AI for Genomics: from CNNs and LSTMs to Transformers

Nikolay Oskolkov, Group Leader (PI) at LIOS, Riga, Latvia

Physalia course, 09.09.2025

Session 2a: Convolutional Neural Networks (CNNs) applications for Microscopy Imaging

@NikolayOskolkov

@oskolkov.bsky.social

Personal homepage:
https://nikolay-oskolkov.com

Image generated at www.canva.com

Session content

Topics we'll cover in this session:

1) Introduction to Convolutional Neural Networks (CNNs)

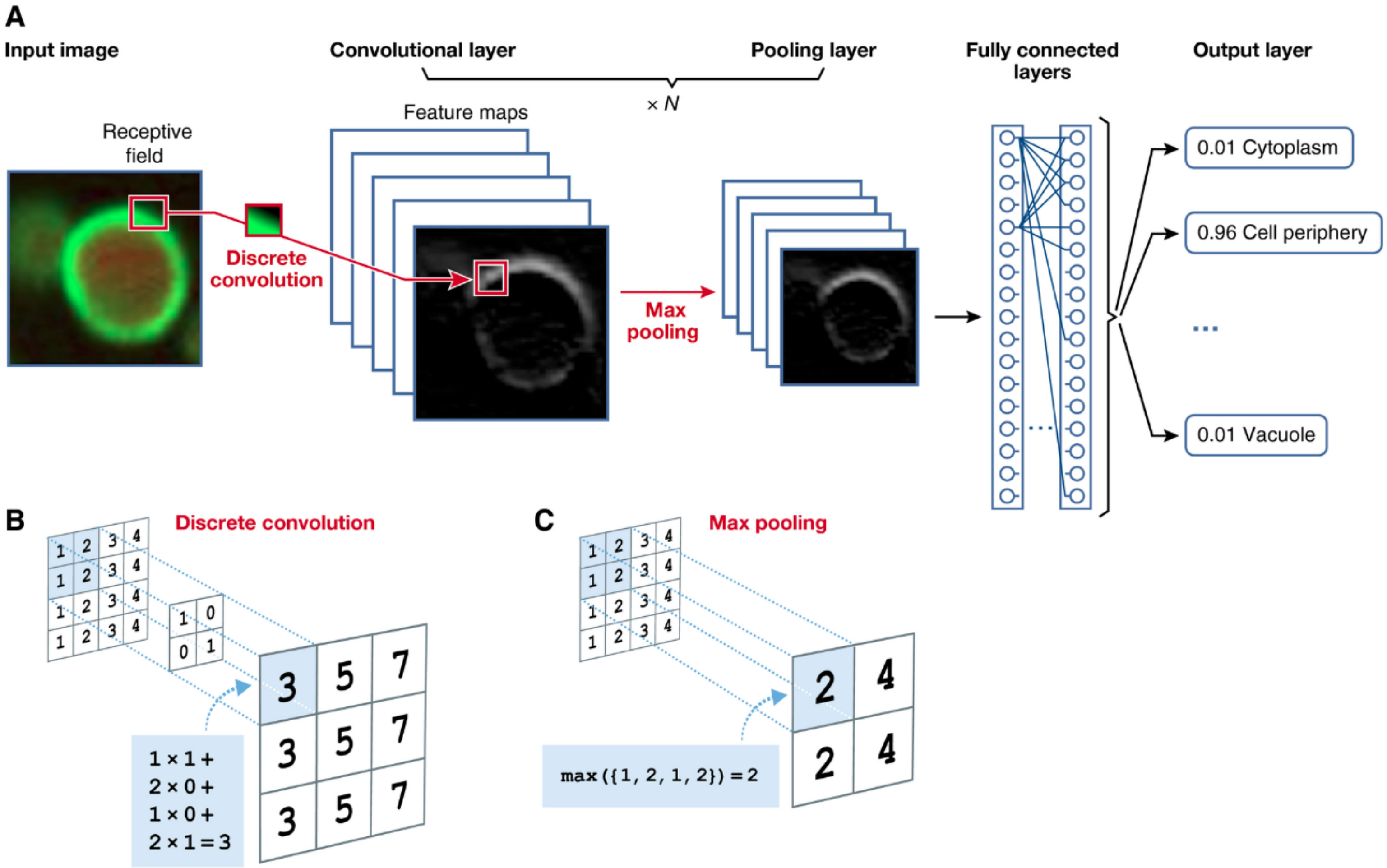
2) Microscopy imaging: object detection and segmentation

