

Understanding CNNs

A bit of history...

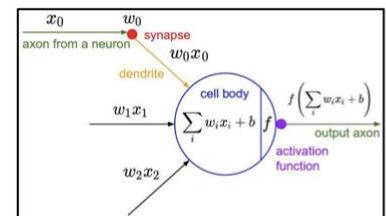
The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

update rule:

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$



Frank Rosenblatt, ~1957: Perceptron



[This image](#) by Rocky Acosta is licensed under CC-BY 3.0

A bit of history:

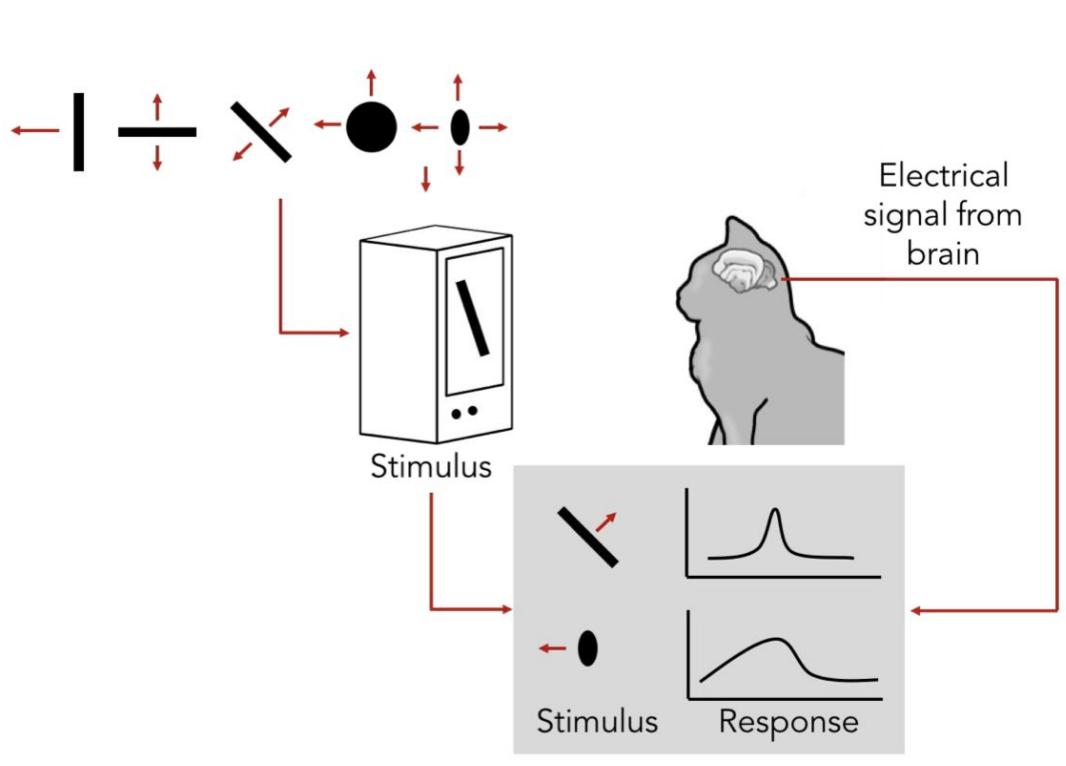
**Hubel & Wiesel,
1959**

RECEPTIVE FIELDS OF SINGLE
NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR
INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...



[Cat image](#) by CNX OpenStax is licensed under CC BY 4.0; changes made

Hierarchical organization

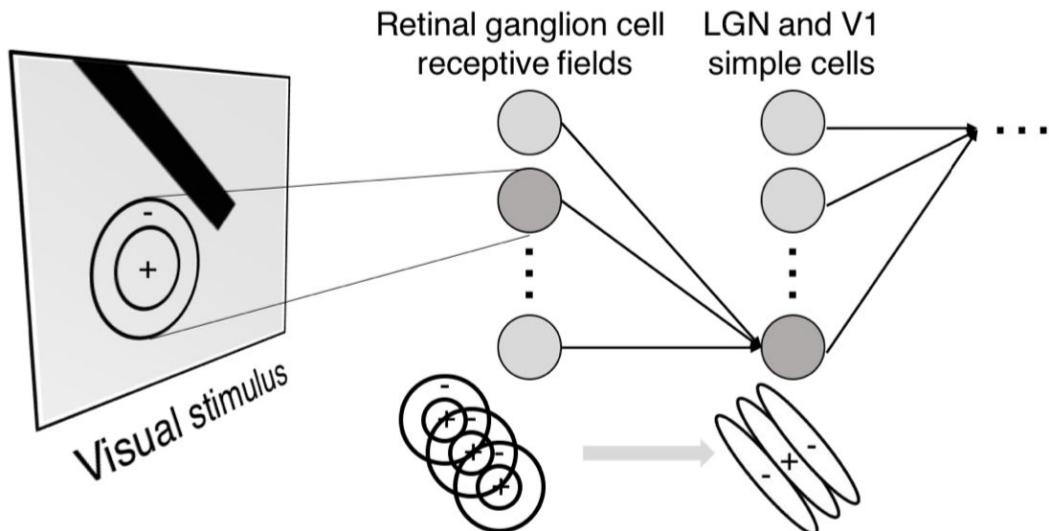
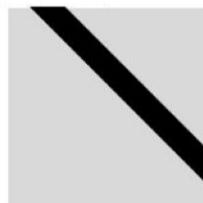


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

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)

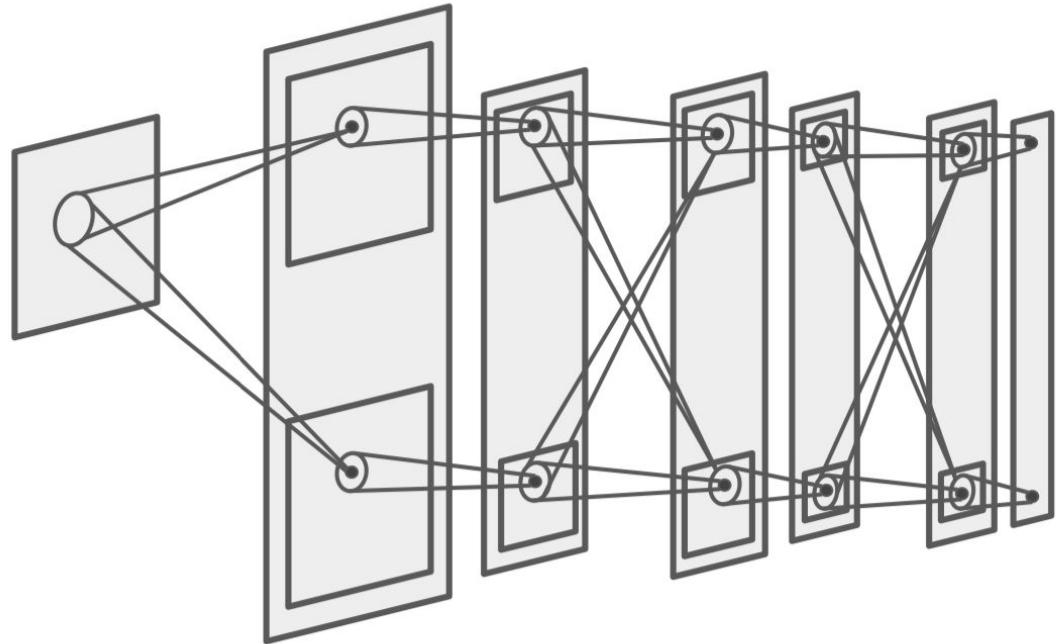
A bit of history:

Neocognitron [Fukushima 1980]

“sandwich” architecture (SCSCSC...)

simple cells: modifiable parameters

complex cells: perform pooling



Fast-forward to today: ConvNets are everywhere

Classification



Retrieval



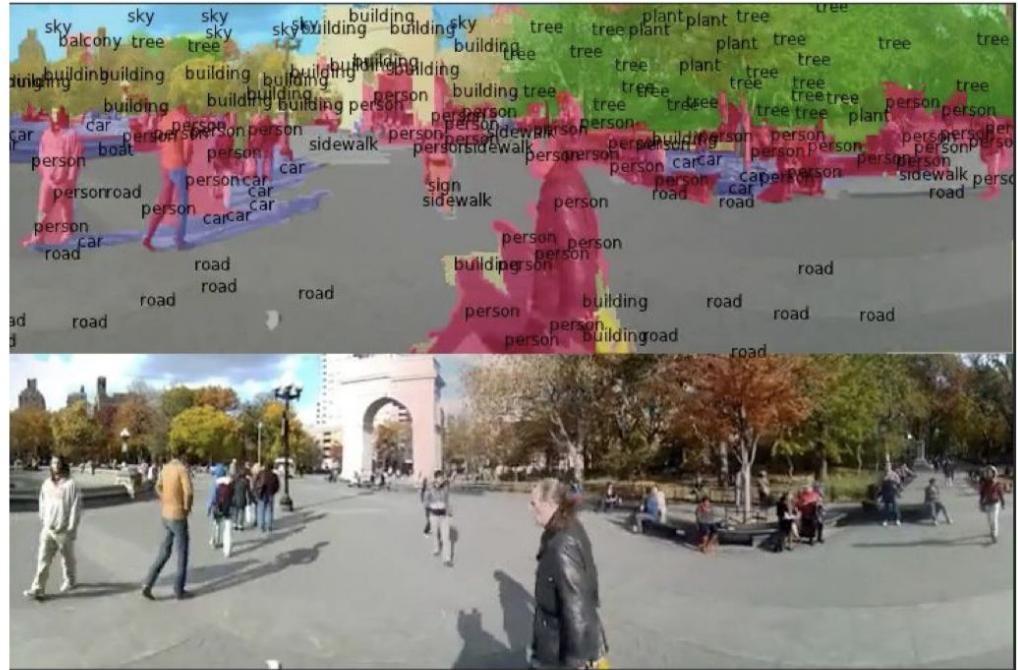
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

Detection



Segmentation



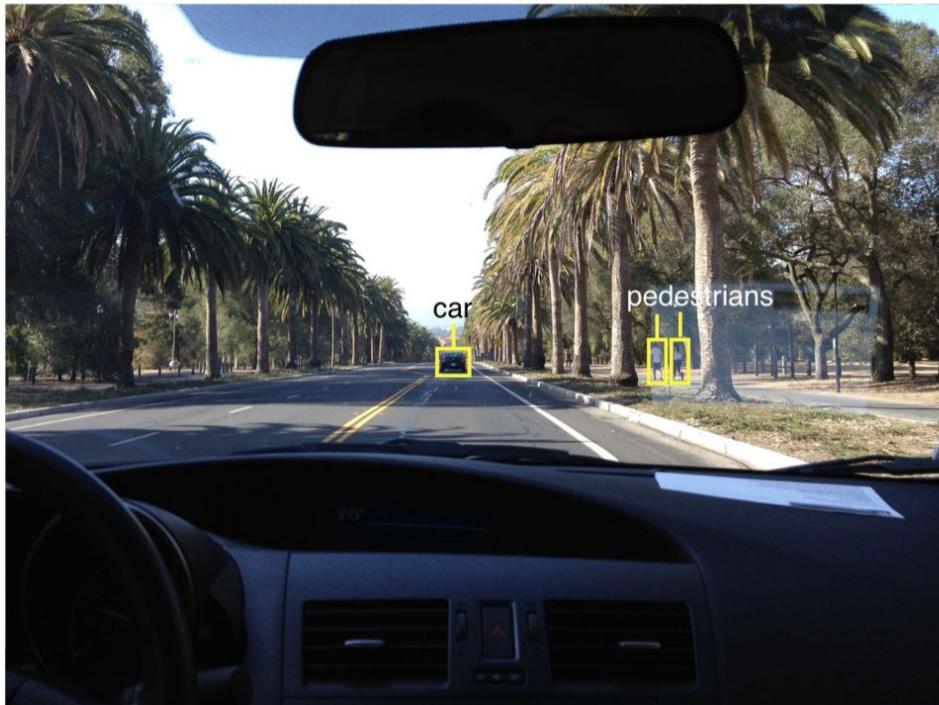
Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

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

Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



[This image](#) by GBPublic_PR is licensed under [CC-BY 2.0](#)

NVIDIA Tesla line

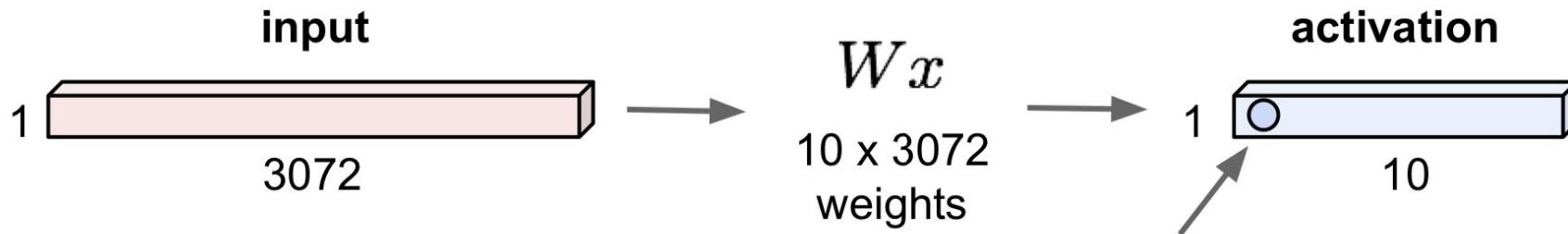
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Diving In...

