



# Advanced NN Architectures: CNNs

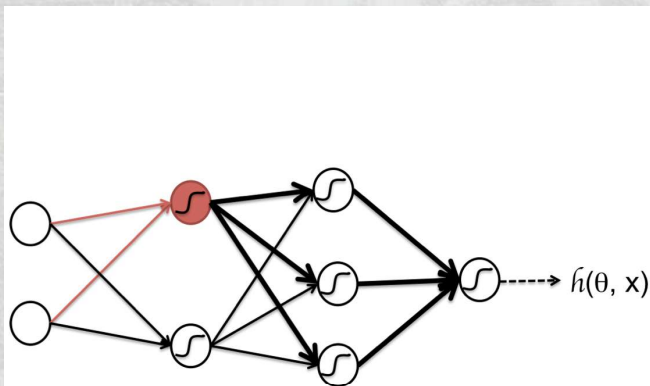
Roberto Basili, Danilo Croce  
Machine Learning, Deep Learning 2024/2025

# Outline

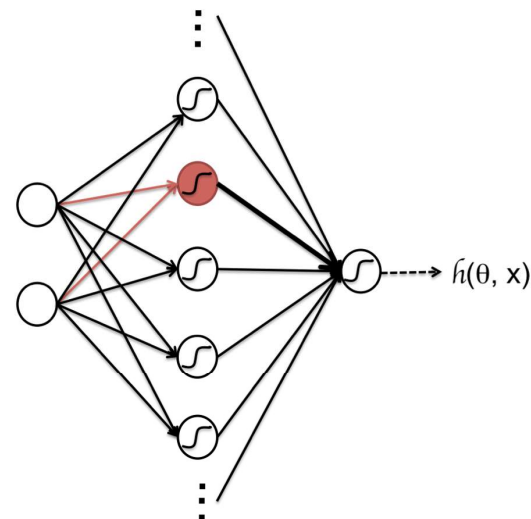
- Architectures and tasks
- Convolutional Neural Networks
  - Filters and Convolutions
  - Pooling
- Imagenet
- Applications of NNs:
  - Image processing: classification, Object Recognition
  - Text Classification:
    - Convolutional NNs over texts
    - Sentiment analysis
    - The Movie Review Dataset

# Deep vs Shallow Networks

- Deep networks should be preferred to Shallow ones
  - when problems are non-linear;
  - it has been observed that a shallow network needs about 10x number of neurons for reaching the expressivity of a deep one



Deep Network

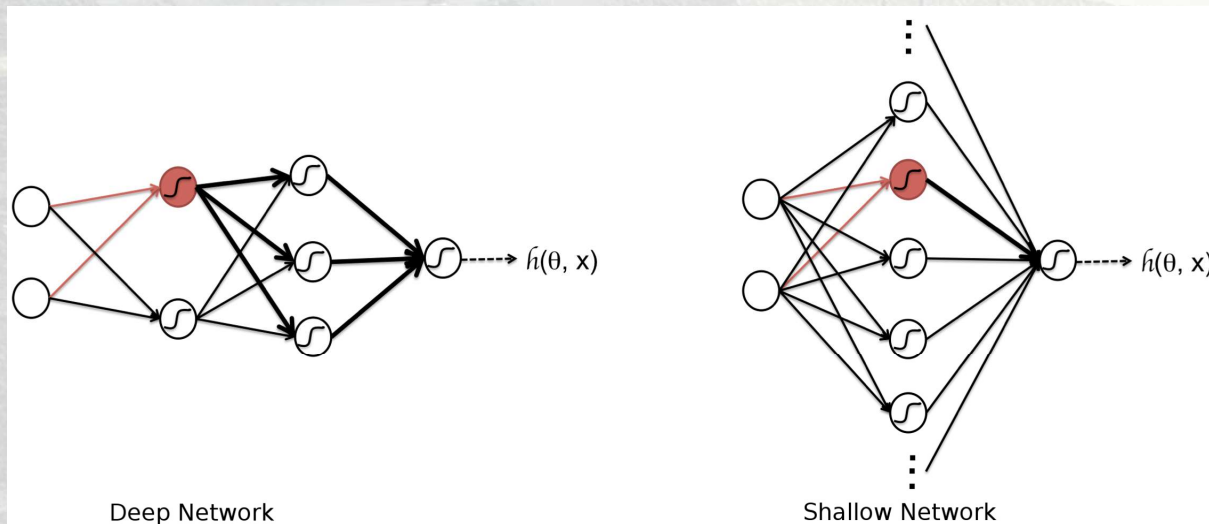


Shallow Network



# Deep vs Shallow Networks: Intuition

- Think of a neuron as a program routine
  - in Deep Networks a neuron computation is re-used many times in the computation
  - in a Shallow Network it is used only once
- Using a shallow network is similar to writing a program without the ability of calling subroutines



# Deep Networks vs. Kernel

- A kernel machine can be thought of as a shallow network having a huge hidden layer
  - this hidden layer is never computed thanks to the kernel trick
- Kernel methods however are expensive
  - they rely on a set of examples, support vectors
  - for large dataset and complex problems this set can be large as well
- Neural networks computation
  - is independent on the dataset,
  - but only on the number of connections that have been chosen



# NN architectures

- Multilayer perceptron (Rumelhart MCClelland, 1980)
- Self-Organizing maps (Kohonen, 1990)
- Boltzman Machines (Hinton, 1998)
- Convolutional Neural Networks (Neocogitron, Fukushima (1980))
- Recurrent Neural Networks (Jordan, 1986), (Elman, 1990)
  - Bidirectional RNNs (Schuster and Paliwal, 1997)
  - BP Through-Time (Robinson & Fallside, 1987)
  - Long Short Time Memories LSTMS, (Hochreiter & Schmidhuber, 1997)
  - Attention mechanisms (firstly discussed by (Larochelle & Hinton, 2010; Denil et al., 2012)).
- Autoencoders (Bengio et al., 2007), Encoder-Decoders (Cho et al., 2015)

# Recent successes in Deep Learning

- Convolution Neural Networks
  - Images related tasks
- Recurrent Neural Networks
  - Language models
  - Speech to Text
  - Machine Translation, Conversation Models
- Attentional Networks
  - Attention mechanisms to amplify dependencies across network components
- Trasformers:
  - Encoding-decoding networks for powerful pretraining
  - Avoid the forgetting problems typical of recurrent architectures
- Large Language Models and Prompting
  - Encoding-Decoding at the Natural Language level
  - Decoding only transformers
  - 0-shot or few-shot Learning
- Advanced architectures: multimodality
  - Image to Captions
  - Text-generated Images (Dall-E)



# Outline

- Architectures and tasks



- Convolutional Neural Networks

- Filters and Convolutions
- Pooling

- Imagenet

- Applications of NNs:

- Image processing: classification, Object Recognition
- Text Classification:
  - Convolutional NNs over texts
  - Sentiment analysis
  - The Movie Review Dataset



# Convolutional Neural Networks

(Le Cun, 1998)

- Mainly used for images related tasks
  - image classification
  - face detection
  - etc...
- **Learn feature representations**
  - by **convolving** over the input
  - with a **filter**, that slides over the input image
- **Compositionality** (local)
  - Each filter composes a local patch of lower-level features into a higher-level representation
- **Location Invariance**
  - the detection of specific patterns is independent of where it occurs

1	0	1
0	1	0
1	0	1

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature

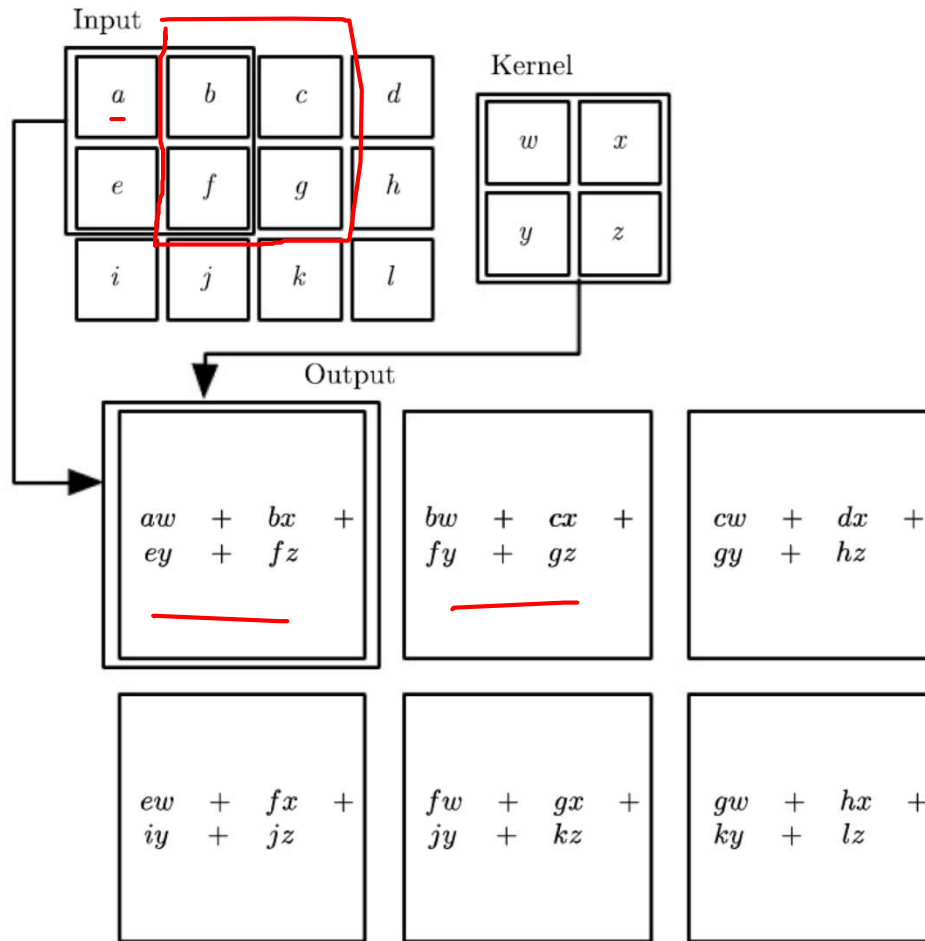
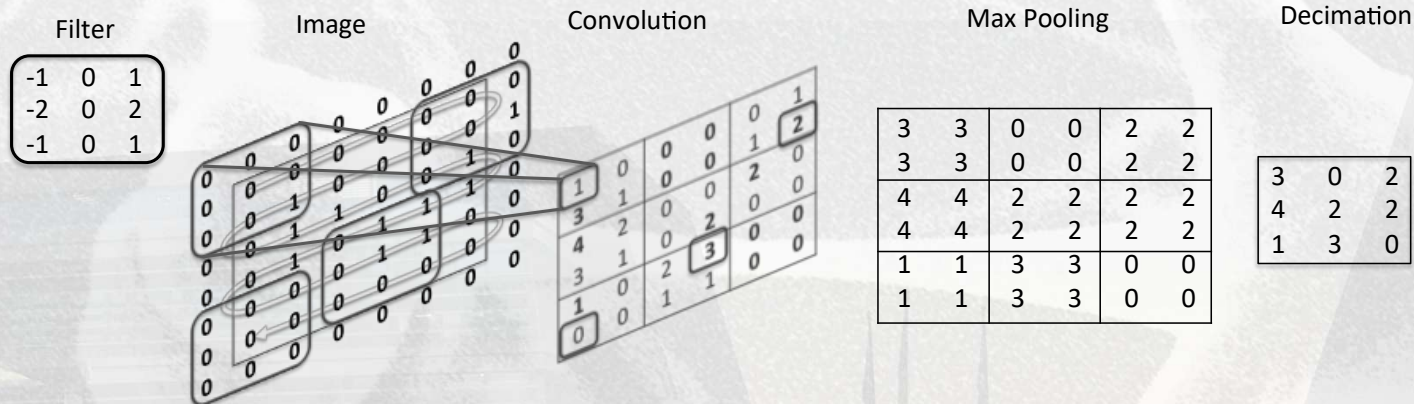


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called “valid” convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

# A further example of: convolution with pooling, and decimation operations



- An image is convolved with a filter; curved rectangular regions in the first large matrix depict a random set of image locations
- Maximum values within small 2x2 regions are indicated in bold in the central matrix
- The results are pooled, using max-pooling then decimated by a factor of two, to yield the final matrix



# Simple filtering example

- Ex. consider the task of detecting edges in an image
- A well known technique is to filter an image with so-called “Sobel” filters, which involves convolving it with

$$\mathbf{W}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathbf{W}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

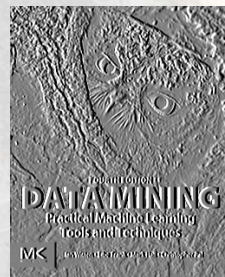
- Applied to the image X below, we have:



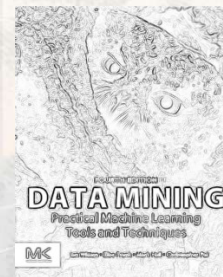
X



$$\mathbf{G}_x = \mathbf{W}_x * \mathbf{X}$$



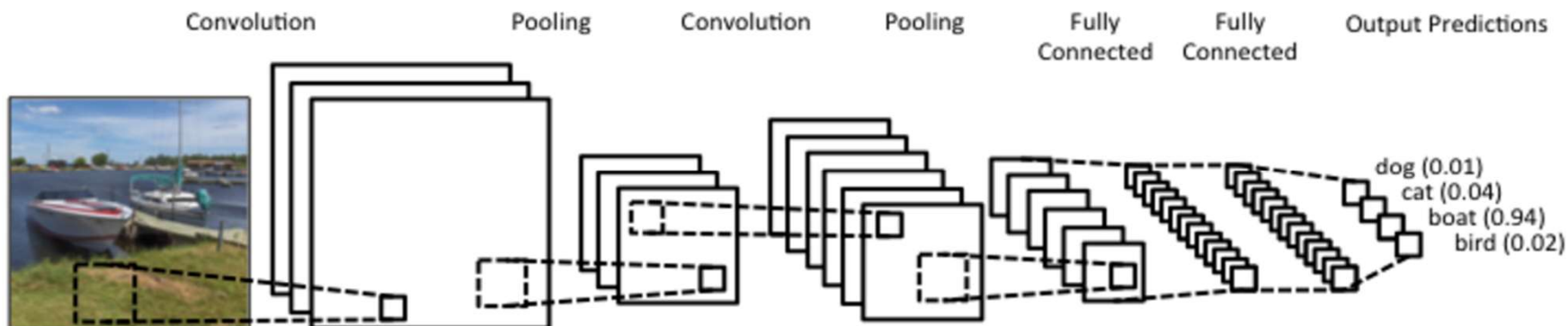
$$\mathbf{G}_y = \mathbf{W}_y * \mathbf{X}$$



