CS 1674: Convolutional Neural Networks

PhD. Nils Murrugarra-Llerena
nem177@pitt.edu

Exam 1 - Stats

# Students	Max Score	Average
31	95	76.21


To join, go to: ahaslides.com/9LQJB
AhaSlides
What option do you prefer to boost your grades?
Get Feedback
Slide 1 selected for PowerPoint
3
39
3/100

Outline

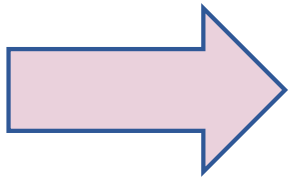
- Neural network Recap
- Convolutional neural networks (CNNs)
  - What is a Convolution?
  - Why CNNs? Special operations: convolution and pooling
- CNN Architectures
  - AlexNet, VGGNet, GoogleNet, Resnet, among others
- Practical matters
  - Tips and tricks for training
  - Transfer learning
  - Software packages

Neural Networks

Slide Credit: Prof. Sandra Avila - UNICAMP

Neural Networks

0	0	3	2
1	1	0	1
4	2	1	2
0	2	1	5

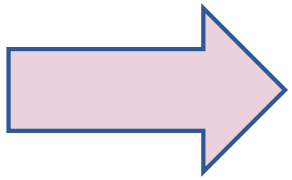


Slide Credit: Prof. Sandra Avila - UNICAMP

Neural Networks

0
0
3
2

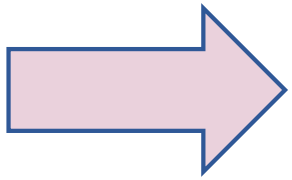
0	0	3	2
1	1	0	1
4	2	1	2
0	2	1	5



Slide Credit: Prof. Sandra Avila - UNICAMP

Neural Networks

0	0	3	2
1	1	0	1
4	2	1	2
0	2	1	5

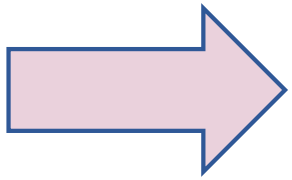


0
0
3
2
1
1
0
1

Slide Credit: Prof. Sandra Avila - UNICAMP

Neural Networks

0	0	3	2
1	1	0	1
4	2	1	2
0	2	1	5



0
0
3
2
1
1
0
1
⋮
2
1
5

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?

What is a Convolution?

Convolution is the process of adding each element of the image to its local neighbors, **weighted by the kernel/filter**.

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 x 5 matrix (image)

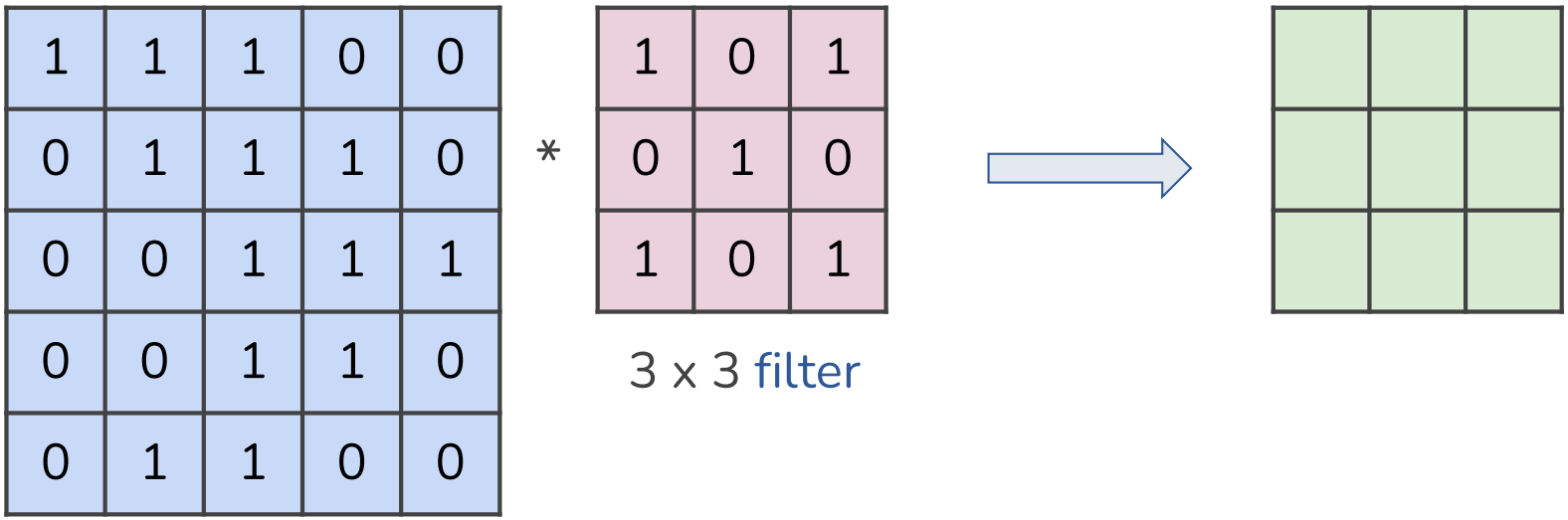
*

1	0	1
0	1	0
1	0	1

3 x 3 filter

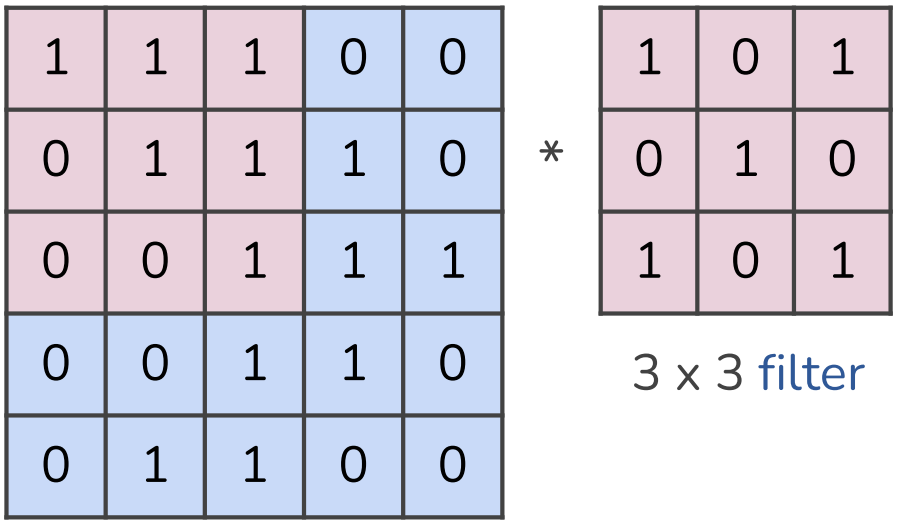
Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



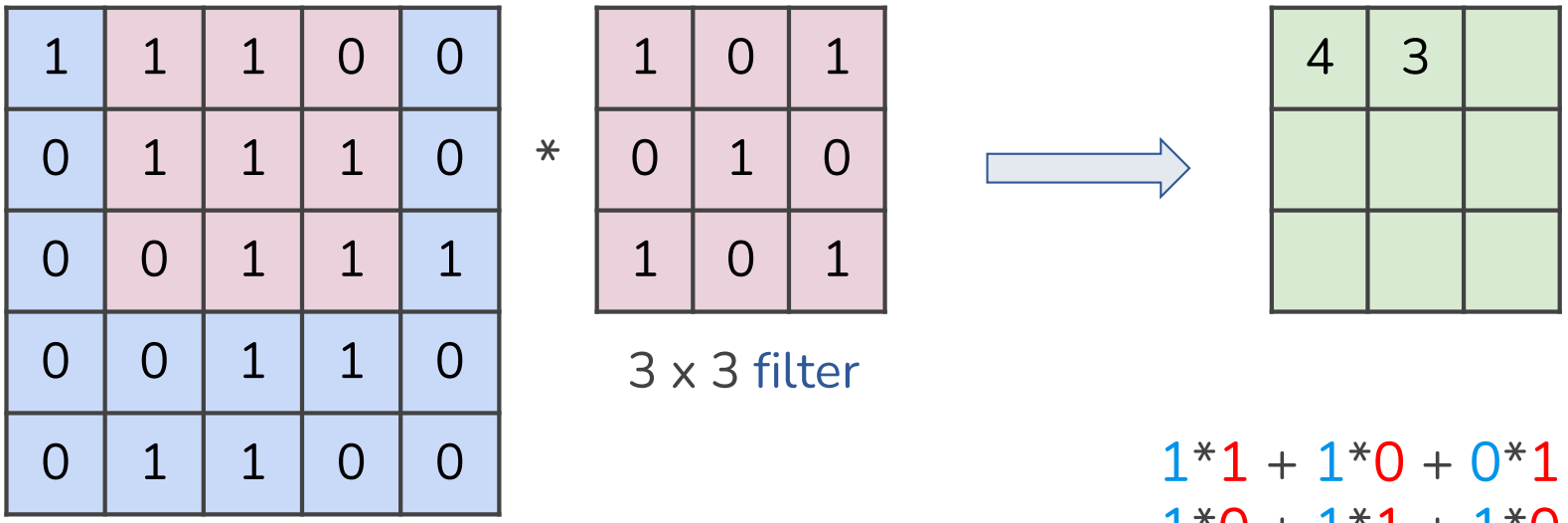
5 x 5 matrix (image)

3 x 3 filter

$1*1 + 1*0 + 1*1 +$
$0*0 + 1*1 + 1*0 +$
$0*1 + 0*0 + 1*1 = 4$

Slide Credit: Prof. Sandra Avila - UNICAMP

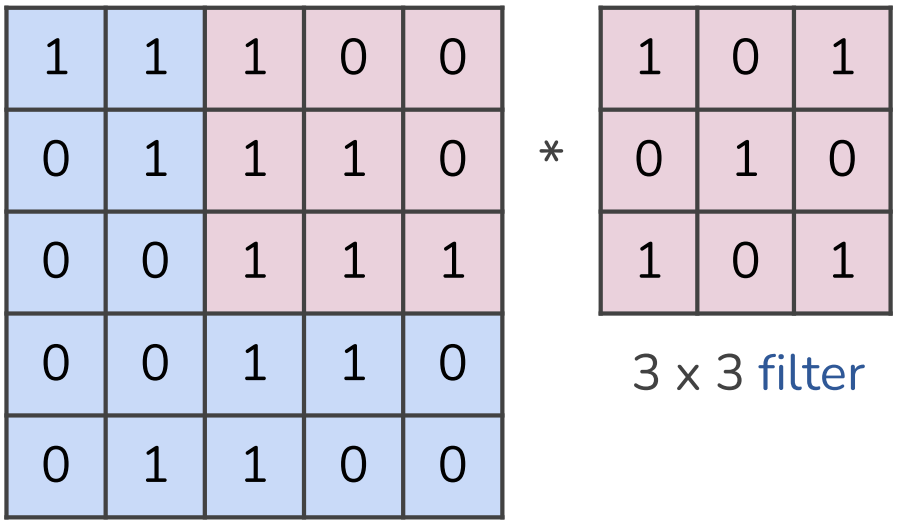
What is a Convolution?



1*1 + 1*0 + 0*1 +
1*0 + 1*1 + 1*0 +
0*1 + 1*0 + 1*1 = 3

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



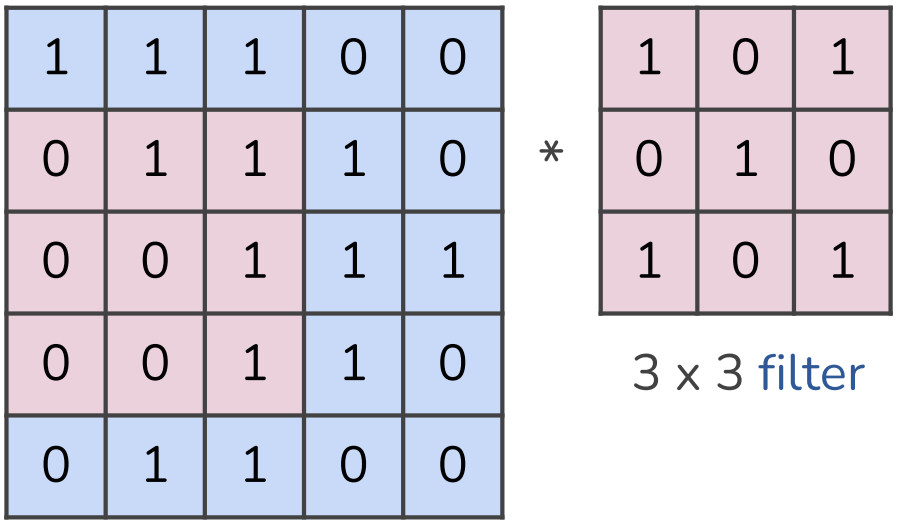
5 x 5 matrix (image)

3 x 3 filter

$1*1 + 0*0 + 0*1 +$
$1*0 + 1*1 + 0*0 +$
$1*1 + 1*0 + 1*1 = 4$

Slide Credit: Prof. Sandra Avila - UNICAMP

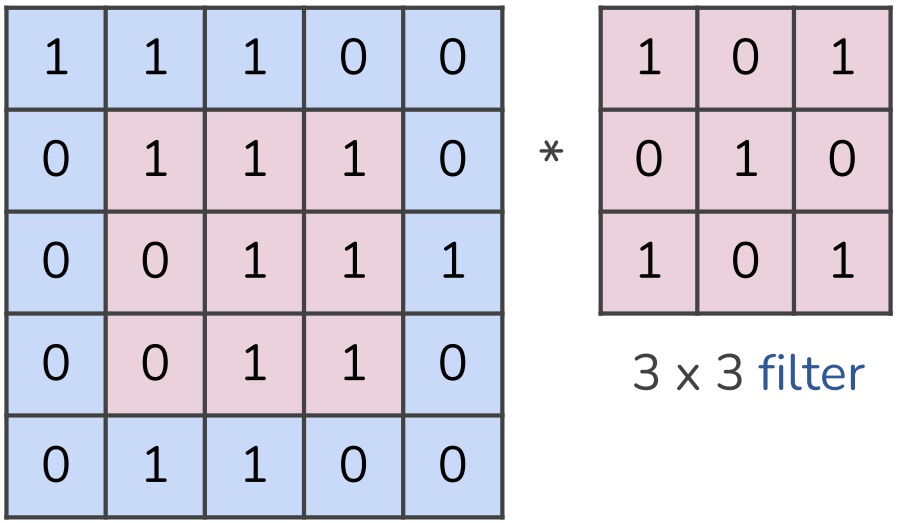
What is a Convolution?



$0*1 + 1*0 + 1*1 +$
$0*0 + 0*1 + 1*0 +$
$0*1 + 0*0 + 1*1 = 2$

Slide Credit: Prof. Sandra Avila - UNICAMP

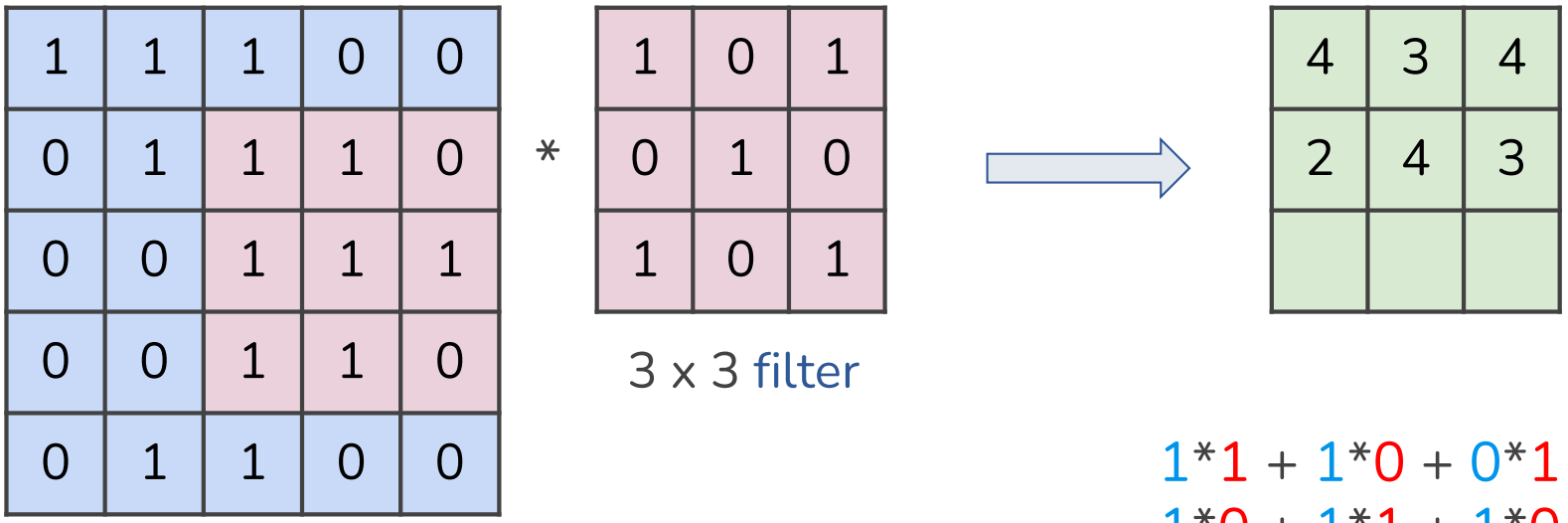
What is a Convolution?



$1*1 + 1*0 + 1*1 +$
$0*0 + 1*1 + 1*0 +$
$0*1 + 1*0 + 1*1 = 4$

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



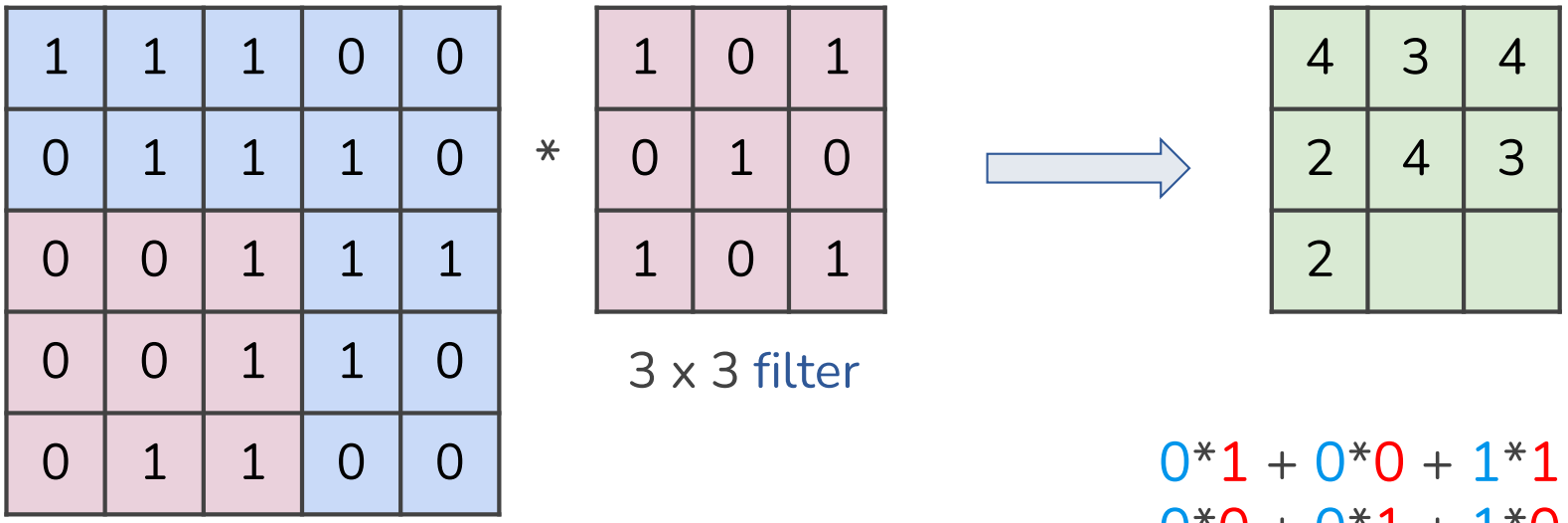
5 x 5 matrix (image)

3 x 3 filter

$1*1 + 1*0 + 0*1 +$
$1*0 + 1*1 + 1*0 +$
$1*1 + 1*0 + 0*1 = 3$

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



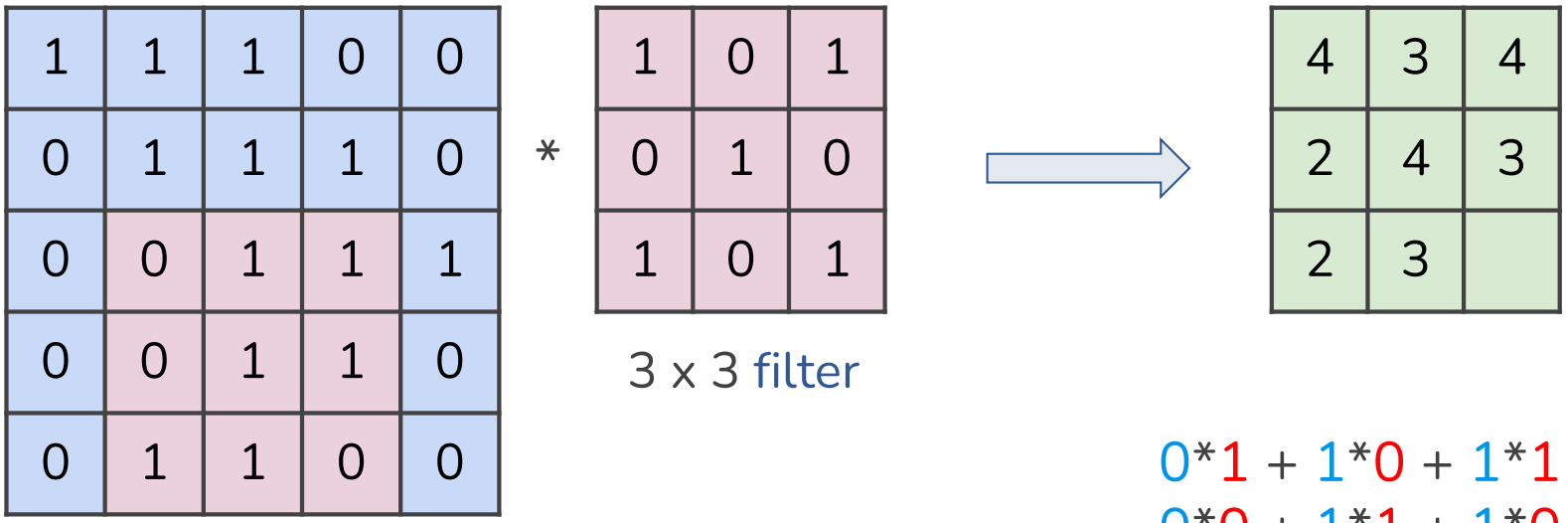
5 x 5 matrix (image)

3 x 3 filter

$0*1 + 0*0 + 1*1 +$
$0*0 + 0*1 + 1*0 +$
$0*1 + 1*0 + 1*1 = 2$

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



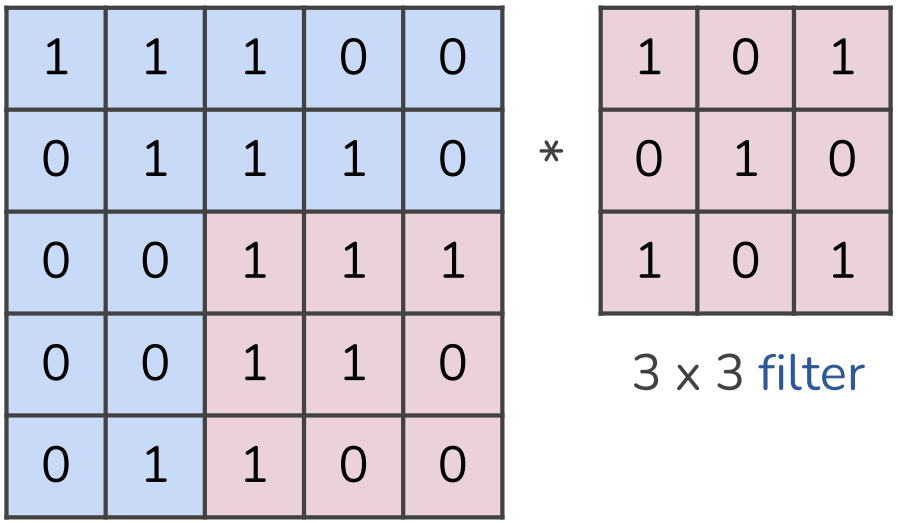
5 x 5 matrix (image)

3 x 3 filter

$0*1 + 1*0 + 1*1 +$
$0*0 + 1*1 + 1*0 +$
$1*1 + 1*0 + 0*1 = 3$

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?



5 x 5 matrix (image)

3 x 3 filter

$1*1 + 1*0 + 1*1 +$
$1*0 + 1*1 + 0*0 +$
$1*1 + 0*0 + 0*1 = 4$

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?

*

0	1	0
1	-4	1
0	1	0

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?

*

Edge Detection

0	1	0
1	-4	1
0	1	0

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?

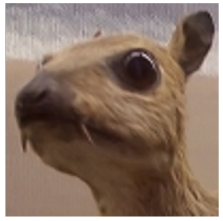
*

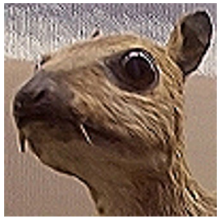
Emboss

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

Slide Credit: Prof. Sandra Avila - UNICAMP

What is a Convolution?





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

Edge Detection

0	-1	0
-1	5	-1
0	-1	0

Sharpen

1/9

1	1	1
1	1	1
1	1	1

Box blur

1/16

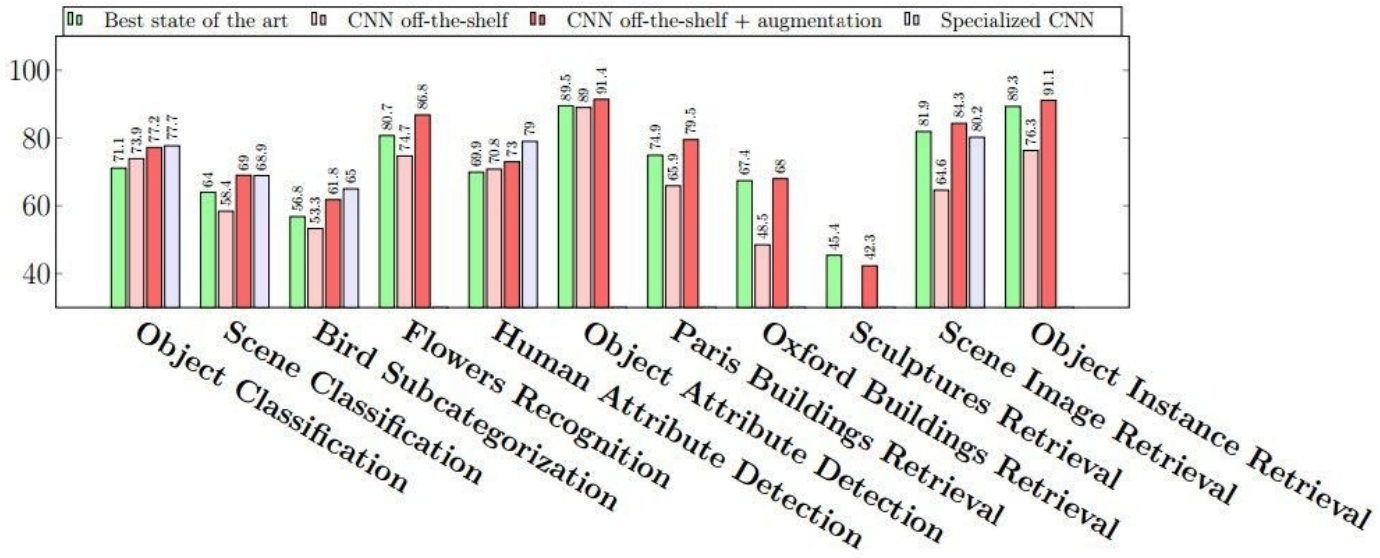
1	2	1
2	4	2
1	2	1

Gaussian blur 3 × 3

Slide Credit: Prof. Sandra Avila - UNICAMP

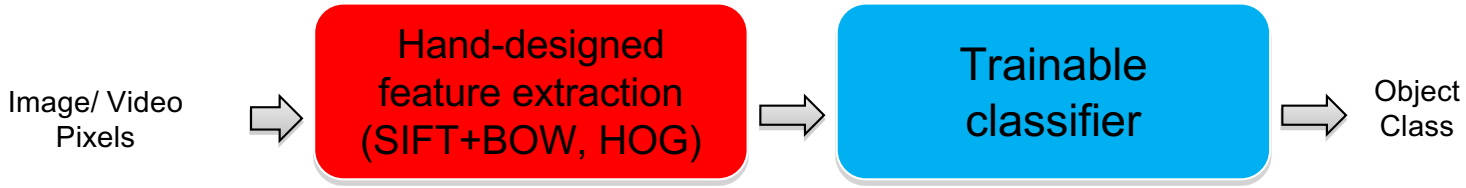
Why Convolutional Neural networks?

- State of the art performance on many problems
- Most papers in recent vision conferences use deep neural networks



Razavian et al., CVPR 2014 Workshops *(4000+ citations)*

Traditional Recognition Approach



- Features are key to recent progress in recognition, but research shows they're flawed… Where next?

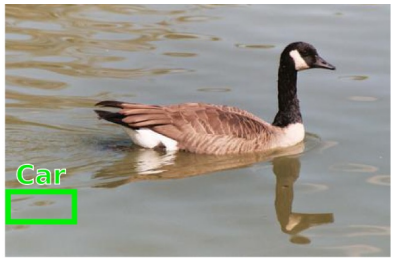


Fig. 1: An image from PASCAL and a high scoring car detection from DPM (Felzenszwalb et al, 2010b). Why did the detector fail?

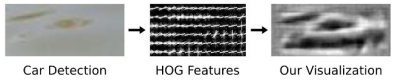
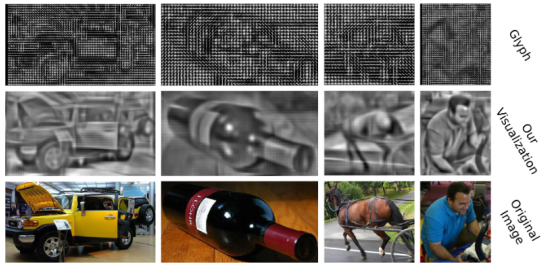
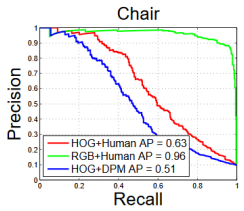


Fig. 2: We show the crop for the false car detection from Figure 1. On the right, we show our visualization of the HOG features for the same patch. Our visualization reveals that this false alarm actually looks like a car in HOG space.





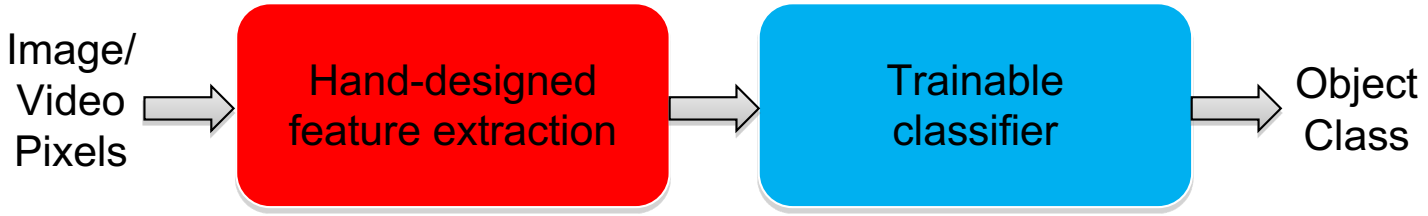
Adapted from Lana Lazebnik, figures from Vondrick: http://www.cs.columbia.edu/~vondrick/ihog/ijcv.pdf

What about learning the features?

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

“Shallow” vs. “deep” architectures

Traditional recognition: “Shallow” architecture

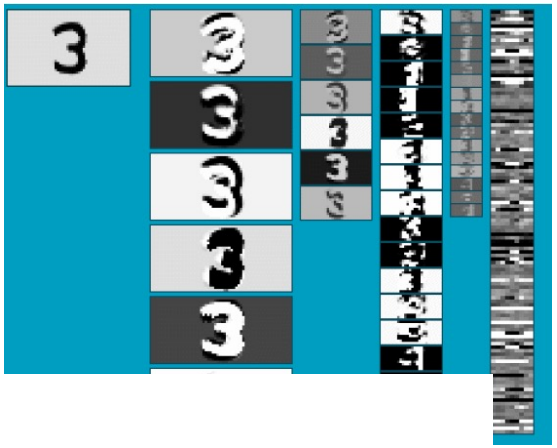


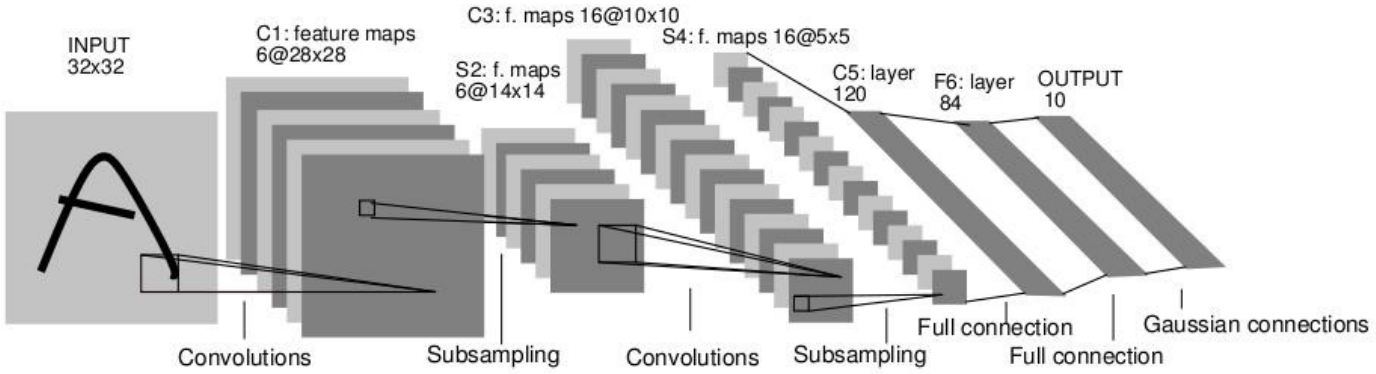
Deep learning: “Deep” architecture

Convolutional Neural Networks

Convolutional Neural Networks (CNN)

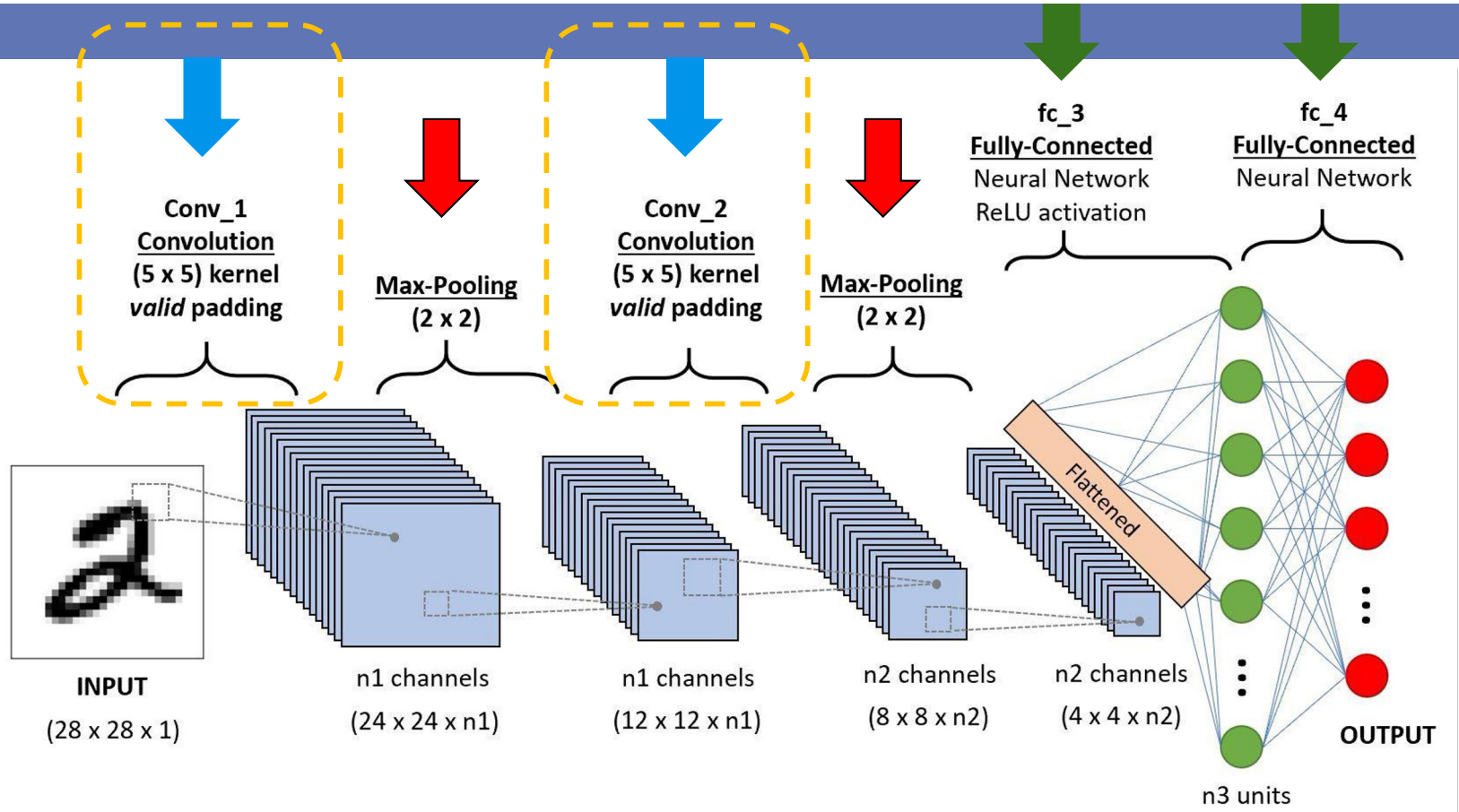
- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant, *more abstract* features
- Classification layer at the end





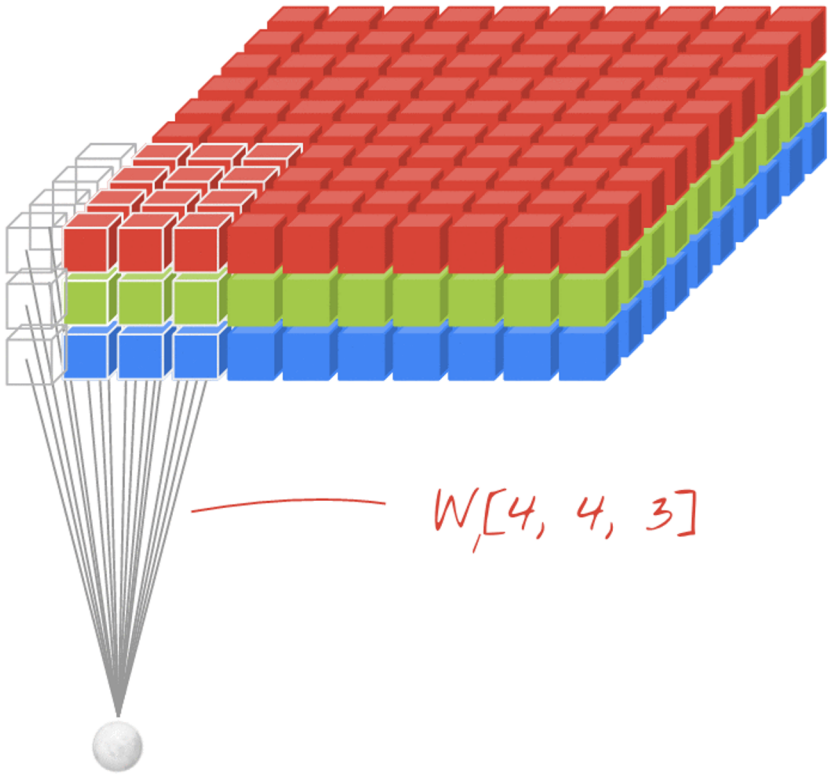
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86(11): 2278–2324, 1998.

Adapted from Rob Fergus



There are a few distinct types of layers (e.g., **CONV**/**POOL**/**FC** are by far the most popular).

Slide Credit: Prof. Sandra Avila - UNICAMP

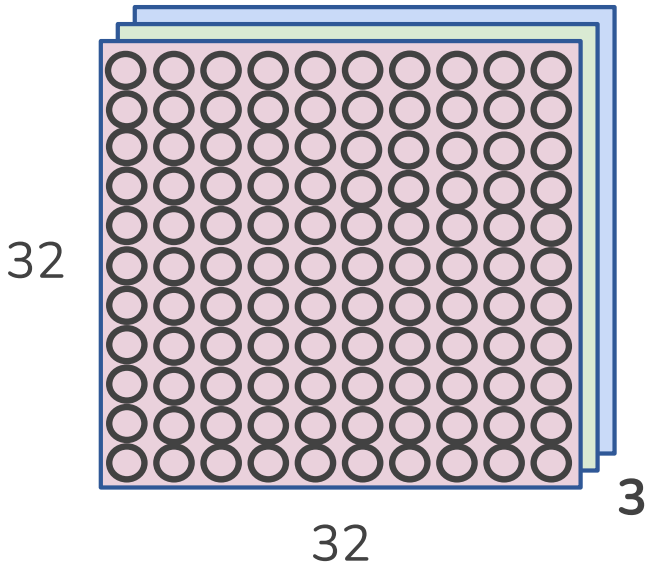


Credit: Martin Görner, @martin_gorner (twitter)

Slide Credit: Prof. Sandra Avila - UNICAMP

Convolution Layer

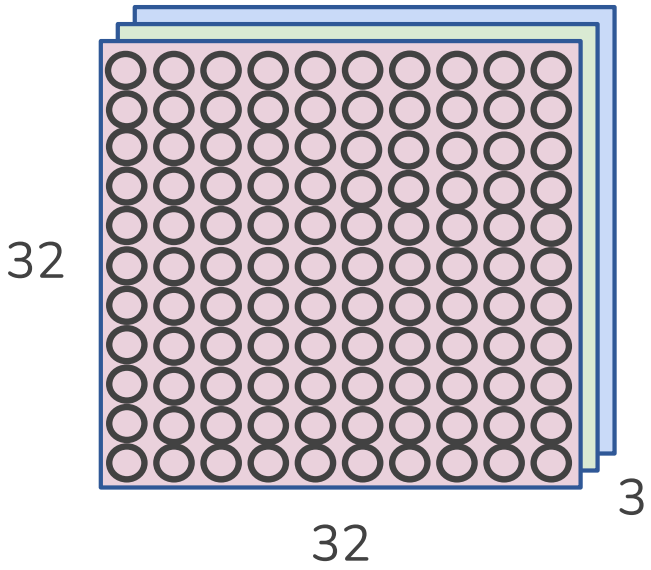
32 x 32 x 3 image ⇒ preserve spatial structure



Slide Credit: Prof. Sandra Avila - UNICAMP

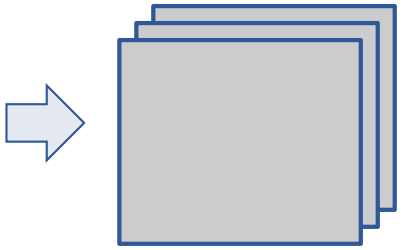
Convolution Layer

32 x 32 x 3 image ⇒ preserve spatial structure



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

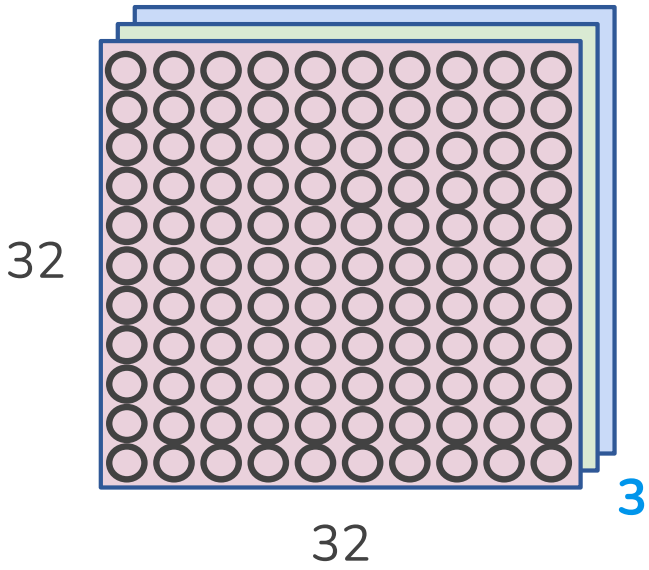
5 x 5 x 3 filter



Slide Credit: Prof. Sandra Avila - UNICAMP

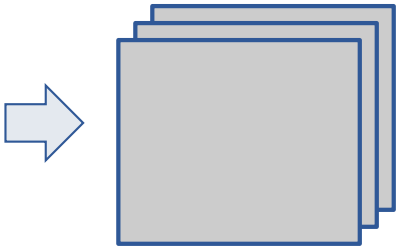
Convolution Layer

32 x 32 x 3 image ⇒ preserve spatial structure



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

5 x 5 x **3** filter

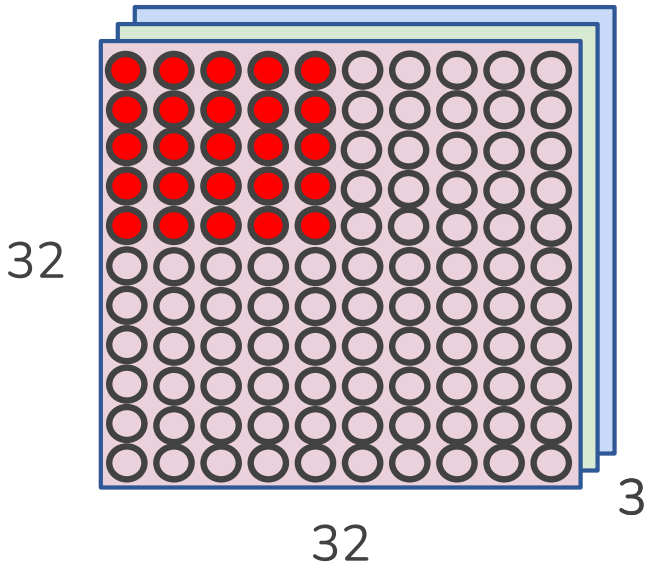


Filters always extend the full depth of the input volume

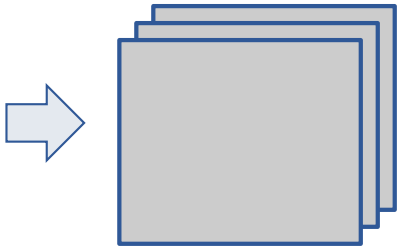
Slide Credit: Prof. Sandra Avila - UNICAMP

Convolution Layer

32 x 32 x 3 image ⇒ preserve spatial structure



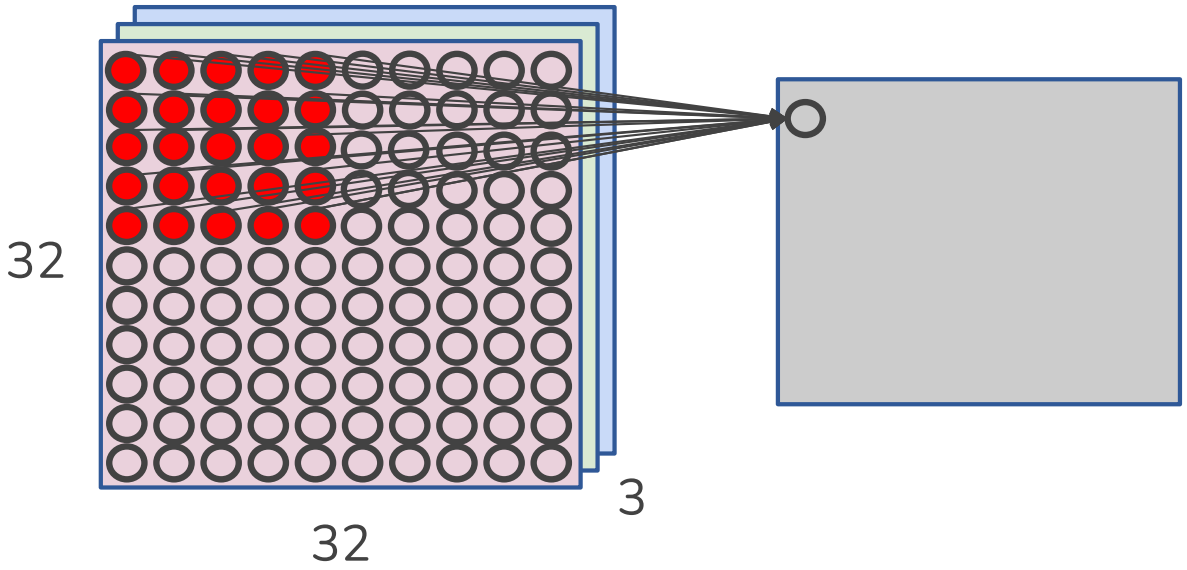
5 x 5 x 3 filter



Slide Credit: Prof. Sandra Avila - UNICAMP

Convolution Layer

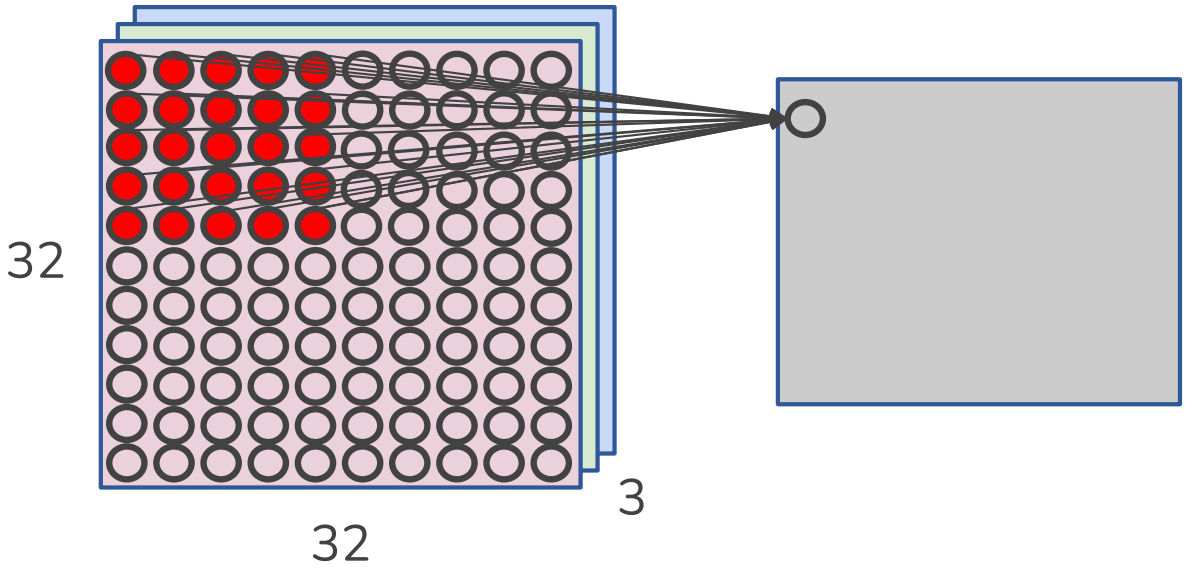
32 x 32 x 3 image ⇒ preserve spatial structure



Slide Credit: Prof. Sandra Avila - UNICAMP

Convolution Layer

32 x 32 x 3 image ⇒ preserve spatial structure



1 number:

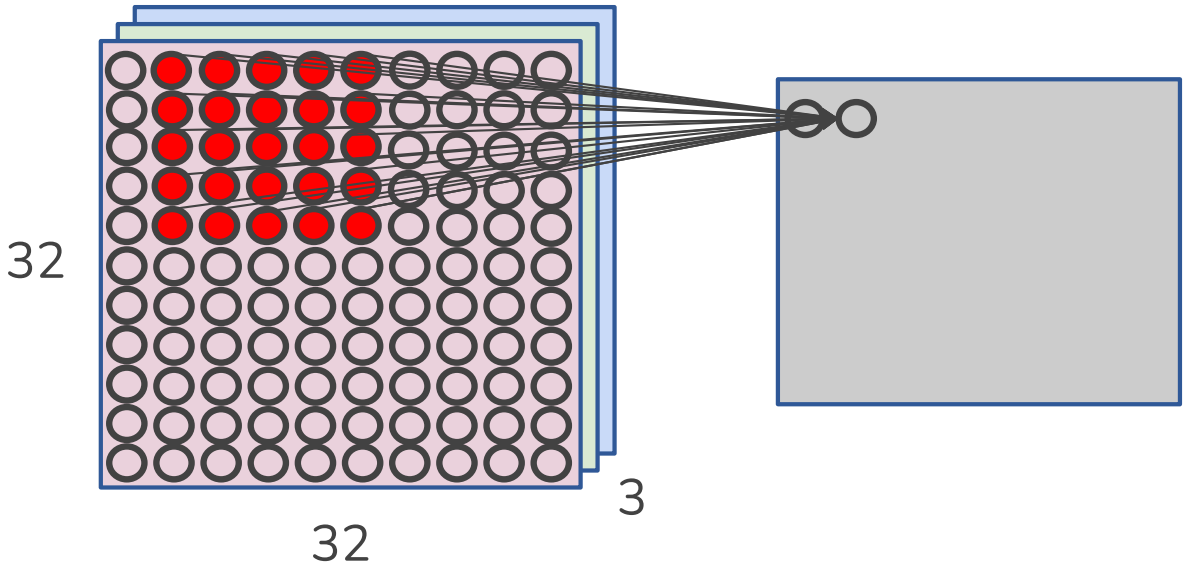
5*5*3 = 75-dimensional dot product + bias)

$w^{\mathrm{T}}x + b$

Slide Credit: Prof. Sandra Avila - UNICAMP

Convolution Layer

32 x 32 x 3 image ⇒ preserve spatial structure



1 number:

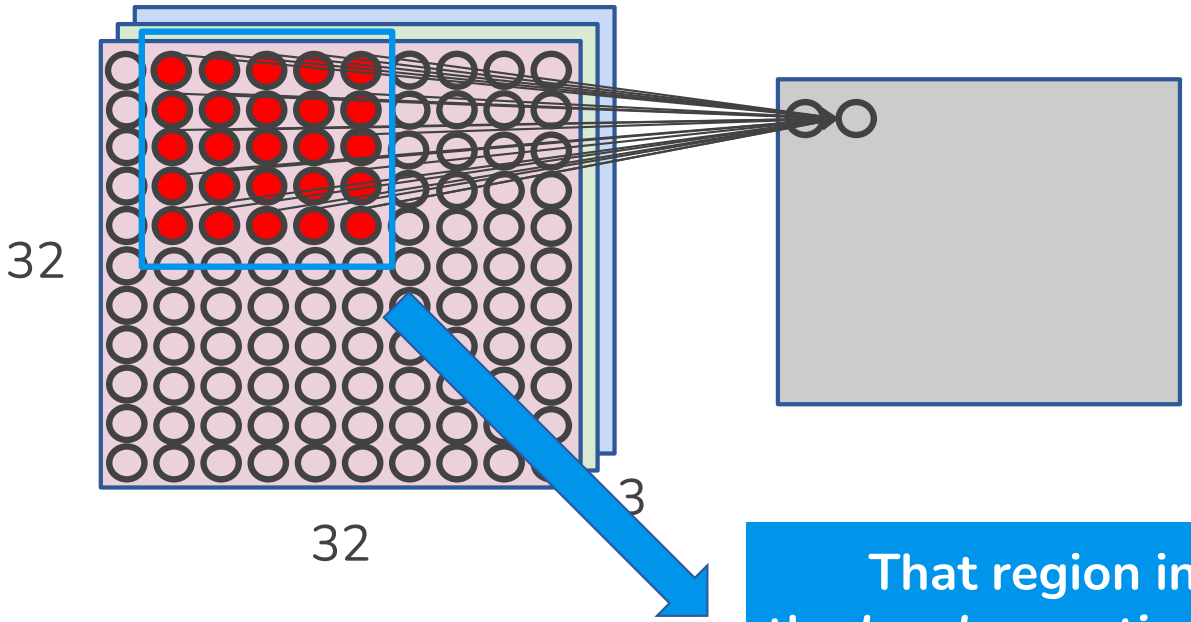
5*5*3 = 75-dimensional dot product + bias)

$w^{\mathrm{T}}x + b$

Slide Credit: Prof. Sandra Avila - UNICAMP

Convolution Layer

32 x 32 x 3 image ⇒ preserve spatial structure



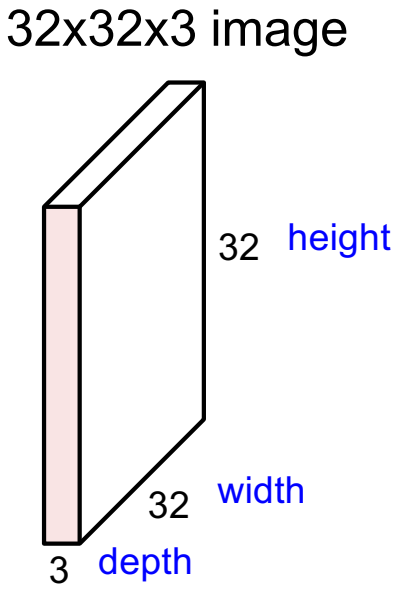
1 number:

5*5*3 = 75-dimensional dot product + bias)

$w^{\mathrm{T}}x + b$

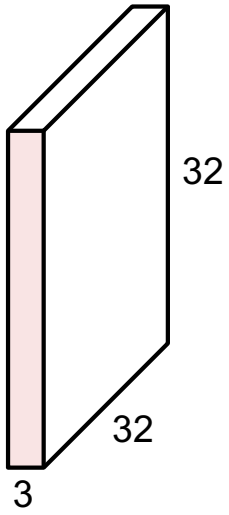
That region in the input image is called the *local receptive field* for the hidden neuron.

