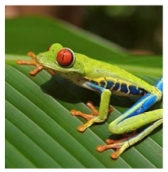
Lecture 5:
Image Classification with CNNs

Image Classification: A core task in Computer Vision

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

→ cat
dog
bird
deer
truck

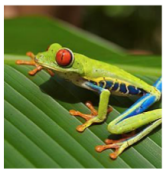
Pixel space



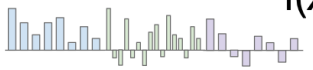
Class scores

f(x) = Wx

Image features



f(x) = Wx



Feature Representation

Class scores

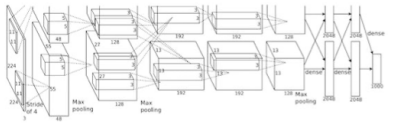
Image features vs. ConvNets

Feature Extraction

f

10 numbers giving scores for classes

training



10 numbers giving scores for classes

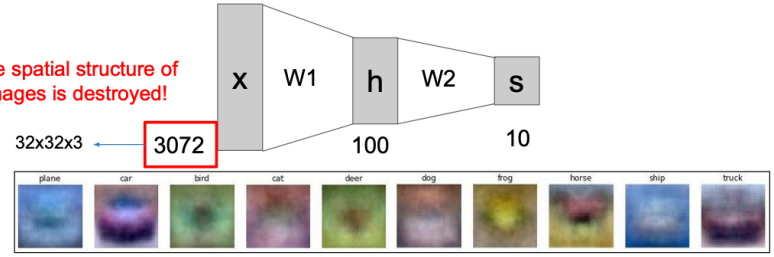
training

Last Time: Neural Networks

Linear score function: $f = Wx$

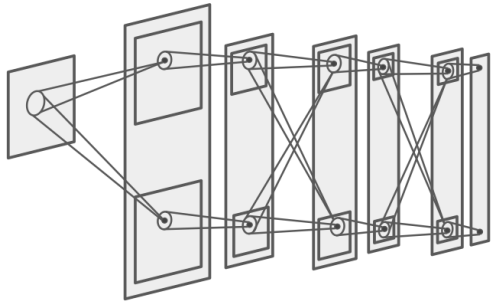
2-layer Neural Network $f = W_2 \max(0, W_1 x)$

The spatial structure of images is destroyed!



A bit of history:

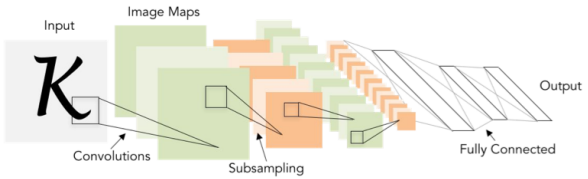
Neocognitron
[Fukushima 1980]



"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history:

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



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

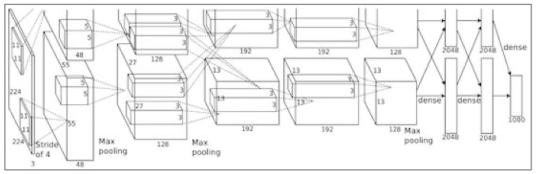


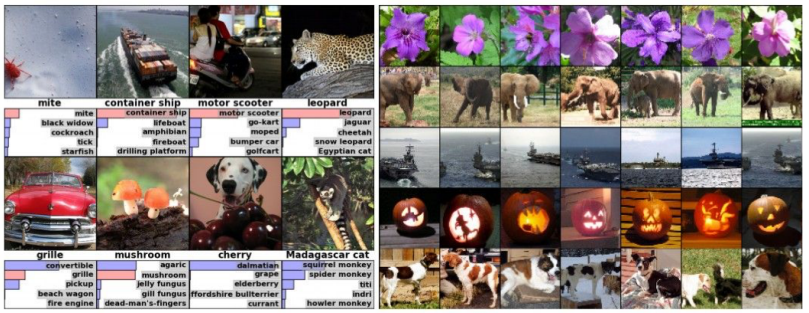
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

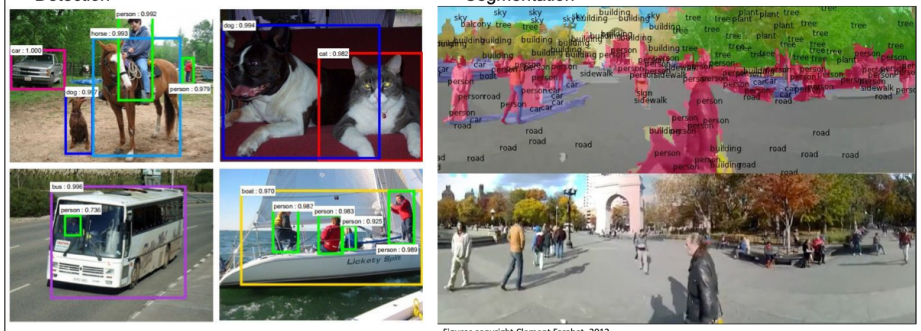


Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

Detection

Segmentation



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, 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]

No errors

Minor errors

Somewhat related

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

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]

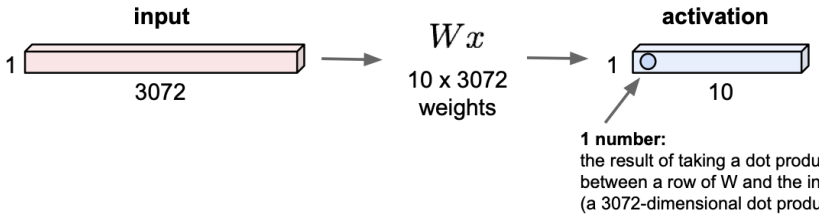
All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/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

