Deep Learning for CV

Nguyen Thi Oanh
Hanoi University of Science and Technology
oanhnt@soict.hust.edu.vn

SOICT, HUST

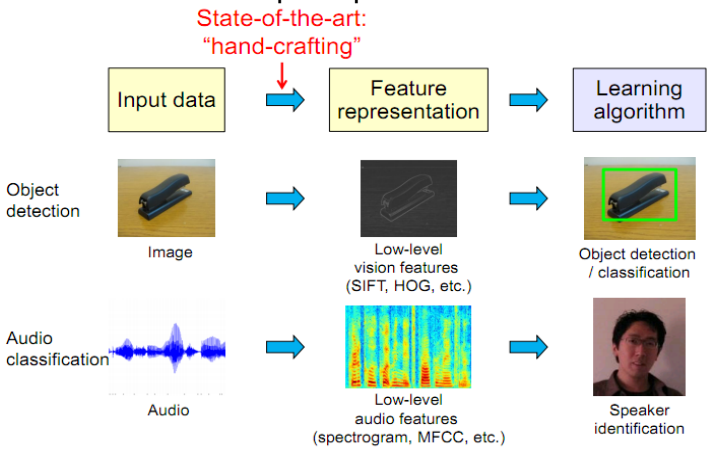
Content

- Introduction
- ML for image classification
- CNN
  - Architectures
  - Training

Computer Vision

3

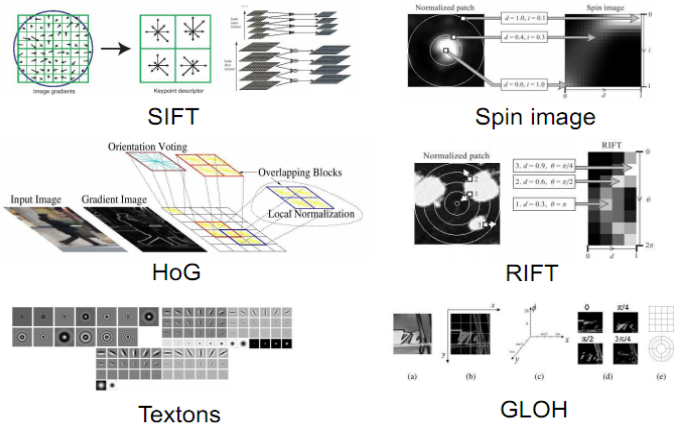
Traditional machine perception



Computer Vision

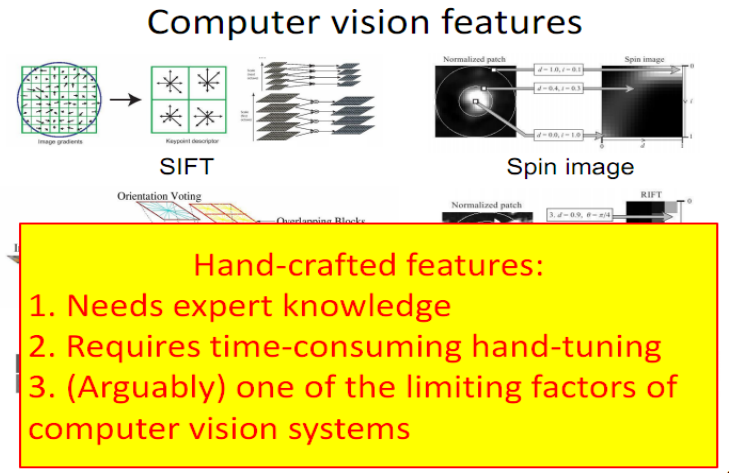
4

Computer vision features



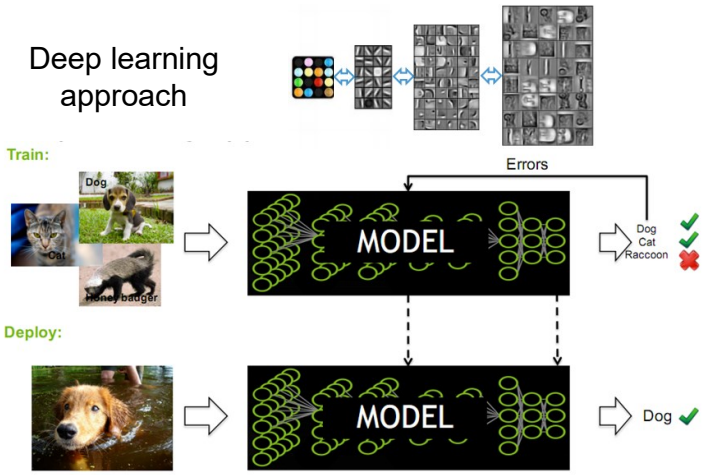
Computer Vision

5



Deep Learning

6

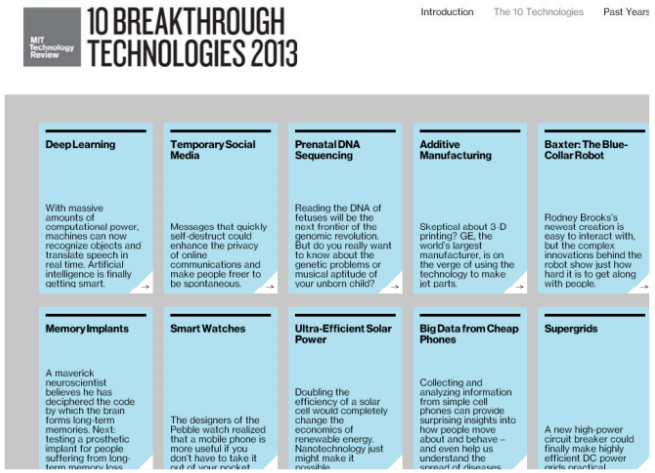


Deep Learning

7

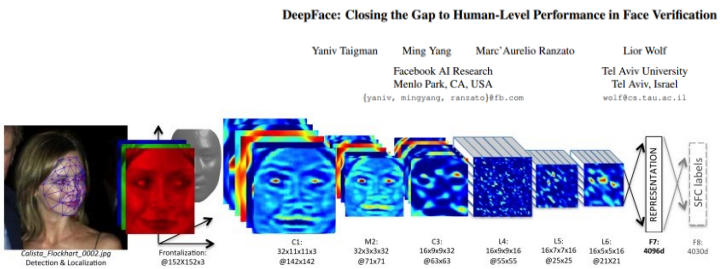
Deep Learning

8



Deep Learning vs Human

9



DeepFace 2014

Closing the Gap to **Human** Level **Performance** in **Face** Verification

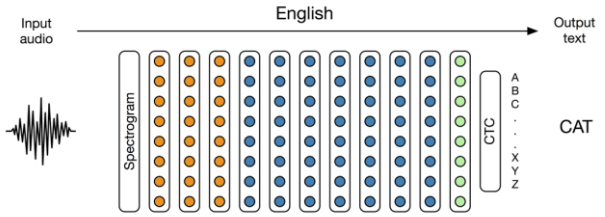
Accuracy

DeepFace: 97.35%

Human: 97.5%

Deep Learning vs Human

10



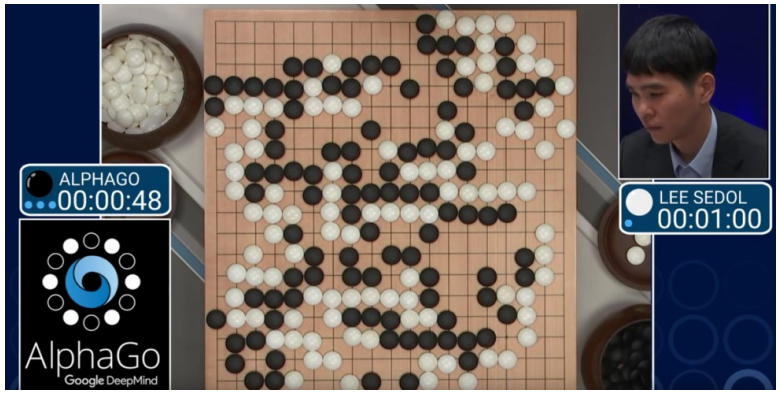
Deep Speech 2015

Baidu has developed a voice system that can recognize English and Mandarin speech **better than people**, in some cases.

"For short phrases, out of context, we seem to be **surpassing human levels of recognition**"- Andrew Ng

Deep Learning vs Human

11

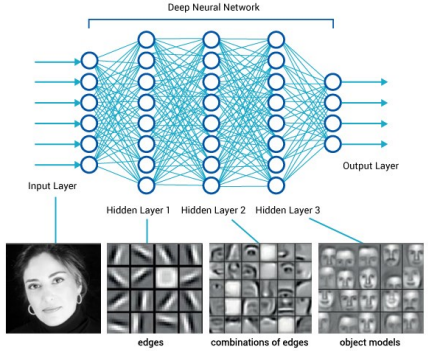


AlphaGo vs Human: 9-1

What is Deep Learning?

