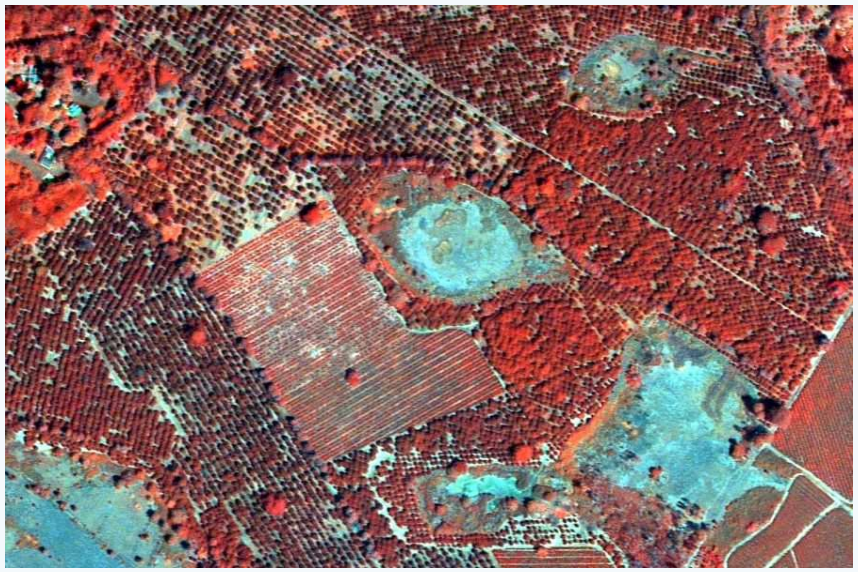
Deep Learning based Hyperspectral Image Processing

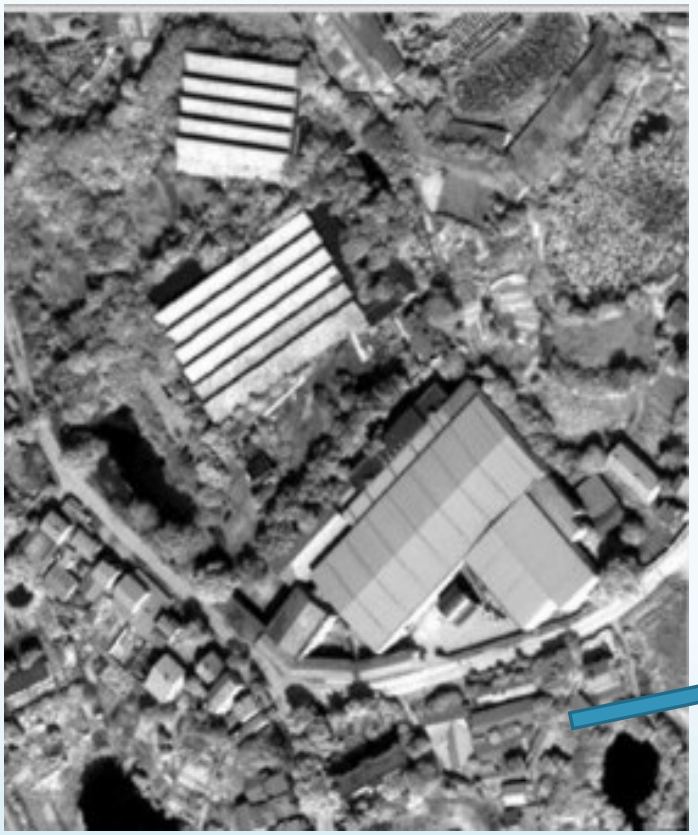
(January 2021)

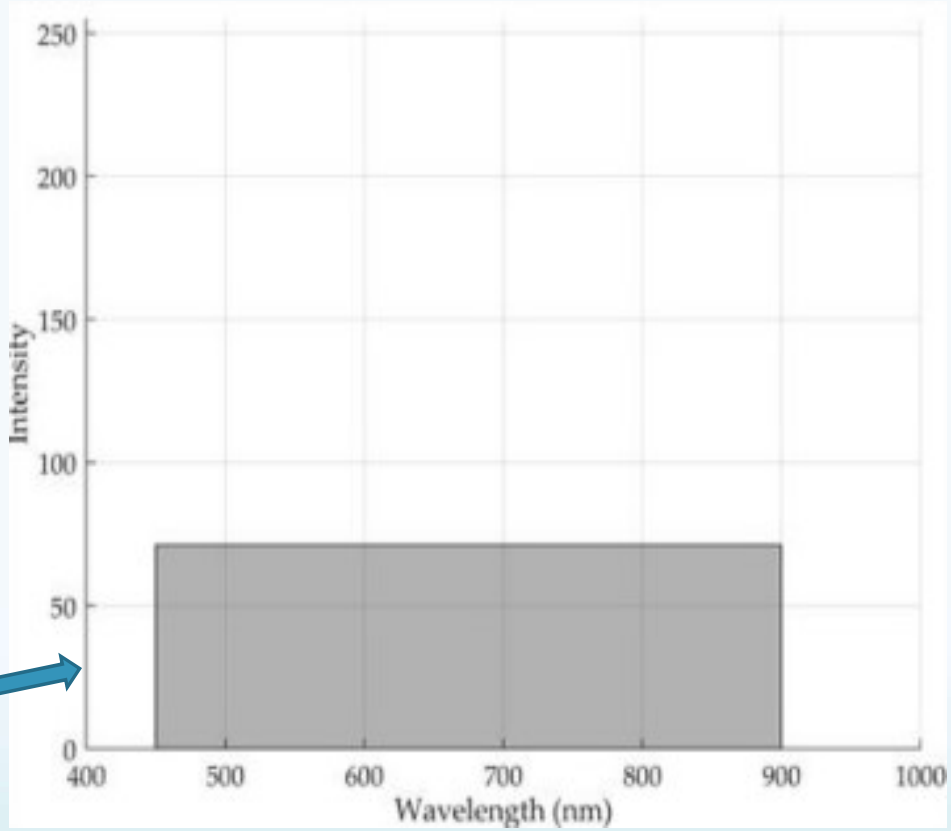
Contents

Satellite Image



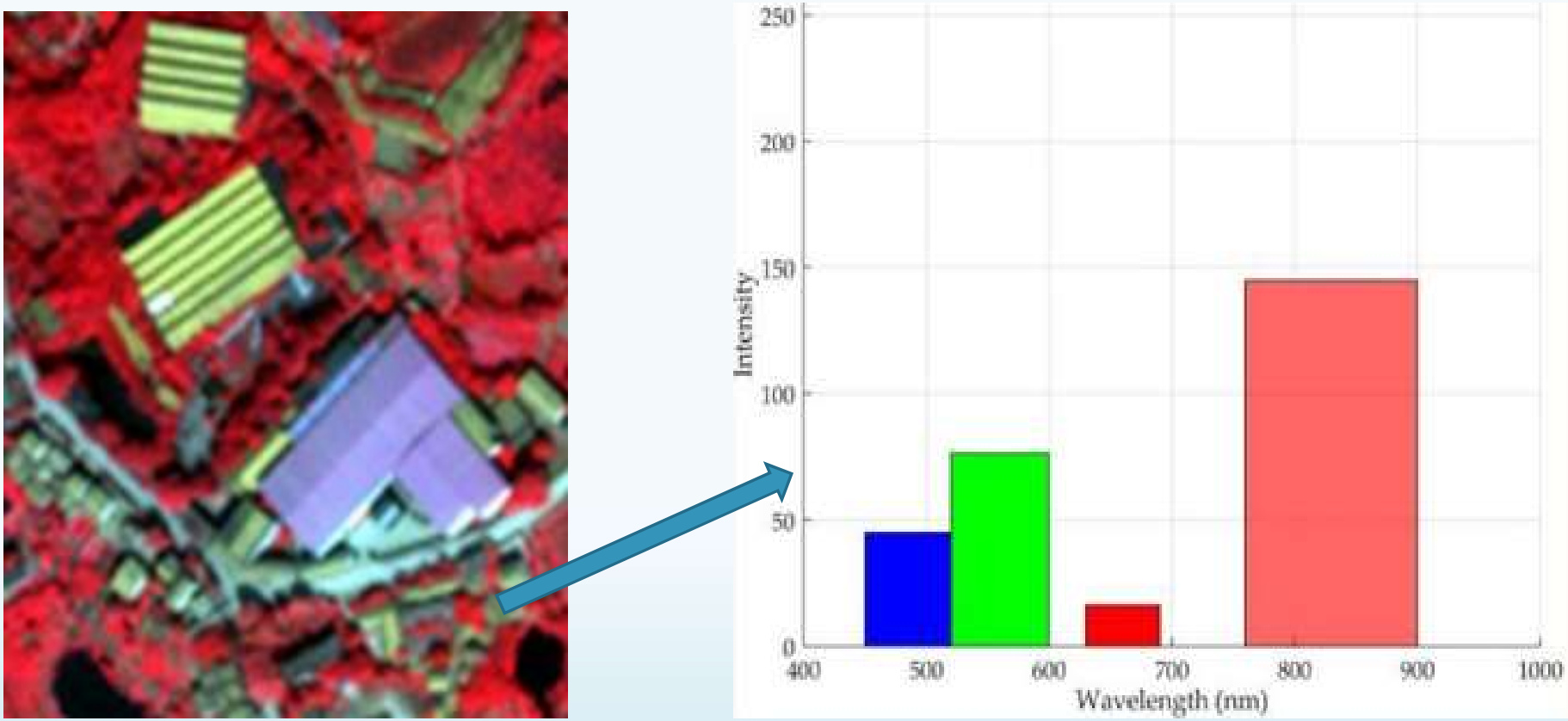
Panchromatic Image





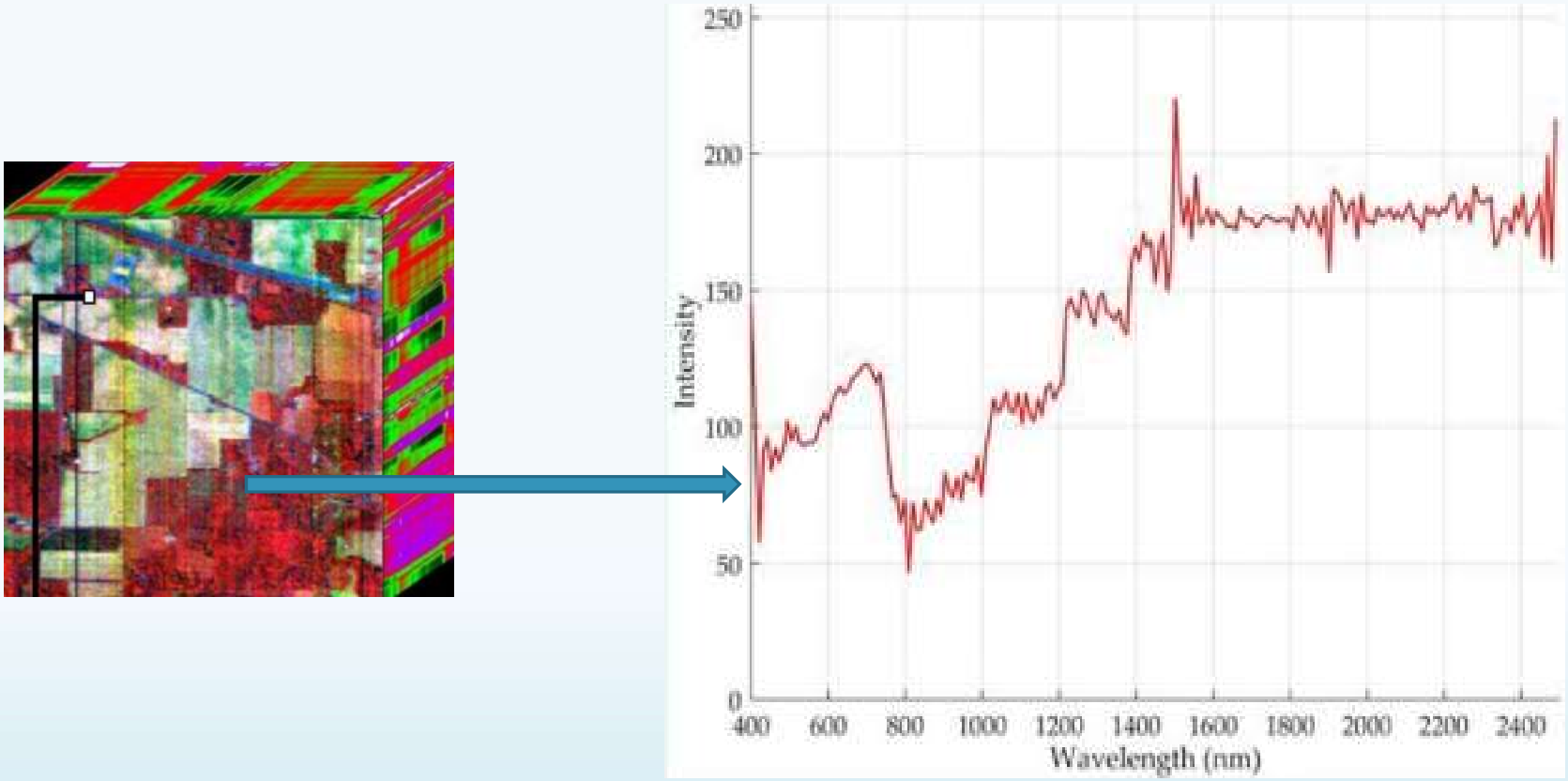
Source: https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0027_DAI6/ch01s02.html

Multispectral Image



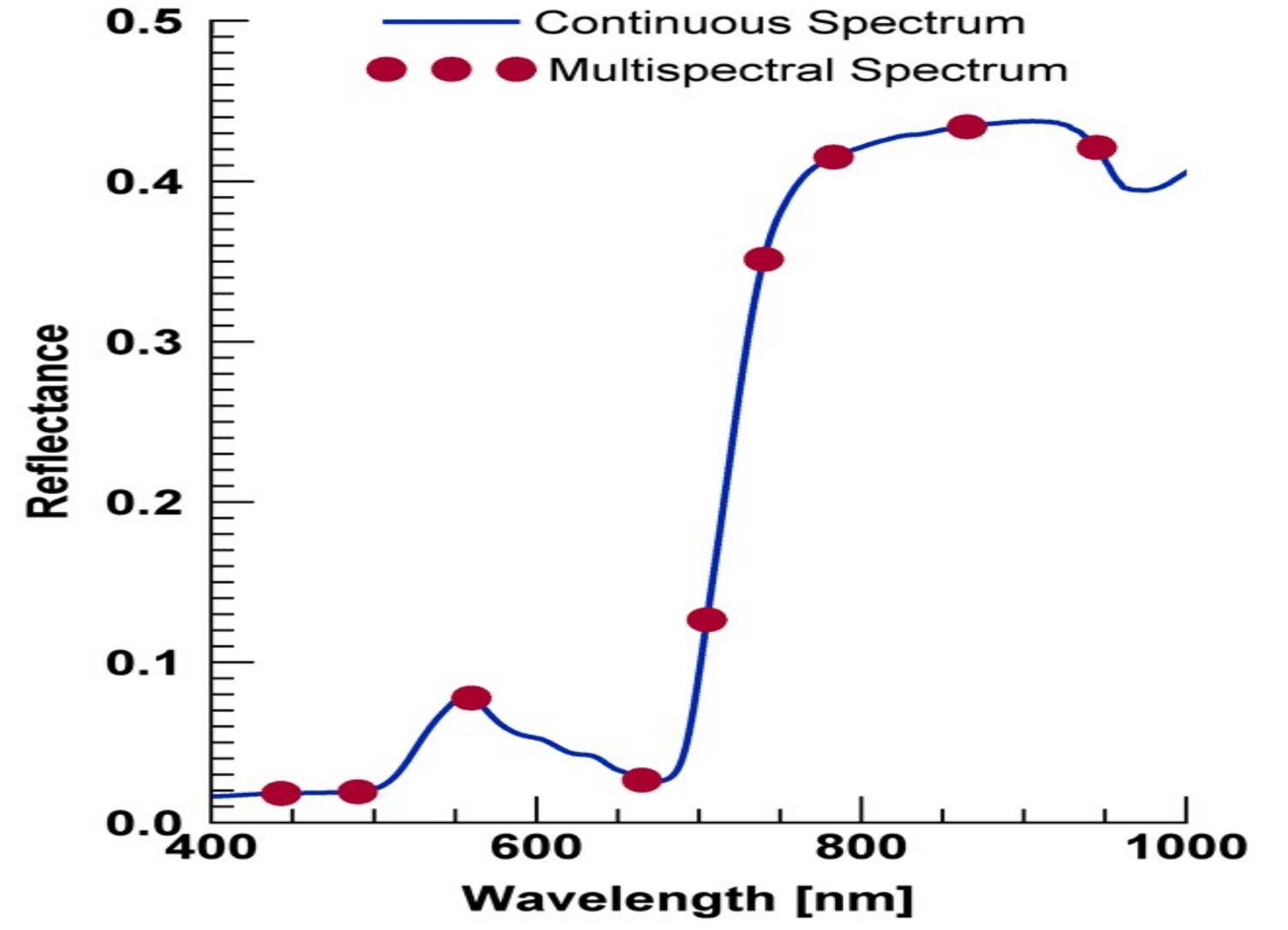
Source: https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0027_DAI6/ch01s02.html

Hyperspectral Image

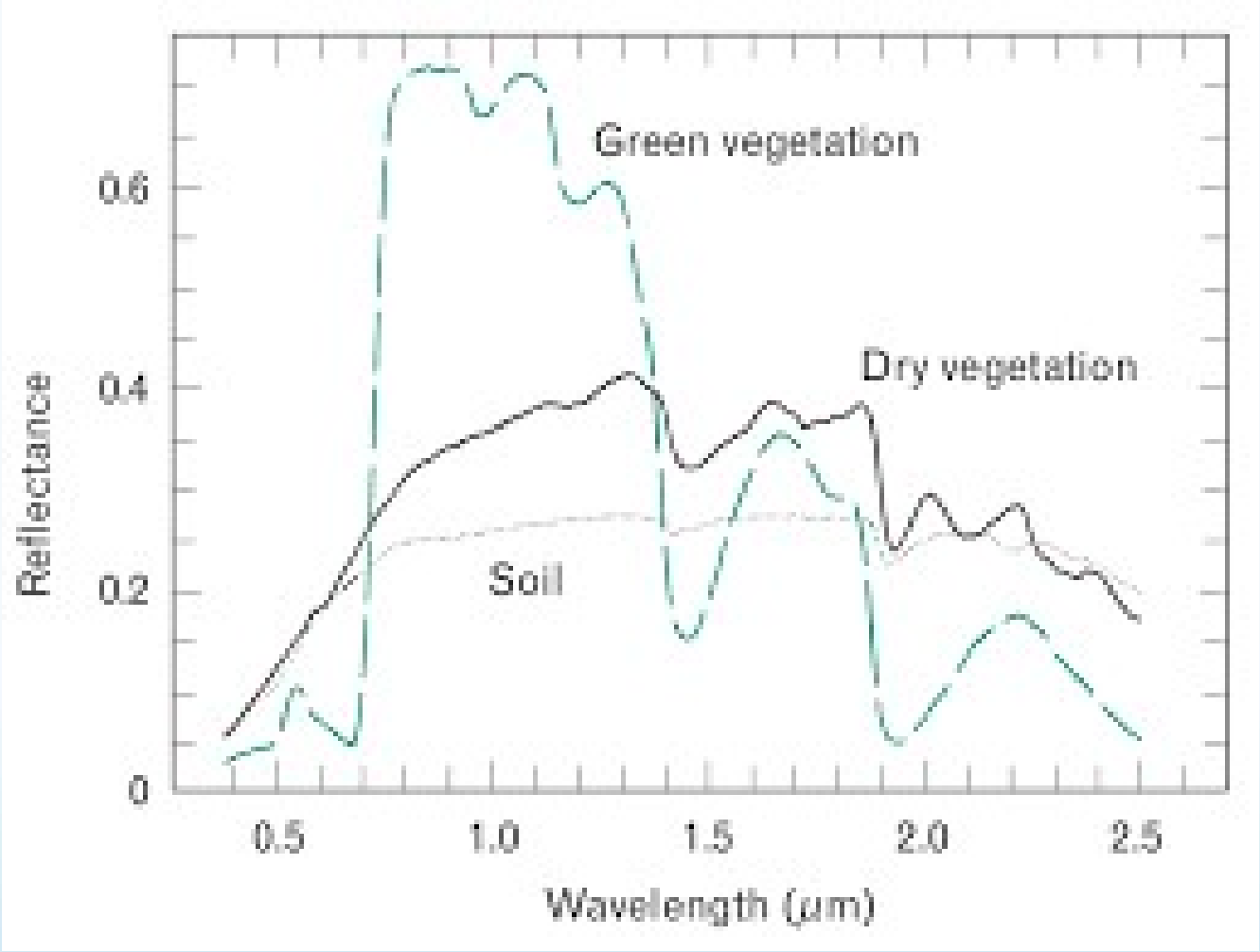


Source: https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0027_DAI6/ch01s02.html

Multispectral vs Hyperspectral Pixel



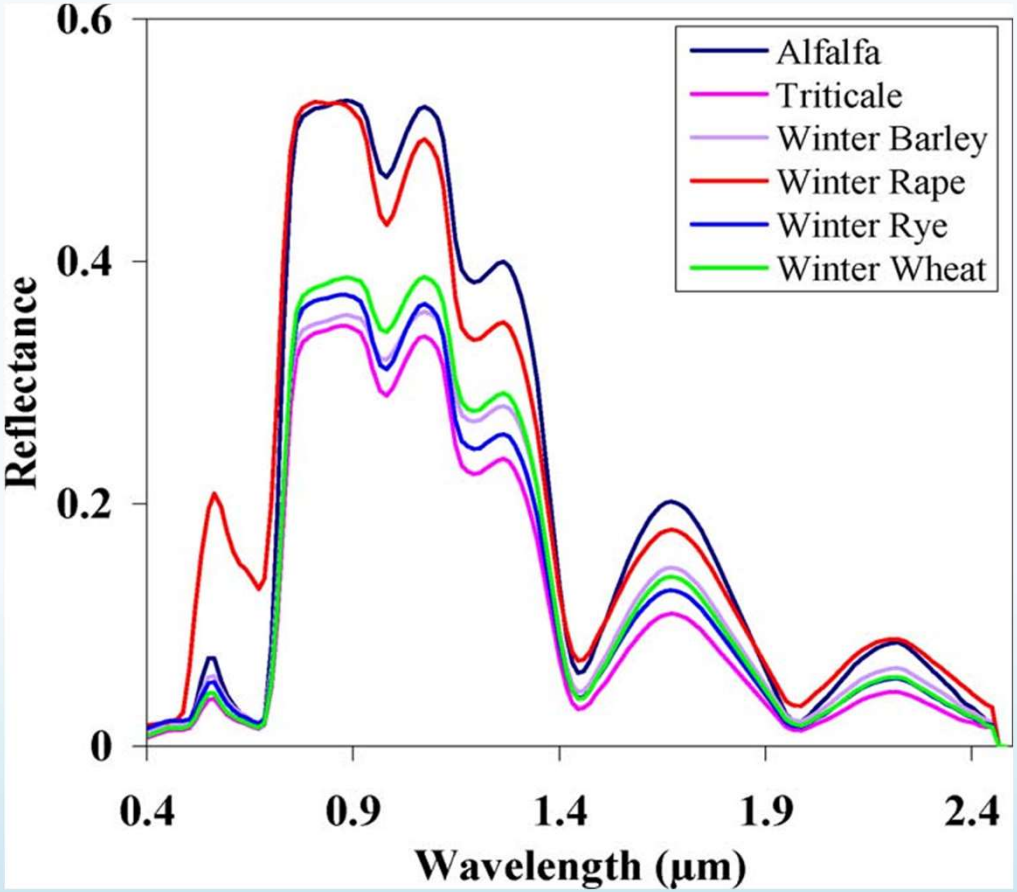
Hyperspectral Pixel



Source: https://www.mdpi.com/2072-4292/9/11/1110/htm

Distinguishing Vegetation Type

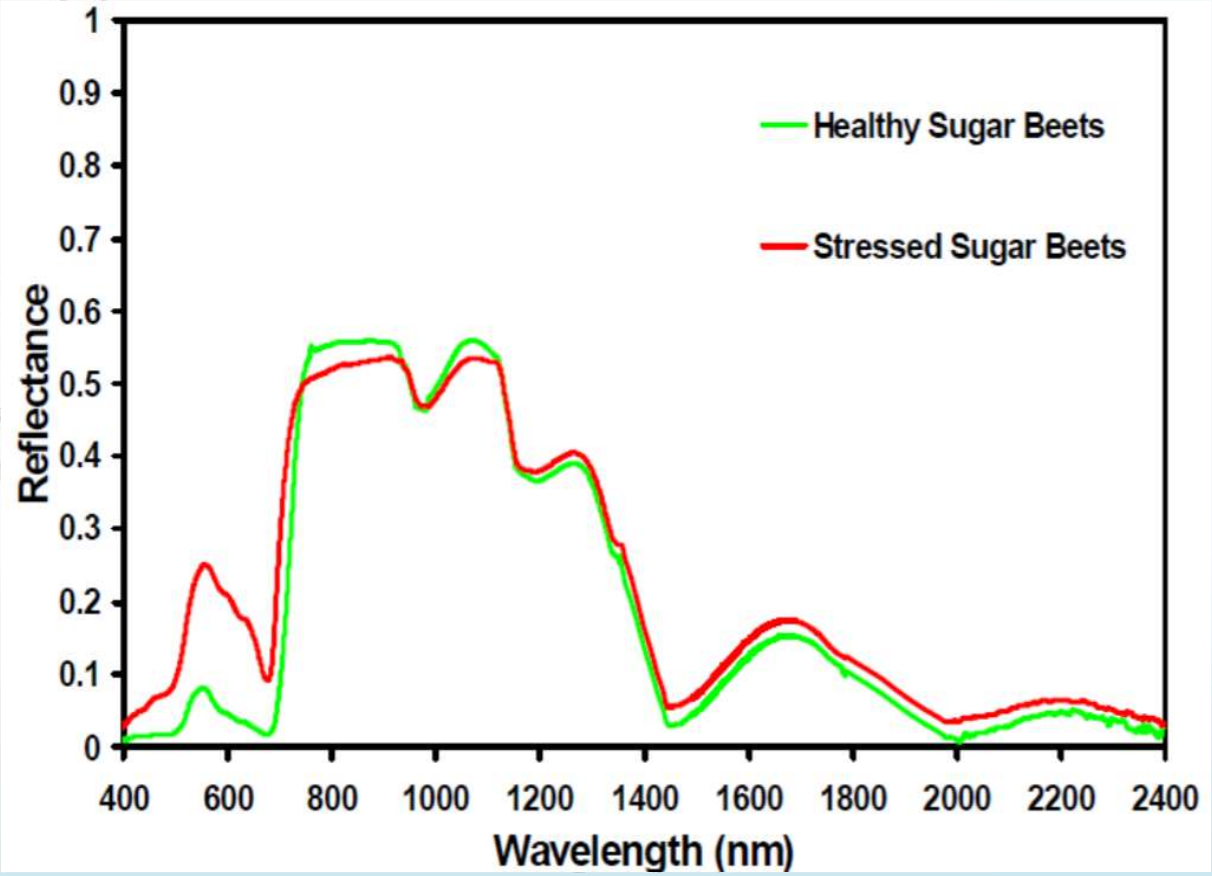
Pixel spectra or each pixel (pixel vector) is checked to detect the vegetation type



Source: https://www.intechopen.com/books/biomass-and-remote-sensing-of-biomass/

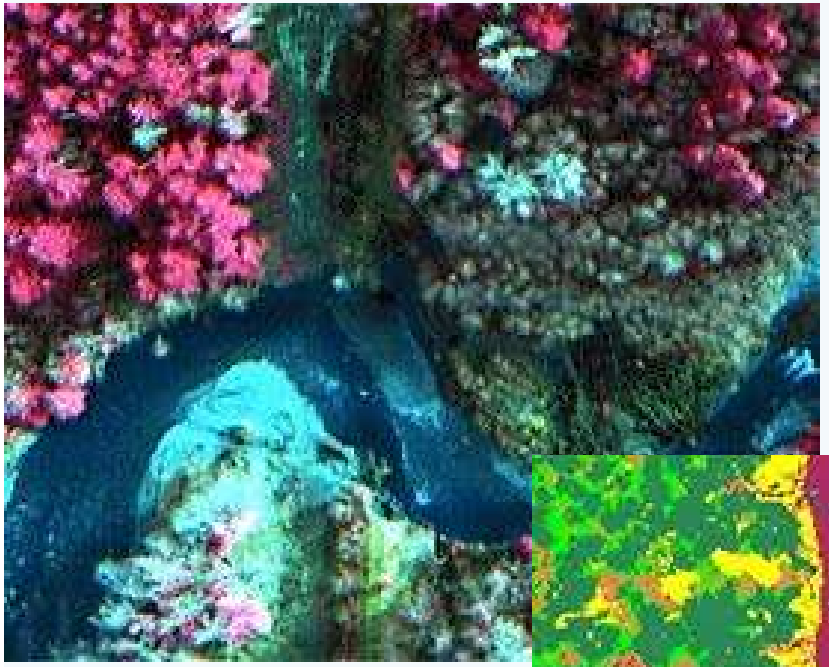
Distinguishing Vegetation Stress

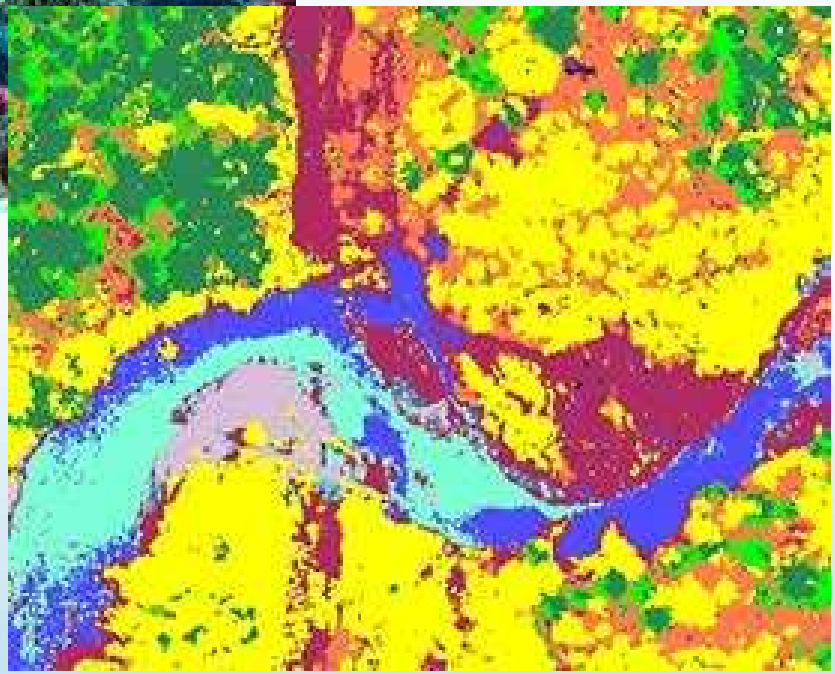
Pixel spectra or each pixel (pixel vector) is checked to detect the vegetation stress



Source: https://www.intechopen.com/books/biomass-and-remote-sensing-of-biomass/

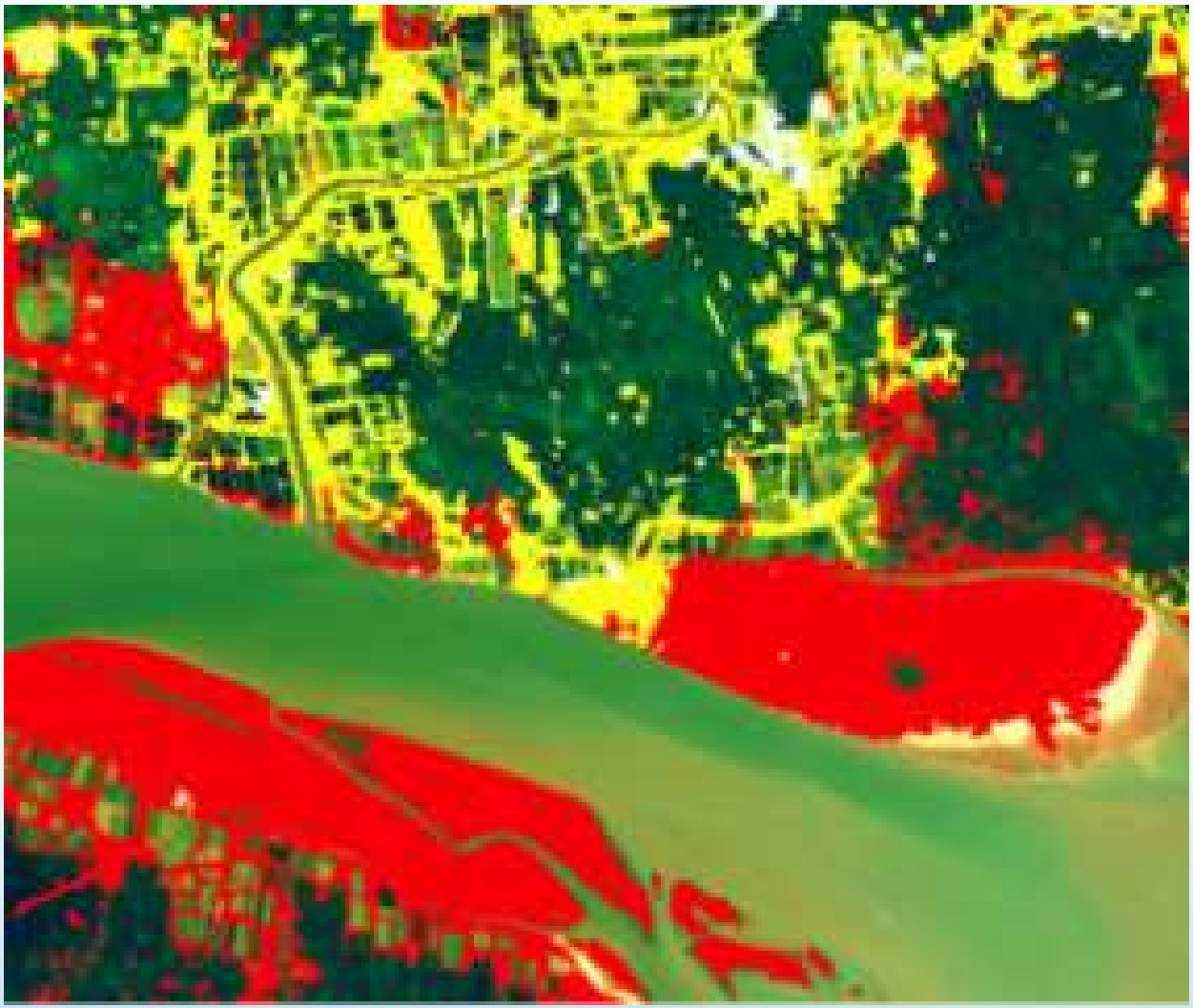
Vegetation Identification





Conifers
Lower branches
Gravel
Deciduous
Dry ground
Wet ground
Water
Deep water

Vegetation Health Monitoring



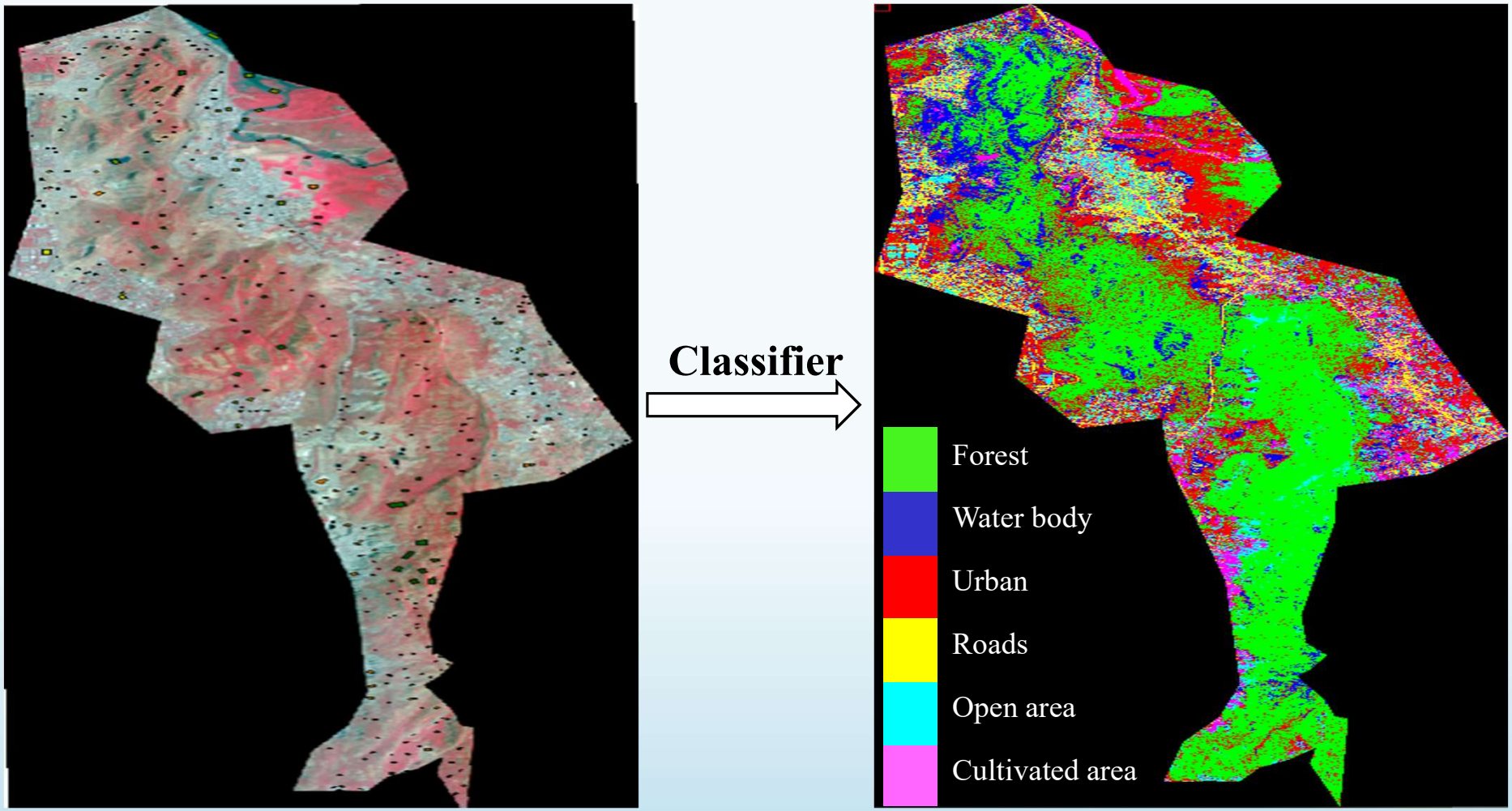
Healthy crops

Fine crops

Crops under stress

Less vegetation

ML based Classification Algorithm



Analysis Spaces

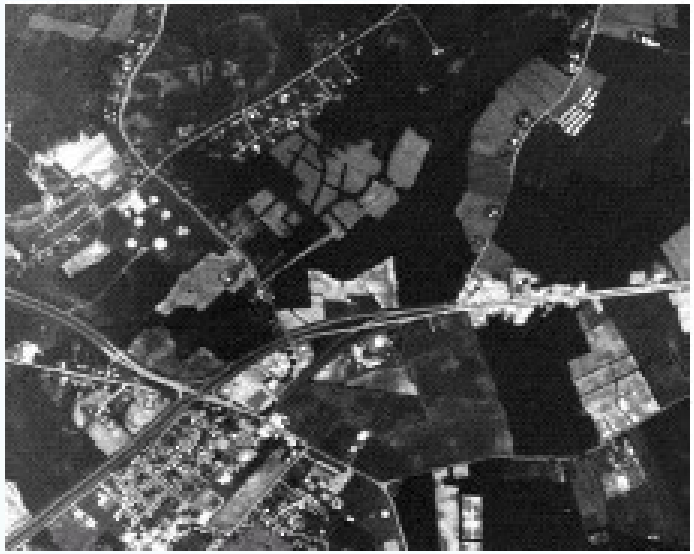
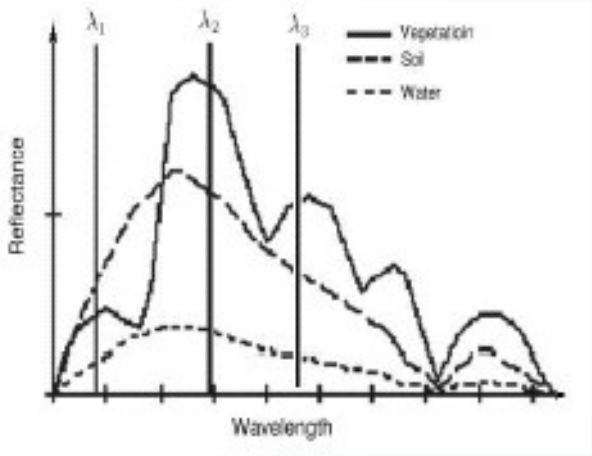
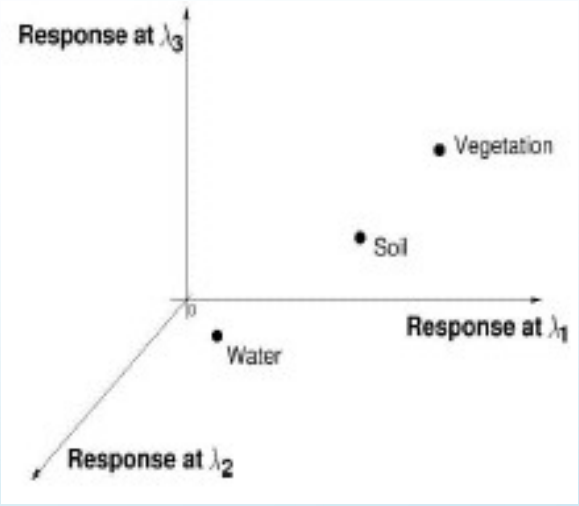


Image space



Spectral space



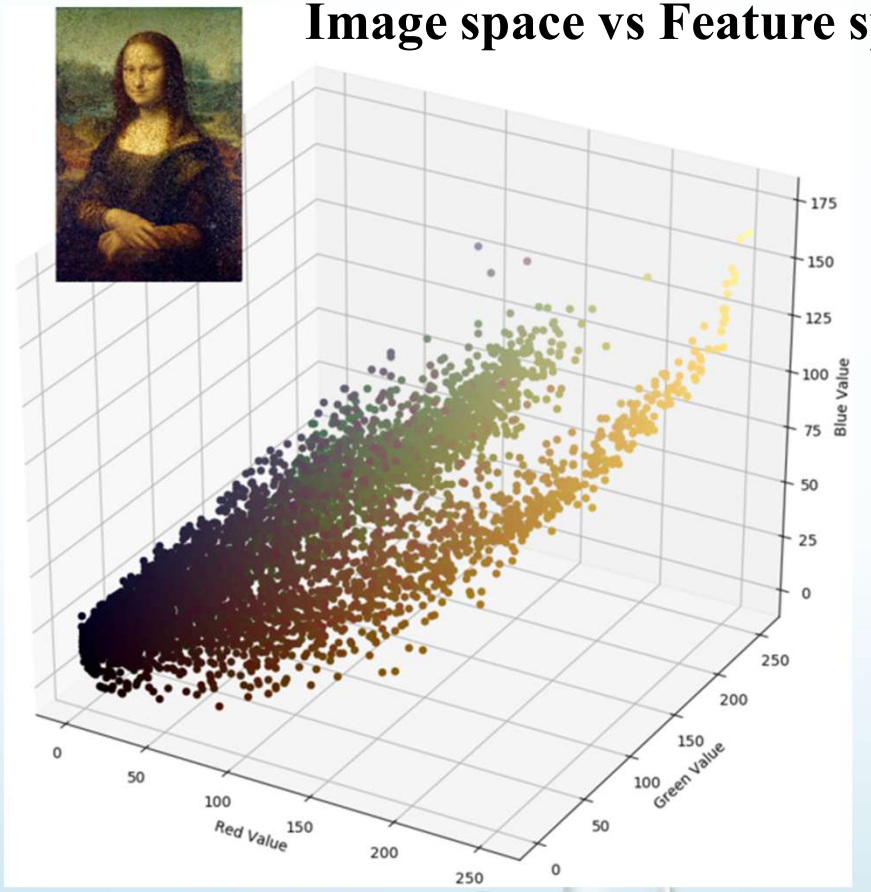
Feature Space

Analysis Spaces

- Image space
  - Pixels are displayed in grey scale.
  - Spatial analysis
- Spectral space
  - Pixels are functions of wavelength.
  - Spectral analysis
- Feature space
  - Pixels are points in n-dimensional space.
  - Relationships among pixels

Feature space

ML algorithms generally work in feature space



Source: https://www.pinterest.com/pin/255086766377701021/

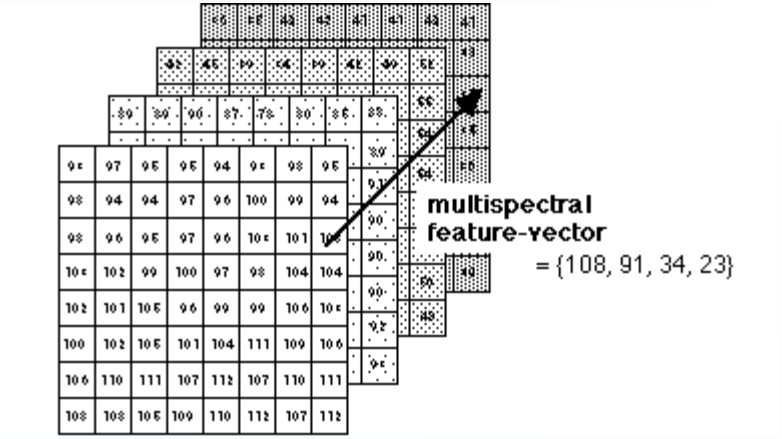
Full Pixel Classification

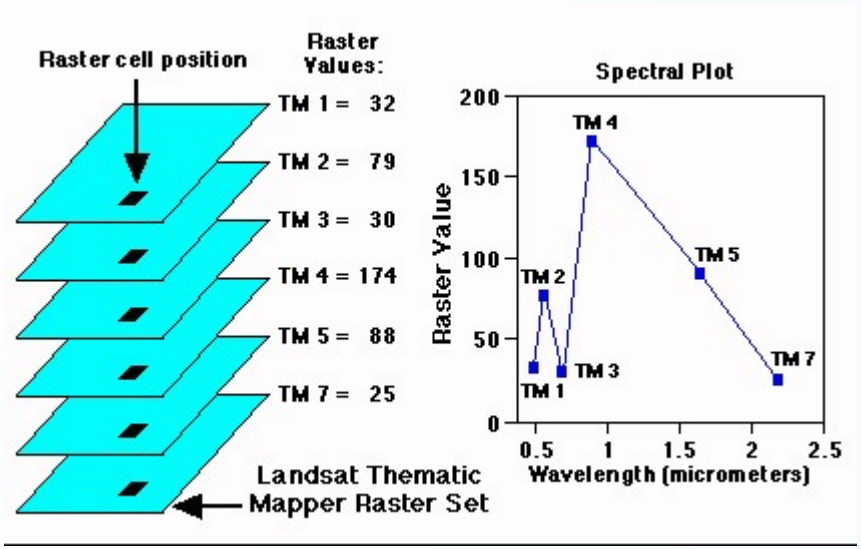
The process of reducing images to information classes. Classification divides the spectral or spatial **feature space** into several classes based on a decision rule.

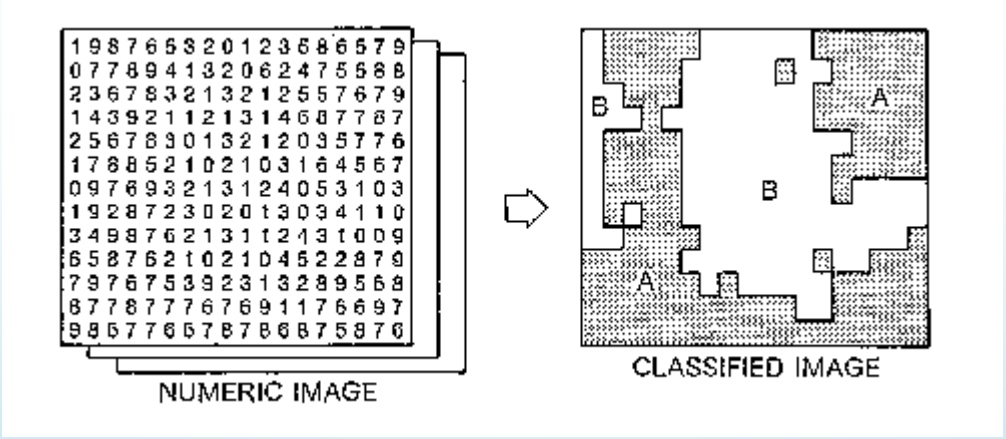
General procedures

- **Feature extraction** : transform the multispectral image by a spatial or spectral transform to a feature image (optional). Ex) selection of bands, filtering, PCA.
- **Training** : extract the pixels to be used for training the classifier to recognize certain categories, or classes. Determine the *discriminant functions* in the feature space. *Supervised* or *unsupervised*
- **Labeling** : apply the discriminant functions to the entire feature image and label all pixels. The output consists of one label for each pixel.

