Deep Learning for Augmented Reality

Vincent Lepetit

examples of application of Deep Learning to Augmented Reality problems

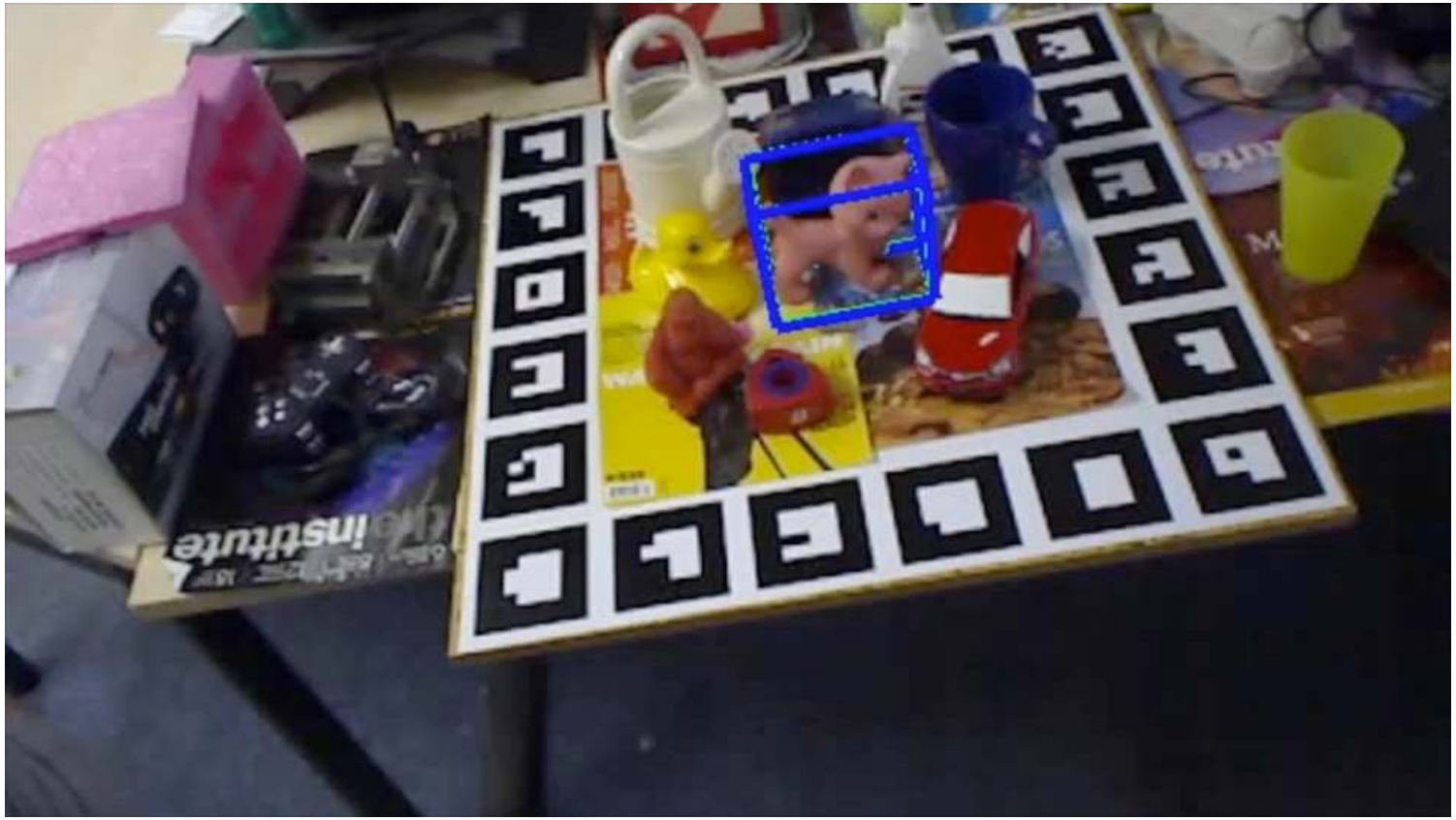
We show qualitative results on the DAVIS dataset.

Images were processed individually frame-by-frame.
No temporal information was used in any way.

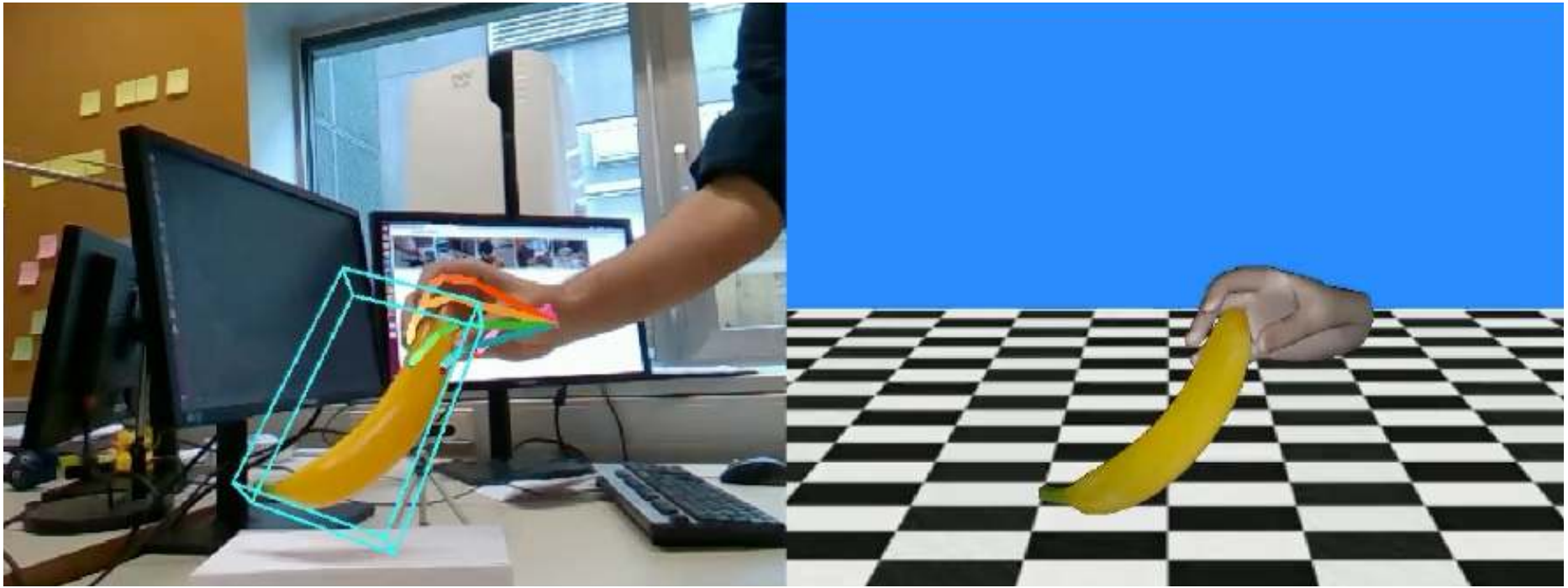
This is zero-shot cross-dataset transfer.
The DAVIS dataset was never seen during training.

Standard AR

Applications

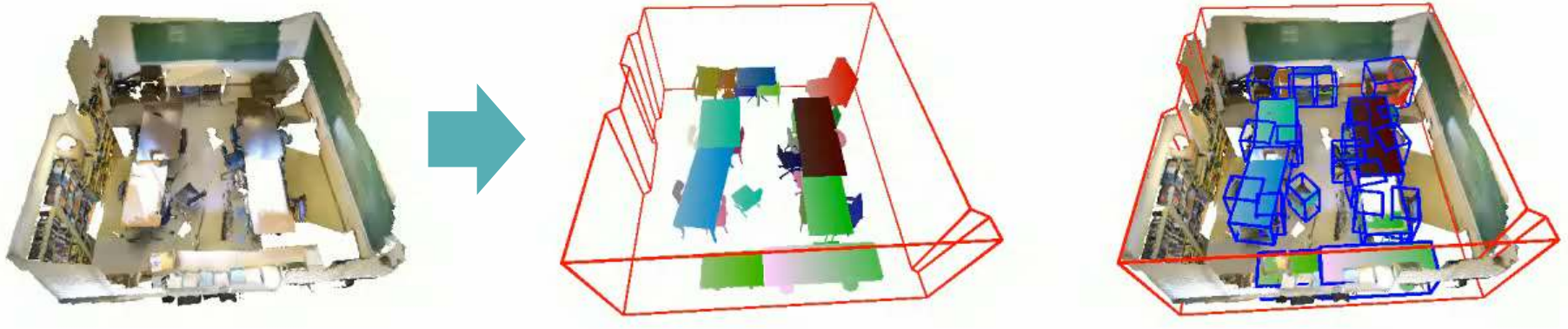


Applications



3D pose estimation from a single image [Machine Learning]

3D Scene Understanding



3D Scene Understanding: an application to AR

Image Retrieval

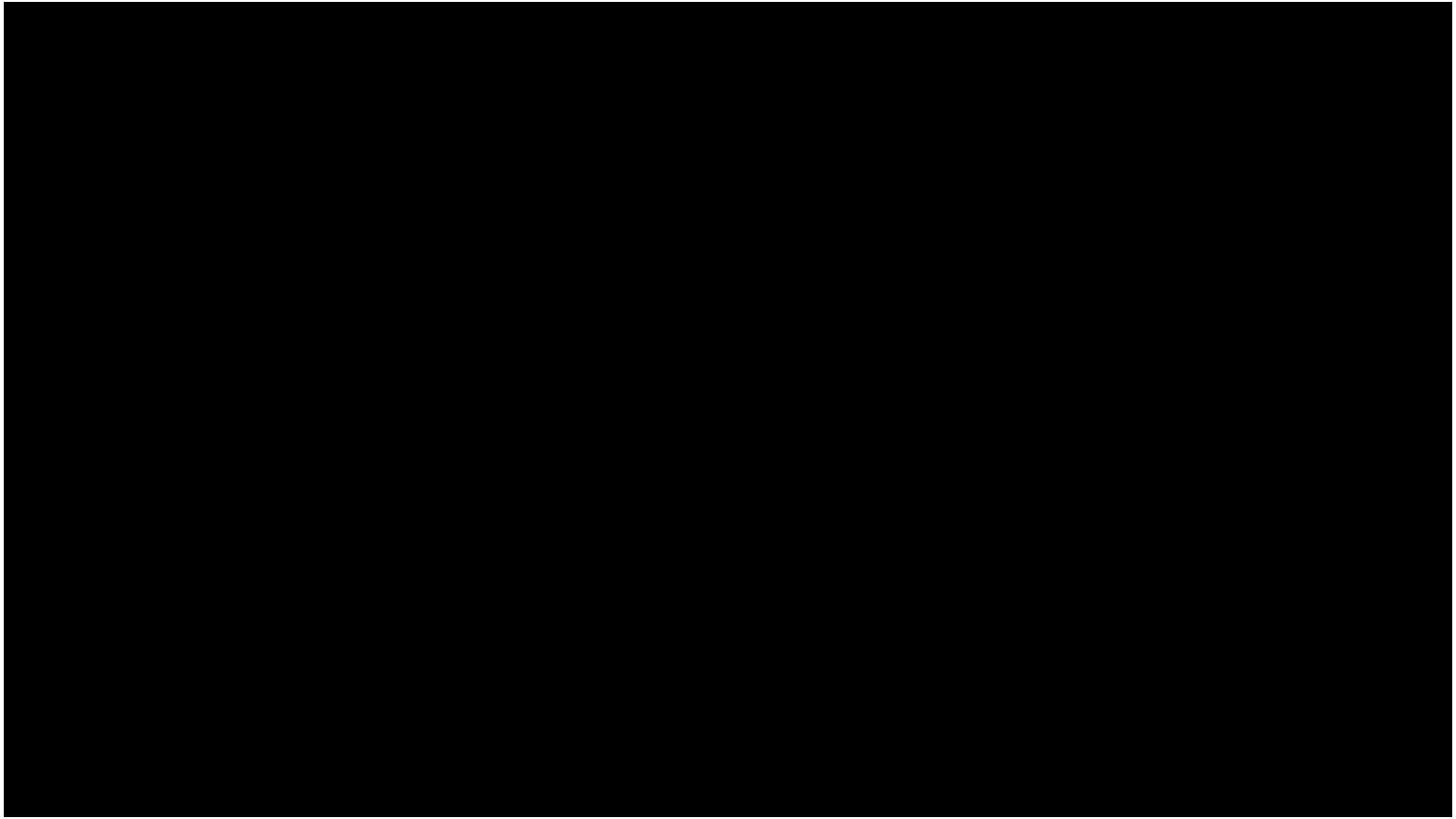


Image-based localization

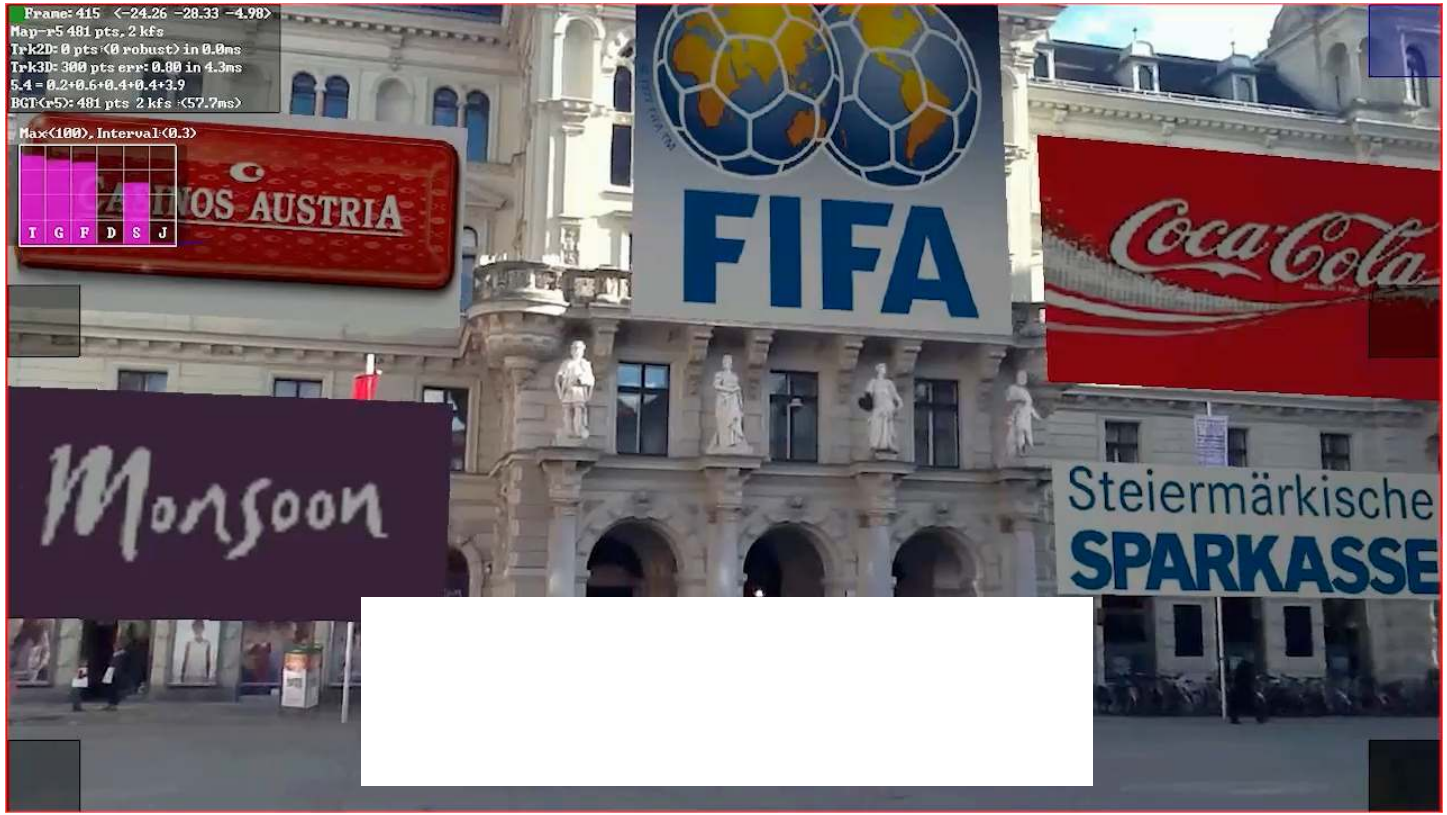
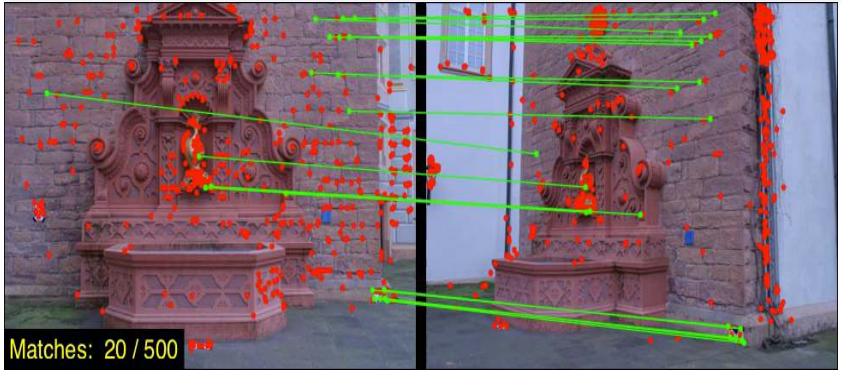