12

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple non-linear transformations.



Content

13

- Introduction
- ML for image classification
- CNN
  - Architectures
  - Training

Convolution

14

12	3	19
25	10	1
9	7	17

*

1	2
3	4

=

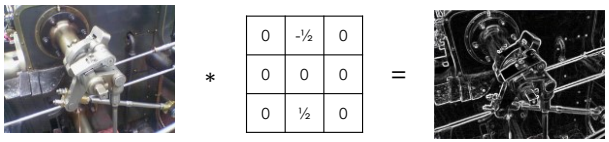
133	75
100	101

$$f[n,m] * h[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l]\, h[n-k, m-l]$$

Why they are useful

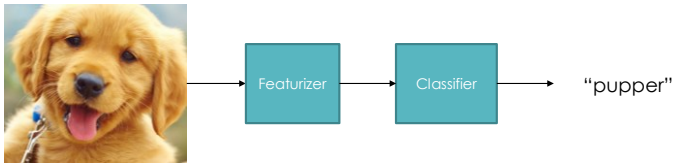
15

Allow us to find **interesting insights/features** from images!



Recall Image Classification…

16



Allow us to use features to put **images in categories**!

Wait a Minute… 17

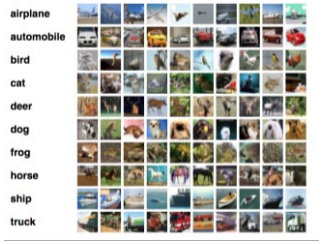
Convolution = Image -> Features

Classification Algorithm = Features -> Category

➔ **Let's put 'em together!**

In Specific… 18

Let's build a **convolution-based** classification algorithm for the CIFAR-10 dataset (10 classes, 32x32 images):



Feature Extractor

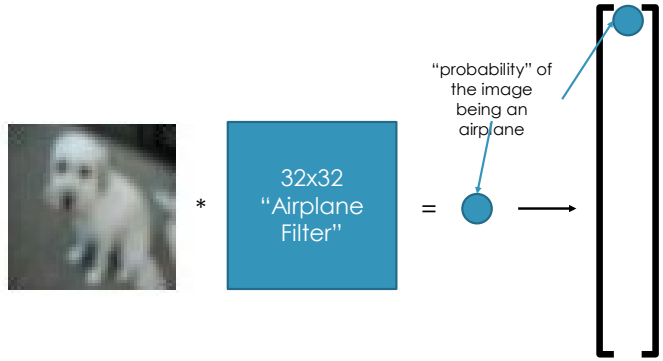
19

Feature Extractor

20

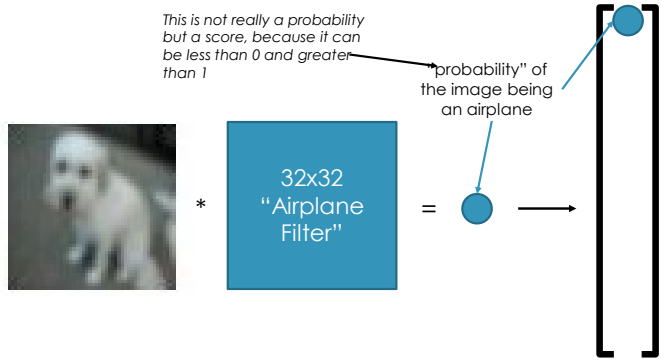
Feature Extractor

21



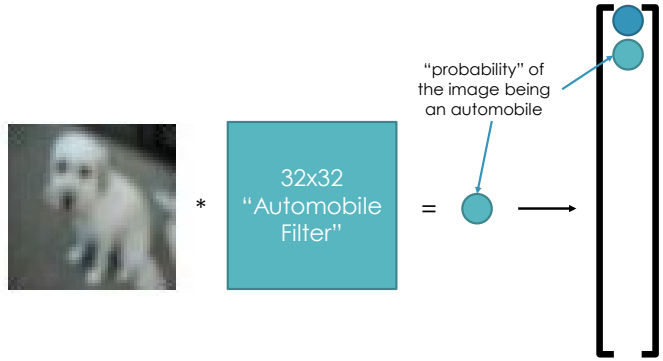
Feature Extractor

22



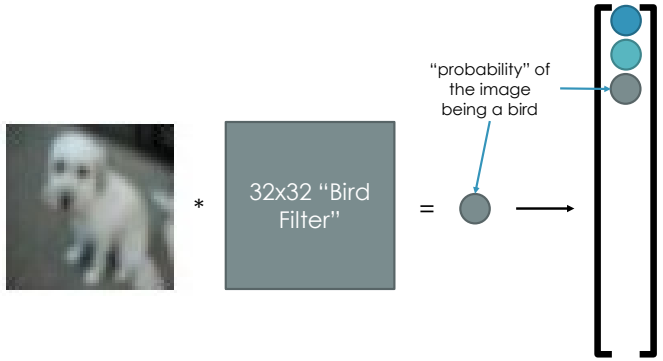
Feature Extractor

23



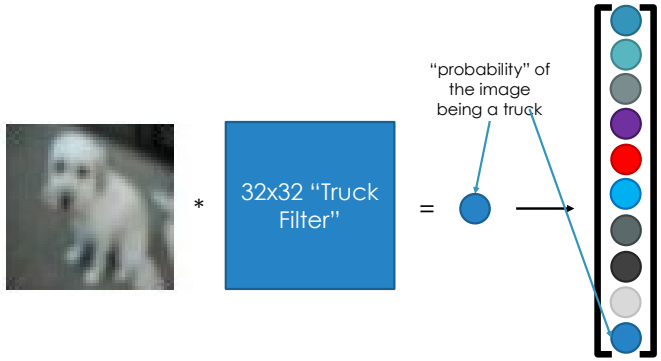
Feature Extractor

24



Feature Extractor

25



Classifier

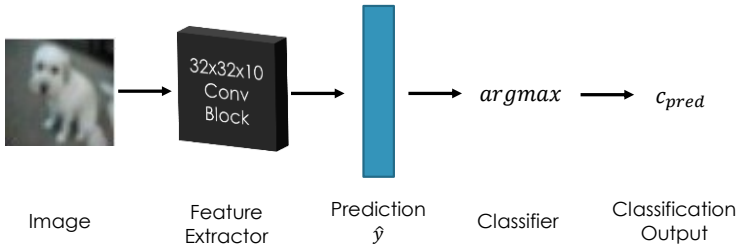
26

$$c_{pred} = \arg\max(\quad)$$

We predict the class that has the highest probability!

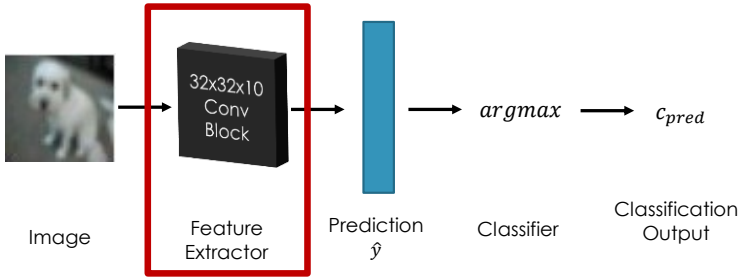
The Whole Shebang

27



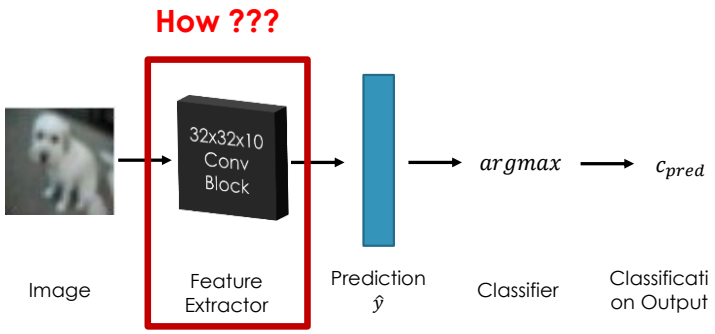
The Whole Shebang

28



The Whole Shebang

29



Reframing convolution

30

12	21
18	31

$*$

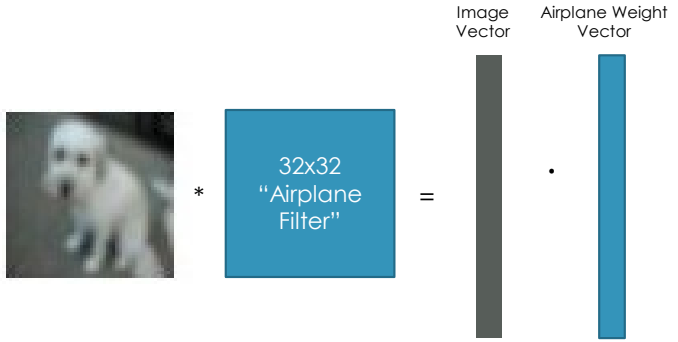
1	2
3	4

$=$

$$\begin{bmatrix}12\\21\\18\\31\end{bmatrix} \cdot \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

