Deep Learning

Lecture 3: Convolutional networks

Gilles Louppe
g.louppe@uliege.be

LIÈGE université

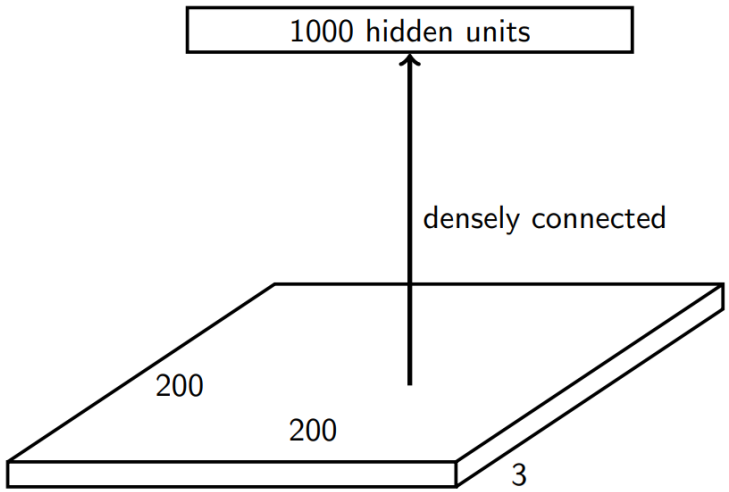
Convolutional networks

Cooking recipe

- Get data (loads of them).
- Get good hardware.
- Define the neural network architecture as a composition of differentiable functions.
  - Stick to non-saturating activation function to avoid vanishing gradients.
  - Prefer deep over shallow architectures.
  - In this lecture, we augment our set of differentiable functions with components tailored for spatial data (images, sequences, etc).
- Optimize with (variants of) stochastic gradient descent.
  - Evaluate gradients with automatic differentiation.

Motivation

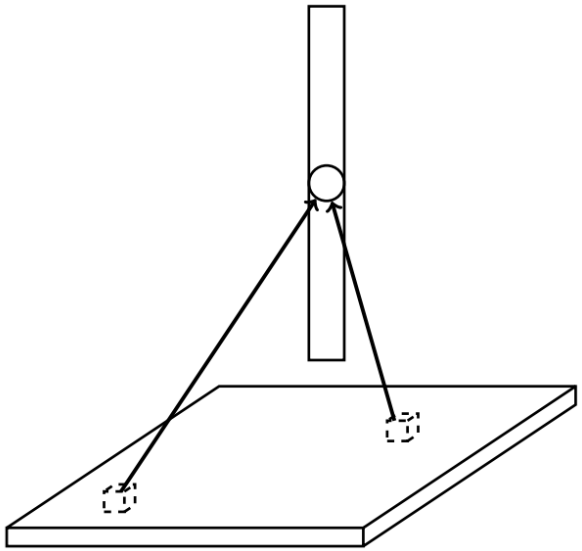
Suppose we want to train a fully connected network that takes 200×200 RGB images as input.



What are the problems with this first layer?

- Too many parameters: $200 \times 200 \times 3 \times 1000 = 120M$.
- What if the object in the image shifts a little?

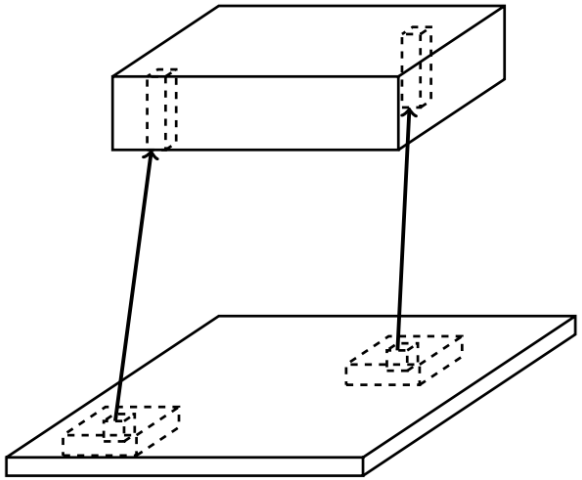
Fully connected layer



In a fully connected layer, each hidden unit $h_j = \sigma(\mathbf{w}_j^T \mathbf{x} + b_j)$ is connected to the entire image.

- Looking for activations that depend on pixels that are spatially far away is supposedly a waste of time and resources.
- Long range correlations can be dealt with in the higher layers.

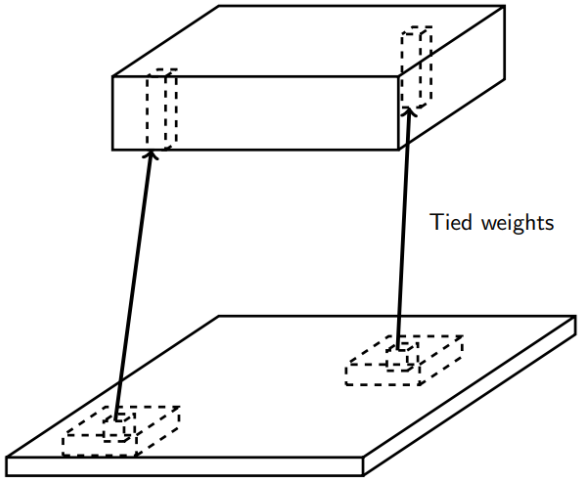
Locally connected layer



In a locally connected layer, each hidden unit h_j is connected to only a patch of the image.

- Weights are specialized locally and functionally.
- Reduce the number of parameters.
- What if the object in the image shifts a little?

Convolutional layer



In a convolutional layer, each hidden unit h_j is connected to only a patch of the image, and share its weights with the other units h_i.

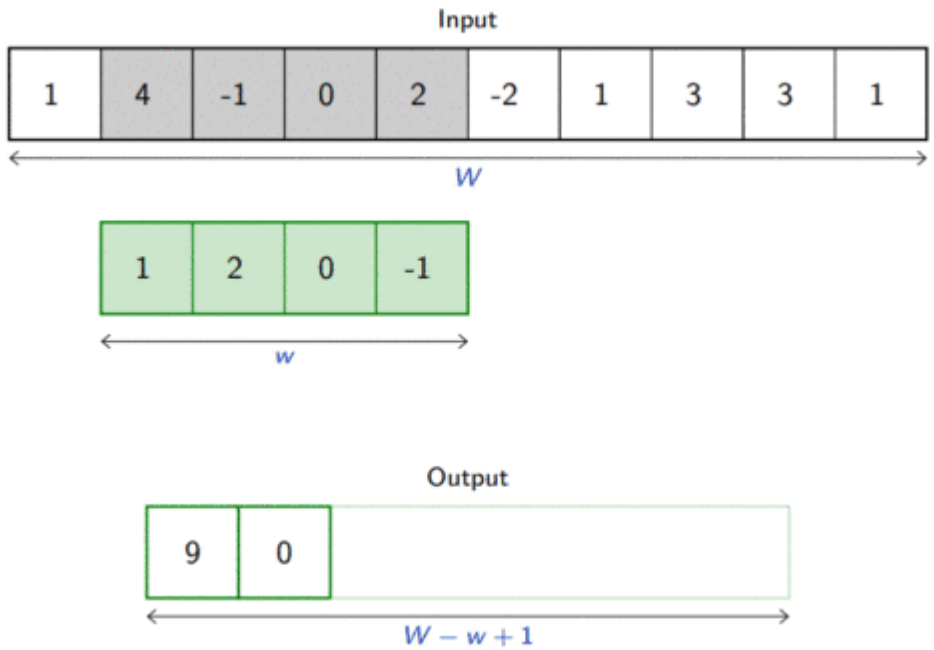
- Weights are specialized functionally, regardless of spatial location.
- Reduce the number of parameters.

