Introduction to Convolutional Neural Networks

Mauricio A. Álvarez PhD,
H.F. Garcia C. Guarnizo (TA)

Universidad Tecnológica de Pereira, Pereira, Colombia

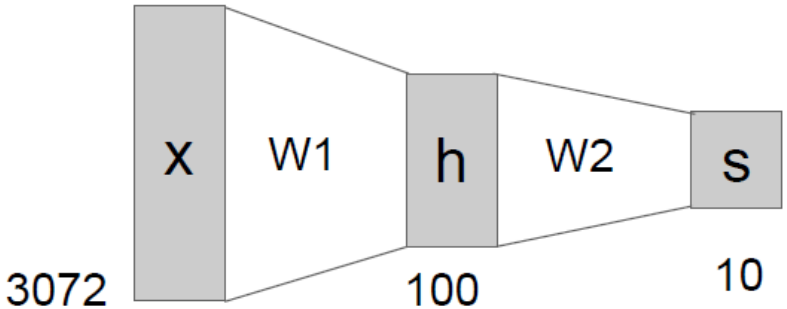
Outline

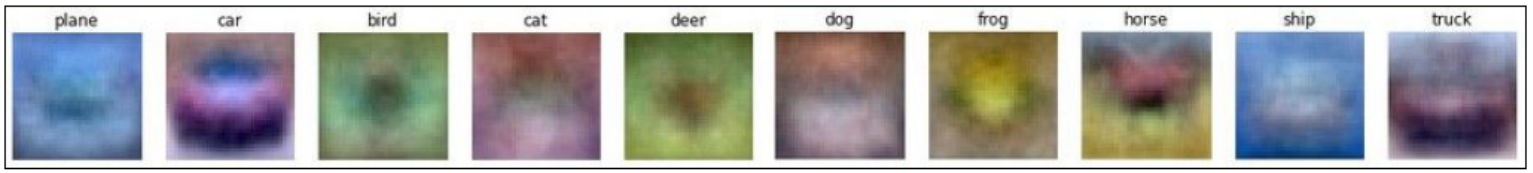
Outline

Last time: Neural Networks

Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$





Next: Convolutional Neural Networks Illustration of

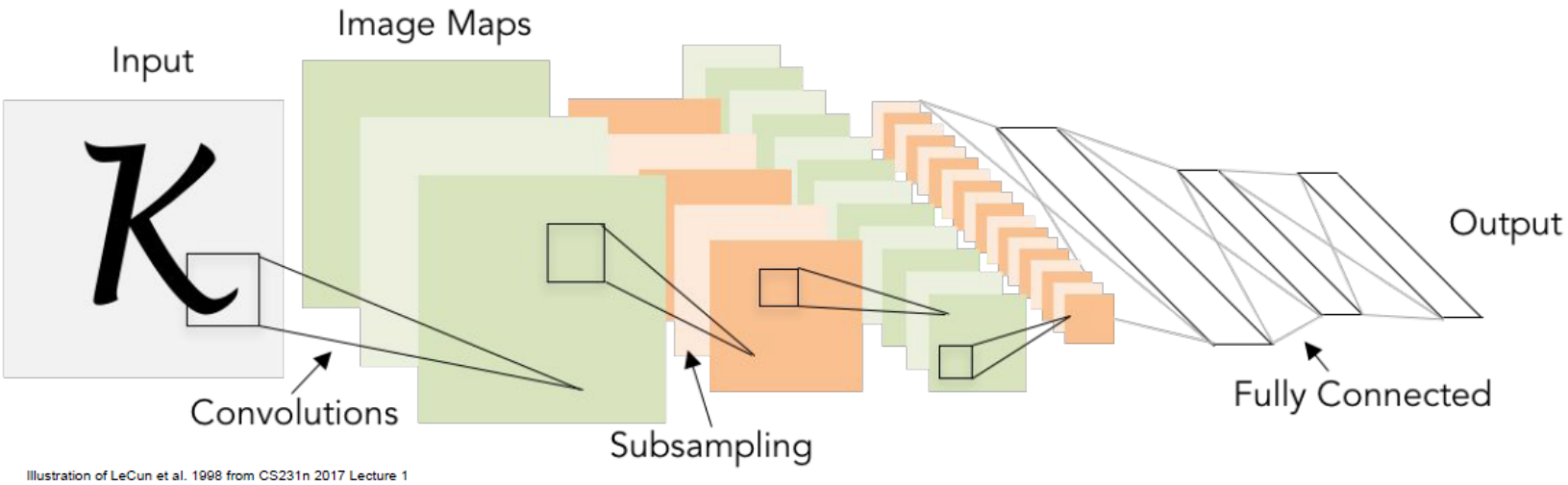


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Un poco de historia

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

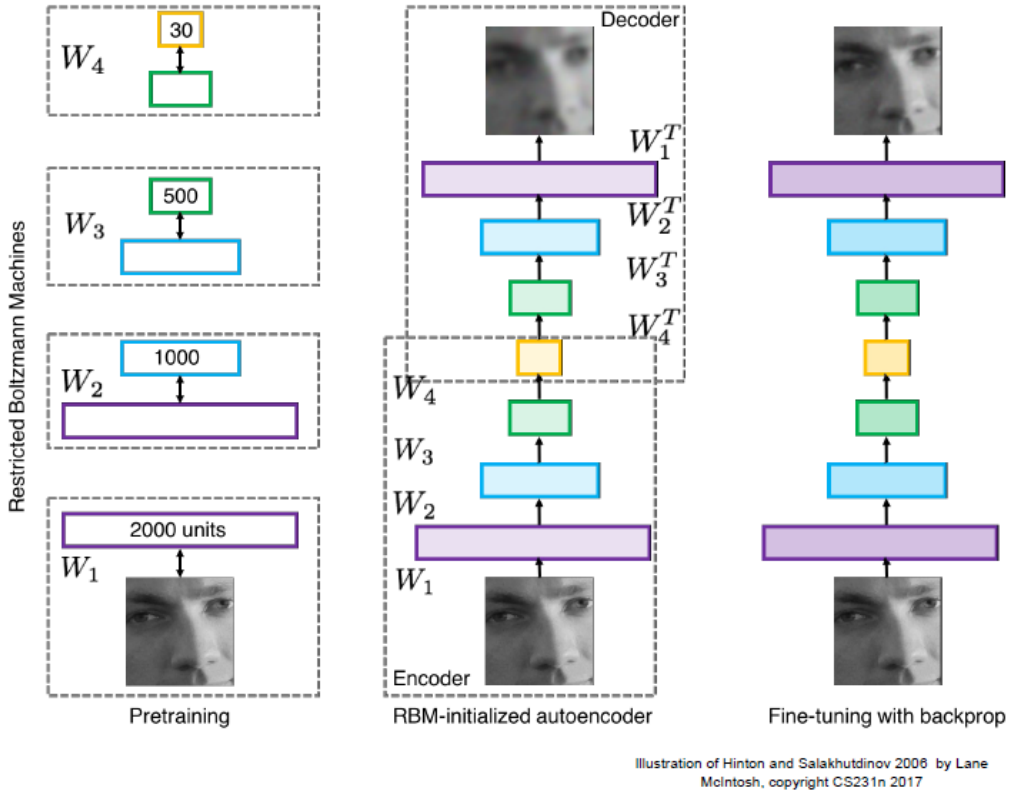


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Primeros resultados representativos I

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

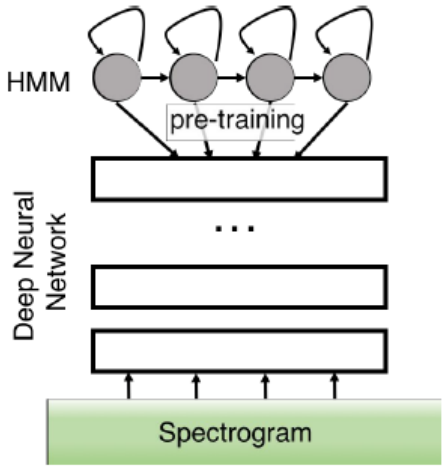
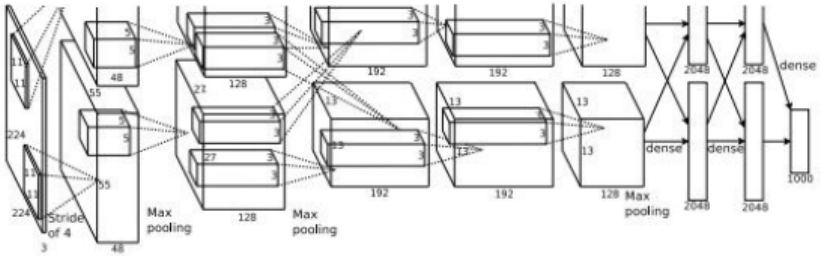
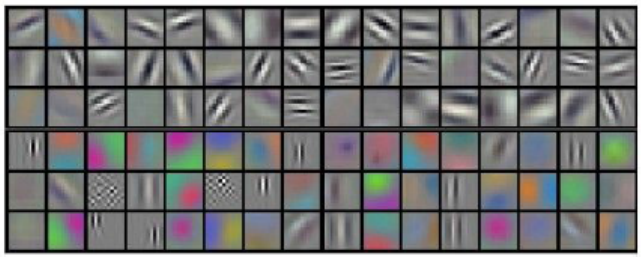


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





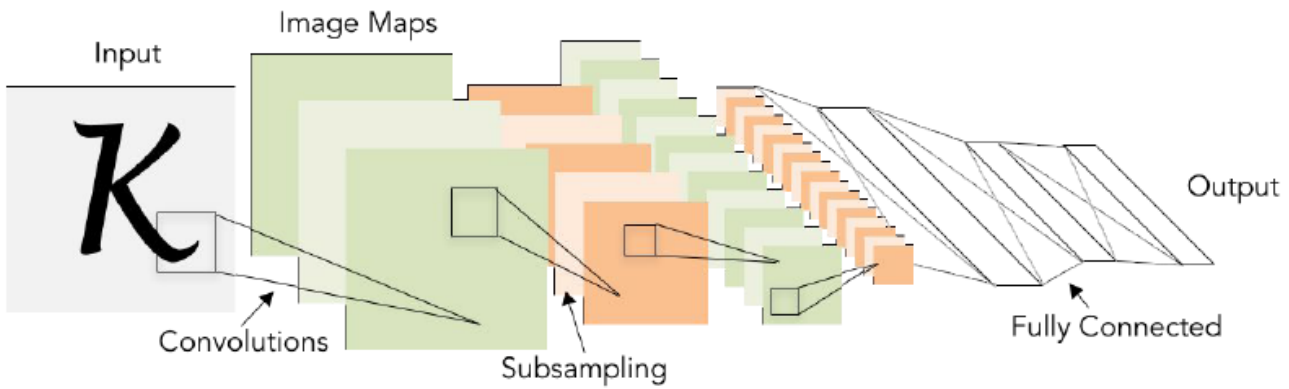
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Primeros resultados representativos II