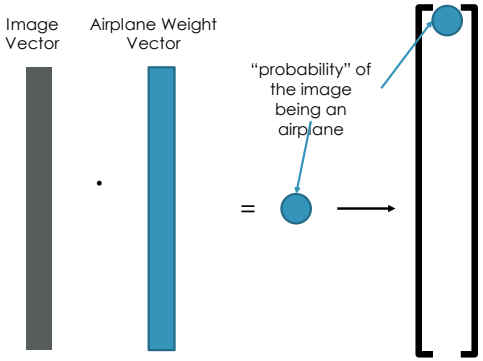
Reframed Feature Extractor 31

Reframed Feature Extractor 32



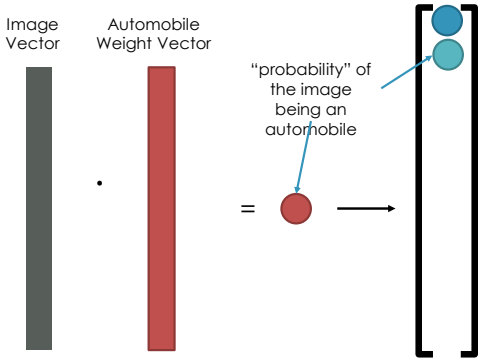
New Feature Extractor

33

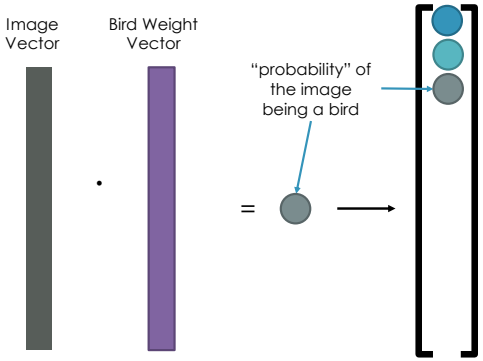


New Feature Extractor

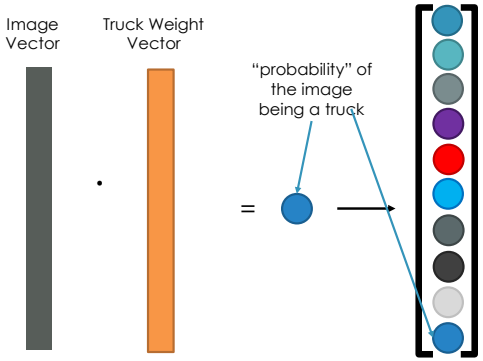
34



New Feature Extractor

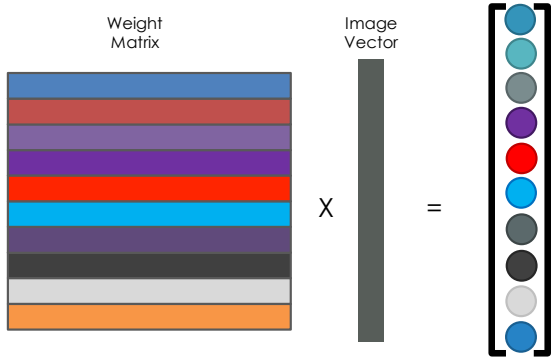


New Feature Extractor



New Feature Extractor

37



New Feature Extractor

38

$$Wx = \hat{y}$$

W: the (10x1024) matrix of weight vectors

x: the (1024x1) image vector

$\hat{y}$: the (10x1) vector of class "probabilities"

New Feature Extractor

39

This simple computation is called a *fully-connected layer*!

$$Wx = \hat{y}$$

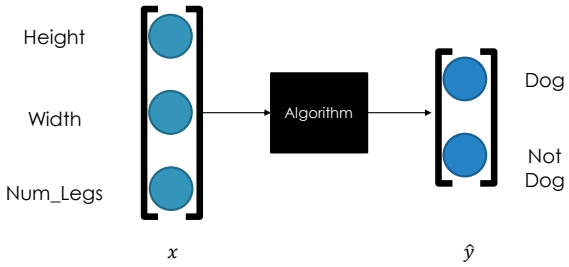
W: the (10x1024) matrix of weight vectors

x: the (1024x1) image vector

$\hat{y}$: the (10x1) vector of class "probabilities"

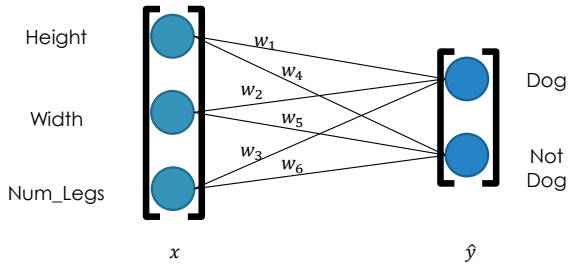
Aside: Fully-Connected Neural Networks

40



Aside: Fully-Connected Neural Networks

41



Aside: Fully-Connected Neural Networks

42

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \end{bmatrix} \cdot \quad = \quad$$

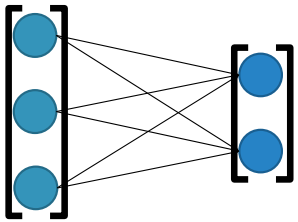
W x $\hat{y}$

$$Wx = \hat{y}$$

Aside: Fully-Connected Neural Networks

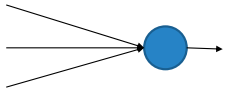
43

"Fully-Connected"



Every node is connected to every other node

"Neural Network"



Kinda looks like a neuron!

New Feature Extractor

44

$$Wx = \hat{y}$$

W: the (10x1024) matrix of weight vectors

x: the (1024x1) image vector

$\hat{y}$: the (10x1) vector of class "probabilities"

New Feature Extractor

45

$$Wx = \hat{y}$$

W: the (10x1024) matrix of weight vectors

x: the (1024x1) image vector

$\hat{y}$: the (10x1) vector of class "probabilities"?

Class Probability Vector

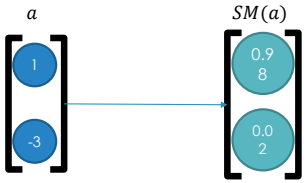
46

- Must have values between 0 and 1
- Must sum to 1
- There's no guarantee either requirement is satisfied!

$$\hat{y} = Wx$$

Softmax Function

47



$$\text{Softmax}\,(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Class Probability Vector

48

- Must have values between 0 and 1
- Must sum to 1

$$\hat{y} = Wx$$

- Must have values between 0 and 1
- Must sum to 1

$$\hat{y} = SM(Wx)$$

System so far…

- Feature extractor:
- Classifier: $\hat{y} = SM(Wx)$

$$c_{pred} = \arg\max(\hat{y})$$

System so far…

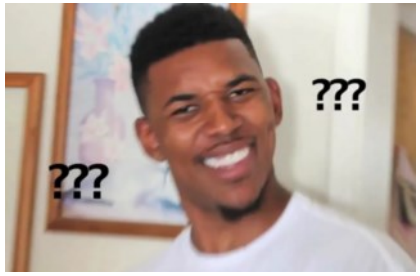
- Feature extractor:
- Classifier: $\hat{y} = SM(Wx)$

$$c_{pred} = \arg\max(\hat{y})$$

System so far…

51

- Feature extractor:
- Classifier:



Using the label

52

Let's compare our prediction with the real answer! For each image, we have the label y which tells us the true class:

X

y

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Dog class index

Key Insight:

53

We want:

$$\arg\max(\hat{y}) = \arg\max(y)$$

Key Insight:

54

We want:

$$\arg\max(\hat{y}) = \arg\max(y)$$

Which we can accomplish by:

$$W^* = \arg\min_W \left(-\sum_{x,y} \log(p_c) \right)$$

Where p_c is the probability of the true class in $\hat{y}$

Cross-Entropy Loss

55

Our loss function represents *how bad we are currently doing*:

$$L = -\log(p_c)$$

Examples:

$p_c = 0 \rightarrow L = -\log(0) = \infty$

$p_c = 0.1 \rightarrow L = -\log(0.1) = 2.3$

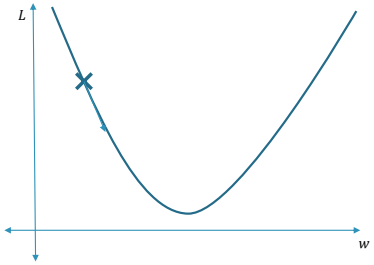
$p_c = 0.9 \rightarrow L = -\log(0.9) = 0.1$

$p_c = 1 \rightarrow L = -\log(1) = 0$

The larger the loss, the worse our prediction. We want to minimize L!