Captions generated by Justin Johnson using Neuraltalk2

Convolutional Neural Networks

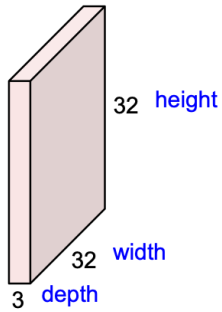
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Convolution Layer

32x32x3 image -> preserve spatial structure

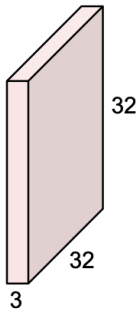


Convolution Layer

Filters always extend the full depth of the input volume

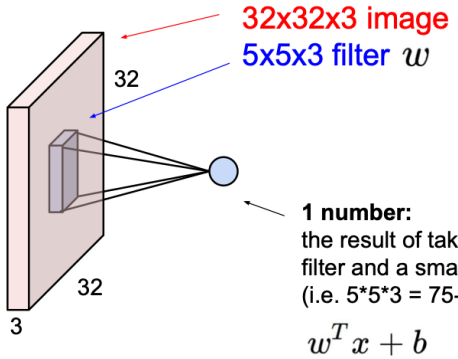
32x32x3 image

5x5x3 filter



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

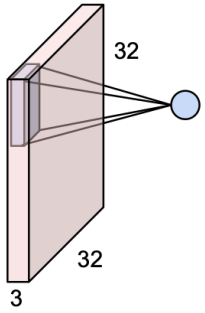
Convolution Layer



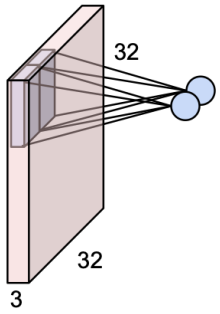
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

