

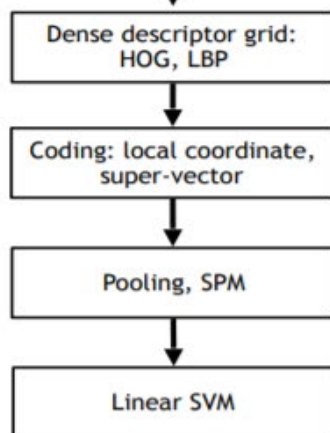
EEE511

Convolutional Neural Networks

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Year 2010

NEC-UIUC

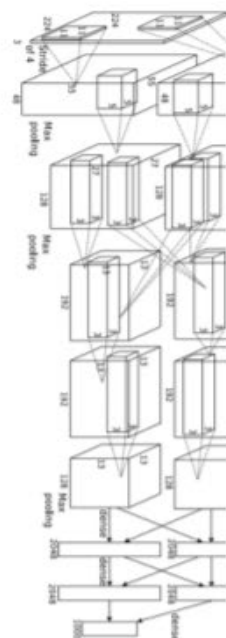


[Lin CVPR 2011]

Lion_image by Swissfrog is licensed under [CC BY 3.0](#)

Year 2012

SuperVision

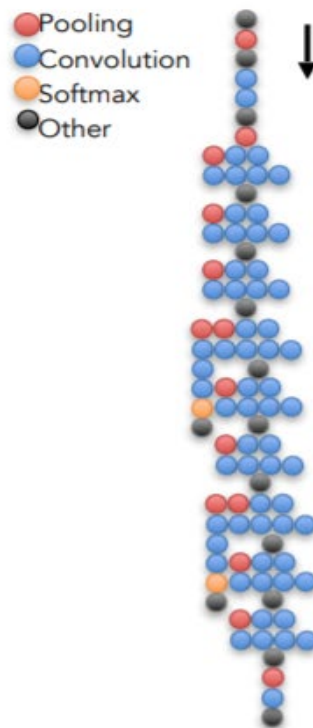


[Krizhevsky NIPS 2012]

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

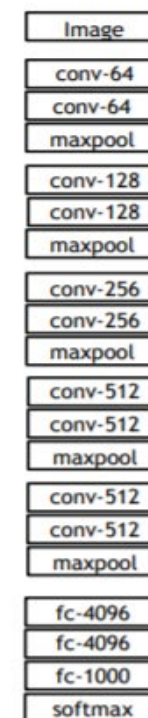
Year 2014

GoogLeNet



[Szegedy arxiv 2014]

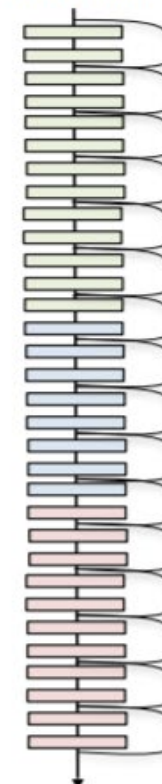
VGG



[Simonyan arxiv 2014]

