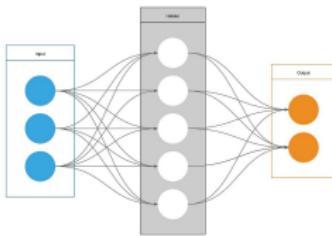


RNNs and CNNs

Cecilia Jarne

cecilia.jarne@unq.edu.ar



Twitter: [@ceciliajarne](https://twitter.com/ceciliajarne)



Cecilia Jarne

Neural Networks and Keras

What is this talk about?

- Recurrent Neural Networks (RNN).
- Convolutional Neural Networks (CNN).
- Motivation and some applications.

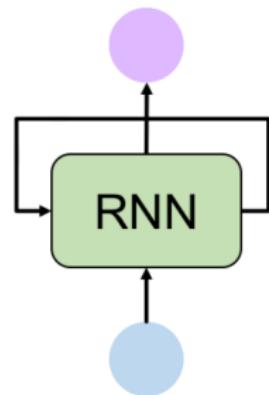
Recurrent Neural Networks (RNN)

- Sequence prediction: given a sequence of elements to predict the next one.
- Some example application are:
 - Text sequence prediction.
 - Time series in general.
 - Speech.
- Models of cortex and other brain areas great recurrence in their connections.

Recurrent Neural Networks

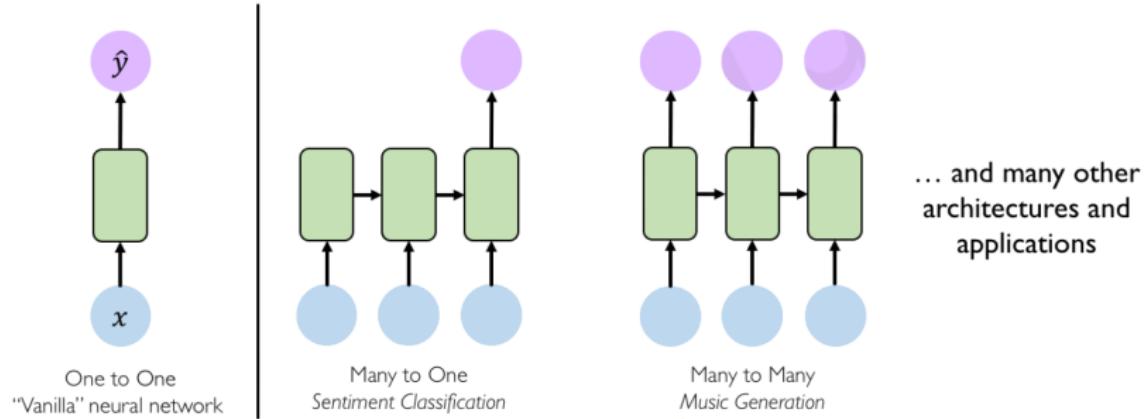
To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence

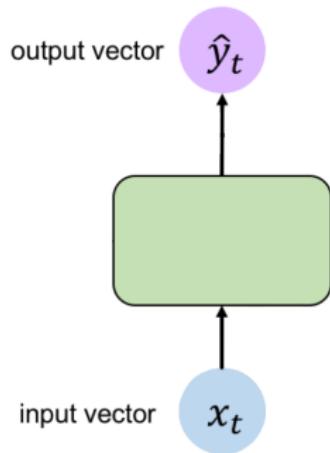


Today: **Recurrent Neural Networks (RNNs)** as
an approach to sequence modeling problems

