Introduction to CNNs

by: Dominiquo Santistevan

Original project written by
Chandani Doshi, Dominiquo Santistevan, Kevin Ng, Tania Yu, Dalitso Banda, Emmanuel Azuh, Qingyang Tan

Overview

1. Motivation
2. Basic Classification
3. Fully Connected Neural Network
4. Convolutional Neural Network

Motivation

What is in our “world”?

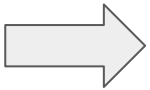
Managing visual sensory input

Distinguishing objects


Sign
STOP
Buckingham

Machine Learning Approach

“the study and construction of algorithms that can learn from and make predictions on data“ - wikipedia



History Excursion

1950’s: Frank Rosenblatt’s first Neural Network in 1957

1970’s: Too much enthusiasm followed by disappointment causing a reduction in funding in AI research known as the **AI Winter**

1980’s: Symbolic reasoning with facts and rules

1990’s: Resurgence of machine learning with new goal of targeting solvable problems probabilistically

Machine Learning Approach

X: n vectors of length m that represent our n images.

Y: n vectors containing labels for our images

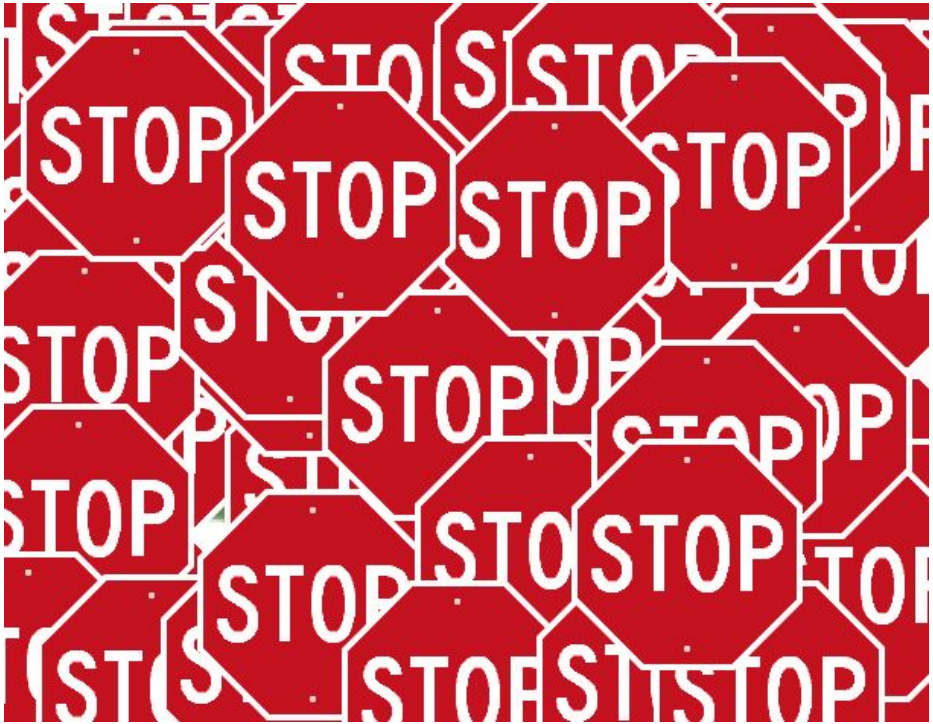
Θ: Model Coefficients

We want to train our Θ with X and Y such that our model will produce Y’ (k vectors of length 1) for a new X’ (k vectors of length m)

Training Our Models

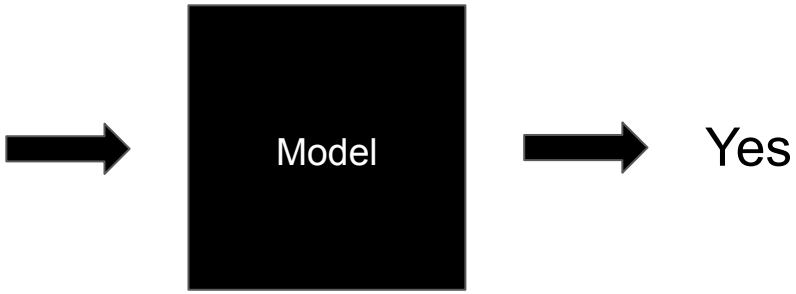
Model needs to see MANY examples to be robust

Learn model parameters to make later judgements



Discriminative Models: Labeling Images

Is this a stop sign?



Before the black box was the magic black box..

Untrained Model

?????

Before the black box was the magic black box..

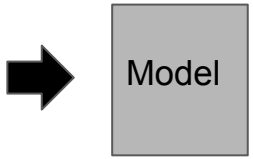
Untrained Model

¯_(ツ)_/¯

Discriminative Models: Limitations

The model needs a lot of data about our world to be robust

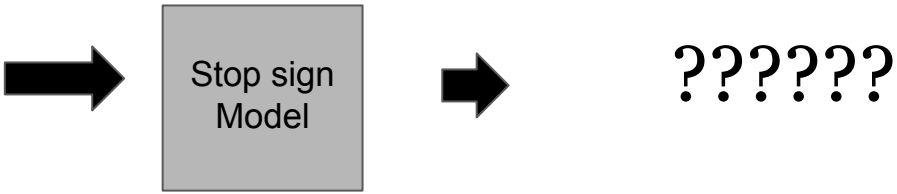
The model can only assign labels that we've seen before



Stop Sign!

Discriminative Models: Limitations

Is this a person?



Discriminative Models: Limitations

Stop sign Model

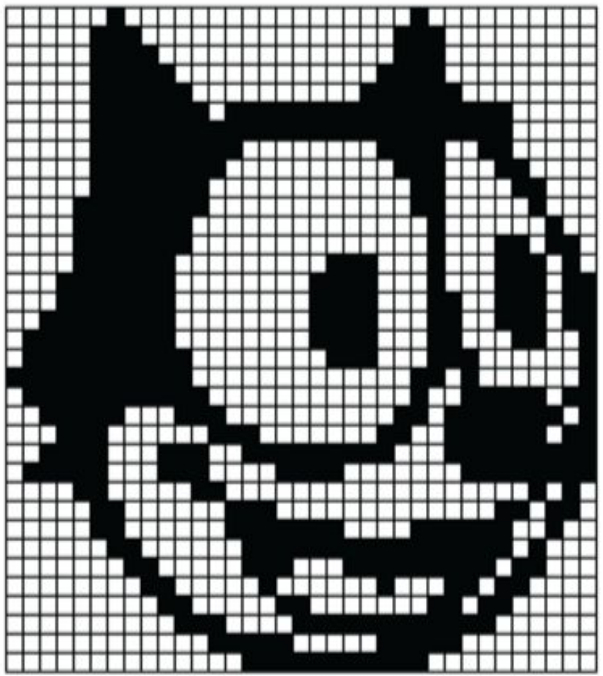
¯_(ツ)_/¯

Toolkit

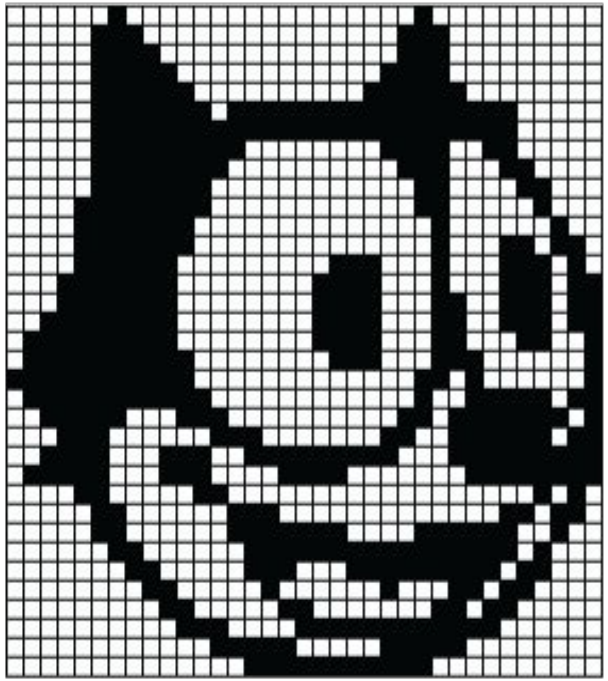
Toolkit

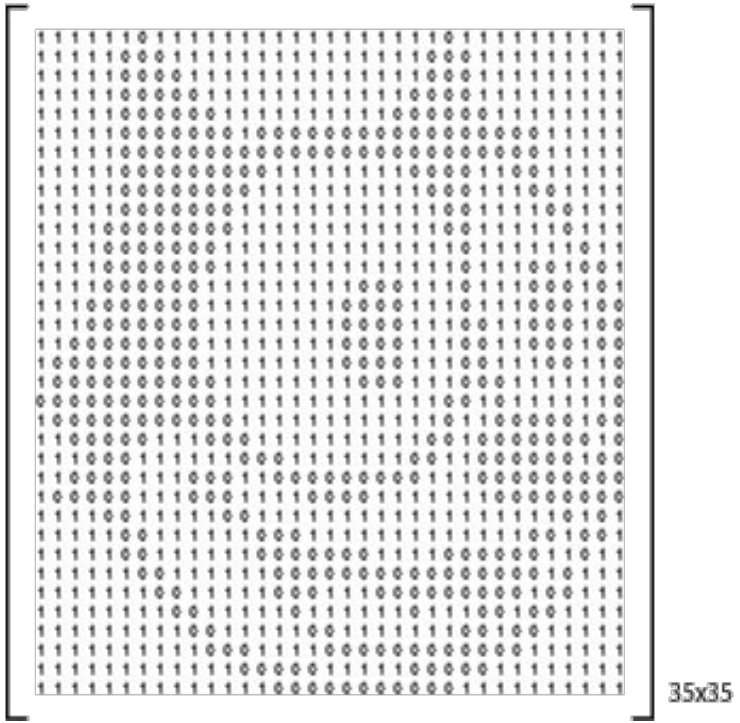
Vectorize

How does a computer “see” an image?



How does a computer “see” an image?





Representing an Image

Many ways to process an image

Histogram of oriented gradients (HOG), Canny Edge Detector, etc.

Focus on naive vectorization for now

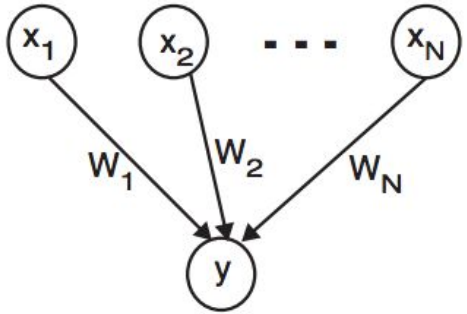
Perceptron

X : [x_1, x_2, …, x_n]

W: [w_1, w_2, …, w_n]

