Deep Learning for Computer Vision Part II: Convolutional Neural Networks

Computer Vision

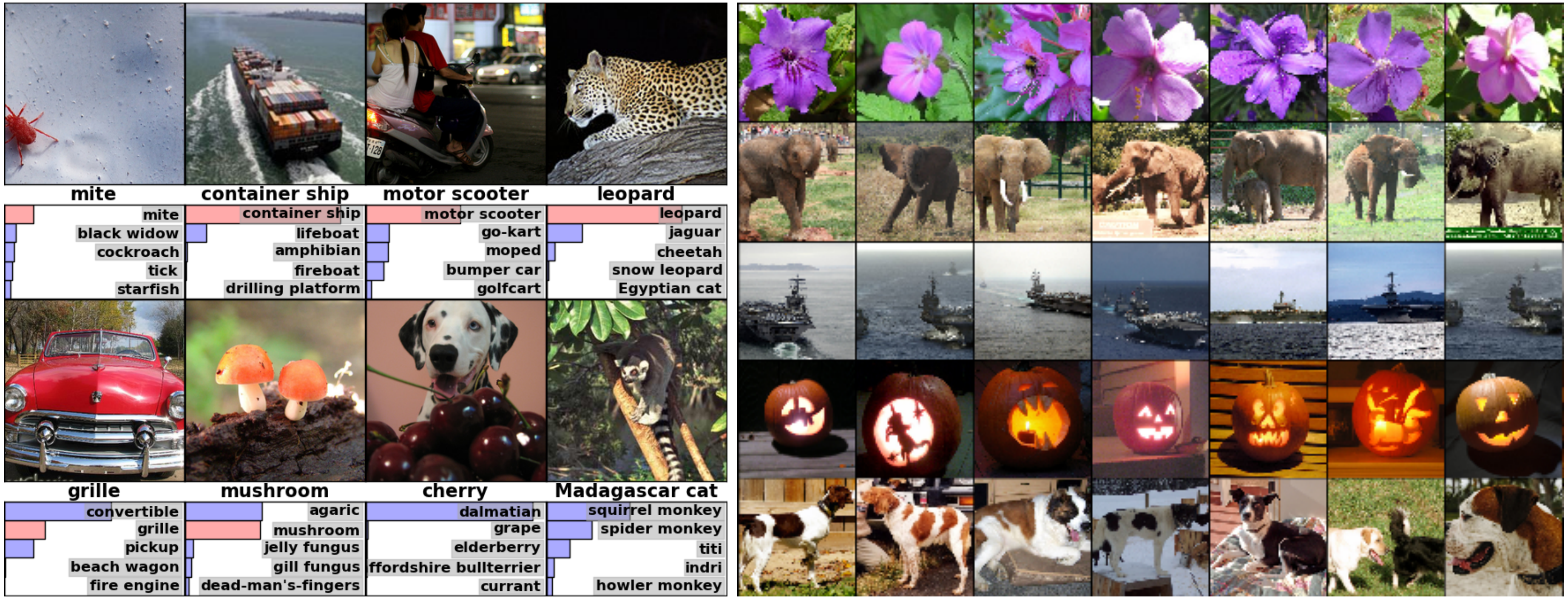


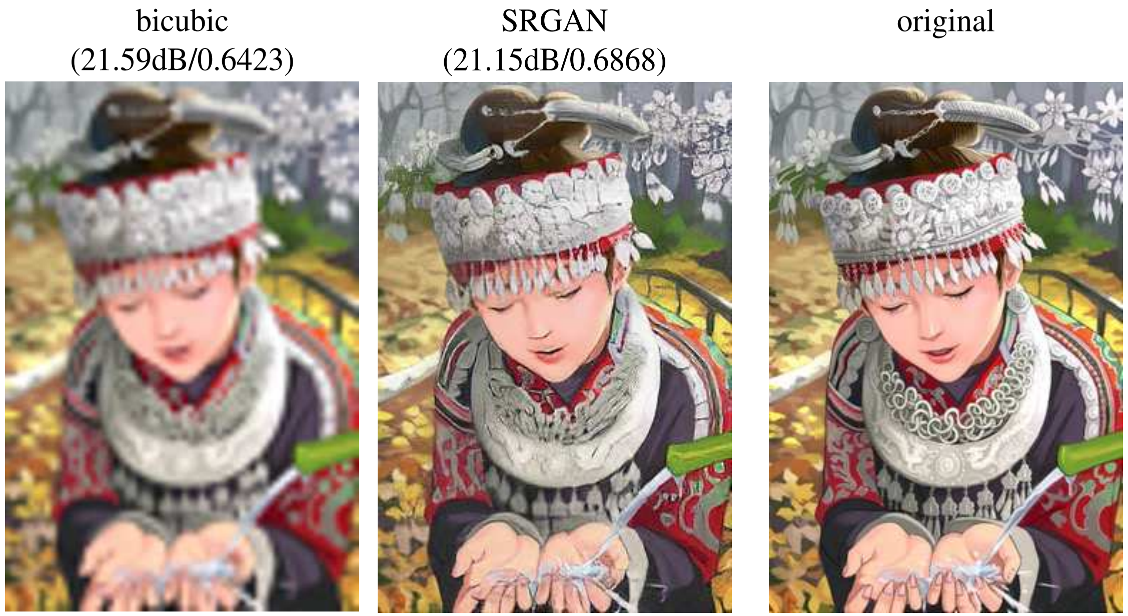
Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010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

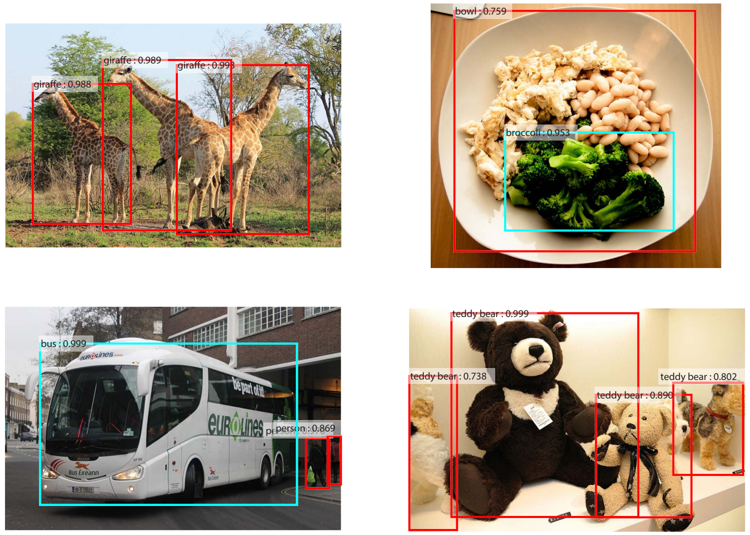
Computer Vision

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]

Computer Vision

Outline

1. Introduction to neural networks
   Basics of neural networks

2. Convolutional neural networks
   Basic applications of deep learning to image analysis and computer vision

3. Advanced topics and applications

Computer Vision

Recap from the lecture on introduction to neural networks

We always considered x as a vector.
What to do when x is an image?

Hidden layers

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

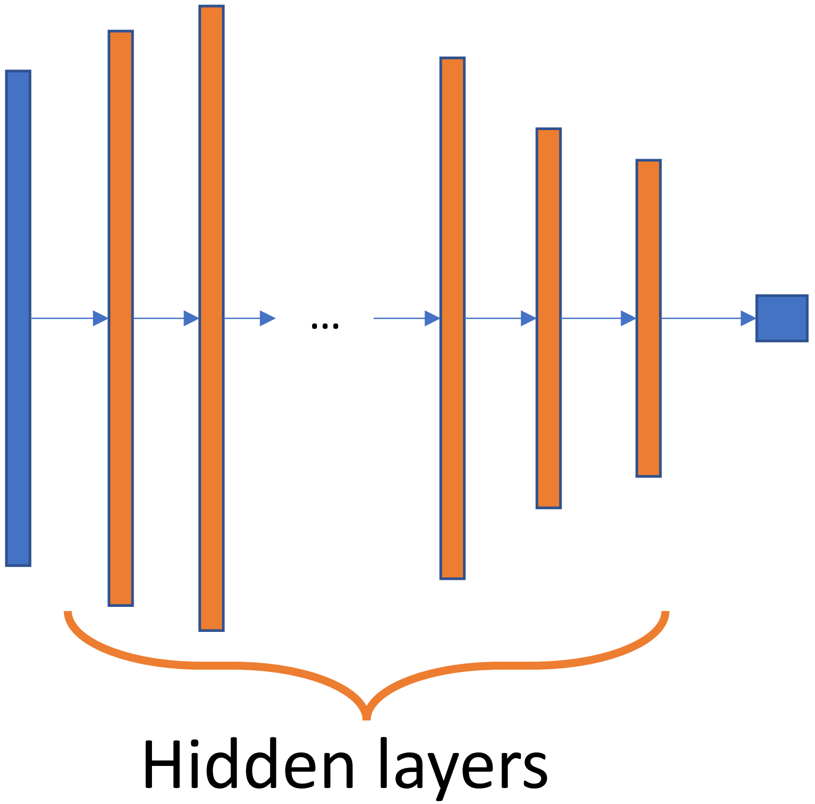
$$h_0 = \mathbf{x},\ h_L = f(\mathbf{x}; \theta)$$

Function view

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

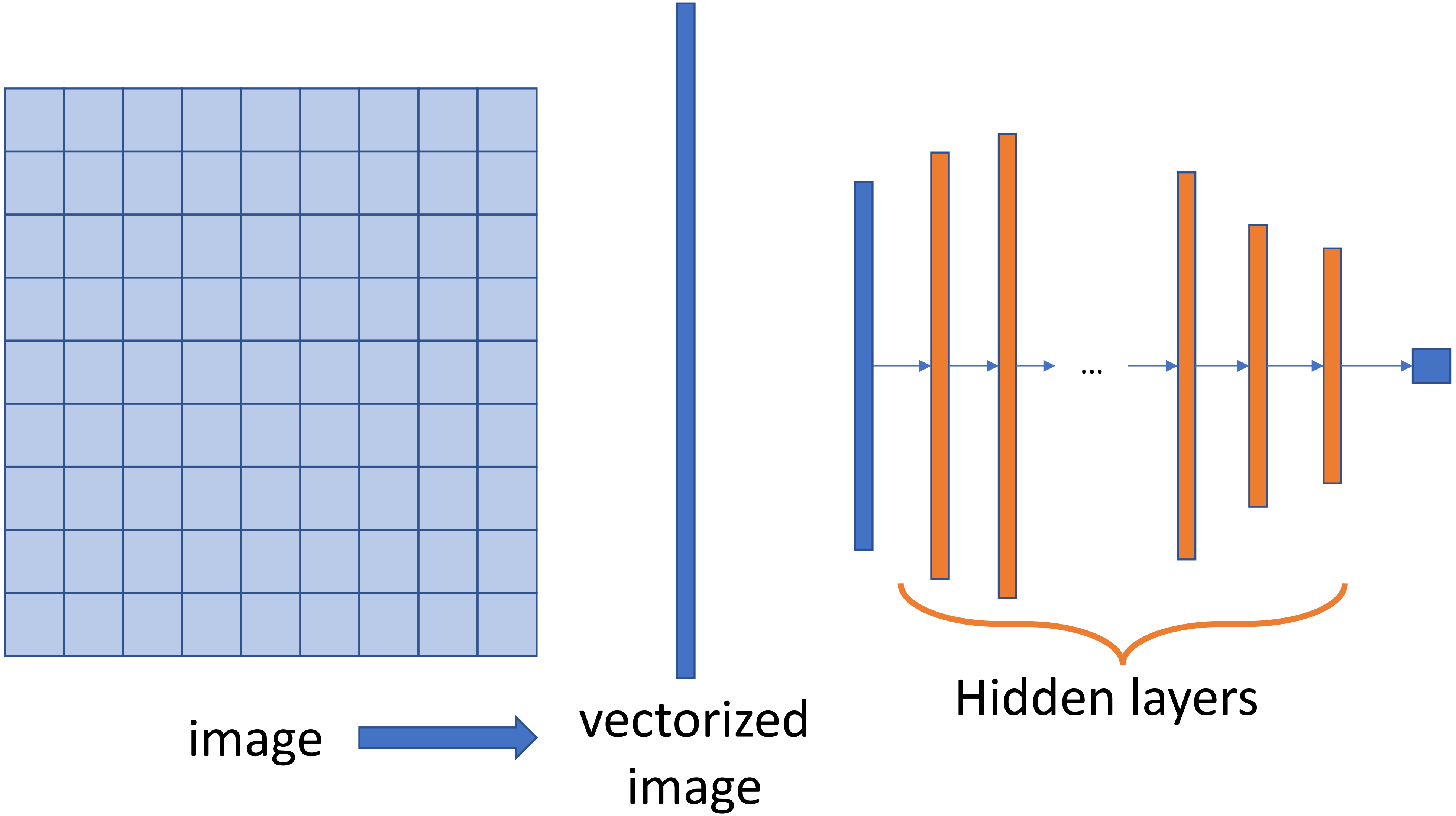
$$f(\mathbf{x}; \theta) = f_L \circ \cdots \circ f_2 \circ f_1(\mathbf{x})$$

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



Computer Vision

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_0xM_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_0xM_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!

Computer Vision

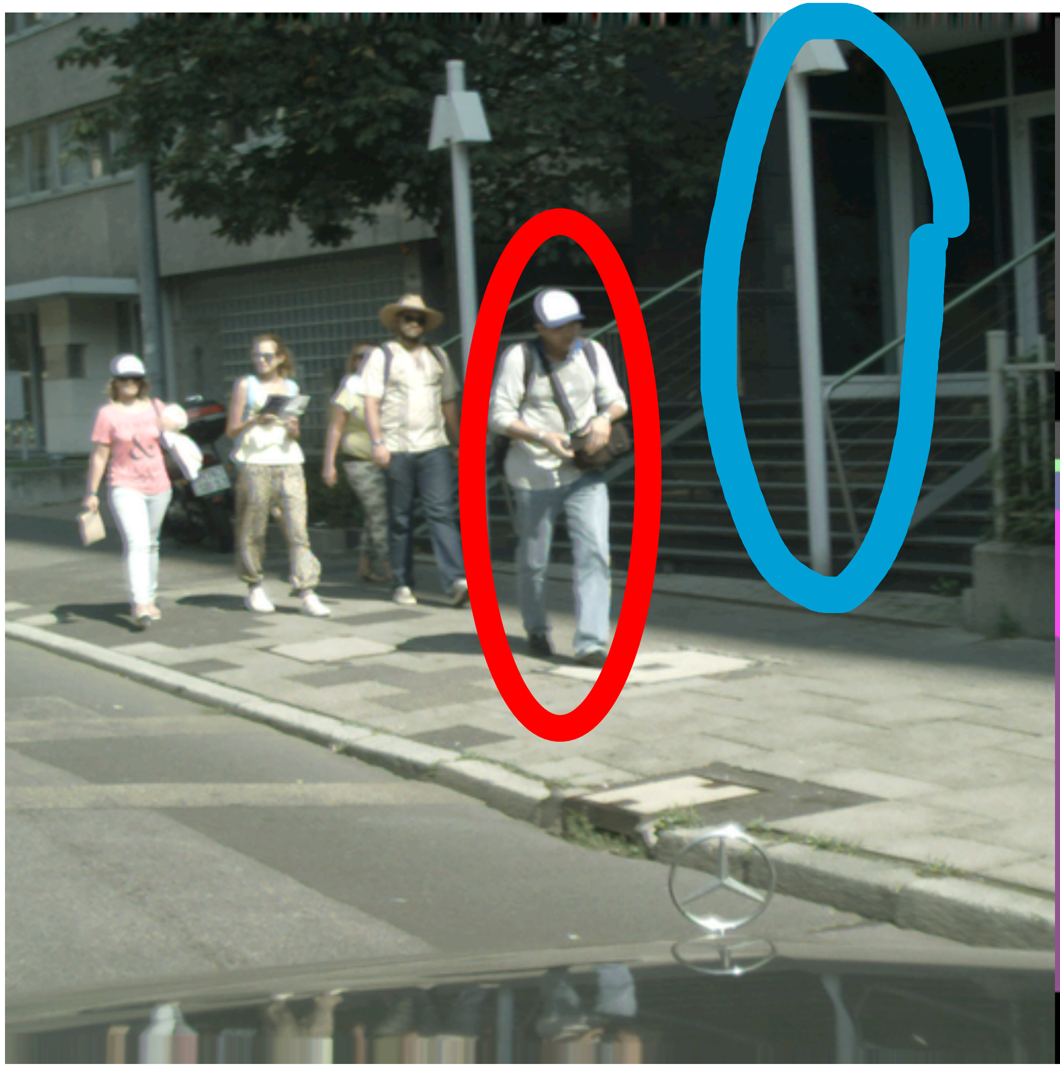
The activation is a bit problematic

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

- 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



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
3. Lack of translation invariance

