Introduction to Deep Learning

Lecture 4

Convolutional Neural Networks

Maria Vakalopoulou & Stergios Christodoulidis

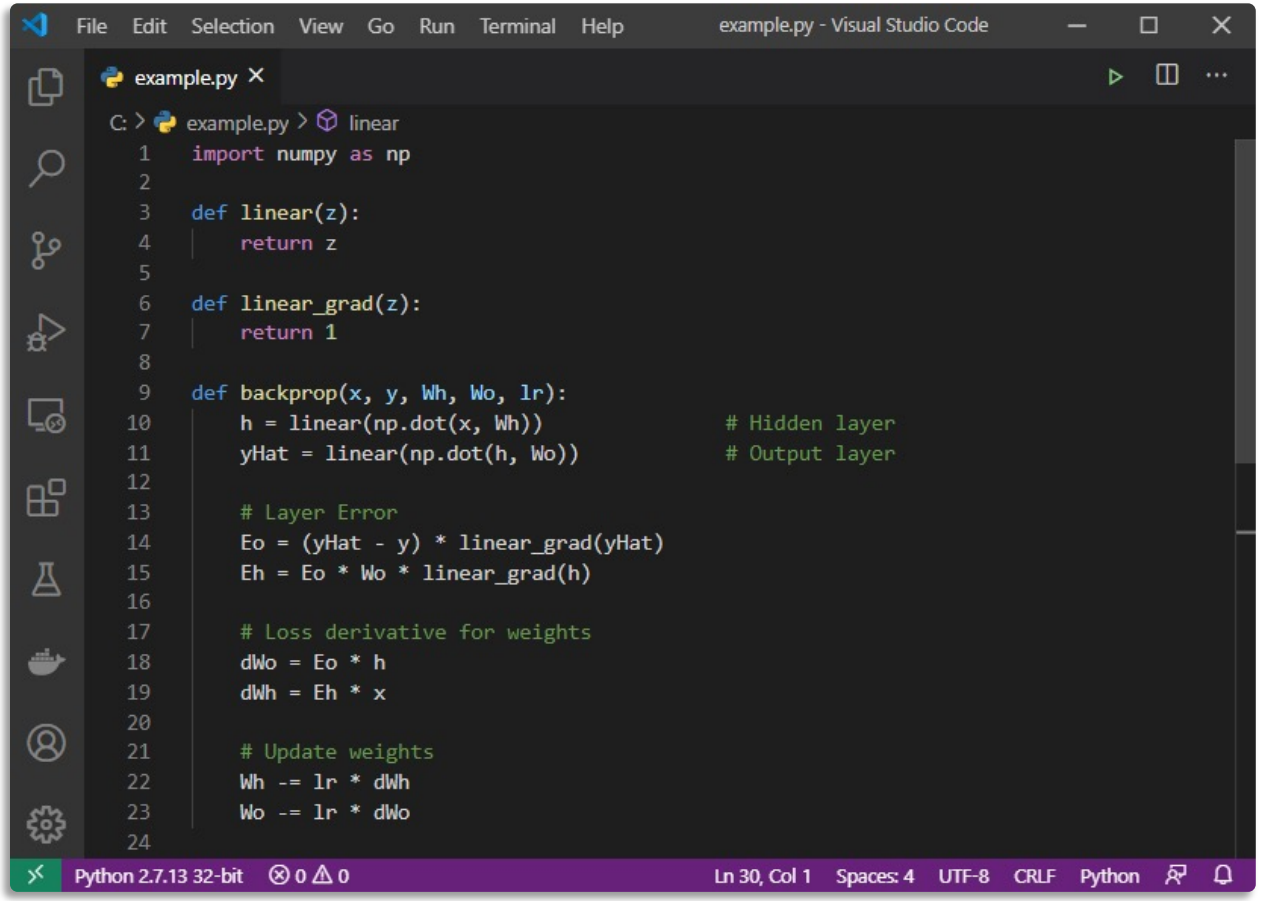
MICS Laboratory

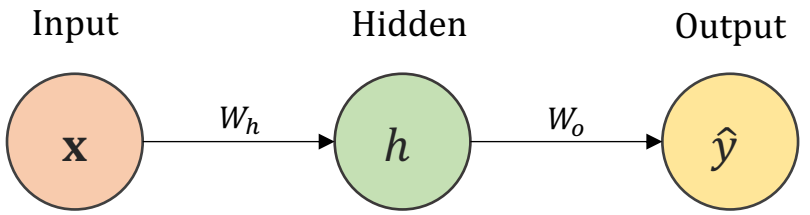
CentraleSupélec
Université Paris-Saclay

Wednesday, November 17, 2020

Last Lectures

Gradient Descent





$$Loss = \mathcal{L}\big(f\,(g(x; W_h); W_o)\big)$$

$$w^{(i+1)} = w^{(i)} - \eta \cdot \frac{d\mathcal{L}}{d\theta}$$

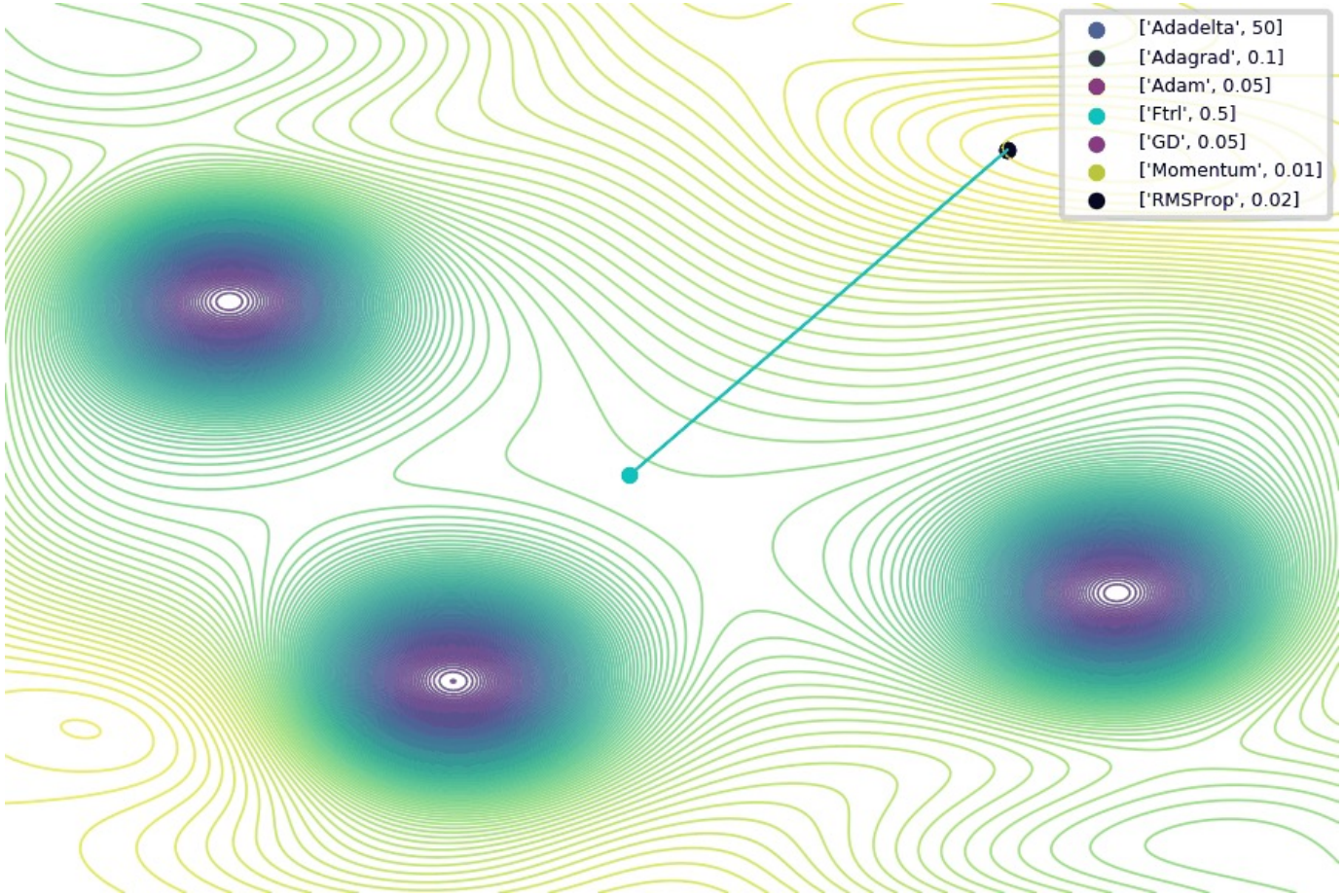
Gradient Descent & Stochastic Gradient Descent

