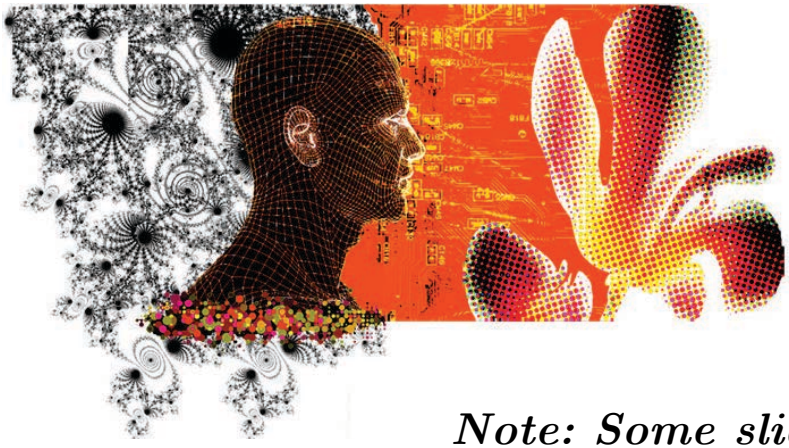
Data Science Training

November 2017

Convolutional Neural Networks

Xavier Bresson

Data Science and AI Research Centre
NTU, Singapore



http://data-science-optum17.tk

Note: Some slides are from Li, Karpathy, Johnson's course on Deep Learning.

Outline

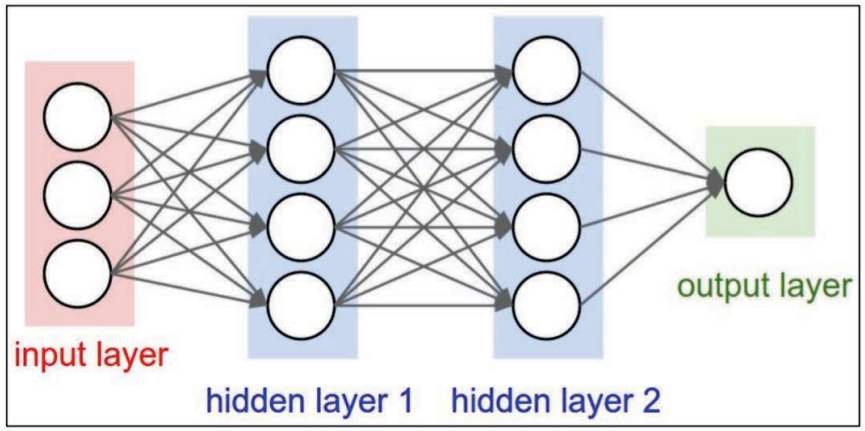
- Data Invariance and Structure
- History of CNNs
- CNNs
- Case Studies
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- Practical Considerations
- Conclusion

Outline

- **Data Invariance and Structure**
- History of CNNs
- CNNs
- Case Studies
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- Practical Considerations
- Conclusion

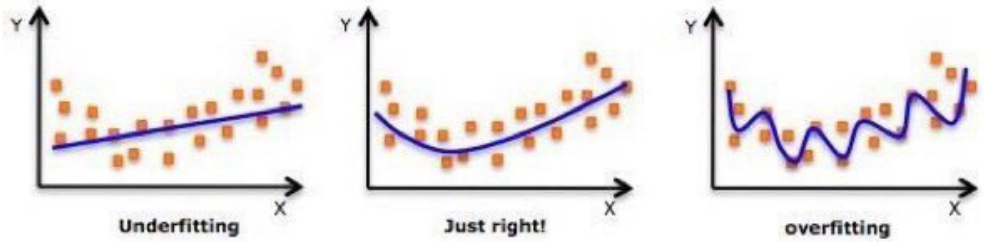
Fully Connected Networks

- FC networks do **not** consider any **structure** of data.



- **Theoretically** they can learn anything (universal approximation theorem), but they are **practically very hard to train** (too much parameters, too long).

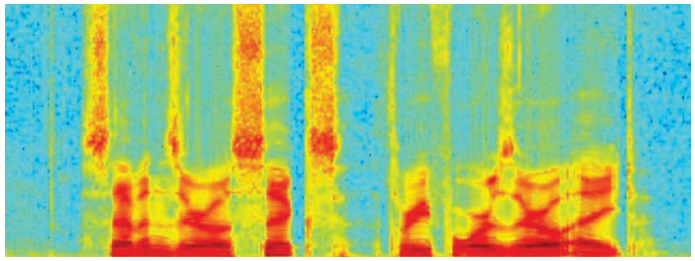
- They are pruned to **overfitting** and do **not generalize** well.



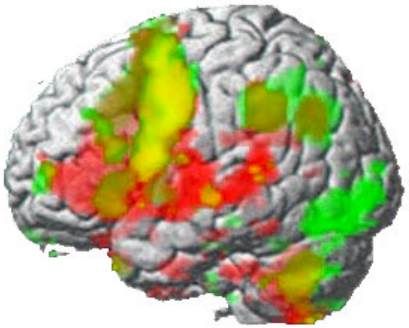
Efficient Neural Network Design

- Identifying data **invariance** is the best way to design NNs.
- **Invariance** refers to common data structure. Observe that data are not random, but exhibit special structures to find and leverage.

Image



Spectogram

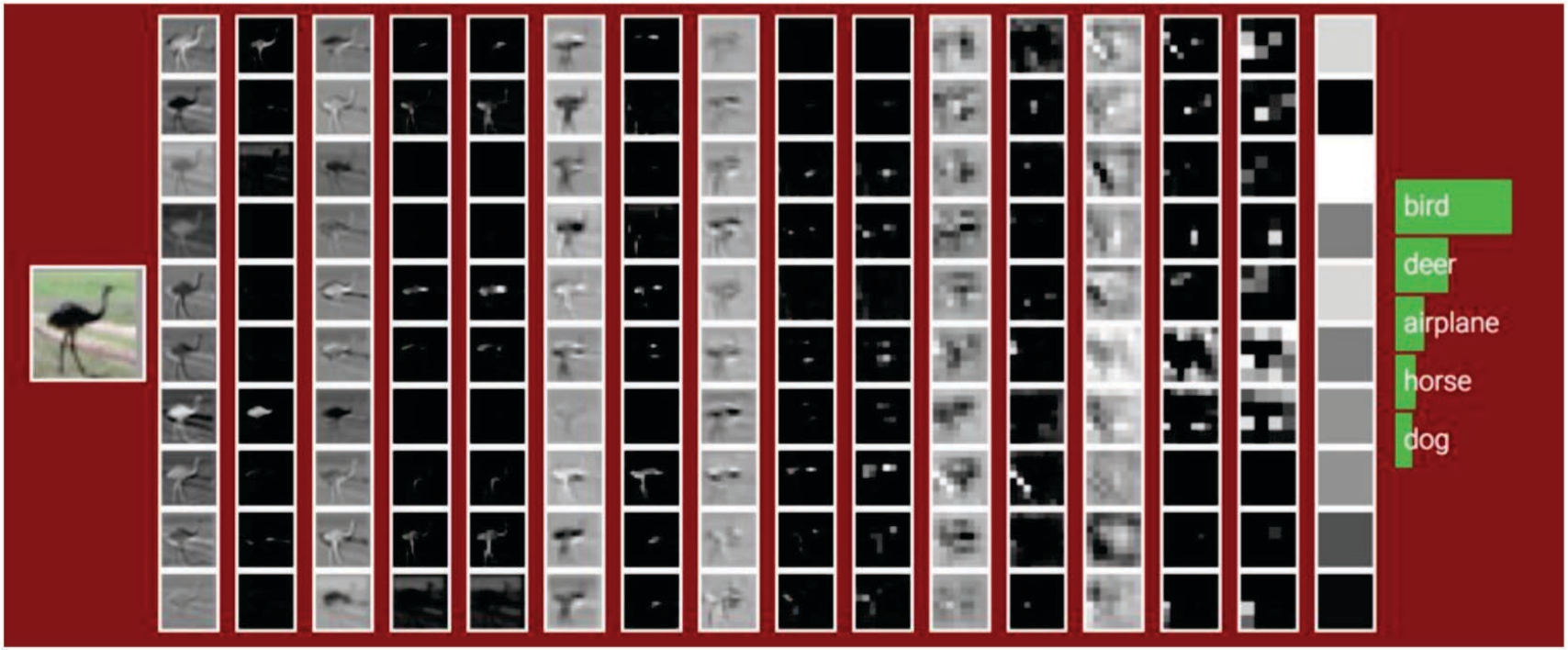


fMRI

Video

Neural Network for Computer Vision

- NNs for image analysis are fundamentally built on the **compositionality** invariance of images.



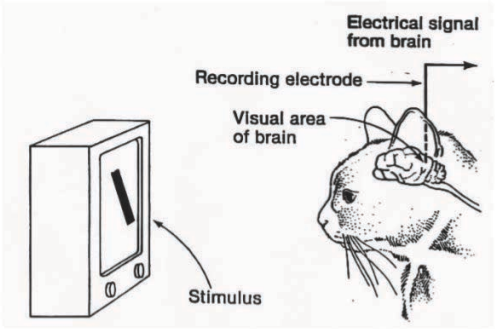
Outline

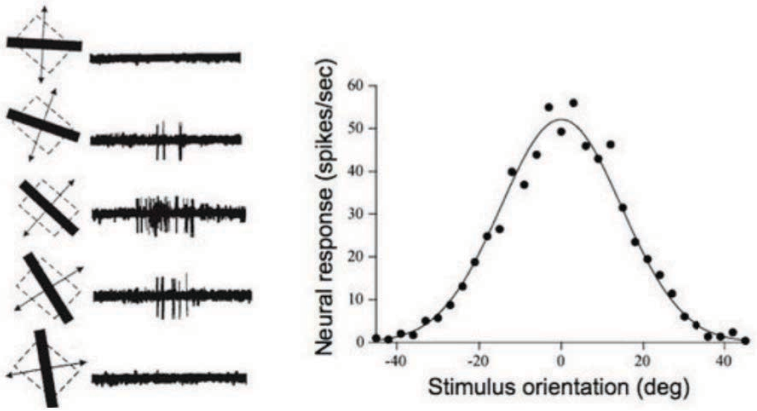
- Data Invariance and Structure
- **History of CNNs**
- CNNs
- Case Studies
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- Practical Considerations
- Conclusion

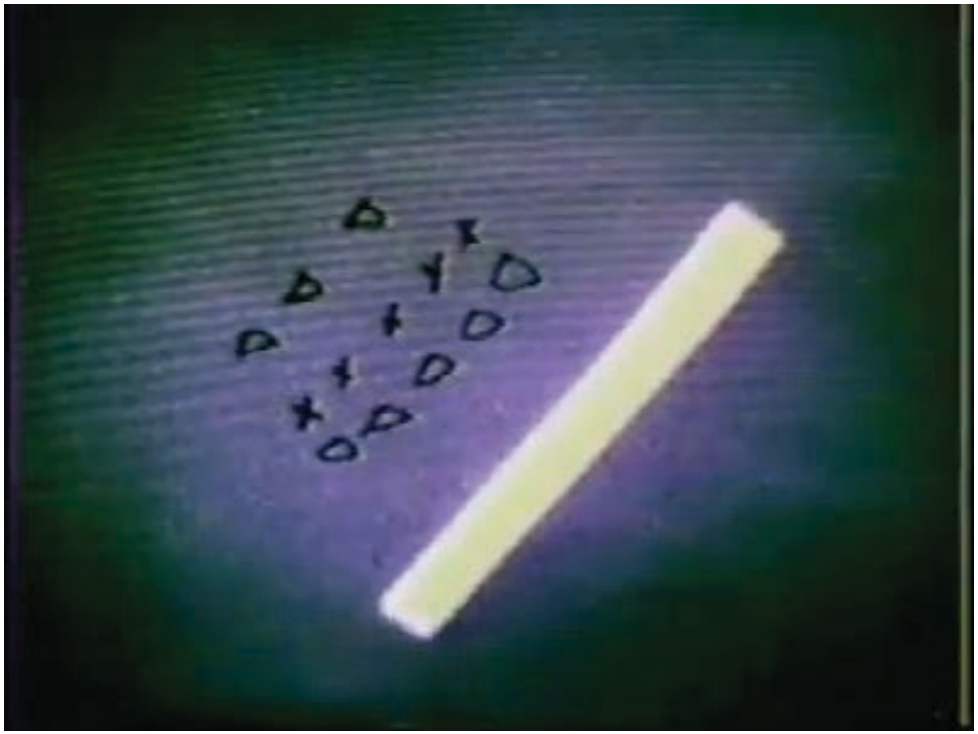
A Bit of History

- **Hubel and Wiesel: Nobel Prize in Medicine** in 1959 for the understanding of the primary visual cortex system (first 2 layers).

- Visual system is composed of **receptive fields** called *V1 cells* that are composed of **neurons that activate depending on the orientation.**

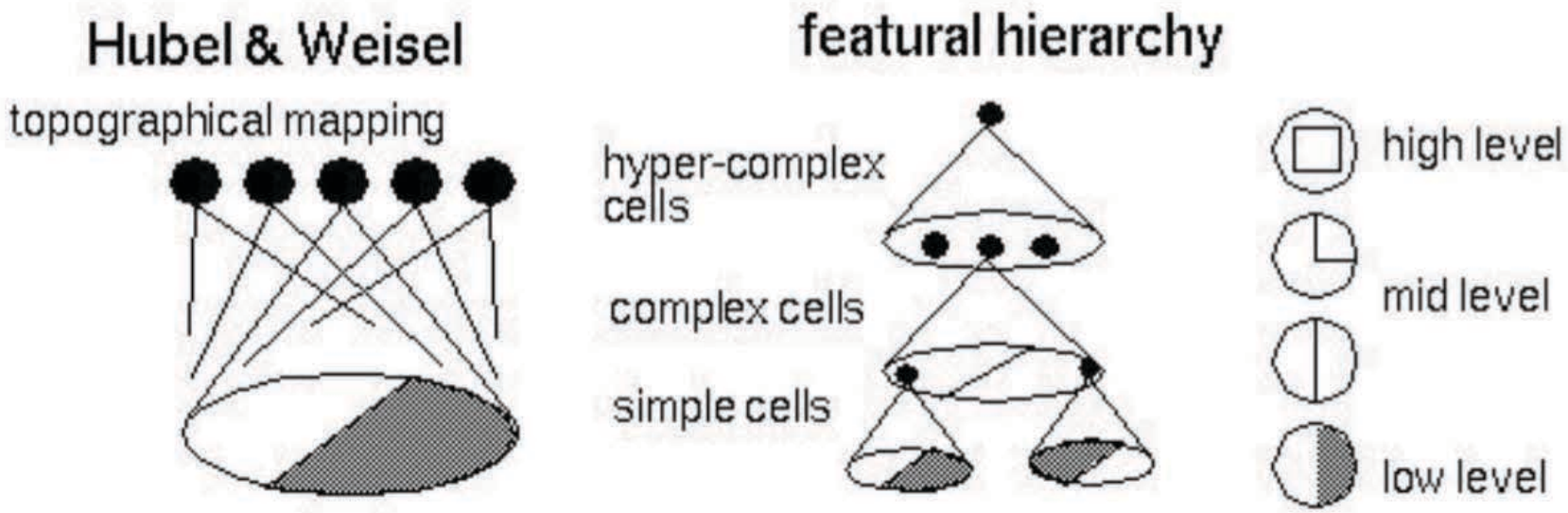






Hierarchical Organization of Visual Neurons

- The **second layer** of the visual cortex is composed of *V2 cells* that takes as inputs the outputs of V1 neurons ⇒ This forms a **hierarchical organization**.



