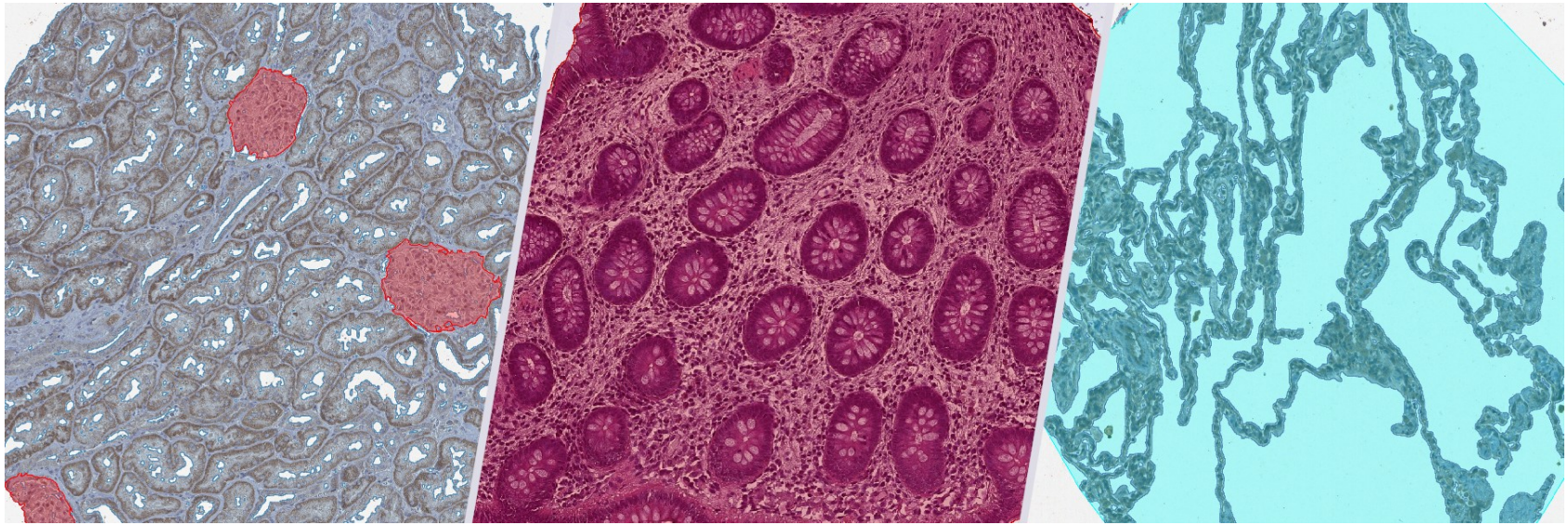
Chapter 4

Tissue segmentation & classification

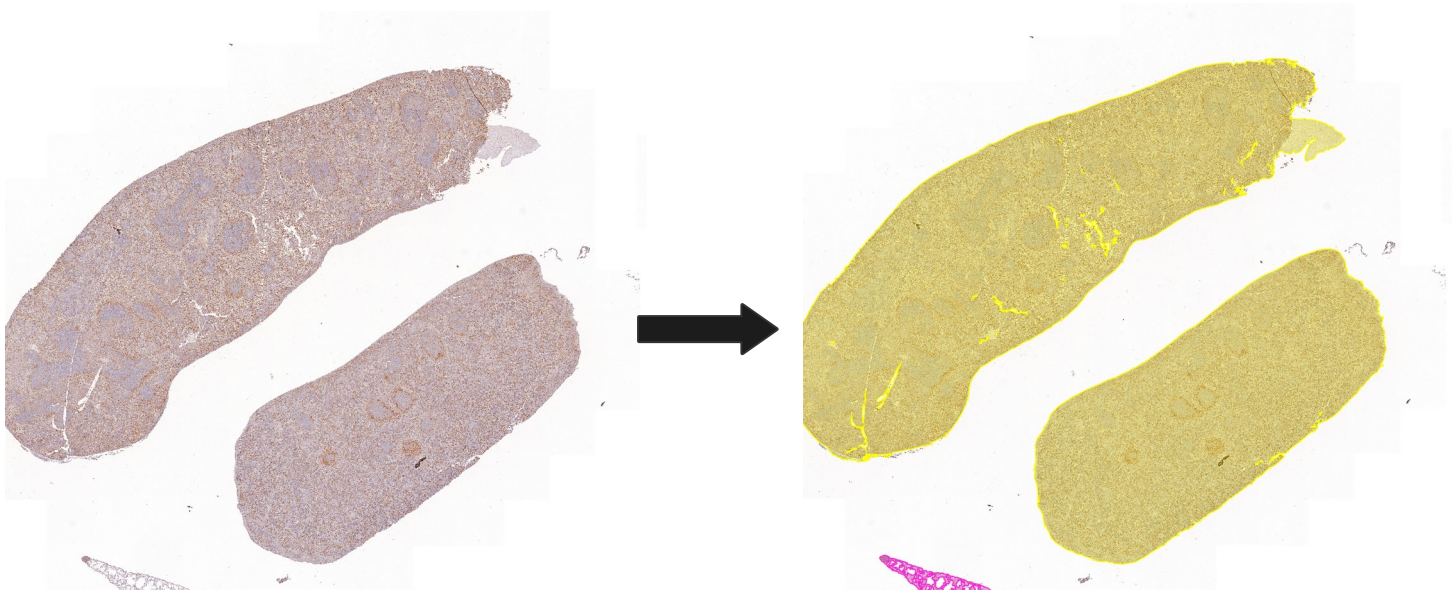


BAECKER Volker
BENEDETTI Clément
HASSEN-KHODJA Cédric

Introduction

Goals:

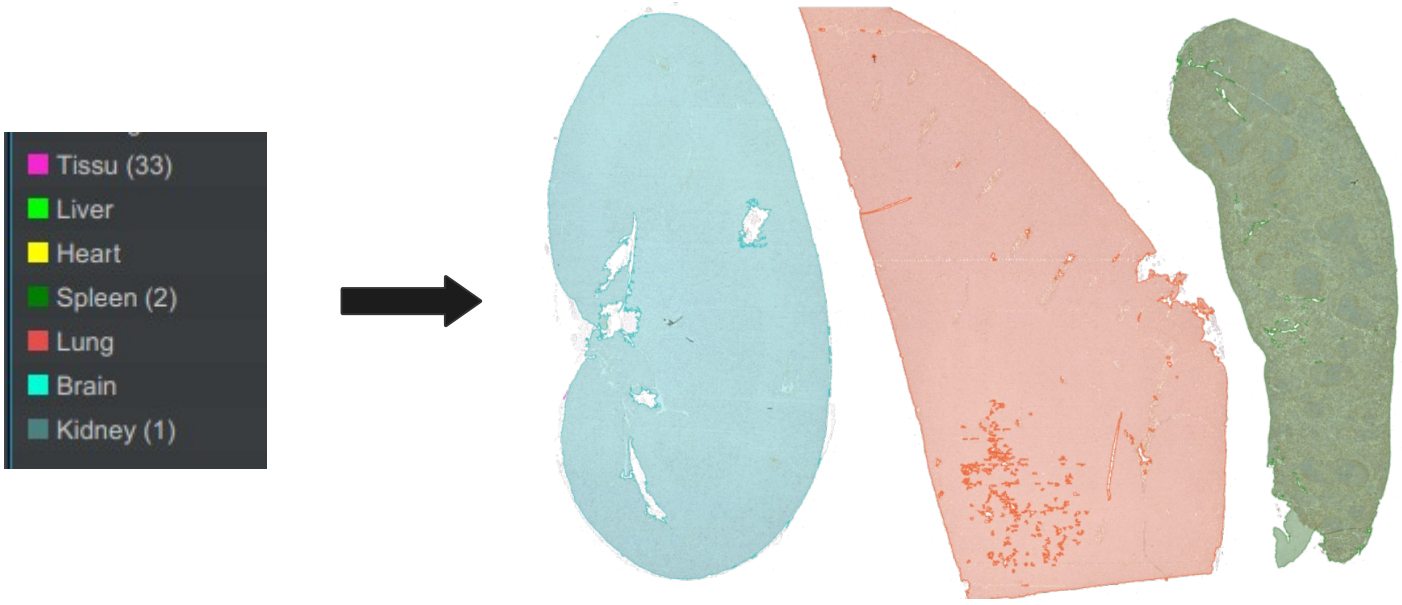
- Extract regions of interest (ROI) → set of annotations.



Introduction

Goals:

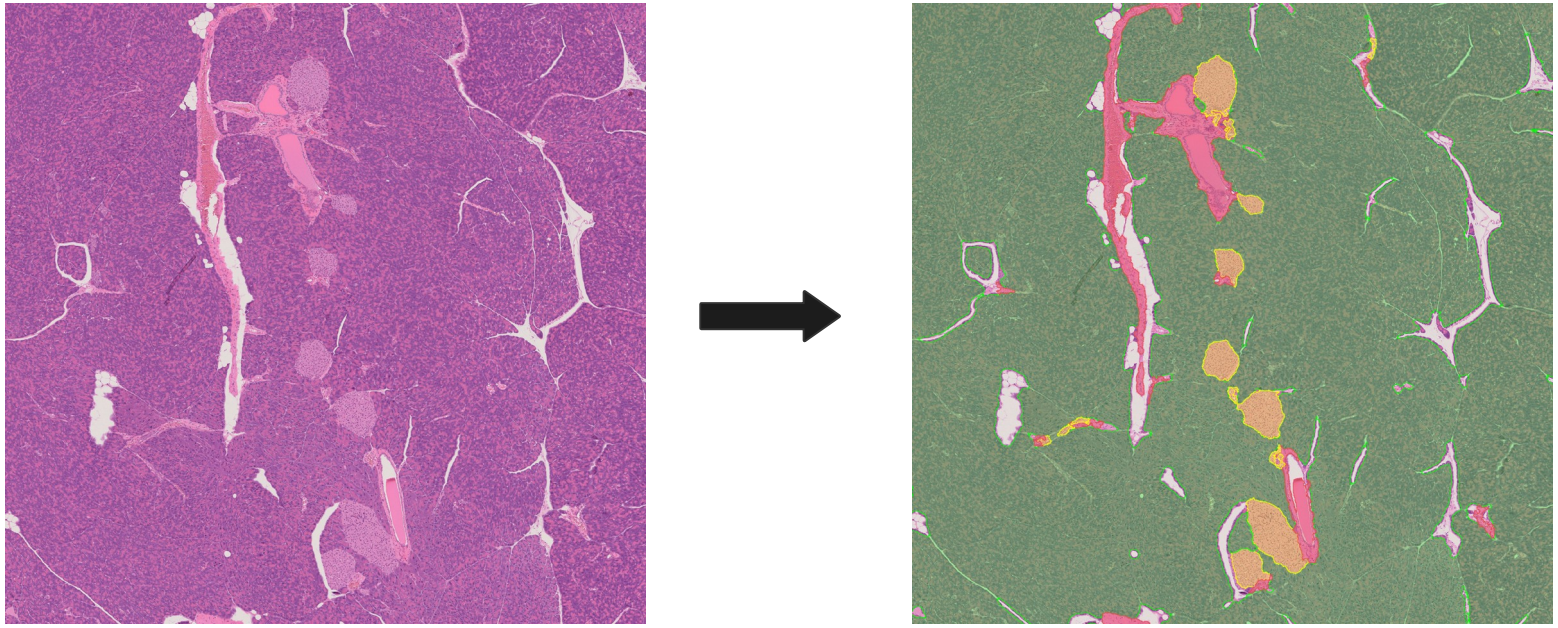
- Extract regions of interest (ROI) → set of annotations.
- Give them a class



Introduction

Goals:

- Extract regions of interest (ROI) → set of annotations.
- Give them a class (automatically if possible).



Introduction

Goals:

- Extract regions of interest (ROI) → set of annotations.
- Give them a class (automatically if possible).
- Extract measurements (area, positivity, …).

Key	Value
Image	slide-2024-02-14T11-50-32...
Object ID	11818cd8-f639-4ff8-a312-...
Object type	Annotation
Name	
Classification	Spleen
Parent	Root object (Image)
ROI	Geometry
Centroid X µm	3120.5199
Centroid Y µm	4121.0943
find-dab: Positive area µm^2	3091209.5
Area µm^2	16916744.52
Perimeter µm	53986.4

1. Manually

Methods:

- With **polygon**(s) to make the outline. 
- With the combo **brush** + **fill holes**. 
- With the **magic wand** at the correct resolution. 

→ **Exercise 4.1: Fully manual segmentation**

1. Manually

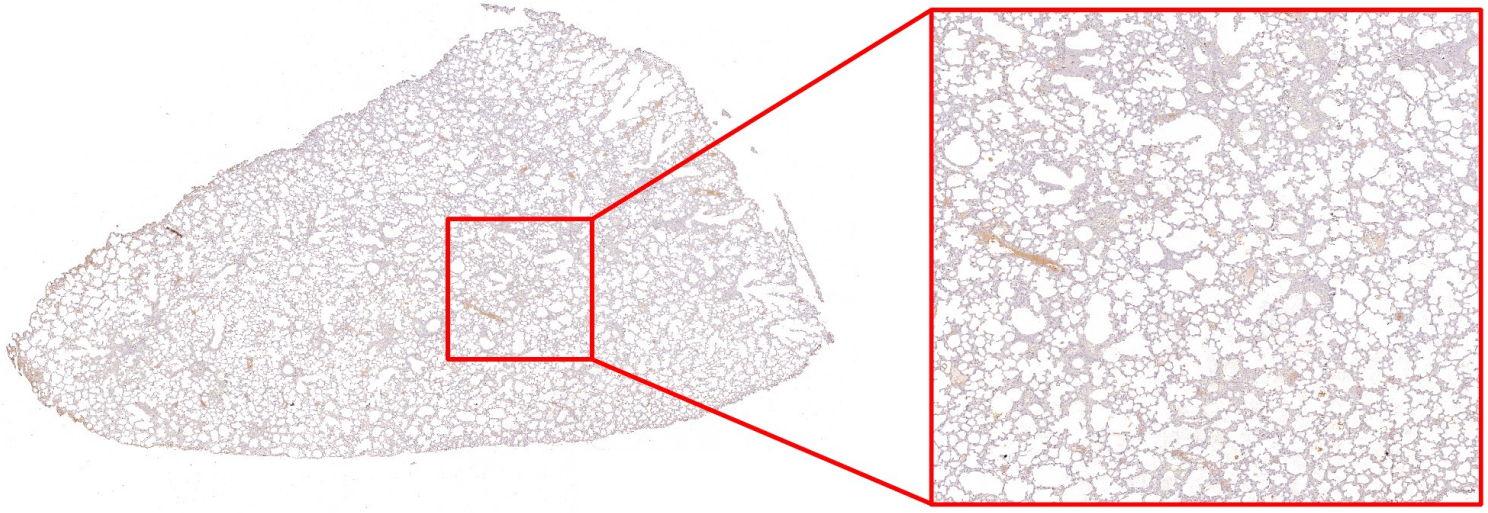
Methods:

- With **polygon**(s) to make the outline.
- With the combo **brush** + **fill holes**.
- With the **magic wand** at the correct resolution.

1. Manually

Methods:

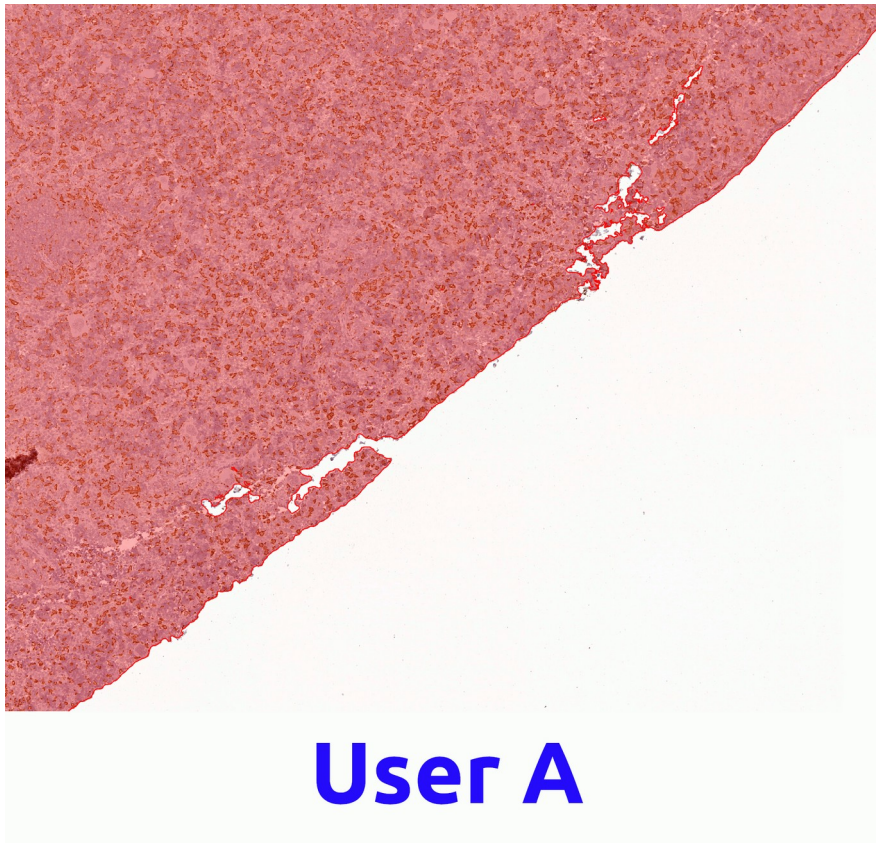
- With **polygon**(s) to make the outline.
- With the combo **brush** + **fill holes**.
- With the **magic wand** at the correct resolution.
  - Very **time-consuming**.



1. Manually

Methods:

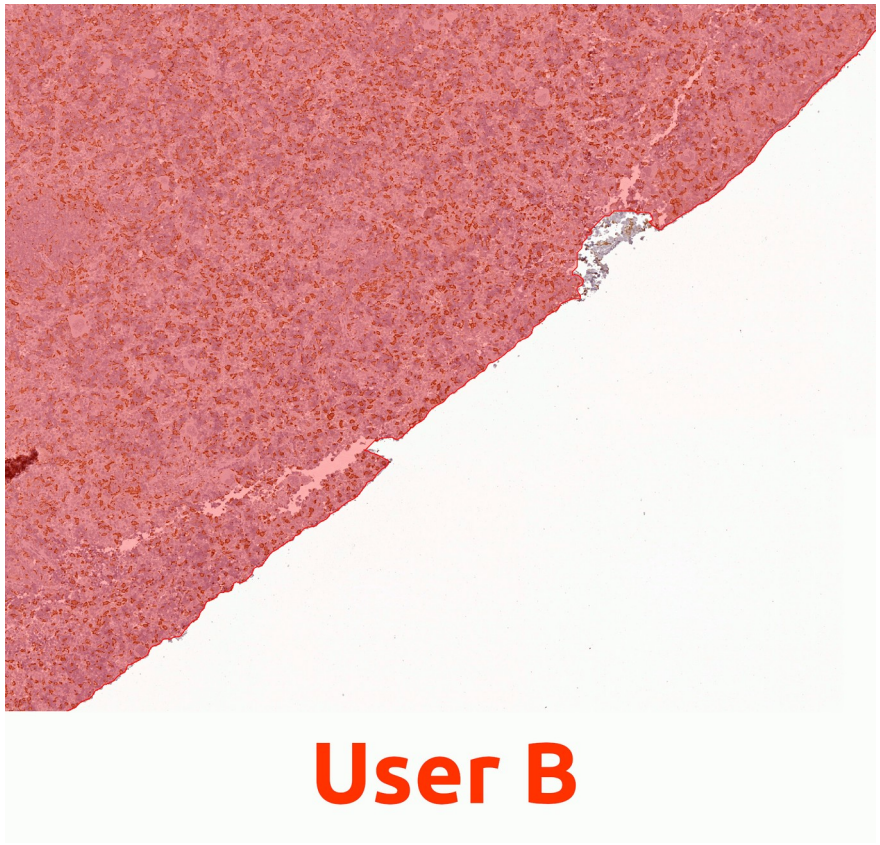
- With **polygon**(s) to make the outline.
- With the combo **brush** + **fill holes**.
- With the **magic wand** at the correct resolution.
  - Very **time-consuming**.
  - Introduces **user bias**.



1. Manually

Methods:

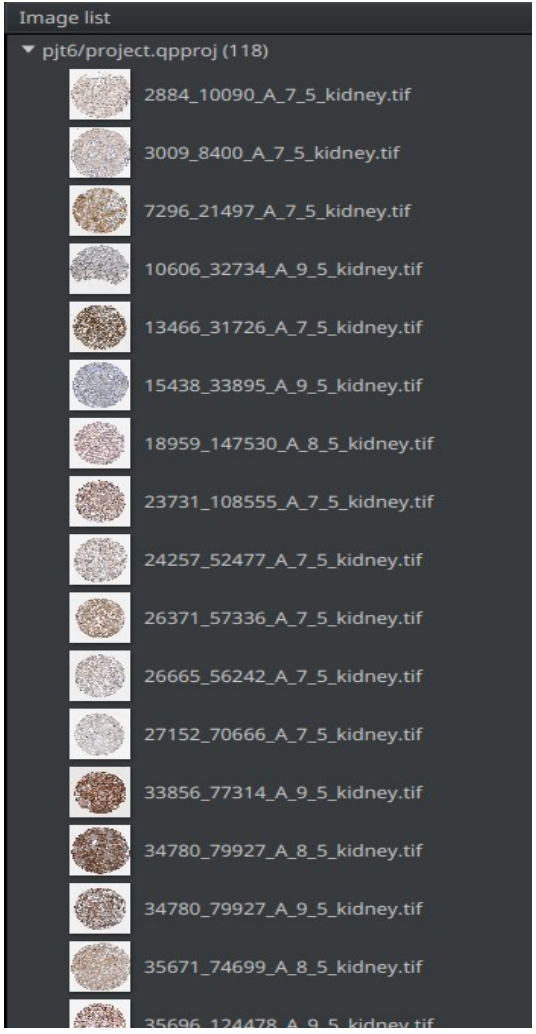
- With **polygon**(s) to make the outline.
- With the combo **brush** + **fill holes**.
- With the **magic wand** at the correct resolution.
  - Very **time-consuming**.
  - Introduces **user bias**.



1. Manually

Methods:

- With **polygon**(s) to make the outline.
- With the combo **brush** + **fill holes**.
- With the **magic wand** at the correct resolution.
  - Very **time-consuming**.
  - Introduces **user bias**.
  - Impossible for **large datasets**.



1. Manually

Methods:

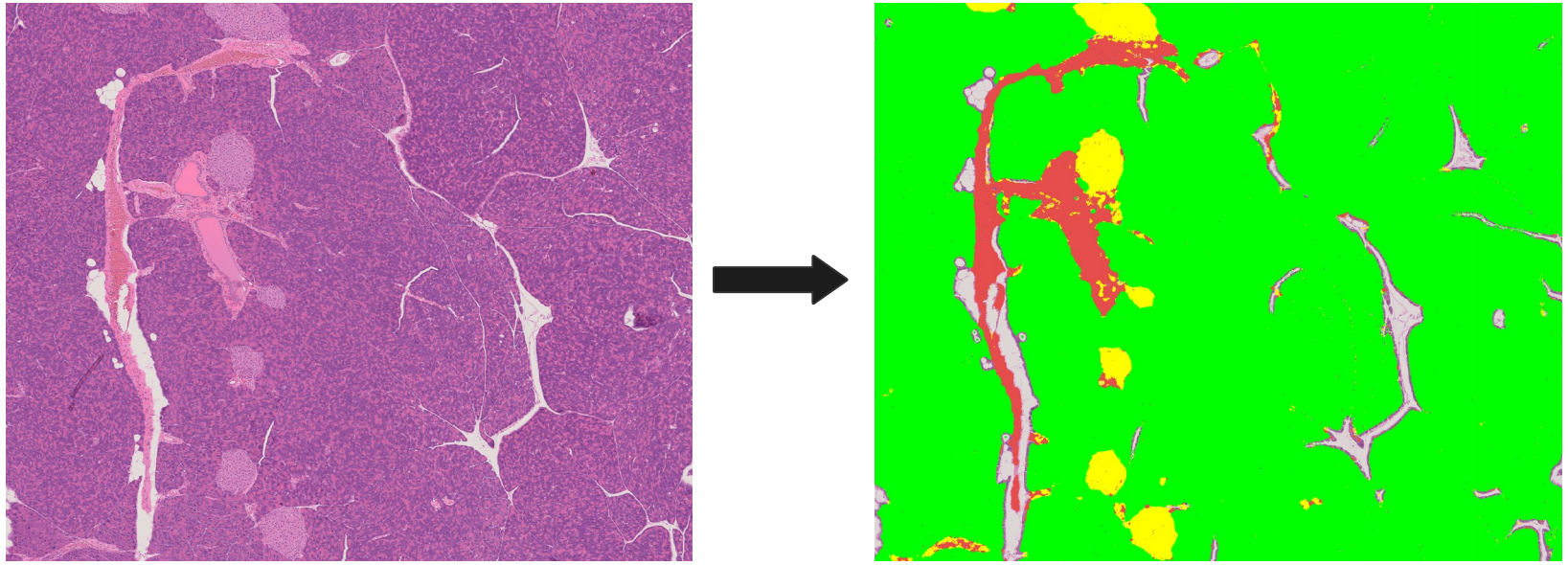
- With **polygon**(s) to make the outline.
- With the combo **brush** + **fill holes**.
- With the **magic wand** at the correct resolution.
  - Very **time-consuming**.
  - Introduces **user bias**.
  - Impossible for **large datasets**.

