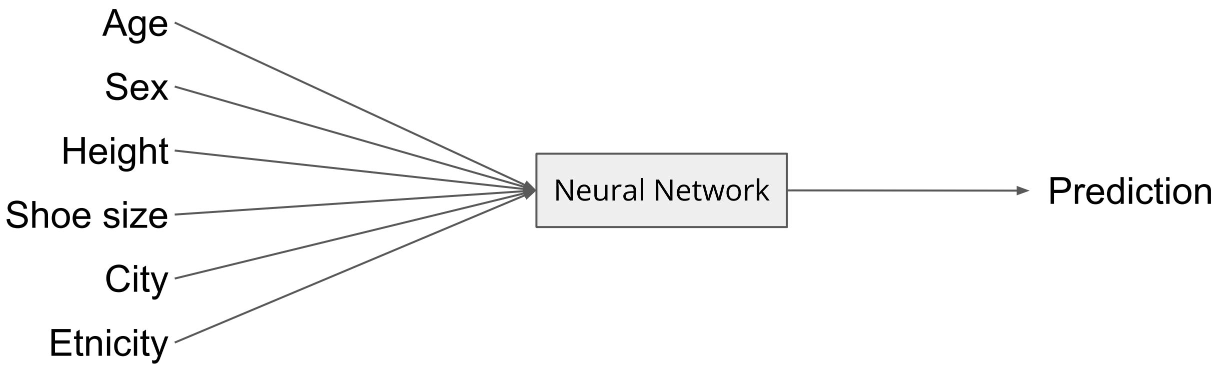
Università degli
Studi di Milano-Bicocca

Convolutional Neural Networks

Prof. Flavio Piccoli - Dott. Mirko Paolo Barbato

So far..

..we used datasets with (mostly) unrelated variables

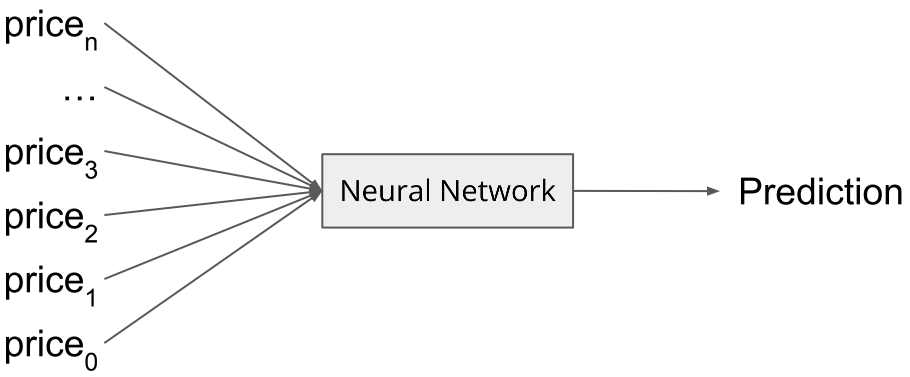


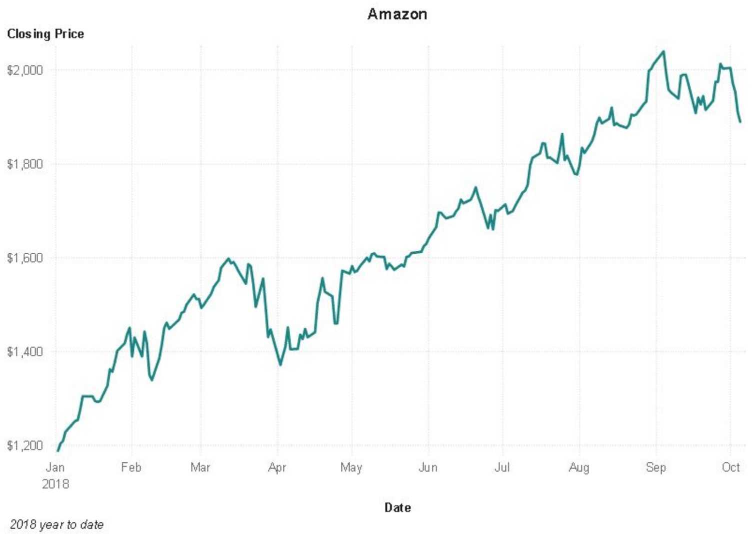
But in real world data exist many types of data that have neighborhood dependency

Time-Series

- Time-series: same variable changing in the time

Example: the price of a stock

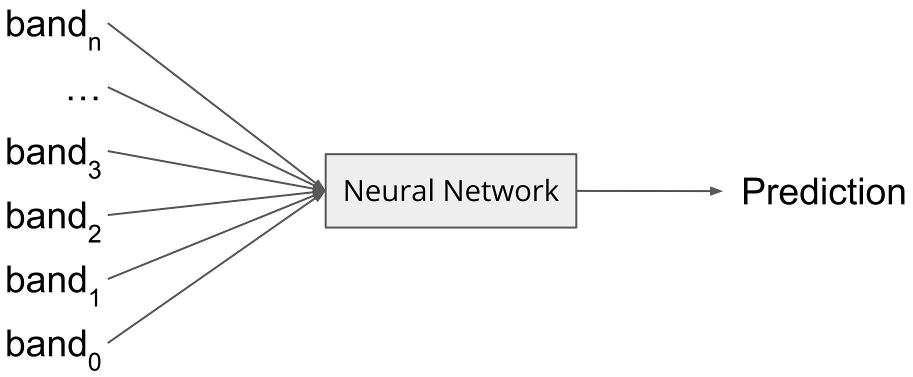


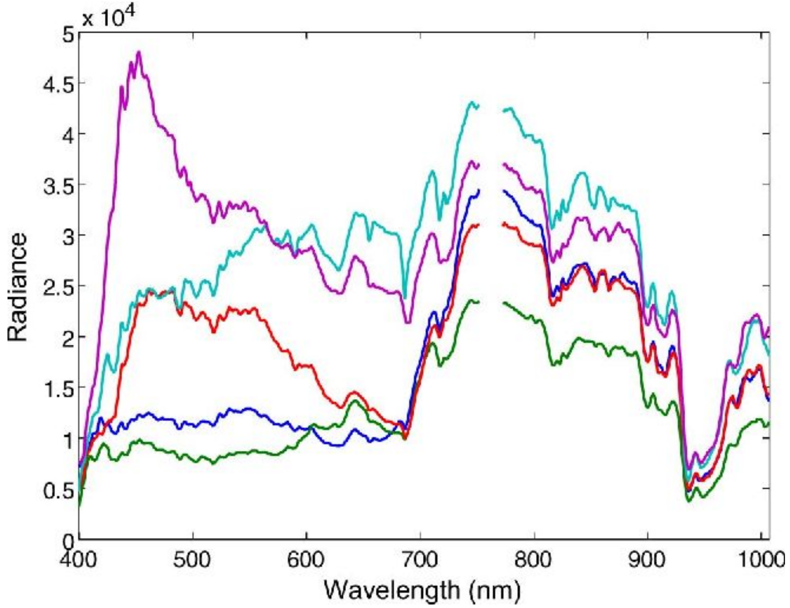


Signals

- Same quantity acquired in different frequencies, spatial locations, etc.

Example: the hyperspectral signal

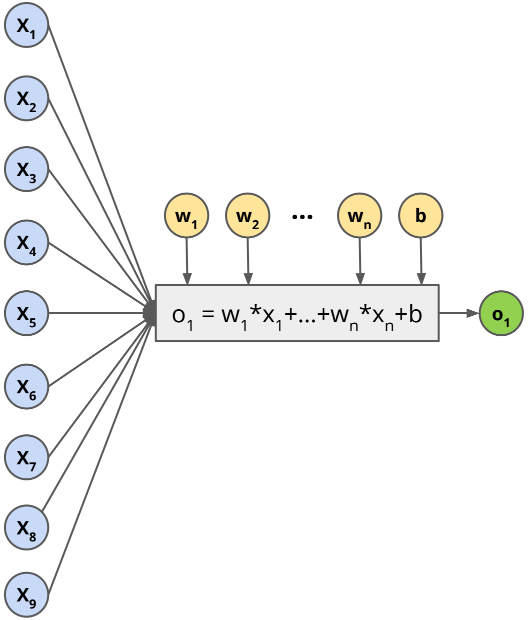




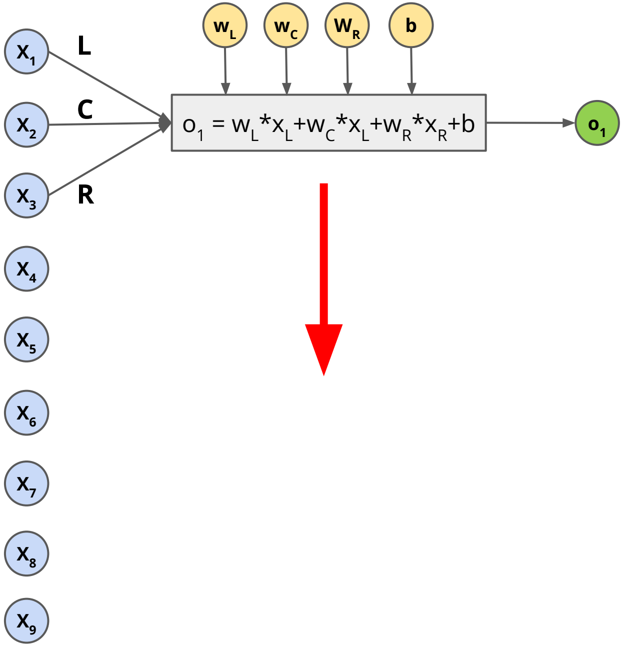
Can we use something smarter than NNs?

- Yes!
- We can use Convolutional Neural Networks
- They exploit local dependencies

Linear Layer



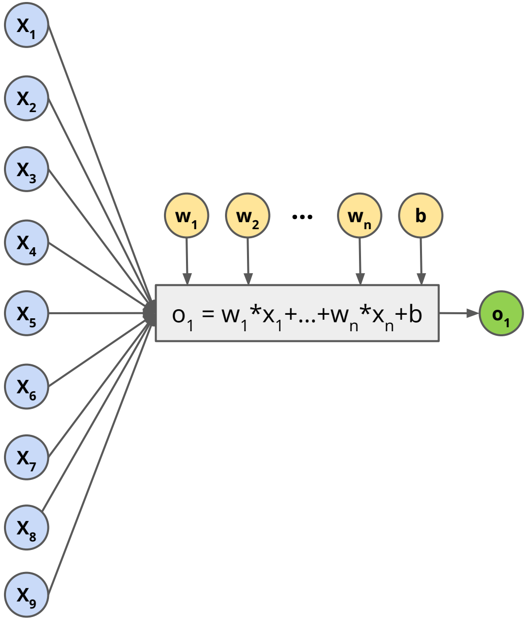
Convolutive Layer



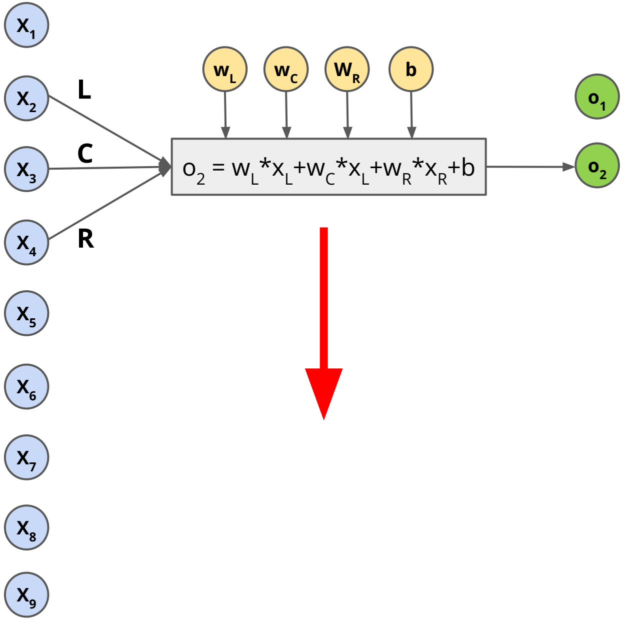
Can we use something smarter than NNs?