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

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

Computer Vision

The non-linearity will remain the same

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

Fully connected architecture

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

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

Convolutional architecture

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

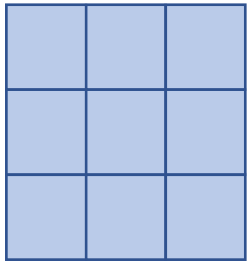
Convolutional layers

Computer Vision

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



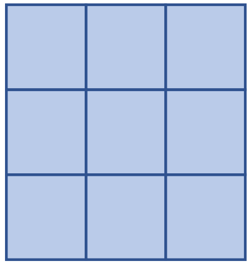
Convolution kernel: w

Computer Vision

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

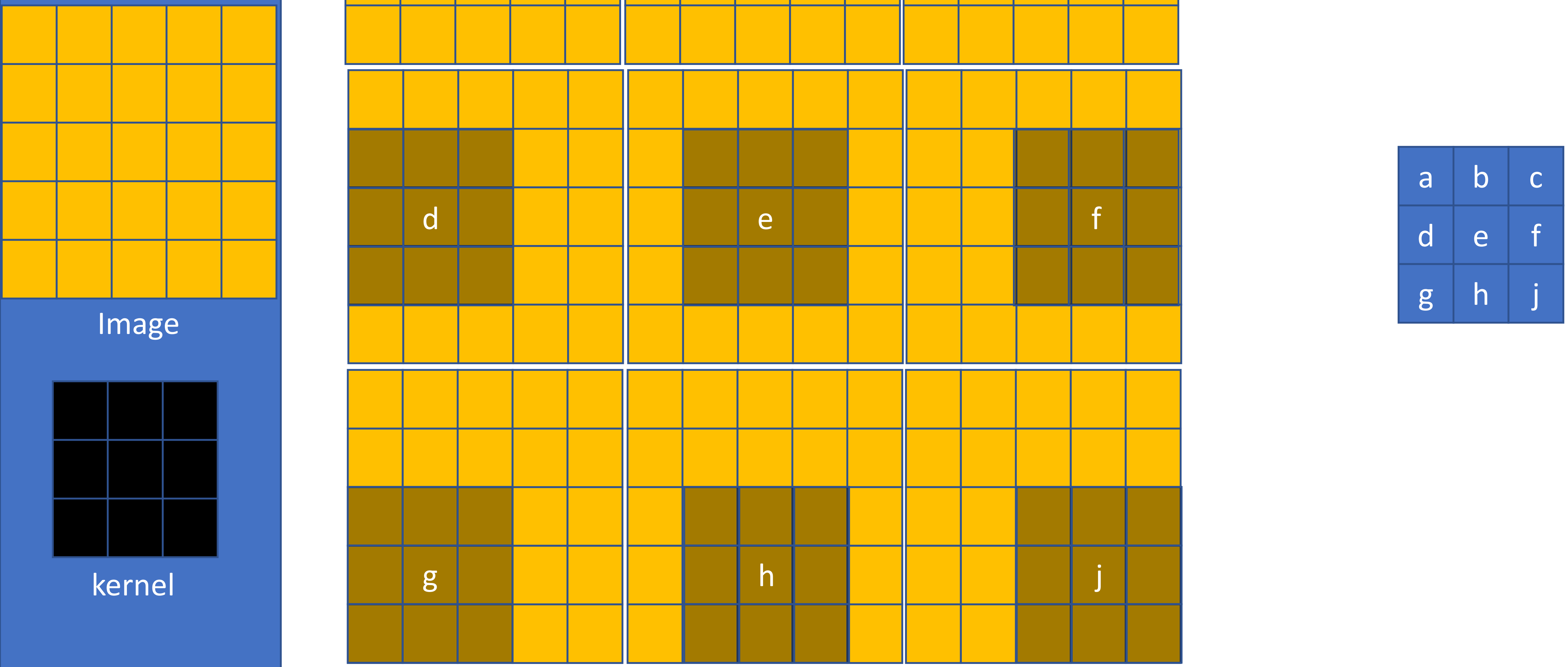
Image: x



Convolution kernel: w

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

Computer Vision



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

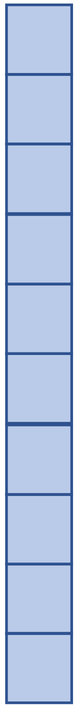
$$a_{l,k} = \sum_j w_{l,kj} * h_{l-1,j} + b_{l,k}$$

- 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.

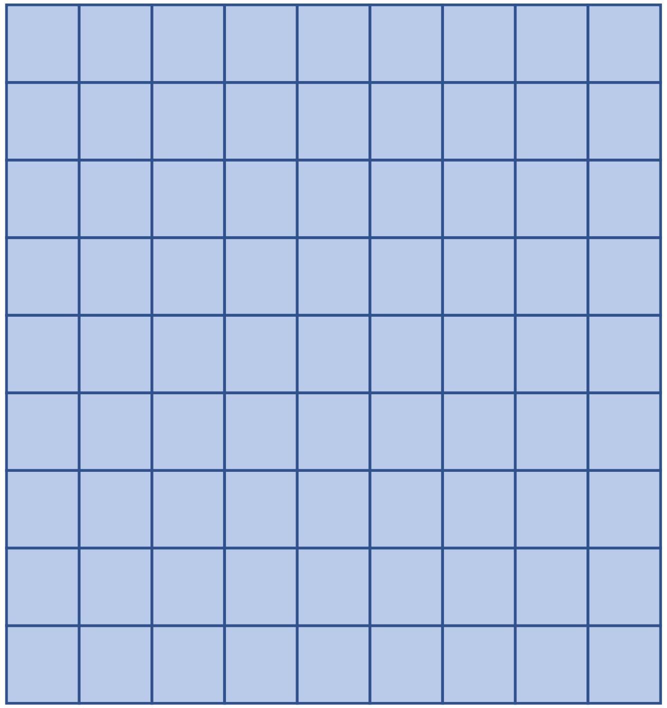
Nonlinearities following activations remain similar

Computer Vision

Image of neurons - channel



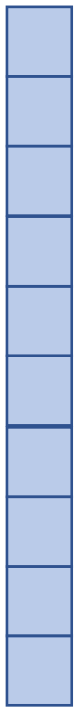
Fully connected
Vector of neurons form a layer



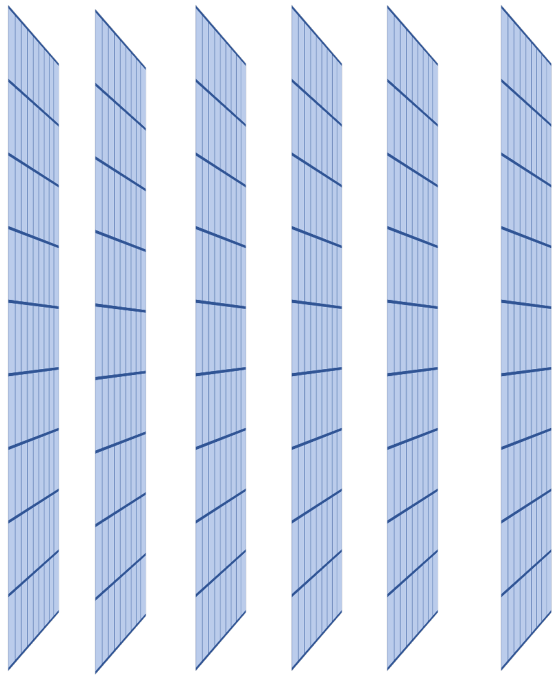
Convolutional architecture
Image of neurons form a channel

Computer Vision

Vector of channels forms a layer

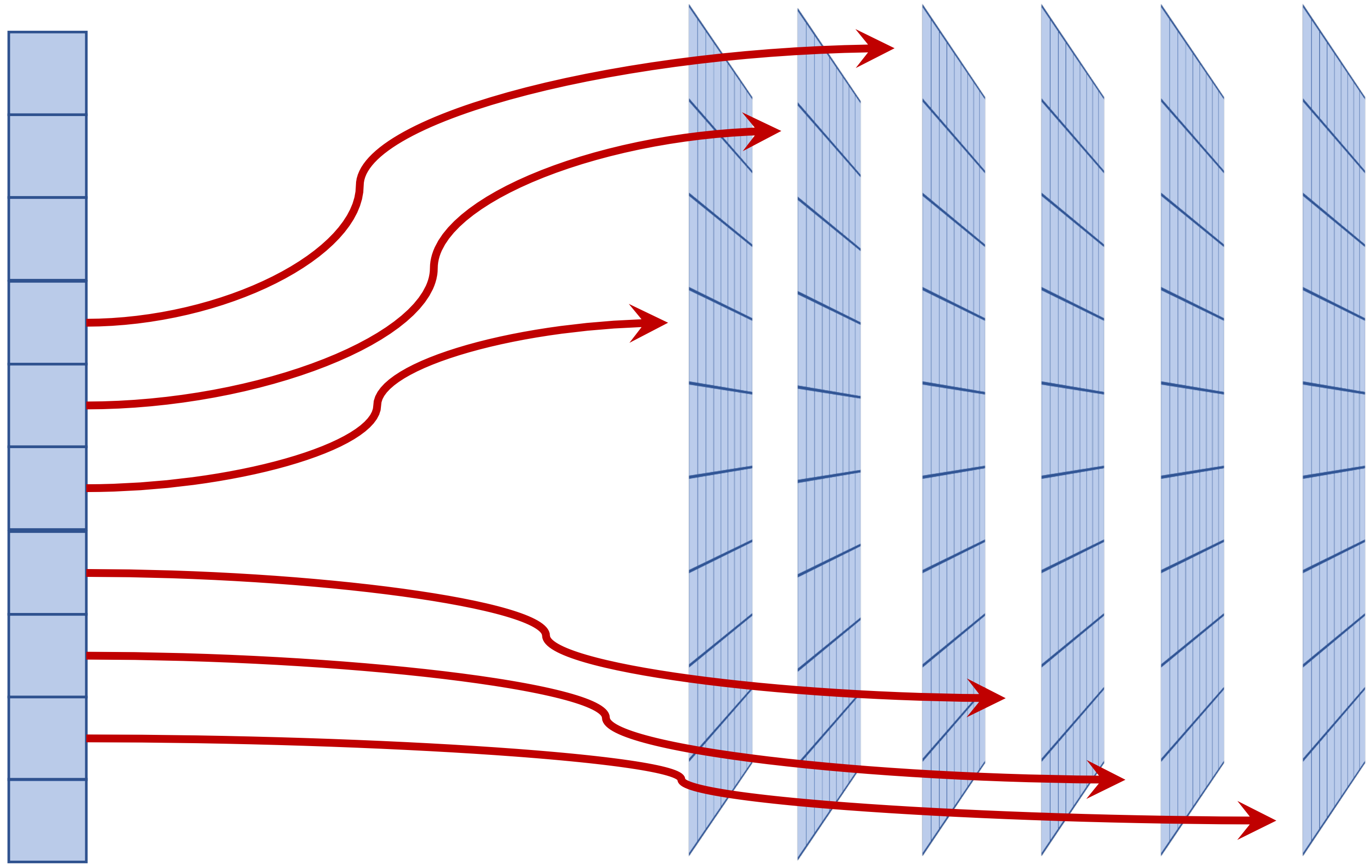


Fully connected
Vector of neurons form a layer



Convolutional architecture
Vector of channels form a layer

Analogy between two: each neuron becomes a channel

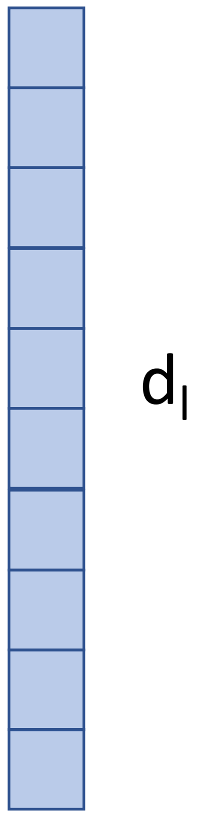


Fully connected
Vector of neurons form a layer

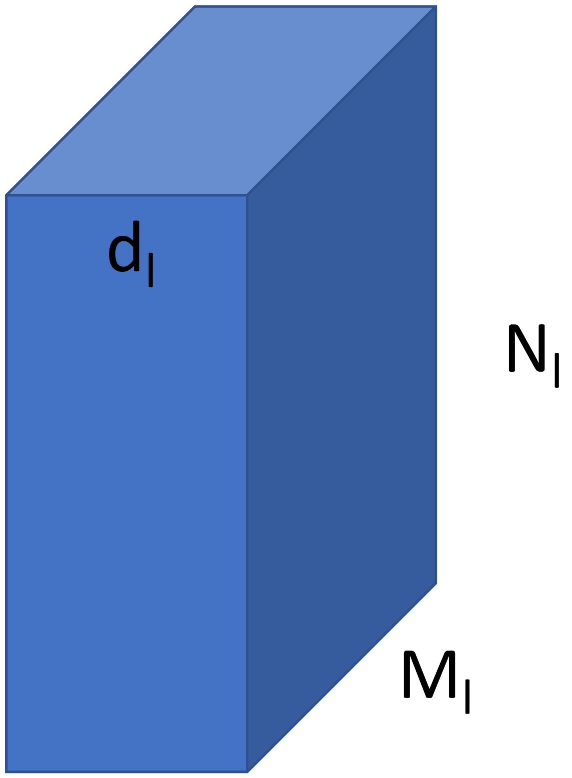
Convolutional architecture
Vector of channels form a layer

Computer Vision

Tensor representation



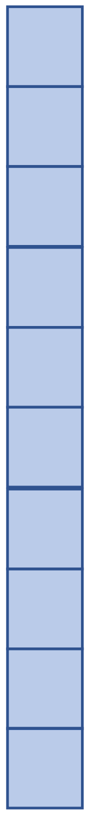
Fully connected architecture:
Vector representation of layers



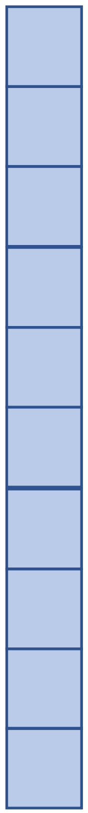
Convolutional architecture:
Tensor representation of layers

Computer Vision

We can also use the same graphical representation



Fully connected architecture:
d_l neurons



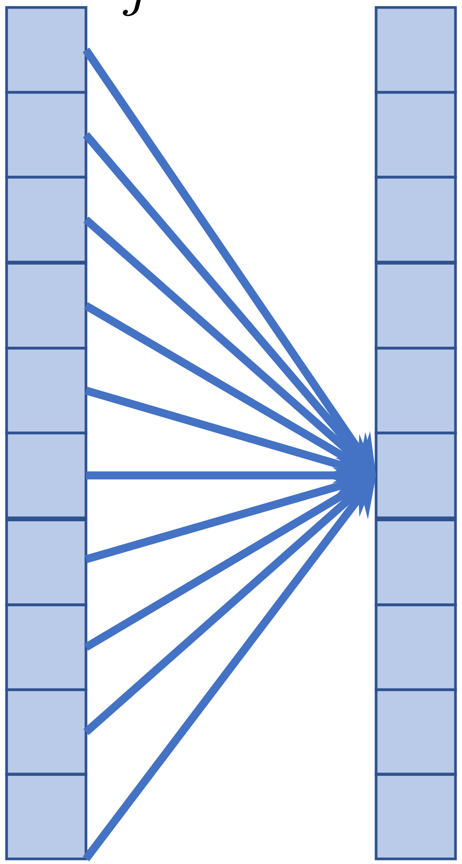
Convolutional architecture:
d_l channels

Computer Vision

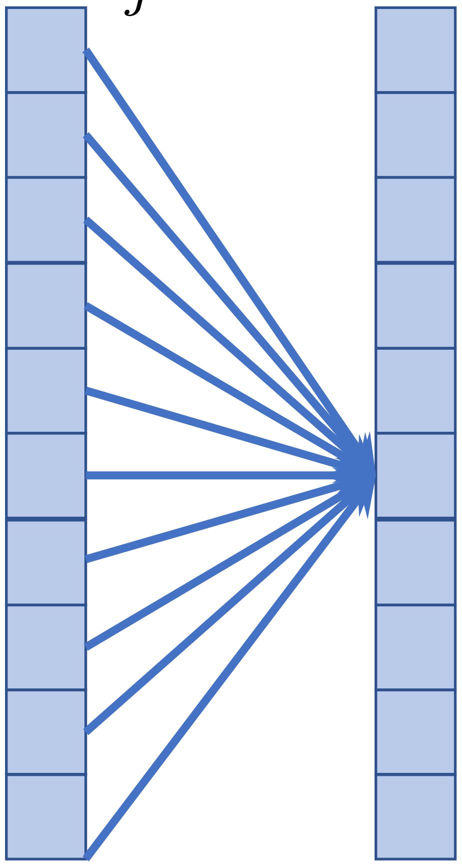
Connections represent convolutions

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

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



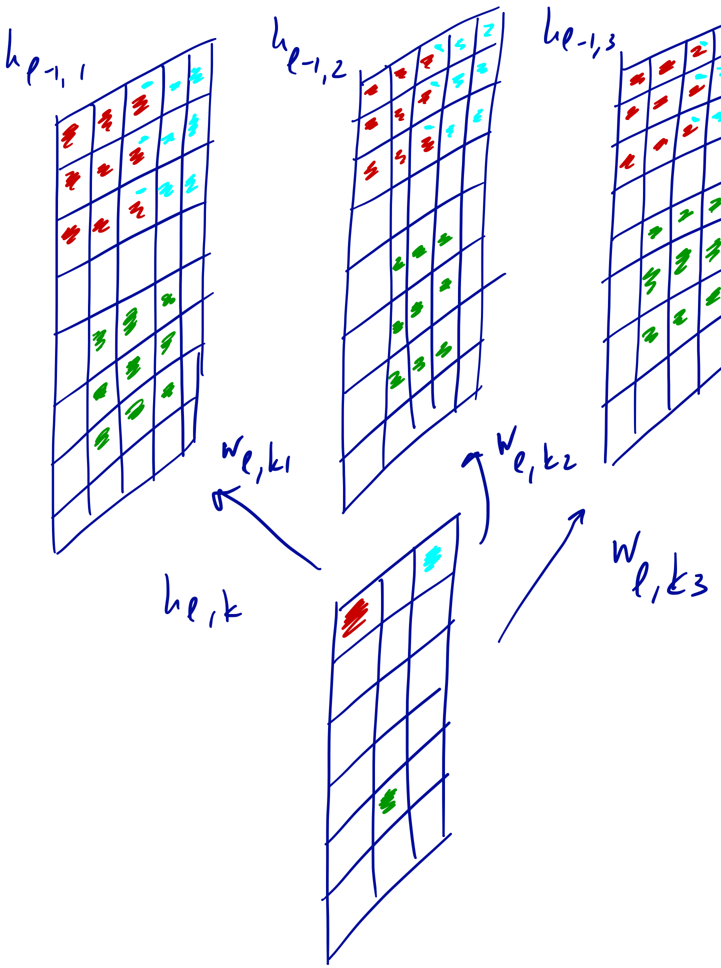
Fully connected link
Multiplication followed by addition