- Gradient Descent:
  - Gradient over the true distribution of the data
  - Computed on all the training samples
  - Might be impossible to capture the true distribution of the data (What is the true distribution of an image representation of a face?)
  - Computational issues

- Stochastic Gradient Descent:
  - Gradient over an approximation of the true gradient
  - Computed on randomly/stochastically selected samples
  - Introduces randomness and helps avoiding overfitting
  - Computational friendly

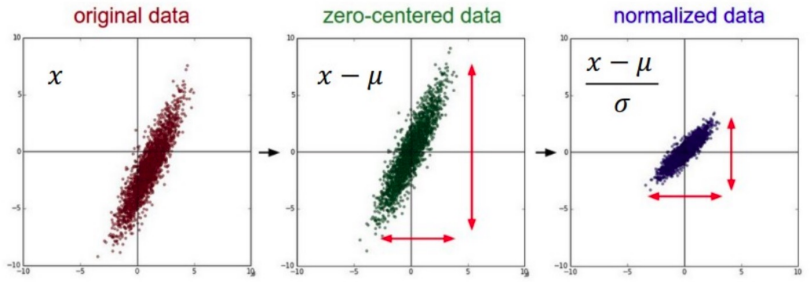
Optimizers

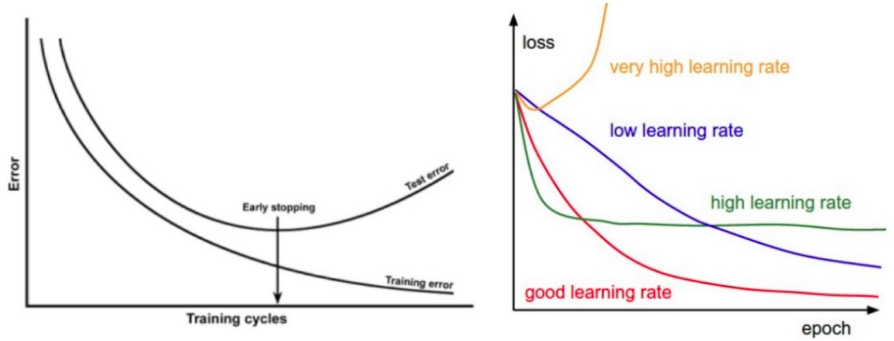
- GD
- SGD with momentum
- RMSprop
- Adam
- Adagrad
- (…)

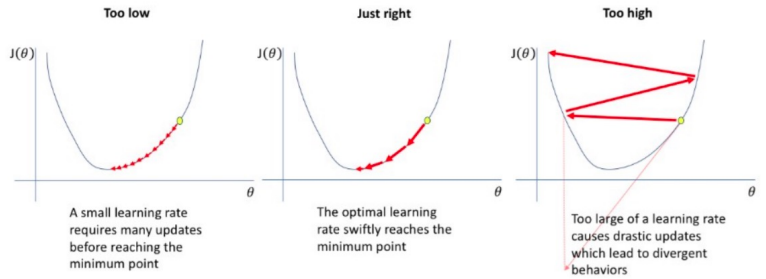


Good Practices for Training

- Input Normalization
- Batch Normalization
- Regularization
- Early Stopping
- Dropout
- Learning Rate Scheduling
- Trainable Parameters Initialization







Today's Lecture

Today's Lecture

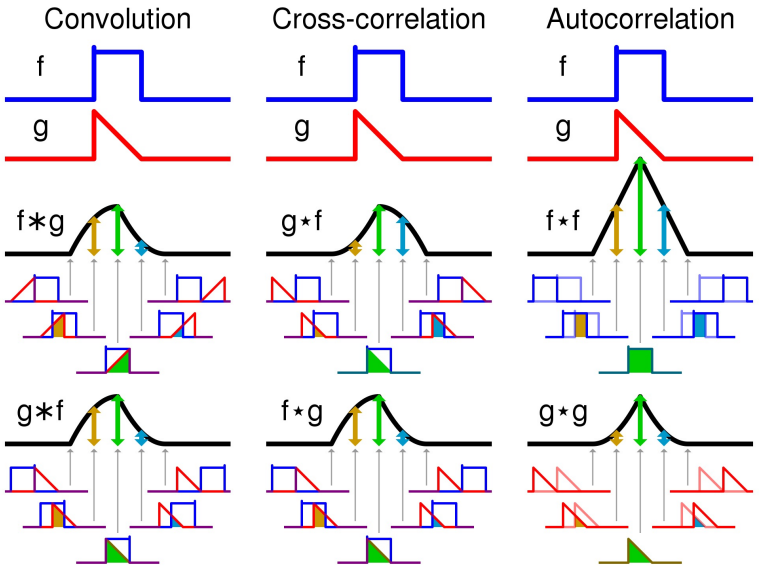
- Convolutions
  - Intuitive explanation of the convolution operation
  - Convolution operation on 2D Images (Examples)
- Convolutional Neural Networks
  - How are they implemented?
- Subsampling/Pooling Modules
  - Receptive Field
- Feature Maps
  - Visualization
- Transfer Learning

Convolutions

Convolutions

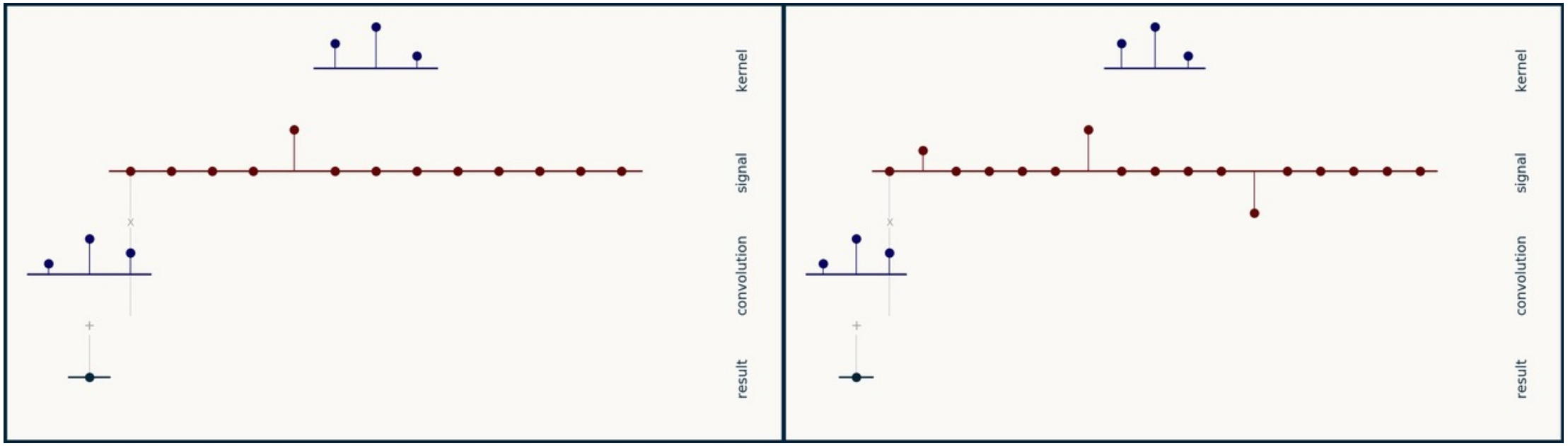
Definition: *The convolution of two functions f (input signal) and g (kernel) is denoted by $*$ and it is defined as the integral of the product of the two functions after one is reversed and shifted.*

$$(f * g)(t) \triangleq \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{+\infty} f(t-\tau)g(\tau)d\tau$$