- Yes!
- We can use Convolutional Neural Networks
- They exploit local dependencies

Linear Layer



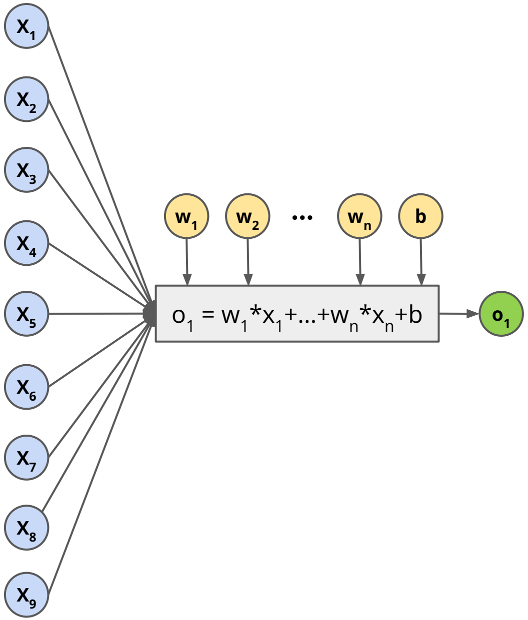
Convolutive Layer



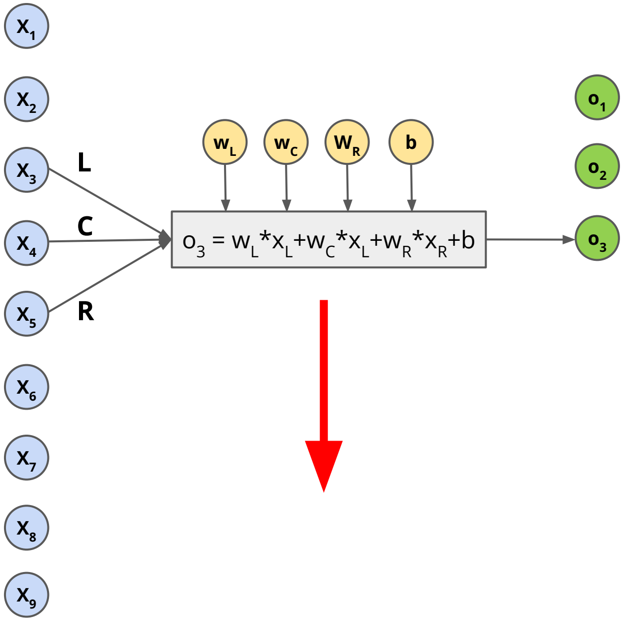
Can we use something smarter than NNs?

- Yes!
- We can use Convolutional Neural Networks
- They exploit local dependencies

Linear Layer

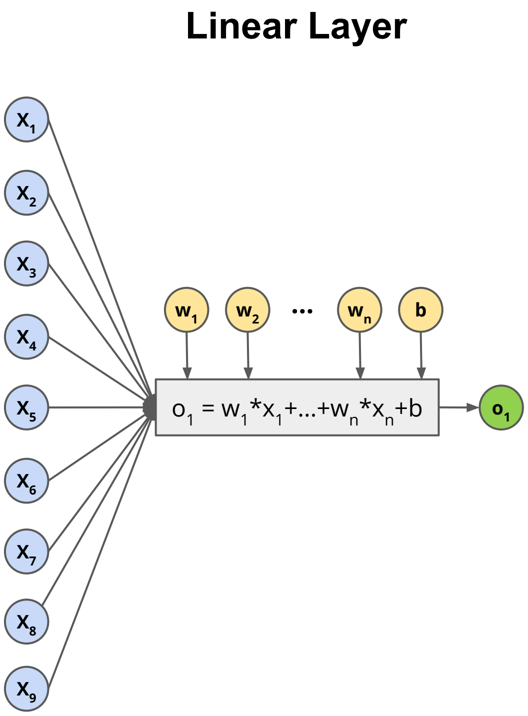


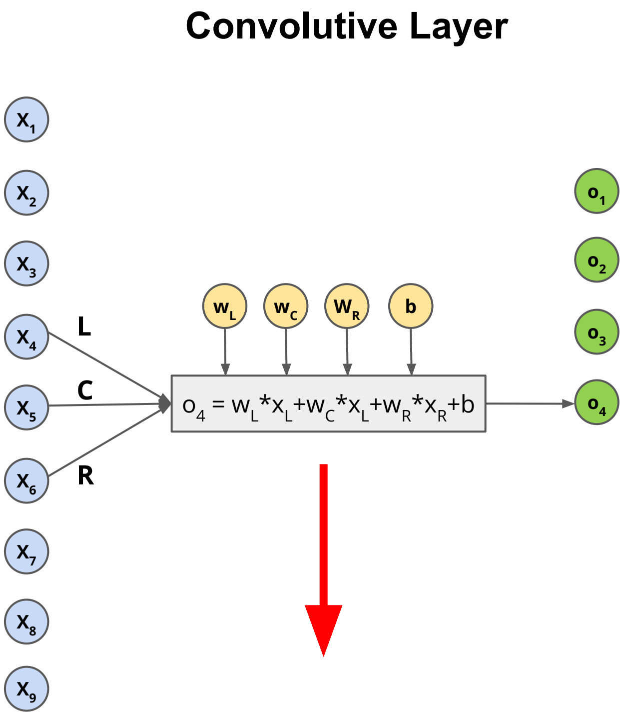
Convolutive Layer



Can we use something smarter than NNs?

- Yes!
- We can use Convolutional Neural Networks
- They exploit local dependencies

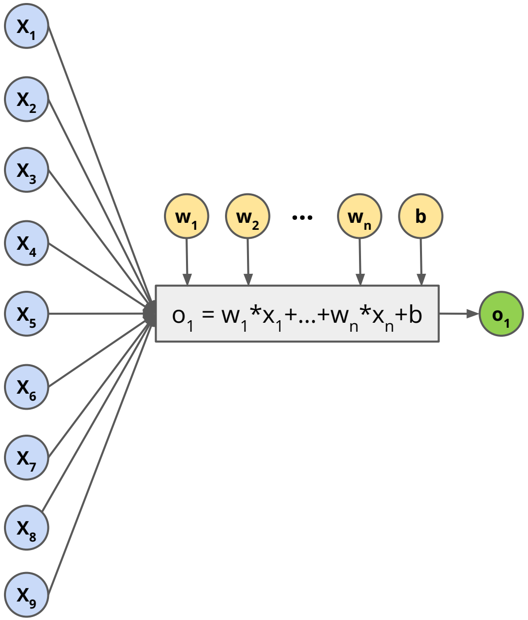




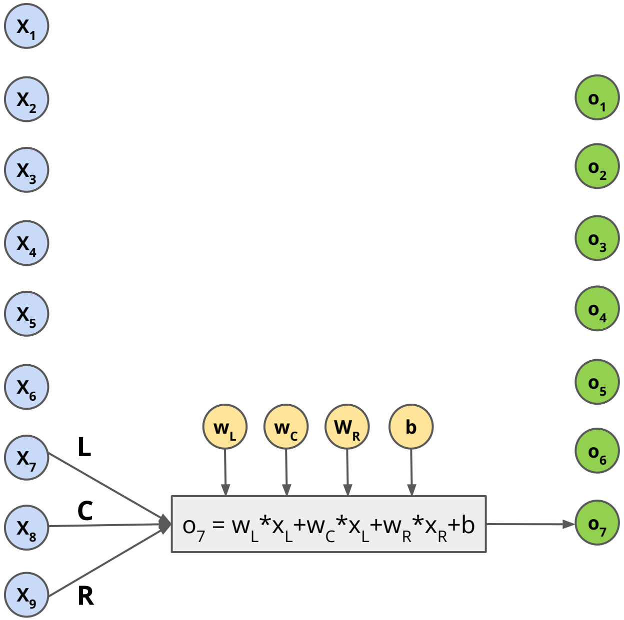
Can we use something smarter than NNs?

- Yes!
- We can use Convolutional Neural Networks
- They exploit local dependencies

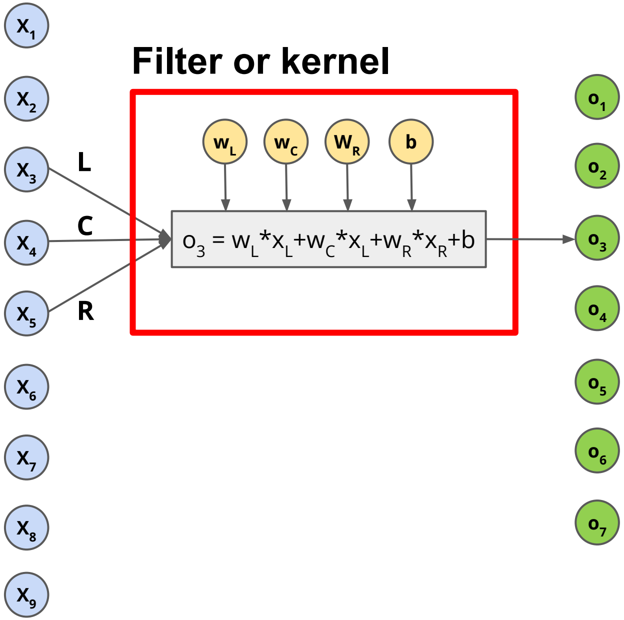
Linear Layer



Convolutive Layer



Convolutive layer - ConvLayer1D

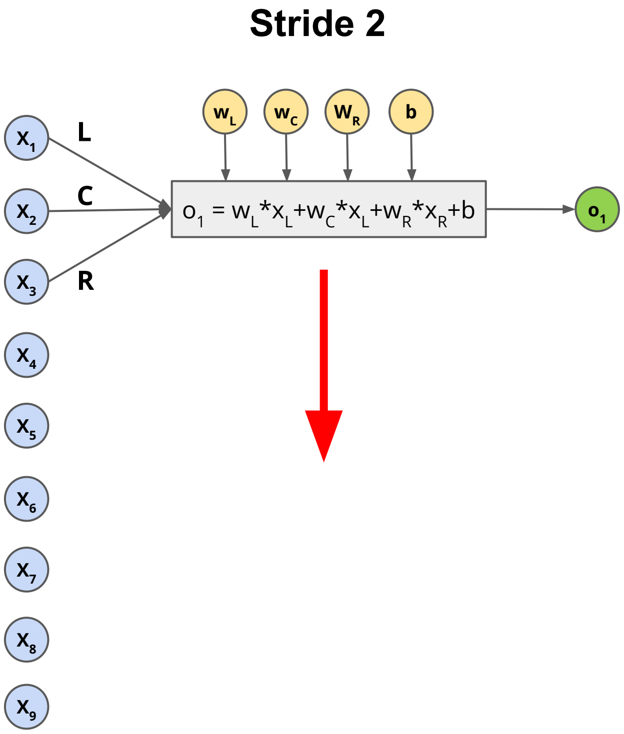


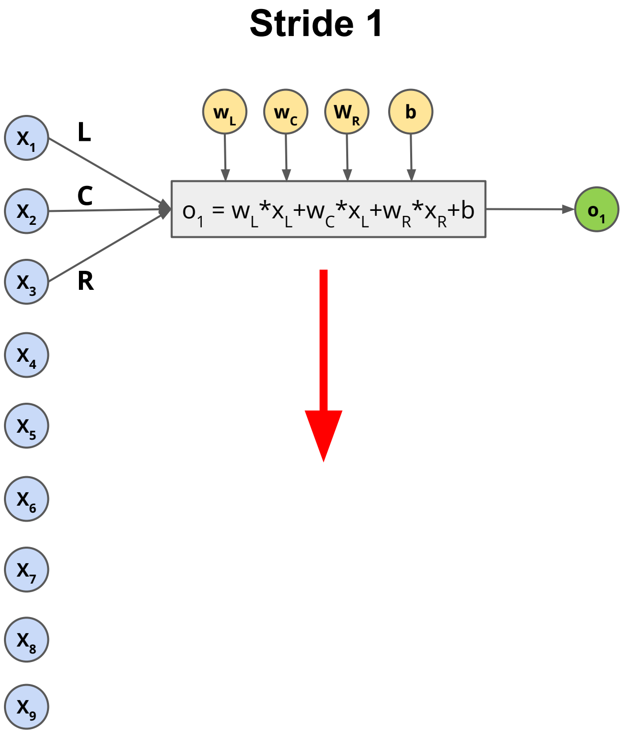
kernel size = number of neighbours considered at once. Typically 3.

stride = step of application. Usually 1 or 2.

