

Lecture 5: Image Classification with CNNs

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 1

April 16, 2024

Image Classification: A core task in Computer Vision



(assume given a set of labels)
{dog, cat, truck, plane, ...}

cat
dog
bird
deer
truck

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 15

April 16, 2024

Pixel space



$$f(x) = Wx$$

Class scores



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 16

April 16, 2024

Image features



$$f(x) = Wx$$

Feature Representation

Class scores

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 17

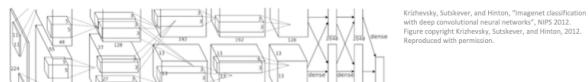
April 16, 2024

Image features vs. ConvNets



f
training

10 numbers giving scores for classes



10 numbers giving scores for classes
training

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 22

April 16, 2024

Last Time: Neural Networks

Linear score function:

2-layer Neural Network

$$f = Wx$$

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

The spatial structure of images is destroyed!

32x32x3 \rightarrow 3072 \rightarrow 100 \rightarrow 10

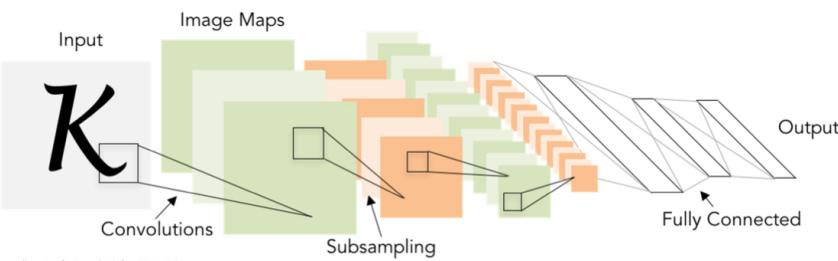


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 23

April 16, 2024

Next: Convolutional Neural Networks



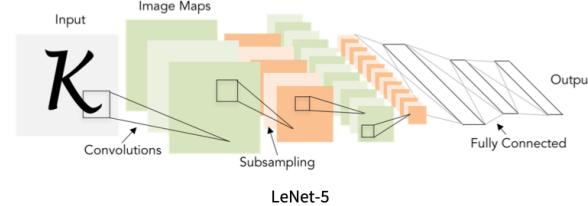
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 24

April 16, 2024

A bit of history:

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 34

April 16, 2024

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]

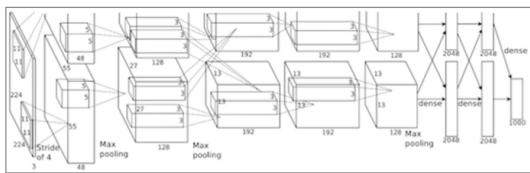


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

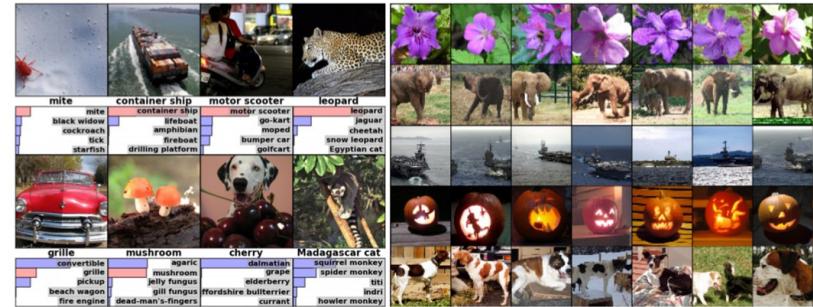
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 35

April 16, 2024

Fast-forward: ConvNets are everywhere

Classification



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

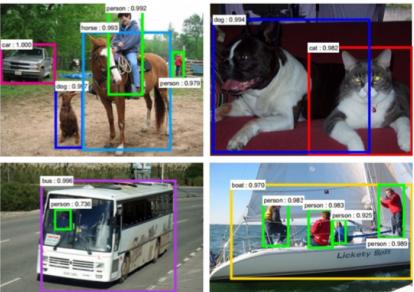
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 36

April 16, 2024

Fast-forward: ConvNets are everywhere

Detection



Figures copyright Shaqio Ren, Kaiming He, Ross Girshick, 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]

No errors



A white teddy bear sitting in the grass

Minor errors



A man in a baseball uniform throwing a ball

Somewhat related



A woman is holding a cat in her hand



A man riding a wave on top of a surfboard



A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

Image Captioning

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

All images are CC0 Public Domain:
<https://creativecommons.org/publicdomain/zero/1.0/>
https://unsplash.com/photos/0e0d9cbe9a70_bear-1643010/
https://unsplash.com/photos/1eef4a5a5a5a_summer-sport-beach-1667114/
https://unsplash.com/photos/1f0a0a0a0a_cat-1669671/
https://unsplash.com/photos/1f0a0a0a0a_lake-meditation-yoga-1669672/
https://unsplash.com/photos/1f0a0a0a0a_surfing-beach-1669673/

