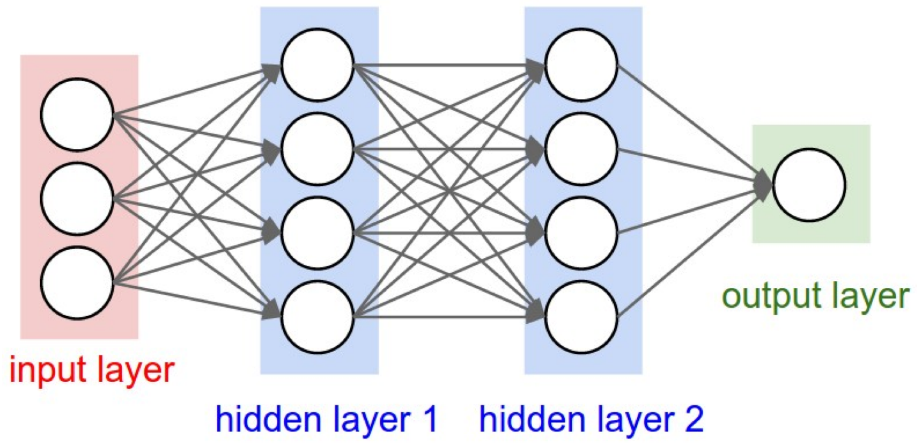
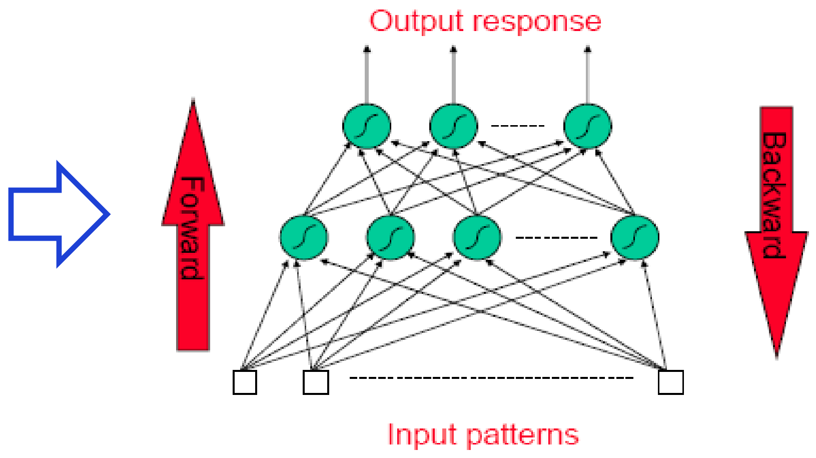
Rensselaer

Lecture 17: Introduction to Convolutional Neural Networks

Dr. Chengjiang Long
Computer Vision Researcher at Kitware Inc.
Adjunct Professor at RPI.
Email: **longc3@rpi.edu**

Recap Previous Lecture





Outline

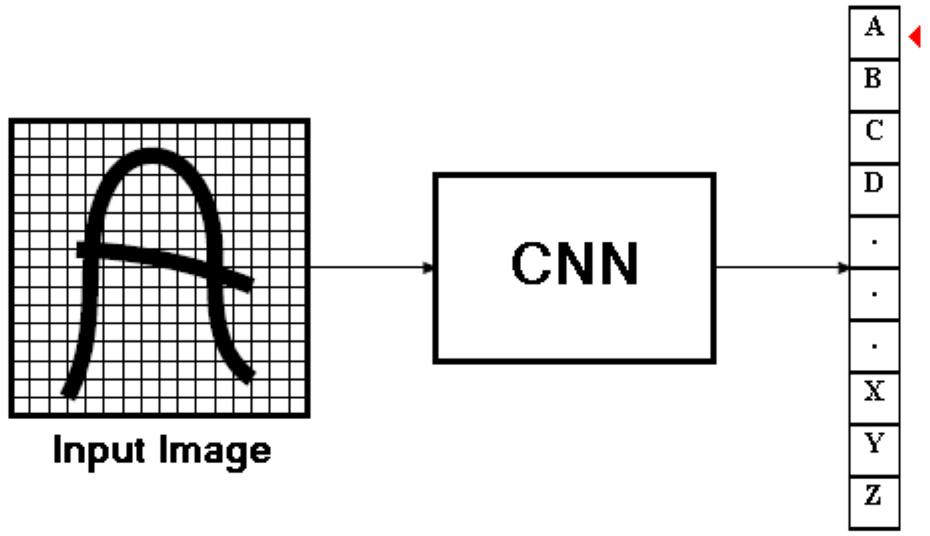
- Introduction to Covolutional Nueral Networks
- Common Layers Used in CNN
- Neural Network Training
- Applications

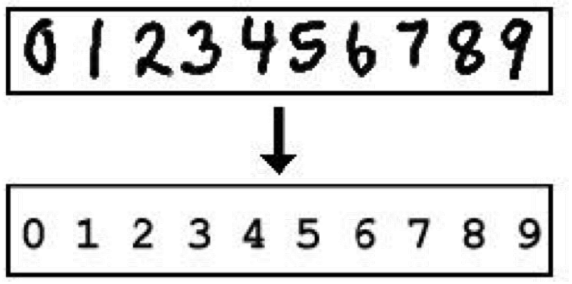
Outline

- **Introduction to Covolutional Nueral Networks**
- Common Layers Used in CNN
- Neural Network Training
- Applications

Introduction

- Convolutional neural networks
  -Signal processing, Image processing
- Improvement over the multilayer perceptron
  -performance, accuracy and some degree of invariance to distortions in the input images





History

- In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

Yann LeCun, Professor of Computer Science
The Courant Institute of Mathematical Sciences
New York University
Room 1220, 715 Broadway, New York, NY 10003, USA.
(212)998-3283 yann@cs.nyu.edu

About CNN’s

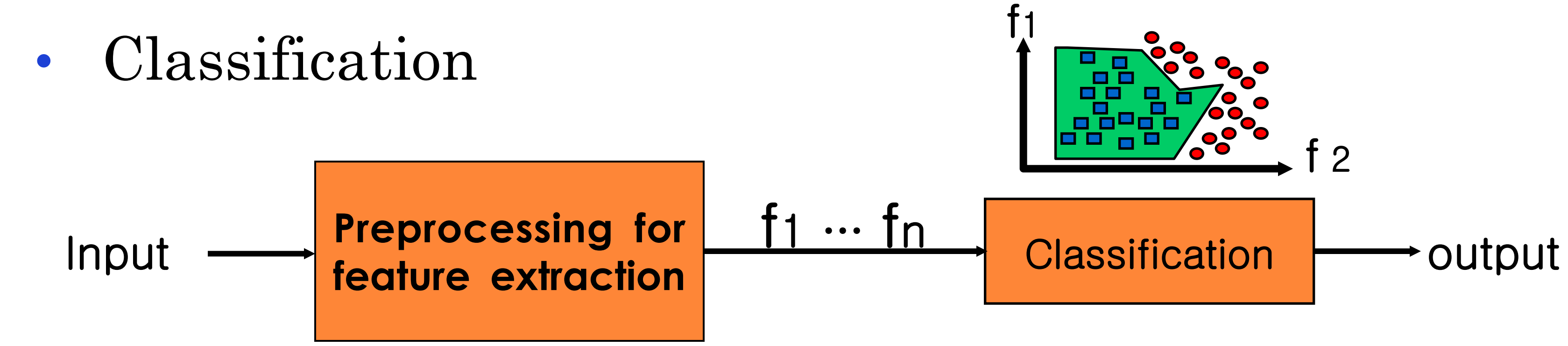
- CNN’s Were neurobiologically motivated by the findings of locally sensitive and orientation selective nerve cells in the visual cortex.
- They designed a network structure that implicitly extracts relevant features.
- Convolutional Neural Networks are a special kind of multi-layer neural networks.

About CNN’s

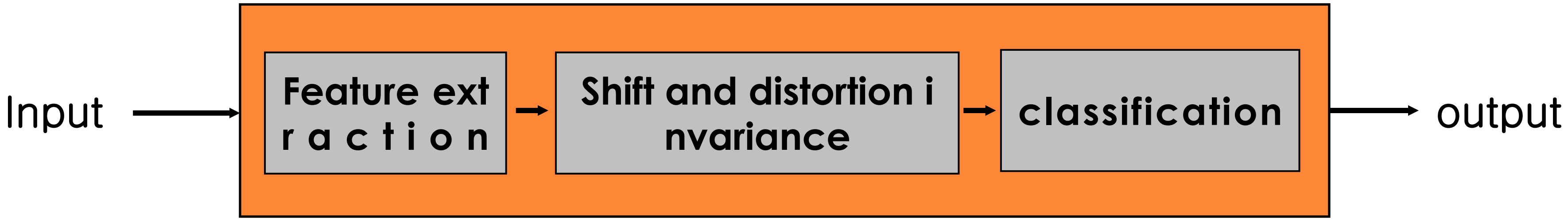
- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as handwritten characters).

Classification

- Classification



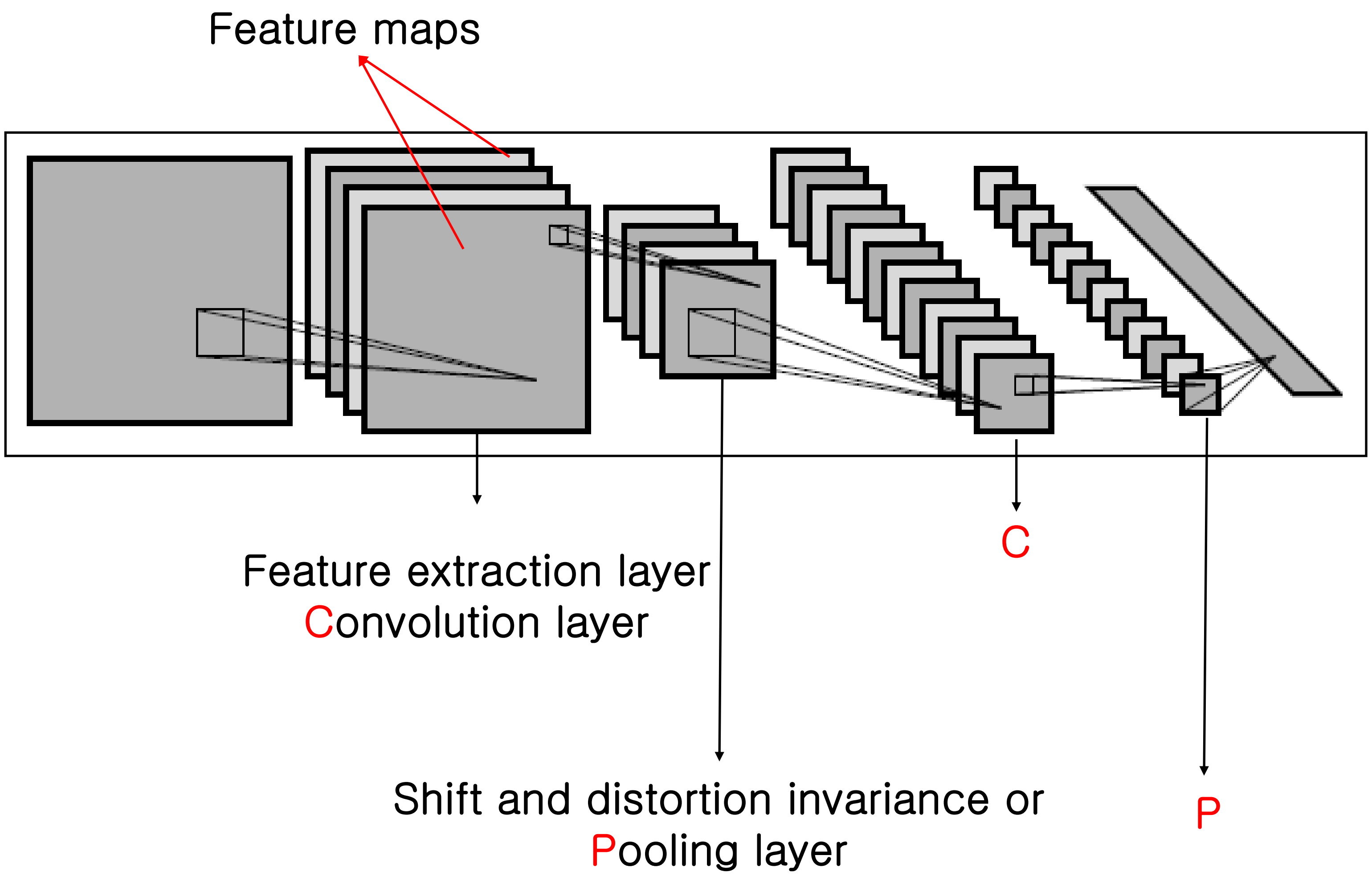
- Convolutional neural network



Outline

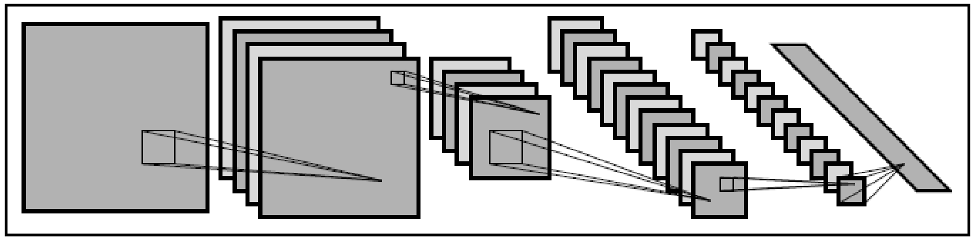
- Introduction to Covolutional Nueral Networks
- **Common Layers Used in CNN**
- Neural Network Training
- Applications

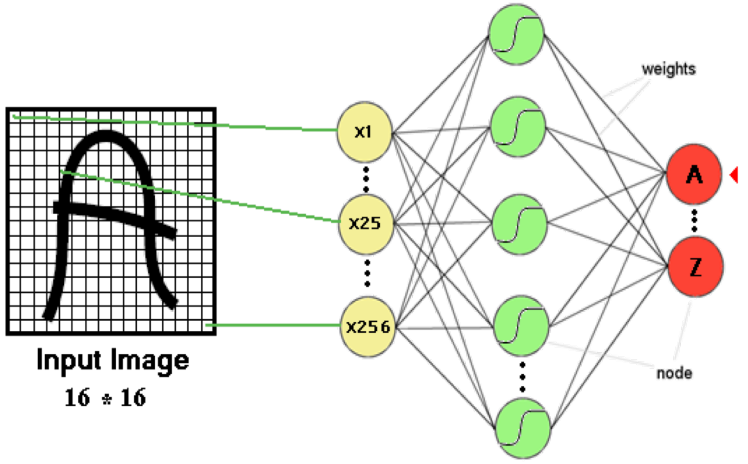
CNN's Topology

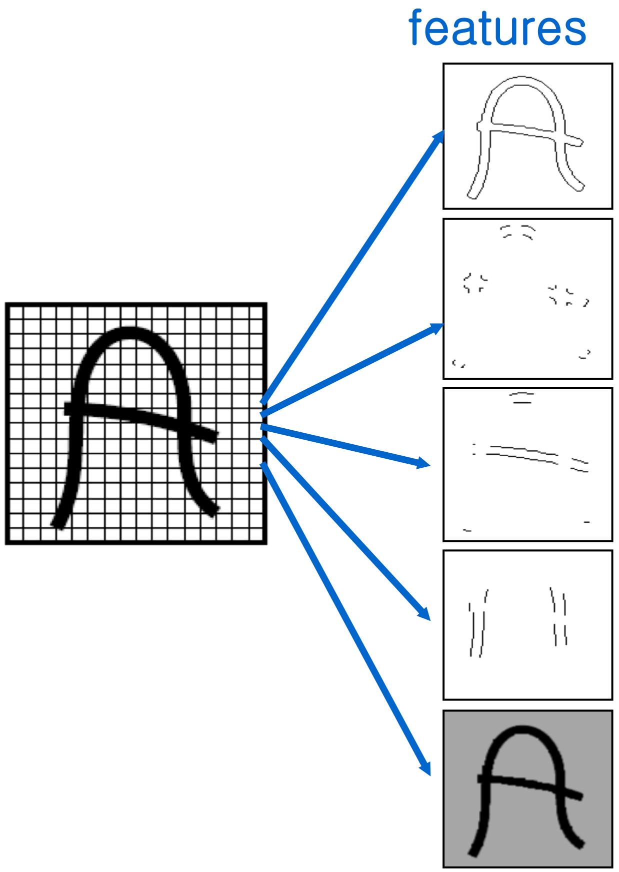


Feature extraction layer or Convolution layer

- detect the same feature at different positions in the input image.