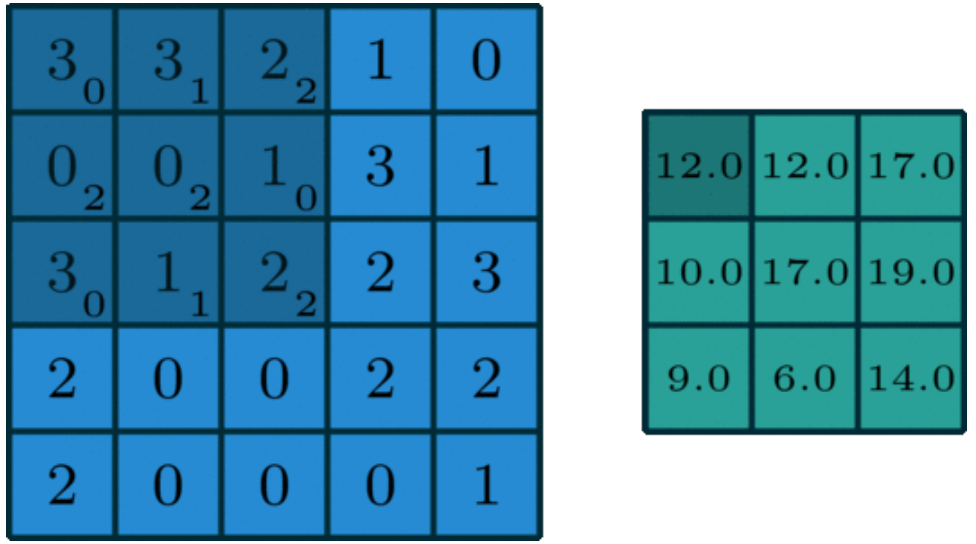
1D Discrete Convolution

$$C[n] \triangleq \sum_{m=-M}^{M} f[m]g[n-m]$$



2D Discrete Convolution

$$C[i,j] \triangleq \sum_{m=0}^{(M_a-1)} \sum_{n=0}^{(N_a-1)} f[m,n] \cdot g[i-m, j-n]$$



[Dumoulin & Visin, A guide to convolution arithmetic for deep learning (2018)]

Examples (1/3)

$$* \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} =$$

Examples (2/3)

$$* \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} =$$

Examples (3/3)

$$* \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} =$$

Convolutions Change the Dimensionality of the Output

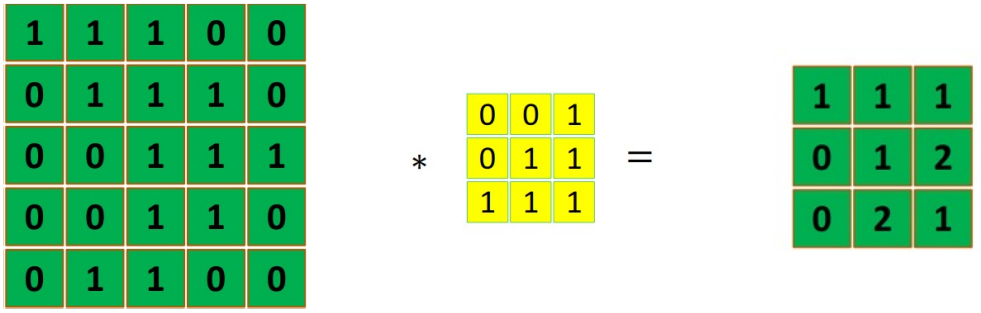
- When convolution is applied the output changes.
- Let's consider a simple 2D example:

  Input: [5x5]

  Kernel: [3x3]

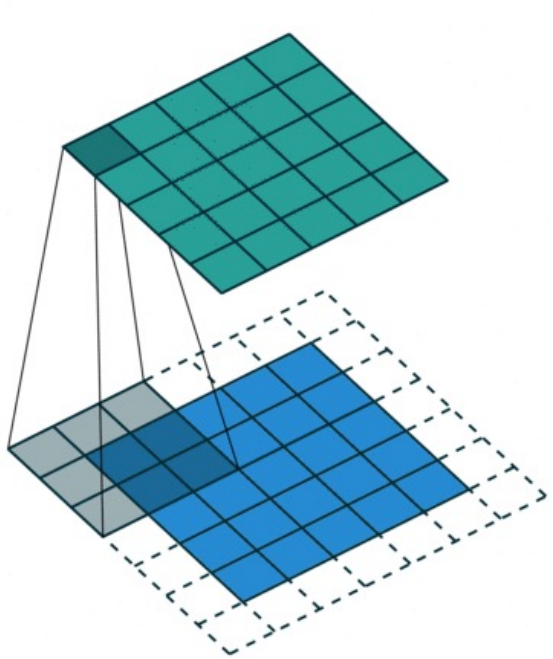
  Output after convolution?

  -> Output size gets smaller!



Padding to Maintain Input Dimensionality

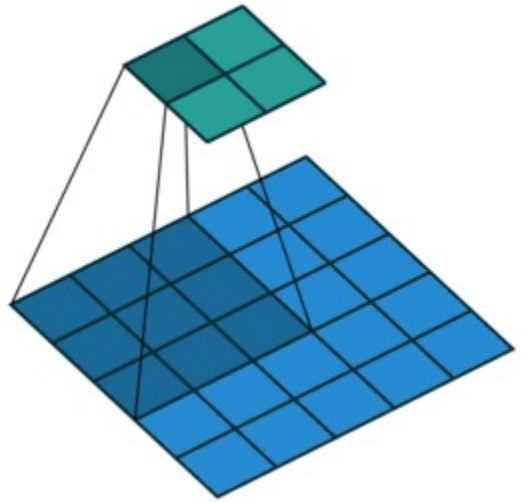
- Expand the size of data (image) with p values around the edges (e.g. constant, zero, mean, symmetric)



[Dumoulin & Visin, A guide to convolution arithmetic for deep learning (2018)]

Strided Convolutions to Decrease Dimensionality

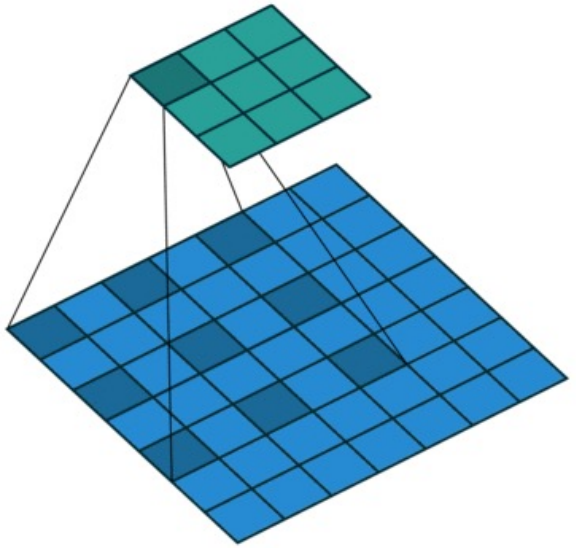
- Increase step size / stride (s) when processing the input



[Dumoulin & Visin, A guide to convolution arithmetic for deep learning (2018)]

Dilated (Atrous) Convolutions to Expand Spatial Coverage

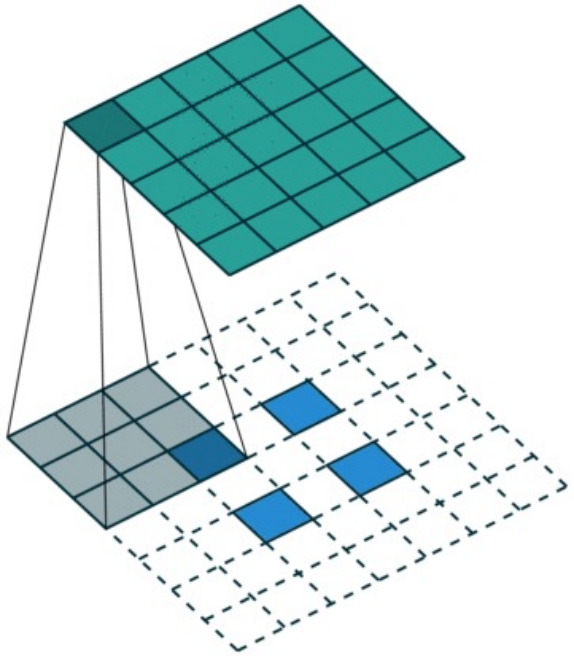
- Change the spacing (d) between the elements of the kernel/filter.



[Dumoulin & Visin, A guide to convolution arithmetic for deep learning (2018)]

Transpose Convolutions to Reverse the Process

- Transposing a convolution with stride 2 is equivalent to padding each pixel of the input/image with zero padding of size 1 pixel.



