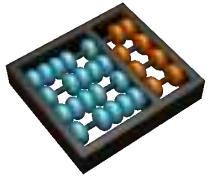
Convolutional Neural Networks

Alexandre Xavier Falcão

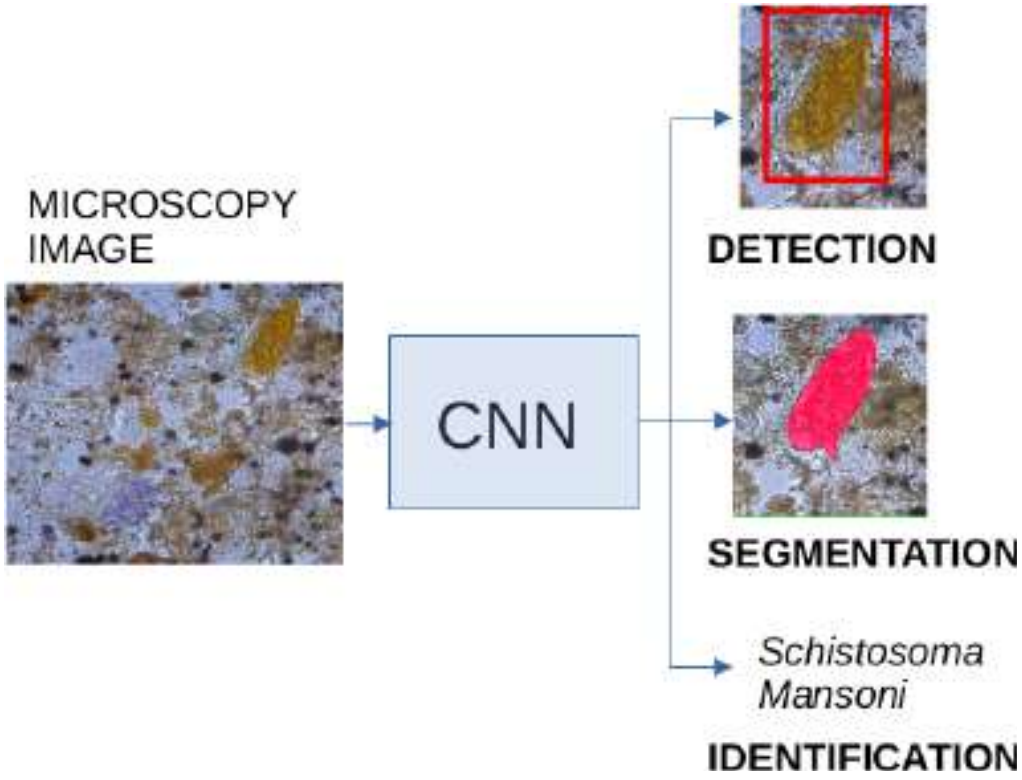
Laboratory of Image Data Science
Institute of Computing — University of Campinas

afalcao@ic.unicamp.br
lids.ic.unicamp.br



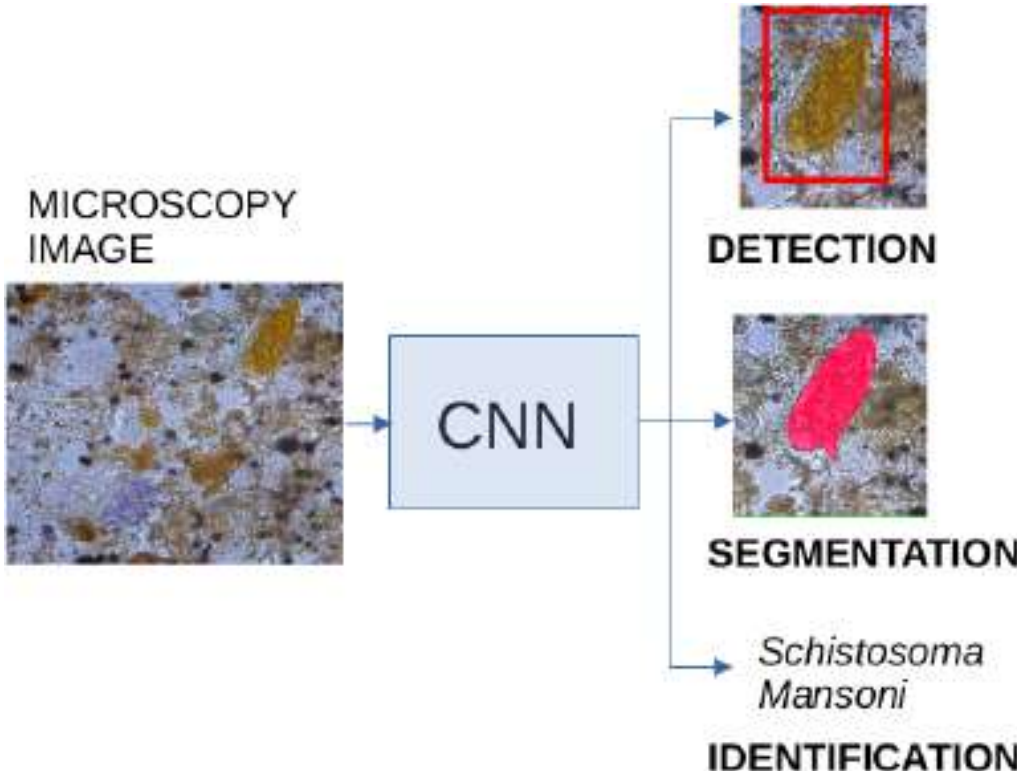
What are our objectives?

First, learn how Convolutional Neural Networks (CNNs) work for image segmentation (detection) and identification [Book20].



What are our objectives?

First, learn how Convolutional Neural Networks (CNNs) work for image segmentation (detection) and identification [Book20].



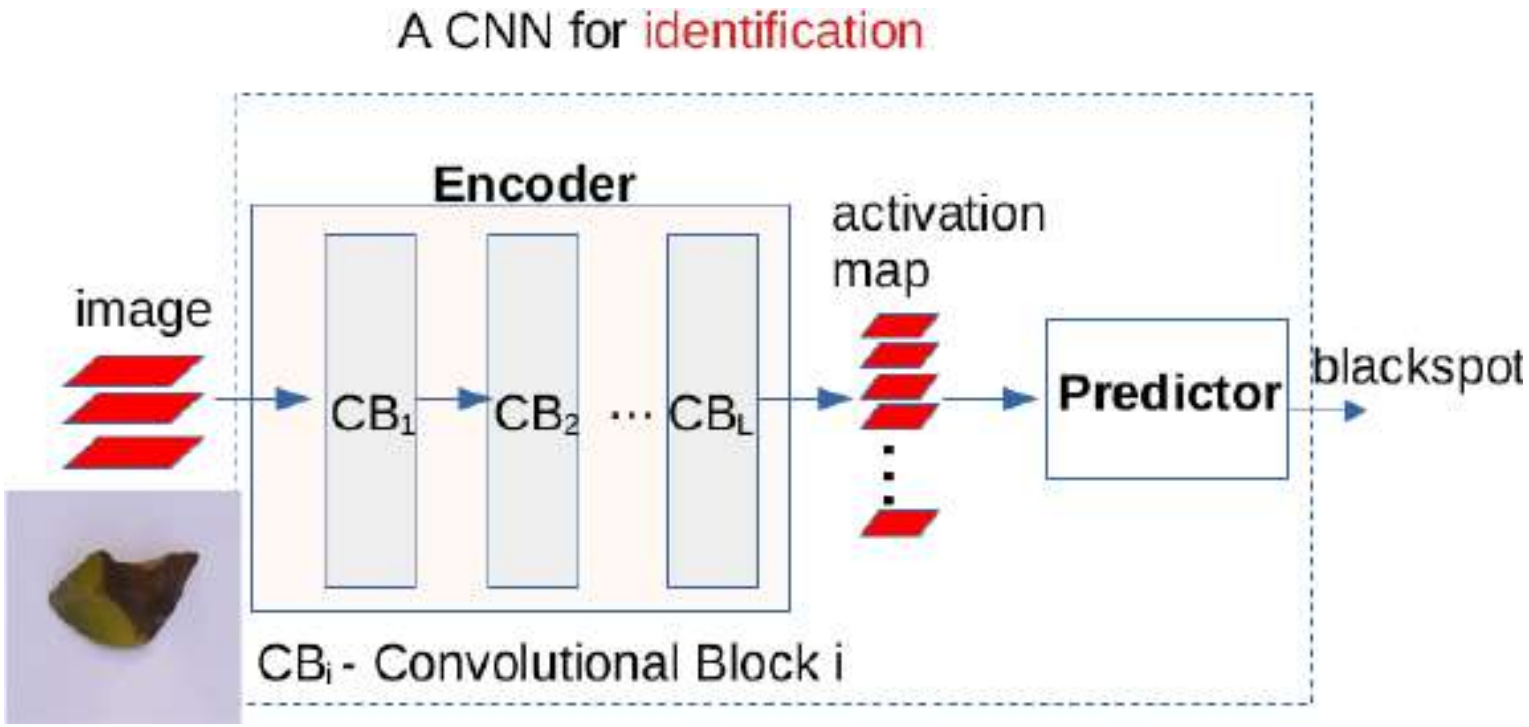
Second, learn how to build explainable encoders for such problems.

Agenda

- What are CNNs?
- Basic definitions and image processing operations.
- CNNs for image identification and segmentation.
- Hands-on – How to design and evaluate CNNs using pytorch.
- Explainable encoders for identification and segmentation.
- Hands-on – CNNs with explainable encoders using pytorch.
- Final remarks.

What are CNNs?

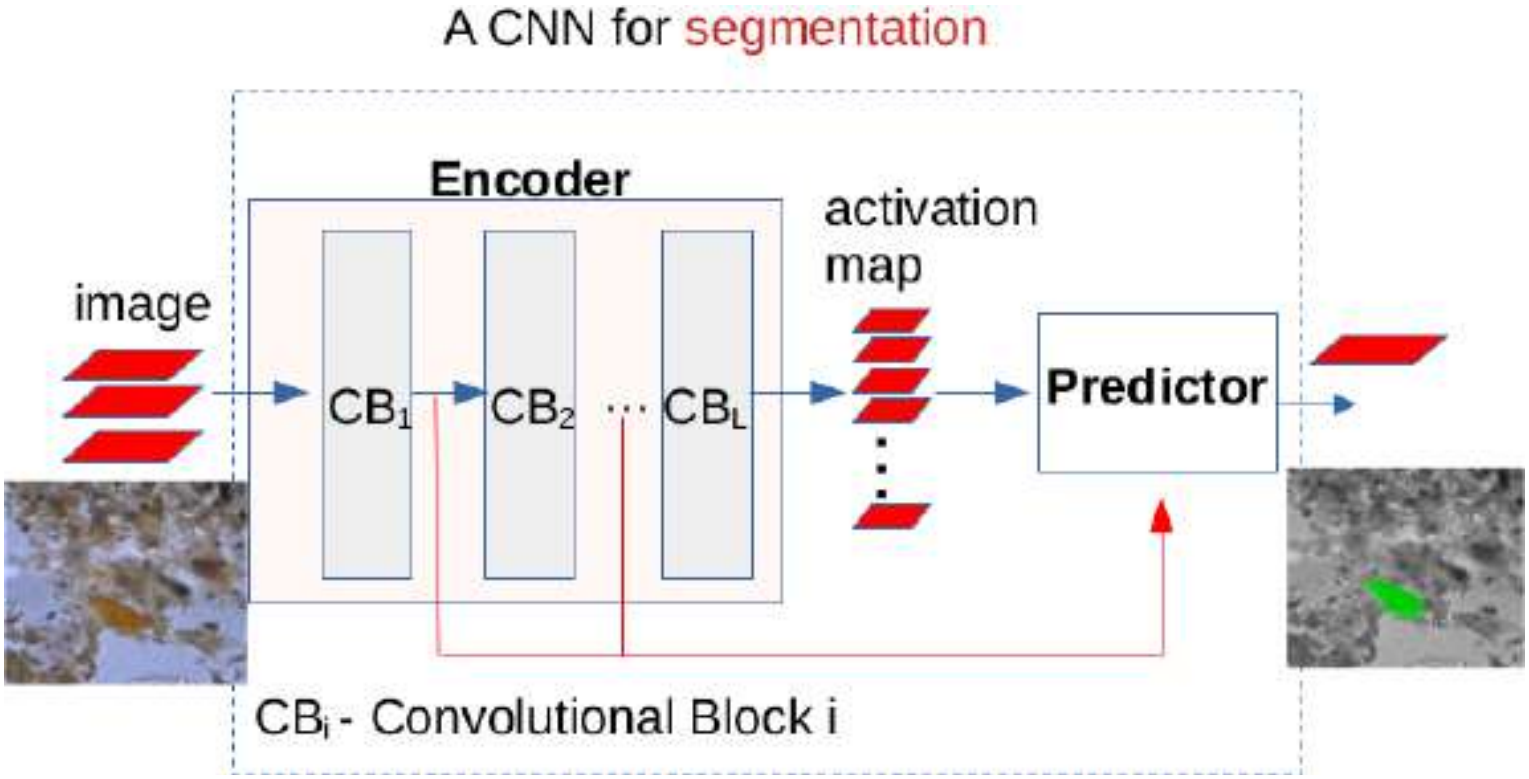
CNNs are neural networks composed by an encoder for feature extraction and a predictor for classification/regression.



The predictor may use the activation map from multiple convolutional blocks, which is usually the case for segmentation.

What are CNNs?

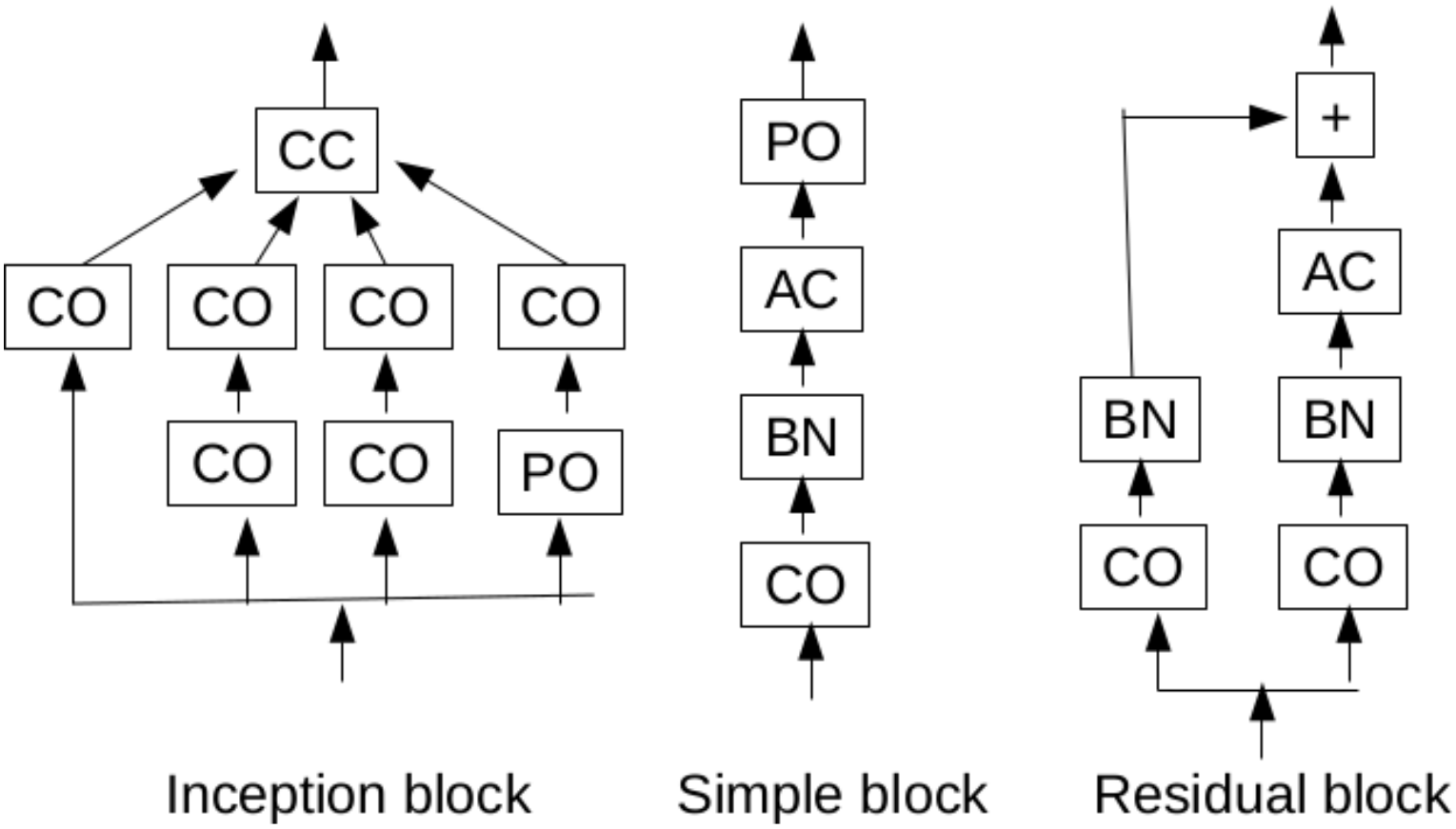
CNNs are neural networks composed by an encoder for feature extraction and a predictor for classification/regression.



The predictor may use the activation map from multiple convolutional blocks, which is usually the case for segmentation.