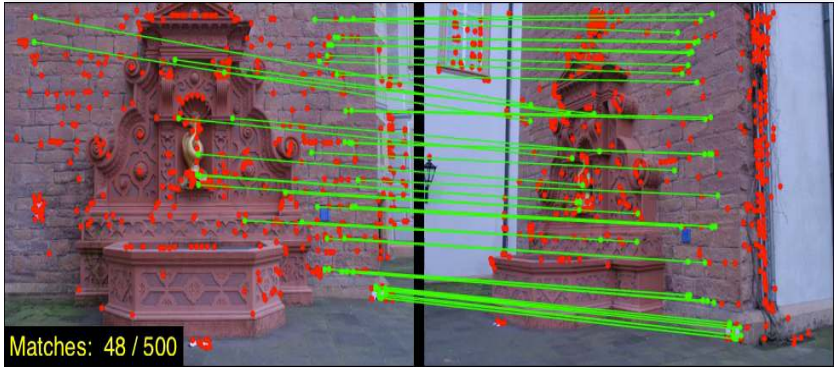
Image Matching

SIFT. Average: 60.2 matches



+64%

LIFT (Ours). Average: 98.6 matches



This Week

- Introduction to “general” deep learning;
- Supervised and self-supervised monocular depth prediction;
- Monocular 3D pose estimation of objects and hands and scene understanding from images;
- 3D model prediction (and transformers);
- Point cloud analysis;
- Keypoint detection and matching.
- 1 exercice on 3d pose estimation.

vision is hard!

why is it so and how
Deep Learning can help?

Why Is Vision So Hard?

Too much information:

A color image of resolution 1000 x 1000 is made of 1000 x 1000 x 3 x 8 = 2.4 10^7 bits.

123	034	089	045	145	178	009	078	044	084	245	190	066	008	055	094	046	098
045	145	178	009	078	066	008	055	123	034	089	059	044	084	245	066	008	055
034	089	045	145	123	034	089	094	046	098	123	034	089	178	009	078	034	009
084	245	190	044	084	055	094	084	245	190	078	044	084	044	084	245	190	123
123	034	089	078	044	084	055	094	046	123	034	089	009	078	044	084	143	162
033	178	055	094	046	098	145	178	009	078	044	084	123	034	089	045	145	178
084	245	190	044	084	055	094	046	098	009	078	044	084	078	123	034	089	056
066	008	055	009	078	009	078	044	034	089	045	145	178	078	044	066	008	055
012	034	089	045	145	178	098	078	123	034	089	034	089	045	145	178	067	034
098	084	245	190	178	009	078	044	084	044	084	245	190	044	084	084	245	190
055	094	046	098	034	089	045	145	178	084	009	078	044	084	245	190	201	206
190	156	123	034	089	009	078	034	089	045	145	123	034	089	009	078	044	084
018	055	094	046	098	078	044	084	034	089	045	044	084	245	190	009	078	075
234	084	245	190	078	044	084	245	190	055	094	046	098	078	044	123	034	089
157	044	084	245	190	046	098	123	034	089	078	044	084	044	084	245	190	012
066	008	055	084	245	190	034	089	045	145	178	009	078	044	044	084	245	190
084	245	190	044	034	089	045	145	178	009	044	084	245	190	044	084	089	043
044	084	245	190	178	009	078	055	044	084	245	190	034	089	044	084	245	190

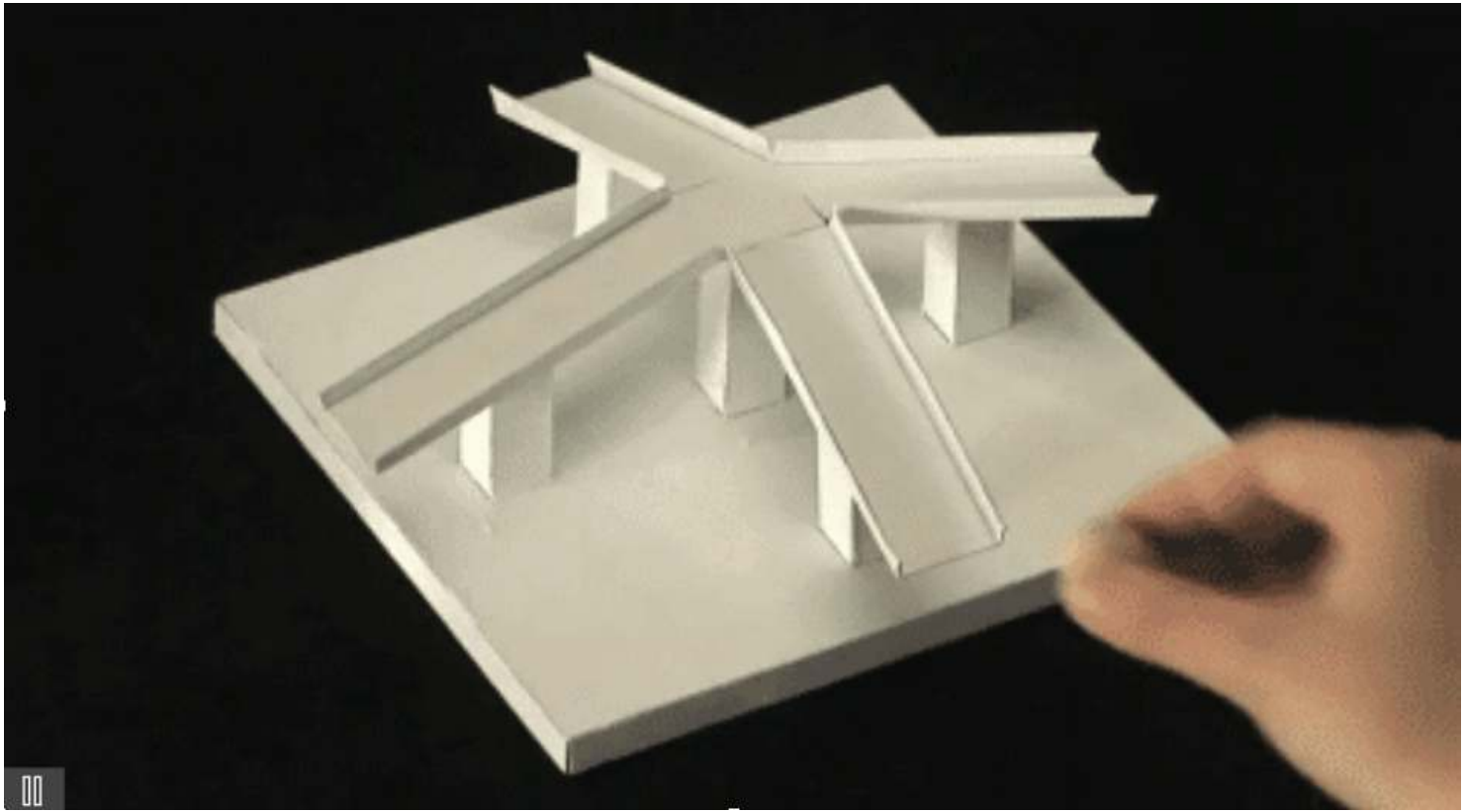
a small image

Why Is Vision So Hard?

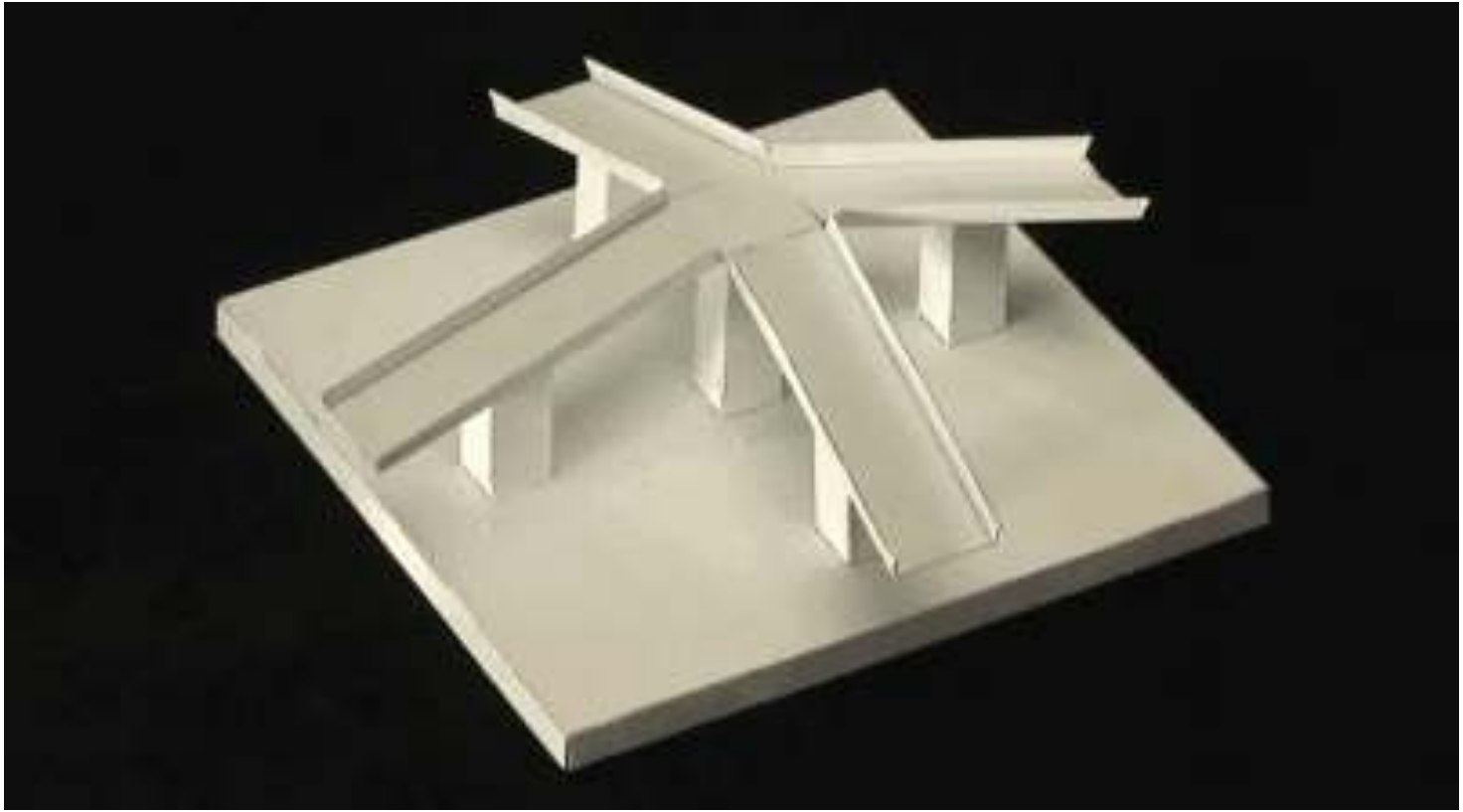
Not enough information

Why Is Vision so Hard?

Why Is Vision so Hard?

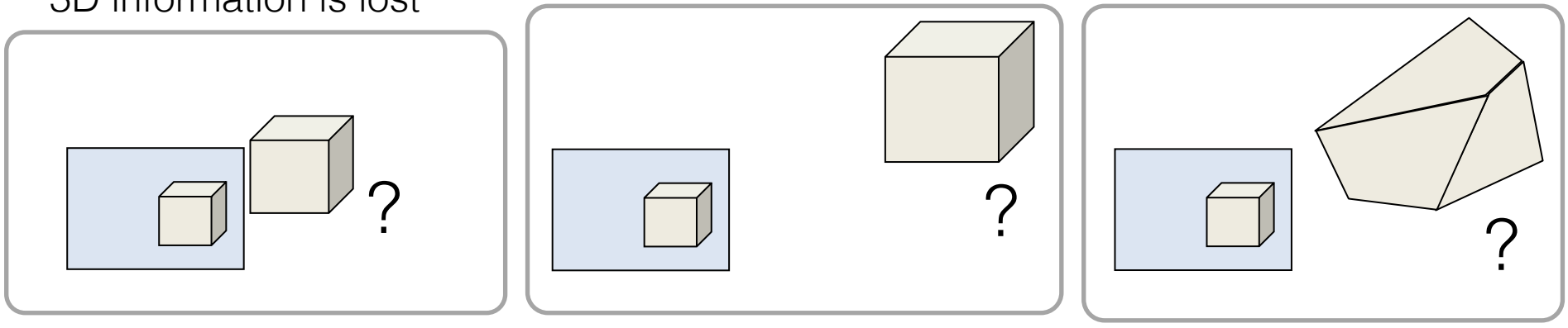


Why Is Vision so Hard?



Why Is Vision So Hard?

Not enough information
3D information is lost



Given an image, there is an infinite family of 3D scenes that could create this image;
How do we should which one is correct?

- Context (e.g. car washing video);
- Prior information (e.g. lines tend to be orthogonal or parallel);
- etc.