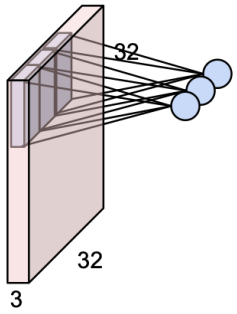
Convolution Layer



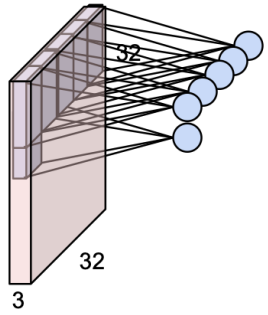
Convolution Layer



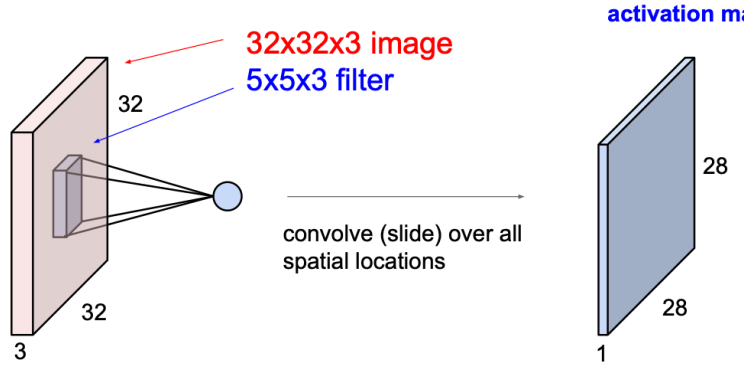
Convolution Layer



Convolution Layer



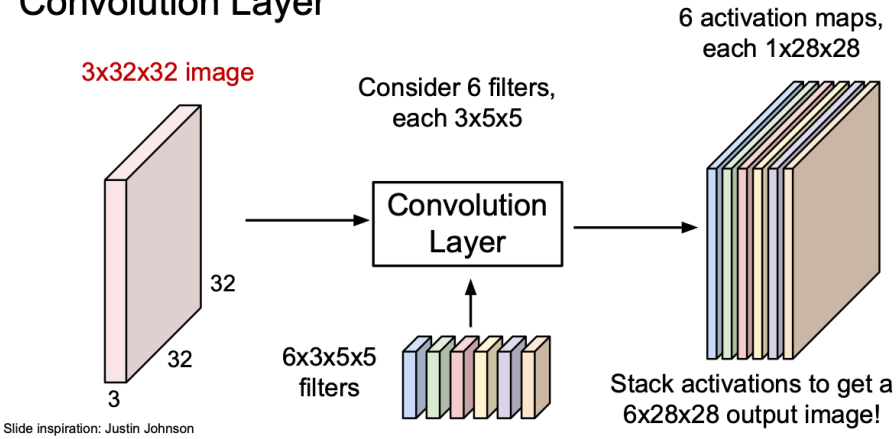
Convolution Layer



Convolution Layer

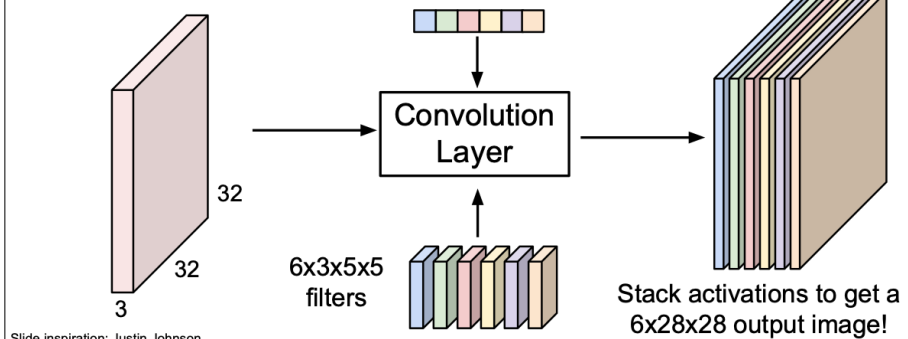
consider a second, green filter

Convolution Layer



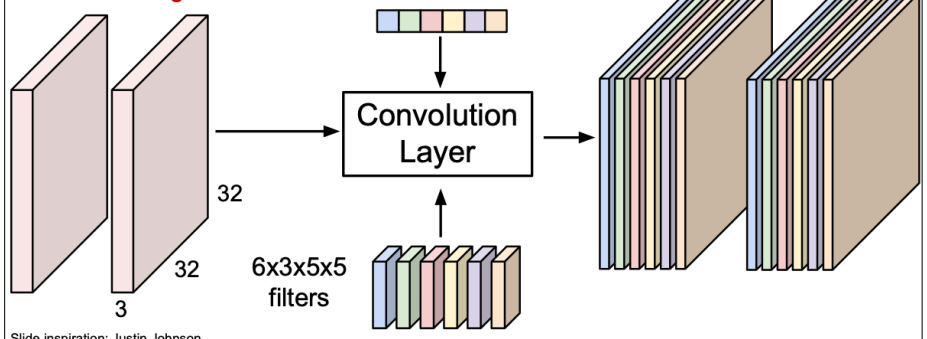
Slide inspiration: Justin Johnson

Convolution Layer



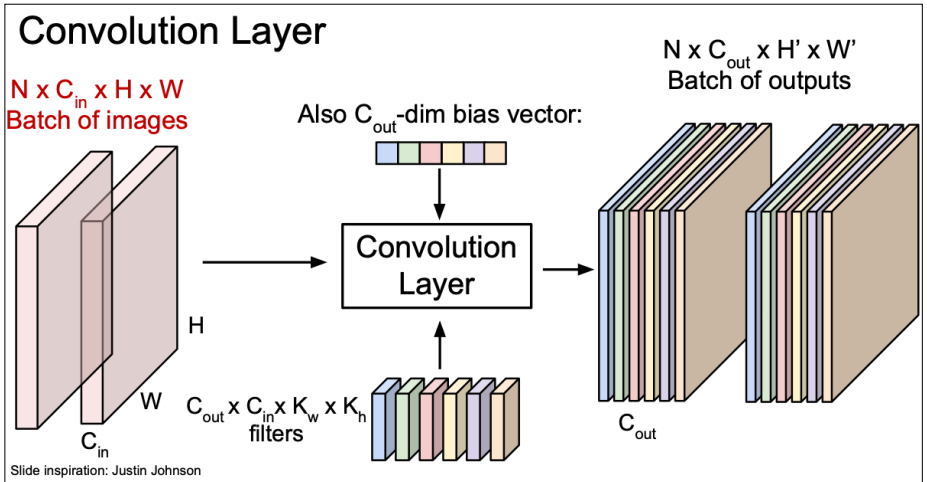
Slide inspiration: Justin Johnson

Convolution Layer



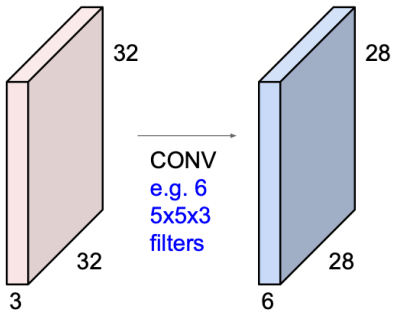
Slide inspiration: Justin Johnson

Convolution Layer

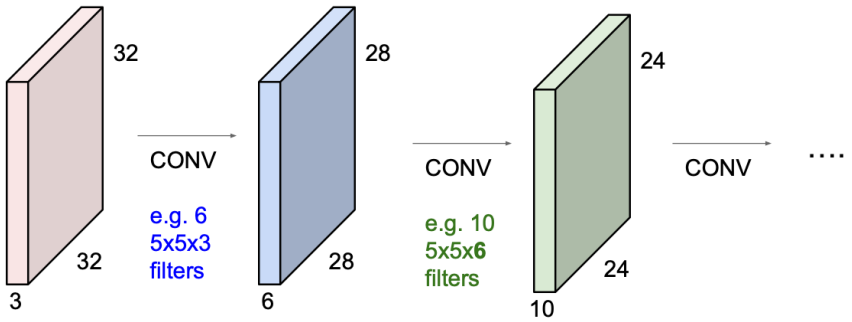


Slide inspiration: Justin Johnson

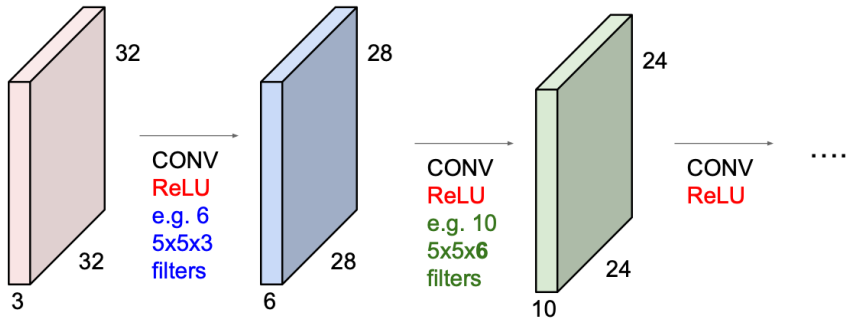
Preview: ConvNet is a sequence of Convolution Layers