⇨ We need an automated method.

2. Pixel classifiers

General:

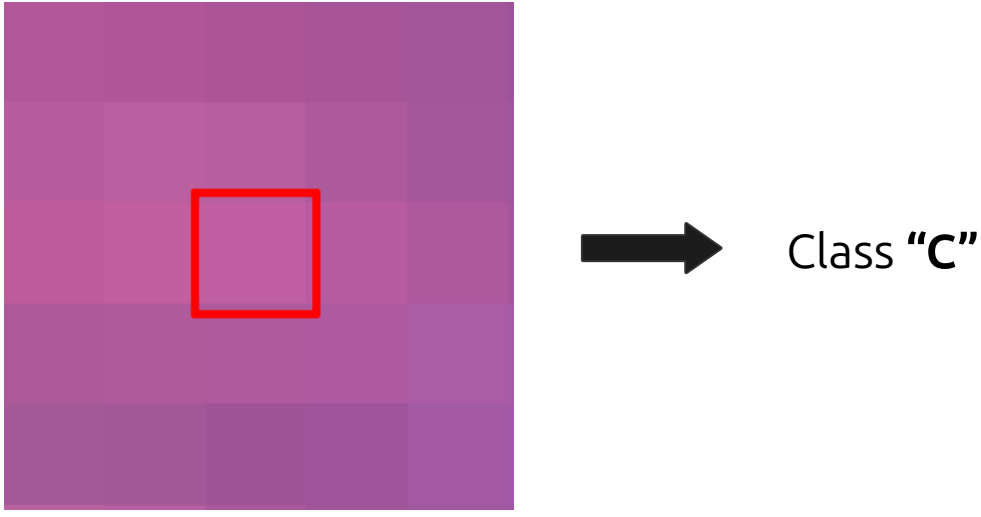
- Algorithm(a **pixel**) ↦ a **class**.



2. Pixel classifiers

General:

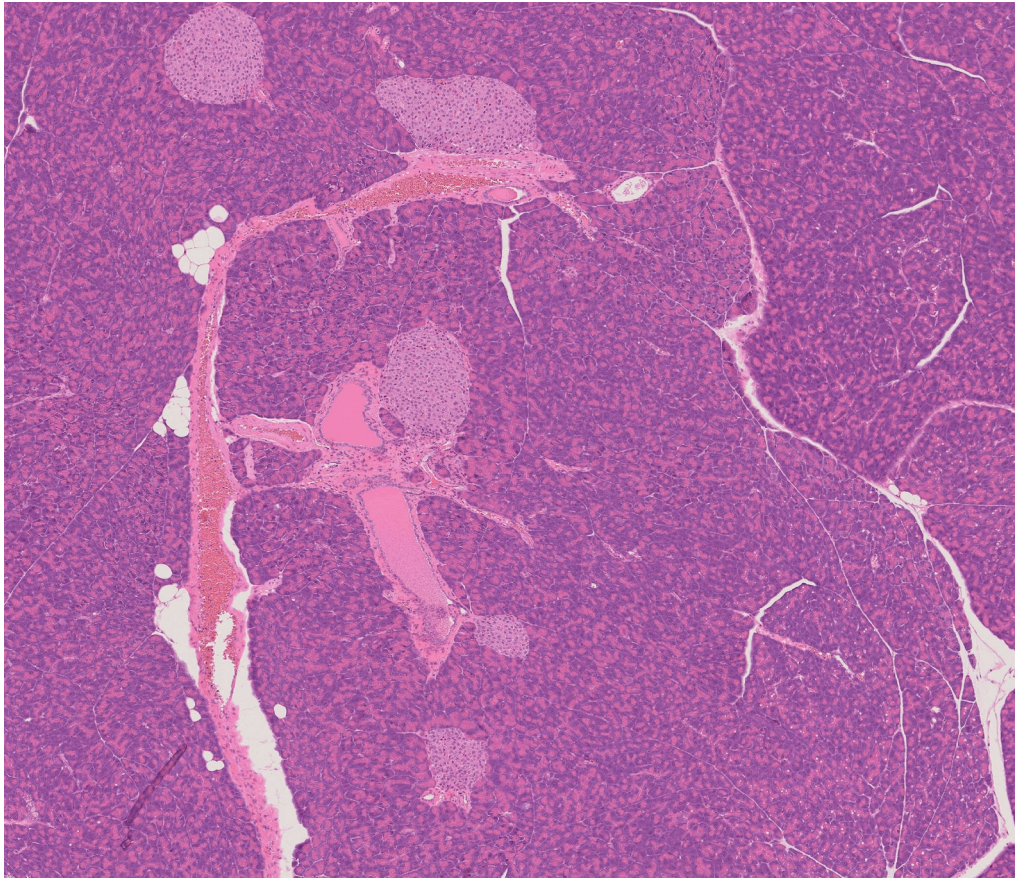
- Algorithm(a **pixel**) ↦ a **class**.
- Works **pixel-wise**, or limited neighborhood knowledge.



2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

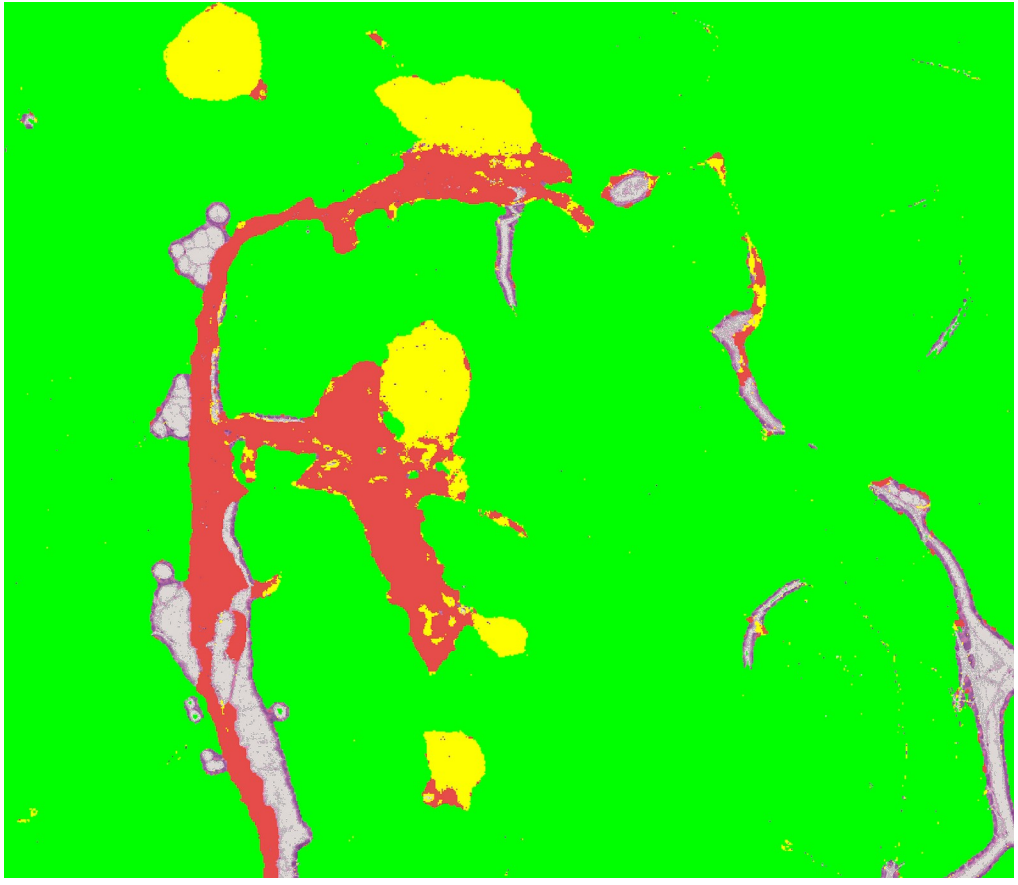
1. Take the input image at the chosen resolution.



2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

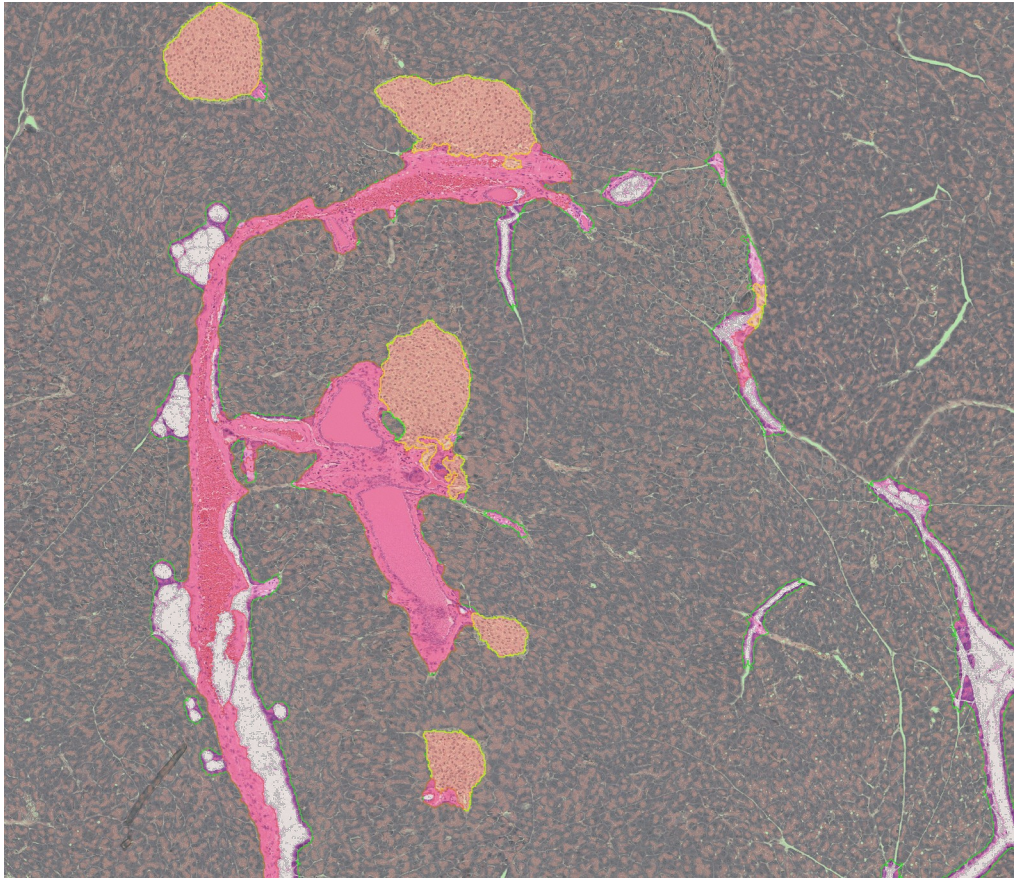
1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.



2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

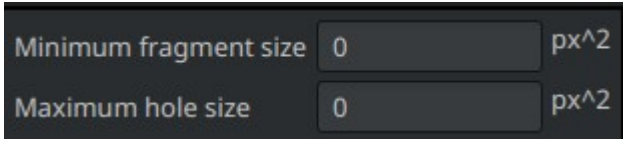
1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.
3. Extract the outline of each class **OR** count pixels.



2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.
3. Extract the outline of each class **OR** count pixels.
4. *Optional:* Fill holes and remove debris.



2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.
3. Extract the outline of each class **OR** count pixels.
4. *Optional:* Fill holes and remove debris.

Note: Pixel classification = transitive state → QuPath doesn't keep the classified image.

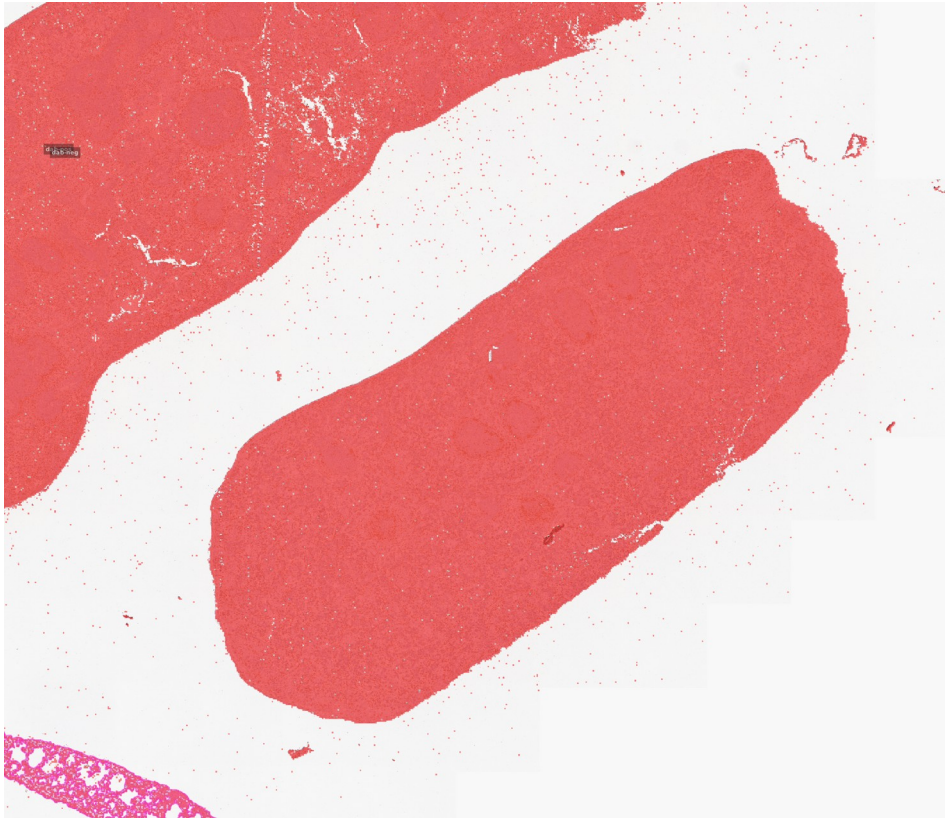
2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.
3. Extract the outline of each class **OR** count pixels.
4. *Optional:* Fill holes and remove debris.

Use cases:

- Segment whole objects (lower resolutions).



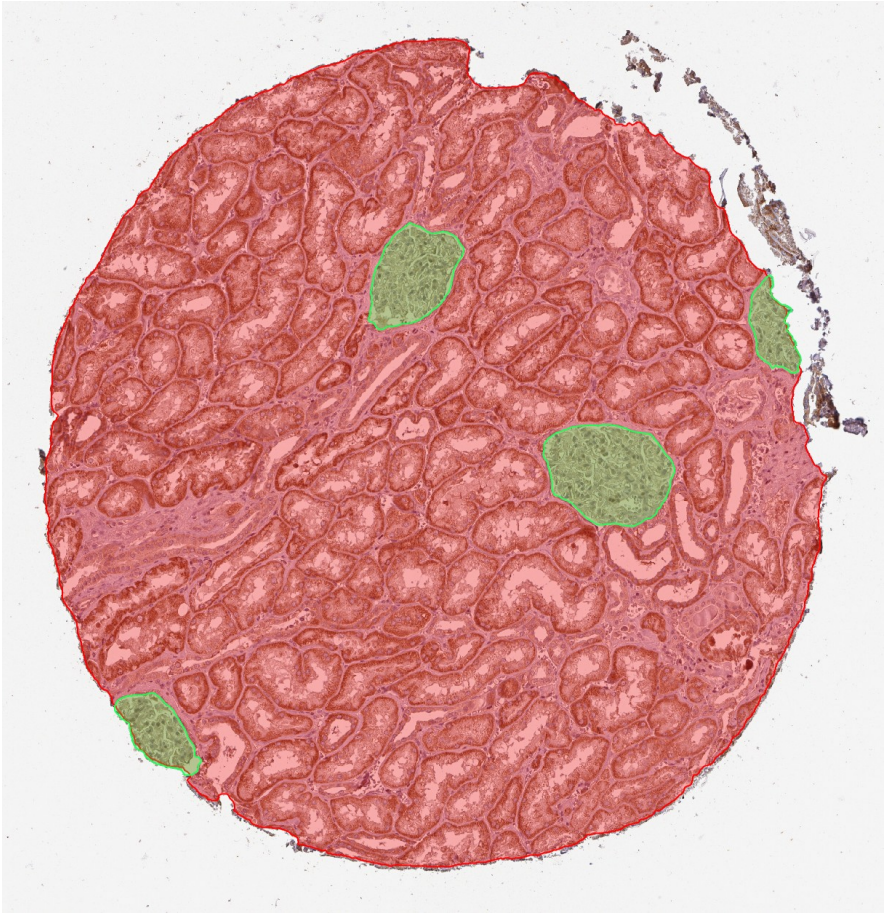
2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.
3. Extract the outline of each class **OR** count pixels.
4. *Optional:* Fill holes and remove debris.

Use cases:

- Segment whole objects (lower resolutions).
- Segment sub-structures (intermediate resolutions).



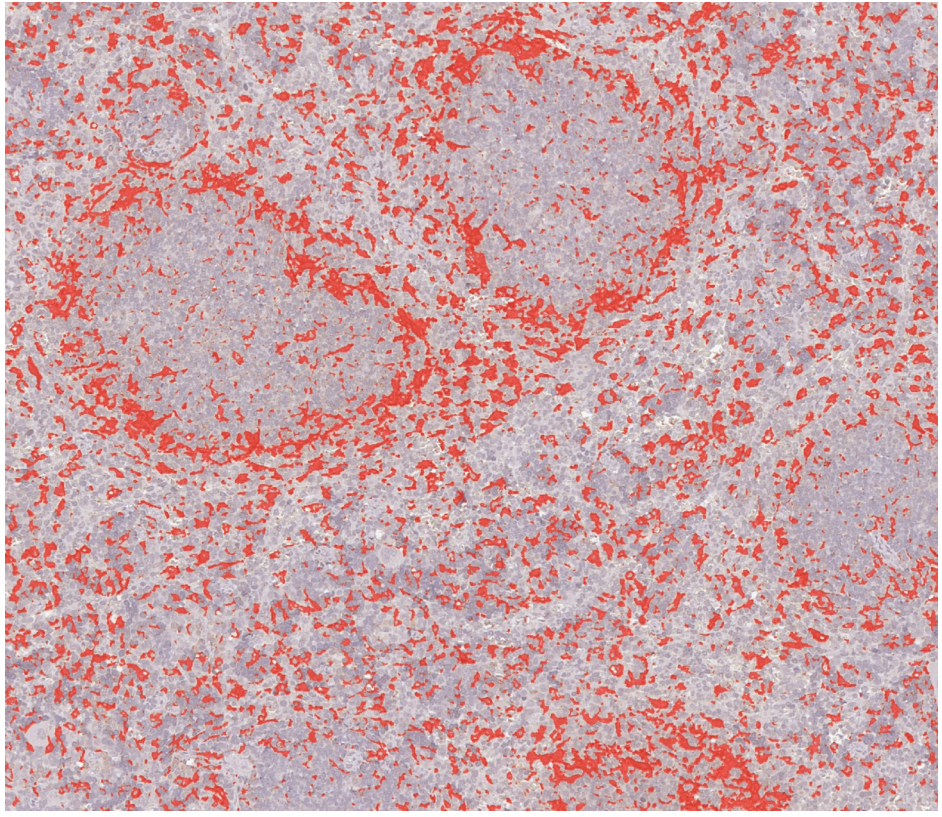
2. Pixel classifiers

Workflow: ⚠️ Under the hood: done automatically

1. Take the input image at the chosen resolution.
2. Apply the pixel classifier to each pixel.
3. Extract the outline of each class **OR** count pixels.
4. *Optional:* Fill holes and remove debris.

Use cases:

- Segment whole objects (lower resolutions).
- Segment sub-structures (intermediate resolutions).
- Measure area of staining (higher resolutions).



2. Pixel classifiers

A. Thresholders

Principle:

- **Most basic** classifier: only **two classes** *(true/false, FG/BG, pos/neg, …)* == **bi-partition**.

2. Pixel classifiers

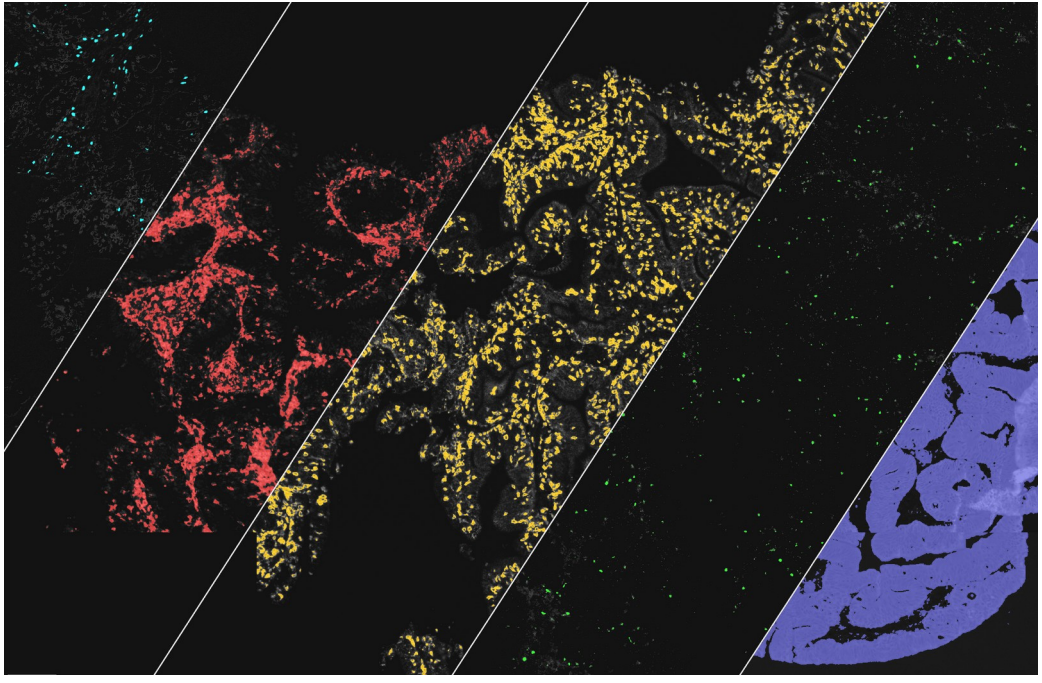
A. Thresholders

Principle:

- **Most basic** classifier: only **two classes** (*true*/*false*, *FG*/*BG*, *pos*/*neg*, …) == **bi-partition**.

Input:

- `[Fluo]` Any channel.



2. Pixel classifiers

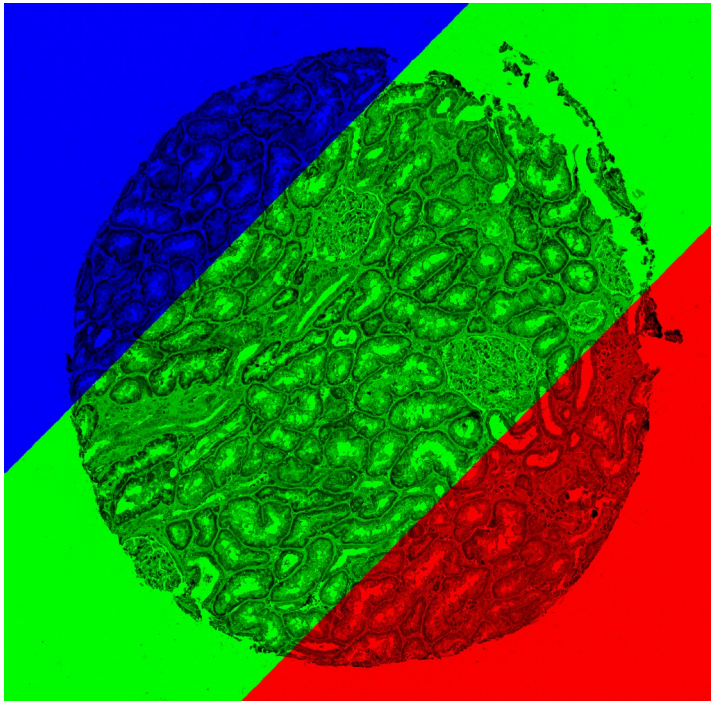
A. Thresholders

Principle:

- **Most basic** classifier: only **two classes** *(true/false, FG/BG, pos/neg, …)* == **bi-partition**.

Input:

- `[Fluo]` Any channel.
- `[IHC]` Red, green or blue channel.



2. Pixel classifiers

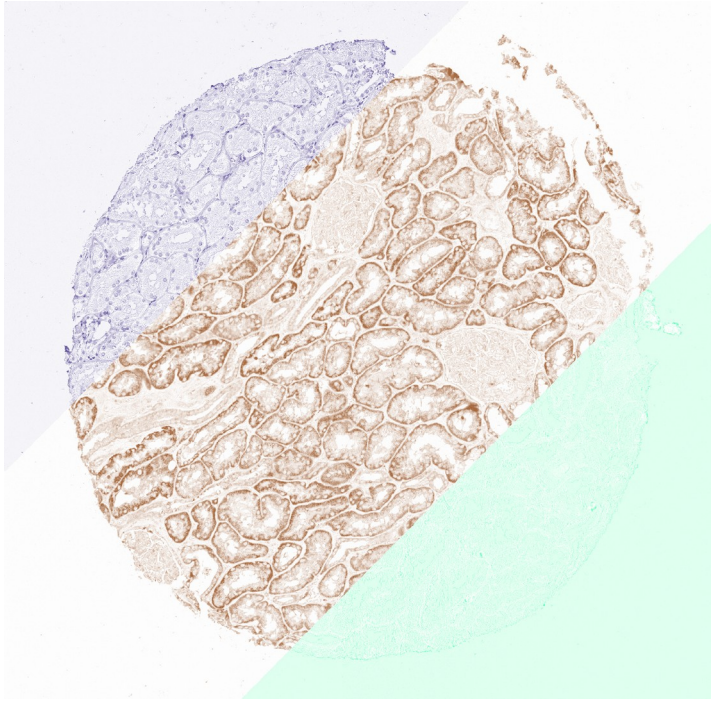
A. Thresholders

Principle:

- **Most basic** classifier: only **two classes** (*true*/*false*, *FG*/*BG*, *pos*/*neg*, …) == **bi-partition**.