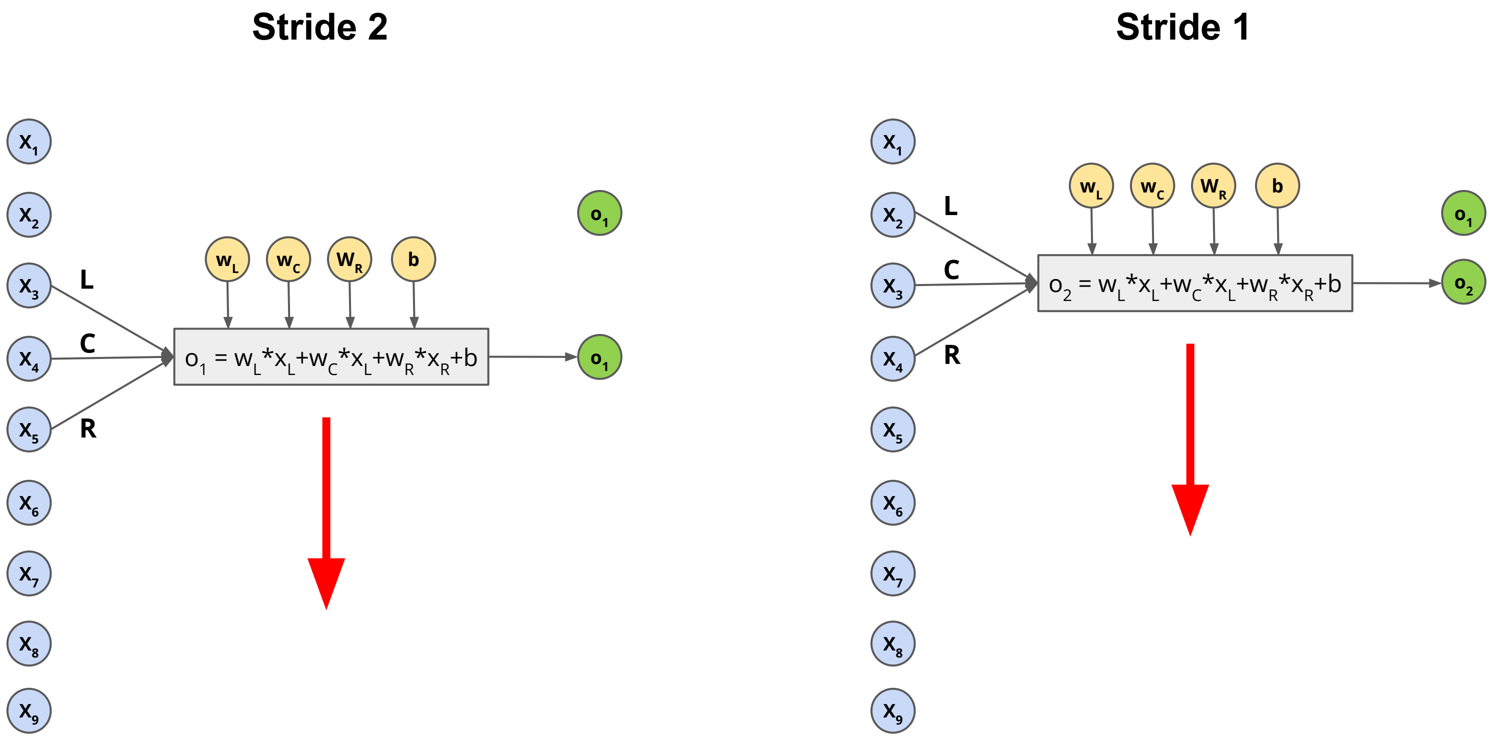
Stride

- it's the length of the step by which the filter moves

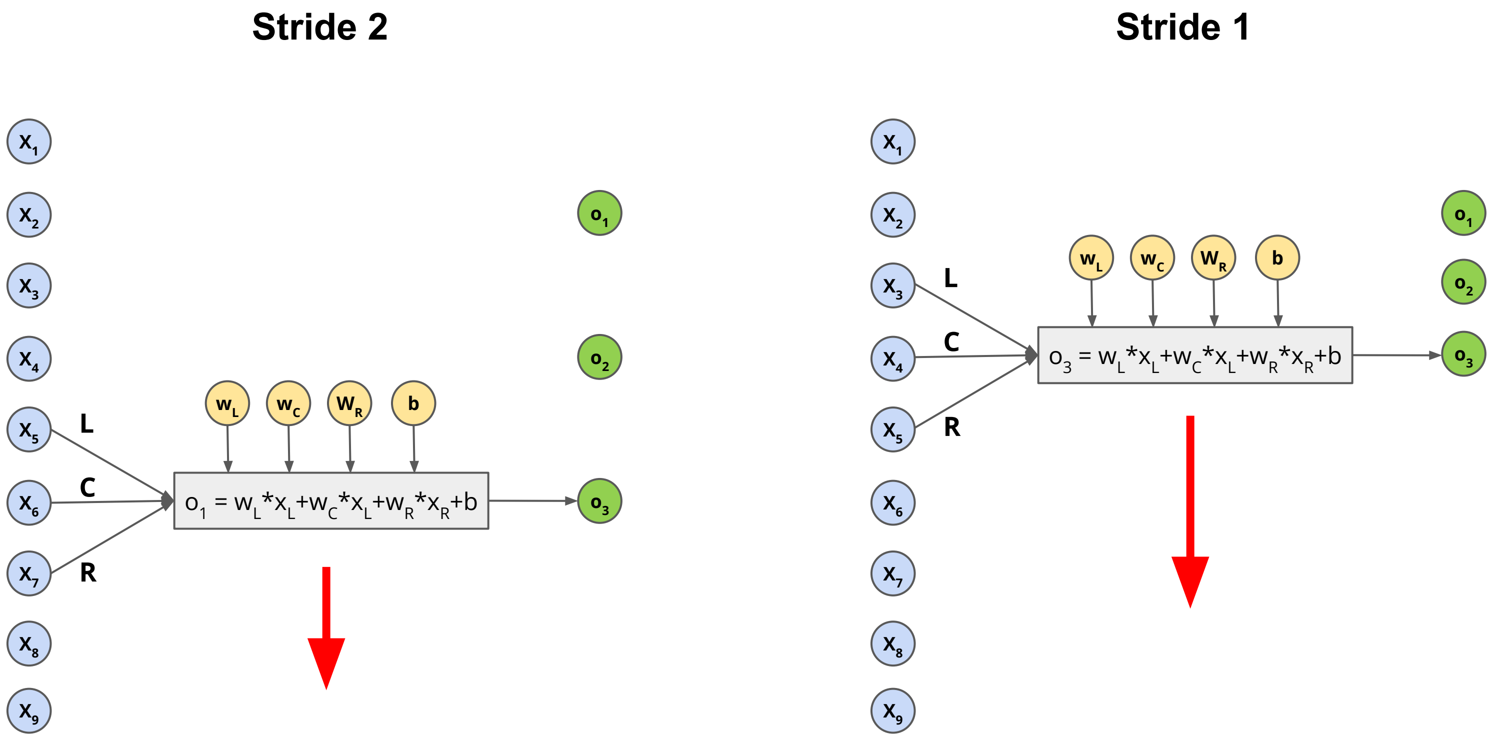




Stride

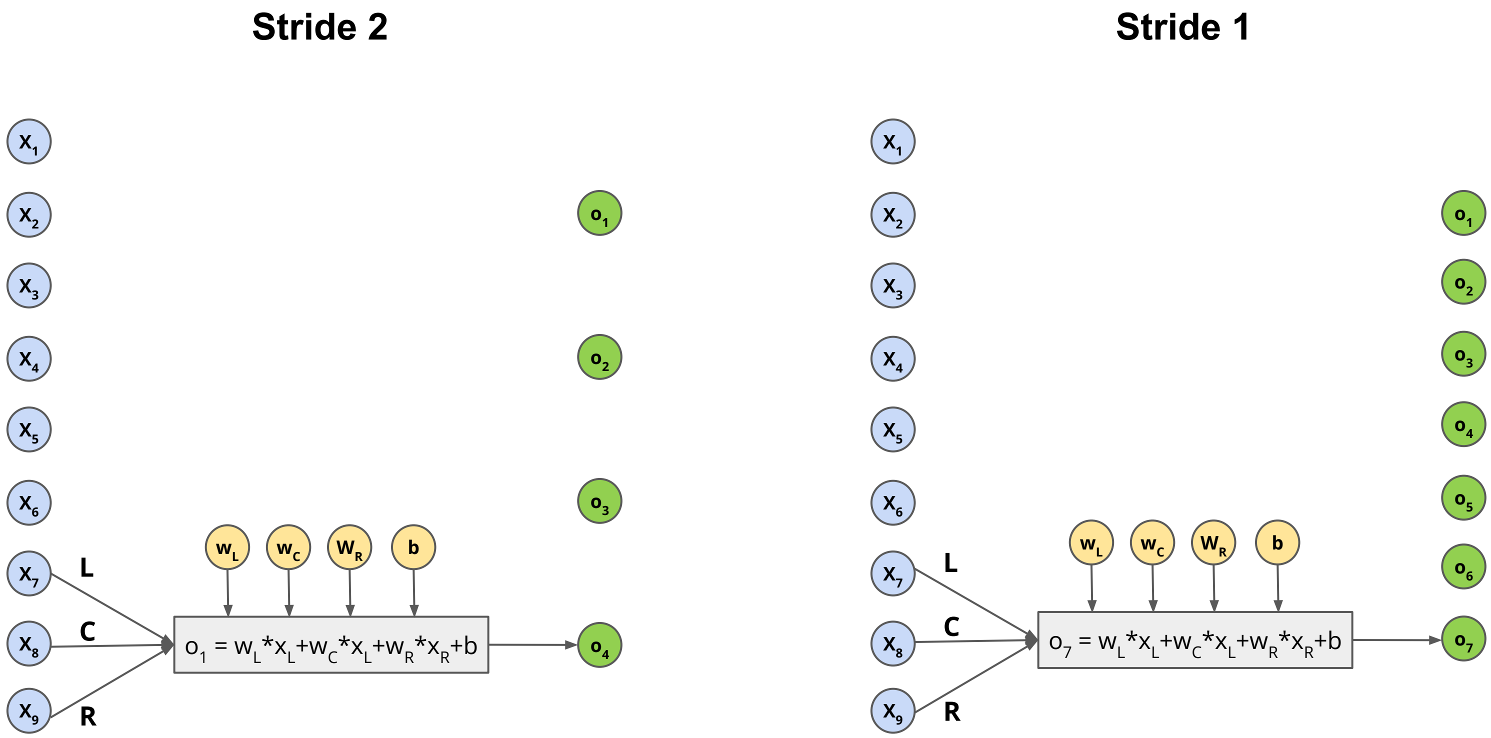

Stride 2
Stride 1
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
L
C
R
w_L w_C W_R b
$o_1 = w_L*x_L+w_C*x_L+w_R*x_R+b$
o_1
$o_2 = w_L*x_L+w_C*x_L+w_R*x_R+b$
o_1 o_2

