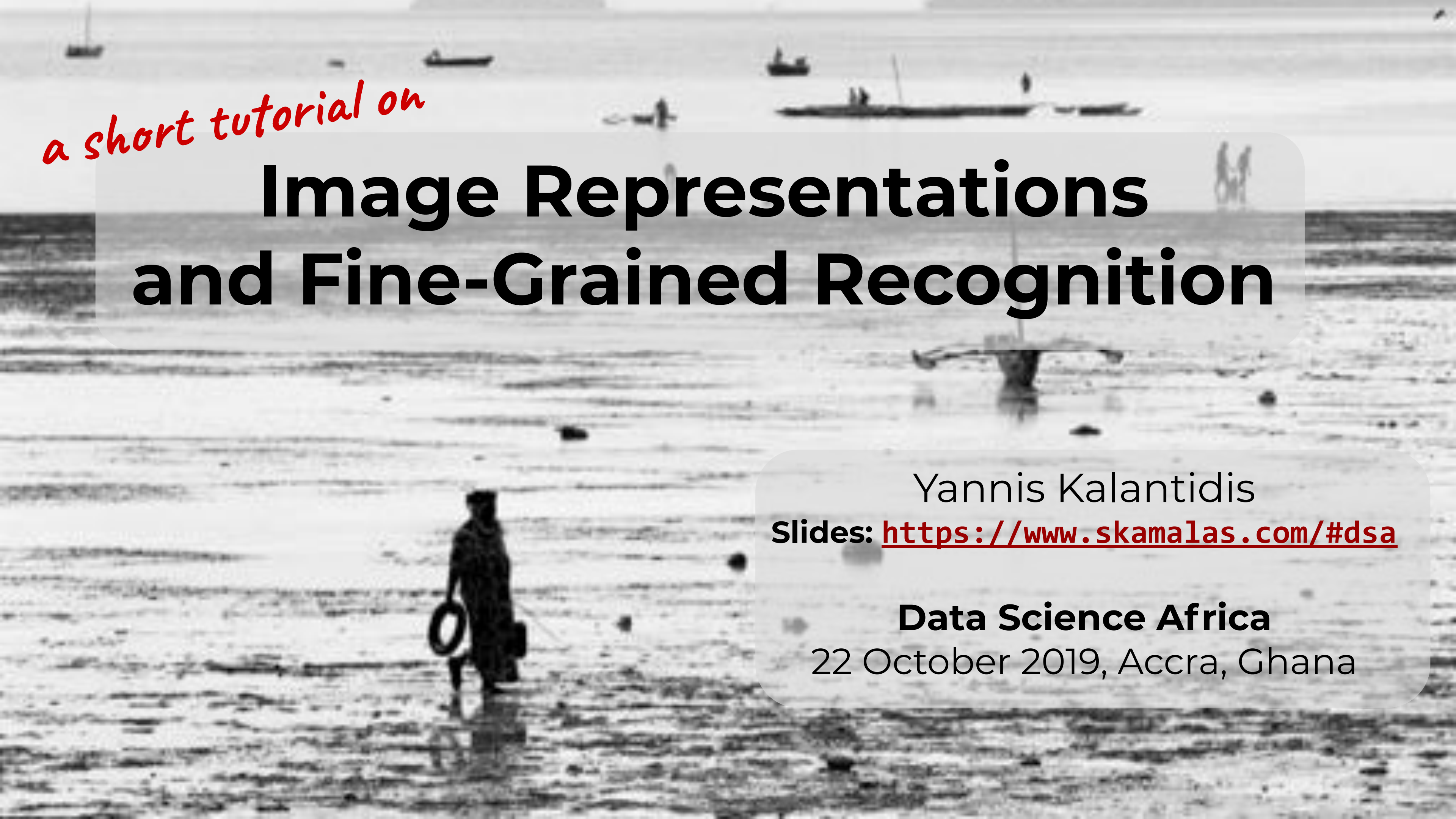


a short tutorial on

Image Representations and Fine-Grained Recognition

Yannis Kalantidis

Slides: https://www.skamalas.com/#dsa

Data Science Africa

22 October 2019, Accra, Ghana

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"?

ICLR | 2020

Eighth International Conference on Learning Representations

ICLR 2020:

The first major ML conference to take place in Africa (Addis Ababa, Ethiopia, April 2020) during the last decades

What is a “representation”?

“Representation” or ”feature” in Machine Learning:

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

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

What is an image/visual 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/visual 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 $x_i, x_j \in \mathbb{R}^d$

$$d(x_i, x_j) = \|x_i - x_j\|^2$$

$$= \sum_{k=1}^{d} (x_i^k - x_j^k)^2$$

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
- Datasets: ImageNet (meh) ...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 2019

iCassava 2019 Fine-Grained Visual Categorization Challenge

Ernest Mwebaze, Timnit Gebru, Andrea Frome
Google Research
[emwebaze,tgebru, afrome]@google.com

Solomon Nsumba, Jeremy Tusubira
Artificial Intelligence lab
Makerere University
[snsumba, jtusubira]@gmail.com

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

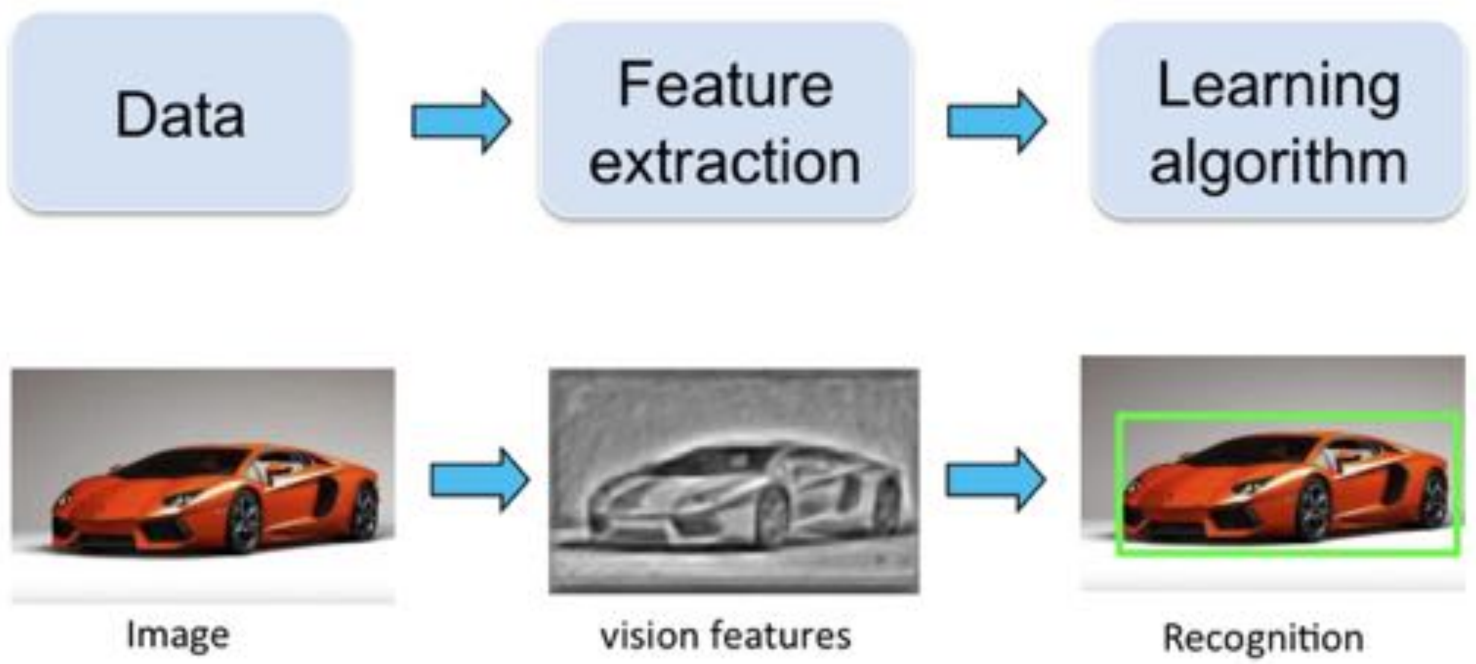
(a) Healthy (b) CBB (c) CGM (d) CMD (e) CBSD

(a) Background effects (b) Time of day effects

(c) Multiple disease (d) Poor focus effects

Paper: https://arxiv.org/pdf/1908.02900.pdf

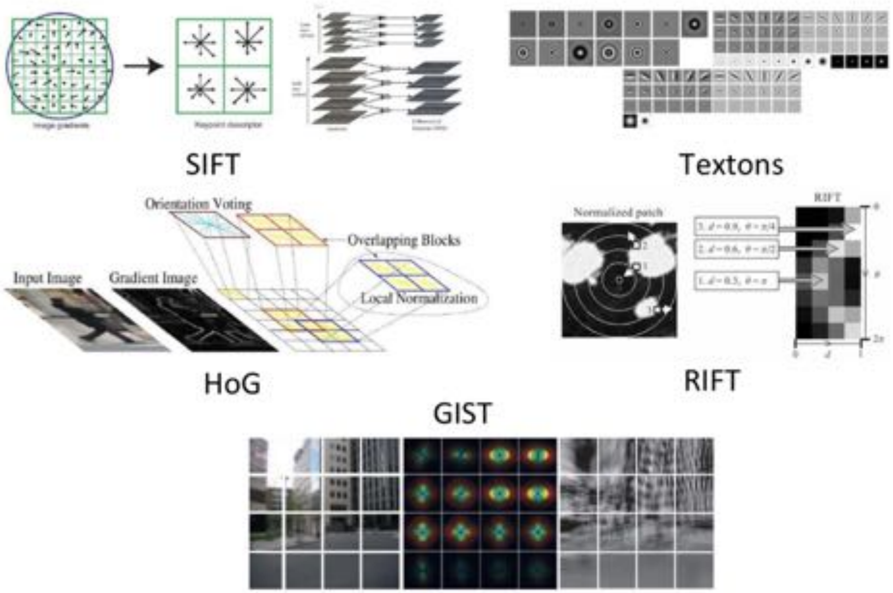
Extracting vs. Learning Representations



Extracting vs. Learning Representations

Feature Extraction:
“hand-crafted” representations

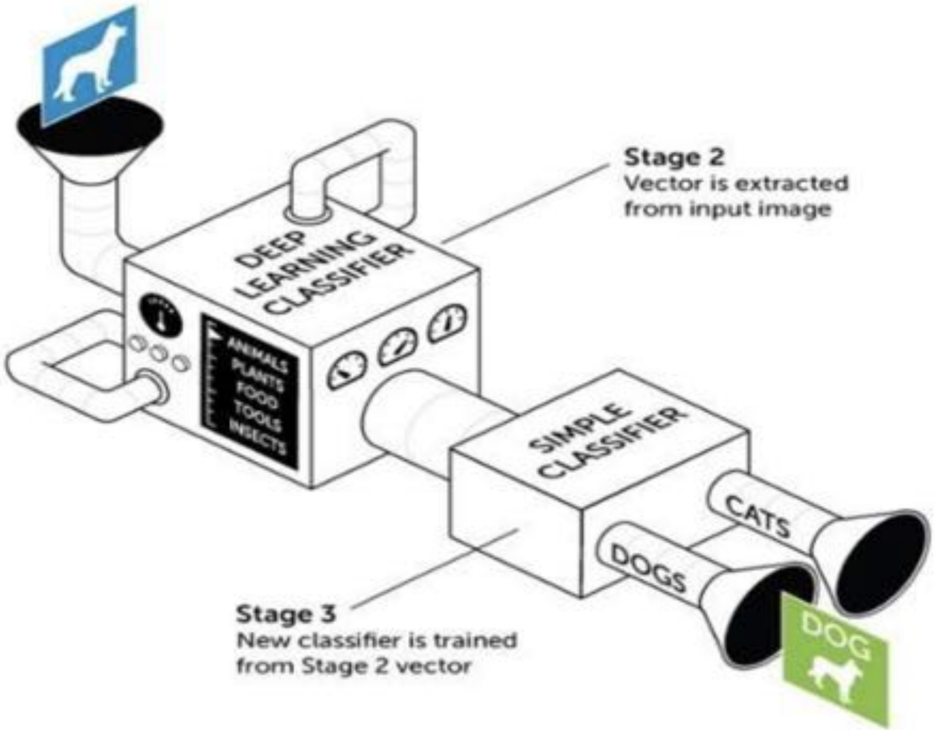
- Utilizes domain knowledge
- Requires domain expertise
- Most common approach for decades



