

# Convolutional Neural Networks

Third lecture

E. Scornet

# Neural Network reborn

Renewed interest in 2006: ["A fast learning algorithm for deep belief nets", Hinton et al. 2006]

Propose a way to train deep neural nets:

- Train the first layer.
- Add a layer on top of it and train only this layer.
- Repeat the process until the network is deep enough.
- Use this network as a warm start to train the whole network.

Technical reasons for this new growing interest:

- Larger datasets
- More powerful computers
- Small number of algorithmic changes
  - ① MSE replaced by cross-entropy
  - ② ReLU (Fukushima, 1975, 1980)

# Using classical networks for images?

No, for two reasons:

- Do not take into account the spatial organization of pixels (if the pixels are permuted, the output of the network would be the same, whereas the image would change drastically)
- Non robust to image shifting

Idea:

- Apply local transformation to a set of nearby pixels (spatial nature of image is used)
- Repeat this transformation over the whole image (resulting in a shift-invariant output)

Not a new idea: trace back to perceptron and studies about the visual cortex of a cat.

The cat is able to

- detect oriented edges, end-points, corners (low-level features)
- combine them to detect more complex geometrical forms (high-level features)

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

## Convolutional neural networks (CNNs)

- Neural networks that use **convolution instead of matrix product** in one of the layers
- A CNN layer typically includes 3 operations: **convolution, activation and pooling**
- Using the more general idea of **parameters sharing**, instead of **full connection** (convolution instead of matrix product)

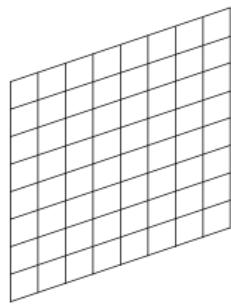
Convolution operator in neural networks is as follows

$$O(i, j) = (I \star K)(i, j) = \sum_k \sum_l I(i + k, j + l)K(k, l)$$

- $I$  is the input and  $K$  is called the kernels
- The kernel  $K$  will be **learned** (replaces the weights  $\mathbf{W}$  in a fully connected layer)

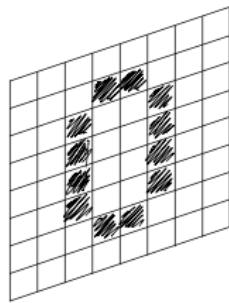
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)



## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)



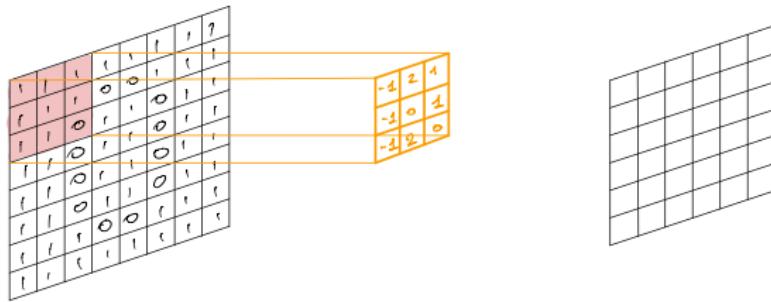
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)

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

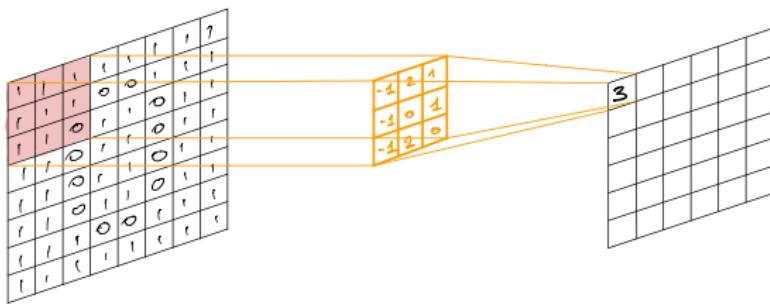
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



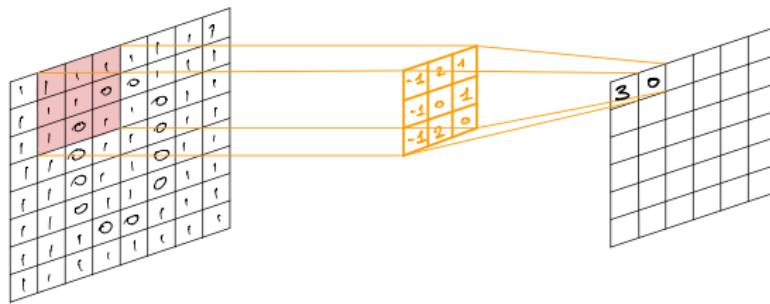
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



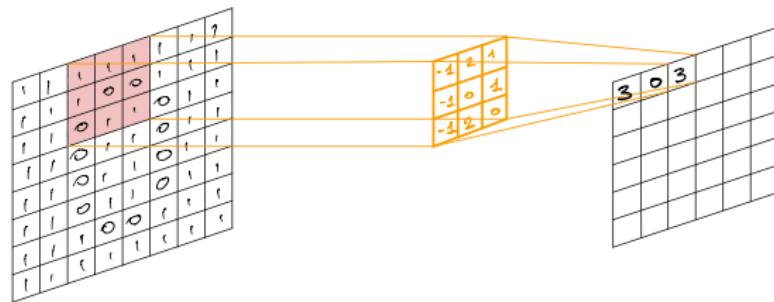
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



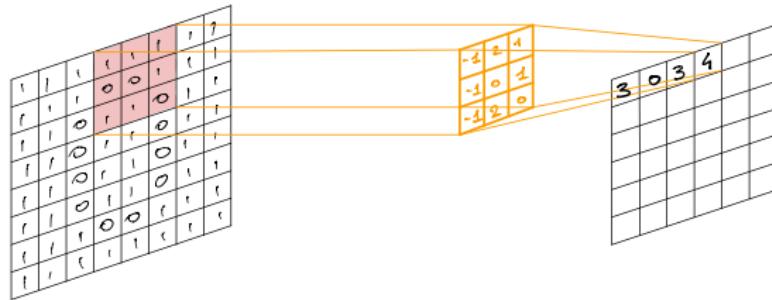
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



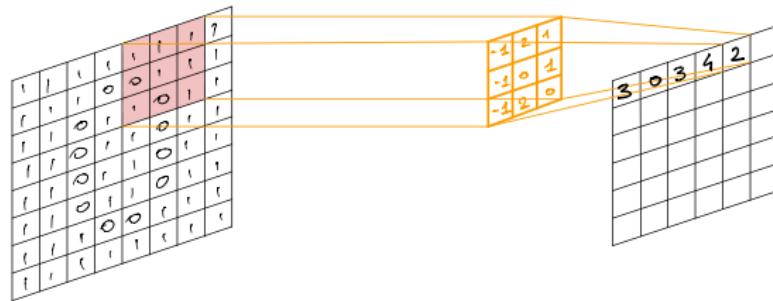
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



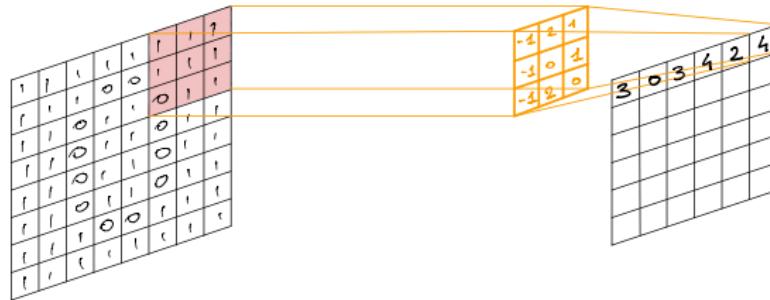
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



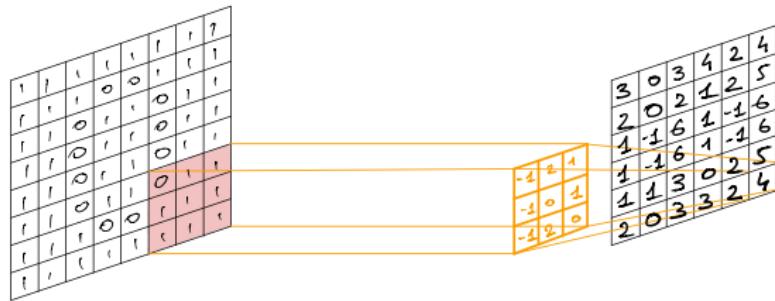
## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$



## Convolution - Black and White images

- Size of the input image is  $8 \times 8 \times 1$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 1$

Input

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

Output / Feature map

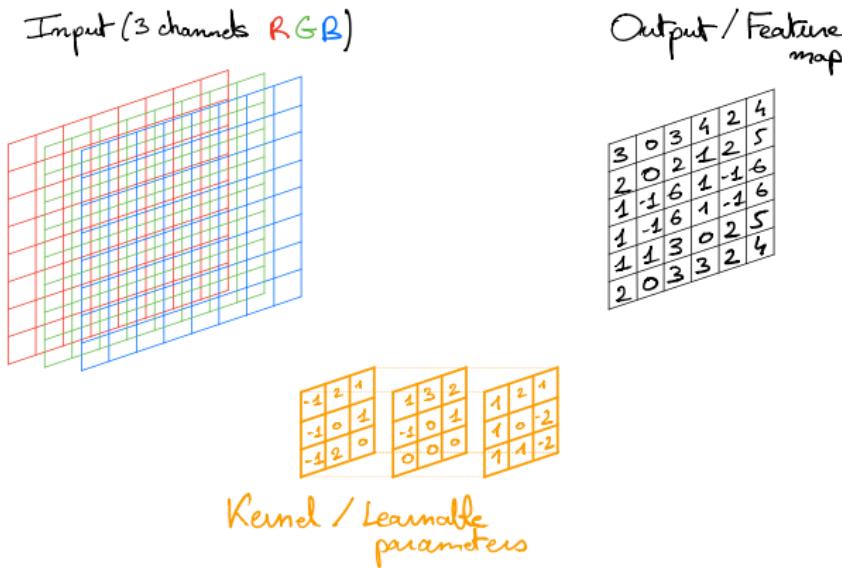
3	0	3	4	2	4
2	0	2	1	2	5
1	-1	6	1	-1	6
1	-1	6	1	-1	6
1	-1	6	0	2	5
2	1	3	3	2	4
2	0	3	3	2	4

Kernel / Learnable parameters

-1	2	1
1	0	1
-1	2	0

## Convolution - RGB

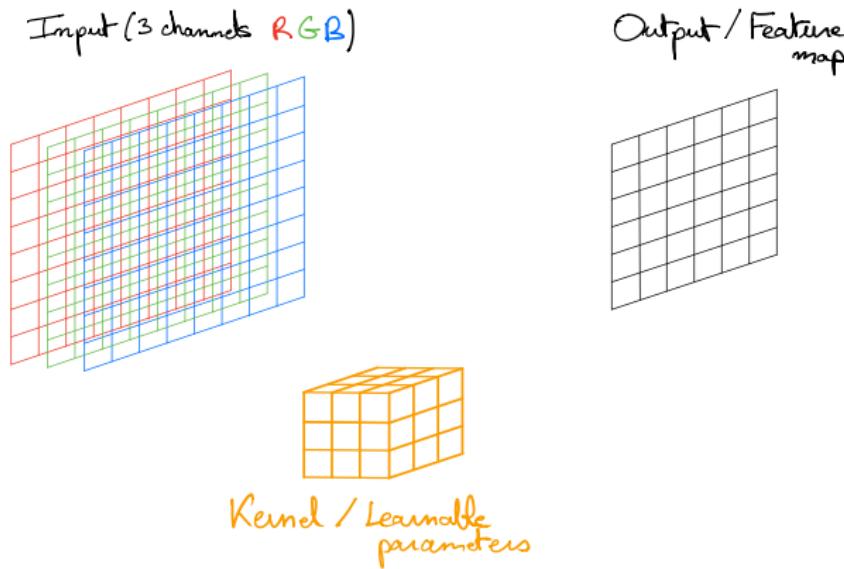
- Size of the input image is  $8 \times 8 \times 3$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 3$



Warning: every filter is small spatially (along width and height), but extends through the full depth of the input volume.

## Convolution - RGB

- Size of the input image is  $8 \times 8 \times 3$  (height, width, depth)
- Size of the kernel is  $3 \times 3 \times 3$

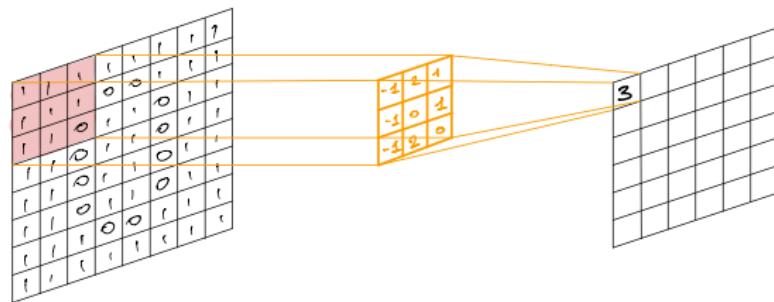


Warning: every filter is small spatially (along width and height), but extends through the full depth of the input volume.

## Parameters of convolutional layer 1/4

Four hyperparameters control the size of the output volume: the kernel size, the depth of the output volume, the stride and the zero-padding.

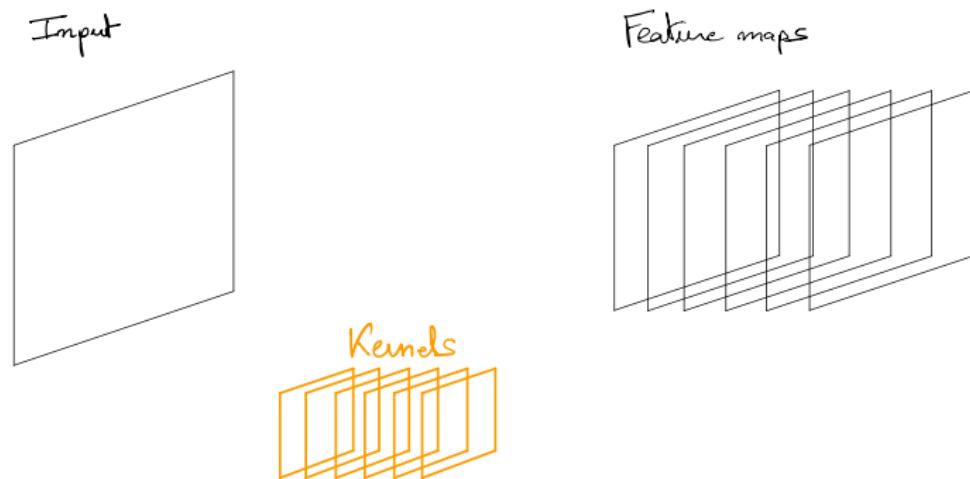
- The **size of the kernel** (typically  $3 \times 3$ ,  $5 \times 5$ ).



## Parameters of convolutional layer 2/4

Four hyperparameters control the size of the output volume: the kernel size, the depth of the output volume, the stride and the zero-padding.

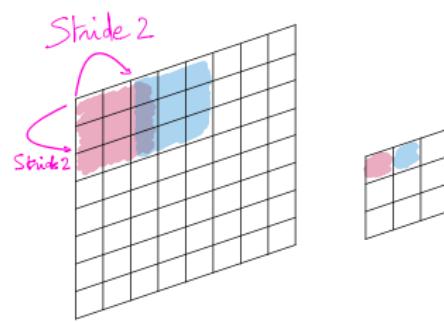
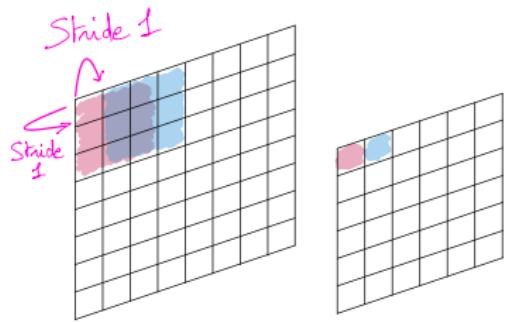
- The **size of the kernel**,
- The **depth of the output volume**, i.e., the number of filters/activation maps/feature maps.