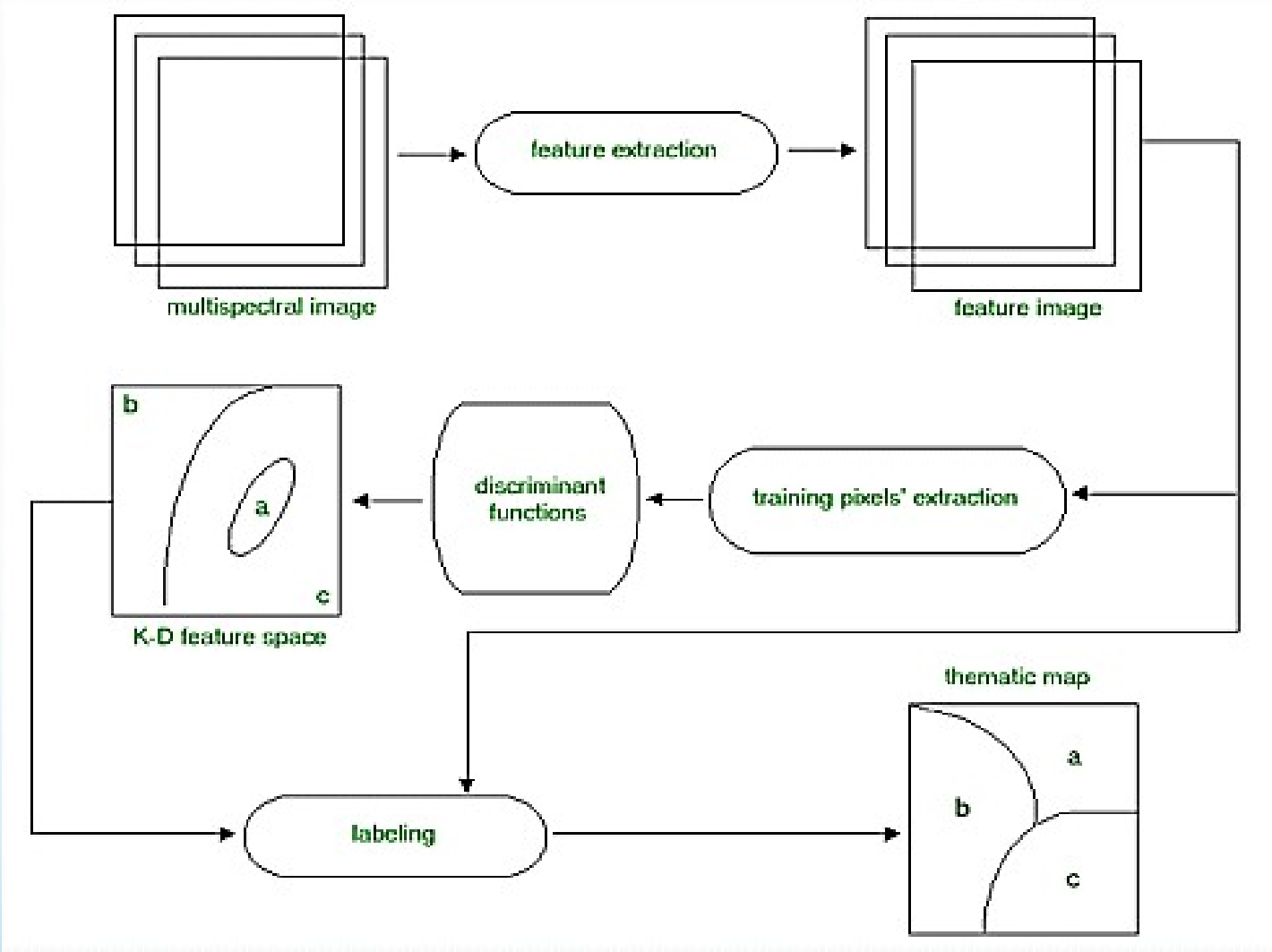
Classification – feature space







Classification – feature space

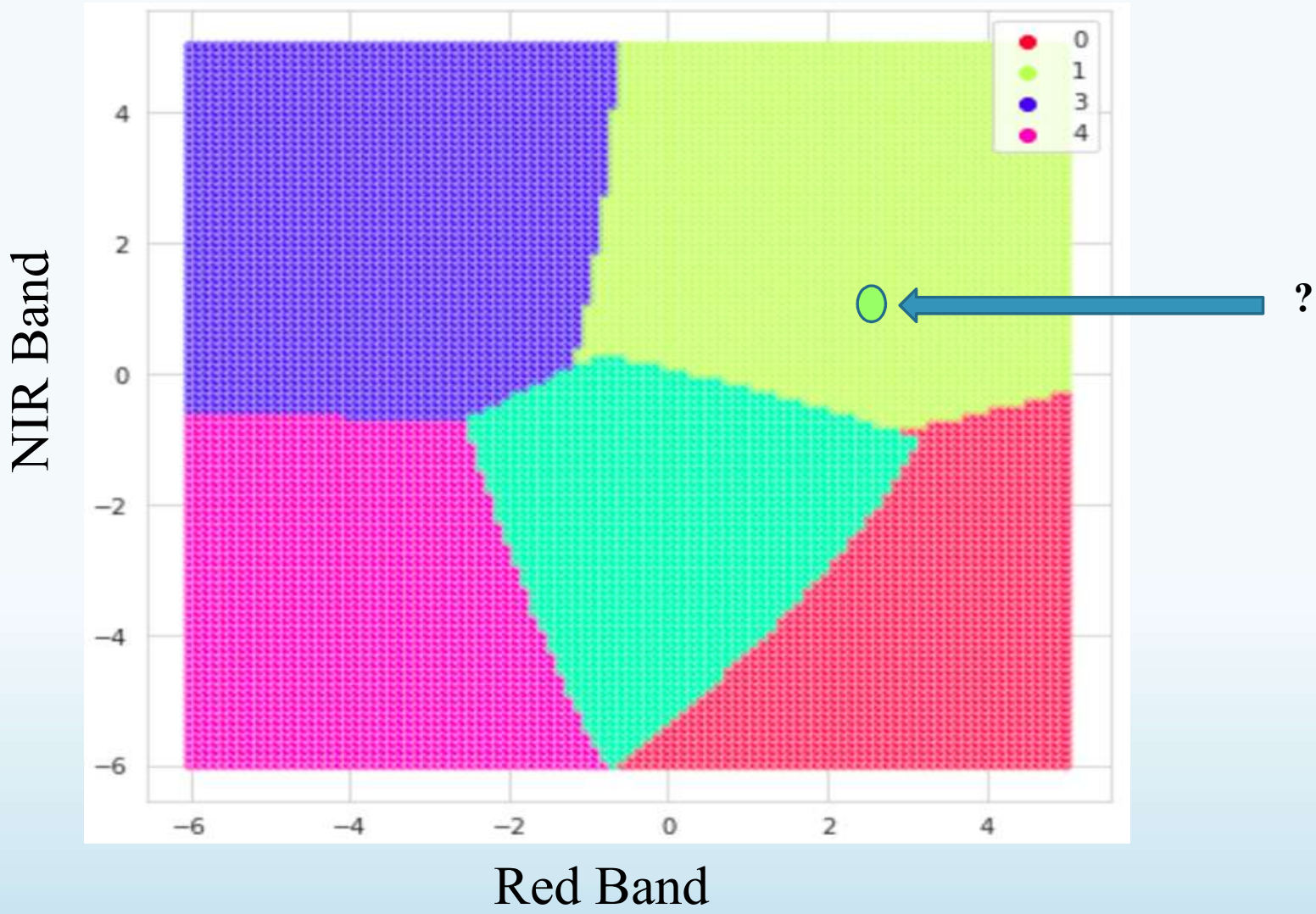


Classification – feature space

By the use of **labeling method** (classifier):

- Non-parametric: do not use statistics
  - Level-slice classifier
  - Parallelepiped classifier
  - Histogram estimation classifier
  - Nearest neighbors classifier
  - Artificial neural network classifier
- Parametric: use mean, covariance
  - Nearest mean classifier
    (Minimum distance classifier)
  - Maximum likelihood classifier

Classification in feature space



Source: https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning

Issues with Hyperspectral Images

Huge Dimension: High-dimensional nature of hyperspectral data introduces important limitations in supervised classifiers, such as the limited availability of training samples or the inherently complex structure of the data

Coarse spatial resolution: There is a need to address the presence of mixed pixels resulting from insufficient spatial resolution and other phenomena in order to properly model the hyperspectral data

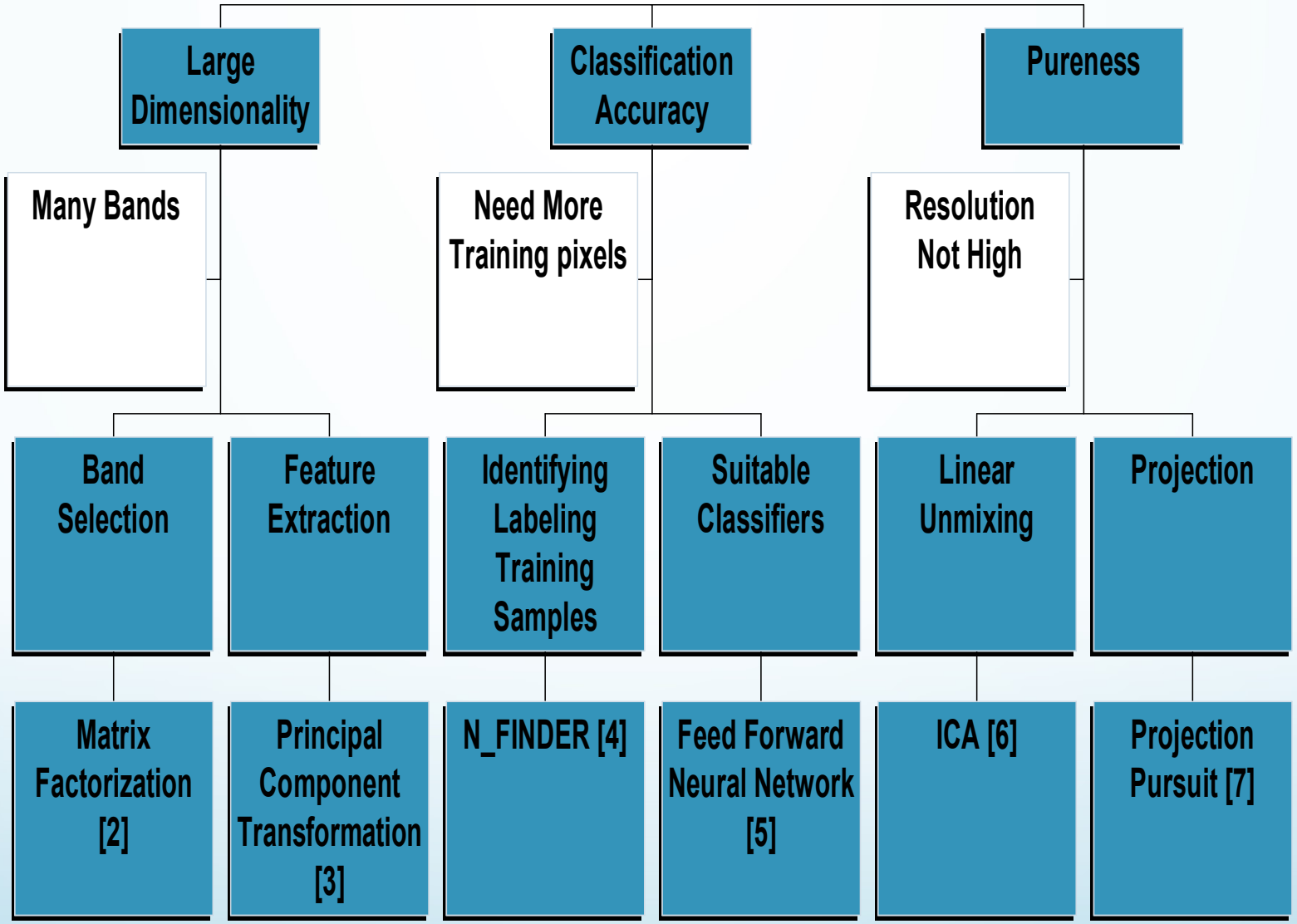
Issues with Hyperspectral Images

Presence of redundant bands and bad bands: Need to remove redundant and corrupted bands

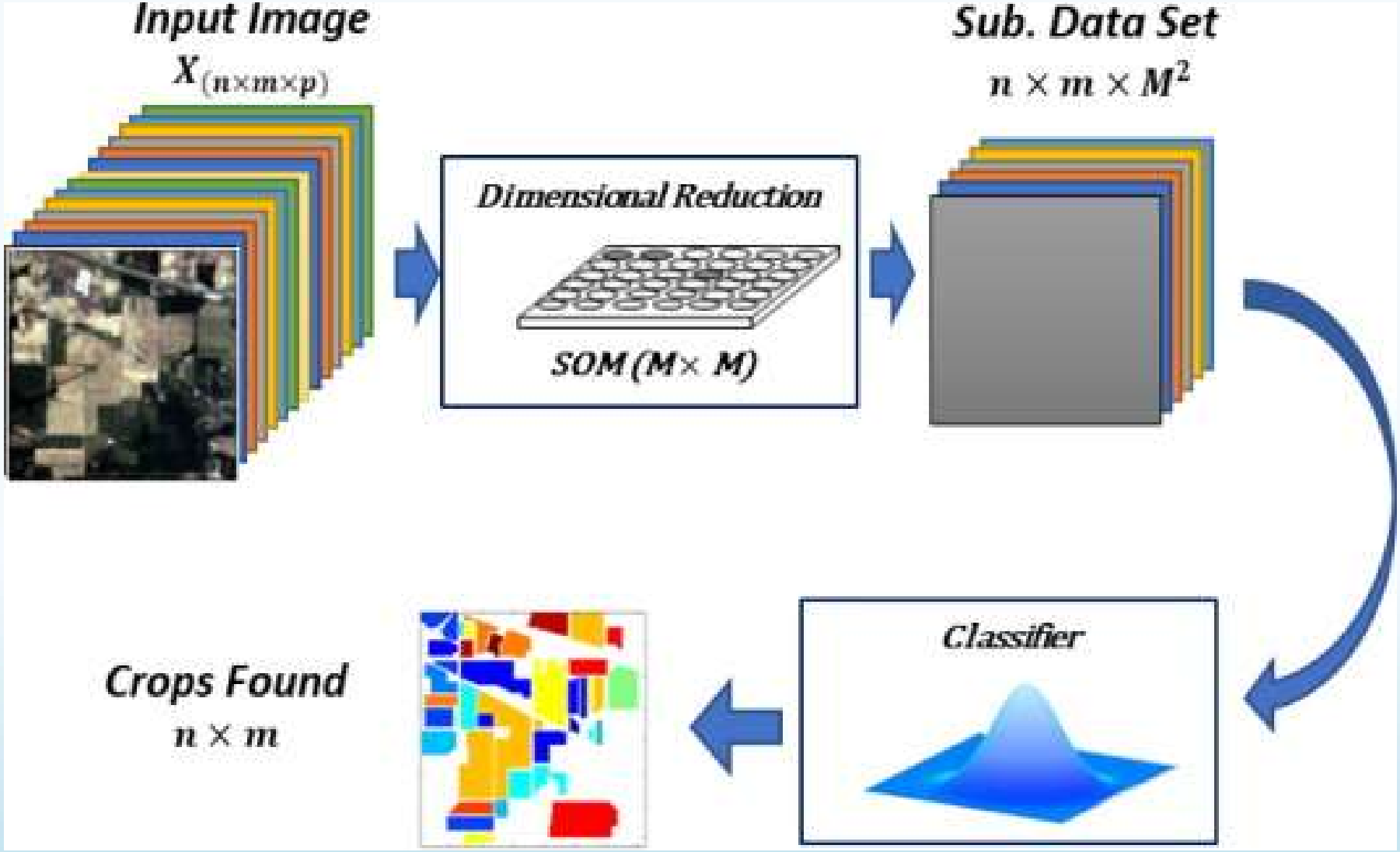
Atmospheric noise effect: The energy over the narrow spectra are very sensitive to the atmospheric noise

Computationally intensive: There is a need to develop computationally efficient algorithms, able to provide a response in a reasonable time and thus address the computational requirements of time-critical remote sensing applications

Challenges and Approaches

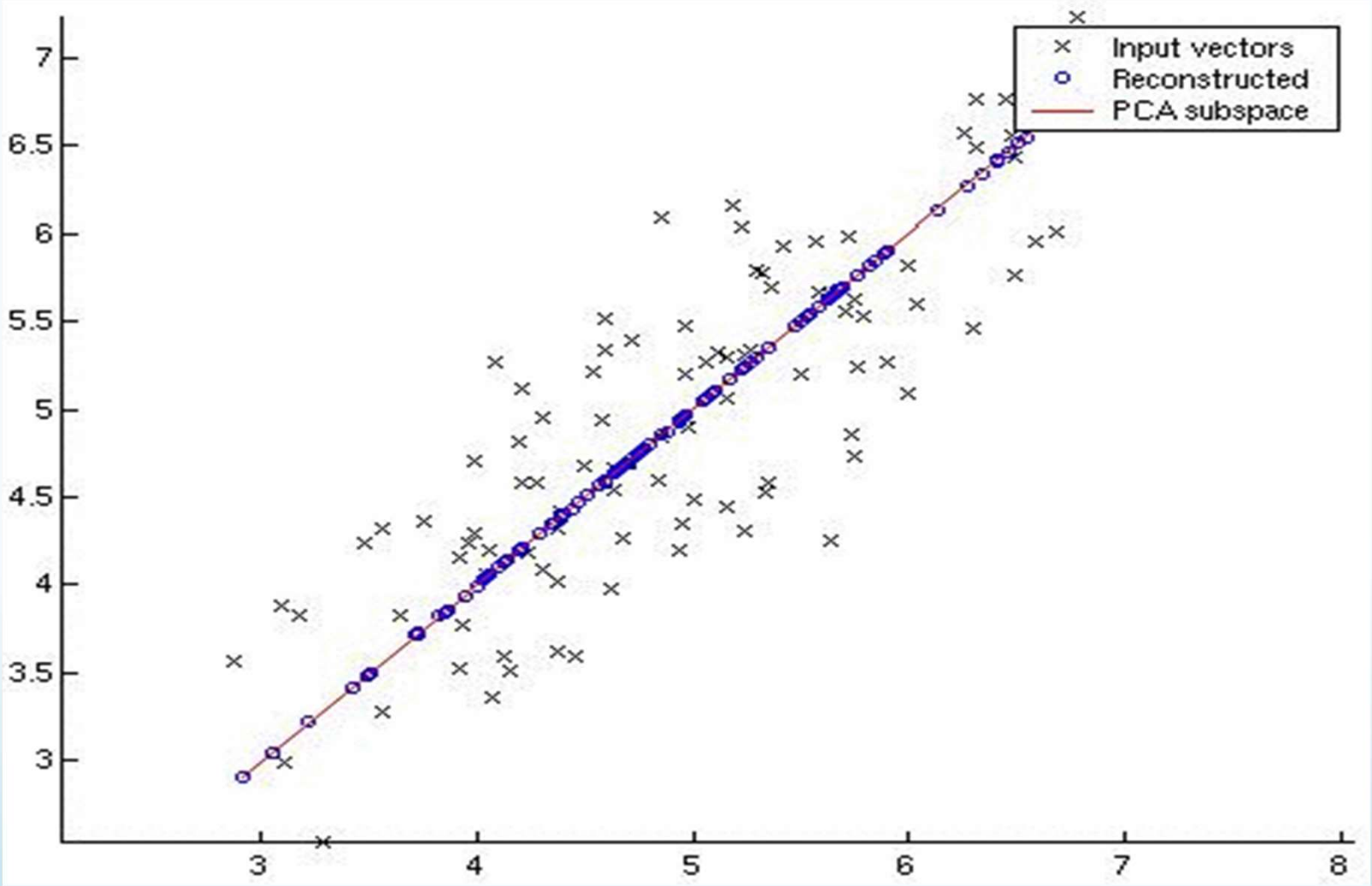


Dimensionality Reduction



Source: https://www.sciencedirect.com/science/article/pii/S221431732030189X

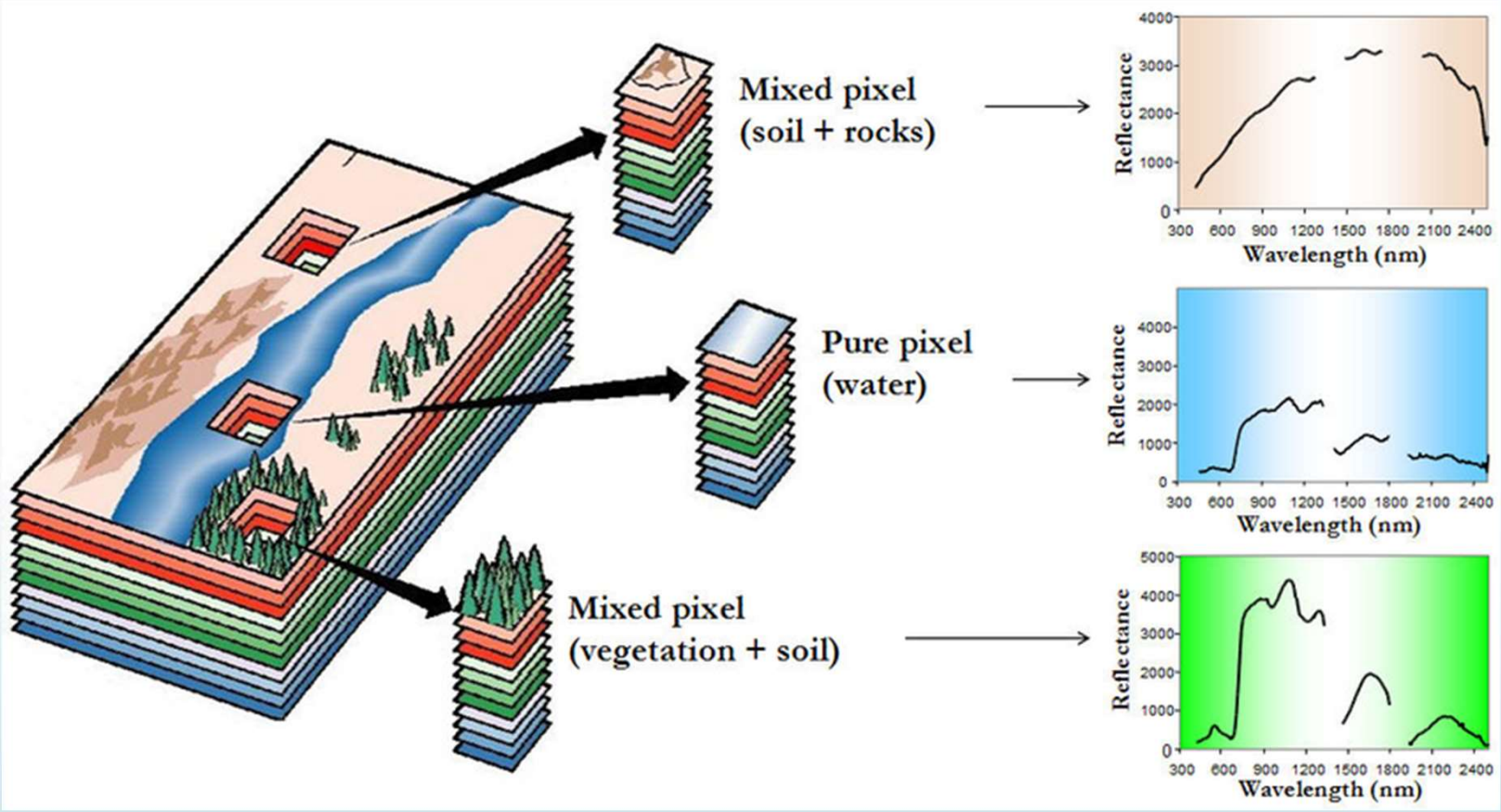
Dimensionality Reduction



Source: https://math.stackexchange.com/questions/1146/intuitive-way-to-understand-principal-component-analysis

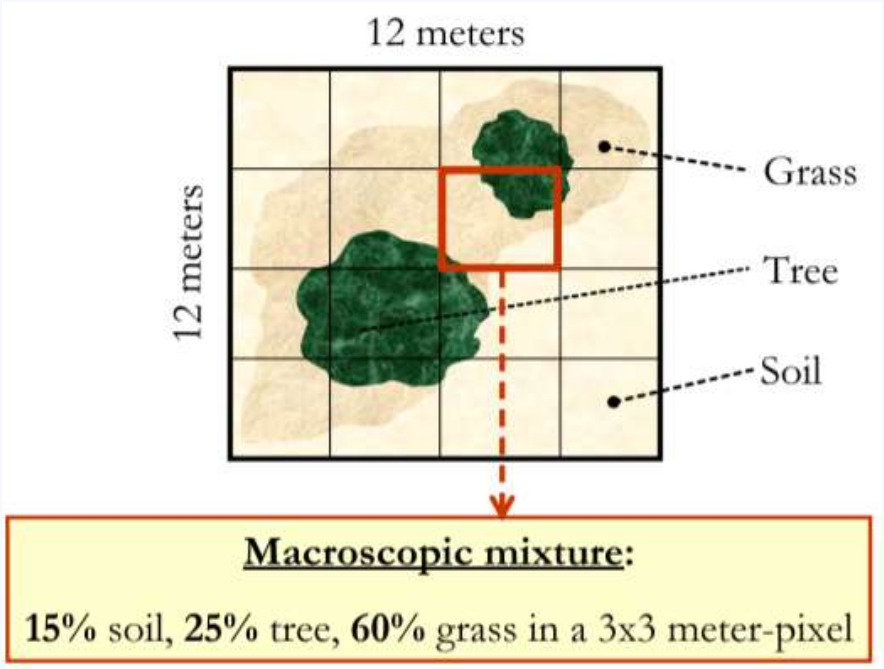
Spectral Mixing

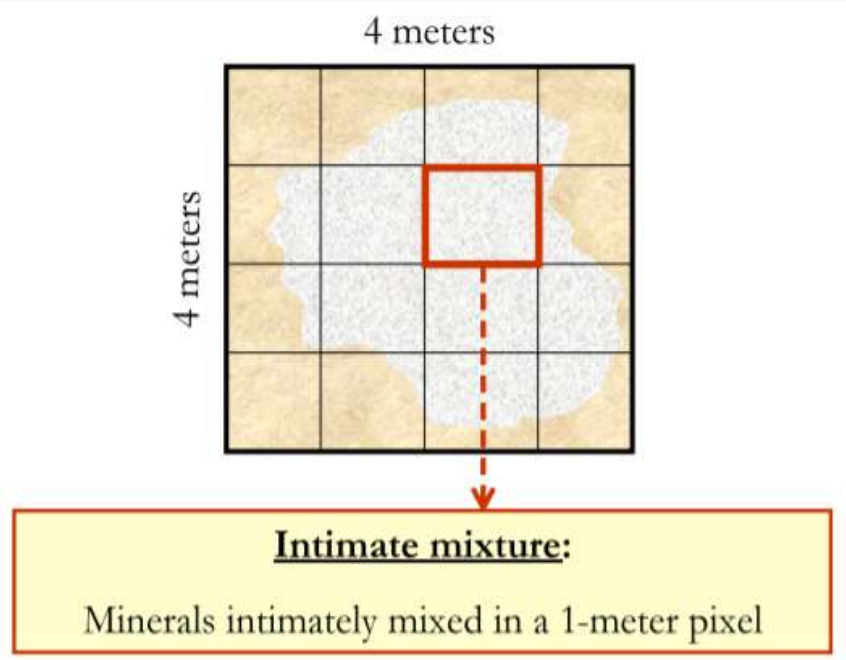
Coarse spatial resolution causes the disparate material substances in the IFOV to contribute to the spectrum measured at a single pixel (mixed pixels).



Source: https://www.researchgate.net/figure/The-mixture-problem-in-remotely-sensed-hyperspectral-data-analysis_fig1_226364968

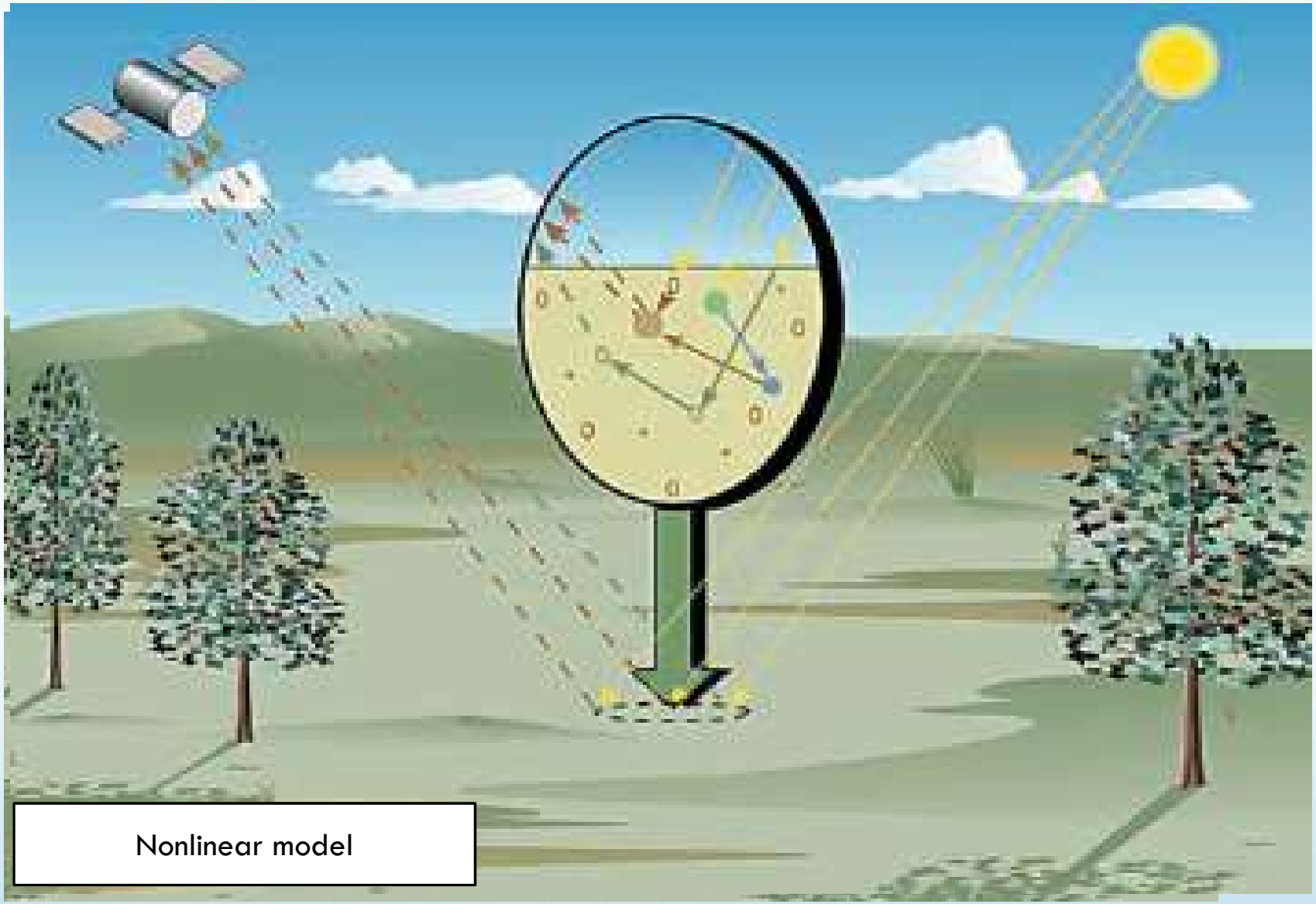
Spectral Mixing





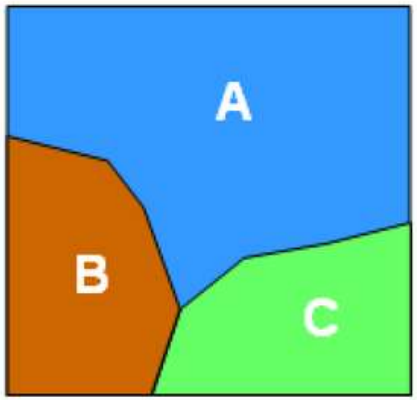
Source: https://www.sfpt.fr/hyperspectral/wp-content/uploads/2013/01/cours_Licciardi.pdf

Spectral Mixing



Source: https://www.researchgate.net/figure/The-mixture-problem-in-remotely-sensed-hyperspectral-data-analysis_fig1_226364968

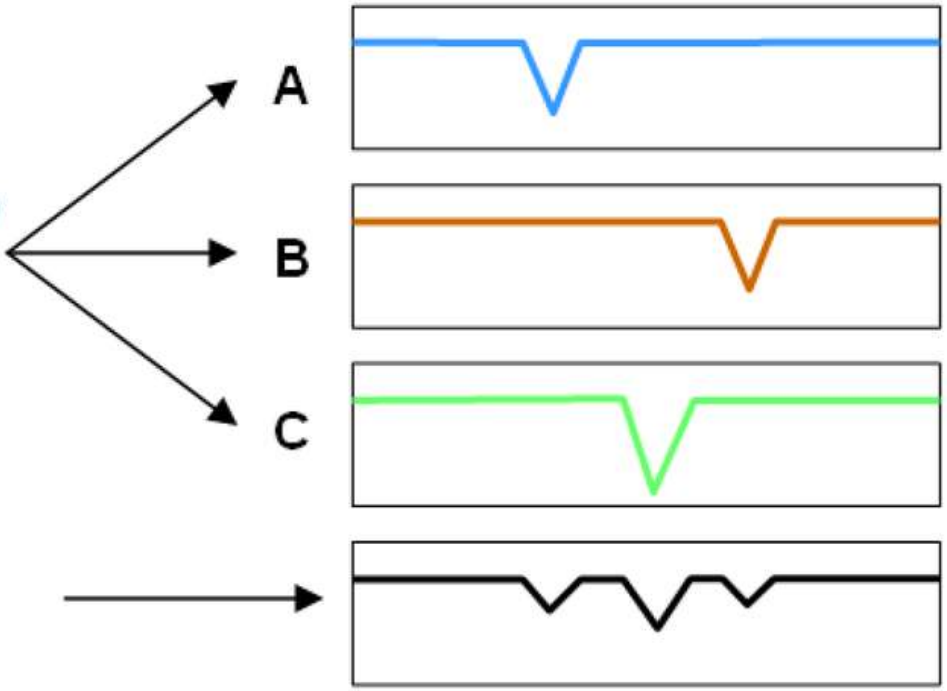
Linear Spectral Mixing



IFOV of pixel

A single pixel with three materials A, B, and C	
Material	Fraction
A	0.50
B	0.25
C	0.25

Each endmember has a unique spectrum



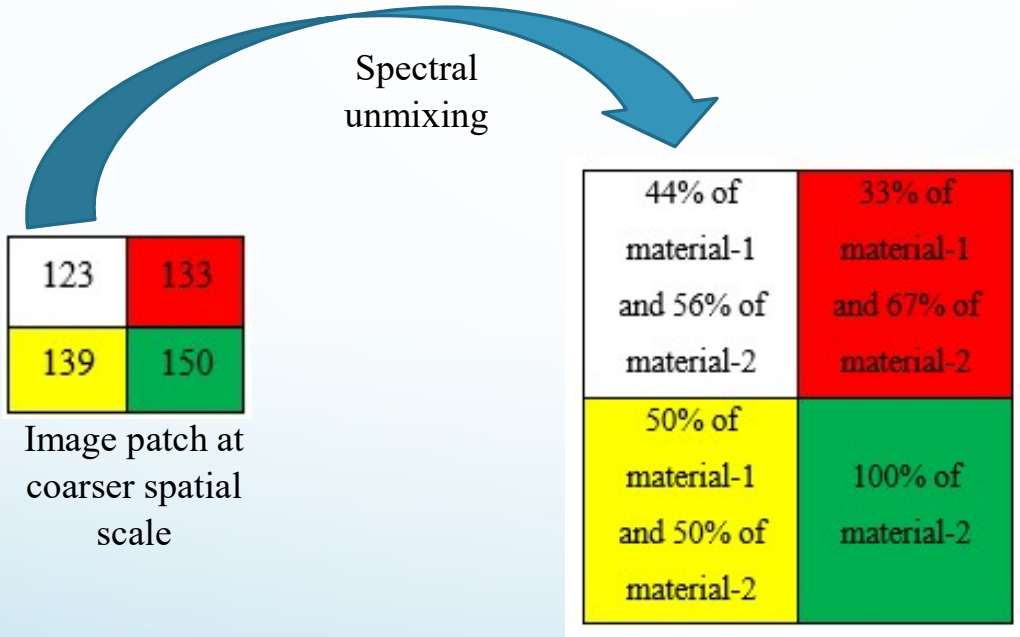
The mixed spectrum is just a weighted average

Mix=0.50*A+0.25*B+0.25*C

Source: https://www.mdpi.com/journal/remotesensing/special_issues/hyperspectral-remote-sens

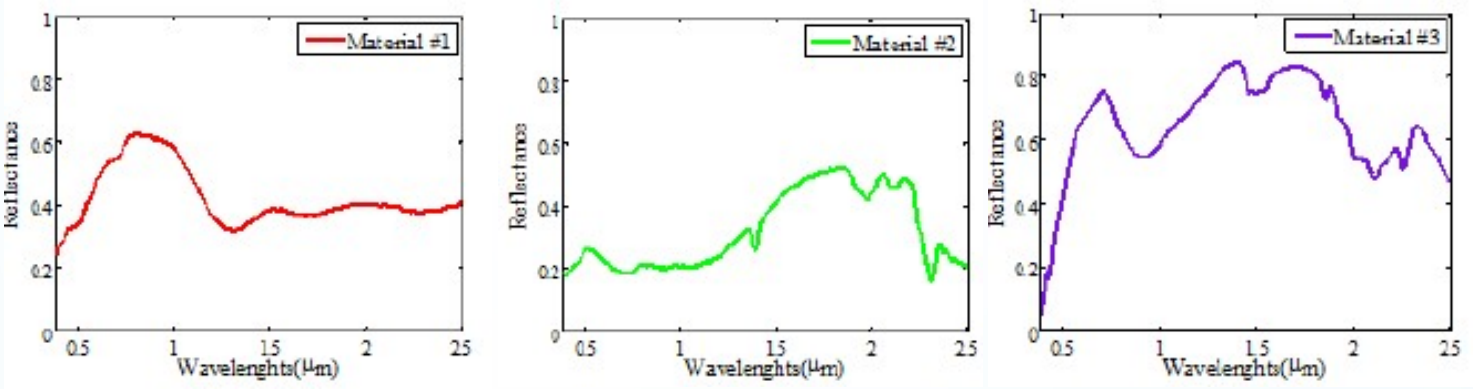
Spectral Unmixing

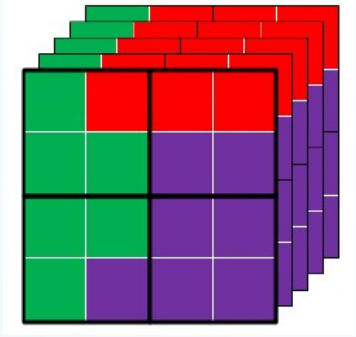
Mixed pixels are decomposed into a collection of constituent spectra (endmembers) and a set of corresponding fractions (abundances) through **spectral unmixing**



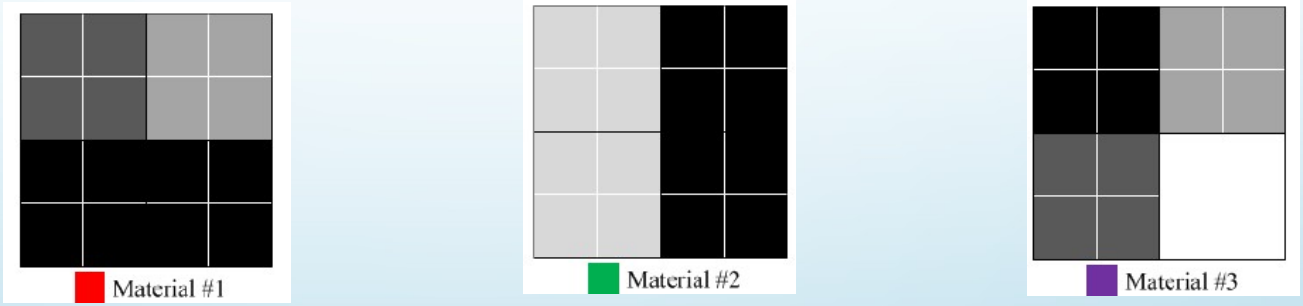
Unsupervised Spectral unmixing

Endmember Detection





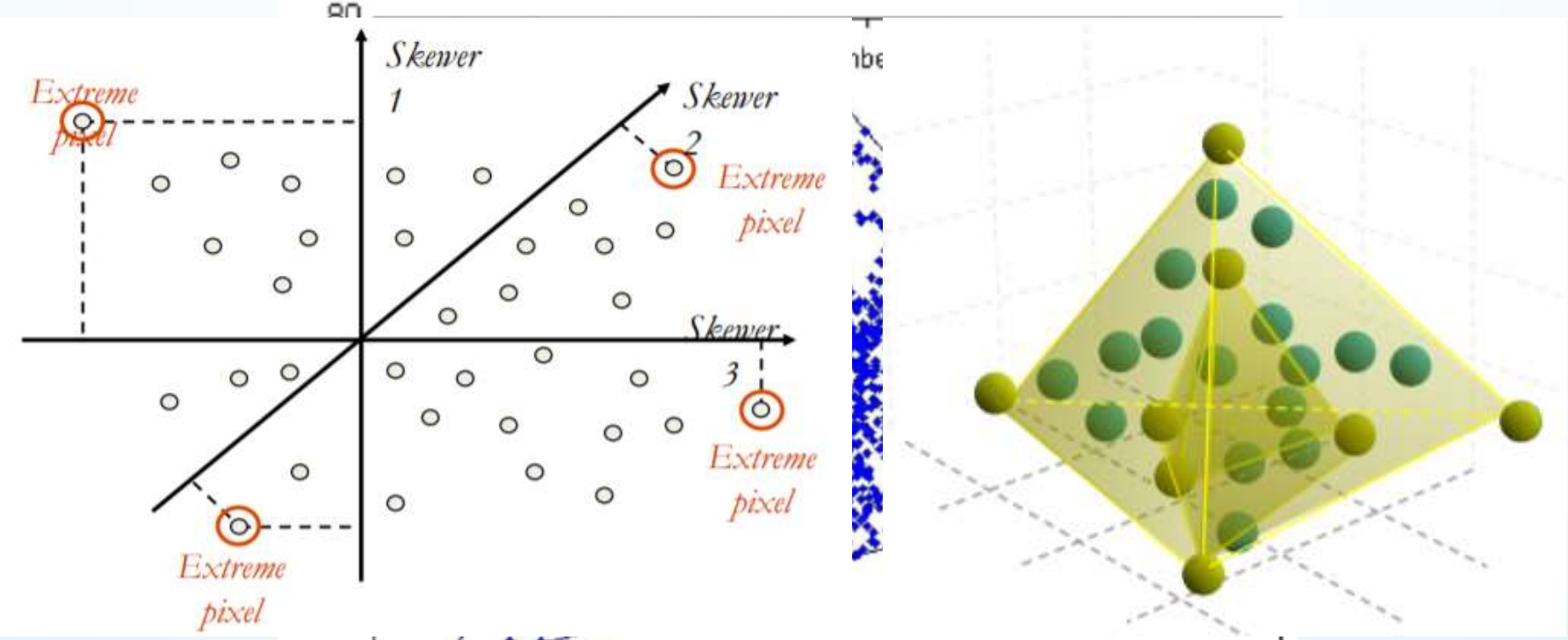
Abundance Estimation



Source: https://www.sfpt.fr/hyperspectral/wp-content/uploads/2013/01/cours_Licciardi.pdf

Unsupervised Spectral unmixing

Endmember Detection

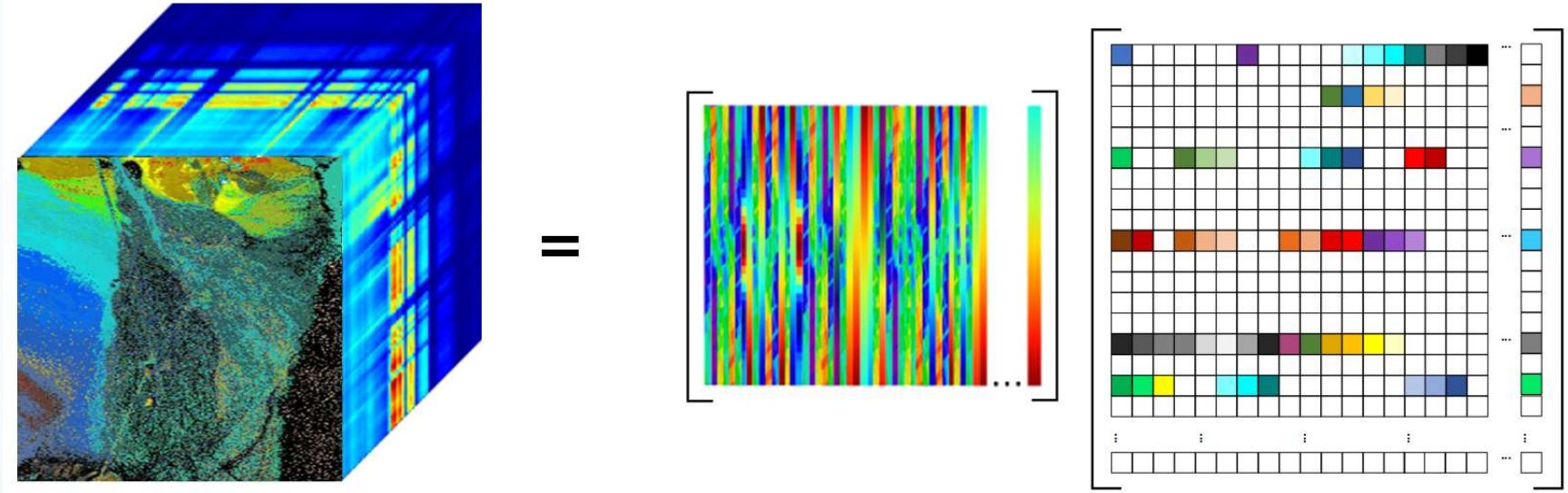


Pixel Purity Index **N_FINDR**

Source: https://www.sfpt.fr/hyperspectral/wp-content/uploads/2013/01/cours_Licciardi.pdf

Unsupervised Spectral unmixing

Abundance Estimation

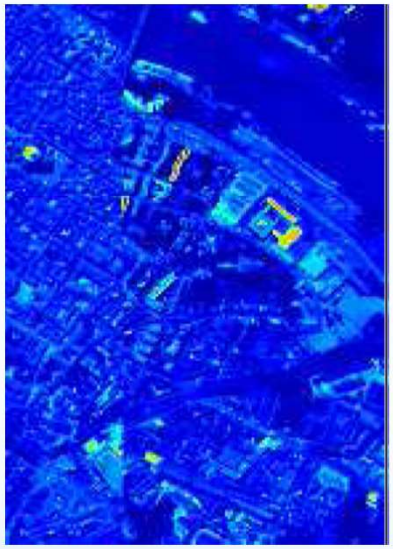


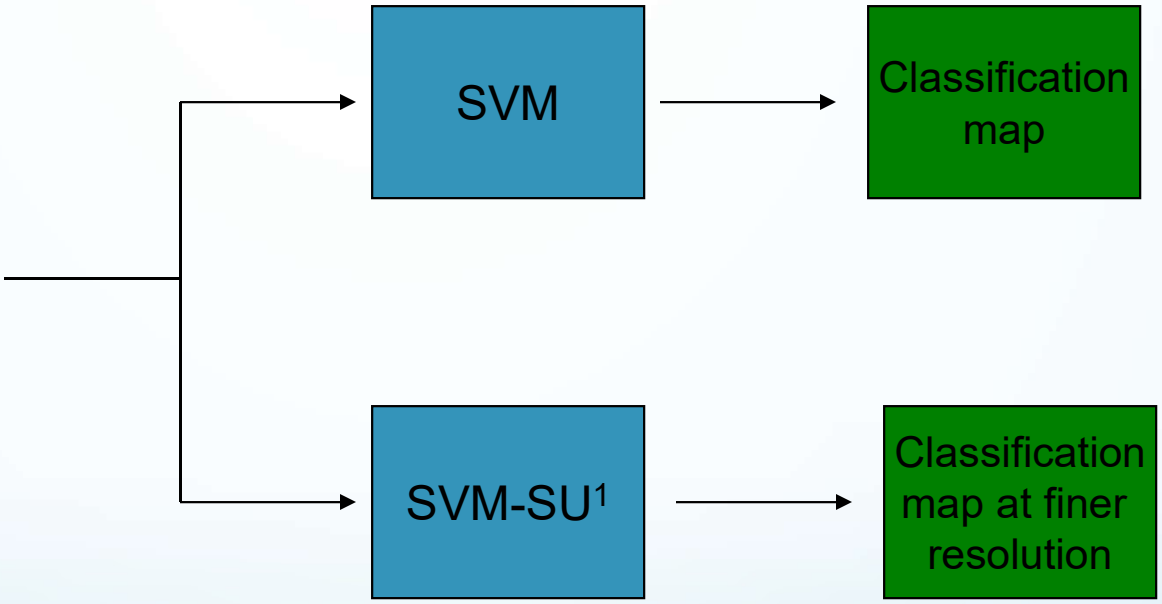
Hyperspectral Data (Y) = Spectral Library (A) Abundance Matrix (X)

$$Y = AX$$

$$\min_{A}(Y - AX) \;\; s.t. \;\; X > 0 \; and \; \sum_{j} A_j = 1$$

Supervised Spectral unmixing

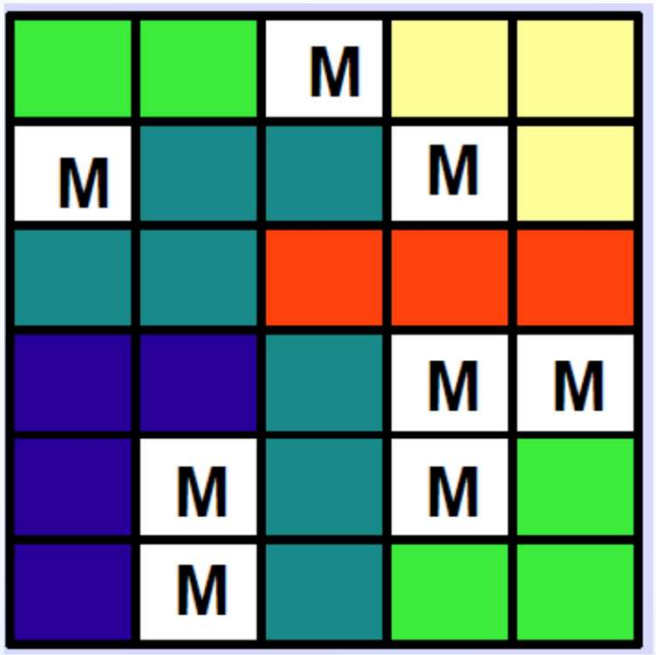




Low resolution image

1. Villa *et al.*, Spectral Unmixing to obtain classification maps at a finer resolution, *Journal of Selected Topics in Signal Processing*, 2011

SVM based Spectral unmixing



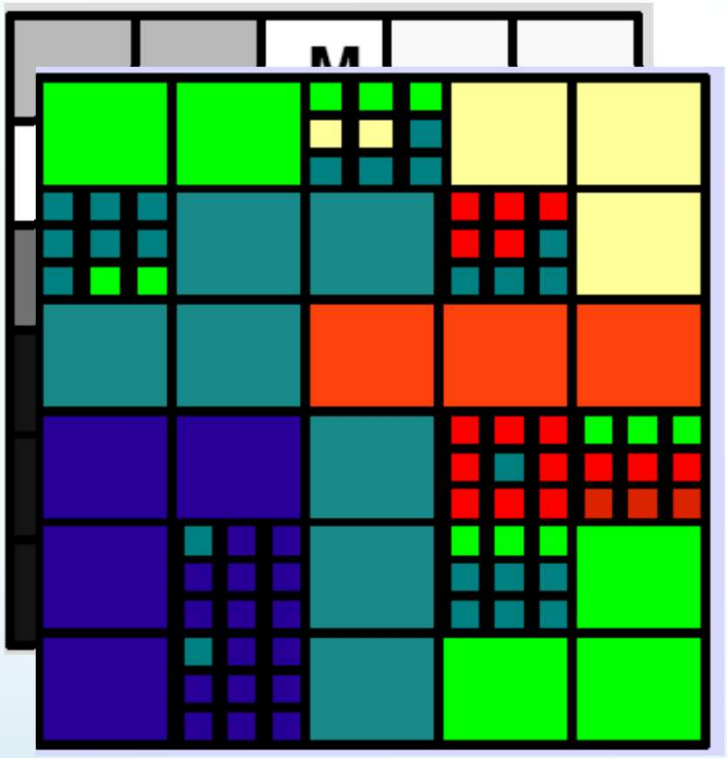
SVM – SPECTRAL UNMIXING(*):

1) PROBABILISTIC SVM DETERMINES WHICH PIXELS CAN BE CONSIDERED AS PURE (IF PROB > TRESHOLD)

2) SPECTRAL UNMIXING IS USED TO RETRIEVE CLASS ABUNDANCES WITHIN MIXED PIXELS

(*)A. Villa, J. Chanussot, J.A. Benediktsson and C. Jutten., Spectral Unmixing to obtain classification maps at a finer resolution, Journal of Selected Topics in Signal Processing, vol. 5, n. 3, May 2011

SVM based Spectral unmixing



SVM – SPECTRAL UNMIXING:

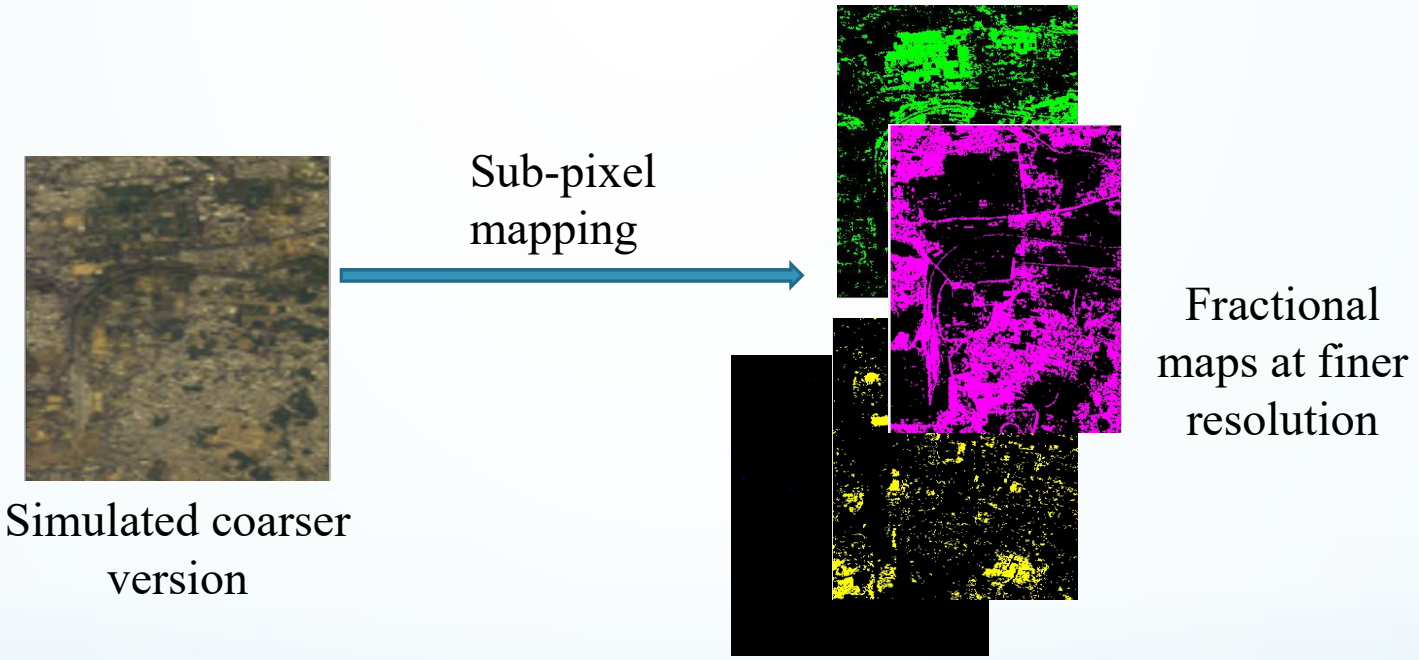
1) PROBABILISTIC SVM DETERMINES WHICH PIXELS CAN BE CONSIDERED AS PURE (IF PROB > TRESHOLD)

2) SPECTRAL UNMIXING IS USED TO RETRIEVE CLASS ABUNDANCES WITHIN MIXED PIXELS

AND TO FILL "UPSAMPLED" SUB-PIXELS

Sub-pixel classification

- Transforms fractional abundances to finer scale endmember (pure pixel) maps

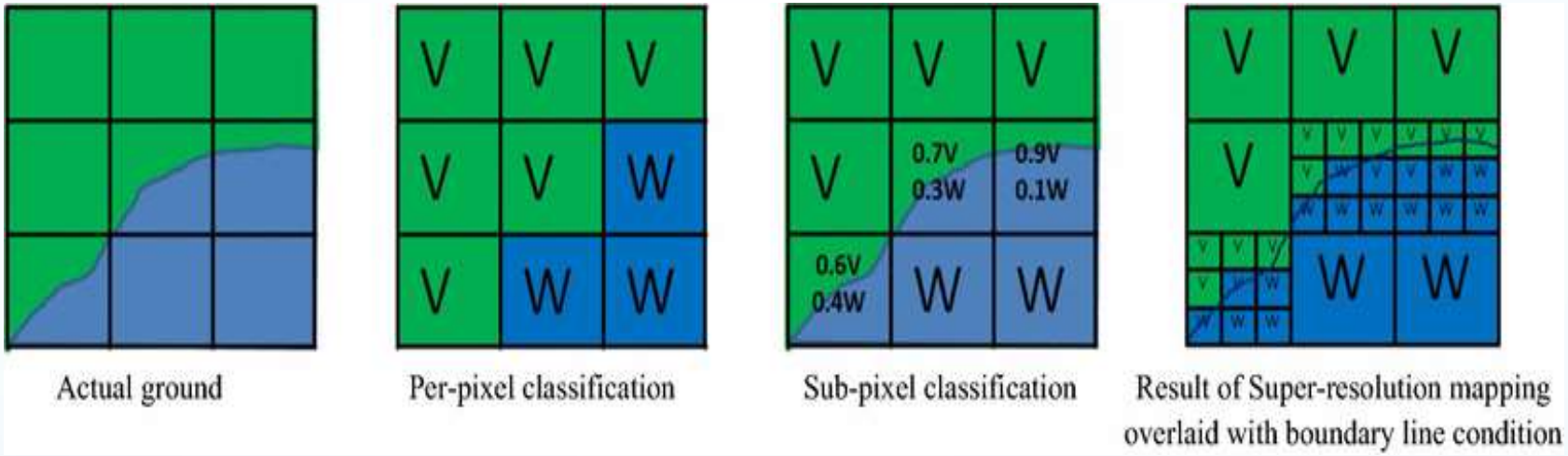


- Sub-pixel classification = spectral unmixing + finer spatial scale mapping

- Less ill posed as compared to super-resolution

Sub-pixel classification

Sub-pixel classification vs Perpixel Classifciation

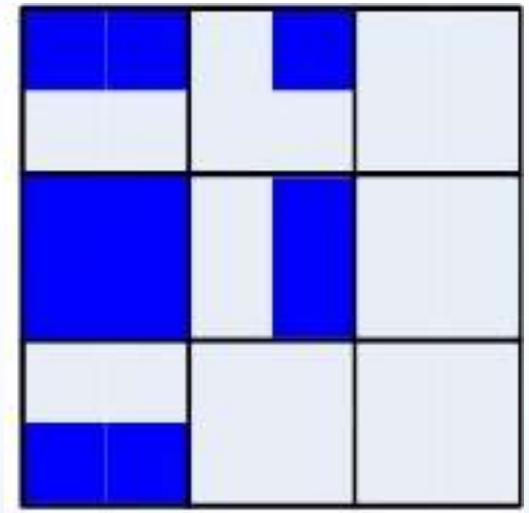


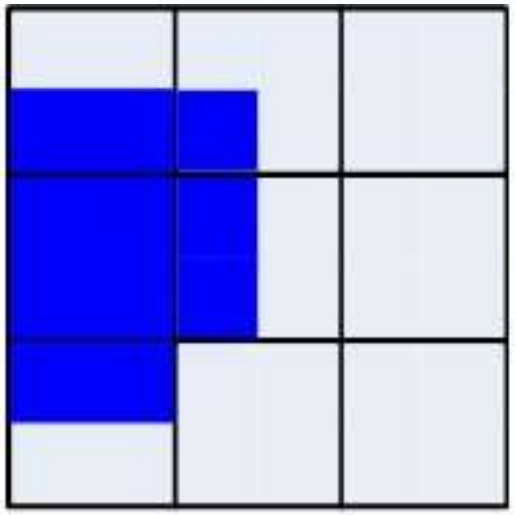
Source: https://www.sfpt.fr/hyperspectral/wp-content/uploads/2013/01/cours_Licciardi.pdf

Sub-pixel classification

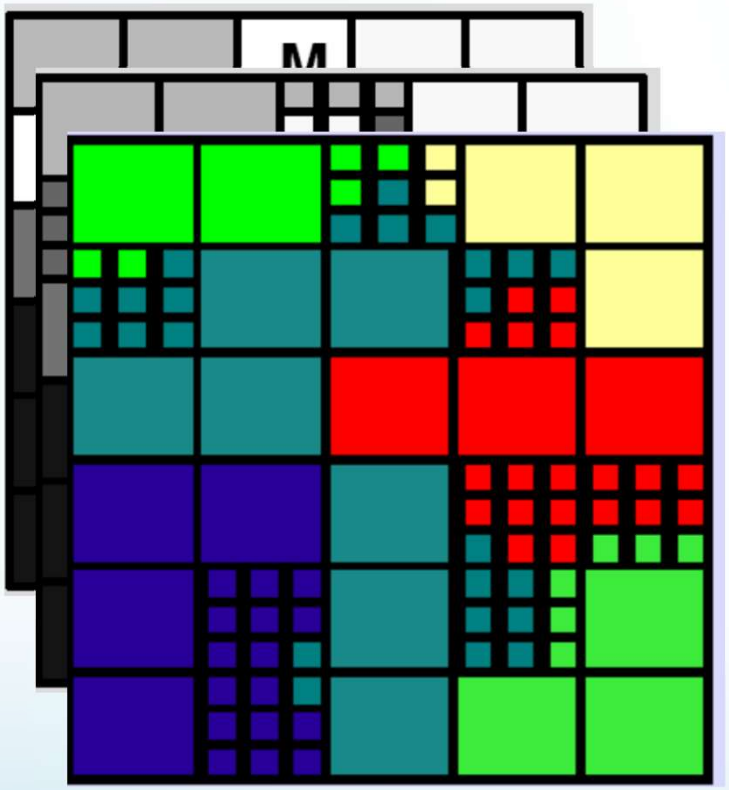
Sub-pixel mapping optimizes an initial configuration to maximize spatial contiguity

0.5	0.25	0
1.0	0.5	0
0.5	0	0





SVM based Sub-pixel classification



SVM – SPECTRAL UNMIXING:

1) PROBABILISTIC SVM DETERMINES WHICH PIXELS CAN BE CONSIDERED AS PURE (IF PROB > THRESHOLD)

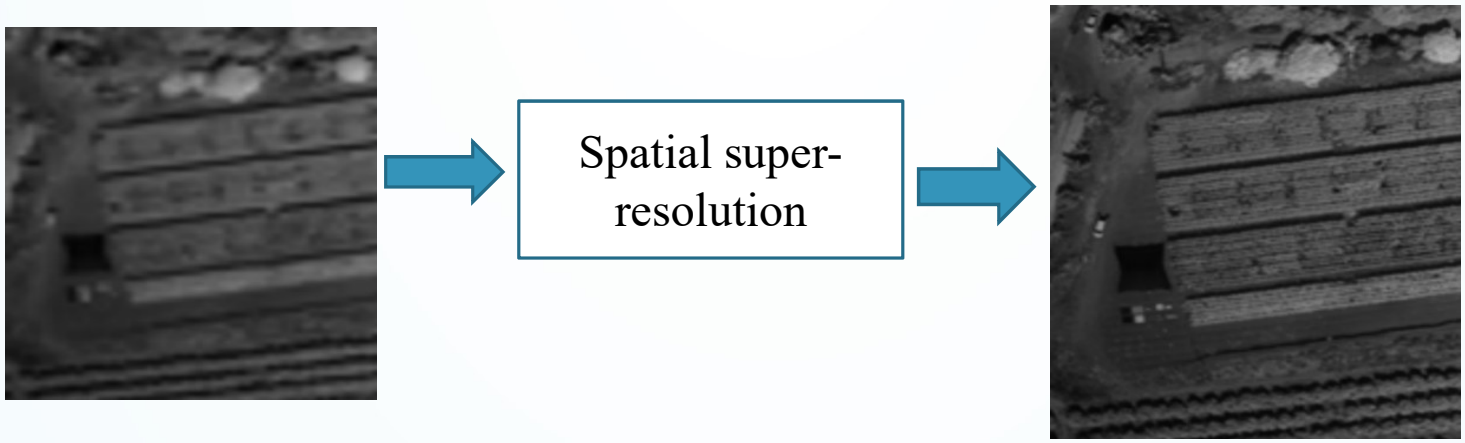
2) SPECTRAL UNMIXING IS USED TO RETRIEVE CLASS ABUNDANCES WITHIN MIXED PIXELS AND TO FILL "UPSAMPLED" SUB-PIXELS

3) FINAL SPATIAL REGULARIZATION IS PERFORMED (**COST FUNCTION** TO BE MINIMIZED: **TOTAL PERIMETER** OF THE CONNECTED AREAS)

The results is a thematic map at a finer resolution → useful to assess possibilities offered by HSI at low-medium spatial resolution.