Captions generated by Justin Johnson using NeuralTalk2

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 37

April 16, 2024

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 43

April 16, 2024

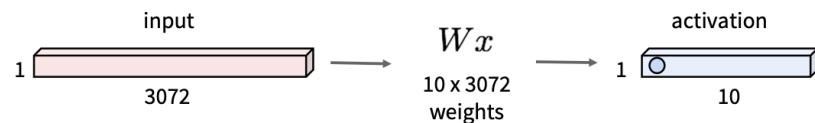
Convolutional Neural Networks

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 45 April 16, 2024

Recap: Fully Connected Layer

32x32x3 image \rightarrow stretch to 3072 x 1

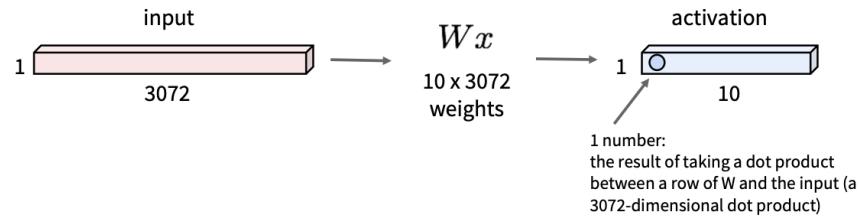


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 46 April 16, 2024

Fully Connected Layer

32x32x3 image \rightarrow stretch to 3072 x 1

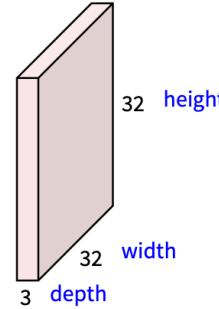


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 47 April 16, 2024

Convolution Layer

32x32x3 image \rightarrow preserve spatial structure

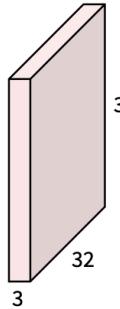


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 48 April 16, 2024

Convolution Layer

32x32x3 image



5x5x3 filter



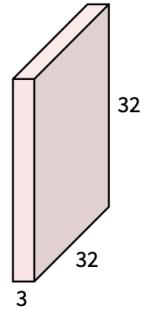
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 49 April 16, 2024

Convolution Layer

32x32x3 image



5x5x3 filter



Filters always extend the full
depth of the input volume

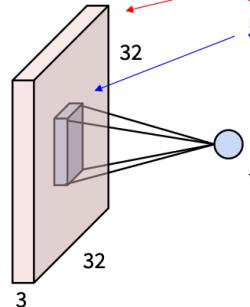
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 50 April 16, 2024

Convolution Layer

32x32x3 image
5x5x3 filter w

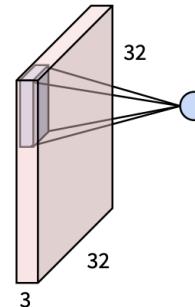


1 number:
the result of taking a dot product between the
filter and a small 5x5x3 chunk of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)
 $w^T x + b$

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 51 April 16, 2024

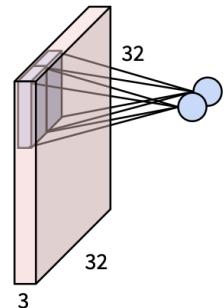
Convolution Layer



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 52 April 16, 2024

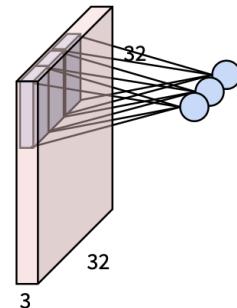
Convolution Layer



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 53 April 16, 2024

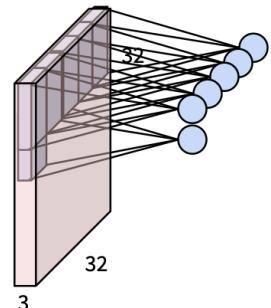
Convolution Layer



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 54 April 16, 2024

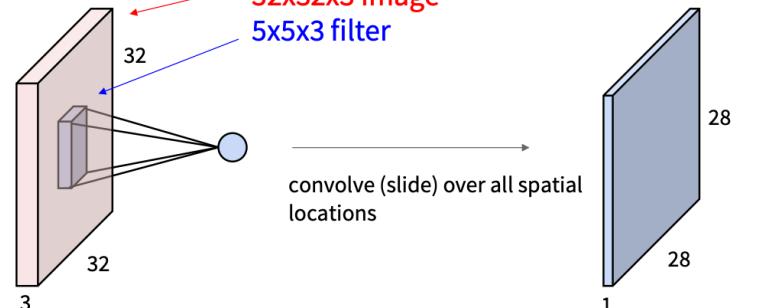
Convolution Layer



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 55 April 16, 2024

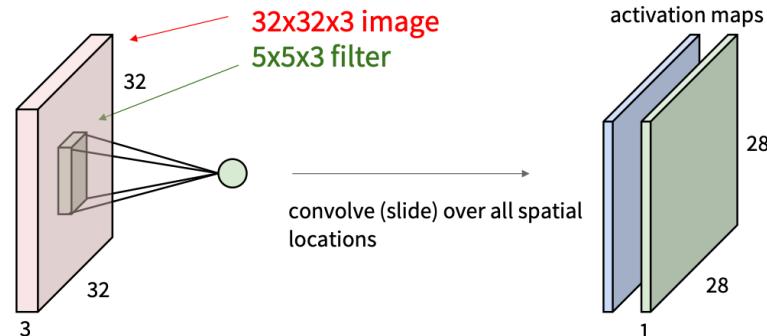
Convolution Layer



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 56 April 16, 2024

Convolution Layer



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 57

April 16, 2024

consider a second, green filter

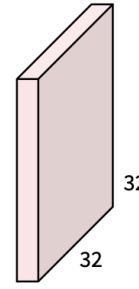
32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

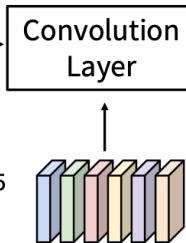
activation maps
28
1
28

Convolution Layer

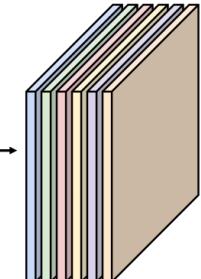
3x32x32 image



Consider 6 filters,
each 3x5x5



6 activation maps,
each 1x28x28



Stack activations to get a
6x28x28 output image!

