

Convolutional Neural Networks (CNNs)

Computer Vision Problems

Image Classification



64x64



Cat
Dog
...
...
House
Car

$$\begin{bmatrix} 0.8 \\ 0.2 \\ \\ 0 \\ 0 \end{bmatrix}$$

Object detection

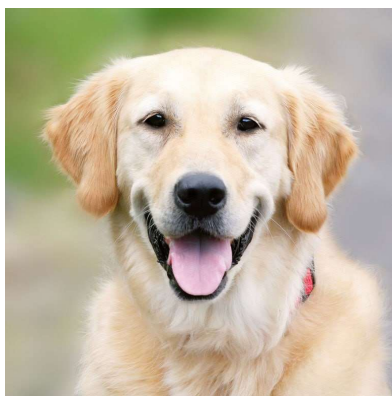


Computer Vision Problems

- **Historically**, computer vision was on **hand-crafted features**, constructed and used as input to simple learning algorithms (Hartley and Zisserman, 2004).
- Today features are no longer hand-crafted but **learned** by CNN architectures.
- Intuitively, CNN architectures learn alone what are the ‘key features’ of different classes (e.g., cat, dog, car, house)



Furry
Round shape
Triangle ears



Furry
Round shape
Long ears
Long tongue



Squared shape
Wheels
Licence plate



Windows
Doors
Steps

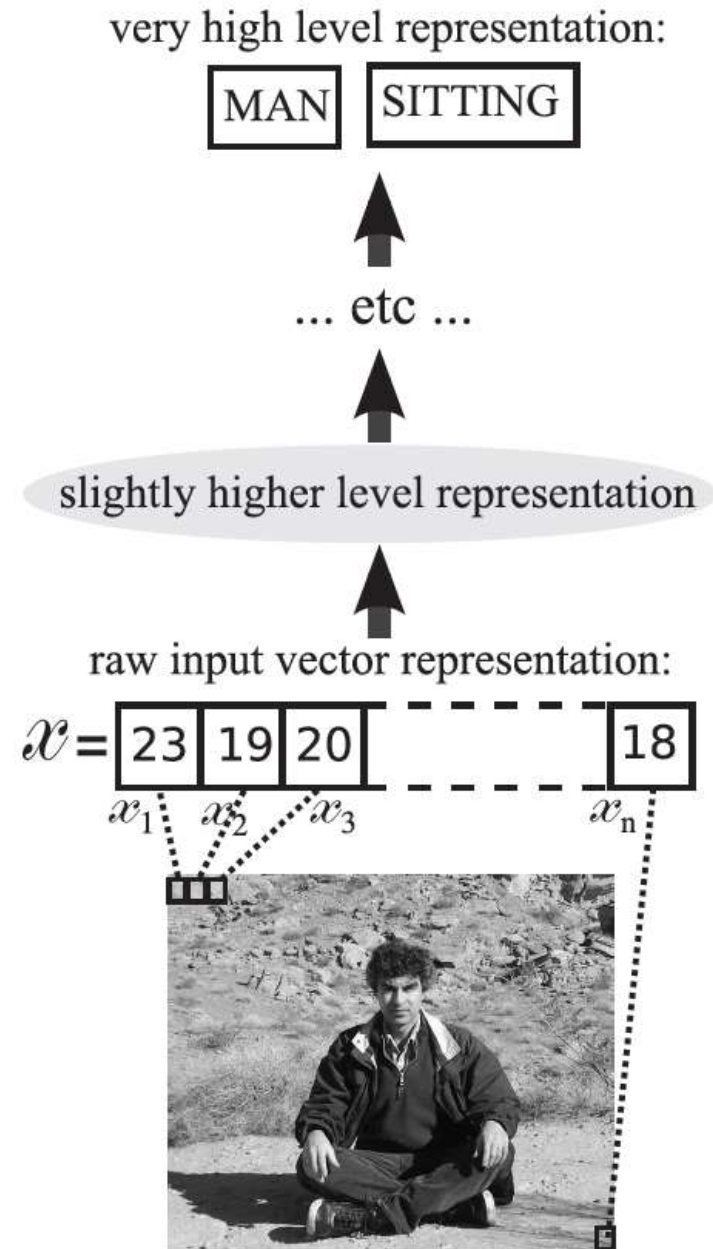
Although usually features learned by CNNs cannot be described in such intelligible terms!

Why CNNs?

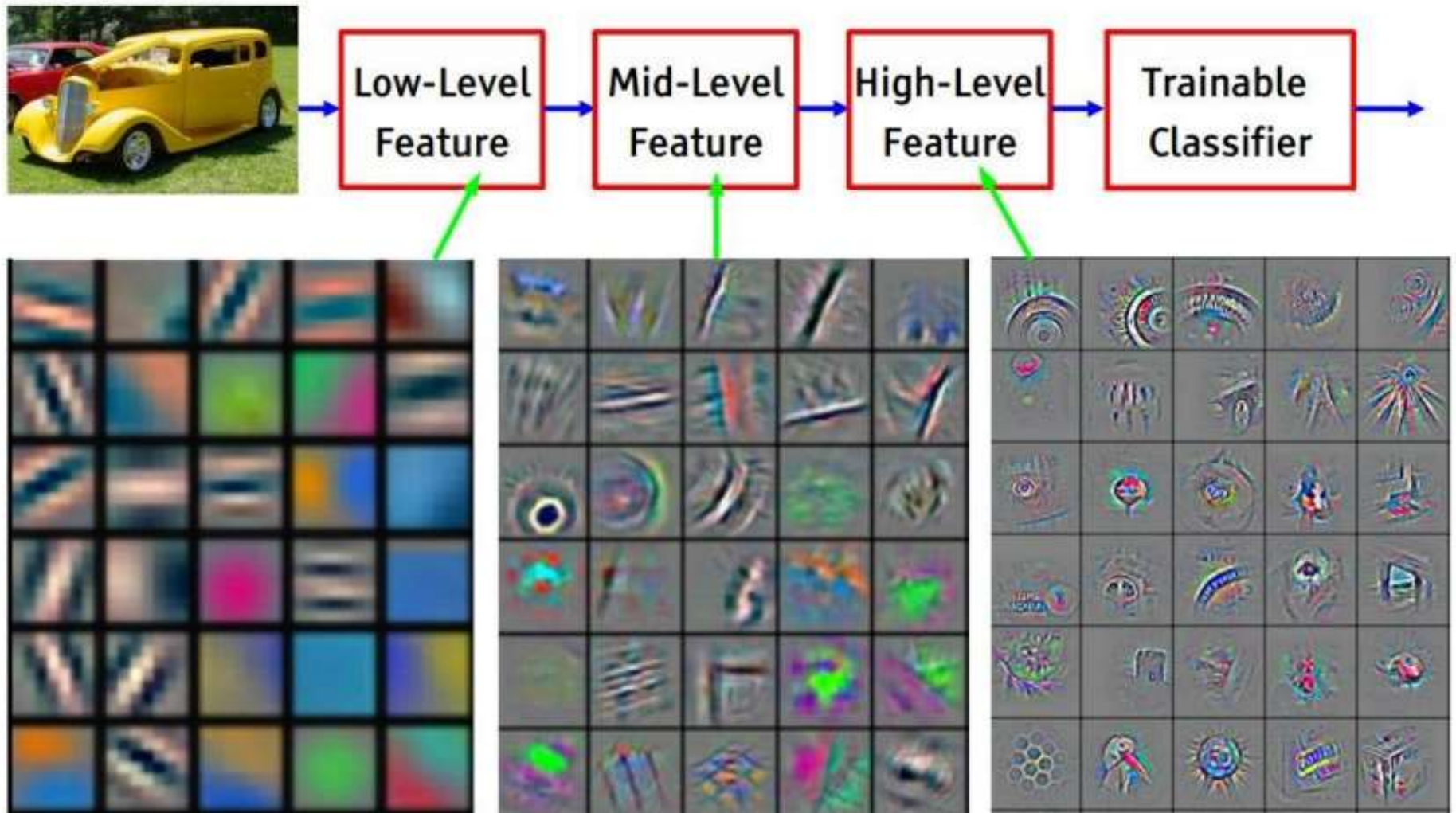
- Traditional Neural Networks don't scale well for image data.
- CNNs are designed specifically for image recognition tasks.
- Key advantages: local connections, shared weights, pooling, hierarchy of features.

Why CNN? (and why Deep?)

1. Passage through all these phases is necessary since it is not possible to pass from raw pixels to abstract concepts («MAN»):
2. The set of images for which «MAN» would be appropriate form a very convoluted region
3. Better to go through a set of abstractions or features
4. The focus of deep neural networks is to go through all these levels of abstraction



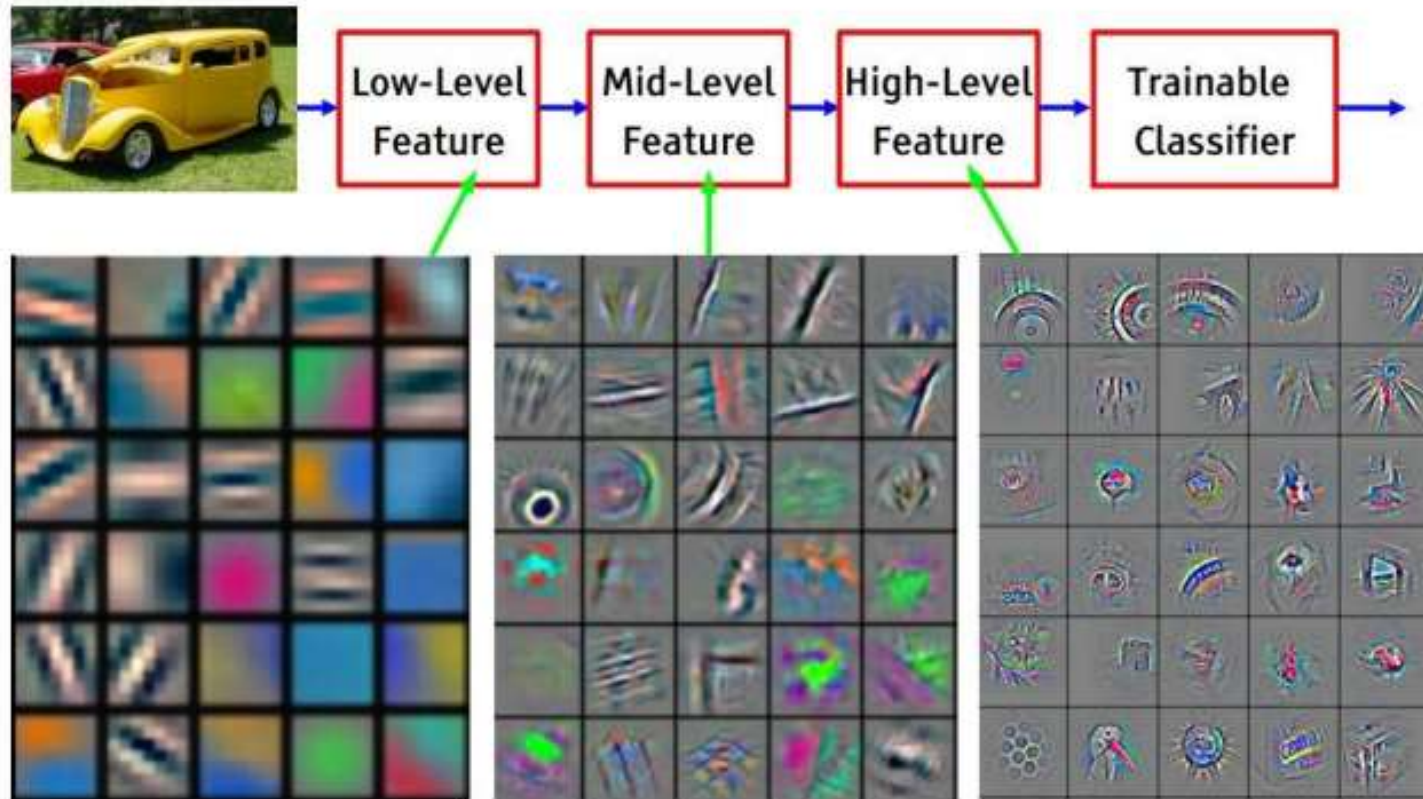
Hierarchy of features



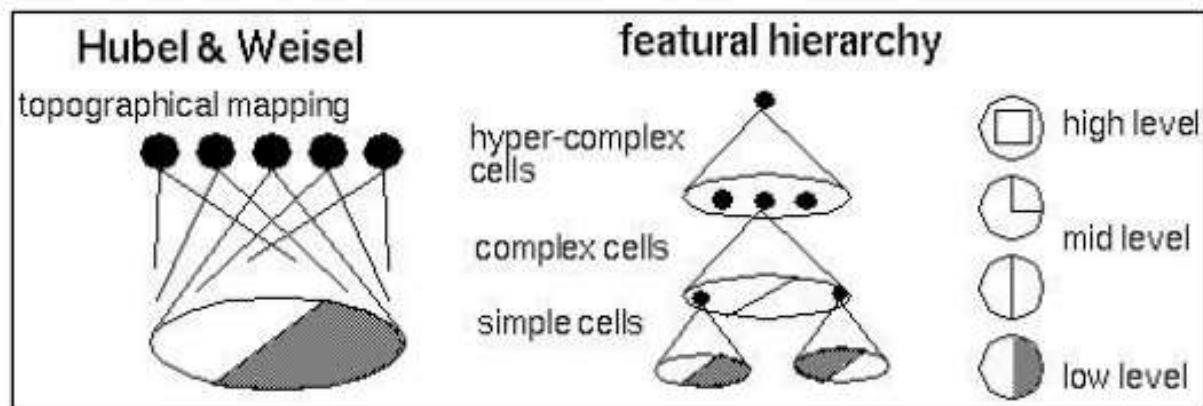
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

These are the features that maximize the activation of neurons at a given level.

