

Deep Learning for  
Computer Vision  
Part II: Convolutional  
Neural Networks

# Computer Vision

## Introduction

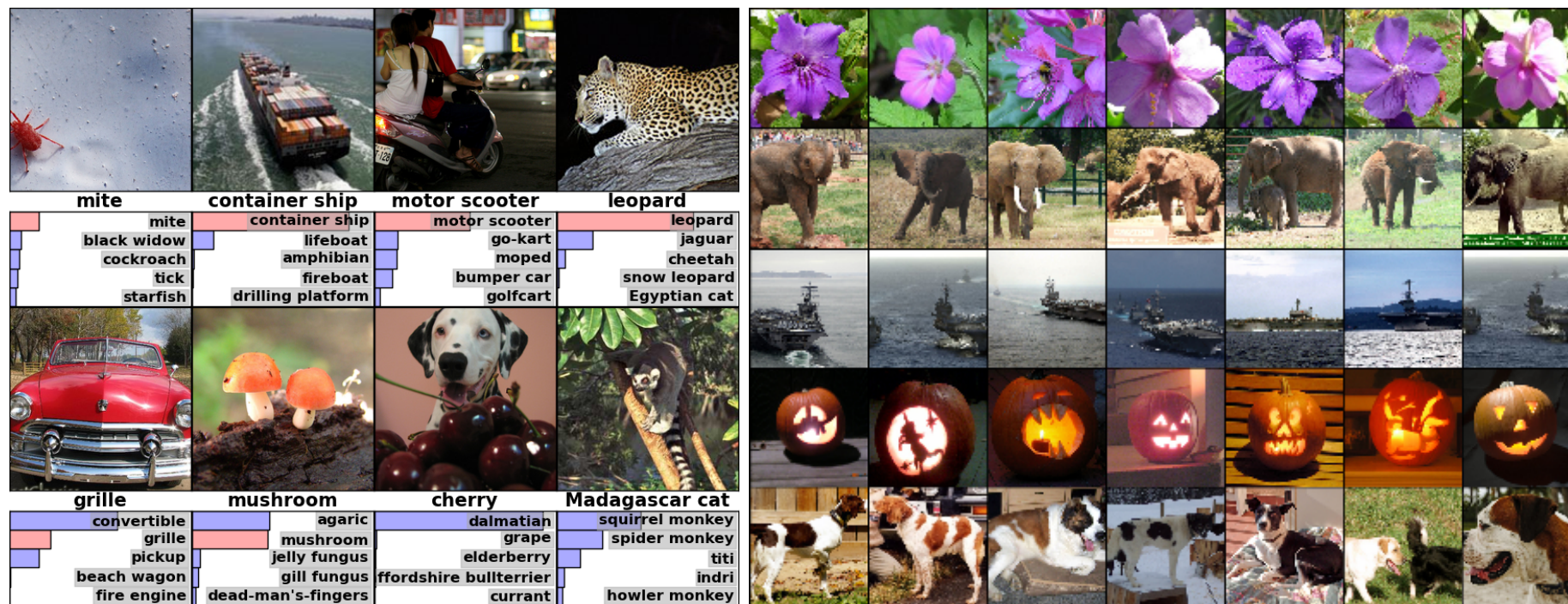


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 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.

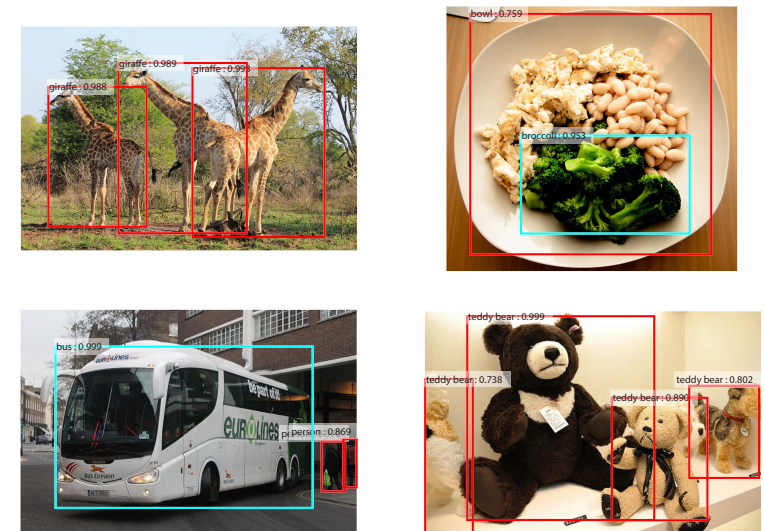
[Figure from Krizhevsky, Sutskever and Hinton 2012 – Predictions on ImageNet with CNNs]

# Today

- Multilayered neural networks is an essential tool in computer vision and image analysis
- Enhancement, detection, segmentation, recognition,...



[Figure from Ledig et al. 2017]



[Figure from Ren et al. 2016 - Faster R-CNN]

## Outline

1. Introduction to neural networks – this week  
Basics of neural networks
2. Convolutional neural networks– 13.12  
Basic applications of deep learning to image analysis and computer vision
3. Advanced topics and applications – 20.12  
Unsupervised learning and recurrent neural networks

# Recap from lecture on introduction to neural networks

Hidden layers

$$h_l = \sigma(\mathbf{W}_l h_{l-1} + b_l)$$

$$h_0 = \mathbf{x}, \quad h_L = f(\mathbf{x}; \theta)$$

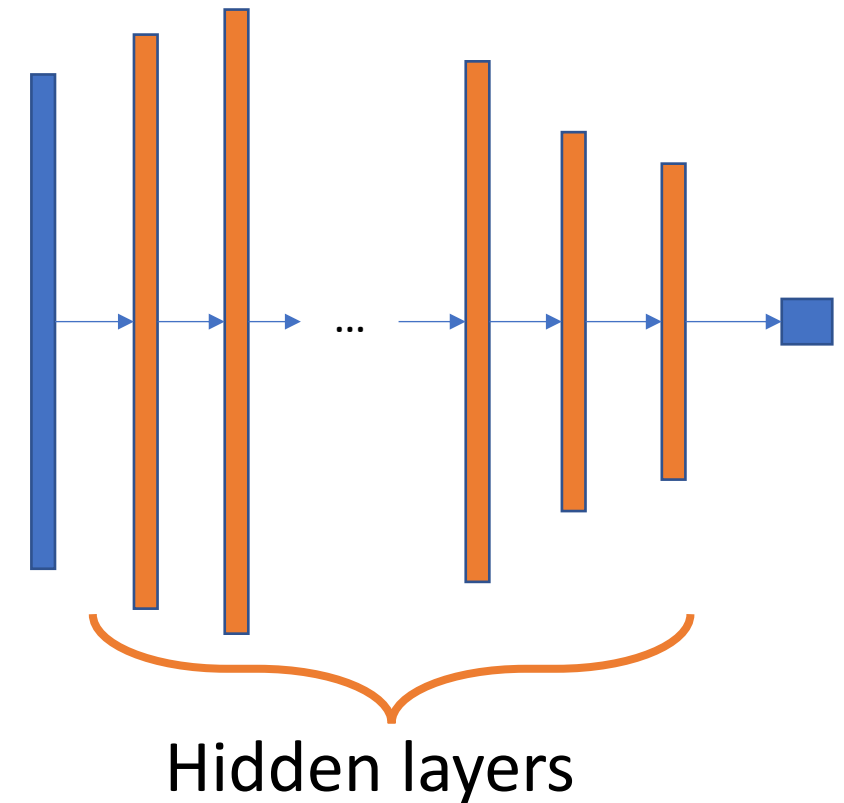
Function view

$$f_l(h_{l-1}; \theta_l) = \sigma(\mathbf{W}_l h_{l-1} + b_l)$$

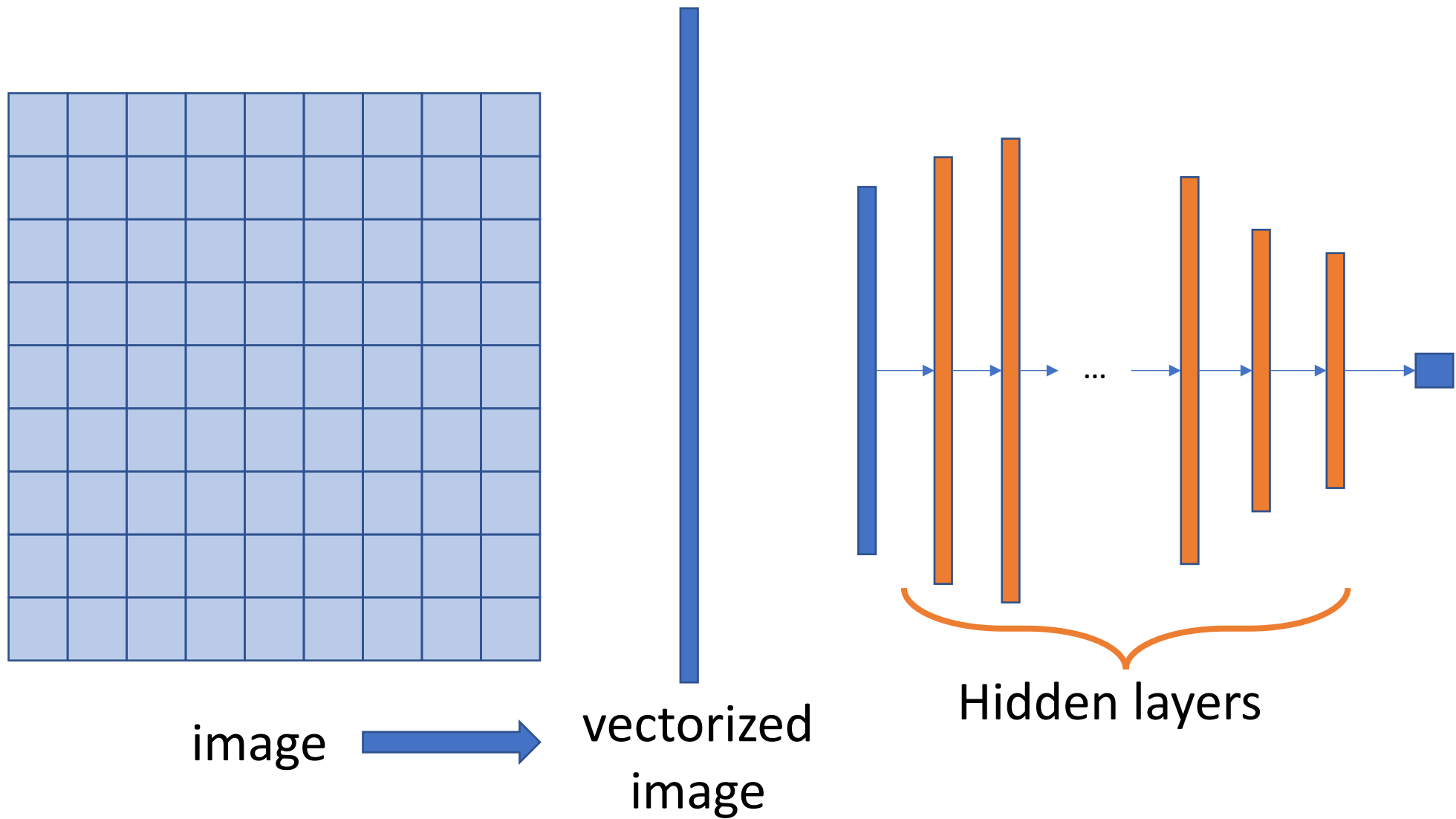
$$f(\mathbf{x}; \theta) = f_L \circ \dots \circ f_2 \circ f_1(\mathbf{x})$$

$$\theta = \{\theta_L, \dots, \theta_1\}$$

We always considered  $\mathbf{x}$  as a vector.  
What to do when  $\mathbf{x}$  is an image?



# Naïve approach



# The activation is a bit problematic

$$a_l = \mathbf{W}_l h_{l-1} + b_l$$

- The linear transformation has two issues

1. Fully connected links leads to too many parameters.

Assume the input is an image of size  $N_0 \times M_0$ , i.e.  $\mathbf{x} \in \mathbb{R}^{N_0 \times M_0}$

And the hidden layer has  $d_1$  size, i.e.  $h_1 \in \mathbb{R}^{d_1}$

$$\mathbf{W}_1 \in \mathbb{R}^{(d_1) \times (N_0 \times M_0)}$$

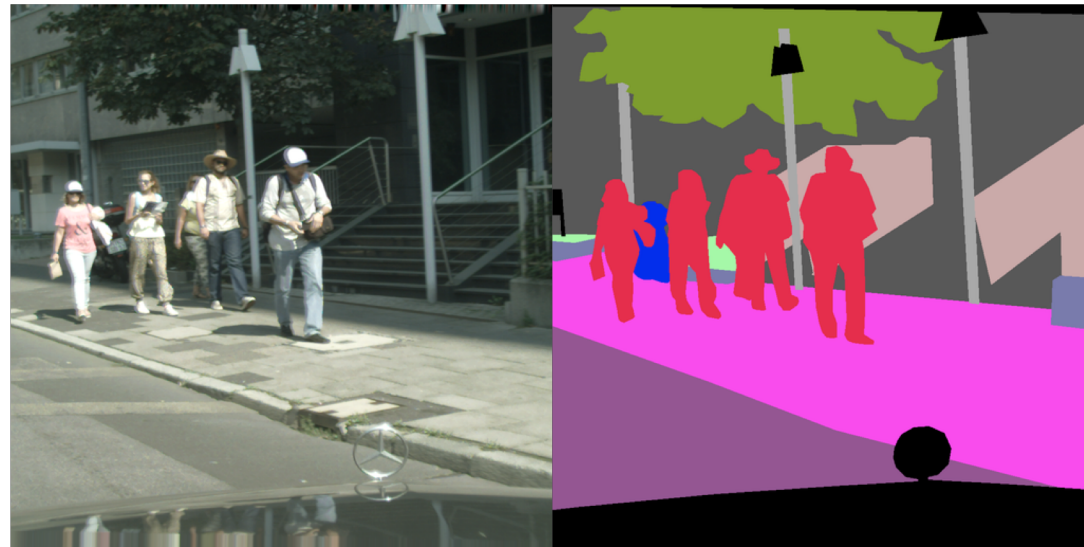
For example,  $(N_0 \times M_0) = (64, 64)$  and  $(d_1) = (128)$  leads to 524288 parameters in only the first layer!

**With a huge compression from 64x64 dimensions to 128!**

# The activation is a bit problematic

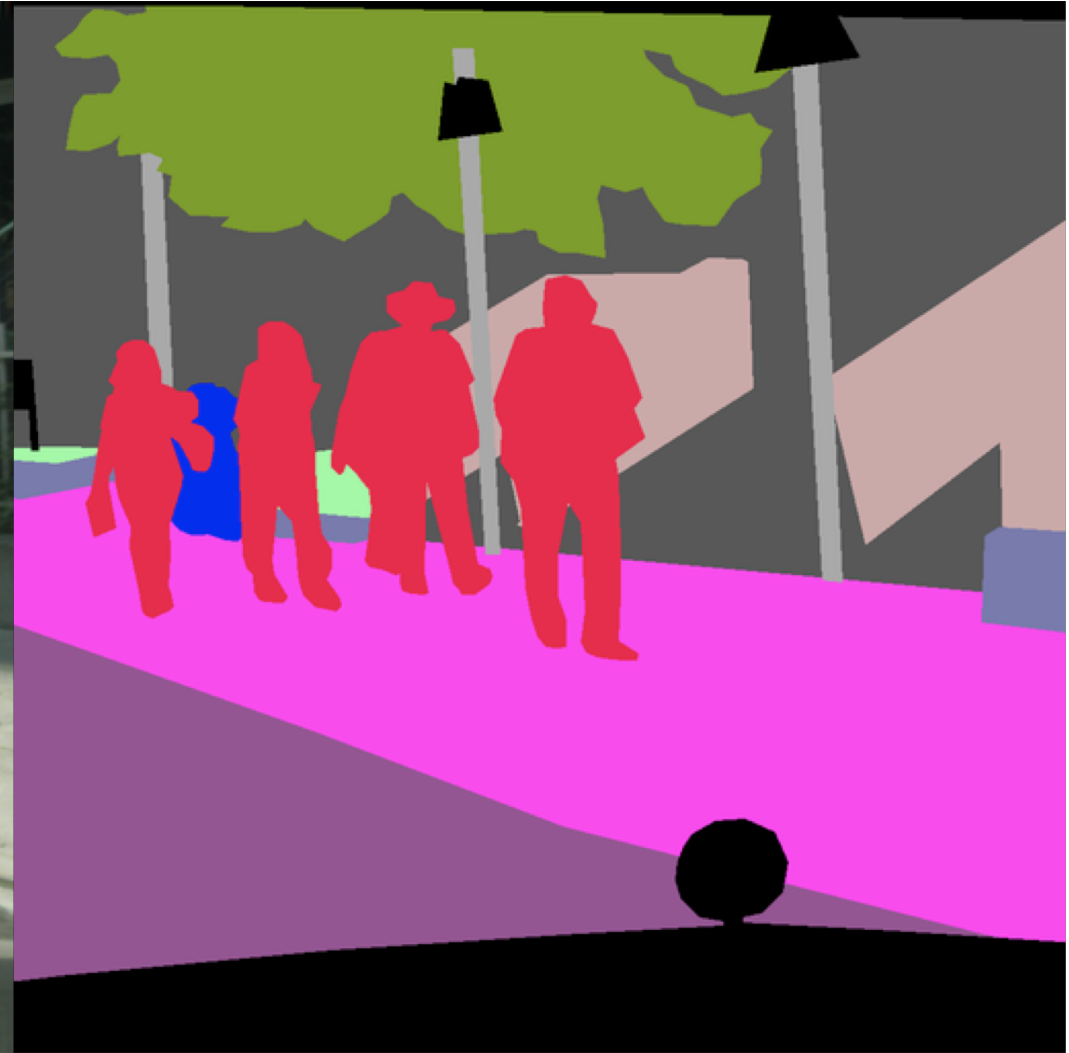
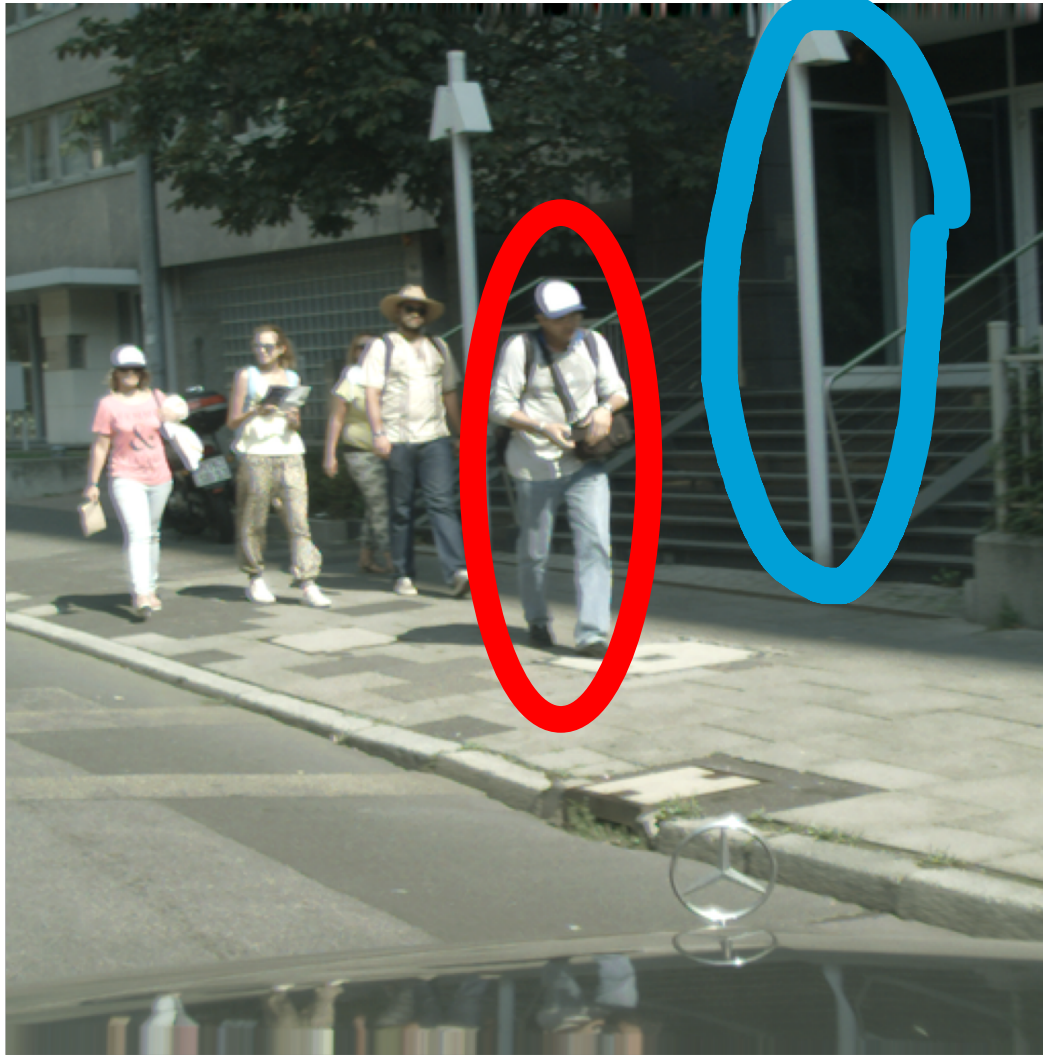
$$a_l = \mathbf{W}_l h_{l-1} + b_l$$

- The linear transformation has three issues
  1. Fully connected links leads to too many parameters.
  2. Images are composed of a hierarchy of local statistics





# Computer Vision



# The activation is a bit problematic

$$a_l = \mathbf{W}_l h_{l-1} + b_l$$

- The linear transformation has two issues
  1. Fully connected links leads to too many parameters.
  2. Images are composed of a hierarchy of local statistics
  3. Lack of translation invariance



# The activation is a bit problematic

$$a_l = \mathbf{W}_l h_{l-1} + b_l$$

- The linear transformation has two issues
  1. Fully connected links leads to too many parameters.
  2. Images are composed of a hierarchy of local statistics

Fully connected architecture  $a_{l,k} = \sum_j w_{l,kj} h_{l-1,j} + b_{l,k}$

Convolutional architecture  $a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$

# The activation is a bit problematic

$$a_l = \mathbf{W}_l h_{l-1} + b_l$$

- The linear transformation has two issues
  1. Fully connected links leads to too many parameters.
  2. Images are composed of a hierarchy of local statistics

Fully connected architecture

$$h_{l,k} = \sigma \left( \sum_j w_{l,k,j} h_{l-1,j} + b_{l,k} \right)$$

Convolutional architecture

$$h_{l,k} = \sigma \left( \sum_j w_{l,k}^* h_{l-1}^* + b_{l,k} \right)$$



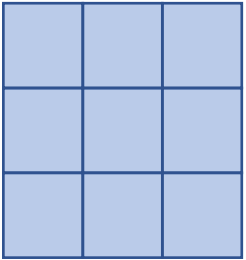
Convolutional layers



# Remember convolution $w * x$

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

Image: x

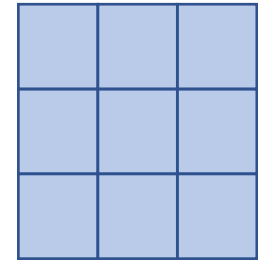


Convolution kernel: w

# Remember convolution $w * x$

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

Image: x

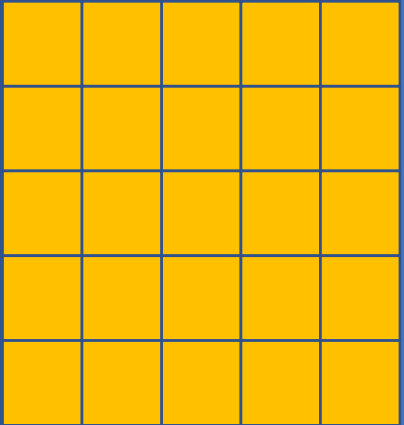


Convolution kernel: w

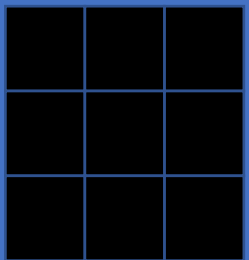
$$a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$$