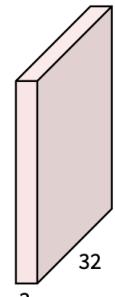
Slide inspiration: Justin Johnson

Lecture 5 - 58

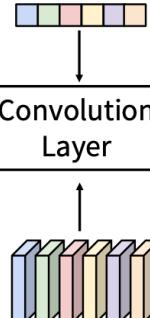
April 16, 2024

Convolution Layer

3x32x32 image



Also 6-dim bias vector:

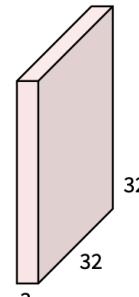


6 activation maps,
each 1x28x28

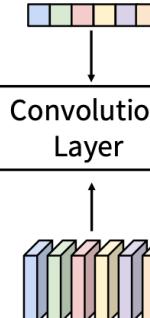
Stack activations to get a
6x28x28 output image!

Convolution Layer

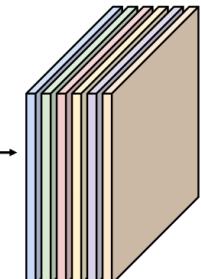
3x32x32 image



Also 6-dim bias vector:



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



Stack activations to get a
6x28x28 output image!

Slide inspiration: Justin Johnson

Lecture 5 - 60

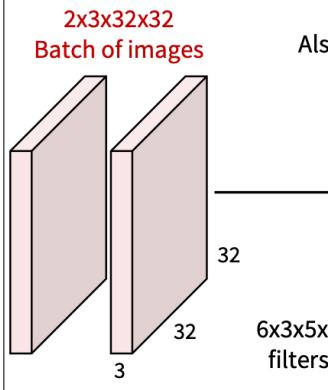
April 16, 2024

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 59

April 16, 2024

Convolution Layer



Also 6-dim bias vector:
Convolution Layer
filters

Batch of outputs
 $2 \times 6 \times 28 \times 28$

Slide inspiration: Justin Johnson

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 61

April 16, 2024

Convolution Layer

N x C_{in} x H x W
Batch of images

C_{in}

H

W

N

C_{out}

C_{out} x C_{in} x K_w x K_h
filters

Also C_{out}-dim bias vector:
Convolution Layer

C_{out}

N x C_{out} x H' x W'
Batch of outputs

H'

W'

N

C_{out}

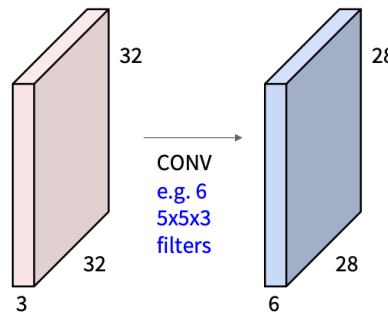
Slide inspiration: Justin Johnson

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 62

April 16, 2024

Preview: ConvNet is a sequence of Convolution Layers

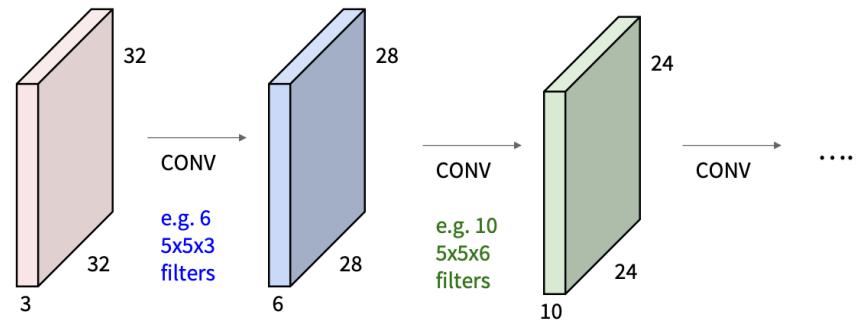


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 63

April 16, 2024

Preview: ConvNet is a sequence of Convolution Layers

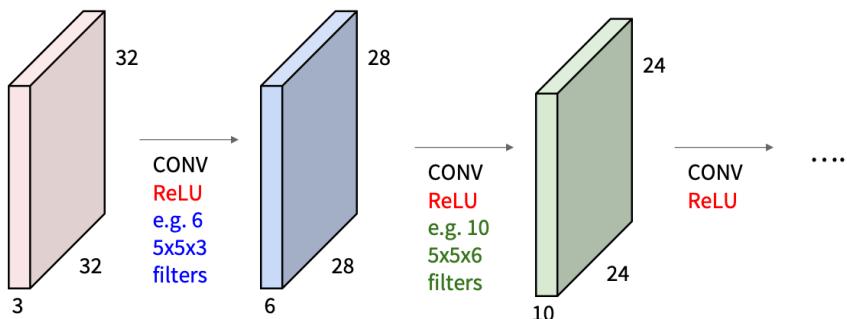


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 64

April 16, 2024

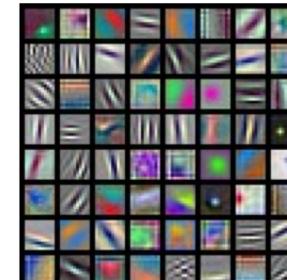
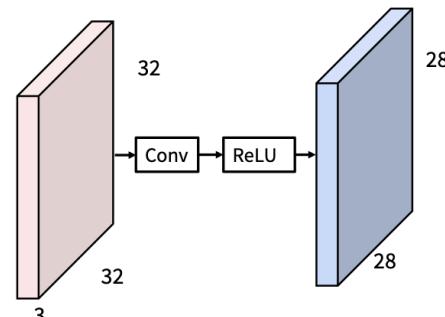
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 65 April 16, 2024

