

...a short tutorial on

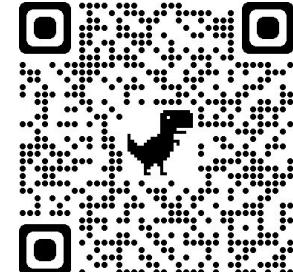
Image Representations and Fine-Grained Recognition

Yannis Kalantidis
Research Scientist
NAVER LABS Europe

Virtual talk @ HUST
17th May 2021

Personal webpage:
<https://www.skamalas.com>

Slides:



<https://www.skamalas.com/#hust>

Overview

- Introduction
 - What is a “representation”?
 - Extracting vs. Learning Representations
- Convolutional Neural Networks (CNNs)
 - Basic components and architectures
 - Pytorch example
- Training Convolutional Neural Networks
 - Loss function and regularization
 - Important tips for training image models
- Fine-grained recognition
 - Best practices for fine-grained Recognition
 - Tackling small and imbalanced datasets

Overview

- **Introduction**
 - **What is a “representation”?**
 - **Extracting vs. Learning Representations**
- Convolutional Neural Networks (CNNs)
 - Basic components and architectures
 - Pytorch example
- Training Convolutional Neural Networks
 - Loss function and regularization
 - Important tips for training image models
- Fine-grained recognition
 - Best practices for fine-grained Recognition
 - Tackling small and imbalanced datasets

What is a “representation”?

“Representation” or “feature” in Machine Learning:

(usually) a **compact** vector that describes the input data

$$x = [x^1, \dots, x^d], x \in \mathbb{R}^d$$

What is an image representation?

in **Computer Vision**

(usually) a **compact** vector that describes the visual content of an image

A **global** image representation

- a high-dimensional vector
- a set of classes present

What is an image representation?

in **Computer Vision**

(usually) a **compact** vector that describes the visual content of an image

A **global** image representation

- a high-dimensional vector
- a set of classes present

But can also be **multiple local features**

- a histogram of edges or gradients
- a set of regions of interest and their features

Comparing visual representations

- Measure similarity/*distance* between images. Let $x_i, x_j \in \mathbb{R}^d$

$$\begin{aligned} d(x_i, x_j) &= \|x_i - x_j\|^2 \\ &= \sum_{k=1}^d (x_i^k - x_j^k)^2 \end{aligned}$$

What are visual representations useful for?

Classification

- Given a set of classes/labels and an unseen image, classify the image

Detection/Segmentation

- Use multiple features, usually from a set of regions

Video understanding

- Tracking and spatio-temporal localization

Cross-modal search and generation

- Image captioning and description

Image Classification

- Given a set of classes/labels and an unseen image, classify the image

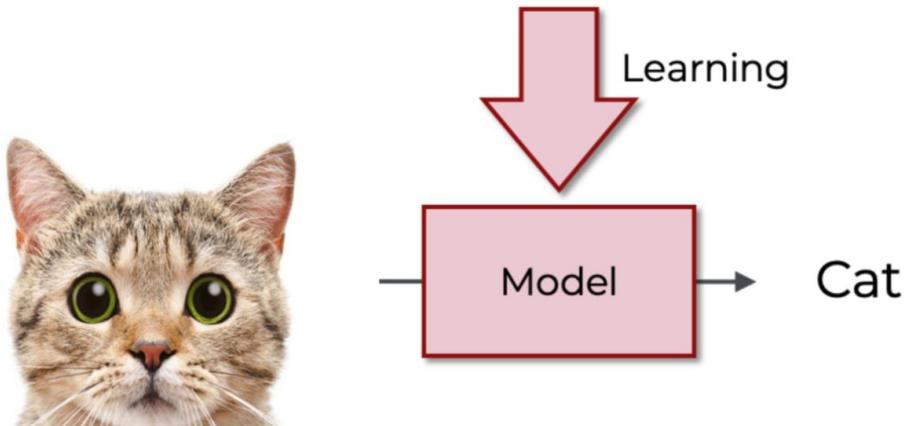
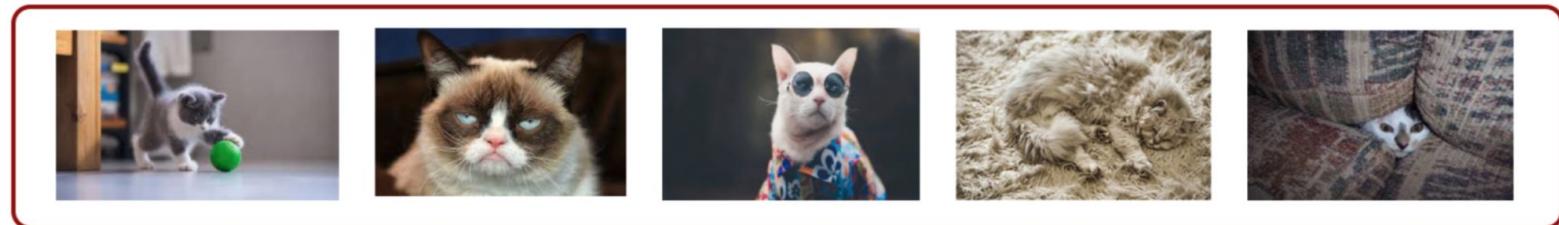


Image Classification

- Given a set of classes/labels and an unseen image, classify the image

We need to learn a ***classifier*** on top of the representations

$$f(x_i; W) = Wx_i$$

Image Classification

- Given a set of classes/labels and an unseen image, classify the image
- Datasets: ImageNet ...or [identify snake species](#) [1], [crops from space](#) [2] or [cassava leaf diseases](#) [3]

We need to learn a **classifier** on top of the representations

$$f(x_i; W) = Wx_i$$

[\[1\] Snake species classification challenge](#)

[\[2\] Farm Pin Crop Detection Challenge @ zindi.africa](#)

[\[3\] iCassava Challenge 2019](#)

Image Classification: iCassava

iCassava 2019 Fine-Grained Visual Categorization Challenge

Ernest Mwebaze, Timnit Gebru, Andrea Frome
Google Research
[\[emwebaze,tgebru,afrome\]@google.com](mailto:[emwebaze,tgebru,afrome]@google.com)

Solomon Nsumba, Jeremy Tusubira
Artificial Intelligence lab
Makerere University
[\[snsumba,jtusubira\]@gmail.com](mailto:[snsumba,jtusubira]@gmail.com)

Chris Omongo
National Crops Resources Research Institute
P.O. Box 7084 Kampala, Uganda.
chrisomongo@gmail.com



(a) Healthy



(b) CBB



(c) CGM



(d) CMD



(e) CBSD



(a) Background effects

(b) Time of day effects



(c) Multiple disease

(d) Poor focus effects

FGVC8

The Eight Workshop on Fine-Grained Visual Categorization

June 25th 2021

Organized in conjunction with CVPR 2021



10,000 species, with a training dataset of 2.7M images

NYBG
NEW YORK BOTANICAL GARDEN



2.5M images representing nearly 66,000 species
from the Americas, Oceania and the Pacific.

Any datasets from Vietnam??

The screenshot shows a Kaggle dataset page titled "Vietnamese Foods". The page features a large image of a bowl of pho. At the top left, there's a "Dataset" icon. On the right, there's a navigation bar with a back arrow, a number "6", and a forward arrow. Below the title, it says "More than 24.000 images of 29 Vietnamese foods with urls". A profile picture of Quan Dang and the text "Quan Dang • updated 2 months ago" are visible. The overall theme is food-related.

Note that I do not endorse this dataset.
...it was just the only one I found :)

<https://www.kaggle.com/quandang/vietnamese-foods>

A	B
1199	Image
1200	Url https://scontent-hkt1-1.cdninstagram.com/v/f51_2885_15/e35/p1080x1080/117810513_190355020310945_117810513_n.jpg
1201	Image

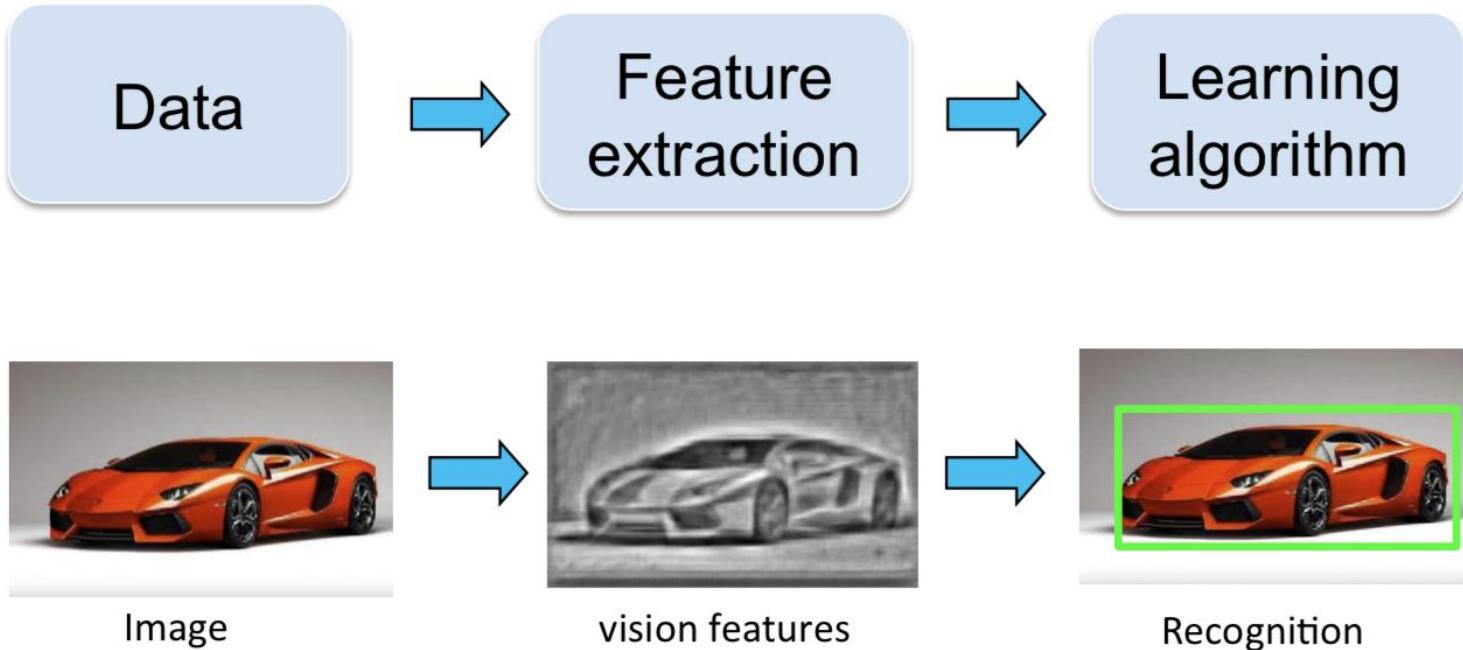
TOOL GẮN NHẶN

Tên hình: Image 1200.jpg
Nguồn lấy: Instagram
Trạng thái: Chưa check
Nhận hiện tại:
Phổ
Di chuyển Xóa hình

+ Bình béo Bánh bột lọc Bánh canh Bánh chưng Bánh cuốn Bánh cắn Bánh gỏi Bánh khọt