## Parameters of convolutional layer 3/4

Four hyperparameters control the size of the output volume: the kernel size, the depth of the output volume, the stride and the zero-padding.

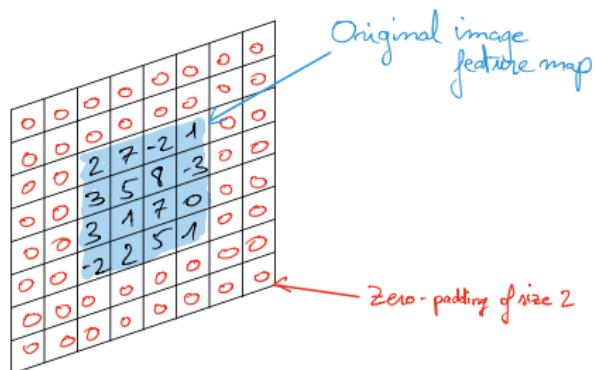
- The **size of the kernel**,
- The **depth of the output volume**,
- The **stride**, i.e., of how many pixels do we move the filter horizontally and vertically.  
Usually, stride is equal to one (rarely to two, and even more rarely larger).



## Parameters of convolutional layer 4/4

Four hyperparameters control the size of the output volume: the kernel size, the depth of the output volume, the stride and the zero-padding.

- The **size of the kernel**,
- The **depth of the output volume**,
- The **stride**,
- The **size of the zero-padding**, i.e. the number of zeros we add to the borders of the image. This can be used to obtain a constant image size between the input and the output.



## How to choose zero-padding?

Let

- $I$  the height/width of the input
- $O$  the height/width of the output
- $P$  the size of the zero-padding
- $K$  the height/width of the filter
- $S$  the stride

What is the relation between these quantities? How do we choose the zero-padding to obtain an output of the same size as the input?

## How to choose zero-padding?

Let

- $I$  the height/width of the input
- $O$  the height/width of the output
- $P$  the size of the zero-padding
- $K$  the height/width of the filter
- $S$  the stride

What is the relation between these quantities? How do we choose the zero-padding to obtain an output of the same size as the input?

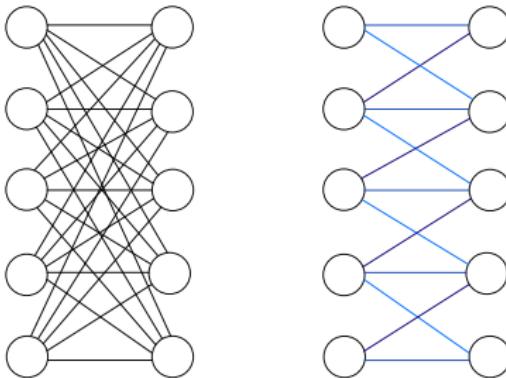
$$O = \left\lfloor \frac{2P + I - K}{S} \right\rfloor + 1$$

## Why convolution?

- Same transformation applied to all parts of the image (takes into account the spatial dependence between pixels and object-shift invariance)
- Input image contains millions of pixel values, but we want to detect small meaningful features such as edges with kernels that use only few hundred of pixels
- When using a matrix product, all input and output units are connected, whereas convolution connects only output neurons with several pixels of the input image.

Convolution involves weight sharing (a form of regularization) and requires less parameters which improves memory, is more statistically efficient and computationally faster.

## Sparse connections



- Left: when using matrix multiplication, all outputs are connected to all inputs. We say that **connectivity is dense**
- Right: in a convolution with a kernel of width 3, only three outputs are affected by the input  $x$ . We say that the **connectivity is sparse**

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.

3	0	3	4	2	4
2	0	2	1	2	5
1	-1	6	1	-1	6
1	-1	6	1	-1	6
1	1	3	0	2	5
2	0	3	3	2	4

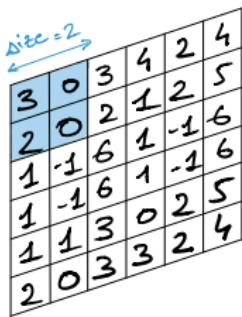
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



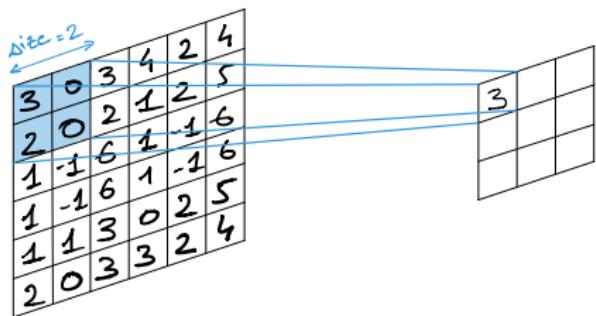
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



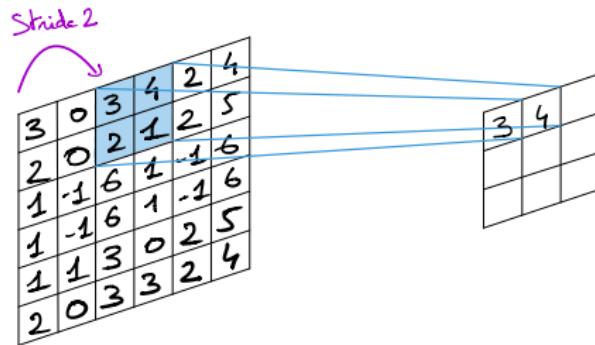
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



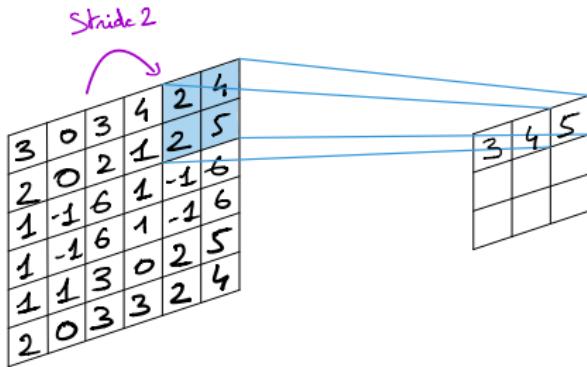
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



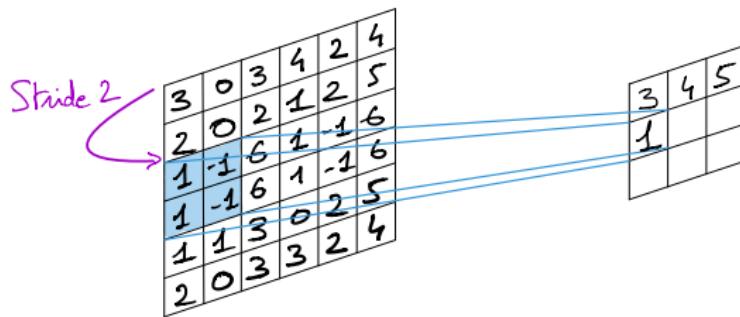
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



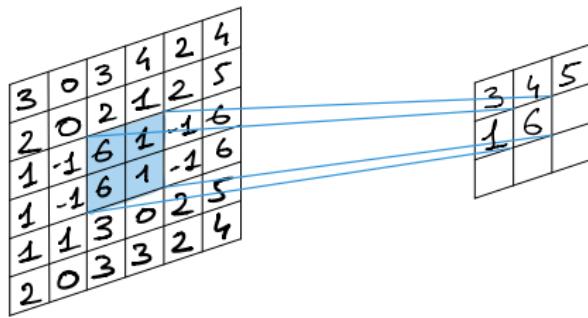
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



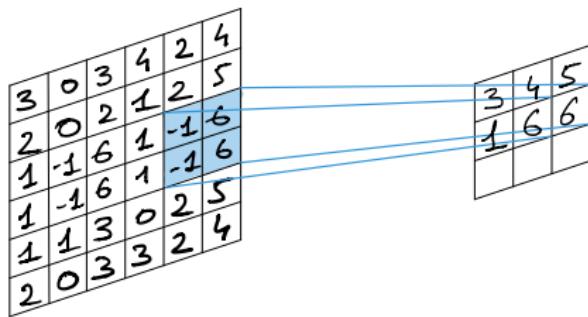
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



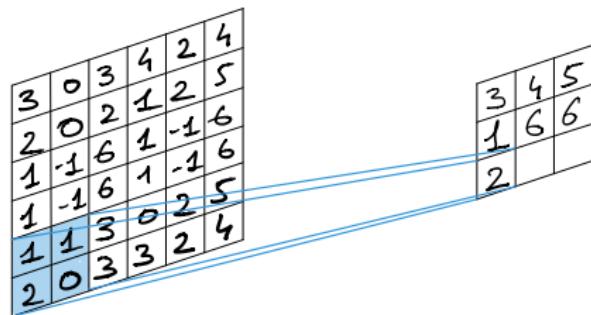
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



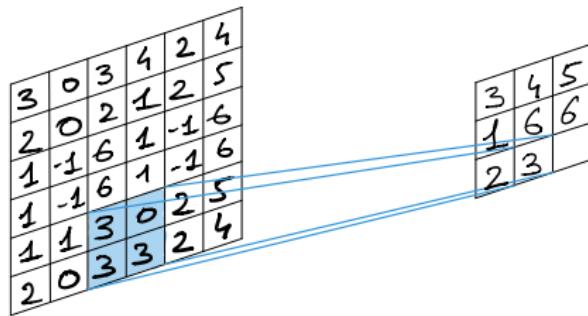
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



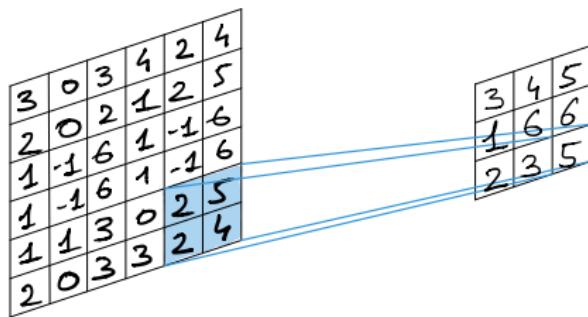
Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



Parameters:

- Stride  $S = 2$
- Spatial extend  $F = 2$

Usually,  $S = F = 2$  and more rarely  $F = 3, S = 2$  (overlapping pooling).

## Pooling

- Pooling layers compute each pixel of the output as a summary statistic of neighboring input pixels at the corresponding location.
- The most widely used is the max aggregation, called max-pooling
- Pooling helps the representation to become approximately invariant to small translations of the input
- If a small translation is applied, output of the layer is almost unchanged
- Very useful if we care more about the presence of some feature than its position in the image: for face detection (presence of eyes is more important than where they are)
- Pooling also allows to handle inputs with different sizes: pictures can have different sizes, but the output classification layer must be of fixed size

# A possible architecture of a CNN

Consider a grayscale image. Each kernel of the first layer produces one feature map.

Input

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

Output / Feature  
map

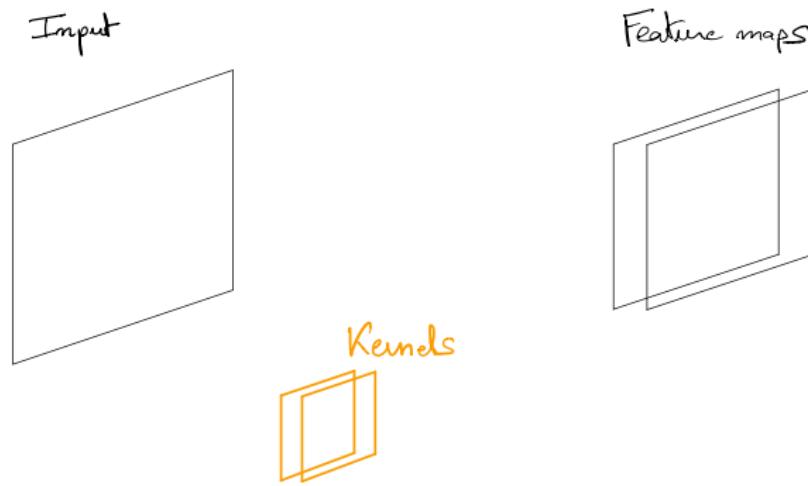
3	0	3	4	2	4
2	0	2	1	2	5
1	-1	6	1	-1	6
1	-1	6	1	-1	6
1	1	3	0	2	5
2	0	3	3	2	4

-1	2	4
-1	0	1
-1	0	0
-1	2	0

Kernel / Learnable  
parameters

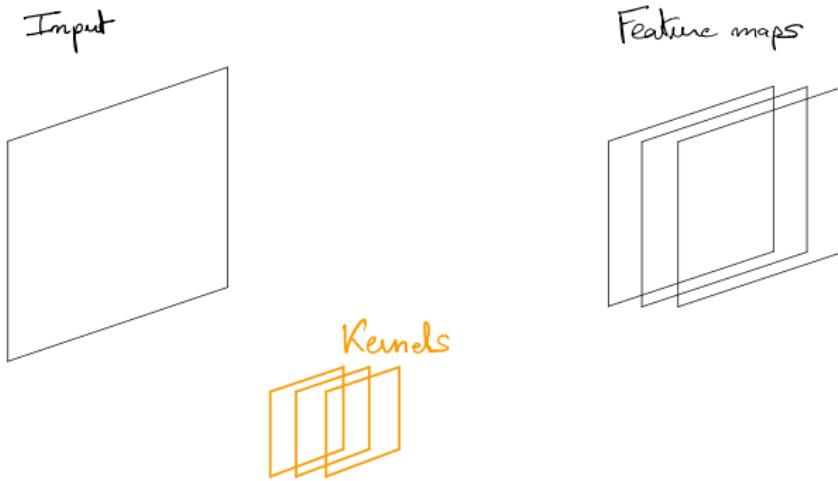
## A possible architecture of a CNN

Consider a grayscale image. Each kernel of the first layer produces one feature map.



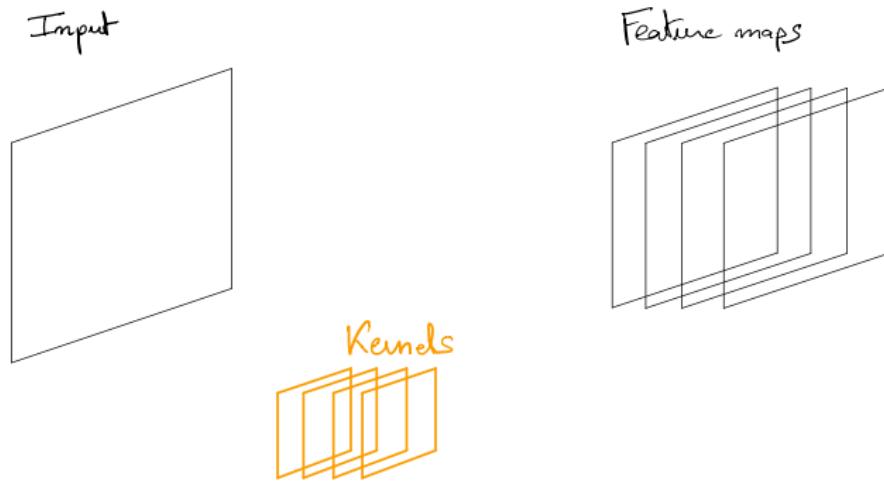
## A possible architecture of a CNN

Consider a grayscale image. Each kernel of the first layer produces one feature map.



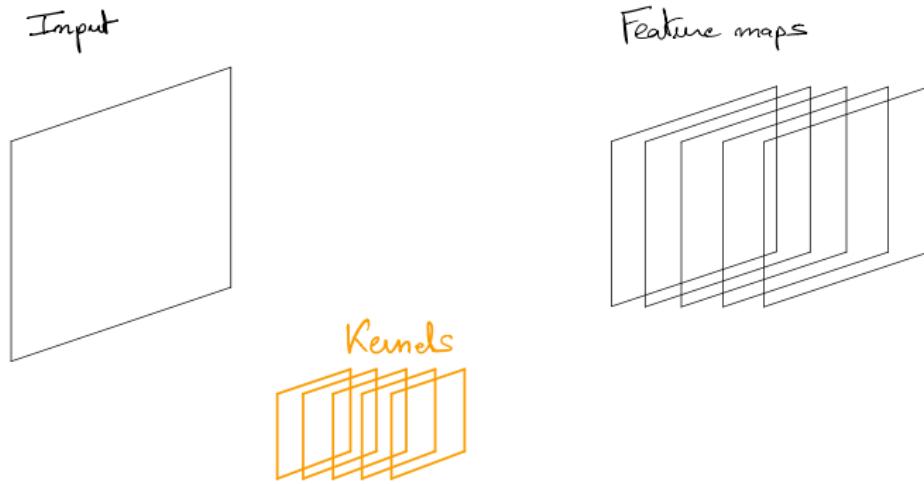
## A possible architecture of a CNN

Consider a grayscale image. Each kernel of the first layer produces one feature map.