Extracting vs. Learning Representations

Representation Learning

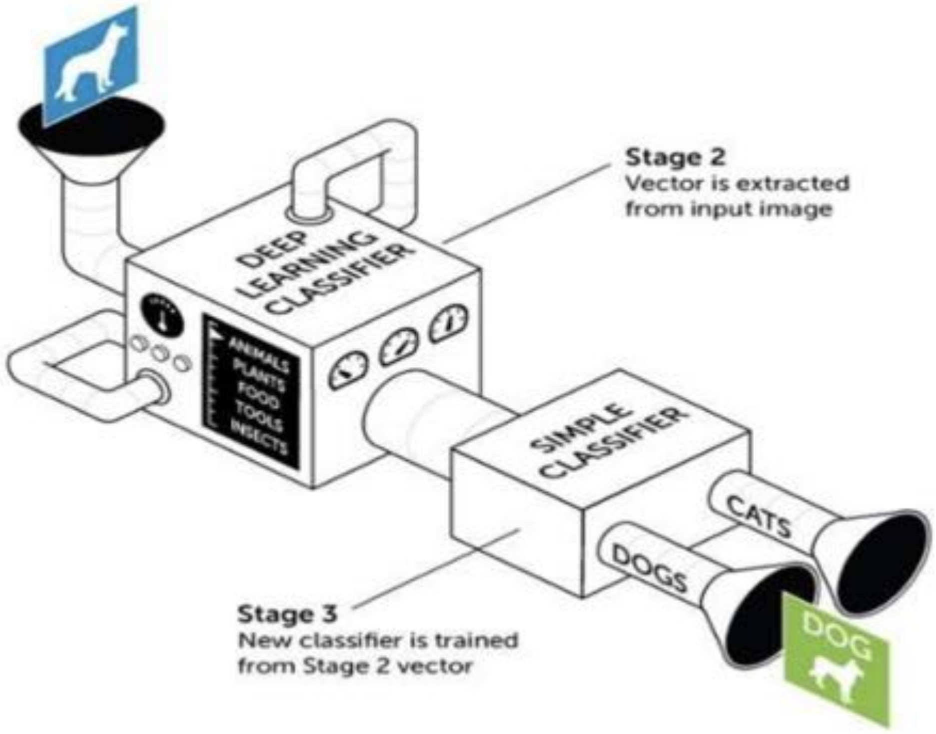
- Don't design features
- Design ***models*** that output representations and predictions
- Don't tell the model how to solve your task; tell the model what result you want to get



Extracting vs. Learning Representations

Representation Learning

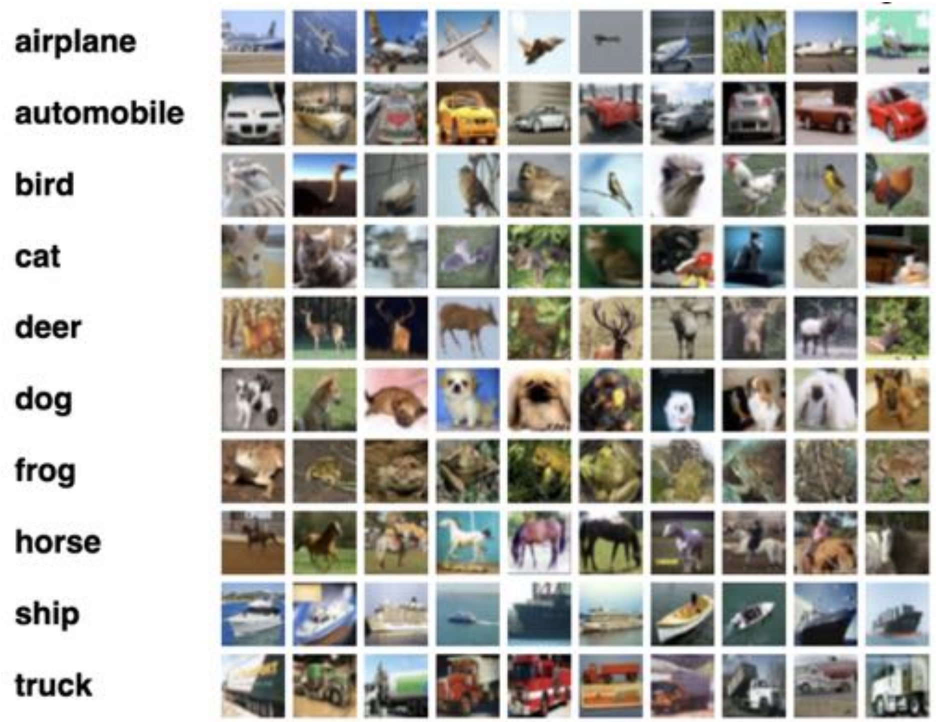
1) Collect a dataset of images and labels
2) Use machine learning to train a model and classifier
3) Evaluate on new images



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

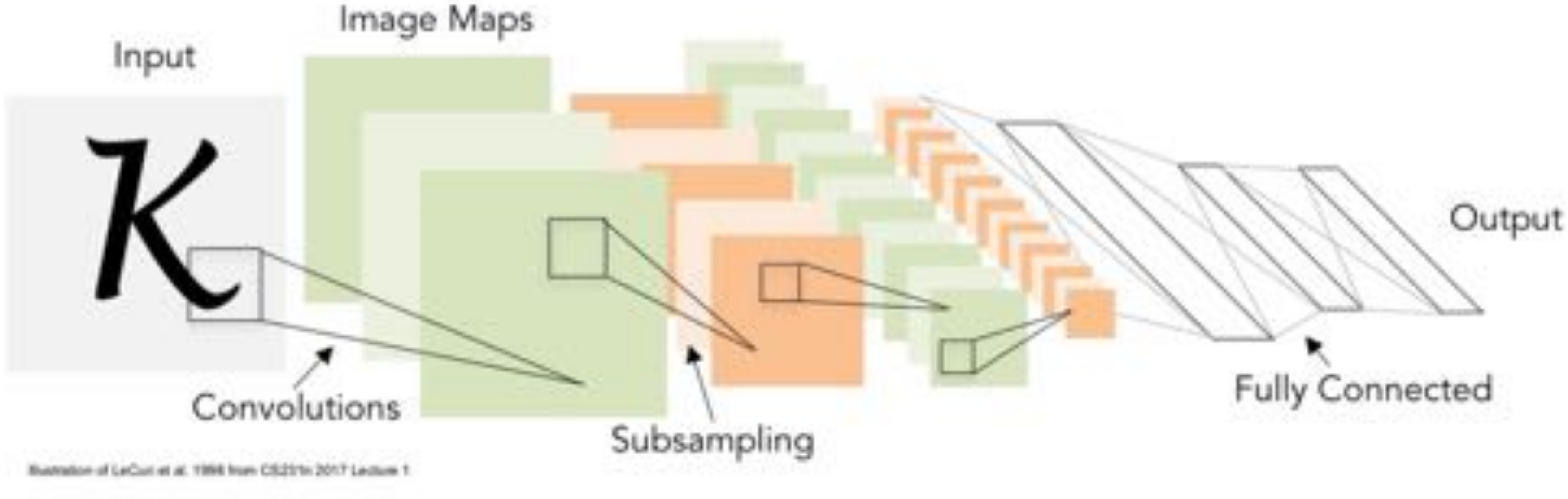
Which model to use?

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)

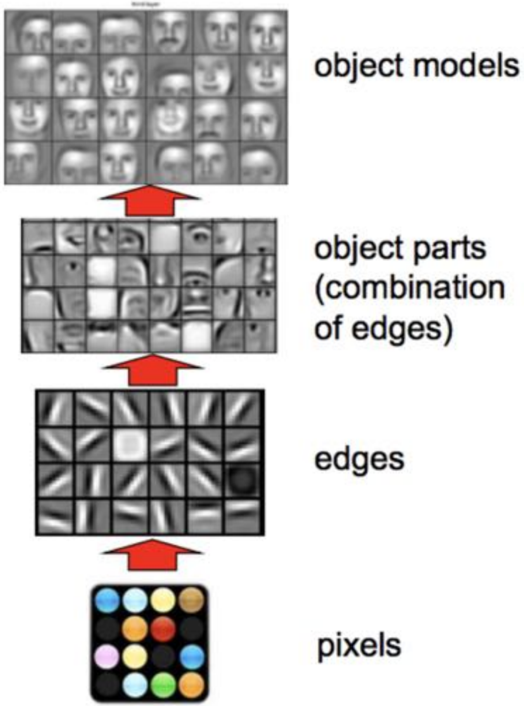


Slide Credits: Li, Johnson & Yeung, Stanford 2019

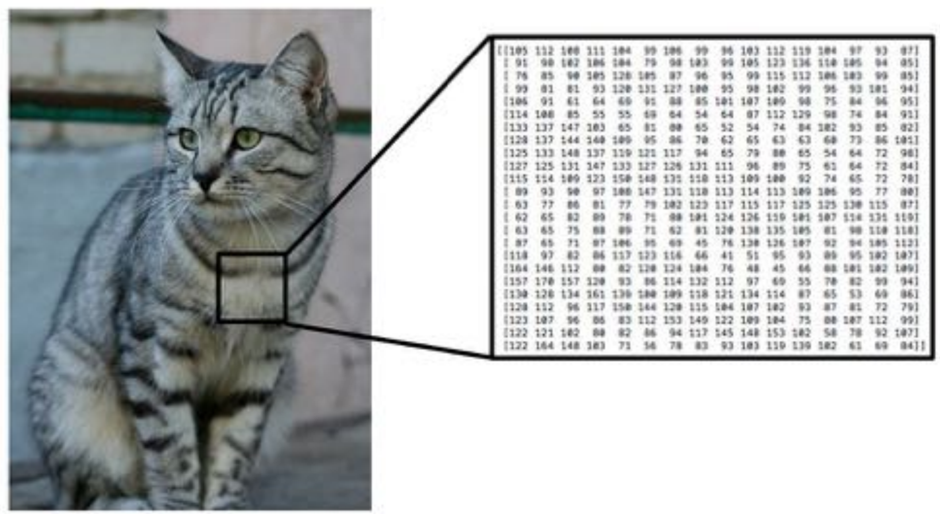
Why “Deep” Networks?

