Introduction to convnets

CS 20: TensorFlow for Deep Learning Research
Lecture 6
1/31/2017

Agenda

Computer Vision

Convolutional Neural Networks

Convolution

Pooling

Feature Visualization

Slides adapted from Justin Johnson

Used with permission.

Convolutional Neural Networks: Deep Learning with Images

Computer Vision - A bit of history

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

Goals - General

The primary goal of the project is to construct a system of programs which will divide a vidisector picture into regions such as

likely objects

likely background areas

chaos.

We shall call this part of its operation FIGURE-GROUND analysis.

It will be impossible to do this without considerable analysis of shape and surface properties, so FIGURE-GROUND analysis is really insepar-able in practice from the second goal which is REGION DESCRIPTION.

The final goal is OBJECT IDENTIFICATION which will actually name objects by matching them with a vocabulary of known objects.

https://dspace.mit.edu/bitstream/handle/1721.1/6125/AIM-100.pdf

Computer Vision - A bit of history

https://xkcd.com/1425/

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

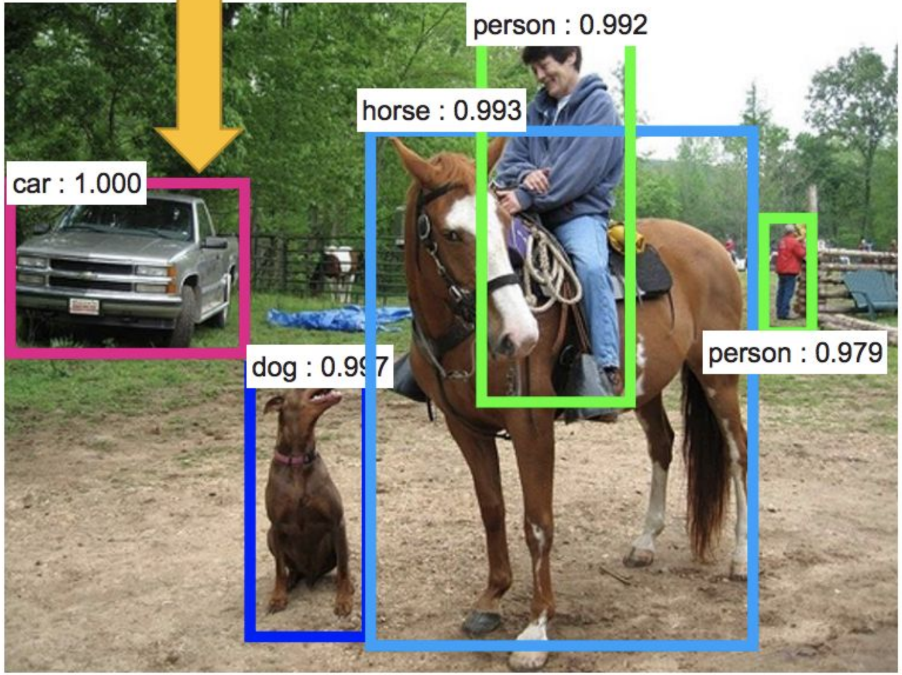
✗

Russakovsky et al. arXiv, 2014

Object Detection = What, and Where

Localization
Where?

Recognition
What?



Object Segmentation

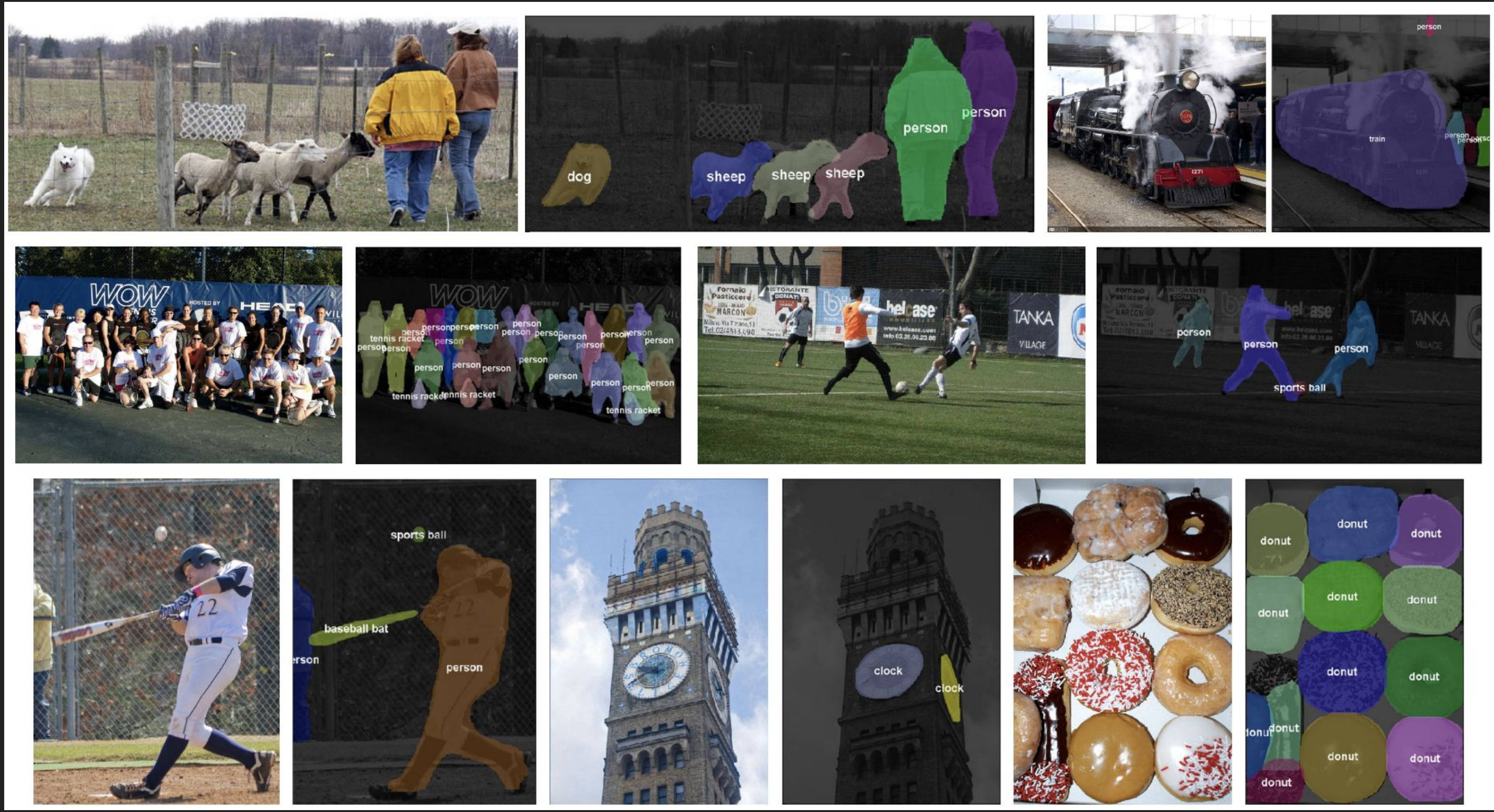
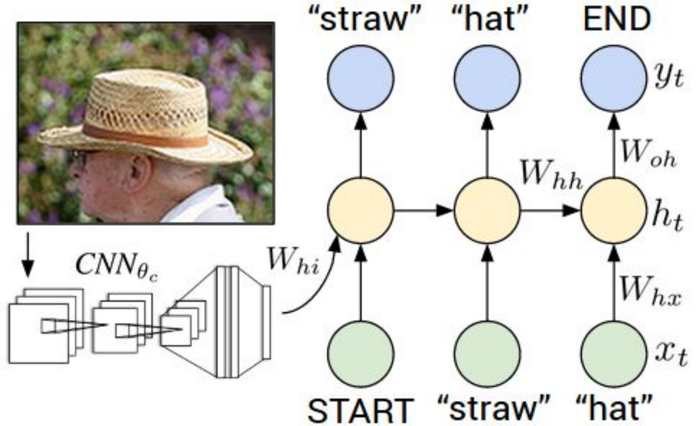


Figure credit: Dai, He, and Sun, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, CVPR 2016

Pose Estimation

Figure credit: Cao et al, “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”, arXiv 2016

Image Captioning



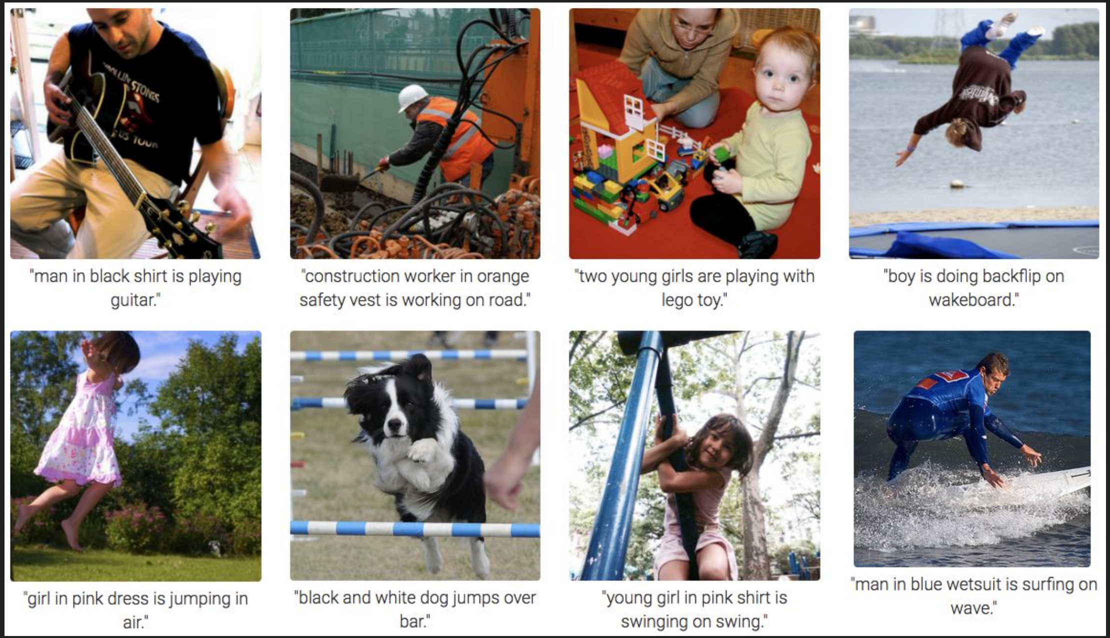


Figure credit: Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Dense Image Captioning

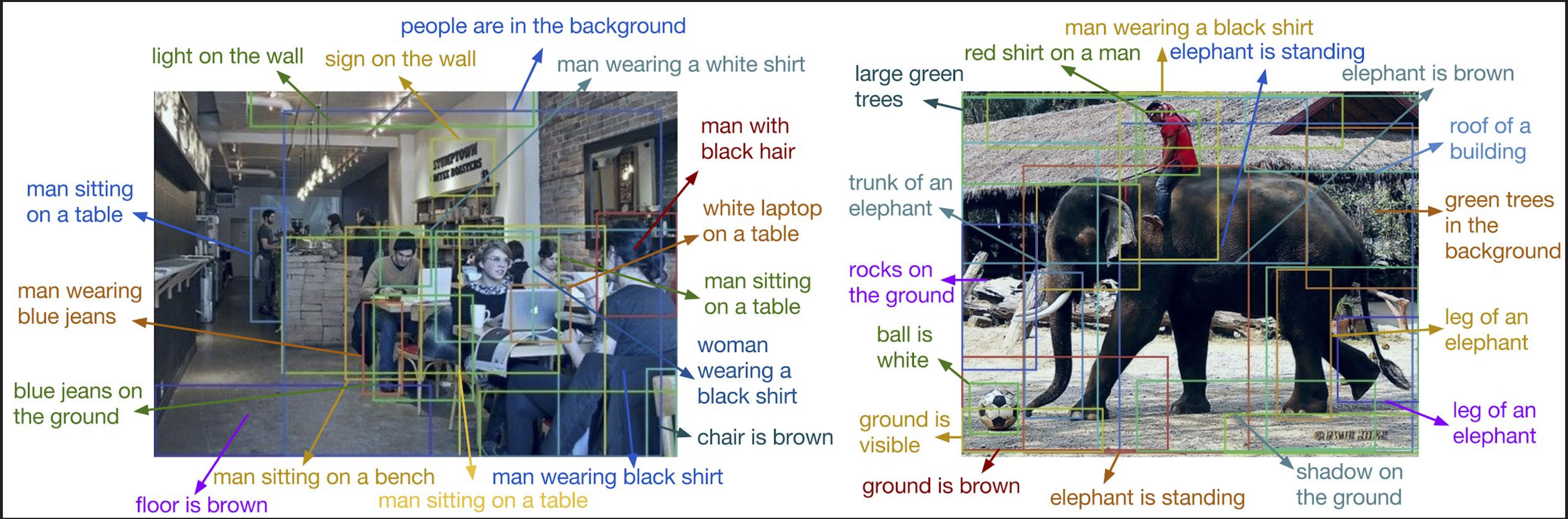


Figure credit: Johnson*, Karpathy*, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016

Visual Question Answering

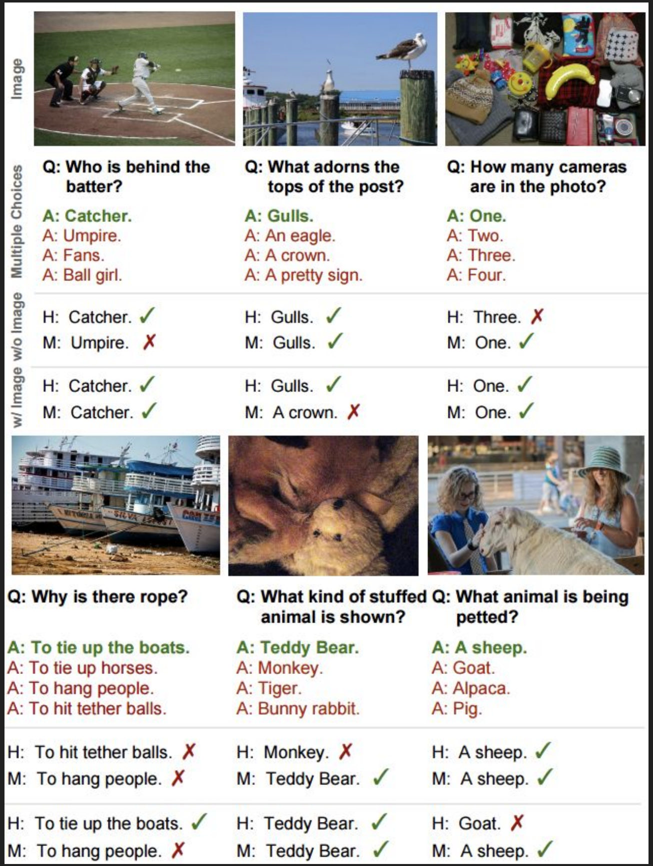


Figure credit: Agrawal et al, "VQA: Visual Question Answering", ICCV 2015 (left)
Zhu et al, "Visual7W: Grounded Question Answering in Images", CVPR 2016 (right)

Image Super-resolution

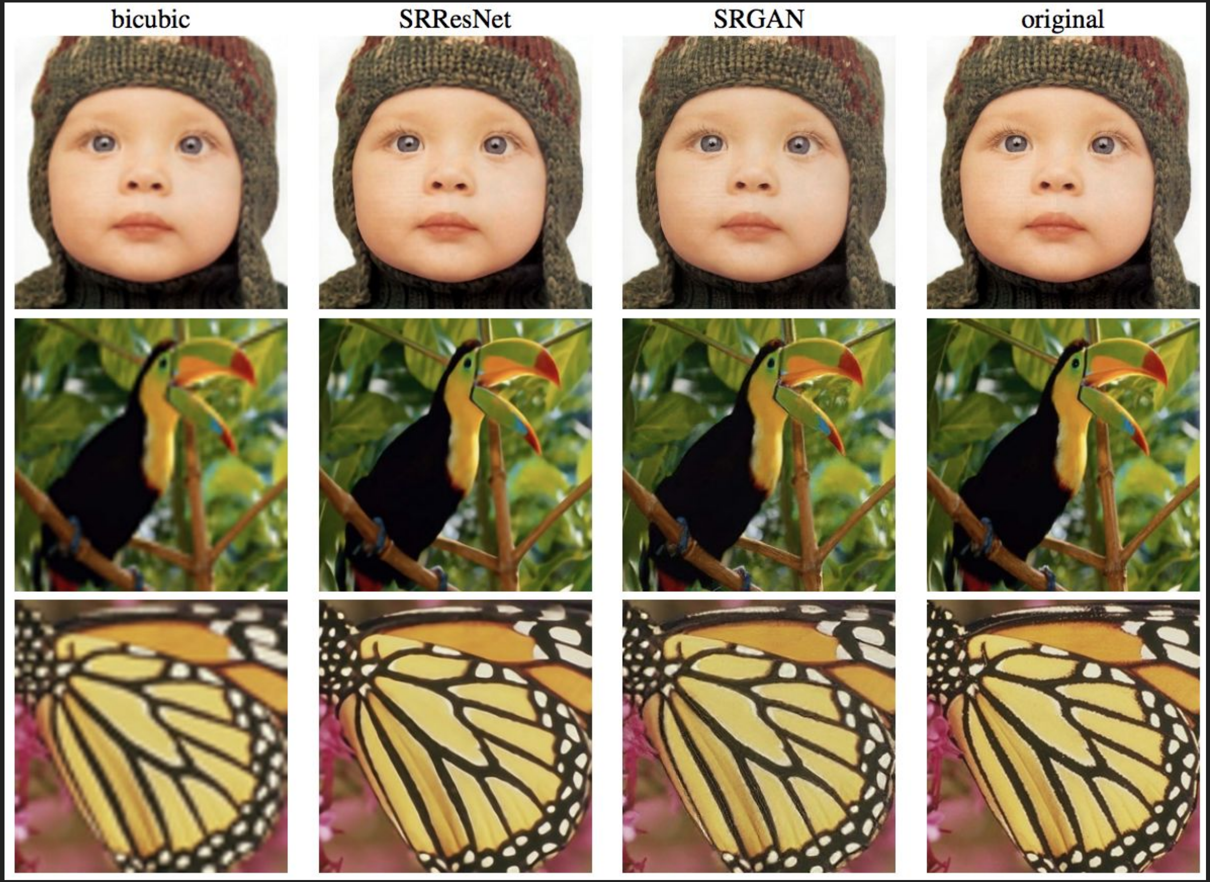
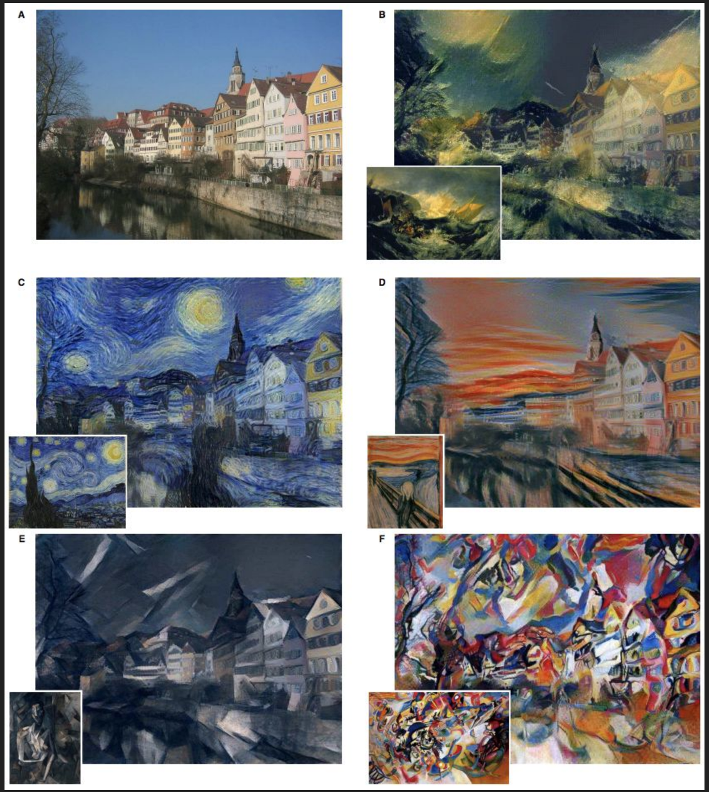