Khemona

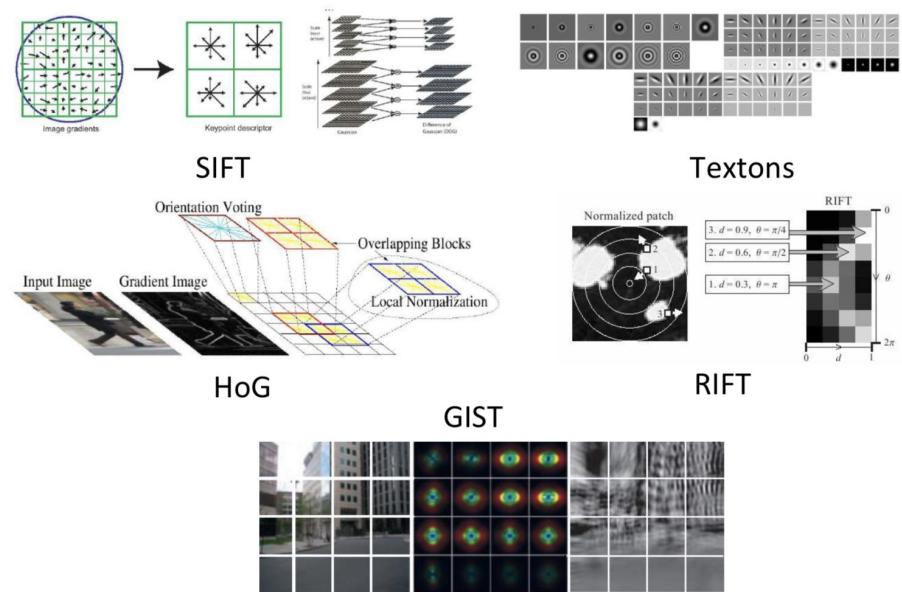
Extracting vs. Learning Representations



Extracting vs. Learning Representations

Feature Extraction: “hand-crafted” representations

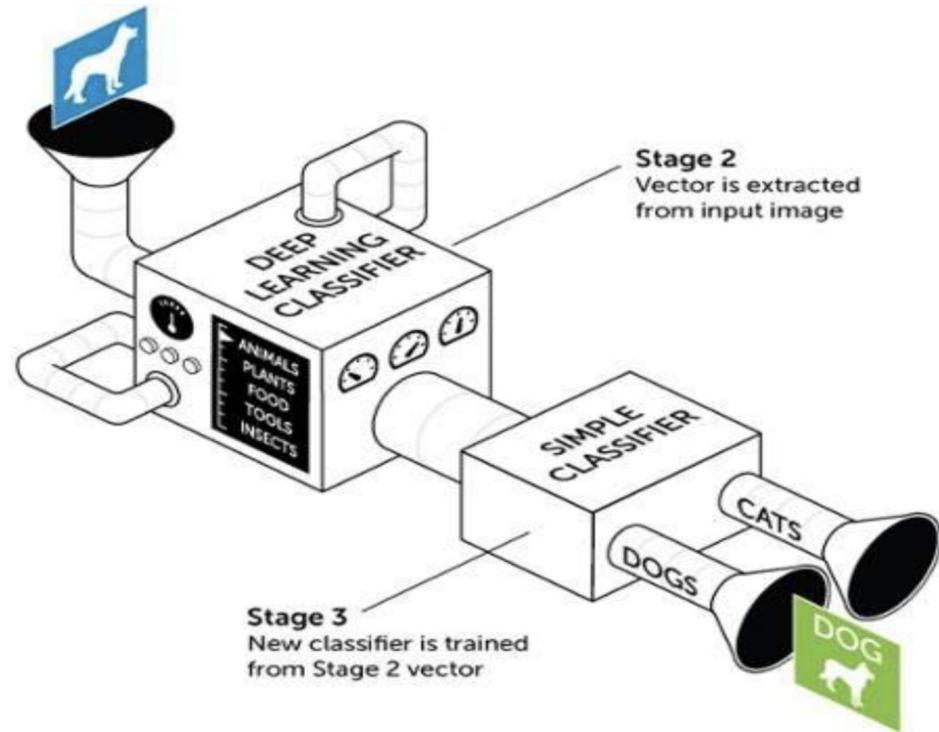
- Utilizes domain knowledge
- Requires domain expertise
- Most common approach for decades



Extracting vs. Learning Representations

Representation Learning

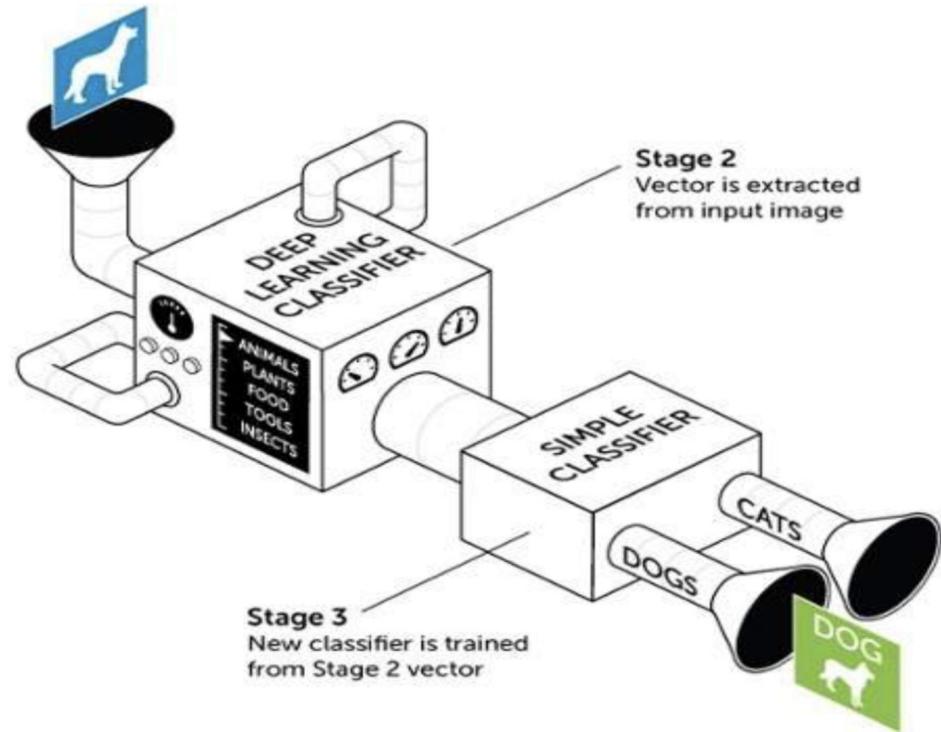
- Don't design features
- Design **models** that output representations and predictions
- Don't tell the model how to solve your task; tell the model what result you want to get



Extracting vs. Learning Representations

Representation Learning

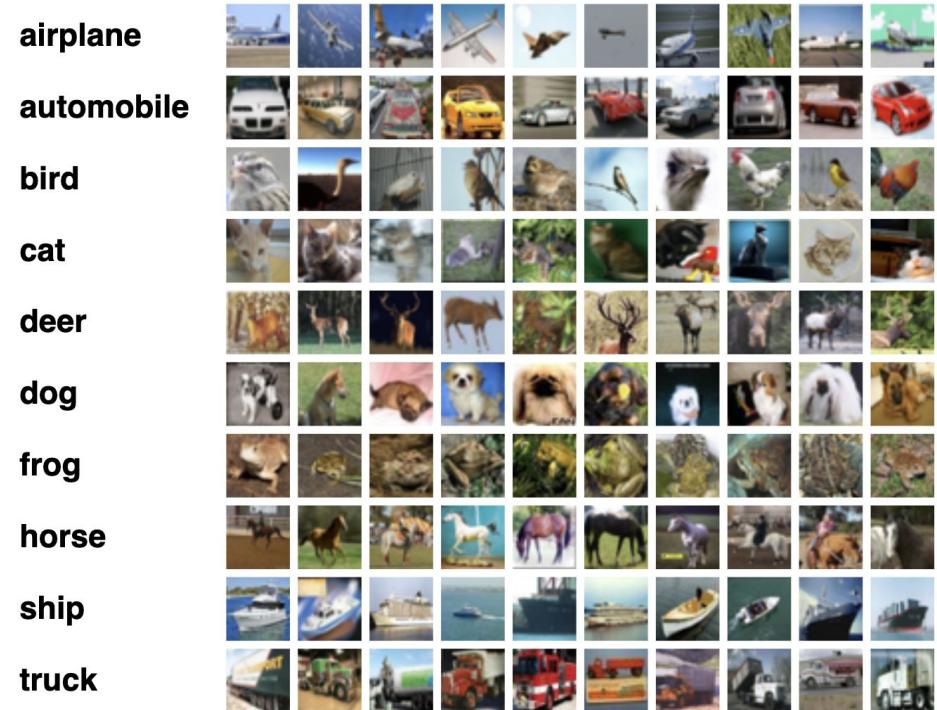
- 1) Collect a dataset of images and labels
- 2) Use machine learning to train a model and classifier
- 3) Evaluate on new images



Learning Image Representations

Dataset

- Images
- Labels/Annotations



[CIFAR10 dataset]

Learning Image Representations

Model

- Use dataset to learn the parameters of a **model** that gives you a representation and a **classifier**
- Given the model and classifier, predict the label for a new image

Which model to use?

Deep Convolutional Neural Networks

Overview

- Introduction
 - What is a “representation”?
 - Extracting vs. Learning Representations
- **Convolutional Neural Networks (CNNs)**
 - **Basic components and architectures**
 - **Pytorch example**
- Training Convolutional Neural Networks
 - Loss function and regularization
 - Important tips for training image models
- Fine-grained recognition
 - Best practices for fine-grained Recognition
 - Tackling small and imbalanced datasets

Convolutional Neural Networks (CNNs)

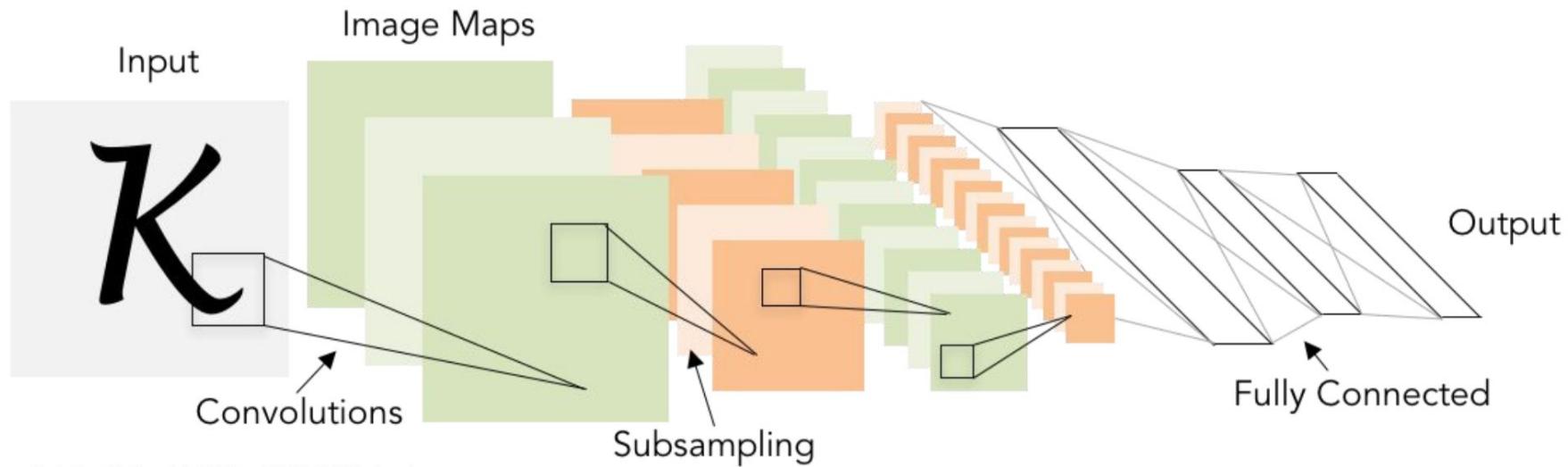
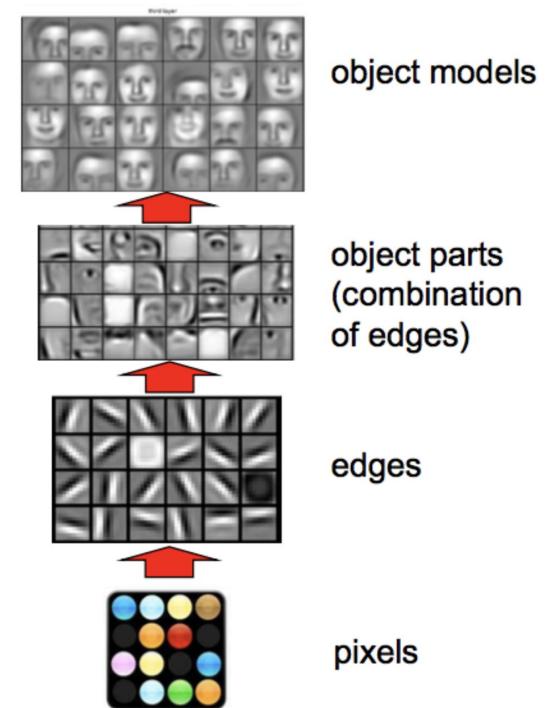


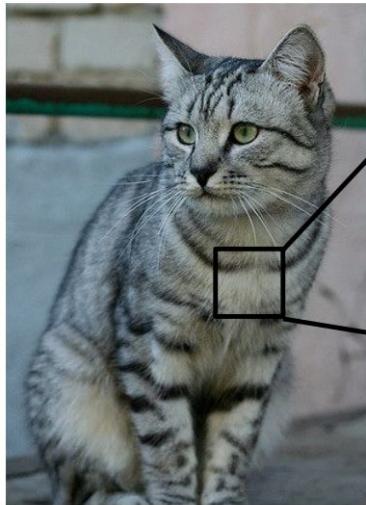
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Why “Deep” Networks?