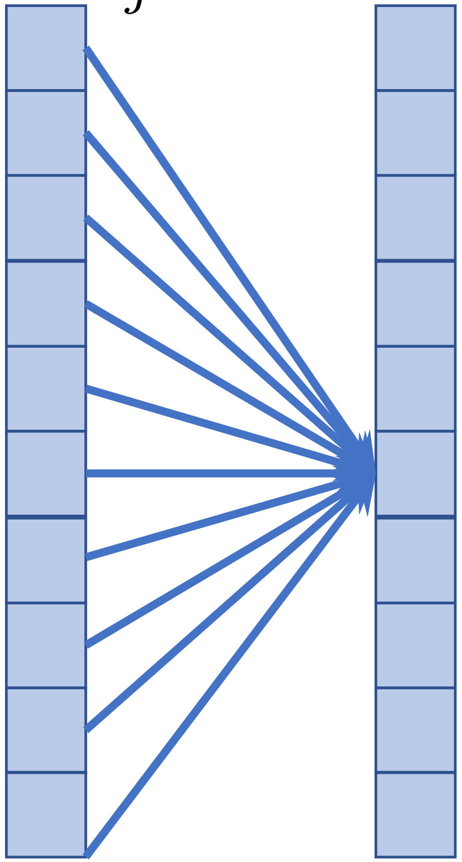
Convolutional link
Convolution followed by addition

Computer Vision

Connections represent convolutions



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

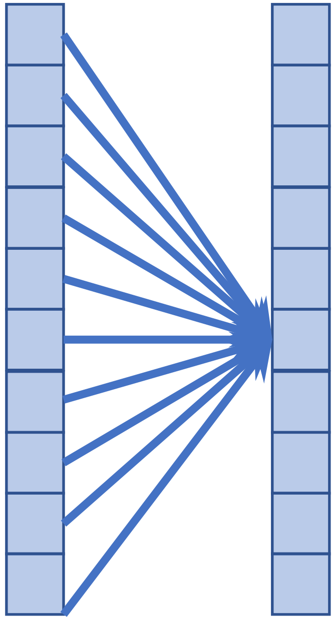


Same filter is used for all neurons in the same channel: **Weight sharing**

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

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

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

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

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

$$(k_1 \times k_2) : \text{kernel size}$$

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

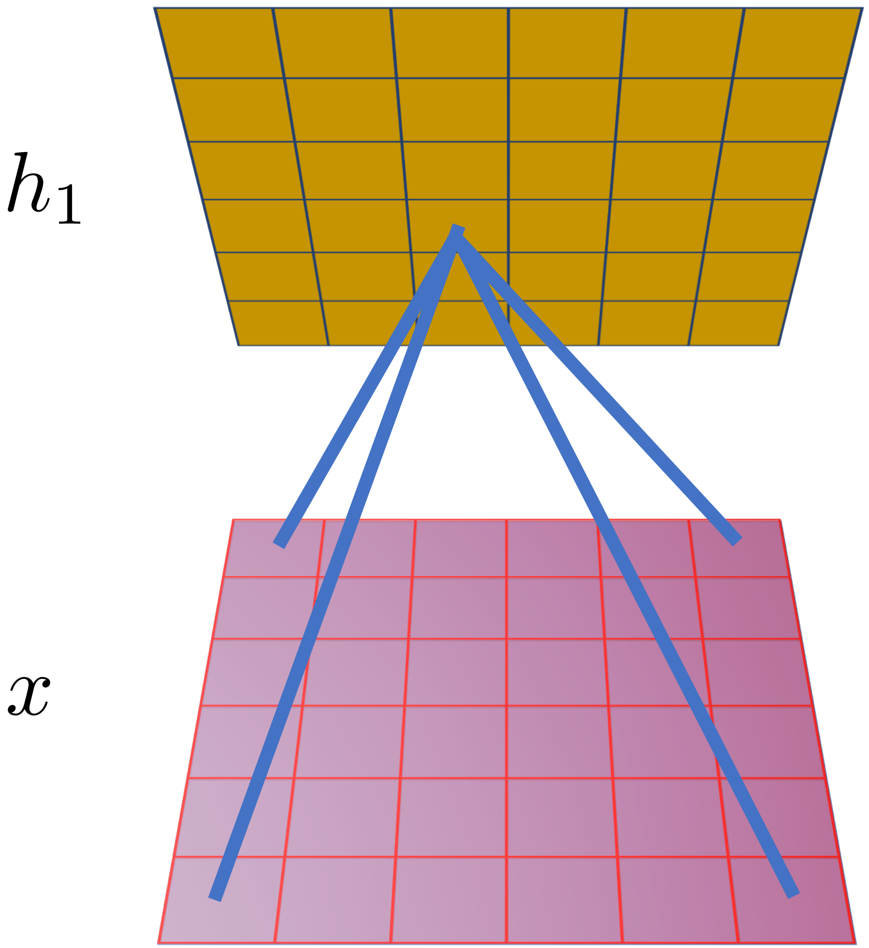
Larger layers with **sparser** connections with lower number of parameters

Computer Vision

Sparser connections

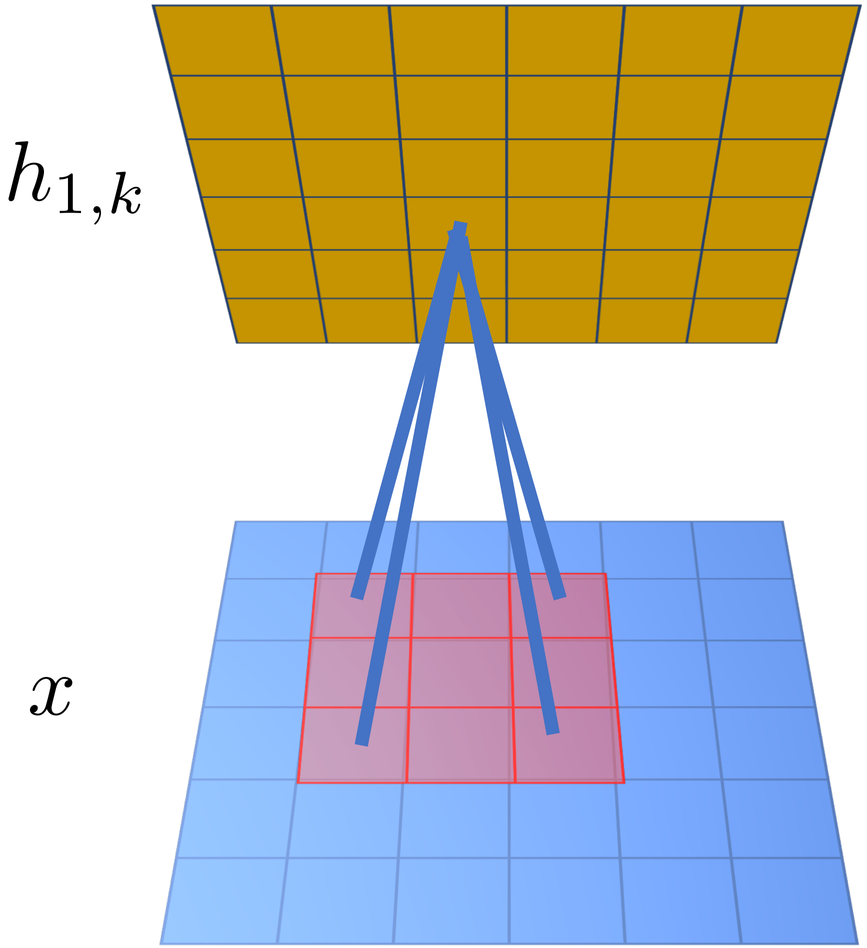
Naïve approach – fully connected

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



Convolutional link

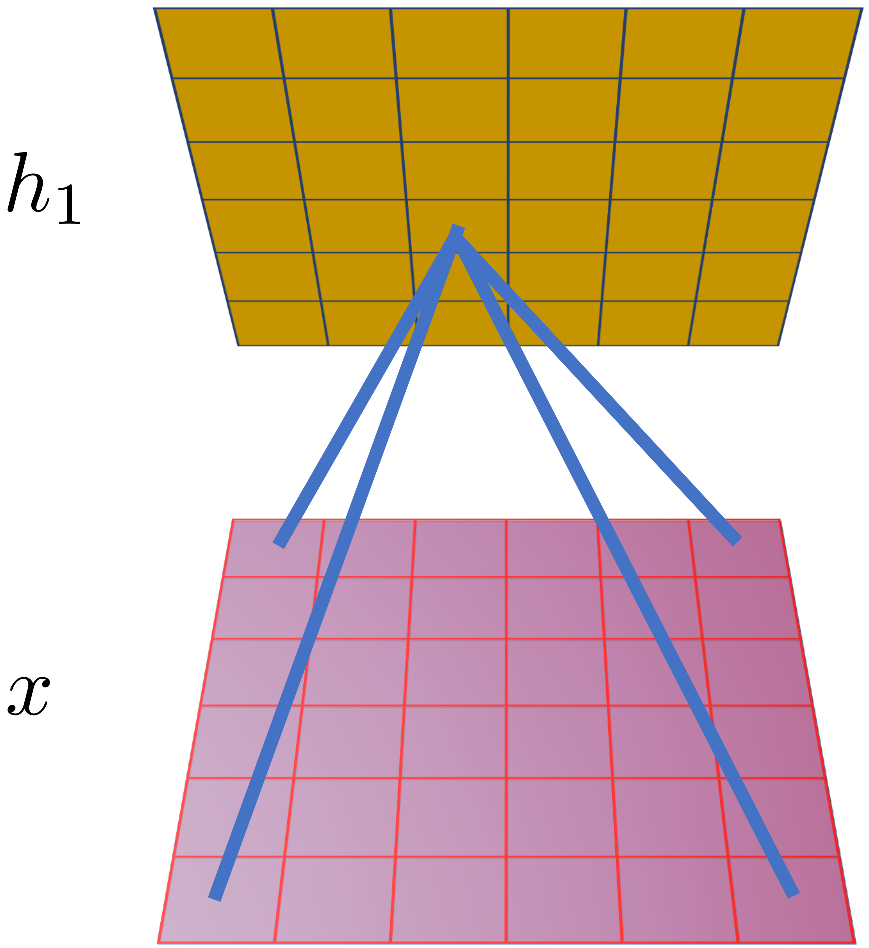
$$\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (k_1 \times k_2)}$$



Weight sharing – fewer parameters

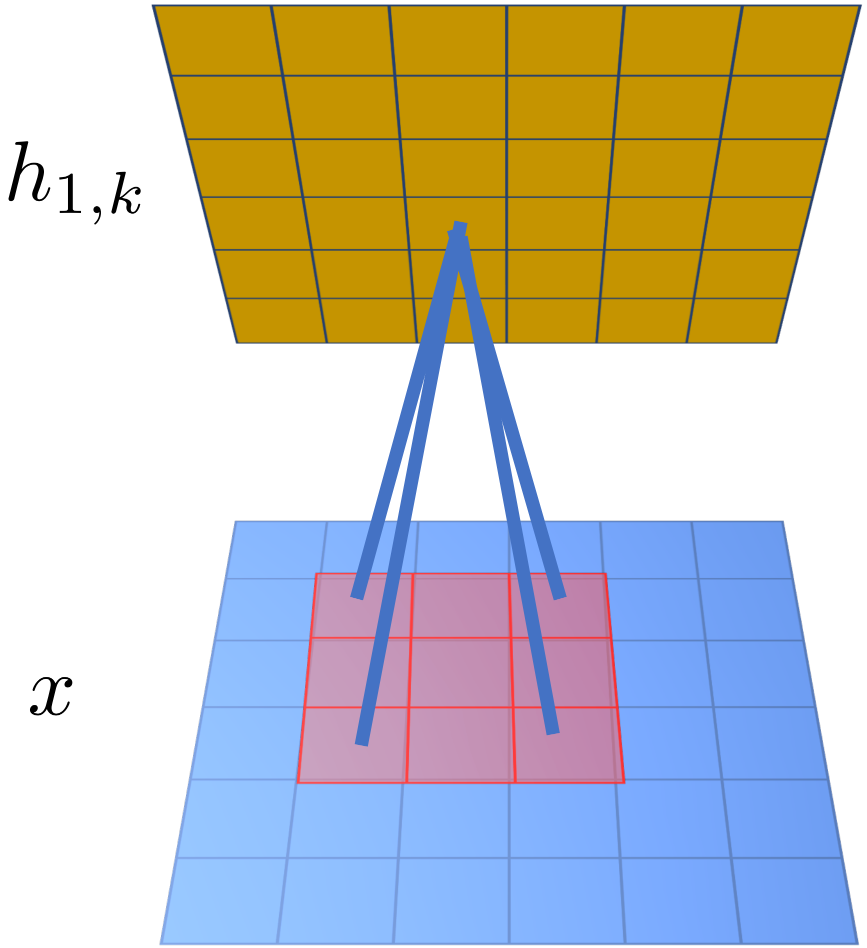
Naïve approach – fully connected

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



Convolutional link

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



Larger hidden layers

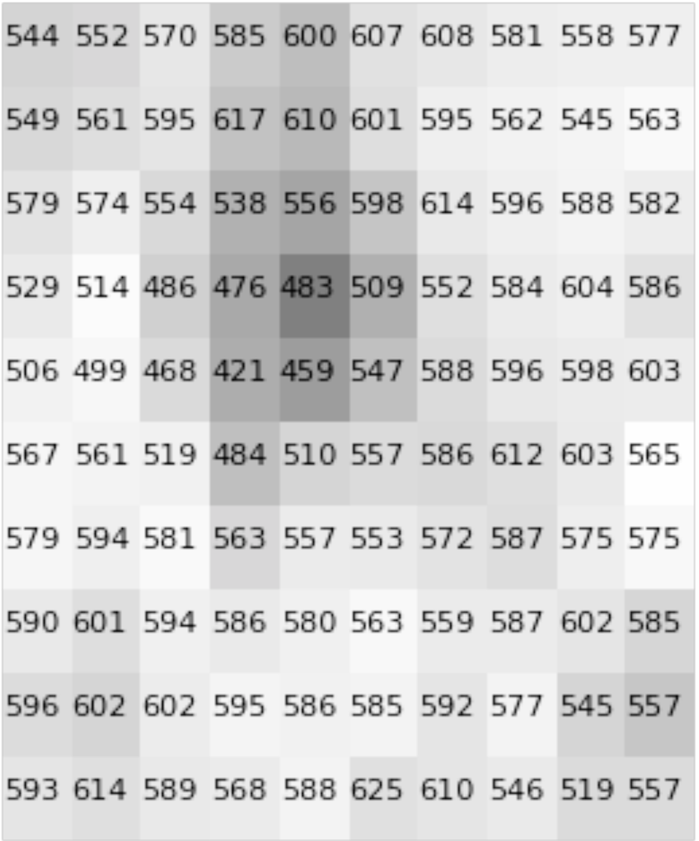
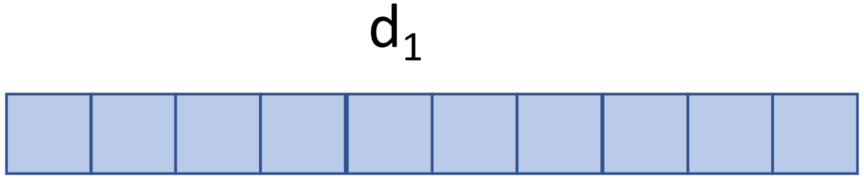
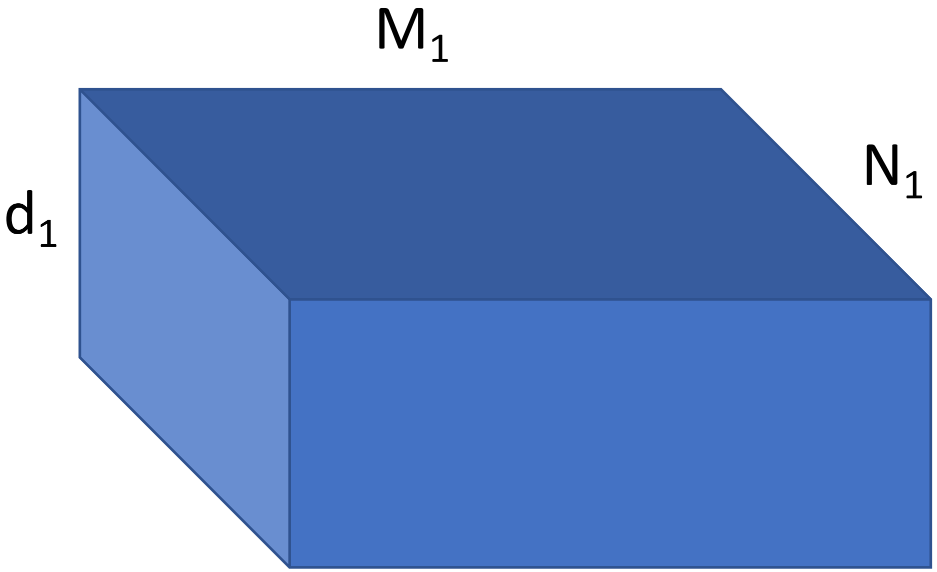