Fully Connected Layer

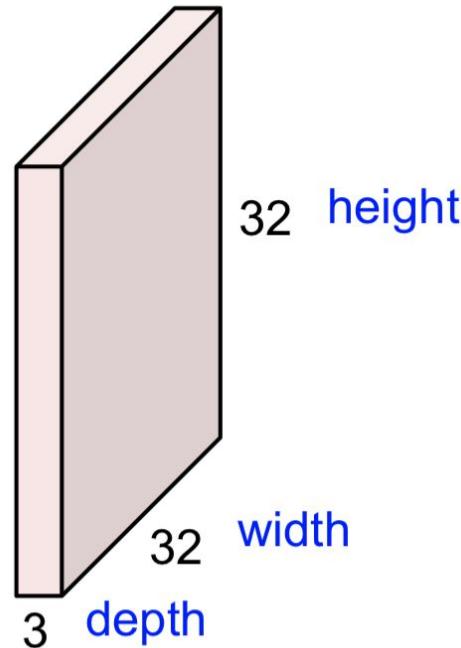
32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

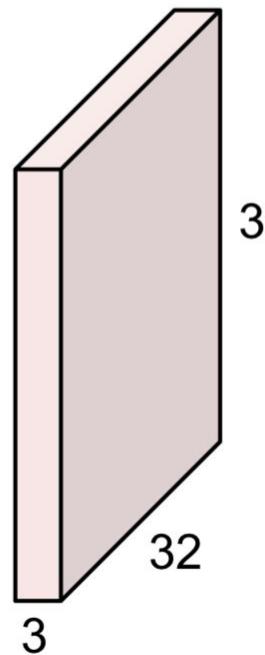
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image



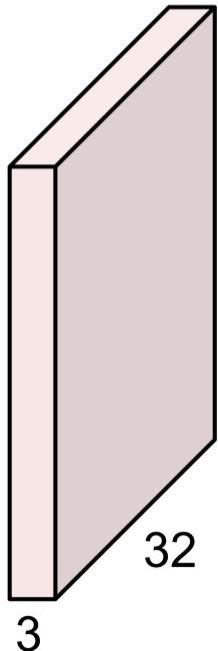
5x5x3 filter



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

Convolution Layer

32x32x3 image



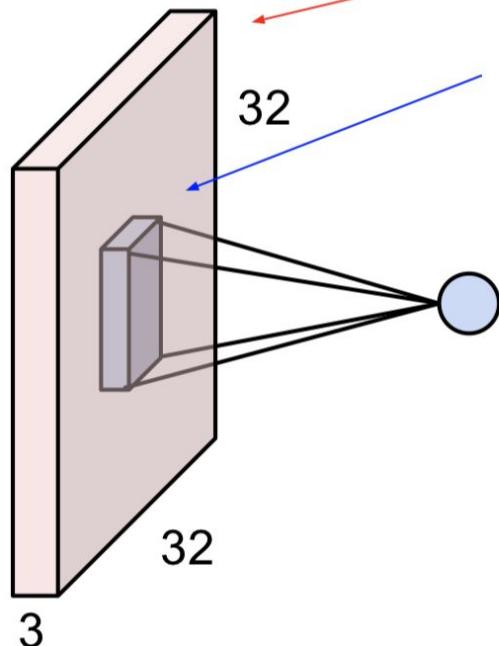
5x5x3 filter



Filters always extend the full depth of the input volume

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

Convolution Layer



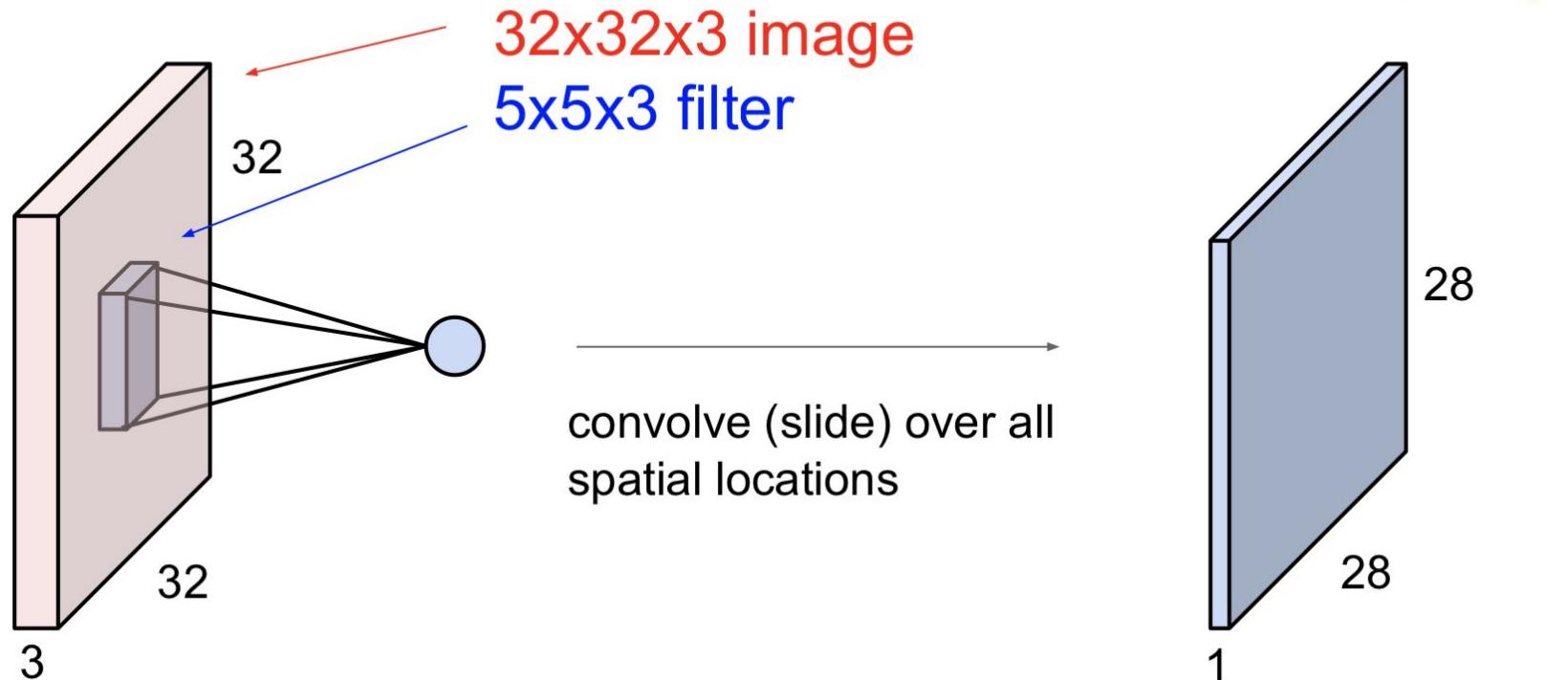
32x32x3 image
5x5x3 filter w

1 number:

the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

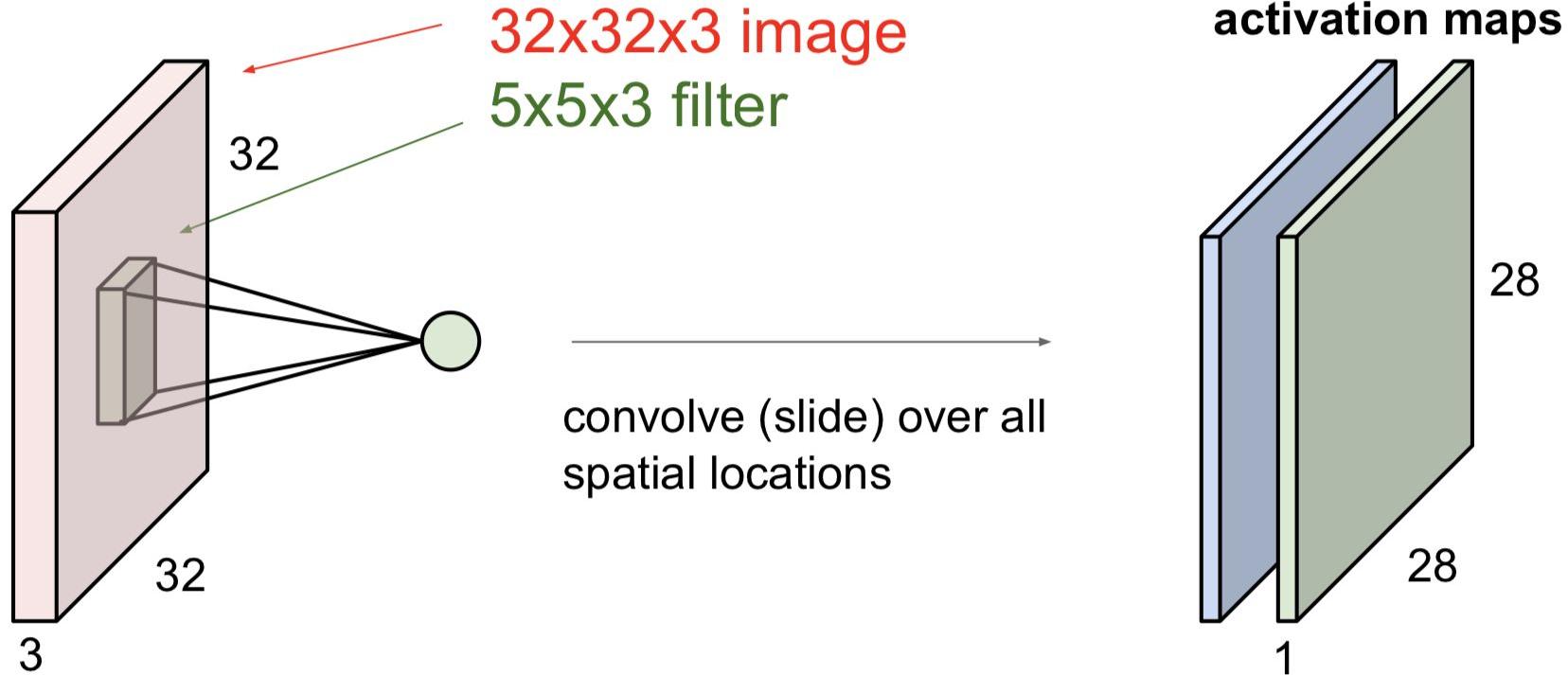
$$w^T x + b$$

Convolution Layer

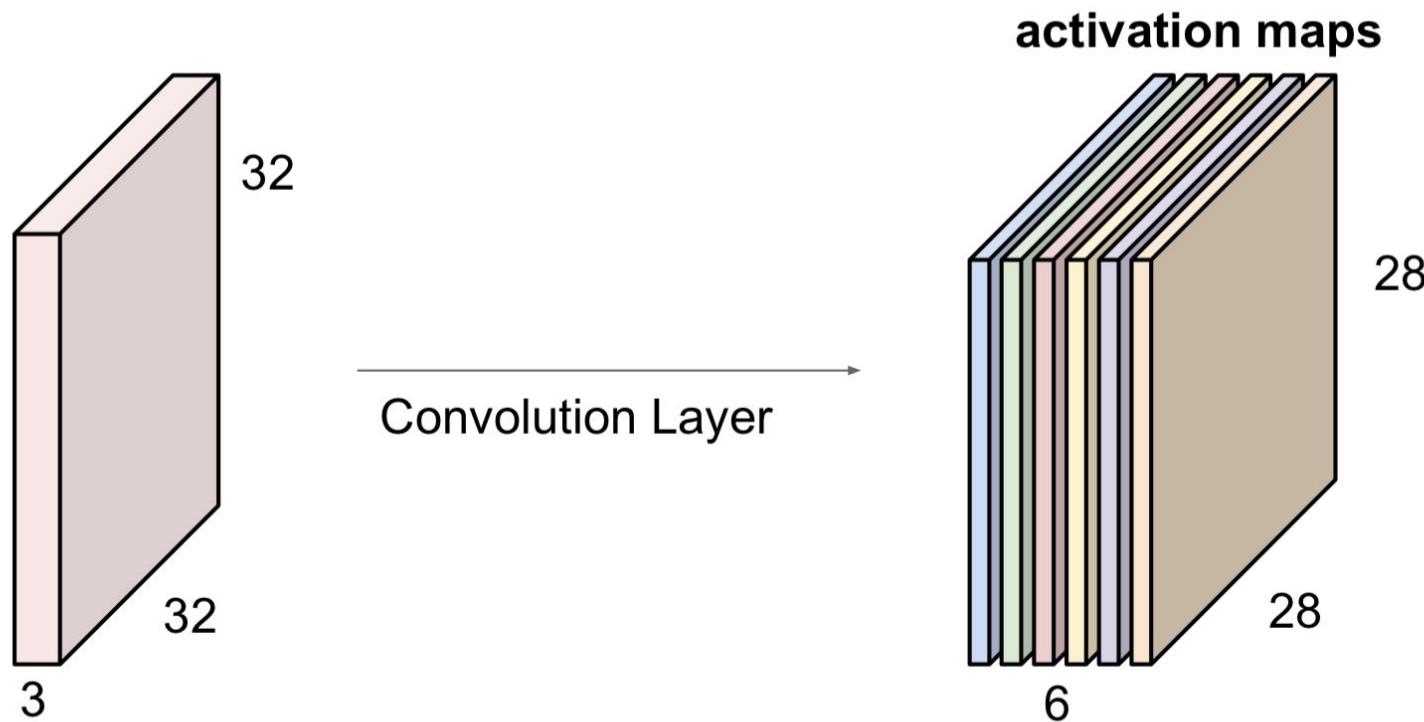


Convolution Layer

consider a second, green filter

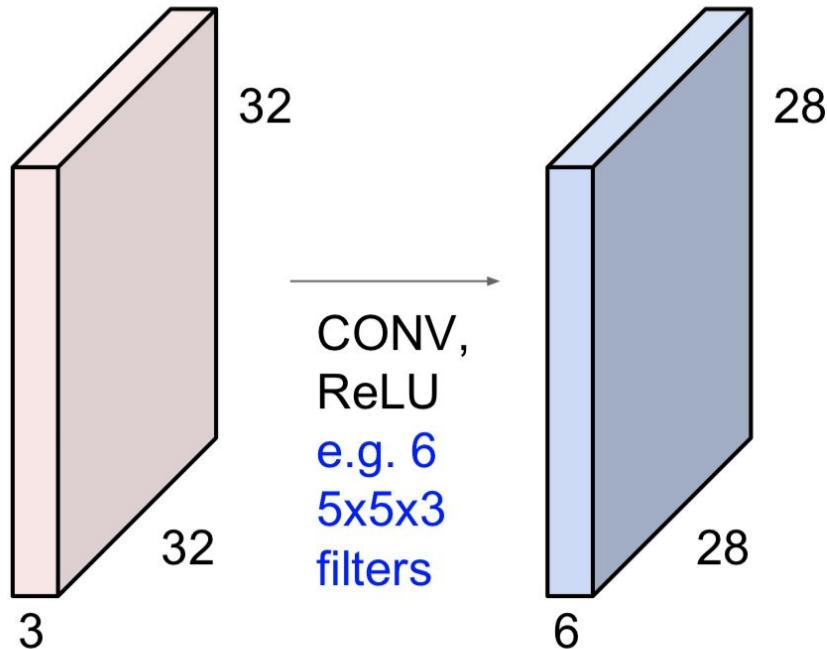


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

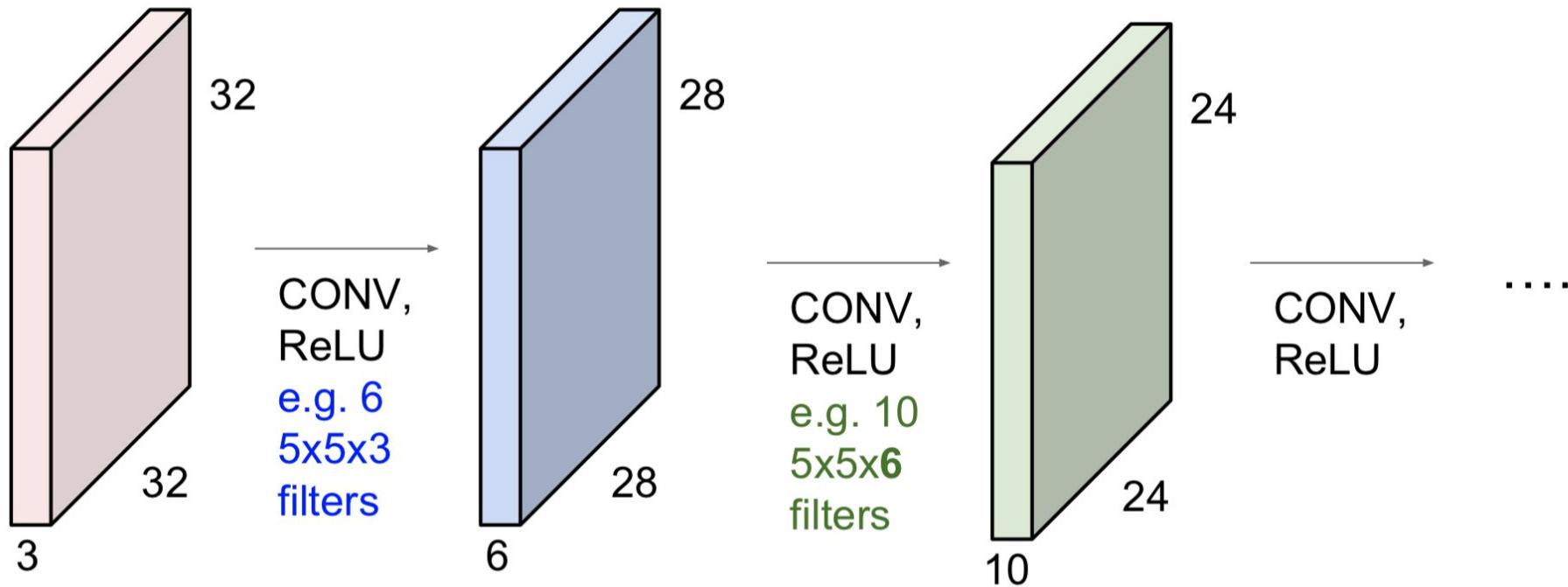


We stack these up to get a “new image” of size 28x28x6!

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



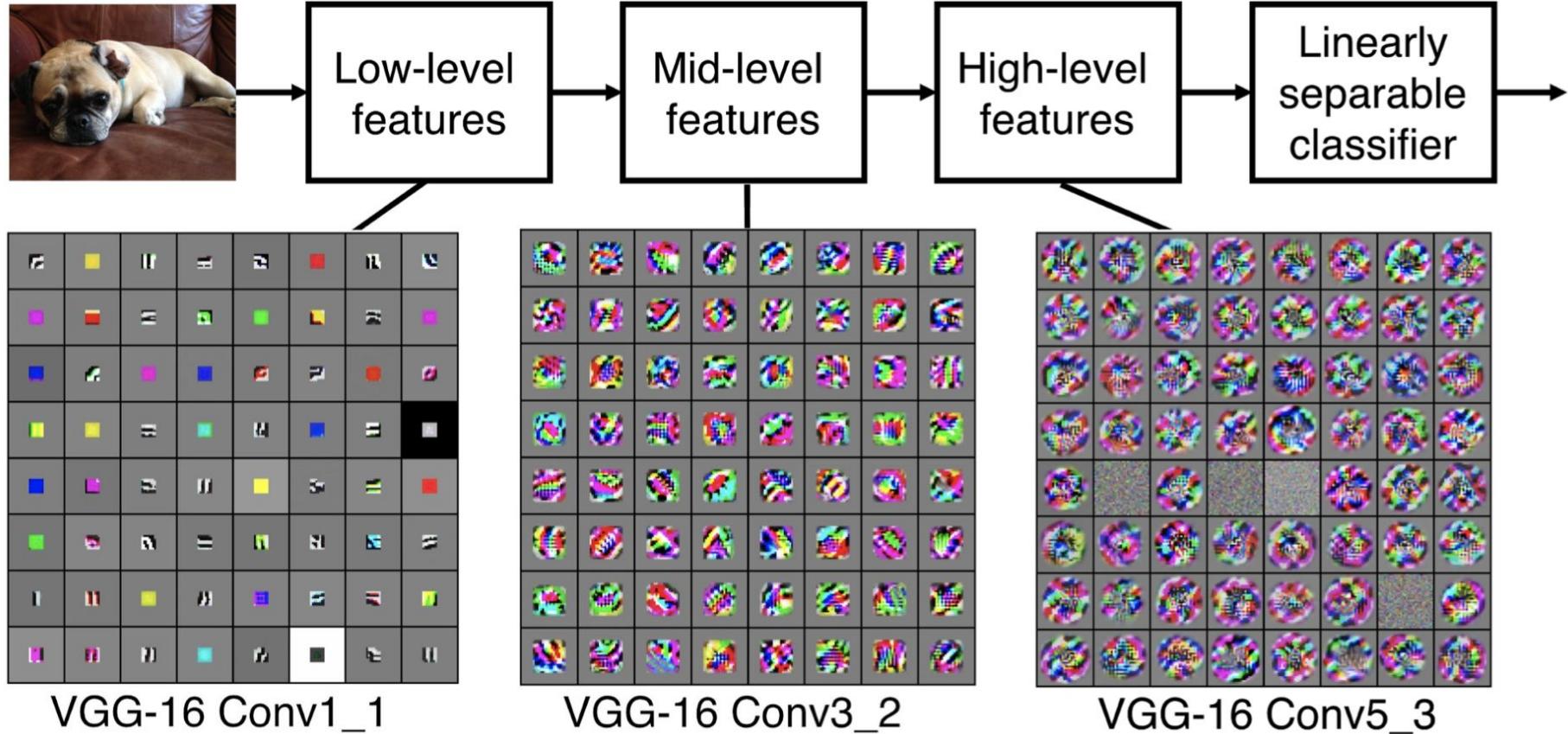
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview

[Zeiler and Fergus 2013]

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

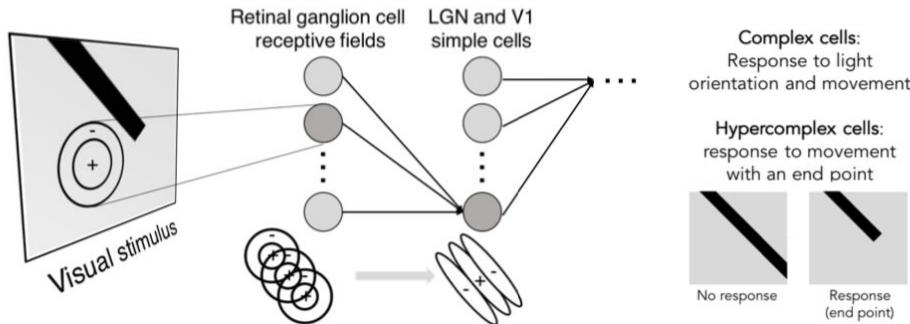
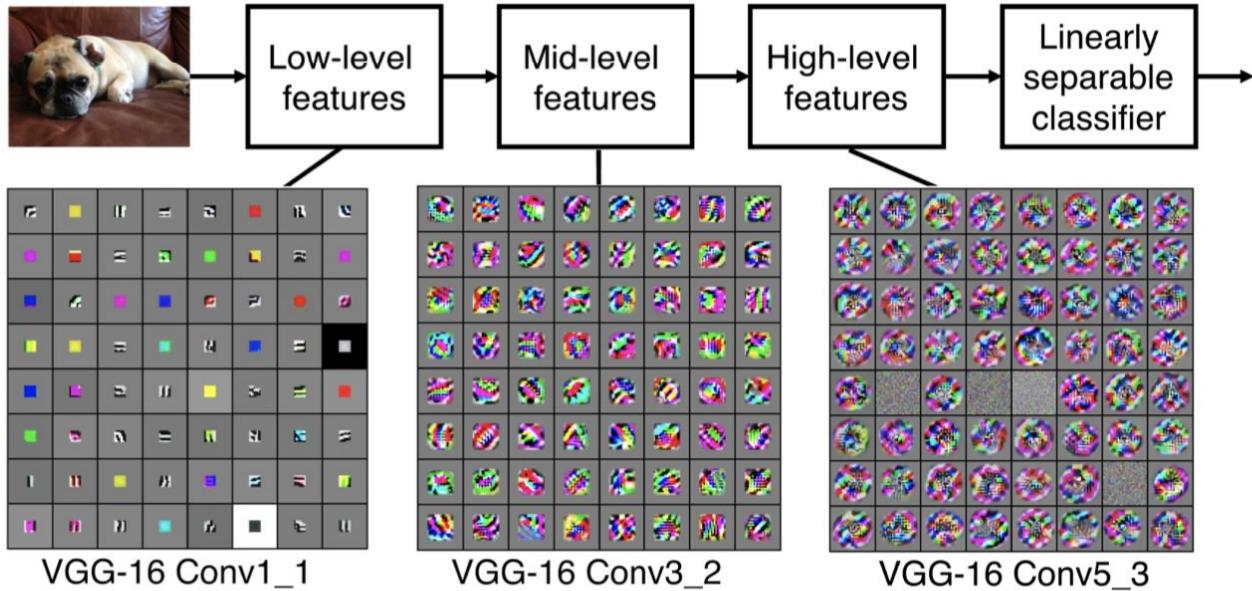