[Dumoulin & Visin, A guide to convolution arithmetic for deep learning (2018)]

Calculating the Output of a Convolutional Operation

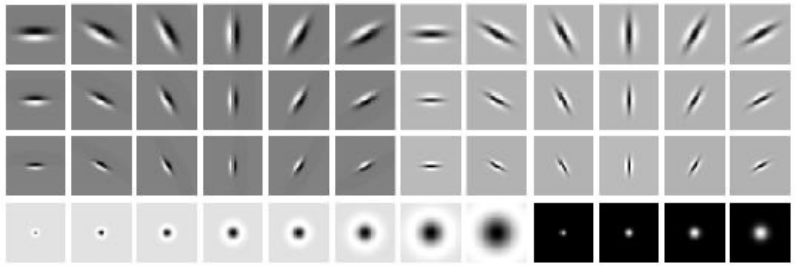
- We should take into consideration:
  - Input Size: n_{inp}
  - Kernel Size: k
  - Stride Size: s
  - Dilation Size: d
  - Padding Size: p

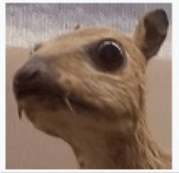
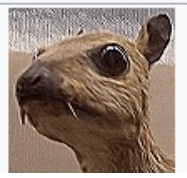
- Output size:

$$n_{out} = \frac{n_{inp} + 2p - (d(k-1)+1)}{s} + 1$$

Convolutions Summary

- Convolutions can be used to locally process the input signal
- Convolution result highlight patterns similar to the kernel
- Can be quite handy in signal processing
- The size of the output of a discrete convolution operation changes with respect to its input.
- Historically used for image processing



Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

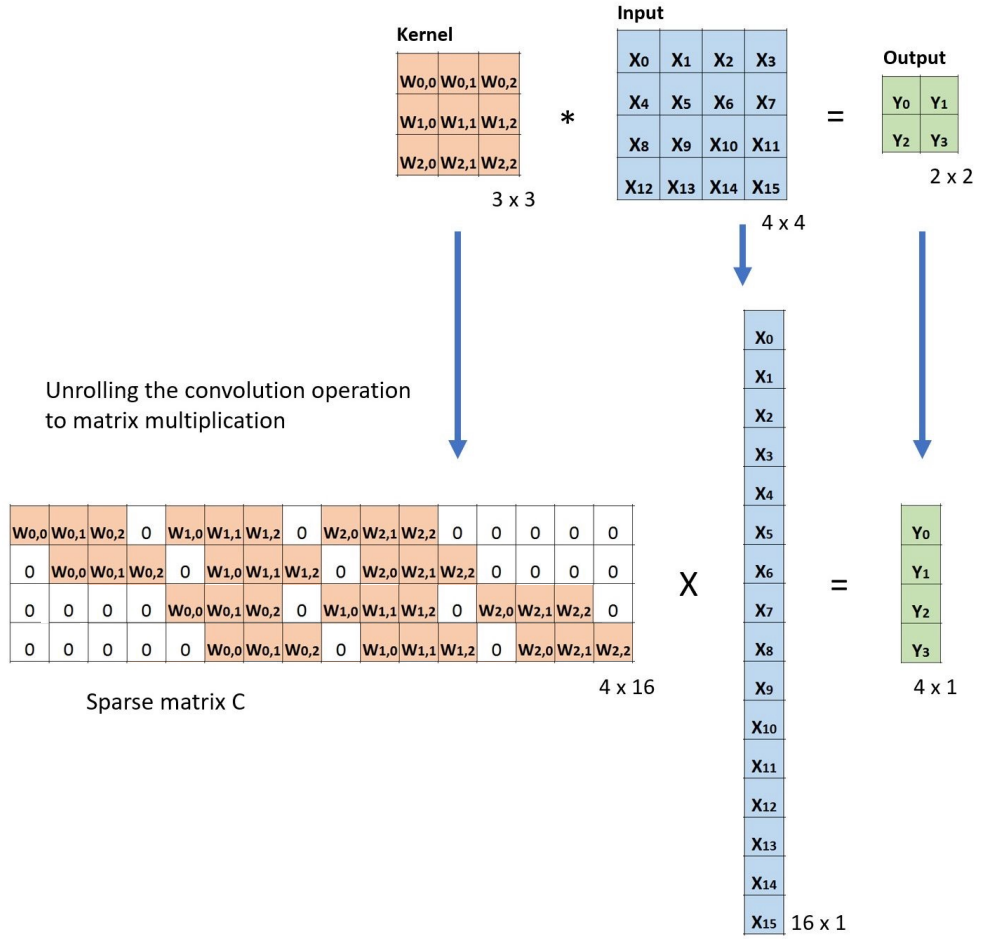
Convolutional Neural Networks (ConvNets)

How can convolutions be used in Neural Networks?

- **Convolution kernels can be trainable.**
- Essentially it can be performed by applying a dot product (Toepliz matrix transformation).
- The convolution operation can be applied in any number of dimensions (1D, 2D, 3D, … etc.)
- The gradient w.r.t. it's parameters and inputs:

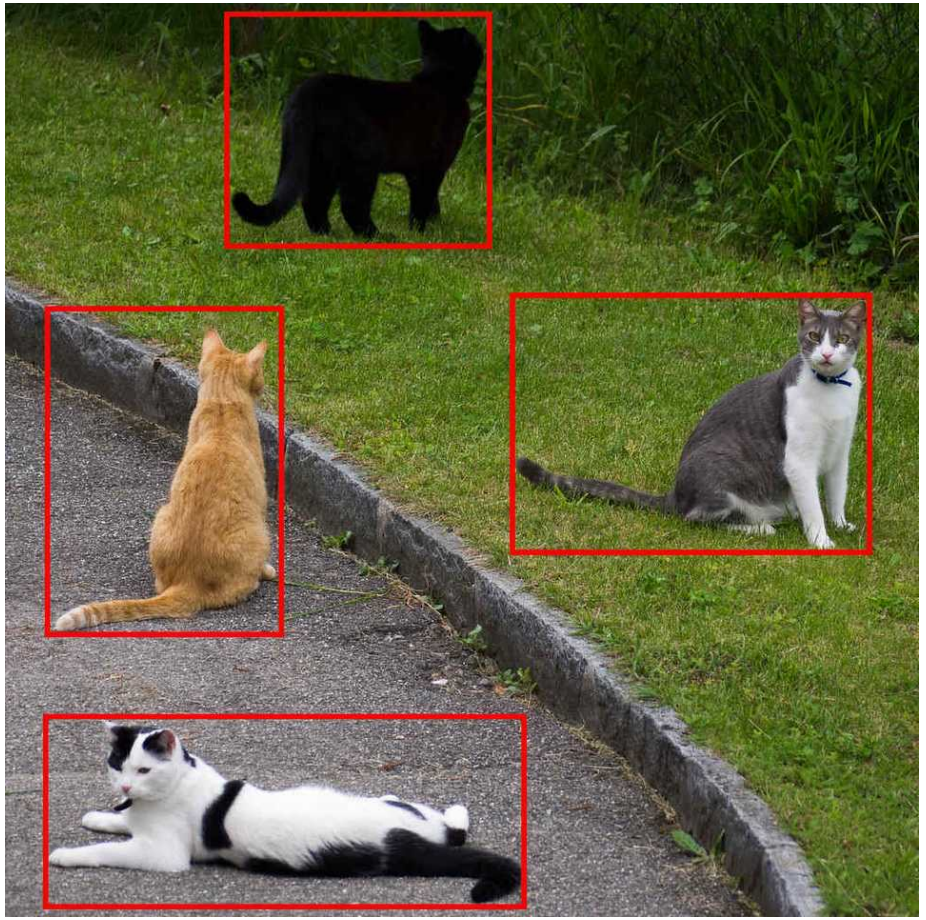
$$\frac{\partial a_{rc}}{\partial w_{ij}} = x_{r-i,c-j}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_r \sum_c \frac{\partial \mathcal{L}}{\partial a_{rc}} x_{r-i,c-j}$$



Images in the context of Neural Networks

- ConvNets were initially utilized mostly for images.
- How images are different?
- Images are 2D:
  - 3D if you also count the extra channels (RBG, depth, hyperspectral, etc.)
  - 4D also in the case of some medical image volumes or video.
- What does a 2D/3D/4D input really mean?
  - Neighboring variables are locally correlated
  - Spatial Structure