Image: $x \in \mathbb{R}^{M_0 \times N_0}$



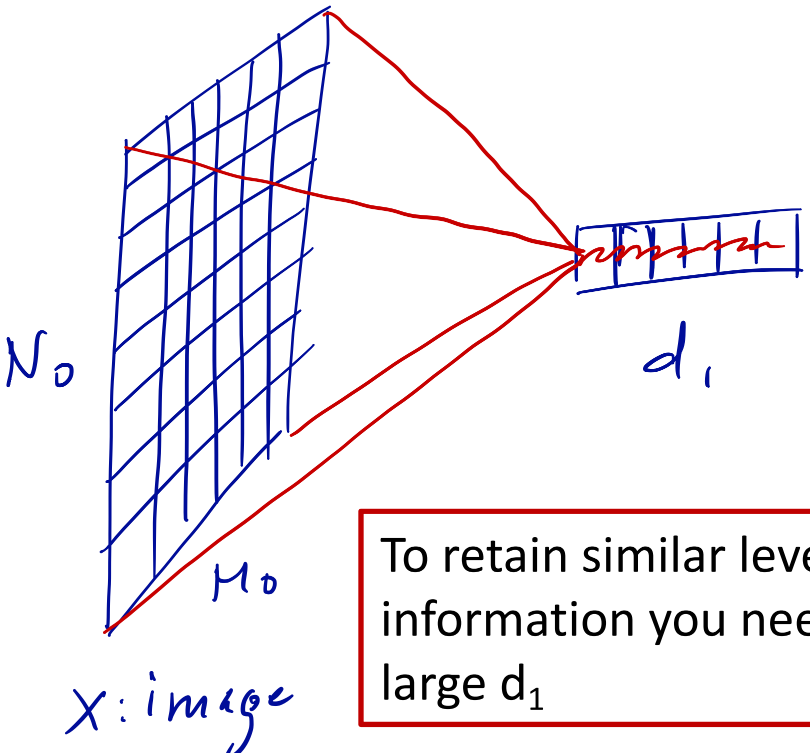
Fully connected: $h_1 \in \mathbb{R}^{d_1}$



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

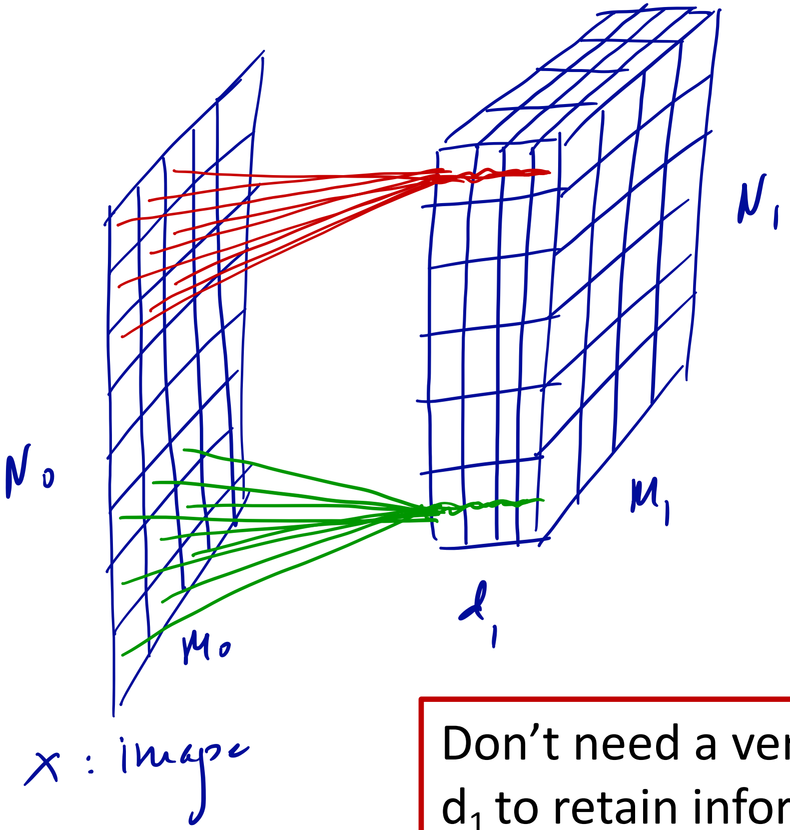
Computer Vision

Local vs. global information gathering



To retain similar level of information you need a large d_1

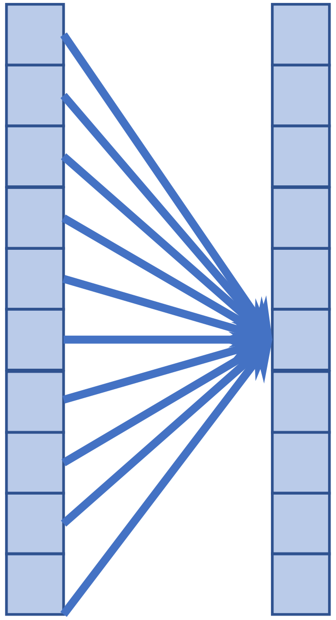
Fully connected architecture:
The entire image needs to be represented in a single vector



Don't need a very large d_1 to retain information

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

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

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

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

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

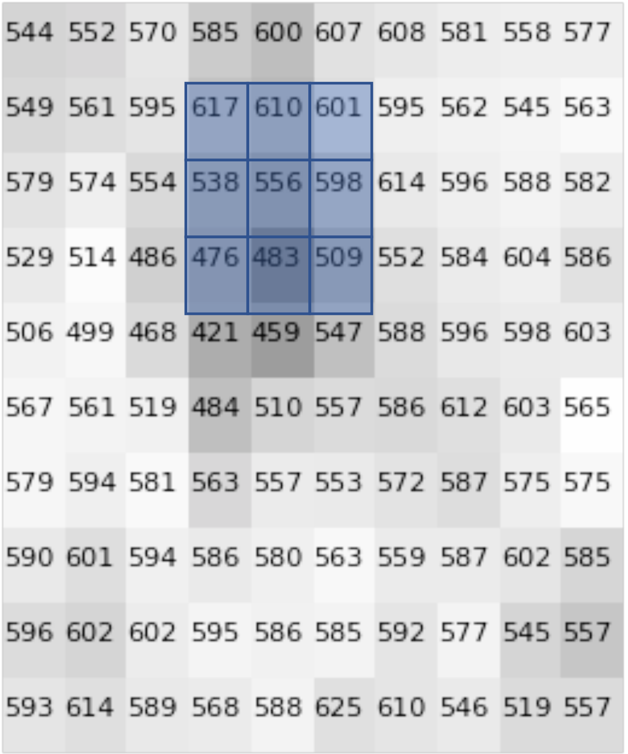
$$\mathbf{W}_l \in \mathbb{R}^{d_{l-1} \times d_l \times (k_1 \times k_2)}$$

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

Computer Vision

Channel size

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

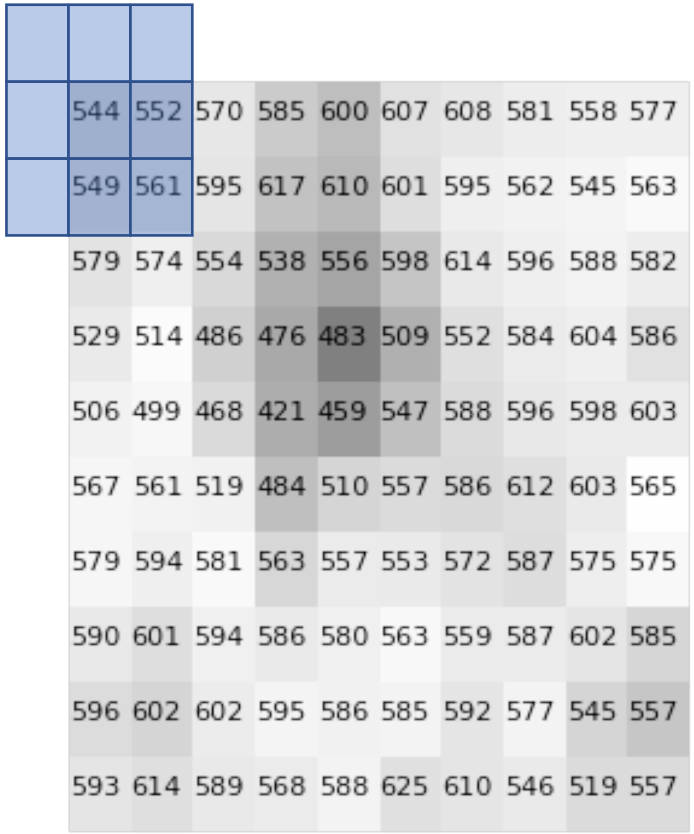


$$a_{ij} = \sum_{p} \sum_{q} x_{(i-p)(j-q)} w_{(p)(q)}$$

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

Channel size

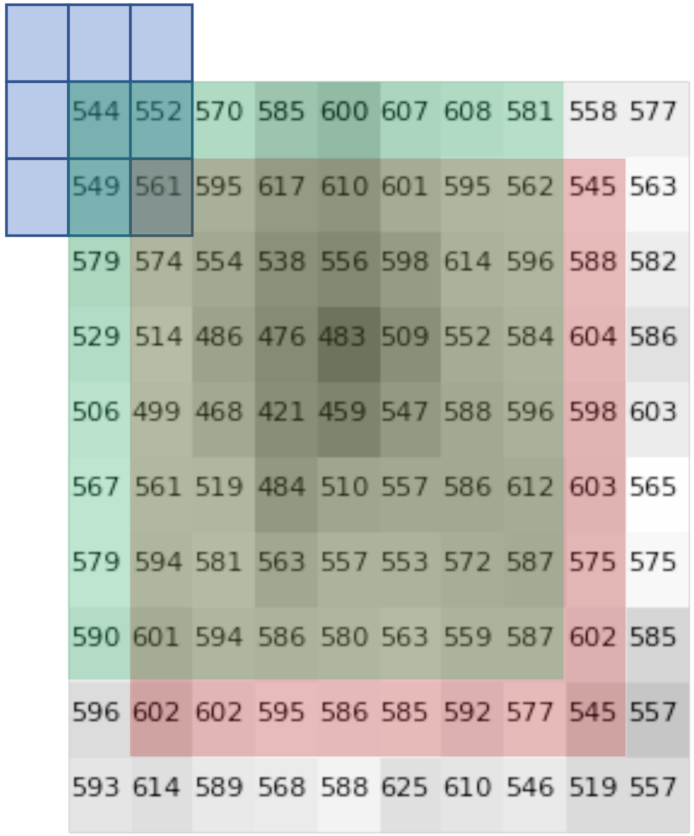
- Channel size is linked with kernel size and the type of convolution



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

- 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 loose 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 loose two pixels at the bottom and right of the image

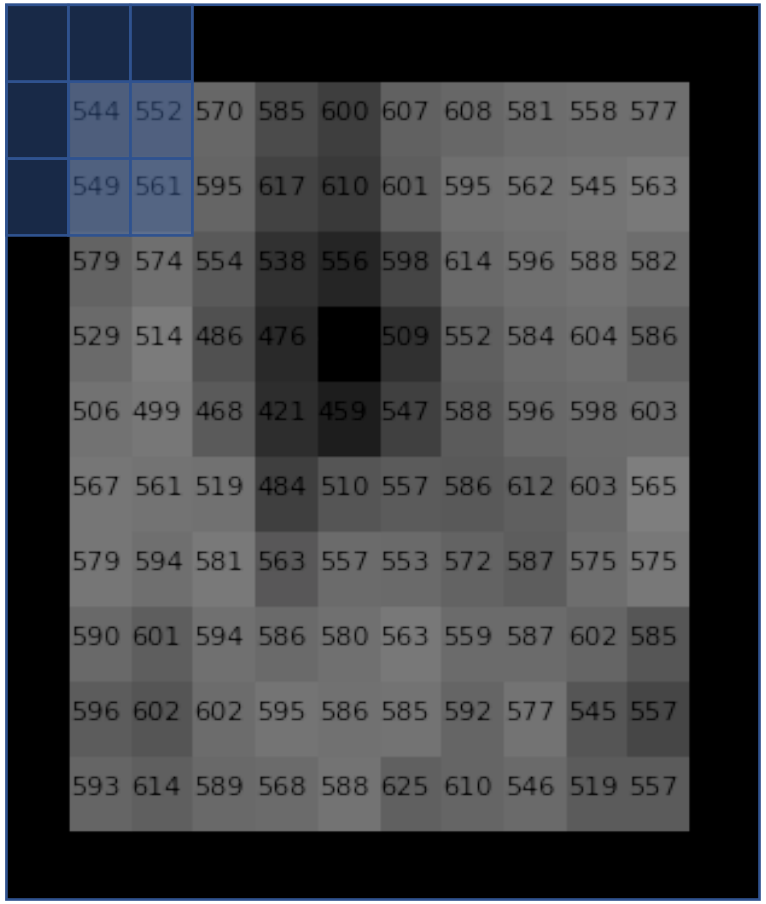
$$h_{l-1} \in \mathbb{R}^{d_{l-1} \times (M_{l-1} \times N_{l-1})}$$

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

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

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

Same padding



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

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

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

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

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

- 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

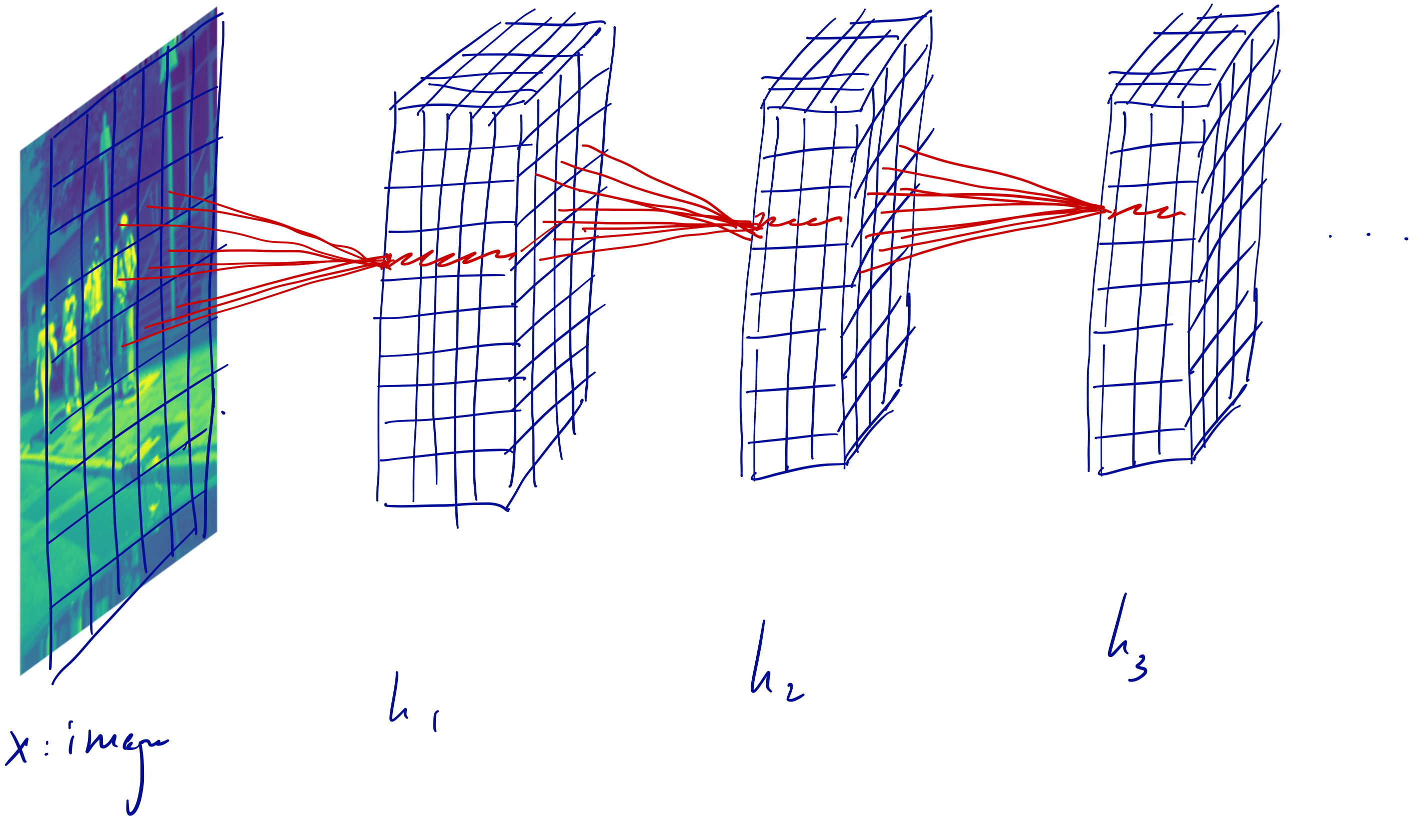
$$x \in \mathbb{R}^{64\times 64} \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 16$$

- 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

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

Computer Vision

Hierarchically aggregating local spatial features

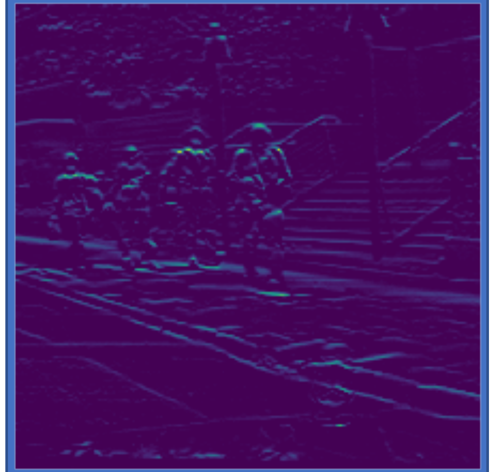


Extracting task specific features from the images.
As the layers progress, more global information is encoded.