3D Perception in Humans

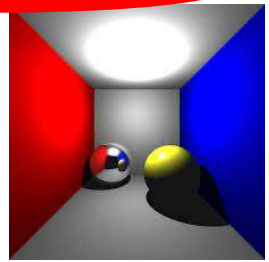
monocular cues

from prior knowledge

from context

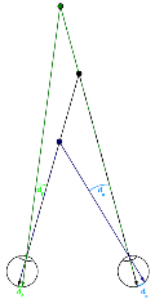
from motion

from focus and defocus

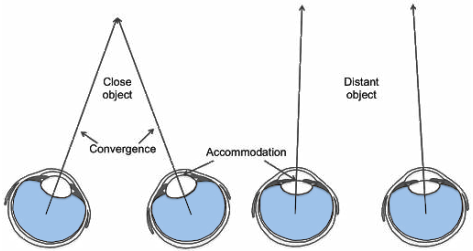


from shading and cast shadows

from atmospheric effects



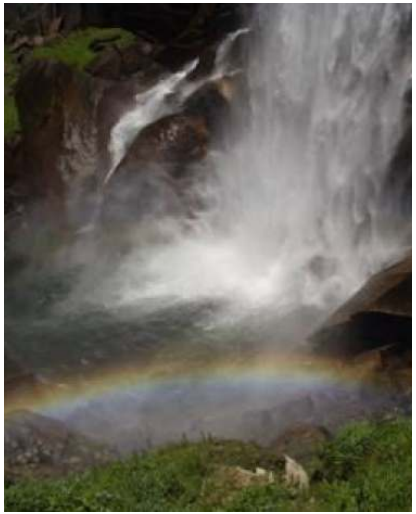
from binocular parallax



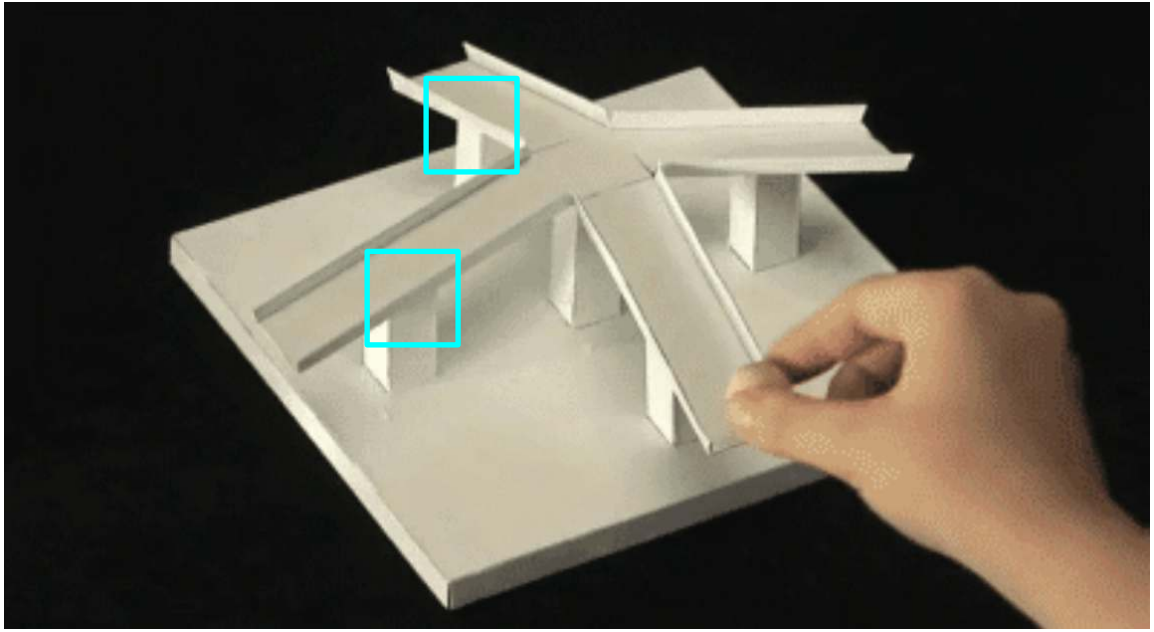
from convergence

binocular cues

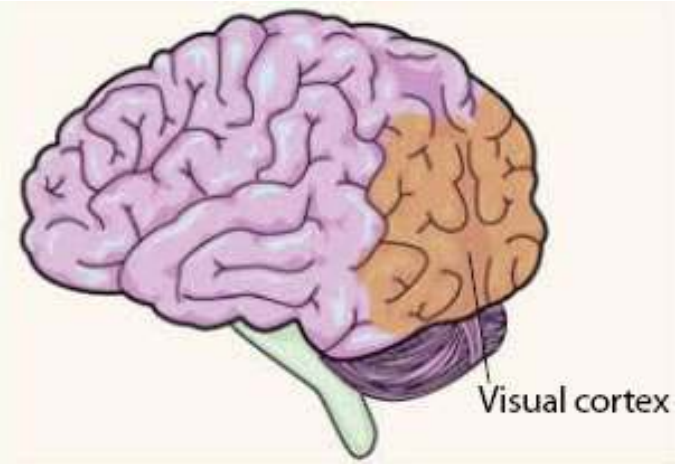
Complex Light Effects



contour “illusions”



Human Vision

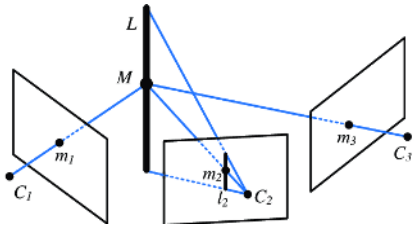


The visual cortex represents about 20-50% of our brain.

Human vision is unconscious (most of the time).

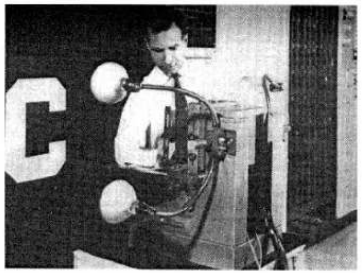
Intuitions about how human vision works are often wrong...

A Short History of Computer Vision



~2000: beginning of Machine Learning being used in Computer Vision

~1958: Perceptron



~1992: first Deep Learning systems in Computer Vision

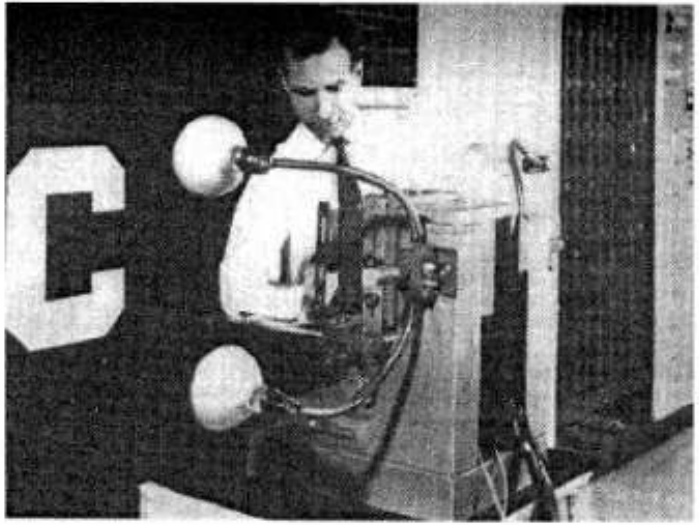
1980s-:

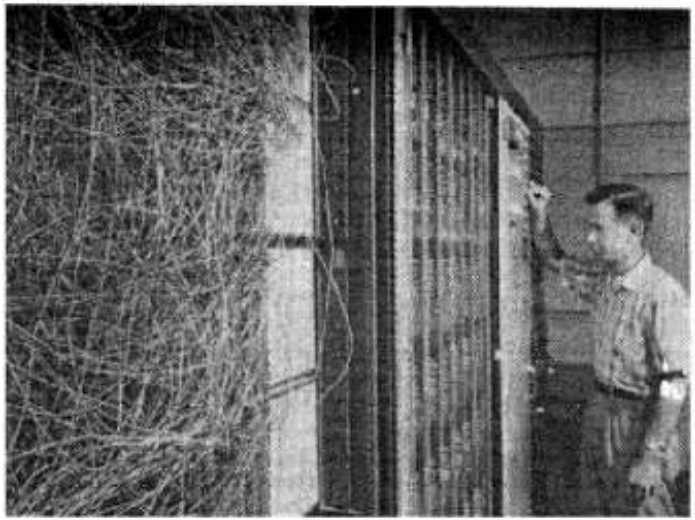
- low-level Computer Vision (edge, segment, keypoint detection, etc.);
- [3D] object tracking;
- ...

~2010: new beginning of Deep Learning in Computer Vision

“general” Deep Learning

Perceptron (1958, Frank Rosenblatt)





Perceptron

Input data are represented as vectors $\mathbf{x} = \begin{bmatrix} \vdots \end{bmatrix}$

Supervised binary classification:

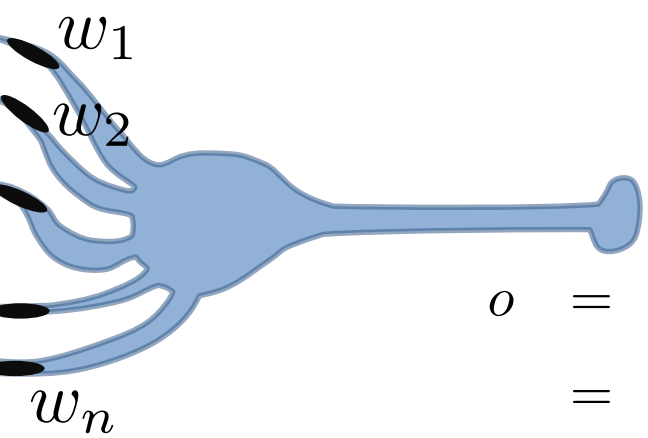
$\mathbf{x}$

→ `{cat, non-cat}`

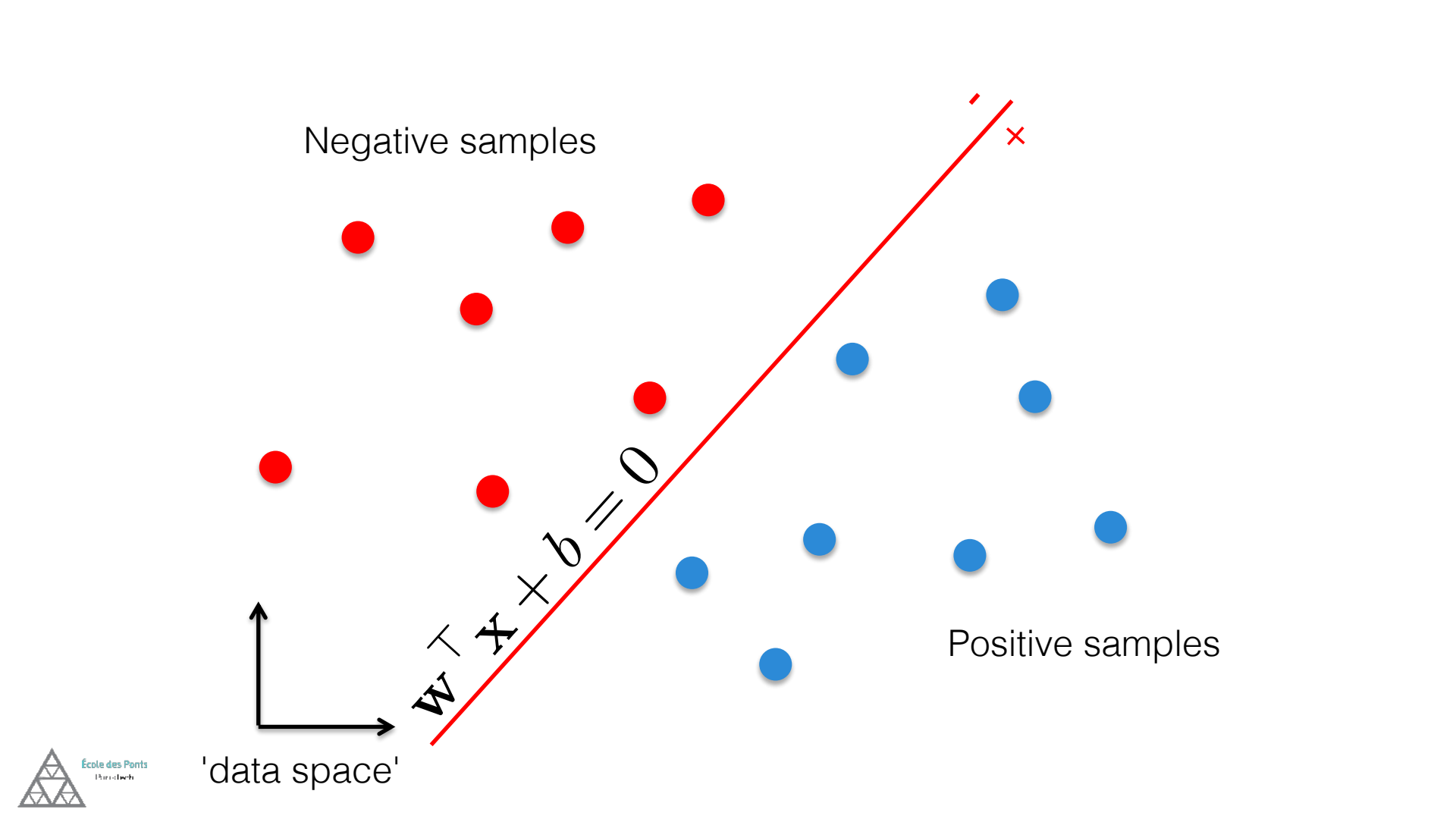
$$f : \mathbf{x} \in \mathbb{R}^n \rightarrow \{+1, -1\}$$

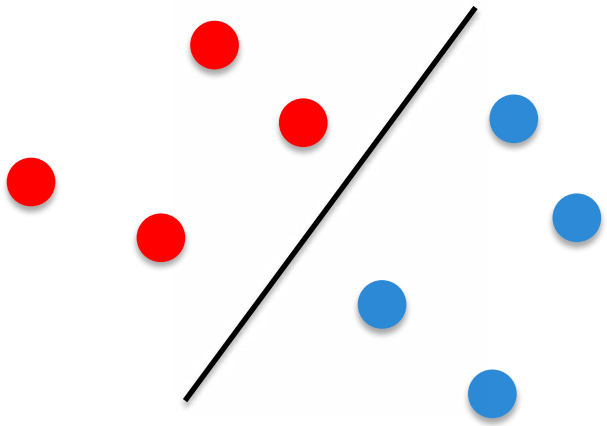
We are looking for a function $f(\mathbf{x})$ of the form:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

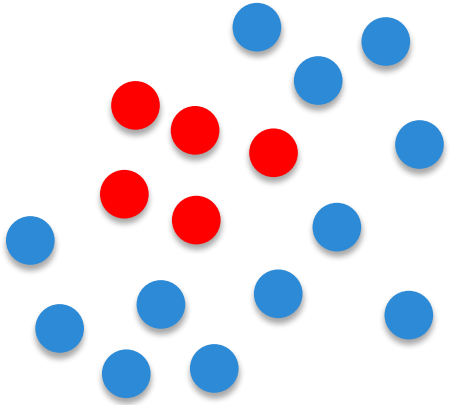


$$\begin{aligned} o &= \sum_i w_i x_i + b \\ &= \mathbf{w}^\top \mathbf{x} + b \end{aligned}$$


Negative samples
$w^\top x + b = 0$
Positive samples
'data space'