y : true label

Produce a decision boundary



Perceptron Example: Stop Sign

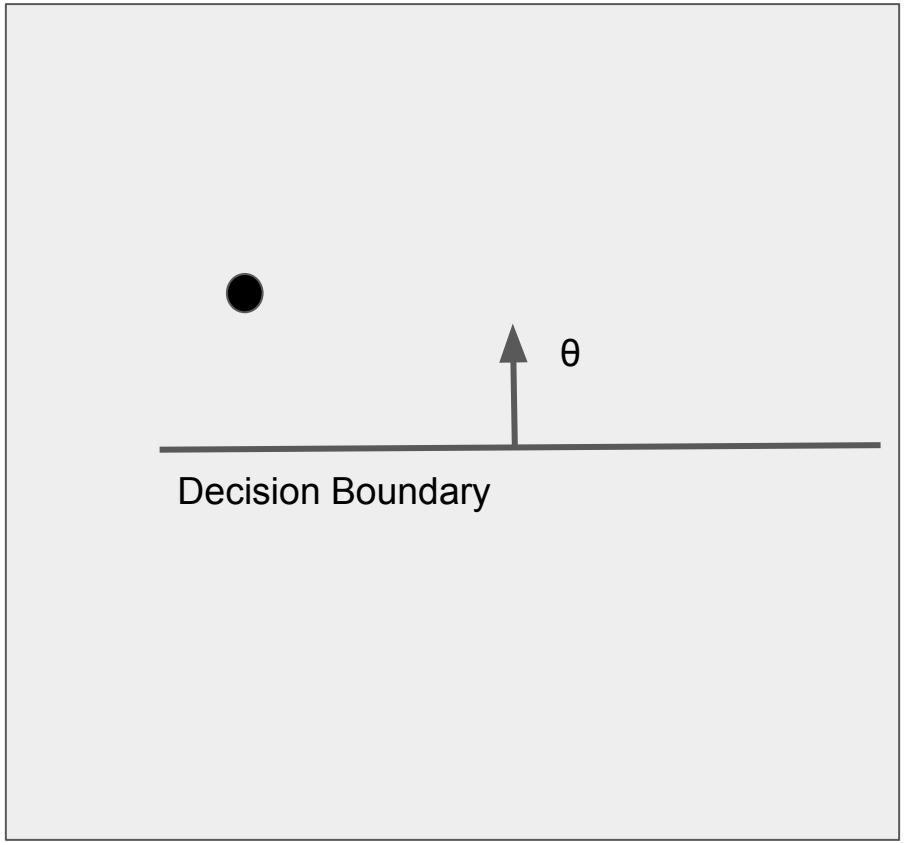
x: Stop Sign Vector

Θ: Model Parameters

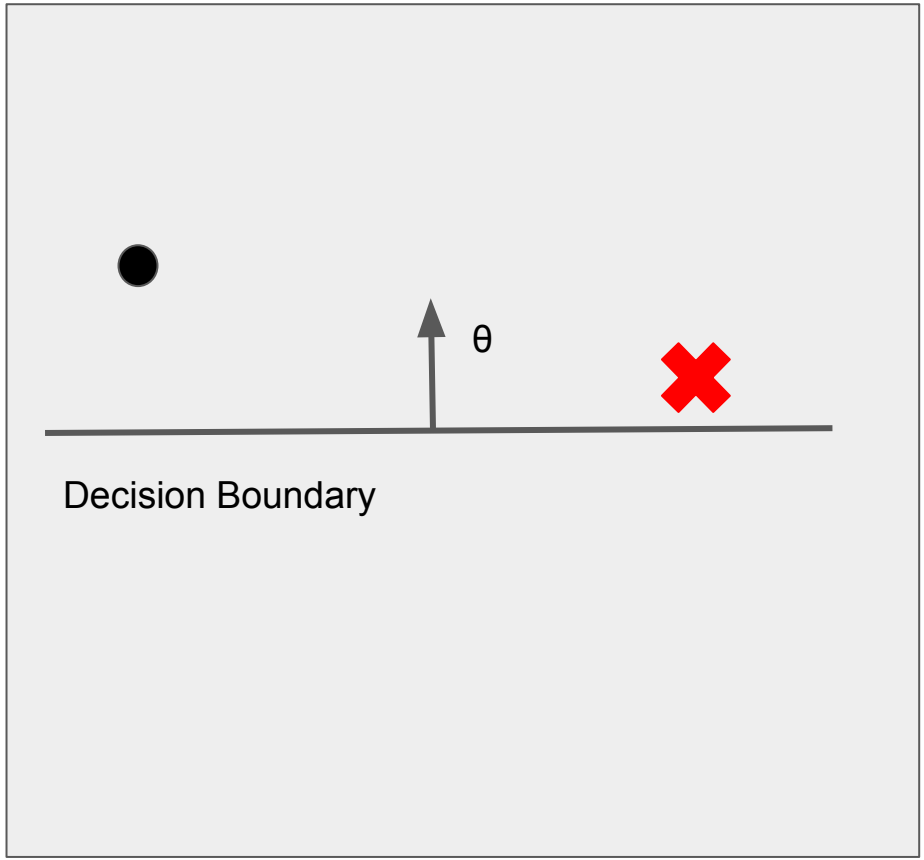
f(x,Θ): Classification Model

$$f(\mathbf{x};\theta) = \text{sign}\left(\theta_1 x_1 + \ldots + \theta_d x_d\right) = \text{sign}\left(\theta^T \mathbf{x}\right)$$

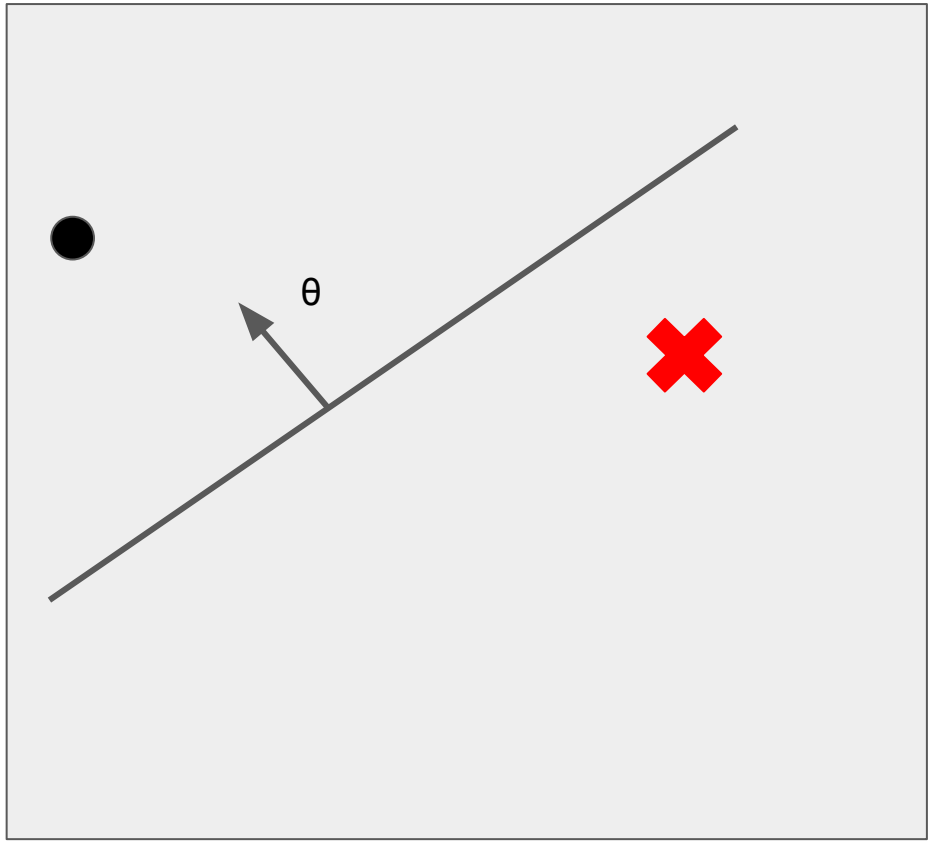
$\theta = [\theta_1, \ldots, \theta_d]^T$ is a column vector of real valued parameters.

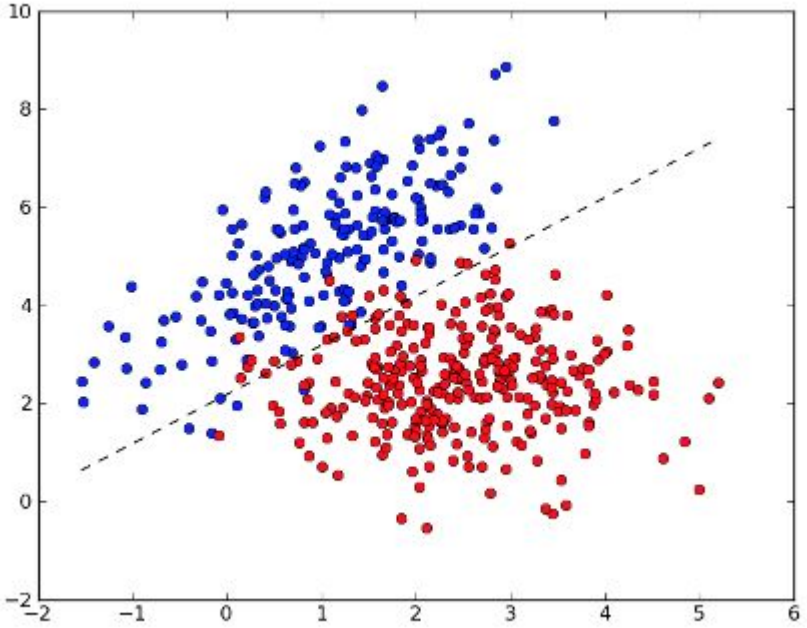


$$\theta' \leftarrow \theta + y_t \mathbf{x}_t \quad \text{if} \quad y_t \neq f(\mathbf{x}_t; \theta)$$



Perceptron updates decision boundary



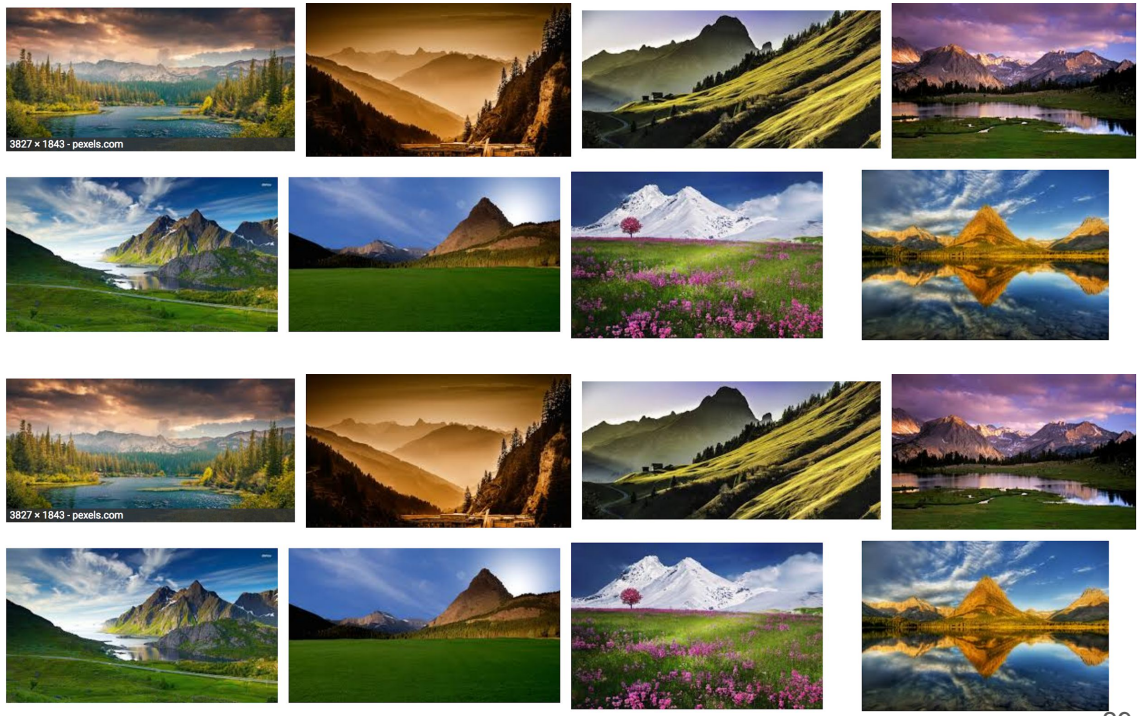

10
8
6
4
2
0
−2
−2
−1
0
1
2
3
4
5
6

Toolkit

Vectorize 

Perceptron

What is a “good” model?



What is a “good” model?

16 landscape images

1 stop sign image

If our model learns to labels every image: “Not a stop sign”

16/17 ≈ 94% correctly labeled

Is this good??

What is a “good” model?

Recall - Of all points label=y, how many did the model correctly label y?

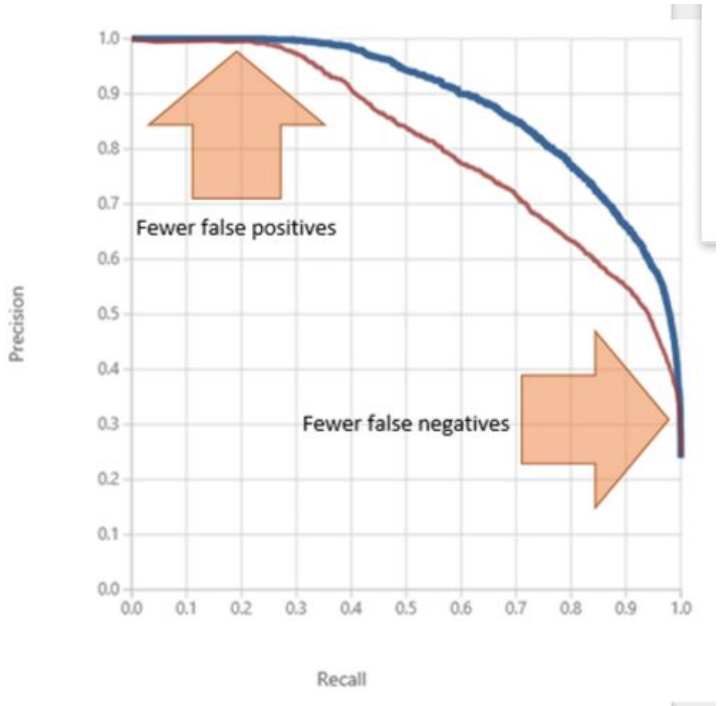
For all images labeled stop sign, how many were correctly labeled?

0.00%

What is a “good” model?

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$



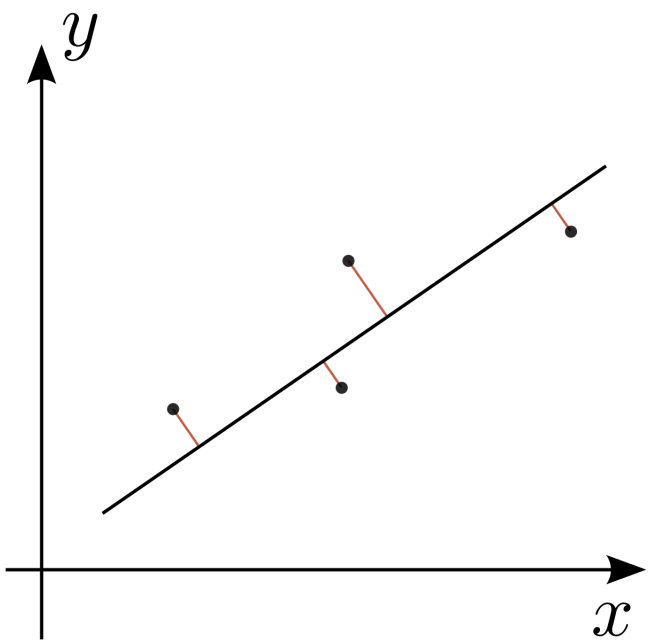
Toolkit

Vectorize 

Perceptron

Precision/Recall

How does our model get better??



Model Error

Mean Squared Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Mean Absolute Error

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}.$$

But there are several others (Hinge Loss, cross entropy loss, etc).

Toolkit

Vectorize 

Perceptron

Precision/Recall

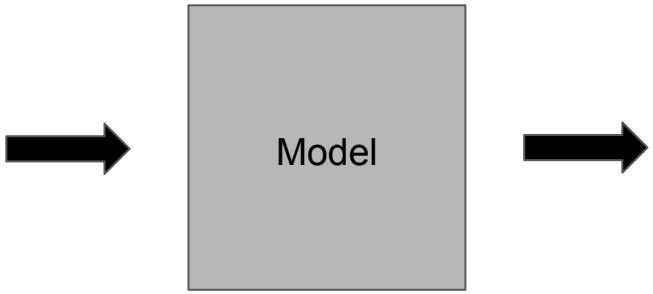
Model Loss

Handling multiple labels

What if we want our model to do more than say yes or no to a question?