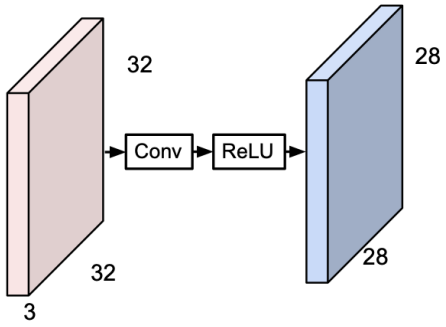
Preview: ConvNet is a sequence of Convolution Layers



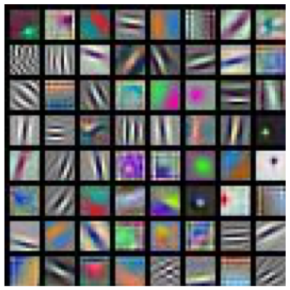
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



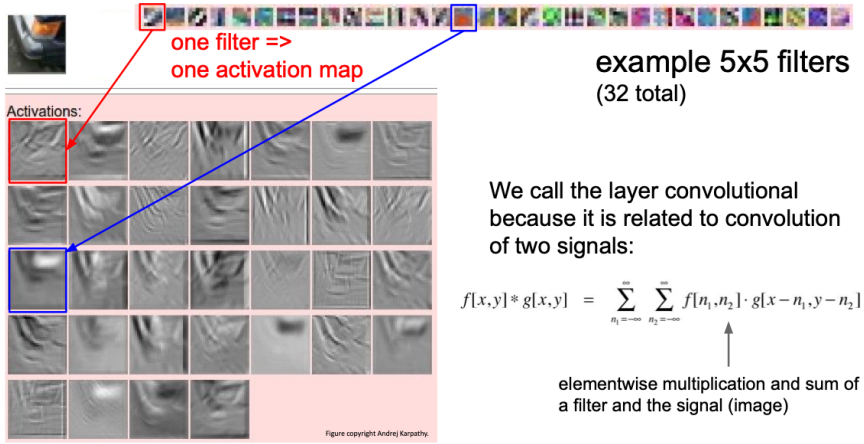
Preview: What do convolutional filters learn?



First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11



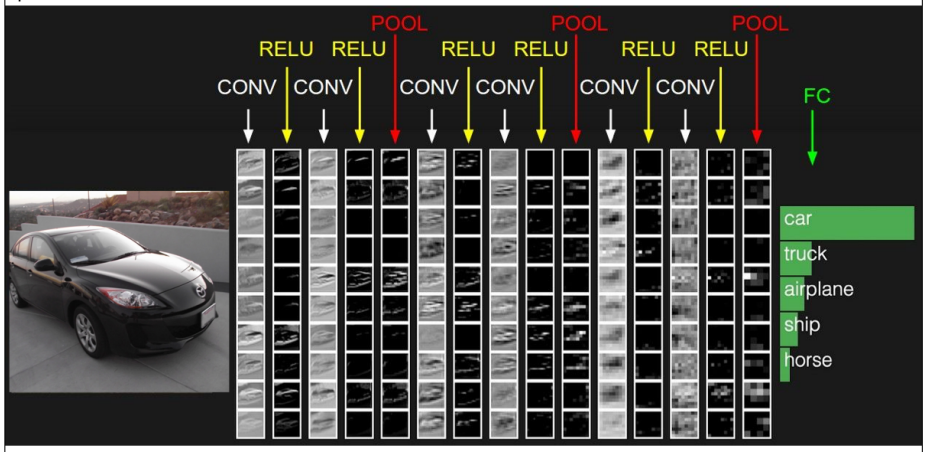
example 5x5 filters
(32 total)

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

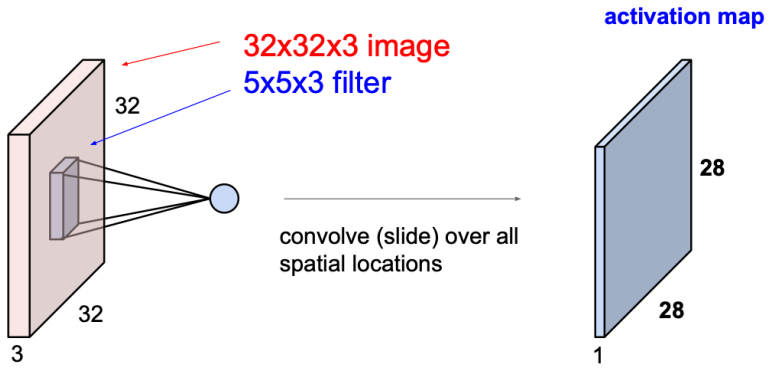
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

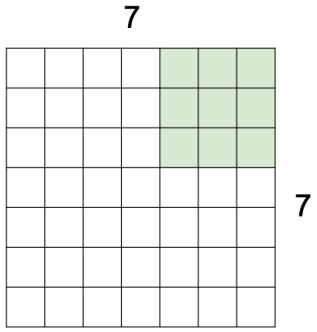
preview:



A closer look at spatial dimensions:



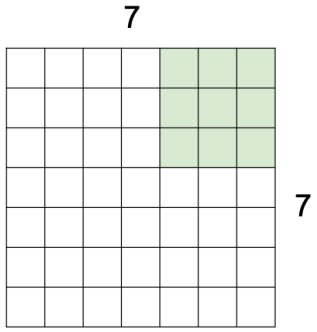
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