Convolutions: More detail

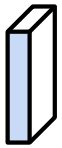


Convolutions: More detail

32x32x3 image



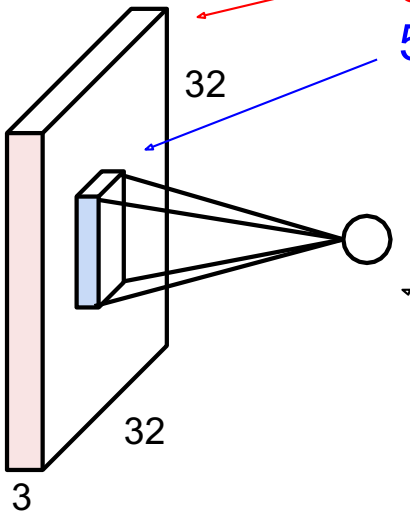
5x5x3 filter



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

Convolutions: More detail

Convolution Layer

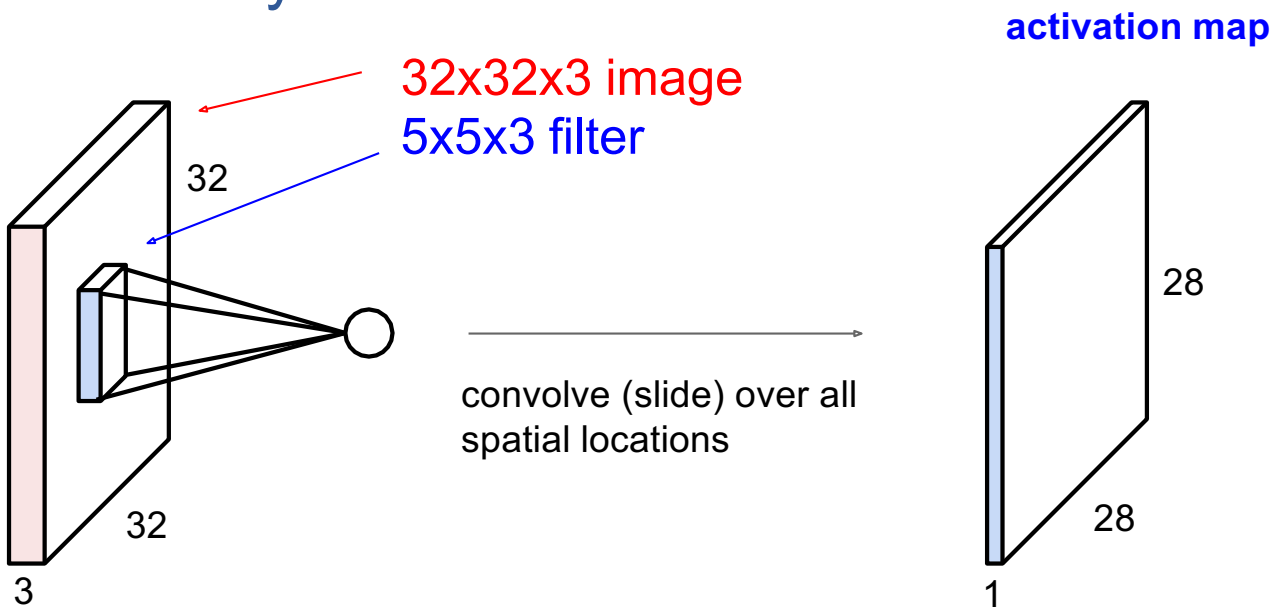


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolutions: More detail

Convolution Layer



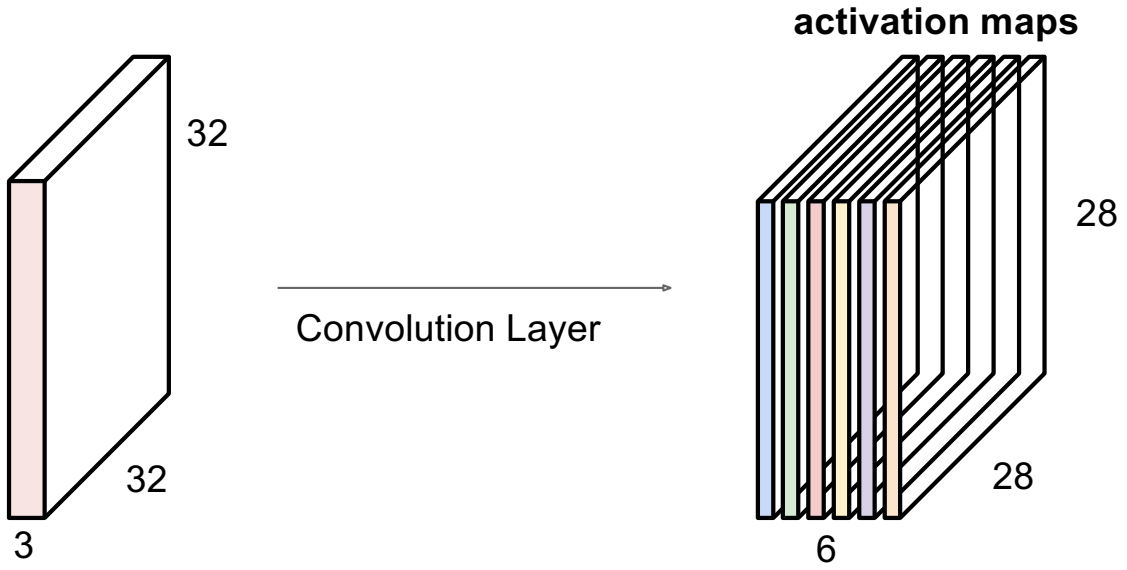
Convolutions: More detail

Convolution Layer

consider a second, green filter

Convolutions: More detail

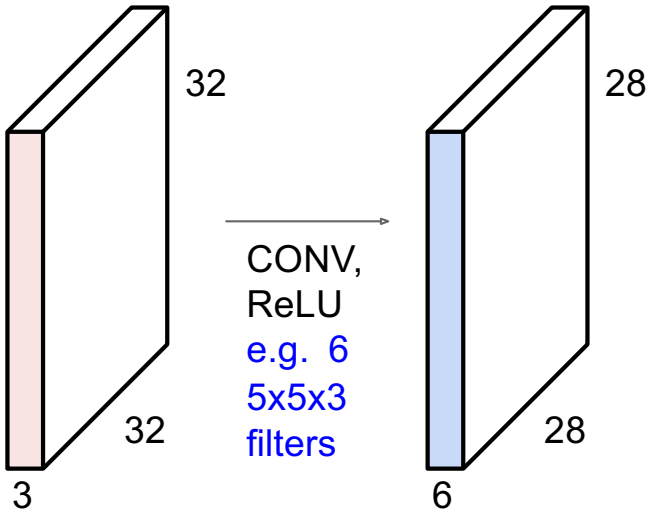
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolutions: More detail

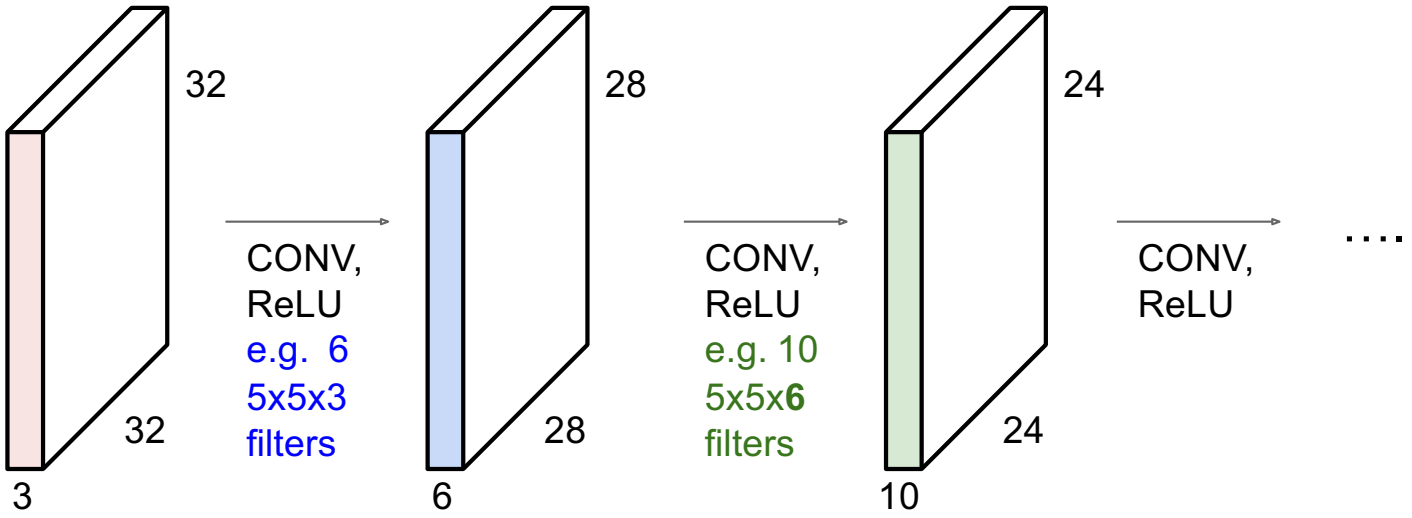
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Andrej Karpathy

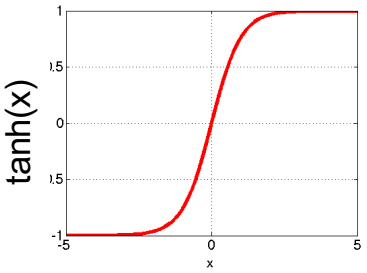
Convolutions: More detail

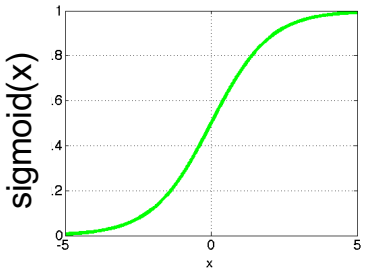
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

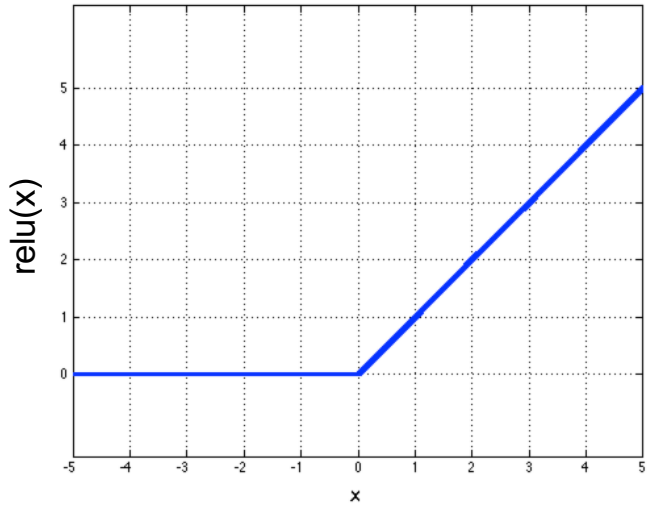


Convolutions: Non Linearity

- Per-element (independent)
- Some options:
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - Rectified linear unit (ReLU)
    - Avoids saturation issues

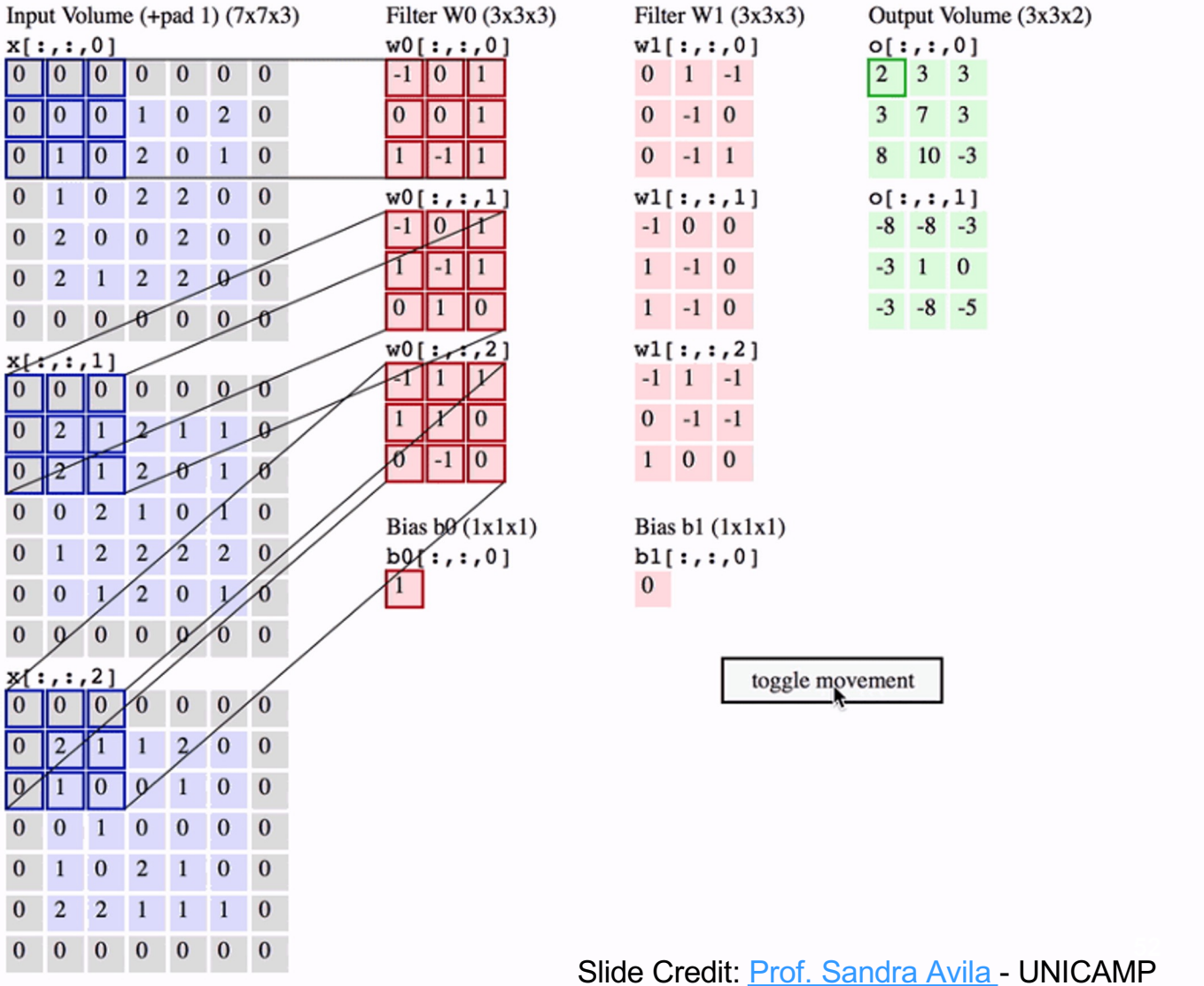






Adapted from Rob Fergus

http://cs231n.github.io/convolutional-networks

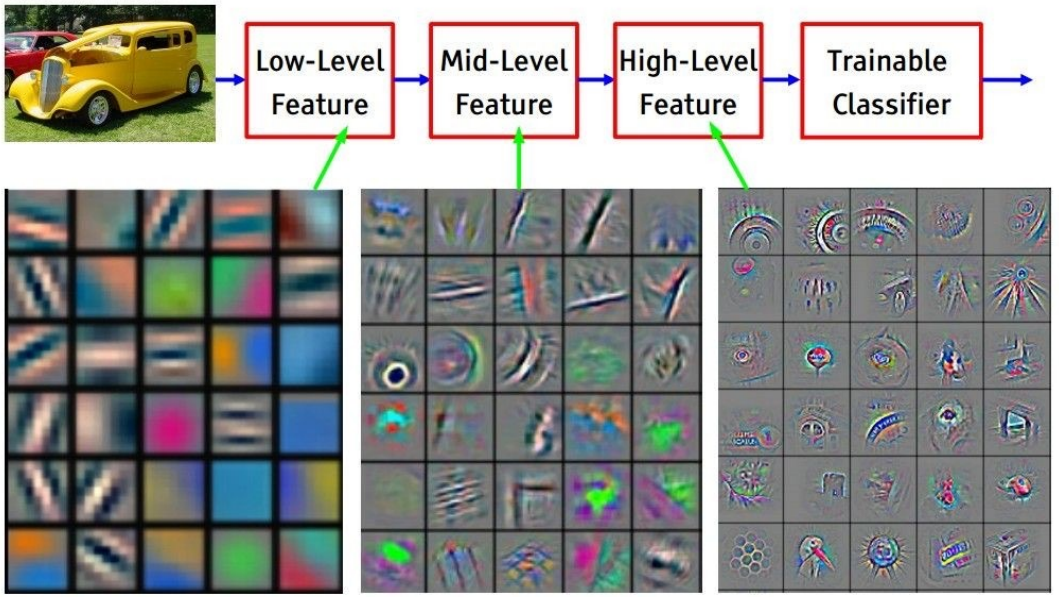


Slide Credit: Prof. Sandra Avila - UNICAMP

Convolutions: More detail

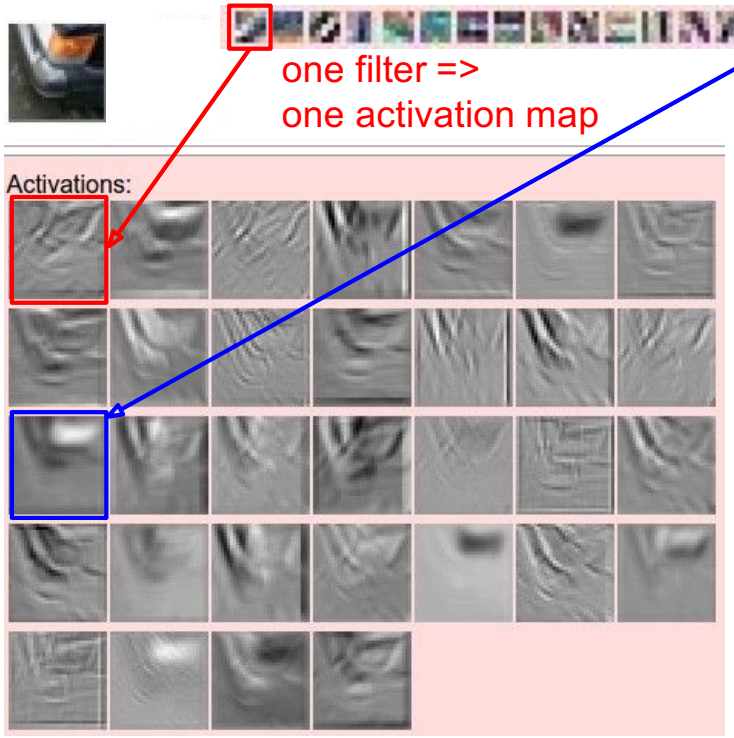
Preview

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutions: More detail



example 5x5 filters
(32 total)

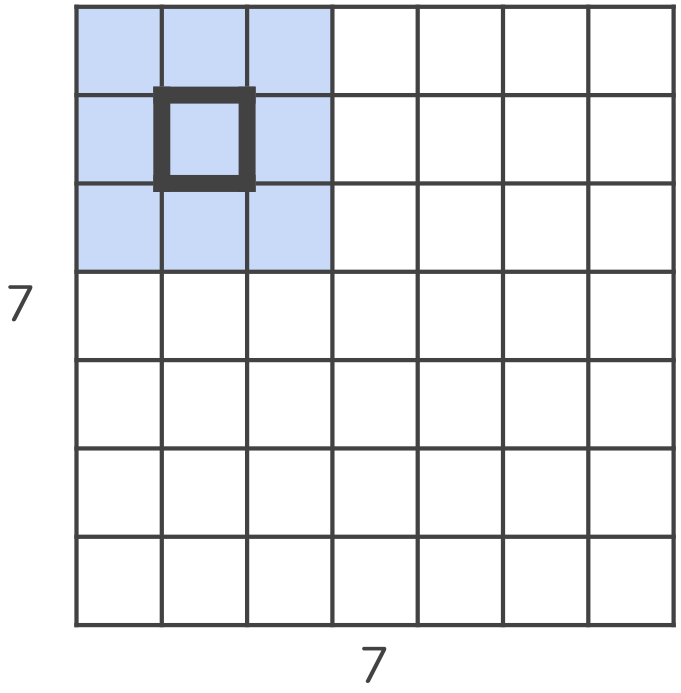
We call the layer convolutional because it is related to convolution of two signals:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

Element-wise multiplication and sum of a filter and the signal (image)

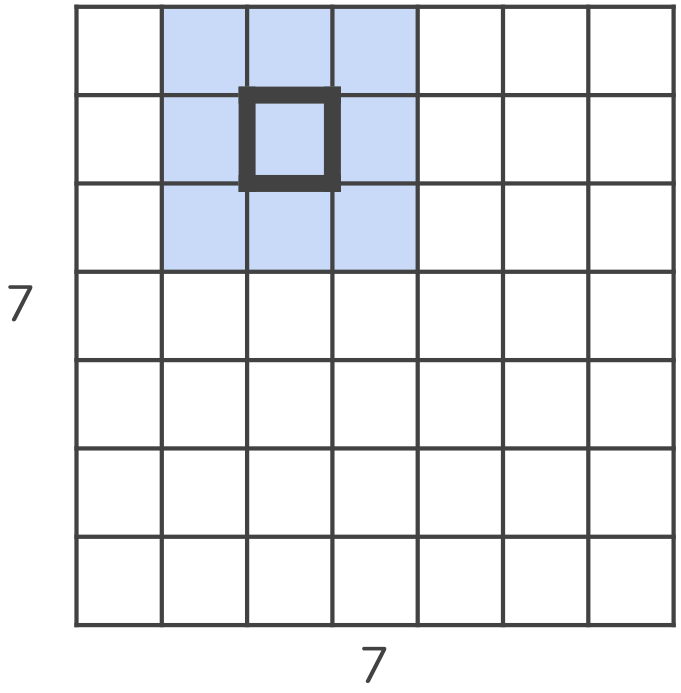
Adapted from Andrej Karpathy, Kristen Grauman

A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

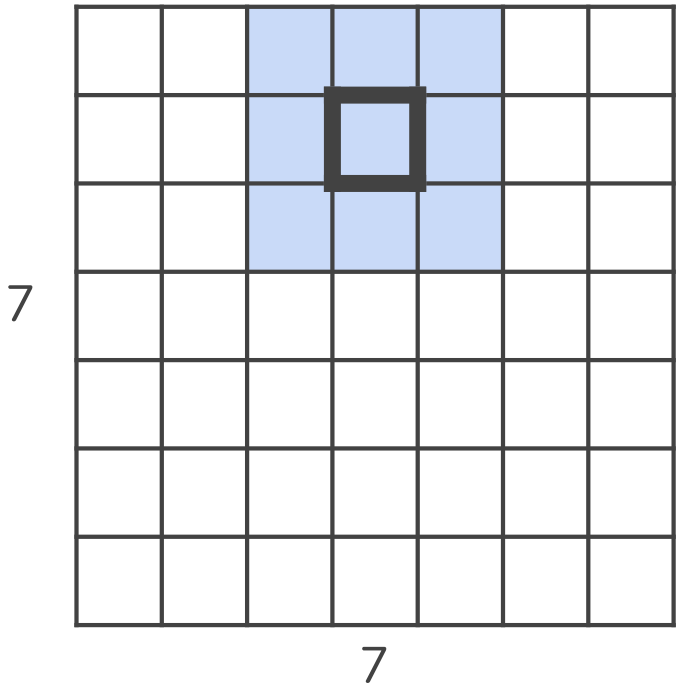
A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

Andrej Karpathy

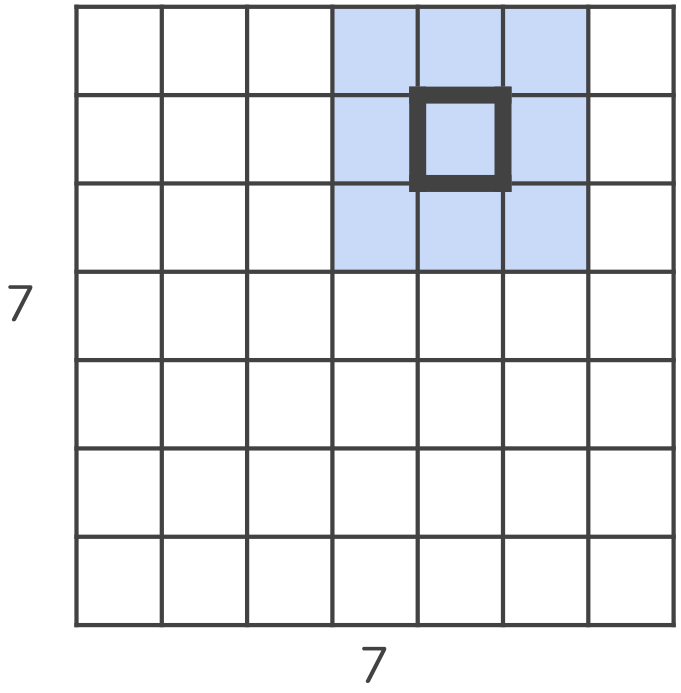
A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

Andrej Karpathy

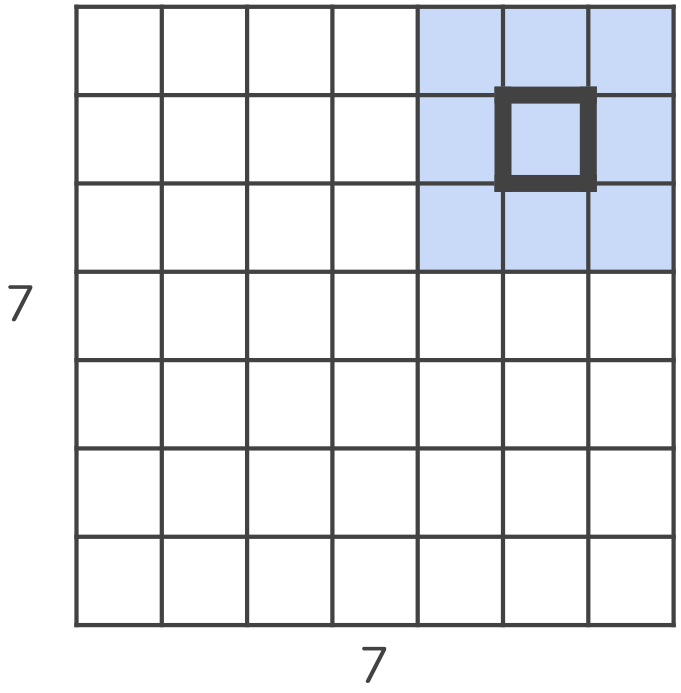
A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

Andrej Karpathy

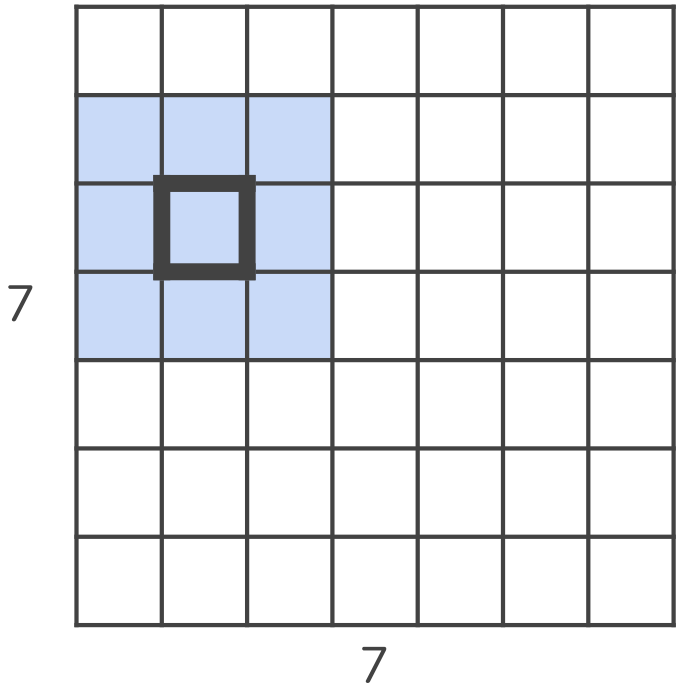
A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

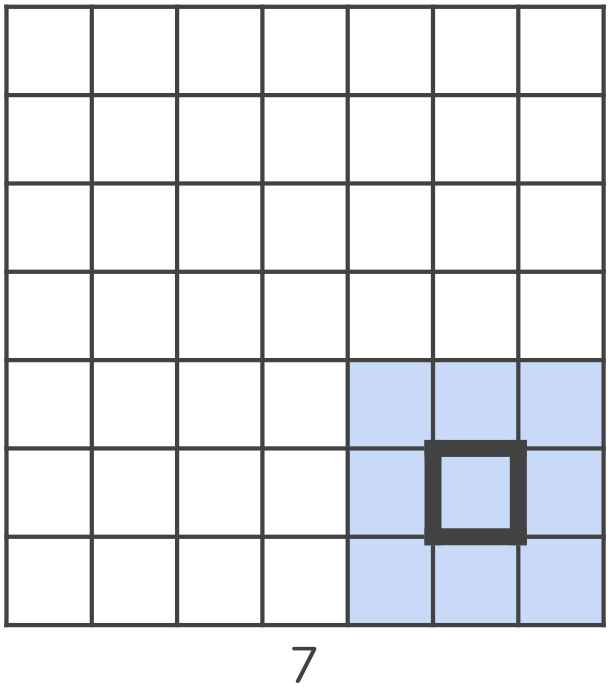
Andrej Karpathy

A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

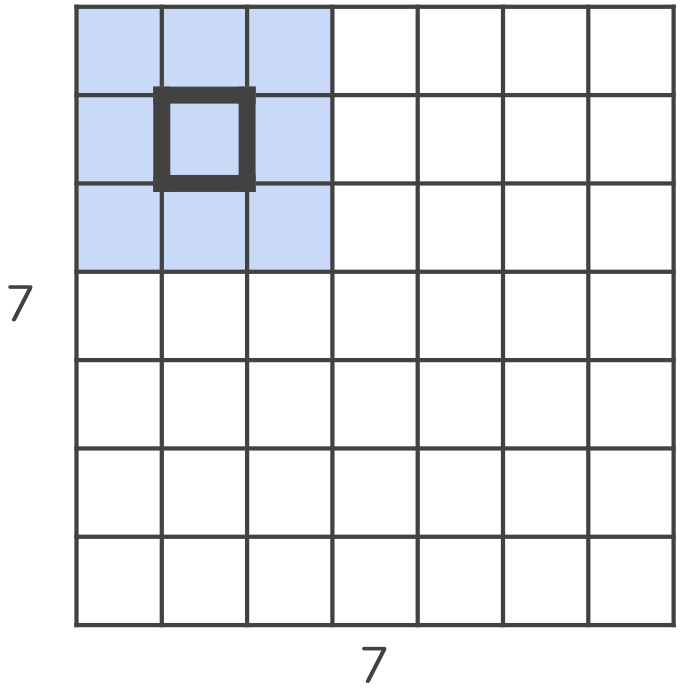
A Closer Look at Spatial Dimensions



7 x 7 input (spatially)
assume 3 x 3 filter

⇒ **5 x 5 output**

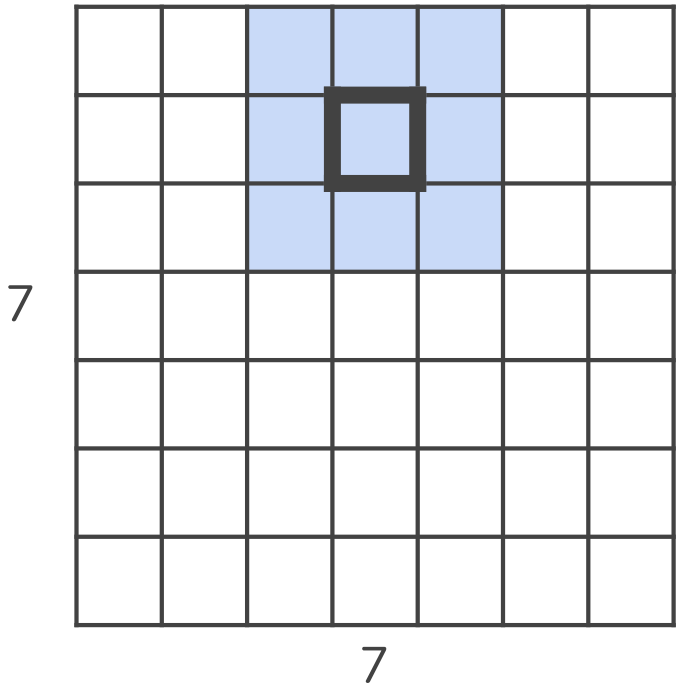
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

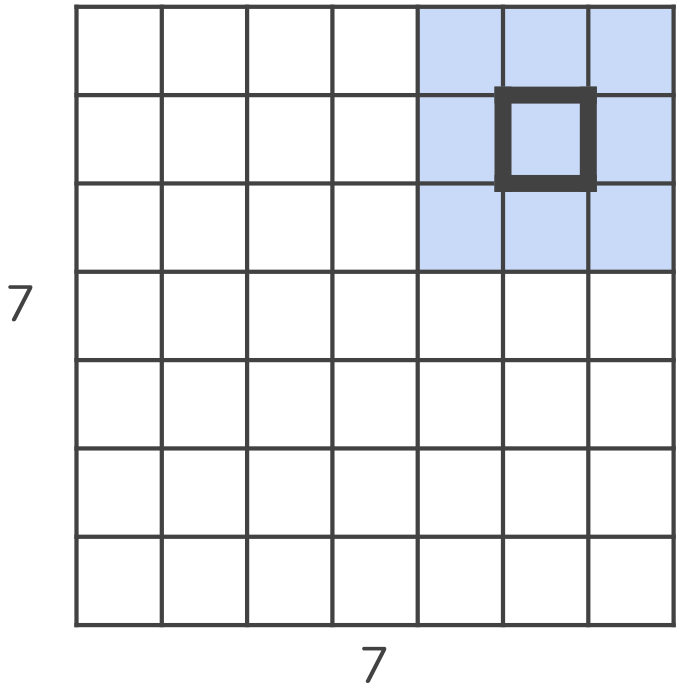
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

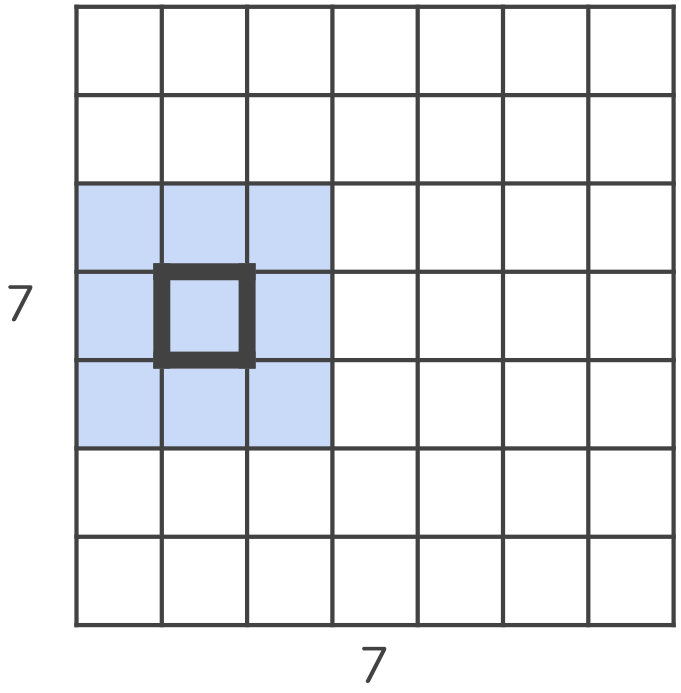
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

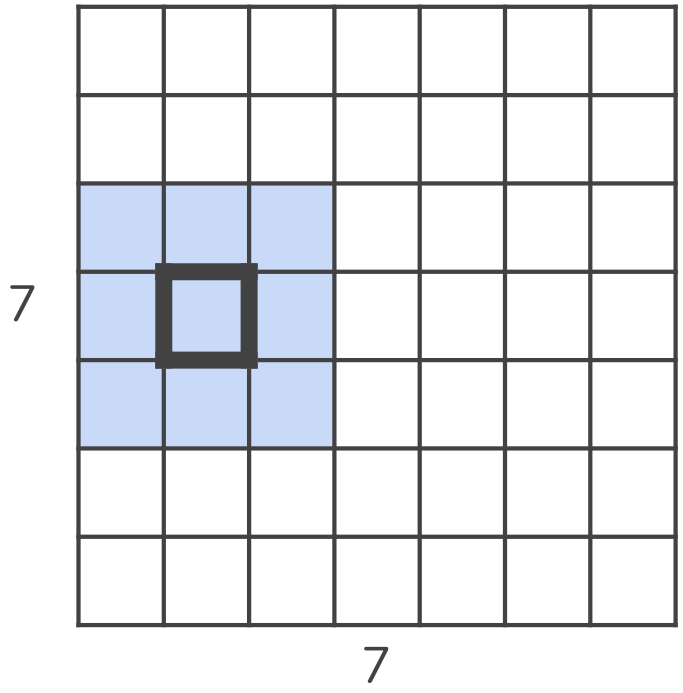
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

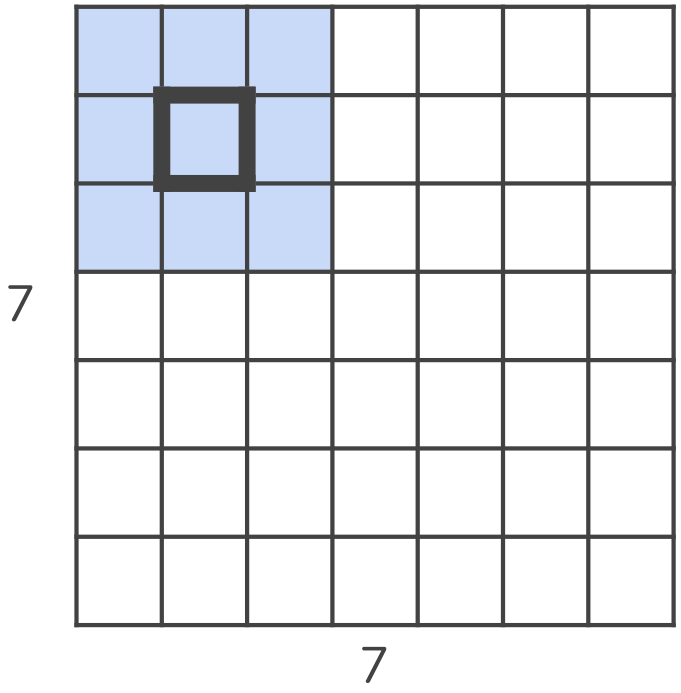
A Closer Look at Spatial Dimensions



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

⇒ **3 x 3 output**

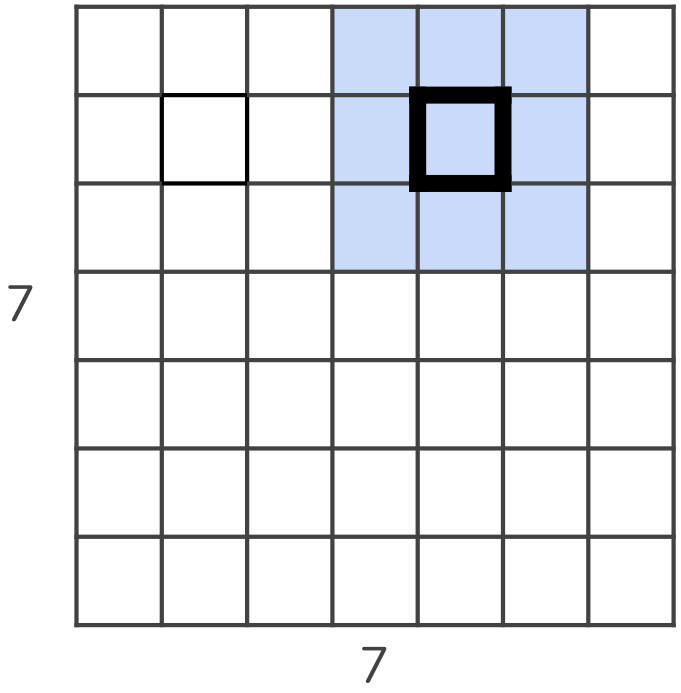
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

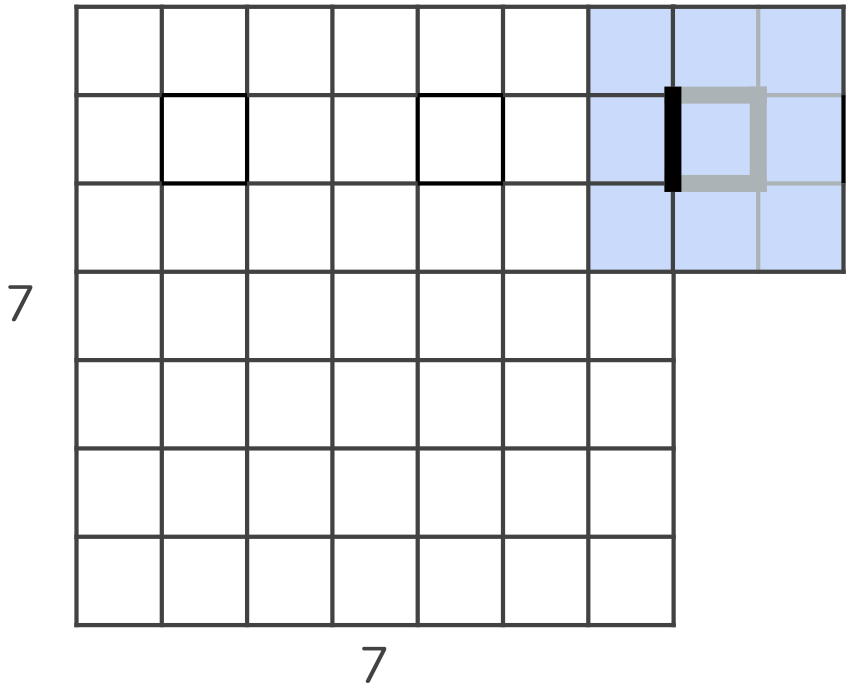
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

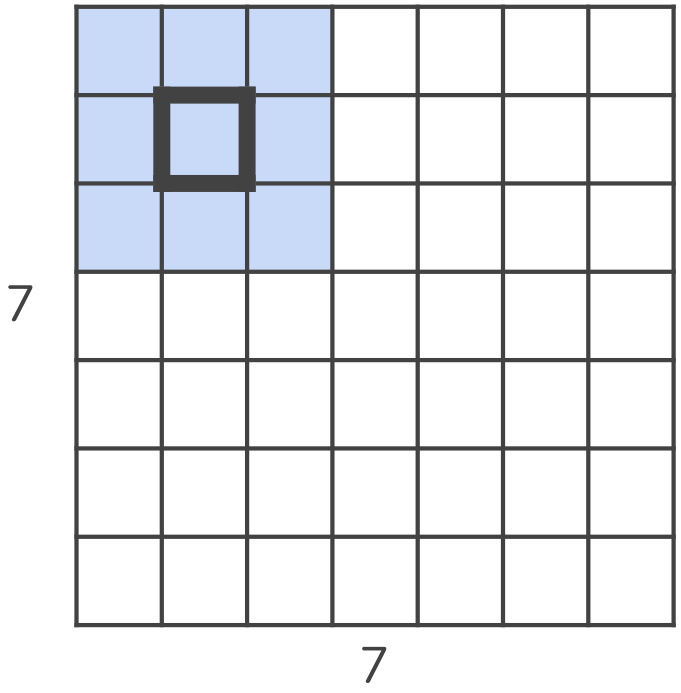
A Closer Look at Spatial Dimensions



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

Andrej Karpathy

A Closer Look at Spatial Dimensions

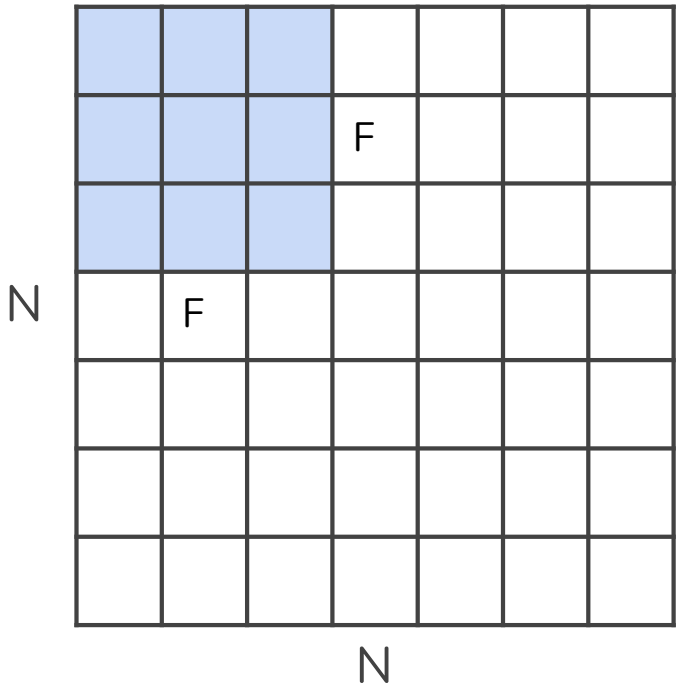


7 x 7 input (spatially)
assume 3 x 3 filter
applied with **stride 3?**

Doesn't fit!
cannot apply 3 x 3 filter on 7 x 7 input with stride 3.

Andrej Karpathy

A Closer Look at Spatial Dimensions

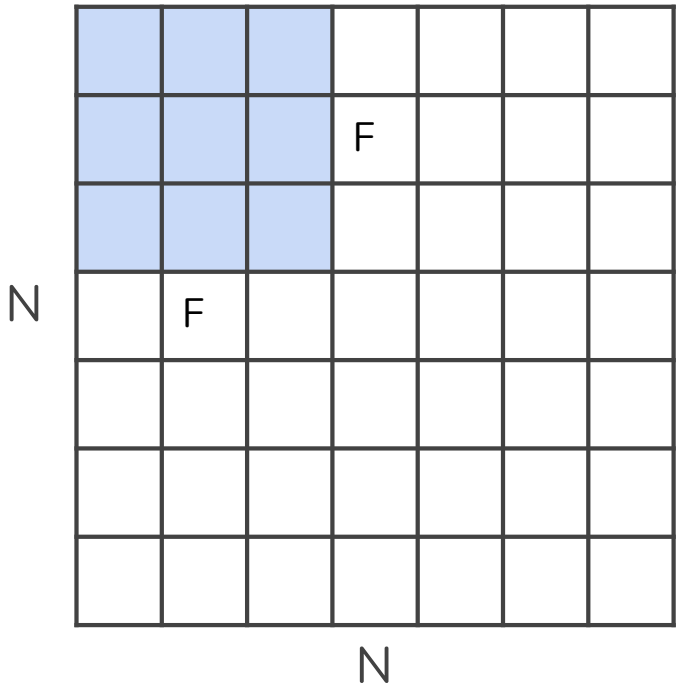


Andrej Karpathy

Output size:

(N - F) / stride + 1

A Closer Look at Spatial Dimensions



Andrej Karpathy

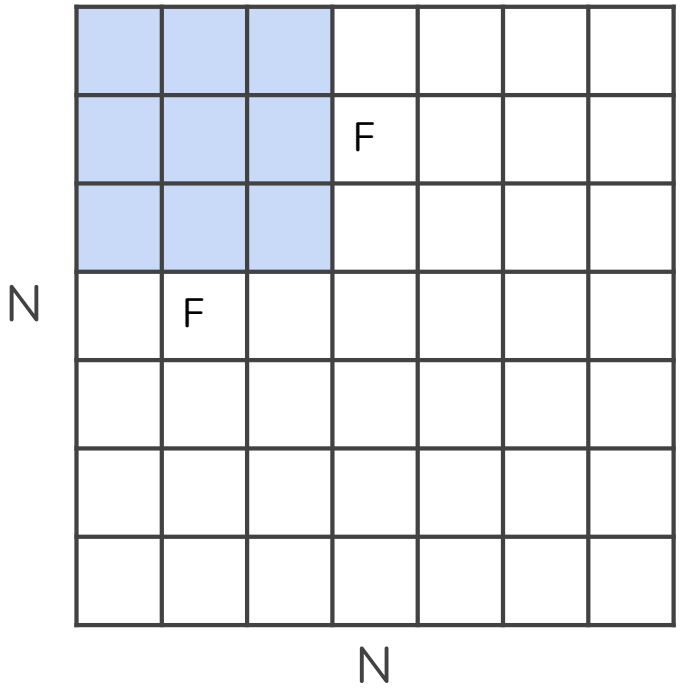
Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 ⇒ (7 - 3)/1 + 1 = 5

A Closer Look at Spatial Dimensions



Andrej Karpathy

Output size:

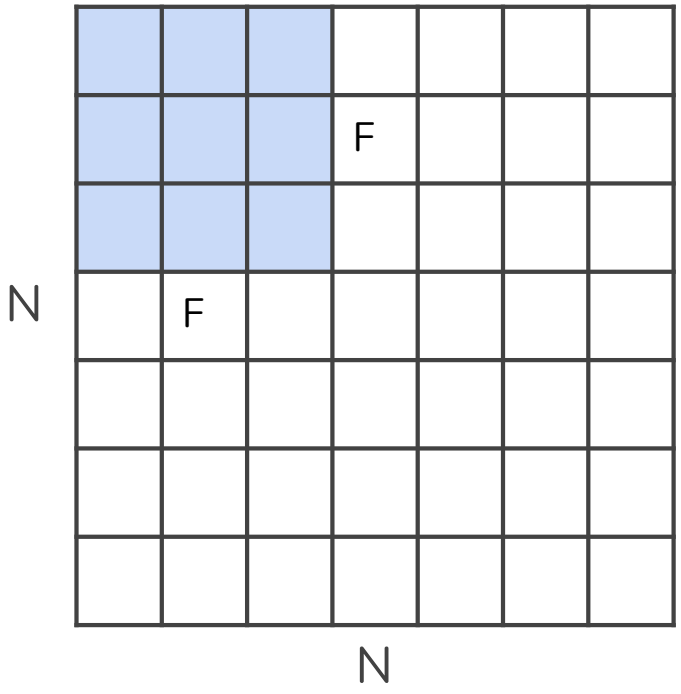
(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 ⇒ (7 - 3)/1 + 1 = 5

stride 2 ⇒ (7 - 3)/2 + 1 = 3

A Closer Look at Spatial Dimensions



Andrej Karpathy

Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 ⇒ (7 - 3)/1 + 1 = 5

stride 2 ⇒ (7 - 3)/2 + 1 = 3

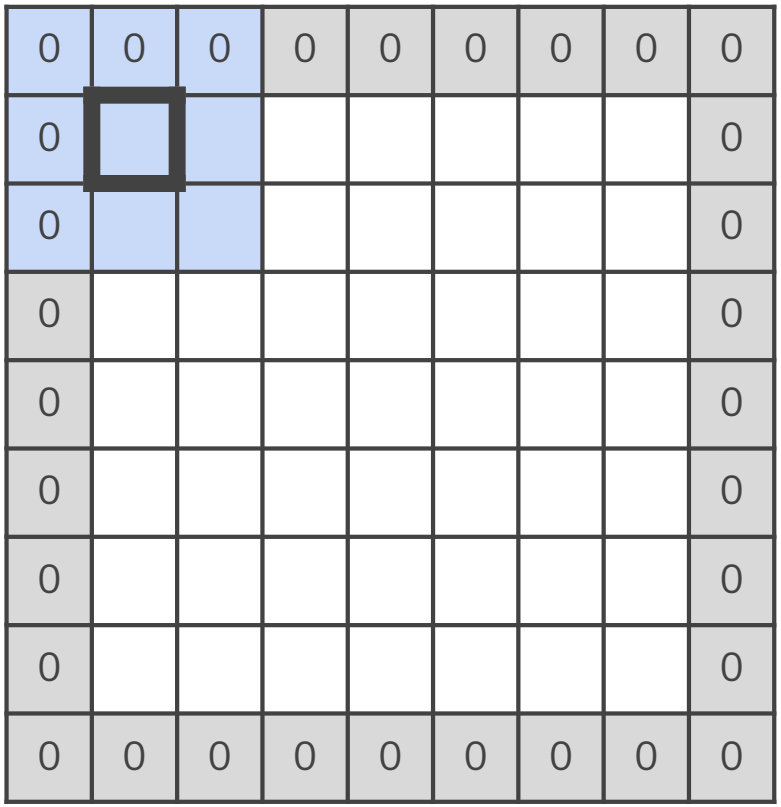
stride 3 ⇒ (7 - 3)/3 + 1 = 2.33

In Practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Andrej Karpathy

In Practice: Common to zero pad the border

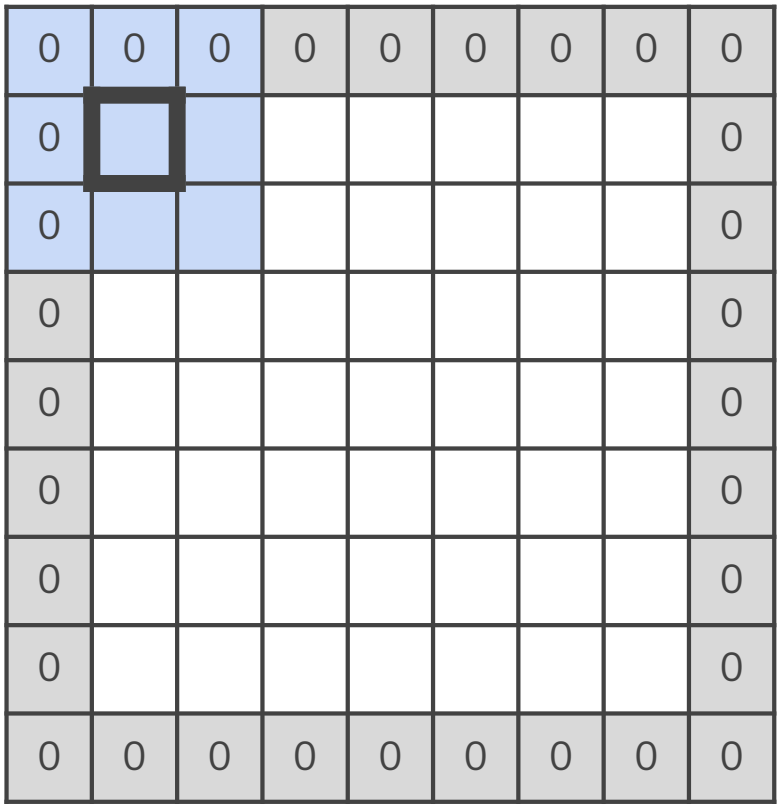


Andrej Karpathy

7 x 7 input,
3 x 3 filter applied
with **stride 1 with pad 1**

What is the output?

In Practice: Common to zero pad the border



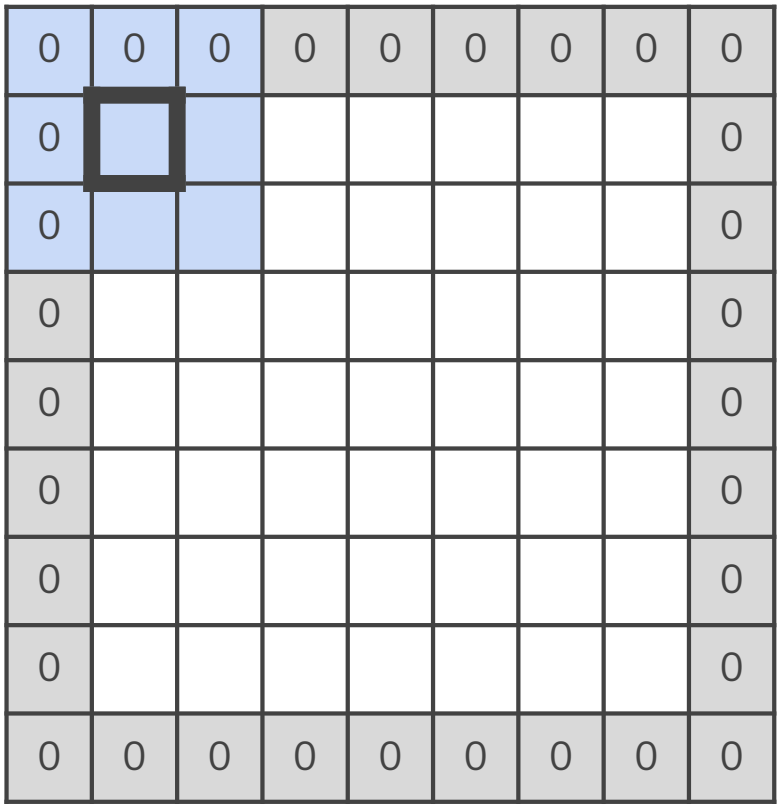
Andrej Karpathy

7 x 7 input,
3 x 3 filter applied
with **stride 1 with pad 1**

What is the output?
7 x 7 output

Output Size
(N + 2*padding - F) / stride + 1

In Practice: Common to zero pad the border



Andrej Karpathy

In general, common to see CONV layers with stride 1, filters of size F x F, and zero-padding with size **(F-1)/2 (will preserve size spatially)**.

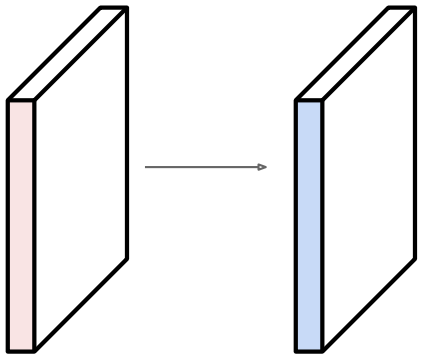
e.g. F = 3 ⇒ zero pad with size 1

e.g. F = 5 ⇒ zero pad with size 2

e.g. **F = 7 ⇒ zero pad with size 3**

Convolutions: Output Size

Examples time:

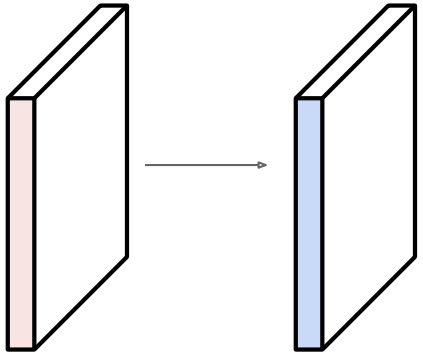


Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Output volume size: ?

Convolutions: Output Size

Examples time:

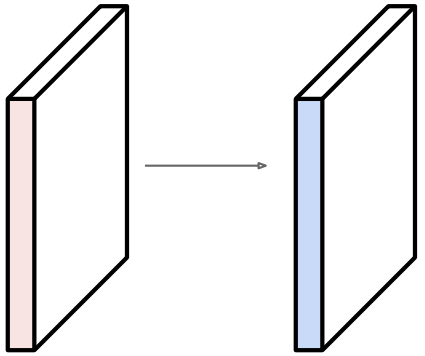


Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Convolutions: Number of Parameters

Examples time:

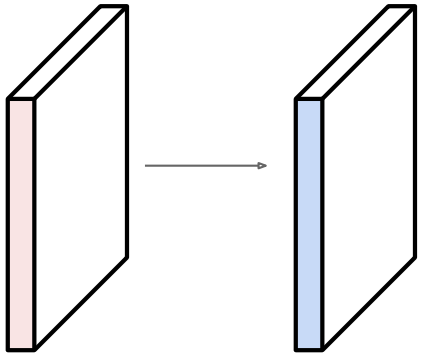


Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?