Input:

- `[Fluo]` Any channel.
- `[IHC]` Red, green or blue channel.
- `[IHC]` Any deconvoluted channel.



2. Pixel classifiers

A. Thresholders

Principle:

- **Most basic** classifier: only **two classes** *(true/false, FG/BG, pos/neg, …)* == **bi-partition**.

Input:

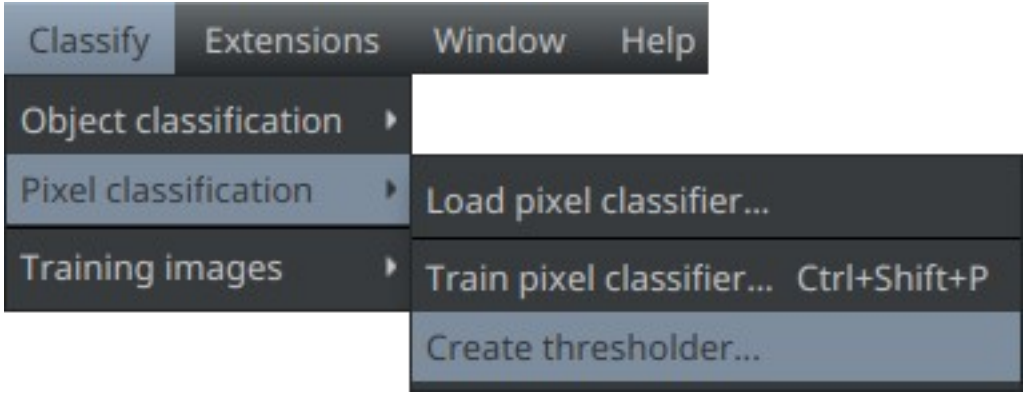
- `[Fluo]` Any channel.
- `[IHC]` Red, green or blue channel.
- `[IHC]` Any deconvoluted channel.
- `[Both]` Possibly pre-processed:
  - Projection (min/max/avg/…) of all channels
  - Filtered (Gaussian, Laplacial of Gaussian, gray morphology, ...)

2. Pixel classifiers

A. Thresholders

In QuPath:

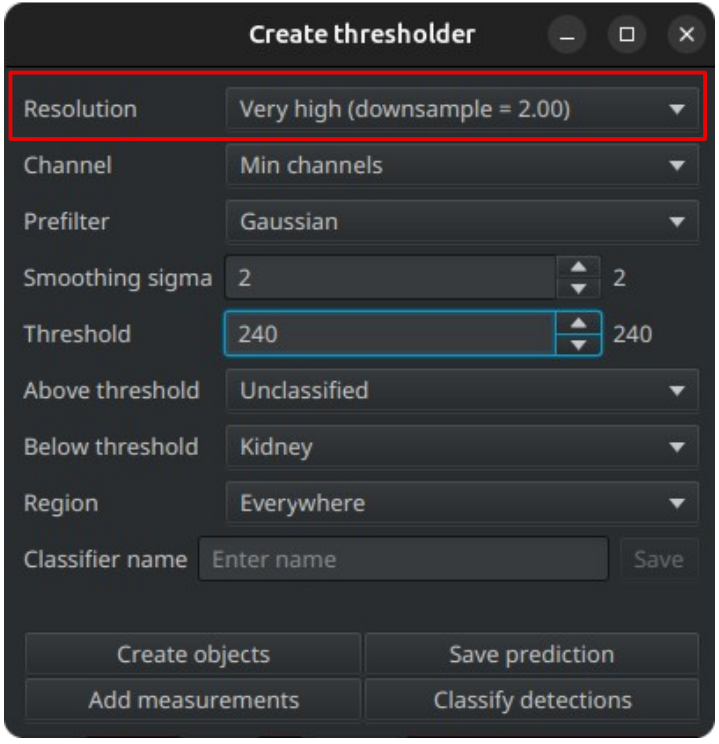
- Can be found in “Classify” > “Pixel classification” > “Create thresholder”.



2. Pixel classifiers

A. Thresholders

In QuPath:



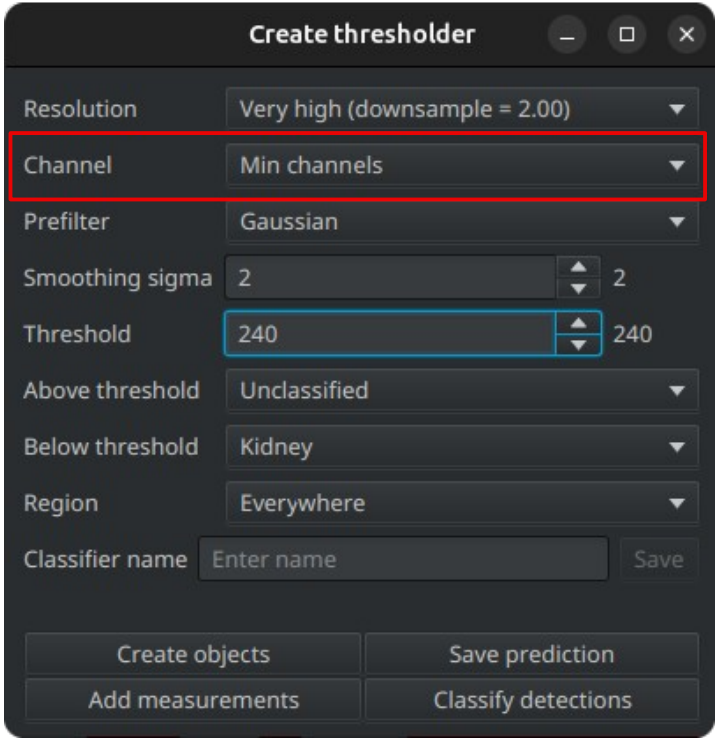
Resolution:

- Depending on the object, you don't need the full resolution.
  - Full organ: **low** resolution.
  - Sub-structure: **medium** resolution.
  - Spots/filaments: **full** resolution.

2. Pixel classifiers

A. Thresholders

In QuPath:



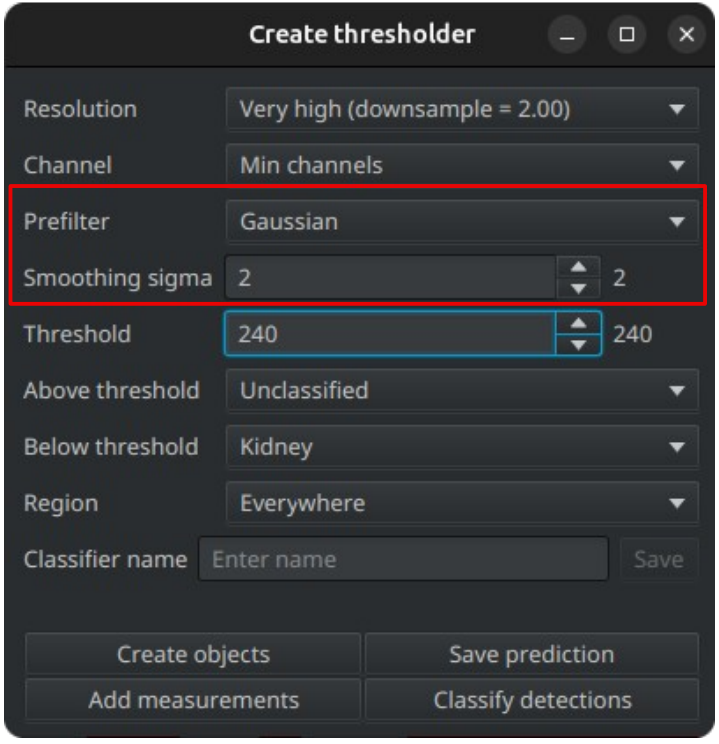
Channel:

- Which channel to use?
  - A precise channel (fluo / RGB)
  - A deconvoluted channel (H, DAB, E, …)
  - An aggregate of channels (min channel, max channel)

2. Pixel classifiers

A. Thresholders

In QuPath:



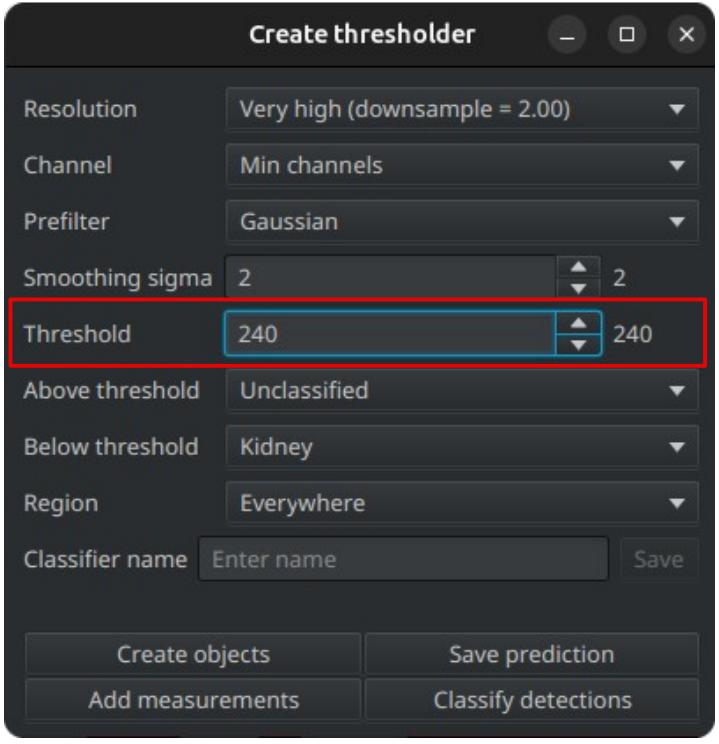
Prefilter & sigma:

- Prefilter the channel before thresholding.
  - Denoise
  - Highlight some structures
- Smoothing sigma
  - Sigma used for the prefilter
  - 0: prefilter deactivated

2. Pixel classifiers

A. Thresholders

In QuPath:



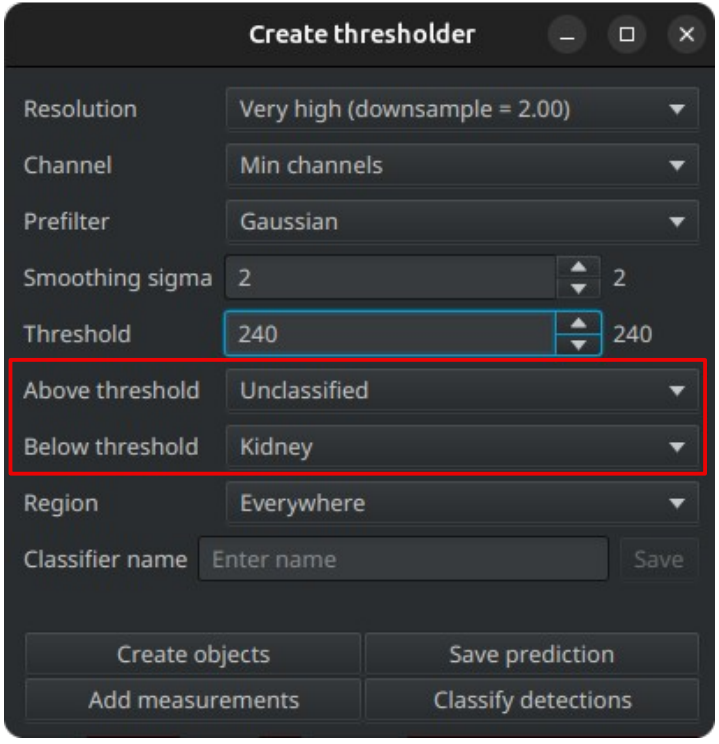
Thresold:

- Intensity threshold

2. Pixel classifiers

A. Thresholders

In QuPath:



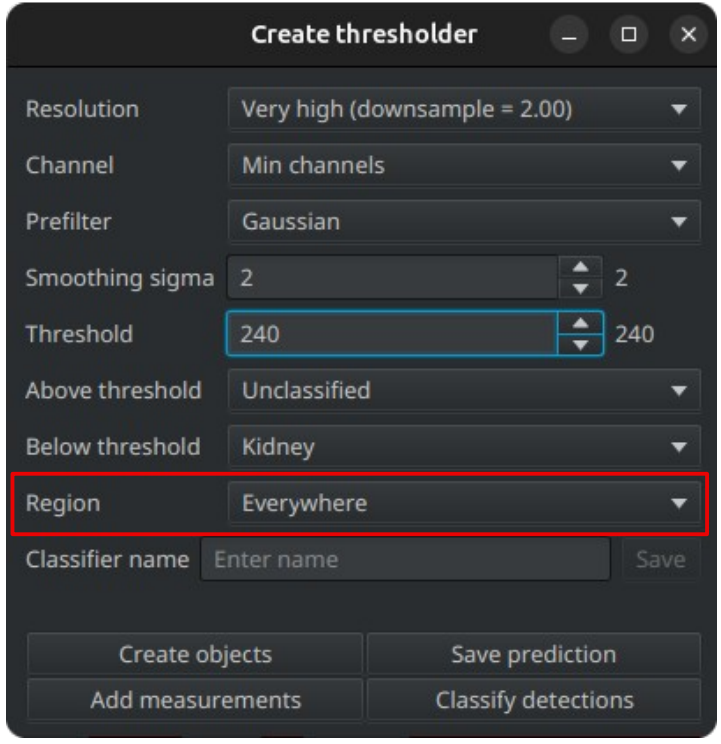
Above and below threshold:

- Class given to the pixel depending if its smaller or bigger than the threshold.
  - IHC: Object below threshold (light background)
  - Fluo: Object above threshold (dark background)

2. Pixel classifiers

A. Thresholders

In QuPath:



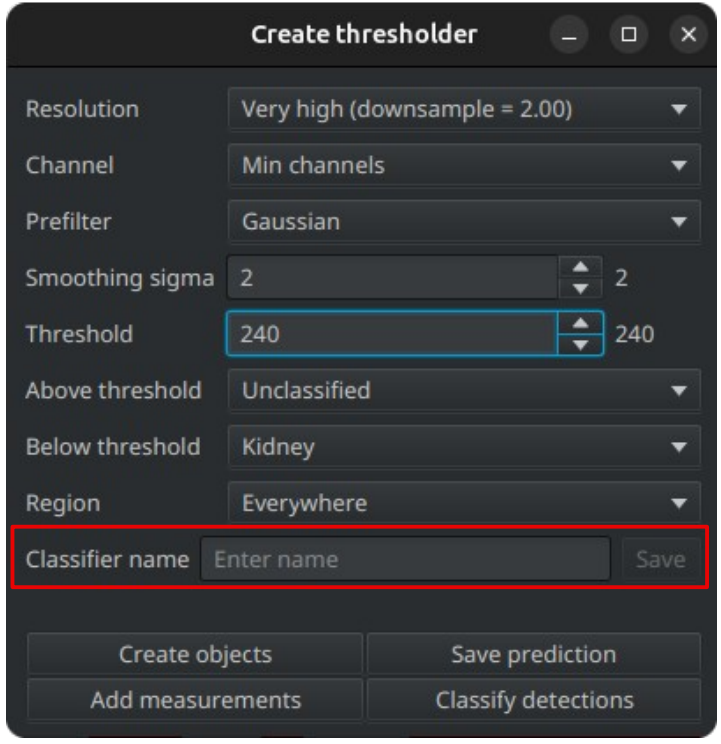
Region:

- Where to apply the threshold
  - Everywhere
  - In the currant annotation

2. Pixel classifiers

A. Thresholders

In QuPath:



Classifier name:

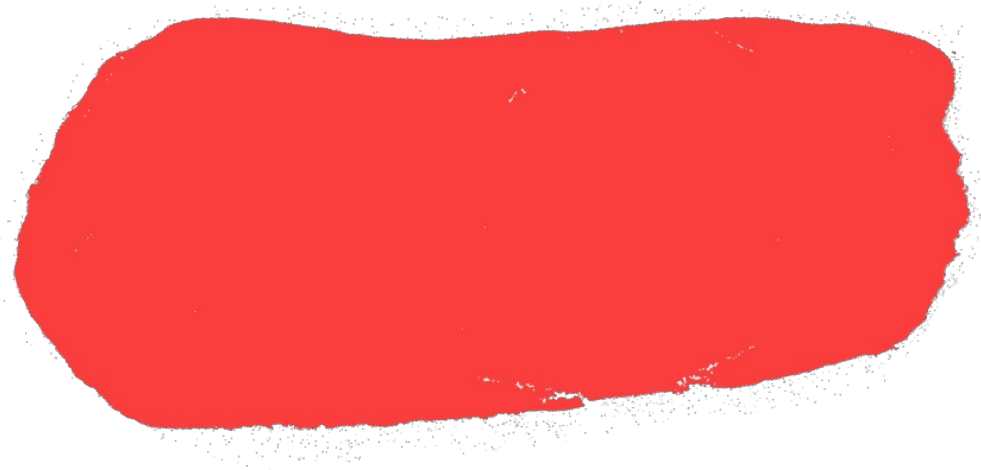
- Name of the classifier so it can:
  - Be reused later
  - Be called from a script

2. Pixel classifiers

A. Thresholders

→ The result looks great!

→ Let’s try our thresholder on another image.



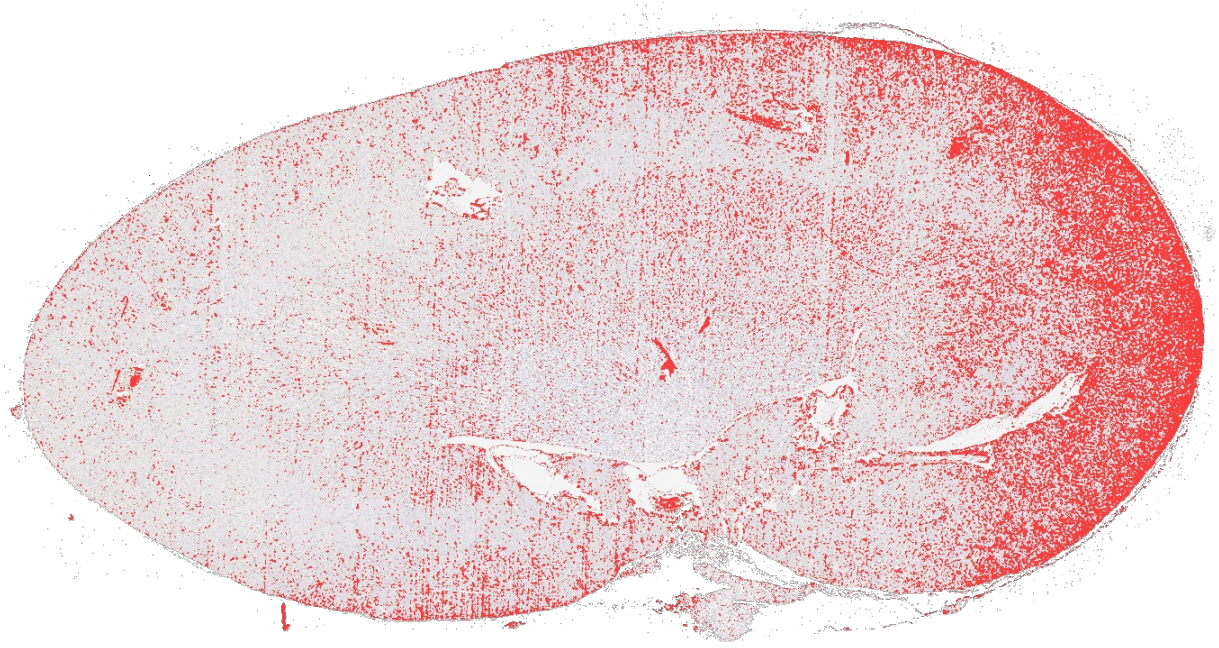
→ Exercise 4.2: Using a threshold pixel classifier

2. Pixel classifiers

A. Thresholders

→ The result looks great!

→ Let’s try our thresholder on another image.



2. Pixel classifiers

A. Thresholders

Drawbacks:

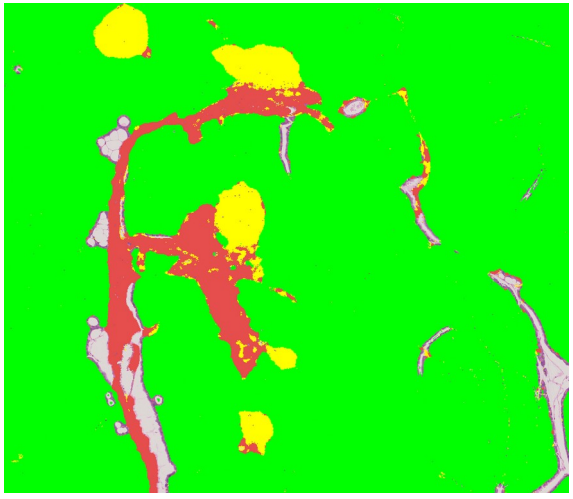
- Can't generalize: the value is fixed
- Sensible to field non-uniformity

Examples:

- You changed of machine / lamp / laser / exposure time / …
- Some organs are lighter than others.

2. Pixel classifiers

A. Thresholders



Drawbacks:

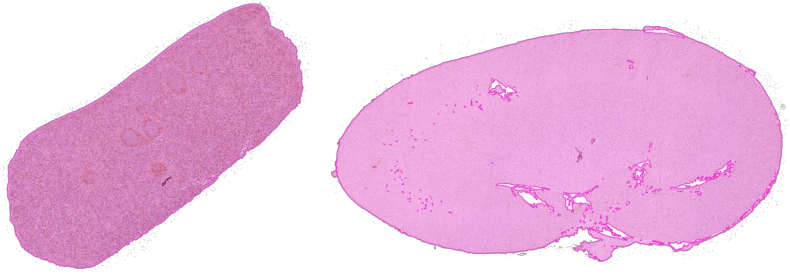
- Can't generalize: the value is fixed
- Sensible to field non-uniformity

Examples:

- You changed of machine / lamp / laser / exposure time / …
- Some organs are lighter than others.

We would like…

- More than two classes.
- Something not sensible to intensities.

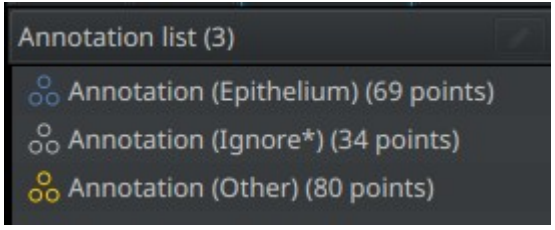


2. Pixel classifiers

B. Random trees

Principle:

- **Machine learning** algorithm.
- Requires **manual examples** → a **points cloud per class**.

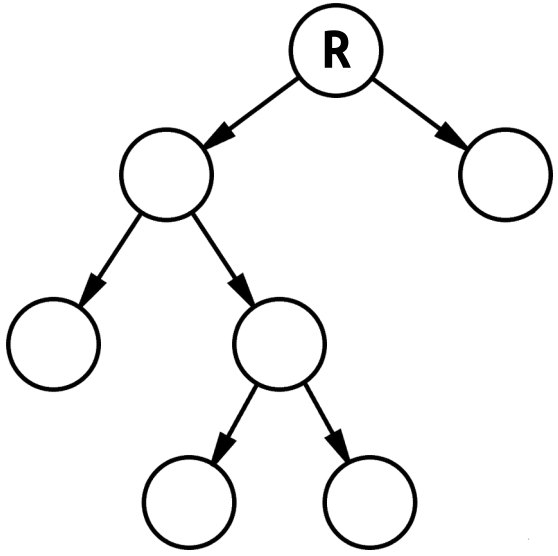


2. Pixel classifiers

B. Random trees

Principle:

- **Machine learning** algorithm.
- Requires **manual examples** → a **points cloud per class**.
- **Random-trees** == collection of **decision trees**.
  - Graph with a **root**, nodes have **0 or 2 children**.

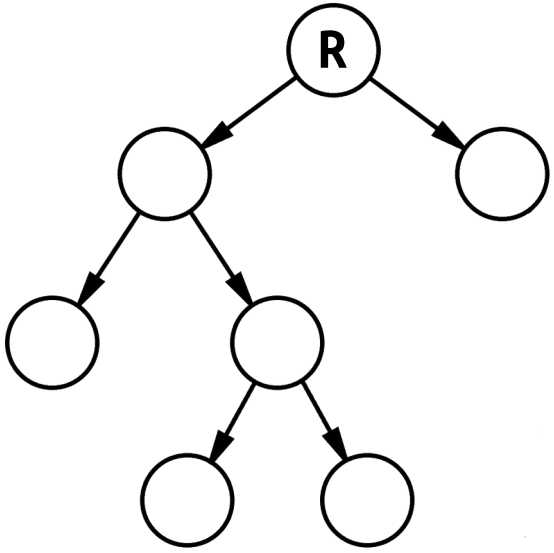


2. Pixel classifiers

B. Random trees

Principle:

- **Machine learning** algorithm.
- Requires **manual examples** → a **points cloud per class**.
- **Random-trees** == collection of **decision trees**.
  - Graph with a **root**, nodes have **0 or 2 children**.
  - Asks a **question**, the answer is an **item from a set**.