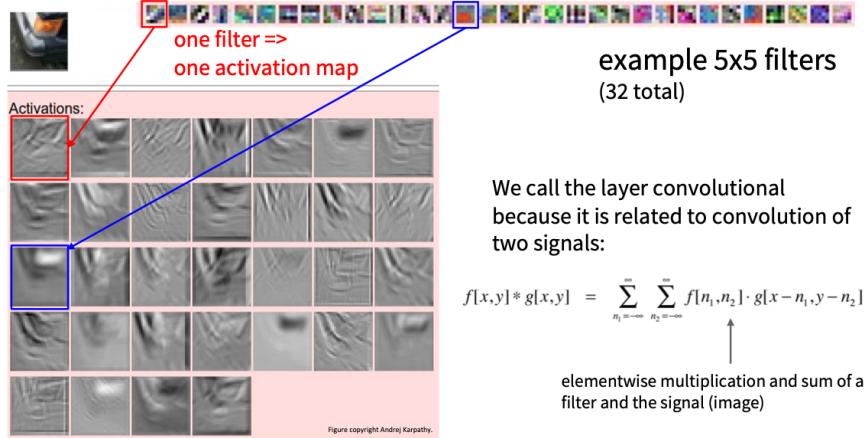
Preview: What do convolutional filters learn?



AlexNet: 64 filters, each $3 \times 11 \times 11$

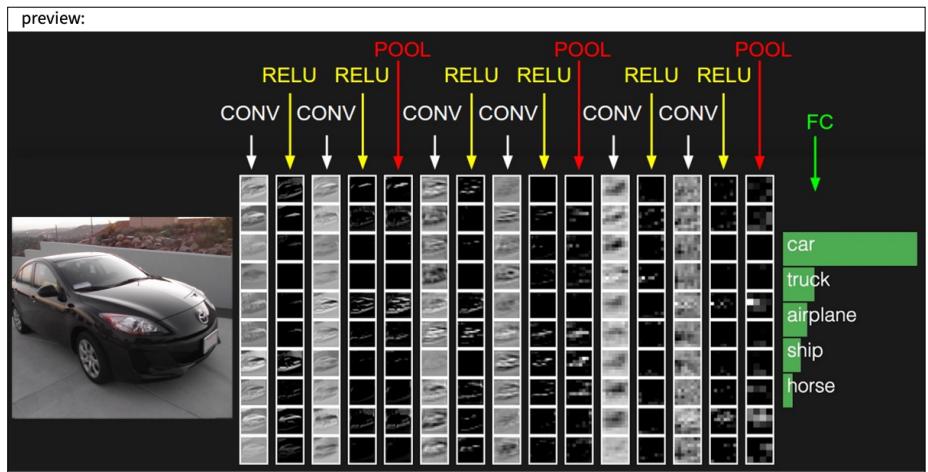
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 68 April 16, 2024



Fei-Fei Li, Ehsan Adeli

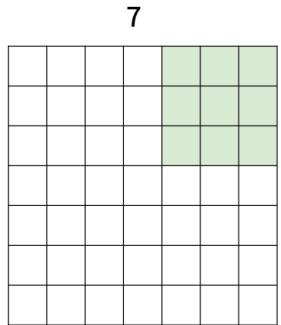
Lecture 5 - 69 April 16, 2024



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 70 April 16, 2024

A closer look at spatial dimensions:

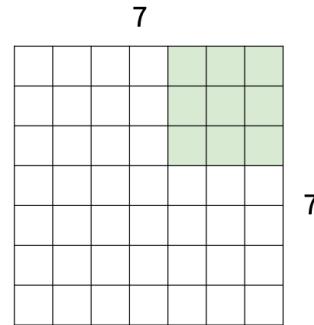


7x7 input (spatially)
assume 3x3 filter
=> 5x5 output

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 76 April 16, 2024

A closer look at spatial dimensions:

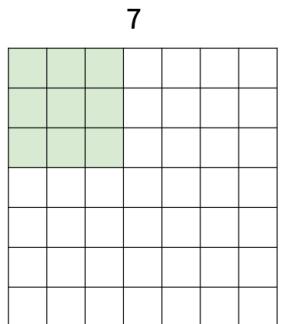


7x7 input (spatially)
assume 3x3 filter
applied with stride 2
=> 3x3 output!

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 79 April 16, 2024

A closer look at spatial dimensions:



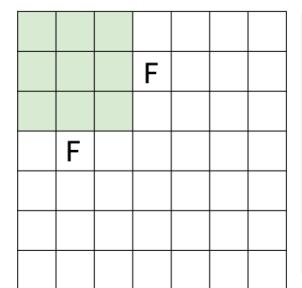
7x7 input (spatially)
assume 3x3 filter
applied with stride 3?