Spatial super-resolution

- Reconstruction of finer scale images from coarser ones



Input coarse image

High resolution image

- Entire hypercube need to be considered simultaneously to ensure spectral fidelity
- Hyperspectral SISR is highly ill posed

Machine Learning Algorithms

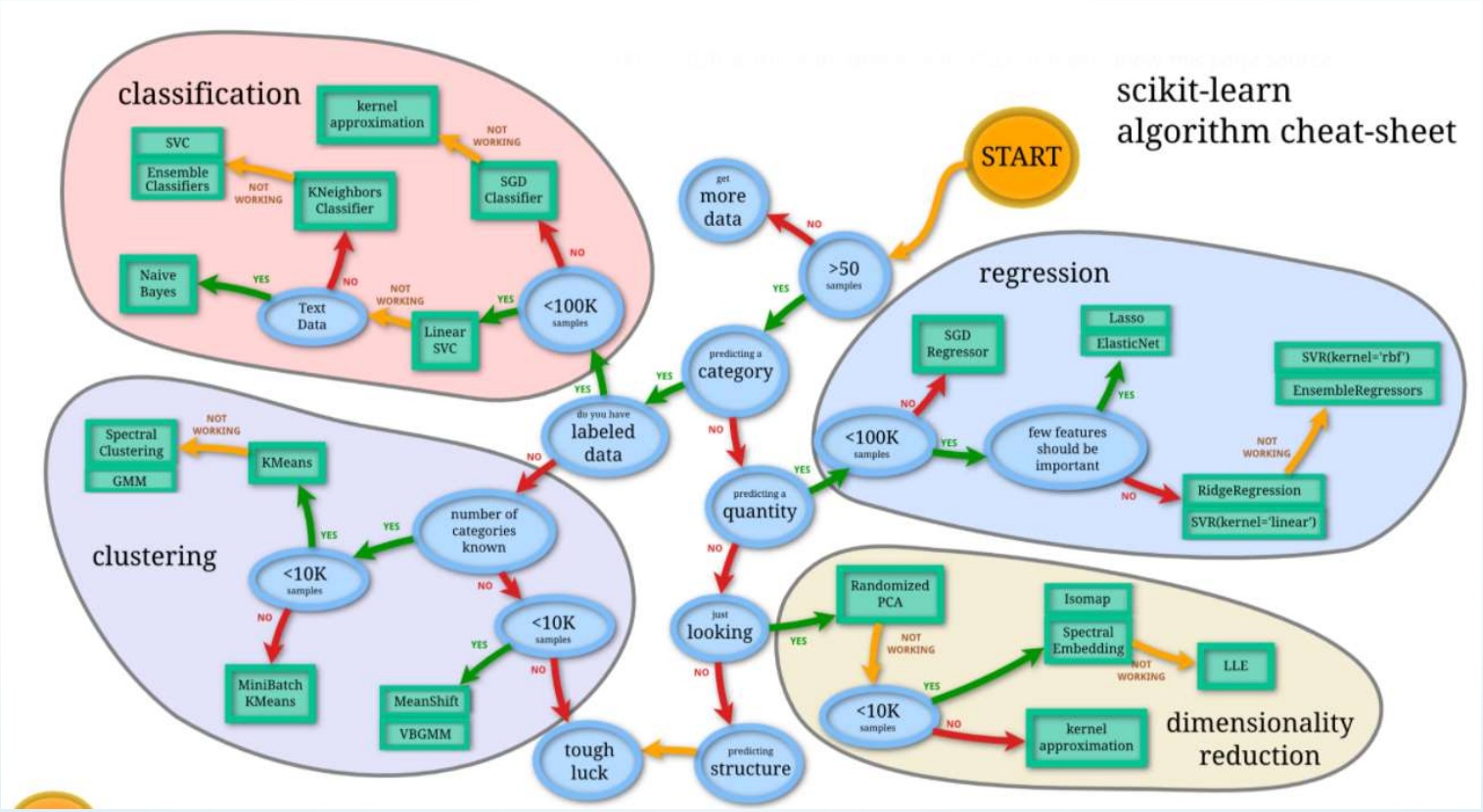
Unsupervised clustering techniques group together similar pixels and assign same label

- K-means,
- Fuzzy C-means, etc…

Supervised classification algorithms assign label to an unknown pixel based on the available training data

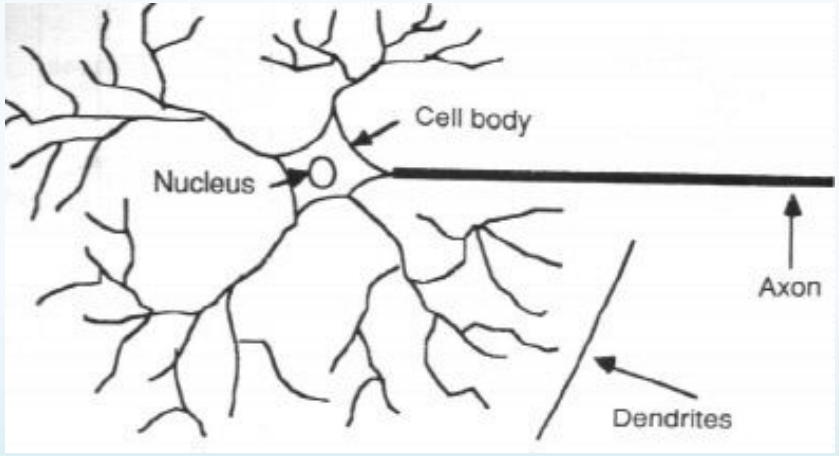
- Neural network
- Support Vector Machines
- Decision trees, etc…

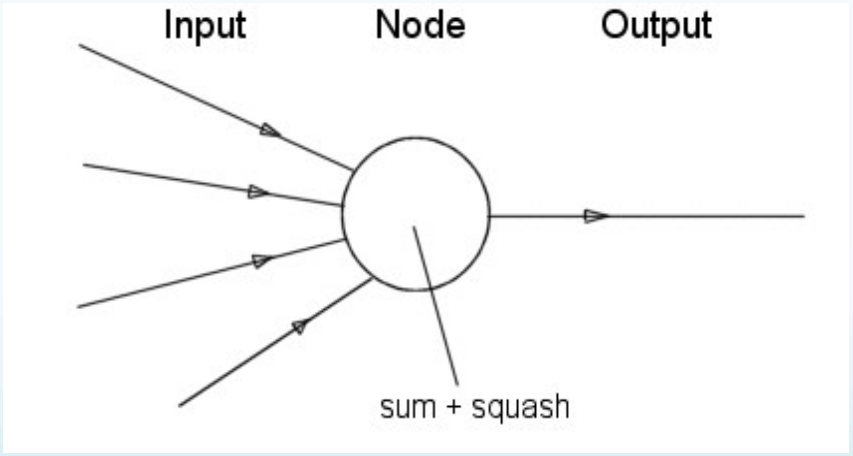
Machine Learning Algorithms



Artificial Neural Networks

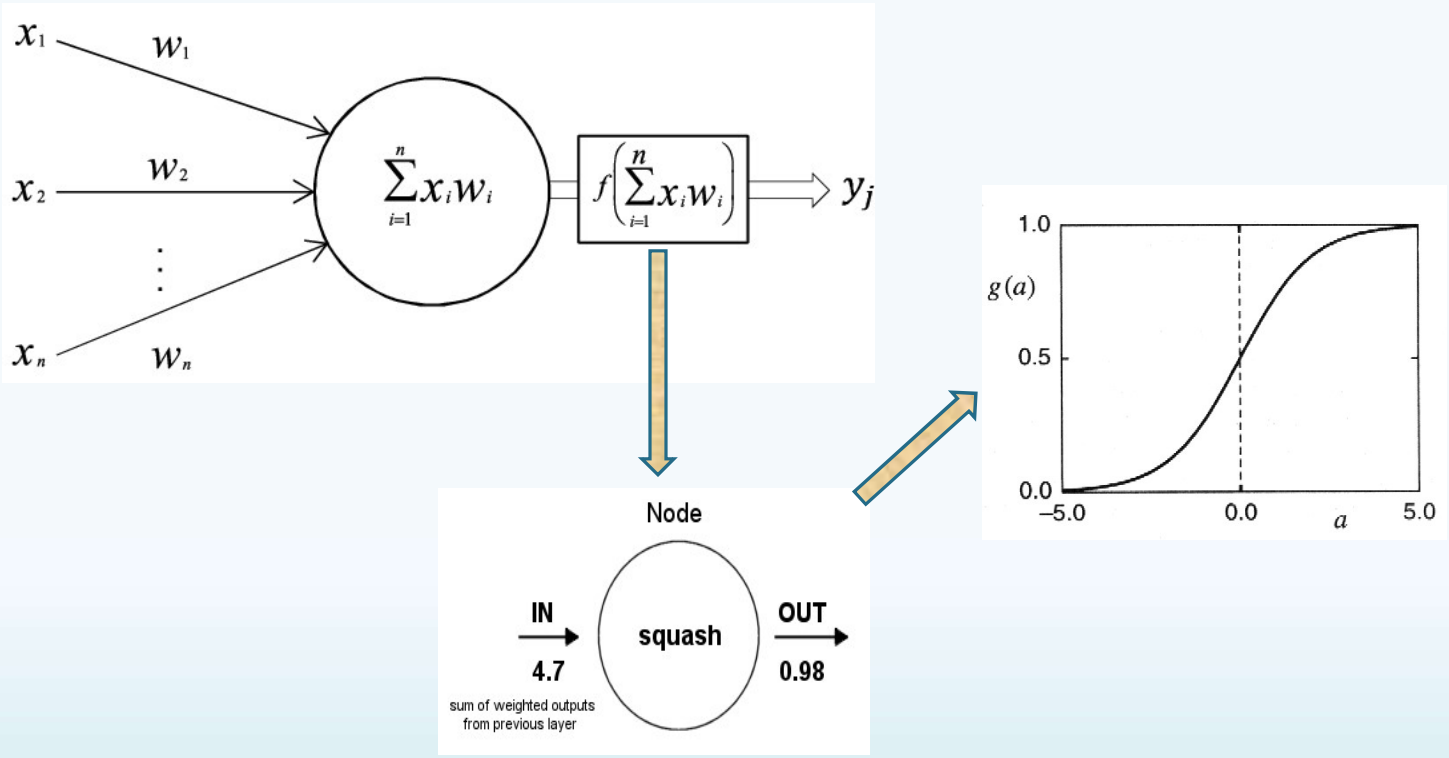
- ANNs are computational models that mimic biological neurons
- Each nodes compute weighted sum of the input data which is squashed to yield the output





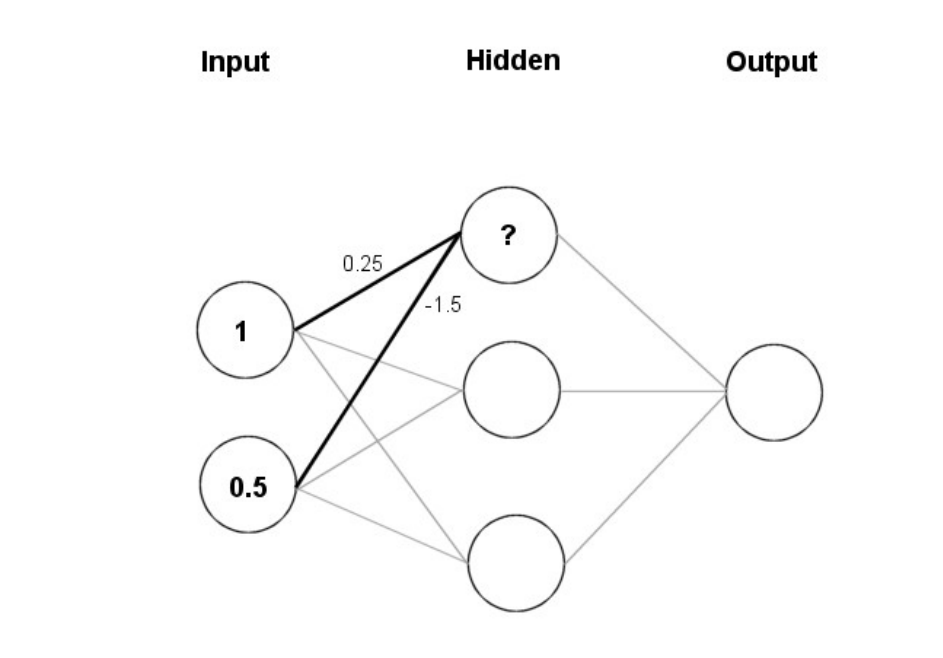
Source: https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b

Artificial Neural Networks



Source: https://www.semanticscholar.org/paper//4807bf70ab59417a295d92/figure/0

Artificial Neural Networks



$$(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) = \mathbf{-0.5}$$

Squashing: $$\frac{1}{1+e^{0.5}} = 0.3775$$

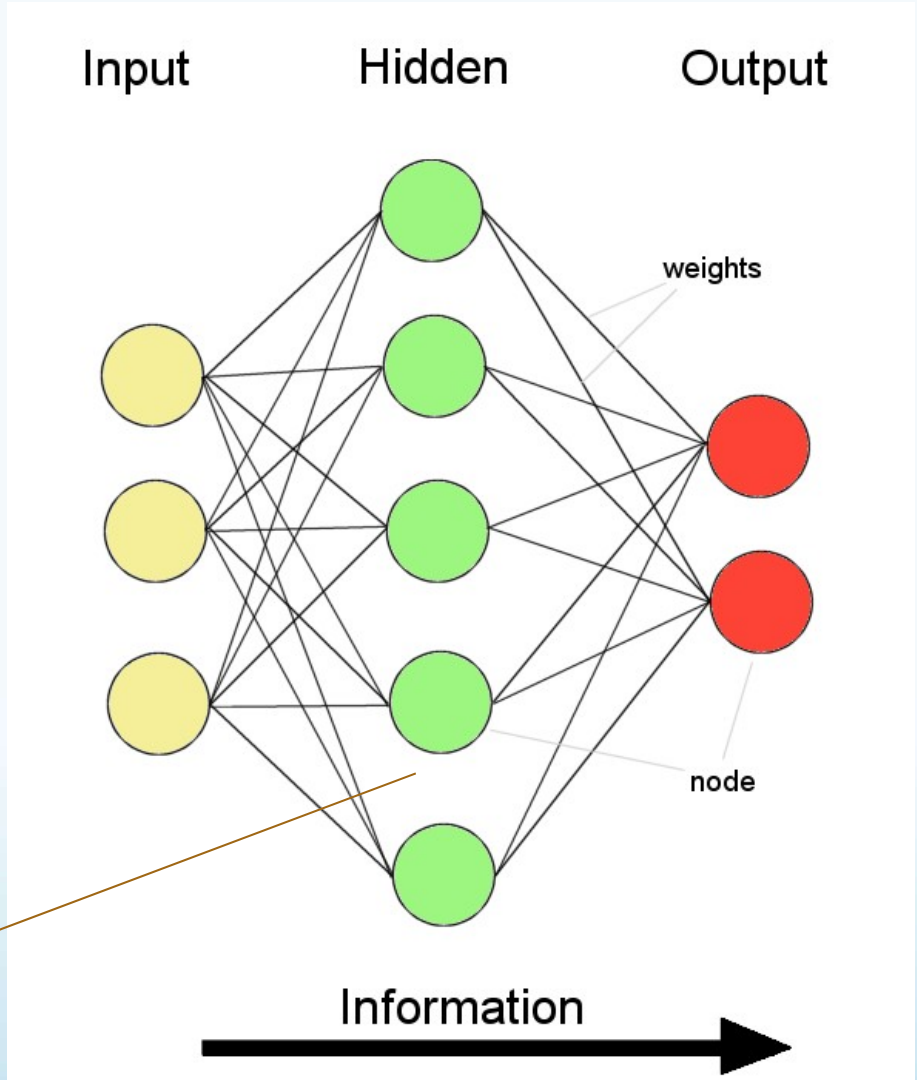
Artificial Neural Networks

Information flow in ANN

Data is presented to *Input layer*

Passed on to *Hidden Layer*

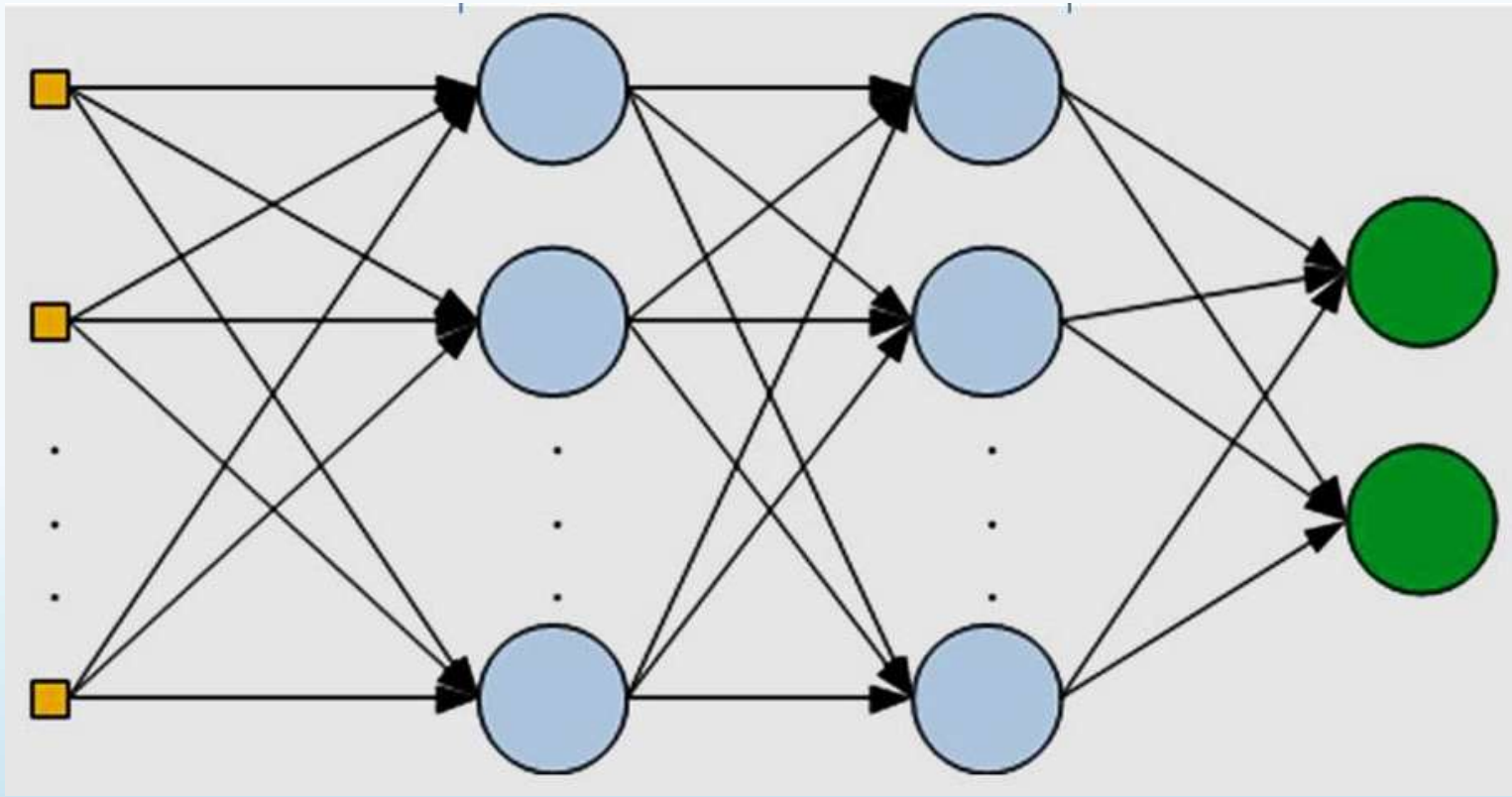
Passed on to *Output layer*



Internal representation (interpretation) of data

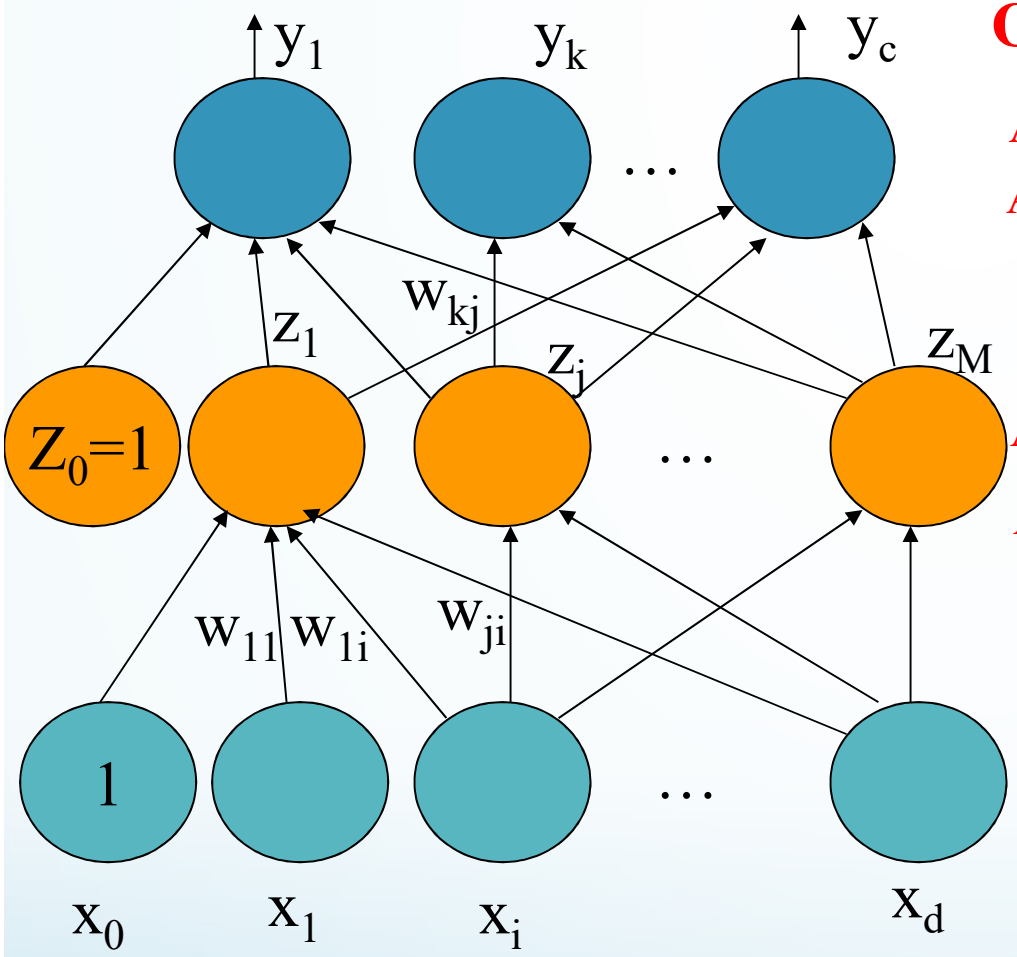
Source: https://rpubs.com/justin-garza/698638

Multi-Layer Feed-forward Networks



Source: https://sebastianraschka.com/faq/docs/

Two-layer neural network



Output

Activation function: f (linear,sigmoid, softmax)

Activation of unit a_k: $\sum_{j=0}^{M} w_{kj} z_j$

Activation function: g (linear, tanh, sigmoid)

Activation of unit a_j: $\sum_{i=0}^{d} w_{ji} x_i$

$$y_k = f\left(\sum_{j=0}^{M} w_{kj} \times g\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right)$$

Adjust weights by training

- How to adjust weights?
- Adjust weights using known examples (training data) (x_1,x_2,x_3,…,x_d,t).
- Try to adjust weights so that the difference between the output of the neural network y and t (target) becomes smaller and smaller.
- Goal is to minimize error (difference) as we did for one layer neural network

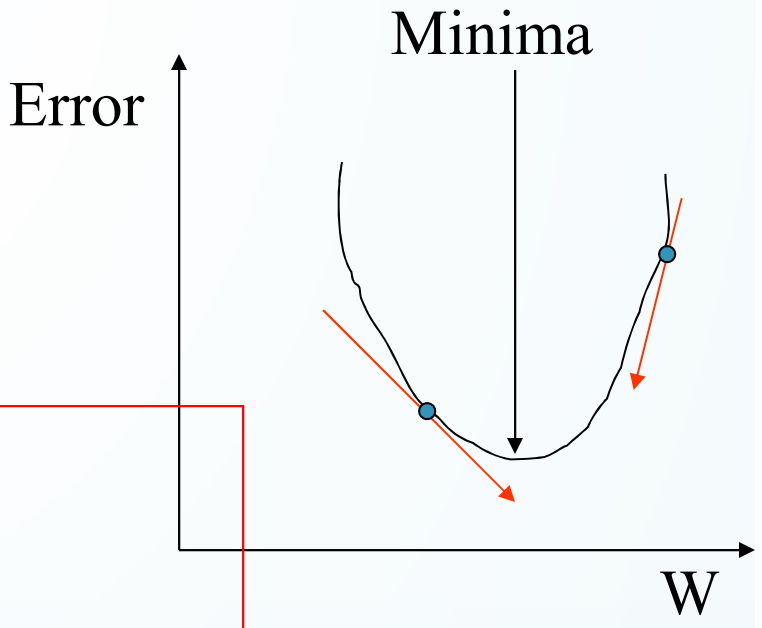
Adjust weights using gradient descent (back-propagation)

Known:

Data: $(x_1,x_2,x_3,\ldots,x_n)$ target t.

Unknown weights w:

$w_{11}, w_{12},\ldots..$

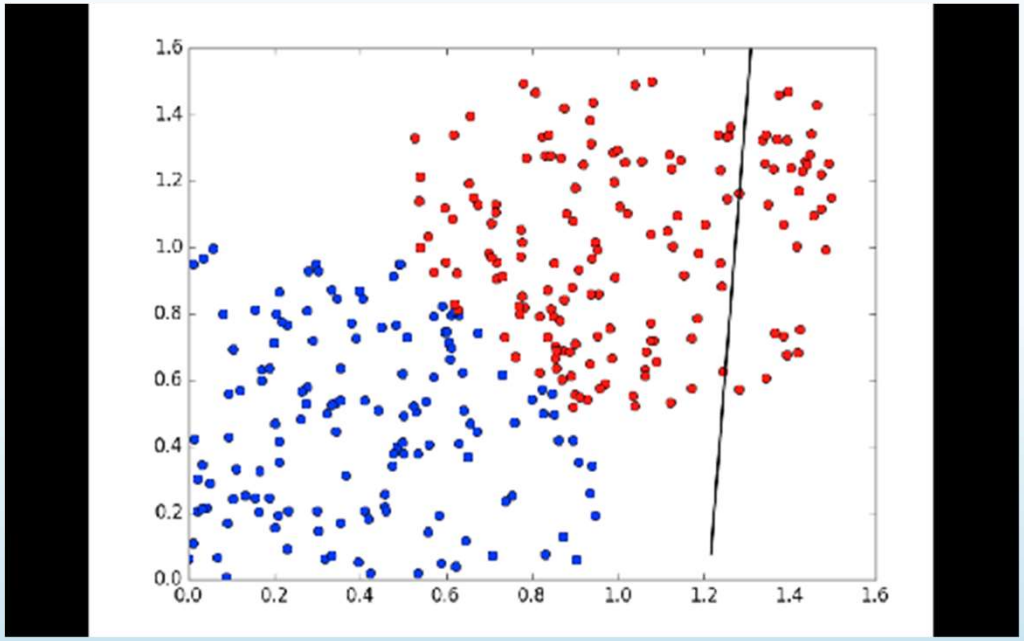


Randomly initialize weights
Repeat
for each example, compute output y
calculate error $E = (y-t)^2$
compute the derivative of E over w: $dw = \frac{\partial E}{\partial w}$
$w_{new} = w_{prev} - \eta * dw$
Until error doesn't decrease or max num of iterations

Note: η is learning rate or step size.

Artificial Neural Networks

- Data is presented to the network in the form of activations in the input layer
  - Each spectral band is a variable (four band image: 4 input nodes)
- Error is propagated backwards to train the network
- Generate decision surfaces based on the training data that separate the data



Source: https://devashishshankar.wordpress.com/tag/deep-learning/

Neural network learning

- Forward propagation: present an example (data) into neural network. Compute activation into units and output from units.

- Backward propagation: propagate error back from output layer to the input layer and compute derivatives (or gradients).

Neural network learning

Initialize weights w

Repeat

For each data point x, do the following:

Forward propagation: compute outputs and activations

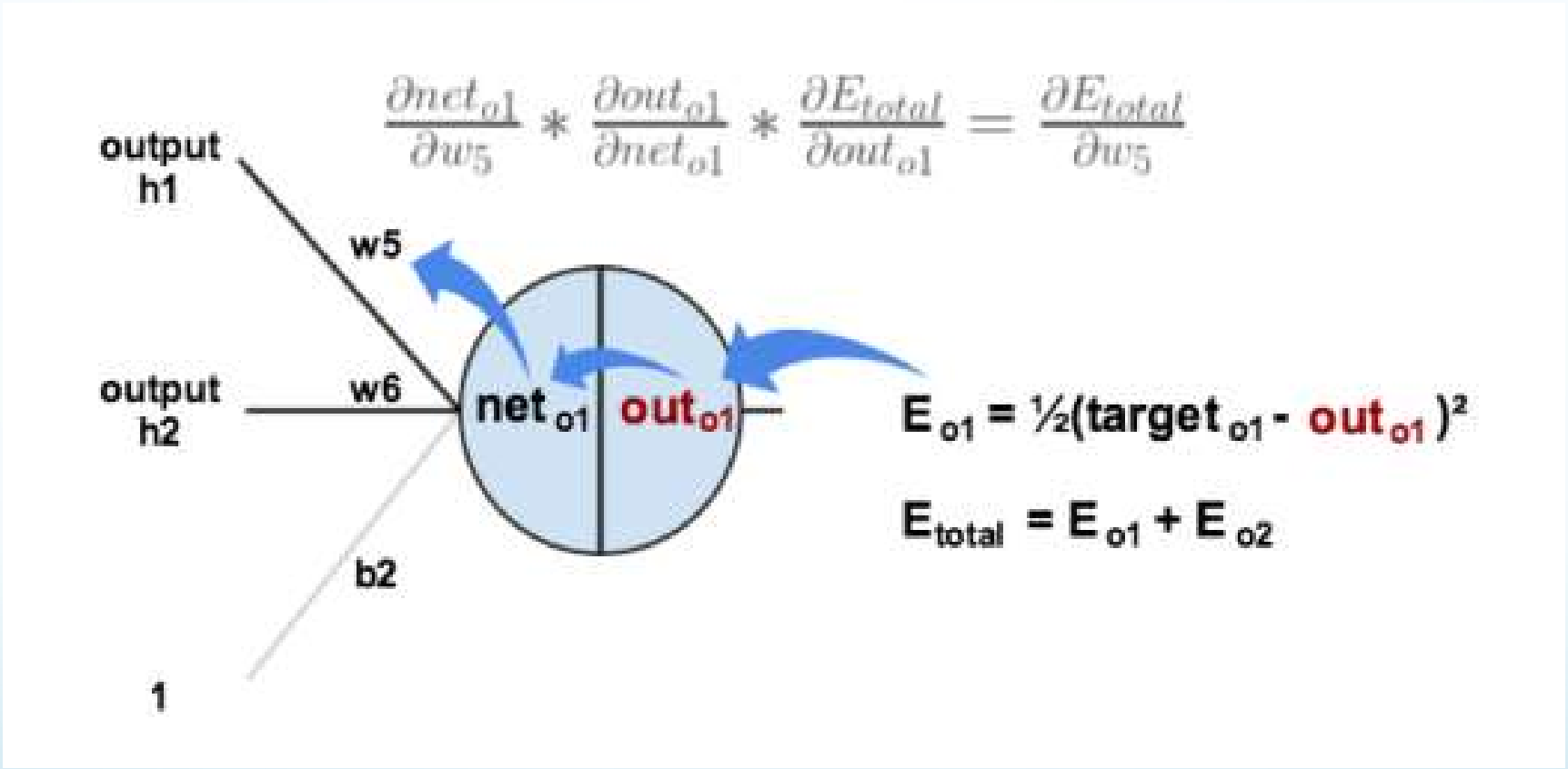
Backward propagation: compute errors for each output units and hidden units. Compute gradient for each weight.

Update weight $w = w - \eta \, (\partial e / \partial w)$

Until a number of iterations or errors drops below a threshold.

Neural network learning

Error propagation for output layer

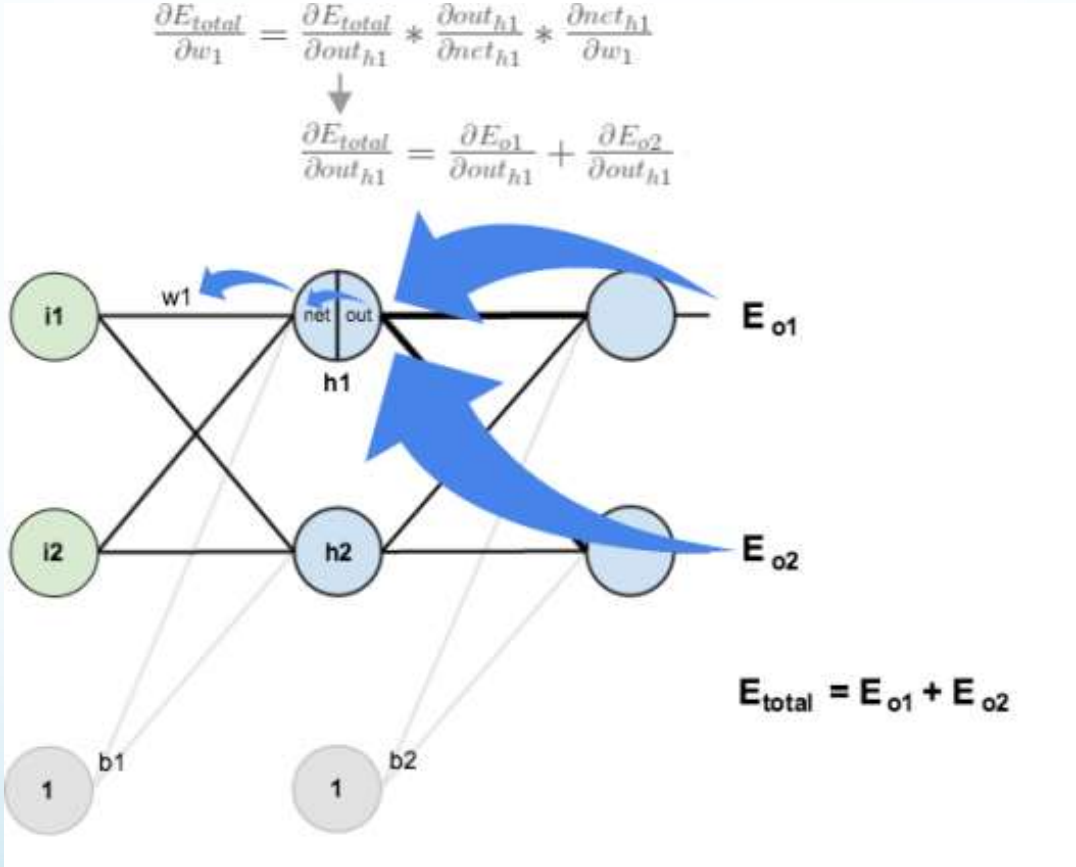


Neural network learning

Error propagation for hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



Neural network learning

Error propagation for hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

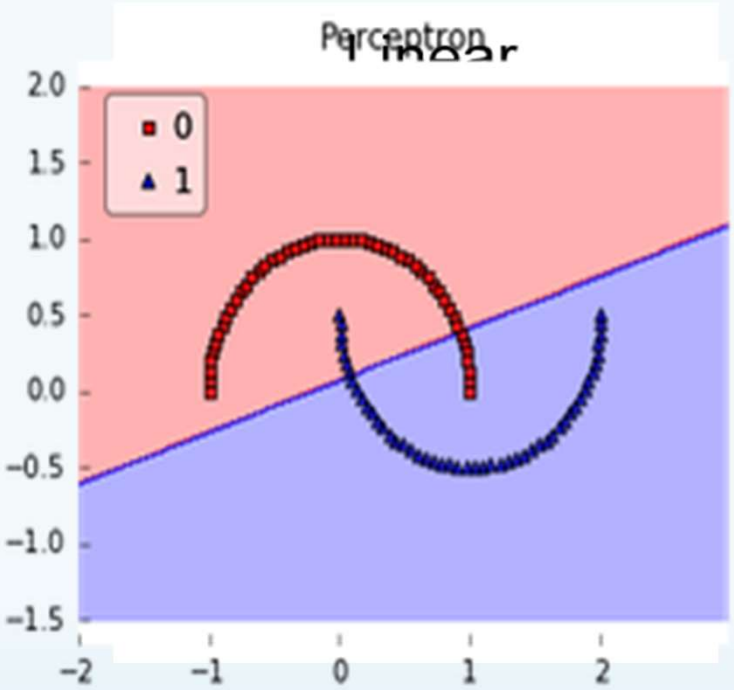
$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \delta_o * w_{ho}) * out_{h1}(1 - out_{h1}) * i_1$$

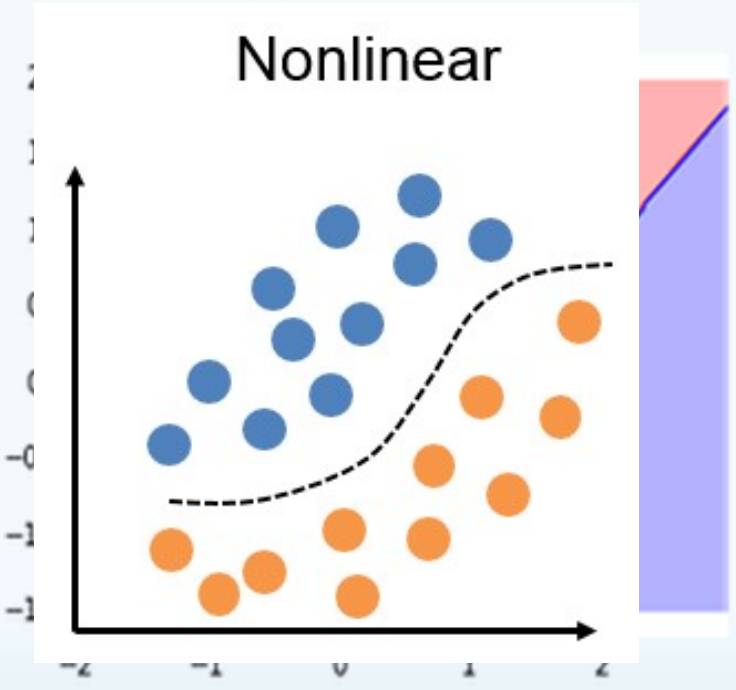
$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

For a detailed worked out example:

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

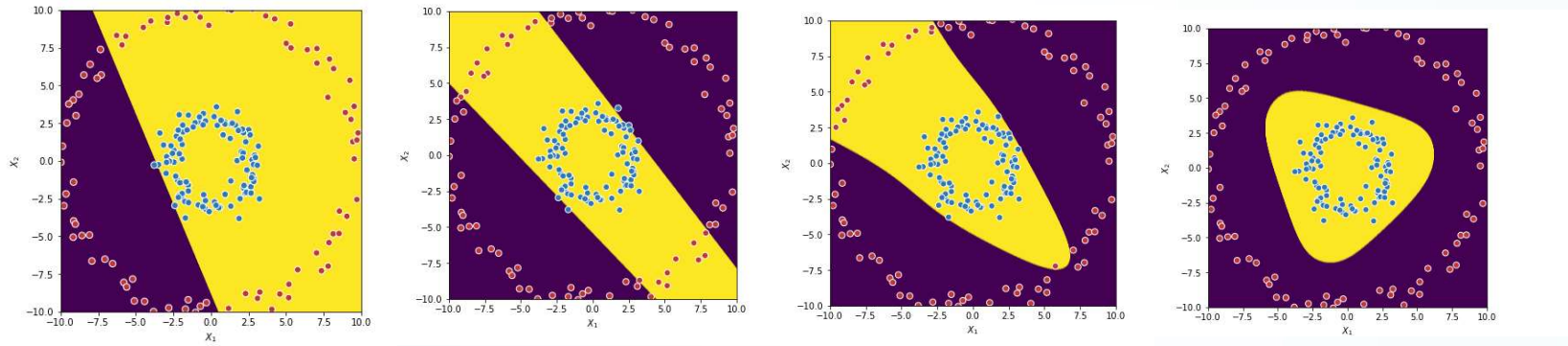
Decision boundaries of ANNs



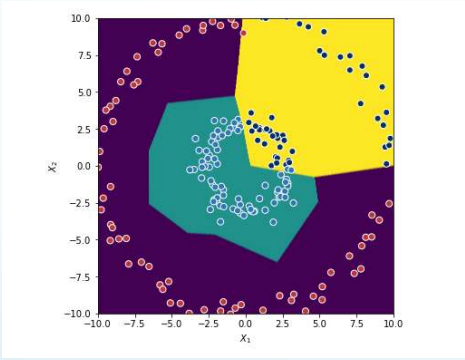


Source: https://sebastianraschka.com/faq/docs/

Decision boundaries of ANNs



Number of Neurons =1 to 10 in a single layer with non linear activation



Number of Neurons = 10 for a 3 class problem

Overfitting

The training data contains information about the regularities in the mapping from input to output. But it also contains noise

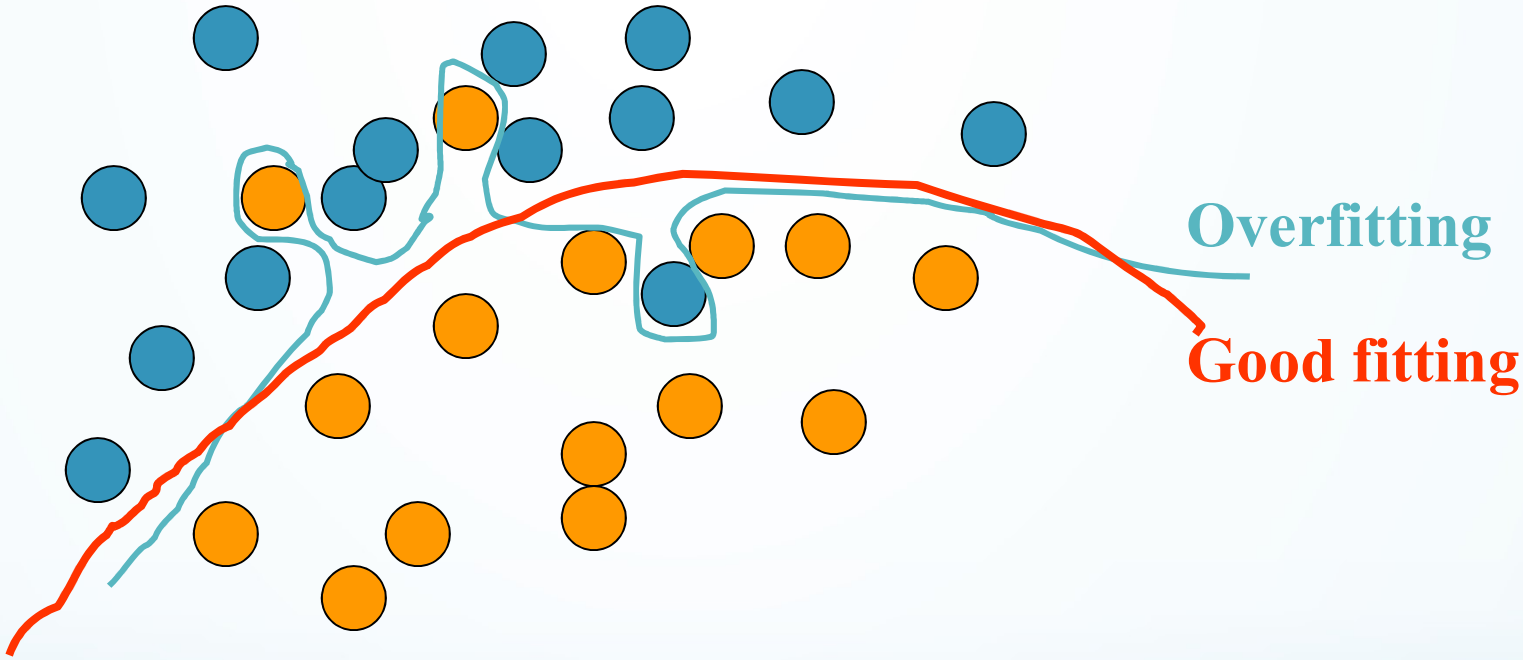
- The target values may be unreliable.
- There is sampling error. There will be accidental regularities just because of the particular training cases that were chosen.

When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.

- So it fits both kinds of regularity.
- If the model is very flexible it can model the sampling error really well. This is a disaster.

G. Hinton, 2006

Example of Overfitting and Good Fitting



Overfitting function can not generalize well to unseen data.

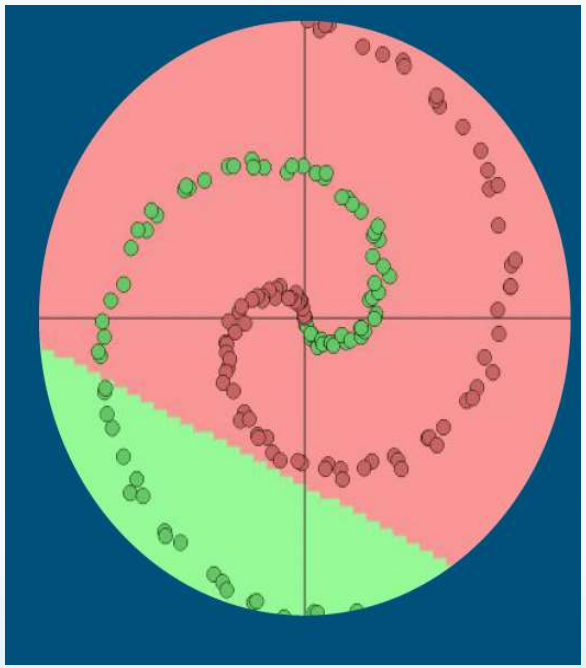
Preventing Overfitting

- Use A model that has the right capacity:
  - Enough to model the true regularities
  - Not enough to also model the spurious regularities (assuming they are weaker).
- Standard ways to limit the capacity of a neural net:
  - Limit the number of hidden units.
  - Limit the size of the weights.
  - Stop the learning before it has time to overfit.

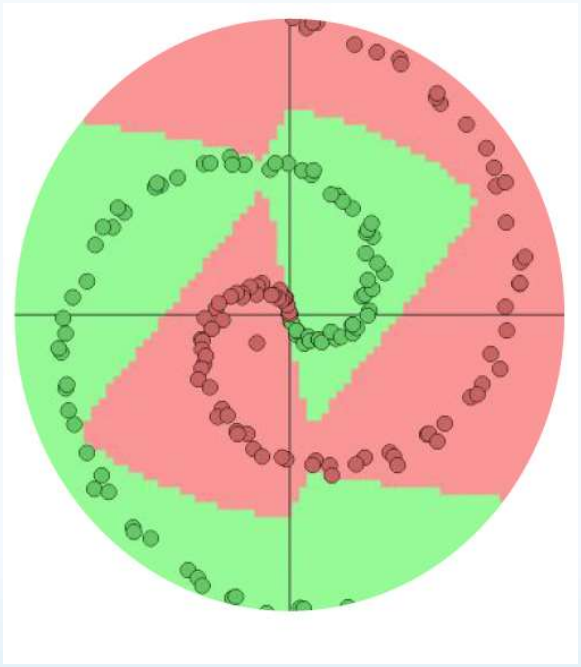
G. Hinton, 2006

Deeper the Better?

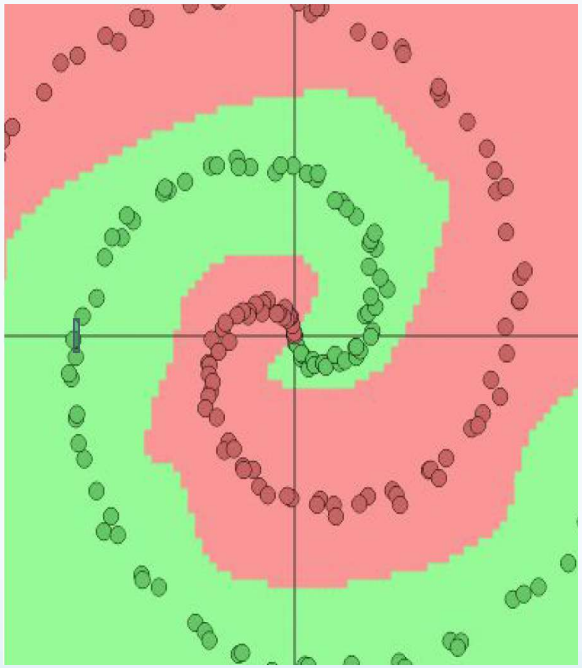
Deeper networks can better model complex decision surfaces



Depth=1



Depth=2



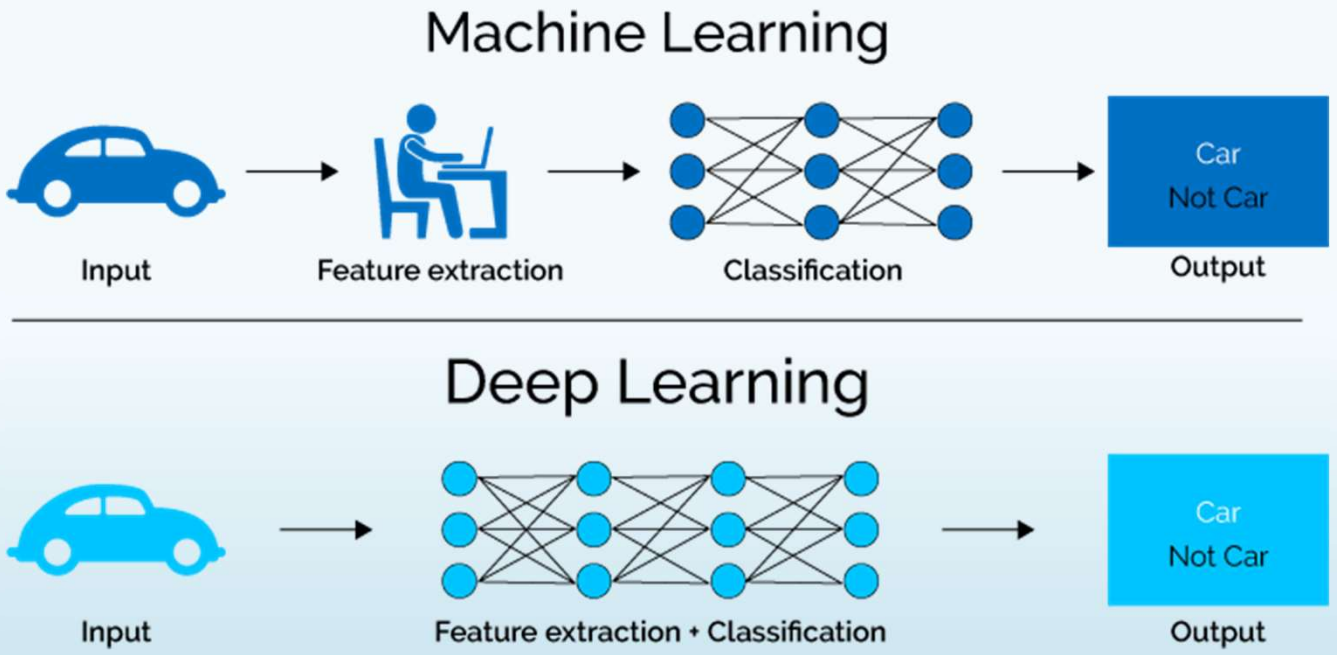
Depth=3

Source: http://cs.stanford.edu/people/karpathy/convnetjs/

What is Deep Learning (DL) ?