Hierarchy of features



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



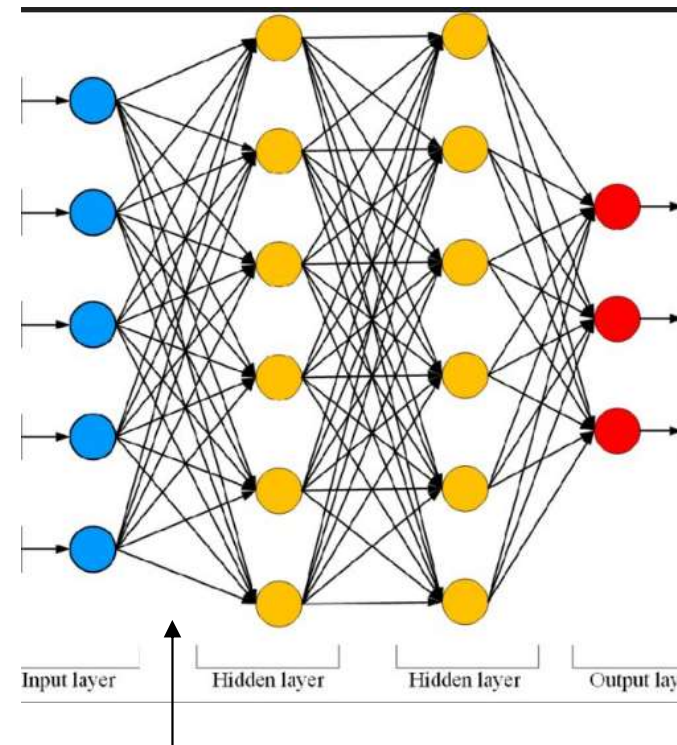
- Multilayer perceptrons don't scale well on big size images



$64*64*3$



$1000*1000*3 = 3.000.000$



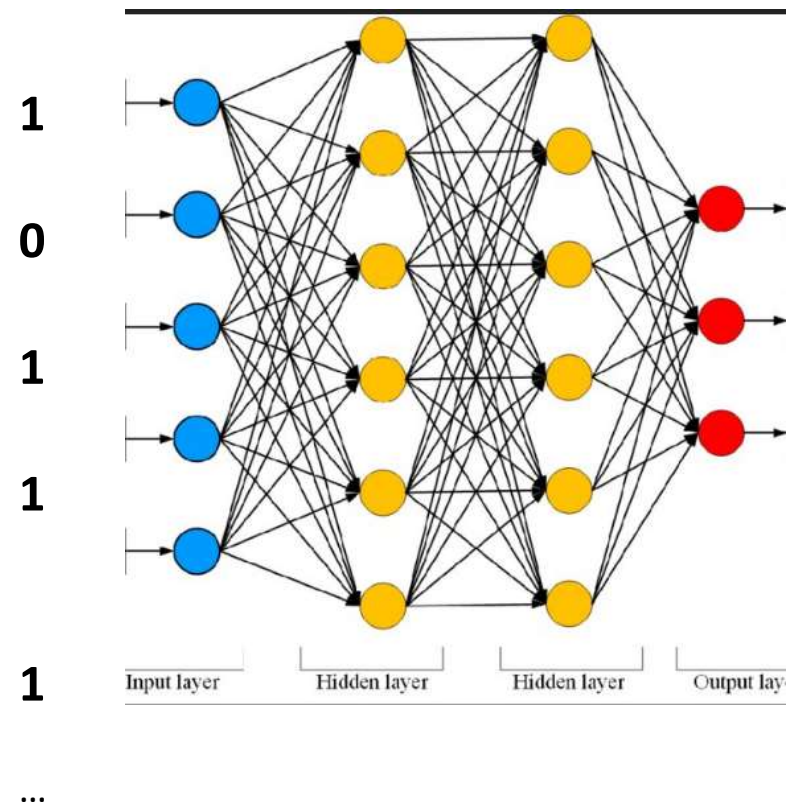
W has 3.000.000.000 weights!!!!

credits: deeplearning.ai, ng

- Multilayer perceptrons lose the spatial information of an image

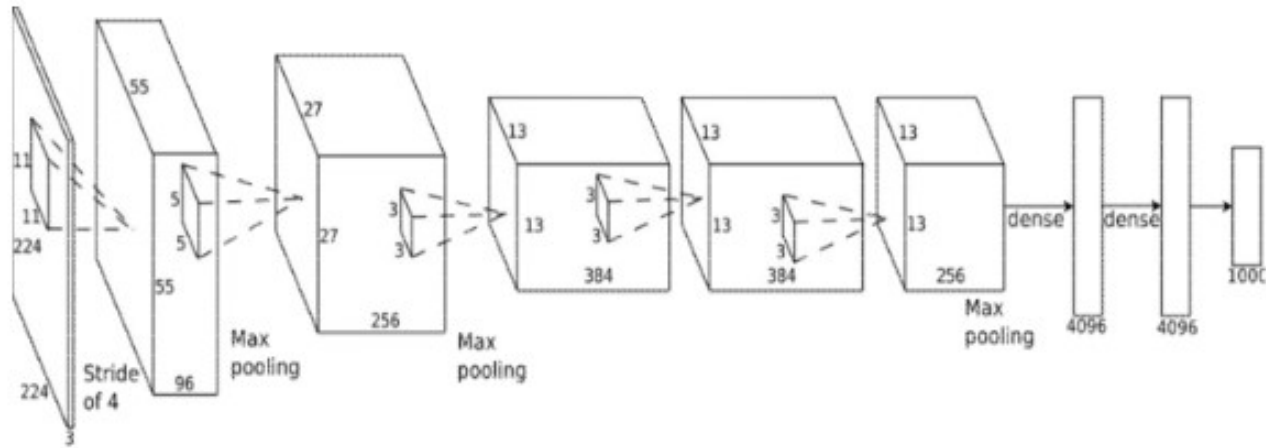


100*200



No spatial information preserved!

CNN Architecture



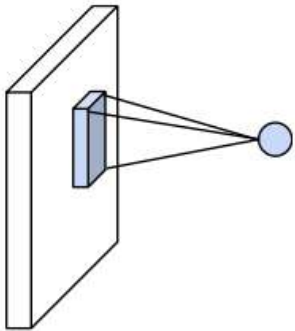
Alexnet

Krizhevsky, Sutskever, Hinton (2012)

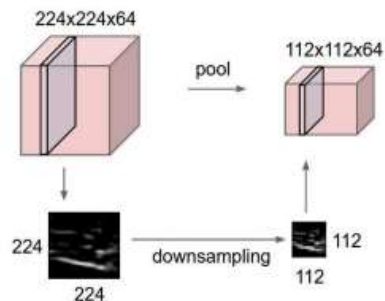
CNN Architecture

CNNs' basic elements:

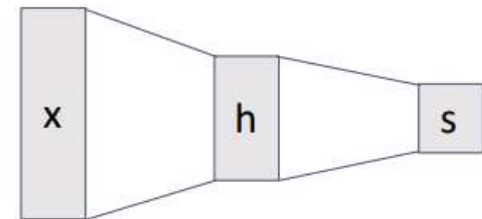
Convolution Layers



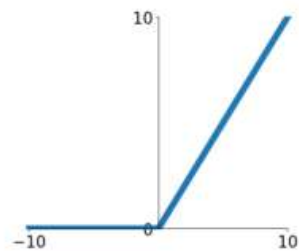
Pooling Layers



Fully-Connected Layers

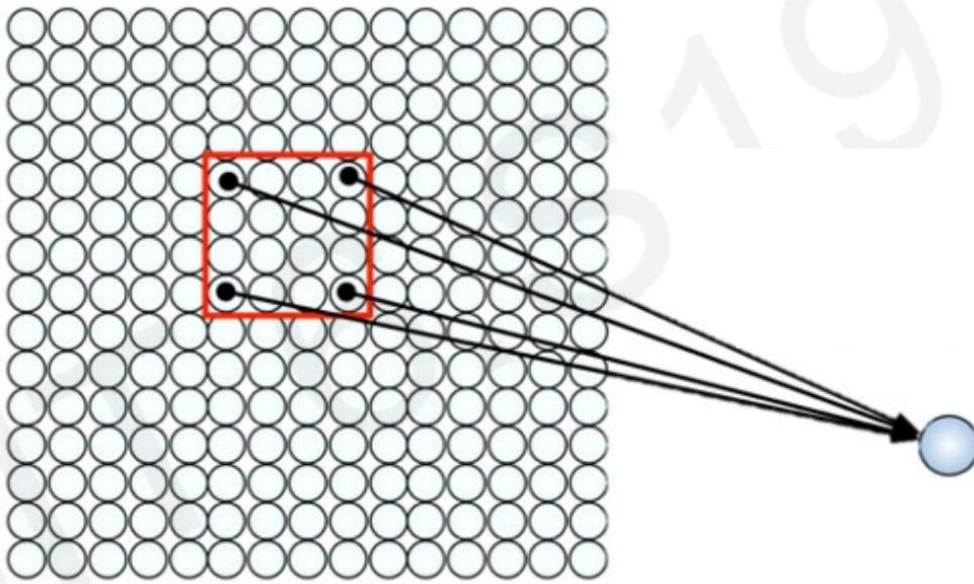


Activation Function



Convolution Layers

Instead of linearizing the image, and losing spatial information, hidden units are connected to a portion (a patch) of the image: the unit's **receptive field**.



This unit only sees a small region of the image

Weight vector = **filter, or kernel** that determines the kind of features for which the unit activates the most.

Unit activation $z = \text{RELU}(w^T x + w_0)$, with w and x are **vector's representation** of pixel values and weights. w_0 bias

Convolution Layers

.Examples of filters:

X-Shape Filter

1	0	1
0	1	0
1	0	1

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

4		

Application of
the filter to
the image
(first unit)

Same as $w^T x$ for w and x vectorized

Convolution Layers

.Examples of filters:

Vertical Edge
Filter

1	0	-1
1	0	-1
1	0	-1

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

*

1	0	-1
1	0	-1
1	0	-1

=

0	3	3	0
0	3	3	0
0	3	3	0
0	3	3	0

Convolution Layers

.Examples of filters:

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

Vertical Edge
Filter

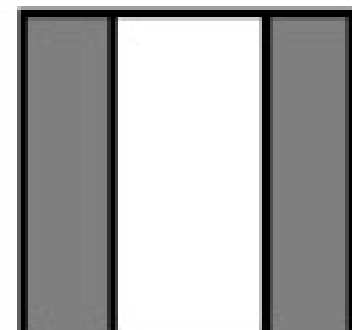
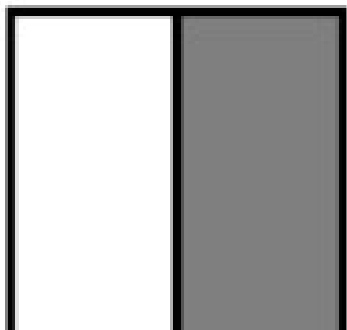
1	0	-1
1	0	-1
1	0	-1

*

1	0	-1
1	0	-1
1	0	-1

=

0	3	3	0
0	3	3	0
0	3	3	0
0	3	3	0



Convolution Layers: «Feature Maps»

- Nearby units are connected to adjacent regions of the image with the same weight vector
- i.e. detect the **same feature** in a different area of the image.
- Weight sharing!**

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

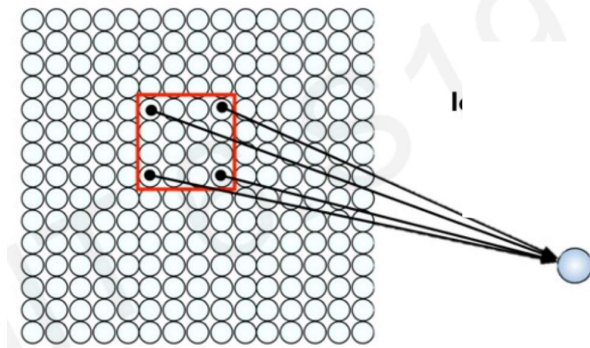
12	12	17
10	17	19
9	6	14

Convolution Layers: «Feature Maps»




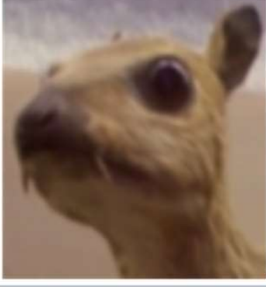
- Units of the hidden layer form a **feature map**
- **Convolutional layers** contain a set of feature maps, each with a different filter

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14



Other examples of Convolution filters:

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolution Layers: Properties

- Sparse weights
- Parameter (weight) sharing
- Locality-based analysis

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

- **Equivariance to translation** (if a feature – say an eye- is detected in a position, it can also be detected in another position never seen before by means of the same filter. It will activate a unit in the same layer but possibly in a different location)

Translation equivariance

- At a low level (feature extraction) : same filter detects same feature (e.g., an edge, or an eye) even if in a different position. This will activate a nearby unit in the same feature map



≠Translation invariance

- At the classification level, particular object should be assigned the same classification irrespective of its position within the image. (achieved with small translations or otherwise with data augmentation)



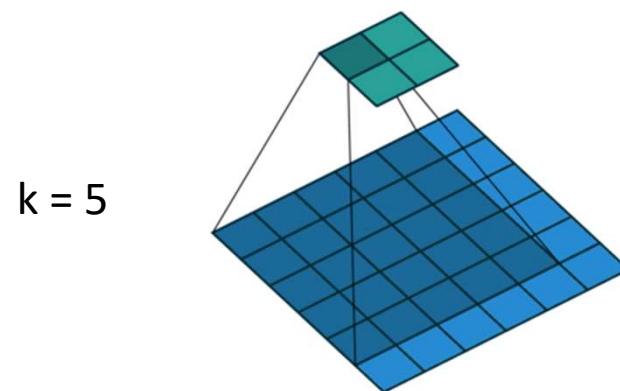
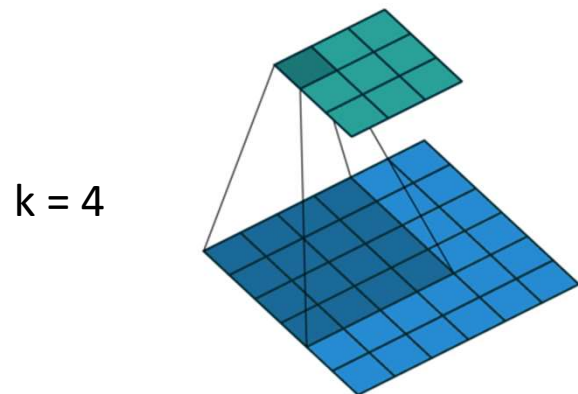
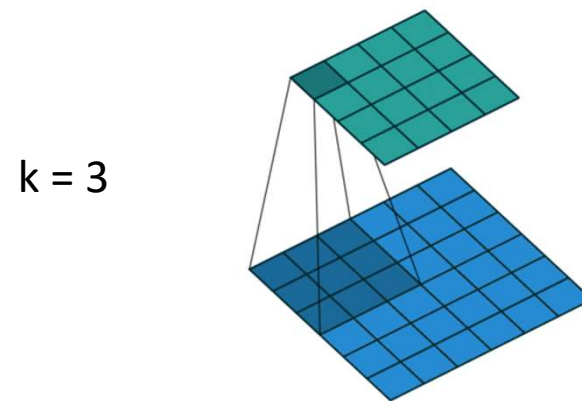
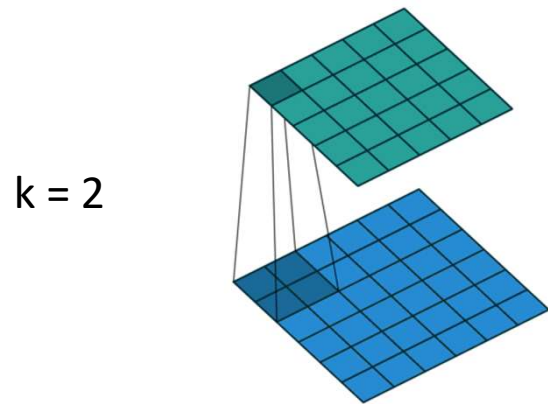
Convolution Layers: Hyperparameters

- **Kernel size** (ex 3×3 , usually odd)
- **Stride** 1: filter is moved one pixel at a time, S it is moved in larger steps of size S ($S = \text{stride}$)
- **Padding** the original image with additional pixels around the outside (so that the feature map can have the same dimensions as the original image)
- **The number of convolutional filters** (of feature maps in a convolutional layer)

