Notebooks and Napari Widgets for Deep Learning

Brian Northan

Notebooks and Napari Widgets for Deep Learning

Brian Northan

Contract R&D Engineer
Imaging Processing

True North Intelligent Algorithms

Contributor: ImageJ-Ops, CLIJ, Image.sc

Outline

- Napari basics
- Train/Validate on one image
  - with Napari ROIs

Outline

- Sparse
  - Stardist with sparse labels
- Scale
  - Cellpose at different scales
  - Stardist receptive field
  - thin protrusions
  - Lady bugs at different scales

Outline

- Segment anything and everything (SAM)
  - visualizing overlapping SAM labels
  - Segment everything and filter

Instance segmentation

Links to projects used in this presentation

Link to slides

Course material from virtual I2k2024

tnia-python - My helper library

Links to projects used in this presentation

easy-augment-batch-dl deep learning on image sets with UNETs, Cellpose, Stardist, SAM and friends.

segment-everything - Utilities for rendering and training prompt-based and overlapping segmentation models (like SAM).

napari-segment-everything - This is a Napari GUI for segment-everything

Under Construction

Training, Validation and Test sets

- Training - data that is input to the model
  - ideally get almost 'perfect' results when predicting on training set
- Validation

  - Data not fed to model
  - used to tune parameters during training.

- Test
  - unseen partition
  - used to evaluate how well model generalize

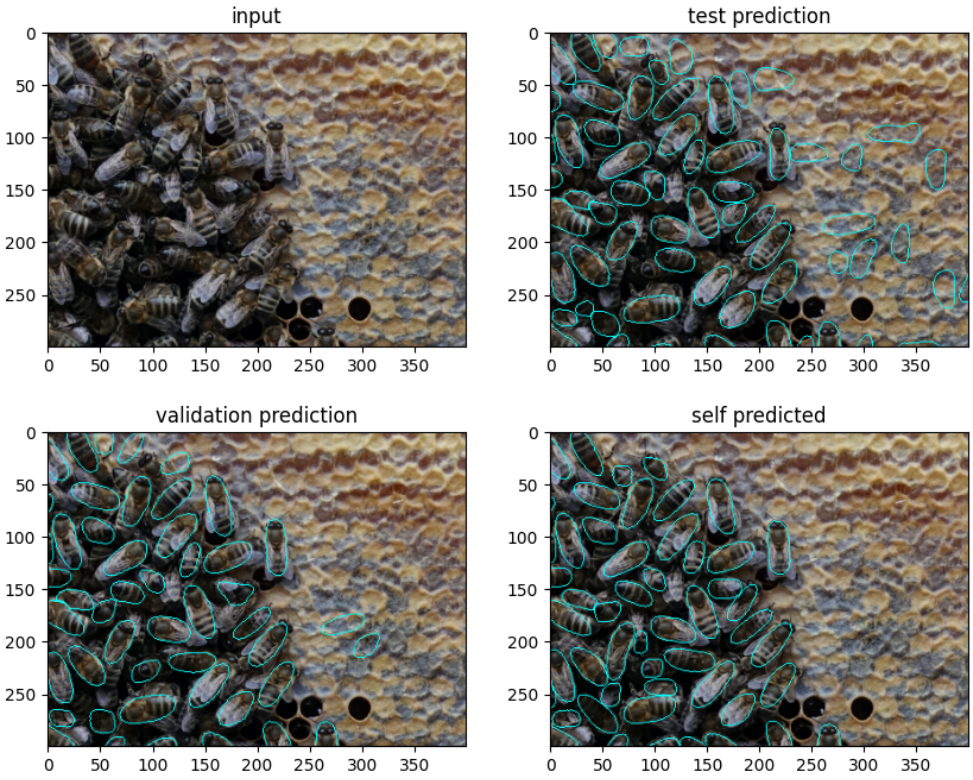


Figure 1: Bee Predictions

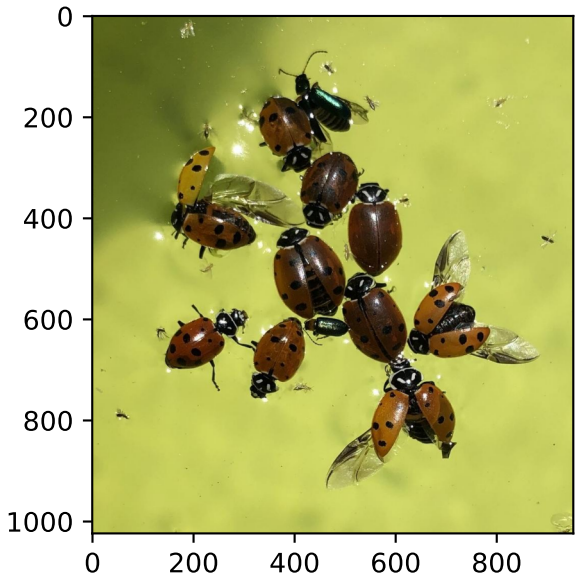
Napari basics

Why use Napari for DL workflows ?

- ROIs
- Layers
- Labelling tools

Test image

```
from skimage.io import imread
import os
import napari
import matplotlib.pyplot as plt
import numpy as np

data_path = r'./data'
parent_path = os.path.join(data_path, 'ladybugs_SAM')
img = imread(os.path.join(parent_path, '26638467_41374651.jpg'))
plt.imshow(img)
```



Start Napari and add layers

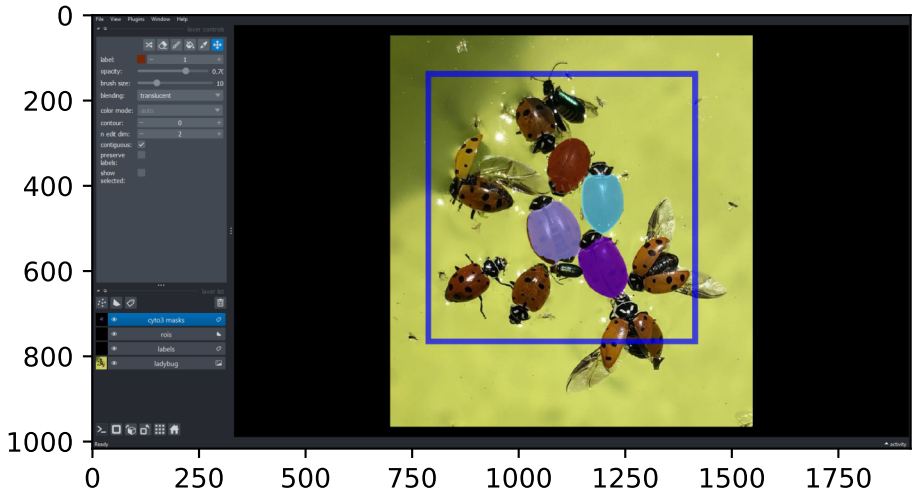
```
viewer = napari.Viewer()

labels = np.zeros([img.shape[0], img.shape[1]], dtype=np.uint16)

viewer.add_image(img, name='ladybug')
viewer.add_labels(labels, name='labels')
rois_layer = viewer.add_shapes(face_color='transparent', edge_width=15, edge_color='blue', na

box = [[100,100], [100,800],[800,800], [800,100]]
rois_layer.add_rectangles(box)
```

Add a cellpose layer

```
from cellpose import models, io
model_cyto3 = models.CellposeModel(gpu=True, model_type="cyto3")
results = np.zeros([img.shape[0], img.shape[1]], dtype=np.uint16)
viewer.add_labels(results, name='cyto3 masks')
results = model_cyto3.eval(img, diameter=140)
viewer.layers['cyto3 masks'].data = results[0]
```

```
WARNING: more than 2 channels given, use 'channels' input for specifying channels - just usi
```

```
screenshot = viewer.screenshot(r'./napari_screenshot.png', canvas_only=False)
plt.imshow(screenshot)
```



Train and test model on a single image

Why?

- Proof of concept
- Grants, SBIRs
- Data sharing restrictions

How?

- Can't just train/predict on same image
- Use ROIs to divide image into training/validation

Train and test model on a single image

- Explore and label data with napari-easy-augment-batch-dl
  - draw and label ROIs
- Augment and crop to create patches (hundreds)
- Train
- Predict and fix mistakes

Augmentation

- From single label can genrate hundred of augmented patches
- Flip, rotate, resize
- Random adjust brightness/contrast
- Random adjust gamma
- Color jitter
- Elastic transform

Augmentation

- Save explicitly?
  - can be helpful for troubleshooting, performance and repeatability
- Augment 'inline'
  - More variation, save diskspace

Train

- train on augmented patches

Prediction

- after training go back and predict
  - correct mistakes

Sparse Labeling

- See CellSparse discusion

Instance Labeling

- 2 groups
  - Objects (assigned unique index)
  - background (no index)
  - Need to label everything in a patch

Sparse Labeling

- 3 groups
  - Objects
  - Background
  - Not labeled

Sparse labeling

- Why ?
  - Save work
  - Better sampling

Sparse Labeling

- Why not ?
  - Fewer ‘edges’ between tightly clustered objects.