$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

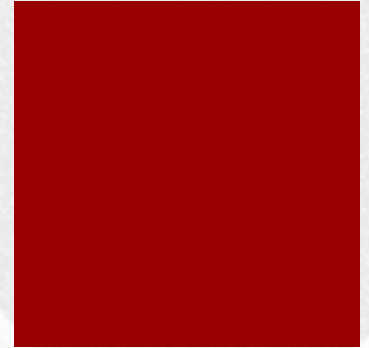
# Convolutional Neural Networks

- **CNNs automatically learn the parameters of the filters**
  - a filter is a matrix of parameters
  - the key aspect is that a filter is adopted for the whole image
- Convolution can be applied in **multiple** layers
  - a layer  $l+1$  is computed by convolving over output produced in layer  $l$
  - Pooling is an operation often adopted for taking the most informative features that are learned after a convolution step





# Pooling and subsampling layers



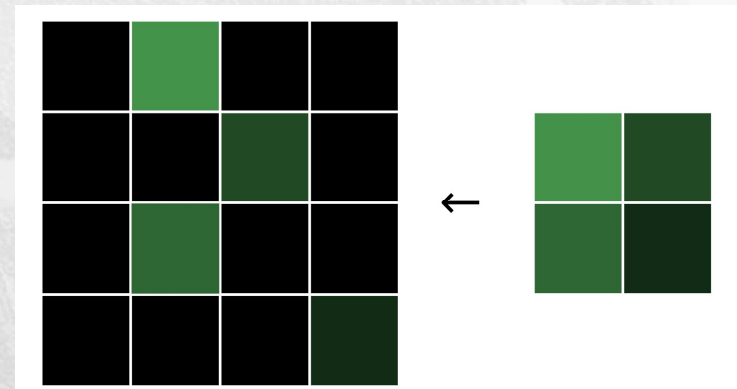
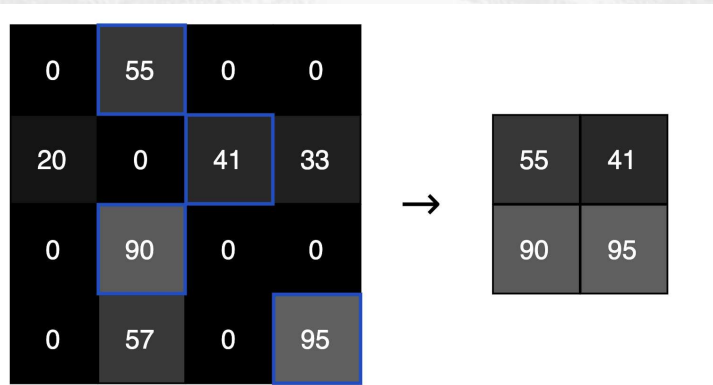
- What are the consequences of backpropagating gradients through max or average pooling layers?
- **Max pooling**: the units that are responsible for the maximum within each zone  $j, k$  —the “winning units”— are the only to get the *backpropagated gradient*
- **Average pooling**: the averaging is simply a special type of convolution with a fixed kernel that computes the (possibly weighted) average of pixels in a zone
  - the required gradients are therefore like std conv. layers
- The subsampling step either samples every  $n^{\text{th}}$  output, or avoids needless computation by only evaluating every  $n^{\text{th}}$  pooling computation



# Training in CNN:

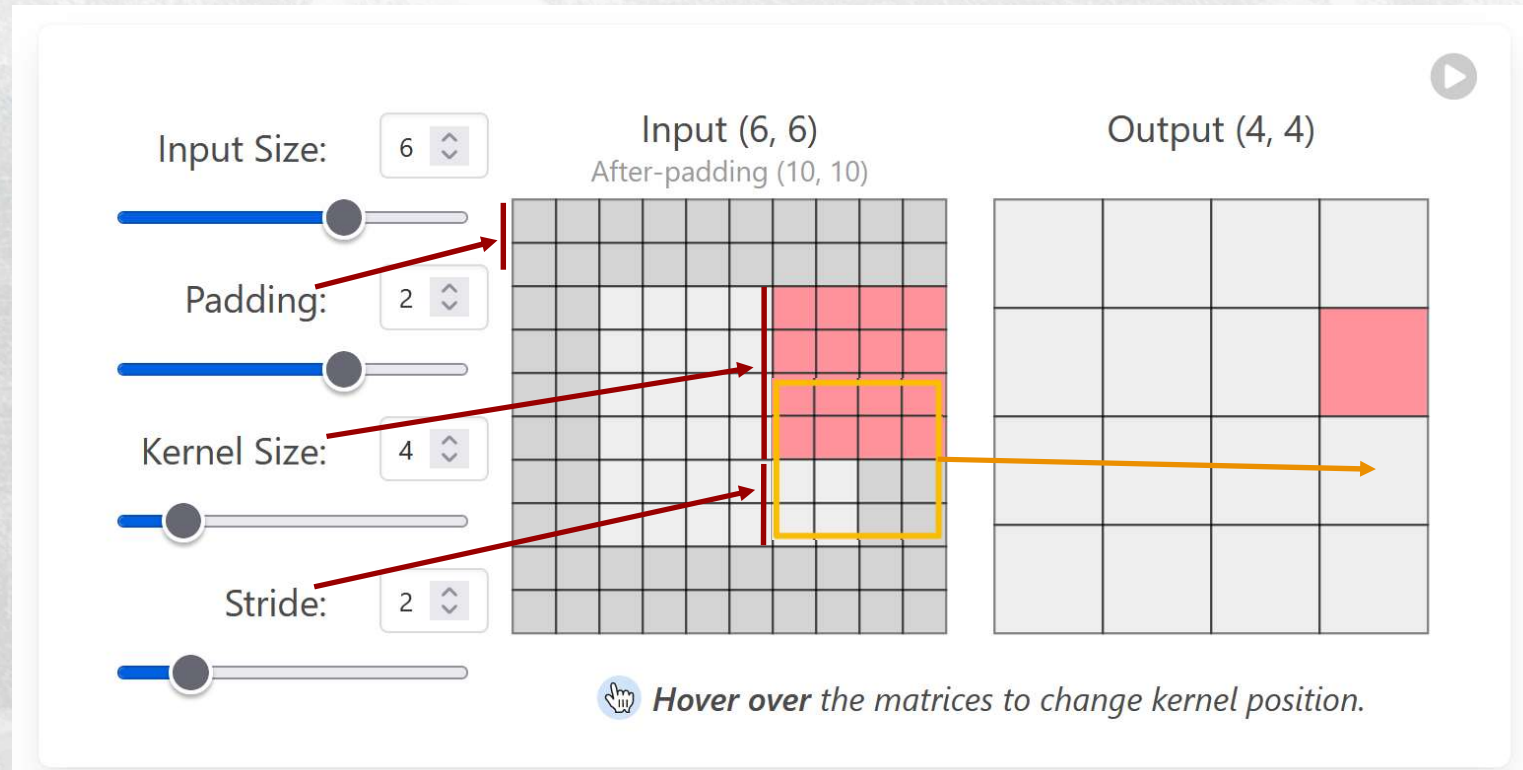
## Backpropagation and Max Pooling

- A Max Pooling layer can't be trained because it doesn't actually have any weights
- It still supports a method for it to calculate gradients



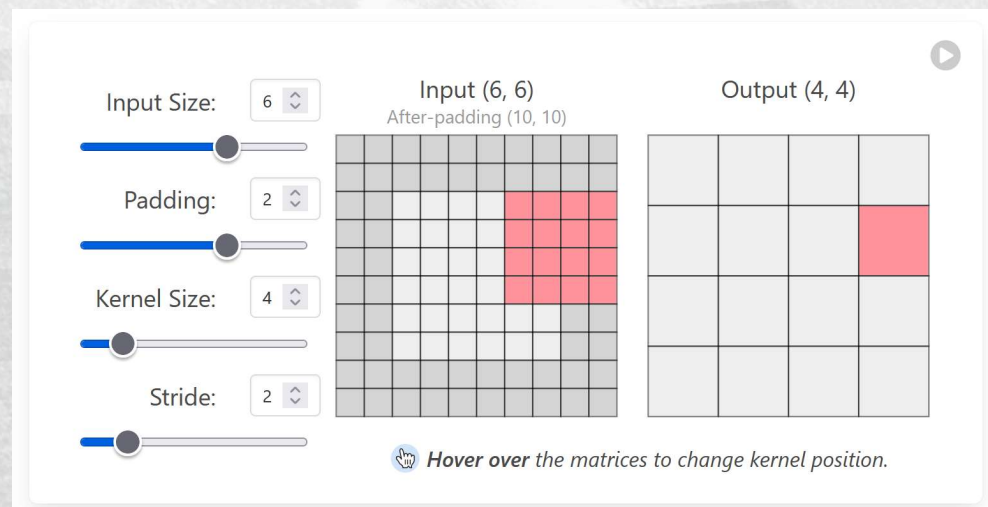
- How is  $\partial L / \partial \text{inputs}$  ?
  - An input pixel that isn't the max value in its 2x2 block have zero marginal effect on the loss, as any slightly change of its value wouldn't change the output at all!
    - $\partial L / \partial \text{inputs} = 0$  for any non-max pixels.
  - On the other hand, an input pixel that is the max value would have its value passed through to the output, so  $\partial \text{output} / \partial \text{input} = 1$ , meaning  $\partial L / \partial \text{input} = \partial L / \partial \text{output}$ .

# Training a CNN: terminology



# Dimensions

- The dimension of the output is the following



$$O = \frac{InputD - KernelD + 2PaddingD}{StrideD} + 1$$

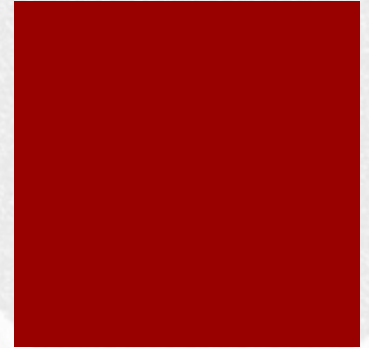
$$\text{In the example: } O = \frac{InputD - KernelD + 2PaddingD}{StrideD} + 1 = \frac{6 - 4 + 2 \cdot 2}{2} + 1 = 3 + 1 = 4$$



# Outline

- Architectures and tasks
- Convolutional Neural Networks
  - Filters and Convolutions
  - Pooling
- ➔ ■ Imagenet
- Applications of NNs:
  - Image processing: classification, Object Recognition
  - Text Classification:
    - Convolutional NNs over texts
    - Sentiment analysis
    - The Movie Review Dataset

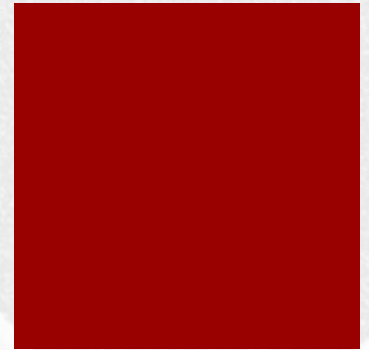
# The ImageNet challenge



- Crucial in demonstrating the effectiveness of deep CNNs
- Problem: recognize object categories in Internet imagery
- The 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) classification task - classify image from Flickr and other search engines into 1 of 1000 possible object categories
- Serves as a standard benchmark for deep learning
- The imagery was hand-labeled based on the presence or absence of an object belonging to these categories
- There are 1.2 million images in the training set with 732-1300 training images available per class
- A random subset of 50,000 images was used as the validation set, and 100,000 images were used for the test set where there are 50 and 100 images per class respectively



# ImageNet Home Page



## IMGENET Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)

[Introduction](#) [News](#) [History](#) [Timetable](#) [Challenges](#) [FAQ](#) [Citation](#) [Contact](#)

### Introduction

This challenge evaluates algorithms for object localization/detection from images/videos at scale. Most successful and innovative teams will be invited to present at **CVPR 2017 workshop**.

- I. Object localization for 1000 categories.
- II. Object detection for 200 fully labeled categories.
- III. Object detection from video for 30 fully labeled categories.