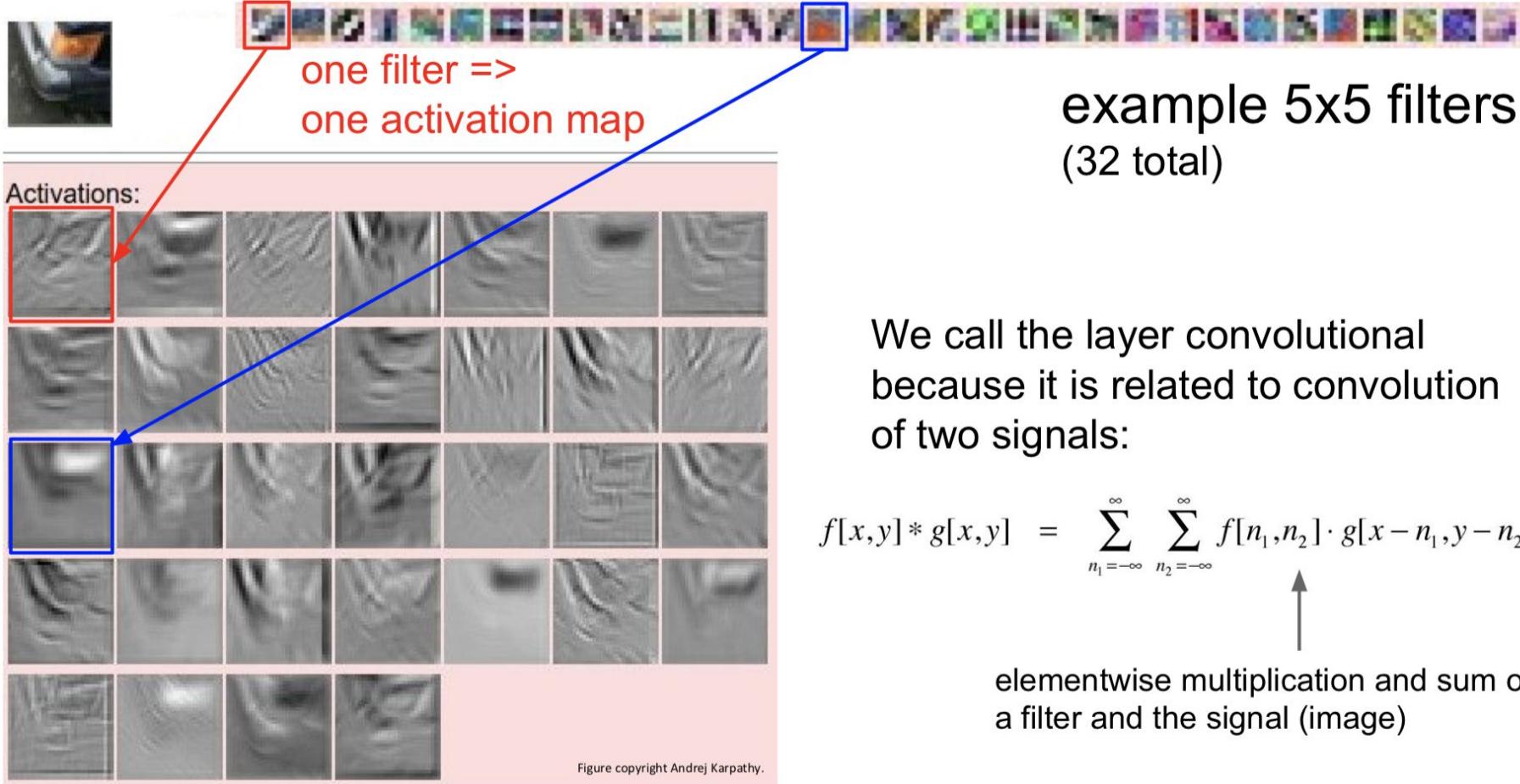
VGG-16 Conv1_1

VGG-16 Conv3_2

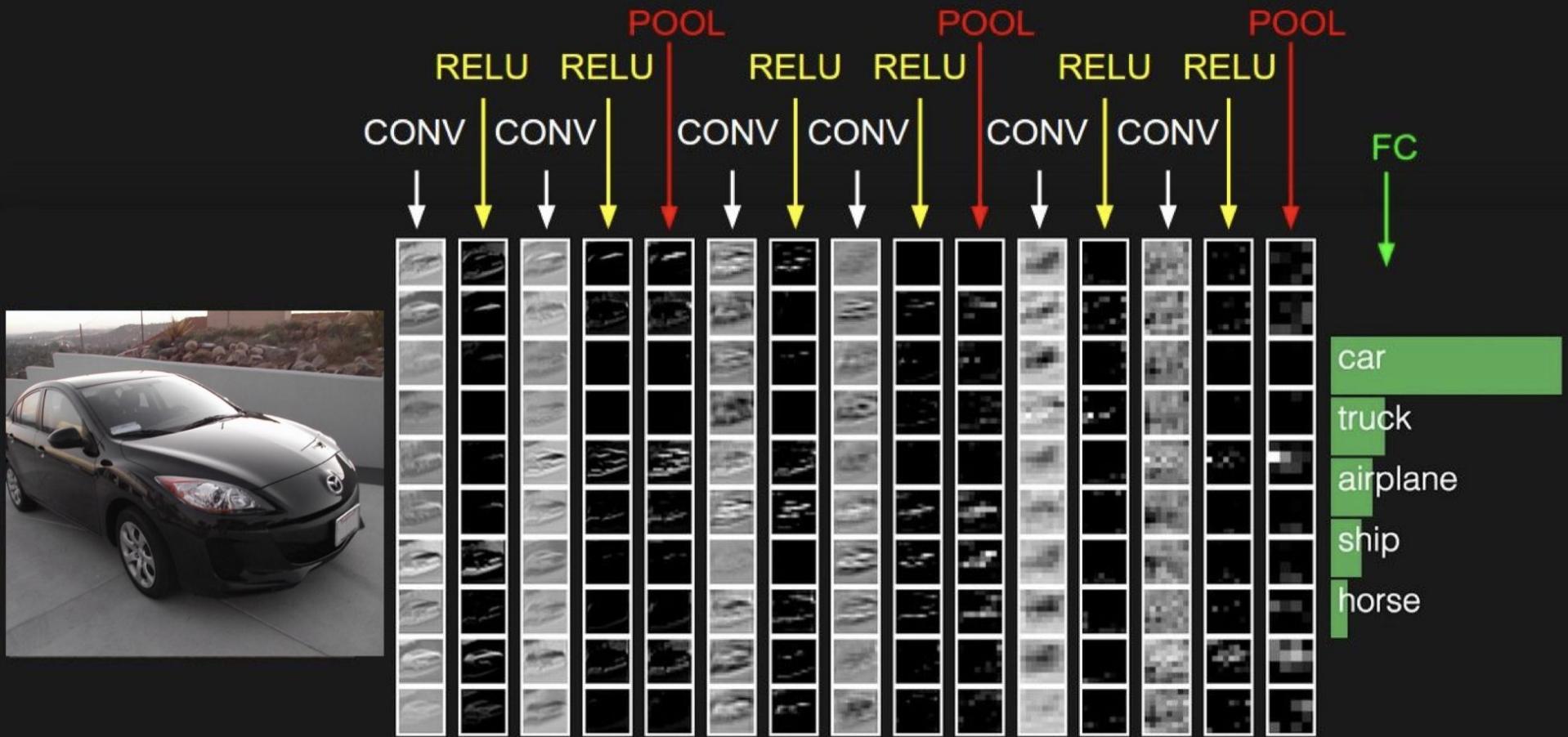
VGG-16 Conv5_3

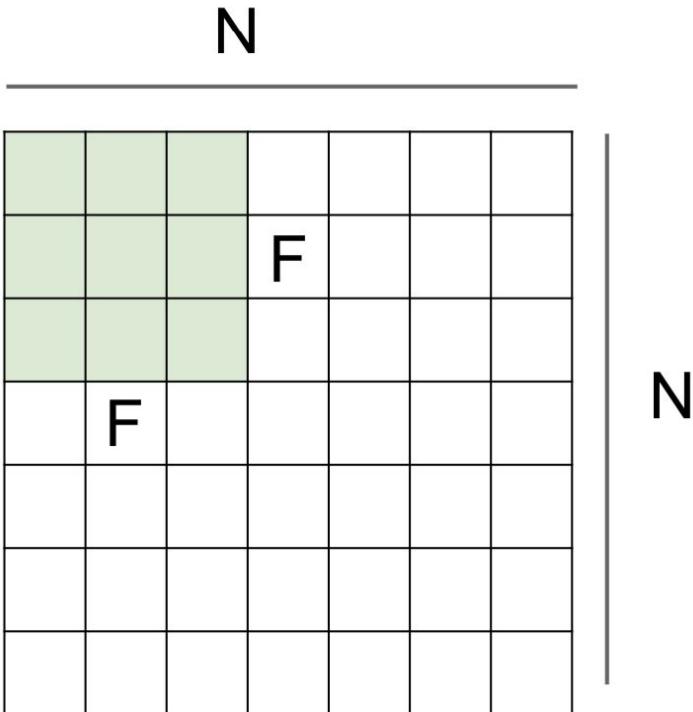
Preview





preview:



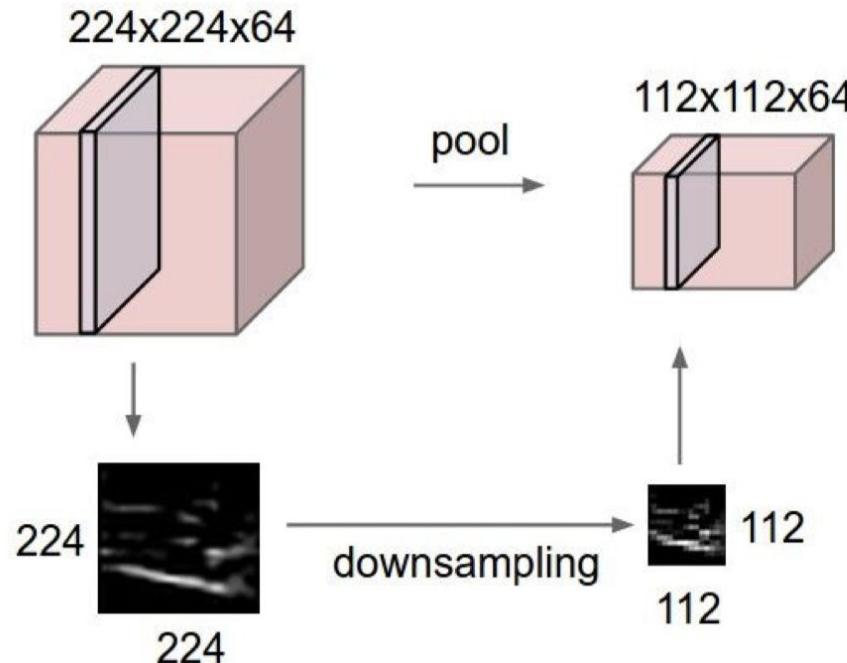


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33$:\

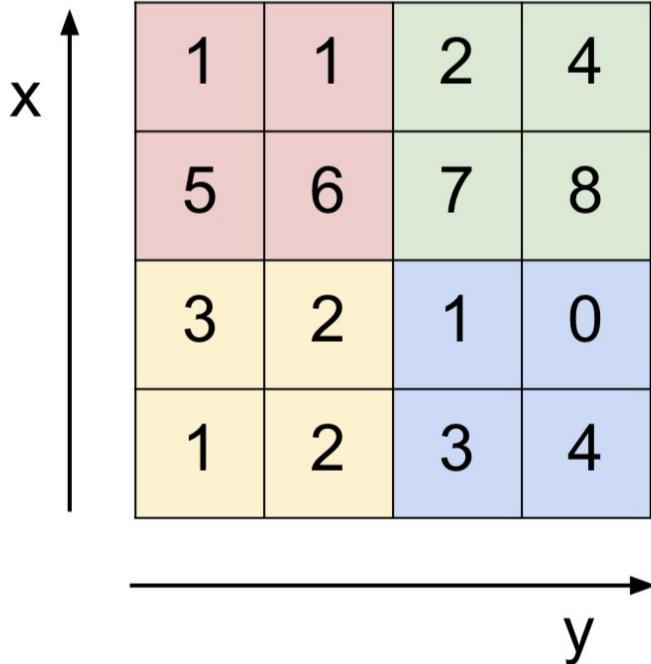
Pooling layer

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

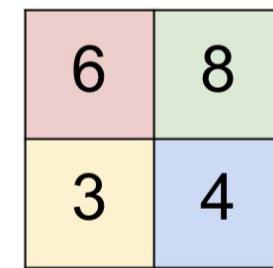


MAX POOLING

Single depth slice

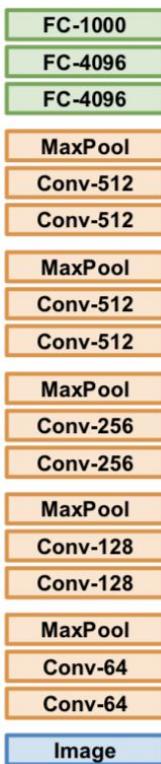


max pool with 2x2 filters
and stride 2



Transfer Learning with CNNs

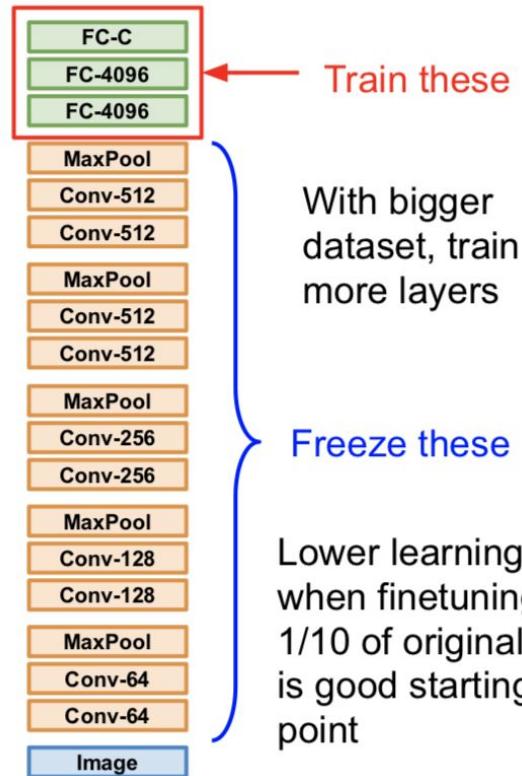
1. Train on Imagenet



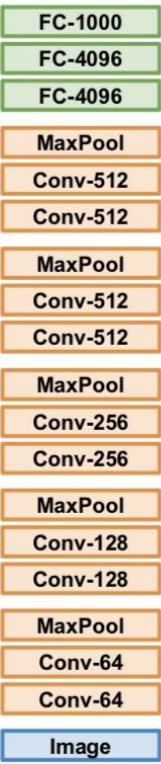
2. Small Dataset (C classes)



3. Bigger dataset



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

CNN Architectures

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- SENet
- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- DenseNet
- FractalNet
- MobileNets
- NASNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