A machine learning subfield for learning **representations** of data. Exceptionally effective at **learning patterns (spectral and spatial)**.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**



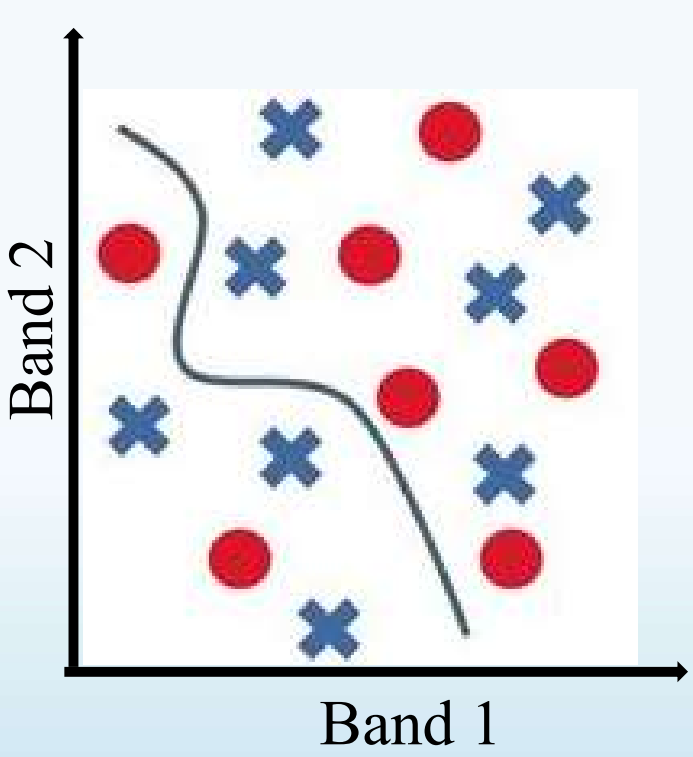
Source: https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

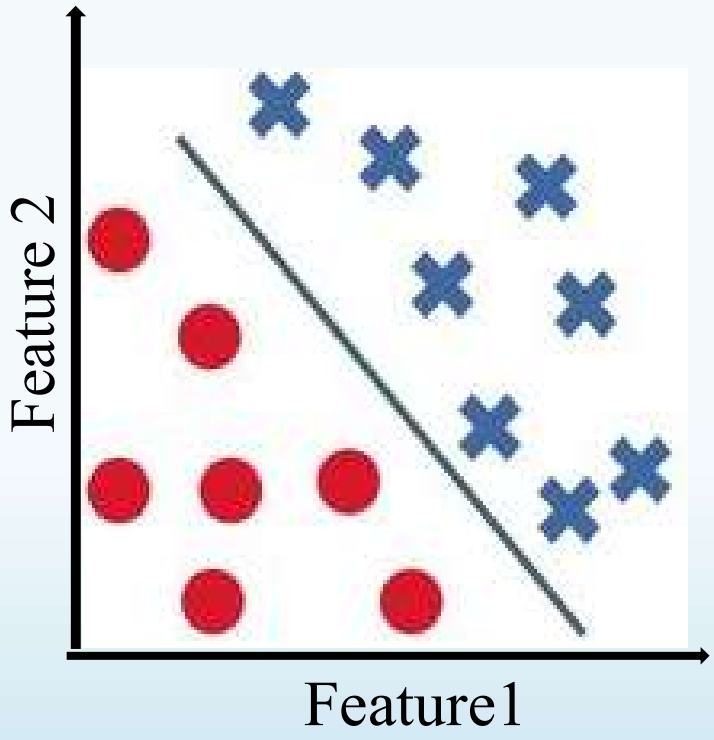
Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn in both unsupervised and supervised manner
- Can learn **spatial context**
- Effective **end-to-end** joint system learning

Image Space Vs Learned Feature Spaces

Input data is more separable in the learned feature space



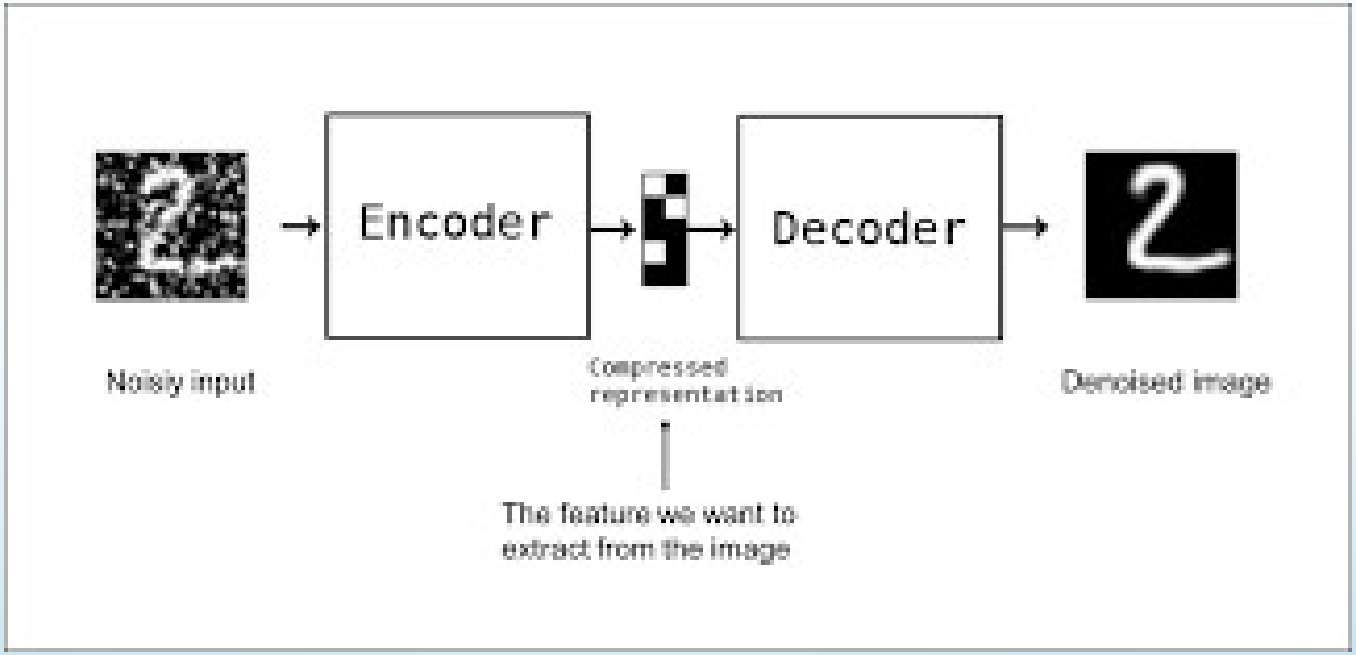


Source: Salakhutdinov et. al. ICML, 2007, Salakhutdinov and Mnih, 2008

How to Learn Features?

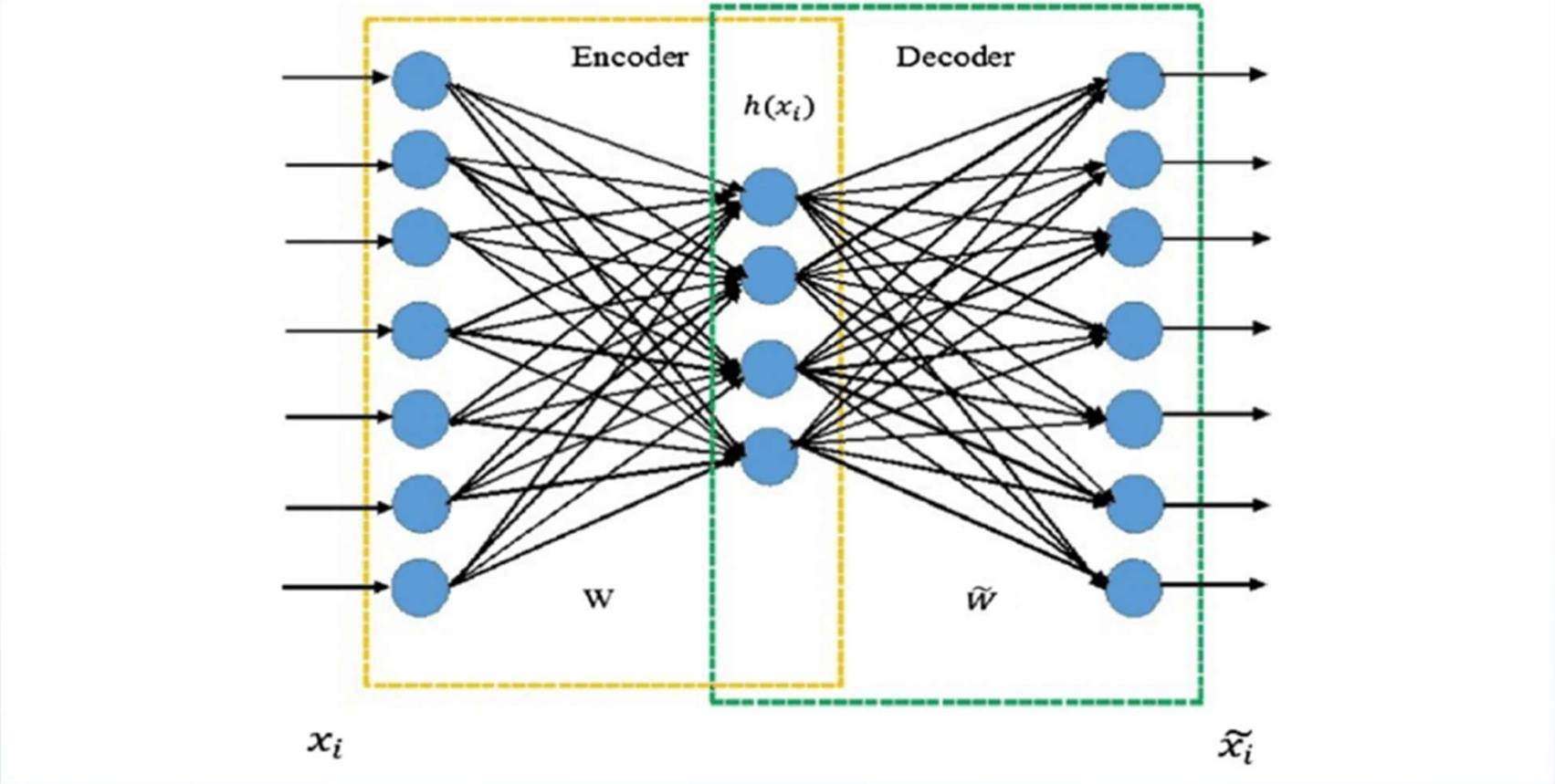
Autoencoders are employed to learn features that can be used for classification and other modelling purposes.

To avoid overfitting and trivial mapping, denoising, variational, and contractive autoencoder versions are used.



Source: Ahmed et al., 2018

Autoencoder



Source: Ahmed et al., 2018

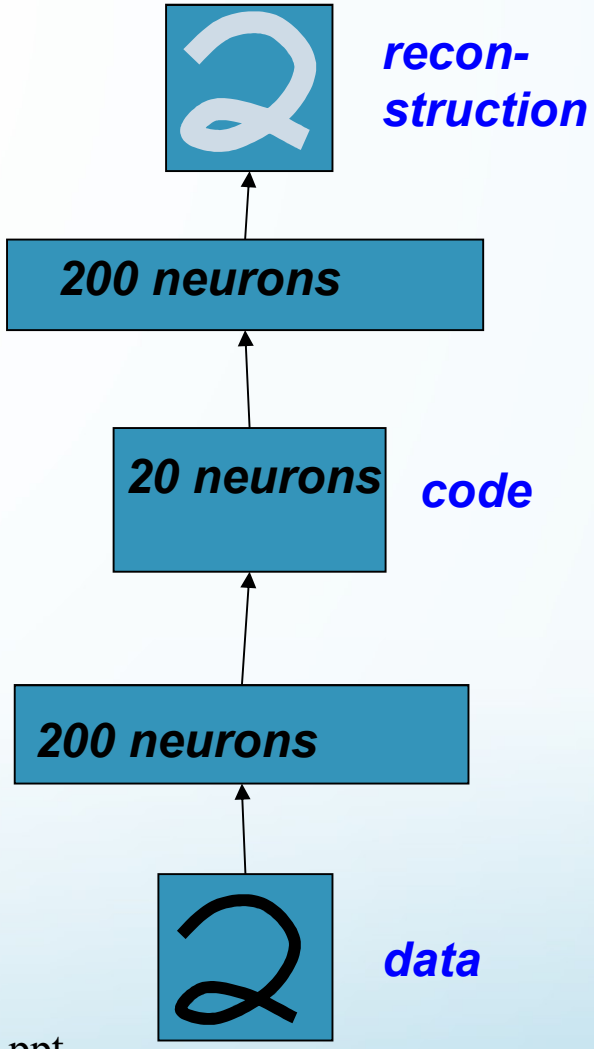
Self-supervised Backpropagation

Autoencoders define the desired output to be the same as the input.

- Trivial to achieve with direct connections

It is useful if we can squeeze the information through some kind of bottleneck:

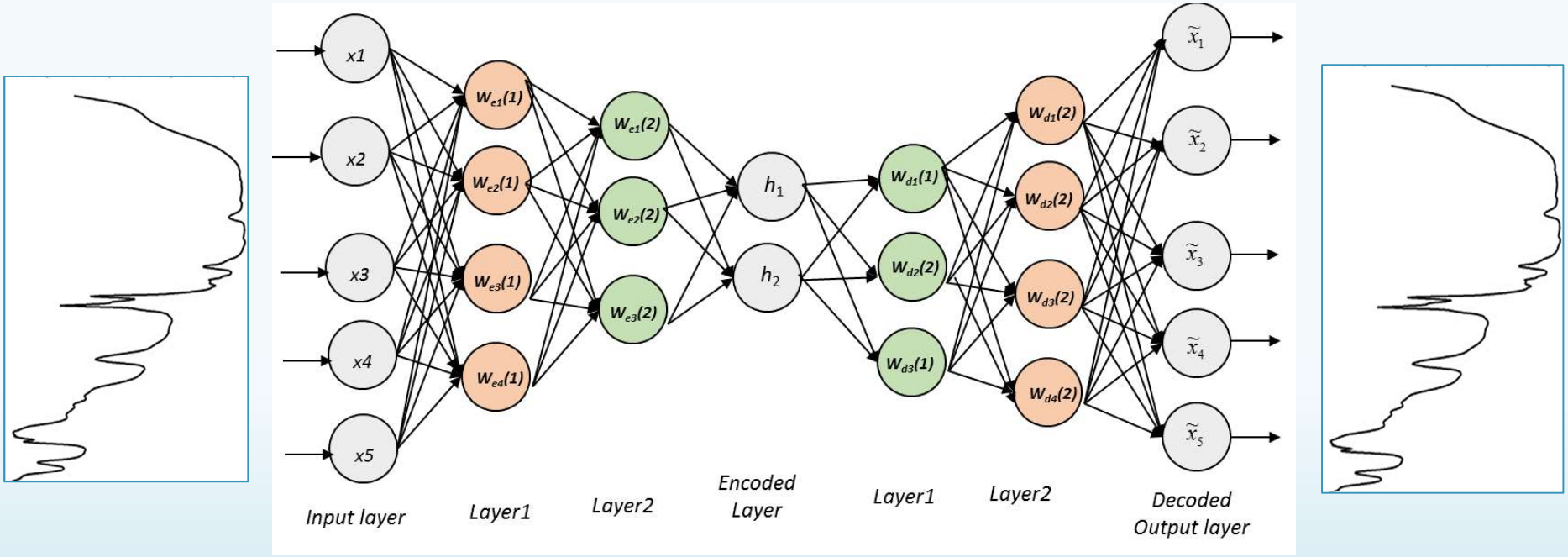
- If we use a linear network this is very similar to principal components analysis



Source: https://www.cs.toronto.edu/~hinton/csc2515/notes/lec7post.ppt

Autoencoder

Representations can be learned in the hidden layers of autoencoders



Source: Ahmed et al., 2018

Self-supervised Backprop and PCA

- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared reconstruction error.
- The m hidden units will span the same space as the first m principal components
  - Their weight vectors may not be orthogonal
  - They will tend to have equal variances

Self-supervised Backprop in Deep Autoencoders

- We can put extra hidden layers between the input and the bottleneck and between the bottleneck and the output.
  - This gives a non-linear generalization of PCA
- It should be very good for non-linear dimensionality reduction.
  - It is very hard to train with backpropagation
  - So deep autoencoders have been A big disappointment.

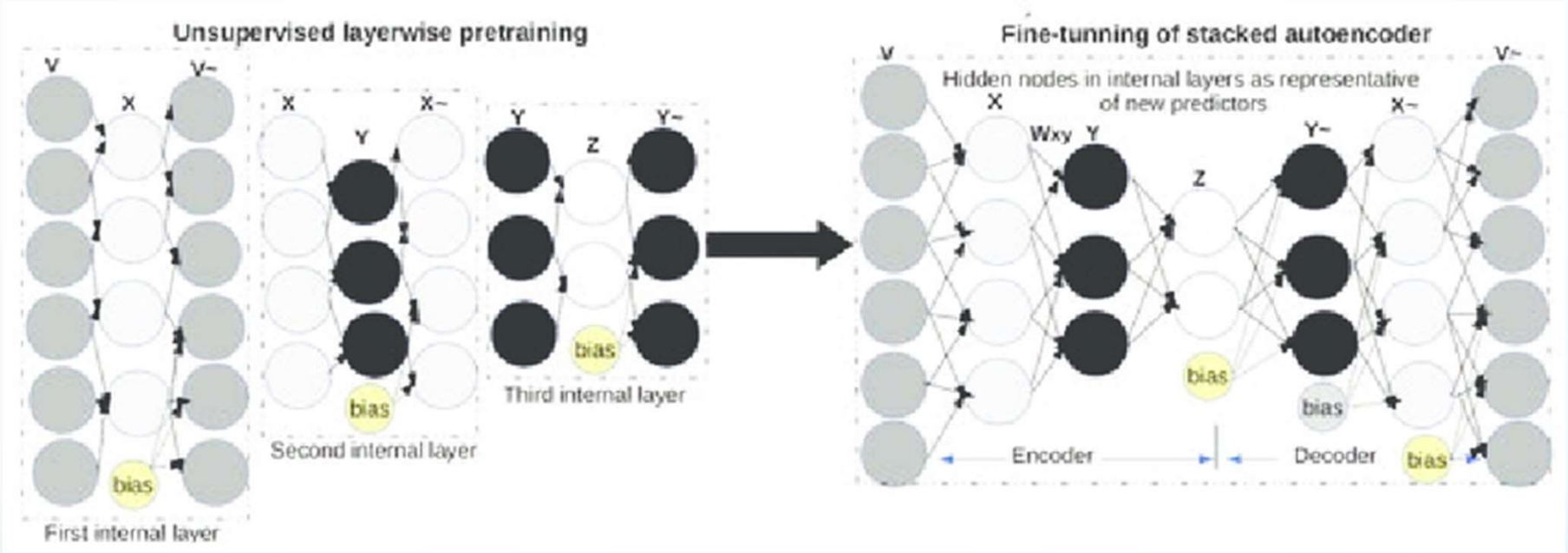
Some Problems with Back Propagation

- The amount of information that each training case provides about the weights is at most the log of the number of possible output labels.
  - So to train a big net we need lots of labeled data.
- In nets with many layers of weights the backpropagated derivatives either grow or shrink multiplicatively at each layer.
  - Learning is tricky either way.
- Dumb gradient descent is not a good way to perform a global search for a good region of a very large, very non-linear space.
  - So deep nets trained by backpropagation are rare in practice.

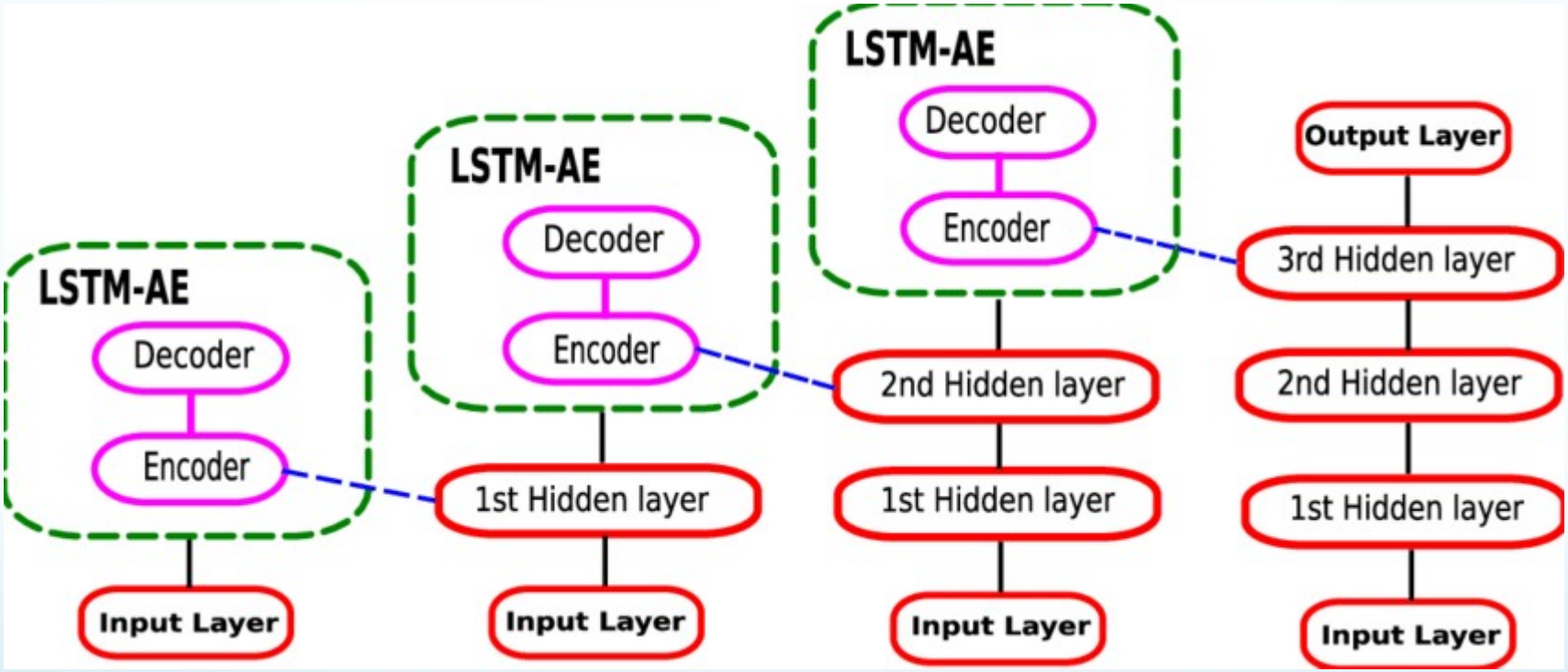
Layer wise pre training

- Use greedy unsupervised learning to find a sensible set of weights one layer at a time. Then fine-tune with backpropagation.
- Greedily learning one layer at a time scales well to really deep networks.
- Most of the information in the final weights comes from modeling the distribution of input vectors.
  - The precious information in the labels is only used for the final fine-tuning.
- We do not start backpropagation until we already have sensible weights that already do well at the task.
  - So the fine-tuning is well-behaved and quite fast.

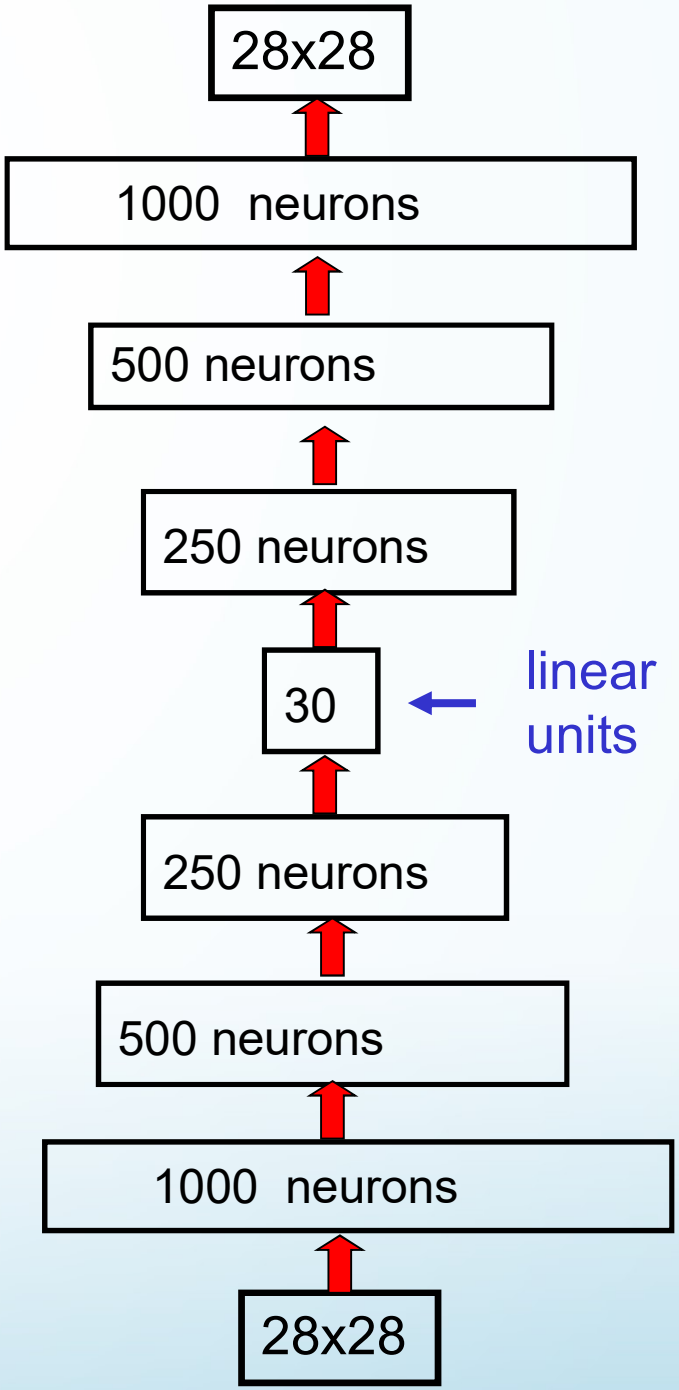
Layer wise pre training



Layer wise pre training



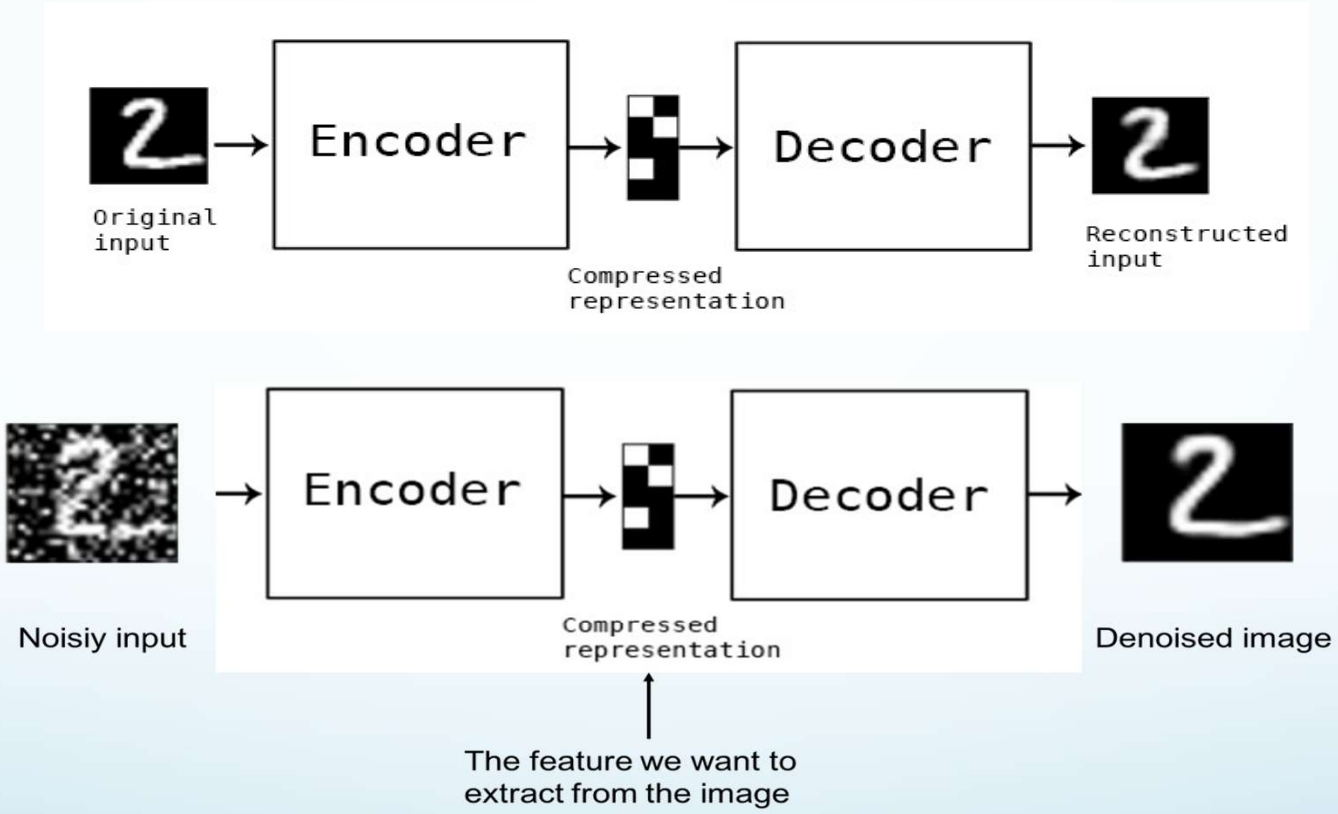
Stacked Autoencoder



Source: https://www.cs.toronto.edu/~hinton/csc2515/notes/lec7post.ppt

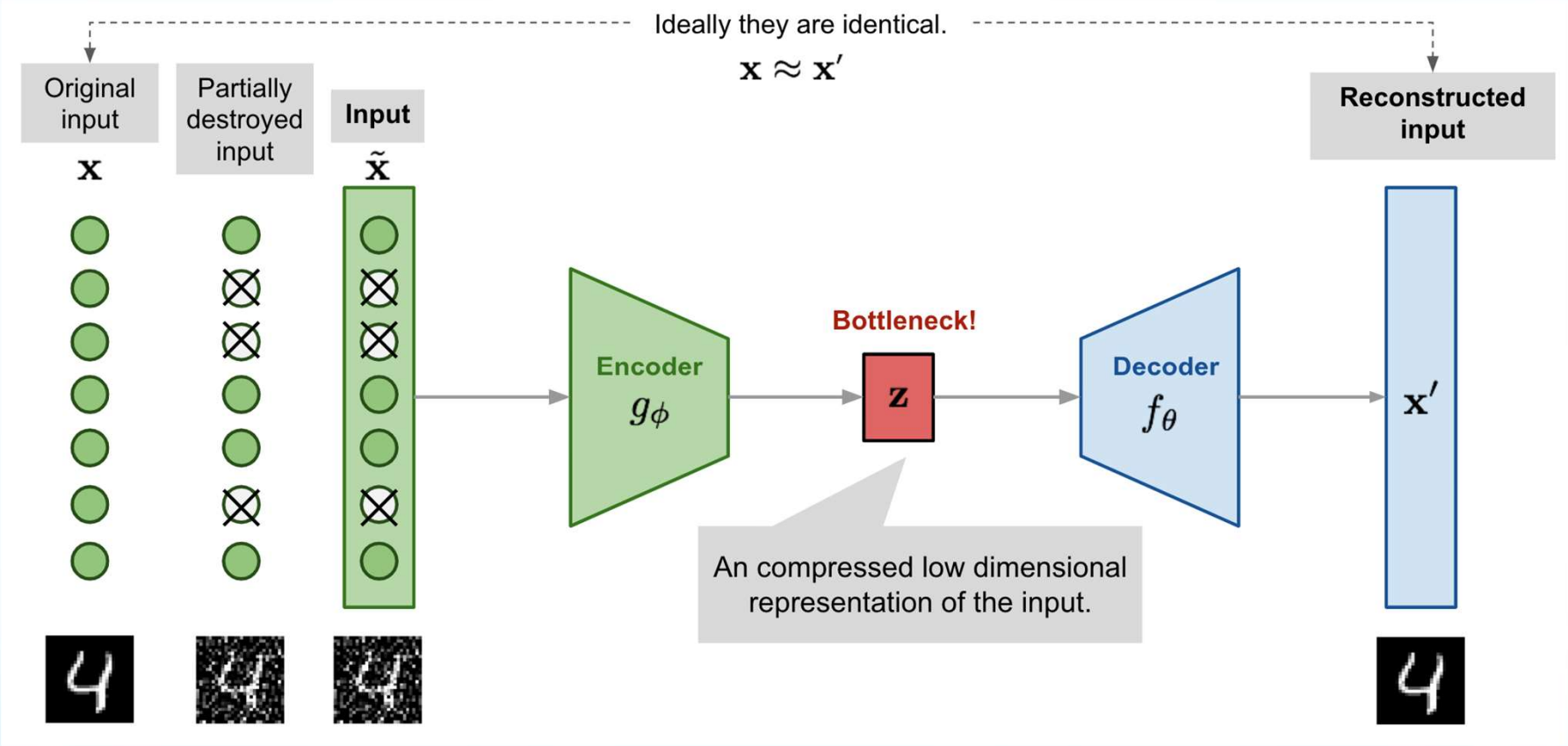
Normal vs Denoising Autoencoder

Denoising encoder-decoder are trained with noisy input to reconstruct the denoised version



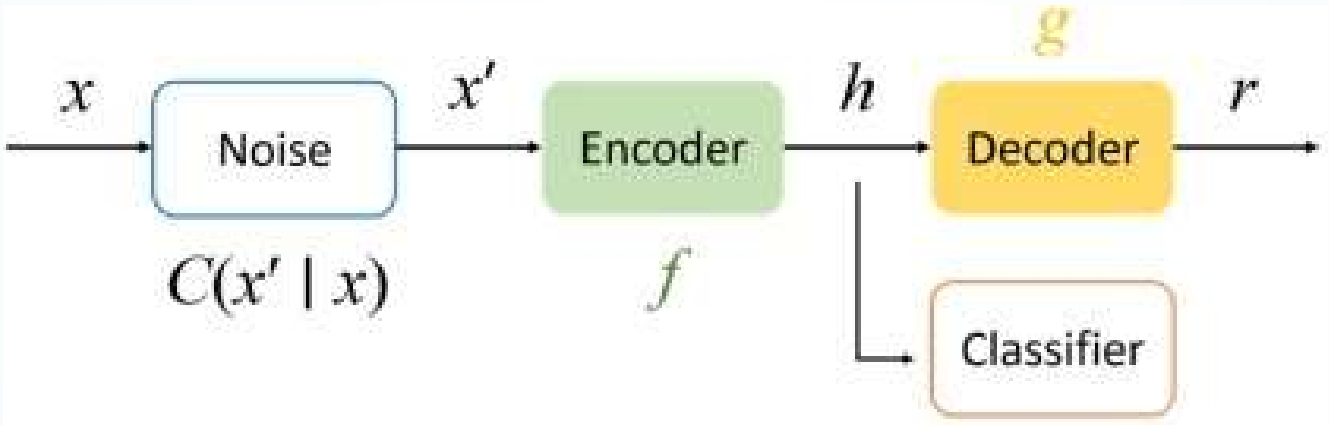
Source: https://blog.keras.io/building-autoencoders-in-keras.html

Denoising Autoencoder



Source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

Denoising Autoencoder



Source: https://www.sciencedirect.com/topics/engineering/denoising-autoencoders

Sparse and Contractive Autoencoders

Sparse autoencoder applies a "sparse" constraint on the hidden unit activation to avoid overfitting and improve robustness.

- It forces the model to only have a small number of hidden units being activated at the same time, or in other words, one hidden neuron should be inactivate most of time.

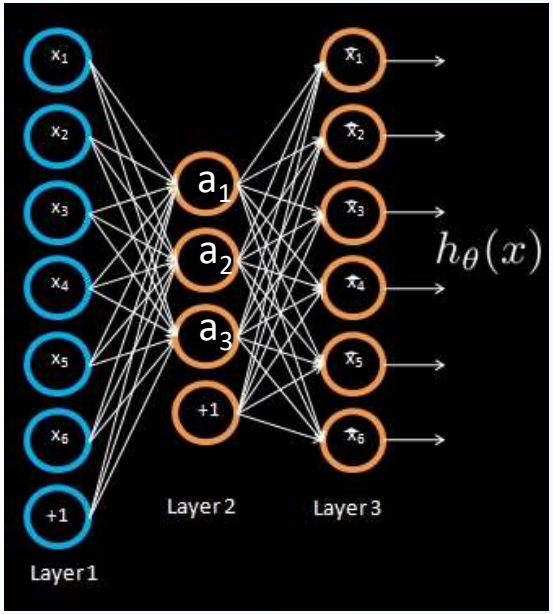
Contractive autoencoder encourages the learned representation to stay in a contractive space for better robustness.

- It adds a term in the loss function to penalize the representation being too sensitive to the input, and thus improve the robustness to small perturbations around the training data points.

Unsupervised feature learning with a Sparse AE

Training a sparse autoencoder.

Given unlabeled training set $x_1, x_2, \ldots$



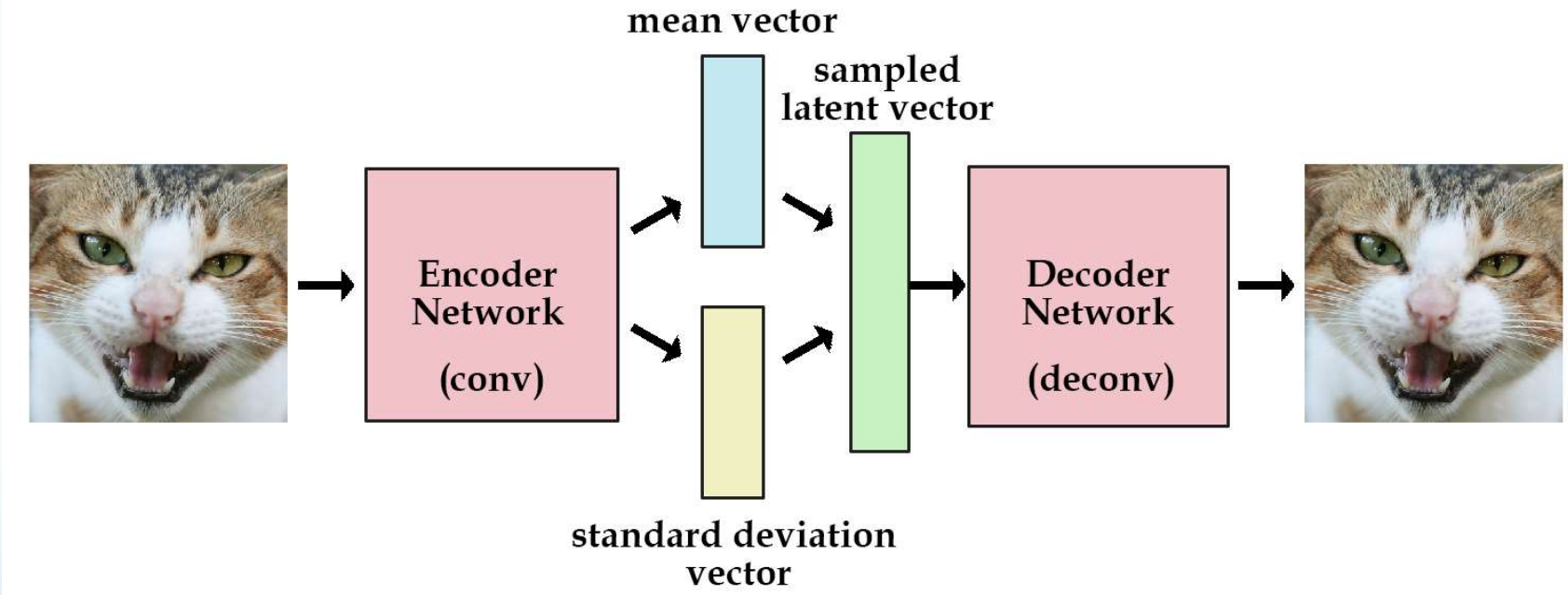
$$\min_\theta \|h_\theta(x) - x\|^2 + \lambda \sum_j |a_j|$$

Reconstruction error term

L_1 sparsity term

Variational Autoencoder

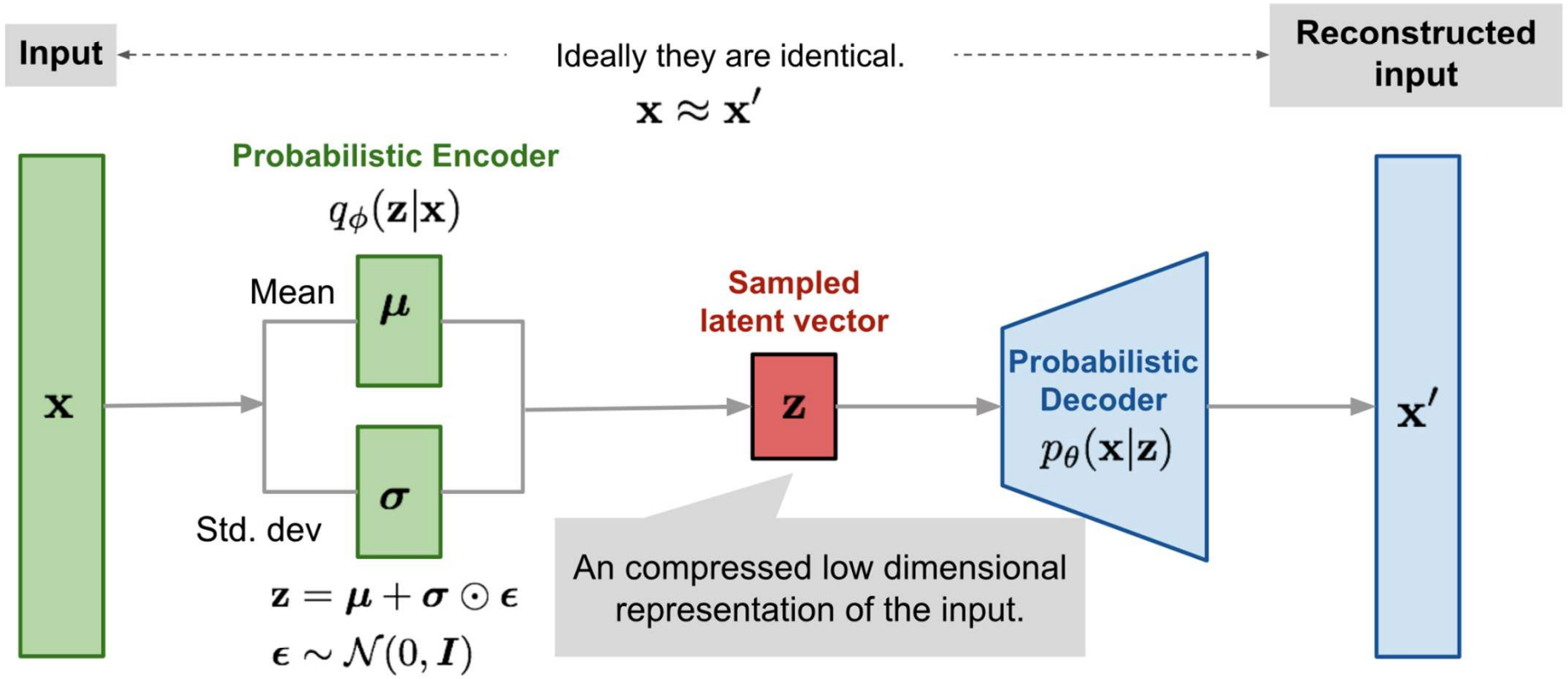
Variational AE constraints the compressed representations to be in a distribution (Gaussian, Mixture of Gaussian etc.)



Further advancements, give the facility to bind certain observable characters with certain latent variables (Disentanglement)

Source: https://www.doc.ic.ac.uk/~js4416/163/website/autoencoders/variational.html

Variational Autoencoder



Source: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

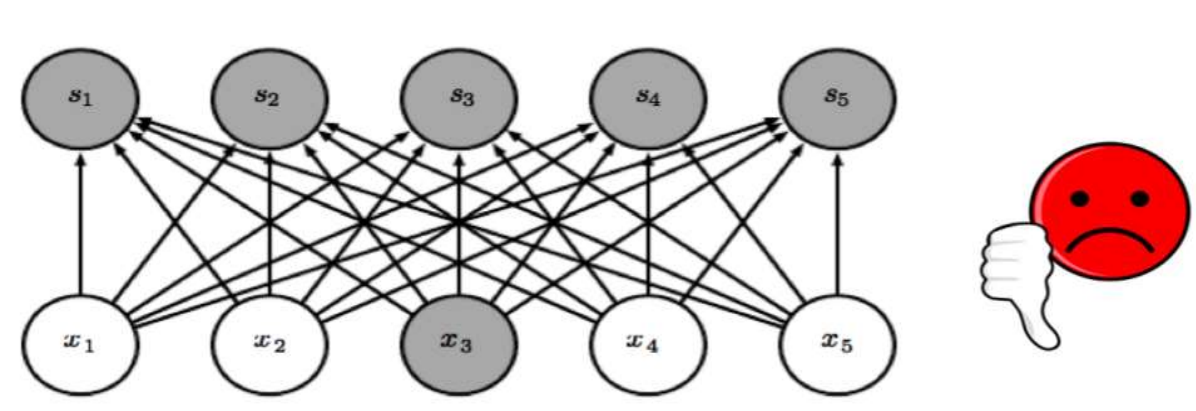
Issues with Fully Connected Neural Network

In fully connected networks:

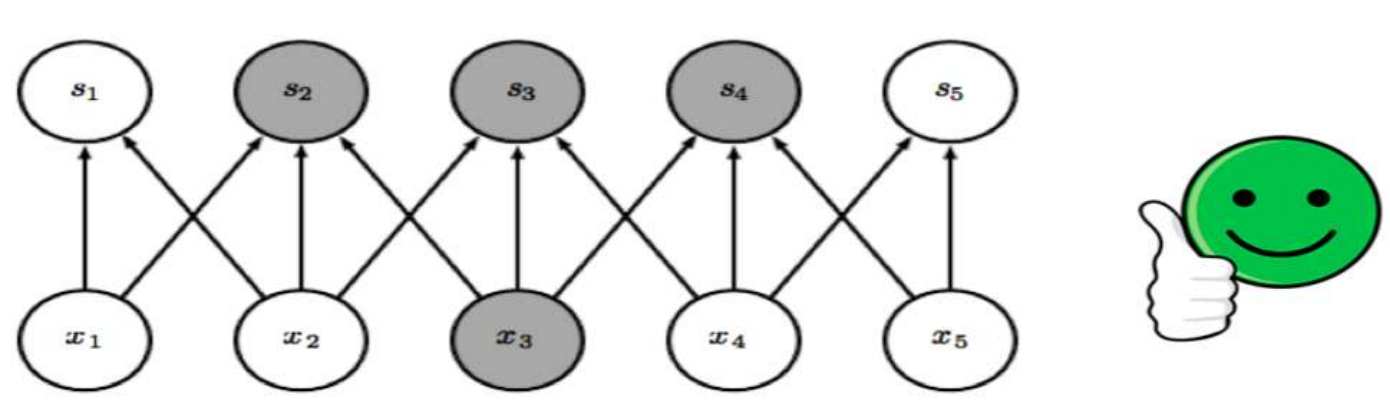
- Number of trainable parameters becomes extremely large
- The topology of the input data is completely ignored
- Little or no invariance to shifting, scaling, and other forms of distortion

Convolutional Neural Networks

Unlike fully connected networks, CNNs use shared weights to reduce the number of parameters



Fully Connected Network



Convolutional network

Source: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

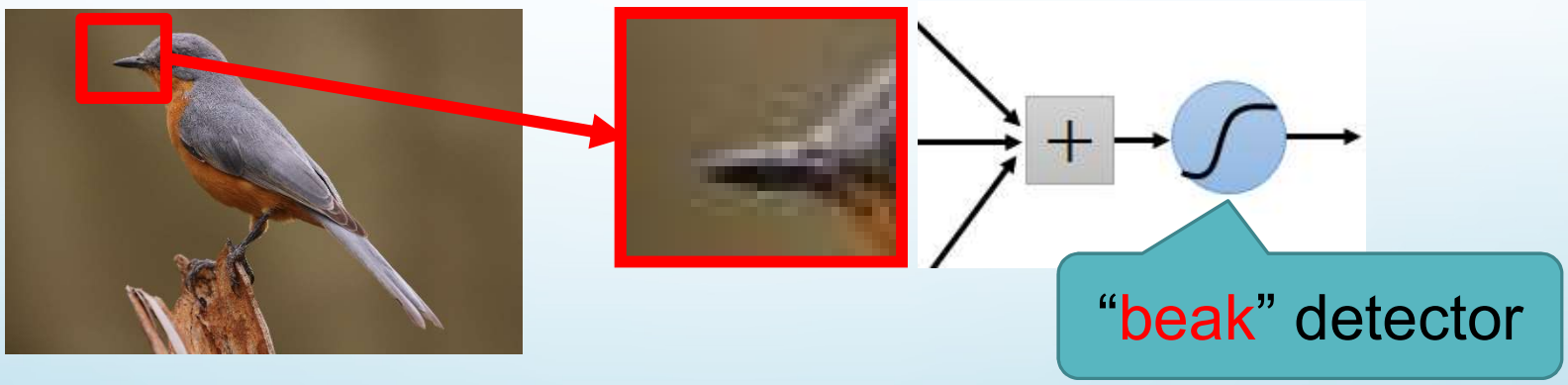
Convolutional neural networks

Some patterns in the image are much smaller than the whole image and appears in different places

CNNs represent a small region with fewer parameters

Filters (shared weights) are employed for learning the patterns.

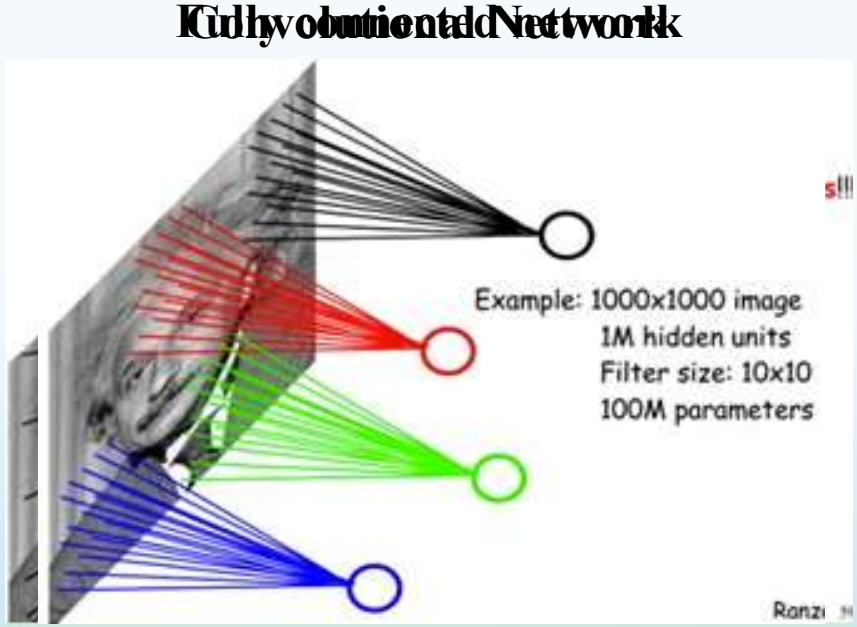
A convolutional layer has a number of filters that does convolutional operation.



Source: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

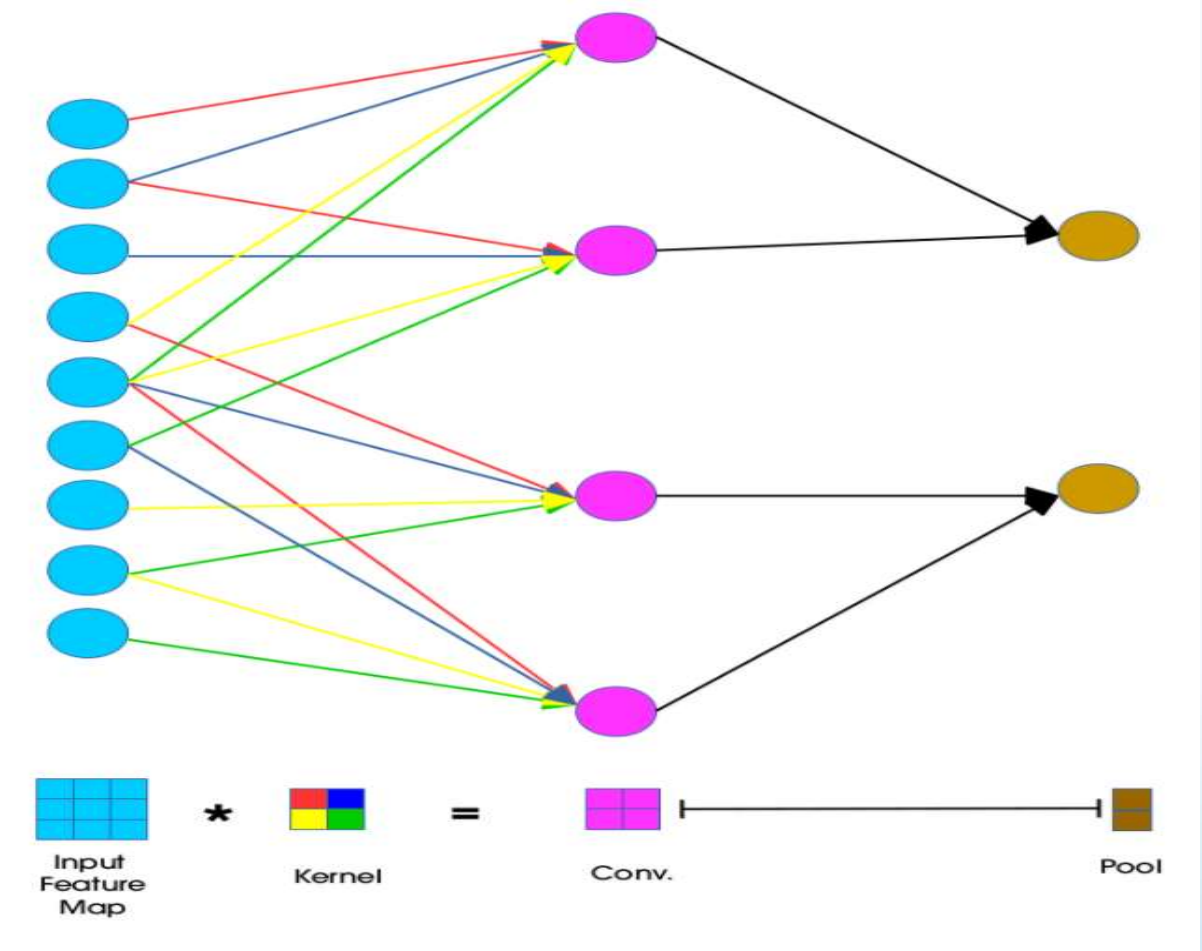
Convolutional Neural Networks (CNNs)

- Learned shared weights simulate convolutional filters or kernels
- Convolutional layer has a number of filters that does convolutional operation.

