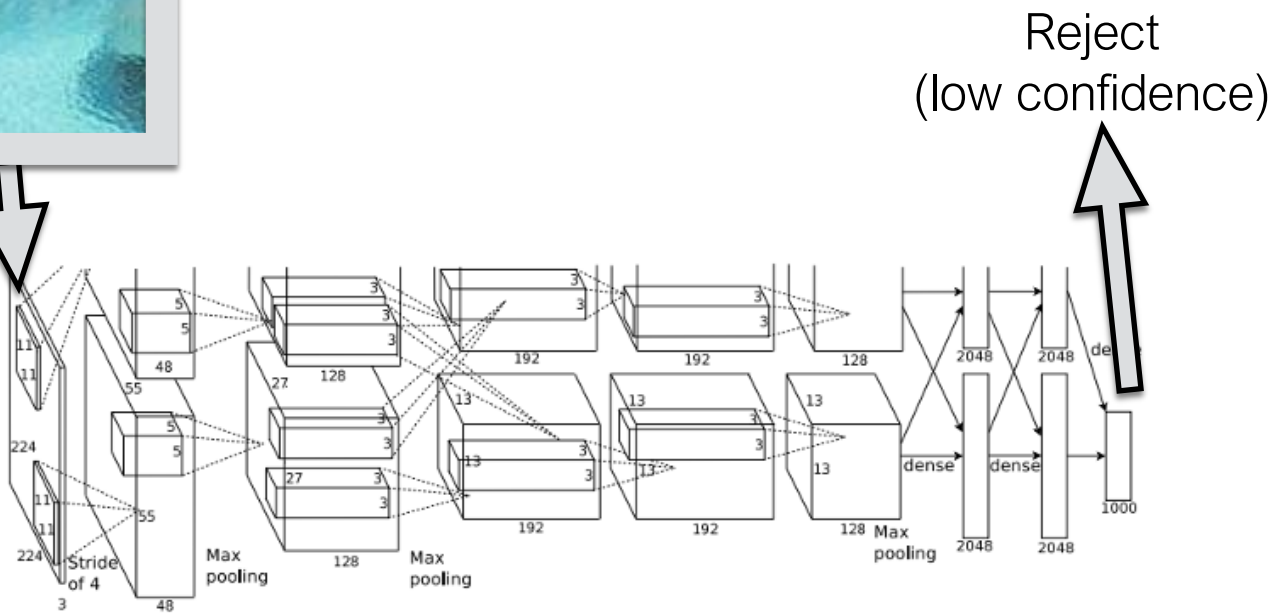
2. Evaluate each proposal

Object Detection

2. Evaluate each proposal

Object Detection



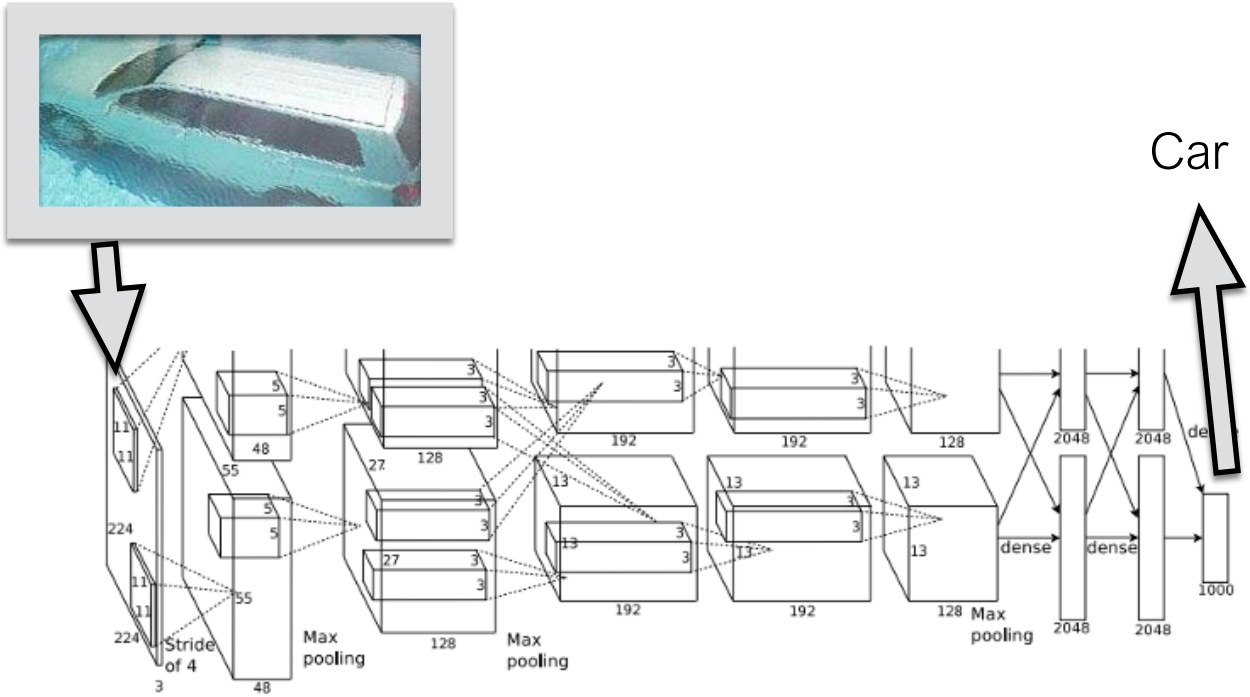


2. Evaluate each proposal

Object Detection

2. Evaluate each proposal

Object Detection



2. Evaluate each proposal

Object Detection

For more:
R-CNN (https://github.com/rbgirshick/rcnn) and Fast R-CNN (https://github.com/rbgirshick/fast-rcnn)

USC Viterbi School of Engineering

Demo

- 3D Face Reconstruction from a Single Image