Convolution

For one-dimensional tensors, given an input vector $\mathbf{x} \in \mathbb{R}^W$ and a convolutional kernel $\mathbf{u} \in \mathbb{R}^w$, the discrete convolution $\mathbf{x} \star \mathbf{u}$ is a vector of size $W - w + 1$ such that

$$(\mathbf{x} \star \mathbf{u})_i = \sum_{m=0}^{w-1} x_{i+m} u_m.$$

Technically, $\star$ denotes the cross-correlation operator. However, most machine learning libraries call it convolution.



Credits: Francois Fleuret, EE559 Deep Learning, EPFL, 2018.

Convolutions generalize to multi-dimensional tensors:

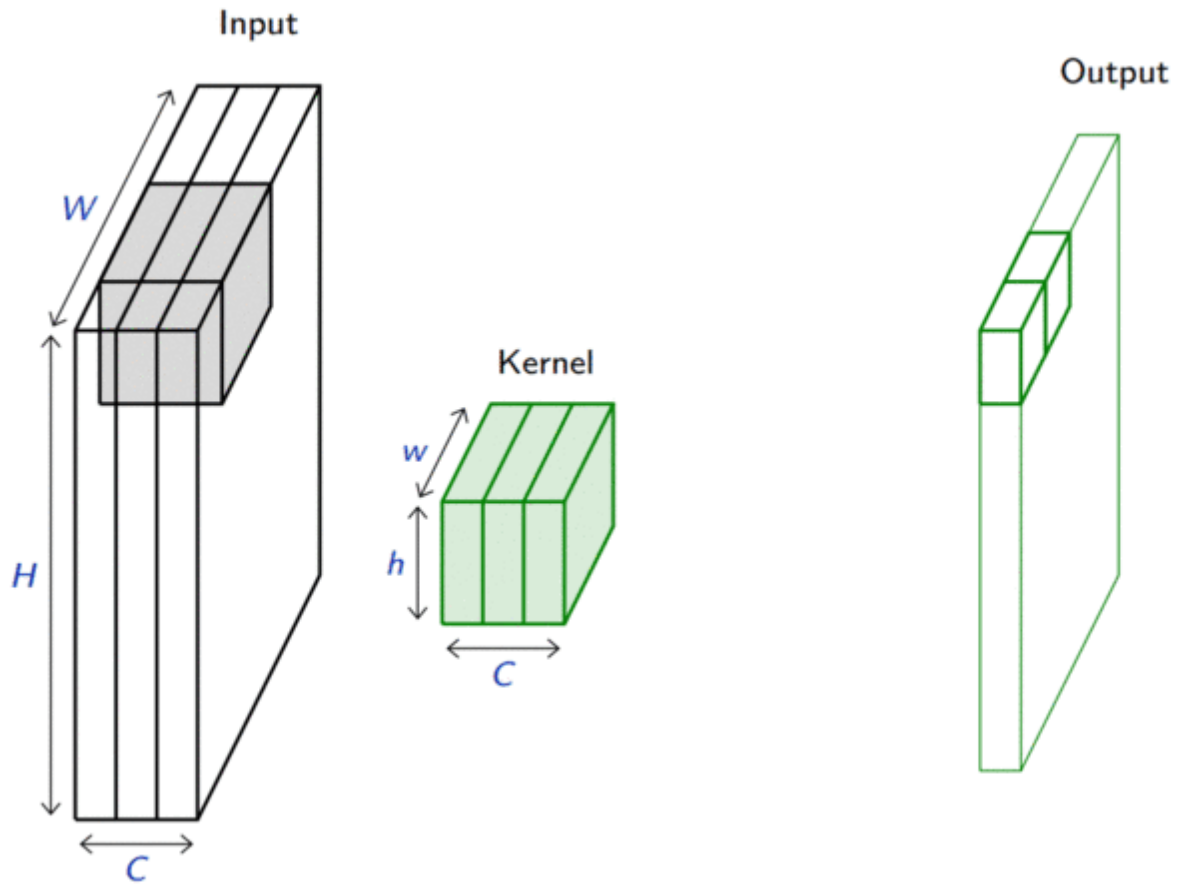
- In its most usual form, a convolution takes as input a 3D tensor $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$, called the input feature map.
- A kernel $\mathbf{u} \in \mathbb{R}^{C \times h \times w}$ slides across the input feature map, along its height and width. The size $h \times w$ is called the receptive field.
- At each location, the element-wise product between the kernel and the input elements it overlaps is computed and the results are summed up.

- The final output $\mathbf{o}$ is a 2D tensor of size $(H - h + 1) \times (W - w + 1)$ called the output feature map and such that:

$$\mathbf{o}_{j,i} = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1} (\mathbf{x}_c \star \mathbf{u}_c)_{j,i} = \mathbf{b}_{j,i} + \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,j+n,i+m} \mathbf{u}_{c,n,m}$$

  where $\mathbf{u}$ and $\mathbf{b}$ are shared parameters to learn.

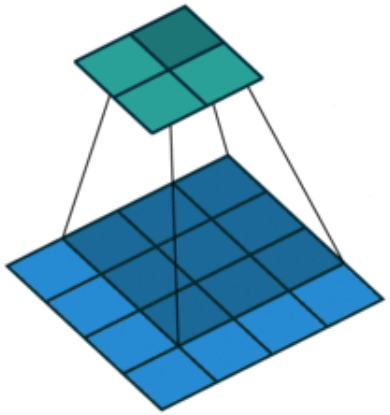
- D convolutions can be applied in the same way to produce a $D \times (H - h + 1) \times (W - w + 1)$ feature map, where D is the depth.



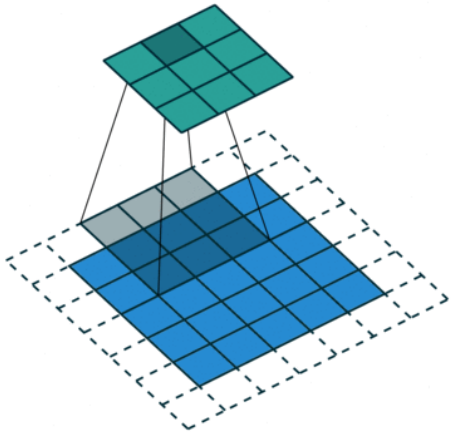
Credits: Francois Fleuret, EE559 Deep Learning, EPFL, 2018.

In addition to the depth D, the size of the output feature map can be controlled through the stride and with zero-padding.

- The stride S specifies the size of the step the kernel slides with.
- Zero-padding specifies whether the input volume is pad with zeroes around the border. This makes it possible to produce an output volume of the same size as the input. A hyper-parameter P controls the amount of padding.



No padding ($P = 0$),
no strides ($S = 1$).



Padding ($P = 1$),
strides ($S = 2$).

Equivariance

A function f is equivariant to g if $f(g(\mathbf{x})) = g(f(\mathbf{x}))$.