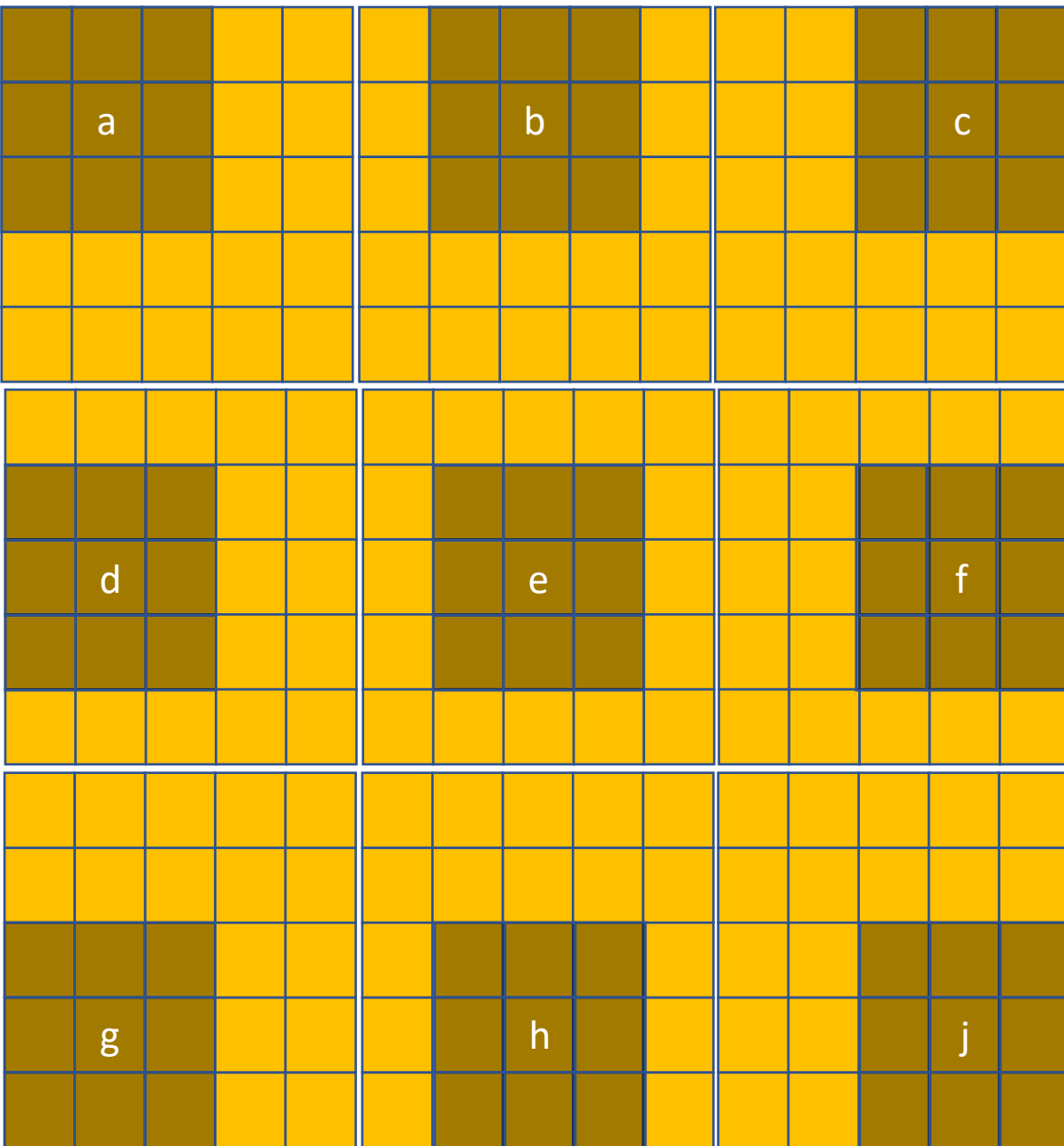
# Computer Vision



Image



kernel



a	b	c
d	e	f
g	h	j

# Convolutions instead of projections

$$a_{l,k} = \sum_j w_{l,kj} h_{l-1,j} + b_{l,k}$$

- Each  $h_{l-1,j}$  is a number and so is  $a_{l,k}$
- Each  $h_l$  is a vector of neurons and  $a_l$  is a vector of activation
- There is a separate  $w_{l,kj}$  that is linking each neuron  $h_{l-1,j}$  to each  $a_{l,k}$ .
- High dimensions of  $h_{l-1}$  and  $a_l$  lead to high number of weights

$$a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$$

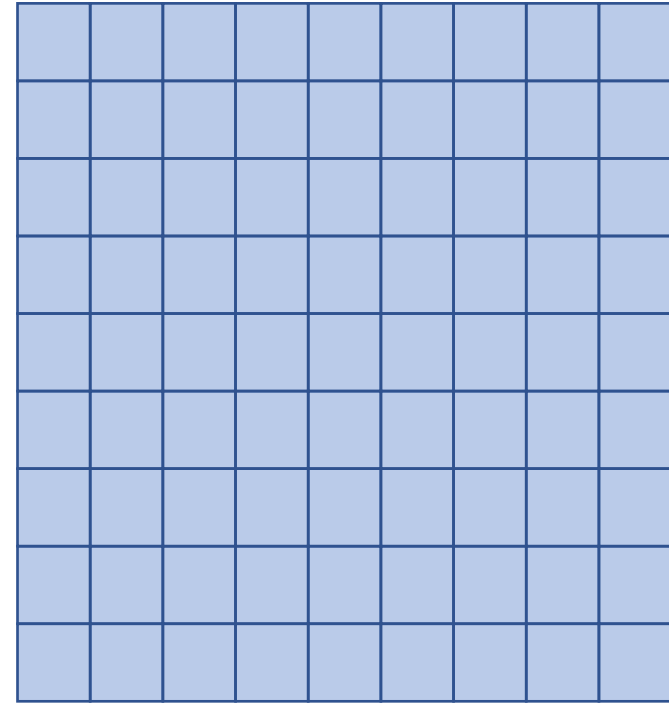
- Each  $h_{l-1,j}$  is **an image of neurons** and so is  $a_{l,k}$ .
- There is a separate **convolutional kernel** linking images  $h_{l-1,j}$  and  $a_{l,k}$ . Same kernel applies to the entire image of neurons
- In the literature  $h_{l,j}$  are called **channels**
- $w_{l,kj}$  are convolutional filters.

Nonlinearities following activations remain similar

# Image of neurons - channel



Fully connected  
Vector of neurons form a layer

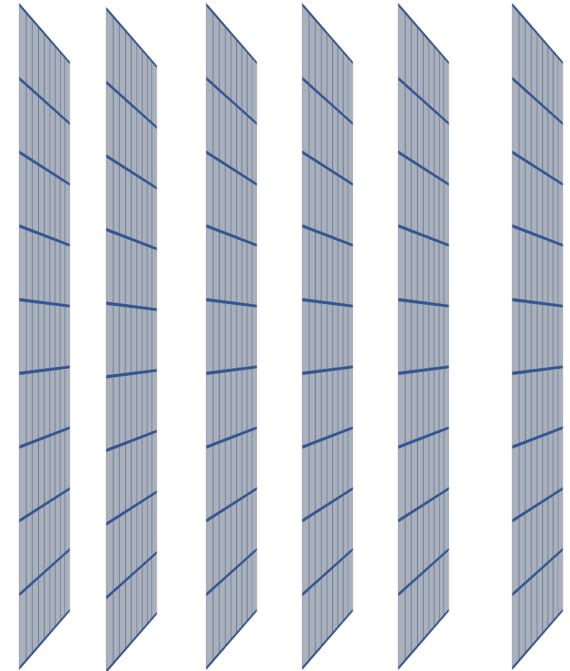


Convolutional architecture  
Image of neurons form a channel

# Vector of channels forms a layer

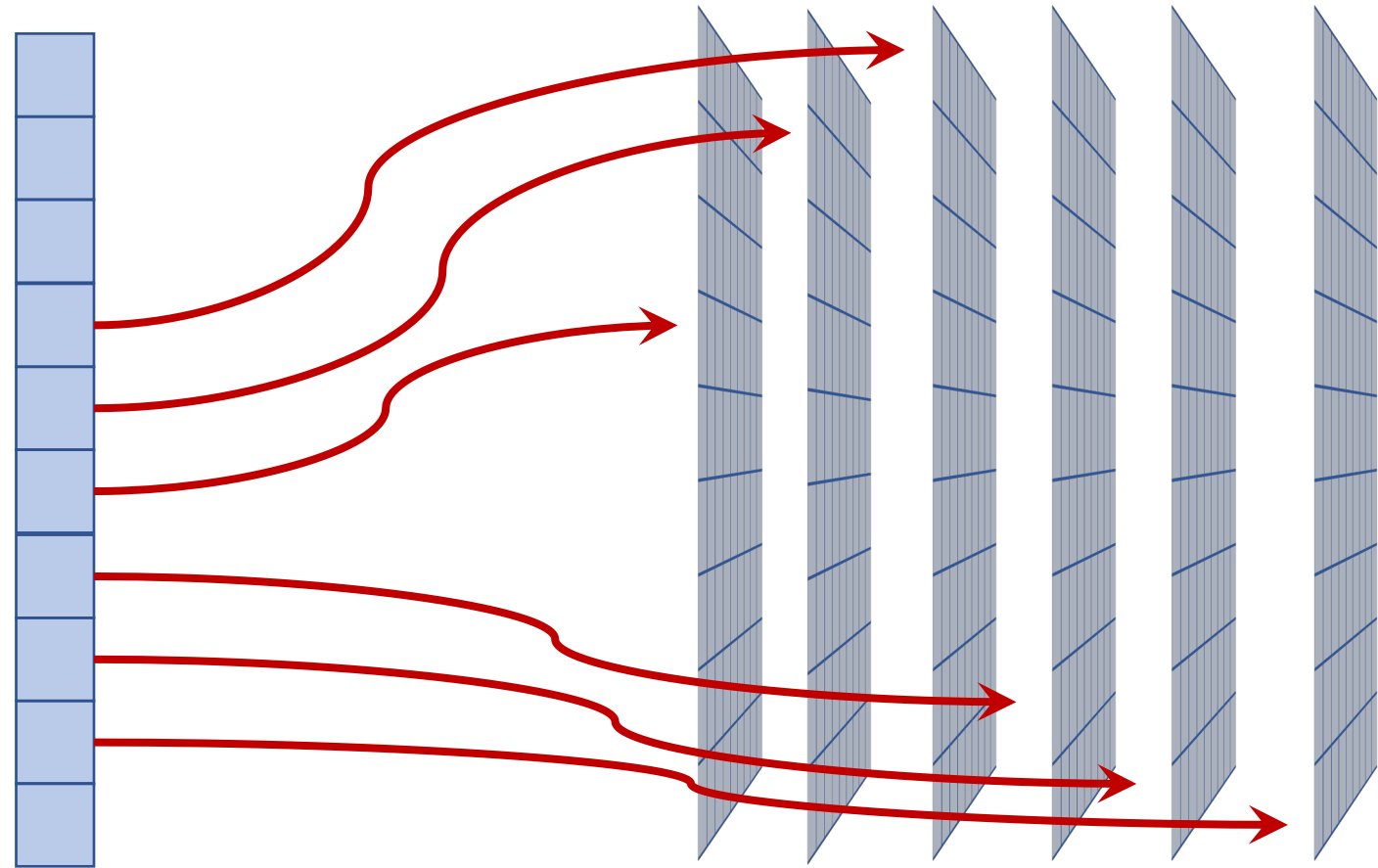


Fully connected  
Vector of neurons form a layer



Convolutional architecture  
Vector of channels form a layer

# Analogy between two: each neuron becomes a channel



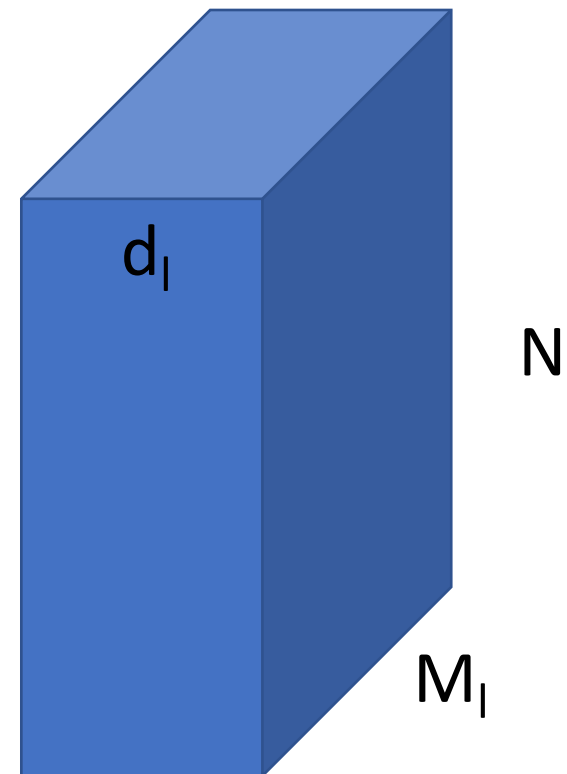
Fully connected  
Vector of neurons form a layer

Convolutional architecture  
Vector of channels form a layer

# Tensor representation



Fully connected architecture:  
Vector representation of layers



Convolutional architecture:  
Tensor representation of layers

We can also use the same graphical representation



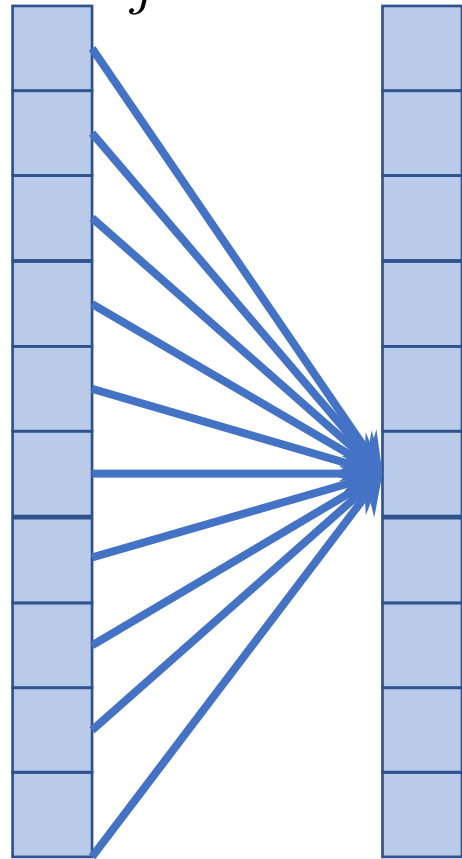
Fully connected architecture:  
 $d_1$  neurons



Convolutional architecture:  
 $d_1$  channels

# Connections represent convolutions

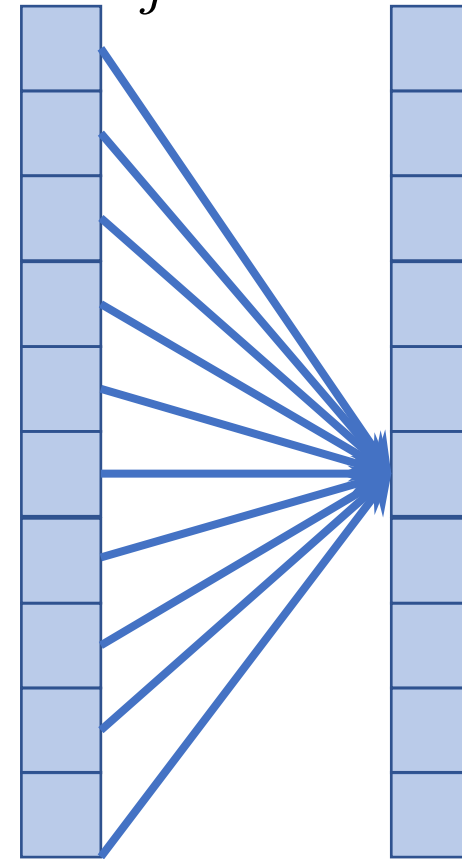
$$a_{l,k} = \sum_j w_{l,kj} h_{l-1,j} + b_{l,k}$$



Fully connected link

Multiplication followed by addition

$$a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$$

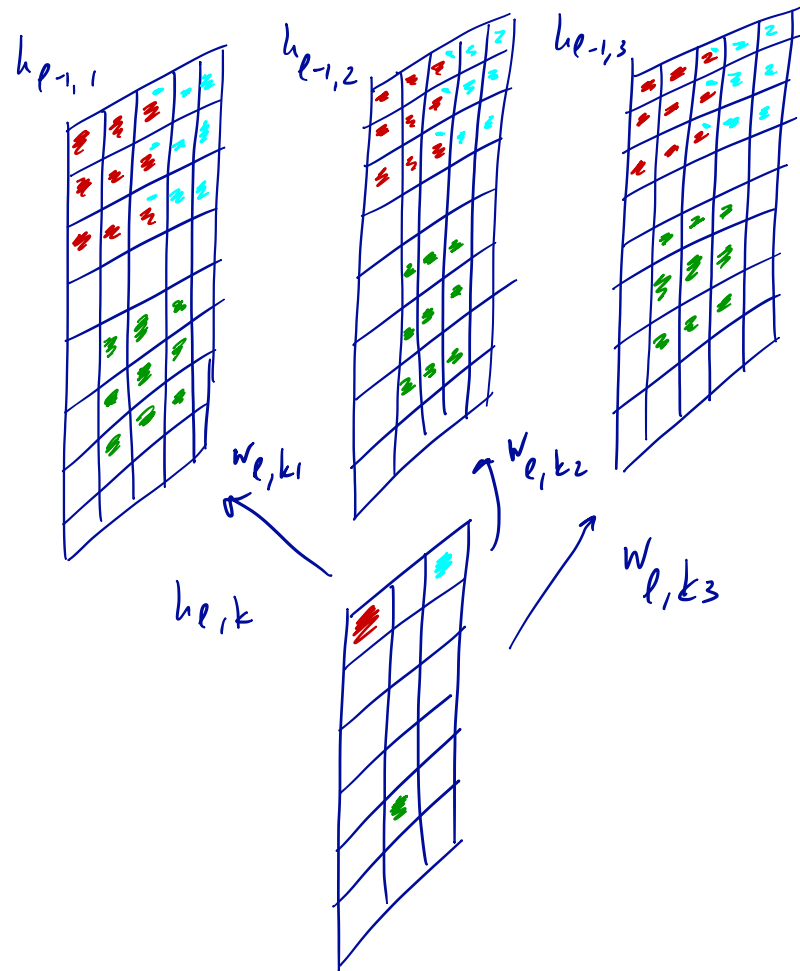


Convolutional link

Convolution followed by addition

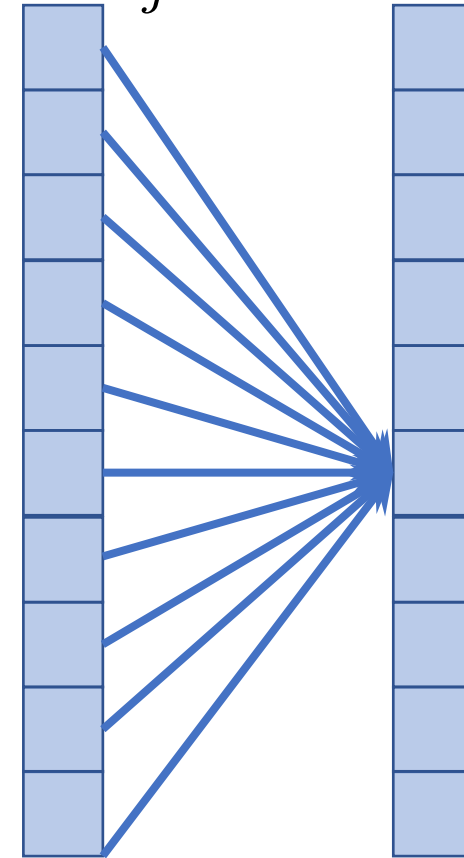


# Connections represent convolutions



Same filter is used for all neurons in the same channel: **Weight sharing**

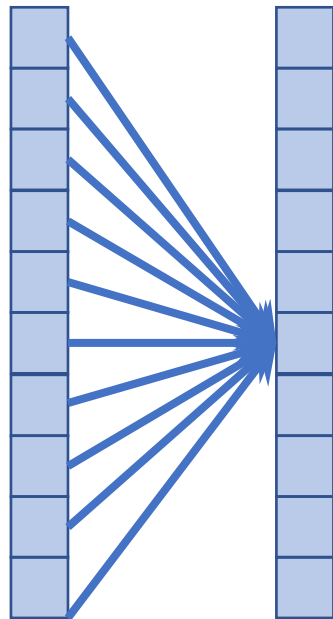
$$a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$$



Convolutional link  
Convolution followed by addition

# Layer size and number of parameters at the beginning

Fully connected



$$x \in \mathbb{R}^{N_0 \times M_0}$$

$$h_1 \in \mathbb{R}^{d_1}$$

$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (N_0 \times M_0)}$$

Convolutional

$$x \in \mathbb{R}^{N_0 \times M_0}$$

$$h_1 \in \mathbb{R}^{d_1 \times (N_1 \times M_1)}$$

$$\mathbf{W}_1 = \{w_{1,1}, w_{1,2}, \dots, w_{1,d_1}\}$$

$$w_{1,j} \in \mathbb{R}^{k_1 \times k_2}$$

$(k_1 \times k_2)$  : kernel size

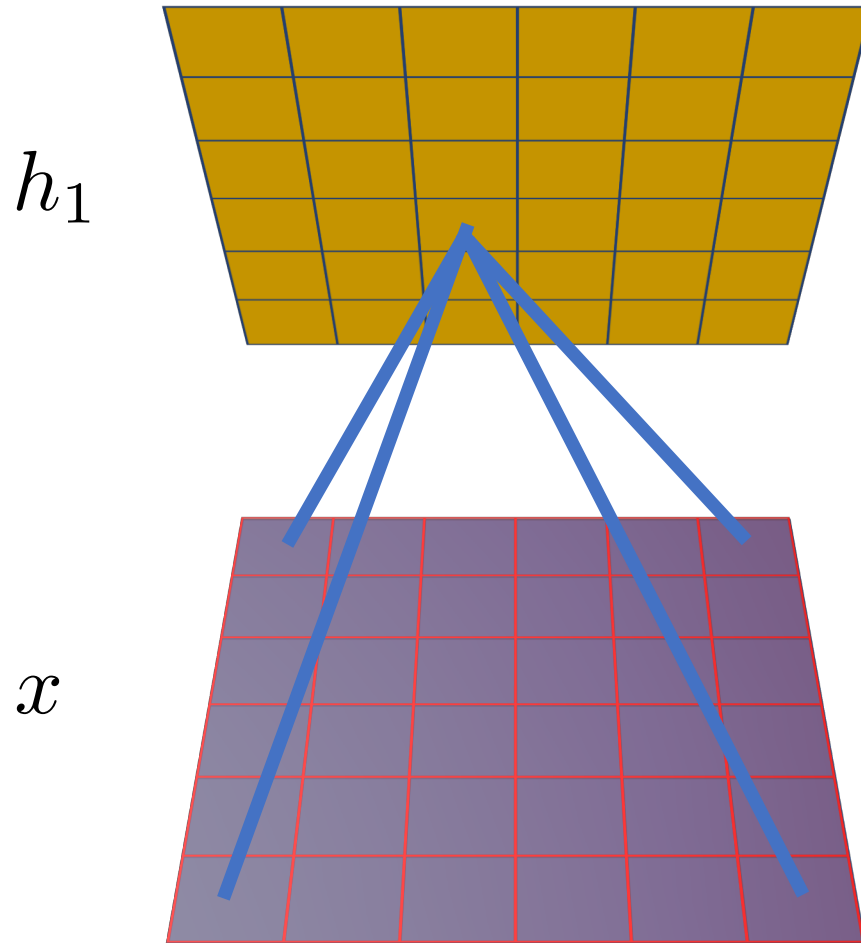
$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (k_1 \times k_2)}$$