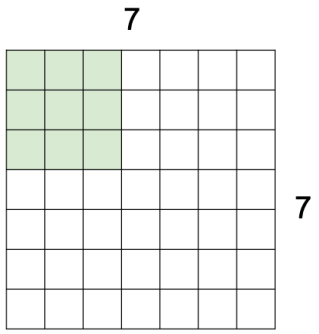
=> 5x5 output

A closer look at spatial dimensions:



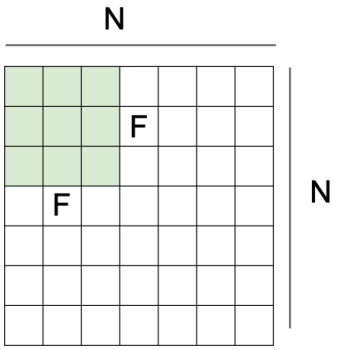
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

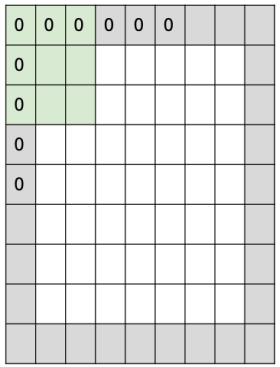
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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

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

In practice: Common to zero pad the border

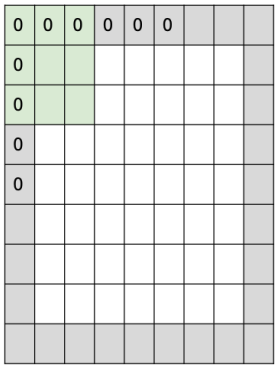


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N + 2P - F) / stride + 1

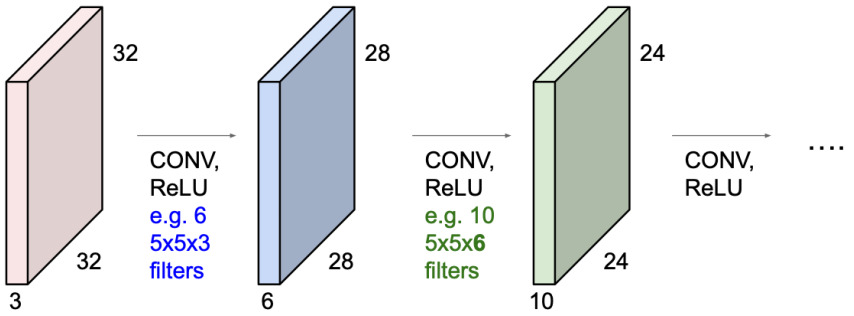
In practice: Common to zero pad the border



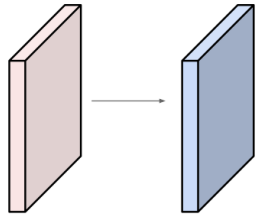
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

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



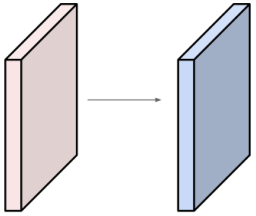
Examples time:



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

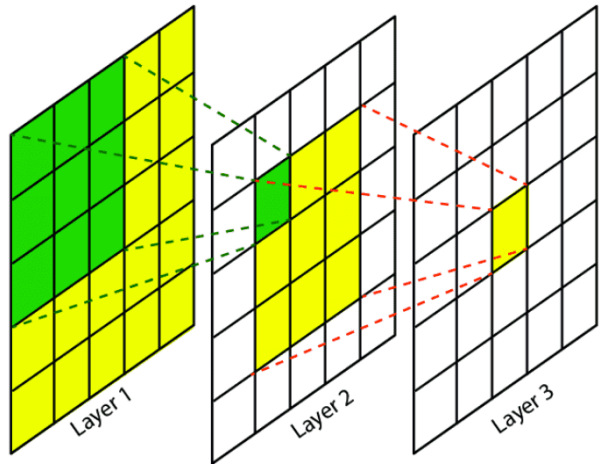
Number of parameters in this layer?

Receptive Fields

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

Input Output

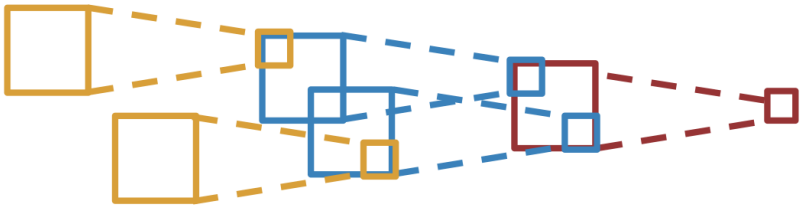
Slide inspiration: Justin Johnson



https://theaisummer.com/receptive-field/

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)



Input

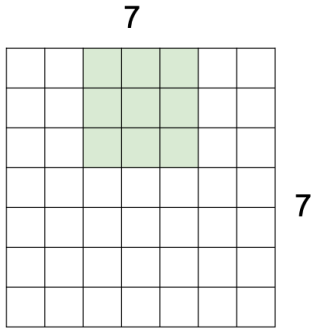
Problem: For large images we need many layers for each output to "see" the whole image image

Output

Solution: Downsample inside the network

Slide inspiration: Justin Johnson

Solution: Strided Convolution



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

=> 3x3 output!