Computer Vision

Hierarchically gathering local spatial statistics

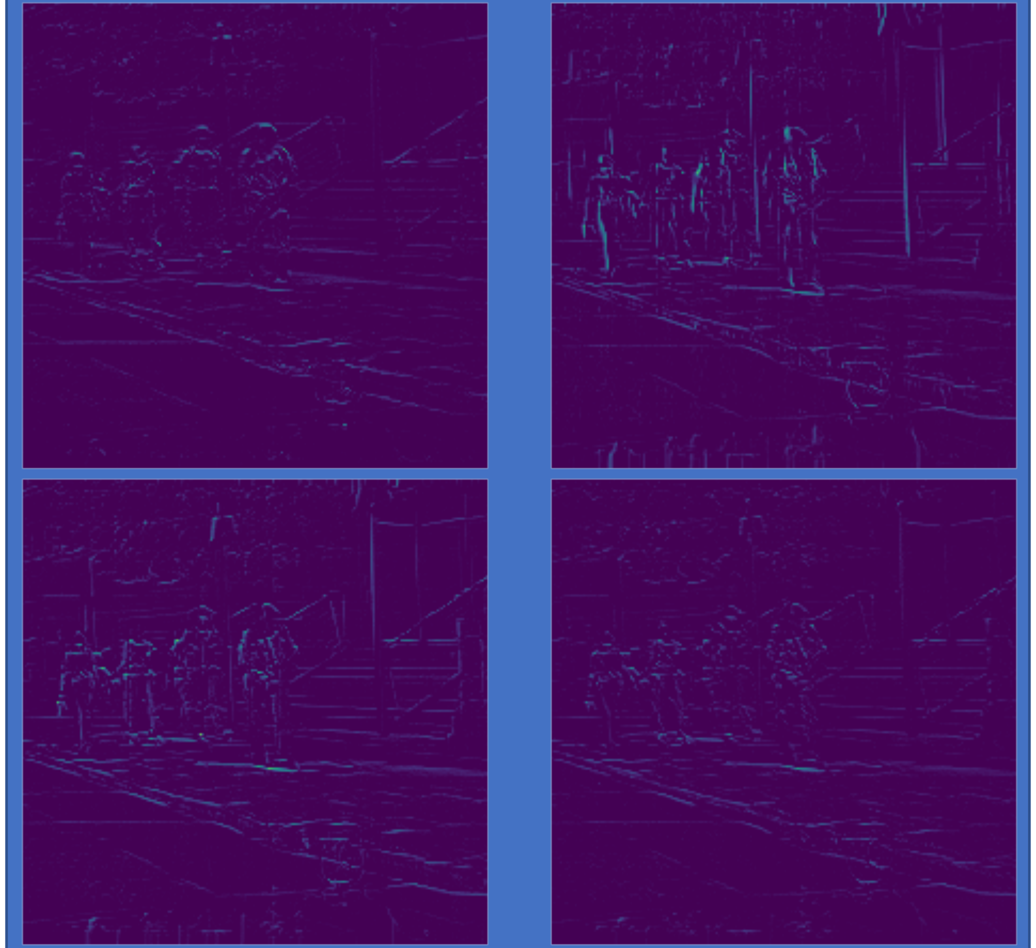
Layer 1



x derivative after Relu

y derivative after Relu

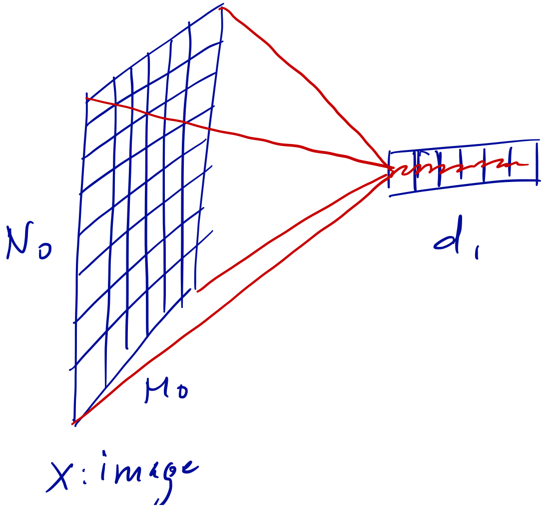
Layer 2



Different channels in the 2nd layer
Weighted sums of 2nd order derivatives

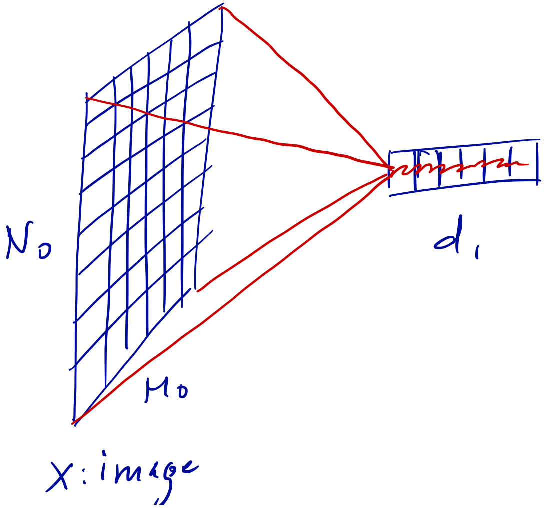
Computer Vision

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

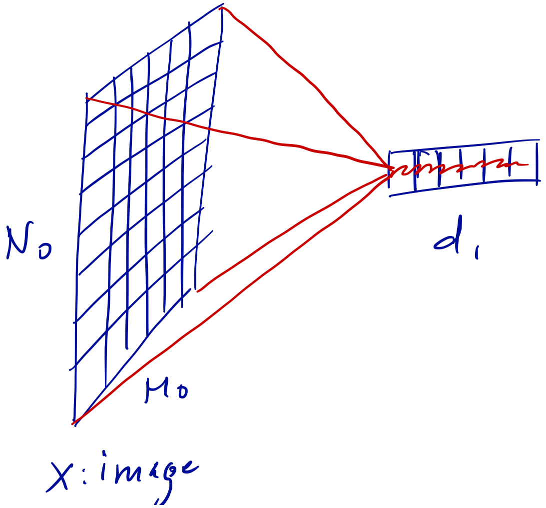


- What are some applications where these two images should lead to

1. Identical outputs
2. Different outputs

Computer Vision

Question

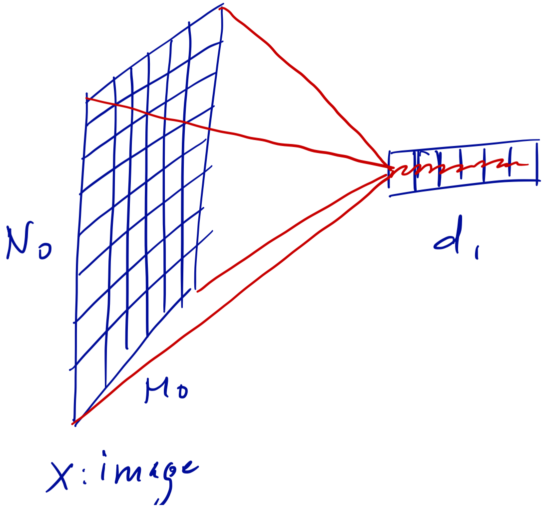


- What are some applications where these two images should lead to

1. Identical outputs
   - Recognition
   - Detection
2. Different outputs
   - Localization
   - Segmentation

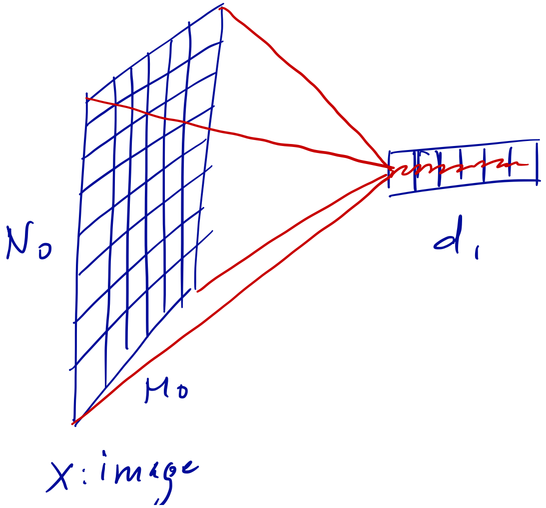
Computer Vision

Translation invariance is native to fully connected networks



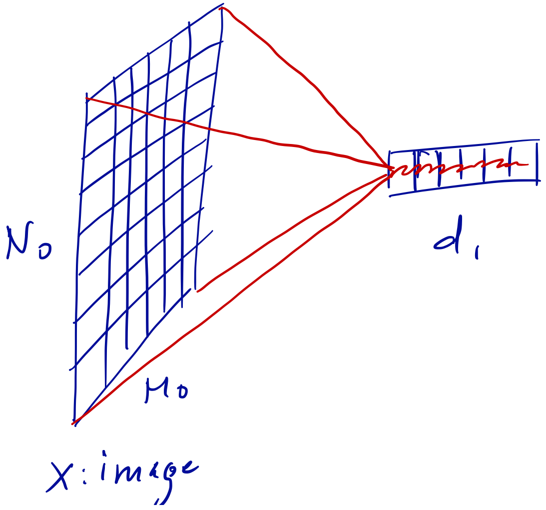
- These images will most likely lead to a very different activations in the hidden layer
- Rest of the network will see different activations
- In most vision applications, these images should lead to identical outputs
- It is possible to teach a fully connected network to be invariant to translations

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

Computer Vision

Translation invariance

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

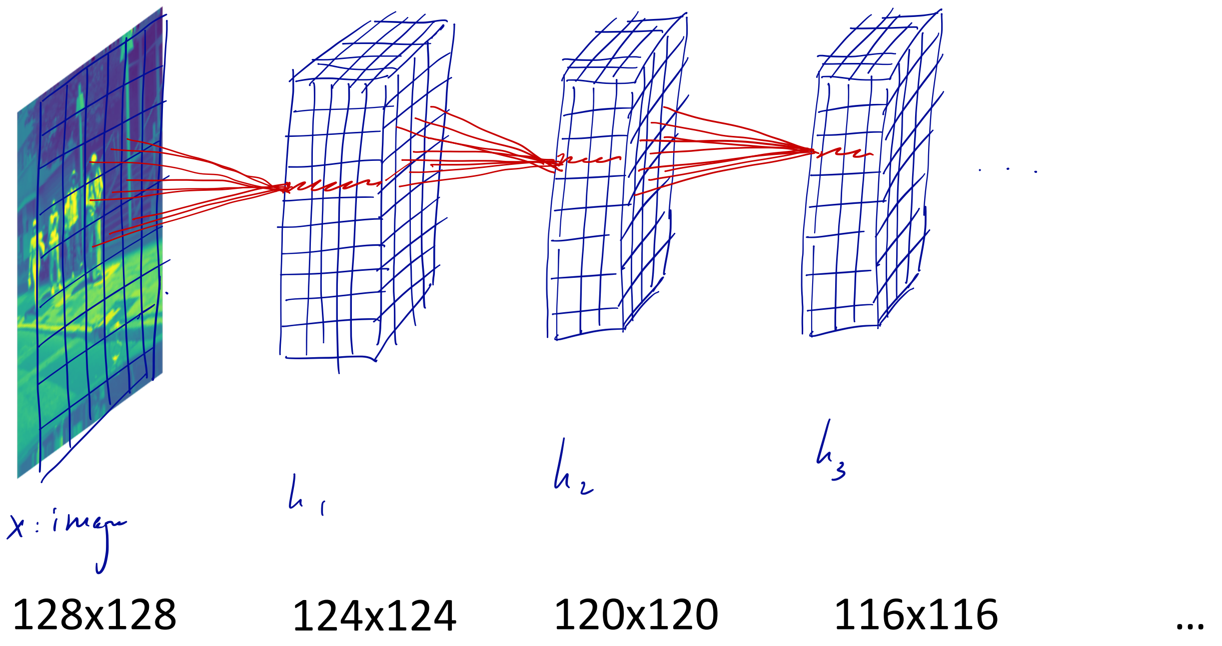
Convolutional layers is a great idea but

- 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

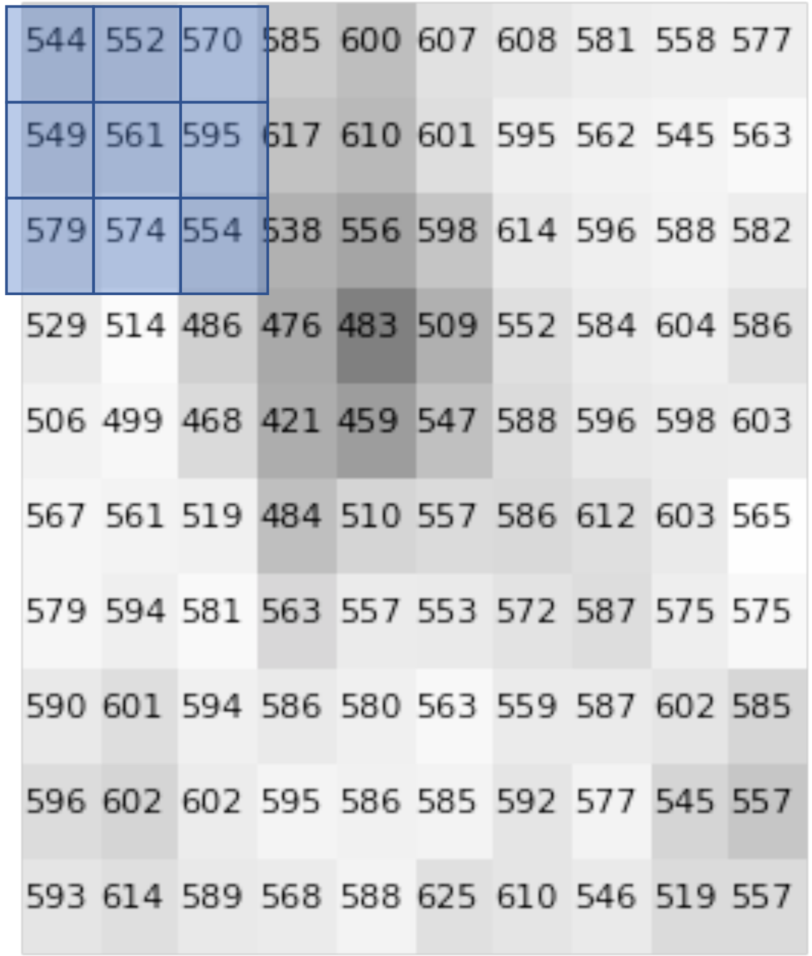
Strides

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

Computer Vision

Strides



- 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

Computer Vision

Strides

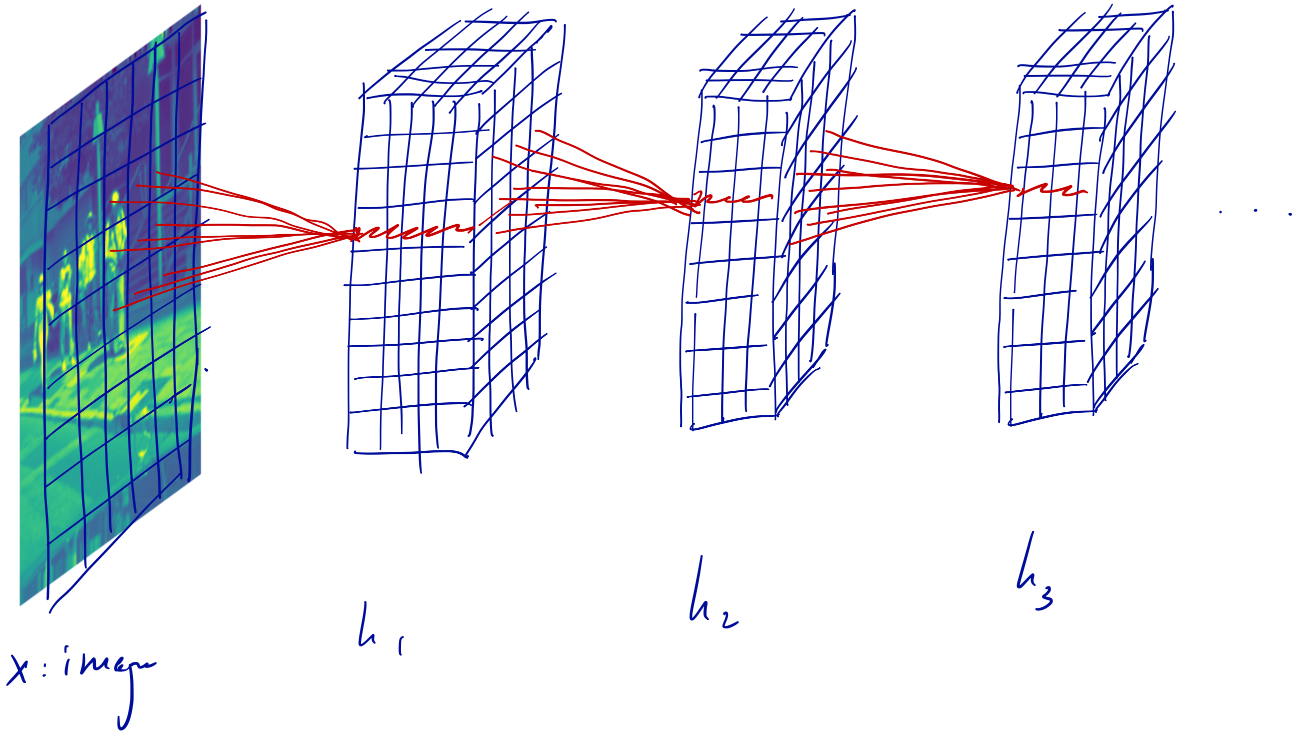
$$M_l = \left\lfloor \frac{M_{l-1} - k_1}{s_1} \right\rfloor + 1 \quad N_l = \left\lfloor \frac{N_{l-1} - k_2}{s_2} \right\rfloor + 1$$

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

Computer Vision

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]