3) Grad-CAM for interpreting Deep Learning models

4) Unsupervised clustering of images and batch detection

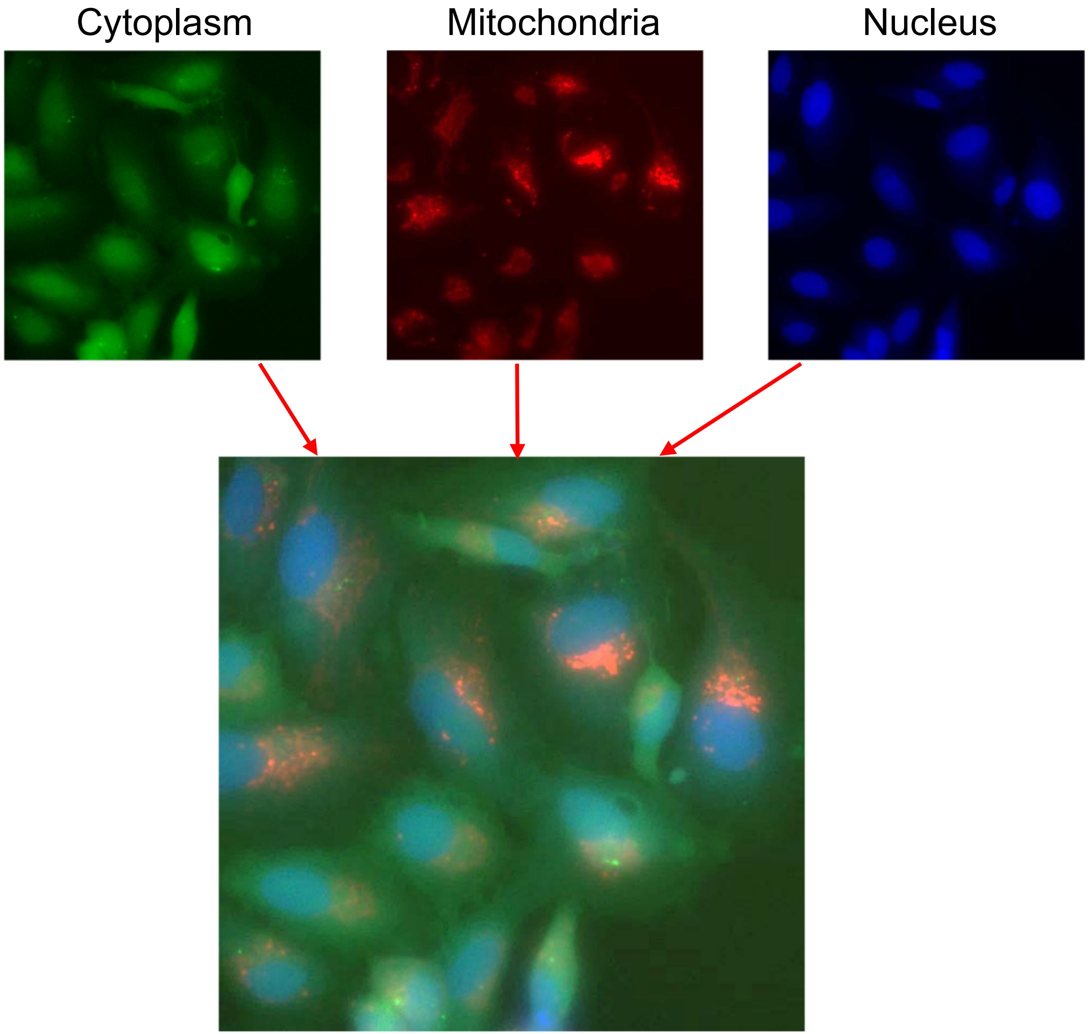
5) Comparing Faster RCNN and Mask-RCNN for object detection

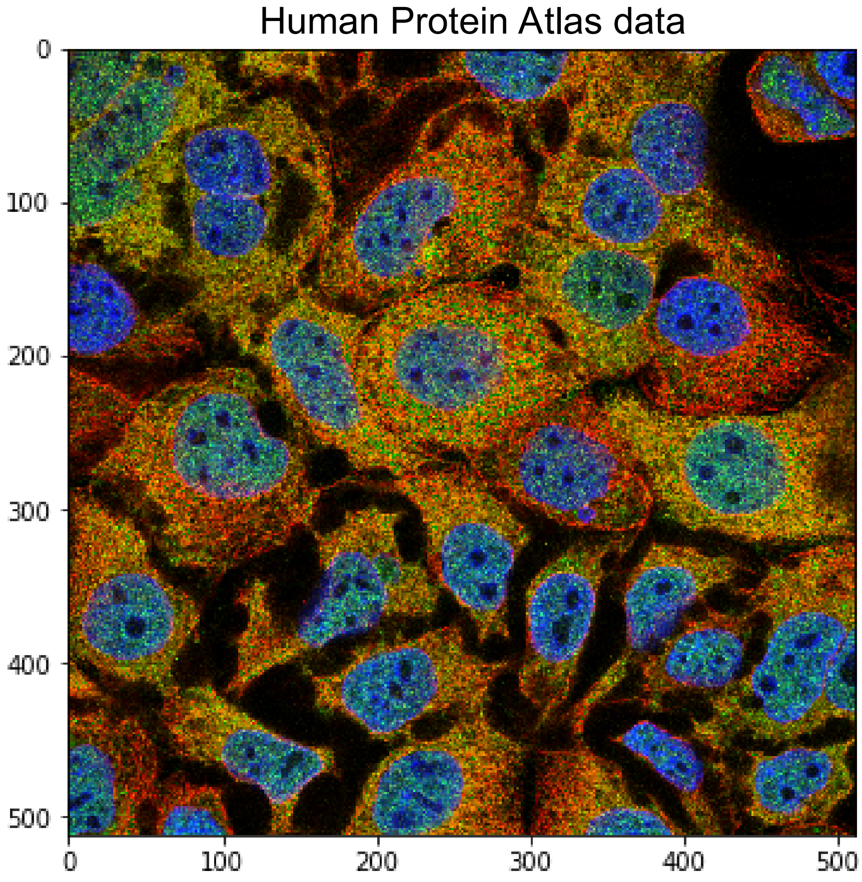
Convolutional Neural Networks (CNNs)