Convolution layer: summary

Common settings:

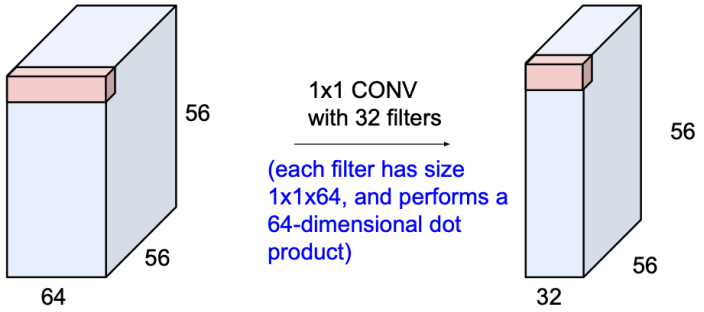
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:
- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

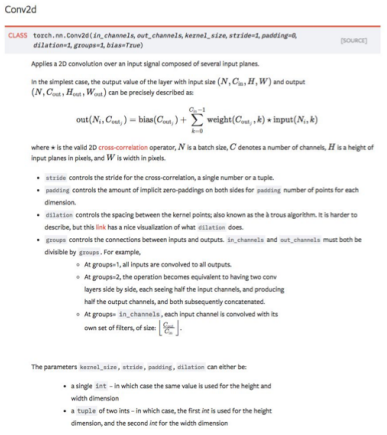
This will produce an output of W_2 x H_2 x K
where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

(btw, 1x1 convolution layers make perfect sense)



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 as:

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

where $\star$ 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 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 à 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 groups=1, all inputs are convolved to all outputs.
  - At 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 groups= in_channels, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{out}}{C_{in}} \right\rfloor$.

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

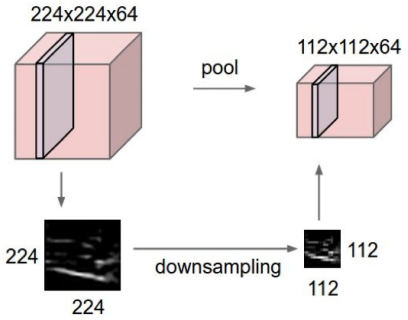
- a single int – in which case the same value is used for the height and width dimension
- a tuple of two ints – in which case, the first int is used for the height dimension, and the second int for the width dimension

PyTorch is licensed under BSD 3-clause.

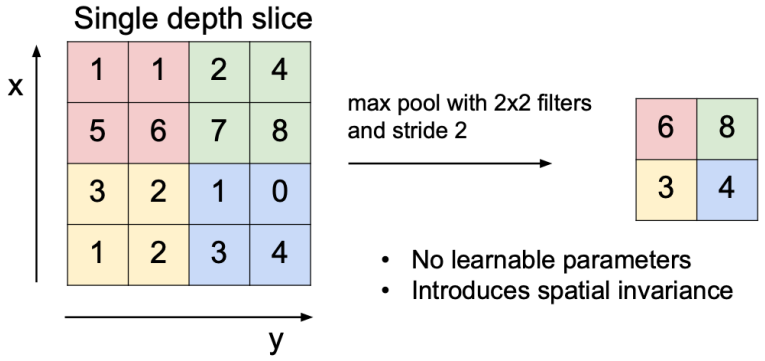
Conv layer needs 4 hyperparameters:
- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Pooling layer

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



MAX POOLING



- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 2 hyperparameters:
- The spatial extent **F**
- The stride **S**

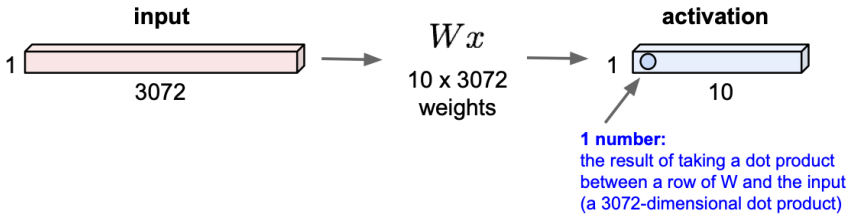
This will produce an output of W_2 x H_2 x C where:
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