Perceptron [Rosenblatt'57]

- **Application:** *Character recognition*

  Perceptron is only **hardware** (circuits, electronics), no code/simulations.

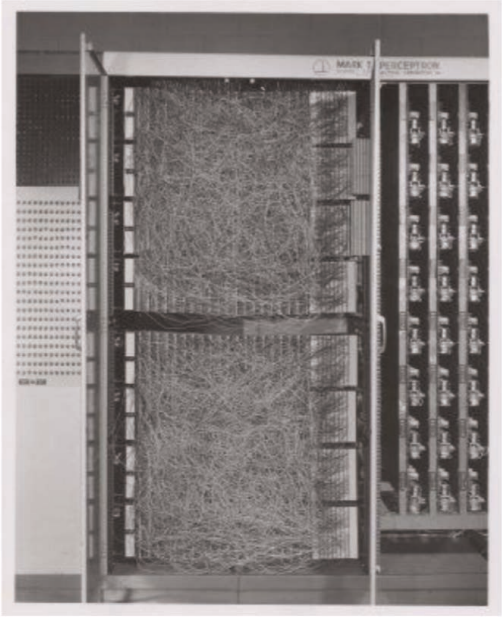
  Perceptron was connected to a camera that produced 400-pixel images.

  Update rule was empirical: $W^{t+1} = W^t + \alpha(D - Y^t)X$

  Activation was binary: $\sigma(x) = \begin{cases} 1 & \text{if } \langle w, x \rangle + b > 0 \\ 0 & \text{otherwise} \end{cases}$

  No concepts of loss function, no gradient, no backpropagation $\Rightarrow$ Learning was bad.

  Multilayer perceptron in 1960: stack perceptron, still hardware.



Neurocognitron [Fukushima'80]

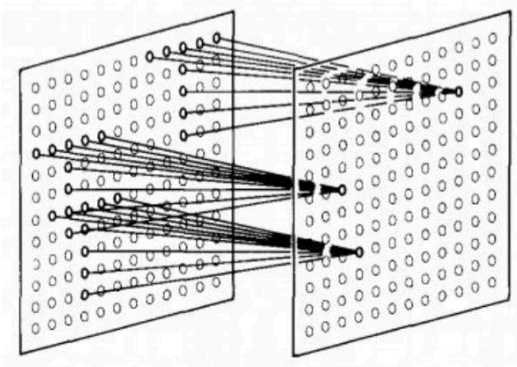
- **Application:** *Handwritten character recognition*

  Direct implementation of Huber-Wiesel simple and complex cells (V1 and V2 cells) with hierarchical organization.

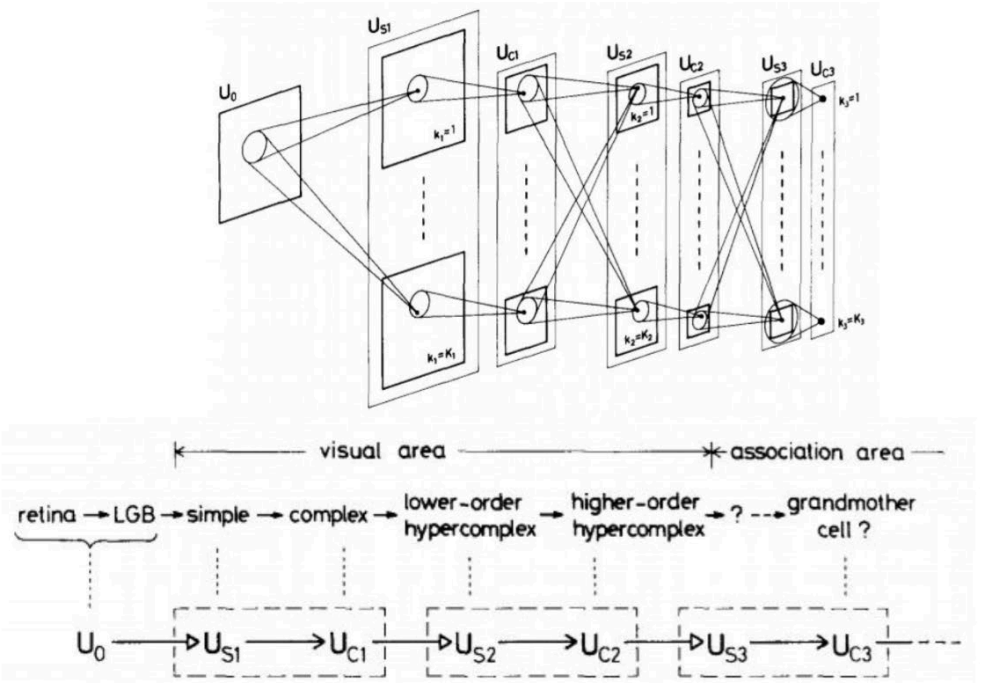
  Introduction of concepts of **local features (reception fields)**.

  No concepts of loss function, no gradient, no backpropagation ⇒ Learning was bad.

  Strong inspiration for convolutional neural networks (CNNs)

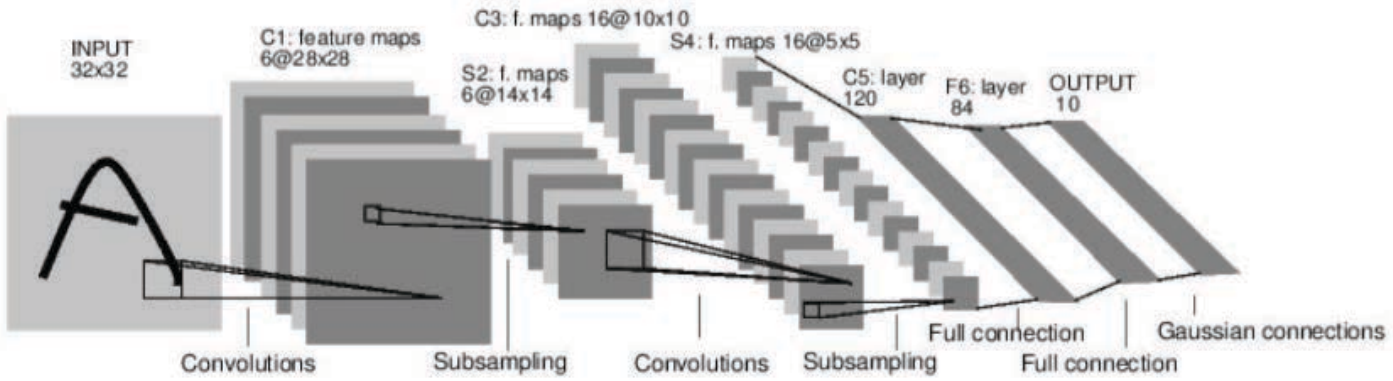


Complex cells: perform pooling



Convolutional Neural Networks (CNNs) [LeCun-Bengio-et.al'98]

- **Among top 10 algorithms** in data science



LeNet-5

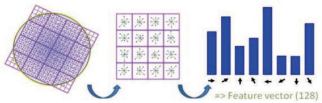
- **Computational issue in 1998:** Very long to train for large-scale/deep networks.

LeNet5 vs. SVM

- SVM-based approach:

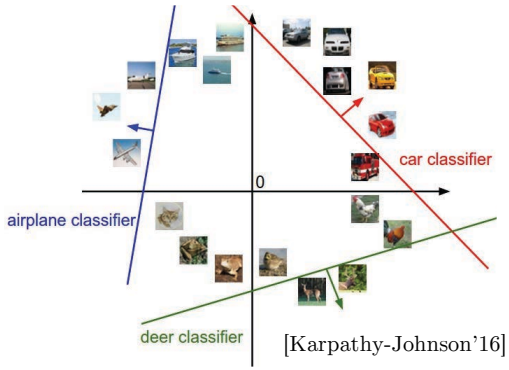
Input image

Handcrafted Features



SIFT, Lowe, 1999
(most cited paper in CV)

SVM classifier

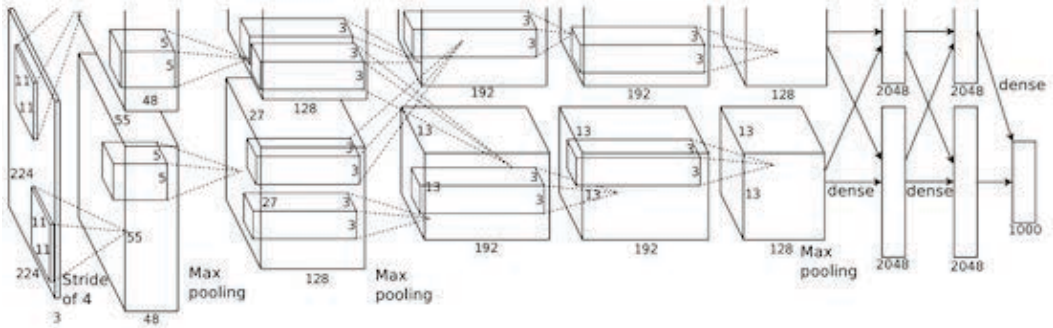


[Karpathy-Johnson'16]

- LeNet5-based approach:

Input image

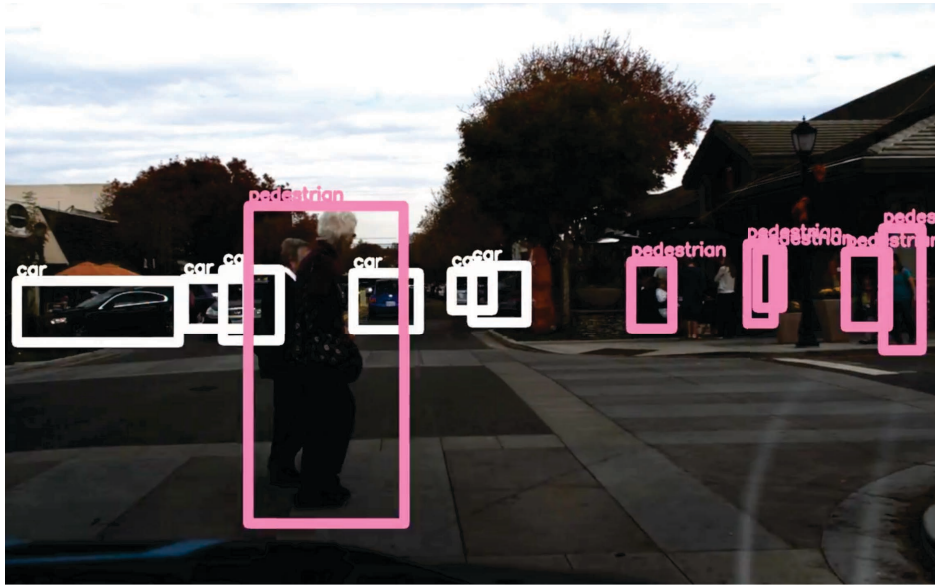
Features + Classifier



LeNet5 vs. SVM

- SVM-based approach:

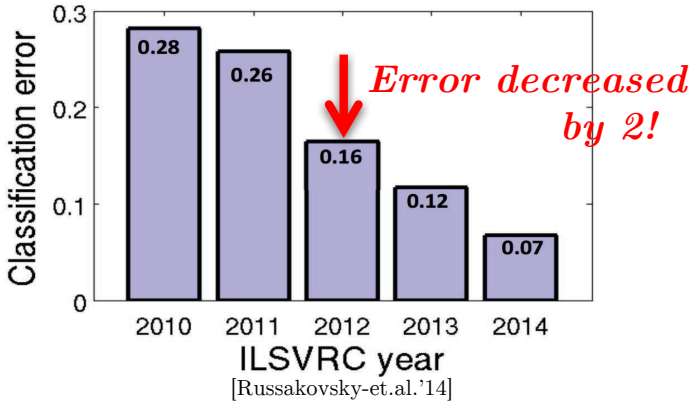
- LeNet5-based approach:



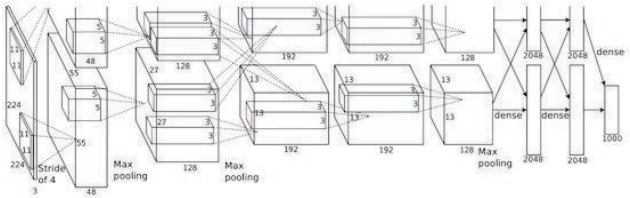
2012: Deep Learning Breakthrough

- **ImageNet [Fei Fei-et.al.'09]:** International Image Classification Challenge
  1,000 object classes and 1,431,167 images

[L. Fei-Fei-et.al.'09]



[Russakovsky-et.al.'14]



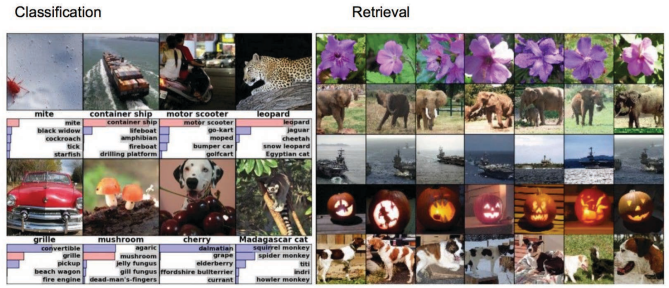
AlexNet
[Krizhevsky-et.al'12]

GPU

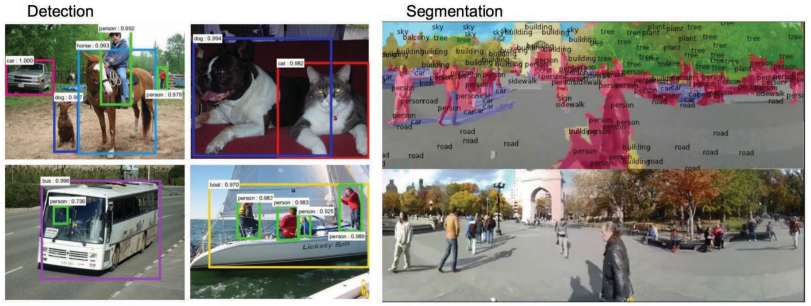
- **Before 2012:** Handcrafted features + SVM classification
  After 2012: Learn filters and classifier with neural networks (**end-to-end system**).