Figure credit: Ledig et al, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, arXiv 2016

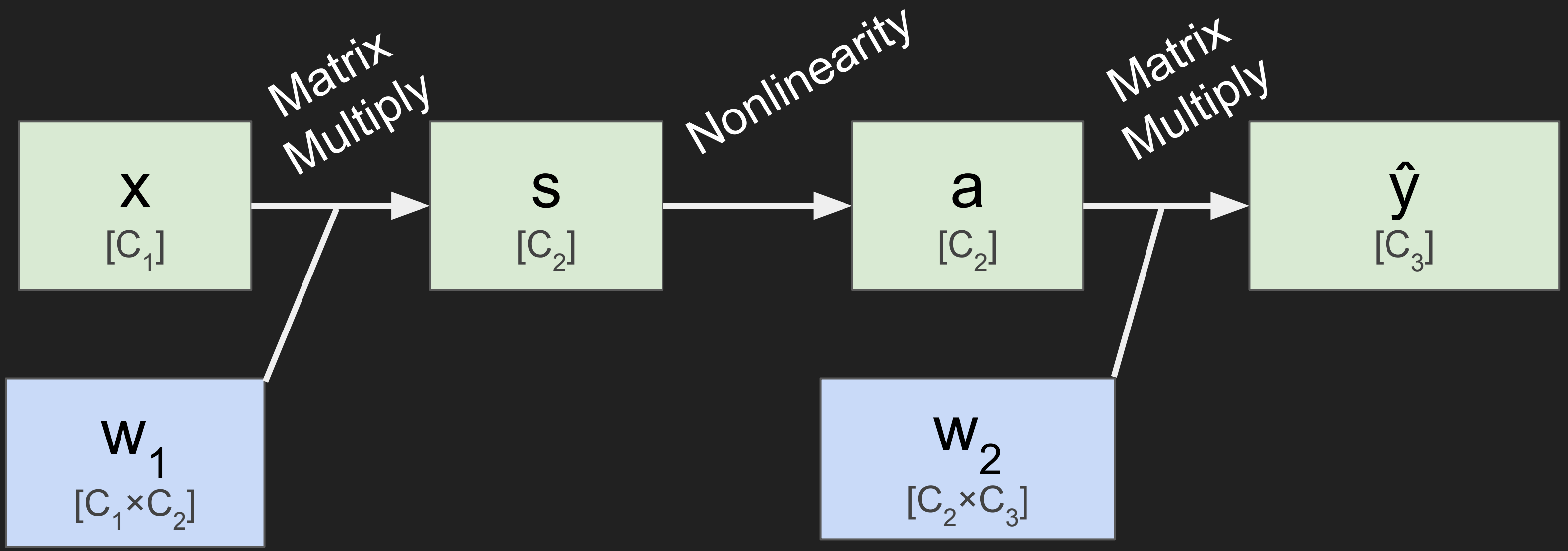
Art generation



Gatys, Ecker, and Bethge, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016 (left)
Mordvintsev, Olah, and Tyka, “Inceptionism: Going Deeper into Neural Networks” (upper right)
Johnson, Alahi, and Fei-Fei: “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016 (bottom left)

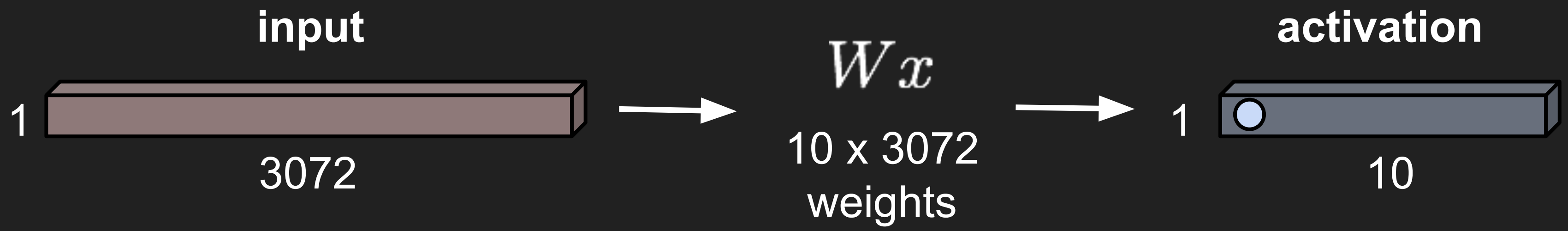
Convolutional Neural Networks

Recall: fully connected neural network

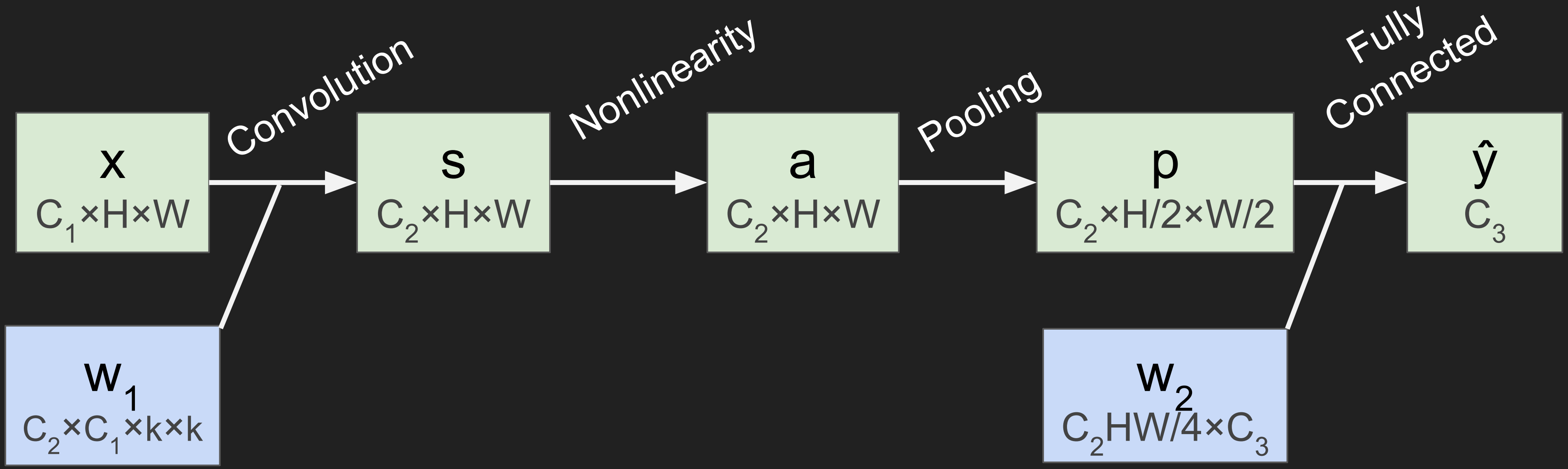


Recall: fully connected neural network

32x32x3 image -> stretch to 3072 x 1



Convolutional Neural Network



Convolution

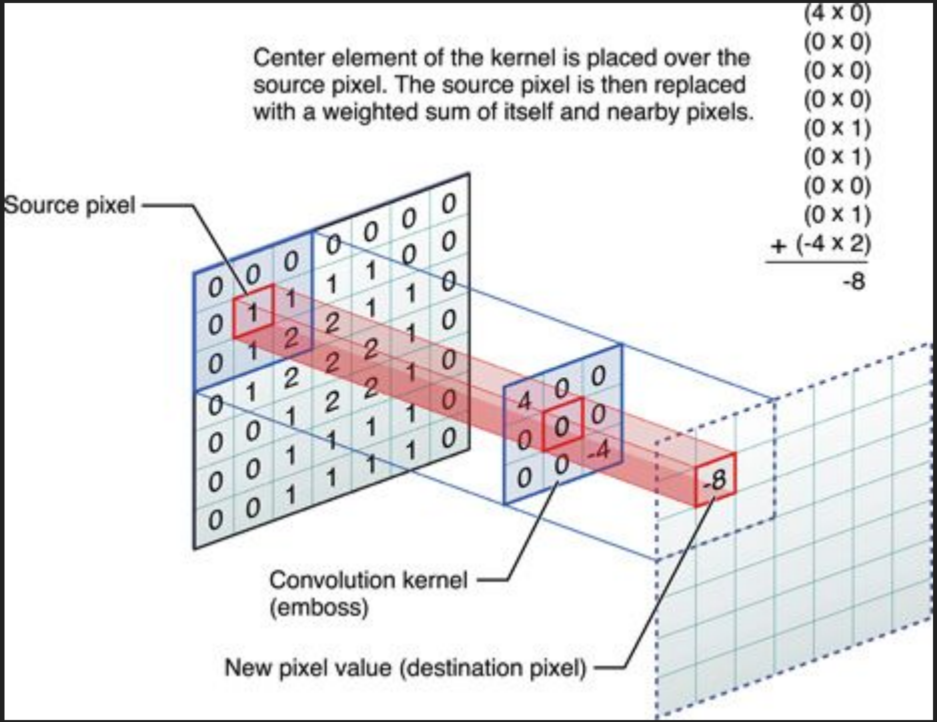
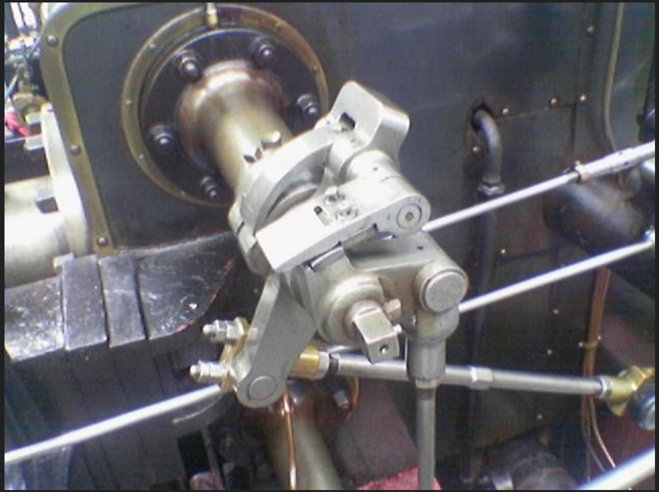
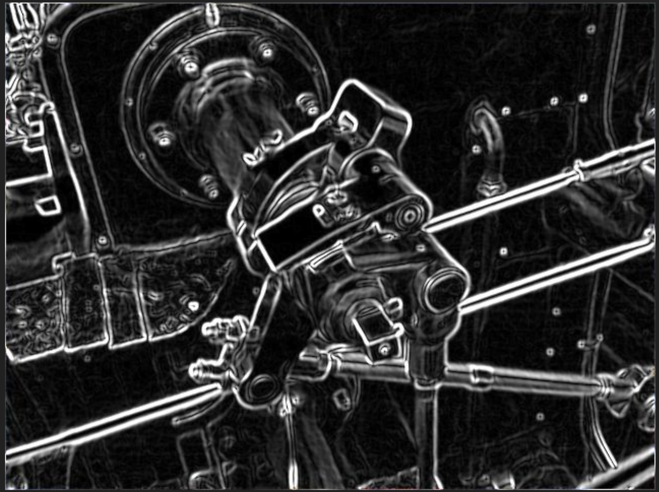


Image courtesy Apple

Convolving “filters” is not a new idea

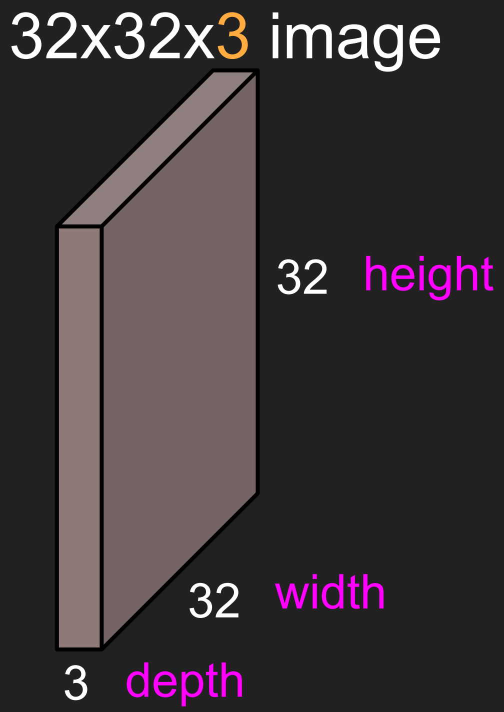




Sobel operator:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

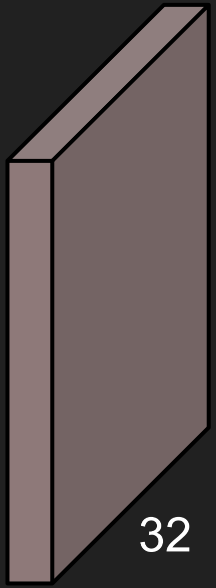
Convolution Layer



Slide credit: CS231n Lecture 7

Convolution Layer

32x32x3 image



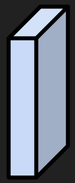
32 height

32 width

3 depth

Filters always extend the full depth of the input volume

5x5x3 filter



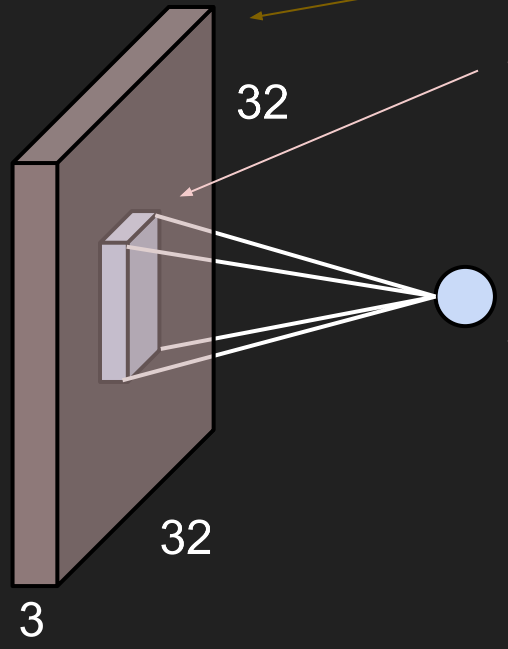
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Slide credit: CS231n Lecture 7