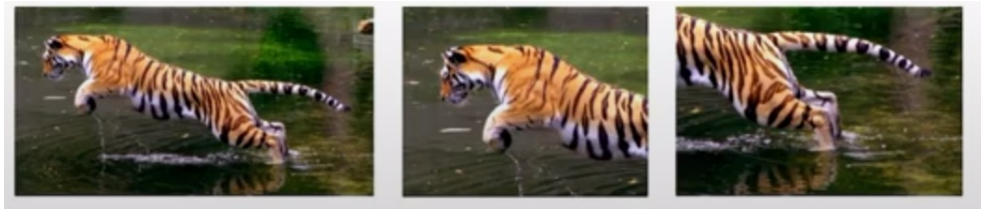


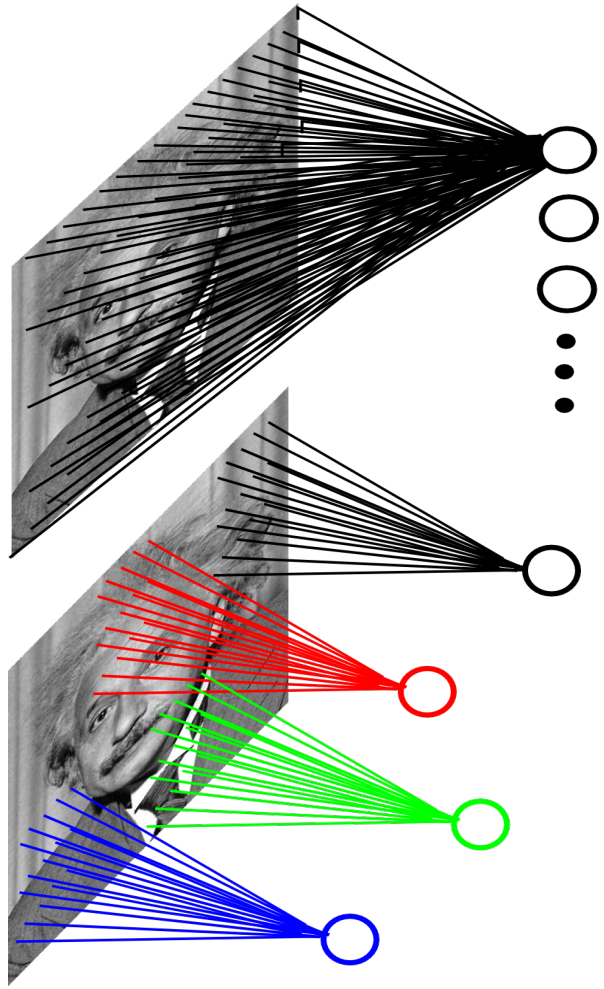
Images in the context of Neural Networks

- Huge dimensionality
  - A 200x200 (Grey) image amounts to 40K input variables
  - 1-layered NN with 1000 neurons → 40M params

- Images are stationary signals → they share features
  - Cropping/shifting/occluding → still an image
  - Possibly with same semantics
  - Basic natural image statistics stay the same

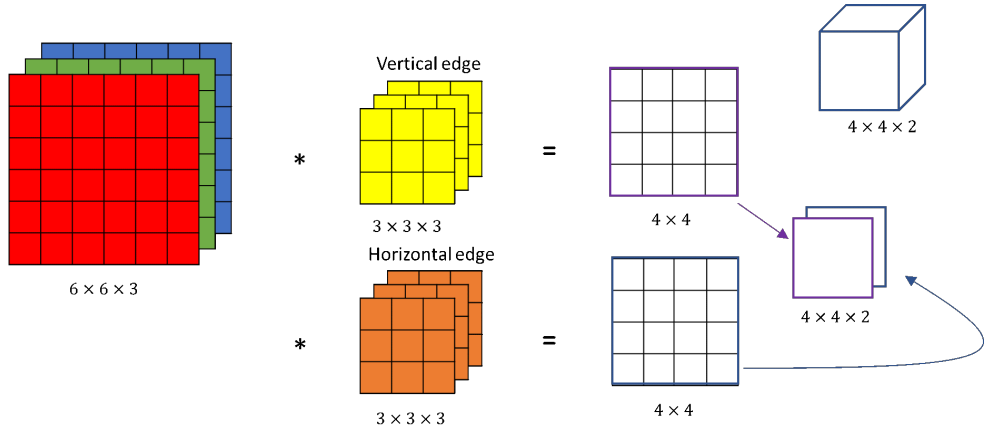
→ Do we really need a single neuron for each pixel value?





Convolutional Modules on Neural Networks for Images

- Input dimension of a multi-channel image:
  - Width x Height x Channels (RGB)
- Multiple Kernels (n_k) are typically trained on the same convolutional module.
- Multiple convolution outputs are stacked together forming a multi-channel response (number of channels equal to the number of kernel)
- Advantages:
  - Preservation of spatial correspondence
  - Less trainable parameters (local connectivity and parameter sharing)



Subsampling/Pooling Modules

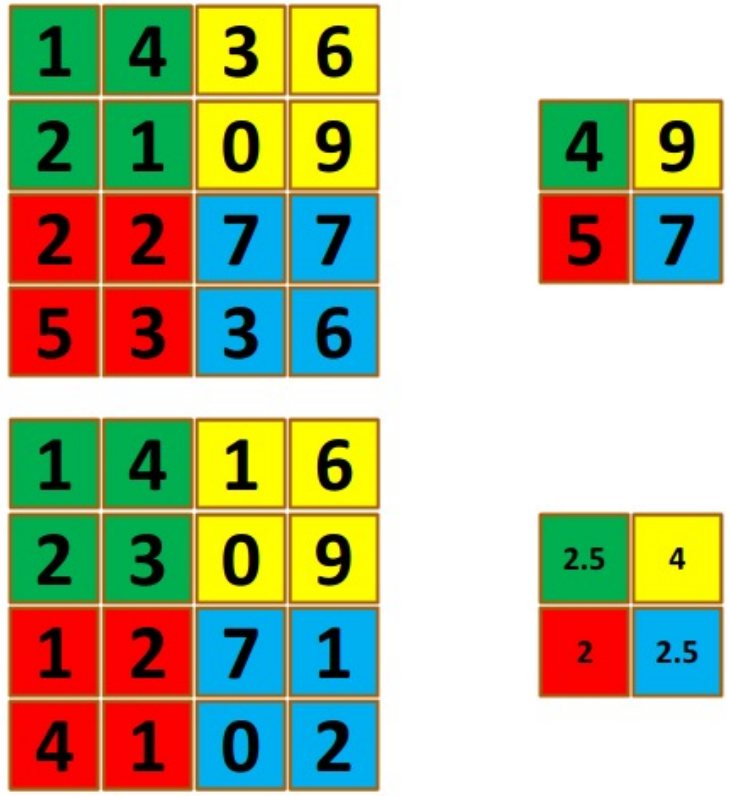
Subsampling/Pooling Modules

- Aggregate multiple values into a single value
  - Invariance to small transformations
  - Reduces the size of the layer output/input to next layer → Faster computations
  - Keeps most important information for the next layer
  - Pooling size is a hyperparameter
- Max pooling:

$$\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & otherwise \end{cases}$$

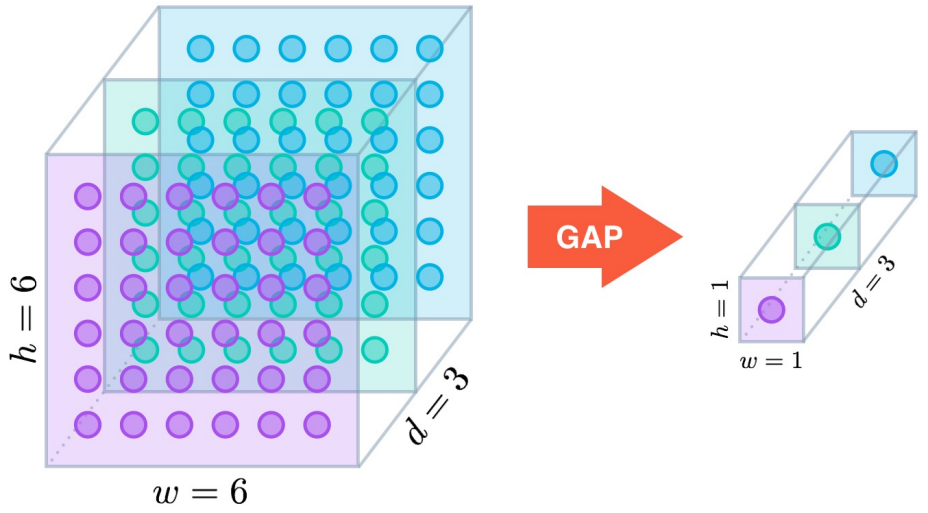
- Average pooling:

$$\frac{\partial a_{rc}}{\partial x_{ij}} = \frac{1}{r \cdot c}$$



Global Average Pooling (GAP)

- Reduces the whole feature map into a single values
- Parameter free
- Initially suggested using the average operation
- Other aggregations functions might also be used (min, max)
- The feature representation losses it's spatial characteristics
- Enforces correspondences between feature maps and output categories → enforces feature maps to be confidence maps of concepts (categories).