  http://www.cs.nott.ac.uk/~psxasj/3dme/

Demo

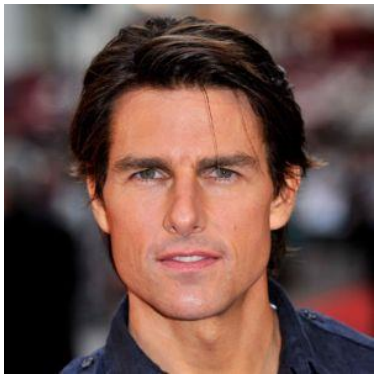
- 3D Face Reconstruction from a Single Image

  http://www.cs.nott.ac.uk/~psxasj/3dme/

Demo

- 3D Face Reconstruction from a Single Image

  http://www.cs.nott.ac.uk/~psxasj/3dme/



Demo

- 3D Face Reconstruction from a Single Image

  http://www.cs.nott.ac.uk/~psxasj/3dme/

Demo

- 3D Face Reconstruction from a Single Image

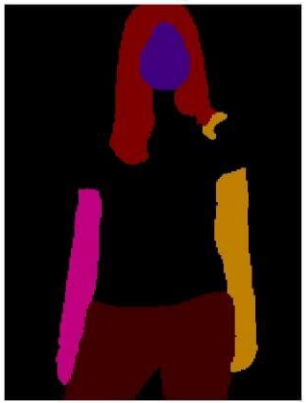
  http://www.cs.nott.ac.uk/~psxasj/3dme/