Angermueller et al., Molecular Systems Biology 2016

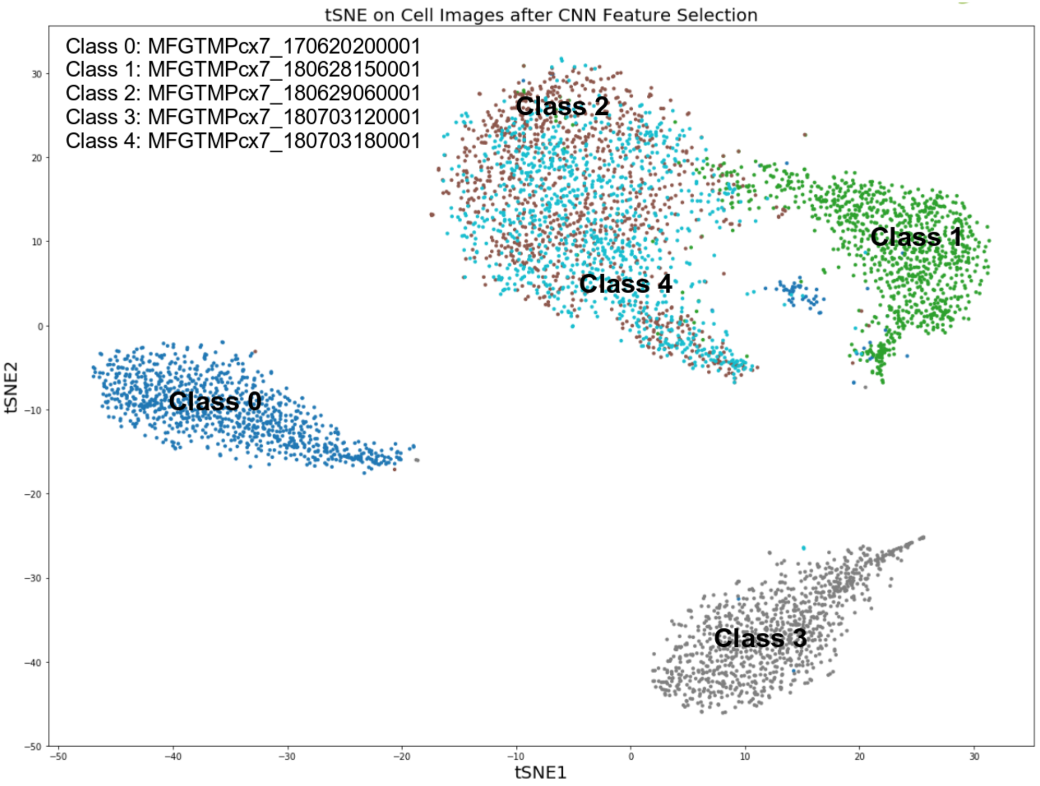
Merging 3 RGB channels, and even 4 channels