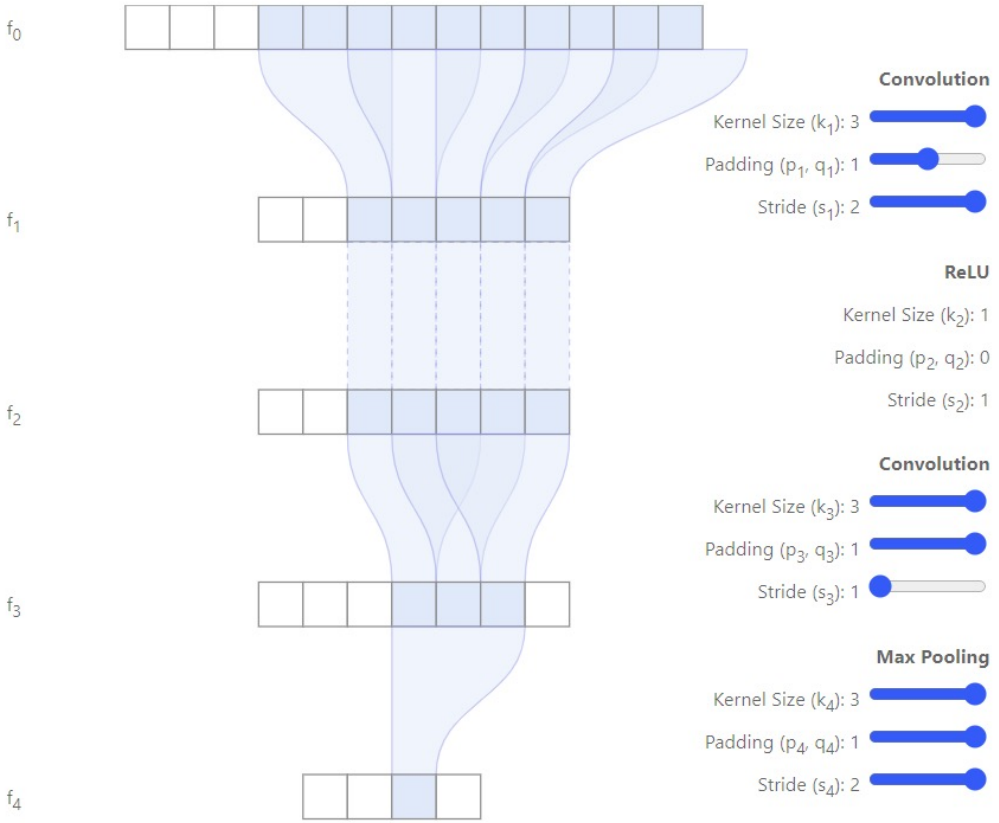
[Lin et al., Network In Network, ICLR (2014)]

Receptive Field

- The size of the region in the input that produces the feature.
- Calculating receptive field:

$$r_0 = \sum_{l=1}^{L} \left((k^{ext}{}_l - 1) \prod_{i=1}^{l-1} s_i \right) + 1$$

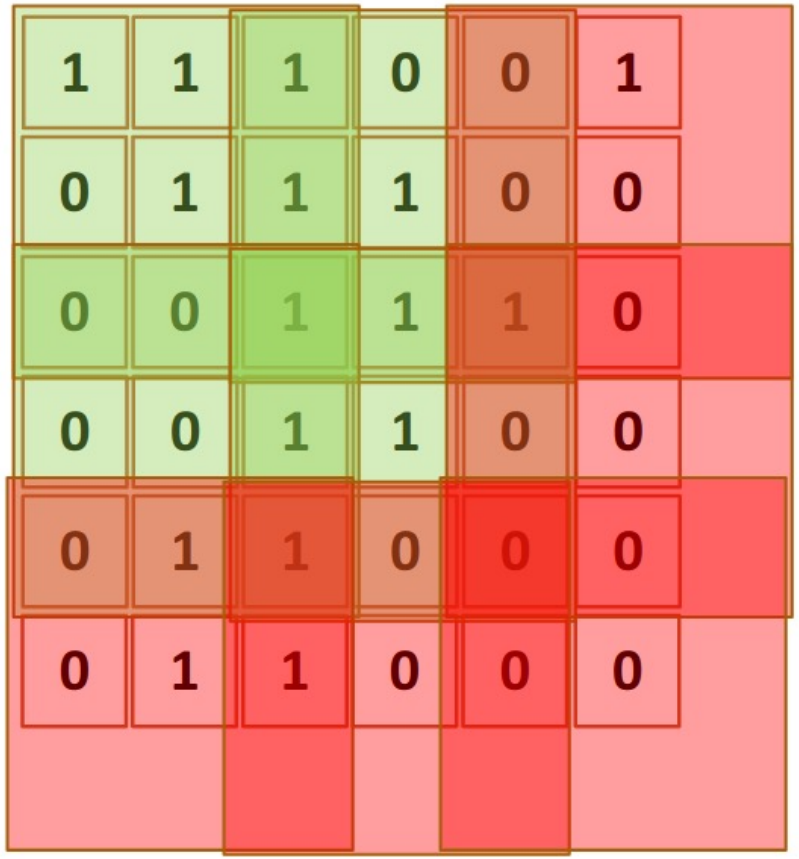
- Does the receptive field covers the entire input?
  - In the most SOTA networks the receptive field covers the entire input image
- How fast/slow does the receptive field grows?



[André Araujo et al., Computing Receptive Fields of Convolutional Neural Networks, distill.pub (2019)]

Good Practices

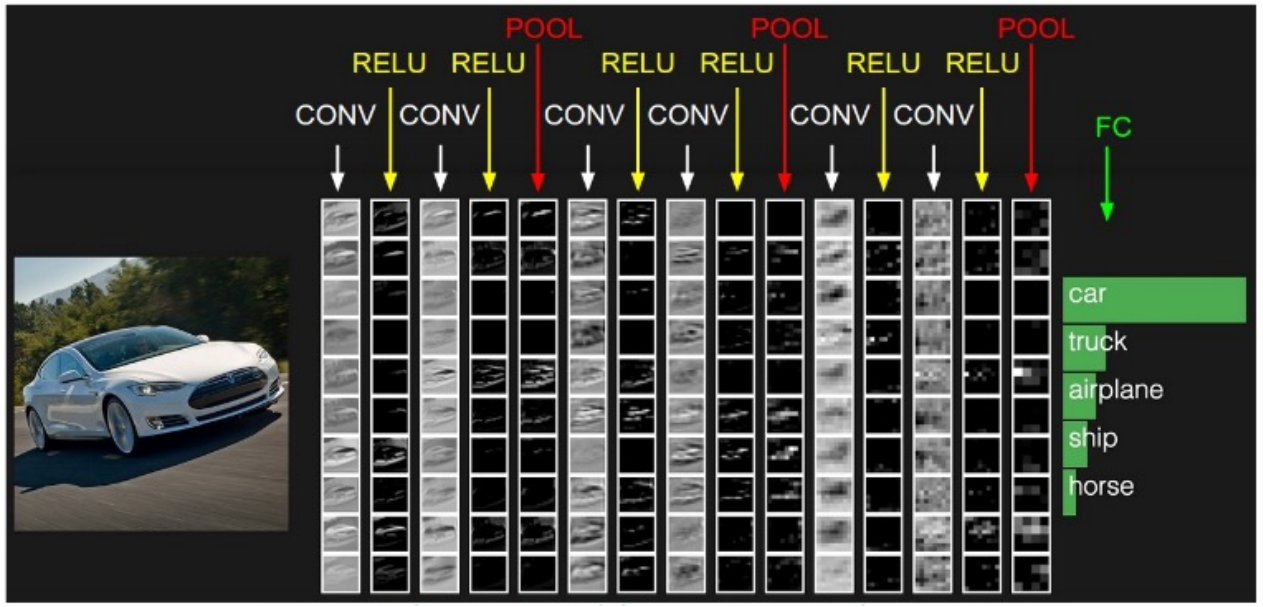
- Resize the image to have a size in the power of 2
- Prefer odd sizes for kernels in order to avoid spatial shifting (no mid point)
- Avoid hyper-parameters that do not click (filter larger than the input, or really high strides)
- Make sure that your receptive field is not extremely small or large in relation with the input
- Parameters that are typically used:
  - $n_{in} = \{32, 64, 224, 256, \dots\}$ (can be divided by 2 multiple times)
  - $k = \{3, 5\}$
  - $s = \{1, 2\}$
  - $p = \ (function\ of\ the\ kernel\ extend)$