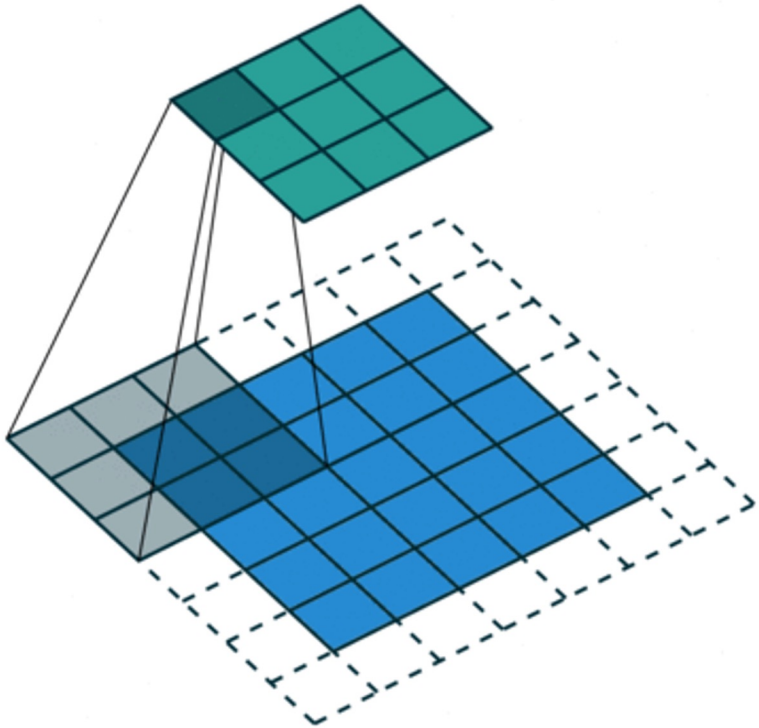
Convolutions: Number of Parameters

Examples time:

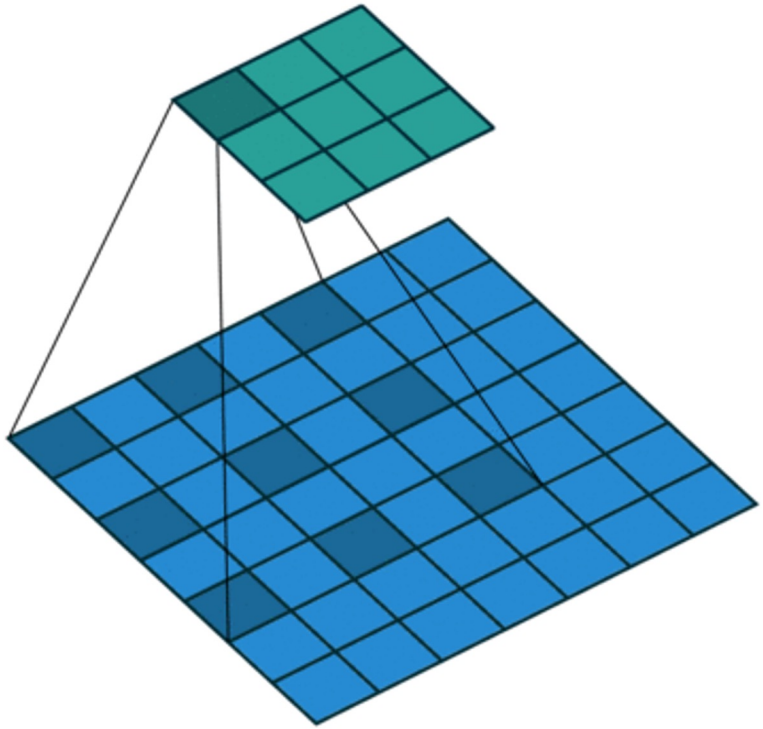


Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**



Standard Convolution

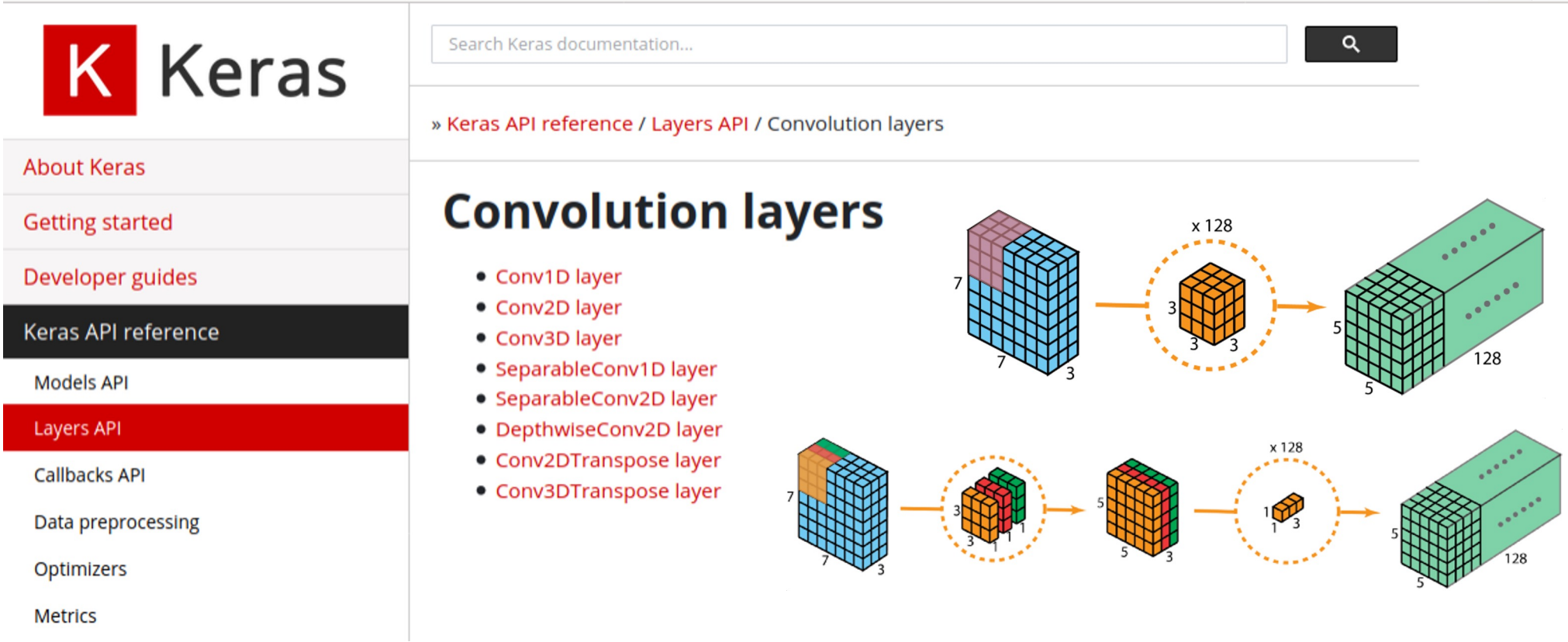


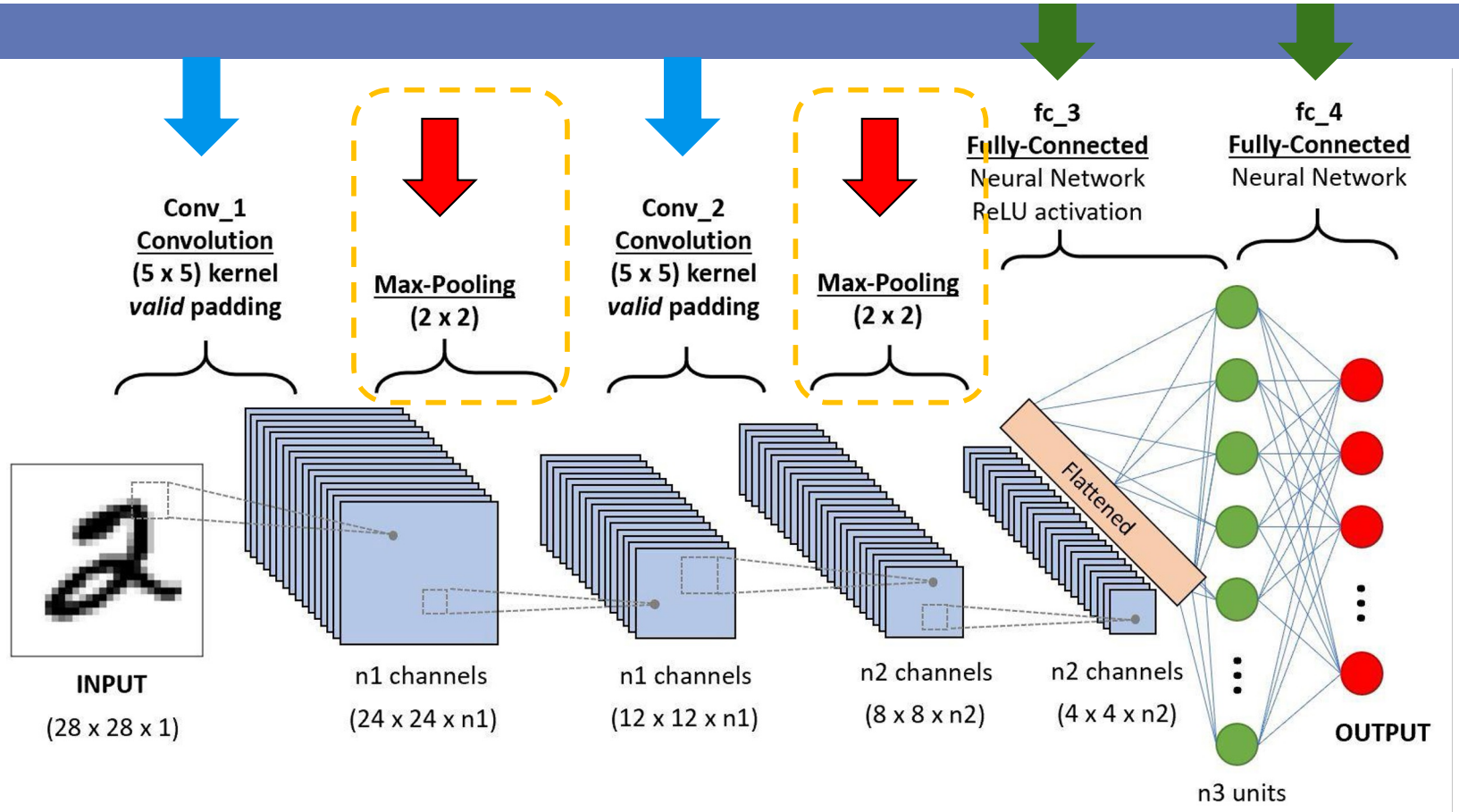
Dilated Convolution

Slide Credit: Prof. Sandra Avila - UNICAMP

Convolutions Layers

https://keras.io/api/layers/convolution_layers





There are a few distinct types of layers (e.g., **CONV**/**POOL**/**FC** are by far the most popular).

Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently

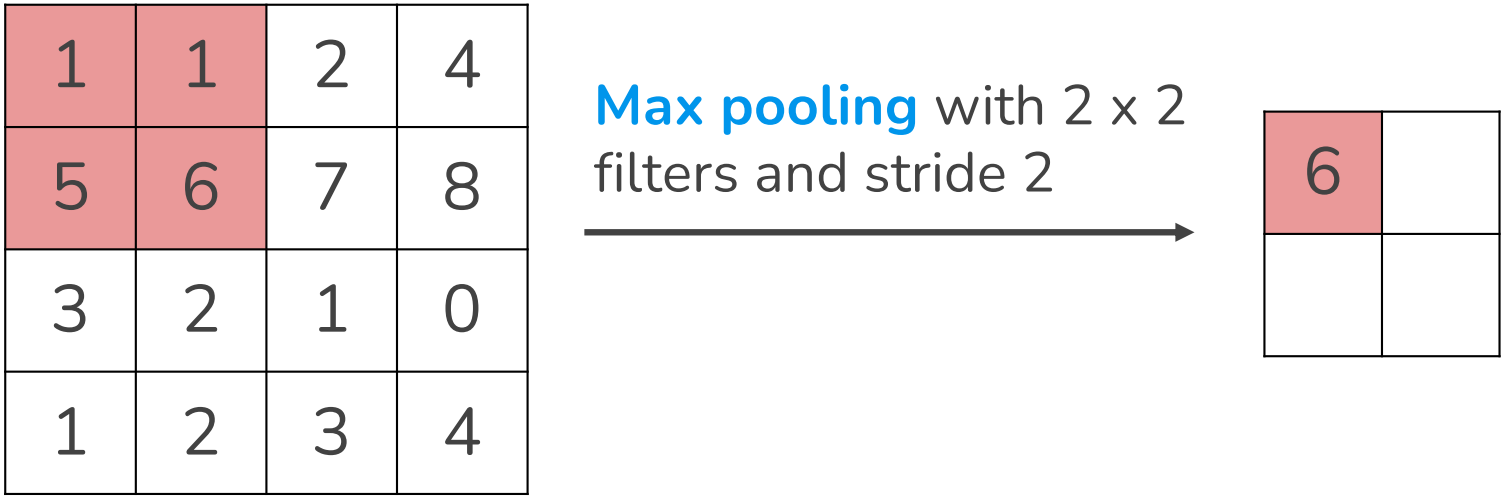
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Max pooling with 2 x 2 filters and stride 2

Slide Credit: Prof. Sandra Avila - UNICAMP

Pooling Layer

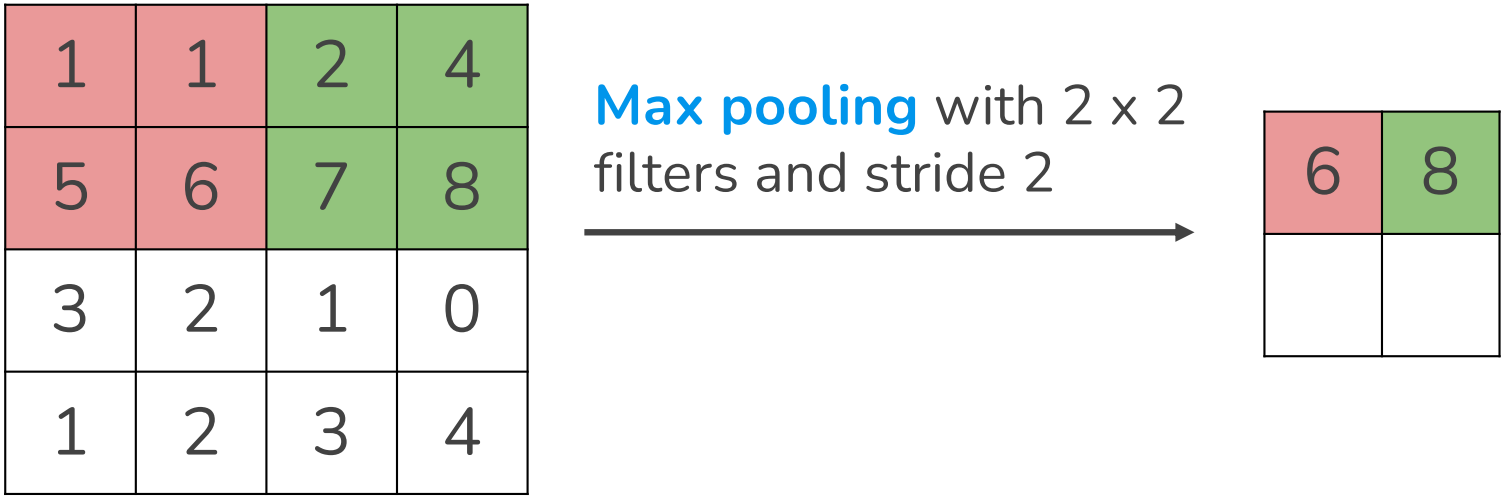
- Makes the representations smaller and more manageable
- Operates over each activation map independently



Slide Credit: Prof. Sandra Avila - UNICAMP

Pooling Layer

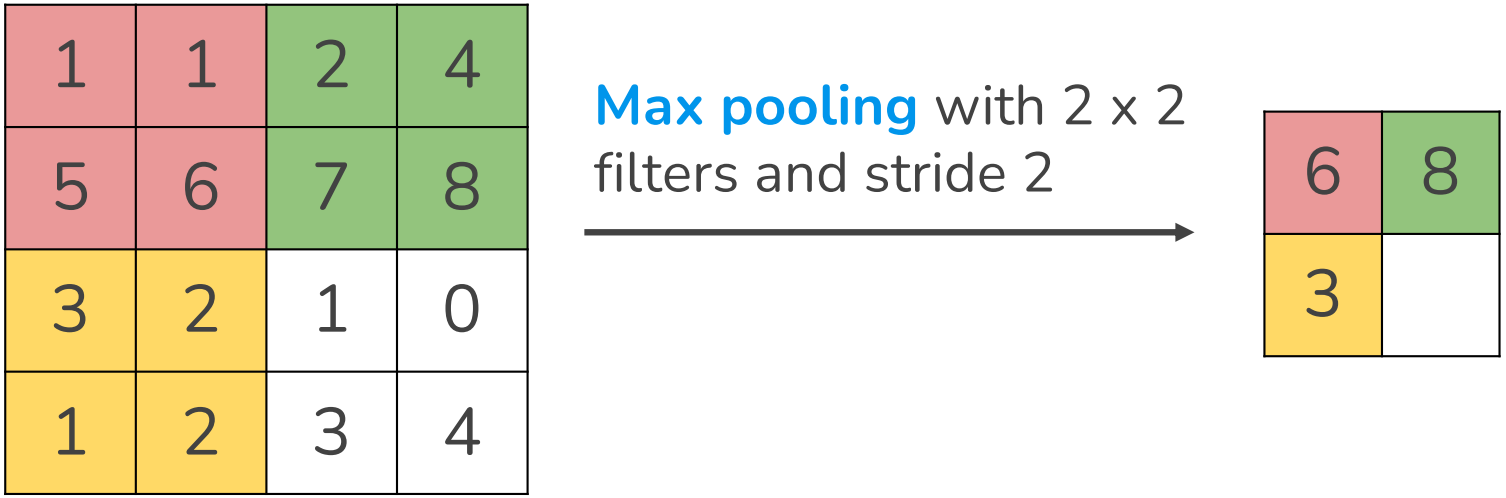
- Makes the representations smaller and more manageable
- Operates over each activation map independently



Slide Credit: Prof. Sandra Avila - UNICAMP

Pooling Layer

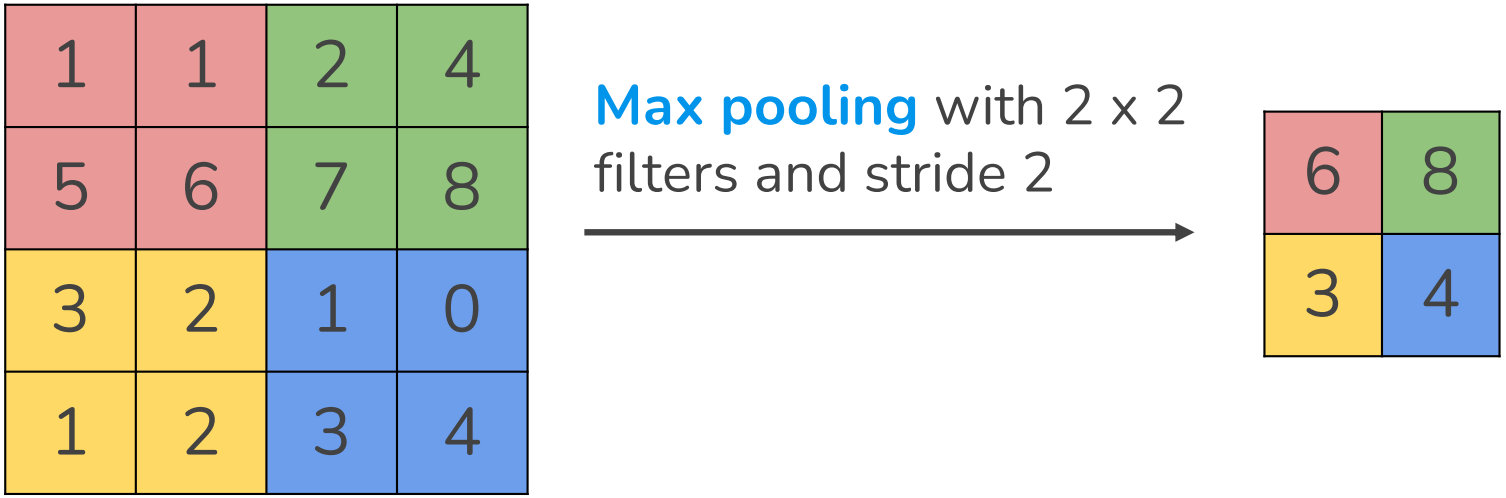
- Makes the representations smaller and more manageable
- Operates over each activation map independently



Slide Credit: Prof. Sandra Avila - UNICAMP

Pooling Layer

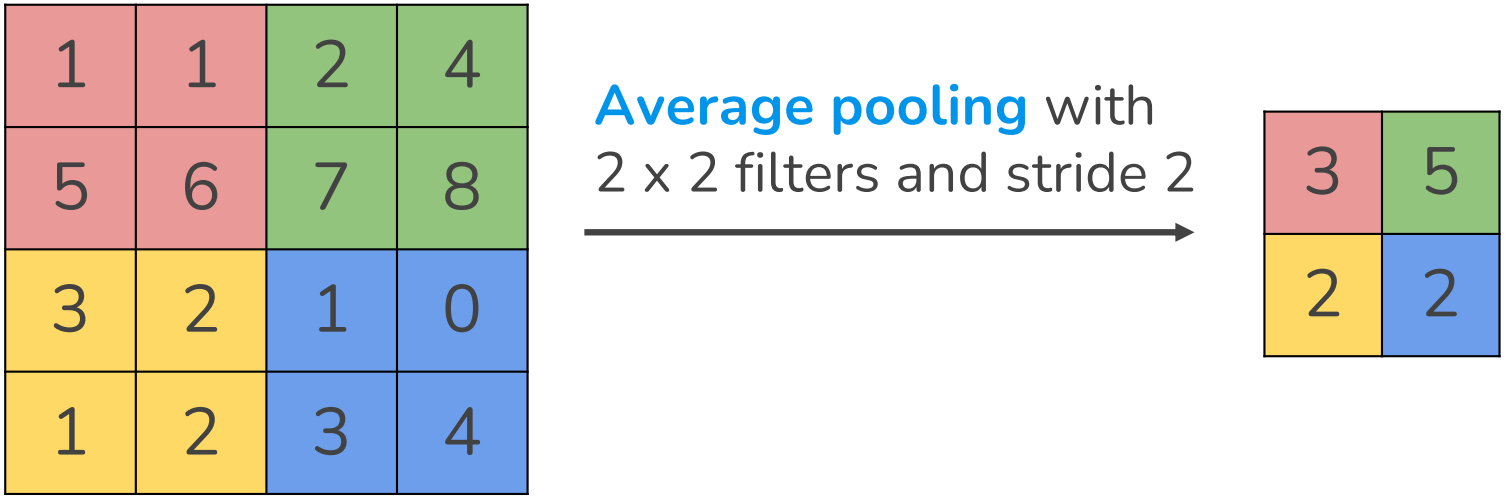
- Makes the representations smaller and more manageable
- Operates over each activation map independently



Slide Credit: Prof. Sandra Avila - UNICAMP

Pooling Layer

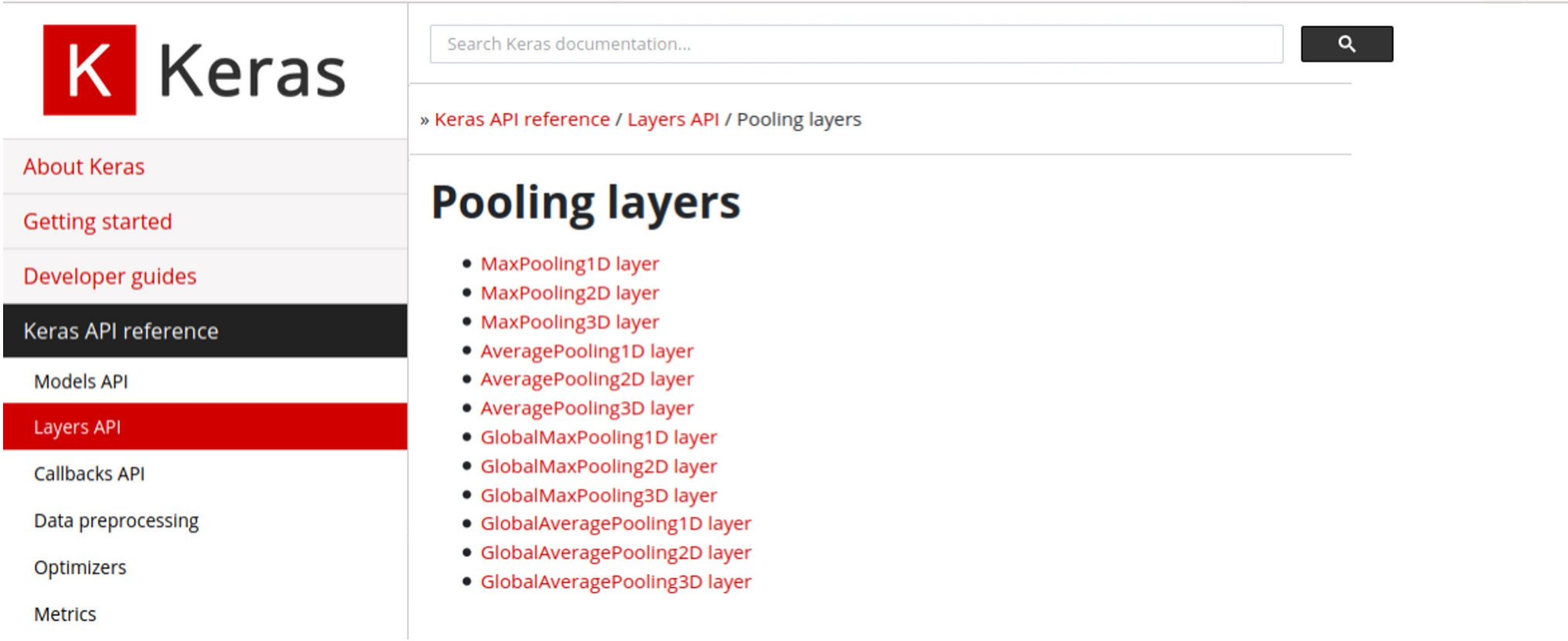
- Makes the representations smaller and more manageable
- Operates over each activation map independently

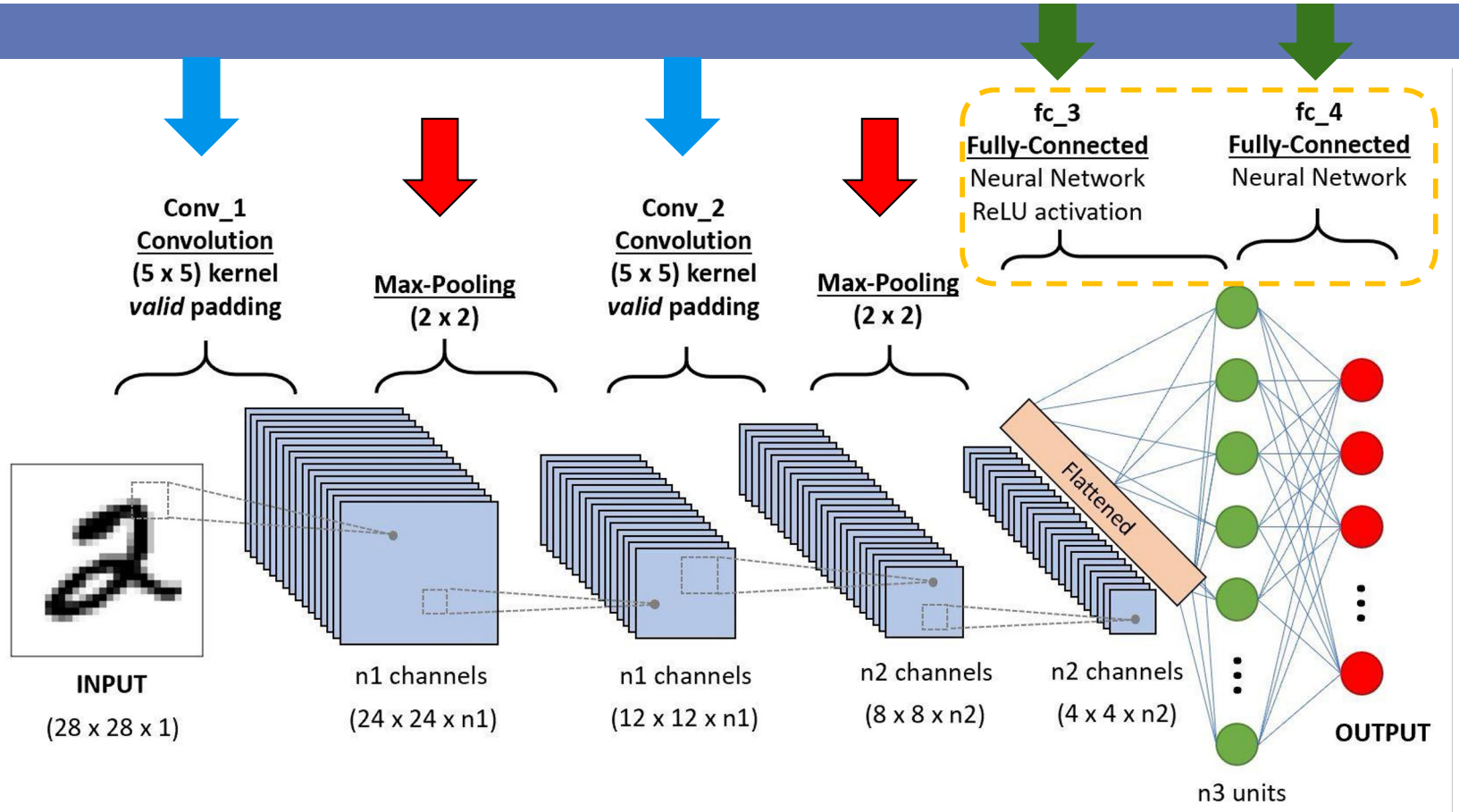


Slide Credit: Prof. Sandra Avila - UNICAMP

Pooling Layers

https://keras.io/api/layers/pooling_layers





There are a few distinct types of layers (e.g., **CONV**/**POOL**/**FC** are by far the most popular).

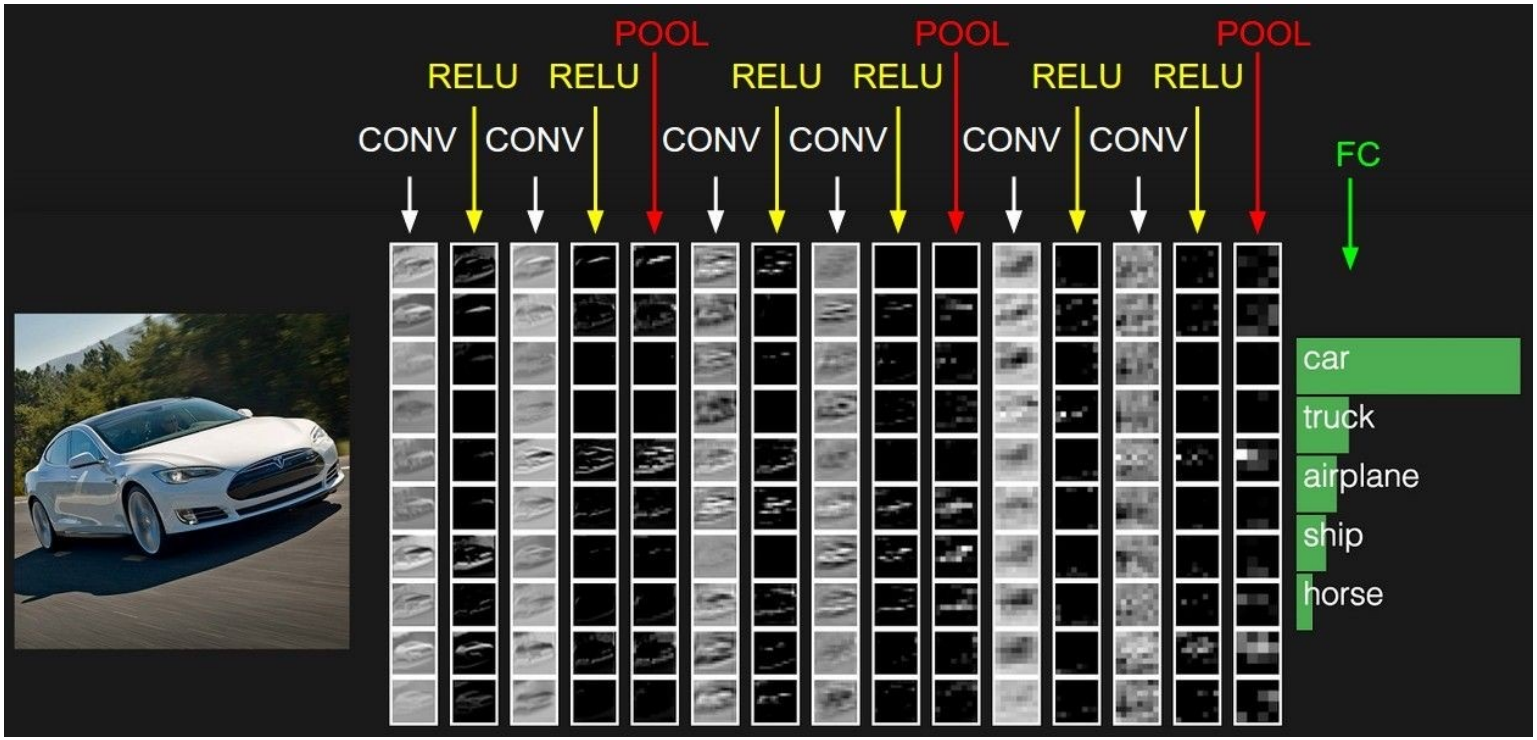
Slide Credit: Prof. Sandra Avila - UNICAMP

Fully Connected Layer

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

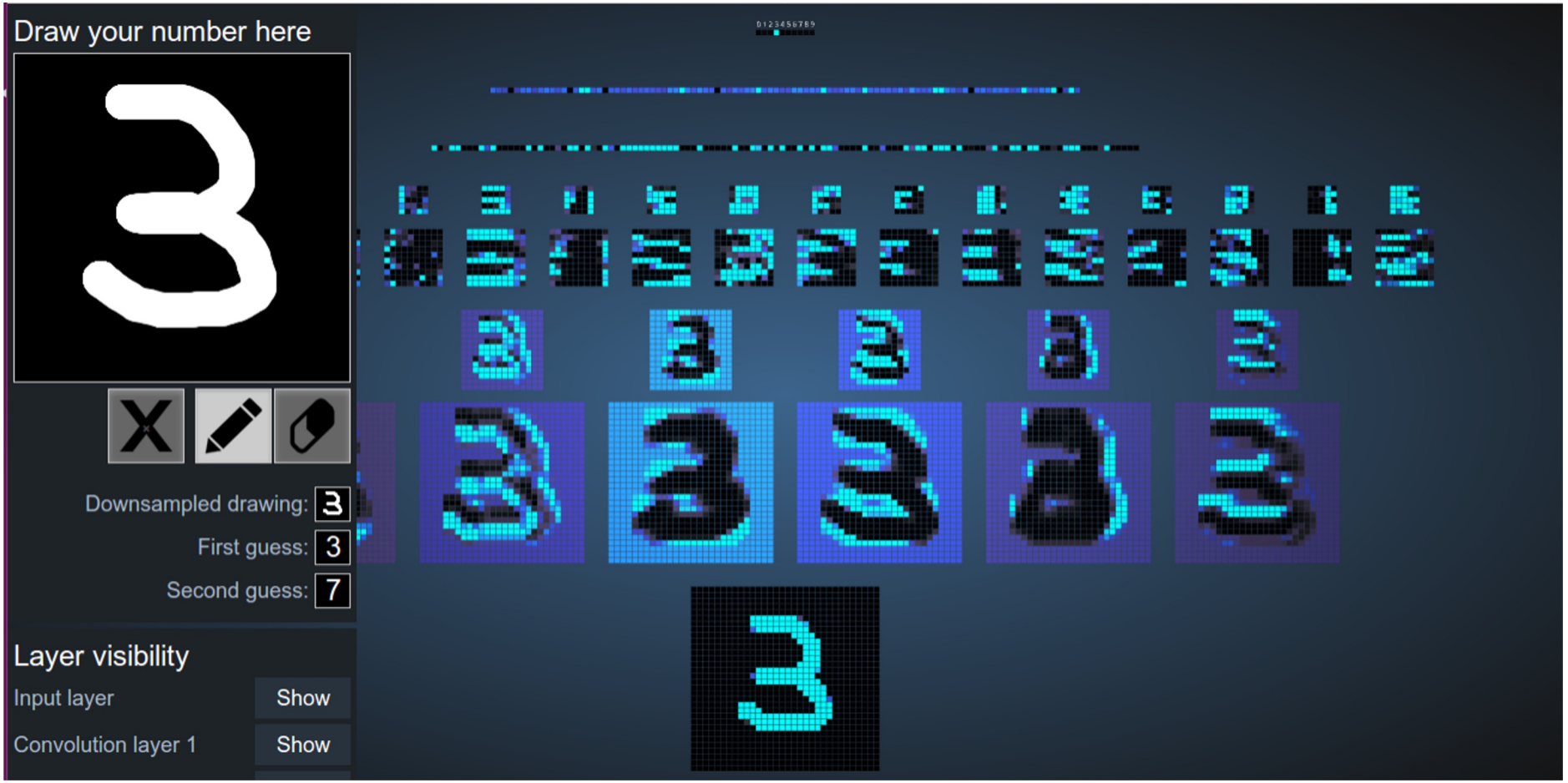
Slide Credit: Prof. Sandra Avila - UNICAMP

Putting it all together



Andrej Karpathy

https://adamharley.com/nn_vis/cnn/2d.html

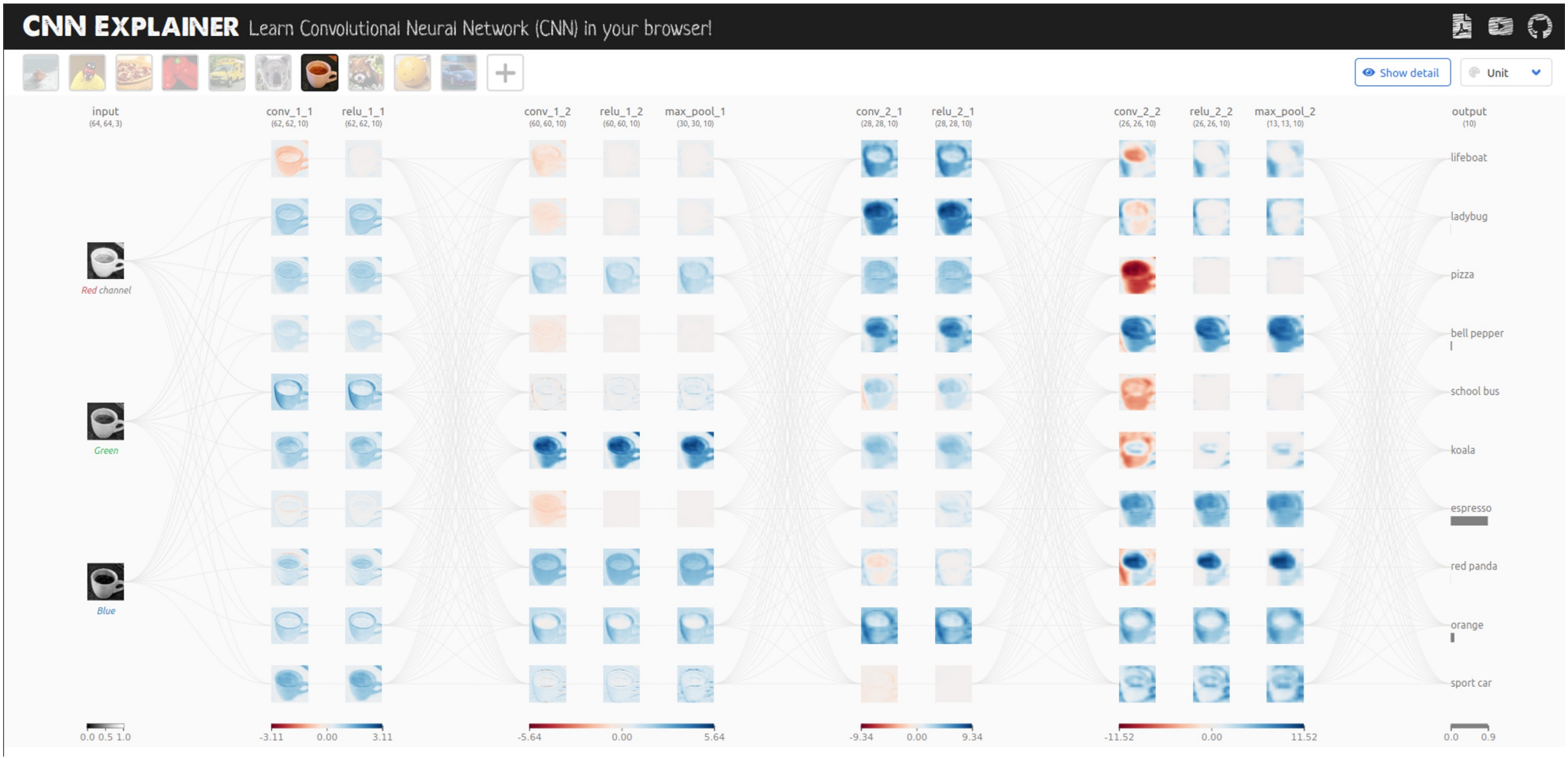


Visualizing a CNN trained on Handwritten Digits

- Input image: 1024 pixels (32 x 32 image)
- CONV 1 (+ RELU): 6 5 × 5 (stride 1) filters
- POOL 1: 2 × 2 max pooling (with stride 2)
- CONV 2 (+ RELU): 16 5 × 5 (stride 1) filters
- POOL 2: 2 × 2 max pooling (with stride 2)
- 2 FC layers:
  - 120 neurons in the first FC layer
  - 100 neurons in the second FC layer
- Output layer: 10 neurons in the third FC

Slide Credit: Prof. Sandra Avila - UNICAMP

https://poloclub.github.io/cnn-explainer

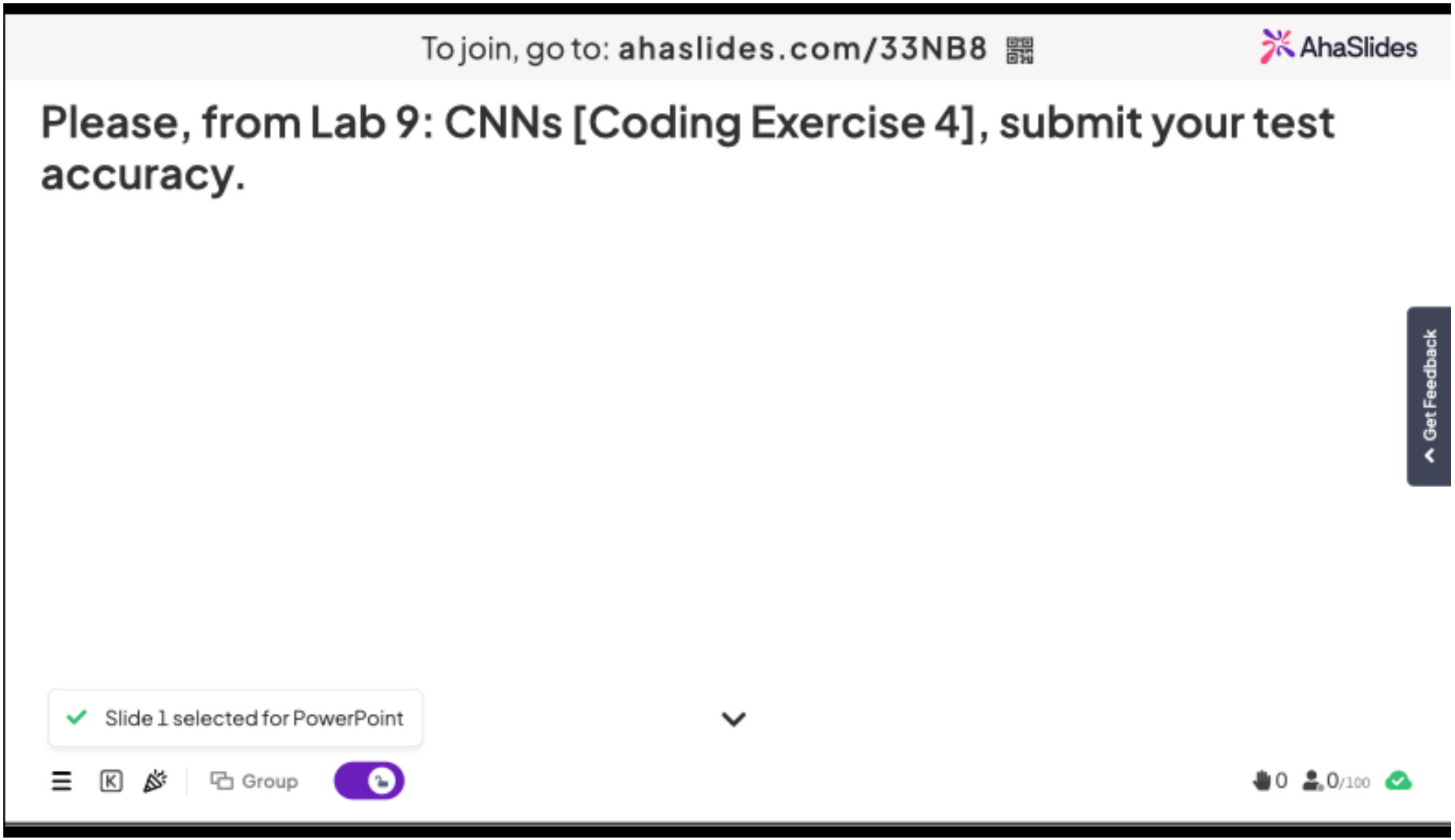


Lab 9: CNN

Duration: 15 min

Steps:

1. Introduce to a classmate
2. Work in pairs on the exercise
3. Submit your answers on AhaSlides


To join, go to: ahaslides.com/33NB8
AhaSlides
Please, from Lab 9: CNNs [Coding Exercise 4], submit your test accuracy.
Get Feedback
Slide 1 selected for PowerPoint
Group
0
0/100

Assignment 6: CNN

CNN Architectures

CNN Architectures

- CNN Architectures
  - LeNet (1998)
  - **AlexNet (2012)**
  - ZFNet (2013)
  - VGGNet (2014)
  - GoogLeNet (2014)
  - ResNet (2015)

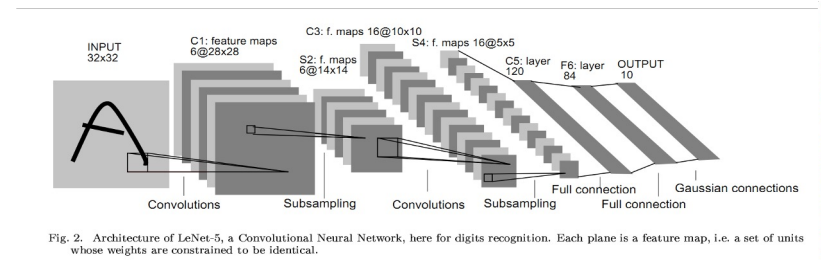
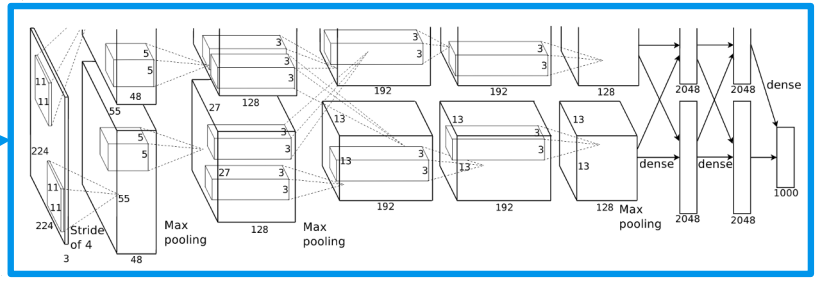
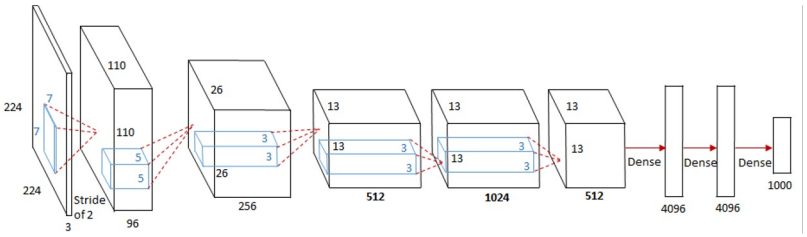
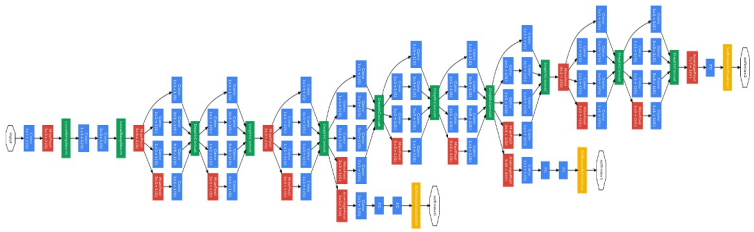
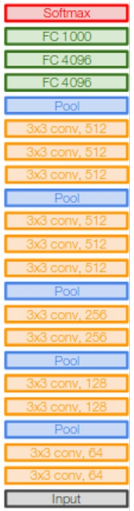


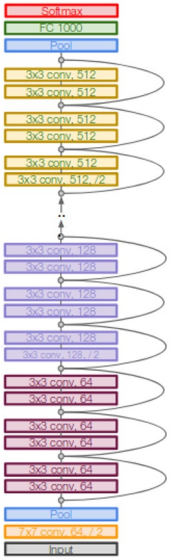
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

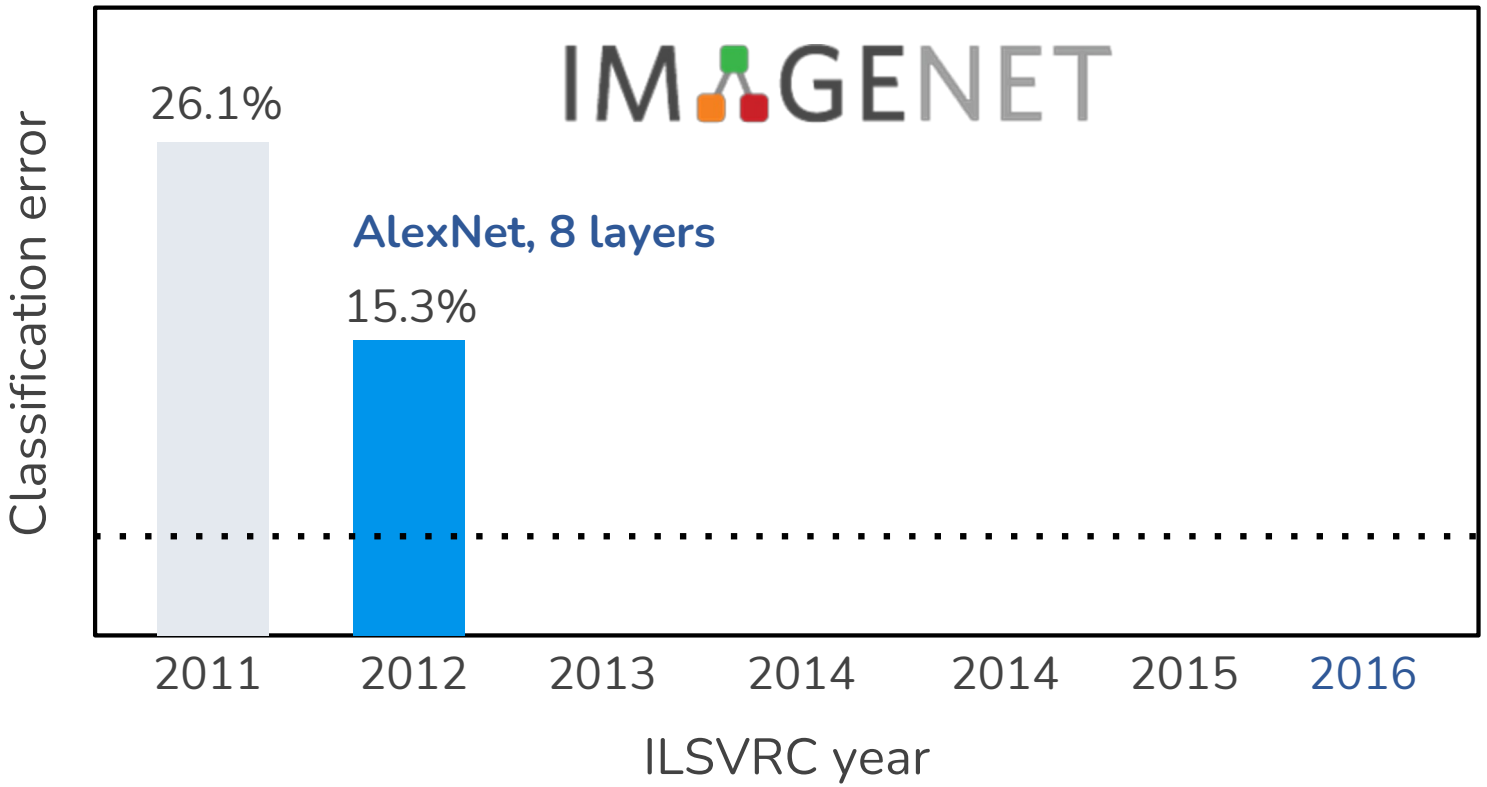








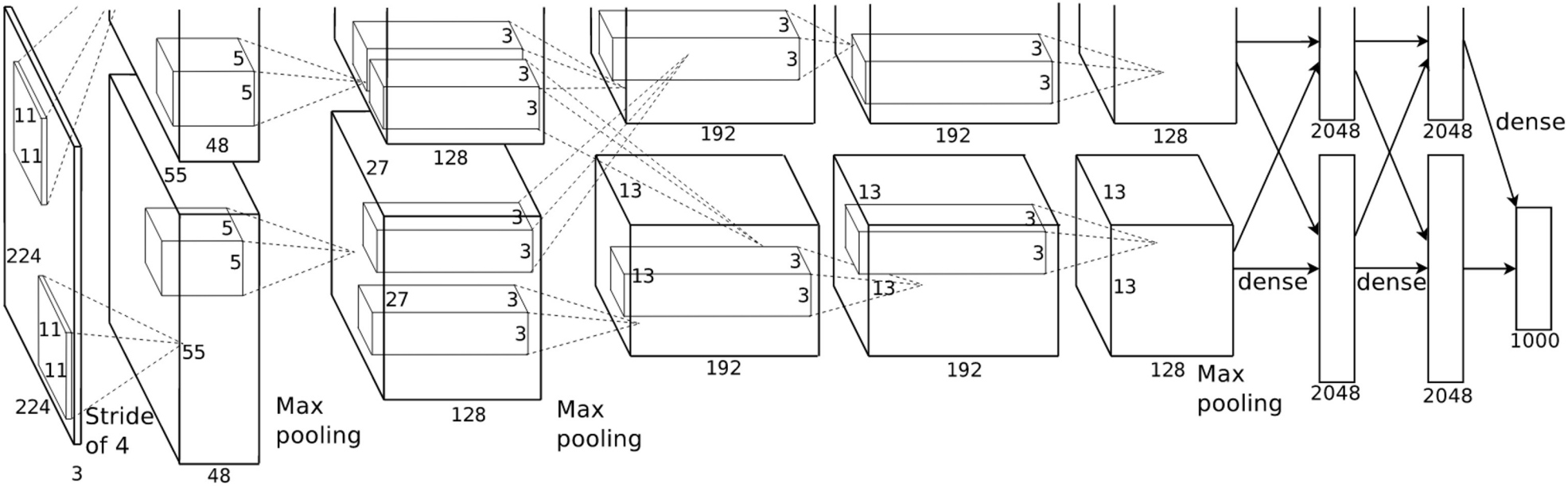




"ImageNet classification with deep convolutional neural networks". NIPS, 2012.

Slide Credit: Prof. Sandra Avila - UNICAMP

AlexNet [Krizhevsky et al., 2012]



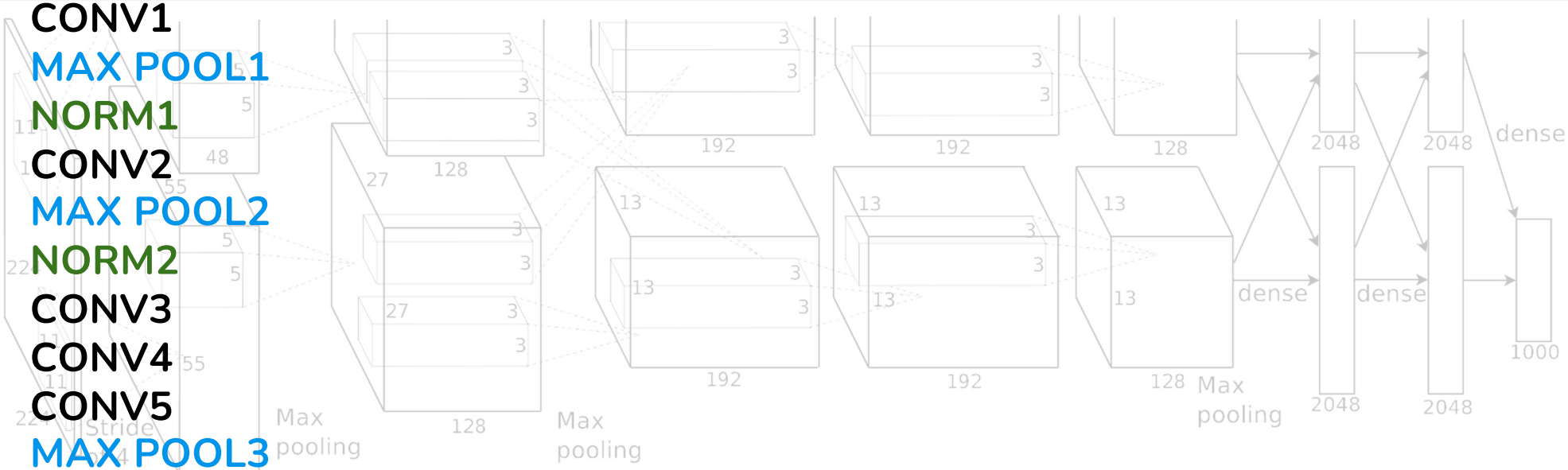
"ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.