Sparse Labels

And the predictions

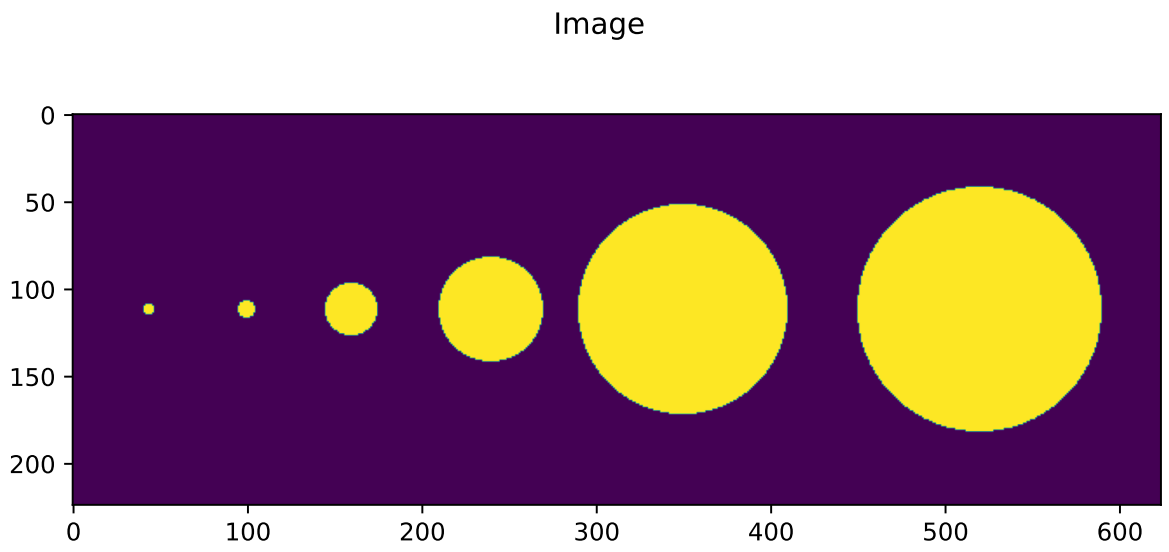
Cellpose trained to learn scale

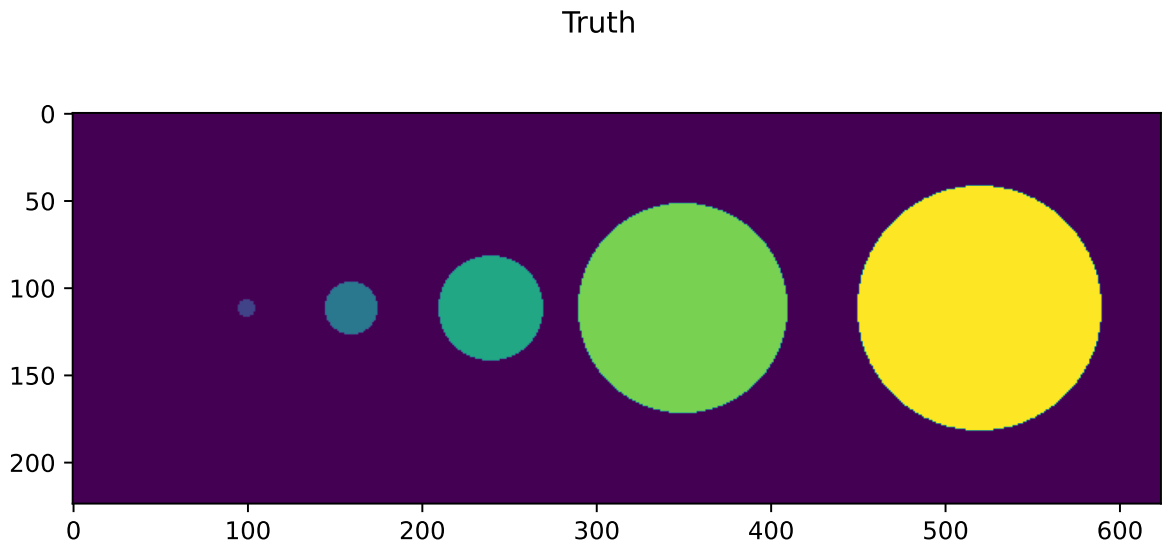
- Use same Cellpose model to detect spheres of varying sizes in the same image
- Common to use diameter parameter to rescale image
  - May not work if different scales in the same image

Phantom Image

```
import raster_geometry as rg
import numpy as np
from tnia.simulation.phantoms import add_small_to_large_2d
import matplotlib.pyplot as plt
from tnia.plotting.plt_helper import imshow_multi2d, imshow2d
import math
```

```
width, height = 624, 224

image = np.zeros([height, width], dtype=np.float32)
truth = np.zeros([height, width], dtype=np.float32)

rs = [3, 5, 15, 30, 60, 70]

x_ = 44

# for train will be a list of the 224 by 224 circle images that will be used for training
for_train = []

i = 0
for r in rs:
    x, y = x_, 112
    size = [math.ceil(r*2), math.ceil(r*2)]
    size = [224, 224]
    temp=rg.circle(size, r)
    for_train.append(temp)
    add_small_to_large_2d(image, temp, x, y, mode='replace_non_zero')
    add_small_to_large_2d(truth, i*temp, x, y, mode='replace_non_zero')
    x_ = x_ + 50+2*r
    i += 1


fig = imshow2d(image, width=8, height=3.5)
fig.suptitle('Image')
fig = imshow2d(truth, width=8, height=3.5)
stop=fig.suptitle('Truth')
```