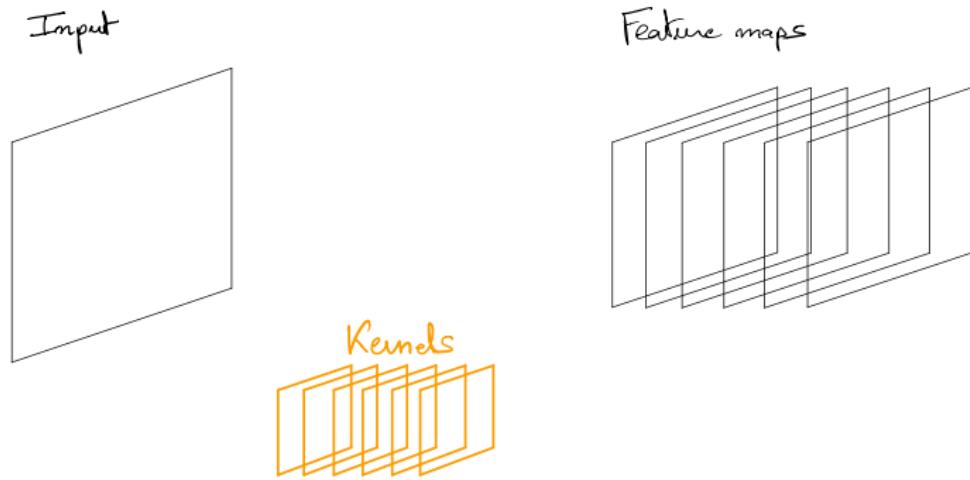
## A possible architecture of a CNN

Consider a grayscale image. Each kernel of the first layer produces one feature map.



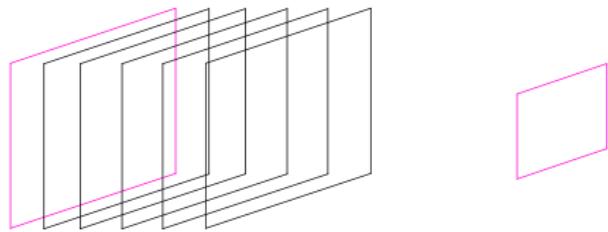
## A possible architecture of a CNN

Consider a grayscale image. Each kernel of the first layer produces one feature map.



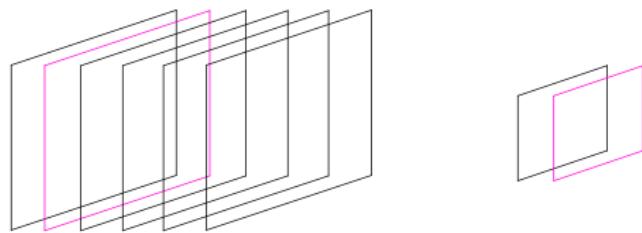
## A possible architecture of a CNN

The pooling layer operates on each feature map separately.



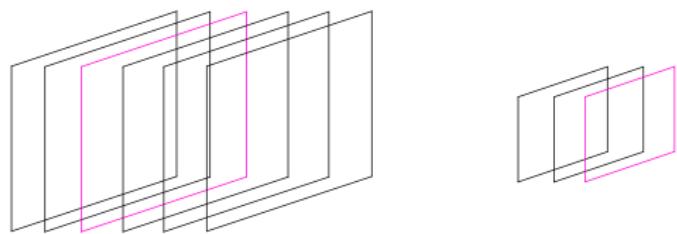
## A possible architecture of a CNN

The pooling layer operates on each feature map separately.



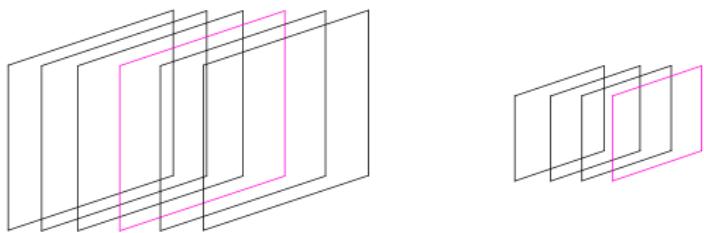
## A possible architecture of a CNN

The pooling layer operates on each feature map separately.



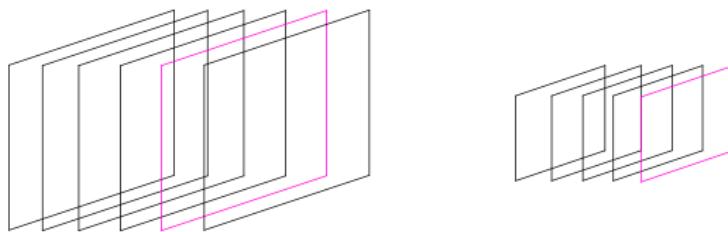
## A possible architecture of a CNN

The pooling layer operates on each feature map separately.



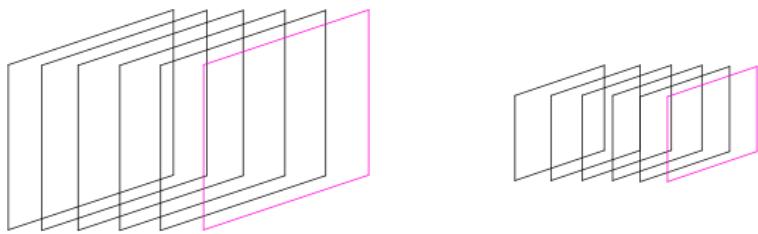
## A possible architecture of a CNN

The pooling layer operates on each feature map separately.

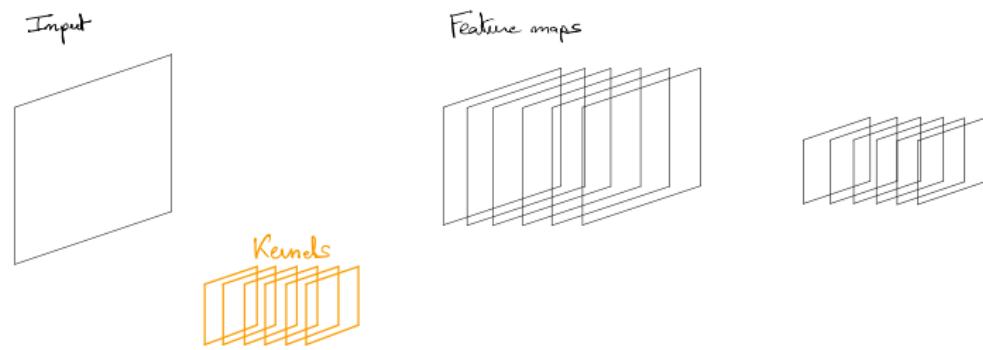


## A possible architecture of a CNN

The pooling layer operates on each feature map separately.

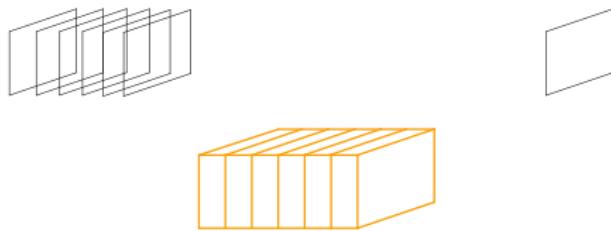


# A possible architecture of a CNN



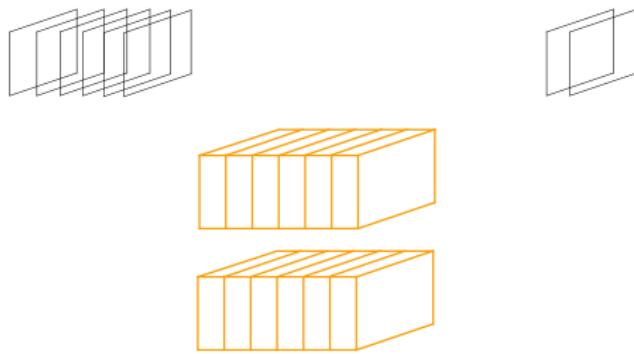
## A possible architecture of a CNN

A convolutional layer operates on the feature maps output by the pooling layer. Each kernel is a volume whose depth equals the depth of the input volume.



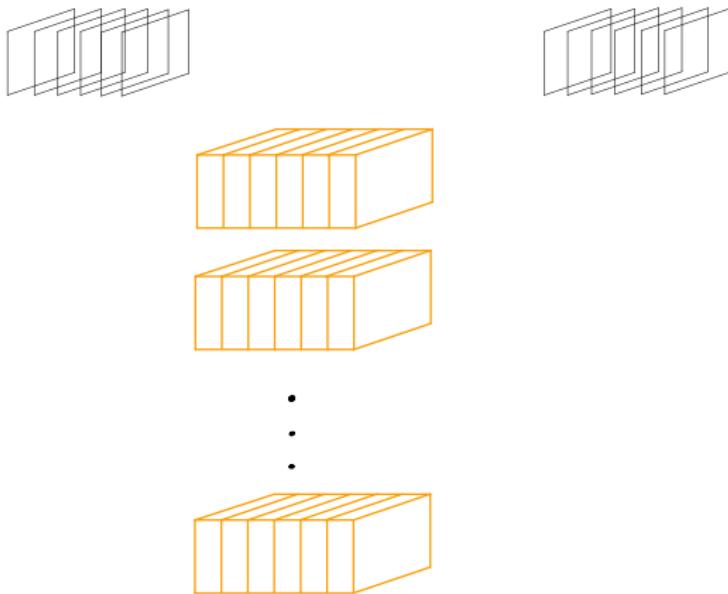
## A possible architecture of a CNN

A convolutional layer operates on the feature maps output by the pooling layer. Each kernel is a volume whose depth equals the depth of the input volume.

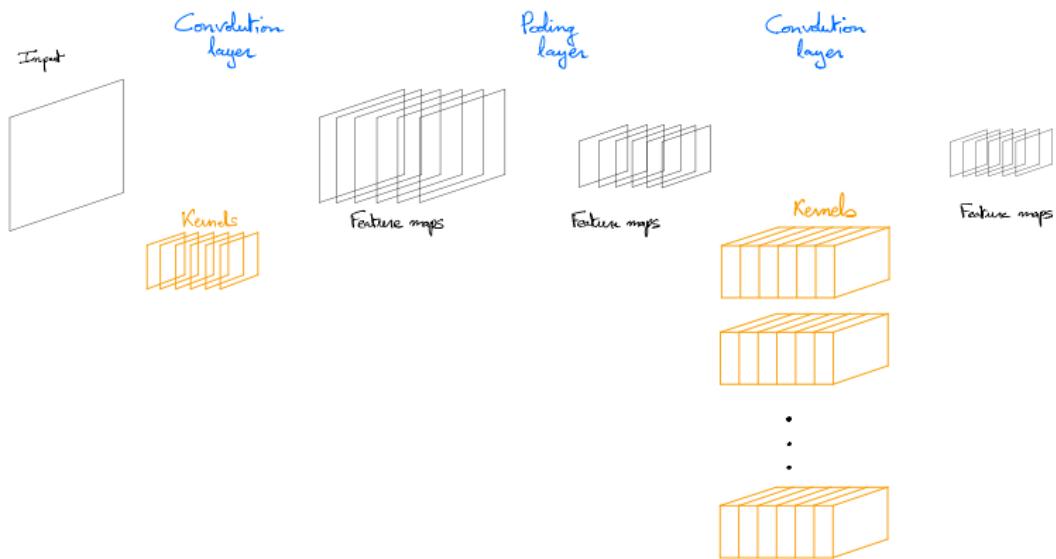


## A possible architecture of a CNN

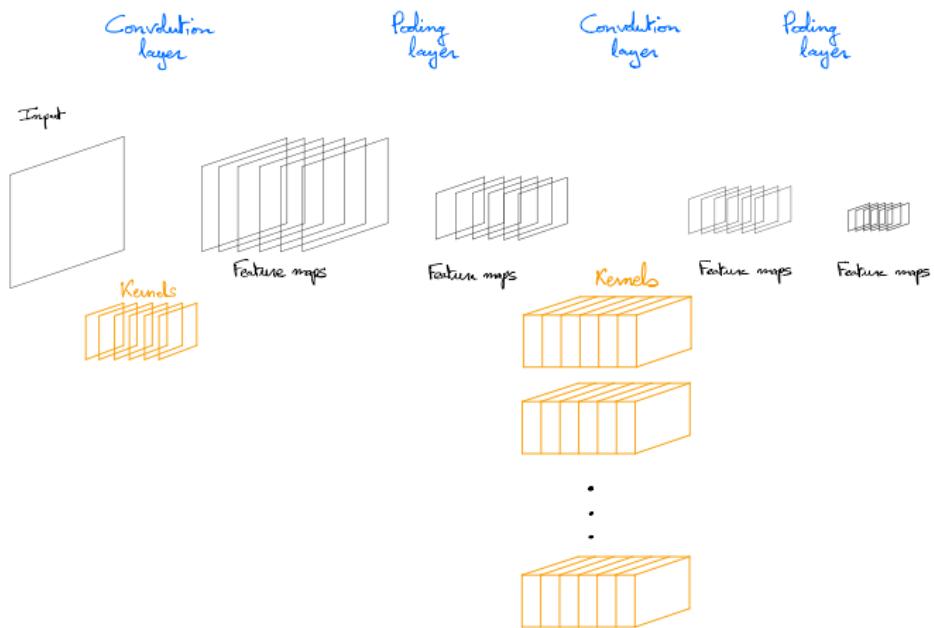
A convolutional layer operates on the feature maps output by the pooling layer. Each kernel is a volume whose depth equals the depth of the input volume.



# A possible architecture of a CNN



# A possible architecture of a CNN



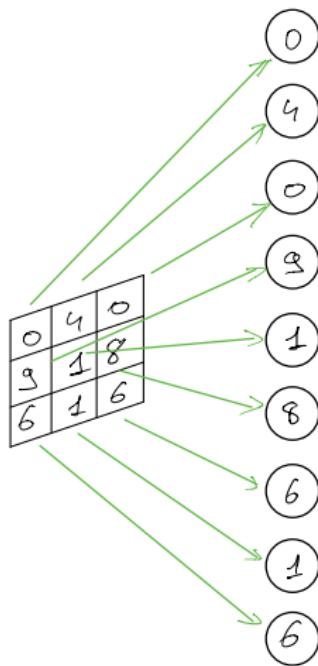
## A possible architecture of a CNN

At the end of the network, the feature maps are flattened in order to apply a classic neural networks.

0	4	0
3	1	8
6	1	6

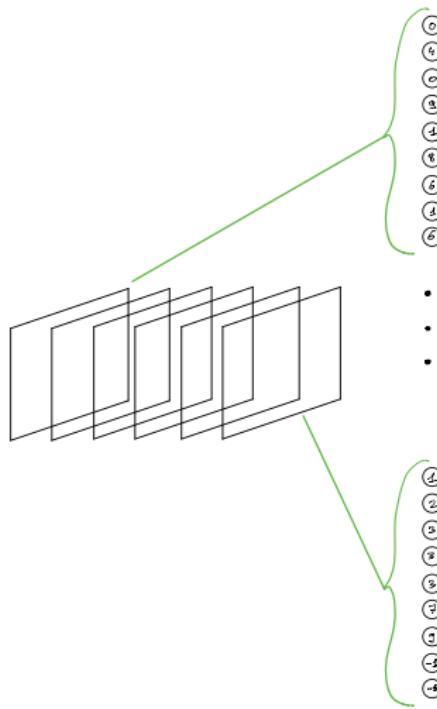
## A possible architecture of a CNN

At the end of the network, the feature maps are flattened in order to apply a classic neural networks.