Slide Credit: Prof. Sandra Avila - UNICAMP

AlexNet [Krizhevsky et al., 2012]

Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
MAX POOL3
FC6
FC7
FC8

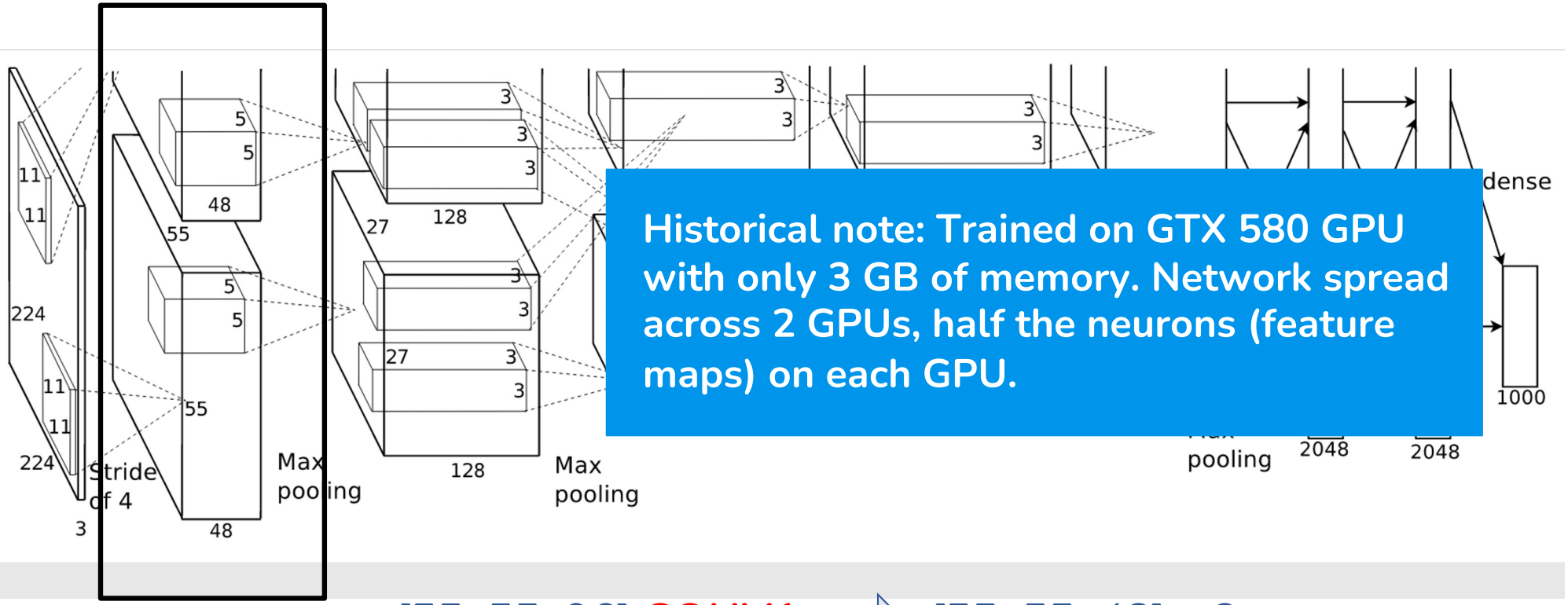


Slide Credit: Prof. Sandra Avila - UNICAMP

AlexNet [Krizhevsky et al., 2012]

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

AlexNet [Krizhevsky et al., 2012]



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

[55x55x96] **CONV1** ⇨ [55x55x48] x 2

Slide Credit: Prof. Sandra Avila - UNICAMP

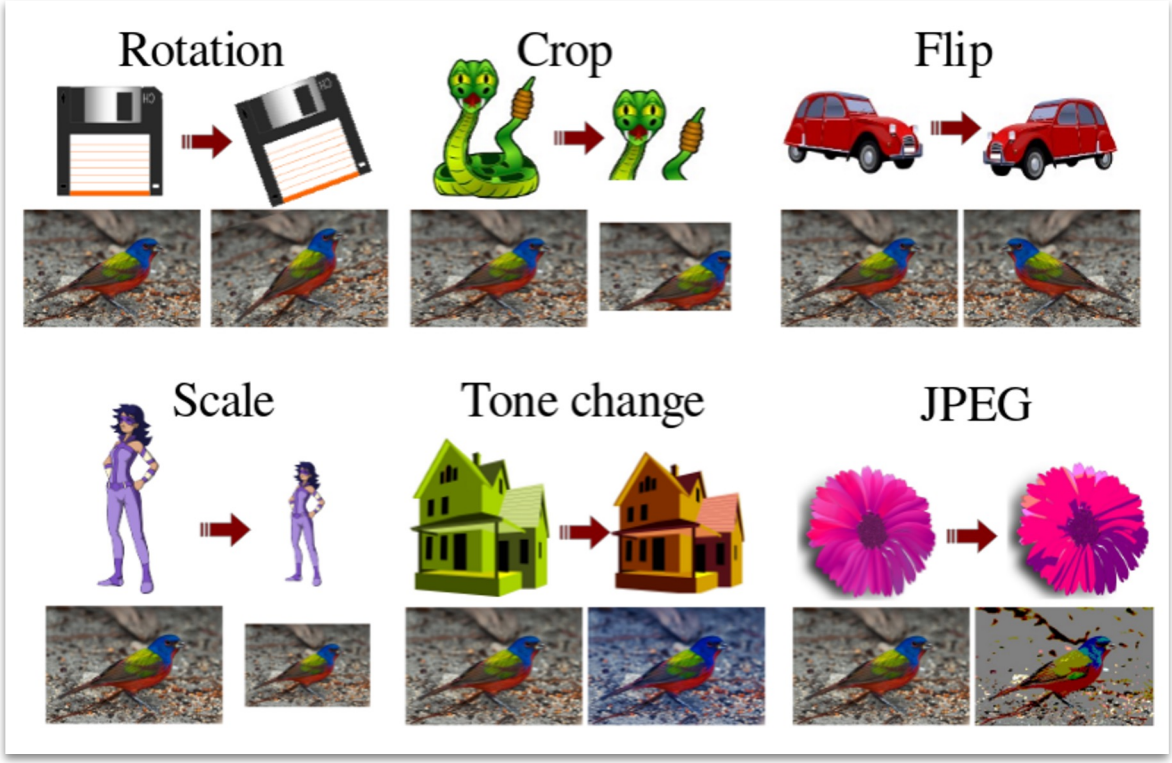
AlexNet [Krizhevsky et al., 2012]

Details:

- 60 million learned parameters
- first use of ReLU
- used Norm layers
- heavy **data augmentation**
- dropout 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.3%
- 5-6 days to train on 2 GTX 580 3GB GPUs

Slide Credit: Prof. Sandra Avila - UNICAMP

Data Augmentation

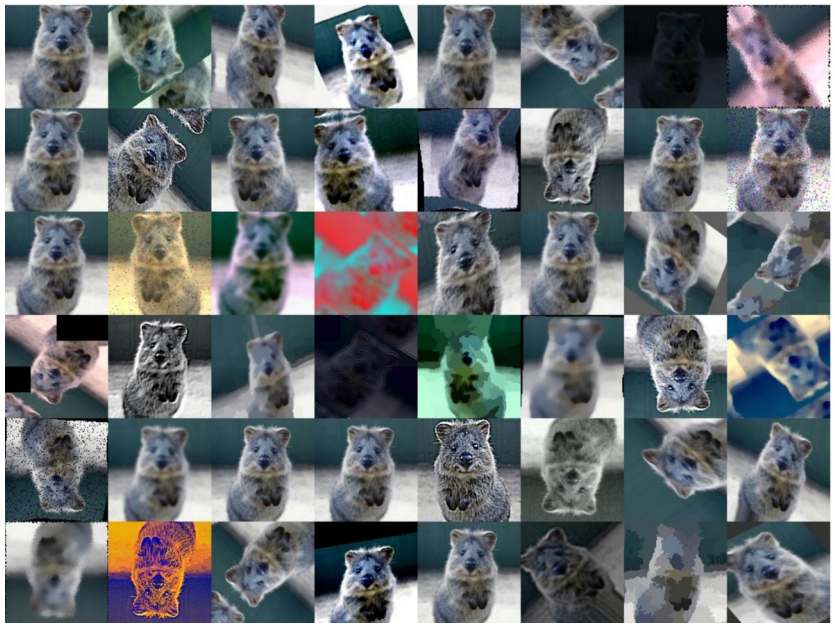


"Transformation Pursuit for Image Classification", CVPR 2014.

Slide Credit: Prof. Sandra Avila - UNICAMP

Data Augmentation

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



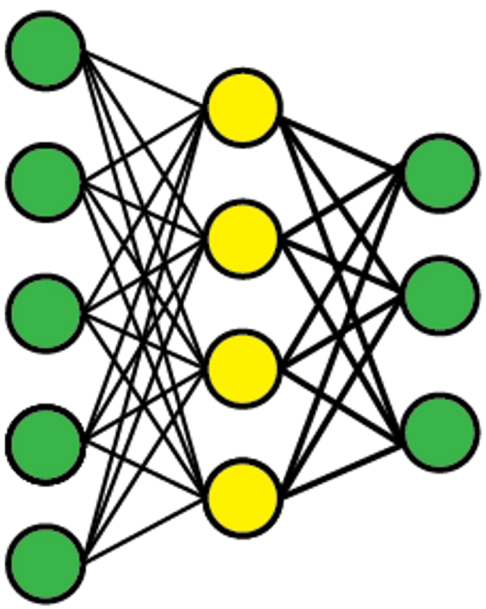
Jia-bin Huang, Image: https://github.com/aleju/imgaug

AlexNet [Krizhevsky et al., 2012]

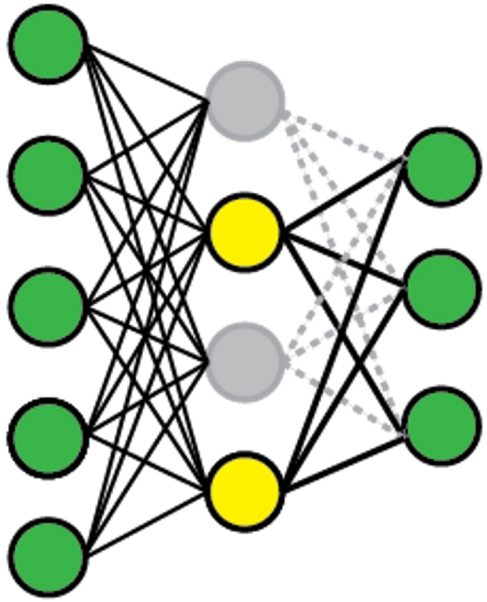
Details:

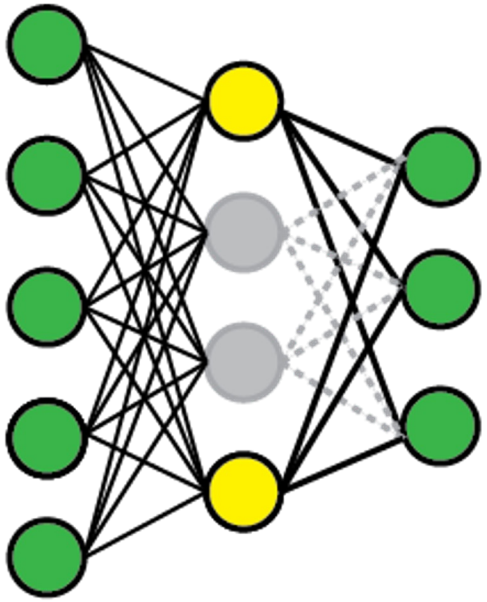
- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- **dropout** 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.3%

Dropout [Hinton et al., 2012]



Standard Network



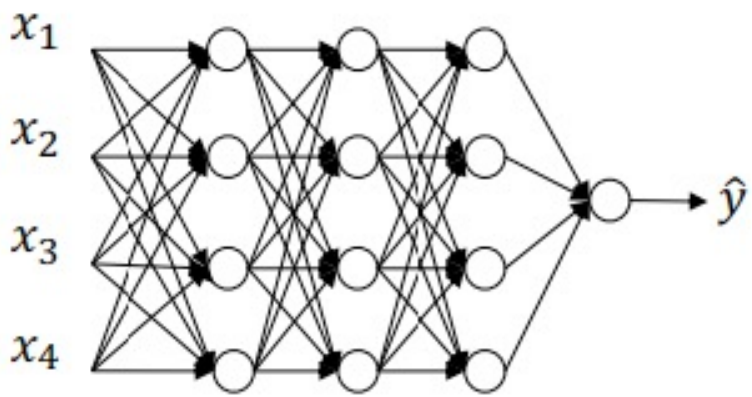


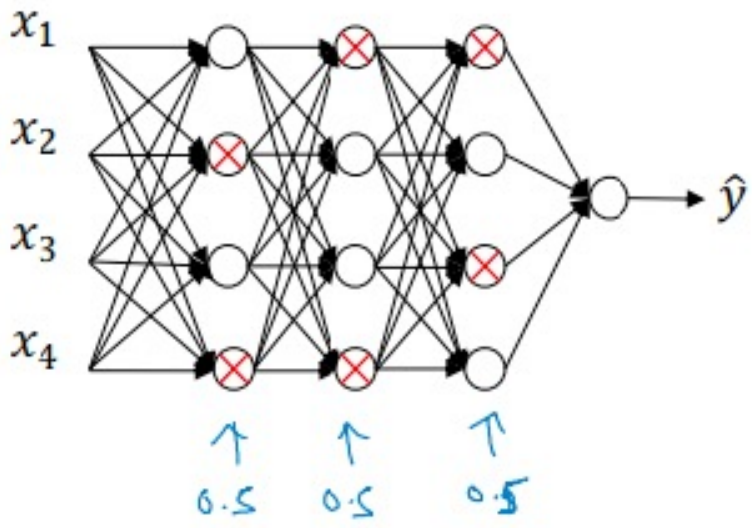
After applying dropout

"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Slide Credit: Prof. Sandra Avila - UNICAMP

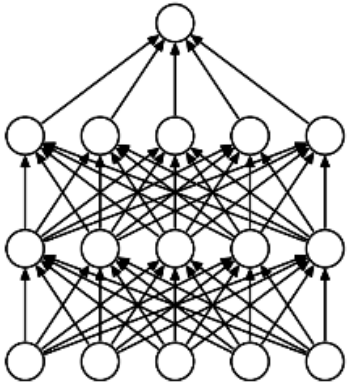
Regularization: Dropout



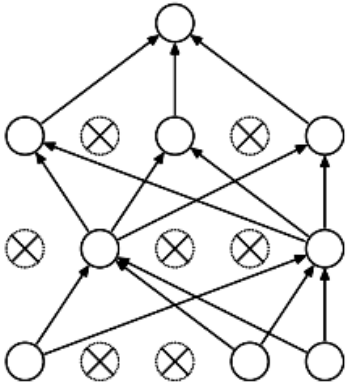


Adapted from Deep Coursera Specialization [Andrew Ng]

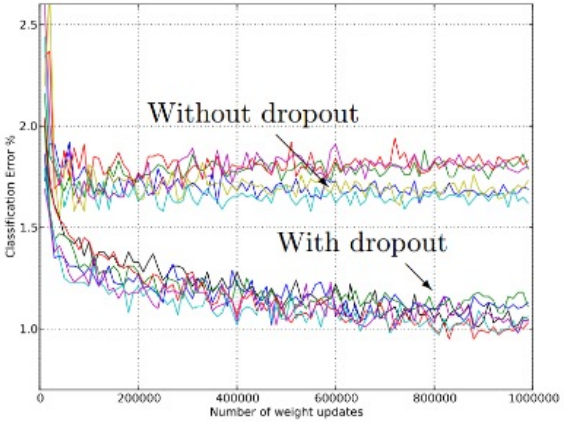
Regularization: Dropout



(a) Standard Neural Net



(b) After applying dropout.



- Randomly turn off some neurons
- Allows individual neurons to independently be responsible for performance

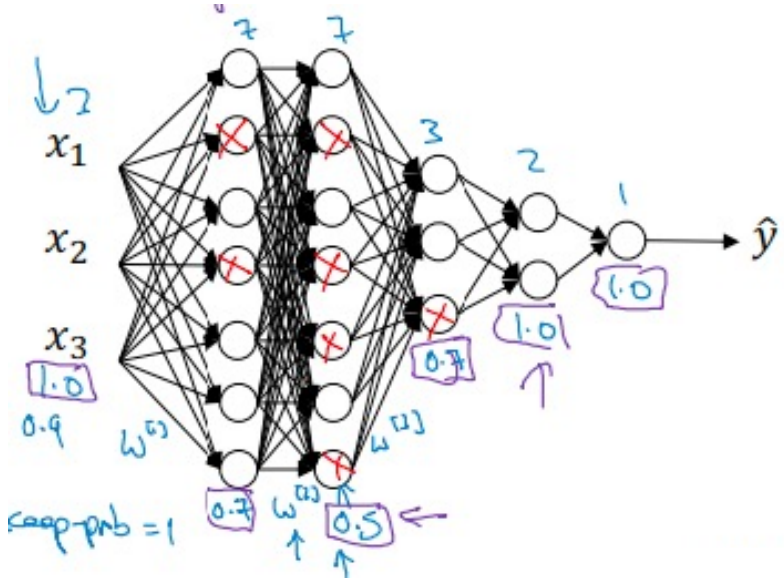
Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Adapted from Jia-bin Huang

Regularization: Why does dropout work?

Intuition: Can't rely on any one feature, so have to spread out weights (knowledge).

Have redundant information across different neurons

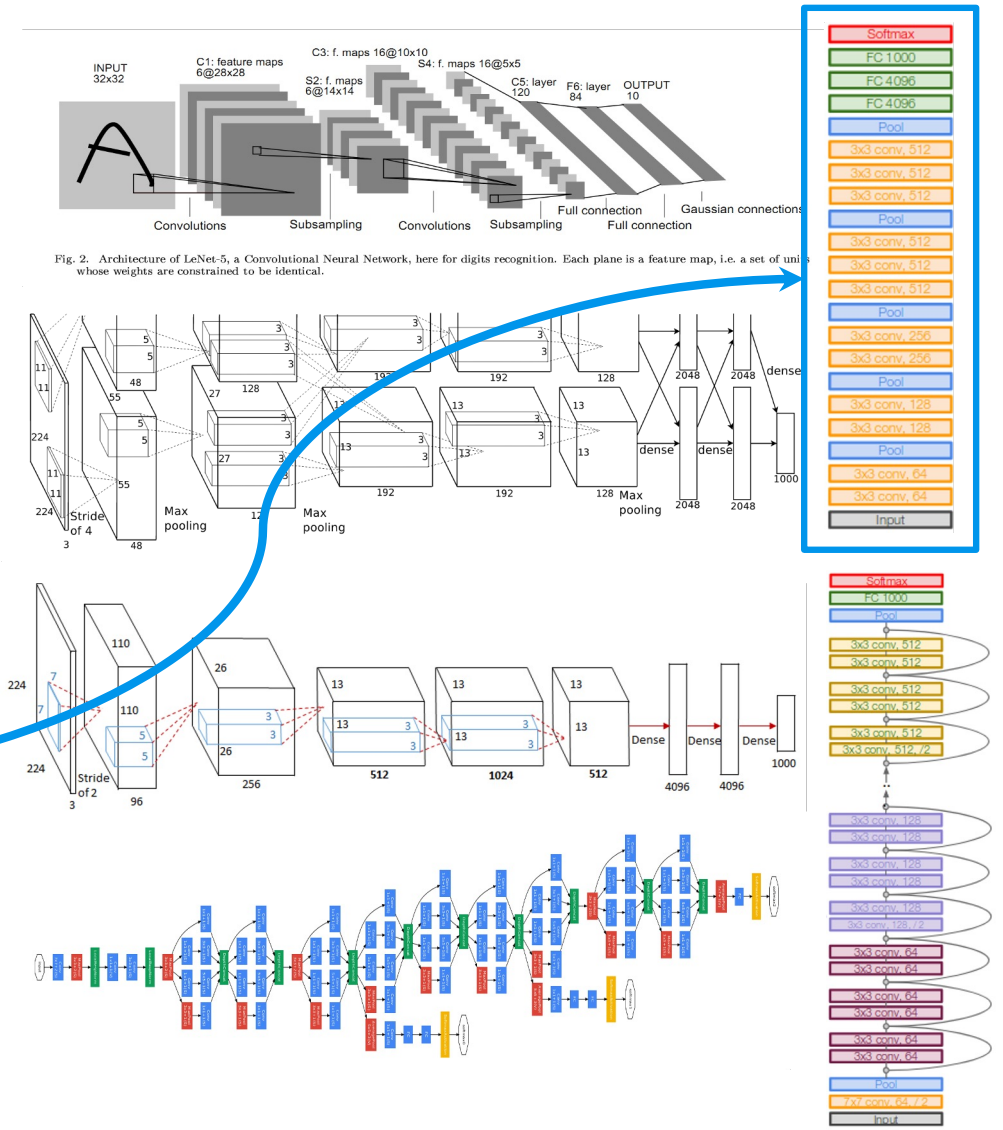


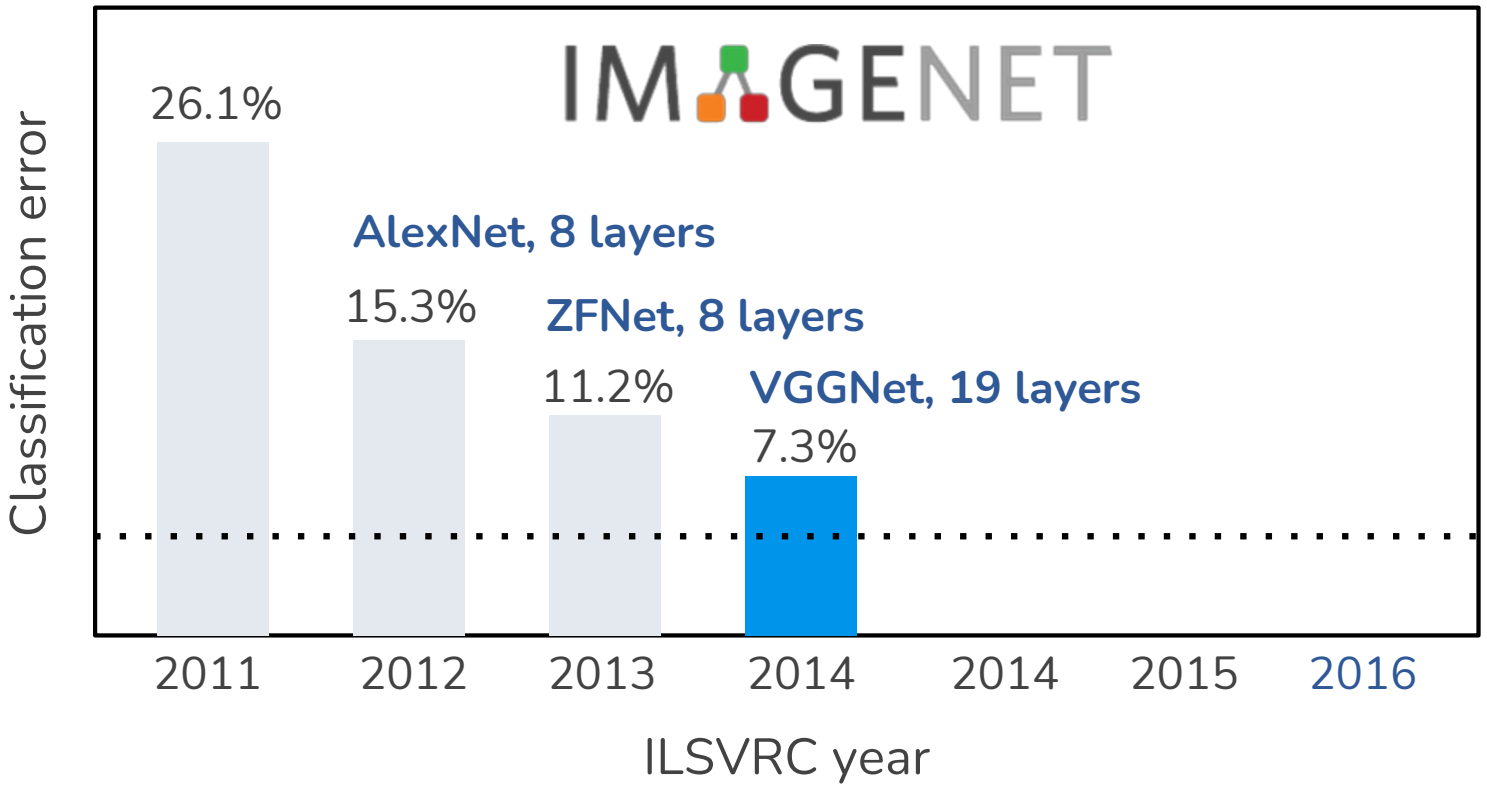
Adapted from Deep Coursera Specialization [Andrew Ng]

[Extra] Lab 8: Regularization Section 2

CNN Architectures

- CNN Architectures
  - LeNet (1998)
  - AlexNet (2012)
  - ZFNet (2013)
  - **VGGNet (2014)**
  - GoogLeNet (2014)
  - ResNet (2015)





"Very Deep Convolutional Networks for Large-Scale Image Recognition", https://arxiv.org/pdf/1409.1556

Slide Credit: Prof. Sandra Avila - UNICAMP

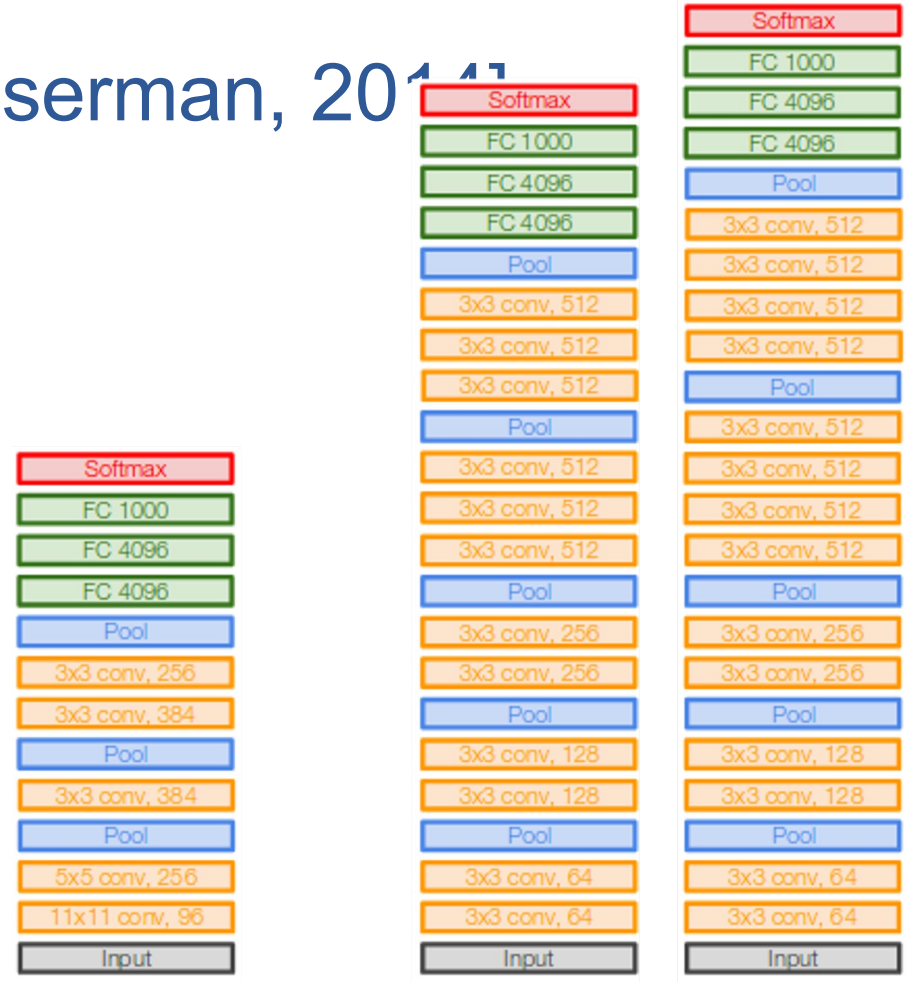
VGGNet [Simonyan & Zisserman, 2014]

Small filters, Deeper networks

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

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

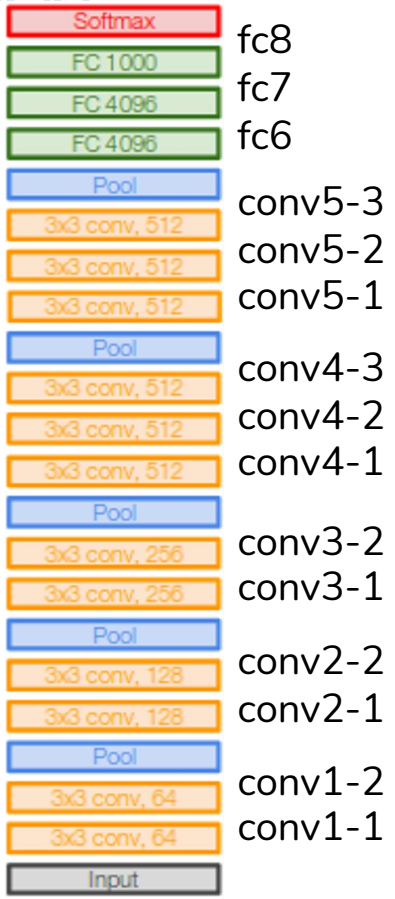
11.2% in ILSVRC'13 (ZFNet)
7.3% in ILSVRC'14



VGGNet [Simonyan & Zisserman, 2014]

Details:

- 138M parameters
- 2nd in classification, 1st in localization
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



AlexNet VGG16

Slide Credit: Prof Sandra Avila - UNICAMP

CNN Architectures

- CNN Architectures
  - LeNet (1998)
  - AlexNet (2012)
  - ZFNet (2013)
  - VGGNet (2014)
  - **GoogLeNet (2014)**
  - ResNet (2015)

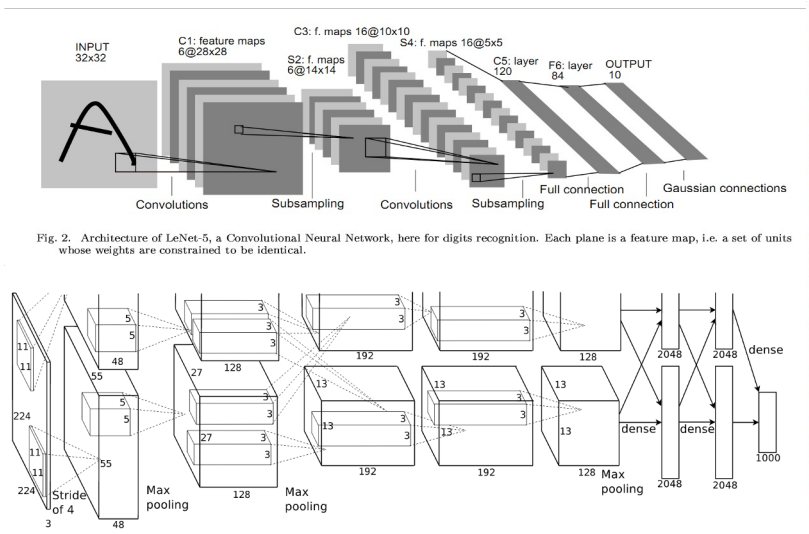
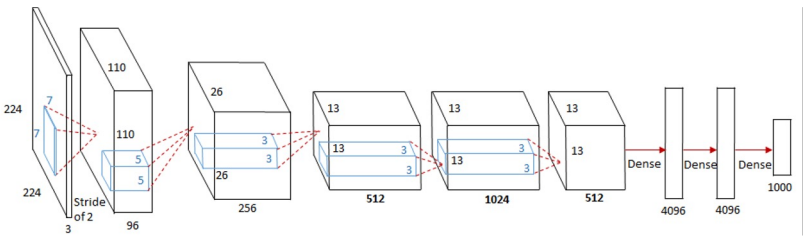
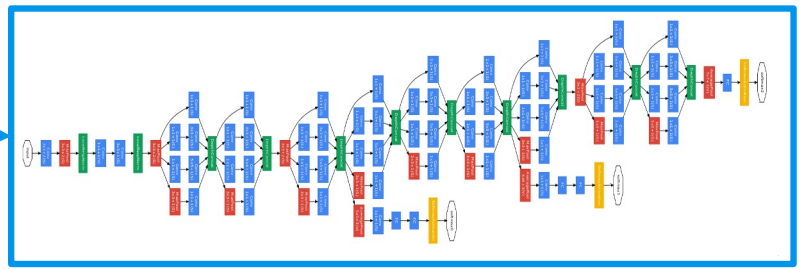
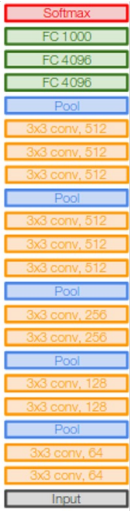
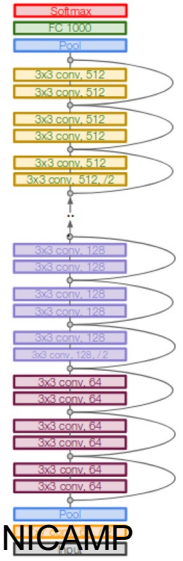


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

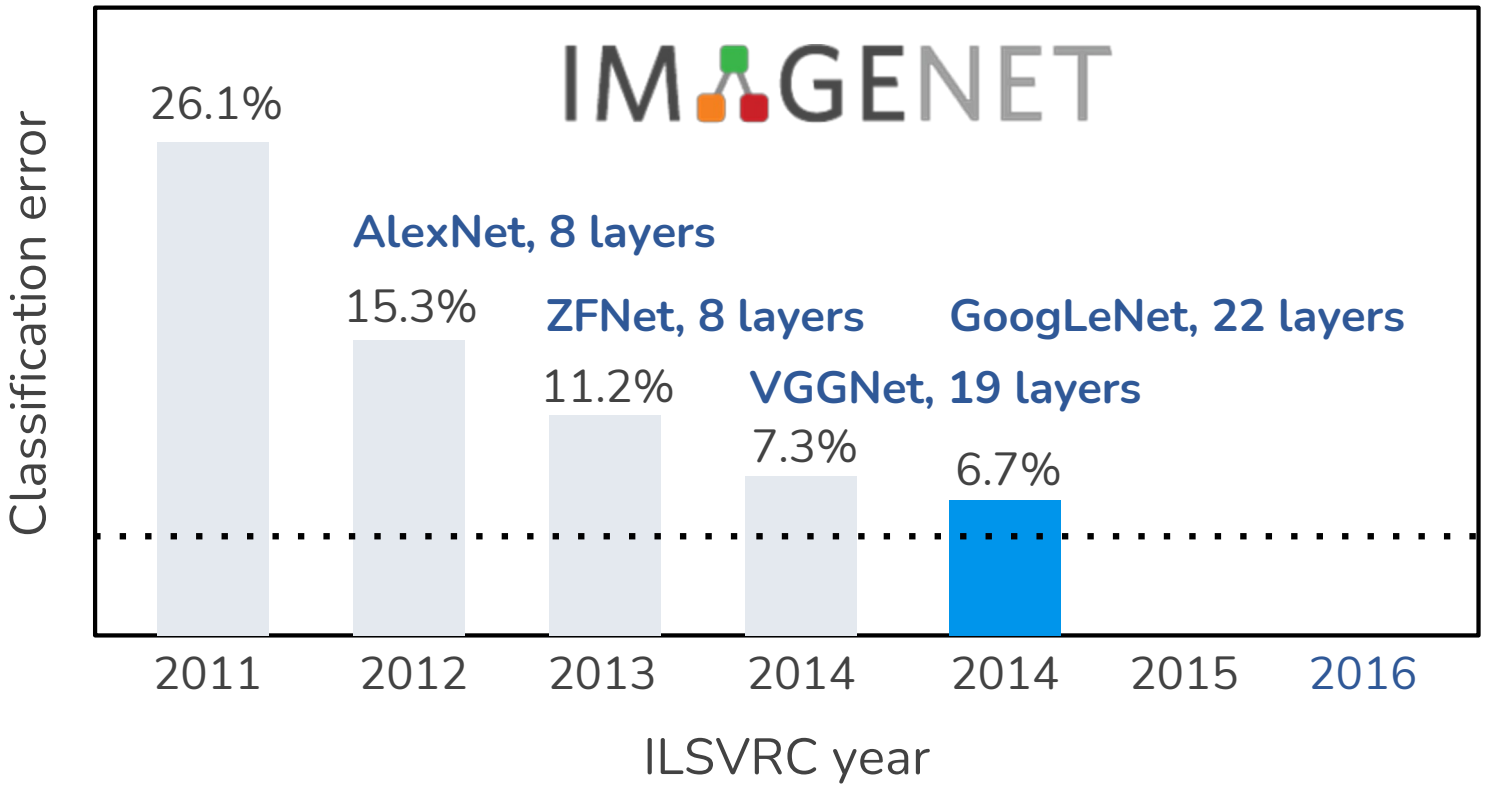








Slide Credit: Prof. Sandra Avila - UNICAMP



"Very Deep Convolutional Networks for Large-Scale Image Recognition", https://arxiv.org/pdf/1409.1556

GoogLeNet [Szegedy et al., 2014]

Deeper networks, with computational efficiency

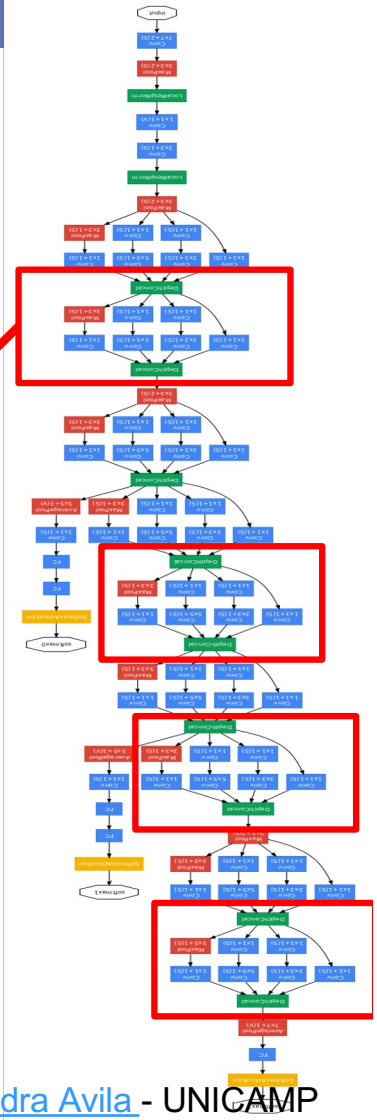
- 22 layers
- Inception module
- No FC layers
- Only 5 million parameters!

  12x less than AlexNet

Slide Credit: Prof. Sandra Avila - UNICAMP

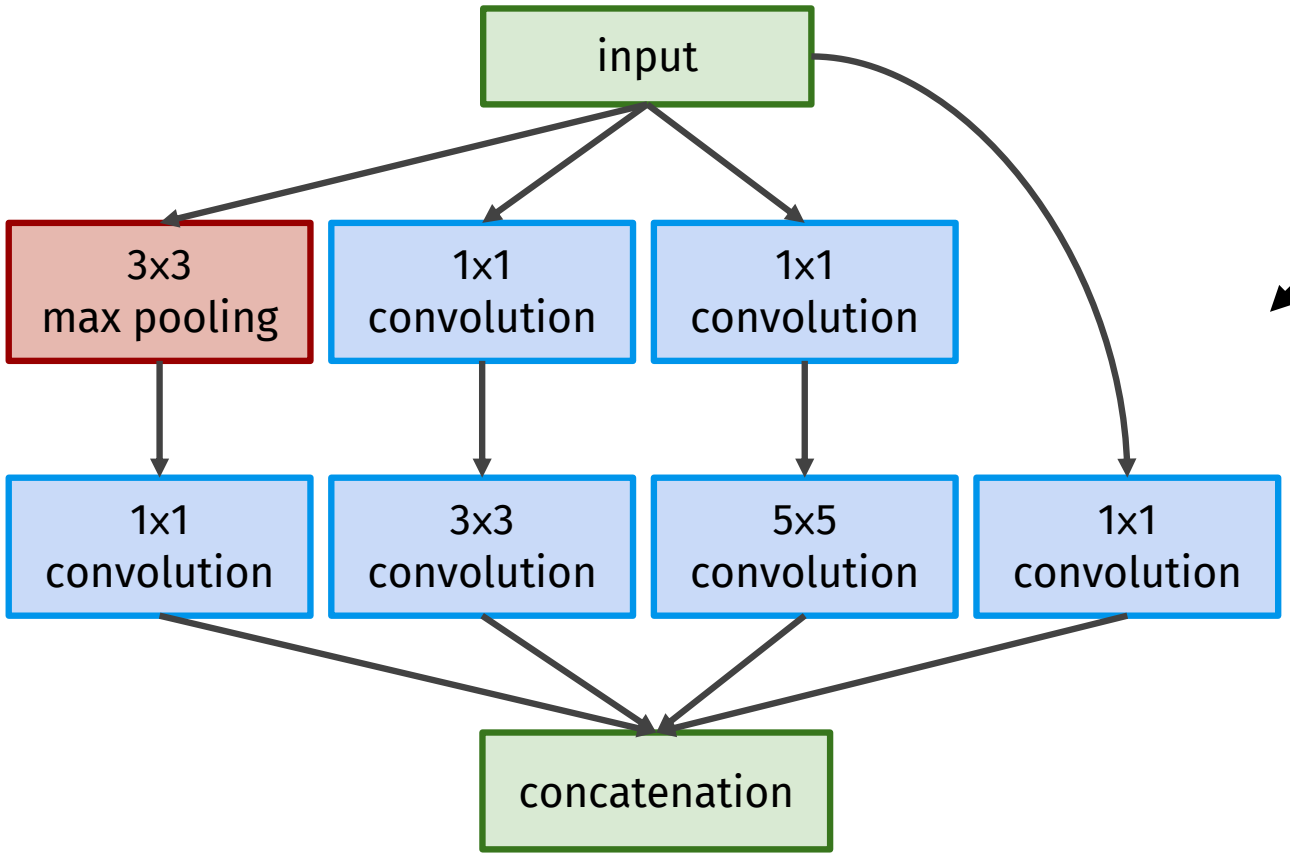
GoogLeNet [Szegedy et al., 2014]

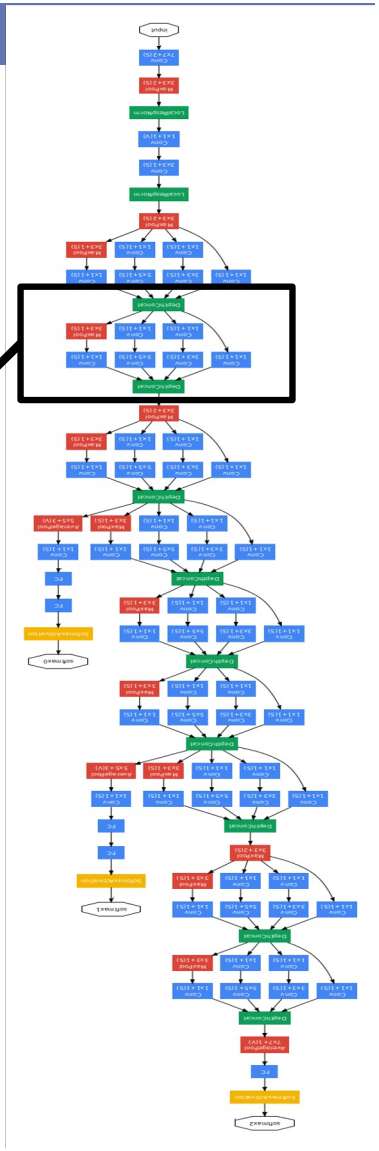
Inception module: design a good local network topology (network within a network) and then stack these modules on top of each other.

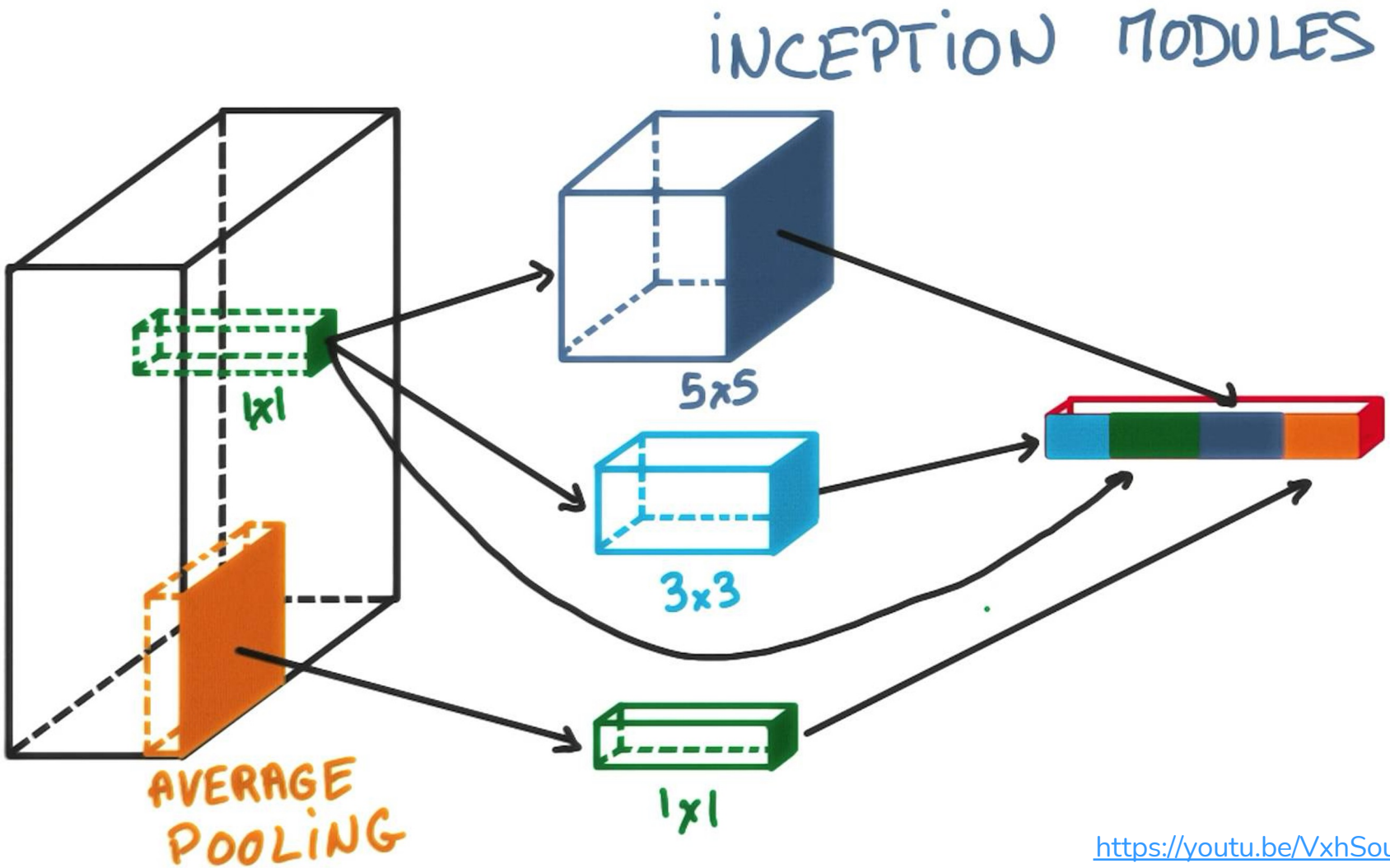


Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

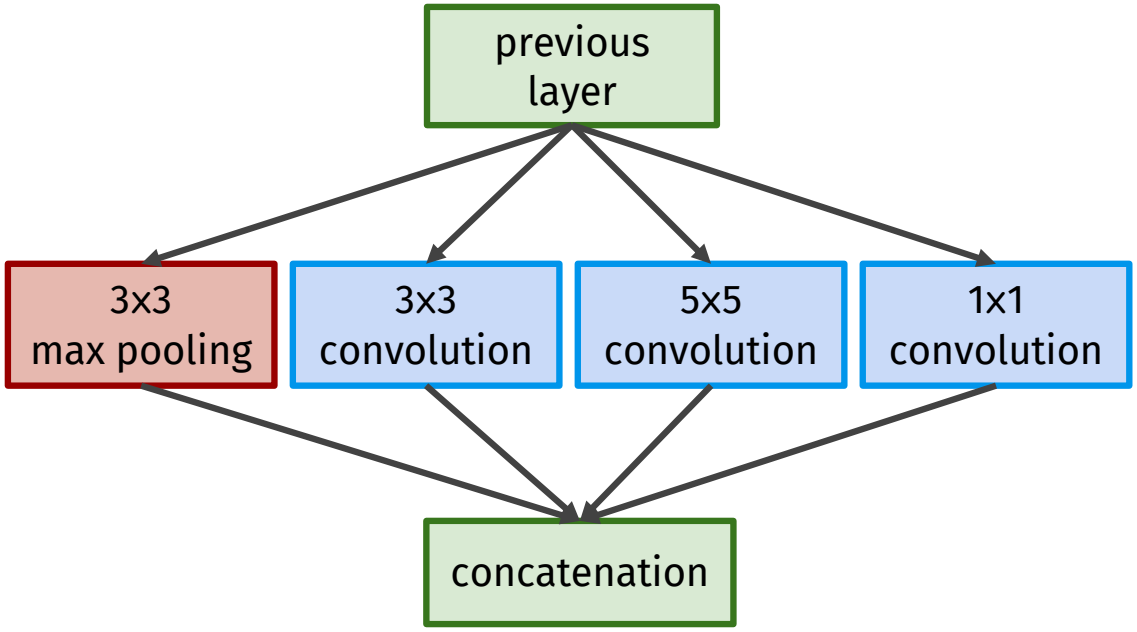






https://youtu.be/VxhSouuSZDY

GoogLeNet [Szegedy et al., 2014]



Naive Inception Module

Apply parallel filters on the input from previous layer:

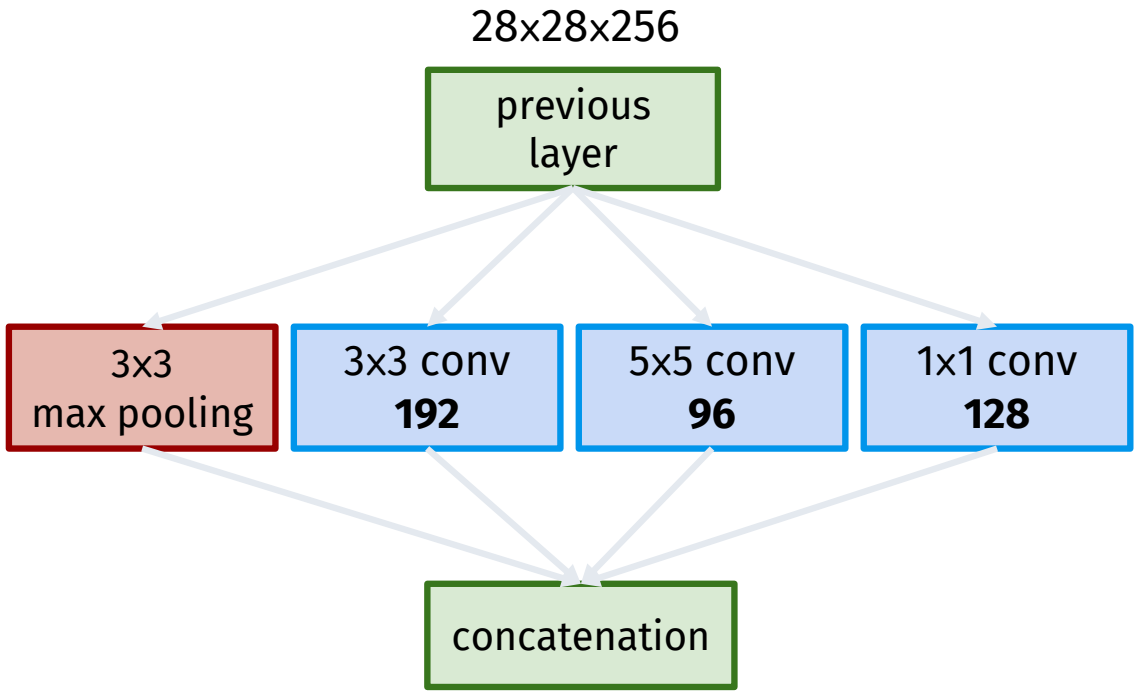
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Q: What is the problem with this? **Computational complexity!**

GoogLeNet [Szegedy et al., 2014]

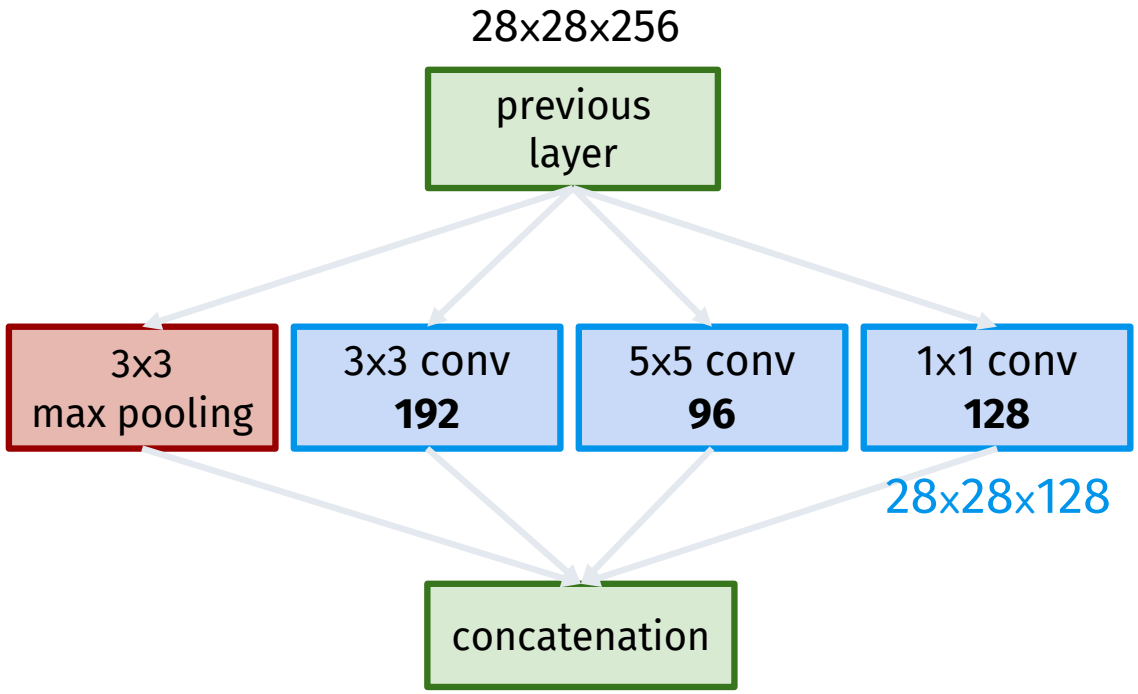
Example: What is the output size of the **1x1 conv, with 128 filters**?



Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

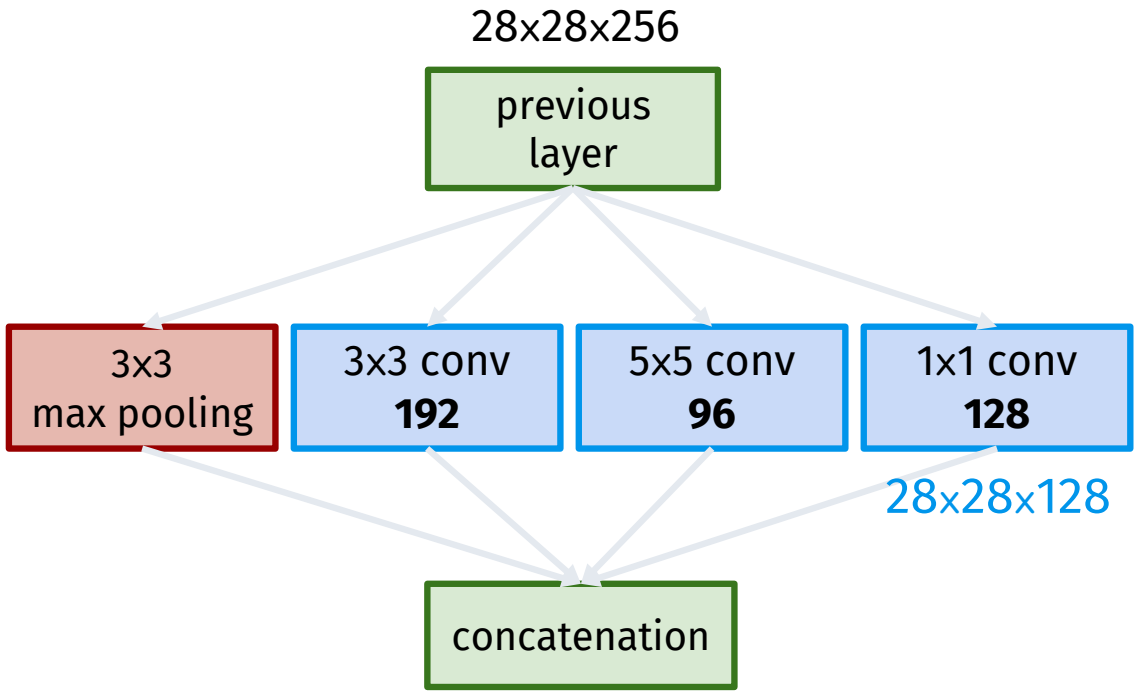
Example: What is the output size of the **1x1 conv, with 128 filters**?



Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

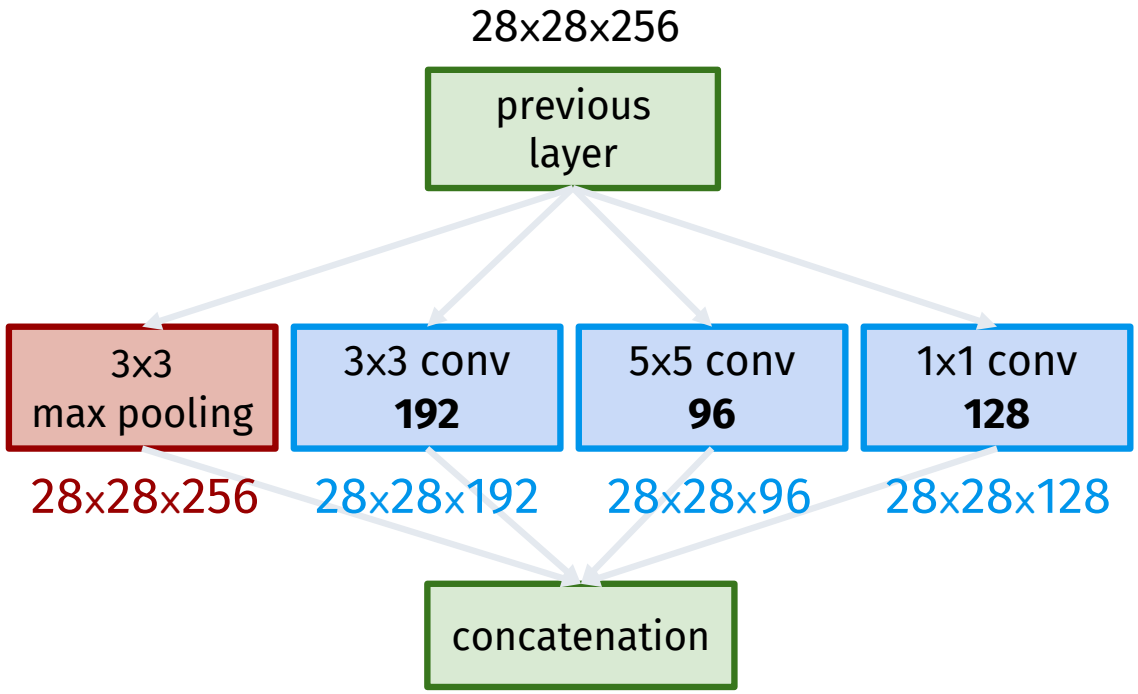
Example: What are the output sizes of all different filter operations?



Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

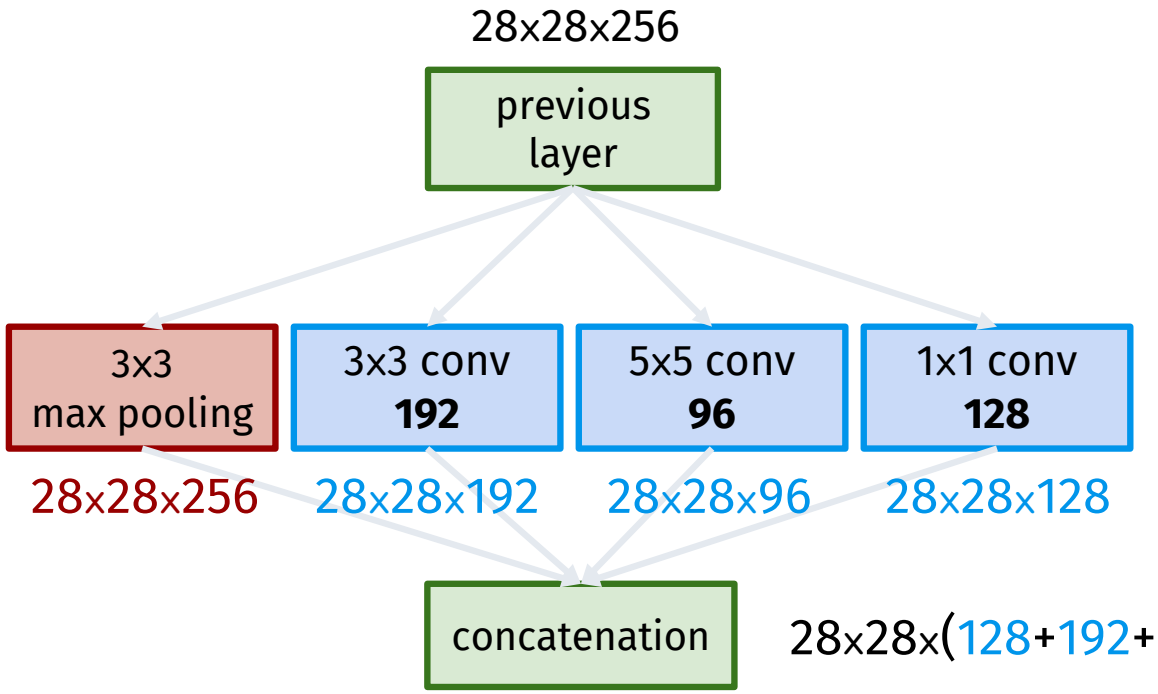
Example: What are the output sizes of all different filter operations?



Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?