What if we want to be able to assign from a set of labels?

Multiple Labels



Stop Sign: 99%

Pedestrian: .5%

Car: .5%

Handling multiple labels

Softmax regression used in multi-class classification problems

Labels are **mutually exclusive**

Softmax regression consists of ten linear classifiers of the form:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1 | x^{(i)}; \theta) \\ p(y^{(i)} = 2 | x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k | x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Toolkit

Vectorize 

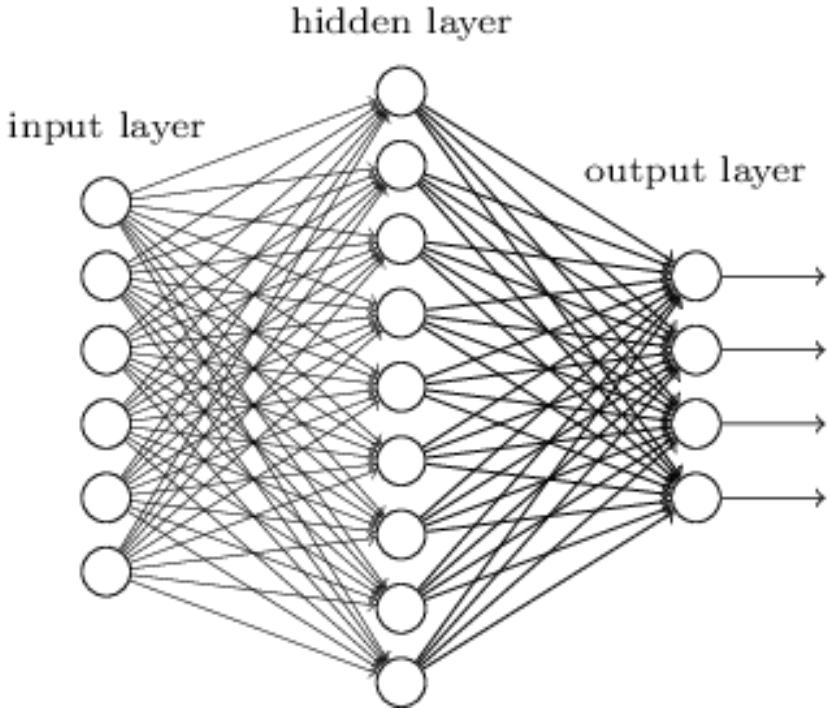
Perceptron

Precision/Recall

Model Loss

Multi-class labeling

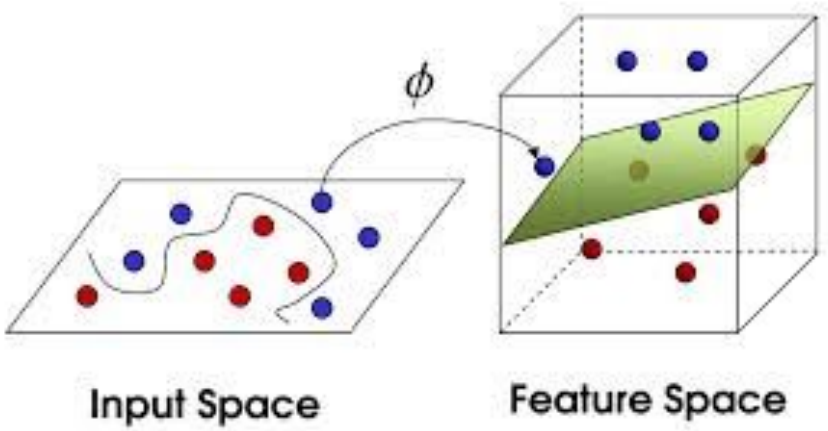
Fully Connected Neural Net (FCNN)



Why is this special?

Much like kernel functions, the non linearities of the hidden layers allow us to be more robust.

We can essentially “learn” this kernel function.



How do we learn coefficients for FCNN?

Recall for perceptron: $\theta' \leftarrow \theta + y_t \mathbf{x}_t \text{ if } y_t \neq f(\mathbf{x}_t; \theta)$

We need to get a little more sophisticated.

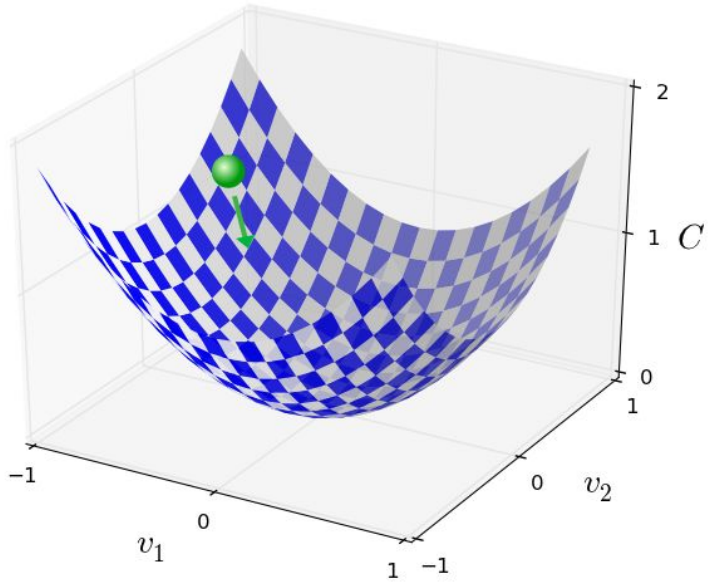
Also recall our **loss function** L(x,y,Θ)

How do we learn coefficients for FCNN?

Stochastic Gradient Descent

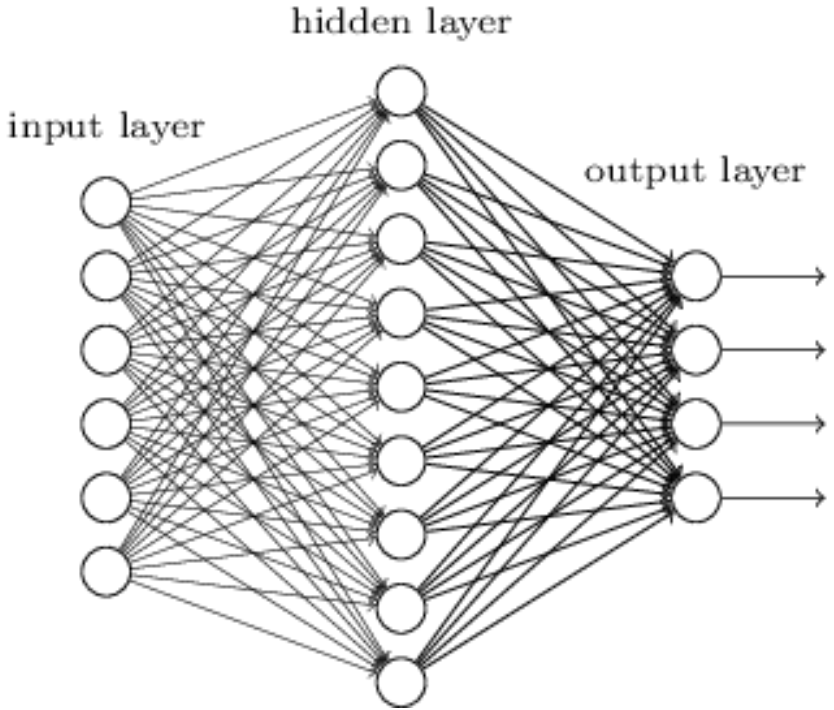
BACKPROPAGATION!

Walk model in the direction of smaller loss using gradients and sampling



Stochastic Gradient Descent: Backpropagation

A layer's input is dependent on the values from the previous layer and so on…



Stochastic Gradient Descent: Backpropagation

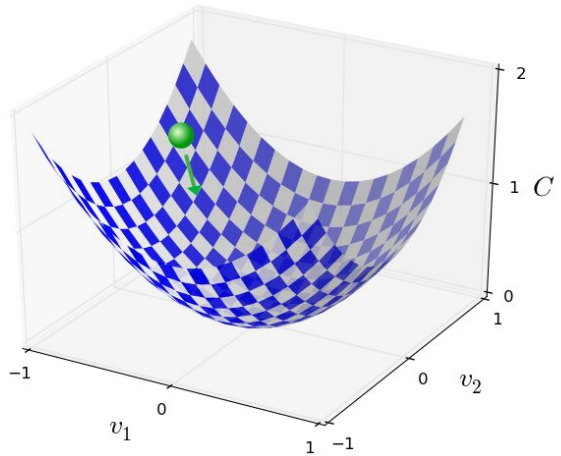
E: Squared Error of model

O_j: Output for neuron j

W_ij: Weight between neurons i,j

Net_j: weighted sum of outputs from layer

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net_j}} \frac{\partial \mathrm{net_j}}{\partial w_{ij}}$$



Toolkit

Vectorize 

Perceptron

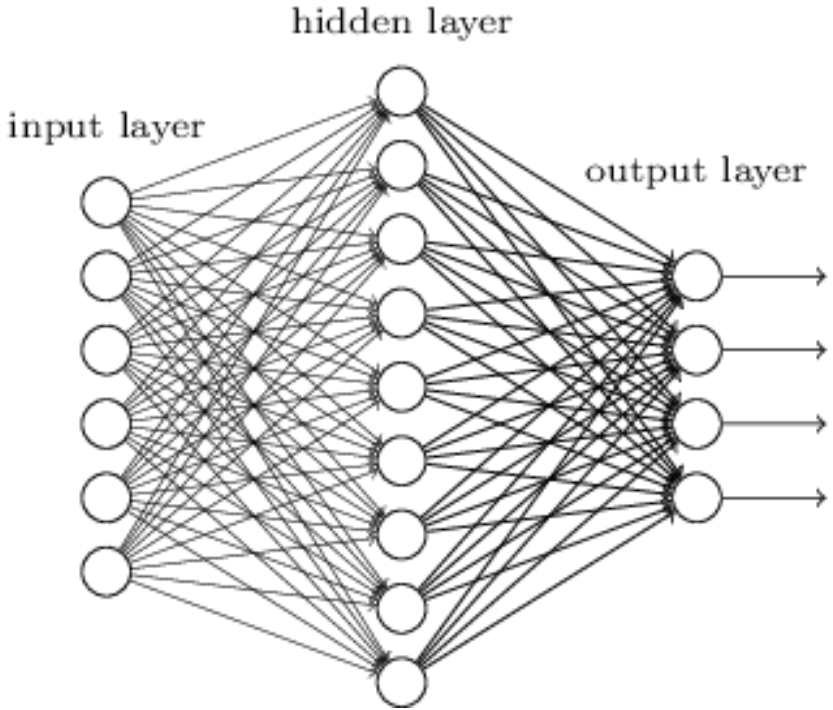
Precision/Recall

Model Loss

Multi-class labeling

Fully Connected Neural Network

Issues with Fully Connected Neural Net

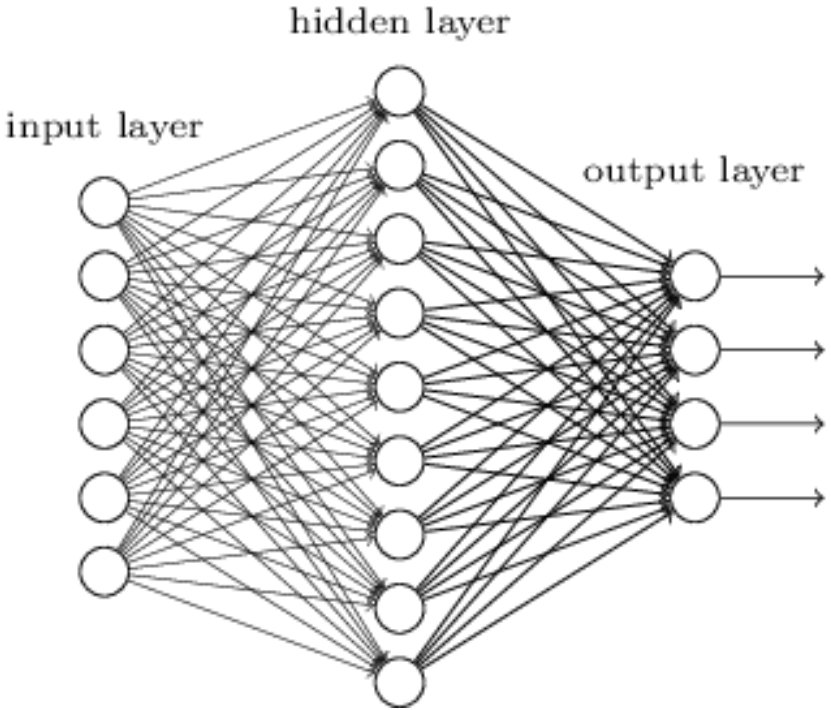


Issues with Fully Connected Neural Net

Many parameters; several hidden layers

Very dense; training is resource intensive

Does not exploit locality of images; Too general



What are some sources of error for our FCNN?

What are some sources of error for our FCNN?