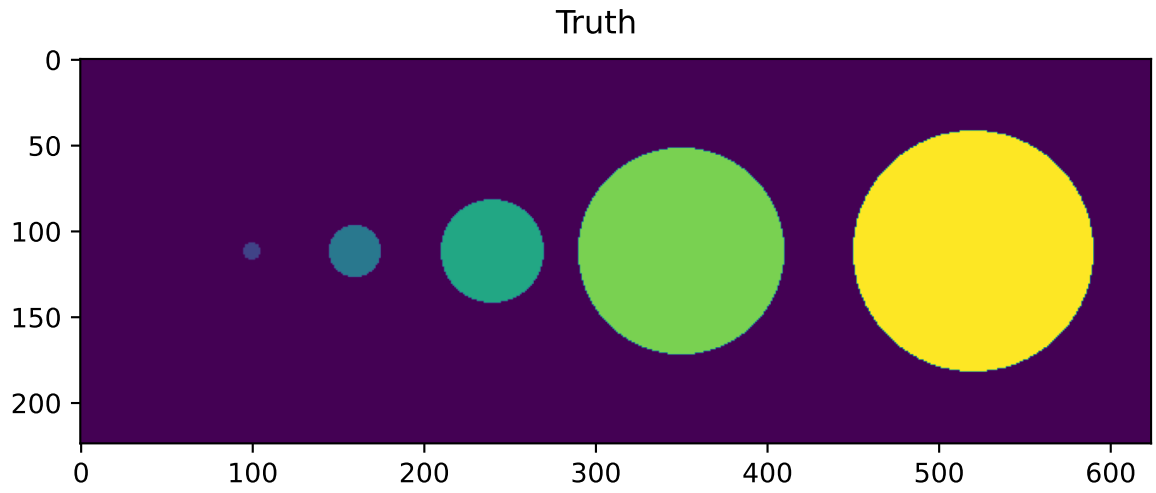


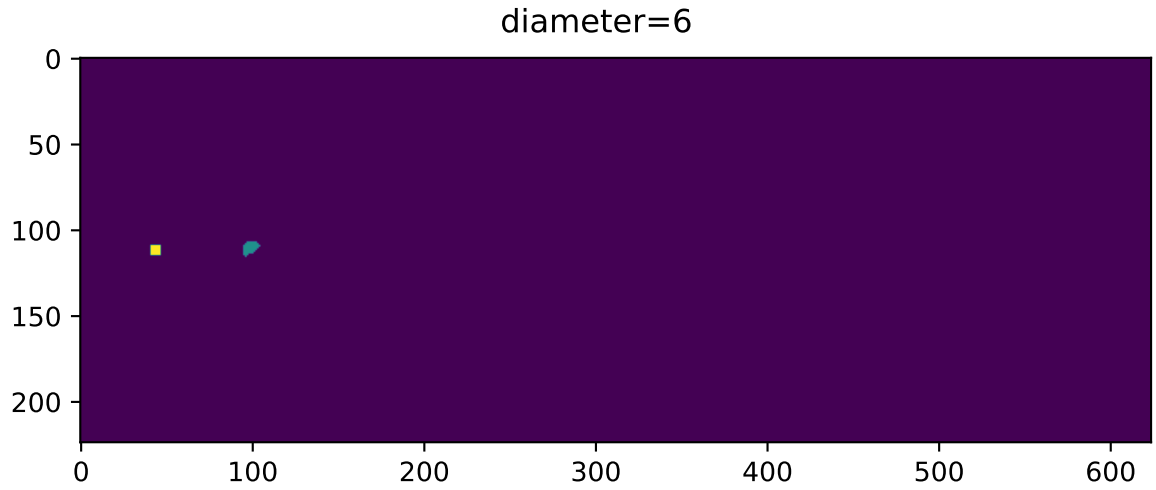


Process with Cyto2 and different diameters

diameter=6

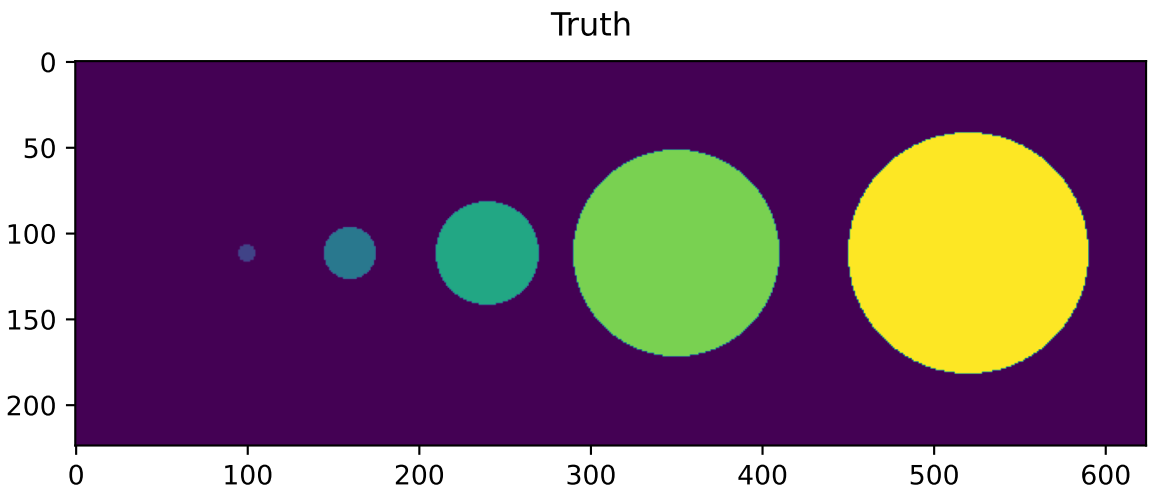
```
fig = imshow2d(truth, width=8, height=height)
stop=fig.suptitle('Truth')
fig = imshow2d(labels_d6, width=8, height=height)
stop=fig.suptitle('diameter=6')
```

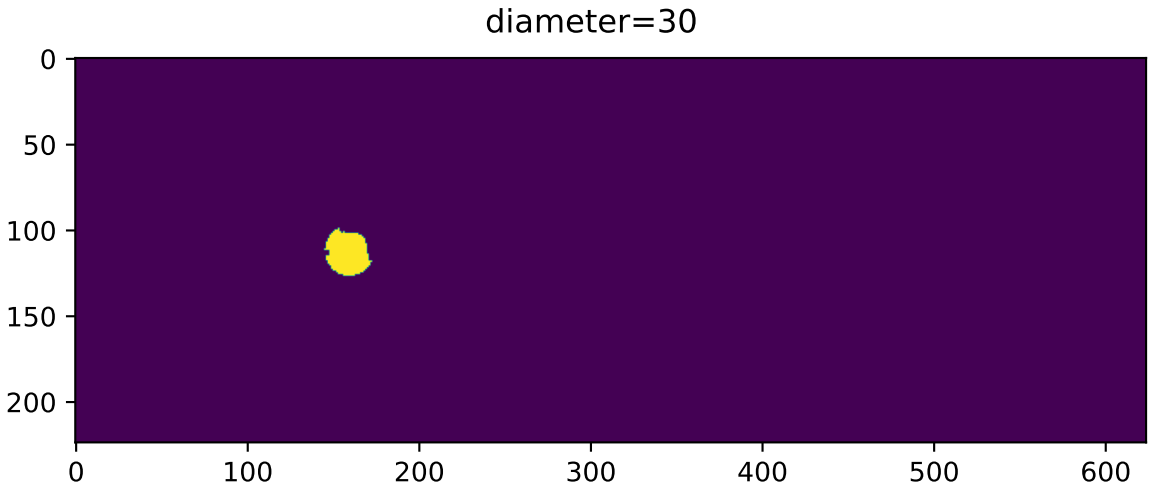




diameter=30

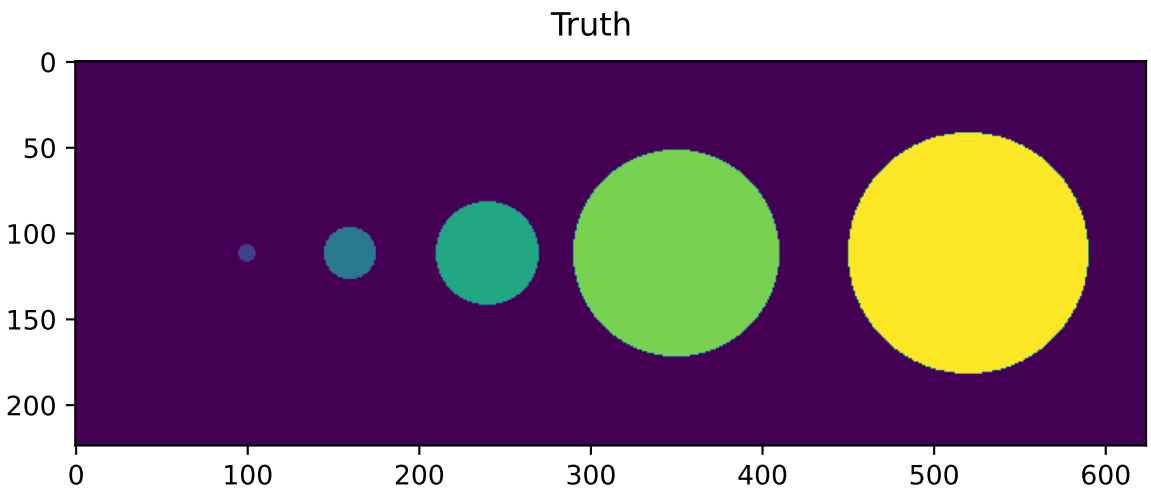
```
fig = imshow2d(truth, width=8, height=height)
stop=fig.suptitle('Truth')
fig = imshow2d(labels_default, width=8, height=height)
stop=fig.suptitle('diameter=30')
```

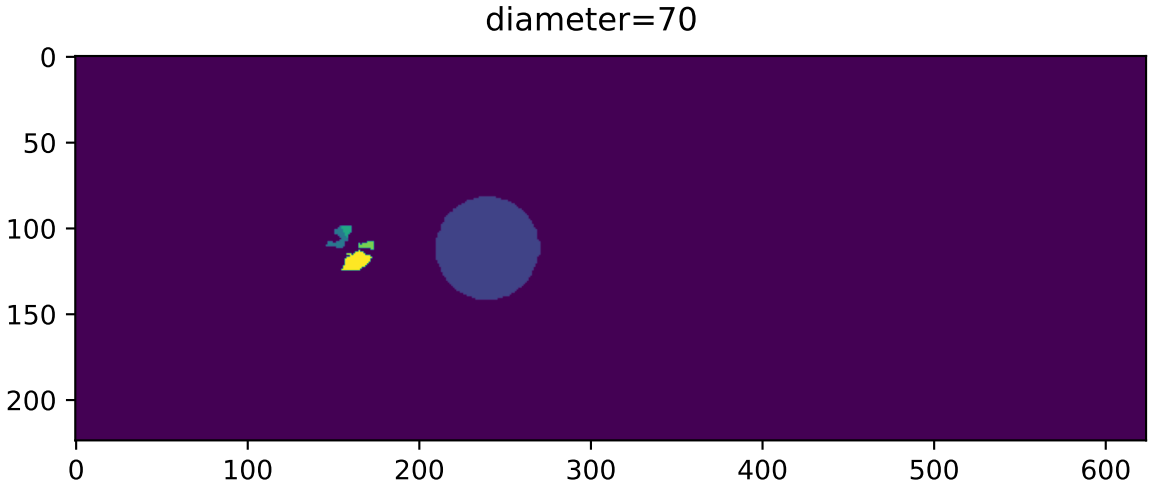




diameter=70

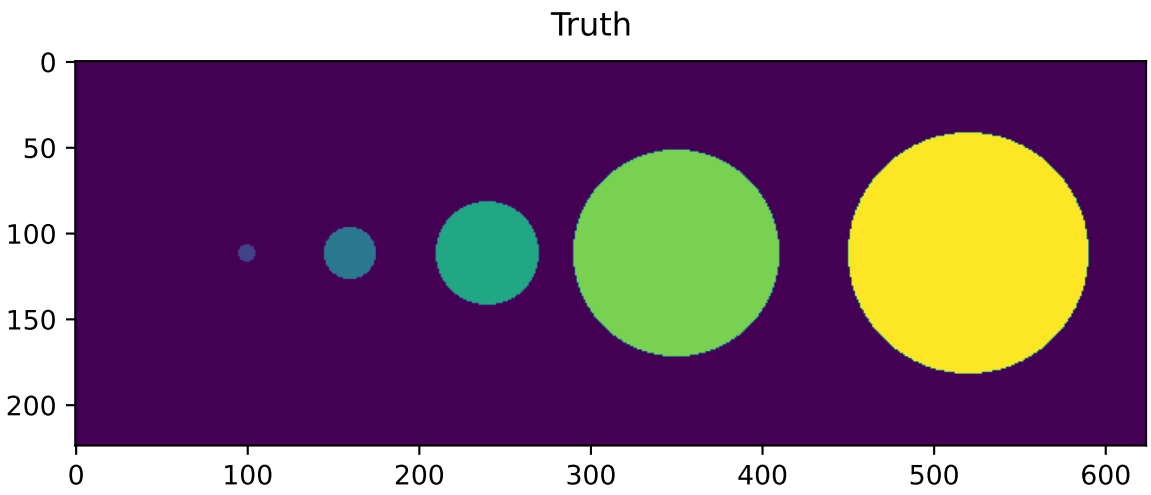
```
fig = imshow2d(truth, width=8, height=height)
stop=fig.suptitle('Truth')
fig = imshow2d(labels_d70, width=8, height=height)
stop=fig.suptitle('diameter=70')
```