- 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

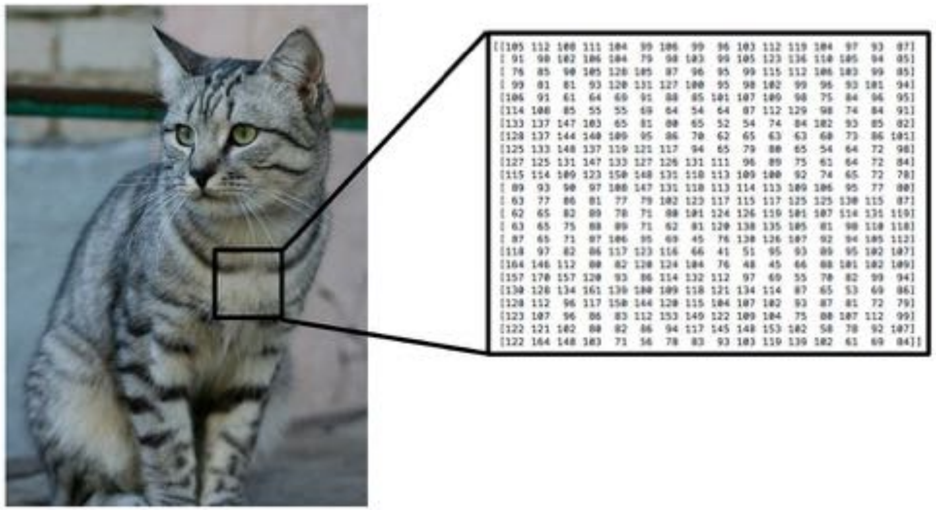


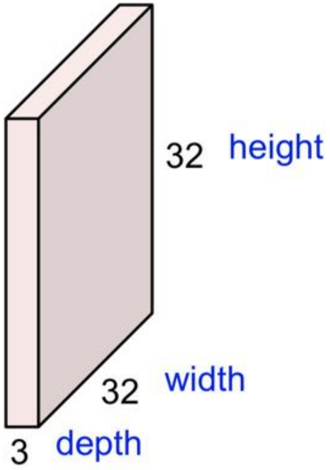
Data/Input representation: Pixel intensities



Slide Credits: Li, Johnson & Yeung, Stanford 2019

Data/Input representation: Pixel intensities

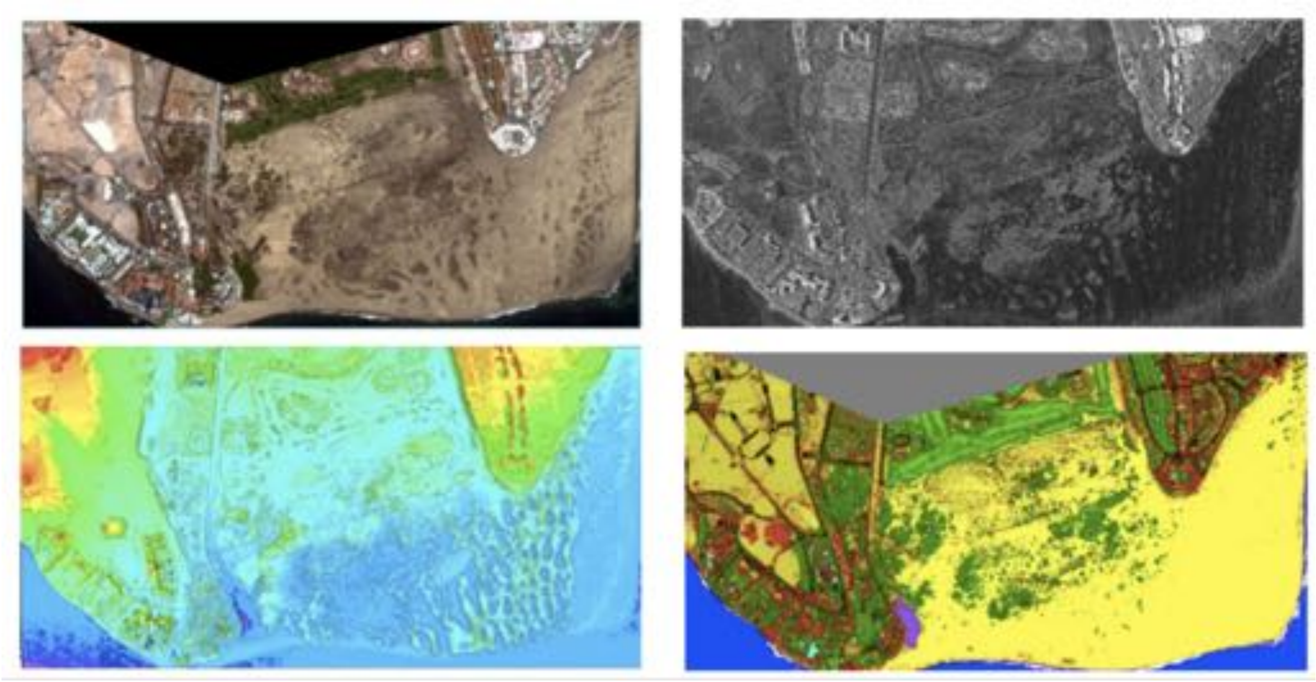




RGB data ***tensor***

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Data/Input representation: Pixel intensities

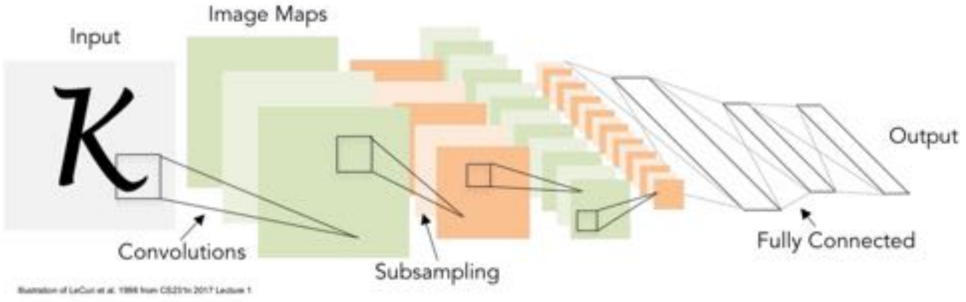


Multi-spectral data

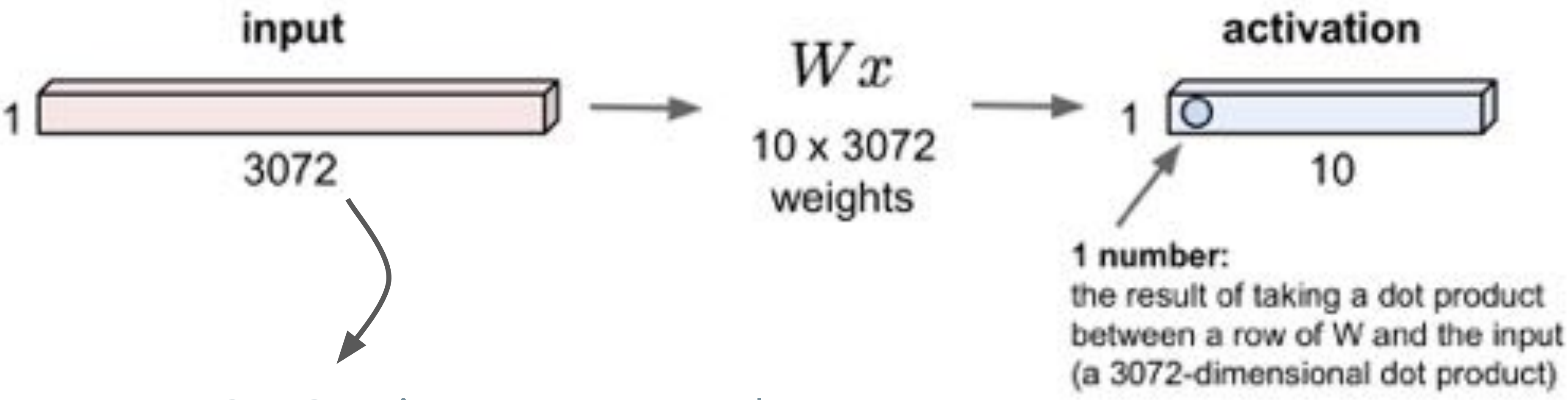
Convolutional Neural Networks (CNNs)

Basic components:

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



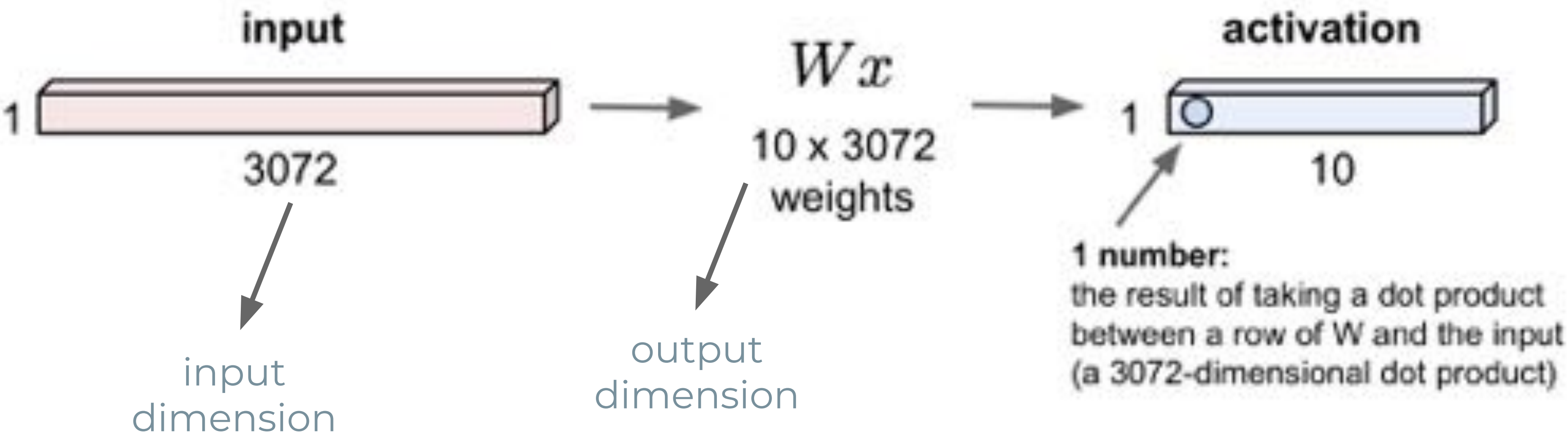
Fully Connected Layer



e.g. a 32x32x3 image → stretch to 3072 x 1 (spatial structure is lost)

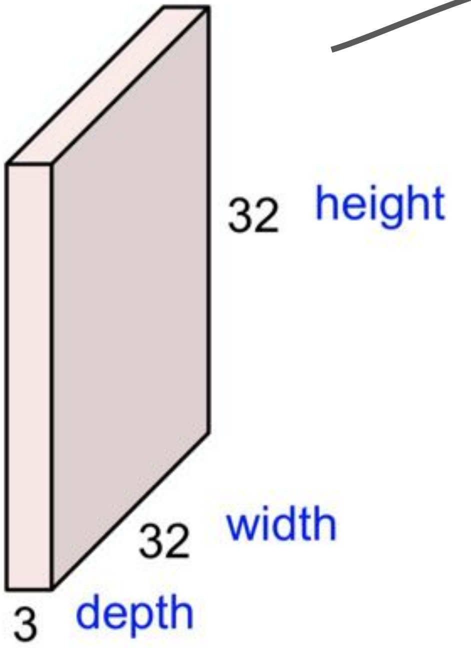
Slide Credits: Li, Johnson & Yeung, Stanford 2019