Viewpoint variation

Illumination

Scale

Deformation

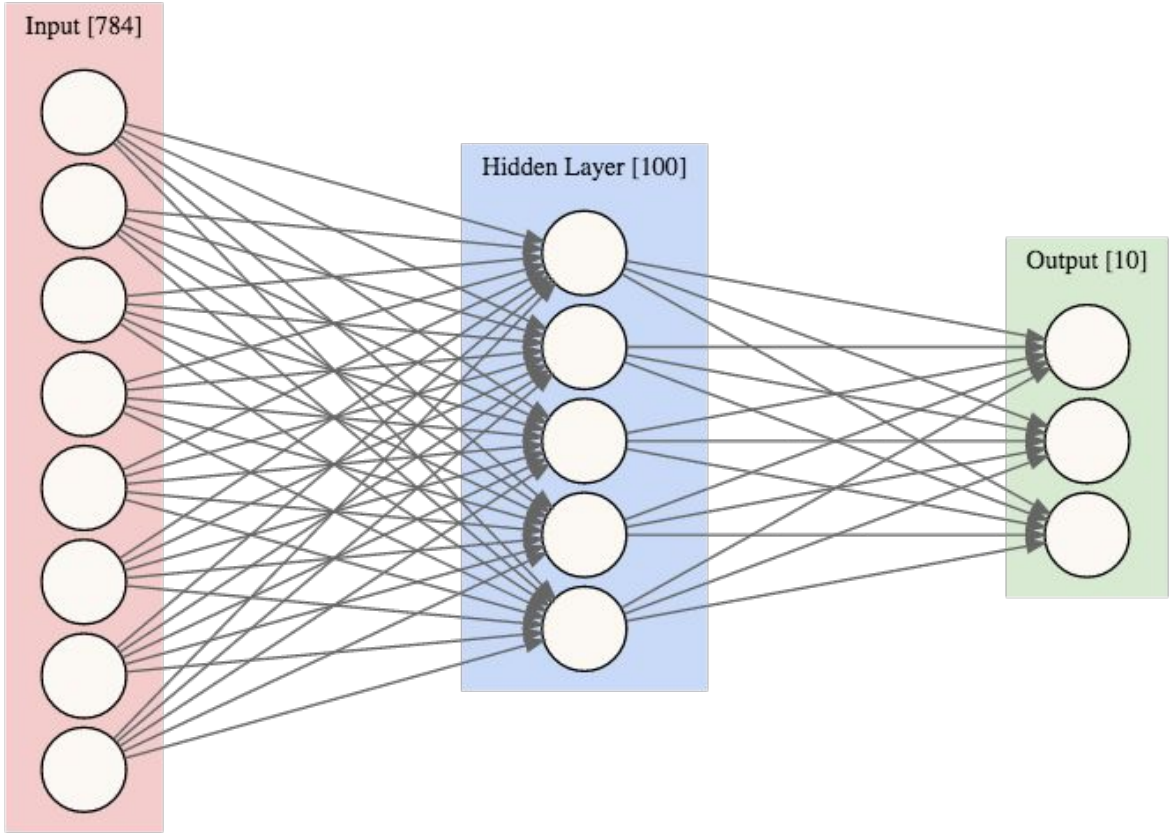
Intra-class variation

Background clutter

Viewpoint Variation

Illumination

How Do We Handle Too Many Parameters?



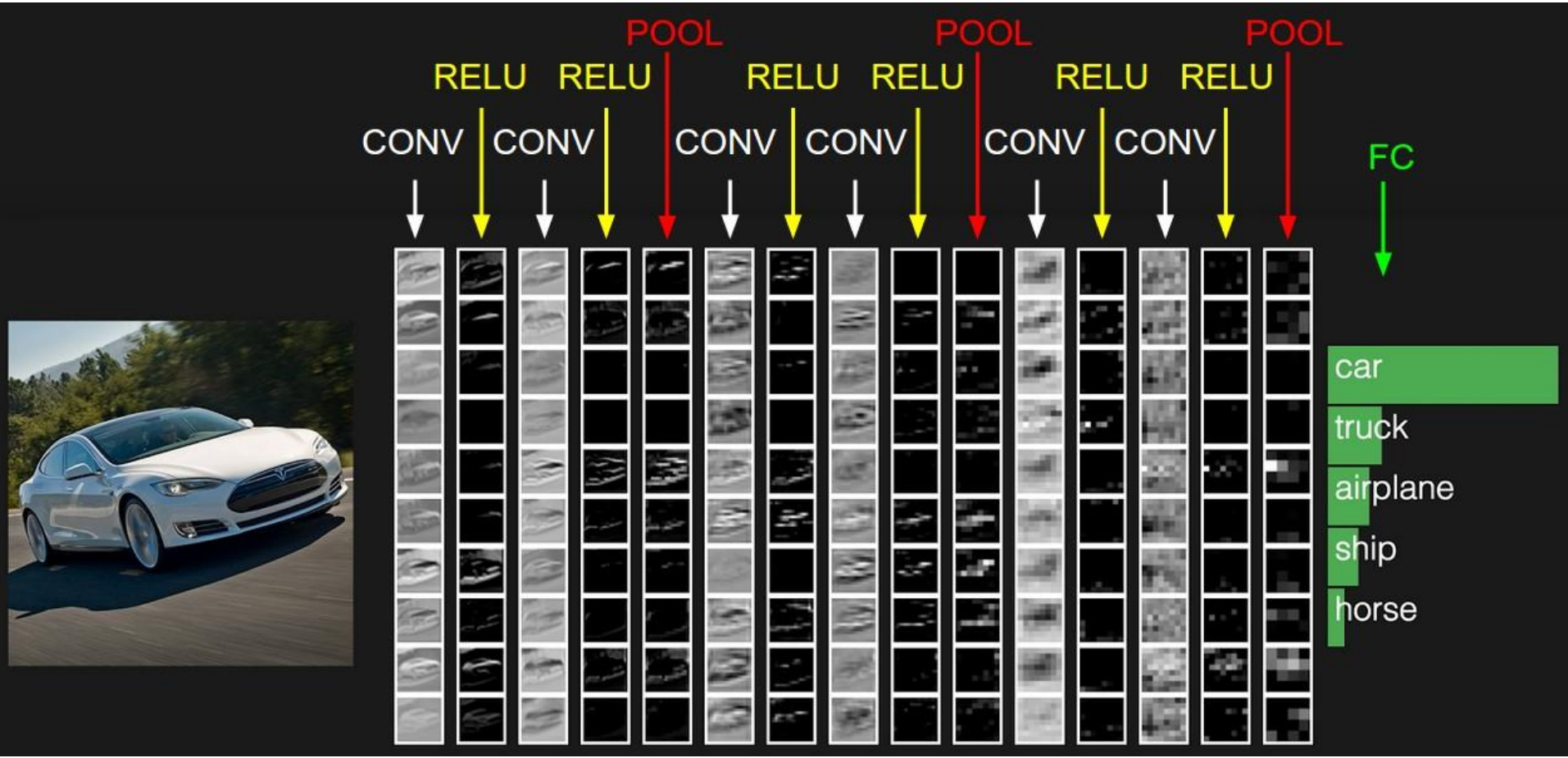
Convolutional Neural Network

- Type of FFNN inspired by the visual cortex of a cat (kinda)
- Several layers of convolutions with nonlinear activation functions
- Use convolutions over input layer to compute output

Layers of CNN

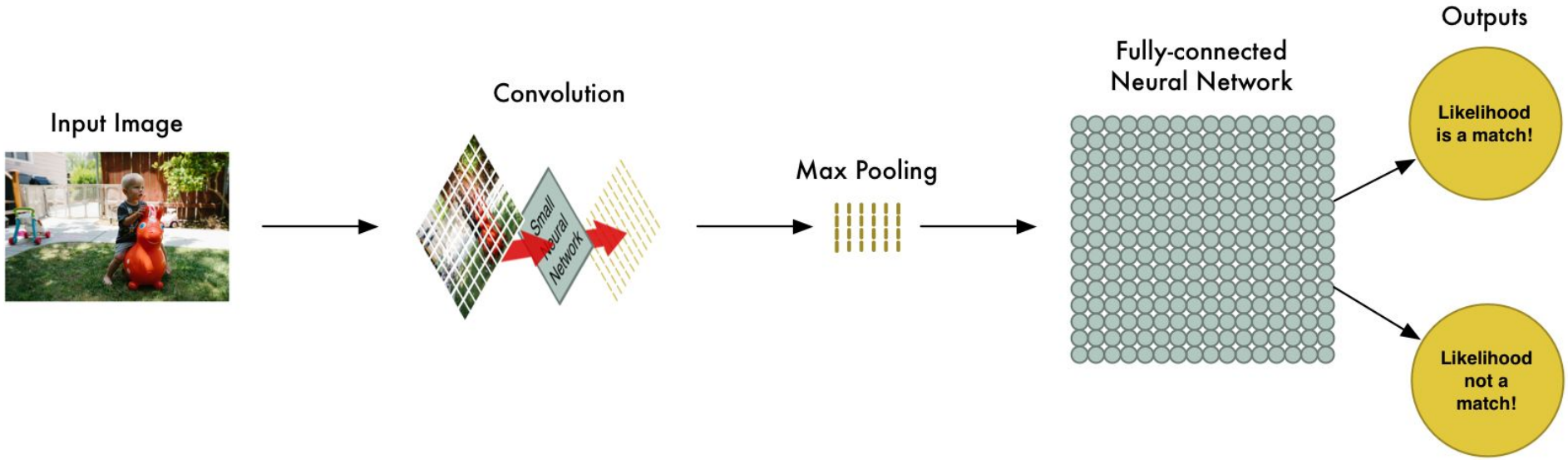
Three commonly used layers:

1. Convolution Layers
2. Pooling/Subsampling Layers
3. Fully Connected Layers


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

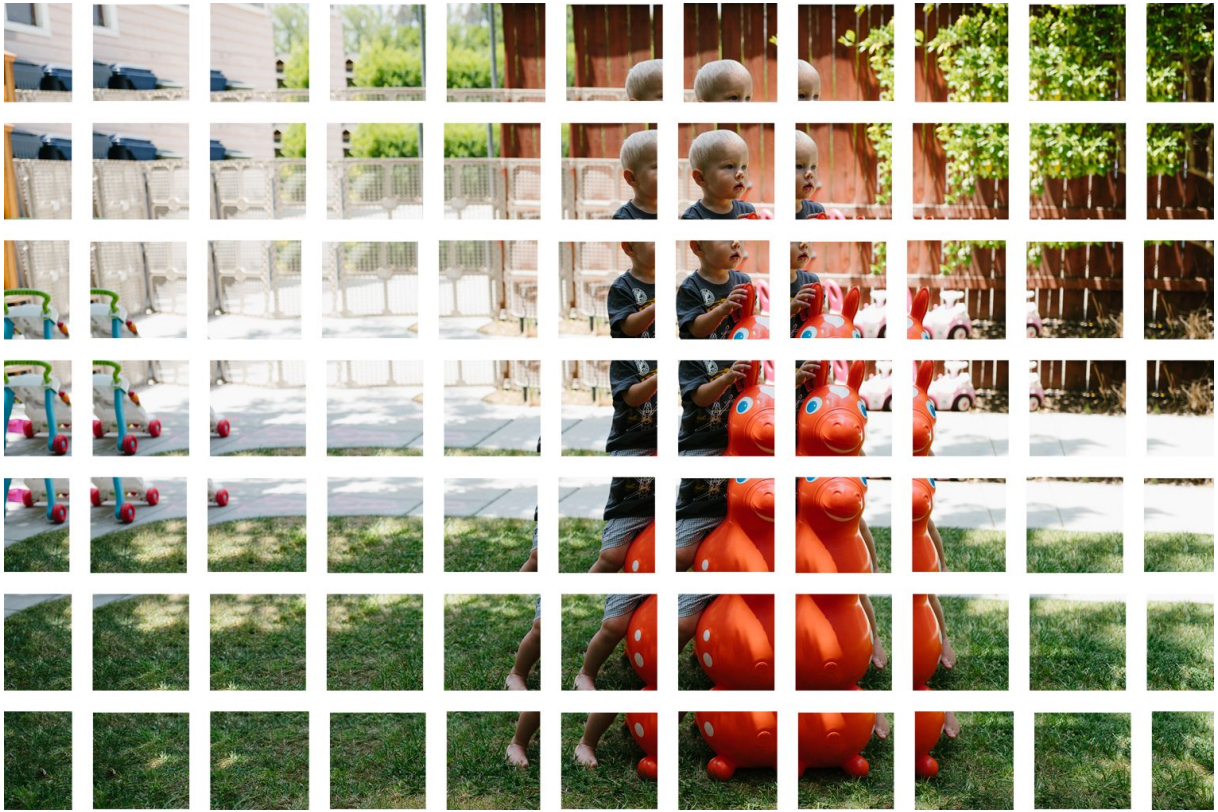
A Simple Example

Simple Pipeline

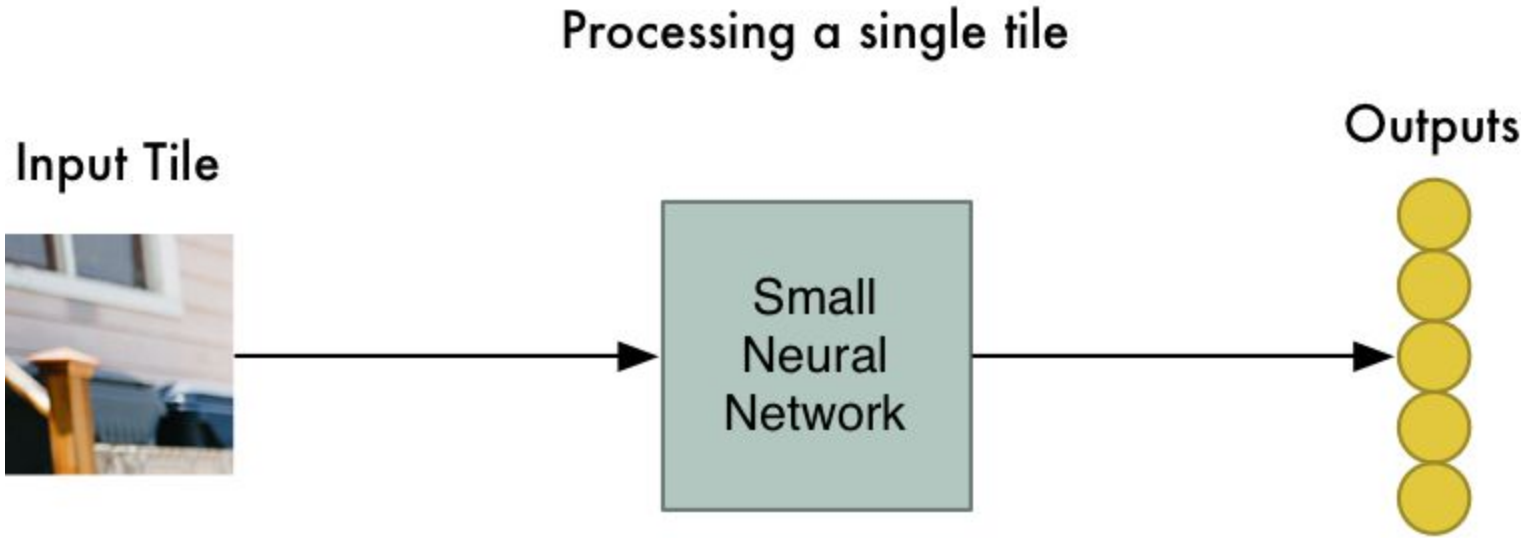


Convolution

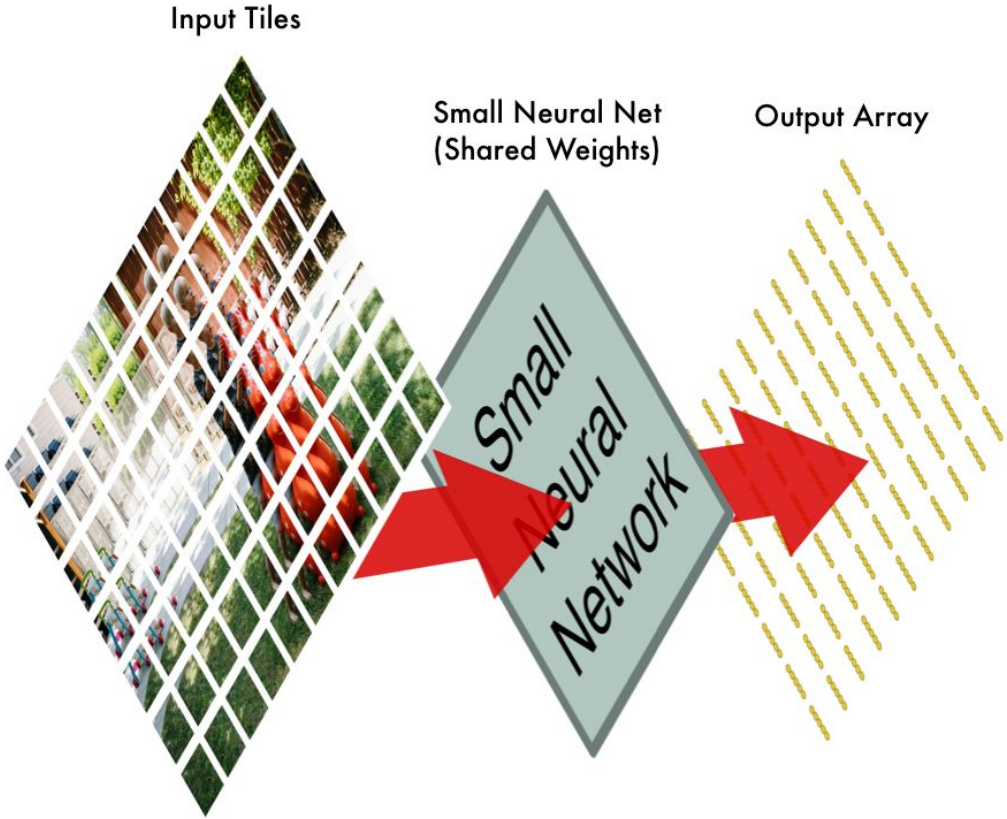
“Sliding Window”



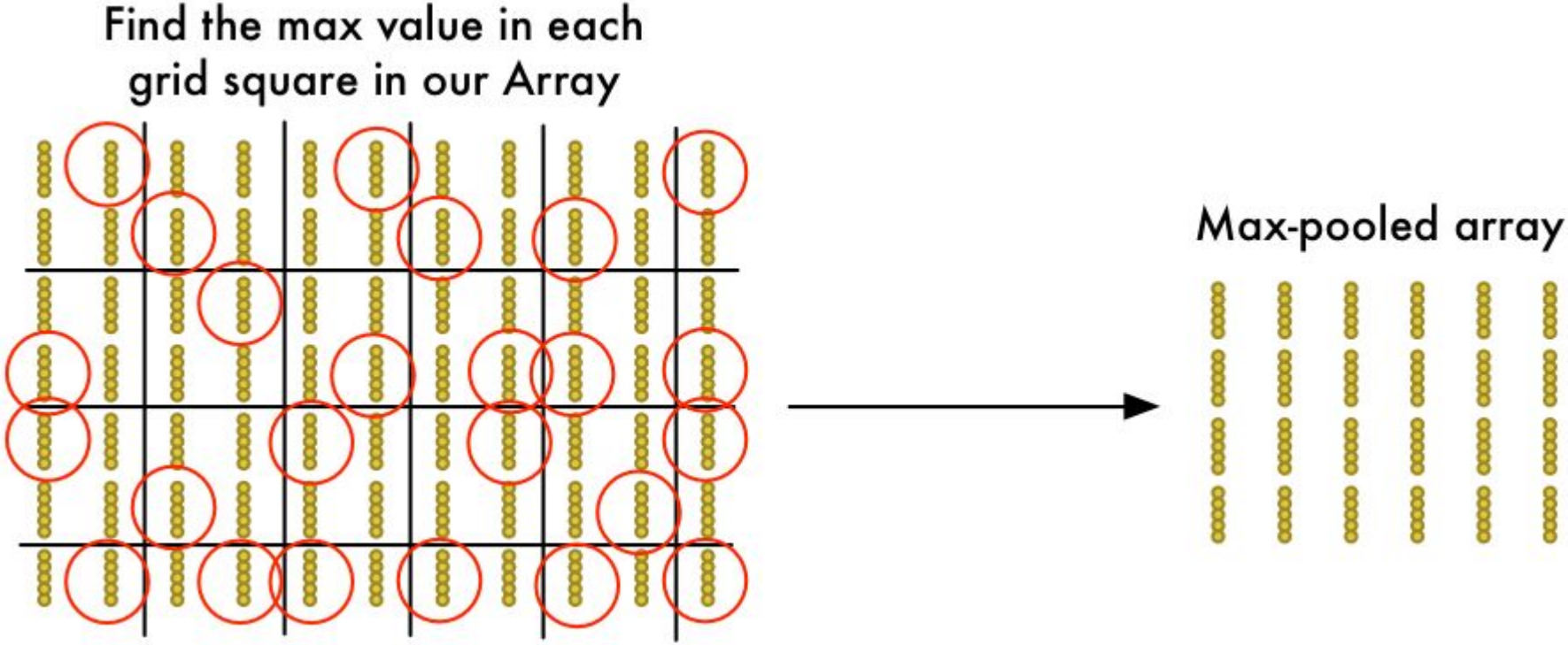
Filter each window using a small neural network



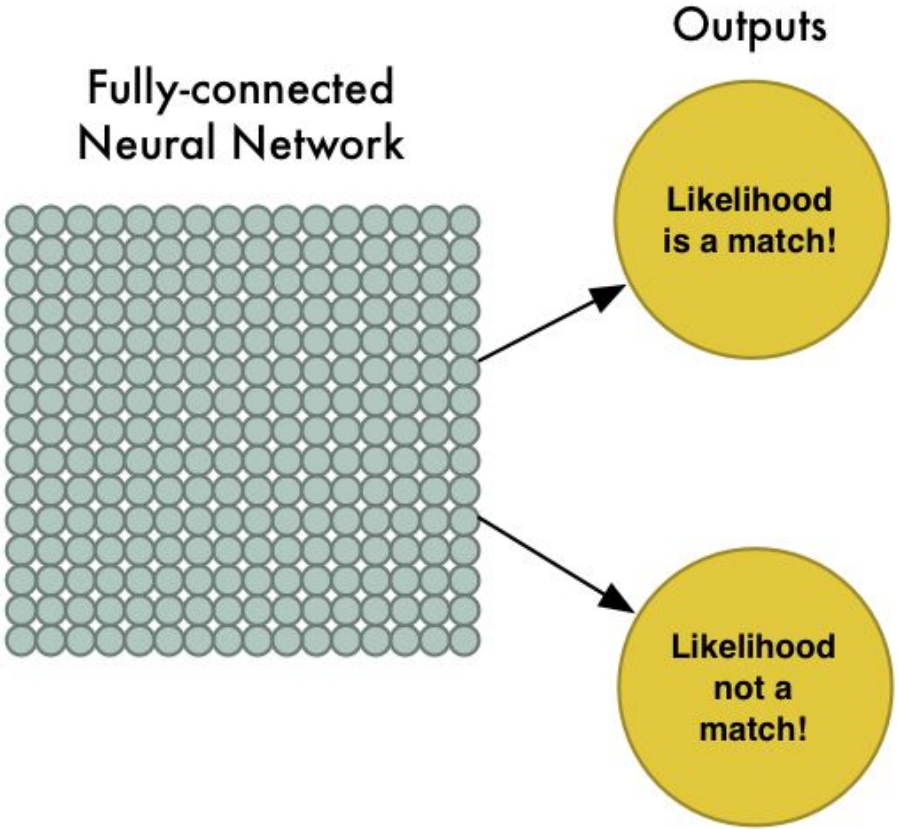
Every window uses the same neural network as a filter



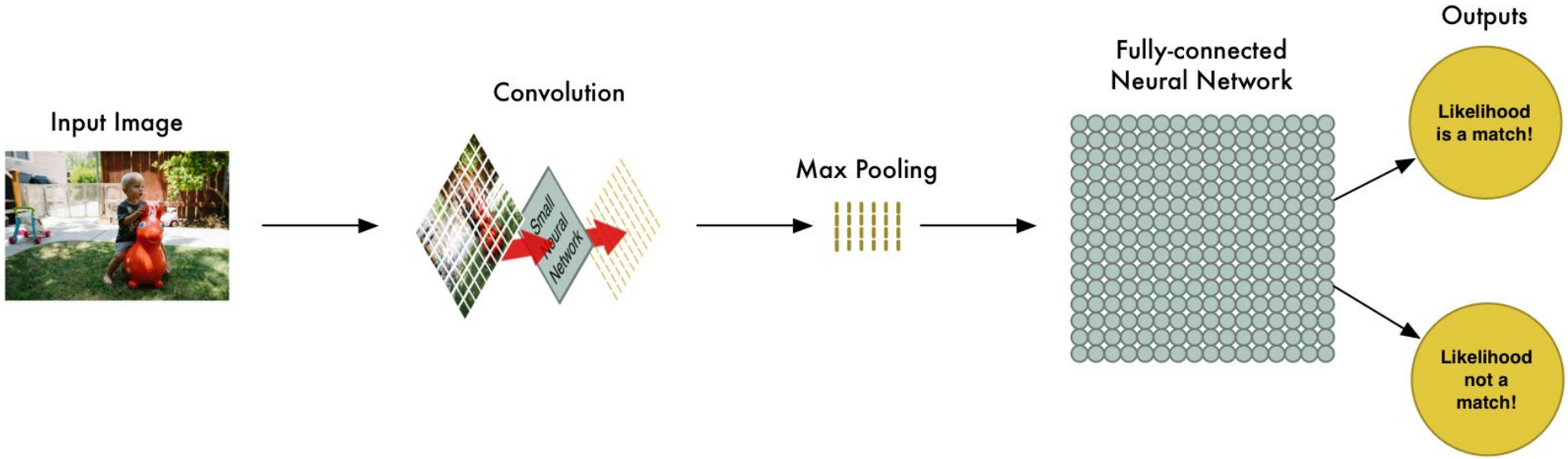
Max Pooling


Find the max value in each grid square in our Array
Max-pooled array

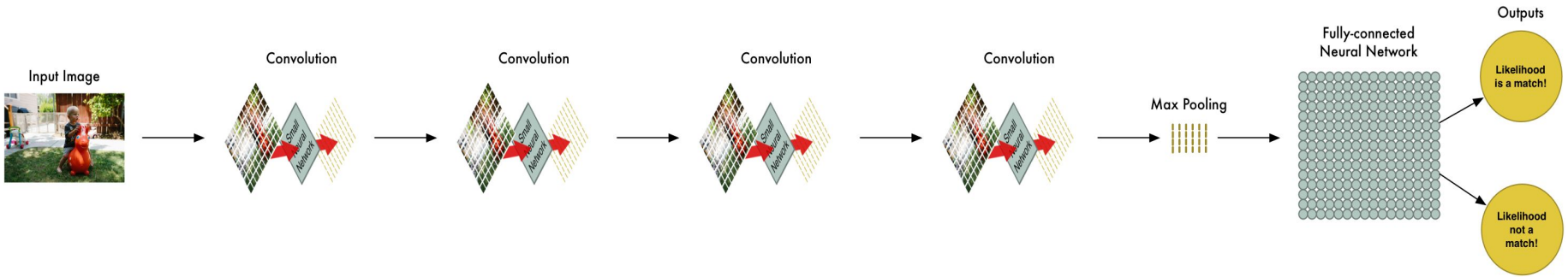
Fully-Connected Layer



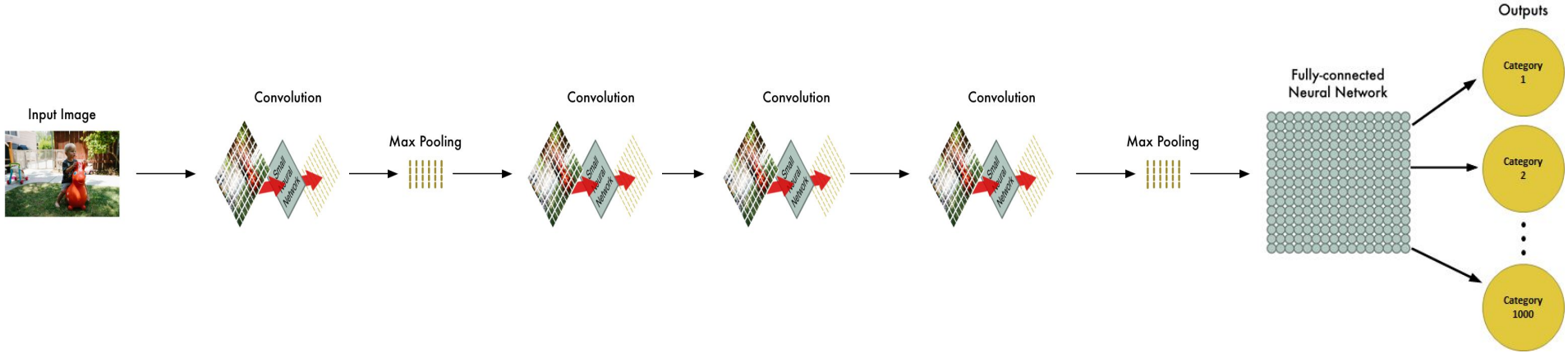
Basic workflow



Can repeat layers as many times as needed



Can repeat layers as many times as needed!