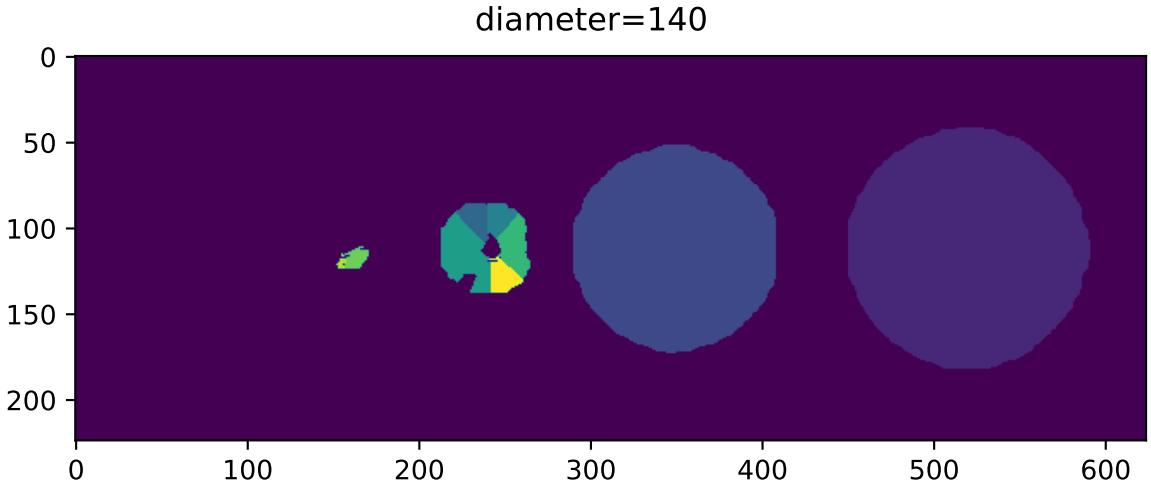




diameter = 140

```
fig = imshow2d(truth, width=8, height=height)
stop=fig.suptitle('Truth')
fig = imshow2d(labels_d140, width=8, height=height)
stop=fig.suptitle('diameter=140')
```



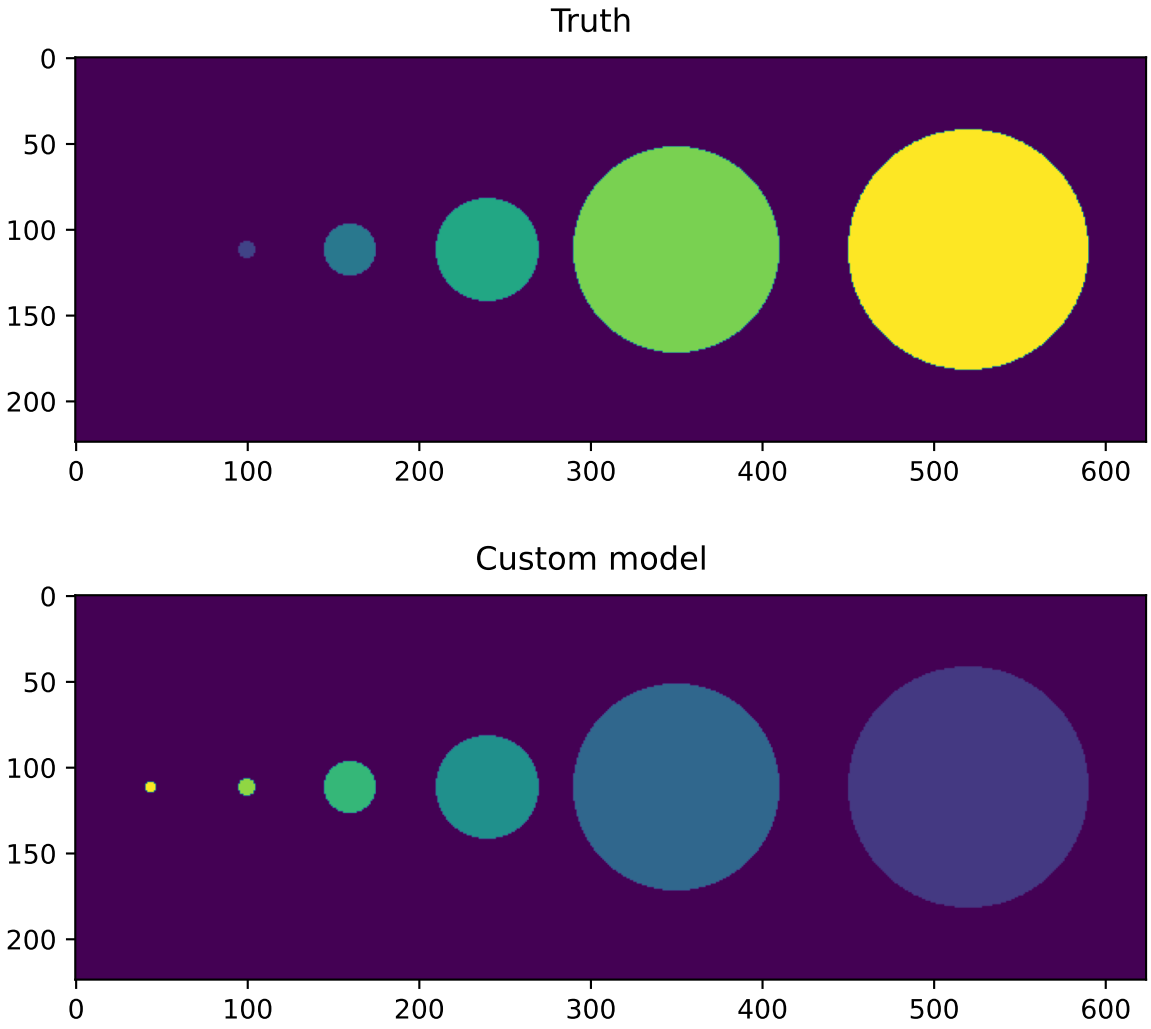


Train custom model

```
train.train_seg(model_custom.net, X, Y,
    channels=[0,0],
    save_path=model_path,
    n_epochs=180,
    min_train_masks=1,
    normalize = False,
    rescale = False,
    model_name=model_name)
```

Custom model (diameter=30)

```
fig = imshow2d(truth, width=8, height=height)
stop=fig.suptitle('Truth')
fig = imshow2d(labels_custom, width=8, height=height)
stop=fig.suptitle('Custom model')
```

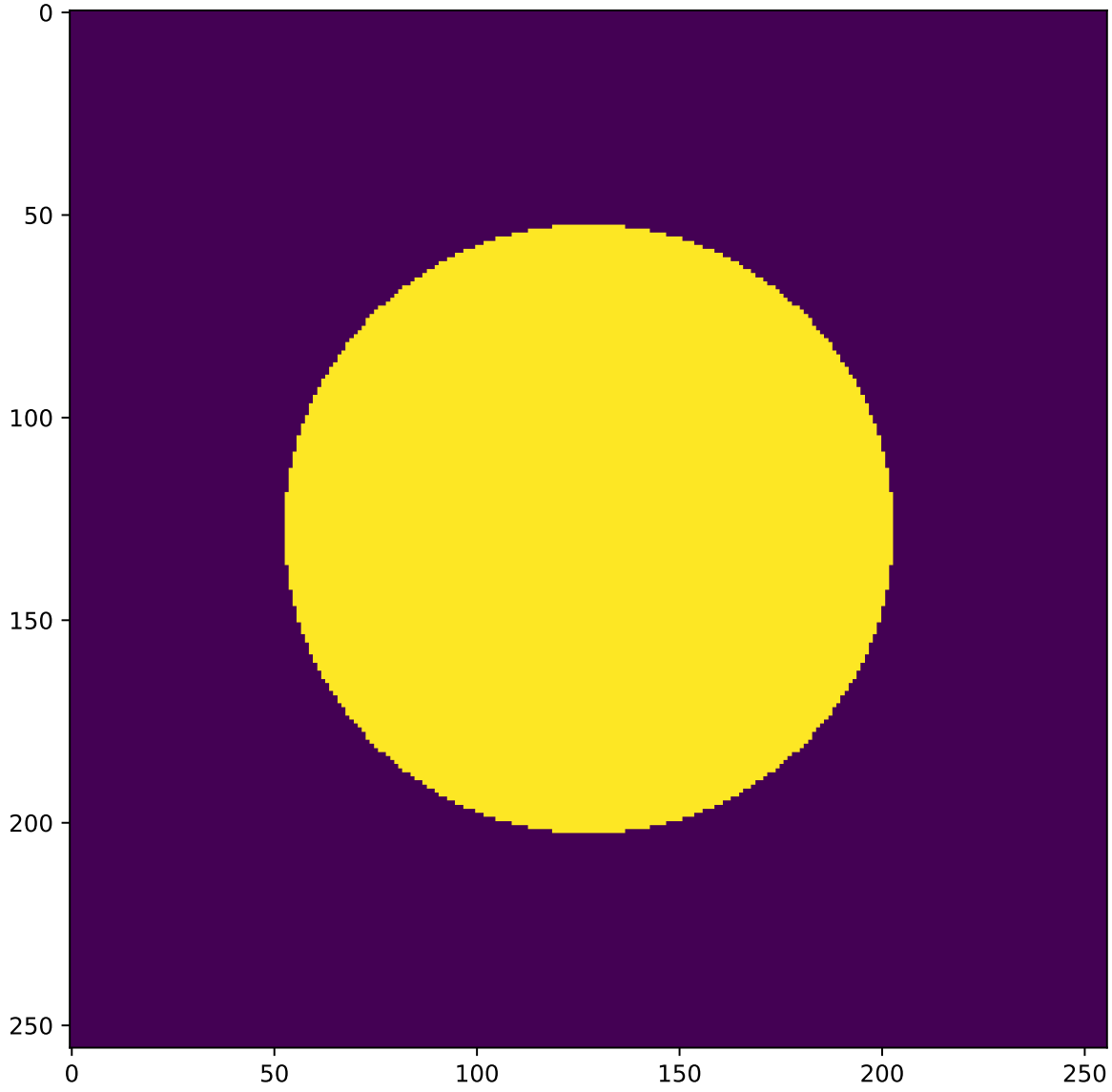


Stardist receptive field

- region of input that influences output
- impacted by striding, kernel size and number of layers