Fully Connected Layer



Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

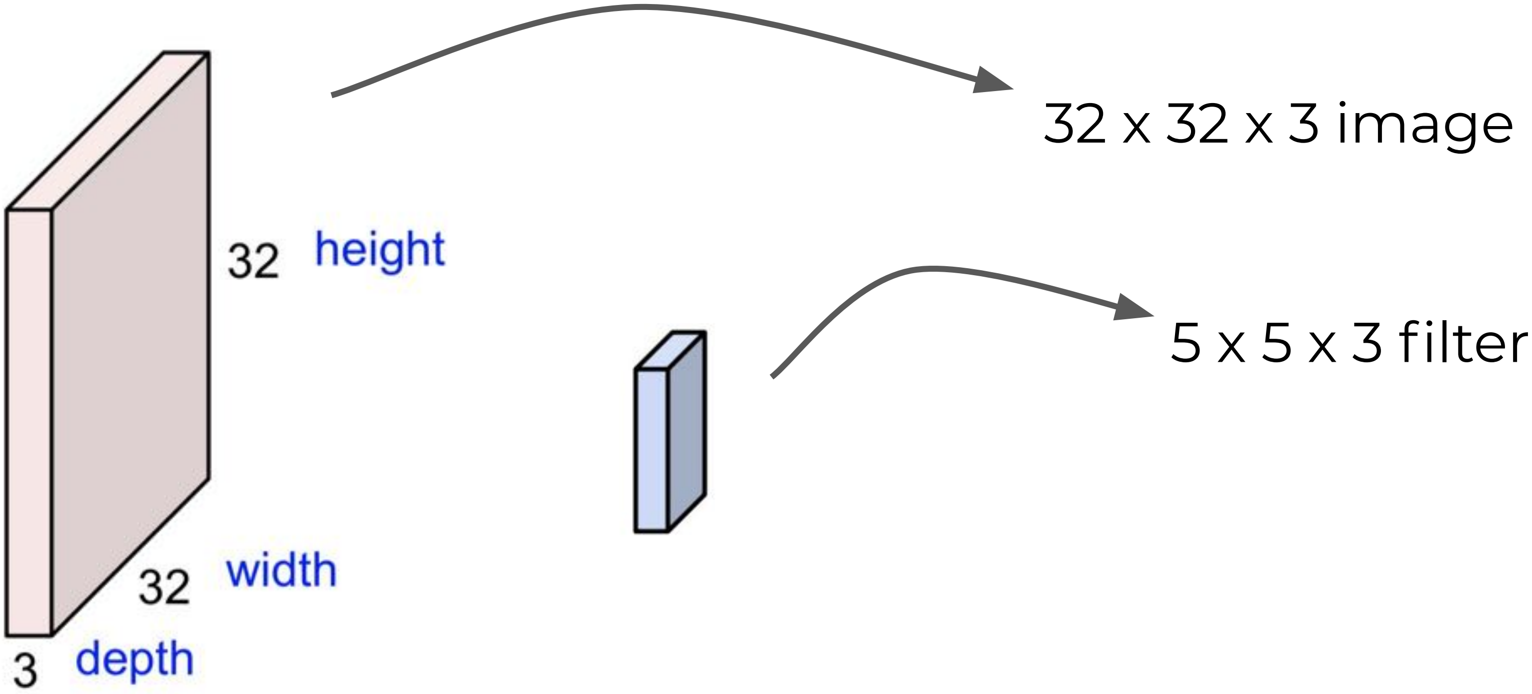


32 x 32 x 3 image

preserve
spatial
structure

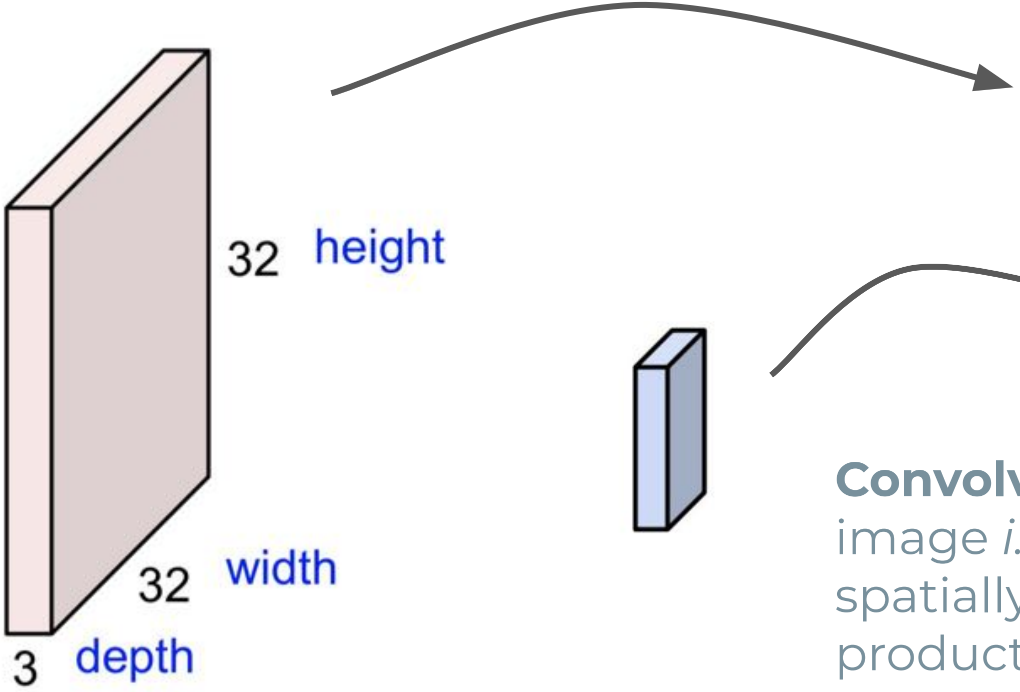
Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer



Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

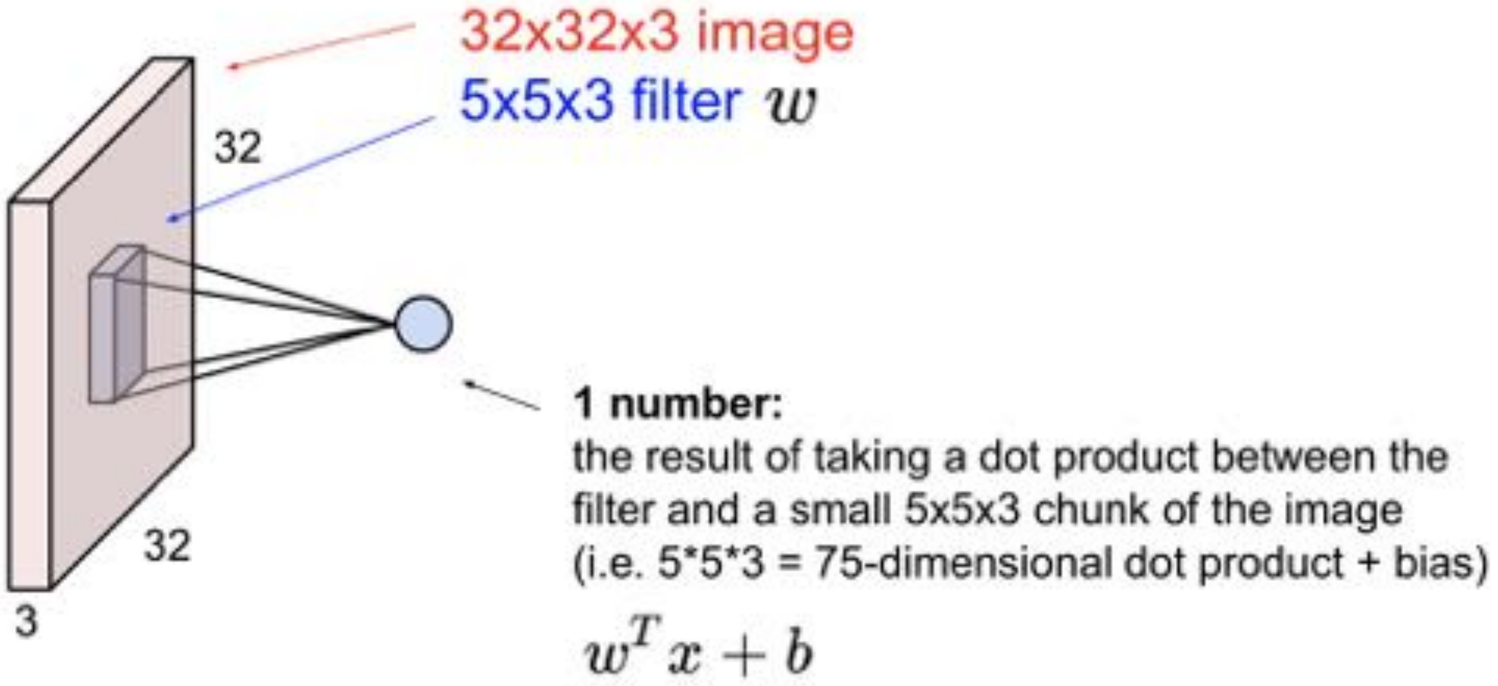


32 x 32 x 3 image

5 x 5 x 3 filter

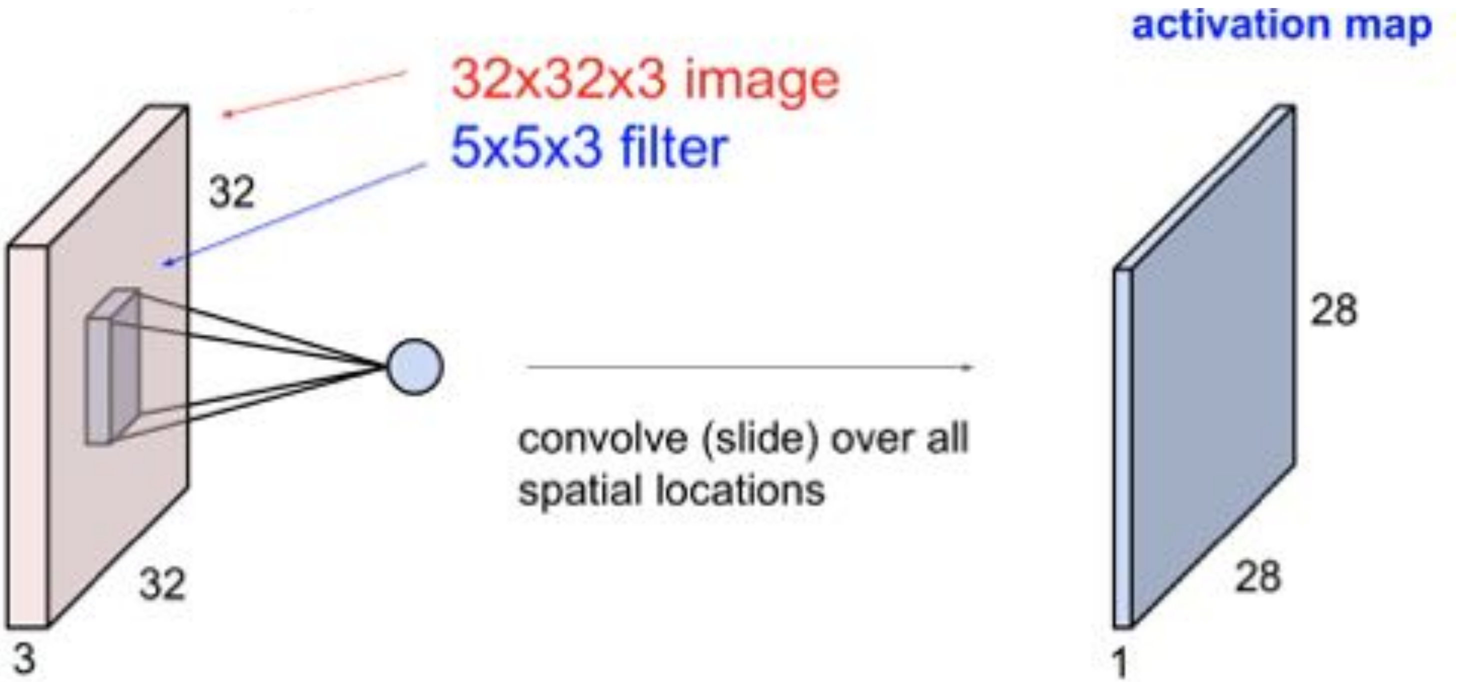
Convolve the filter with the image *i.e.*, “slide over the image spatially, computing dot products”