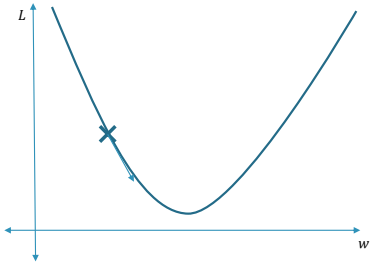
Minimizing Loss

56



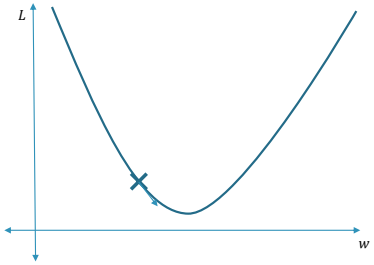
Minimizing Loss

57



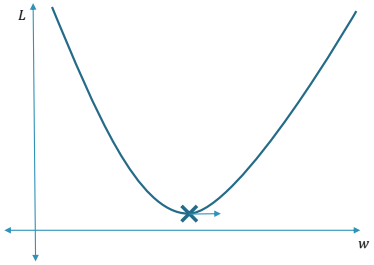
Minimizing Loss

58



Minimizing Loss

59



Minimizing Loss

60

Gradient Descent Pseudocode

61

```
for i in {0,…,num_epochs}:
    for x, y in data:
        ŷ = SM(Wx)
        L = CE(ŷ, y)
        dL/dW =???
        W := W − α dL/dW
```

Getting the Gradient

62

$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

Getting the Gradient

63

$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = \frac{dL}{dz}(x)$$

Getting the Gradient

64

$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = (SM(z) - y)(x)$$

Getting the Gradient

65

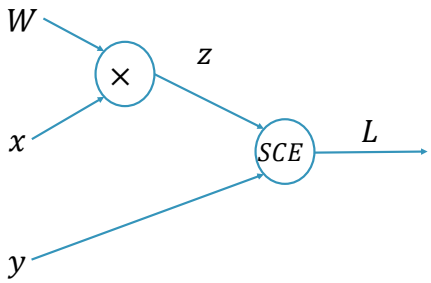
$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = (SM(z) - y)(x^T)$$

BACKPROPAGATION!

What is Backprop?

66

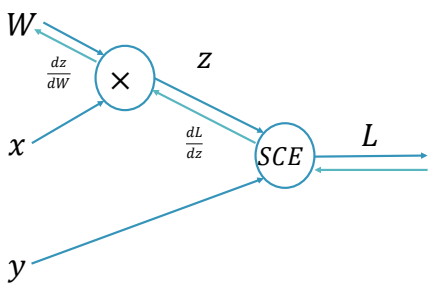


$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

What is Backprop?

67

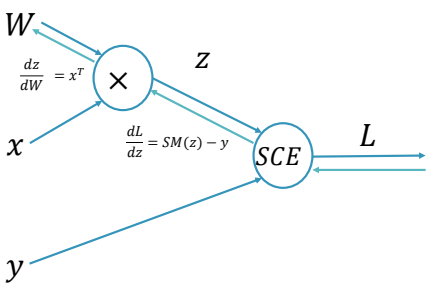


$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

What is Backprop?

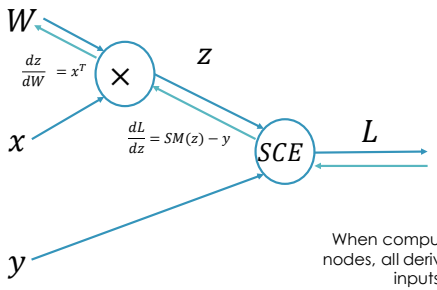
68



$$z = Wx$$
$$L = SCE(z, y)$$

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

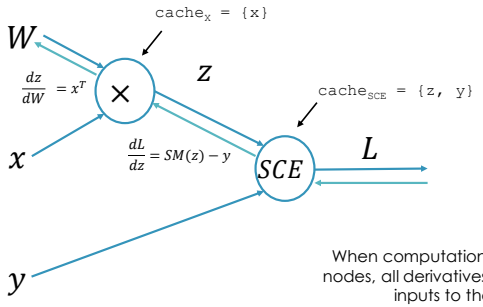
What is Backprop?



$$z = Wx$$
$$L = SCE(z, y)$$
$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

When computations are treated as nodes, all derivatives depend only on inputs to that node.

What is Backprop?



$$z = Wx$$
$$L = SCE(z, y)$$
$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

When computations are treated as nodes, all derivatives depend only on inputs to that node.

So, we can cache the initial computation and reuse!

Gradient Descent Pseudocode (Updated)

71

```
for i in {0,…,num_epochs}:
    for x, y in data:
        ŷ = SM(Wx)
        L = CE(ŷ, y)
        dL/dW = backprop(L)
        W := W − α dL/dW
```

Gradient Descent Pseudocode (Updated)

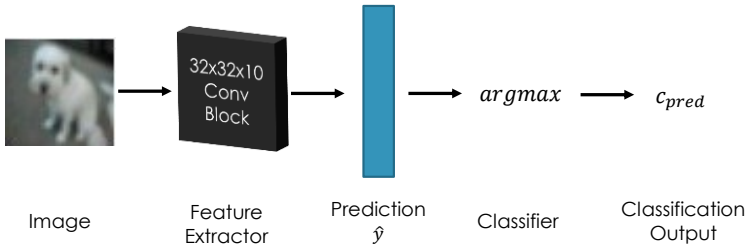
72

```
for i in {0,…,num_epochs}:
    for x, y in data:
        ŷ = SM(Wx)
        L = CE(ŷ, y)
        dL/dW = backprop(L)
        W := W − α dL/dW
```

MACHINE LEARNING!

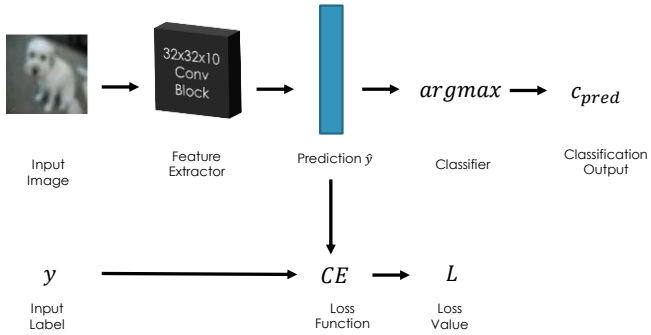
Simple Classification System

73



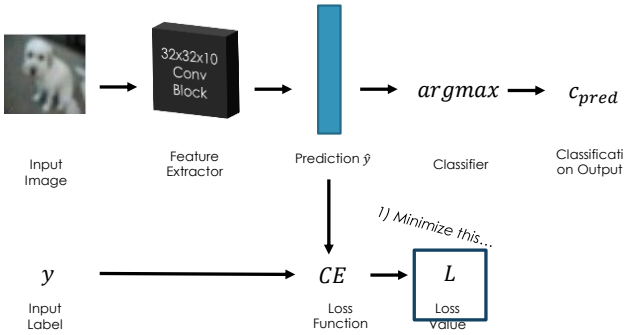
Simple Classification System (modified)

74



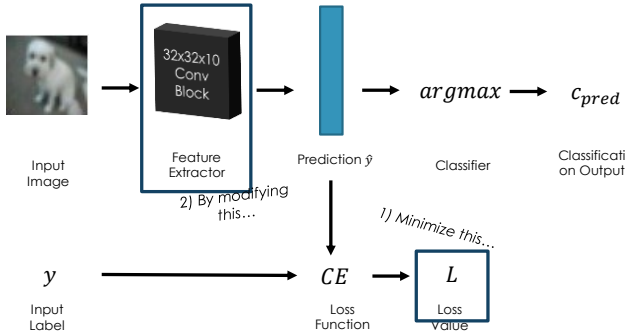
Simple Classification System (modified)

75



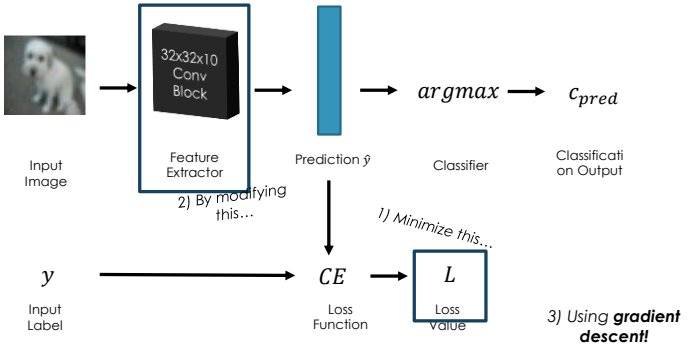
Simple Classification System (modified)

76



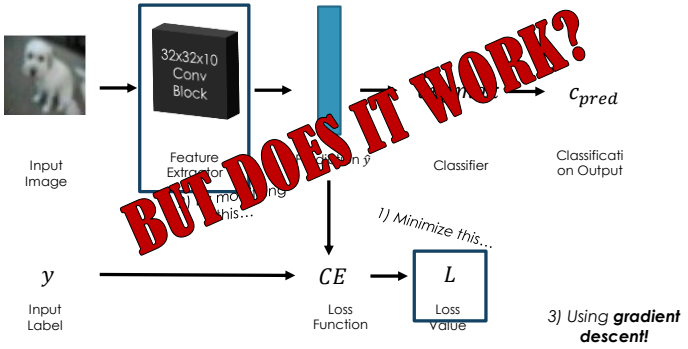
Simple Classification System (modified)

77



Simple Classification System (modified)

78



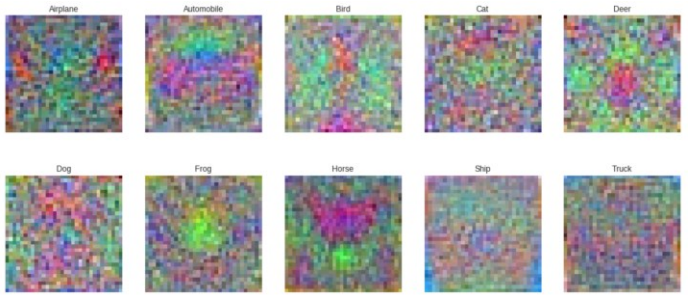
Simple System's Performance

79

- ~40% accuracy on CIFAR-10 test
  - Best class: Truck (~60%)
  - Worst class: Horse (~16%)
- Check out the model at: **https://tinyurl.com/cifar10**
- What about the filters? What do they look like?