Create a 2D Object

```
import raster_geometry as rg
from tnia.plotting.plt_helper import imshow2d
import numpy as np
import matplotlib.pyplot as plt

radius = 75
xy_spacing = 1

image_size = 256
xy_pixels_radius = int(radius / xy_spacing)

#print(f'xy_pixels_radius {xy_pixels_radius}')

ellipsoid2D = rg.ellipse([image_size, image_size], [xy_pixels_radius, xy_pixels_radius]).asty

fig = imshow2d(ellipsoid2D, 8,8)
temp = fig.suptitle('ellipsoid2D')
```



Design Stardist 2D Network

- Larger grid
  - means larger stride thus 'see's' more 1st level

- Larger kernel
  - see's more each level
- More levels
  - each level images are downsampled
  - kernels at that level process larger spacial region

Empiricaly measure receptive field

- see stardist 3D training example notebook
- applies the model to a single point
- measures how far information - flows
  - doesn't matter if the network has been trained
  - just interested in information flow

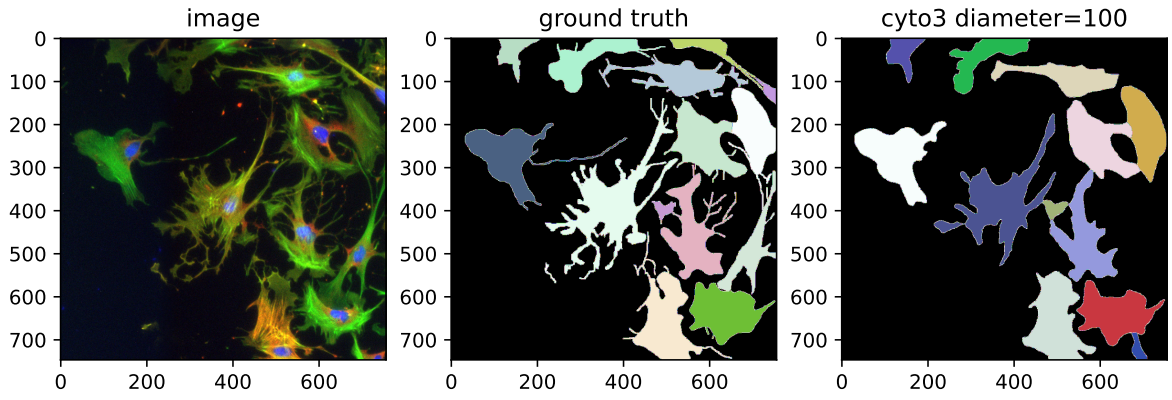
Large objects with protrusions

- Image set courtesy of Arkajyoti Sarkar
- See this Image.sc question

- Combination of large cells and long fine irregular protrusions

Cyto3 Results

```
fig = imshow_multi2d([img, ground_truth, masks_cyto3[0]], ['image', 'ground truth', 'cyto3 di
```



Challenges and considerations

- Need to retrain to detect protrusions better
- Cellpose rescales during training which could result in some loss of fine features.
- Receptive field? During training and prediction do we need the entire cell and protrusions in receptive field of neural network?
  - Receptive field may be more important in stardist, Cellpose 'seems' to predict good flows with tiling.

Self-prediction

- Train on one image, predict on same image.
- Not going to get good generalization this way!
  - but… a useful experiment to determine if model can code structure of interest.
- Try to tune parameters to get 'perfect' self-prediction then move on to training a real model that generalizes.

Training Strategy

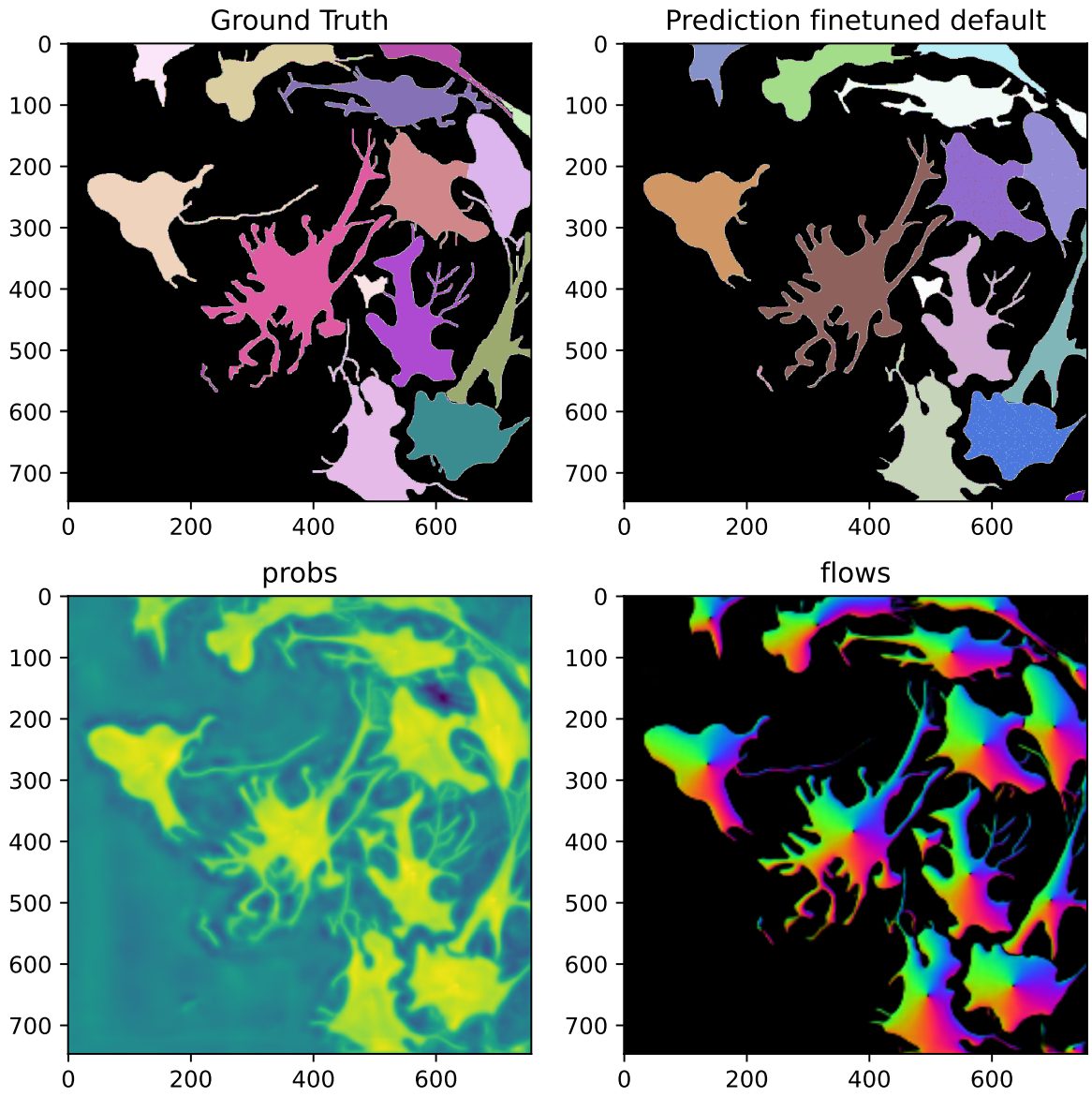
- Turn rescaling off to avoid losing resolution and this features.
- Use larger bsize (crop size) to get more information in each training patch.
  - (turned out this may not be that important)

Training code

```
train.train_seg( model_finetuned_no_rescale_bsize_512.net, X_list, Y_list,
                 channels=[2,3],
                 save_path=parent_path,
                 n_epochs=500,
                 min_train_masks=0,
                 rescale = False,
                 model_name='no_rescale_bsize_512',
                 normalize=False,
                 bsize=512)
```