Convolution Layer



Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer



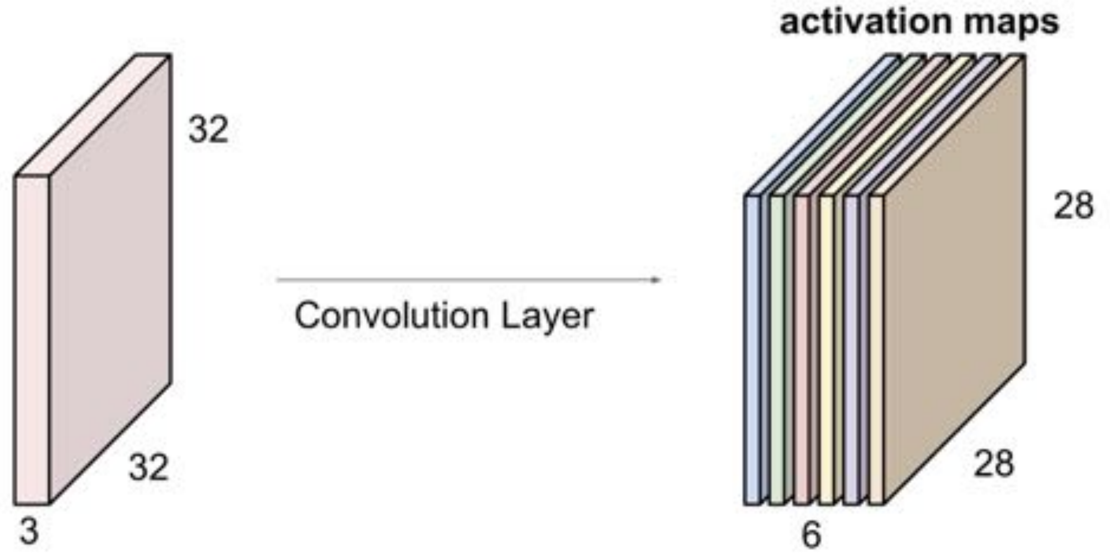
Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

a second filter

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

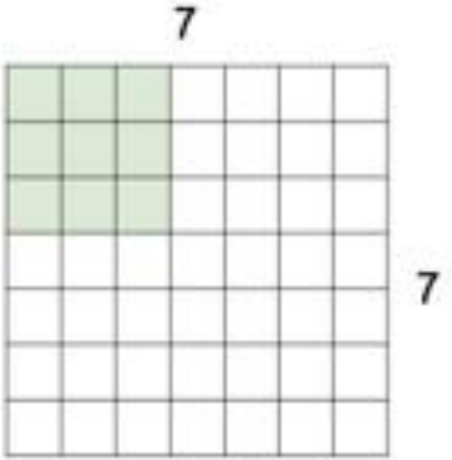


for 6 filters, output is 28 x 28 x **6**

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

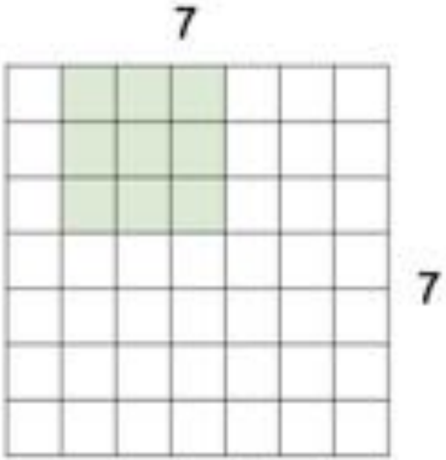
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Layer

A closer look at spatial dimensions:

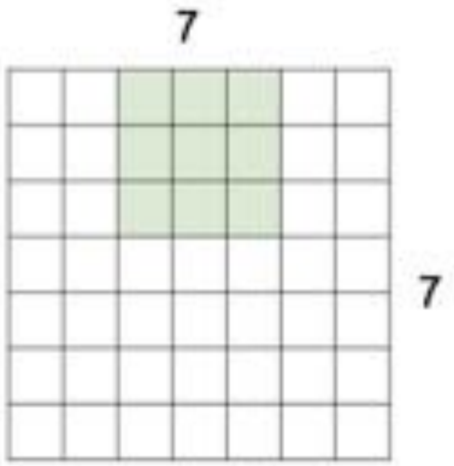


7x7 input (spatially)
assume 3x3 filter

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

A closer look at spatial dimensions:

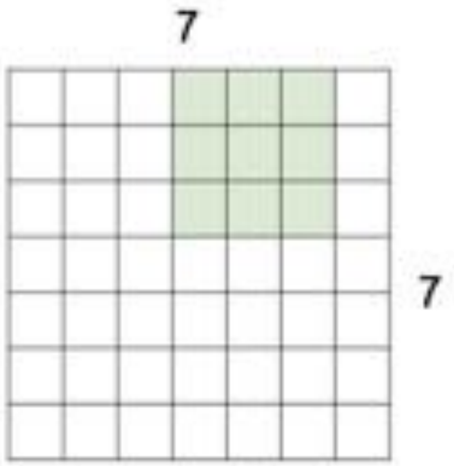


7x7 input (spatially)
assume 3x3 filter

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer

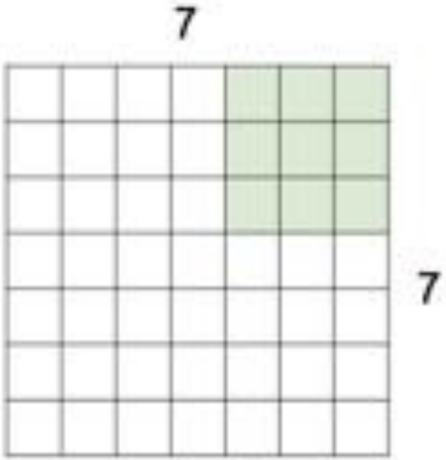
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Layer

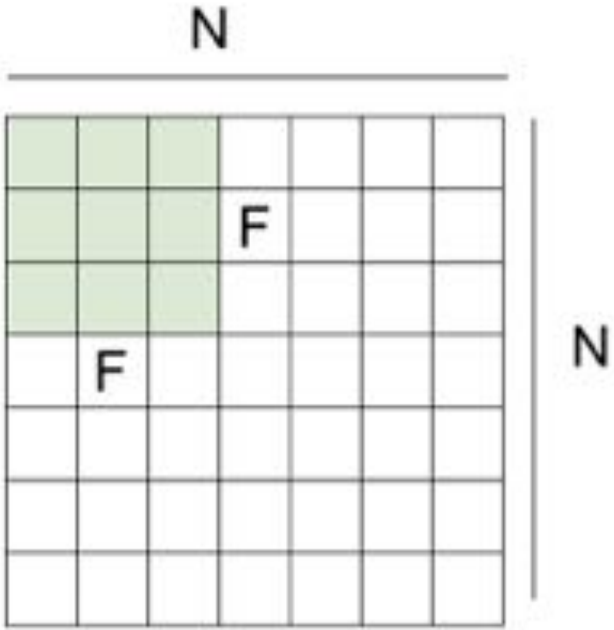
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Convolution Layer - stride

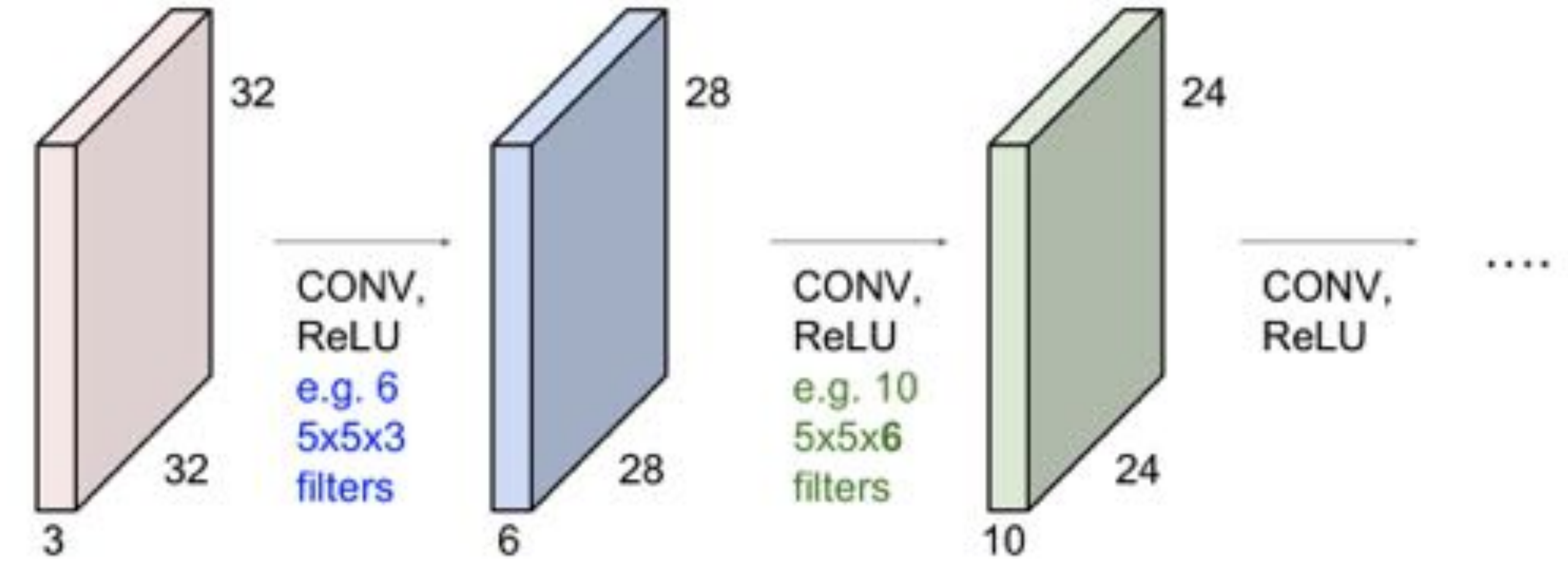


Output size:
(N - F) / stride + 1

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 :\

Slide Credits: Li, Johnson & Yeung, Stanford 2019

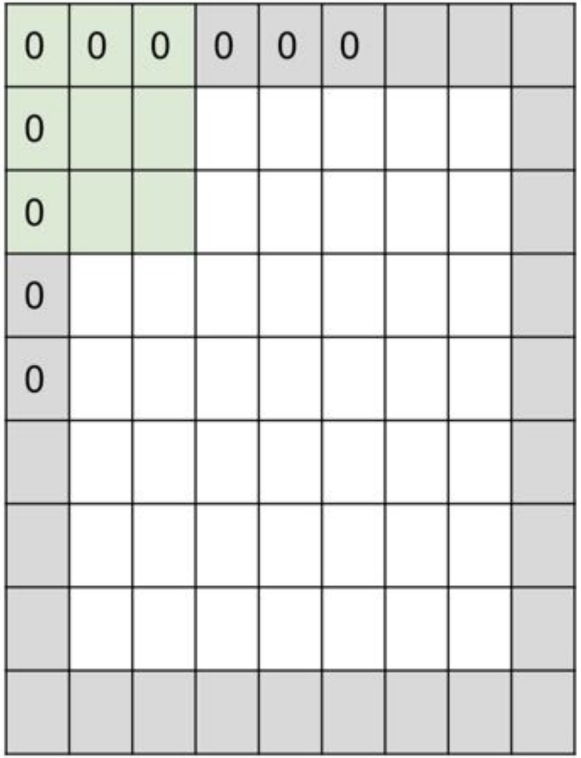
Convolution Layer - stride



shrinking too fast!

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Convolution Layer - padding

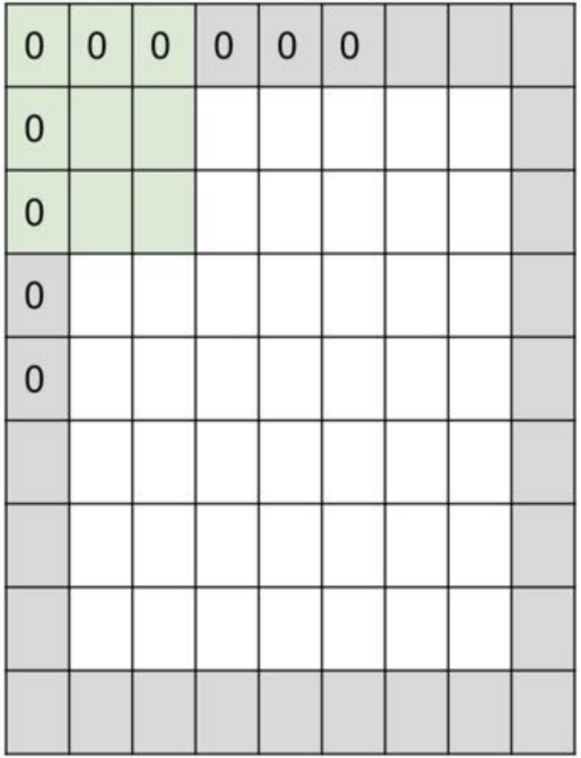


- input 7x7
- 3x3 filter
- stride = 1
- pad with 1 pixel border

what is the output?