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

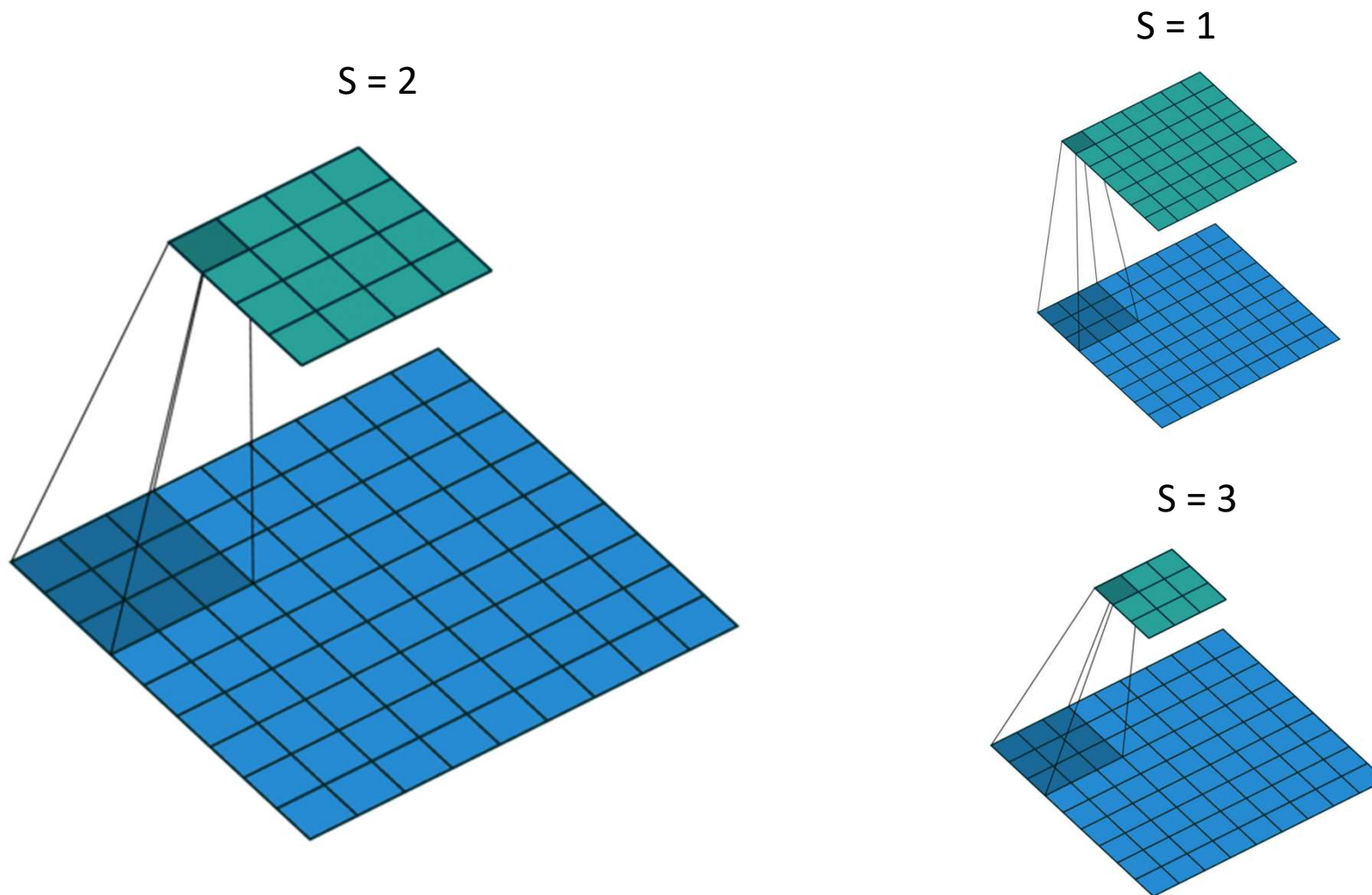
12	12	17
10	17	19
9	6	14

Convolution: kernel size



Larger kernel size \rightarrow smaller feature map

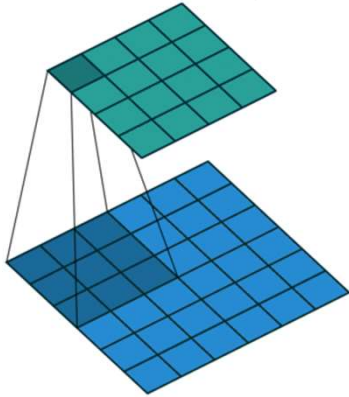
Convolution: stride



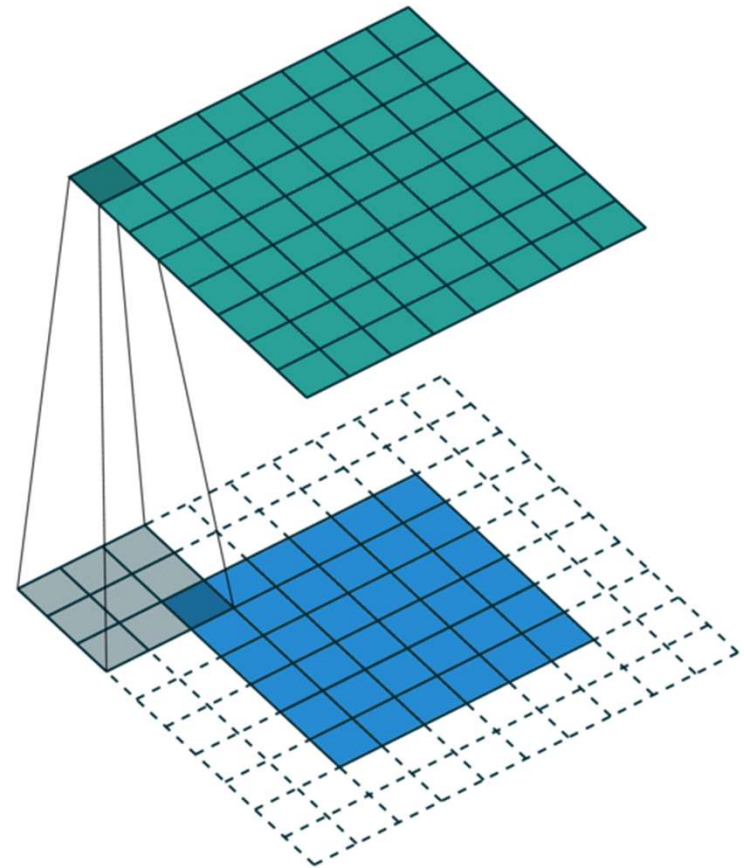
Larger stride \rightarrow smaller feature map

Convolution: padding

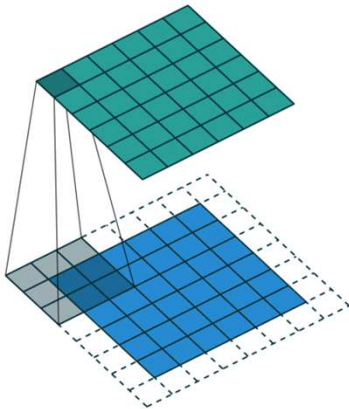
No padding



“Full” padding



“Same” padding



Convolution Layers: Filters

- In classical image processing, kernels' weights were **handcrafted** by experts
- Instead, Convolutional Neural Networks **learn** the best weights (with backpropagation algorithm)
- (learned filters are much less neat and interpretable than the examples we made)

Convolution Layers: Filters

- In classical image processing, kernels' weights were **handcrafted** by experts
- Instead, Convolutional Neural Networks **learn** the best weights (with backpropagation algorithm)
- (learned filters are much less neat and interpretable than the examples we made)

Hand crafted:

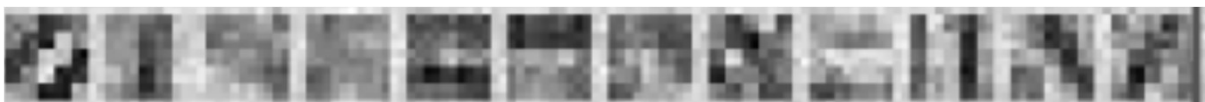
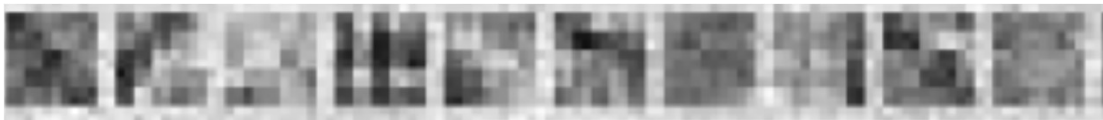
1	0	-1
1	0	-1
1	0	-1



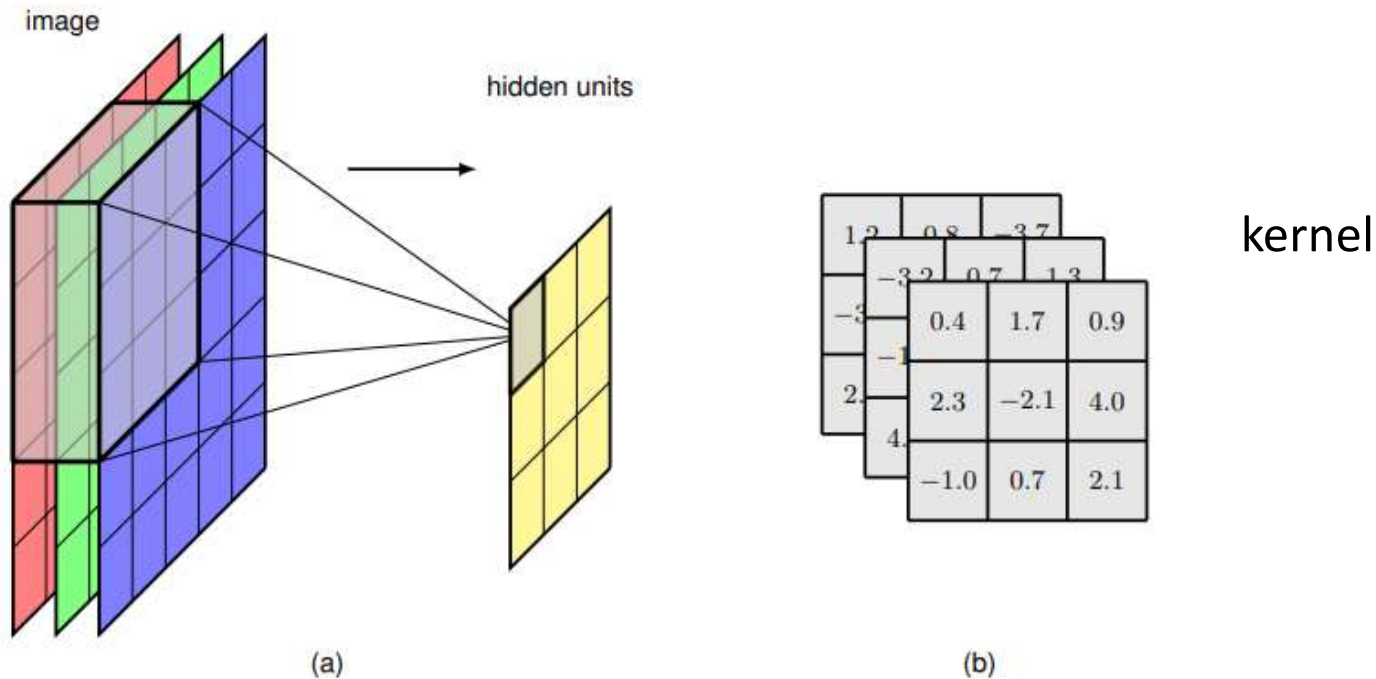
1	0	1
0	1	0
1	0	1



Versus learned:



Multidimensional convolution (input level)



A multi-dimensional filter that takes input from across the R, G, and B channels. The kernel here has 27 weights, 9×3 , plus a bias parameter not shown) and can be visualized as a $3 \times 3 \times 3$ tensor.

Multidimensional convolution (input level)

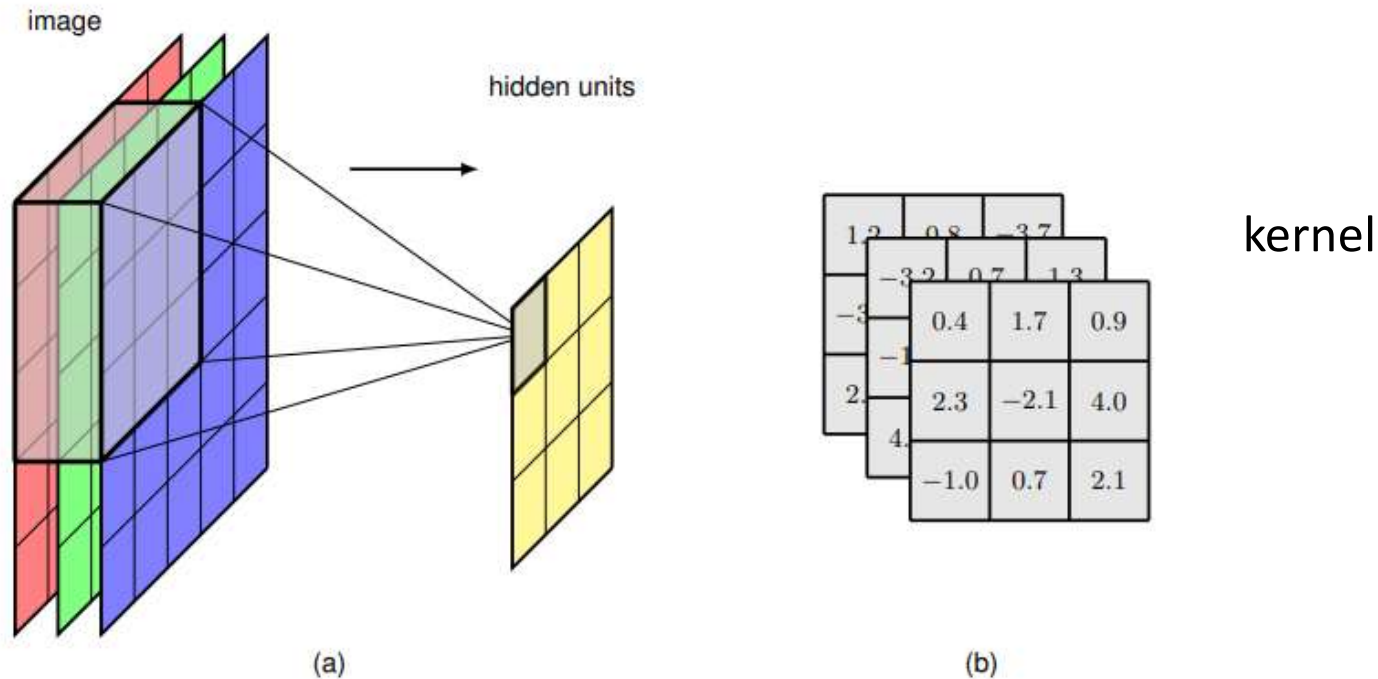


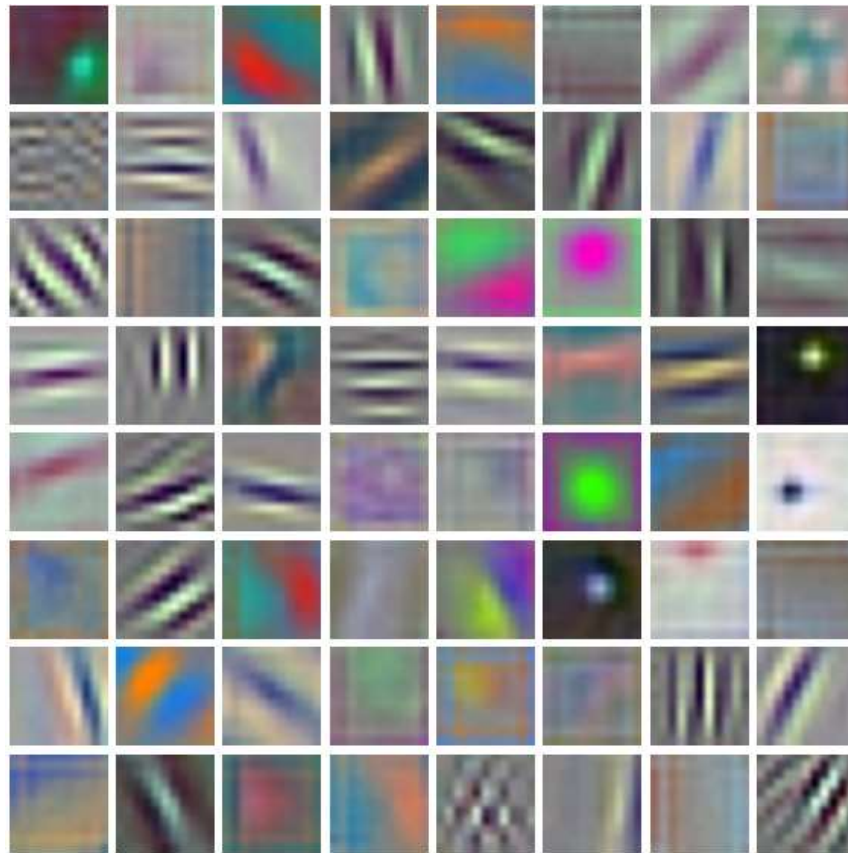
Illustration of a multi-dimensional filter that takes input from across the R, G, and B channels. The kernel here has 27 weights, 9×3 , plus a bias parameter not shown) and can be visualized as a $3 \times 3 \times 3$ tensor, with 3 channels, representable in colors:



NB: this holds for the first level of kernels. For higher level filters, the number of input channels is in general higher than 3, whence not represented directly

Visualizing filters (first convolutional layer)

Other filter extracted by the first convolutional level of AlexNet



Visualizing filters (other convolutional layers)

Filters in deeper levels cannot be visualized directly (more than 3 channels). In this case, visualize the input patch that maximizes activation for all neurons of a given feature map



Examples of image that produce the strongest activation in the hidden units in a network trained on ImageNet data. The top nine activations in each feature map are arranged as a 3×3 grid for four randomly chosen channels in each of the corresponding layers. We see a steady progression in complexity with depth, from simple edges in first layer to complete objects in last layer . [Zeiler and Fergus (2013)]

Other visualization techniques: saliency maps

Saliency maps can be used to identify those regions of an image that are most significant in determining the class label



Original image



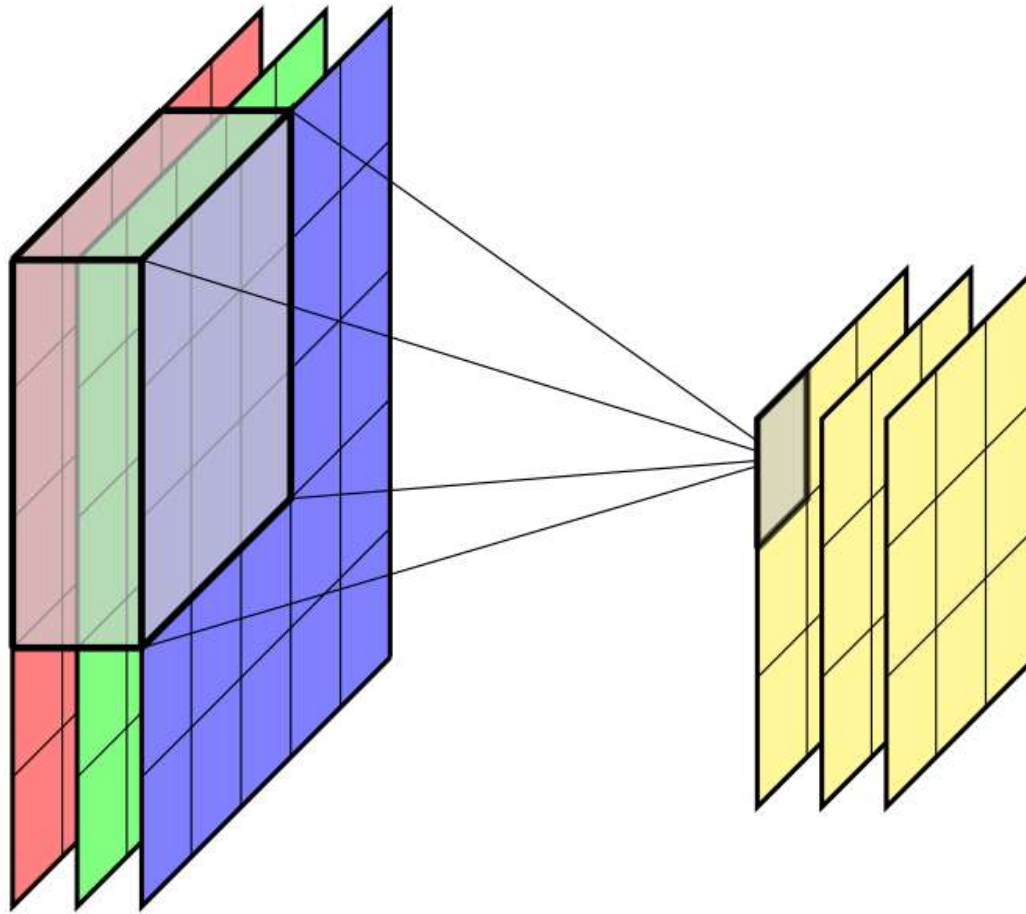
Saliency map for 'dog'



Saliency map for 'cat'

Saliency maps for the VGG-16 network with respect to the 'dog' and 'cat' categories.
[Selvaraju et al. (2016)]

Multidimensional convolution (multiple independent kernels, multiple feature maps)

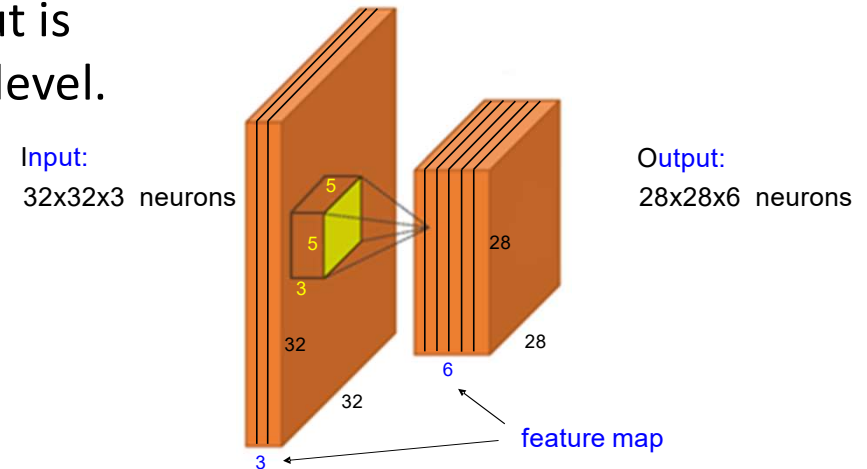


multiple independent filters lead to multiple feature maps, forming a single
“convolutional layer”.

Multidimensional convolution : how many weights

- The filter operates on a portion of the input volume. In the example each neuron in the volume of output is connected to $5 \times 5 \times 3 = 75$ neurons of the previous level.

- Each slice of neurons (same depth) denotes a feature map. In the example we have 6 feature maps (dimension 28×28) in the output volume



- Weights are shared in a feature map. Neurons in the same feature map process different portions of the input volume **in the same way**.
- Each feature map can be seen as the result of a specific filtering of input.

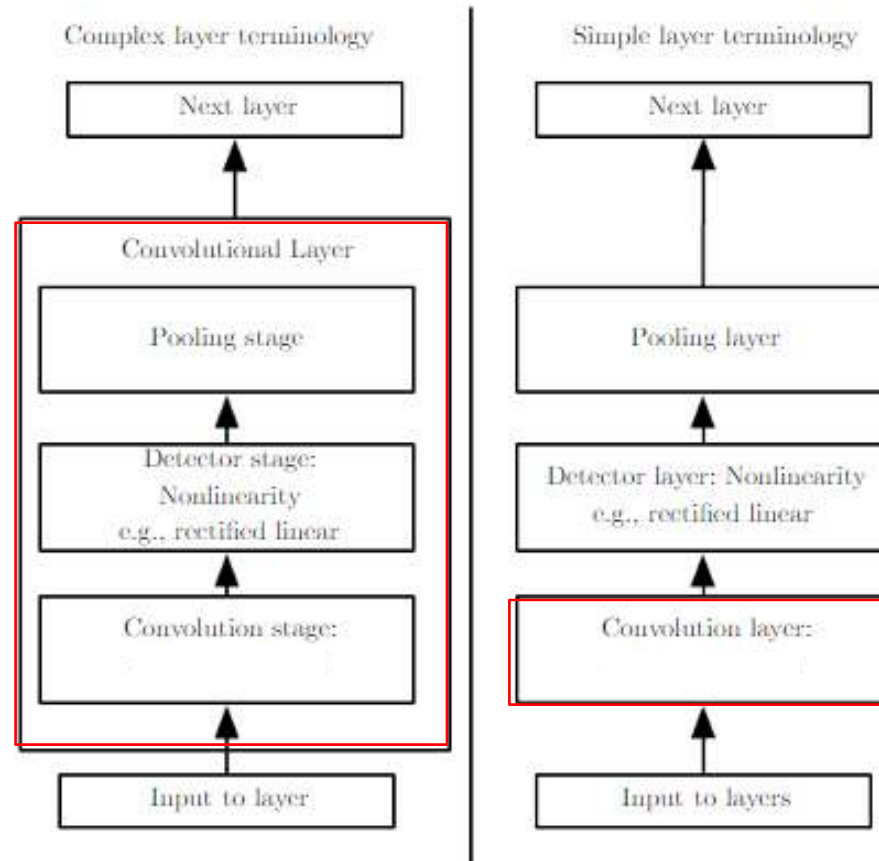
In the example the number of possible connections between the two layers

- is $(28 \times 28 \times 6) \times (5 \times 5 \times 3) = 352800$
but the total number of weights to be learned is $6 \times (5 \times 5 \times 3 + 1) = 456$. (due to weight sharing)

How many weights in an equivalent portion of MLP?

A typical layer of a convolutional network

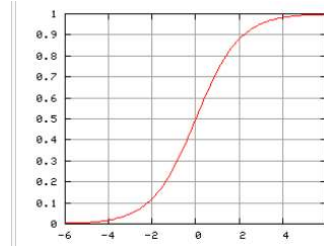
Two terminological variants:



Activation function: Relu

- MLP historically used the sigmoid activation function. In deep networks, the use of the sigmoid is problematic for back propagation (problem of vanishing gradient)

- Sigmoid for large and small values derivative is close to 0



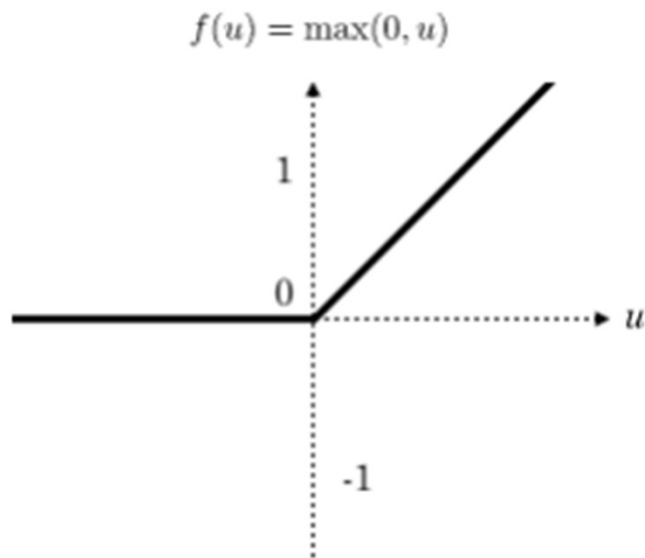
- In weight update:

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1}) \odot \sigma'(\mathbf{a}^l).$$

- If ϕ' is close to 0 \rightarrow product very small \rightarrow vanishing gradient!

Activation function: Relu

- To solve the problem we use Relu ([Rectified Linear](#)) as activation function:

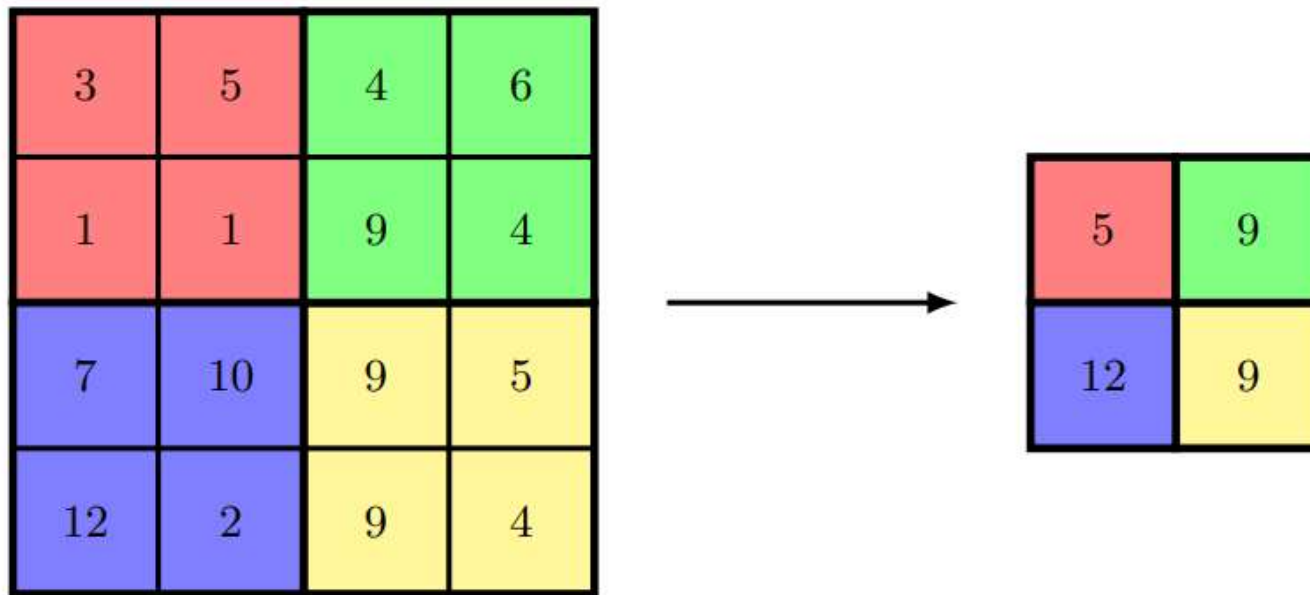


the derivative is 0 for
negative values and 1
for positive values

- This leads to sparse activations (part of the neurons are off) → simpler activation patterns → more generalizable
- unit acts as a **feature detector** that signals when it finds a sufficiently good match to its kernel

Pooling layer

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- E.g., the max pooling operation reports the maximum output within a rectangular neighborhood.