- Inspiration from mammal brains
 - [\[Rumelhart et al 1986\]](#)
- Train each layer with the representations of the previous layer to learn a higher level abstraction
- Pixels → Edges → Contours →
Object parts → Object categories
- Local Features → Global Features

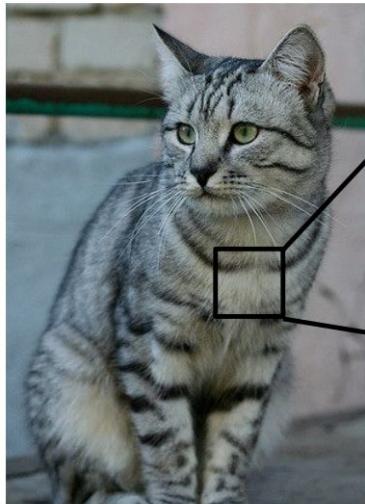


Data/Input representation: Pixel intensities

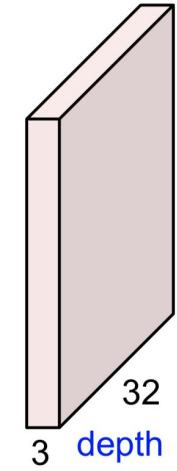


[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
[91 98 102 106 104 79 98 103 99 105 123 136 118 105 94 90 85]
[76 85 98 105 128 105 87 95 99 102 112 106 103 90 89 85]
[99 81 81 93 120 131 127 100 95 102 98 96 93 104 94]
[104 89 95 102 100 98 95 104 107 109 108 98 92 96 95]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 88 65 52 54 74 84 102 93 85 82]
[128 137 144 140 105 95 86 70 62 65 63 63 68 73 86 101]
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 98 89 75 61 64 72 84]
[115 114 109 123 150 148 131 118 113 109 108 92 74 65 72 78]
[89 93 98 97 101 147 131 118 113 114 113 109 106 95 77 80]
[63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]
[62 65 82 89 78 71 88 101 124 126 119 101 107 114 131 119]
[63 65 75 88 88 71 62 81 120 130 130 105 81 98 110 118]
[87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
[118 97 102 88 97 123 116 66 51 50 95 93 89 95 104 107]
[144 146 139 108 93 89 95 104 108 106 106 81 104 102 109]
[157 170 157 120 93 86 114 132 112 97 69 55 78 82 99 104]
[130 128 134 161 139 109 109 118 121 134 114 87 65 53 69 85]
[120 112 96 117 150 144 128 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 153 149 122 109 102 75 88 107 112 99]
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]

Data/Input representation: Pixel intensities

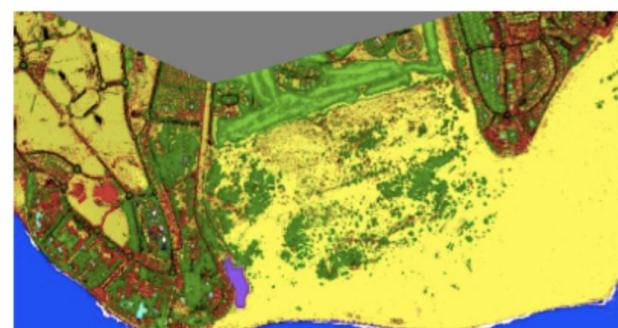
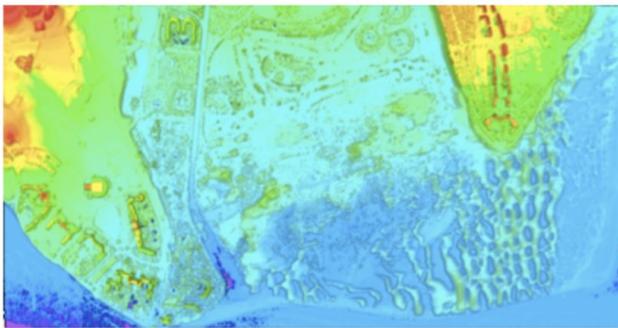


[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
[91 98 102 106 104 79 98 103 99 105 123 136 118 105 94 85]
[76 85 98 105 128 105 87 95 99 102 112 106 103 89 85]
[99 81 81 93 120 131 127 100 95 102 98 96 93 104 94]
[104 85 85 85 95 100 105 100 95 102 98 96 93 104 94]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 88 65 52 54 74 84 102 93 85 82]
[128 137 144 140 105 95 86 70 62 65 63 63 68 73 86 101]
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 98 89 75 61 64 72 84]
[115 114 109 123 150 148 131 118 113 109 108 92 74 65 72 78]
[89 93 98 97 100 147 131 118 113 114 113 109 106 95 77 80]
[63 77 86 81 77 79 102 123 117 115 117 125 130 115 87]
[62 65 82 89 78 71 88 101 124 126 119 101 107 114 131 119]
[63 65 75 88 88 71 62 81 120 130 130 105 81 98 110 118]
[87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
[118 97 132 88 97 123 116 66 51 57 95 93 89 95 108 107]
[144 146 139 88 98 114 118 76 48 45 66 81 104 102 109]
[157 170 157 120 93 86 114 132 112 87 69 55 78 82 99 104]
[130 128 134 161 130 100 109 118 121 134 114 87 65 53 69 85]
[120 112 96 117 150 144 128 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 121 153 149 122 109 102 75 88 107 112 99]
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 102 61 69 84]
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]



RGB data **tensor**

Data/Input representation: Pixel intensities

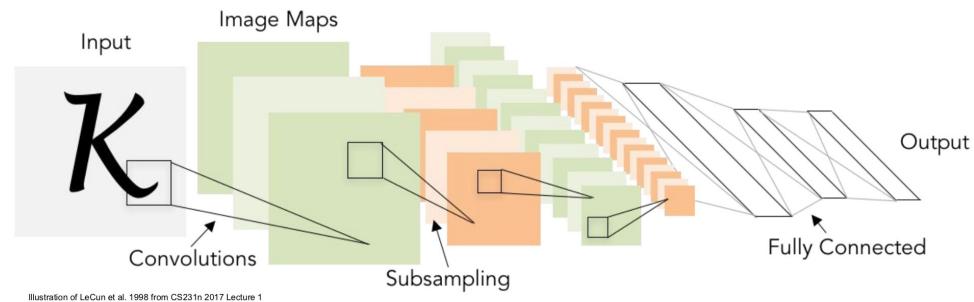


Multi-spectral data

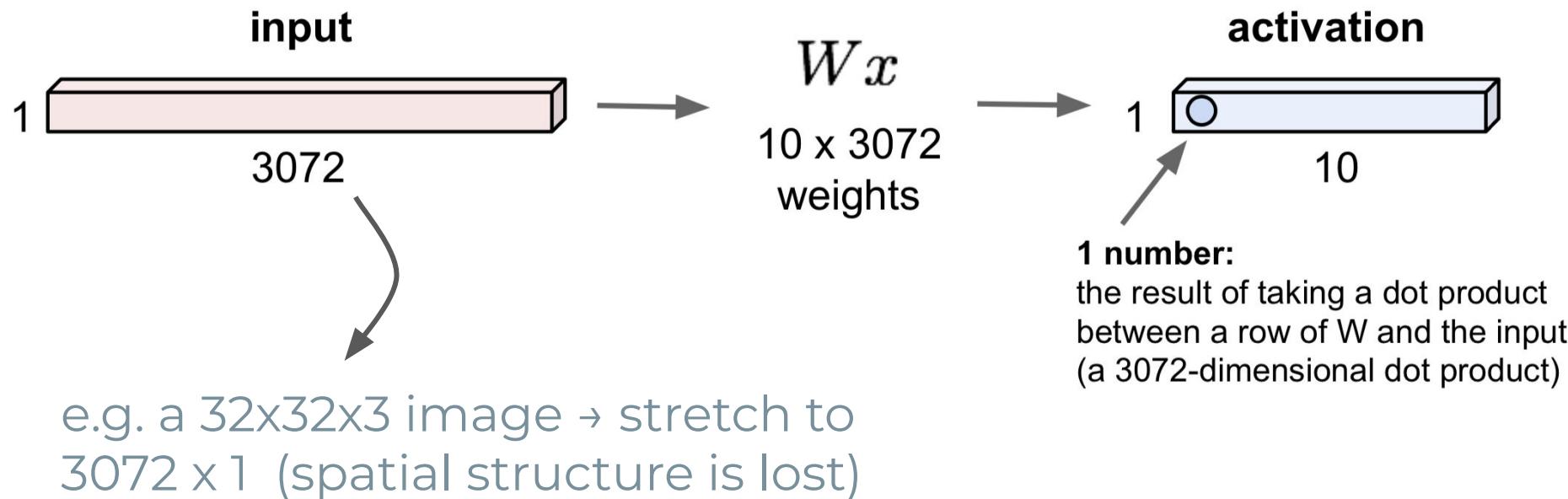
Convolutional Neural Networks (CNNs)

Basic components:

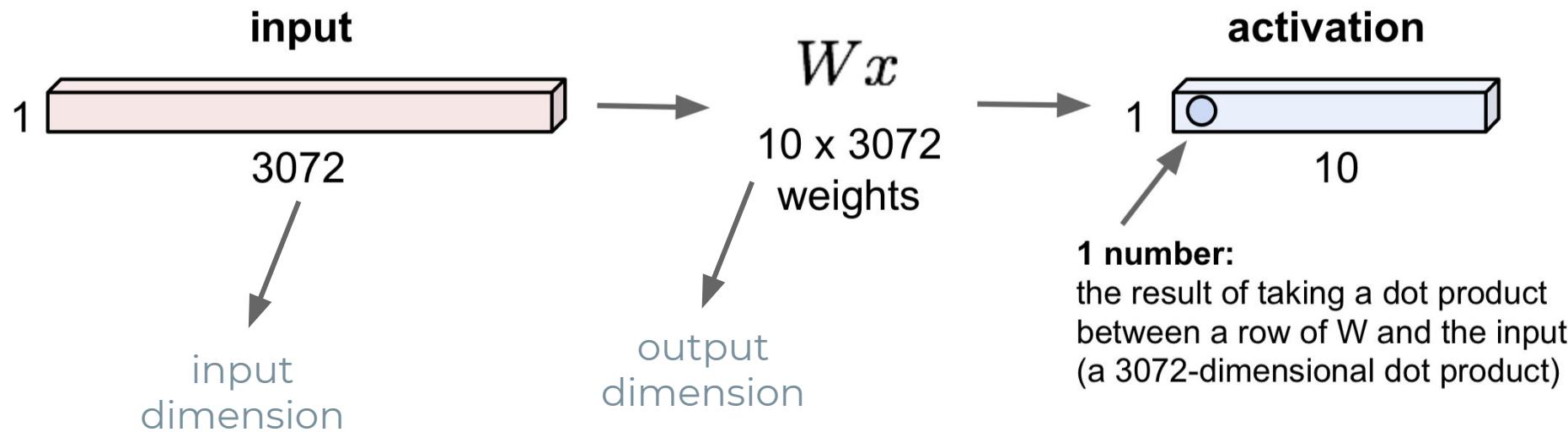
- Fully Connected layer
- Convolutions
- Activation Functions (non-linearities)
- Subsampling/Pooling
- Residual Connections