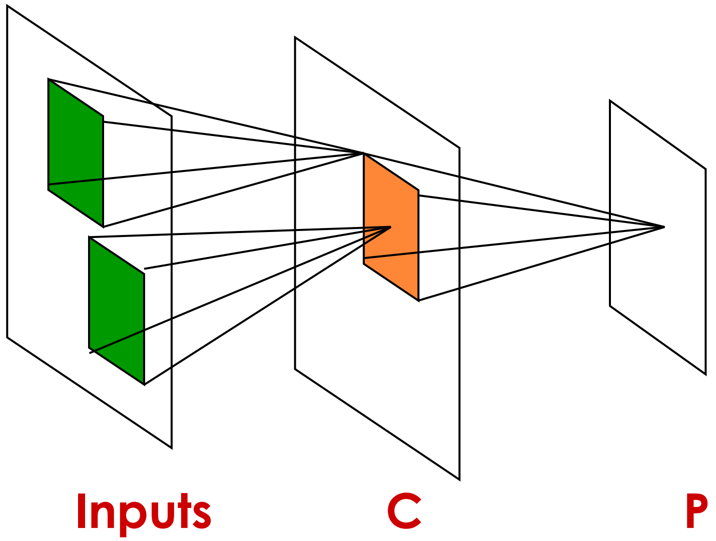






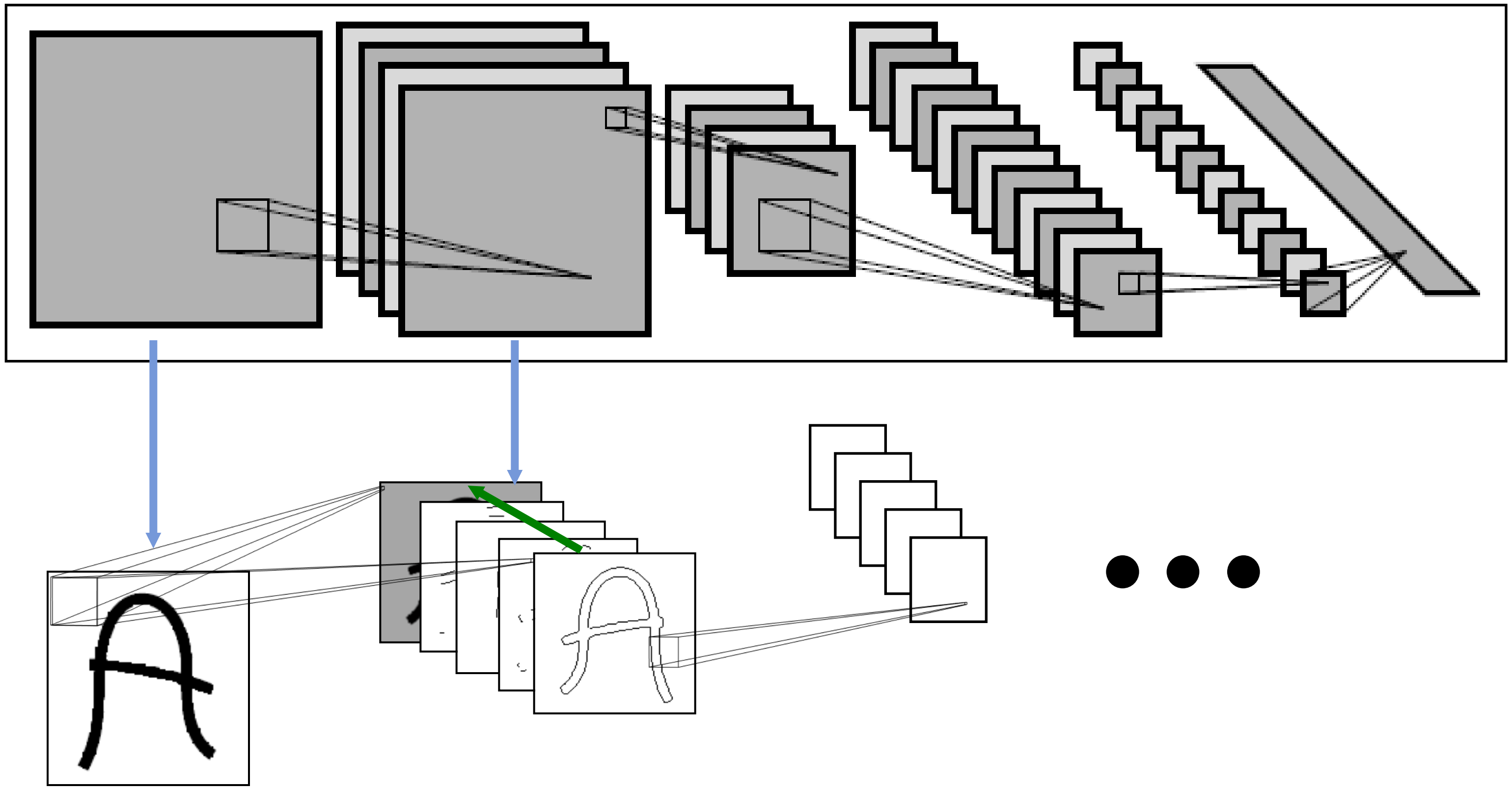
Feature extraction

- Shared weights: all neurons in a feature share the same weights (but not the biases).
- In this way all neurons detect the same feature at different positions in the input image.
- Reduce the number of free parameters.

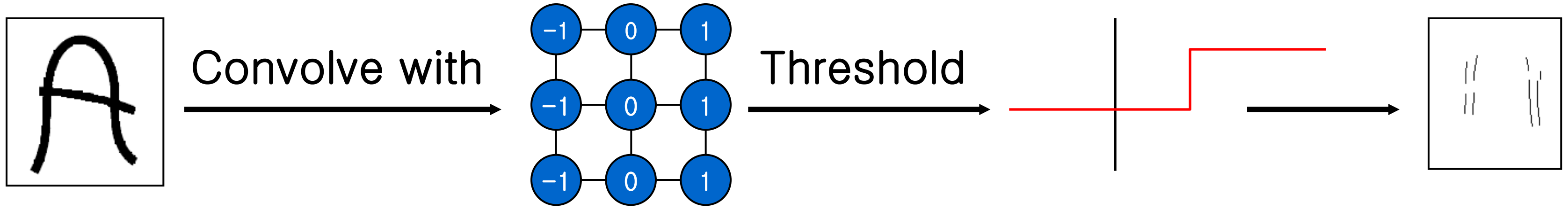


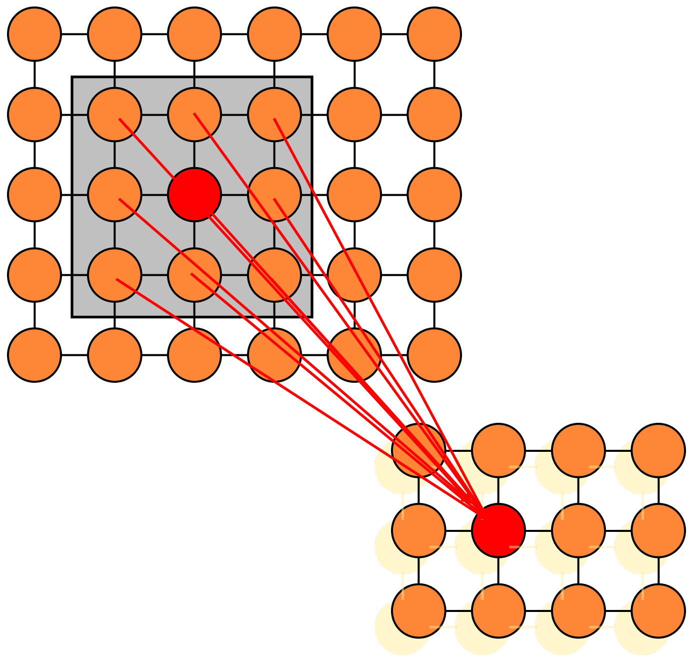
Feature extraction

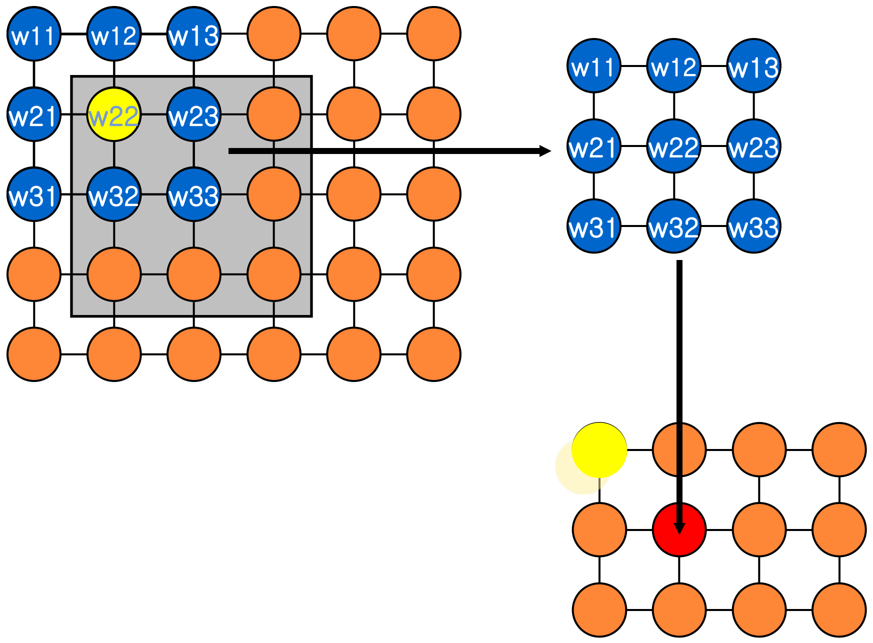
- If a neuron in the feature map fires, this corresponds to a match with the template.



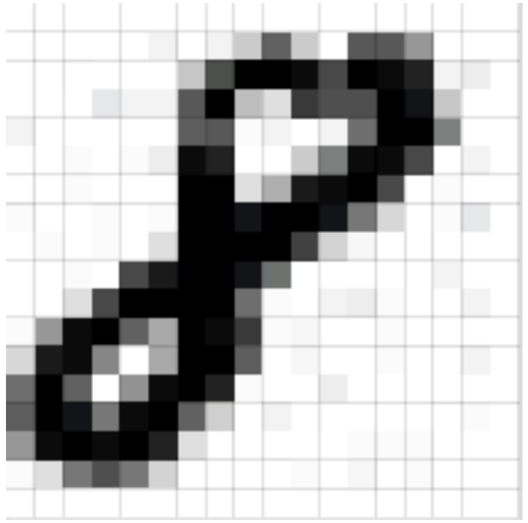
Feature extraction

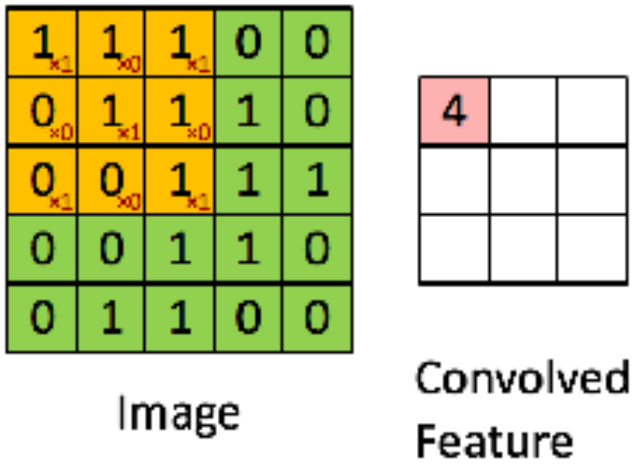






Convolutional layer





Convolutional layer

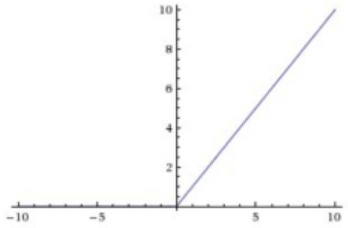
The size of the Feature Map (Convolved Feature) is controlled by three parameters: depth, stride, and zero-padding.

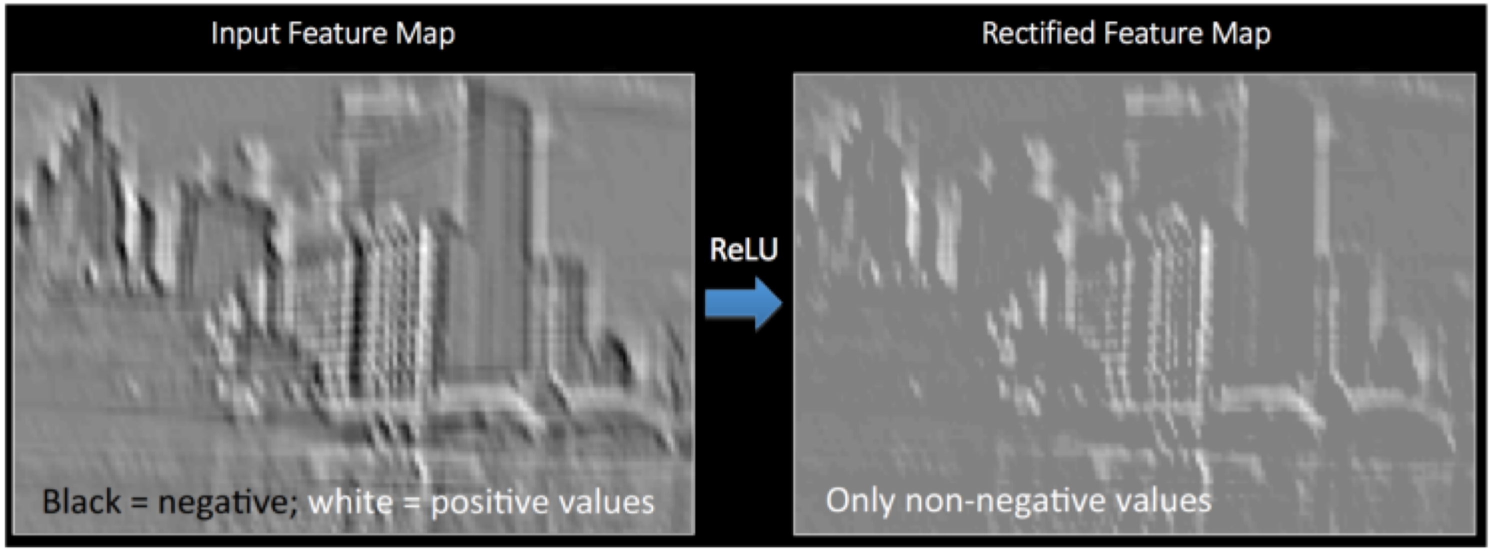
Note*: stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.*

Introducing Non Linearity (ReLU)

- ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by

Output = Max(zero, Input)