Stride


Stride 2
Stride 1
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
L
C
R
w_L
w_C
w_R
b
$o_1 = w_L*x_L+w_C*x_L+w_R*x_R+b$
$o_3 = w_L*x_L+w_C*x_L+w_R*x_R+b$
o_1
o_2
o_3

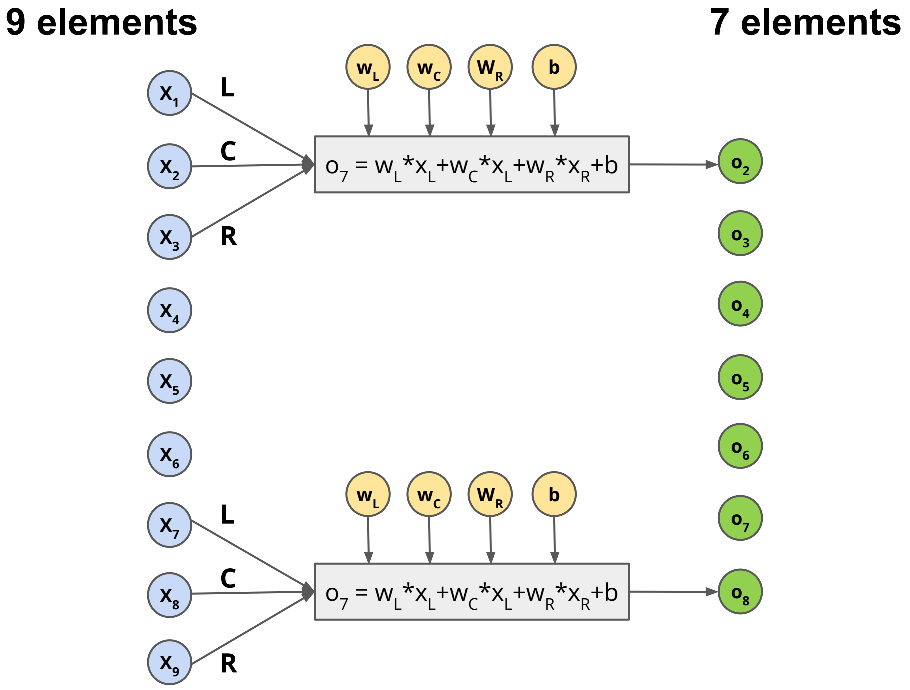
Stride

- stride 2 halves the size



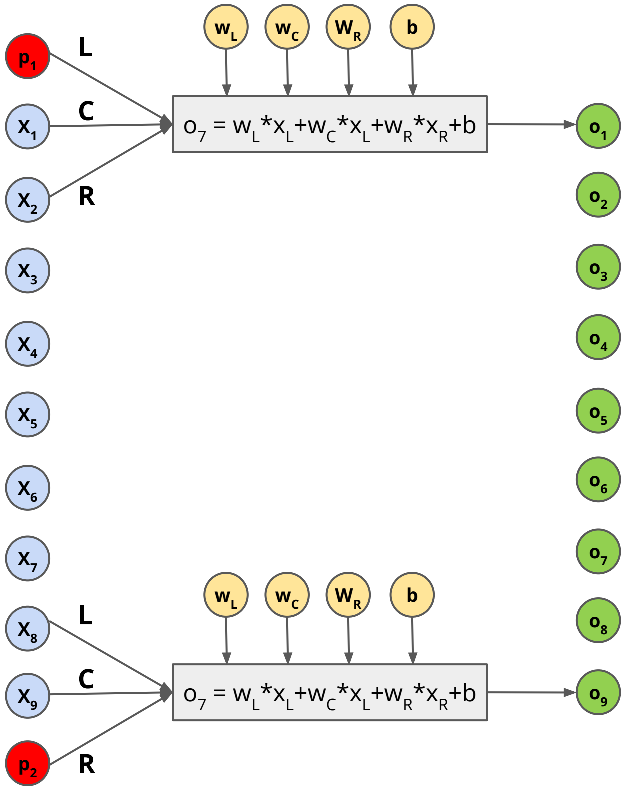
Padding

- Suppose you want an output having the same size of the input
- You need to add padding



Padding

- Suppose you want an output having the same size of the input
- You need to add padding



9 elements
+ 2 pad.

9 elements

There are several types of padding:

- constant
- reflect
- replicate
- circular

Default is 'constant' with value 0

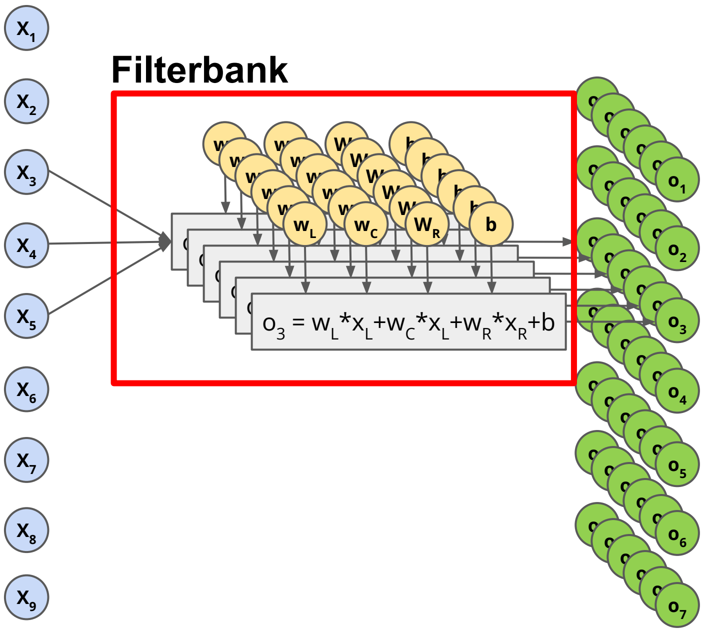
To maintain the same dimension, pad should be:

$$pad = \frac{kernel\_size - 1}{2}$$

Convolutive layer - ConvLayer1D

- Of course we can use multiple filters at once and obtain several activations

torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=**1,** padding=**1**)



Number of input channels

Input usually has 1.