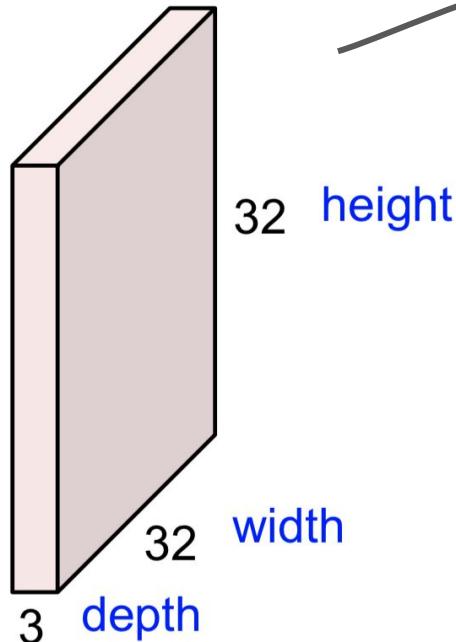
Fully Connected Layer



Fully Connected Layer



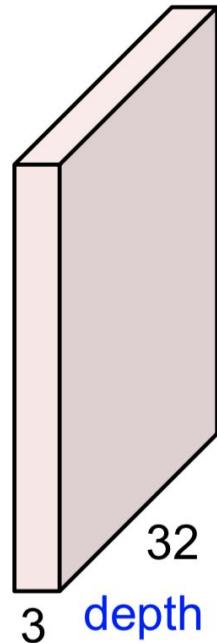
Convolution Layer



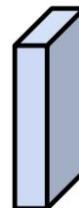
32 x 32 x 3 image

preserve
spatial
structure

Convolution Layer

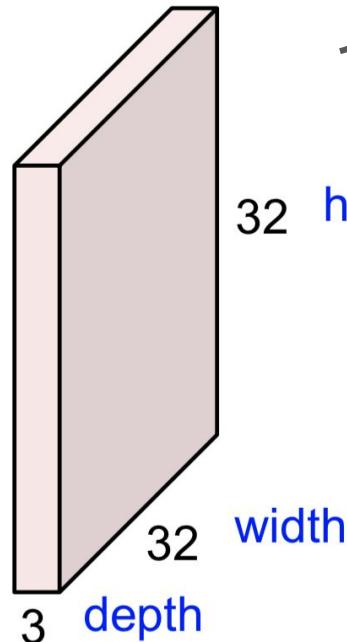


$32 \times 32 \times 3$ image



$5 \times 5 \times 3$ filter

Convolution Layer



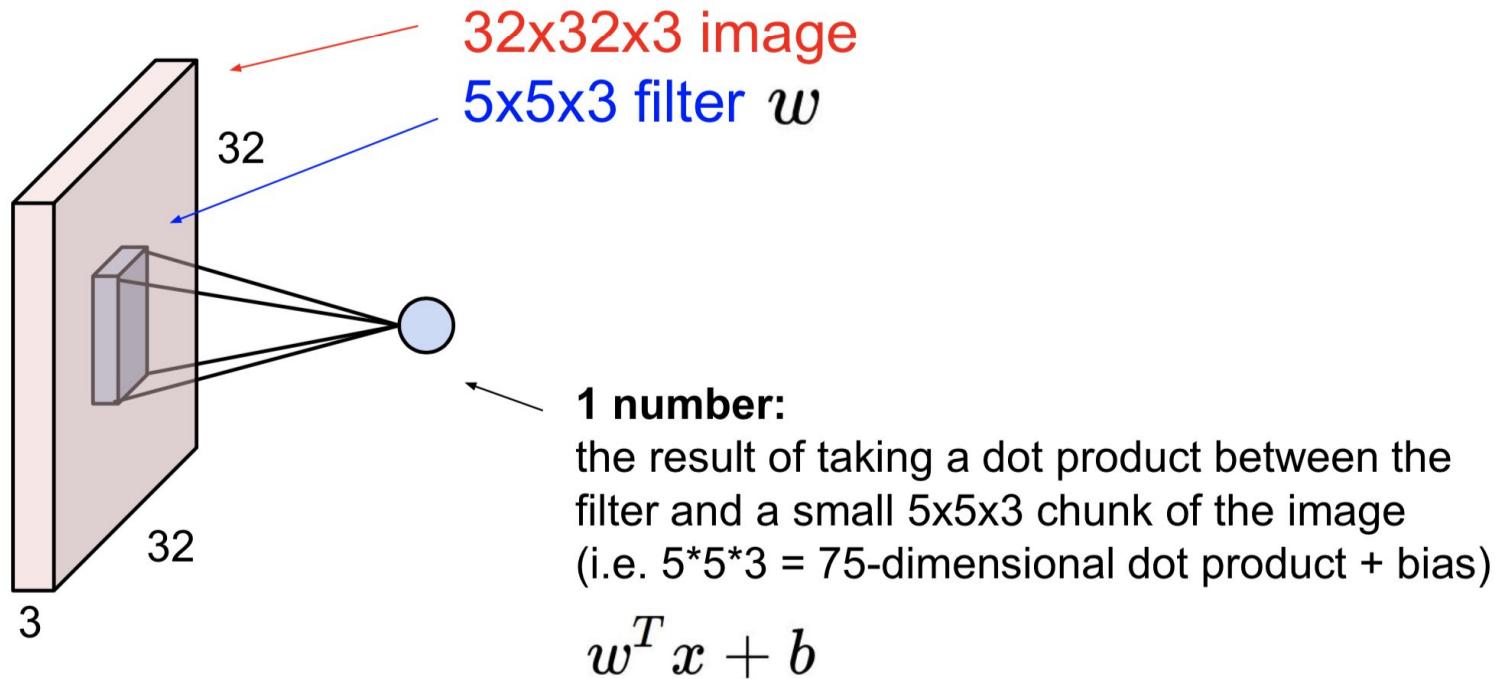
32 x 32 x 3 image



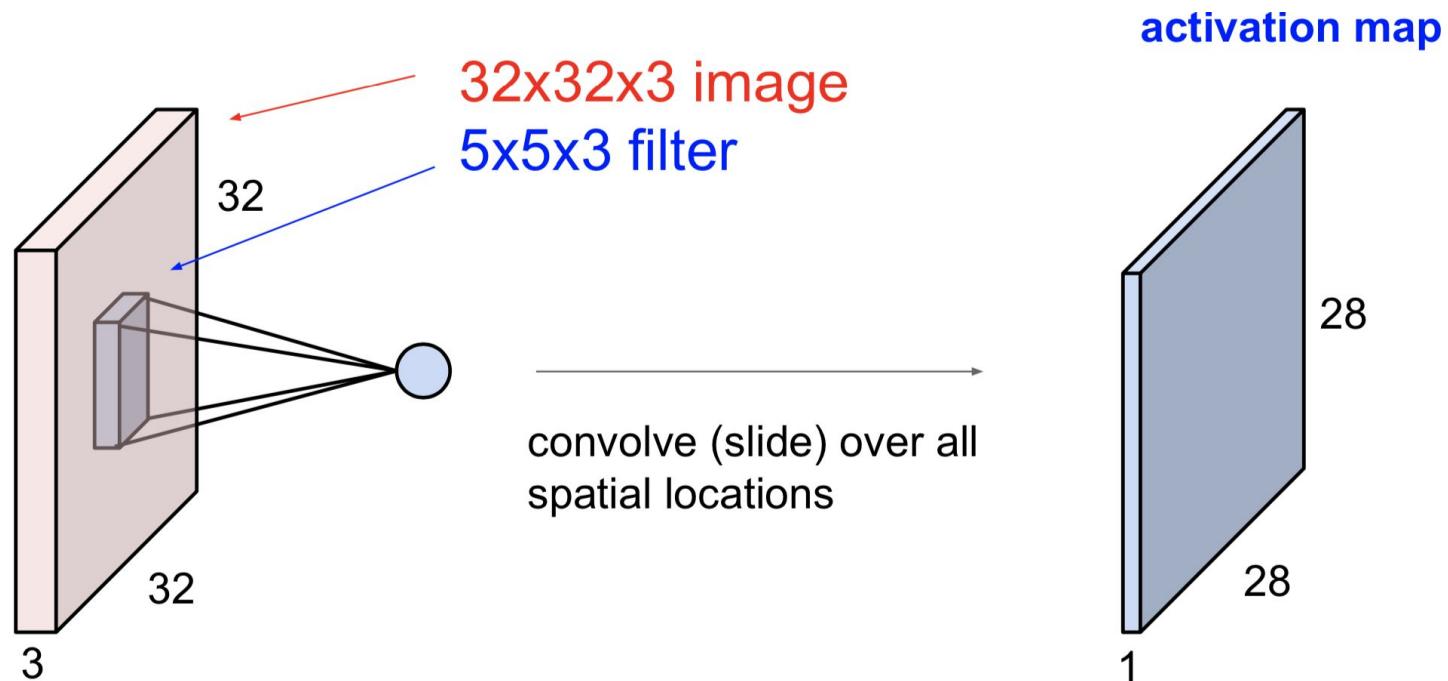
5 x 5 x 3 filter

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

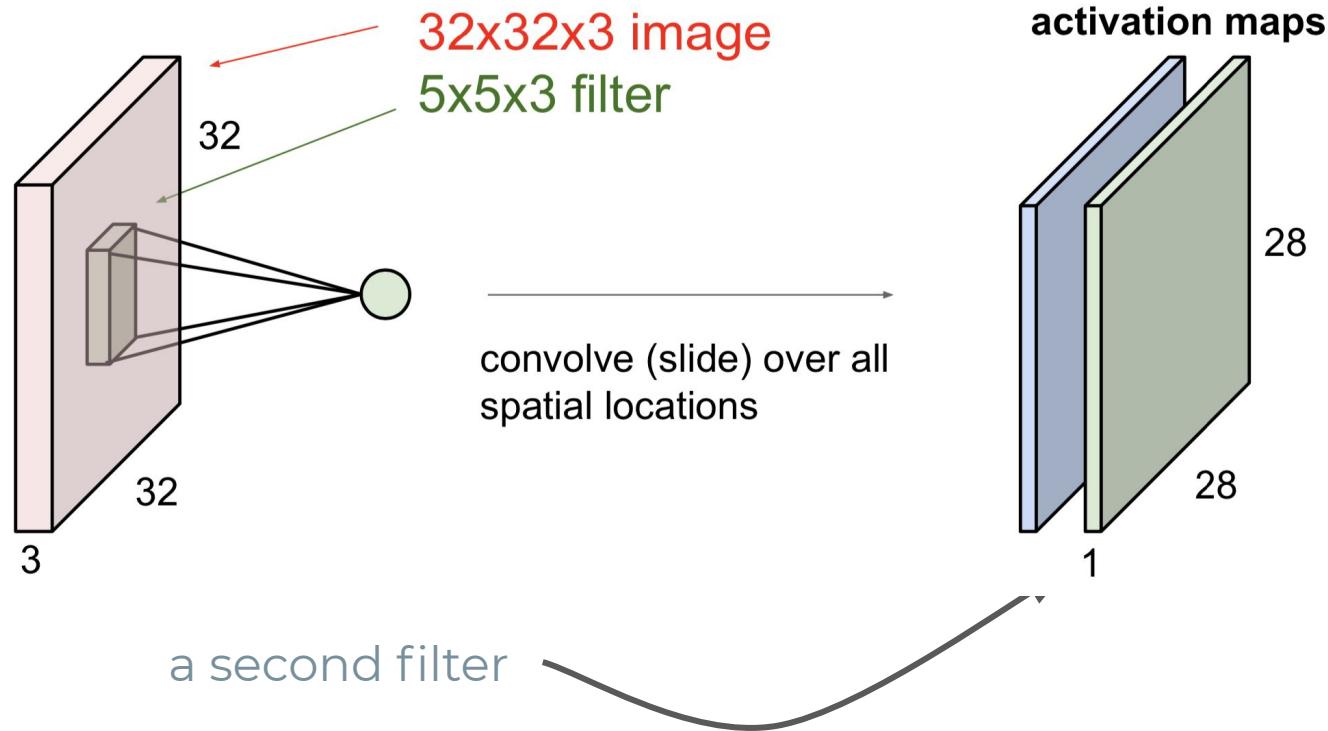
Convolution Layer



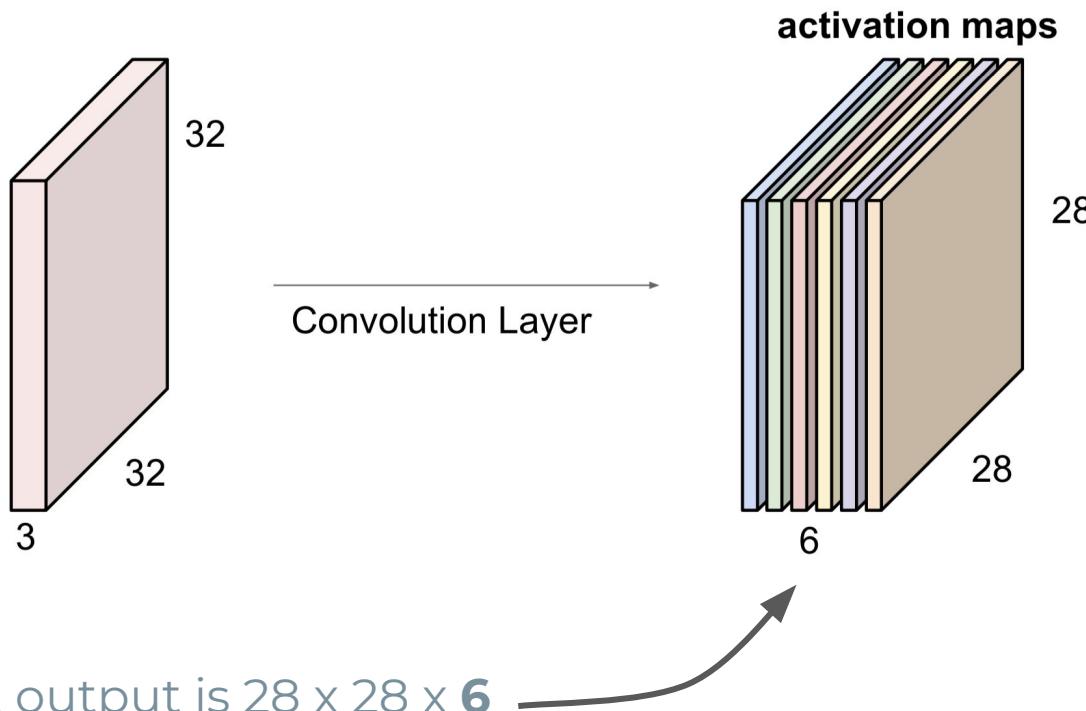
Convolution Layer



Convolution Layer

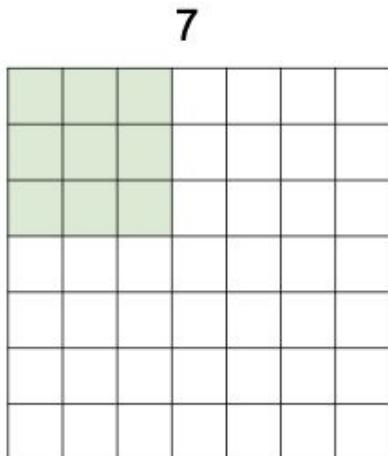


Convolution Layer



Convolution Layer

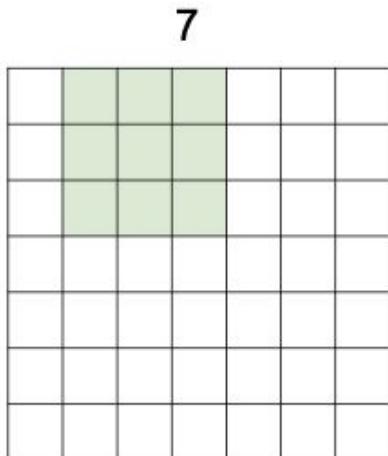
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Layer

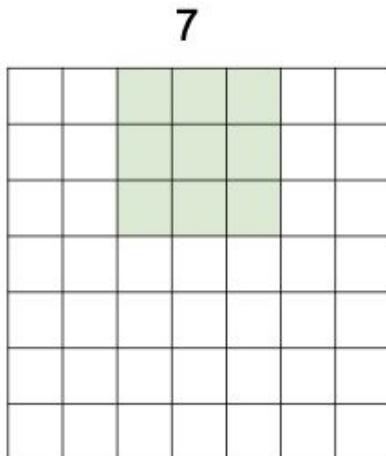
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Layer

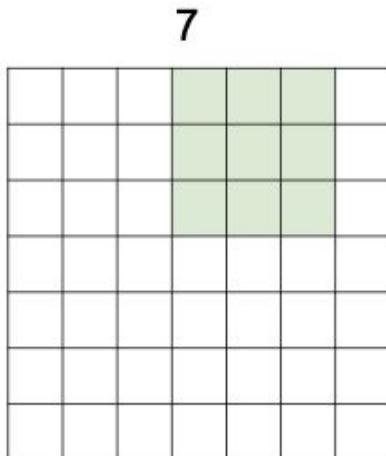
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Layer

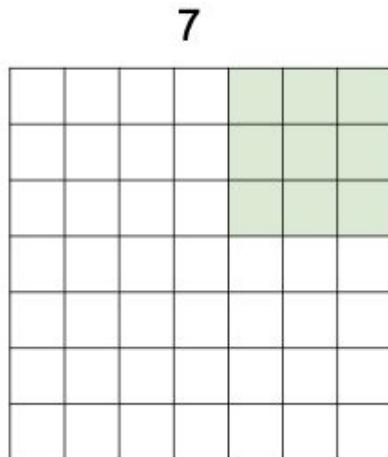
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Layer

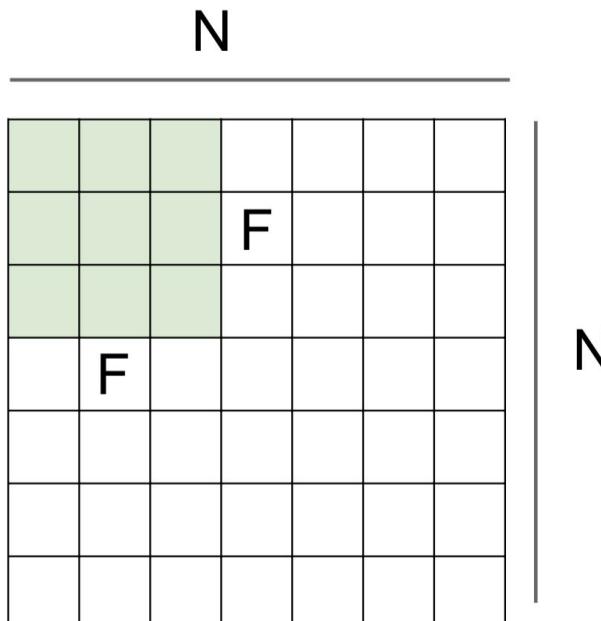