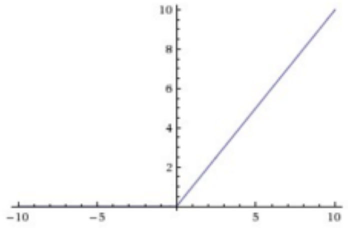


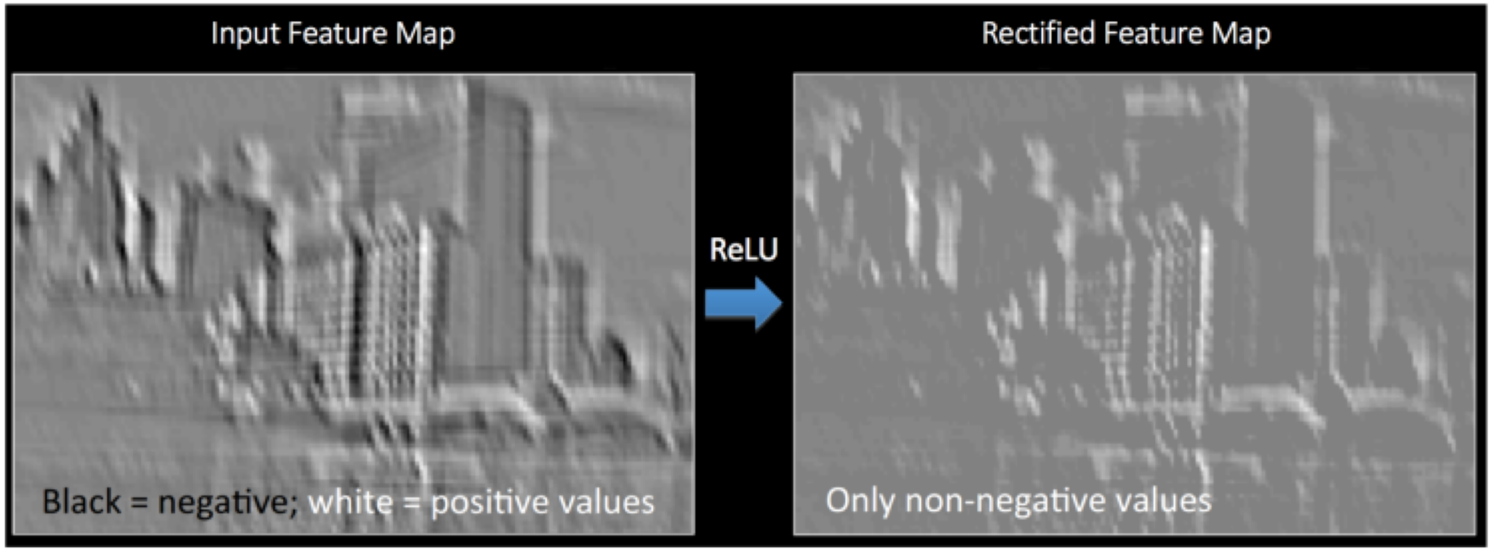


Introducing Non Linearity (ReLU)

- ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by

Output = Max(zero, Input)

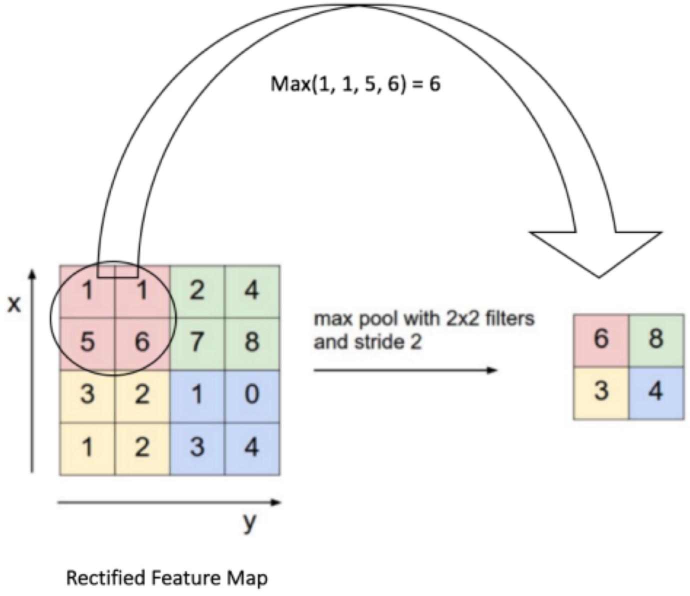




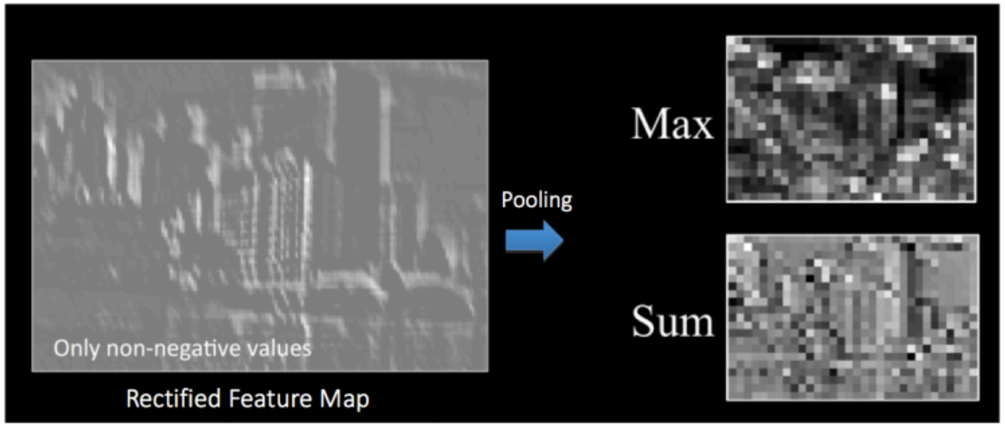
Introducing Non Linearity (ReLU)

- ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero.
- The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

Pooling layer

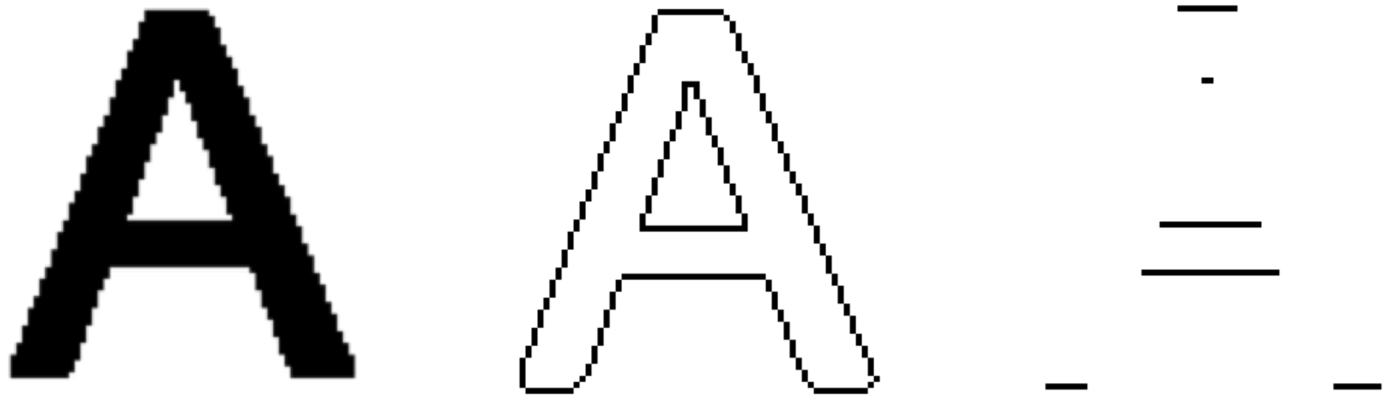


$$\begin{array}{|c|c|c|c|}\hline 1 & 1 & 2 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 3 & 2 & 1 & 0 \\ \hline 1 & 2 & 3 & 4 \\ \hline\end{array} \xrightarrow{\text{max pool with 2x2 filters and stride 2}} \begin{array}{|c|c|}\hline 6 & 8 \\ \hline 3 & 4 \\ \hline\end{array}$$



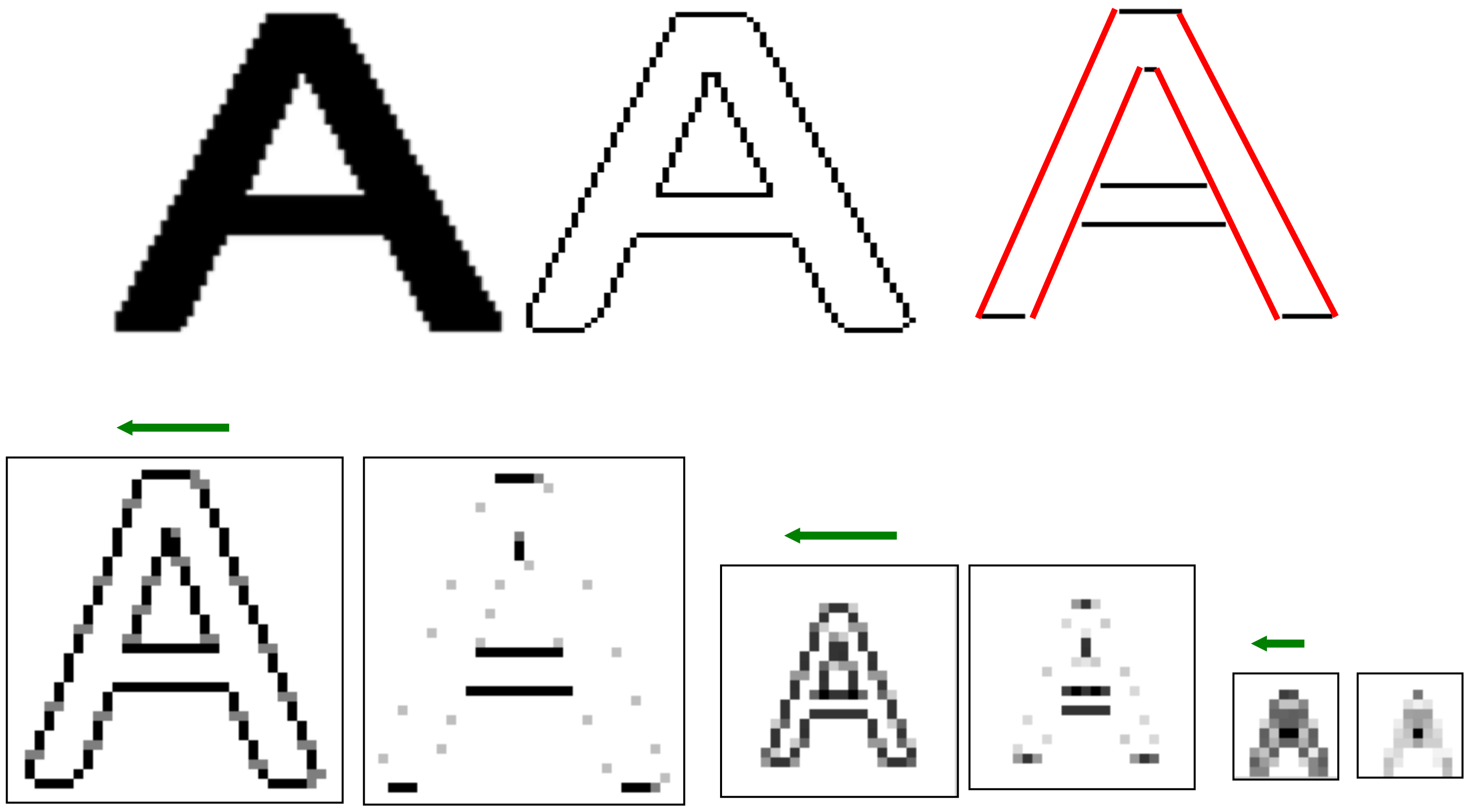
Pooling layer

- the pooling layers reduce the spatial resolution of each feature map
- By reducing the spatial resolution of the feature map, a certain degree of shift and distortion invariance is achieved.