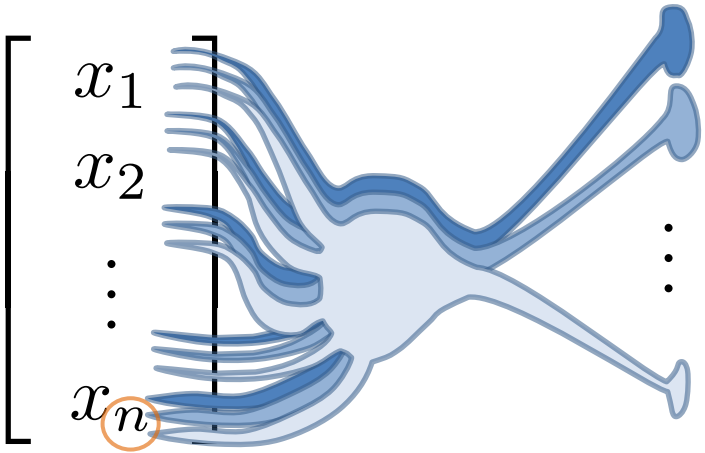


linearly separable



nonlinearly separable

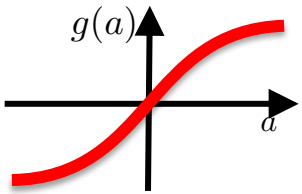
- The Perceptron: A 1-layer “network”;
- A 2-layer network (= 1-hidden layer network);

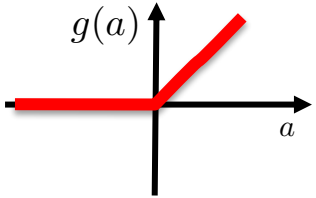


$$h_1 = g\left(\sum_i W_{1,i} x_i + b_1\right)$$

$g(\,)$: non-linear function

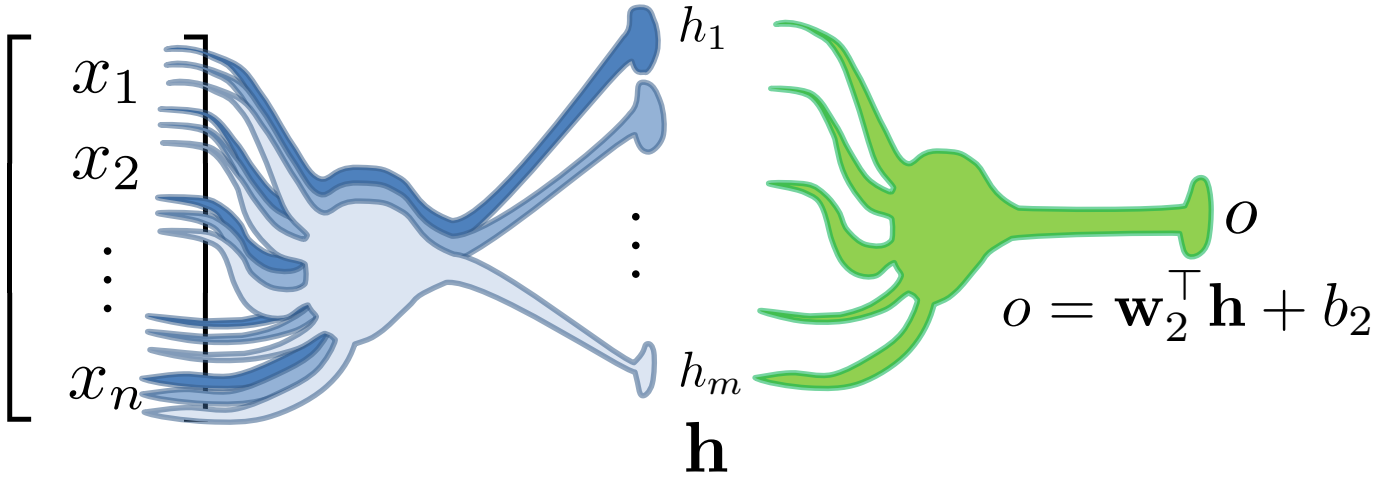
$$h_m = g\left(\sum_i W_{m,i} x_i + b_m\right)$$



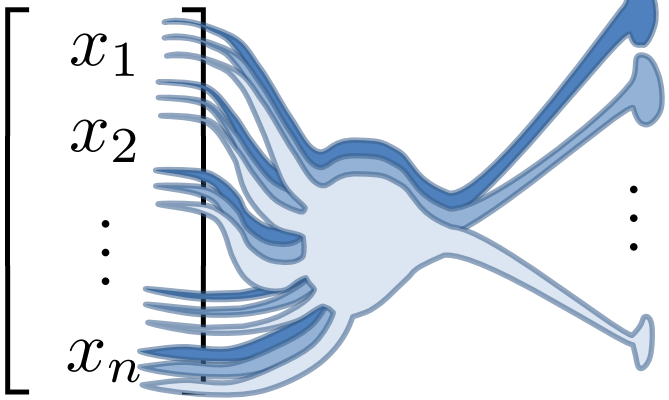


Rectified Linear Unit (ReLU)

$$g(a) = \max(0, a)$$



$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}_2^\top \mathbf{h} + b_2 \geq 0 \ , \\ -1 & \text{otherwise} \ . \end{cases}$$

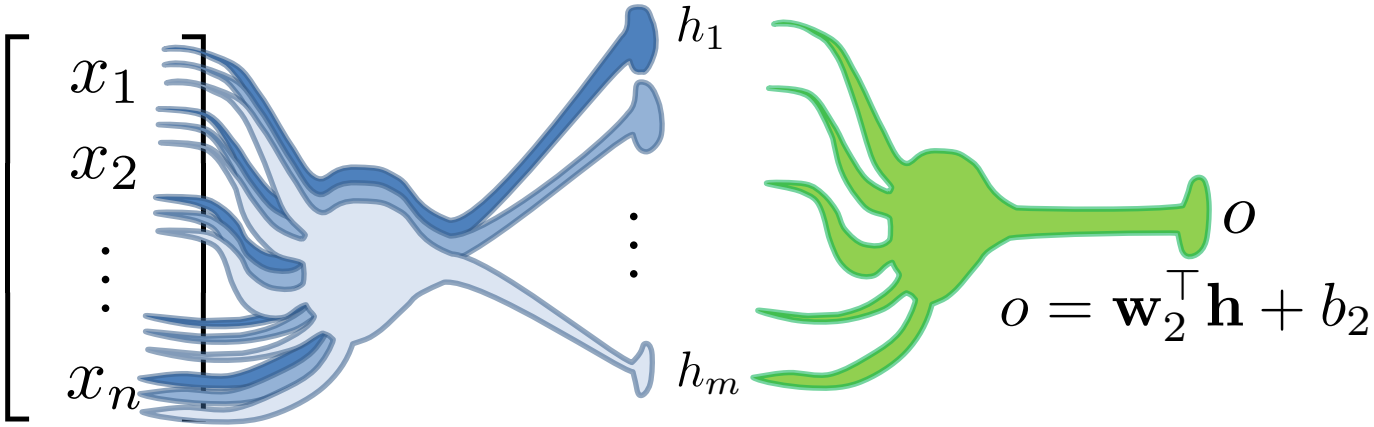


$$h_1 = g\left(\sum_i W_{1,i} x_i + b_1\right)$$

$$h_m = g\left(\sum_i W_{m,i} x_i + b_m\right)$$

$$\rightarrow \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Two-Layer Network



$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ o(\mathbf{x}) = \mathbf{w}_2^\top \mathbf{h}(\mathbf{x}) + b_2 \end{cases}$$

$g(\,)$: non-linear function

The coefficients of matrix $\mathbf{W}$, vectors $\mathbf{b}$ and $\mathbf{w_2}$, scalar b_2 are the *parameters* of the network

$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ o(\mathbf{x}) = \mathbf{w}_2^\top \mathbf{h}(\mathbf{x}) + b_2 \end{cases}$$

$g(\,)$: non-linear function

Universal Approximation Theorem

Any continuous function can be approximated under mild conditions *as closely as* wanted by a two-layer network:

$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ o(\mathbf{x}) = \mathbf{w}_2^\top \mathbf{h}(\mathbf{x}) + b_2 \end{cases}$$

Universal Approximation Theorem

Proves that any continuous function can be approximated by a two-layer network:

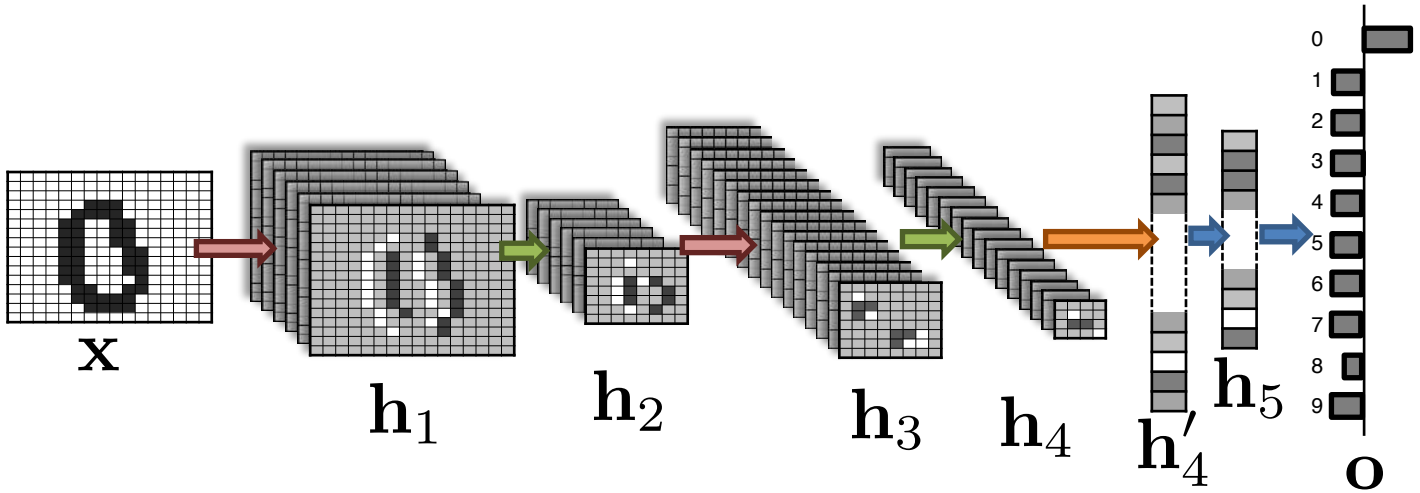
$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ o(\mathbf{x}) = \mathbf{w}_2^\top \mathbf{h}(\mathbf{x}) + b_2 \end{cases}$$

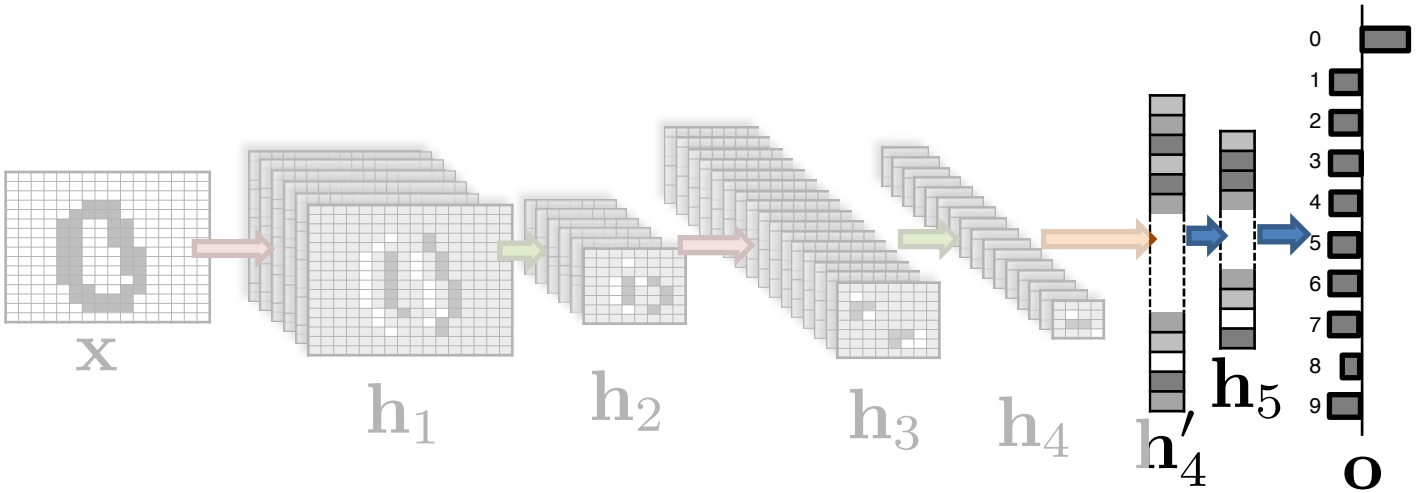
Can require a very large $\mathbf{h}$.

Deeper networks can mitigate this problem.

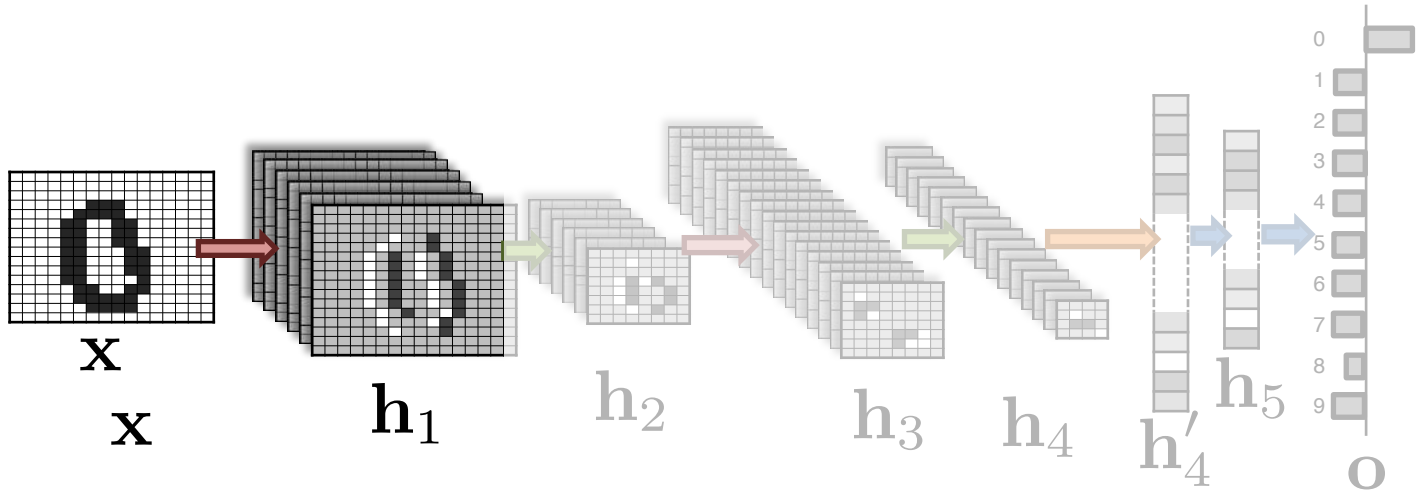
a first Deep Network

Dealing with Images

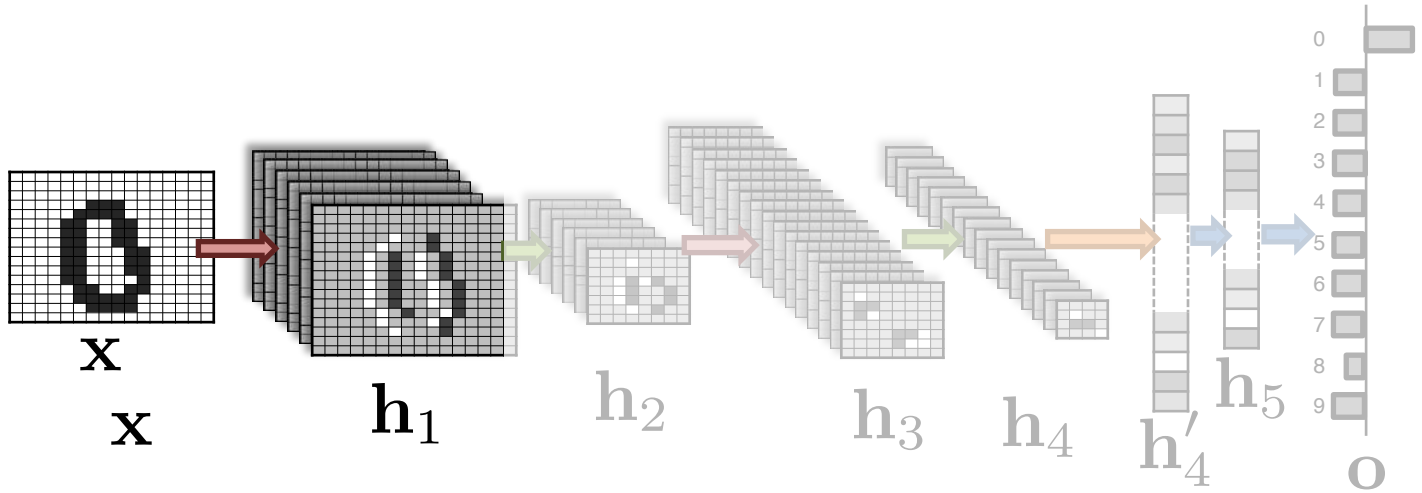




$$\mathbf{h}_5 = g(\mathbf{W}_5\mathbf{h}'_4 + \mathbf{b}_5)$$
$$\mathbf{o} = \mathbf{W}_6\mathbf{h}_5 + \mathbf{b}_6$$