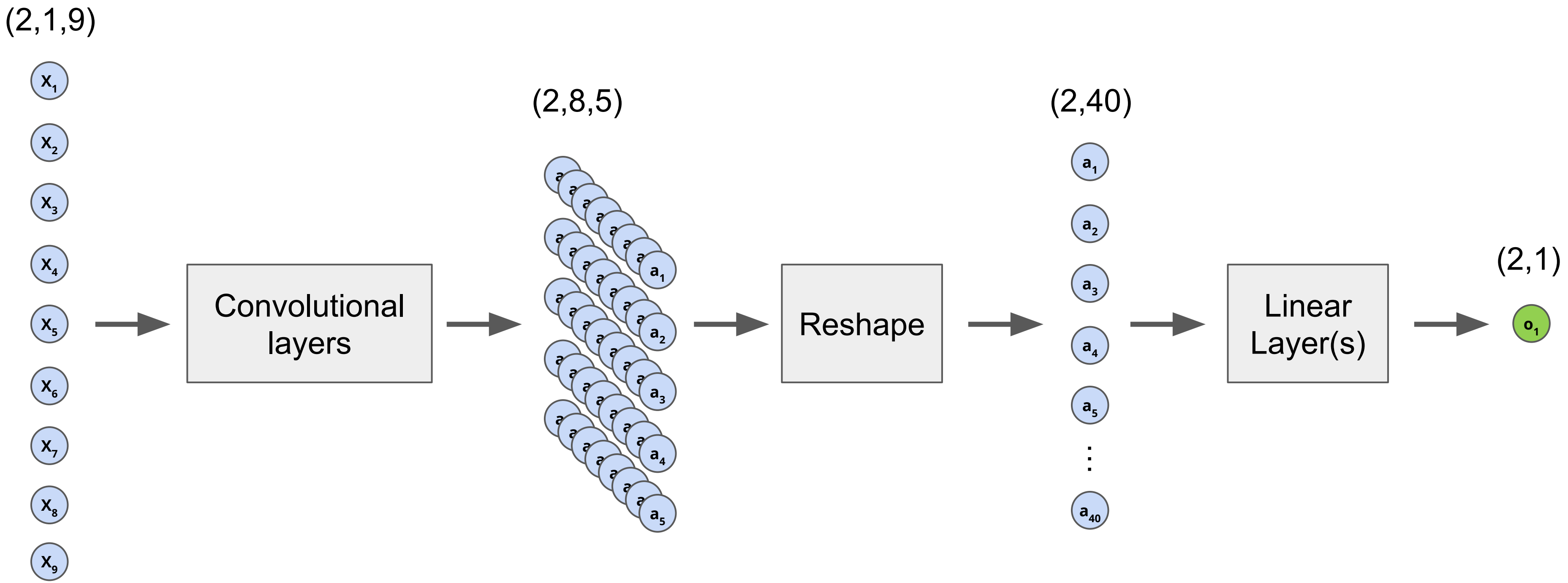
Number of filters

Determines the

number of outputs

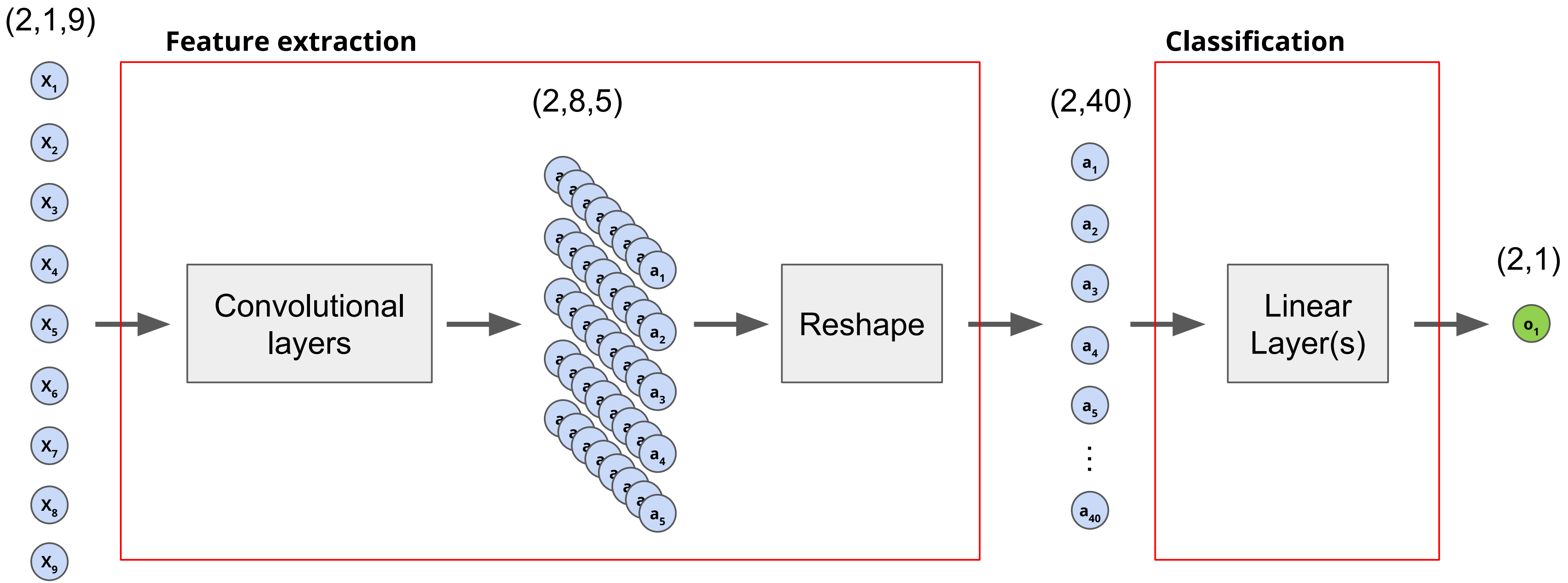
Structure of a CNN

- Suppose to have a batch of 2 samples.
  - Each sample is a signal of 9 elements (1ch)
- Suppose that convolutional layers produce activations of 5 elements having 8 channels



Structure of a CNN

- Feature extraction uses convolutional layers to extract meaningful features
- Classification uses fully connected layers (linear layers) to take a decision



Example in Pytorch

```
class CNN(nn.Module):

    def __init__(self):
        # initialize super class
        super(CNN, self).__init__()
        # define conv layers
        self.layer1 = nn.Conv1d( 1, 32, kernel_size=3, stride=2, padding=1)
        self.layer2 = nn.ReLU()
        self.layer3 = nn.Conv1d(32, 64, kernel_size=3, stride=2, padding=1)
        self.layer4 = nn.ReLU()
        self.layer5 = nn.Conv1d(64, 128, kernel_size=3, stride=2, padding=1)
        self.layer6 = nn.ReLU()
        # define linear layer
        self.layer7 = nn.Linear(384, 1)


    def forward(self, x):
        # apply convolution layers
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.layer6(x)
        # reshape from (10, 128, 3) to (10, 384)
        x = x.reshape(x.shape[0], -1)
        # fully connected
        x = self.layer7(x)
        # return output
        return x
```

```
# create cnn
cnn = CNN()
# create fake input
inp = torch.rand(10, 1, 20)
# compute output
out = cnn(inp)
```

Example in Pytorch

```python
class CNN(nn.Module):

    def __init__(self):
        # initialize super class
        super(CNN, self).__init__()
        # define conv layers
        self.layer1 = nn.Conv1d( 1, 32, kernel_size=3, stride=2, padding=1)
        self.layer2 = nn.ReLU()
        self.layer3 = nn.Conv1d(32, 64, kernel_size=3, stride=2, padding=1)
        self.layer4 = nn.ReLU()
        self.layer5 = nn.Conv1d(64, 128, kernel_size=3, stride=2, padding=1)
        self.layer6 = nn.ReLU()
        # define linear layer
        self.layer7 = nn.Linear(384, 1)


    def forward(self, x):
        # apply convolution layers
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.layer6(x)
        # reshape from (10, 128, 3) to (10, 384)
        x = x.reshape(x.shape[0], -1)
        # fully connected
        x = self.layer7(x)
        # return output
        return x
```

```python
# create cnn
cnn = CNN()
# create fake input
inp = torch.rand(10, 1, 20)
# compute output
out = cnn(inp)
```

Convolutional layers
Last activation has shape (10, 128, 3)

Reshaping
to shape (10, 384). 384 is 128*3.

Linear layers
Project 128 elements in 1c, the output

Batch normalization

This layer controls the range of the activations

A layer with weights to be learned, is easier (or sometime it doen’t work elsewise) if input data has:

- mean = 0
- stddev = 1

Running mean

$$bnorm(x) = \frac{x - mean(x)}{stddev(x)}$$

Running standard deviation

Batch normalization layer is used many times inside a network

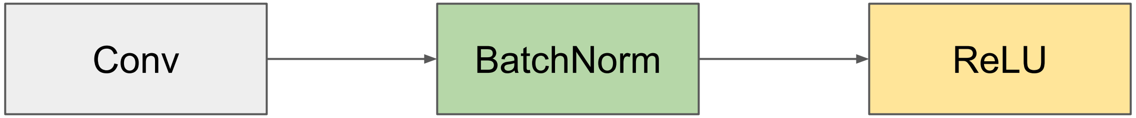
In PyTorch:

```
torch.nn.BatchNorm1d(num_features, eps=1e-05, momentum=0.1)
```

Layer order

Which is the correct order of the conv, ReLU, BatchNorm layers?

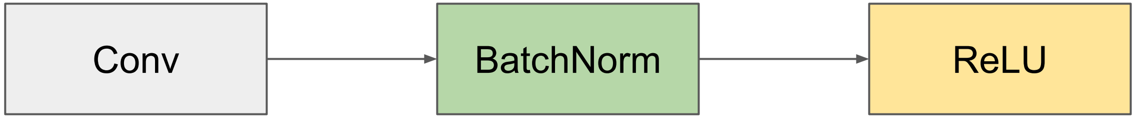
OR



Layer order

Which is the correct order of the conv, ReLU, BatchNorm layers?

OR



There is not a better order, depends from the task / dataset

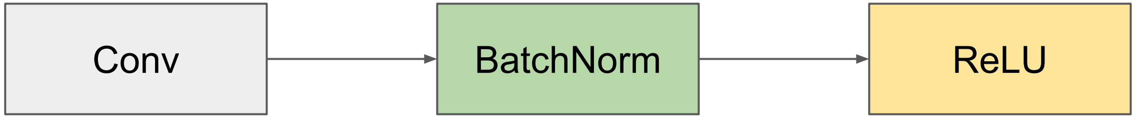
Layer order

Which is the correct order of the conv, ReLU, BatchNorm layers?

In this configuration, at the end activations are normalized

Layer order

Which is the correct order of the conv, ReLU, BatchNorm layers?



In this configuration instead, at the end activations are sparse (i.e. full of zeros)

Old approaches vs convolutional neural networks

The main difference is that in traditional approaches you have to select the most appropriate features for the task while with CNNs this is done automatically

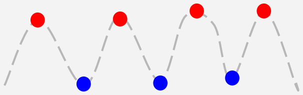
TRADITIONAL APPROACH

REQUIRES DOMAIN KNOWLEDGE FEATURE ENGINEERING

SIGNAL

→ Handcrafted feature extraction →

FEATURES



→ Classification → prediction

For example the audio signal

For example:
- Amplitude Envelope
- RMSEnergy
- Zero-Crossing Rate

DEEP-LEARNING-BASED APPROACH

DOES NOT REQUIRE ANY DOMAIN KNOWLEDGE

SIGNAL

AUTOMATIC FEATURE EXTRACTION + CLASSIFICATION

→ CNN → prediction

The hyperspectral signal

Reflectance vs Irradiance

Reflectance

- is the measure of the proportion of electromagnetic energy reflected by a surface material
- is unitless and measured in percentages

Irradiance

- is the amount of energy received by a surface per unit area
- Cameras measure irradiance