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

Primeros resultados representativos III

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

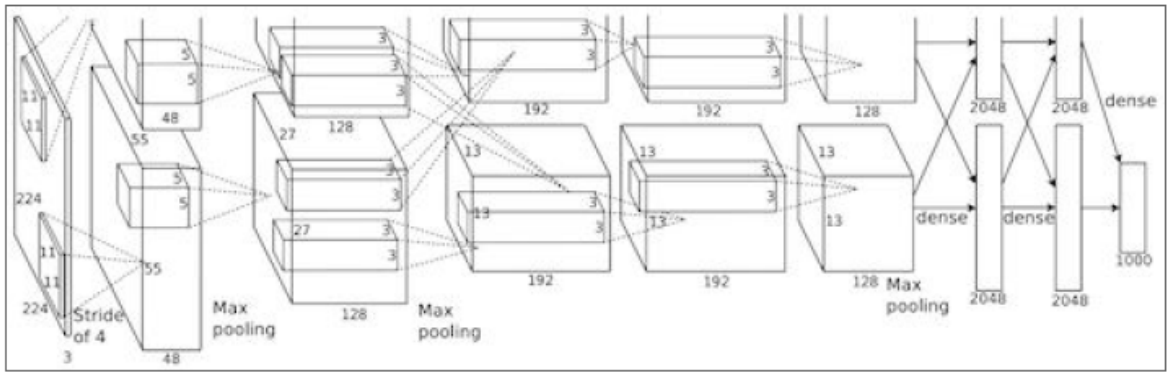
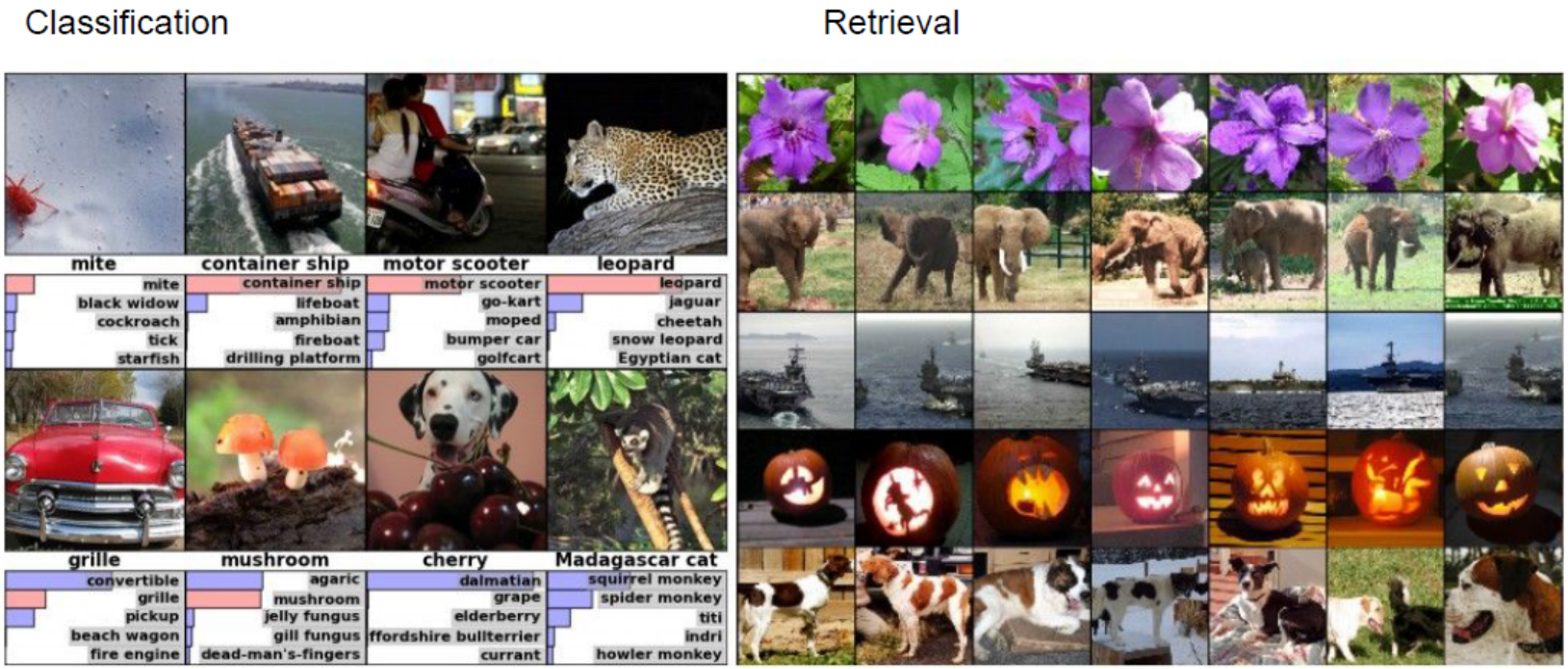


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

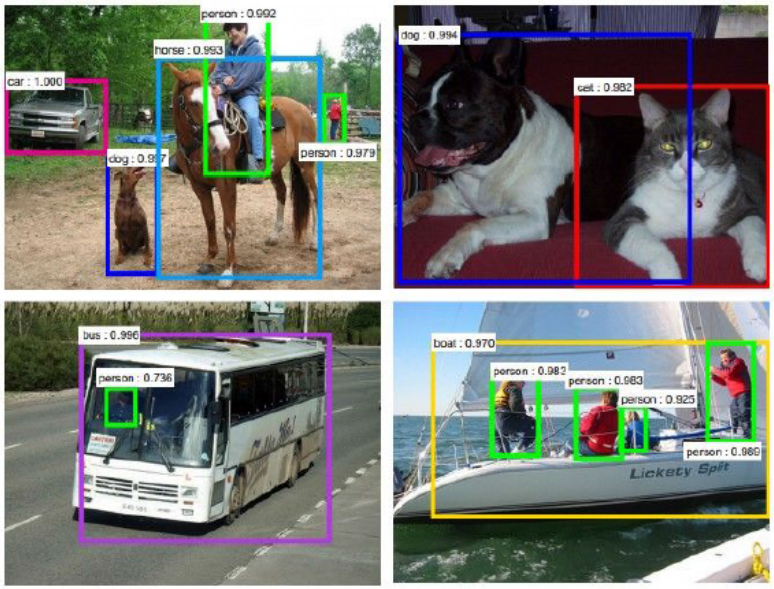
Fast-forward to today: ConvNets are everywhere!!



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

Fast-forward to today: ConvNets are everywhere!! II

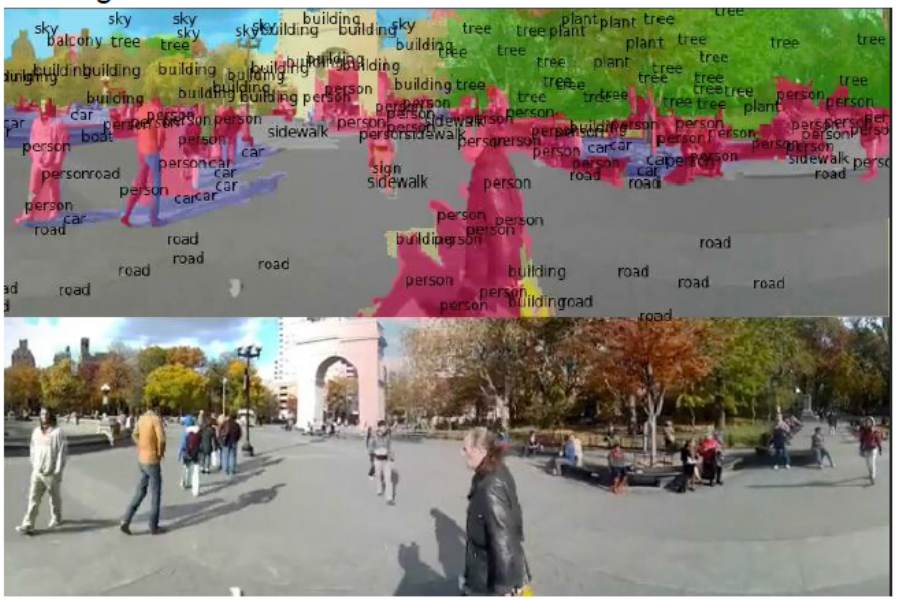
Detection



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

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

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere!! III

Photo by Lane McIntosh. Copyright CS231n 2017.

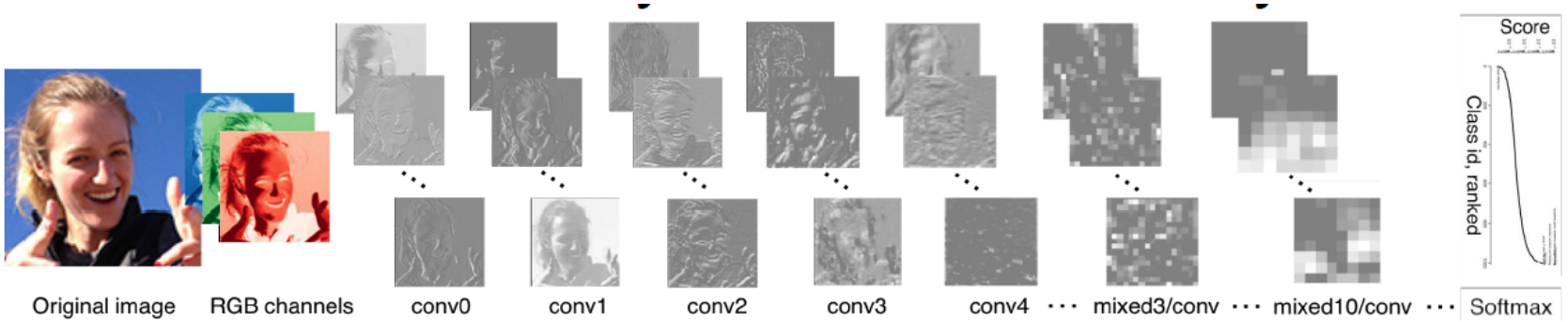
self-driving cars

NVIDIA Tesla line

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

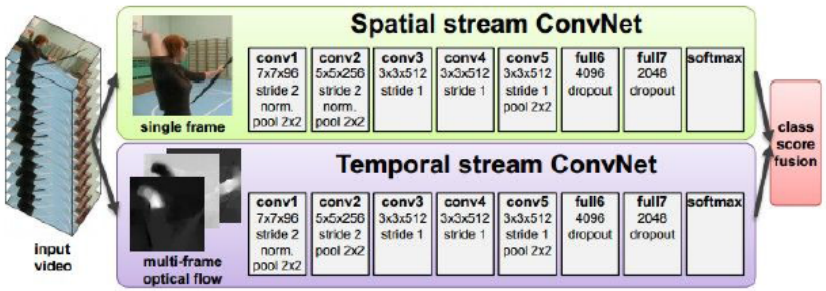
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere!! IV



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

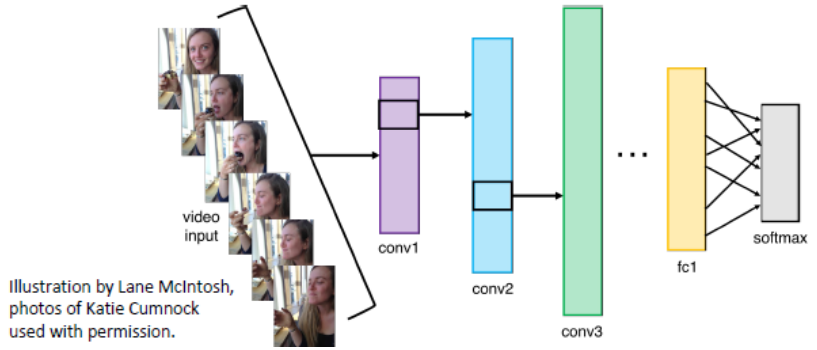
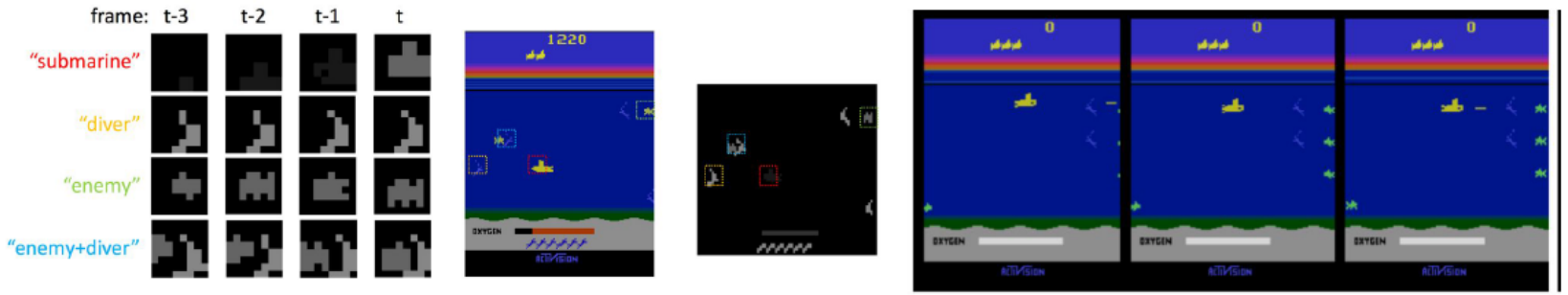