Slide Credits: Li, Johnson & Yeung, Stanford 2019

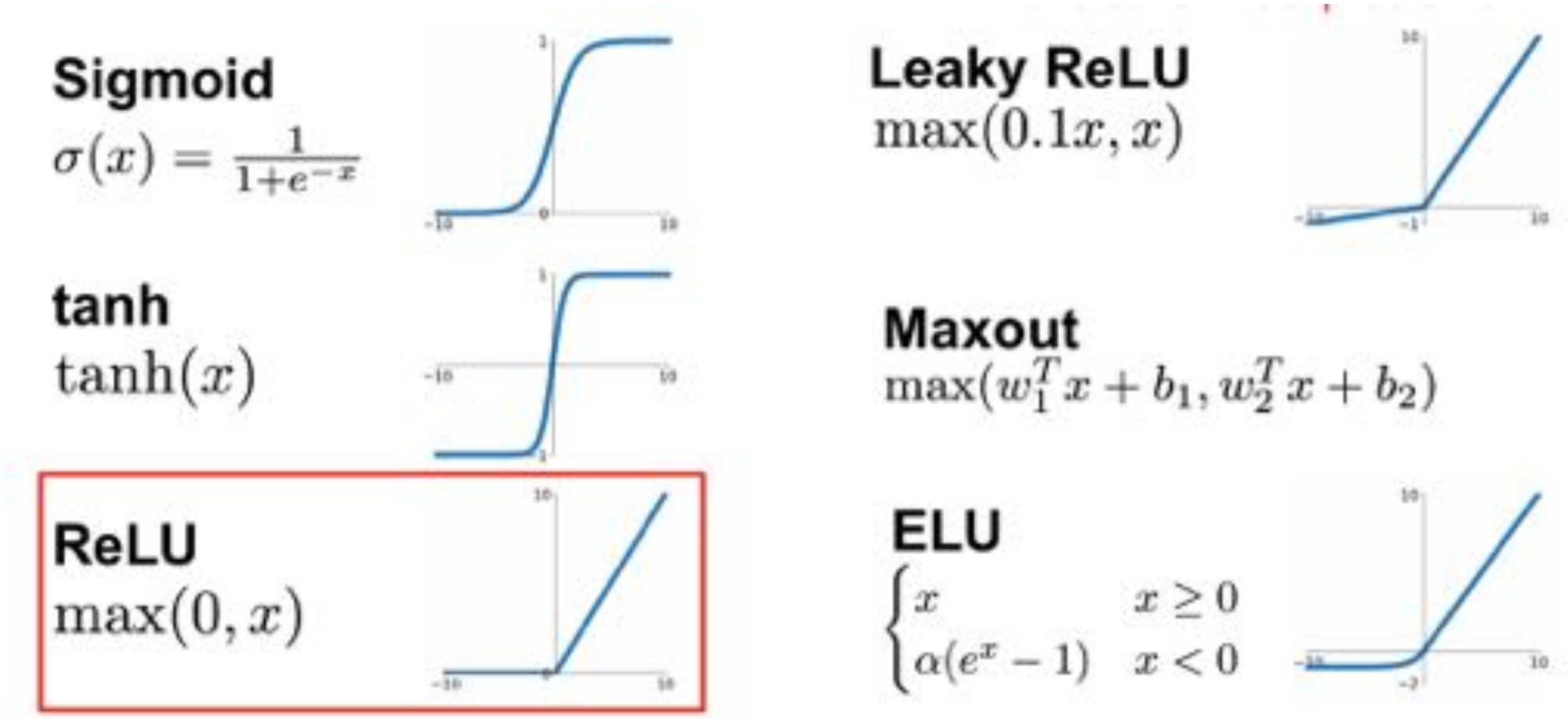
Convolution Layer - padding



- 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)

Slide Credits: Li, Johnson & Yeung, Stanford 2019

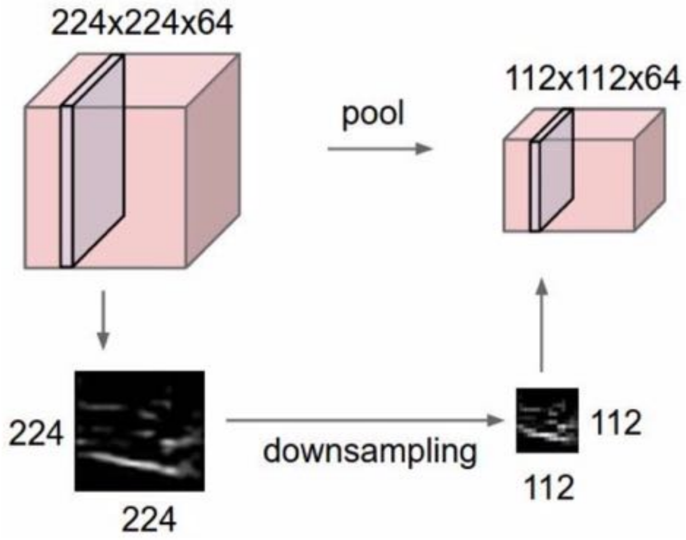
Activation Function: ReLU



Slide Credits: Li, Johnson & Yeung, Stanford 2019

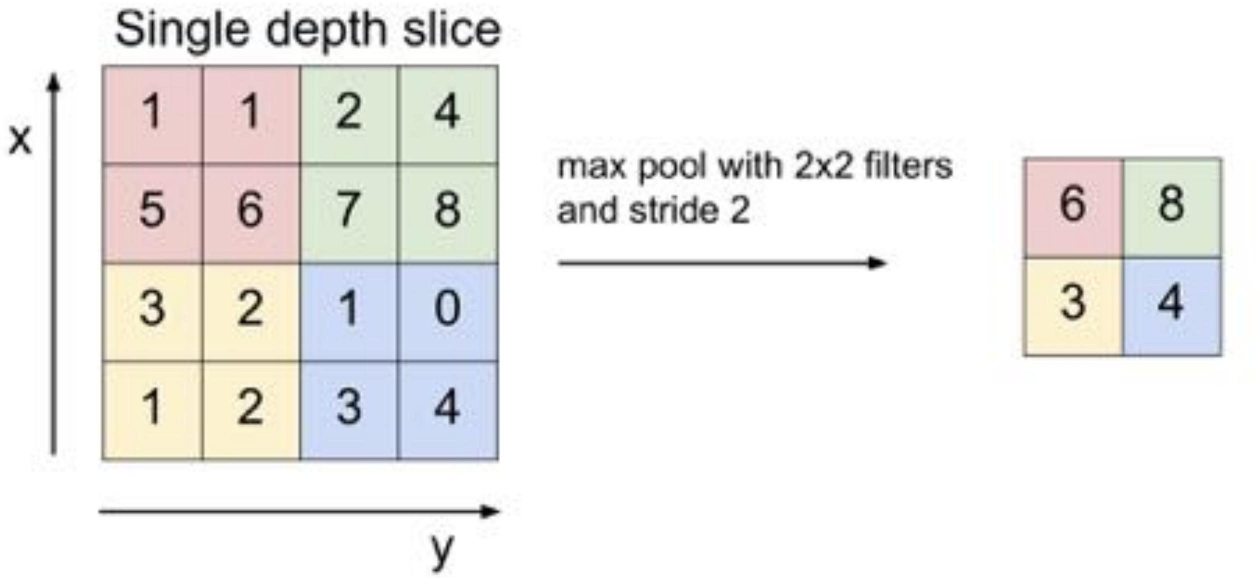
Pooling Layer

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



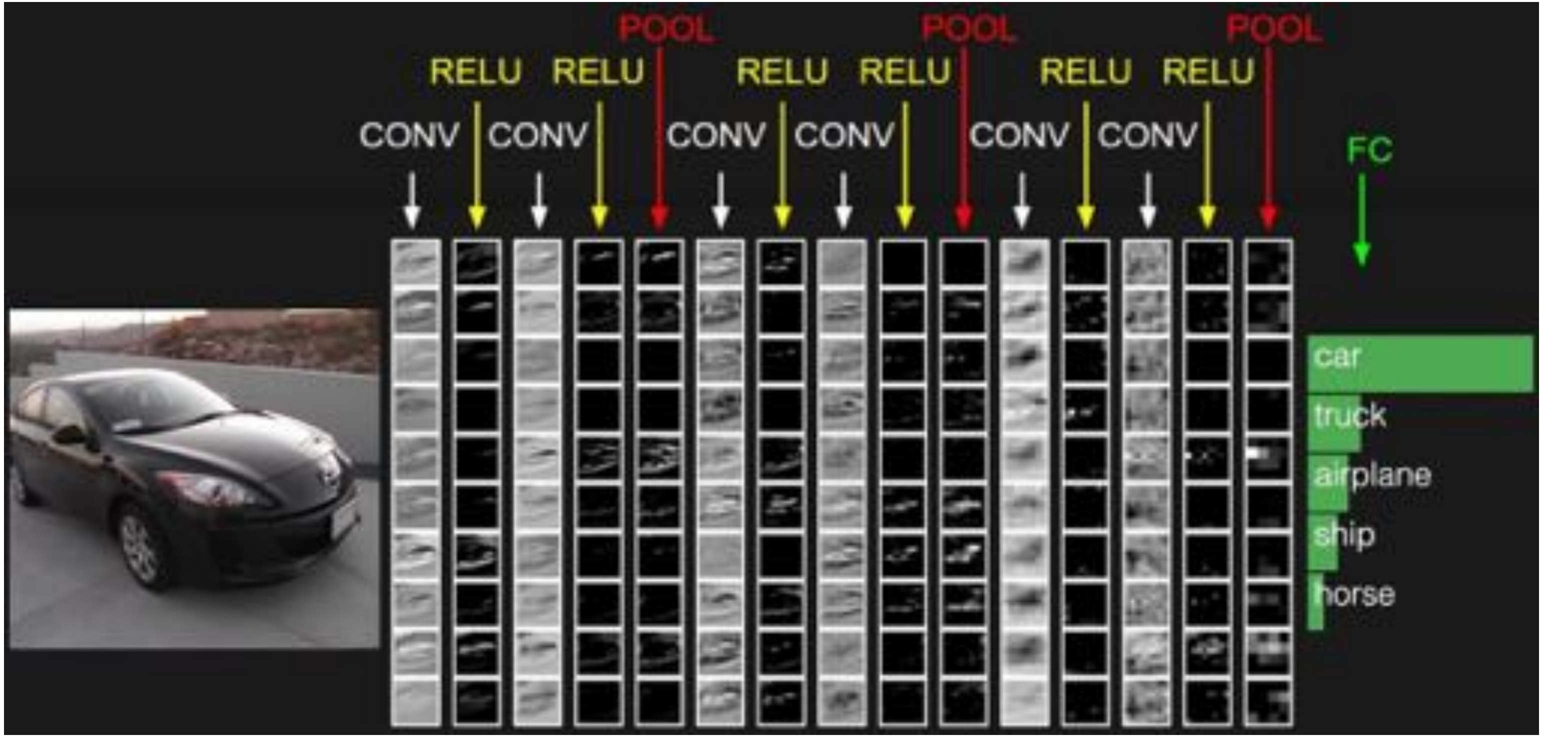
Slide Credits: Li, Johnson & Yeung, Stanford 2019