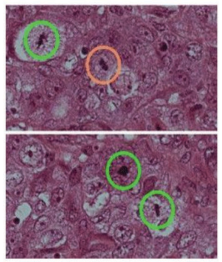
CNNs are Everywhere in Computer Vision

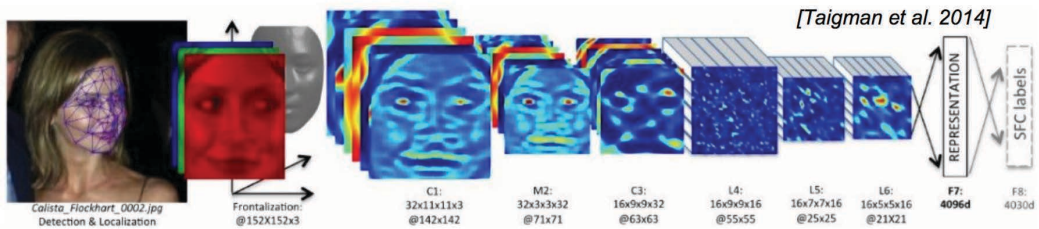
- Classification, Retrieval
- Detection, Segmentation
- Self-driving car
- Face detection
- Medicine
- Go game
- Arts/deep dreams

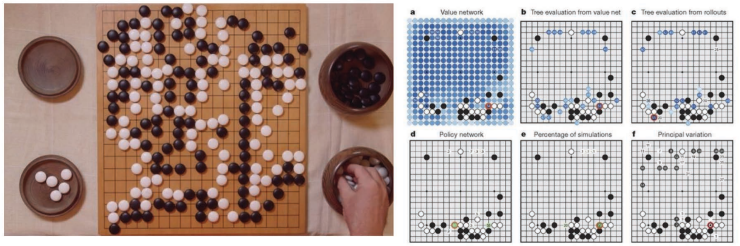


self-driving cars







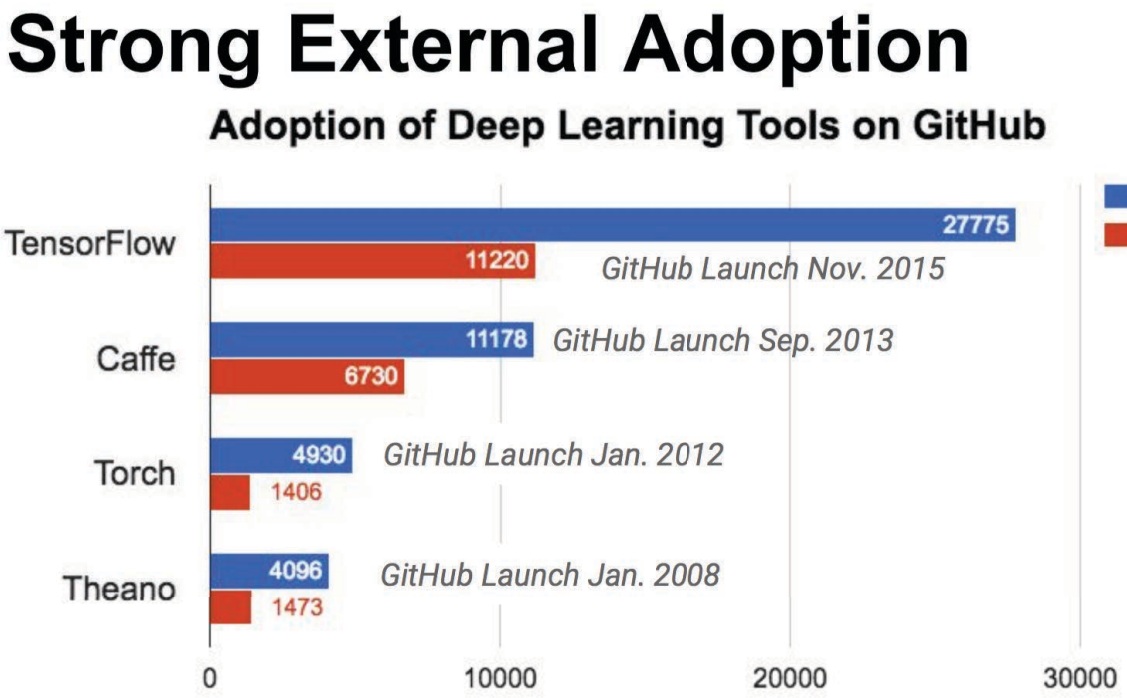


CNNs are Used by Top IT Companies

- Facebook (PyTorch software)  
- Google (TensorFlow software, Google Brain, Deepmind)
- Microsoft    
- Tesla (AI Open) 
- Amazon (DSSTNE software)  
- Apple 
- IBM 

TensorFlow, 2016

- Open, standard software for general machine learning
  Great for Deep Learning in particular
  First released Nov 2015 Apache 2.0 license



[Jeffrey Dean]

50,000+ binary installs in 72 hours, 500,000+ since November, 2015
Most forked new repo on GitHub in 2015 (despite only being available in Nov, '15)

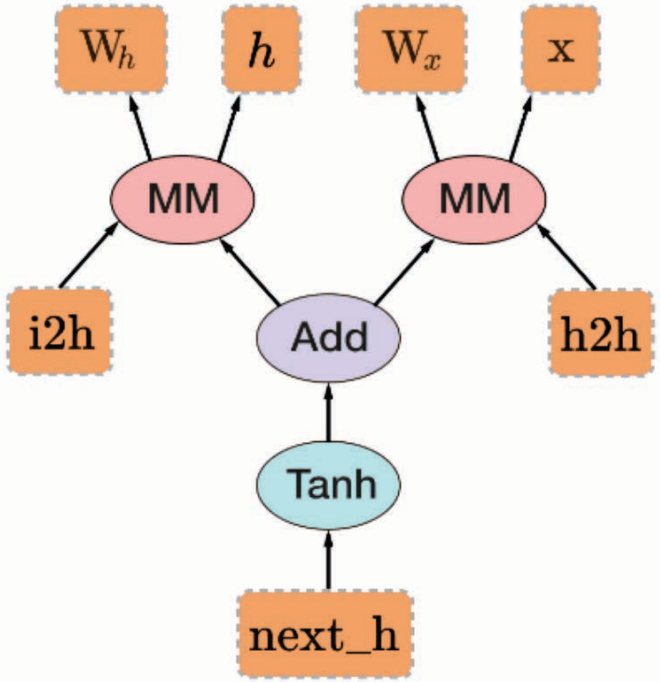
PyTorch, 2017

- Dynamic computational graph:

PYTORCH

A graph is created on the fly

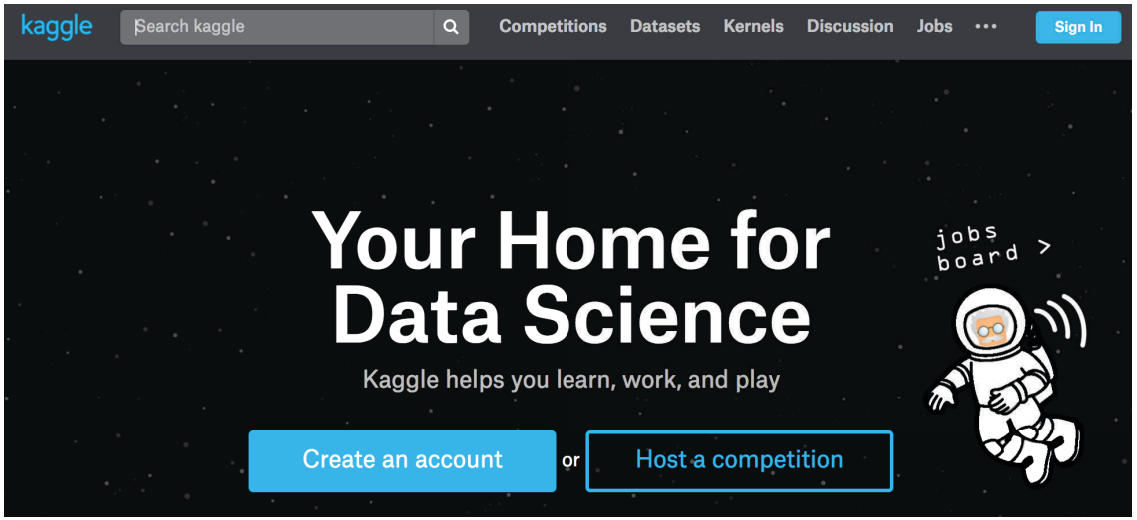
```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()
```



Keras, 2017

- **High-level software**, uses in back-end Tensorflow, Theano, Caffee, Deeplearning4j.

- Most **Kaggle** competitions won by deep learning used Keras.



Outline

- Data Invariance and Structure
- History of CNNs
- **CNNs**
- Case Studies
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- Practical Considerations
- Conclusion

Convolutional Neural Networks [LeCun-Bengio-et.al'98]

- CNNs are **extremely efficient** at extracting meaningful statistic patterns in large-scale and high-dimensional image datasets.

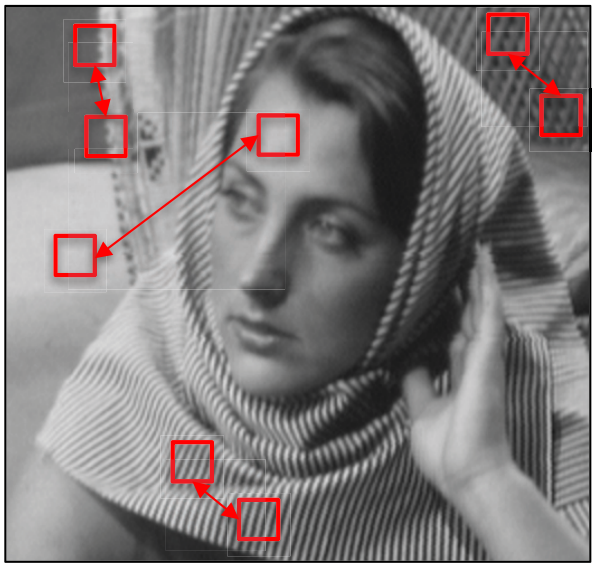
- **Key idea:** Learn data **compositionality invariance**, that are **hierarchical local stationary structures** (later explained).

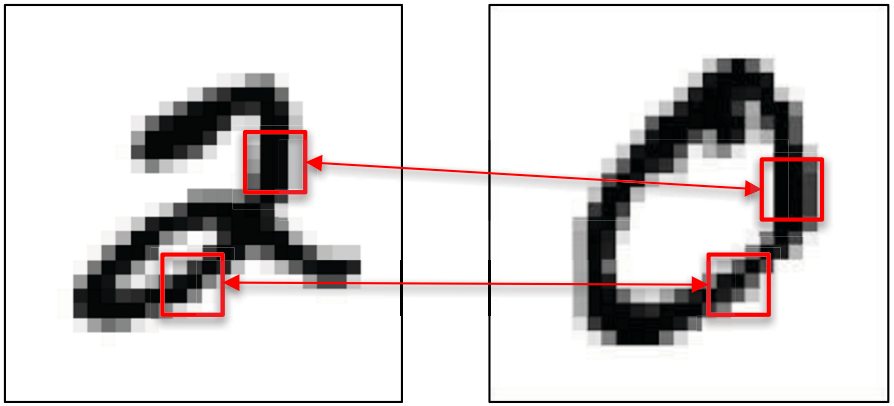
- **Why CNNs are good?** It is open (math) question to prove the performances of CNNs.

  Note: Despite the lack of theory, the entire ML and CV communities have shifted to deep learning techniques. Ex: NIPS'16: 2,326 submissions, 328 DL (14%), Convex Optimization 90 (3.8%).

Local Stationarity

- **Assumption:** Data are **locally stationary** ⇔ Similar **local patches** are shared across the data domain:

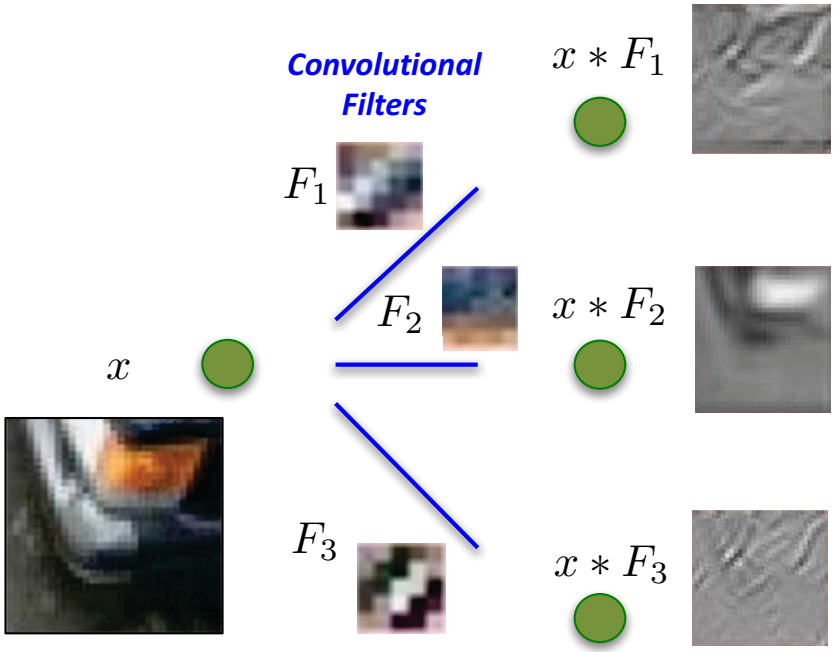




Convolutional Layer

- **How to extract local stationary patterns?**

  Convolutional layers (filters with compact support kernels)