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Fast-forward to today: ConvNets are everywhere!!
v

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

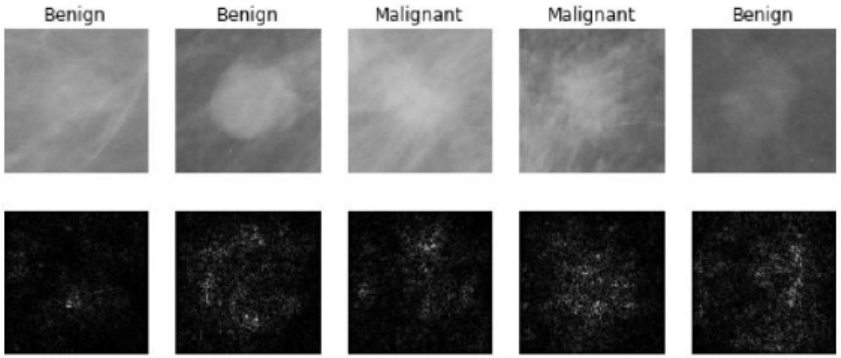
[Toshev, Szegedy 2014]



[Guo et al. 2014]

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

Fast-forward to today: ConvNets are everywhere!! VI



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Sermanet et al. 2011]
[Ciresan et al.]

Photos by Lane McIntosh.
Copyright CS231n 2017.

Fast-forward to today: ConvNets are everywhere!! VII

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

Fast-forward to today: ConvNets are everywhere!! VIII

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

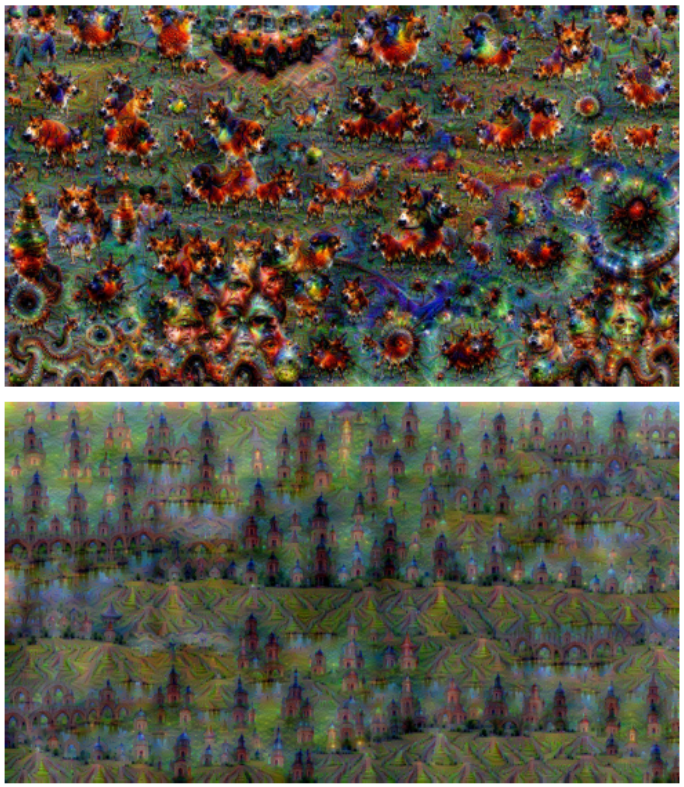
Image Captioning

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

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

Captions generated by Justin Johnson using Neuraltalk2

Fast-forward to today: ConvNets are everywhere!! IX



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

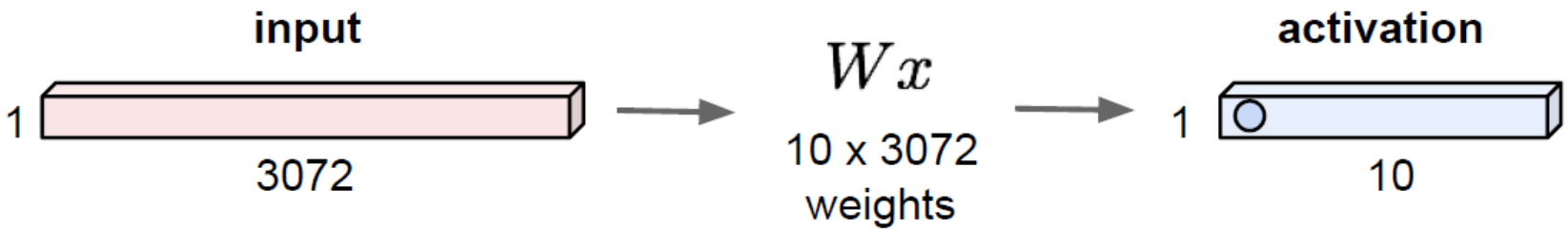
Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;

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

Outline

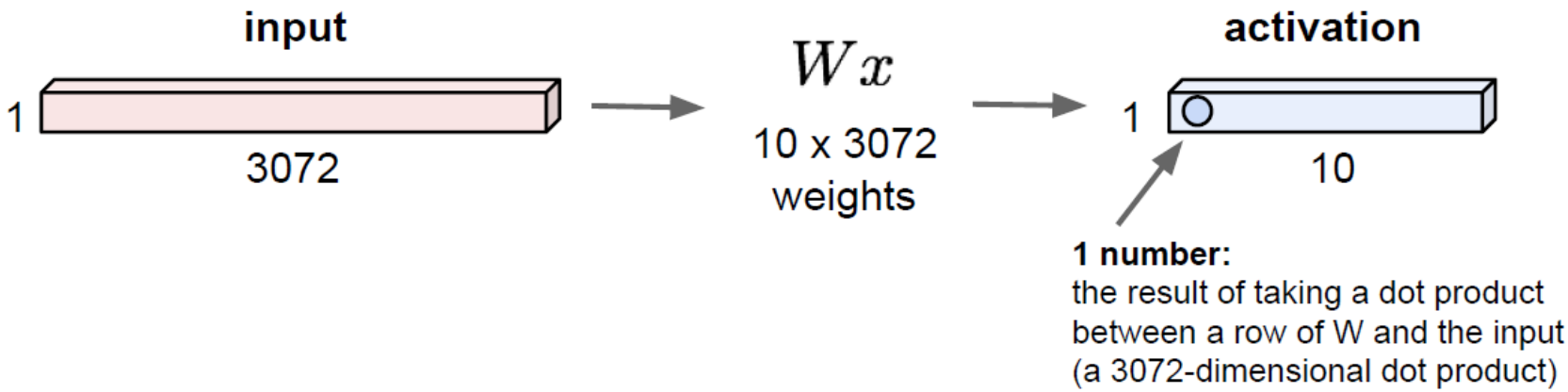
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



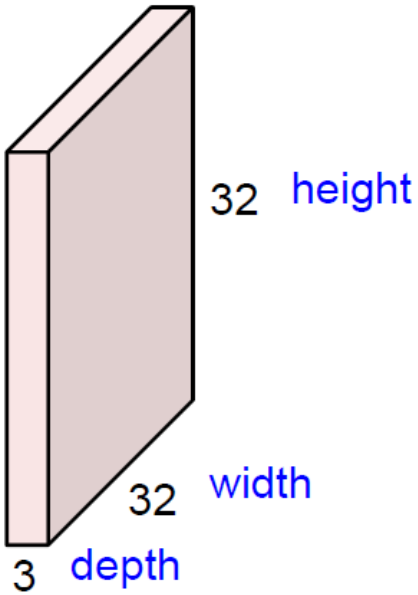
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



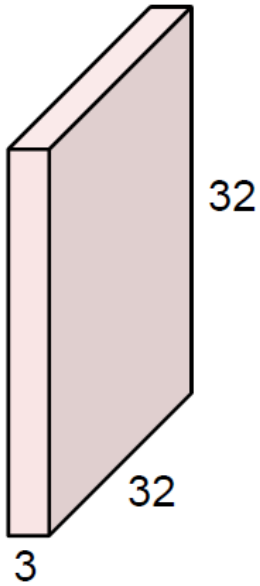
Convolution Layer

32x32x3 image -> preserve spatial structure

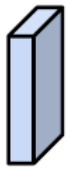


Convolution Layer

32x32x3 image



5x5x3 filter



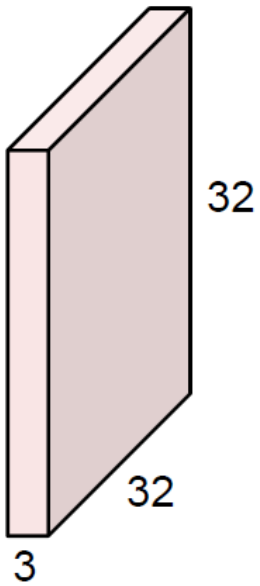
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutional Neural Networks V

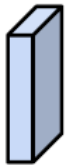
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