Year 2015

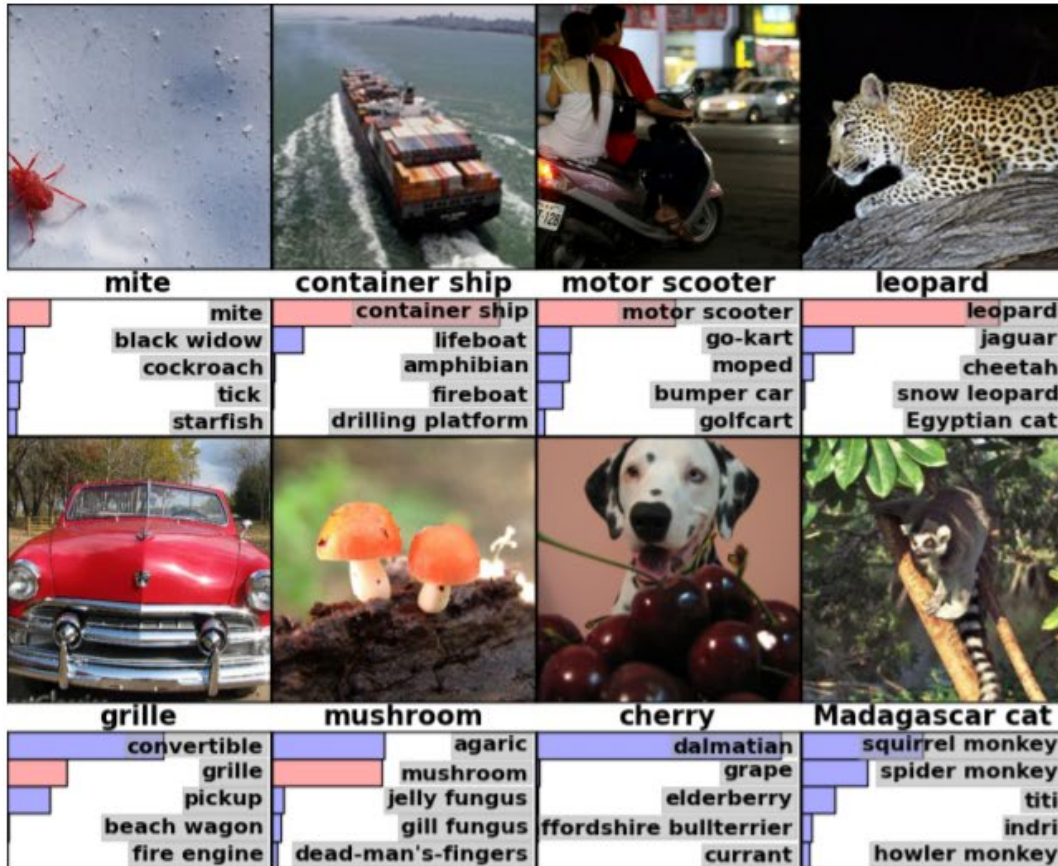
MSRA



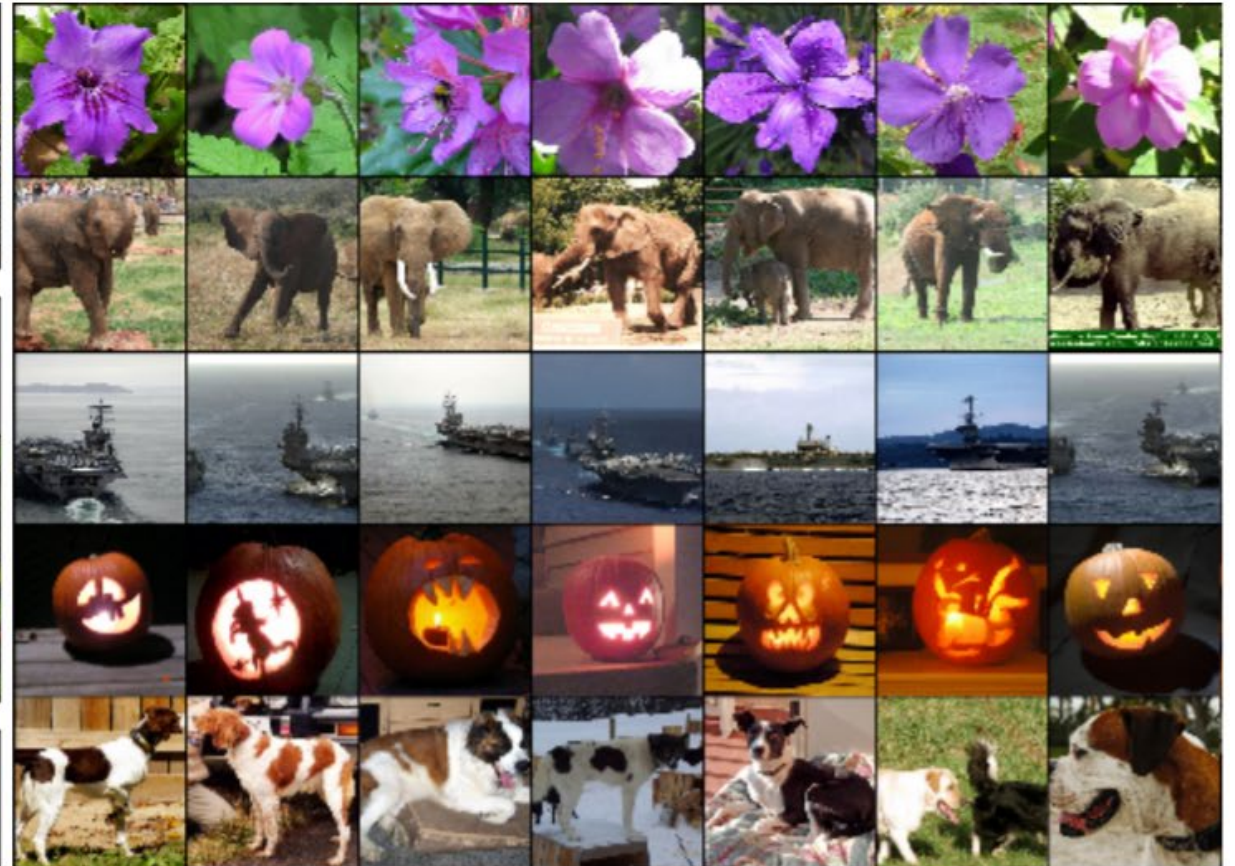
[He ICCV 2015]