Convolution Layer

32x32x3 image

5x5x3 filter w

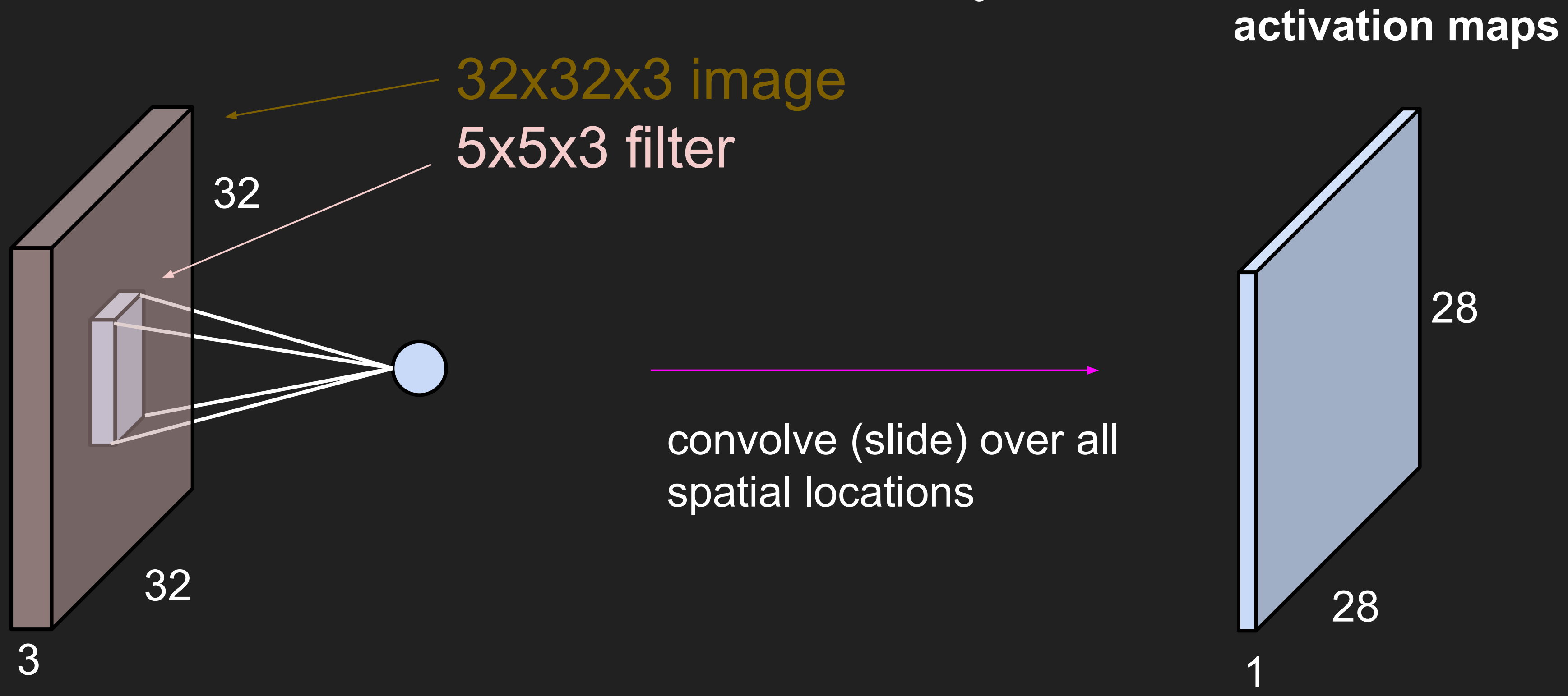


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

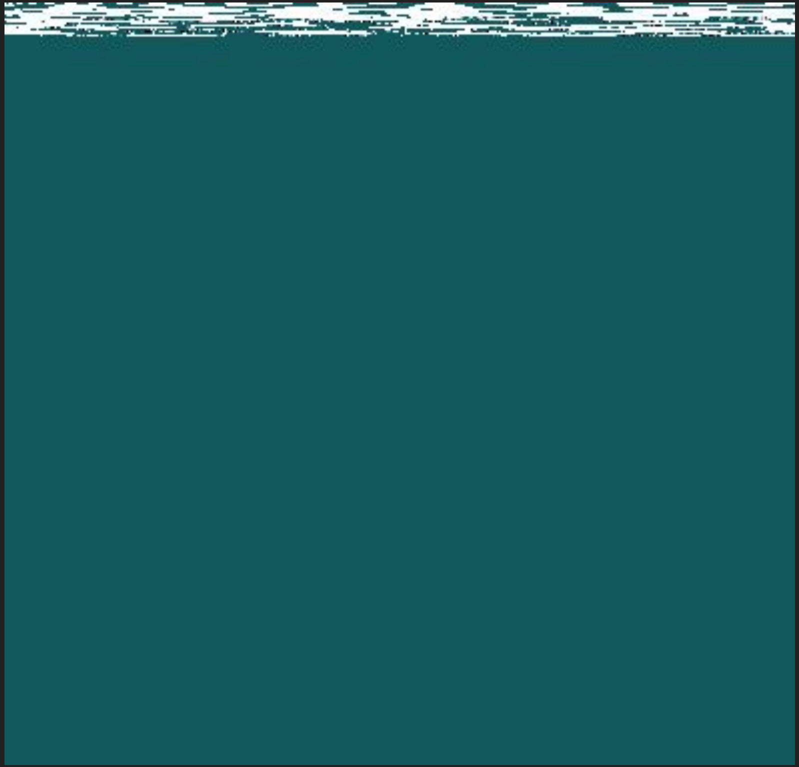
$$w^T x + b$$

Slide credit: CS231n Lecture 7

Convolution Layer



Slide credit: CS231n Lecture 7



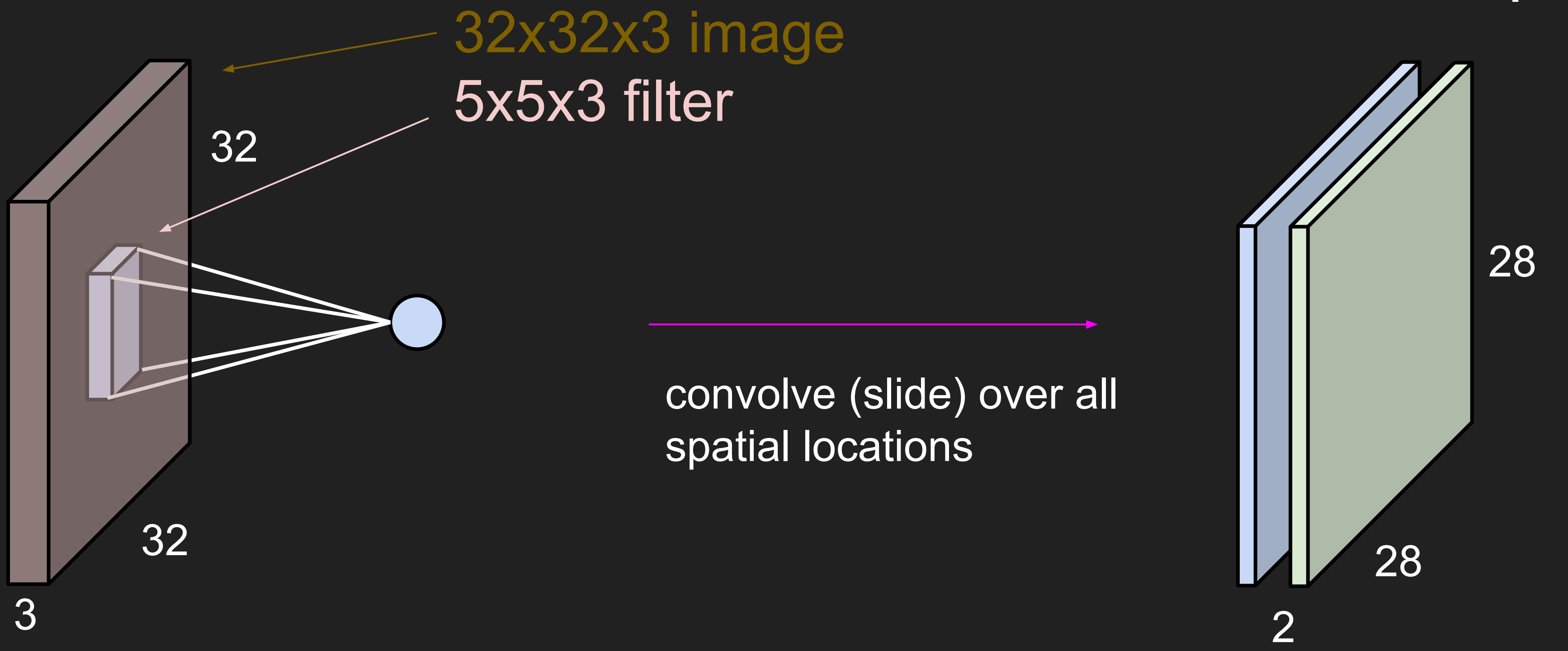
Output

Filter

Input

Padding

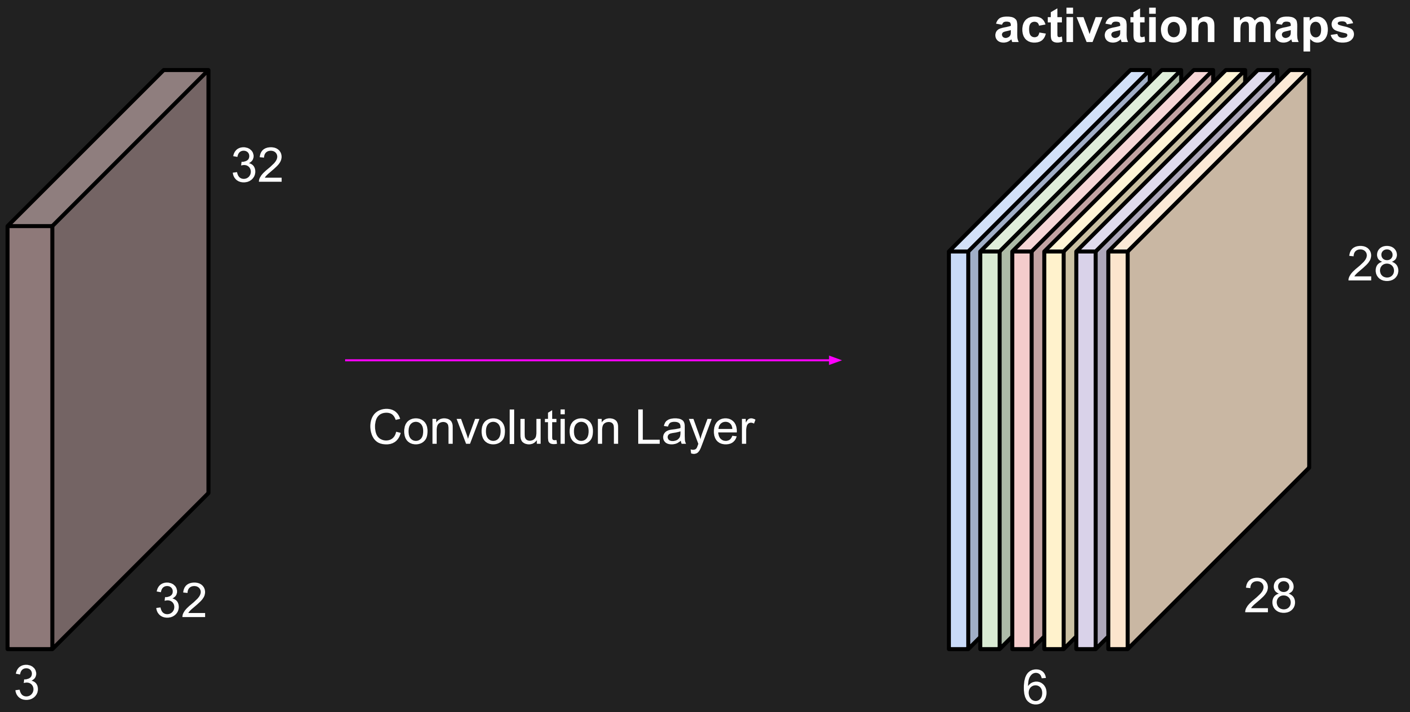
Convolution Layer



consider a second, green filter

Slide credit: CS231n Lecture 7

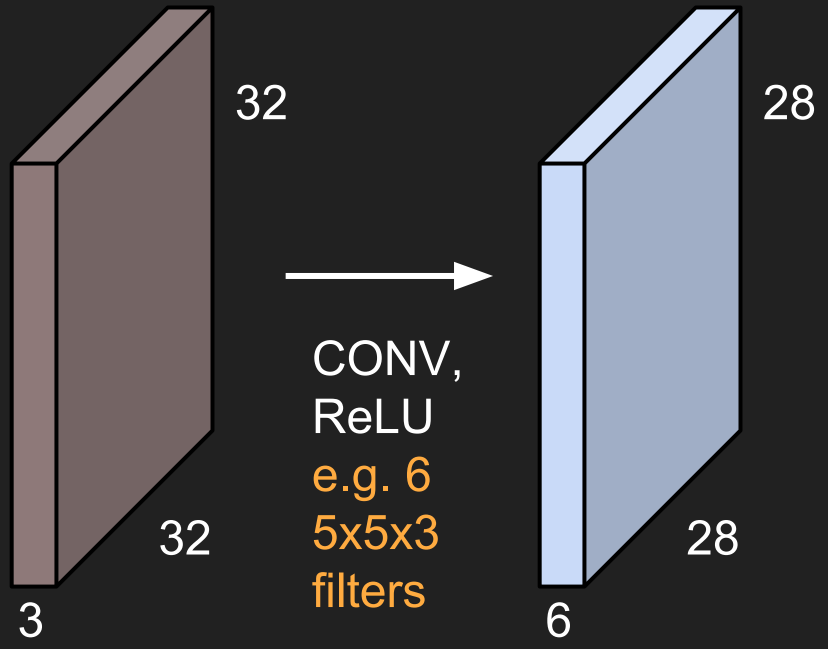
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



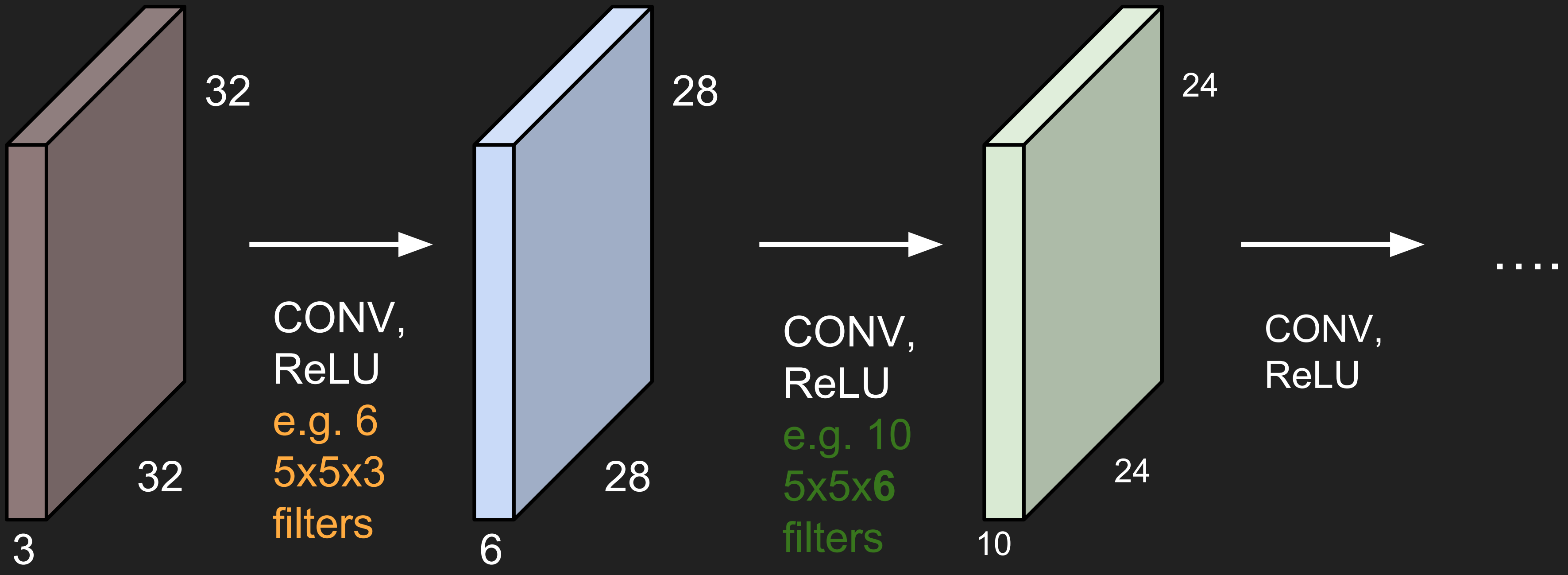
We stack these up to get a new "image" of size 28x28x6!

Slide credit: CS231n Lecture 7

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



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



Slide credit: CS231n Lecture 7

Two key insights

1) Features are **hierarchical**

   Composing high-complexity features out of low-complexity features is more efficient than learning high-complexity features directly.

   e.g.: having an “circle” detector is useful for detecting faces… and basketballs

2) Features are **translationally invariant**

   If a feature is useful to compute at (x, y) it is useful to compute that feature at (x’, y’) as well