What fruit is it?

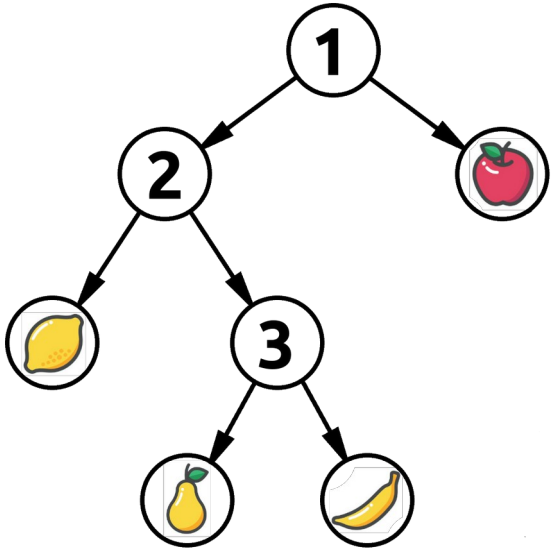
Outputs: {🍐, 🍋, 🍌, 🍎}

2. Pixel classifiers

B. Random trees

Principle:

- **Machine learning** algorithm.
- Requires **manual examples** → a **points cloud per class**.
- **Random-trees** == collection of **decision trees**.
  - Graph with a **root**, nodes have **0 or 2 children**.
  - Asks a **question**, the answer is an **item from a set**.
  - Each **leaf** corresponds to **an answer**.



What fruit is it?

Outputs: {🍐, 🍋, 🍌, 🍎}

2. Pixel classifiers

B. Random trees

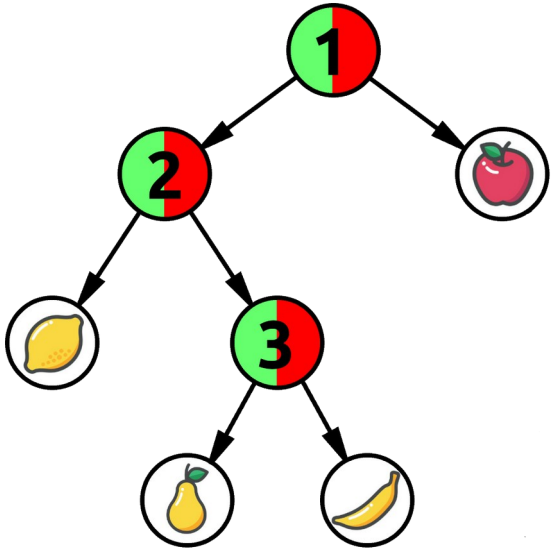
Principle:

- **Machine learning** algorithm.
- Requires **manual examples** → a **points cloud per class**.
- **Random-trees** == collection of **decision trees**.
  - Graph with a **root**, nodes have **0 or 2 children**.
  - Asks a **question**, the answer is an **item from a set**.
  - Each **leaf** corresponds to **an answer**.
  - Navigate through with **Yes**/**No** **questions**.

    1) Is it yellow?

    2) Is it sour?

    3) Is it easy to peel?



Outputs: {🍐, 🍋, 🍌, 🍎}

What fruit is it?

2. Pixel classifiers

B. Random trees

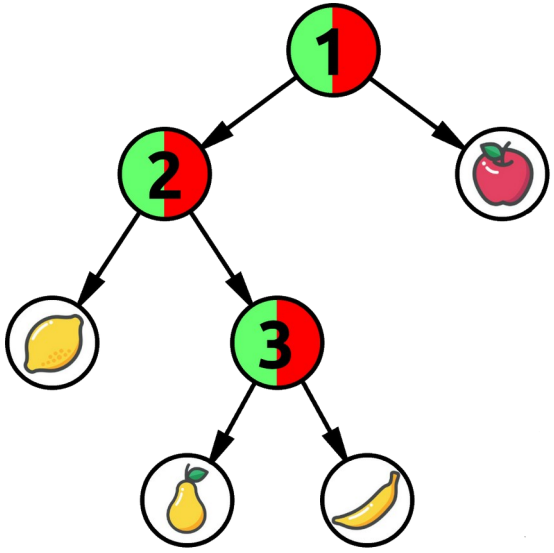
Principle:

- **Machine learning** algorithm.
- Requires **manual examples** → a **points cloud per class**.
- **Random-trees** == collection of **decision trees**.
  - Graph with a **root**, nodes have **0 or 2 children**.
  - Asks a **question**, the answer is an **item from a set**.
  - Each **leaf** corresponds to **an answer**.
  - Navigate through with **Yes**/**No** **questions**.

    1) Is it yellow?

    2) Is it sour?

    3) Is it easy to peel?
- **At prediction**: look at the input and answer the questions.



Outputs: {🍐, 🍋, 🍌, 🍎}

What fruit is it?

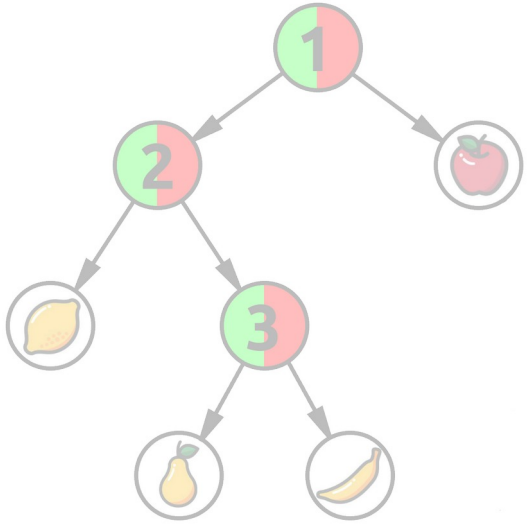
2. Pixel classifiers

B. Random trees

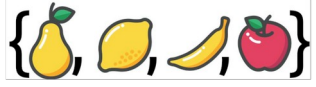
Example:

- **Input:** 

1) Is it yellow?
2) Is it sour?
3) Is it easy to peel?



What fruit is it?

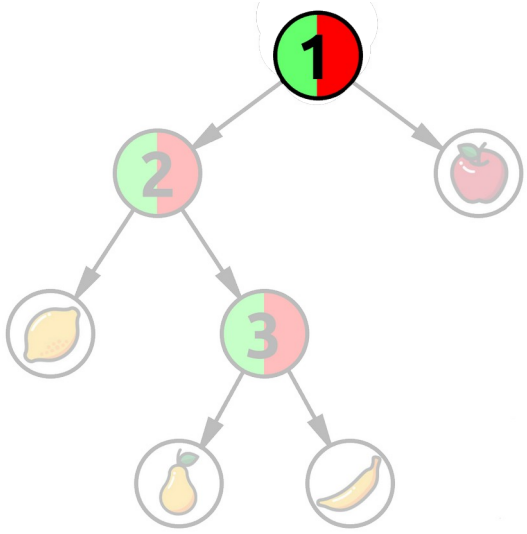
Outputs: 

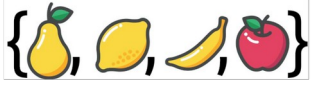
2. Pixel classifiers

B. Random trees

Example:

- **Input:** 
  1) Is it yellow?
  2) Is it sour?
  3) Is it easy to peel?



Outputs: 

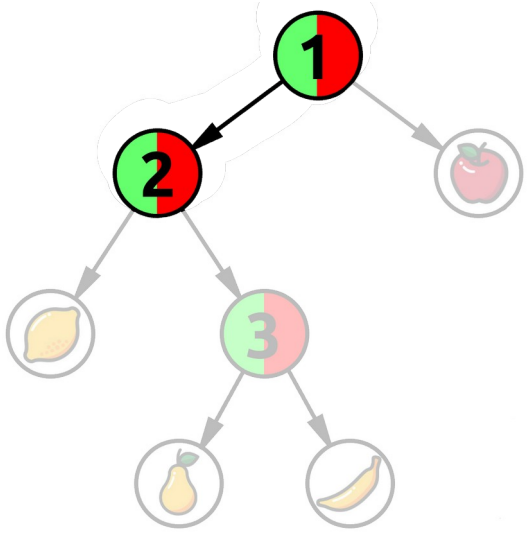
What fruit is it?

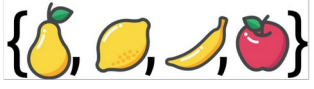
2. Pixel classifiers

B. Random trees

Example:

- **Input:** 
  1) **Is it yellow?**
  2) Is it sour?
  3) Is it easy to peel?



Outputs: 

What fruit is it?

2. Pixel classifiers

B. Random trees

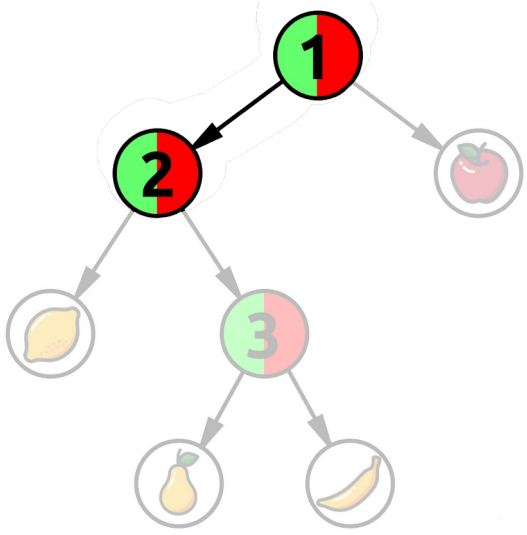
Example:

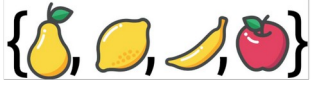
- **Input:** 

  1) Is it yellow?

  2) Is it sour?

  3) Is it easy to peel?



Outputs: 

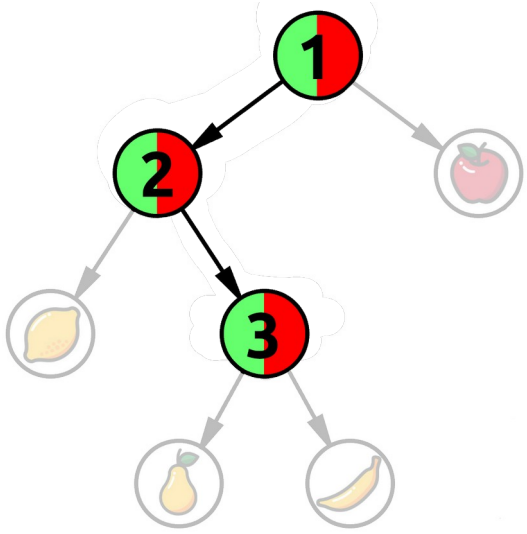
What fruit is it?

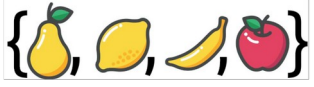
2. Pixel classifiers

B. Random trees

Example:

- **Input:** 
  1) Is it yellow?
  2) Is it sour?
  3) Is it easy to peel?



Outputs: 

What fruit is it?

2. Pixel classifiers

B. Random trees

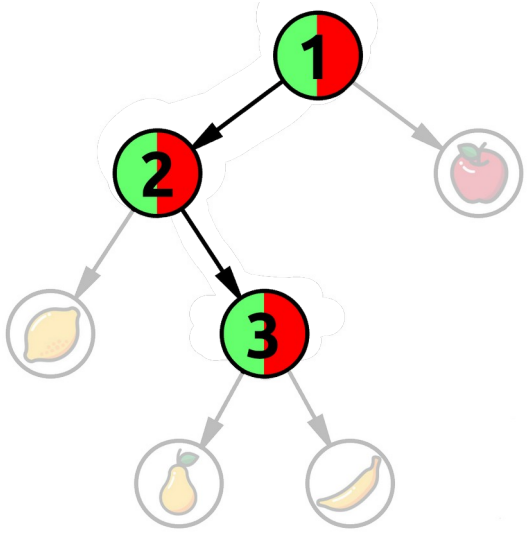
Example:

- **Input:** 

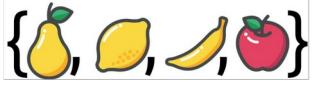
  1) Is it yellow?

  2) Is it sour?

  3) Is it easy to peel?



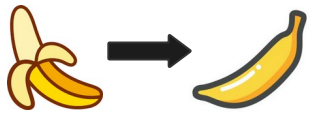
What fruit is it?

Outputs: 

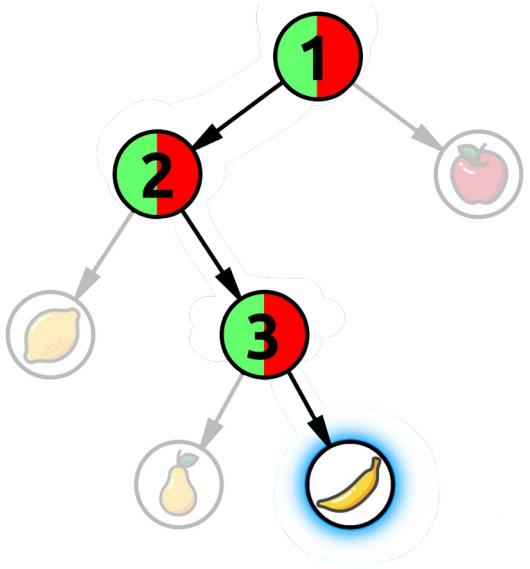
2. Pixel classifiers

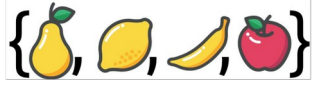
B. Random trees

Example:



- **Input:**
  1) Is it yellow?
  2) Is it sour?
  3) Is it easy to peel?



Outputs: 

What fruit is it?

2. Pixel classifiers

B. Random trees

Features vector:

- Here, we had **three features**, each one had **a value**.

Question	Feature	Values set	Value	Answer
Is it yellow?	Color	Yellow, red, blue, green, pink, ...	Yellow	Yes
Is it sour?	Taste	Sour, sweet, bitter, salty, ...	Sour	No
Is it easy to peel?	Peeling difficulty	Neutral, easy, hard, ...	Easy	Yes

2. Pixel classifiers

B. Random trees

Features vector:

- Here, we had **three features**, each one had **a value**.

Question	Feature	Values set	Value	Answer
Is it yellow?	Color	Yellow, red, blue, green, pink, ...	Yellow	Yes
Is it sour?	Taste	Sour, sweet, bitter, salty, ...	Sour	No
Is it easy to peel?	Peeling difficulty	Neutral, easy, hard, ...	Easy	Yes

- What features for **an image**?
  - Intensity? → Similar output to a “Thresholder”…

2. Pixel classifiers

B. Random trees

Features vector:

- Here, we had **three features**, each one had **a value**.

Question	Feature	Values set	Value	Answer
Is it yellow?	Color	Yellow, red, blue, green, pink, ...	Yellow	Yes
Is it sour?	Taste	Sour, sweet, bitter, salty, ...	Sour	No
Is it easy to peel?	Peeling difficulty	Neutral, easy, hard, ...	Easy	Yes

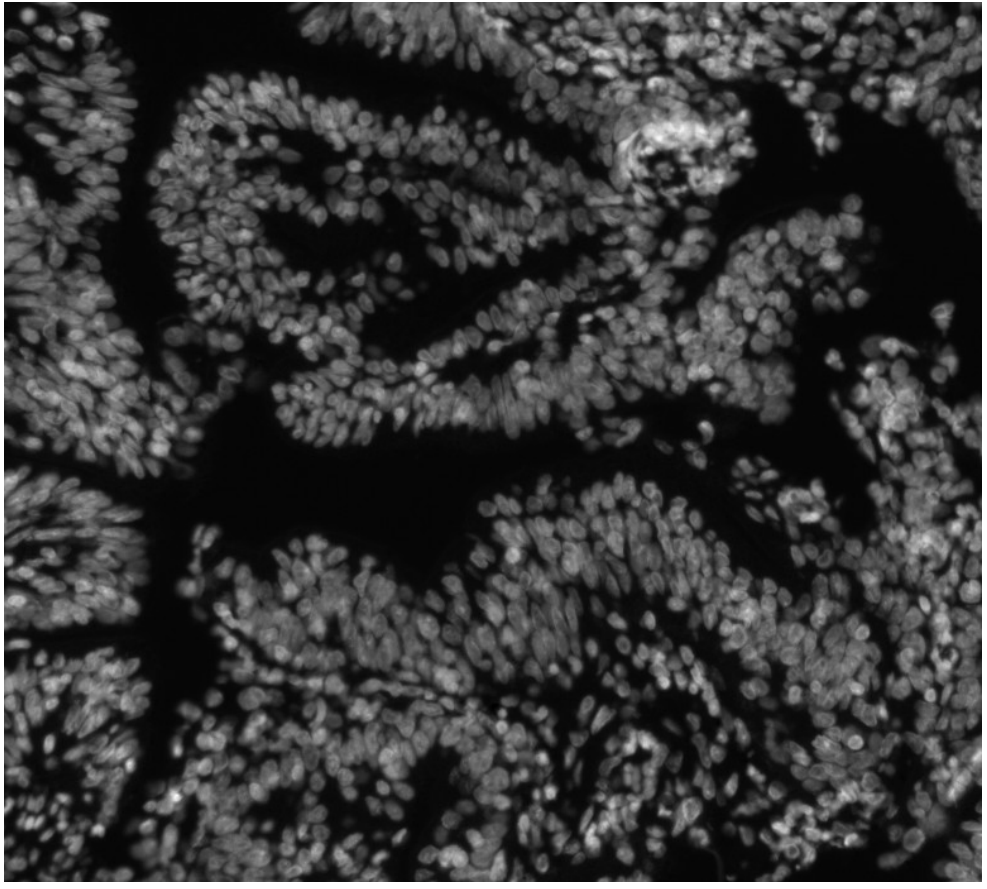
- What features for **an image**?
  - Intensity? → Similar output to a “Thresholder”…
  - Use the result of **filters**!

2. Pixel classifiers

B. Random trees

Features vector:

- Principle:
  - Take the input image.

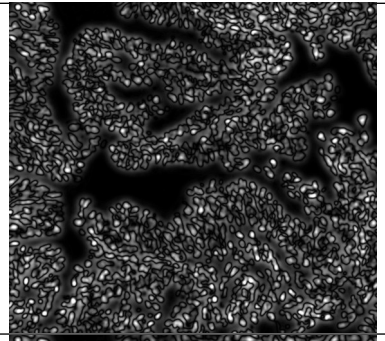
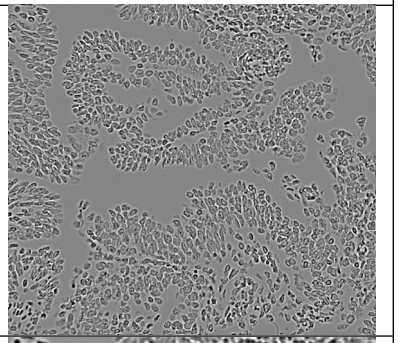
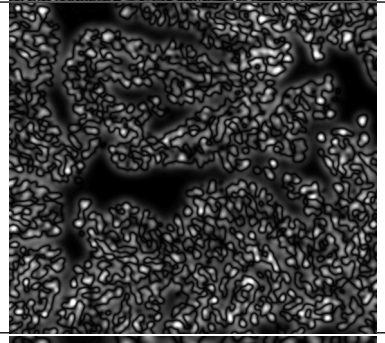
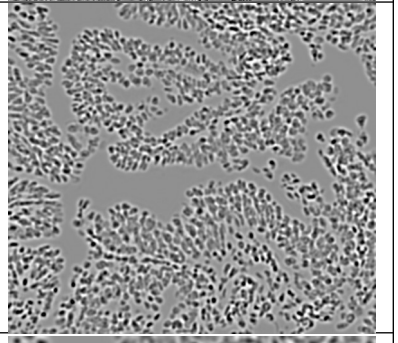
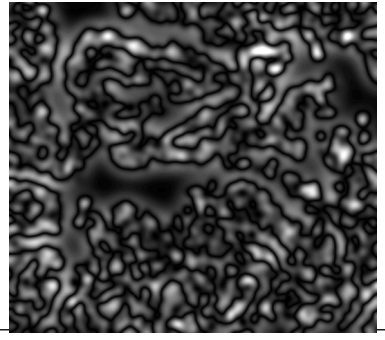
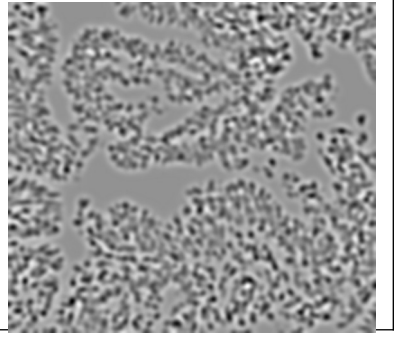


2. Pixel classifiers

B. Random trees

Features vector:

- Principle:
  - Take the input image.
  - Apply different filters at different radii.

	Difference of Gaussian	Laplacian of Gaussian
Small σ		
Medium σ		
Large σ		

2. Pixel classifiers

B. Random trees

Features vector:

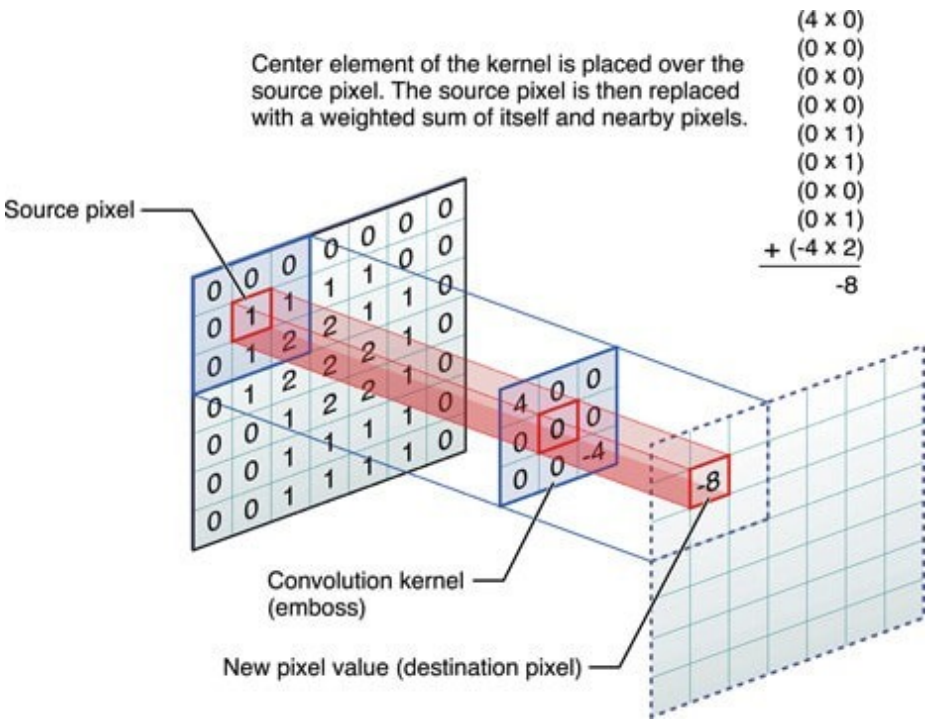
- Principle:
  - Take the input image.
  - Apply different filters at different radii.
  - For **each pixel**, its **features vector** is the collection of the values:
    - For each channel
    - For each feature
    - For each radius
  - **C** channels, **F** features and **R** radii ⇨ Vector of C×F×R values per pixel.

2. Pixel classifiers

B. Random trees

Features vector:

- How are filters processed? — **Convolution filters**
- Used for:
  - Gaussian
  - Laplacian of Gaussian
  - Weighted deviation
  - ...

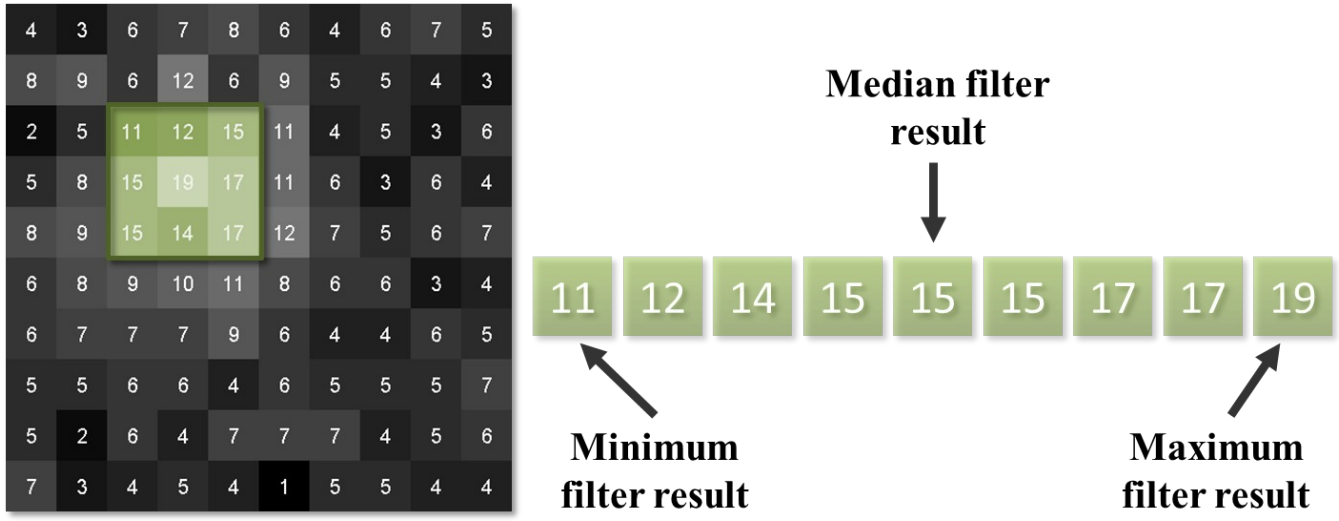


2. Pixel classifiers

B. Random trees

Features vector:

- How are filters processed? — **Non-linear filters**



2. Pixel classifiers

B. Random trees

Features vector:

- What feature(s) to choose?