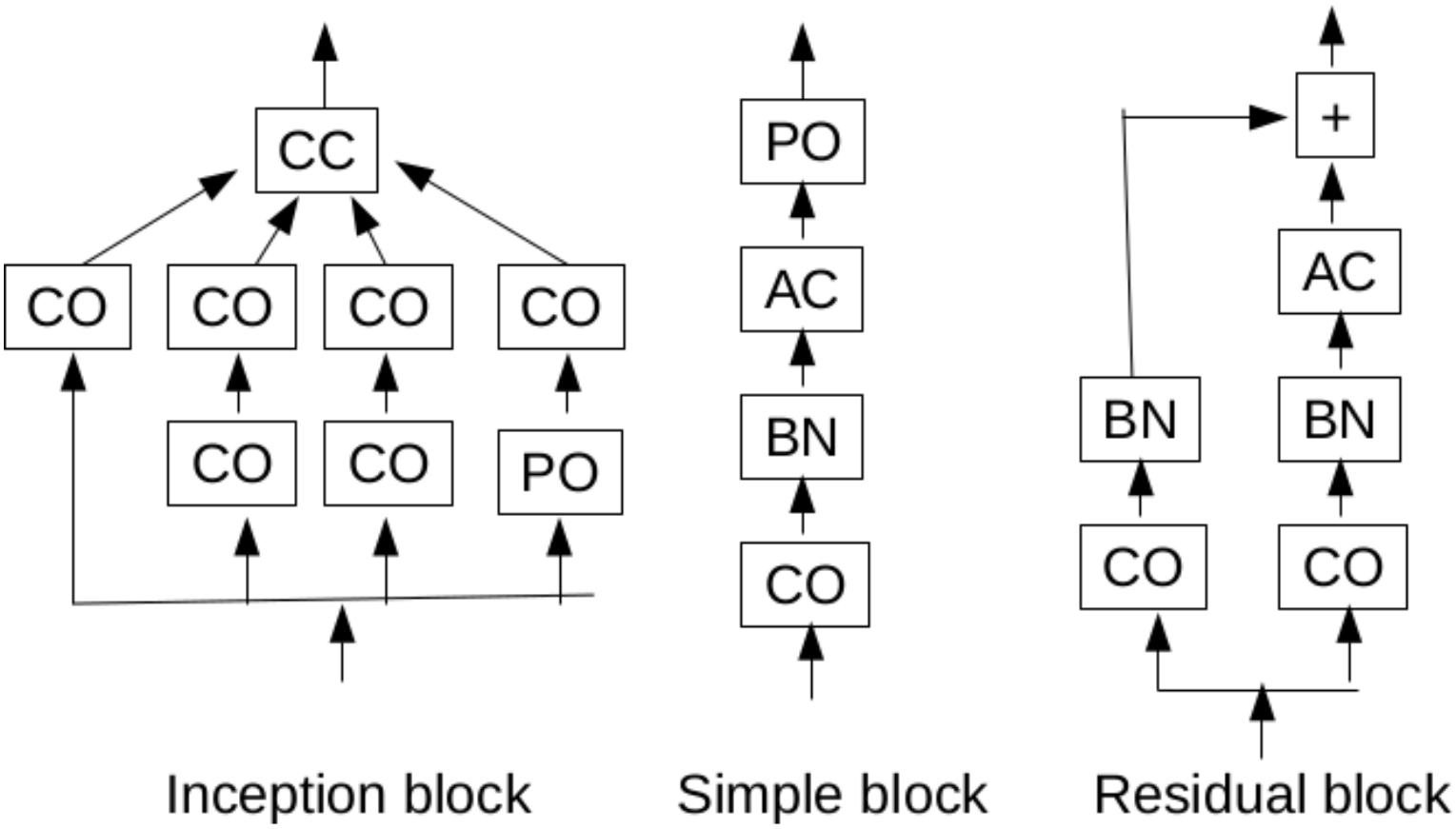
What is the content of a convolutional block?

Convolutional blocks may contain image operations such as convolution (CO), batch normalization (BN), activation (AC), pooling (PO), concatenation (CC), adition (+), etc.



What is the content of a convolutional block?

Convolutional blocks may contain image operations such as convolution (CO), batch normalization (BN), activation (AC), pooling (PO), concatenation (CC), adition (+), etc.



Inception block Simple block Residual block

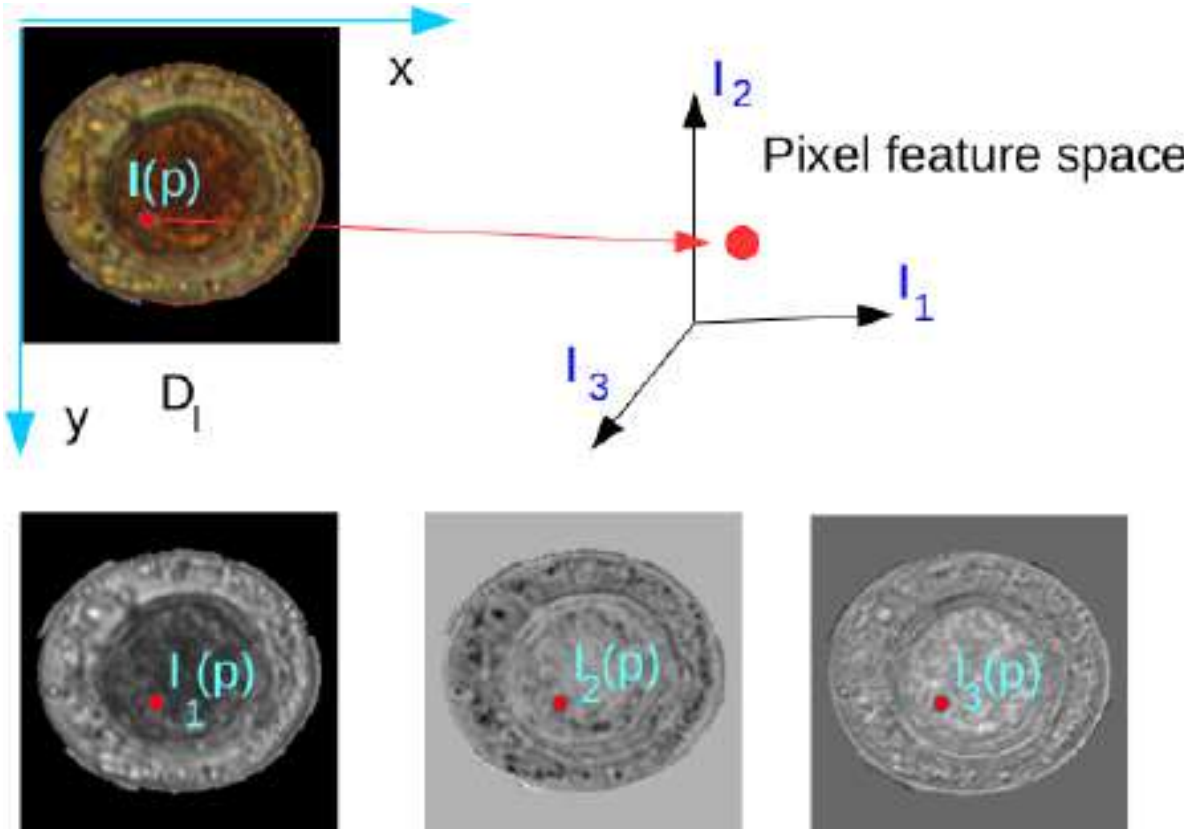
What are those image processing operations?

Basic definitions and image processing operations

- Multichannel image.
- Adjacency relation and image patch.
- Kernel and kernel bank.
- Convolution, bias and activation (one perceptron per pixel).
- Other image processing operations.

Multichannel image

A 2D multichannel image $\hat{I}$ is a pair $(D_I, \mathbf{I})$ in which $\mathbf{I}(p) \in \Re^m$ assigns m scalar values to each pixel $p \in D_I \subset \mathcal{Z}^2$.



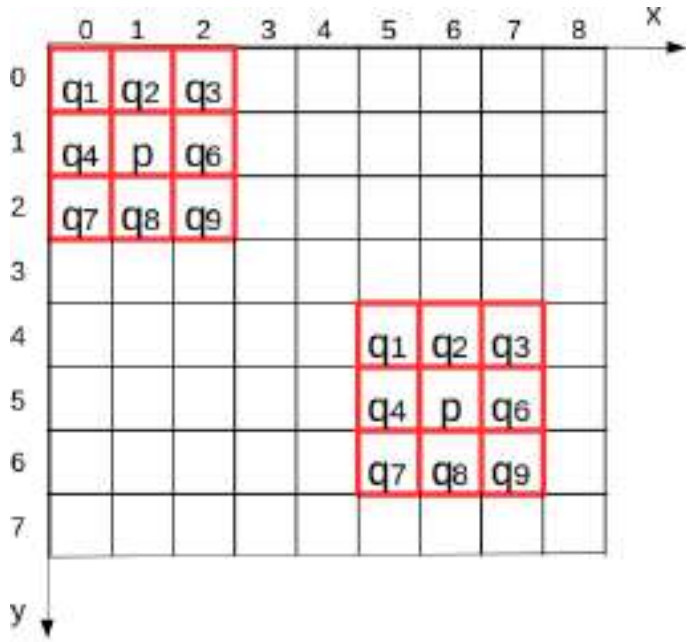
For $m = 3$ channels, each pixel p is represented by a point $\mathbf{I}(p) = (I_1(p), I_2(p), I_3(p)) \in \Re^3$.

Adjacency relation

An adjacency relation A is a binary relation that considers the distance between pixels. CNNs adopt rectangular relations in 2D.

$$A(p) = \{q \in D_I | \quad |x_q - x_p| \leq \frac{w}{2} \quad and \quad |y_q - y_p| \leq \frac{h}{2}\}$$

where $p = (x_p, y_p) \in D_I \subset \mathcal{Z}^2$.

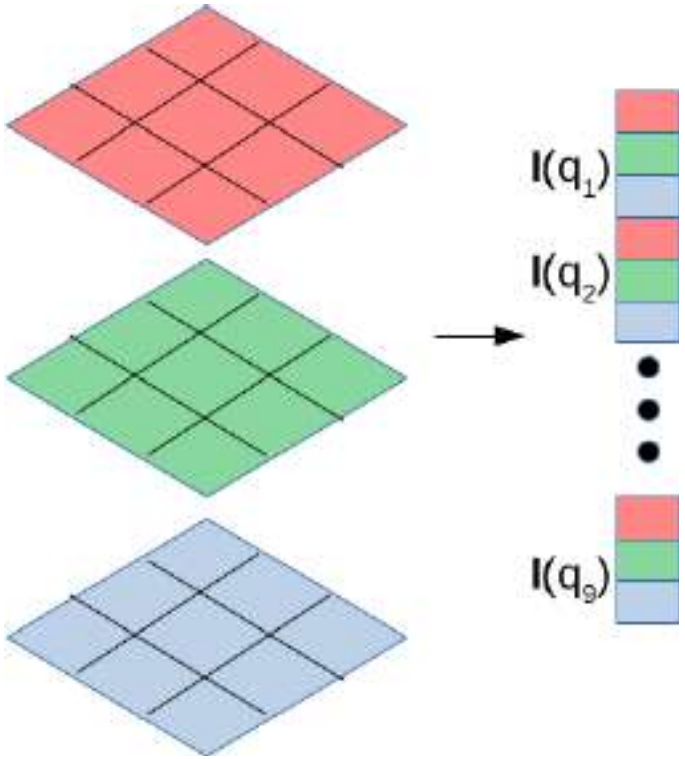


Two examples for pixels p, $p = (1, 1)$ and $p = (6, 5)$, with $w = h = 3$, $A(p) = \{q_1, q_2, \ldots, q_9\}$ where $q_5 = p$.

Image patch

Let $|A(p)| = n$ be a fixed number for any $p \in D_I$ (e.g., $n = w \times h$ when A is rectangular).

The concatenation of the attributes $\mathbf{I}(q_1) \bullet \mathbf{I}(q_2) \bullet \ldots \bullet \mathbf{I}(q_n)$ of the adjacent pixels of p defines an image patch $\mathbf{X}(p) \in \Re^{n \times m}$ centered at p.



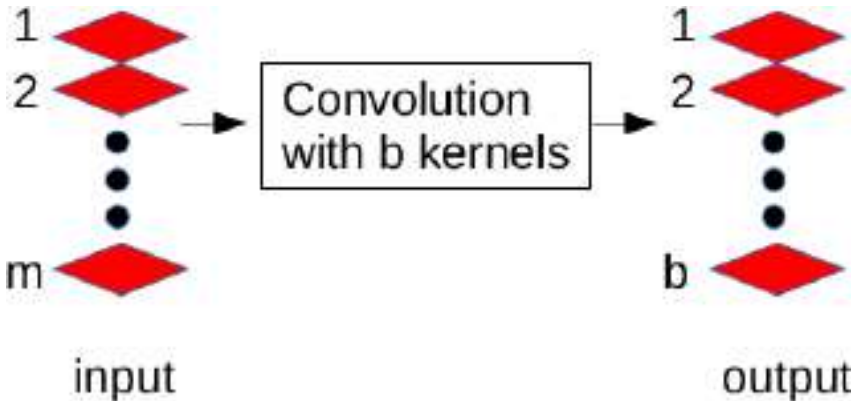
Kernel and kernel bank

- A kernel is a weight vector $\mathbf{W} \in \Re^{n \times m}$ with the same size of the image patches. A kernel bank is just a set of kernels.

Kernel and kernel bank

- A kernel is a weight vector $\mathbf{W} \in \Re^{n \times m}$ with the same size of the image patches. A kernel bank is just a set of kernels.

- The convolution between an input image (activation map), with m channels, and a kernel bank with b kernels, with m channels each, outputs an activation map with b channels.

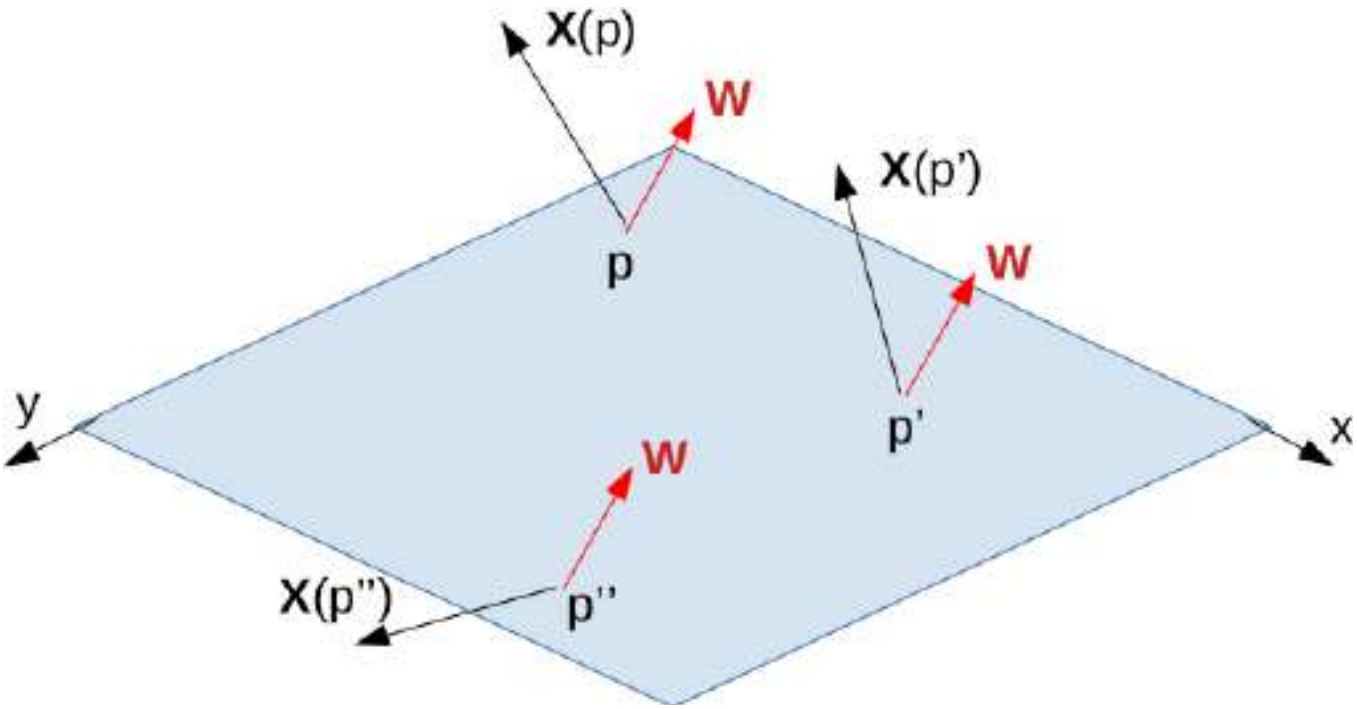


Convolution

The convolution between a multichannel image $\hat{I} = (D_I, \mathbf{I})$ and a kernel $\mathbf{W}$ with the same size of the patches $\mathbf{X}(p)$ defined by some adjacency relation A generates a single-channel activation map $\hat{J} = (D_I, J)$ such that

$$J(p) \quad = \quad \langle \mathbf{X}(p), \mathbf{W} \rangle$$

as $\mathbf{W}$ slides over the image.



Convolution

When convolving $\hat{I} = (D_I, \mathbf{I})$ with a bank $\{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_b\}$ of b kernels, the output activation map $\hat{J} = (D_I, \mathbf{J})$ has b channels, such that

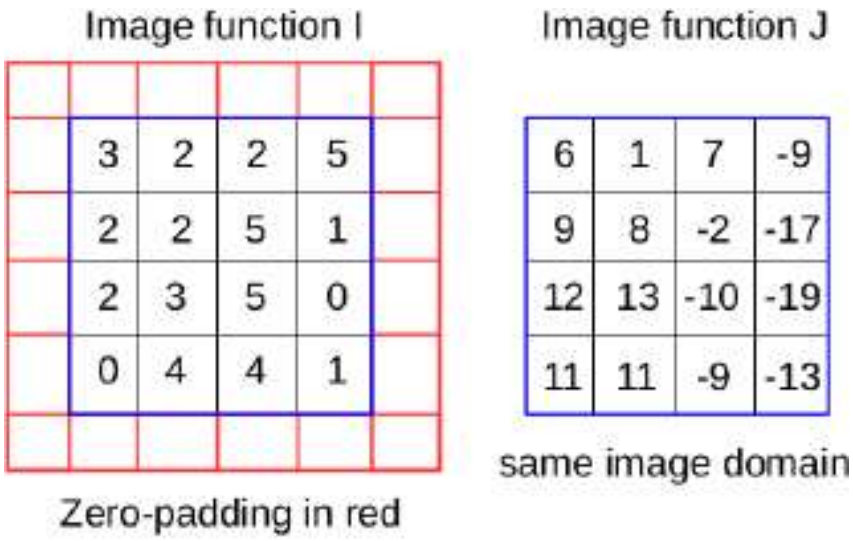
$$J_k(p) \quad = \quad \langle \mathbf{X}(p), \mathbf{W}_k \rangle$$

for $k = 1, 2, \ldots, b$.