**Larger layers with sparser connections with lower number of parameters**

# Sparser connections

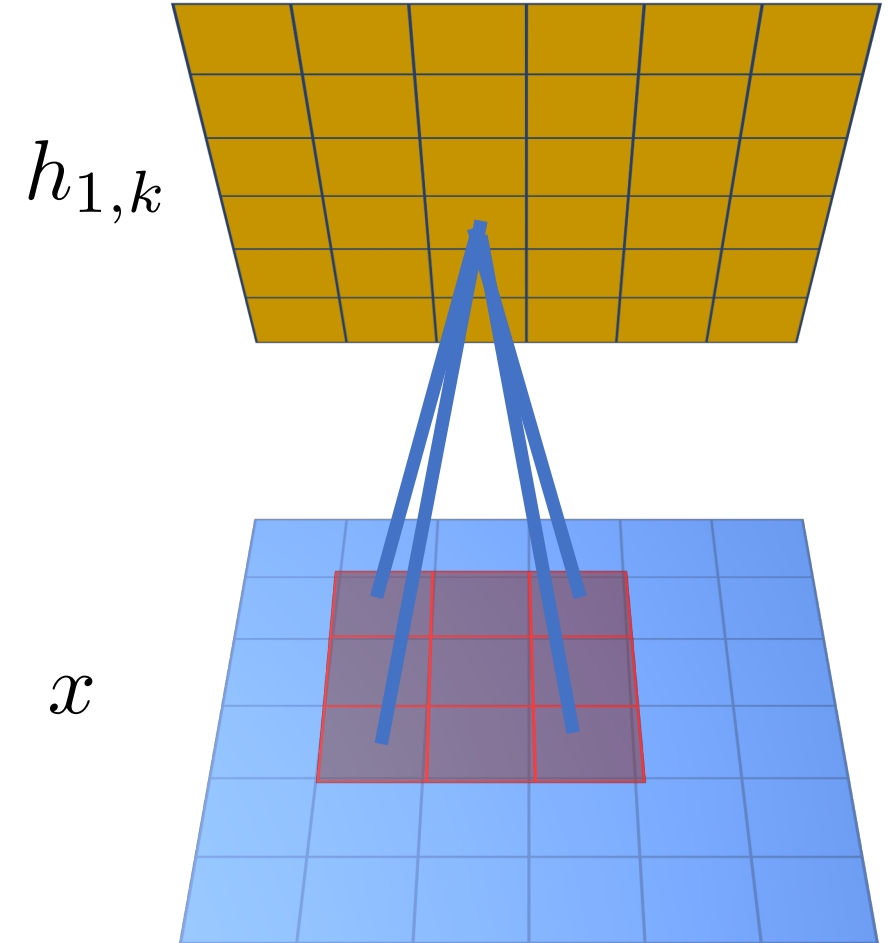
Naïve approach – fully connected

$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (N_0 \times M_0)}$$



Convolutional link

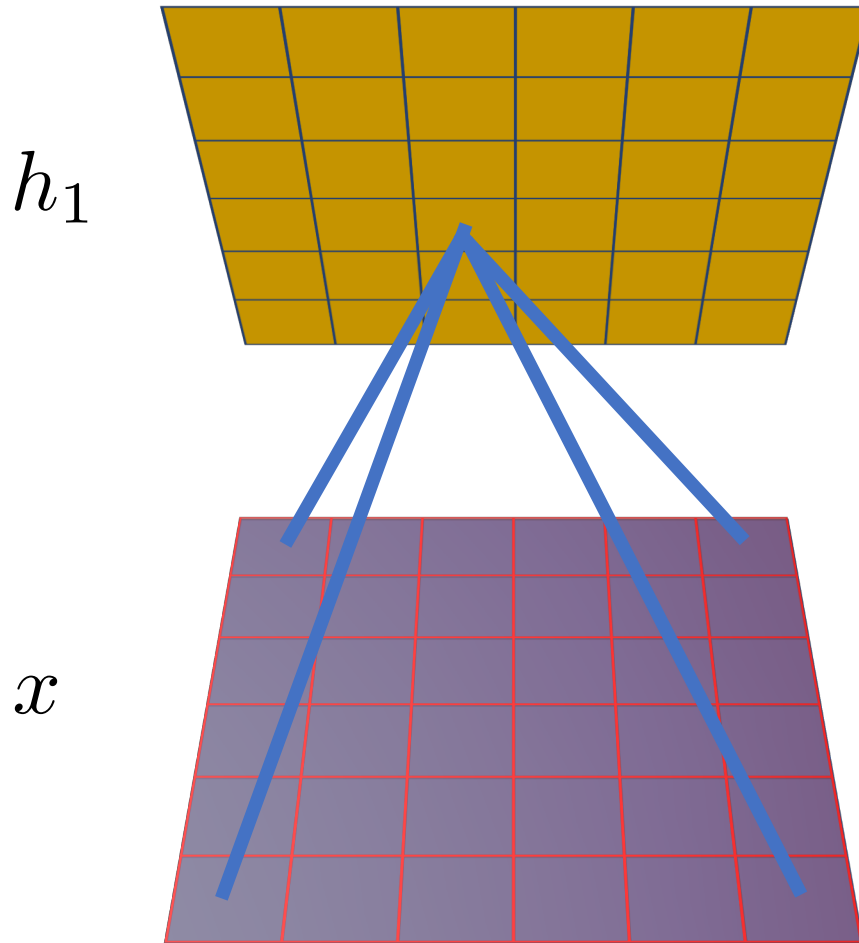
$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (k_1 \times k_2)}$$



# Weight sharing – fewer parameters

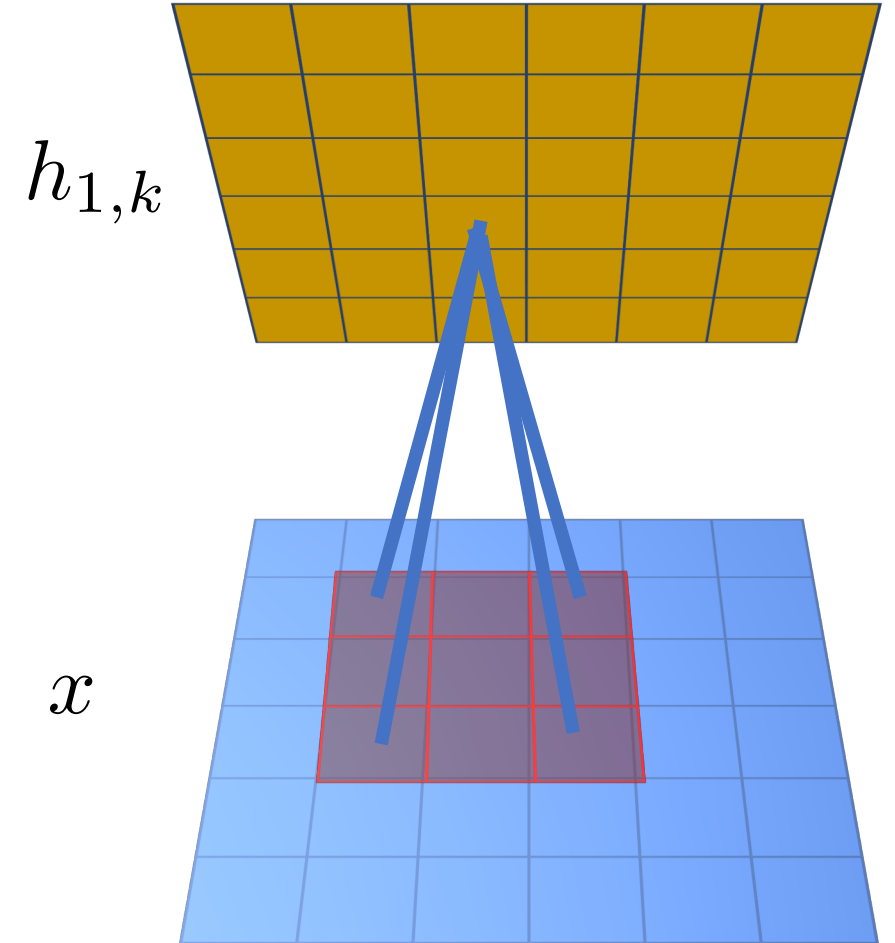
Naïve approach – fully connected

$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (N_0 \times M_0)}$$



Convolutional link

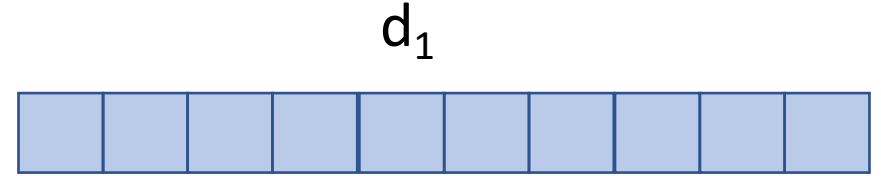
$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (k_1 \times k_2)}$$



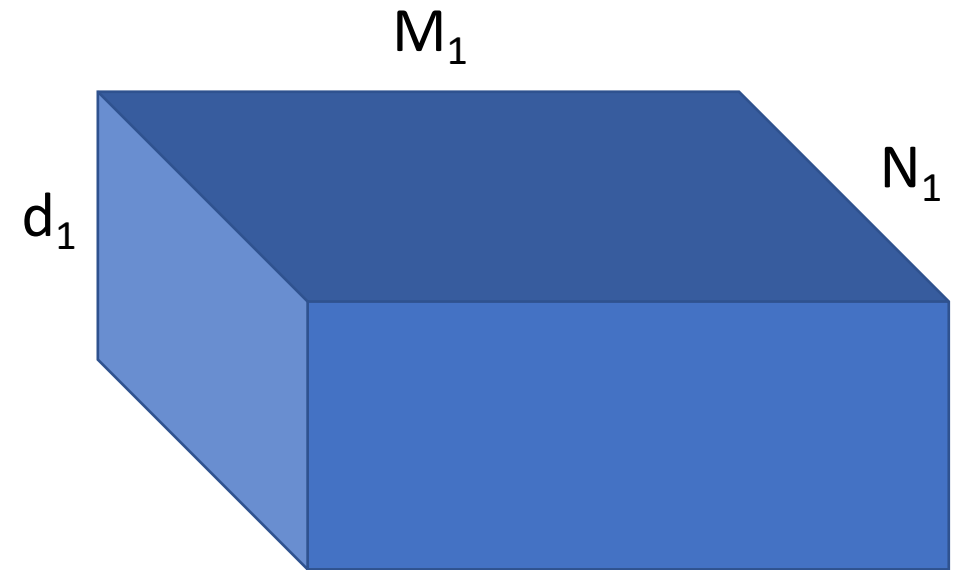
# Larger hidden layers

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

Image:  $x \in \mathbb{R}^{M_0 \times N_0}$

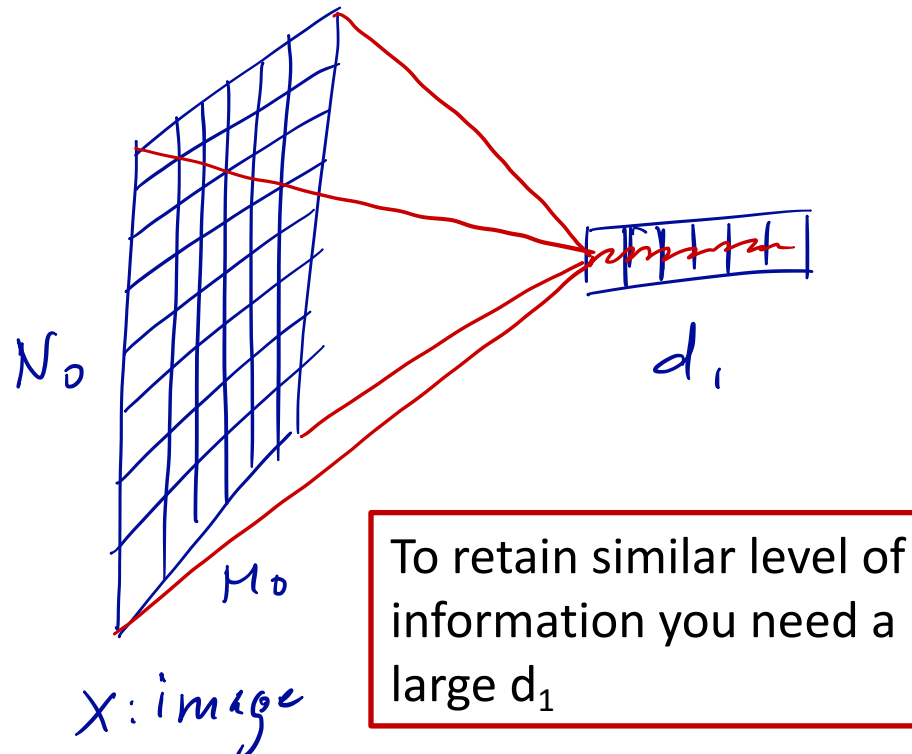


Fully connected:  $h_1 \in \mathbb{R}^{d_1}$

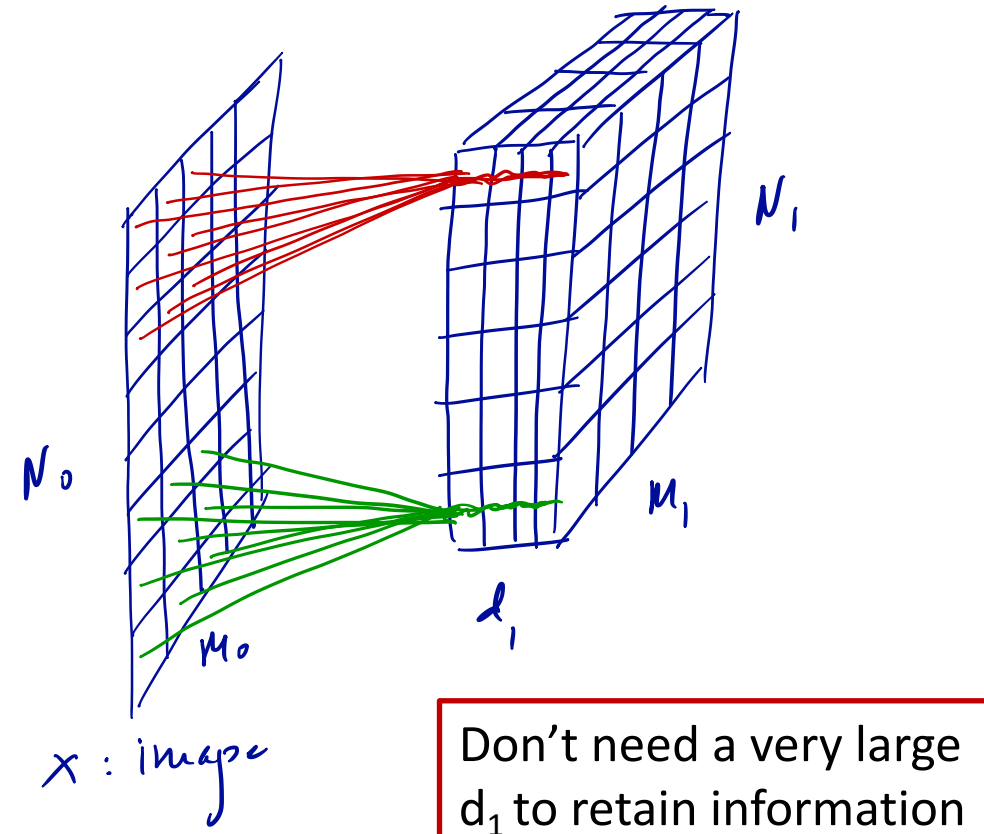


Convolutional:  $h_1 \in \mathbb{R}^{d_1 \times (N_1 \times M_1)}$

# Local vs. global information gathering



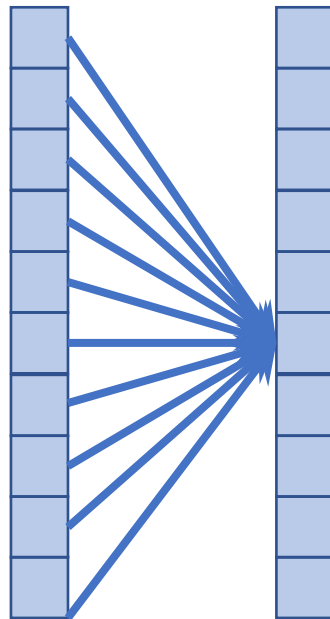
Fully connected architecture:  
The entire image needs to be represented in a single vector



Convolutional architecture:  
Local neighborhoods are represented in a single vector

# Layer size and number of parameters at an intermediate layer

Fully connected



$$h_{l-1} \in \mathbb{R}^{d_{l-1}}$$

$$h_l \in \mathbb{R}^{d_l}$$

$$\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$$

Convolutional

$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times (N_{l-1} \times M_{l-1})}$$

$$h_l \in \mathbb{R}^{d_l \times (N_l \times M_l)}$$

$$\mathbf{W}_l = \{w_{l,jk}\},$$

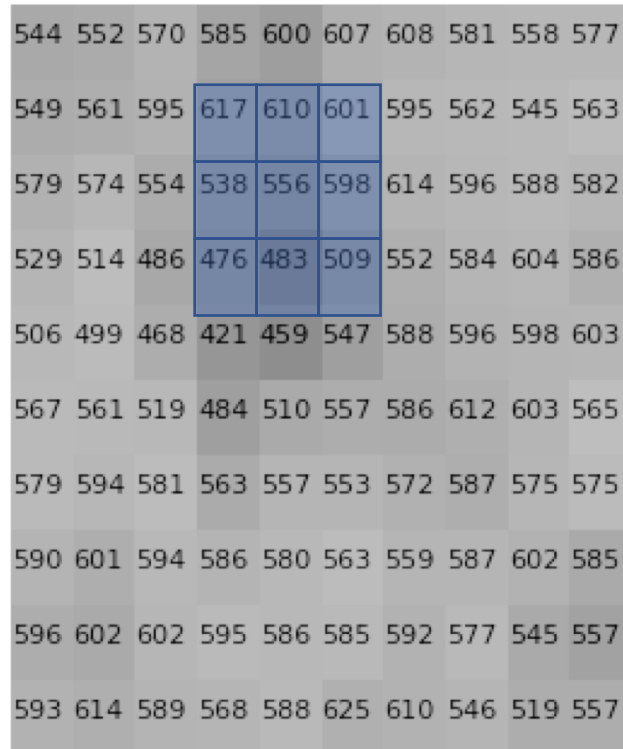
$$j = 1, \dots, d_{l-1}, k = 1, \dots, d_l$$

$$\mathbf{W}_l \in \mathbb{R}^{d_{l-1} \times d_l \times (k_1 \times k_2)}$$

**Larger** layers with **sparser** connections with lower number of parameters

## Channel size

- Channel size is linked with kernel size and the type of convolution



544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

$$a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$$

- When the kernel is placed in the image – no problem



# Channel size

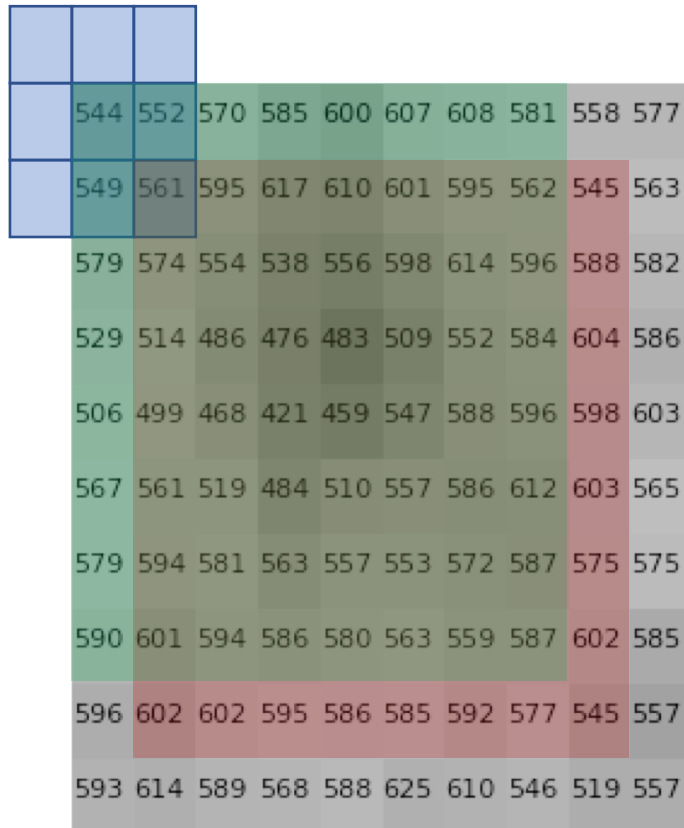
- Channel size is linked with kernel size and the type of convolution

	544	552	570	585	600	607	608	581	558	577
	549	561	595	617	610	601	595	562	545	563
	579	574	554	538	556	598	614	596	588	582
	529	514	486	476	483	509	552	584	604	586
	506	499	468	421	459	547	588	596	598	603
	567	561	519	484	510	557	586	612	603	565
	579	594	581	563	557	553	572	587	575	575
	590	601	594	586	580	563	559	587	602	585
	596	602	602	595	586	585	592	577	545	557
	593	614	589	568	588	625	610	546	519	557

$$a_{ij} = \sum_p \sum_q x_{(i-p)(j-q)} w_{(p)(q)}$$

- When the kernel is placed in the image – no problem
- When it is placed on the boundary – it is not well defined
- Out-of-boundary values are not defined
- Two options:
  1. Valid convolution: only evaluate convolution when all the elements are defined
  2. Padding (Same): pad the boundaries so that result of the convolution will have the same size

## Valid convolution



- If the kernel is centered, i.e.  $w_{(0)(0)}$  is the center of the kernel, then convolution can only be evaluated within the red area
- You lose a pixel at each end of the picture
- If the kernel center is the top left corner, then the green area is the valid area
- You lose two pixels at the bottom and right of the image

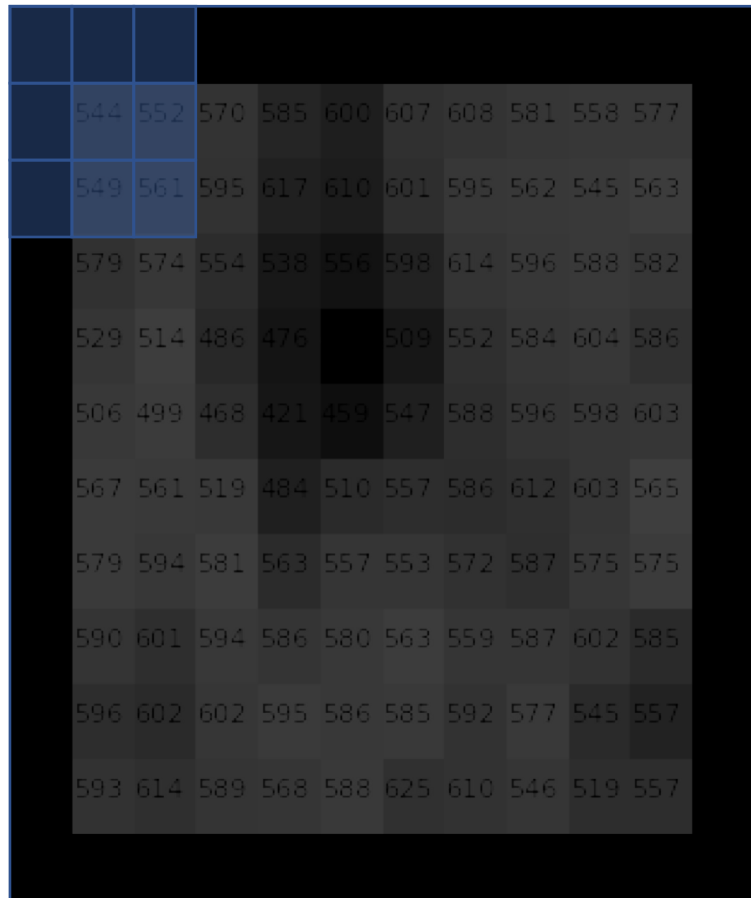
$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times (M_{l-1} \times N_{l-1})}$$

$$w_{l,..} \in \mathbb{R}^{k_1 \times k_2}$$

$$h_l \in \mathbb{R}^{d_l \times (M_l \times N_l)}$$

$$M_l = M_{l-1} - k_1 + 1 \quad N_l = N_{l-1} - k_2 + 1$$

## Same padding



- Alternatively you can pad the image on the boundaries so that channels will have the same size across layers

$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times (M_{l-1} \times N_{l-1})}$$

$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times ((M_{l-1} + k_1 - 1) \times (N_{l-1} + k_2 - 2))}$$

$$h_l \in \mathbb{R}^{d_l \times (M_l \times N_l)}$$

$$M_l = M_{l-1} \quad N_l = N_{l-1}$$

- Where you pad depends on where the center of the kernel is.
- Commonly you would use centered kernels – padding around the image as shown on the left
- The value you pad is a parameter, 0 is used often but you can use symmetric padding for certain applications

## Question: Number of parameters

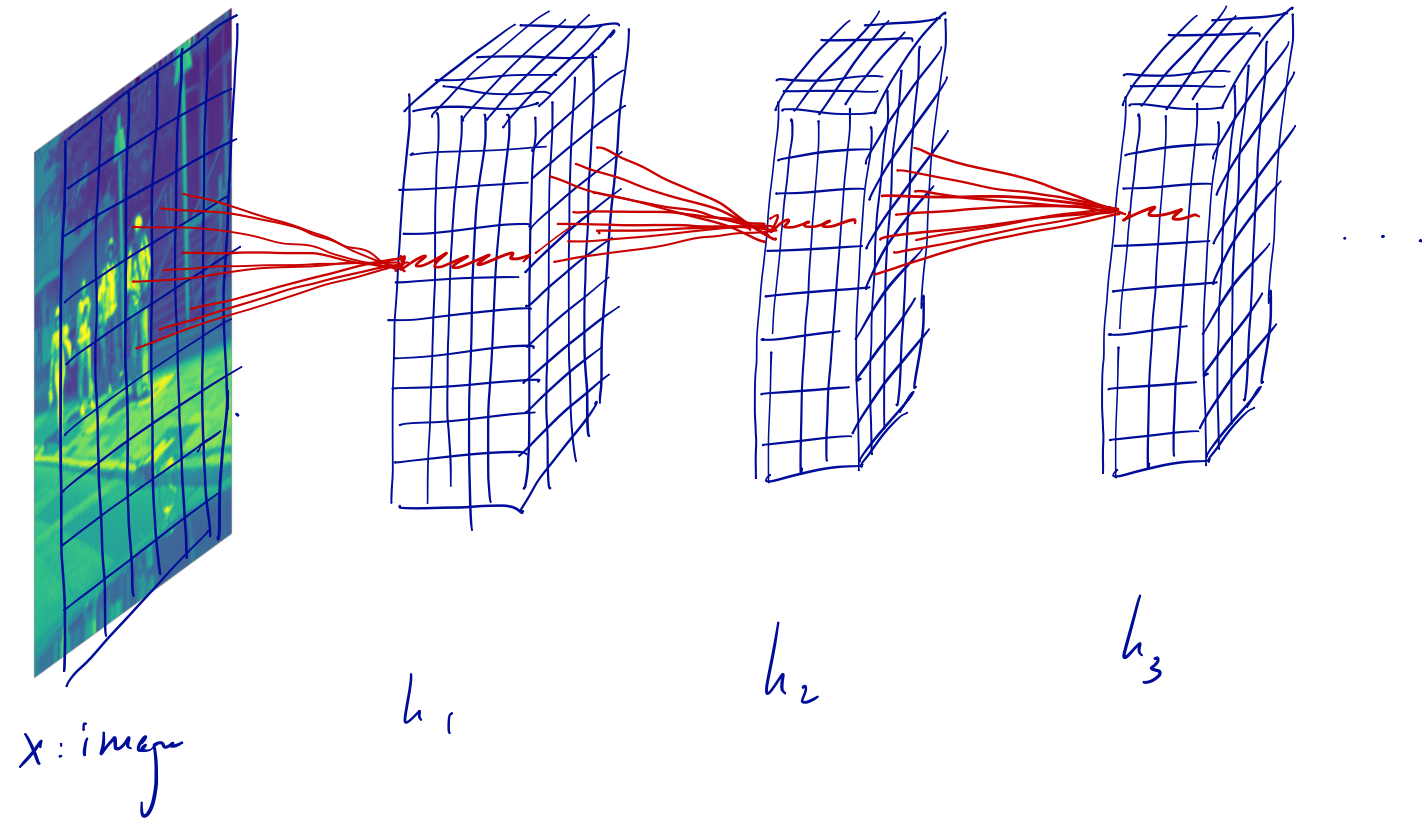
- Classification network with 16 possible output classes
- How many parameters are in the following fully connected networks, where each arrow is a fully connected link

$$x \in \mathbb{R}^{64 \times 64} \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 16$$

- How many parameters and how many hidden neurons in each layer in the following convolutional neural network, where double arrows are convolutional links with 5x5 kernels and “valid” convolutions, and arrows are fully connected links

$$x \in \mathbb{R}^{64 \times 64} \Rightarrow 16 \Rightarrow 16 \Rightarrow 16 \rightarrow 16$$

# Hierarchically aggregating local spatial features

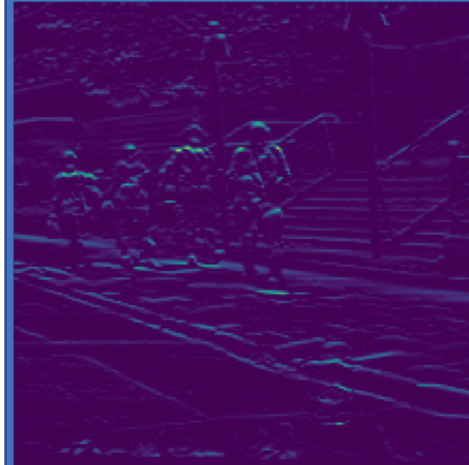


Extracting task specific features from the images.  
As the layers progress, more global information is encoded.