CNN in computer vision

classification

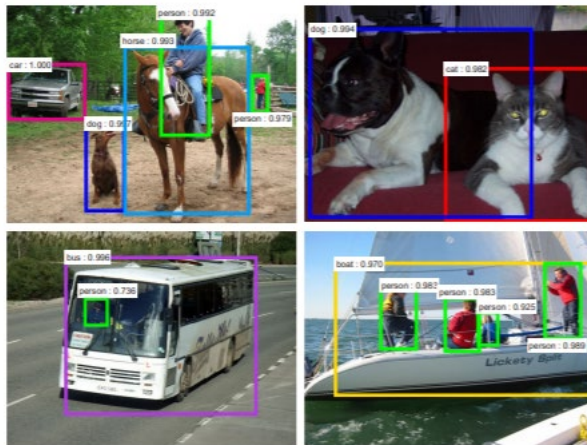


retrieval



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

detection



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems*. 2015.

segmentation



Farabet, Clement, et al. "Learning hierarchical features for scene labeling." *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2012): 1915-1929.

pose estimation



Figure 8. Visualization of pose results on images from LSP. Each pose is represented as a stick figure, inferred from predicted joints. Different limbs in the same image are colored differently, same limb across different images has the same color.



Figure 9. Visualization of pose results on images from FLIC. Meaning of stick figures is the same as in Fig. 8 above.

Toshev, Alexander, and Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.

Background

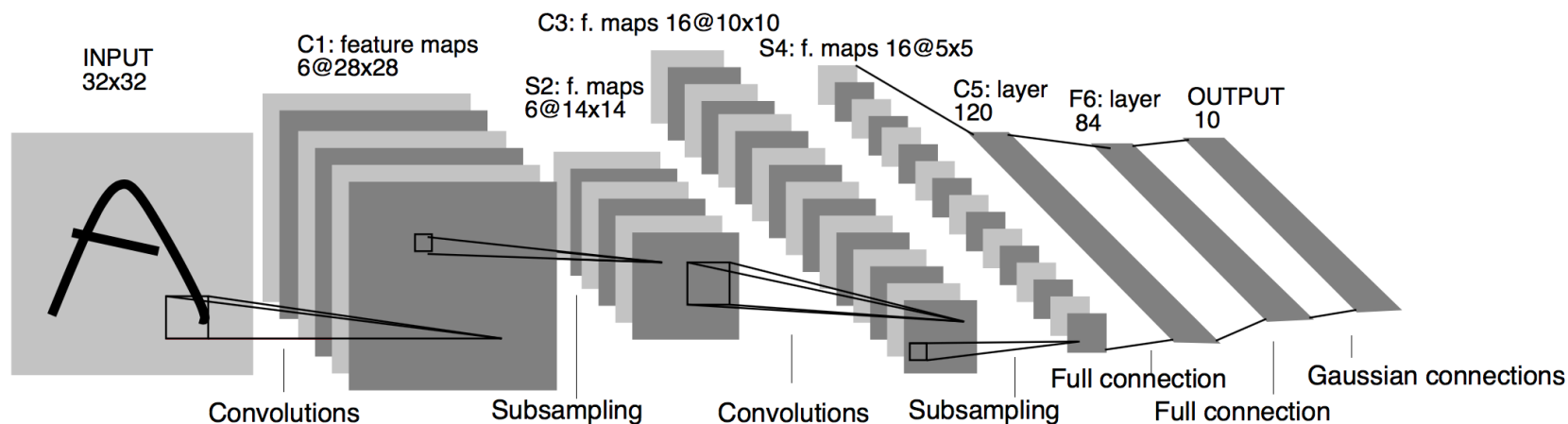
Hubel & Wiesel's demonstration of simple, complex and hypercomplex cells in the cat's visual cortex (1950s, 1960s)