Pooling: types

Max pooling

3	3	2	1	0	0
0	1	3	1	3	1
2	2	3	2	0	0
2	2	2	0	0	0
1	0	1	3	2	2
3	2	0	3	0	1

3.0	3.0	3.0
2.0	3.0	0.0
3.0	3.0	2.0

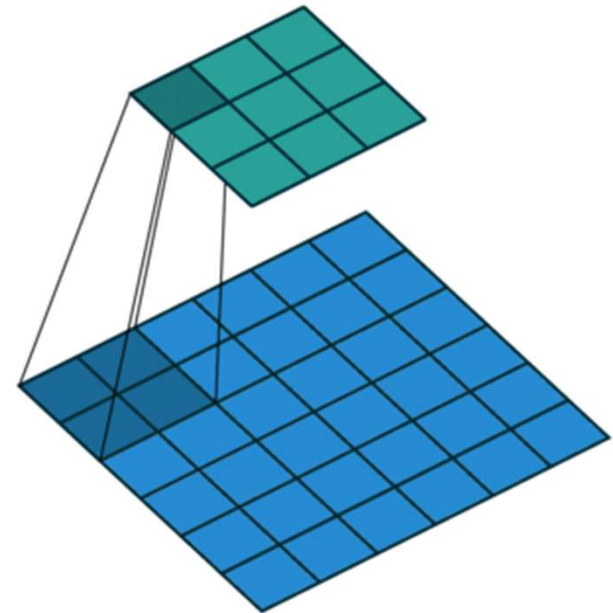
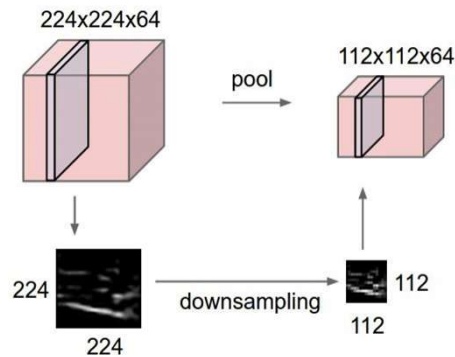
Average pooling

3	3	2	1	0	0
0	1	3	1	3	1
2	2	3	2	0	0
2	2	2	0	0	0
1	0	1	3	2	2
3	2	0	3	0	1

1.8	1.8	1.0
2.0	1.8	0.0
1.5	1.8	1.2

Pooling layer

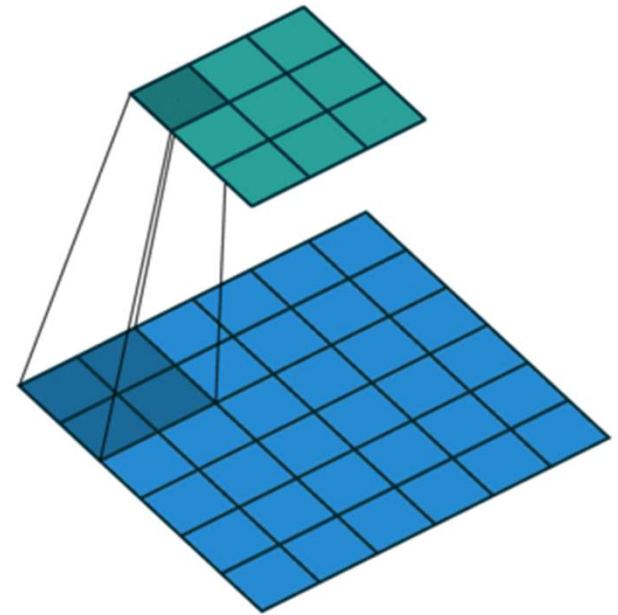
- **Pooling** reduces dimensionality (downsampling)



- **Pooling** selects the most informative values
- **Pooling** implements Invariance w.r.t. (small) translations
- Pooling layers have no weights!

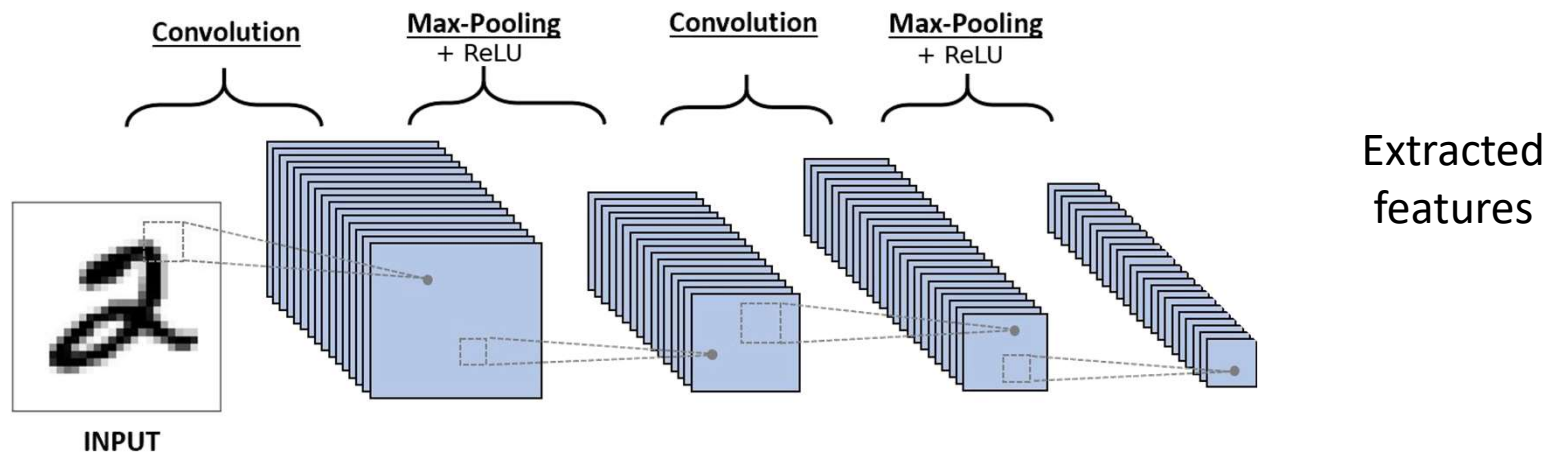
Pooling layer

- Pooling does not have learnable parameters but only hyperparameters (fixed with the architecture):
- A standard choice is max-pooling over 2×2 kernels with stride 2.
- In general, non-overlapping regions are pooled.
- Pooling is usually applied to each channel of a feature map independently.
- Less frequent: pooling can be applied across multiple channels of a feature map.



CNNs Overall Architecture

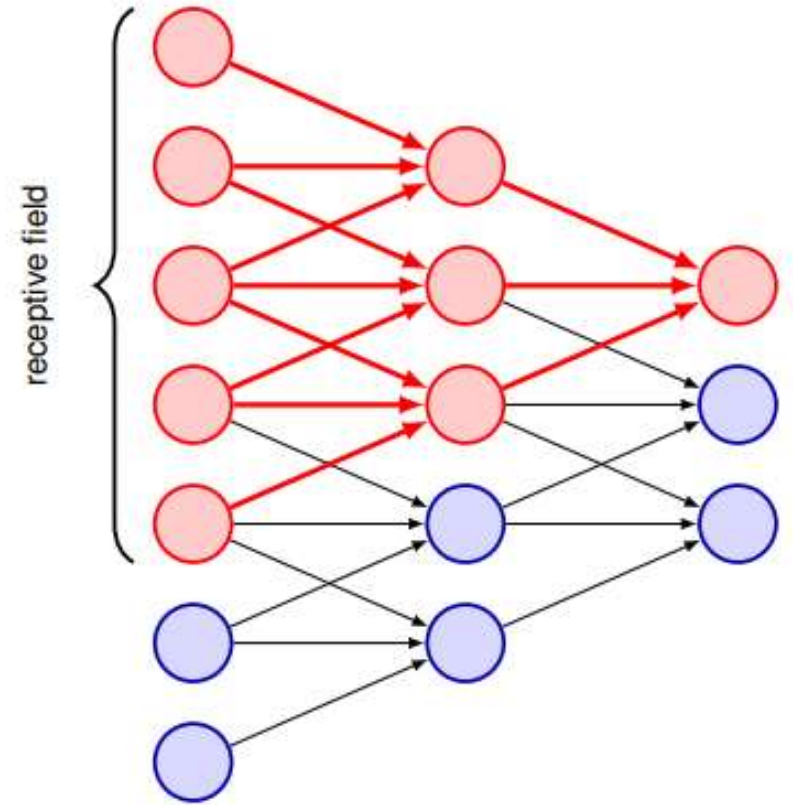
- CNNs contain multiple such layers of convolution and pooling in succession, in which the output channels of a particular layer, form the input channels of the next layer (analogously to RGB channels at the input level)
- One or more convolutional blocks extract progressively higher level features
- Example structure:



E.g., if previous level has n channel, each filter will have n channels (e.g., if feature maps of a previous convolutional level are 6, filters to the subsequent level might have dimension $5 \times 5 \times 6$)

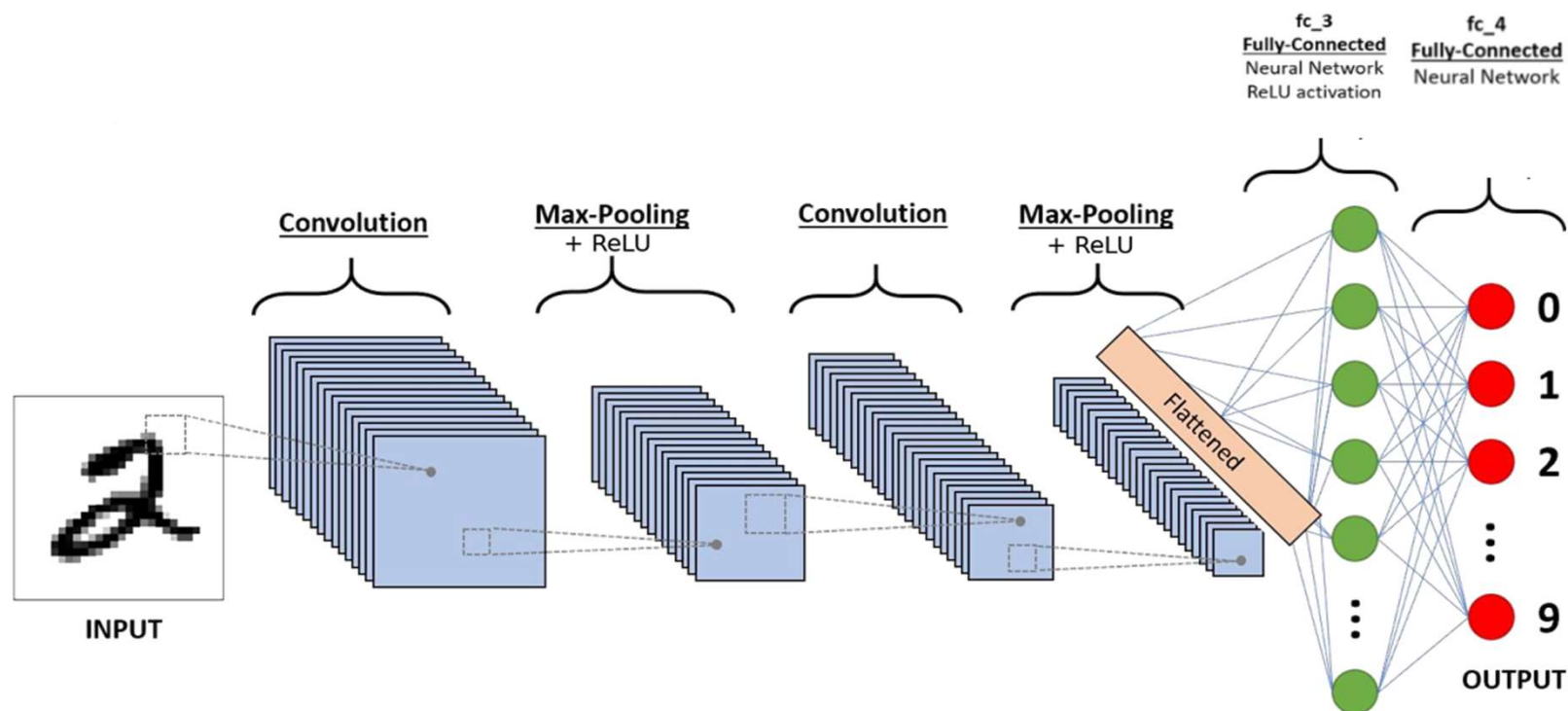
Multilayer Convolutions

When there are multiple layers of convolution, the effective receptive fields with depth in a multilayer convolutional network.



CNNs Overall Architecture

- After Convolutional blocks there are one or more **Fully Connected** layers



COST FUNCTION

Final output: SoftMax

Some common choices in modern CNN (when used as classifiers) are:

1) The use of a final layer of **SoftMax** function:

It consists of s neurons (one for each class) fully-connected to the neurons of the previous layer.

The neuron inputs v_k are calculated in the usual way, but as activation function for neuron k -th we use:

-

$$z_k = f(v_k) = \frac{e^{v_k}}{\sum_{c=1 \dots s} e^{v_c}}$$

the values z_k can be viewed as probabilities: they belong to $[0 \dots 1]$ and their sum is 1.

Loss function: Cross-Entropy

=2) Loss function = multi-class **Cross-Entropy** (instead of mean squared error, already introduced for MLP):

The cross-entropy measures how much the predicted distribution p **differs** from the desired output.

$$\text{Cross-Entropy Loss} = - \sum_{i=1}^S y_i \cdot \log(p_i)$$

- y_i is the desired output for neuron i
- p_i is the predicted value

Backpropagation is then applied with this loss function

CNNs Training and Testing

- The model is trained on a **Training Set**
- After a given number of epochs, the model is evaluated on the **Validation Set** (to monitor learning, adjust hyperparameters, decide when to stop training for early stop if overfitting).
- The model is tested to see what it has learned and its ability to generalize to new data on the **Test Set**

Typical Data Split Proportions

1. Training Set: 70% to 80%
2. Validation Set: 10% to 15%
3. Test Set: 10% to 15%

ImageNet

- The development of more powerful convolutional networks was accelerated through the introduction of a large-scale benchmark data set: **ImageNet**: 14million natural images, hand labelled into 22,000 categories
- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** : 1,000 non-overlapping categories, 1.28 million training images, 50,000 validation images, and 100,000 test images



Loss function: Cross-Entropy

= Loss function = **Cross-Entropy** (instead of mean squared error, already introduced for MLP):

The cross-entropy measures how much the predicted distribution p **differs** from the desired output.

$$\text{Cross-Entropy Loss} = - \sum_{i=1}^S y_i \cdot \log(p_i)$$

- y_i is the desired output for neuron i
- p_i is the predicted value

Loss function: Cross-Entropy

= The use of **Cross-Entropy** as loss function (instead of mean squared error, already introduced for MLP):

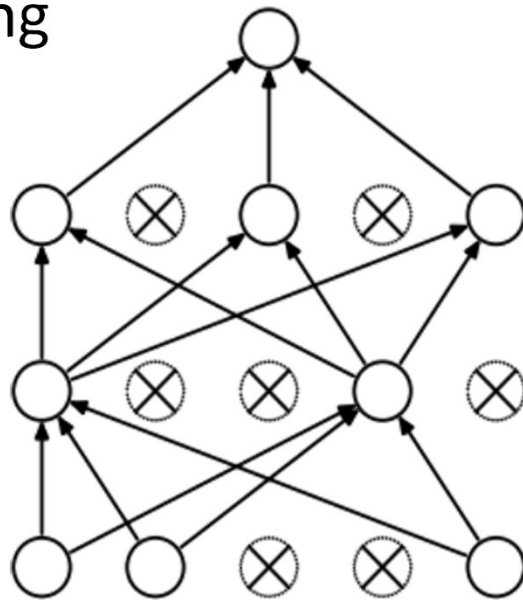
The cross-entropy between two discrete distributions p and q measures how much q **differs** from p : it is defined by

$$H(p, q) = - \sum_v p(v) \cdot \log(q(v))$$

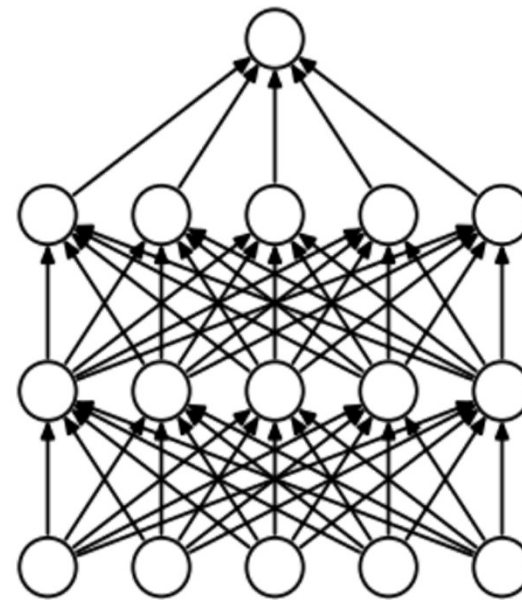
Techniques to avoid overfitting:

Dropout Regularization

- **During training**, some neurons are randomly ignored or “dropped out”, with a given probability p
- Speeds up training process
- Encourages sparsity, thus enhances generalization and avoids overfitting



Training



Test

Techniques to avoid overfitting:

Dropout Regularization

- Typically implemented as a “layer” that zeroes the previous neurons’ output with probability p
- **Placed after activation functions**
- You will need more – but faster – training iterations!