# Goal

## ImageNet

## ILSVRC



- Over 1
- Rough
- Collect Turk
- Annual competition of image classification at large scale
- 1.2M images in 1K categories
- Classification: make 5 guesses about the image label



EntleBucher



Appenzeller

# Object Location task

- Images, Class labels and Bounding boxes

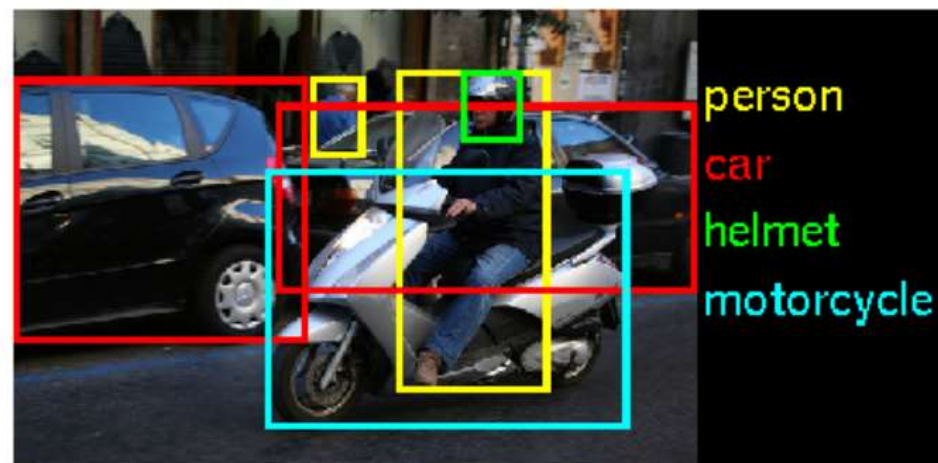
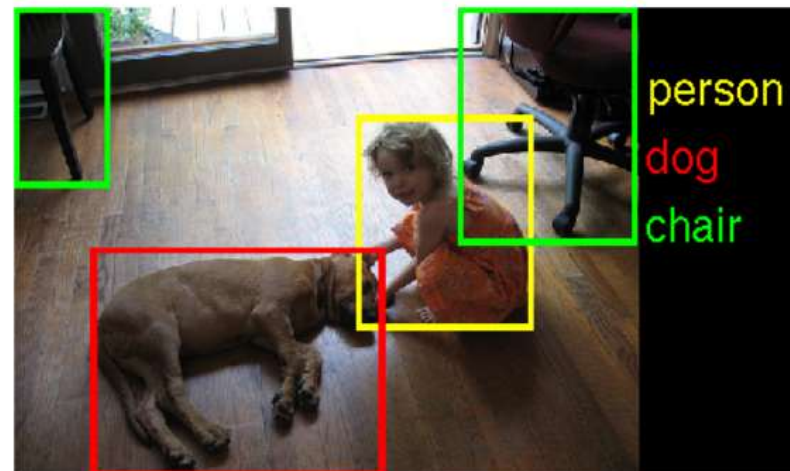
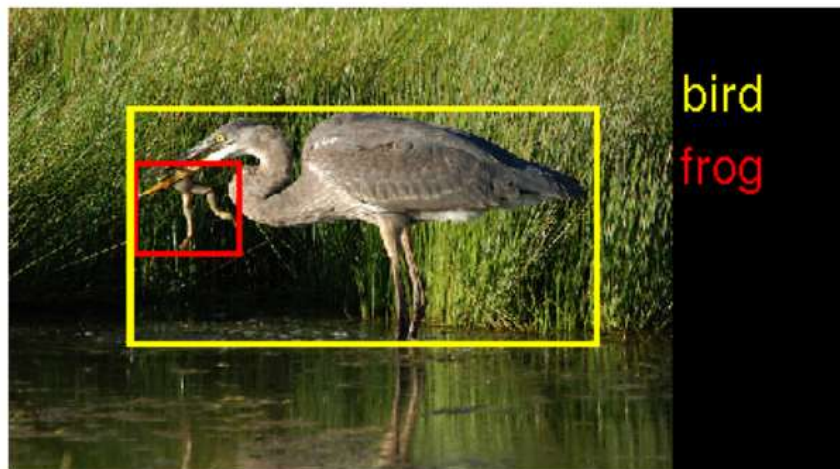
The ground truth labels for the image are  $C_k, k = 1, \dots, n$  with  $n$  class labels. For each ground truth class label  $C_k$ , the ground truth bounding boxes are  $B_{km}, m = 1 \dots M_k$ , where  $M_k$  is the number of instances of the  $k^{\text{th}}$  object in the current image.

Let  $d(c_i, C_k) = 0$  if  $c_i = C_k$  and 1 otherwise. Let  $f(b_i, B_k) = 0$  if  $b_i$  and  $B_k$  have more than 50% overlap, and 1 otherwise. The error of the algorithm on an individual image will be computed using:

$$e = \frac{1}{n} \cdot \sum_k \min_i \min_m \max\{d(c_i, C_k), f(b_i, B_{km})\}$$

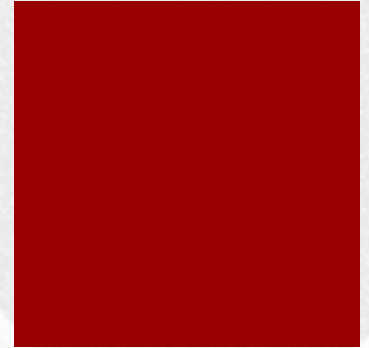


# ILSVRC2014 Examples





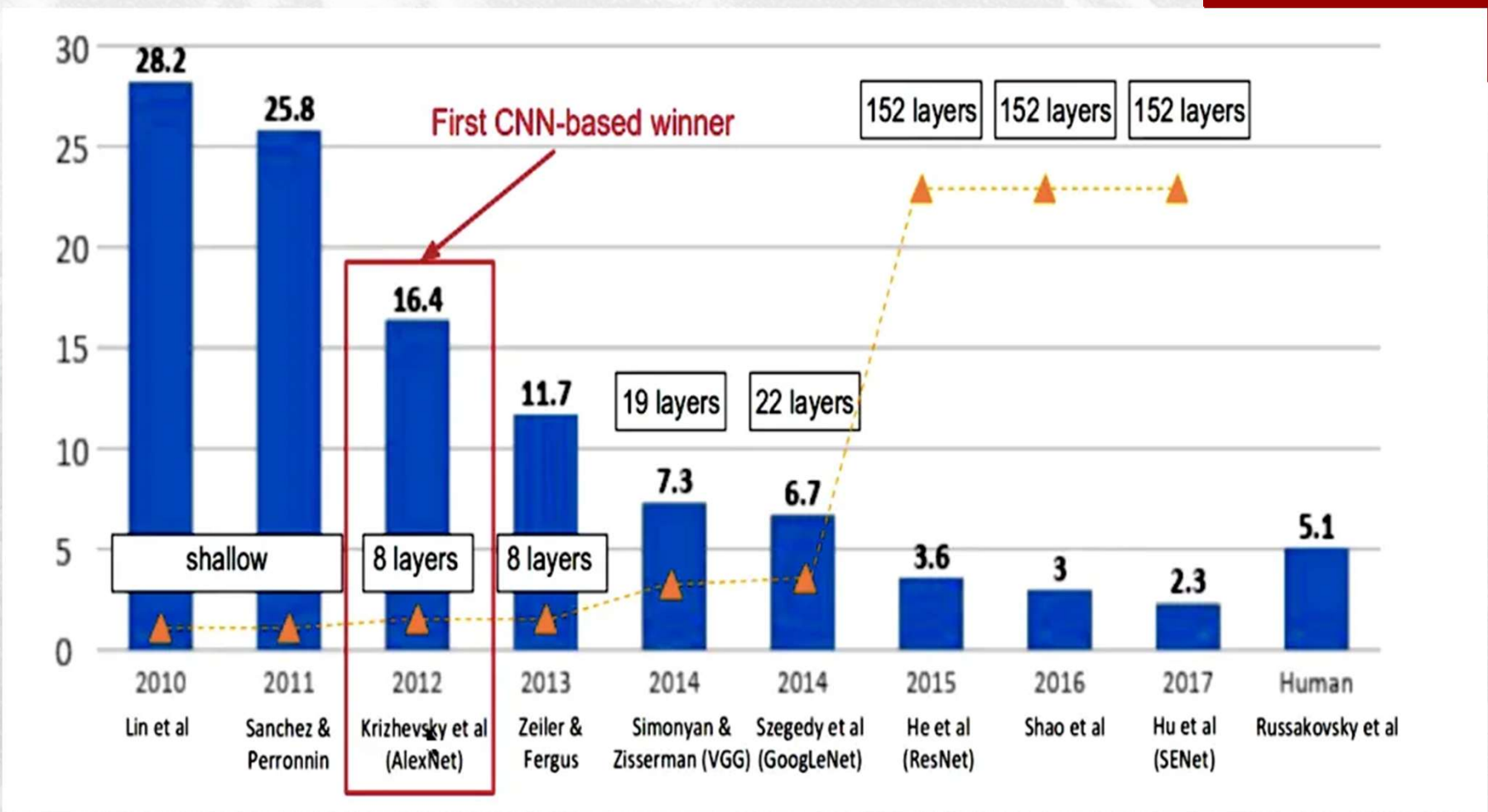
# A plateau, then rapid advances



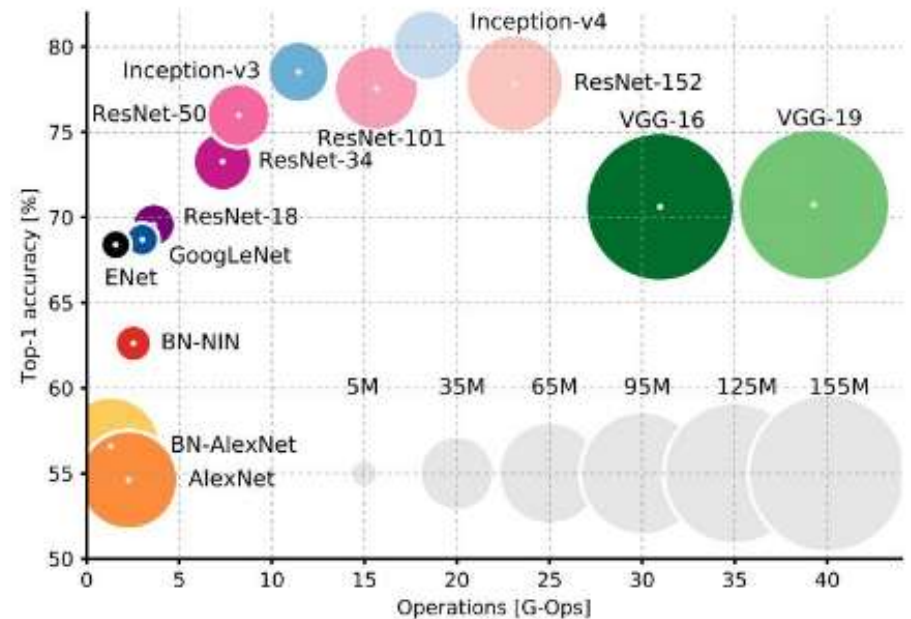
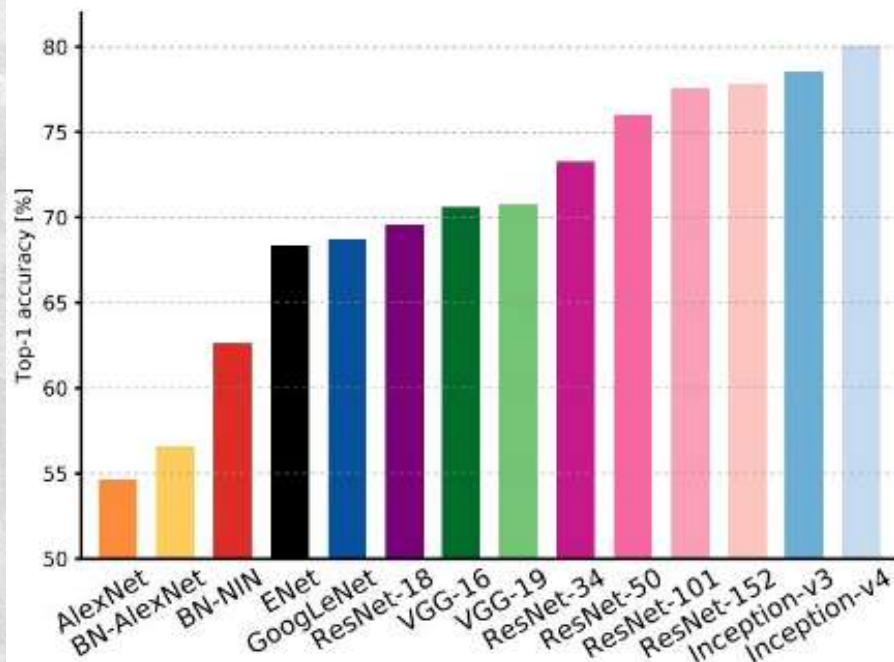
- “Top-5 error” is the % of times that the target label does not appear among the 5 highest-probability predictions
- Visual recognition methods not based on deep CNNs hit a plateau in performance at 25%

Name	Layers	Top-5 Error (%)	References
AlexNet	8	15.3	Krizhevsky et al. (2012)
VGG Net	19	7.3	Simonyan and Zisserman (2014)
ResNet	152	3.6	He et al. (2016)

- Note: the performance for human agreement has been measured at 5.1% top-5 error
- Smaller filters have been found to lead to superior results in deep networks: the methods with 19 and 152 layers use filters of size 3×3



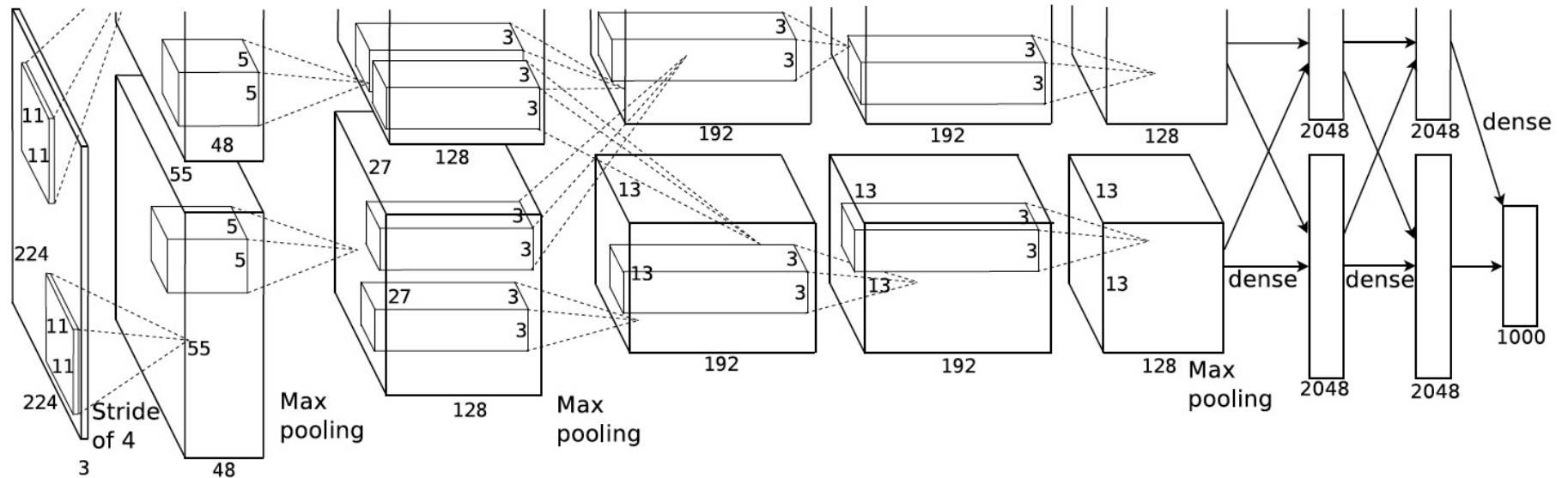
# Applications, size and accuracy



An Analysis of Deep Neural Network Models for Practical Applications, 2017.