Hierarchical organization

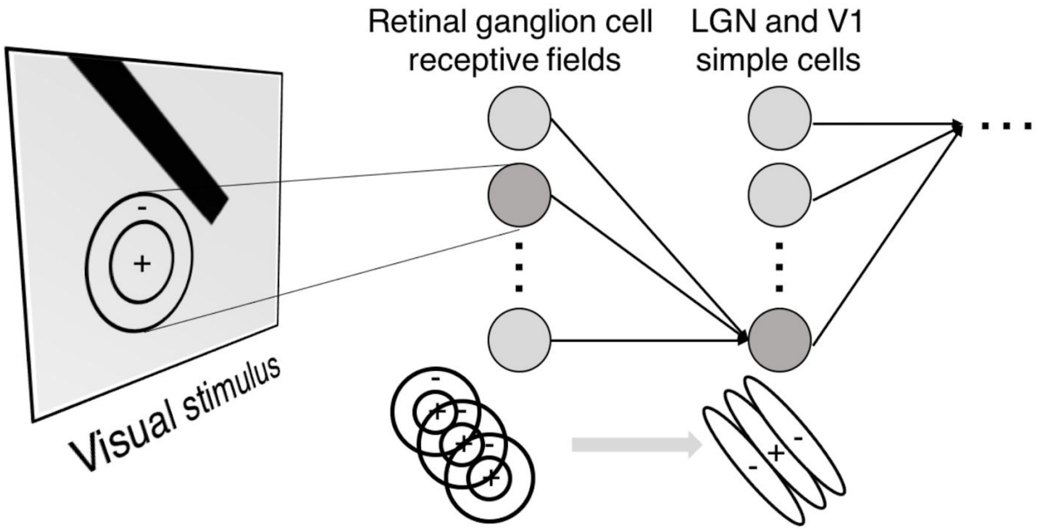


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

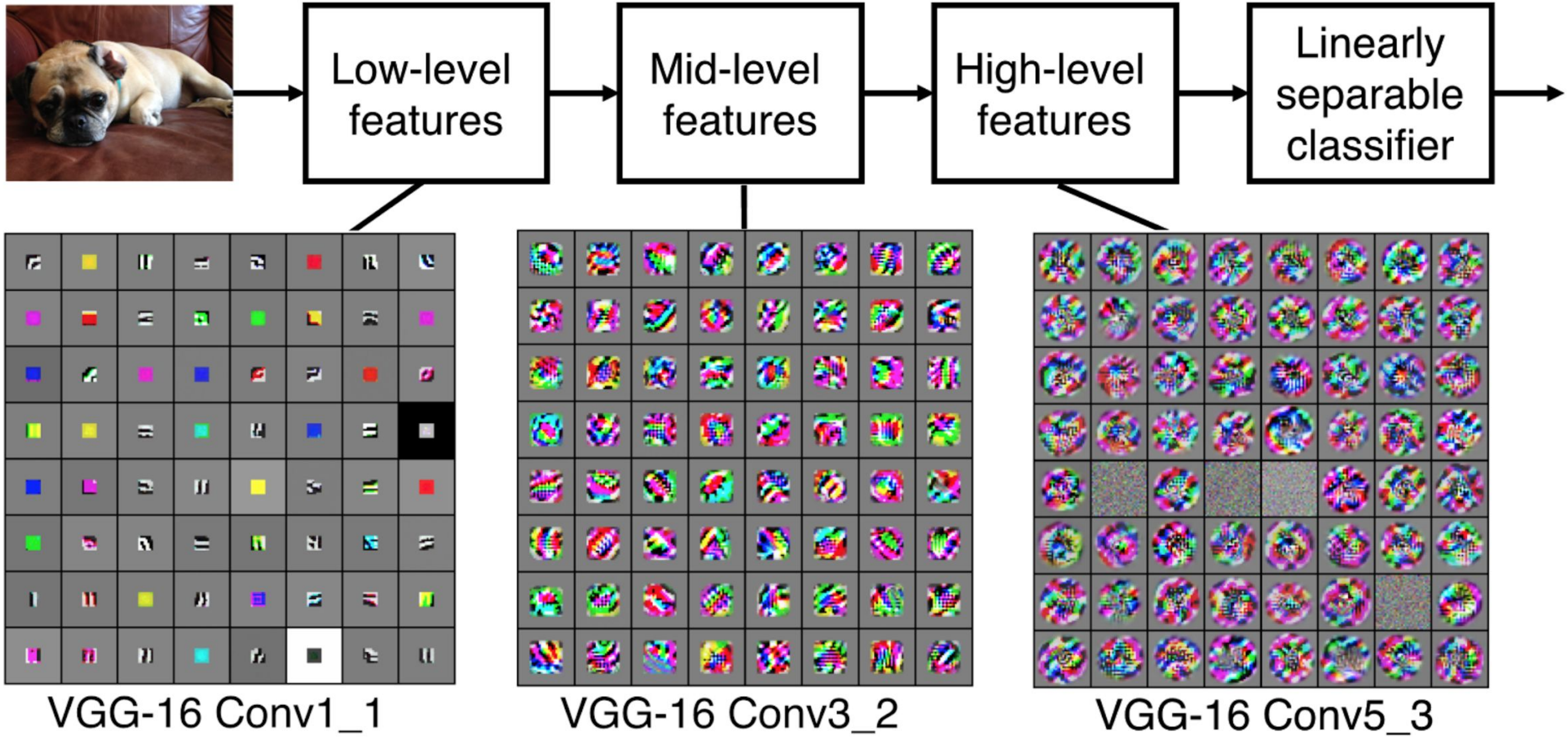
No response

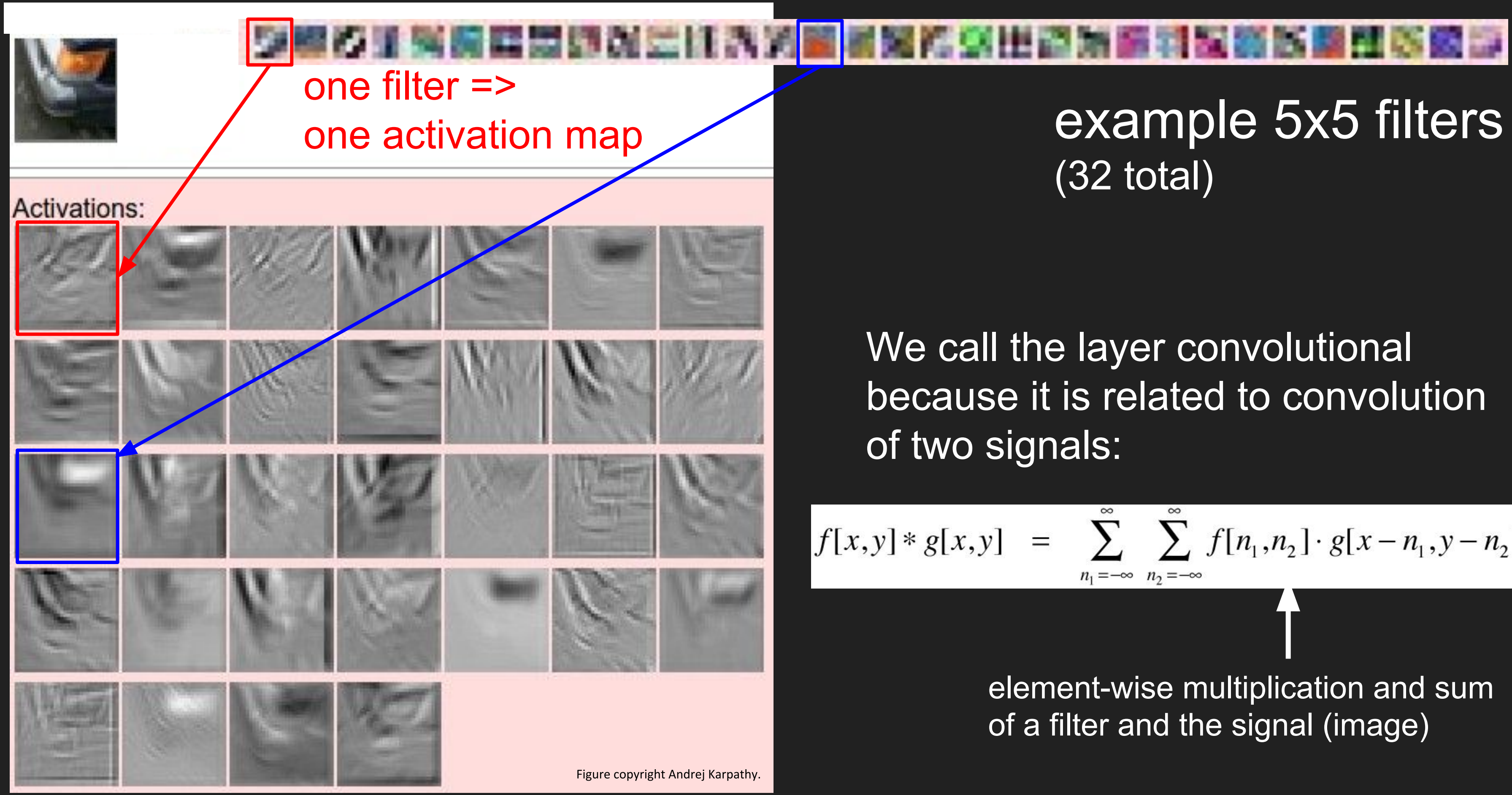
Response (end point)

Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].





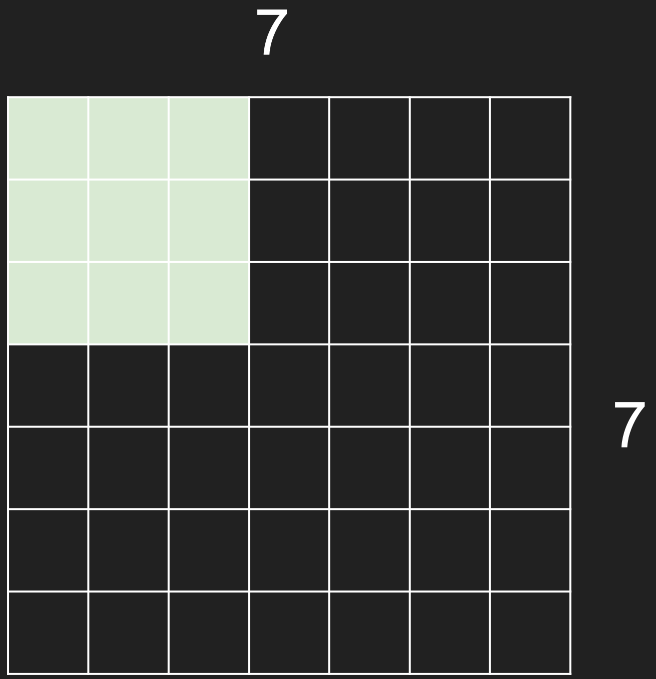
example 5x5 filters
(32 total)

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

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

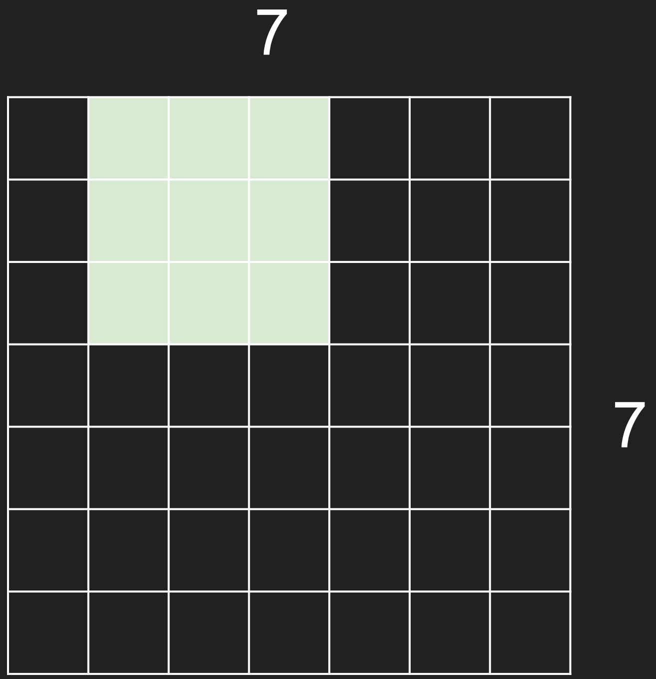
element-wise multiplication and sum of a filter and the signal (image)

A closer look at spatial dimensions:



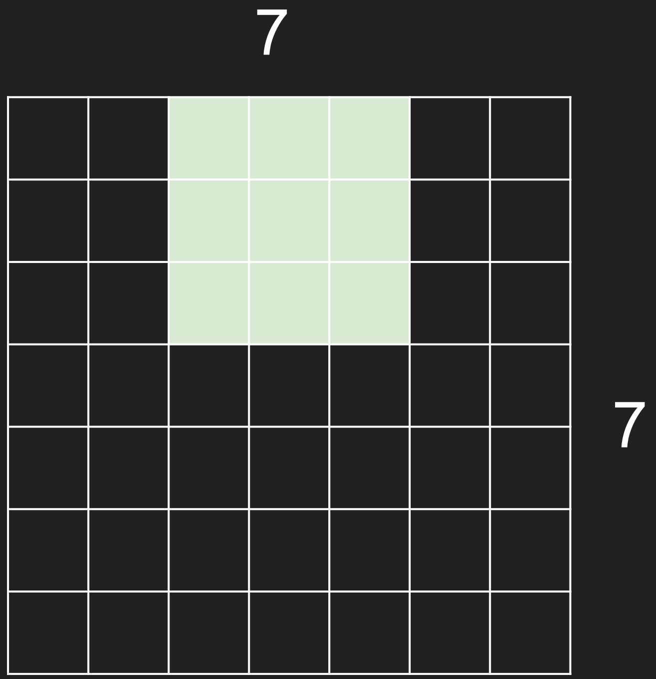
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



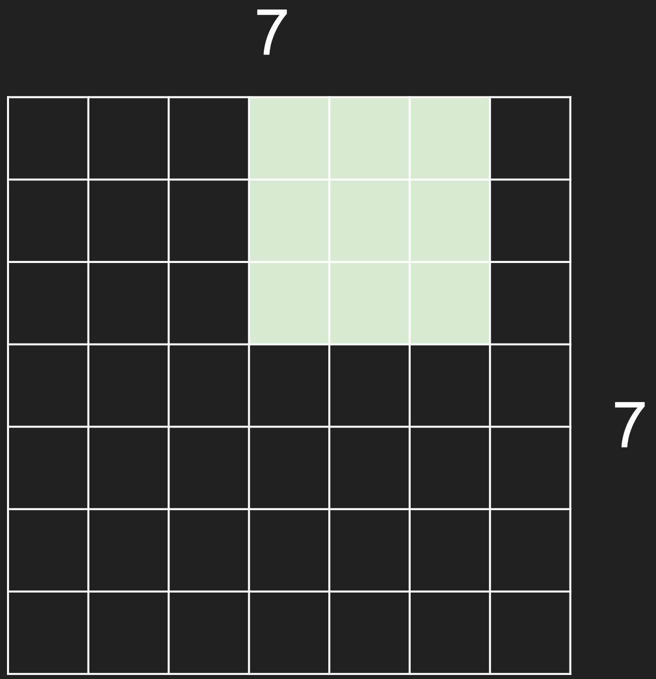
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



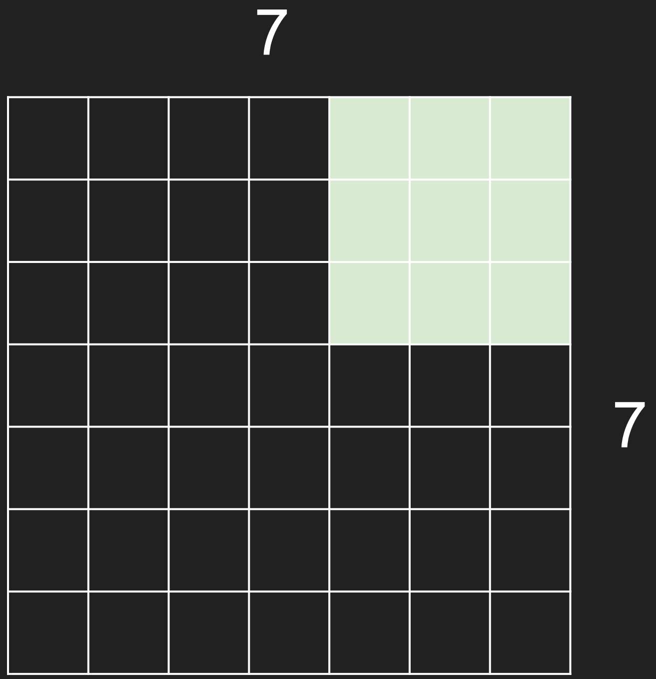
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

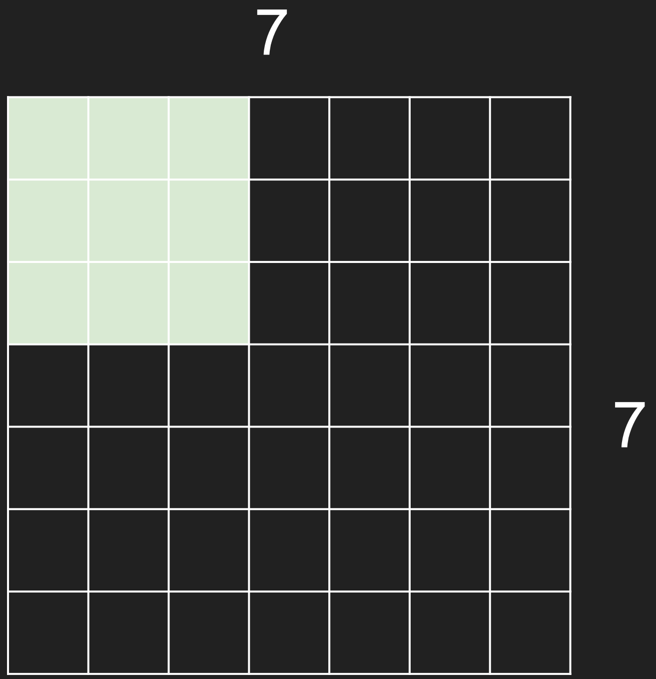
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

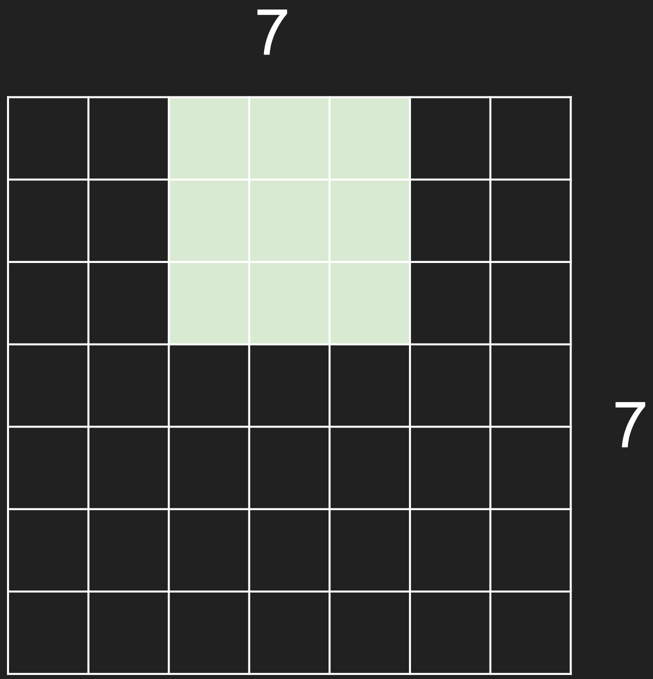
=> 5x5 output

A closer look at spatial dimensions:



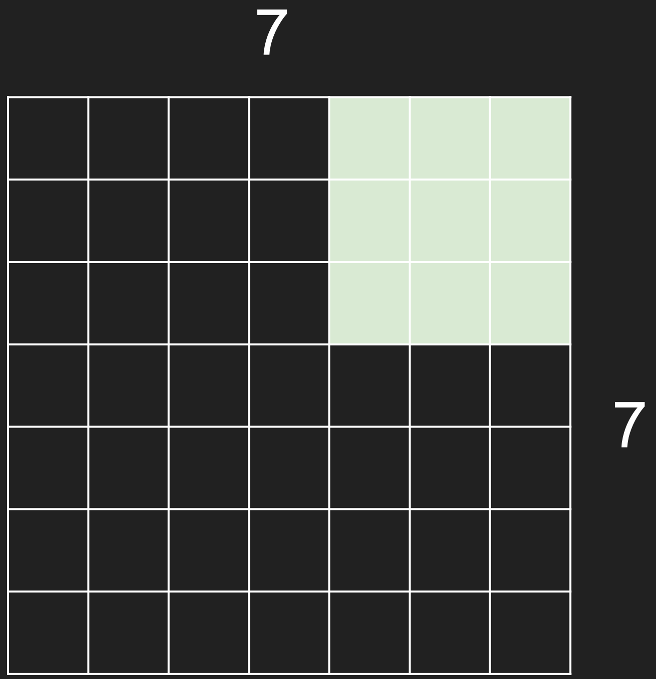
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



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

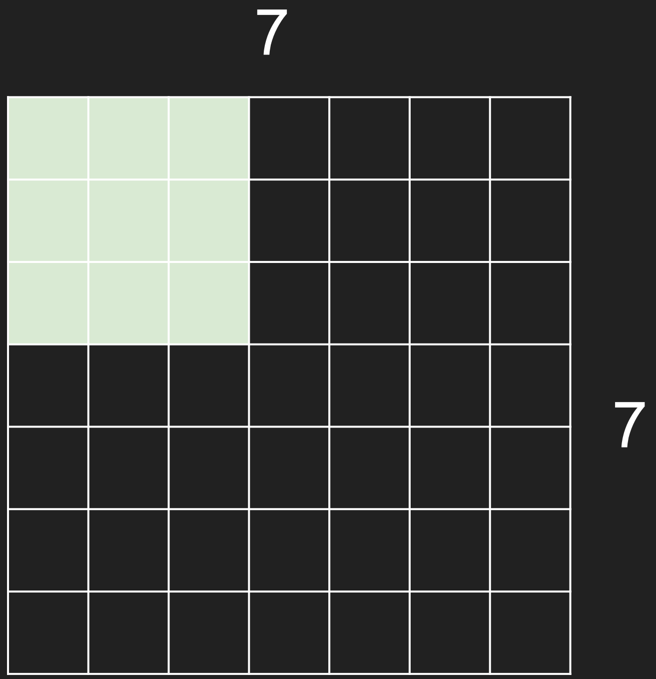
A closer look at spatial dimensions:



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

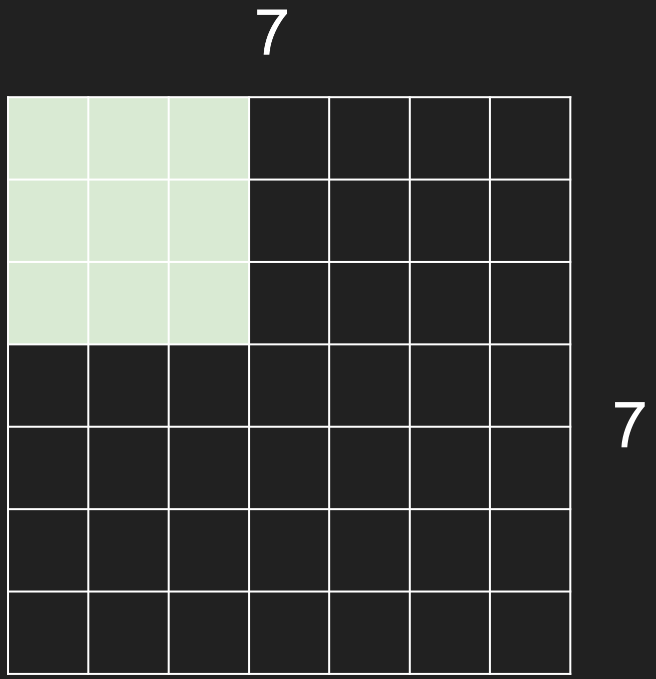
=> 3x3 output!