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

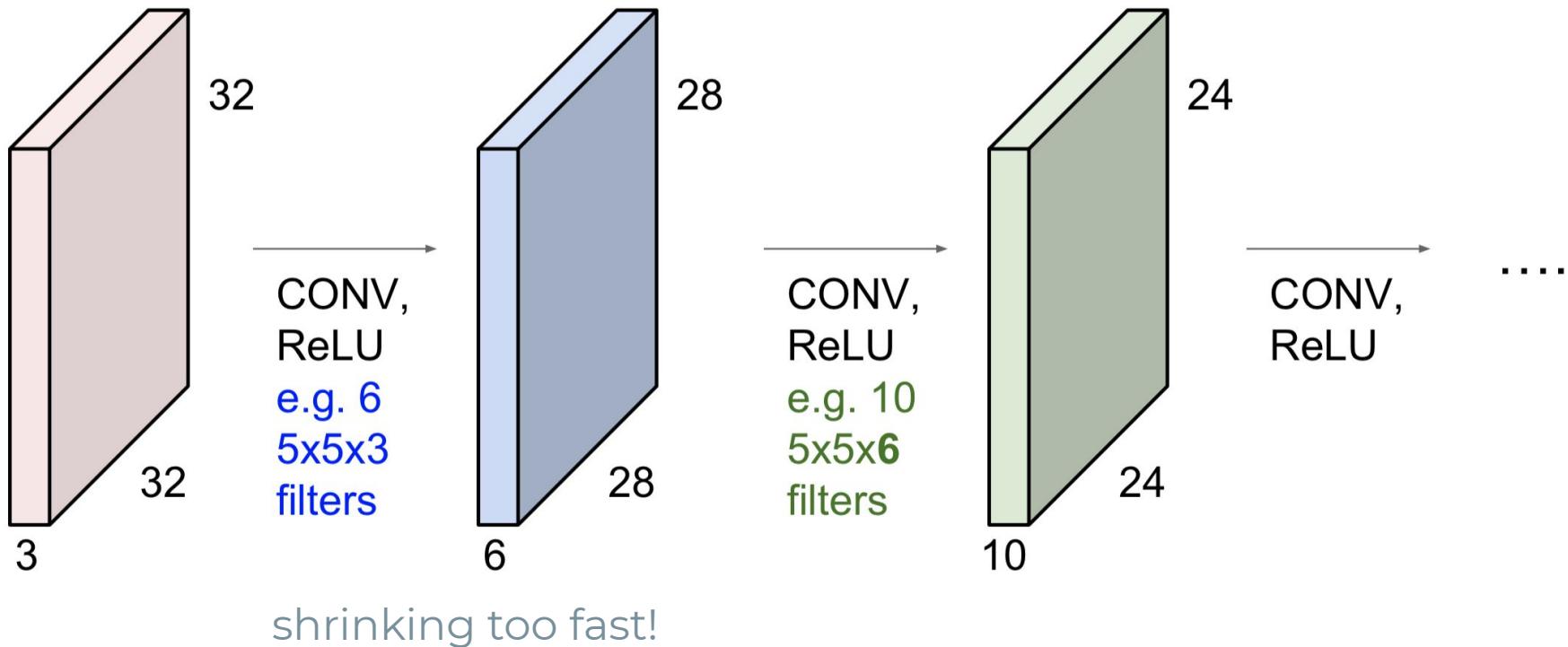
Convolution Layer - stride



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 => $(7 - 3)/3 + 1 = 2.33$:\

Convolution Layer - stride



Convolution Layer - padding

0	0	0	0	0	0		
0							
0							
0							
0							

- input 7x7
- 3x3 filter
- stride = 1
- pad with 1 pixel border

what is the output?

Convolution Layer - padding

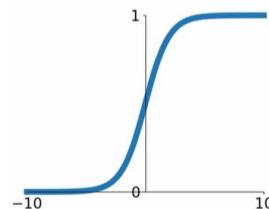
0	0	0	0	0	0			
0								
0								
0								
0								

- input 7x7
- 3x3 filter
- stride = 1
- pad with 1 pixel border
- **7x7 output**
- It is common to see conv layers with stride 1, filters of size FxF, and zero-padding with $(F-1)/2$. (will preserve size spatially)

Activation Function: ReLU

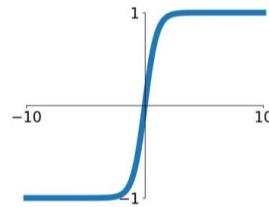
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



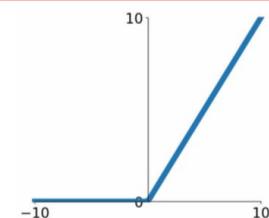
tanh

$$\tanh(x)$$



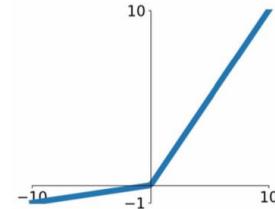
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

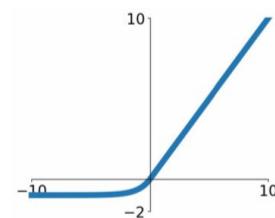


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

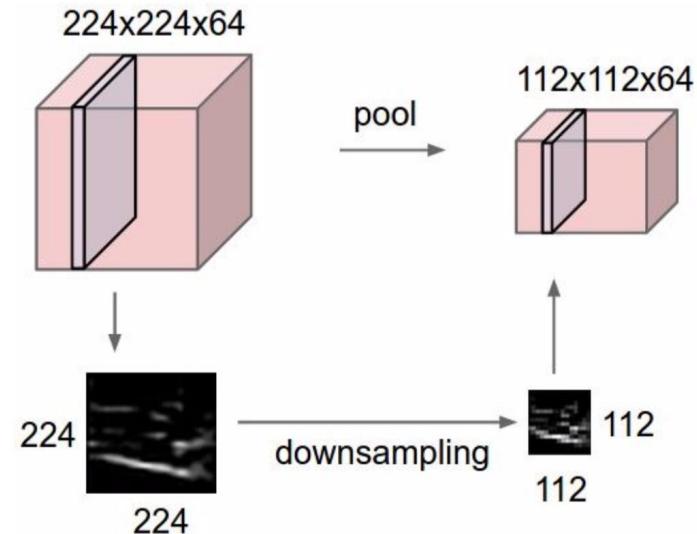
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