# Hierarchically gathering local spatial statistics



Layer 1



x derivative after Relu



y derivative after Relu

Layer 2

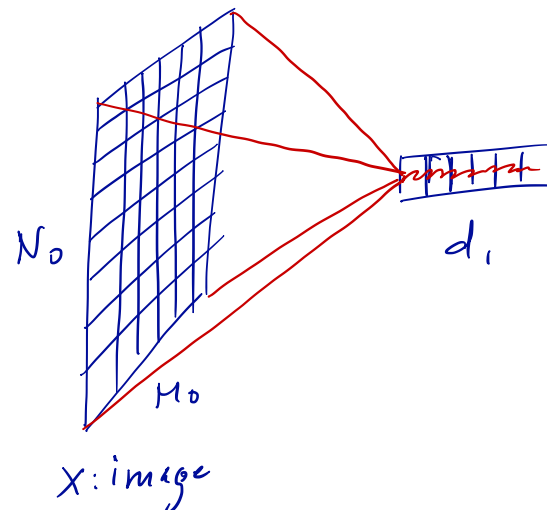


Different channels in the 2<sup>nd</sup> layer  
Weighted sums of 2<sup>nd</sup> order derivatives

# Translation invariance is native to fully connected networks



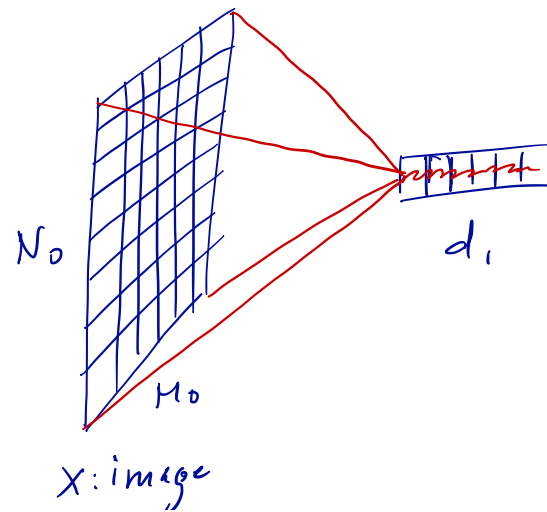
- These images will most likely lead to a very different activations in the hidden layer
- Rest of the network will see different activations
- In most vision applications, these images should lead to identical outputs



# Question

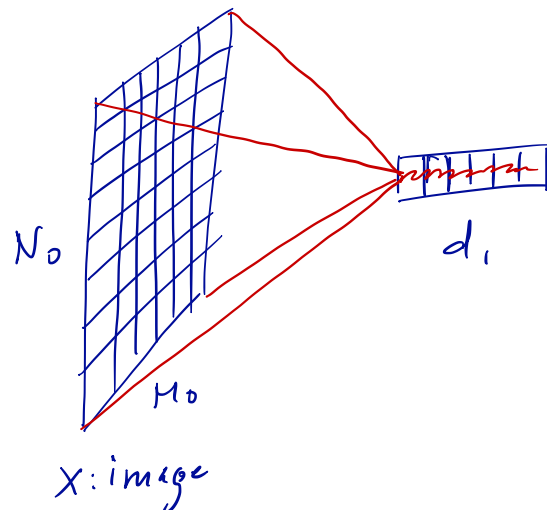


- What are some applications where these two images should lead to
  1. Identical outputs
  2. Different outputs



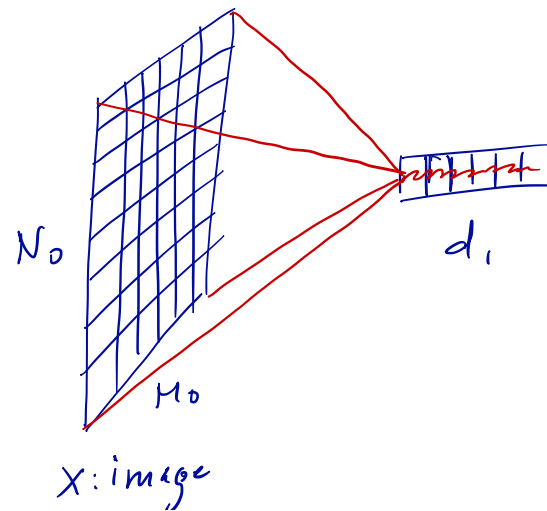


## Question



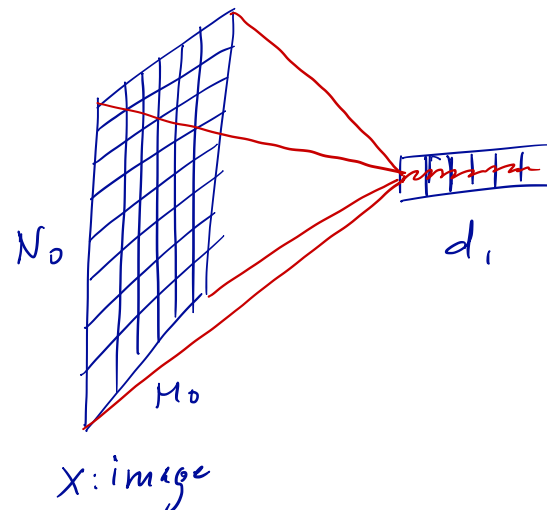
- What are some applications where these two images should lead to
  1. Identical outputs
    - Recognition
    - Detection
  2. Different outputs
    - Localization
    - Segmentation

# Translation invariance is native to fully connected networks



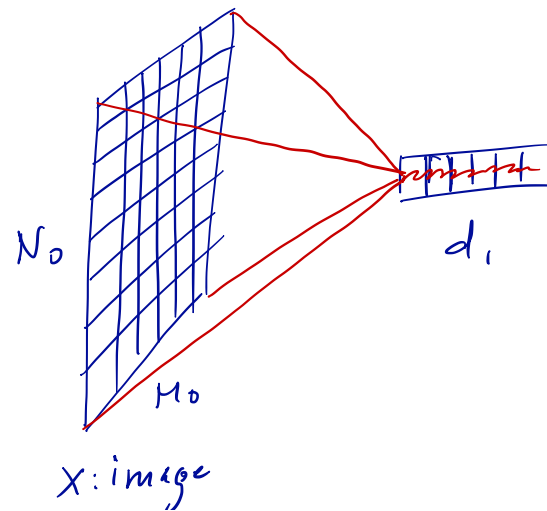
- These images will most likely lead to a very different activations in the hidden layer
- Rest of the network will see different activations
- In most vision applications, these images should lead to identical outputs
- It is possible to teach a fully connected network to be invariant to translations

## Question



- How can we teach a fully connected network to be invariant to translations?

# Translation invariance is native to fully connected networks



- These images will most likely lead to a very different activations in the hidden layer
- Rest of the network will see different activations
- In most vision applications, these images should lead to identical outputs
- It is possible to teach a fully connected network to be invariant to translations by having random translations of each image in your training set
- Not the most elegant way
- Convolution operation can help – it is translation equivariant

# Translation equivariance

- Equivariance: applying a transformation to the input yields the same result as applying the transformation to the output

$$f(T \circ x) = T \circ f(x)$$

- Convolution is linear shift invariant, so?

# Translation equivariance

- Equivariance: applying a transformation to the input yields the same result as applying the transformation to the output

$$f(T \circ x) = T \circ f(x)$$

- Convolution is linear shift invariant, it has translation equivariance



# Translation invariance

- It does not have translation invariance
- In what application do we need translation equivariance  
translation invariance



# Translation invariance

- It does not have translation invariance
- In what application do we need  
translation equivariance >> segmentation, localization  
translation invariance >> recognition





## Convolutional layers is a great idea but

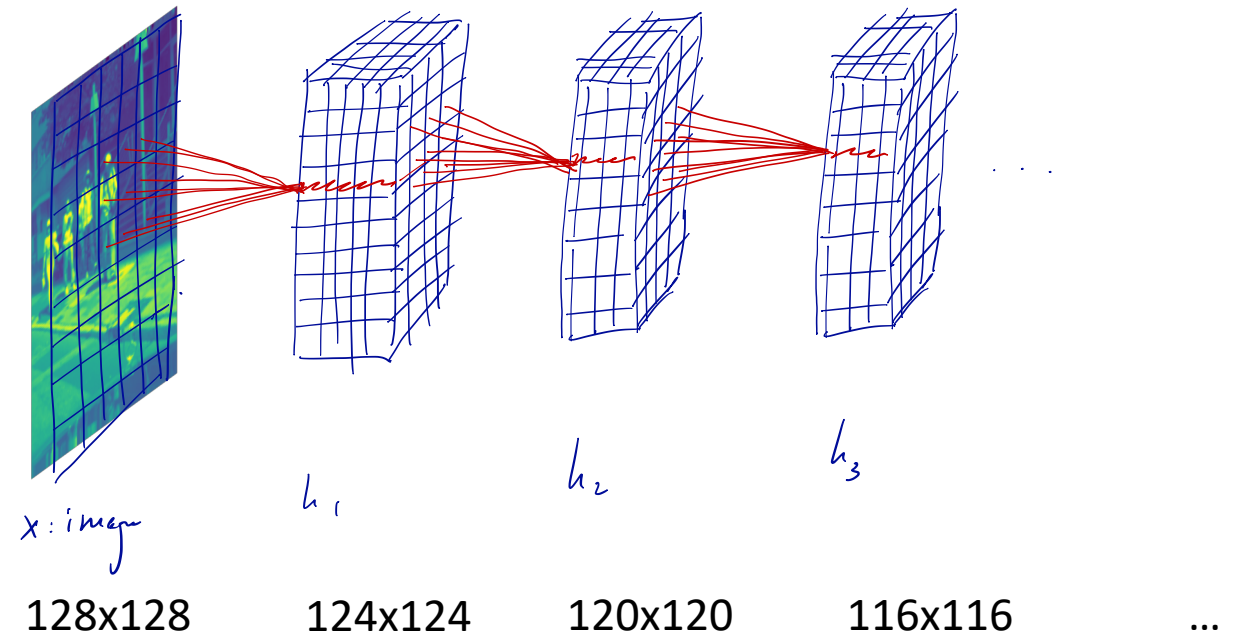
- Translation invariance would be very useful
- Dimensionality reduction requires many parameters

Assume we use convolution kernels of size 5x5 and valid padding in a recognition system >> output channel size should be 1x1 (with number channels equal to the number of classes)

You would need many layers.

Or very large kernels.

There might another way to do this...



## Strides

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- In a normal convolution you only move one pixel in each direction, not skipping any pixels

## Strides

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- In a normal convolution you only move one pixel in each direction, not skipping any pixels

## Strides

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- In a normal convolution you only move one pixel in each direction, not skipping any pixels
- However, you can decide to skip several pixels while shifting your kernel >> instead you can skip 2 pixels

## Strides

Channel size change with valid convolutions

$$M_l = \left\lfloor \frac{M_{l-1} - k_1}{s_1} \right\rfloor + 1 \quad N_l = \left\lfloor \frac{N_{l-1} - k_2}{s_2} \right\rfloor + 1$$

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- In a normal convolution you only move one pixel in each direction, not skipping any pixels
- However, you can decide to skip several pixels while shifting your kernel >> instead you can skip 2 pixels
- You take a stride of 3 instead of 1
- Reduction in channel size increases

## Strides

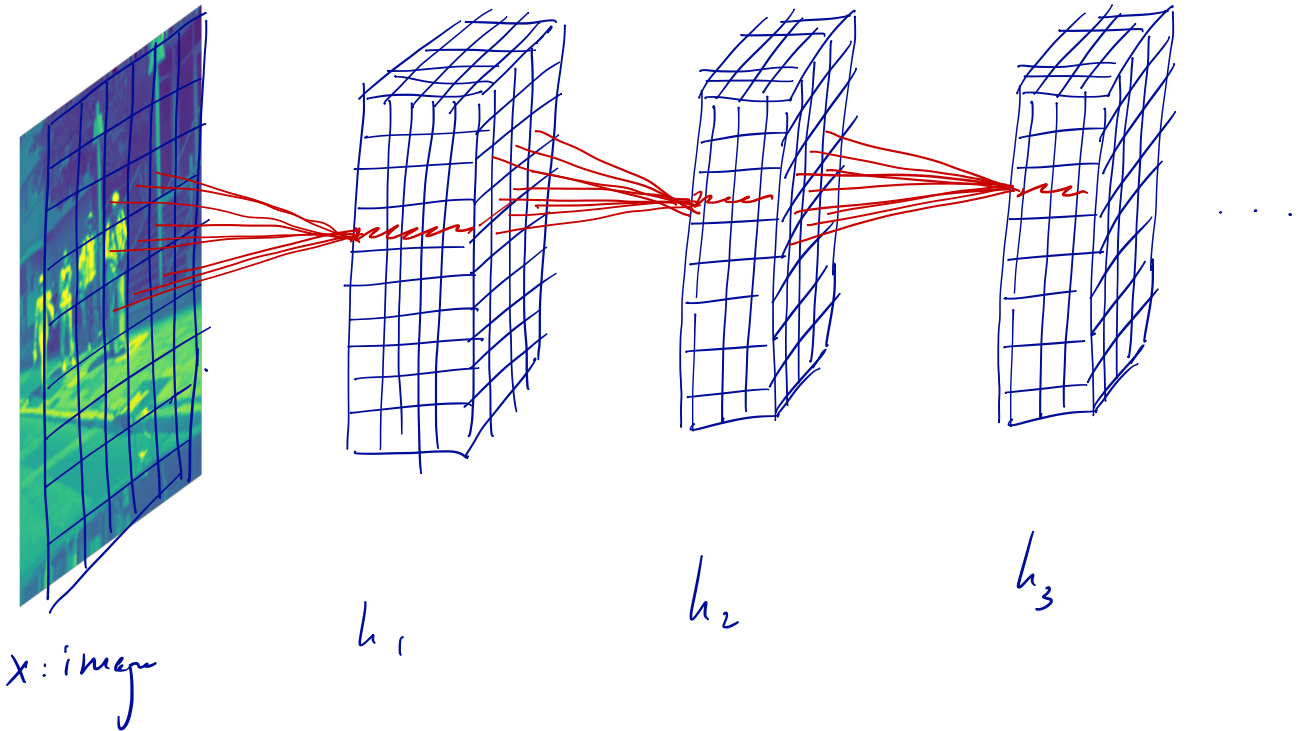
$$M_l = \left\lfloor \frac{M_{l-1} - k_1}{s_1} \right\rfloor + 1 \quad N_l = \left\lfloor \frac{N_{l-1} - k_2}{s_2} \right\rfloor + 1$$

Assume we use convolution kernels of size 5x5, valid padding **with stride 2** in a recognition system >> output channel size should be 1x1 (with number channels equal to the number of classes)

The dimension drops very quickly. The rate of drop will be faster if stride is increased.

You lose information. Higher stride means higher loss of information. You do not gain translation invariance.

If used, it is most common to use stride 2 in all directions.



128x128

124x124

120x120

116x116

...

62x62

29x29

12x12



Pooling layers





## Pooling

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- Pool information in a neighborhood
- Represents the region with one number >> summarizes information
- Applied to each channel separately
- **Max-pooling** – maximum of the activation values
- **Min-pooling** – minimum of the activation values
- Both are non-linear operations, like median filtering
- **Averaging pooling** – linear operator
- Max-pooling is the most commonly used version

## Max pooling

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- Represents the entire region with the neuron that achieves the highest activation
- Leads to partial local translation invariance
- 617 in the highlighted area can be in any of the neurons, the pooled value will not change
- Does not lead to complete translation invariance
- Often applied with strides equal to the size of the kernel


## Max pooling

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557

- Represents the entire region with the neuron that achieves the highest activation
- Leads to partial local translation invariance
- 617 in the highlighted area can be in any of the neurons, the pooled value will not change
- Does not lead to complete translation invariance
- Often applied with strides equal to the size of the kernel

## Dimensionality reduction

544	552	570	585	600	607	608	581	558	577
549	561	595	617	610	601	595	562	545	563
579	574	554	538	556	598	614	596	588	582
529	514	486	476	483	509	552	584	604	586
506	499	468	421	459	547	588	596	598	603
567	561	519	484	510	557	586	612	603	565
579	594	581	563	557	553	572	587	575	575
590	601	594	586	580	563	559	587	602	585
596	602	602	595	586	585	592	577	545	557
593	614	589	568	588	625	610	546	519	557



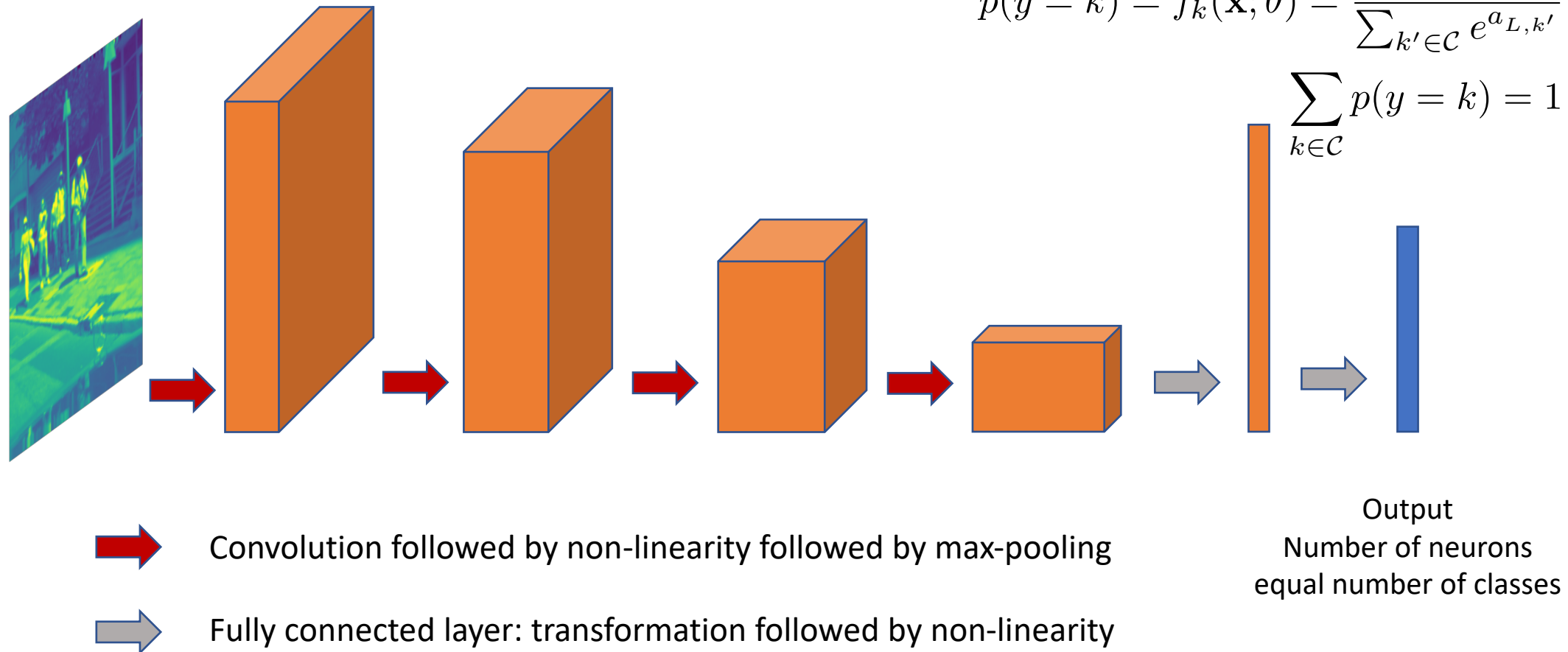
617	614
614	625

- Leads to a substantial dimensionality reduction
- Even when the pooling kernel is of size 2x2, it can halve the image!
- As the size of the pooling kernel increase, the reduction increases as well
- Non-linear dimensionality reduction
- Only the most prominent activation is transmitted to the next layer
- More advanced pooling mechanisms exist  
CapsuleNets [Sabour, Frosst and Hinton 2017]