Computer Vision

Recognition network with pooling

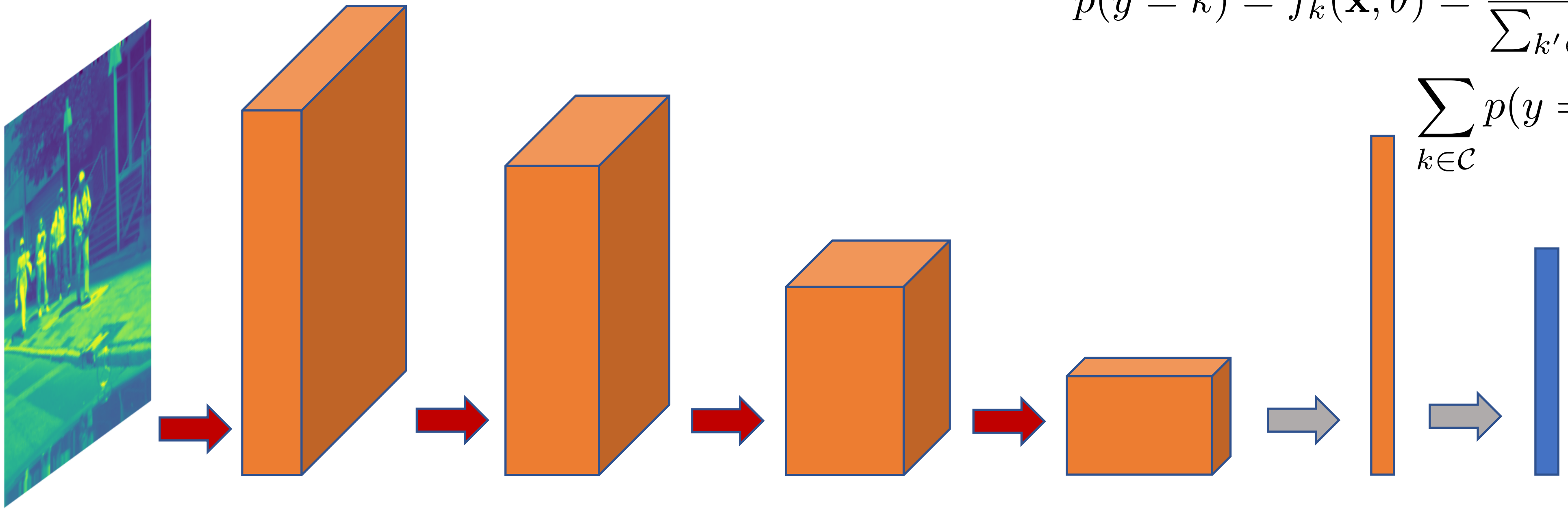
Computer Vision

Network architecture for a simple object recognition system

$$a_L = \mathbf{W}_L h_{L-1} + b_L \in \mathbb{R}^K$$

$$p(y=k) = f_k(\mathbf{x};\theta) = \frac{e^{a_{L,k}}}{\sum_{k' \in \mathcal{C}} e^{a_{L,k'}}}$$

$$\sum_{k \in \mathcal{C}} p(y=k) = 1$$



Convolution followed by non-linearity followed by max-pooling

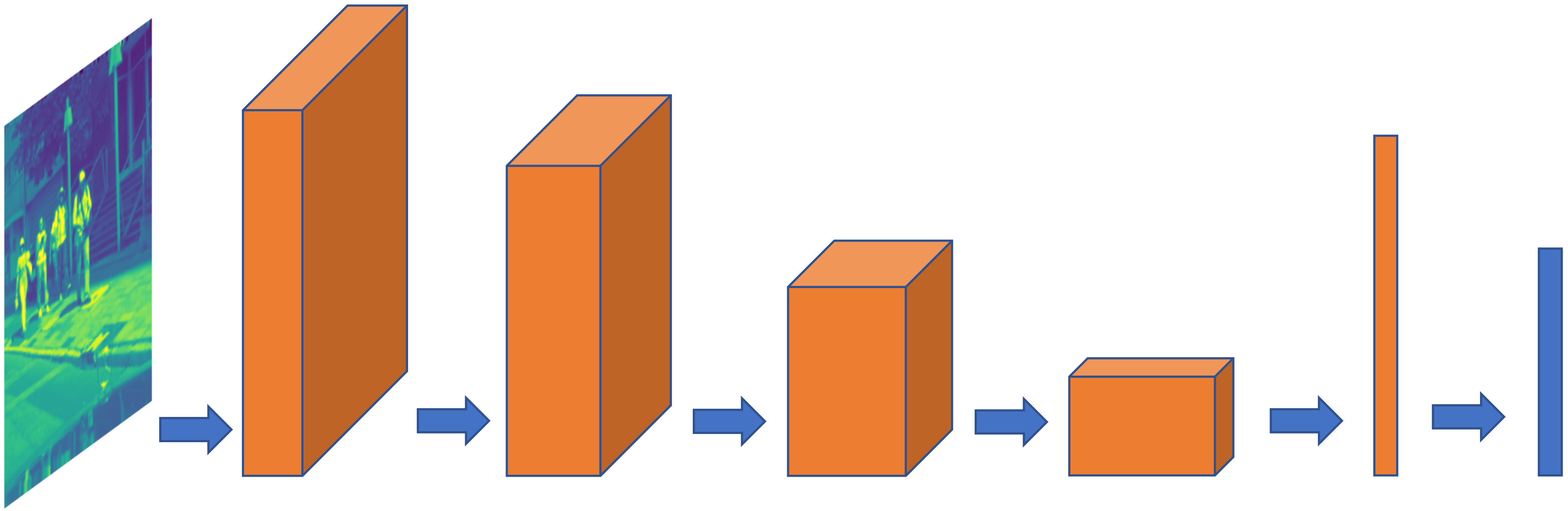
Fully connected layer: transformation followed by non-linearity

Output
Number of neurons
equal number of classes

Computer Vision

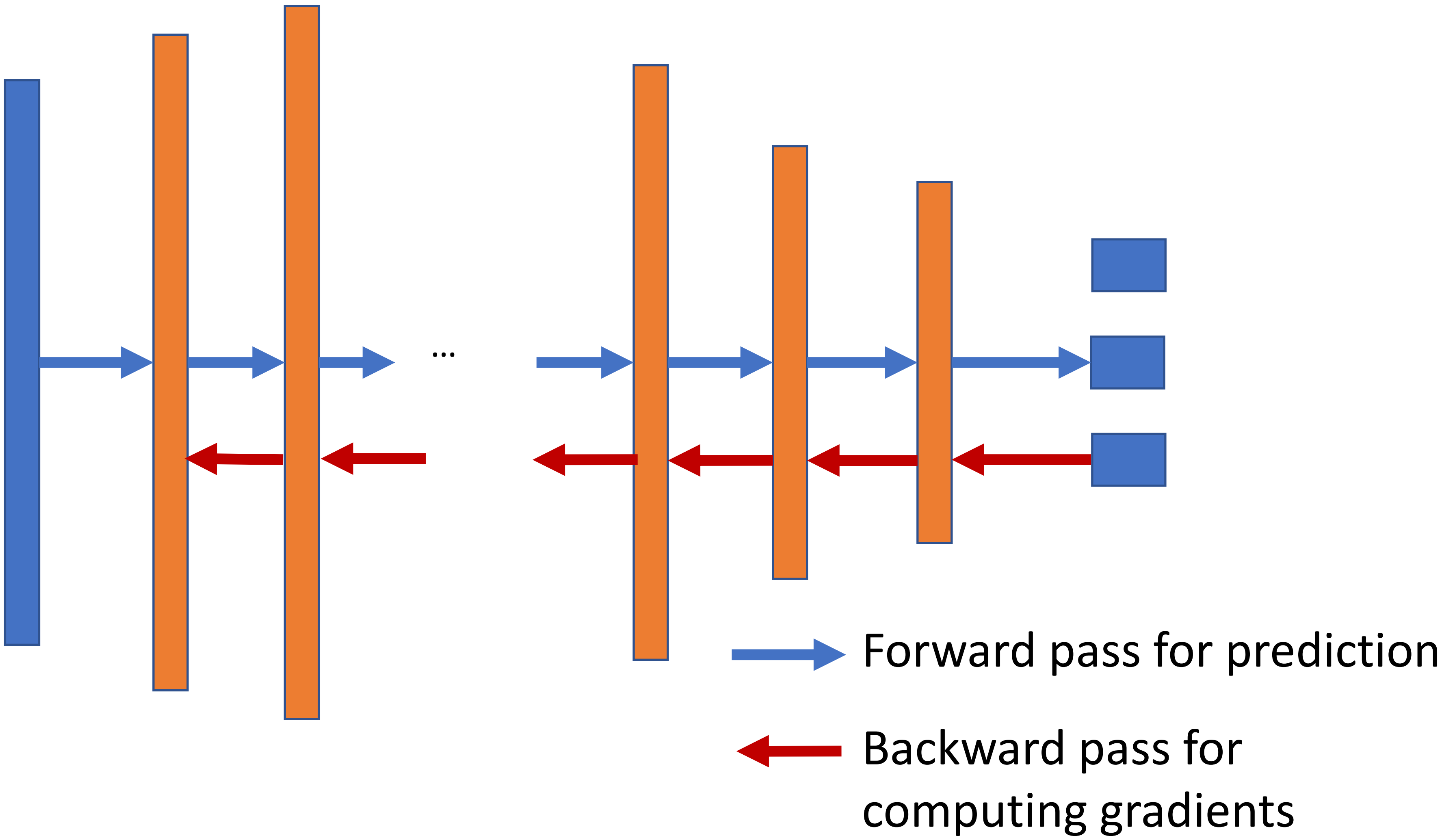
Putting it all together: Basic Convolutional neural network (CNN)

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



Computer Vision

Remember backpropagation



Computer Vision

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

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

Cost function

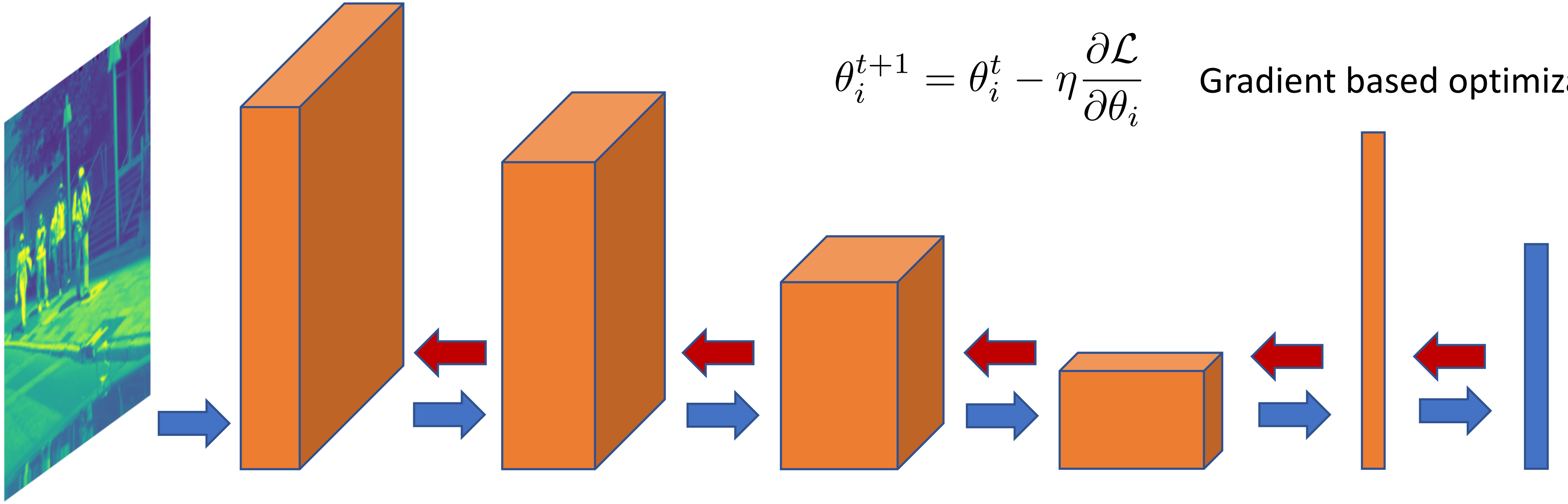
$$\mathcal{L}(y_n, f(\mathbf{x}_n;\theta)) = -\sum_{k \in \mathcal{C}} \log(f_k(\mathbf{x}_n;\theta))\mathbf{1}(y_n = k)$$

Optimization

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

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

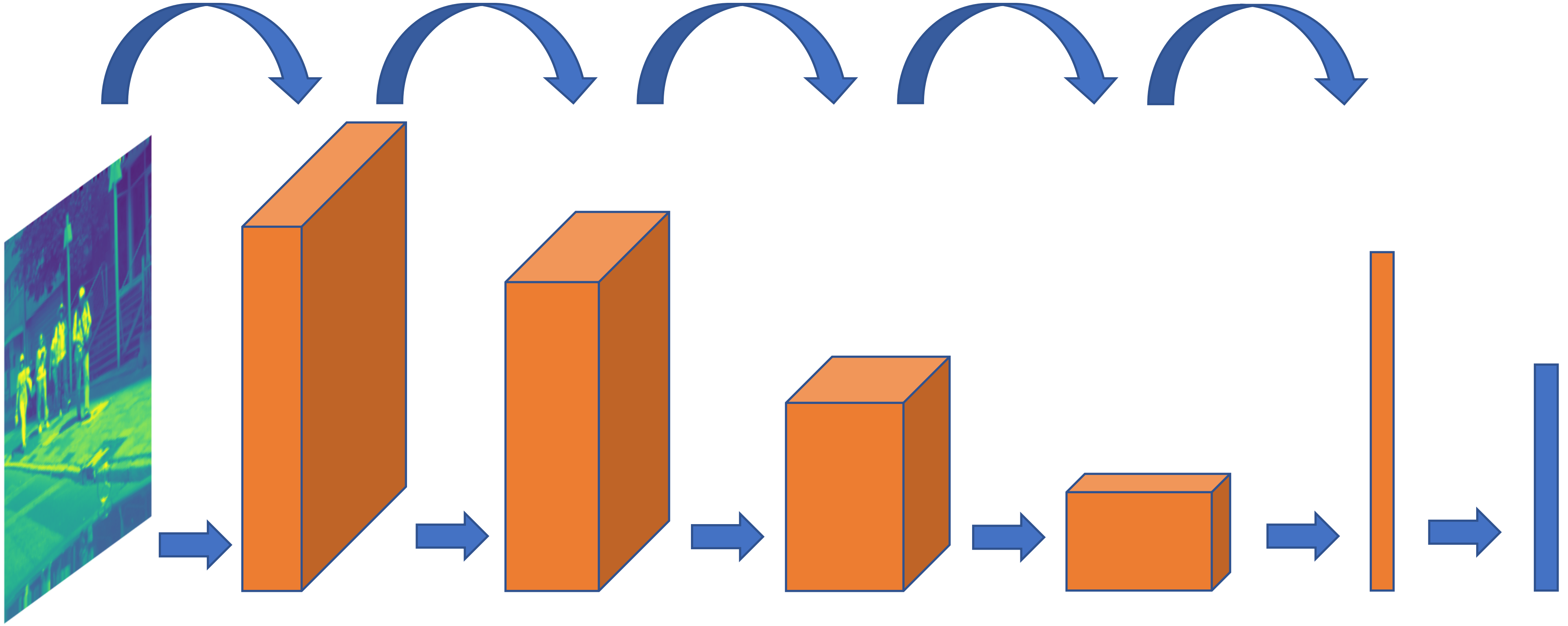
Gradient based optimization



Computer Vision

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



Computer Vision

Essential blocks lead to powerful algorithms

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

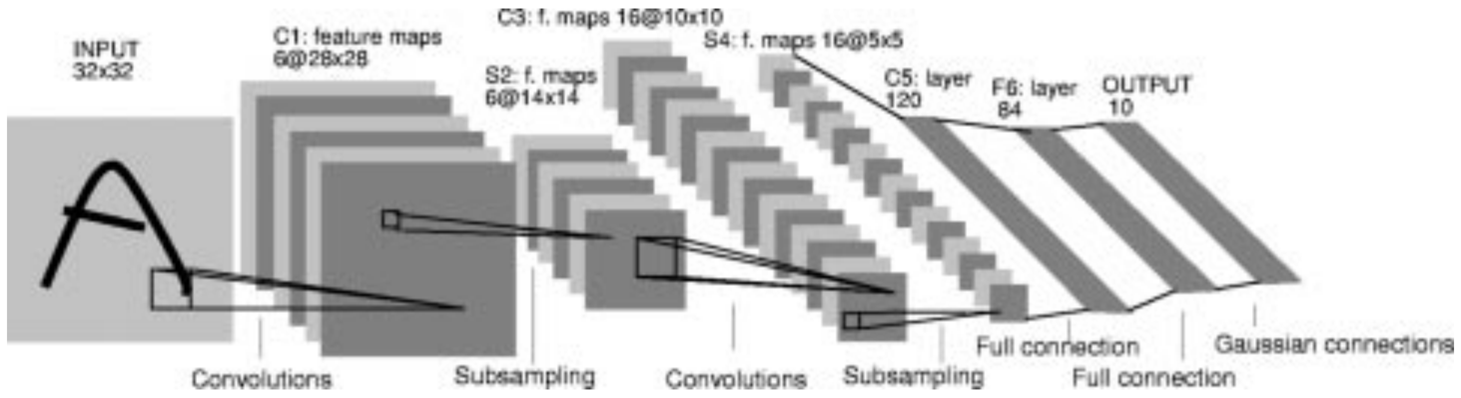


Fig. 2. Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.