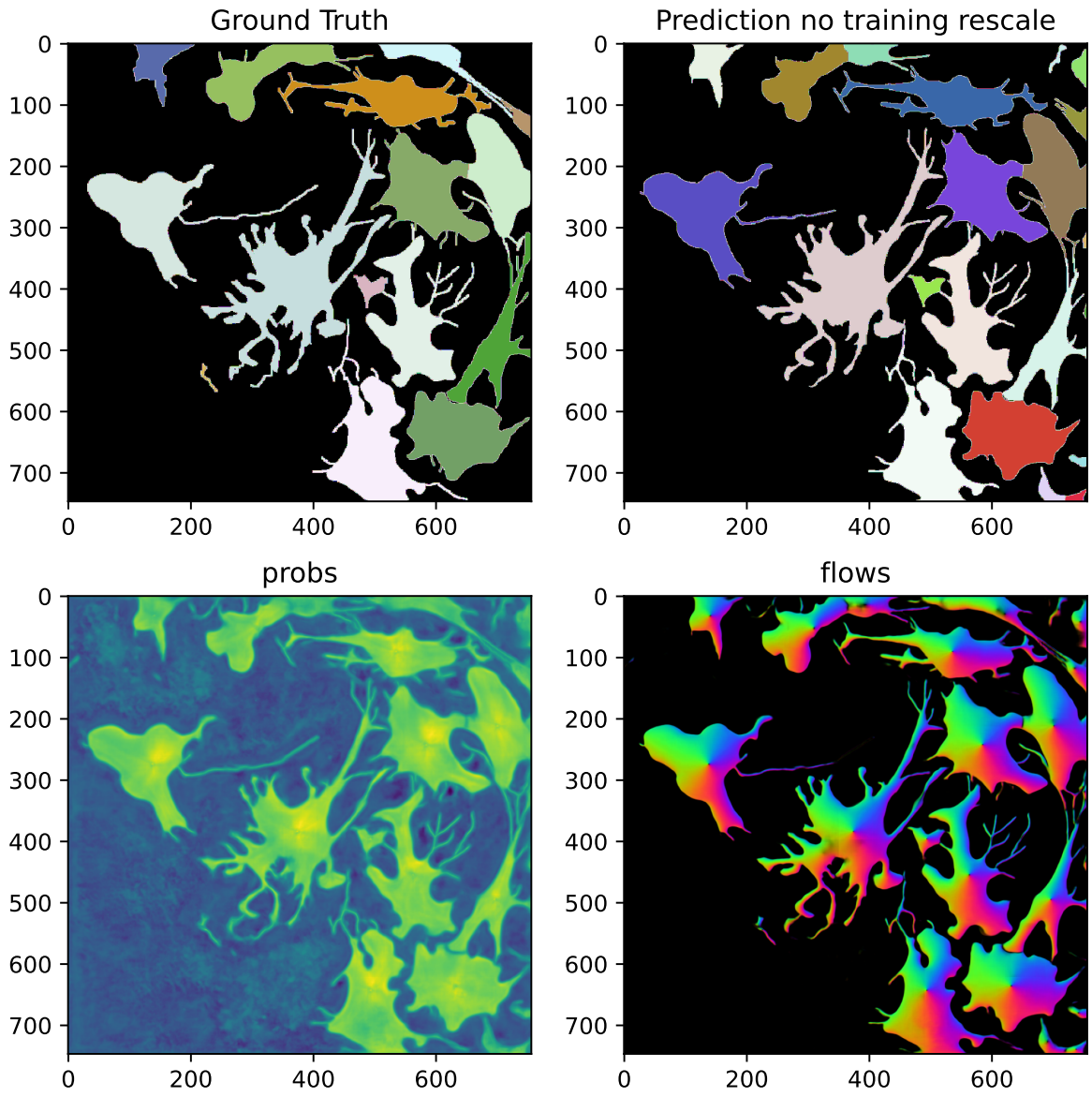
Retrained defaults

```
fig = imshow_multi2d([ground_truth, masks_defaults[0], masks_defaults[1][2], masks_defaults[
```



Retrained no rescaling

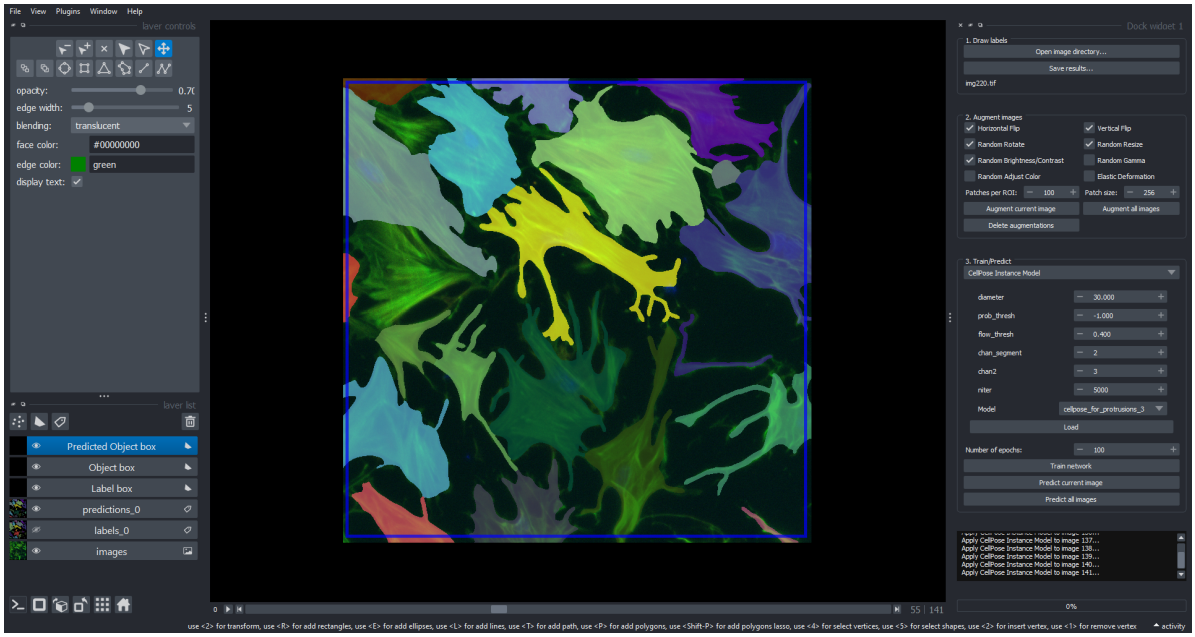
```
mask_no_rescale = model_finetuned_no_rescale_bsize_512.eval(img, diameter=30, niter=2000, cha
fig = imshow_multi2d([ground_truth, mask_no_rescale[0], mask_no_rescale[1][2], mask_no_rescal
```



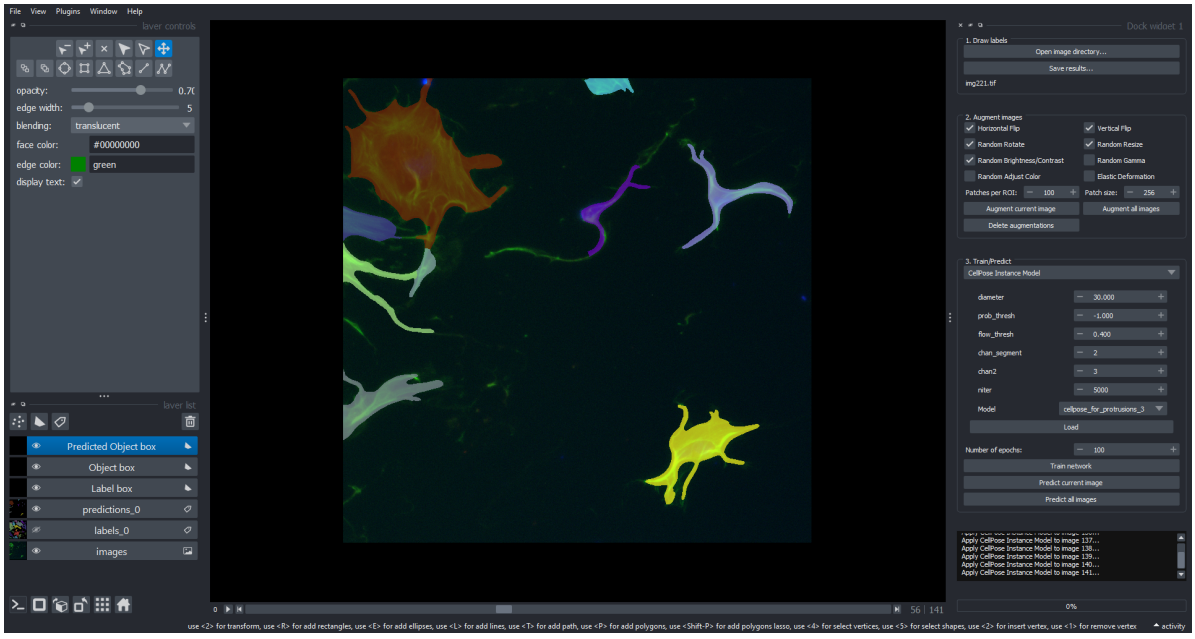
Model training on sequence

- 140 images
- 10 labeled
- no rescaling during training
- bsize = 512
- niter = 5000 during prediction