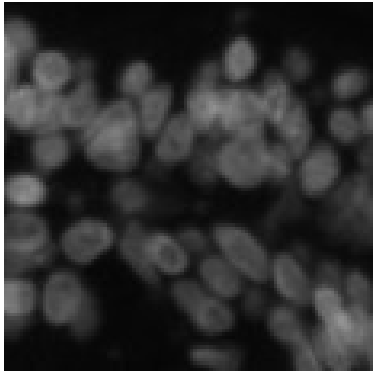
2. Pixel classifiers

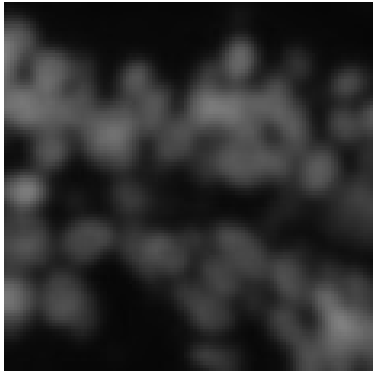
B. Random trees

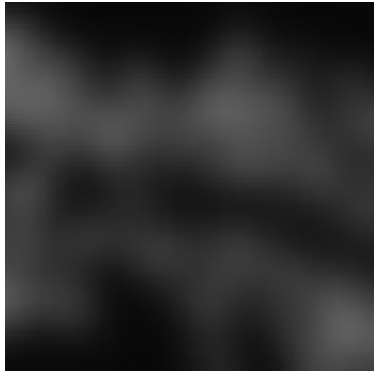
Features vector:

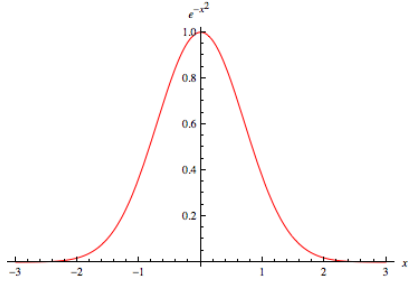
- What feature(s) to choose? — **Gaussian**
  - Smooths everything.
  - Gathers neighboring information into the current pixel.

⇨ Very generalist.









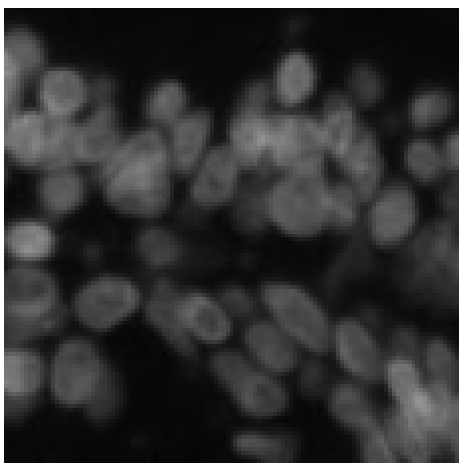
2. Pixel classifiers

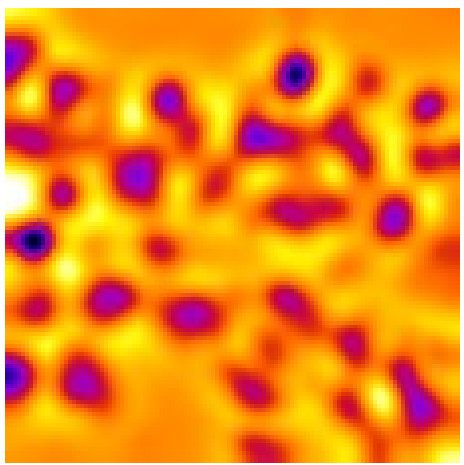
B. Random trees

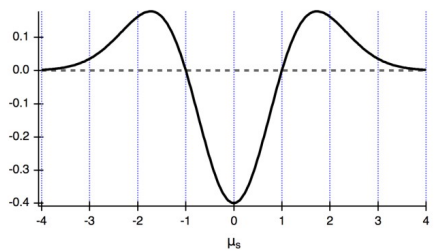
Features vector:

- What feature(s) to choose? — **Laplacian of Gaussian**
  - **Band-pass** filter → given **σ**:
    - Smooths high frequencies
    - Ignores lower frequencies
    - Highlights target frequencies

Nuclei (σ=4):







2. Pixel classifiers

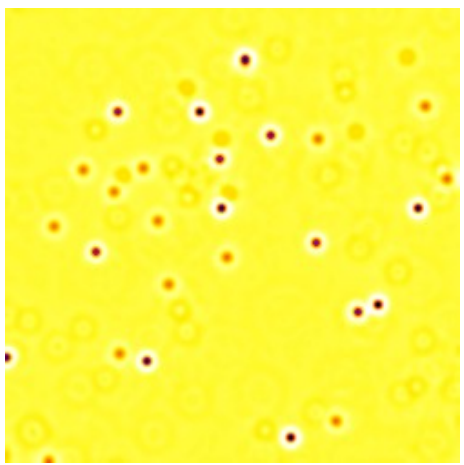
B. Random trees

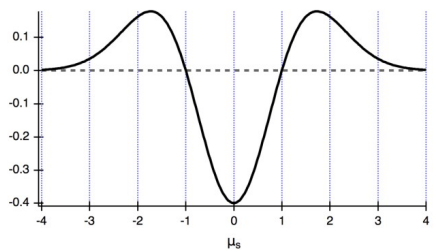
Features vector:

- What feature(s) to choose? — **Laplacian of Gaussian**
  - **Band-pass** filter → given **σ**:
    - Smooths high frequencies
    - Ignores lower frequencies
    - Highlights target frequencies

⇨ Blobby things, some edges

Spots (σ=2):





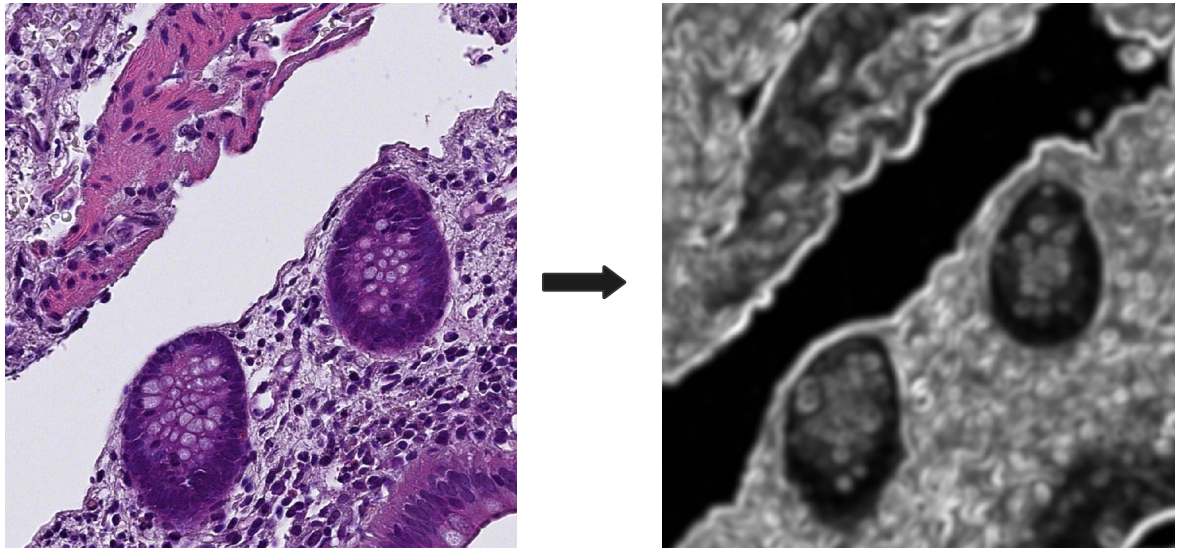
2. Pixel classifiers

B. Random trees

Features vector:

- What feature(s) to choose? — **Weighted deviation**
  - Starts from the result of the **Gaussian filter**.
  - Takes the **values in the kernel** and processes the **standard deviation**.

⇨ Textured vs. smooth objects



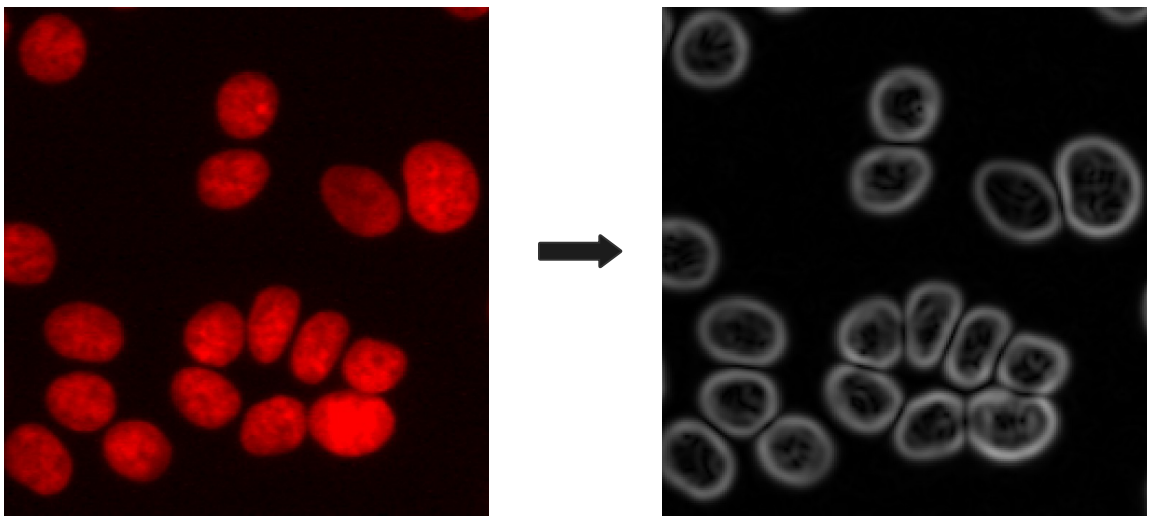
2. Pixel classifiers

B. Random trees

Features vector:

- What feature(s) to choose? — **Gradient magnitude**
  - Processes ∇_x et ∇_y, and assembles them as a vector $V(\nabla_x, \nabla_y)$.
  - The result is the **magnitude of that vector**.

⇨ Edges



2. Pixel classifiers

B. Random trees

Features vector:

- What feature(s) to choose?

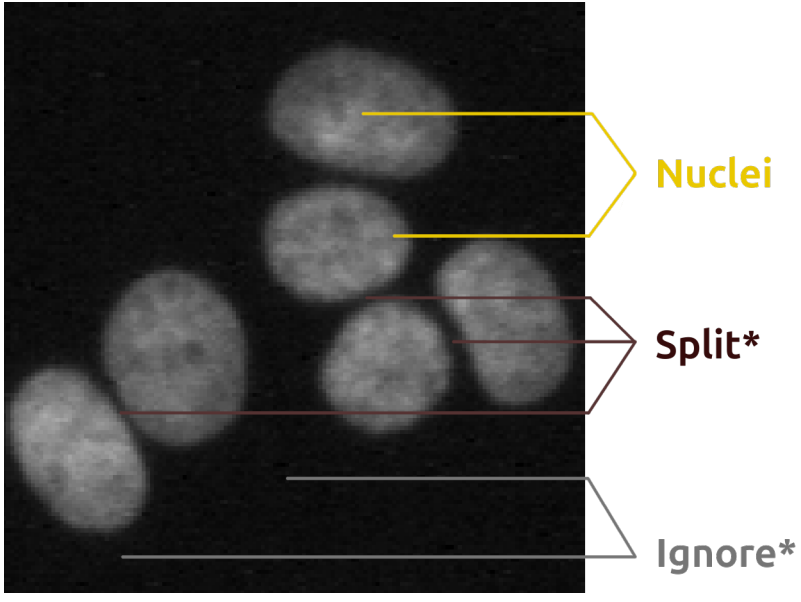
Feature	Purpose
Gaussian filter	General-purpose (color & intensity)
Laplacian of Gaussian	Blobby things, some edges
Weighted deviation	Textured vs. smooth areas
Gradient magnitude	Edges
Structure tensor eigenvalues	Long, stringy things
Structure tensor coherence	'Oriented' regions (e.g. aligned cells, fibers)
Hessian determinant	Blobby things (more specific than Laplacian)
Hessian eigenvalues	Long, stringy things

2. Pixel classifiers

B. Random trees

Training phase:

- Use N point clouds (no polygons: neighboring pixels carry similar information).
- One point cloud == one texture == one class.
- Ex: For nuclei
  - Class 1: Ignore*
  - Class 2: Nucleus
  - Class 3: Split*

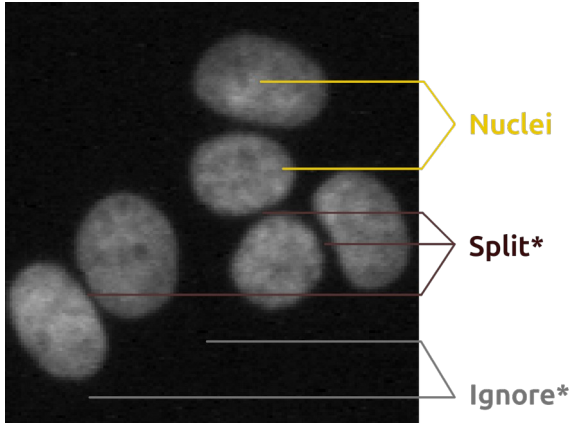


2. Pixel classifiers

B. Random trees

Training phase:

- Use N point clouds (no polygons: neighboring pixels carry similar information).
- One point cloud == one texture == one class.
- Ex: For nuclei
  - Class 1: Ignore*
  - Class 2: Nucleus
  - Class 3: Split*



- Build a **set of examples**: “this vector of features corresponds to this class”.

⚠ Keep **point clouds balanced**: bias in inference.

2. Pixel classifiers

B. Random trees

Training phase:

- We want the classifier to resist to:
  - Intensity shifts
  - Staining variation
  - Minor biological variations

→ Resist to **image-to-image variations**.

2. Pixel classifiers

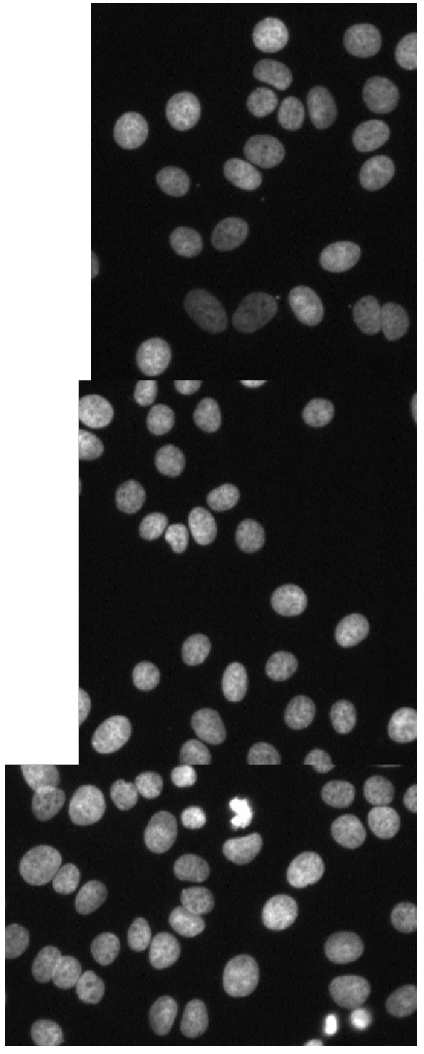
B. Random trees

Training phase:

- We want the classifier to resist to:
  - Intensity shifts
  - Staining variation
  - Minor biological variations

→ Resist to **image-to-image variations**.

- Don’t use a single image: use a **training image**!
  - = **aggregate** of **representative areas** picked from **several images**.

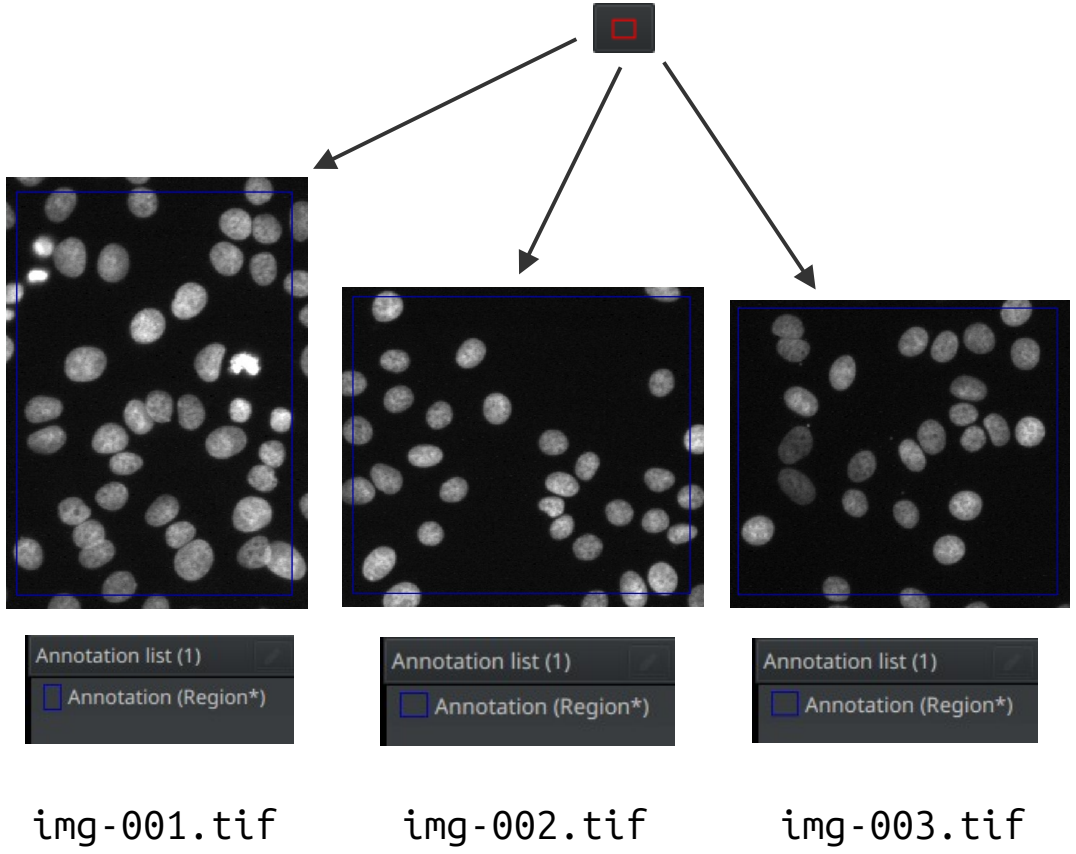


2. Pixel classifiers

B. Random trees

In QuPath: 1. Create a training image

- Find **representative regions** in your images pool.
- Make **rectangle annotations** around them.
- Give them the **Region*** class.

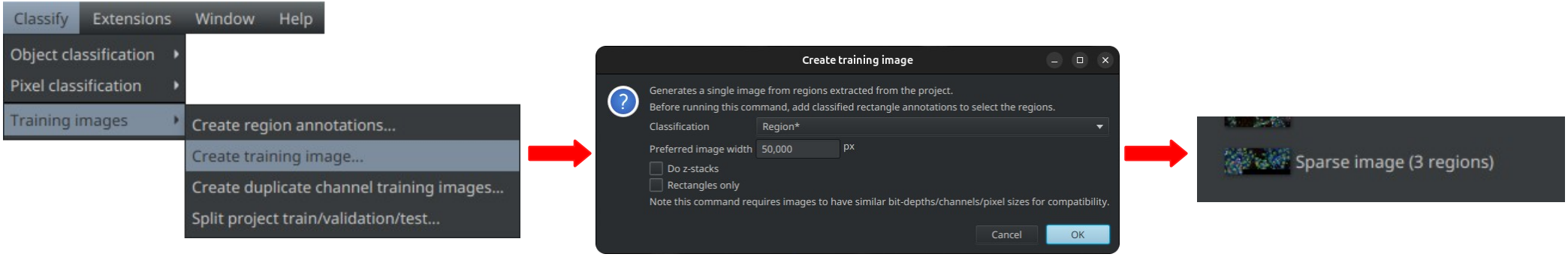


2. Pixel classifiers

B. Random trees

In QuPath: 1. Create a training image

- Find **representative regions** in your images pool.
- Make **rectangle annotations** around them.
- Give them the **Region*** class.
- **Assemble** the regions in one image.

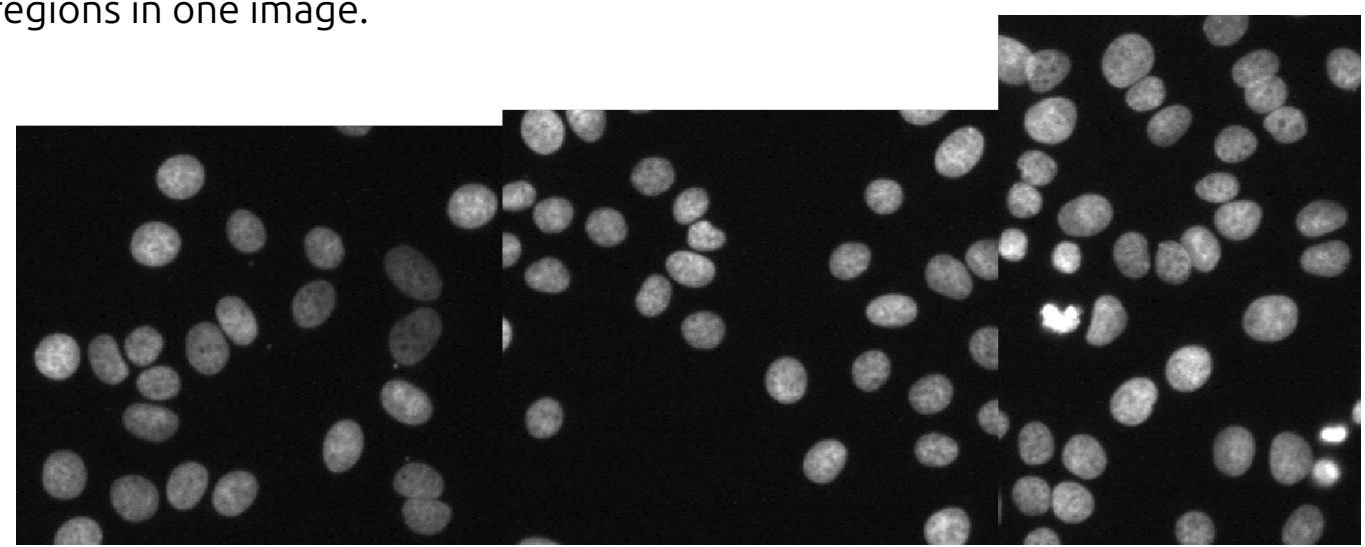


2. Pixel classifiers

B. Random trees

In QuPath: 1. Create a training image

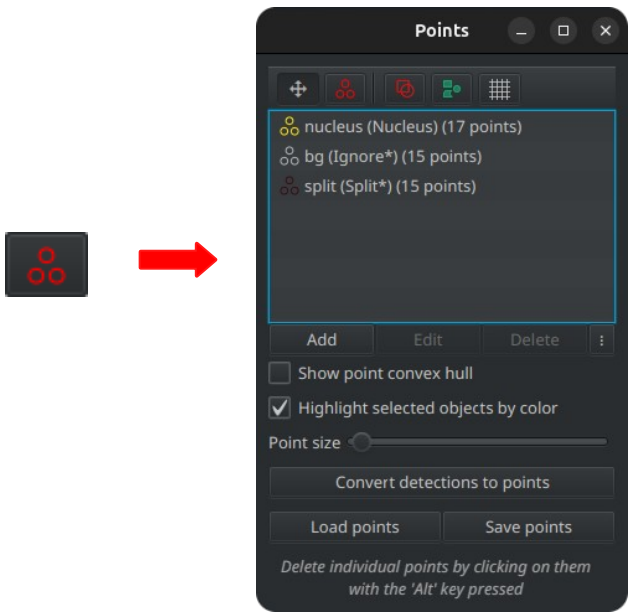
- Find **representative regions** in your images pool.
- Make **rectangle annotations** around them.
- Give them the **Region*** class.
- **Assemble** the regions in one image.