- Parameter sharing used in a convolutional layer causes the layer to be equivariant to translation.
- That is, if g is any function that translates the input, the convolution function is equivariant to g.
    - E.g., if we move an object in the image, its representation will move the same amount in the output.
- This property is useful when we know some local function is useful everywhere (e.g., edge detectors).
- However, convolutions are not equivariant to other operations such as change in scale or rotation.

Pooling

When the input volume is large, pooling layers can be used to reduce the input dimension while preserving its global structure, in a way similar to a down-scaling operation.

Consider a pooling area of size $h \times w$ and a 3D input tensor $\mathbf{x} \in \mathbb{R}^{C \times (rh) \times (sw)}$.

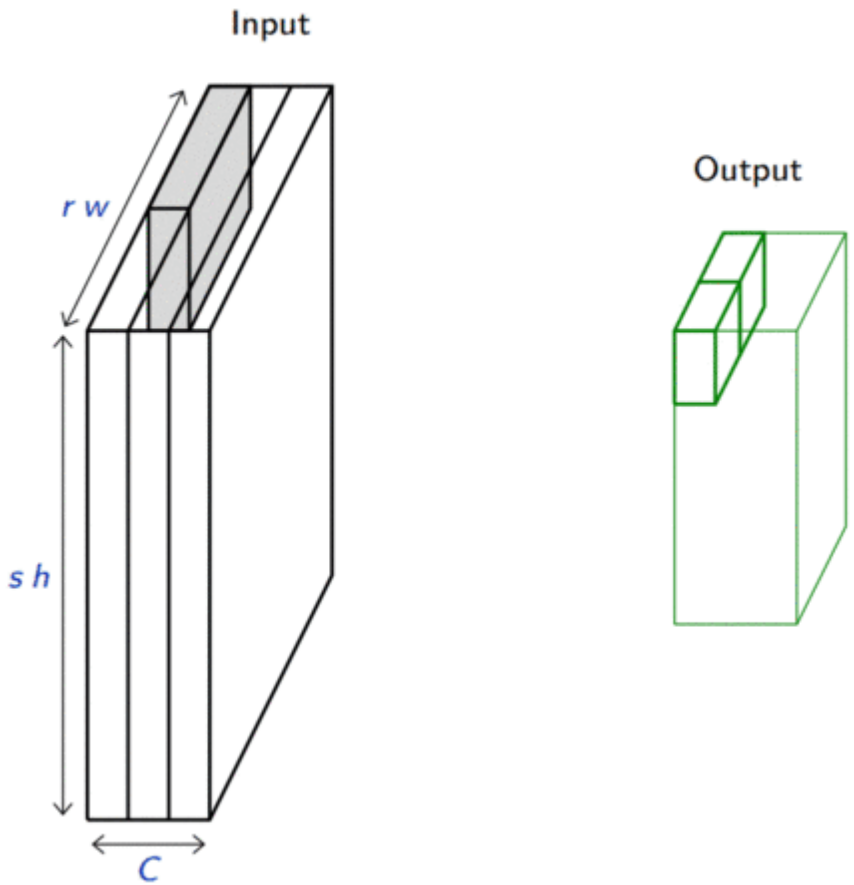
- Max-pooling produces a tensor $\mathbf{o} \in \mathbb{R}^{C \times r \times s}$ such that

$$\mathbf{o}_{c,j,i} = \max_{n<h, m<w} \mathbf{x}_{c,rj+n,si+m}.$$

- Average pooling produces a tensor $\mathbf{o} \in \mathbb{R}^{C \times r \times s}$ such that

$$\mathbf{o}_{c,j,i} = \frac{1}{hw} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,rj+n,si+m}.$$

Pooling is very similar in its formulation to convolution. Similarly, a pooling layer can be parameterized in terms of its receptive field, a stride S and a zero-padding P.



Credits: Francois Fleuret, EE559 Deep Learning, EPFL, 2018.

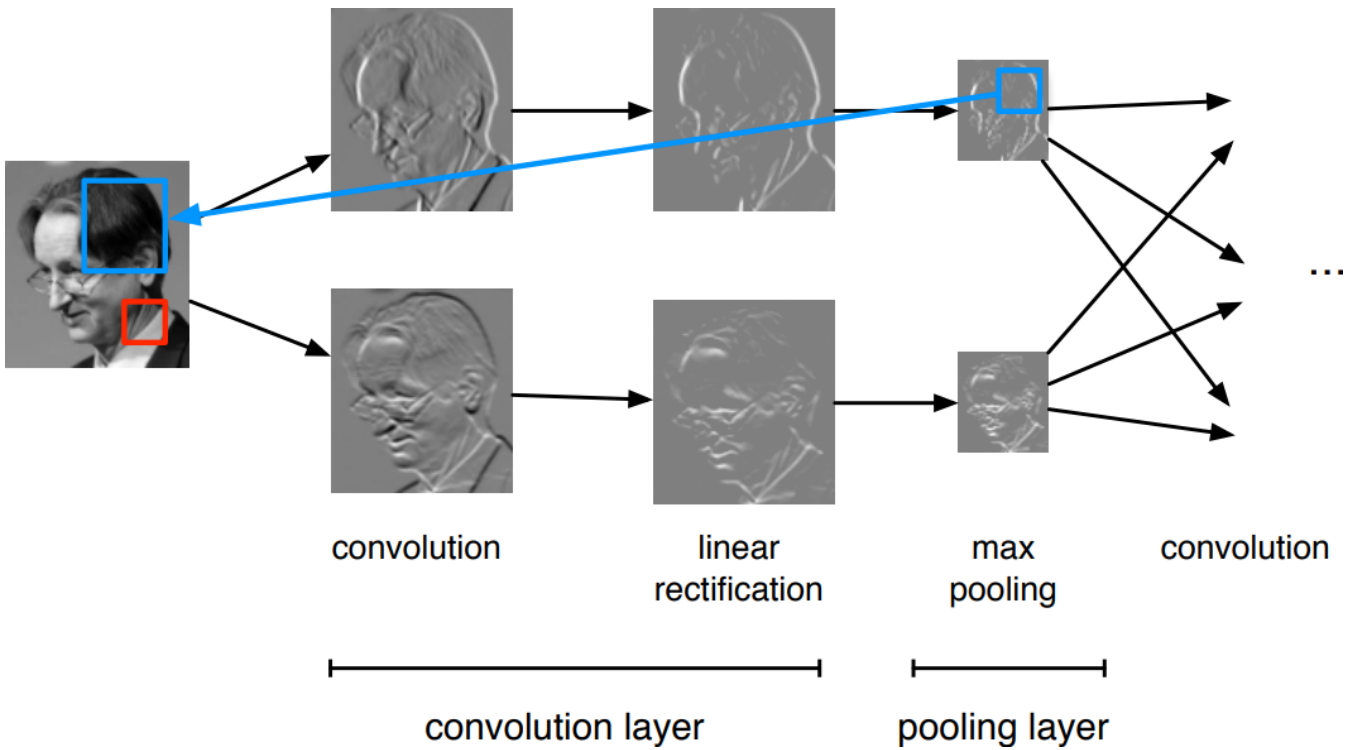
Invariance

A function f is invariant to g if $f(g(\mathbf{x})) = f(\mathbf{x})$.

- Pooling layers can be used for building inner activations that are (slightly) invariant to small translations of the input.
- Invariance to local translation is helpful if we care more about the presence of a pattern rather than its exact position.

Layer patterns

A convolutional network can often be defined as a composition of convolutional layers (`CONV`), pooling layers (`POOL`), linear rectifiers (`RELU`) and fully connected layers (`FC`).