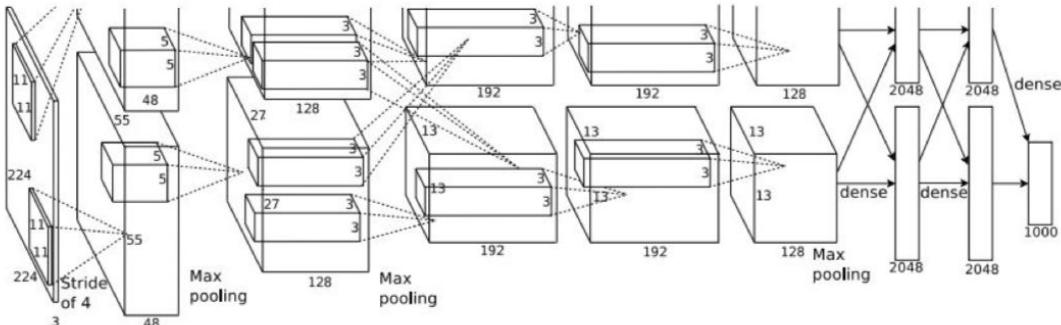


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

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)

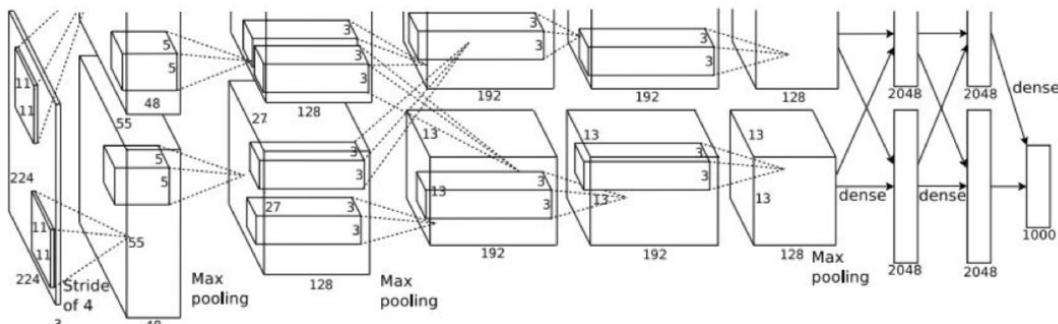
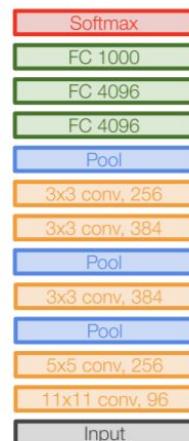


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

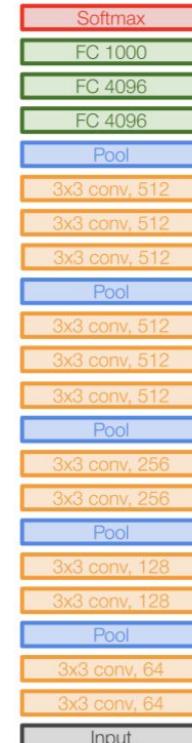
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

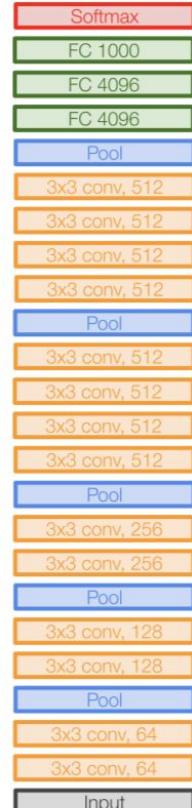
Q: Why use smaller filters? (3x3 conv)



AlexNet



VGG16



VGG19

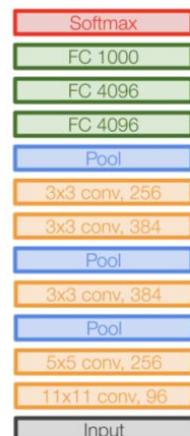
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

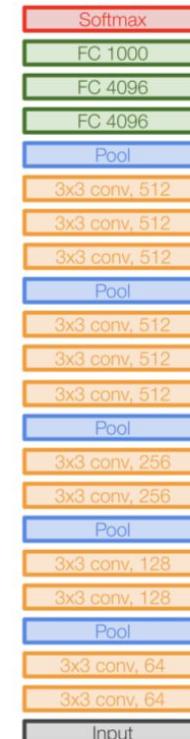
Q: Why use smaller filters? (3x3 conv)

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

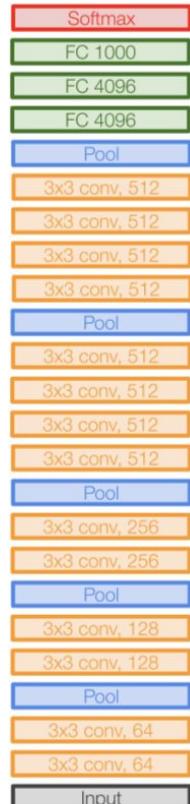
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



AlexNet



VGG16



VGG19

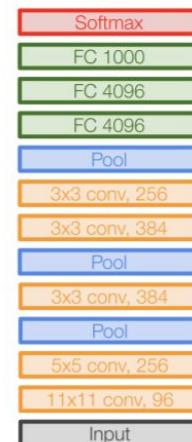
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

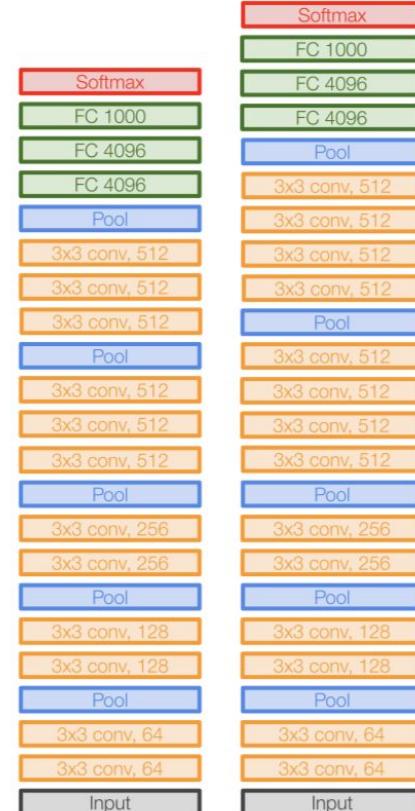
Q: Why use smaller filters? (3x3 conv)

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

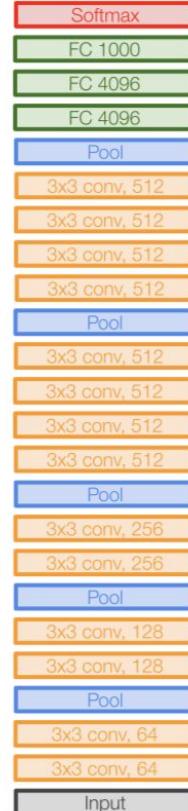
[7x7]



AlexNet



VGG16



VGG19

Case Study: VGGNet

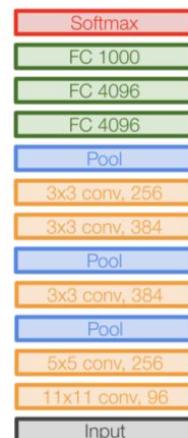
[Simonyan and Zisserman, 2014]

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

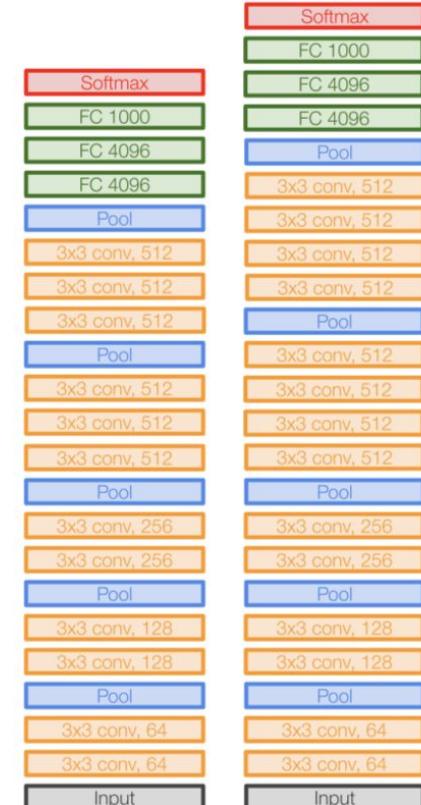
Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer

But deeper, more non-linearities

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



AlexNet



VGG16

VGG19

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

Note:

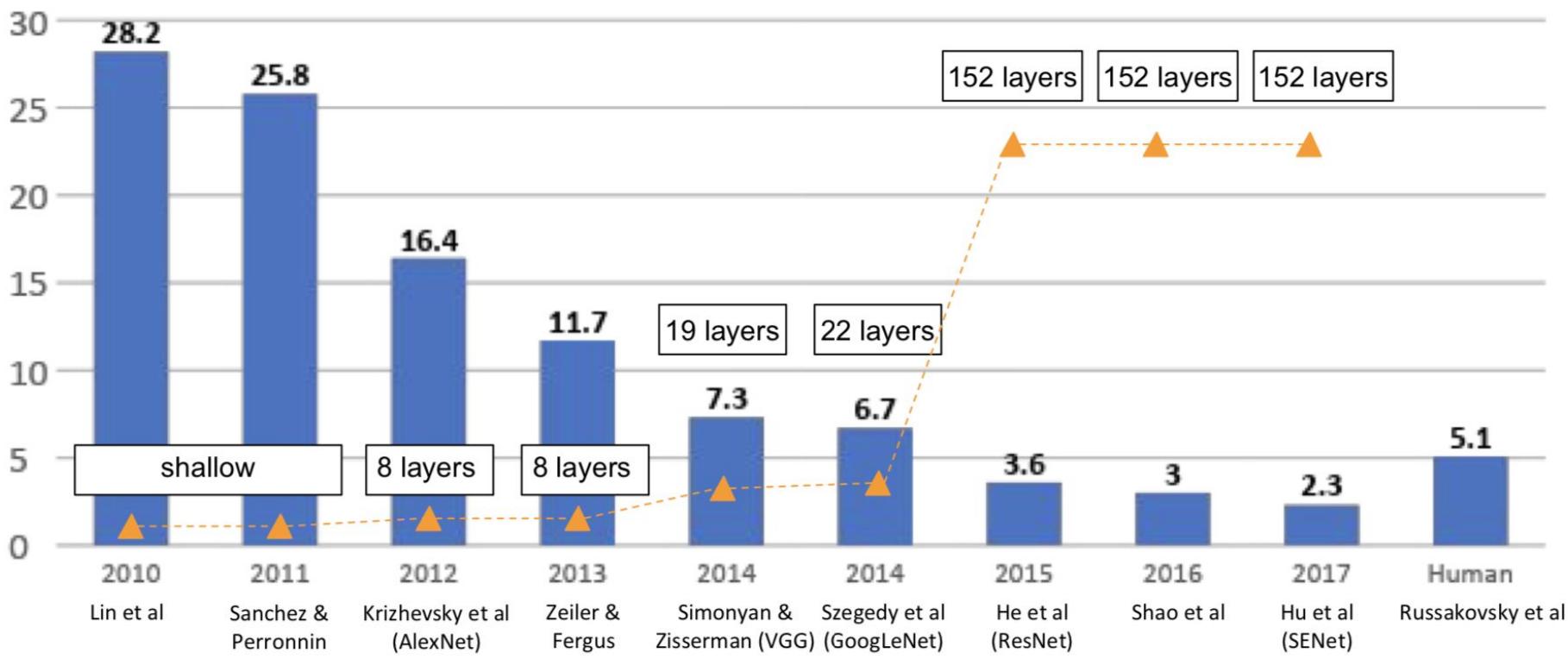
Most memory is in early CONV

Most params are in late FC

TOTAL memory: $24M \times 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

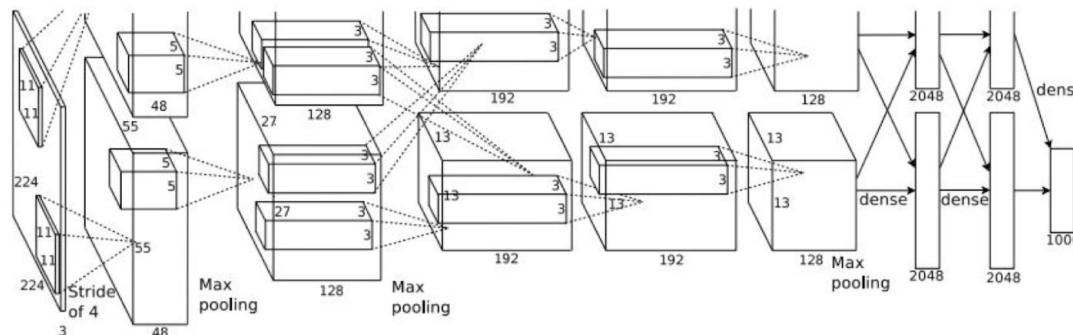


What's going on inside ConvNets?

This image is CC0 public domain



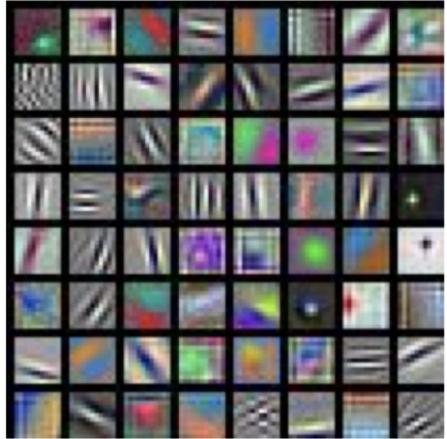
Input Image:
 $3 \times 224 \times 224$



What are the intermediate features looking for?