Surface reflectance

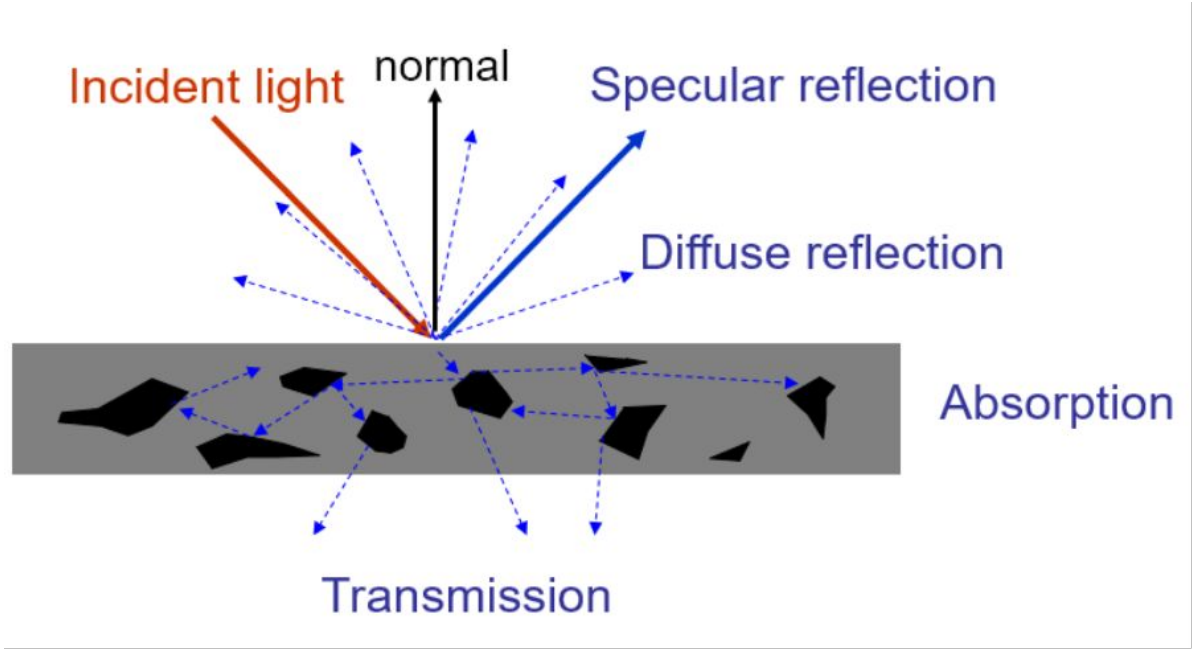


Image courtesy by Merav Mazouz &Matan Kolath

Reflectance model

We are interested in the reflected radiation

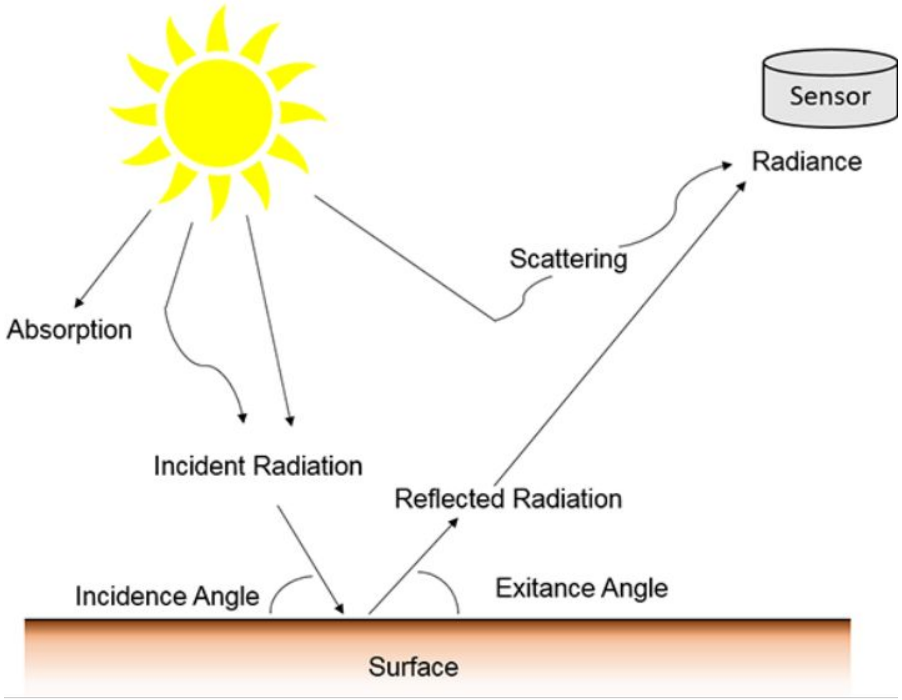


Image courtesy by Merav Mazouz &Matan Kolath

Reflectance model

In particular, we are interested in the light that is reflected by materials

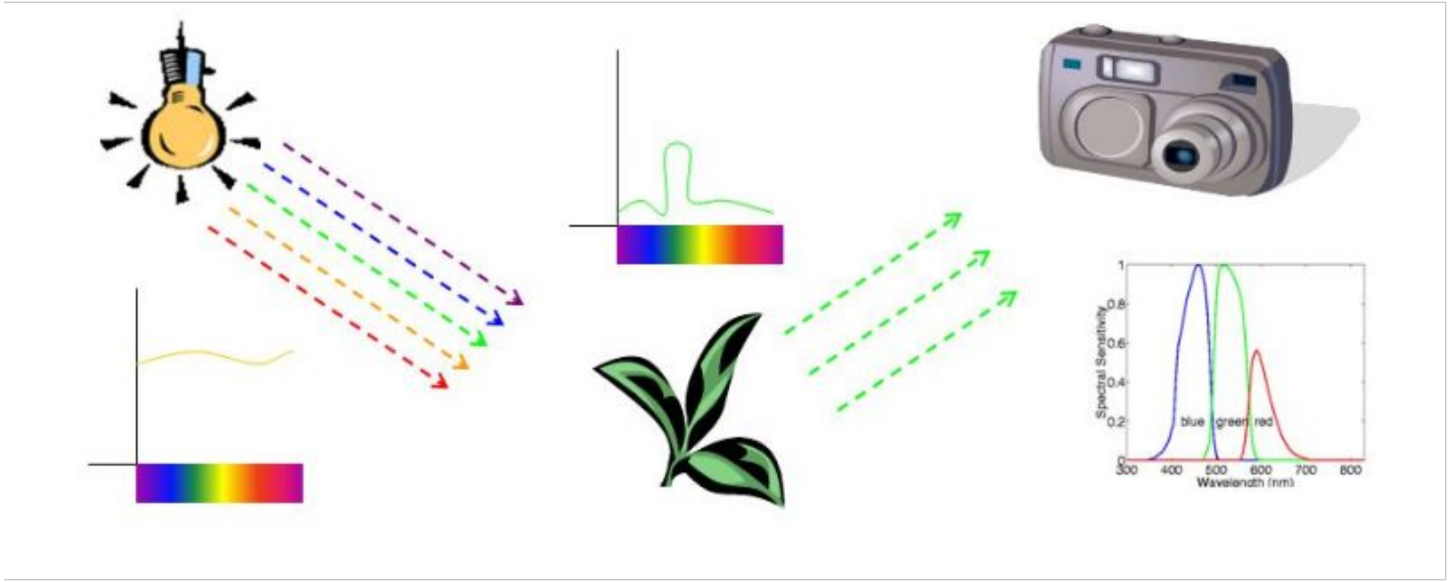
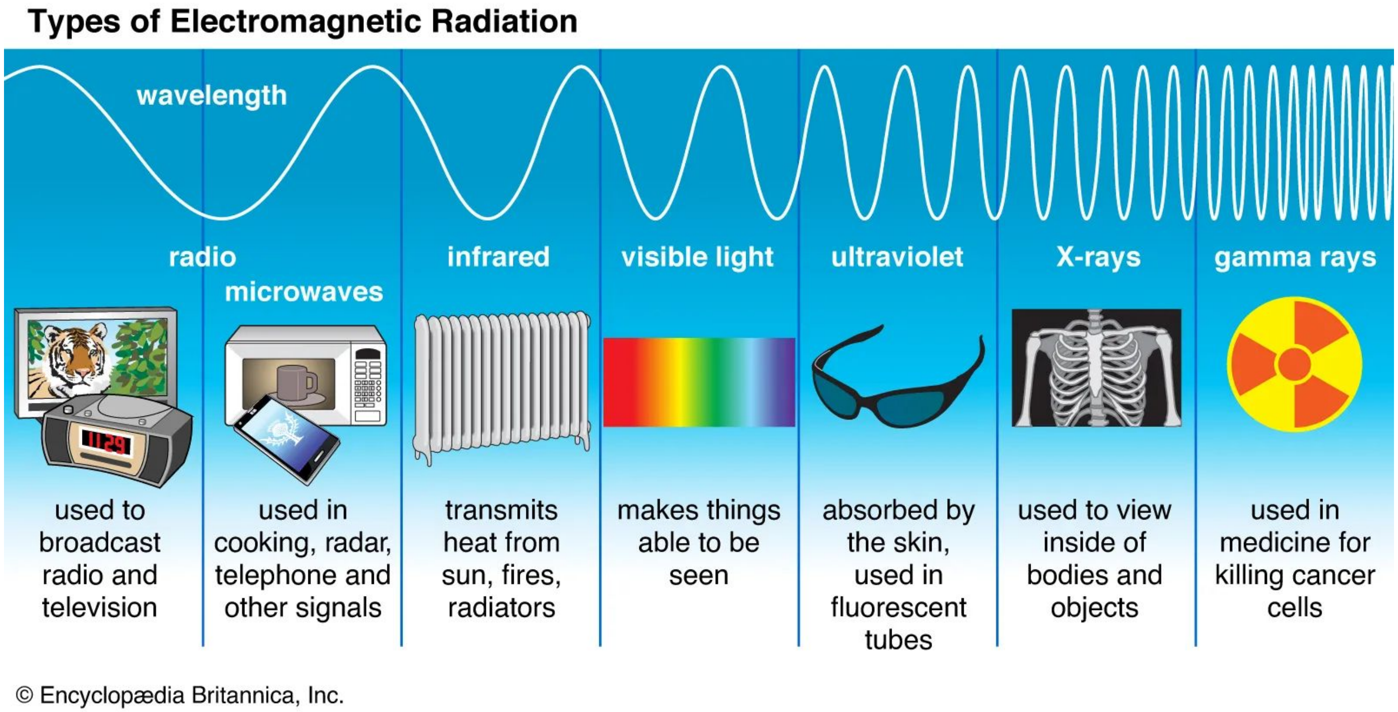
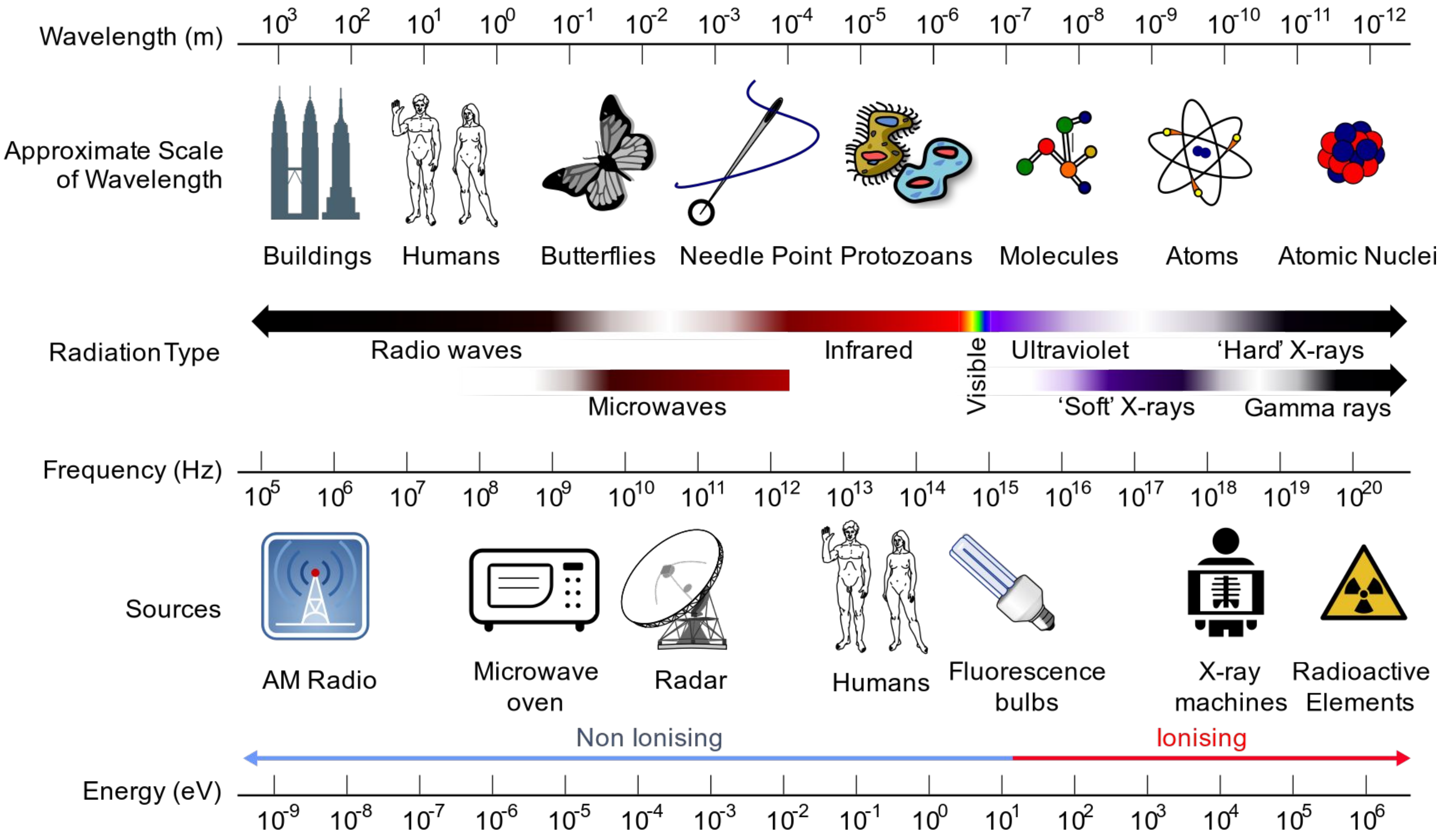