Visualizing the Filters

80



Content

81

- Introduction
- Computer Vision & ML
- CNN
  - Architectures
  - Training

Convolutions

82

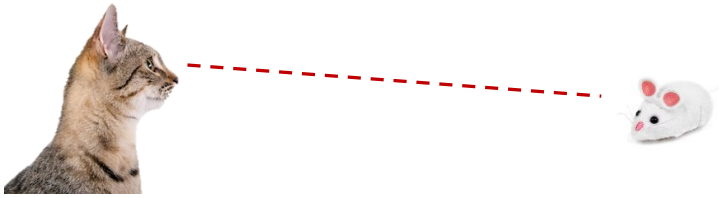
Building a stronger convolution-based feature extractor ??

Convolutions = Insights

More Convolutions = More Insights?

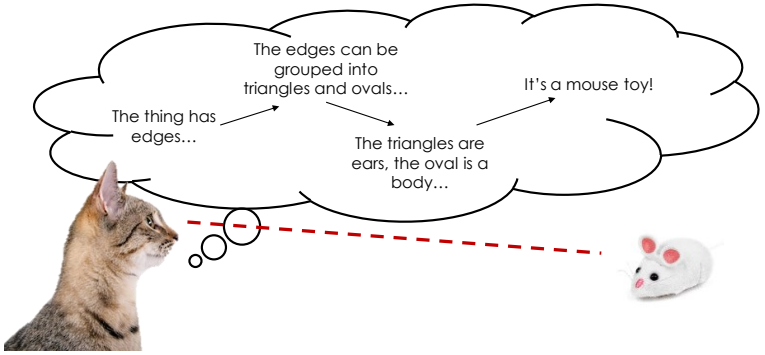
Recall Hubel and Weisel…

83



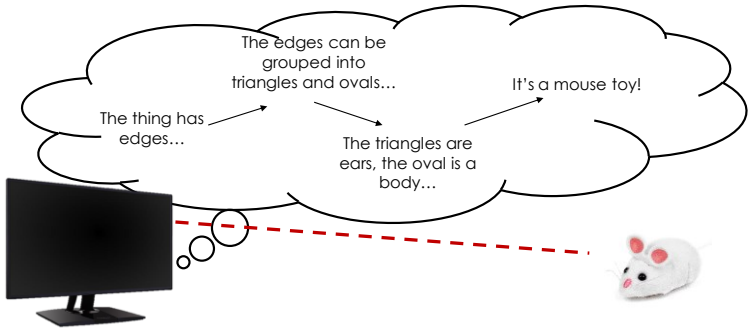
Recall Hubel and Weisel…

84



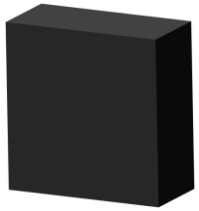
Recall Hubel and Weisel…

85

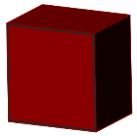


Convolutions Across Channels

86



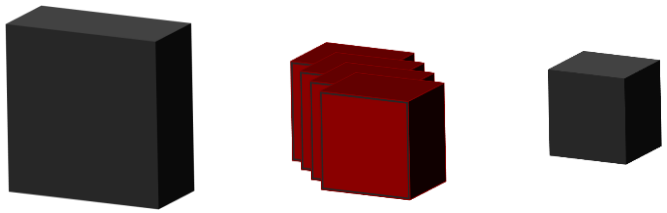
$28 \times 28 \times 3$
Image



$15 \times 15 \times 3$
Filter

$14 \times 14 \times 1$
Output

Convolutions Across Channels 87

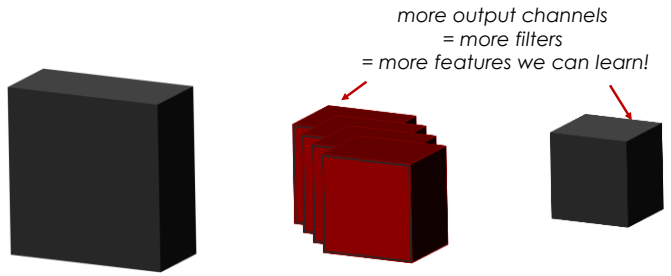


$28 \times 28 \times 3$
Image

$15 \times 15 \times 3 \times 4$
Filter

$14 \times 14 \times 4$
Output

Convolutions Across Channels 88



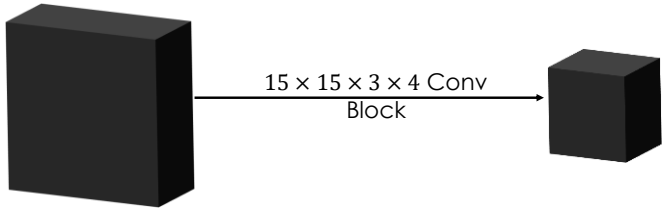
$28 \times 28 \times 3$
Image

$15 \times 15 \times 3 \times 4$
Filter

$14 \times 14 \times 4$
Output

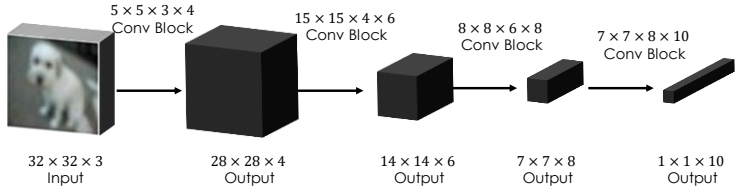
Convolutions Across Channels

89

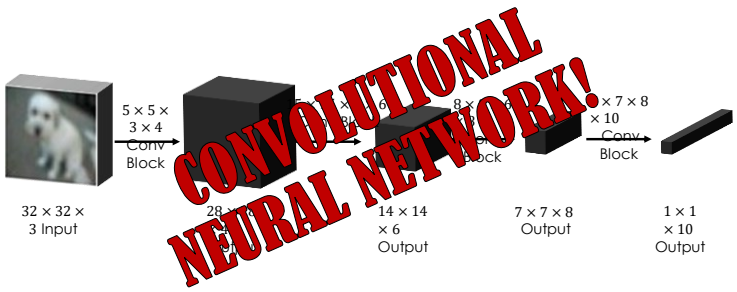


Stacking Convolutions

90



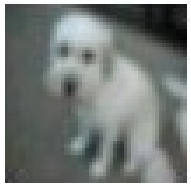
Stacking Convolutions



Convolutional Neural Networks (ConvNets)

- Neural networks which involve the stacking of multiple convolutional layers to produce output
- Often times end in fully-connected layers as the “classifier”

Why Do They Work So Well?

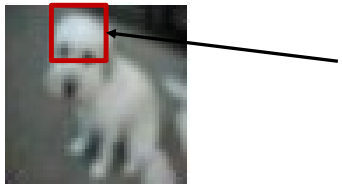


Why Do They Work So Well?

Why Do They Work So Well?

Why Do They Work So Well?

Why Do They Work So Well?



This is the neural network's "receptive field"—it's able to see!