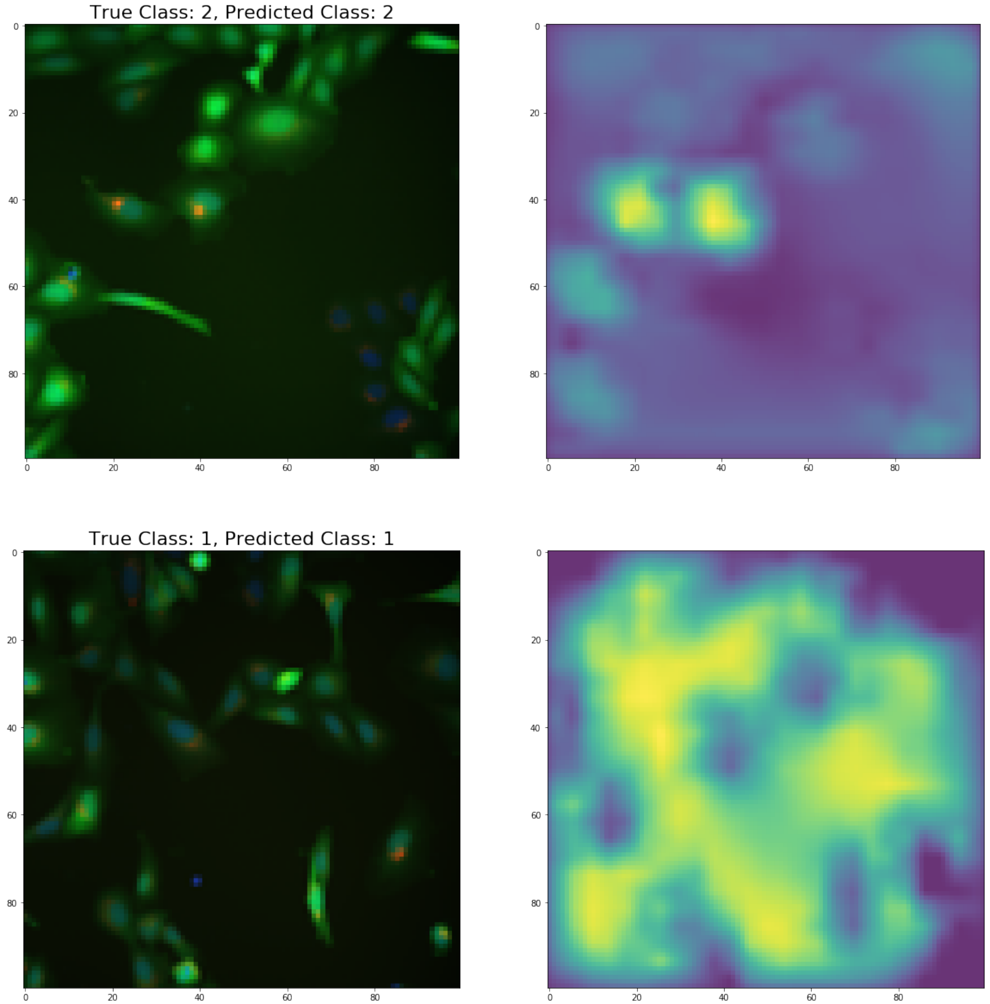




One way to merge the 4 channels in Python would be to realize that yellow = red + green and add half of the yellow channel to the red and the other half to the green channel.

Grad-CAM to interpret image clusters





Different batches had different background

OpenCV-Python Tutorials

Rotation

Rotation of an image for an angle θ is achieved by the transformation matrix of the form

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

But OpenCV provides scaled rotation with adjustable center of rotation so that you can rotate at any location you prefer. The modified transformation matrix is given by

$$\begin{bmatrix} \alpha & \beta & (1- \alpha) \cdot center.x - \beta \cdot center.y \\ - \beta & \alpha & \beta \cdot center.x + (1- \alpha) \cdot center.y \end{bmatrix}$$

where:

$$\begin{array}{l} \alpha = scale \cdot \cos \theta , \\ \beta = scale \cdot \sin \theta \end{array}$$

To find this transformation matrix, OpenCV provides a function, **cv.getRotationMatrix2D**. Check out the below example which rotates the image by 90 degree with respect to center without any scaling.

```
img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
rows,cols = img.shape

# cols-1 and rows-1 are the coordinate limits.
M = cv.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),90,1)
dst = cv.warpAffine(img,M,(cols,rows))
```

See the result:

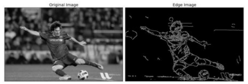
image

Canny Edge Detection in OpenCV

OpenCV puts all the above in single function, **cv.Canny()**. We will see how to use it. First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Fourth argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is L2gradient which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above which is more accurate, otherwise it uses this function: $Edge\_Gradient \; (G) = |G_x| + |G_y|$. By default, it is False.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
edges = cv.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

See the result below:

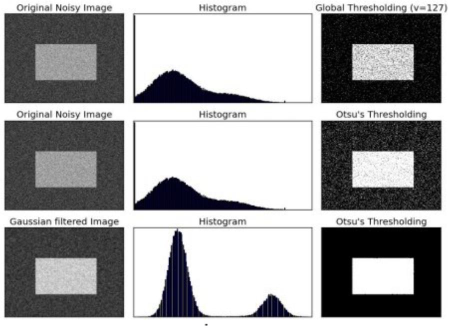


image

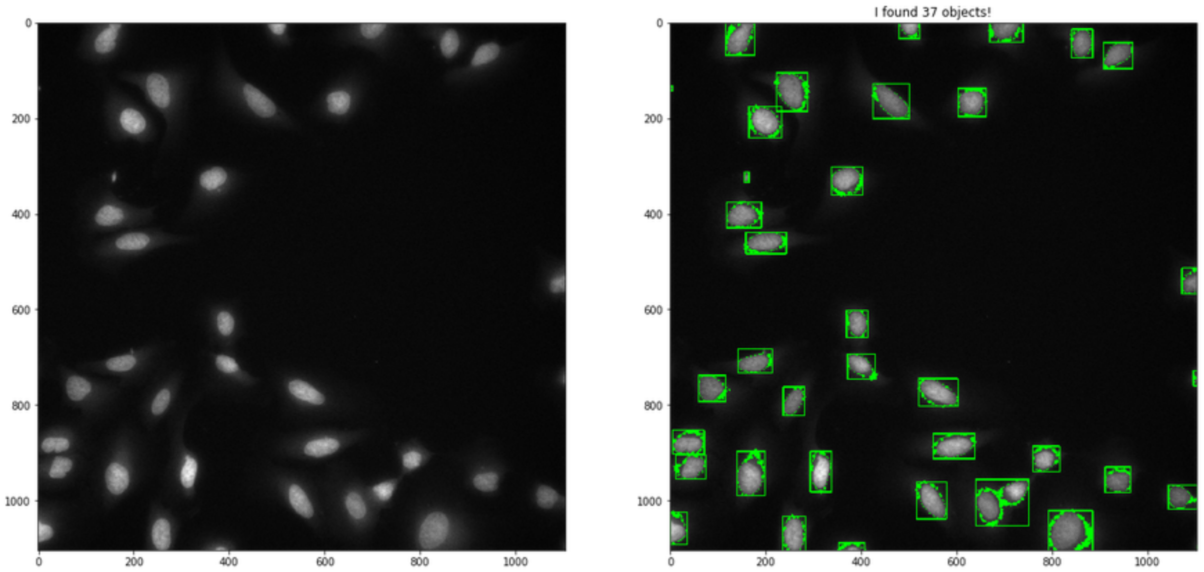
Examples | Java documentation

```
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('noisy2.png', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"

# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

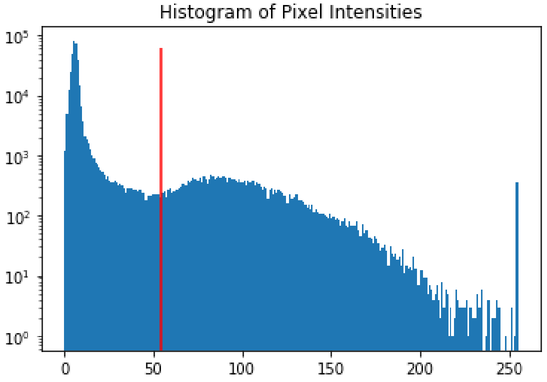
Result:

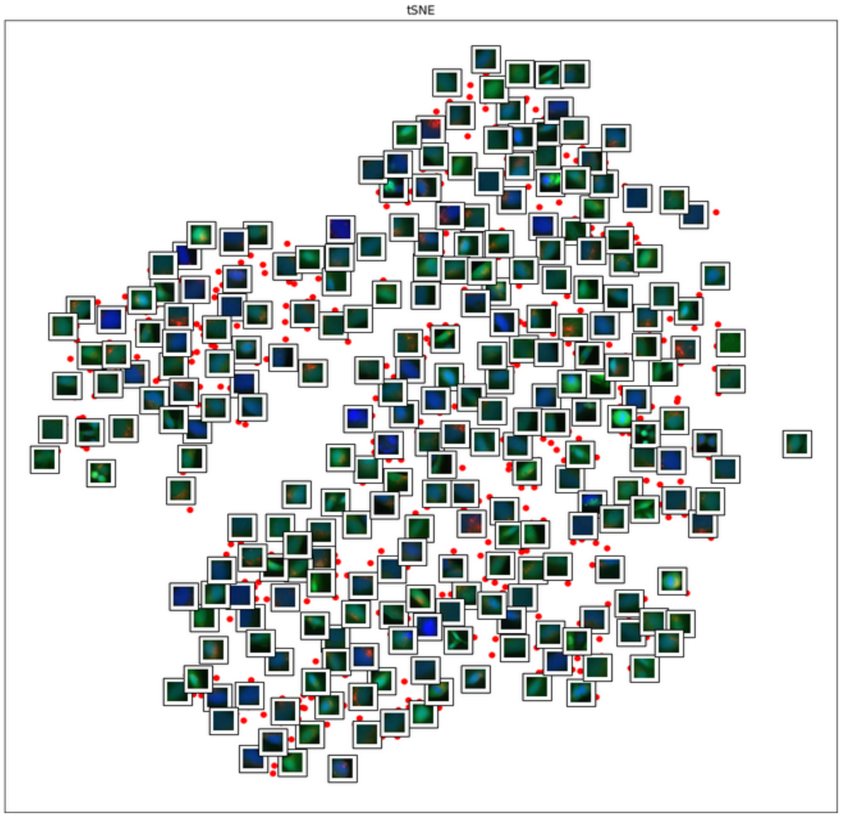