The most common convolutional network architecture follows the pattern:

$$\texttt{INPUT} \rightarrow [[\texttt{CONV} \rightarrow \texttt{RELU}]*N \rightarrow \texttt{POOL?}]*M \rightarrow [\texttt{FC} \rightarrow \texttt{RELU}]*K \rightarrow \texttt{FC}$$

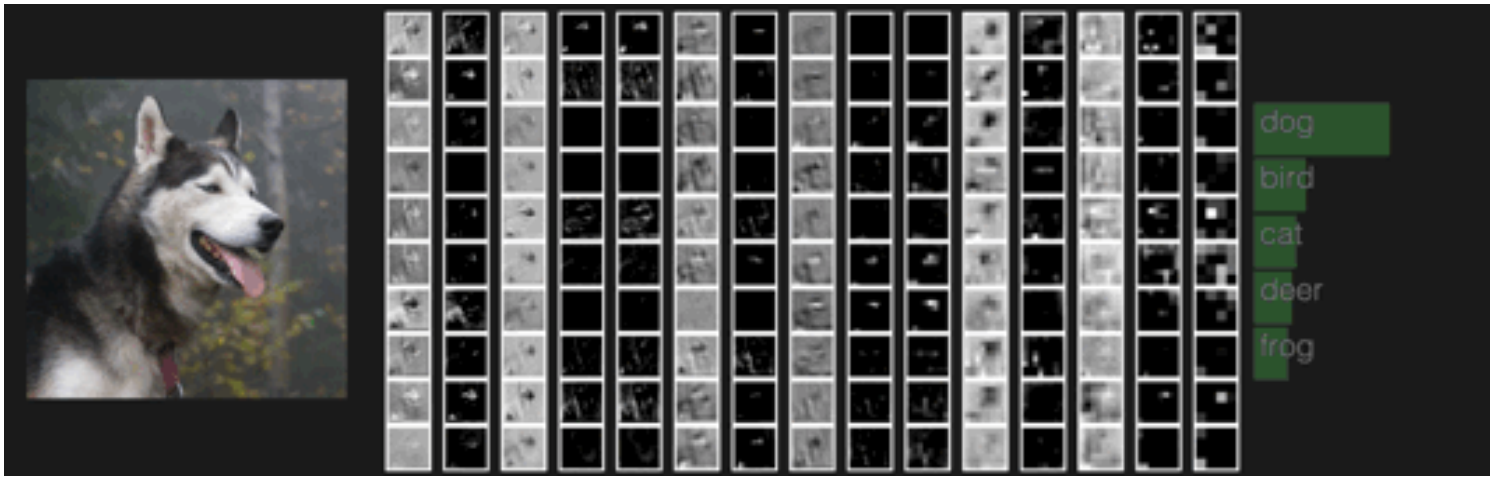
where:

- $*$ indicates repetition;
- $\texttt{POOL?}$ indicates an optional pooling layer;
- $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$);
- the last fully connected layer holds the output (e.g., the class scores).

Some common architectures for convolutional networks following this pattern include:

- $\texttt{INPUT} \rightarrow \texttt{FC}$, which implements a linear classifier ($N = M = K = 0$).
- $\texttt{INPUT} \rightarrow \left[\texttt{FC} \rightarrow \texttt{RELU}\right]*K \rightarrow \texttt{FC}$, which implements a K-layer MLP.
- $\texttt{INPUT} \rightarrow \texttt{CONV} \rightarrow \texttt{RELU} \rightarrow \texttt{FC}$.
- $\texttt{INPUT} \rightarrow \left[\texttt{CONV} \rightarrow \texttt{RELU} \rightarrow \texttt{POOL}\right]*2 \rightarrow \texttt{FC} \rightarrow \texttt{RELU} \rightarrow \texttt{FC}$.
- $\texttt{INPUT} \rightarrow \left[\left[\texttt{CONV} \rightarrow \texttt{RELU}\right]*2 \rightarrow \texttt{POOL}\right]*3 \rightarrow \left[\texttt{FC} \rightarrow \texttt{RELU}\right]*2 \rightarrow \texttt{FC}$.

Note that for the last architecture, two `CONV` layers are stacked before every `POOL` layer. This is generally a good idea for larger and deeper networks, because multiple stacked `CONV` layers can develop more complex features of the input volume before the destructive pooling operation.


dog
bird
cat
deer
frog

LeNet-1

(LeCun et al, 1993)

LeNet-5

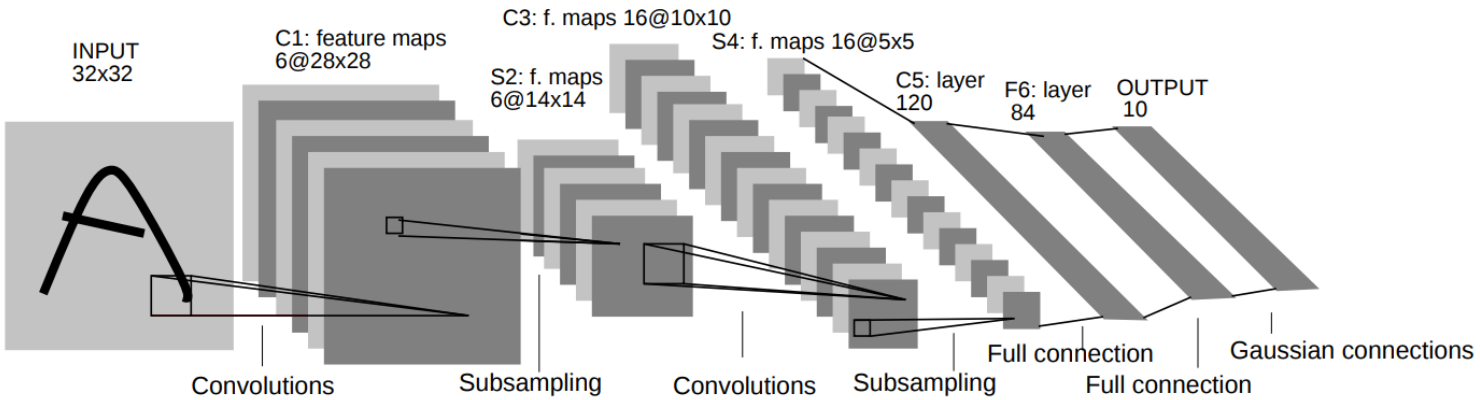


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

(LeCun et al, 1998)