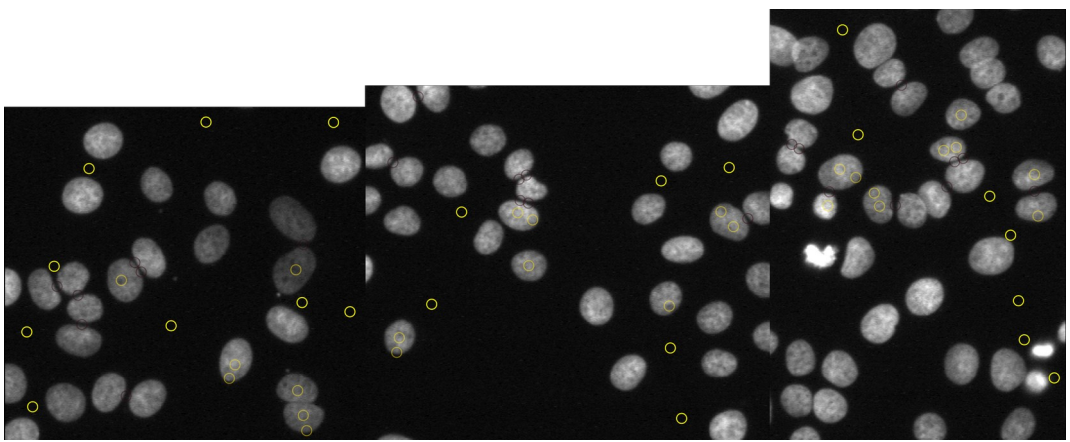
2. Pixel classifiers

B. Random trees

In QuPath: 2. Create your first set of annotations



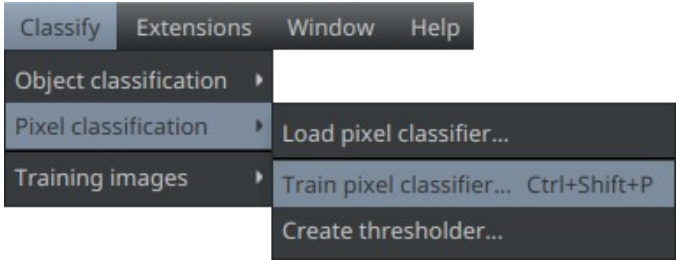
- Create **point clouds**.
- Give them a **class**.
- Add a **similar** number of points per class.
- 10 to 15 points per class is enough to start.
- Class name ends with '*' → ignored class.

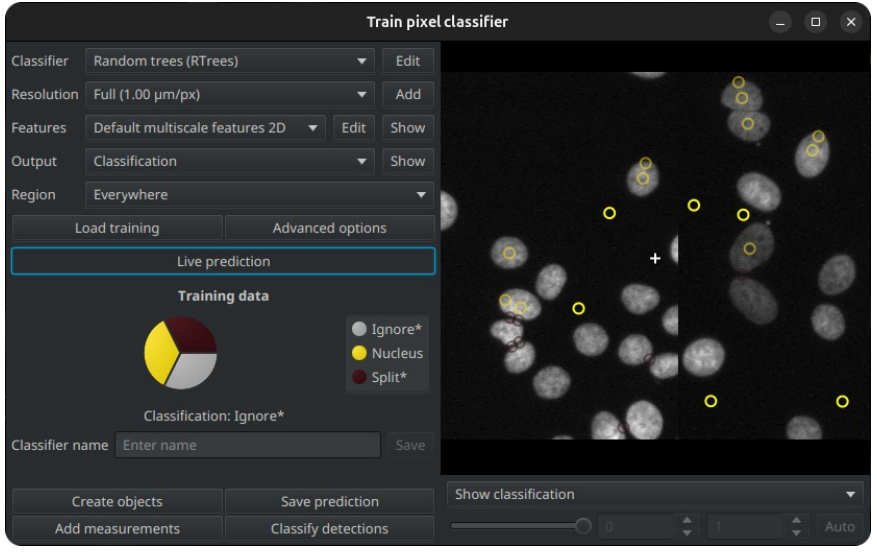


2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool

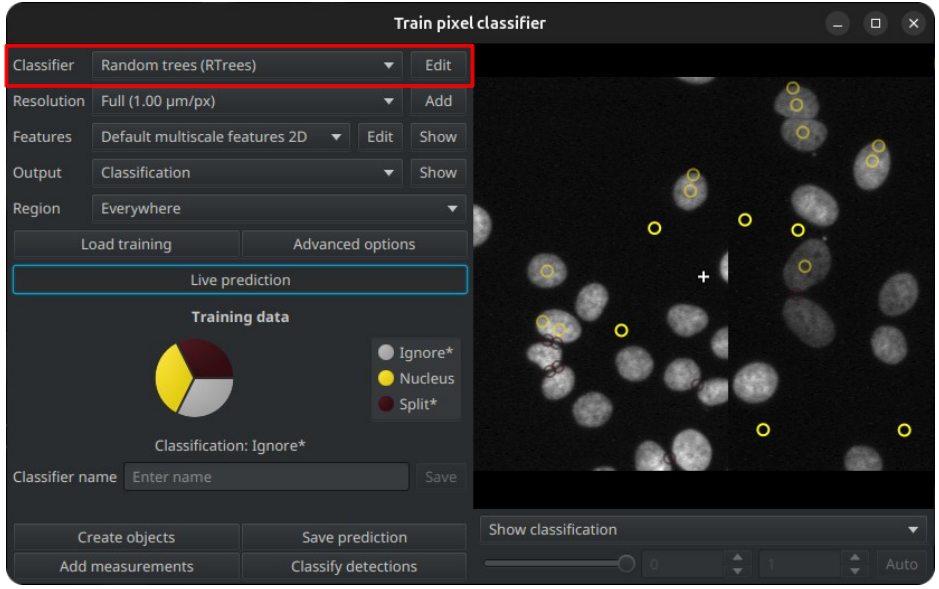




2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



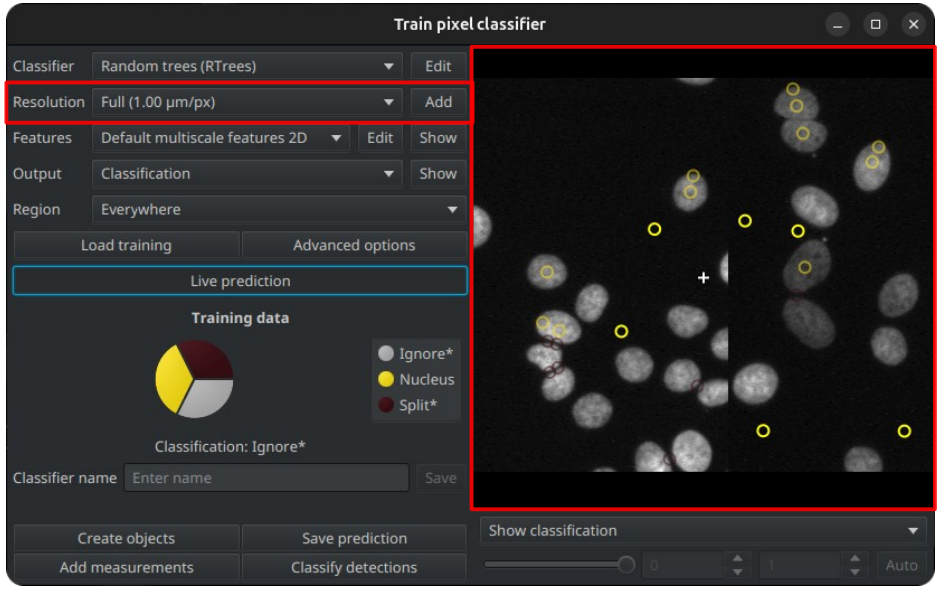
Classifier:

- Classification algorithm to use.
- Use the “Random trees” as explained earlier.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



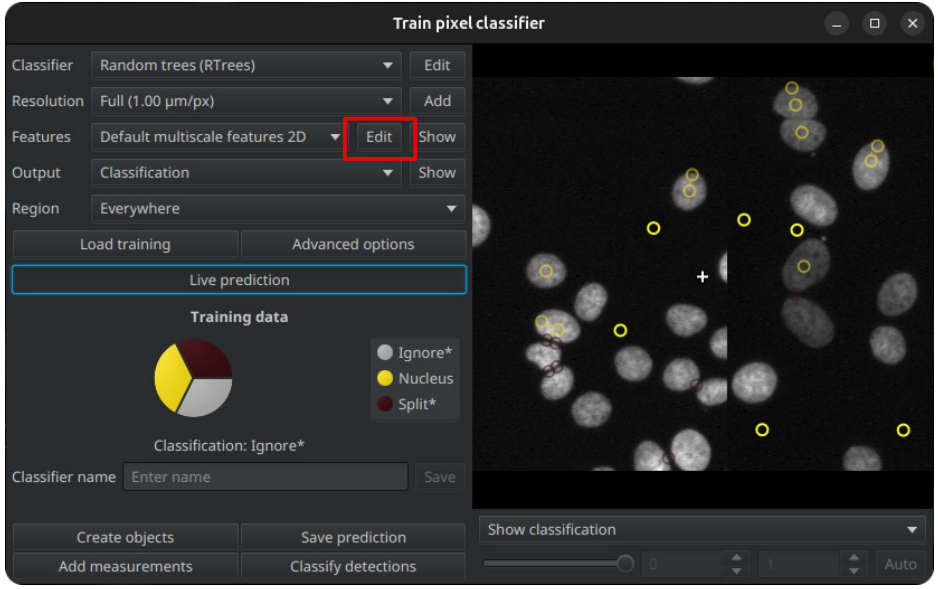
Resolution:

- Same as the resolution for the thresholder.
- Tiny viewer = image at this given resolution.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



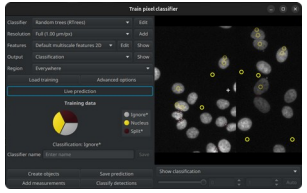
Features > Edit:

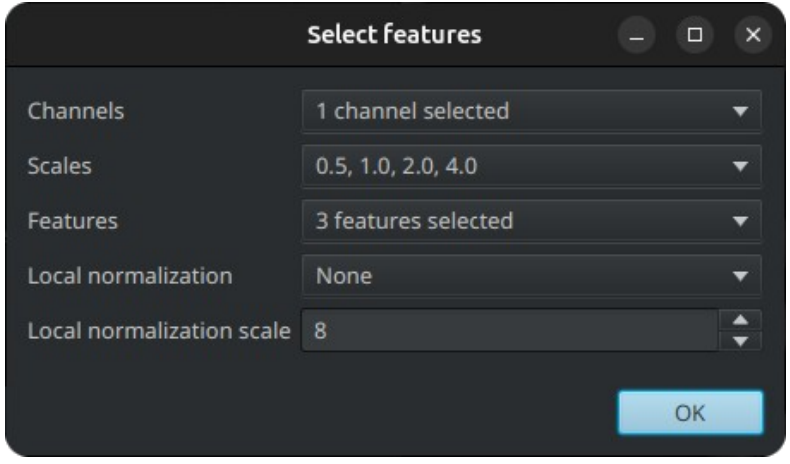
- The dropdown menu is useless (empty).
- “Edit” allows you to choose the **channels**, **features** and **radii** for the random trees.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool





→ Channels to use

→ List of radii for each filter

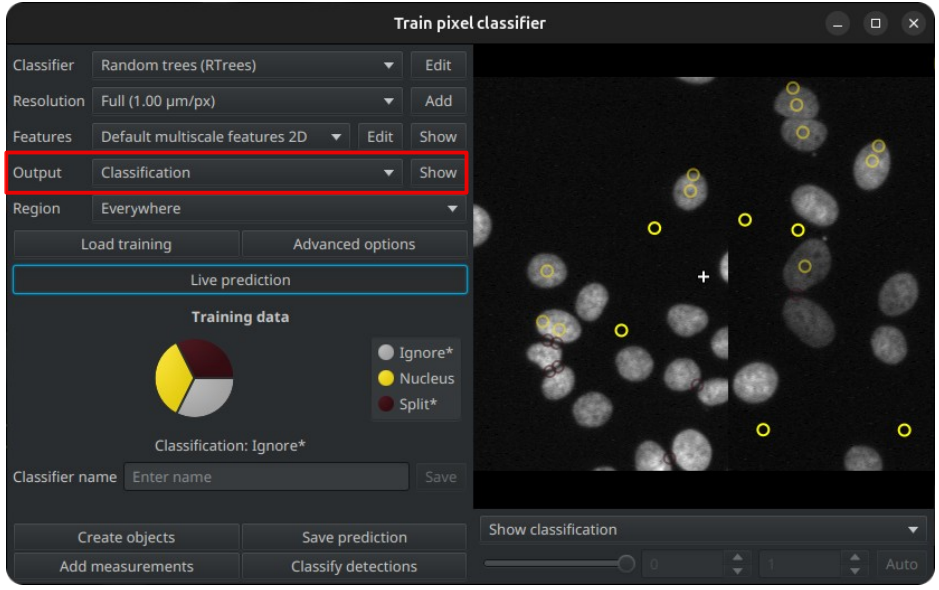
→ List of filters to apply filter

Settings for the normalization per tile.
It is discouraged to use it.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



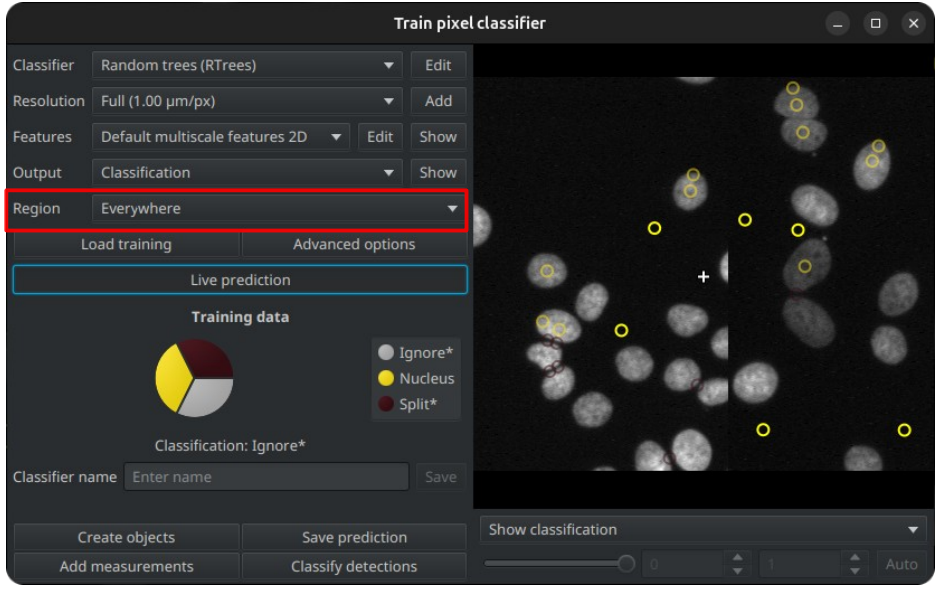
Output:

- Whether you want to see:
  - The predicted class (“Classification”)
  - The probability that each class had (“Probabilities”)

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



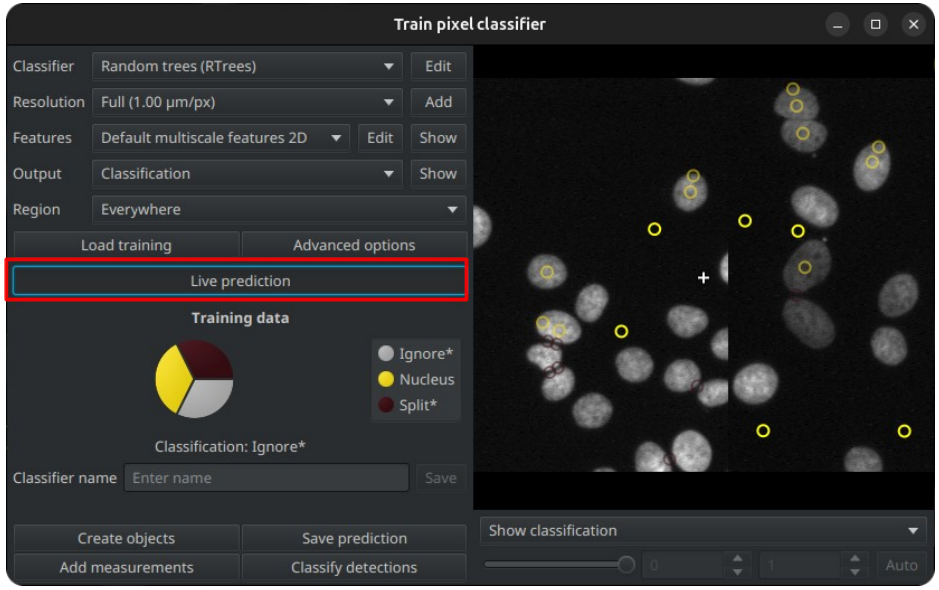
Region:

- Where to apply the classifier for the preview.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



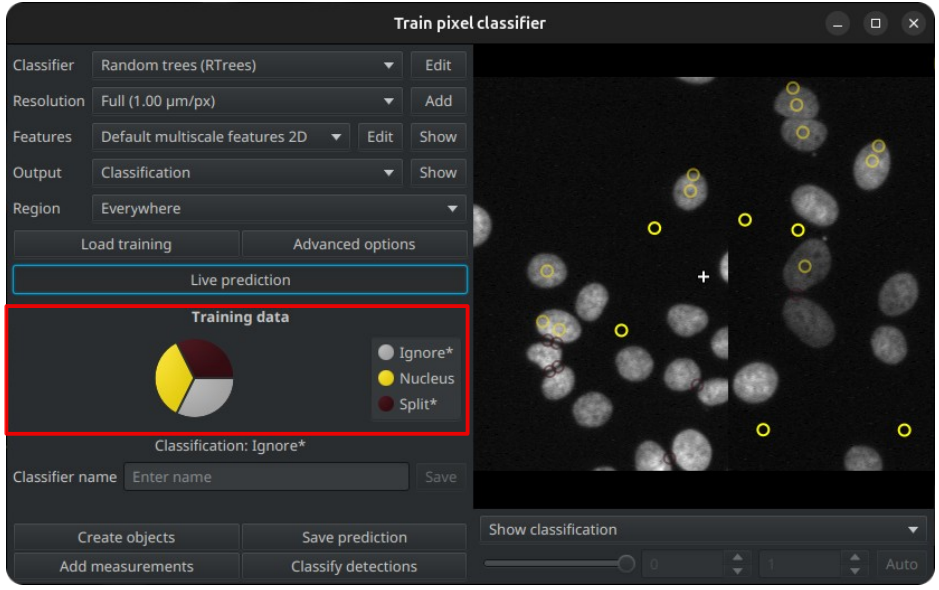
Live prediction:

- Allows you to see in real time the effect of adding or removing points.
- Keep it active as you train your classifier.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



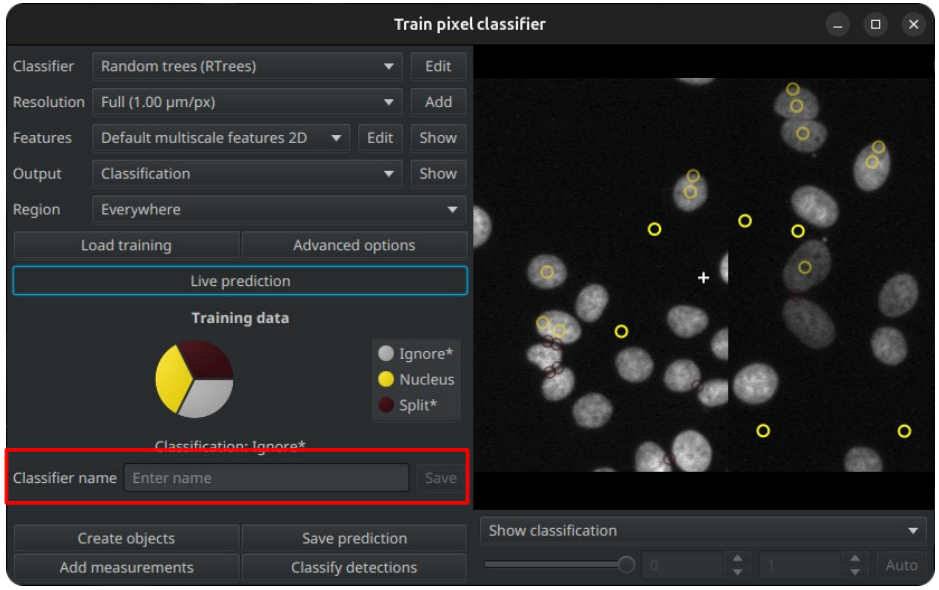
Training data:

- Proportion of each class in the training data.
- Keep the pieces of the pie chart as close to be equal as you possibly can.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



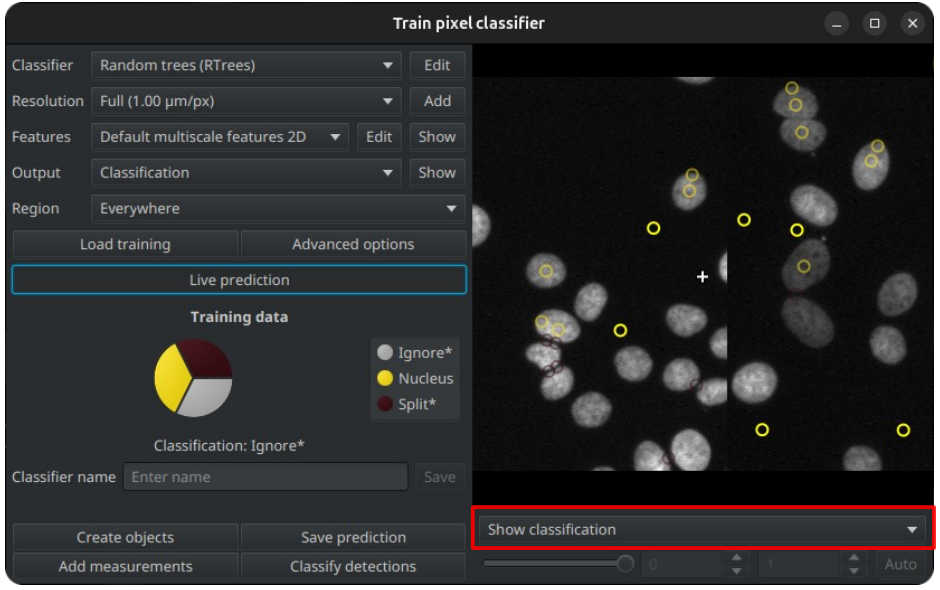
Classifier name:

- How to call this classifier so you can:
  - Reuse it later
  - Call it from a script

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



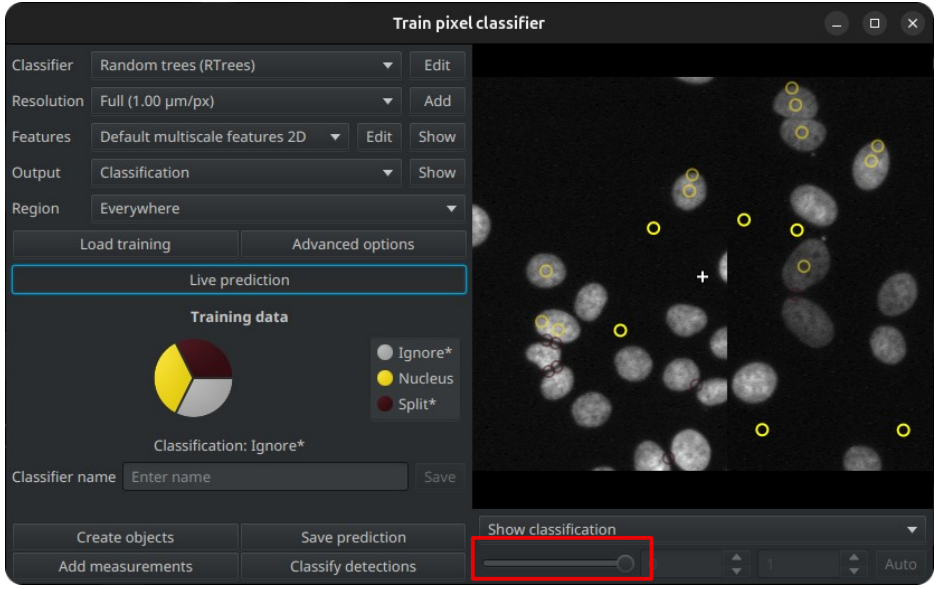
Filters result:

- Doesn’t do anything if the “Live preview” is off.
- Allows you to see the result of each filter at each radius for each channel.

2. Pixel classifiers

B. Random trees

In QuPath: 3. Launch the training tool



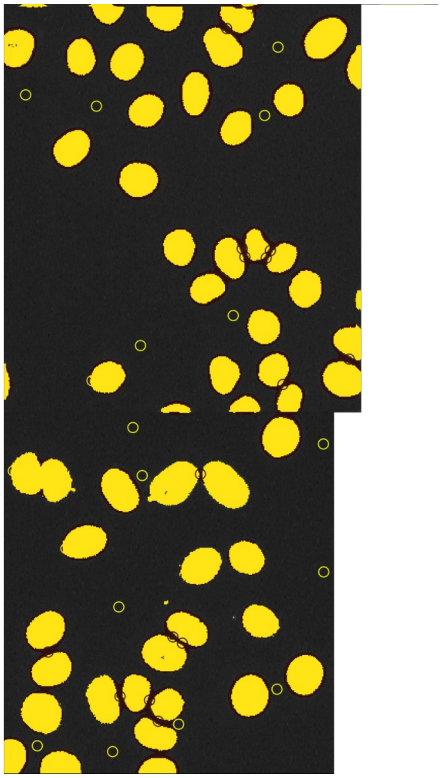
Output opacity:

- Makes the preview transparent.
- Useful to control the image below.