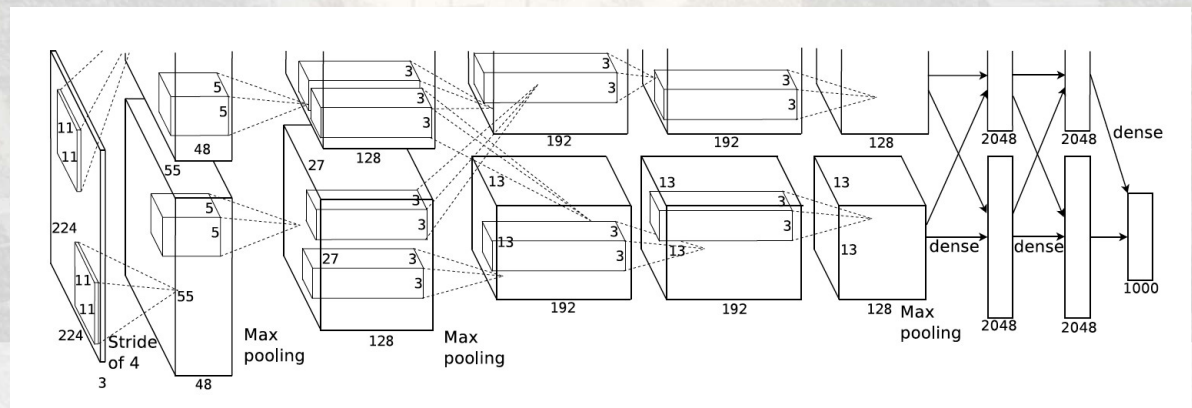
# An example: AlexNet (8 Layers)

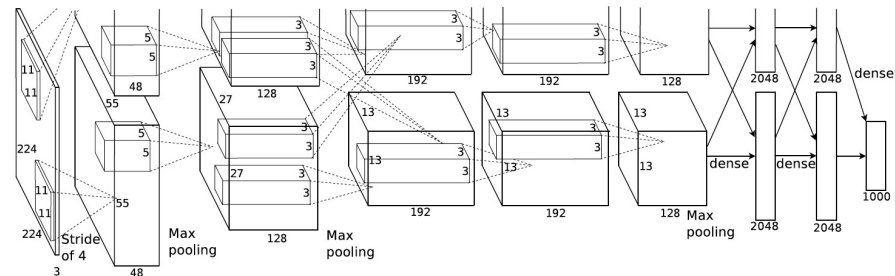


AlexNet (2017) won the 2012 ImageNet competition with a top-5 error rate of 15.3%, compared to the second place top-5 error rate of 26.2%

# AlexNet: the architecture

- It has 8 layers with learnable parameters.
- The input to the Model is RGB images.
- It has 5 convolution layers with a combination of max-pooling layers.
- Then it has 3 fully connected layers.
- The activation function used in all layers is **Relu**, whereas **Softmax** is used in the output layer
- It used **two Dropout layers**.
- The total number of parameters in this architecture is **62.3 million**.





# AlexNet: Overview

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

$$\hat{y}_i = \frac{e^{y_i}}{\sum_{j=1}^{1000} e^{y_j}}$$

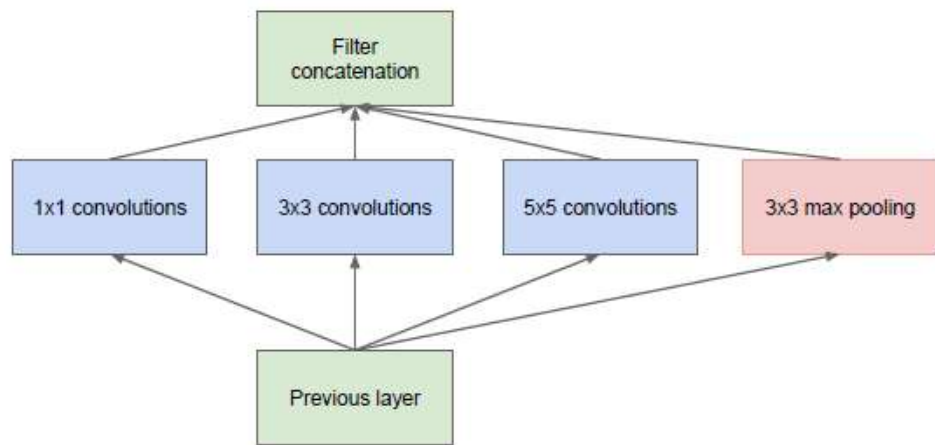


# What has been learnt?

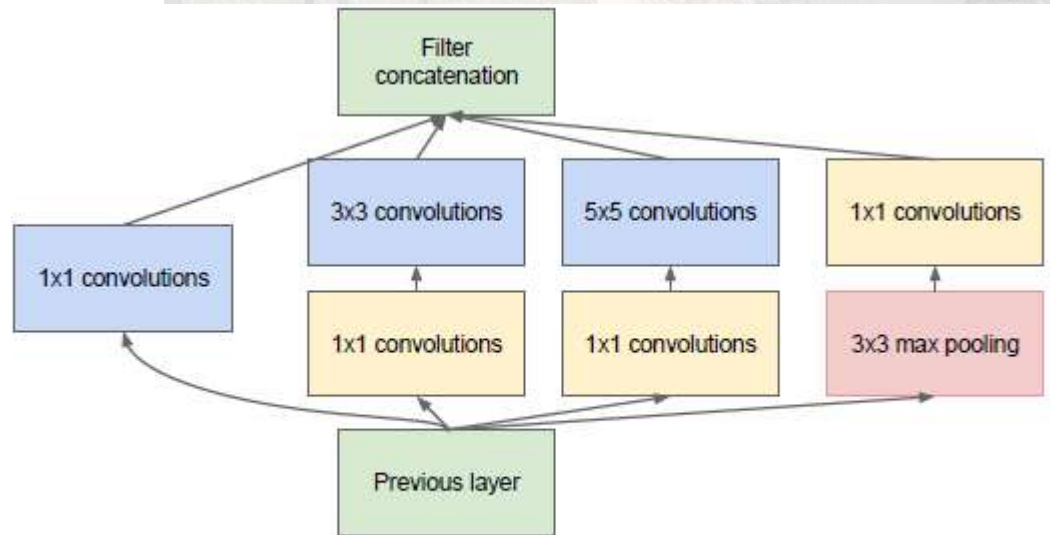


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

# GoogleLeNet (Inception V1)



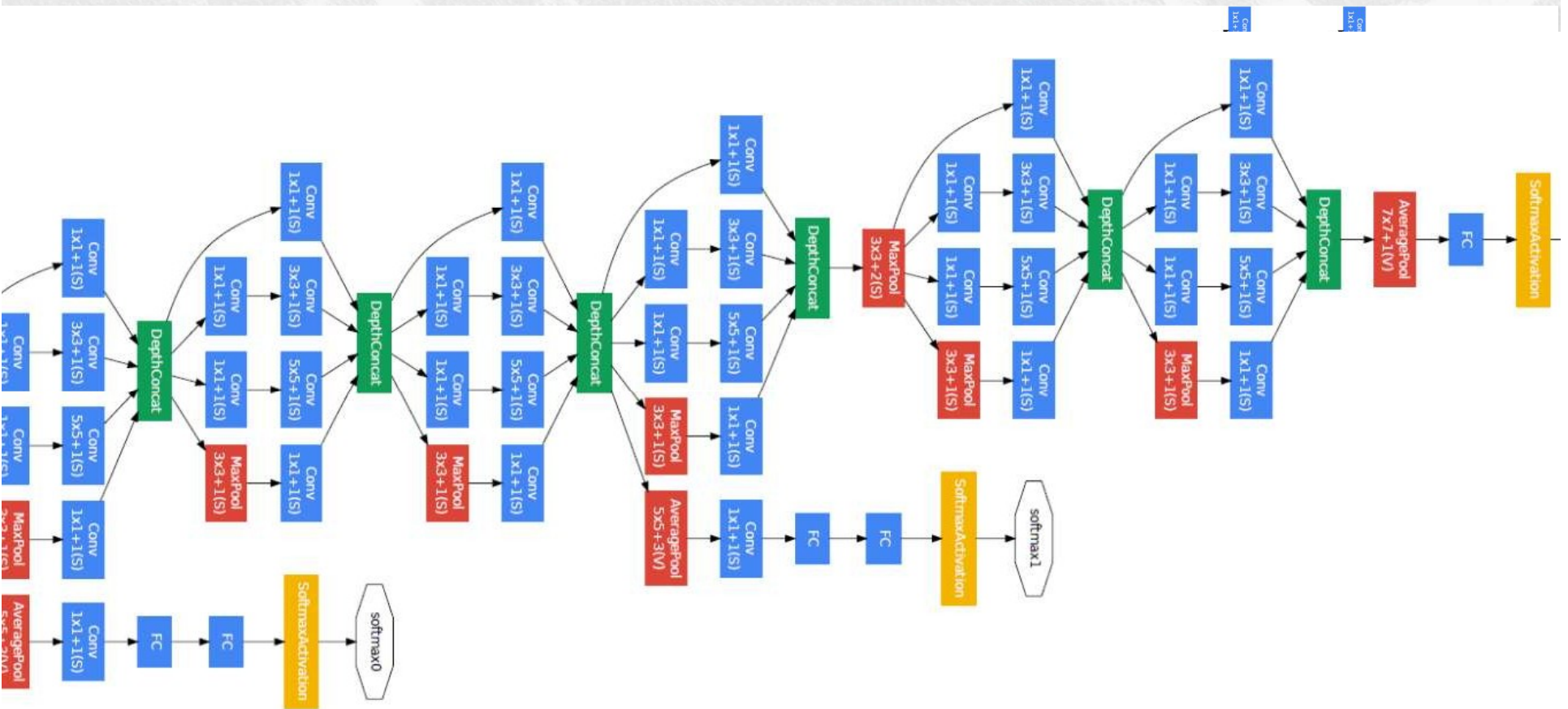
(a) Inception module, naïve version



(b) Inception module with dimensionality reduction



1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

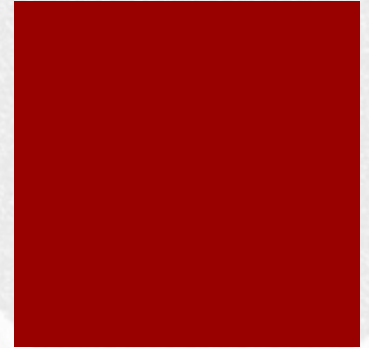




# Parameters in GoogleLeNet

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								

# An example: 1x1 convolutions



- A 1x1 convolution simply maps **an input pixel** with all its channels to **an output pixel**, not looking at anything around itself.
- It is often **used to reduce the number of depth channels**, since it is often very slow to multiply volumes with extremely large depths.

```
input (256 depth) -> 1x1 convolution (64 depth) -> 4x4 convolution (256 depth)
input (256 depth) -> 4x4 convolution (256 depth)
```

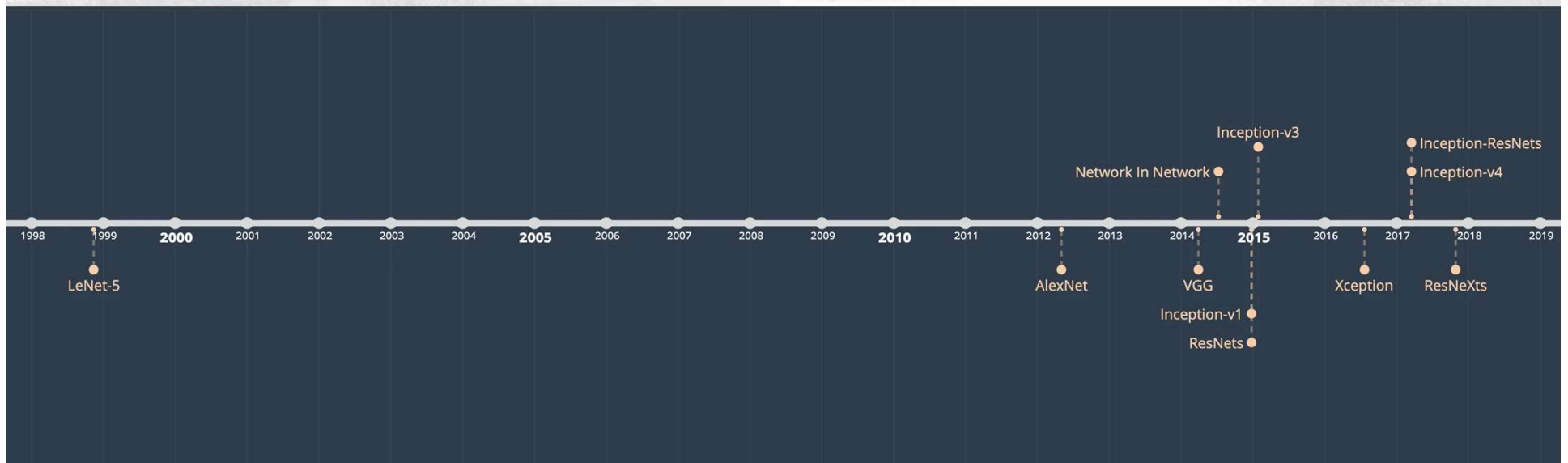
- The second chain is about ~3.7x slower.