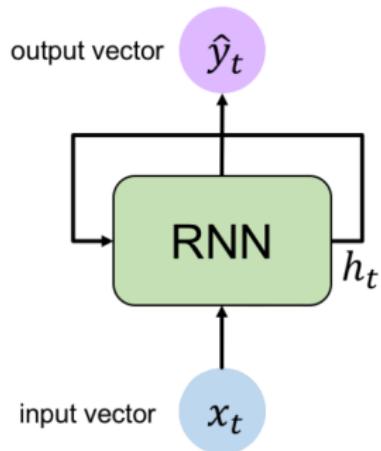
Recurrent Neural Networks



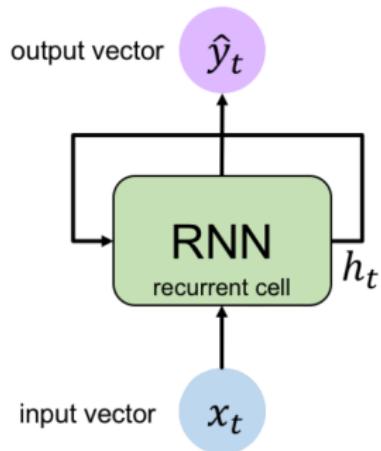
Recurrent Neural Networks



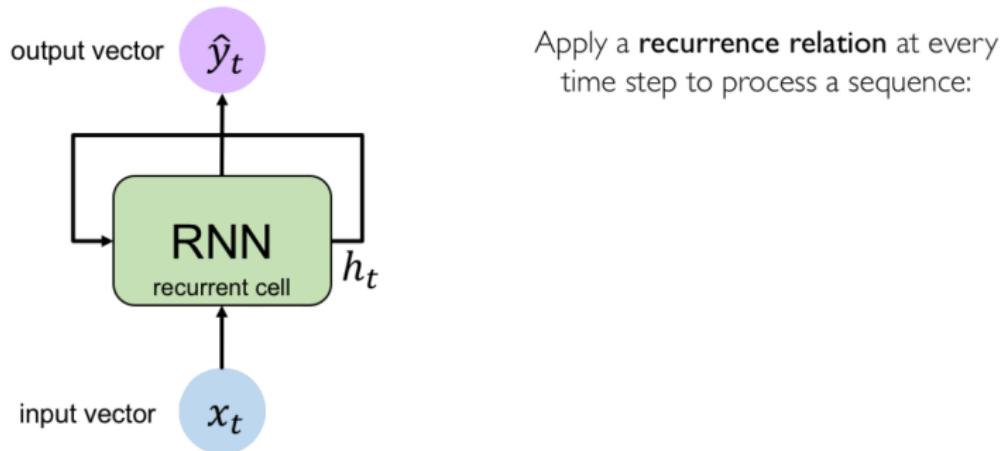
Recurrent Neural Networks



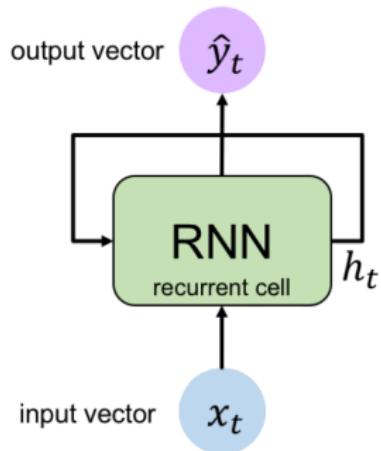
Recurrent Neural Networks



Recurrent Neural Networks



Recurrent Neural Networks

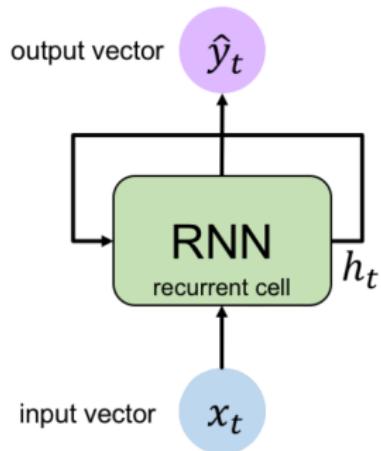


Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(h_{t-1}, x_t)$$

cell state function parameterized by W old state input vector at time step t

Recurrent Neural Networks



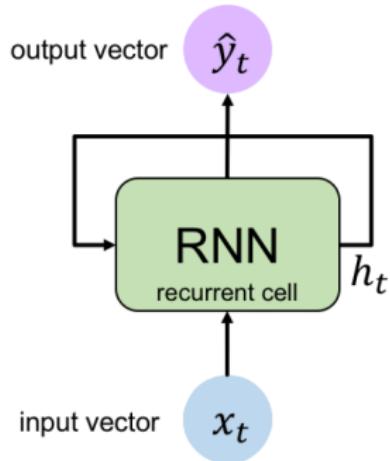
Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(h_{t-1}, x_t)$$