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

- Then silence for a long time

Computer Vision

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

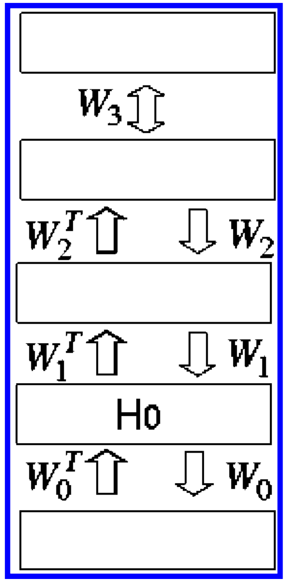
Computer Vision

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

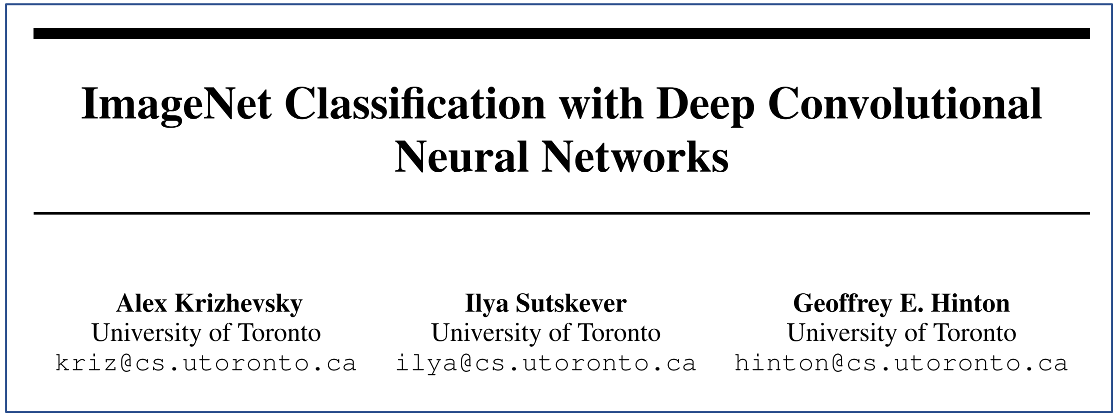
Yee-Whye Teh
tehyw@comp.nus.edu.sg
Department of Computer Science, National University of Singapore, Singapore 117543



- 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

Computer Vision

Then in 2012

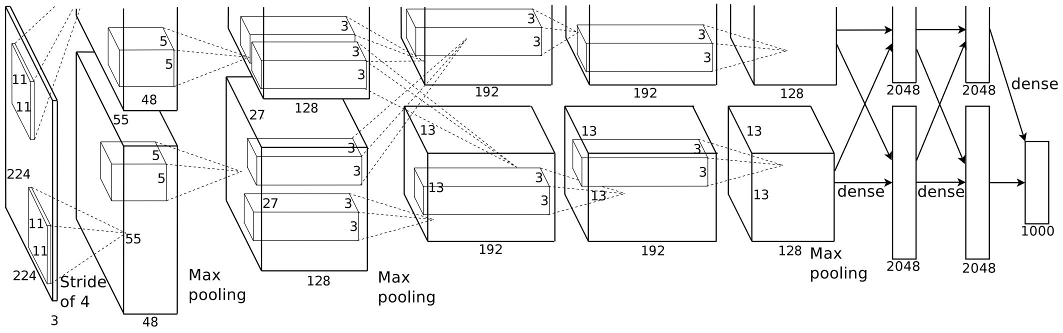

ImageNet Classification with Deep Convolutional Neural Networks

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

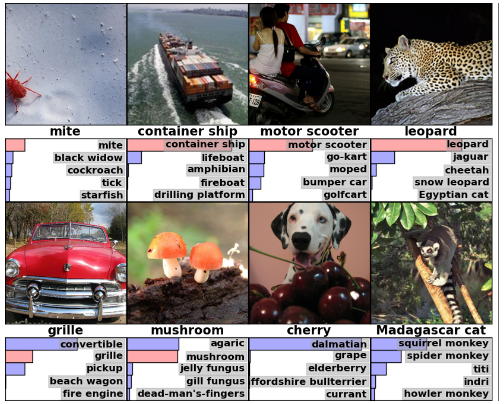
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

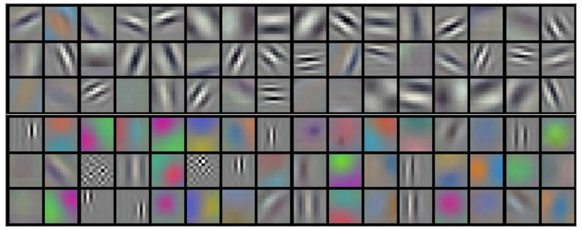
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

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



Network





First layer filters 11x11

Computer Vision

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

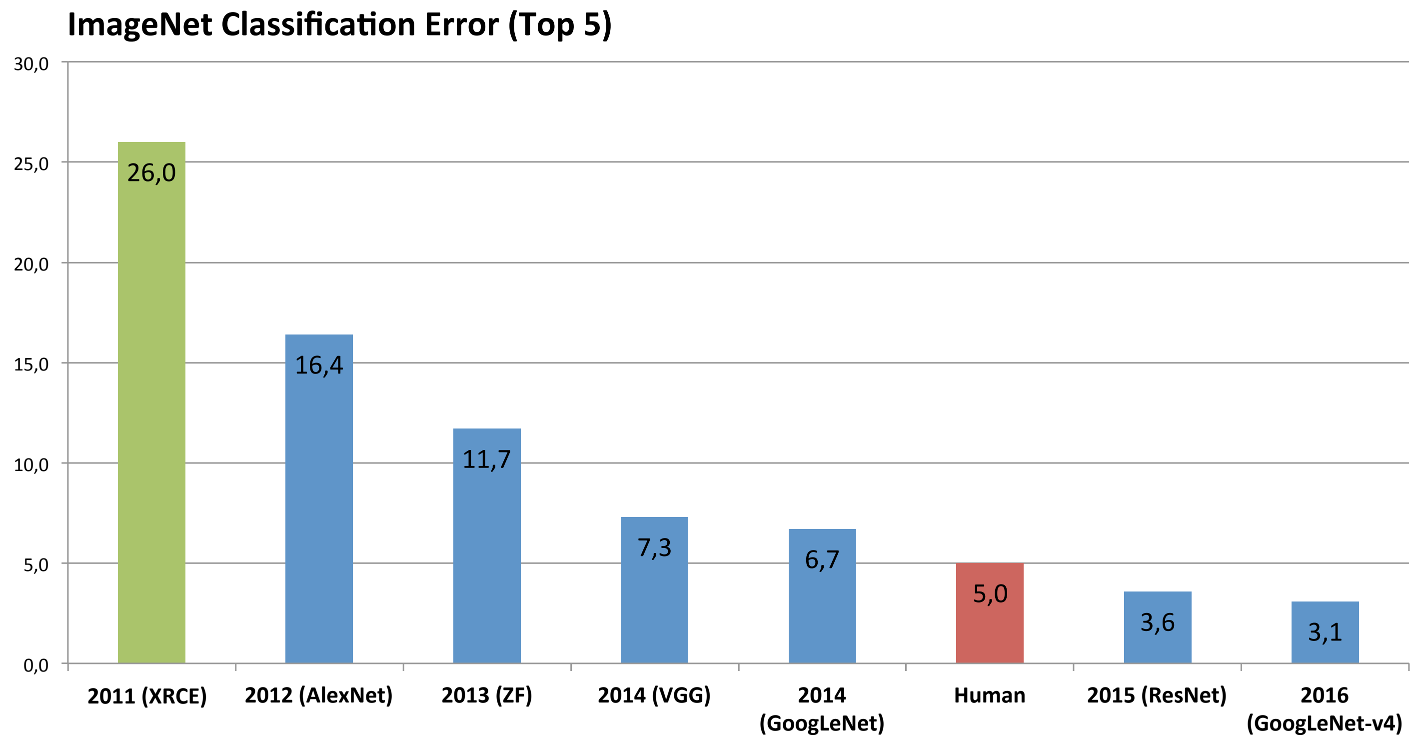
Authors   Authors and affiliations

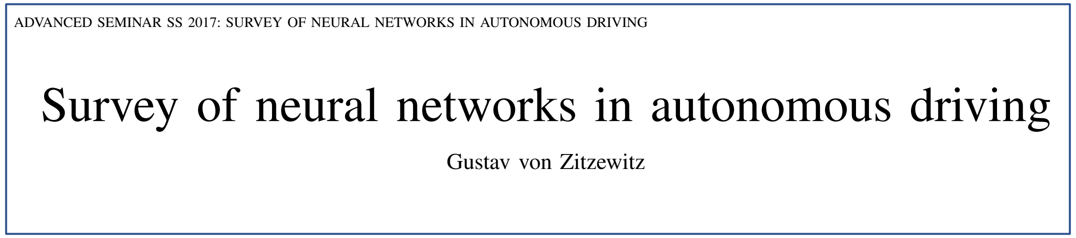
Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei

- 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

Computer Vision

Historical evolution of the ImageNet Challenge



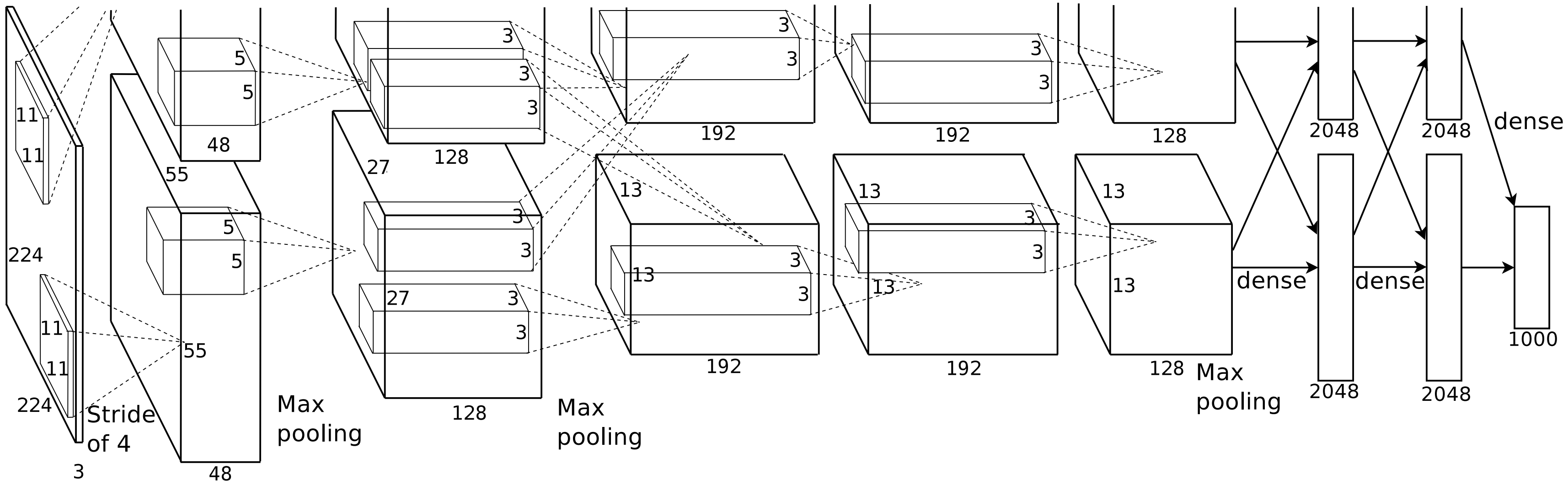

ADVANCED SEMINAR SS 2017: SURVEY OF NEURAL NETWORKS IN AUTONOMOUS DRIVING

Survey of neural networks in autonomous driving

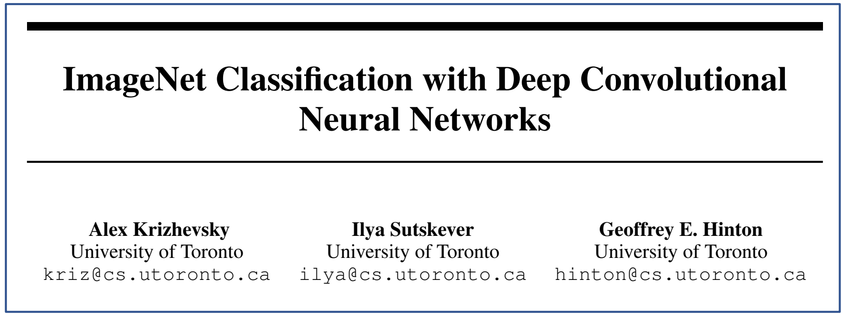
Gustav von Zitzewitz

Computer Vision

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.

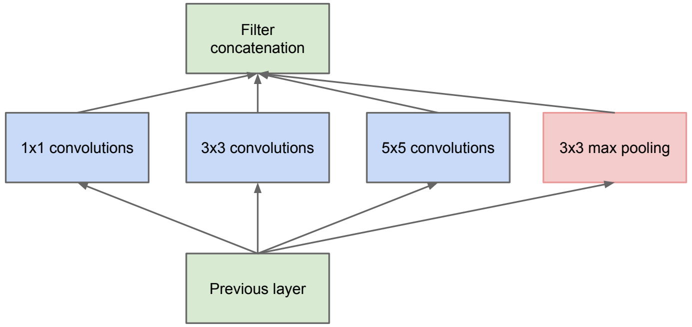
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	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 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

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

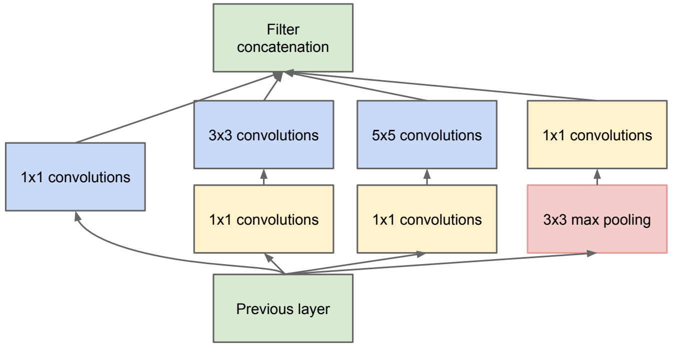
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Inception - 2014

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

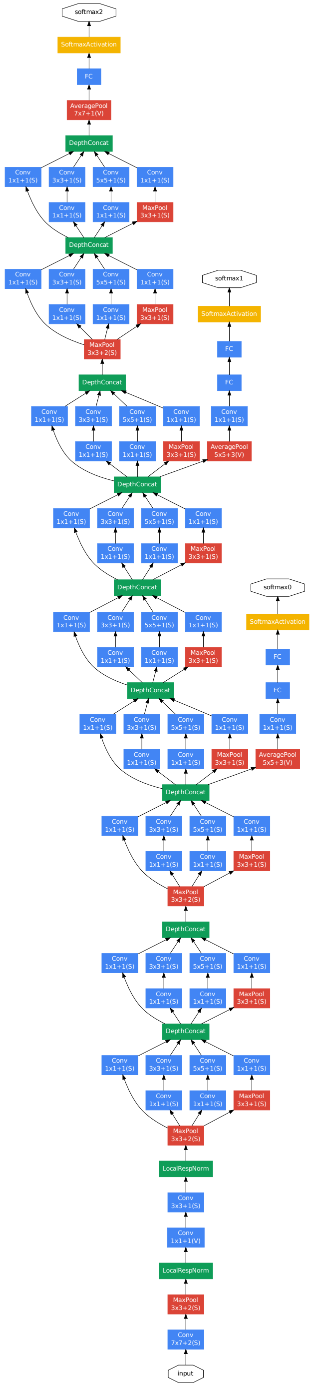


(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2: Inception module



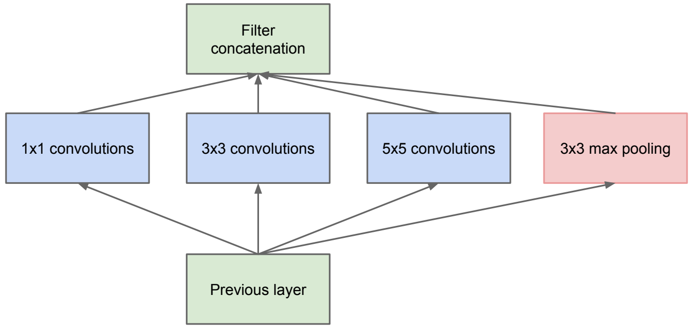
Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.

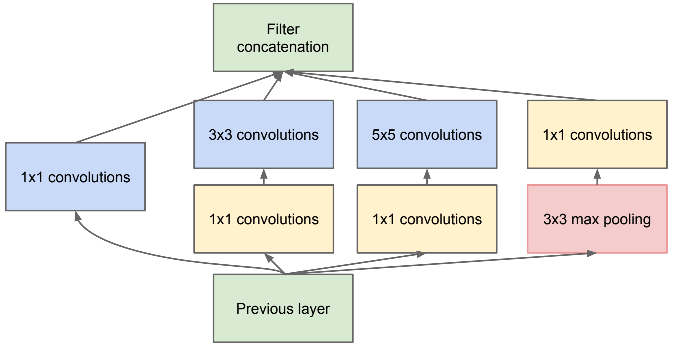
[CVPR 2015]

Computer Vision

More on the inception module



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2: Inception module

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]

[1]Google Inc. [2]University of North Carolina, Chapel Hill

[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.