Image courtesy by Merav Mazouz &Matan Kolath

The electromagnetic spectrum

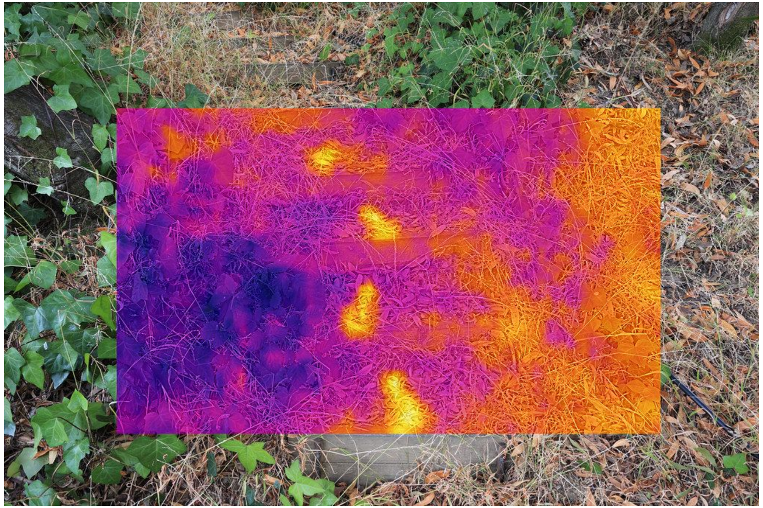


The electromagnetic spectrum



The electromagnetic spectrum

- Some animals can see a wider range of the spectrum

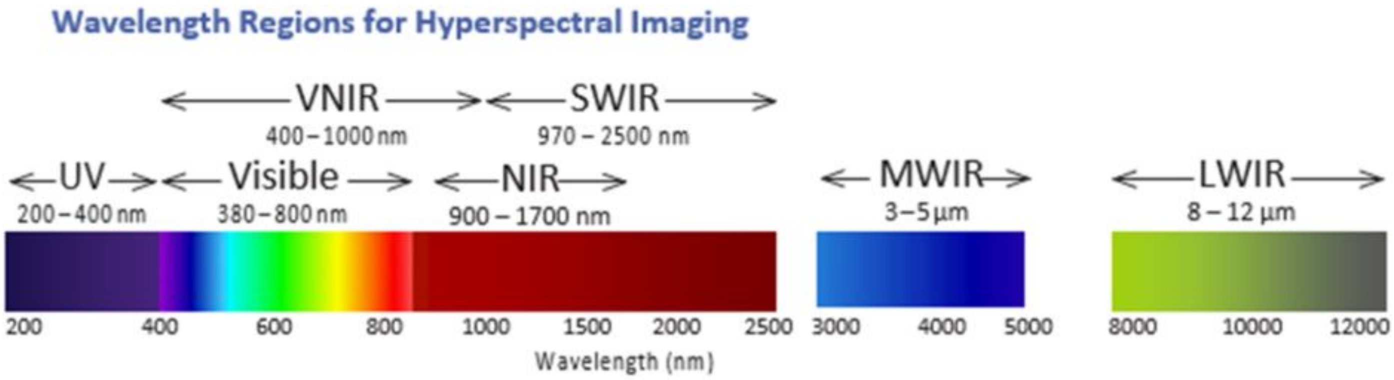


- Data doesn’t exists solely in the visible light spectrum
- Hyperspectral imaging uses data from more then just the visible light spectrum

The hyperspectral spectrum

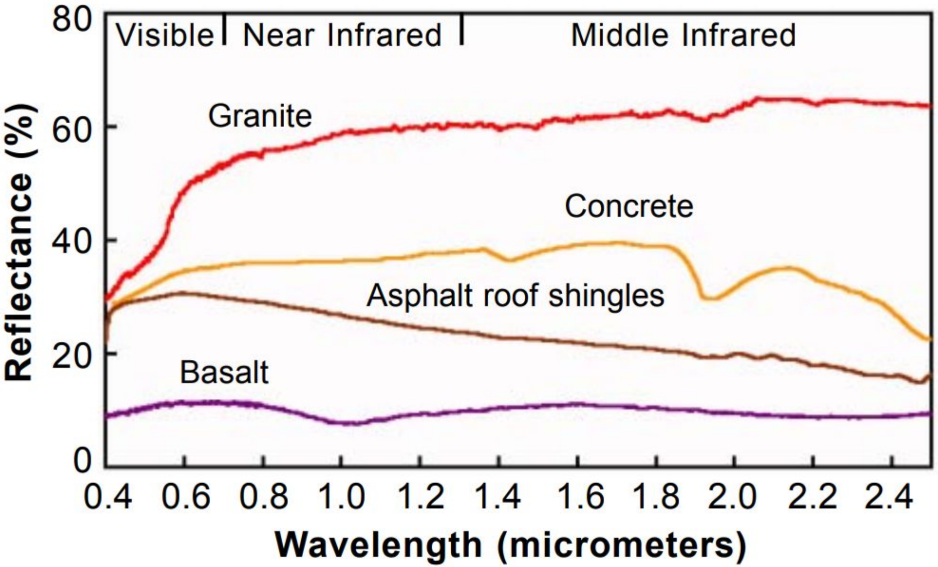
- It is composed of two portions of the EM spectrum:
  - **VNIR** = visible and near-infrared (VNIR) - [400, 1100] nm
  - **SWIR** = Short-wave infrared (SWIR) - [900 – 1700] nm

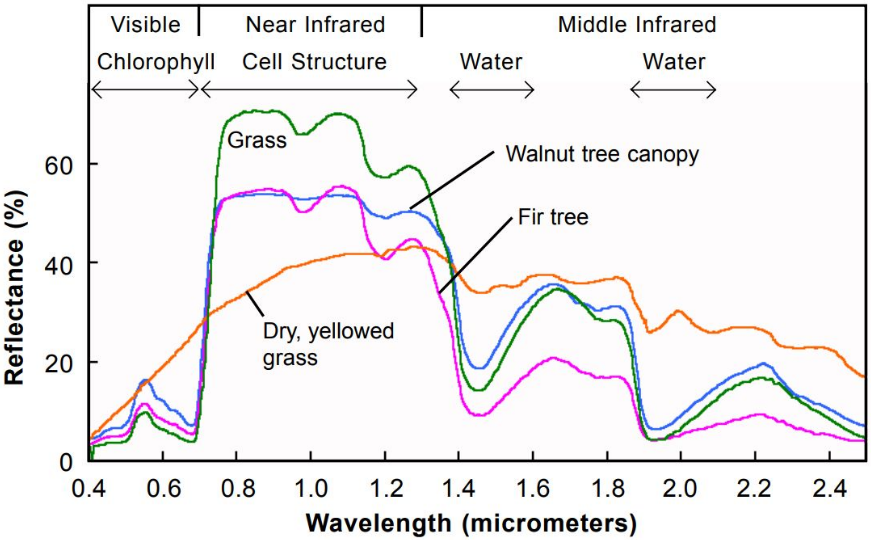
- VNIR includes the visible portion of the spectrum



Spectral signature

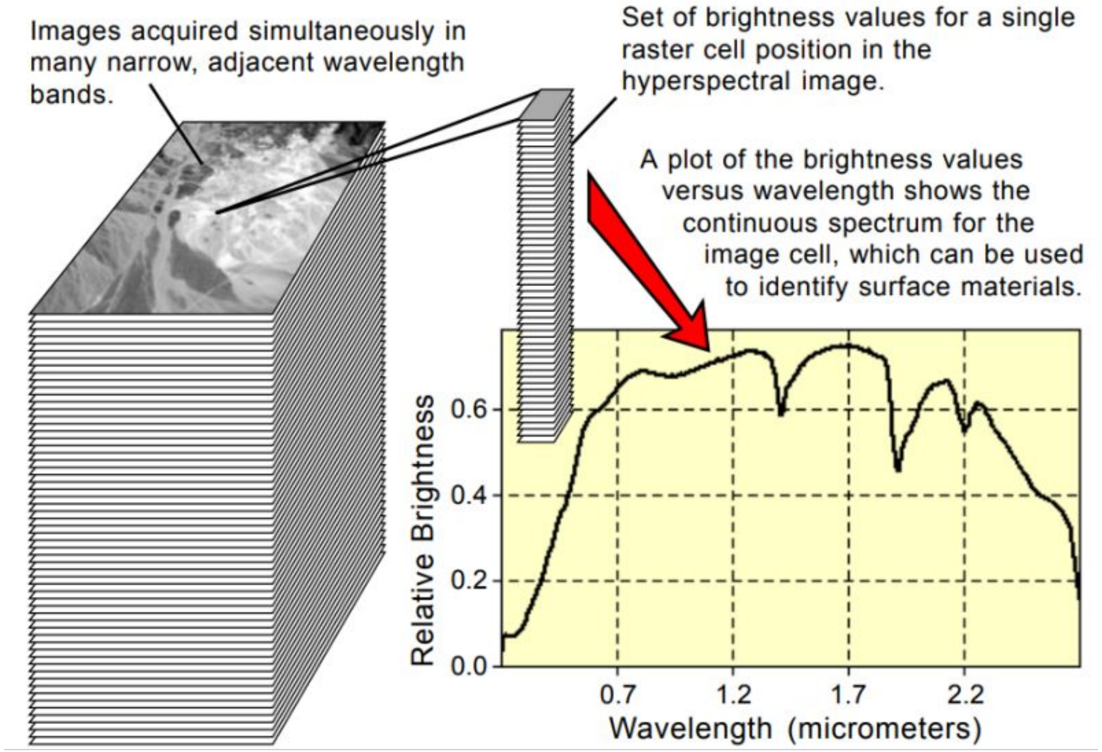
- Every material has a unique spectral signature characterized by it's reflectance graph





Hyperspectral imaging

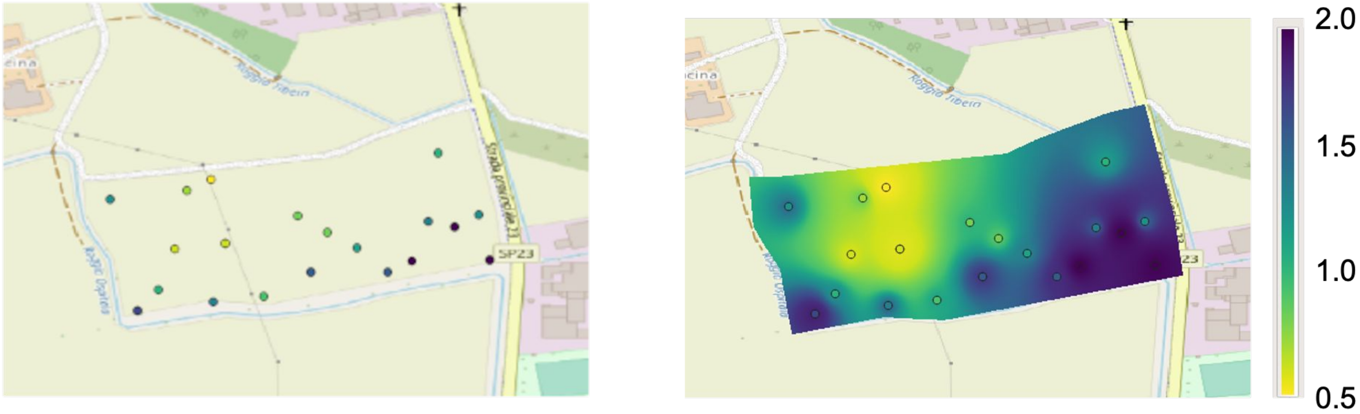
Many narrow wavelength are measured in close proximity to create a continuous graph of wavelength to reflectance