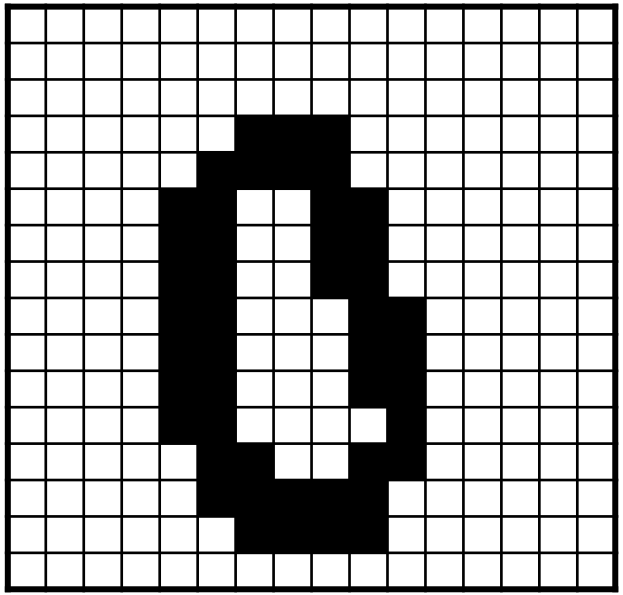
$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$



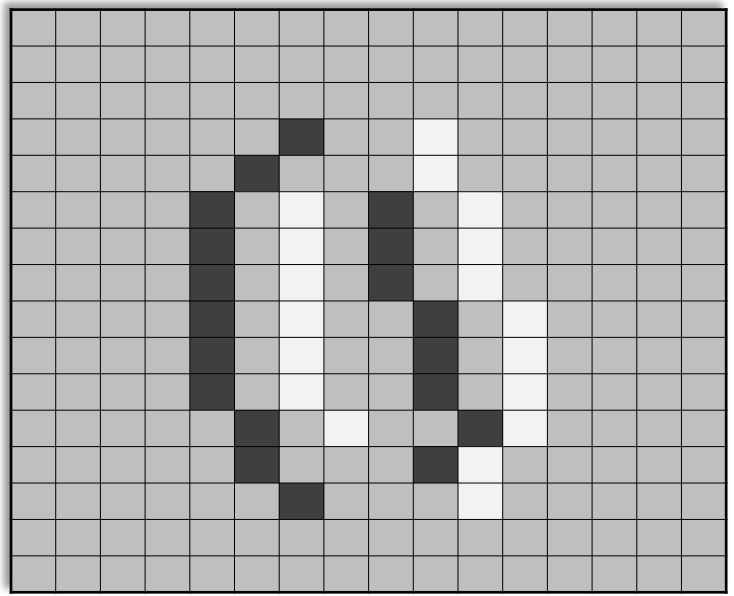
Product of convolution: $\mathbf{h}_{1,1} = g(\mathbf{f}_{1,1} * \mathbf{x} + \mathbf{b}_{1,1})$

$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$

Convolution: Example



$\mathbf{x}$

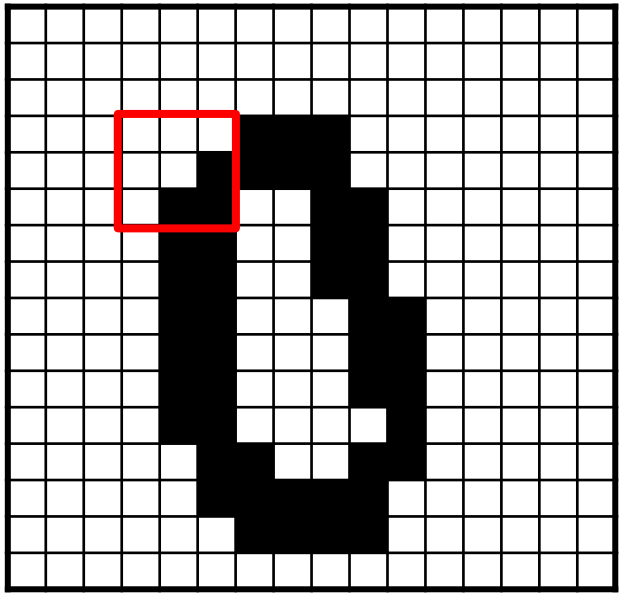


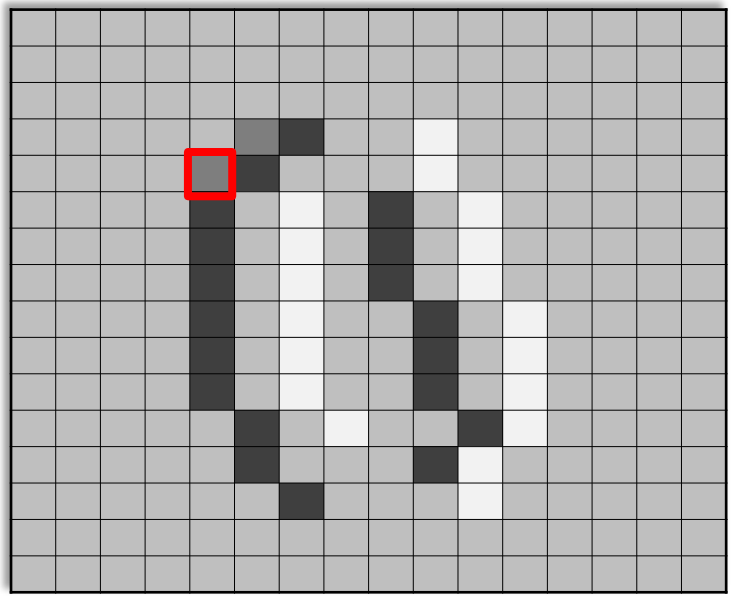
$\mathbf{h}_{1,1}$

$$\mathbf{f}_{1,1} = \begin{array}{|c|c|c|}\hline \textbf{-1} & \textbf{0} & \textbf{+1} \\ \hline \textbf{-1} & \textbf{0} & \textbf{+1} \\ \hline \textbf{-1} & \textbf{0} & \textbf{+1} \\ \hline \end{array}$$

$$\mathbf{h}_{1,1} = g(\mathbf{f}_{1,1} * \mathbf{x} + \mathbf{b}_{1,1})$$

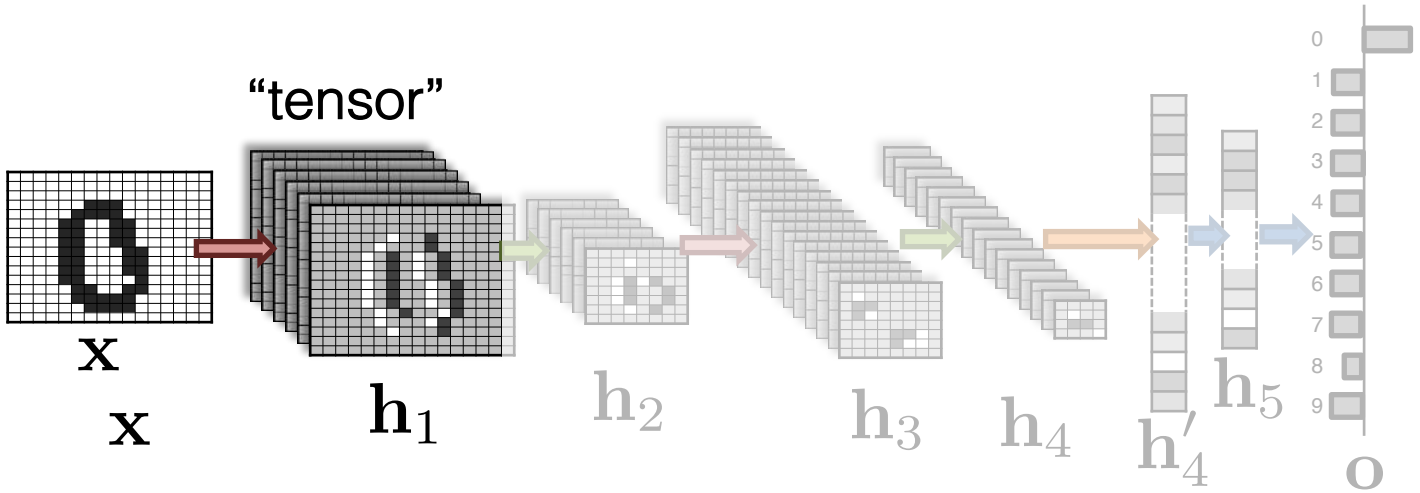
Numerical Example





$$\mathbf{f}_{1,1} = \begin{array}{|c|c|c|}\hline \textbf{-1} & \textbf{0} & \textbf{+1} \\ \hline \textbf{-1} & \textbf{0} & \textbf{+1} \\ \hline \textbf{-1} & \textbf{0} & \textbf{+1} \\ \hline \end{array}$$

$$\begin{aligned} h &= (-1) \times 255 + 0 \times 255 + (+1) \times 255 + \\ &\quad (-1) \times 255 + 0 \times 255 + (+1) \times 0 + \\ &\quad (-1) \times 255 + 0 \times 0 + (+1) \times 0 \\ &= -255 + 0 + 255 \\ &\quad -255 + 0 + 0 \\ &\quad -255 + 0 + 0 \\ &= -510 \end{aligned}$$

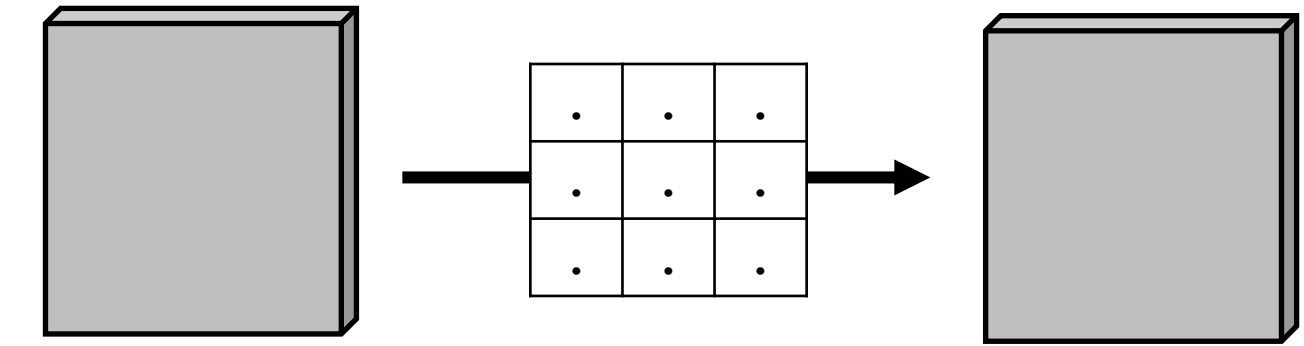


Product of convolution: $\mathbf{h}_{1,1} = g(\mathbf{f}_{1,1} * \mathbf{x} + \mathbf{b}_{1,1})$

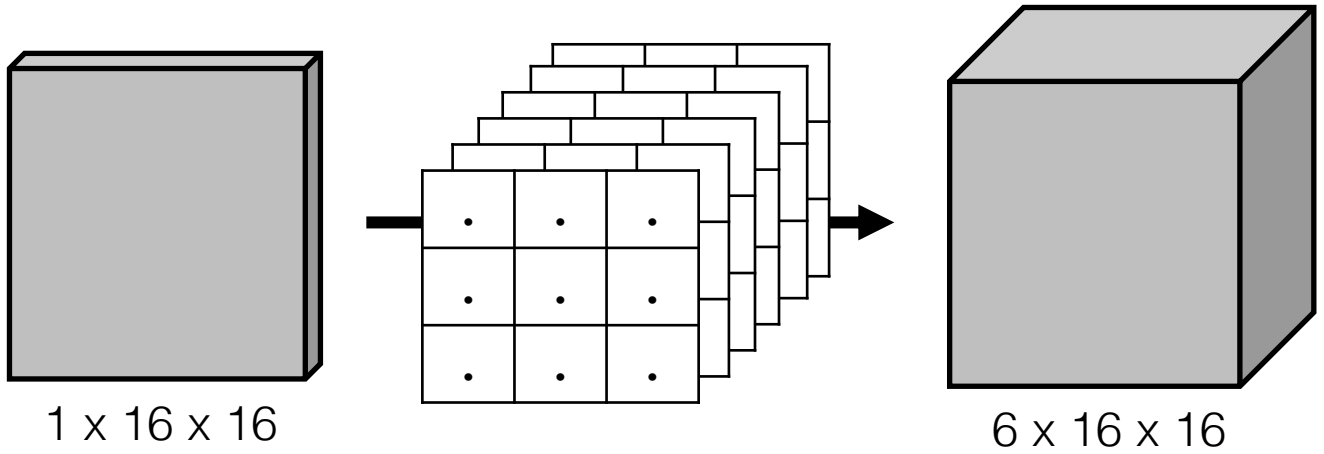
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$

“tensor”