A closer look at spatial dimensions:



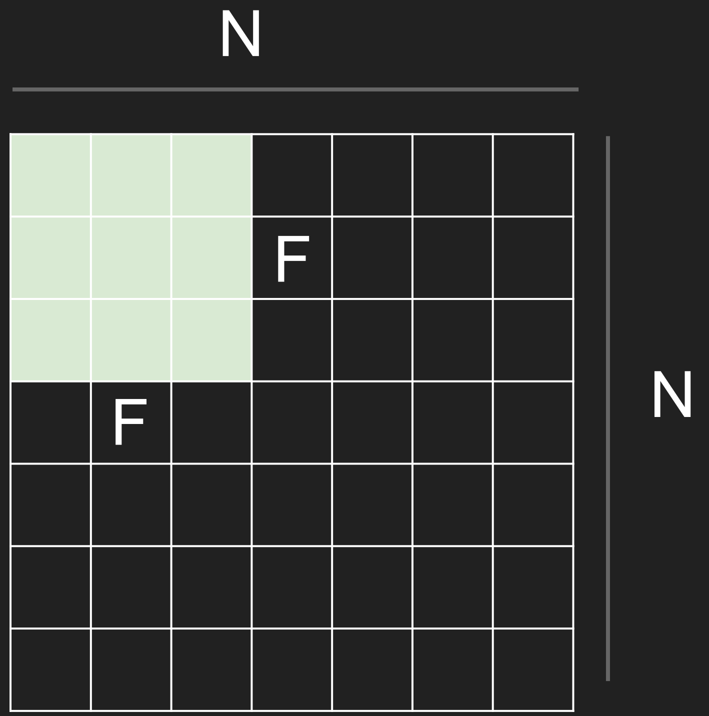
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

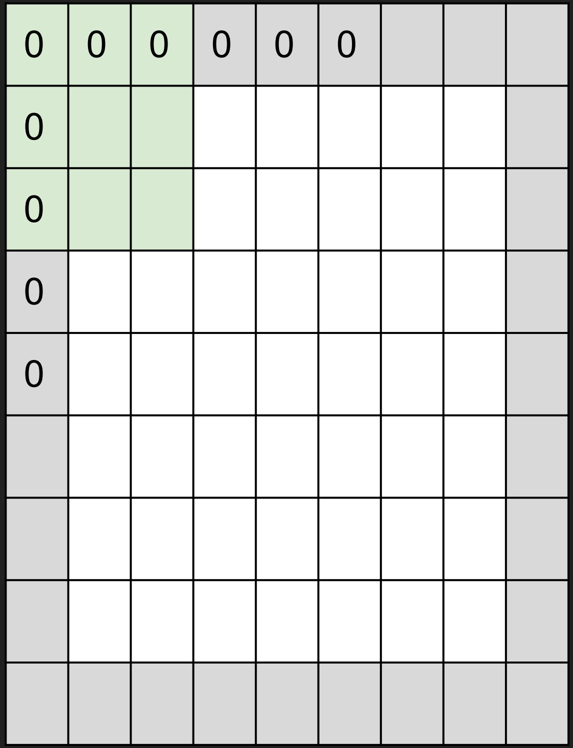
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

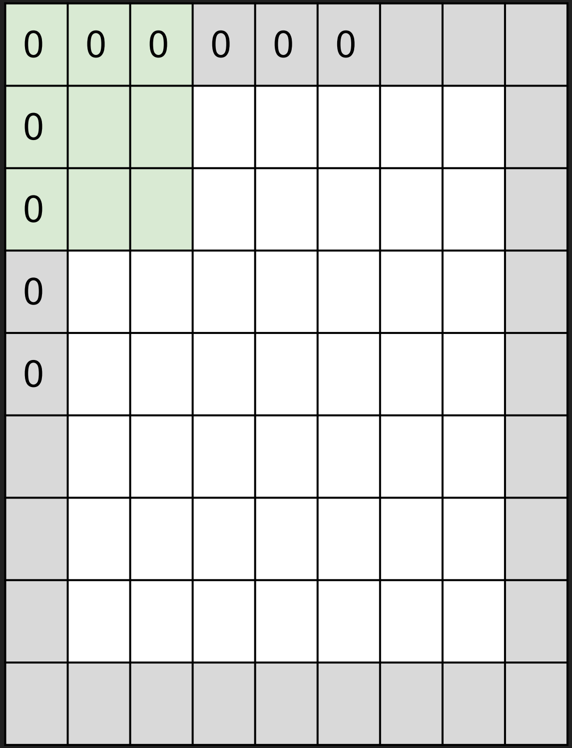
In practice: Common to zero pad the border



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

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

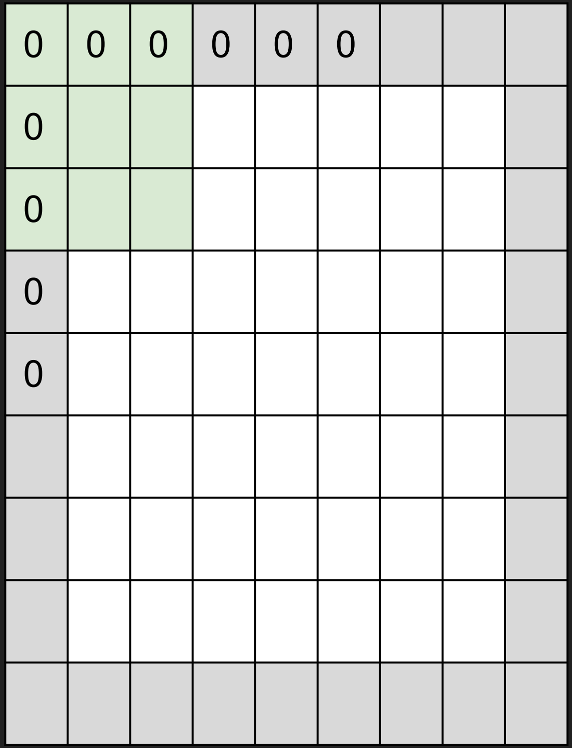
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

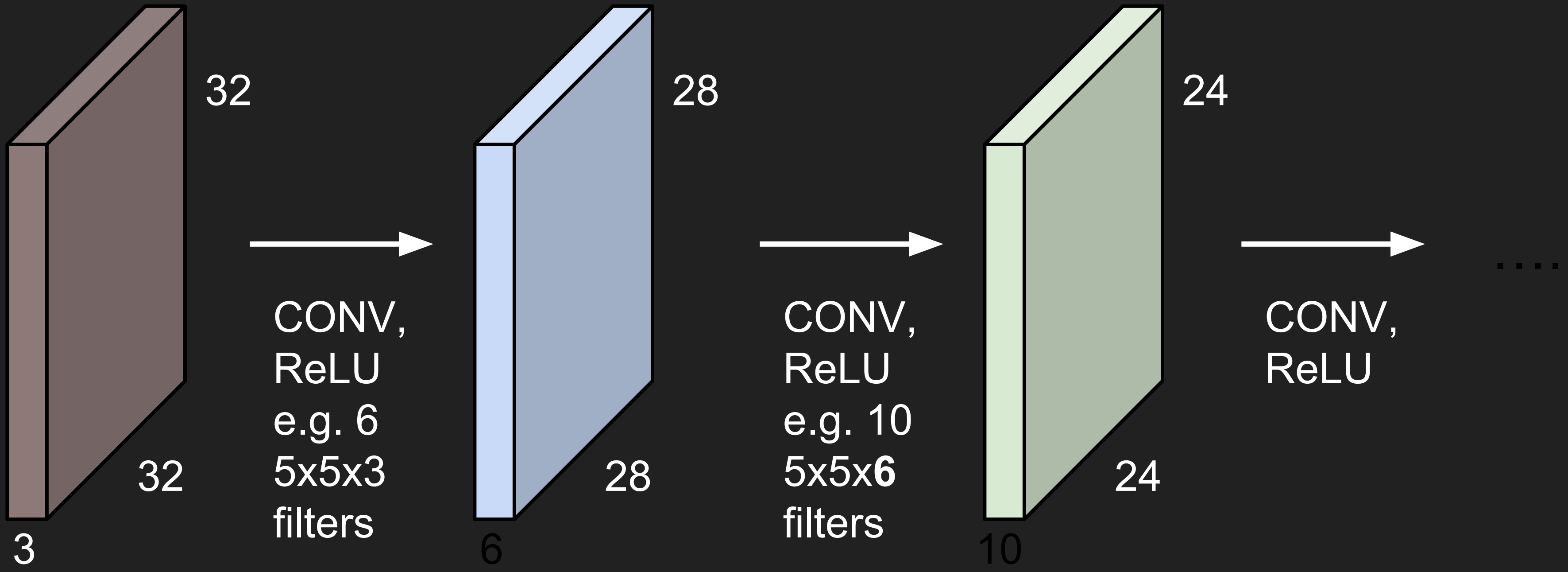


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.



Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

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

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

tf.layers.conv2d

```
conv2d(
    inputs,
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format='channels_last',
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    trainable=True,
    name=None,
    reuse=None
)
```

- **`inputs`**: Tensor input.
- **`filters`**: Integer, the dimensionality of the output space (i.e. the number of filters in the convolution).
- **`kernel_size`**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **`strides`**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **`padding`**: One of `"valid"` or `"same"` (case-insensitive).

Defined in `tensorflow/python/layers/convolutional.py`.

Functional interface for the 2D convolution layer.

This layer creates a convolution kernel that is convolved (actually cross-correlated) with the layer input to produce a tensor of outputs. If `use_bias` is True (and a `bias_initializer` is provided), a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

TensorFlow Padding Options

"VALID" = without padding:

```
inputs:         1  2  3  4  5  6  7  8  9  10 11 (12 13)
                |________________|                dropped
                               |_________________|
```

"SAME" = with zero padding:

```
            pad|                                      |pad
inputs:      0 |1  2  3  4  5  6  7  8  9  10 11 12 13|0  0
            |________________|
                           |_________________|
                                          |________________|
```

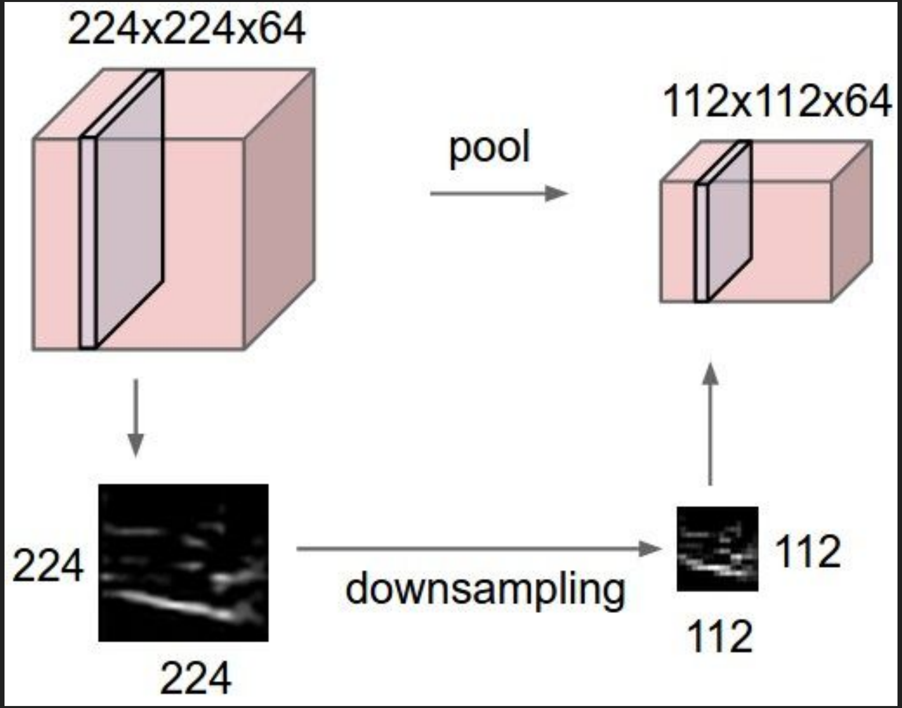
Input width = 13

Filter width = 6

Stride = 5

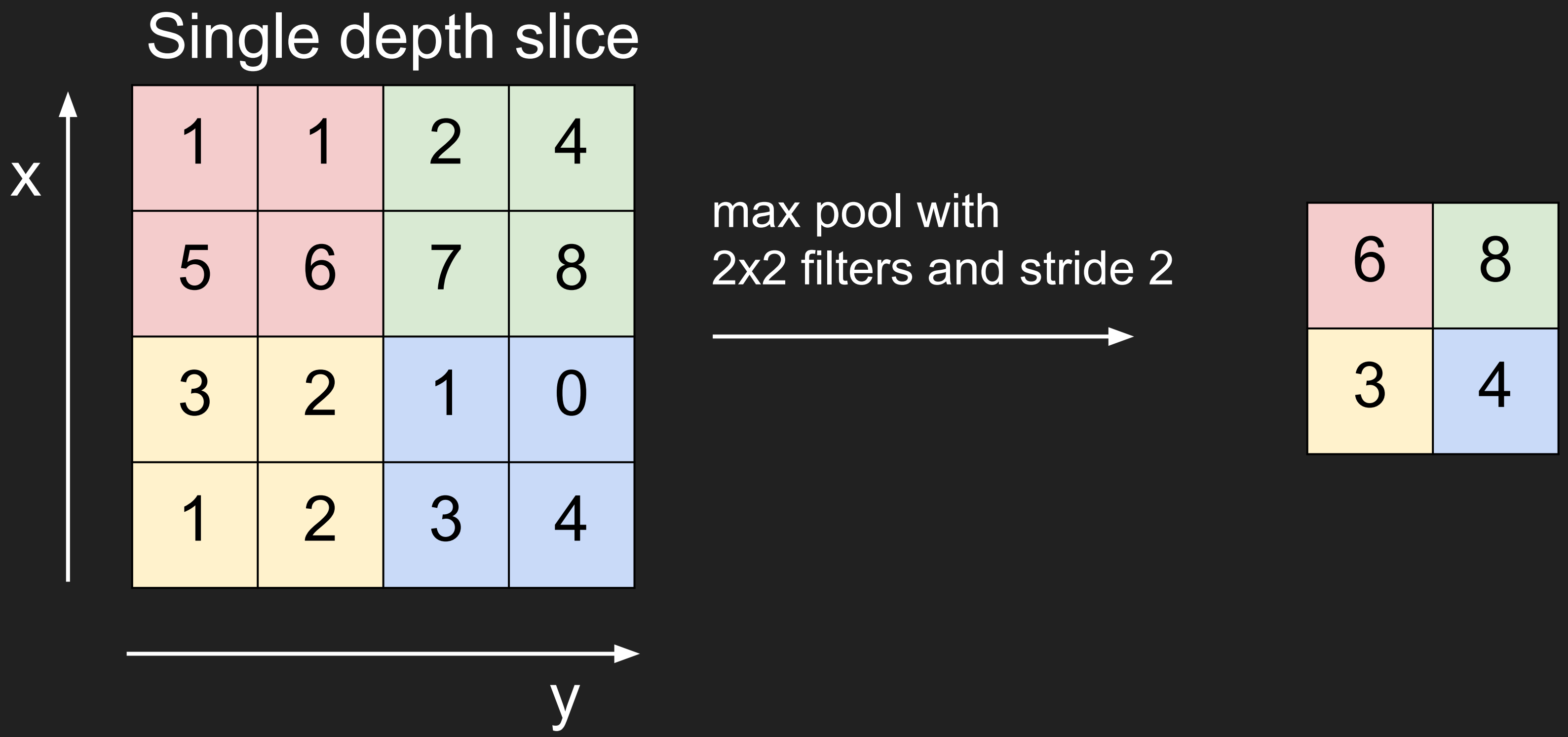
Pooling Layer

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



Slide credit: CS231n Lecture 7

Max Pooling



Slide credit: CS231n Lecture 7

Max Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

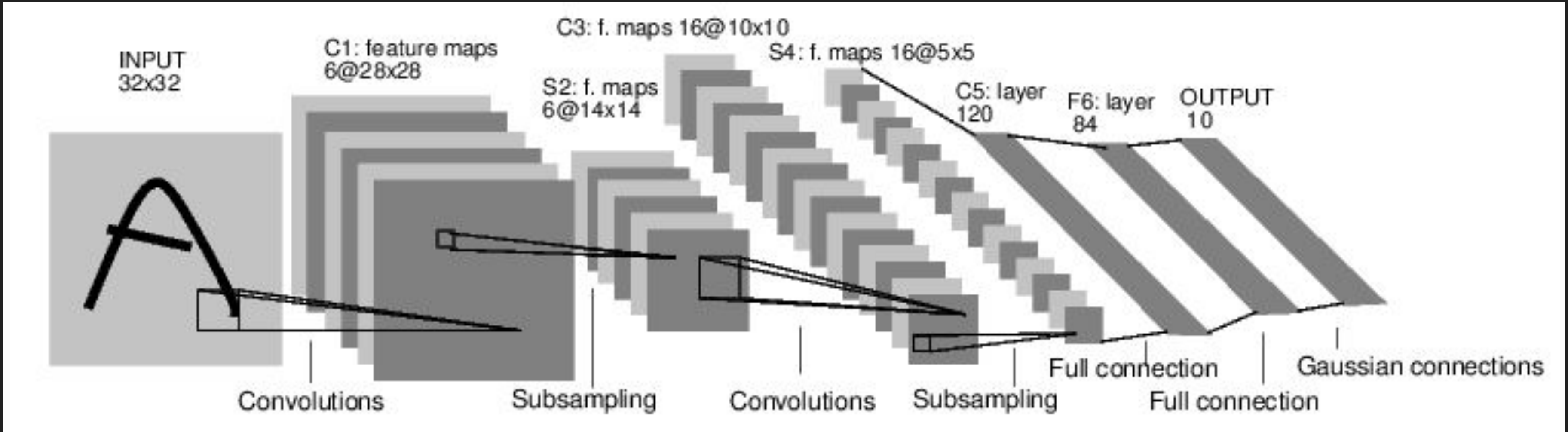
Common settings:

F = 2, S = 2

F = 3, S = 2

Slide credit: CS231n Lecture 7

Case study: LeNet-5 [LeCun et al., 1998]



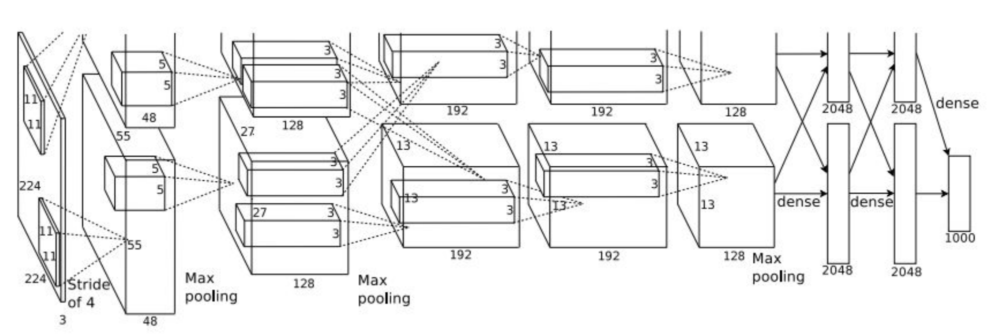
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Slide credit: CS231n Lecture 7

Case study: AlexNet [Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture

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



Slide credit: CS231n Lecture 7

Case study: VGGNet [Simonyan and Zisserman, 2014]

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

best model

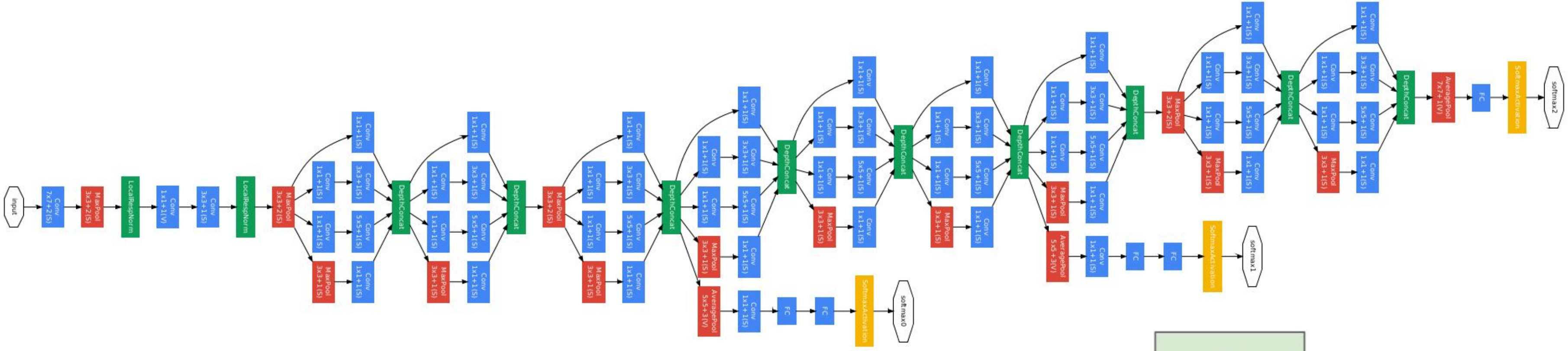
11.2% top 5 error in ILSVRC 2013
-> 7.3% top 5 error

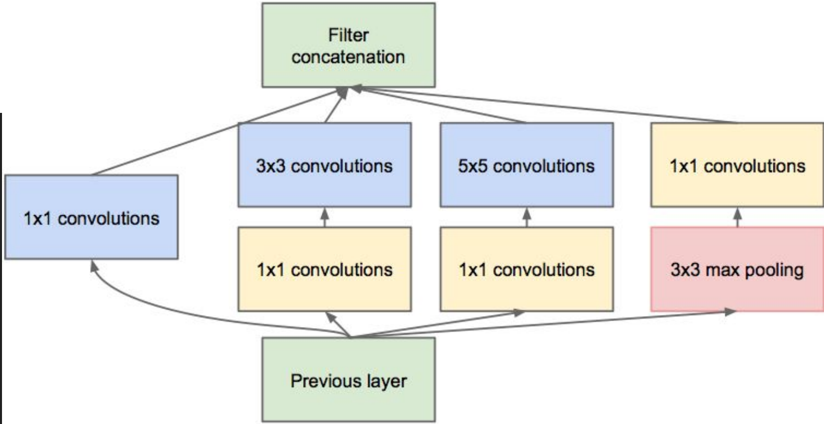
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

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

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

Case study: GoogLeNet [Szegedy et al., 2014]



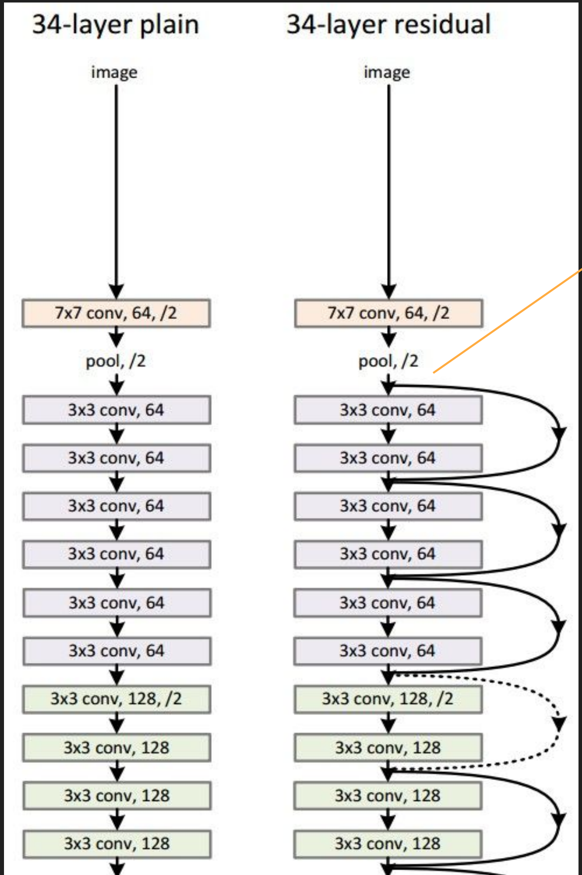


Inception module

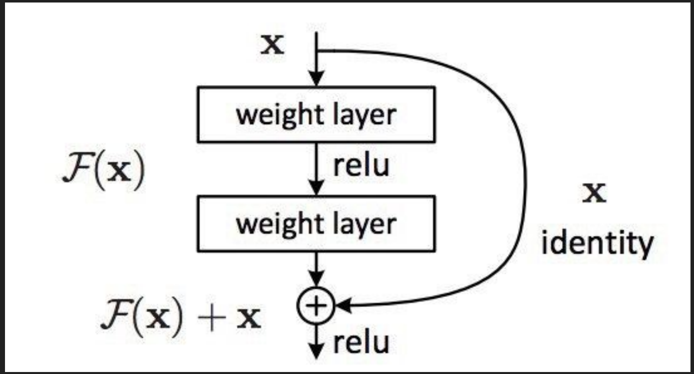
ILSVRC 2014 winner (6.7% top 5 error)

Slide credit: CS231n Lecture 7

Case study: ResNet [He et al., 2015]

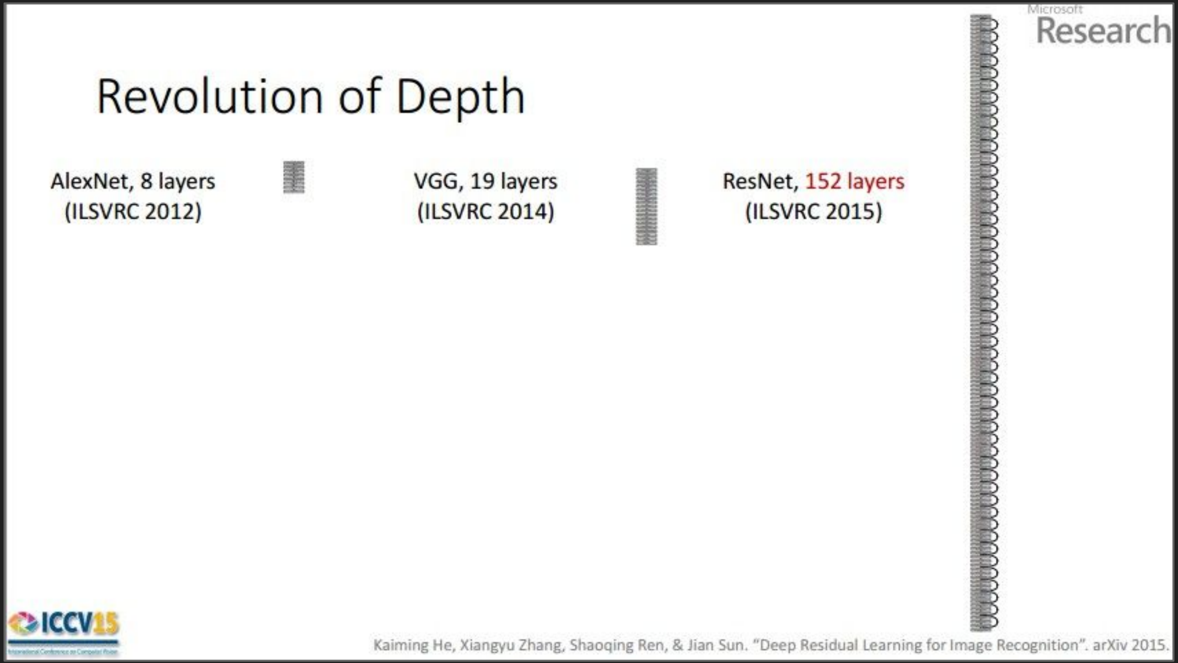


spatial dimension only
56x56!



Slide credit: CS231n Lecture 7

Case study: ResNet [He et al., 2015]



(slide from Kaiming He's ICCV 2015 presentation)

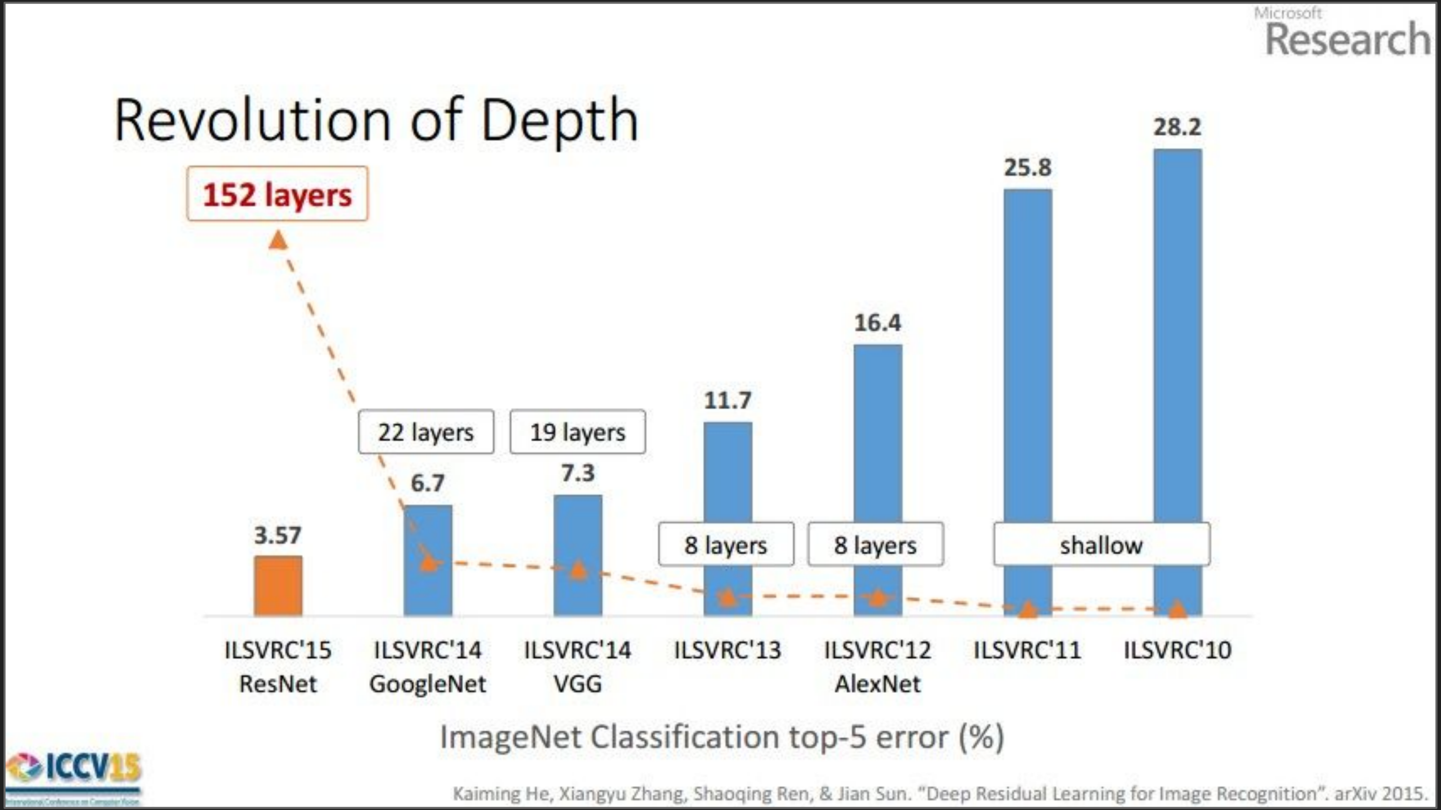
2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet!
(even though it has 8x more layers)

ILSVRC 2015 winner
(3.6% top 5 error)

Slide credit: CS231n Lecture 7

Case study: ResNet [He et al., 2015]



(slide from Kaiming He's ICCV 2015 presentation)

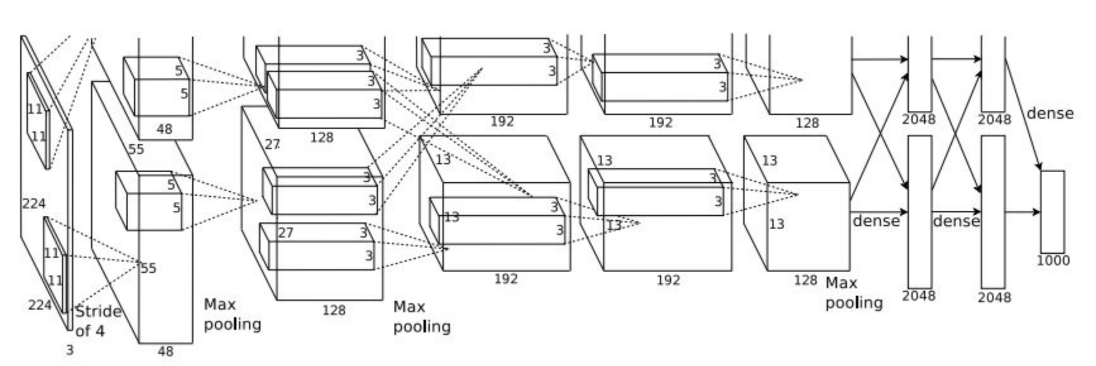
Slide credit: CS231n Lecture 7

Visualizing ConvNet Features

What’s going on inside ConvNets?

This image is CC0 public domain

Input Image:
3 x 224 x 224

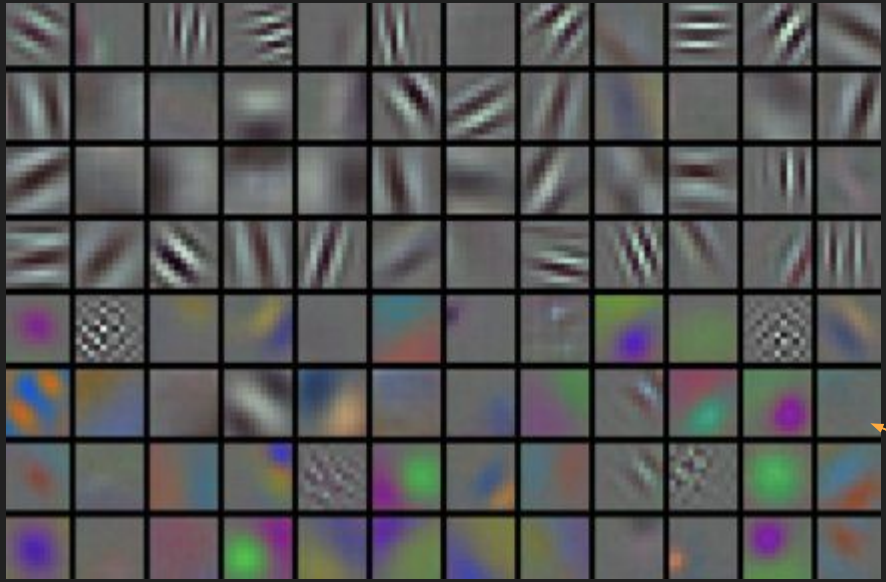


Class Scores:
1000 numbers

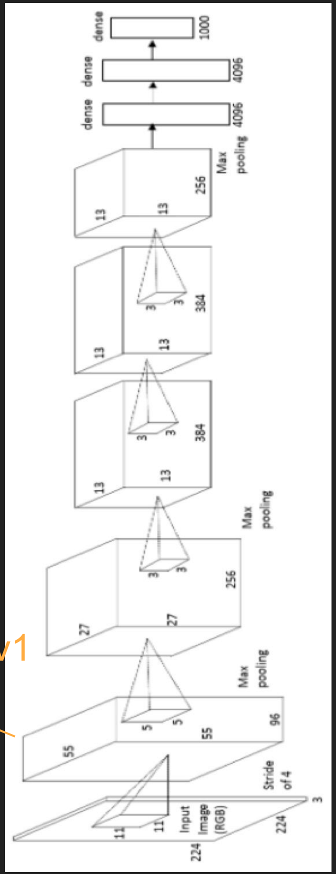
What are the intermediate features looking for?

Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figure reproduced with permission.