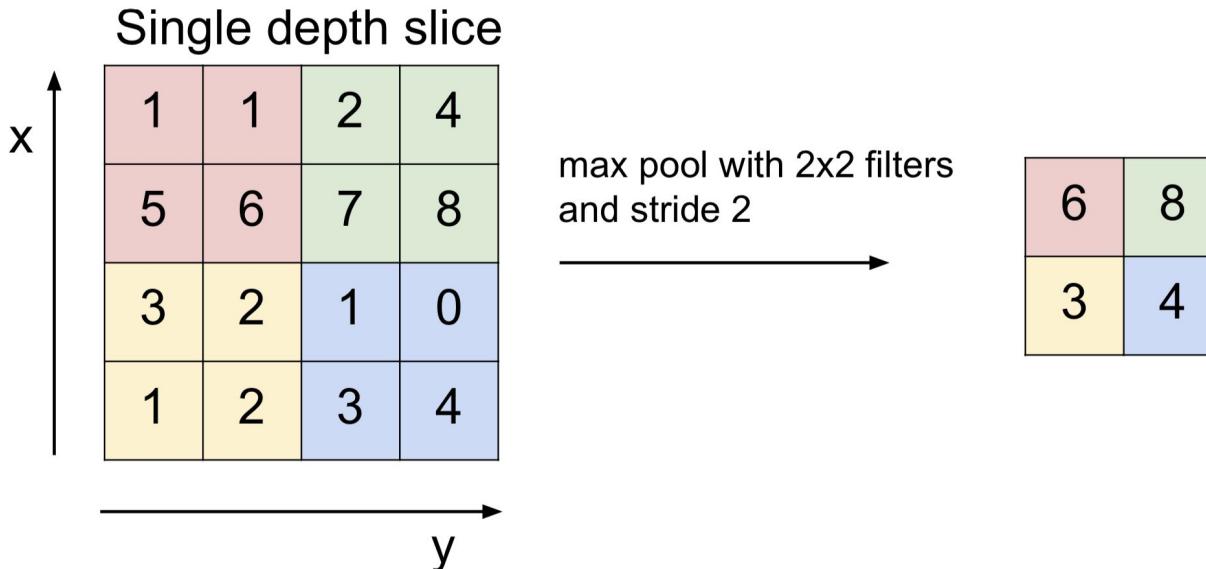


Pooling Layer

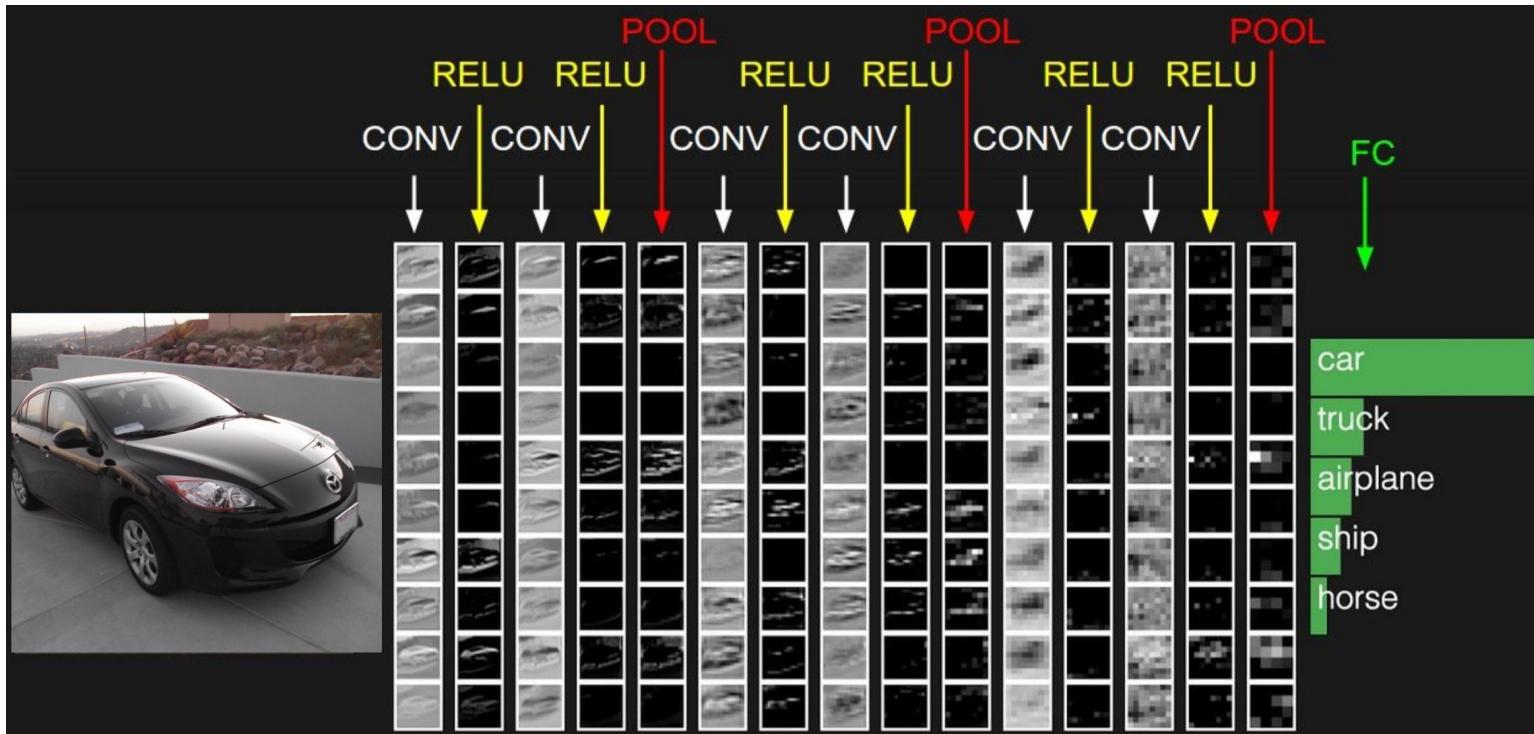
- Subsampling/downsampling
- operates on each activation map independently
- Typical pooling functions:
 - max
 - average



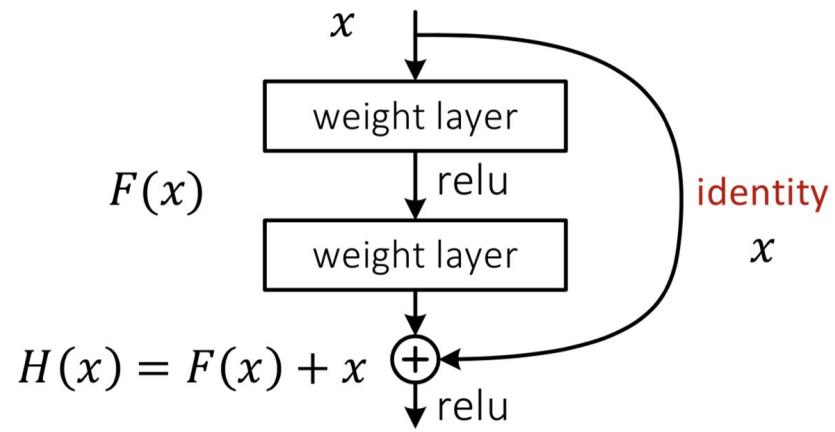
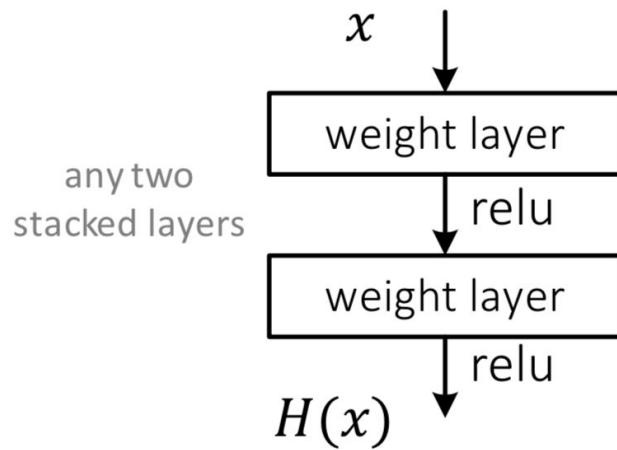
Pooling Layer: Max pooling



Putting it all together



Residual Connections [He et al. 2016]

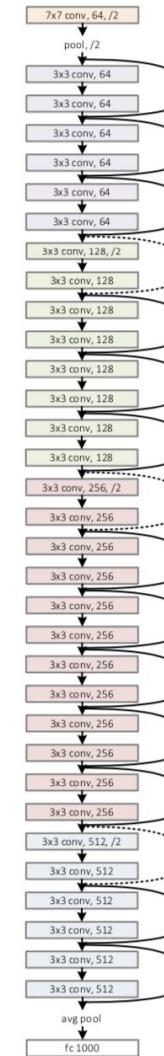
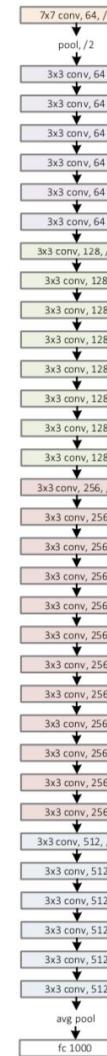


ResNet [He et al. 2016]

Simple but deep design:

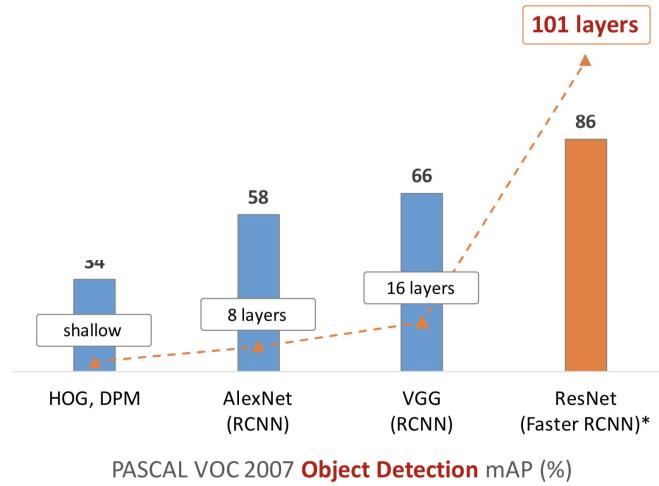
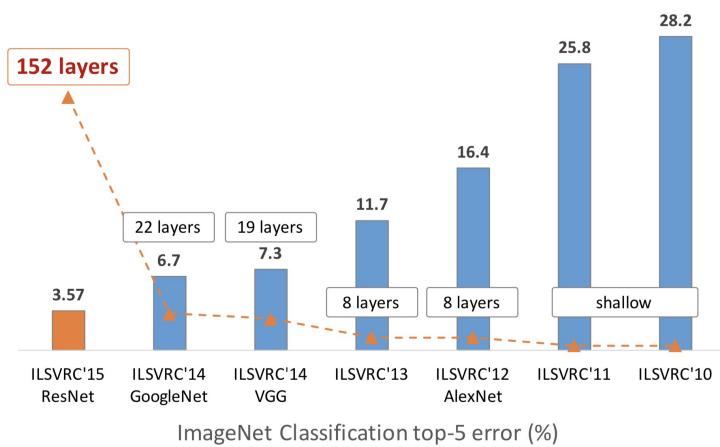
- all 3×3 conv
- spatial size / 2 \Rightarrow # filters $\times 2$
(same complexity per layer)
- Global Average Pooling (GAP)

plain net



ResNet

ResNet [He et al. 2016]



Pytorch example

```
import torch
model = torch.hub.load('pytorch/vision', 'resnet18', pretrained=True)
# or any of these variants
# model = torch.hub.load('pytorch/vision', 'resnet34', pretrained=True)
# model = torch.hub.load('pytorch/vision', 'resnet50', pretrained=True)
# model = torch.hub.load('pytorch/vision', 'resnet101', pretrained=True)
# model = torch.hub.load('pytorch/vision', 'resnet152', pretrained=True)
model.eval()
```

Pytorch example

```
# Download an example image from the pytorch website
import urllib
url, filename = ("https://github.com/pytorch/hub/raw/master/dog.jpg", "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)
```

Pytorch example

```
# sample execution (requires torchvision)
from PIL import Image
from torchvision import transforms
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)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the model
```

Pytorch example

```
# move the input and model to GPU for speed if available
if torch.cuda.is_available():
    input_batch = input_batch.to('cuda')
    model.to('cuda')

with torch.no_grad():
    output = model(input_batch)
# Tensor of shape 1000, with confidence scores over Imagenet's 1000 classes
print(output[0])
# The output has unnormalized scores. To get probabilities, you can run a softmax on it.
print(torch.nn.functional.softmax(output[0], dim=0))
```

Recent advances on (hand-crafted) Convolutional Neural Network architectures (*incomplete and biased list warning)

- [ResNeXt](#) [CVPR 2017]
- [Inception-v4](#) [AAAI 2017]
- [Squeeze-Excitation Nets](#) [CVPR 2018]
- [Non-Local Networks](#) [CVPR 2018]
- [EfficientNet](#) [ICML 2019]
- [Global Reasoning Networks](#) [CVPR 2019]
- [Octave Convolutions](#) [ICCV 2019]

all the approaches above come with open-source code and models

Recent advances in CNN architectures

Neural Architecture Search

- [AutoML NeurIPS 2018 Tutorial](#) [U. Freiburg, U. Eindhoven]
- [Neural Architecture Search with Reinforcement Learning](#) [Google]
- [NAS state-of-the-art overview](#) [Microsoft]

Recent advances on visual representation learning architectures

(*incomplete and biased list warning)

- Transformer-based models (2020-2021)
 - [ViT](#), [T2t-ViT](#), [BiT](#), [DeiT](#), [LeViT](#), and many more
- MLP-based models (May(!) 2021)
 - eg [MLP-Mixer](#) and ...2-3 more the last week

Again, all approaches above come with open-source code and models

Overview

- Introduction
 - What is a “representation”?
 - Extracting vs. Learning Representations
- Convolutional Neural Networks (CNNs)
 - Basic components and architectures
 - Pytorch example
- **Training Convolutional Neural Networks**
 - **Loss function and regularization**
 - **Important tips for training image models**
- Fine-grained recognition
 - Best practices for fine-grained Recognition
 - Tackling small and imbalanced datasets

How to train a CNN model?