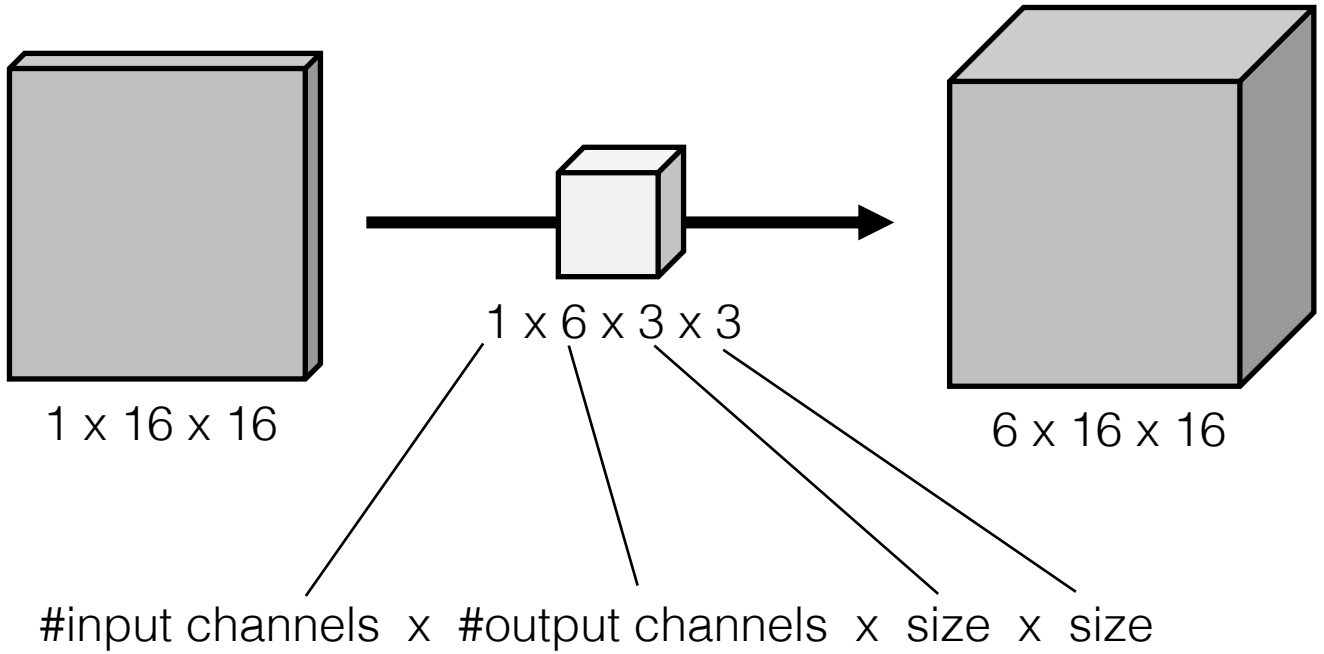
Tensors



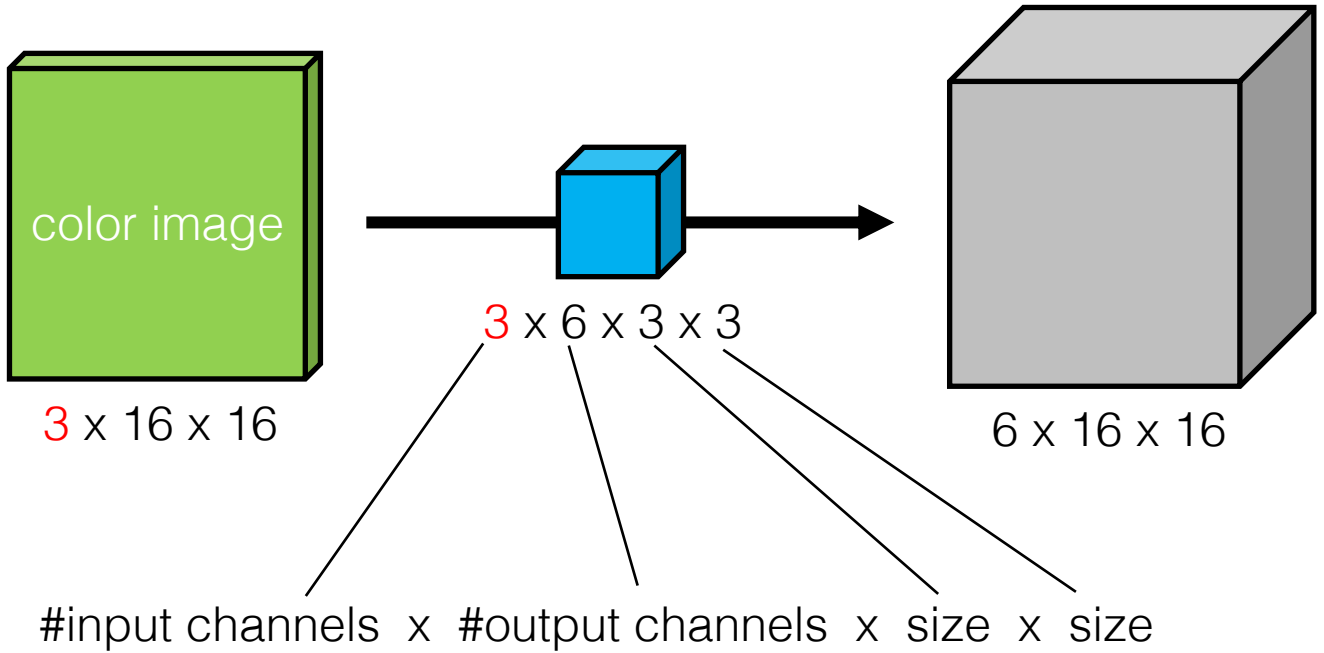
Tensors

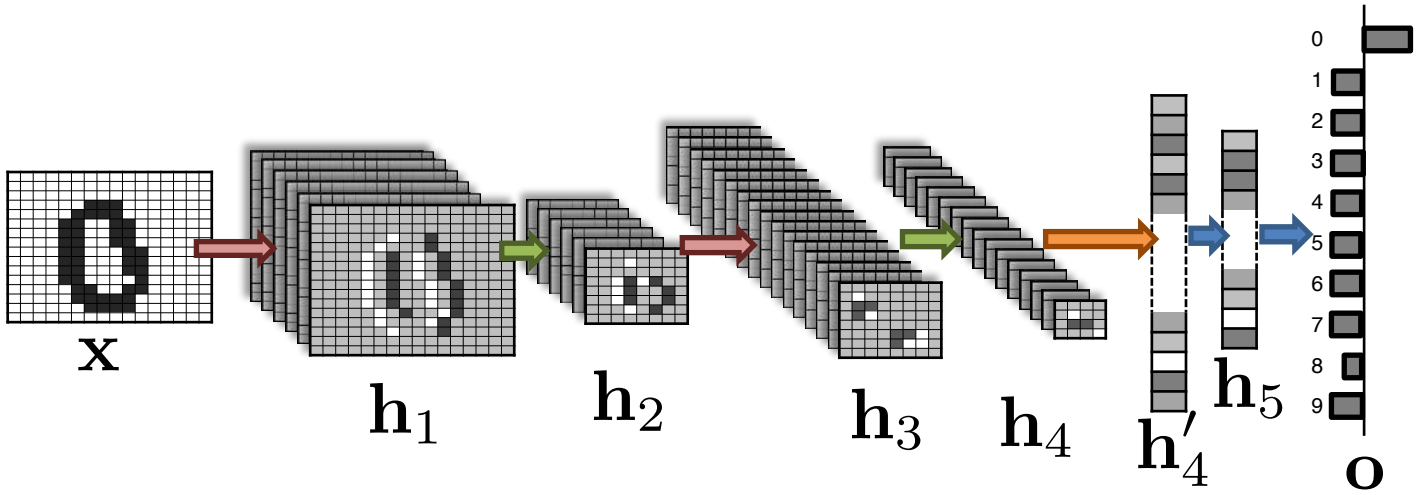


Tensors

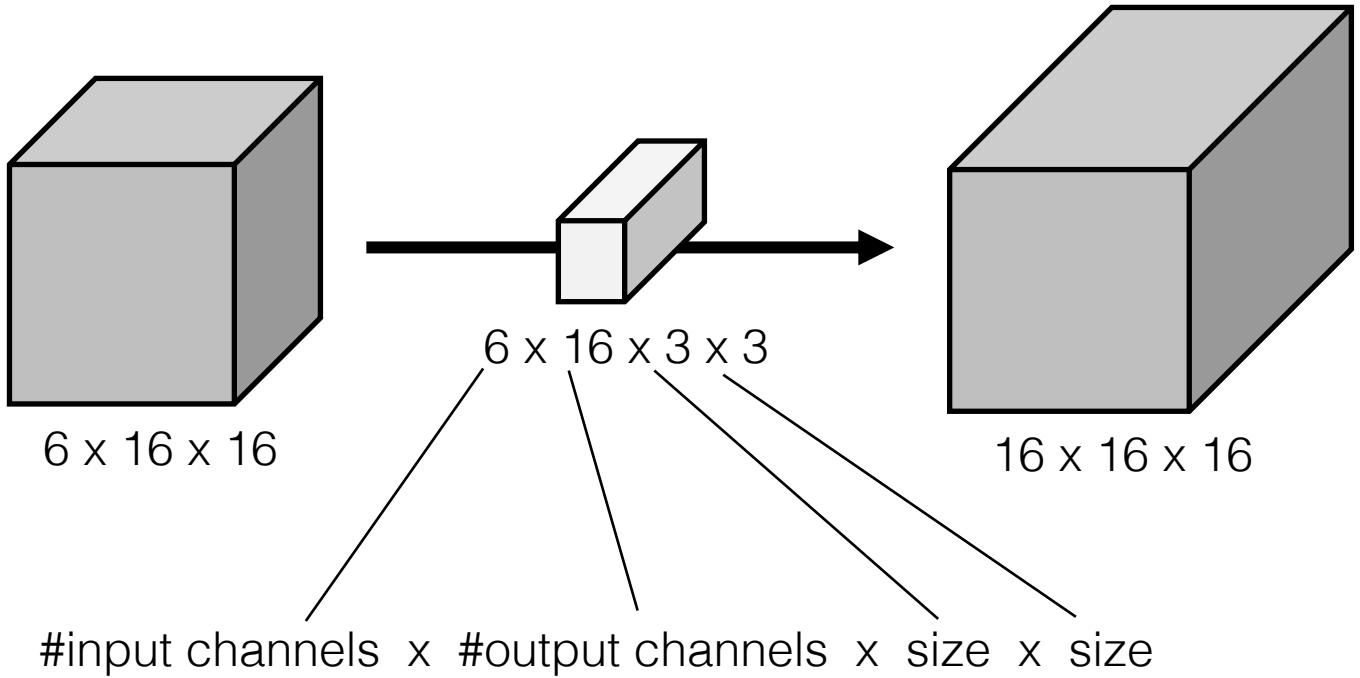

1 x 16 x 16
1 x 6 x 3 x 3
6 x 16 x 16
#input channels x #output channels x size x size

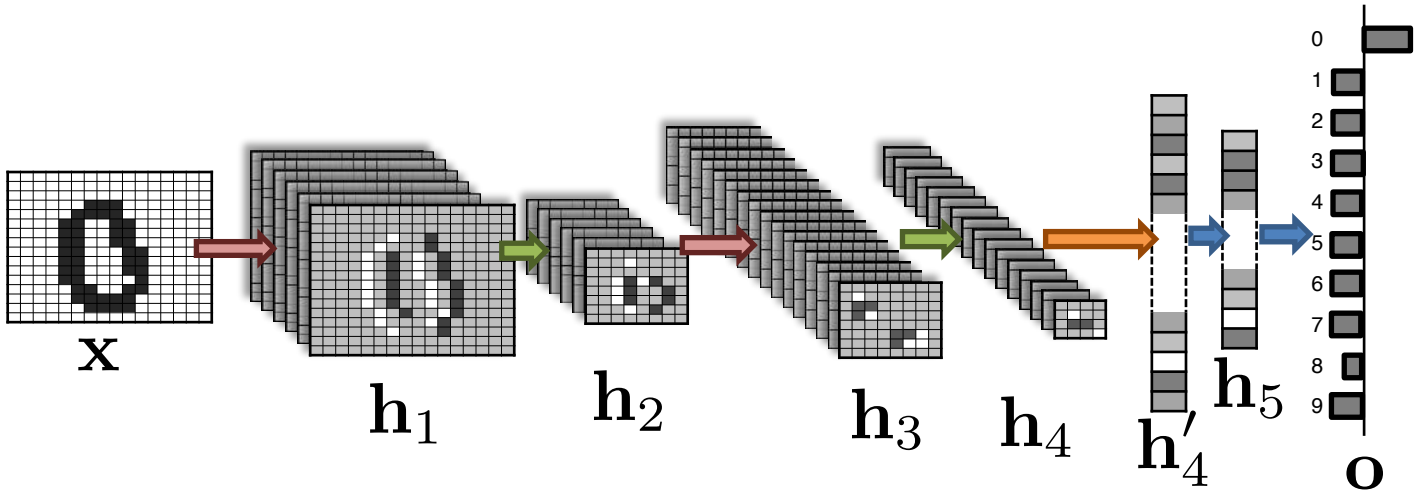
Tensors


color image
3 x 16 x 16
3 x 6 x 3 x 3
6 x 16 x 16
#input channels x #output channels x size x size

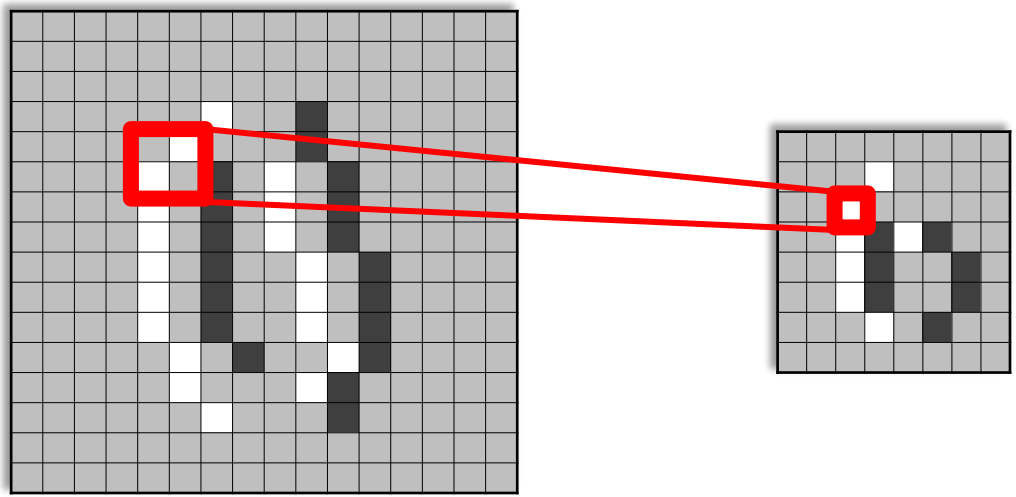

0
1
2
3
4
5
6
7
8
9
x
h_1
h_2
h_3
h_4
h'_4
h_5
o

Tensors


6 x 16 x 16
6 x 16 x 3 x 3
16 x 16 x 16
#input channels x #output channels x size x size