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

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 81 April 16, 2024

N



Output size:
 $(N - F) / \text{stride} + 1$

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

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 82 April 16, 2024

In practice: Common to zero pad the border

0	0	0	0	0	0	0	
0							
0							
0							
0							

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

(recall):
$$(N - F) / \text{stride} + 1$$

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 83 April 16, 2024

In practice: Common to zero pad the border

0	0	0	0	0	0	0	
0							
0							
0							
0							

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

7x7 output!

(recall):
$$(N + 2P - F) / \text{stride} + 1$$

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 84 April 16, 2024

In practice: Common to zero pad the border

0	0	0	0	0	0	0	
0							
0							
0							
0							

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

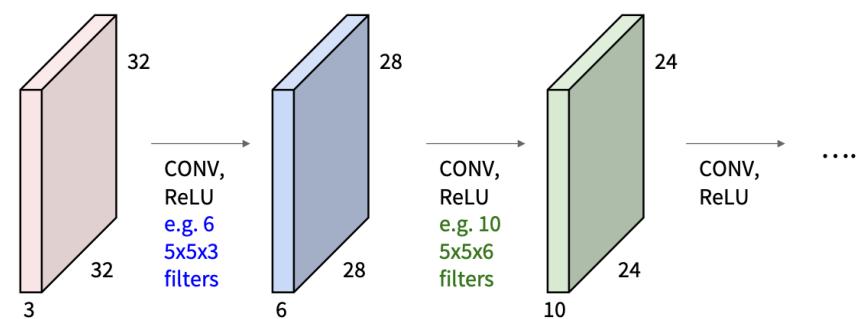
7x7 output!
in general, common to see CONV layers with stride 1,
filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will
preserve size spatially)
e.g. $F=3 \Rightarrow$ zero pad with 1
 $F=5 \Rightarrow$ zero pad with 2
 $F=7 \Rightarrow$ zero pad with 3

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 85 April 16, 2024

Remember back to...

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

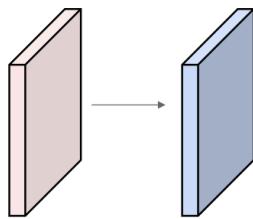


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 86 April 16, 2024

Examples time:

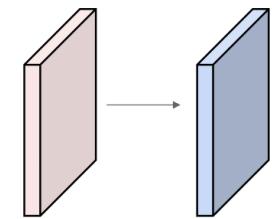
Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2



Output volume size: ?

Examples time:

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

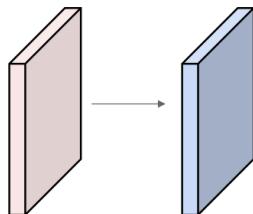


Output volume size:

$(32+2*2-5)/1+1 = 32$ spatially, so
32x32x10

Examples time:

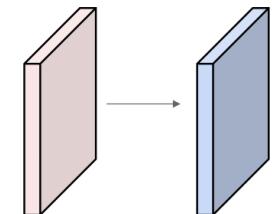
Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

Examples time:

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

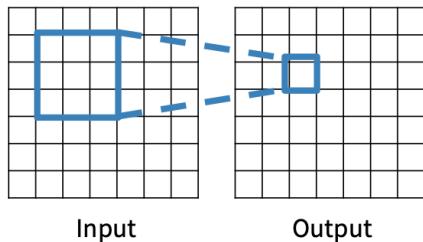


Number of parameters in this layer?
each filter has $5*5*3 + 1 = 76$ params
=> $76*10 = 760$

(+1 for bias)

Receptive Fields

For convolution with **kernel size K**, each element in the output depends on a $K \times K$ receptive field in the input



Slide inspiration: Justin Johnson

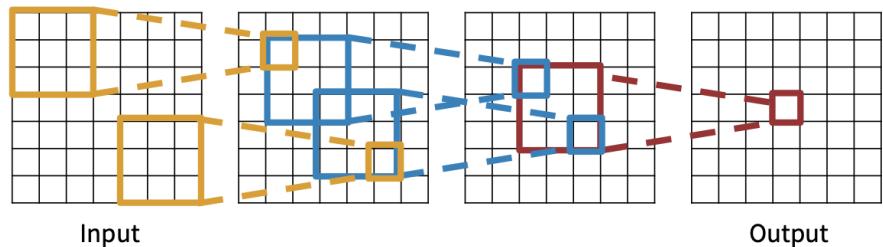
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 91

April 16, 2024

Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size
With L layers the receptive field size is $1 + L * (K - 1)$



Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

Slide inspiration: Justin Johnson

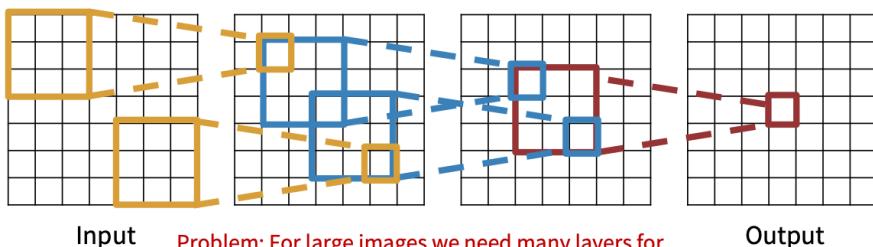
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 92

April 16, 2024

Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size
With L layers the receptive field size is $1 + L * (K - 1)$



Slide inspiration: Justin Johnson

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 94

April 16, 2024

Convolution layer: summary

Common settings:

Let's assume input is $W_1 \times H_1 \times C$
Conv layer needs 4 hyperparameters:

- Number of filters K
 - The filter size F
 - The stride S
 - The zero padding P
- K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
 - F = 5, S = 1, P = 2
 - F = 5, S = 2, P = ? (whatever fits)
 - F = 1, S = 1, P = 0

This will produce an output of $W_2 \times H_2 \times K$
where:

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

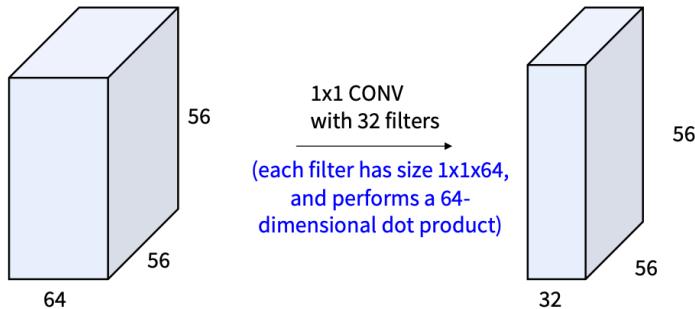
Number of parameters: F^2CK and K biases

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 98

April 16, 2024

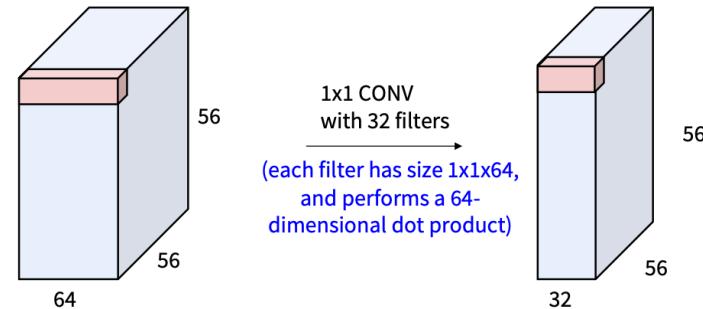
(btw, 1x1 convolution layers make perfect sense)



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 99 April 16, 2024

(btw, 1x1 convolution layers make perfect sense)