(Mini-batch) Stochastic Gradient Descent (SGD)

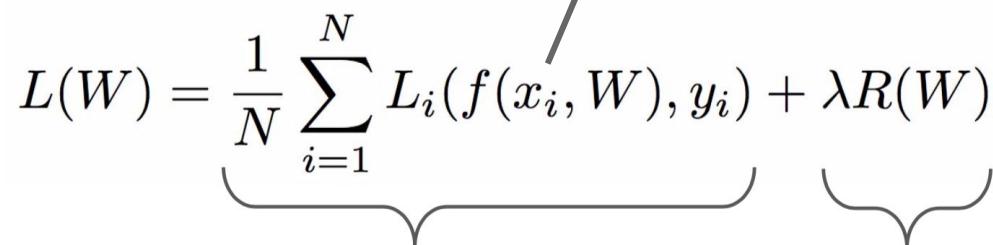
Loop:

1. Sample a batch of data
2. Forward prop it through the model
3. Calculate loss function
4. Backprop to calculate the gradients
5. Update the parameters using the gradient

How to train a CNN model?

Loss function

- Cross entropy loss

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$


Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

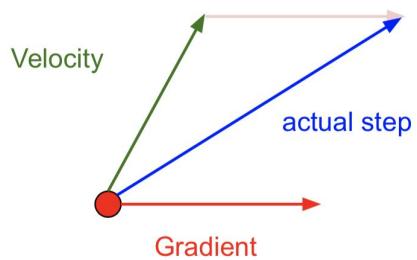
How to train a CNN model?

(most commonly used) **Optimizer**

- Stochastic Gradient Descent (SGD) with momentum

$$v_t = \mu v_{t-1} + (1 - \mu) \nabla_W L$$

$$W' = W - v_t$$



Combine gradient at current point with velocity to get step used to update weights

How to train a CNN model?

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

a.k.a. Weight decay

(see also [\[Zhang et al. ICLR 2019\]](#))

More complex:

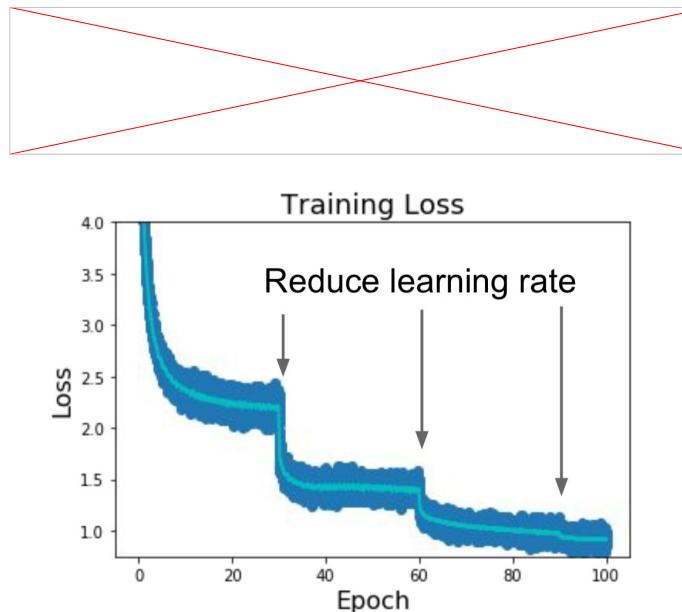
Dropout

Batch normalization

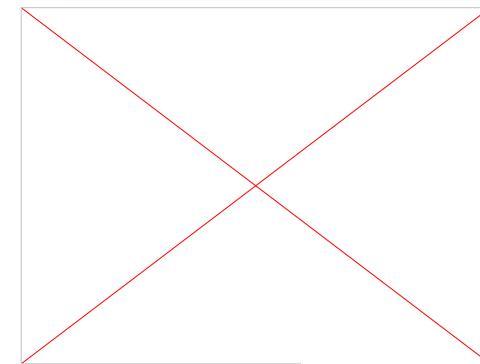
Stochastic depth, fractional pooling, etc

Learning Rate (LR)

- Which LR to use? a) Start large and decay; b) use warm-up.



Cosine: $\alpha_t = \frac{1}{2}\alpha_0 (1 + \cos(t\pi/T))$

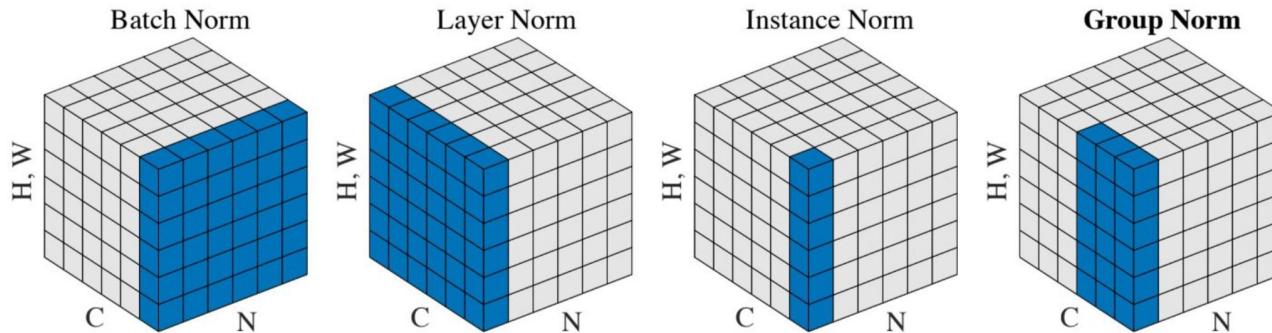


α_0 : Initial learning rate
 α_t : Learning rate at epoch t
 T : Total number of epochs

Very important training tips

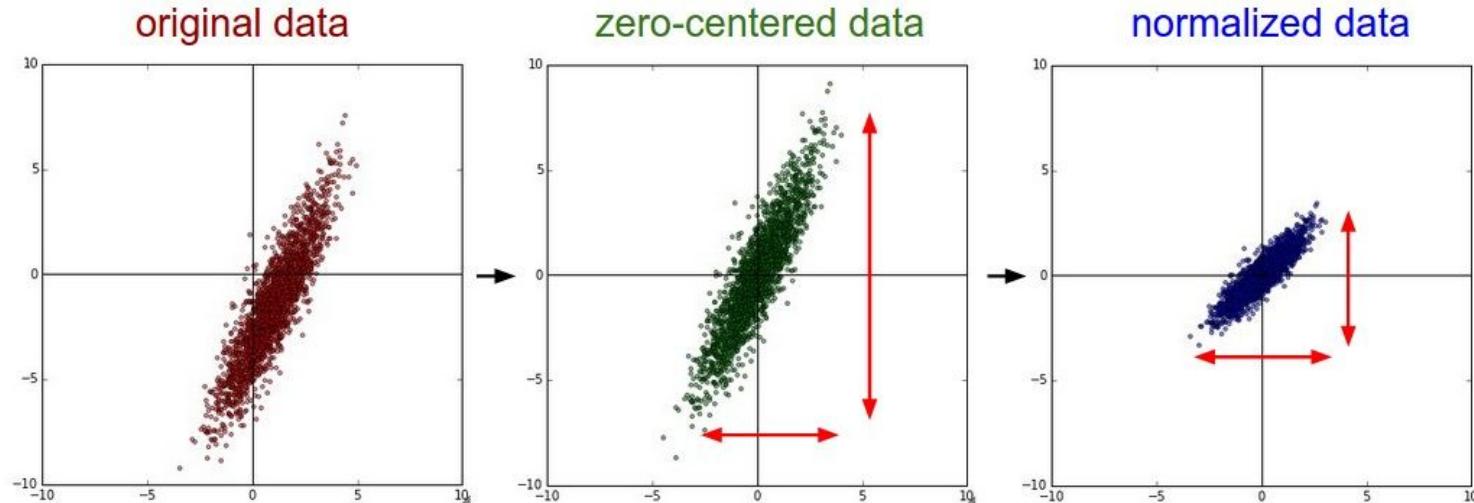
- Batch Normalization [\[Ioffe & Szegedy 2015\]](#)
 - make each dimension zero-mean unit-variance
 - also [LayerNorm](#), [GroupNorm](#), and others

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



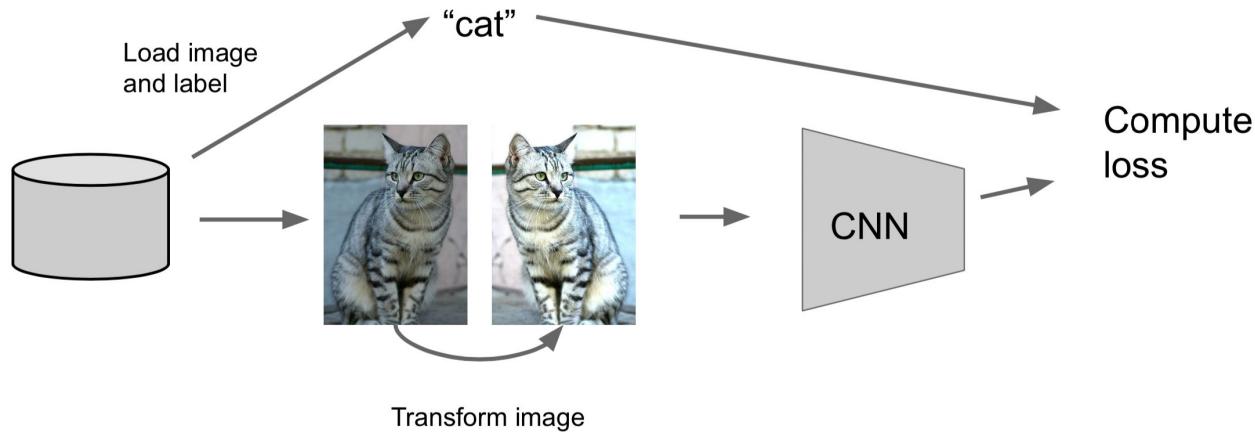
Very important training tips

- Data preprocessing
 - Subtract per-channel mean and divide by per-channel std dev.