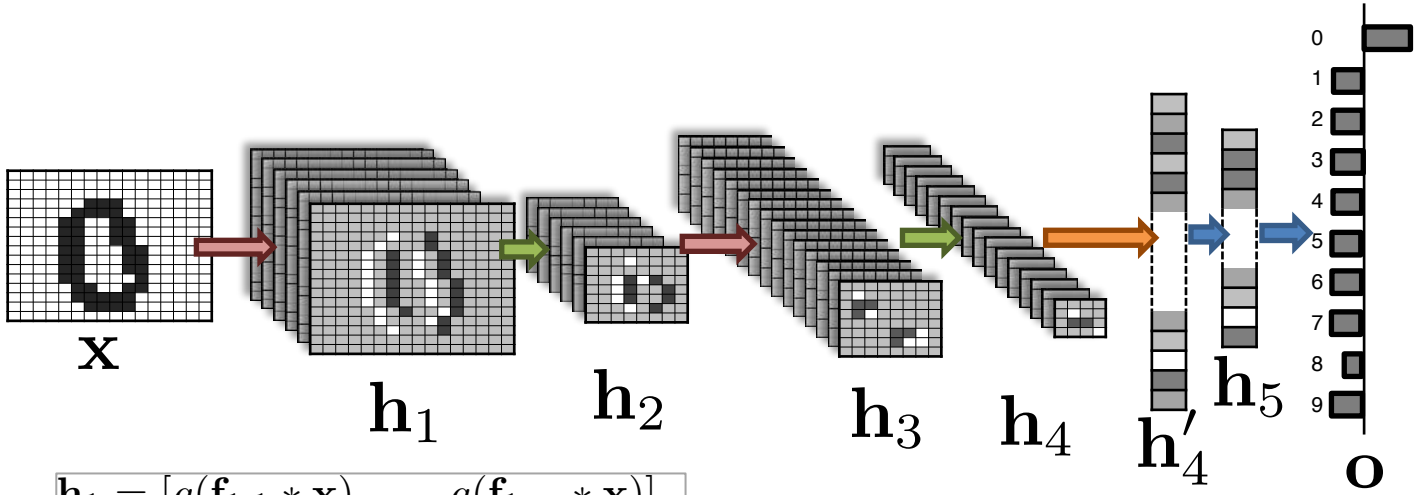

0
1
2
3
4
5
6
7
8
9
x
h_1
h_2
h_3
h_4
h'_4
h_5
o

Subsampling / Pooling



For example, max-pooling:

$$\mathbf{h}_i[u, v] = \max\{ \quad \mathbf{h}_{i-1}[2u, \quad 2v], \quad \mathbf{h}_{i-1}[2u, \quad 2v+1], \quad \mathbf{h}_{i-1}[2u+1, \quad 2v], \quad \mathbf{h}_{i-1}[2u+1, \quad 2v+1] \quad \}$$



$$
\begin{aligned}
\mathbf{h}_1 &= [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})] \\
\mathbf{h}_2 &= \text{pooling}(\mathbf{h}_1) \\
\mathbf{h}_3 &= [g(\mathbf{f}_{3,1} * \mathbf{h}_2), \ldots, g(\mathbf{f}_{3,n} * \mathbf{h}_2)] \\
\mathbf{h}_4 &= \text{pooling}(\mathbf{h}_3) \\
\mathbf{h}'_4 &= \text{Vec}(\mathbf{h}_4) \\
\mathbf{h}_5 &= g(\mathbf{W}_5 \mathbf{h}'_4 + \mathbf{b}_5) \\
\mathbf{o} &= \mathbf{W}_6 \mathbf{h}_5 + \mathbf{b}_6
\end{aligned}
$$

convolutional layers

pooling layers

fully-connected layers

Optimization

Finding the Parameters

$$\begin{cases} \mathbf{h}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ \mathbf{o}(\mathbf{x}) = \mathbf{W}_2\mathbf{h}(\mathbf{x}) + \mathbf{b}_2 \end{cases}$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, *and* $\mathbf{b}_2$*?*

→ By minimizing a loss function. The loss function can be adapted to the problem.

$$(\widehat{\mathbf{W}}, \widehat{\mathbf{b}}, \widehat{\mathbf{W}_2}, \widehat{\mathbf{b}_2}) = \underset{(\mathbf{W},\mathbf{b},\mathbf{W}_2,\mathbf{b}_2)}{\arg\min} \mathcal{L}(\mathbf{W}, \mathbf{b}, \mathbf{W}_2, \mathbf{b}_2)$$

Optimisation: (Variants of) gradient descent.

Loss Function

The loss function can be virtually any function that is differentiable.

Basic loss functions correspond to the cross-entropy between the expected values and the predicted ones:

$$\mathcal{L}(\Theta) = -\log \prod_{i=1}^{N} p_{\text{model}}(\mathbf{e}_i \mid \mathbf{o}(\mathbf{x}_i; \Theta)), \tag{2}$$

where

- $\{(\mathbf{x}_i, \mathbf{e}_i)\}_{i=1..N}$ is the training set;
- $\mathbf{o}(\mathbf{x}_i; \Theta)$ is the output of the network with parameters Θ for input $\mathbf{x}_i$;
- $p_{\text{model}}(\mathbf{e} \mid \mathbf{o}(\mathbf{x}; \Theta))$: How we compute the probability for a value $\mathbf{e}$ given the output $\mathbf{o}(\mathbf{x}; \Theta)$ of the network.

(the cross-entropy can also be seen here as the negative log-likelihood of the training set given the predictions of the network)

Regression Problems

If we take: $p_{\text{model}}(\mathbf{e} \mid \mathbf{o}(\mathbf{x};\Theta)) = \mathcal{N}(\mathbf{e};\, \mathbf{o}(\mathbf{x};\Theta), \sigma\mathbf{I})$:

$$\begin{aligned} \mathcal{L}(\Theta) &= -\log \textstyle\prod_{i=1}^{N} p_{\text{model}}(\mathbf{e}_i \mid \mathbf{o}(\mathbf{x}_i;\Theta)) \\ &= k_1 \textstyle\sum_{i=1}^{N} \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\|^2 + k_2 \,. \end{aligned} \tag{3}$$

(because $\mathcal{N}(\mathbf{e};\, \mathbf{o}, \sigma\mathbf{I}) = c_1 \exp(-\|\mathbf{e}-\mathbf{o}\|^2/c_2)$)

If we ignore constants k_1 and k_2 that do not influence minimum $\hat{\Theta}$:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{N} \|\mathbf{e}_i - \mathbf{o}(\mathbf{x}_i;\Theta)\|^2 \,, \tag{4}$$

which is simply the mean squared error (MSE).

Classification Problems

Training set: $\{(\mathbf{x}_i, c_i)\}_{i=1..N}$, where $c_i = [1; C]$ is the expected class index, among C possible classes.

In this case, the output $\mathbf{o}(\mathbf{x}_i; \Theta)$ of the network is taken to be a C-vector: $\mathbf{o}(\mathbf{x}_i; \Theta) \in \mathbb{R}^C$.

Let's introduce vector $\mathbf{d} = \mathrm{softmax}(\mathbf{o})$:

$$\mathbf{d}_i = \frac{\exp(\mathbf{o}_i)}{\sum_{j=1}^{C} \exp(\mathbf{o}_j)} \,. \tag{5}$$

- The $\mathrm{softmax}$ operation is a soft version of the operation "maximum value of $\mathbf{o}$ is changed to 1, the other values are changed to 0".
- The $\mathrm{softmax}$ operation transforms $\mathbf{o}$ into a distribution $\mathbf{d}$ over the C classes:

$$\begin{aligned} &\forall i \, \mathbf{d}_i \in [0, 1] \,; \\ &\sum_{i=1}^{C} \mathbf{d}_i = 1 \,. \end{aligned} \tag{6}$$

Classification Problems (2)

Training set: $\{(\mathbf{x}_i, c_i)\}_{i=1..N}$, where $c_i = [1; C]$ is the expected class index, among C possible classes.

Take: $p_{\text{model}}(c \mid \mathbf{o}(\mathbf{x}; \Theta)) = \mathbf{d}_c$ with $\mathbf{d} = \text{softmax}(\mathbf{o}(\mathbf{x}; \Theta))$.

Loss function:

$$\begin{aligned} \mathcal{L}(\Theta) &= -\log \prod_{i=1}^{N} p_{\text{model}}(c_i \mid \mathbf{o}(\mathbf{x}_i; \Theta)) \\ &= \sum_{i=1}^{N} -\log \text{softmax}(\mathbf{o}(\mathbf{x}_i; \Theta))_{c_i} \end{aligned} \tag{7}$$

Example of Loss Function

For example:

$$\mathcal{L}(\mathbf{W}, \mathbf{b}, \mathbf{W}_2, \mathbf{b}_2) = -\sum_{(\mathbf{x},d)\in\mathcal{T}} \log\left(\frac{\exp(\mathbf{o}(\mathbf{x})_d)}{\sum_i \exp(\mathbf{o}(\mathbf{x})_i)}\right)$$

Training sample

$(\,$  $, 2)$

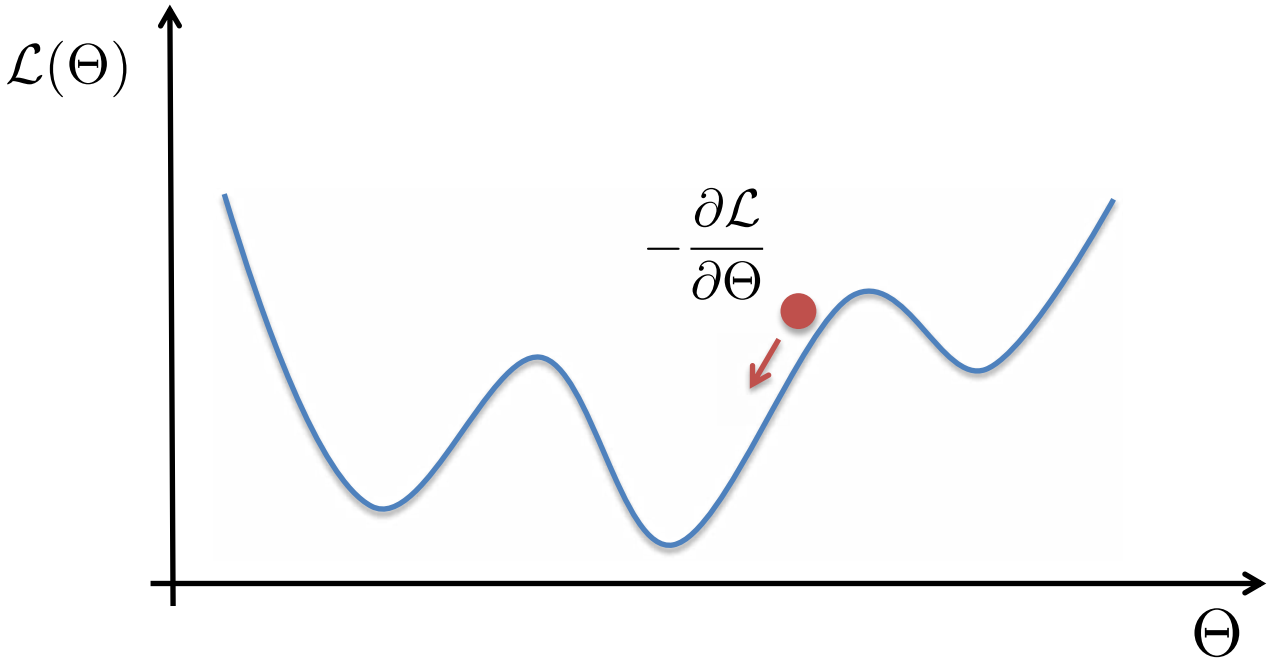
Predicted output for the training sample

Training set $\mathcal{T}$

Optimization for Deep Models

Variants of Gradient Descent:

$$\Theta \leftarrow \Theta - \lambda \frac{\partial \mathcal{L}(\Theta)}{\partial \Theta}. \tag{11}$$



Stochastic gradient descent

Loss function:

$$\mathcal{L}(\Theta) = \sum_i L\left(o(\mathbf{x}_i; \Theta), \mathbf{e}_i\right) .$$

$$\Theta^{t+1} \leftarrow \Theta^t - \eta_t G(\Theta^t) ,$$

where

$$G(\Theta^t) = \frac{\partial}{\partial \Theta} L\left(o(\mathbf{x}_{i_t}; \Theta_t), \mathbf{e}_{i_t}\right) .$$

Can converge despite the fact that $\mathcal{L}(\Theta)$ is not convex.

Momentum-based SGD

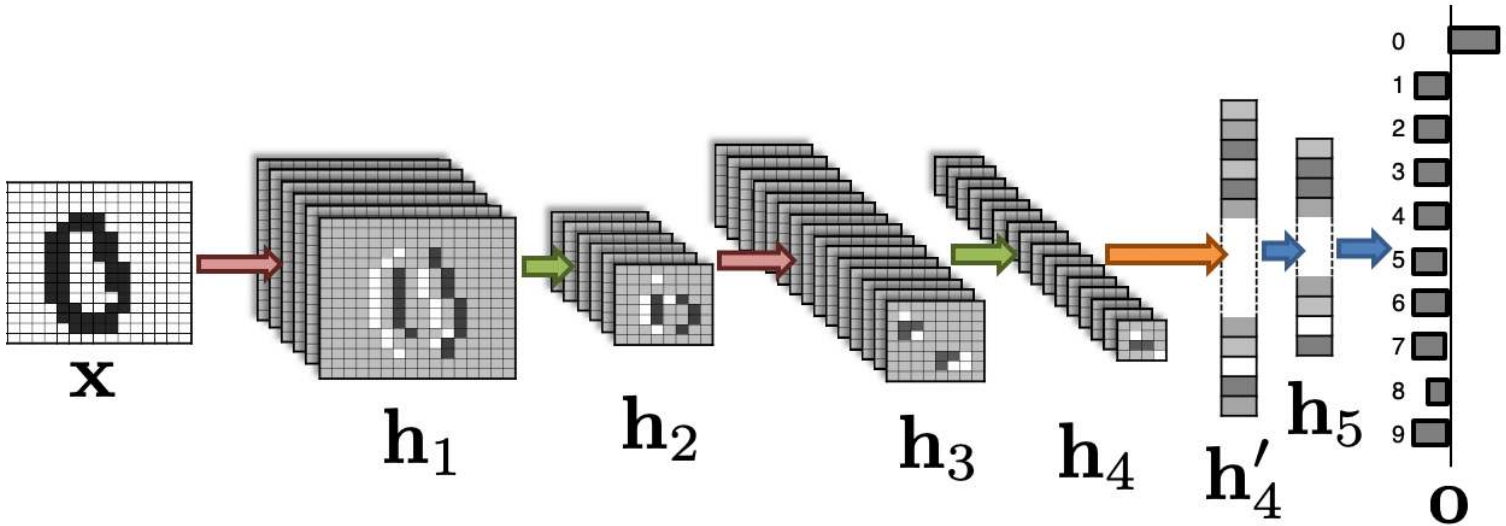
$$\begin{cases} \mathbf{g}^t \leftarrow \rho \mathbf{g}^{t-1} + (1-\rho) G(\Theta^t) \\ \Delta\Theta^t \leftarrow -\eta_t \mathbf{g}^t \\ \Theta^{t+1} \leftarrow \Theta^t + \Delta\Theta^t \end{cases}$$

Adam

$$
\begin{cases}
\mathbf{g}^t \leftarrow G(\Theta^t) \\
\mathbf{s}^t \leftarrow \rho_1 \mathbf{s}^{t-1} + (1-\rho_1)\mathbf{g}^t \\
\mathbf{r}^t \leftarrow \rho_2 \mathbf{r}^{t-1} + (1-\rho_2)\mathbf{g}^t \odot \mathbf{g}^t \\
\hat{\mathbf{s}}^t \leftarrow \frac{\mathbf{s}^t}{1-(\rho_1)^t} \\
\hat{\mathbf{r}}^t \leftarrow \frac{\mathbf{r}^t}{1-(\rho_2)^t} \\
\Delta\Theta^t \leftarrow -\frac{\lambda}{\delta+\sqrt{\hat{\mathbf{r}}^t}} \odot \hat{\mathbf{s}}^t \\
\Theta^{t+1} \leftarrow \Theta^t + \Delta\Theta^t
\end{cases}
$$