Convolution at pixel *(i,j)*:

$$\big(x * F\big)(i,j) = \sum_{t_i,t_j} x(i,j).F(i-t_i, j-t_j)$$

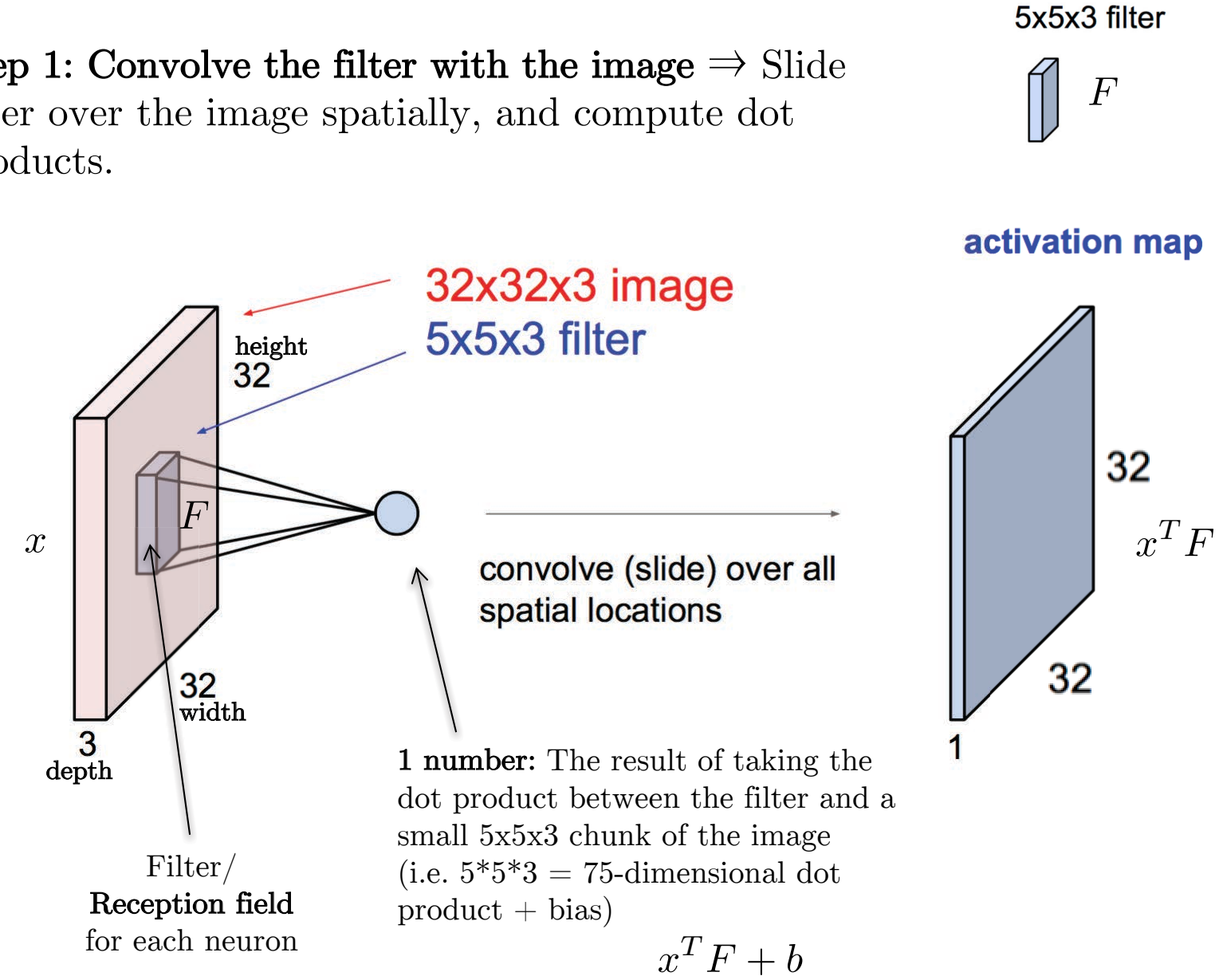
At pixel *(i,j)*, simple linear **dot products**:

$$x^T F$$

⇒ **Very fast on GPUs: O(1).**

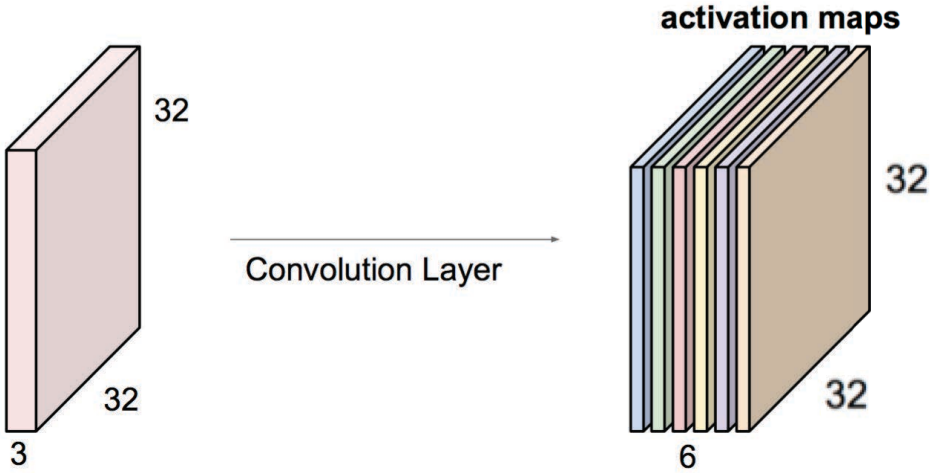
Convolutional Layer

- **Step 1: Convolve the filter with the image** $\Rightarrow$ Slide filter over the image spatially, and compute dot products.

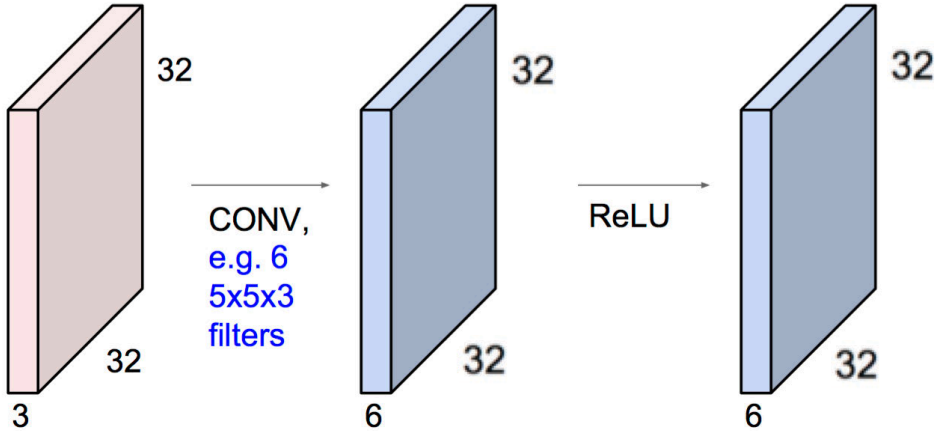


Convolutional Layer

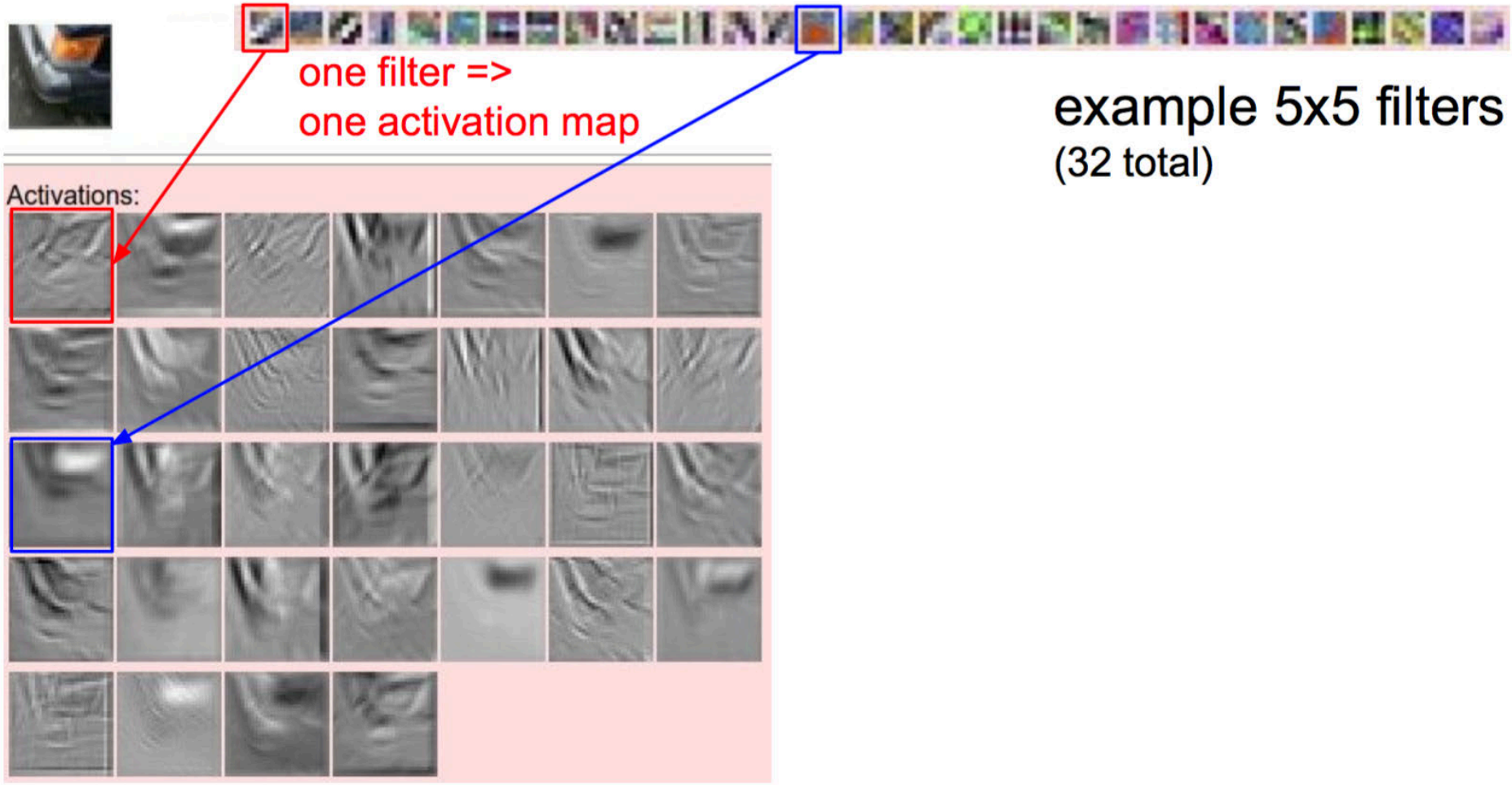
- **Step 2: Produce a stack of activation maps** ⇒ Using 6 5x5 filters, we get 6 separate activation maps that we stack up to get a "feature image" of size 28x28x6.



- **Step 3: Apply a non-linear activation function**, like ReLU: ***max(0,.)***



Activation Maps from Convolutional Layer



Hierarchical Features

- **Assumption:** Local stationary patterns can be composed to **form more abstract complex patterns:**

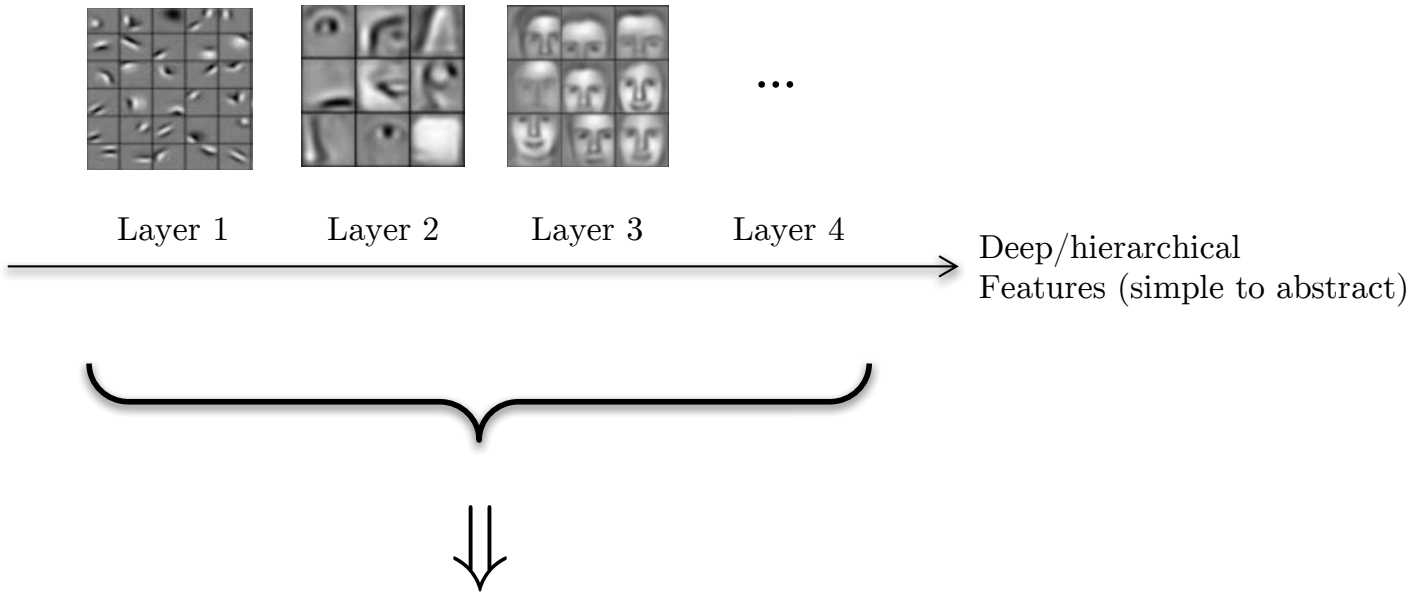
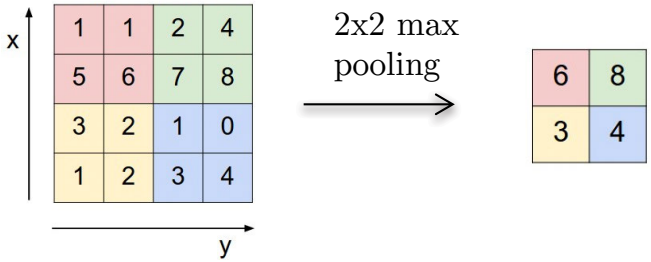
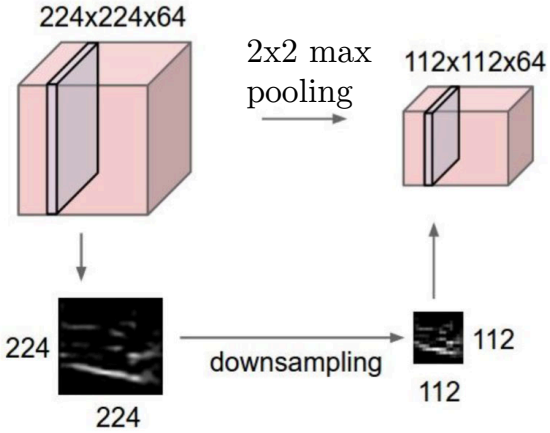


Image data are **compositional** ⇒ They are formed from **hierarchical local stationary patterns**.