# Recognition network with pooling

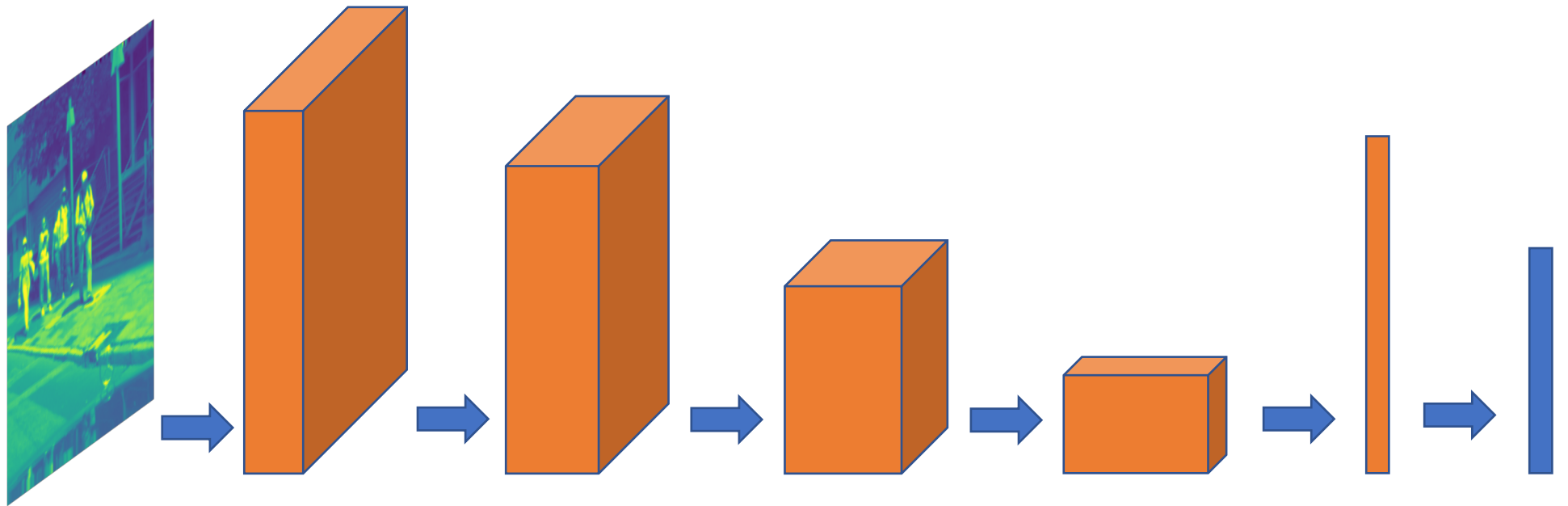


# Network architecture for a simple object recognition system

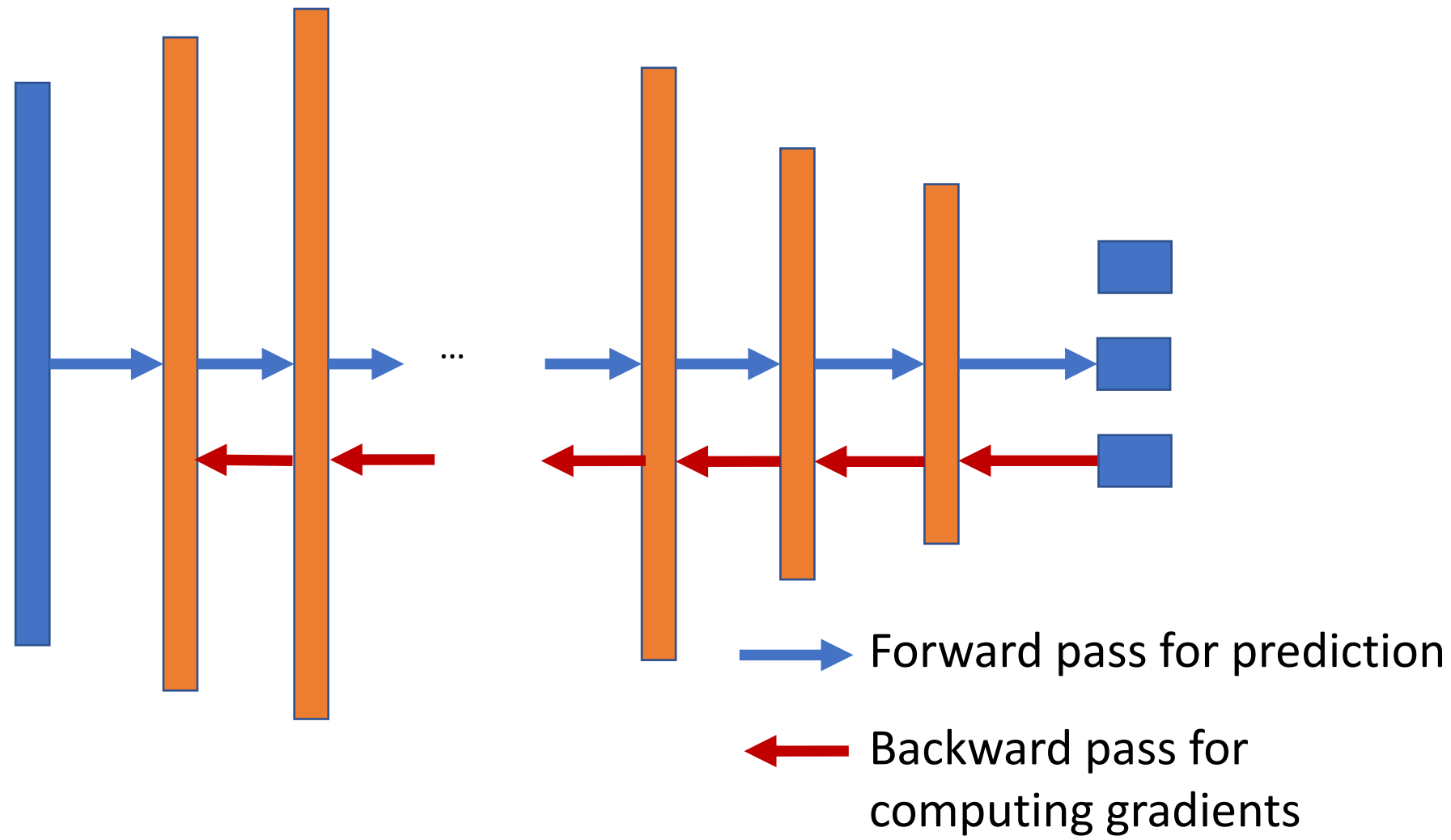


# Putting it all together: Basic Convolutional neural network (CNN)

$$a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K \quad p(y = k) = f_k(\mathbf{x}; \theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}} \quad \sum_{k \in \mathcal{C}} p(y = k) = 1$$



# Remember backpropagation





# Putting it all together: Classification Convolutional neural network (CNN)

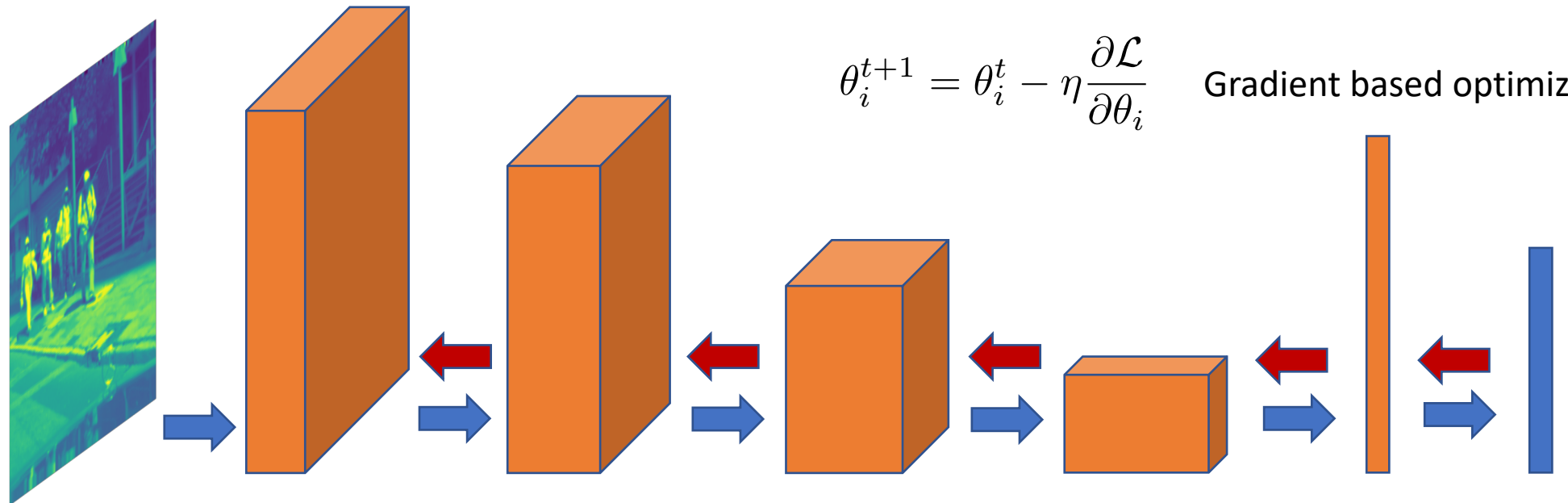
$$a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K \quad p(y = k) = f_k(\mathbf{x}; \theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}} \quad \sum_{k \in \mathcal{C}} p(y = k) = 1$$

Cost function  $\mathcal{L}(y_n, f(\mathbf{x}_n; \theta)) = - \sum_{k \in \mathcal{C}} \log(f_k(\mathbf{x}_n; \theta)) \mathbf{1}(y_n = k)$

Optimization  $\theta^* = \arg_{\theta} \min \sum_{n=1}^N \mathcal{L}(\mathbf{y}_n, f(\mathbf{x}_n; \theta))$

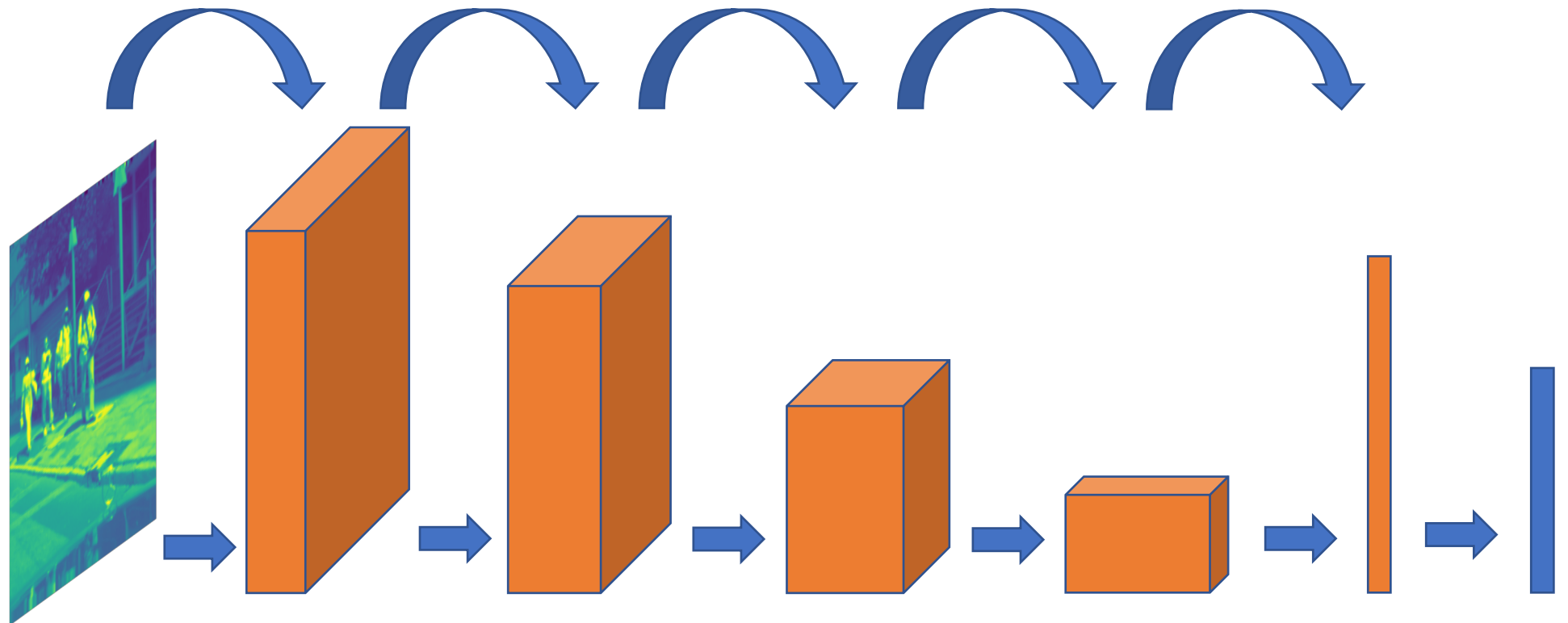
$$\theta_i^{t+1} = \theta_i^t - \eta \frac{\partial \mathcal{L}}{\partial \theta_i}$$

Gradient based optimization



# Progressively aggregating spatial information to reach a global decision

Local features are extracted and aggregated throughout the network  
The last layers “sees” the entire image and the features encode global information



# Essential blocks lead to powerful algorithms

- Convolutional layers and pooling are the essential blocks
- They have been used to create complicated networks
- First one

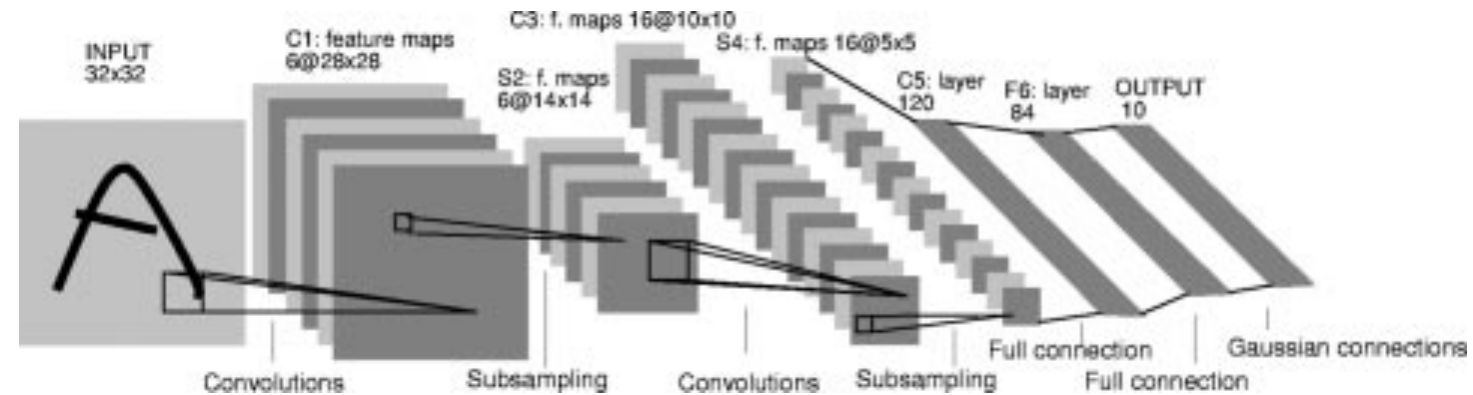


Fig. 2. Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.

[Lecun, Bottou, Bengio and Haffner; Gradient-based learning applied to document recognition; 1998]

- Then silence for a long time

## Why silence

- Models had too many parameters
- They overfit for small datasets
- We did not have very large datasets
- Even for large sets, we did not have enough computation power to train the models until...



General purpose Graphical Processing Units (GPUs)  
Allowed parallel processing

## Then first this happened in 2006

### A Fast Learning Algorithm for Deep Belief Nets

**Geoffrey E. Hinton**

*hinton@cs.toronto.edu*

**Simon Osindero**

*osindero@cs.toronto.edu*

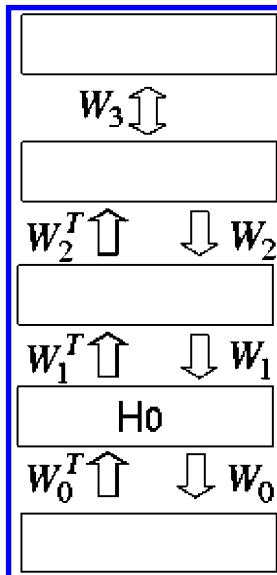
*Department of Computer Science, University of Toronto, Toronto, Canada M5S 3G4*

**Yee-Whye Teh**

*tehyw@comp.nus.edu.sg*

*Department of Computer Science, National University of Singapore,  
Singapore 117543*

- A fast algorithm to train deep belief networks by stacking restricted Boltzmann machines
- Layer-wise pre-training with contrastive divergence
- Set a state-of-the-art performance on MNIST
- However, this did not use GPUs yet



## Then in 2012

**ImageNet Classification with Deep Convolutional Neural Networks**

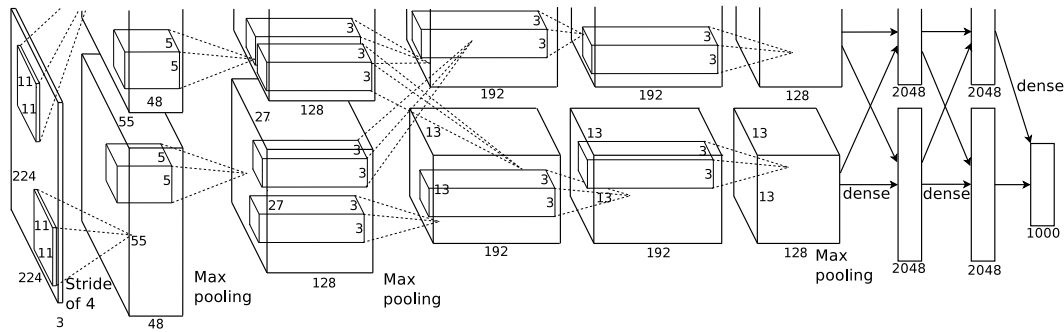
---

**Alex Krizhevsky**  
University of Toronto  
kriz@cs.utoronto.ca

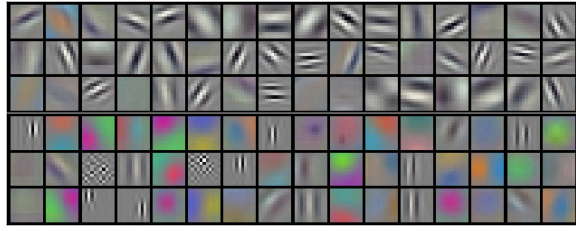
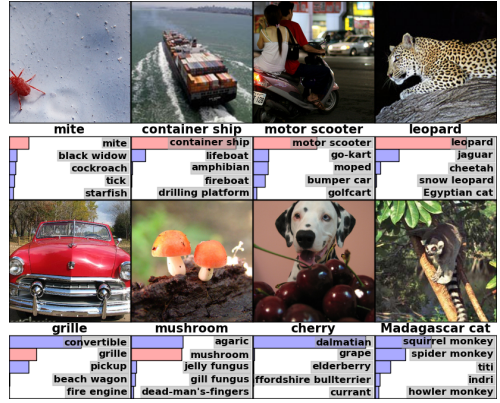
**Ilya Sutskever**  
University of Toronto  
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**  
University of Toronto  
hinton@cs.utoronto.ca

- Krizhevsky et al. almost halved the error rate in the ImageNet challenge
- A simple CNN



Network



First layer filters 11x11

## A word about the ImageNet challenge

### **ImageNet: A Large-Scale Hierarchical Image Database**

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei  
Dept. of Computer Science, Princeton University, USA  
{jiadeng, wdong, rsocher, jial, li, feifeili}@cs.princeton.edu

[CVPR 2009]

[International Journal of Computer Vision](#)

December 2015, Volume 115, [Issue 3](#), pp 211–252 | [Cite as](#)

### ImageNet Large Scale Visual Recognition Challenge

Authors

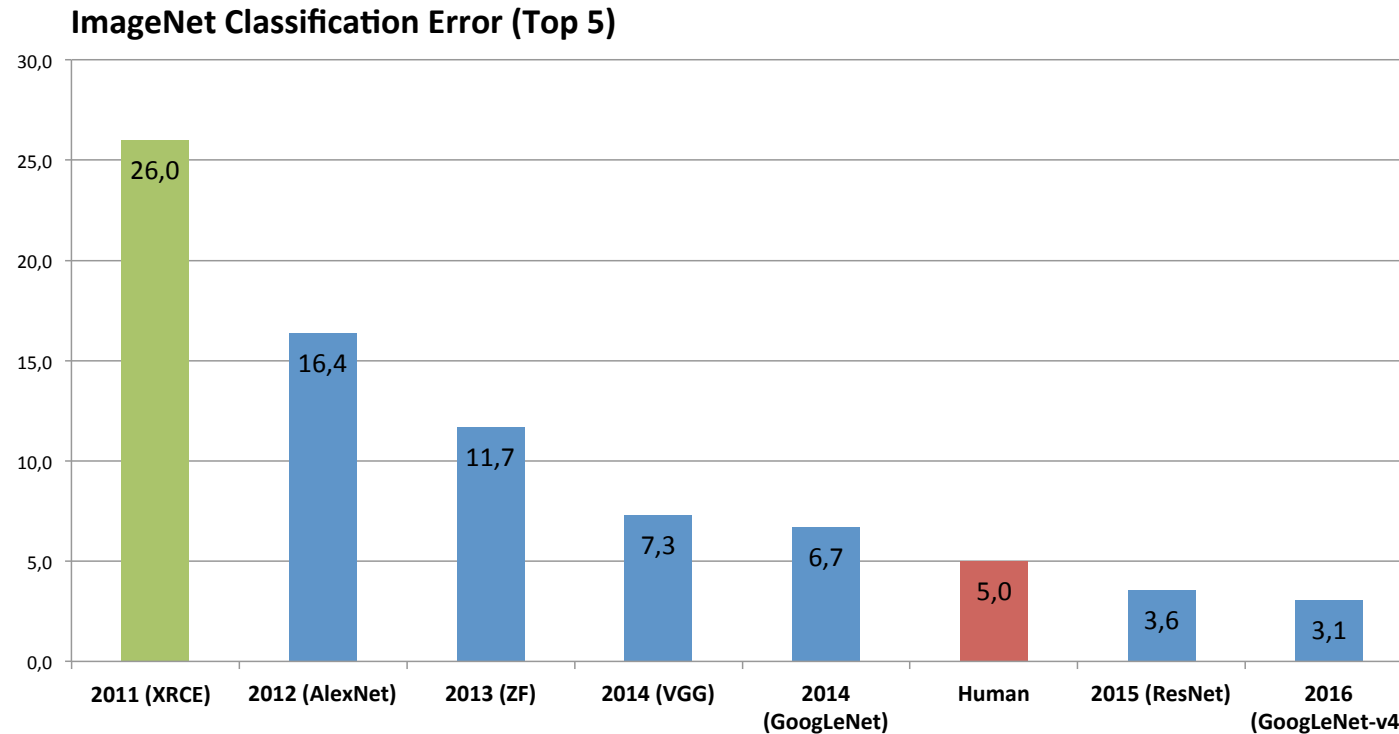
[Authors and affiliations](#)

Olga Russakovsky , Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy,

Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei

- Large scale object recognition challenge started in 2010
- 1.2 million training images
- 1000 object categories
- 200k validation and test images
- 2017 – 3 challenges:
  - Object localization
  - Object detection
  - Object detection from video

# Historical evolution of the ImageNet Challenge



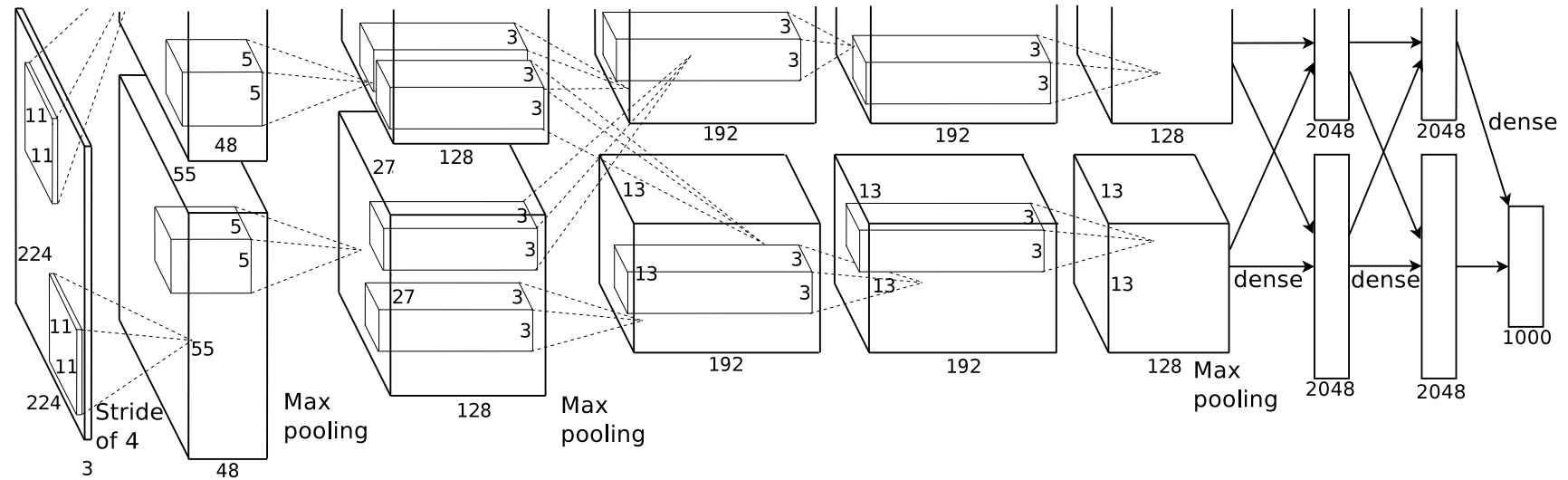
ADVANCED SEMINAR SS 2017: SURVEY OF NEURAL NETWORKS IN AUTONOMOUS DRIVING

Survey of neural networks in autonomous driving

Gustav von Zitzewitz



## How the networks evolved - 2012



GPU implementation of the basic CNN using two GPUs and communication between the units  
Varying size of convolutional filters in the different layers

### ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky  
University of Toronto  
kriz@cs.utoronto.ca

Ilya Sutskever  
University of Toronto  
ilya@cs.utoronto.ca

Geoffrey E. Hinton  
University of Toronto  
hinton@cs.utoronto.ca

## VGGNet- 2014

- VGG network
- [Table from the publication]
- First **deep** network
- 3x3 convolutional filters across the network
- Still the same principles as 1998

VERY DEEP CONVOLUTIONAL NETWORKS  
FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan\* & Andrew Zisserman\*

Visual Geometry Group, Department of Engineering Science, University of Oxford  
{karen,az}@robots.ox.ac.uk

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.