<https://www.youtube.com/watch?v=jw6nBWo21Zk>

Fukushima introduced “Neocognitron”, origin of the CNN architecture (1980)

Yann LeCun et al. (1989) trained CNN kernels using BP. The procedure is suited to broader image recognition problems and it became the foundation of modern computer vision.

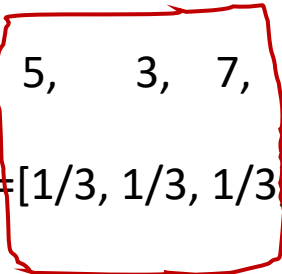
Yann LeCun et al. (1998), LeNet-5, for recognizing hand-written checks



Convolution - a simple 1D illustration

$$x=[1, 5, 3, 7, 5, 9, 7, 14]$$

$$y=[1/3, 1/3, 1/3]$$


$$x=[1, 5, 3, 7, 5, 9, 7, 14]$$
$$y=[1/3, 1/3, 1/3]$$

$$5 \times 1/3 + 3 \times 1/3 + 7 \times 1/3 = 5$$

$$x * y = [3, 5, \dots]$$

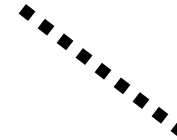
Convolution - a simple 2D illustration

[illegible]

3X3 filter

-1	0	1
-2	0	2
-1	0	1

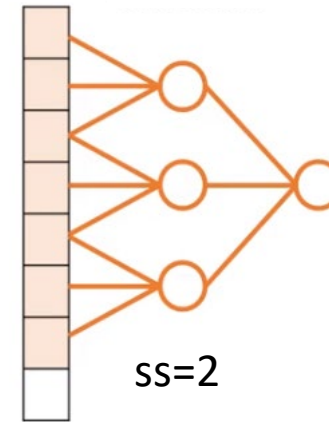
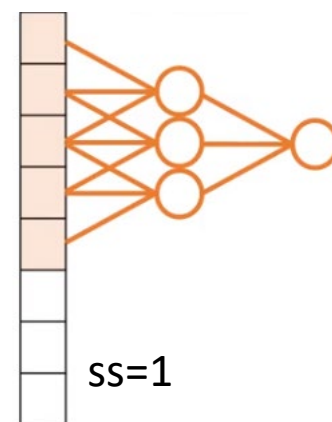
0	0	4	4	0	0
0	0	4			

[illegible]

Define padding & stride size by illustration

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

padding



stride

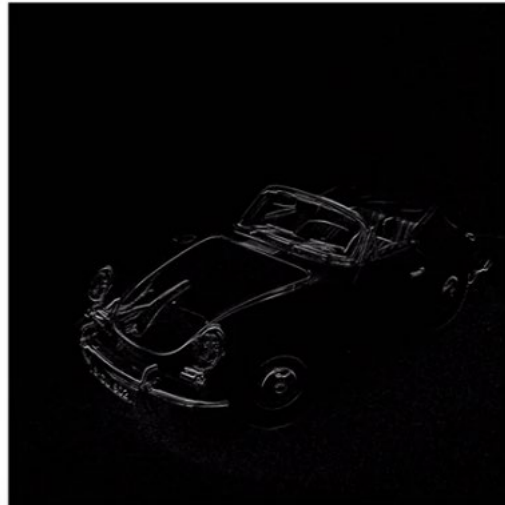
Outputs of the 2 convolution kernels/filters