Other techniques to avoid overfitting:

Early Stopping:

- monitors the model's performance on a **validation set** during training and stops training when performance starts to degrade (indicating overfitting).
 - It is a way to keep the model from continuing to learn noise in the training data after the optimal point.
-
- Therefore there is the **Training Set** (80%), then the **Validation Set** (20%), on which training is not performed, used to assess generalization. Then a **Test Set**.

Other techniques to avoid overfitting:

Data Augmentation:

- Not a regularization technique in the traditional sense. It artificially increases the size of the training dataset by applying random transformations (like rotation, flipping, scaling) to the training images.
- This helps the model generalize better by exposing it to variations it might encounter in real-world scenarios.

Other techniques to avoid overfitting:

Batch Normalization:

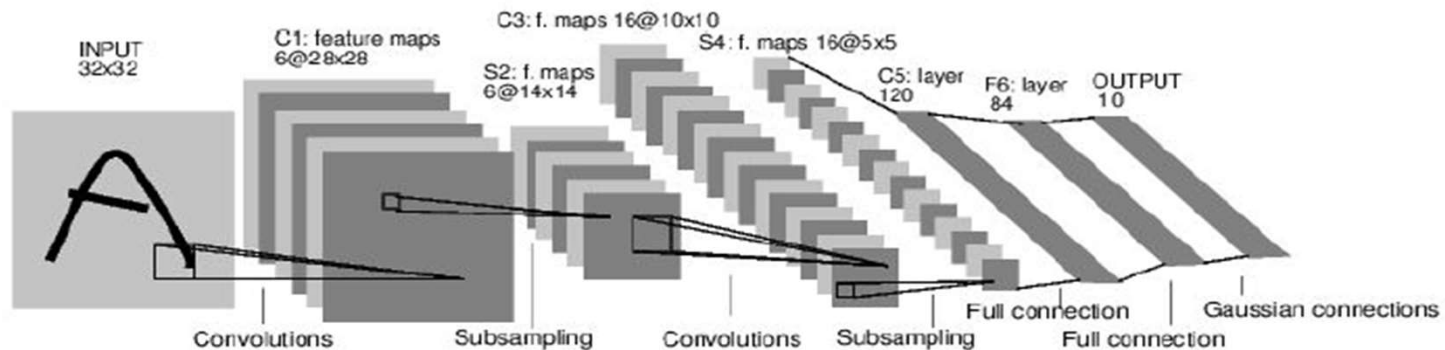
- It normalizes the output of each layer, which can reduce overfitting by introducing noise to the training process.

Weight Decay:

- adds a penalty to the loss function that encourages weights to be small, thus preventing overfitting.

Examples of CNNs

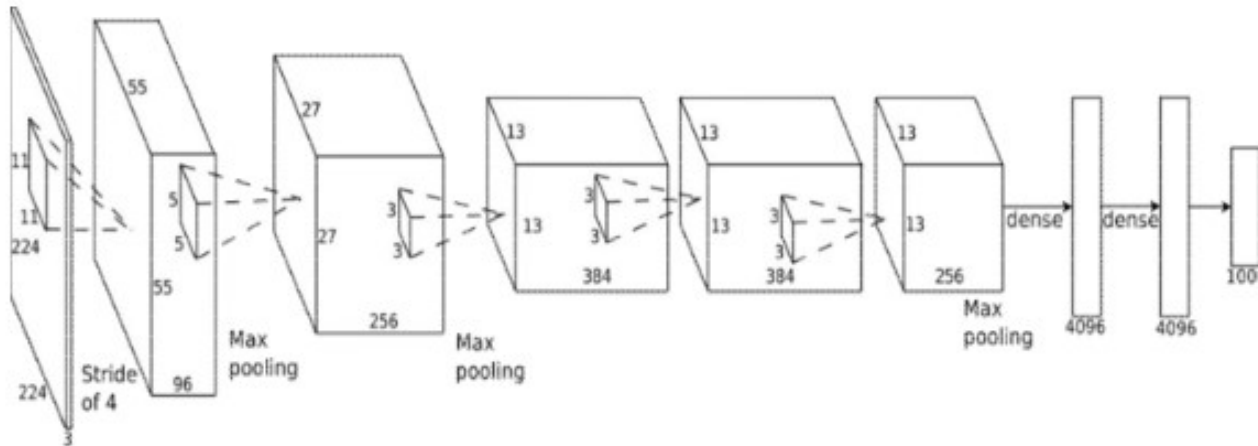
LeNet: Early example, used to classify low-resolution monochrome images of handwritten digits



[LeNet-5, LeCun 1980]

Examples of CNNs

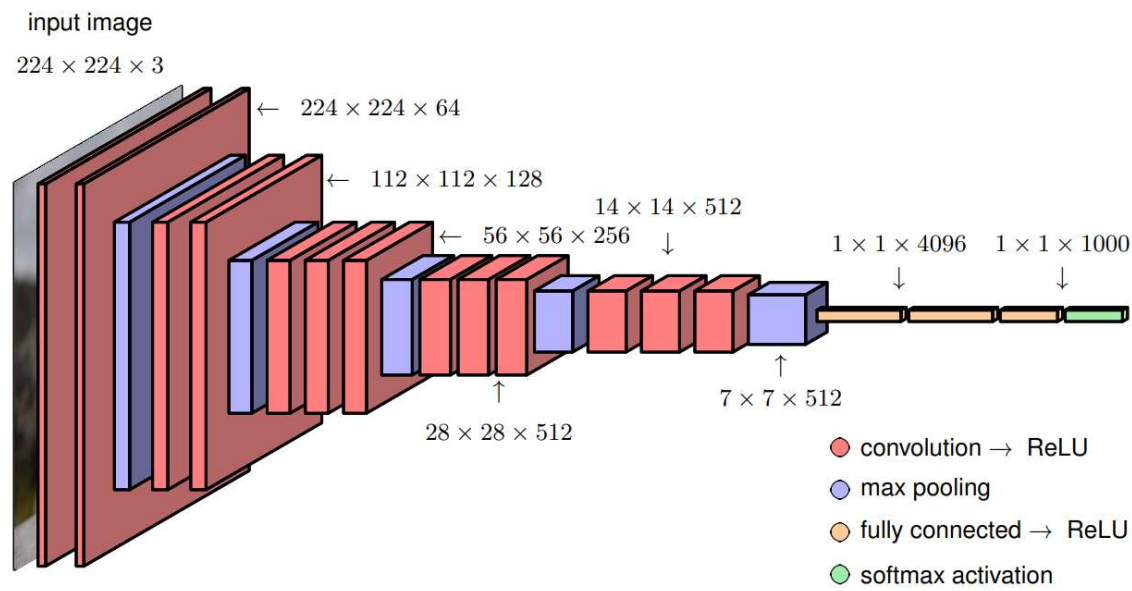
- **AlexNet** (Alex Krizhevsky, Sutskever, and Hinton, 2012) won the 2012 competition with top-5 error rate of 15.3%.



- Key aspects of this model:
- use of the ReLU activation function,
- the application of GPUs to train the network
- use of dropout regularization.

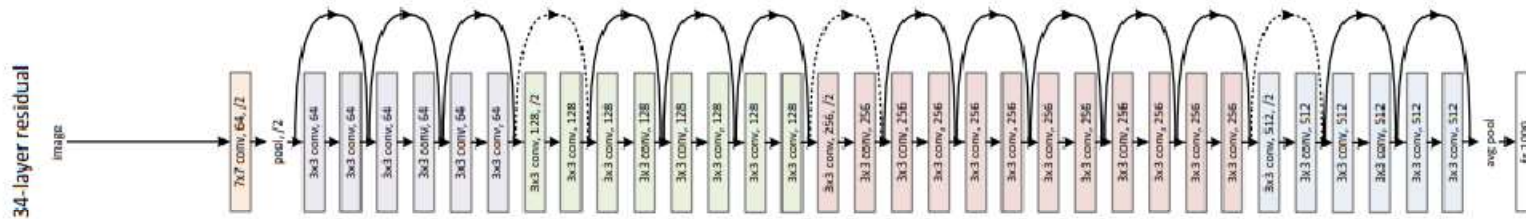
Examples of CNNs

VGG-16 model (Simonyan and Zisserman, 2014)



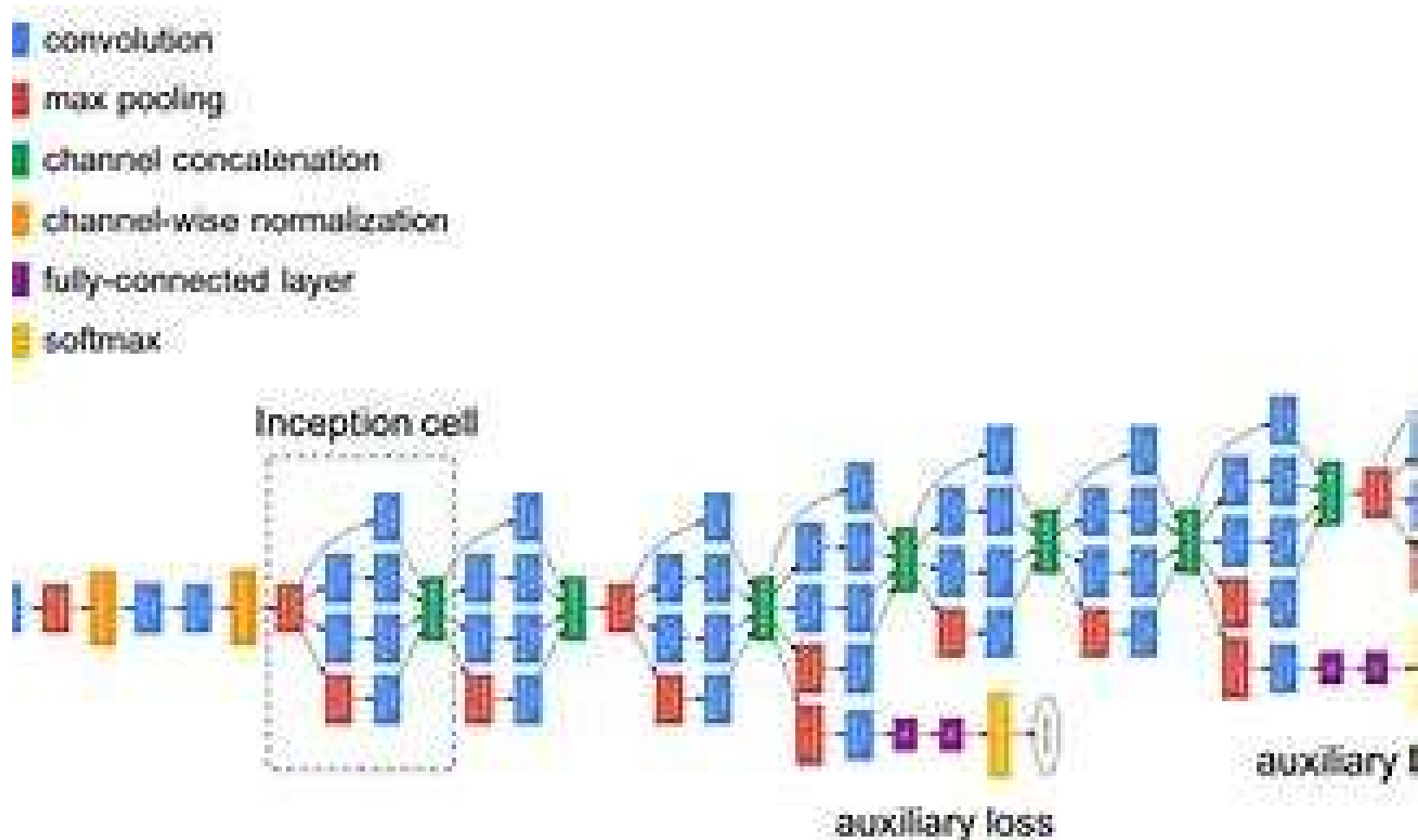
Examples of CNNs

ResNet Family



Examples of CNNs

Inception Family (GoogLeNet, V2, V3..)



Transfer Learning

Training and Transfer Learning

The training of complex CNN on large datasets (e.g. ImageNet) can require a lot of machine time even if executed on GPU.

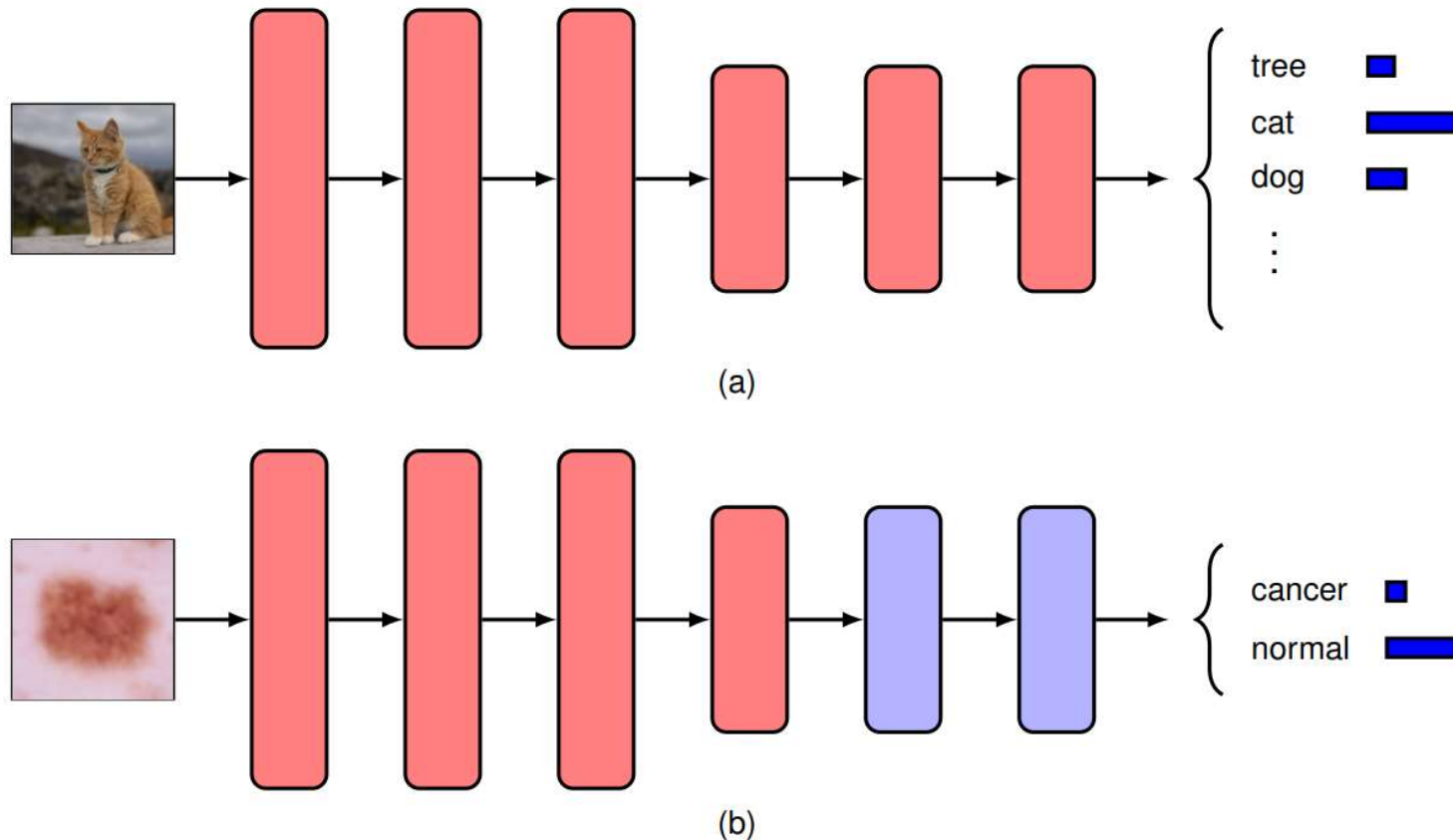
Fortunately, once the network has been trained, the classification of a new pattern is generally fast (e.g. 10-100 ms).