Why CNN over Fully Connected Neural Nets

- Fewer parameters → faster training and less noise
  - Ex. Image of size 200x200x3 (32pixels by 32 pixels with 3 color channels)
    - Fully Connected: 200*200*3 = 120,000 parameters
    - CNN (with **150x150x3** window): **150*150*3 = 67,500 parameters**

Graphic

Number of parameters can go even further down depending on how many pooling layers are used

At some point, the overhead cost of keeping track of convolution and pooling layer parameters make it difficult and time consuming to actually train the CNN.

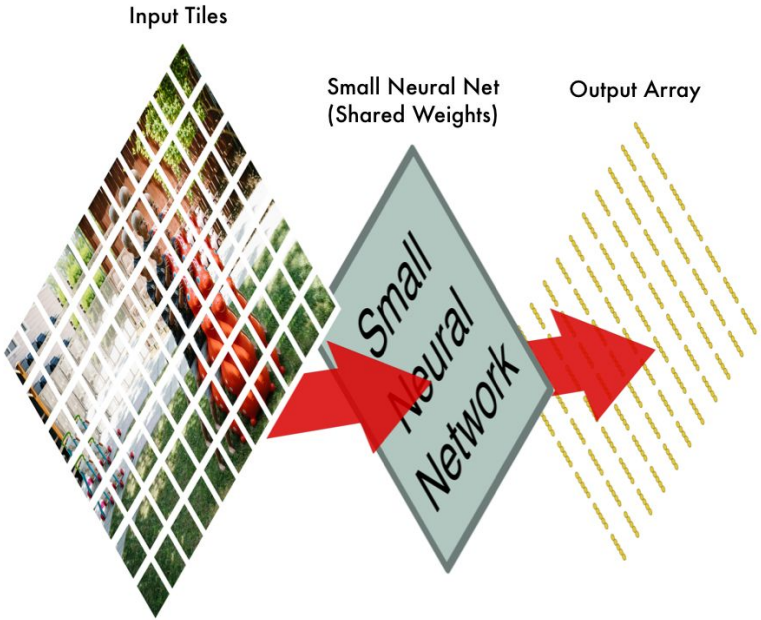
For example: Visual Geometry Group at University of Oxford implemented a 19 layer NN that needed to be trained incrementally starting at 11 layers as weights were assigned. There are een deeper neural nets now but they are not quite CNNs so we wont talk much about them.

Why CNN over Fully Connected Neural Nets

- Fewer parameters → faster training and less noise
- Sliding window → resistant to changes in size, lighting, occlusions, shifts and more

Why CNN over Fully Connected Neural Nets

- Fewer parameters → faster training and less noise
- Sliding window → resistant to changes in size, lighting, occlusions, shifts and more
- Reused coefficients → less memory required

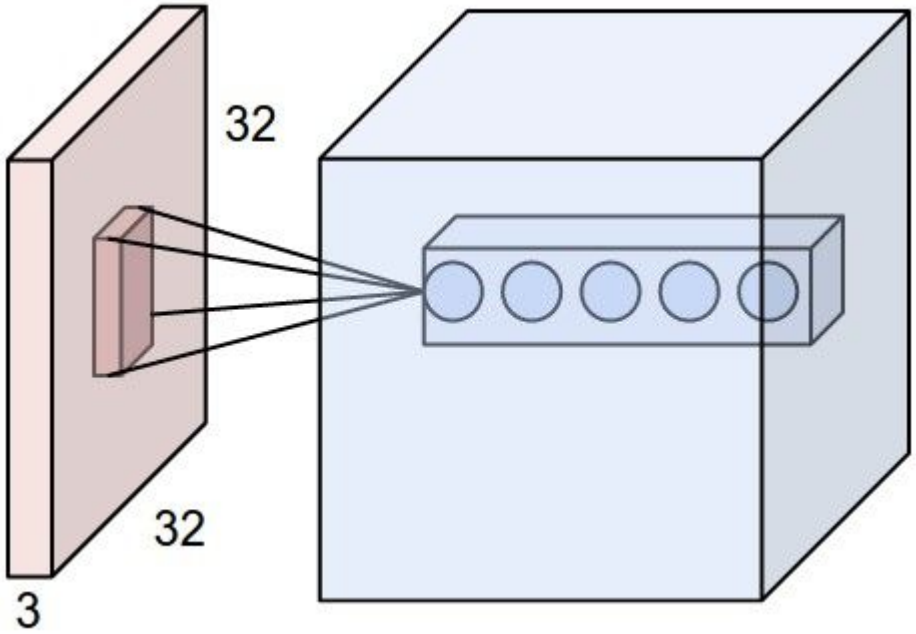


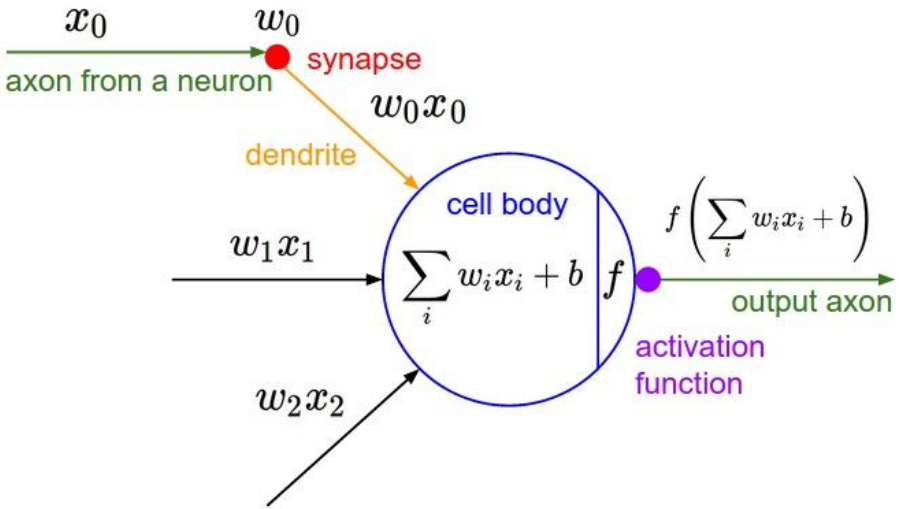
Layers of CNN

Three commonly used layers:

1. Convolution Layers
    a. Non-Linear Layers
2. Pooling/Subsampling Layers
3. Fully Connected Layers

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Blur	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	





Typical Block Diagram of a CNN

Dimensional Analysis

Input Layer: Height x Width x Depth or with CIFAR-10i 32x32x3

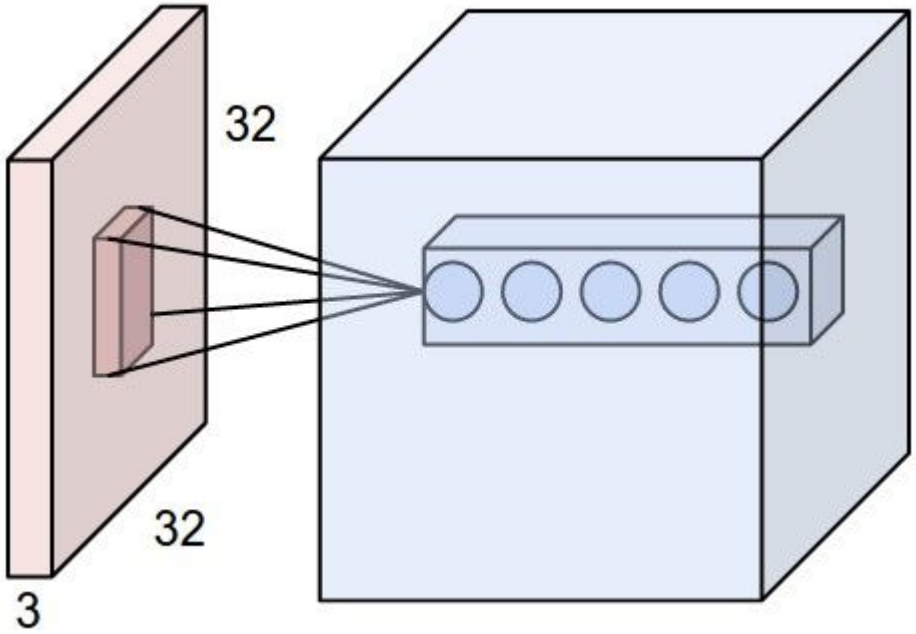
Convolution filter: KxKxD or 5x5x3

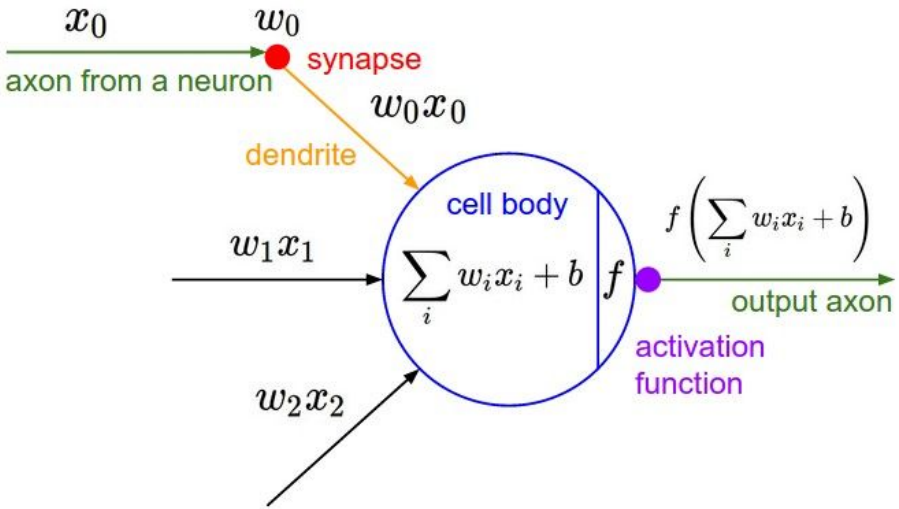
We convolve this filter over the original input to produce a 2 dimensional activation map that is the response to our filter.

Dimensional Analysis

Determining the size of the output of the Convolution Layer depends on:

- The depth
- Stride
- Zero-padding





Typical Block Diagram of a CNN

Convolution Layers

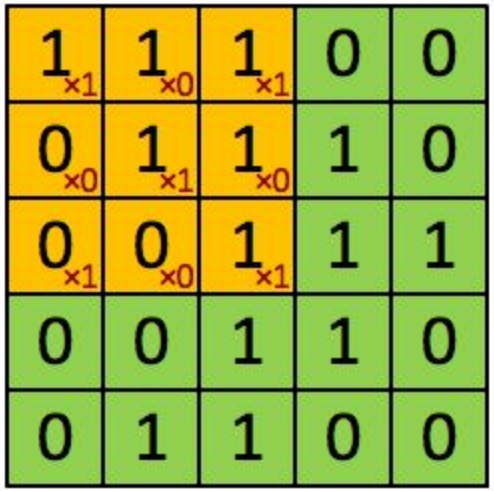
- convolution operation extracts different features of the input
- The first convolution layer extracts low-level features like edges, lines, and corners
- Higher-level layers extract higher-level features