-1	0	1
-2	0	2
-1	0	1



-1	-2	-1
0	0	0
1	2	1

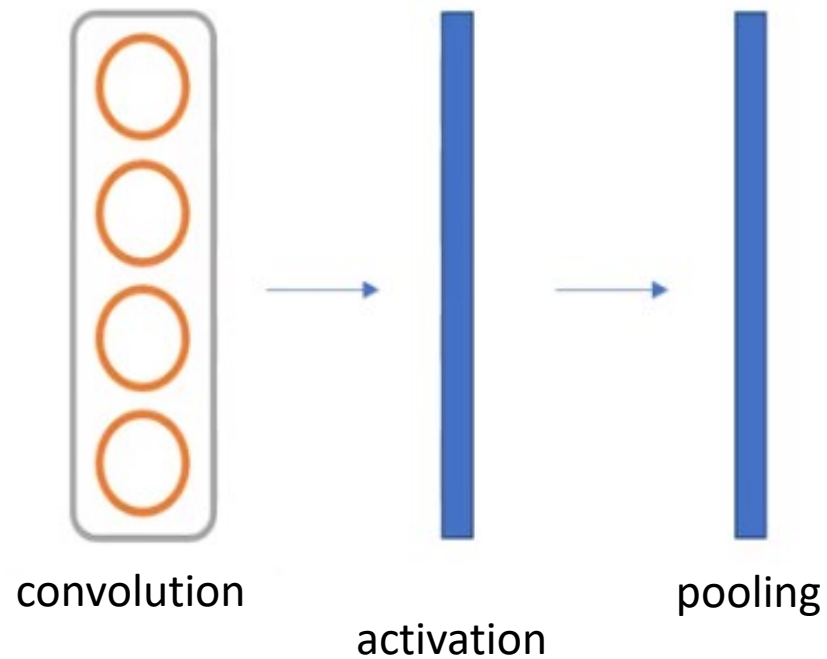


RGB – 3 filters



?	?	?
?	?	?
?	?	?

Max pooling/Average pooling by illustration



0	0	5	1
0	2	4	0
1	0	1	0
0	0	0	3

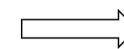
Max pooling

(Invariant to small translation)

2	5
1	3

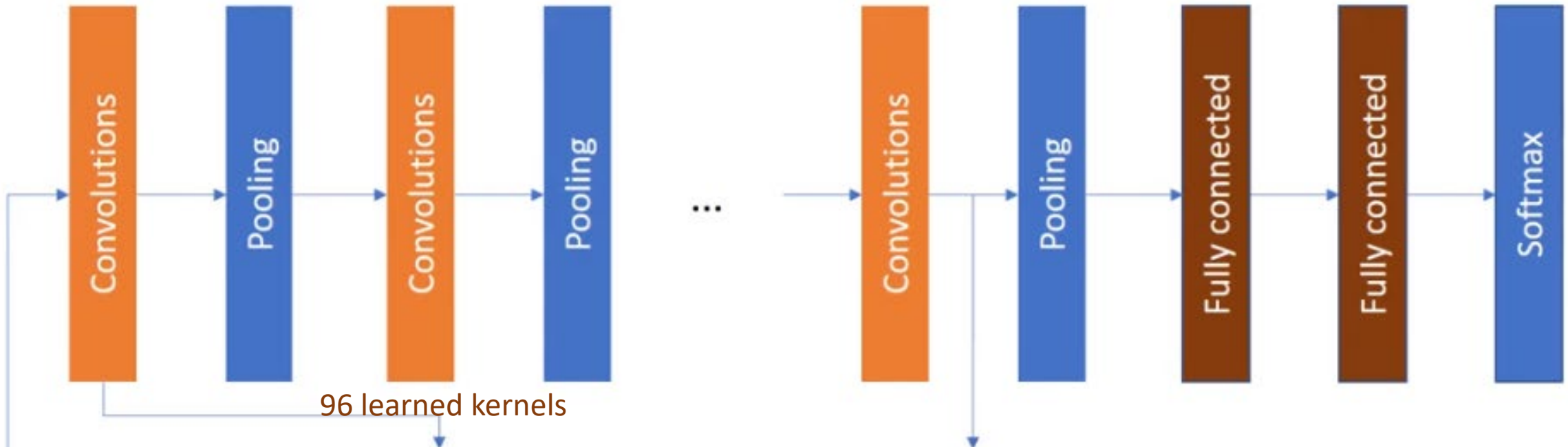
Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2