$\mathbf{J}(p) = (J_1(p), J_2(p), \ldots, J_b(p)) \in \Re^b$ then contains the activation values at p.

Convolution

Zero-padding is usually adopted when defining $\mathbf{X}(p)$. Kernel and patch must have the same shape before their vectorization. This example shows the case the input image and kernel have single channels.



Activation

Among several activation functions, the Rectified Linear Unit (ReLU) is the most popular.

Activation

Among several activation functions, the Rectified Linear Unit (ReLU) is the most popular.

From the multichannel output $\hat{J} = (D_I, \mathbf{J})$ of a convolution, ReLU creates a map $\hat{R} = (D_I, \mathbf{R})$, $\mathbf{R}(p) = (R_1(p), R_2(p), \ldots, R_b(p))$,

$$R_k(p) \quad = \quad \max\{0, J_k(p)\},$$

for $p \in D_I$ and $k \in [1, b]$.

Activation

Among several activation functions, the Rectified Linear Unit (ReLU) is the most popular.

From the multichannel output $\hat{J} = (D_I, \mathbf{J})$ of a convolution, ReLU creates a map $\hat{R} = (D_I, \mathbf{R})$, $\mathbf{R}(p) = (R_1(p), R_2(p), \ldots, R_b(p))$,

$$R_k(p) \quad = \quad \max\{0, J_k(p)\},$$

for $p \in D_I$ and $k \in [1, b]$.

Image function J

6	1	7	-9
9	8	-2	-17
12	13	-10	-19
11	11	-9	-13

Image function R

6	1	7	0
9	8	0	0
12	13	0	0
11	11	0	0

Convolution followed by activation

Kernel 3 x 3 x 1

-1	0	1
-2	0	2
-1	0	1

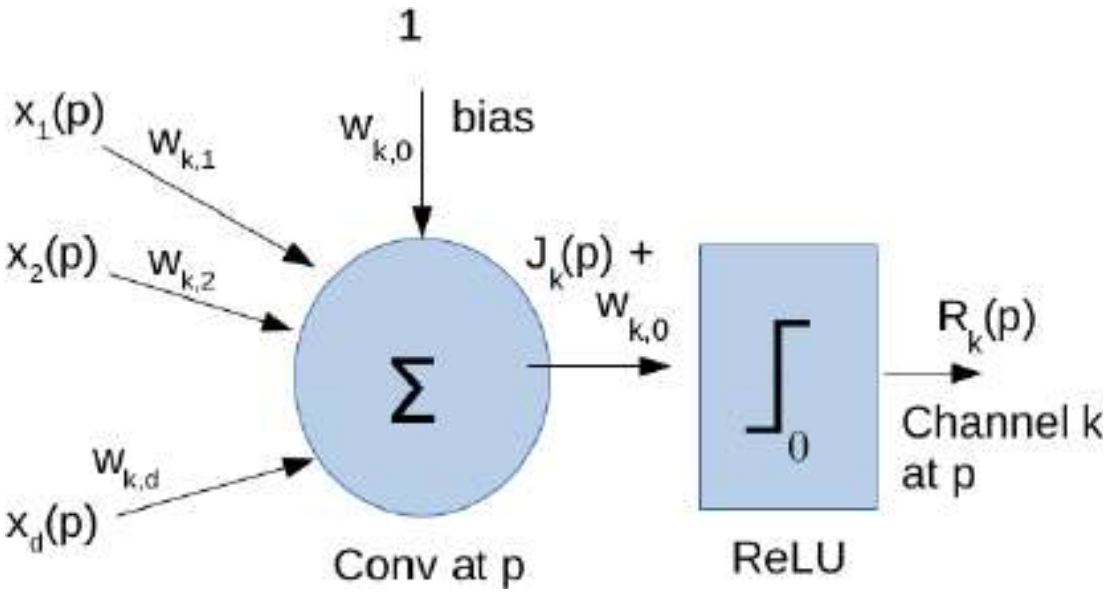
After convolution

After ReLU

Transitions from dark to bright are enhanced.

Convolution, bias, and activation

For patch $\mathbf{X}(p) = (x_1(p), x_2(p), \ldots, x_d(p))$ and k-th kernel $\mathbf{W}_k = (w_{k,1}, w_{k,2}, \ldots, w_{k,d})$, $k \in [1, b]$, $d = n \times m$, convolution plus bias $w_{k,0} \in \Re$ followed by ReLU defines one perceptron per pixel p (artificial neuron).

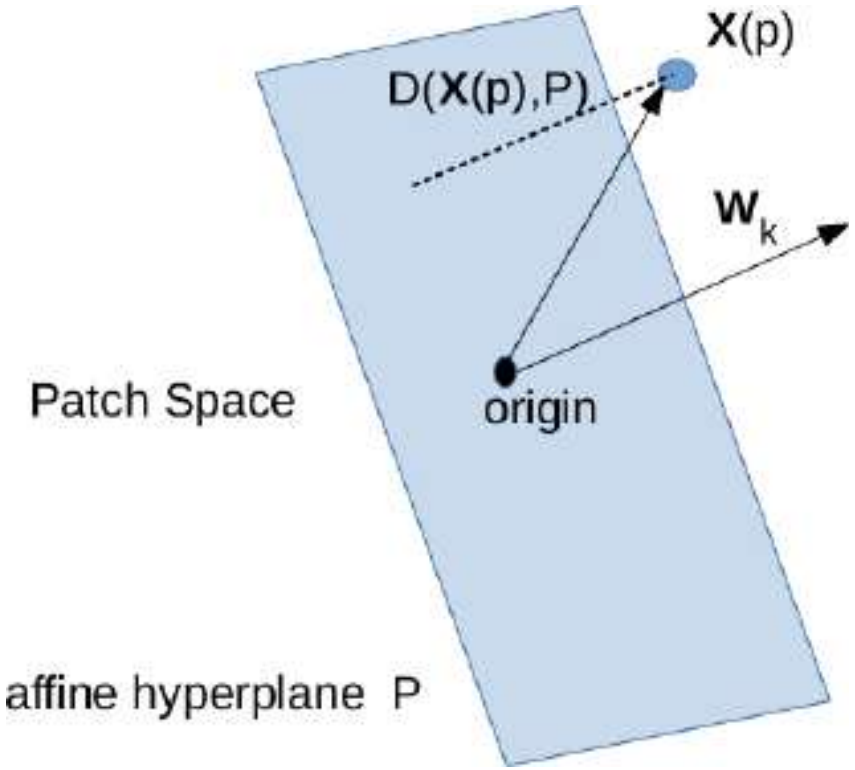


$$J_k(p) = \sum_{i=1}^{d} x_i w_{k,i}$$

$$R_k(p) = \max\{0, J_k(p) + w_{k,0}\}$$

Convolution, bias, and activation

Each kernel $\mathbf{W}_k$ is orthogonal to an affine hyperplane P and convolution plus bias measures a signed distance $D(\mathbf{X}(p), P)$ from $\mathbf{X}(p)$ to P.



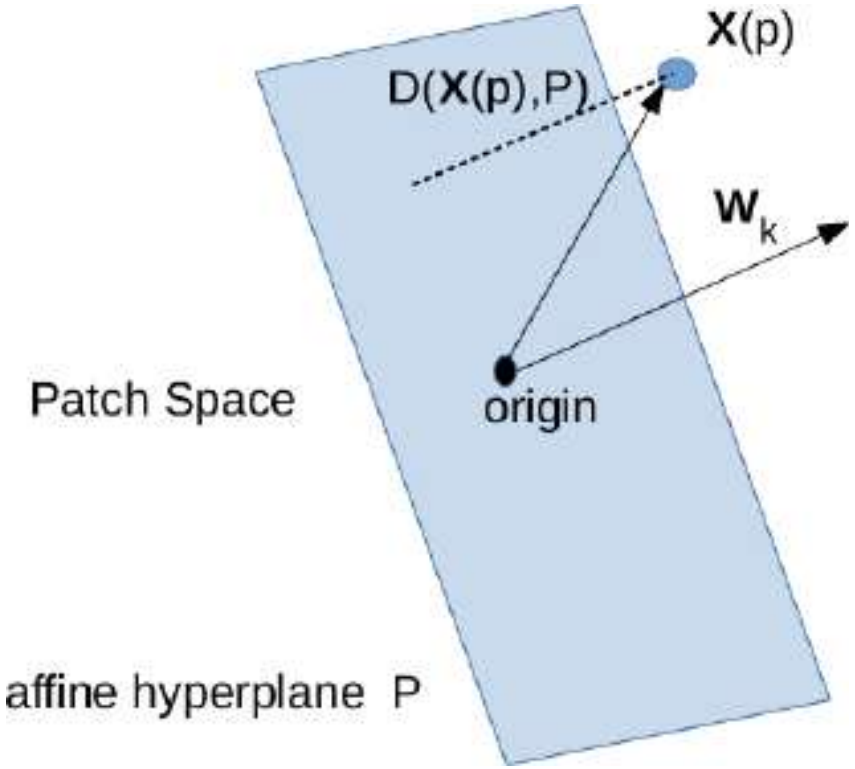
$$D(\mathbf{X}(p), P) \quad = \quad \langle \mathbf{X}(p), \mathbf{W}_k \rangle + w_{k,0} = J_k(p) + w_{k,0}.$$

Alternatively, one may force unit norm to $\mathbf{W}_k$.

Convolution, bias, and activation

The perceptron at p selects $R_k(p)$ as a local feature only when the activation

$$\langle \mathbf{X}(p), \mathbf{w}_k \rangle + w_{k,0} \quad = \quad J_k(p) + w_{k,0} > 0,$$



meaning that, the bias moves P such that $\mathbf{X}(p)$ falls in its positive side.

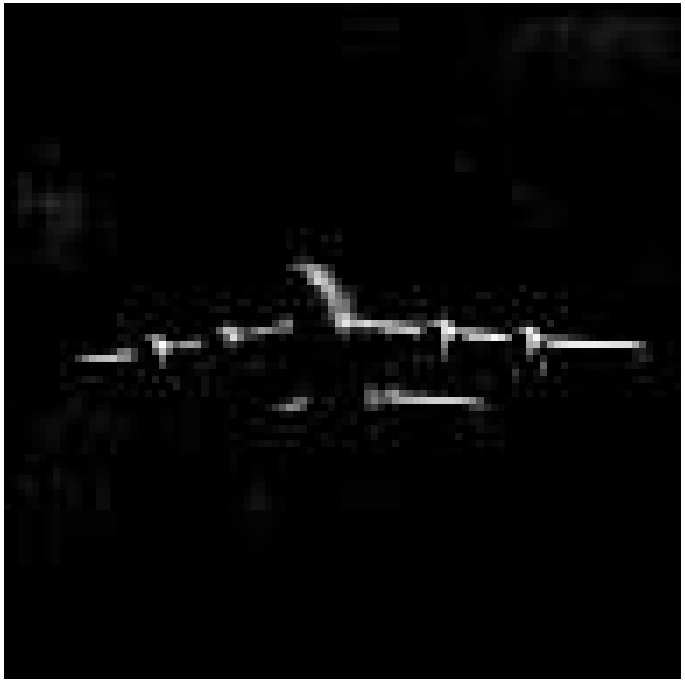
Convolution, bias, and activation

Therefore, the convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should activate parts that best represent classes (object/background).

Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Convolution, bias, and activation

Therefore, the convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should activate parts that best represent classes (object/background).



Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

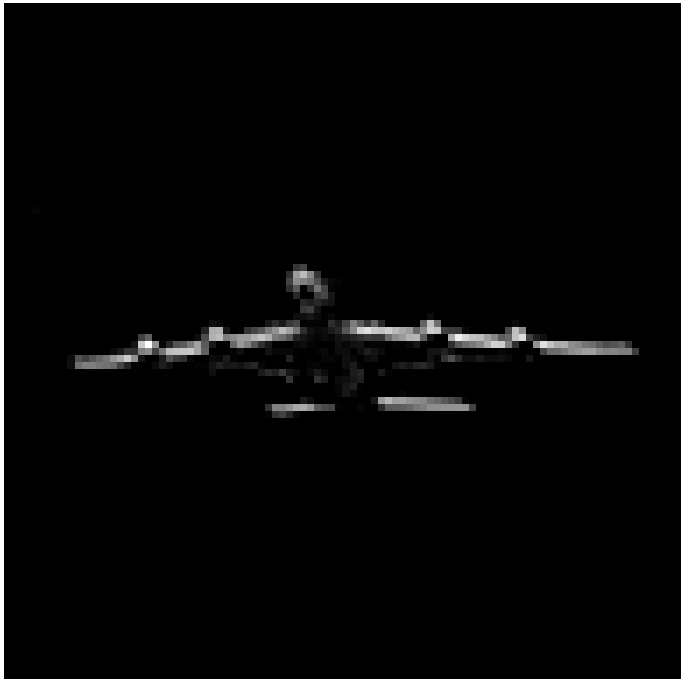
Convolution, bias, and activation

Therefore, the convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should activate parts that best represent classes (object/background).

Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Convolution, bias, and activation

Therefore, the convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should activate parts that best represent classes (object/background).



Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Convolution, bias, and activation

Therefore, the convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should activate parts that best represent classes (object/background).

Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Other image processing operations

- Some pooling types.
- Some normalization types.
- Transpose convolution.
- Flattening.

Pooling

The activations $R_k(p)$ related to an object might also appear at nearby positions within and across images.

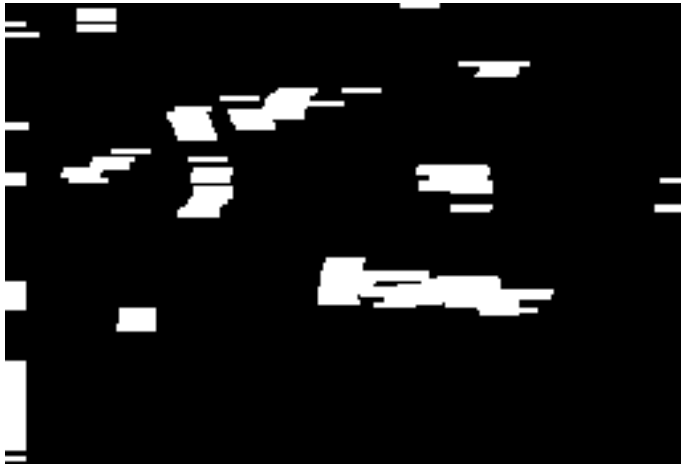
Pooling

The activations $R_k(p)$ related to an object might also appear at nearby positions within and across images.

Max-pooling can aggregate them by transforming $\hat{R} = (D_I, \mathbf{R})$ into $\hat{P} = (D_P, \mathbf{P})$, $\mathbf{P}(p) = (P_1(p), P_2(p), \ldots, P_b(p))$,

$$P_k(p) = \max_{\forall q \in B(p)} \{R_k(q)\},$$

where B is an adjacency relation.



In this case, the widest component is the plate.

Pooling

The activations $R_k(p)$ related to an object might also appear at nearby positions within and across images.

Max-pooling can aggregate them by transforming $\hat{R} = (D_I, \mathbf{R})$ into $\hat{P} = (D_P, \mathbf{P})$, $\mathbf{P}(p) = (P_1(p), P_2(p), \ldots, P_b(p))$,

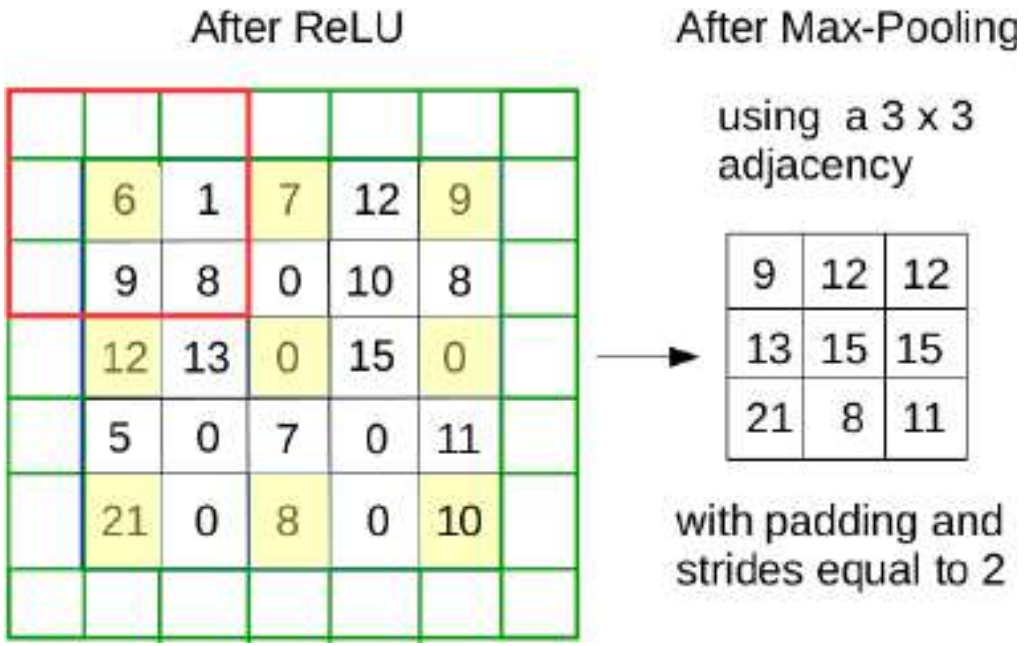
$$P_k(p) = \max_{\forall q \in B(p)} \{R_k(q)\},$$

where B is an adjacency relation.