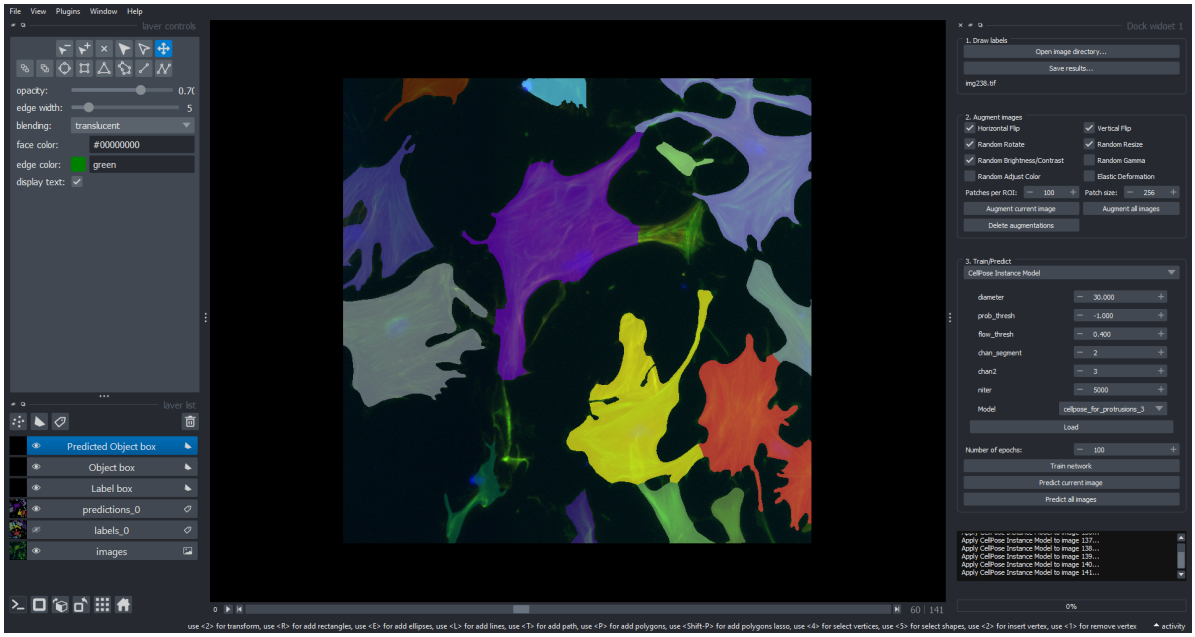
Model training on sequence



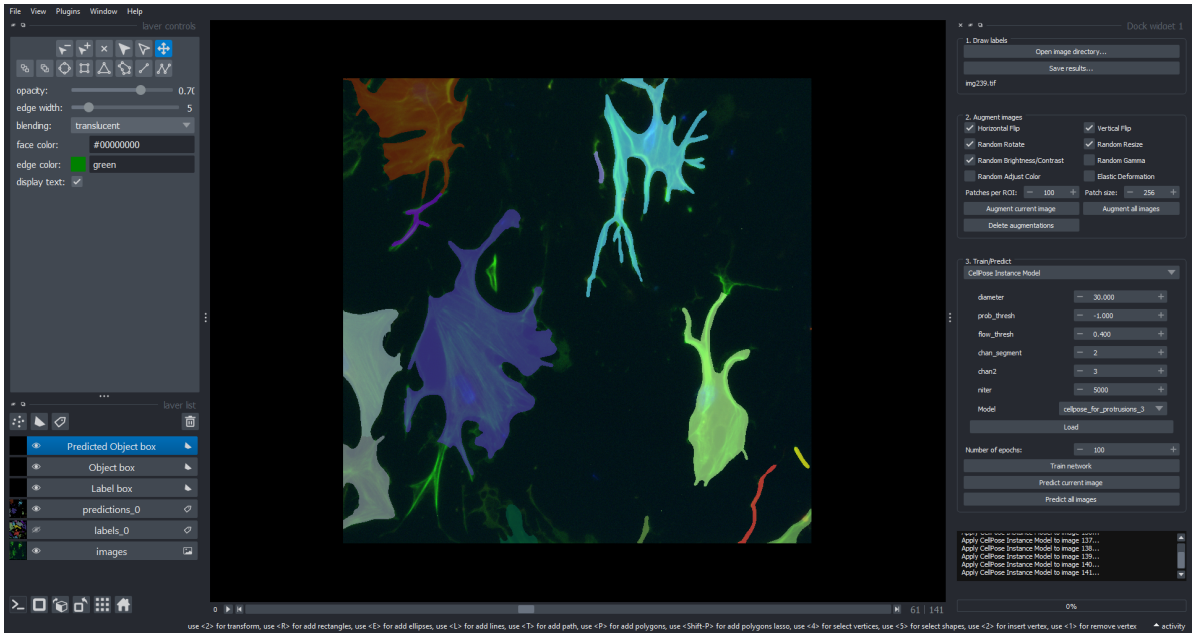
Model training on sequence



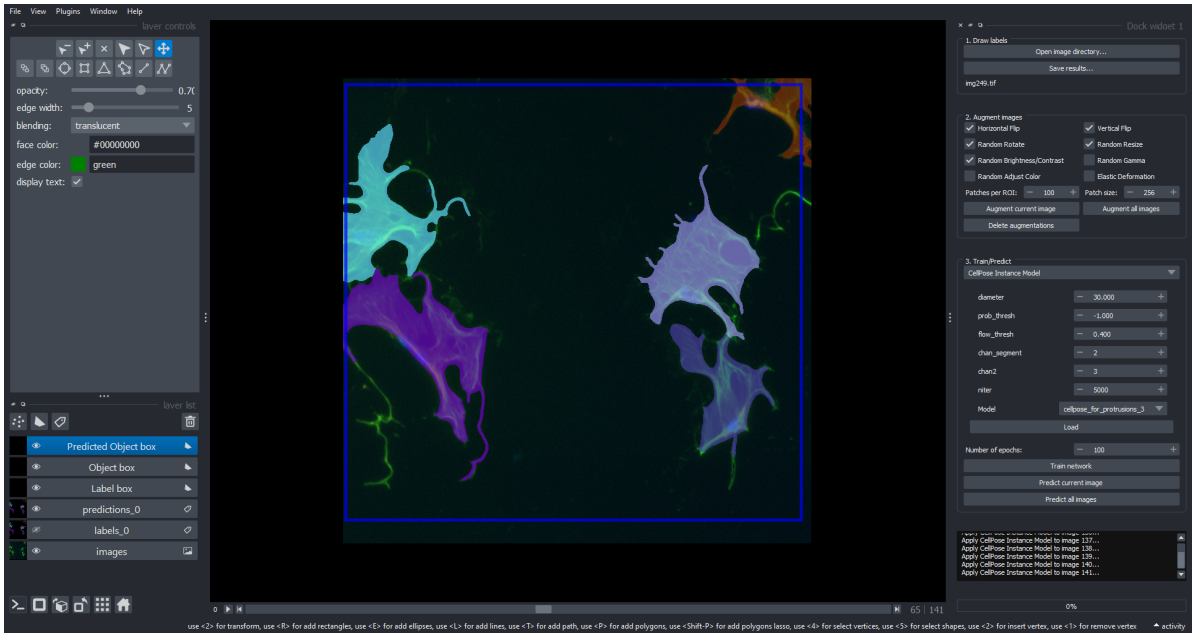
Model training on sequence



Model training on sequence



Model training on sequence



Ladybugs at different scales

How to handle this ?

- Scale related parameters
  - Cellpose `diameter`
  - Stardist `scale`
- What if hundreds of images ?
  - Can we train for different scales ?

Self-Prediction

- predict the training set

Validation

- predict other images in series

What next ?

- only 3 training images !
- correct and relabel
  - bigger training set
- train for longer
  - more training time needed for larger objects ?
- consider auto-rescaling ?
- consider other models

Segment Everything ?

- Can we segment everything with one generalist model ?
  - maybe someday ?

What is SAM ?

- SAM stands for Segment Anything Model.
- Prompt based generalist image segmentation.
  - Prompt is hint
  - can be a point, bounding box, or even a mask (first guess)

What is SAM ?

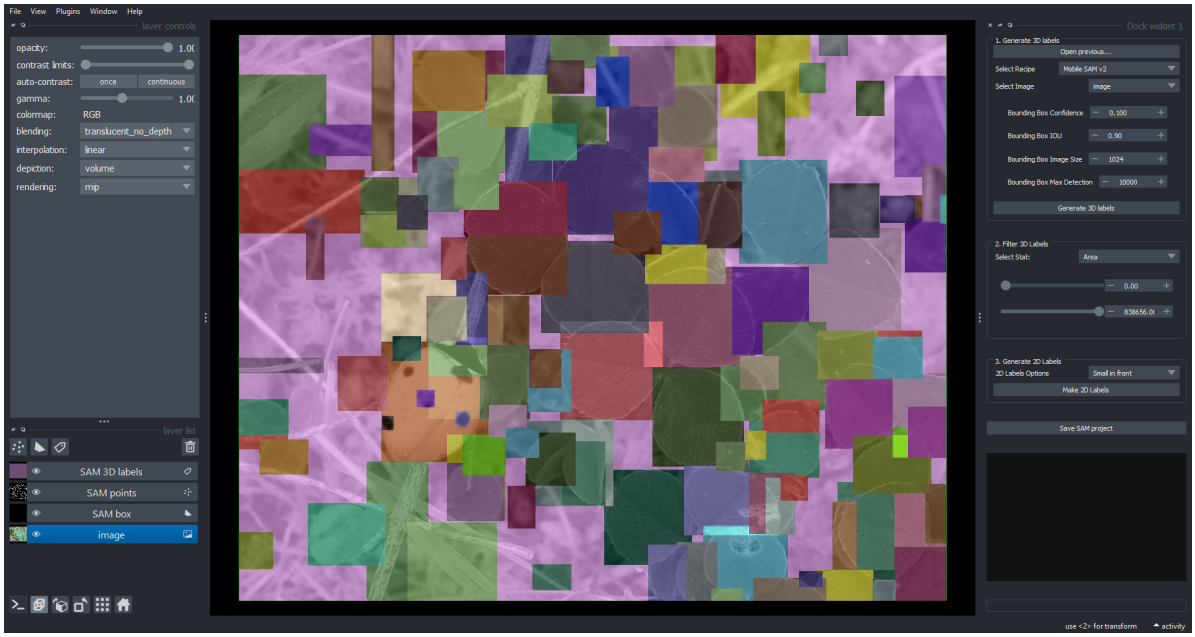
- Good at detecting overlapping objects (spots on Ladybugs)
  - multiple prompts -> multiple masks
  - masks may overlap
  - if we want 2D label image need to 'project' 3D collection to 2D