# A summary (2017)

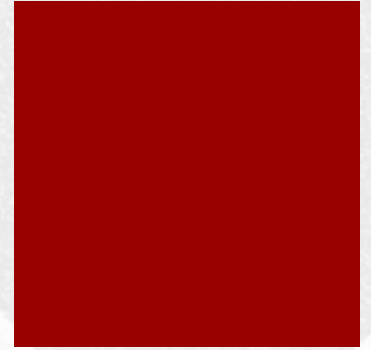
Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	<u>ResNet(152)</u>	Kaiming He	1st	3.6%	



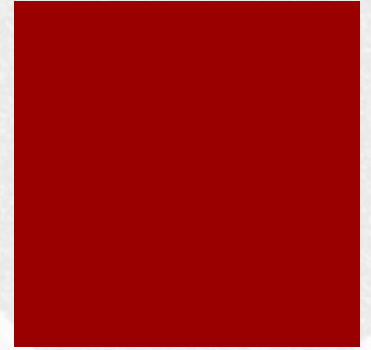
# CNN Timeline



# Visualization



# Visualizing the filters learned by a CNN



- Learned edge-like filters and texture-like filters are frequently observed in the early layers of CNNs trained using natural images
- Since each layer in a CNN involves filtering the feature map below, so as one moves up the receptive fields become larger
- Higher- level layers learn to detect larger features, which often correspond to textures, then small pieces of objects



# How to visualize hidden layers

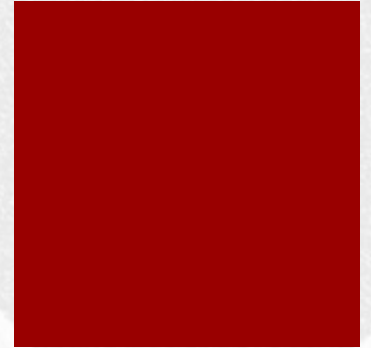
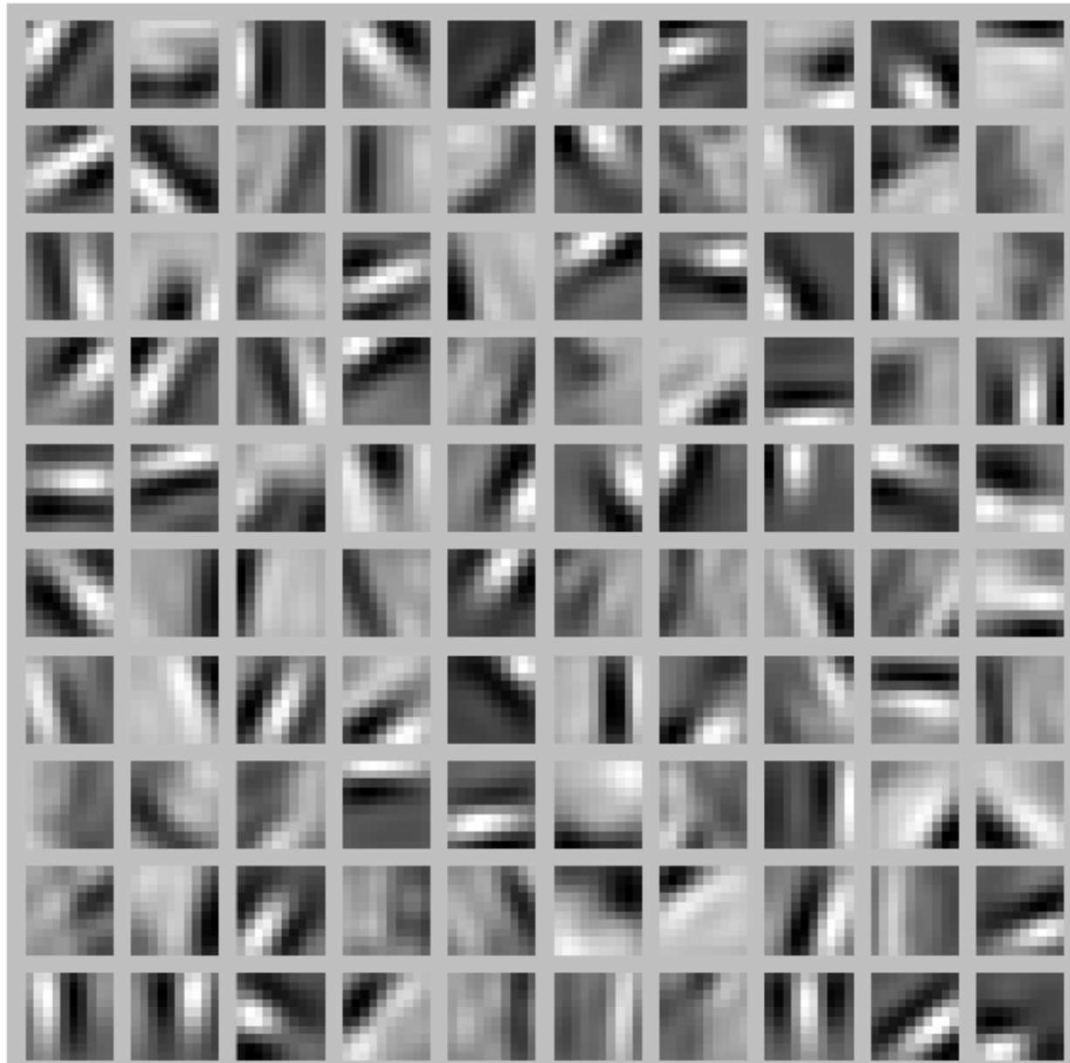
- Imagine to train a neural classifier on  $10 \times 10$  images, so that  $n = 100$ . Each hidden unit  $i$  computes a function of the input:

$$a_i^{(2)} = f \left( \sum_{j=1}^{100} W_{ij}^{(1)} x_j + b_i^{(1)} \right)$$

- What input image  $\underline{x}$  would cause  $a_i^{(1)}$  to be maximally activated?
- (When  $\|\underline{x}\|^2 = \sum_{i=1}^{100} x_i^2 \leq 1$ ) the input which maximally activates hidden unit  $i$  is given by setting pixel  $x_j$  to:

$$x_j = \frac{W_{ij}^{(1)}}{\sqrt{\sum_{j=1}^{100} (W_{ij}^{(1)})^2}}$$

Example: 100 hidden units



# Visualizing the filters learned by a CNN



First Layer



Second Layer



Third Layer

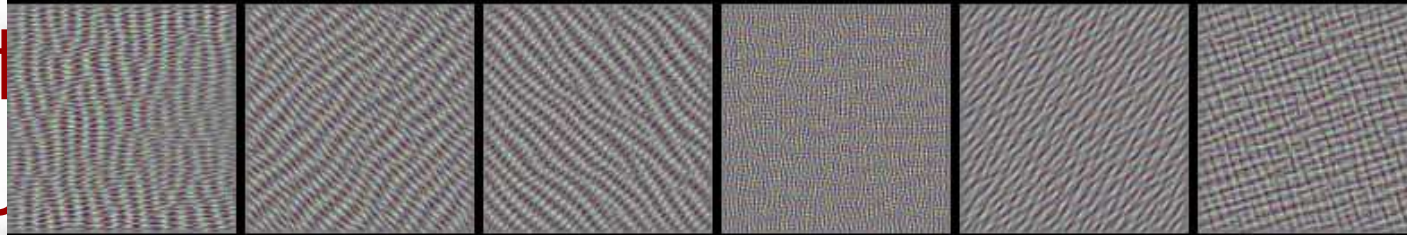
(Imagery kindly provided by Matthew Zeiler)

- Above are the strongest activations of random neurons projecting the activation back into image space using the deconvolution approach of Zeiler and Fergus (2013).

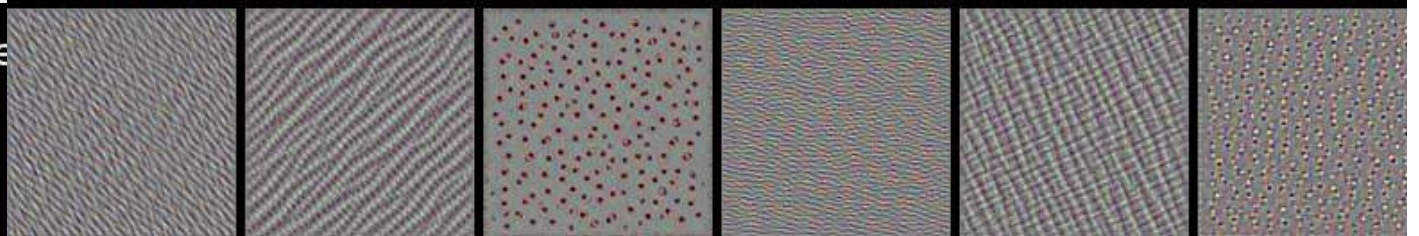


# ConvNet filters visu

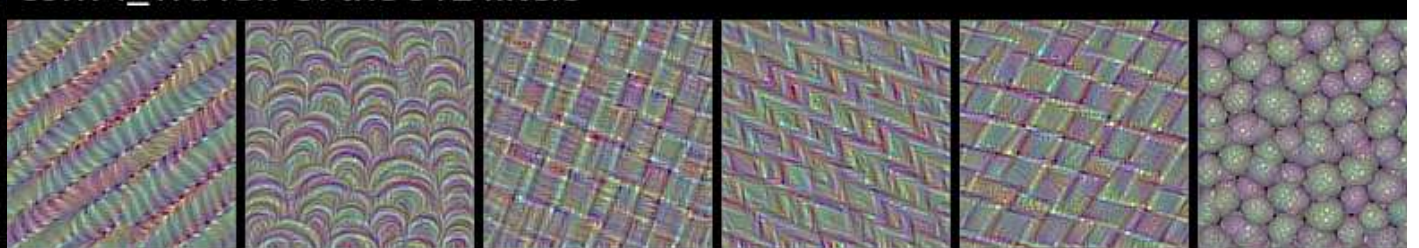
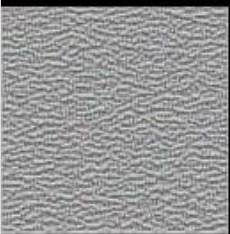
conv3\_1: a few of the 256 filters



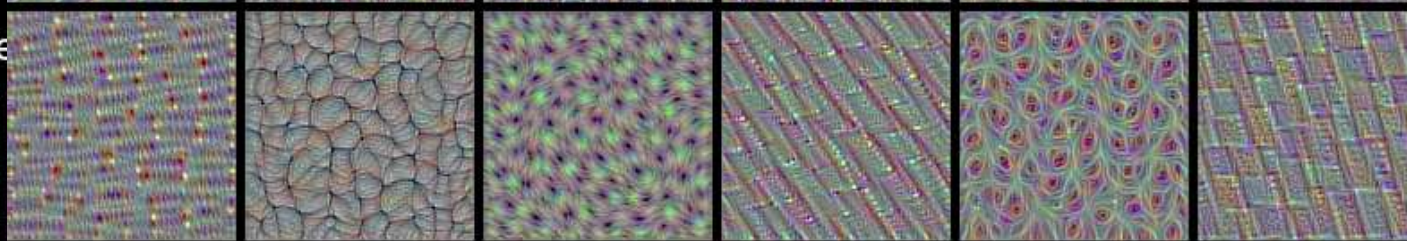
conv1\_1: a few of the 16 filters



conv4\_1: a few of the 512 filters



conv2\_1: a few of the 64 filters



conv5\_1: a few of the 512 filters





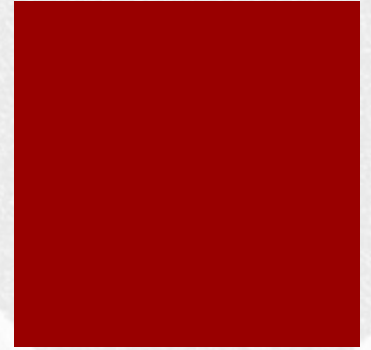
# An interesting visualization tool

- CNN Explainer

- <https://poloclub.github.io/cnn-explainer/>
- Paper at: <https://arxiv.org/abs/2004.15004>



# Current CNNs: YOLO



- **YOLO** stands for **You Only Look Once** and is a popular real-time object detection algorithm.
- YOLO unifies a multi-step process by using a single neural network to perform both classification and prediction of bounding boxes for detected objects.
- As such, it is heavily optimized for detection performance, about 45 frames per second.
- It can run much faster than running two separate neural networks to detect and classify objects separately.