3.75	1.25
4	2

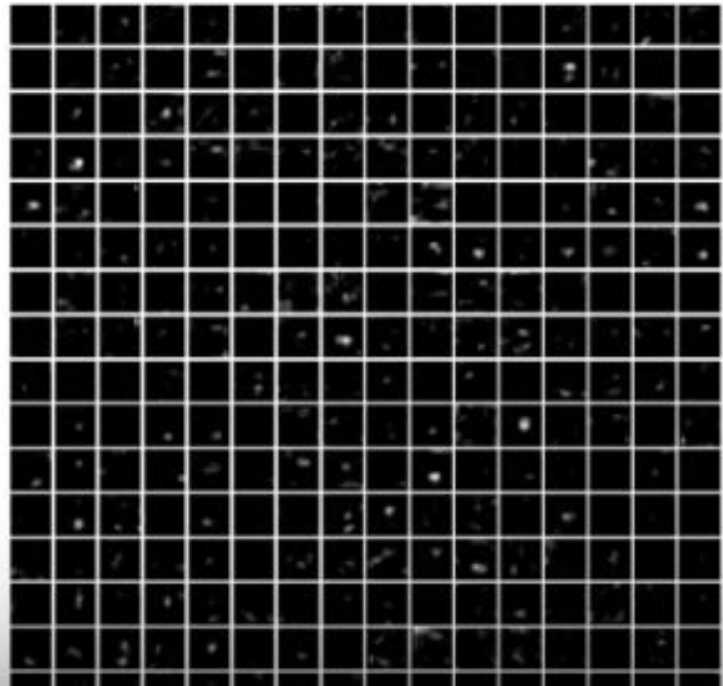
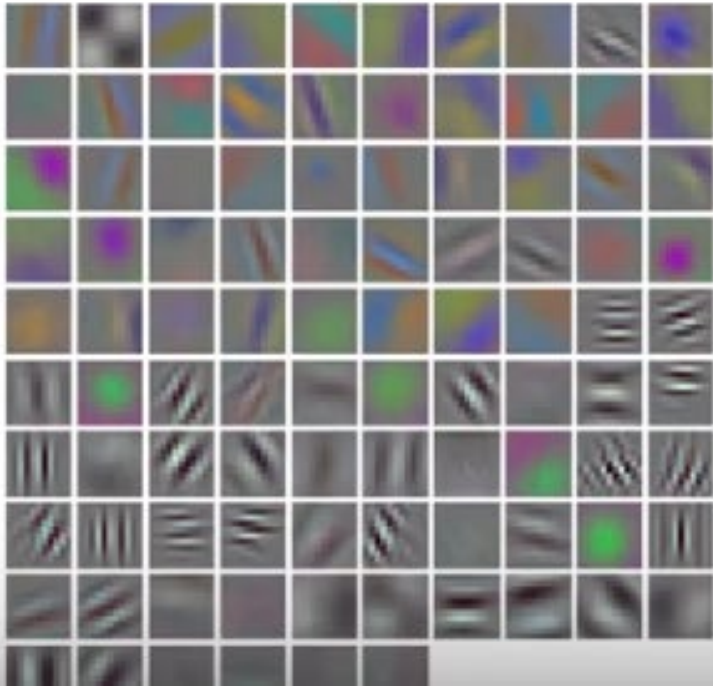
How to explain what's going on in deep CNN – well, it is difficult



96 learned kernels



Sensitive to edges (GPU1), color contrast (GPU2)...(primitive)



Dots - abstract, specific concepts (faces, print text...)

Some insights...

- ReLUs in CNN makes training faster
- Parallel computation on GPUs with improved efficiency
- Prevent overfitting by dropout – randomly drop out a node with a given probability (hyperparameter), “ensemble learning” less feasible for deep NN
- Prevent overfitting by data augmentation (but sometimes not obvious how to do it)
- Depth in deep networks matters, bigger networks and more data tend to result in robust models
- Deep network size limited by memory on GPUs
- Training time depends on how much a designer is willing to wait
- Data/activity processing (AlexNet as an example): mean removal for each pixel in training set, local activity normalization prior to applying ReLUs
- Weight initialization (e.g., 0 mean 0.01 SD Gaussian distribution), taking into consideration of positive input to ReLUs

Hyperparameters

Architecture

Depth/blocks of network

Number of filters/kernels

Filter size

Stride size

Padding

Pooling size & type

Batch size

Dropout rate

Learning rate

Number of epochs

Parameters associated w/ normalization

...