cell state function parameterized by W old state input vector at time step t

Note: the same function and set of parameters are used at every time step

Recurrent Neural Networks



Output Vector

$$\hat{y}_t = \mathbf{W}_{hy}^T h_t$$

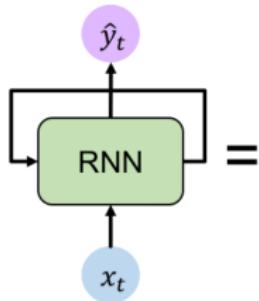
Update Hidden State

$$h_t = \tanh(\mathbf{W}_{hh}^T h_{t-1} + \mathbf{W}_{xh}^T x_t)$$

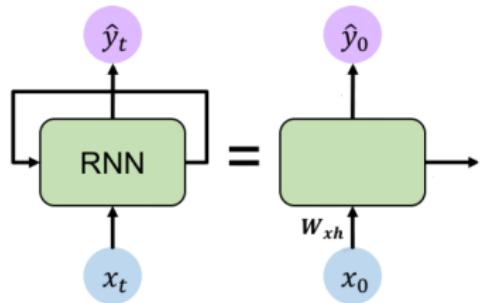
Input Vector

$$x_t$$

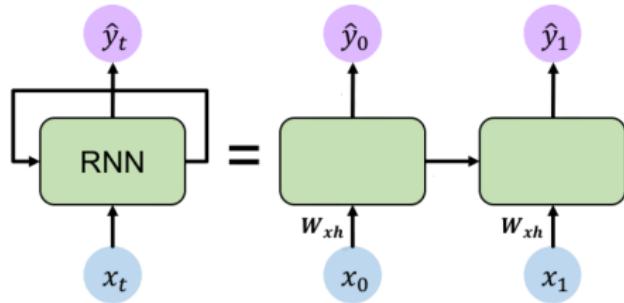
Recurrent Neural Networks



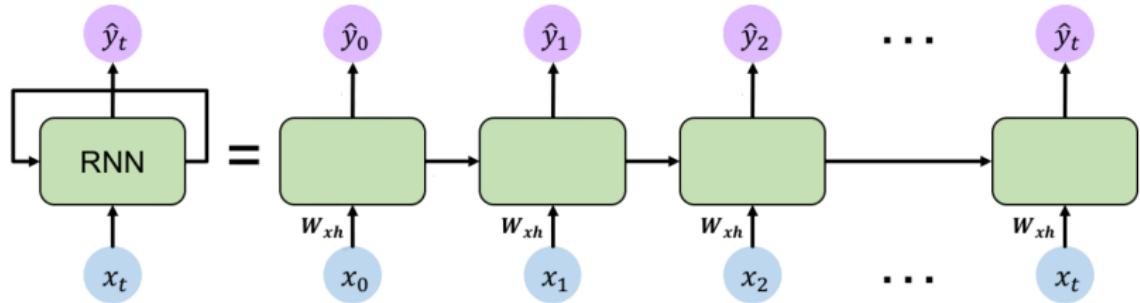
Recurrent Neural Networks



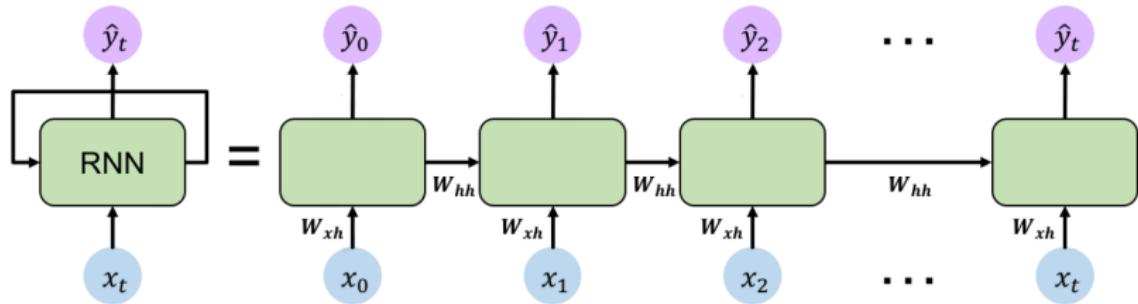
Recurrent Neural Networks



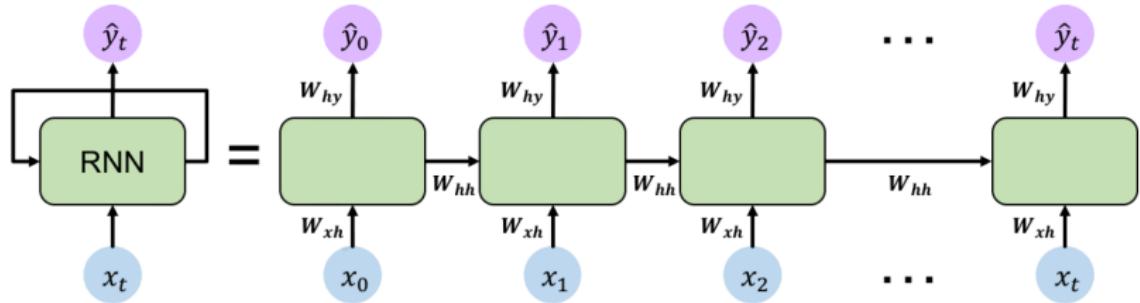
Recurrent Neural Networks



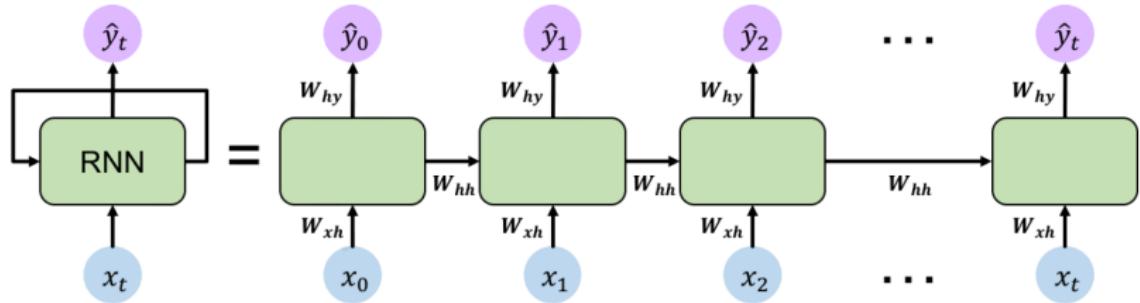
Recurrent Neural Networks



Recurrent Neural Networks

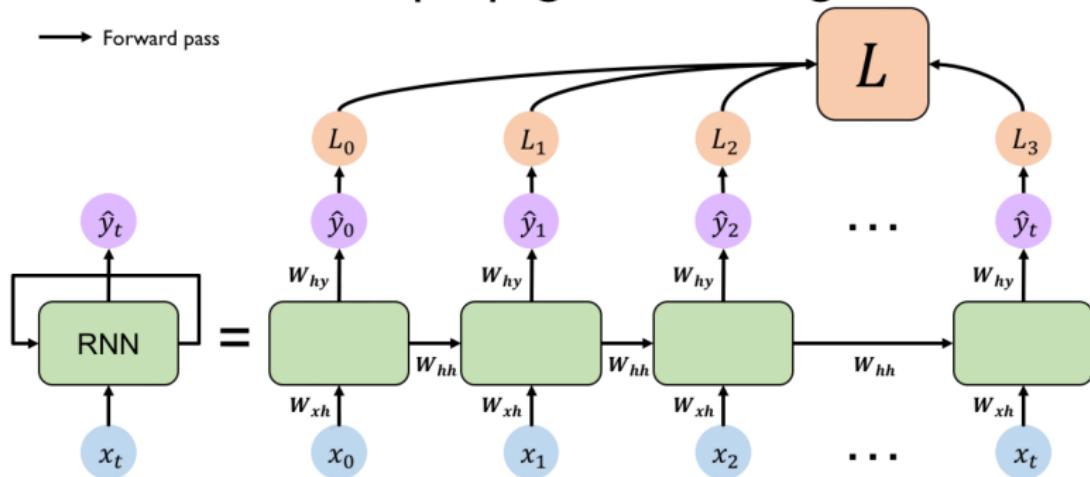


Backpropagation through time



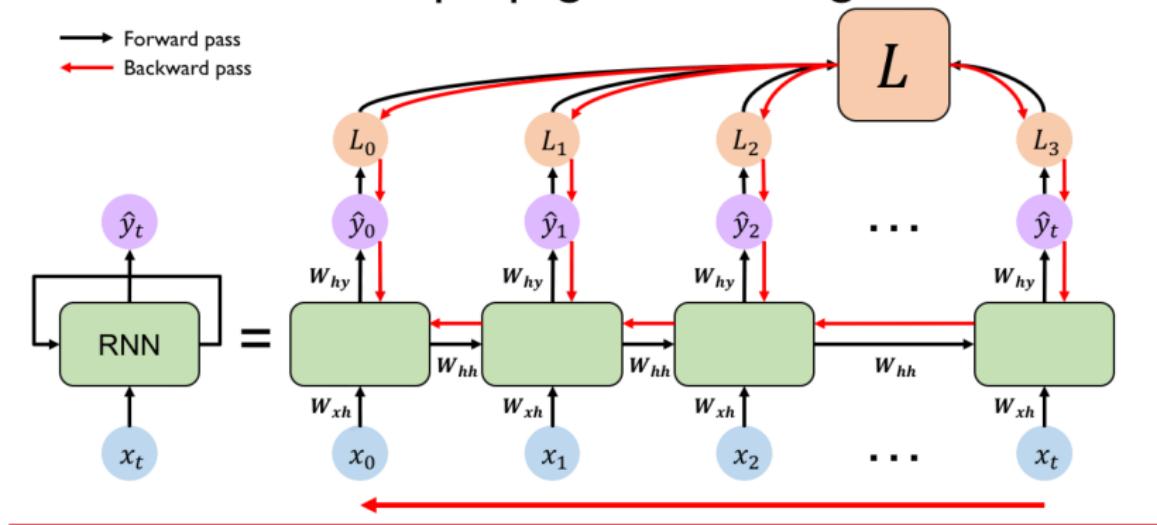
Backpropagation through time

RNNs: Backpropagation Through Time



Backpropagation through time

RNNs: Backpropagation Through Time



Backpropagation through time: Long time dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

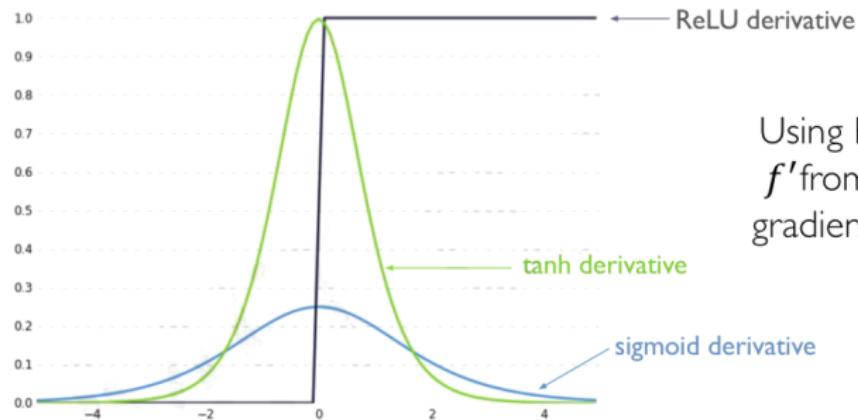


Errors due to further back time steps
have smaller and smaller gradients



Bias parameters to capture short-term
dependencies

How to solve it:



Using ReLU prevents
 f' from shrinking the
gradients when $x > 0$

How to solve it:

Initialize **weights** to identity matrix

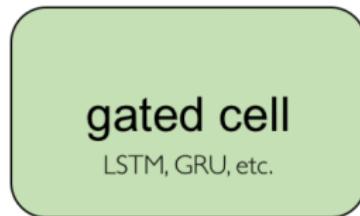
Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

How to solve it:

Idea: use a more **complex recurrent unit with gates** to control what information is passed through



Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Summary on RNN:

- RNN are well suitable for sequence modeling tasks.
- Model sequences via a recurrence relation.
- Training RNN with backpropagation through time.
- Gated cells allow let us model long time dependencies.
- We can model music generation, classification, machine translation and others.

Convolutional Neural Networks (CNN): A motivation

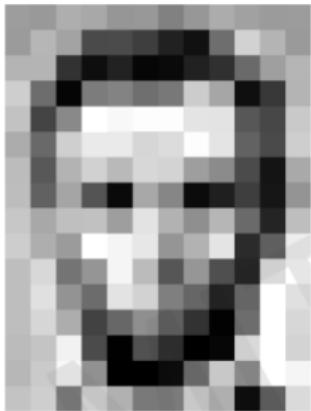
Most is all about Computer Vision, with implications on