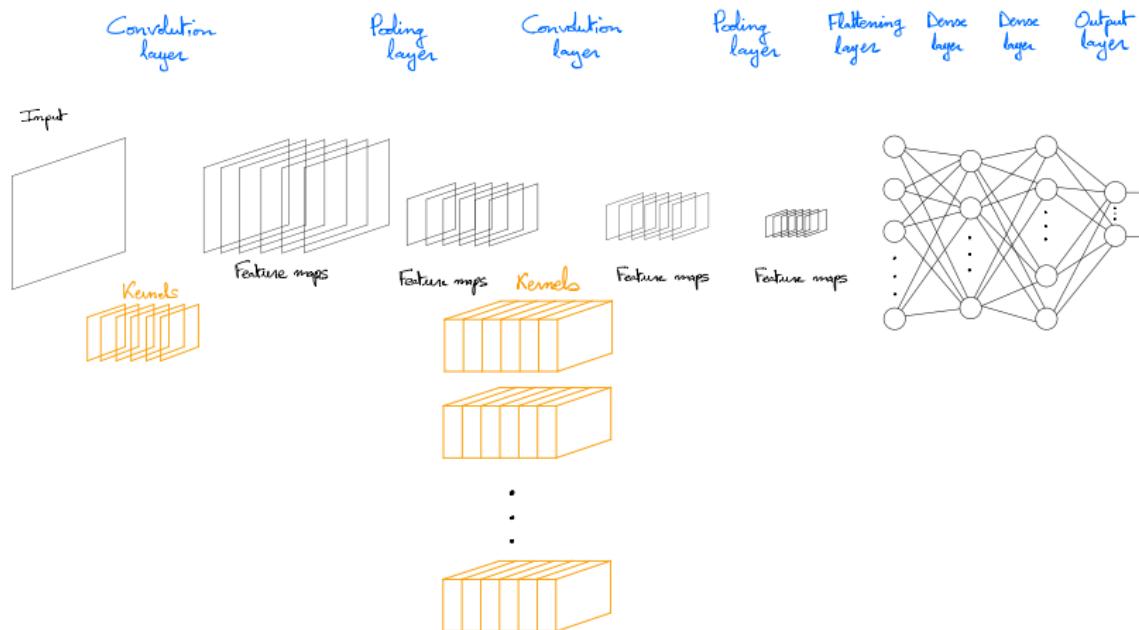
## A possible architecture of a CNN

At the end of the network, the feature maps are flattened in order to apply a classic neural networks.



# A possible architecture of a CNN

The full architecture is summarized below.



# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# Data processing

## Normalizing data

For each channel R, G, B, compute the pixels mean over all images in the whole data set. Subtract this value to each channel of each image. → you do not lose relative information between images.

## Data augmentation

- ① Sampling ["Imagenet large scale visual recognition challenge", Russakovsky et al. 2015]
- ② Translation/shifting ["Deep convolutional neural networks and data augmentation for environmental sound classification", Salamon and Bello 2017]
- ③ Horizontal reflection/mirroring ["Mirror, mirror on the wall, tell me, is the error small?", H. Yang and Patras 2015]
- ④ Rotating ["Holistically-nested edge detection", Xie and Tu 2015]
- ⑤ Various photometric transformations ["Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture", Eigen and Fergus 2015]

## Prediction

At test time, patches are extracted from the new images together with some of its reflection/translation/... A prediction is made for each of these artificial images and they are aggregated to make the final prediction.

# Adding noise - Data augmentation and regularization

- Add noise to input

[“Training with noise is equivalent to Tikhonov regularization”, Bishop 1995]

[“Adding noise to the input of a model trained with a regularized objective”, Rifai et al. 2011]

[“Explaining and harnessing adversarial examples”, Goodfellow et al. 2014]

- Add noise to weights

[“An analysis of noise in recurrent neural networks: convergence and generalization”, Jim et al. 1996]

[“Practical variational inference for neural networks”, Graves 2011]

- Add noise to output

[“Randomizing outputs to increase prediction accuracy”, Breiman 2000]

- Select the best data transformations (computationally expensive, many re-training steps).

[“Transformation pursuit for image classification”, Paulin et al. 2014]

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

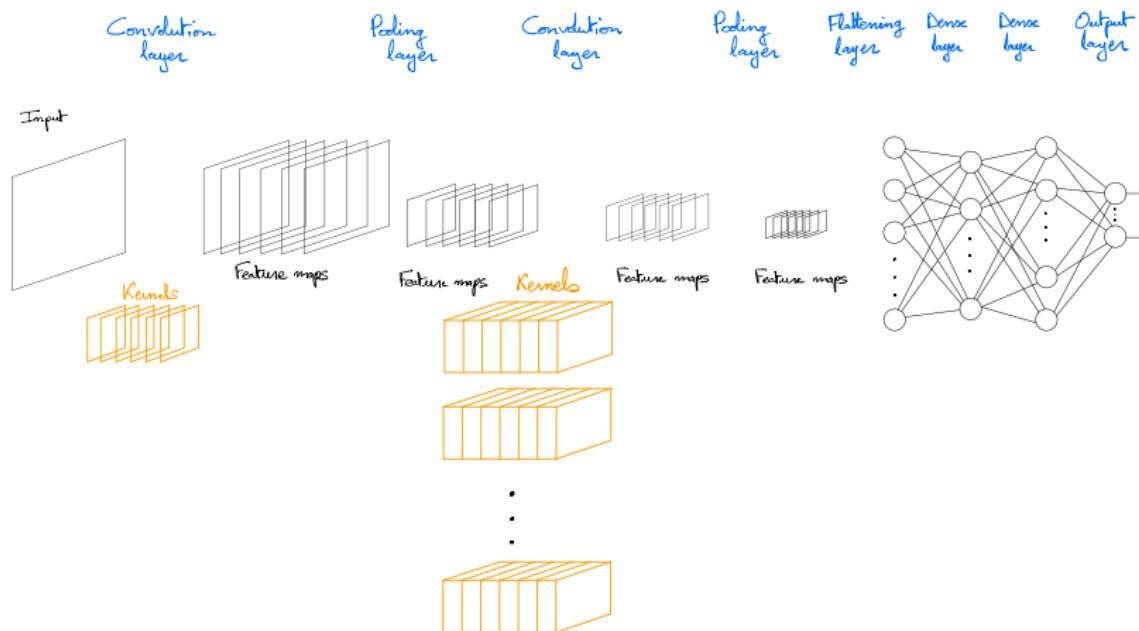
## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

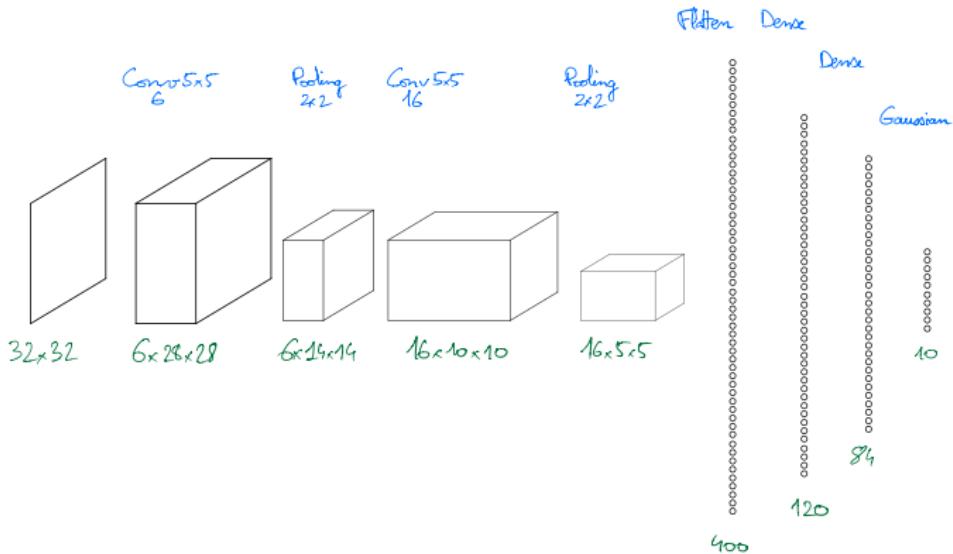
# LeNet

[“Generalization and network design strategies”, LeCun et al. 1989]

[“Gradient-based learning applied to document recognition”, LeCun et al. 1998]



# LeNet



First layer: convolutional layer C1

- Kernel size =  $5 \times 5 + \text{a bias}$
- Stride = 1 (overlapping contiguous receptive fields)
- Zero-padding = 0
- Output: 6 different feature maps, each one resulting from the convolution with a kernel  $5 \times 5$  to which the activation function  $\sigma$  is applied.

## Second layer: subsampling/pooling layer S2

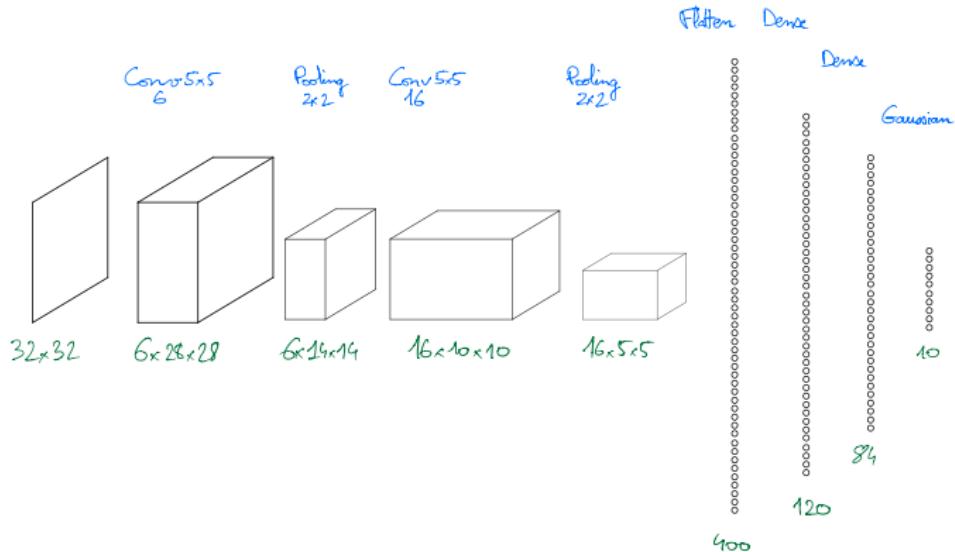
- Type of pooling: averaging.
- Kernel size =  $2 \times 2$
- Stride = 2 (non-overlapping receptive fields)
- Zero-padding = 0
- Output: one feature map per input feature map resulting from the operation  $\sigma((2 \times 2 \text{ averaging})w + b)$ .

## Third-layer: convolutional layer C3

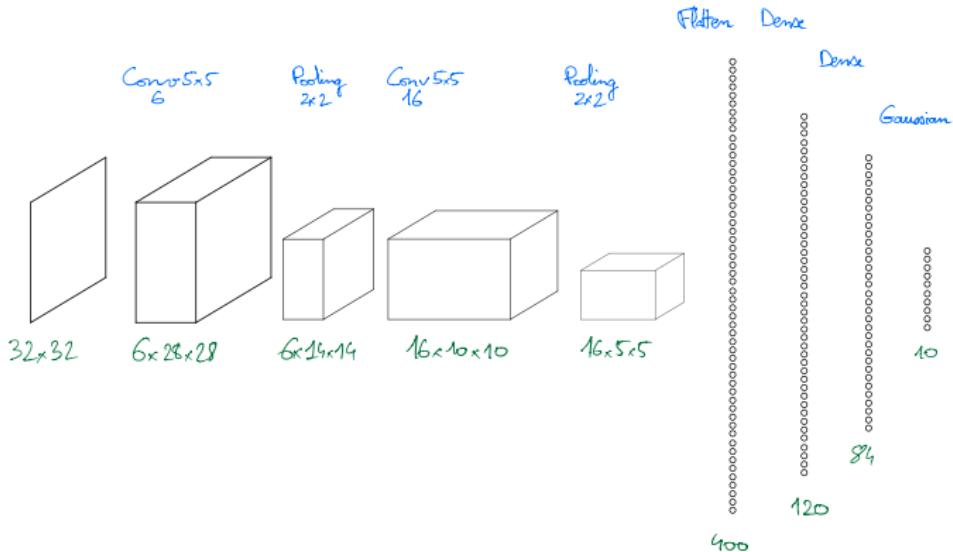
- Warning: this layer operates on several feature maps whereas layer C1 operates on the input image (depth = 1).
- Here each feature map is connected to some specific input feature maps in order to
  - ▶ Reduce the number of connections
  - ▶ Break the symmetry between the different layers of the network.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5			X	X	X	X			X	X	X	X		X	X	X

## What about the remaining layers



## What about the remaining layers



- S4: Pooling layer as before
- C5: Convolutional layer connected to all previous feature maps.
- F6: fully-connected layer with 84 units
- Output: a specific layer

**Bi-pyramidal structure:** the number of feature maps increases while the spatial resolution decreases.

## Output layer

### Radial Basis function units

The  $j$ th neuron of the output layer computes

$$\|z - w_j\|_2^2 = \sum_{i=1}^{84} (x_i - w_{j,i})^2,$$

where  $z$  is the vector of size 84 produced by layer F6 and  $w_j = (w_{j,1}, \dots, w_{j,84})$  is the weight vector of the  $j$ th neuron.

### Gaussian connections

Assuming that the vector in layer F6 are Gaussian, neuron  $j$  outputs the negative log likelihood of a Gaussian distribution with mean  $w_j$  and covariance matrix  $I$ .

In other words, each neuron outputs the square euclidean distance between its parameter vector and the input.

### Question.

How to choose  $w_j \in \{-1, 1\}^{84}$ ?

## Output layer and activation function

To choose  $w_0 \in \{-1, 1\}^{84}$ , use a stylized version of the image of 0 of size  $7 \times 12 = 84$ .  
The pixel of this image are the parameters  $w_j$  of the output neuron  $j = 0$ .

Why do not use a one-hot encodeage?

LeCun et al. 1998 states that it does not work with more than few dozens of classes since it requires output units to be off most of the time which is difficult to achieve with sigmoid functions.

### Activation function

$$\sigma(x) = A \tanh(\alpha x),$$

where  $A = 1.7159$ ,  $\alpha = 2/3$ .

→ Prevent saturation since neurons outputs belong to  $\{-1, 1\}$

- $\sigma(1) = 1$
- $\sigma(-1) = -1$ .

## Criterion to optimize

Let  $[f_\theta(x)]_j = \|z - w_j\|_2^2$  be the output of the  $j$ th neuron of the output layer, where  $z$  is the vector produced by layer F6.

Then the error for one observation  $(x, y)$  is defined as

$$E(\theta) = \sum_{j=0}^9 [f_\theta(x)]_j \mathbb{1}_{y=j} + \log \left( e^{-C} + \sum_{j=0}^9 e^{-[f_\theta(x)]_j} \right),$$

where  $C > 0$  is a constant.

The second term acts as a regularization since it forces the parameters of the neurons  $j \neq y$  to be far from the input vector of layer F6.

## Criterion to optimize

Let  $[f_\theta(x)]_j = \|z - w_j\|_2^2$  be the output of the  $j$ th neuron of the output layer, where  $z$  is the vector produced by layer F6.

Then the error for one observation  $(x, y)$  is defined as

$$E(\theta) = \sum_{j=0}^9 [f_\theta(x)]_j \mathbb{1}_{y=j} + \log \left( e^{-C} + \sum_{j=0}^9 e^{-[f_\theta(x)]_j} \right),$$

where  $C > 0$  is a constant.

The second term acts as a regularization since it forces the parameters of the neurons  $j \neq y$  to be far from the input vector of layer F6.

This is equivalent to

$$E(\theta) = -\log \left( \frac{e^{-[f_\theta(x)]_y}}{e^{-C} + \sum_{k=0}^9 e^{-[f_\theta(x)]_k}} \right),$$

which is very close to the **negative log likelihood of a softmax output layer**.

## Optimization procedure

Related to stochastic gradient descent:

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \frac{\eta}{\mu + h_{jj}} \frac{\partial E_i}{\partial \theta_j},$$

where  $E_i$  is the loss of a single observation,  $\eta$  is the initial learning rate,  $\mu$  a hand-picked constant and  $h_{jj}$  is the  $j$ th diagonal element of the Hessian matrix associated to  $E_i$ .