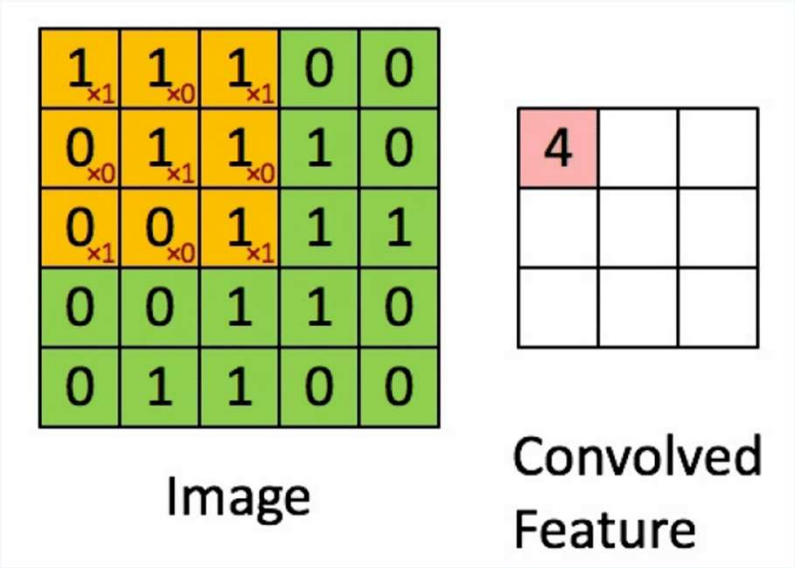


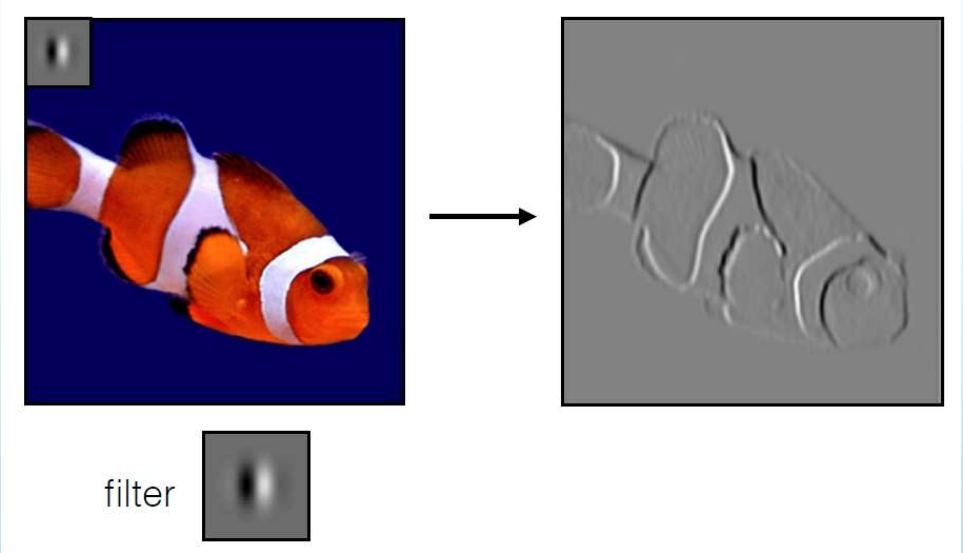
Source: https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

Convolutional Neural Networks (CNNs)

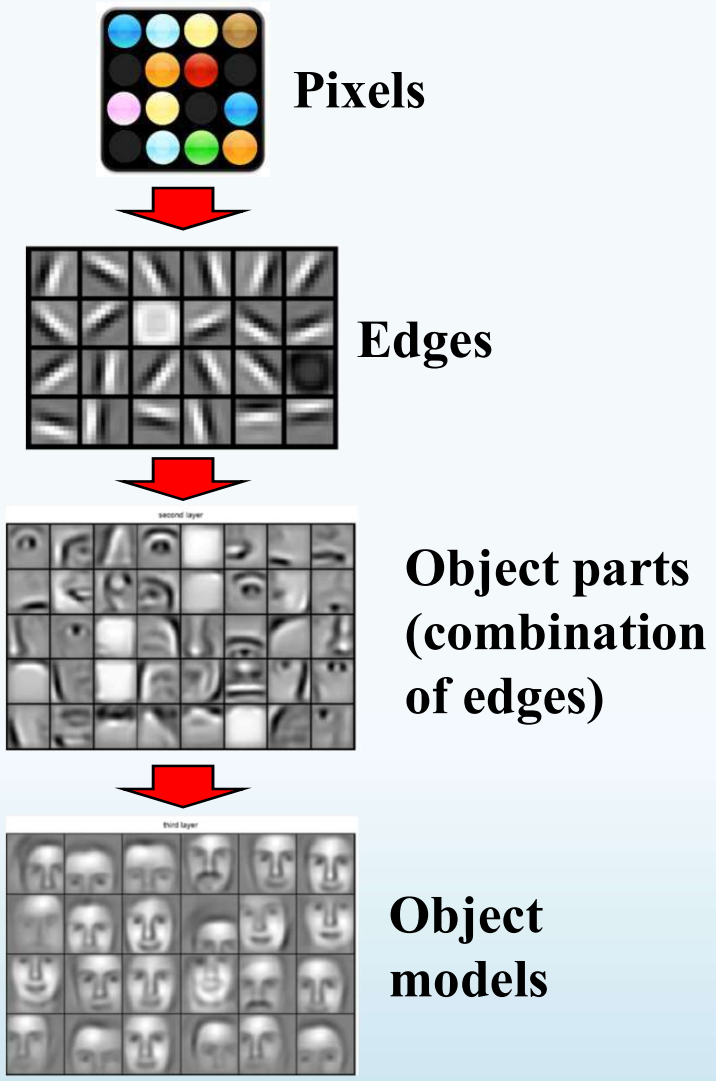


Convolutional Neural Networks (CNNs)





Feature Learning in Convolution Neural Networks



Source: presentations of Andrew NG

Convolutional Layer

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

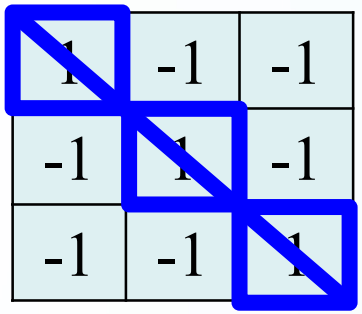
Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

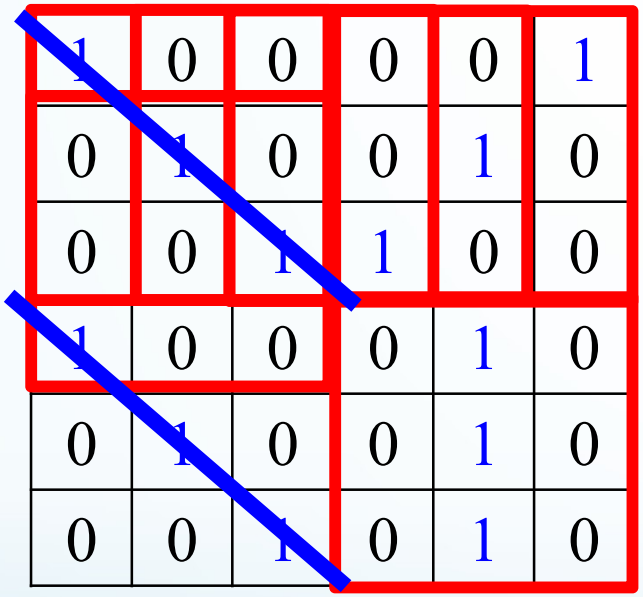
Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolution

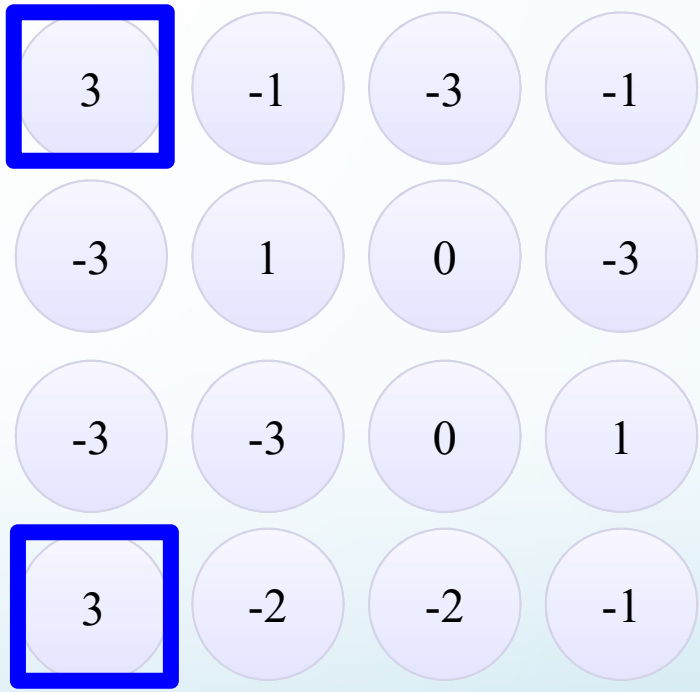


Filter 1

stride=1



6 x 6 image



Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

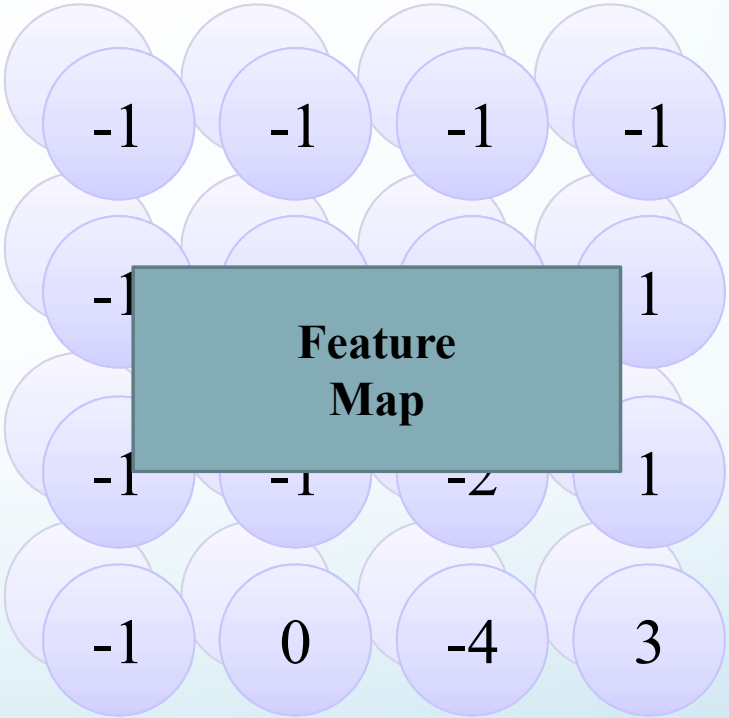
Filter 2

Stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

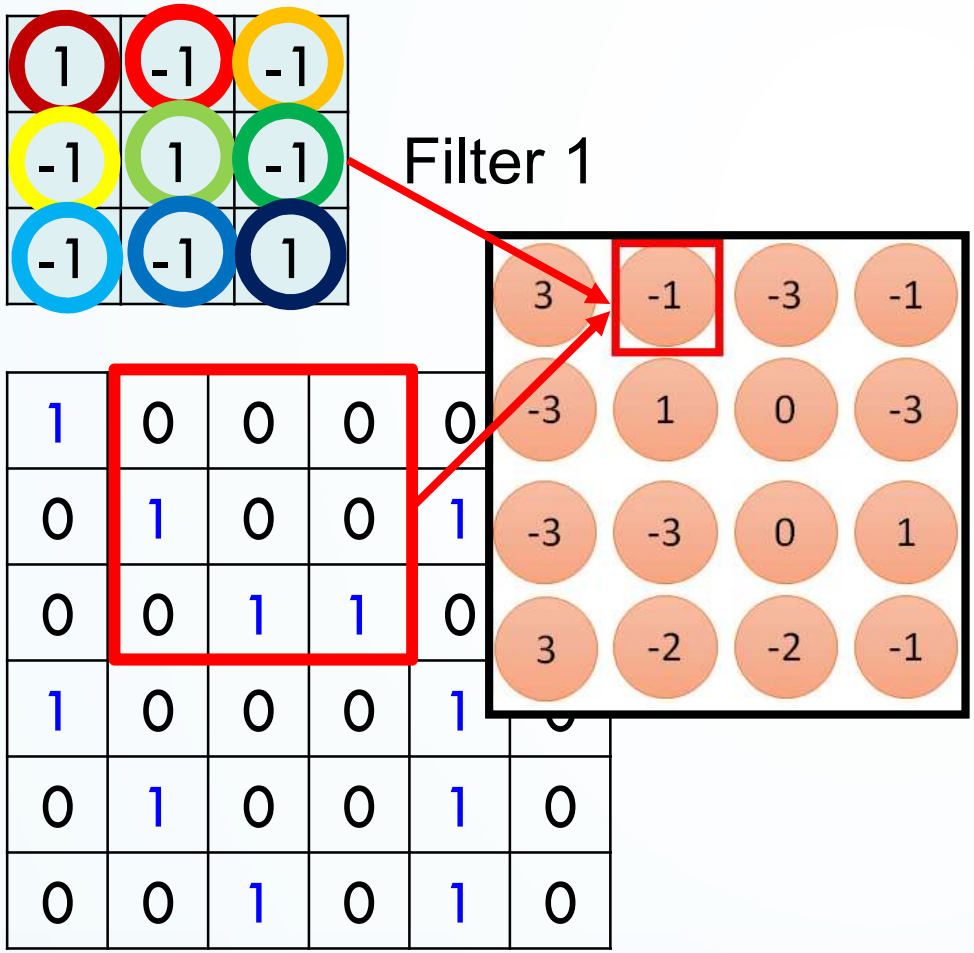
6 x 6 image

Repeat this for each filter

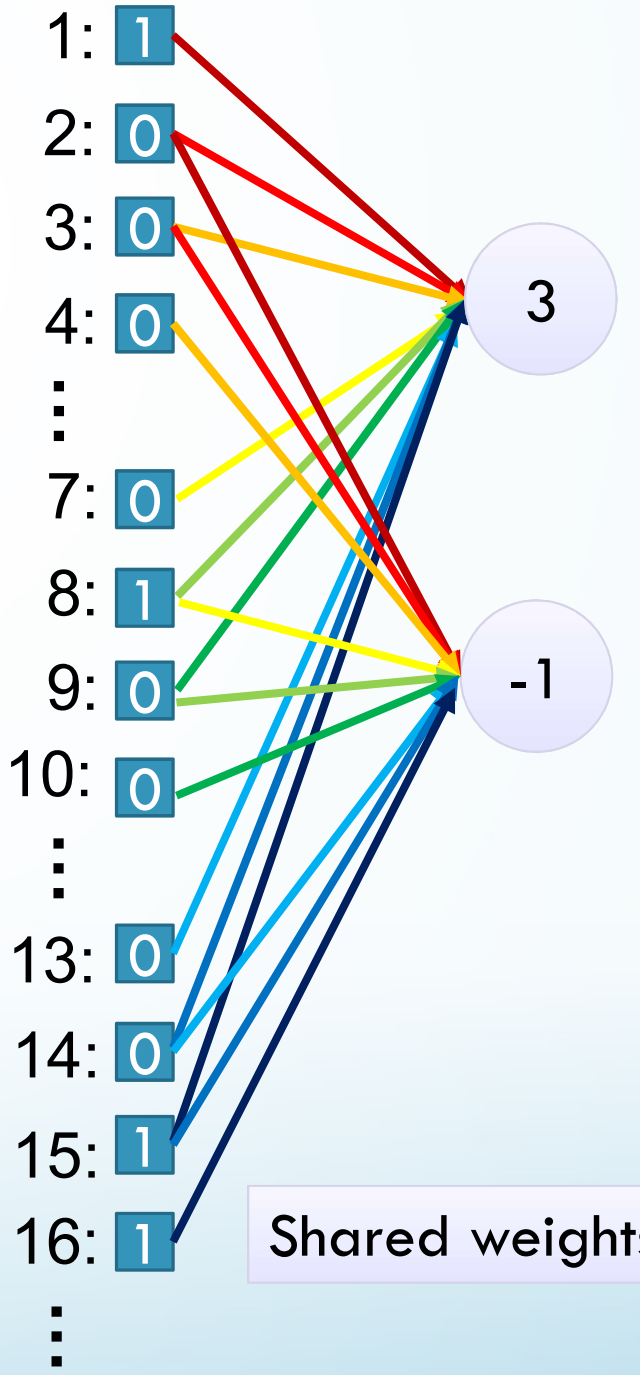


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/



Fewer parameters

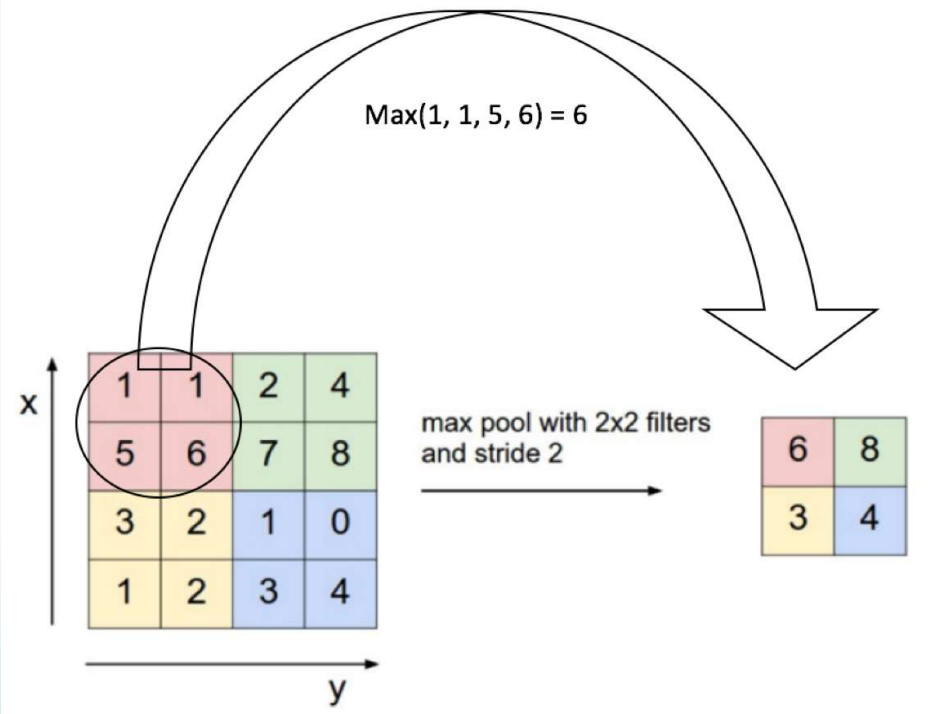


Shared weights

Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Pooling Layer

Pooling layers combine the outputs of neuron clusters at one layer into a single neuron in the next layer.



Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Why Pooling?

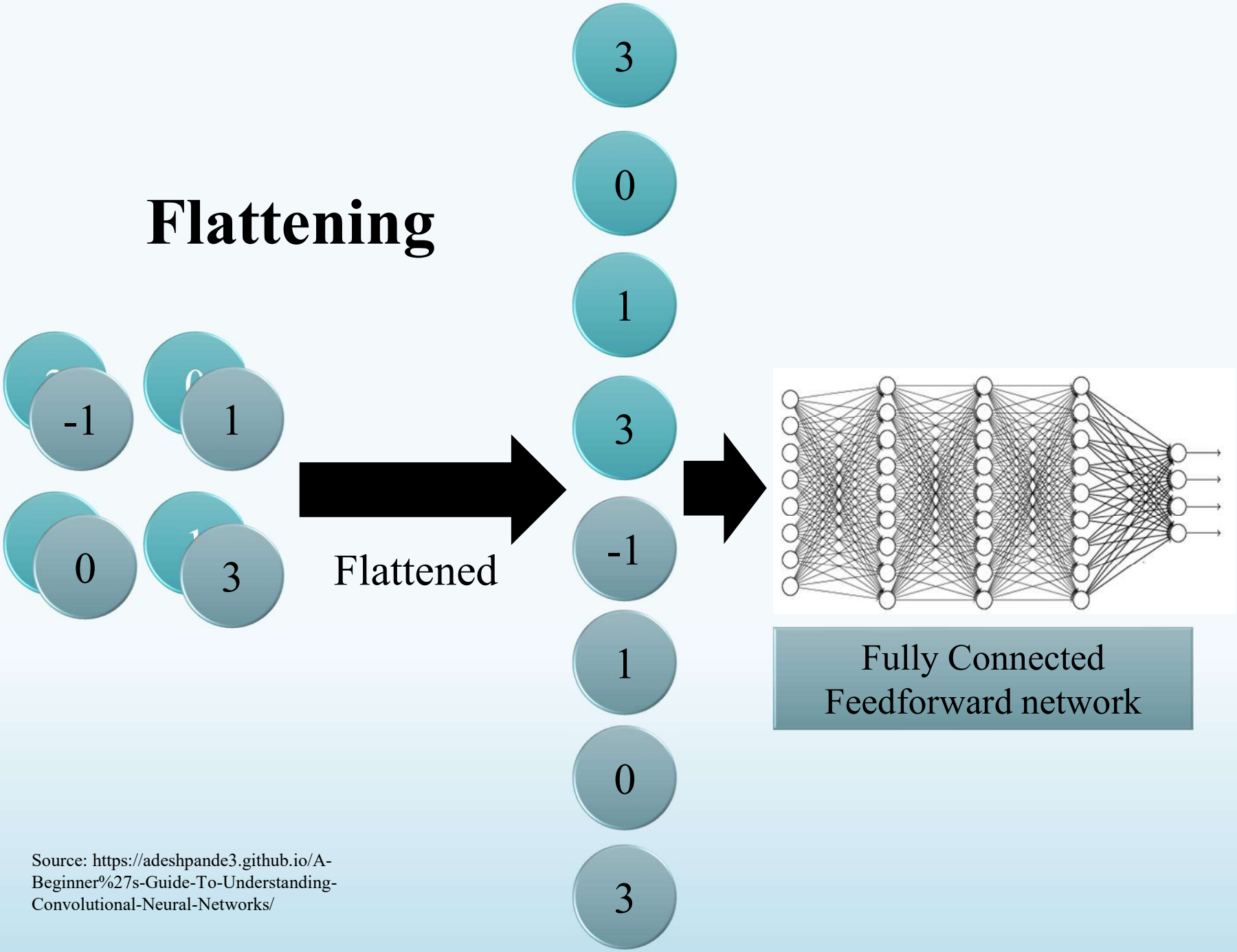
Transform the joint feature representation into a form that preserves important information while discarding irrelevant detail.

Achieving invariance to changes in position or lighting conditions, robustness to clutter, and compactness of representation, are all common goals of pooling.

Popular pooling strategies

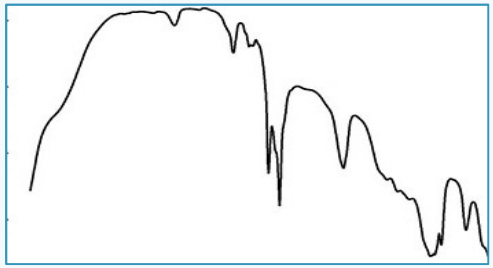
- Max-pooling: take the maximum value in each block
- Average-pooling: average all values in each block

Flattening



Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

The Whole CNN



Convolution

Features

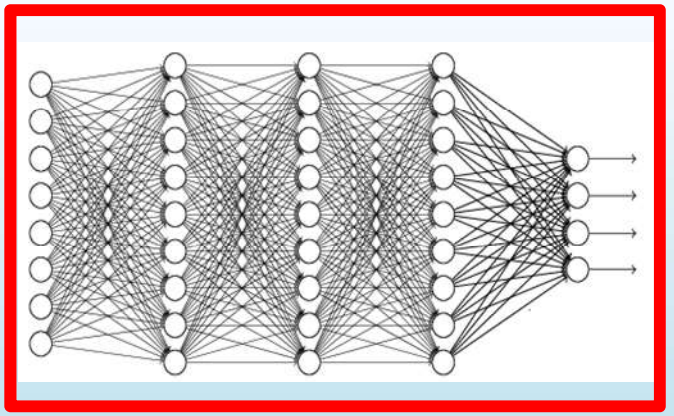
Max Pooling

Convolution

Max Pooling

Flattened

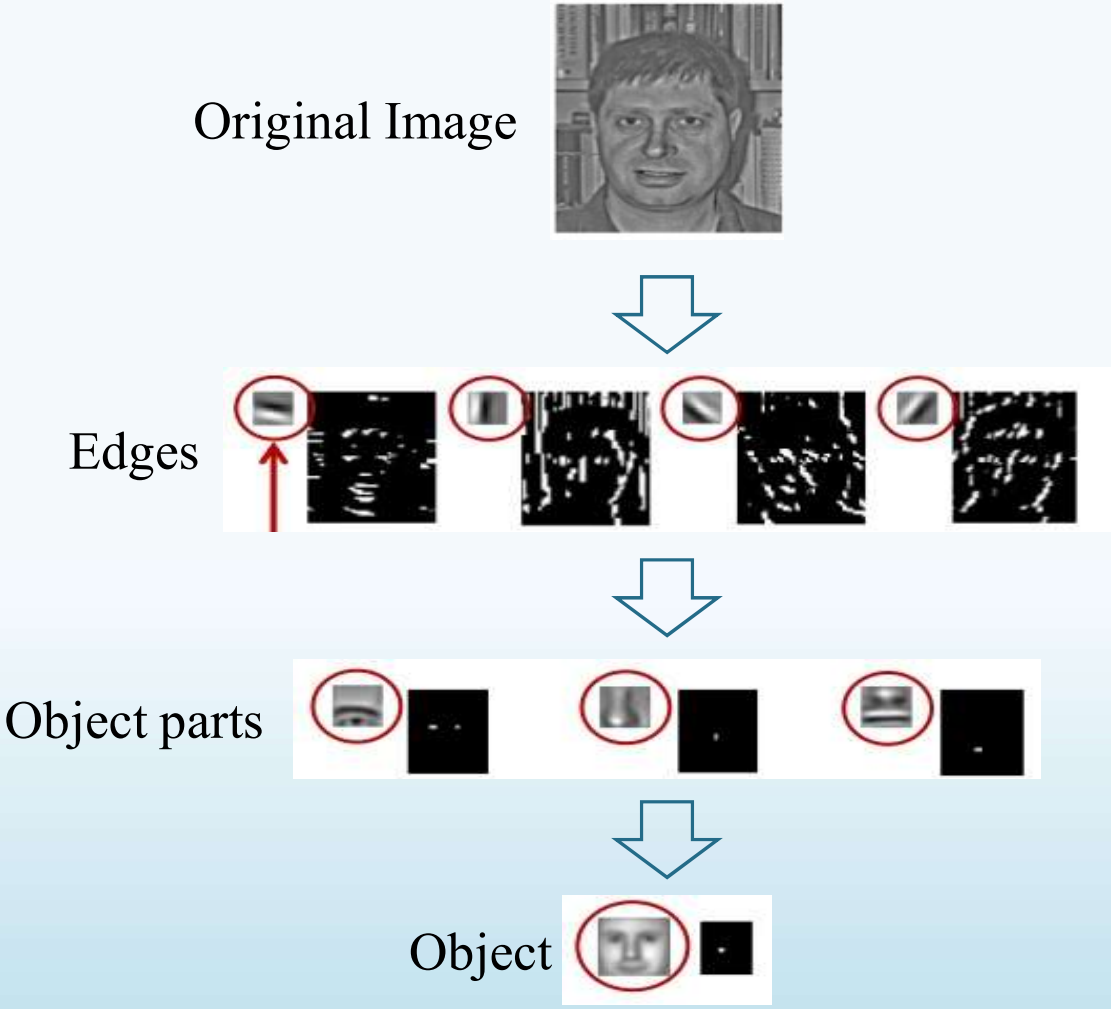
Fully connected network



wheat rice…

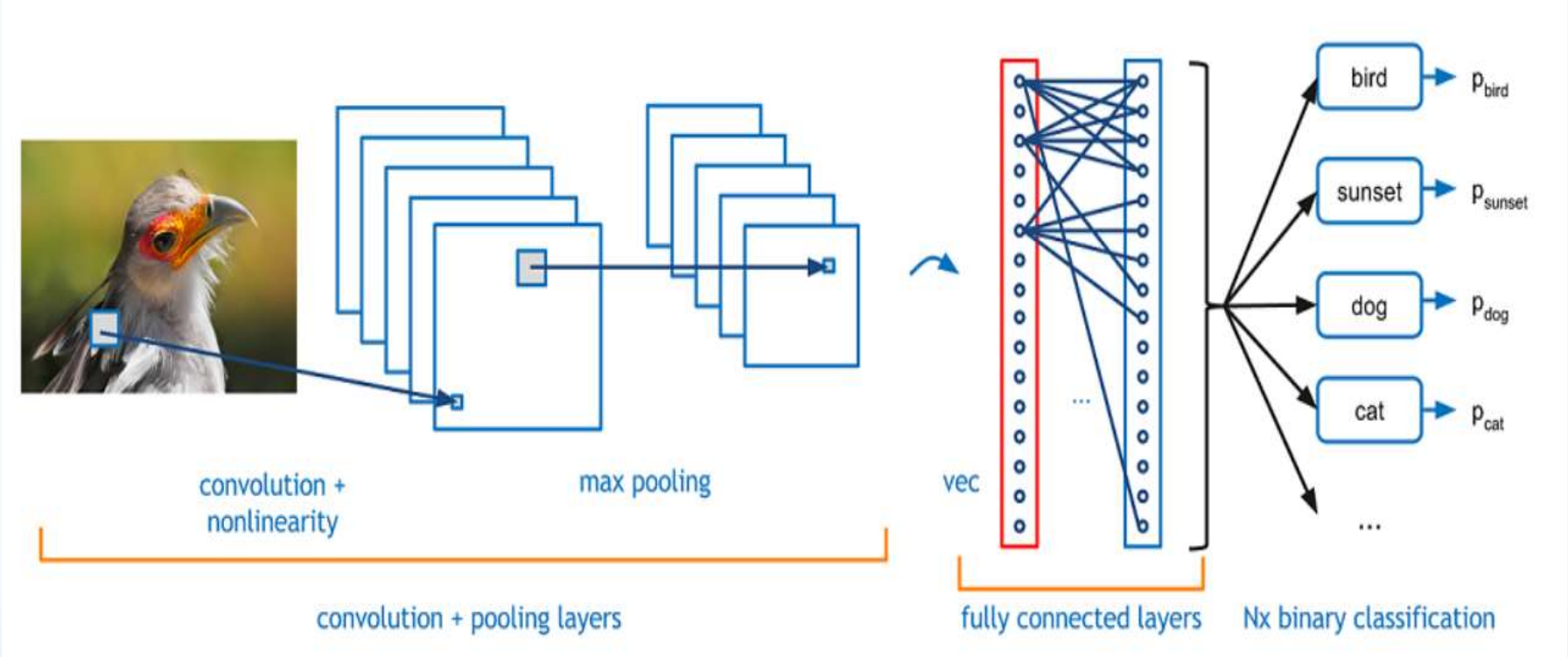
CNN Learned Feature Space

Deep feature space representations can model the semantics in the images



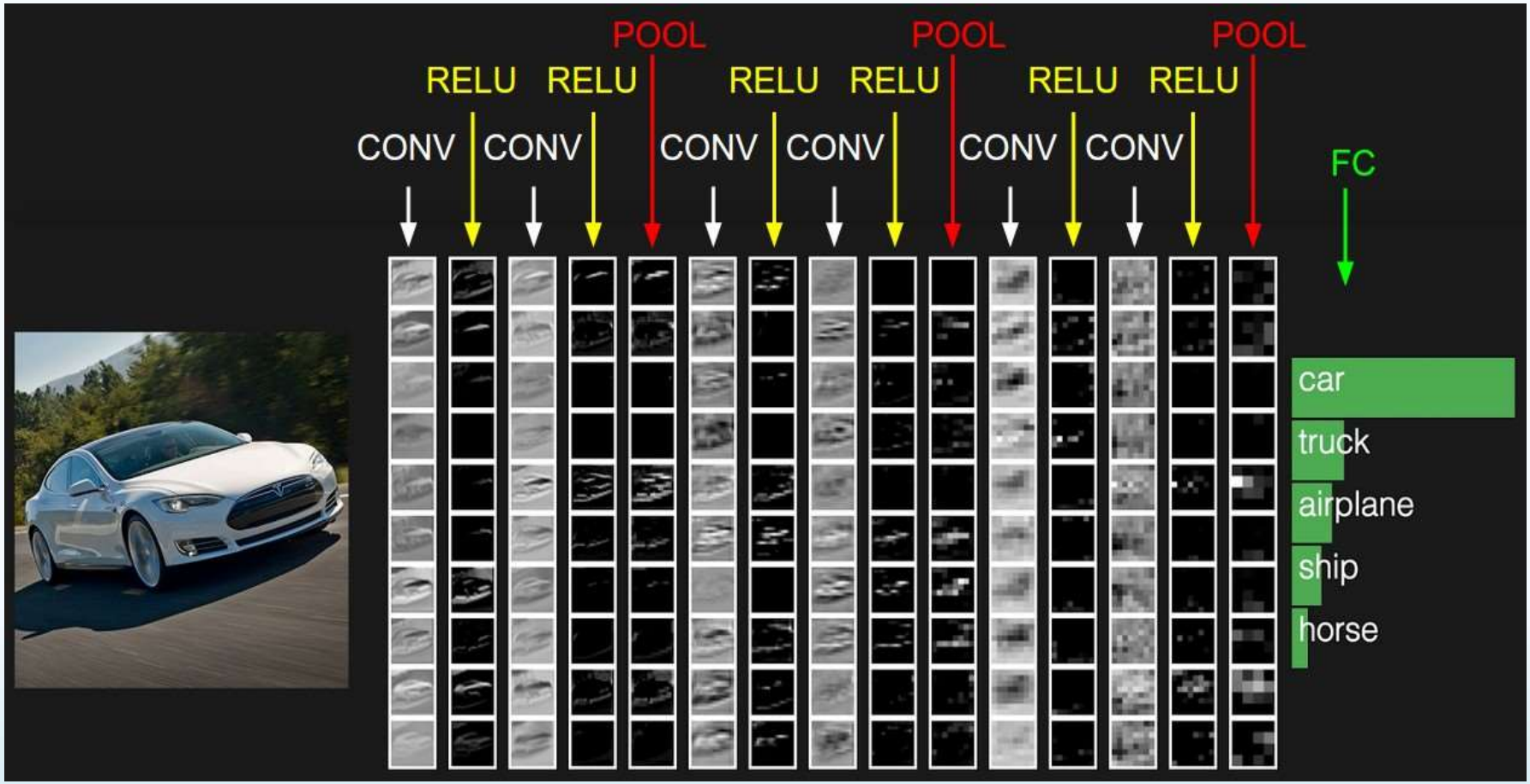
Source: Ahmed et al., 2018

CNN for Scene Recognition



Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

CNN for Scene Recognition



Source: http://cs231n.github.io/convolutional-networks/

CNN for Scene Recognition

Source: http://cs231n.github.io/convolutional-networks/

Advantages of Convolution Neural Networks for Remote Sensing Image Analysis

Deep convolution networks are the best available choice for modelling Earth observation processes due to:

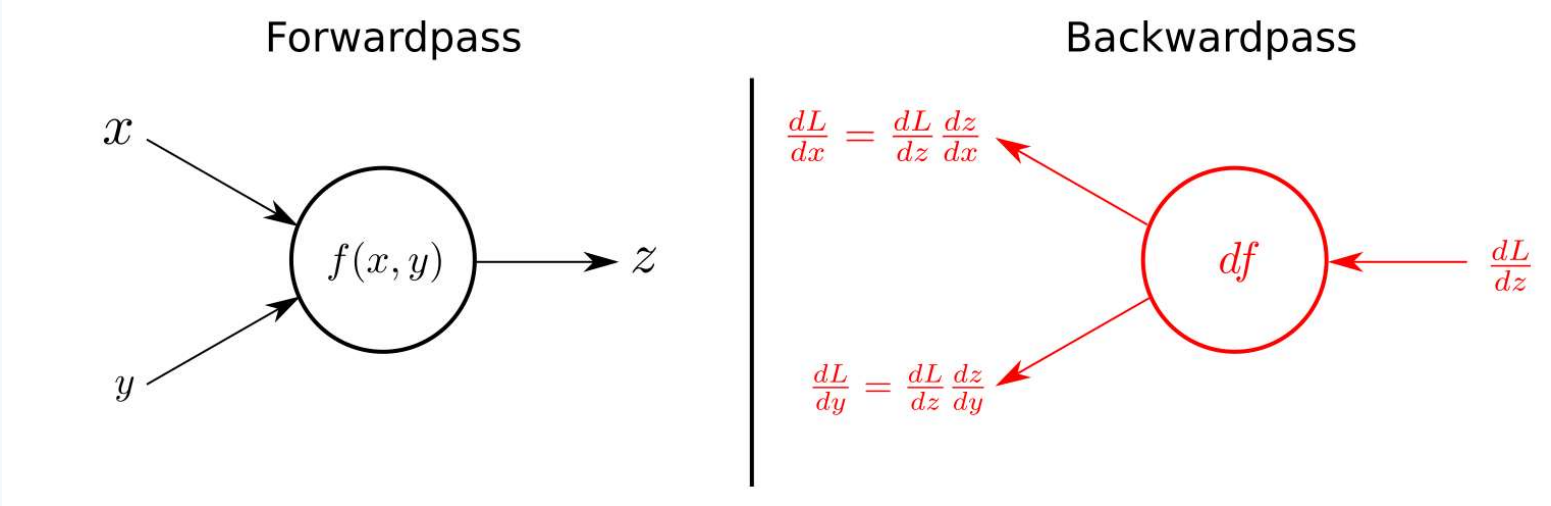
Capability in modelling the spatial context

Ability to learn hierarchical features

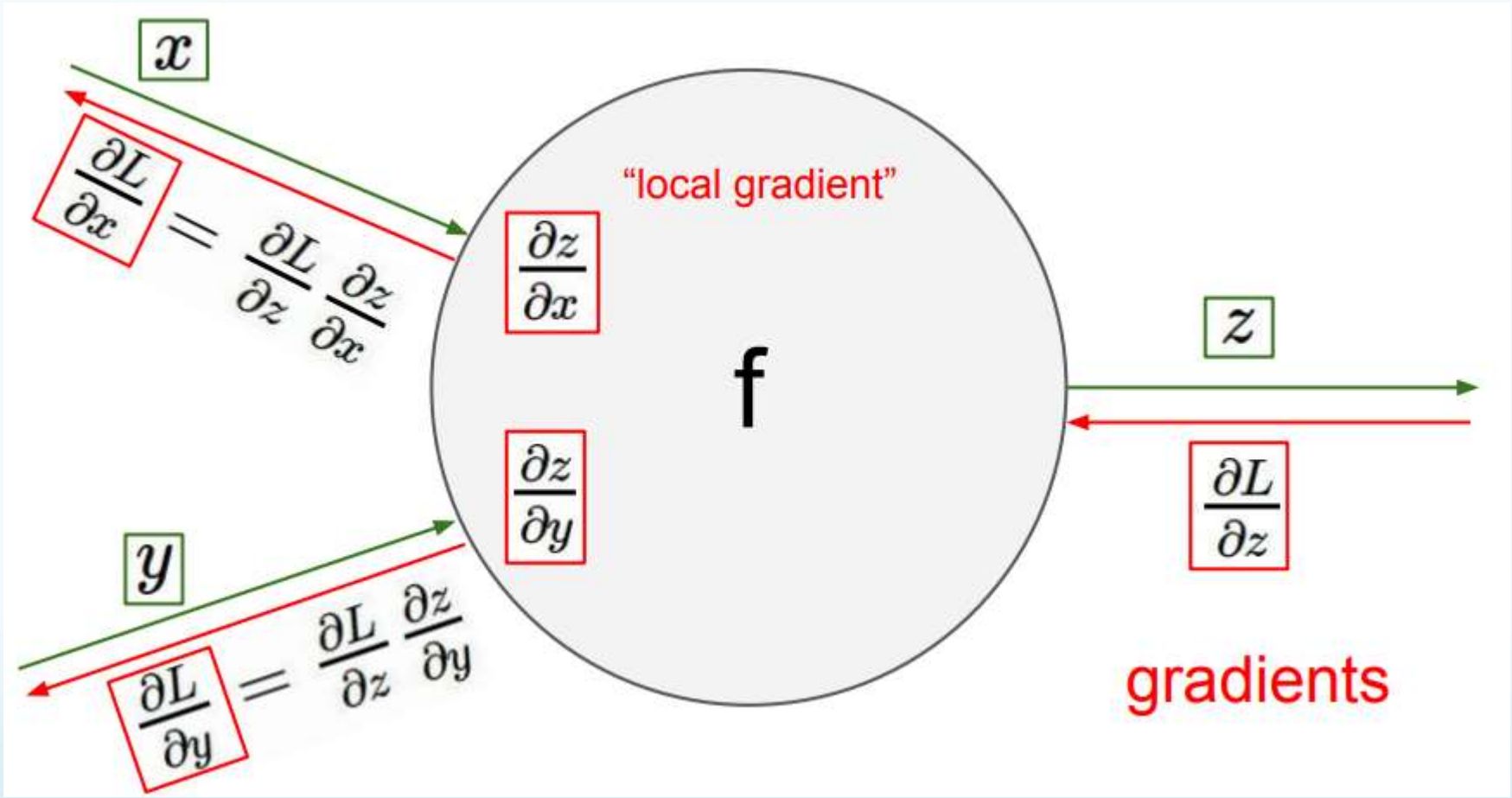
Less number of hyper parameters

Transfer learning: enable the use of pre-trained networks

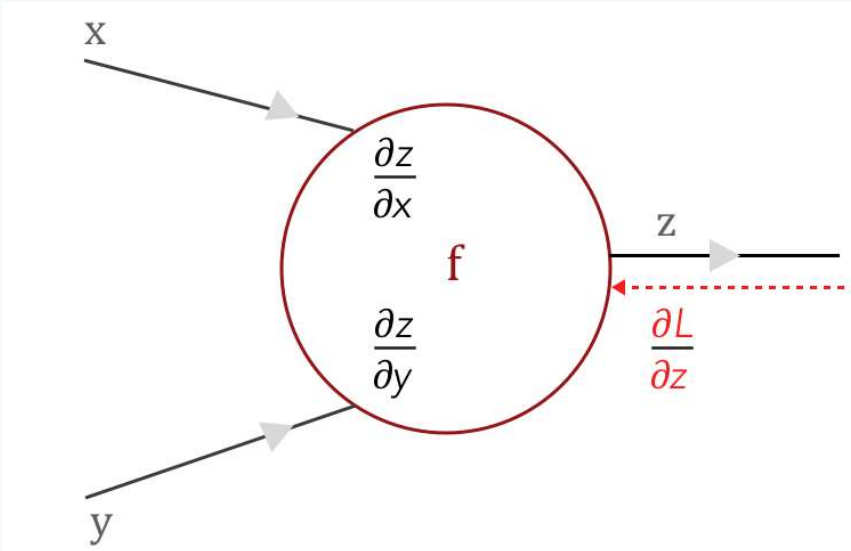
Back Propagation in CNN



Back Propagation in CNN

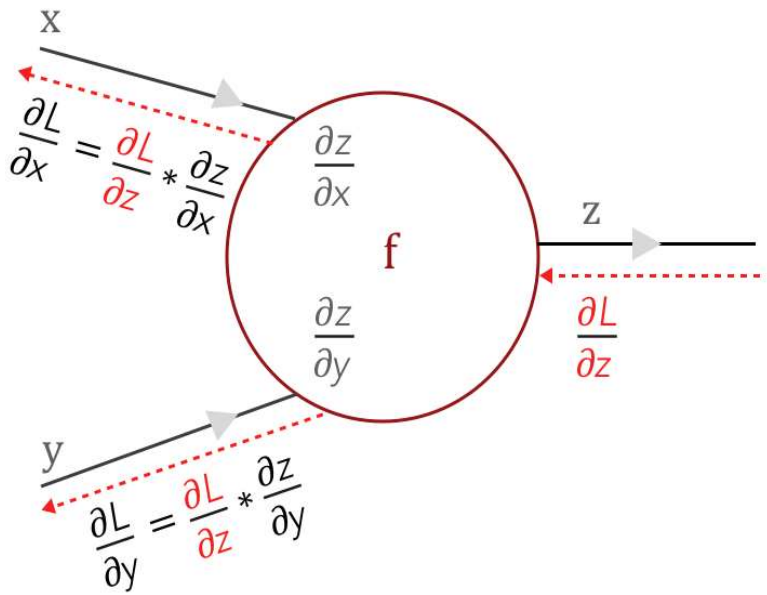


Back Propagation in CNN



$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ *are local gradients*

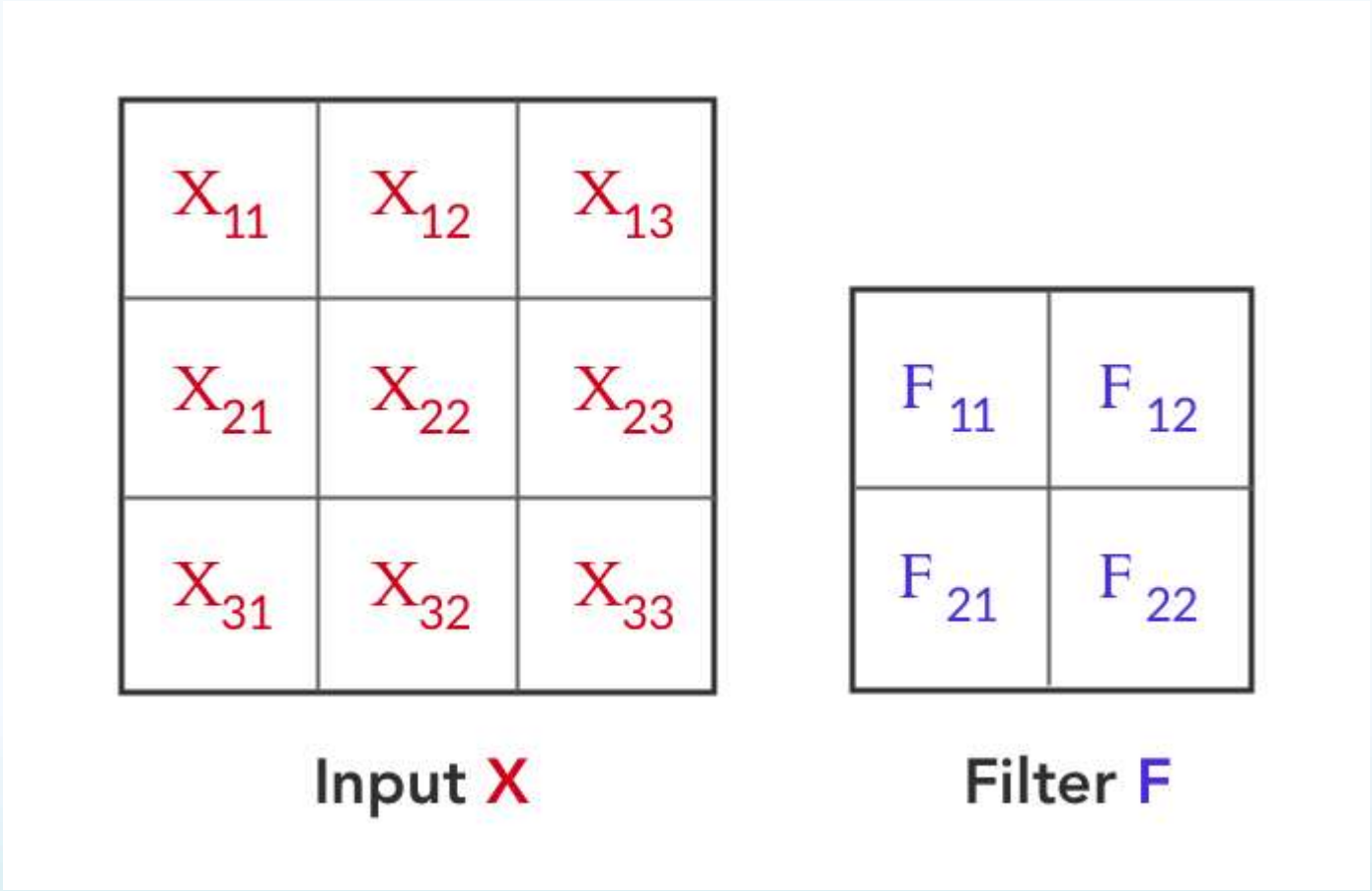
$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*



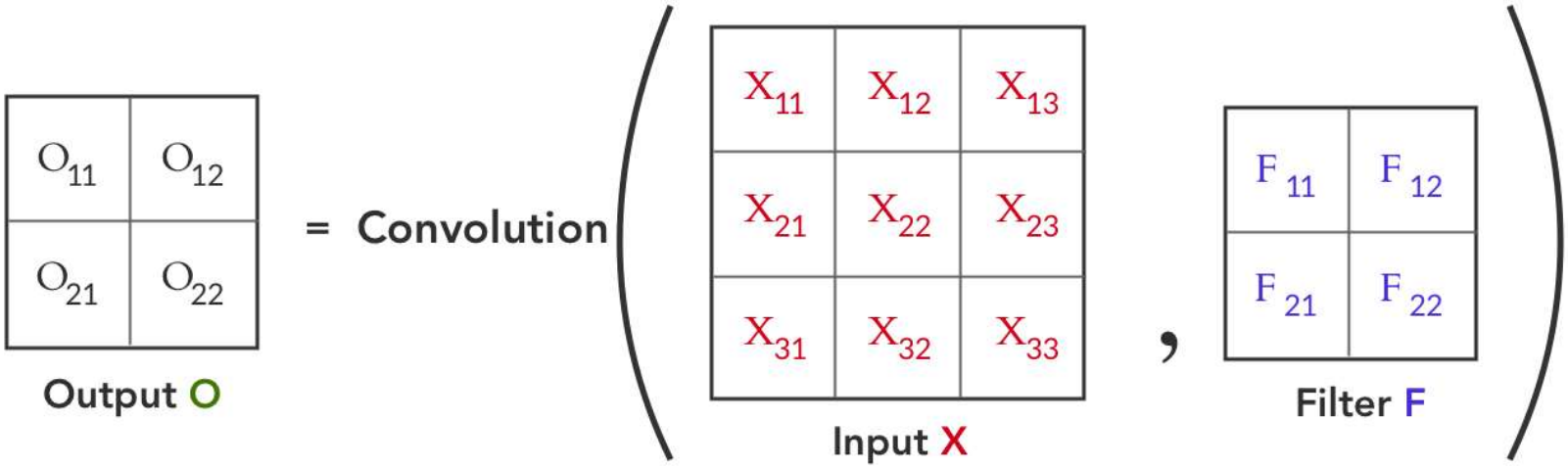
$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

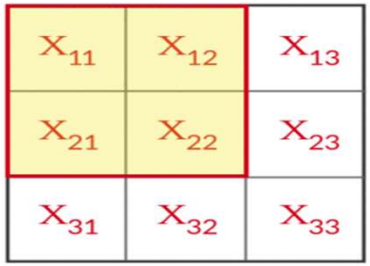
Back Propagation in CNN



Back Propagation in CNN



Back Propagation in CNN



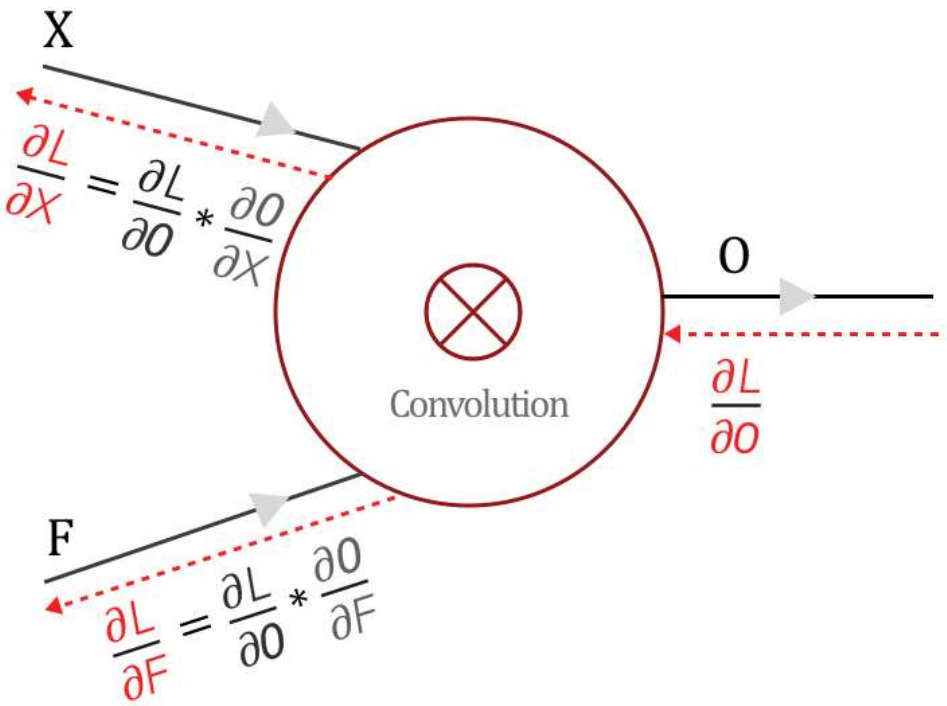
Input X

Filter F

$X_{11}F_{11}$	$X_{12}F_{12}$	X_{13}
$X_{21}F_{21}$	$X_{22}F_{22}$	X_{23}
X_{31}	X_{32}	X_{33}

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

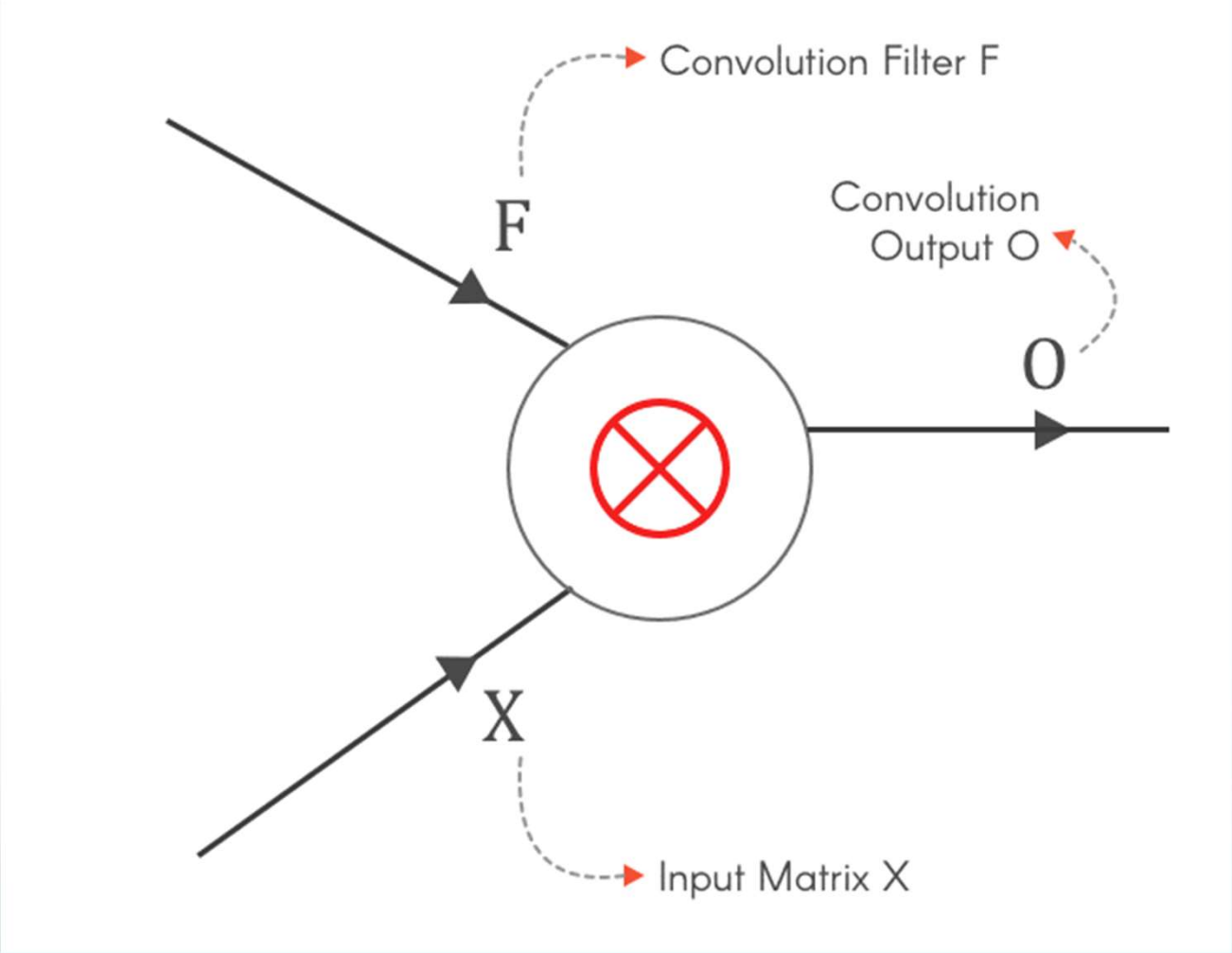
Back Propagation in CNN



$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

Back Propagation in CNN



Back Propagation in CNN

*Step 1: Finding the local gradient — **∂O/∂F:***

This means we have to differentiate Output Matrix O with Filter F. From our convolution operation, we know the values. So let us start differentiating the first element of O- O^{11} with respect to the elements of F — F^{11} , F^{12}, F^{21} and F^{22}

Local Gradients → (A)

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Finding derivatives with respect to F_{11} , F_{12} , F_{21} *and* F_{22}

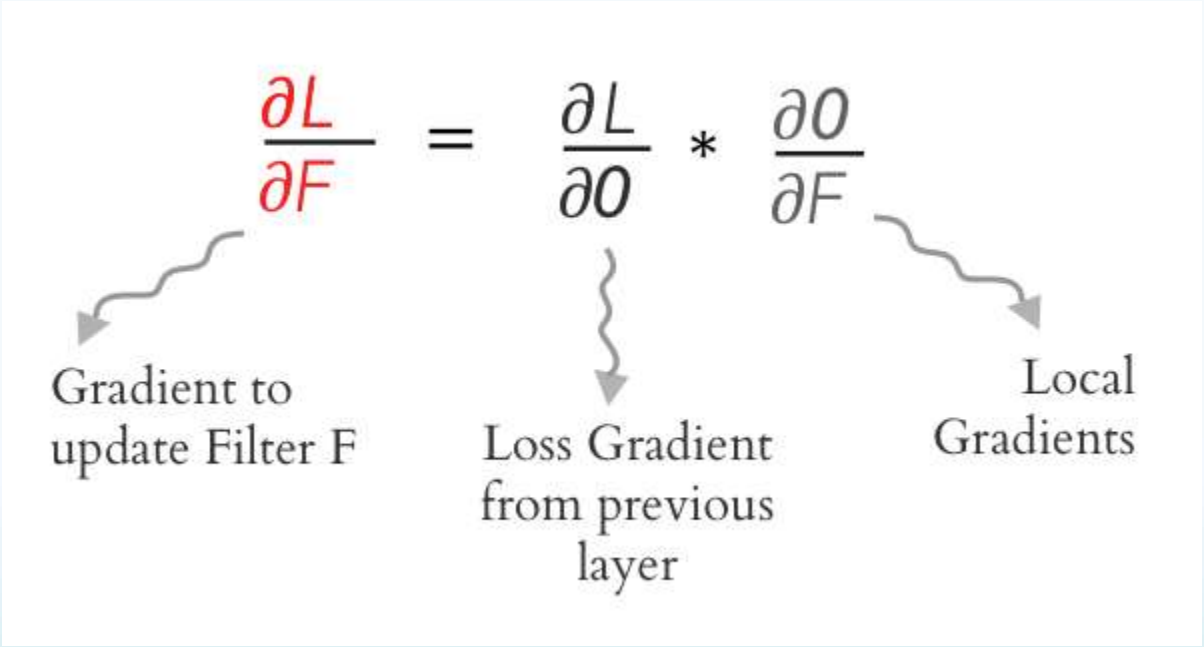
$$\frac{\partial O_{11}}{\partial F_{11}} = X_{11} \quad \frac{\partial O_{11}}{\partial F_{12}} = X_{12} \quad \frac{\partial O_{11}}{\partial F_{21}} = X_{21} \quad \frac{\partial O_{11}}{\partial F_{22}} = X_{22}$$

Similarly, we can find the local gradients for O_{12} , O_{21} *and* O_{22}

Back Propagation in CNN

Step 2: Using the Chain rule:

As described in our previous examples, we need to find **∂L/∂F** as:



O and **F** are matrices. And **∂O/∂F** will be a partial derivative of a matrix **O** with respect to a matrix **F**! On top of it we have to use the chain rule. This does look complicated but thankfully we can use the formula below to expand it.

Back Propagation in CNN

For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

Expanding, we get..