Visualizing ConvNets

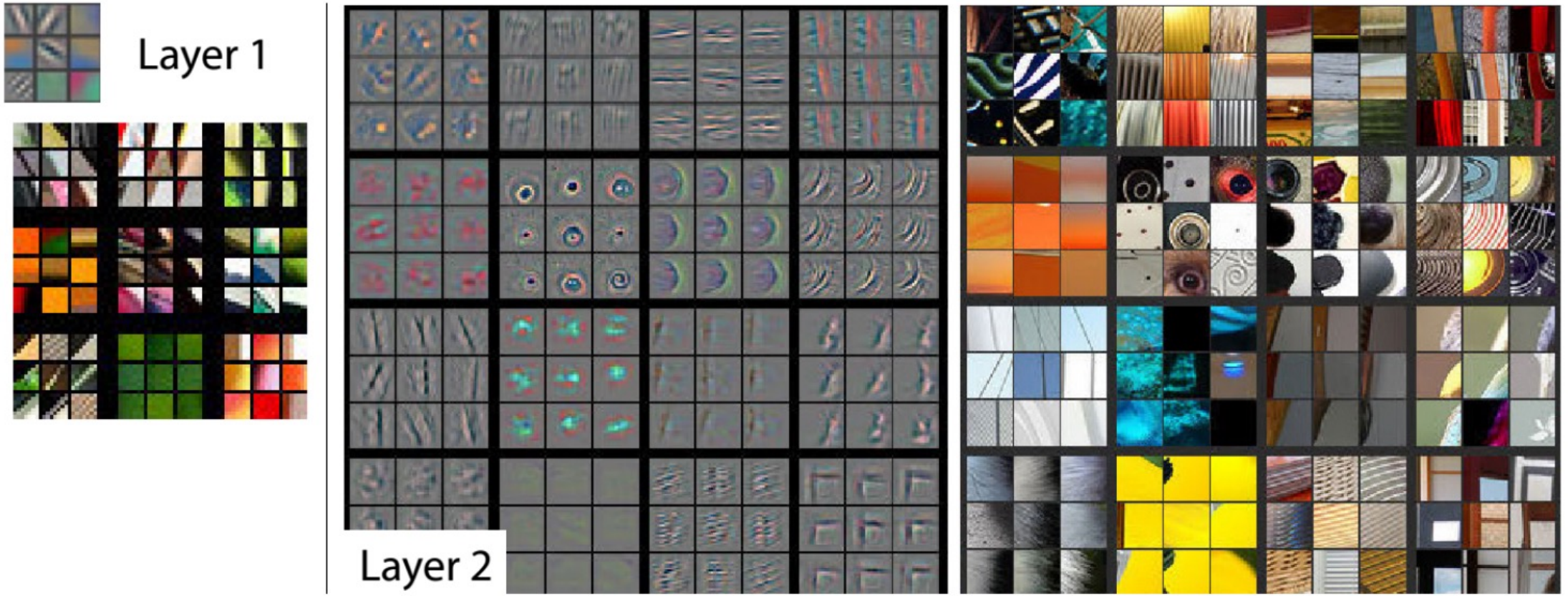
Feature Maps

- Convolution activations → feature maps
- A deep network has several hierarchical layers
- Hence several hierarchical feature maps going from less to more abstract



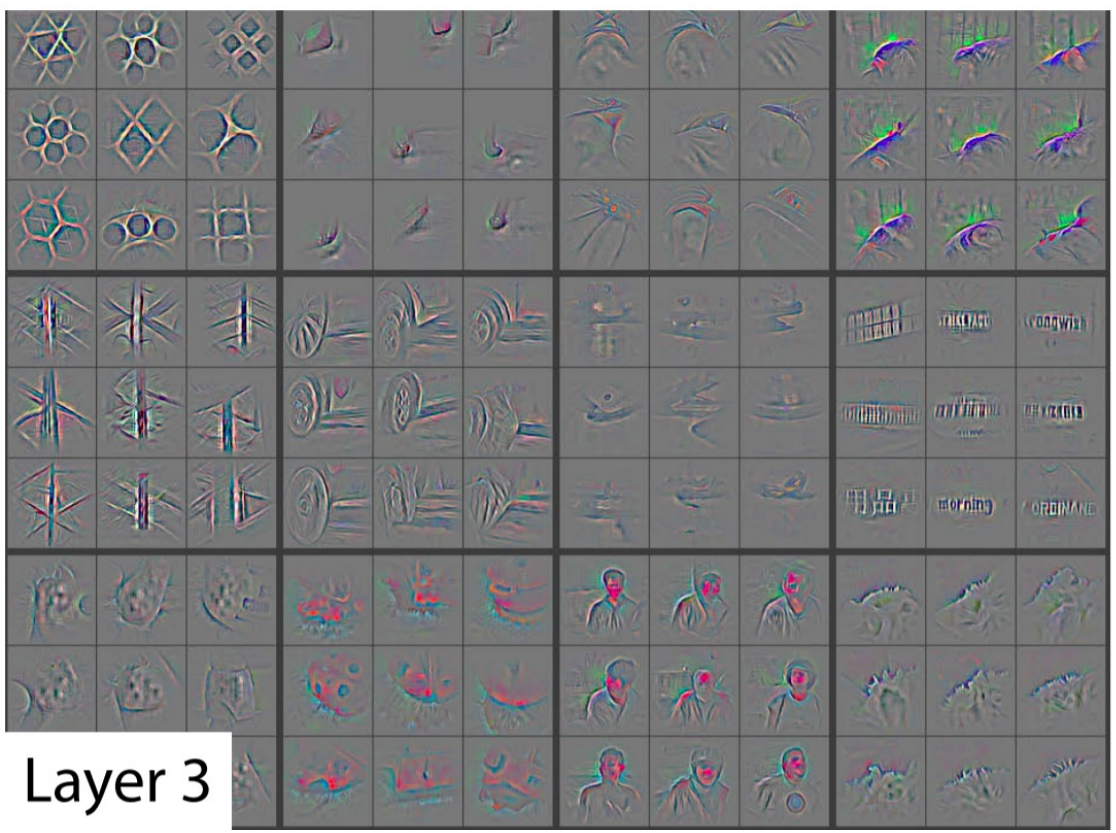
[https://cs231n.github.io/convolutional-networks/]