In this case, the widest component is the plate.

Pooling with stride

It is also common to apply padding and down-sampling with displacements (strides) $s_x \geq 1$ and $s_y \geq 1$.



For a $w \times h$ rectangular adjacency and image domain D_I with $n_x \times n_y$ pixels, the image domain D_P will have $\lfloor \frac{2n_x - w}{2s_x} \rfloor \times \lfloor \frac{2n_y - h}{2s_y} \rfloor$ pixels without padding and $\lceil \frac{n_x}{s_x} \rceil \times \lceil \frac{n_y}{s_y} \rceil$ pixels with padding.

Pooling

Other examples that create $\hat{P} = (D_I, \mathbf{P})$ by pooling are min-pooling and average pooling.

- Min-pooling:

$$P_k(p) = \min_{\forall q \in B(p)} \{R_k(q)\}.$$

- Average pooling:

$$P_k(p) = \frac{1}{|B(p)|} \sum_{\forall q \in B(p)} \{R_k(q)\}.$$

Indeed, any other image filtering could be used here to eliminate undesirable features and/or aggregate the desirable ones for better image processing.

Normalization

Normalizations may be applied to any image $\hat{I} = (D_I, \mathbf{I})$ with m channels or, in batch, to a set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^n$ of m-channel images before/after any step in a convolutional layer.

They are important to avoid discrepancies among local activations along the network.

They create a new image $\hat{N} = (D_I, \mathbf{N})$ with m channels or a new set $\mathcal{N} = \{\hat{N}_j\}_{j=1}^n$ of m-channel images.

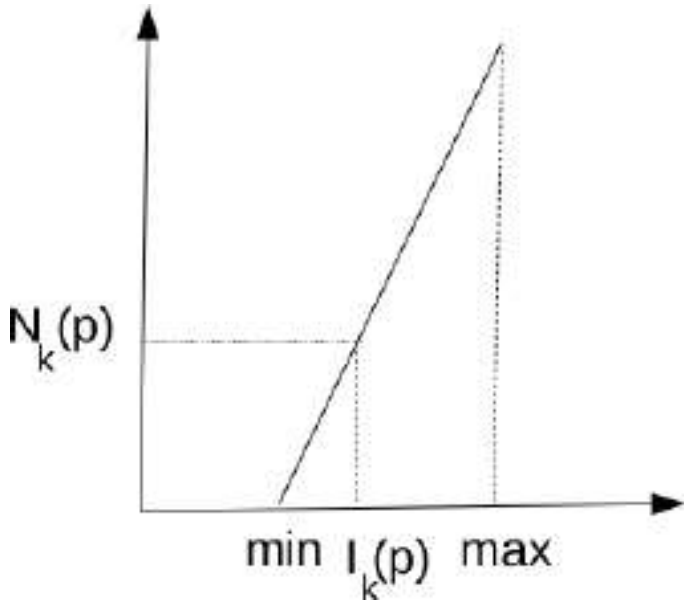
Linear normalization

For $\hat{N} = (D_I, \mathbf{N})$, $\mathbf{N}(p) = (N_1(p), N_2(p), \ldots, N_m(p))$,

$$N_k(p) = \frac{I_k(p) - \min_{\forall q \in D_I}\{I_k(q)\}}{\max_{\forall q \in D_I}\{I_k(q)\} - \min_{\forall q \in D_I}\{I_k(q)\}},$$

$$N_k(p) = \frac{I_k(p) - \min_{j=1}^{n}\{I_{j,k}(p)\}}{\max_{j=1}^{n}\{I_{j,k}(p)\} - \min_{j=1}^{n}\{I_{j,k}(p)\}},$$

$k \in [1, m]$ and $p \in D_I$, we have a linear normalization.



Batch normalization

Batch normalization is very useful to standardize local activations and eliminate the need of bias learning.

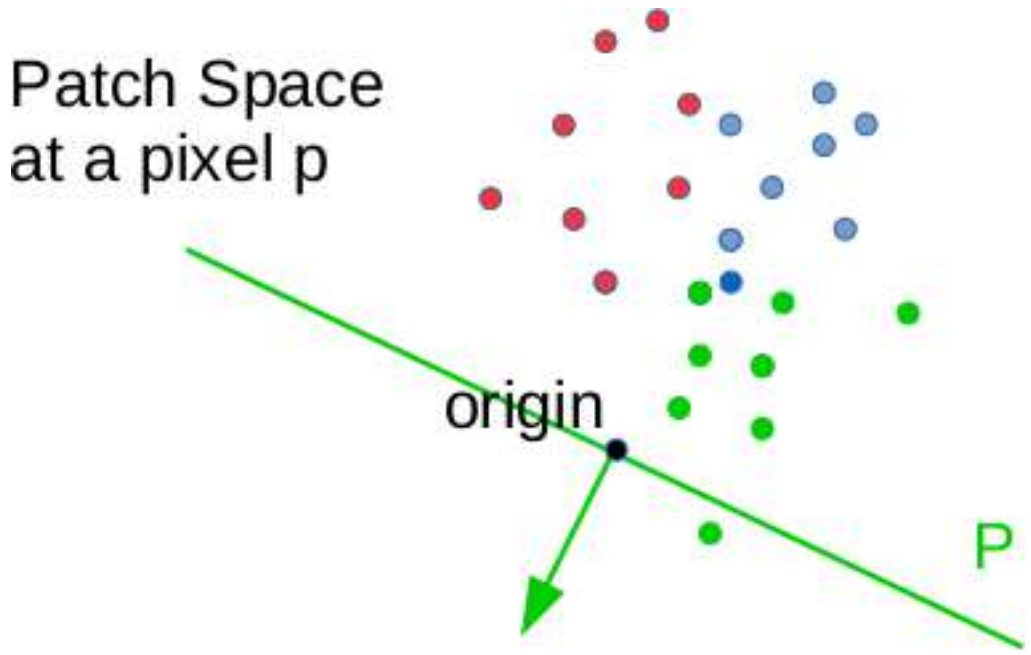
It creates an image $\hat{N} = (D_I, \mathbf{N})$, with

$$
\begin{aligned}
N_k(p) &= \frac{I_k(p) - \mu_k(p)}{\sigma_k(p)}\gamma + \beta, \\
\mu_k(p) &= \frac{1}{n}\sum_{j=1}^{n} I_{j,k}(p), \\
\sigma_k^2(p) &= \frac{1}{n-1}\sum_{j=1}^{n}(I_k(p) - \mu_k(p))^2,
\end{aligned}
$$

for $k \in [1, m]$, $p \in D_I$, and $\gamma, \beta \in \Re$ are parameters that can be learned and even undo this operation. Let $\gamma = 1$ and $\beta = 0$ be their default values.

Batch normalization

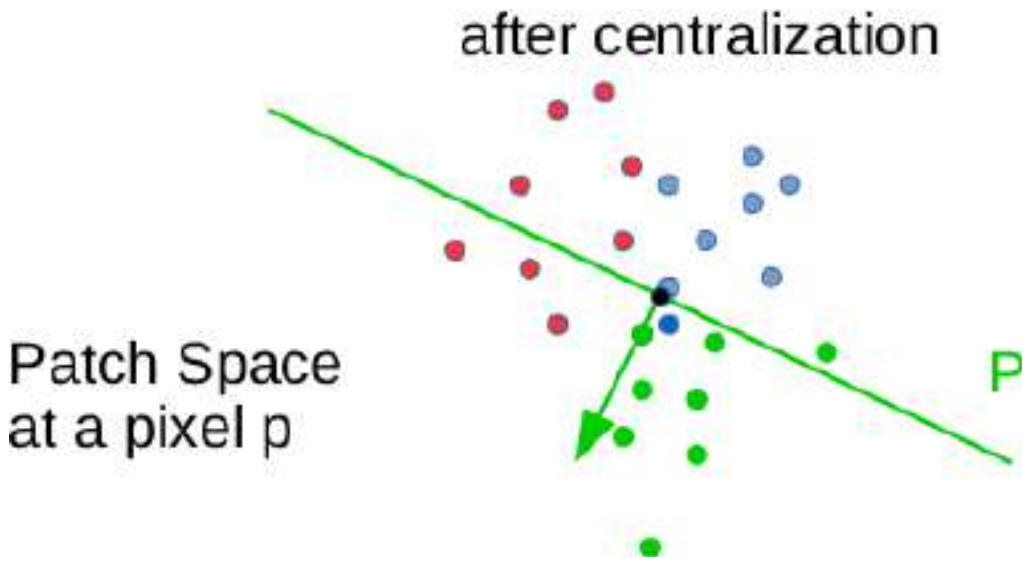
Batch normalization affects the patch space with points $\mathbf{X}_j(p)$ from an image set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{n}$ for all pixels $p \in D_I$.



Just the centralization of the point cloud already shows that training can adjust a kernel to select more activations from a given class with no need of bias.

Batch normalization

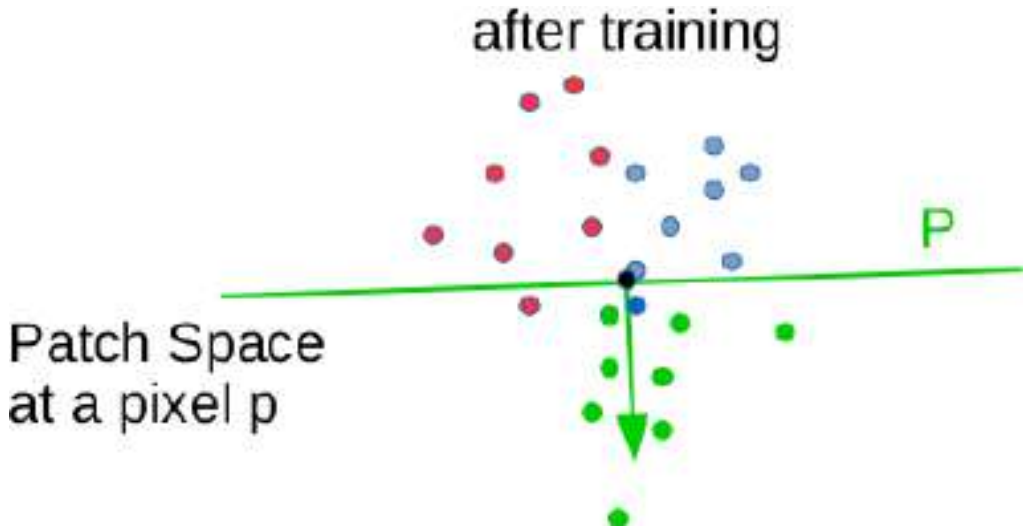
Batch normalization affects the patch space with points $\mathbf{X}_j(p)$ from an image set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{n}$ for all pixels $p \in D_I$.



Just the centralization of the point cloud already shows that training can adjust a kernel to select more activations from a given class with no need of bias.

Batch normalization

Batch normalization affects the patch space with points $\mathbf{X}_j(p)$ from an image set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{n}$ for all pixels $p \in D_I$.



Just the centralization of the point cloud already shows that training can adjust a kernel to select more activations from a given class with no need of bias.

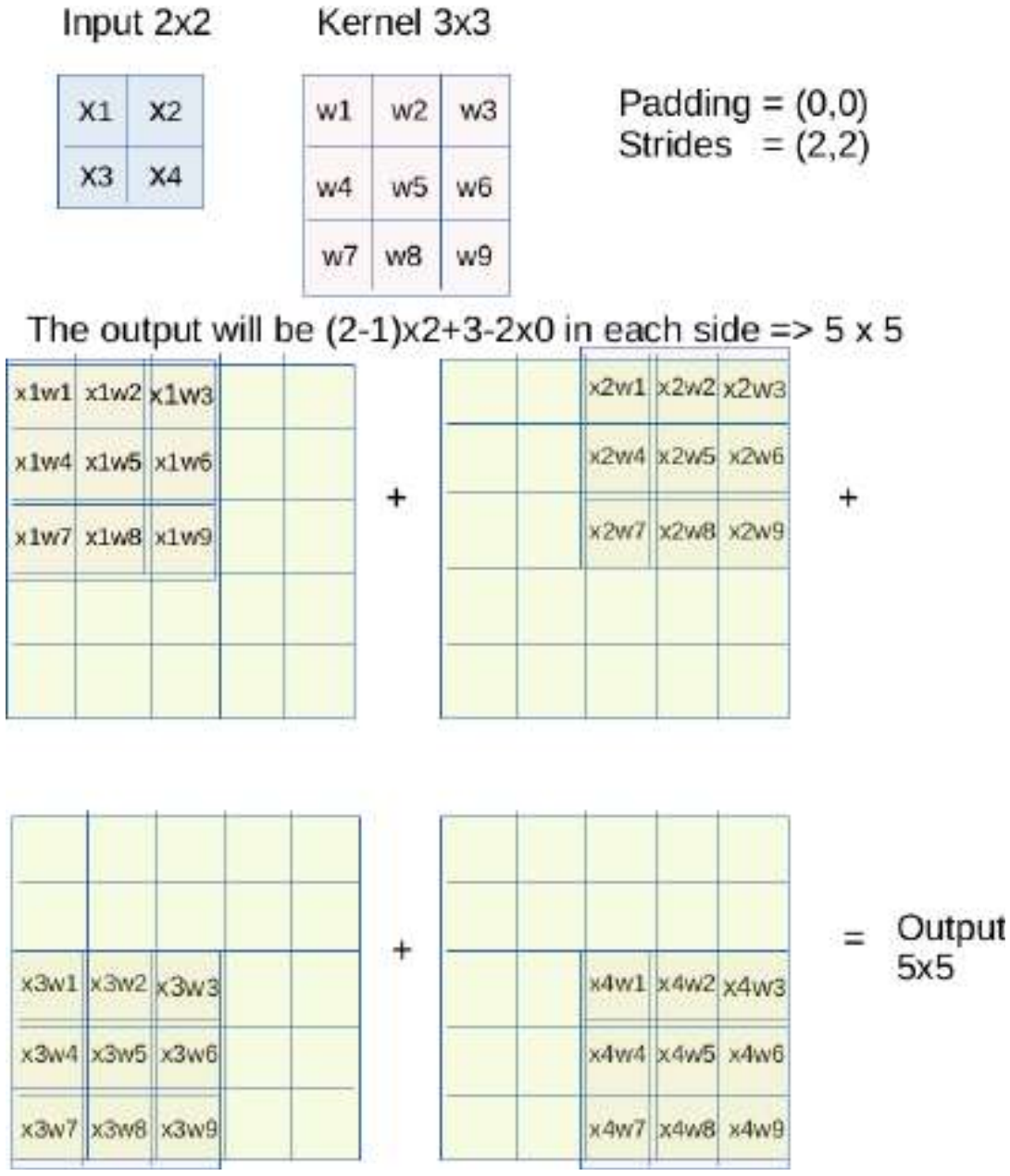
Transposed convolution

- The transposed convolution is a way to perform up-sampling by exploring a kernel of size (K_x, K_y), padding (P_x, P_y) and strides (S_x, S_y). It transforms an image with (N_x, N_y) pixels into another with (O_x, O_y) pixels, where $O_* = (N_* - 1)S_* + K_* - 2P_*$.

- Predictors for image segmentation often use this operation.

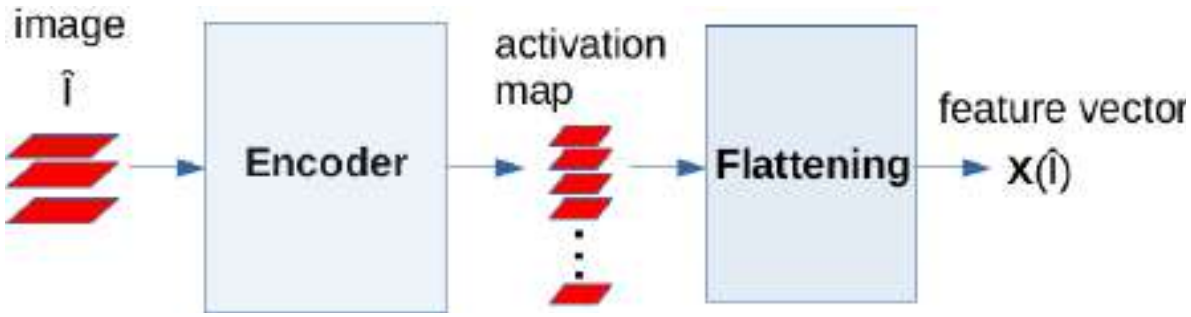
- Transposed convolution can also be substituted by a linear/cubic interpolation.

Transposed convolution



Flattening

Flattening is the vectorization of each channel B_k, $k = 1, 2, \ldots, b$, of an activation map followed by their concatenation.