2. Pixel classifiers

B. Random trees

In QuPath: 4. Iteratively train the classifier



Steps:

- Start to training tool with your first set of points.
- Choose the algo.
- Choose the feature by checking that they make sense.
- Check for errors with the “Live preview”.
- Add points where you see errors.
- When the preview is good, name and save the classifier.

→ Exercise 4.3: Using a N-classes pixel classifier

3. Pixel classifiers

C. Semi-automatic (SAM)

Principle:

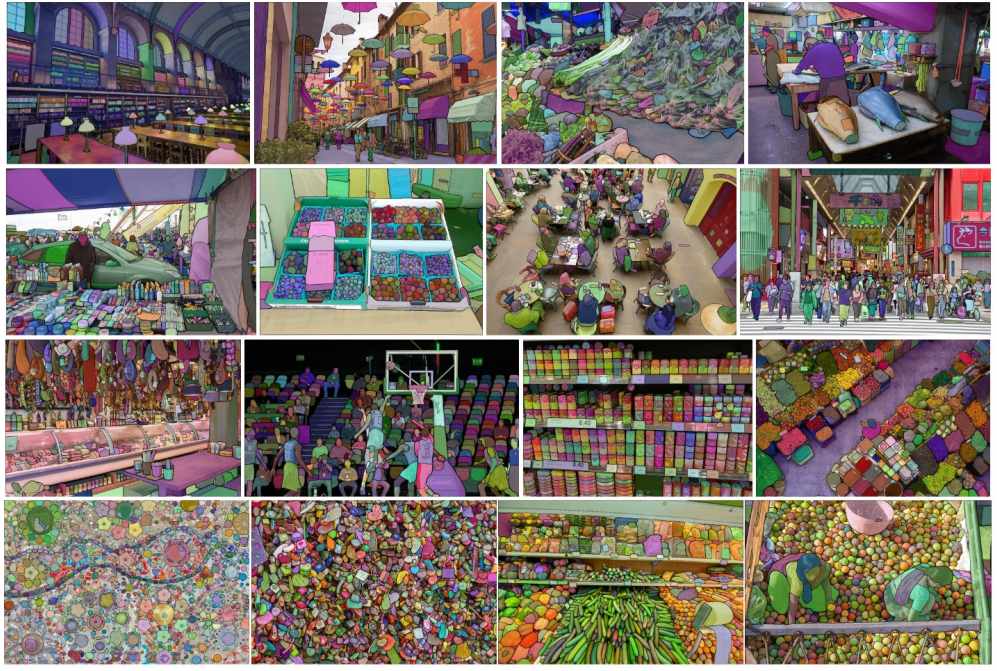
- **Deep-learning** architecture, by Meta (Facebook, Instagram, …).

3. Pixel classifiers

C. Semi-automatic (SAM)

Principle:

- **Deep-learning** architecture, by Meta (Facebook, Instagram, …).
- Not trained with **biological images**.

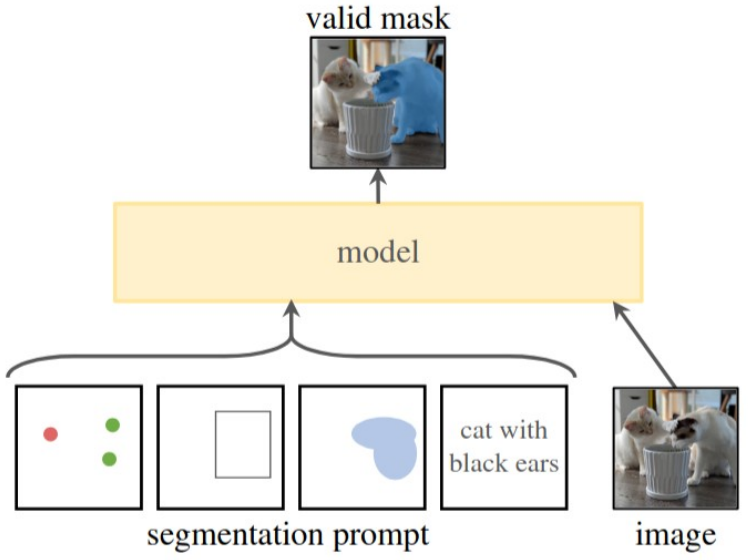


3. Pixel classifiers

C. Semi-automatic (SAM)

Principle:

- **Deep-learning** architecture, by Meta (Facebook, Instagram, …).
- Not trained with **biological images**.
- **Input:** a user prompt
  - Points: Background vs. foreground
  - Rectangles: Bounding-box
  - Masks: Rough mask
  - Text: Description of what you want
  - Image: Find similar objects

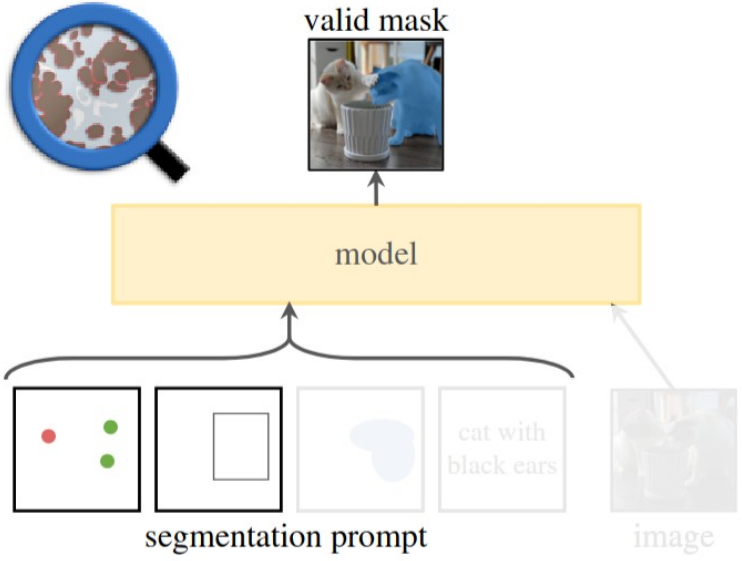


3. Pixel classifiers

C. Semi-automatic (SAM)

Principle:

- **Deep-learning** architecture, by Meta (Facebook, Instagram, …).
- Not trained with **biological images**.
- **Input:** a user prompt
  - Points: Background vs. foreground
  - Rectangles: Bounding-box
  - Masks: Rough mask
  - Text: Description of what you want
  - Image: Find similar objects



3. Pixel classifiers

C. Semi-automatic (SAM)

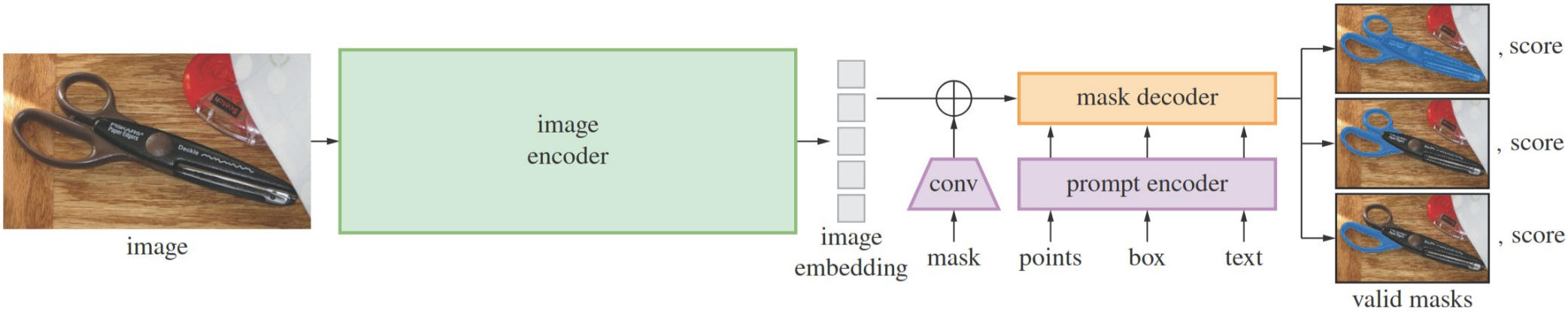
Principle:

- **Deep-learning** architecture, by Meta (Facebook, Instagram, …).
- Not trained with **biological images**.
- **Input:** a user prompt
  - Points: Background vs. foreground
  - Rectangles: Bounding-box
  - Masks: Rough mask
  - Text: Description of what you want
  - Image: Find similar objects
- **Output:** three masks + three scores
  - Tries address ambiguous prompts
  - Score = prediction of IoU

3. Pixel classifiers

C. Semi-automatic (SAM)

Architecture:

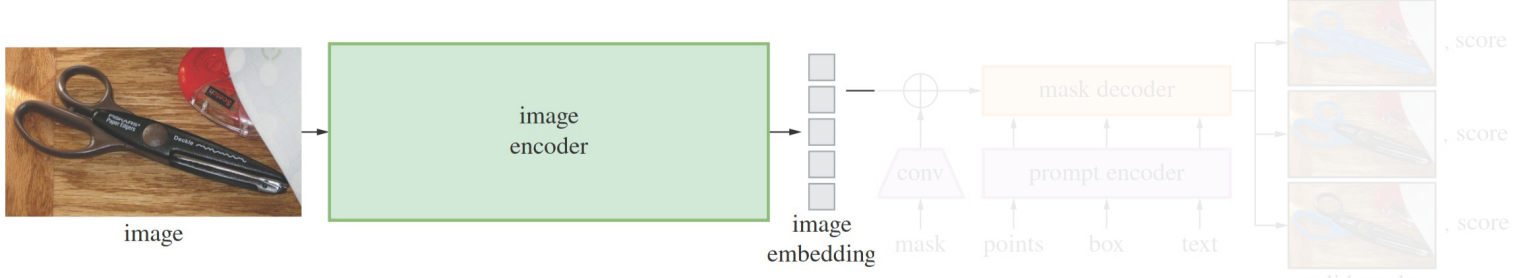


3. Pixel classifiers

C. Semi-automatic (SAM)

Architecture: **1. Image encoding**

- Image fed to **image encoder** (based on a **Masked Auto-Encoder**).
  - MAE == Auto-encoder with **hidden parts** in input images.
- Same encoder:
  - Whatever the image is
  - Whatever the prompt is
- Produces "**image embedding**" == optimized version of **all the possible features** in the image.

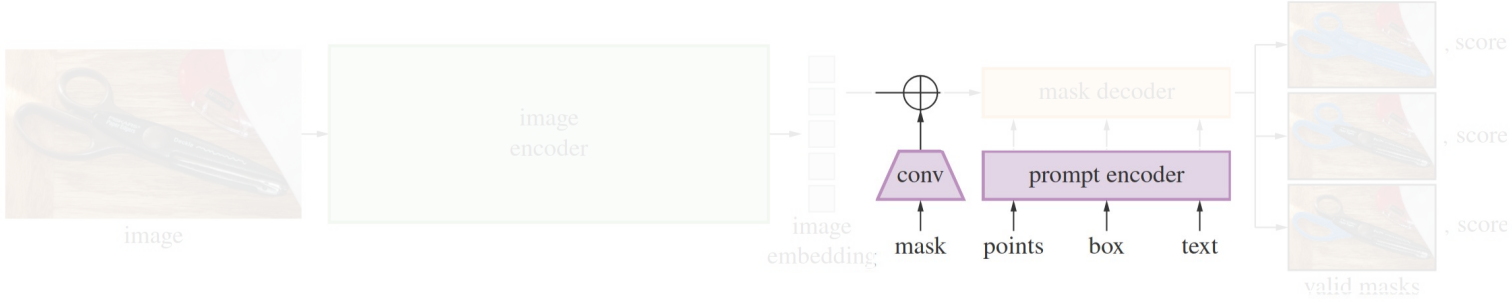


3. Pixel classifiers

C. Semi-automatic (SAM)

Architecture: 2. Prompt encoding

- **Mask/image:** integrated to image embedding using a CNN and summing.
- Sparse prompts:
  - **Points/rectangles:** positional encoding
  - **Text:** CLIP architecture (common text encoder with pre-trained weights)

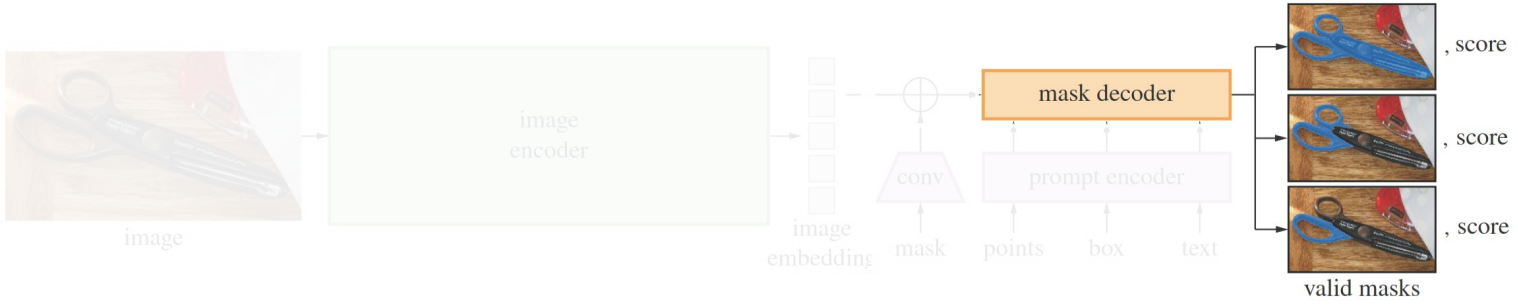


3. Pixel classifiers

C. Semi-automatic (SAM)

Architecture: 3. Mask decoding

- Uses the **prompt** to **interrogate** the features from a region in the **image embedding**.
- Predicts:
  - Three **masks**
  - Three **scores**



3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: **SAM API as a server**

- **SAM** is implemented in **Python**.
- **QuPath** is implemented in **Java**.

Out of the box → don't communicate

⇒ How to use SAM from QuPath?

3. Pixel classifiers

C. Semi-automatic (SAM)

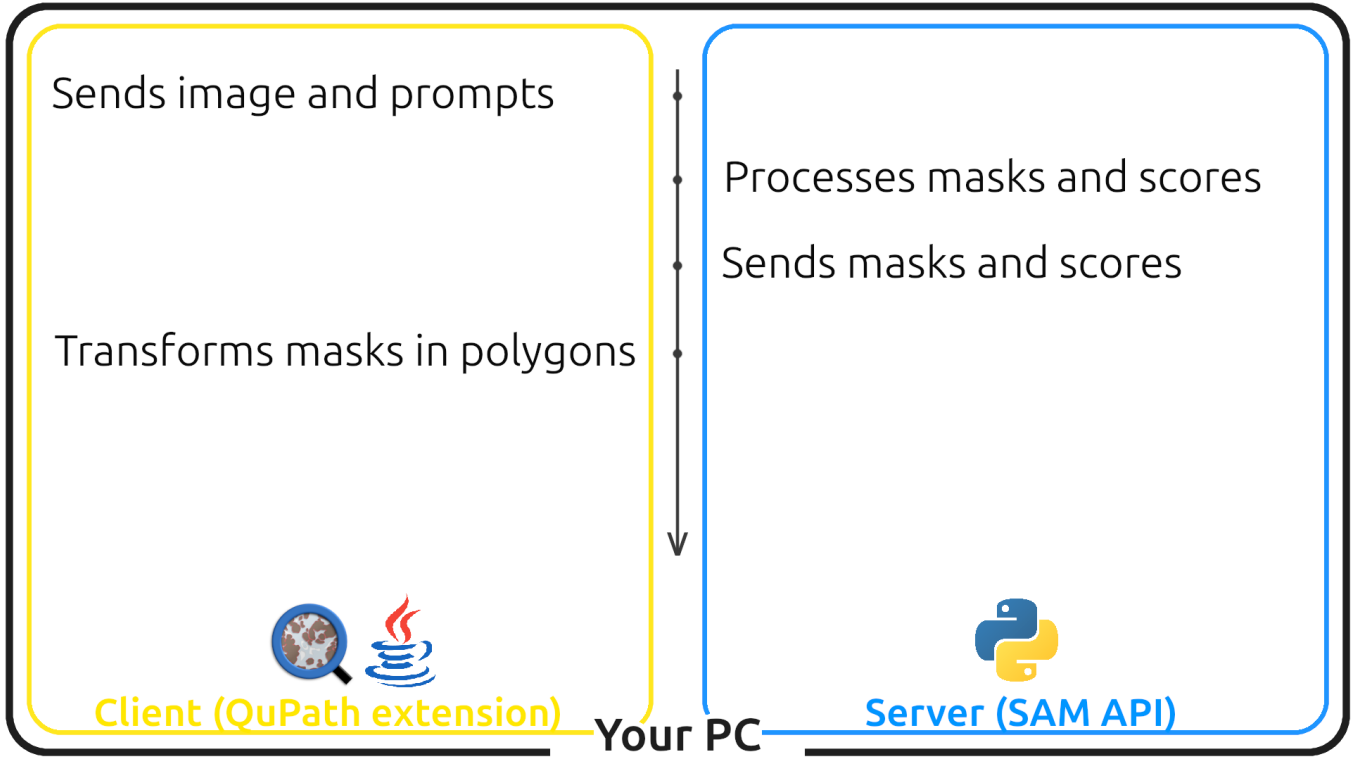
In QuPath: SAM API as a server

- **SAM API** → server → local (on your computer)
- **SAM extension** → client → local (on your computer)
- Communicate through HTTP requests

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: **SAM API as a server**



3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 1. Launch the server

```
conda activate samapi
uvicorn samapi.main:app --workers 2
```