Downsampling and Pooling

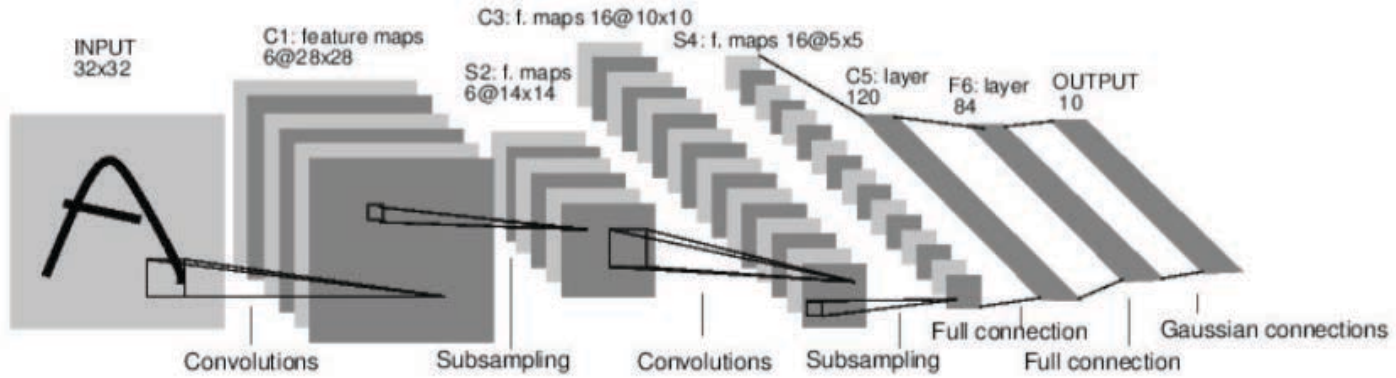
- **How to extract multiscale hierarchical patterns?**
  ⇒ **Downsampling** of data domain (s.a. image grid)
  ⇒ **Pooling** (s.a. max, average)





Trading Spatial Resolution with Feature Resolution

- **Other key advantage of downsampling:** Keeping same computational complexity while increasing #filters (that capture data structure).



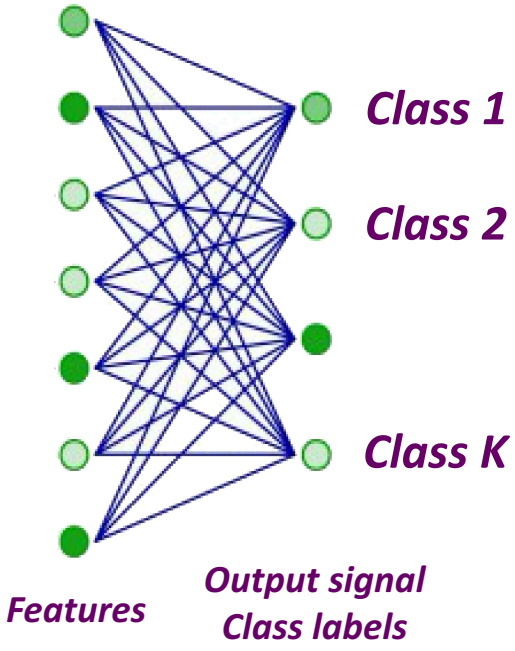
LeNet-5

Classifier

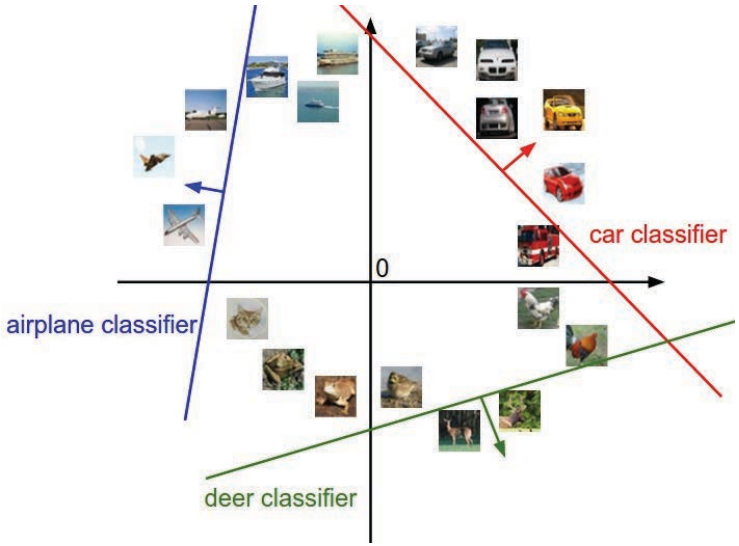
- **Classifier:** After extracting multiscale locally stationary features, **feed them to a classification function** with the training labels.

- How to design a (linear) classifier?

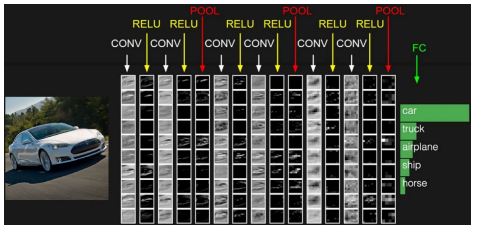
  $\Rightarrow$ Fully connected neural networks



$$x^{out} = W x^{layer}$$



Architecture of CNNs



Convolutional layer

Non-linear activation + Downsampling + Pooling

Compositional layer

Filters

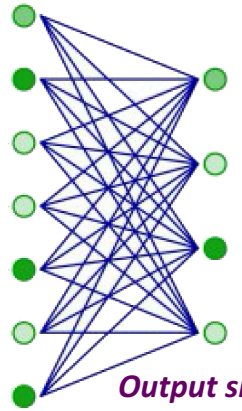
F_1 F_2 F_3

$x * F_1$ $x * F_2$ $x * F_3$

x

E.g. image

Cascade of compositional layers



Output signal

Class labels

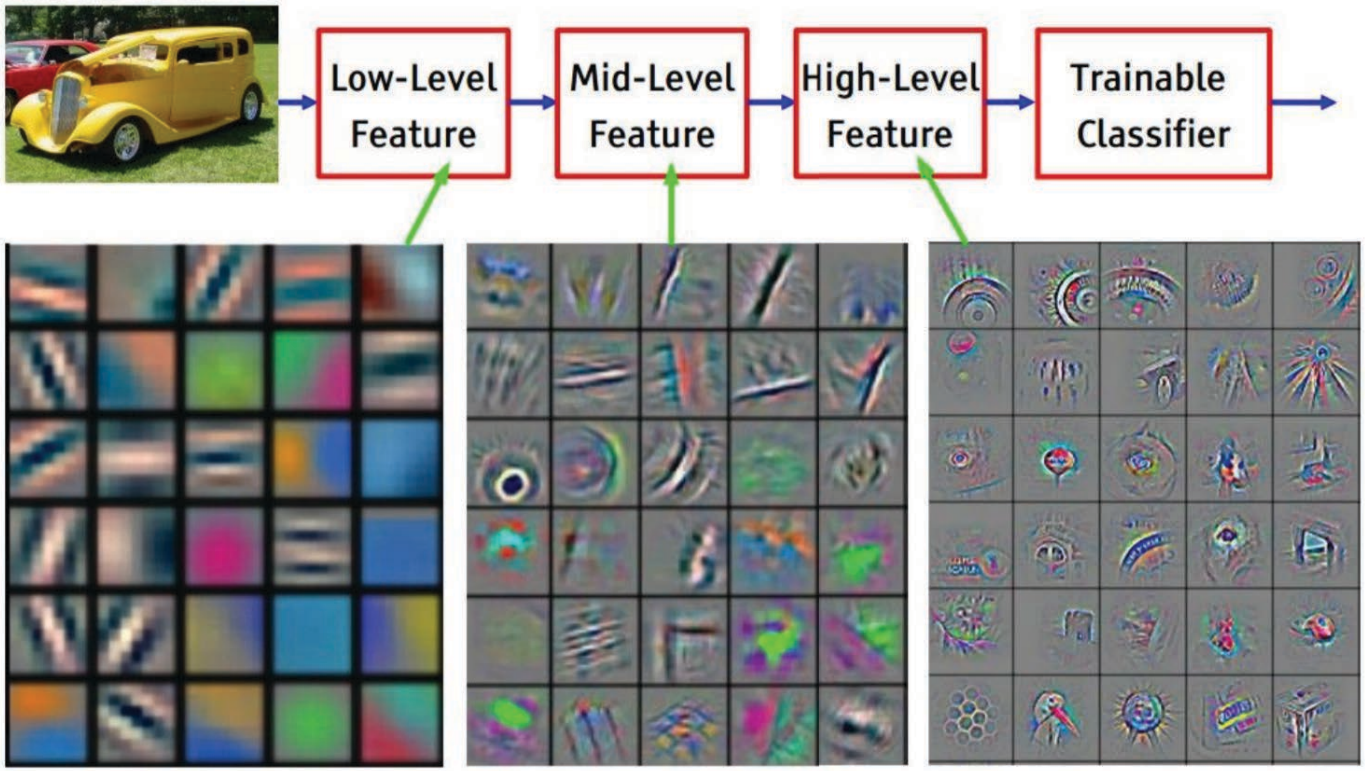
$x^{l=0} = x$ $x^{l=0}_{\text{conv}}$ $x^{l=1}$ $x^{l=1}_{\text{conv}}$ x^{l} $y \in \mathbb{R}^{n_c}$

Input signal

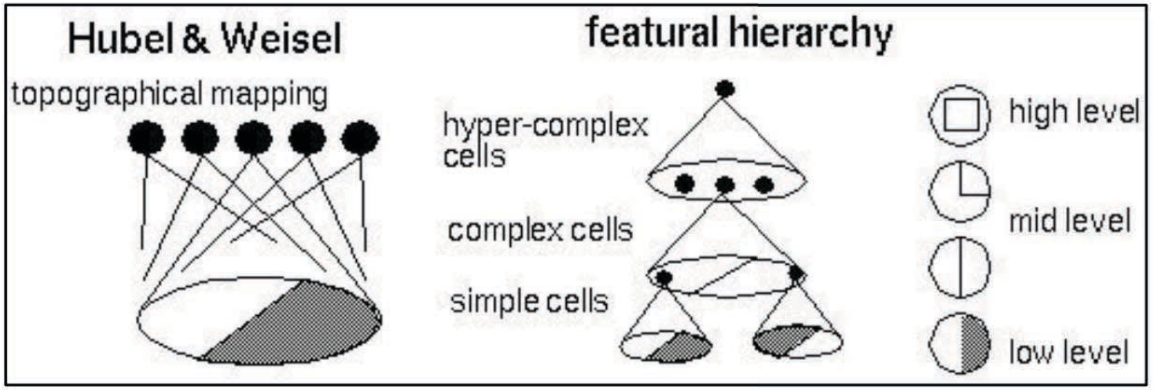
Compositional layers (extract multiscale local stationary features)

Fully connected layers (Classification function)

Visualizing CNN Layers



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



Demo: LeNet5

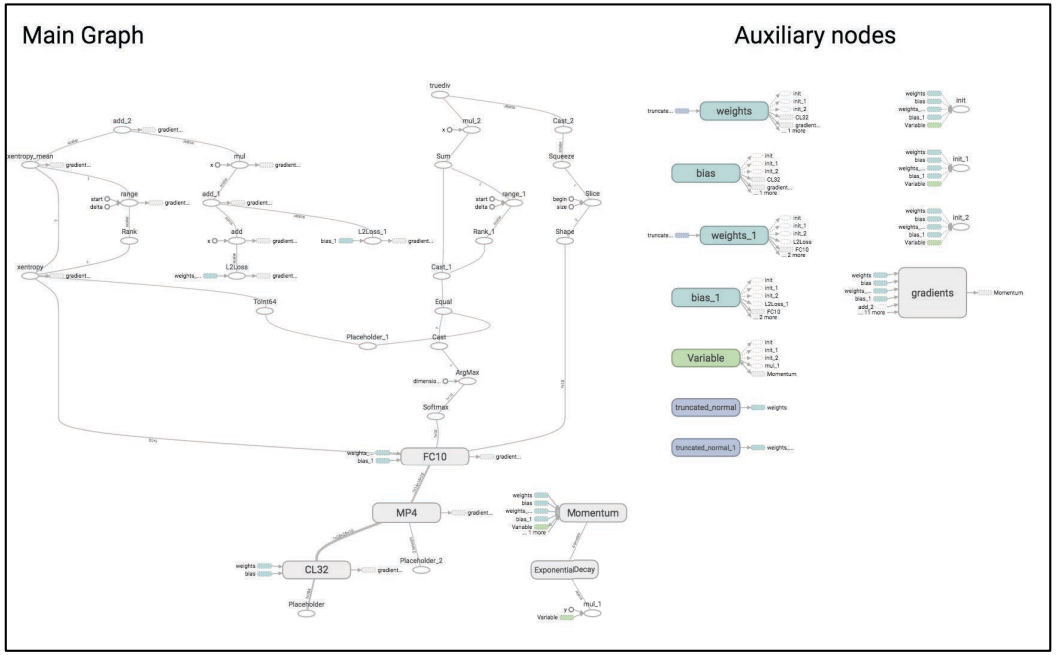
- Run **code01.ipynb**

LeNet5: CL32-MP4-CL64-MP4-FC512-FC10

```
F1=32 # Number of features for 1st CL layer
F2=64 # Number of features for 2nd CL layer
FEAT2 = 7*7* F2
FC1=512 # Number of nodes for 1st FC layer

class CNN_LeNet5(base_model):

    def __init__(self, K):
```

```
print('Training time: {:.2f}s'.format(time.process_time() - t_start))
```

```
num_epochs= 2 , nb_iter= 1100
iter=0, freq_iter=10, training time: 0.00s, acc_train=-1.00, loss_train=-1.00
iter=10, freq_iter=10, training time: 1.82s, acc_train=28.00, loss_train=2.23
iter=100, acc_train=86.00, loss_train=0.57, acc_test=84.86, acc_test_nodropout=89.46, test time=15.39s
iter=200, acc_train=91.00, loss_train=0.46, acc_test=89.69, acc_test_nodropout=92.17, test time=16.46s
iter=300, acc_train=91.00, loss_train=0.31, acc_test=91.46, acc_test_nodropout=93.74, test time=16.49s
iter=400, acc_train=89.00, loss_train=0.59, acc_test=92.96, acc_test_nodropout=94.72, test time=16.36s
iter=500, acc_train=92.00, loss_train=0.23, acc_test=93.88, acc_test_nodropout=95.16, test time=16.58s
iter=600, acc_train=98.00, loss_train=0.15, acc_test=94.54, acc_test_nodropout=95.76, test time=16.46s
iter=700, acc_train=91.00, loss_train=0.33, acc_test=95.02, acc_test_nodropout=96.30, test time=16.61s
iter=800, acc_train=96.00, loss_train=0.20, acc_test=94.98, acc_test_nodropout=96.39, test time=16.39s
iter=900, acc_train=94.00, loss_train=0.24, acc_test=95.71, acc_test_nodropout=96.76, test time=16.08s
iter=1000, acc_train=92.00, loss_train=0.27, acc_test=95.75, acc_test_nodropout=96.87, test time=16.41s
iter=1100, acc_train=97.00, loss_train=0.14, acc_test=96.06, acc_test_nodropout=97.09, test time=16.63s
```



TensorBoard

Outline

- Data Invariance and Structure
- History of CNNs
- CNNs
- **Case Studies**
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- Practical Considerations
- Conclusion

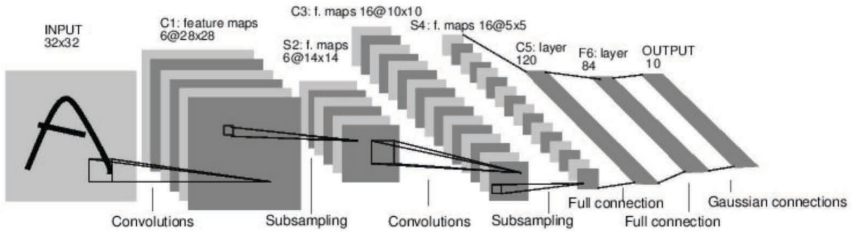
Case Studies

- **LeNet5 [LeCun-Bengio-et.al'98]:**

  Input is 32x32

  Architecture is CL-PL-CL-PL-FC-FC

  Accuracy on MNIST is 99.6%



- **AlexNet [Krizhevsky-et.al'12]:**

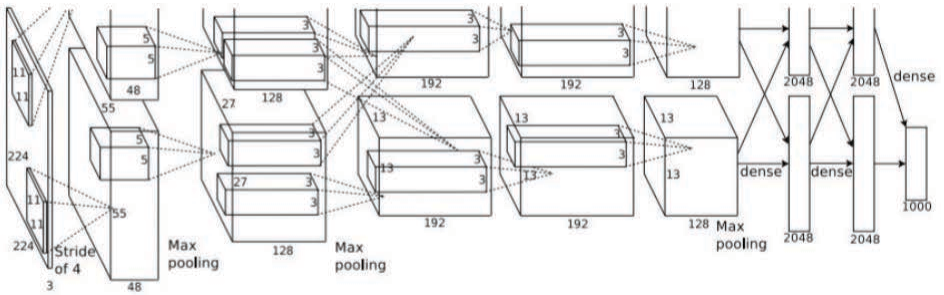
  Input is 227x227x3

  Architecture is 7CL-3PL-2FC

  Predication error on ImageNet is 15.4%

  Note: CL1 with 96 filters 11x11: 227x227x3 → 55x55x96 (stride=4),
  #parameters=(11x11x3)x96=35K

  PL1 2x2: 55x55x96 → 27x27x96,
  #parameters=0!