- Facial detection and recognition.
- Healthcare, medicine and Biology.
- Self driving vehicles.

What do computers see?



What do computers see?



| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 187 | 153 | 174 | 168 | 150 | 162 | 129 | 161 | 172 | 161 | 155 | 156 |
| 185 | 182 | 163 | 74 | 75 | 62 | 55 | 17 | 110 | 210 | 180 | 154 |
| 186 | 180 | 50 | 14 | 34 | 4 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 261 | 237 | 239 | 239 | 238 | 227 | 87 | 71 | 201 |
| 172 | 106 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 168 | 139 | 75 | 20 | 169 |
| 189 | 97 | 166 | 64 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 166 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 206 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 316 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 238 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 180 | 202 | 237 | 145 | 6 | 9 | 18 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 129 | 200 | 175 | 13 | 96 | 218 |

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

What do computers see?



| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 187 | 183 | 174 | 168 | 160 | 152 | 129 | 193 | 172 | 163 | 165 | 156 |
| 185 | 182 | 163 | 74 | 75 | 62 | 55 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 60 | 14 | 54 | 6 | 10 | 33 | 48 | 106 | 189 | 181 |
| 206 | 109 | 6 | 124 | 191 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 261 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 106 | 207 | 233 | 233 | 214 | 220 | 239 | 238 | 98 | 74 | 206 |
| 188 | 68 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 76 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 168 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 206 | 174 | 158 | 252 | 236 | 231 | 149 | 178 | 238 | 43 | 55 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 234 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 64 | 103 | 143 | 96 | 90 | 2 | 109 | 249 | 215 |
| 187 | 196 | 238 | 71 | 1 | 81 | 47 | 0 | 6 | 217 | 259 | 211 |
| 189 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 136 | 243 | 236 |
| 196 | 206 | 123 | 207 | 177 | 121 | 129 | 200 | 175 | 13 | 96 | 218 |