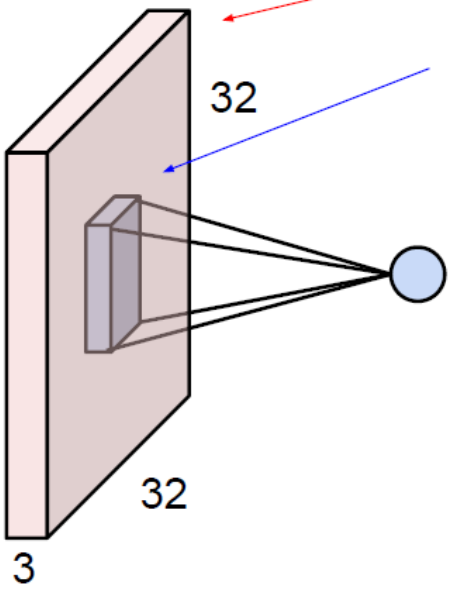


5x5x3 filter



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

Convolution Layer



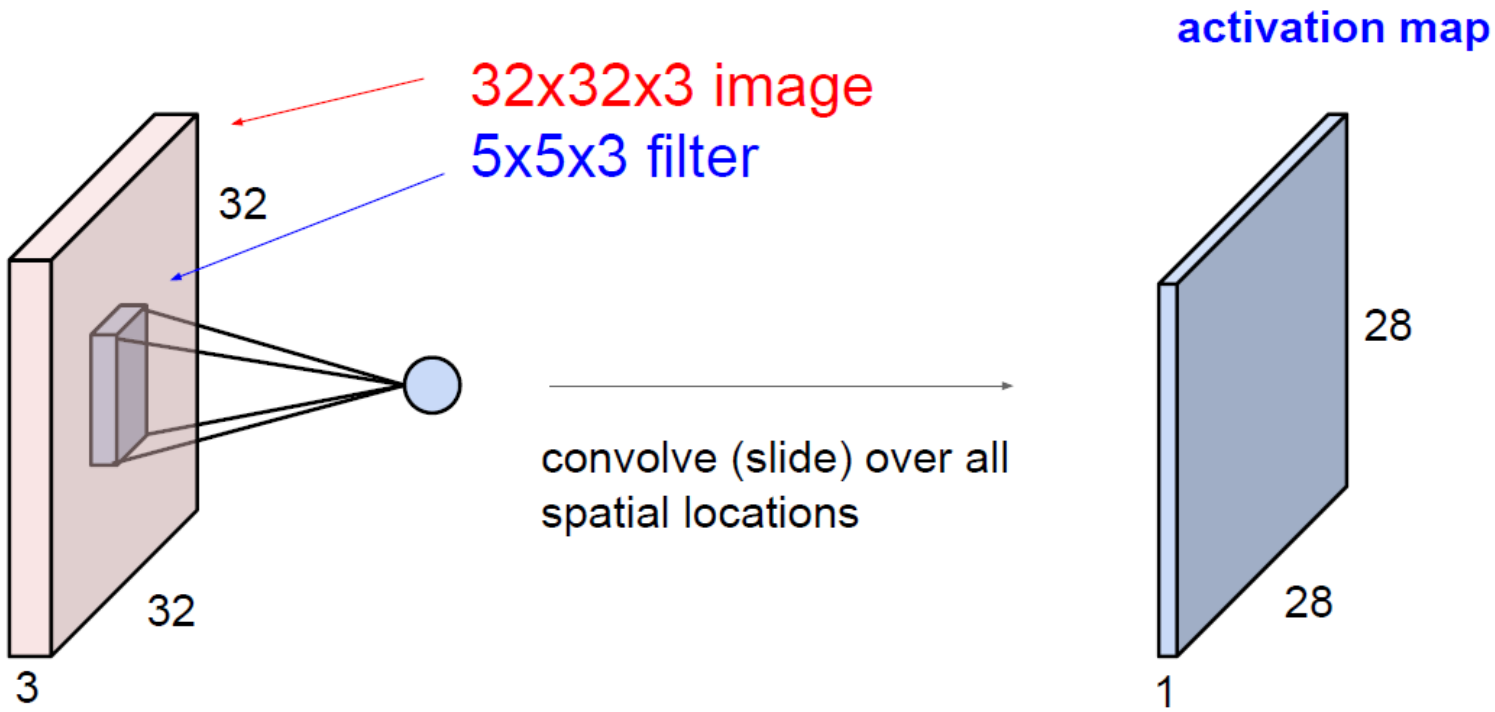
32x32x3 image

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)

$$w^T x + b$$

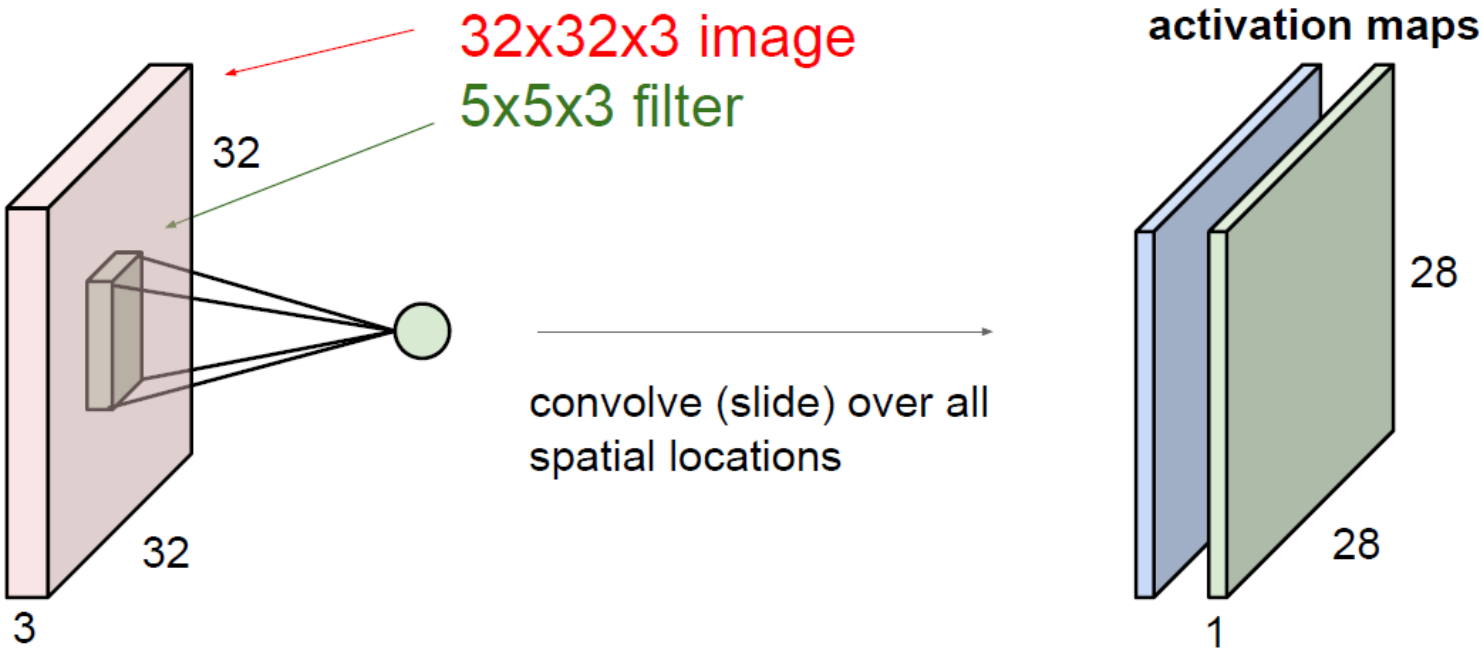
Convolution Layer



Convolutional Neural Networks VIII

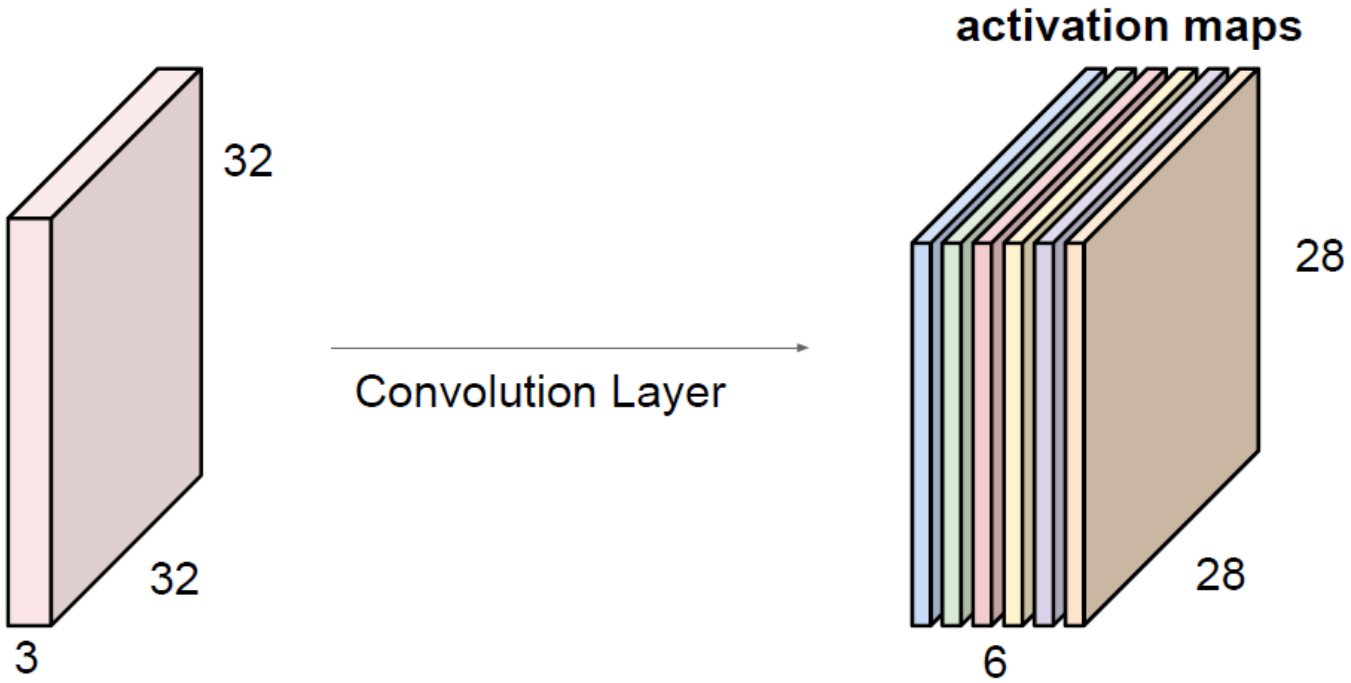
Convolution Layer

consider a second, green filter



Convolutional Neural Networks IX

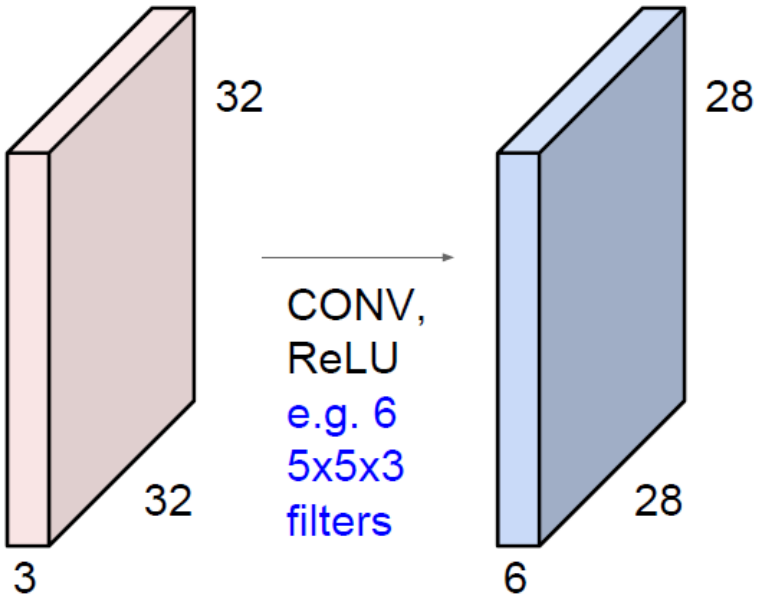
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