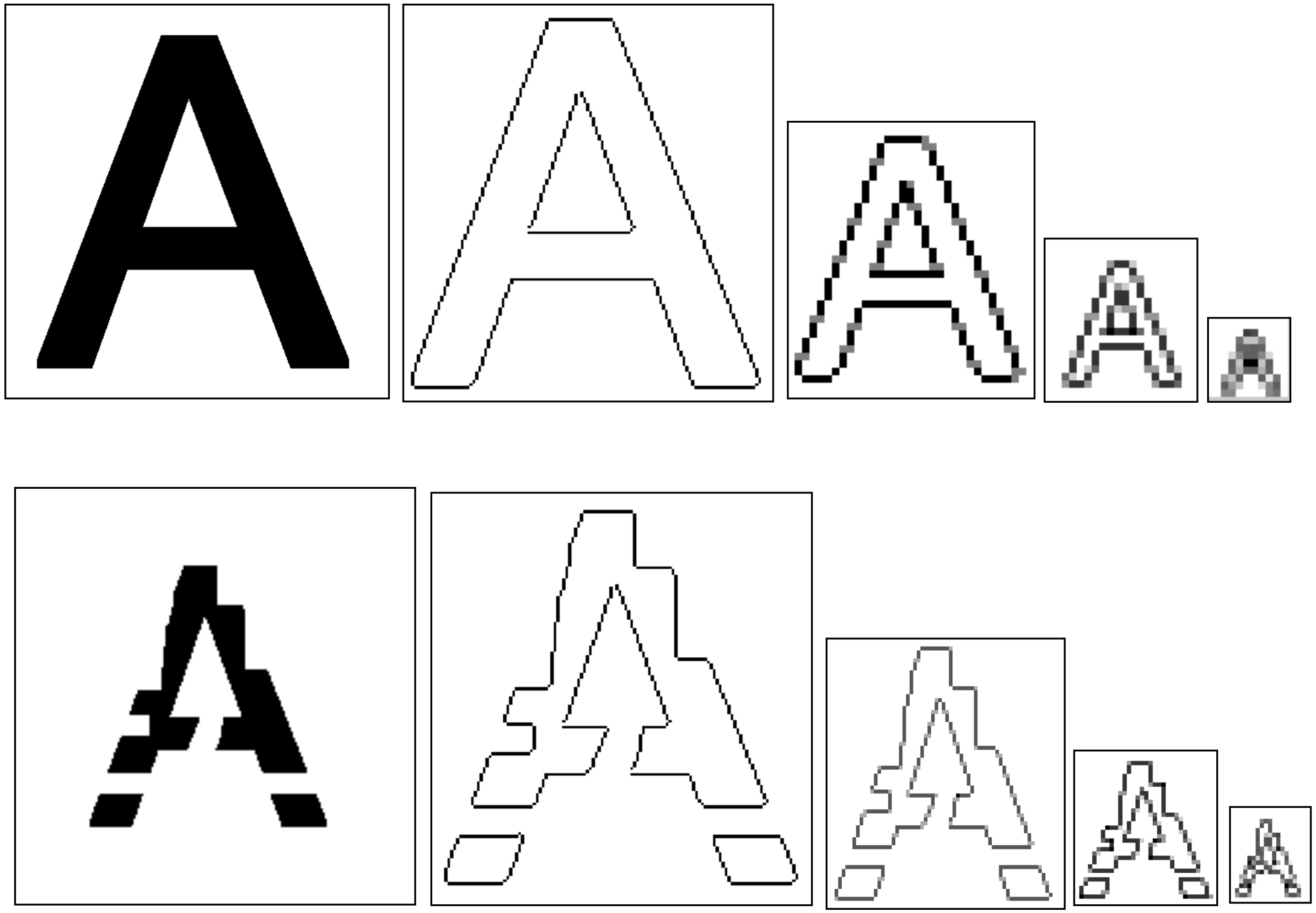


Pooling layer

- the pooling layers reduce the spatial resolution of each feature map

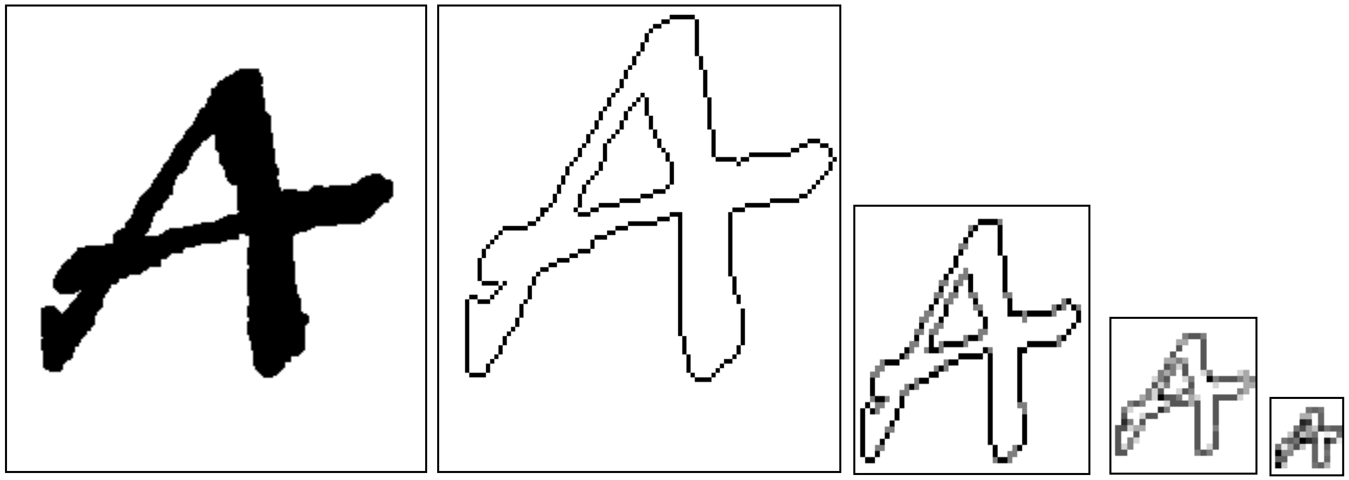


Pooling layer



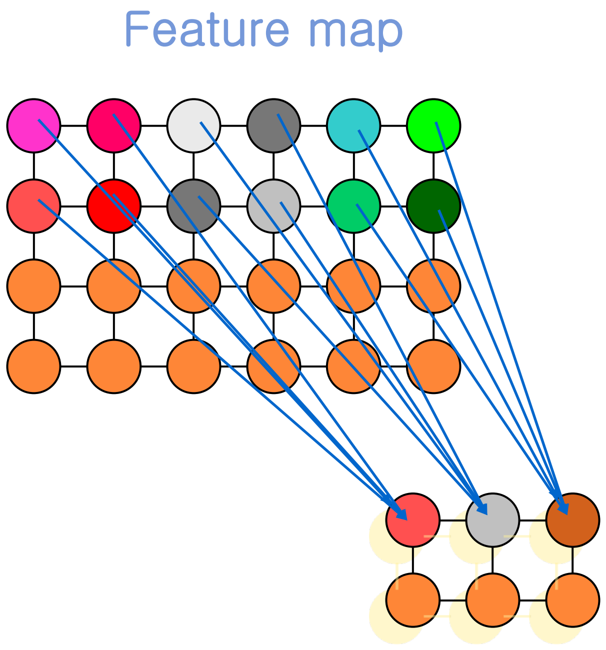
Pooling layer

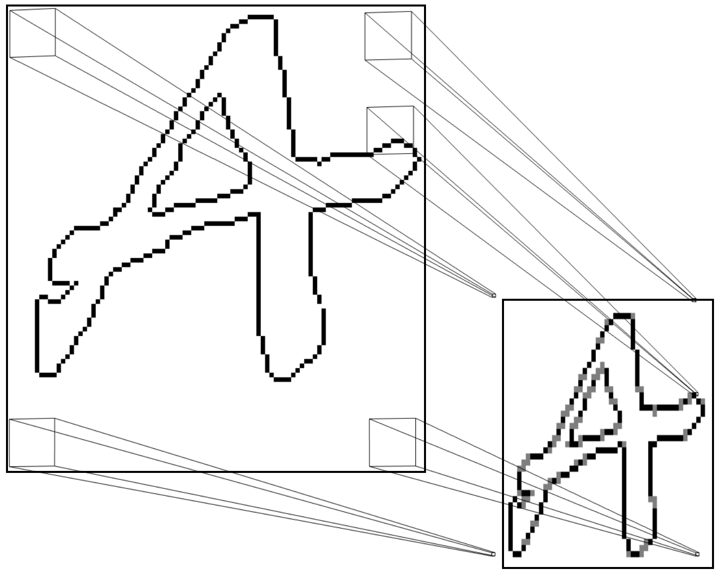
- The weight sharing is also applied in subsampling layers.



Pooling layer

- the weight sharing is also applied in pooling layers
- reduce the effect of noises and shift or distortion





Pooling Layer

- In particular, pooling

  - makes the input representations (feature dimension) smaller and more manageable

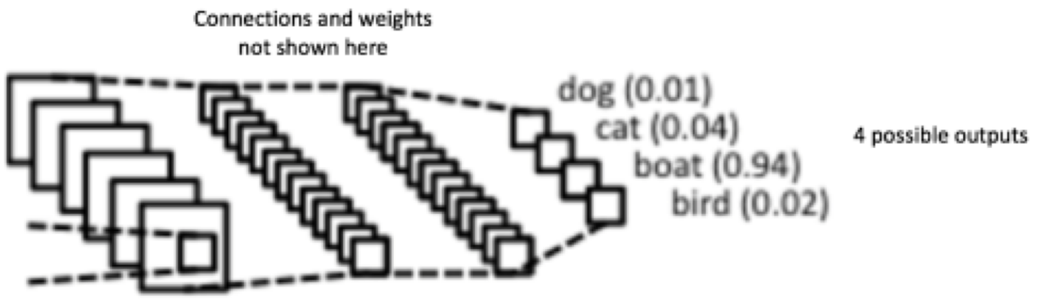
  - reduces the number of parameters and computations in the network, therefore, controlling overfitting [4]

  - makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).

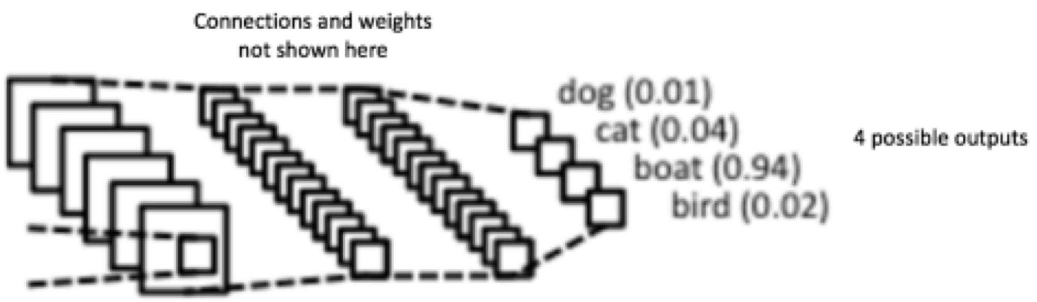
  - helps us arrive at an almost scale invariant representation of our image (the exact term is "equivariant"). This is very powerful since we can detect objects in an image no matter where they are located (read [18] and [19] for details).

Fully Connected Layer

- The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post).
- The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer.

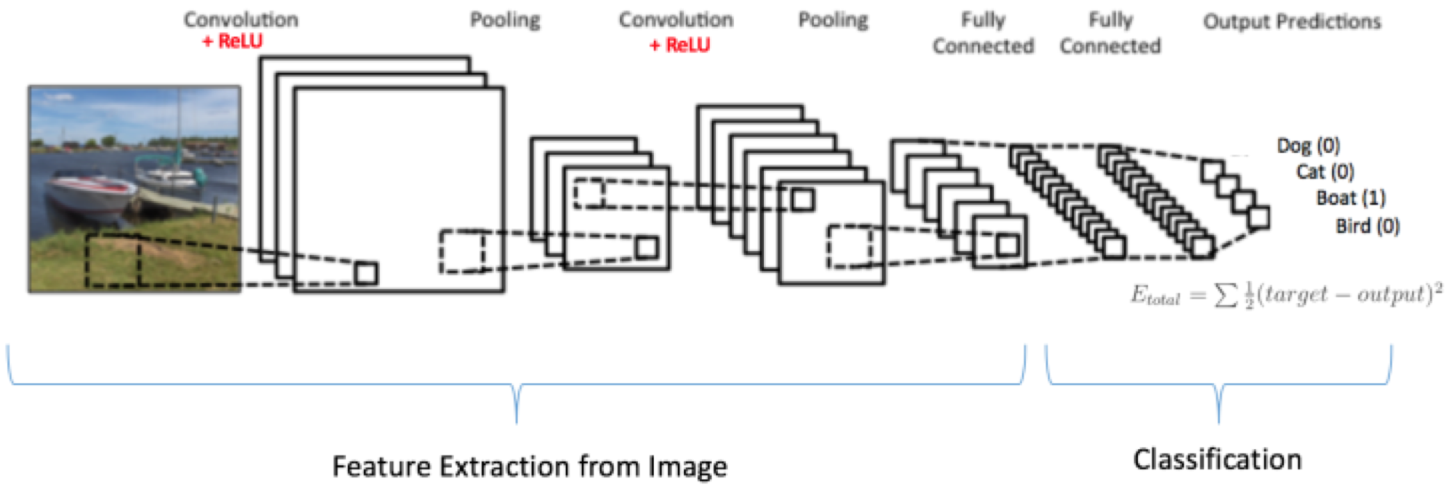


Fully Connected Layer

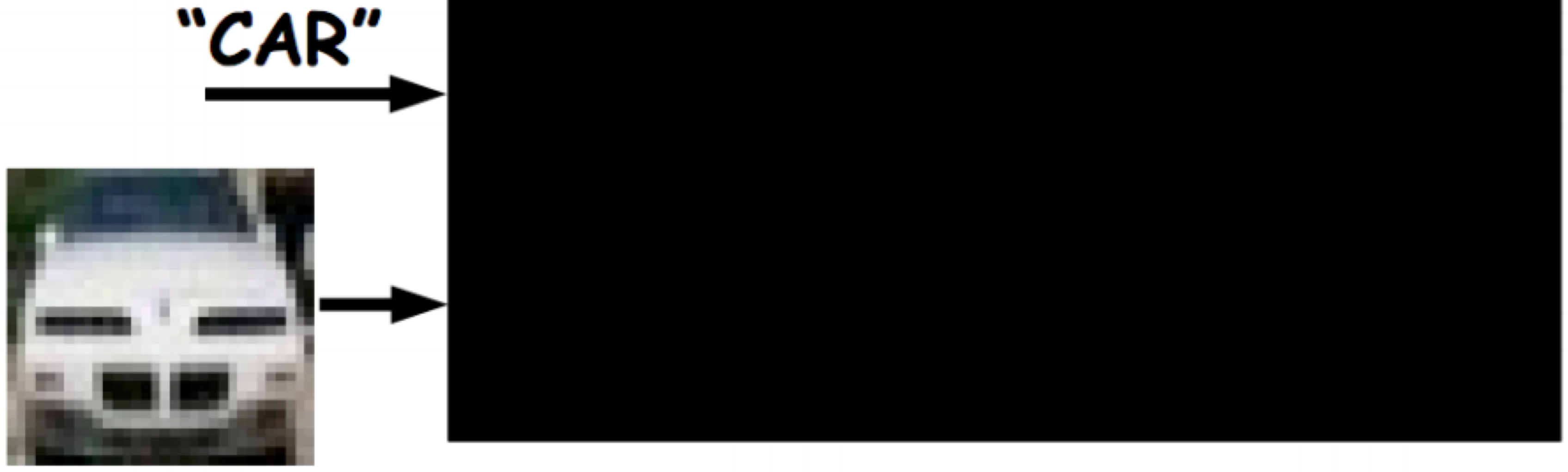


- This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

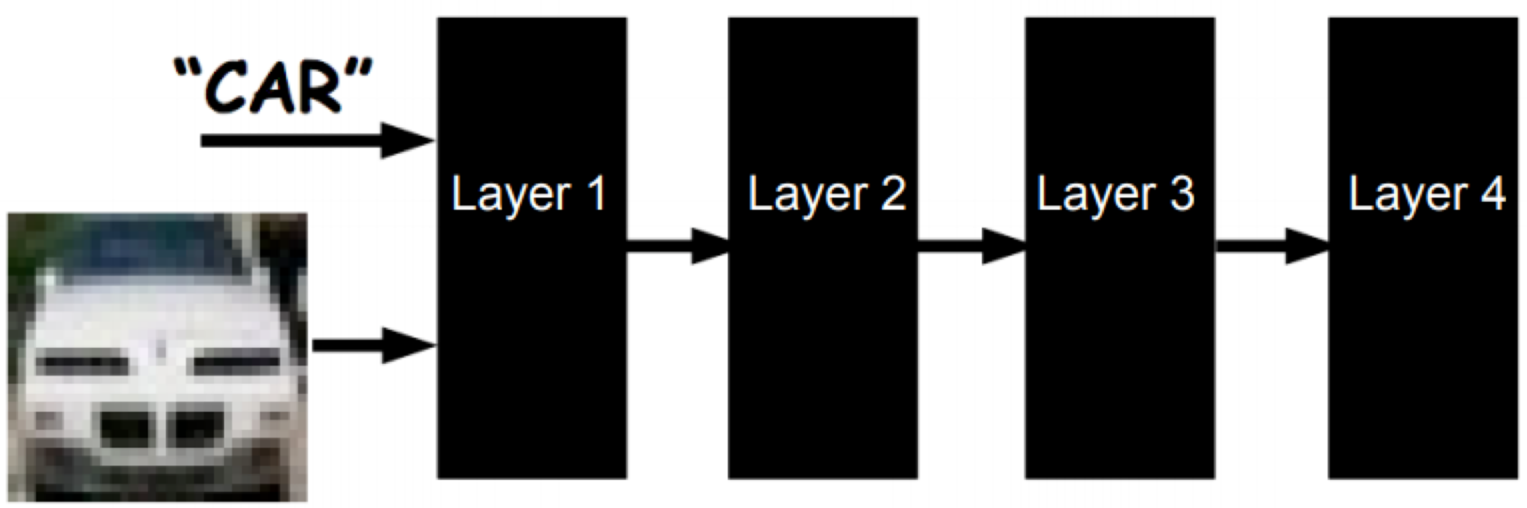
Putting it all together



Intuition behind Deep Neural Nets



Intuition behind Deep Neural Nets



NOTE: Each black box can have trainable parameters.
Their composition makes a highly non-linear system.

- The final layer outputs a probability distribution of categories.

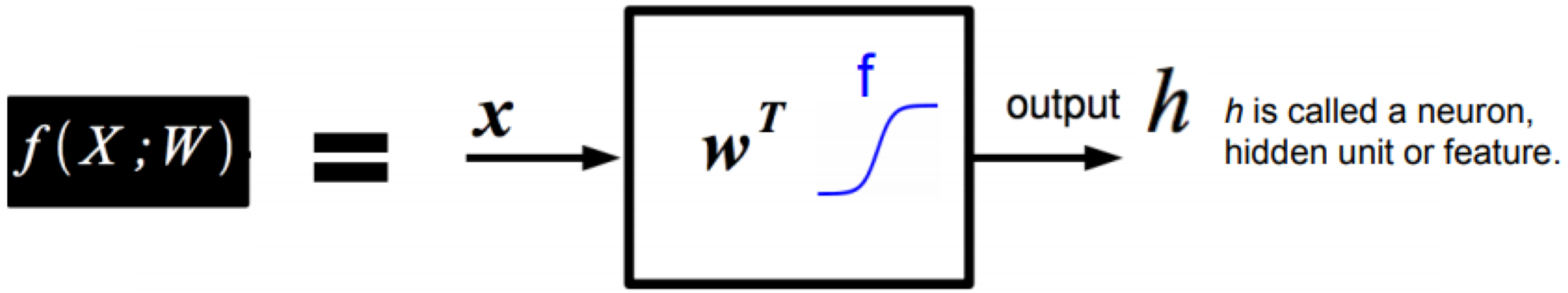
A simple single layer Neural Network

- Consists of a linear combination of input through a nonlinear function:

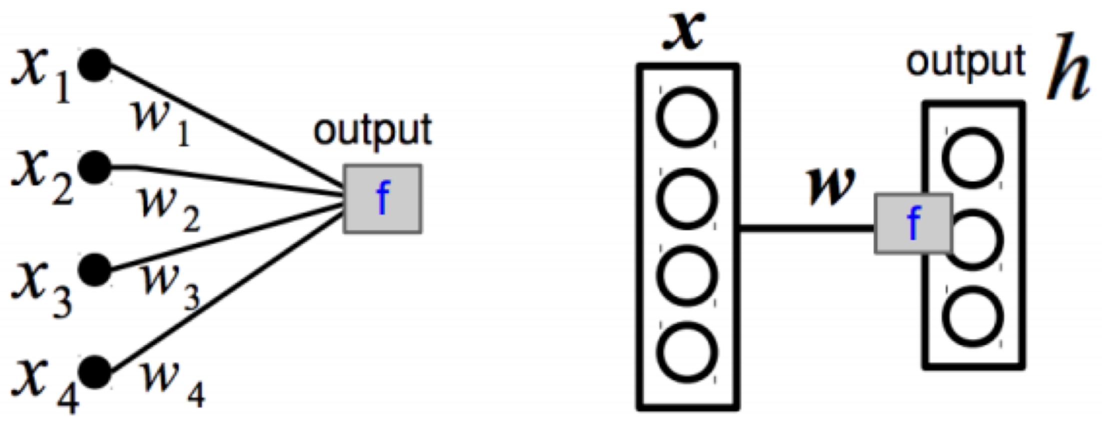
$$\begin{aligned} z &= Wx + b \\ a &= f(z) \end{aligned}$$

W is the weight parameter to be learned.
x is the output of the previous layer
f is a simple nonlinear function. Popular choice is max(x,0),
called ReLu (Rectified Linear Unit)

1 layer: Graphical Representation



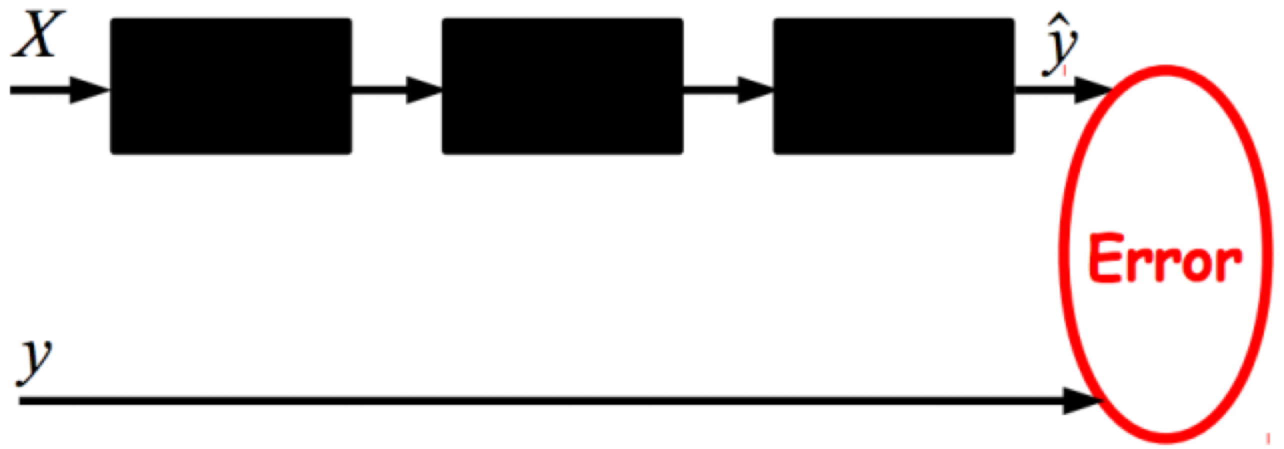
h is called a neuron, hidden unit or feature.