What the computer sees

| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 197 | 153 | 174 | 168 | 160 | 152 | 129 | 151 | 172 | 161 | 195 | 196 |
| 195 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 189 | 181 |
| 206 | 109 | 5 | 124 | 191 | 111 | 120 | 204 | 166 | 15 | 66 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 106 | 207 | 233 | 233 | 214 | 220 | 239 | 238 | 98 | 74 | 206 |
| 188 | 68 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 76 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 168 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 206 | 174 | 158 | 252 | 236 | 231 | 149 | 178 | 238 | 43 | 55 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 90 | 2 | 109 | 249 | 215 |
| 187 | 196 | 238 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 189 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 136 | 243 | 236 |
| 196 | 206 | 123 | 207 | 177 | 121 | 129 | 200 | 175 | 13 | 96 | 218 |

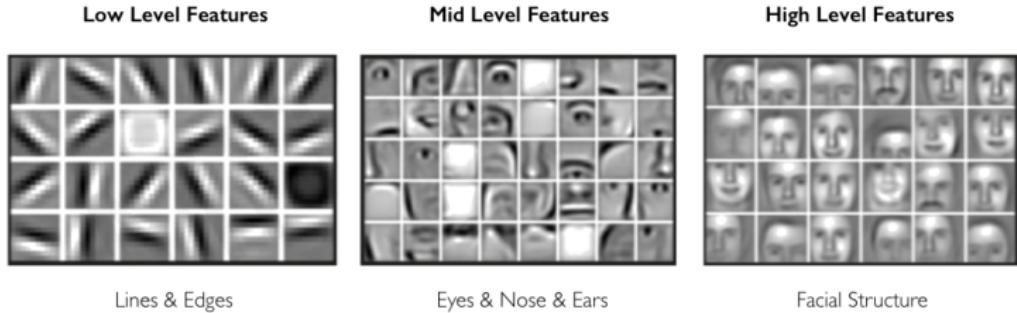
An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Tasks in computer vision

(As in other NN problems)

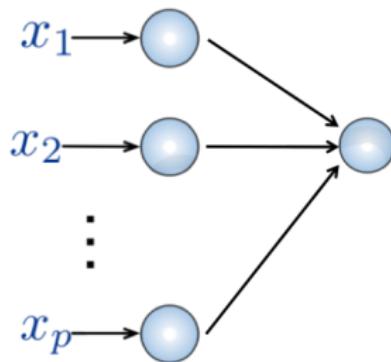
- Regression
- Classification
- High level feature detection

Features



Learning visual features

- Input:**
- 2D image
 - Vector of pixel values

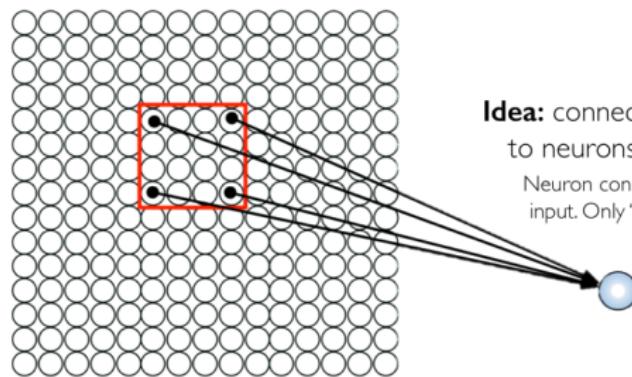


- Fully Connected:**
- Connect neuron in hidden layer to all neurons in input layer
 - No spatial information!
 - And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

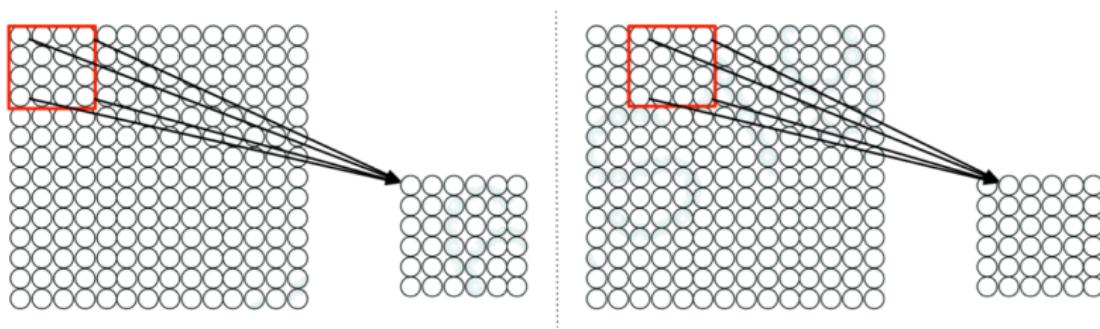
Using spatial structure

Input: 2D image.
Array of pixel values



Idea: connect patches of input
to neurons in hidden layer:
Neuron connected to region of
input. Only "sees" these values.

Using spatial structure

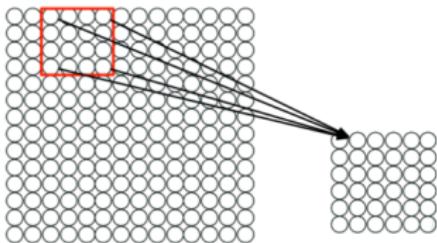


Connect patch in input layer to a single neuron in subsequent layer.

Use a sliding window to define connections.

How can we **weight** the patch to detect particular features?

Applying filters to extract features



- Filter of size 4×4 : 16 different weights
- Apply this same filter to 4×4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

Convolutional operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

The diagram shows the convolution process between a 5x5 input image and a 3x3 filter. The input image is a green grid with values: 1, 1, 1, 0, 0; 0, 1, 1, 1, 0; 0, 0, 1, 1, 1; 0, 0, 1, 1, 0; 0, 1, 1, 0, 0. The filter is a yellow grid with values: 1, 0, 1; 0, 1, 0; 1, 0, 1. A large multiplication symbol (\otimes) is positioned between the two grids. Below the image grid is the label "image". Below the filter grid is the label "filter".