# YOLO: the output size

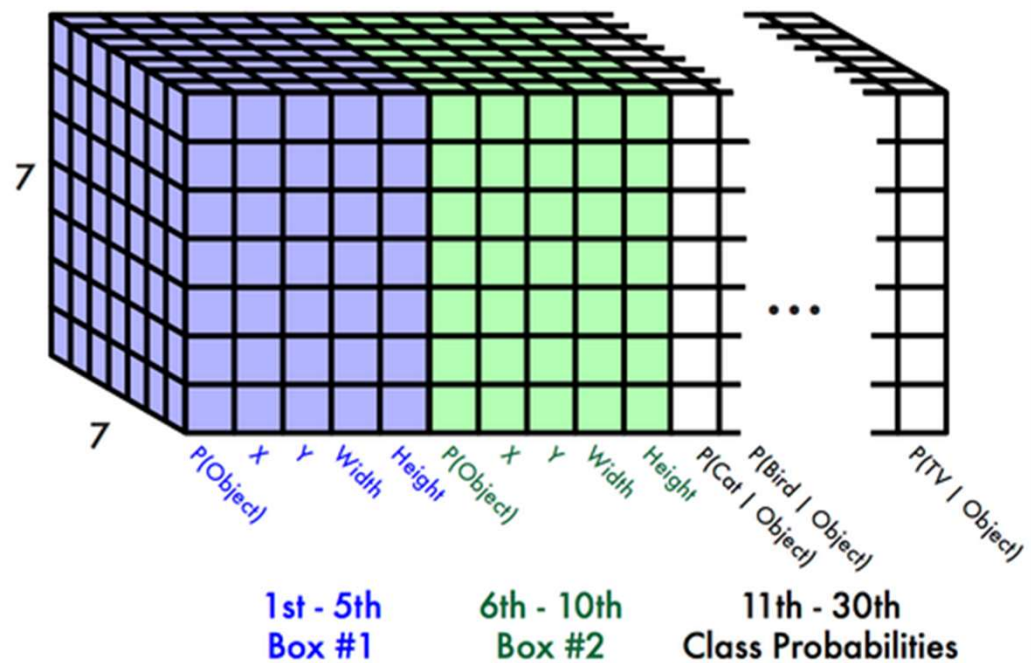
Each cell predicts:

- For each bounding box:
  - 4 coordinates (x, y, w, h)
  - 1 confidence value
- Some number of class probabilities

For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

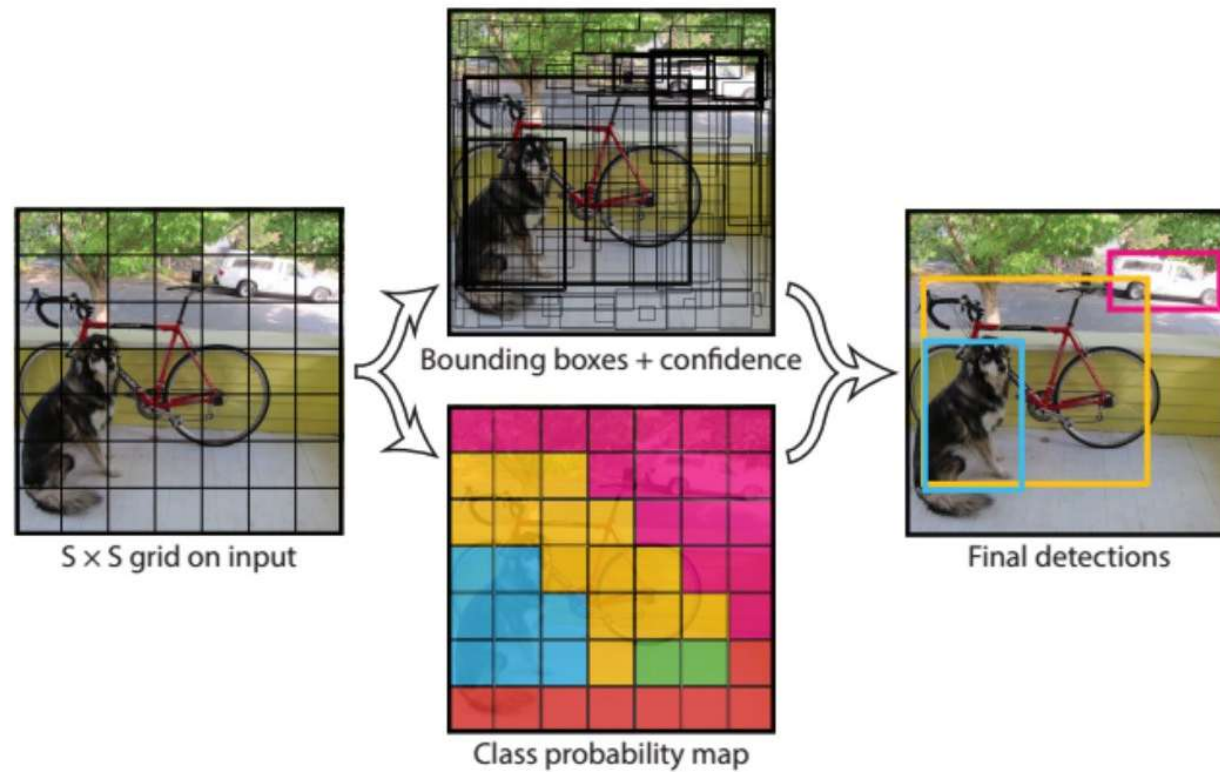
$7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30$  tensor = **1470 outputs**



See also: <https://pjreddie.com/publications/>



# Current CNNs: Yolo

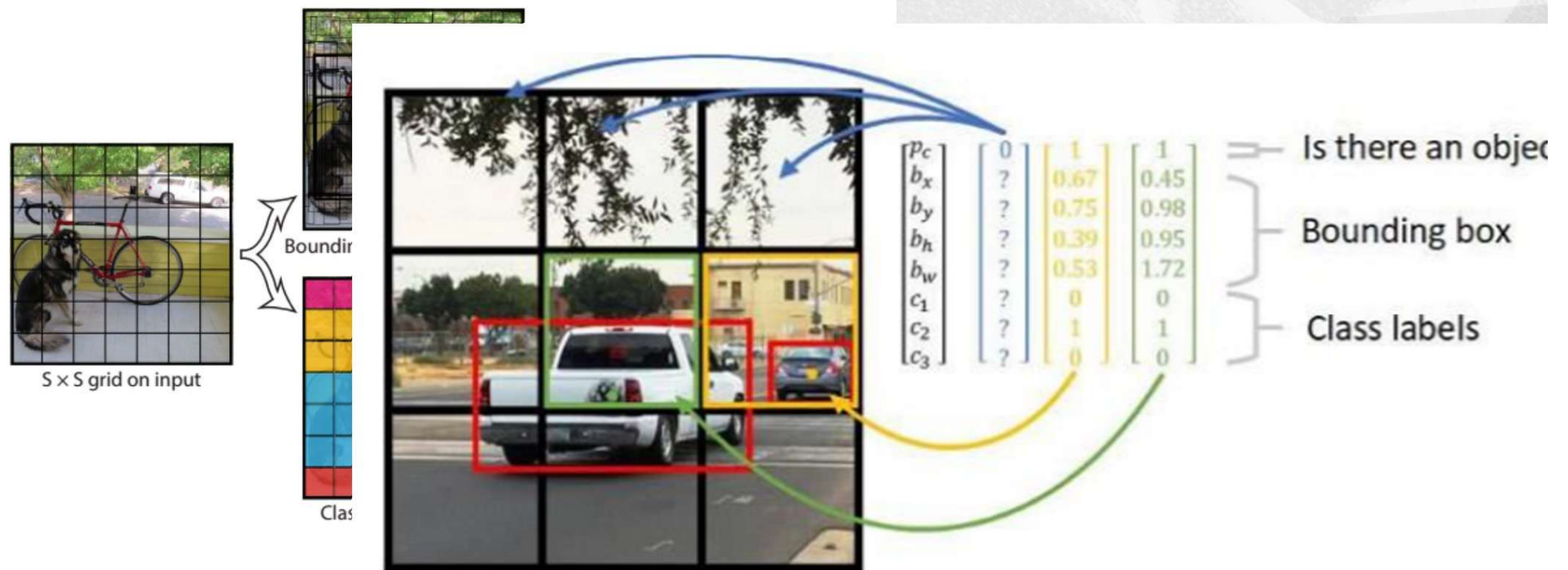


**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

From You Only Look Once: Unified, Real-Time Object Detection, Joseph Redmon et al., 2016., <https://arxiv.org/pdf/1506.02640v5.pdf>

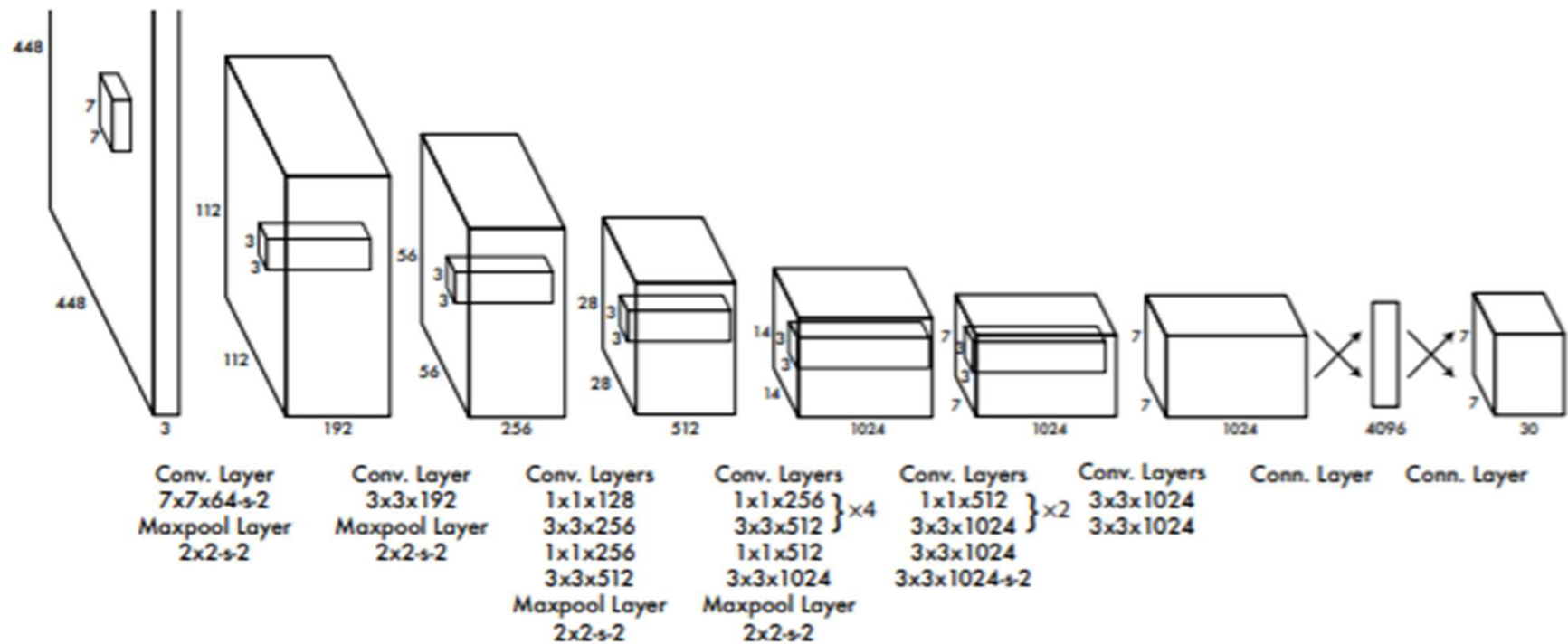


# Current CNNs: Yolo



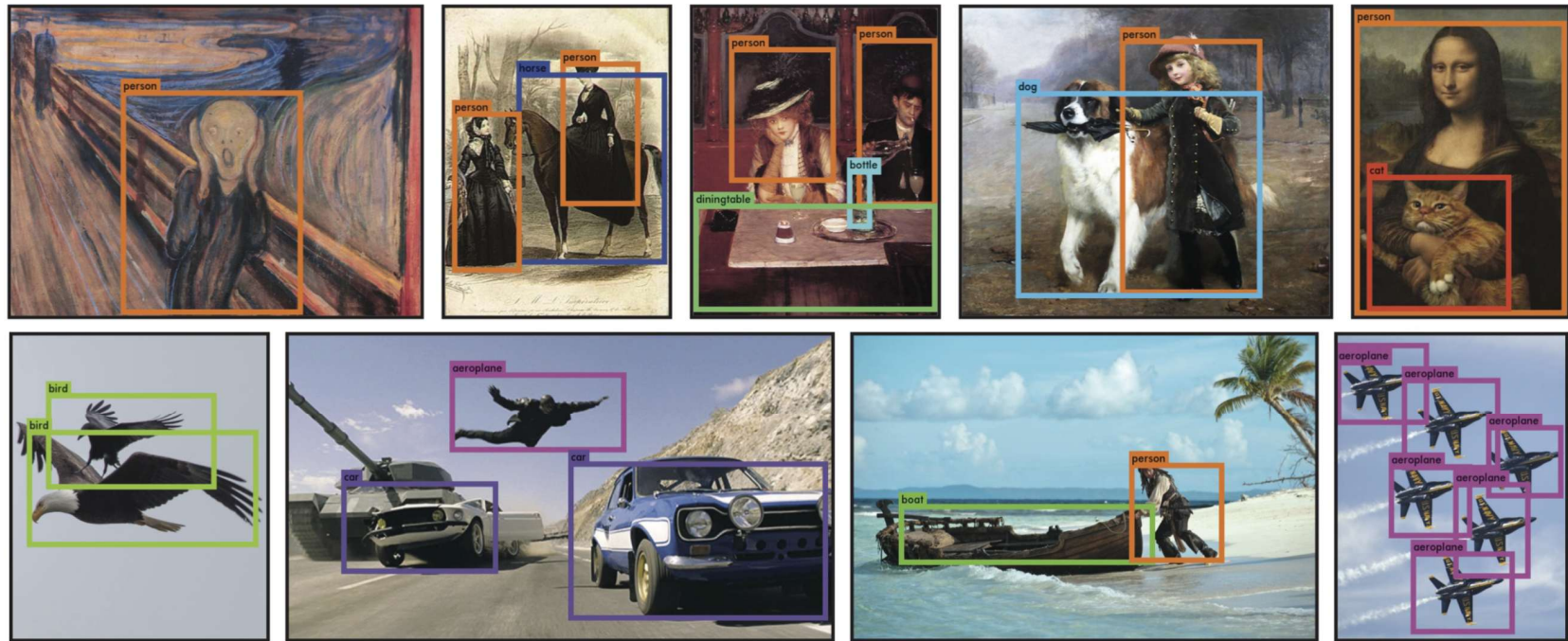
**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