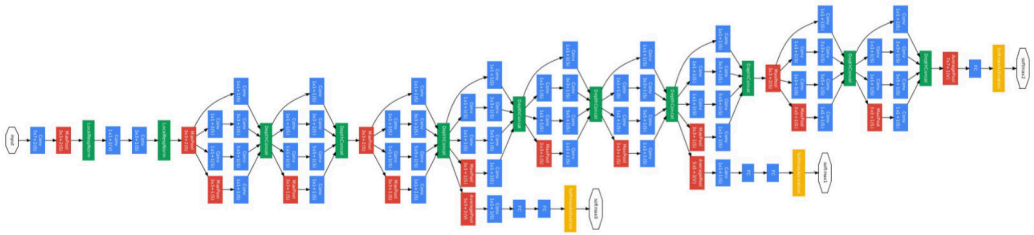
Case Studies

- **GoogleNet [Szegedy-et.al'14]:**

  Input is 227x227x3

  Architecture is 22 layers

  Predication error on ImageNet is 6.7%



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

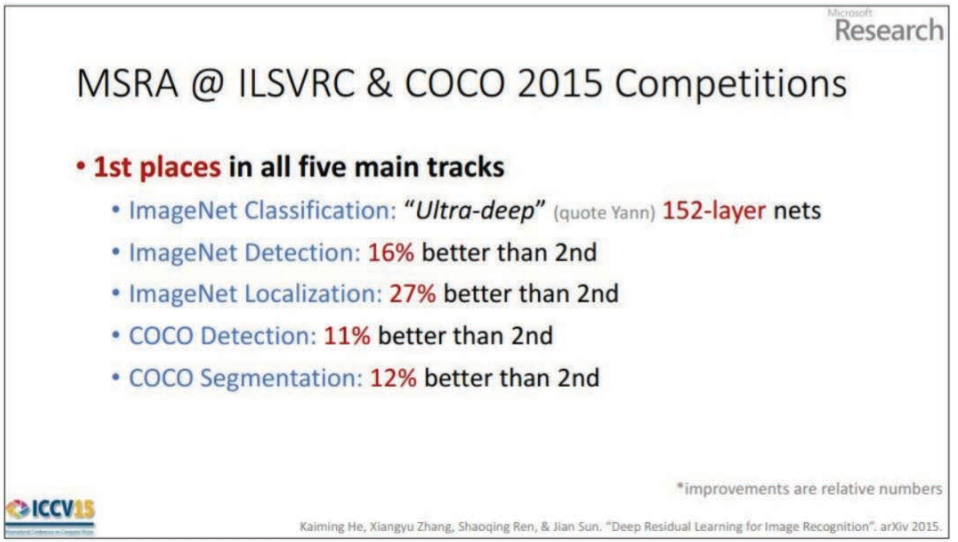
Architecture

- **ResNet [He-et.al'15]:** Microsoft Asia

  Input is 227x227x3

  Architecture is **152 layers!**

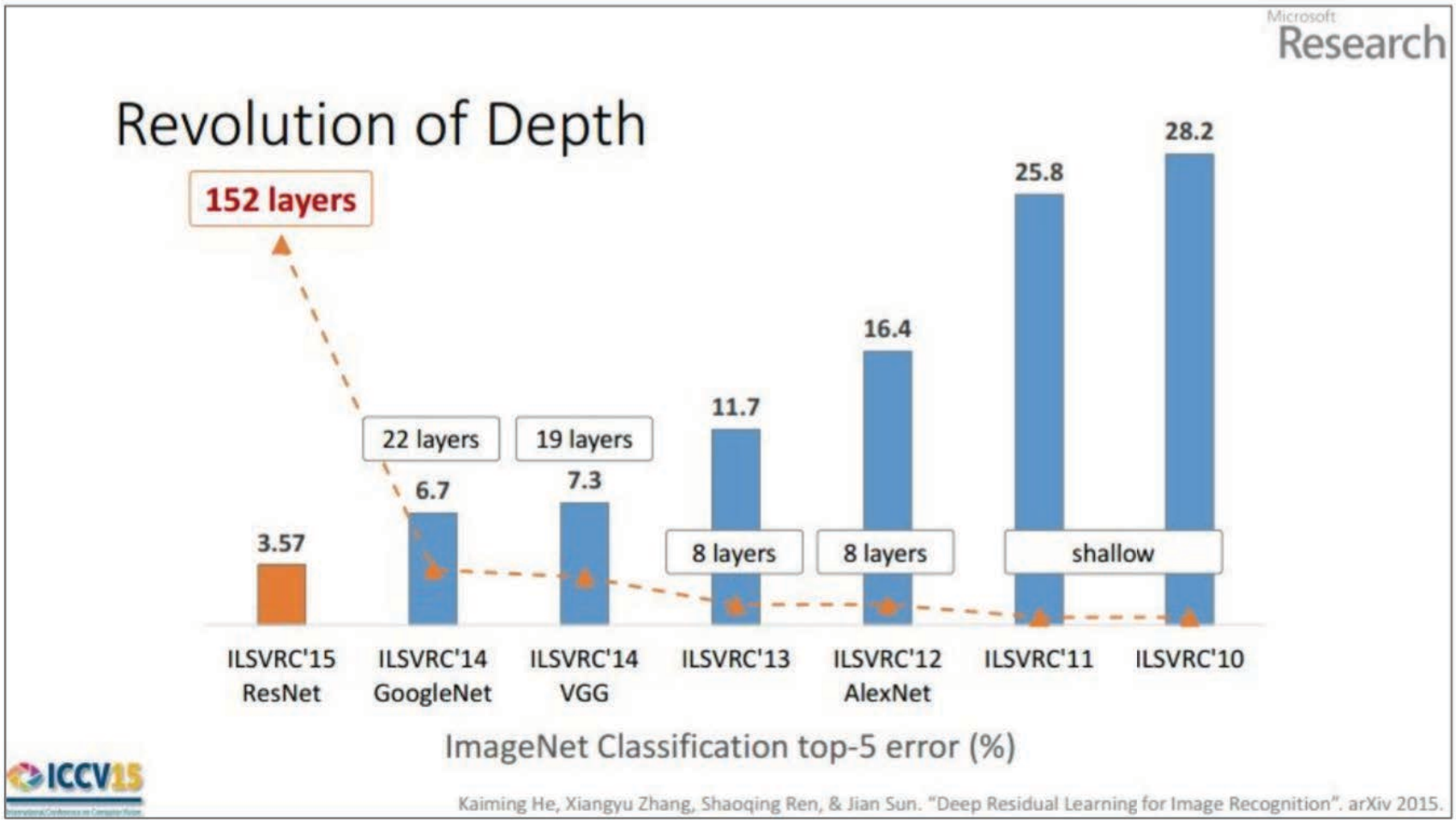
  Predication error on ImageNet is 3.6%



Outline

- Data Invariance and Structure
- History of CNNs
- CNNs
- Case Studies
- **Depth Revolution**
- Transfer Learning
- Applications of CNNs in Computer Vision
- Beyond Computer Vision
- Practical Considerations
- Conclusion

The Deeper The Better



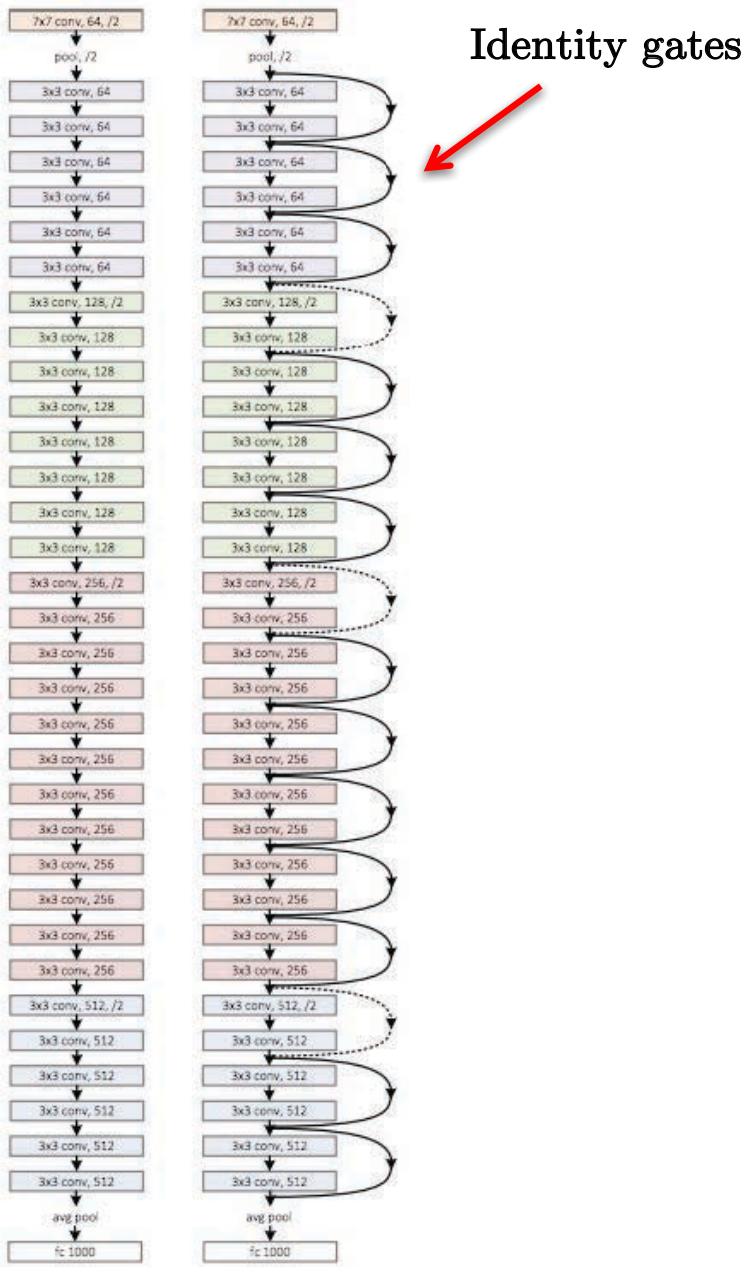
How to Make your NN deep?

- **ResNet [He-et.al'15]: 152 layers**

- **Issue:** Learning depends on **backpropagation**, which is limited by **vanishing gradient issue** (amplitude of gradient decreases at each gate) ⇒ It is hard to learn more than 10 layers.

$$\frac{\partial W}{\partial t} = -\tau \nabla L(W)$$

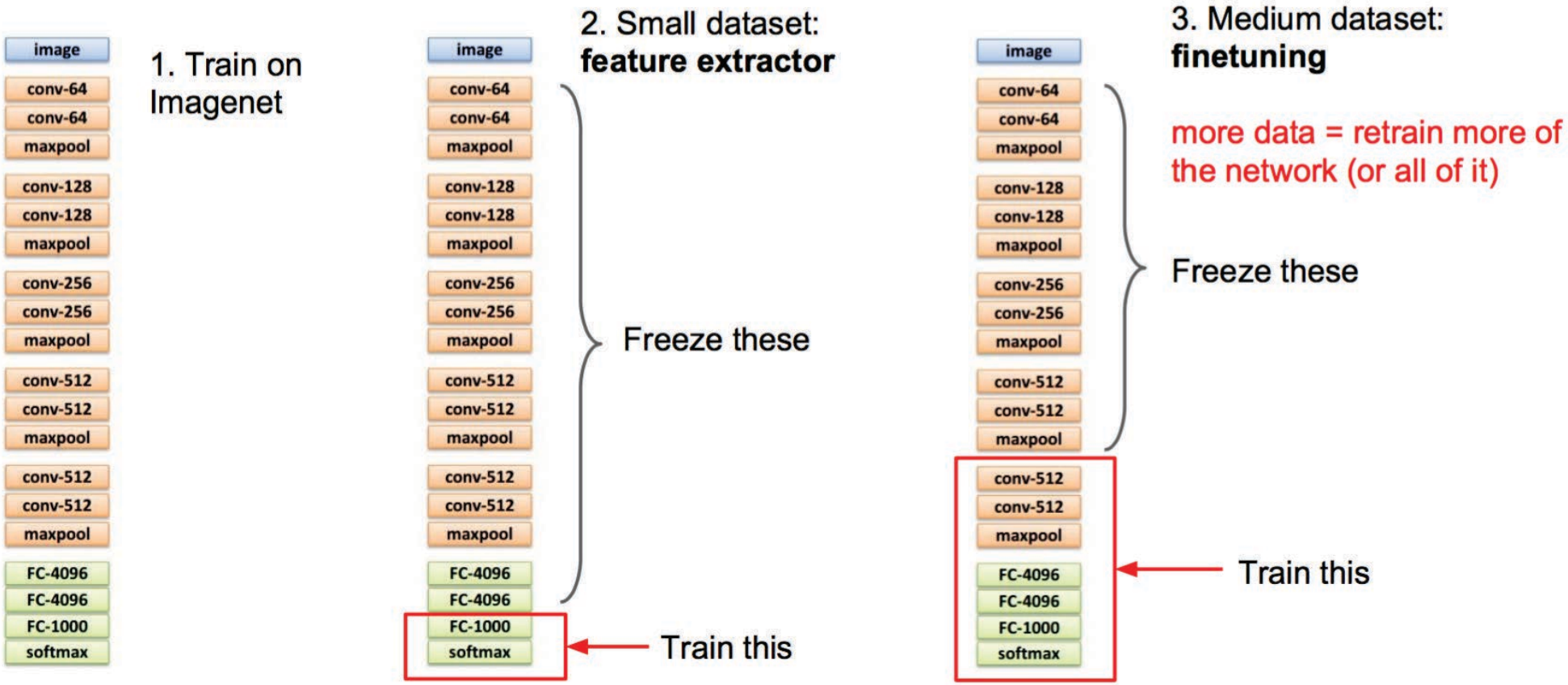
- **Key idea: Identity gates** do not suffer from vanishing gradient issue ⇒ Add them to the architecture to keep the gradient alive.