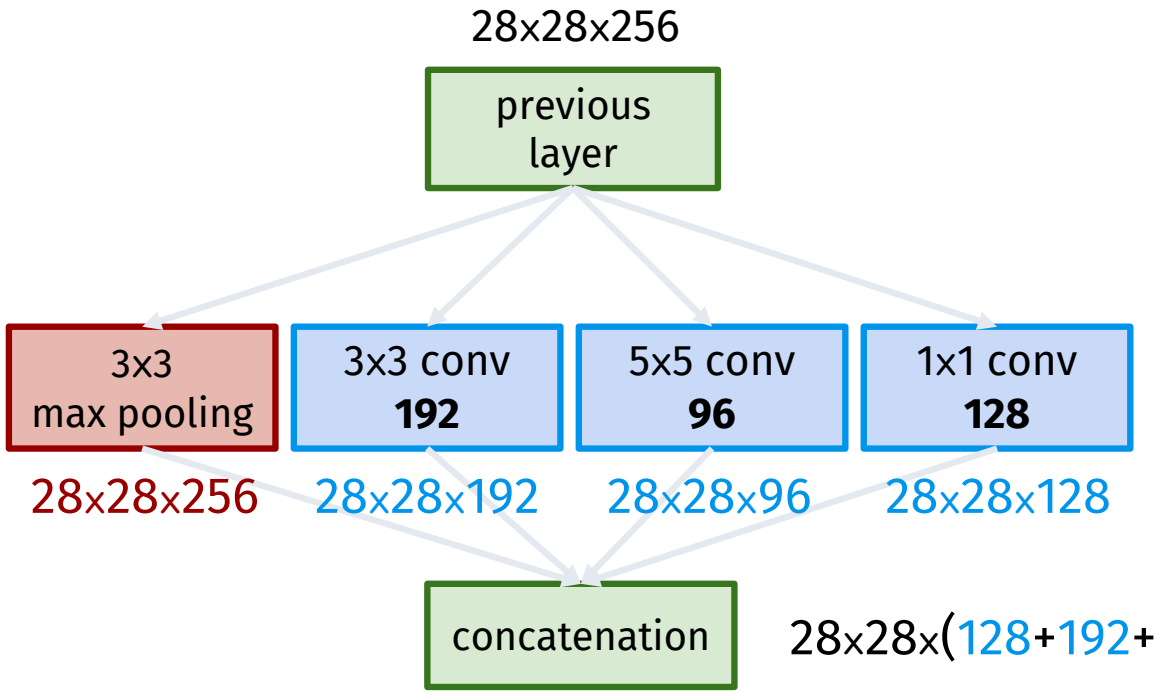


28x28x(128+192+96+256) = 28x28x672

Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?

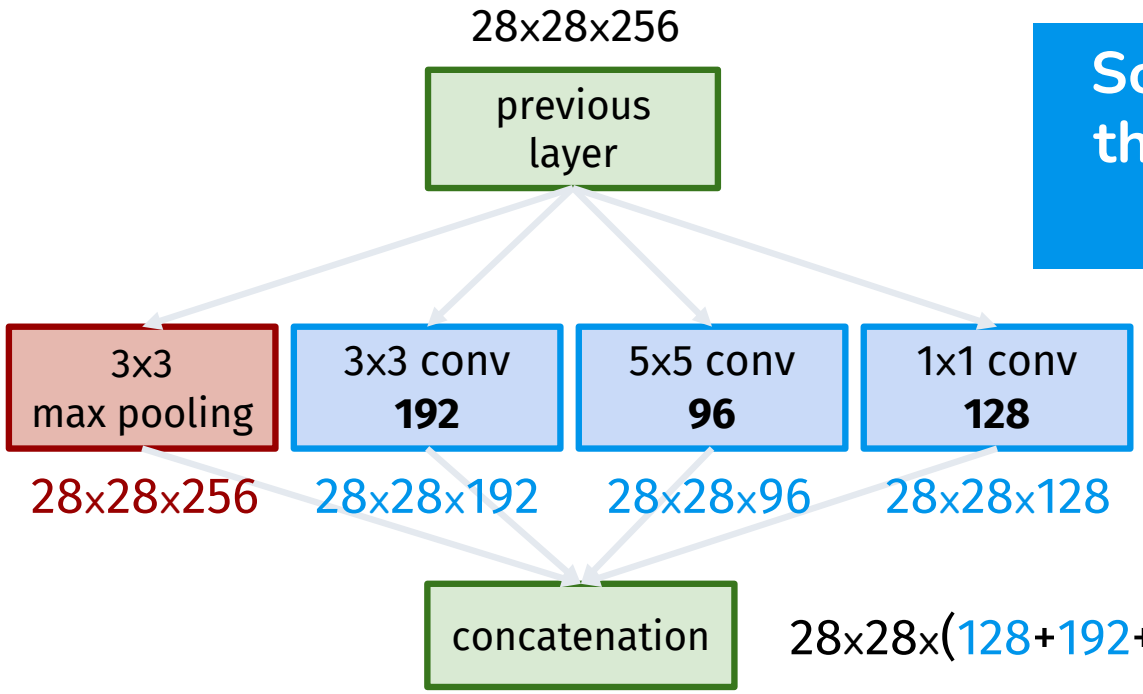


Total: 854M ops!!!

28x28x(128+192+96+256) = 28x28x672

GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?



Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature depth

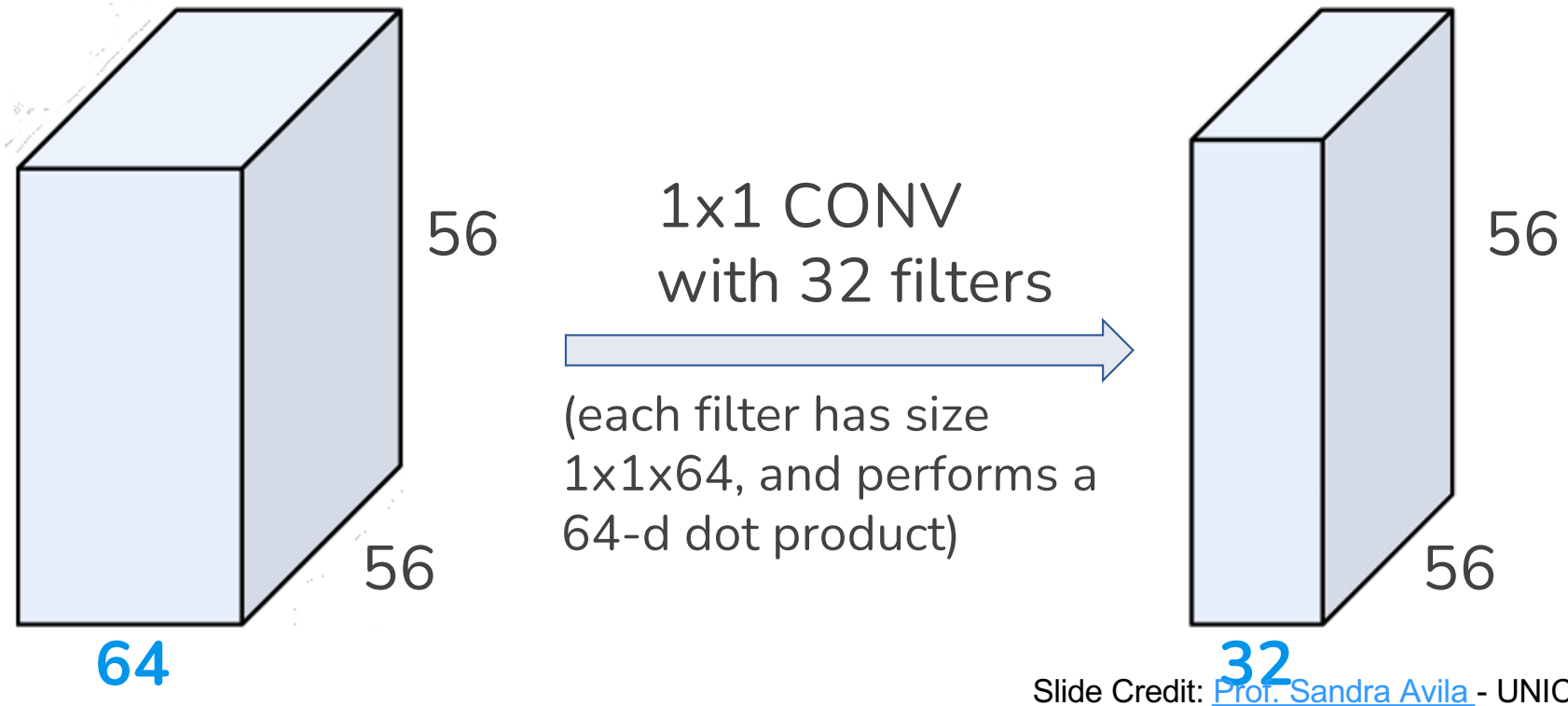
Total: 854M ops!!!

28x28x(128+192+96+256) = 28x28x672

Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

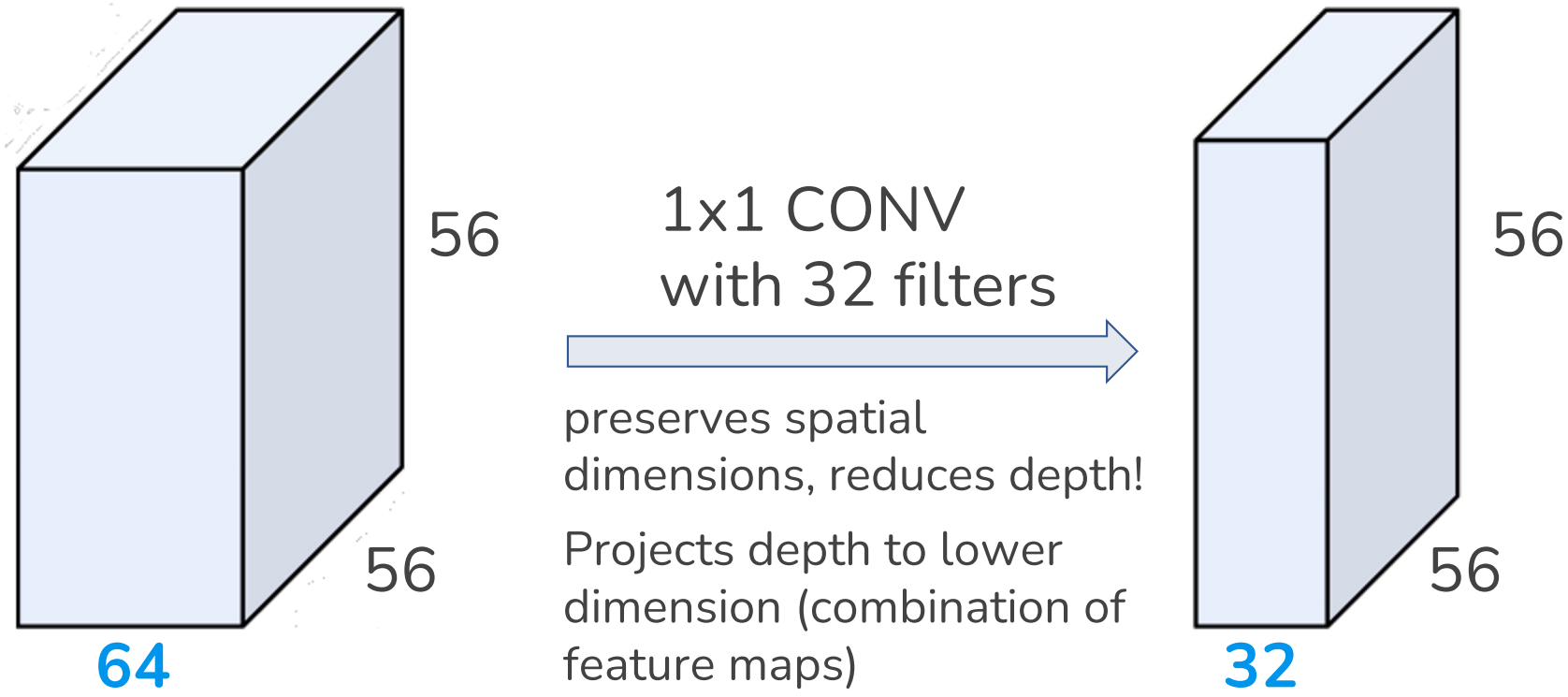
Reminder: 1x1 convolutions



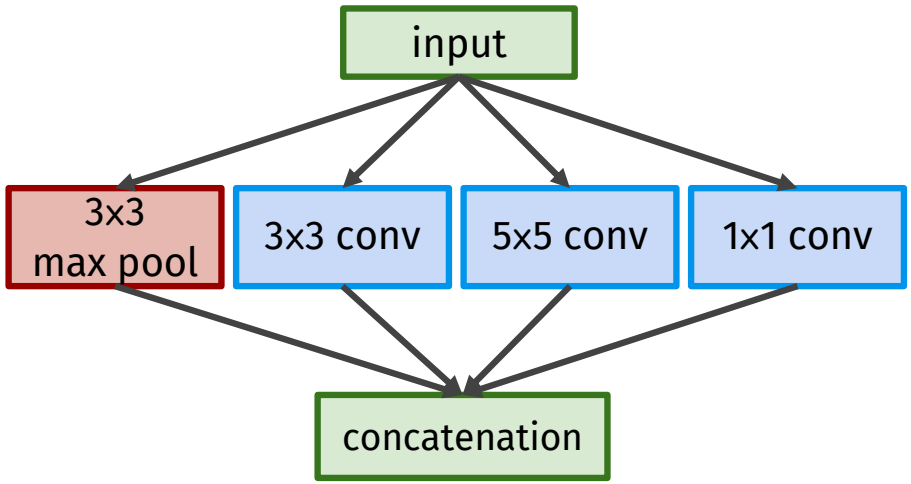
Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

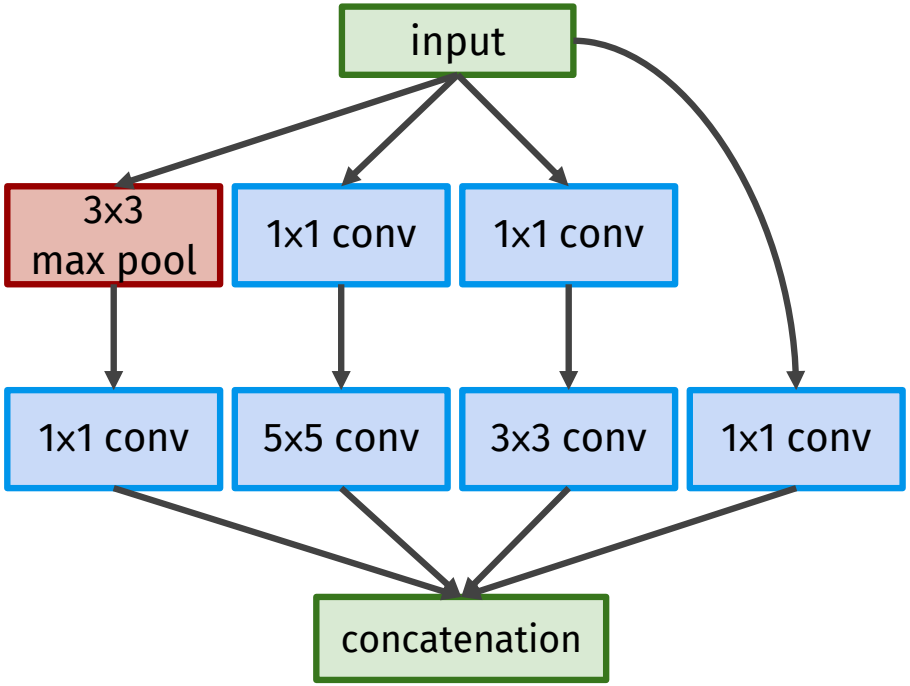
Reminder: 1x1 convolutions



GoogLeNet [Szegedy et al., 2014]



Naive Inception Module

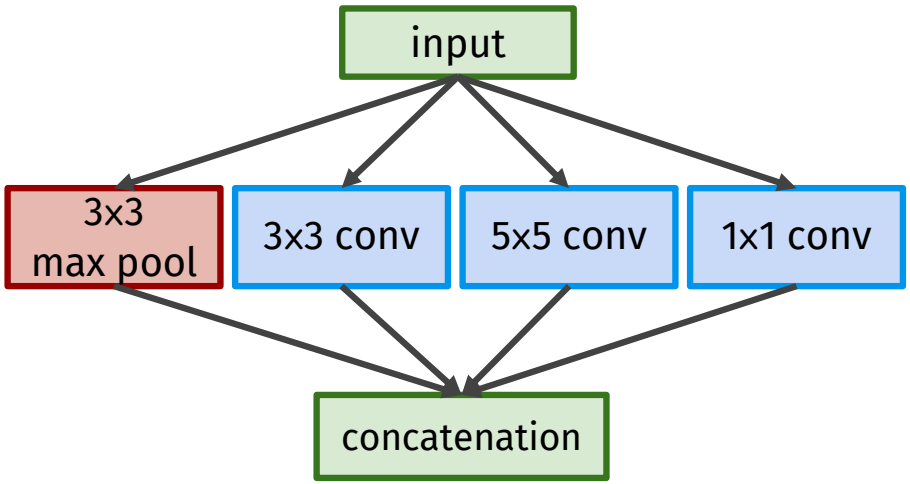


Inception Module

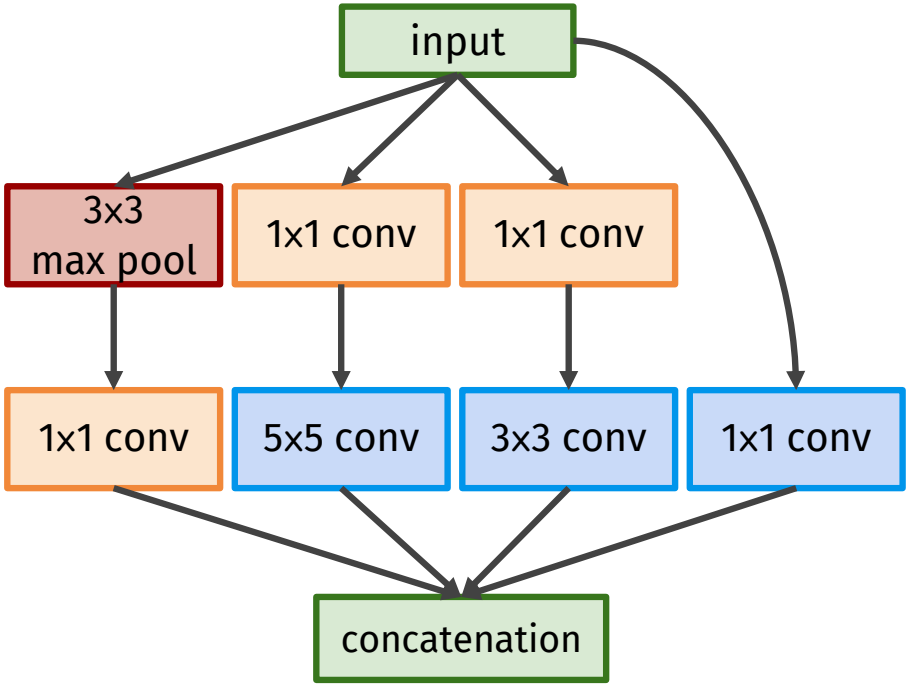
Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

1x1 conv
"bottleneck" layers

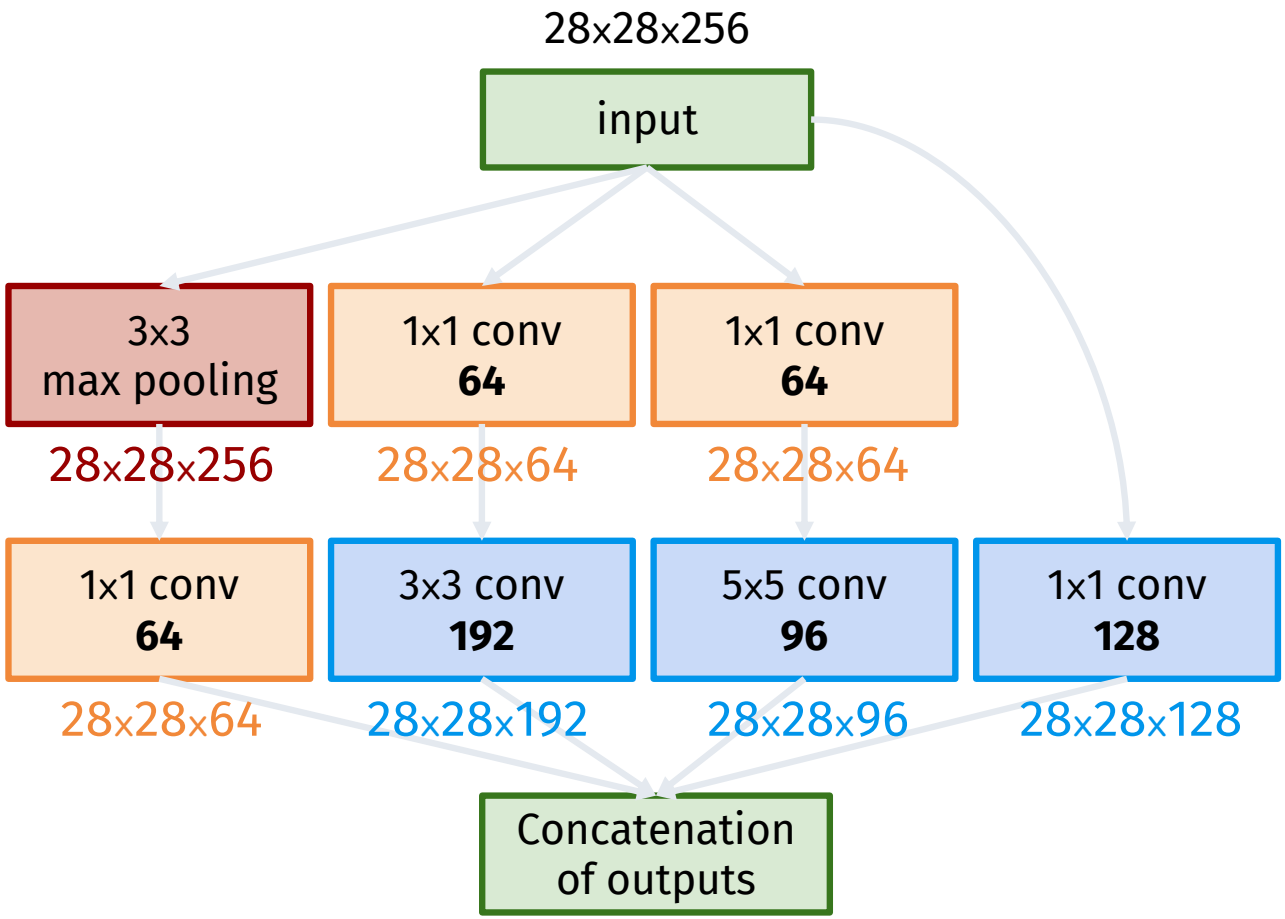


Naive Inception Module



Inception Module

Slide Credit: Prof. Sandra Avila - UNICAMP

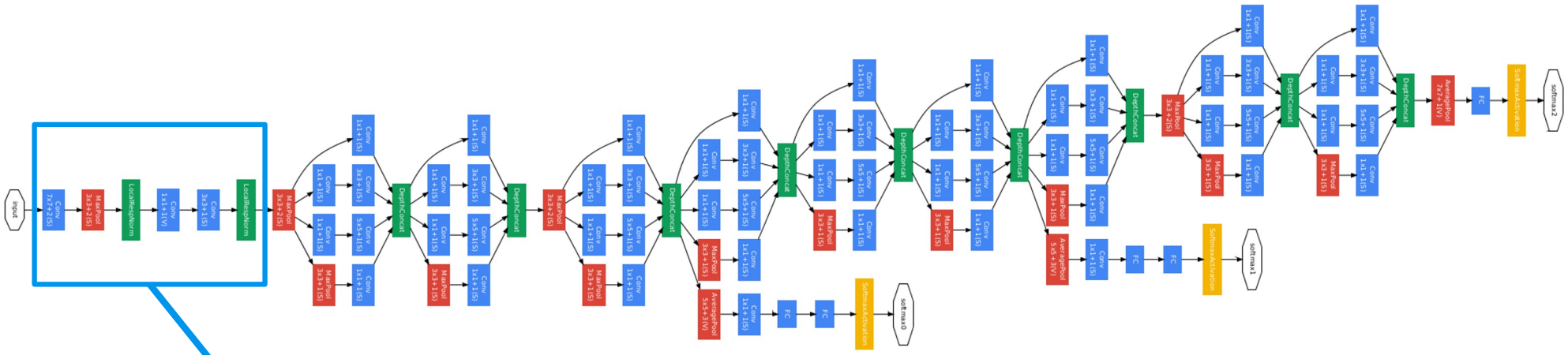


28x28x(128+192+96+64) = 28x28x480

358M ops!!!
(vs. 854M)

Slide Credit: Prof. Sandra Avila - UNICAMP

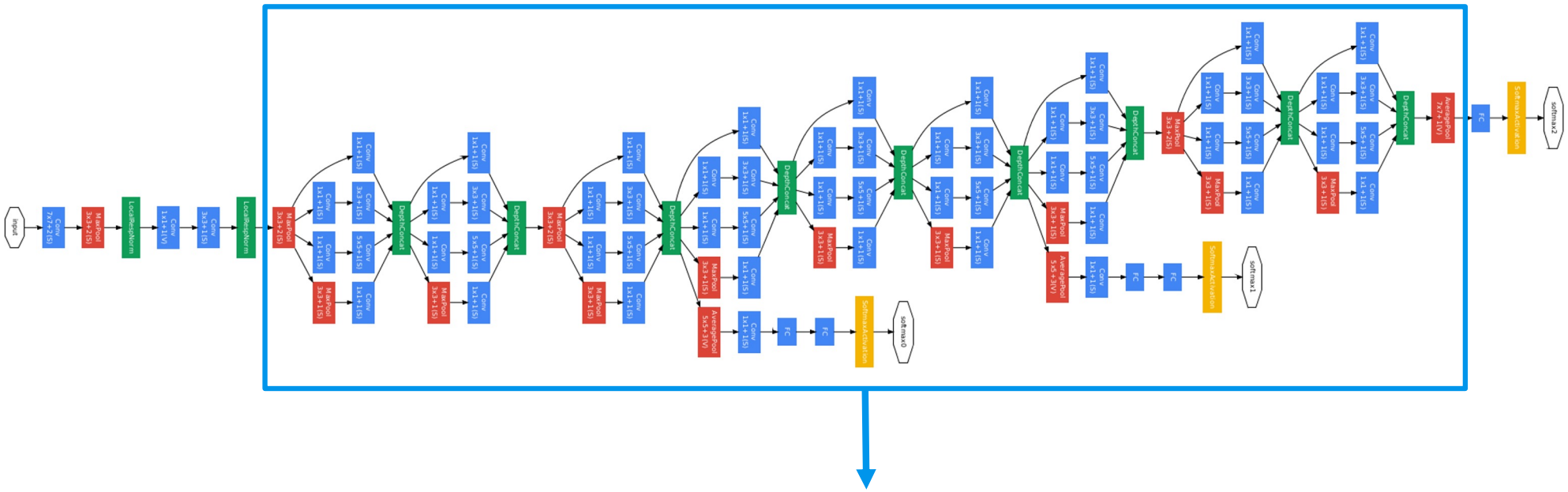
GoogLeNet [Szegedy et al., 2014]



Conv-Pool 2x Conv-Pool

Slide Credit: Prof. Sandra Avila - UNICAMP

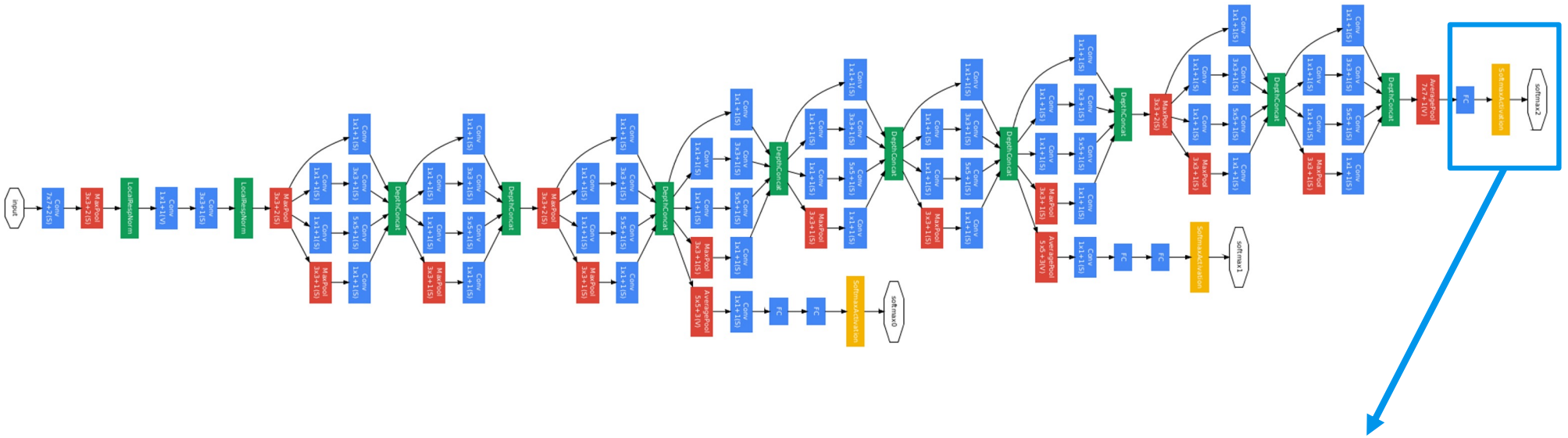
GoogLeNet [Szegedy et al., 2014]



Stacked Inception Modules

Slide Credit: Prof. Sandra Avila - UNICAMP

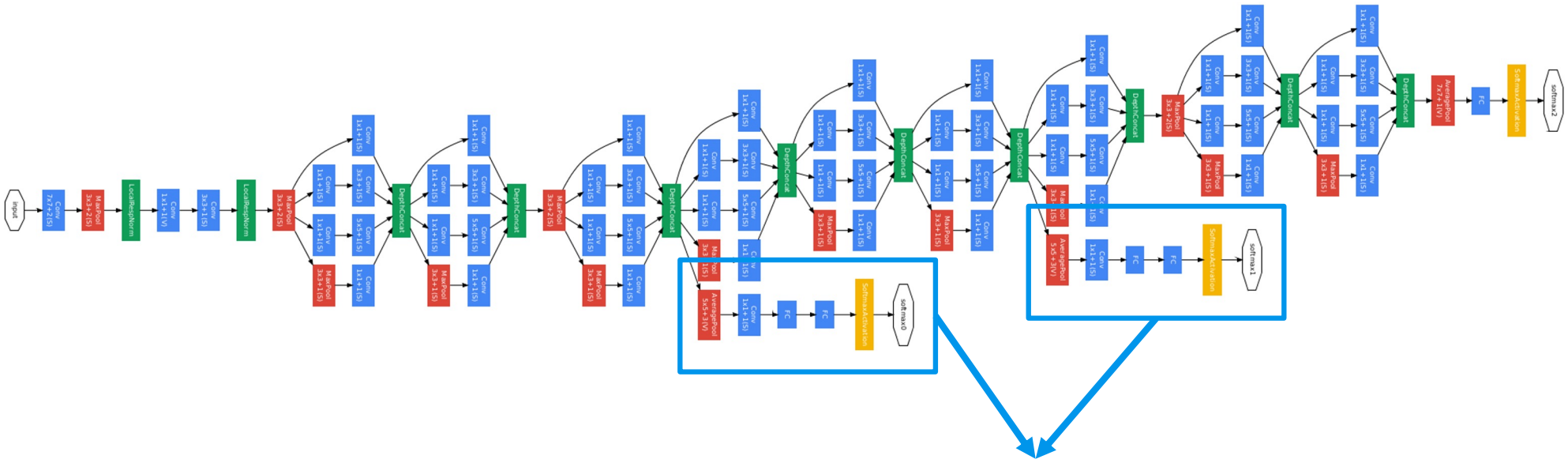
GoogLeNet [Szegedy et al., 2014]



Classifier Output

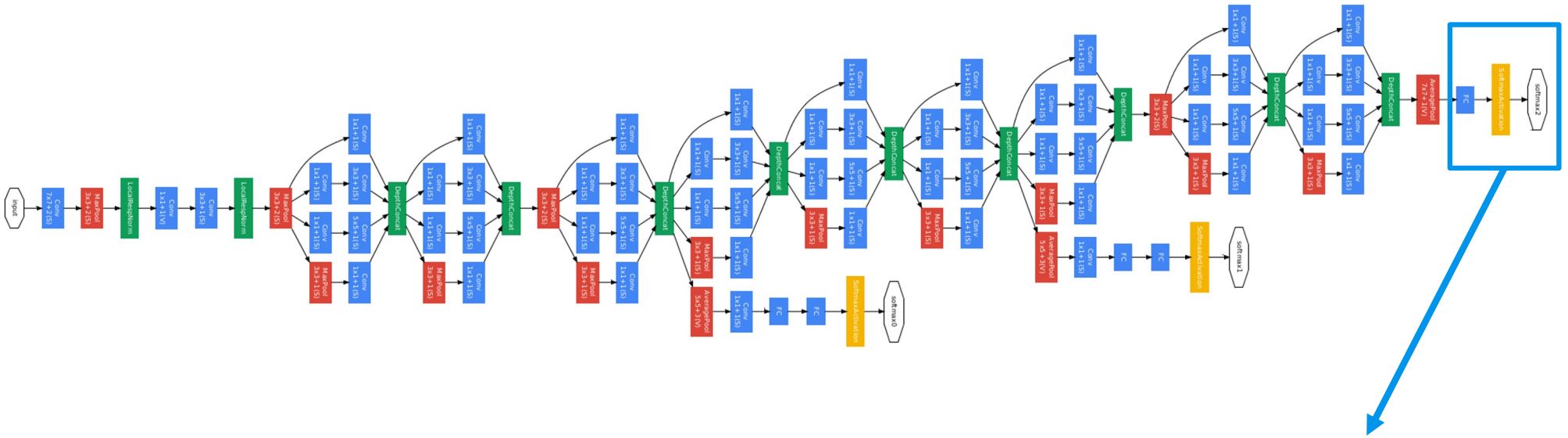
Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]



Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]



The total loss function is a weighted sum of the auxiliary loss and the real loss.

```
total_loss = real_loss + 0.3*aux_loss_1 + 0.3*aux_loss_2
```

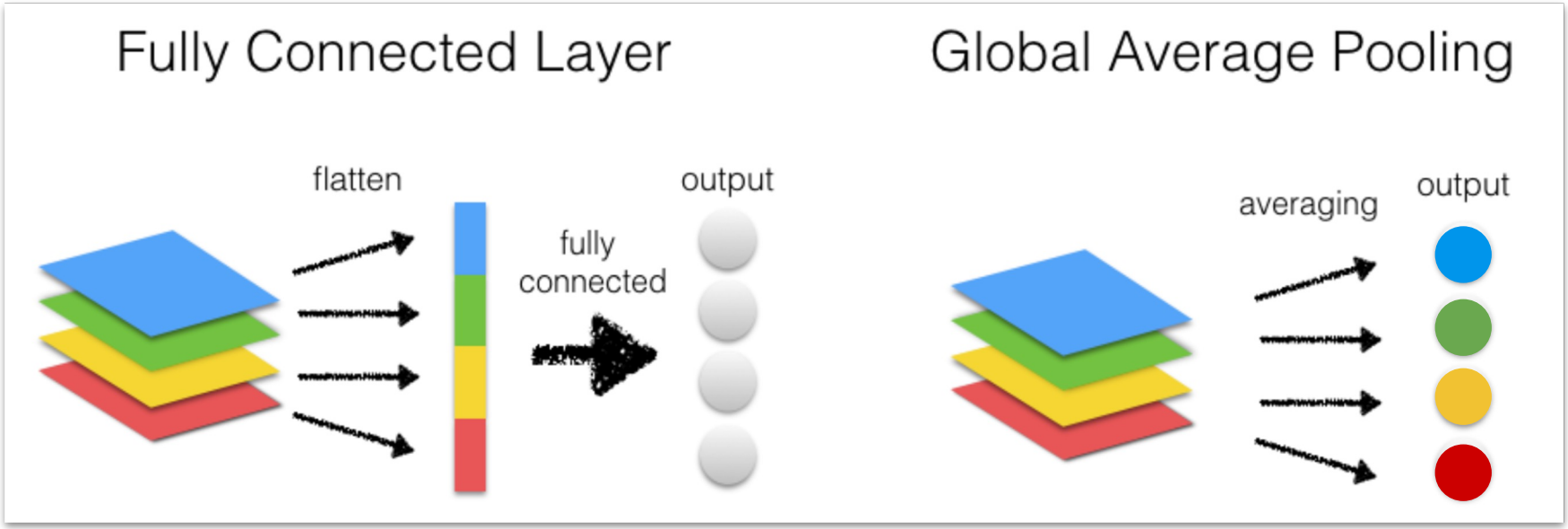
Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]

- GoogLeNet has **9 inception modules** stacked linearly.
- It is **22 layers deep** (27, including the pooling layers).
- It uses **global average pooling** at the end of the last inception module.
- GoogLeNet = Inception v1
- Inception v2, v3, v4, Inception-ResNet v1, v2:
  https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet [Szegedy et al., 2014]



Slide Credit: Prof. Sandra Avila - UNICAMP

CNN Architectures

- CNN Architectures
  - LeNet (1998)
  - AlexNet (2012)
  - ZFNet (2013)
  - VGGNet (2014)
  - GoogLeNet (2014)
  - **ResNet (2015)**

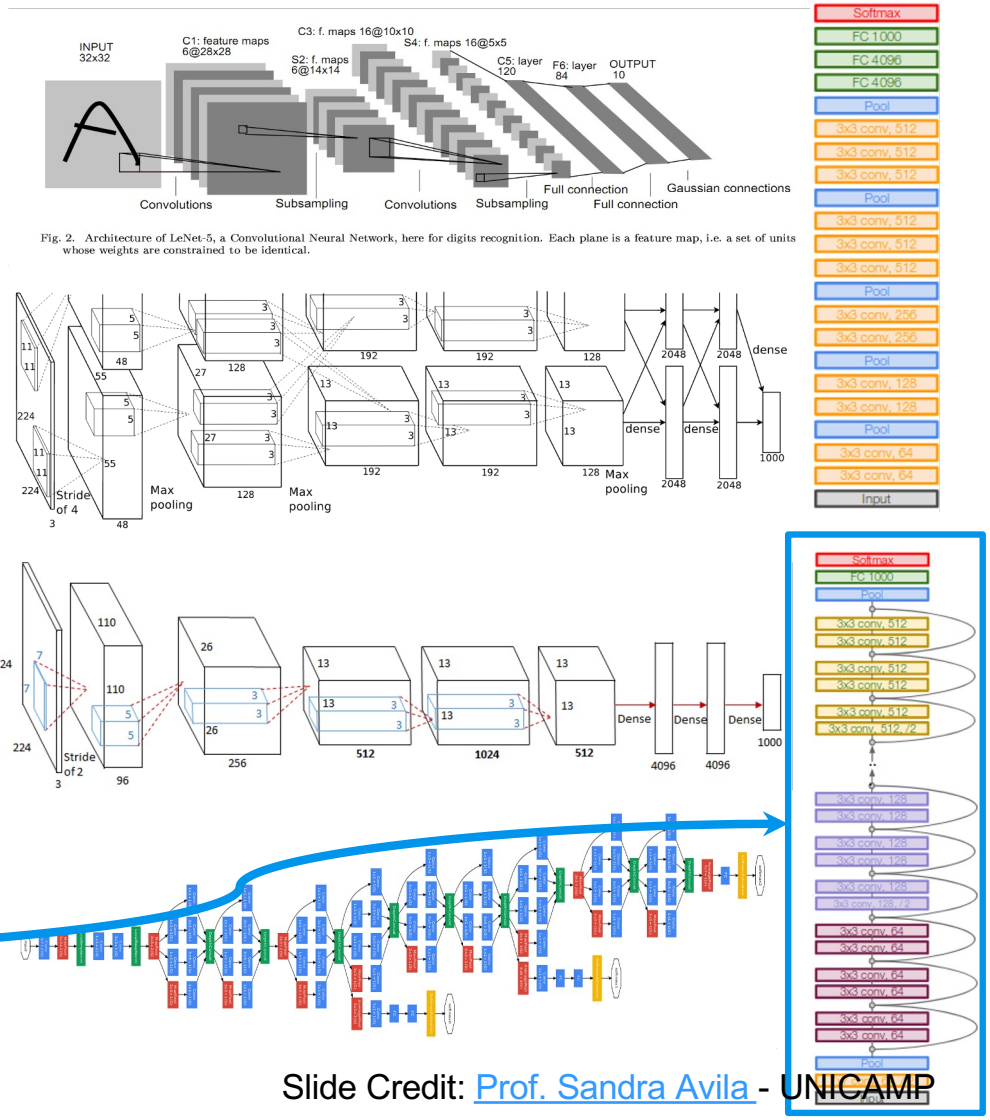
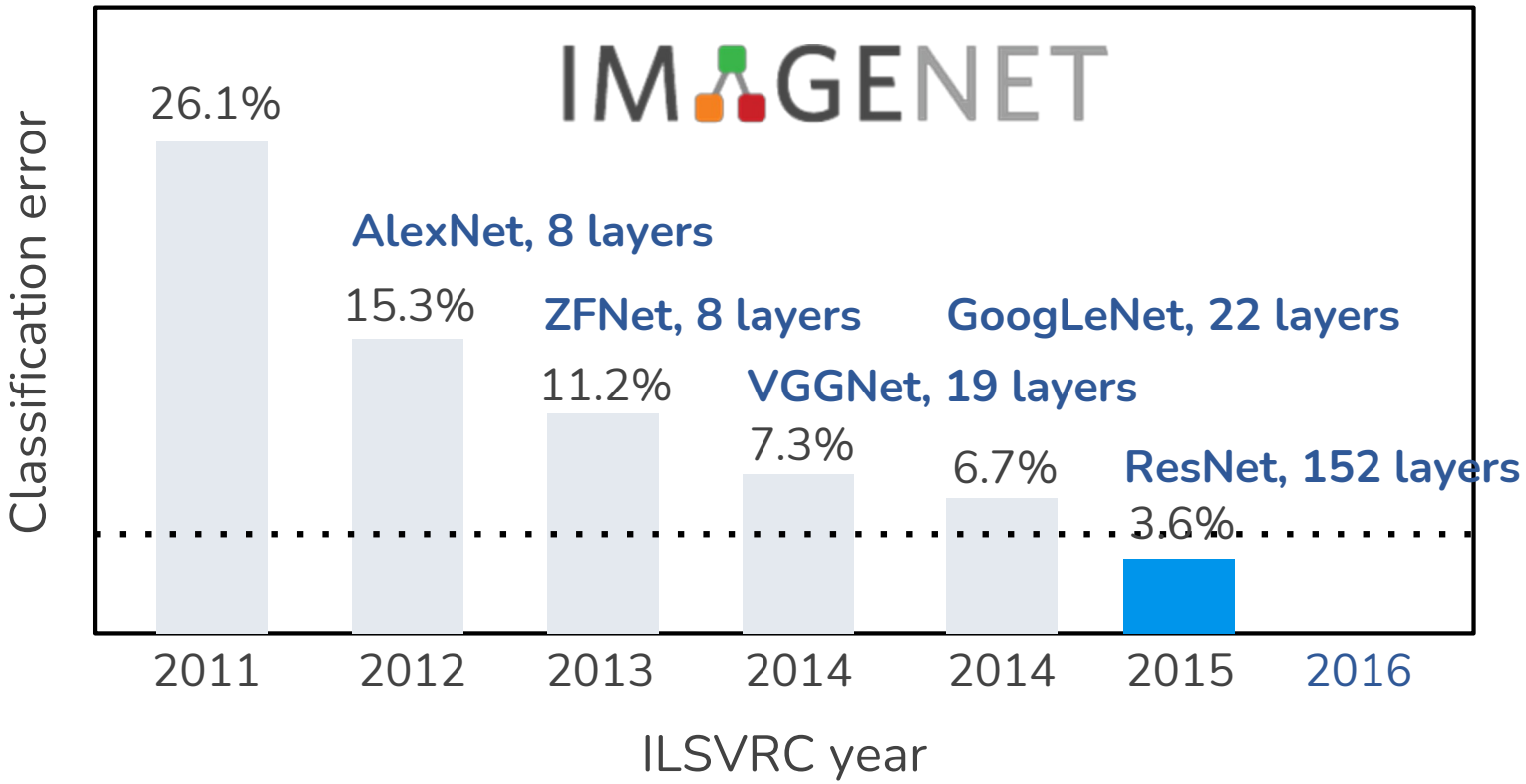


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Slide Credit: Prof. Sandra Avila - UNICAMP



"Deep Residual Learning for Image Recognition", CVPR 2016, https://arxiv.org/pdf/1512.03385

Slide Credit: Prof. Sandra Avila - UNICAMP

ResNet [He et al., 2015]

ResNet @ ILSVRC & COCO 2015 Competitions

1st place in ALL five main tracks

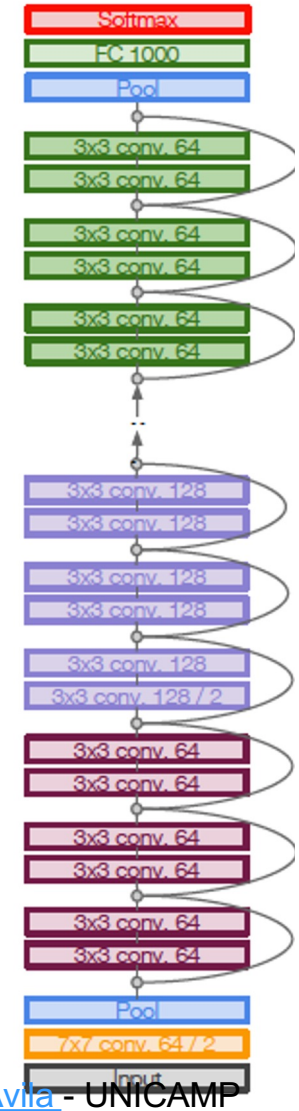
- ImageNet Classification: “Ultra-deep” 152-layer nets
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

Slide Credit: Prof. Sandra Avila - UNICAMP

ResNet [He et al., 2015]

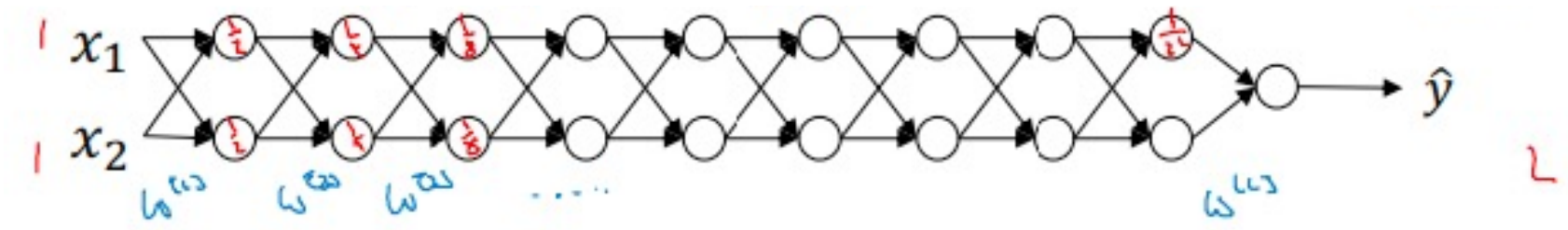
Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)



Slide Credit: Prof. Sandra Avila - UNICAMP

ResNet [He et al., 2015] – Vanishing Gradient



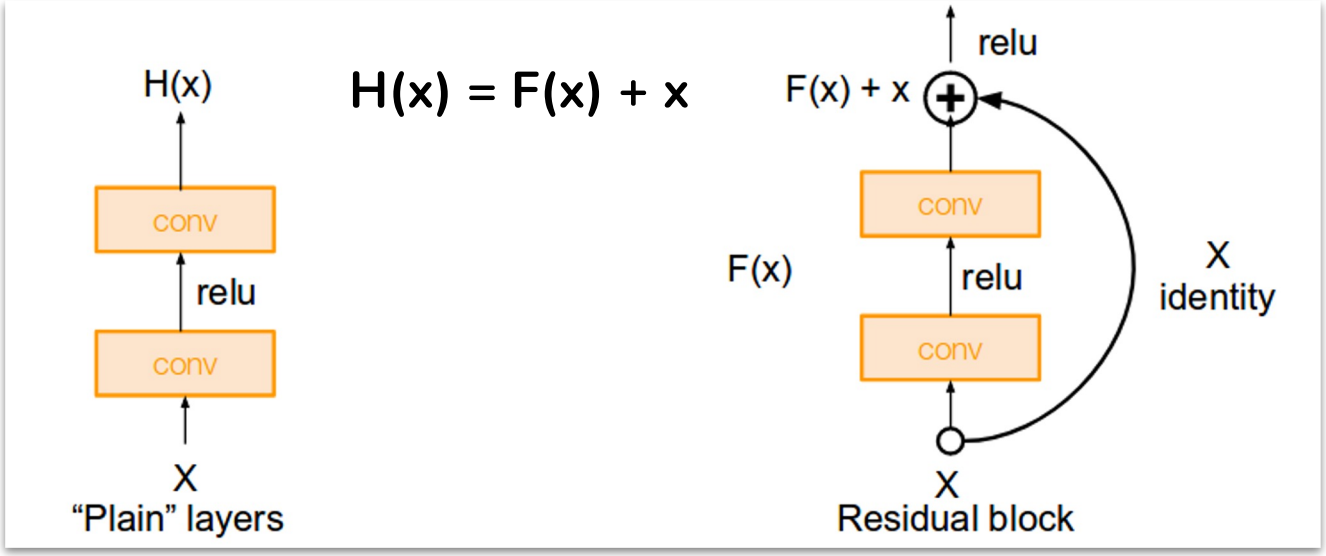
Overflow/Underflow

Vanishing/exploding gradients

Adapted from Deep Coursera Specialization [Andrew Ng]

ResNet [He et al., 2015]

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

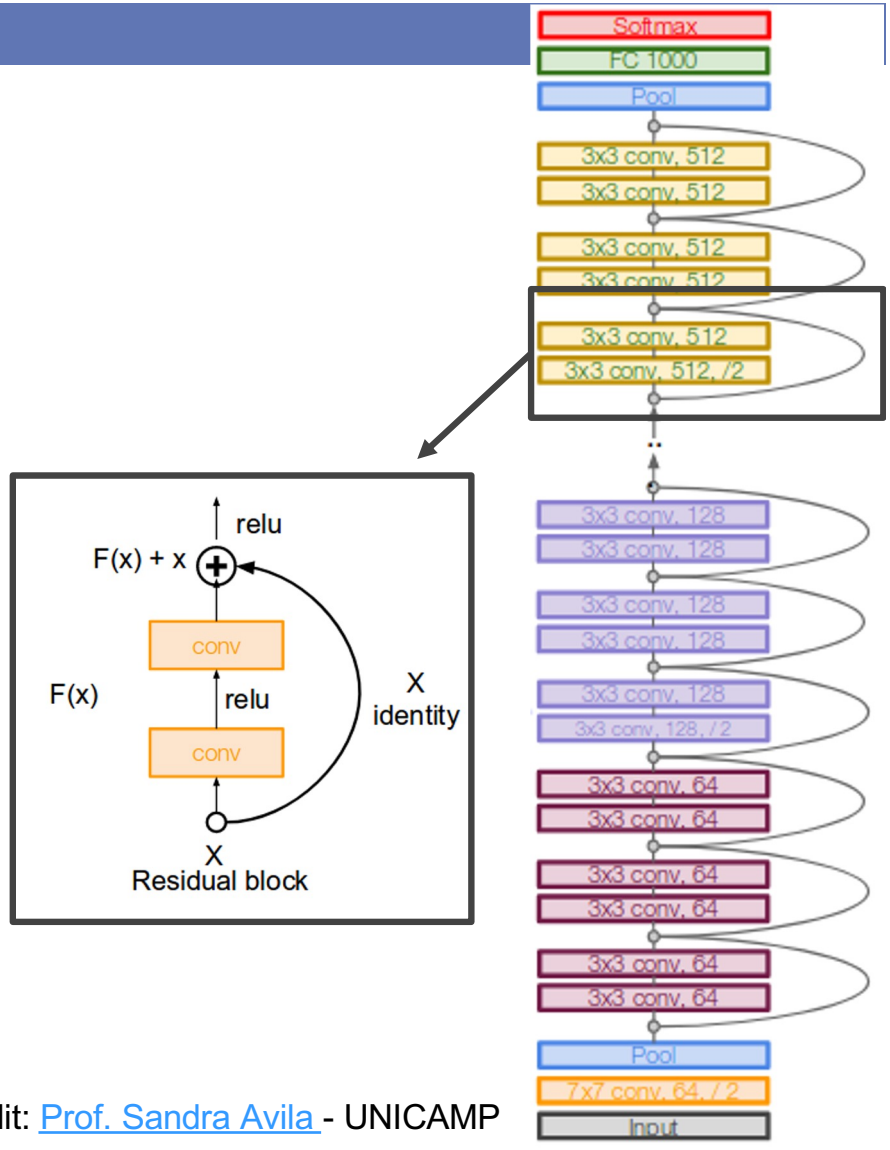


Slide Credit: Prof. Sandra Avila - UNICAMP

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
- No FC layers at the end (only FC 1000 to output classes)



Slide Credit: Prof. Sandra Avila - UNICAMP

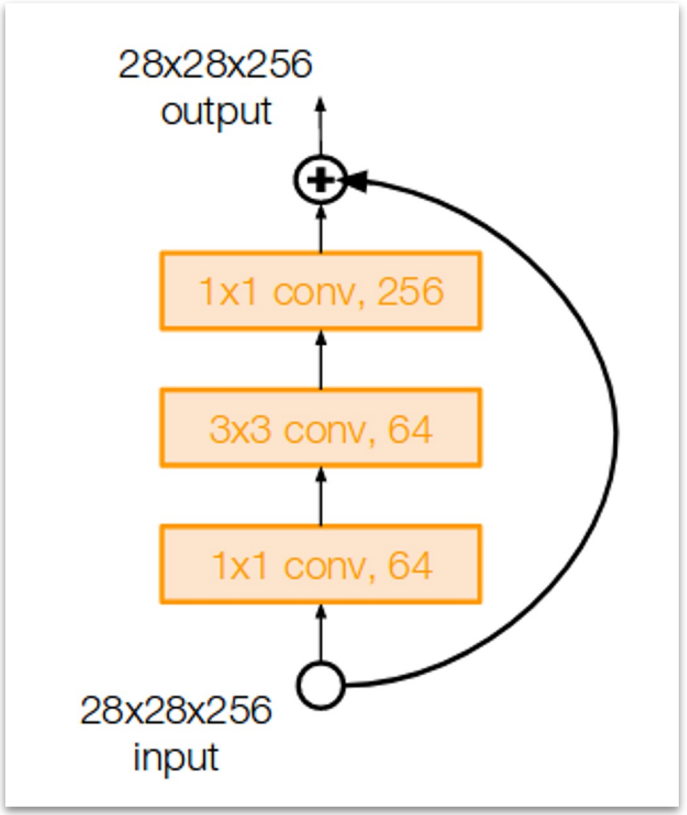
ResNet [He et al., 2015]

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

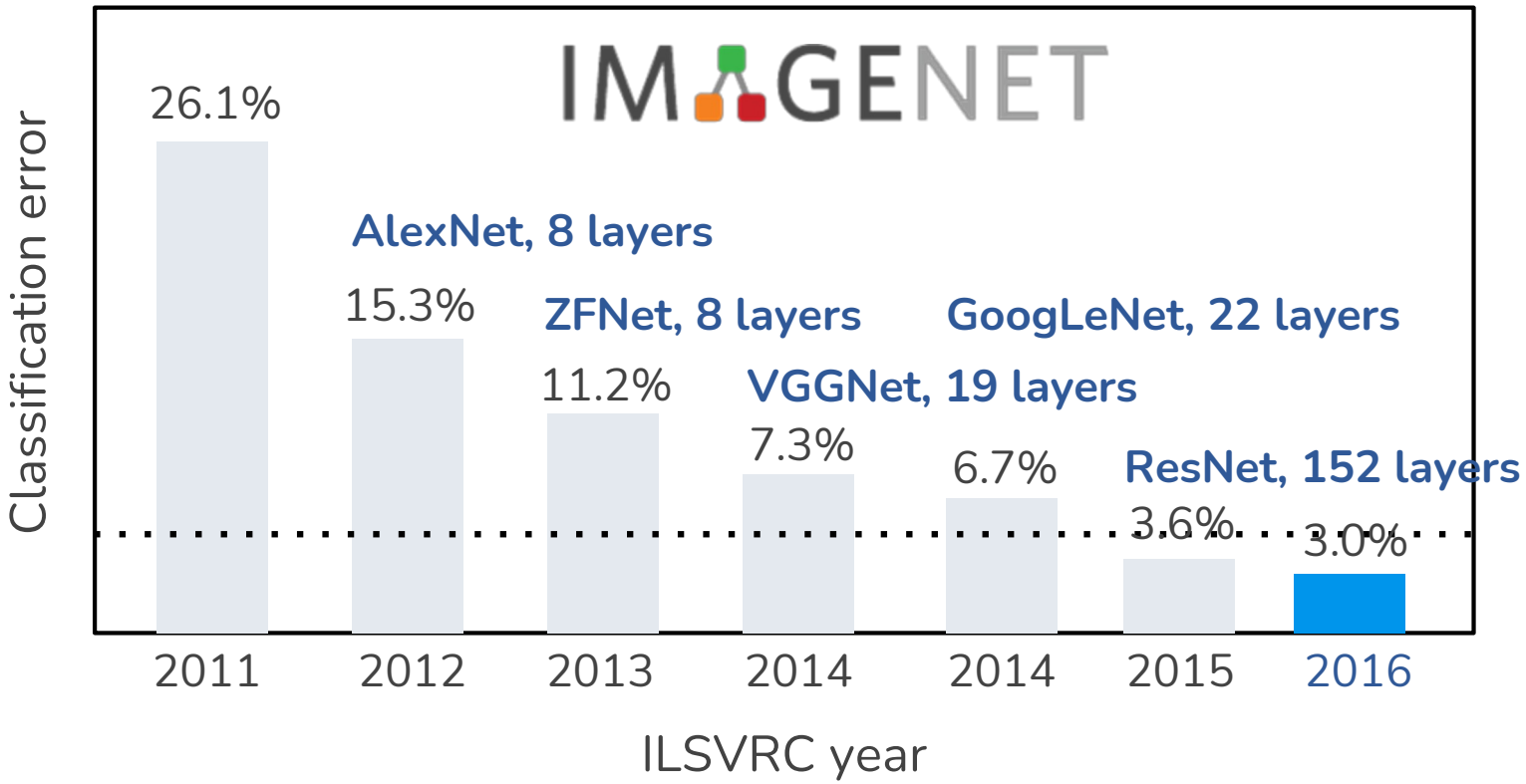
1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64



Slide Credit: Prof. Sandra Avila - UNICAMP



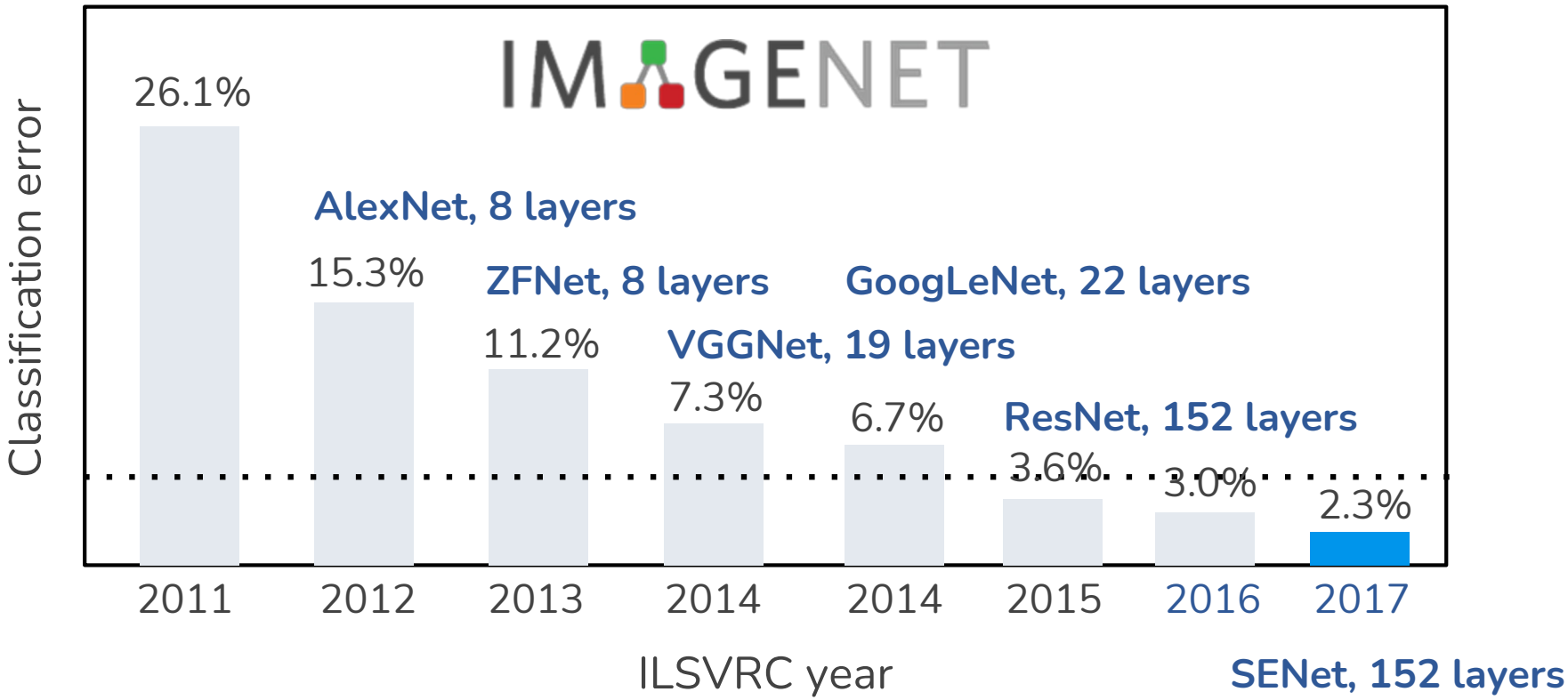
"Good Practices for Deep Feature Fusion", ECCV 2016, http://image-net.org/challenges/talks/2016/Trimps-Soushen@ILSVRC2016.pdf (Slides only)

Good Practices for Deep Feature Fusion
[Shao et al., 2016]

- Training
  - Multi-scale augmentation & large mini-batch size
- Testing
  - Multi-scale & flip & dense fusion

	Error (%)
Inception-v3	4.20
Inception-v4	4.01
Inception-ResNet-v2	3.52
ResNet-200	4.26
Wrm-68-3	4.65
Fusion (Test)	**2.99**

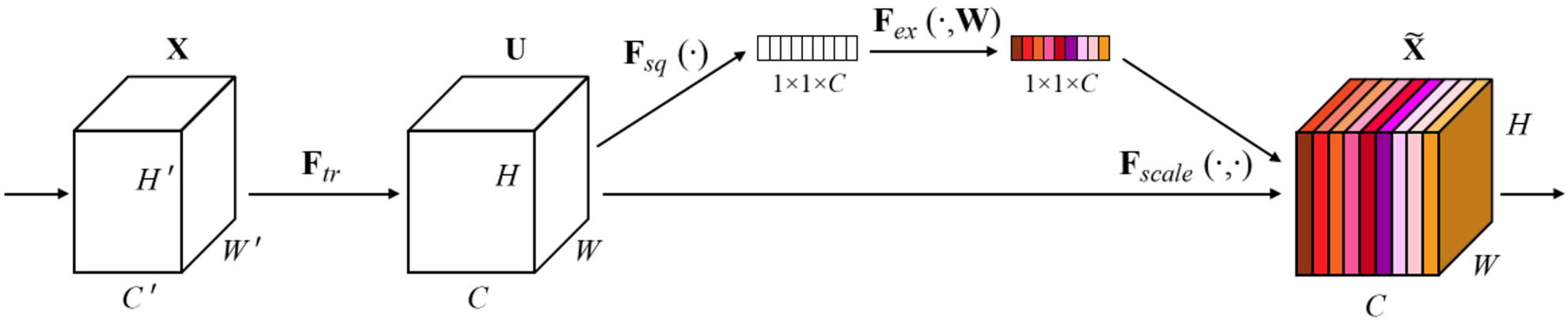
Slide Credit: Prof. Sandra Avila - UNICAMP



"Squeeze-and-Excitation Networks", CVPR 2018, https://arxiv.org/pdf/1709.01507

Slide Credit: Prof. Sandra Avila - UNICAMP

SENets [Hu et al. 2017]



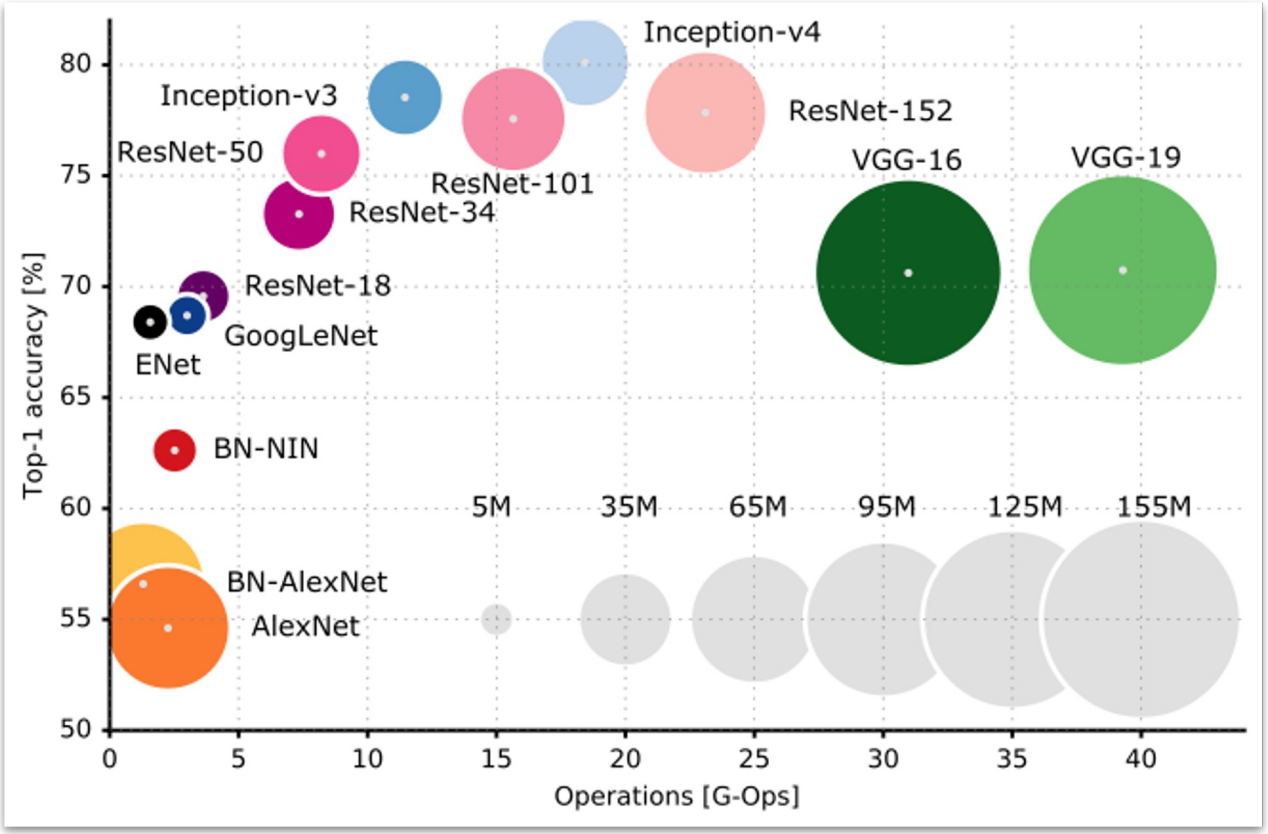
Add a “feature recalibration” module that **learns** to **adaptively reweight feature maps**.