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 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 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 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

## Inception - 2014

- Combining different kernels – a new idea
- [Figures from the publication]
- Allowed VERY deep networks
- GoogLeNet

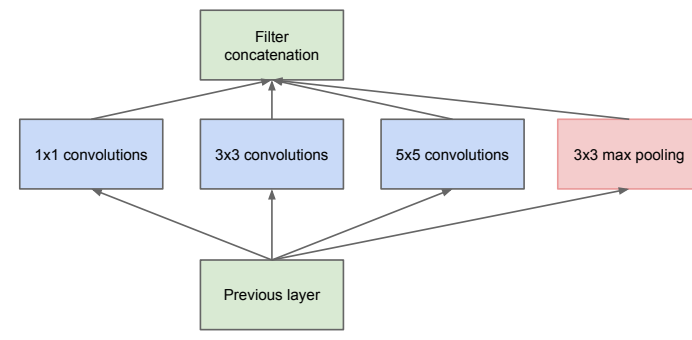
### Going Deeper with Convolutions

Christian Szegedy<sup>1</sup>, Wei Liu<sup>2</sup>, Yangqing Jia<sup>1</sup>, Pierre Sermanet<sup>1</sup>, Scott Reed<sup>3</sup>,  
 Dragomir Anguelov<sup>1</sup>, Dumitru Erhan<sup>1</sup>, Vincent Vanhoucke<sup>1</sup>, Andrew Rabinovich<sup>4</sup>

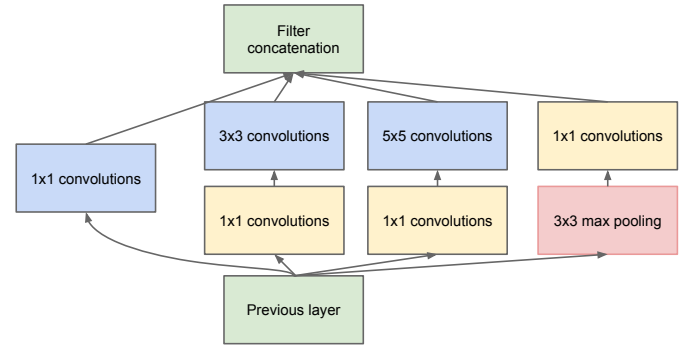
<sup>1</sup>Google Inc. <sup>2</sup>University of North Carolina, Chapel Hill

<sup>3</sup>University of Michigan, Ann Arbor <sup>4</sup>Magic Leap Inc.

[CVPR 2015]



(a) Inception module, naive version



(b) Inception module with dimensionality reduction

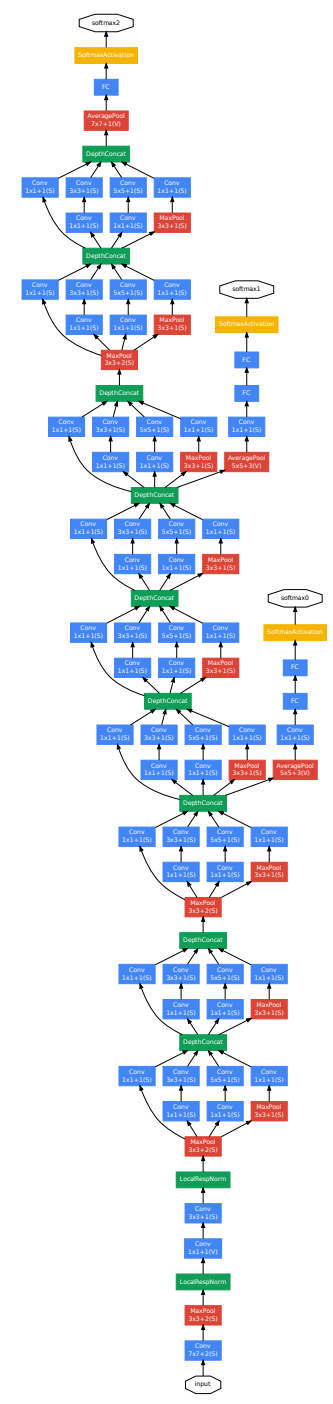


Figure 2: Inception module

## ResNet - 2015

- Residual modeling – a new idea
- [Figures from the publication]
- Allowed VERY deep networks

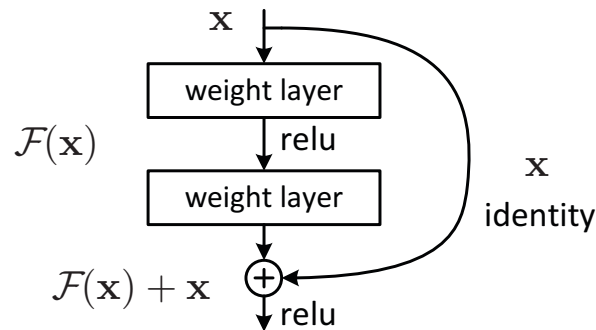
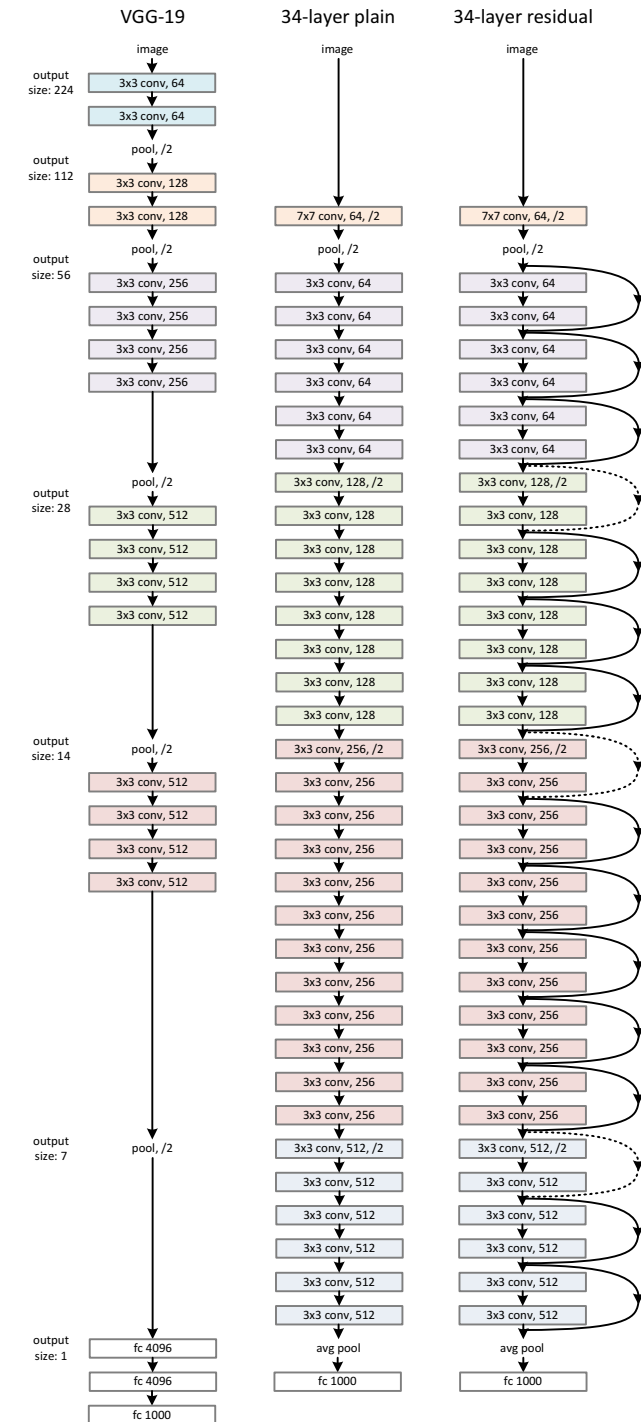


Figure 2. Residual learning: a building block.

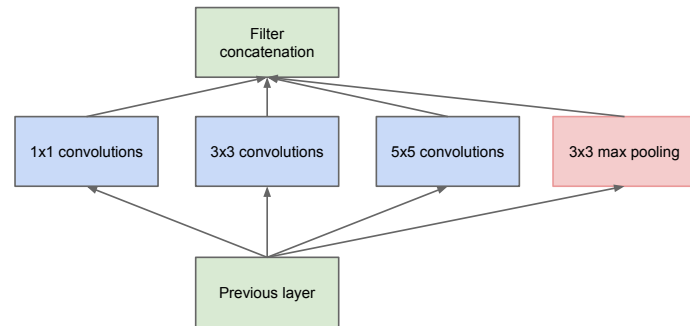
### Deep Residual Learning for Image Recognition

Kaiming He    Xiangyu Zhang    Shaoqing Ren    Jian Sun  
 Microsoft Research  
 {kahe, v-xiangz, v-shren, jiansun}@microsoft.com

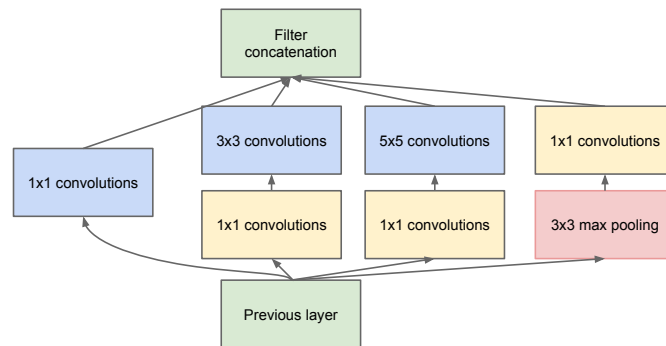
[CVPR 2016]



## More on the inception module



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2: Inception module

### Going Deeper with Convolutions

$$a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$$

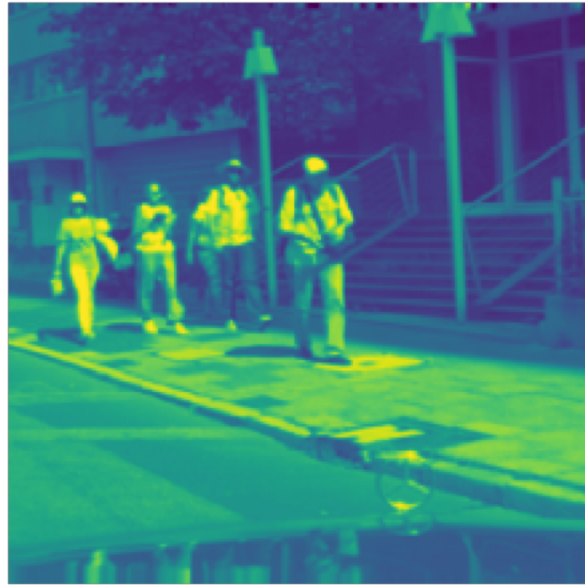
- Conventional way was to combine multiple convolutional filters of same size.
- Inception network combines multiple convolutional filters of different sizes.
- 1x1 convolutions do not aggregate information over space but only over different channels

Christian Szegedy<sup>1</sup>, Wei Liu<sup>2</sup>, Yangqing Jia<sup>1</sup>, Pierre Sermanet<sup>1</sup>, Scott Reed<sup>3</sup>,  
Dragomir Anguelov<sup>1</sup>, Dumitru Erhan<sup>1</sup>, Vincent Vanhoucke<sup>1</sup>, Andrew Rabinovich<sup>4</sup>

<sup>1</sup>Google Inc. <sup>2</sup>University of North Carolina, Chapel Hill

<sup>3</sup>University of Michigan, Ann Arbor <sup>4</sup>Magic Leap Inc.

# Computer Vision



Image



3x3

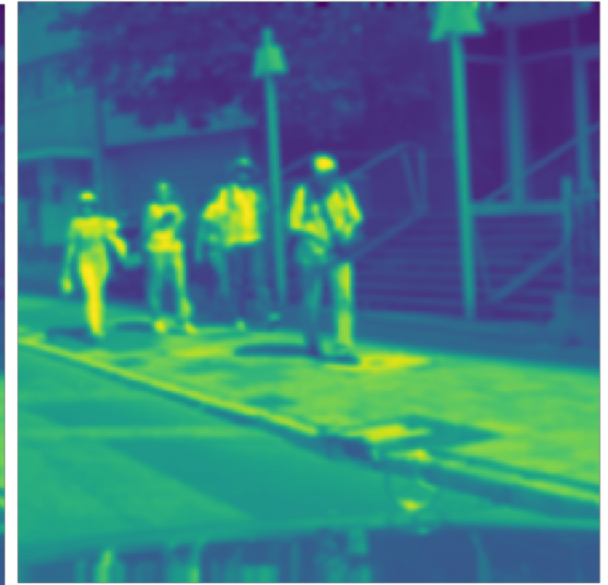


5x5

Inception module aggregates all this information in one layer.  
It learns to use the necessary components from each filter size



7x7



9x9

## More on Residual units and ResNet

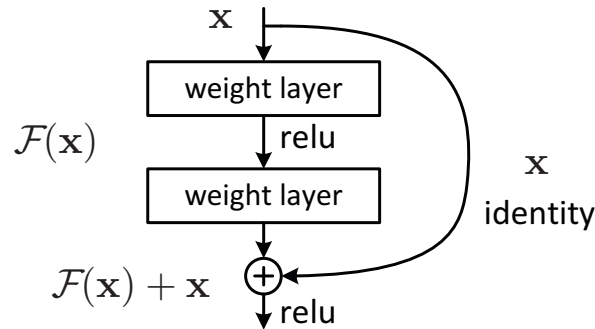


Figure 2. Residual learning: a building block.

- Conventional layer

$$h_l = f_l(h_{l-1})$$

difficult to model an identity function

- Residual layer

$$h_l = f_l(h_{l-1}) + h_{l-1}$$

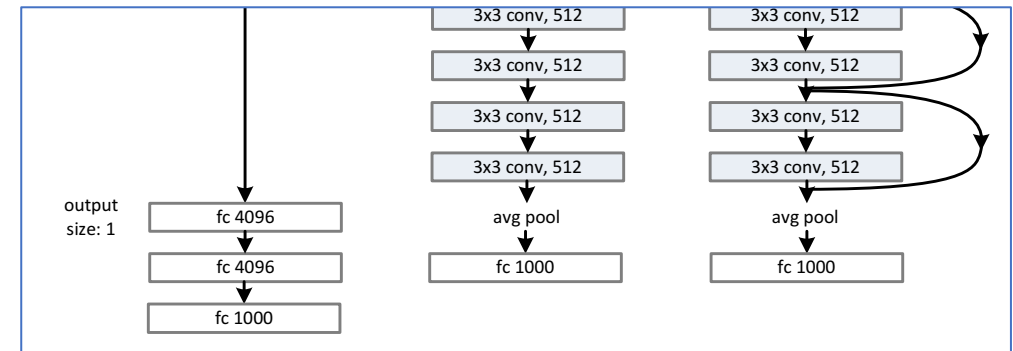
only modeling the residual allows identity maps

### Deep Residual Learning for Image Recognition

Kaiming He    Xiangyu Zhang    Shaoqing Ren    Jian Sun

Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com



## So far only image-wide classification

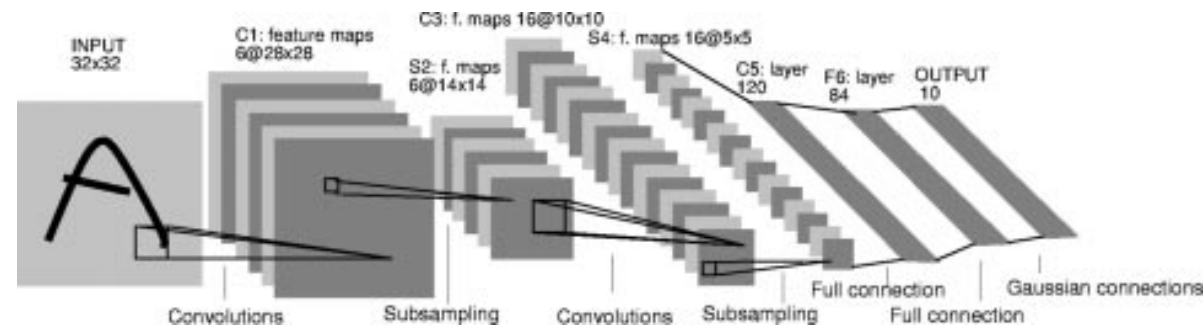
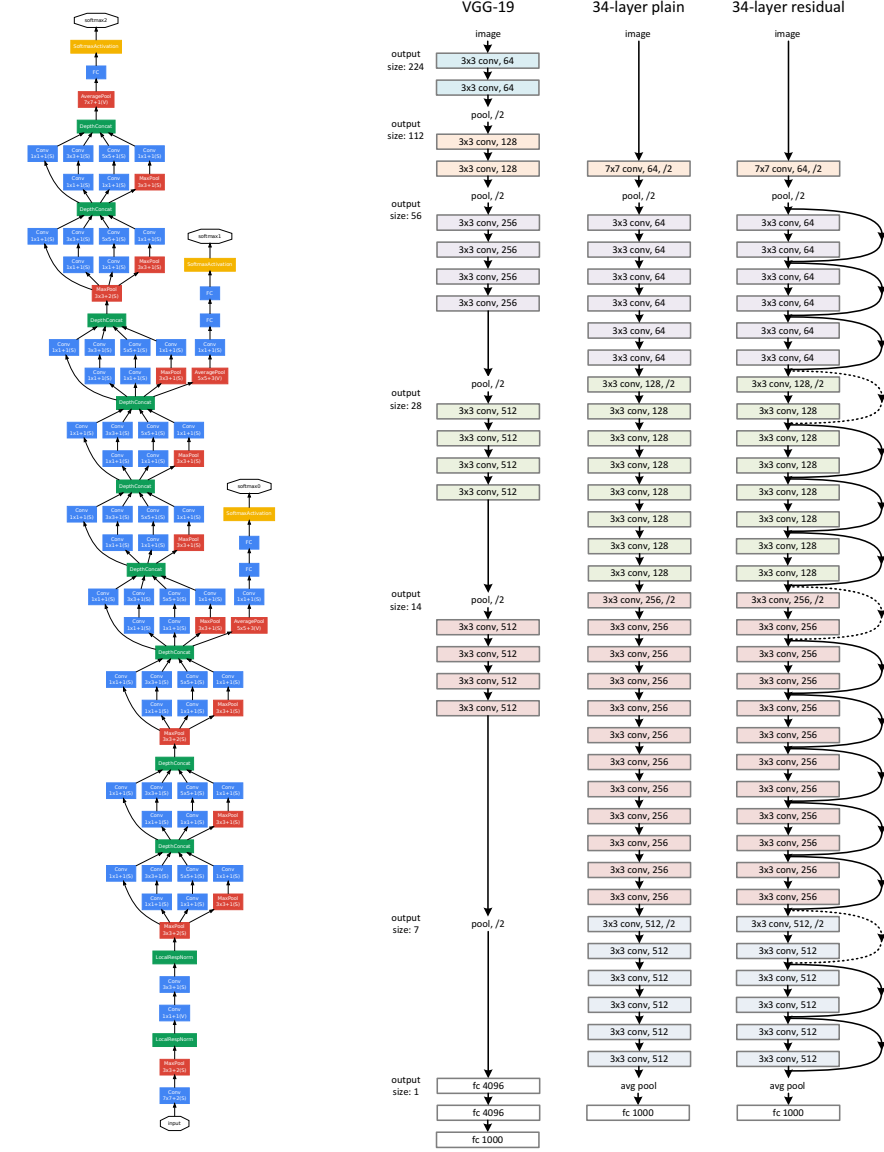
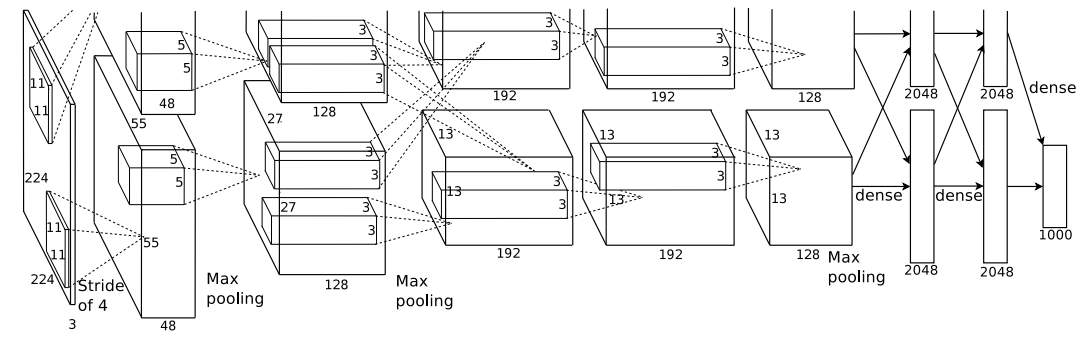


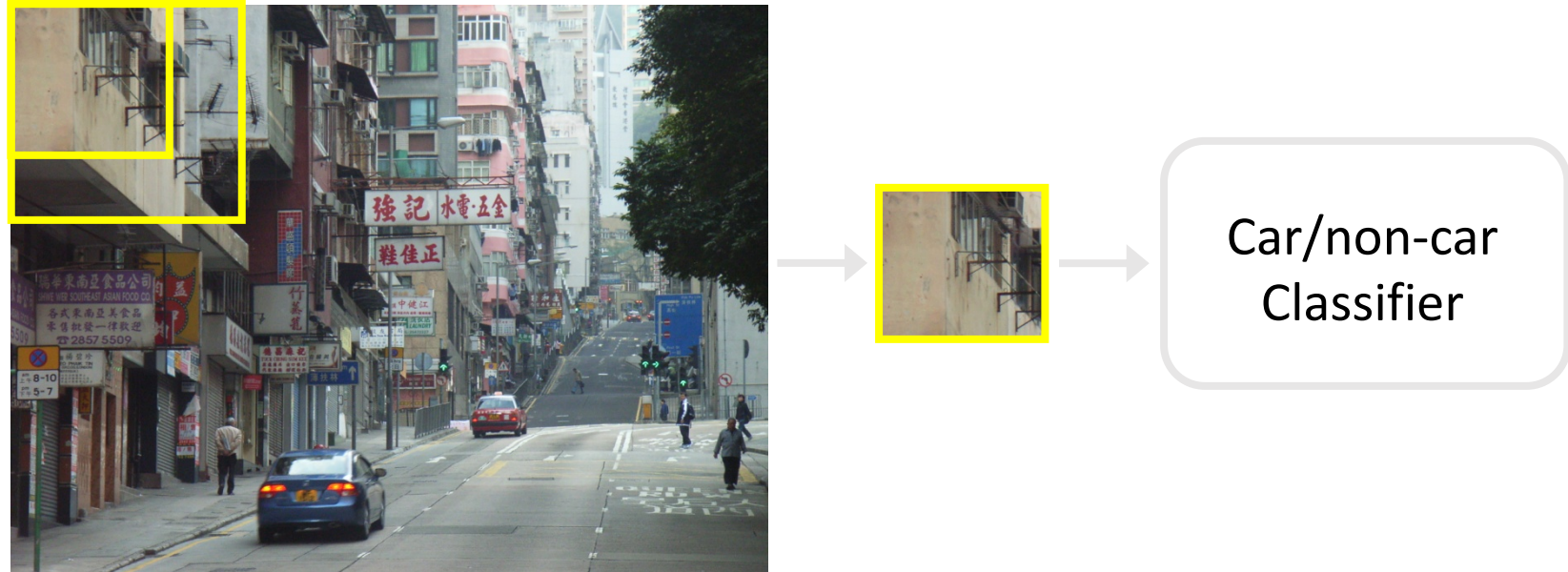
Fig. 2. Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.