# Yolo: the architecture





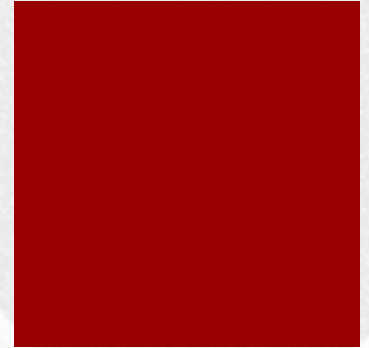
# Yolo: Results



**Figure 6: Qualitative Results.** YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.



# Integrating image and texts

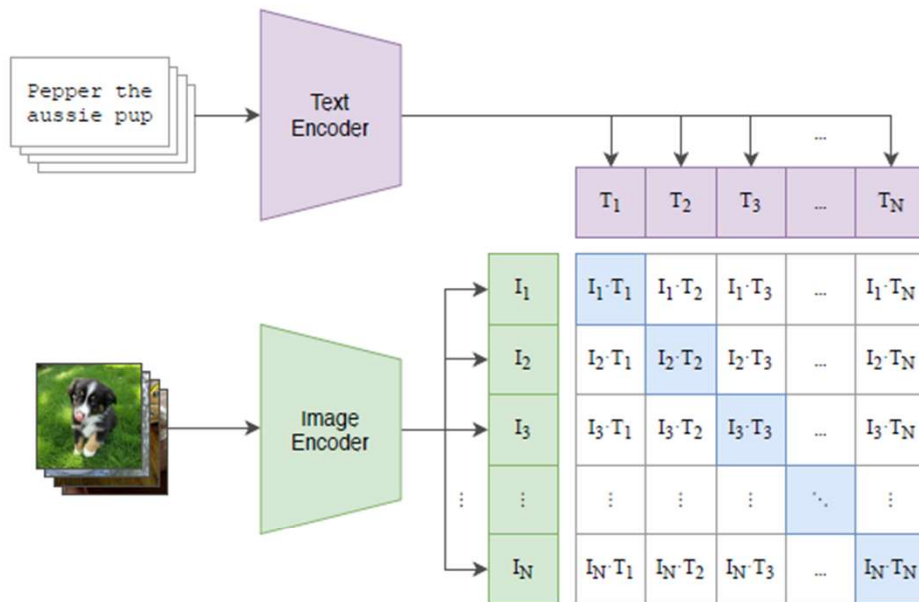


- Object Recognition usually employs ad hoc training data sets implying ad hoc CNN models
- The paper (\*) demonstrates that the simple pre-training task of predicting which caption goes with which image is an efficient and scalable way to learn SOTA image representations from scratch on a dataset of 400 million (image, text) pairs collected from the internet.
- **After pre-training**, natural language is used to reference learned visual concepts (or describe new ones) enabling zero-shot transfer of the model to downstream tasks.
- **Zero-shot learning**: solving an object recognition task without ANY training example
- **The IDEA: Optimizing the behaviours of image classifiers trained with natural language supervision at large scale.**

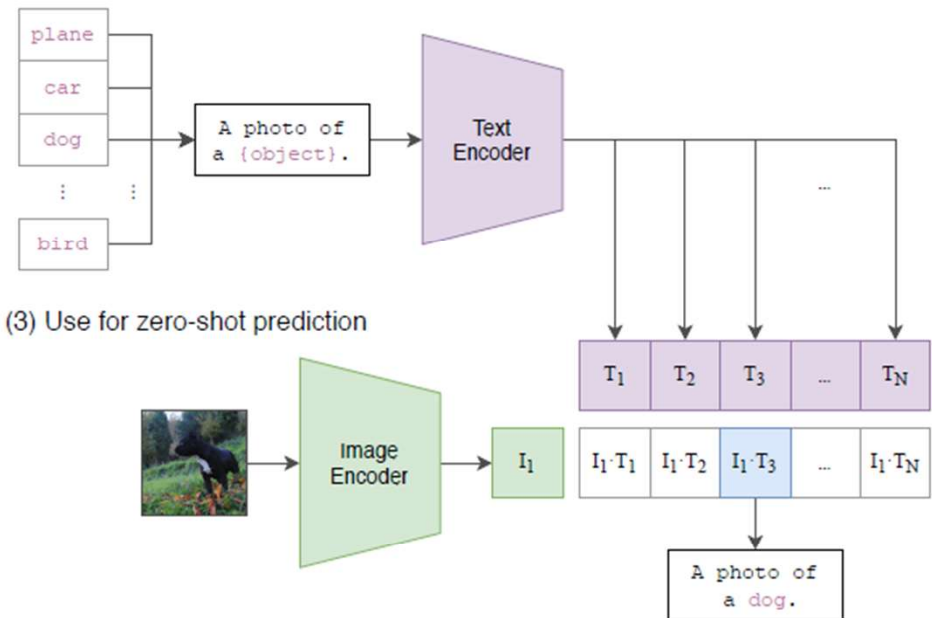
(\*) Learning Transferable Visual Models From Natural Language Supervision, Redford et al, 2021  
<https://arxiv.org/abs/2103.00020v1>

# CLIP (Contrastive Language-Image Pre-training)

(1) Contrastive pre-training



(2) Create dataset classifier from label text



(3) Use for zero-shot prediction

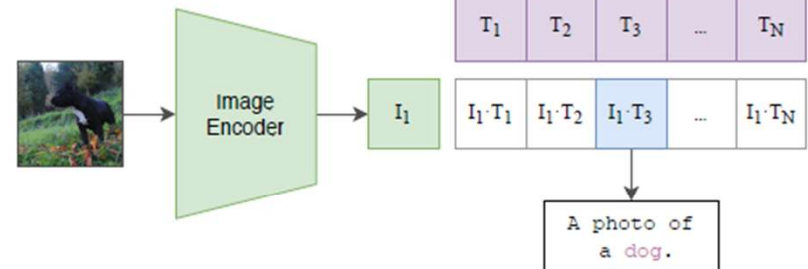


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.



# Outline

- Architectures and tasks
- Convolutional Neural Networks
  - Filters and Convolutions
  - Pooling
- Imagenet
- Applications of NNs:
  - Image processing: classification, Object Recognition
  - Text Classification:
    - Convolutional NNs over texts
    - Sentiment analysis
    - The Movie Review Dataset





# Sentence encoding & convolution

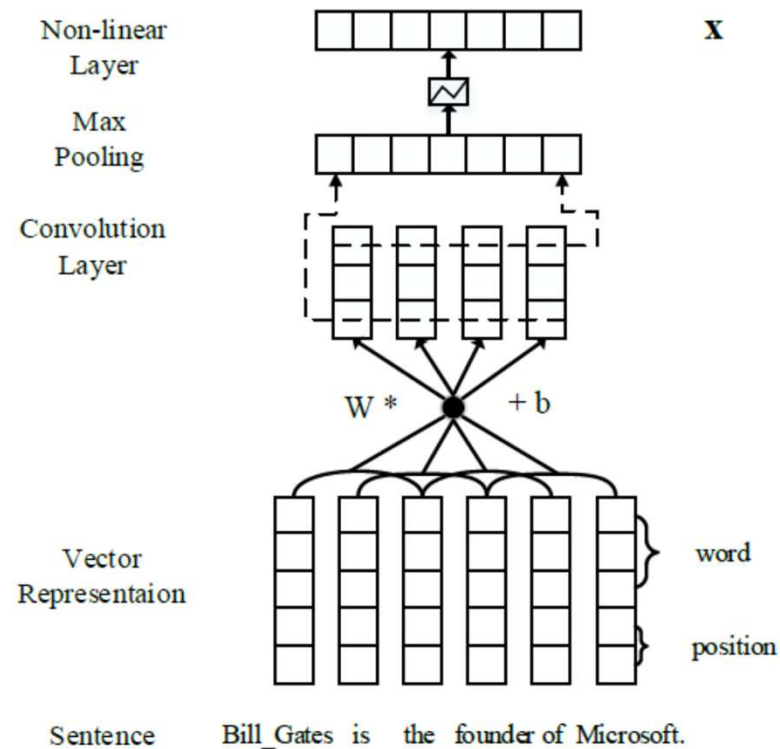
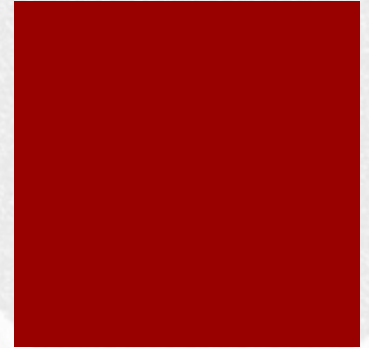


Figure 2: The architecture of CNN/PCNN used for sentence encoder.

from Yankai Lin, Shiqi Shen, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2016.  
[Neural Relation Extraction with Selective Attention over Instances](#). ACL 2016.

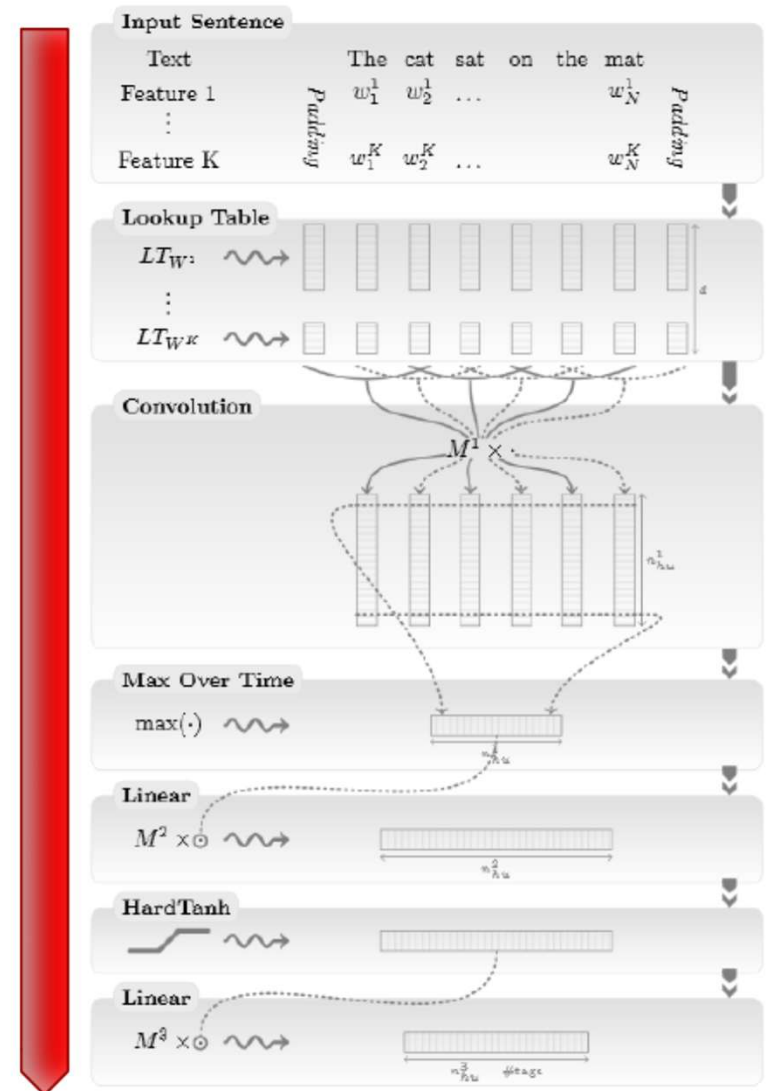
# From Collobert et al., 2011

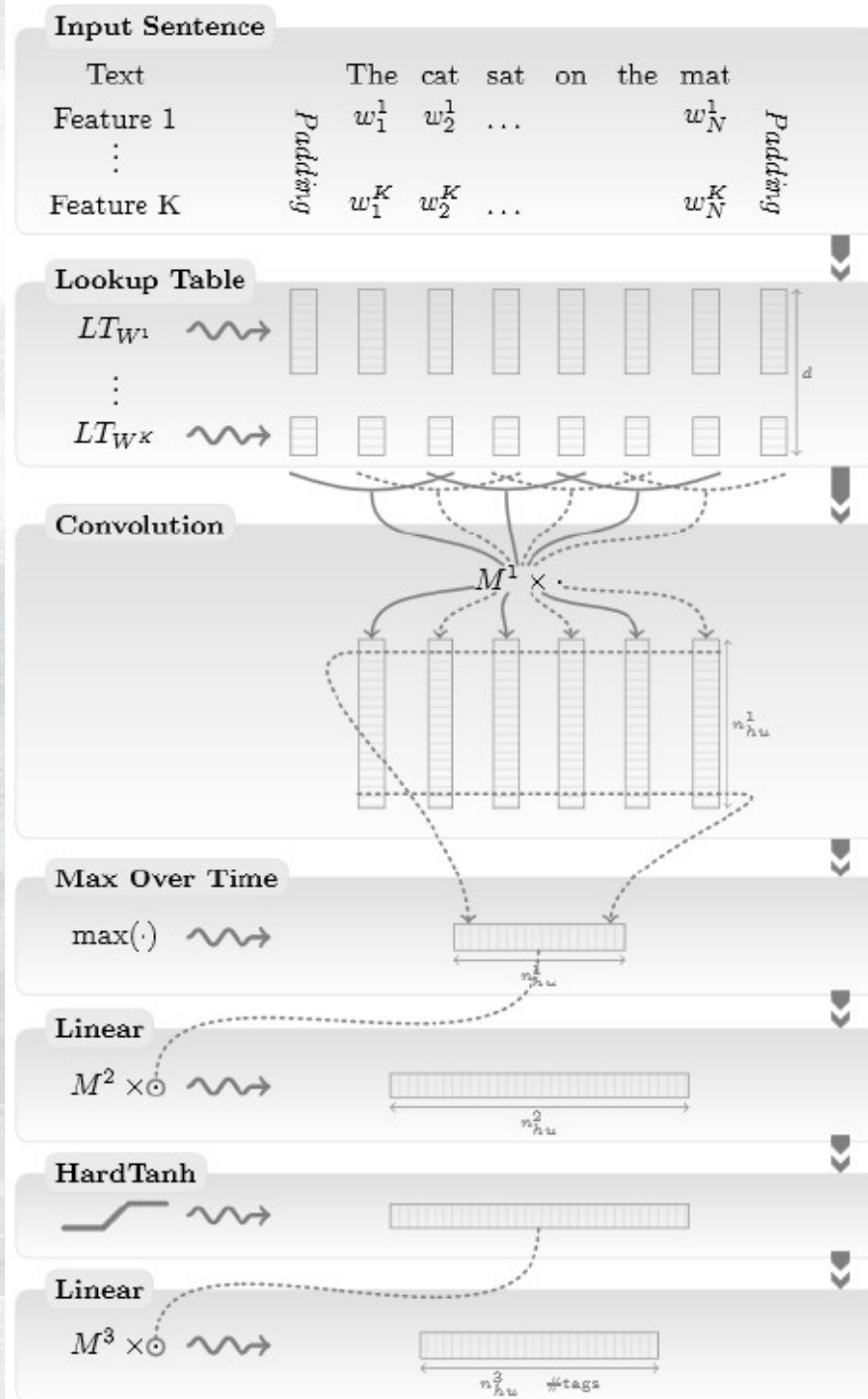


- In this contribution, we try to excel on **multiple benchmarks** while *avoiding task-specific engineering*. Instead *we use a single learning system* able to **discover adequate internal representations**. In fact we view the *benchmarks as indirect measurements of the relevance of the internal representations* discovered by the learning procedure, and we posit that these *intermediate representations are more general than any of the benchmarks*.
- Our desire to avoid task-specific engineered features prevented us from using a large body of linguistic knowledge
- The architecture takes the input sentence and learns several layers of feature extraction that process the inputs. The features computed by the deep layers of the network are automatically trained by backpropagation to be relevant to the task.