AlexNet

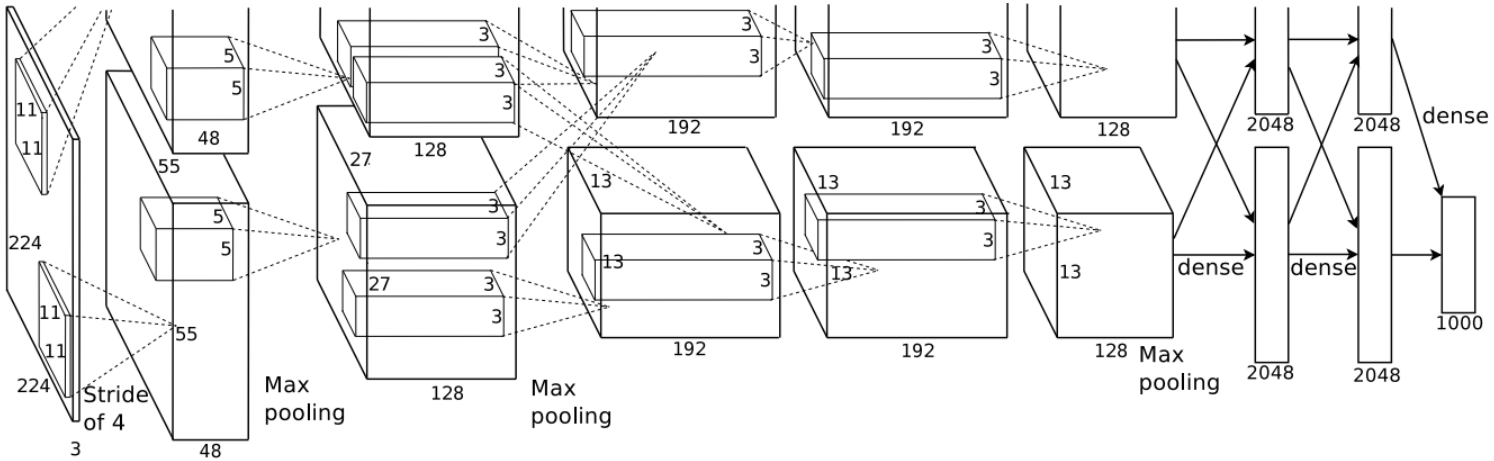


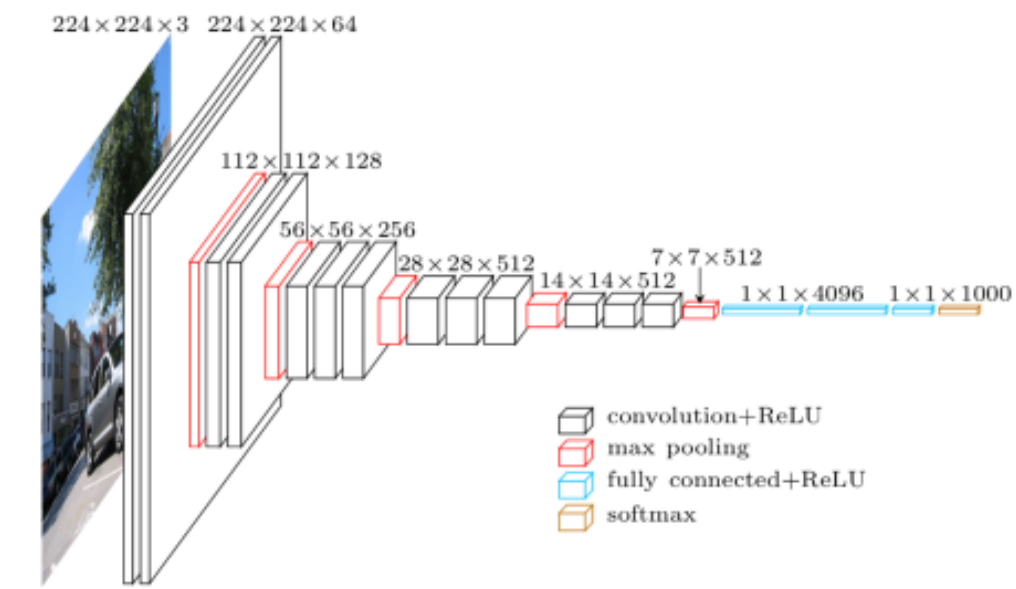
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

(Krizhevsky et al, 2012)

GoogLeNet

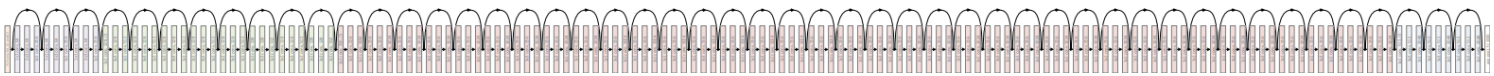
(Szegedy et al, 2014)

VGGNet



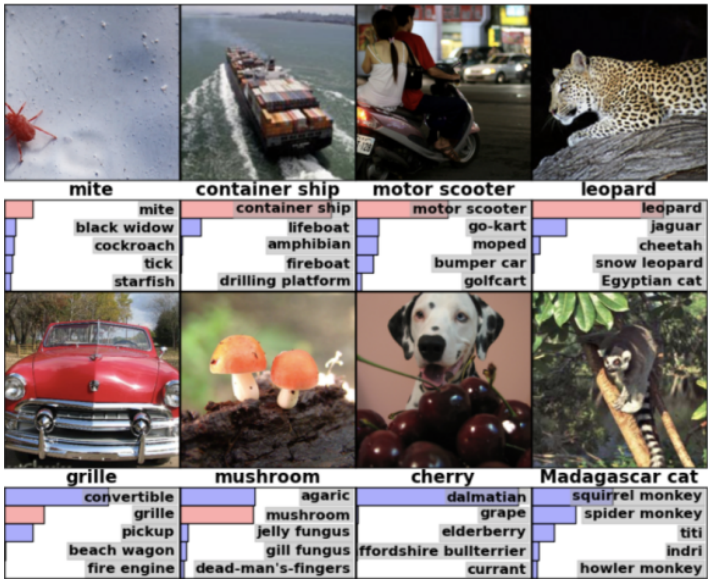
(Simonyan and Zisserman, 2014)

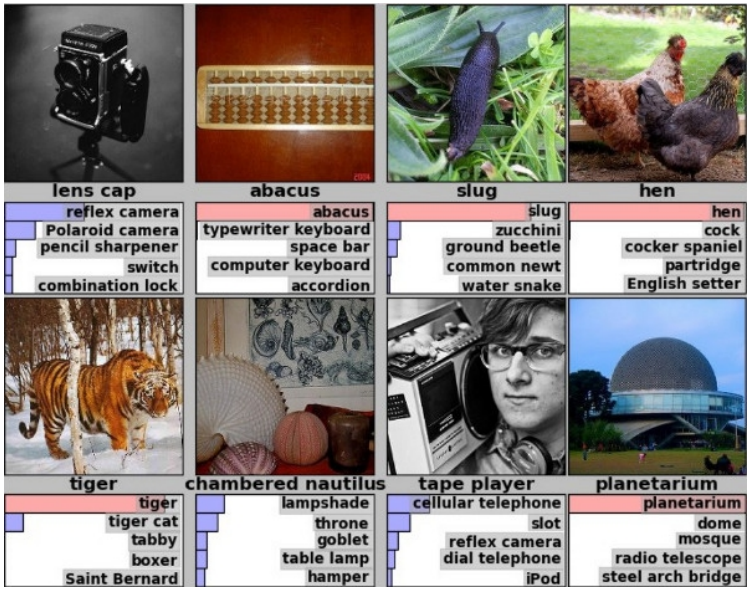
ResNet



(He et al, 2015)