Virtual Clothing Try On

Virtual Clothing Try On

Orig. person segmentation

Transformed segmentation

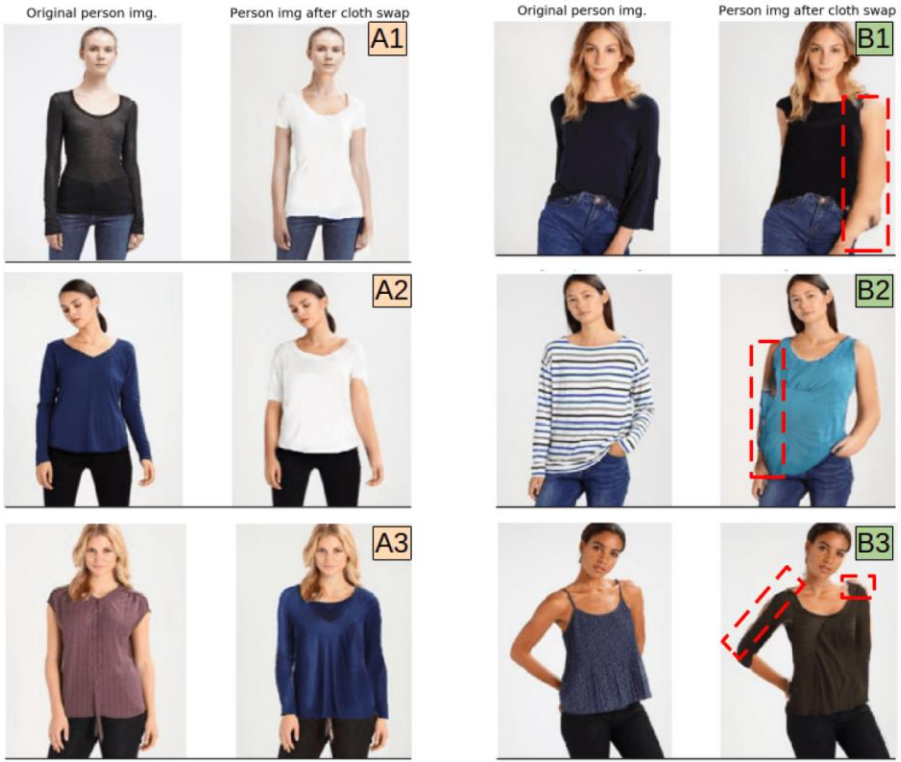


Target cloth img.

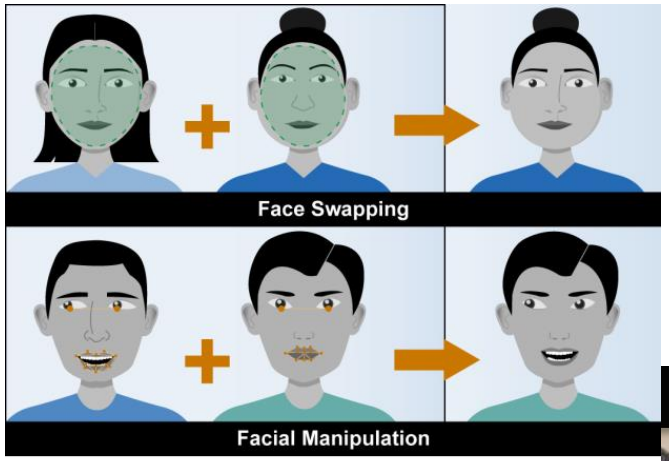
Original person img.

Person img after cloth swap

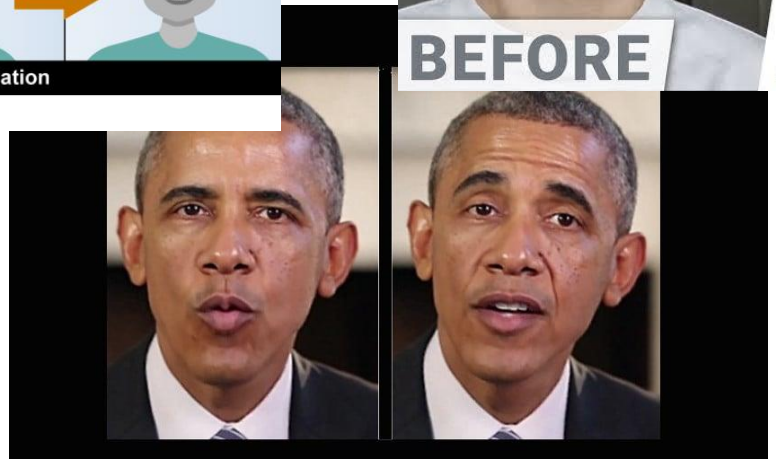
Virtual Clothing Trv On



DeepFake



Source: GAO. | GAO-20-379SP



USC Viterbi School of Engineering

Detecting DeepFake