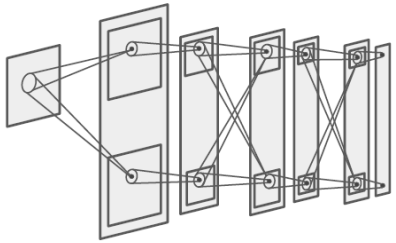
History of ConvNets

98

A bit of history:

Neocognitron
[Fukushima 1980]

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

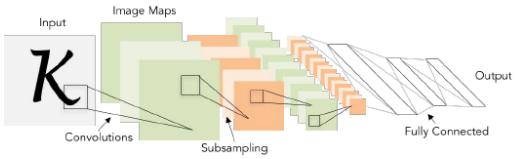


History of ConvNets

A bit of history:

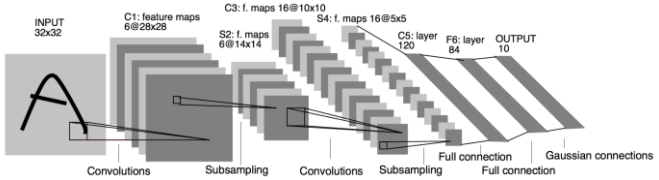
Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

History of ConvNets

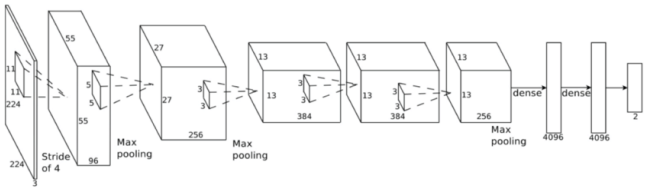


LeNet – 1998

MNIST is 0-9
0.7% test accuracy on MNIST

History of ConvNets

101

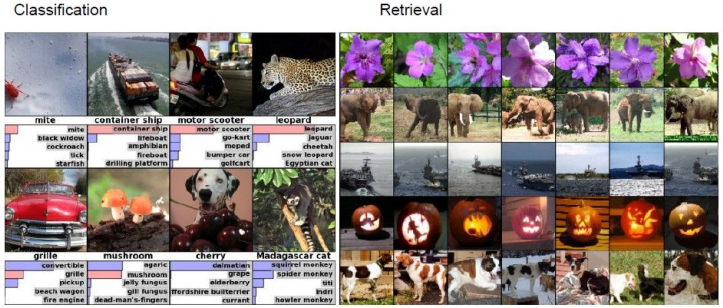


AlexNet – 2012

History of ConvNets

104

Fast-forward to today: ConvNets are everywhere

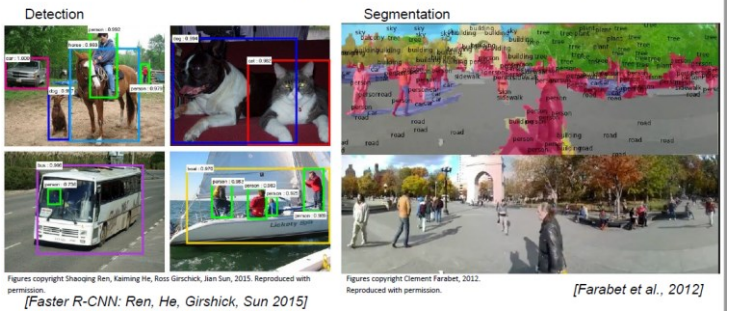


Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

History of ConvNets

105

Fast-forward to today: ConvNets are everywhere



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

History of ConvNets

106

Fast-forward to today: ConvNets are everywhere

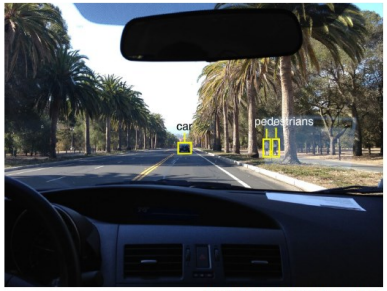


Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

This image by GBPublic_PR is licensed under CC-BY 2.0

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

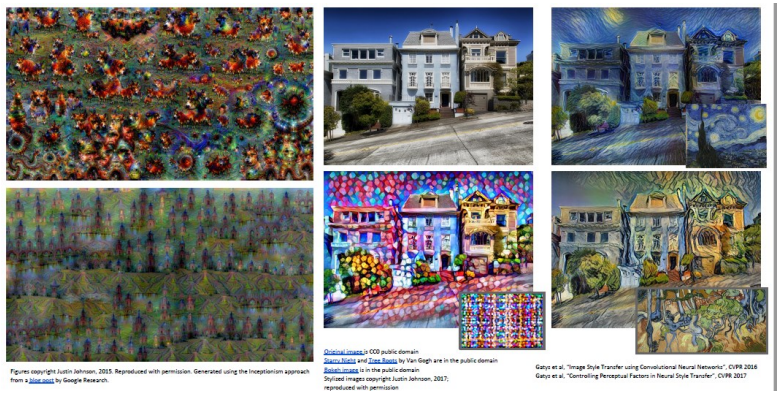
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

History of ConvNets

107

History of ConvNets

108

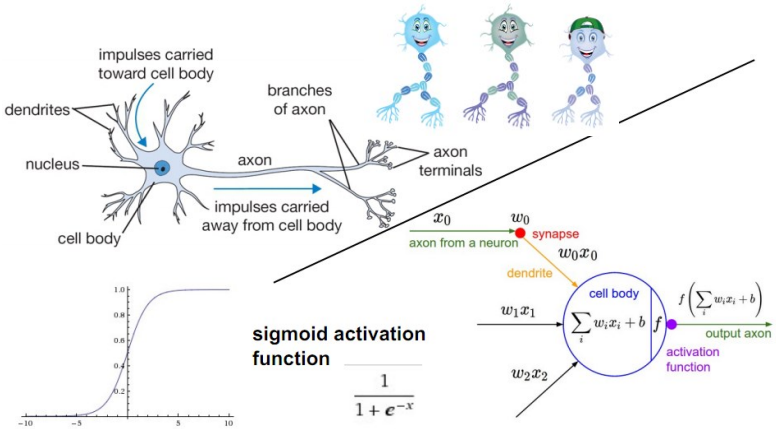


NN vs CNN

109

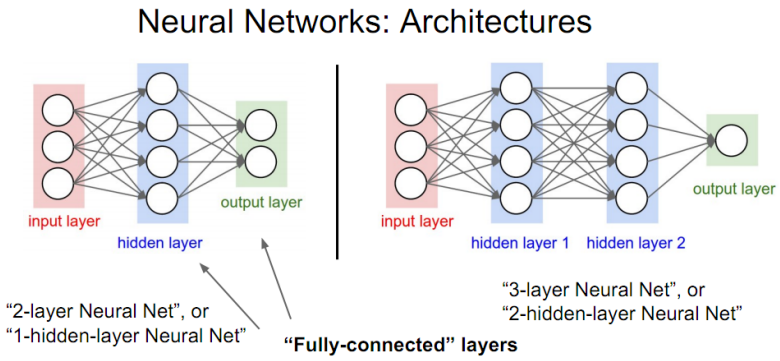
Neuron

110



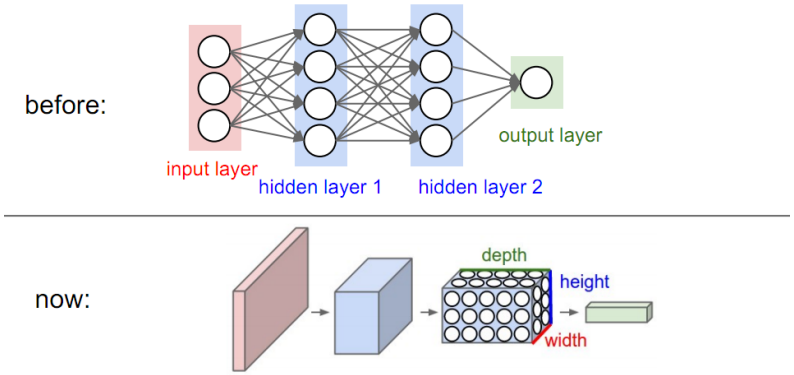
Neural Networks

111



ConvNets

112

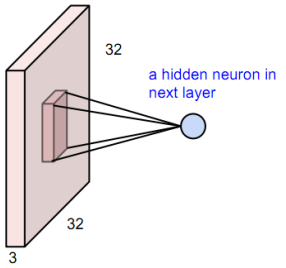


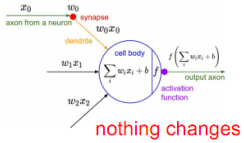
ConvNets

113

Convolutional Neural Networks are just Neural Networks BUT:

Local connectivity





nothing changes

image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:
- local in space (5x5 inside 32x32)
- but full in depth (all 3 depth channels)