Spectrograph

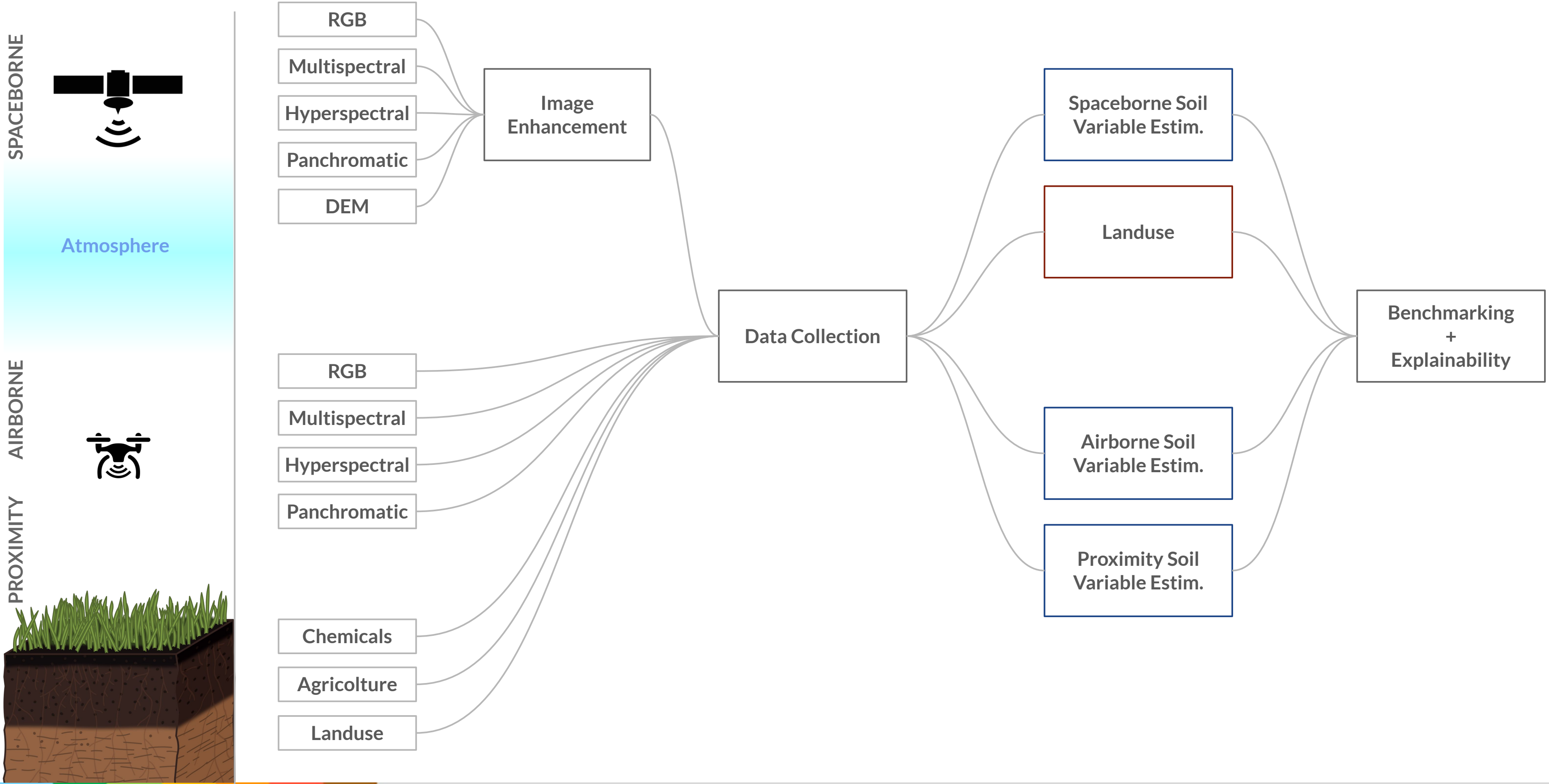
Digital soil mapping

- It is the computer-assisted production of digital maps of soil types and soil properties
- In the first place, several samples must be collected and analyzed in laboratory
  - it is very expensive
  - it is very time-consuming
- Then, it is possible to generate soil maps by interpolating on other locations



- Is it possible to create a deep-learning-based system to avoid manual analysis?
  - yes! Analyzing the hyperspectral signal
  - it is possible to mount the acquisition device on-board of drones and satellites

Setup


SPACEBORNE
Atmosphere
AIRBORNE
PROXIMITY
RGB
Multispectral
Hyperspectral
Panchromatic
DEM
Image Enhancement
RGB
Multispectral
Hyperspectral
Panchromatic
Chemicals
Agricolture
Landuse
Data Collection
Spaceborne Soil Variable Estim.
Landuse
Airborne Soil Variable Estim.
Proximity Soil Variable Estim.
Benchmarking + Explainability

Lucas soil dataset

It's a dataset for digital soil mapping through the analysis of the hyperspectral signal

Each sample is composed of:

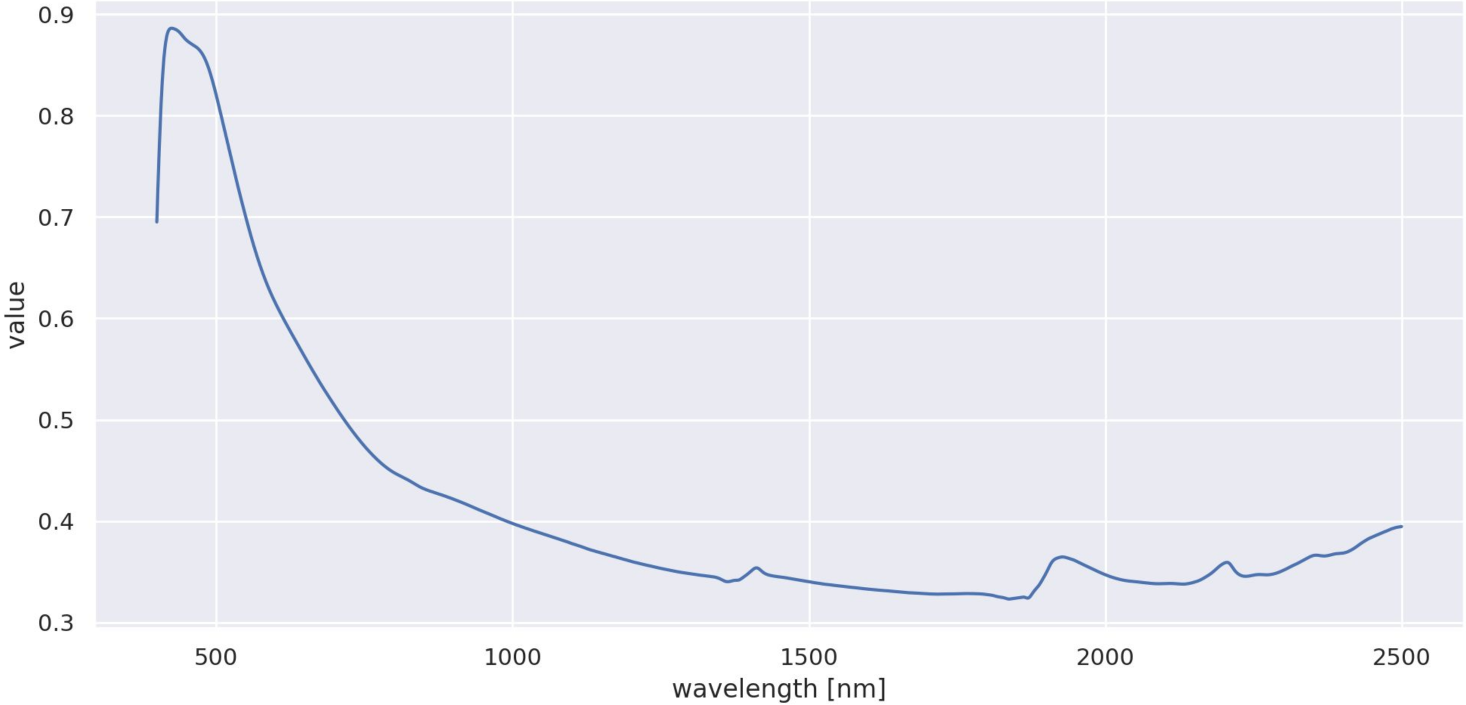
- GPS **position**
  - *GPS_LAT, GPS_LONG*

- **hyperspectral** signal 400 nm - 2499.5 nm
  - *spc.<WAVELENGTH>* columns

- 12 **soil variables**:
  - *coarse, clay, silt, sand, pH.in.CaCl2, pH.in.H2O, OC, CaCO3, N, P, K, CEC*

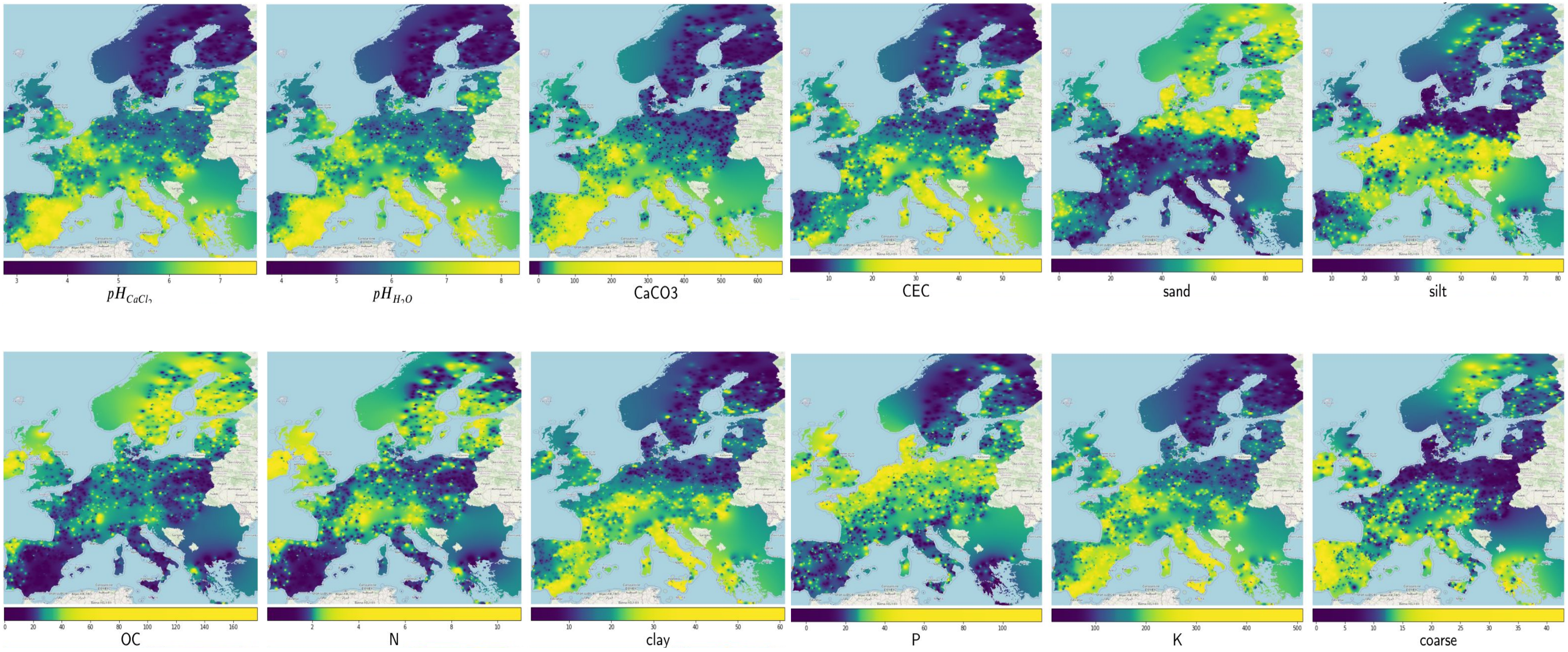
Lucas soil dataset

Example



Lucas soil dataset

Interpolation of the ground truths



Exercise

Train a convolutional neural network capable of estimating soil properties from hyperspectral signal

- On e-learning you will find:
  - the dataset
  - few lines of code needed to load the data in a proper way