The expression of  $h_{jj}$  is quite complicated since  $\theta_j$  appears in different connections:

$$h_{jj} = \sum_{(i,m) \in V_j} \sum_{(k,l) \in V_j} \frac{\partial^2 E_i}{\partial u_{im} \partial u_{kl}},$$

where  $u_{im}$  is the connection between units  $i$  and  $m$ , and  $V_j$  is the set of pairs  $(i, m)$  such that the connection between  $i$  and  $m$  involves the weight  $\theta_j$ .

An approximation of each diagonal terms  $h_{jj}$  is performed at the beginning of each epoch, using the first 500 observations (whole data set being composed of 60000 observations).

## Parameters

Weight initialization: uniform distribution  $U([-2.4/F_i, 2.4/F_i])$ , where  $F_i$  is the number of inputs (fan-in) of the unit which the connection belongs to.

→ Keep the weighted sum in the same range for each unit.

## Gradient descent

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \frac{\eta}{\mu + h_{jj}} \frac{\partial E_i}{\partial \theta_j},$$

with  $\mu = 0.02$ .

Optimization lasts 20 epochs:

- $\eta = 0.0005$  for the first two epochs,
- $\eta = 0.0002$  for the next three epochs,
- $\eta = 0.0001$  for the next three epochs,
- $\eta = 0.00005$  for the next four epochs,
- $\eta = 0.00001$  for the remaining epochs,

# Results



The 82 patterns misclassified by LeNet5. Below each image is displayed the correct answer (left) and the prediction (right). These errors are mostly caused by genuinely ambiguous patterns, or by digits written in a style that are under represented in the training set.

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- **AlexNet (2012)**
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

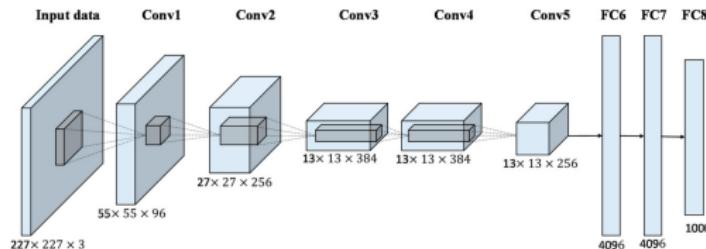
- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# AlexNet

[“Imagenet classification with deep convolutional neural networks”, Krizhevsky et al. 2012]

Ingredients:

- Activation function (ReLU)
- Local Response Normalization (LRN)
- Overlapping pooling ( $3 \times 3$  window with a stride  $S = 2$  which reduces overfitting)
- Dropout
- Data augmentation



## ReLU activation function

According to Krizhevsky et al. 2012, Convolutional neural networks with ReLU activation functions can be trained several times faster than the same networks using tanh function.

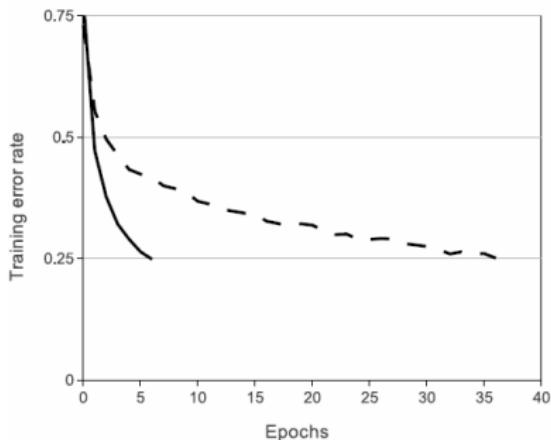


Figure: A four-layer convolutional neural network with ReLU (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh (dashed line). The learning rates for each network were chosen independently to make training as fast as possible.

## Local Response Normalization / Brightness normalization

Let  $a_{x,y}^i$  the activity of a neuron resulting of kernel  $i$  applied to the position  $(x, y)$  followed by a ReLU function and  $b_{x,y}^i$  the corresponding renormalized activity which is given by

$$b_{x,y}^i = a_{x,y}^i \left( C + \alpha \sum_{j=\max(0, i-q/2)}^{\min(Q-1, i+q/2)} (a_{x,y}^j)^2 \right)^{-\beta},$$

where the sum is taken over  $q$  adjacent feature maps at the same spatial position, and  $Q$  is the total number of feature maps in this layer.

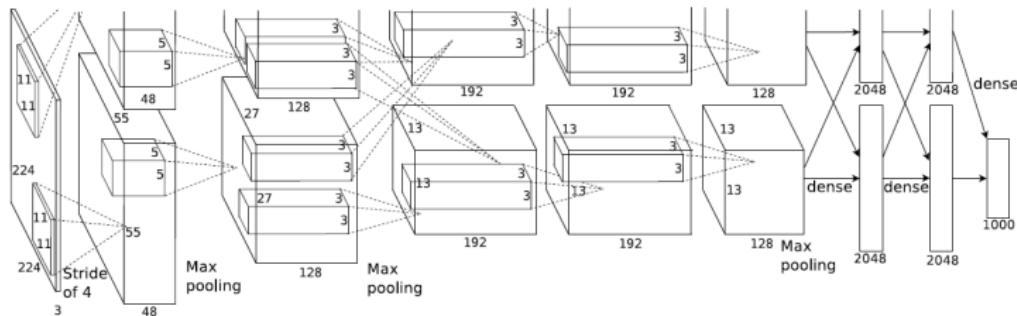
Constants (determined with validation set):  $C = 2, q = 5, \alpha = 10^{-4}, \beta = 0.75$ .

Note that the ordering of feature maps is arbitrary and determined before training. This renormalization creates a **competition between the different feature maps**.

[“What is the best multi-stage architecture for object recognition?”, Jarrett et al. 2009]

They propose a similar normalization procedure where the mean activity is subtracted (local contrast normalization).

## Overall architecture



Key-point: architecture is split across two GPU, which, most of the time, do not communicate with each other.

- Connectivity of each convolutional layer
- ReLu are applied right after all convolutional layers and fully connected layers
- Local Response Normalization is applied after ReLU in the first and second convolutional layer
- Max-pooling is applied after the first, second and fifth convolutional layers.

# Optimization

Initialization:

- Weights:  $\mathcal{N}(0, 0.0001)$
- Biases of second, fourth and fifth convolutional layers and biases of fully connected layers set to 1 (seems to accelerate the early stages of learning, prevent dying ReLU phenomenon). Other biases are set to 0.

Stochastic gradient descent with momentum

$$\begin{aligned} v^{(k+1)} &= 0.9v^{(k)} - 0.0005\eta\theta^{(k)} - \frac{\eta}{B} \sum_{i \in \mathcal{B}} \nabla \ell_i(\theta^{(k)}) \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)}, \end{aligned}$$

with batch size  $|\mathcal{B}| = B = 128$ .

The second term in the first equation corresponds to the  $L_2$  regularization of the loss with a constant  $\lambda = 0.0005$  (weight decay of 0.0005).

Learning rate is the same for all layers with the following heuristic:

- Initialization:  $\eta = 0.01$
- Divide  $\eta$  by 10 when the validation error stop improving (done three times here).
- 90 epochs on 1.2 million images: 6 days.

## Numerical results

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
<i>SIFT + FVs</i> [7]	—	—	26.2%
1CNN	40.7%	18.2%	—
5CNNs	38.1%	16.4%	16.4%
1CNN*	39.0%	16.6%	—
7CNNs*	36.7%	15.4%	15.3%

- First line is the second runner-up.
- Second and third lines are results output by the averaging over 1 or 5 CNN described before.
- Last two lines correspond to networks with an extra convolutional layer after the last pooling layer which has been trained on Image Net Fall 2011 then “fine-tuned” on the ImageNet 2012 data base.

AlexNet has a very similar architecture to LeNet, but is deeper, bigger, and features Convolutional Layers stacked on top of each other: previously, pooling layers followed immediately each convolutional layer.

# Results

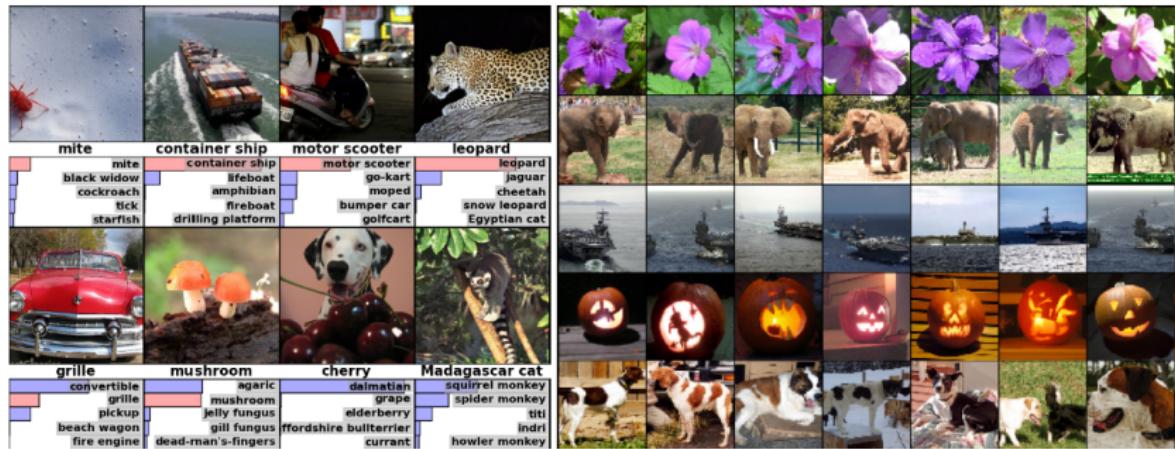


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010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

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

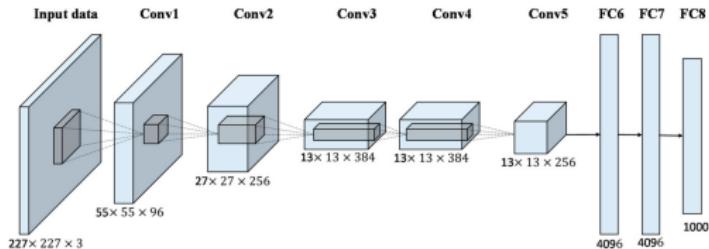
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)**
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# ZFNet: Improve upon AlexNet

[“Visualizing and understanding convolutional networks”, Zeiler and Fergus 2014]



Aim at finding out what the different feature maps are searching for in order to obtain a better tuning of network architecture.

In ZFNet, feature maps are not divided across two different GPU. Thus connections between layers are less sparse than for AlexNet.

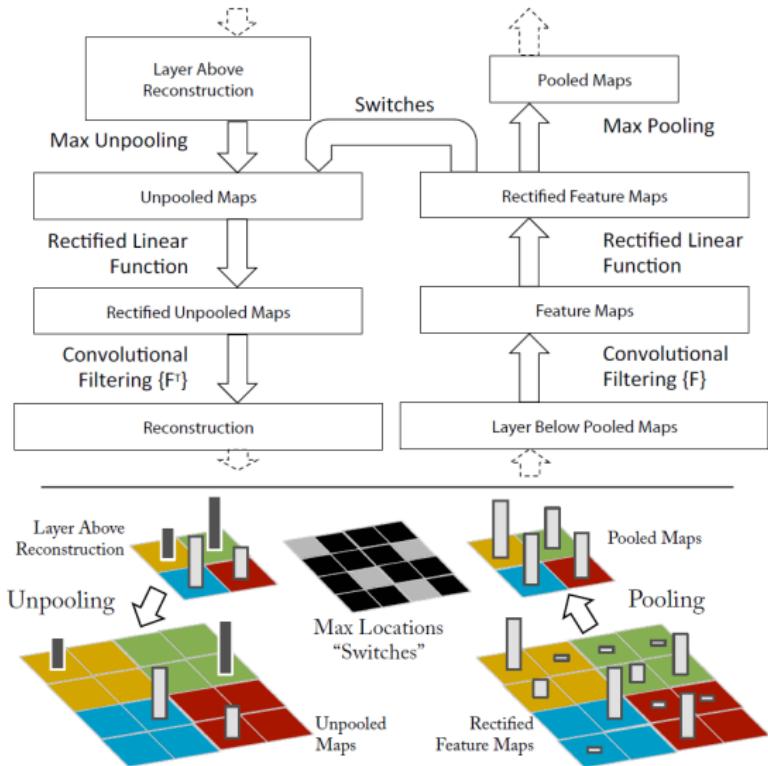
# Deconvnet

Find the pixels that maximize the activation of a given feature map.

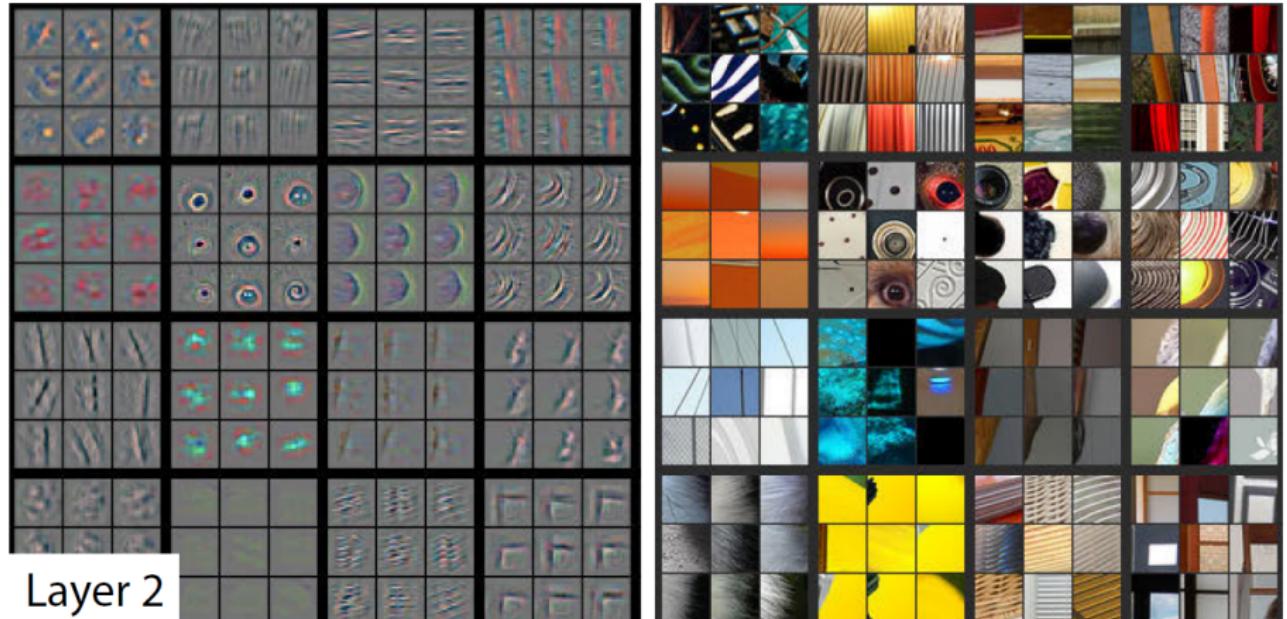
How? Invert the network.