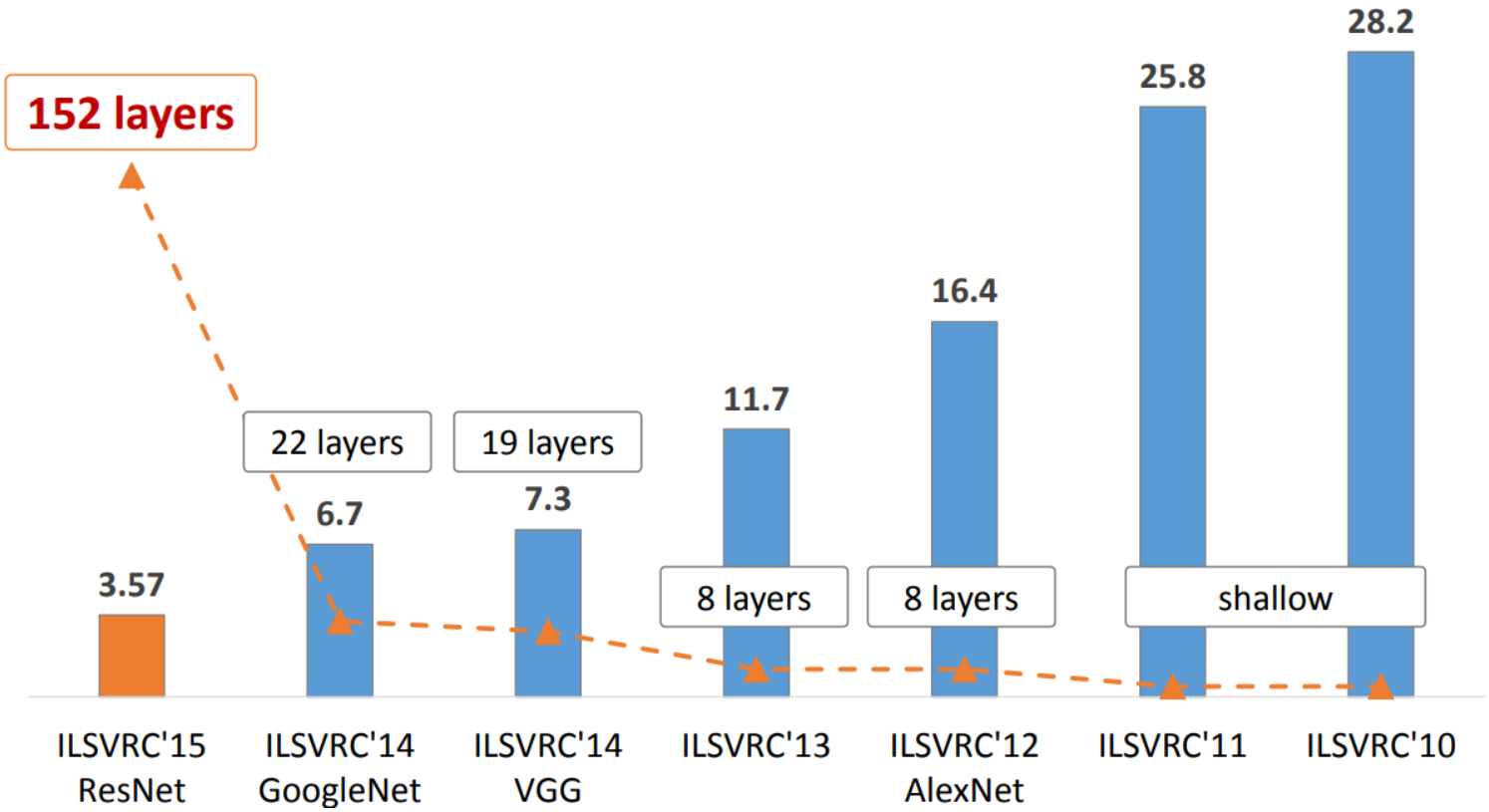
Object recognition





The ImageNet challenge:

- 1000 object classes
- 1200000 training examples
- 100000 test examples


152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

Credits: Keras documentation

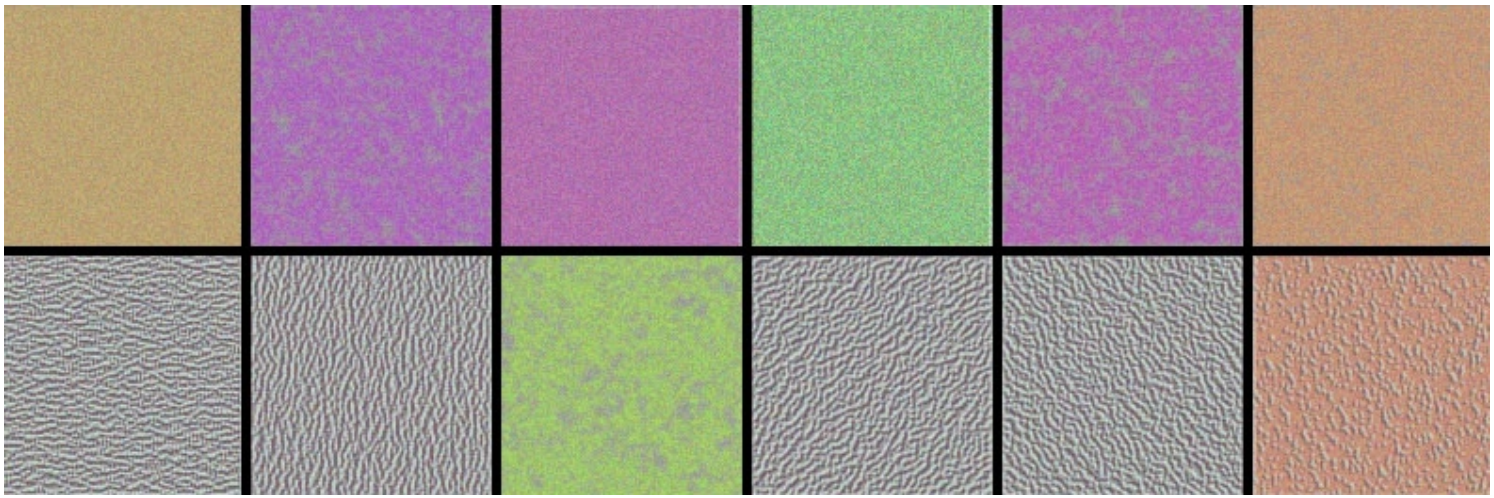
Maximum response samples

What does a convolutional network see?

Convolutional networks can be inspected by looking for input images $\mathbf{x}$ that maximize the activation $\mathbf{h}_{\ell,d}(\mathbf{x})$ of a chosen convolutional kernel $\mathbf{u}$ at layer ℓ and index d in the layer filter bank.