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

\otimes

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

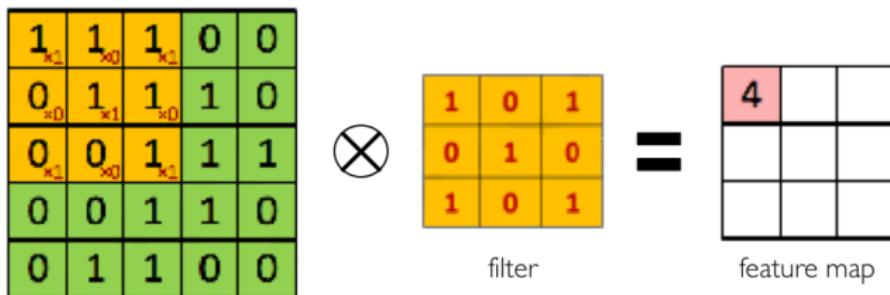
image

filter

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs...

Convolutional operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



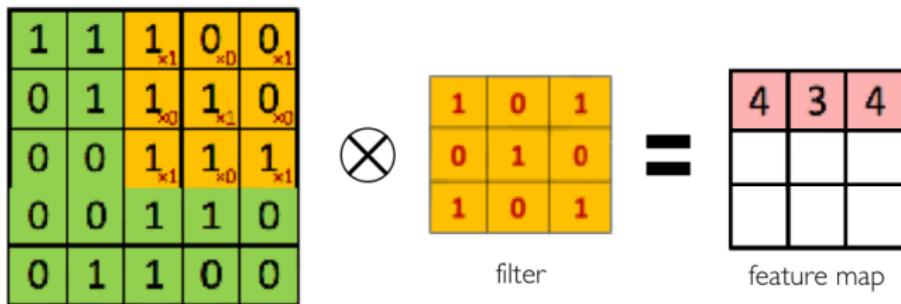
Convolutional operation

We slide the 3×3 filter over the input image, element-wise multiply, and add the outputs:

The diagram illustrates a convolutional operation. On the left is an input image represented as a 5x5 grid of green cells. The values in the cells are: Row 1: 1, 1_{x1}, 1_{x0}, 0_{x1}, 0; Row 2: 0, 1_{x0}, 1_{x1}, 1_{x0}, 0; Row 3: 0, 0_{x1}, 1_{x0}, 1_{x1}, 1; Row 4: 0, 0, 1, 1, 0; Row 5: 0, 1, 1, 0, 0. The cells in the second row are highlighted in yellow, and the cells in the third row are also highlighted in yellow, indicating the receptive field of each output unit in the feature map. To the right of the input image is a multiplication symbol (\otimes). Next to it is a 3x3 grid labeled "filter" containing the values: 1, 0, 1; 0, 1, 0; 1, 0, 1. To the right of the filter is an equals sign (=). Finally, to the right of the equals sign is a 3x3 grid labeled "feature map" containing the values: 4, 3, (empty cell); (empty cell), (empty cell), (empty cell); (empty cell), (empty cell), (empty cell).

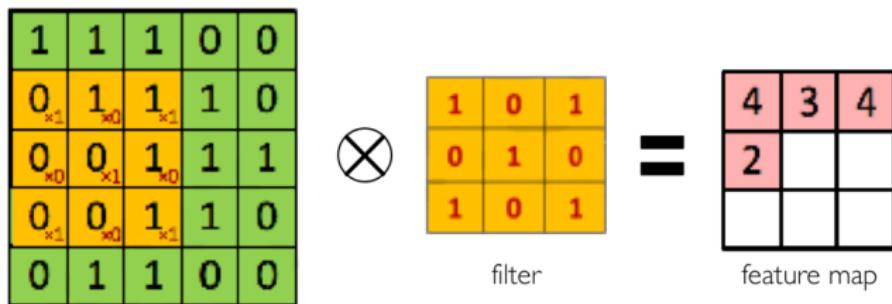
Convolutional operation

We slide the 3×3 filter over the input image, element-wise multiply, and add the outputs:



Convolutional operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Convolutional operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The diagram illustrates a convolutional operation. On the left is an input image represented as a 5x5 grid of green squares. The values in the grid are: Row 1: 1, 1, 1, 0, 0; Row 2: 0, 1 (with $\times 1$ below it), 1 (with $\times 0$ below it), 1 (with $\times 1$ below it), 0; Row 3: 0, 0 (with $\times 0$ below it), 1 (with $\times 1$ below it), 1 (with $\times 0$ below it), 1; Row 4: 0, 0 (with $\times 1$ below it), 1 (with $\times 0$ below it), 1 (with $\times 1$ below it), 0; Row 5: 0, 1, 1, 0, 0. In the center is a 3x3 filter represented as a yellow grid with red numbers: 1, 0, 1; 0, 1, 0; 1, 0, 1. To the right of the filter is an equals sign (=). To the right of the equals sign is a 3x3 feature map represented as a pink grid: 4, 3, 4; 2, 4, (empty); (empty), (empty), (empty).

| | | | | |
|---|-----------------|-----------------|-----------------|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 $\times 1$ | 1 $\times 0$ | 1 $\times 1$ | 0 |
| 0 | 0 $\times 0$ | 1 $\times 1$ | 1 $\times 0$ | 1 |
| 0 | 0 $\times 1$ | 1 $\times 0$ | 1 $\times 1$ | 0 |
| 0 | 1 | 1 | 0 | 0 |

\otimes

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

=

| | | |
|---|---|---|
| 4 | 3 | 4 |
| 2 | 4 | |
| | | |

feature map

Convolutional operation

We slide the 3×3 filter over the input image, element-wise multiply, and add the outputs:

The diagram illustrates a convolutional operation. On the left is an input image represented as a 5x5 grid of green squares. The values in the grid are: Row 1: 1, 1, 1, 0, 0; Row 2: 0, 1, 1_{x1}, 1_{x0}, 0_{x1}; Row 3: 0, 0, 1_{x0}, 1_{x1}, 1_{x0}; Row 4: 0, 0, 1_{x1}, 1_{x0}, 0_{x1}; Row 5: 0, 1, 1, 0, 0. The middle part shows a yellow square labeled "filter" containing the values 1, 0, 1; 0, 1, 0; 1, 0, 1. To the right of the filter is an equals sign (=). To the right of the equals sign is a pink square labeled "feature map" containing the values 4, 3, 4; 2, 4, 3; and two empty squares below them.