No annotation: unsupervised cell clustering

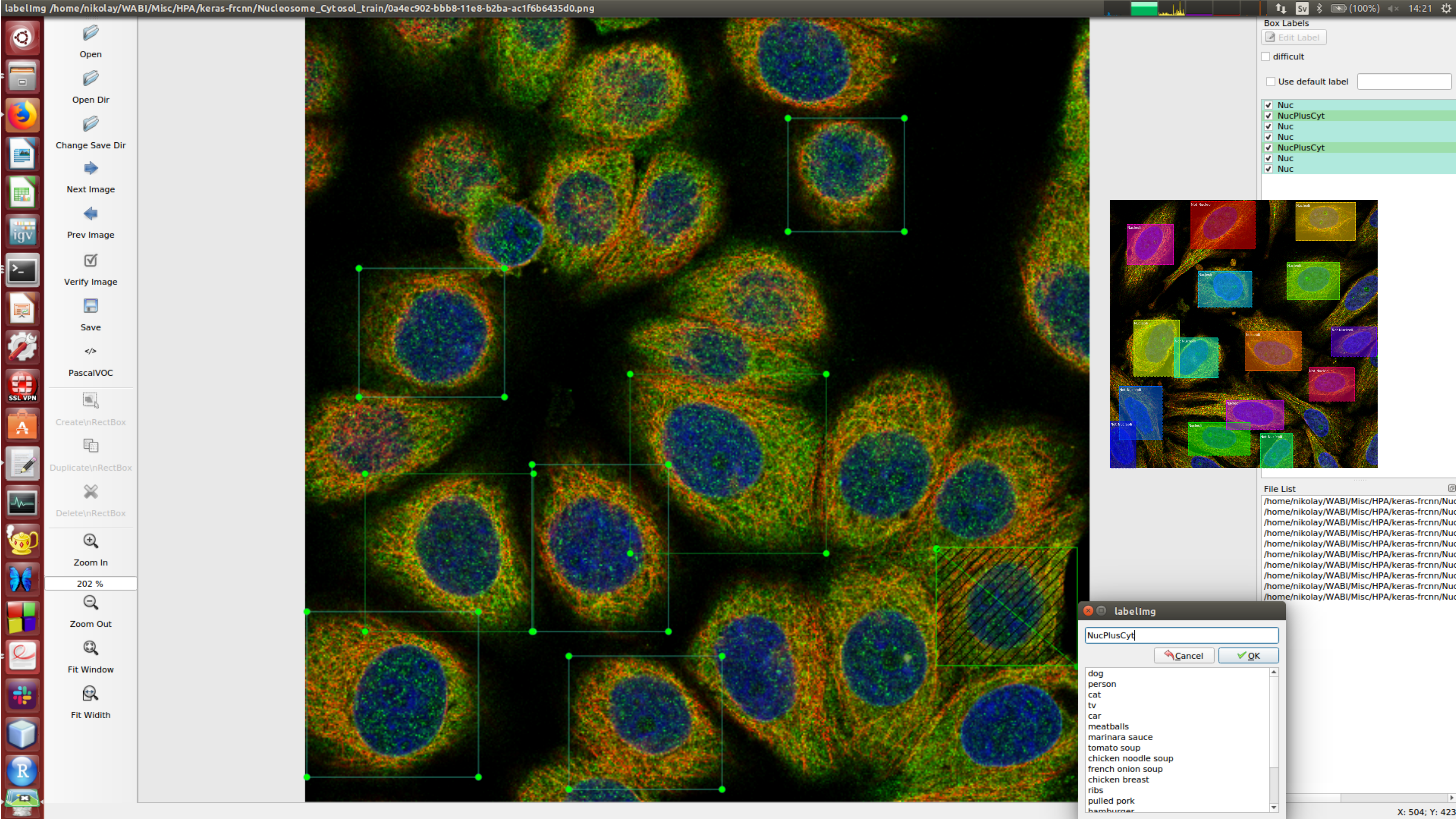


Cells were located by thresholding and cropped.

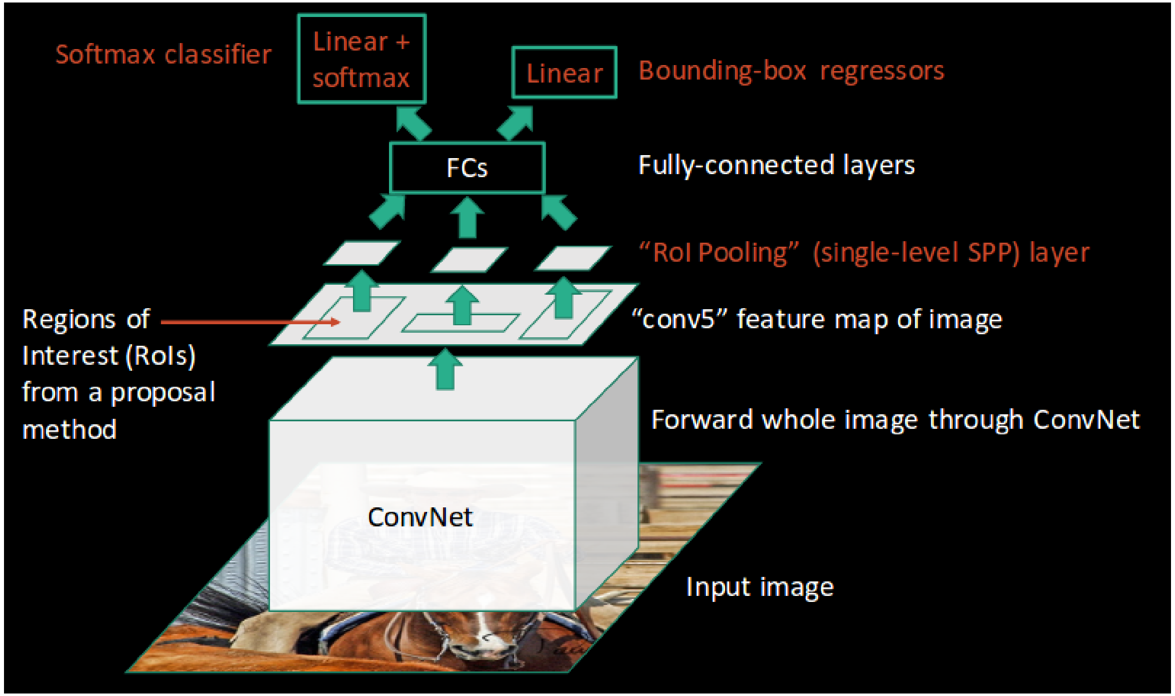


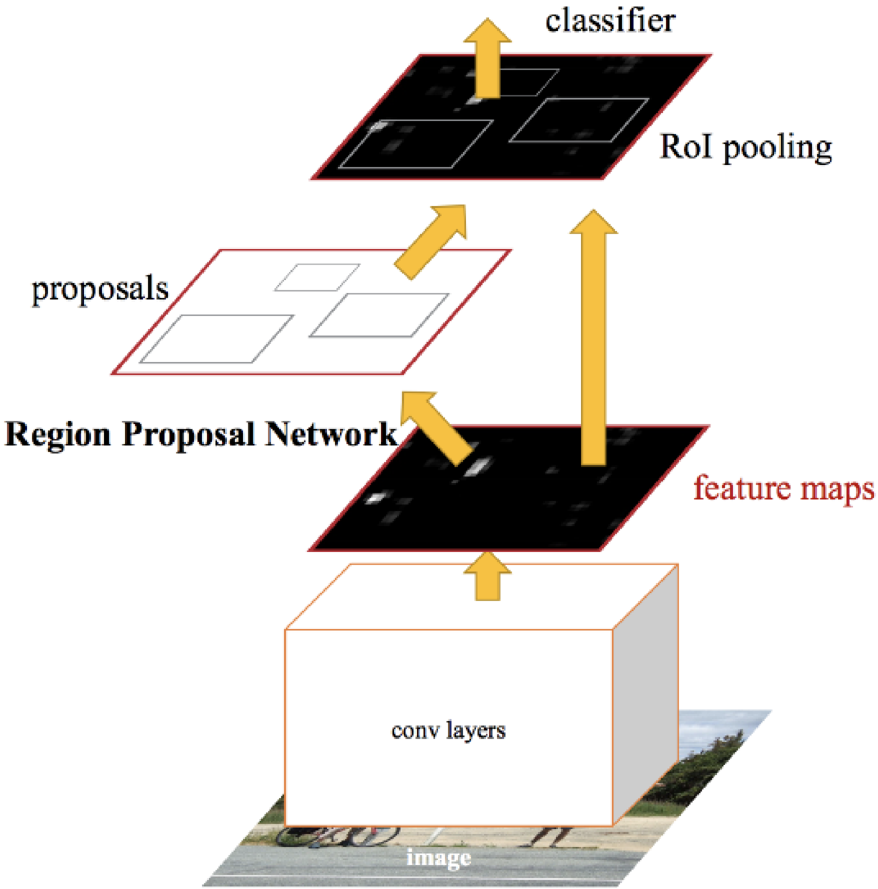


Features were extracted by Convolutional Autoencoder. Cells were visualized by tSNE using the bottleneck layer of Convolutional Autoencoder.


labelImg /home/nikolay/WABI/Misc/HPA/keras-frcnn/Nucleosome_Cytosol_train/0a4ec902-bbb8-11e8-b2ba-ac1f6b6435d0.png