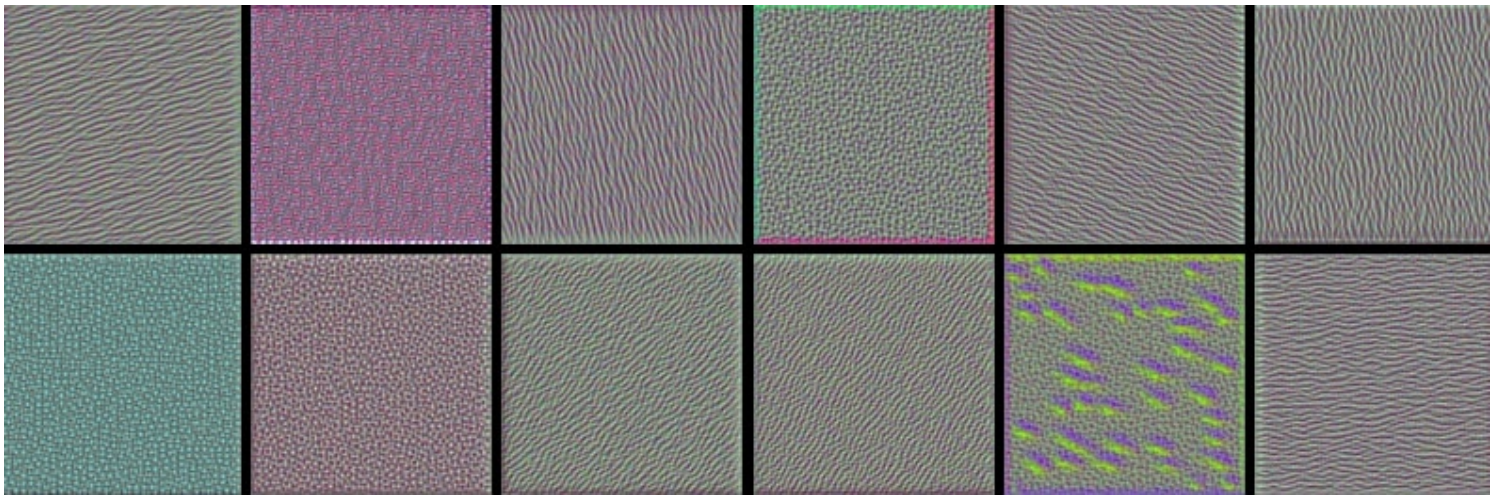
Such images can be found by gradient ascent on the input space:

$$\begin{aligned} \mathcal{L}_{\ell,d}(\mathbf{x}) &= ||\mathbf{h}_{\ell,d}(\mathbf{x})||_2 \\ \mathbf{x}_0 &\sim U[0,1]^{C\times H\times W} \\ \mathbf{x}_{t+1} &= \mathbf{x}_t + \gamma \nabla_{\mathbf{x}} \mathcal{L}_{\ell,d}(\mathbf{x}_t) \end{aligned}$$



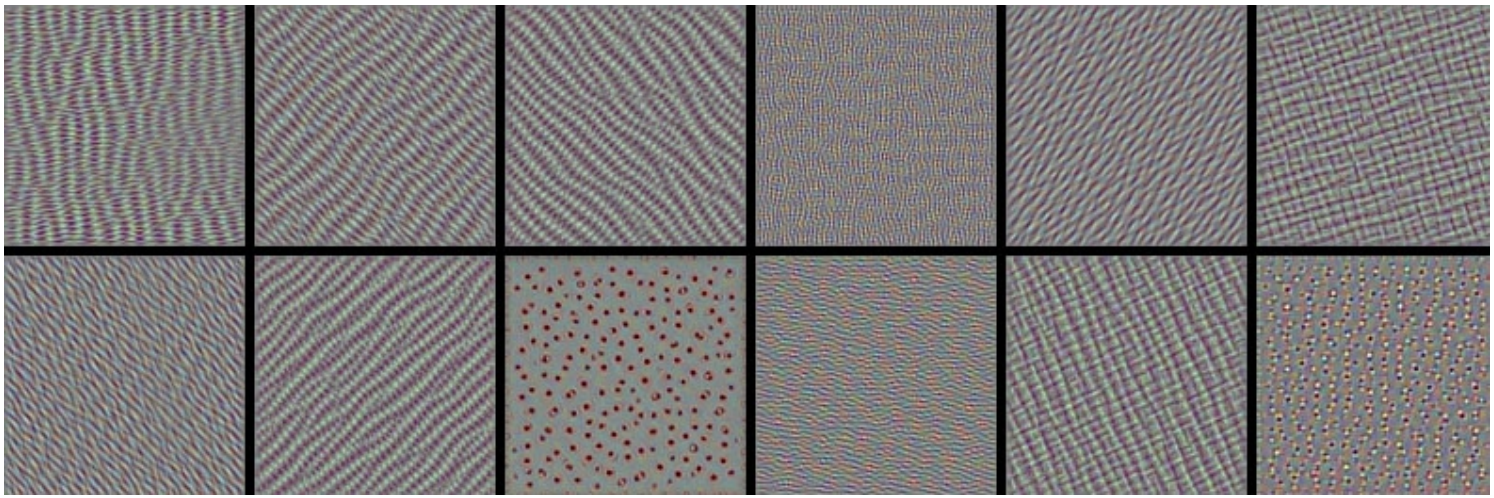
VGG16, convolutional layer 1-1, a few of the 64 filters

Credits: How convolutional neural networks see the world



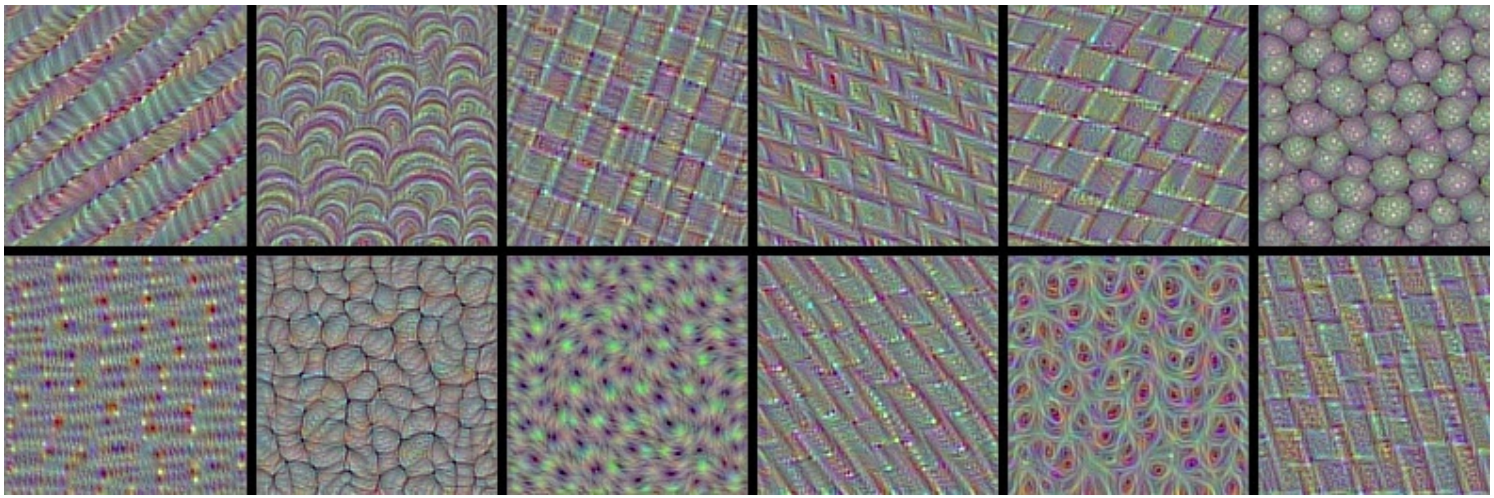
VGG16, convolutional layer 2-1, a few of the 128 filters