Outline

- Introduction to Covolutional Nueral Networks
- Common Layers Used in CNN
- **Neural Network Training**
- Applications

Joint training architecture overview

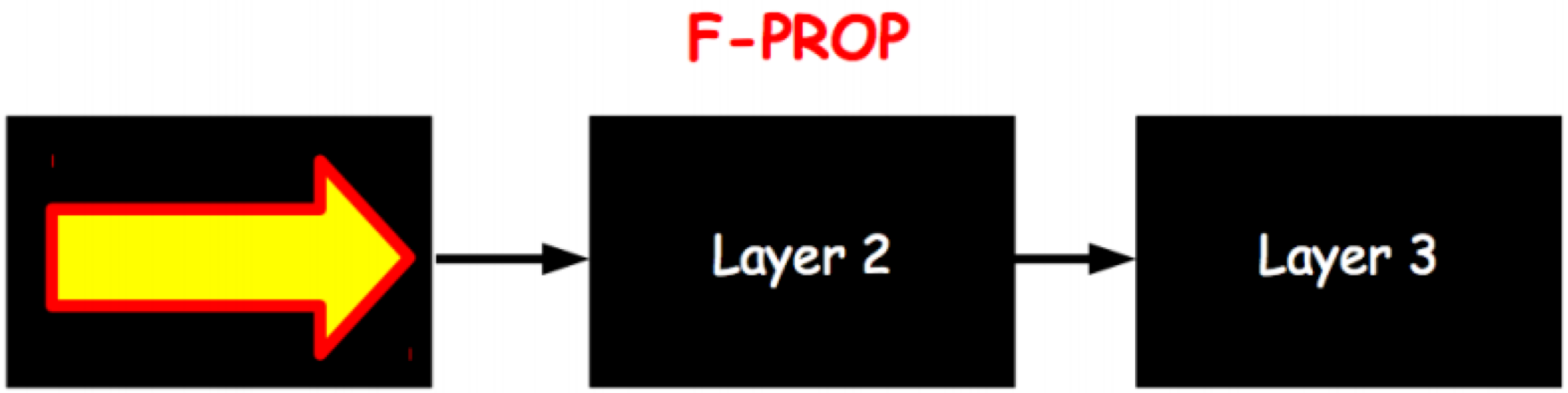


NOTE: **Multi-layer neural nets** with more than two layers are nowadays called **deep nets**!!

NOTE: User must specify number of layers, number of hidden units, type of layers and loss function.

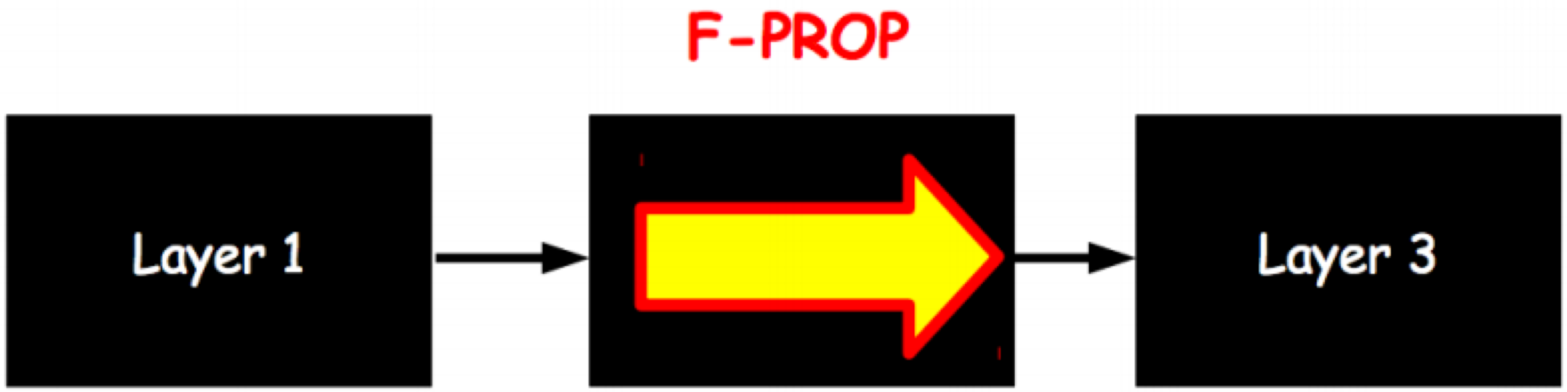
Neural Net Training

A) Compute loss on small mini-batch



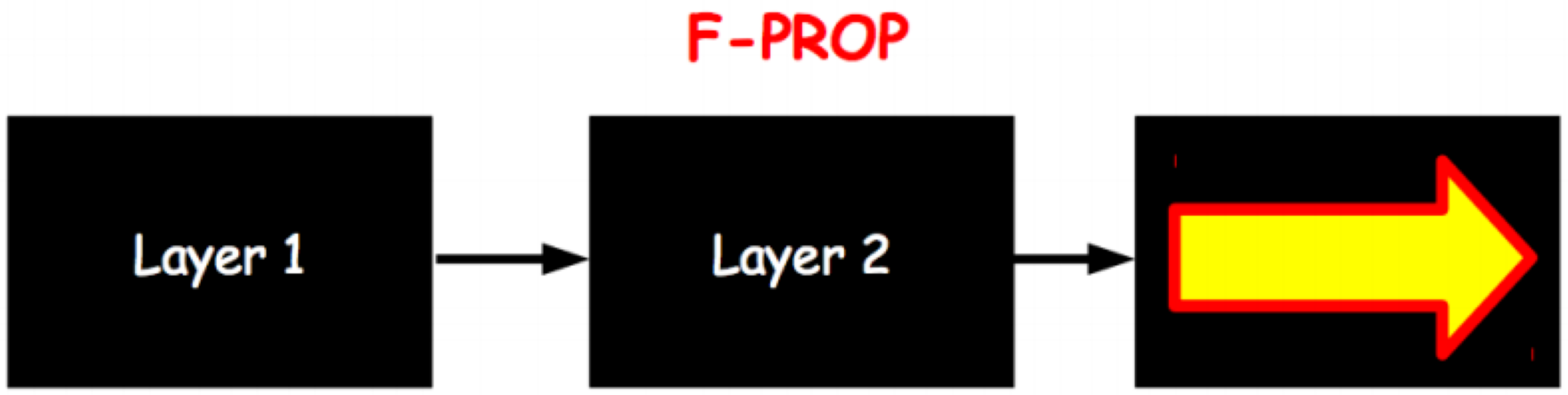
Neural Net Training

A) Compute loss on small mini-batch



Neural Net Training

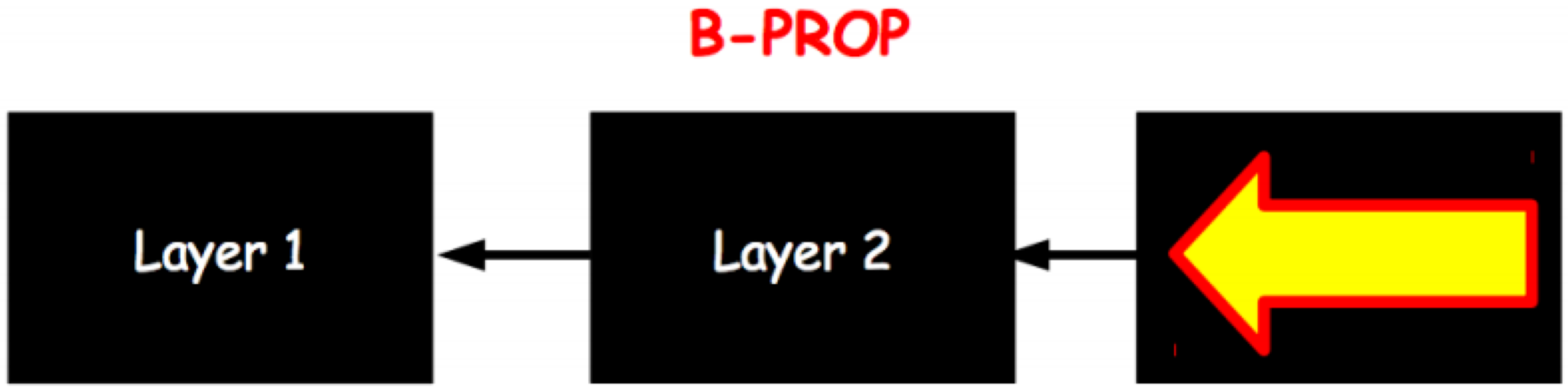
A) Compute loss on small mini-batch



Neural Net Training

A) Compute loss on small mini-batch

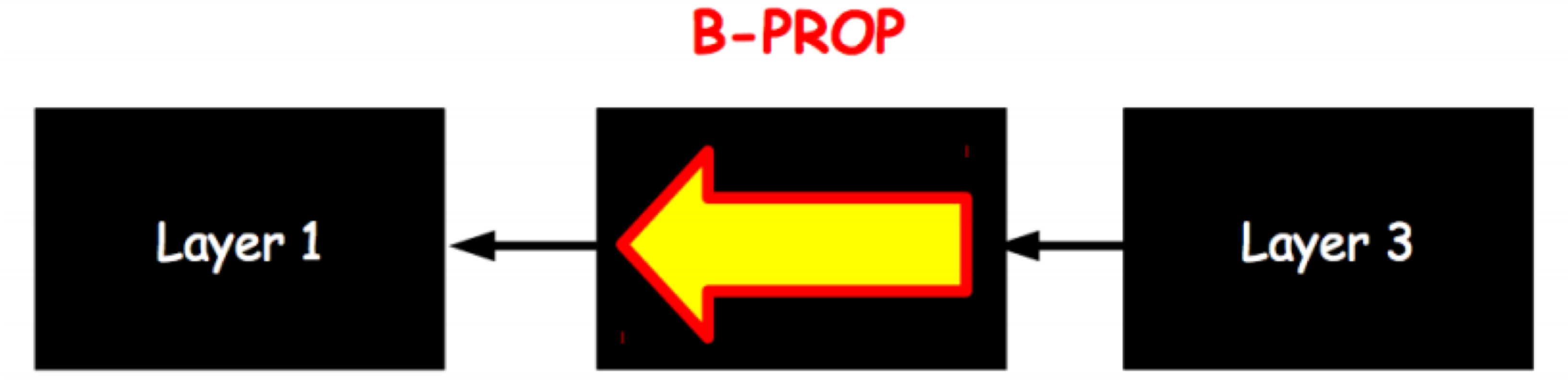
B) Compute gradient w.r.t. parameters



Neural Net Training

A) Compute loss on small mini-batch

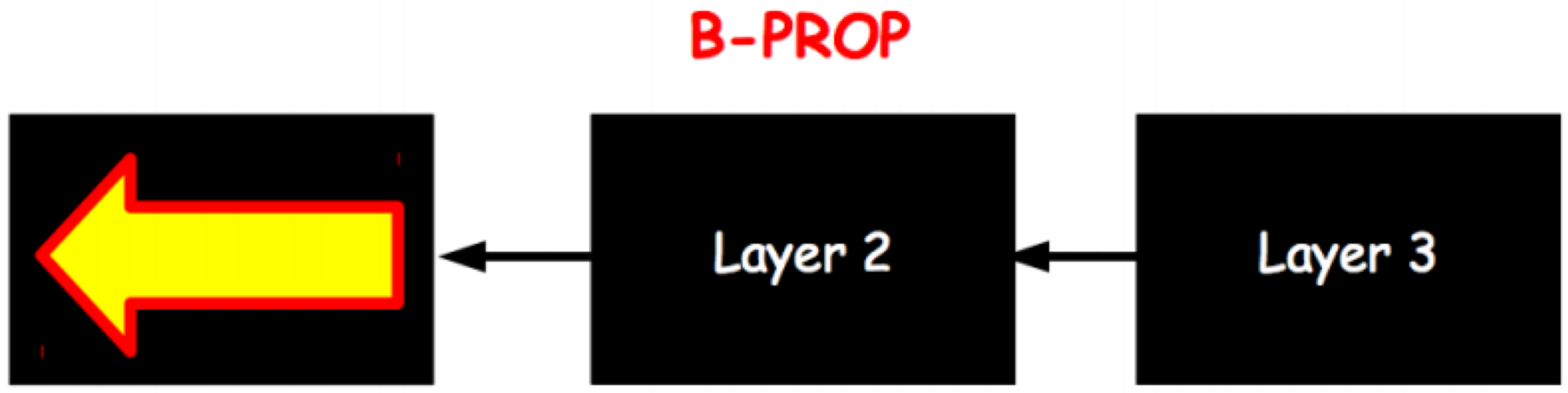
B) Compute gradient w.r.t. parameters



Neural Net Training

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters

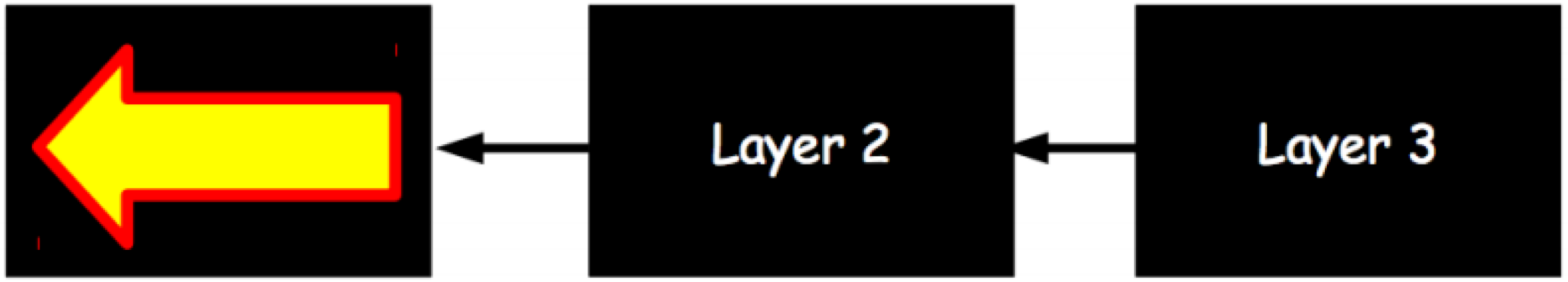


Neural Net Training

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters

C) Use gradient to update parameters $W \leftarrow W - \eta \frac{dL}{dW}$



Disadvantages

- From a memory and capacity standpoint the CNN is not much bigger than a regular two layer network.
- At runtime the convolution operations are computationally expensive and take up about 67% of the time.
- CNN's are about 3X slower than their fully connected equivalents (size wise).