Outline

- Data Invariance and Structure
- History of CNNs
- CNNs
- Case Studies
- Depth Revolution
- **Transfer Learning**
- Applications of CNNs in Computer Vision
- Practical Considerations
- Conclusion

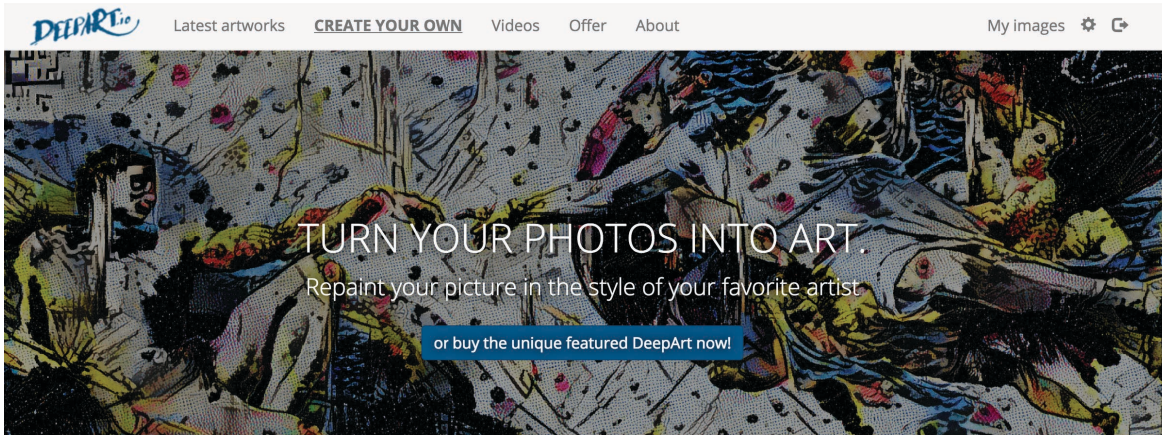
Transfer Learning

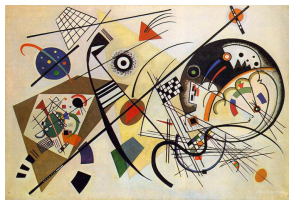


⇒ **Always use existing trained architectures for new applications** (features learned by the 1.5M images of ImageNet have a lot of contextual and sematic information)

DeepArts

https://deepart.io





Outline

- Data Invariance and Structure
- History of CNNs
- CNNs
- Case Studies
- Depth Revolution
- Transfer Learning
- **Applications of CNNs in Computer Vision**
- Practical Considerations
- Conclusion

CNN-based Computer Vision

- A general framework:

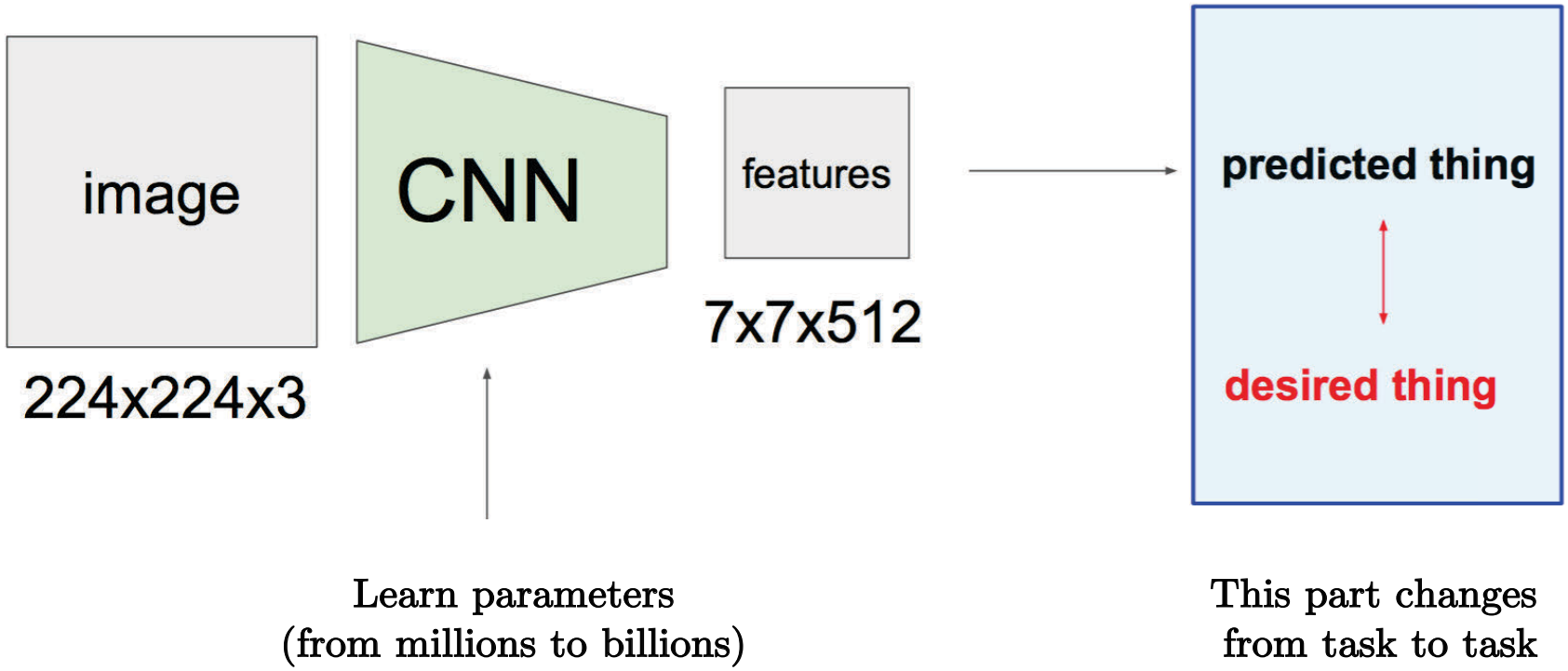
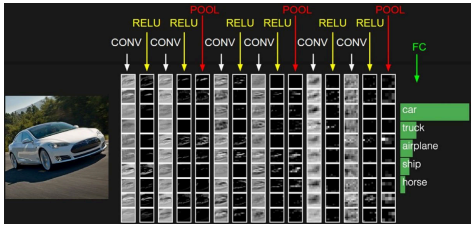
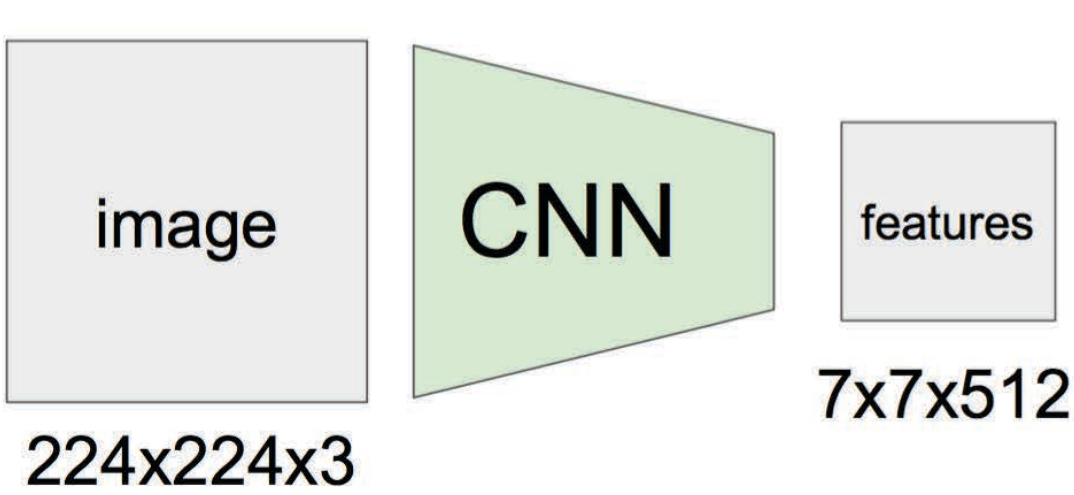


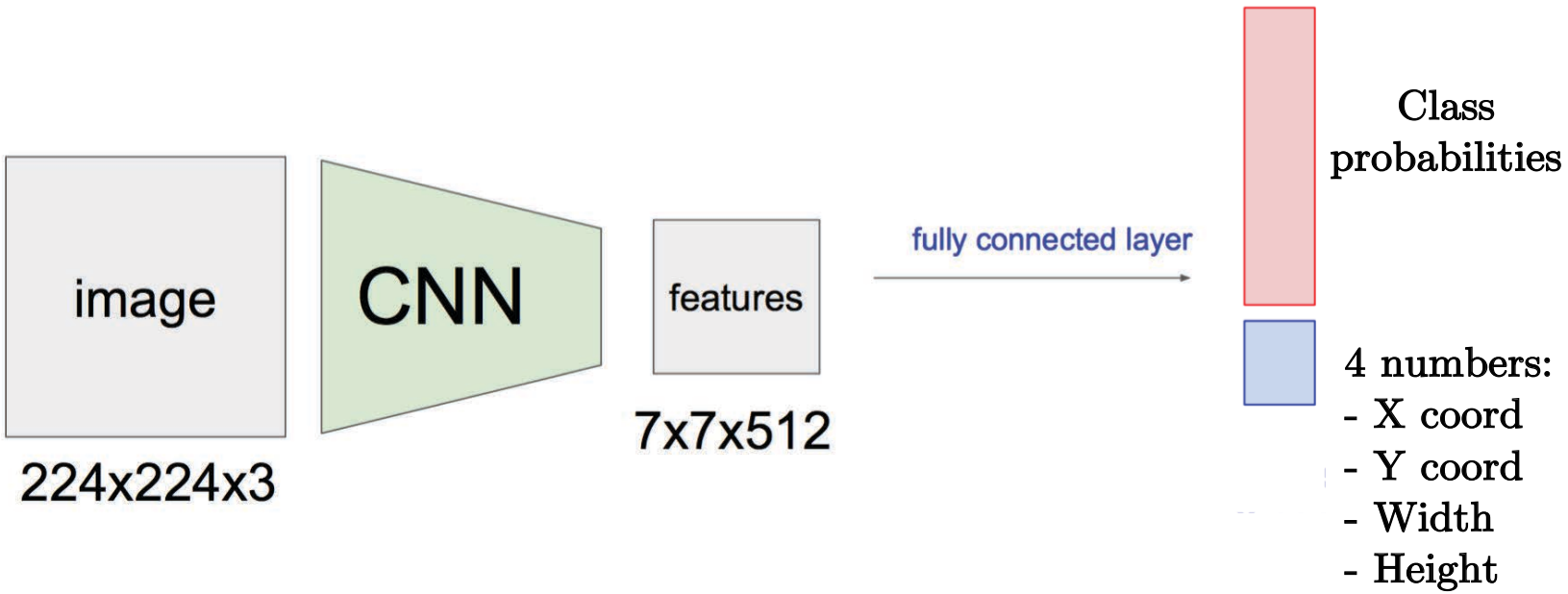
Image Classification



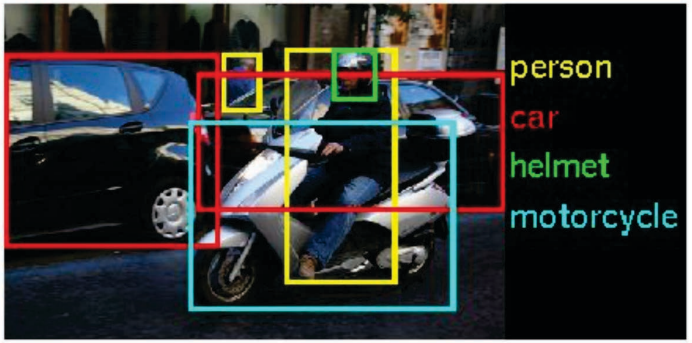


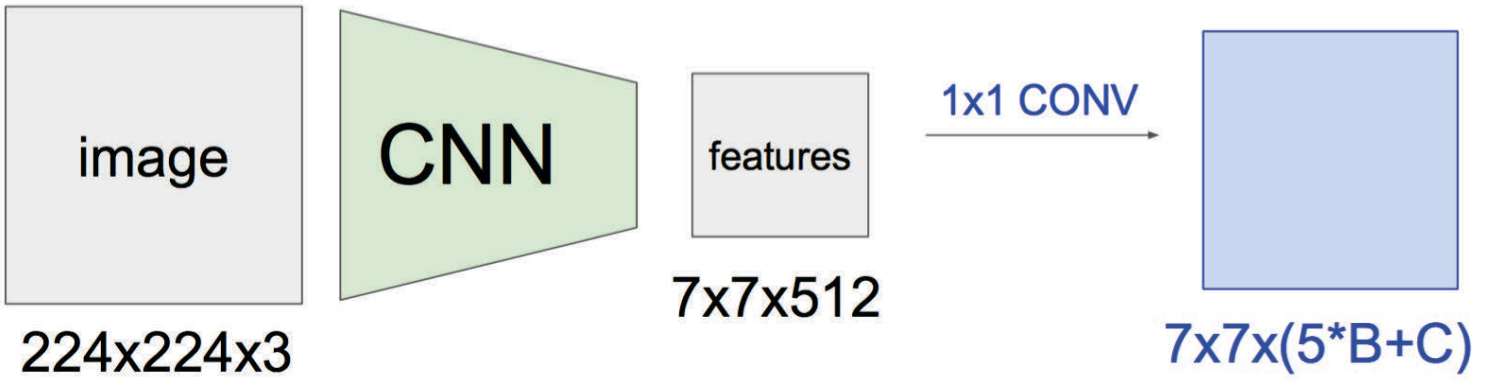
A vector of 1000 numbers [ImageNet] giving probabilities for different classes.