Credits: How convolutional neural networks see the world



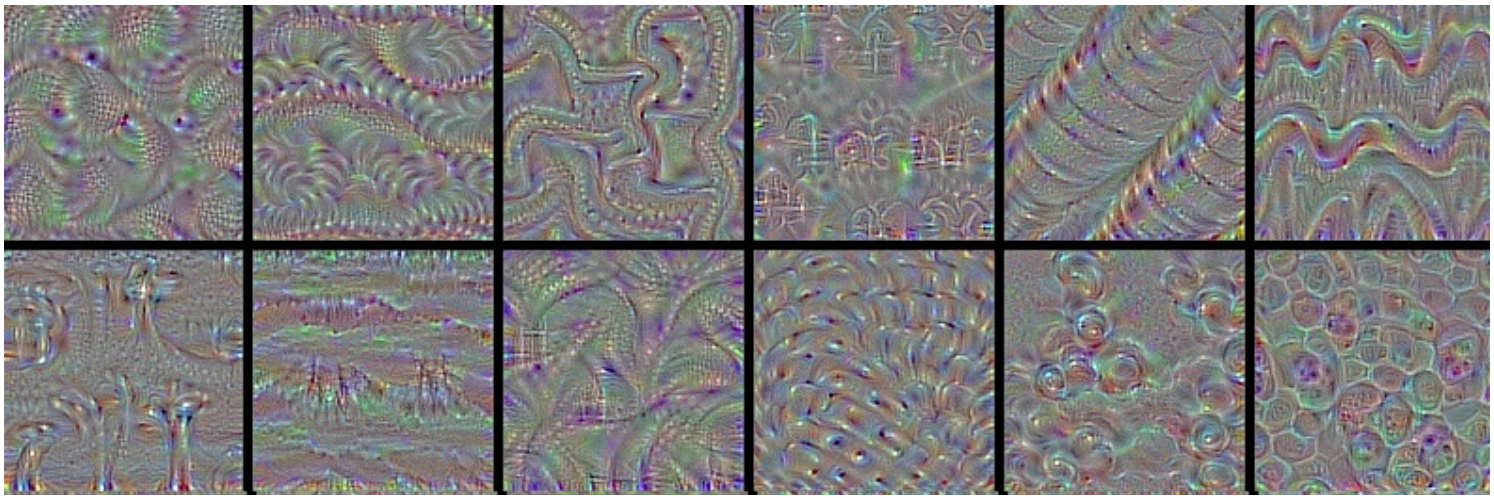
VGG16, convolutional layer 3-1, a few of the 256 filters

Credits: How convolutional neural networks see the world



VGG16, convolutional layer 4-1, a few of the 512 filters

Credits: How convolutional neural networks see the world



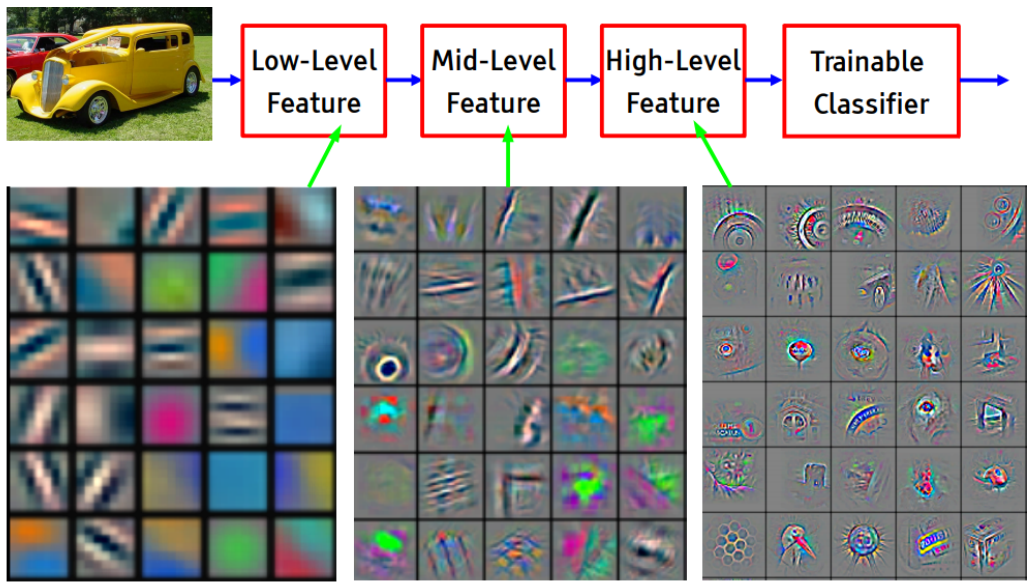
VGG16, convolutional layer 5-1, a few of the 512 filters

Credits: How convolutional neural networks see the world

Some observations:

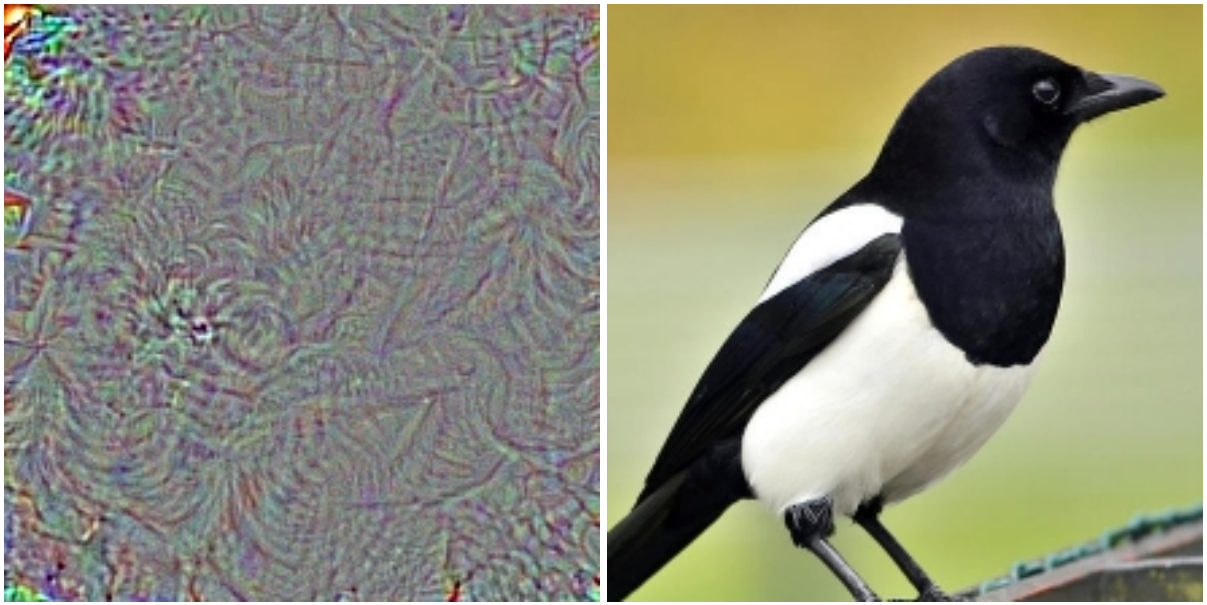
- The first layers appear to encode direction and color.
- The direction and color filters get combined into grid and spot textures.
- These textures gradually get combined into increasingly complex patterns.

In other words, the network appears to learn a hierarchical composition of patterns.



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

What if we build images that maximize the activation of a chosen class output?



The left image is predicted with 99.9% confidence as a magpie.

Credits: How convolutional neural networks see the world

Deep Dream

Start from an image $\mathbf{x}_t$, offset the image by a random jitter, enhance some layer activation at multiple scales, zoom in, repeat on the produced image $\mathbf{x}_{t+1}$.

The end.

References

- CS231n Convolutional networks (Fei-Fei Li et al, Stanford)
- EE-559 Deep learning (Francois Fleuret, EPFL)

Further readings:

- Feature Visualization (Olah et al, 2017)
- The Building Blocks of Interpretability (Olah et al, 2018)