| | | | | |
|---|---|-----------------|-----------------|-----------------|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 _{x1} | 1 _{x0} | 0 _{x1} |
| 0 | 0 | 1 _{x0} | 1 _{x1} | 1 _{x0} |
| 0 | 0 | 1 _{x1} | 1 _{x0} | 0 _{x1} |
| 0 | 1 | 1 | 0 | 0 |

\otimes

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

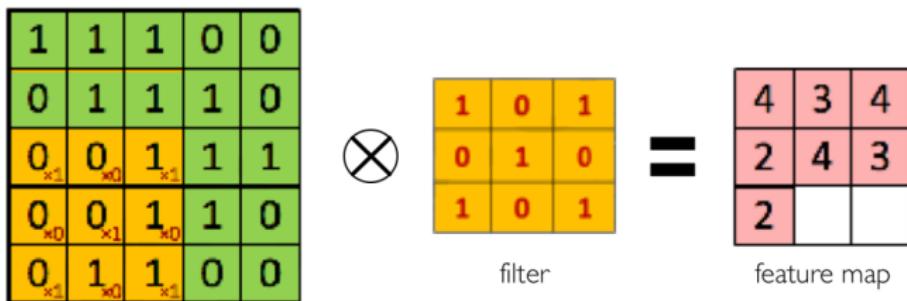
=

| | | |
|---|---|---|
| 4 | 3 | 4 |
| 2 | 4 | 3 |
| | | |

feature map

Convolutional operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Convolutional operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The diagram illustrates a convolutional operation. On the left, an input image is shown as a 5x5 grid of green cells. Some cells contain numerical values (1, 0) or mathematical expressions ($\times 1$, $\times 0$). A 3x3 yellow filter is applied to the image. The result is a feature map on the right, which is a 3x3 grid of pink cells containing numerical values (4, 3, 4; 2, 4, 3; 2, 3, blank).

Input image (5x5):

| | | | | |
|---|---|------------|------------|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | $\times 1$ | $\times 0$ | 1 |
| 0 | 0 | $\times 0$ | $\times 1$ | 1 |
| 0 | 1 | $\times 1$ | $\times 0$ | 0 |

filter (3x3):

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

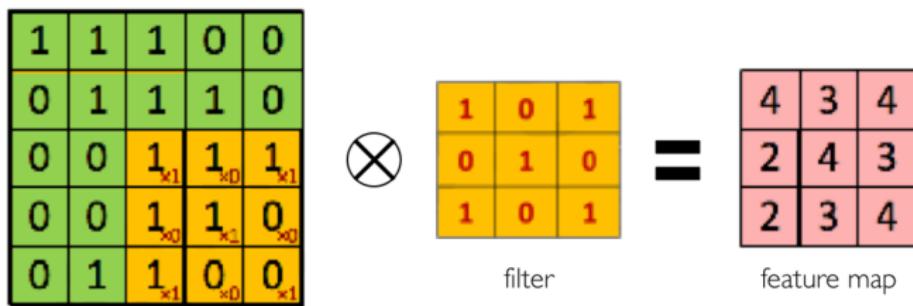
=

feature map (3x3):

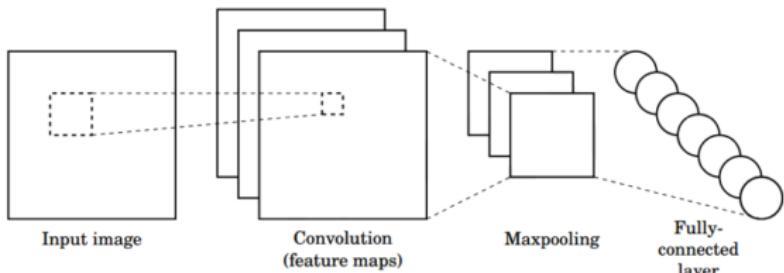
| | | |
|---|---|---|
| 4 | 3 | 4 |
| 2 | 4 | 3 |
| 2 | 3 | |

Convolutional operation

We slide the 3×3 filter over the input image, element-wise multiply, and add the outputs:



CNNs for Classification



- 1. Convolution:** Apply filters to generate feature maps.
- 2. Non-linearity:** Often ReLU.
- 3. Pooling:** Downsampling operation on each feature map.

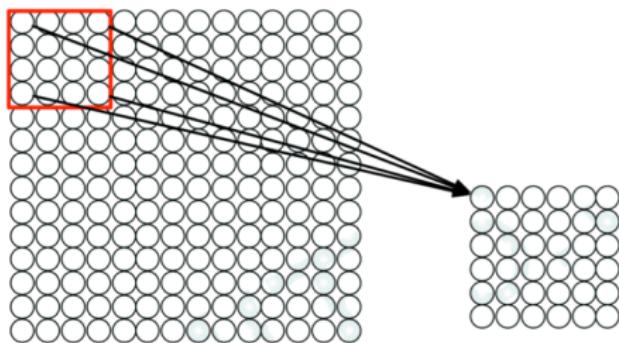
`tf.keras.layers.Conv2D`

`tf.keras.activations.*`

`tf.keras.layers.MaxPool2D`

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity



$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

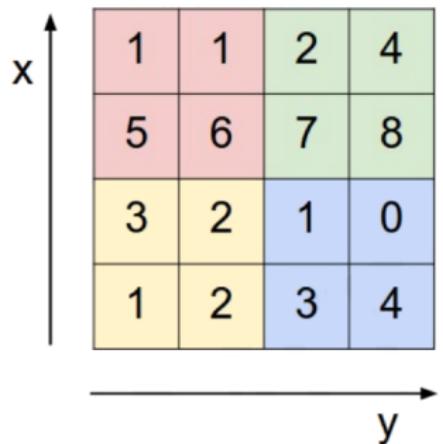
 `tf.keras.layers.Conv2D`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