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 100 April 16, 2024

Example: CONV layer in PyTorch

```
Conv2d
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True)
[SOURCE]
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{out}, H_{out}, W_{out})$ can be precisely described:

$$\text{out}(N_i, C_{out,j}) = \text{bias}(C_{out,j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out,j}, k) * \text{input}(N_i, k)$$

where $*$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is a width in pixels.

- stride controls the stride for the cross-correlation, a single number or a tuple.
- padding controls the amount of implicit zero-paddings on both sides for padding number of points for each dimension.
- dilation controls the spacing between the kernel points, also known as the a trous algorithm. It is harder to describe, but this [link](#) has a nice visualization of what dilation does.
- groups controls the connections between inputs and outputs. in_channels and out_channels must both be divisible by groups . For example:

- At $\text{groups}=1$, all inputs are convolved to all outputs.

At $\text{groups}=2$, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.

At $\text{groups}=\text{in_channels}$, each input channel is convolved with its own set of filters, of size $\left[\frac{C_{out}}{C_{in}}\right]$.

Conv layer needs 4 hyperparameters:

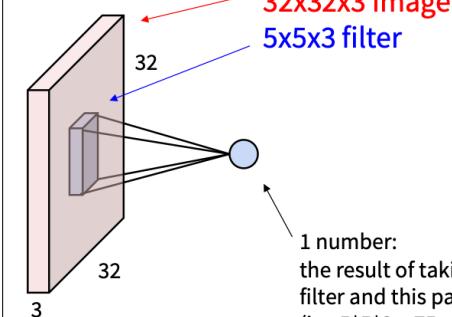
- Number of filters K
- The filter size F
- The stride S
- The zero padding P

The parameters kernel_size , stride , padding , dilation can either be:

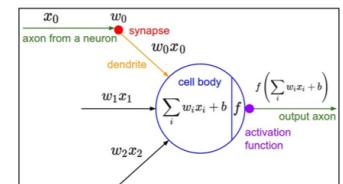
- a single int – in which case the same value is used for the height and width dimensions
- a tuple of two ints – in which case, the first int is used for the height dimension, and the second int for the width dimension

PyTorch is licensed under [BSD 3-clause](#)

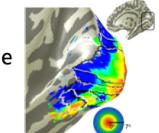
The brain/neuron view of CONV Layer



Fei-Fei Li, Ehsan Adeli



It's just a neuron with local connectivity...



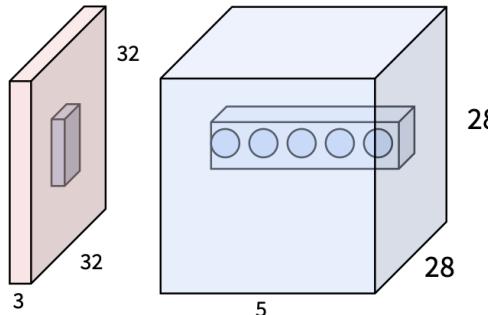
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product)

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 101 April 16, 2024

Lecture 5 - 103 April 16, 2024

The brain/neuron view of CONV Layer



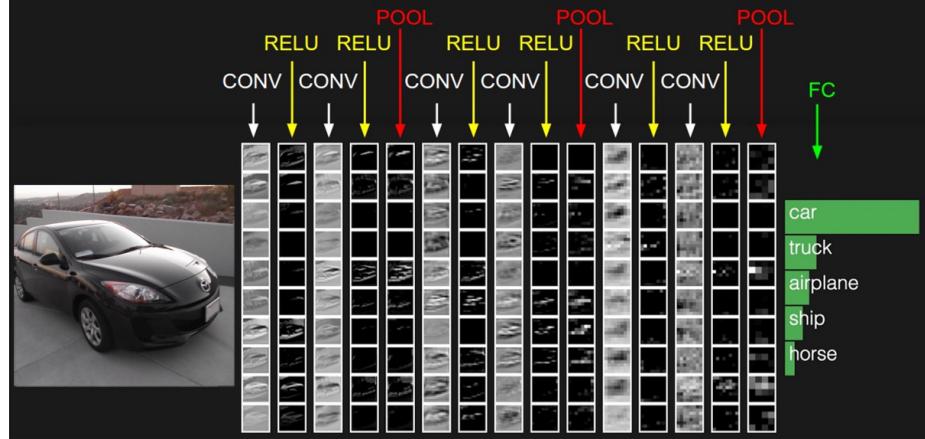
E.g. with 5 filters,
CONV layer consists of neurons
arranged in a 3D grid
($28 \times 28 \times 5$)

There will be 5 different neurons
all looking at the same region in
the input volume

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 104 April 16, 2024

Neural Network Architecture

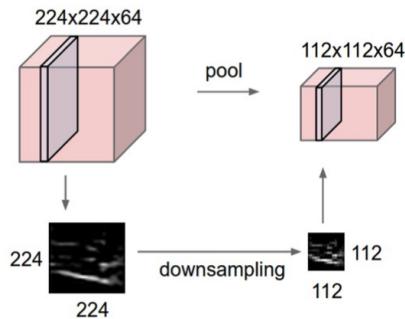


Fei-Fei Li, Ehsan Adeli

Lecture 5 - 106 April 16, 2024

Pooling layer

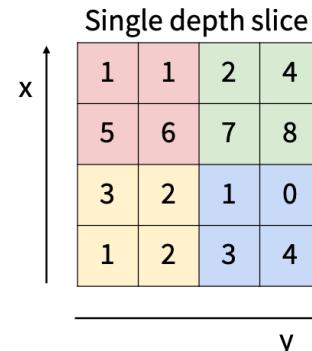
- makes the representations smaller and more manageable
- operates over each activation map independently



Fei-Fei Li, Ehsan Adeli

Lecture 5 - 107 April 16, 2024

MAX POOLING



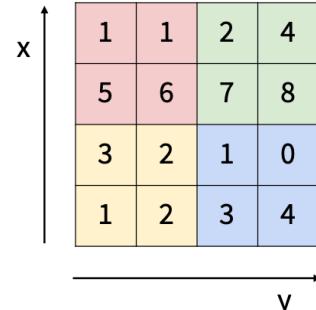
Fei-Fei Li, Ehsan Adeli

Lecture 5 - 108 April 16, 2024

MAX POOLING

Single depth slice

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



max pool with 2x2 filters
and stride 2

- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 2 hyperparameters:

- The spatial extent F
- The stride S

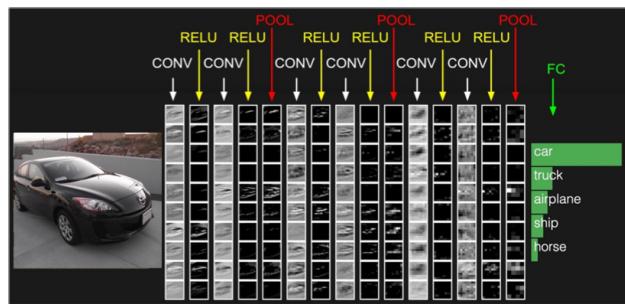
This will produce an output of $W_2 \times H_2 \times C$ where:

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

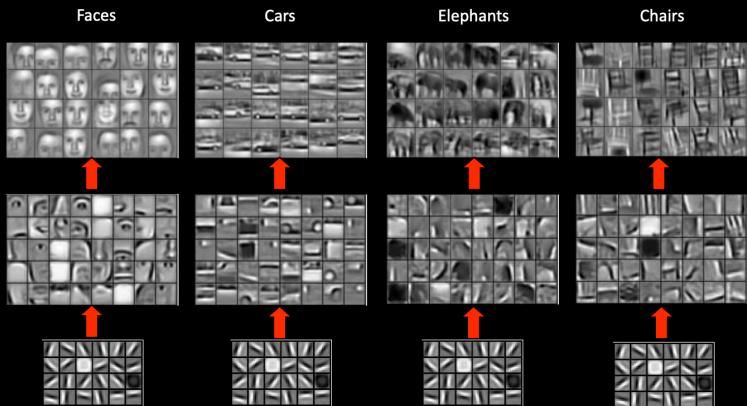
Number of parameters: 0

Fully Connected Layer (FC layer)

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



Features learned from training on different object classes.



Alexnet

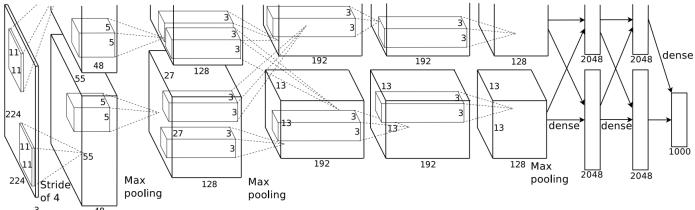


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

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

Pytorch