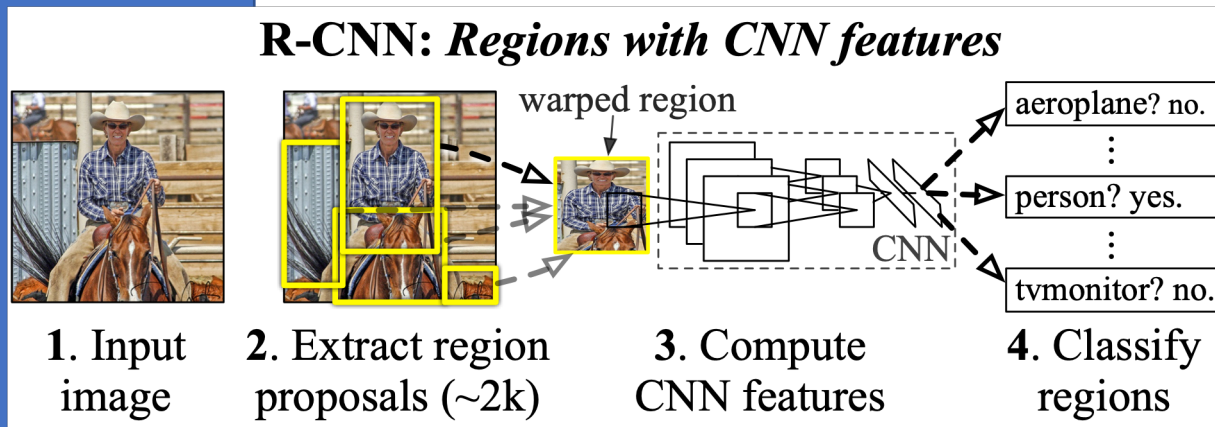
## Detection via classification: recall

- If object may be in a cluttered scene, slide a window around looking for it.



- a brute-force approach with many local decisions.

## Regions with CNN features



- [Images adapted from the article]
- Very similar idea as sliding windows
- However, instead of sliding windows, they use **region proposal** schemes to make the algorithm very fast.
- Three modules:
  1. Region proposals by **selective search** [Uijlings et al. 2012] and warp regions to a fixed size
  2. CNNs to extract features
  3. SVM to classify

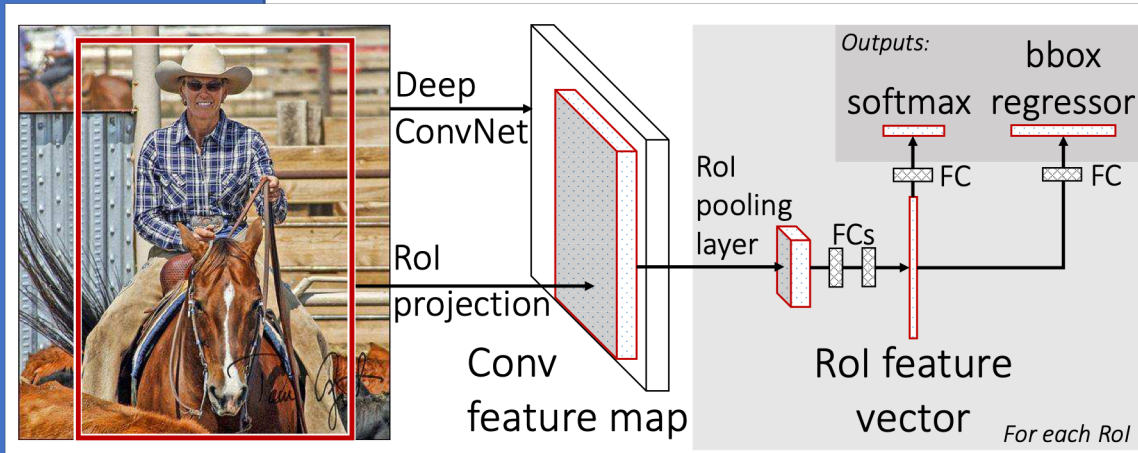
Multi-stage method and can be slow

Ross Girshick<sup>1</sup> Jeff Donahue<sup>1,2</sup> Trevor Darrell<sup>1,2</sup> Jitendra Malik<sup>1</sup>

<sup>1</sup>UC Berkeley and <sup>2</sup>ICSI

{rbg, jdonahue, trevor, malik}@eecs.berkeley.edu

## Fast Region-based CNN – Fast R-CNN

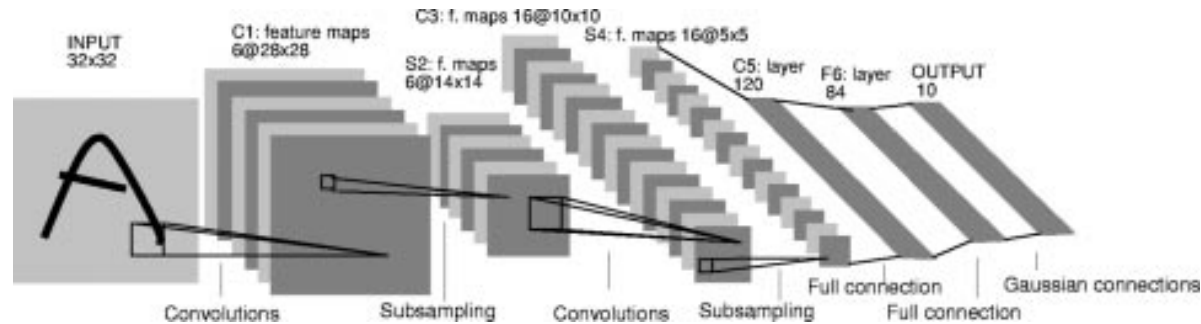


### Fast R-CNN

Ross Girshick  
Microsoft Research  
rbg@microsoft.com

- [Images adapted from publication]
- Unifying the feature extraction and classification into one step
  1. Region proposals are constructed
  2. A CNN takes input the image and region proposals and classifies each region.
- Roi pooling layer makes it possible
- Unified multi-class training
- Faster at test time

## How do we extend this to segmentation?



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



- Classification CNN progressively reduces the size of the network
- Segmentation needs to keep the size of the image constant
- Pixel-wise classification
- Commonly used training loss is pixel-wise cross entropy

## Two competing factors

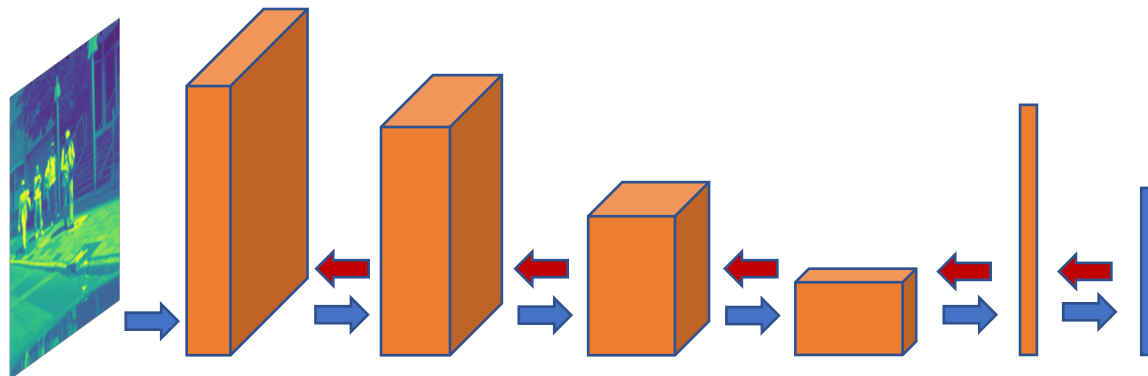
### **Integrating larger context**

- Decision for each pixel should look at a large portion of the image to decide the semantic label.
- But this means information for neighboring pixels are very similar – difficult to disambiguate boundaries.

### **Distinguishing neighboring boundary pixels**

- Network should be able to distinguish neighboring pixels – focusing on fine level details.
- Only focusing on fine level details cannot integrate the larger image context and have trouble determining the semantic labels.

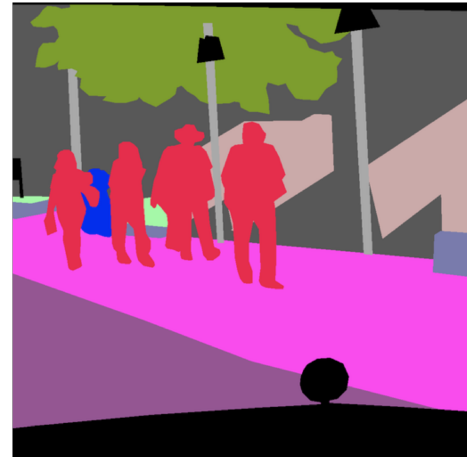
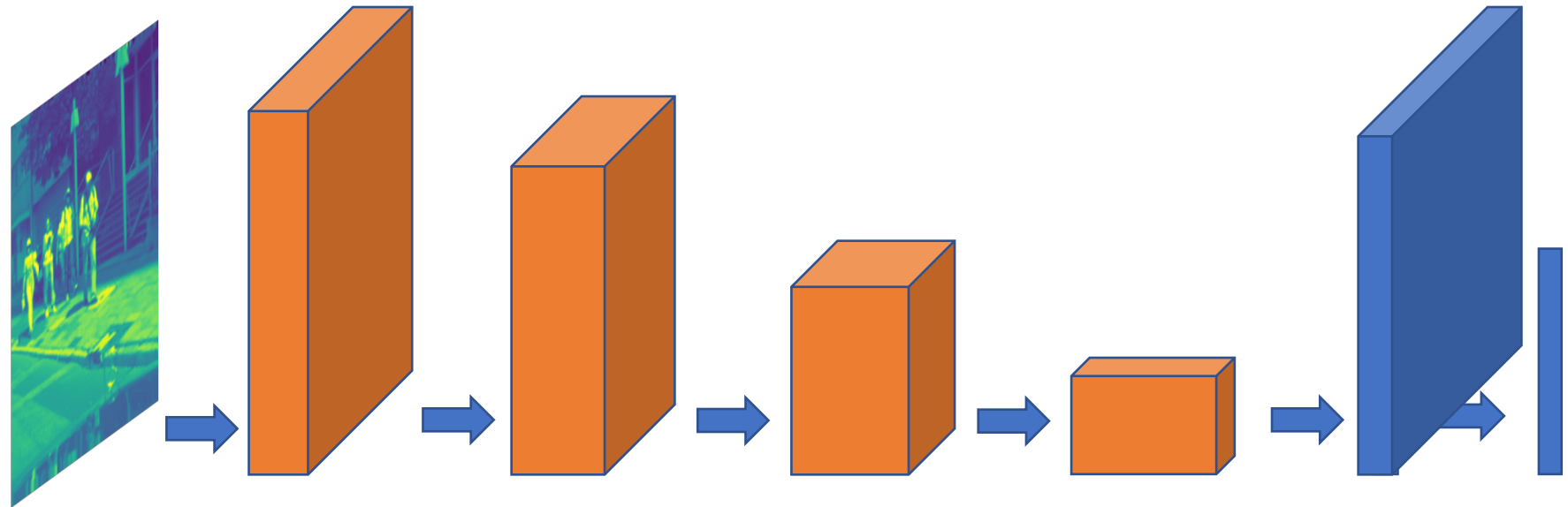
## Naïve approach



- Classification network for each pixel
- Sliding window approach
- Each window only predicts the central pixel's class
- Conceptually easy
- Straightforward training
- Can be slow in test time
- Good in context not very good in boundaries

# Fully convolutional network

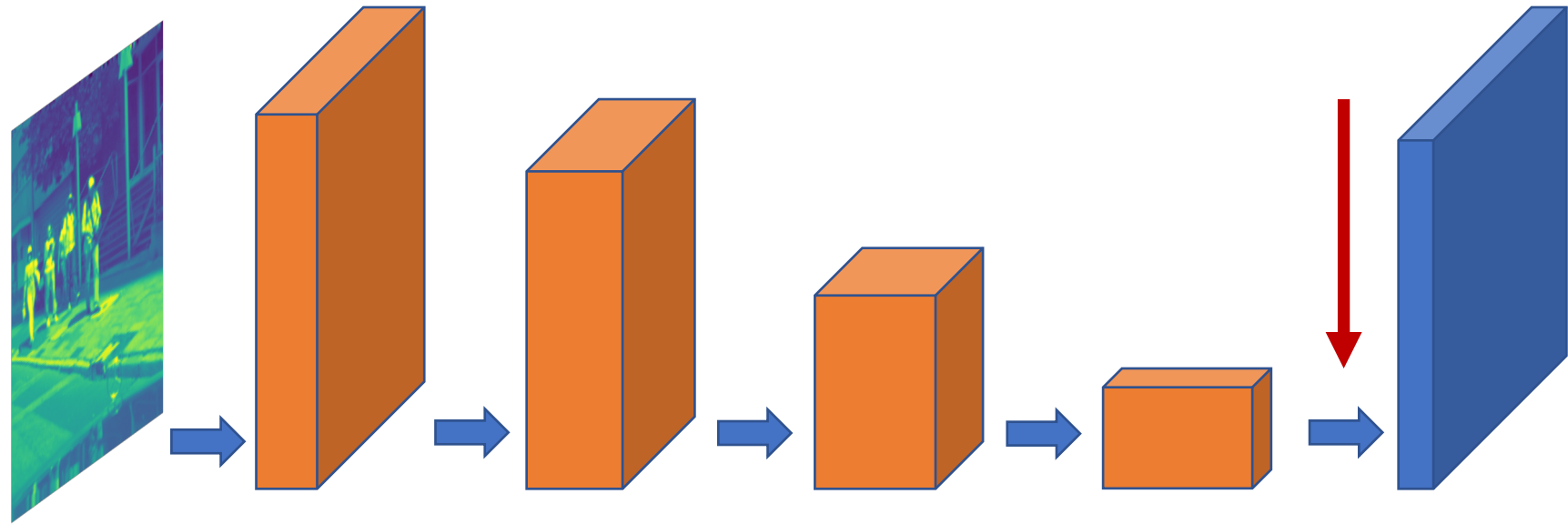
Depth equals the number of possible semantic classes



Pixel-wise Softmax

## Fully Convolutional Networks for Semantic Segmentation

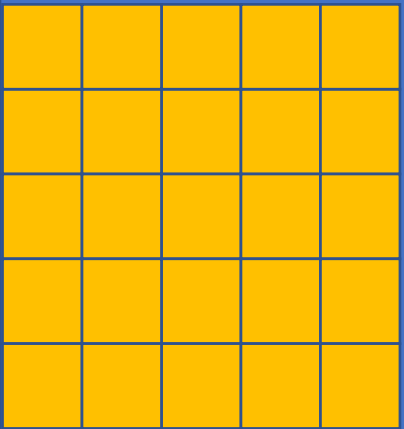
# How did that happen?



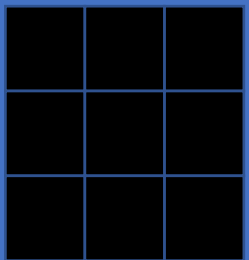
- The penultimate layer is deep but has a very small channel size
- The last, output, layer has a very large channel size – as large as the input image
- There are two basic ways of doing this: transposed convolutions and upsampling



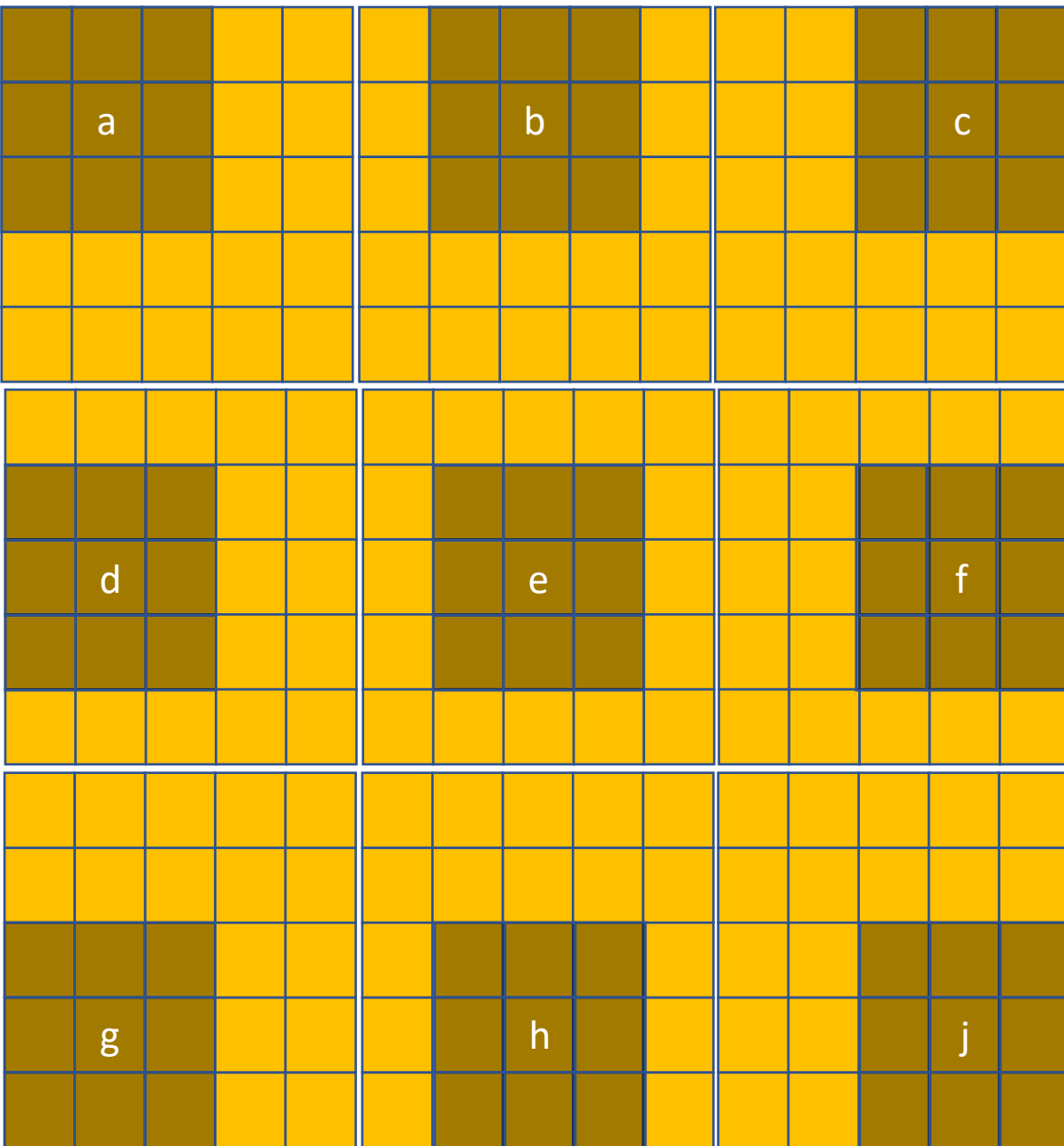
# Computer Vision



Image



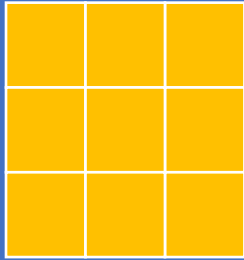
kernel



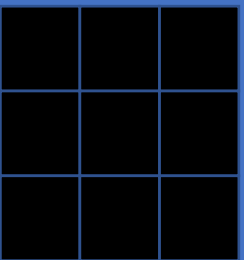
# Convolution

a	b	c
d	e	f
g	h	j

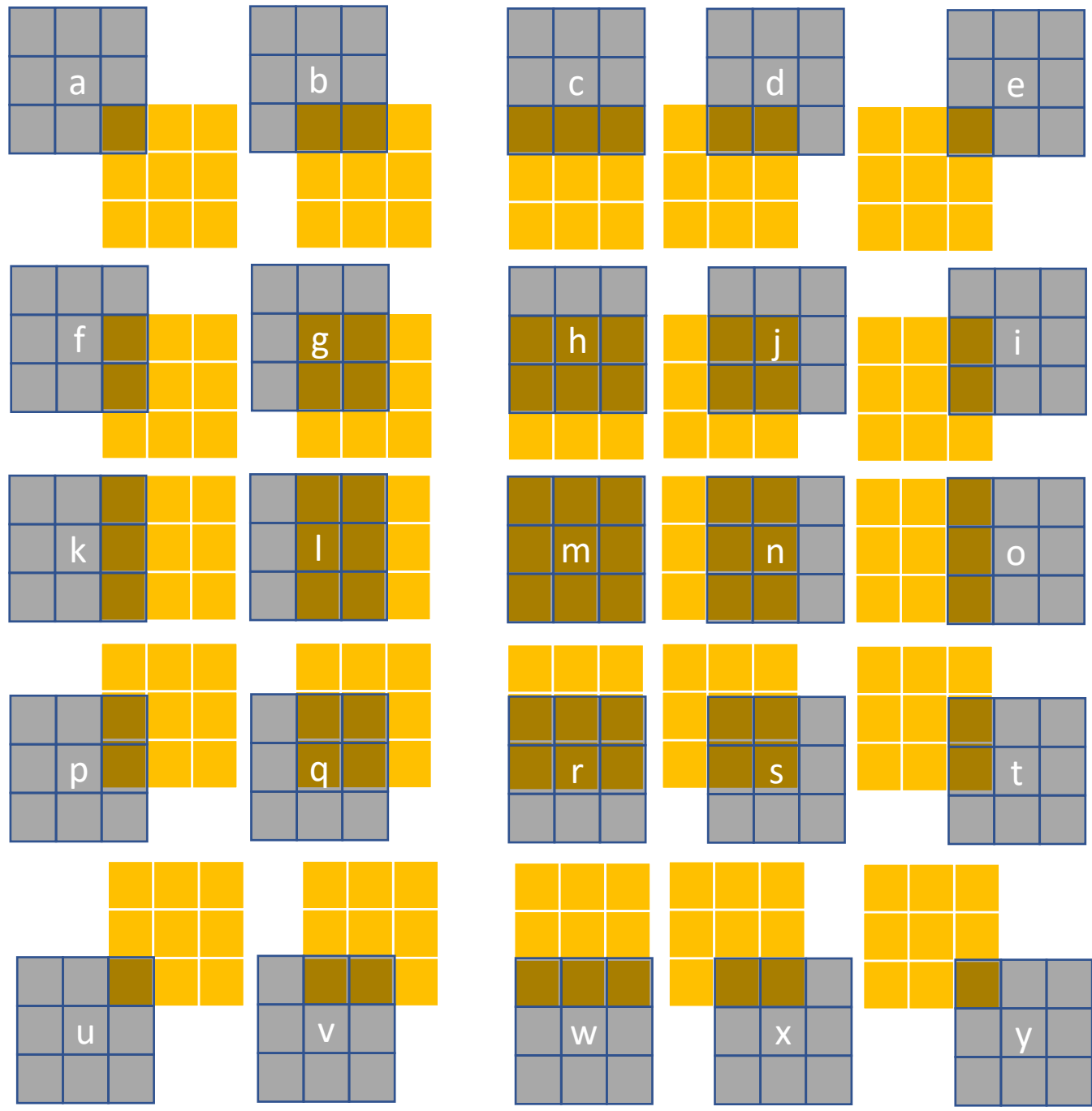
# Computer Vision



Image



kernel

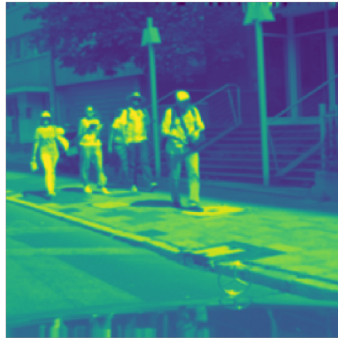


# Transposed Convolution

Sometimes referred to as deconvolution but that is not correct!