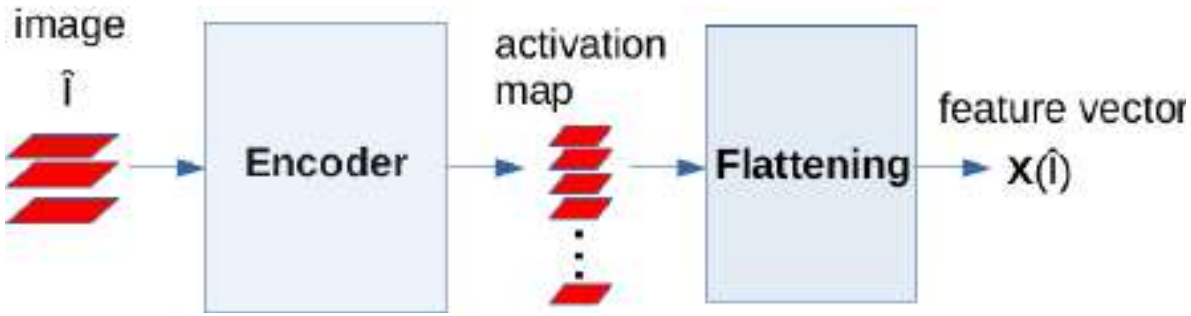


It represents a global feature vector for the input image $\hat{I}$.

$$\mathbf{X}(\hat{I}) \quad = \quad vec(B_1) \bullet vec(B_2) \bullet \ldots \bullet vec(B_b)$$

Flattening

Flattening is the vectorization of each channel B_k, $k = 1, 2, \ldots, b$, of an activation map followed by their concatenation.



It represents a global feature vector for the input image $\hat{I}$.

$$\mathbf{X}(\hat{I}) \quad = \quad vec(B_1) \bullet vec(B_2) \bullet \ldots \bullet vec(B_b)$$

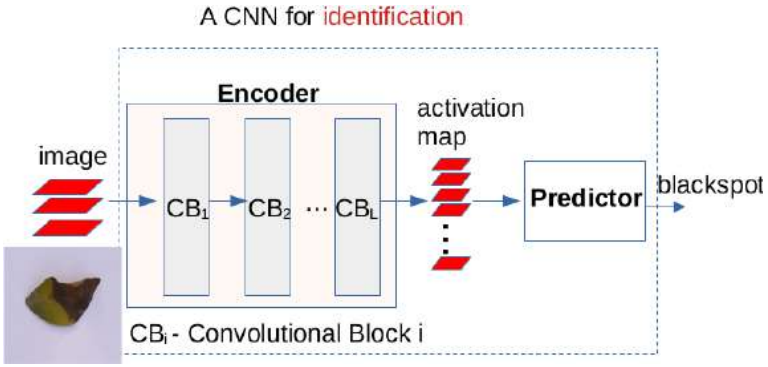
The size of $\mathbf{X}(\hat{I})$ will depend on D_I, b, zero-padding and stride options in the encoder.

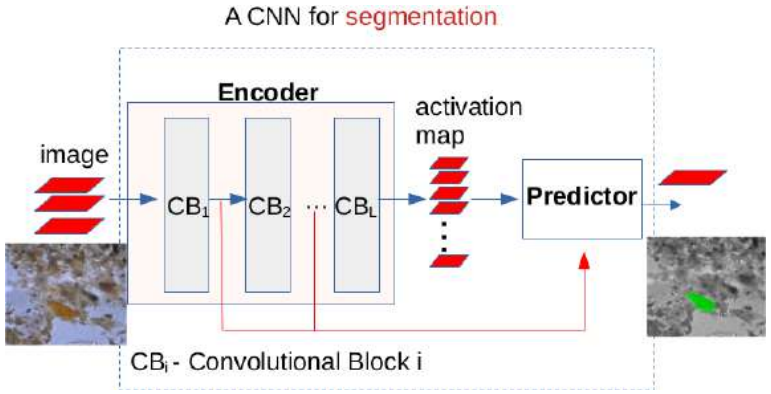
Agenda

- What are CNNs?
- Basic definitions and image processing operations.
- CNNs for image identification and segmentation.
- Hands-on – How to design and evaluate CNNs using pytorch.
- Explainable encoders for identification and segmentation.
- Hands-on – CNNs with explainable encoders using pytorch.
- Final remarks.

CNNs for identification and segmentation

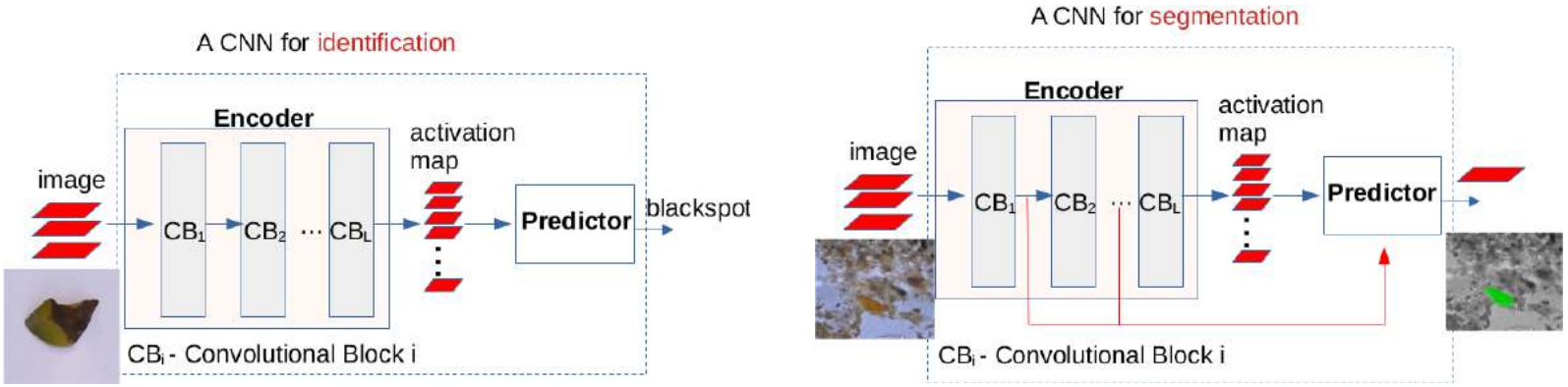
CNNs for identification and segmentation mainly differ in the predictor.





CNNs for identification and segmentation

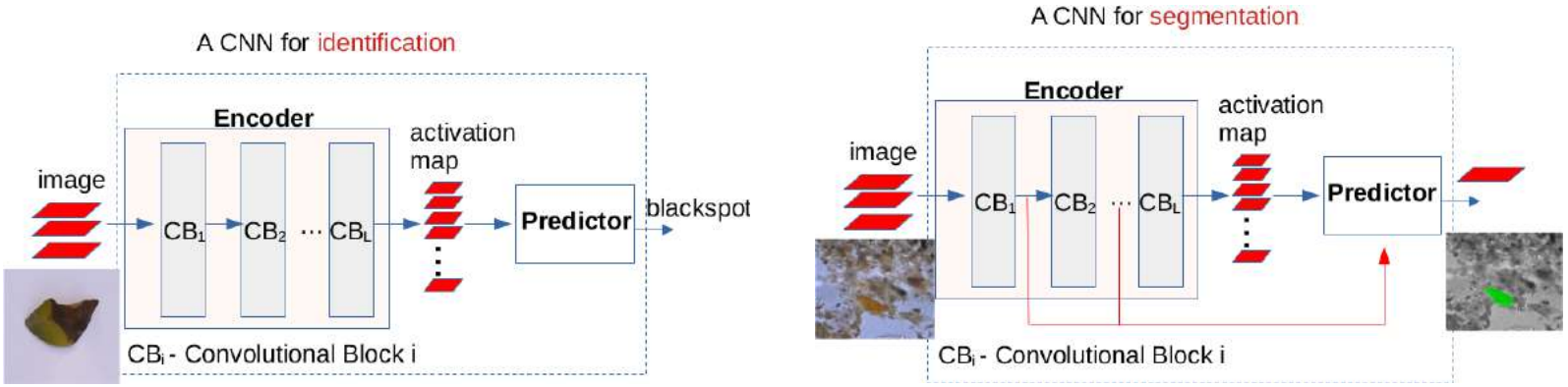
CNNs for identification and segmentation mainly differ in the predictor.



- For identification, they use flattening followed by a pattern classifier, usually a multi-layer perceptron (MLP) due to the training by backpropagation. However, it could be a support vector machine (SVM).

CNNs for identification and segmentation

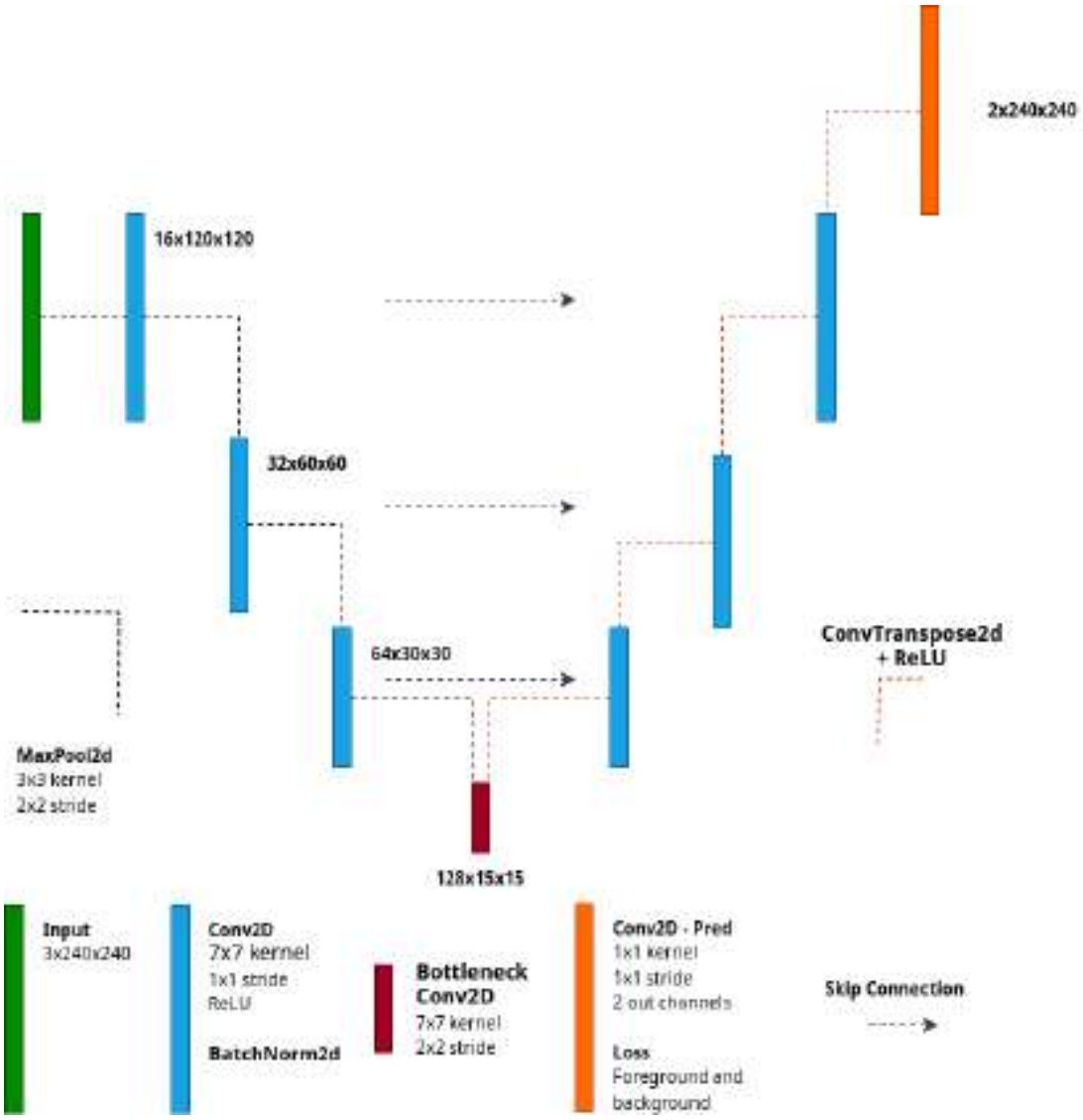
CNNs for identification and segmentation mainly differ in the predictor.



- For identification, they use flattening followed by a pattern classifier, usually a multi-layer perceptron (MLP) due to the training by backpropagation. However, it could be a support vector machine (SVM).
- For segmentation, they may use transposed and normal convolutions, batch normalization, activations, skip connections, and post-processing. Famous examples are the U-shaped architectures.

A U-shaped architecture for image segmentation

A U-shaped architecture for semantic segmentation with two objects (classes).



Agenda

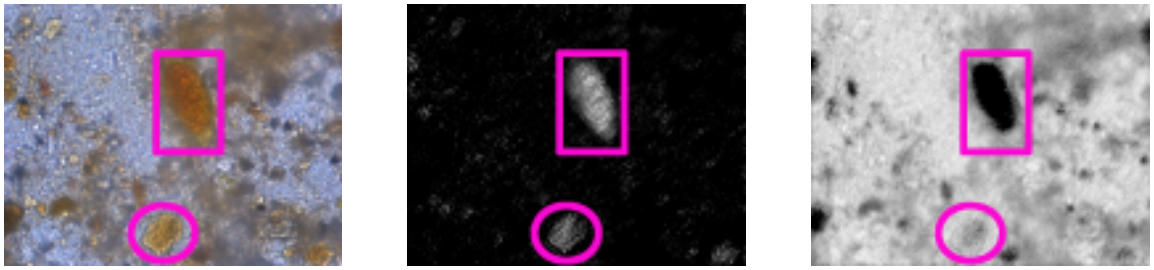
- What are CNNs?
- Basic definitions and image processing operations.
- CNNs for image identification and segmentation.
- Hands-on – How to design and evaluate CNNs using pytorch.
- Explainable encoders for identification and segmentation.
- Hands-on – CNNs with explainable encoders using pytorch.
- Final remarks.

Agenda

- What are CNNs?
- Basic definitions and image processing operations.
- CNNs for image identification and segmentation.
- Hands-on – How to design and evaluate CNNs using pytorch.
- Explainable encoders for identification and segmentation.
- Hands-on – CNNs with explainable encoders using pytorch.
- Final remarks.

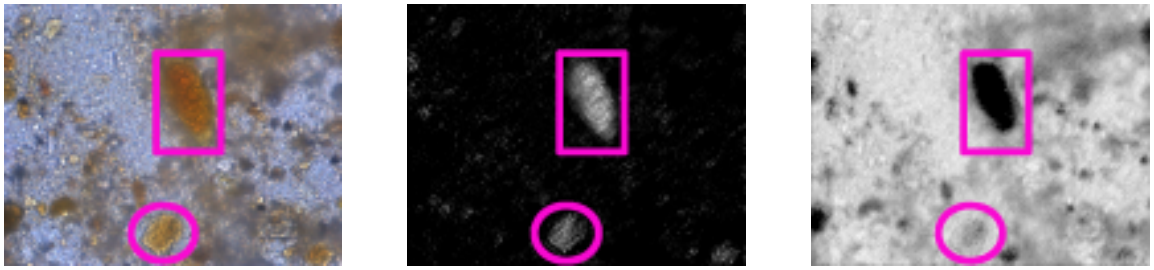
Explainable encoders for identification and segmentation

- For segmentation, the kernels should create foreground and background activations such that the latter can suppress false positives from the former in the point-wise convolution of the last decoder layer.



Explainable encoders for identification and segmentation

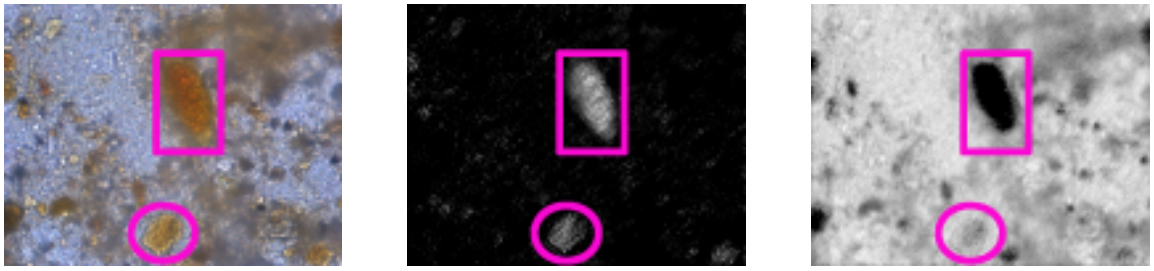
- For segmentation, the kernels should create foreground and background activations such that the latter can suppress false positives from the former in the point-wise convolution of the last decoder layer.



- A similar effect should be observed for identification by separating pixel activations of each class such that the classes can be linearly separated in the resulting global feature space.

Explainable encoders for identification and segmentation

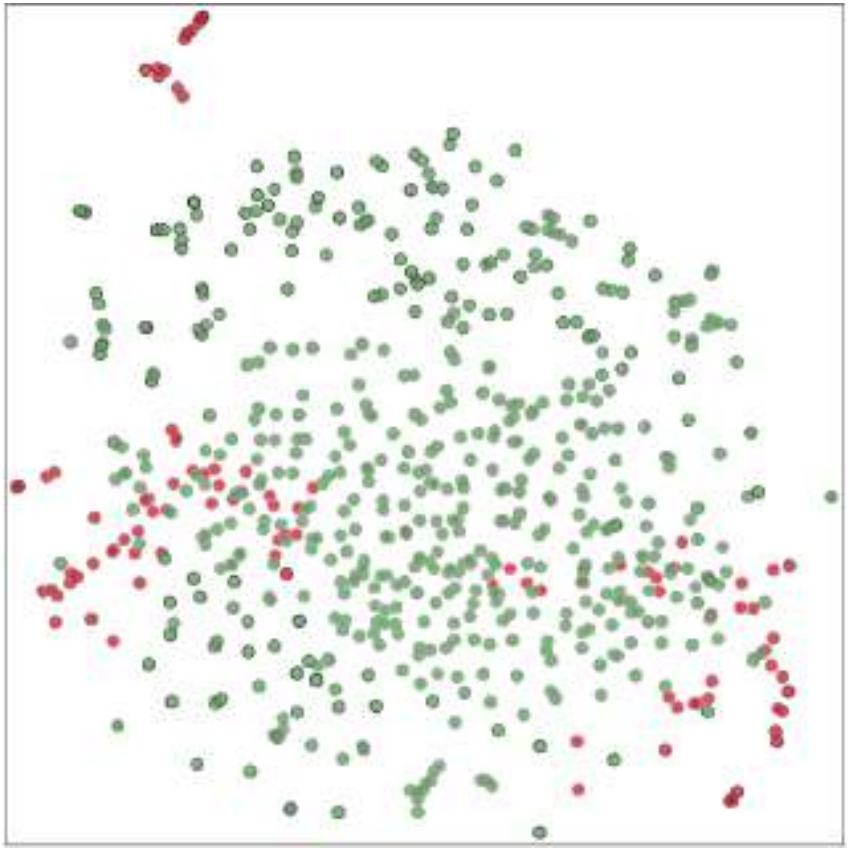
- For segmentation, the kernels should create foreground and background activations such that the latter can suppress false positives from the former in the point-wise convolution of the last decoder layer.