Class Scores:
1000 numbers

First Layer: Visualize Filters



AlexNet:
 $64 \times 3 \times 11 \times 11$



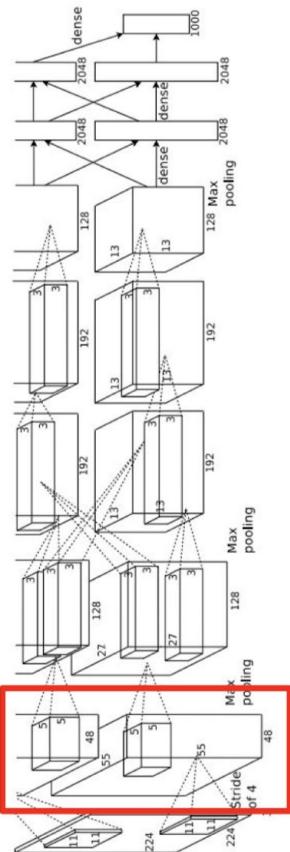
ResNet-18:
 $64 \times 3 \times 7 \times 7$



ResNet-101:
 $64 \times 3 \times 7 \times 7$



DenseNet-121:
 $64 \times 3 \times 7 \times 7$



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(these are taken from ConvNetJS
CIFAR-10 demo)



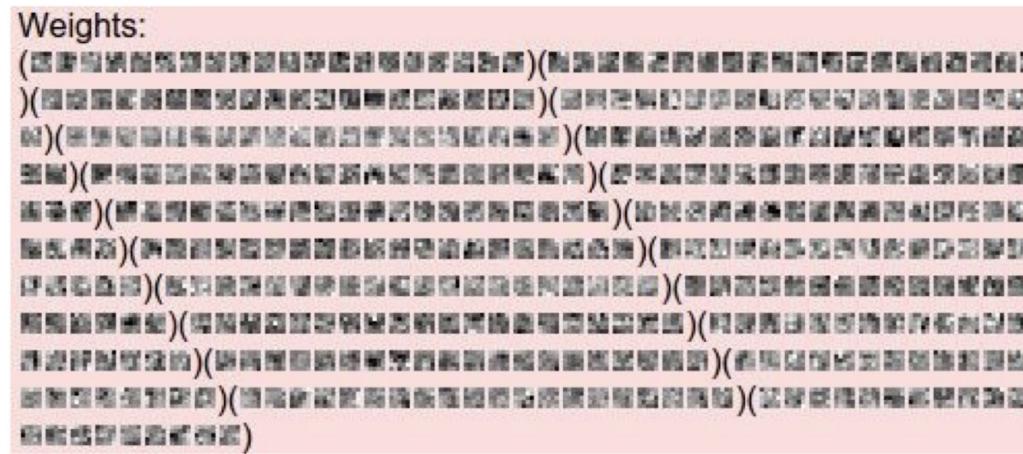
layer 1 weights

$16 \times 3 \times 7 \times 7$



layer 2 weights

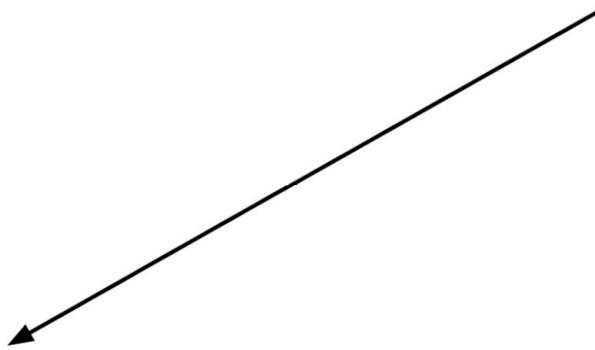
$20 \times 16 \times 7 \times 7$



layer 3 weights

$20 \times 20 \times 7 \times 7$

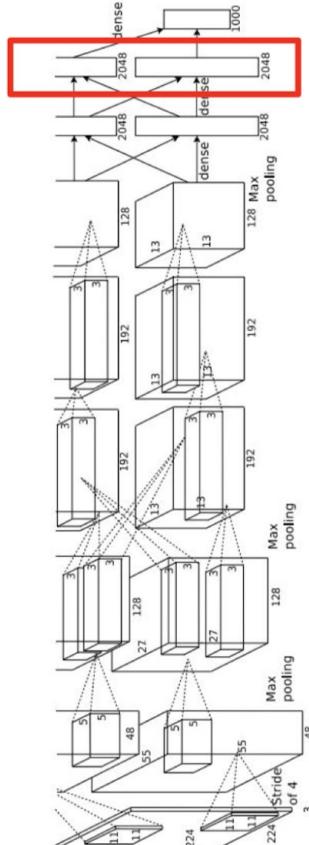
Last Layer



FC7 layer

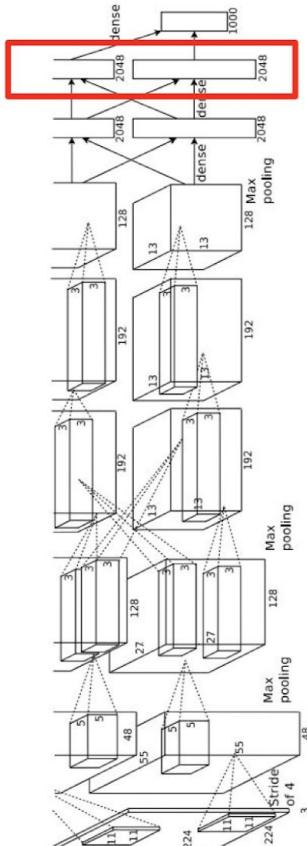
4096-dimensional feature vector for an image
(layer immediately before the classifier)

Run the network on many images, collect the
feature vectors



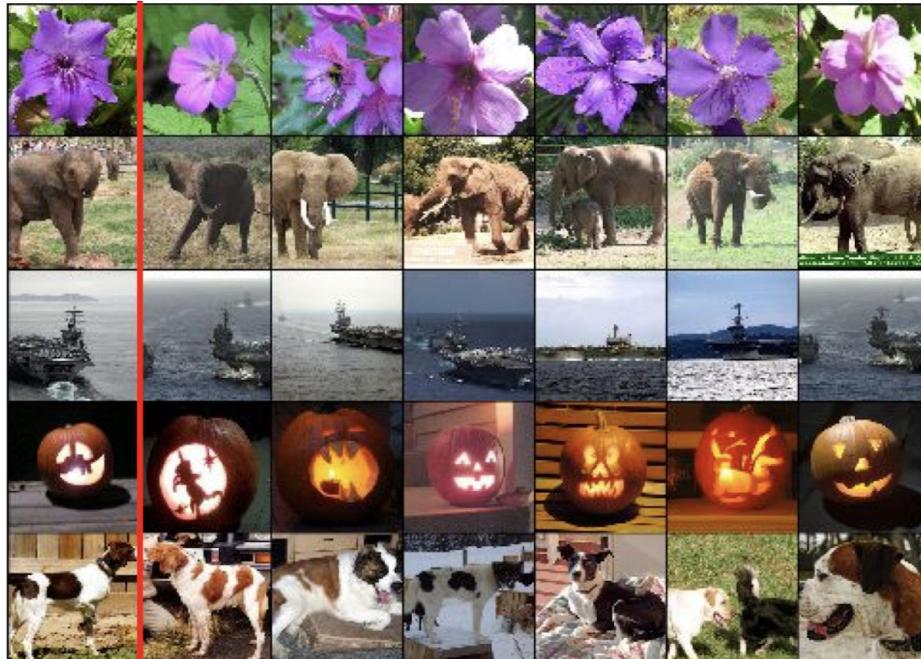
Last Layer: Nearest Neighbors

4096-dim vector

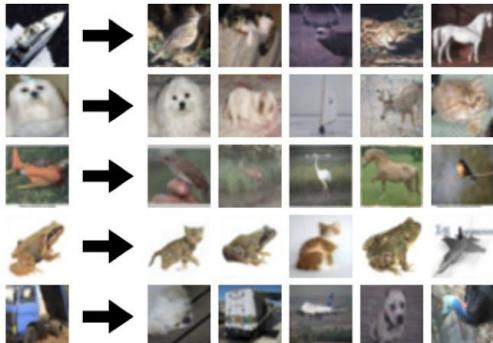


Test image

L2 Nearest neighbors in feature space



Recall: Nearest neighbors
in pixel space

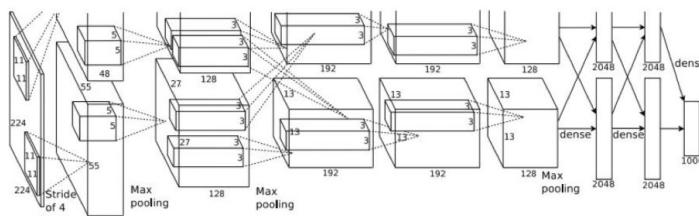


Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.

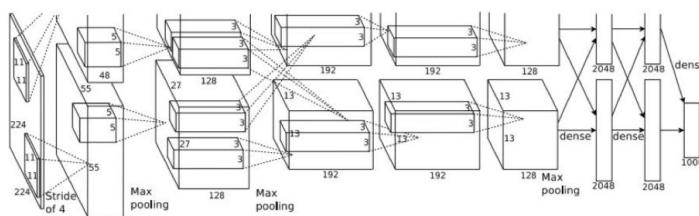
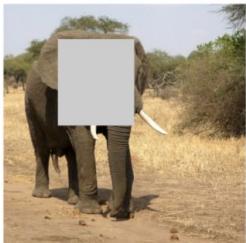
Figures reproduced with permission.

Which pixels matter: Saliency via Occlusion

Mask part of the image before feeding to CNN,
check how much predicted probabilities change



$$P(\text{elephant}) = 0.95$$



$$P(\text{elephant}) = 0.75$$

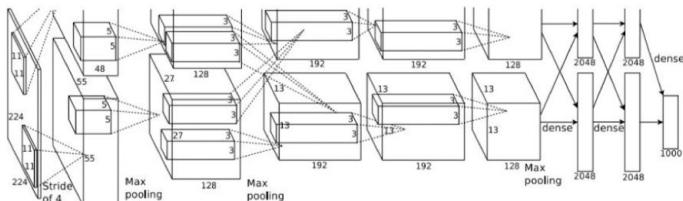
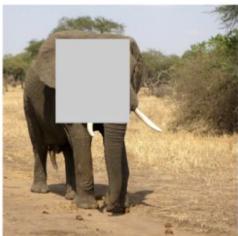
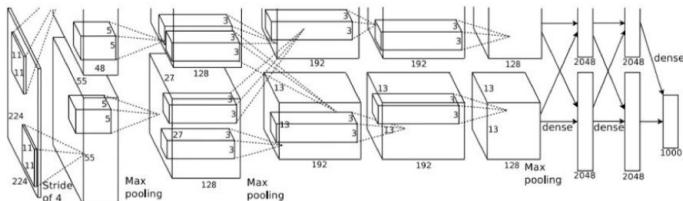
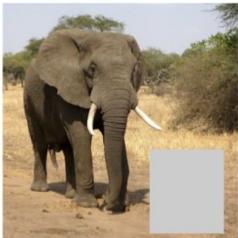
Boat image is CC0 public domain

Elephant image is CC0 public domain

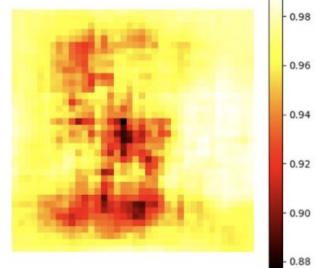
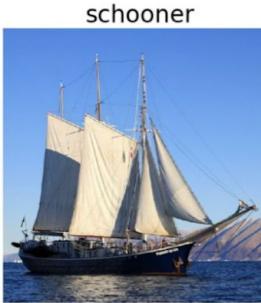
Go-Karts image is CC0 public domain

Which pixels matter: Saliency via Occlusion

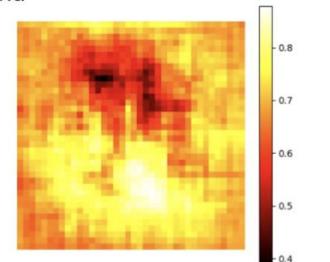
Mask part of the image before feeding to CNN,
check how much predicted probabilities change



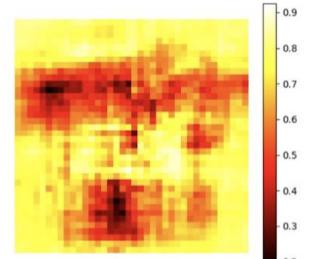
Boat image is CC0 public domain
Elephant image is CC0 public domain
Go-Karts image is CC0 public domain