Disadvantages

- Convolution operation

  -4 nested loops (2 loops on input image & 2 loops on kernel)

- Small kernel size

  -make the inner loops very inefficient as they frequently JMP.

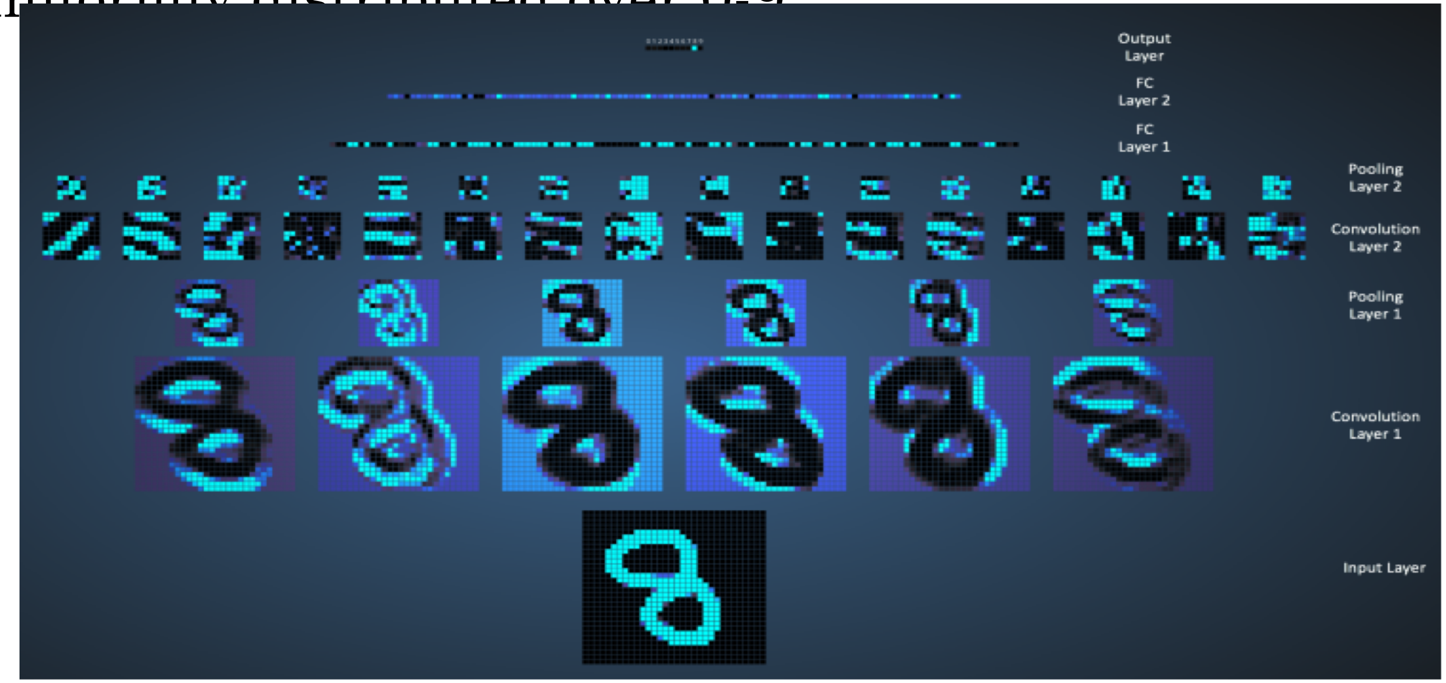
- Cash unfriendly memory access
  - Back-propagation require both row-wise and column-wise access to the input and kernel image.
  - 2D Images represented in a row-wise-serialized order.
  - Column-wise access to data can result in a high rate of cash misses in memory subsystem.

Outline

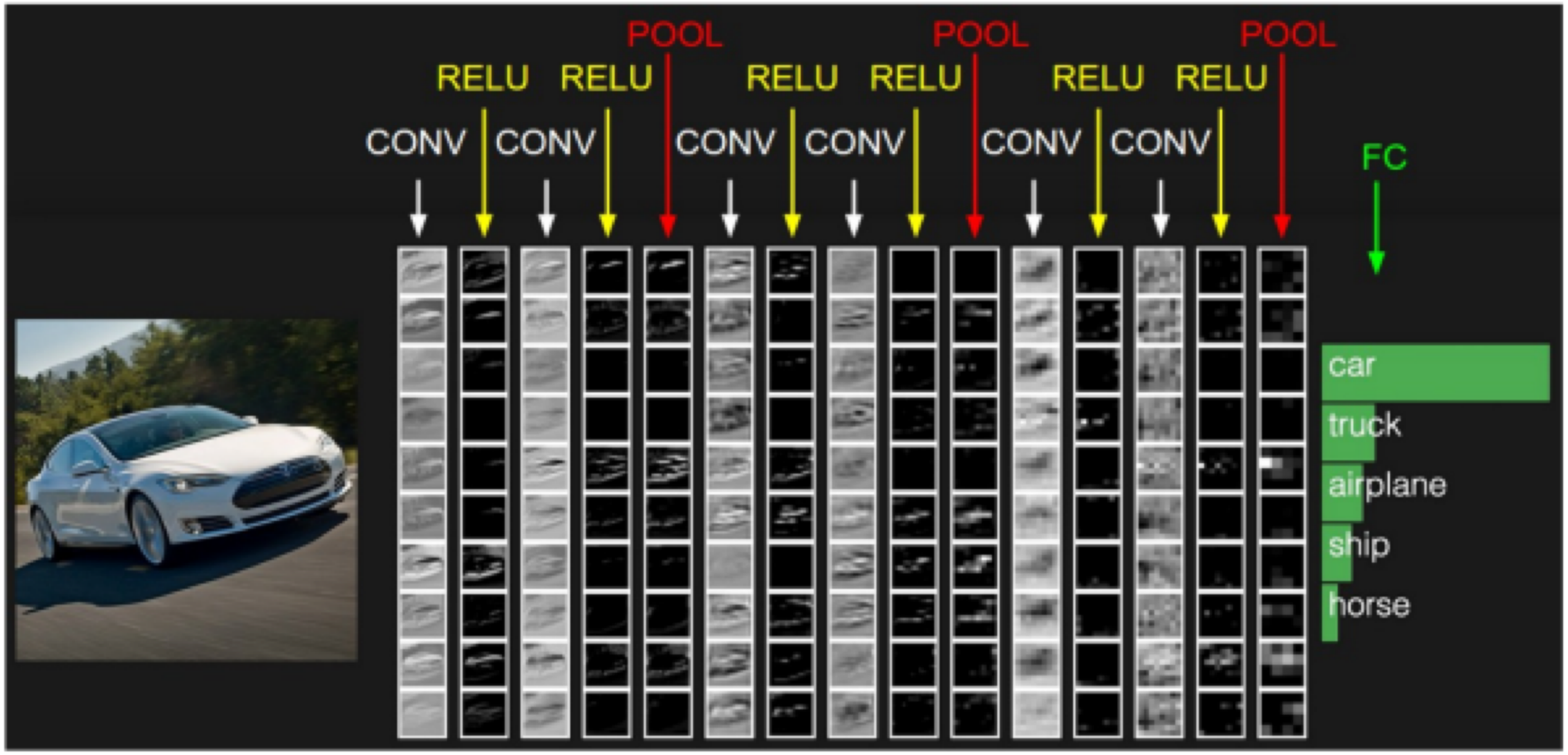
- Introduction to Covolutional Nueral Networks
- Common Layers Used in CNN
- Neural Network Training
- **Applications**

Application

- A Convolutional neural network achieves 99.26% accuracy on a modified NIST database of hand-written digits.
- MNIST database : Consist of 60,000 hand written digits uniformly distributed over 0-9

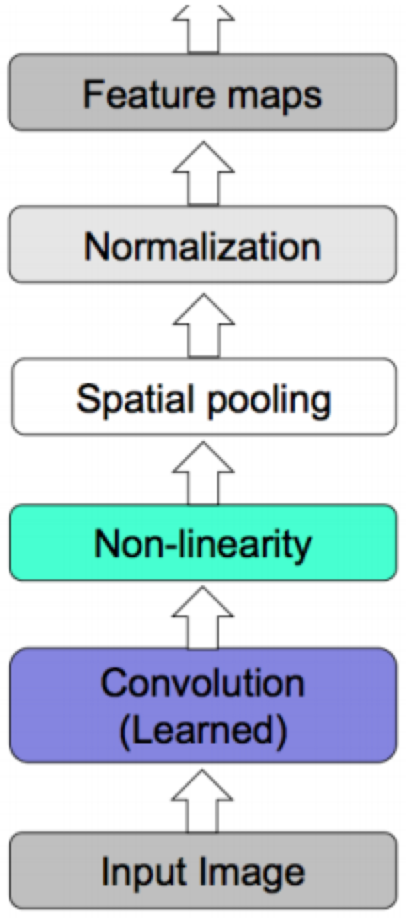


Application

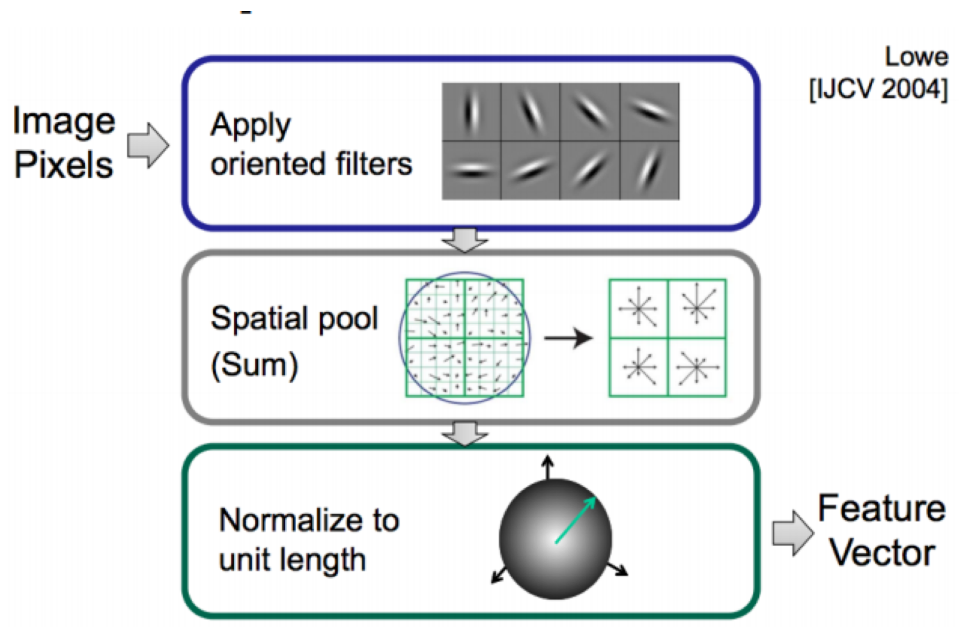


Summary of a typical convolutional layer

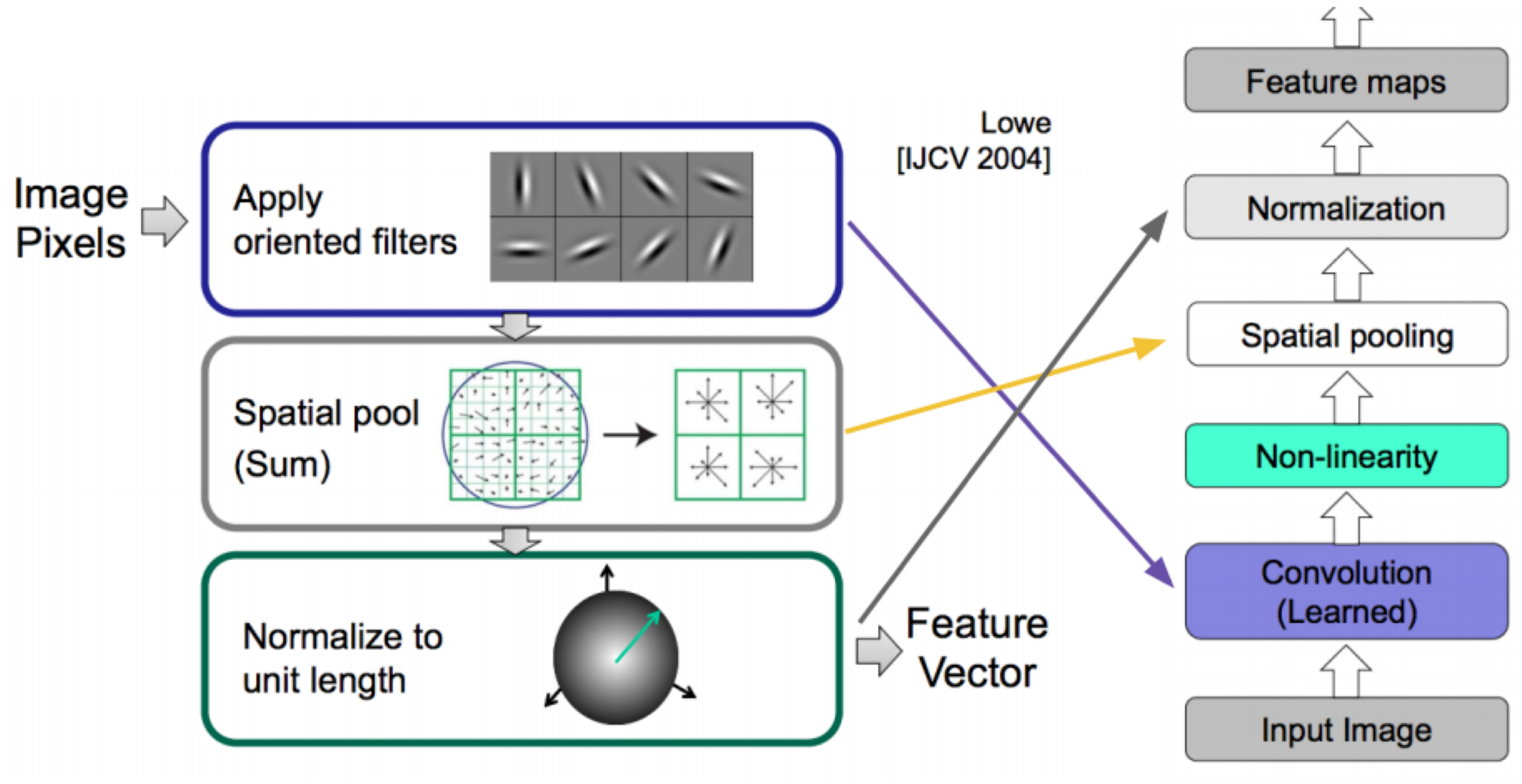
- Doing all of this consists one layer.
  - Pooling and normalization is optional.
- Stack them up and train just like multilayer neural nets.
- Final layer is usually fully connected
- neural net with output size == number of classes