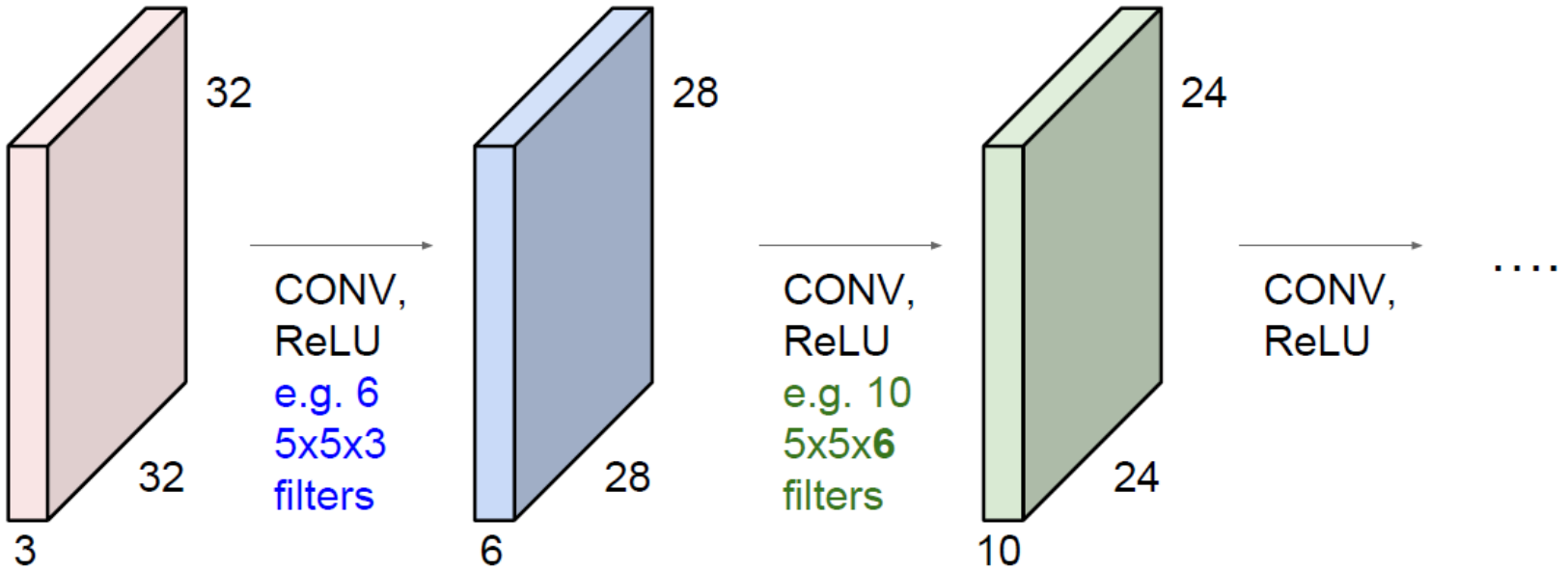
Convolutional Neural Networks X

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



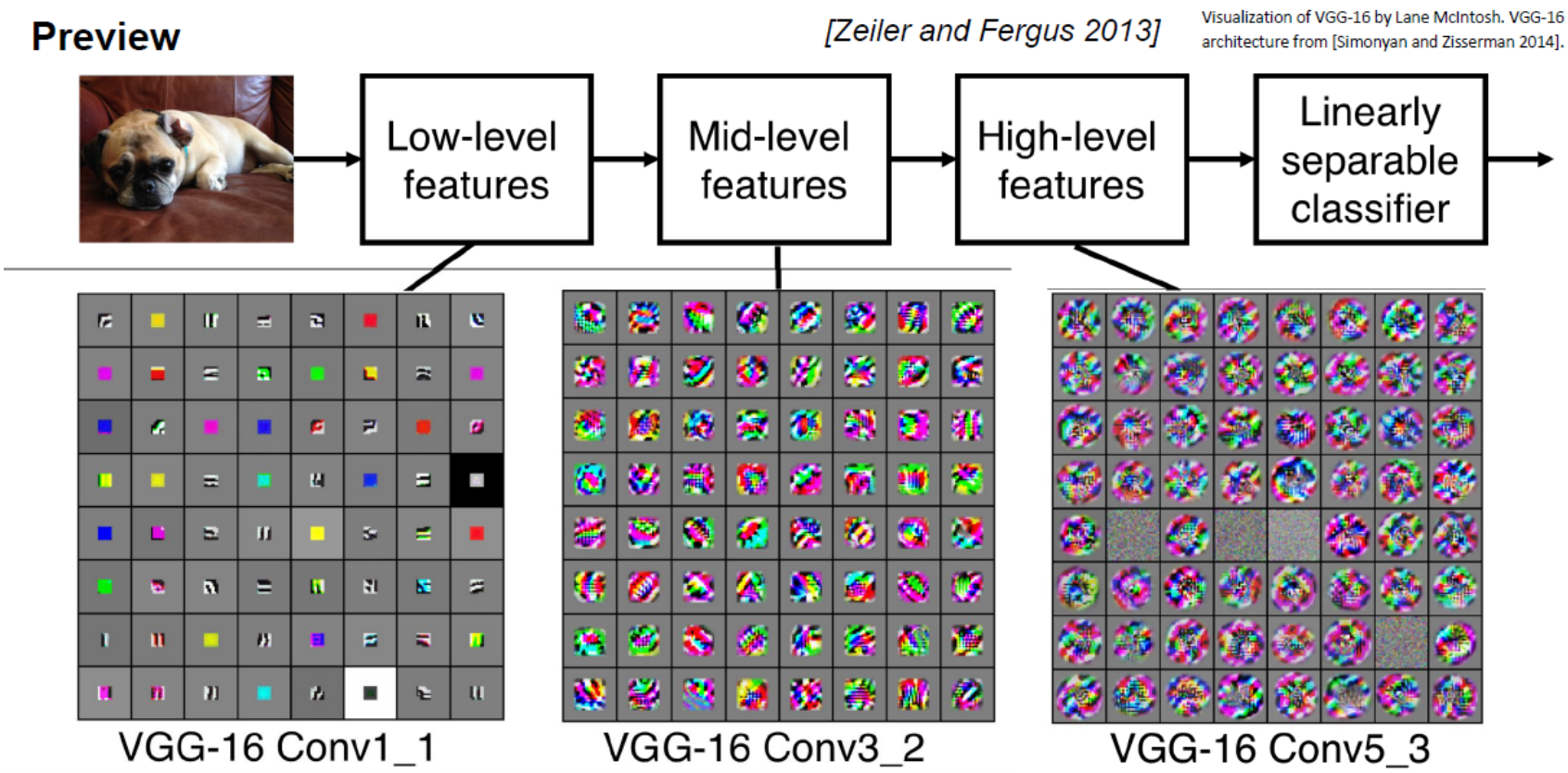
Convolutional Neural Networks XI

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



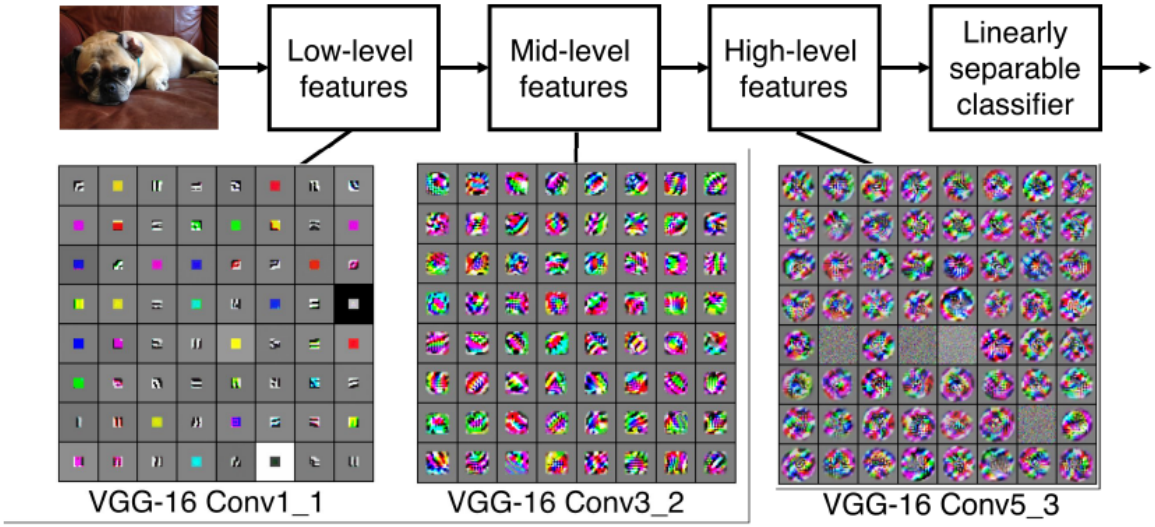
Outline

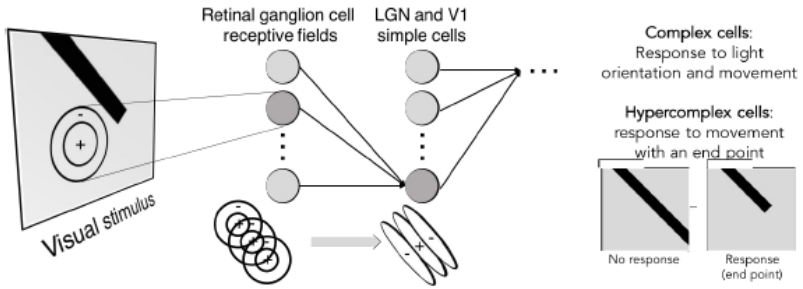
CNN preview I



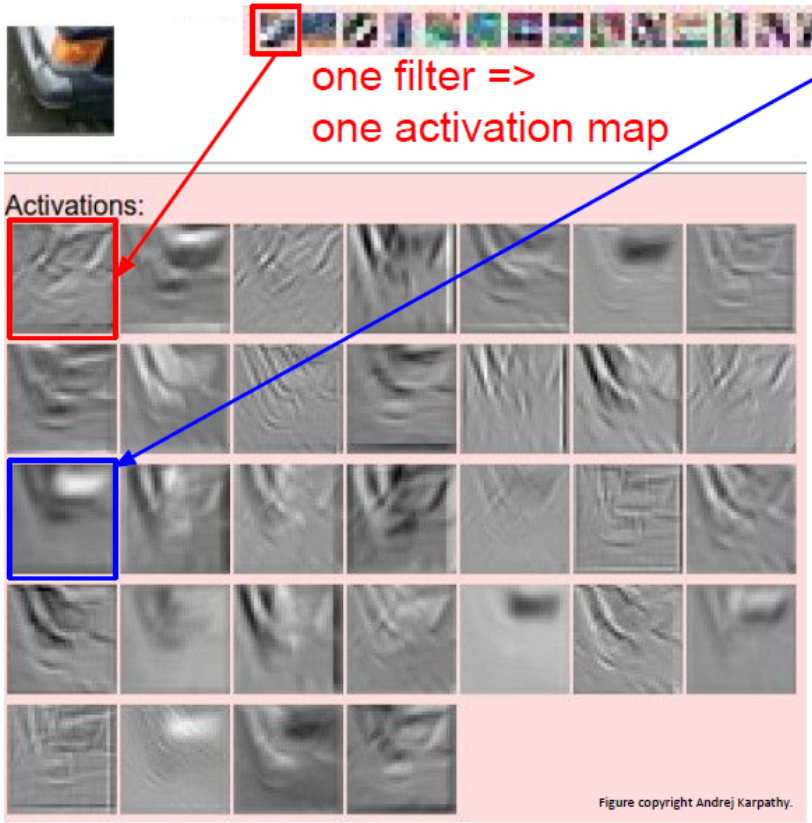
CNN preview II

Preview





CNN preview III



example 5x5 filters
(32 total)