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

- 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

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

Computer Vision

ResNet - 2015

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

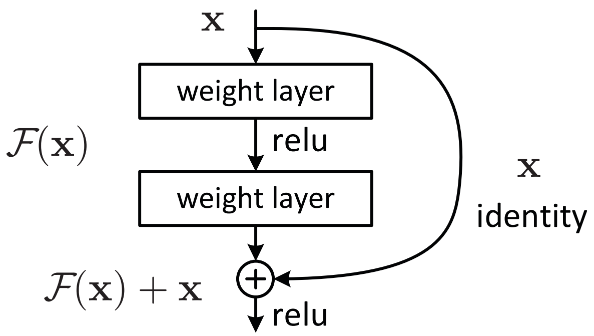
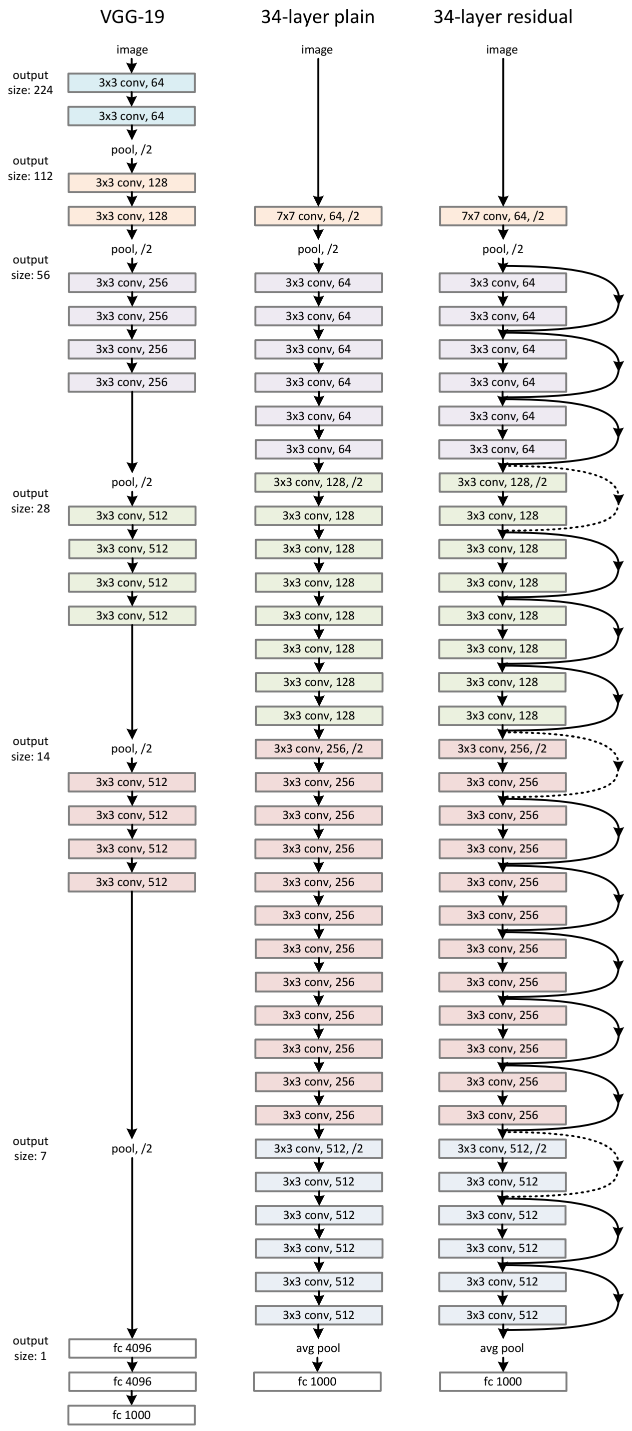


Figure 2. Residual learning: a building block.

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]



Computer Vision

More on Residual units and ResNet

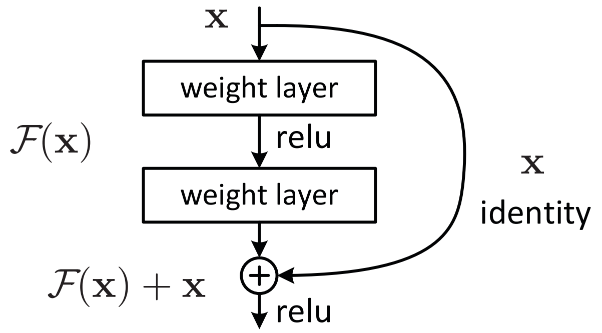


Figure 2. Residual learning: a building block.

- Conventional layer

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

difficult to model an identity function

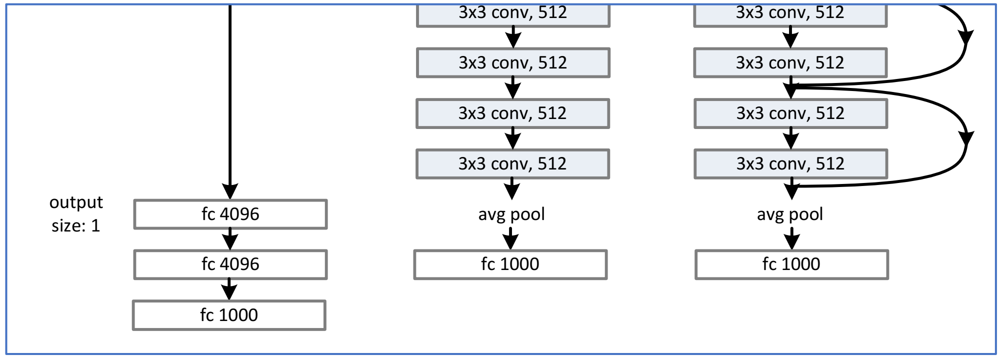
- Residual layer

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

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



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
  - Drop-out
  - Regularization
  - Data augmentation
  - Transfer learning

Number of parameters in a network

- Let us assume the following network

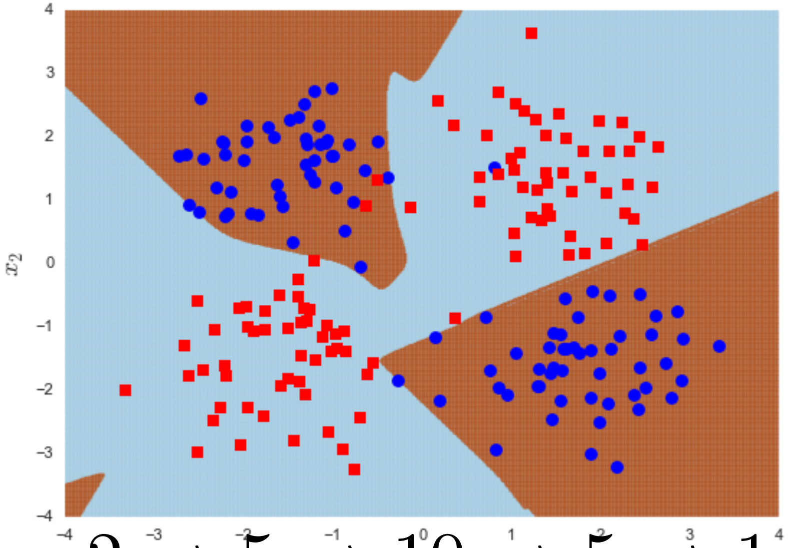
$$\underbrace{10}_{\text{Input dimesion}} \rightarrow \underbrace{20 \rightarrow 30 \rightarrow 10}_{\text{Hidden dimensions}} \rightarrow \underbrace{1}_{\text{Output dimension}}$$

- Total number of parameters:
  10x20 + 20 + 20x30 + 30 + 30x10 + 10 + 10 = 1170
- To determine the large number of parameters we need large number of samples
- Modern networks have many number of parameter, reaching millions

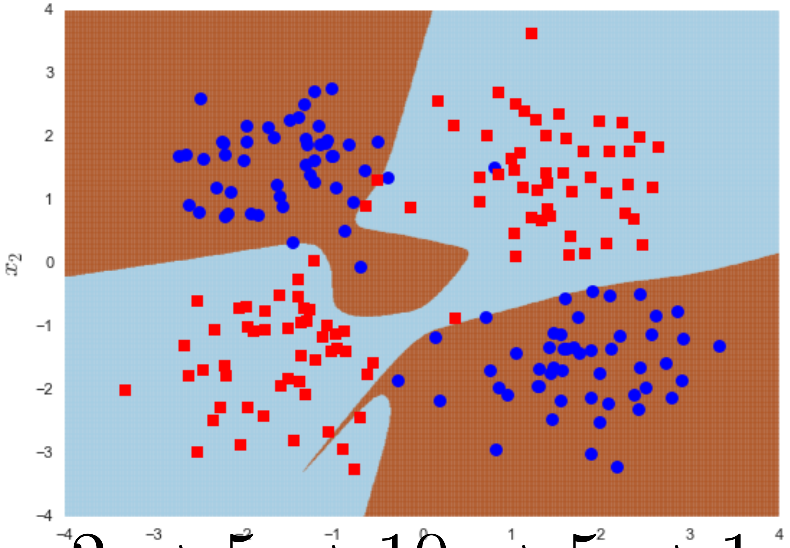
Over-fitting

- When the model has too many parameters and not enough training samples
- Model can learn noise in the training samples
- Perfect prediction on training data
- Bad prediction in validation data
- Will not be able to generalize

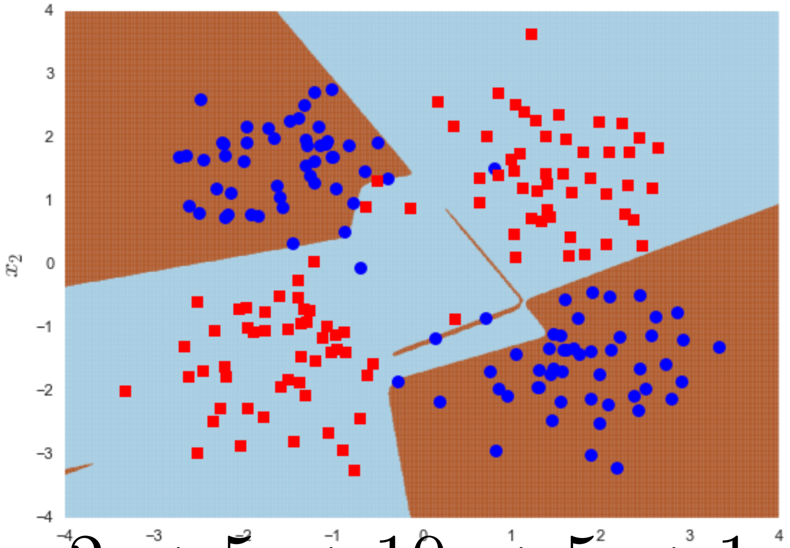
Over-fitting – examples on the toy data



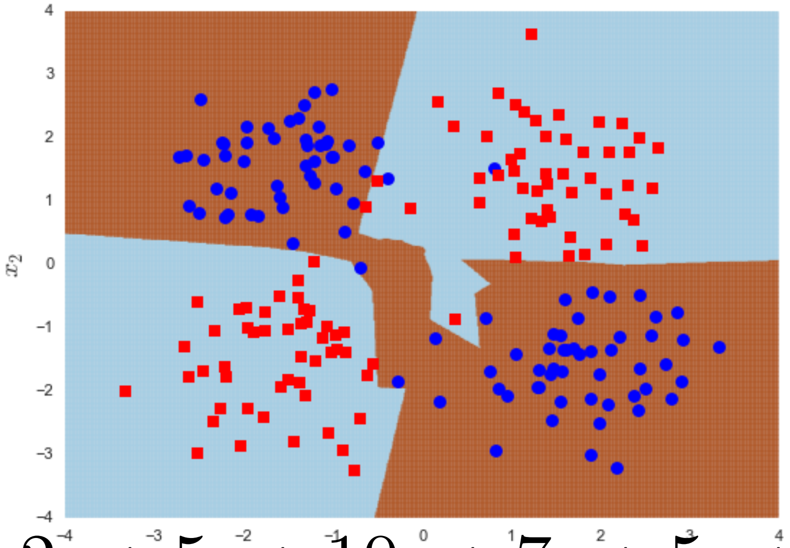
$2 \rightarrow 5 \rightarrow 10 \rightarrow 5 \rightarrow 1$



$2 \rightarrow 5 \rightarrow 10 \rightarrow 5 \rightarrow 1$



$2 \rightarrow 5 \rightarrow 10 \rightarrow 5 \rightarrow 1$



$2 \rightarrow 5 \rightarrow 10 \rightarrow 7 \rightarrow 5 \rightarrow 1$

Over-fitting – how to overcome it?

Two most obvious strategies:

1. Best strategy is to have **more data**
   a) Most reliable strategy
   b) Data collection can be expensive or not even feasible
   c) Labeling can be expensive
2. Reduce the size of your network, i.e. less parameters
   a) Also reliable strategy
   b) It may lead to performance loss

Other strategies

1. Best strategy is to have **more data**
2. Reduce the size of your network, i.e. less parameters
3. Regularization

Computer Vision

Regularization

- Regularization [Krogh and Hertz, NIPS 1992]
- Similar strategy is taken for many other learning algorithms, ridge regression, SVM, sparse regression,…

$$\min \mathcal{L} + \lambda \mathcal{R}\left(\theta\right)$$

$$\mathcal{R}(\theta) = \frac{1}{2}\sum \theta_i^2$$

Weight decay
L_2 regularization

$$\mathcal{R}(\theta) = \sum |\theta_i|$$

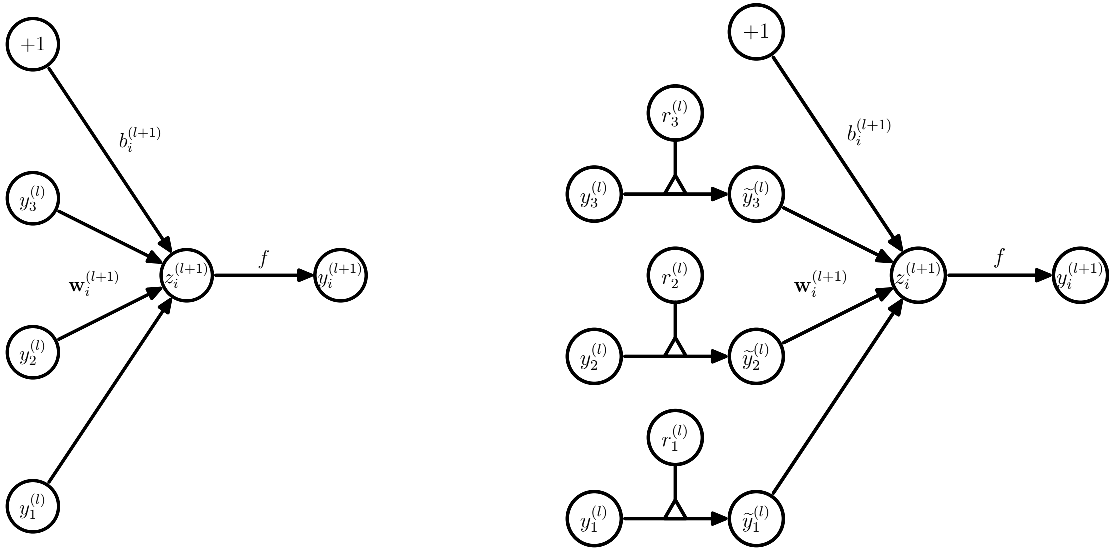
Sparse weights
L_1 regularization

Other strategies

1. Best strategy is to have **more data**
2. Reduce the size of your network, i.e. less parameters
3. Regularization
4. Drop-out

Drop-out

- Drop-out [Hinton et al. 2012, Srivastava, Hinton, Krizhevsky, JMLR 2014] [Figure from latter]
- Randomly set some of the activations to zero during training
- At test time run it will all the activations on.



(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

Computer Vision

Drop-out

- [Figure from Srivastava, Hinton, Krizhevsky, JMLR 2014]
- Network builds redundancies, need to create multiple paths
- Effectively smaller networks than constructed

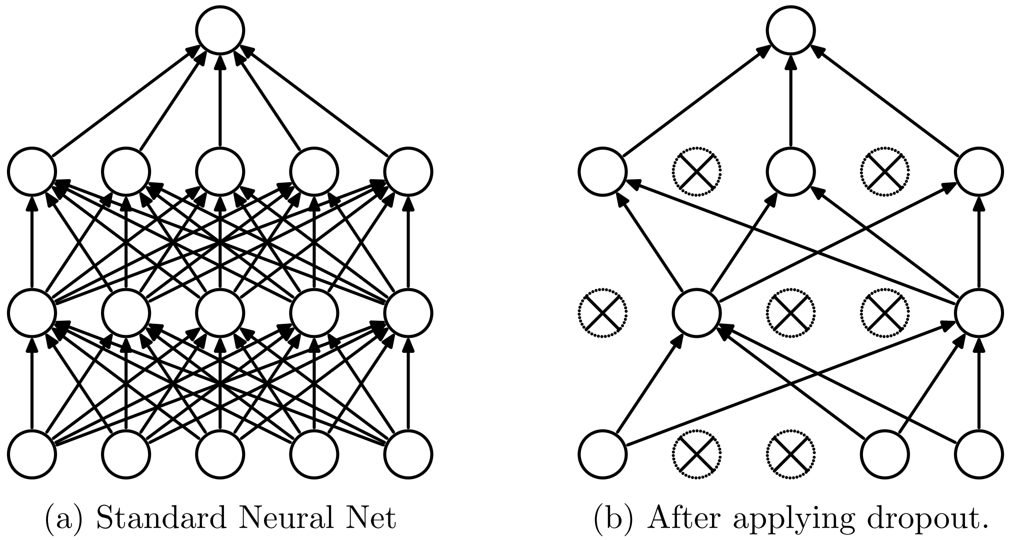


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Other strategies

1. Best strategy is to have **more data**
2. Reduce the size of your network, i.e. less parameters
3. Regularization
4. Drop-out
5. Multi-task learning

Multi-task learning

- There may be more than one task for the same dataset
  - Recognition of objects + distinguishing indoor from outdoor scenes
  - Segmentation + denoising of images
- Minimization of two loss functions simultaneously

$$\mathcal{L}(\theta) = \lambda_1 \mathcal{L}_1(\theta) + \lambda_2 \mathcal{L}_2(\theta)$$

- Note the similarity to regularization

$$\mathcal{L}(\theta) + \lambda \mathcal{R}(\theta)$$

Other strategies

1. Best strategy is to have **more data**
2. Reduce the size of your network, i.e. less parameters
3. Regularization
4. Drop-out
5. Multi-task learning
6. Data augmentation

Computer Vision

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

Other strategies

1. Best strategy is to have **more data**
2. Reduce the size of your network, i.e. less parameters
3. Regularization
4. Drop-out
5. Multi-task learning
6. Data augmentation
7. Transfer learning

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
- Initial task can also be unsupervised – for instance denoising with added noise.

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 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	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 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
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