Visualizing CNN features: Look at filters

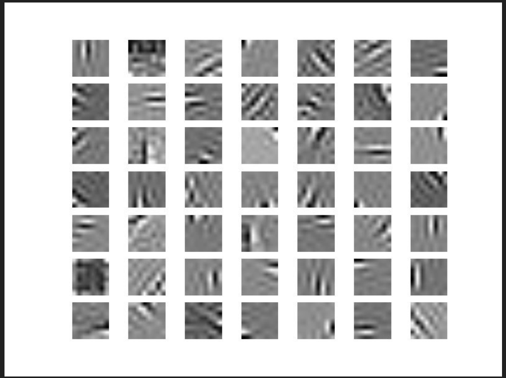
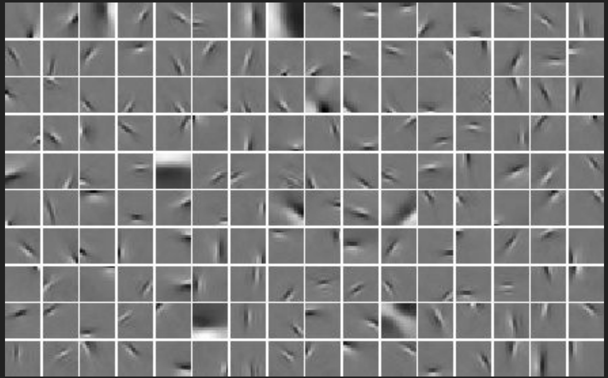
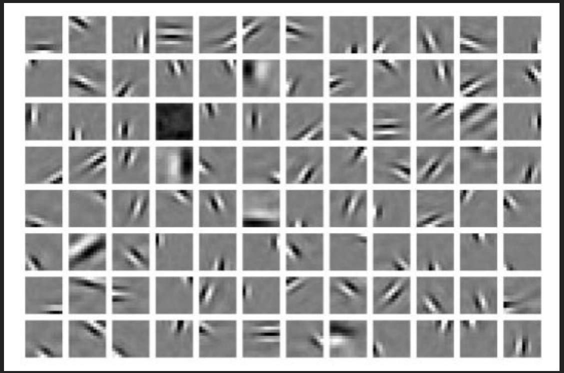
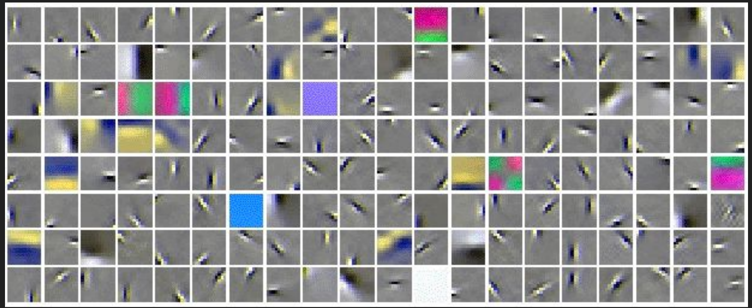
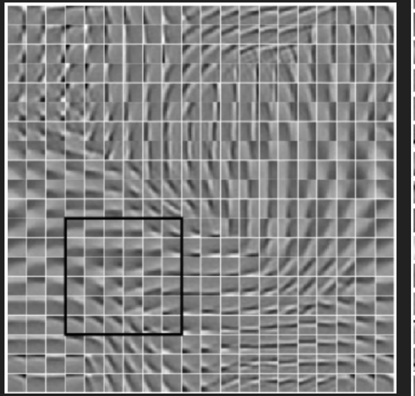
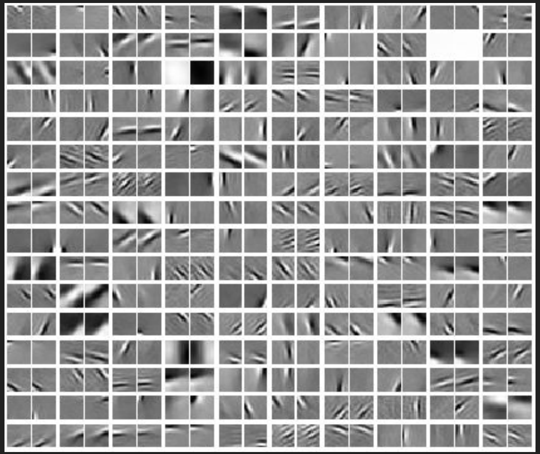


conv1



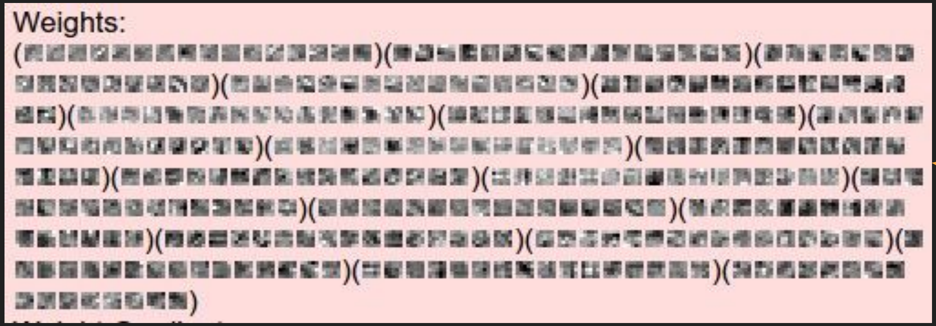
Slide credit: CS231n Lecture 9

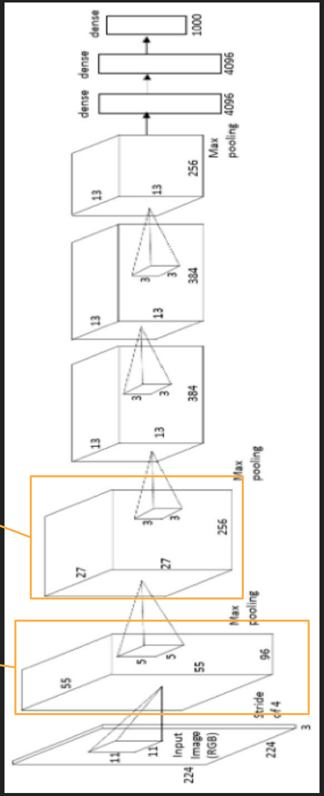
First layers: networks learn similar features

Slide credit: CS231n Lecture 9

Visualizing CNN features: Look at filters



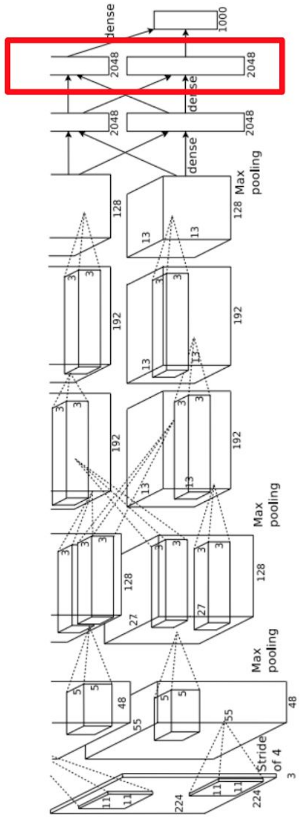


Filters from higher layers don't make much sense

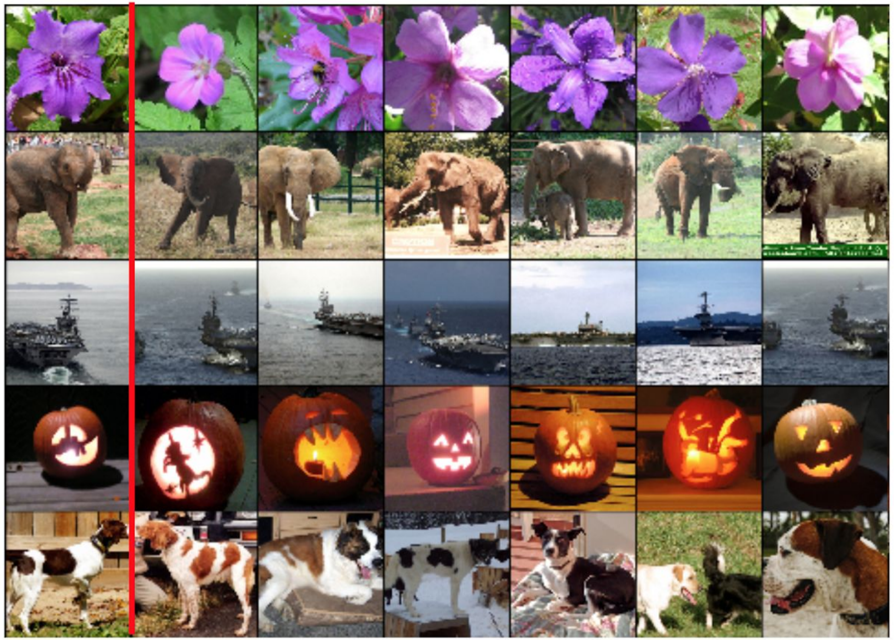
Slide credit: CS231n Lecture 9
Filters from ConvNetJS CIFAR-10 model

Last Layer: Nearest Neighbors

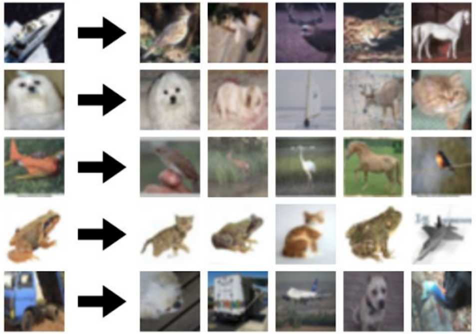
4096-dim vector



Test image L2 Nearest neighbors in feature space



Recall: Nearest neighbors in pixel space



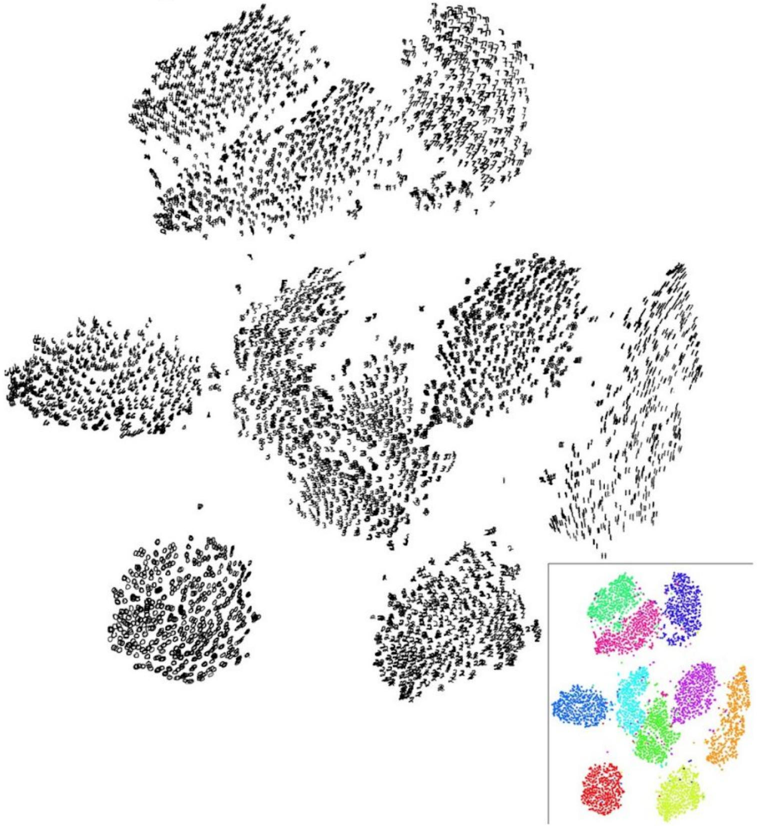
Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figures reproduced with permission.

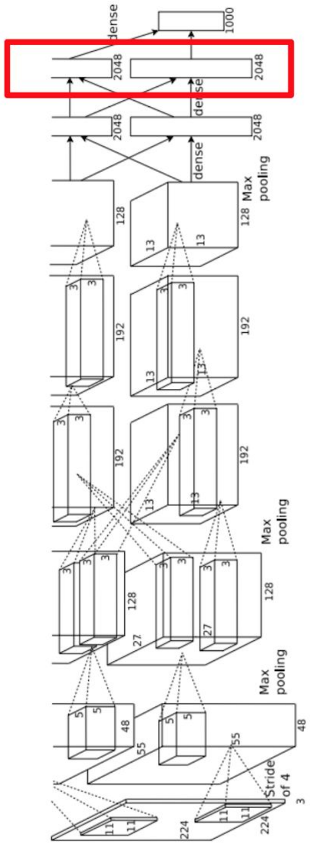
Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

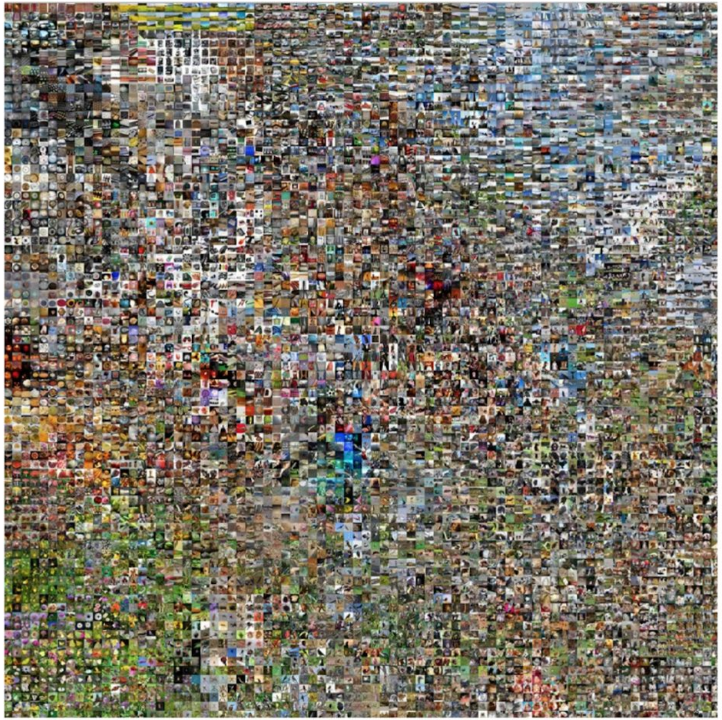
More complex: **t-SNE**

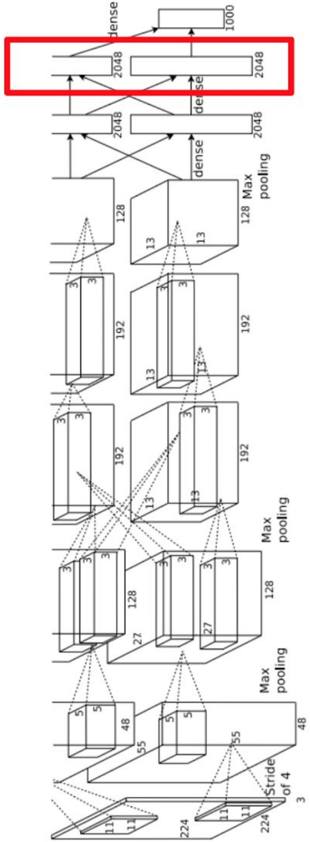




Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008

Last Layer: Dimensionality Reduction





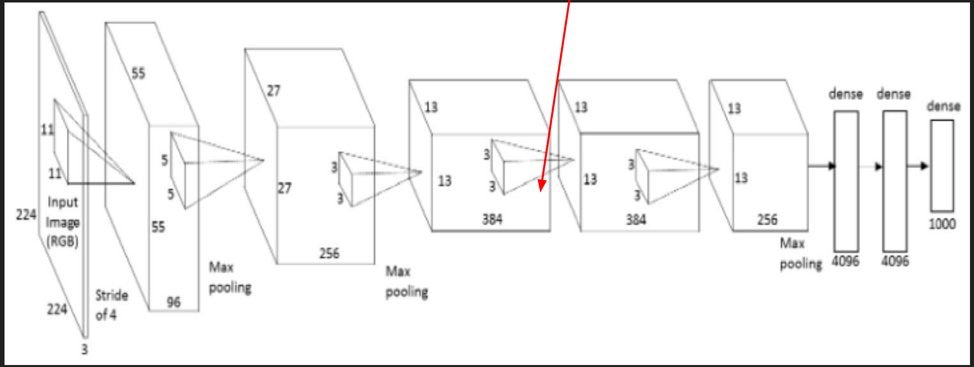
Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figure reproduced with permission.

See high-resolution versions at
http://cs.stanford.edu/people/karpathy/cnnembed/

Visualizing CNN features: (guided) backprop

Choose an image

Choose a layer and a neuron in a CNN

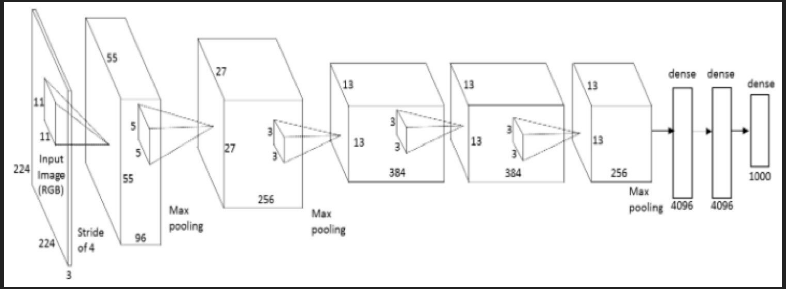


Question:
How does the chosen neuron respond to the image?