Pooling Layer: Max pooling



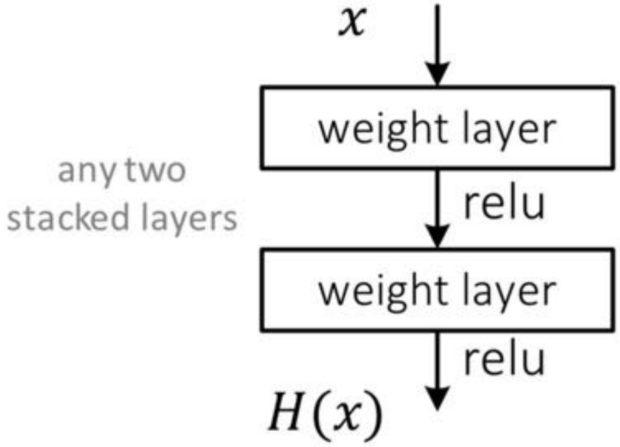
Slide Credits: Li, Johnson & Yeung, Stanford 2019

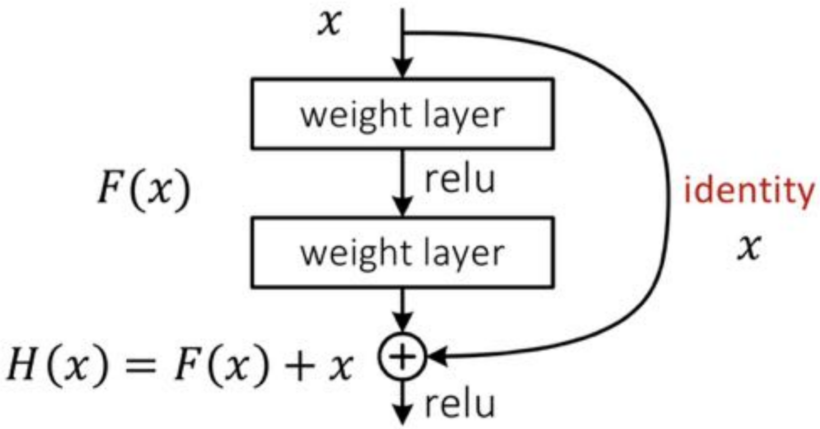
Putting it all together



Slide Credits: Li, Johnson & Yeung, Stanford 2019

Residual Connections [He *et al.* 2016]



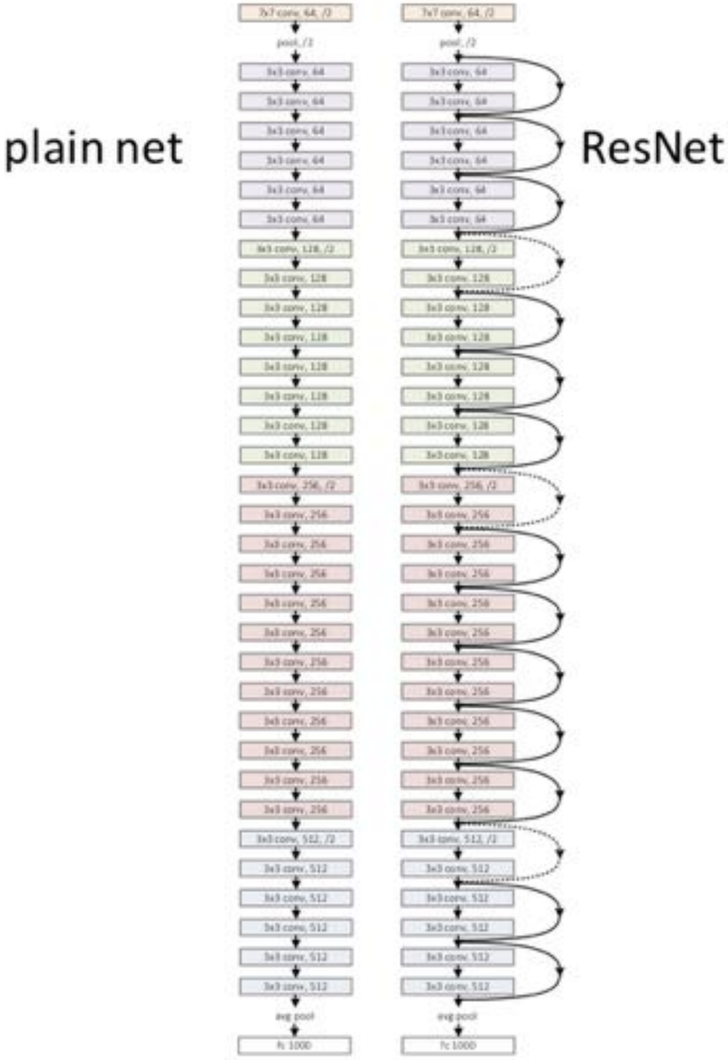


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

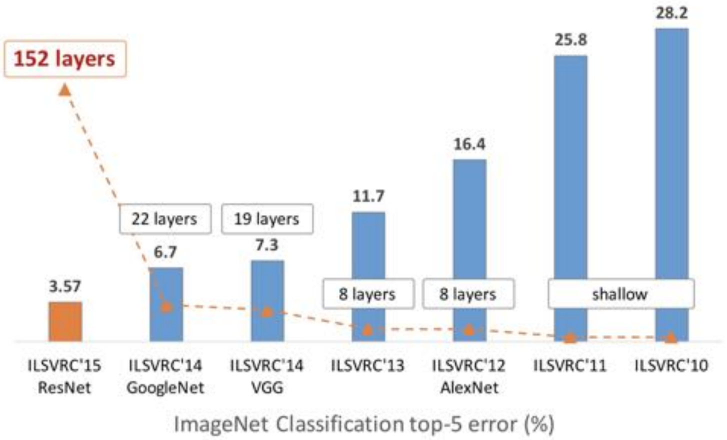
ResNet [He *et al.* 2016]

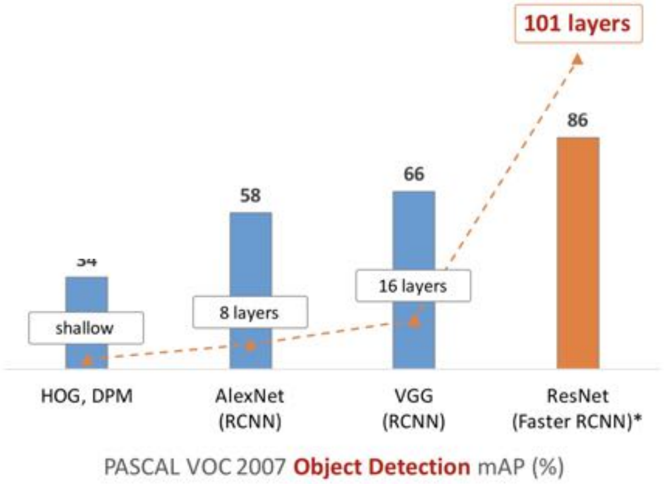
Simple but deep design:

- all 3 x 3 conv
- spatial size / 2 ⇒ # filters x 2
  (same complexity per layer)
- Global Average Pooling (GAP)



ResNet [He *et al.* 2016]





Pytorch example

```python
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()
```

https://pytorch.org/hub/pytorch_vision_resnet/

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)
```

https://pytorch.org/hub/pytorch_vision_resnet/

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
```