Reminder: Fully Connected Layer

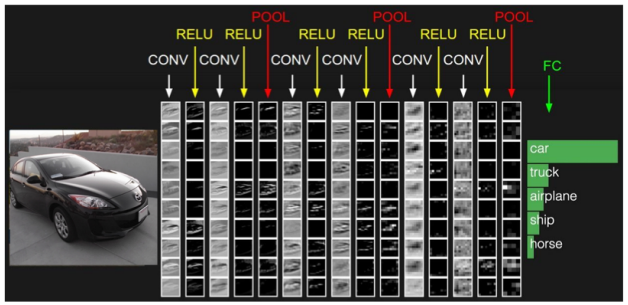
32x32x3 image -> stretch to 3072 x 1

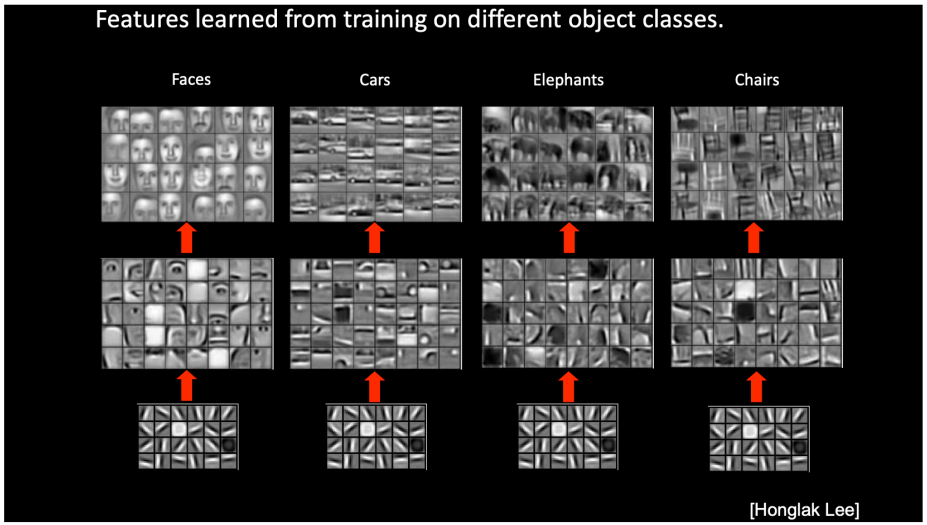
Each neuron looks at the full input volume



Fully Connected Layer (FC layer)

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





Alexnet

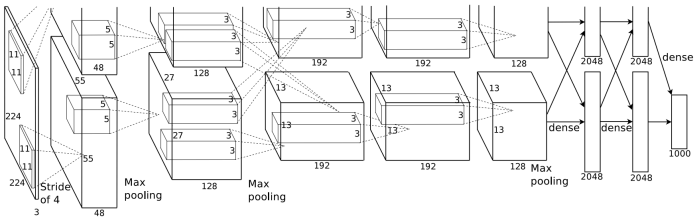


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input 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

```python
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

Pytorch

```python
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
```

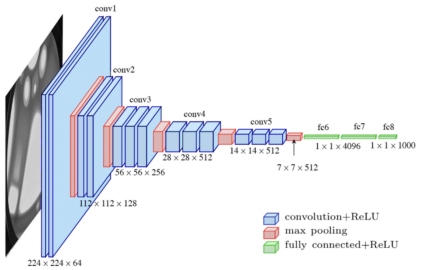
VGG 16

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

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

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

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



"Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR, 2015

Keras

```
def VGG16():
    model = Sequential()

    model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', input_shape=(224,224,3)))
    model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2), strides=2))

    filters_convs = [(128, 2), (256, 3), (512, 3), (512,3)]
    for n_filters, n_convs in filters_convs:
        for _ in np.arange(n_convs):
            model.add(Conv2D(filters=n_filters, kernel_size=(3,3), padding='same', activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2), strides=2))

    model.add(Flatten())
    model.add(Dense(4096, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1000,activation = 'softmax'))

    opt = SGD(lr = 0.01)
    model.compile(loss = categorical_crossentropy, optimizer = opt, metrics = ['accuracy'])

    return model
```

https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484

Keras

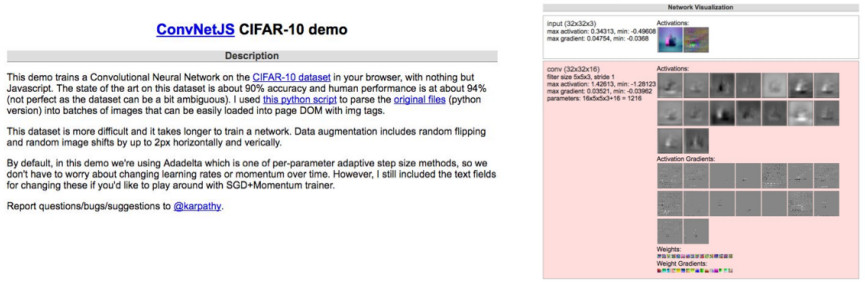
```
# load vgg model
from keras.applications.vgg16 import VGG16

# load the model
model = VGG16()

# prints summary
model.summary()
# Total params: 138,357,544
# Trainable params: 138,357,544
```

```
tf.keras.applications.VGG16(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

[ConvNetJS demo: training on CIFAR-10]



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 <= K <= 2.
- But recent advances such as ResNet/GoogLeNet have challenged this paradigm

Transfer learning

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

Transfer Learning with CNNs

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

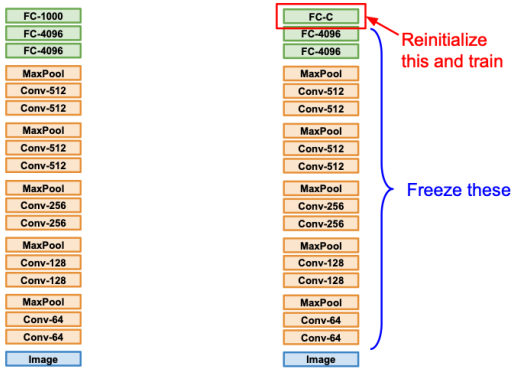
1. Train on Imagenet

Transfer Learning with CNNs

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

1. Train on Imagenet

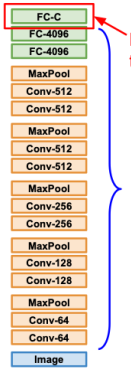
2. Small Dataset (C classes)



Transfer Learning with CNNs

1. Train on Imagenet

2. Small Dataset (C classes)



Reinitialize this and train

Freeze these

Finetuned from AlexNet

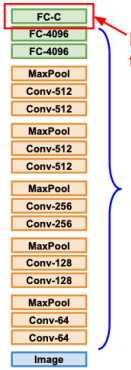
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