```
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

AlexNet was originally introduced in "ImageNet Classification with Deep Convolutional Neural Networks". This implementation is based instead on the "One Weird Trick for Parallelizing Convolutional Neural Networks" paper.

single-column model with 64, 192, 384, 384, 256 filters in the five convolutional layers, respectively

```
import torch
from PIL import Image
from torchvision import transforms

model = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet', pretrained=True)
model.eval()

input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
# create a mini-batch as expected by the model
input_batch = input_tensor.unsqueeze(0)

if torch.cuda.is_available():
    input_batch = input_batch.to('cuda')
    model.to('cuda')

with torch.no_grad():
    output = model(input_batch)
    probabilities = torch.nn.functional.softmax(output[0], dim=0)

print(torch.topk(probabilities, 5))
# https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt
```

Pytorch

[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

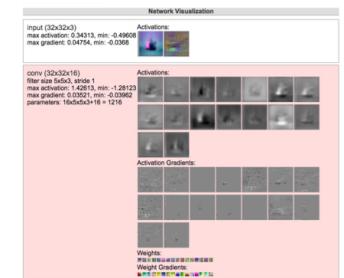
Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is about 94% (not perfect as the dataset can be a bit ambiguous). I used this [python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).



<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

Summary

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

Fei-Fei Li, Ehsan Adeli

Lecture 5 - 113 April 16, 2024

Transfer learning

Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 130 April 17, 2024

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

Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 131 April 17, 2024

Transfer Learning with CNNs

1. Train on Imagenet



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

Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 132 April 17, 2024

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

this is a vanilla
differentiable function...

Fei-Fei Li, Ehsan Adeli, Zane Durante

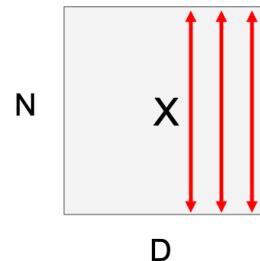
Lecture 6 - 12

April 17, 2024

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



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

Per-channel mean,
shape is D

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

Per-channel var,
shape is D

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

Normalized x,
Shape is $N \times D$

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

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 14

April 17, 2024

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$

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

Per-channel mean,
shape is D

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

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

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

Per-channel var,
shape is D

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

Normalized x,
Shape is $N \times D$

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

Output,
Shape is $N \times D$

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 15

April 17, 2024

Batch Normalization: Test-Time

Estimates depend on minibatch;
can't do this at test-time!

Input: $x : N \times D$

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

Per-channel mean,
shape is D

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

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

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

Per-channel var,
shape is D

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

Normalized x,
Shape is $N \times D$

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

Output,
Shape is $N \times D$

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 16

April 17, 2024

Batch Normalization: Test-Time

Input: $x : N \times D$

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

Per-channel mean, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

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

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

Per-channel var, shape is D

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

Normalized x, Shape is $N \times D$

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

Output, Shape is $N \times D$

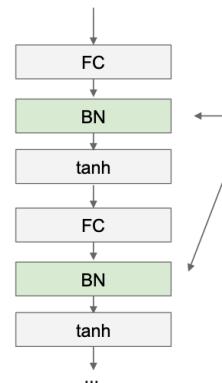
Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 17

April 17, 2024

[Ioffe and Szegedy, 2015]

Batch Normalization



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

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

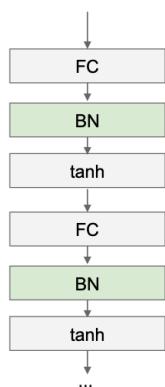
Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 18

April 17, 2024

Batch Normalization

[Ioffe and Szegedy, 2015]



- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as a kind of regularization during training
- Behaves differently during training and testing: this is a very common source of bugs!

Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$$\mathbf{x} : N \times D$$

Normalize

$$\mu, \sigma : 1 \times D$$

$$\gamma, \beta : 1 \times D$$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Batch Normalization for **convolutional** networks
(Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x} : N \times C \times H \times W$$

Normalize

$$\mu, \sigma : 1 \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 19

April 17, 2024

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 20

April 17, 2024

Layer Normalization

Batch Normalization for fully-connected networks

$\mathbf{x}: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Layer Normalization for fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$\mathbf{x}: N \times D$

Normalize

$\mu, \sigma: N \times 1$

$\gamma, \beta: 1 \times D$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Ba, Kiros, and Hinton, "Layer Normalization", arXiv 2016

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 21

April 17, 2024

Instance Normalization

Batch Normalization for convolutional networks

$\mathbf{x}: N \times C \times H \times W$

Normalize

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Instance Normalization for convolutional networks
Same behavior at train / test!

$\mathbf{x}: N \times C \times H \times W$

Normalize

$\mu, \sigma: N \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$$\mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 6 - 22

April 17, 2024

Data Augmentation

Transfer Learning

Normalizing Activations

Learning Rate Scheduling

Modern Optimizers (Adam, AdamW)

Proposal (due Mar 28th)

- Title/Team Members
- Introduction (1 page)
 - provide context and clearly define the problem/task
- Data (.5 to 1 page)
 - describe the nature of the data (include figures if available)
 - indicate whether you are using an existing dataset or you are collecting/developing your own
 - provide some stats: number of samples, number of features (e.g., dimension of images)
- Methods (.5 to 1 page)
 - describe your proposed/tentative approach
- Preliminary Results (optionally — if available)

(Unity)

Quiz 2 => Thursday 27th

CNN Assignment => Thursday 3rd