Viewing and exploring overlapping labels

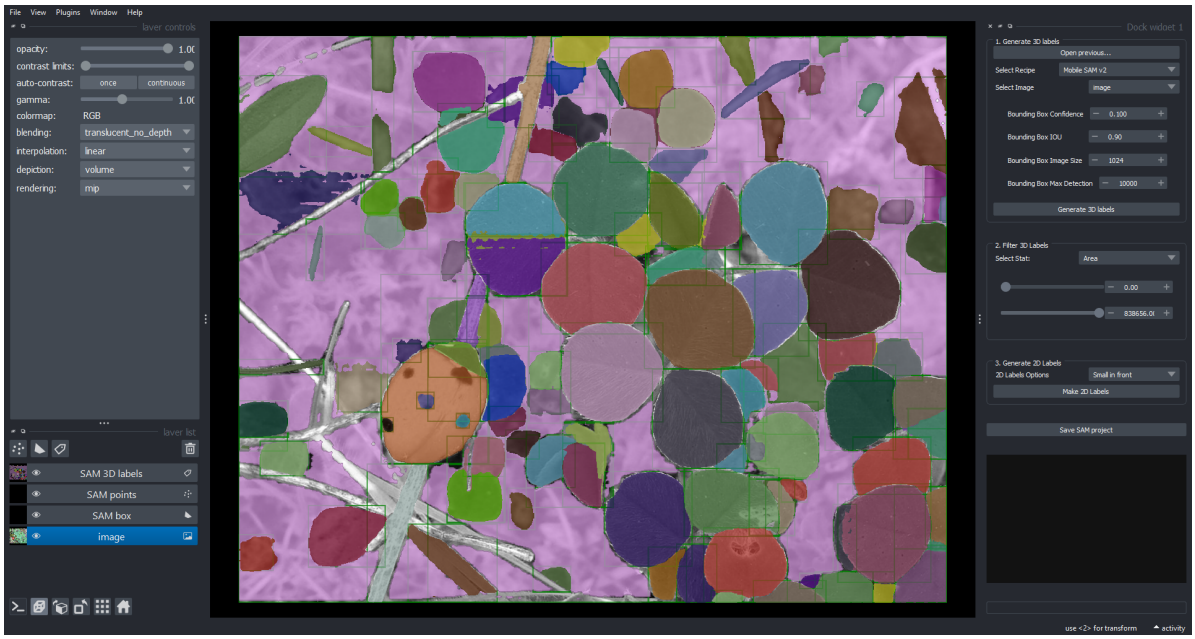
- this example uses mobile SAM and napari-segment-everything

Viewing and exploring overlapping labels

Segment bounding boxes



SAM with Yolo bbs as prompt



Rotate to see relationships

Pan and zoom

Filtering Labels

- SAM can return hundreds or thousands of labels.
- Many of the labels may not be of interest.

Strategies to Filter labels

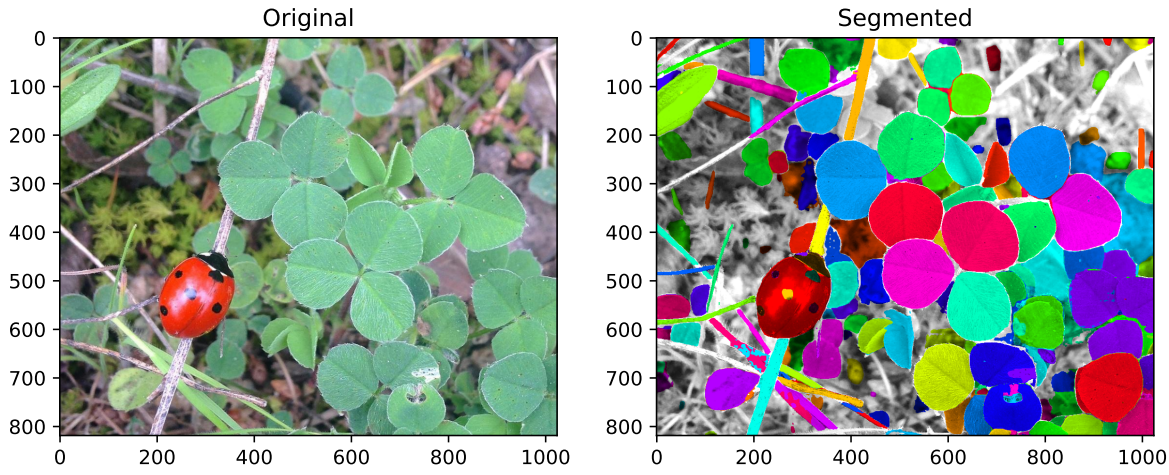
- Train the prompt detector to also classify objects
  - YOLO can be trained to classify then only keep the prompts corresponding to classes of interest.
- Filter the labels by a criteria so we only keep objects of interest.
- This example shows the latter strategy.

```
from segment_everything.detect_and_segment import segment_from_stacked_labels
from segment_everything.prompt_generator import YoloDetector
from segment_everything.weights_helper import get_weights_path
from segment_everything.stacked_labels import StackedLabels
from segment_everything.detect_and_segment import segment_from_stacked_labels
import matplotlib.pyplot as plt
from tnia.plotting.plt_helper import random_label_cmap, mask_overlay, imshow_multi2d
import numpy as np
from skimage.io import imread
import os

conf = 0.1
iou = 0.8
imagesz = 4096
descriptor = "MobileSAM Model"
boxes = True

yolo_detecter = YoloDetector(str(get_weights_path("ObjectAwareModel")), "ObjectAwareModelFro

data_path = r'./data'
parent_path = os.path.join(data_path, 'ladybugs_SAM')
#image_name = os.path.join(parent_path, '5784124_7185843.jpg')
```

```
image_name = os.path.join(parent_path, '620818_780868.jpg')
img = imread(image_name)
```

Original and Segmented

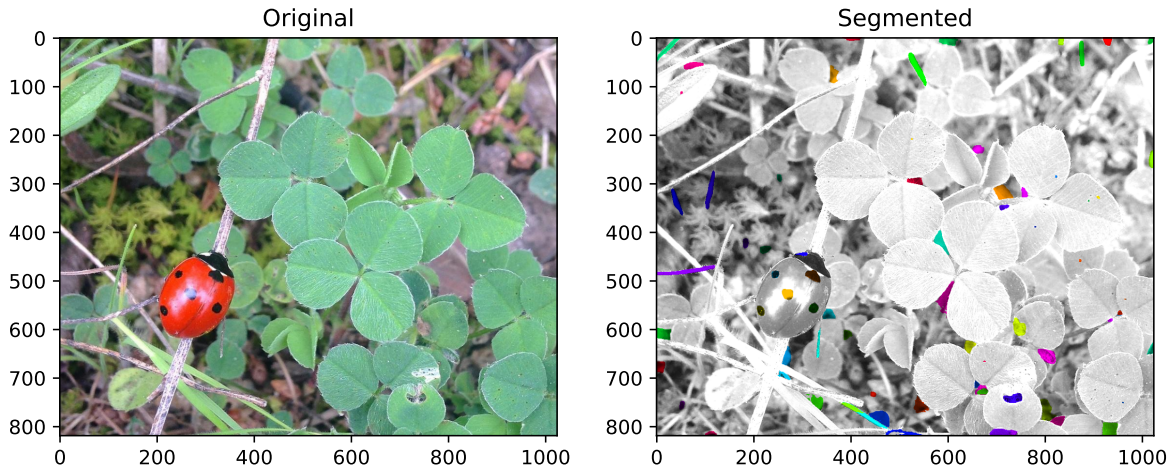
```
overlay_img = mask_overlay(img, labels)

#plt.imshow(overlay_img, cmap=random_label_cmap())
fig = imshow_multi2d([img, overlay_img], ["Original", "Segmented"], 1, 2)
```



Filter

```
#add_properties_to_label_image(img, segmented_stacked_labels.mask_list)
segmented_stacked_labels.add_properties_to_label_image()

stat_limits = {
    "area": {"min": 0, "max": 1000},
    "label_num": {"min": -100000000, "max": 10000000000},
    "solidity": {"min": 0.0, "max": 1.0},
    "circularity": {"min": 0.0, "max": 1.0},
    "mean_intensity": {"min": 0, "max": 255},
    "10th_percentile_intensity": {"min": 0, "max": 255},
    "mean_hue": {"min": 0, "max": 360},
```

```
    "mean_saturation": {"min": 0.0, "max": 1000},
    "predicted_iou": {"min": 0.0, "max": 1.0},
    "stability_score": {"min": 0.0, "max": 1.0}
}

segmented_stacked_labels.filter_labels_3d_multi(stat_limits)
labels = segmented_stacked_labels.make_2d_labels(type="min")

overlay_img = mask_overlay(img, labels)

#plt.imshow(overlay_img, cmap=random_label_cmap())
fig = imshow_multi2d([img, overlay_img], ["Original", "Segmented"], 1, 2)
```



Look at labels

Overlays