Precisely:

- Choose a layer
- Choose a feature map
- Run the network on a validation set
- Choose the image maximizing the activation of this feature map
- "Backpropagate" this activation to obtain a stylized image in the pixel space

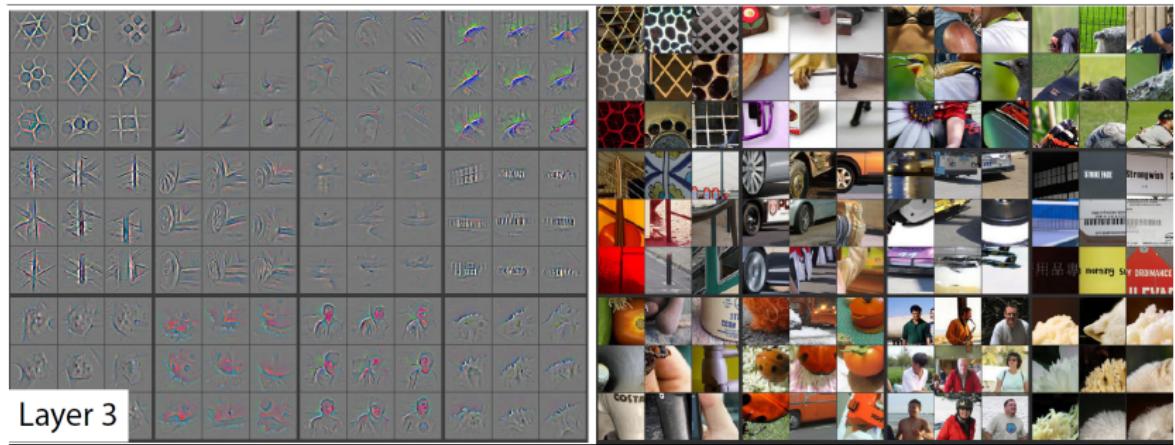


# Results

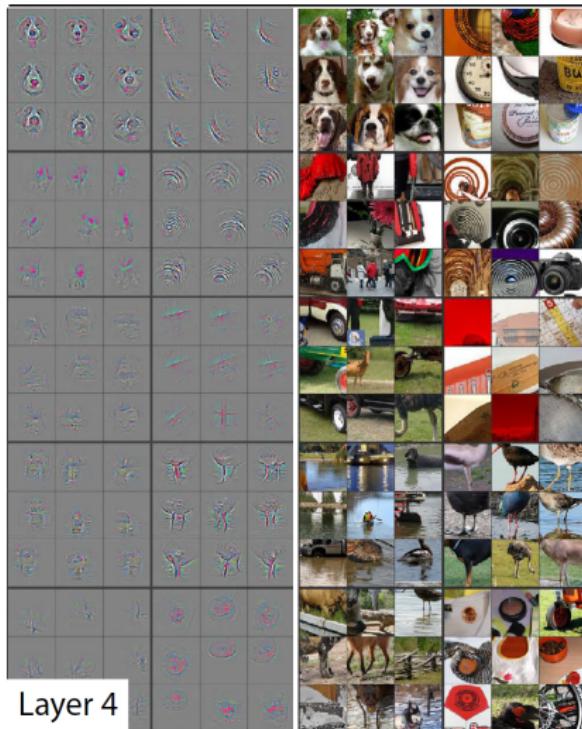


Top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using the previous deconvolutional network approach.

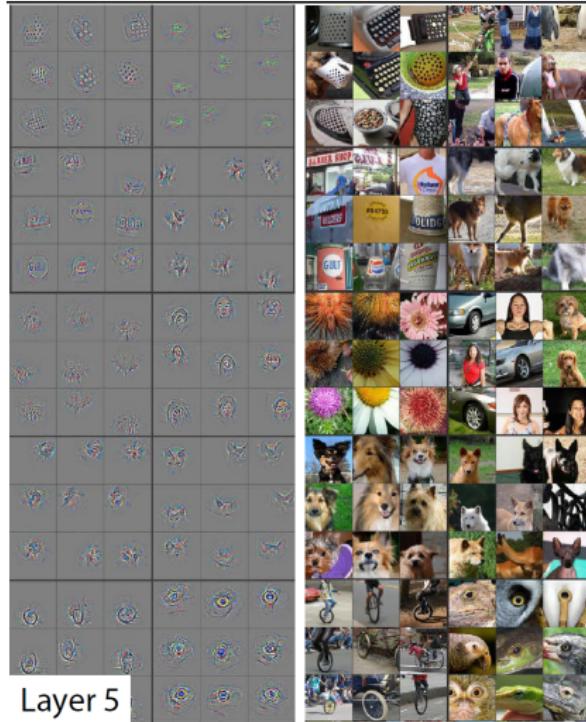
# Results



# Results



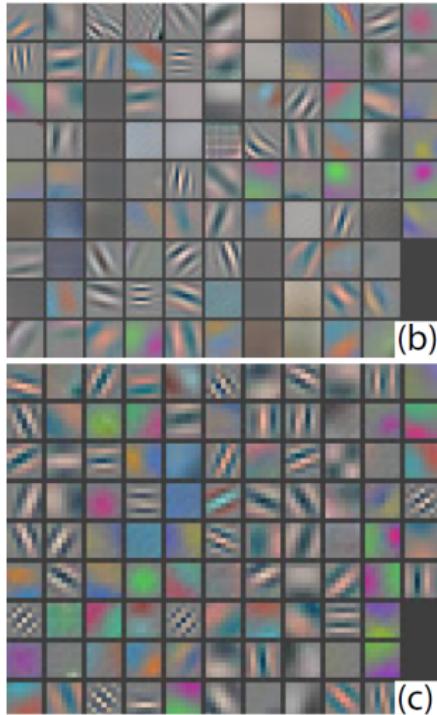
# Results



## Remarks

- strong grouping within each feature map,
- greater invariance at higher layers
- exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1).

## Visualization of previous modifications



(b): 1st layer features from Krizhevsky et al. 2012.

(c): 1st layer features of ZFNet.

## Visualization of previous modifications



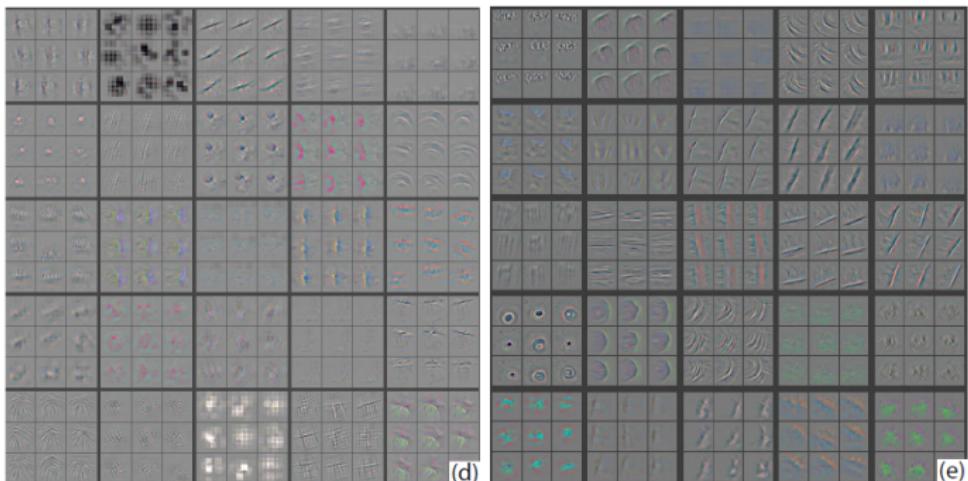
(b): 1st layer features from Krizhevsky et al. 2012.

(c): 1st layer features of ZFNet.

Differences: smaller stride (2 vs 4) and filter size (7x7 vs 11x11)

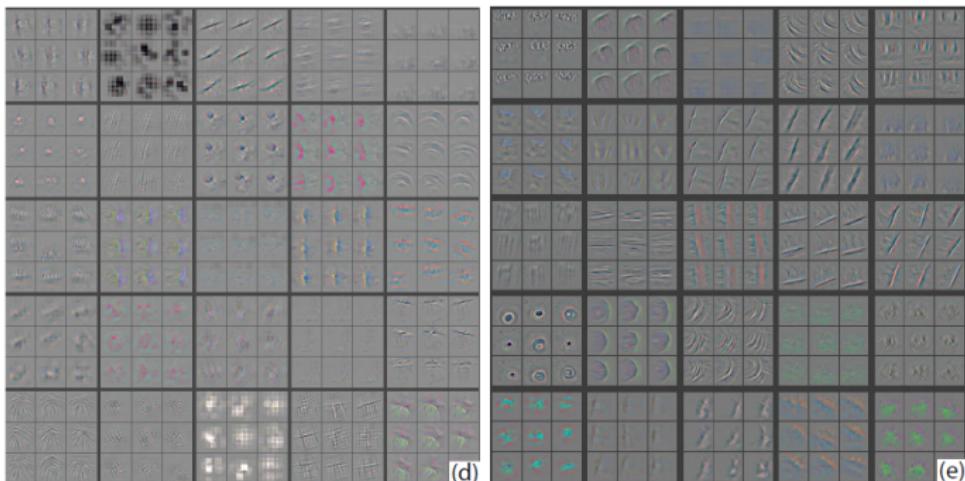
Results in more distinctive features and fewer dead features.

## Visualization of previous modifications



(d): Visualizations of 2nd layer features from Krizhevsky et al. 2012; (e): Visualizations of the 2nd layer features of ZFNet.

## Visualization of previous modifications



(d): Visualizations of 2nd layer features from Krizhevsky et al. 2012; (e): Visualizations of the 2nd layer features of ZFNet.

Feature maps in (e) are **cleaner**, with **no aliasing artefacts** that are visible in (d).

## Conclusion regarding AlexNet

- First layer filters are a mix of high and low frequency information, with little coverage of middle frequencies.
  - Reduced the first layer filter size from  $11 \times 11$  to  $7 \times 7$ .
- Aliasing artifacts are present in second layer because of the large stride of 4 used in the first convolutional layer.
  - change the stride from 4 to 2.

With these modifications:

- Winner of the ILSVRC 2013
- Improvement on AlexNet by
  - ▶ expanding the size of the middle convolutional layers
  - ▶ making the stride and filter size on the first layer smaller.

## ZF Net final structure

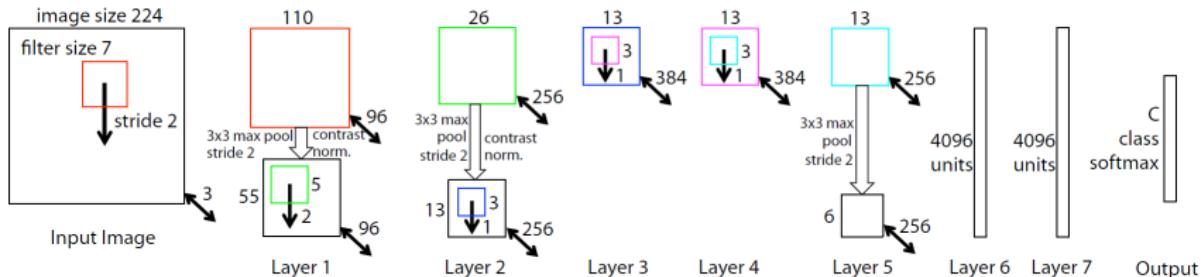
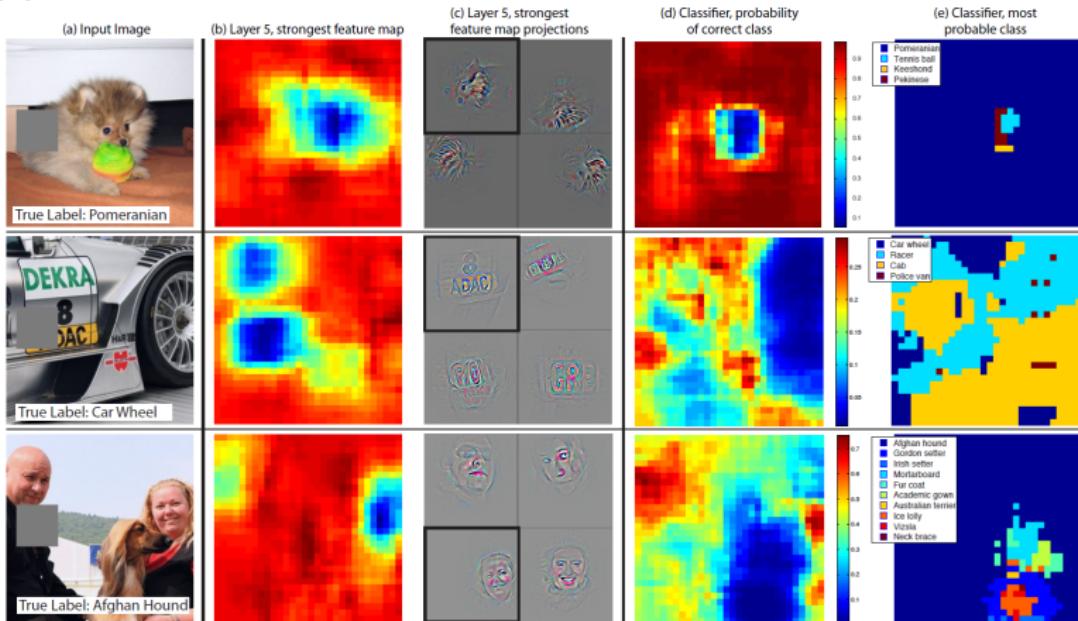


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ( $6 \cdot 6 \cdot 256 = 9216$  dimensions). The final layer is a  $C$ -way softmax function,  $C$  being the number of classes. All filters and feature maps are square in shape.

## Results in classification

Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	—	—	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	—
(Krizhevsky et al., 2012), 5 convnets	38.1	16.4	16.4
(Krizhevsky et al., 2012)*, 1 convnets	39.0	16.6	—
(Krizhevsky et al., 2012)*, 7 convnets	36.7	15.4	15.3
Our replication of			
(Krizhevsky et al., 2012), 1 convnet	40.5	18.1	—
1 convnet as per Fig. 3	38.4	16.5	—
5 convnets as per Fig. 3 - (a)	36.7	15.3	15.3
1 convnet as per Fig. 3 but with			
layers 3, 4, 5: 512,1024,512 maps - (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	<b>36.0</b>	<b>14.7</b>	<b>14.8</b>

# Occlusion



Three test examples where we systematically cover up different portions of the scene with a gray square (1st column) and see how the top (layer 5) feature maps ((b) & (c)) and classifier output ((d) & (e)) changes.

(b): for each position of the gray scale, we record the total activation in one layer 5 feature map (the one with the strongest response in the unoccluded image).

(c): a visualization of this feature map projected down into the input image (black square), along with visualizations of this map from other images. The first row example shows the strongest feature to be the dog's face. When this is covered-up the activity in the feature map decreases (blue area in (b)).

(d): a map of correct class probability, as a function of the position of the gray square. E.g. when the dog's face is obscured, the probability for pomeranian drops significantly.

(e): the most probable label as a function of occluder position.

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

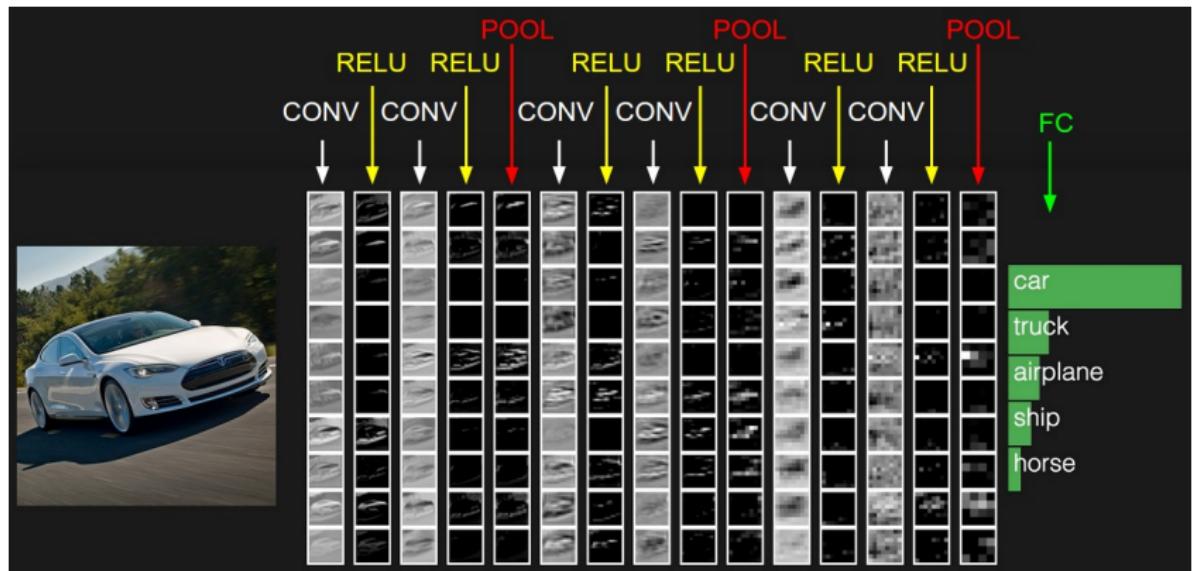
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)**
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# Tiny VGGnet

[“Very deep convolutional networks for large-scale image recognition”, Simonyan and Zisserman 2014b]



## Network features

### Convolutional layers:

- Small receptive field:  $3 \times 3$  (smallest ones capable of capturing the notion of top/down, left/right!)
- Stride of 1
- Spatial resolution is preserved after convolution

### Max-pooling layers:

- $2 \times 2$  kernel
- Stride of 2

All hidden layers use ReLU activation functions.