Object Localization



Object Detection





For each of 7x7 locations:
- [x,y,width,height, confidence]*B
- class

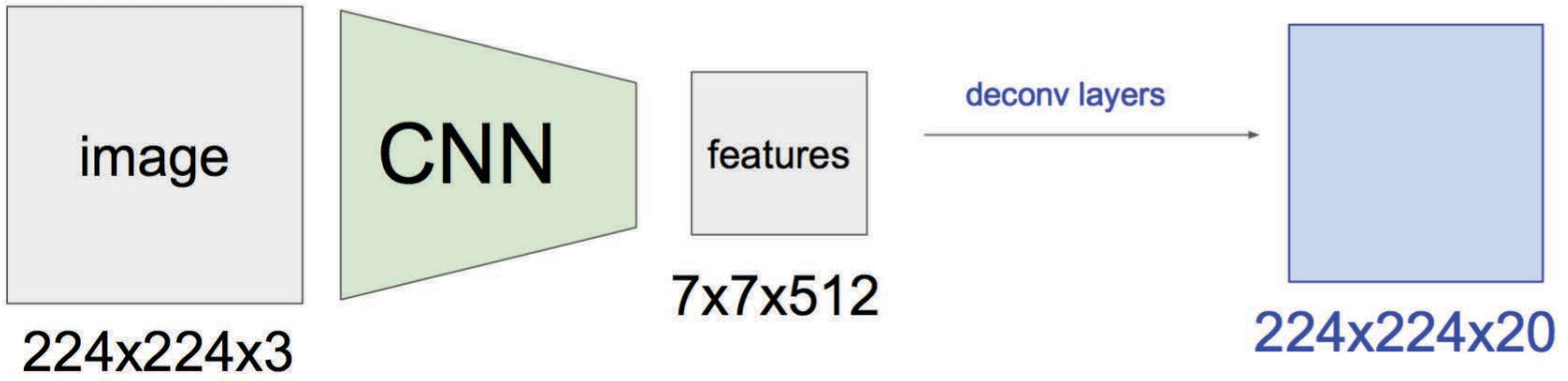
Object Detection

- YOLO (You Only Look Once): **Real-Time** Object Detection
  http://pjreddie.com/darknet/yolo

Image Segmentation

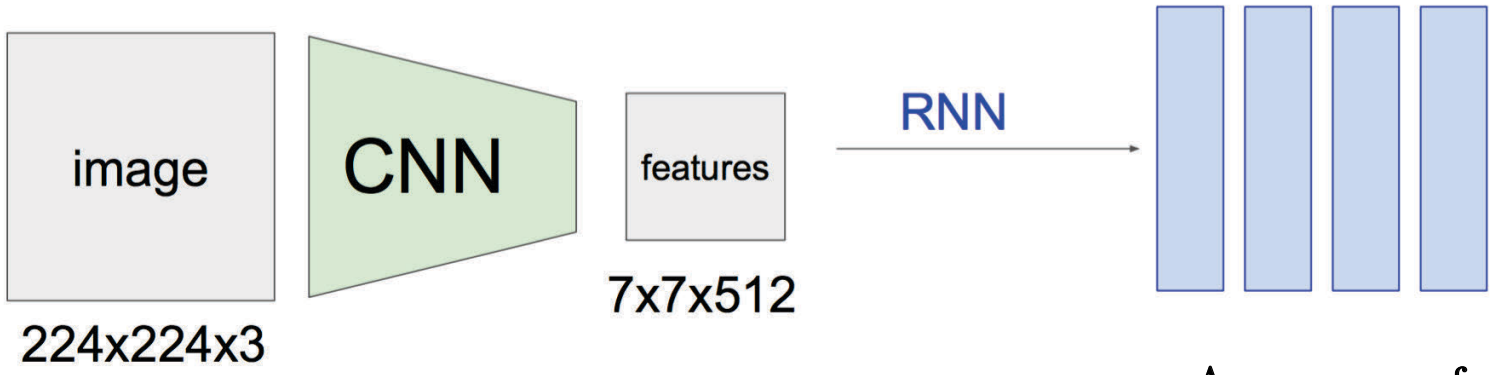
Image

Class image



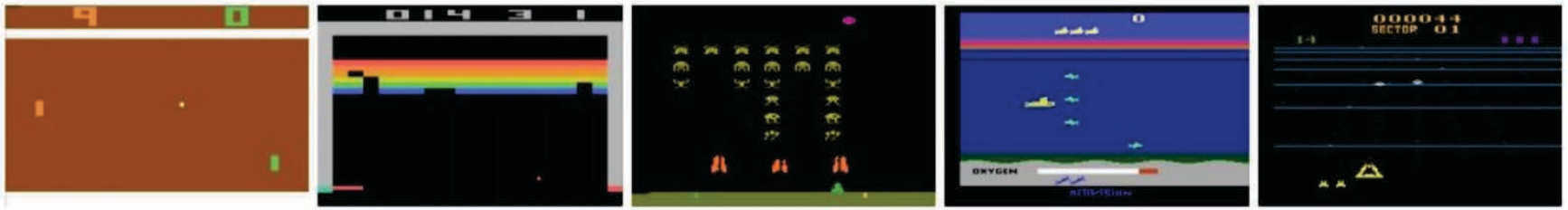
Class probabilities
at each pixel.

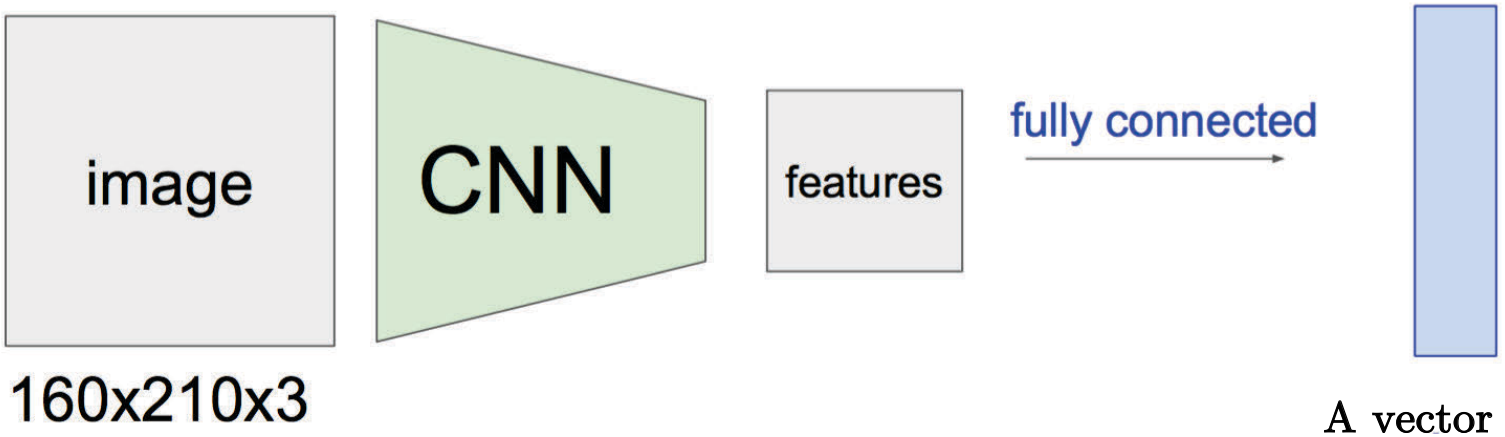
Image Captioning



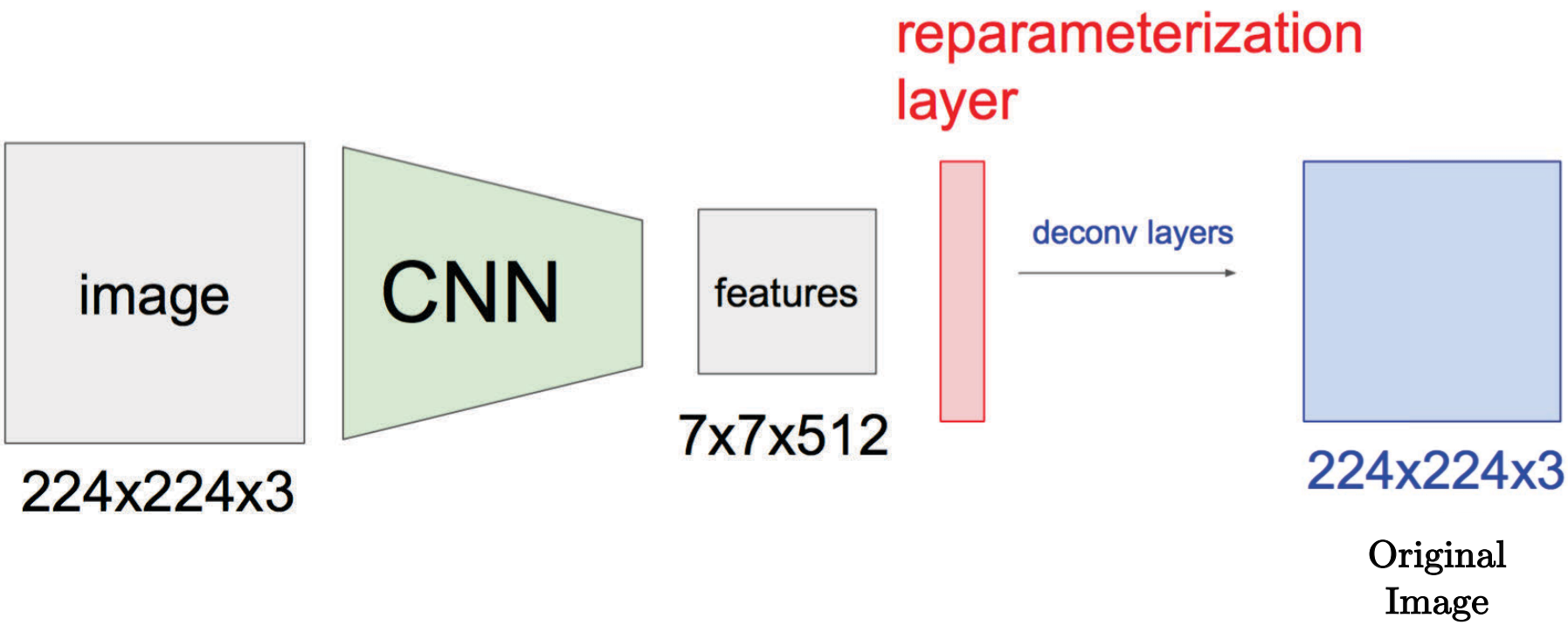
A sequence of 10,000-dimensional vectors giving probabilities of different words in the caption.

Reinforcement Learning

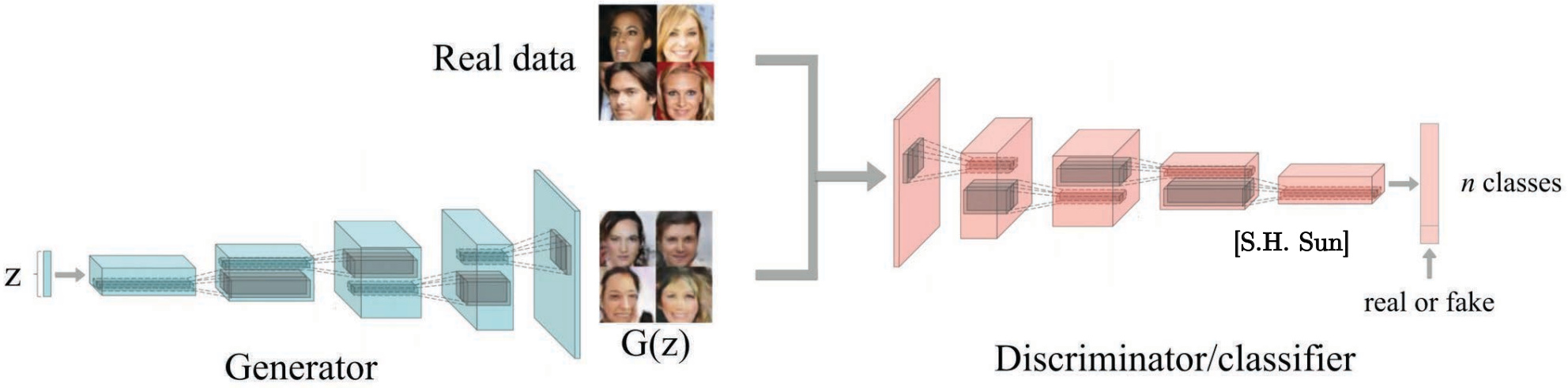




Variational Autoencoders



Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)

NVIDIA'17

Outline

- Data Invariance and Structure
- History of CNNs
- CNNs
- Case Studies
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- **Practical Considerations**
- Conclusion

Practical Considerations to apply CNNs

- **Which hardware?**
  1. *Buy a ready-to-go machine:*
     NVIDIA DevBox (4 TitanX GPUs)
  2. *Build your own machine:*
     https://graphific.github.io/posts/building-a-deep-learning-dream-machine
  3. *Machine in the cloud:*
     Amazon AWS, Microsoft Azure, Google Clould

- **Which framework?**
  1. *Google TensorFlow*
  2. *Facebook PyTorch*
  3. *Keras*
  4. *Theano* $\Rightarrow$ **All use Python.**
  5. *Caffee*
  6. *Lasagne*

Practical Considerations to apply CNNs

- **Which architecture?**

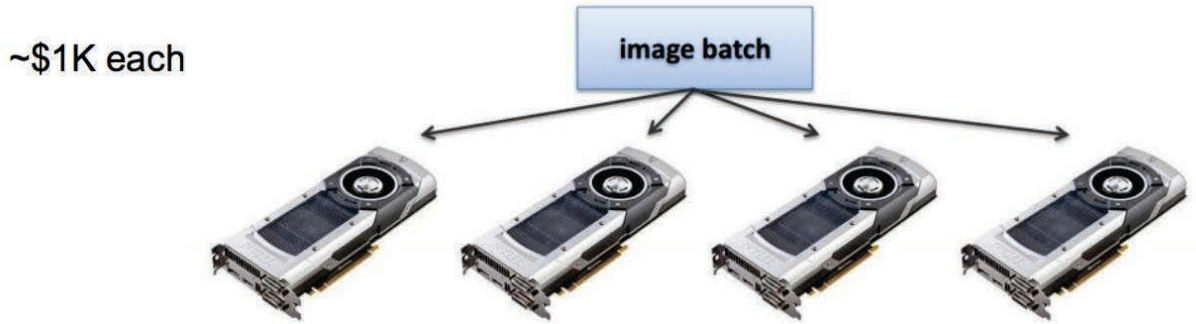
  1. Use existing architectures (like ResNet) with trained weights, and adapt to your problem (remove/add layers).

  2. Use hyperparameters of existing architectures.

- **Existing architectures:**

  1. VGG: 2-3 weeks training with 4 GPUs

  2. ResNet 101: 2-3 weeks with 4 GPUs



Outline

- Data Invariance and Structure
- History of CNNs
- CNNs
- Case Studies
- Depth Revolution
- Transfer Learning
- Applications of CNNs in Computer Vision
- Practical Considerations
- **Conclusion**

Conclusion

- **CNNs are a game changer:**
  1. *Breakthrough for all Computer Vision-related problems*
  2. *Revive dream of Artificial Intelligence*
  3. *Deep learning = Big Data + GPUs/Cloud + Neural Networks*
  4. *Big question why it works so well?*

- **CNNs for unstructured data:** Beyond Computer Vision
  1. *Generalization of CNNs to non-Euclidean domains/graph-structured data*
  2. *Same learning complexity as CNNs while being universal to graphs*

Questions?