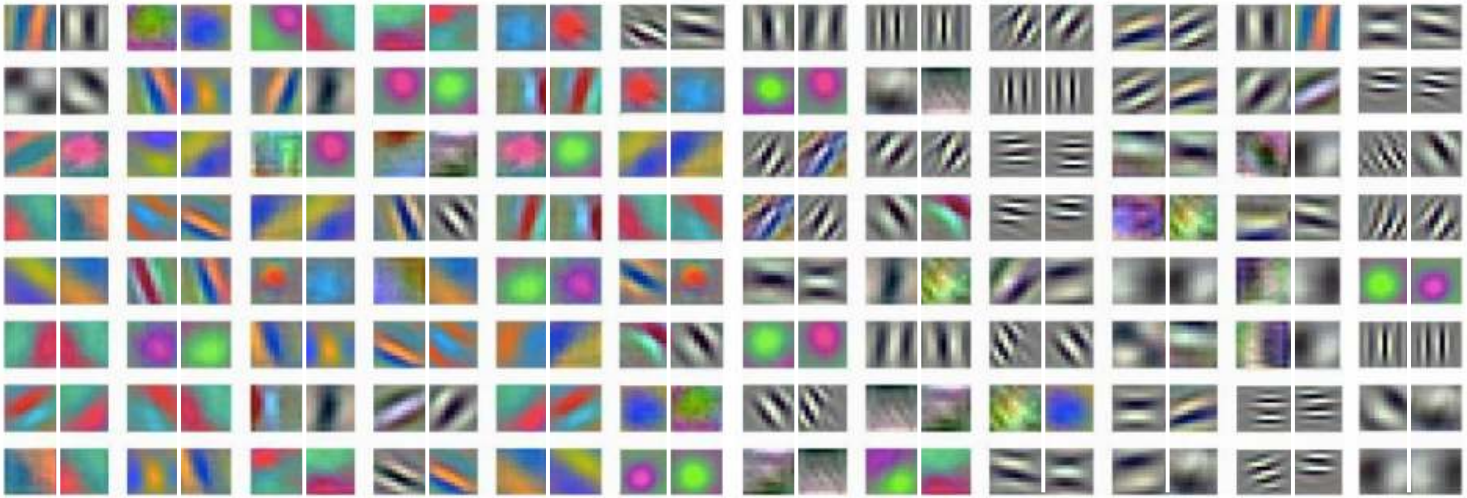
with $\epsilon \approx 0.001$, $\rho_1 \approx 0.9$, $\rho_2 \approx 0.999$.

Finding the Hyperparameters (Number of Layers, Number of Filters, Sizes of the Filters)?



- In practice, often from previous experience...;
- Automatically, using 'AutoML'.

Learned Filters for the First Layer for Natural Images

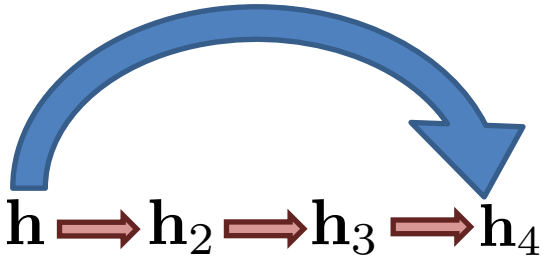


$\{\mathbf{f}_{1,j}\}_j$

Skip Connections

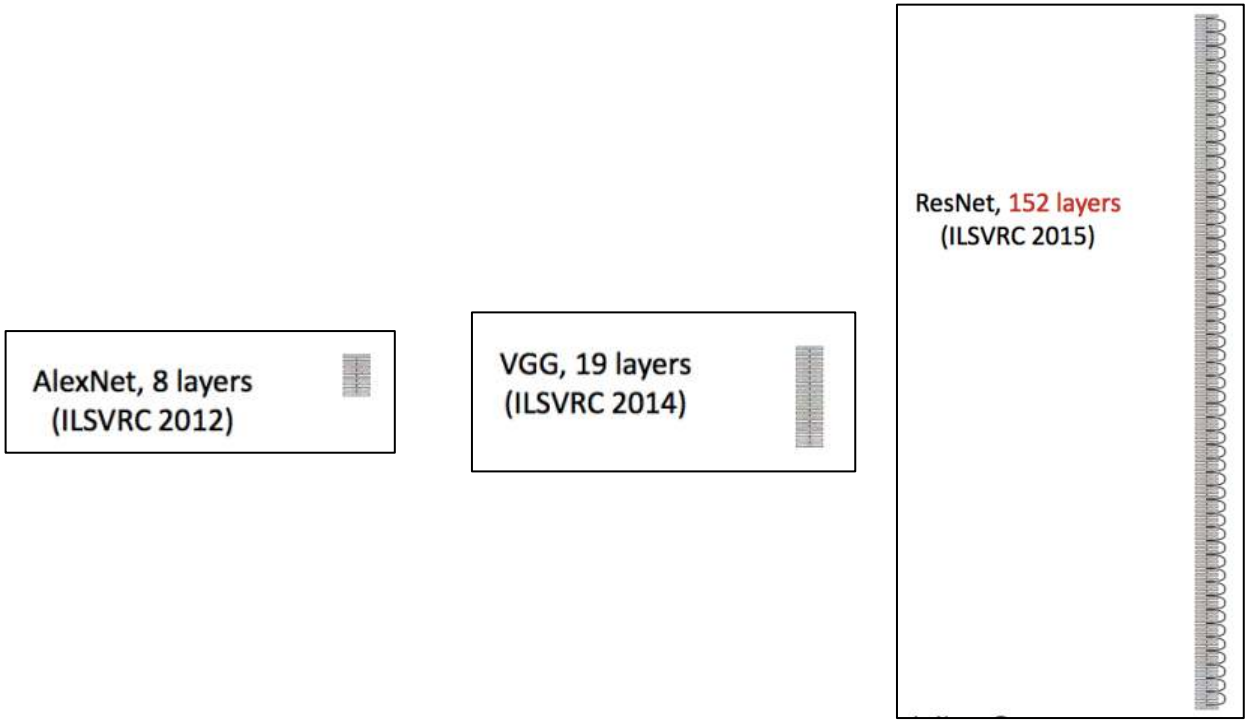
For example (Residual module):

$$\mathbf{h}_2 = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3$$
$$\mathbf{h}_4 = g(\mathbf{h} + \mathbf{h}_3)$$

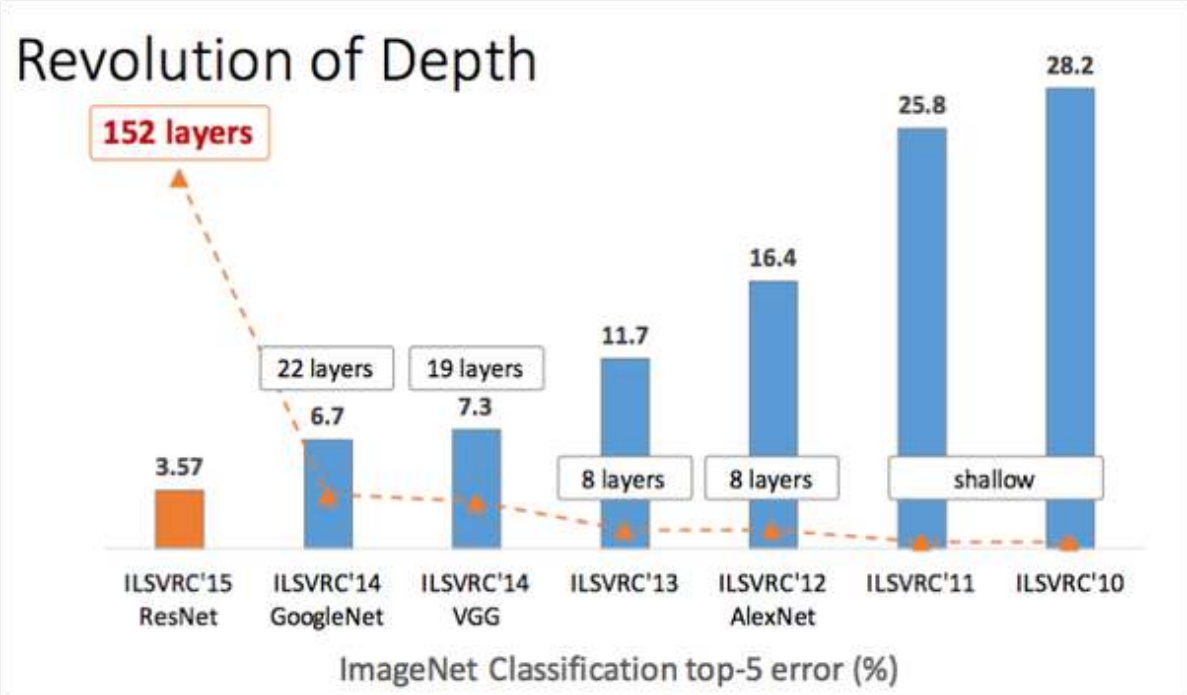


Limits vanishing and exploding gradients.

ResNet [He et al, CVPR 2016]

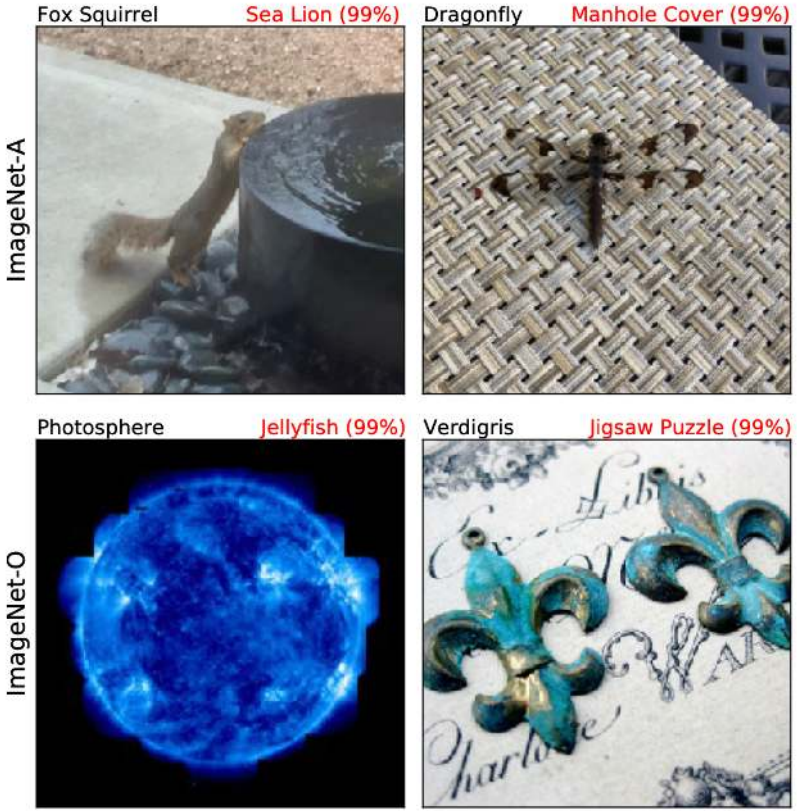


ResNet [He et al, CVPR 2016]



deep learning is not error-proof! (yet?)

some failures



Natural Adversarial Examples. CVPR 2021

recognizing objects

recognizing objects

(a) Output prediction on original images.

(b) Prediction when foreground is whitened.

recognizing objects

(a) Output prediction on original images.

(b) Prediction when foreground is whitened.

Beyond supervised learning

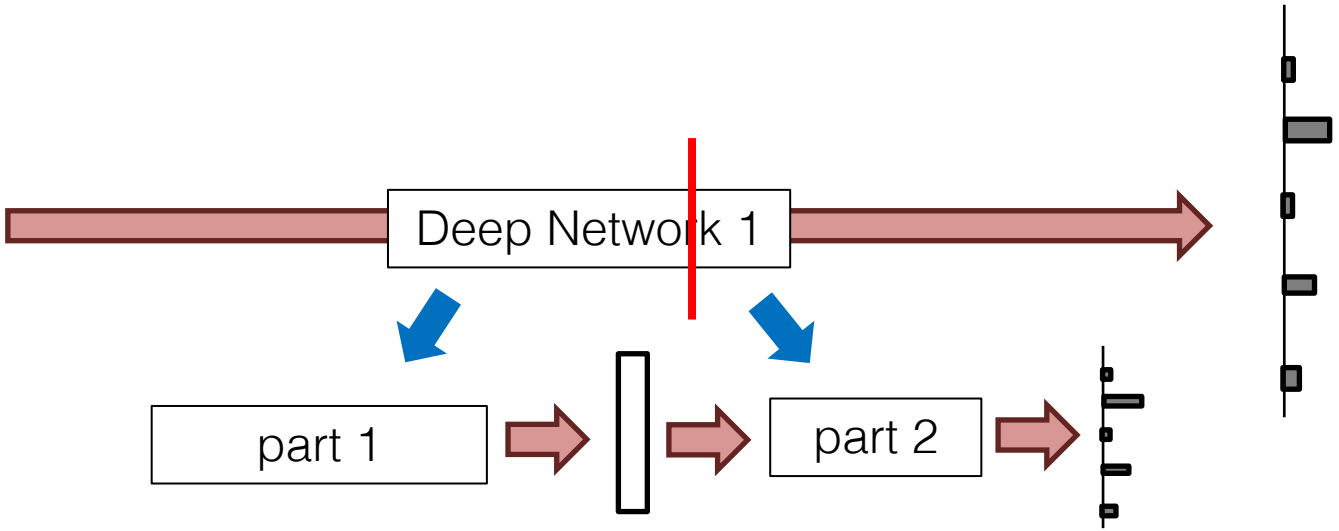
- We can use a Deep Network to approximate any continuous function;

- We can use any loss function as long as it is differentiable;

→ very flexible!

Image Embedding

Image Embedding



‘features’ or ‘embedding’

Code (PyTorch)

```
class Net(Module):
  def __init__(self):
    super(Net, self).__init__()
    self.cnn_layers = Sequential(
      Conv2d(1, 4, kernel_size=3, stride=1, padding=1),
      BatchNorm2d(4),
      ReLU(inplace=True),
      MaxPool2d(kernel_size=2, stride=2),
      Conv2d(4, 4, kernel_size=3, stride=1, padding=1),
      BatchNorm2d(4),
      ReLU(inplace=True),
      MaxPool2d(kernel_size=2, stride=2),
    )
    self.linear_layers = Sequential(
      Linear(4 * 7 * 7, 10)
    )

  def forward(self, x):
    x = self.cnn_layers(x)
    x = x.view(x.size(0), -1)
    x = self.linear_layers(x)
  return x


# ...
```

```
# ...

  def train(epoch):
    model.train()
    tr_loss = 0
    # getting the training set
    x_train = Variable(train_x),
    y_train = Variable(train_y)
    x_val = Variable(val_x)
    y_val = Variable(val_y)
    optimizer.zero_grad()
    output_train = model(x_train)
    output_val = model(x_val)
    loss_train = criterion(output_train, y_train)
    loss_val = criterion(output_val, y_val)
    train_losses.append(loss_train)
    val_losses.append(loss_val)
    loss_train.backward()
    optimizer.step()
    tr_loss = loss_train.item()
```

```
for epoch in range(n_epochs):
  train(epoch)
```