- A similar effect should be observed for identification by separating pixel activations of each class such that the classes can be linearly separated in the resulting global feature space.

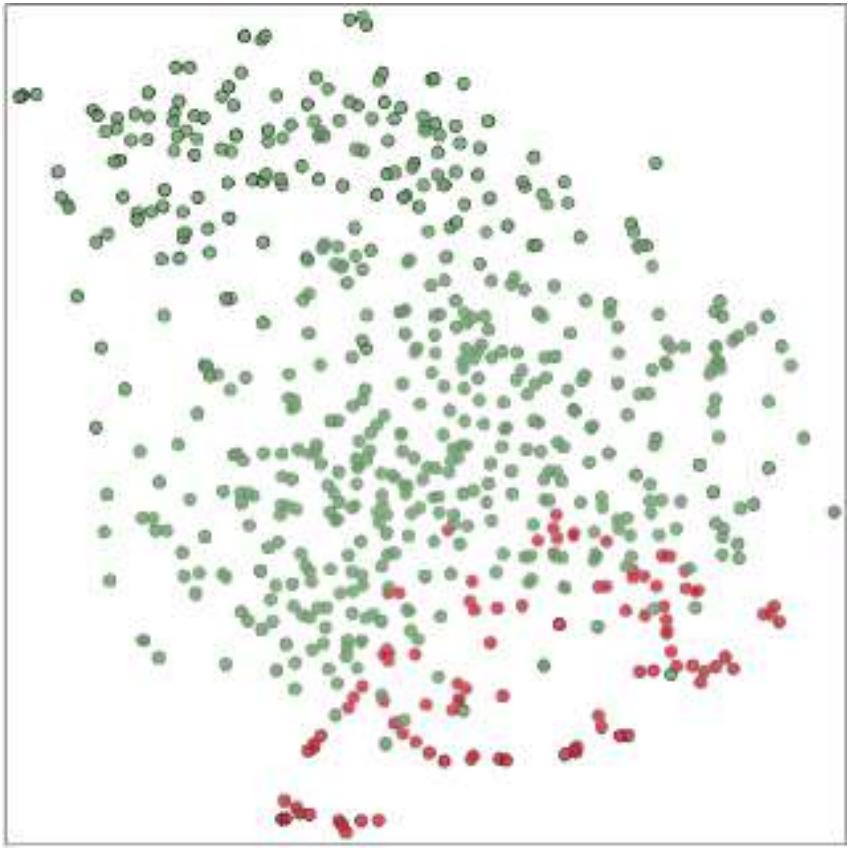
- For identification, such class separation may be observed when the global feature space of deeper convolutional/dense layers are non-linearly projected.

Explainable encoders for identification and segmentation



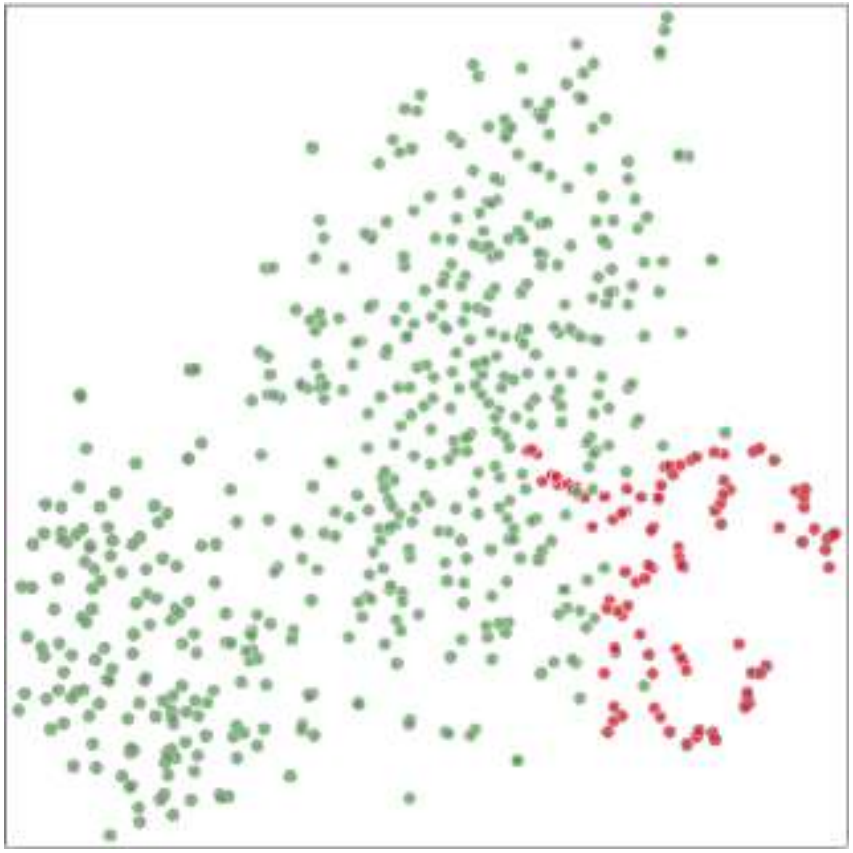
t-SNE projection after encoder block 10 in VGG-16 for image classification of larvae of helminth and impurities.

Explainable encoders for identification and segmentation



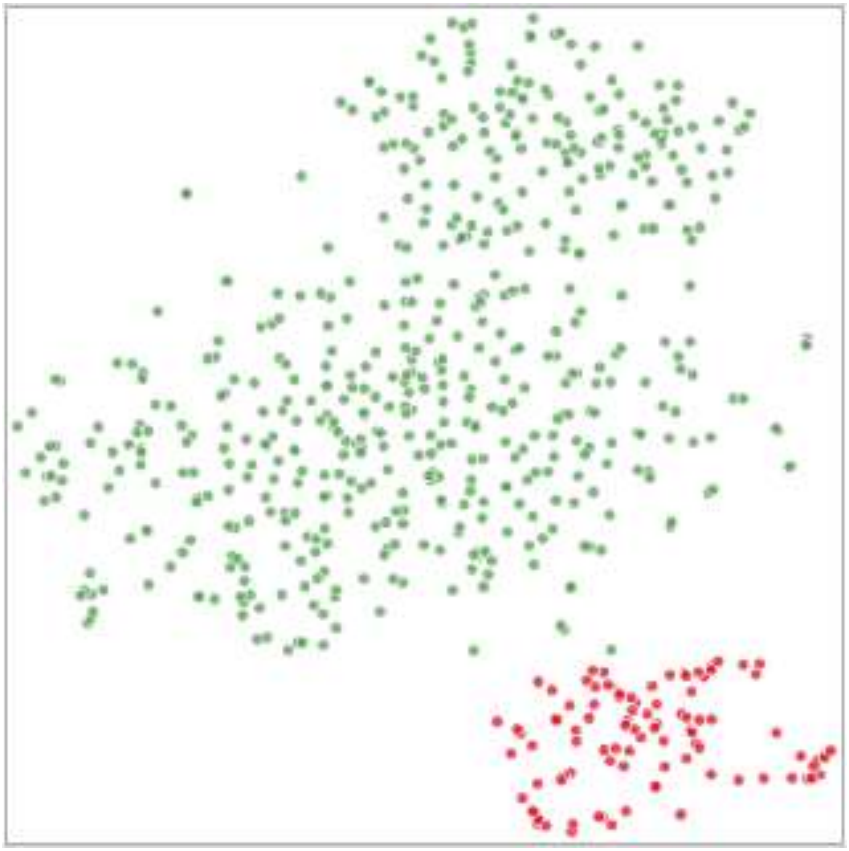
t-SNE projection after encoder block 11 in VGG-16 for image classification of larvae of helminth and impurities.

Explainable encoders for identification and segmentation



t-SNE projection after encoder block 12 in VGG-16 for image classification of larvae of helminth and impurities.

Explainable encoders for identification and segmentation



t-SNE projection after encoder block 13 in VGG-16 for image classification of larvae of helminth and impurities.

Explainable encoders for identification and segmentation

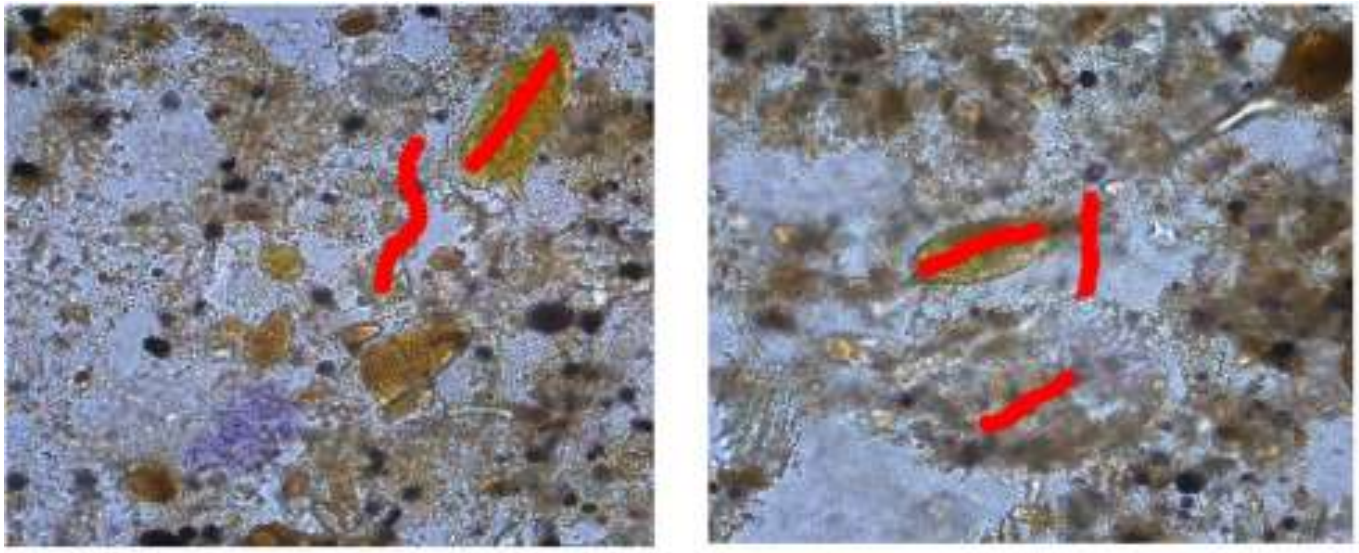
- We present a methodology, named Feature Learning from Image Markers (FLIM), to estimate kernels from disks/strokes drawn in a few representative images (e.g., two per class), with no need for backpropagation [GRSL20].

- FLIM has been successfully used for detection, identification, and segmentation in remote sensing [GRSL20,GRSL22], natural [NEURIPS20,SIBGRAPI20], and medical images [EMBC21, GRSL22, ARXIV23, SIPAIM23].

- It can create lightweight models competitive or superior to deep models for some applications [GRSL20, EMBC21, ARXIV23], and the models usually outperform the same architecture trained from scratch by backpropagation.

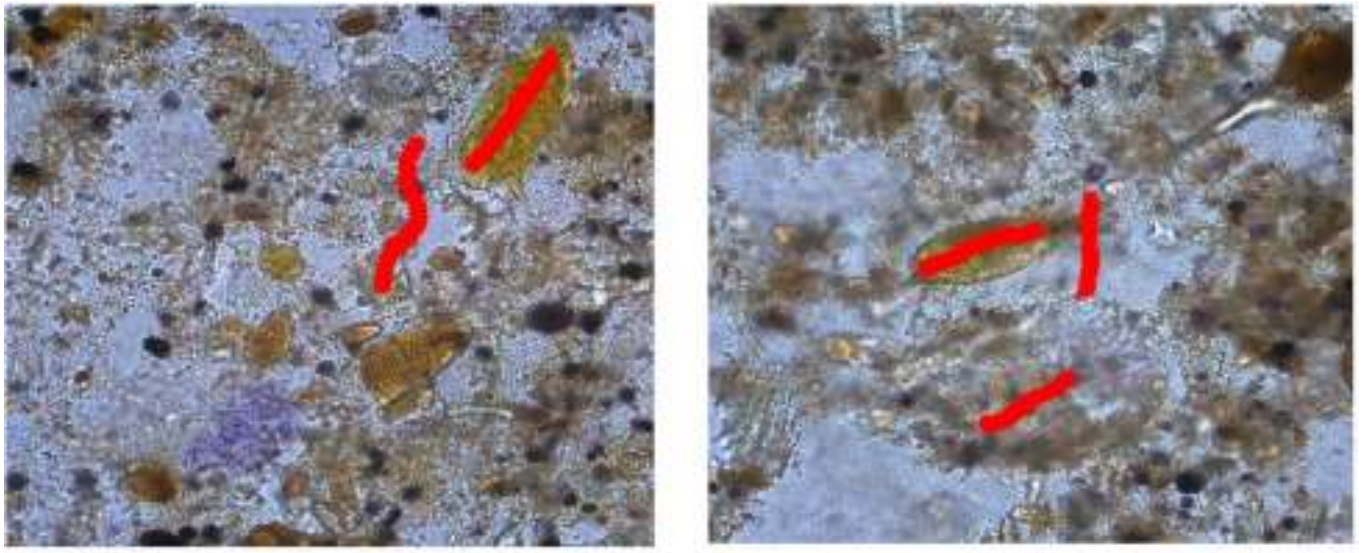
Feature Learning from Image Markers (FLIM)

For segmentation, the expert draws markers on discriminative regions of a few representative images as input.



Feature Learning from Image Markers (FLIM)

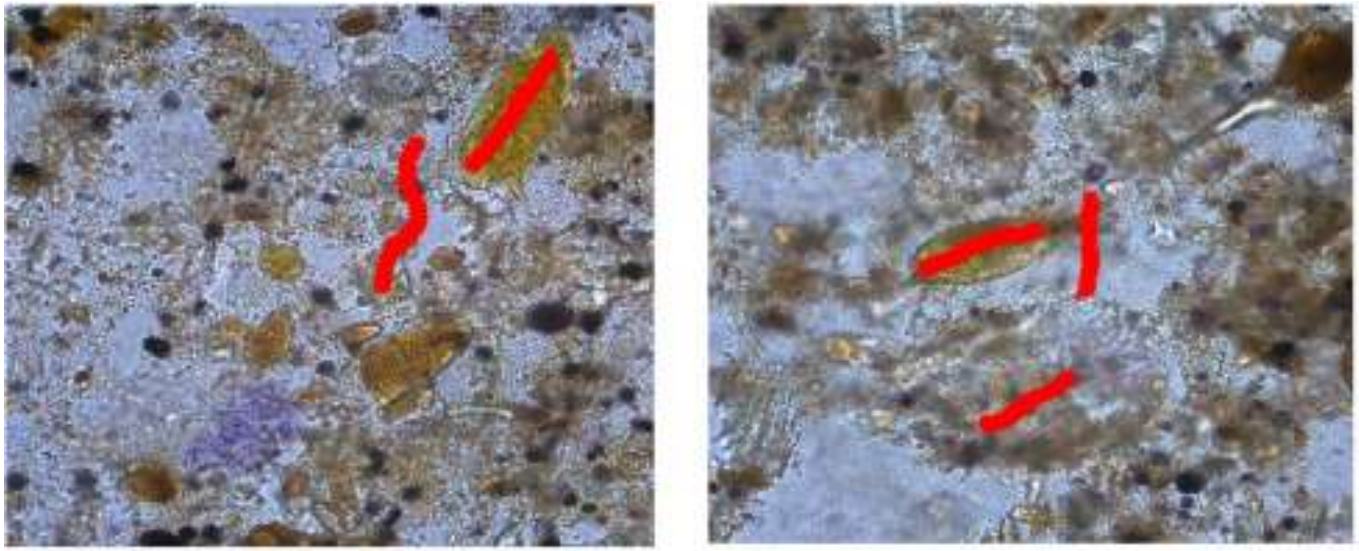
For segmentation, the expert draws markers on discriminative regions of a few representative images as input.



- The kernels are automatically estimated block by block from this input for a given encoder architecture.

Feature Learning from Image Markers (FLIM)

For segmentation, the expert draws markers on discriminative regions of a few representative images as input.



- The kernels are automatically estimated block by block from this input for a given encoder architecture.
- The expert may intervene by adding/removing markers, eliminating kernels, or selecting more images.

How does training work?

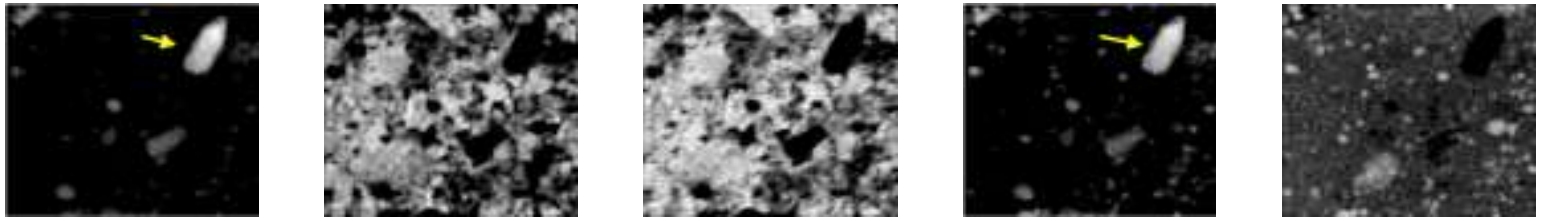
From image patches centered at marker pixels, we wish to

- identify groups of patches that represent patterns of interest in the images,
- estimate one kernel **W** per group, such that the convolution between **W** and an image can
  - activate regions whose patterns are similar to the ones represented by **W** and
  - deactivate image regions with dissimilar patterns.