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

Transfer Learning with CNNs

1. Train on Imagenet

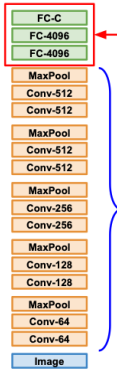
2. Small Dataset (C classes)



Reinitialize this and train

Freeze these

3. Bigger dataset



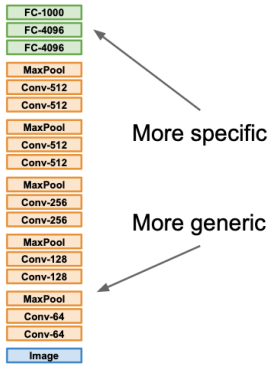
Train these

With bigger dataset, train more layers

Freeze these

Lower learning rate when finetuning; 1/10 of original LR is good starting point

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



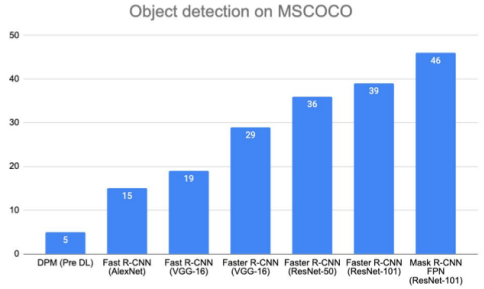
More specific

More generic

	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

Transfer learning with CNNs - Architecture matters

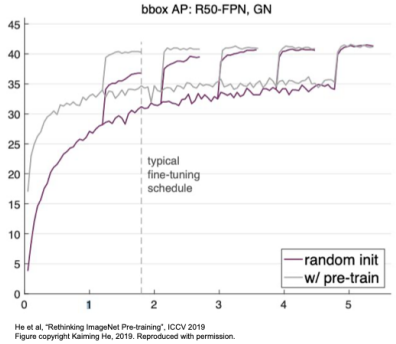
Object detection on MSCOCO



Girshick, "The Generalized R-CNN Framework for Object Detection", ICCV 2017 Tutorial on Instance-Level Visual Recognition

Transfer learning with CNNs is pervasive…

But recent results show it might not always be necessary!



He et al, "Rethinking ImageNet Pre-training", ICCV 2019
Figure copyright Kaiming He, 2019. Reproduced with permission.

Training from scratch can work just as well as training from a pretrained ImageNet model for object detection

But it takes 2-3x as long to train.

They also find that collecting more data is better than finetuning on a related task

Batch Normalization

Consider a single layer $y = Wx$

The following could lead to tough optimization:
- Inputs x are not *centered around zero* (need large bias)
- Inputs x have different scaling per-element (entries in W will need to vary a lot)

Idea: force inputs to be "nicely scaled" at each layer!

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:

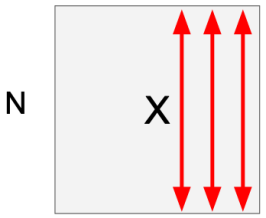
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

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 x D

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

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

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

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

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

Batch Normalization: Test-Time

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

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

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

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

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

Batch Normalization: Test-Time

Input: $x : N \times D$

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

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

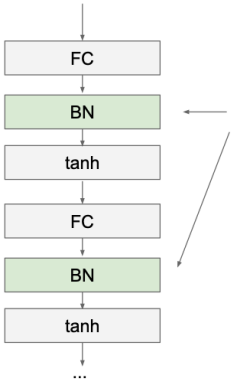
$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

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

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

Batch Normalization

[Ioffe and Szegedy, 2015]

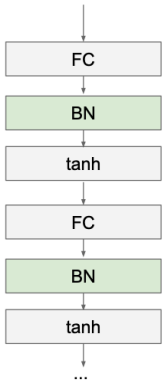


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

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

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 regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$x: N \times D$

Normalize

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

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

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

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

$x: N \times C \times H \times W$

Normalize

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

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

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

Layer Normalization

Batch Normalization for fully-connected networks

$x: N \times D$

Normalize

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

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

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

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

$x: N \times D$

Normalize

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

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

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

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

Instance Normalization

Batch Normalization for convolutional networks

$x: N \times C \times H \times W$

Normalize

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

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

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

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

$x: N \times C \times H \times W$

Normalize

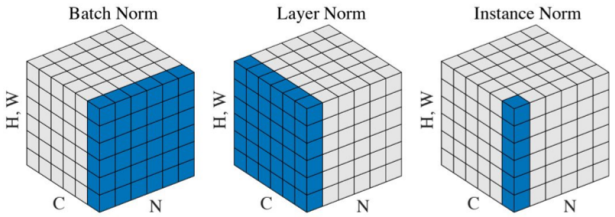
$\mu, \sigma: N \times C \times 1 \times 1$

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

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

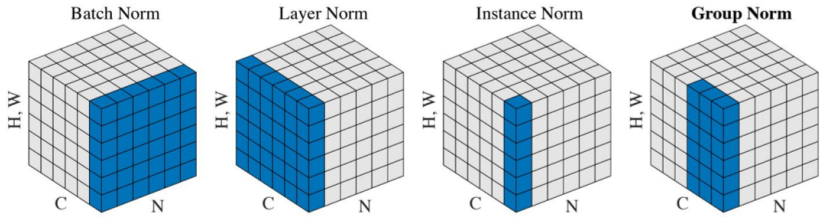
Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Comparison of Normalization Layers



Wu and He, "Group Normalization", ECCV 2018

Group Normalization



Wu and He, "Group Normalization", ECCV 2018