Slide Credit: Prof. Sandra Avila - UNICAMP

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

"Squeeze-and-Excitation Networks", CVPR 2018, https://arxiv.org/pdf/1709.01507

The size of the blobs is proportional to the number of network parameters.

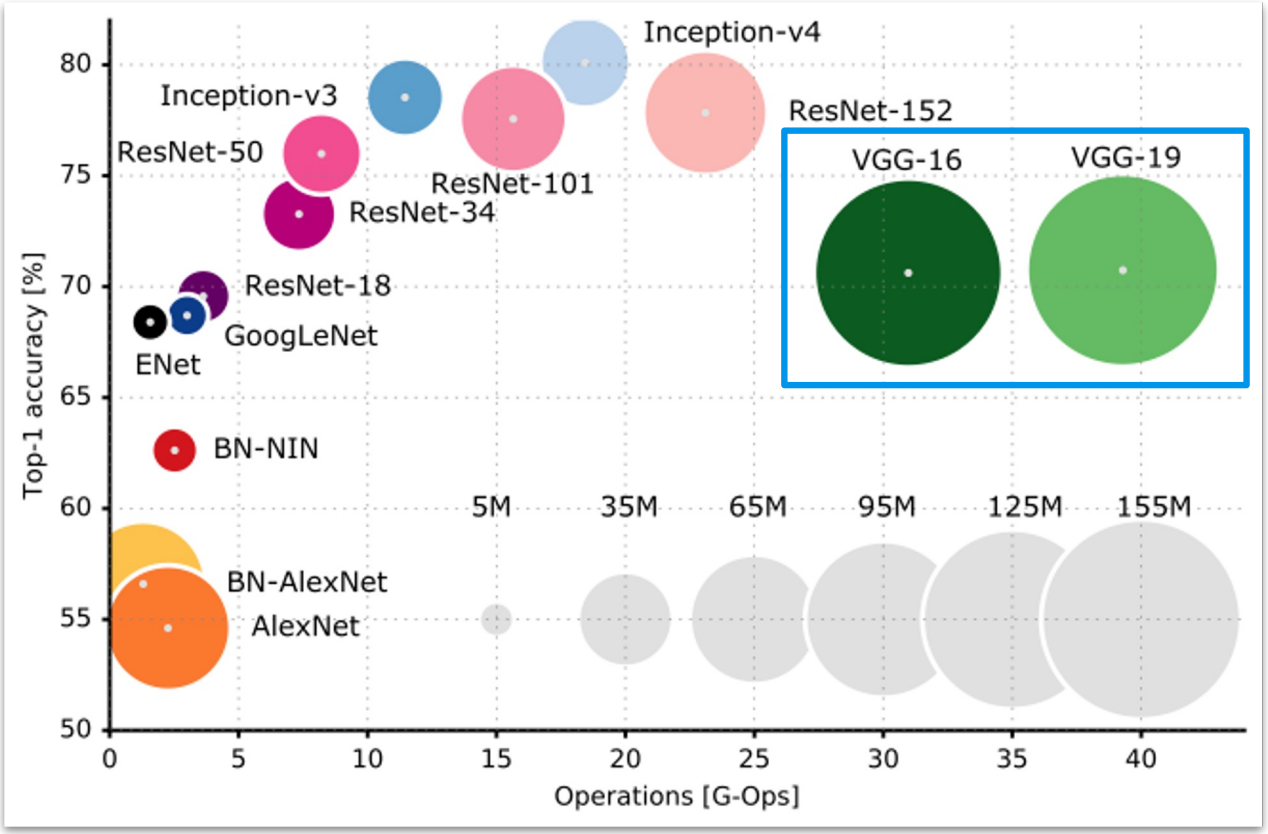


https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba

Slide Credit: Prof. Sandra Avila - UNICAMP

VGG: Highest memory, most operations

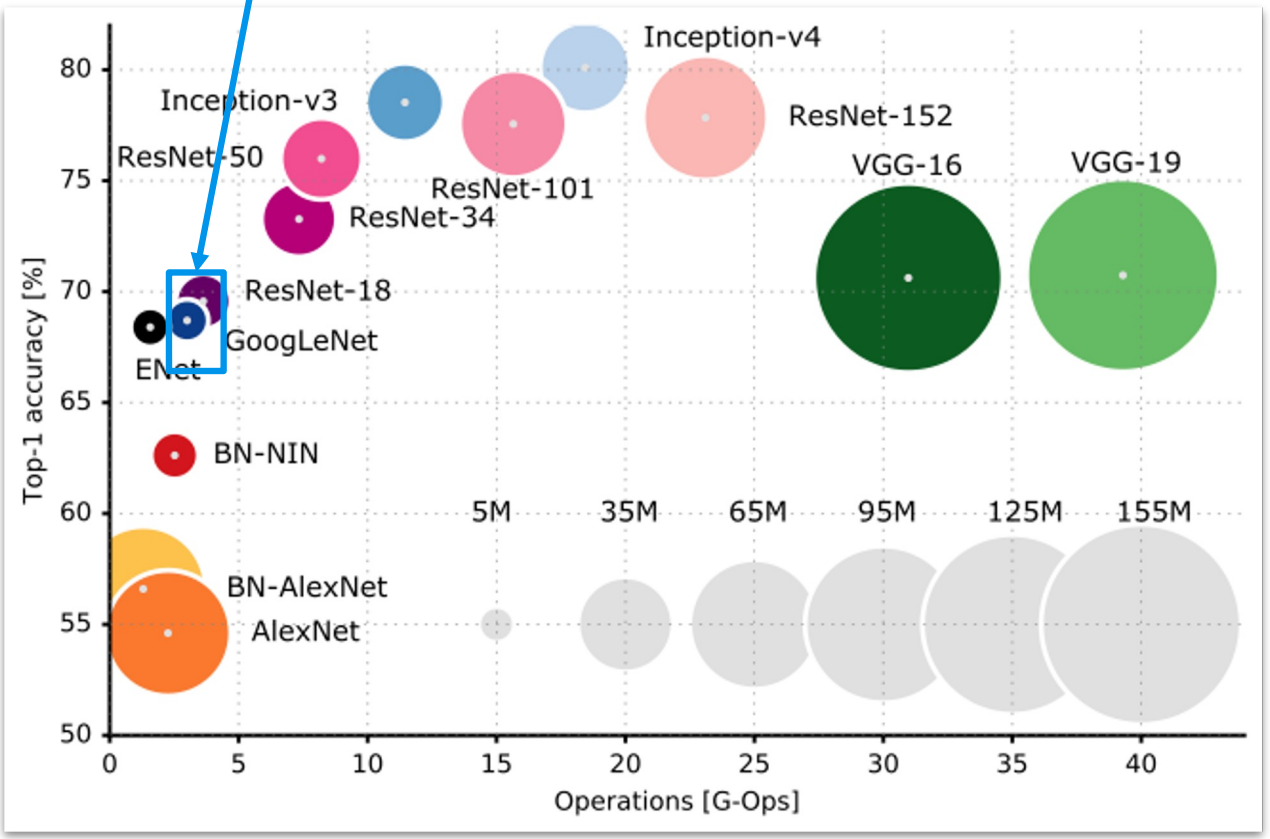
The size of the blobs is proportional to the number of network parameters.



https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba

Slide Credit: Prof. Sandra Avila - UNICAMP

GoogLeNet: most efficient



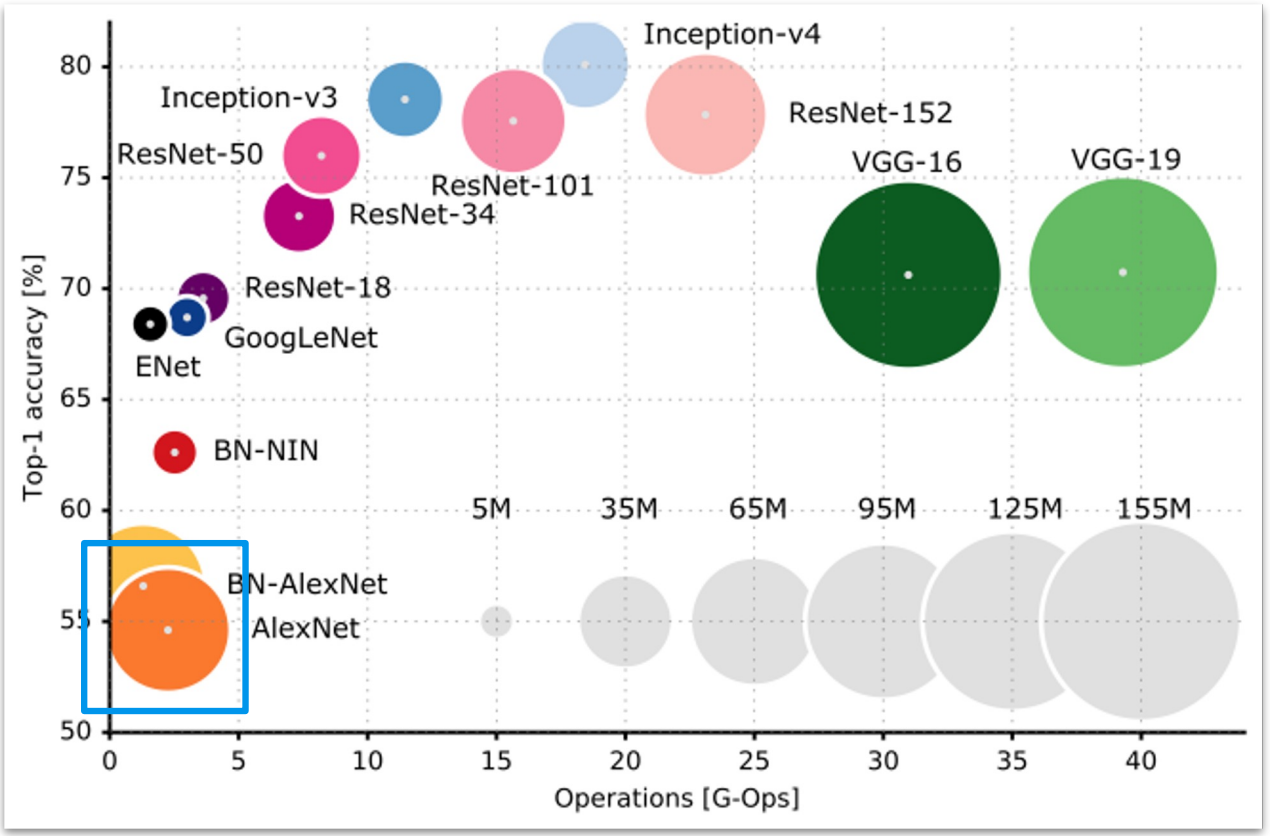
https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba

The size of the blobs is proportional to the number of network parameters.

Slide Credit: Prof. Sandra Avila - UNICAMP

AlexNet: Smaller compute, still memory heavy, lower accuracy

The size of the blobs is proportional to the number of network parameters.

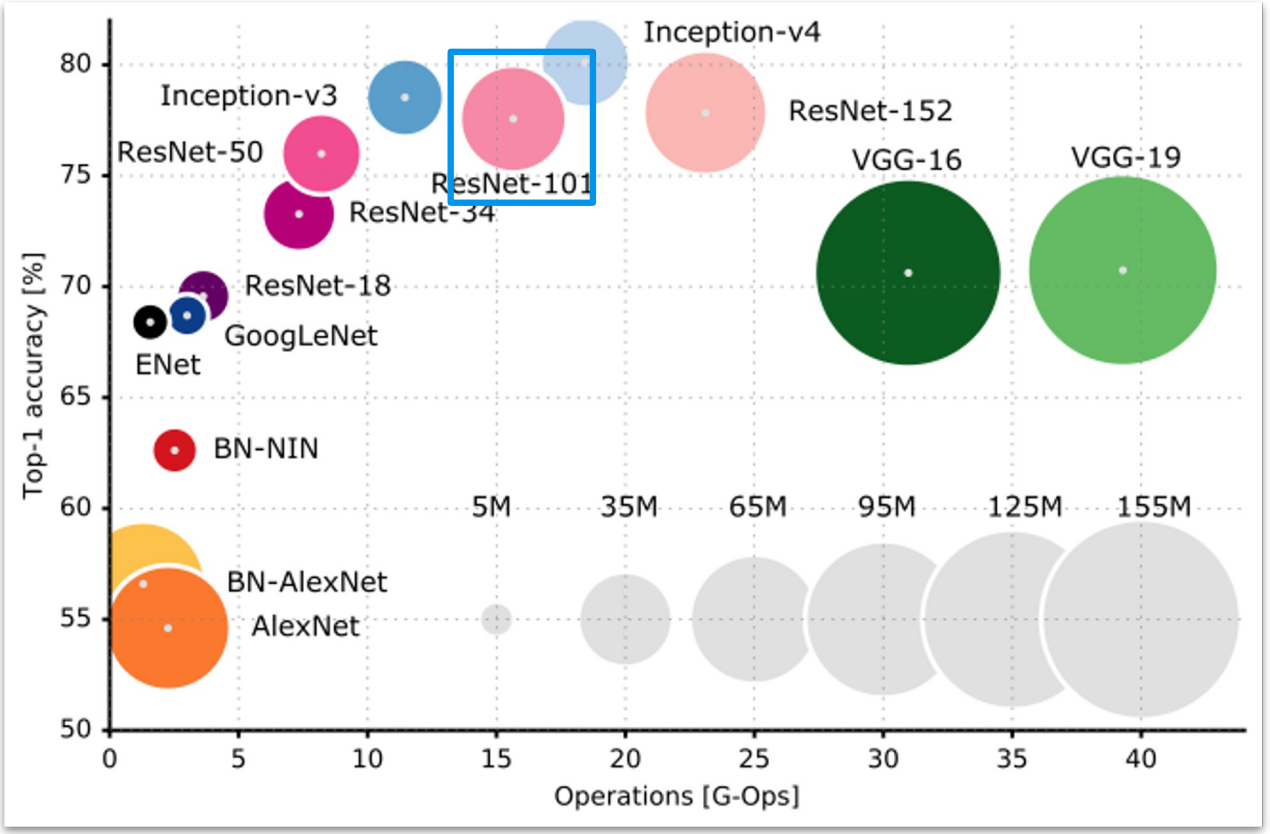


https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba

Slide Credit: Prof. Sandra Avila - UNICAMP

ResNet: Moderate efficiency depending on model, highest accuracy

The size of the blobs is proportional to the number of network parameters.



https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba

Slide Credit: Prof. Sandra Avila - UNICAMP

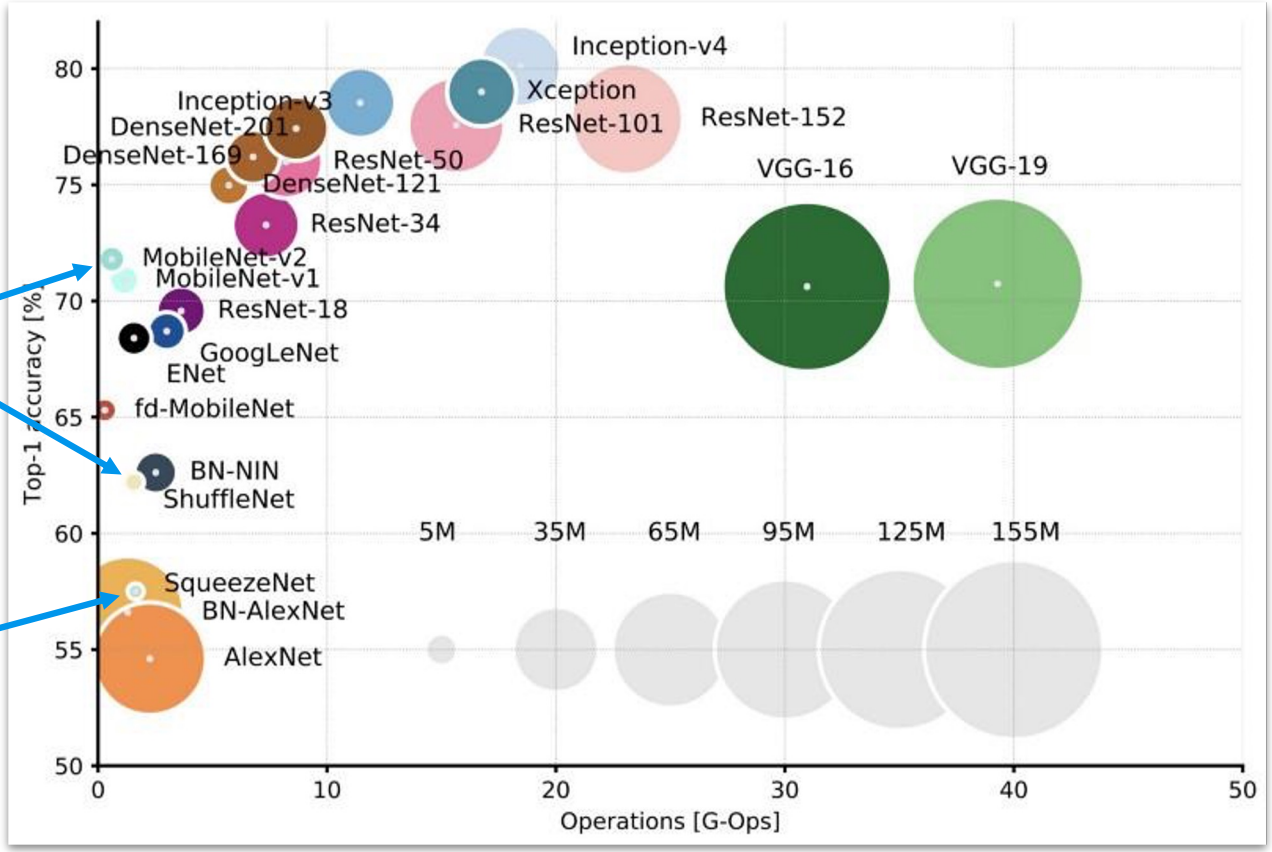
ShuffleNet

MobileNet

Xception

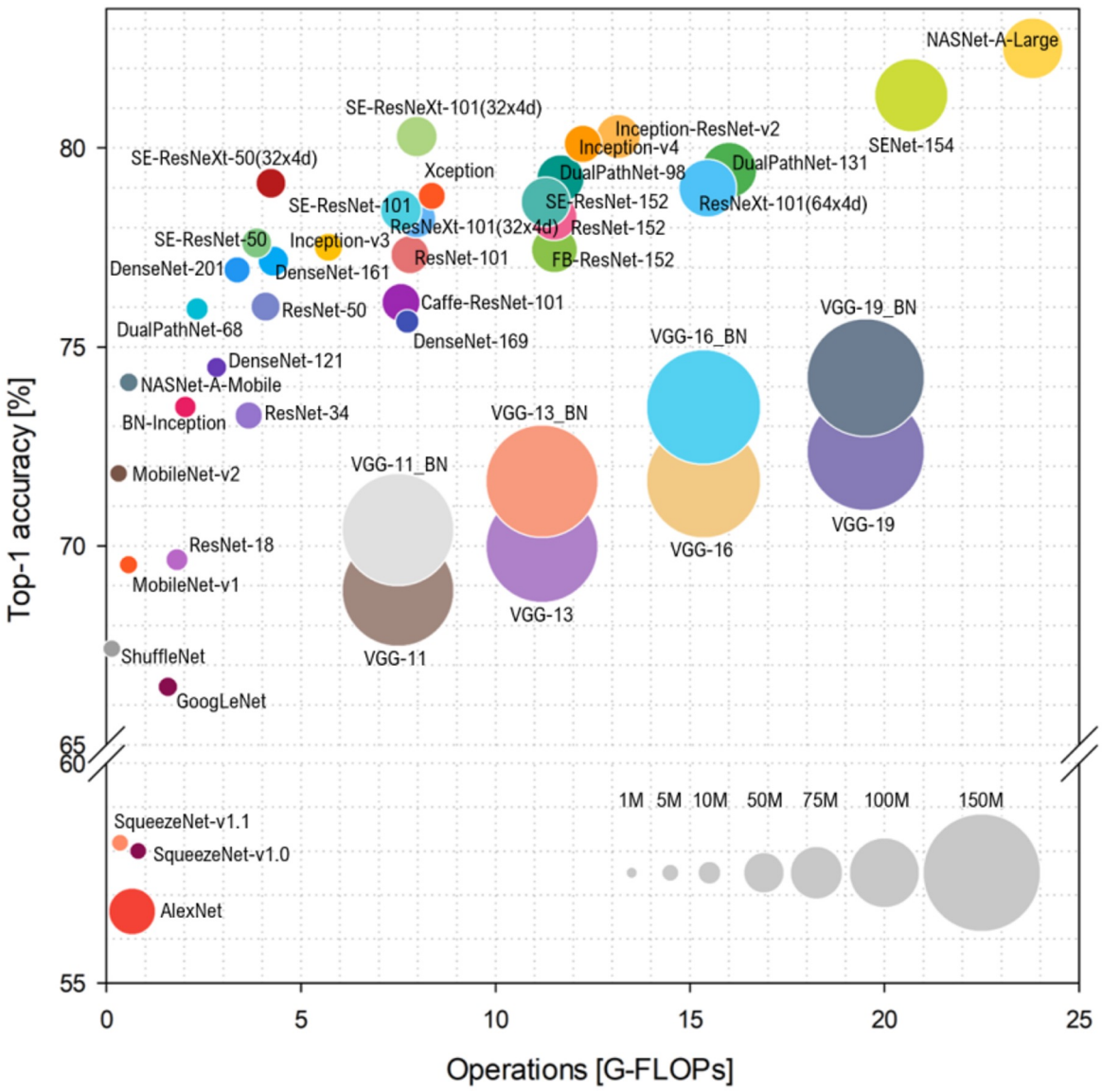
DenseNet

SqueezeNet



https://medium.com/towards-data-science/neural-network-architectures-156e5bad51ba

Slide Credit: Prof. Sandra Avila - UNICAMP



"Benchmark Analysis of Representative Deep Neural Network Architectures", Nov. 2018. https://doi.org/10.1109/ACCESS.2018.2877890

https://keras.io/api/applications/

Keras Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

Weights are downloaded automatically when instantiating a model. They are stored at `~/.keras/models/`.

Upon instantiation, the models will be built according to the image data format set in your Keras configuration file at `~/.keras/keras.json`. For instance, if you have set `image_data_format=channels_last`, then any model loaded from this repository will get built according to the TensorFlow data format convention, "Height-Width-Depth".

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5

https://keras.io/api/applications/

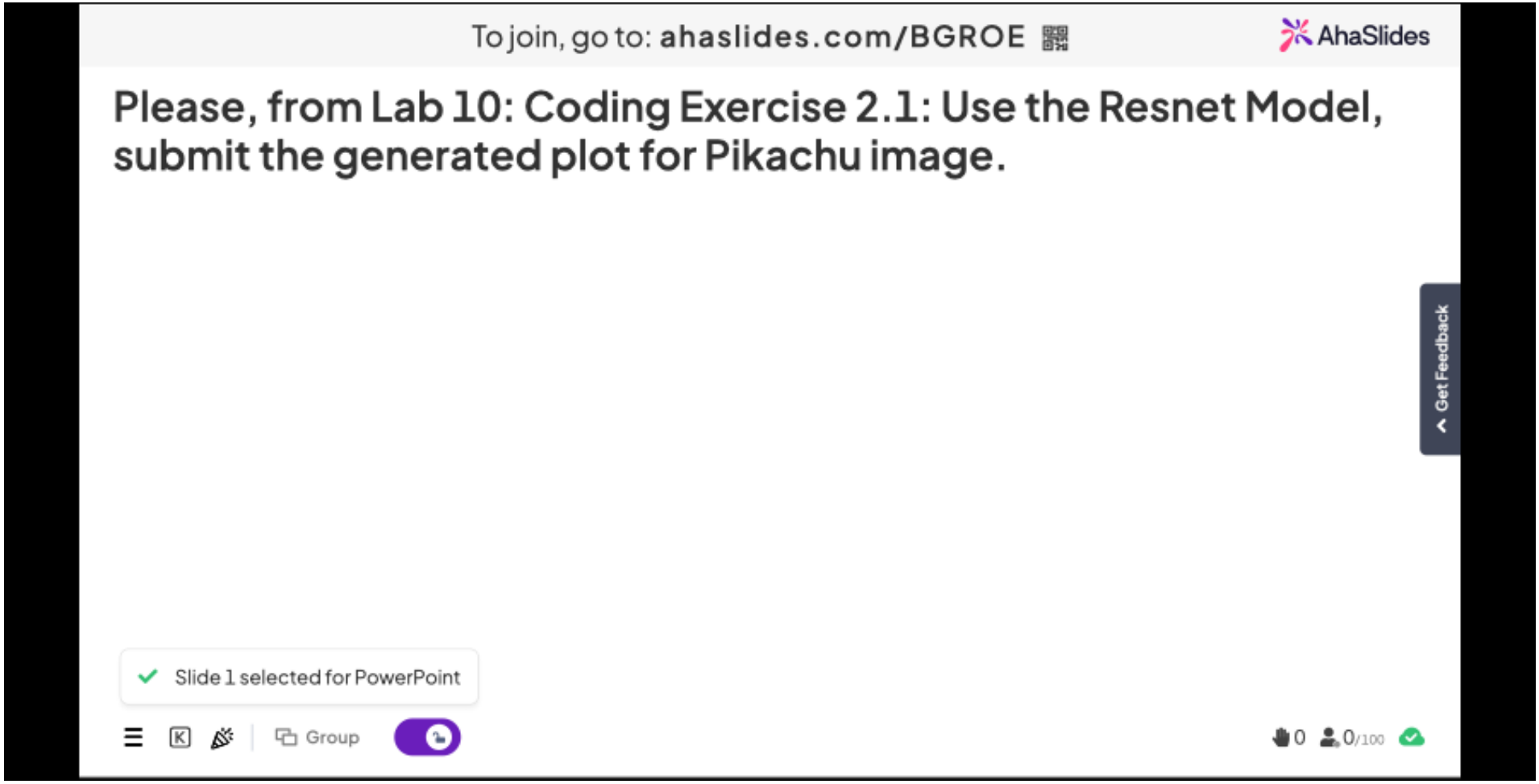
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-
ConvNeXtTiny	109.42	81.3%	-	28.6M	-	-	-

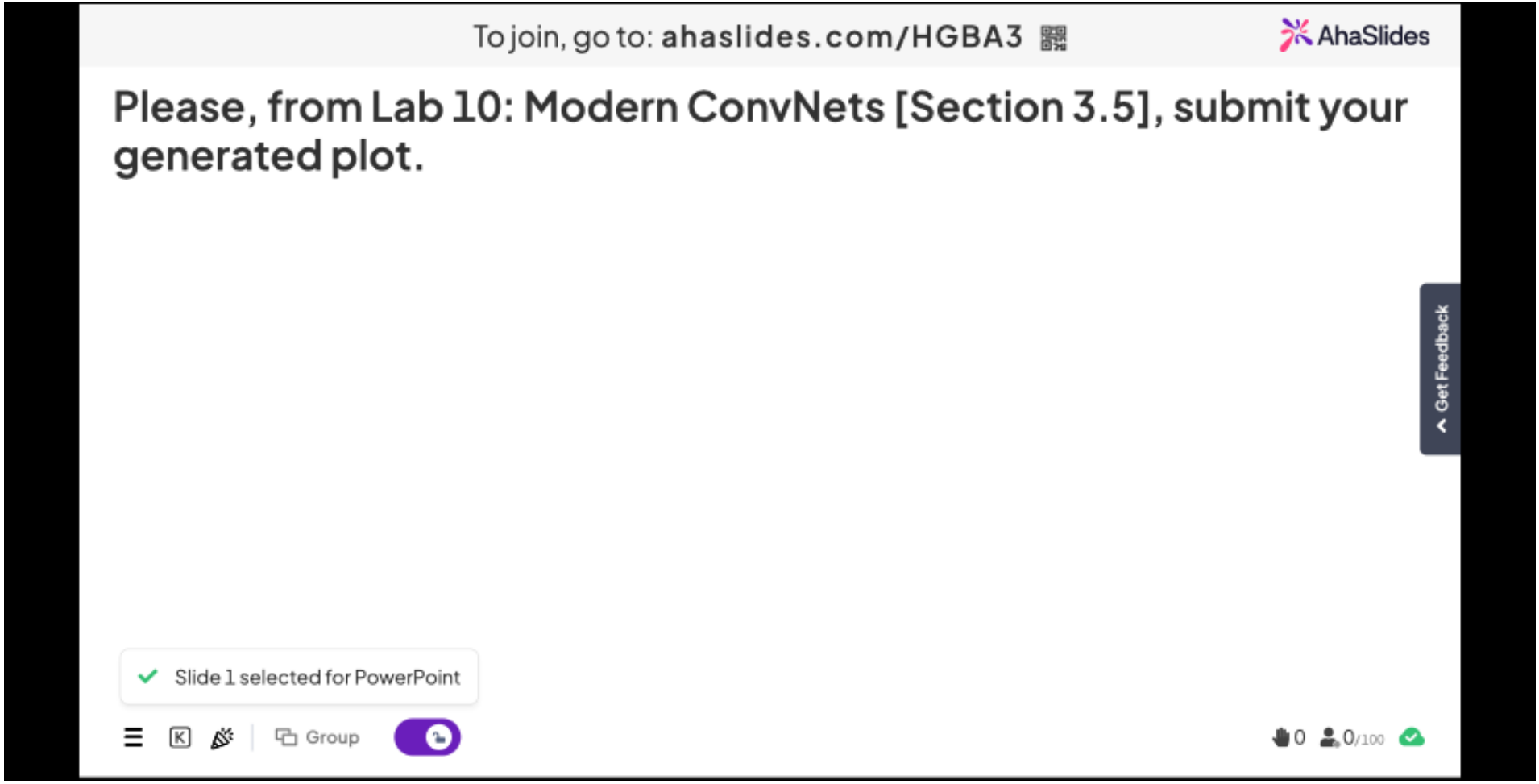
Lab 10: Using Modern ConvNets

Duration: 15 min

Steps:

1. Introduce to a classmate
2. Work in pairs on the exercise
3. Submit your answers on AhaSlides


To join, go to: ahaslides.com/BGROE
AhaSlides
Please, from Lab 10: Coding Exercise 2.1: Use the Resnet Model, submit the generated plot for Pikachu image.
Get Feedback
Slide 1 selected for PowerPoint
Group
0
0/100


To join, go to: ahaslides.com/HGBA3
AhaSlides
Please, from Lab 10: Modern ConvNets [Section 3.5], submit your generated plot.
Get Feedback
Slide 1 selected for PowerPoint
Group
0
0/100

Assignment 6: Using Modern ConvNets

Additional Resources

Convolutions

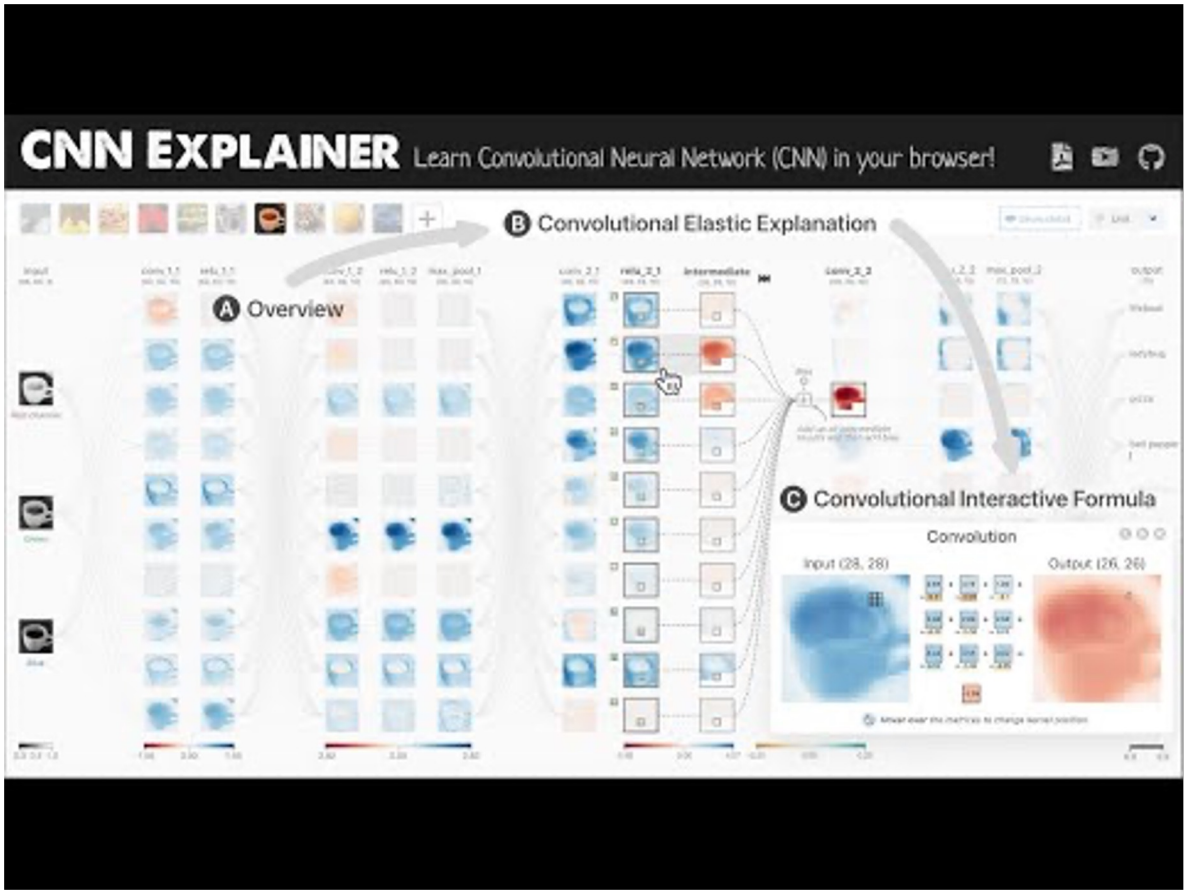
"A Guide to Convolution Arithmetic for Deep Learning" https://arxiv.org/pdf/1603.07285.pdf (Jan. 2018)

"Convolution animations" https://github.com/vdumoulin/conv_arithmetic

"A Comprehensive Introduction to Different Types of Convolutions in Deep Learning" (Jan. 2019) https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

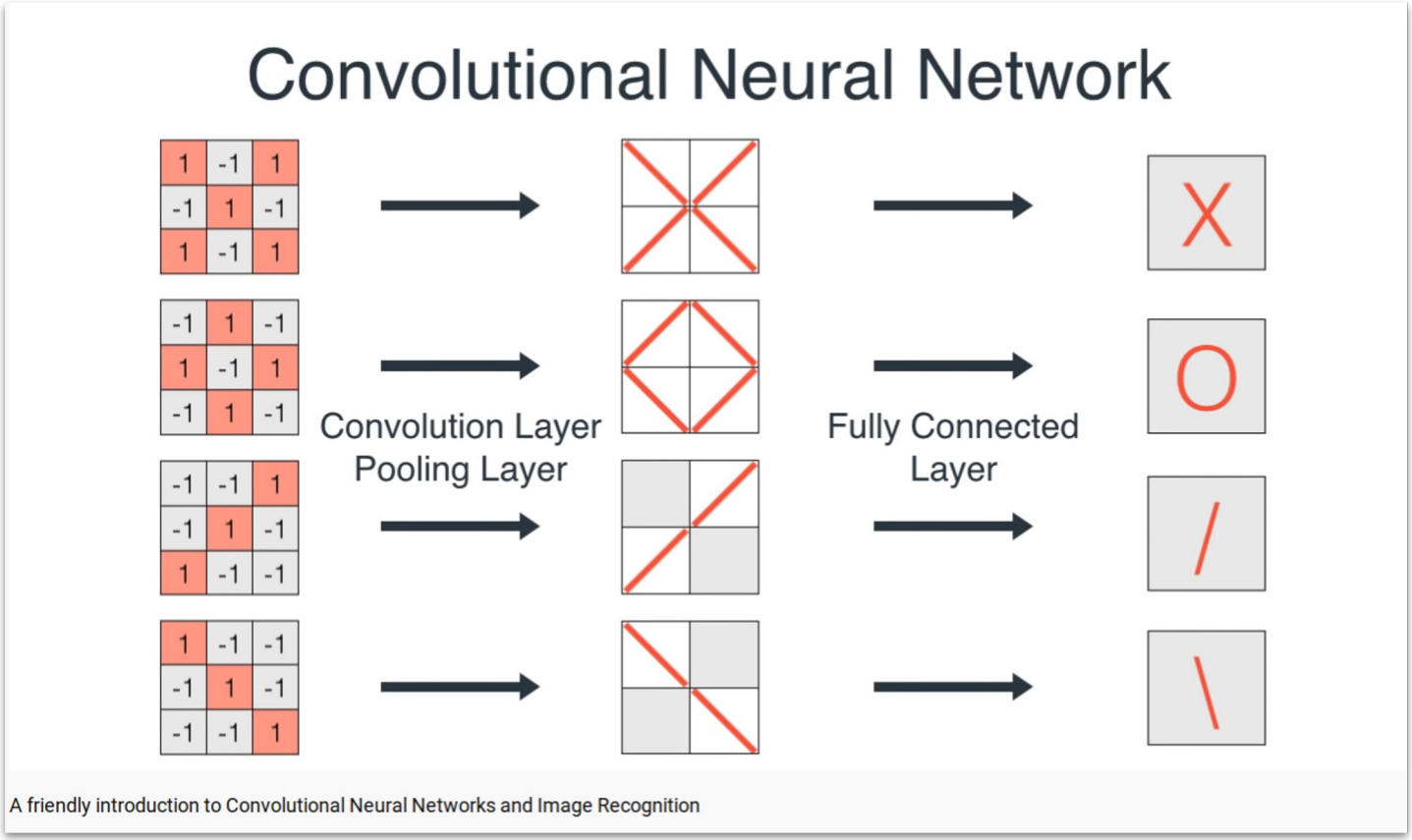
Slide Credit: Prof. Sandra Avila - UNICAMP

https://youtu.be/HnWIHWFbuUQ (3 min)



Slide Credit: Prof. Sandra Avila - UNICAMP

"A friendly introduction to Convolutional Neural Networks and Image Recognition" https://youtu.be/2-Ol7ZB0MmU



Slide Credit: Prof. Sandra Avila - UNICAMP

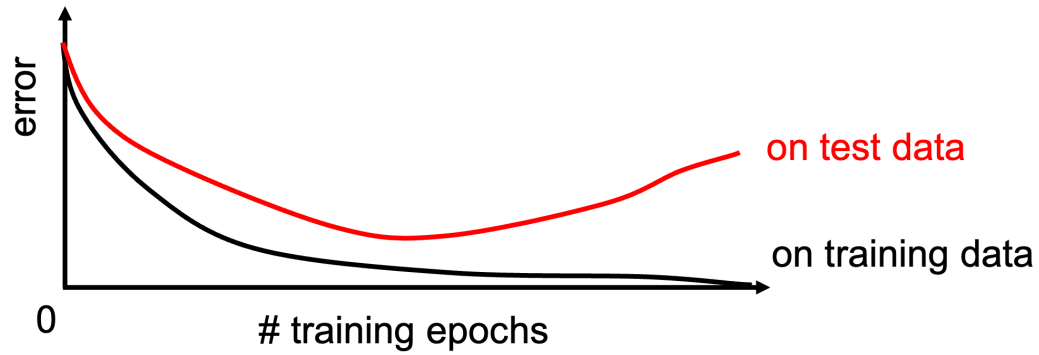
Practical matters

Comments on training algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data, with good choice of hyperparameters (e.g. learning rate).
- Thousands of epochs (epoch = network sees all training data once) may be required, hours or days to train.
- May be hard to set learning rate and to select number of hidden units and layers

Over-training prevention

- Running too many epochs can result in over-fitting.



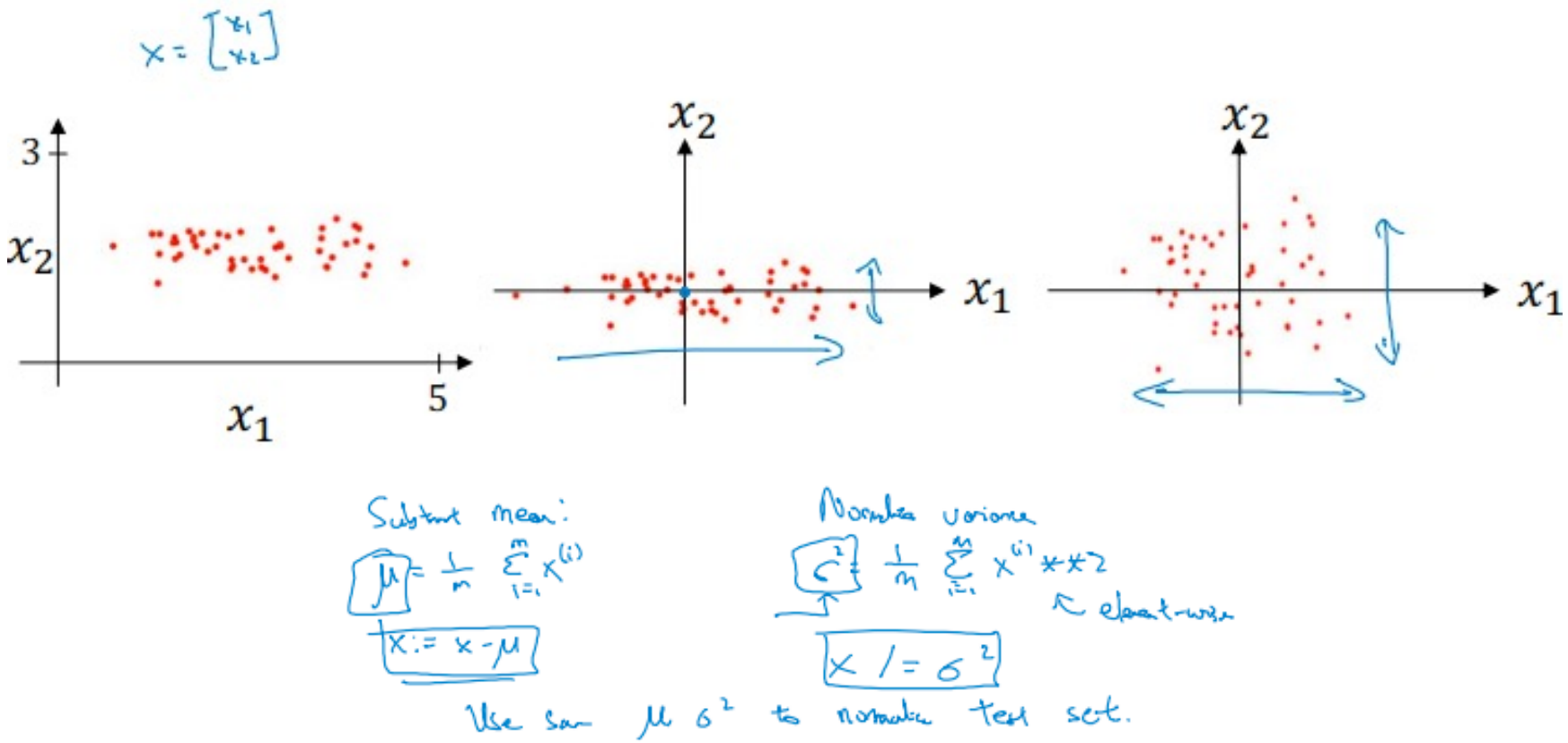
- Keep a hold-out validation set and test accuracy on it after every epoch.
- Stop training when additional epochs actually increase validation error.

Adapted from Ray Mooney

Training: Best practices

- Data
  - Center (subtract mean from) your data
  - Use data augmentation
  - Use mini-batch
- Weights/activations
  - To initialize weights, use “Xavier initialization”
  - Use regularization
  - Use RELU (most common), don’t use sigmoid
- Hyperparameters:
  - Learning rate: too high? Too low?
  - Use cross-validation to pick best model

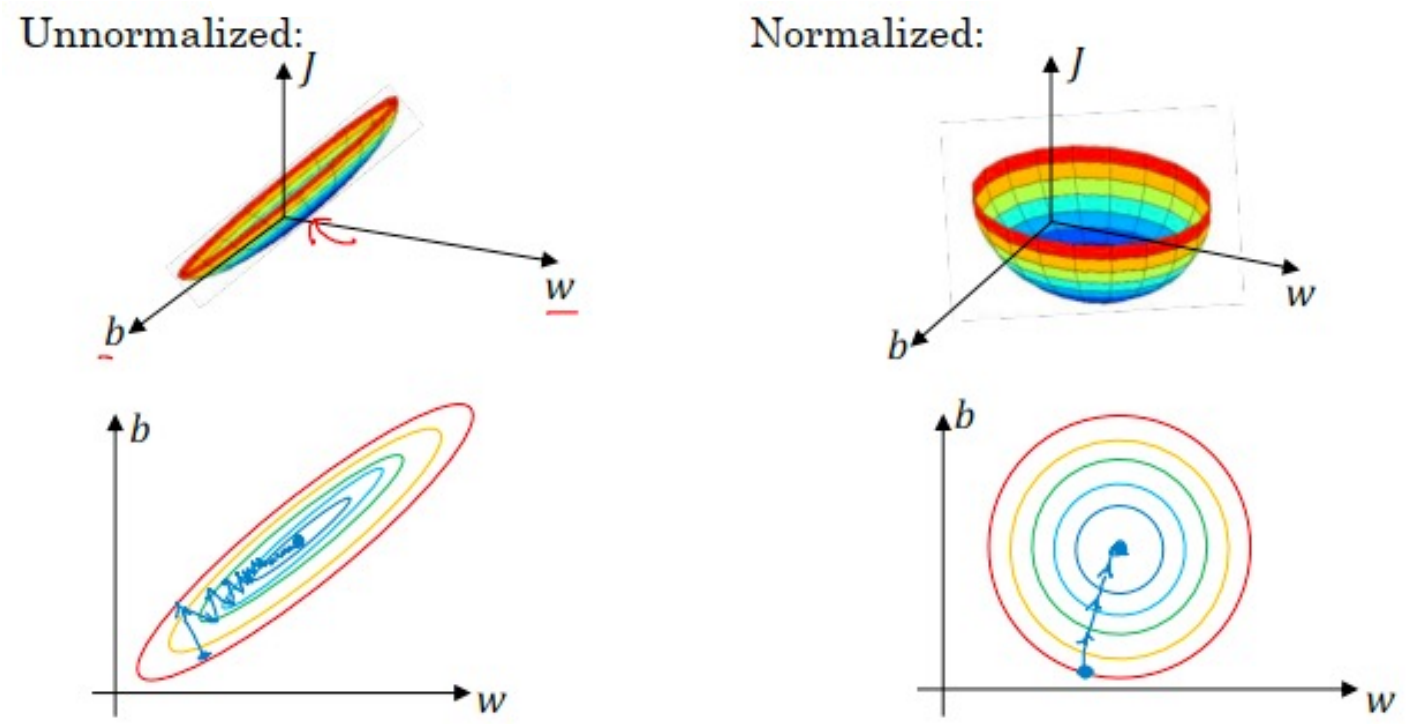
Best practices: Data



Adapted from Deep Coursera Specialization [Andrew Ng]

Best practices: Data. Why normalize inputs?