How does training work?

From image patches centered at marker pixels, we wish to

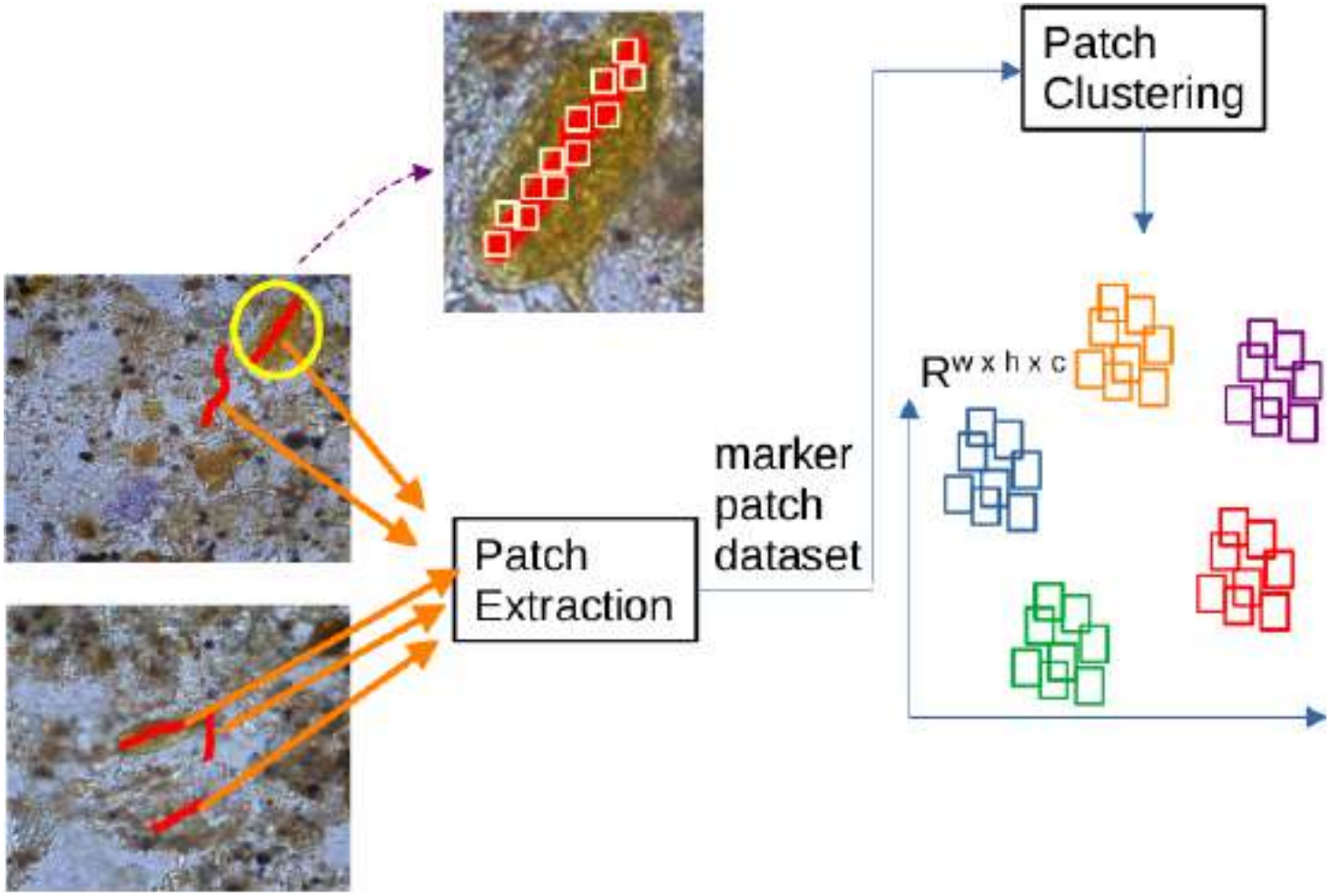
- identify groups of patches that represent patterns of interest in the images,
- estimate one kernel **W** per group, such that the convolution between **W** and an image can
  - activate regions whose patterns are similar to the ones represented by **W** and
  - deactivate image regions with dissimilar patterns.



\+ - - \+ -

How does training work?

A patch $\mathbf{X}(p) \in \Re^{w \times h \times c}$ at a pixel p with width w, height h, and c channels is a local feature vector of size $n = w \times h \times c$.



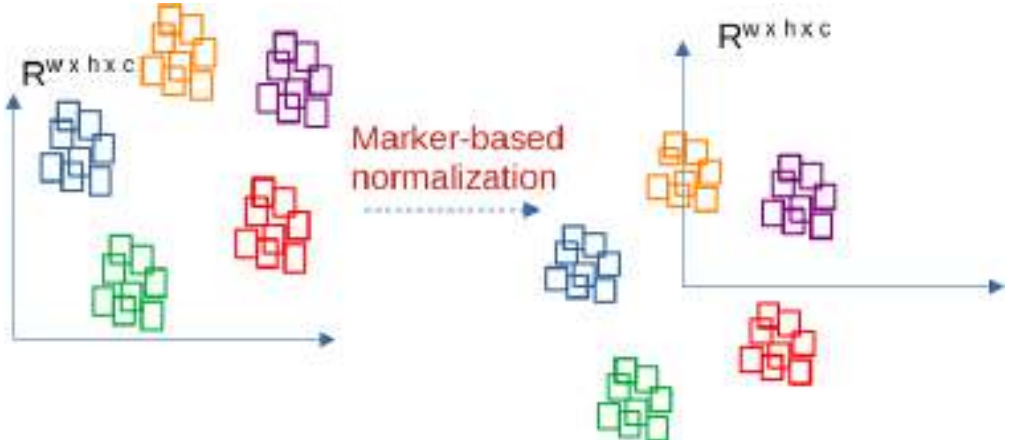
Patches centered at marker pixels are grouped into a given number of clusters.

How does training work?

Patches $\mathbf{X}(p) = (X_1(p), X_2(p), \ldots, X_n(p))$ in the marker patch dataset $\mathcal{X}$ are normalized as $\mathbf{Z}(p) = (Z_1(p), Z_2(p), \ldots, Z_n(p))$, where

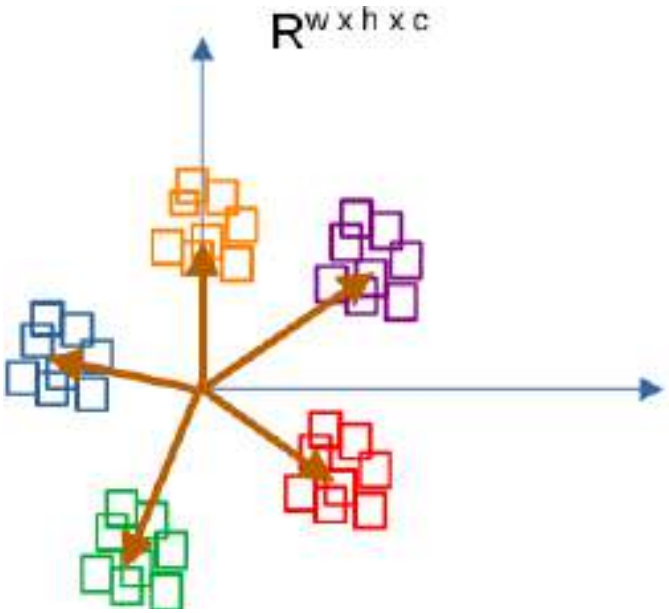
$$
\begin{aligned}
Z_i(p) &= \frac{X_i(p) - \mu_i}{\sigma_i + \epsilon}, \\
\mu_i &= \frac{1}{|\mathcal{X}|} \sum_{X(p) \in \mathcal{X}} X_i(p), \\
\sigma_i^2 &= \frac{1}{|\mathcal{X}|} \sum_{X(p) \in \mathcal{X}} (X_i(p) - \mu_i)^2,
\end{aligned}
$$

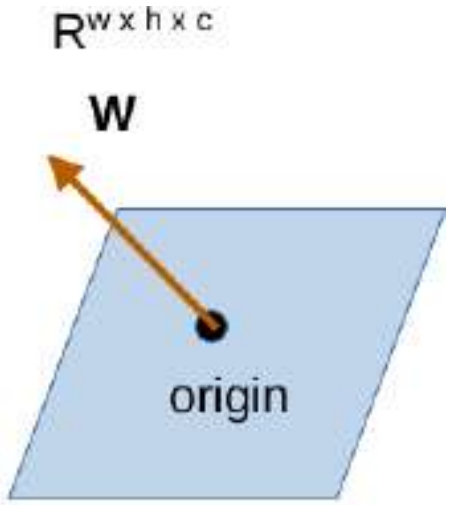
$i = 1, 2, \ldots, n$ and a very small $\epsilon > 0$



How does training work?

The kernels $\mathbf{W} \in \Re^{w \times h \times c}$ are the cluster centers.

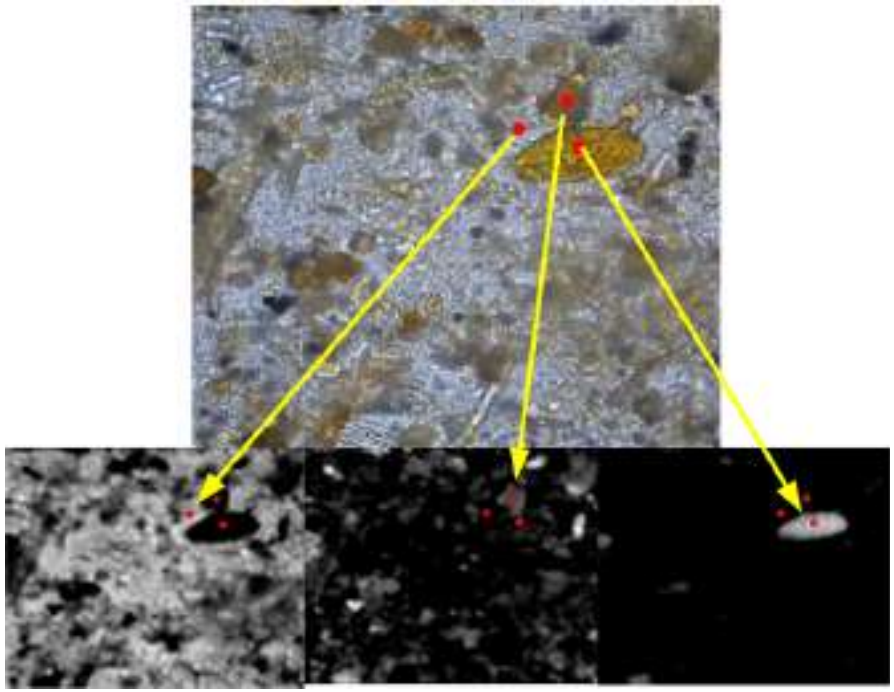




Each kernel $\mathbf{W}$ is orthogonal to a hyperplane in $\Re^{w \times h \times c}$ and marker-based normalization aims to isolate each cluster (and similar patches) in the positive side of the corresponding hyperplane.

How does training work?

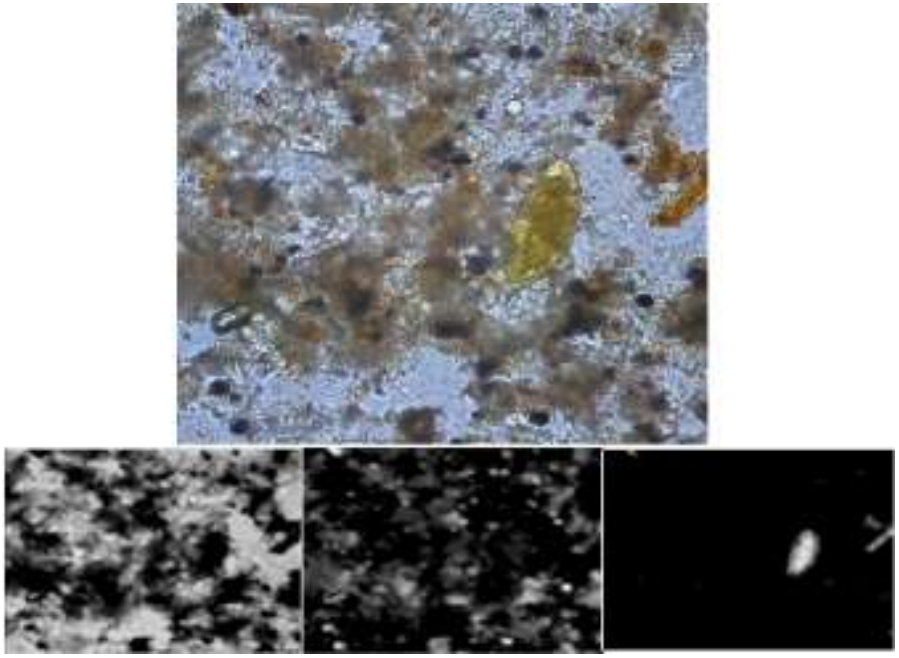
- The convolution $\hat{I} * \mathbf{W}$ between an image $\hat{I}$ with marker-based normalized patches $\mathbf{Z}(p)$ and $\mathbf{W}$ outputs an image $\hat{D}$ with pixel values $D(p) = \langle \mathbf{Z}(p), \mathbf{W} \rangle$.
- ReLU activation eliminates negative values of $D(p)$, and max-pooling aggregates foreground activations within a given neighborhood of each pixel.



Corresponding activations for one kernel per marker.

How does training work?

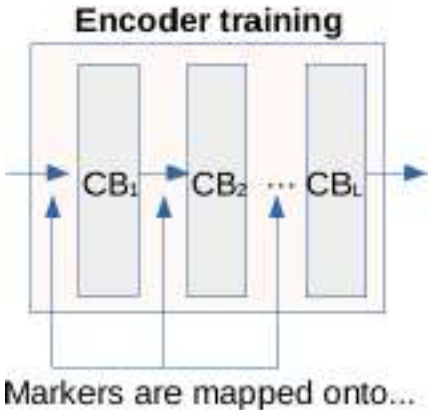
- The convolution $\hat{I} * \mathbf{W}$ between an image $\hat{I}$ with marker-based normalized patches $\mathbf{Z}(p)$ and $\mathbf{W}$ outputs an image $\hat{D}$ with pixel values $D(p) = \langle \mathbf{Z}(p), \mathbf{W} \rangle$.
- ReLU activation eliminates negative values of $D(p)$, and max-pooling aggregates foreground activations within a given neighborhood of each pixel.



Corresponding activations for one kernel per marker.

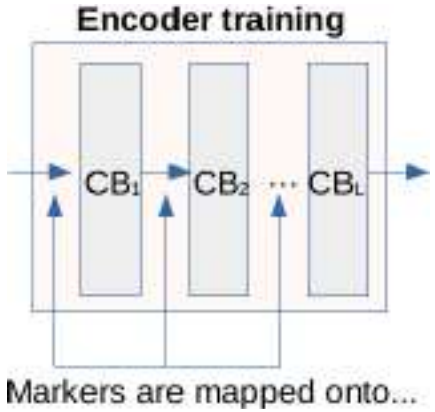
How does training work?

- The exact process is repeated for each convolutional block using the markers mapped onto the activation maps of the previous block.

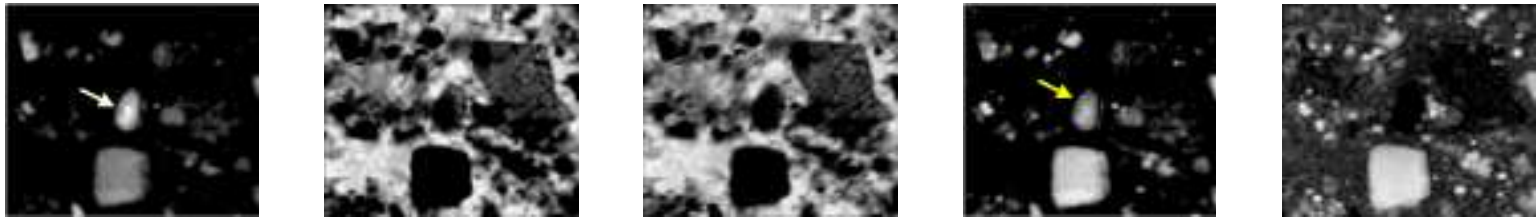


How does training work?

- The exact process is repeated for each convolutional block using the markers mapped onto the activation maps of the previous block.



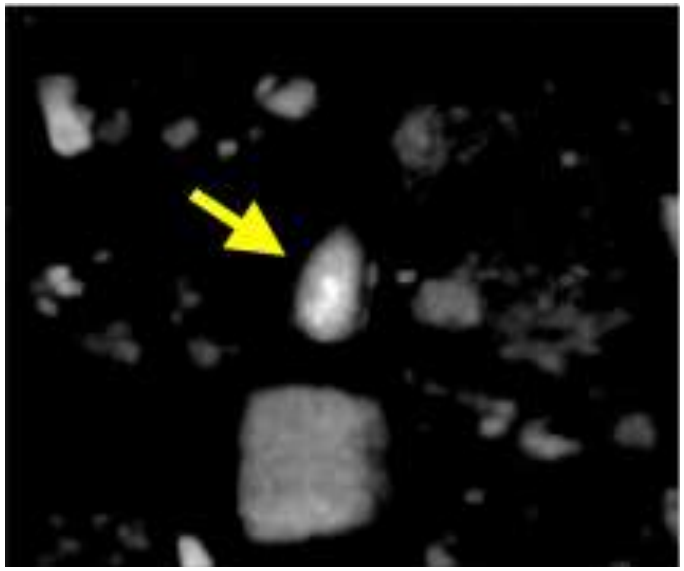
- The expert may examine the activation maps of other images, remove redundant kernels, or draw markers in new images.