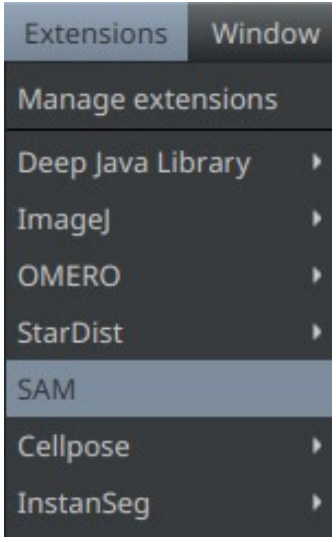
- Server: waits for requests

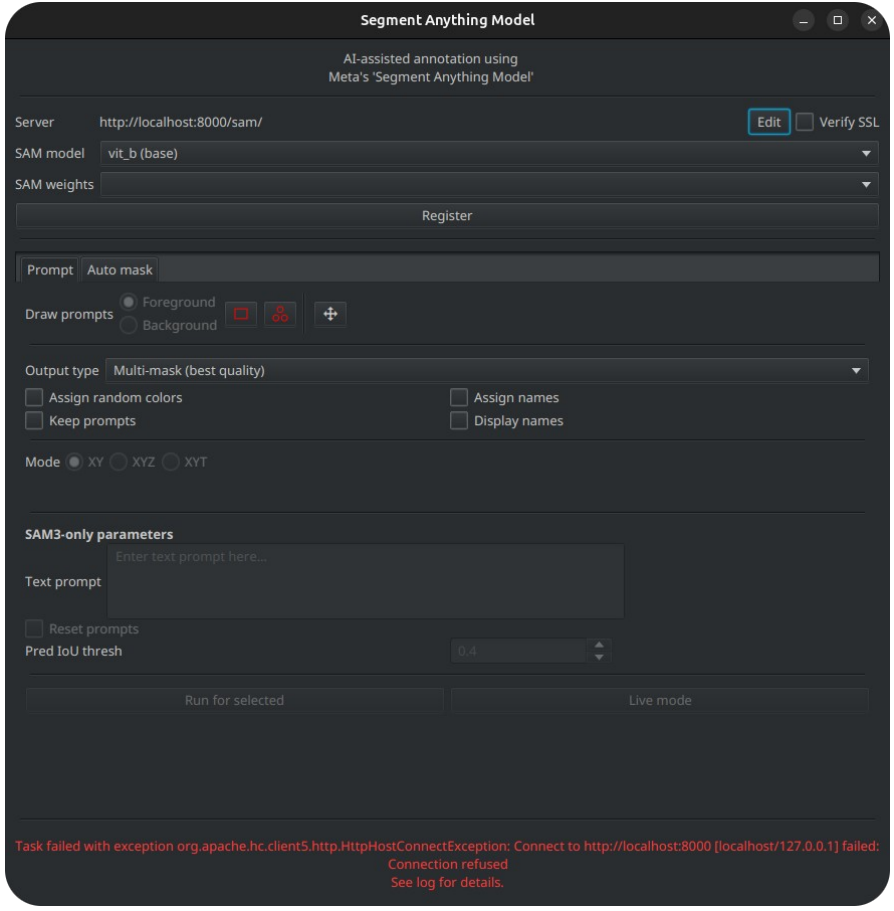
```
(samapi) ➜  ~ uvicorn samapi.main:app --workers 2
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started parent process [956360]
/home/clement/miniconda3/envs/samapi/lib/python3.12/site-packages/samapi/main.py:314: UserWarning: cu
da device found but got the error CUDA out of memory. Tried to allocate 768.00 MiB. GPU 0 has a total
 capacity of 5.67 GiB of which 299.00 MiB is free. Process 956364 has 3.17 GiB memory in use. Includi
ng non-PyTorch memory, this process has 2.15 GiB memory in use. Of the allocated memory 1.96 GiB is a
llocated by PyTorch, and 48.79 MiB is reserved by PyTorch but unallocated. If reserved but unallocate
d memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation
.  See documentation for Memory Management  (https://pytorch.org/docs/stable/notes/cuda.html#environm
ent-variables) - using CPU for inference
  warnings.warn(
INFO:     Registering default weights. This step may take a while for the first time.
INFO:     Registering default weights. This step may take a while for the first time.
INFO:httpx:HTTP Request: HEAD https://huggingface.co/facebook/sam3/resolve/main/sam3.pt "HTTP/1.1 401
 Unauthorized"
WARNING:  Failed to register weights for ModelType.sam3: GatedRepoError: 401 Client Error. (Request I
D: Root=1-69b2be7a-0474be6d30a9bb8d1c2d95ff;bced09ff-3b98-48ad-8db6-d48aea1bd0cf)

Cannot access gated repo for url https://huggingface.co/facebook/sam3/resolve/main/sam3.pt.
Access to model facebook/sam3 is restricted. You must have access to it and be authenticated to acces
s it. Please log in.. This model will not be available.
INFO:     Started server process [956363]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:httpx:HTTP Request: HEAD https://huggingface.co/facebook/sam3/resolve/main/sam3.pt "HTTP/1.1 401
 Unauthorized"
WARNING:  Failed to register weights for ModelType.sam3: GatedRepoError: 401 Client Error. (Request I
D: Root=1-69b2be7b-6fb52135374b63096e52d46a;4cca72e8-23e1-43a5-a563-8b9631b02c2b)

Cannot access gated repo for url https://huggingface.co/facebook/sam3/resolve/main/sam3.pt.
Access to model facebook/sam3 is restricted. You must have access to it and be authenticated to acces
s it. Please log in.. This model will not be available.
INFO:     Started server process [956364]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)

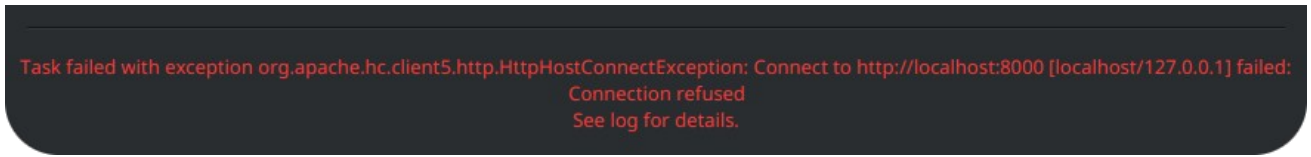




3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)

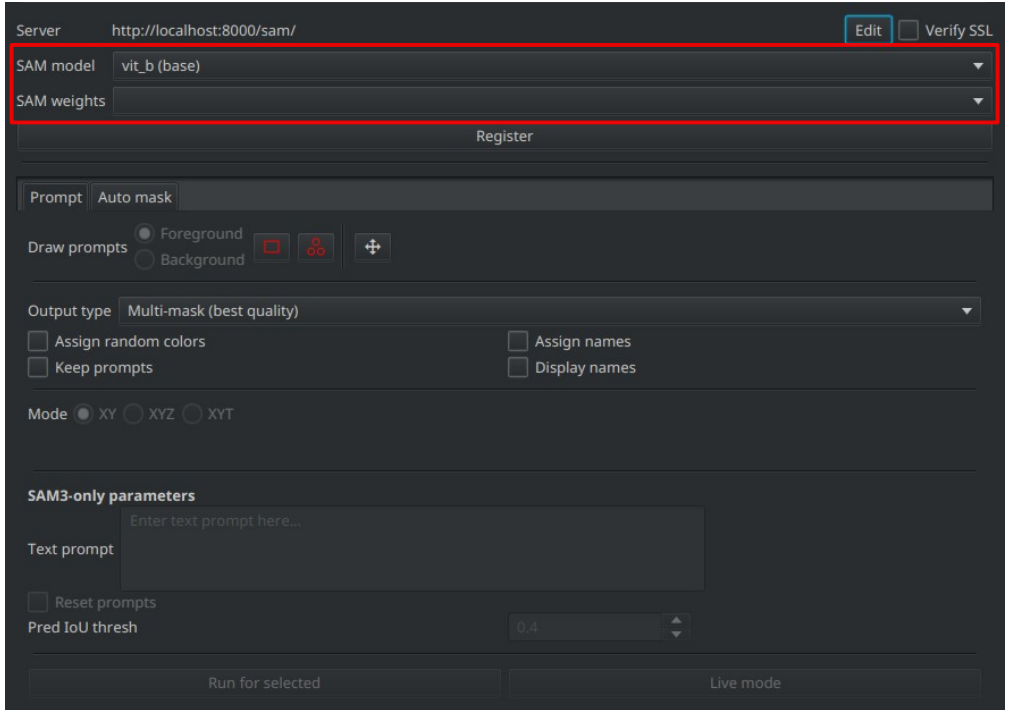


Server not launched!

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



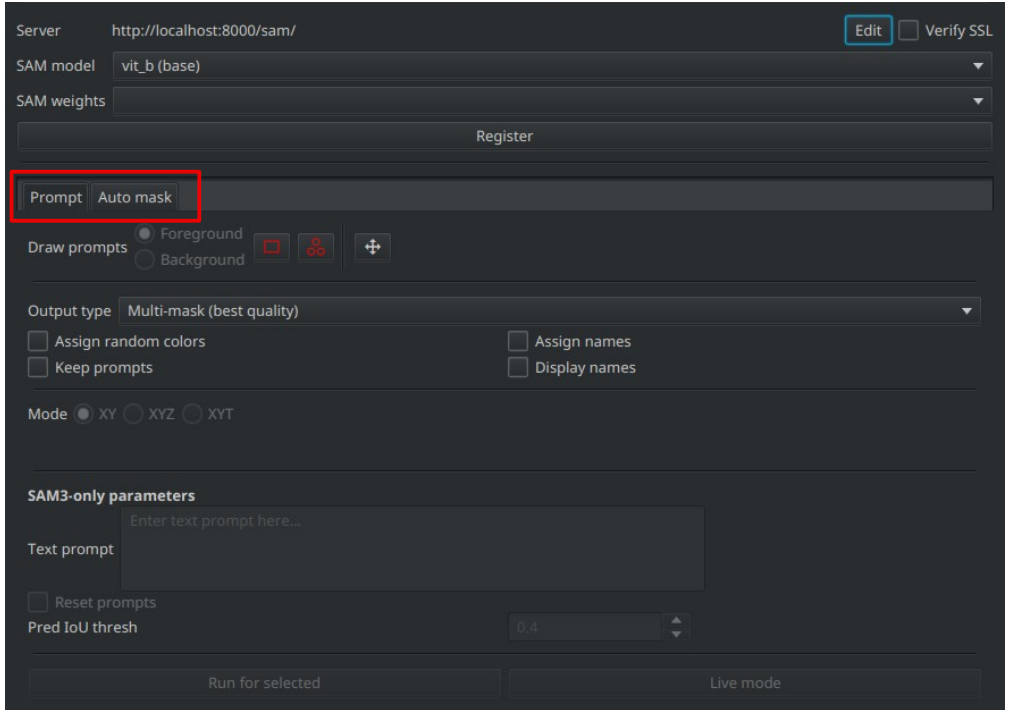
SAM model/weights:

- What model to use?
- Where are the weights of this model stored?
- Tiny > Small > Base > Large > Huge

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



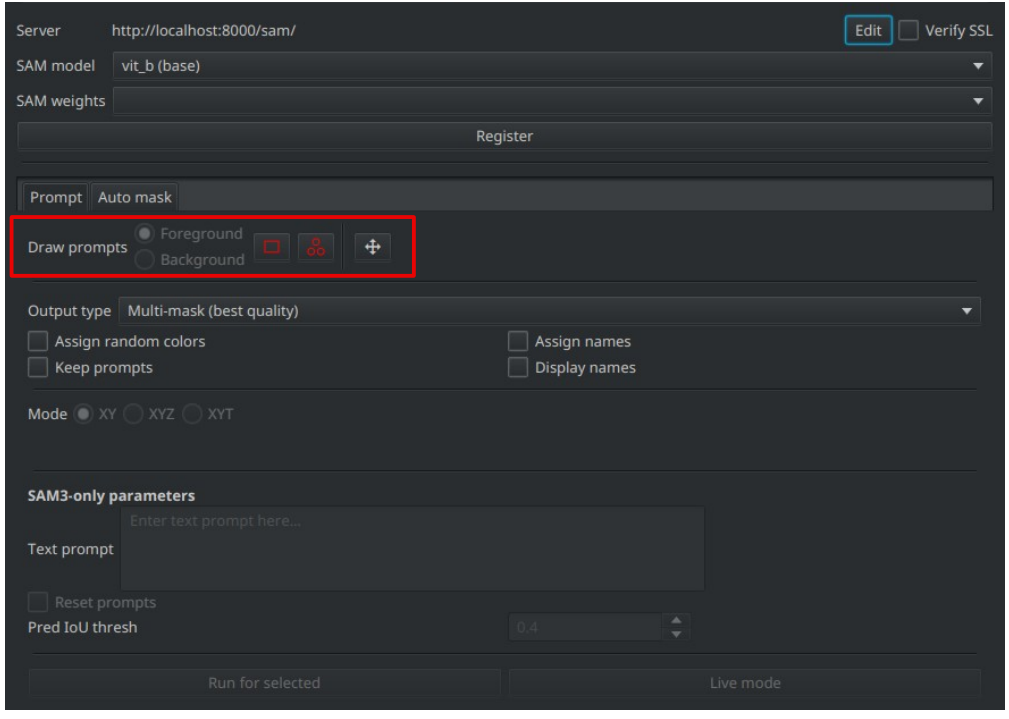
Prompt/auto mask:

- "Prompt": uses your rectangles or points.
- "Auto mask":
  - Places points on evenly spaced grid
  - See what objects come out
  - → Quite unpredictable results

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



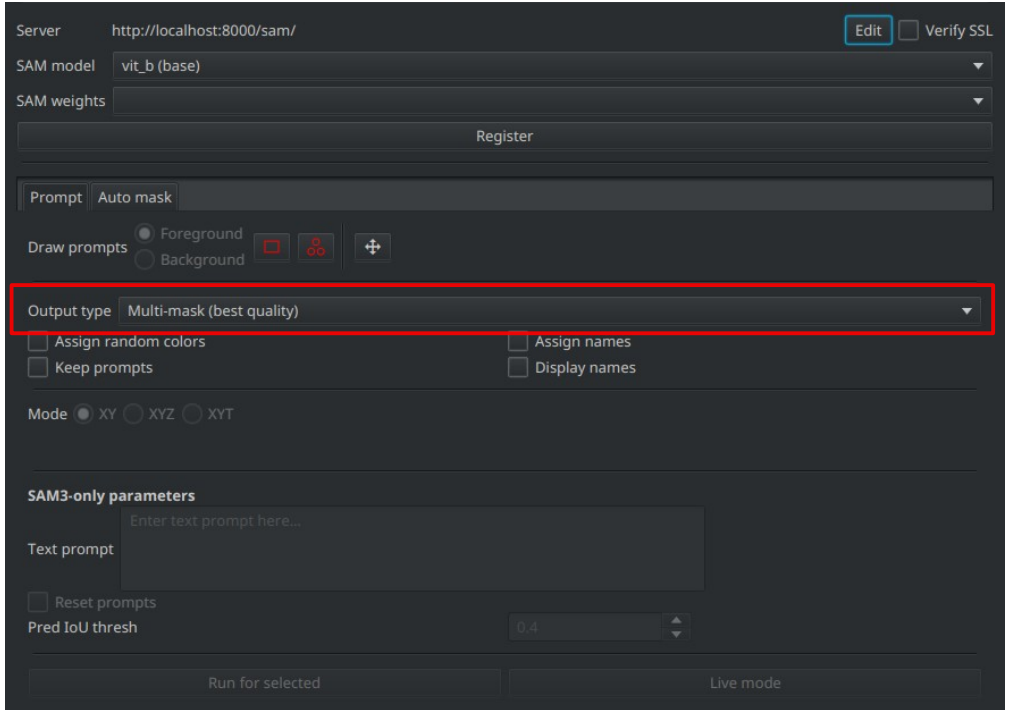
Draw prompts:

- Are you using rectangles or points?
- If points, are they foreground or background?

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



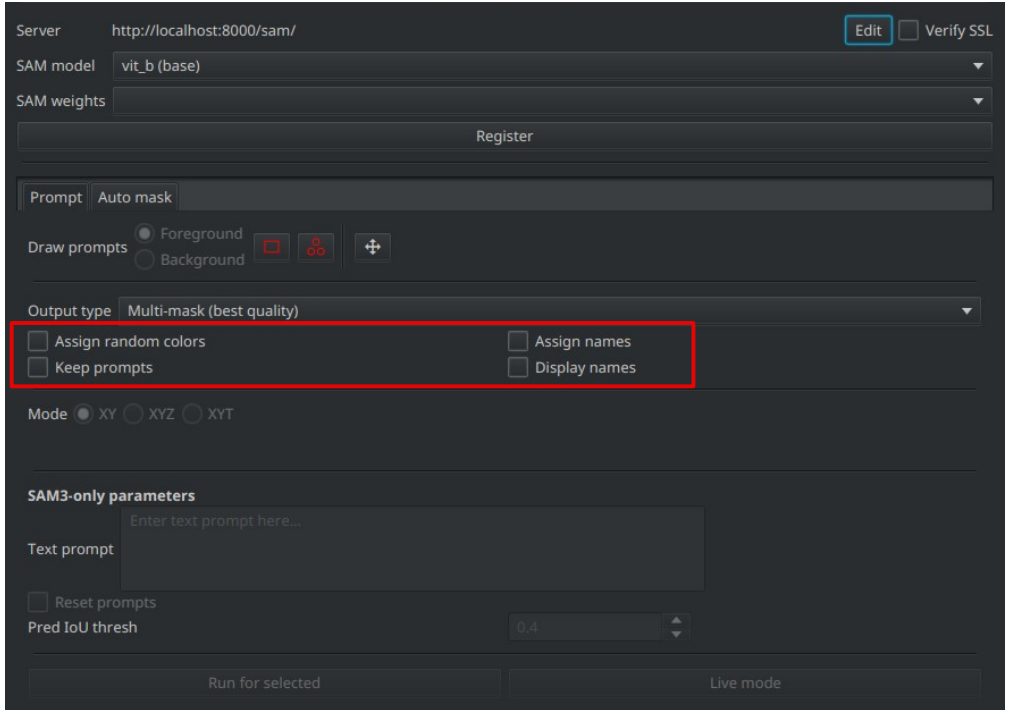
Output type:

- SAM generates 3 masks, which one to keep?
  - "best quality" → highest score
  - "smallest" → smaller area
  - "largest" → largest area
  - "all" → keep all three
  - "single mask" → all three in one annotation

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



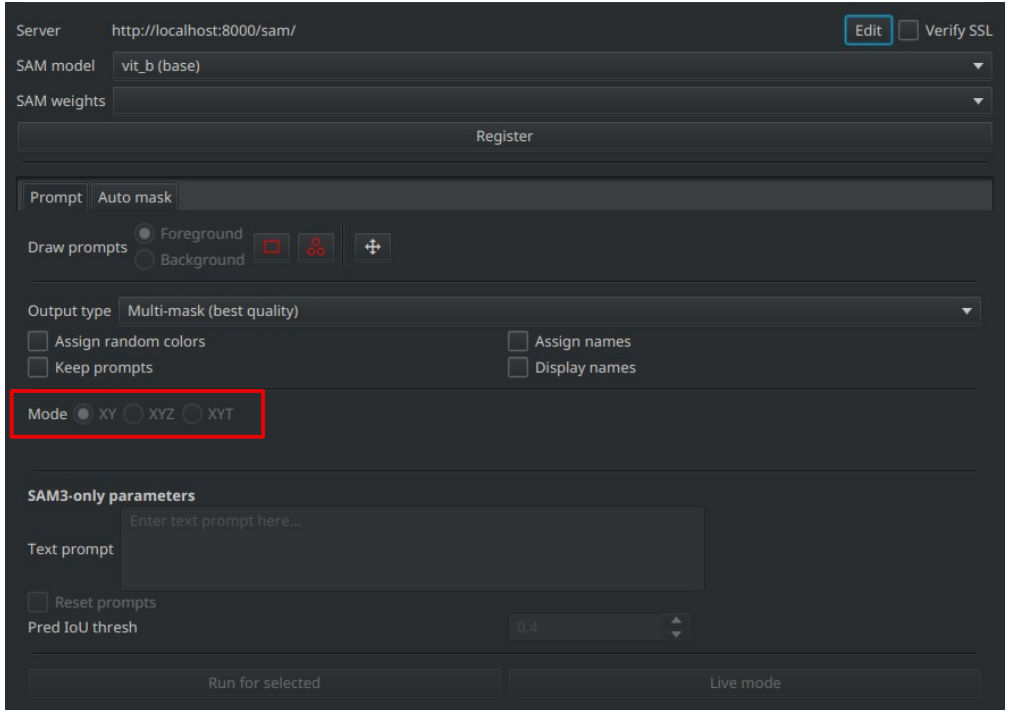
Options:

- "Assign random colors"
  - Gives a random color to each annotation
- "Assign names"
  - Create random names for new annotations
- "Keep prompts"
  - Keep prompts or delete them after prediction
- "Display names"
  - Toggle names display over annotations

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



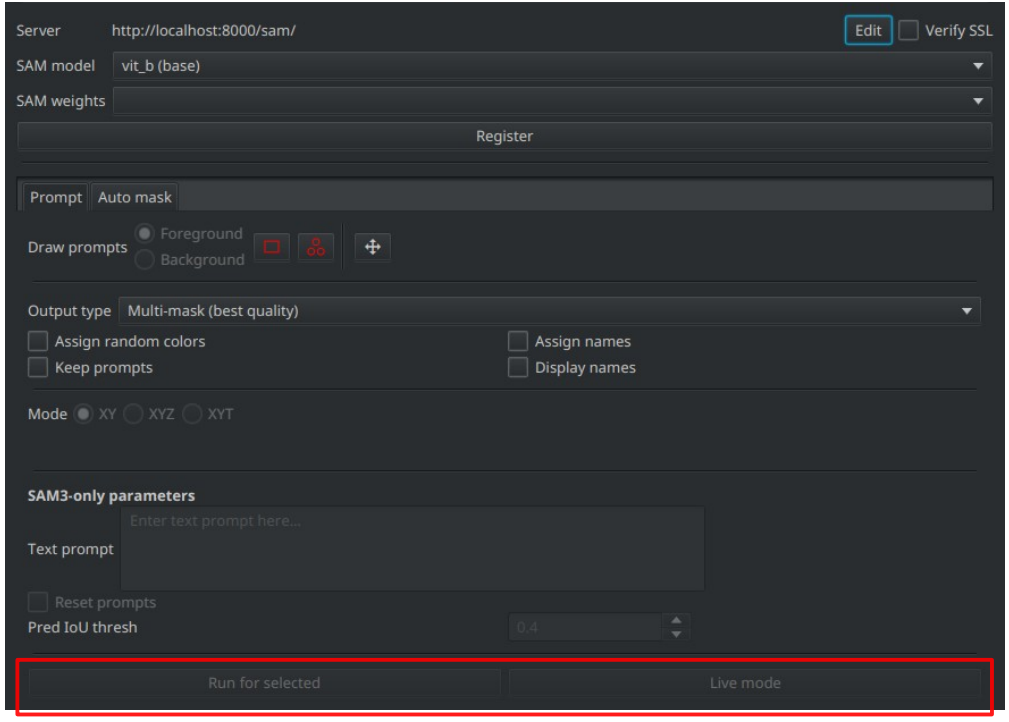
Mode:

- Are you using:
  - A 2D image?
  - A 3D stack?
  - A time series?

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 2. Launch the client (extension)



Mode:

- "Run for selected"
  - Launches prediction for all selected annotations
- "Live mode"
  - Trigger prediction every time a new annotation is created

3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 3. Annotate iteratively

⚠ SAM works only with:
- What is **visible** in your viewer
- At the viewer's **current resolution**

→ The object must fit in the viewer!

3. Pixel classifiers

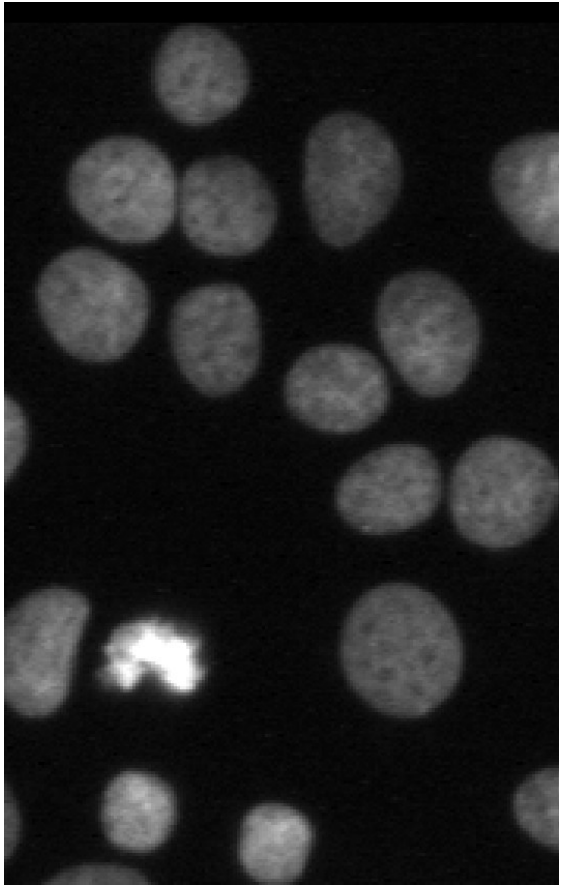
C. Semi-automatic (SAM)

In QuPath: 3. Annotate iteratively

⚠ SAM works only with:
- What is **visible** in your viewer
- At the viewer's **current resolution**

→ The object must fit in the viewer!

1) Locate your objects



3. Pixel classifiers

C. Semi-automatic (SAM)

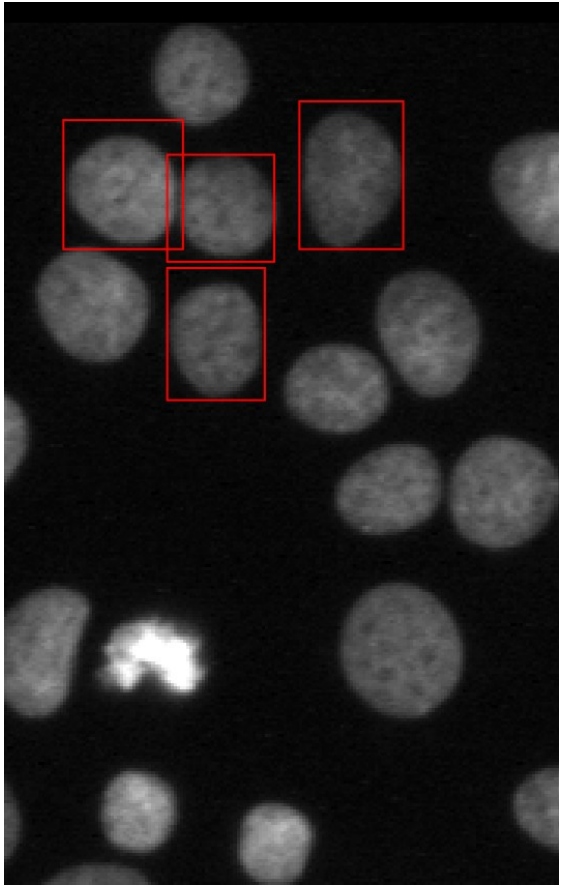
In QuPath: 3. Annotate iteratively

⚠ SAM works only with:
- What is **visible** in your viewer
- At the viewer's **current resolution**

→ The object must fit in the viewer!

1) Locate your objects

2) Make one or more annotations



3. Pixel classifiers

C. Semi-automatic (SAM)

In QuPath: 3. Annotate iteratively

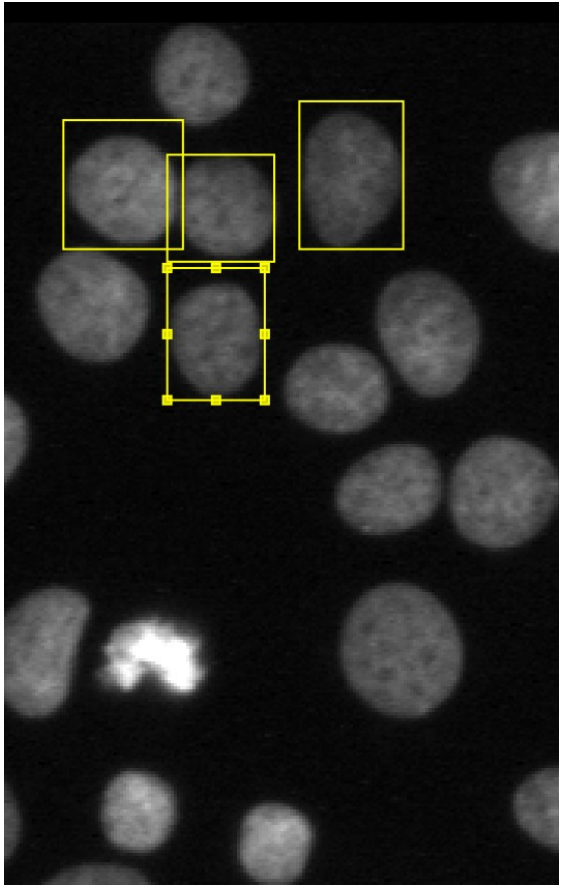
⚠ SAM works only with:
- What is **visible** in your viewer
- At the viewer's **current resolution**

→ The object must fit in the viewer!

1) Locate your objects

2) Make one or more annotations

3) Give them a class and keep them active



3. Pixel classifiers

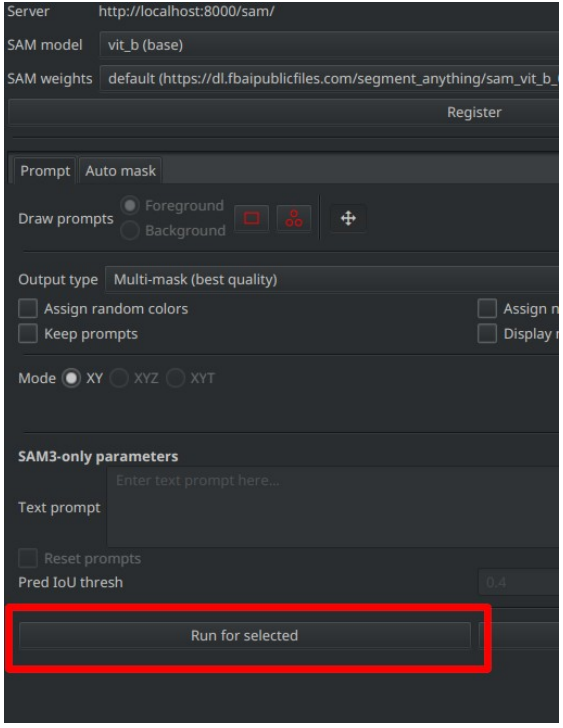
C. Semi-automatic (SAM)

In QuPath: 3. Annotate iteratively

⚠ SAM works only with:
- What is **visible** in your viewer
- At the viewer's **current resolution**

→ The object must fit in the viewer!

1) Locate your objects
2) Make one or more annotations
3) Give them a class and keep them active
4) Click on "Run for selected"



3. Pixel classifiers

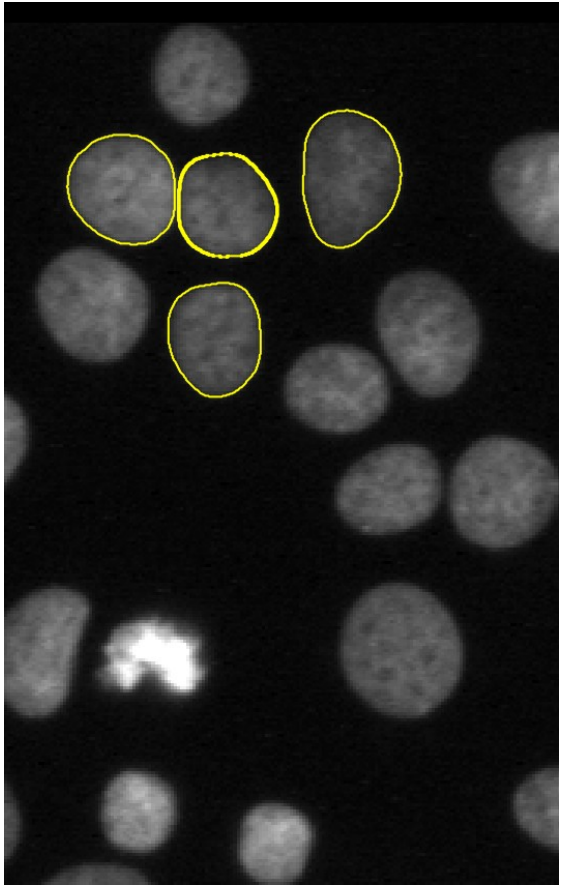
C. Semi-automatic (SAM)

In QuPath: 3. Annotate iteratively

⚠ SAM works only with:
- What is **visible** in your viewer
- At the viewer's **current resolution**

→ The object must fit in the viewer!

1) Locate your objects
2) Make one or more annotations
3) Give them a class and keep them active
4) Click on "Run for selected"
5) Wait for the result



→ Exercise 4.4: Semi-automatic segmentation using SAM