Slide credit: CS231n Lecture 9

Visualizing CNN features: (guided) backprop

1. Feed image into net



2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron

3. Backprop to image:

Guided backpropagation: instead

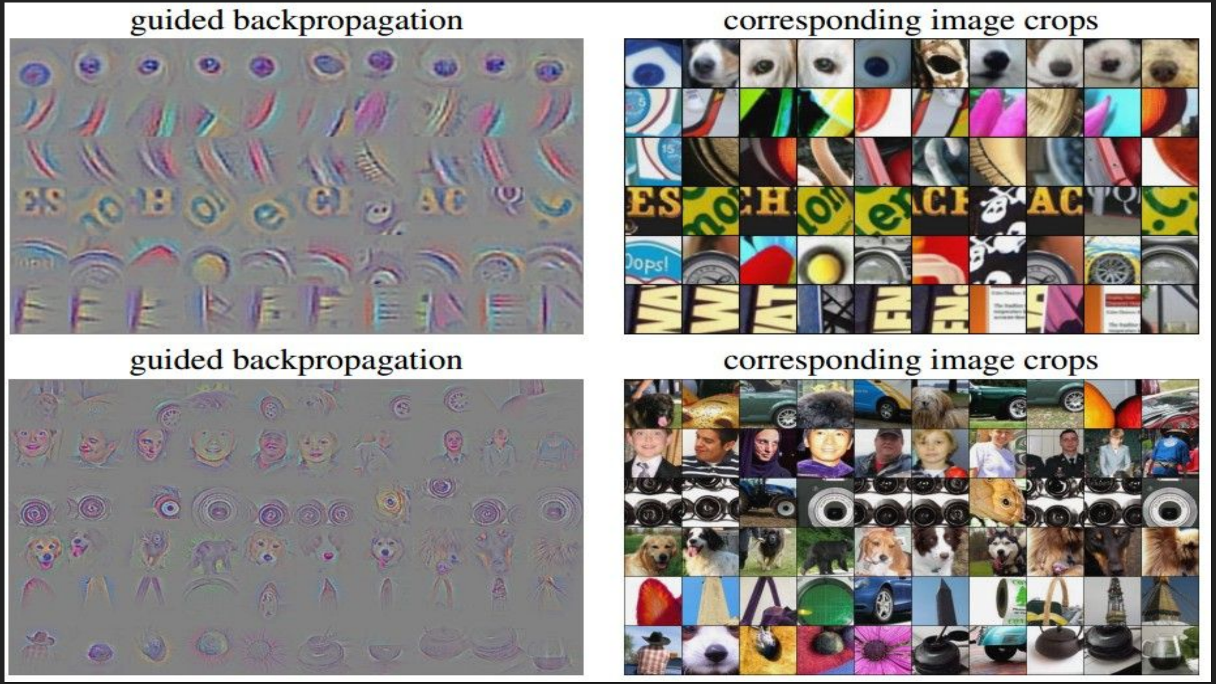
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014.
Dosovitskiy et al., “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015
Slide credit: CS231n Lecture 9

Visualizing CNN features: (guided) backprop

Visualization of patterns learned by the layer **conv6** (top) and layer **conv9** (bottom) of the network trained on ImageNet.

Each row corresponds to one filter.

The visualization using "guided backpropagation" is based on the top 10 image patches activating this filter taken from the ImageNet dataset.



Dosovitskiy et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Slide credit: CS231n Lecture 9

Visualizing CNN features: Gradient ascent

(Guided) backprop:
Find the part of an image that a neuron responds to

Gradient ascent:
Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I \underbrace{\boxed{f(I)}}_{\text{Neuron value}} + \underbrace{\boxed{R(I)}}_{\text{Natural image regularizer}}$$

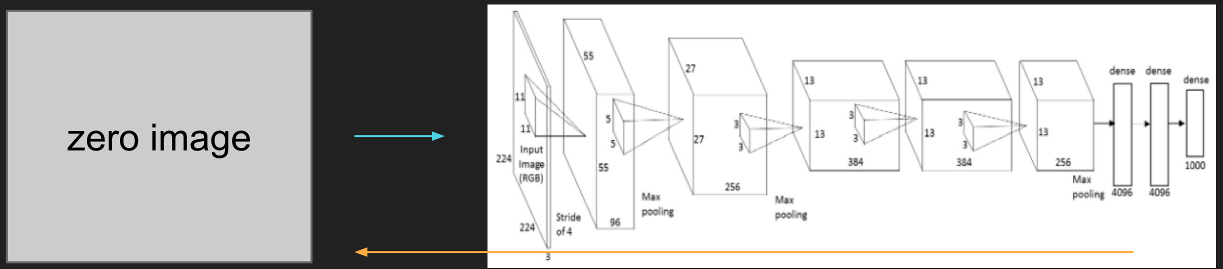
Slide credit: CS231n Lecture 9

Visualizing CNN features: Gradient ascent

$$\arg\max_{I} \boxed{S_c(I)} - \lambda \|I\|_2^2$$

score for class c (before Softmax)

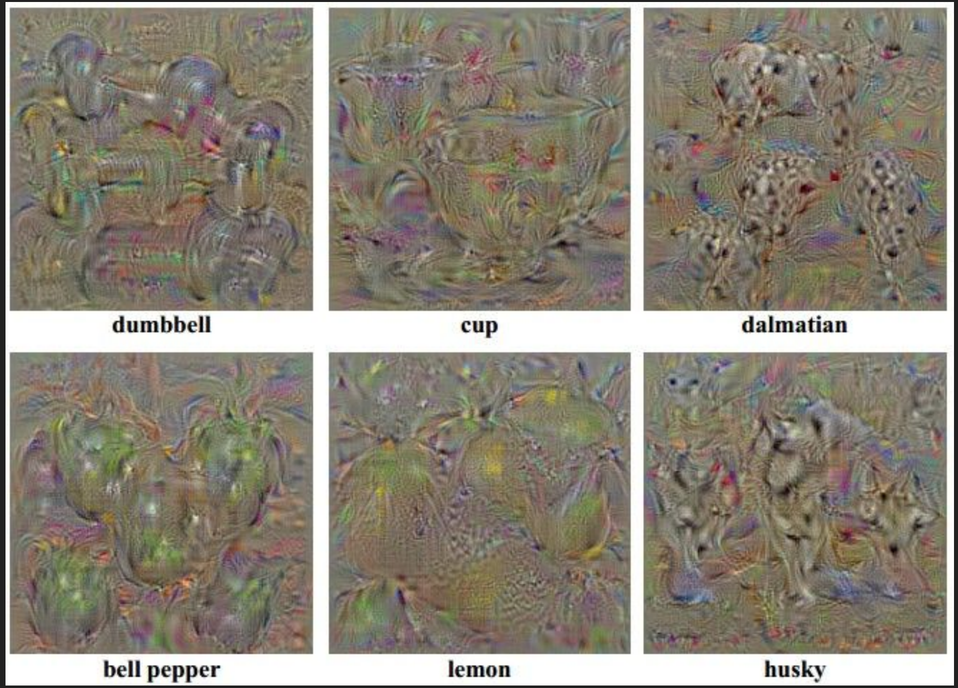
1. Initialize image to zeros

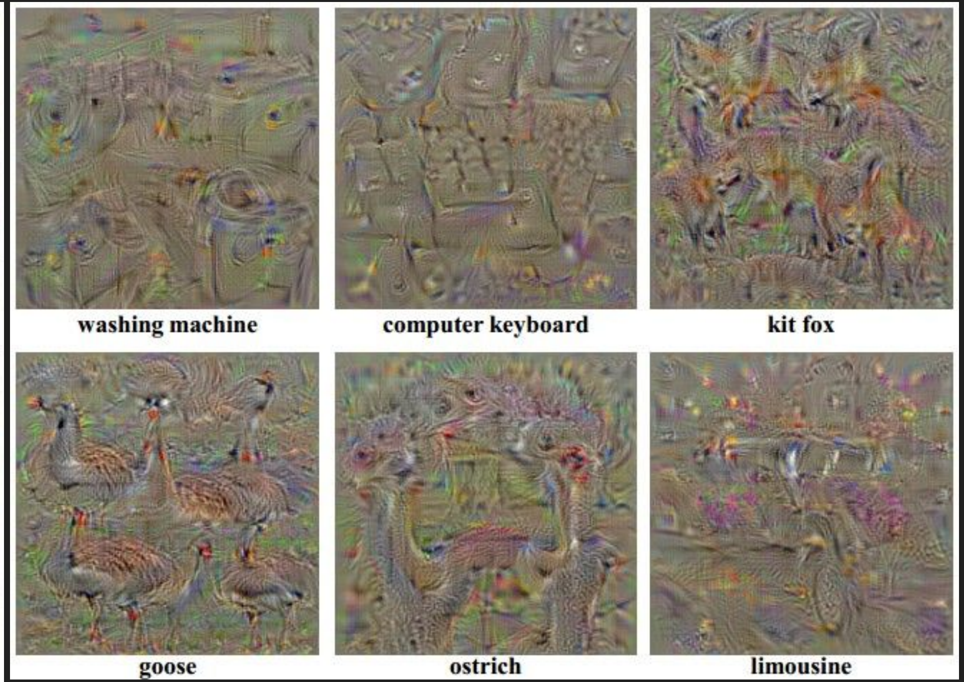


Repeat:
2. Forward image to compute current scores
3. Set gradient of scores to be 1 for target class, 0 for others
4. Backprop to get gradient on image
5. Make a small update to the image

Simonyan et al, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014

Visualizing CNN features: Gradient ascent

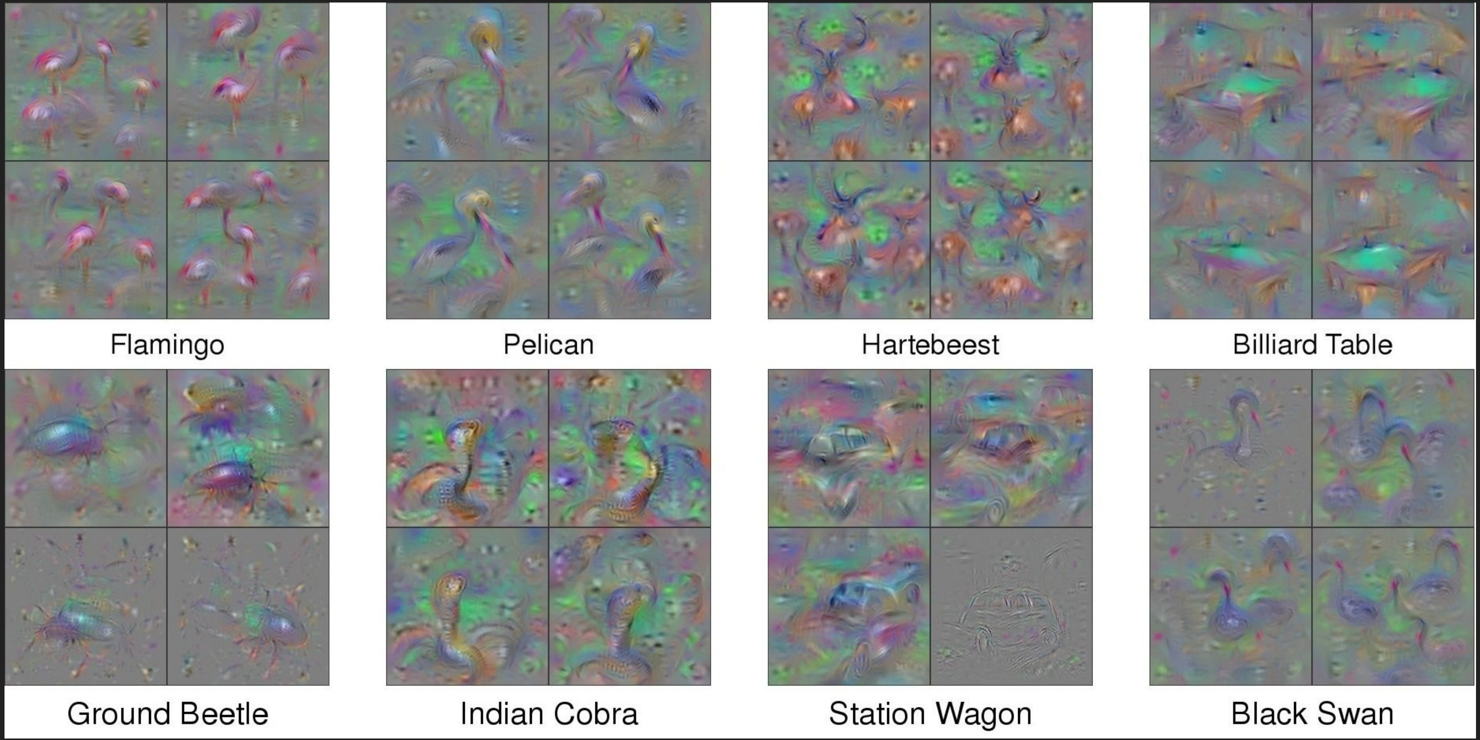




Simonyan et al, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014

Visualizing CNN features: Gradient ascent

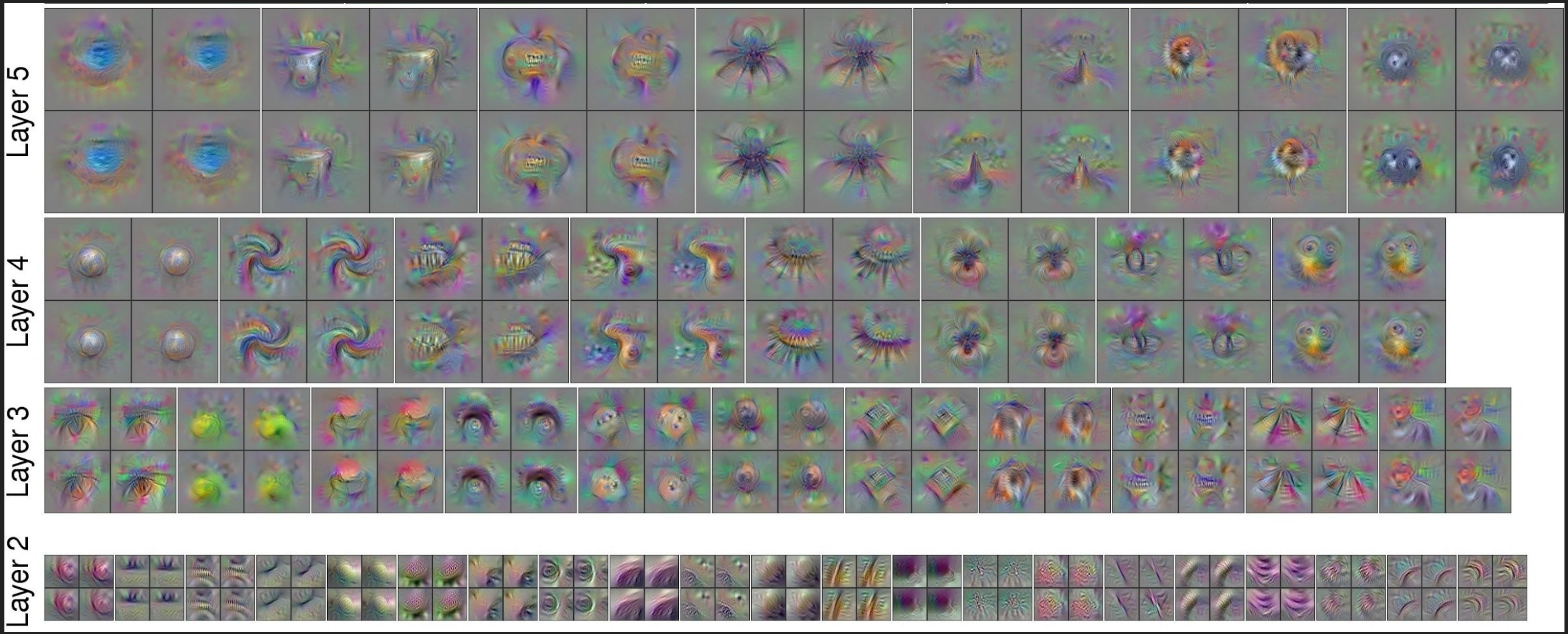
Better image regularizers give prettier results:



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2015

Visualizing CNN features: Gradient ascent

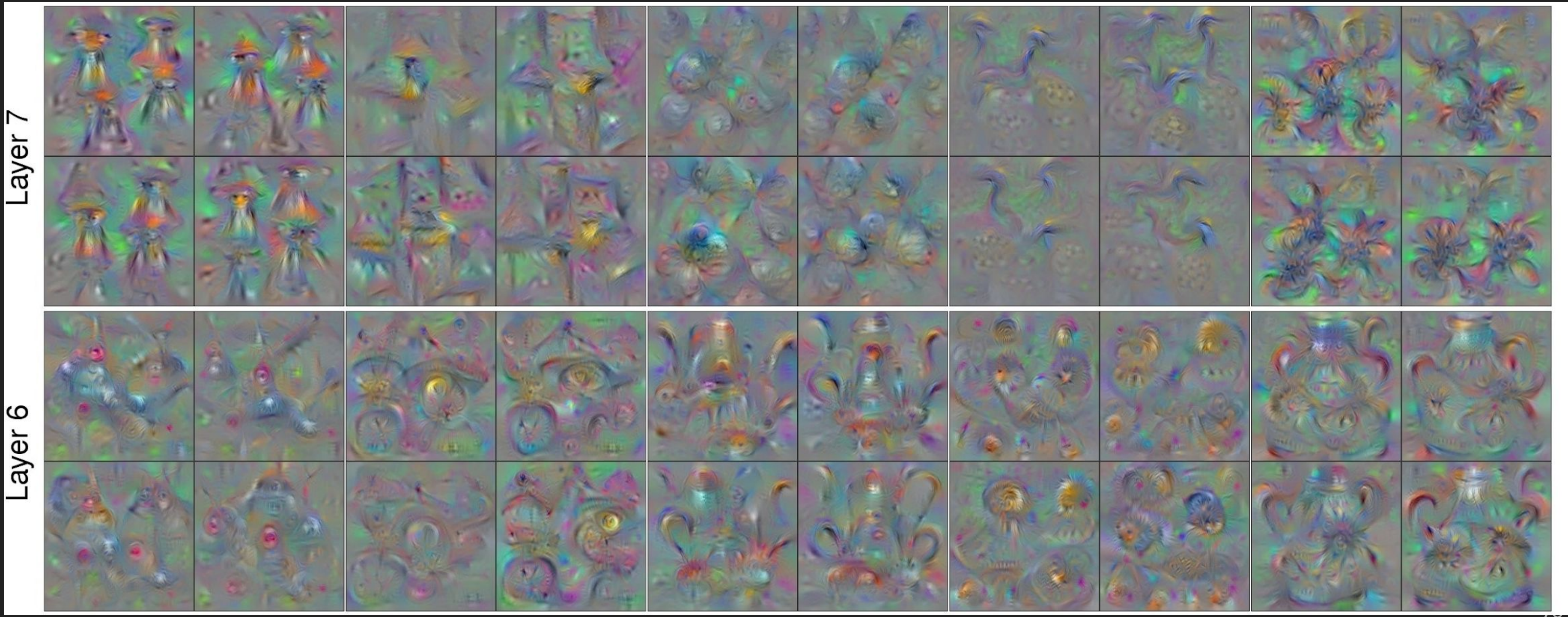
Use the same approach to visualize intermediate features



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2015

Visualizing CNN features: Gradient ascent

Use the same approach to visualize intermediate features



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2015

Visualizing CNN features: Gradient ascent

You can add even more tricks to get nicer results

Nguyen et al, "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks", ICML Visualization for Deep Learning Workshop 2016

Take-aways

Convolutional networks are tailor-made for computer vision tasks.

They exploit:

- Hierarchical nature of features
- Translation invariance of features

“Understanding” what a convnet learns is non-trivial, but some clever approaches exist.

Next class

ConvNet in TensorFlow

Feedback: huyenn@stanford.edu

Thanks!