- 1) applying a window of weights
- 2) computing linear combinations
- 3) activating with non-linear function

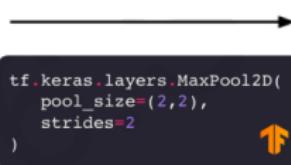
Pooling



Pooling

max pool with 2x2 filters
and stride 2

```
tf.keras.layers.MaxPool2D(  
    pool_size=(2,2),  
    strides=2  
)
```

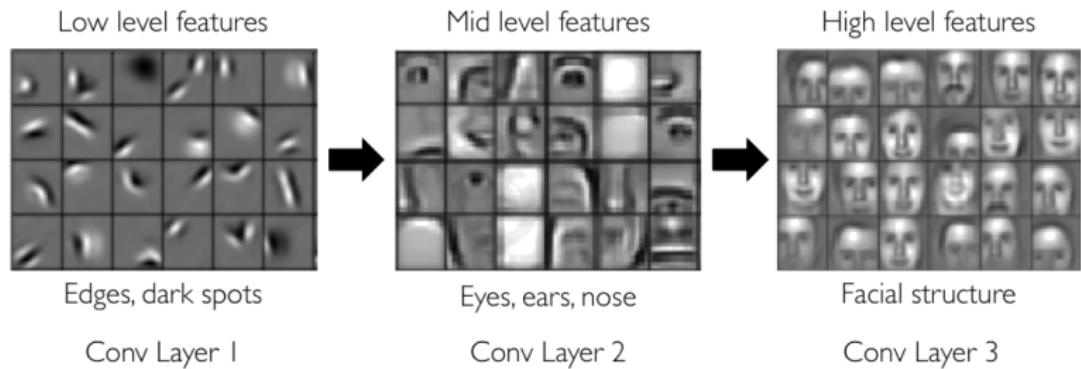


| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

- 1) Reduced dimensionality
- 2) Spatial invariance

How else can we downsample and preserve spatial invariance?

Representation Learning in Deep CNNs



Summary on CNNs

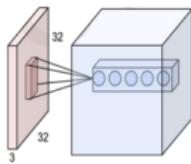
Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



CNNs

- CNN architecture
- Application to classification
- ImageNet



Applications

- Segmentation, image captioning, control
- Security, medicine, robotics



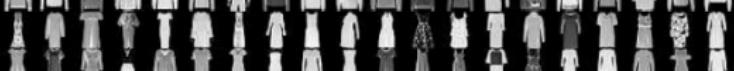
Images datasets: MNIST

<http://yann.lecun.com/exdb/mnist/>



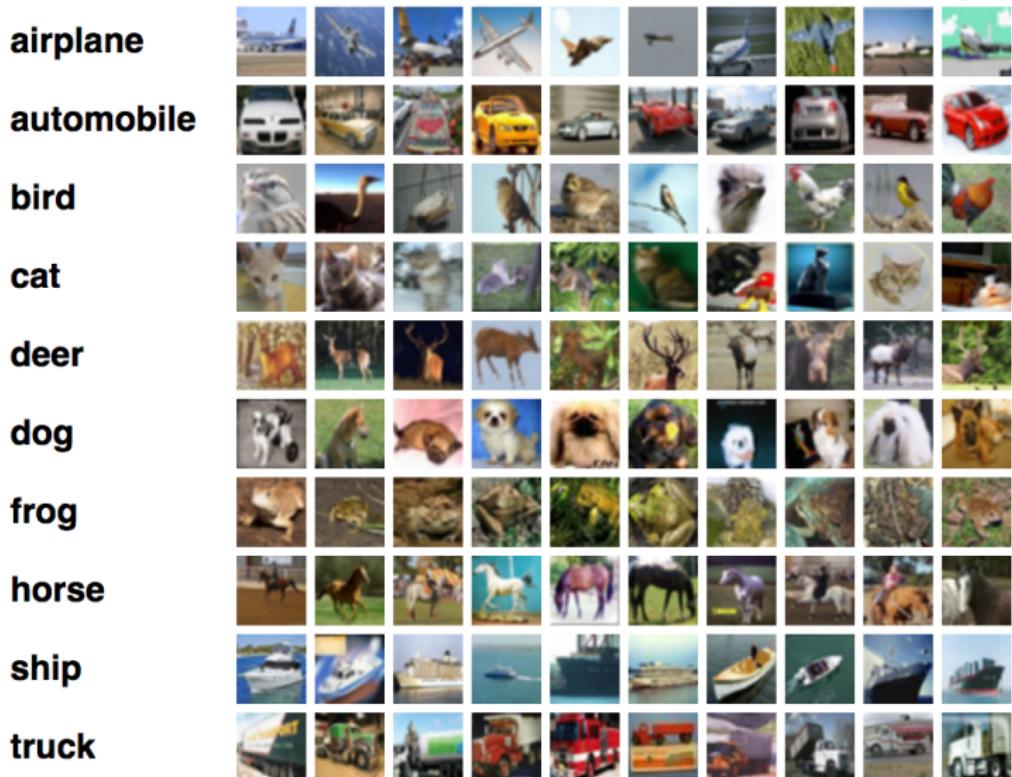
Images datasets: Fashion-MNIST

<https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/>

| Label | Description | Examples |
|-------|-------------|--|
| 0 | T-Shirt/Top |  |
| 1 | Trouser |  |
| 2 | Pullover |  |
| 3 | Dress |  |
| 4 | Coat |  |
| 5 | Sandals |  |
| 6 | Shirt |  |
| 7 | Sneaker |  |
| 8 | Bag |  |

Images datasets: CIFAR

<https://www.cs.toronto.edu/~kriz/cifar.html>



Images datasets: IMAGENET

<http://www.image-net.org/>