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

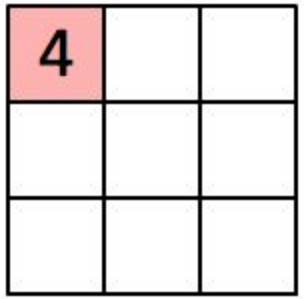
Image

1	0	1
0	1	0
1	0	1

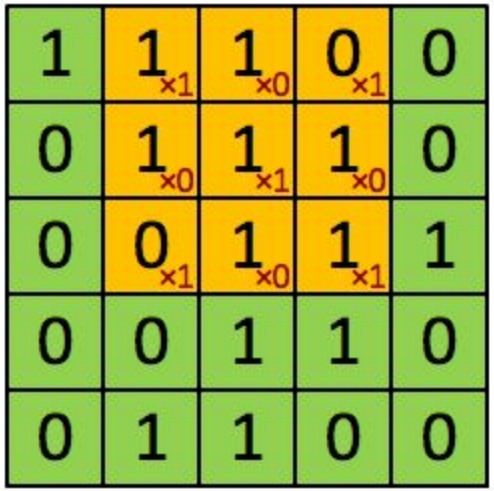
Filter



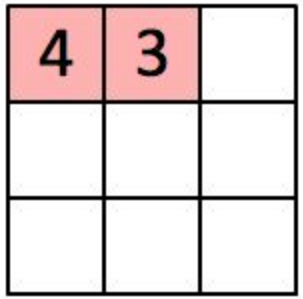
Image



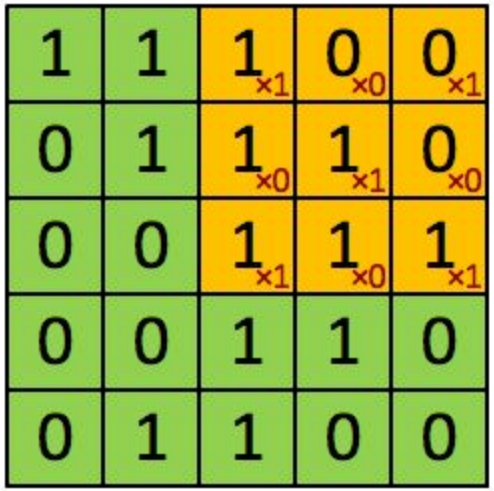
Convolved Feature



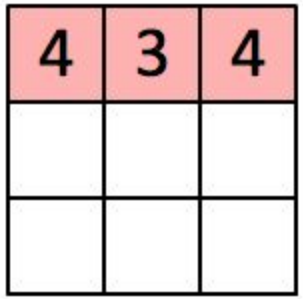
Image



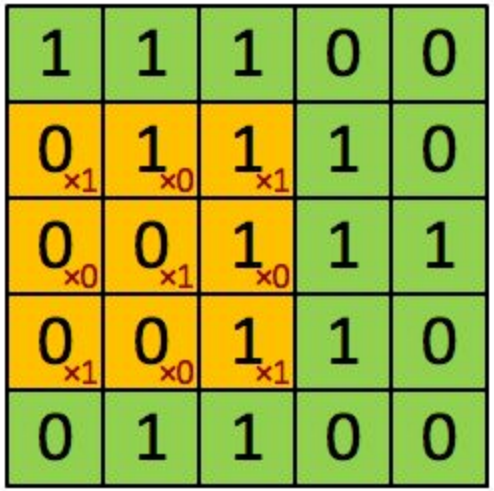
Convolved Feature



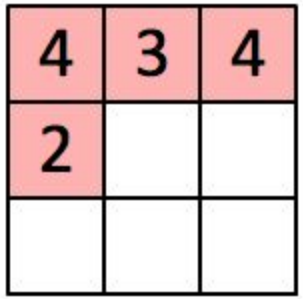
Image



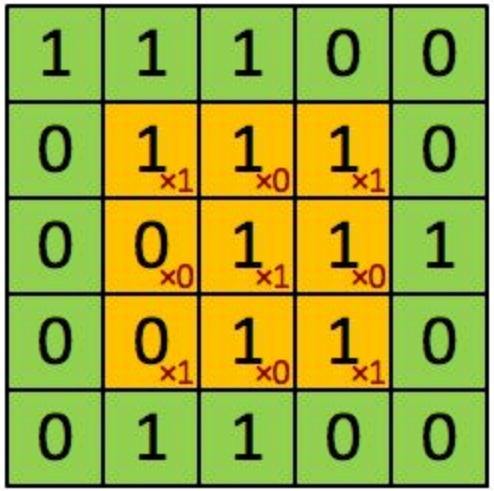
Convolved Feature



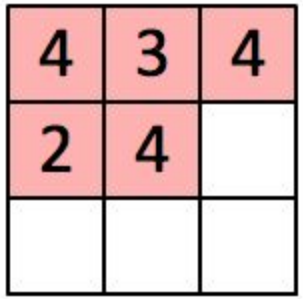
Image



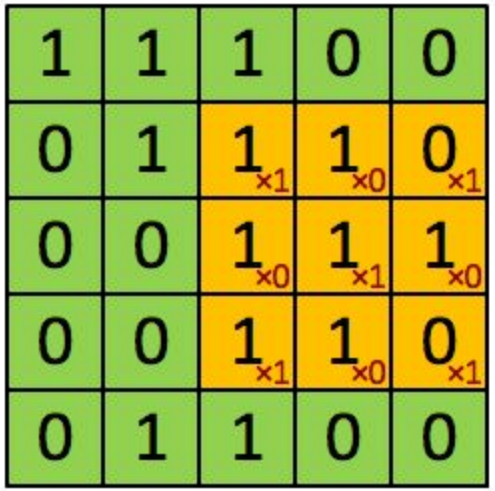
Convolved Feature



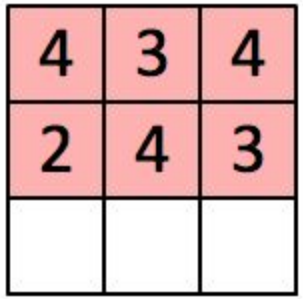
Image



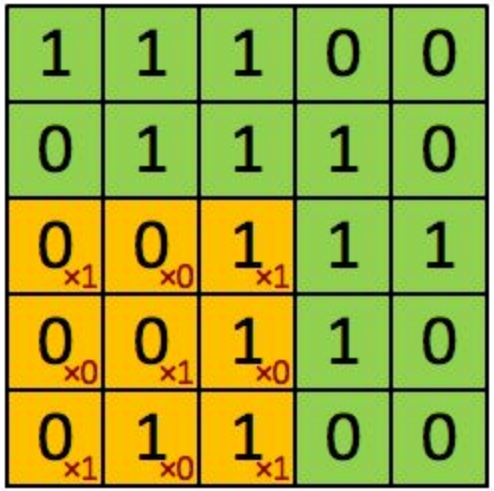
Convolved Feature



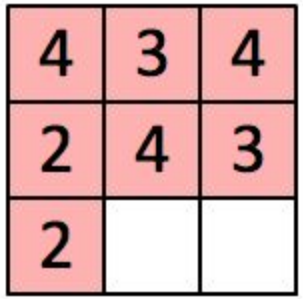
Image



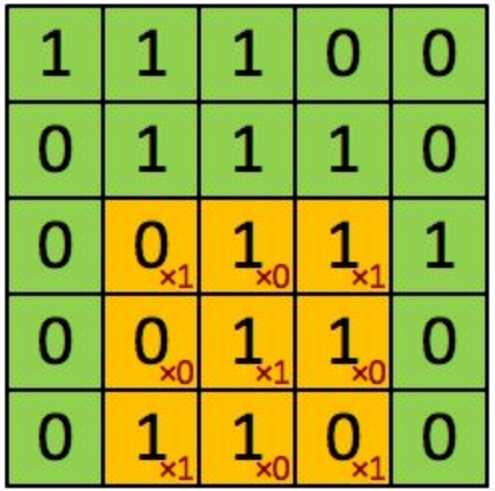
Convolved Feature



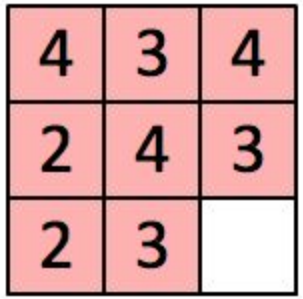
Image



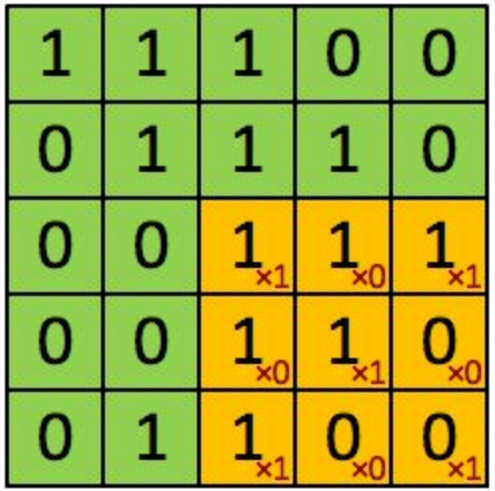
Convolved Feature



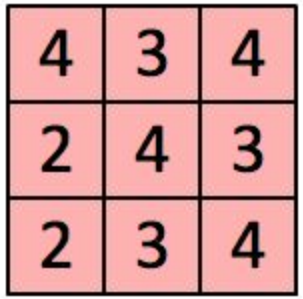
Image



Convolved Feature



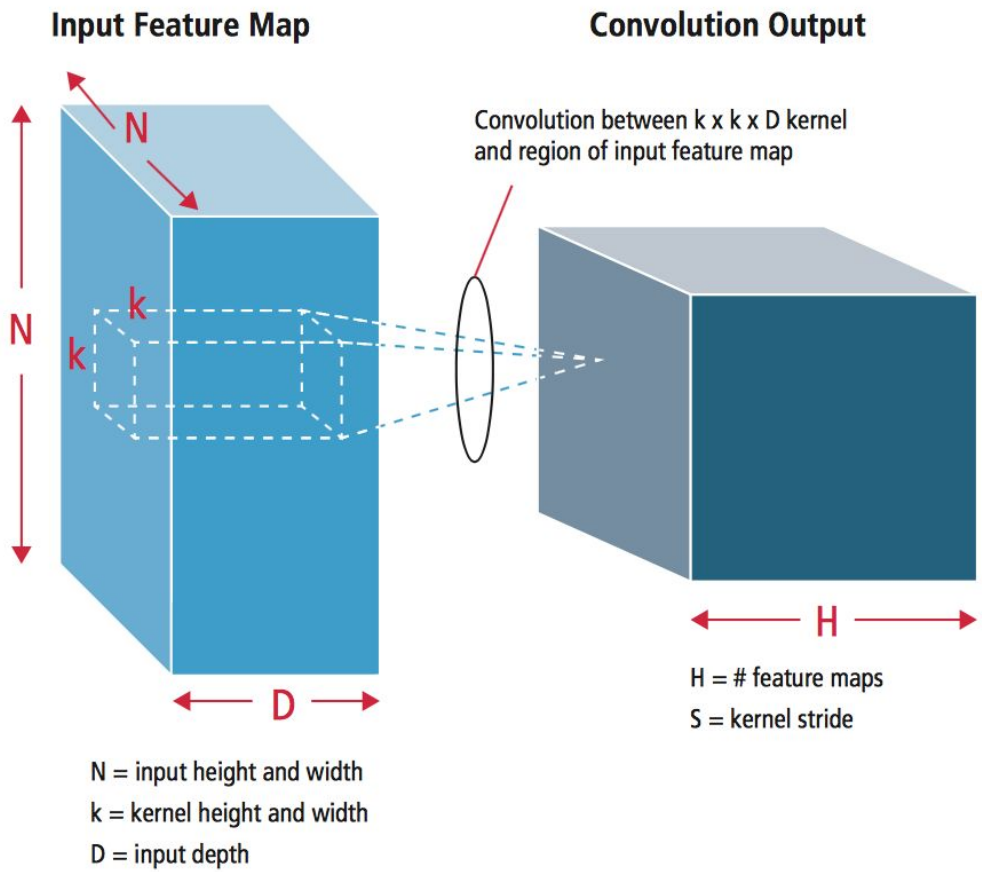
Image



Convolved Feature

Demo

https://cs231n.github.io/assets/conv-demo/index.html

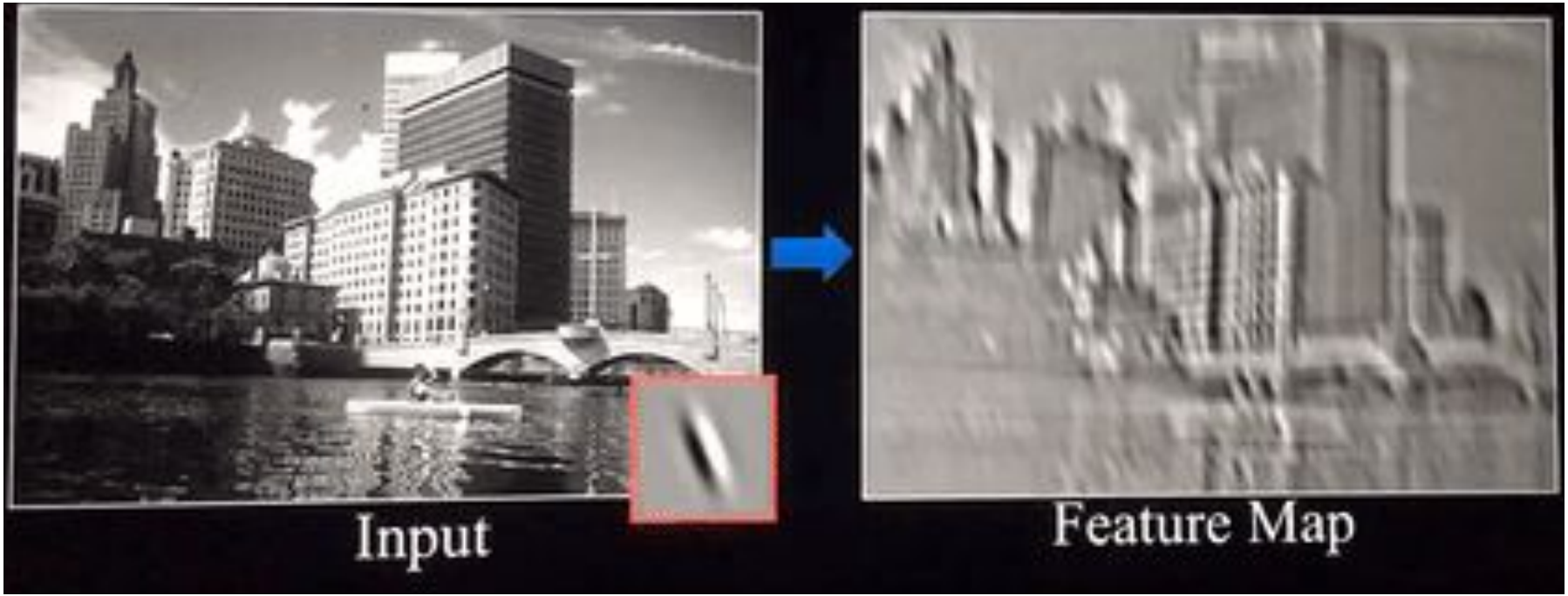

Input Feature Map
Convolution Output
N
N
k
k
D
Convolution between k x k x D kernel
and region of input feature map
H
H = # feature maps
S = kernel stride
N = input height and width
k = kernel height and width
D = input depth