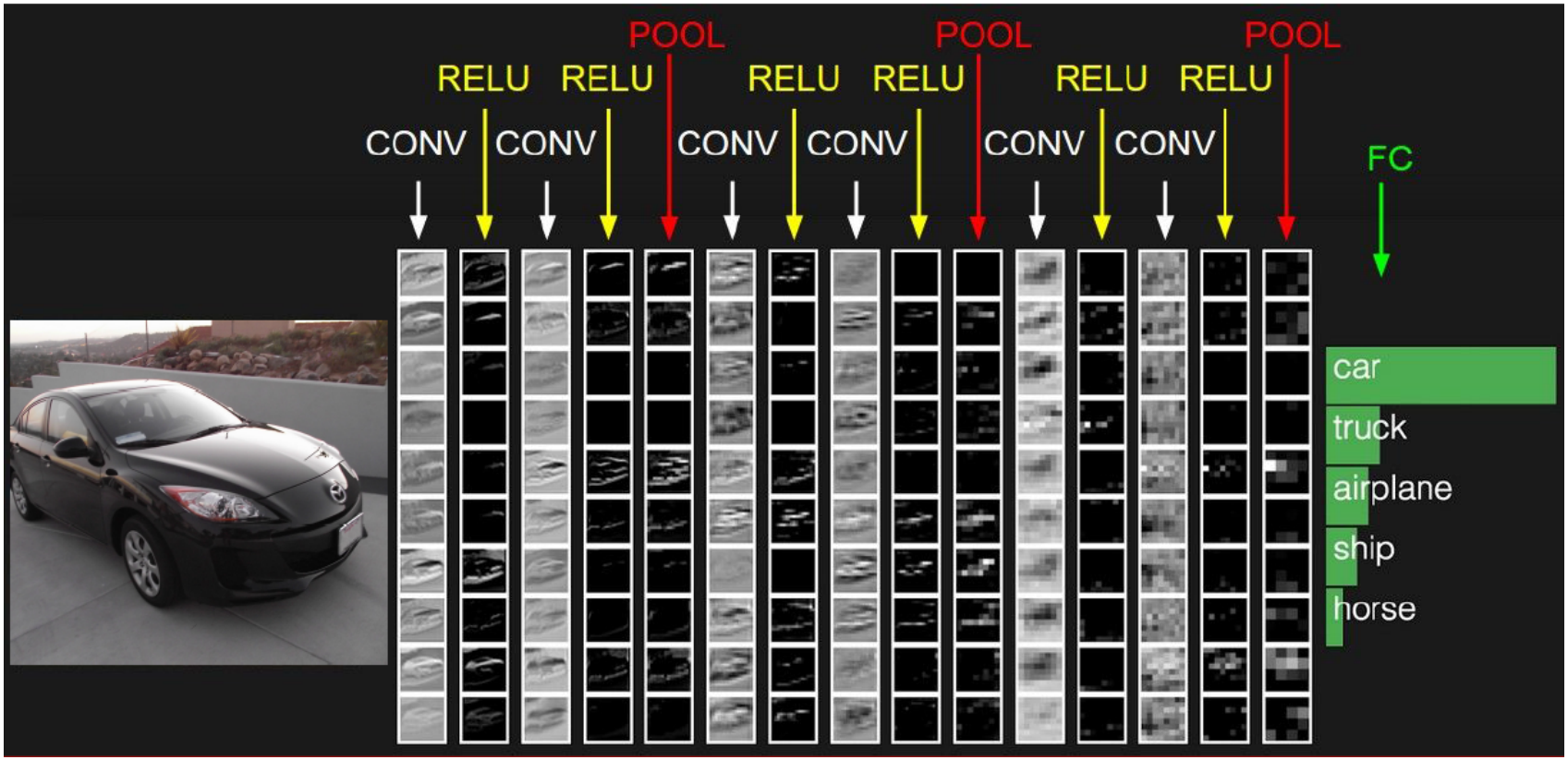
We call the layer convolutional because it is related to convolution of two signals:

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

elementwise multiplication and sum of a filter and the signal (image)

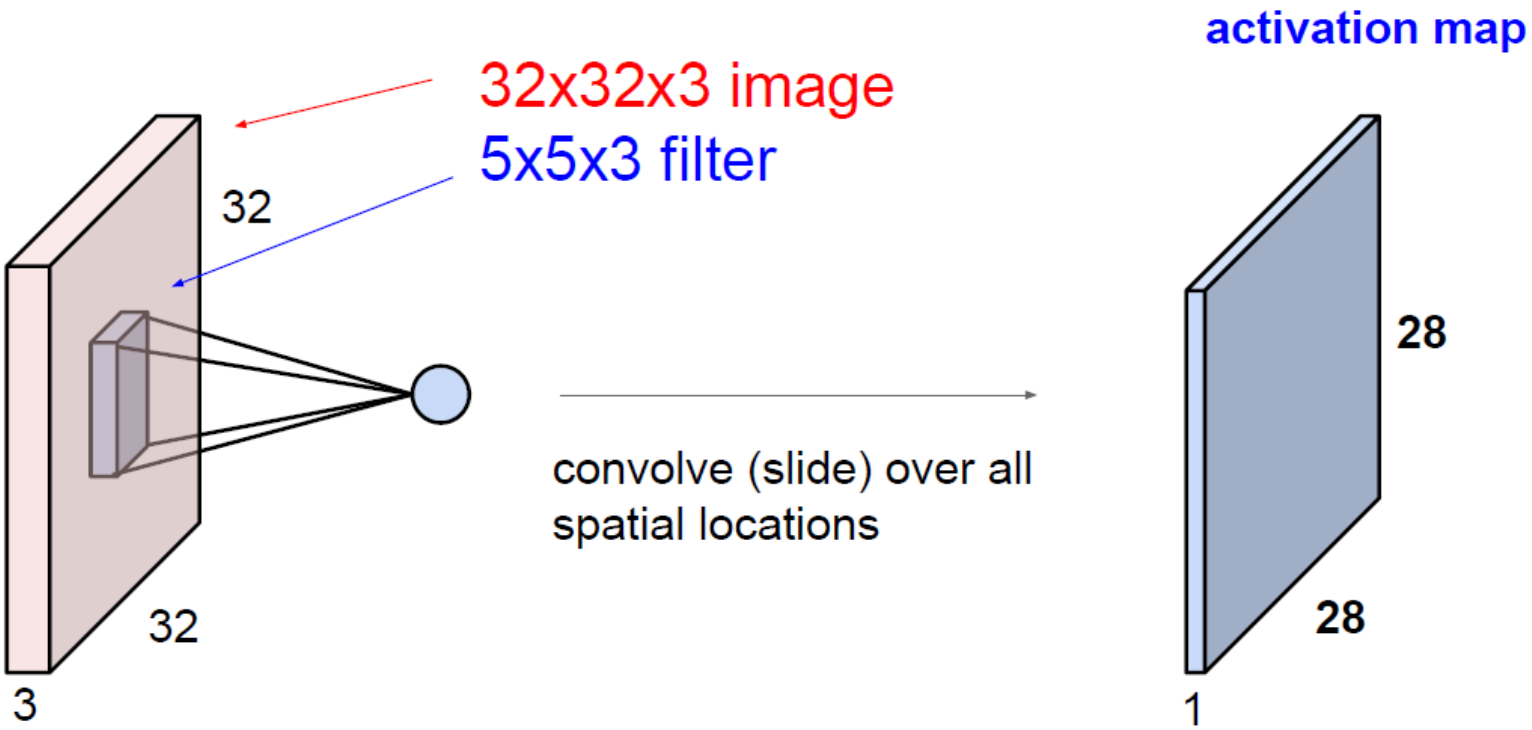
CNN preview IV

preview:



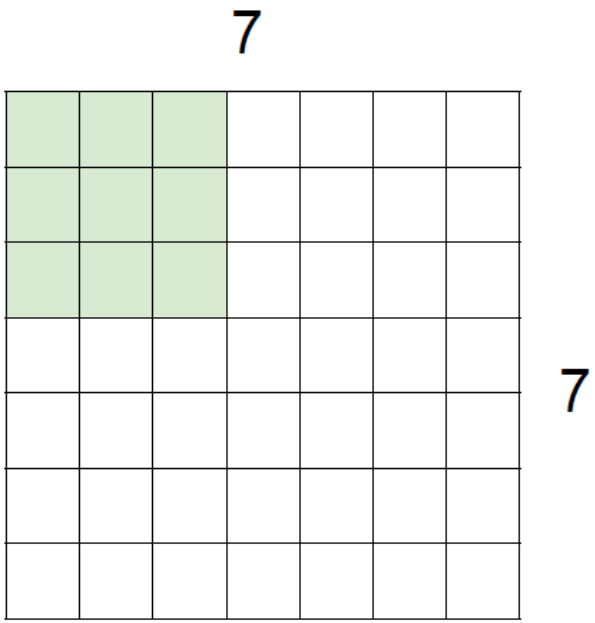
CNN (dimensiones espaciales) I

A closer look at spatial dimensions:



CNN (dimensiones espaciales) II

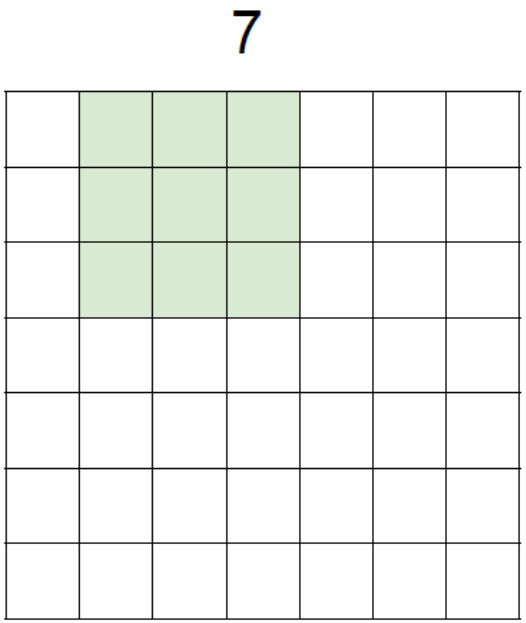
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN (dimensiones espaciales) III

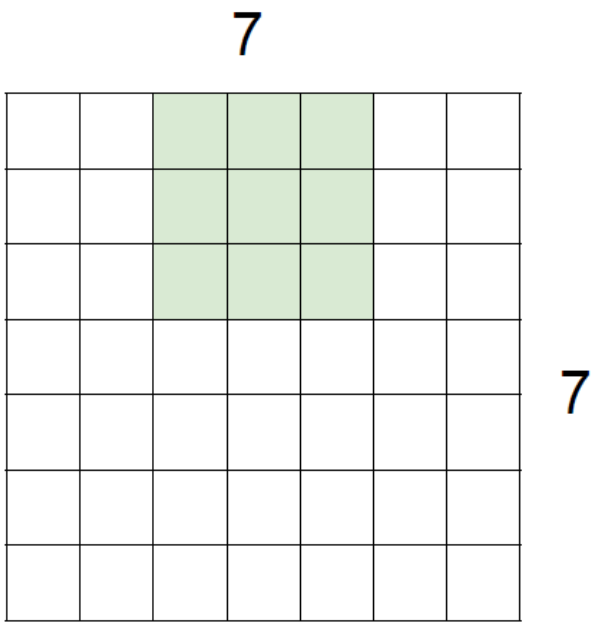
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN (dimensiones espaciales) IV

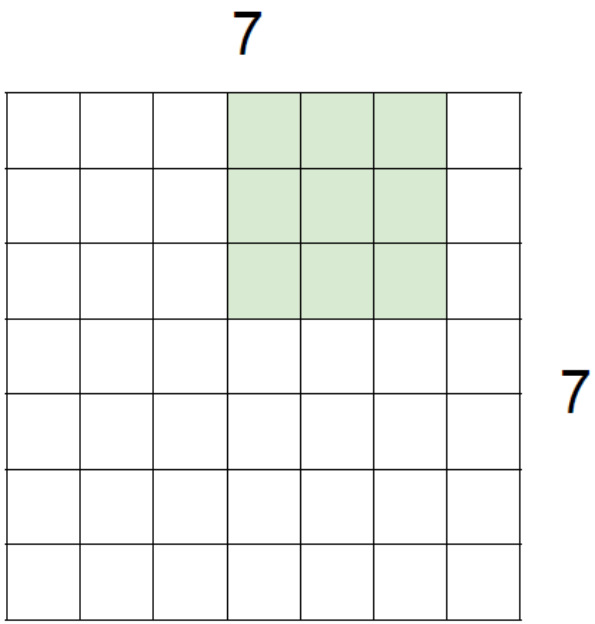
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN (dimensiones espaciales) V