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$



Adapted from Deep Coursera Specialization [Andrew Ng]

Best Practices: Hyperparameters (Coarse Search)

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                              model, two_layer_net,
                                              num_epochs=5, reg=reg,
                                              update='momentum', learning_rate_decay=0.9,
                                              sample_batches = True, batch_size = 100,
                                              learning_rate=lr, verbose=False)
```

note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Best Practices: Hyperparameters (Finer Search)

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

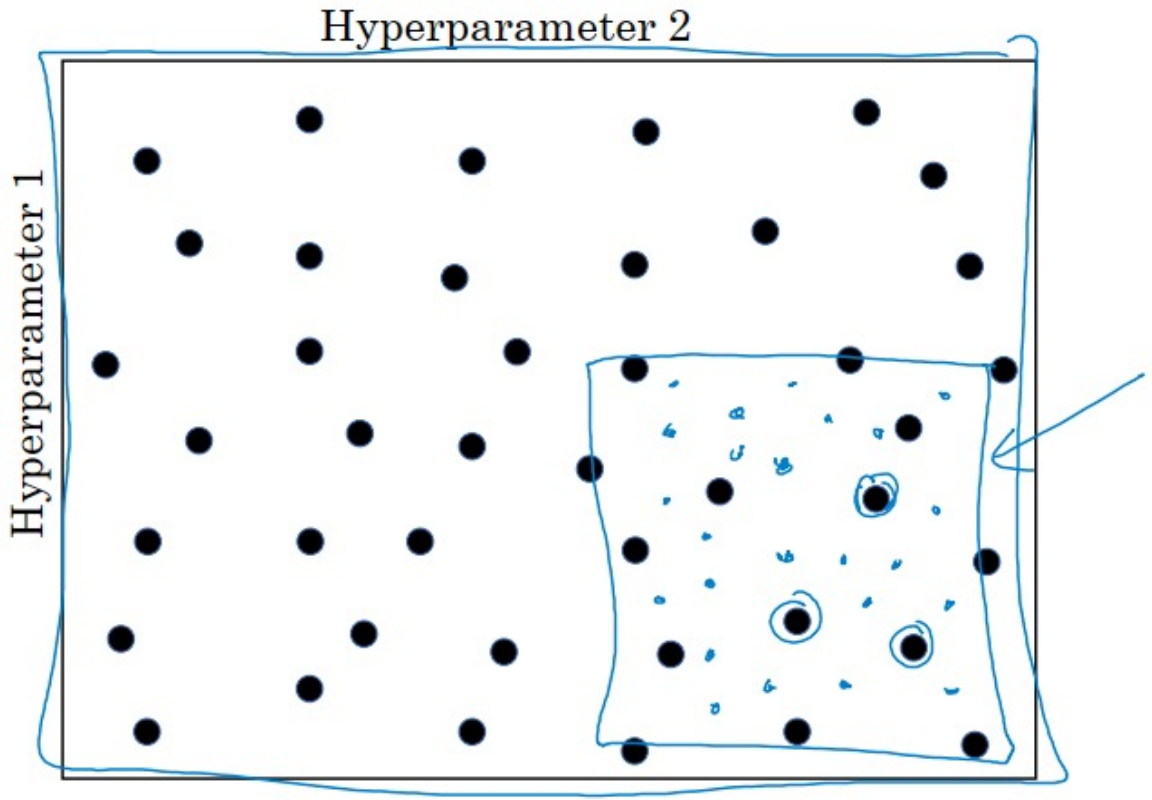
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Best Practices: Hyperparameters (Coarse to fine search)



Adapted from Deep Coursera Specialization [Andrew Ng]

Best Practices: Hyperparameters (Use random search instead of Grid Search)

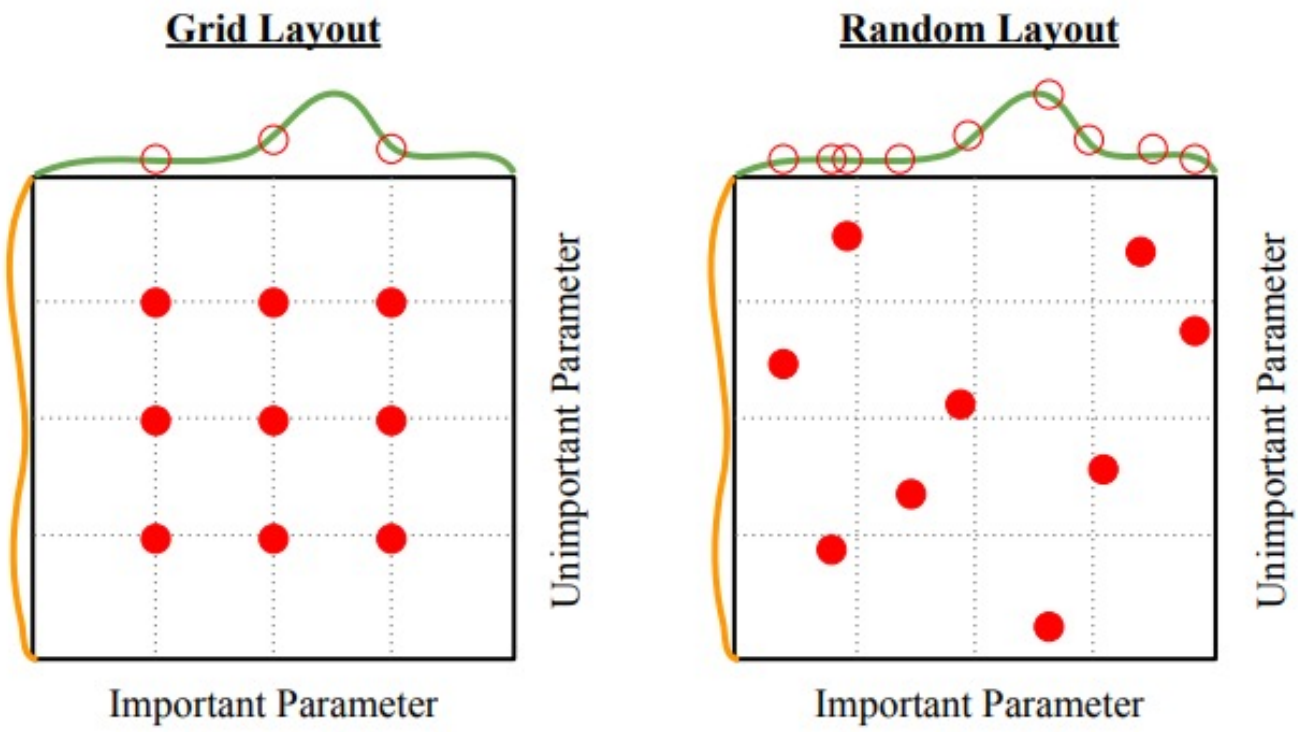


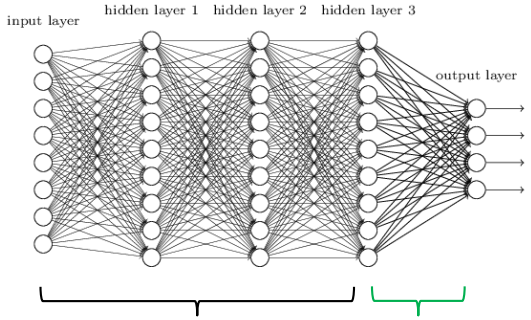
Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

Transfer Learning

"You need a lot of data if you want to train/use CNNs"

Transfer Learning with CNNs

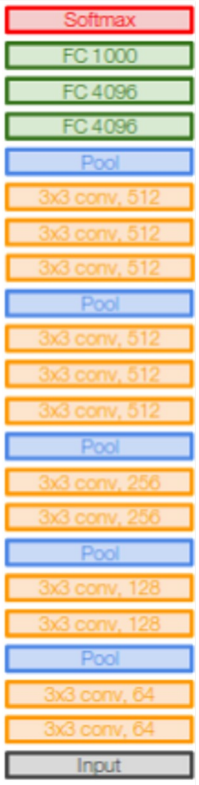
- The more weights you need to learn, the more data you need
- That's why with a deeper network, you need more data to train than for a shallower net
- One possible solution:



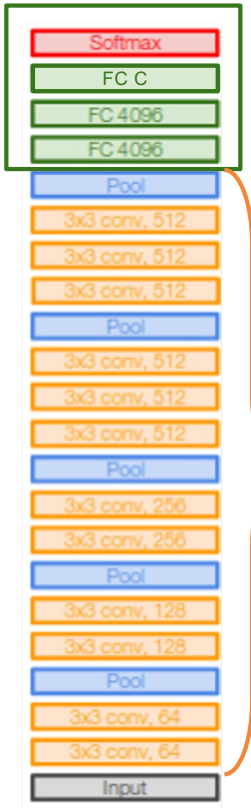
Set these to the already learned weights from another network

Learn these on your own task

Transfer Learning with CNNs



Train on ImageNet



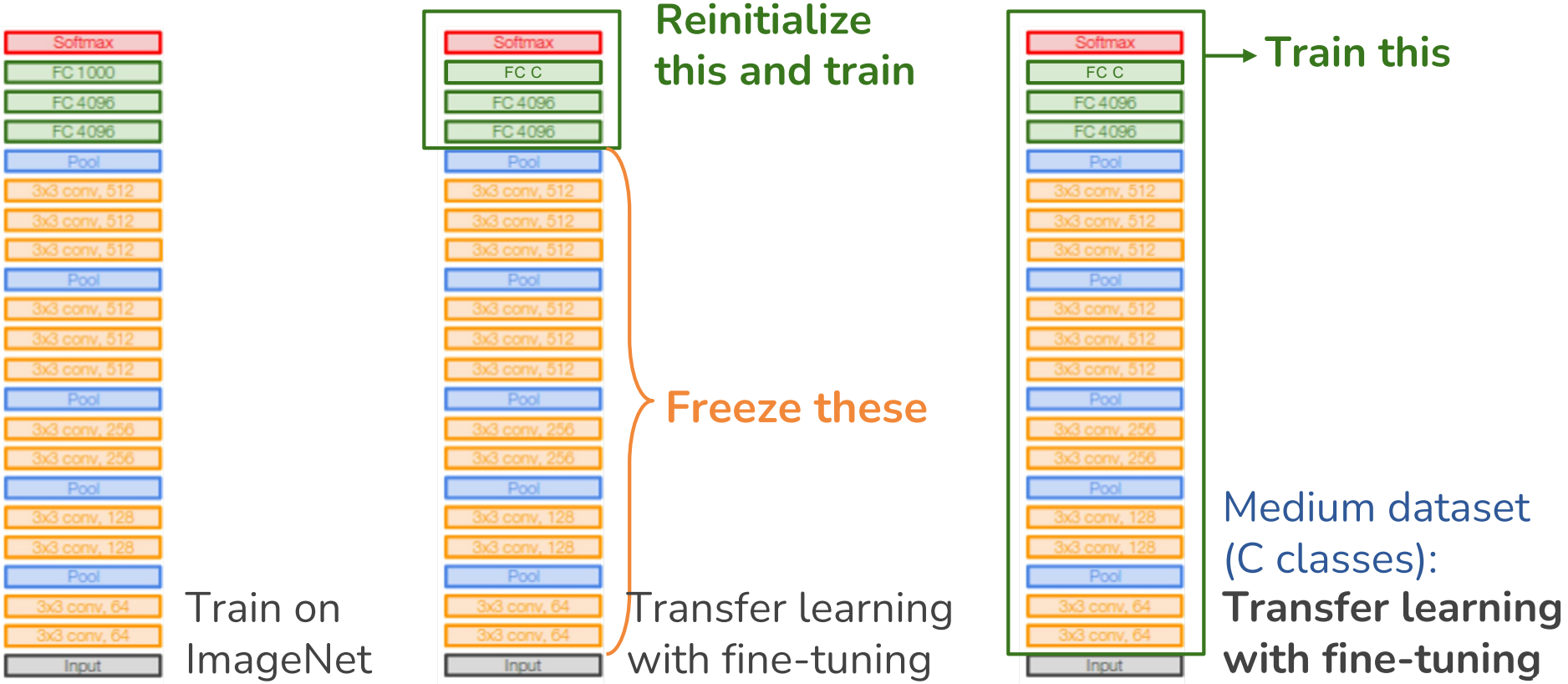
Reinitialize this and train

Freeze these

Small dataset (C classes): Transfer learning with fine-tuning

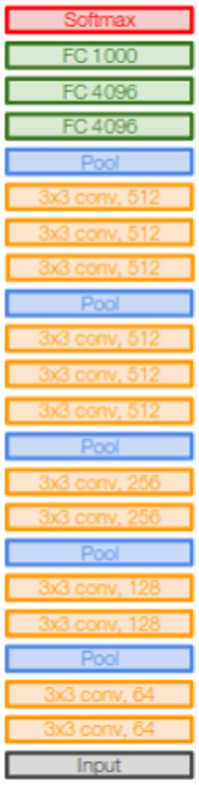
Slide Credit: Prof. Sandra Avila - UNICAMP

Transfer Learning with CNNs

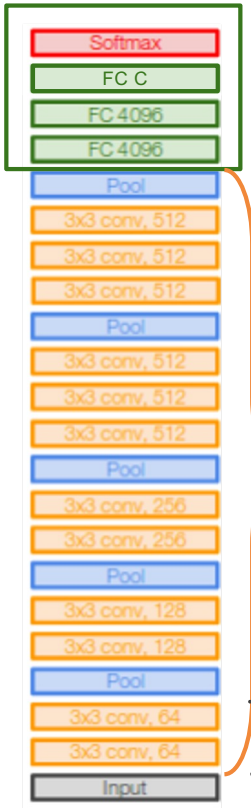


Slide Credit: Prof. Sandra Avila - UNICAMP

Transfer Learning with CNNs



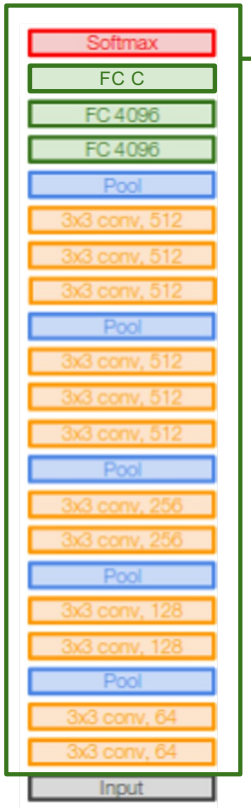
Train on ImageNet



Reinitialize this and train

Freeze these

Transfer learning with fine-tuning



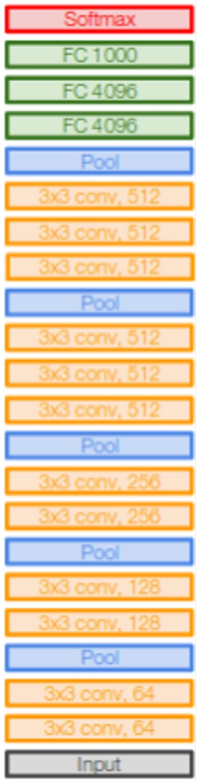
Train this

Lower learning rate when fine-tuning; 1/10 of original LR is good starting point

Transfer learning with fine-tuning

Slide Credit: Prof. Sandra Avila - UNICAMP

Transfer Learning with CNNs



More specific

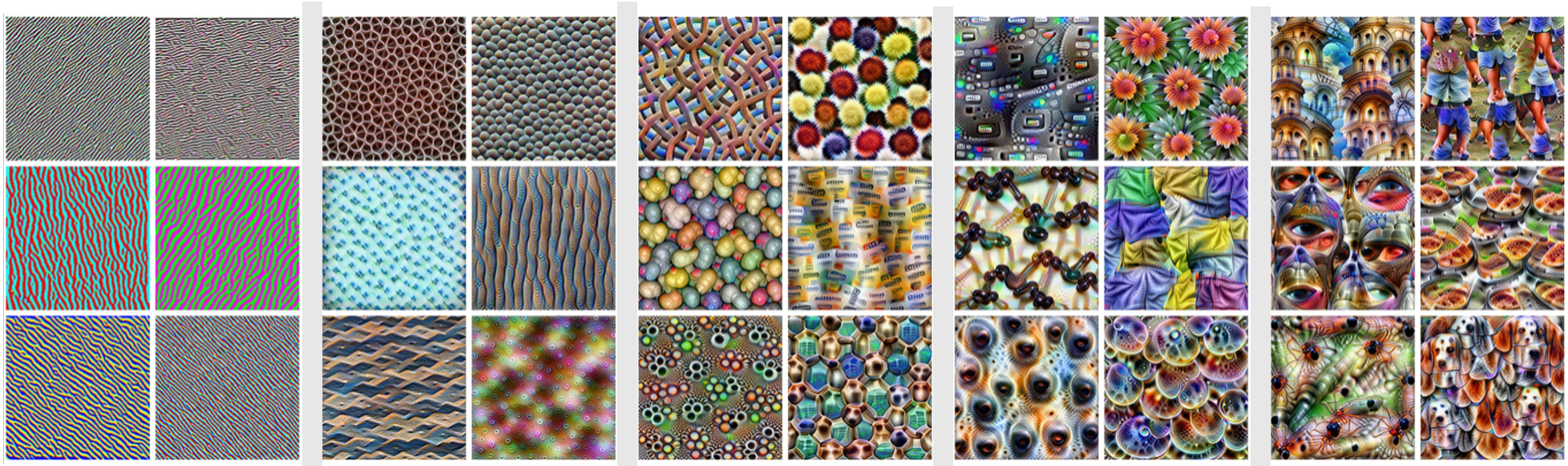
More generic

	Very similar dataset	Very different dataset
Very little data	Use classifier on top layer	You're in trouble... Try classifier from different stages
Quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Adapted from Andrej Karpathy

Slide Credit: Prof. Sandra Avila - UNICAMP

https://distill.pub/2017/feature-visualization

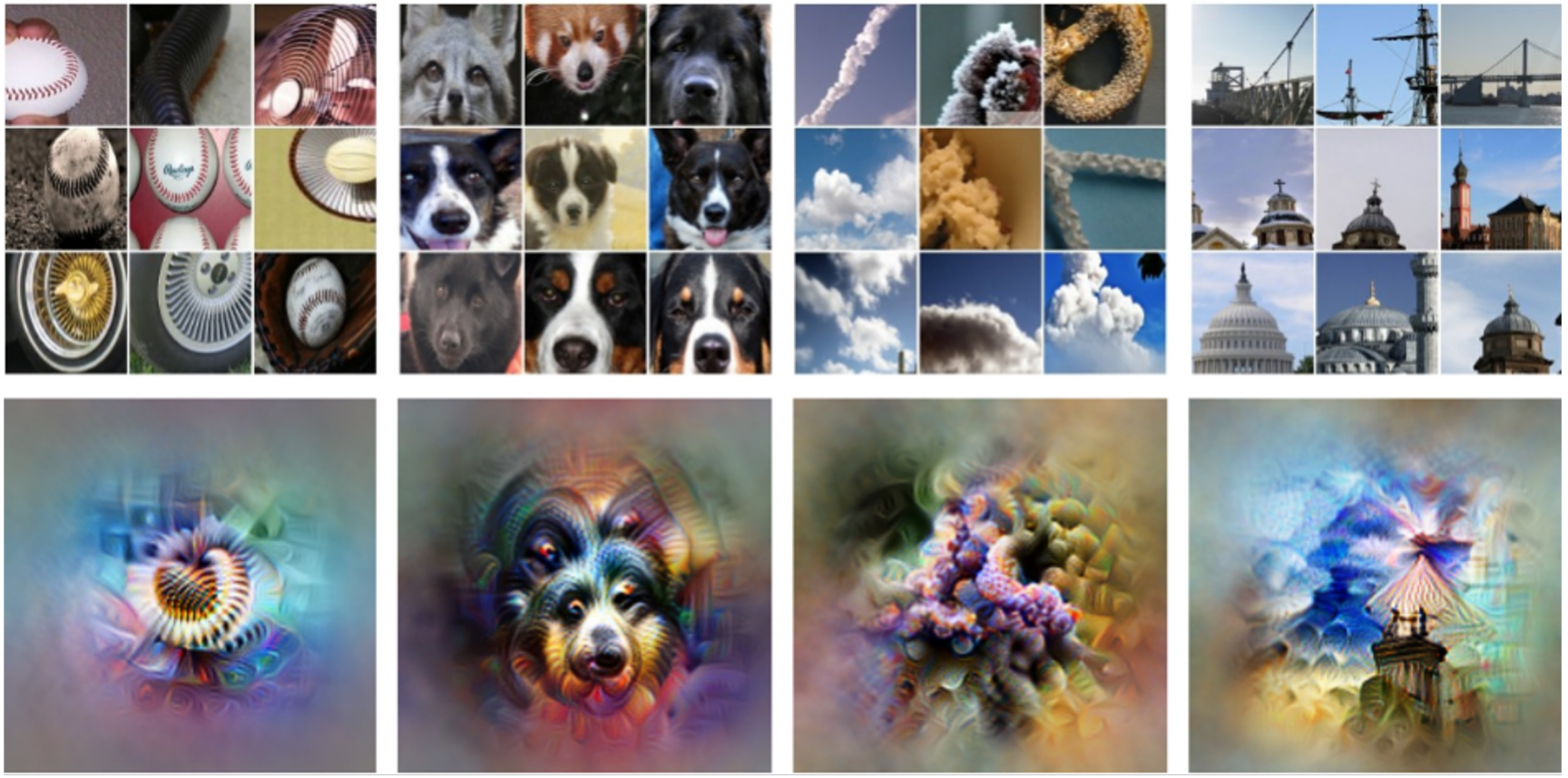


Edges Textures Patterns Parts Objects

More generic → **More specific**

Slide Credit: Prof. Sandra Avila - UNICAMP

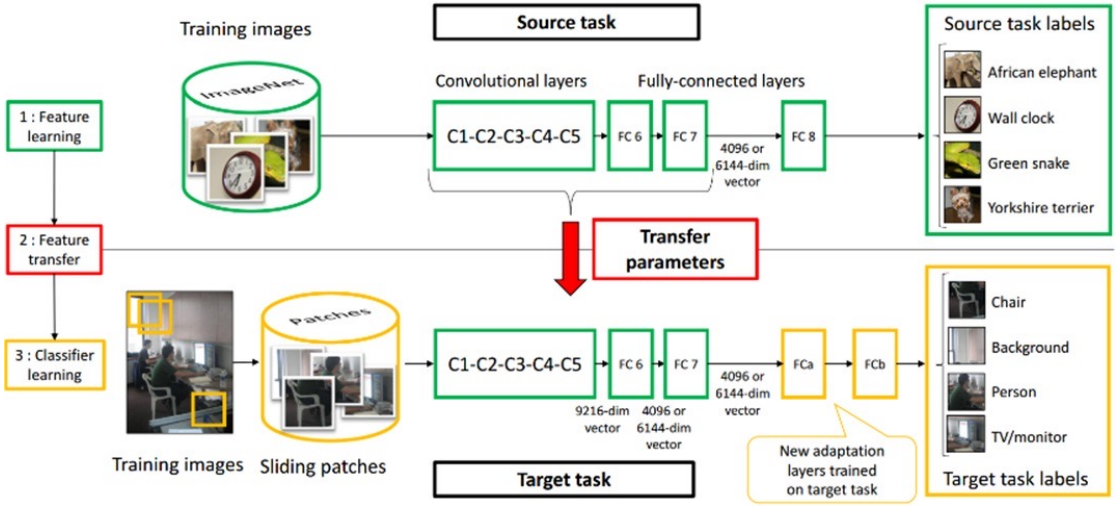
https://distill.pub/2017/feature-visualization



Slide Credit: Prof. Sandra Avila - UNICAMP

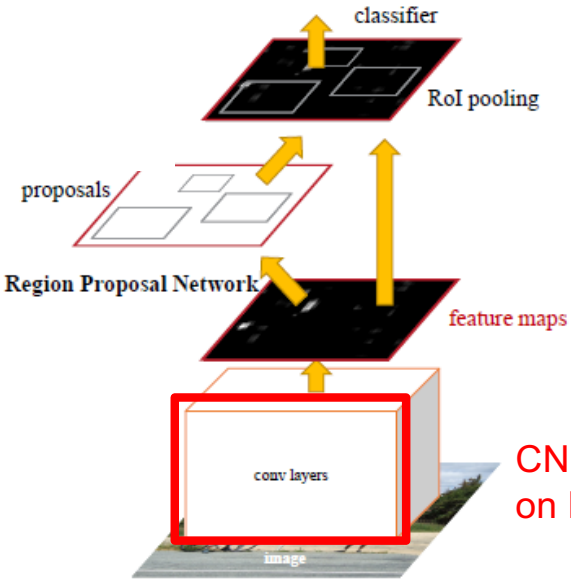
Pre-training on ImageNet

- Have a source domain and target domain
- Train a network to classify ImageNet classes
  - Coarse classes and ones with fine distinctions (dog breeds)
- Remove last layers and train layers to replace them, that predict target classes



Oquab et al., "Learning and Transferring Mid-Level Image Representations…", CVPR 2014

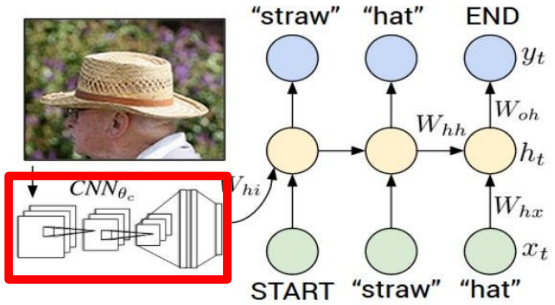
Transfer learning with CNNs is pervasive…



CNN pretrained on ImageNet

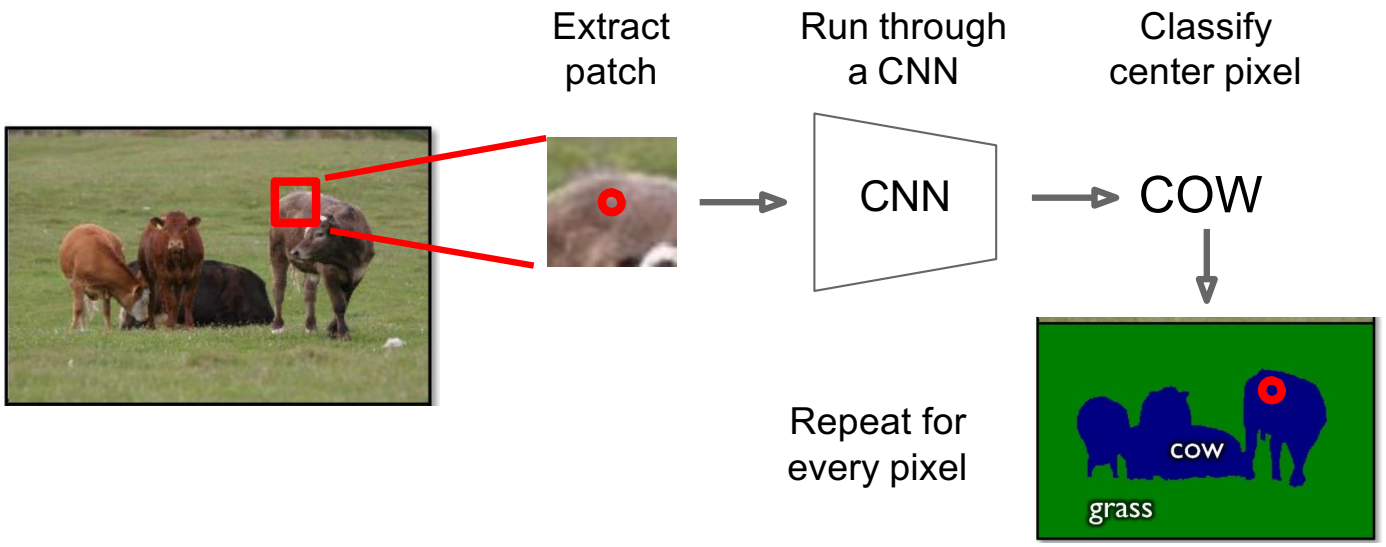
Object Detection
Ren et al., "Faster R-CNN", NIPS 2015

Image Captioning
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

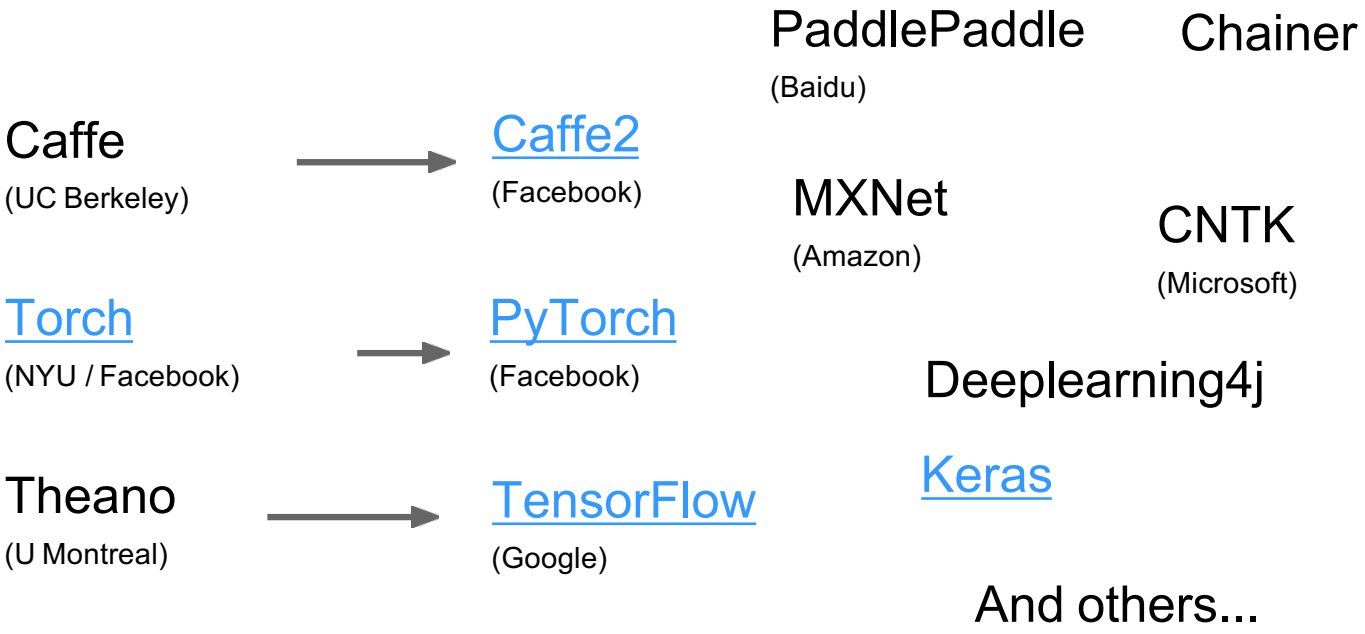


Adapted from Andrej Karpathy

Semantic segmentation



Software: A Zoo of Frameworks!



Some Learning Resources

- http://deeplearning.net/
- http://cs231n.stanford.edu

Summary

- We use deep neural networks because of their strong performance in practice
- Convolutional neural network (CNN)
  - Special operations: Convolution, max pooling
- Common CNN Architectures
- Practices for good training and preventing overfitting
  - Dropout; data augmentation; transfer learning

Extra

CNN Architectures

- CNN Architectures
  - LeNet (1998)
  - AlexNet (2012)
  - ZFNet (2013)
  - VGGNet (2014)
  - GoogLeNet (2014)
  - ResNet (2015)

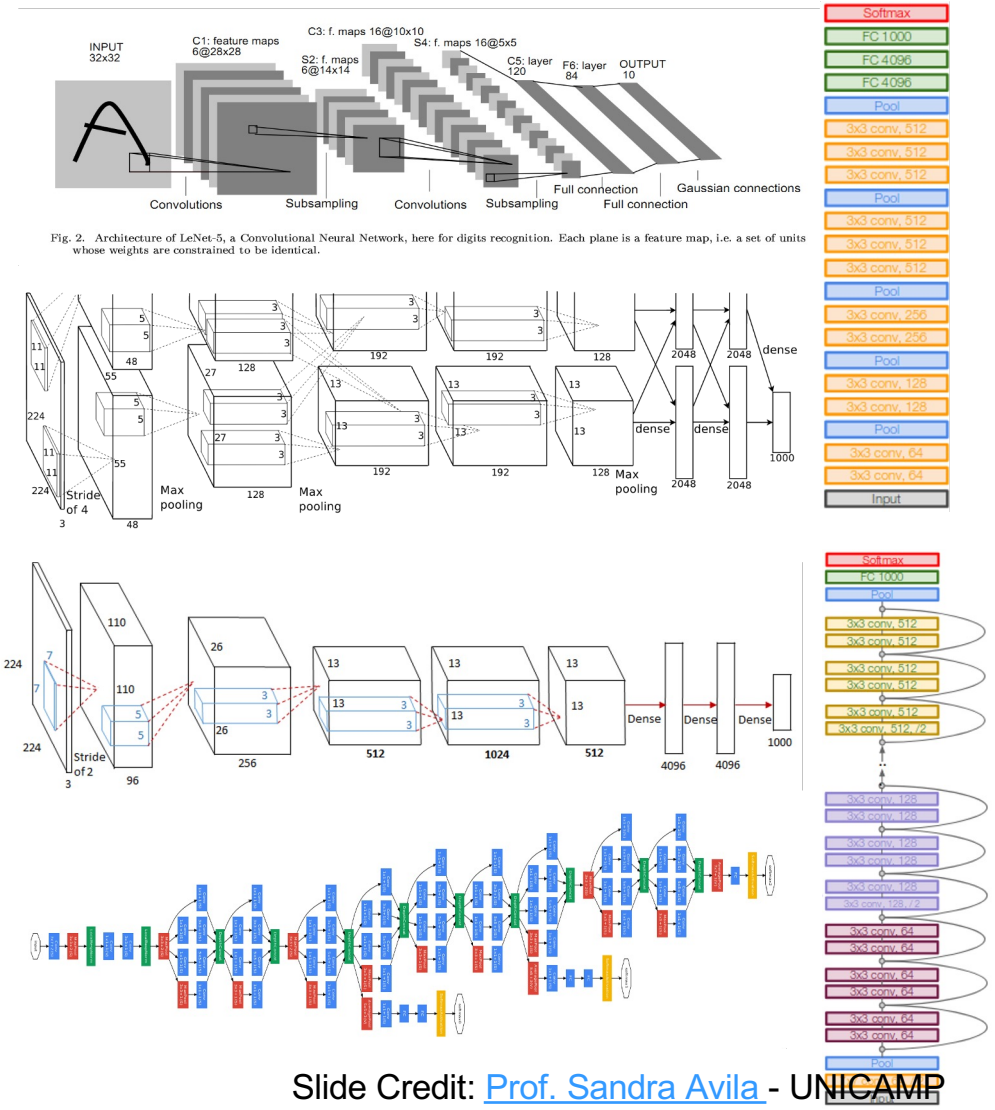


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Slide Credit: Prof. Sandra Avila - UNICAMP

Extra: CNN Architect

- CNN Architectures
  - **LeNet (1998)**
  - AlexNet (2012)
  - ZFNet (2013)
  - VGGNet (2014)
  - GoogLeNet (2014)
  - ResNet (2015)

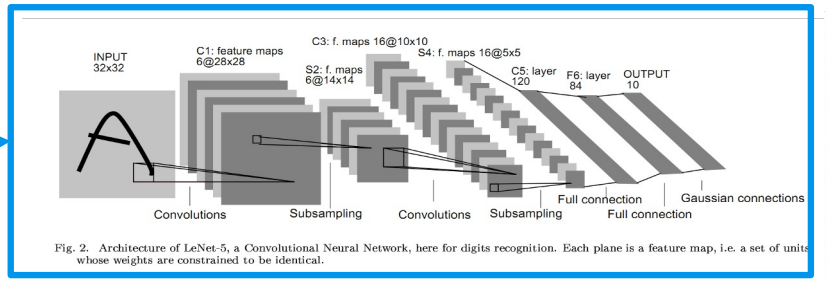
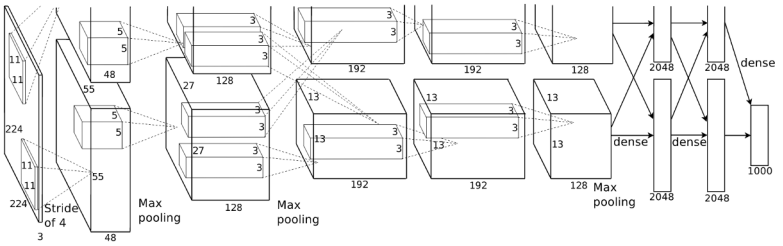
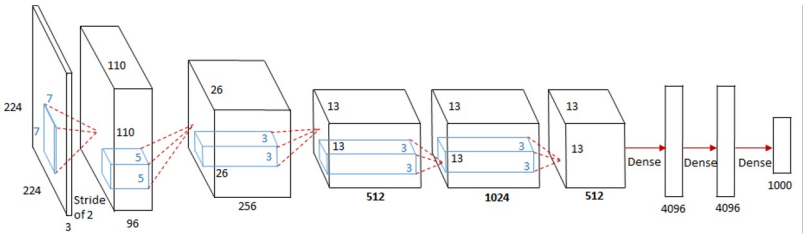
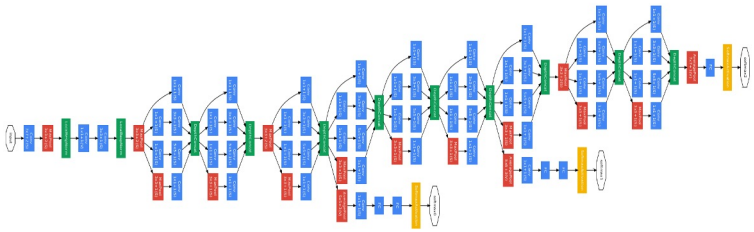
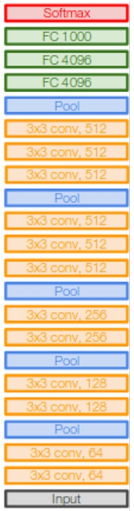


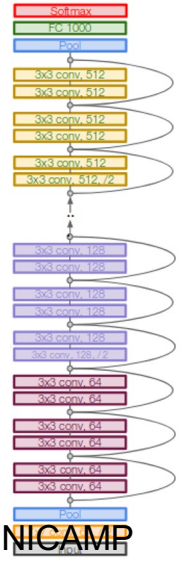
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.





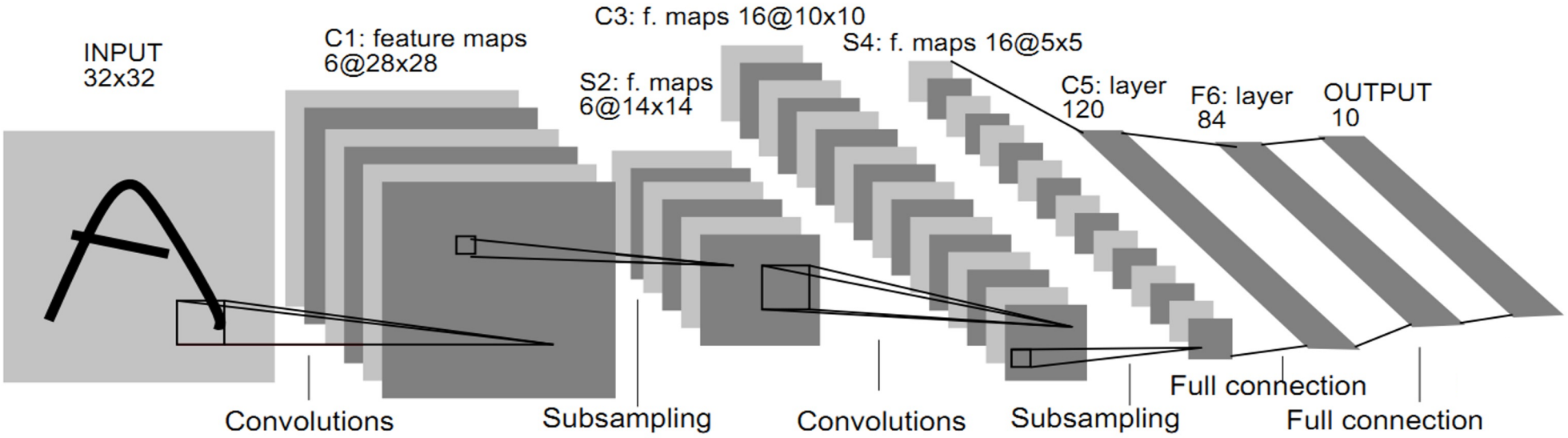






Slide Credit: Prof. Sandra Avila - UNICAMP

LeNet-5 [LeCun et al., 1998]



Convolution filters: 5x5 with stride 1

Subsampling (Pooling) layers: 2x2 with stride 2

[CONV-POOL-CONV-POOL-FC-FC]

CNN Architectures

- CNN Architectures
  - LeNet (1998)
  - AlexNet (2012)
  - **ZFNet (2013)**
  - VGGNet (2014)
  - GoogLeNet (2014)
  - ResNet (2015)

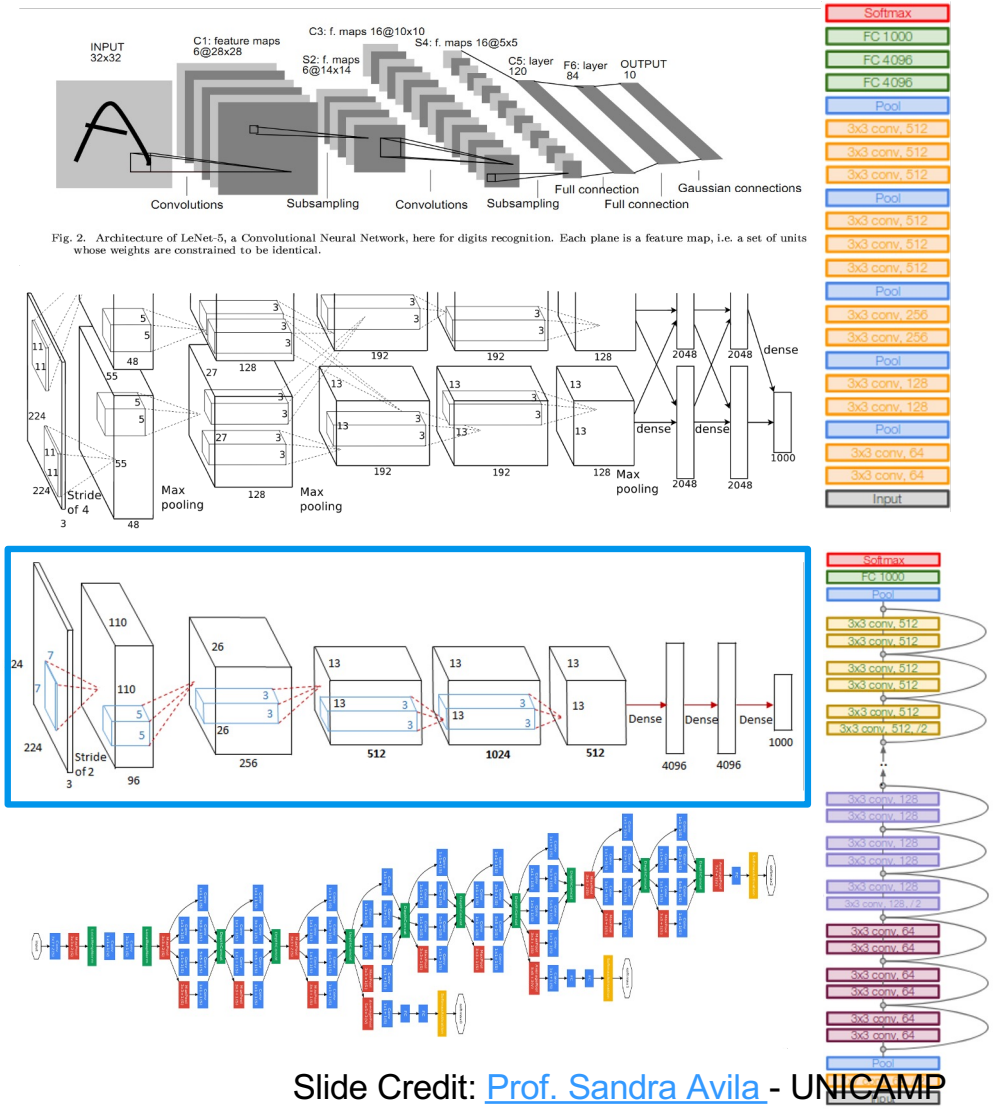
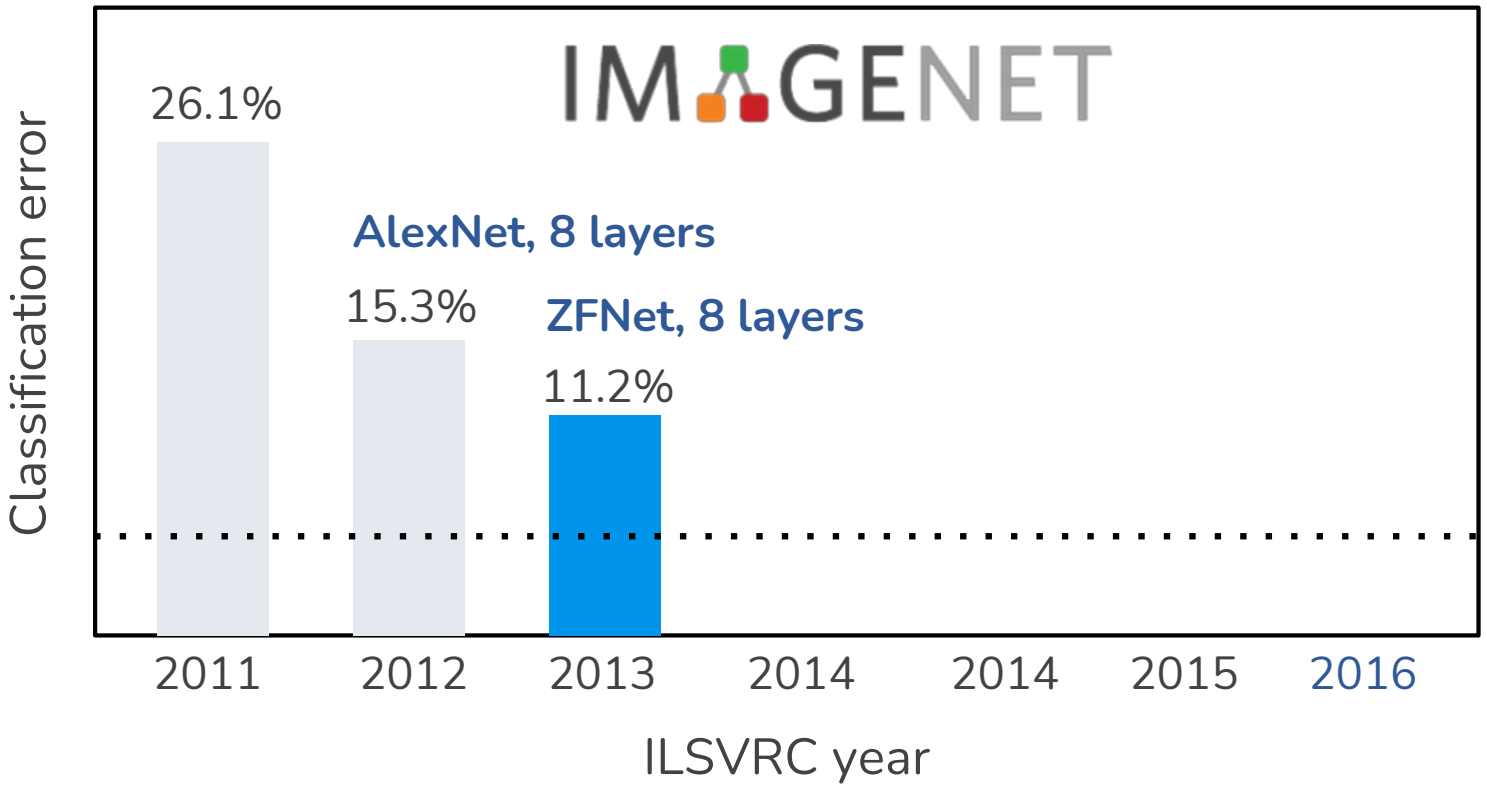


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

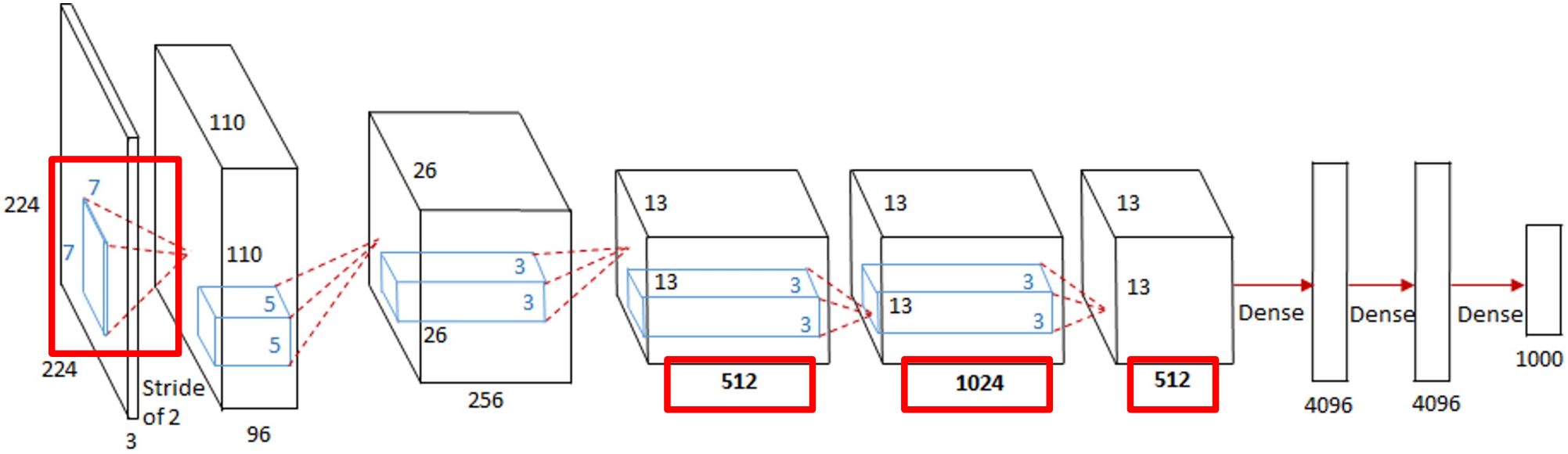
Slide Credit: Prof. Sandra Avila - UNICAMP



"Visualizing and Understanding Convolutional Networks", ECCV 2014.

Slide Credit: Prof. Sandra Avila - UNICAMP

ZFNet [Zeiler & Fergus, 2013]



AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

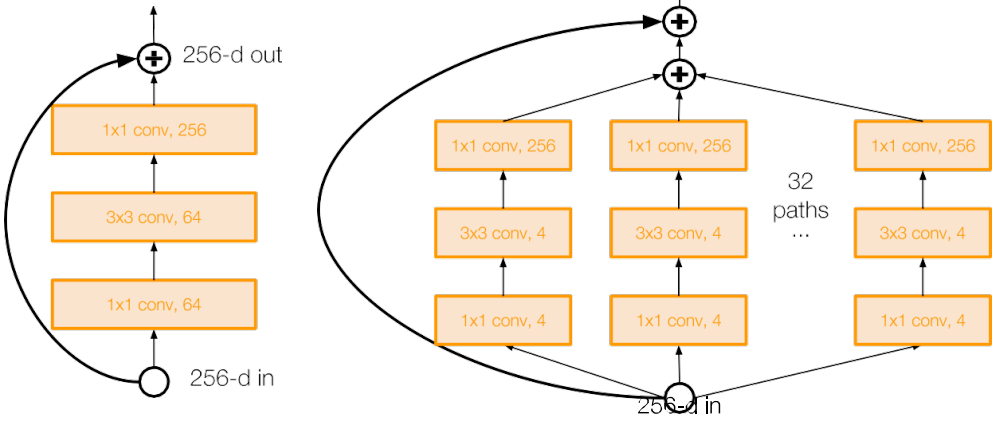
Slide Credit: Prof. Sandra Avila - UNICAMP

Extra: 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

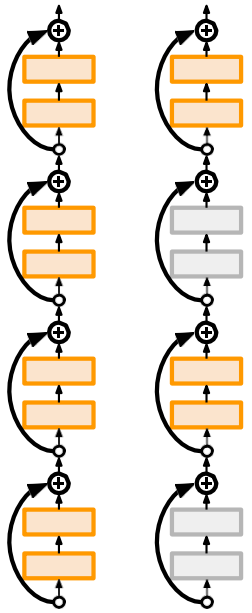


Improving ResNets …

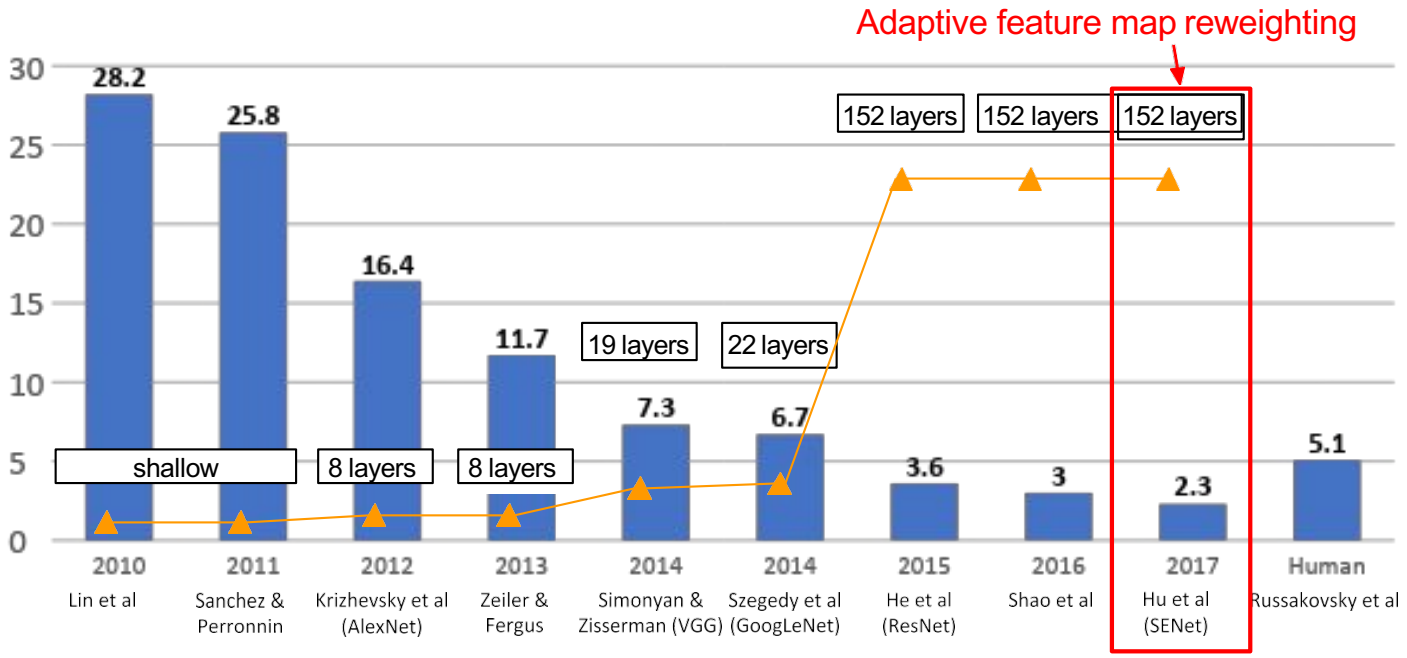
Deep Networks with Stochastic Depth

[Huang et al. 2016]

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

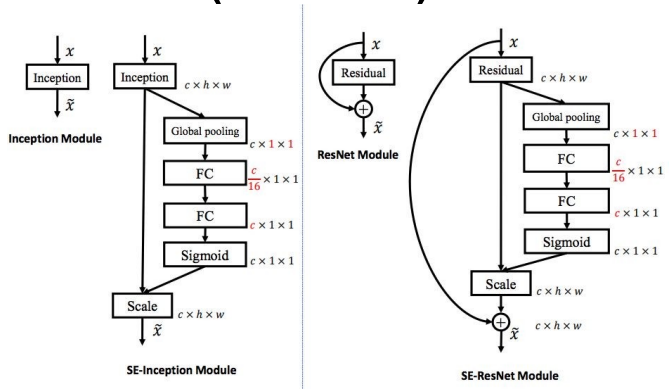


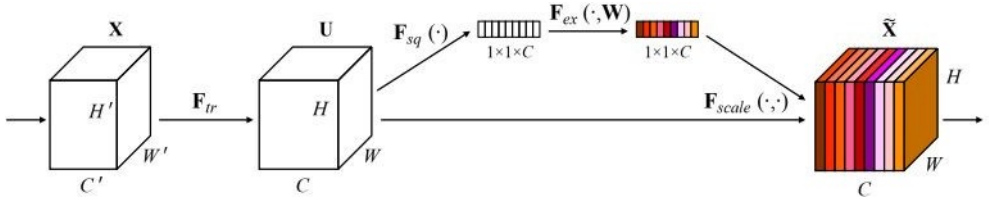
Improving ResNets …

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)



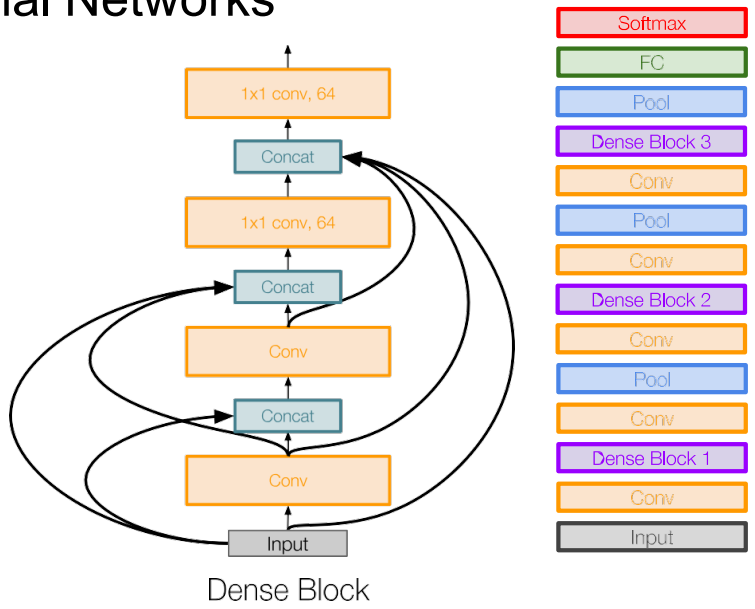


Beyond ResNets …

Densely Connected Convolutional Networks

[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



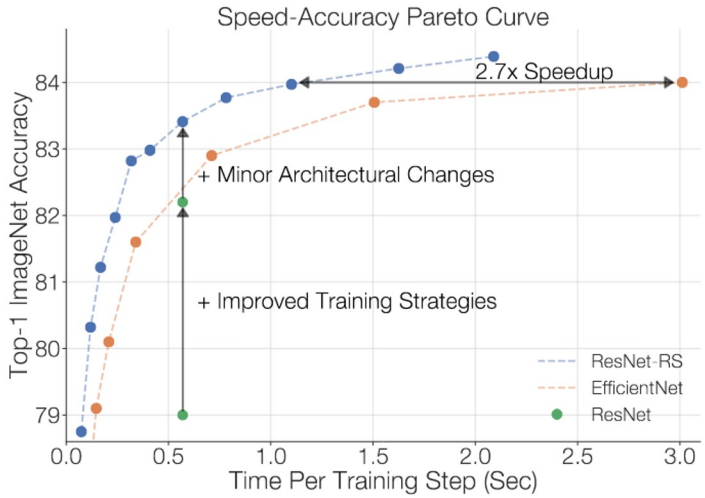
https://arxiv.org/pdf/2103.07579.pdf, NeurIPS 2021

Revisiting ResNets: Improved Training and Scaling Strategies

Irwan Bello [1] William Fedus [1] Xianzhi Du [1] Ekin D. Cubuk [1] Aravind Srinivas [2] Tsung-Yi Lin [1]
Jonathon Shlens [1] Barret Zoph [1]

Abstract

Novel computer vision architectures monopolize the spotlight, but the impact of the model architecture is often conflated with simultaneous changes to training methodology and scaling strategies. Our work revisits the canonical ResNet (He et al., 2015) and studies these three aspects in an effort to disentangle them. Perhaps surprisingly, we find that training and scaling strategies may matter more than architectural changes, and further, that the resulting ResNets match recent state-of-the-art models. We show that the best performing scaling strategy depends on the training regime and offer



https://arxiv.org/pdf/2103.07579.pdf, NeurIPS 2021

Improvements	Top-1	Δ
ResNet-200	79.0	—
+ Cosine LR Decay	79.3	**+0.3**
+ Increase training epochs	78.8 †	-0.5
+ EMA of weights	79.1	**+0.3**
+ Label Smoothing	80.4	**+1.3**
+ Stochastic Depth	80.6	**+0.2**
+ RandAugment	81.0	**+0.4**
+ Dropout on FC	80.7 ‡	-0.3
+ Decrease weight decay	82.2	**+1.5**
+ Squeeze-and-Excitation	82.9	**+0.7**
+ ResNet-D	83.4	**+0.5**

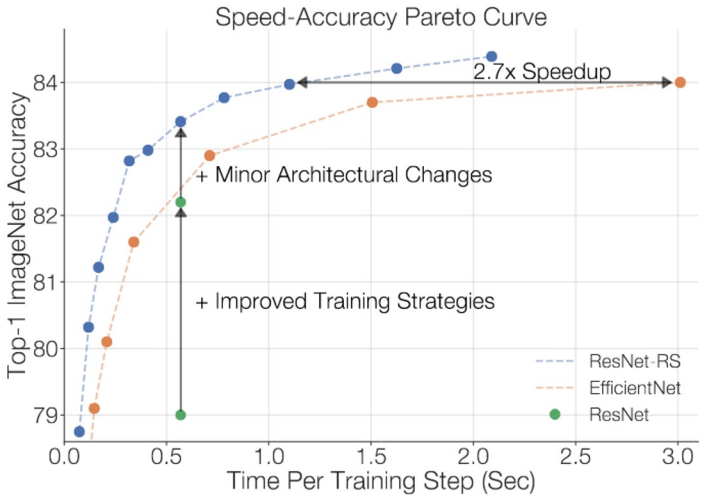
Table 1. **Additive study of the ResNet-RS training recipe.** The colors refer to **Training Methods**, **Regularization Methods** and **Architecture Improvements**. The baseline ResNet-200 was trained for the standard 90 epochs using a stepwise learning rate decay schedule. The image resolution is 256×256. All

.CV]

scaling strategies may matter more than architectural changes, and further, that the resulting ResNets match recent state-of-the-art models. We show that the best performing scaling strategy depends on the training regime and offer

raining and Scaling Strategies

D. Cubuk [1] Aravind Srinivas [2] Tsung-Yi Lin [1]
Barret Zoph [1]



Efficient Networks …

SqueezeNet: AlexNet-level Accuracy With 50x Fewer Parameters and <0.5Mb Model Size

[Iandola et al. 2017]

- Fire modules consisting of a 'squeeze' layer with 1x1 filters feeding an 'expand' layer with 1x1 and 3x3 filters
- AlexNet level accuracy on ImageNet with 50x fewer parameters
- Can compress to 510x smaller than AlexNet (0.5Mb)

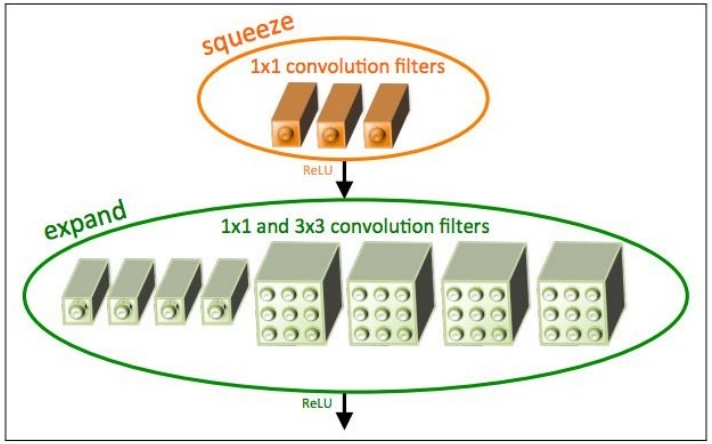


Figure copyright Iandola, Han, Moskewicz, Ashraf, Dally, Keutzer, 2017.

https://arxiv.org/pdf/1905.11946.pdf, ICML 2019

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] **Quoc V. Le** [1]

Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective *compound coefficient*. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet.

To go even further, we use neural architecture search to design a new baseline network

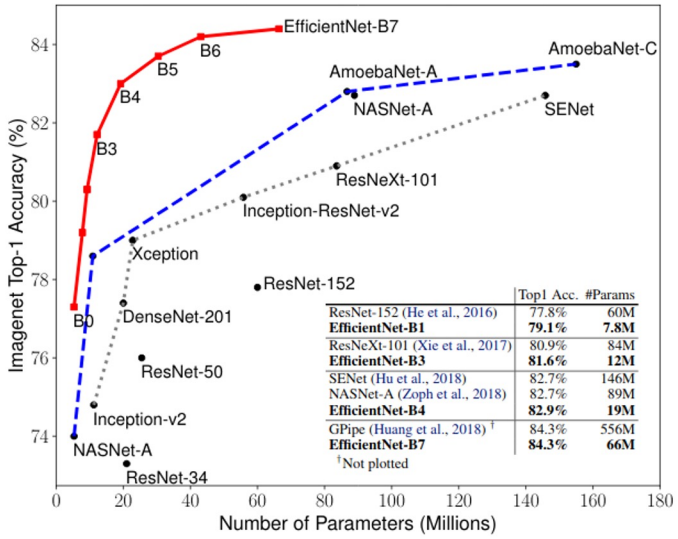
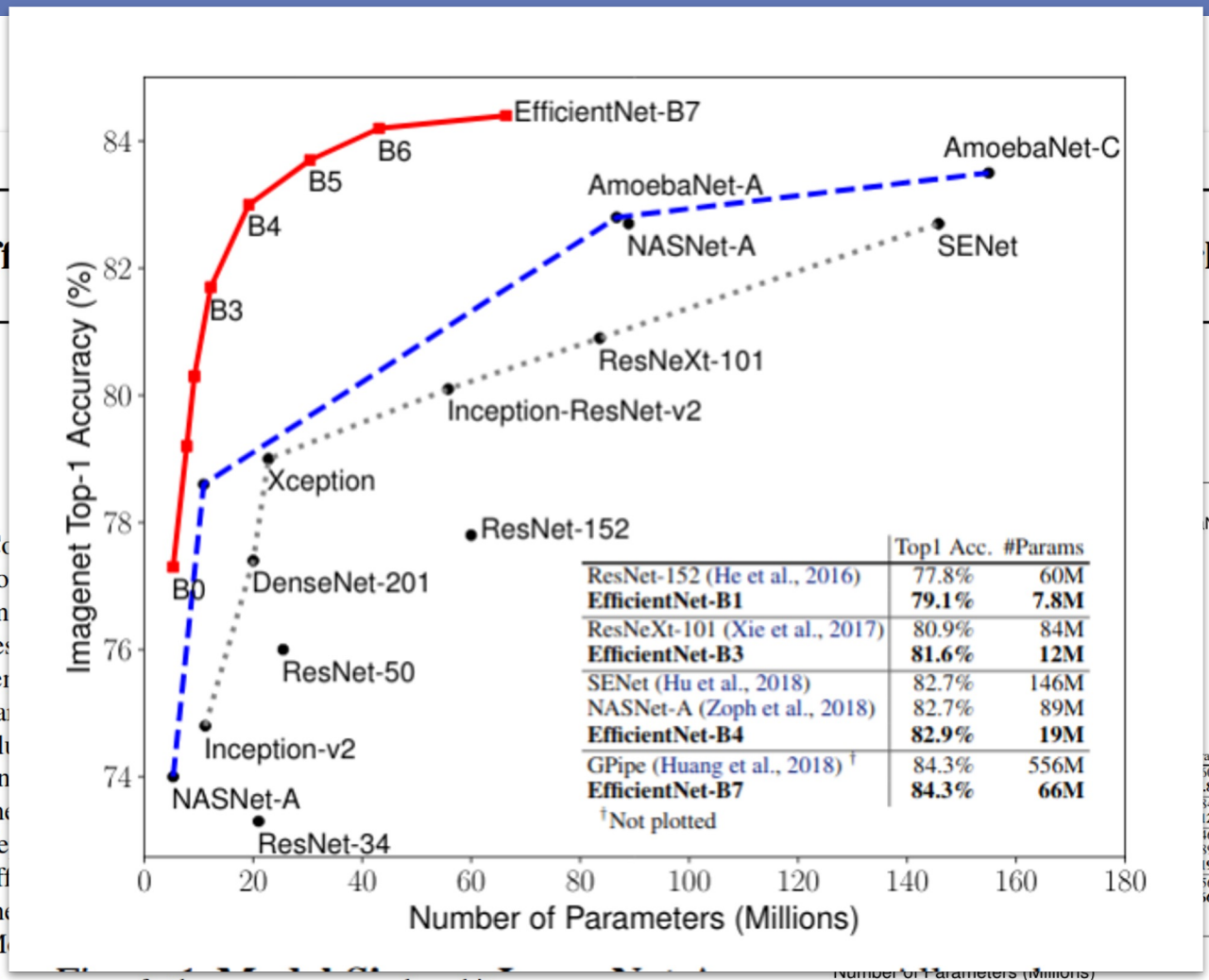


Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly out-



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are

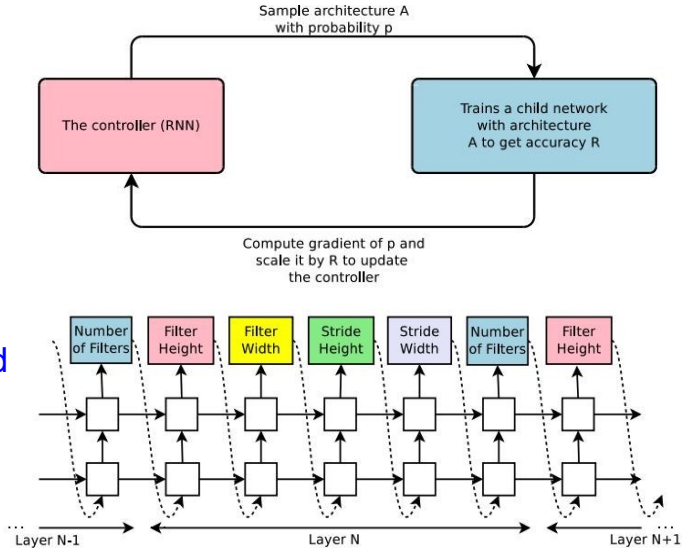
To go even further, we use neural architec-
ture search to design a new baseline network

Meta-Learning: Learning to learn network architectures

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

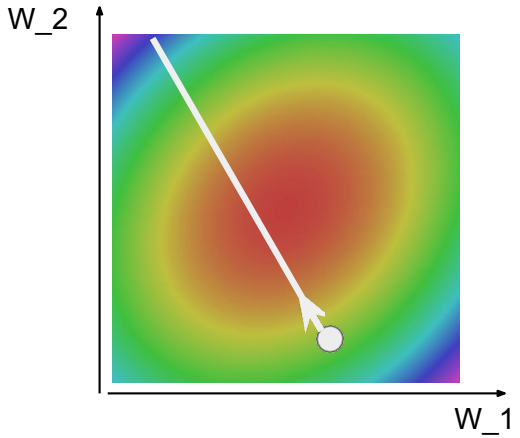
- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and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



Extra: Optimization

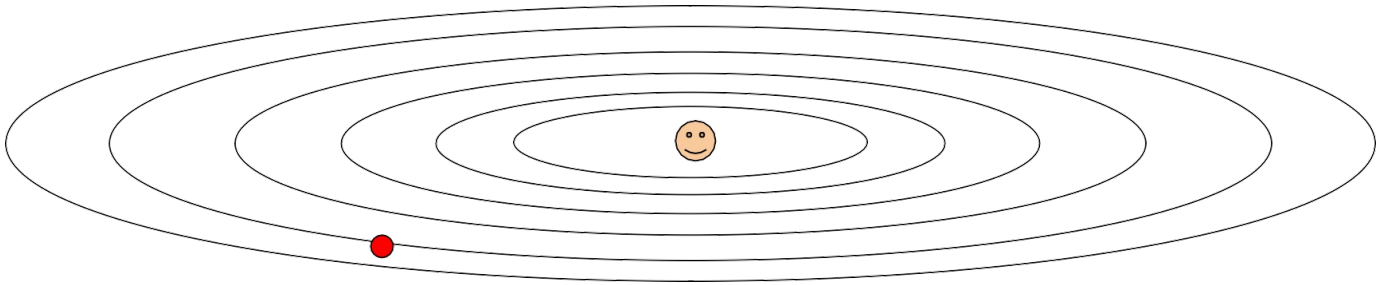
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



Optimization: Problems with SGD

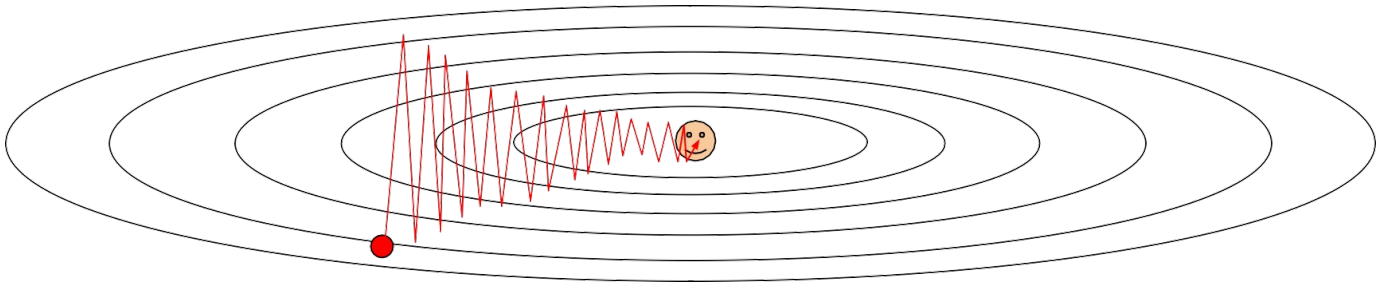
What if loss changes quickly in one direction and slowly in another? What does gradient descent do?



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another? What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction

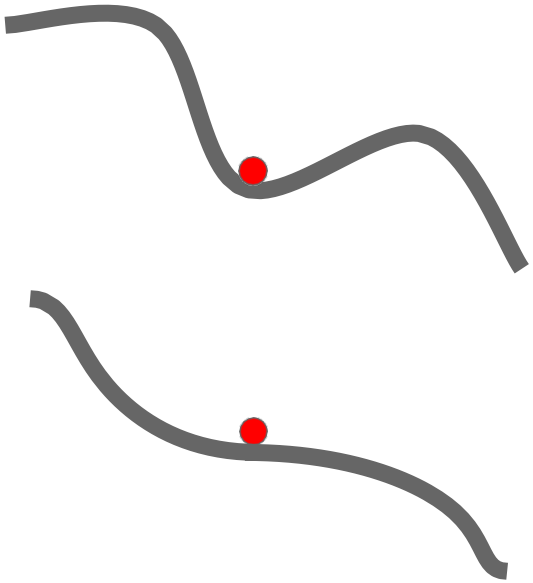


Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck

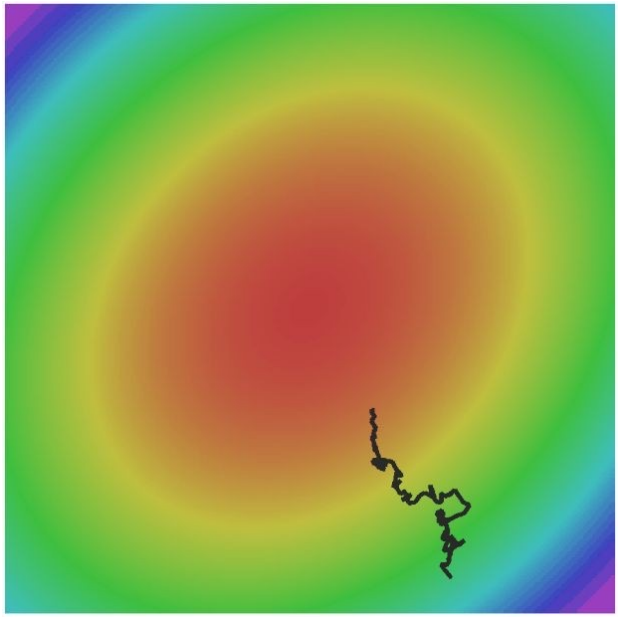


Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



Optimization: SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

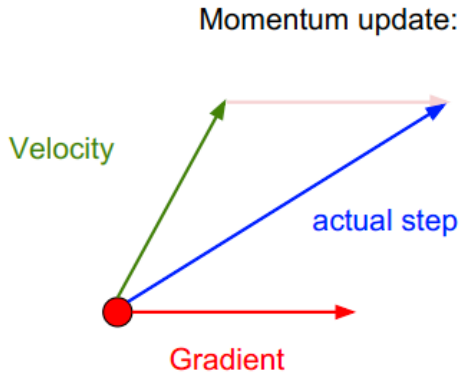
```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