Transfer Learning:

Fine-Tuning:

- Start with a pre-trained network, trained on a similar problem and:
- we replace the output layer with a new layer of softmax neurons (also adapting the n.of classes).
- As initial values for weights we use those of the pre-trained network, except for connections between the penultimate and final layers whose weights are initialized random.
- we carry out new iterations of training (SGD) to optimize weights with respect to the peculiarities of the new dataset (it is not necessary that this one has large size).

Transfer Learning Example



(a) A network is first trained on a task with abundant data, such as object classification of natural images. (b) The early layers of the network (shown in red) are copied from the first task and the final few layers of the network (shown in blue) are then retrained on a new task such as skin lesion classification for which training data is more scarce

Training and Transfer Learning

Variant: reusing Features:

- we use an existing (pre-trained) network without further fine-tuning.
- We extract (at intermediate layers) the feature generated by the network during the forward step on the new dataset
- We use these features to train an external classifier to classify the patterns belonging to the new application domain.

Transfer learning - Images from different nature.

M2CAI (Modeling and Monitoring of Computer Assisted Interventions) Challenge



- Endoscopic videos (large intestine)
resolution of 1920 x 1080, shot at 25 frame per second at the IRCAD research center in Strasbourg, France. 27 training videos ranging from 15mn to 1hour, 15 testing videos
- Used for: monitor surgeons, Trigger automatic actions
- Objective: classification, 1 of 8 classes for each frame
TrocarnPlacement, Preparation, CalotTriangleDissection, ClippingCutting, GallbladderDissection, GallbladderPackaging, CleaningCoagulation, GallbladderRetraction
- Resnet 200 pretrained with ImageNet -> reaches 80% correct classification

Model	Type	Accuracy (%)
InceptionV3	Extraction (repres. of ImageNet)	60.53
InceptionV3	From Scratch (repres. of M2CAI)	69.13
InceptionV3	Fine-tuning (both representations)	79.06
ResNet200	Fine-tuning (both representations)	79.24

Table 2: Accuracy on the validation set.

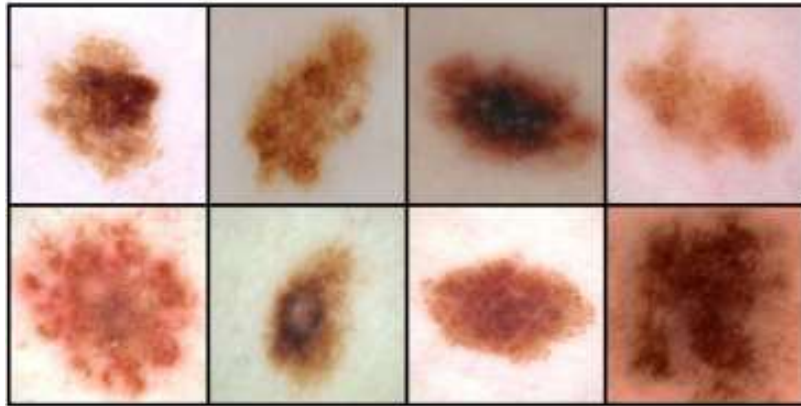
Transfer learning - Images from different nature, Plant classification (Wu 2017)

- Digitized plant collection from Museum of Natural History – Paris
- Largest digitized world collection (8 millions specimens)
- Goal
 - Identify plants characteristics for automatic labeling of worldwide plant collections
 - $O(1000)$ classes, e.g. opposed/alternate leaves; simple/composed leaves; smooth/with teeth leaves,
- Pretrained ResNet



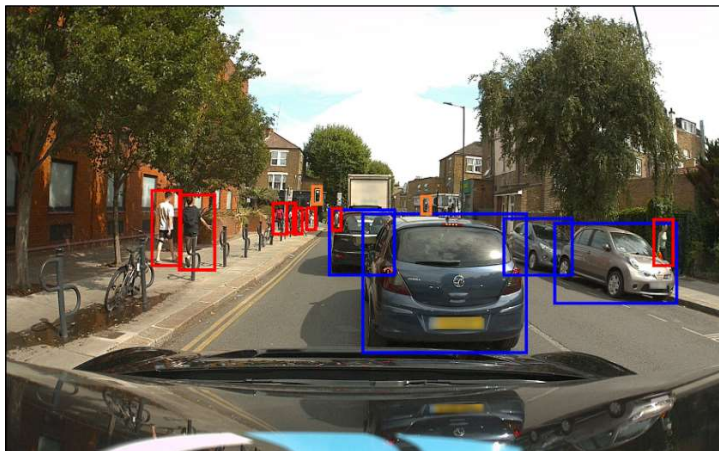
OTHER PROBLEMS AND HOW CNN ARE ADAPTED

- Other computer vision problems for which CNN are used:
- Classification of images (image recognition)



Dangerous malignant melanomas on the top row and benign nevi on the bottom row (difficult for the untrained eye to distinguish between these two classes).

- Object detection



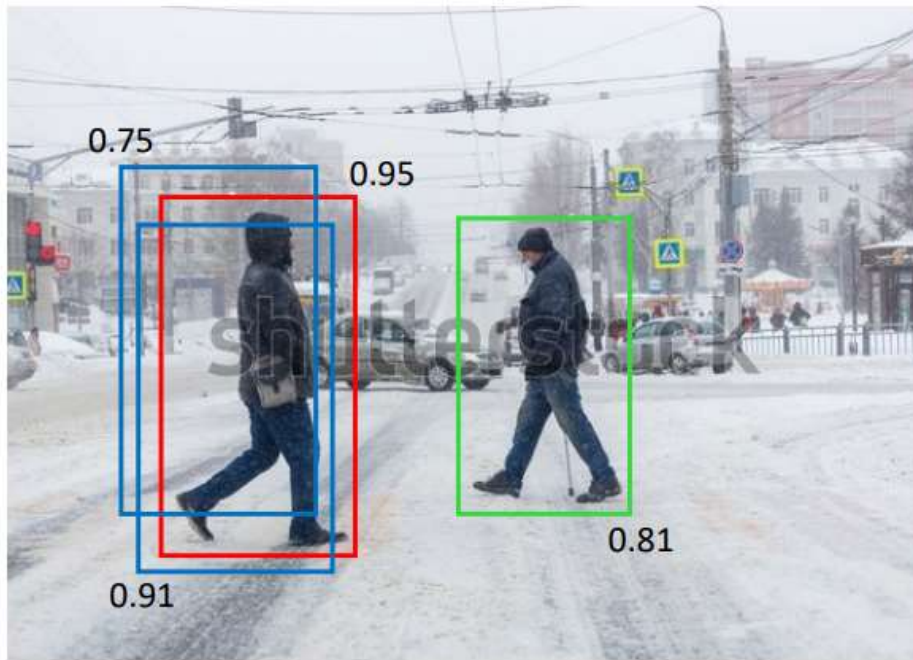
Data from an autonomous vehicle. Red boxes: 'pedestrian'; orange boxes: 'traffic light'; blue boxes: 'car',

- Other computer vision problems for which CNN are used:
- Image segmentation



Each pixel is classified individually dividing the image into regions sharing a common label. Here: blue pixels 'car'; red pixels 'pedestrian'; orange pixels 'traffic light'

- Other computer vision problems for which CNN are used:
- CNNs are adapted to the new task
- For instance, for object detection and localization, a possible approach is
- Use the trained model to detect objects in a new image by 'scanning' an input window across the image for each location
- When an object is detected with high probability, the associated window location then defines the corresponding bounding box.
- Fast R-CNN



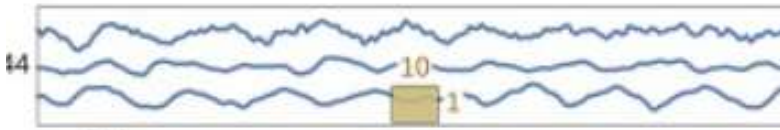
- Other computer vision problems for which CNN are used:

Neural Style Transfer ↴



- Other problems for which CNN are used:

- Although the architecture was originally developed in the context of image analysis, it has also been applied in other domains as analysis of sequential data (e.g., EEG)



- Recently alternative architectures based on transformers have become competitive with convolutional networks in some applications in image analysis

The Neuroscientific basis of Convolutional Neural Networks

A bit of history:

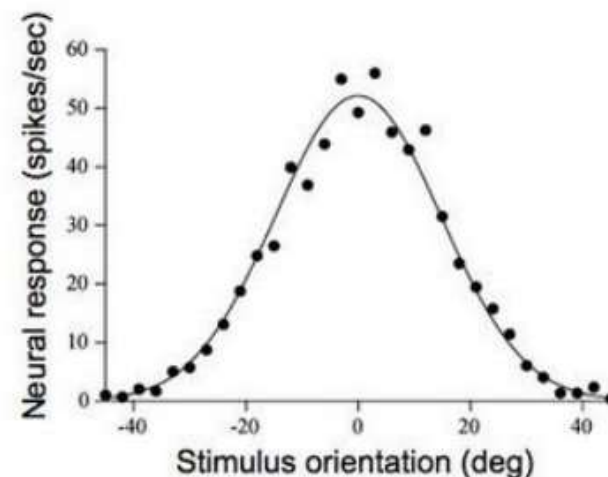
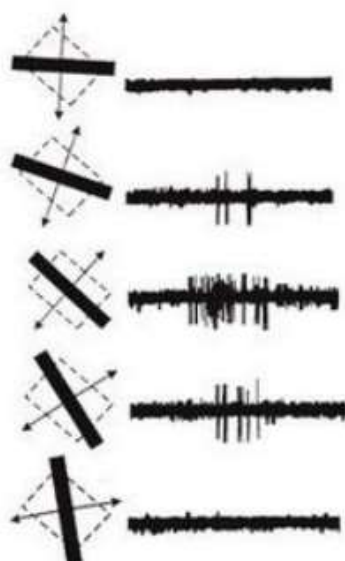
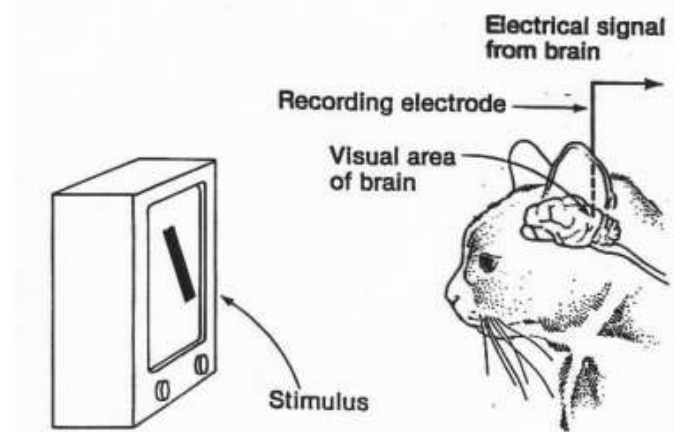
Hubel & Wiesel,
1959

RECEPTIVE FIELDS OF SINGLE
NEURONES IN
THE CAT'S STRIATE CORTEX

1962

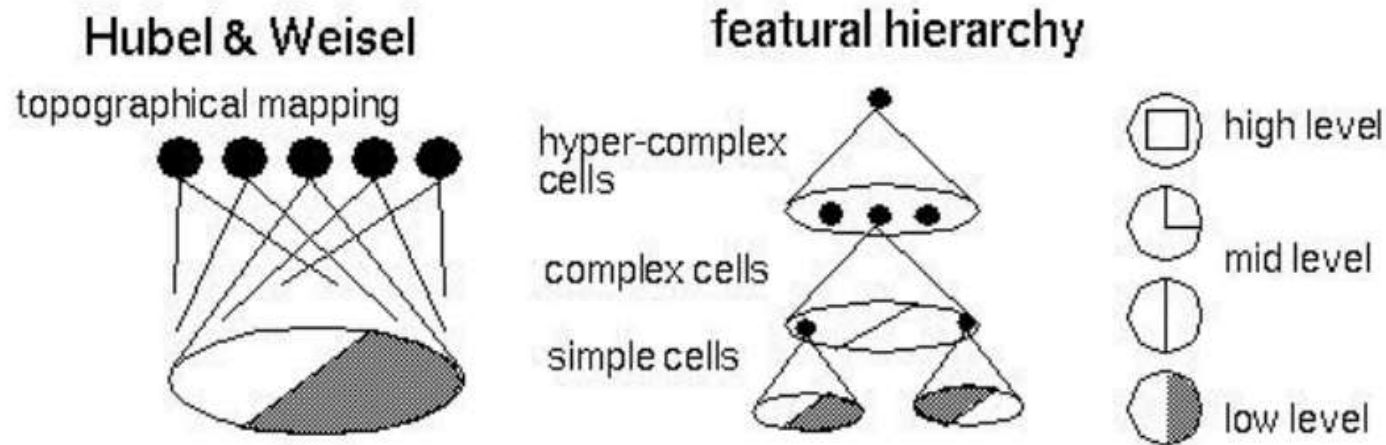
RECEPTIVE FIELDS, BINOCULAR
INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...