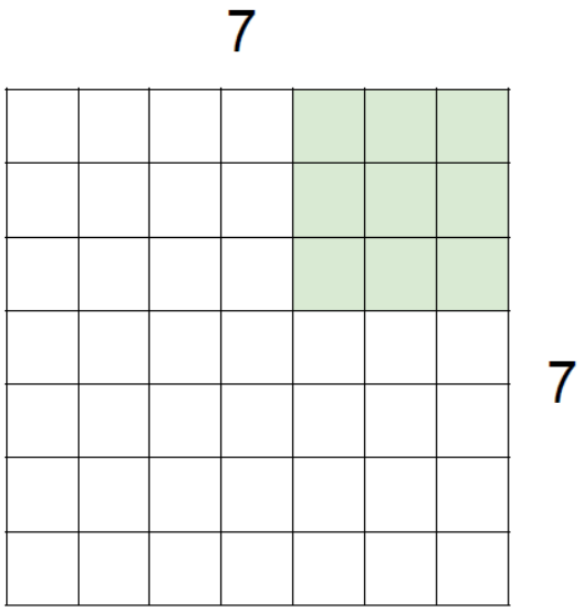
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN (dimensiones espaciales) VI

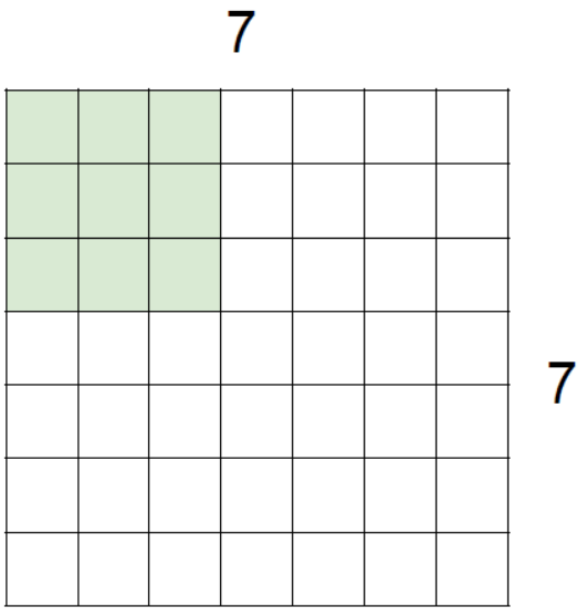
A closer look at spatial dimensions:



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

CNN (dimensiones espaciales) VII

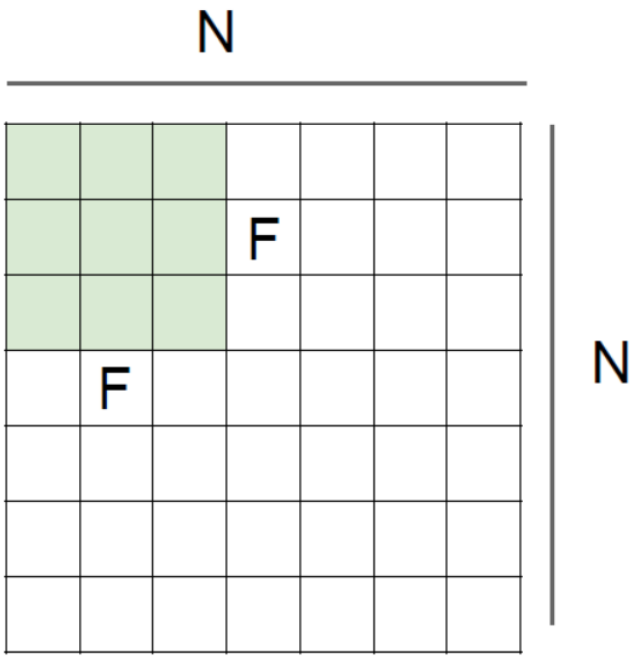
A closer look at spatial dimensions:



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

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

CNN (dimensiones espaciales) VIII

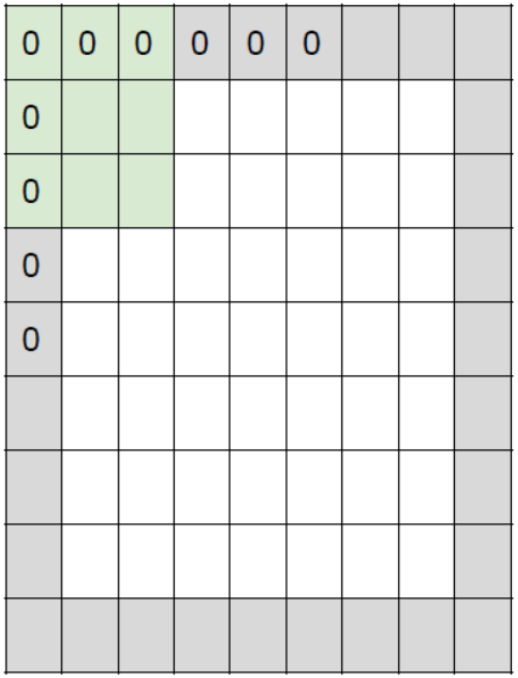


Output size:
(N - F) / stride + 1

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 :\

CNN (dimensiones espaciales) IX

In practice: Common to zero pad the border

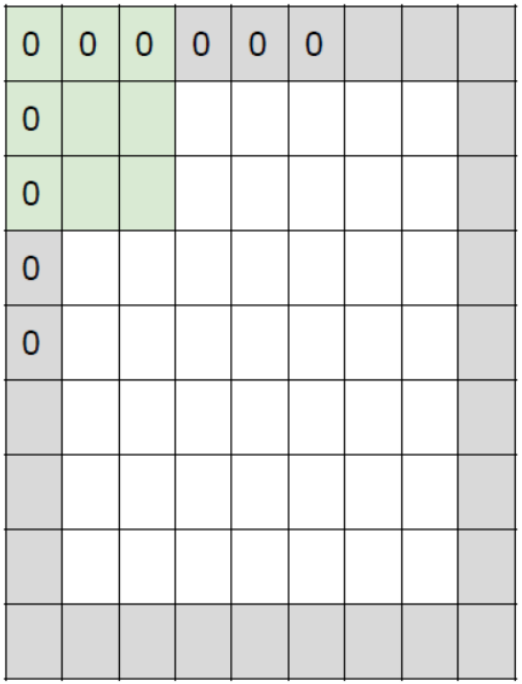


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

CNN (dimensiones espaciales) X

In practice: Common to zero pad the border

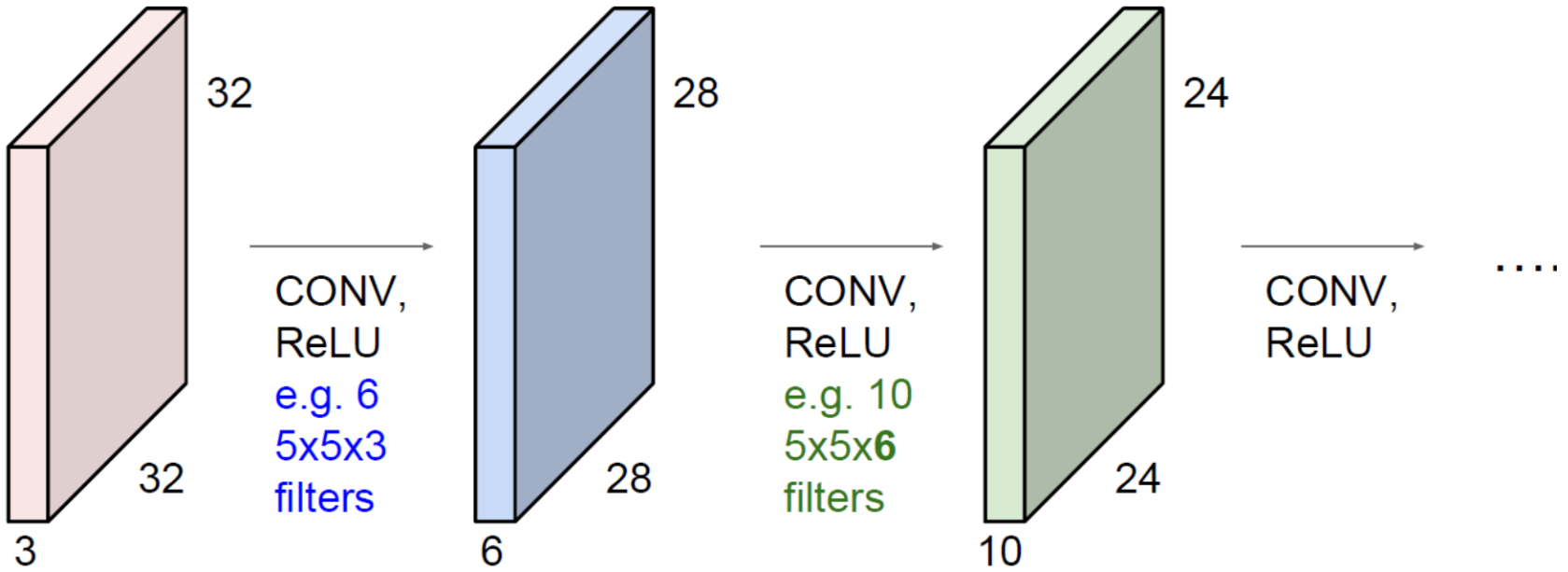


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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 . . . I

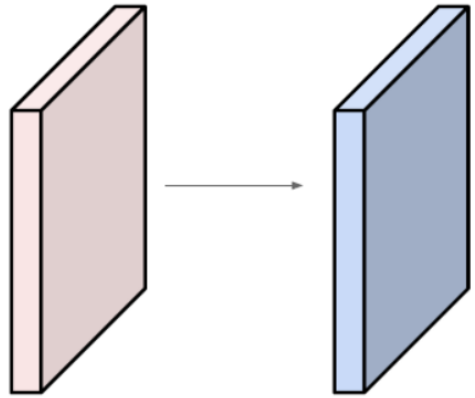
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! $(32 -> 28 -> 24 \ldots)$. Shrinking too fast is not good, doesn't work well.



Remember back to . . . II

Output volume size: $(W - F + 2P)/S + 1$

Examples time:

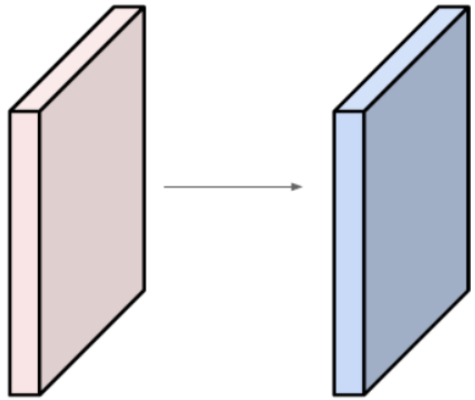


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

Remember back to . . . III

Examples time:



Input volume: **32x32x3**
10 5x5 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**

Conv layer - Summary I

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:
  - $W2 = (W1 - F + 2P)/S + 1$
  - $H2 = (H1 - F + 2P)/S + 1$ (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.