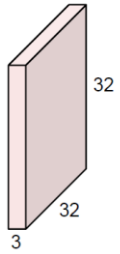
ConvNets

114

Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

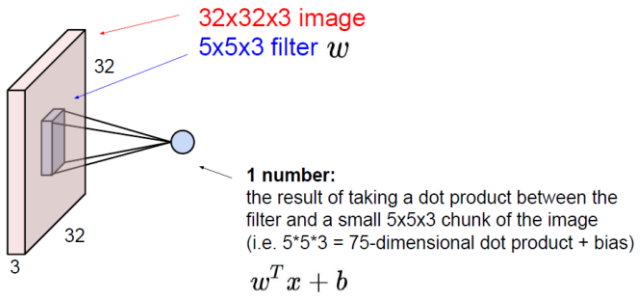


5x5x3 filter

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

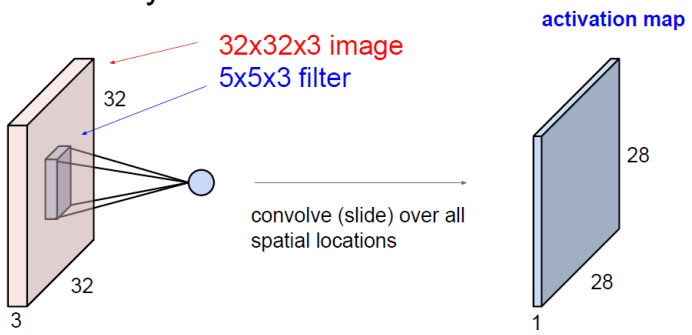
ConvNets

Convolution Layer



ConvNets

Convolution Layer

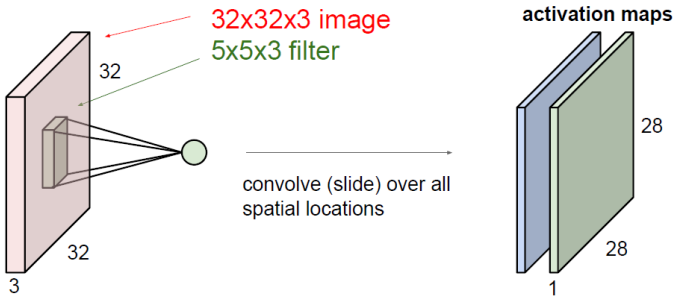


ConvNets

117

Convolution Layer

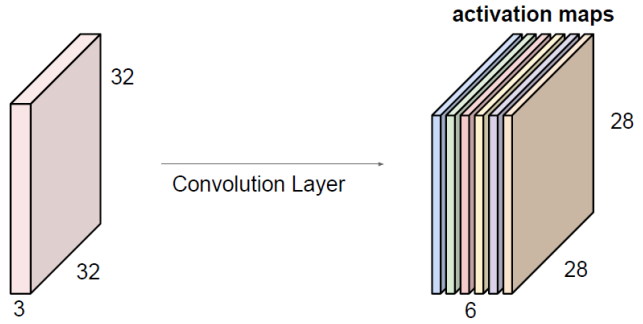
consider a second, green filter



ConvNets

118

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



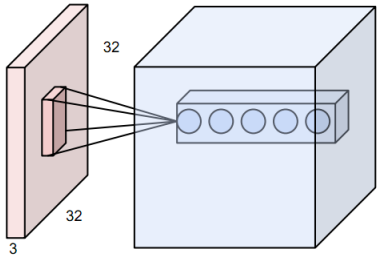
We stack these up to get a "new image" of size 28x28x6!

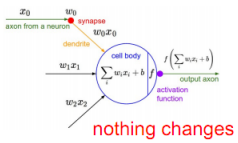
ConvNets

119

Convolutional Neural Networks are just Neural Networks BUT:

Local connectivity



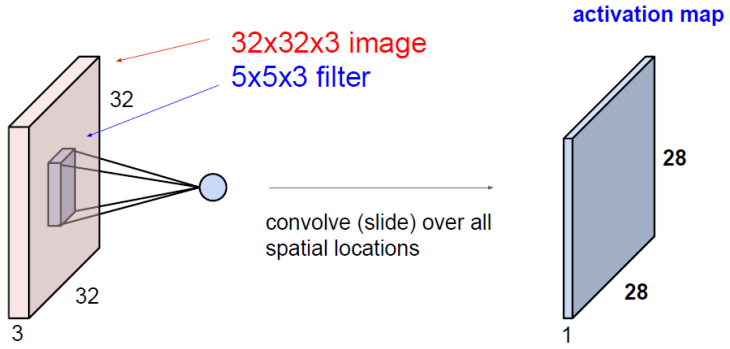


These form a single [1 x 1 x depth] "depth column" in the output volume

ConvNets

120

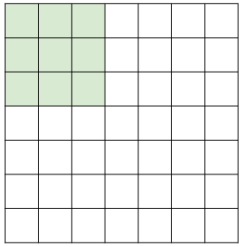
A closer look at spatial dimensions:



ConvNets

12
1

Replicate this column of hidden neurons across space, with some **stride.**

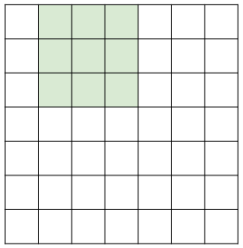


7x7 input
assume 3x3 connectivity, stride 1

ConvNets

12
2

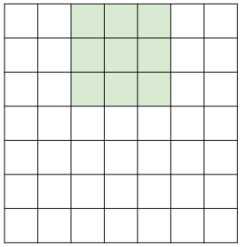
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets

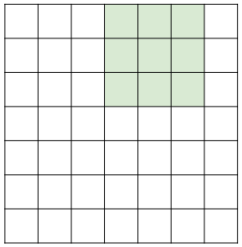
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets

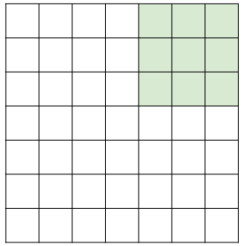
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets

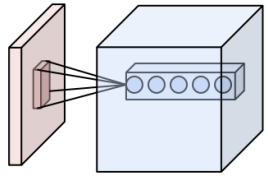
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

ConvNets

Examples time:



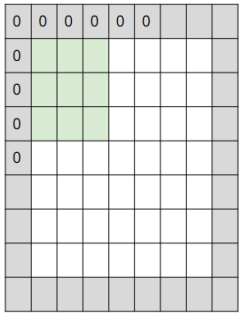
Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x5**
How many weights for each of the 28x28x5 neurons?

ConvNets

127

In practice: Common to zero pad the border



(in each channel)

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

7x7 => preserved size!

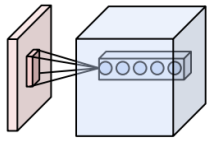
in general, common to see stride 1, size F, and zero-padding with (F-1)/2.
(Will preserve input size spatially)

ConvNets

128

There's one more problem…

Assume input **[32 x 32 x3]**
30 neurons with receptive fields **5x5**, applied at **stride 1/pad1:**
=> Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)
Each neuron has 5*5*3 (=75) weights
=> Number of weights in such layer: 30720 * 75 ~= **3 million :**



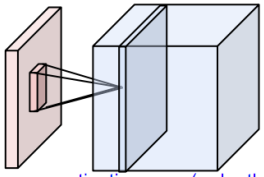
Example trained weights

IDEA: lets not learn the same thing across all spatial locations

ConvNets

These layers are called **Convolutional Layers**

1. Connect neurons only to local receptive fields
2. Use the same neuron weight parameters for neurons in each “depth slice” (i.e. across spatial positions)

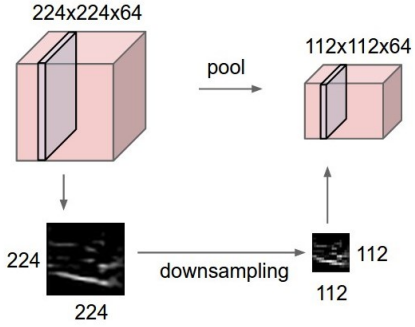


one activation map (a depth slice), computed with one set of weights

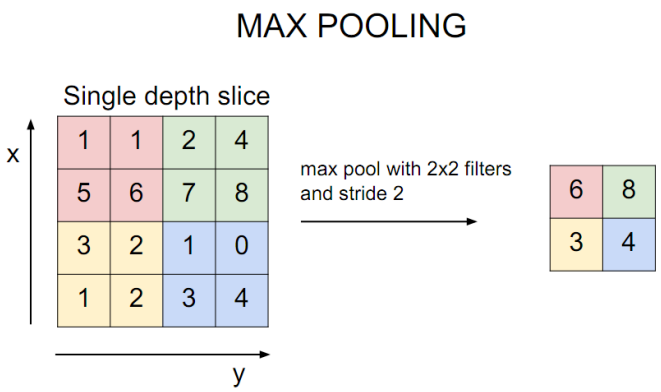
ConvNets

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

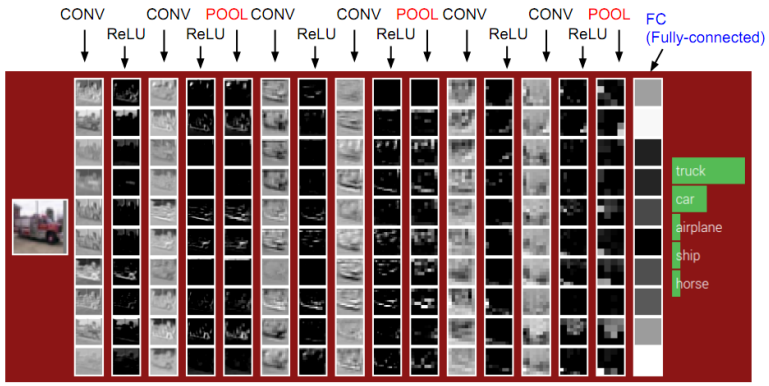
- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)



ConvNets



ConvNets



ConvNets

Modern CNNs:

- use **filter sizes of 3x3** (maybe even 2x2 or 1x1!)
- use **pooling sizes of 2x2** (maybe even less - e.g. fractional pooling!)
- **stride 1**
- **very deep**

INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
where the * indicates repetition, and the POOL? indicates an optional pooling layer.
N >= 0 (and usually N <= 3), M >= 0, K >= 0 (and usually K < 3).