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}}$$

Back Propagation in CNN

Substituting the values of the local gradient — **∂O/∂F from Equation A**, we get

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

If you closely look at it, this represents an operation we are quite familiar with. We can represent it as a **convolution operation between input X** and loss gradient **∂L/∂O as shown below:**

Back Propagation in CNN

Substituting the values of the local gradient — **∂O/∂F from Equation A**, we get

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

If you closely look at it, this represents an operation we are quite familiar with. We can represent it as a **convolution operation between input X** and loss gradient **∂L/∂O as shown below:**

Back Propagation in CNN

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X} \qquad \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O}$$

Loss gradient from previous layer

∂L/∂F is nothing but the convolution between Input X and Loss Gradient from the next layer ∂L/∂O

Back Propagation in CNN

Finding ∂L/∂X:

*Step 1: Finding the local gradient — **∂O/∂X:***

Similar to how we found the local gradients earlier, we can find **∂O/∂X** as:

Local Gradients: → (B)

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Differentiating with respect to X_{11}, X_{12}, X_{21} *and* X_{22}

$$\frac{\partial O_{11}}{\partial X_{11}} = F_{11} \quad \frac{\partial O_{11}}{\partial X_{12}} = F_{12} \quad \frac{\partial O_{11}}{\partial X_{21}} = F_{21} \quad \frac{\partial O_{11}}{\partial X_{22}} = F_{22}$$

Similarly, we can find local gradients for O_{12}, O_{21} *and* O_{22}

Back Propagation in CNN

Step 2: Using the Chain rule:

For every element of X_i

$$\frac{\partial L}{\partial X_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial X_i}$$

Expanding this and substituting from Equation B, we get

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

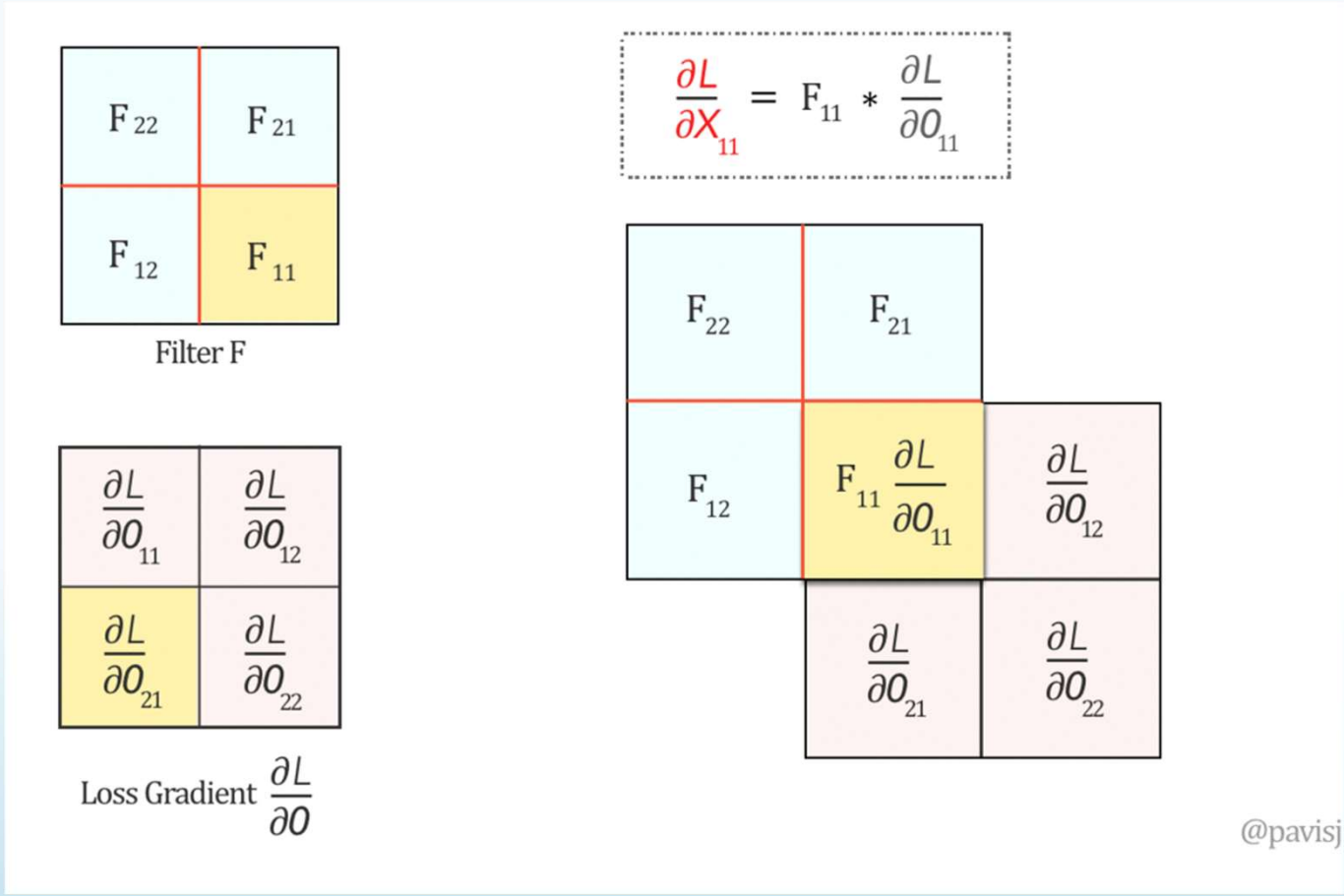
$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$

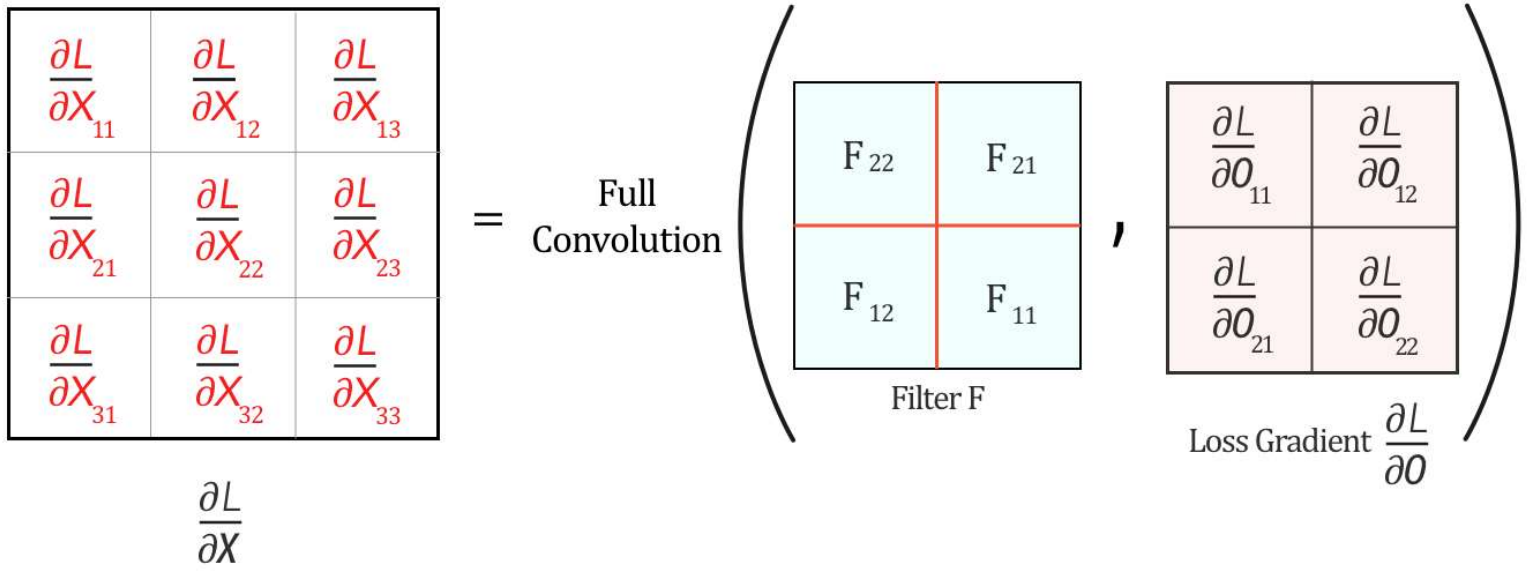
Back Propagation in CNN

∂L/∂X can be represented as 'full' convolution between a 180-degree rotated Filter F and loss gradient ∂L/∂O



Back Propagation in CNN

The full convolution above generates the values of $\partial L/\partial X$ and hence we can represent $\partial L/\partial X$ as



Back Propagation in CNN

Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X\,,\ \text{Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^{\circ}\text{rotated Filter } F\,,\ \text{Loss Gradient } \frac{\partial L}{\partial O}\right)$$

Brief review of the existing hyperspectral classifiers

- Prominent existing approaches:
  - Markov random field based methods (Ghamisi Et Al., 2014; Jia and Richards, 2008)
  - Morphological profile based methods (Dalla et al., 2010; Ghamisi et al., 2016a; Ghamisi et al., 2016b; Fang et al., 2018a; Fang et al., 2018b)
  - Compressive sensing based methods (Wei et al., 2017; Chen et al., 2011; Tang et al., 2014; Fang et al., 2017; Fu et al., 2018; Gan et al., 2018; Su et al., 2018)
  - CNN based methods (Krizhevsky et al., 2017; Rawat and Wang, 2017; Hu et al., 2015; Makantasis et al., 2016; Zhao, and Du, 2016; Gao et al., 2018)

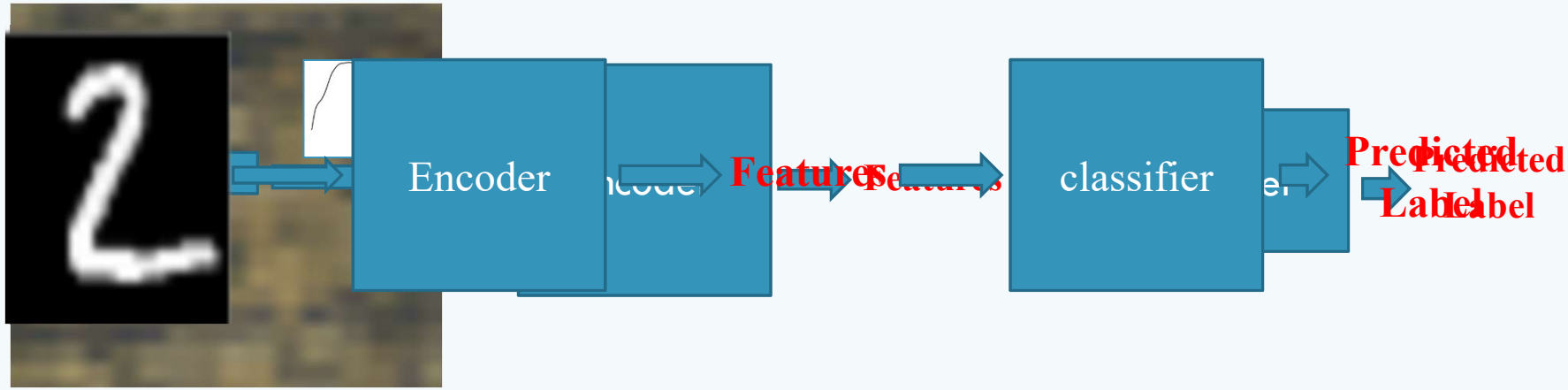
- CNN based approaches yield better results than the conventional approaches:
  - Model spectral-spatial features to address huge intra-class-signature variability
  - Facilitates simultaneous optimization of feature extraction and classification stages
  - LSTM based approach (Mou et al., 2017) considers sequential structure of the hyperspectral pixel spectra

Limitations of the existing hyperspectral image classification approaches

- Most of the existing hyperspectral classifiers
  - Consider pixel-spectra as vectors and ignore the spectral features crucial in distinguishing the spectra.
  - Do not consider information regarding the depth, width, and position of spectral features

- Integration of spectral-spatial feature learning at kernel level affects proper modelling of spectral features
- Coupling of feature extraction techniques with conventional classifiers may not always give acceptable results

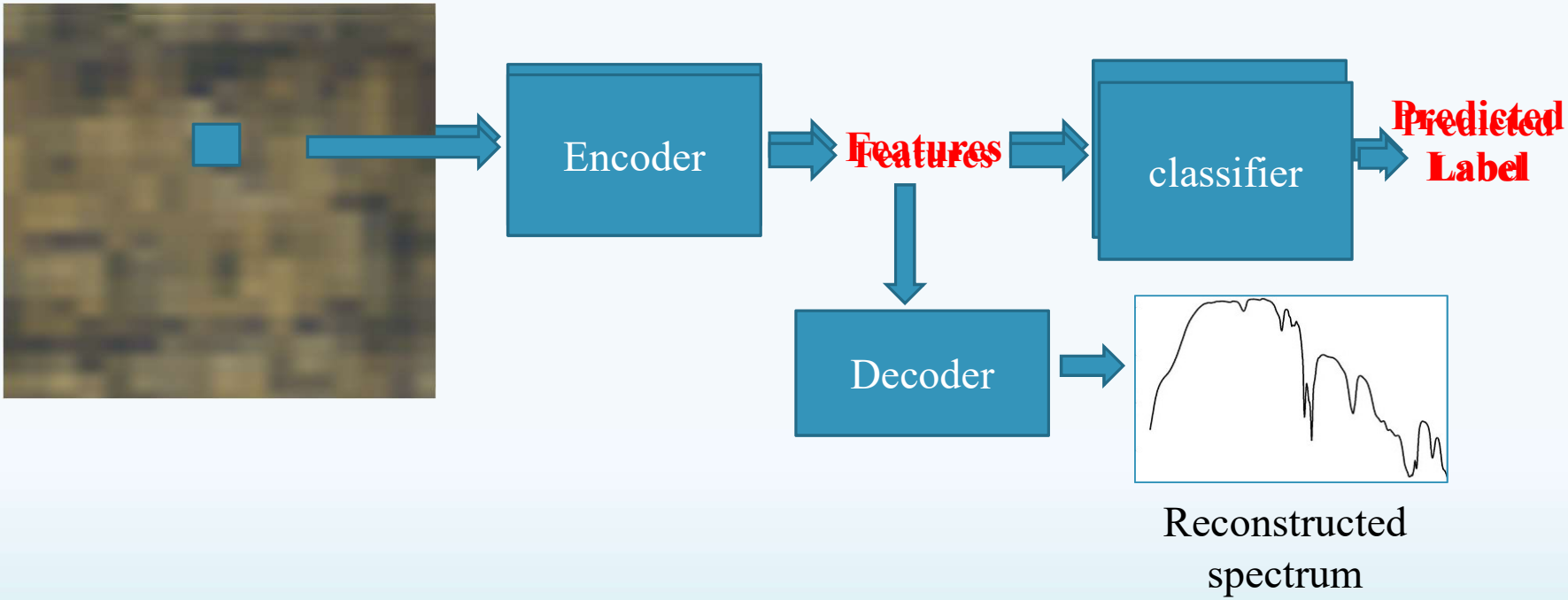
Scene Classification vs Satellite Image Classification



Source: SIP Lab, IIT Bombay & MNIST datsaet

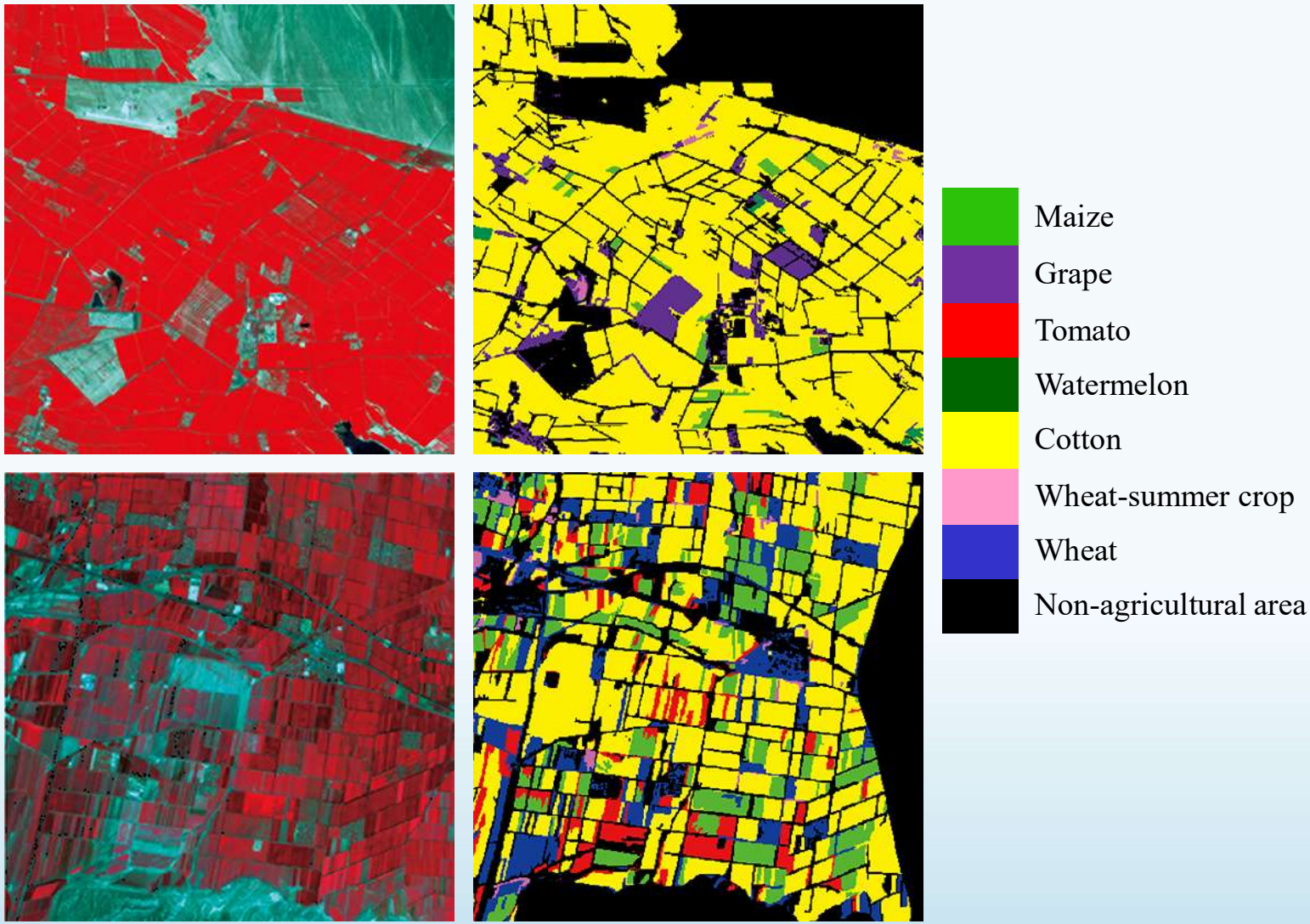
Spatial-spectral Classification

Along with the spectrum, neighbourhood is also employed



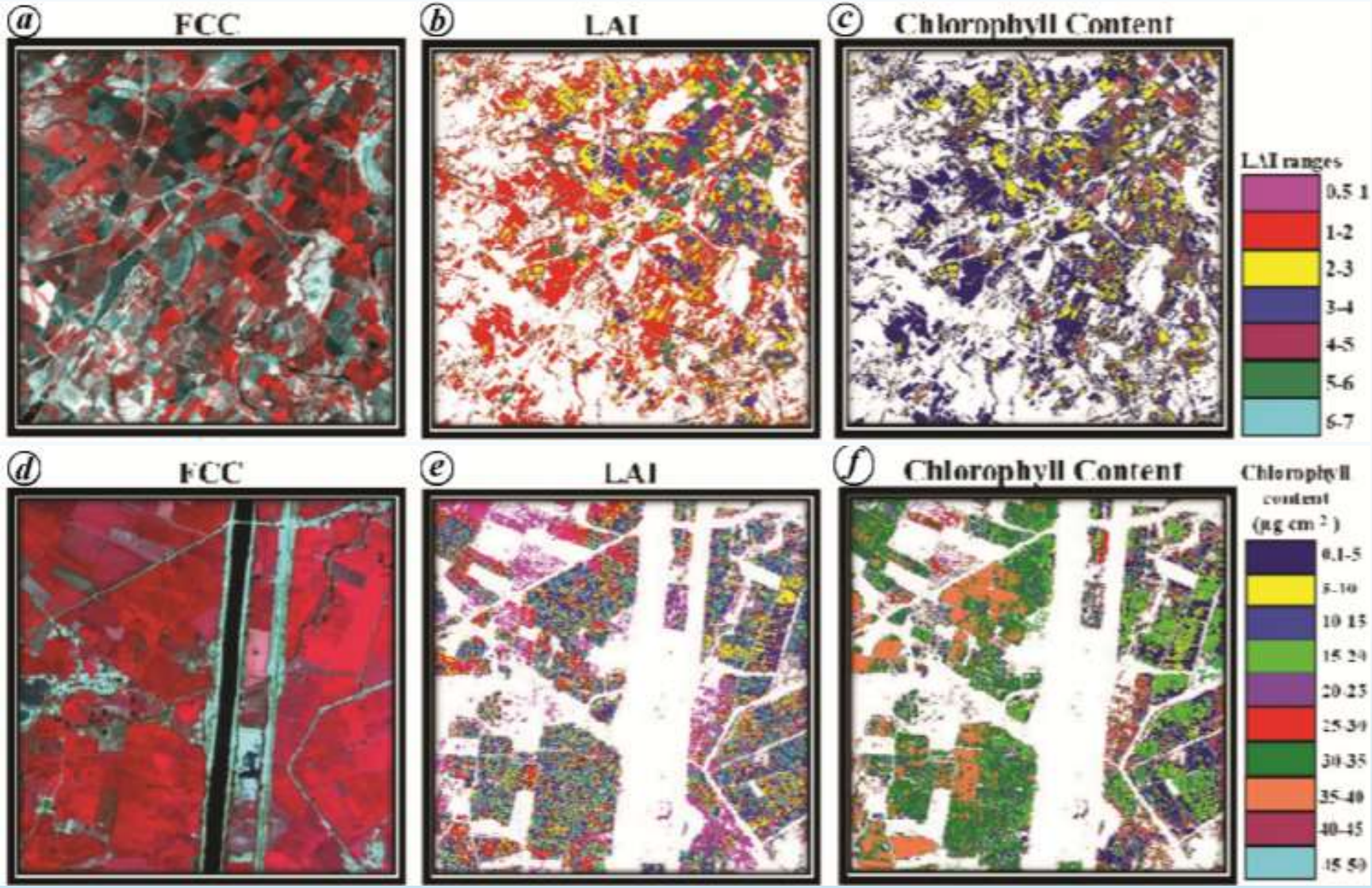
Source: SIP Lab, IIT Bombay & MNIST dataset

CNN based Crop-type Classification



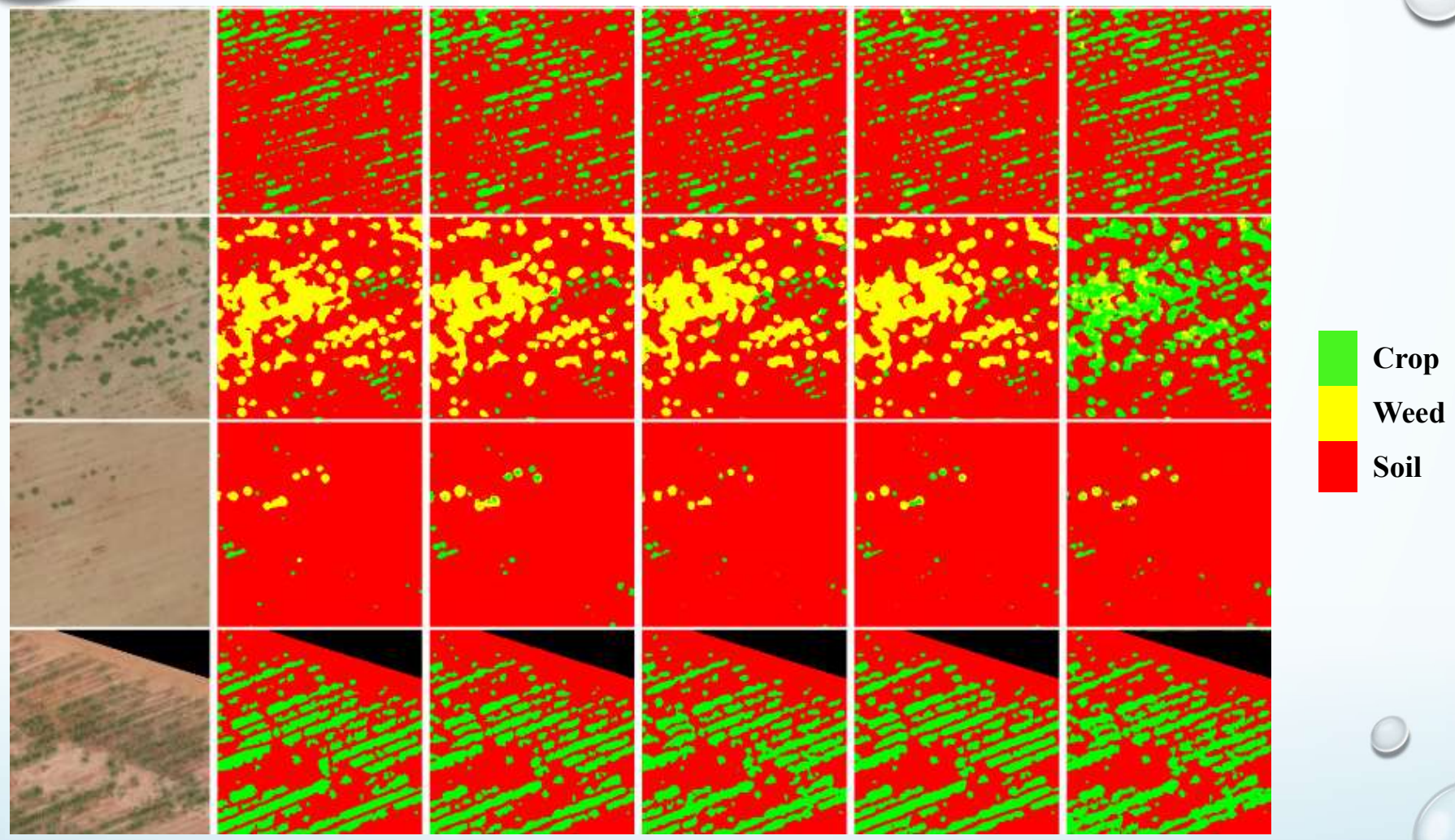
Source: https://www.semanticscholar.org/paper/Deep-Learning-Classification-of-Land-Cover-and-Crop-Kussul-Lavreniuk/

CNN based Chlorophyll and LAI Estimation



Source: https://www.semanticscholar.org/paper/Remote-sensing-for-evaluating-crop-water-stress-at-Taghvaeian

CNN based Weed Detection

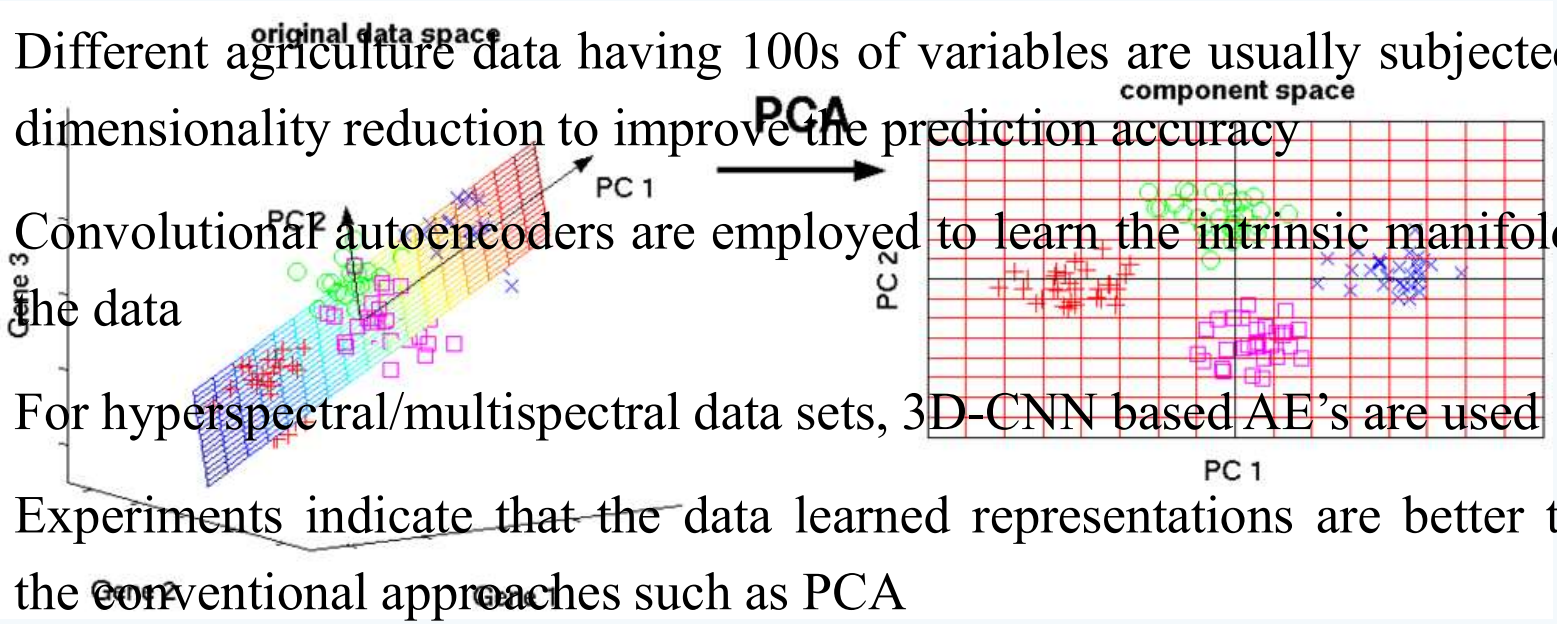


a) Original image b) Ground truth c) FRRN d) PSPNet e) SegNet f) UNet

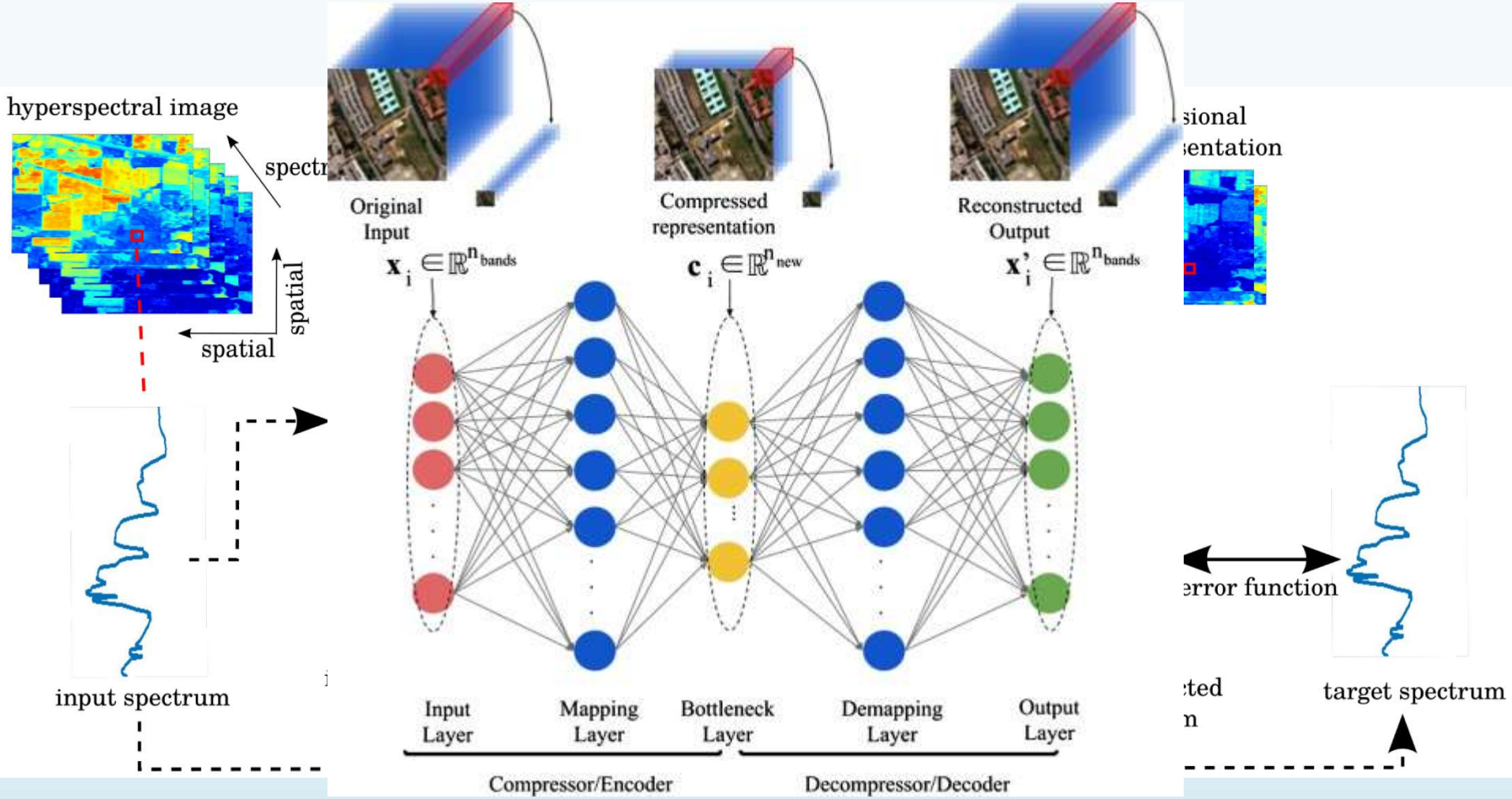
Source: https://medium.com/@awangenh/

Dimensionality Reduction

- Different agriculture data having 100s of variables are usually subjected to dimensionality reduction to improve the prediction accuracy
- Convolutional autoencoders are employed to learn the intrinsic manifold of the data
- For hyperspectral/multispectral data sets, 3D-CNN based AE's are used
- Experiments indicate that the data learned representations are better than the conventional approaches such as PCA

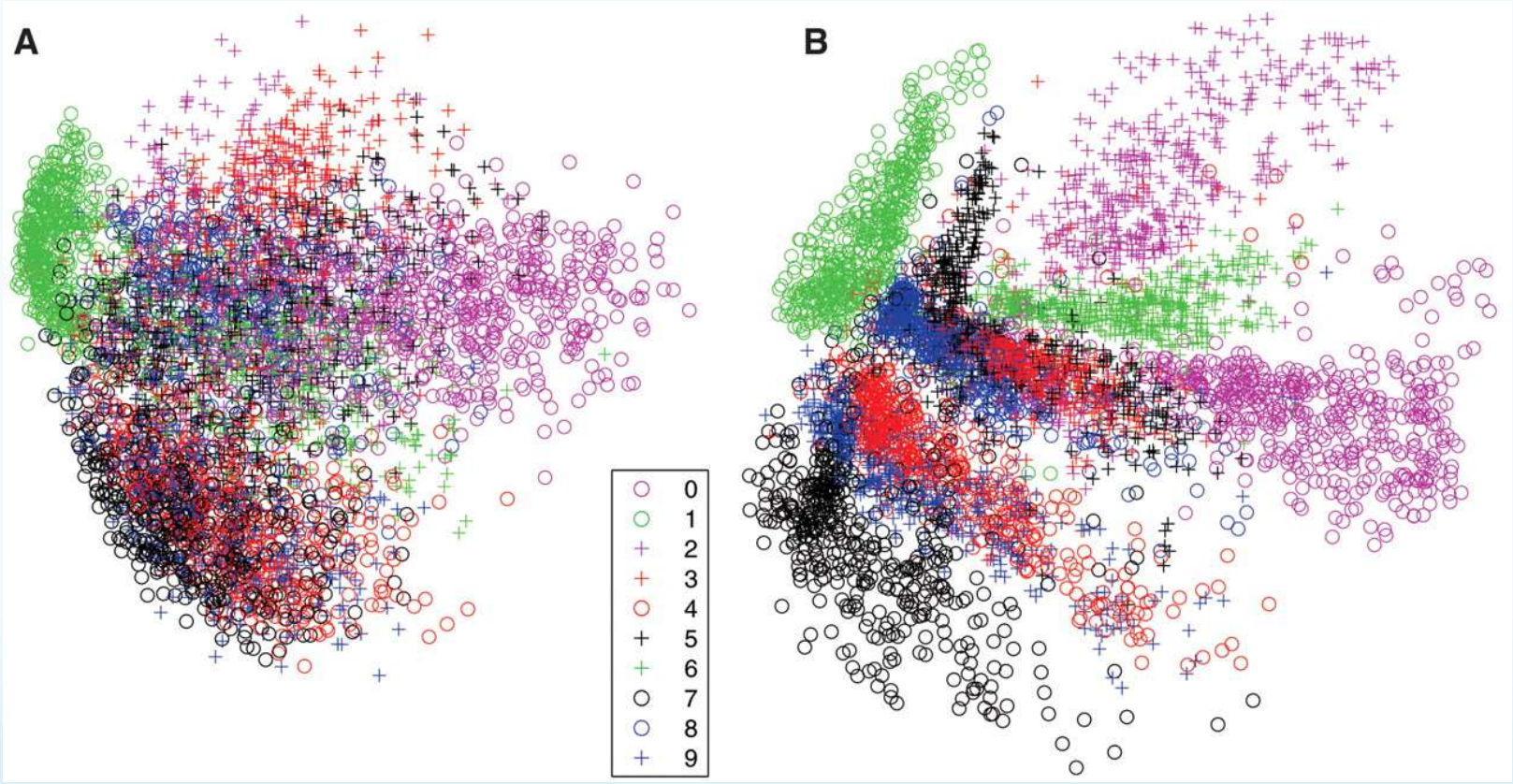


Autoencoder Based Dimensionality Reduction



Source: htttps://goodaudience.com/Fusing-tensorflow-autoencoders-with-music

Autoencoder based Dimensionality Reduction



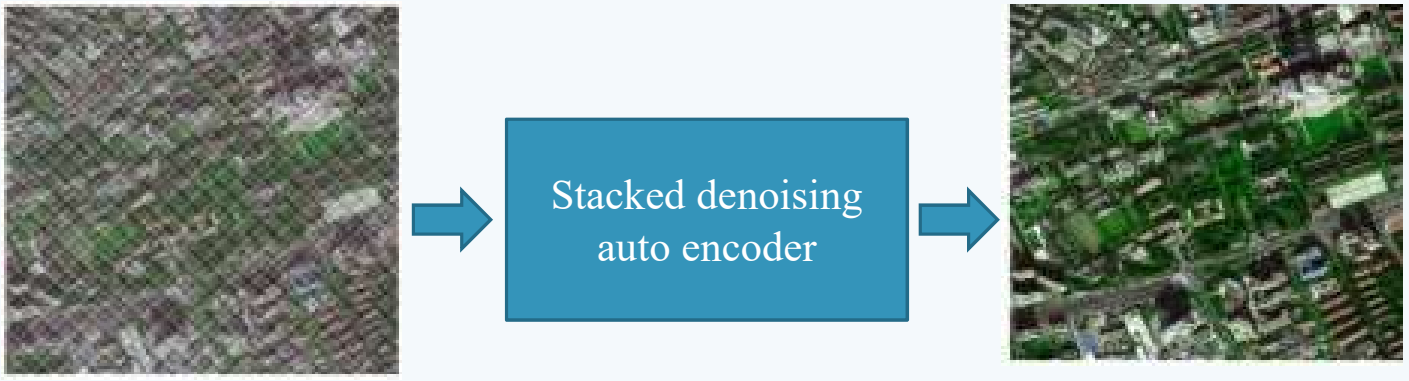
PCA representation

AE based latent representation

Source: https://stats.stackexchange.com/questions/190148/building-an-autoencoder-in-tensorflow-to-surpass-pca

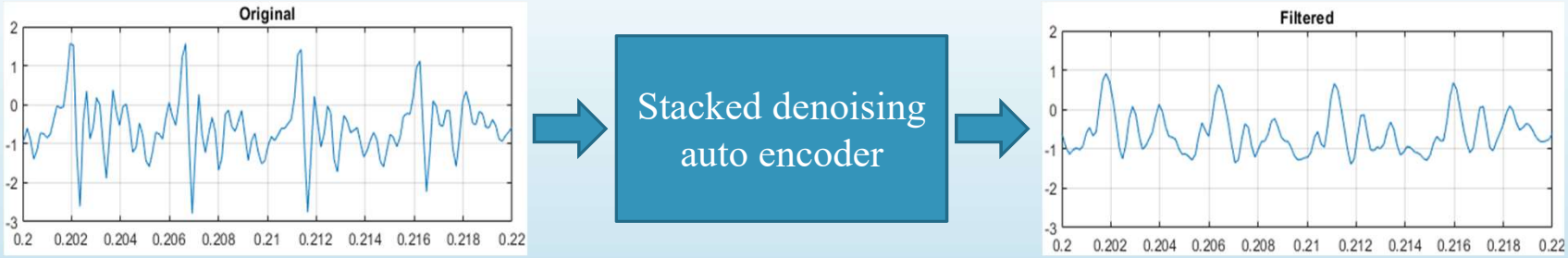
CNN based Image Denoising

Instead of training the network to reconstruct input, denoising autoencoders are trained to reconstruct input from its noisy version (Denoising, cloud removal, etc.)



Noisy image

Denoised image



Brief review of prominent sub-pixel classification approaches

- Sub-pixel mapping approaches generally attempt to maximize the spatial contiguity of the class labels(Lopez et al., 2012; Shi et al., 2014)

- Most of the existing approaches ignore the coarser level variations can better model the end member variability (Feng et al., 2016)

- In conventional approaches, spectral unmixing and sub-pixel mapping are optimized independently (Arun et al., 2016; Jin et al., 2012; Wang et al., 2015)

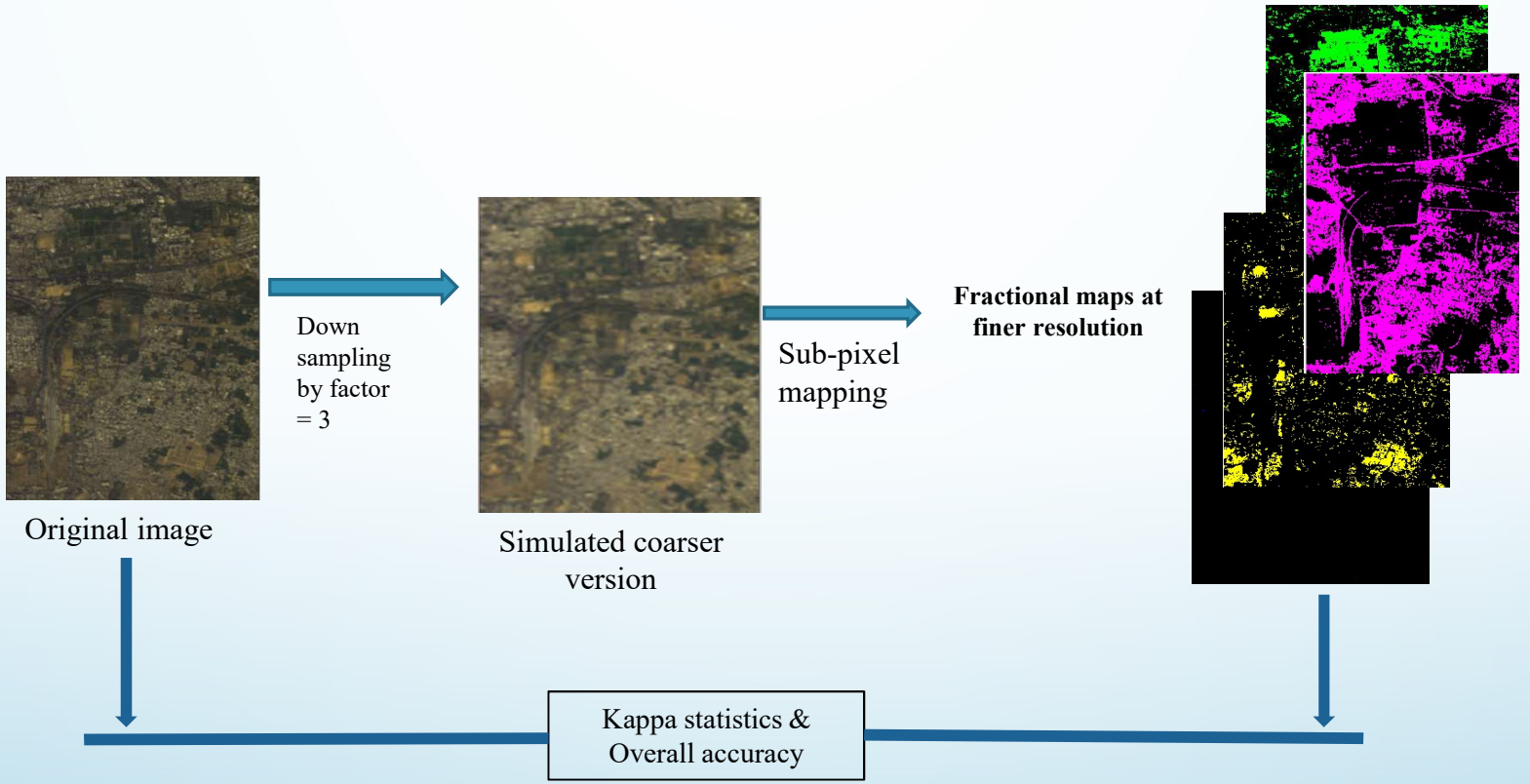
Proposed CNN based sub-pixel classification architecture

In order to jointly optimize the unmixing and mapping stages,

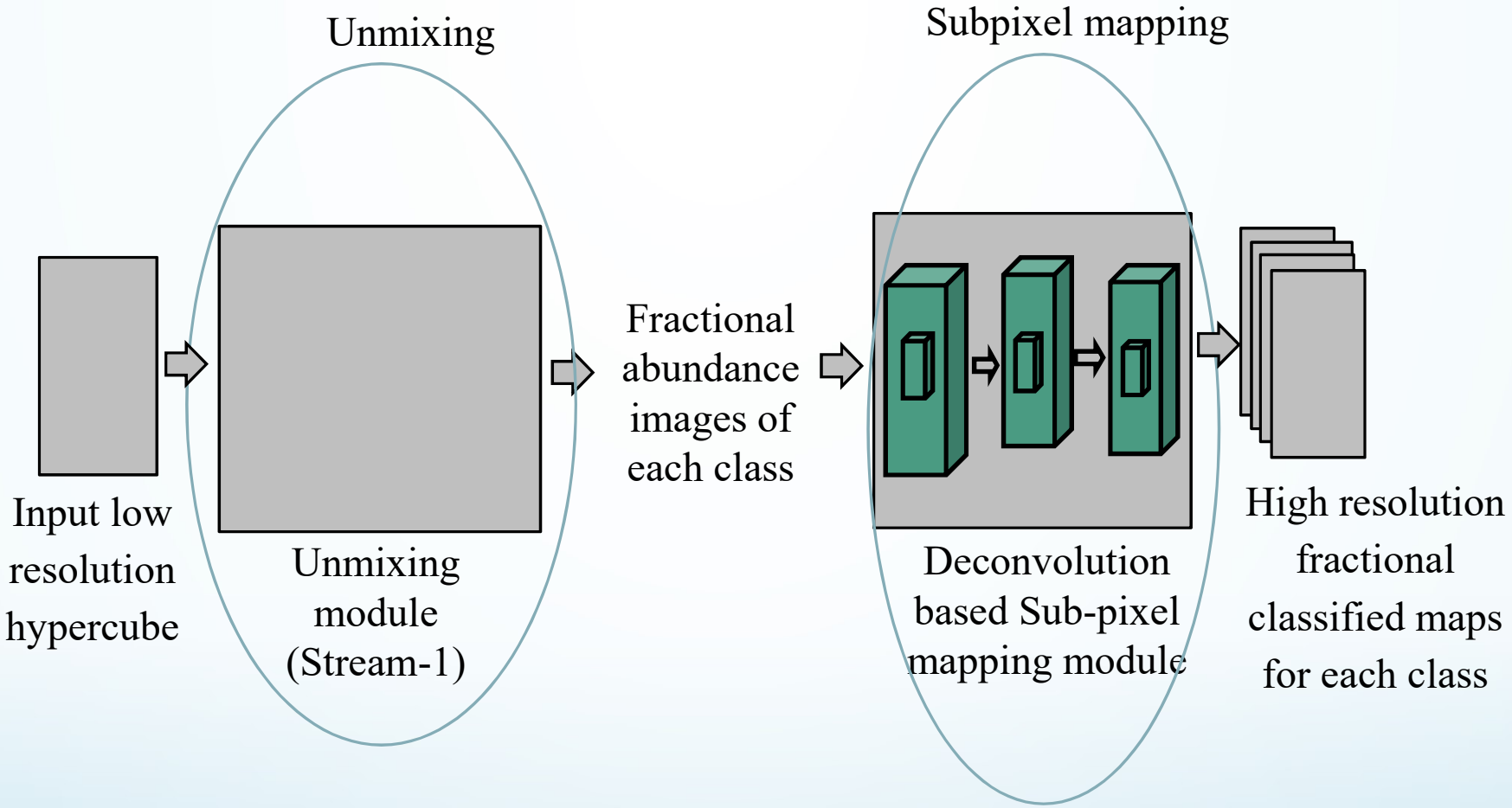
- Unmixing is implemented as an encoder-decoder or LSTM based soft classification module
- Sub-pixel mapping is implemented as a deconvolution network that transforms the fuzzy memberships (class fractions) to finer spatial scale
- Mapping transformation is implemented not to be class specific but based on the coarser pixel distribution (making the problem less ill posed)

TRAINING SUB-PIXEL MAPPING FRAMEWORK

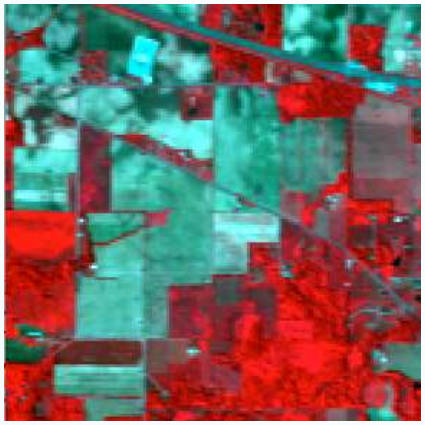
- Downscale a high resolution image and use the simulated version as the input
- Compare the precited maps at high resolution with the classification result of the original ground truth



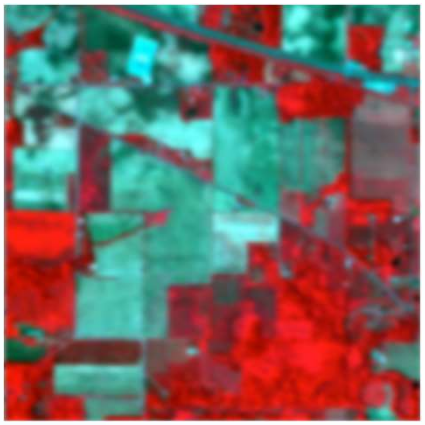
Proposed CNN based sub-pixel classification architecture



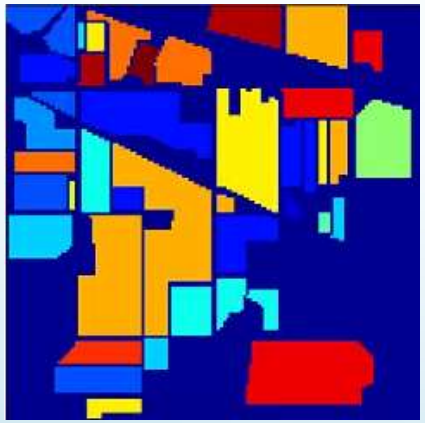
Results of the proposed CNN based sub-pixel mapping approach



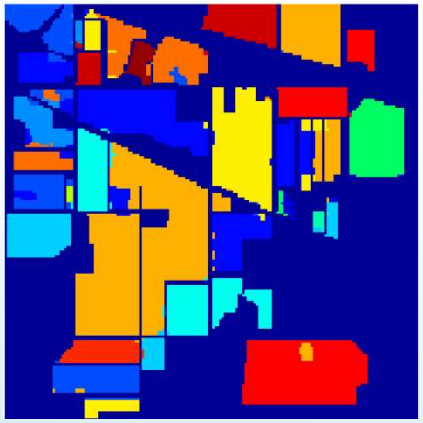
a) Indian Pine image



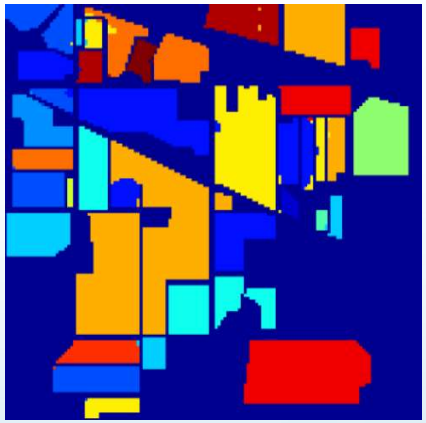
b) Simulated coarse image



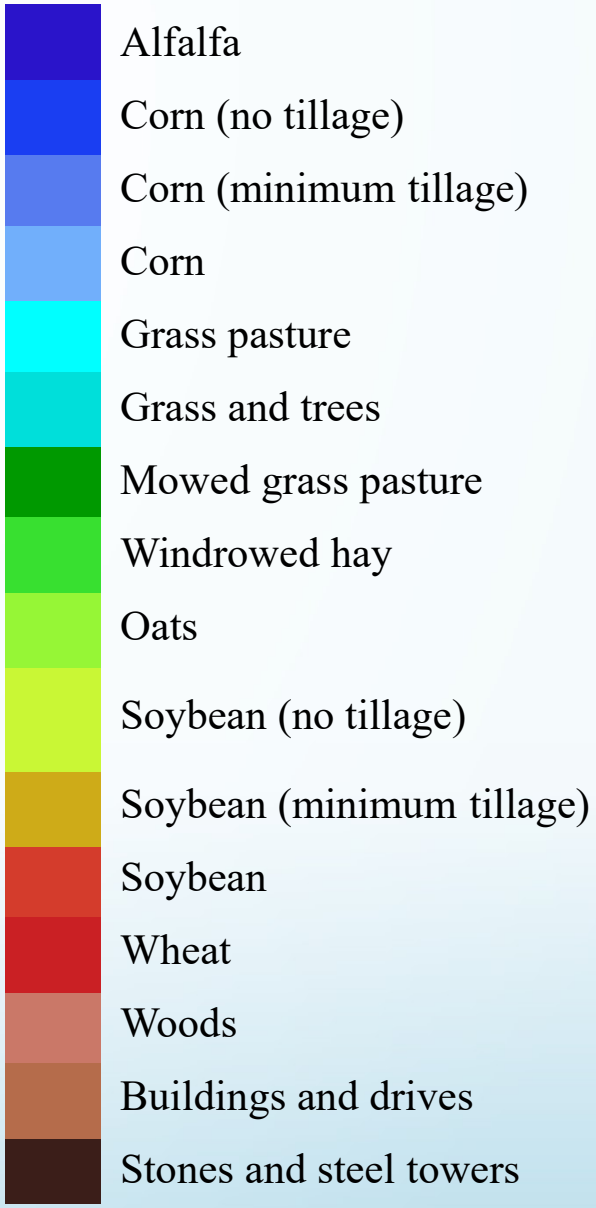
c) Ground truth image



d) Semi-variogram based



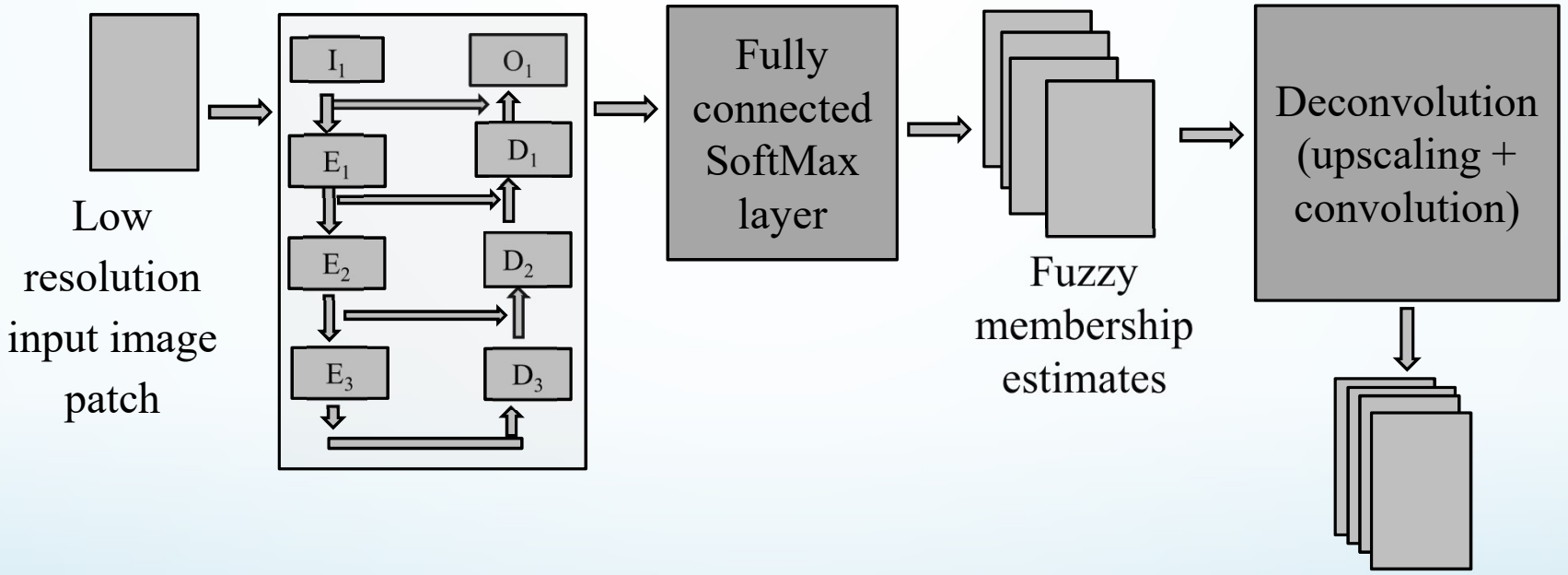
e) Proposed



Implementation Details

Encoder-stream encodes the spectral-spatial information and is mapped by the decoder-stream and Softmax layer into fuzzy memberships.