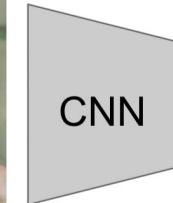
Very important training tips

- Data preprocessing
 - Subtract per-channel mean and divide by per-channel std dev.
- Data Augmentation



Very important training tips

- Data preprocessing
 - Subtract per-channel mean and divide by per-channel std dev.
- Data Augmentation
 - [Auto Augment](#)
 - [Mixup](#)
 - Training: Train on random blends of images
 - Testing: Use Original images



Target label:
cat: 0.4
dog: 0.6

Randomly blend the pixels
of pairs of training images,
e.g. 40% cat, 60% dog

Very important training tips

- Data preprocessing
 - Subtract per-channel mean and divide by per-channel std dev.
- Data Augmentation
 - [Auto Augment](#)
 - [Mixup](#)
- Weight initialization
 - [MSRA init](#) (for ReLU nets): $\text{rand} * \sqrt{2 / d_{\text{in}}}$
 - [Lottery ticket hypothesis](#) [ICLR 2018]
 - [Deconstructing Lottery Ticket Hypothesis](#) [ICLR 2019]

Overview

- Introduction
 - What is a “representation”?
 - Extracting vs. Learning Representations
- Convolutional Neural Networks (CNNs)
 - Basic components and architectures
 - Pytorch example
- Training Convolutional Neural Networks
 - Loss function and regularization
 - Important tips for training image models
- **Fine-grained recognition**
 - **Best practices for fine-grained Recognition**
 - **Tackling small and imbalanced datasets**

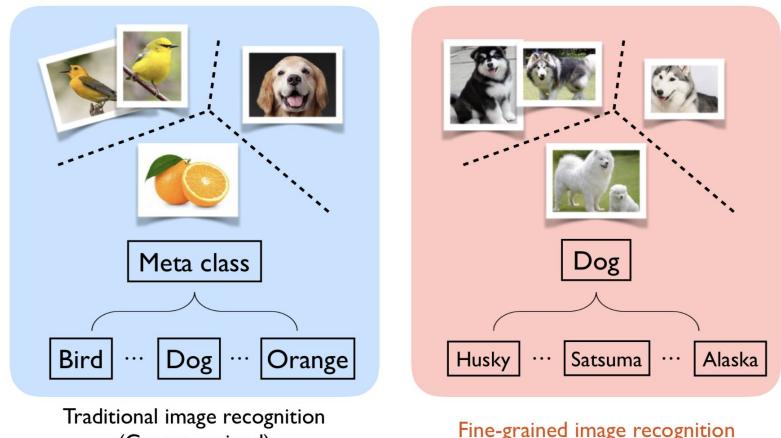
Fine-grained Recognition

Image Classification + Challenges

- Small inter-class variance
- Large intra-class variance

Plus possibly:

- Fewer data overall per class
- Large number of classes
- Imbalanced data per class



8th Fine-Grained Visual Categorization (FGVC) Workshop at CVPR 2021

More realistic competitions:

- iNaturalist challenge
- fashion
- products
- wildlife camera traps
- cultural items

Fine-Grained Recognition in two steps

- 1) Start from a state-of-the-art (possibly pre-trained) model
- 2) Fine-tune depending on amount of available data & compute

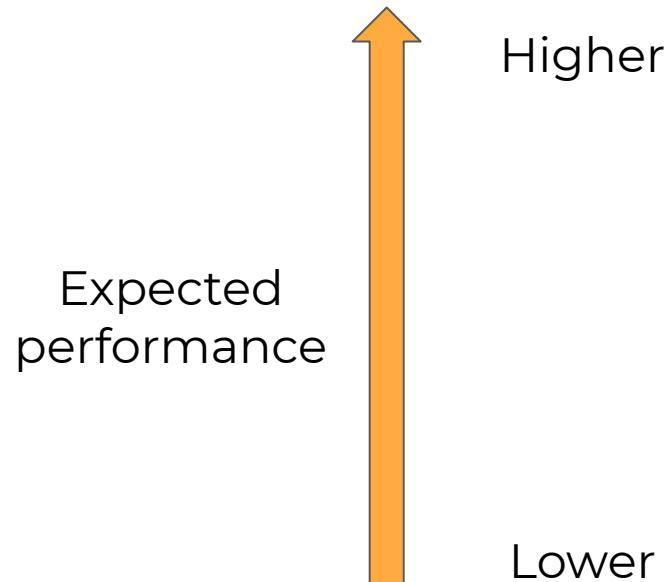
Note: Lots of papers are lately proposing architectures, regularizations and tweaks specific for fine-grained recognition; the above recipe, however, if training is done “the right way”, can empirically give results almost as good as the best of those.

Start from a state-of-the-art model

- Pick one of the best models wrt your resources
 - small: Mobilenet, ShuffleNet, EfficientNet, etc
 - medium: (SE-)/(Oct-)ResNe(X)t50, etc
 - large: SENet-154, Inception-v4, (SE-)/(Oct-)ResNe(X)t-152, etc
- Start from a model pre-trained on a large dataset
 - Pre-train dataset as close to the target domain as possible [\[Ciu et al CVPR 2018\]](#)
 - Lots of publicly available models!
 - Check the github pages of the latest architectures
 - Models after training from 1 Billion images from FB

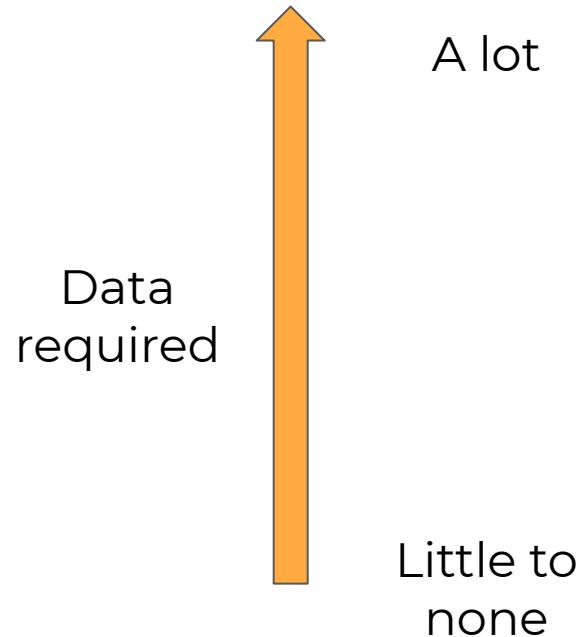
Pre-trained vs train from scratch

- **Train** a model *from scratch* with the data
- **Fine-tune** a *pre-trained* model
- Utilize representations learned from a pre-trained model



Pre-trained vs train from scratch

- **Train** a model *from scratch* with the data
- **Fine-tune** a *pre-trained* model
- Utilize representations learned from a pre-trained model



Fine-tune the model

How much data/computing power do you have?

- Lots
 - Consider training from scratch
 - Fine-tune the full model with a lower learning rate
- Moderate
 - Fine-tune the last few layers of the model with a lower learning rate
- Small
 - Train only the classifier
 - consider a 1-NN classifier! (surprisingly competitive, no training needed)

Best practices for fine-grained recognition

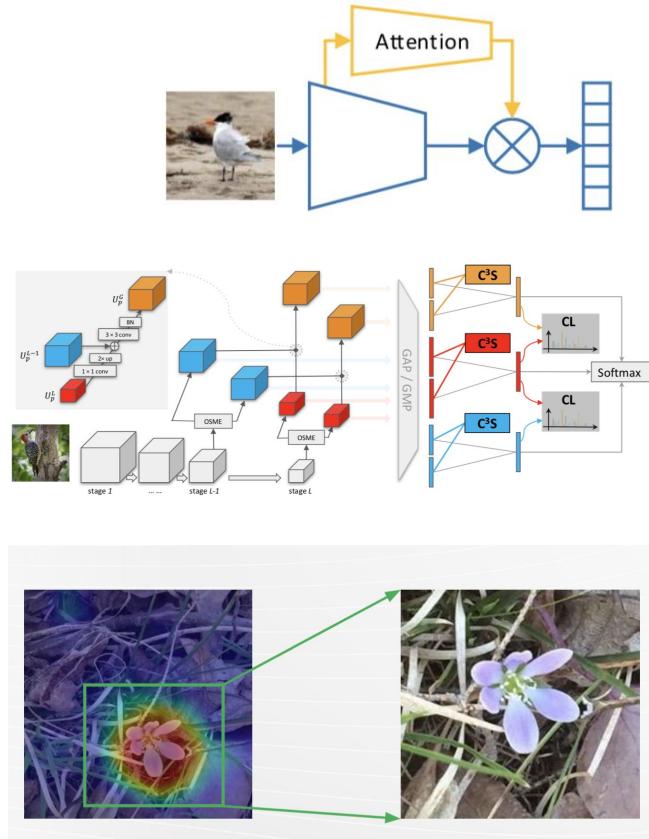
- Utilize all the “general” best practices
 - Data Preprocessing
 - Data Augmentation
 - Carefully tune Weight decay, Learning Rate and schedule
 - Also possibly helpful: Label smoothing, Test-time augmentation
- Utilize any extra domain knowledge (eg part annotations)
- Utilize unlabeled data from the target domain if available (semi-supervised learning)

Dealing with small datasets or small inter-class variance

- Data augmentation is highly important
 - auto augment, mixup, manifold-mixup, generate/hallucinate new data,
- Feature normalization is highly important
 - try L2-norm, centering, PCA
- Test-time augmentations
 - multiple crops
 - Multi-resolution testing, model ensembles
- Train and test image resolution is very important
 - [\[Cui et al. CVPR 2018\]](#), [\[Touvron et al 2019\]](#)

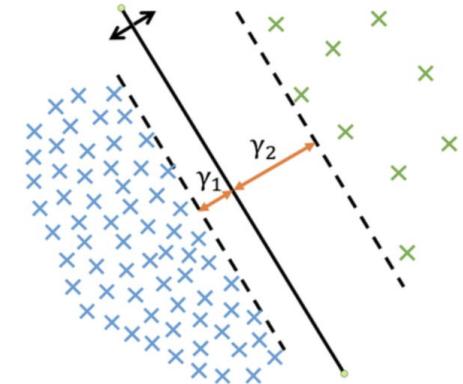
Dealing with small datasets or small inter-class variance

- Weakly Supervised Localization with CAM
 - [\[Zhou et al. CVPR 2016\]](#)
- Attention-based architectures
 - [\[Fu et al. CVPR 2017\]](#), [\[Zheng et al. CVPR 2019\]](#)
 - simplest case: post-hoc add and learn an attention layer before the global average pooling
- GAP → Generalized mean pooling (GeM)
 - [\[Radenovic et al PAMI 2018\]](#)
- Regularizers for multi-scale learning
 - [\[Luo et al. ICCV 2019\]](#)



Dealing with imbalanced data ("long-tailed recognition")

- Label-Distribution-Aware Margin Loss
[\[Cao et al., NeurIPS 2019\]](#)
- Class-balanced loss [\[Cui et al. CVPR 2019\]](#)
- Decouple representation from classifier learning [Kang et al. ICLR 2020]
 - Learn representation without caring about imbalance, fine-tune using uniform sampling
 - ...or just re-balance the classifier



Method	ResNet-50	ResNet-152
CB-Focal†	61.1	-
LDAM†	64.6	-
LDAM+DRW†	68.0	-
Joint	61.7/65.8	65.0/69.0
NCM	58.2/63.1	61.9/67.3
cRT	65.2/67.6	68.5/71.2
τ -normalized	65.6/ 69.3	68.8/ 72.5

results on iNaturalist 2018
(8k species, long-tail)

Summary

- Introduction
 - What is a “representation”?
 - Extracting vs. Learning Representations
- Convolutional Neural Networks (CNNs)
 - Basic components and architectures
 - Pytorch example
- Training Convolutional Neural Networks
 - Loss function and regularization
 - Important tips for training image models
- Fine-grained recognition
 - Best practices for fine-grained Recognition
 - Tackling small and imbalanced datasets

Resources

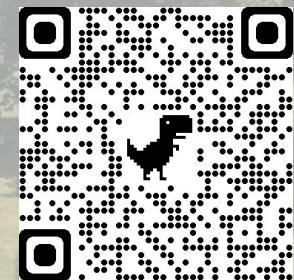
- All pytorch tutorials: <https://pytorch.org/tutorials/>
- Tutorials on [image classification](#), [transfer learning](#) and [fine-tuning](#)
- Pre-trained models to start from:
 - [ImageNet + iNaturalist pre-trained models](#) ([tensorflow](#)) [Ciu et al CVPR 2018]
 - Models trained on ~1 Billion images from IG hashtags from Facebook:
 - [WSL-Images models](#) ([pytorch](#)) [\[Mahajan et al CVPR 2018\]](#)
 - [SSL/SWSL models](#) ([pytorch](#)) **new!** [\[Yalniz et al 2019\]](#)
- Great resource for going deeper (with video lectures): [Stanford CS231n](#)



Thank you!

Questions?

Slides:



<https://europe.naverlabs.com>