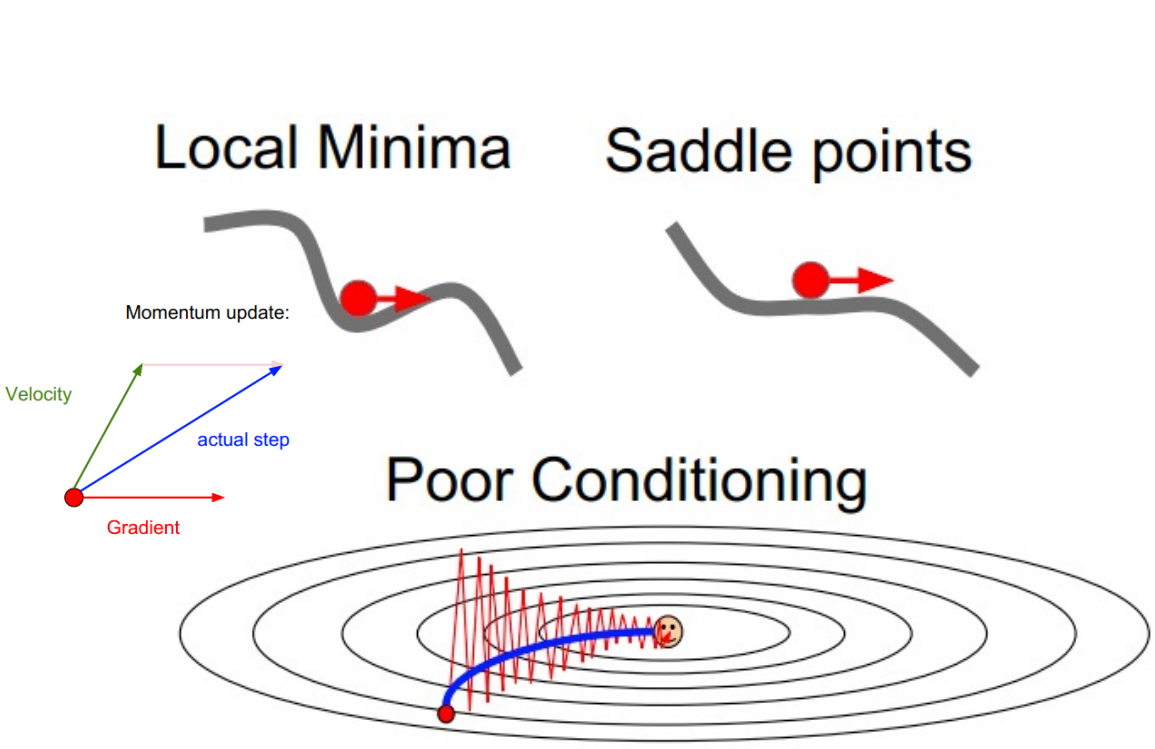
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

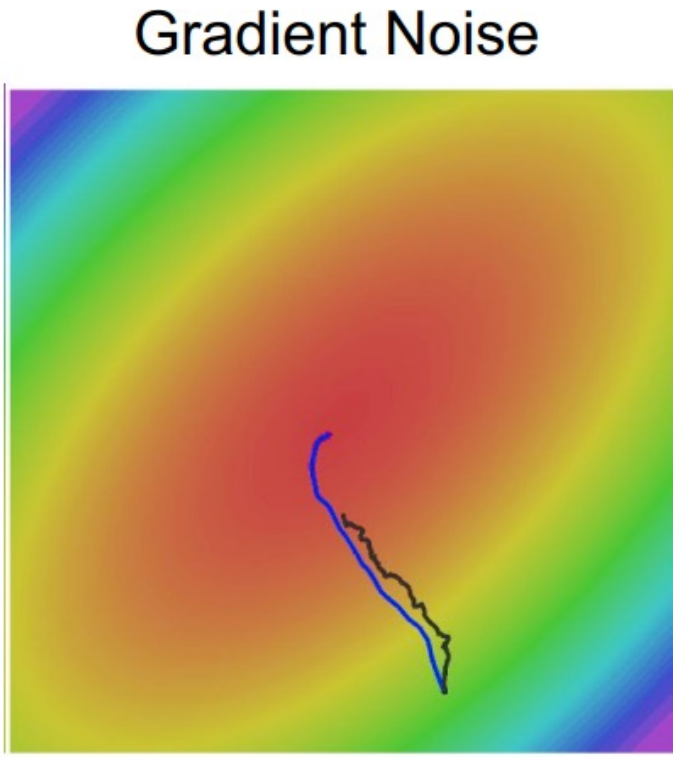


- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Optimization: SGD + Momentum



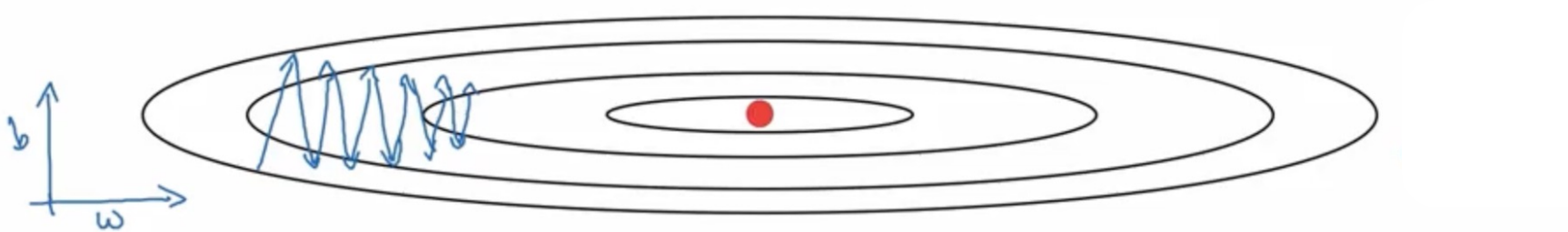


Optimization: RMSProp

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

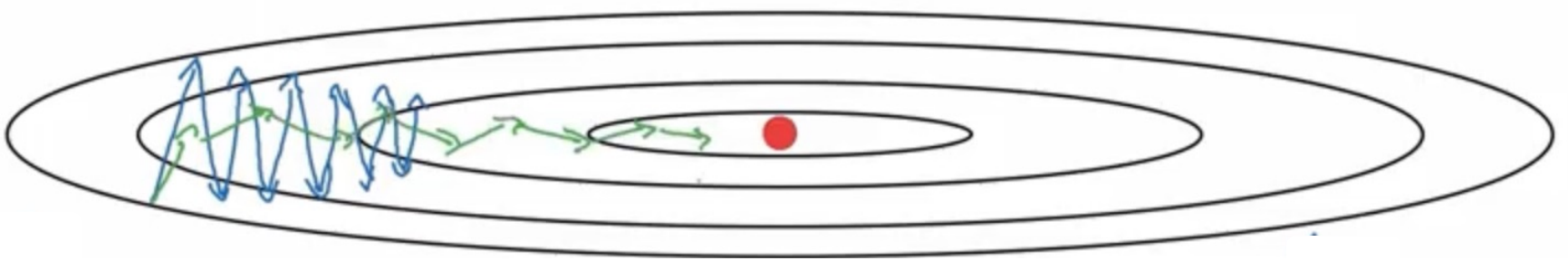
Tieleman and Hinton, 2012



Adapted from Andrew Ng - Coursera

High *b* → High *grad_squared* → High *Denominator* → Decrease magnitude of *dx (gradient)*
Low *w* → Low *grad_squared* → Low *Denominator* → Keep magnitude similar to *dx (gradient)*

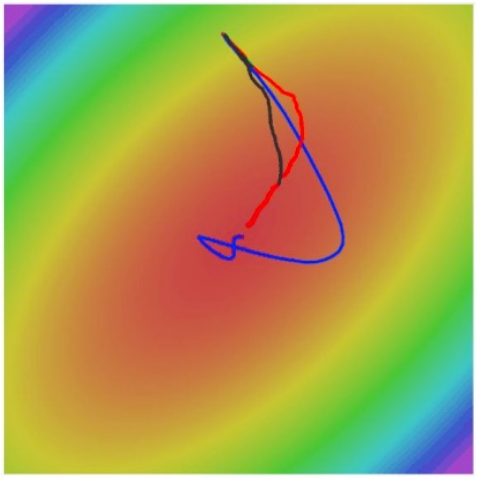
Optimization: RMSProp



SGD
RMSProp

Adapted from Andrew Ng - Coursera

Tieleman and Hinton, 2012



Fei-Fei Li, Andrej Karpathy, Justin Johnson, Serena Yeung

Optimization: Adam

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

RMSProp

beta_1 = rho
first_moment = v_x

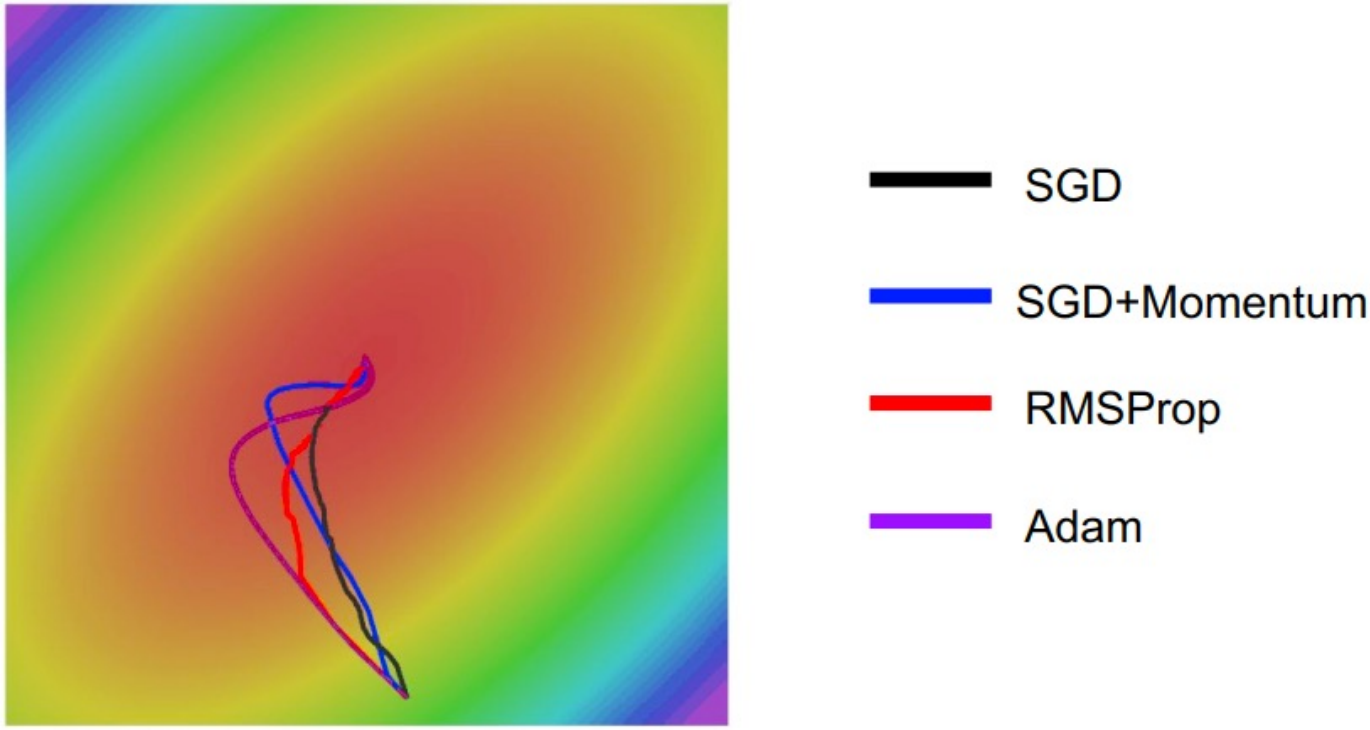
beta2 = decay_rate
second_moment = grad_squared

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9,
beta2 = 0.999, and learning_rate = 1e-3 or 5e-4
is a great starting point for many models!

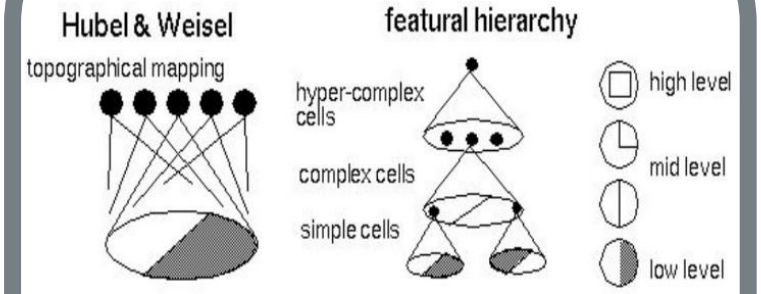
Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Optimization: Adam

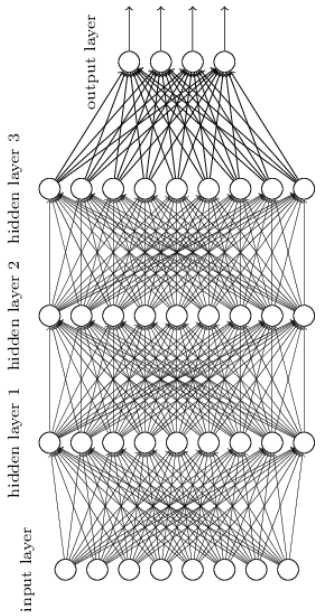


Extra: Understanding CNNs

Recall: Biological analog



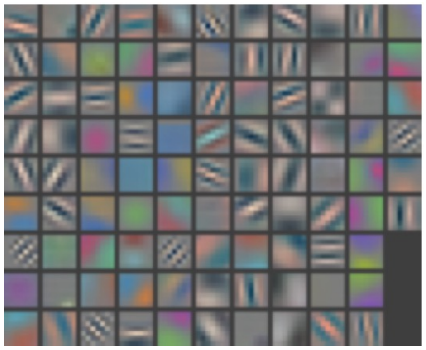
Hubel and Weisel's architecture

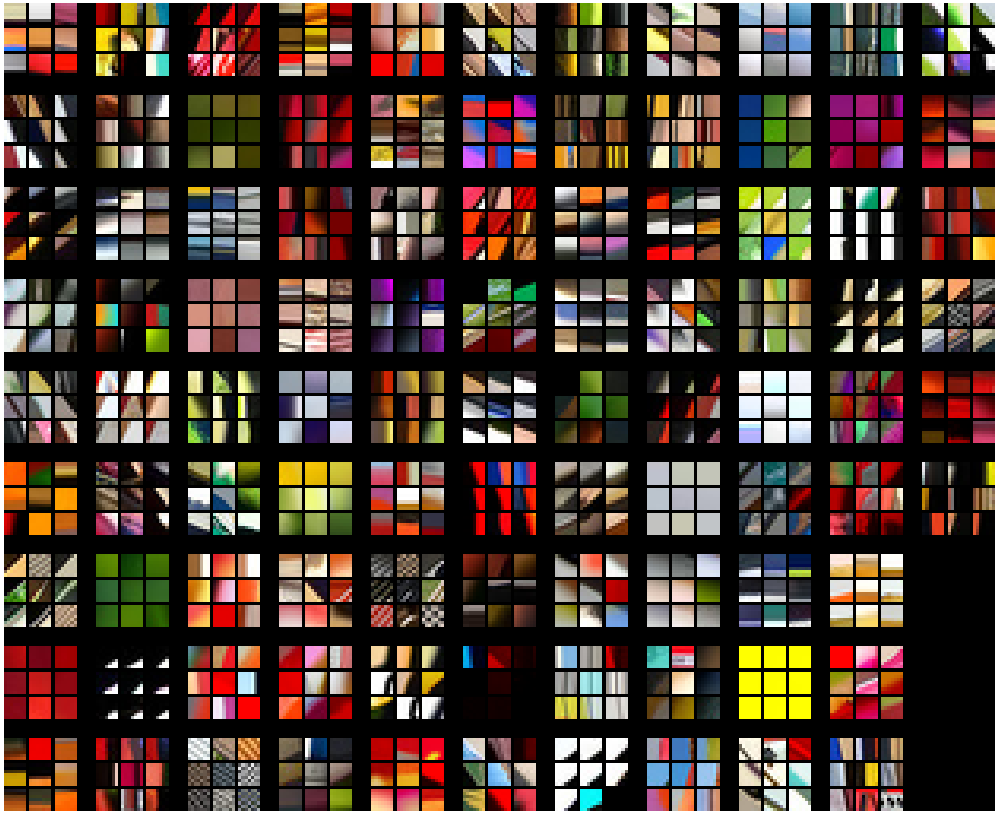


Multi-layer neural network

Adapted from Jia-bin Huang

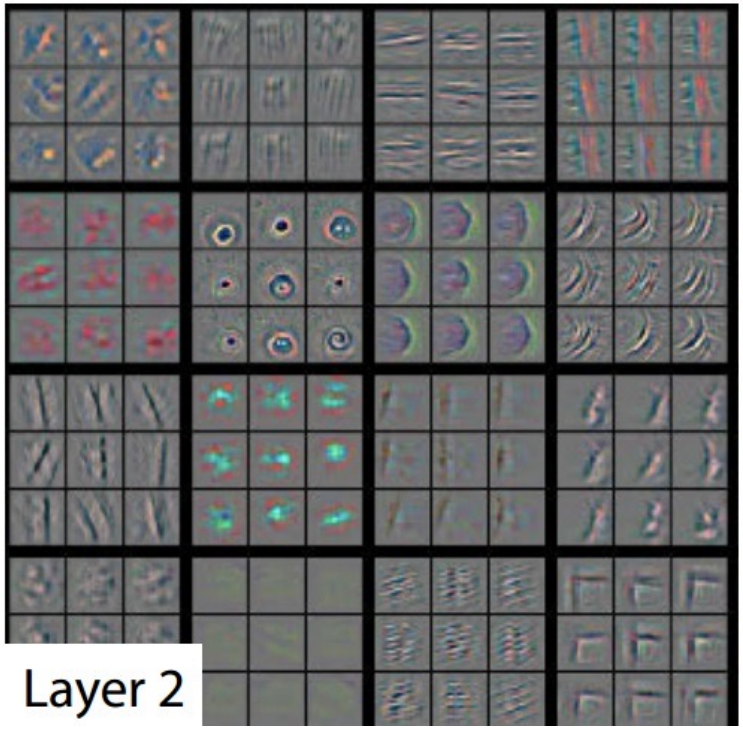
Layer 1

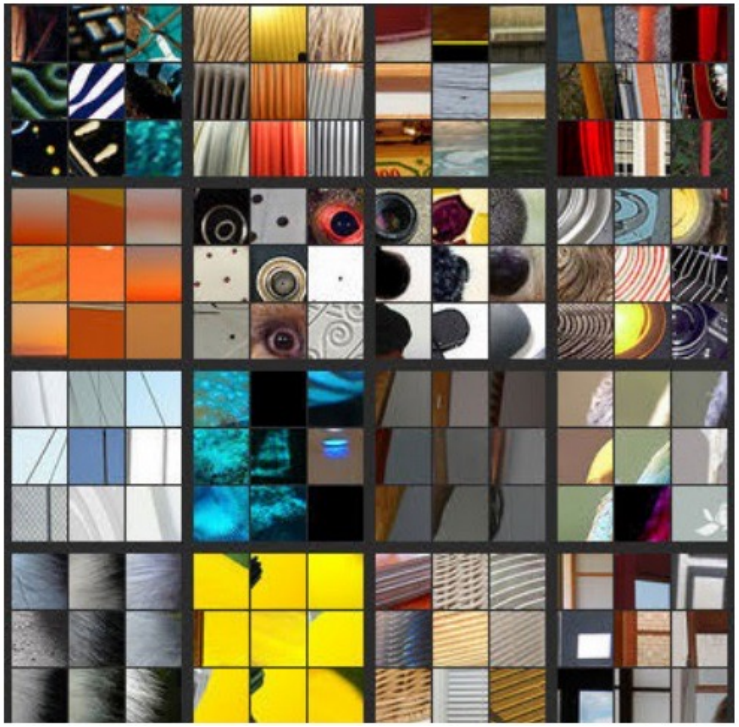




Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 2

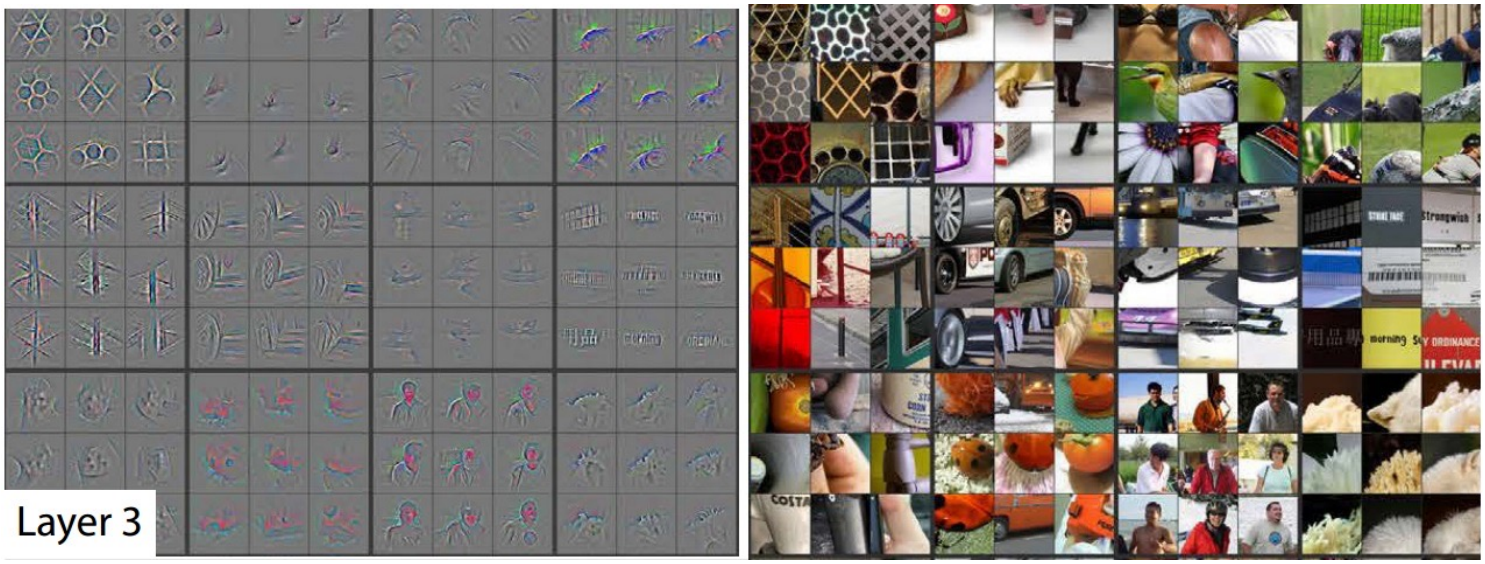




- Activations projected down to pixel level via decovolution
- Patches from validation images that give maximal activation of a given feature map

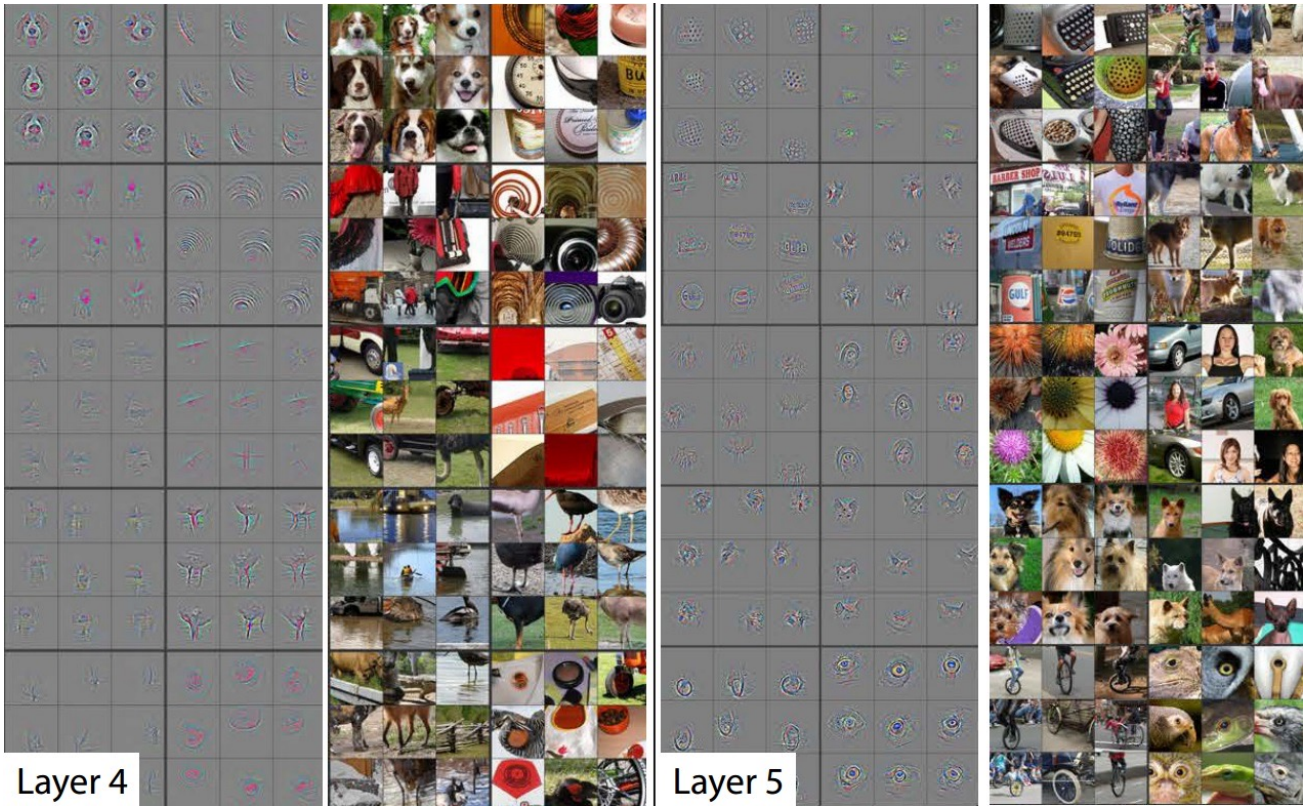
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 3



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 4 and 5



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

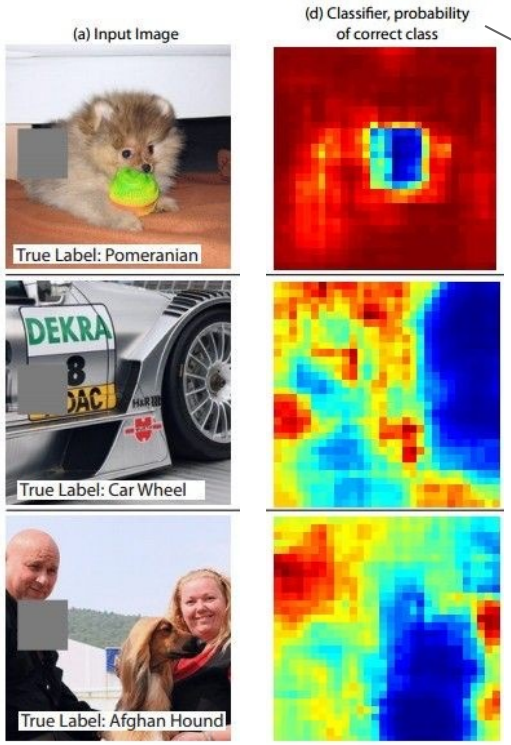
Occlusion experiments

(d) Classifier, probability of correct class

(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]

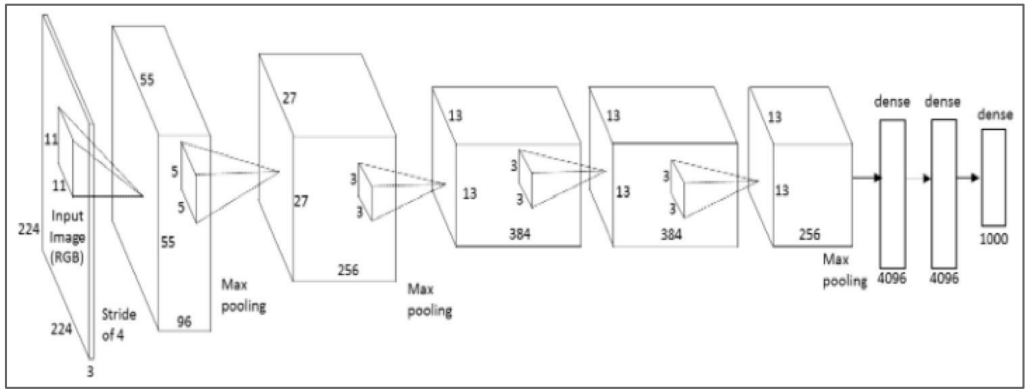
Occlusion experiments



(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]

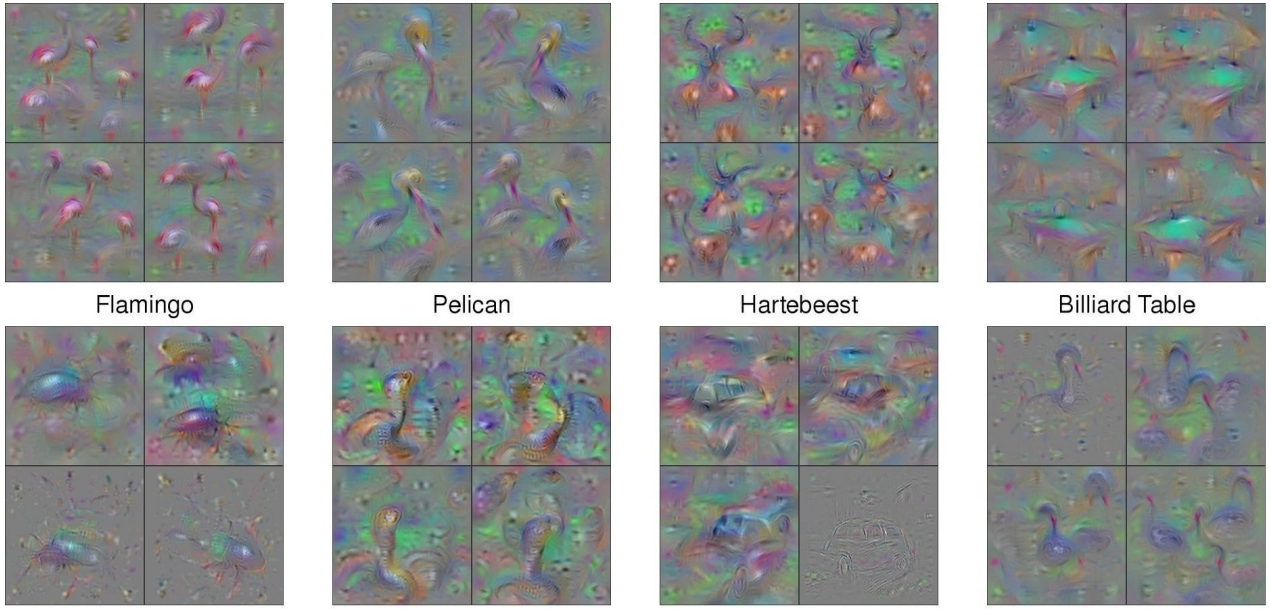
What image maximizes a class score?



Repeat:

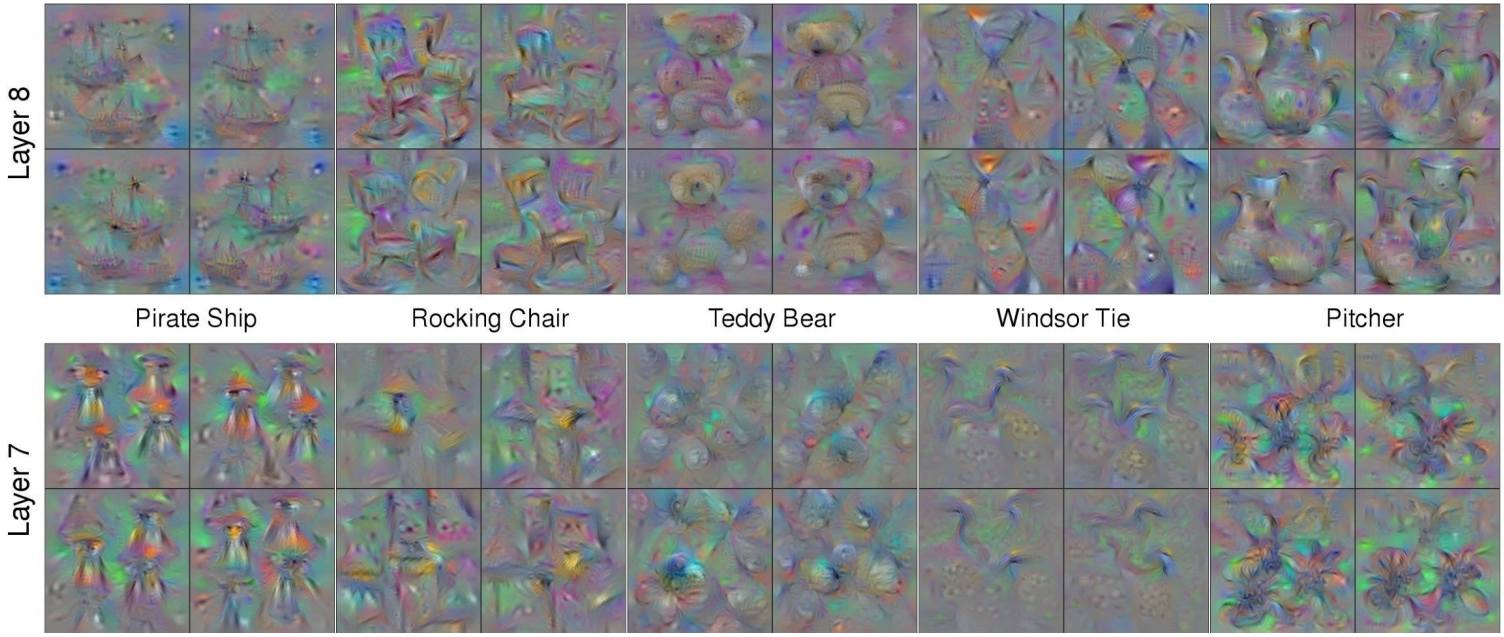
1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an "image update"

What image maximizes a class score?

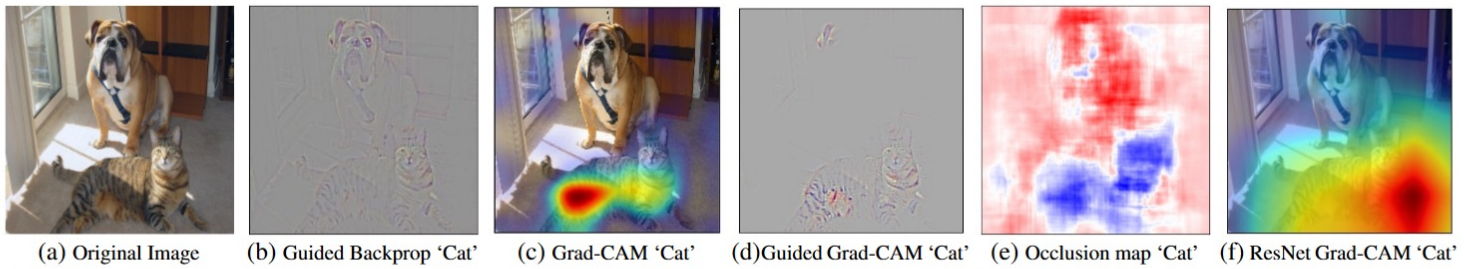


[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015]
http://yosinski.com/deepvis

What image maximizes a class score?



GradCAM



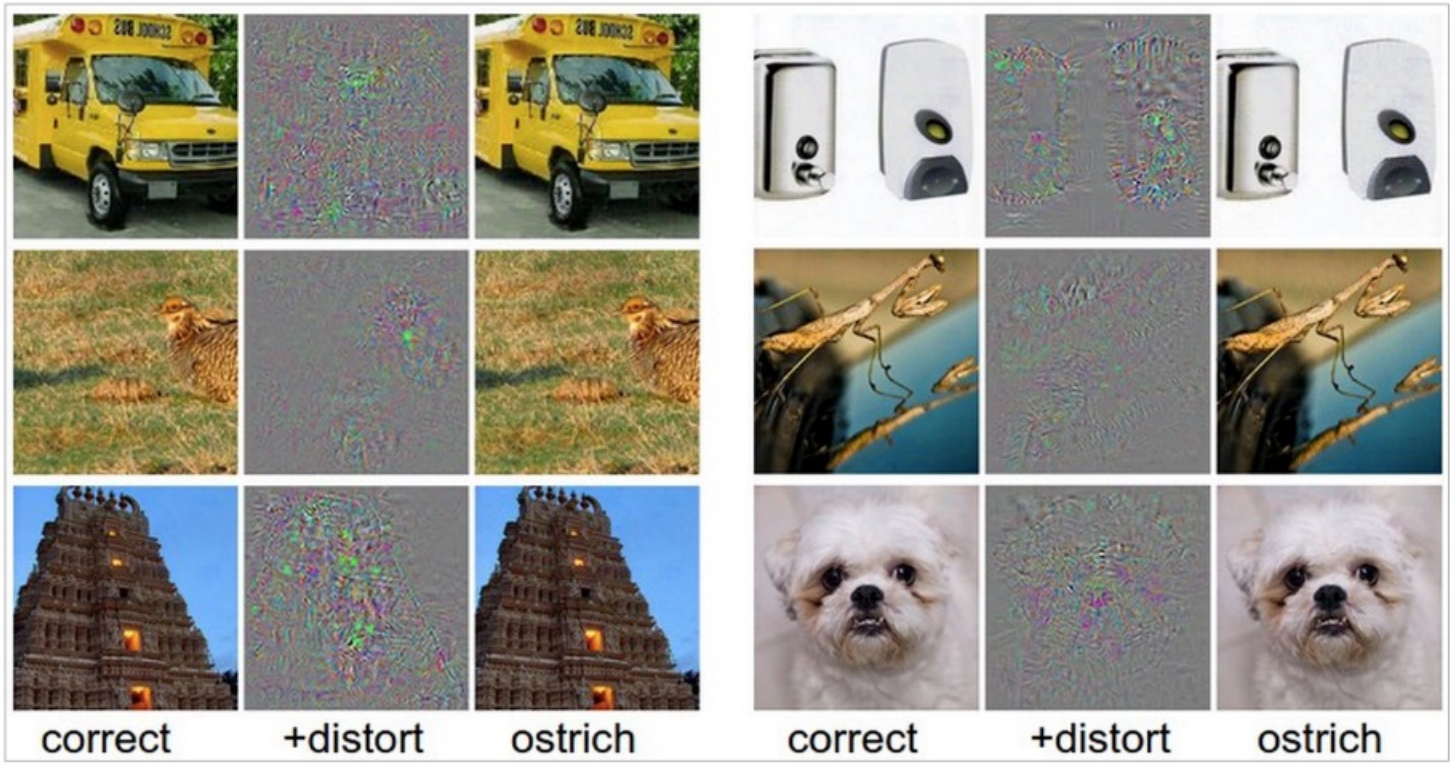
(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d)Guided Grad-CAM 'Cat' (e) Occlusion map 'Cat' (f) ResNet Grad-CAM 'Cat'

(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (j)Guided Grad-CAM 'Dog' (k) Occlusion map 'Dog' (l)ResNet Grad-CAM 'Dog'

Fig. 1: (a) Original image with a cat and a dog. (b-f) Support for the cat category according to various visualizations for VGG-16 and ResNet. (b) Guided Backpropagation [53]: highlights all contributing features. (c, f) Grad-CAM (Ours): localizes class-discriminative regions, (d) Combining (b) and (c) gives Guided Grad-CAM, which gives high-resolution class-discriminative visualizations. Interestingly, the localizations achieved by our Grad-CAM technique, (c) are very similar to results from occlusion sensitivity (e), while being orders of magnitude cheaper to compute. (f, l) are Grad-CAM visualizations for ResNet-18 layer. Note that in (c, f, i, l), red regions corresponds to high score for class, while in (e, k), blue corresponds to evidence for the class. Figure best viewed in color.

Selvaraju et al. ICCV 2017, IJCV 2019

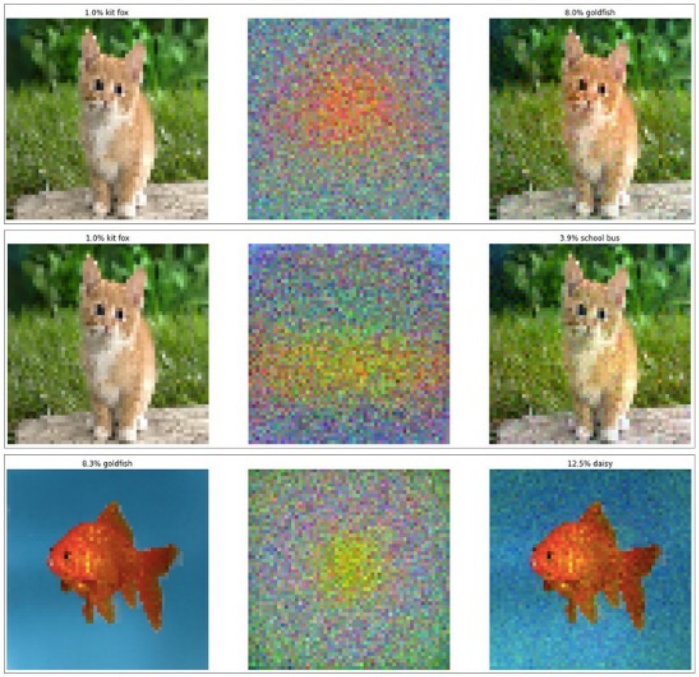
Breaking CNNs



Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

Intriguing properties of neural networks [Szegedy ICLR 2014]

Fooling a linear classifier



Fooled linear classifier: The starting image (left) is classified as a kit fox. That's incorrect, but then what can you expect from a linear classifier? However, if we add a small amount "goldfish" weights to the image (top row, middle), suddenly the classifier is convinced that it's looking at one with high confidence. We can distort it with the school bus template instead if we wanted to.

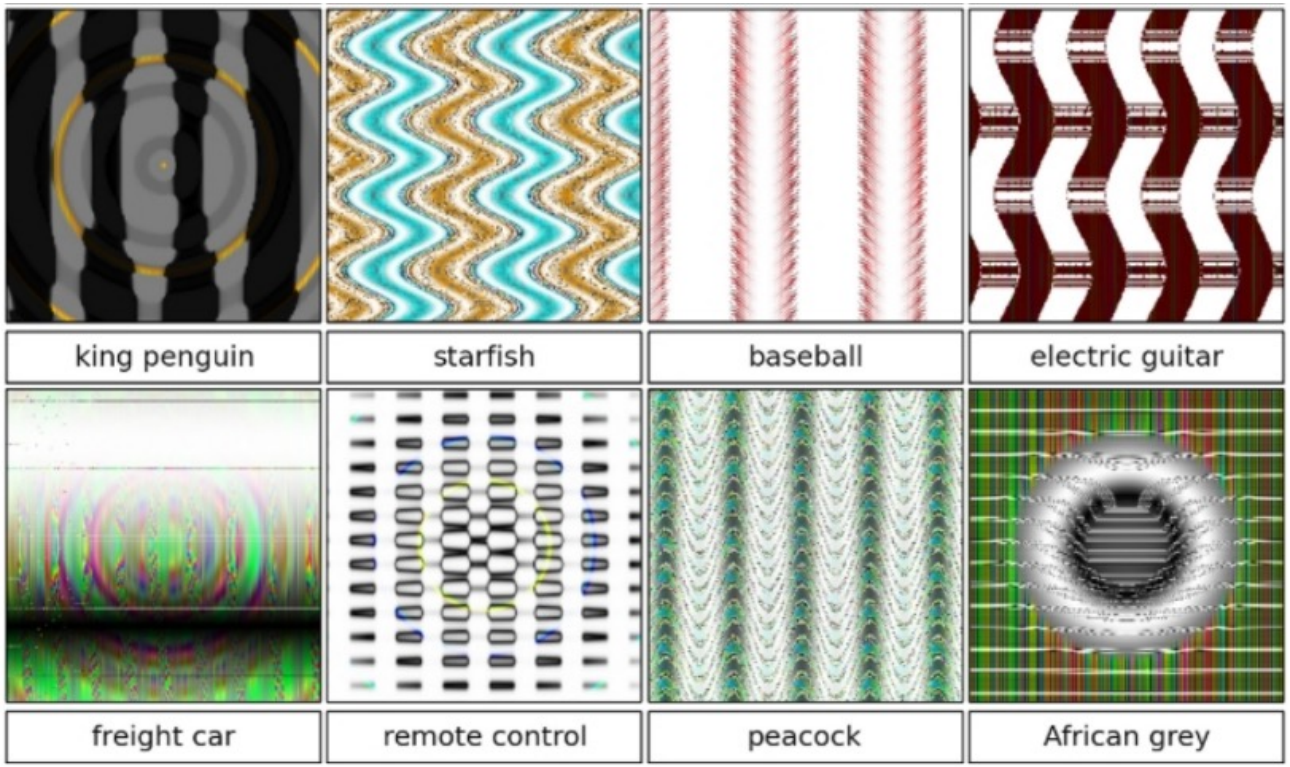
How to fool?

Add a small multiple of the weight vector to the training example:

$\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{w}$

http://karpathy.github.io/2015/03/30/breaking-convnets/

Breaking CNNs



Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen et al. CVPR 2015]

Shape vs Texture

(a) Texture image
81.4% **Indian elephant**
10.3% indri
8.2% black swan

(b) Content image
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict
63.9% **Indian elephant**
26.4% indri
9.6% black swan

Figure 1: Classification of a standard ResNet-50 of **(a)** a texture image (elephant skin: only texture cues); **(b)** a normal image of a cat (with both shape and texture cues), and **(c)** an image with a texture-shape cue conflict, generated by style transfer between the first two images.

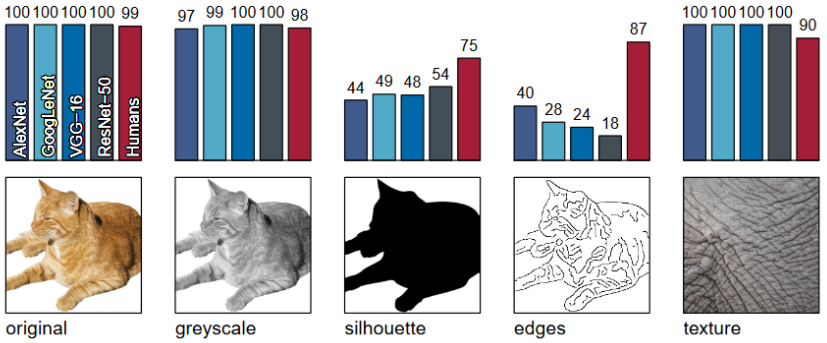


Figure 2: Accuracies and example stimuli for five different experiments without cue conflict.

ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness [Geirhos et al., ICLR 2019]