Compare this to SIFT



Compare this to SIFT

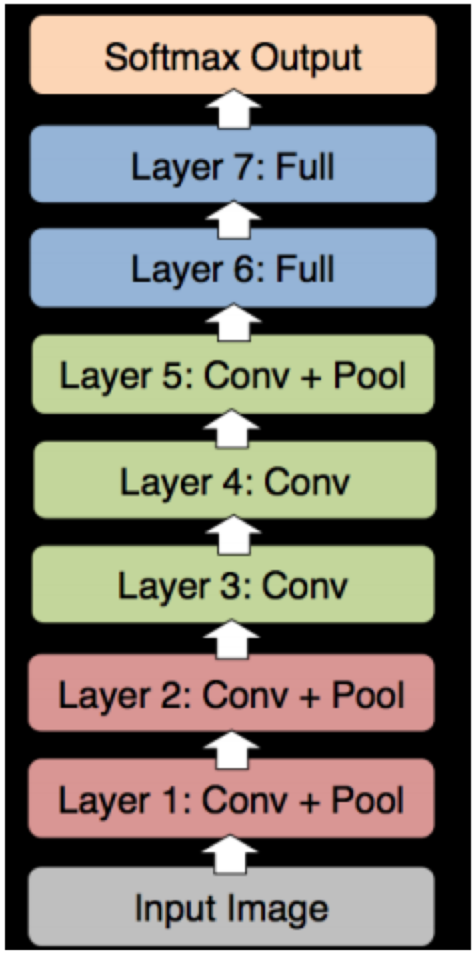


Revisiting the composition idea

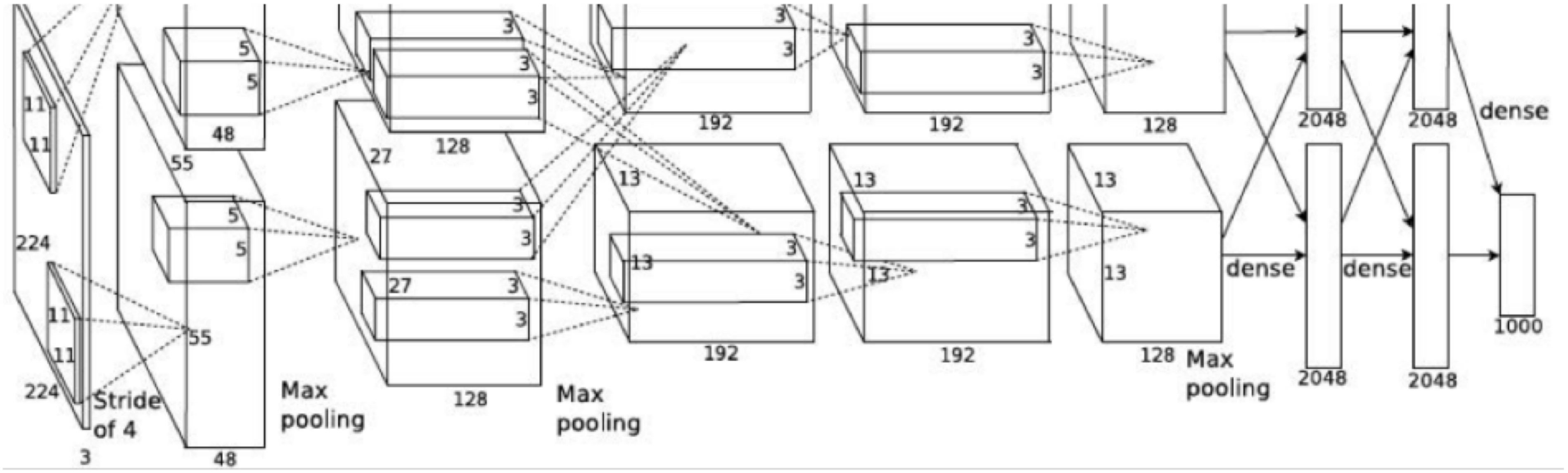
- Every layer learns a feature detector by combining the output of the layer before.
- $\Rightarrow$ More and more abstract features are learned as we stack layers.
- Keep this in mind and let's look at what kind of things ConvNets learn.

Architecture of Alex Krizhevsky et al.

- 8 layers total.
- Trained on Imagenet Dataset (1000 categories, 1.2M training images, 150k test images)
- 18.2% top-5 error
  - Winner of the ILSVRC2012 challenge

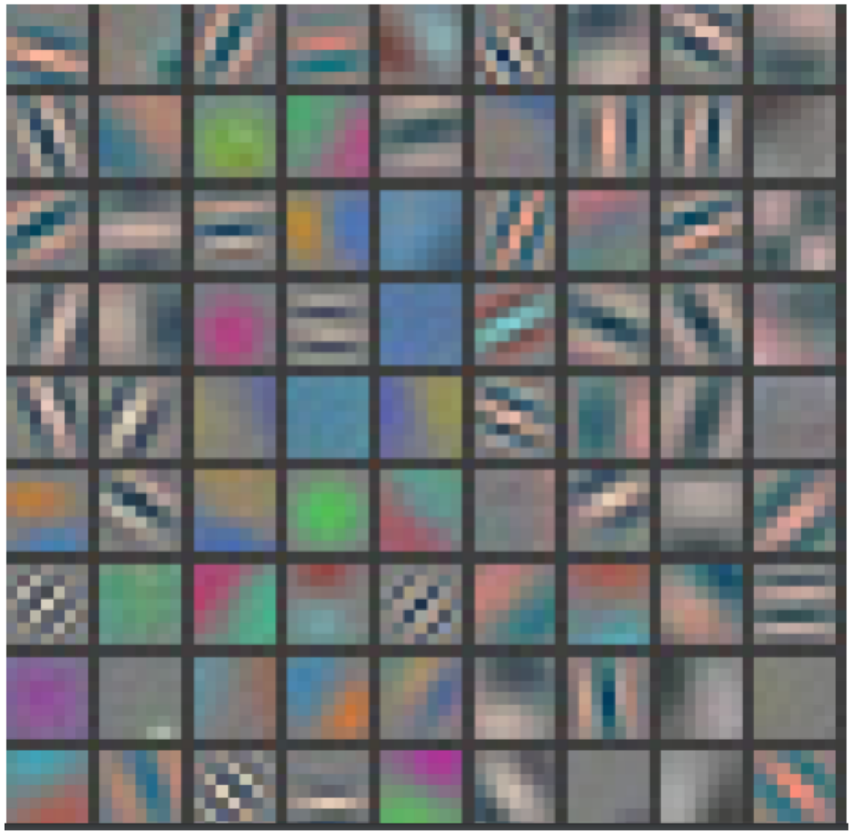


Architecture of Alex Krizhevsky et al.



First layer filters

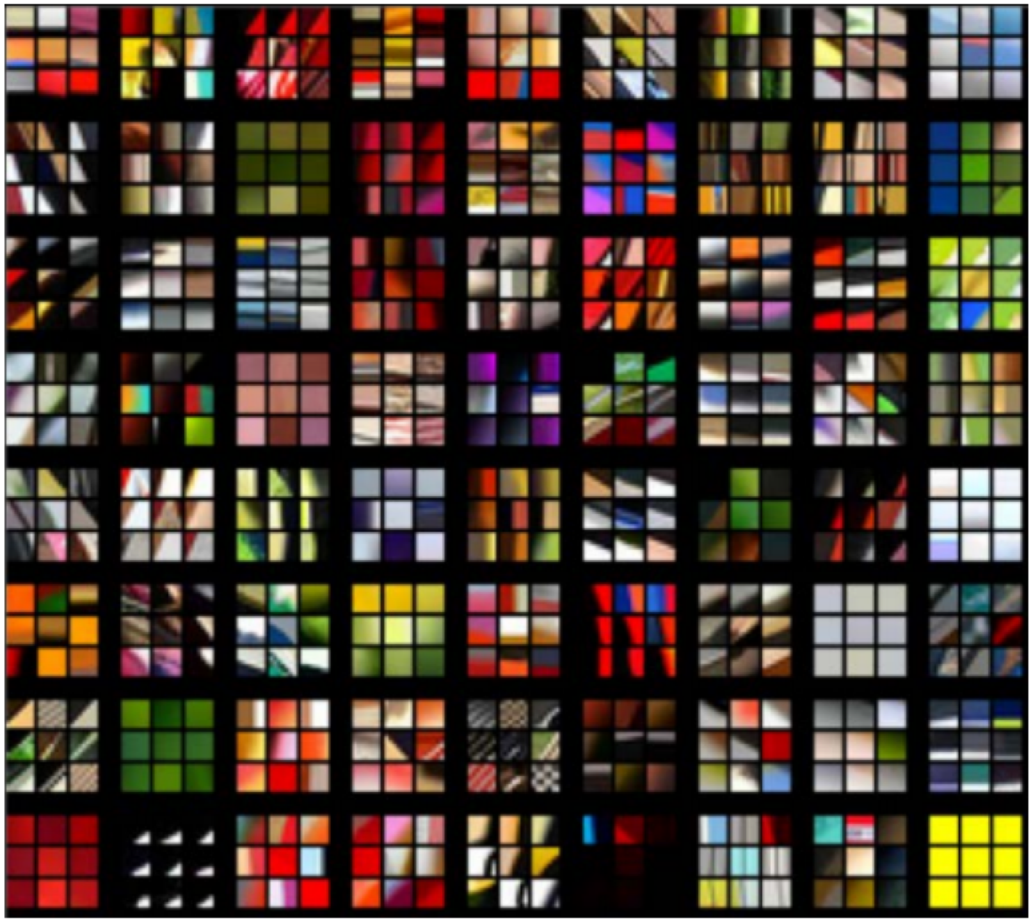
- Showing 81 filters of 11x11x3.
- Capture low-level features like oriented edges, blobs.



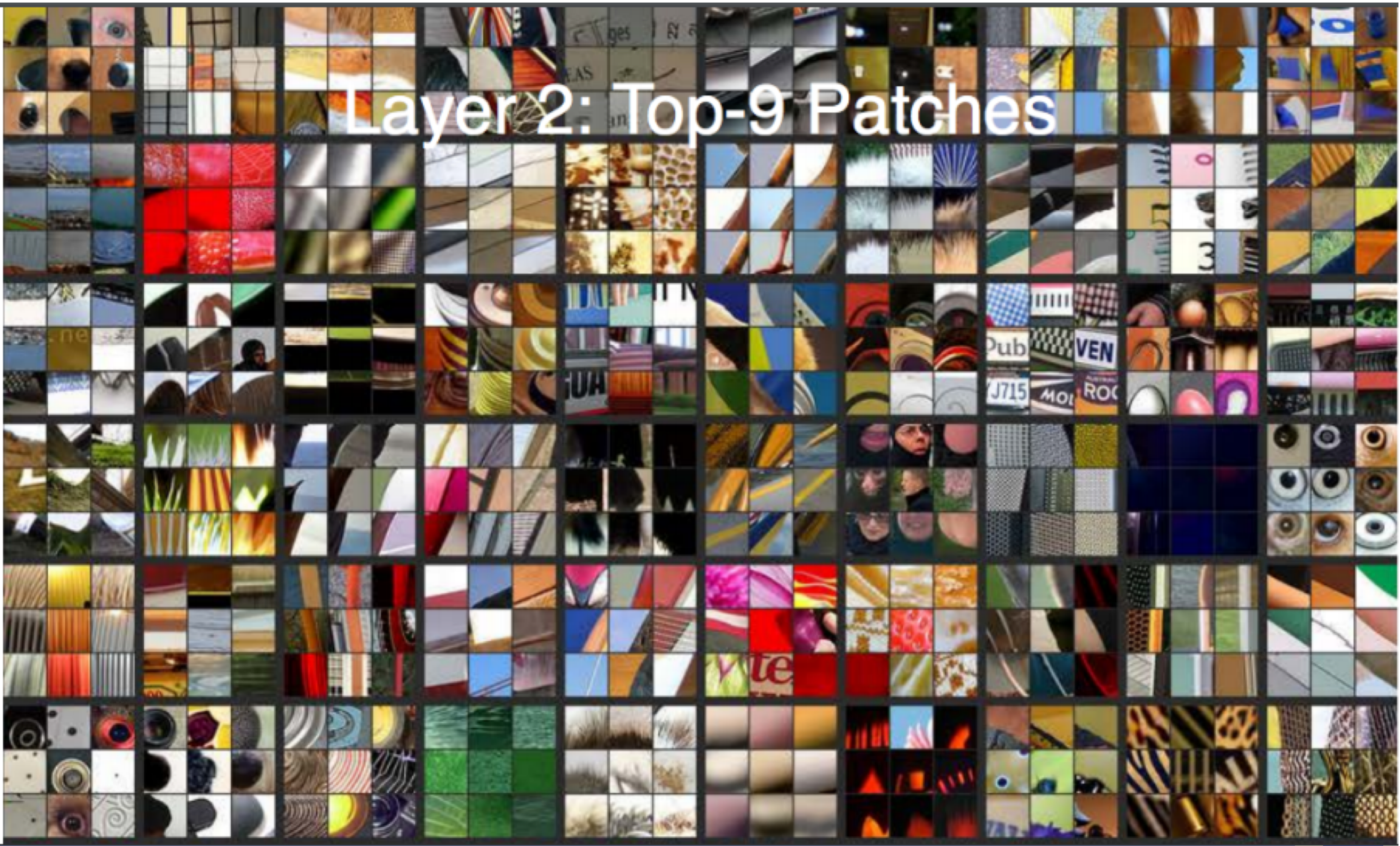
Note these oriented edges are analogous to what SIFT uses to compute the gradients.

Top 9 patches that activate each filter in layer 1

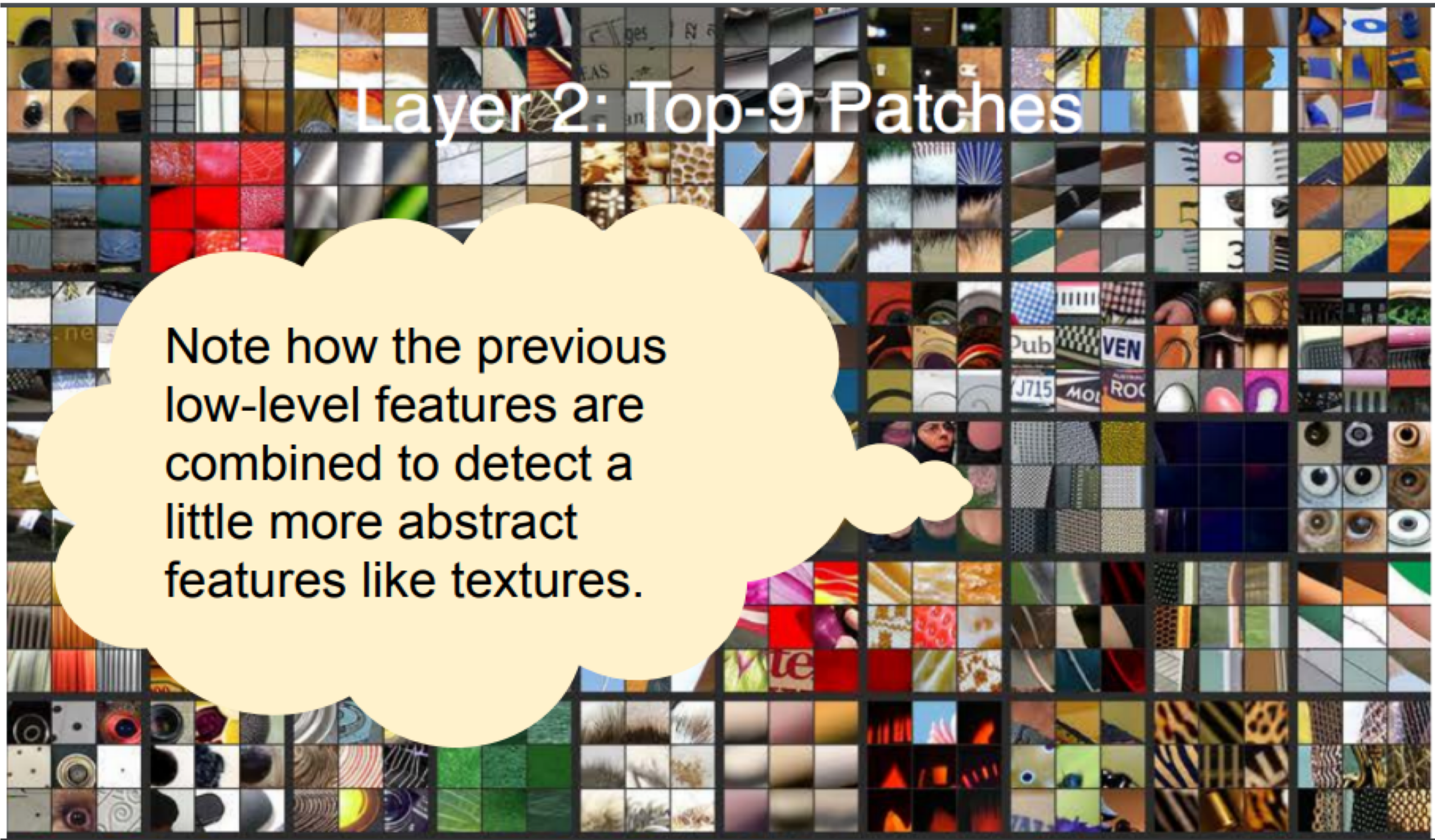
Each 3x3 block shows the top 9 patches for one filter.