Tuning hyperparameters (grid search)

- Select a small subset of the dataset to perform cross validation to be used for determining hyperparameters.
- Begin with a coarse search. Zoom into finer grained search once a potential grid region is identified.
- To perform k-fold cross validation, divide data into k (e.g., 3, 5, 7, 10) non-overlapping sections, leave a section(s) for validation, average the validation results of all folds. Carefully work around the cross validation setup and try to stabilize results.

How gradients are determined in max pooling

0	55	0	0
20	0	41	33
0	90	0	0
0	57	0	95



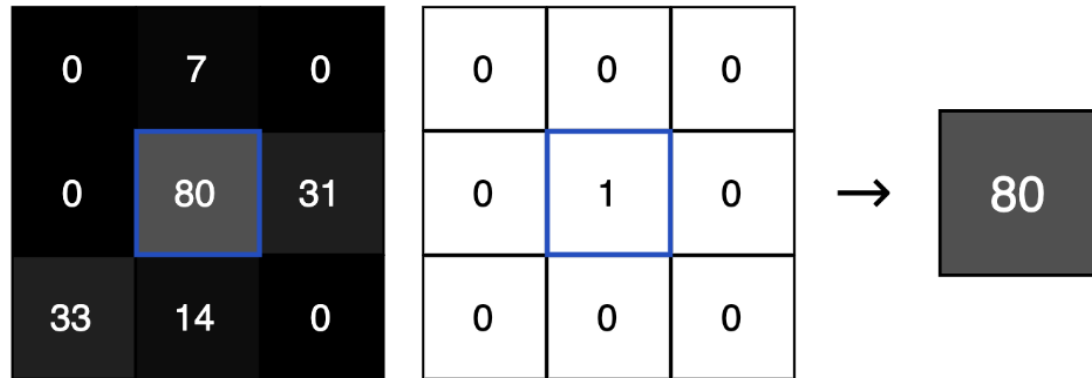
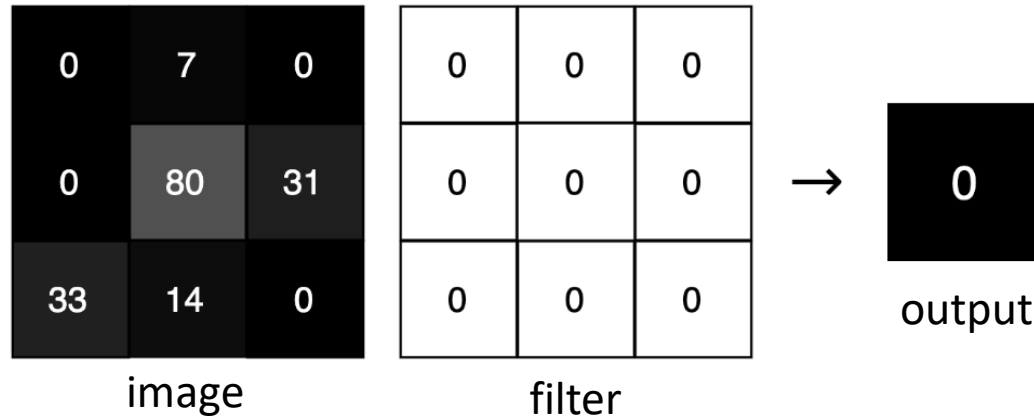
55	41
90	95

Forward path for info flow



Backprop gradient: each gradient value is assigned to where the original max value was, and every other value is zero

How gradients are determined in convolution filter



Notice that - increasing any of the other filter weights by 1 would increase the output by the value of the corresponding image pixel! This suggests that the derivative of a specific output pixel with respect to a specific filter weight is just the corresponding image pixel value.