African elephant, *Loxodonta africana*

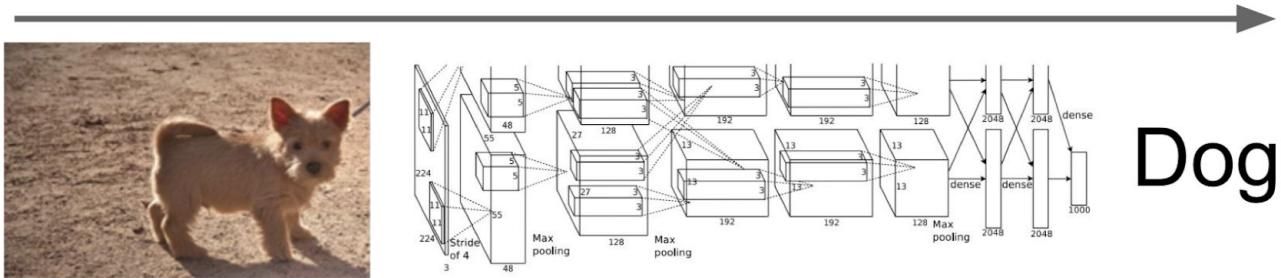


go-kart



Which pixels matter: Saliency via Backprop

Forward pass: Compute probabilities

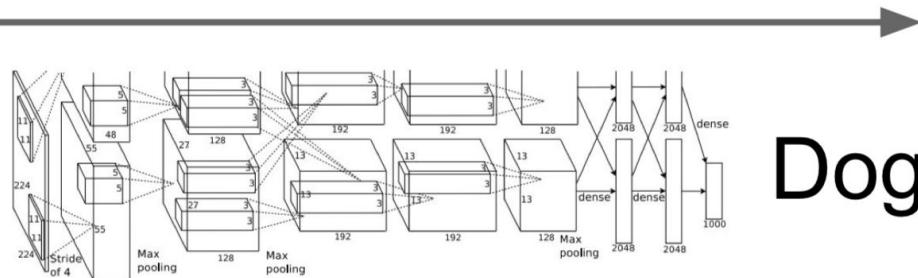


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

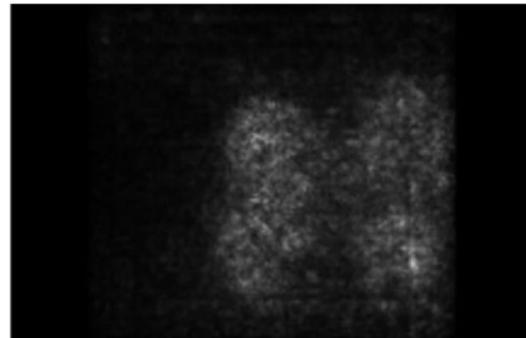
Which pixels matter: Saliency via Backprop

Forward pass: Compute probabilities

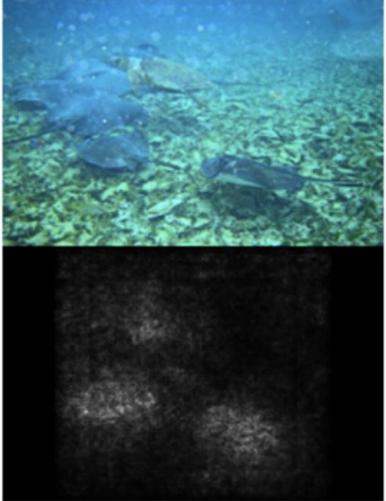
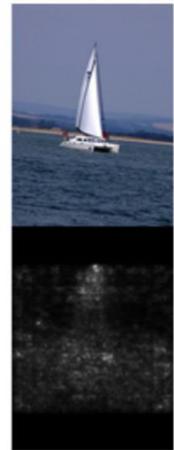


Dog

Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels



Saliency Maps

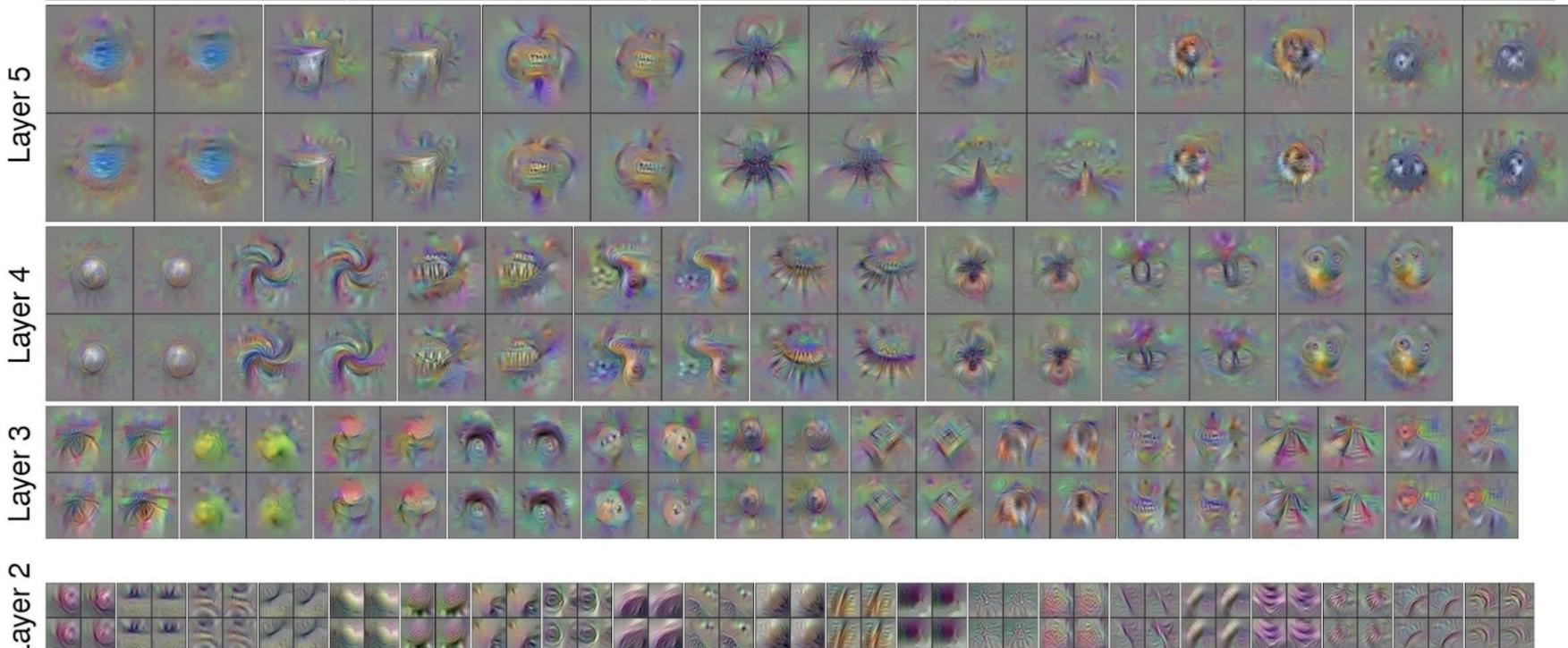


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

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 2014.
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

Visualizing CNN features: Gradient Ascent



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.
Figures copyright Anh Nguyen, Jason Yosinski, and Jeff Clune, 2016; reproduced with permission.

Copyrights@CS231N, Stanford

Fooling Images / Adversarial Examples

- (1) Start from an arbitrary image
- (2) Pick an arbitrary class
- (3) Modify the image to maximize the class
- (4) Repeat until network is fooled

Fooling Images / Adversarial Examples

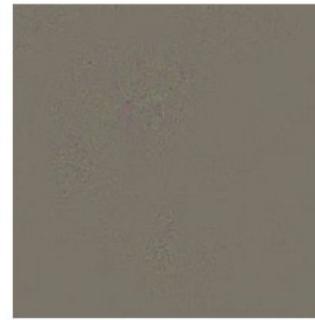
African elephant



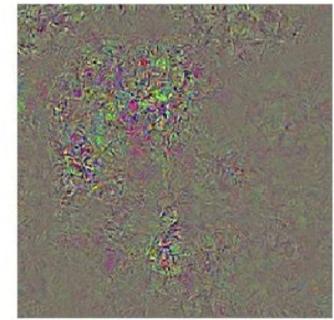
koala



Difference



10x Difference



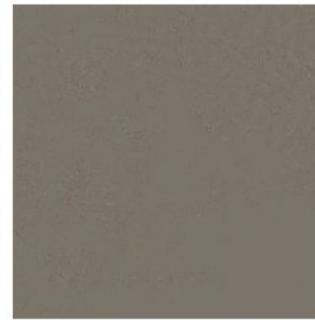
schooner



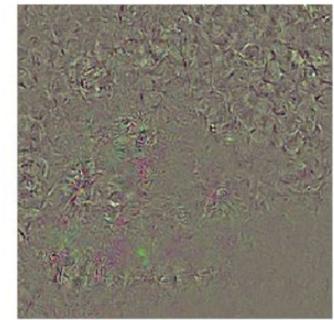
iPod



Difference



10x Difference



Fooling Images / Adversarial Examples

African elephant



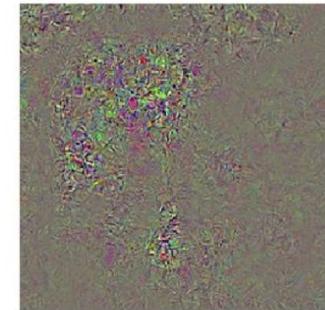
koala



Difference



10x Difference



schooner



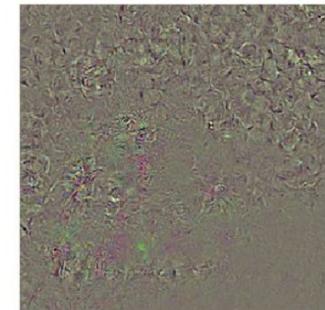
iPod



Difference

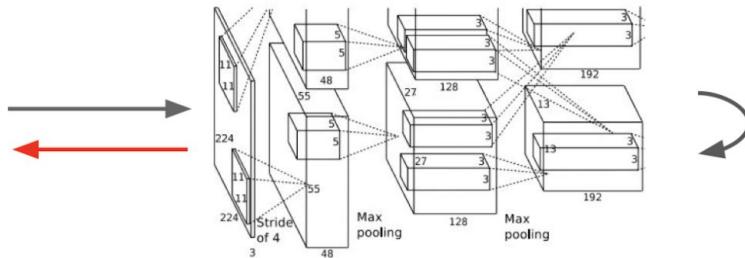


10x Difference



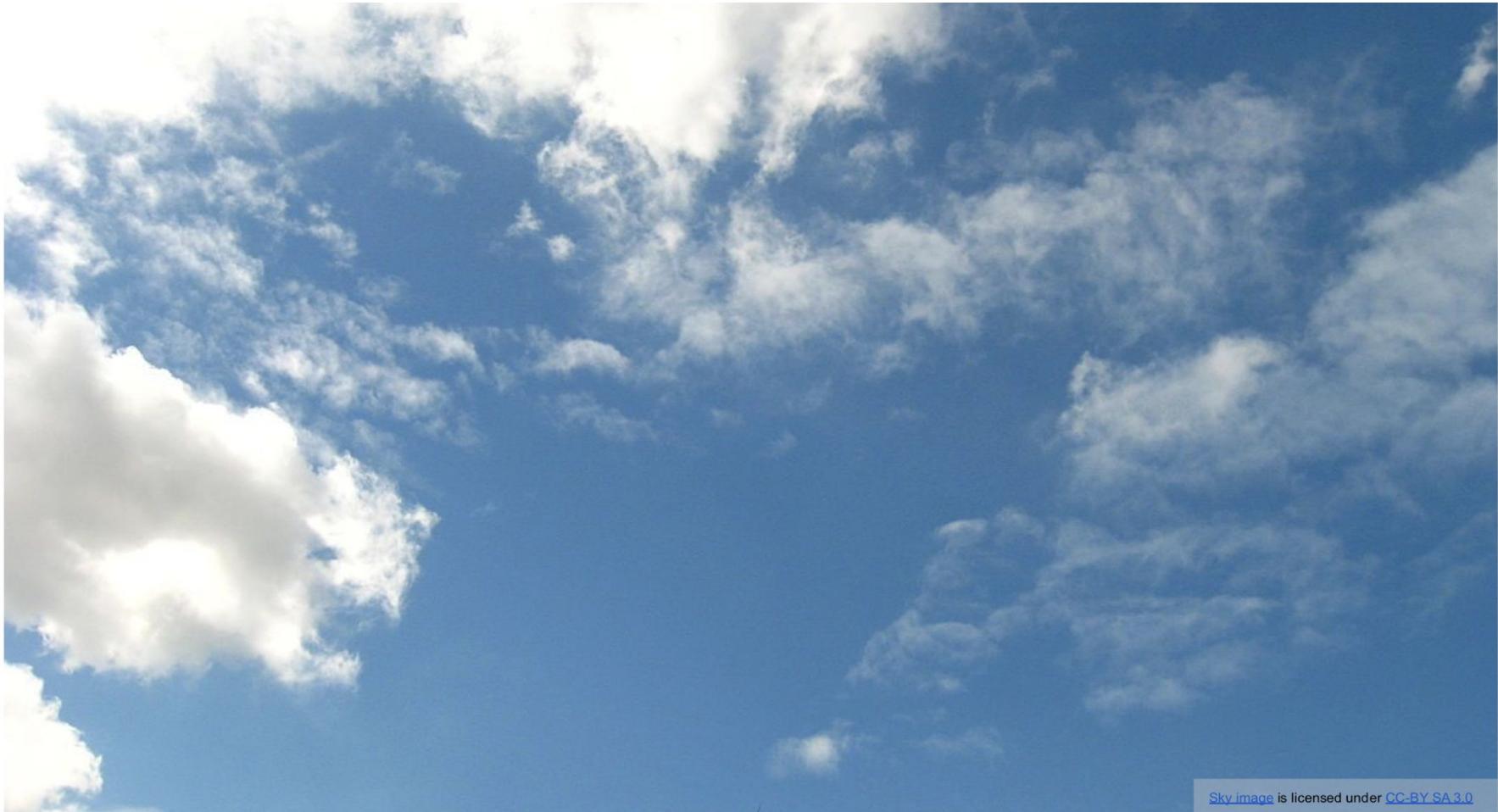
DeepDream: Amplify existing features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network