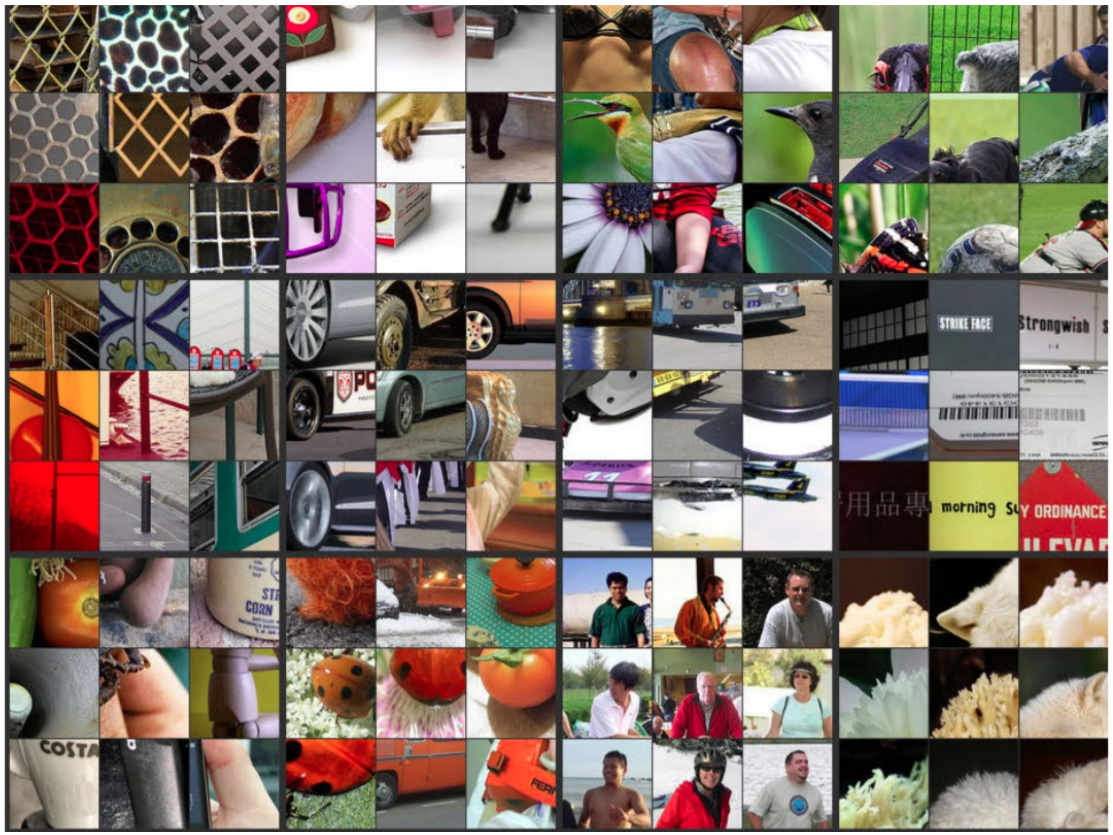
What Input Maximizes Feature Map Outputs?



[M. Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV, (2014)]

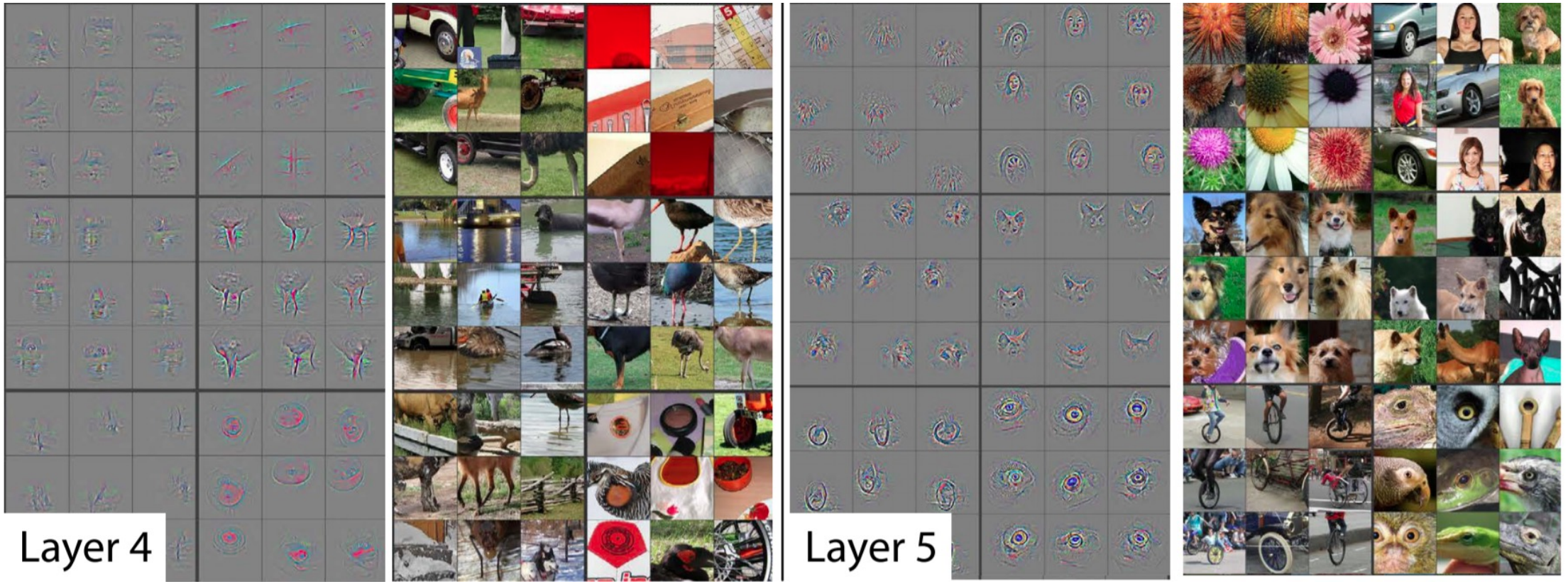
What Input Maximizes Feature Map Outputs?





[M. Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV, (2014)]

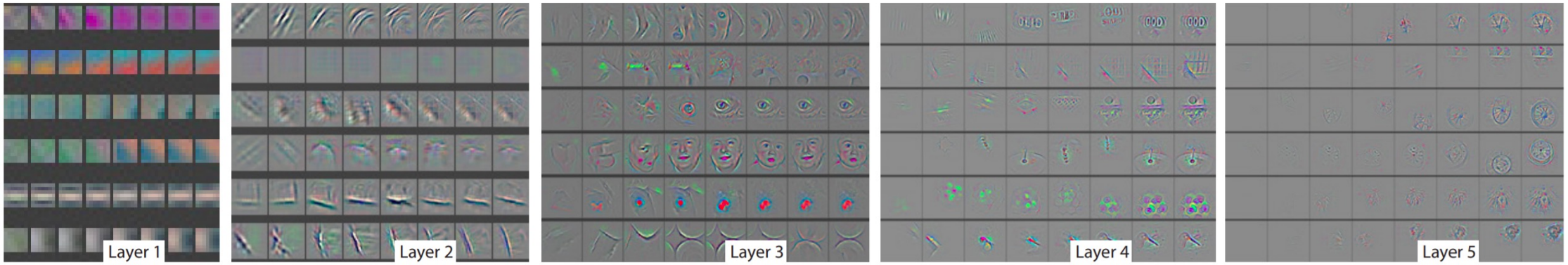
What Input Maximizes Feature Map Outputs?



[M. Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV, (2014)]

What Input Maximizes Feature Map Outputs?

- Evolution of a randomly chosen subset of model features through training.
- How the features change at epochs 1,2,5,10,20,30,40,64



[M. Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV, (2014)]

Transfer Learning

Transfer Learning

- Assume two datasets S and T
- Dataset S (source) is
  - Fully annotated, plenty of images
  - We can build a model h_S
- Dataset T (target) is
  - Not a much annotated, or much fewer images
  - The annotations of S do not need to overlap with T
- We can use the model h_S to learn a better h_T
- This is called transfer learning

(Source, e.g. ImageNet 1M samples)

(Target, 1K samples)

How Can we Use Transfer Learning?

1. Train out model in the source dataset.
2. Drop a number last layers
3. Add a new layer at the end with the correct number of classes
4. Continue the training using the target dataset:
   a. Train all the parameters of the model
   b. Freeze first layers and train only the last ones
   c. Freezing fade out during training

Which Layers to Replace?

- Higher layers capture more semantic information → Good for high level classification
- Lower layers capture more basic information → Good for image-to-image comparison

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

[M. Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ECCV, (2014)]

Take Home Messages

- Convolution operation can be used to detect patterns that are similar with the convolution kernel
- Kernels can be defined as trained parameters → Our model can learn the most relevant patterns that are important for our task.
- Convolutions and Subsampling Modules change the dimensions of their input → Keep track of these dimensions.
- Visualizing ConvNet feature maps shows us that the representations that are learned in the deeper layers are more and more abstract.
- Transfer Learning can be used in order to pre-train the parameters of a network using large dataset with similar characteristics as the studied task.