Encoder-decoder based unmixing sub-network resembles the linknet architecture (chaurasia and Culurciello, 2017).



Link

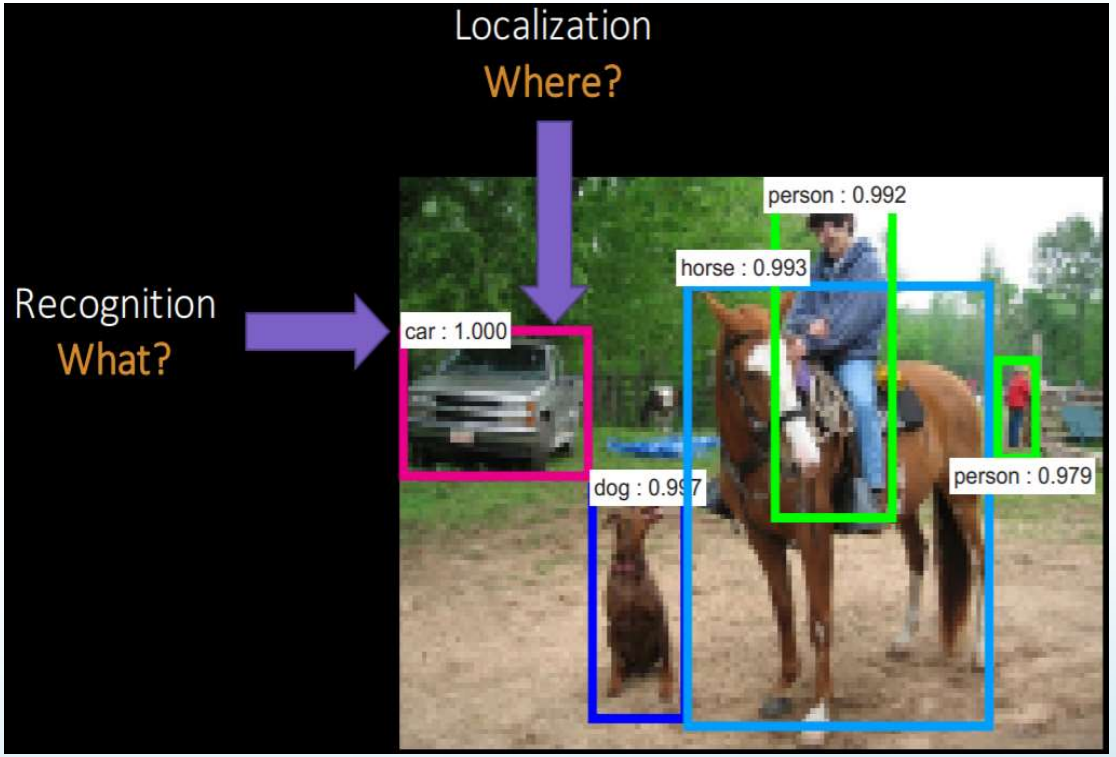
Discussion

- Joint optimization of unmixing and mapping improves the results
- The proposed framework facilitate the use of ground truth samples for improving the abundance estimation and sub-pixel mapping
- CNN learns the features and deconvolution facilitate the automatic prediction of fractional images at high spatial resolution
- Modified Bayesian based hyper-parameter optimization (Zhang et al., 2019) is employed to optimize the hyperparameters

Discussion

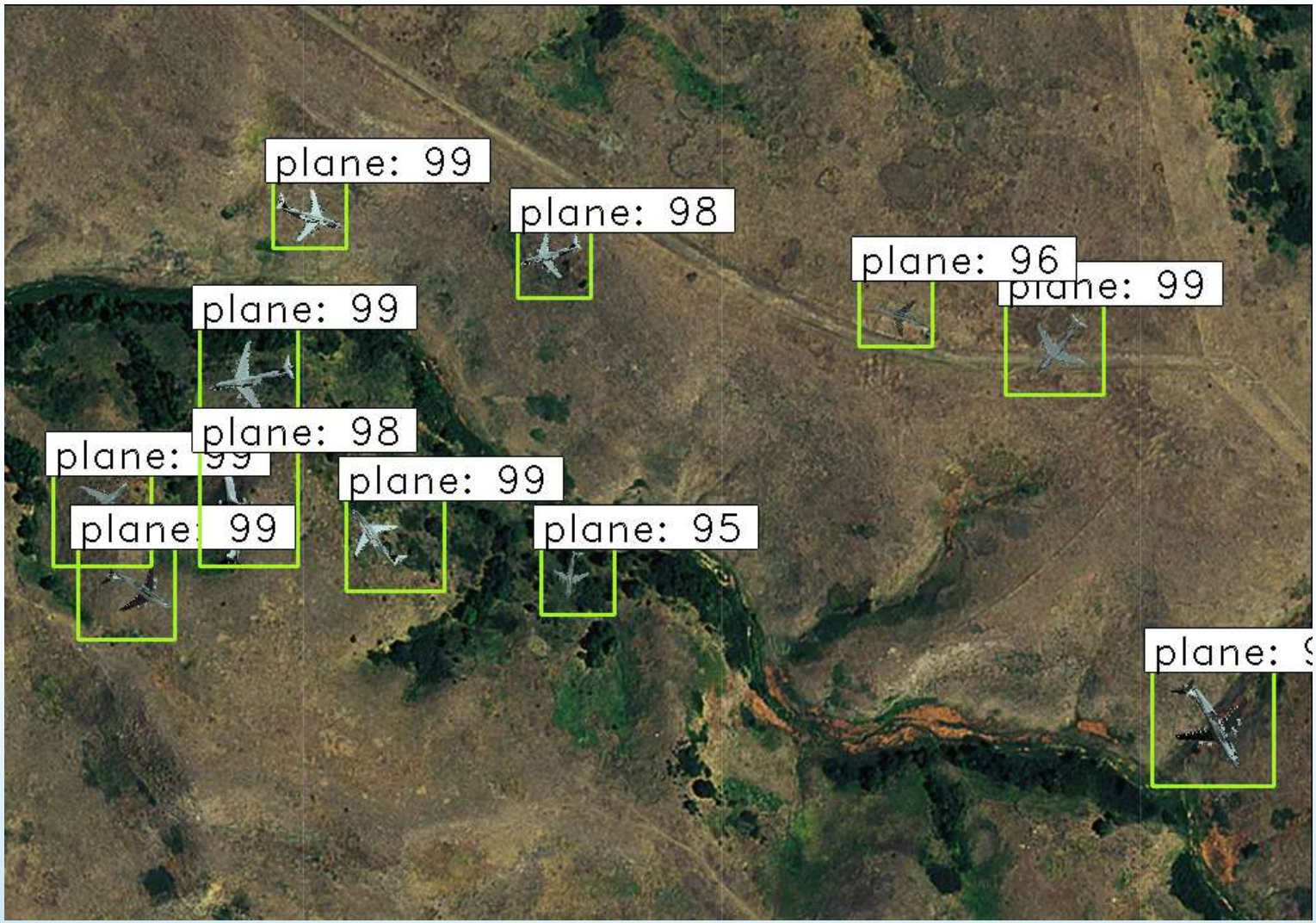
- Increase in number and size of filters improves the accuracy but should be selected based on the number of training samples
- Depth of the network need to be tuned with respect to the available data
- Architectural variations (LSTM, Encoder-decoder, Capsules) can be explored for the unmixing module
- For limited number of training samples, the variogram based model performs better

Region CNN



Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Region CNN

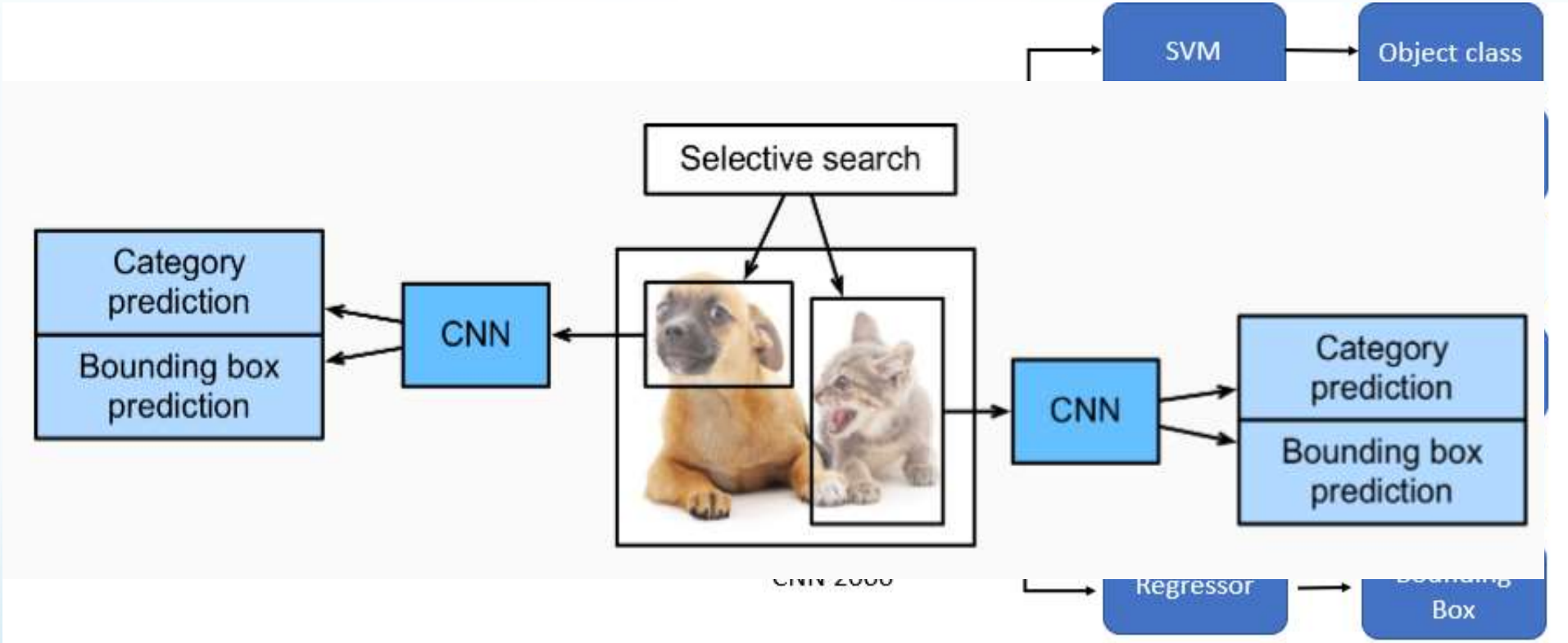


Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Region CNN: Region proposal Generation

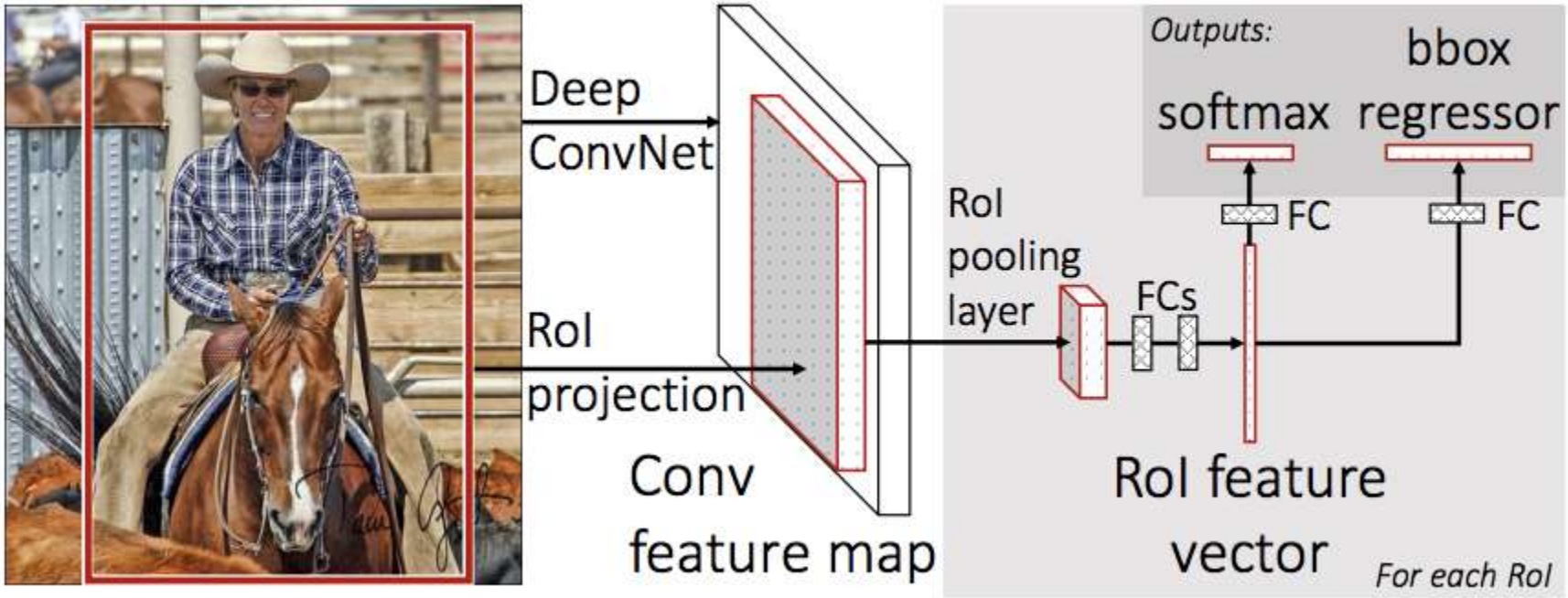
Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Region CNN



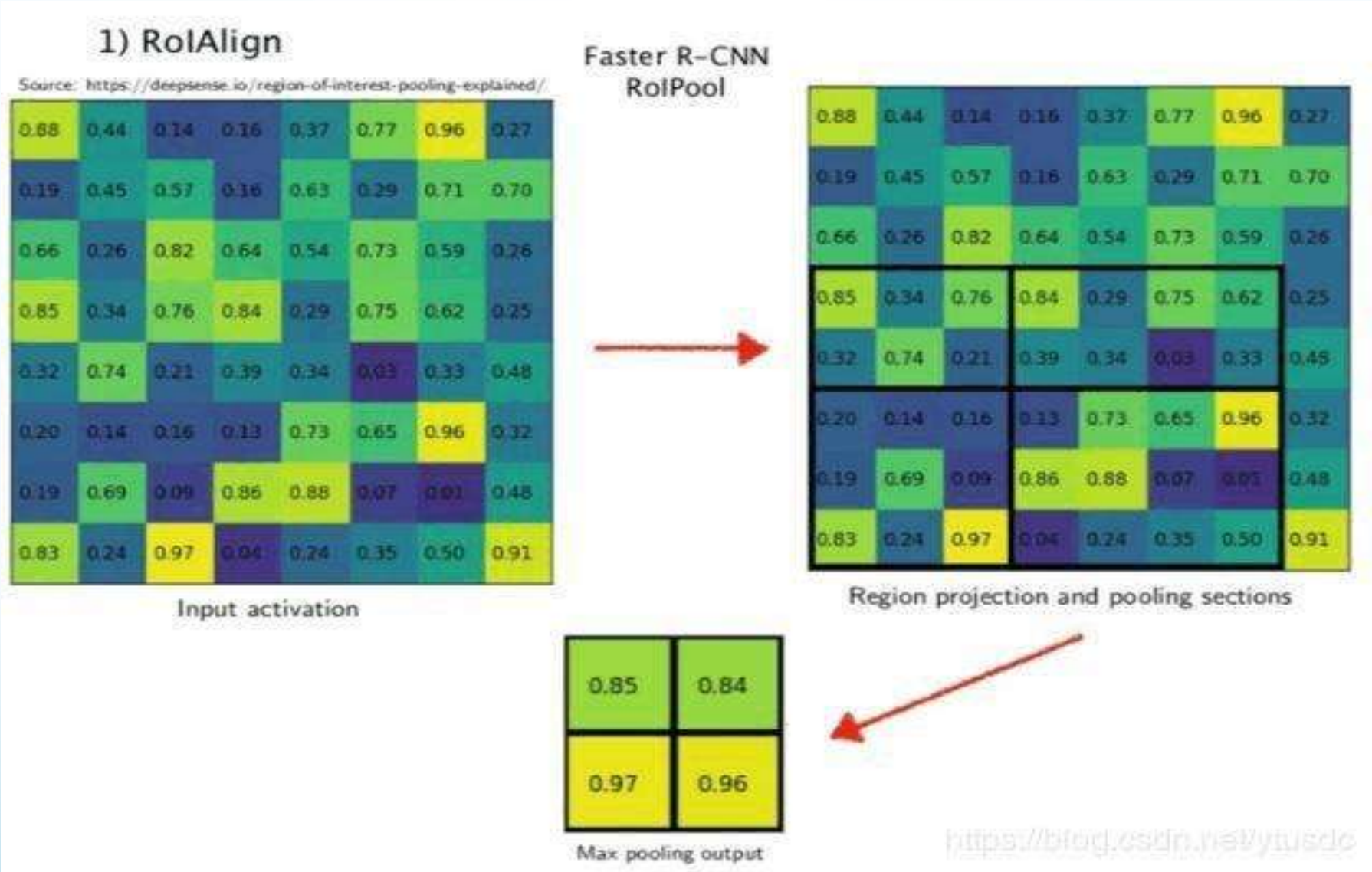
Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Fast Region CNN



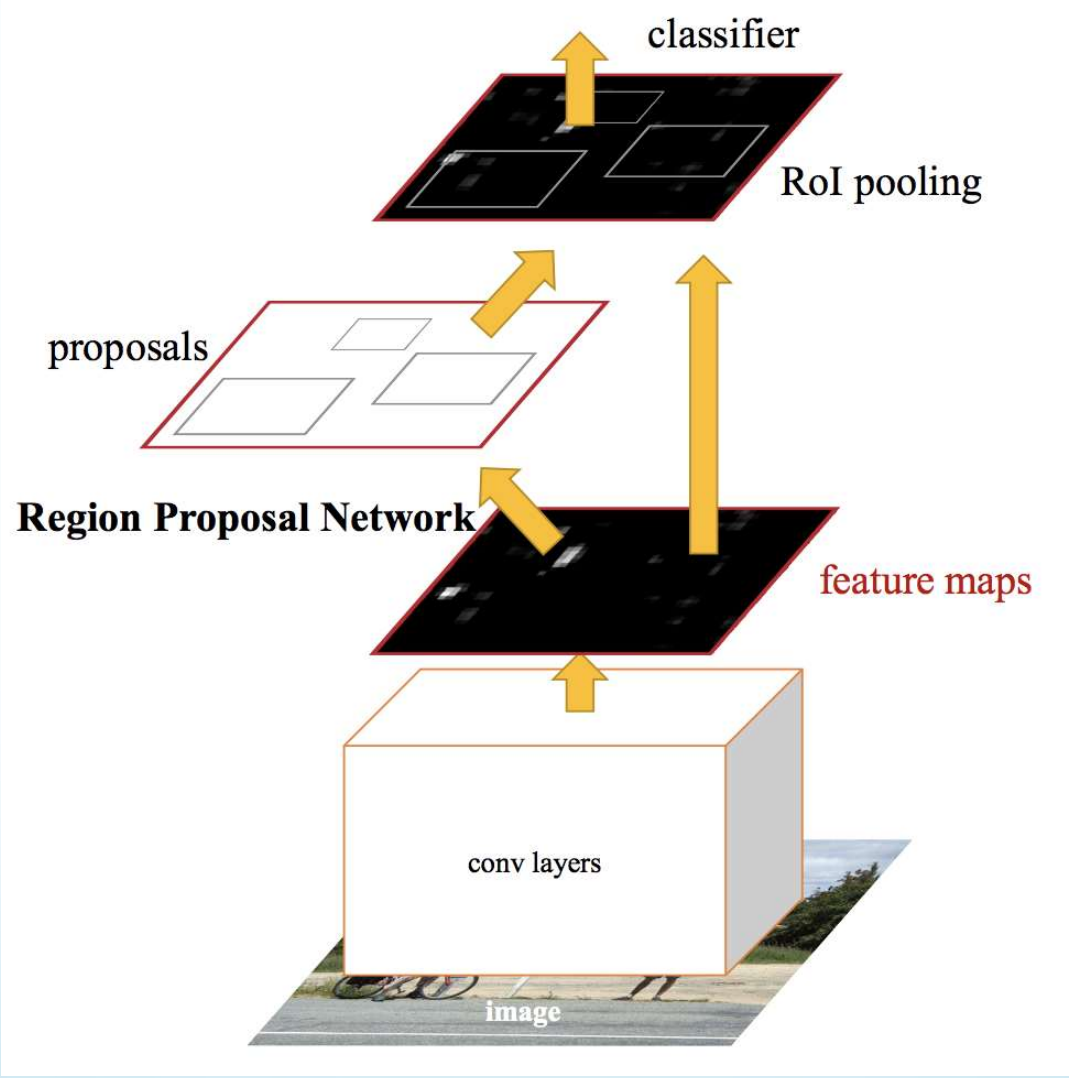
Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Fast Region CNN: ROI Pooling



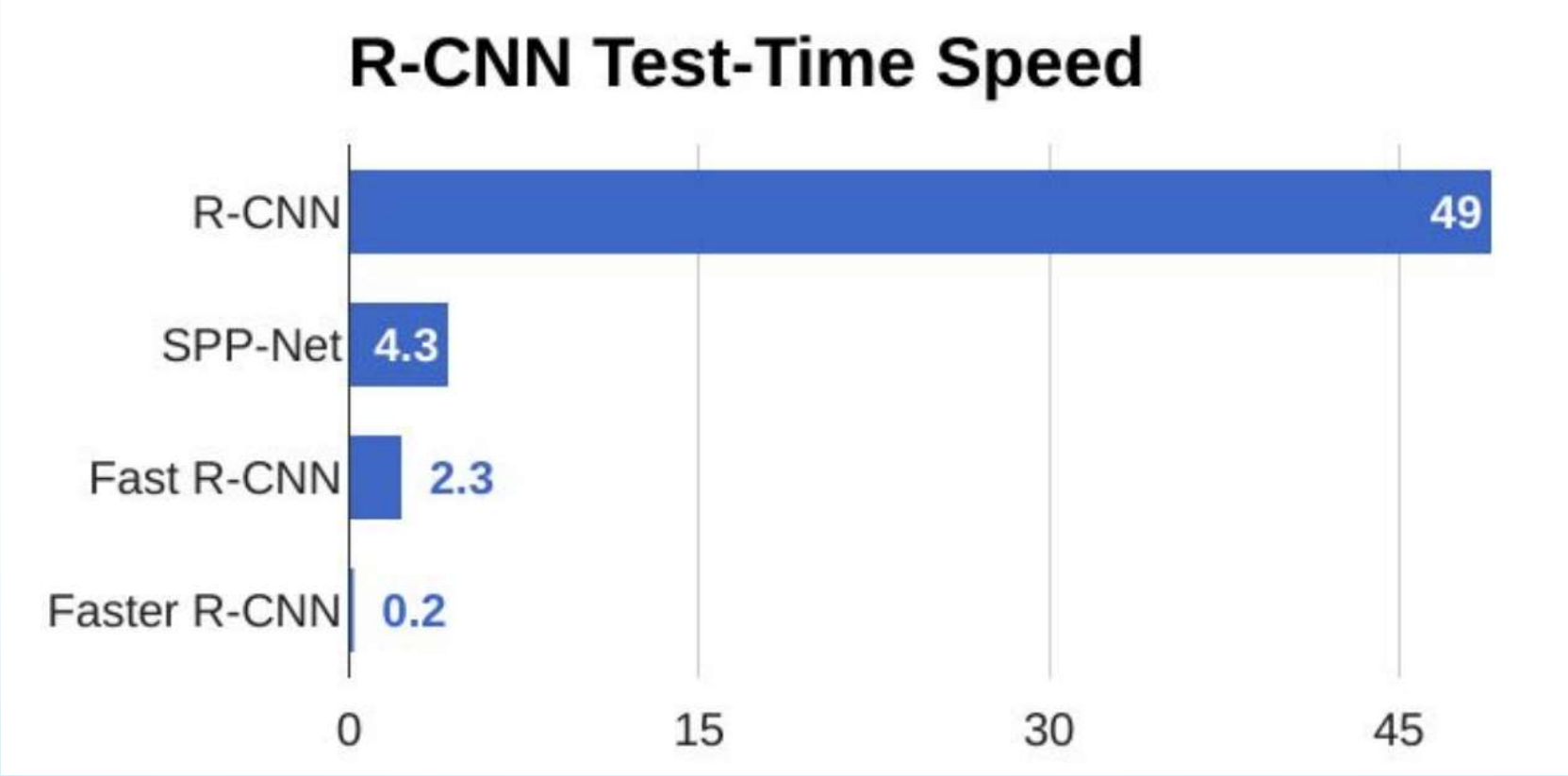
Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Faster Region CNN



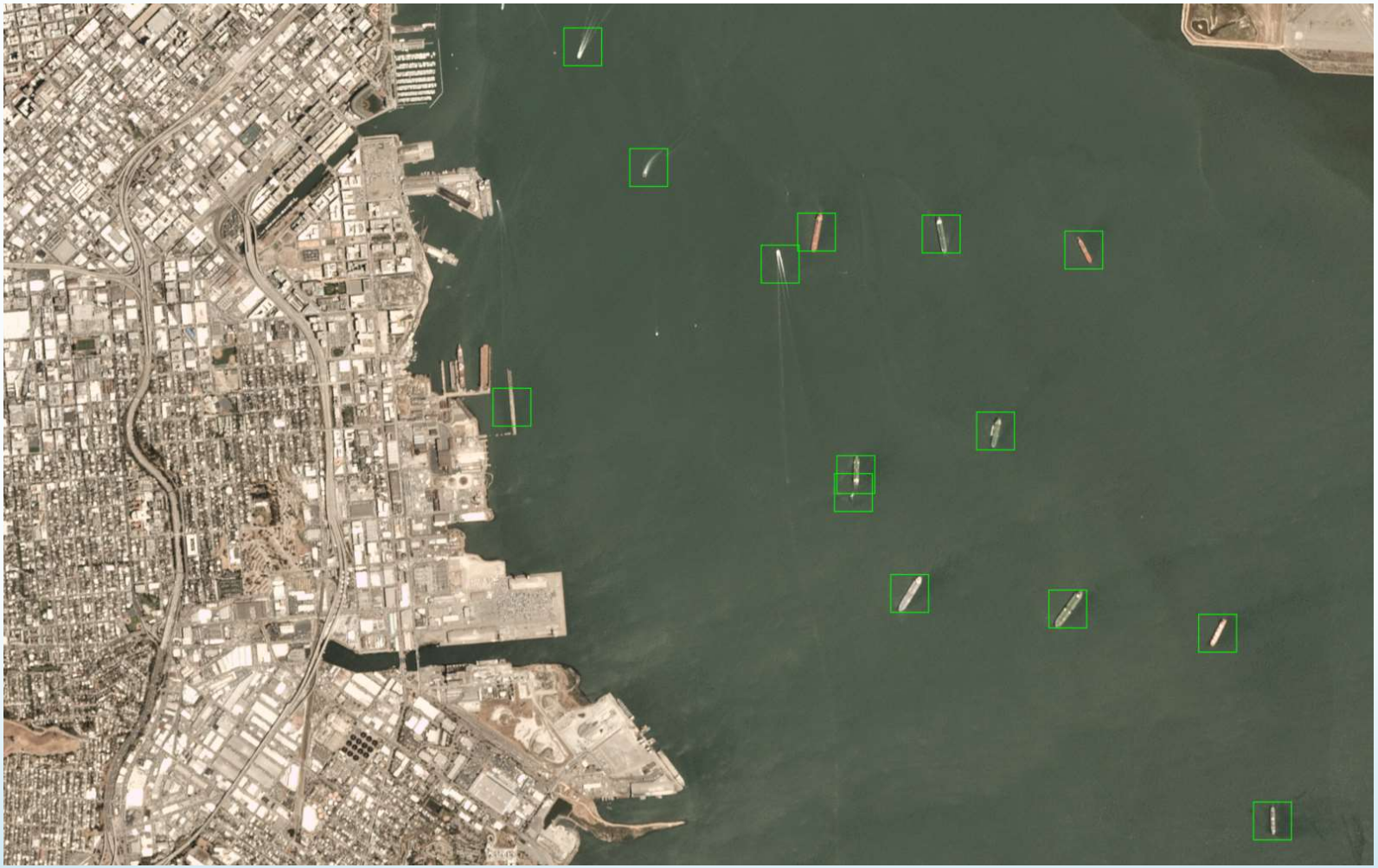
Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Region CNN



Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

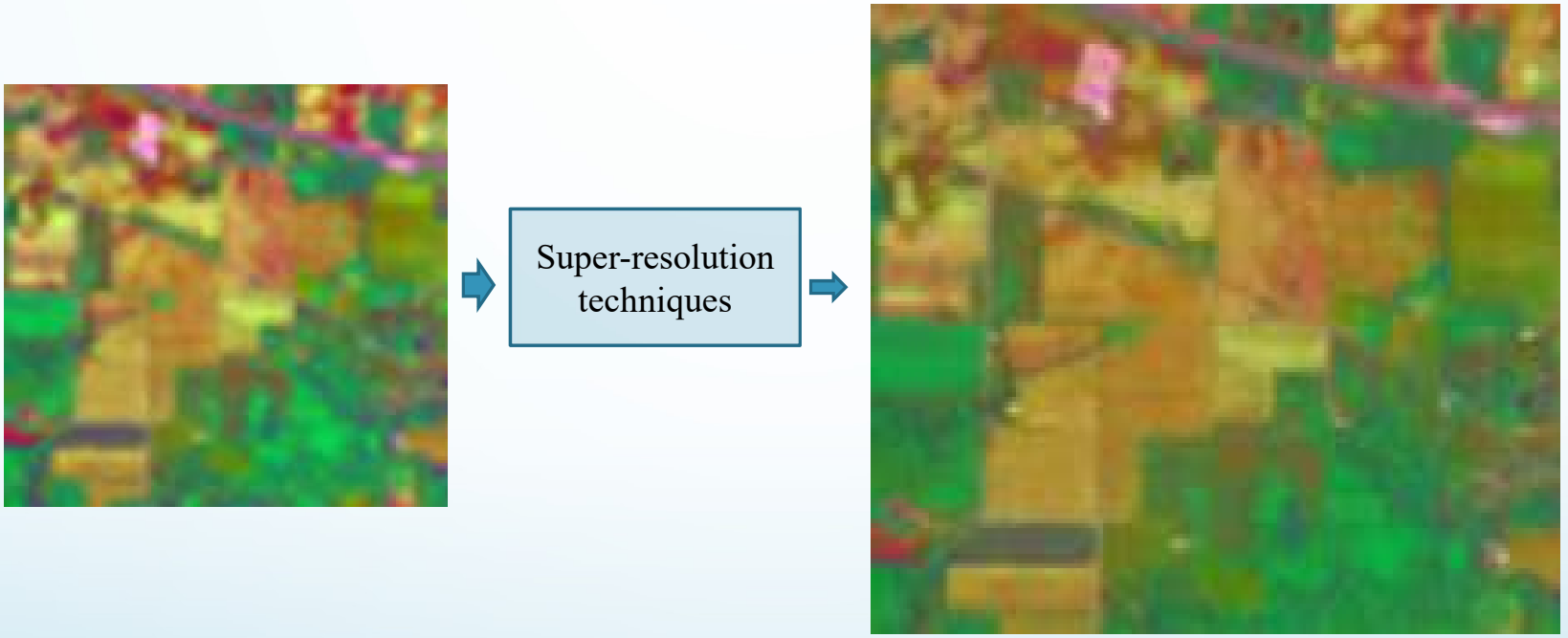
Region CNN



Source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Spatial Super-resolution

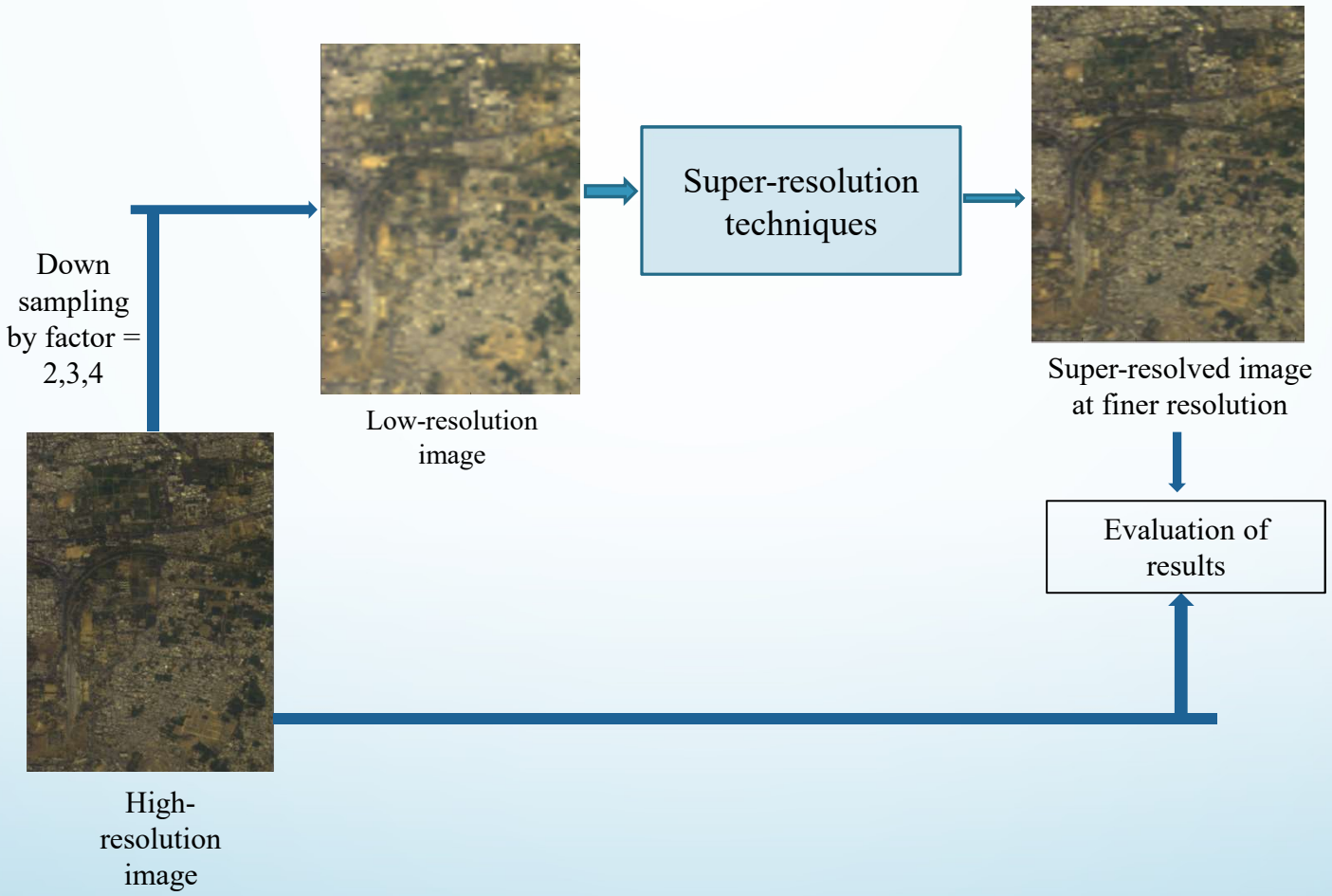
Dictionaries & sparse codes: dictionary (D) and sparse code (α) are the base and coefficient matrix respectively using which an image can be represented as:



Source: https://elad.cs.technion.ac.il/wp-content/uploads/2018/02/Super-Resolution_SIAM_IS_2010.pdf

Training super-resolution framework

- Downscale a high resolution image and use the simulated version as the input
- Compare the precited image at high resolution with the original ground truth



Regularization

Identify the best parameters, **w**, for a regression function

$$y = w_0 + \sum_{i=1}^{N} w_i x_i$$

Recall: **overfitting** happens when a model is capturing idiosyncrasies of the data rather than generalities.

- Often caused by too many parameters relative to the amount of training data.
- Usually weights will be very large

Dealing with overfitting

- Use More Data
- Use A Tuning Set
- **Regularization**

Penalize large weights

- Introduce a penalty term in the loss function.

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}\{t_n - y(x_n, \vec{w})\}^2$$

Regularized regression
(L2-regularization or ridge regression)

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \frac{\lambda}{2}\|\vec{w}\|^2$$

More regularization

- **Regularization**
  - **L0 norm = Number of non zero coefficients**
  - **L1 norm = Absolute sum of coefficients**
  - **L2 norm = Sum of squares**

When penalizing a model using the L2 norm, it is highly unlikely that anything will ever be set to zero, since the reduction in L2 norm going from ε to 0 is almost nonexistent when ε is small. On the other hand, the reduction in l1 norm is always equal to δ, regardless of the quantity being penalized.

More regularization

The penalty term defines the styles of regularization

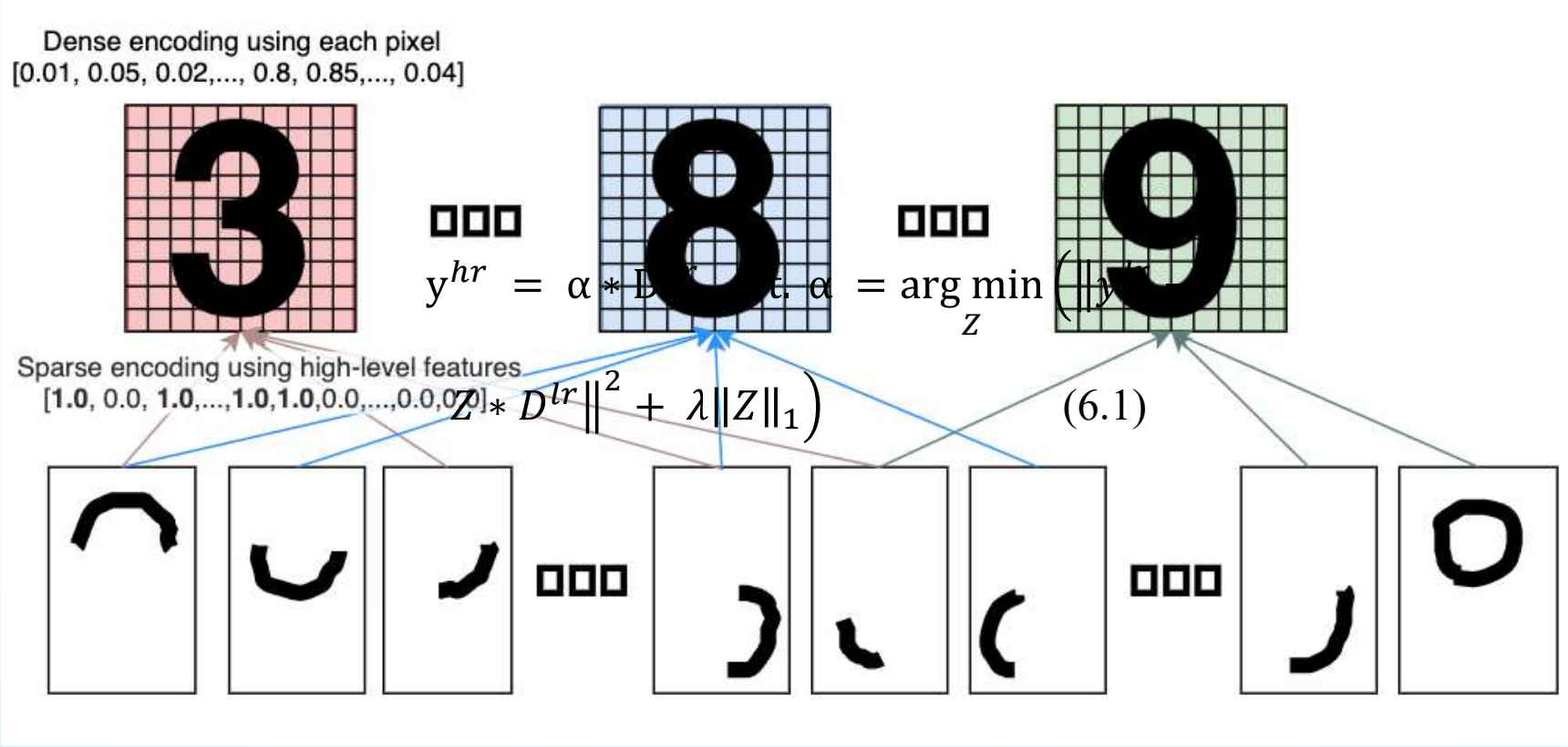
- L2-regularization
- L1-regularization
- L0-regularization
  - **L0-norm** is the optimal subset of features

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \frac{\lambda}{2}\|\vec{w}\|^2$$

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \lambda|\vec{w}|_1$$

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \lambda\sum_{n=0}^{N-1}\delta(w_n \neq 0)$$

Sparse coding



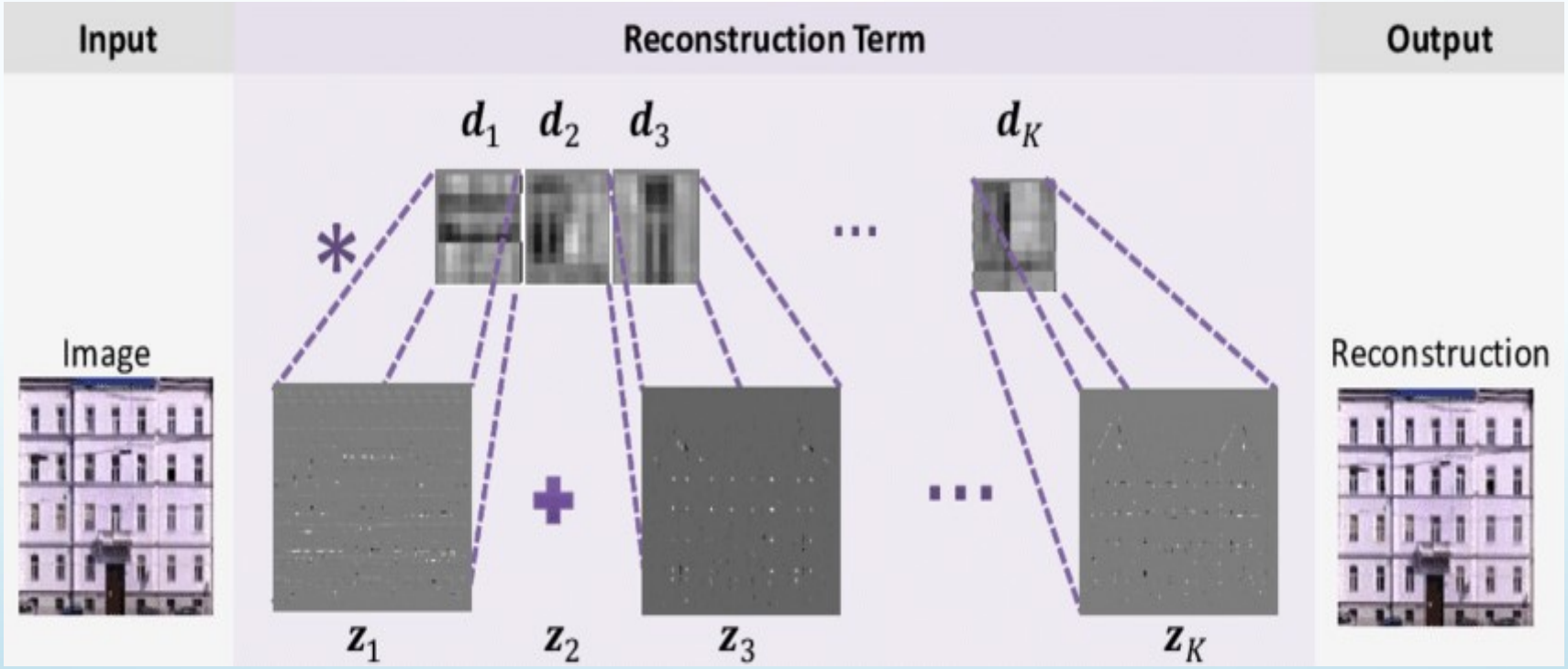
$$y^{hr} = \alpha * D^{hr} \quad s.t. \ \alpha = \arg\min_{Z} \left(\| y^{lr} - Z * D^{lr} \|^2 + \lambda \|Z\|_1 \right) \qquad (6.1)$$

Source: https://elad.cs.technion.ac.il/wp-content/uploads/2018/02/Super-Resolution_SIAM_IS_2010.pdf

Sparse coding

Dictionaries & sparse codes: dictionary (D) and sparse code (α) are the base and coefficient matrix respectively using which an image can be represented as:

$$y = D\alpha$$



Source: https://elad.cs.technion.ac.il/wp-content/uploads/2018/02/Super-Resolution_SIAM_IS_2010.pdf

Sparse coding

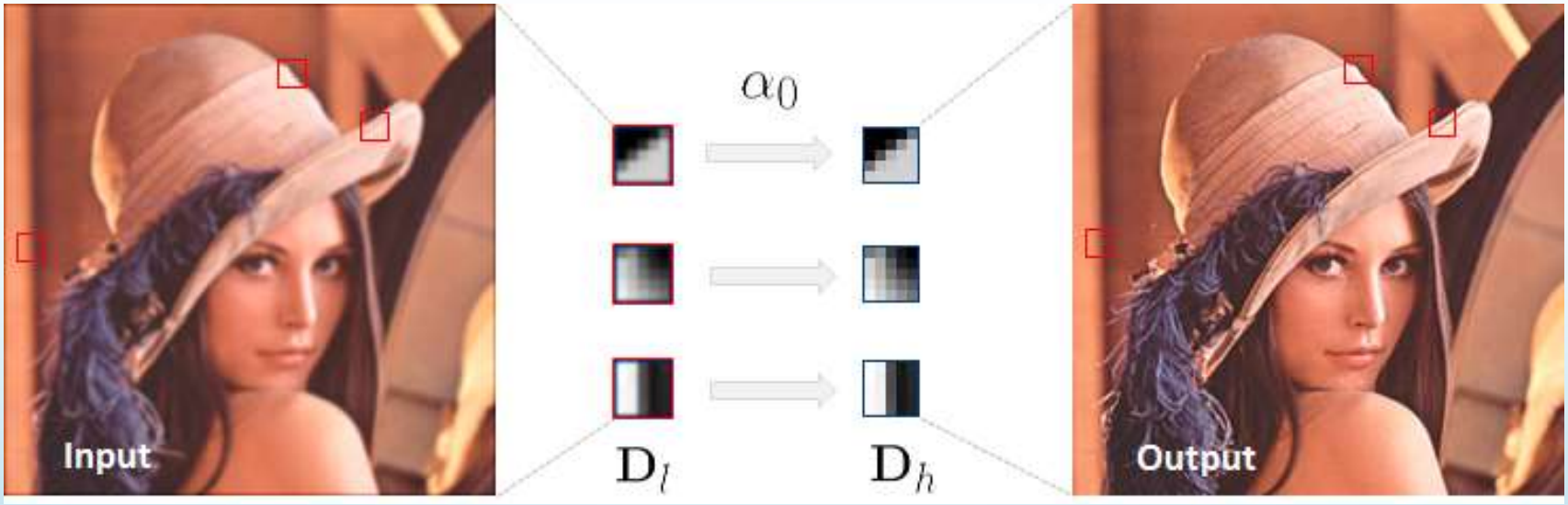
$$\boldsymbol{\alpha} = \arg\min_{\boldsymbol{Z}} \left(\left\| \boldsymbol{y^{lr}} - \boldsymbol{Z} * \boldsymbol{D^{lr}} \right\|^2 + \boldsymbol{\lambda} \|\boldsymbol{Z}\|_1 \right)$$

Source: https://elad.cs.technion.ac.il/wp-content/uploads/2018/02/Super-Resolution_SIAM_IS_2010.pdf

Spatial Super-resolution

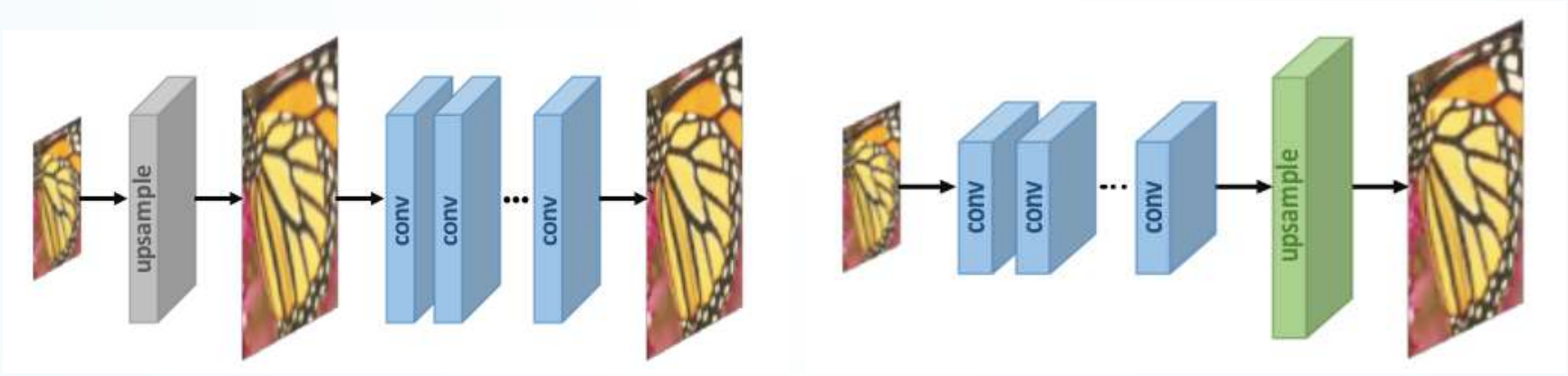
Sparse coding SR technique: low resolution (lr) and high resolution (hr) patches are represented using overcomplete dictionaries (D^{hr} and D^{lr}) along with some sparse linear coefficients ($\alpha_{lr} = \alpha_{hr} = \alpha$).

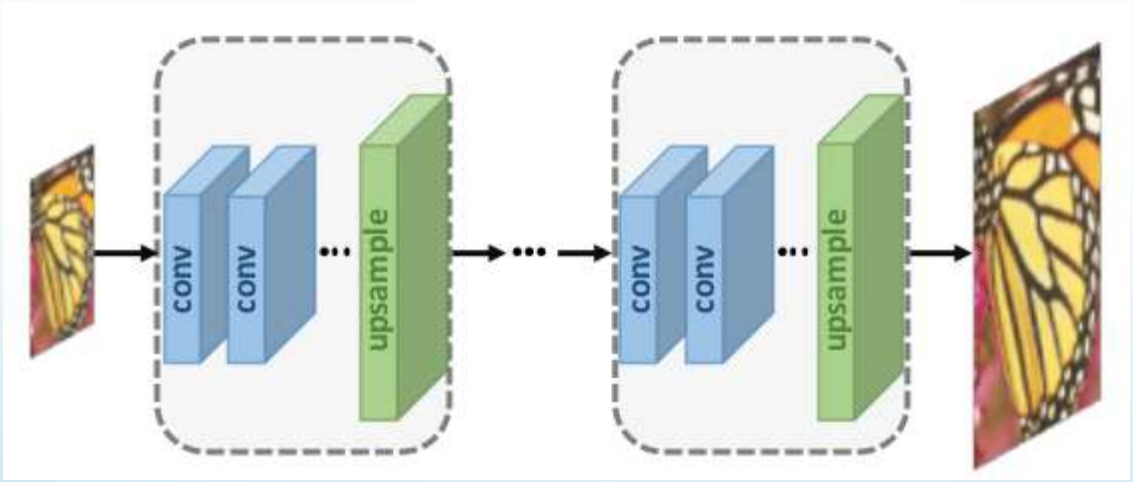
$$\boldsymbol{y^{hr} = D^{hr}\alpha,\ \text{s.t.}\,\alpha = \underset{z}{argmin}\left\|y^{lr} - D^{lr}z\right\|^2 + \lambda\|z\|_1}$$



Source: https://elad.cs.technion.ac.il/wp-content/uploads/2018/02/Super-Resolution_SIAM_IS_2010.pdf

Spatial Super-resolution





Source: https://elad.cs.technion.ac.il/wp-content/uploads/2018/02/Super-Resolution_SIAM_IS_2010.pdf

Limitations of existing SISR approaches for hyperspectral images

- **Matrix factorization based approaches** are optimal for fusion and not for SISR.

- **Sparse code optimization based approaches** are the state-of-the-art but:
  - Suffer bottleneck due to huge spectral dimension of hyperspectral datasets
  - Are too complex for images with huge spectral dimension
  - Initial bicubic upscaling results in reconstruction artefacts

- **Conventional perceptual and image space loss functions** increase the computational complexity and affect the spectral fidelity of reconstructions.