Local Response Normalization layers do not improve performance.

## Insightful remark...

If you stack 3 convolutional layers with receptive fields  $3 \times 3$ , you obtain a convolutional layer with receptive fields  $7 \times 7$ . What is the interest?

- ① Stack of 3 convolutional layers of size  $3 \times 3$ : complexity of  $3 \times 3 \times 3 = 27$ .
- ② One standard convolutional layer of size  $7 \times 7$ : complexity of 49.

In the first case, we cannot obtain every possible layer: the resulting object is a decomposition of three consecutive convolutional layers. There are less possibilities hence less parameters.

# VGGNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv(receptive field size)-(number of channels)”. The ReLU activation function is not shown for brevity.

## Parameters

### Initialization:

- Network  $A$ :  $\mathcal{N}(0, 0.01)$  for weights and 0 for biases.
- For other networks: first four conv layers and last three fully connected layers were initialized using network  $A$  and the remaining layers were initialized randomly.

### Stochastic gradient descent with momentum

$$\begin{aligned}v^{(k+1)} &= 0.9v^{(k)} - 0.0005\eta\theta^{(k)} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla L_i(\theta^{(k)}) \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)},\end{aligned}$$

with batch size  $B = 128$ .

Learning rate is the same for all layers with the following heuristic:

- Initialization:  $\eta = 0.01$
- Divide  $\eta$  by 10 when the validation error stop improving (done three times here).
- 74 epochs.
- $L_2$  penalty with constant  $5.10^{-4}$
- Dropout regularization for the first two fully connected layers (probability  $p = 0.5$ )

## Results

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	23.7	6.8	6.8
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	—	—	7.9
GoogLeNet (Szegedy et al., 2014)(7 nets)	—	—	6.7
MSRA (He et al, 2014)(11 nets)	—	—	8.1
MSRA (He et al., 2014)(1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	—	—	11.7
Clarifai (Russakovsky et al., 2014)(1 net)	—	—	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013)(1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al, 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al, 2014) (1 net)	35.7	14.2	—
Krizhevsky et al. (Krizhevsky et al., 2012)( 5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	—

A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M).

Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- **GoogLeNet (2014)**
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# GoogLeNet

[“Going deeper with convolutions”, Szegedy, W. Liu, et al. 2015]

## Aim.

Increasing the depth and width of state-of-the-art convolutional neural networks while keeping the number of parameters small:

- Can approximate more complex functions
- while being robust to overfitting and computationally appealing.

## How.

Specifically, use of  $1 \times 1$  convolution layers to reduce the number of parameters + apply filters of different sizes  $3 \times 3$ ,  $5 \times 5$  or  $3 \times 3$  max pooling (on each feature maps).

## Details.

- All convolution layers use ReLU activation functions.
- Same spatial resolution for each feature map.

# GoogLeNet - Inception module

Same spatial resolution for each feature map.

Use of  $1 \times 1$  convolution layers to reduce the number of parameters then apply filters of different sizes  $3 \times 3$ ,  $5 \times 5$  or  $3 \times 3$  max pooling (on each feature maps).

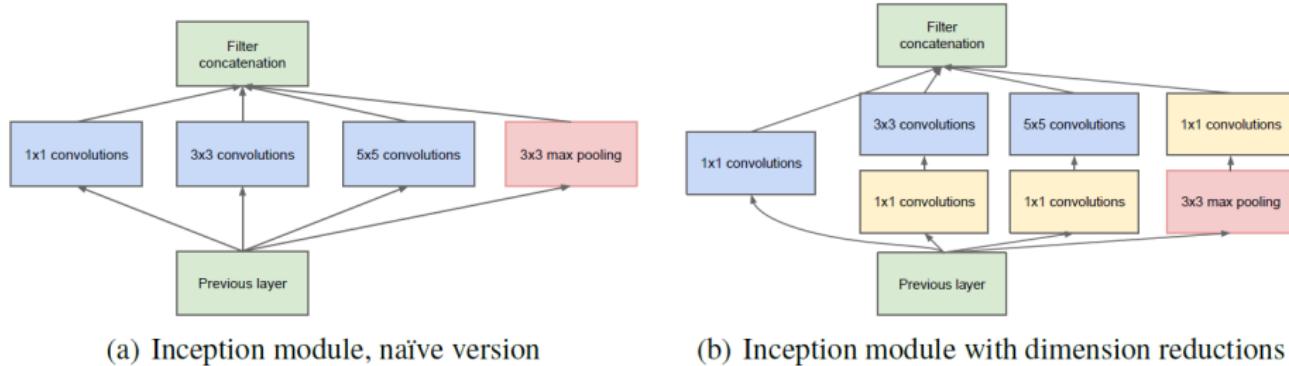


Figure 2: Inception module

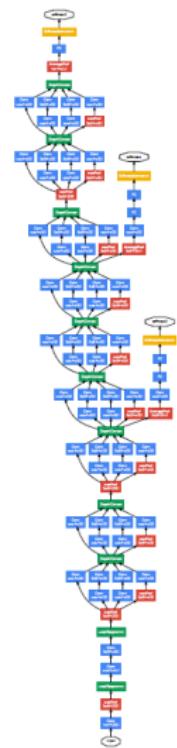
# GoogLeNet - Inception module

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	$7 \times 7 / 2$	$112 \times 112 \times 64$	1							2.7K	34M
max pool	$3 \times 3 / 2$	$56 \times 56 \times 64$	0								
convolution	$3 \times 3 / 1$	$56 \times 56 \times 192$	2		64	192				112K	360M
max pool	$3 \times 3 / 2$	$28 \times 28 \times 192$	0								
inception (3a)		$28 \times 28 \times 256$	2	64	96	128	16	32	32	159K	128M
inception (3b)		$28 \times 28 \times 480$	2	128	128	192	32	96	64	380K	304M
max pool	$3 \times 3 / 2$	$14 \times 14 \times 480$	0								
inception (4a)		$14 \times 14 \times 512$	2	192	96	208	16	48	64	364K	73M
inception (4b)		$14 \times 14 \times 512$	2	160	112	224	24	64	64	437K	88M
inception (4c)		$14 \times 14 \times 512$	2	128	128	256	24	64	64	463K	100M
inception (4d)		$14 \times 14 \times 528$	2	112	144	288	32	64	64	580K	119M
inception (4e)		$14 \times 14 \times 832$	2	256	160	320	32	128	128	840K	170M
max pool	$3 \times 3 / 2$	$7 \times 7 \times 832$	0								
inception (5a)		$7 \times 7 \times 832$	2	256	160	320	32	128	128	1072K	54M
inception (5b)		$7 \times 7 \times 1024$	2	384	192	384	48	128	128	1388K	71M
avg pool	$7 \times 7 / 1$	$1 \times 1 \times 1024$	0								
dropout (40%)		$1 \times 1 \times 1024$	0								
linear		$1 \times 1 \times 1000$	1							1000K	1M
softmax		$1 \times 1 \times 1000$	0								

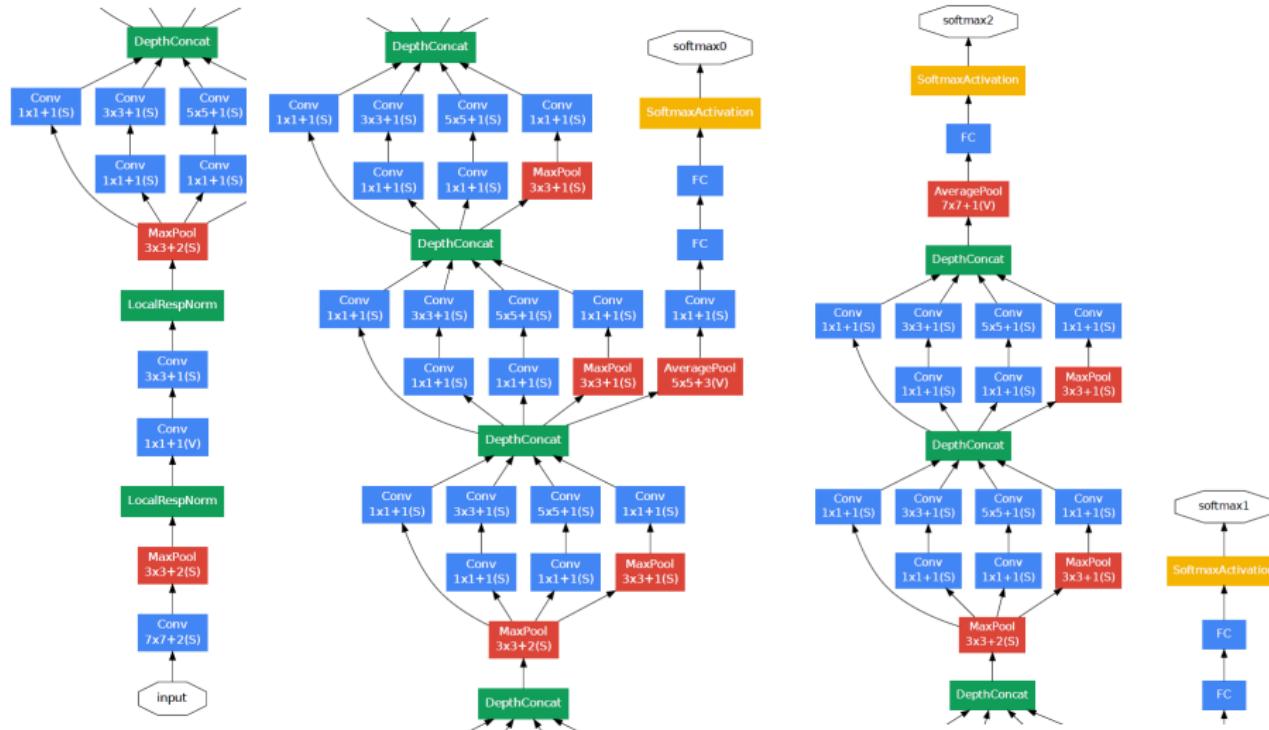
Table 1: GoogLeNet incarnation of the Inception architecture

“3x3 reduce” and “5x5 reduce” stands for the number of 1x1 filters in the reduction layer used before the 3x3 and 5x5 convolutions. One can see the number of 1x1 filters in the projection layer after the built-in max-pooling in the pool proj column.

# Structure of GoogLeNet



## Structure of GoogLeNet



## Deep network - A concern

In order to backpropagate gradient, the authors add some auxiliary classifiers connected to intermediate layers.

During training the loss of auxiliary classifiers is weighted by 0.3 and added to the total loss of the network. Auxiliary networks are removed at inference time.

Auxiliary network put after (4a) and (4d):

- Average pooling layer  $5 \times 5$ , stride of 3
- A  $1 \times 1$  convolution with 128 filters, with ReLU.
- A fully connected layer with 1024 neurons and ReLU
- A dropout layer with a dropout ratio of 70%.
- A linear layer with softmax loss, predicting the same 1000 classes as the main classifier.

## Parameters

### Initialization:

- Weights are drawn from  $\mathcal{N}(0, 1)$  and biases are set to 0.

[“Deep learning via Hessian-free optimization.”, Martens 2010]

### Stochastic gradient descent with momentum

$$\begin{aligned} v^{(k+1)} &= \mu v^{(k)} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla \ell_i(\theta^{(k)} + \mu v^{(k)}) \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)}, \end{aligned}$$

with batch size  $B = 200$ , where

$$\mu^{(k)} = \min\left(1 - 2^{-1-\log_2(\lfloor k/250 \rfloor + 1)}, \mu_{max}\right),$$

where  $\mu_{max} \in \{0, 0.9, 0.99, 0.995, 0.999\}$ .

Learning rate is the same for all layers with the following heuristic:

- Initialization:  $\eta = 0.01$
- Multiply  $\eta$  by 0.96 every 8 epochs.
- Training lasts 125 epochs.

## Results

- Polyak averaging is used to create the final model at inference time.
- 7 different versions of GoogleNet were trained and aggregated to make predictions.

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Main contribution: development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- **ResNet (2016)**
- DenseNet (2017)
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# ResNet (2016)

[“Deep residual learning for image recognition”, He et al. 2016]

**Statement:** Optimization can be hard for some deep networks.

**Solution:** Ease optimization by adding simple paths in the network

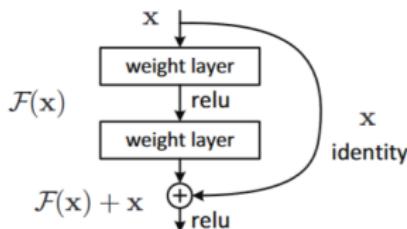


Figure 2. Residual learning: a building block.

→ No extra parameters, no additional computational complexity

## Literature on shortcut connections

Early practice for training multi-layer perceptrons was to add a linear layer between the inputs and the outputs

[*Pattern recognition and neural networks*, Ripley 2007]

Few intermediate classifiers can also be added in intermediary levels in order to ease the optimization:

- ["Going deeper with convolutions", Szegedy, W. Liu, et al. 2015]
- ["Deeply-supervised nets", Lee et al. 2015]

Highway networks have shortcut connections with gating functions. Here, gates are data dependent and have parameters.

- ["Highway networks", Rupesh Kumar Srivastava et al. 2015]
- ["Training very deep networks", Rupesh K Srivastava et al. 2015]

## General Idea

Inspired from VGG nets:

- For the same output feature map size, the layers have the same numbers of filters
- If the feature map size is halved, then the number of filters is doubled to preserve the time complexity per layer

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}_i) + \mathbf{x},$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are respectively the input and the output of a (stack of) layer(s),  $\mathbf{W}_i$  are the weights of this/these layer(s) and  $f(\mathbf{x}, \mathbf{W}_i)$  the output of this/these layer(s).

If dimensions do not match between  $\mathbf{x}$  and  $\mathbf{y}$ , there are two solutions:

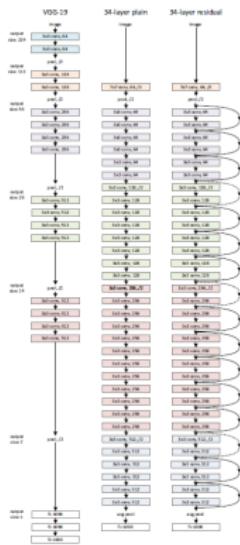
- identity mapping is coupled with extra zero entries padded for increasing dimensions
- Projection shortcut is used to match dimensions via  $1 \times 1$  convolution filters

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}_i) + W_s \mathbf{x},$$

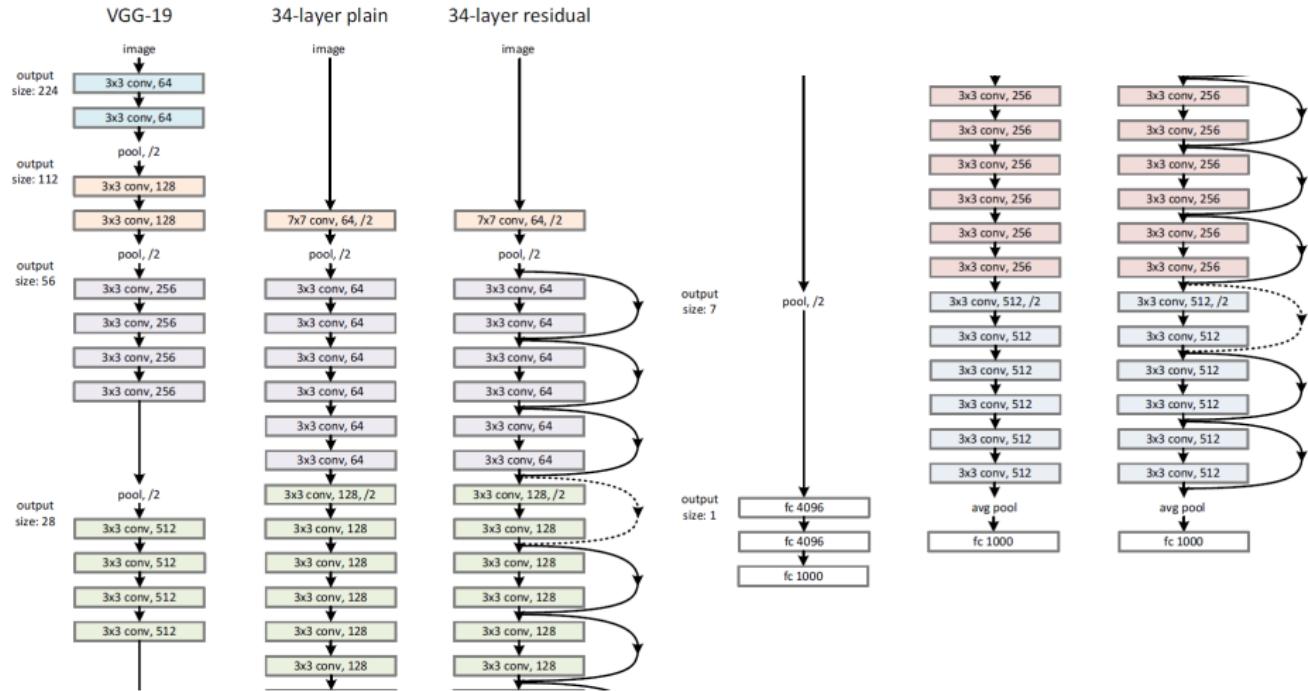
where  $W_s$  is a projection.