How gradients are determined in convolution filter (using equations)

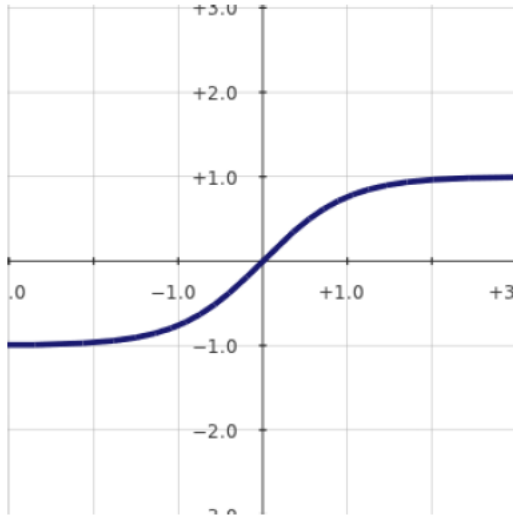
$$\text{out}(i, j) = \text{convolve}(\text{image}, \text{filter})$$

$$= \sum_{x=0}^3 \sum_{y=0}^3 \text{image}(i + x, j + y) * \text{filter}(x, y)$$

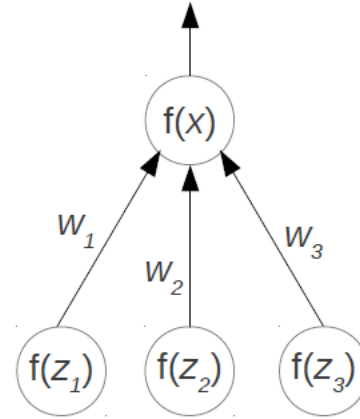
$$\frac{\partial \text{out}(i, j)}{\partial \text{filter}(x, y)} = \text{image}(i + x, j + y)$$

Neurons

$$f(x) = \tanh(x)$$



Very bad (slow to train)



$$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$$

x is called the total input
to the neuron, and $f(x)$
is its output

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



Very good (quick to train)

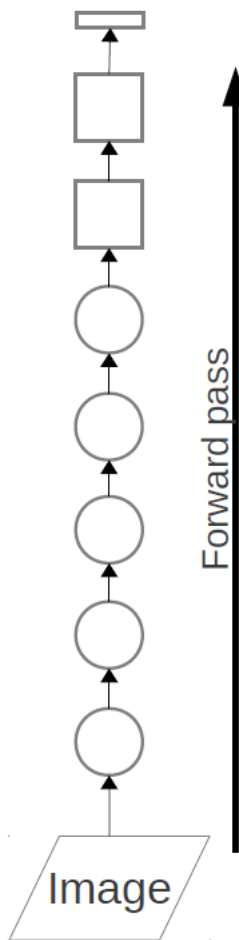
Insight from 2012 AlexNet paper

Training



Local convolutional filters

Fully-connected filters



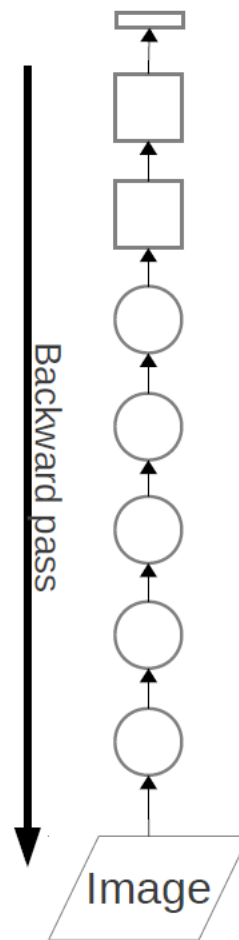
Using stochastic gradient descent and the *backpropagation algorithm* (just repeated application of the chain rule)

One output unit per class

x_i = total input to output unit i

$$f(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{1000} \exp(x_j)}$$

We maximize the log-probability of the correct label, $\log f(x_t)$



Optimizers – using momentum

Recall the backpropagation rule for weight adjustment between the i -th input node and the j -th output node, where η is the learning rate, $\delta_j(n)$ is the local gradient, and $y_i(n)$ is the input

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

With momentum (e.g., using previous gradient):

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \quad 0 \leq |\alpha| < 1$$

- If $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ at n and $n-1$ steps have the same sign, momentum accelerates $\Delta w_{ji}(n)$
- If $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$ at n and $n-1$ steps have opposite sign, momentum slows down $\Delta w_{ji}(n)$

Optimizers – adaptive methods

AdaGrad – how to reduce individual learning rate for individual weight
if gradient is large, reduce learning rate faster
if gradient is small, reduce learning rate slower

RMSProp – for individual weight, also tries to dampen the oscillations like momentum, uses a heuristic (related to $\sqrt{\text{gradient}^2}$, or moving average of gradients)

Adam – combines the heuristics of momentum and RMSProp