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Blur	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Blur	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	


Input
Feature Map

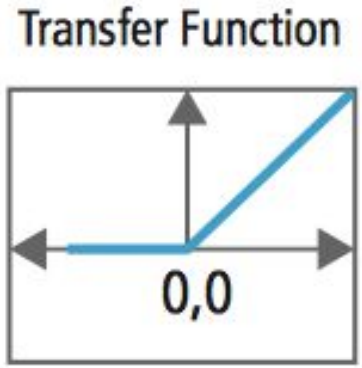
Non-Linear Layers

- Nonlinear “trigger” function
- Functions used:
  - Rectified Linear Units (ReLUs)
  - Continuous Trigger (non-linear) Functions

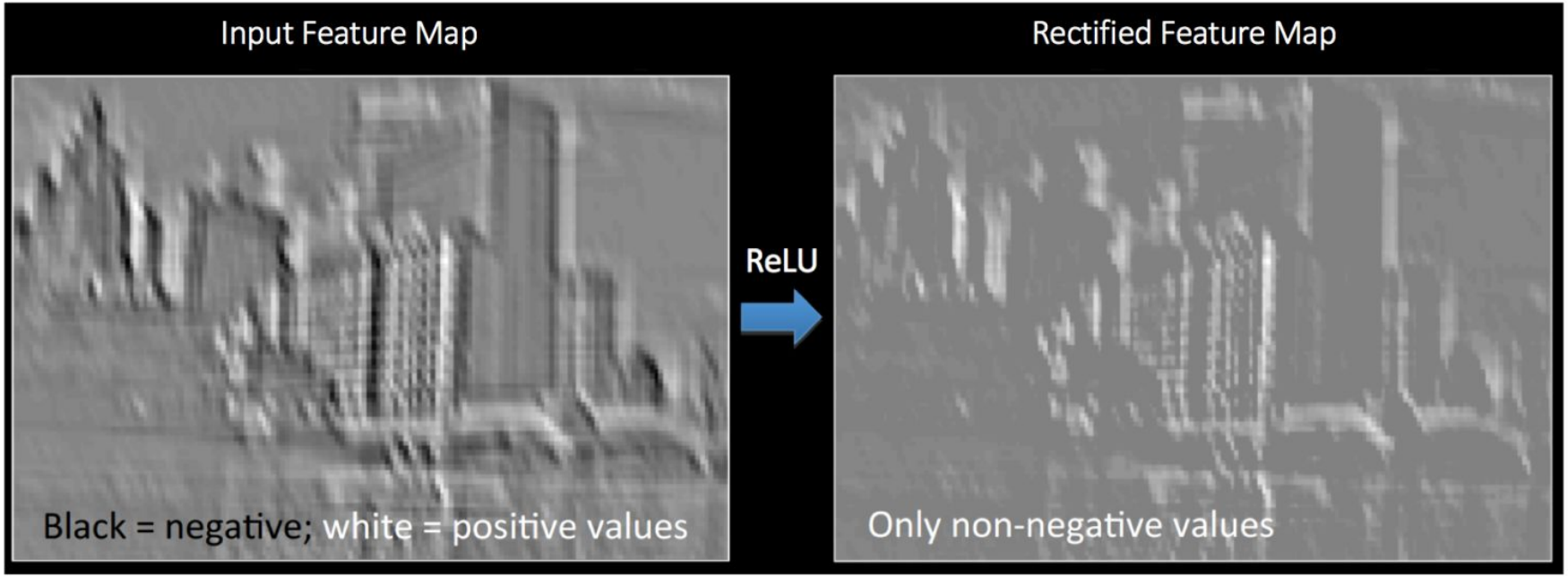
ReLU

$$y = \max(0, x)$$

15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23



15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

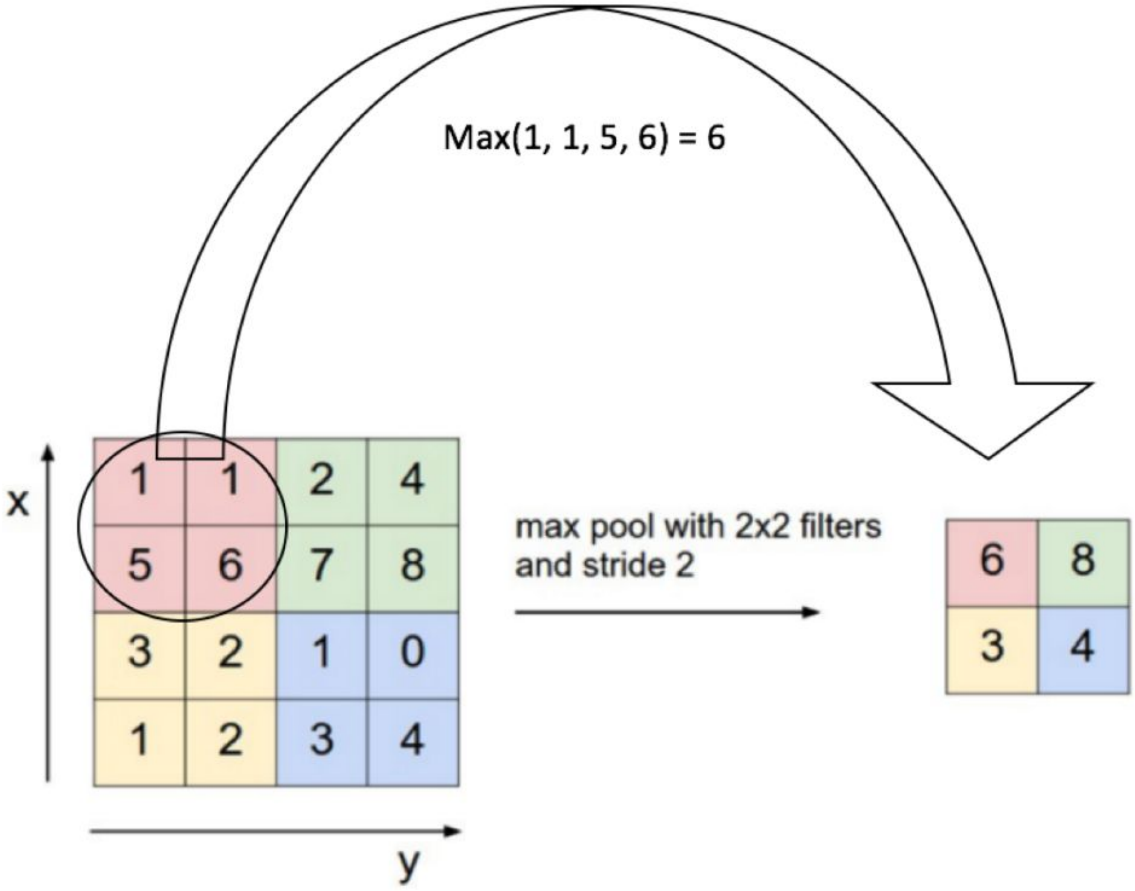

Input Feature Map
Rectified Feature Map
ReLU
Black = negative; white = positive values
Only non-negative values

Pooling/Subsampling Layers

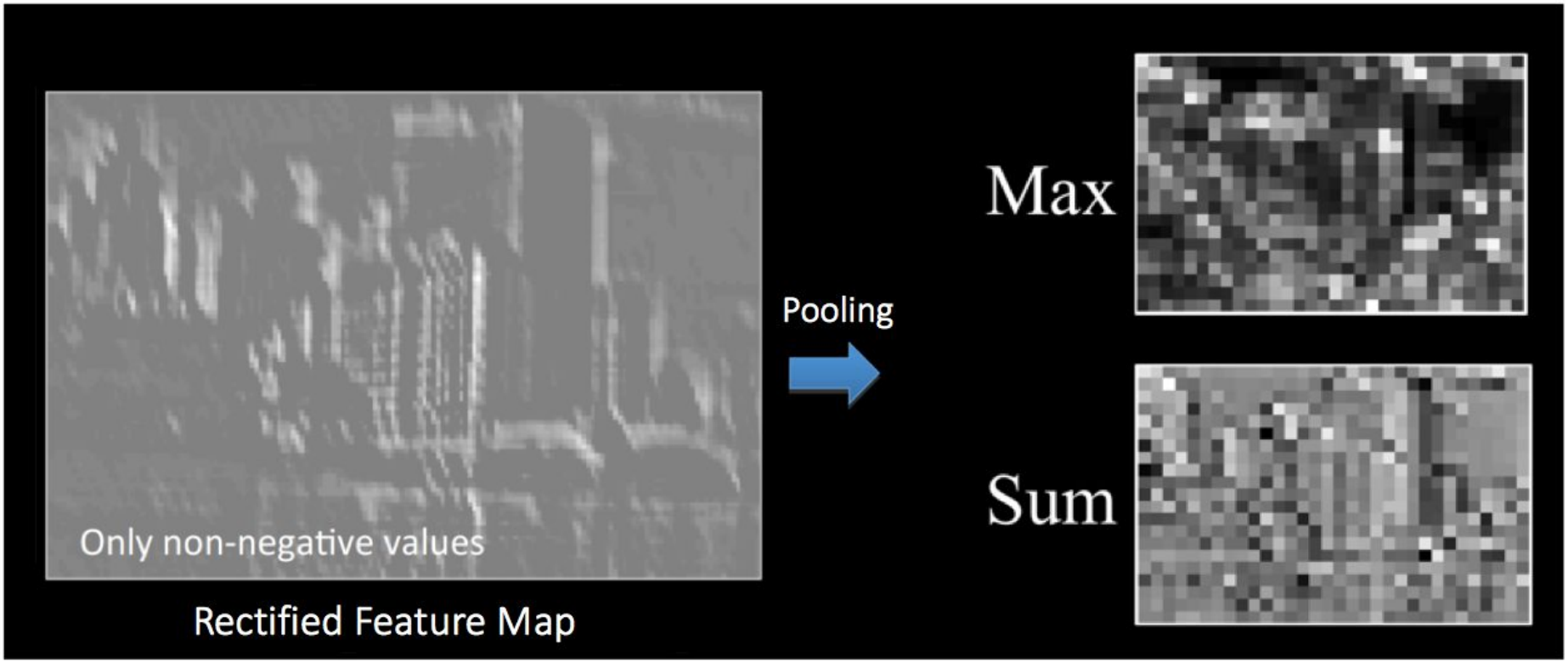
- Reduces resolution of features
- Makes features robust against noise and distortion
- Controls overfitting by reducing number of parameters and computations
- Scale invariant representation of image

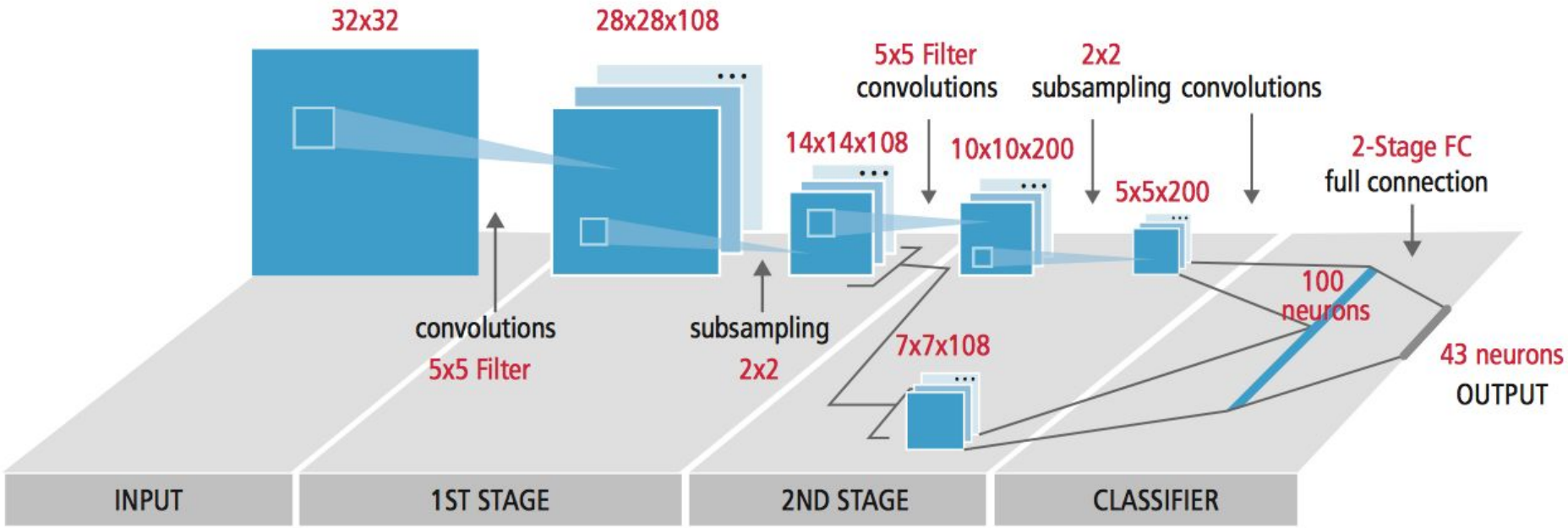
Common Methods of Pooling

- Max Pooling
- Average Pooling
- Sum Pooling



Rectified Feature Map


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum



Typical Block Diagram of a CNN

Fully-Connected Layers

- Final layer of CNN
- Multi Layer Perceptron
- Output from previous layers are high-level features of input image
- Uses weighted sum of these features for classification

What’s next?

Transfer Learning -- reusing trained weights

Scene segmentation -- Training for each individual pixel to learn