Conv layer - Summary II

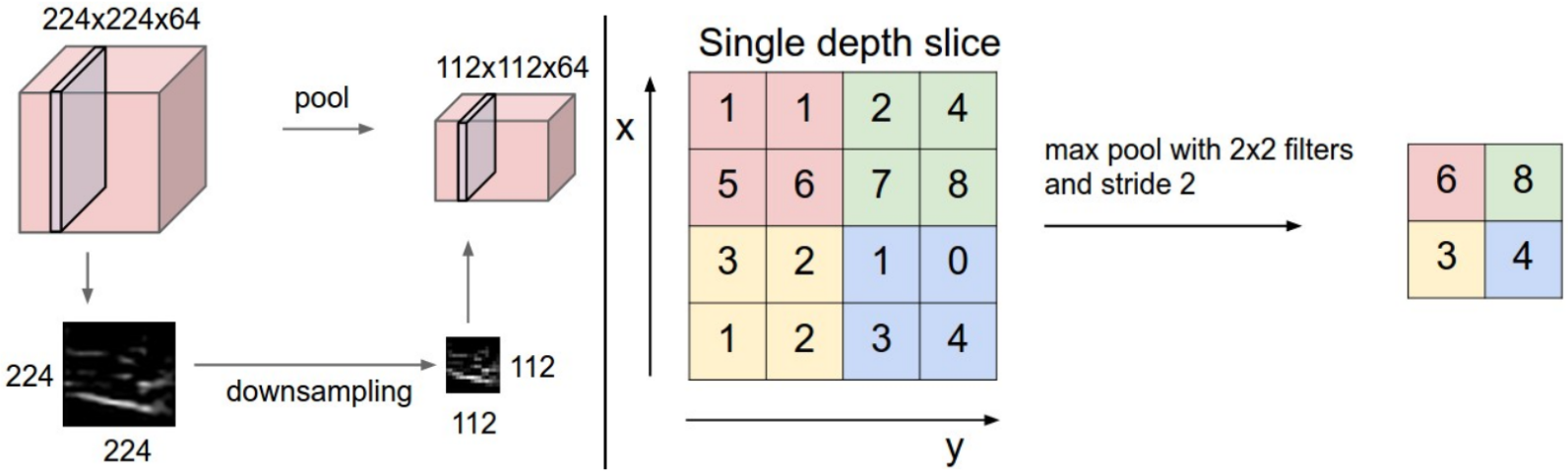
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

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

Pooling layer I

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



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,

Pooling layer II

- the stride S,

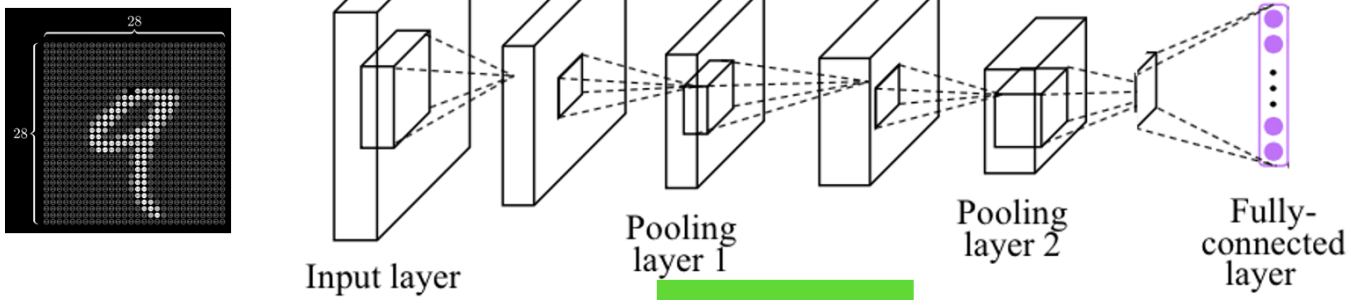
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

- Introduces zero parameters since it computes a fixed function of the input

- For Pooling layers, it is not common to pad the input using zero-padding.

Example: CNN layer in Torch I

Simple CNN




```
self.fc = nn.Linear(320, 10)
```

Linear

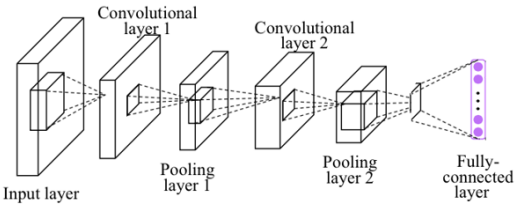
torch.nn.Conv2d (in_channels, out_channels, kernel_siz)

nn.MaxPool2d (kernel_size)

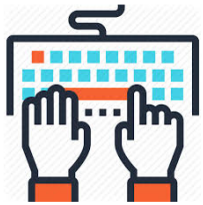
```
self.mp = nn.MaxPool2d(2)
```

```
self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
```

Example: CNN layer in Torch II



Simple CNN



```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10) # 320 -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

```
Train Epoch: 9 [46080/60000 (77%)]    Loss: 0.108415
Train Epoch: 9 [46720/60000 (78%)]    Loss: 0.140700
Train Epoch: 9 [47360/60000 (79%)]    Loss: 0.090830
Train Epoch: 9 [48000/60000 (80%)]    Loss: 0.031640
Train Epoch: 9 [48640/60000 (81%)]    Loss: 0.014934
Train Epoch: 9 [49280/60000 (82%)]    Loss: 0.090210
Train Epoch: 9 [49920/60000 (83%)]    Loss: 0.074975
Train Epoch: 9 [50560/60000 (84%)]    Loss: 0.058671
Train Epoch: 9 [51200/60000 (85%)]    Loss: 0.023464
Train Epoch: 9 [51840/60000 (86%)]    Loss: 0.018025
Train Epoch: 9 [52480/60000 (87%)]    Loss: 0.098865
Train Epoch: 9 [53120/60000 (88%)]    Loss: 0.013985
Train Epoch: 9 [53760/60000 (90%)]    Loss: 0.070476
Train Epoch: 9 [54400/60000 (91%)]    Loss: 0.065411
Train Epoch: 9 [55040/60000 (92%)]    Loss: 0.028783
Train Epoch: 9 [55680/60000 (93%)]    Loss: 0.008333
Train Epoch: 9 [56320/60000 (94%)]    Loss: 0.020412
Train Epoch: 9 [56960/60000 (95%)]    Loss: 0.036749
Train Epoch: 9 [57600/60000 (96%)]    Loss: 0.163087
Train Epoch: 9 [58240/60000 (97%)]    Loss: 0.117539
Train Epoch: 9 [58880/60000 (98%)]    Loss: 0.032256
Train Epoch: 9 [59520/60000 (99%)]    Loss: 0.026360

Test set: Average loss: 0.0483, Accuracy: 9846/10000 (98%)
```

ConvNetJS demo: training on CIFAR-10

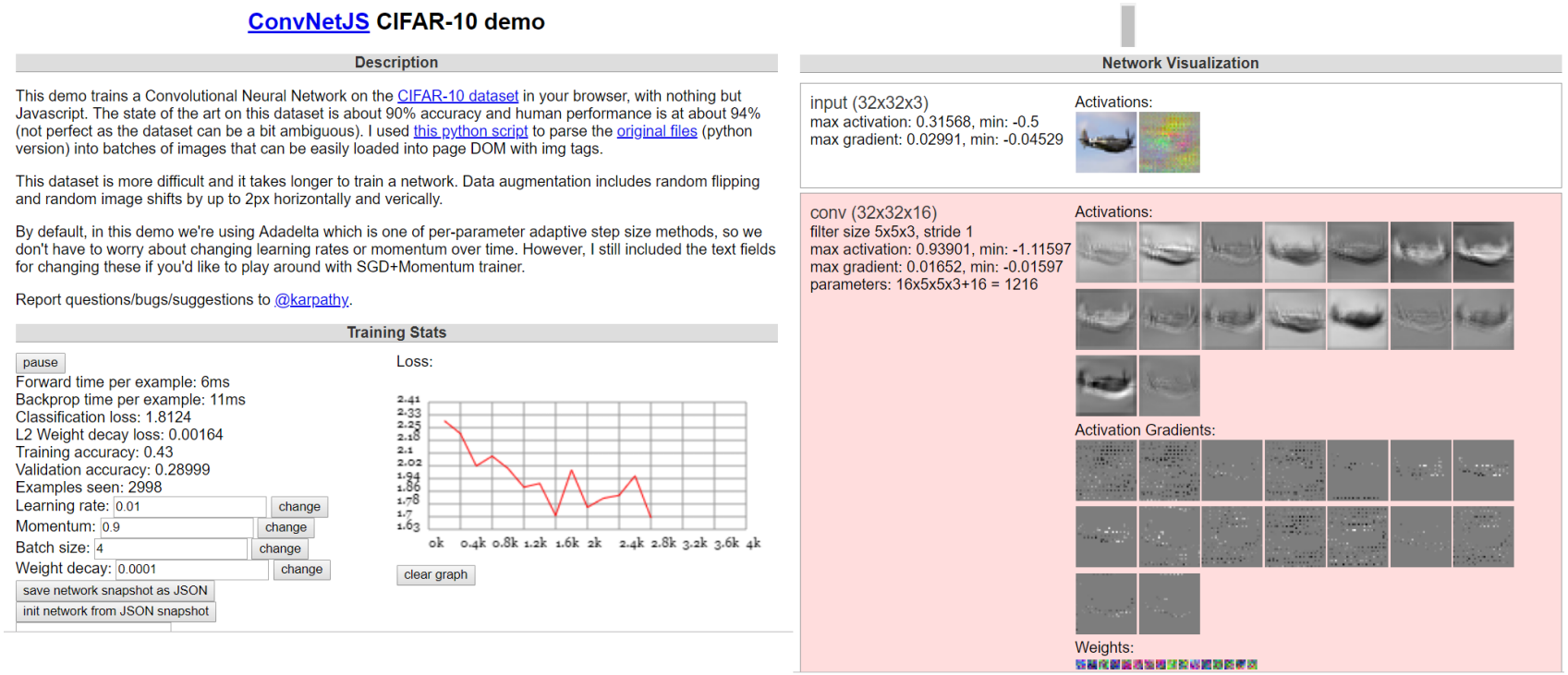
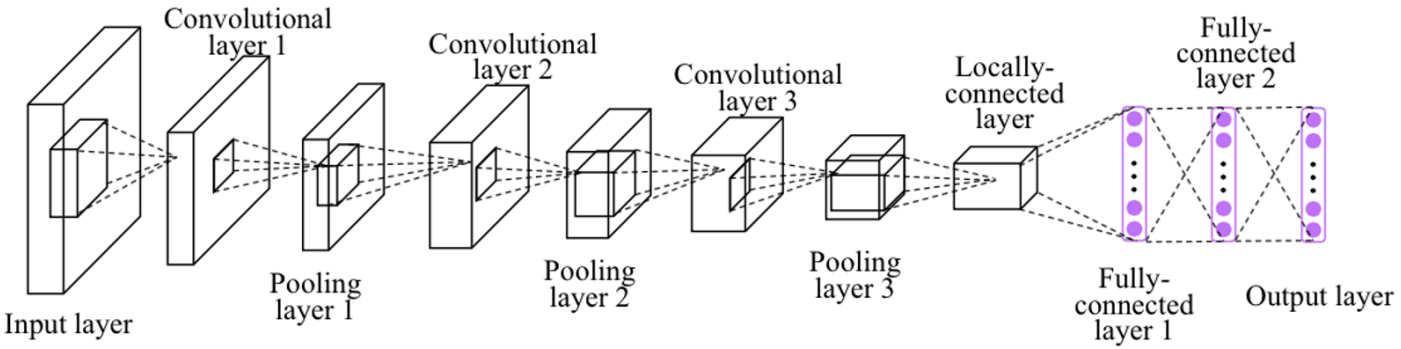


Figure: `https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`

Exercise: Implement CNN more layers



Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like:
  $[(CONV - RELU) * N - POOL?] * M - (FC - RELU) * K, \quad SOFTMAX$
  where N is usually up to ~ 5, M is large, $0 \leq K \leq 2$.
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm

References

1. Fei-Fei Li, Convolutional Neural Networks for Visual Recognition.