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y

# Up-sampling followed by convolution



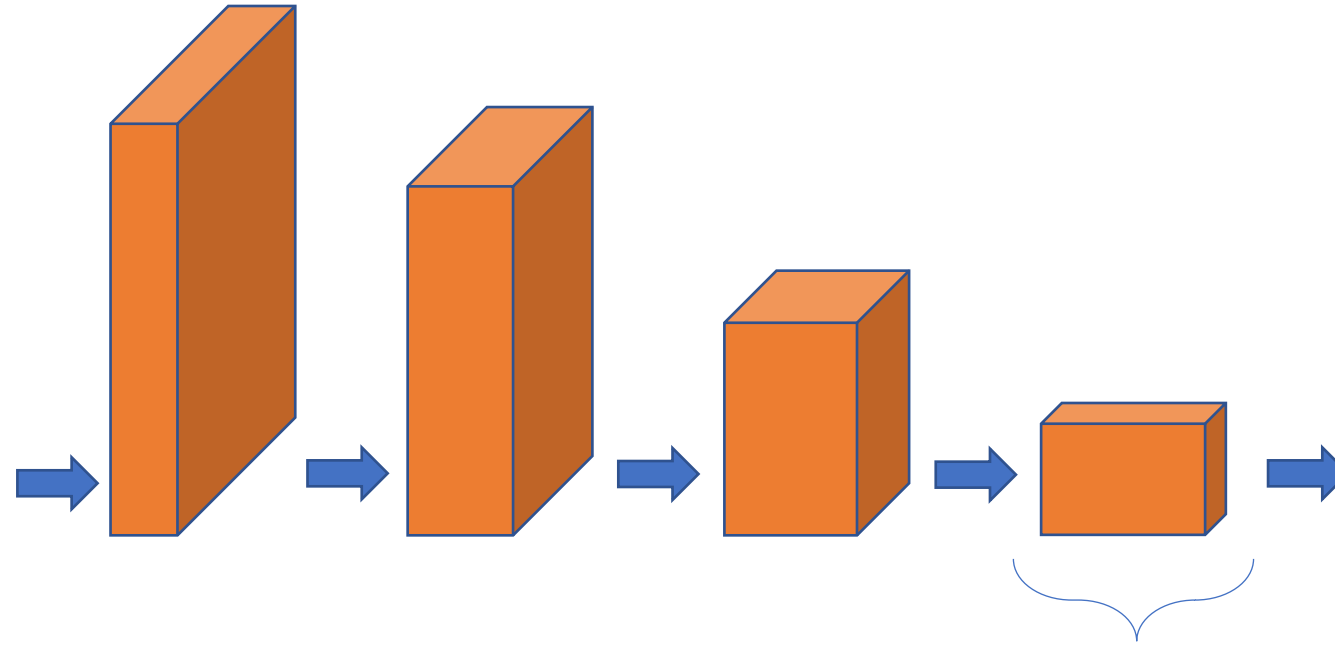
Upsampling  
e.g. bilinear



Convolution filter

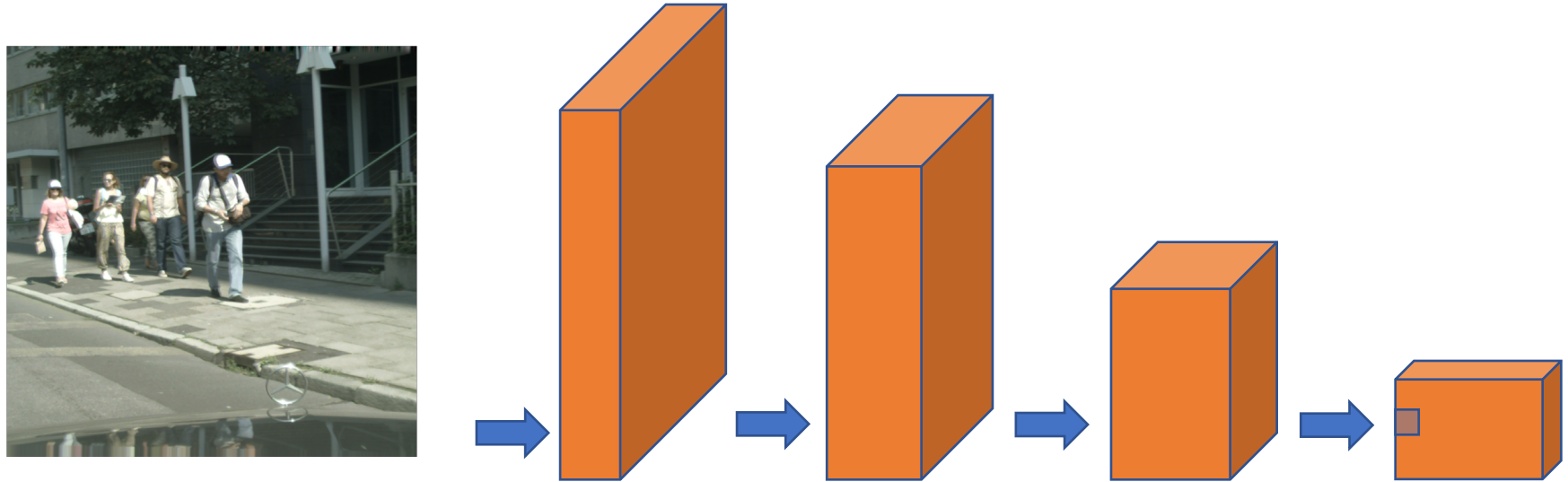


## Receptive fields



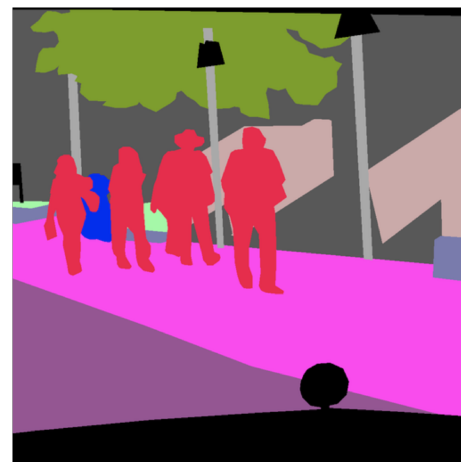
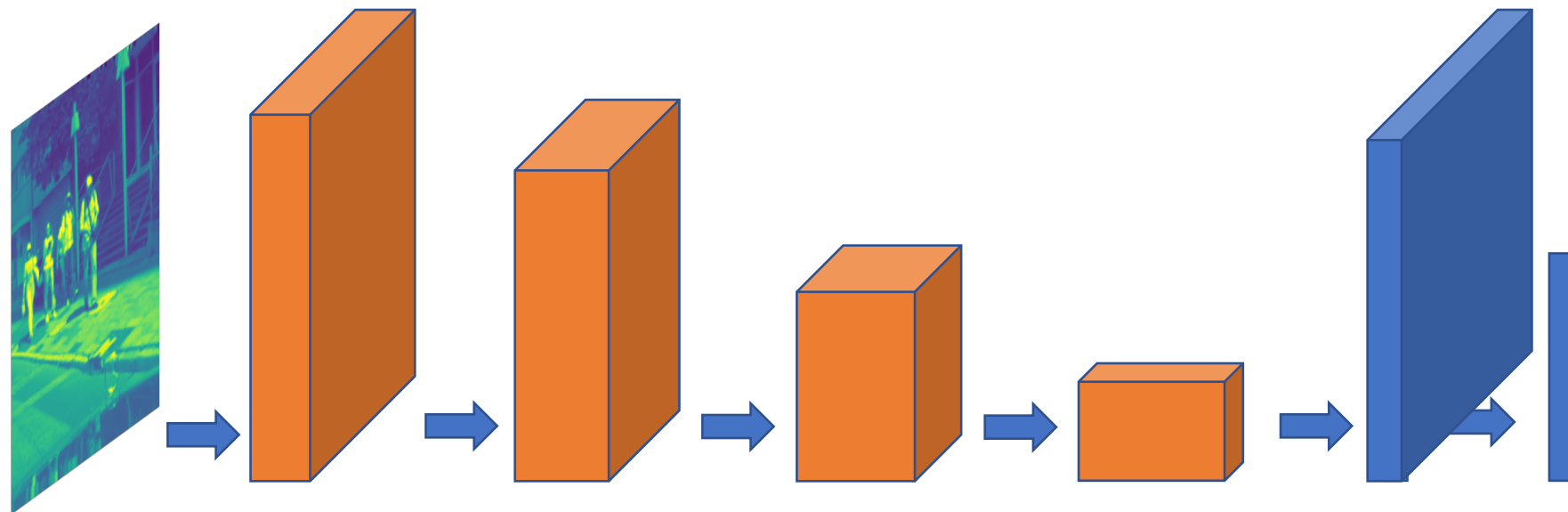
- This channel must see large areas of the image so we can determine semantic labels for the pixels.
- Due to the preceding convolutions and pooling each neuron in this channel sees a large area of the image – it has a **receptive field**

## Receptive fields



- Receptive field grows progressively as we go backwards through the network
- Normally, to grow the receptive field the options are:
  - Convolution, pooling, strides
  - Pooling and strides reduce the size of the channels by a large margin
  - Leads to resolution and coverage loss

# Solves the context integration

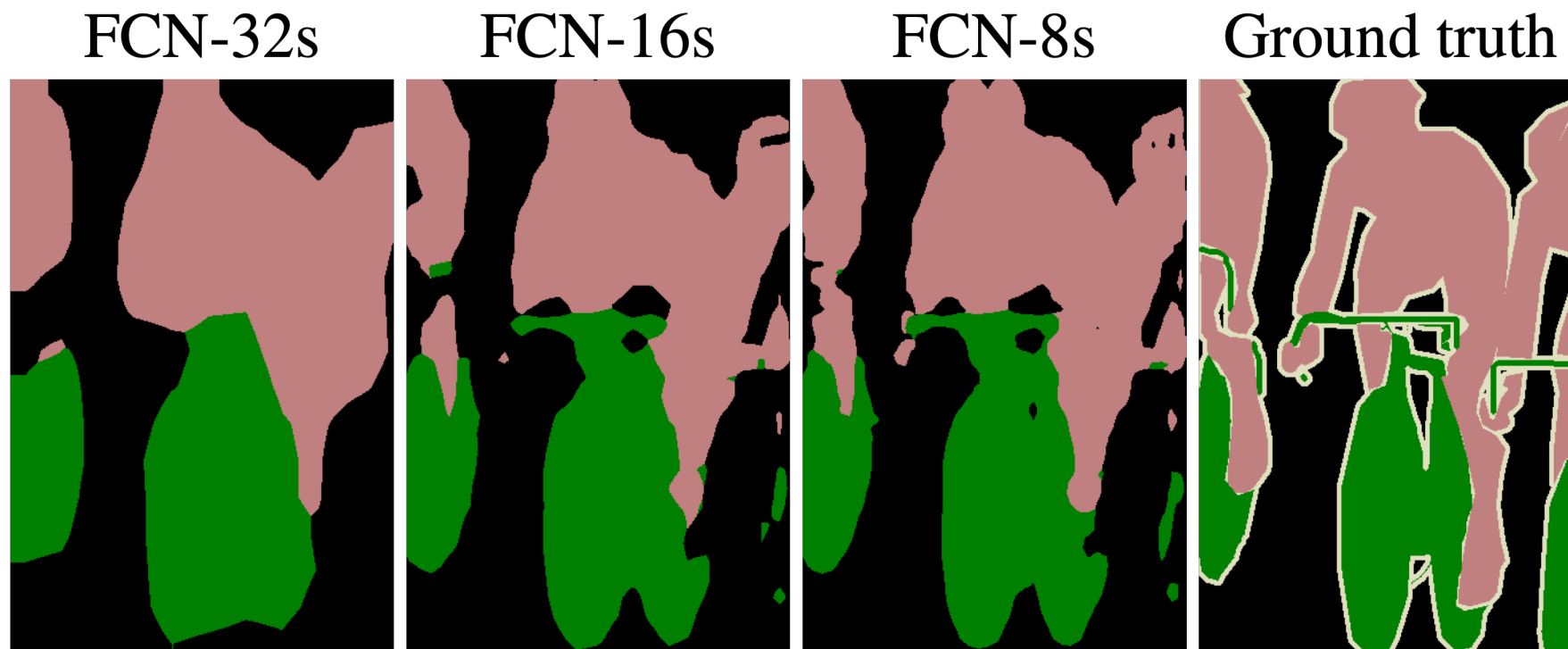


Pixel-wise  
Softmax

## Fully Convolutional Networks for Semantic Segmentation

Jonathan Long\*   Evan Shelhamer\*   Trevor Darrell  
UC Berkeley  
{jonlong, shelhamer, trevor}@cs.berkeley.edu

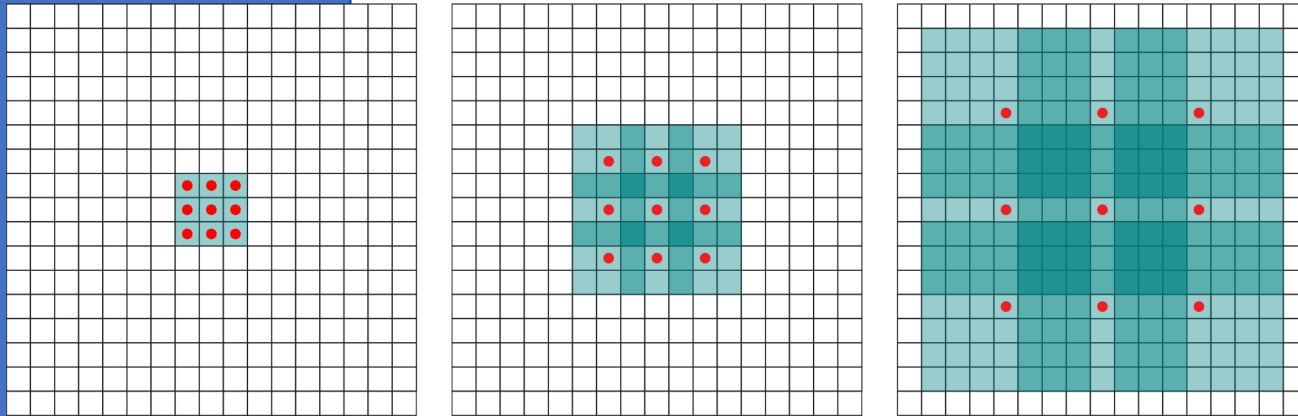
# Trouble with distinguishing between boundary pixels



Fully Convolutional Networks for Semantic Segmentation

Some modifications can mitigate the issue to some extent

## Alternative I: dilated convolutions



- Convolution with dilated filters
- Can achieve substantial increase in receptive field
- No loss of resolution or coverage
- Leads to results that do not lose resolution, maintaining high level details

MULTI-SCALE CONTEXT AGGREGATION BY  
DILATED CONVOLUTIONS

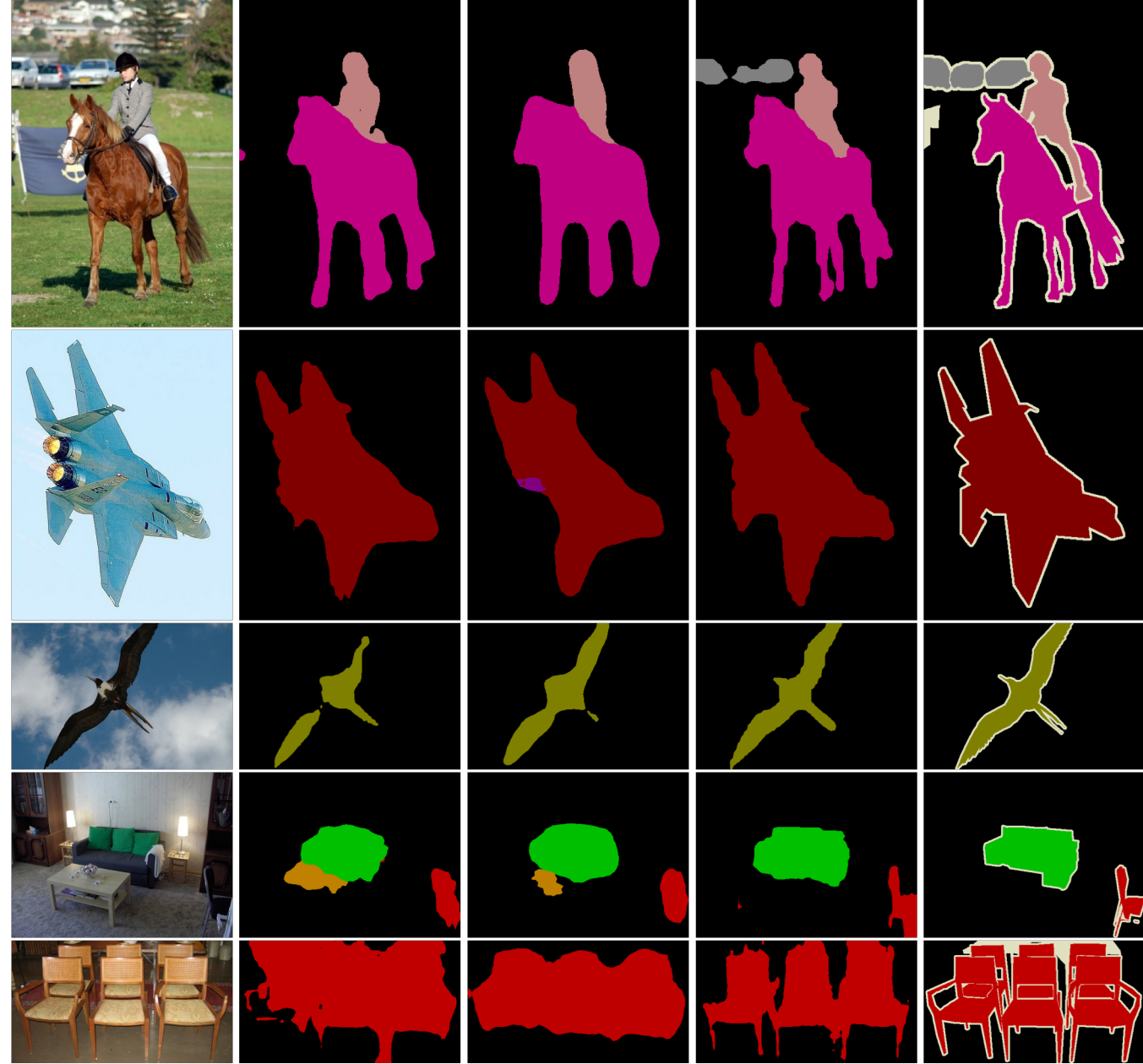


# Computer Vision

## MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS

Fisher Yu  
Princeton University

Vladlen Koltun  
Intel Labs



(a) Image

(b) FCN-8s

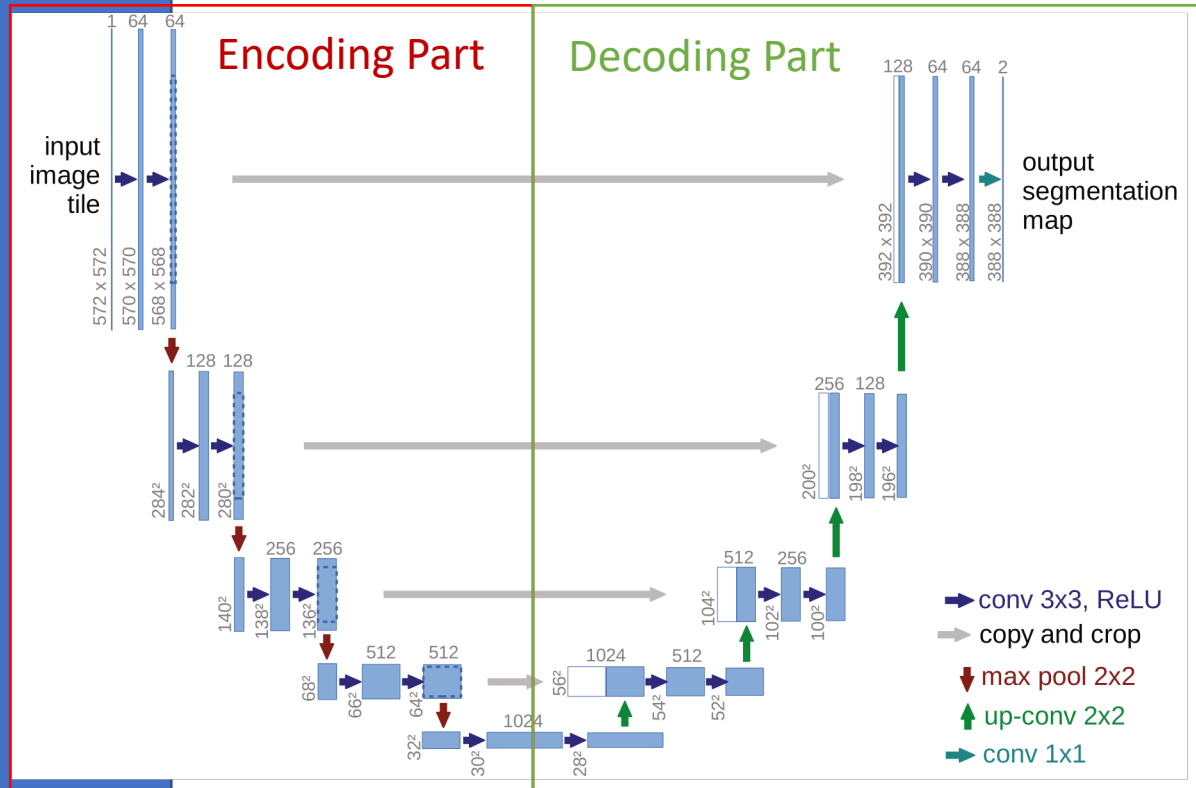
(c) DeepLab

Dilated  
Convolutions

(e) Ground truth

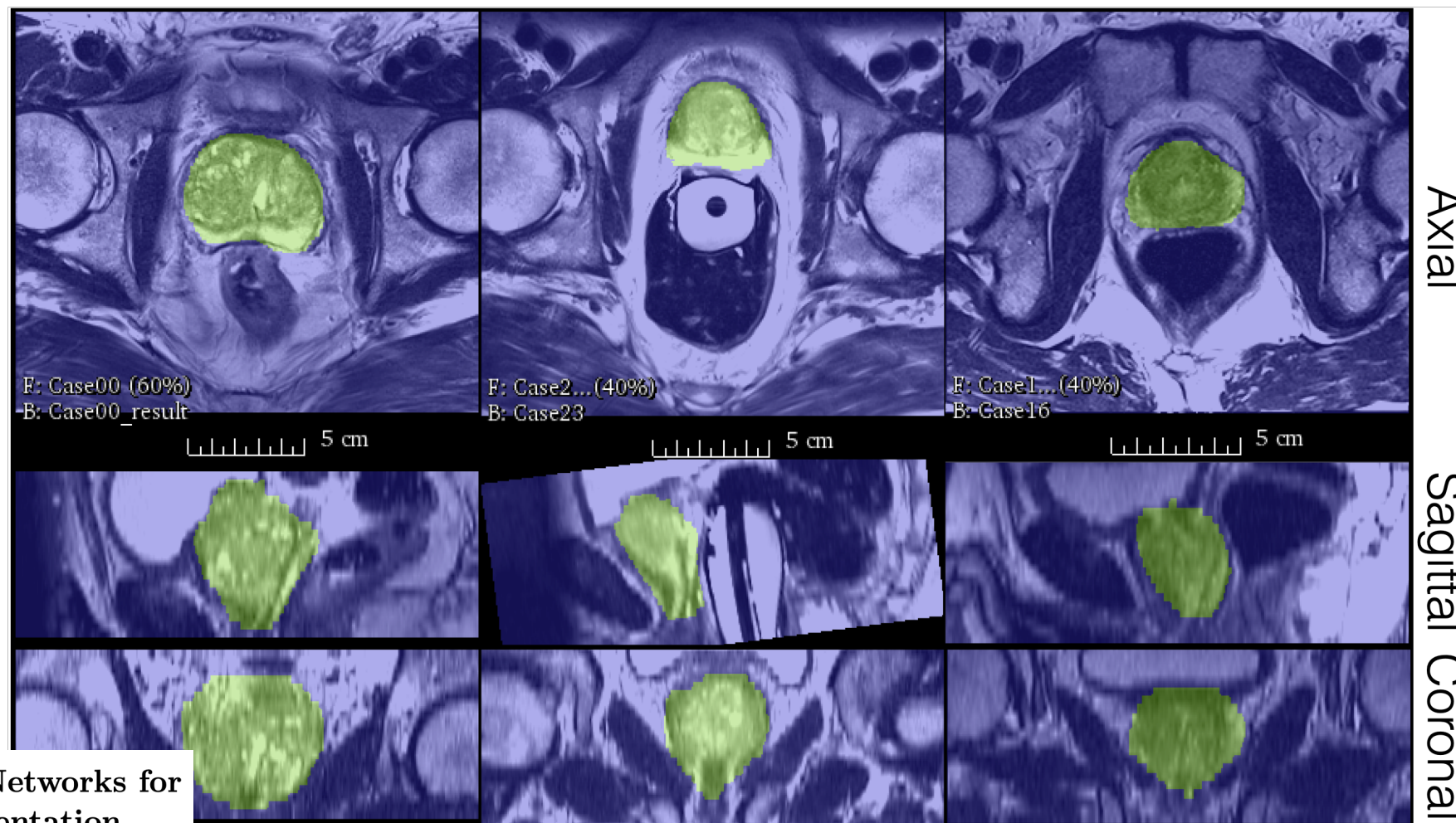
[Image adapted from the publication]

## Alternative II: Skip Connections – U-Net



- The **encoding part** of the network is like the fully convolutional network
- It pools local information and achieve global representation at the bottom layers
- The skip connections passes high-resolution information to retain information necessary to distinguish neighboring boundary pixels.

# Context and fine level



**V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation**

## To help with training

- CNNs have large number of parameters
- It is important to have lots of training images to be able to train deep networks
- In smaller training size scenarios there are various tools that may help
  - Normalization: batch, layer, ...
  - Drop-out
  - Regularization
  - Data augmentation
  - Transfer learning

## Data augmentation

- Augment the training set with random transformations of the observed samples:
  - Rotations, scaling, up-down & left-right flip, ...
- This simple approach is used very widely



## Transfer learning – finetuning

- There are many networks available online that has been trained with ImageNet – with > 1 million images
- Transfer learning takes a pre-trained network and retrains it for the new task and new dataset starting from the learned weights
- You can also consider taking a part of the network instead of the whole

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.

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 × 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 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 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 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144