Choose an image and a layer in a CNN; repeat:

1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image



[Sky image](#) is licensed under [CC-BY SA 3.0](#)

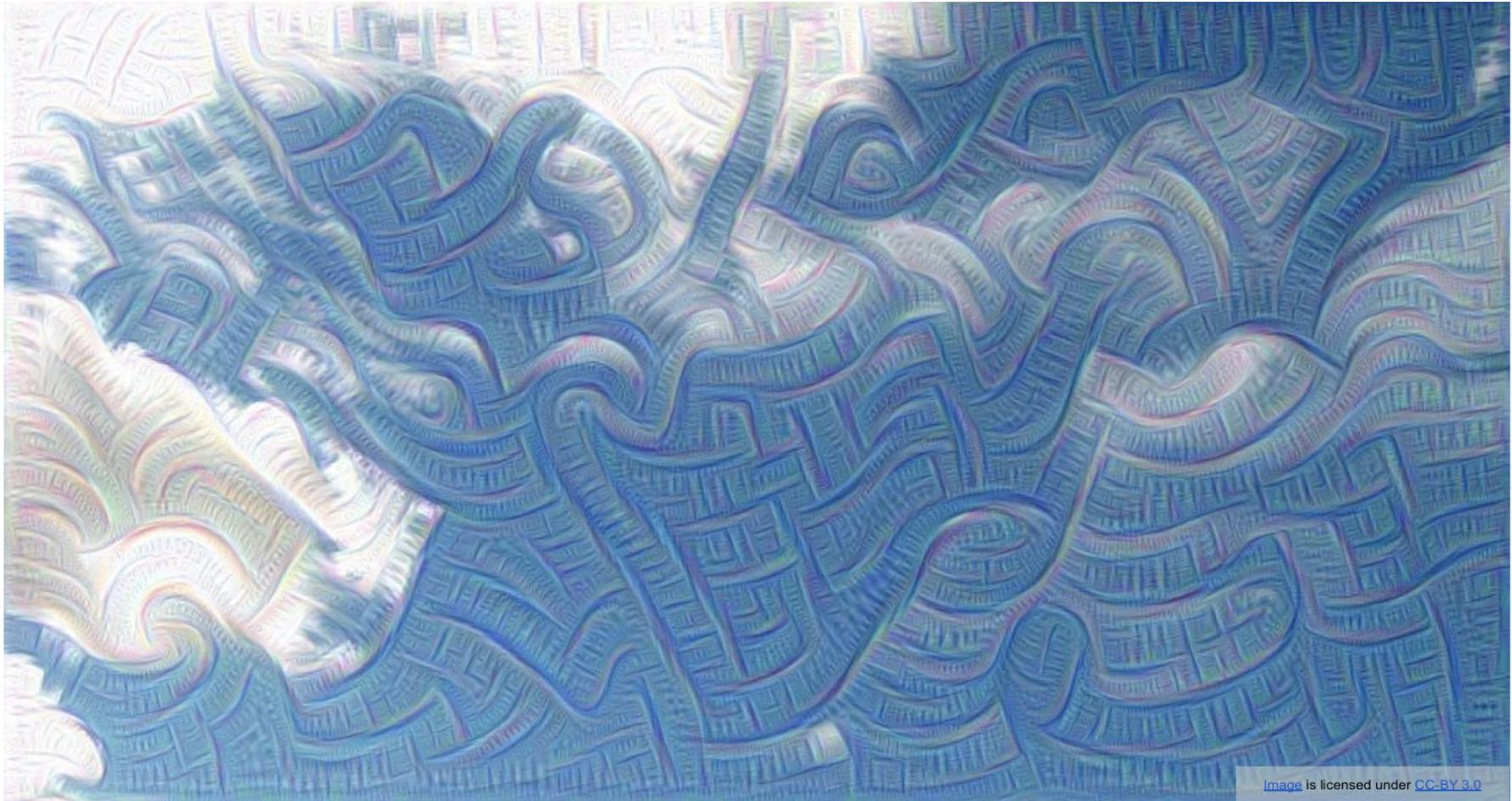
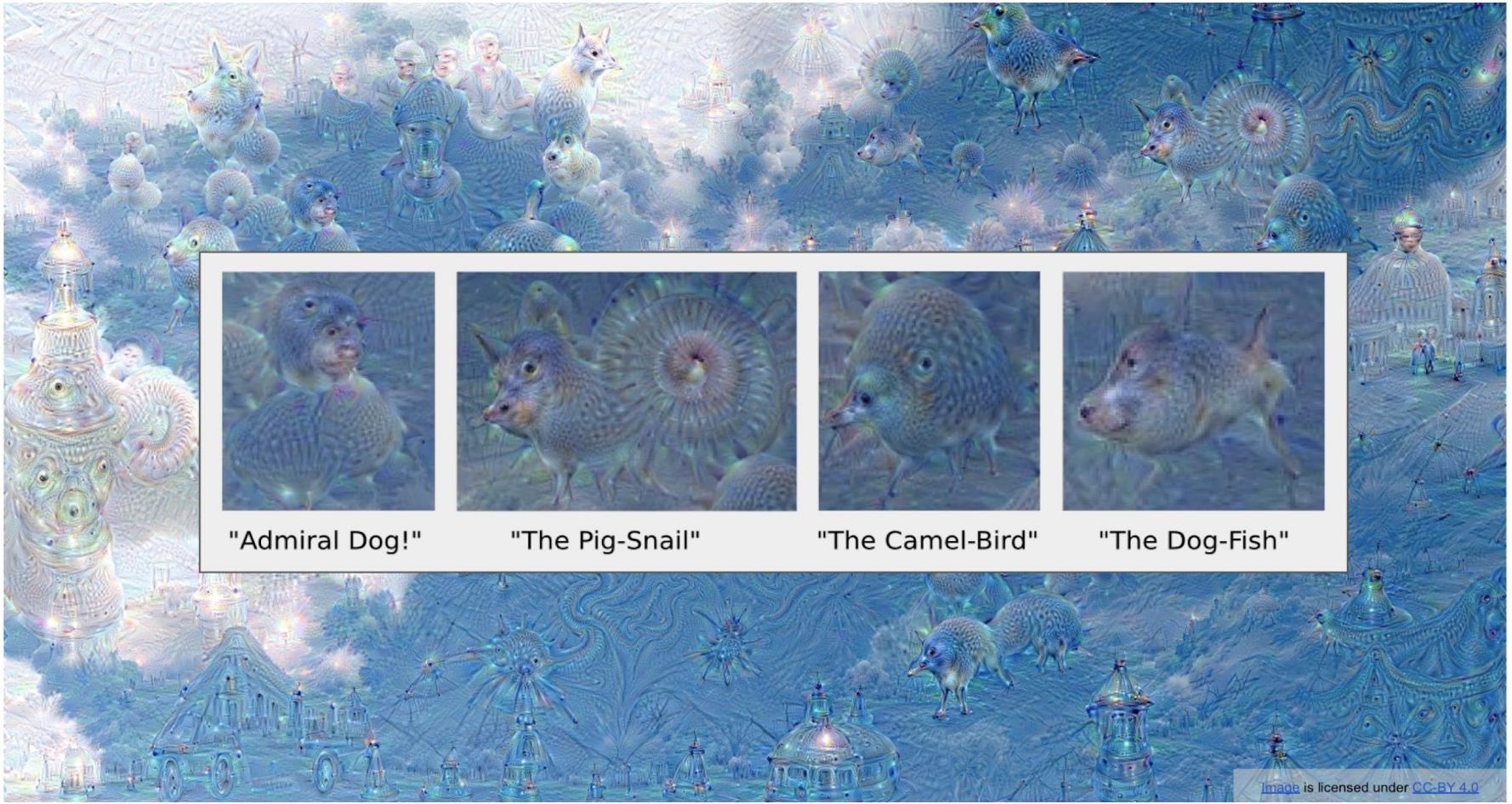


Image is licensed under CC-BY 3.0



"Admiral Dog!"

"The Pig-Snail"

"The Camel-Bird"

"The Dog-Fish"

Image is licensed under CC-BY 4.0

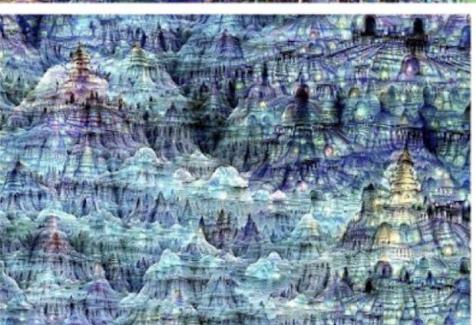
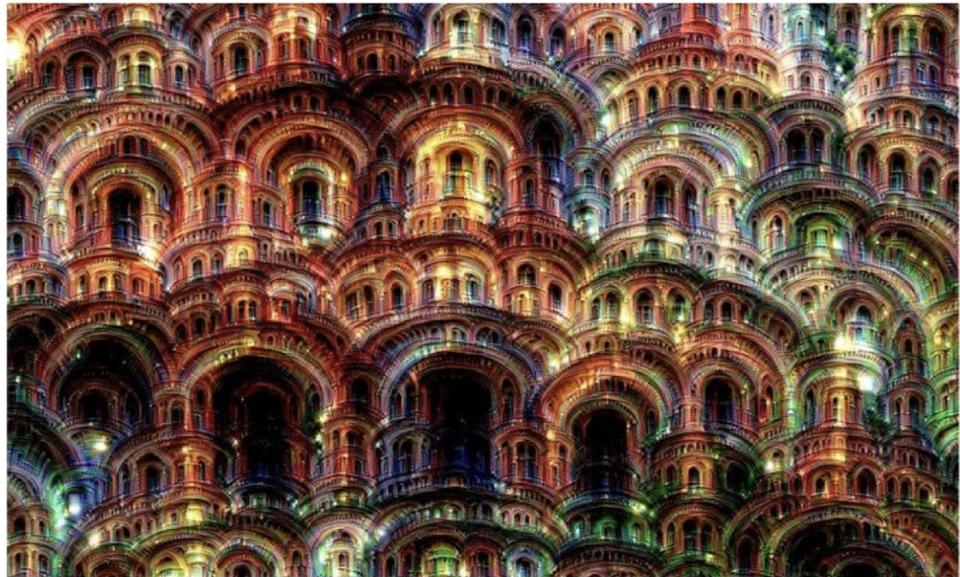
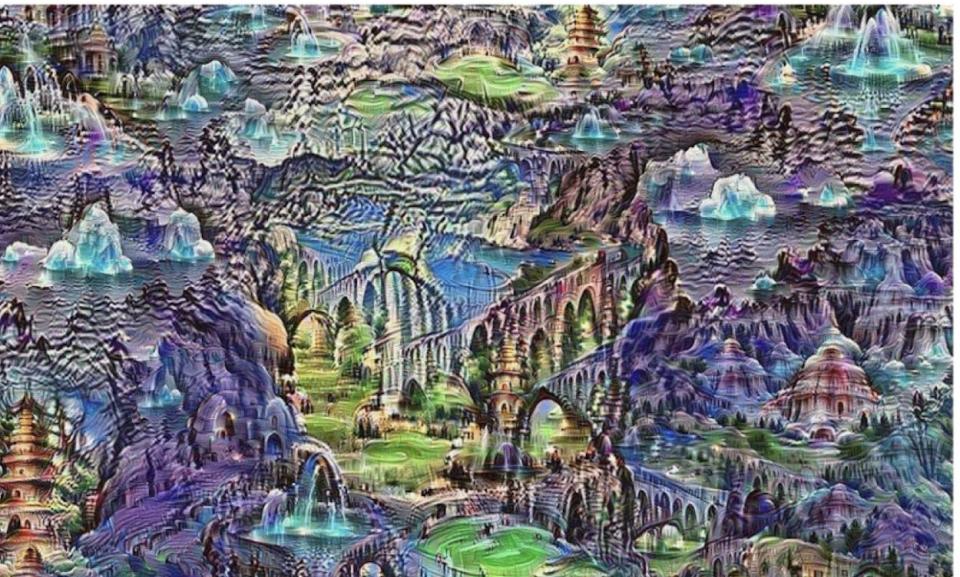


Image is licensed under CC-BY 4.0

Copyrights@CS231N, Stanford

Neural Style Transfer

Content Image



Style Image



[This image](#) is licensed under CC-BY 3.0

[Starry Night](#) by Van Gogh is in the public domain

Neural Style Transfer

Content Image



[This image](#) is licensed under CC-BY 3.0

+

Style Image



[Starry Night](#) by Van Gogh is in the public domain

=

Style Transfer!



[This image](#) copyright Justin Johnson, 2015. Reproduced with permission.

Neural Style Transfer



Summary

- CNNs are utilized in large scale production use cases
- CNNs use layers such as Conv, MaxPool, Fully Connected etc
- CNNs can be visualized to study what they are doing
- There are several architectures used today such as AlexNet, VGGNet, ResNet etc.
- Various methods such as Saliency Maps, Feature Inversion, Style Transfer help understand them

LYLT

Re-defining Loyalty with Artificial Intelligence and Personalization



TOTAL UNREDEEMED LOYALTY POINTS = **\$100B**

Questions?

Contact: rahulraju93@gmail.com

Phone: +917349497007, +16122084881

Thank you!