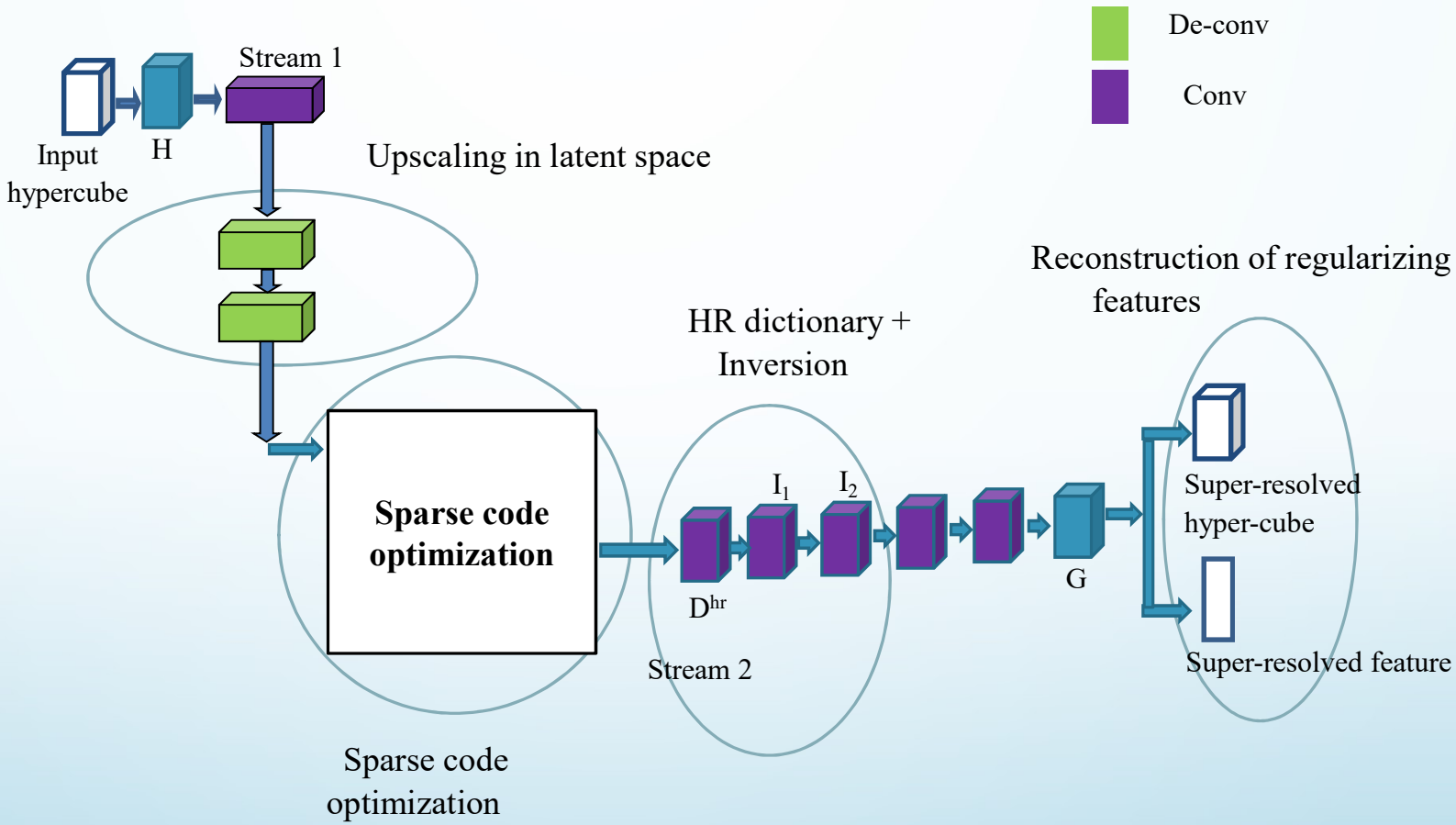
Proposed SR framework for hyperspectral images

The proposed framework

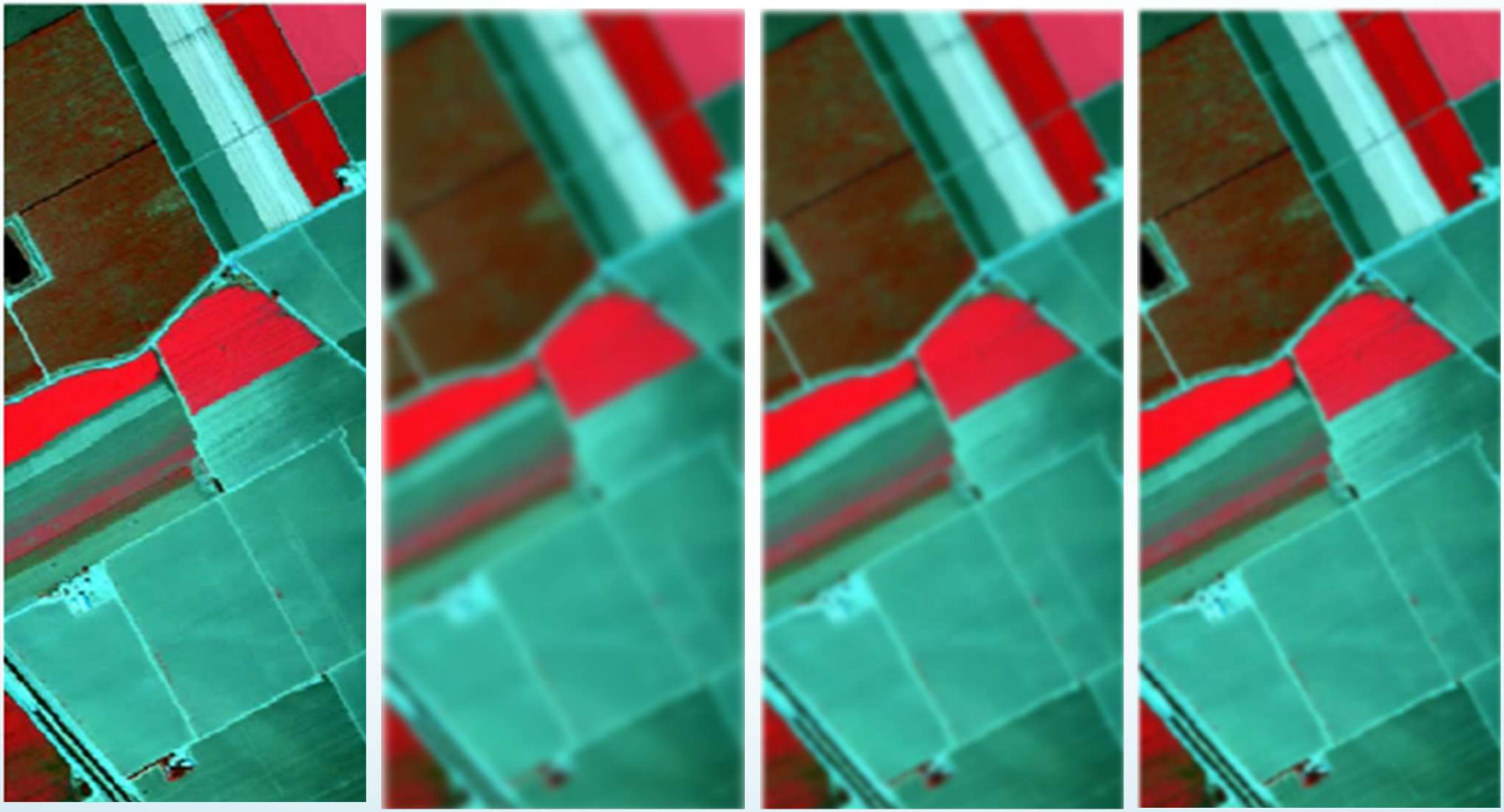
- Facilitates simultaneous consideration of spectral bands to model the spectral and spatial features
- Adopts 3D-CNN based dynamic upscaling instead of bicubic interpolation to remove reconstruction artefacts
- Use feature reconstruction-based loss and hypercube specific sparse coding strategies to avoid bottleneck due to high spectral dimension of input image
- Use hypercube specific loss functions and ensemble strategies to improve computational performance

Proposed SR framework for hyperspectral images

Sparse codes are projected on to the dictionary and are inverted to get the super-resolution estimate



Results of CNN based Super-resolution

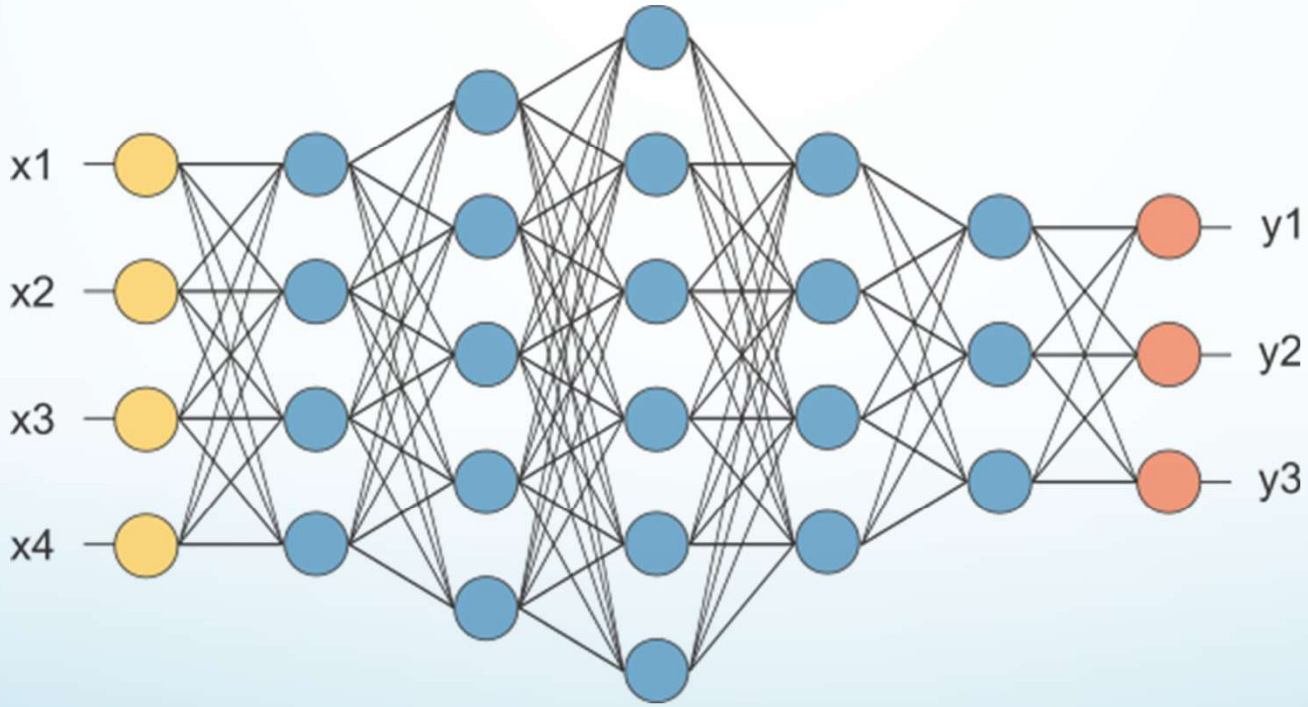


Original image | Simulated coarse image | Proposed FISTA based | Proposed ADMM based

Training image patches used: Indian Pines, KSC, Urban and Cuprite

Time Series Data Analysis

It is difficult to formulate time series using a feed forward network as it becomes computationally intensive



Source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

Feed forward networks vs recurrent networks

Feed forward networks:

- Information only flows one way
- One input pattern produces outputs(responses)
- No sense of time (or memory of previous state)

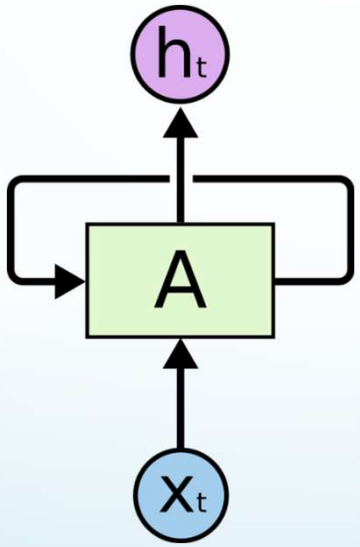
Recurrency

- Nodes connect back to other nodes or themselves
- Information flow is multi-directional
- Has A sense of time and memory of previous state(s)

Source: http://www.khuisf.ac.ir/prof//images/

Recurrent Neural Network

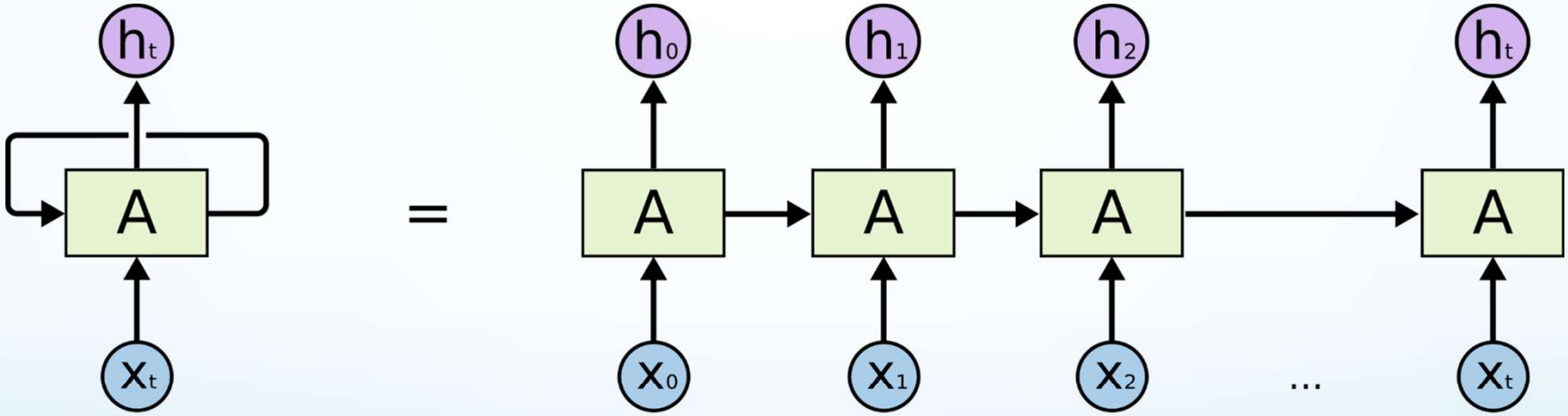
RNN is a chunk of neural network that looks at some sequence of input X and outputs a value H. A loop allows information to be passed from one step of the network to the next.



Source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

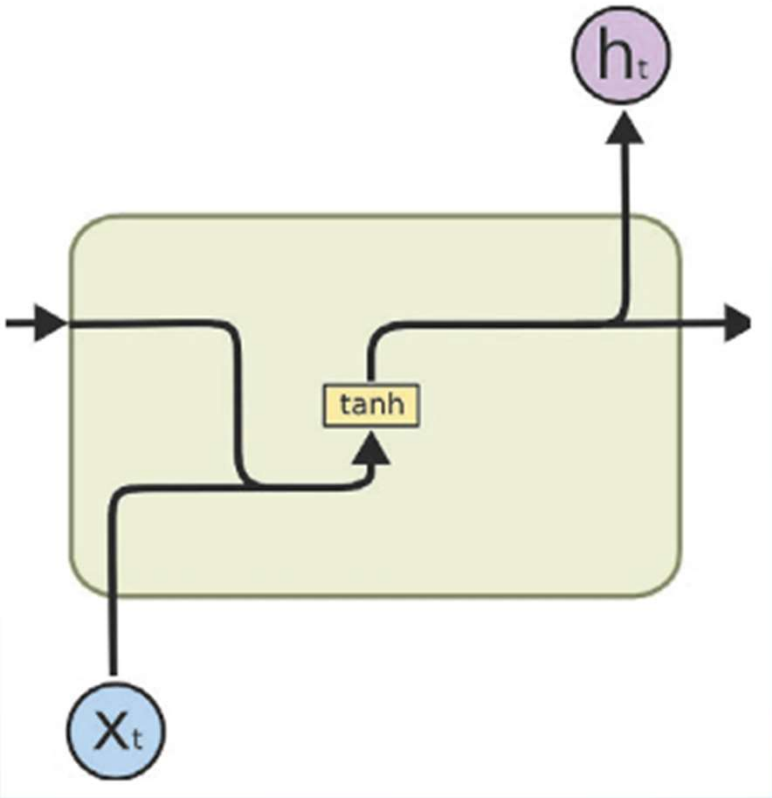
Recurrent Neural Network

Recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.



Source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

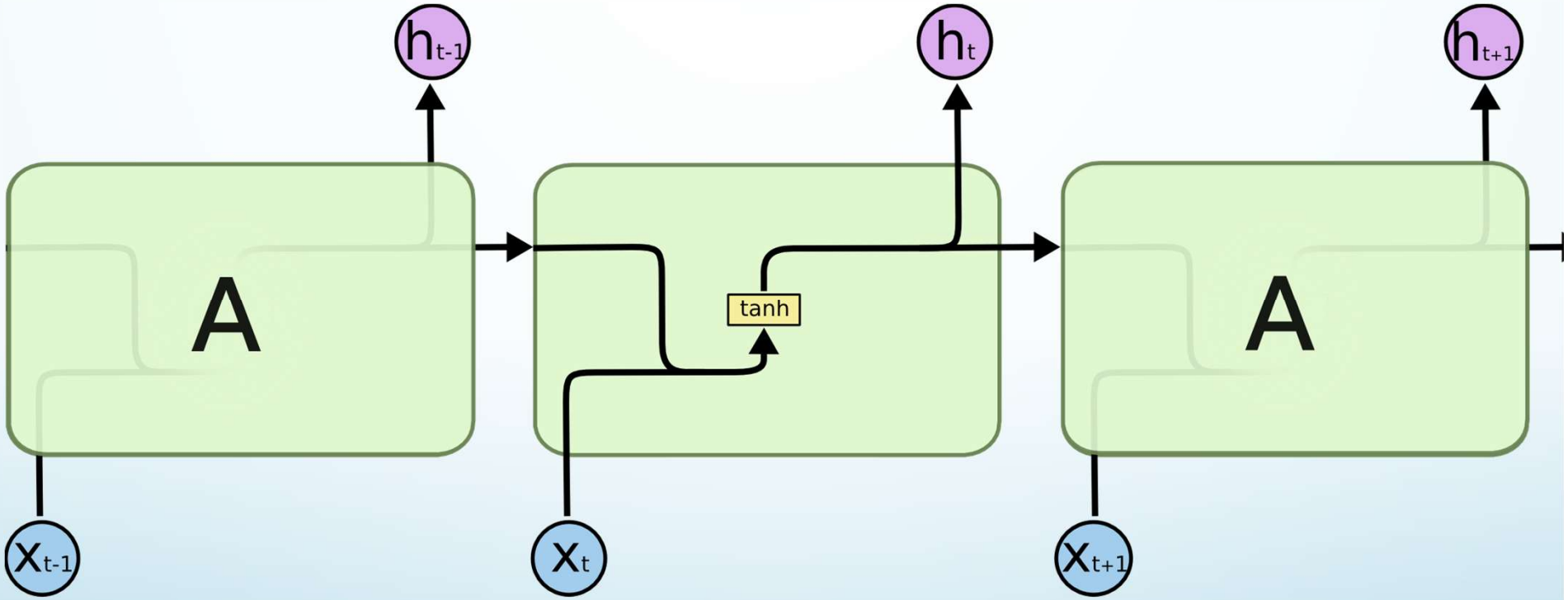
Recurrent Neural Network



$$h_t = \tanh(Ux_t + Wh_{t-1})$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

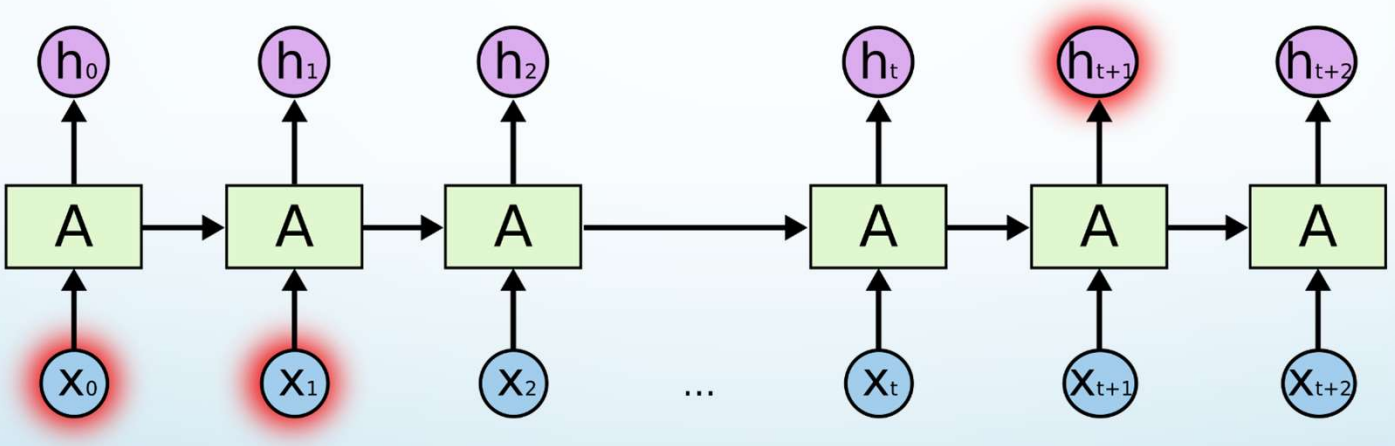
Recurrent Neural Network



Source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

Problems with Recurrent Neural Network

In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Problems with Recurrent Neural Network

- Tends to forget old information.
- Vanishing gradient problem.

LSTMS

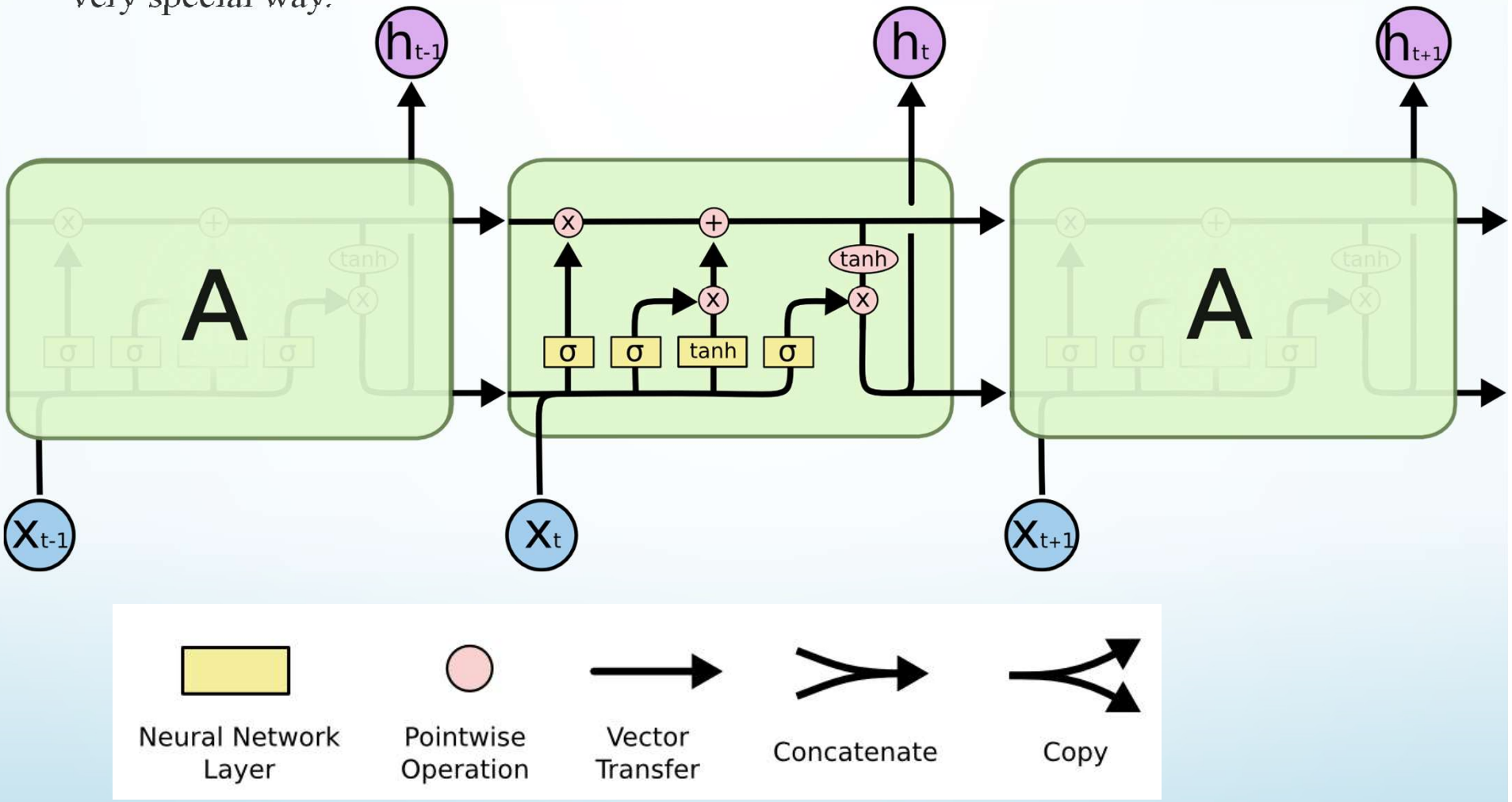
- LSTMs are a class of recurrent networks whose activation at each step depends on that of the previous steps along with the current input.

- Unlike vanilla recurrent models, LSTMs use memory cells to accumulate the state information for modeling long-range dependencies.

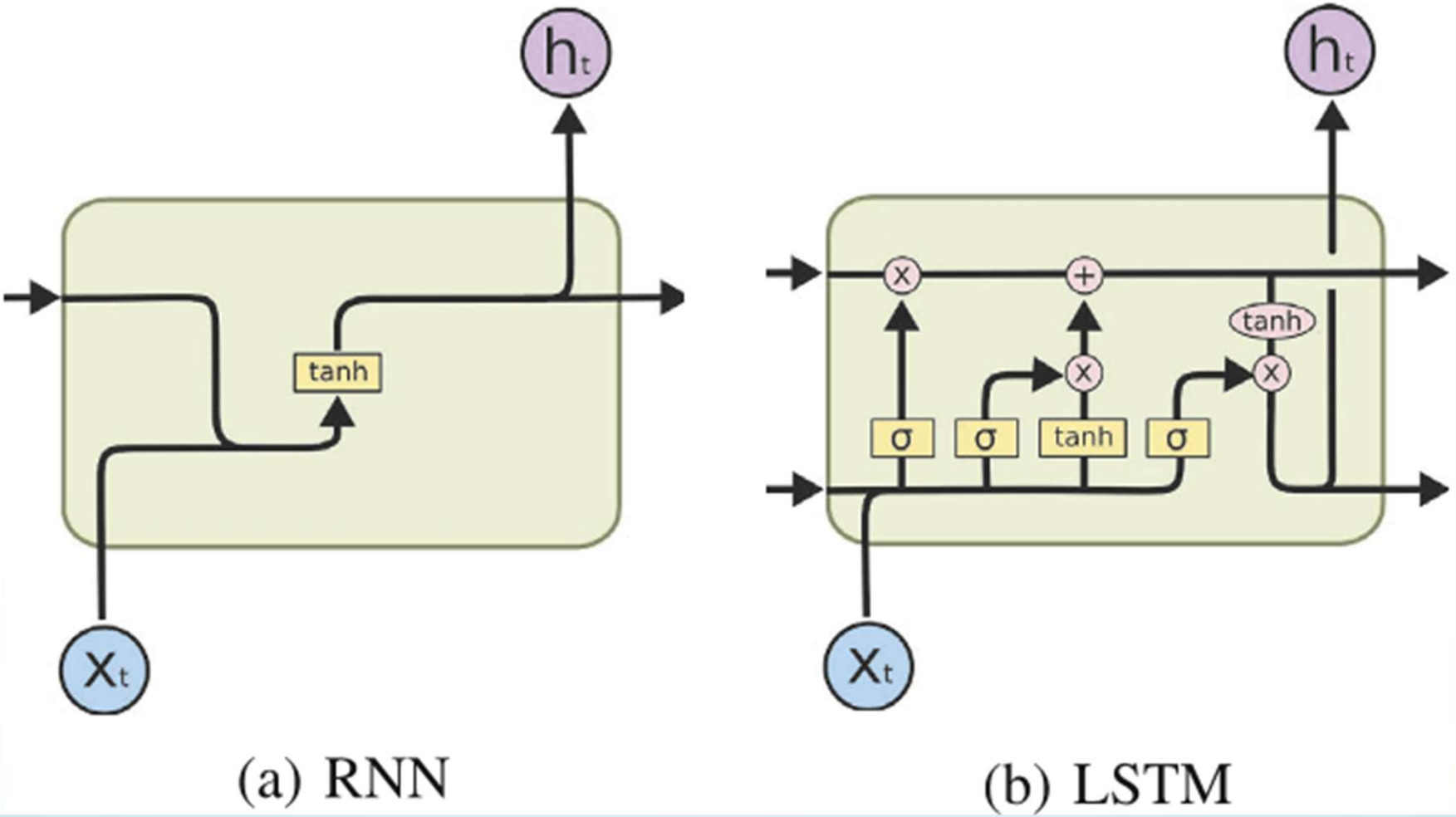
- In addition, various gates are employed to control the information flow from and to the state so that the gradients do not vanish quickly.

LSTMS

LSTMs also have a chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



RNN & LSTM

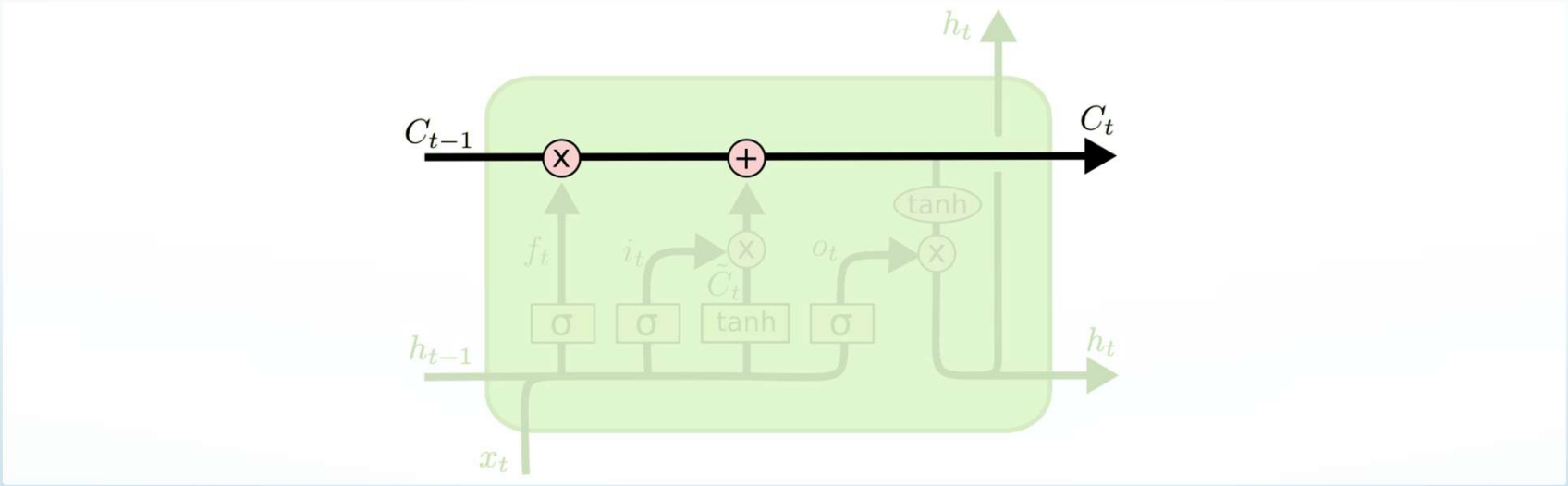


Source: https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

LSTMS

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

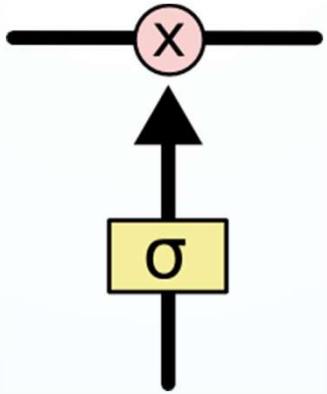


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMS

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

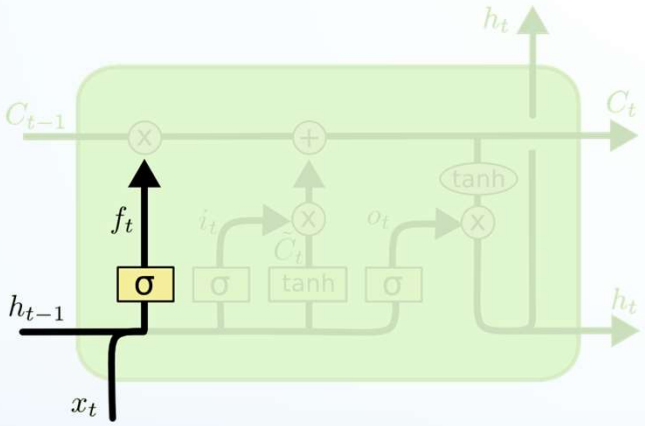


The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

An LSTM has three of these gates, to protect and control the cell state.

LSTMS

Forget gate: what information we're going to throw away from the cell state. It looks at input and hidden state outputs a number between 0 and 1 for each number in the cell state.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

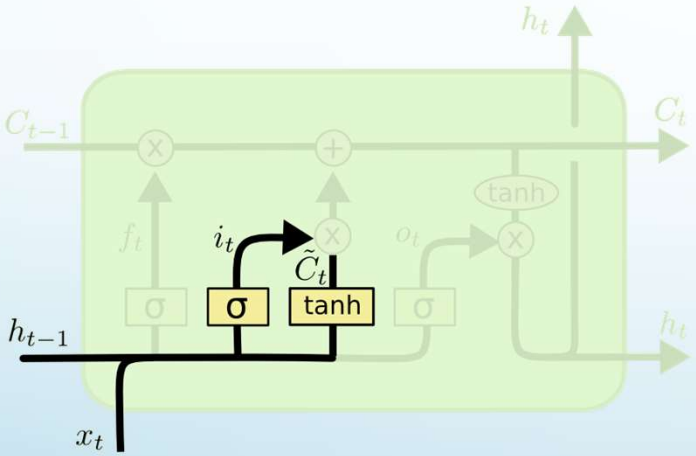
LSTMS

what new information we're going to store in the cell state?

First, a sigmoid layer called the "input gate layer" decides which values we'll update.

Next, a tanh layer creates a vector of new candidate values, that could be added to the state.

In the next step, we'll combine these two to create an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

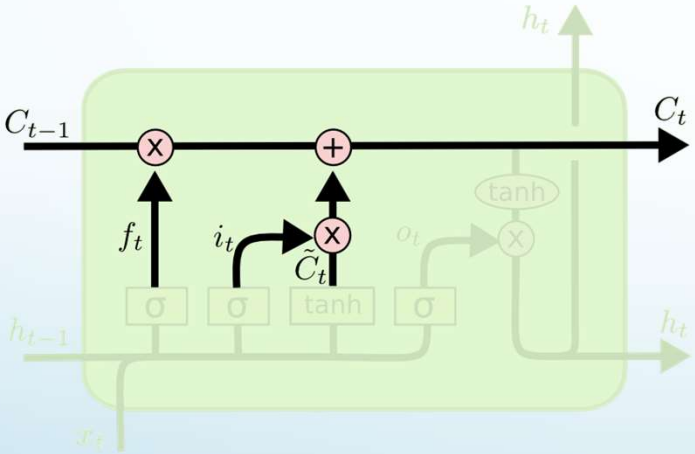
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMS

To update the old cell state into the new cell state.

Multiply the old state by the forget gate output, forgetting the things we decided to forget earlier.

Then add it with the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

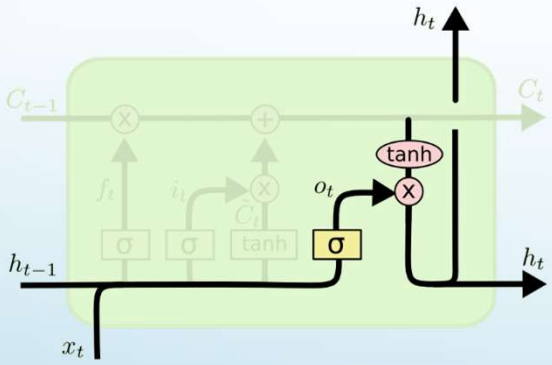
LSTMS

The output will be based on the cell state, but will be a filtered version.

A sigmoid layer which decides what parts of the cell state we're going to output.

The cell state is passed through tanh activation (to push the values to be between −1 and 1)

Multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

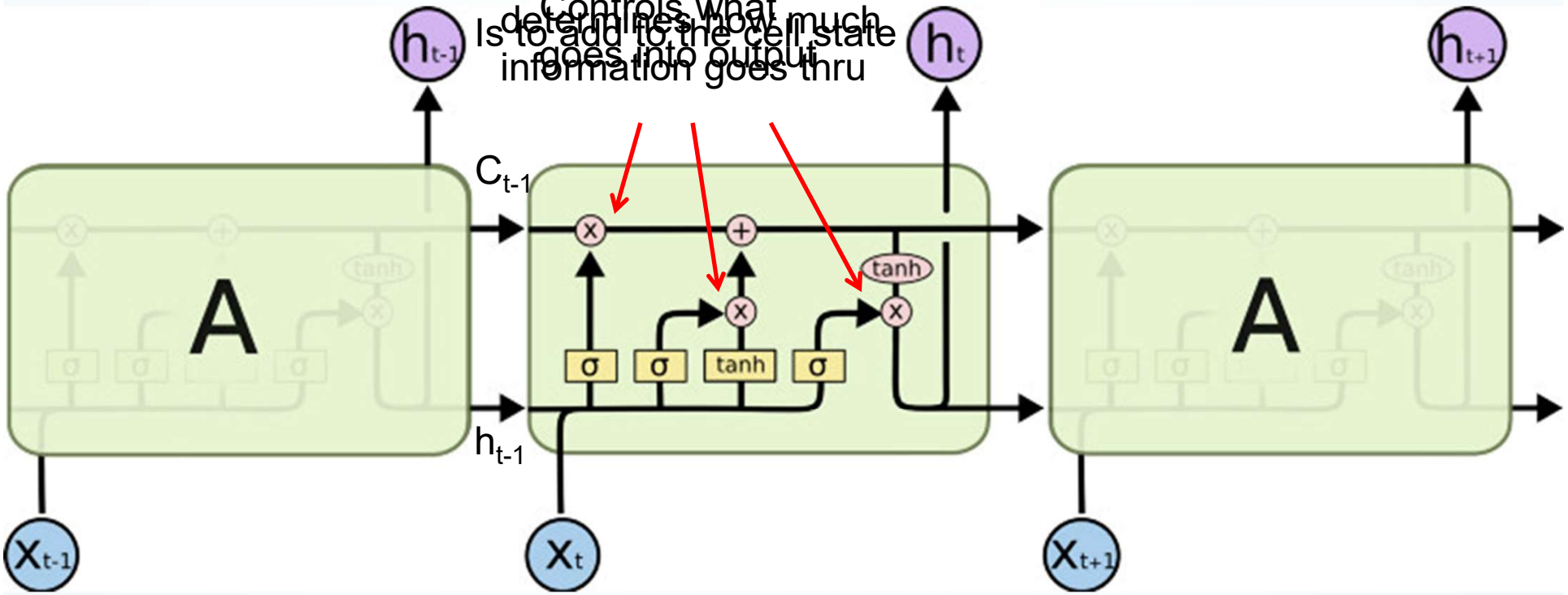


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

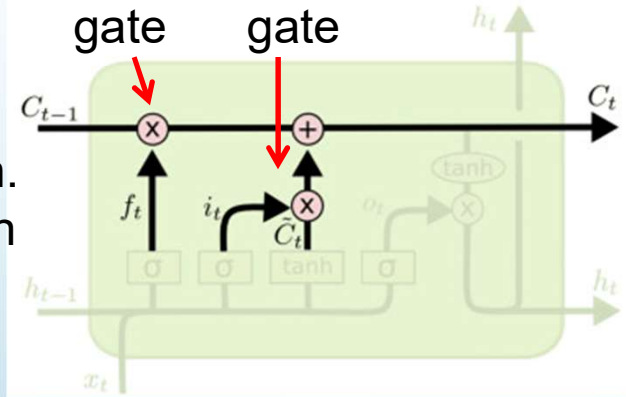
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory



Forget gate

input gate



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

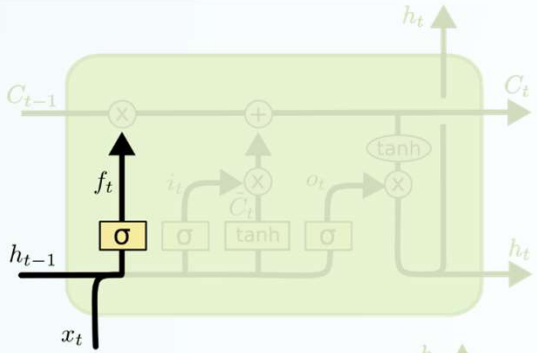
The core idea is this cell state C_t, it is changed slowly, with only minor linear interactions. It is very easy for information to flow along it unchanged.

Why sigmoid or tanh:

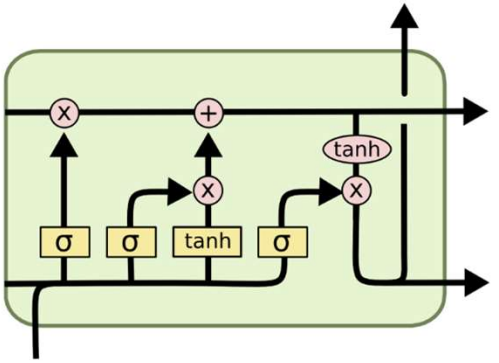
Sigmoid: 0,1 gating as switch.

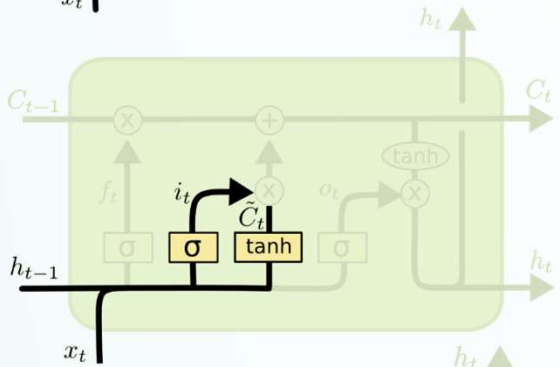
Vanishing gradient problem in LSTM is handled already.

ReLU replaces tanh ok?



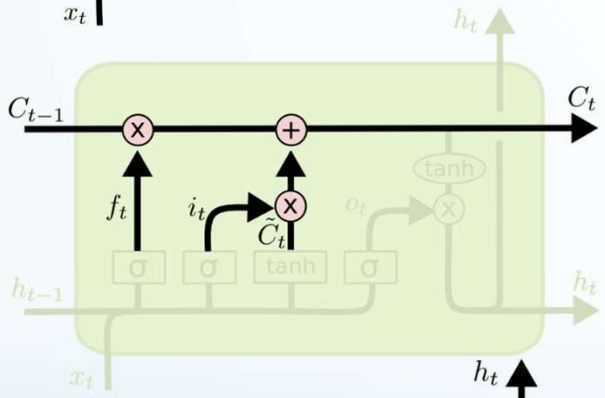
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$





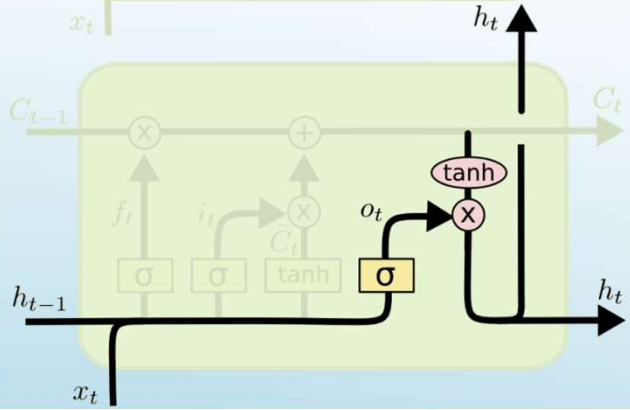
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t decides what component is to be updated.
C'_t provides change contents



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

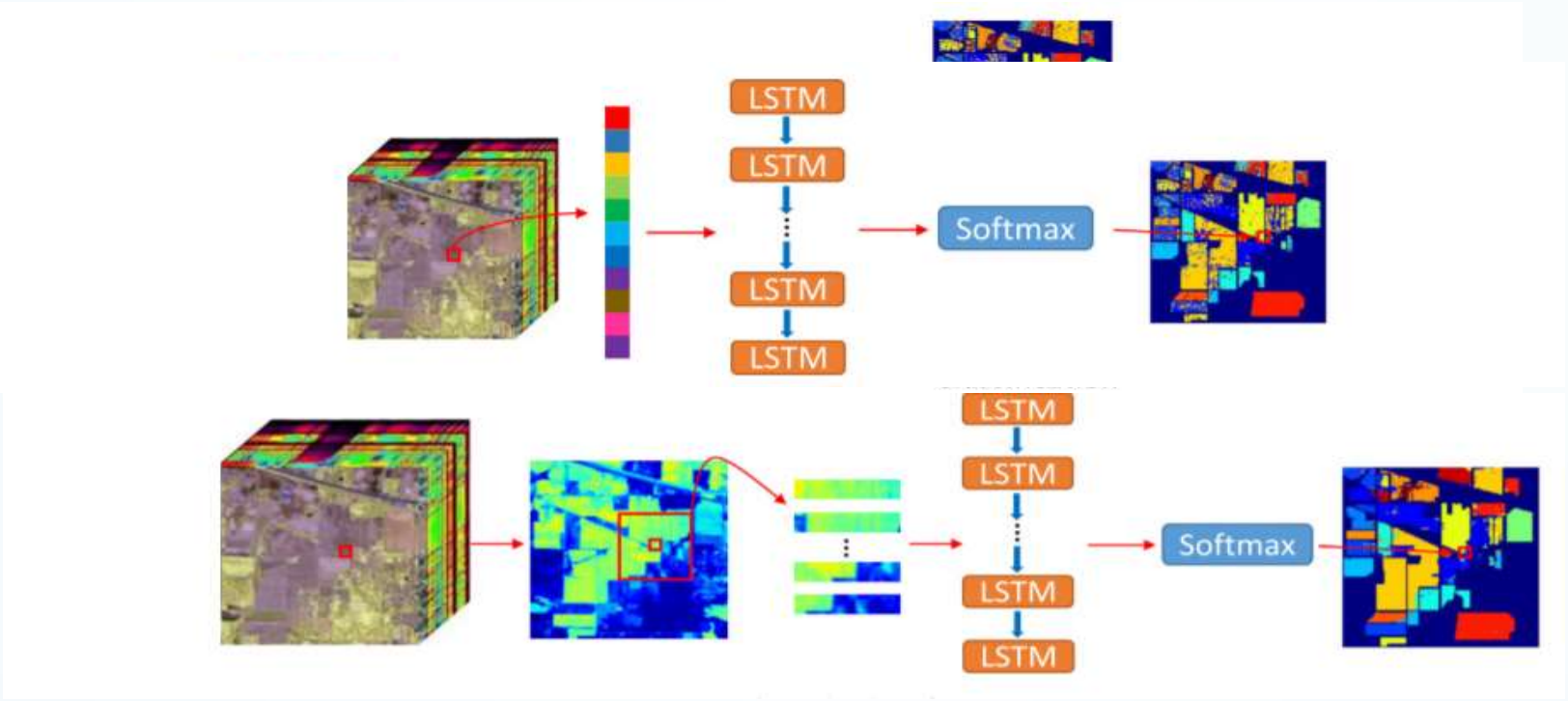
Updating the cell state



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Decide what part of the cell state to output

Long Short Term Memory for hyperspectral Images



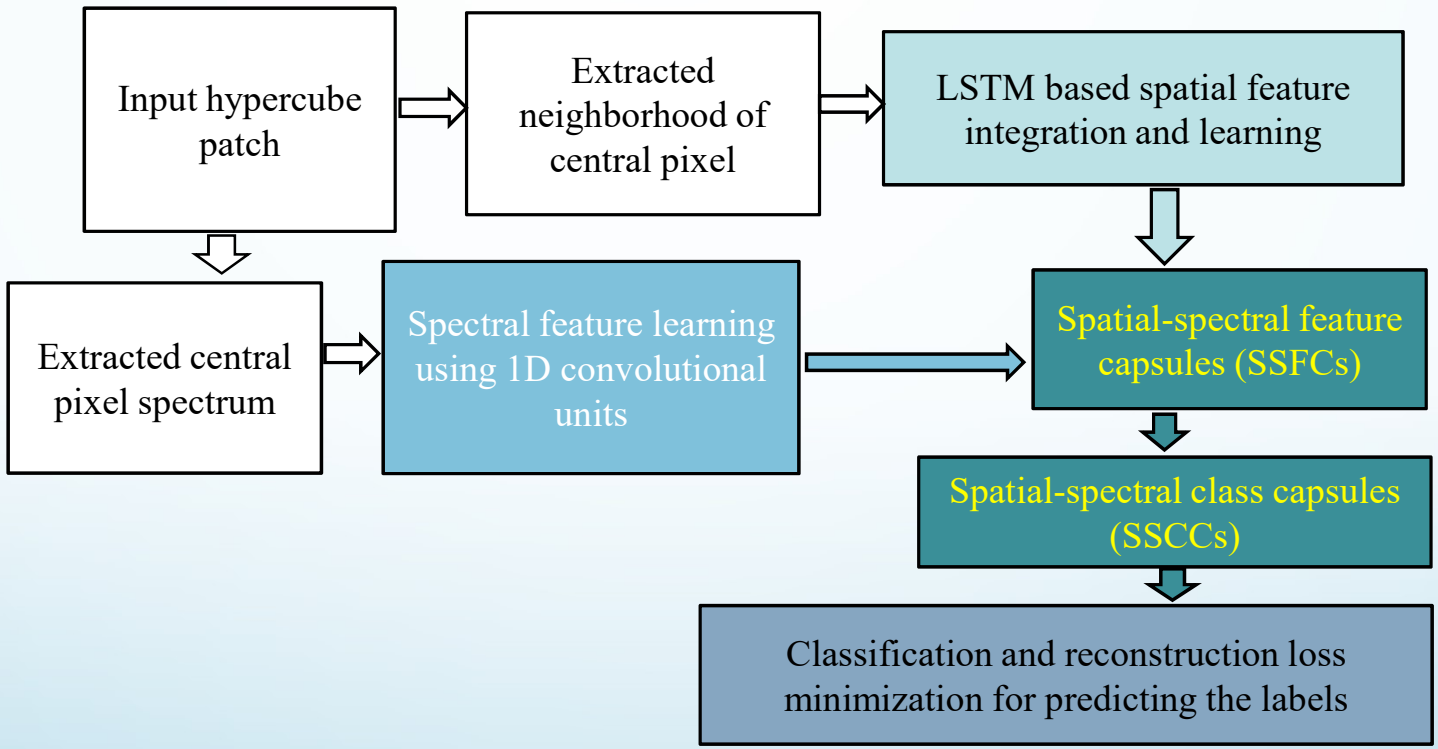
Source: http://h.web.umkc.edu/hangr/papers/Hyperspectral%20image%20classification%20using%20spectral-spatial%20LSTMs.pdf

Capsulenet based spatial-spectral hyperspectral classifier

- Use capsules for better modelling the relative locations and other properties of spatial and spectral features
- Each capsule is a group of neurons whose activity vector represents a specific type of spectral or spatial feature
- Spatial features (from LSTM hidden state) and spectral features (from convolutional units) form capsules.
- Spatial and spectral features are integrated at capsule-level, making the approach scalable towards increase in spectral dimension

Capsule-based spatial-spectral hyperspectral classifier

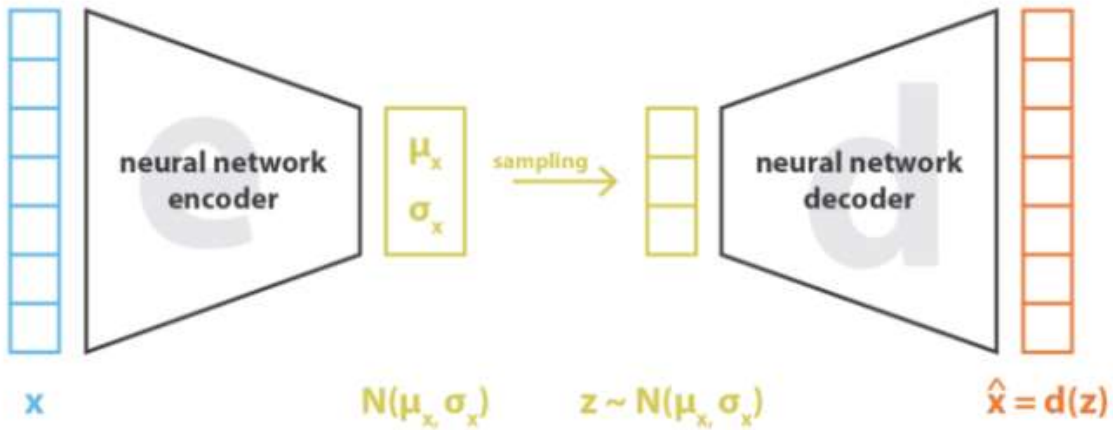
- Spatial features (from LSTM hidden state) and spectral features (from convolutional units) form capsules.
- Modeling of features and their orientations improve generalizability



Generator Networks

- Encoder decoder networks for learning representations
- Can we make the network to generate new images?
- Consider only the decoder portion, given a latent code it can generate an image (if it's a variational encoder input can be sampled from a Gaussian)
- Generative Adversarial Networks are also widely employed where a generator attempt to fool a discriminator

Variational encoders as Generator Networks



$$\text{loss} = \|x - \hat{x}\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

In variational autoencoders, the loss function is composed of a reconstruction term (that makes the encoding-decoding scheme efficient) and a regularisation term (that makes the latent space regular).

Conclusion

ML algorithms such as Support Vector Machine, Random forest, ANN, Self Organising Map, Fuzzy C-Means are popular for different analysis related to hyperspectral image processing.

Among ML techniques, deep learning algorithms have given state-of-the-art results for several automation approaches related to hyperspectral analyses.

Capsulenets are being widely employed for classification of hyperspectral images

Explainable Artificial Intelligence is being investigated to develop interpretable models.

ML, in particular DL, is seen revolutionizing the different aspects of hyperspectral image analysis.

Selected References

Altmann, Y., Halimi, A., Dobigeon, N., & Tourneret, J. Y. (2011). Supervised nonlinear spectral unmixing using a polynomial post nonlinear model for hyperspectral imagery. In ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings (pp. 1009–1012). https://doi.org/10.1109/ICASSP.2011.5946577

Balducci, F., Impedovo, D., & Pirlo, G. (2018). Machine Learning Applications on Agricultural Datasets for Smart Farm Enhancement. Machines, 6(3), 38. https://doi.org/10.3390/machines6030038

Chai, X., Gu, H., Li, F., Duan, H., Hu, X., & Lin, K. (2020). Deep learning for irregularly and regularly missing data reconstruction. Scientific Reports, 10(1), 1–18. https://doi.org/10.1038/s41598-020-59801-x

Charte, D., Charte, F., del Jesus, M. J., & Herrera, F. (2020). An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges. Neurocomputing, 404, 93–107. https://doi.org/10.1016/j.neucom.2020.04.057

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2019). Explaining Explanations: An Overview of Interpretability of Machine Learning.

Herrmann, I., Pimstein, A., Karnieli, A., Cohen, Y., Alchanatis, V., & Bonfil, D. J. (2011). LAI assessment of wheat and potato crops by VENμS and Sentinel-2 bands. Remote Sensing of Environment, 115(8), 2141–2151. https://doi.org/10.1016/j.rse.2011.04.018

Ichoku, C., & Karnieli, A. (1996). A Review of Mixture Modeling Techniques for Sub-Pixel Land Cover Estimation. Remote Sensing Reviews, 13(3–4), 161–186. https://doi.org/10.1080/02757259609532303

Ihuoma, S. O., & Madramootoo, C. A. (2017, September 1). Recent advances in crop water stress detection. Computers and Electronics in Agriculture. Elsevier B.V. https://doi.org/10.1016/j.compag.2017.07.026

Selected References

Kussul, N., Lavreniuk, M., Skakun, S., & Shelestov, A. (2017). Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data. IEEE Geoscience and Remote Sensing Letters, 14(5), 778–782. https://doi.org/10.1109/LGRS.2017.2681128

Ma, L., Liu, Y., Zhang, X., Ye, Y., Yin, G., & Johnson, B. A. (2019, June 1). Deep learning in remote sensing applications: A meta-analysis and review. ISPRS Journal of Photogrammetry and Remote Sensing. Elsevier B.V. https://doi.org/10.1016/j.isprsjprs.2019.04.015

Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7(September). https://doi.org/10.3389/fpls.2016.01419

Nigam, R., Tripathy, R., Dutta, S., Bhagia, N., Nagori, R., Chandrasekar, K., et al. (n.d.). Crop type discrimination and health assessment using hyperspectral imaging. https://aviris-ng.jpl.nasa. Accessed 17 January 2021

Qian, B., Su, J., Wen, Z., Jha, D. N., Li, Y., Guan, Y., et al. (2020). Orchestrating the Development Lifecycle of Machine Learning-based IoT Applications: A Taxonomy and Survey. ACM Computing Surveys, 53(4), 82. https://doi.org/10.1145/3398020

Saleem, M. H., Potgieter, J., & Arif, K. M. (2019, November 1). Plant disease detection and classification by deep learning. Plants. MDPI AG. https://doi.org/10.3390/plants8110468

Saleem, M. H., Potgieter, J., & Arif, K. M. (2019, November 1). Plant disease detection and classification by deep learning. Plants. MDPI AG. https://doi.org/10.3390/plants8110468

Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., & Stefanovic, D. (2016). Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. Computational Intelligence and Neuroscience, 2016. https://doi.org/10.1155/2016/3289801

Selected References

Toda, Y., Okura, F., Ito, J., Okada, S., Kinoshita, T., Tsuji, H., & Saisho, D. (2020). Training instance segmentation neural network with synthetic datasets for crop seed phenotyping. Communications Biology, 3(1), 1–12. https://doi.org/10.1038/s42003-020-0905-5

Vikranth Jeyakumar, J., Noor, J., Cheng, Y.-H., Garcia, L., & Srivastava, M. (2020). How Can I Explain This to You? An Empirical Study of Deep Neural Network Explanation Methods. Advances in Neural Information Processing Systems (Vol. 33). https://github.com/nesl/Explainability-Study. Accessed 14 November 2020

Vindya, N. D., & Vedamurthy, H. K. (2020). Machine learning algorithm in smart farming for crop identification. In Advances in Intelligent Systems and Computing (Vol. 1108 AISC, pp. 19–25). Springer. https://doi.org/10.1007/978-3-030-37218-7_3

Wang, L., Shi, C., Diao, C., Ji, W., & Yin, D. (2016, August 17). A survey of methods incorporating spatial information in image classification and spectral unmixing. International Journal of Remote Sensing. Taylor and Francis Ltd. https://doi.org/10.1080/01431161.2016.1204032

Yuan, Q., Shen, H., Li, T., Li, Z., Li, S., Jiang, Y., et al. (2020). Deep learning in environmental remote sensing: Achievements and challenges. Remote Sensing of Environment, 241, 111716. https://doi.org/10.1016/j.rse.2020.111716


THANKS

a. Landsat ETM+

b. ATLAS

c. QuickBird