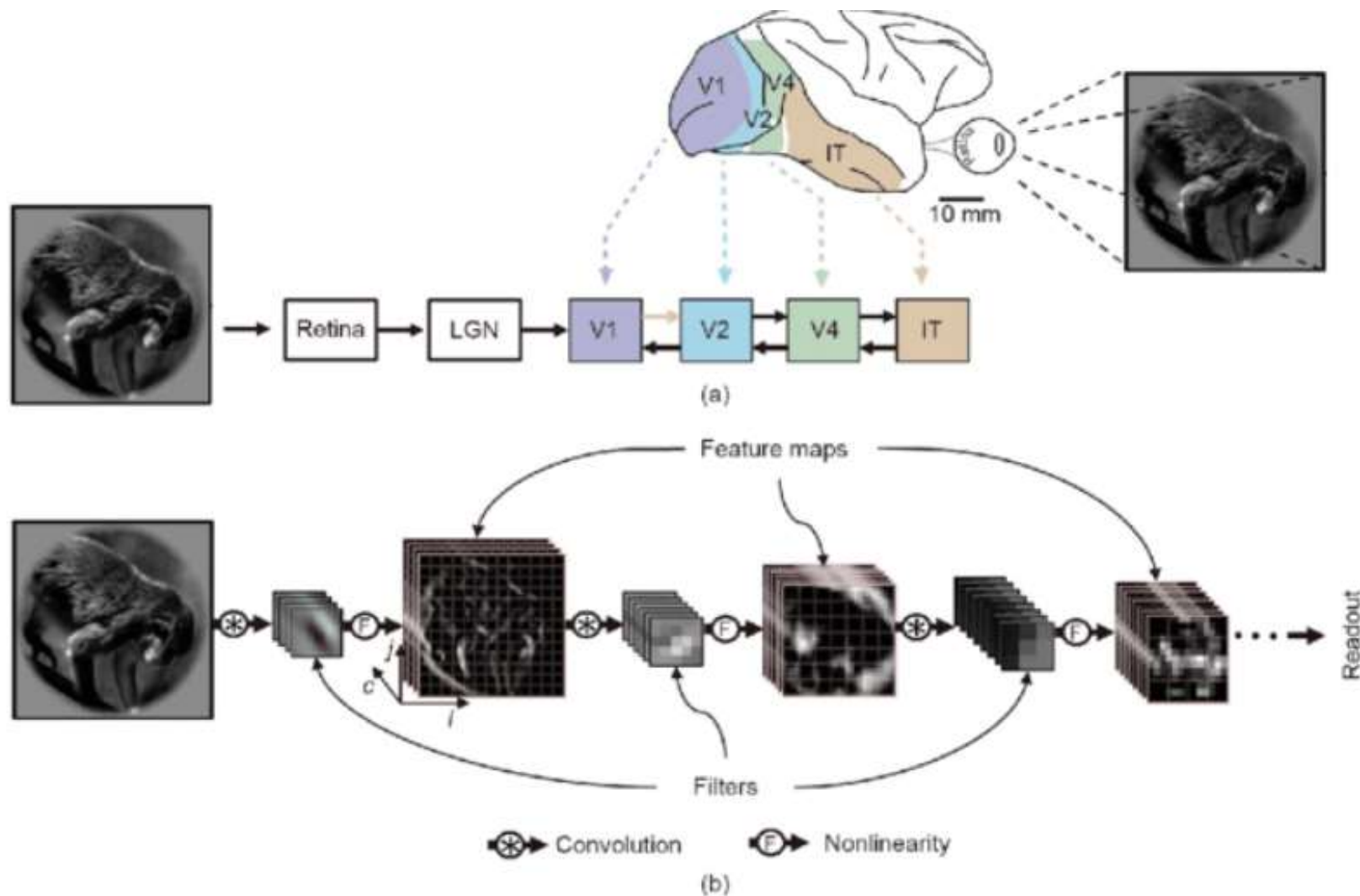


The Neuroscientific basis of Convolutional Neural Networks

Hierarchical organization

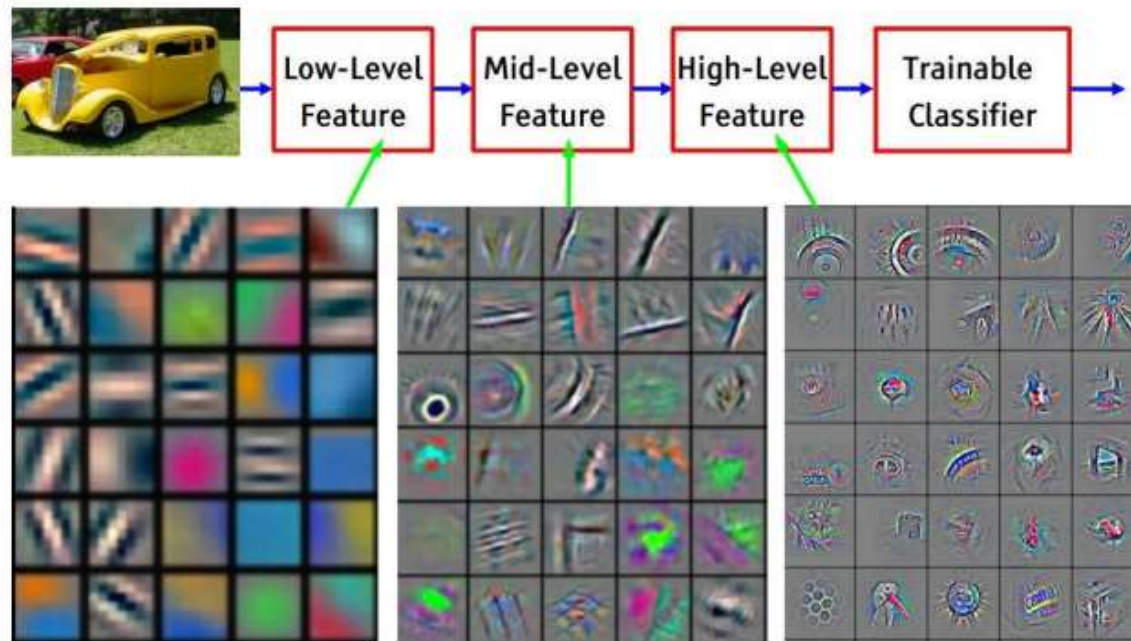


The Neuroscientific basis of Convolutional Neural Networks

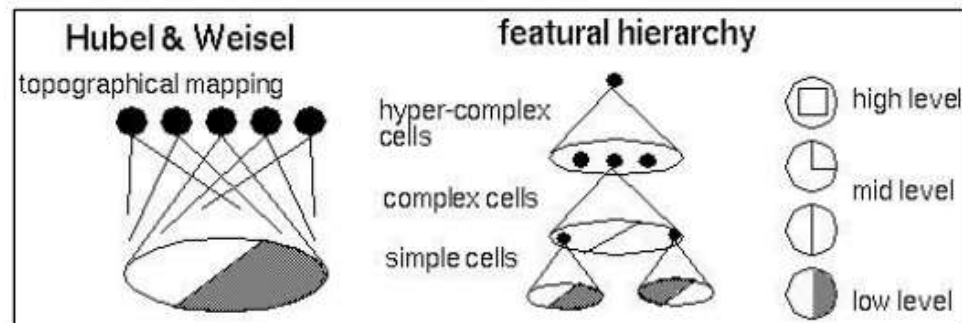


The Neuroscientific basis of Convolutional Neural Networks

Hierarchy of features



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



How to learn invariance?

- A particular object should be assigned the same classification irrespective of its position within the image. This is known as **translation invariance**.



How to learn invariance?

- A particular object should be assigned the same classification irrespective of its position within the image. This is known as **translation invariance**.



How to learn invariance?

- Also changes to the size of the object within the image should also leave its classification unchanged. This is called **scale invariance**.



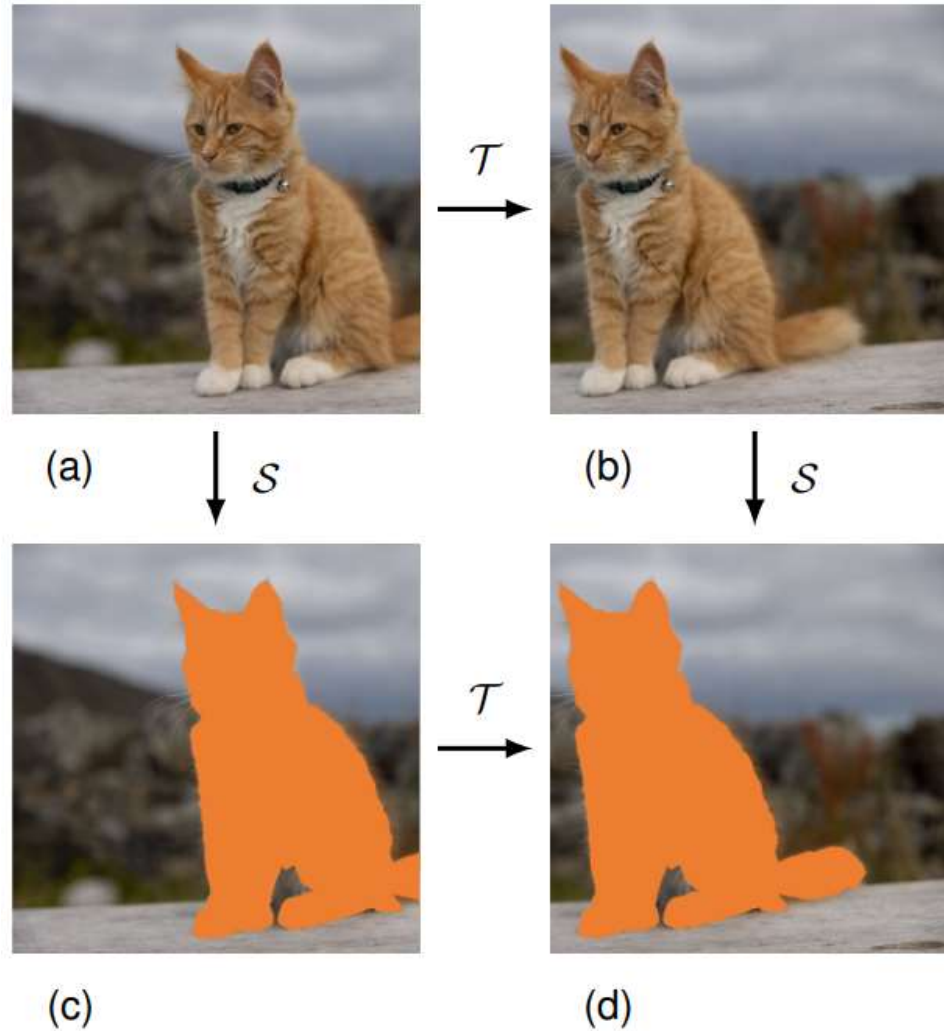
How to learn invariance?

- For multilayer perceptrons to learn invariance one would need to consider huge datasets (data augmentation)



Or equivariance?

- When the input is transformed, the output of the network is itself transformed in a specific way. E.g., $S(T(I)) = T(S(I))$ where S = segmentation, T = translation.



- If an image (a) is first translated to give (b) and then segmented to give (d), the result is the same as if the image is first segmented to give (c) and then translated to give (d)

- overfitting

Convolutional neural network (CNN):

- sparsely connected multilayer network
- with parameter sharing
- designed to encode invariances and equivariances specific to image data

CNN Architecture Overview

- • CNNs are designed to process grid-like data (e.g., images)
- • Key components: Convolution layers, Pooling layers, Fully connected layers
- • CNNs are widely used for image and video recognition tasks

Convolution Operation

- • Convolution applies filters (kernels) to input data
- • Filter detects features such as edges, textures, or objects
- • Stride and padding control how the filter moves over the input data

Example of Convolution

- • A 3x3 filter slides over a 5x5 input image
- • Result is a feature map with reduced dimensions
- • Convolution highlights specific features in the input

Activation Layer (ReLU)

- • ReLU (Rectified Linear Unit) is a common activation function in CNNs
- • ReLU introduces non-linearity to the model
- • ReLU outputs 0 for negative values, and the same value for positive inputs

Pooling Layer

- • Pooling reduces the dimensionality of the feature maps
- • Max pooling and average pooling are common pooling techniques
- • Max pooling takes the maximum value from a region of the feature map

Filters and Kernels in CNNs

- • Filters are used to extract features from input images
- • Different filters detect different features (edges, corners, etc.)
- • The filter matrix is convolved with the input to generate a feature map

Stride and Padding

- • Stride determines how far the filter moves across the input
- • Padding adds extra pixels around the edges of the input to control output size
- • Larger strides reduce the output size, while padding preserves more information

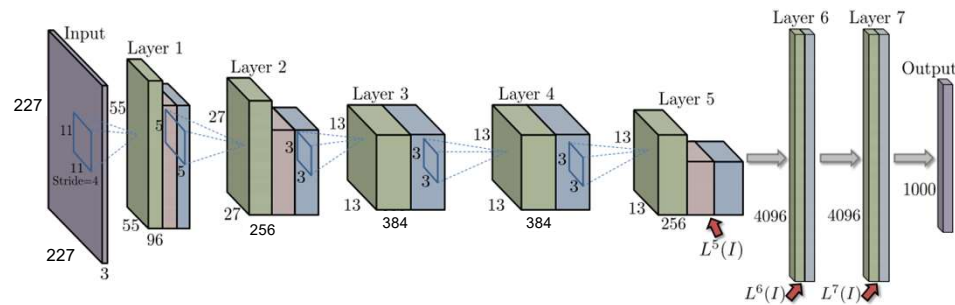
CNN Architectures: LeNet

- • LeNet is one of the earliest CNN models
- • Used for digit recognition tasks (MNIST dataset)
- • Consists of two convolutional layers, pooling, and fully connected layers

CNN Applications: Image Classification

- • CNNs excel in classifying objects in images
- • Examples: Identifying handwritten digits, animals, objects in photos
- • CNNs have state-of-the-art accuracy for image classification tasks

Summarising:



Colors

- Purple: Input (images 227x227x3) and Output (1000 classes)
- Green: Convolution
- Pink: Pooling (max)
- Blue: Relu (activation)

Note

- Layer 6, 7: Fully connected
- Layer 8: Softmax
- Stride: 4 in the first convolution layer, then 1
- Filters: Dimensions from 11x11 to 3x3
- Feature Maps: their number increases moving toward the output
- Total number of parameters: about 60M

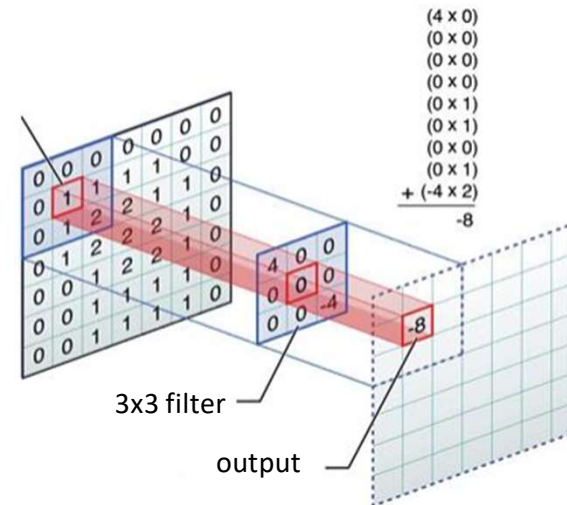
Convolution

A digital filter (a little weight mask 2D) slides on different positions of input; for each position is generated as output value the scalar product between the mask and the portion of the covered input (both treated as vectors).

Example ($x_1=1$, $x_0=0$):

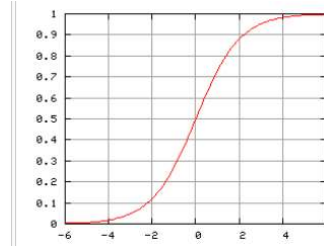
1 _{\times_1}	1 _{\times_0}	1 _{\times_1}	0	0
0 _{\times_0}	1 _{\times_1}	1 _{\times_0}	1	0
0 _{\times_1}	0 _{\times_0}	1 _{\times_1}	1	1
0	0	1	1	0
0	1	1	0	0

4		



Activation function: Relu

- MLP historically used the sigmoid activation function. In deep networks, the use of the sigmoid is problematic for back propagation (problem of vanishing gradient)
- Sigmoid for large and small values derivative is close to 0



- In weight update: $\Delta w_{ji} = \eta \delta_j y_i$

$$\delta_j = \begin{cases} \varphi'(v_j)(d_j - y_j) & \text{IF } j \text{ output node} \\ \varphi'(v_j) \sum_{\text{k of next layer}} \delta_k w_{kj} & \text{IF } j \text{ hidden node} \end{cases}$$

- If ϕ' is close to 0 \rightarrow product very small \rightarrow vanishing gradient!

In a Convolutional Neural Network (CNN) using **softmax** and **cross-entropy loss**, backpropagation involves calculating gradients for both the output and hidden units. Let's go through how these local gradients are calculated.

1.