https://pytorch.org/hub/pytorch_vision_resnet/

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))
```

https://pytorch.org/hub/pytorch_vision_resnet/

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

https://paperswithcode.com/

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]

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

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(\overset{\text{input data}}{x_i}, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

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

Slide Credits: Li, Johnson & Yeung, Stanford 2019

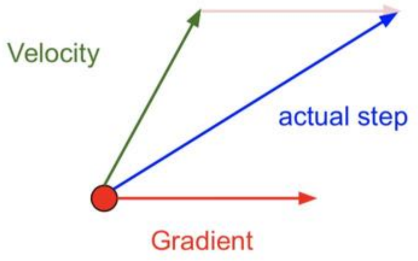
How to train a CNN model?

(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

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

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

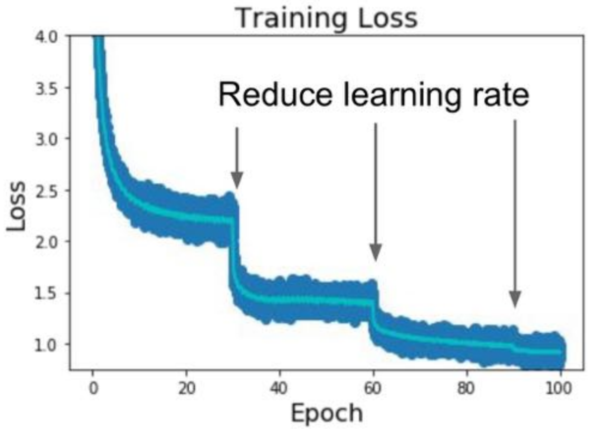
Batch normalization

Stochastic depth, fractional pooling, etc

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Learning Rate (LR)

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



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

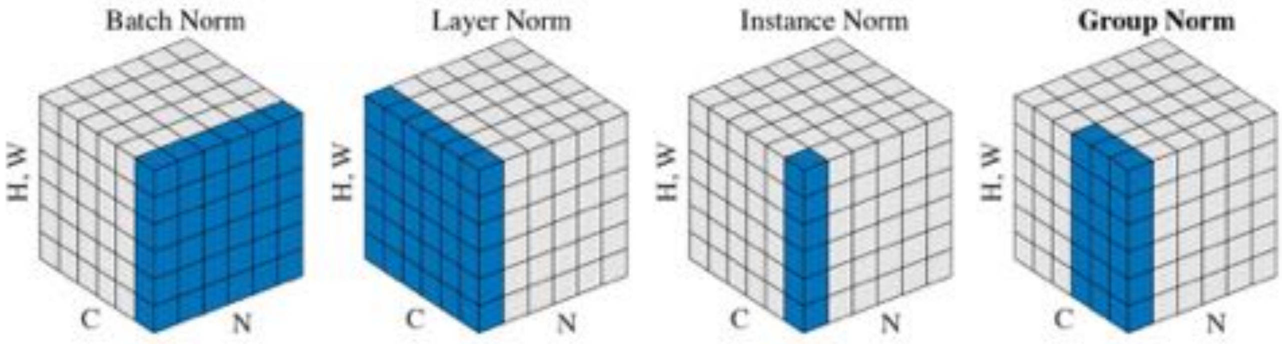
α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Slide Credits: Li, Johnson & Yeung, Stanford 2019

Very important training tips

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

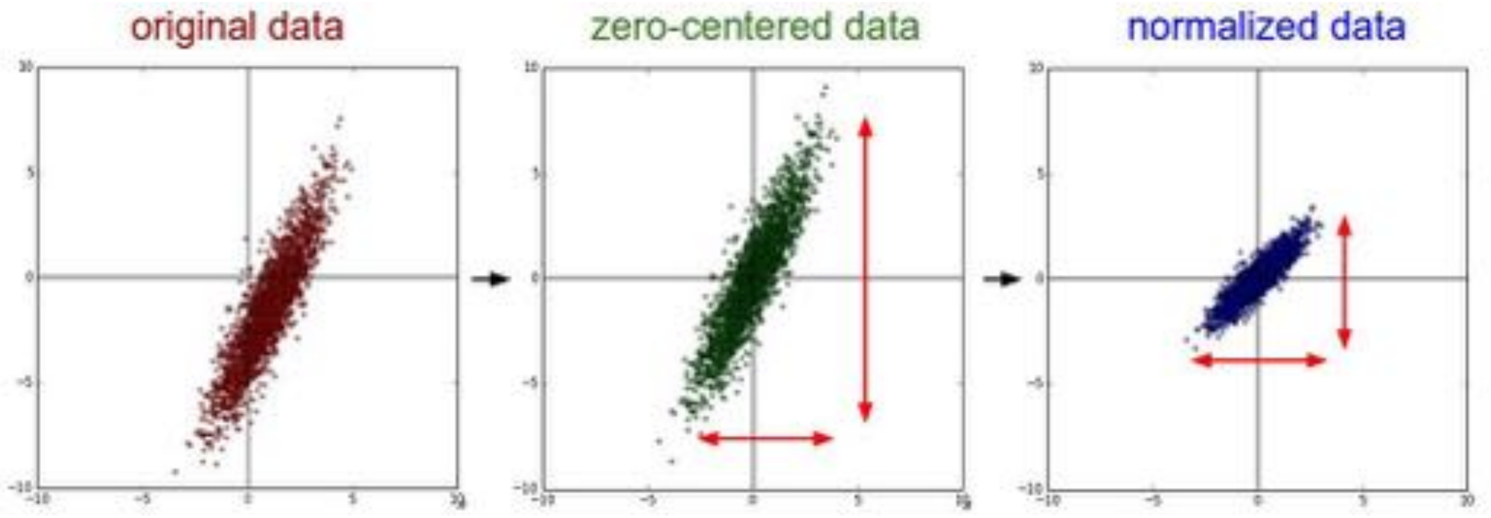
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$



Wu and He, "Group Normalization", ECCV 2018

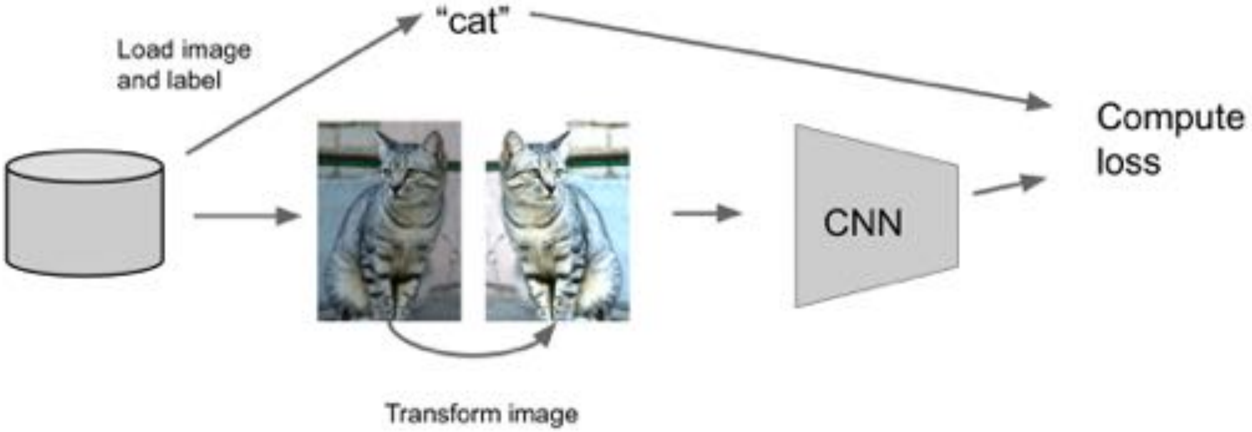
Very important training tips

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



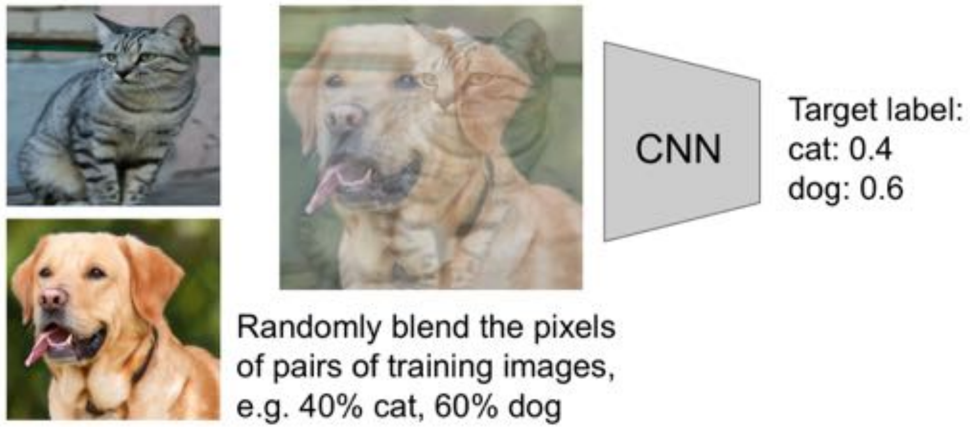
Very important training tips

- 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



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): rand * sqrt(2 / d_{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**

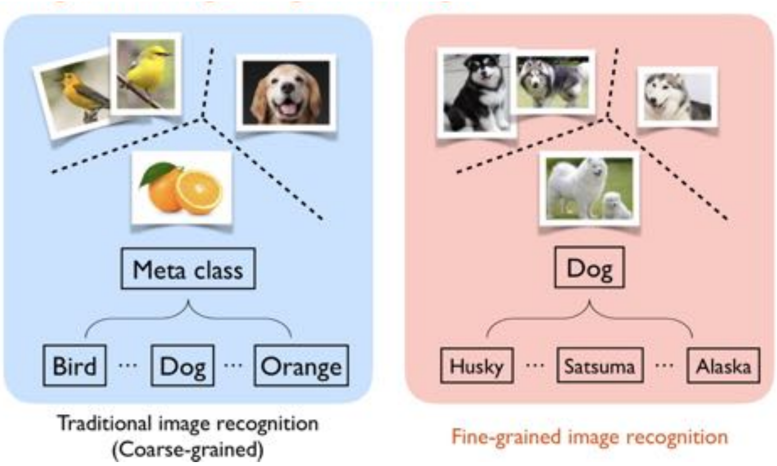
Fine-grained Recognition

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



6th Fine-Grained Visual Categorization (FGVC) Workshop at CVPR 2019

More realistic competitions:

- iNaturalist challenge
- fashion
- products
- wildlife camera traps
- butterflies & moths
- ***cassava leaf disease (iCassava)***

6th Fine-Grained Visual Categorization Workshop @ CVPR 2019

Fine-Grained Recognition in two steps

1) Start from a state-of-the-art (possible pre-trained) model

2) Fine-tune depending on amount of available data & compute

Note: Lots of paper 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

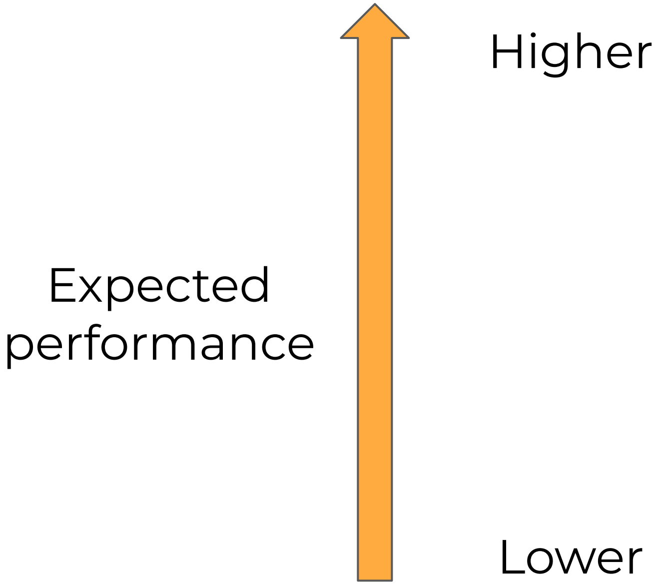
- Start from a model pre-trained on a large dataset
  - Pre-train dataset *as close to the target domain as possible* [Ciu *et al* CVPR 2018]
  - Lots of publicly available models!
    - Check the github pages of the latest architectures
    - Models after training from 1 Billion images from FB

Pre-trained vs train from scratch

- **Train** a model *from scratch* with the data

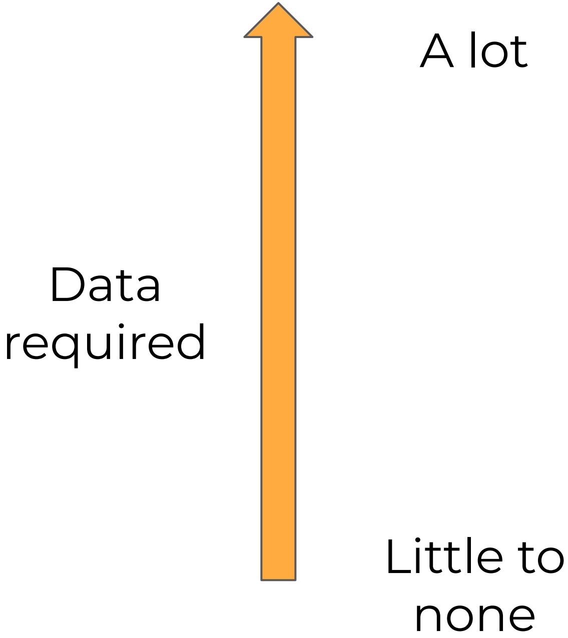
- **Fine-tune** a *pre-trained* model

- 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)

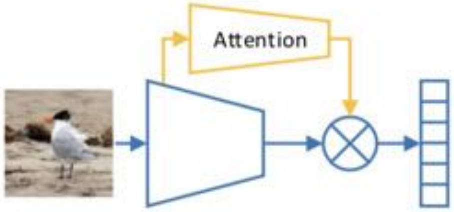
Dealing with small datasets or small inter-class variance

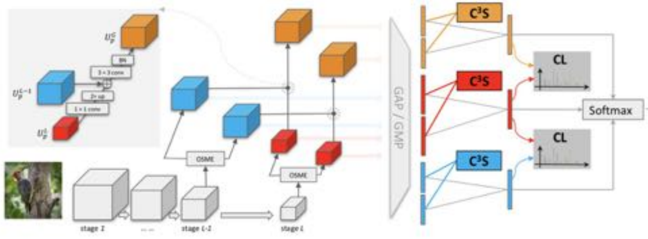
- Data augmentation is highly important
  - auto augment, mixup, manifold-mixup, generate/hallucinate new data,
- Feature normalization is highly important
  - try L2-norm, centering, PCA
- Test-time augmentations
  - multiple crops
  - Multi-resolution testing, model ensembles
- Train and test image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

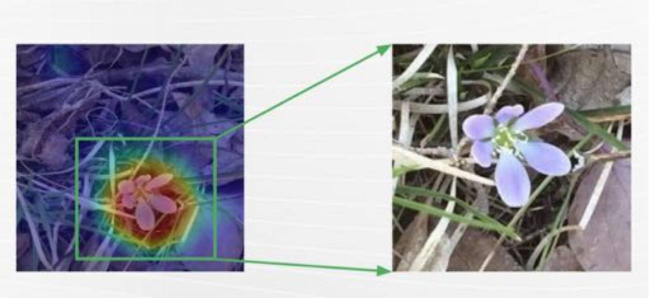
Many tricks in the [FGVC6 iNaturalist 2019 challenge winner slides]

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]

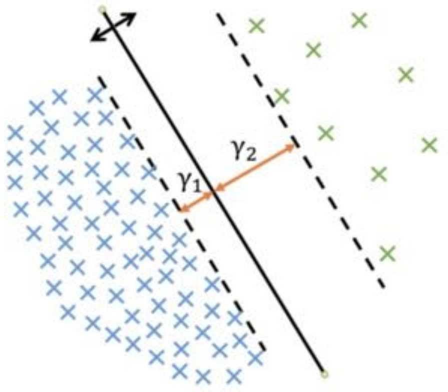






Dealing with imbalanced data (“long-tailed recognition”)

- Label-Distribution-Aware Margin Loss [Cao *et al.*, NeurIPS 2019]
- Class-balanced loss [Cui *et al.* CVPR 2019]
- Decouple representation from classifier learning [*under review*]
  - 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://www.skamalas.com/#dsa