ConvNets

Case study: VGGNet / OxfordNet
(runner-up winner of ILSVRC 2014)
[Simonyan and Zisserman]

best model

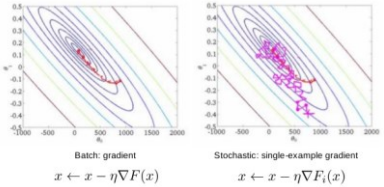
ConvNet Configuration

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

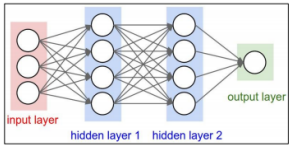
Training ConvNets



Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

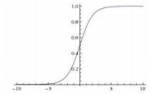


Training ConvNets

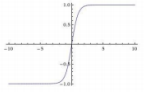
Activation Functions

Sigmoid

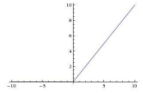
$\sigma(x) = 1/(1+e^{-x})$



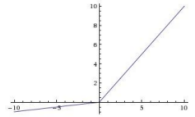
tanh tanh(x)



ReLU max(0,x)

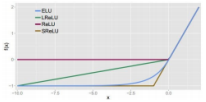


Leaky ReLU
max(0.1x, x)



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

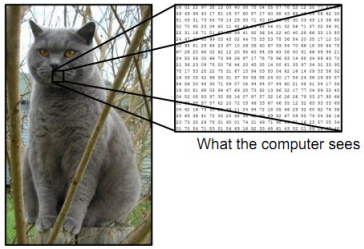
ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



Data Augmentation

Data Augmentation

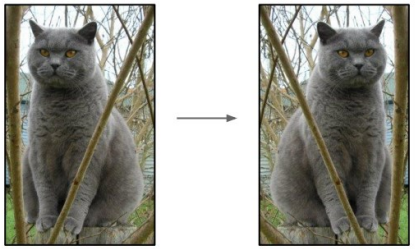
- i.e. simulating “fake” data
- explicitly encoding image transformations that shouldn’t change object identity.



Data Augmentation

Data Augmentation

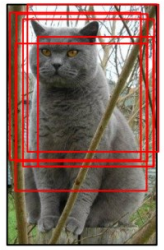
1. Flip horizontally



Data Augmentation

139

Data Augmentation
2. Random crops/scales



Sample these during training
(also helps a lot during test time)

e.g. common to see even up to 150 crops used

Data Augmentation

140

Data Augmentation
3.

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Data Augmentation

141

Data Augmentation

4. Color jittering

(maybe even contrast jittering, etc.)

- Simple: Change contrast small amounts, jitter the color distributions, etc.
- Vignette,... (go crazy)

Transfer learning

142

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

Transfer learning

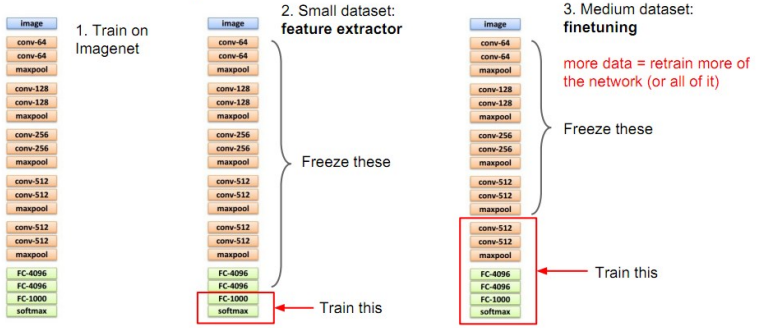
Transfer Learning

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

BUSTED

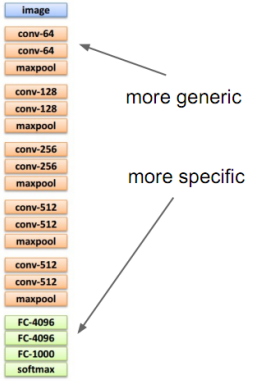
Transfer learning

Transfer Learning with CNNs



Transfer learning

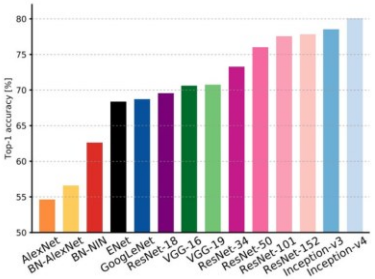
145

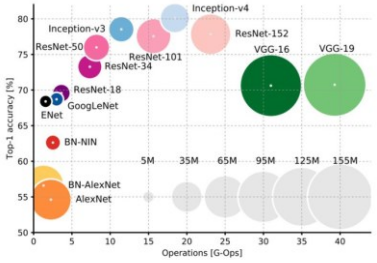


	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

CNN Performances

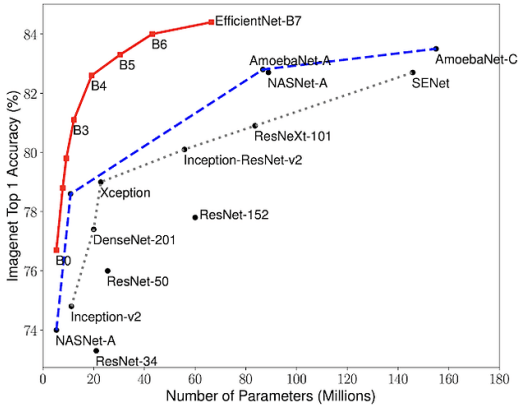
146





CNN Performances

147



EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling, 5/2019

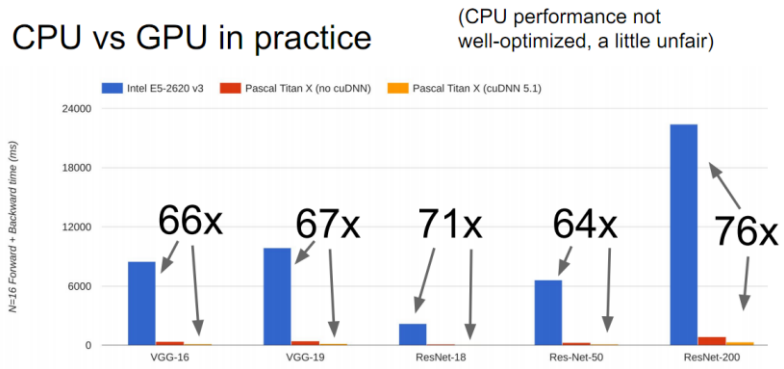
What is CNN dev?

148

- Define the objective
  - What is the input/output?
  - What is the loss/objective function?
- Create the architecture
  - How many conv layers?
  - What size are the convolutions?
  - How many fully-connected layers?
- Define hyperparameters
  - What is the learning rate?
- Train and evaluate
  - How did we do?
  - How can we do better?

CPU vs GPU

149



CPU vs GPU

150

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$339	~540 GFLOPs FP32
GPU (NVIDIA GTX 1080 Ti)	3584	1.6 GHz	11 GB GDDR5 X	$699	~11.4 TFLOPs FP32
TPU NVIDIA TITAN V	5120 CUDA, 640 Tensor	1.5 GHz	12GB HBM2	$2999	~14 TFLOPs FP32 ~112 TFLOP FP16
TPU Google Cloud TPU	?	?	64 GB HBM	$6.50 per hour	~180 TFLOP

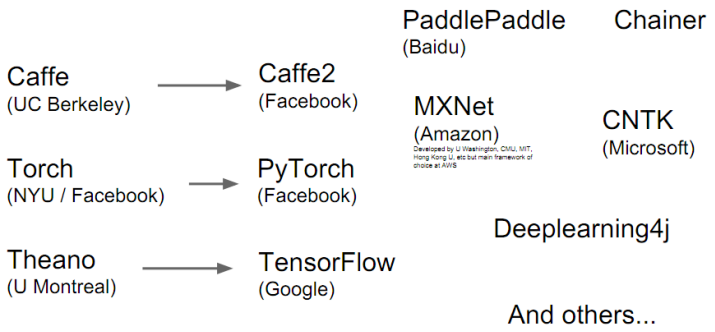
CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

TPU: Specialized hardware for deep learning

A zoo of frameworks

151



References

152

- Fei-Fei Li & Justin Johnson & Serena Yeung. *CS231n – 2019 - lecture 4, lecture 5*
- Shubhang Desai, Ranjay Krishna, and Juan Carlos Niebles. *Lecture: Computer Vision and Machine Learning*. CS131 Stanford Vision and Learning Lab