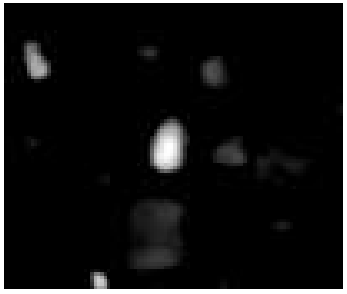
\+ - - \+ -

How does training work?

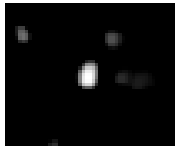
As “deeper” the encoder is, the expert should observe foreground activation maps varying from suitable for object delineation to object detection.



block 1



block 2

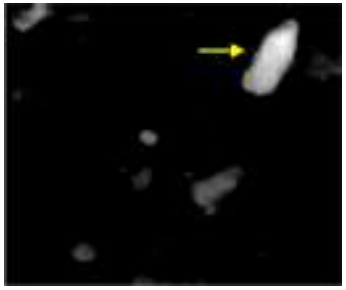


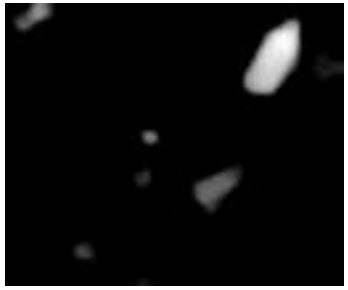
block 3

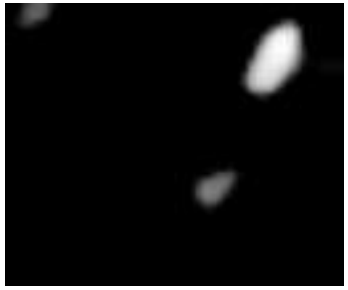
block 4

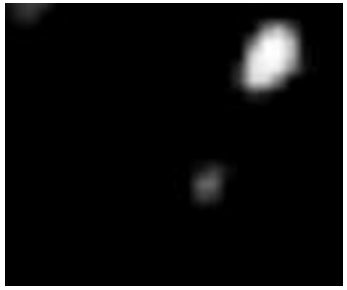
Activation summarization for expert examination

A point-wise convolution (weighted average of the activation maps) followed by ReLU activation can summarize the quality of the activation maps at any block's output – a simple decoder.


block 1

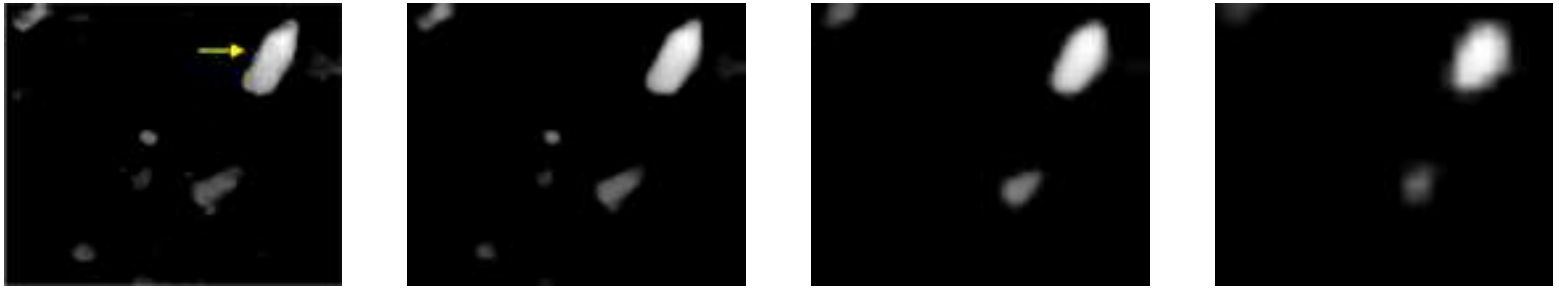

block 2


block 3


block 4

Activation summarization for expert examination

A point-wise convolution (weighted average of the activation maps) followed by ReLU activation can summarize the quality of the activation maps at any block's output – a simple decoder.

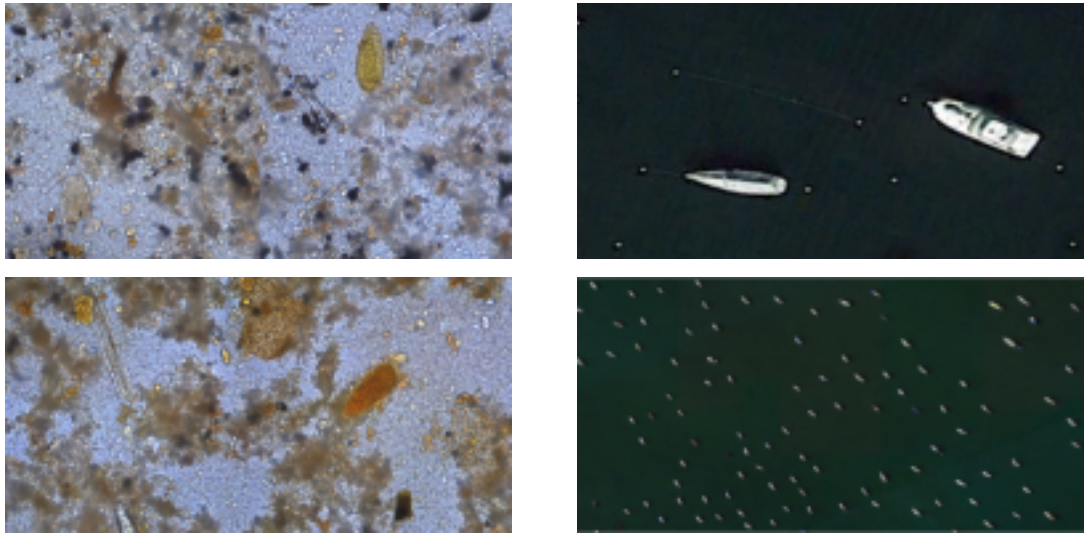


block 1 block 2 block 3 block 4

However, a kernel may activate the foreground in one image and the background in another, asking for an adaptive decoder – a new concep in CNNs.

Activation summarization for expert examination

We have proposed adaptive decoders for distinct object detection problems [ARXIV23].

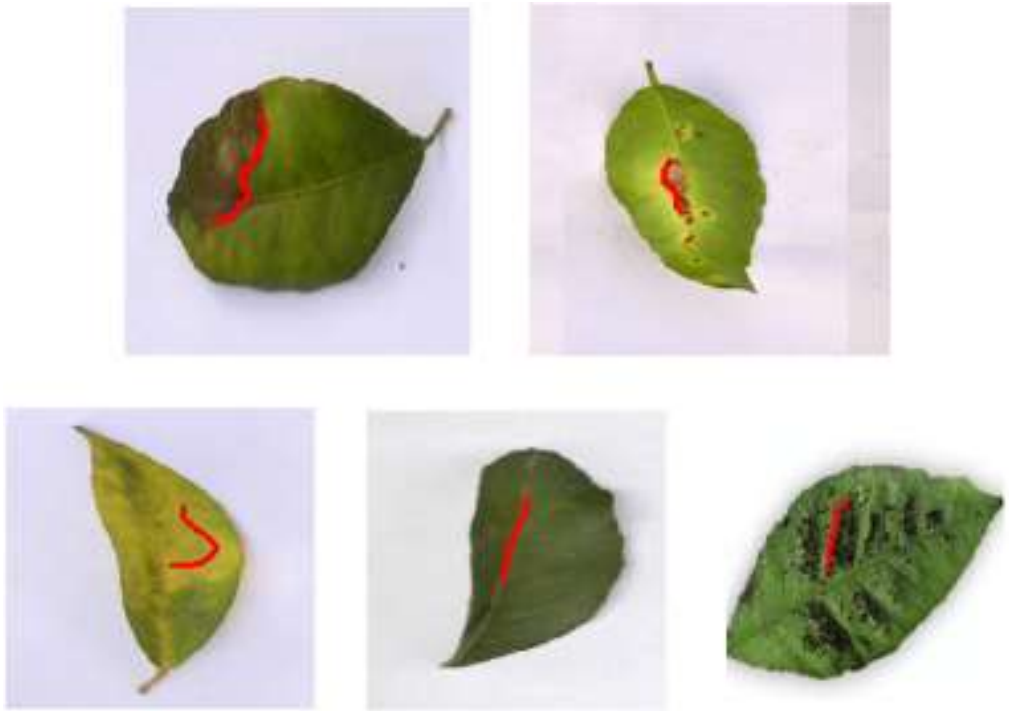


Parasite Eggs Ships

Such decoders may assign weight $w = 1$ to foreground activations, weight -1 to background activations, and weight 0 to neutral activations according to an adaptation function.

How does FLIM extend to identification?

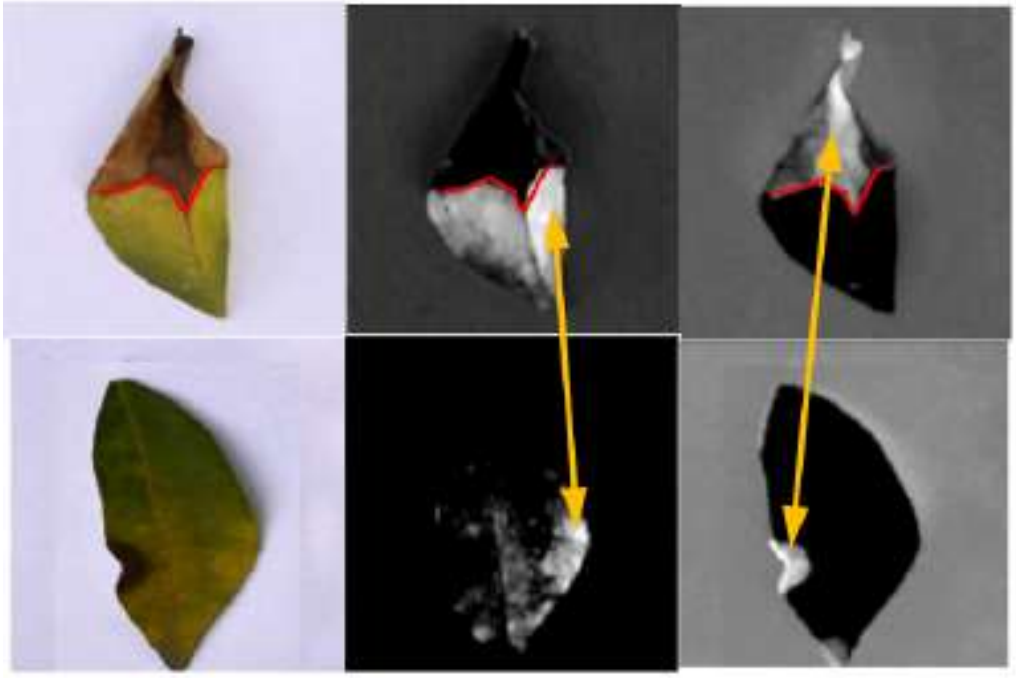
Identification may require more kernels and blocks. The training process is the same, but the best strategy for marker drawing is still under investigation.



The expert can determine the number of kernels per marker and image, but PCA can be applied to reduce the total number of kernels to the desired one in a block.

How does FLIM extend to identification?

Expert assessment of activations is challenging without pixel annotation.

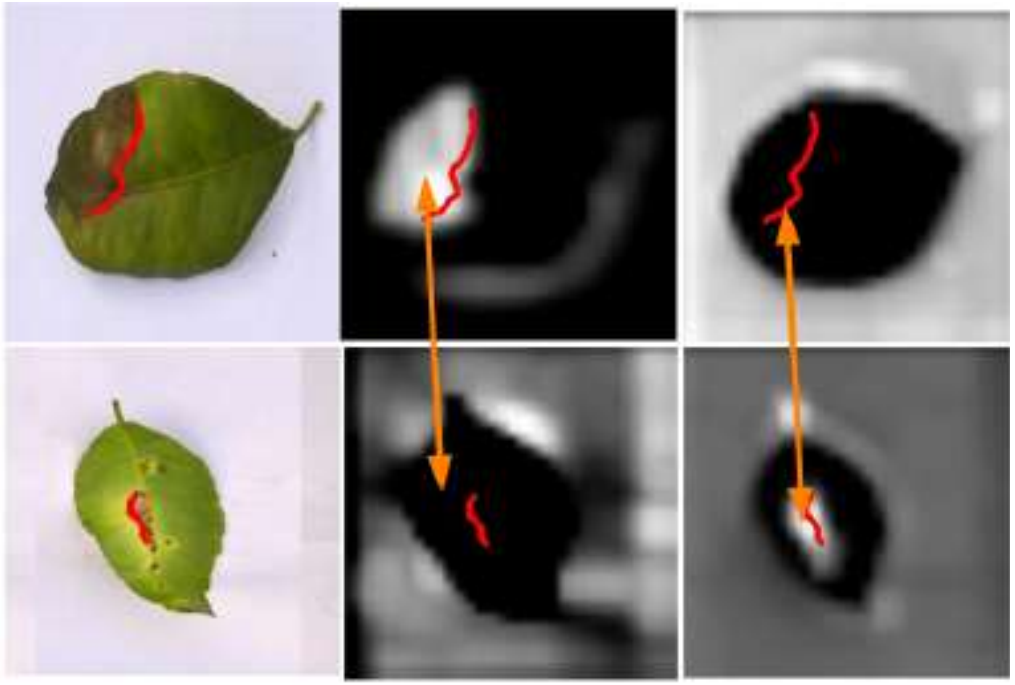


Same class, good (center) and non-good (right) activations

1) Activations from the same class are good at the same channel pixels. Activations in distinct regions of the same channel only increase the class's subspace. 2) For distinct classes, activations in different channels/pixels of the same channel are good.

How does FLIM extend to identification?

Expert assessment of activations is challenging without pixel annotation.



Distinct classes, good activations at the center and on the right.

1) Activations from the same class are good at the same channel pixels. Activations in distinct regions of the same channel only increase the class's subspace. 2) For distinct classes, activations in different channels/pixels of the same channel are good.

Agenda

- What are CNNs?
- Basic definitions and image processing operations.
- CNNs for image identification and segmentation.
- Hands-on – How to design and evaluate CNNs using pytorch.
- Explainable encoders for identification and segmentation.
- Hands-on – CNNs with explainable encoders using pytorch.
- Final remarks.

Final Remarks

- It should be clear the challenges to training CNNs from scratch with backpropagation when the datasets are unbalanced.
- In deep learning, the ultimate goals for experts should be a better understanding and control of the training process.
- FLIM with an adaptive decoder introduces a new way to design explainable and lightweight CNNs without backpropagation.
- In FLIM, one can devise new methods to select images, suggest regions for marker drawing, estimate kernels from markers, and guide the expert's actions.

Acknowledgments

Thank You

FAPESP, CNPq, CAPES, and collaborators (Leonardo João, Azael Sousa, Bianca dos Santos, Jancarlo Gomes, Isabela Borlido, Silvio Guimarães, Felipe Crispim, Matheus Cerqueira, Italos Souza, and Ewa Kijak).

Available codes:
`https://github.com/LIDS-UNICAMP/FLIM`
`https://github.com/LIDS-UNICAMP/ift`
`https://github.com/LIDS-UNICAMP/FLIM-Builder`

References

Book20 V.K. Ayyadevara and Y. Reddy. Modern Computer Vision with PyTorch. ISBN: 9781839213472, Packt, 2020.

GRSL20 I. de Souza and A.X. Falcão. Learning CNN filters from user-drawn image markers for coconut-tree image classification. IEEE Geoscience and Remote Sensing Letters, doi: 10.1109/LGRS.2020.3020098, 2020, `https://arxiv.org/pdf/2008.03549.pdf`.

SIBGRAPI20 I.E. de Souza, B.C. Benato, and A.X. Falcão. Feature Learning from Image Markers for Object Delineation. *33rd SIBGRAPI*, doi: 10.1109/SIBGRAPI51738.2020.00024, 2020.

NEURIPS20 I.E. de Souza, B.C. Benato, F.L. Galcão and A.X. Falcão. Convolutional Neural Networks from Image Markers. Beyond BackPropagation: Novel Ideas for Training Neural Architectures, Neurips Workshop, 2020, `https://arxiv.org/pdf/2012.12108.pdf`.

EMBC21 A.M. Sousa, F. Reis, R. Zerbini, J.L.D. Comba, and A.X. Falcão. CNN Filter Learning from Drawn Markers for the Detection of Suggestive Signs of COVID-19 in CT Images. *43rd EMBC*, doi: 10.1109/EMBC46164.2021.9629806, 2021.

GRSL22 I.E. de Souza, C.L. Cazarin, M.R. Veronez, L. Gonzaga Jr, and A.X. Falcão. User-guided data expansion modeling to train deep neural networks with little supervision. *IEEE Geoscience and Remote Sensing Letters*, doi: 10.1109/LGRS.2022.3201437, 2022.

ARXIV23 L. João, A. Sousa, B. dos Santos, S. Guimarães, J. Gomes, E. Kijak, and A. Falcão. Building Flyweight FLIM-based CNNs with Adaptive Decoding for Object Detection, arXiv 2306.14840, 2023.

SIPAIM23 M.A. Cerqueira, F. Sprenger, B.C.A. Teixeira, and A.X. Falcão. Building Brain Tumor Segmentation Networks with User-Assisted Filter Estimation and Selection. *18th SIPAIM*, doi 10.1117/12.2669770, 2023.