Layer2: Top 9 patches

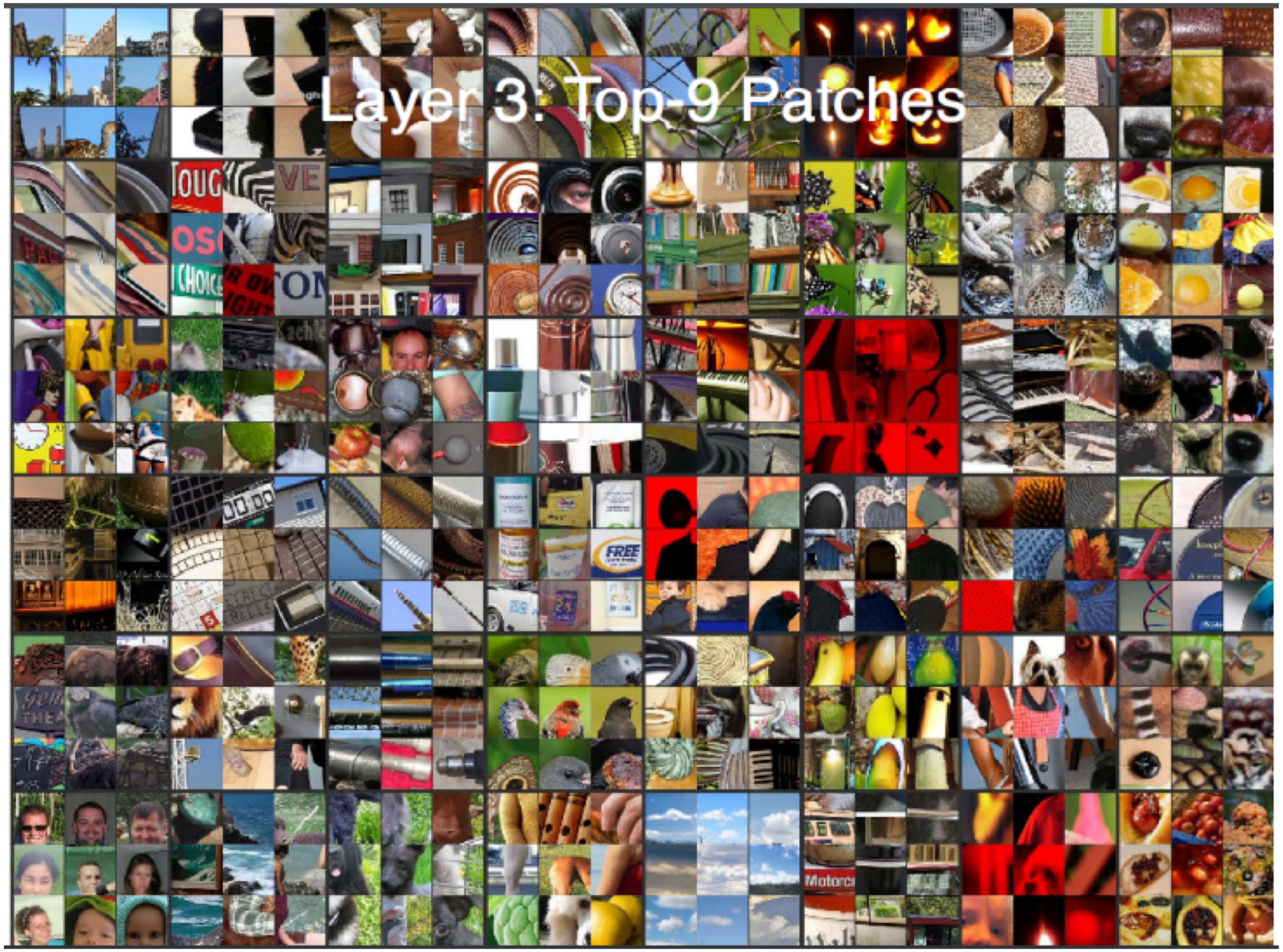


Layer2: Top 9 patches

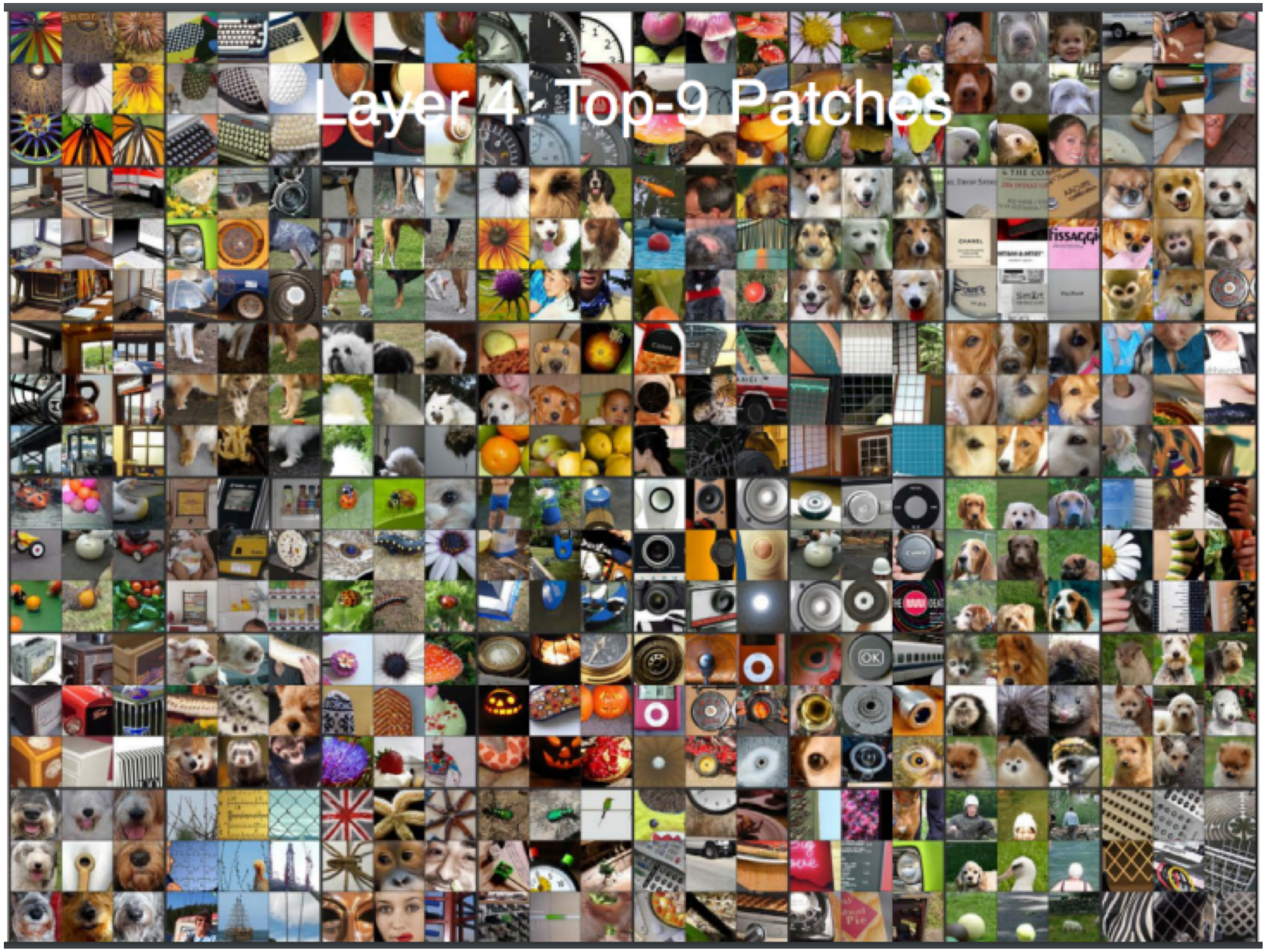


Note how the previous low-level features are combined to detect a little more abstract features like textures.

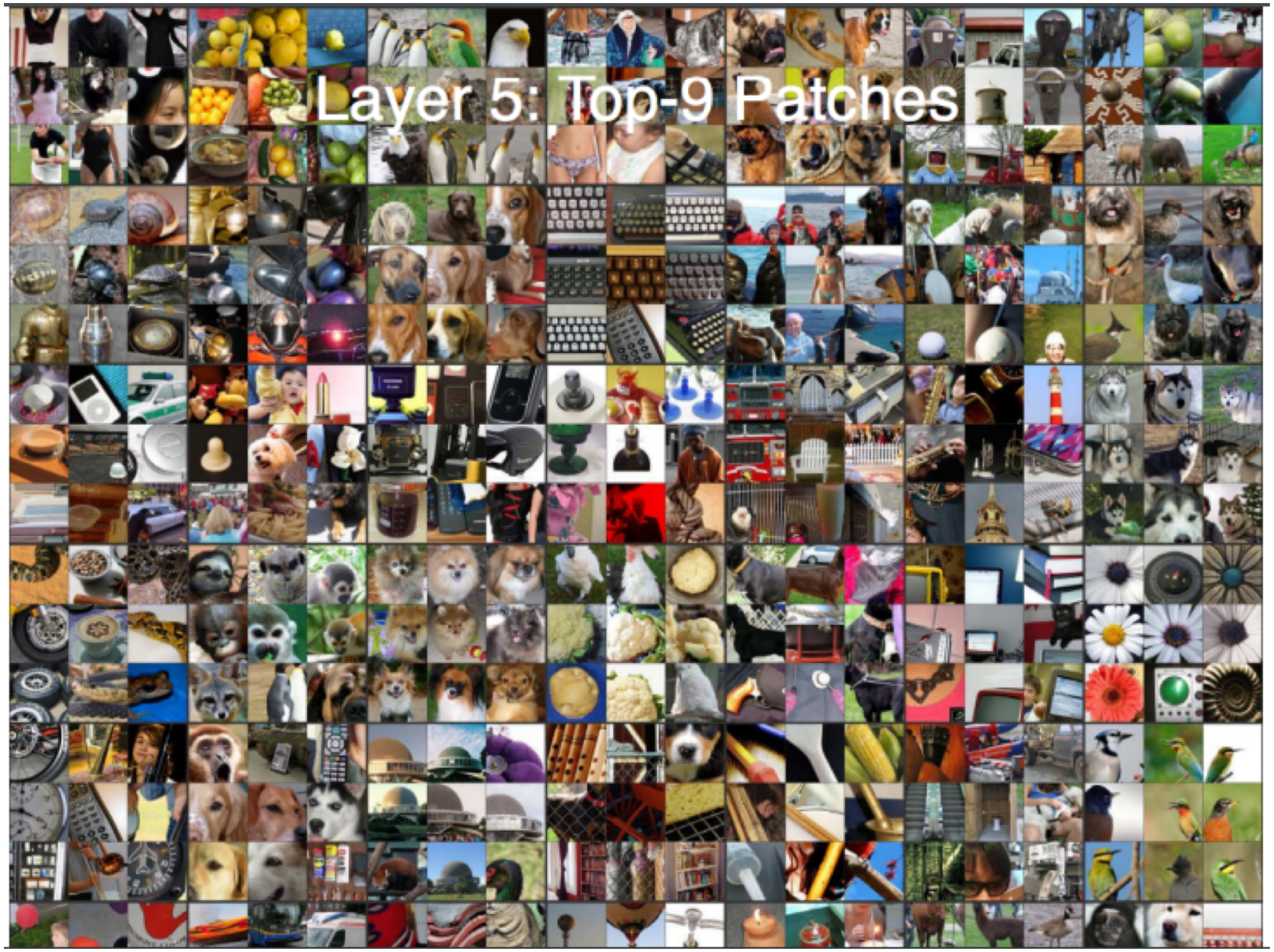
Layer3: Top 9 patches



Layer4: Top 9 patches

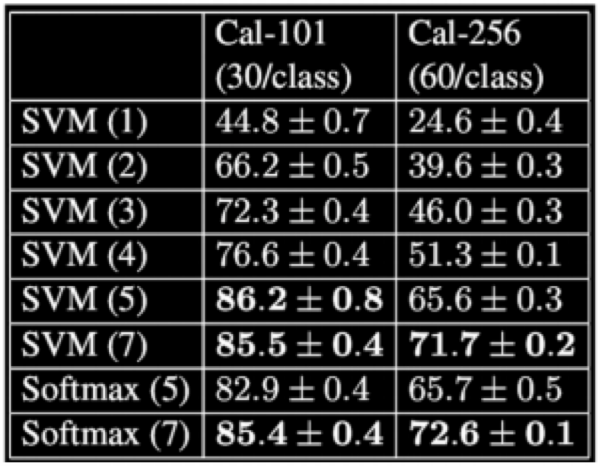


Layer5: Top 9 patches



ConvNets as generic feature extractor

- A well-trained ConvNets is an excellent feature extractor.
- Chop the network at desired layer and use the output as a feature representation to train a SVM on some other vision dataset.
- Improve further by taking a pre-trained ConvNet and re-training it on a
- different datase

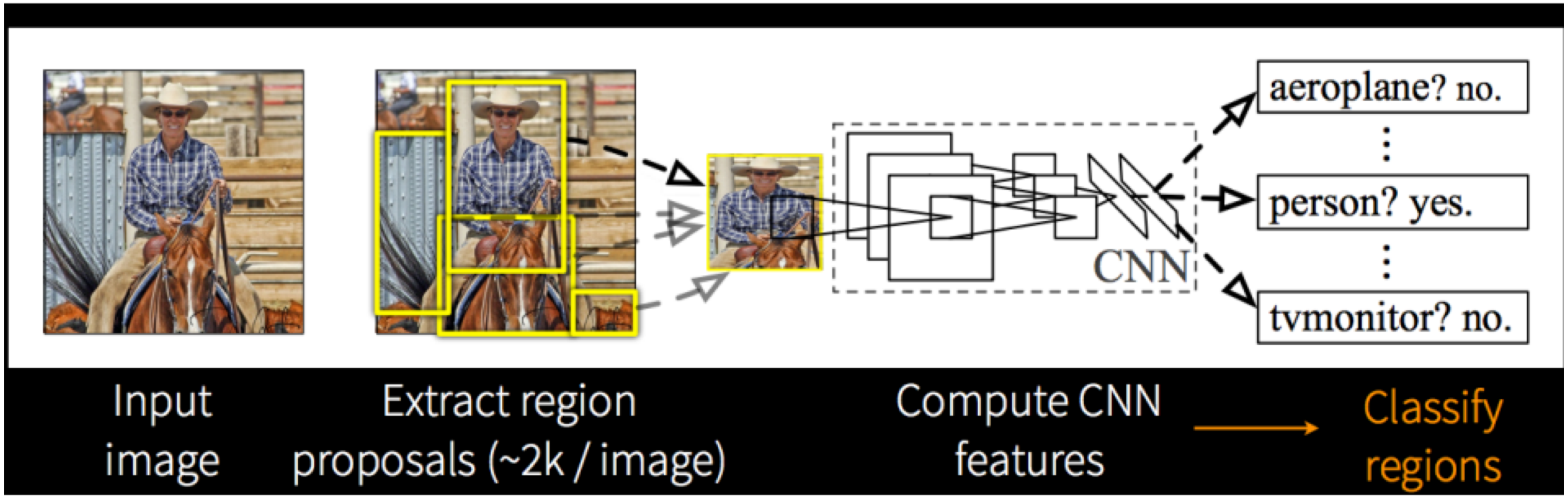


	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$
Softmax (5)	82.9 ± 0.4	65.7 ± 0.5
Softmax (7)	$\mathbf{85.4 \pm 0.4}$	$\mathbf{72.6 \pm 0.1}$

One way to do detection with ConvNets

- Since ConvNets extract discriminative features, one can crop images at the object bounding box and train a good SVM on each category.
- ⇒ Extract regions that are likely to have objects, then apply ConvNet + SVM on each and use the confidence to do maximum suppression.

R-CNN: Regions with CNN features



- Best performing method on PASCAL 2012
- Detection improving previous methods by 30%

ConvNet Libraries

- Tensorflow
- PyTorch
- Caffe
- Torch
- Theano
- ....

Q & A