- ▶ Collobert and Weston used CNNs to achieve (near) state-of-the-art results on many traditional NLP tasks, such as POS tagging, SRL, etc.
- ▶ CNN at the bottom + CRF on top.
- ▶ Collobert et al., “Natural Language Processing (almost) from scratch”, JLMR 2011.

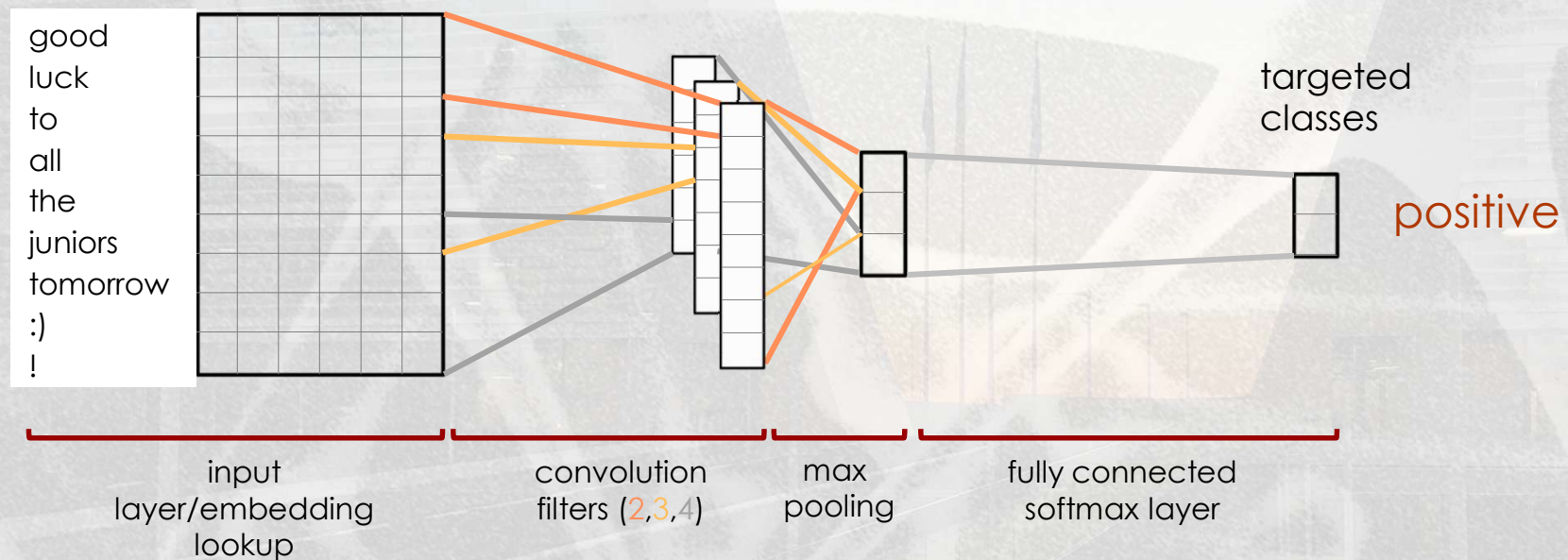






# A CNN architecture for sentence classification (Kim,2014)

*good luck to all the juniors tomorrow :) !*



# Multi-channel CNNs

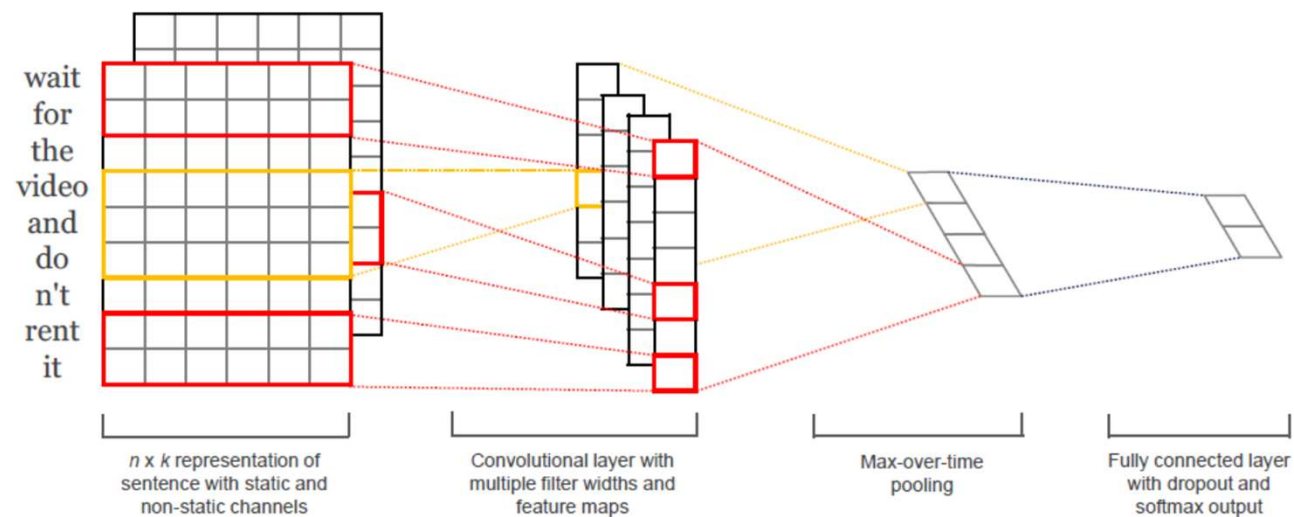


Figure 1: Model architecture with two channels for an example sentence.

- ▶ Two “channels” of embeddings (i.e. look-up tables).
- ▶ One is allowed to change, while one is kept fixed.
- ▶ Both initialized with word2vec.



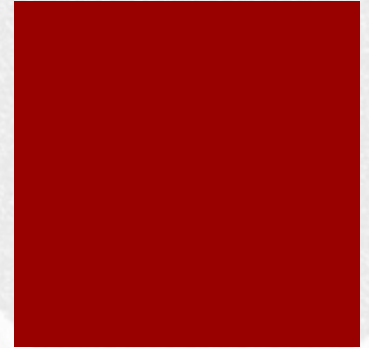
# Datasets

Sentence/phrase-level classification tasks

<b>Data</b>	$c$	$l$	$N$	$ V $	$ V_{pre} $	Prev SotA
MR	2	20	10662	18765	16448	79.5
SST-1	5	18	11855	17836	16262	48.7
SST-2	2	19	9613	16185	14838	87.8
Subj	2	23	10000	21323	17913	93.6
TREC	6	10	5952	9592	9125	95.0
CR	2	19	3775	5340	5046	82.7
MPQA	2	3	10606	6246	6083	87.2

- ▶  $c$ : number of labels
- ▶  $l$ : average sentence length
- ▶  $N$ : number of sentences
- ▶  $|V|$ : vocab size ( $|V_{pre}|$  is words already in word2vec)

# MR dataset (Pang & Lee, 2005)



## ■ Negative:

- "it's so laddish and juvenile , only teenage boys could possibly find it funny . "
- while the performances are often engaging , this loose collection of largely improvised numbers would probably have worked better as a one-hour *tv* documentary .

## ■ Positive

- if you sometimes like to go to the movies to have fun , wasabi is a good place to start .
- gosling provides an amazing performance that dwarfs everything else in the film .



# SST (Stanford Sentiment Treebank, 2013)


- *This was the worst restaurant I have ever had the misfortune of eating at.*
- *The restaurant was a bit slow in delivering their food, and they didn't seem to be using the best ingredients.*
- *This restaurant is pretty decent— its food is acceptable considering the low prices.*
- *This is the best restaurant in the Western Hemisphere, and I will definitely be returning for another meal!*

Complex cases:

*I do not hate this restaurant.* (Negation)

*I just love being served cold food!* (Sarcasm)


*The food is unnervingly unique.* (Negative words being positive)



<b>Data</b>	Prev SotA	CNN-rand	CNN-static	CNN-nonstatic
MR	79.5	76.1	81.0	81.5
SST-1	48.7	45.0	45.5	48.0
SST-2	87.8	82.7	86.8	87.2
Subj	93.6	89.6	93.0	93.4
TREC	95.0	91.2	92.8	93.6
CR	82.7	79.8	84.7	84.3
MPQA	87.2	83.4	89.6	89.5

- ▶ Fine-tuning vectors helps, though not that much.
- ▶ Perhaps our embeddings are overfitting (given the relatively small training sample)?





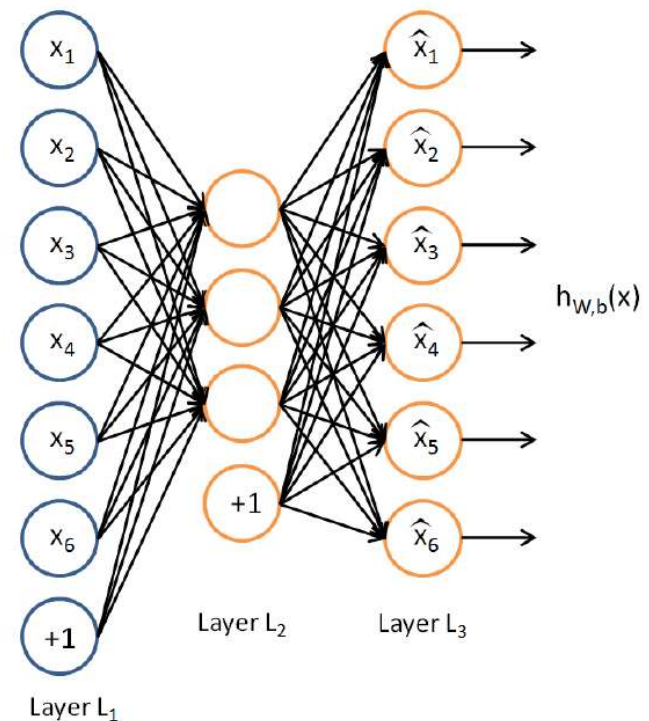
<b>Data</b>	Prev SotA	CNN-nonstatic	CNN-multichannel
MR	79.5	81.5	81.1
SST-1	48.7	48.0	47.4
SST-2	87.8	87.2	88.1
Subj	93.6	93.4	93.2
TREC	95.0	93.6	92.2
CR	82.7	84.3	85.0
MPQA	87.2	89.5	89.4

- Performance is not statistically different from CNN-nonstatic.

# What's next: autoencoders

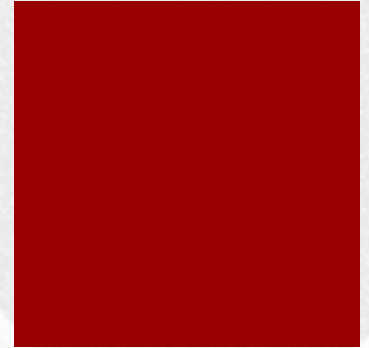
- An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the input itself, i.e., it uses

$$y(i) = x(i)$$





# Autoencoders



- Suppose the inputs  $\underline{x}$  are the **pixel intensity values** from a **10×10 image** (100 pixels) so  $n = 100$ , and
- there are  $s_2 = 50$  hidden units in layer L2.
- Note that we also have  $\underline{y} \in \mathbb{R}^{100}$ .
- Since there are only 50 hidden units, the network is forced to learn a **compressed representation** of the input. I.e., given only the vector of hidden unit activations  $\underline{a}^{(2)} \in \mathbb{R}^{50}$ , it must try to reconstruct the 100-pixel input  $\underline{x}$ .
- Compressed representation may be seen as *lower dimensional embeddings*
- *Images? Sentences? Longer Texts?*

# Bibliography

- Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard and W. Hubbard: Handwritten Digit Recognition: applications of Neural Net Chips and Automatic Learning, IEEE Communication, 41-46, invited paper, November 1989
- Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278{2324, 1998a.
- Bengio Yoshua. Learning Deep Architectures for AI. Foundations and Trends in Machine Learning 2 (1): 1–127.
- Deep Visual-Semantic Alignments for Generating Image Descriptions. Andrej Karpathy, Li Fei-Fei, CVPR 2015
- Y. Kim, Convolutional Neural Networks for Sentence Classification, Proc. of EMNLP, Doha, Qatar, 2014.
- J. Redmon et al., You Only Look Once: Unified, Real-Time Object Detection, 2016., <https://arxiv.org/pdf/1506.02640v5.pdf>
- Alec Radford, Jong Wook Kim, Chris Hallacy, A. Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, I. Sutskever, Learning Transferable Visual Models From Natural Language Supervision, Published in International Conference on Machine Learning, 26 February 2021
- Convolutional Neural Networks tutorial:
  - <http://cs231n.github.io/> : stanford course on CNN for visual recognition with online (free) materials
  - <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>
  - Yann LeCun and Yoshua Bengio. 1998. Convolutional networks for images, speech, and time series. In The handbook of brain theory and neural networks, Michael A. Arbib (Ed.). MIT Press, Cambridge, MA, USA 255-258.



# Other Resources

- Some of this slides are based on
  - <https://cs.stanford.edu/~quocle/tutorial1.pdf>
  - [http://web.stanford.edu/class/cs294a/sparseAutoencoder\\_2011new.pdf](http://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf)
- Recent works
  - YOLO. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788).  
<https://doi.org/10.1109/CVPR.2016.91>
    - See also a survey such as: <https://arxiv.org/html/2408.09332v1>
  - CLIP: Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. *International Conference on Machine Learning*.  
<https://arxiv.org/abs/2103.00020>