Besides, “when the shortcuts go across feature maps of two different sizes, they are performed with a stride of 2”.

# Structure of ResNet



# Structure of ResNet



## Parameters

Initialization, as in He et al. 2015: weights are drawn from  $\mathcal{N}(0, 2/n_L)$  ( $n_L$  is the number of neurons in the previous layer); biases are set to 0.

### Stochastic gradient descent with momentum

$$\begin{aligned}v^{(k+1)} &= 0.9v^{(k)} - 0.0001\eta\theta^{(k)} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla L_i(\theta^{(k)}) \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)},\end{aligned}$$

with batch size  $B = 256$ .

Learning rate is the same for all layers with the following heuristic:

- Initialization:  $\eta = 0.1$
- Divide  $\eta$  by 10 when the validation error stop improving (done three times here).
- 120 epochs.

Miscellaneous:

- Batch normalization after each convolution and before activation
- No dropout

## Results

method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
<b>ResNet (ILSVRC'15)</b>	<b>3.57</b>

- Winner of ILSVRC 2015
- Special skip connections and heavy use of batch normalization
- No fully connected layers at the end of the network.

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- **DenseNet (2017)**
- Many other CNN

## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# DenseNet

[“Densely Connected Convolutional Networks.”, G. Huang et al. 2017]

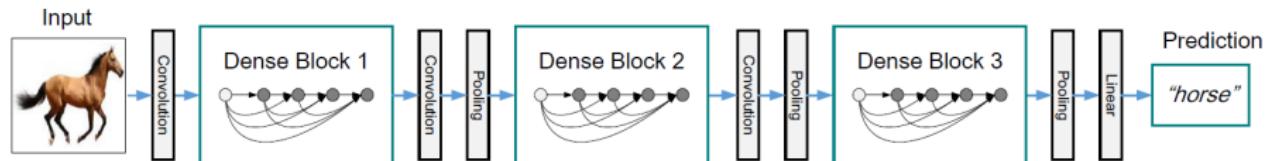


Figure: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling

# DenseNet

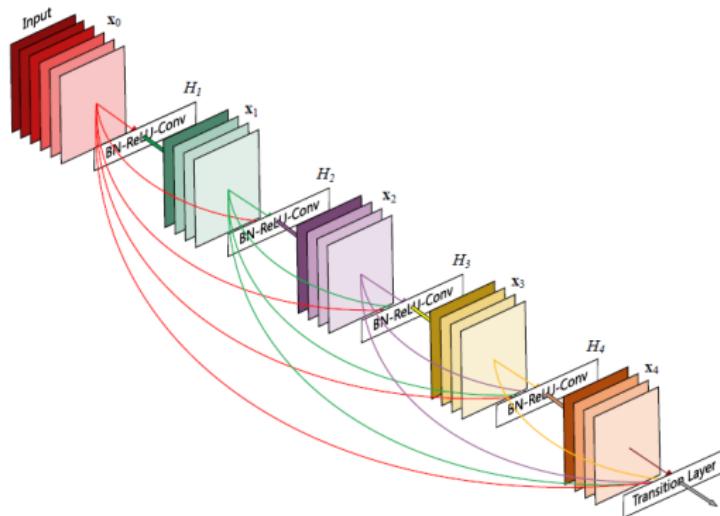


Figure: A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

## Ingredients

Let  $\mathbf{x}_\ell$  be the input of the  $\ell$ th layer. Usually,

$$\mathbf{x}_\ell = f_\ell(\mathbf{x}_{\ell-1}).$$

**Dense Block.** Inside a dense block,

$$\mathbf{x}_\ell = f_\ell(\mathbf{x}_0, \dots, \mathbf{x}_{\ell-1}).$$

The functions  $f_\ell$  are composed of three consecutive operations:

- ① First, a batch normalization
- ② Then, activation function ReLU
- ③ Finally,  $3 \times 3$  convolutional layer (feature map sizes are kept fixed)

**Transition layers.**

- ① Batch normalization
- ②  $1 \times 1$  convolution
- ③  $2 \times 2$  average pooling

# Ingredients

## Growth rate $k$

If each function  $f_\ell$  produces  $k$  feature maps, the inputs of the  $\ell$ th layer has  $k_0 + k(\ell - 1)$  channels. Narrow layers (typically  $k = 12$ ) give good results.  
→ Indeed, each layer has access to each previous layer and thus to the “collective knowledge” of the network.

## Bottleneck layer - DenseNet-B

A way to improve computational efficiency is to introduce  $1 \times 1$  convolutional layers: inside dense block, for each layer

$$\text{BN} - \text{ReLU} - \text{Conv } (1 \times 1) - \text{BN} - \text{ReLU} - \text{Conv } (3 \times 3)$$

$\text{Conv } 1 \times 1$  are set to produce  $4k$  feature maps.

## Compression layer - DenseNet-C

Throw away a fraction  $\theta \in [0, 1]$  (typically  $\theta = 0.5$ ) of feature maps at transition layers.

# Architecture

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$		$7 \times 7$ conv, stride 2		
Pooling	$56 \times 56$		$3 \times 3$ max pool, stride 2		
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$			$1 \times 1$ conv	
	$28 \times 28$			$2 \times 2$ average pool, stride 2	
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$			$1 \times 1$ conv	
	$14 \times 14$			$2 \times 2$ average pool, stride 2	
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$			$1 \times 1$ conv	
	$7 \times 7$			$2 \times 2$ average pool, stride 2	
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$			$7 \times 7$ global average pool	
				1000D fully-connected, softmax	

## Parameters

Initialization, as in He et al. 2015: weights are drawn from  $\mathcal{N}(0, 2/n_L)$  ( $n_L$  is the number of neurons in the previous layer); biases are set to 0.

### Stochastic gradient descent with momentum

$$\begin{aligned}v^{(k+1)} &= 0.9v^{(k)} - 0.0001\eta\theta^{(k)} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla L_i(\theta^{(k)}) \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)},\end{aligned}$$

with batch size  $B = 256$ .

Learning rate is the same for all layers with the following heuristic:

- Initialization:  $\eta = 0.1$
- Divide  $\eta$  by 10 at epoch 30 and 60.
- 90 epochs.

Miscellaneous:

- Batch normalization after each convolution and before activation
- No dropout

# DenseNet Results

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17] with Dropout/Drop-path	21 21	38.6M 38.6M	10.18 7.33	5.22 4.60	35.34 28.20	23.30 23.73	2.01 1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110 1202	1.7M 10.2M	11.66 -	5.23 4.91	37.80 -	24.58 -	1.75 -
Wide ResNet [42] with Dropout	16 28 16	11.0M 36.5M 2.7M	- - -	4.81 4.17 -	- - -	22.07 20.50 -	- - 1.64
ResNet (pre-activation) [12]	164 1001	1.7M 10.2M	11.26* 10.56*	5.46 4.62	35.58* 33.47*	24.33 22.71	- -
DenseNet ( $k = 12$ ) DenseNet ( $k = 12$ ) DenseNet ( $k = 24$ )	40 100 100	1.0M 7.0M 27.2M	<b>7.00</b> <b>5.77</b> <b>5.83</b>	5.24 <b>4.10</b> <b>3.74</b>	<b>27.55</b> <b>23.79</b> <b>23.42</b>	24.42 <b>20.20</b> <b>19.25</b>	1.79 1.67 <b>1.59</b>
DenseNet-BC ( $k = 12$ ) DenseNet-BC ( $k = 24$ ) DenseNet-BC ( $k = 40$ )	100 250 190	0.8M 15.3M 25.6M	<b>5.92</b> <b>5.19</b> -	4.51 <b>3.62</b> <b>3.46</b>	<b>24.15</b> <b>19.64</b> -	22.27 <b>17.60</b> <b>17.18</b>	1.76 1.74 -

**Table 2:** Error rates (%) on CIFAR and SVHN datasets.  $k$  denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. “+” indicates standard data augmentation (translation and/or mirroring). \* indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

# Outline

## 1 Foundations of CNN

- Convolution layer
- Pooling layer
- Data preprocessing

## 2 Famous CNN

- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

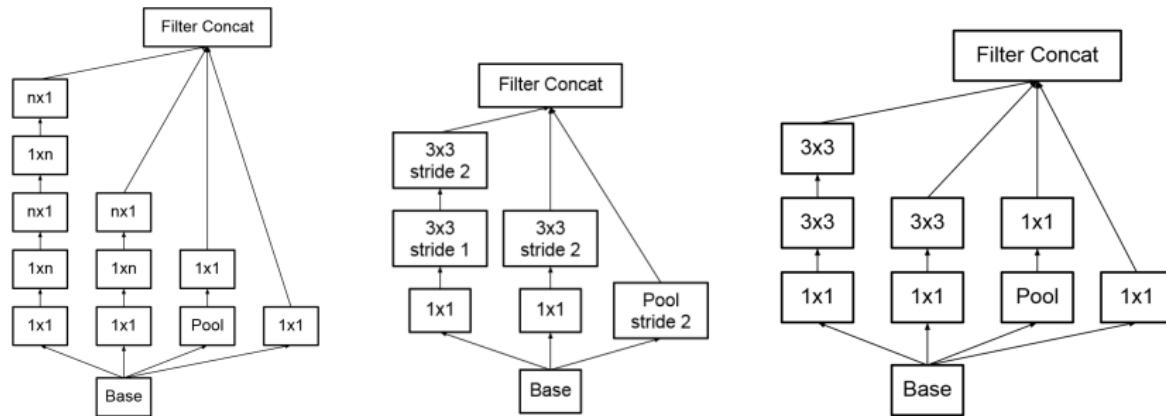
## 3 Applications

- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

# Inception V2-V3

Based on GoogLeNet Inception module

[“Rethinking the inception architecture for computer vision”, Szegedy, Vanhoucke, et al. 2016]



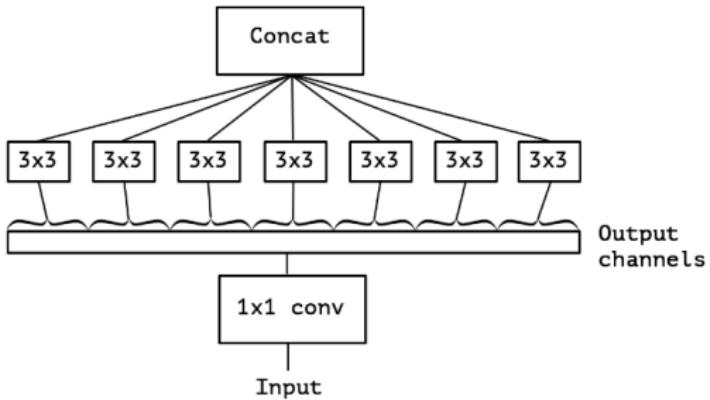
New ideas:

- Using asymmetric convolutions  $1 \times n$  and  $n \times 1$  (for  $n = 3, 5, 7$ ) can be useful in the middle layers of the networks for feature maps of size  $m \times m$  (for  $12 \leq m \leq 20$ ).
- Label smoothing using a uniform distribution over labels

# Xception

[“Xception: Deep learning with depthwise separable convolutions”, Chollet 2017]

Stands for “Extreme Inception” and builds upon Inception module in GoogLeNet.

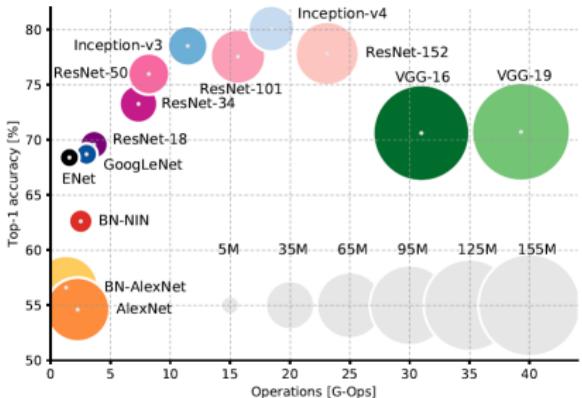
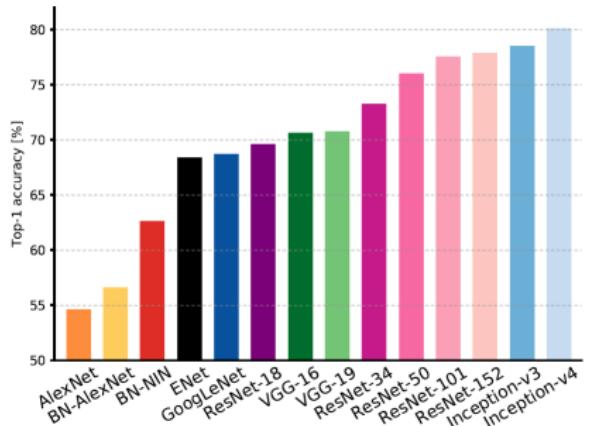


The main ideas:

- Perform  $1 \times 1$  convolutions
- Apply  $3 \times 3$  (or other filter size) convolutions to each previous feature map (the one created by  $1 \times 1$  convolutions) separately.

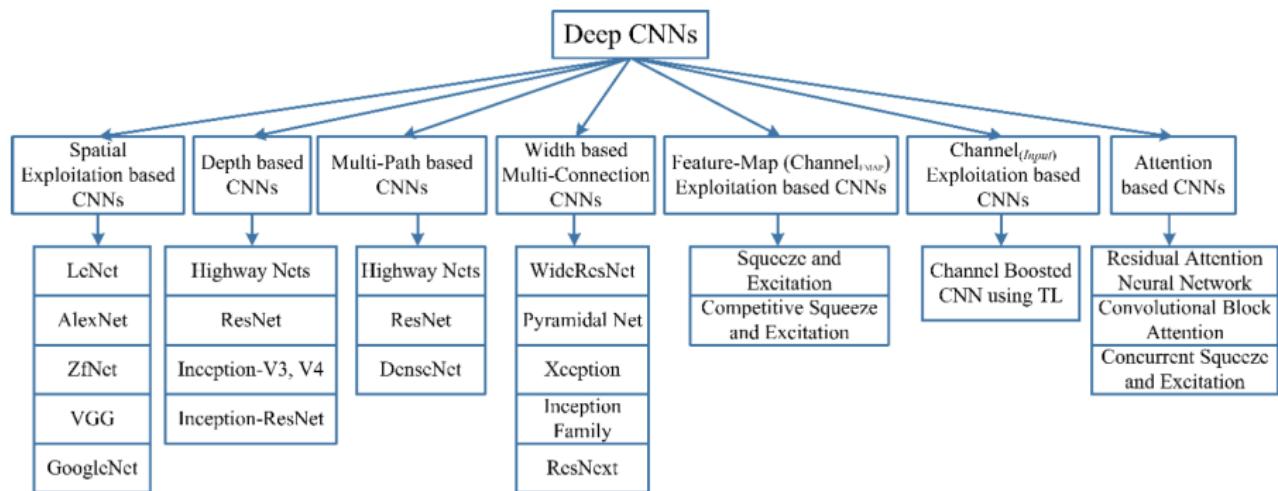
→ Decoupled the depth ( $1 \times 1$  convolutions) and the spatial transformations (convolutions on each feature map separately).

# Comparison of several CNN



[“An analysis of deep neural network models for practical applications”, Canziani et al. 2016]

# CNN Taxonomy



See this very detailed review paper ["A survey of the recent architectures of deep convolutional neural networks", Khan et al. 2020]