(100%)
14:21
Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
PascalVOC
Create\nRectBox
Duplicate\nRectBox
Delete\nRectBox
Zoom In
202 %
Zoom Out
Fit Window
Fit Width
Box Labels
Edit Label
difficult
Use default label
Nuc
NucPlusCyt
Nuc
Nuc
NucPlusCyt
Nuc
Nuc
File List
/home/nikolay/WABI/Misc/HPA/keras-frcnn/Nuc
labelImg
NucPlusCyt
Cancel
OK
dog
person
cat
tv
car
meatballs
marinara sauce
tomato soup
chicken noodle soup
french onion soup
chicken breast
ribs
pulled pork
X: 504; Y: 423

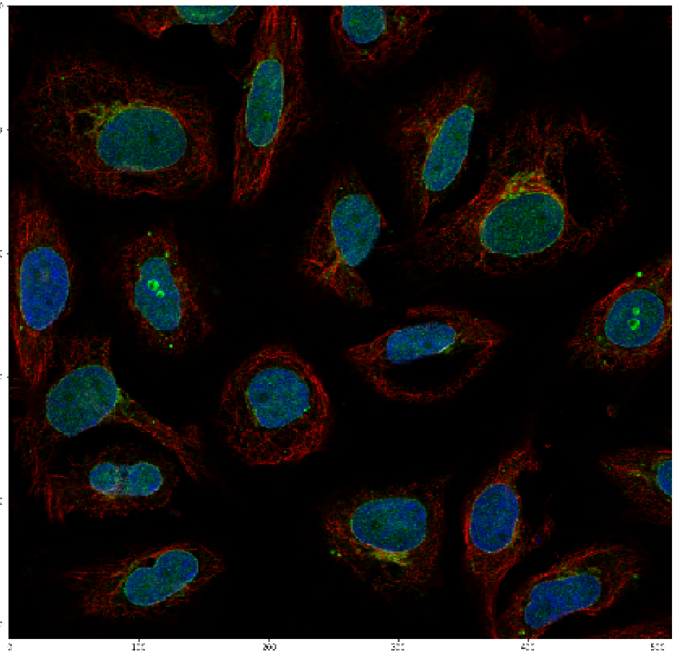
Object detection with RCNN models

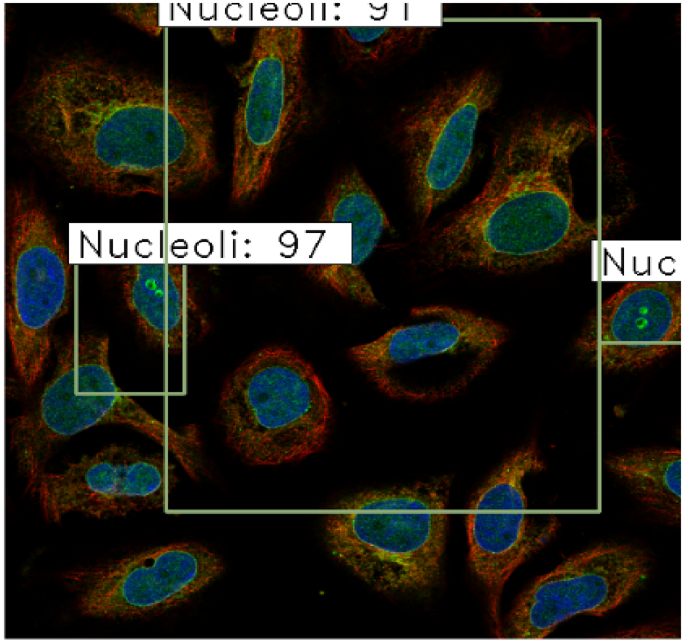


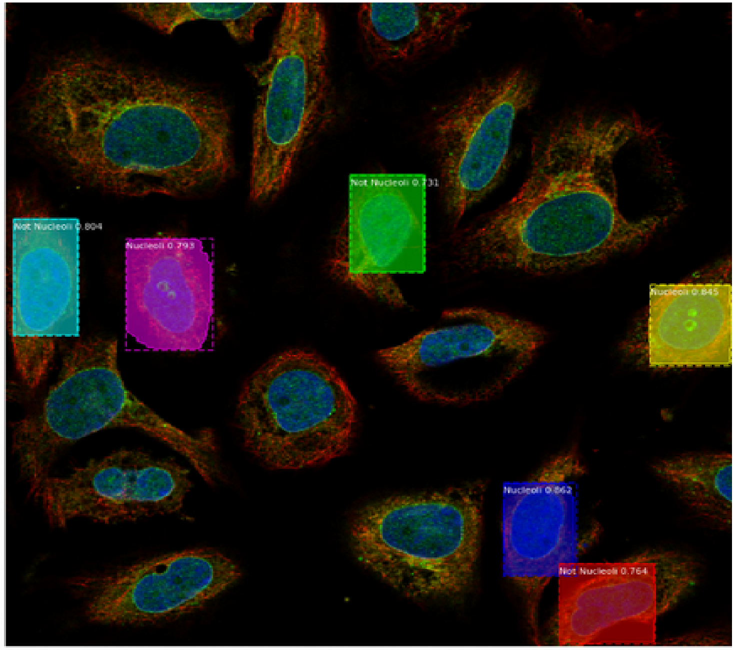


FastRCNN and FasterRCNN models

Faster RCNN vs. Mask-RCNN for Nucleoli object detection

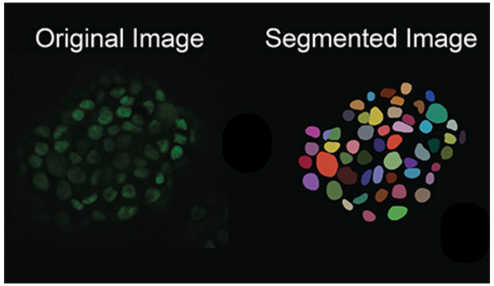


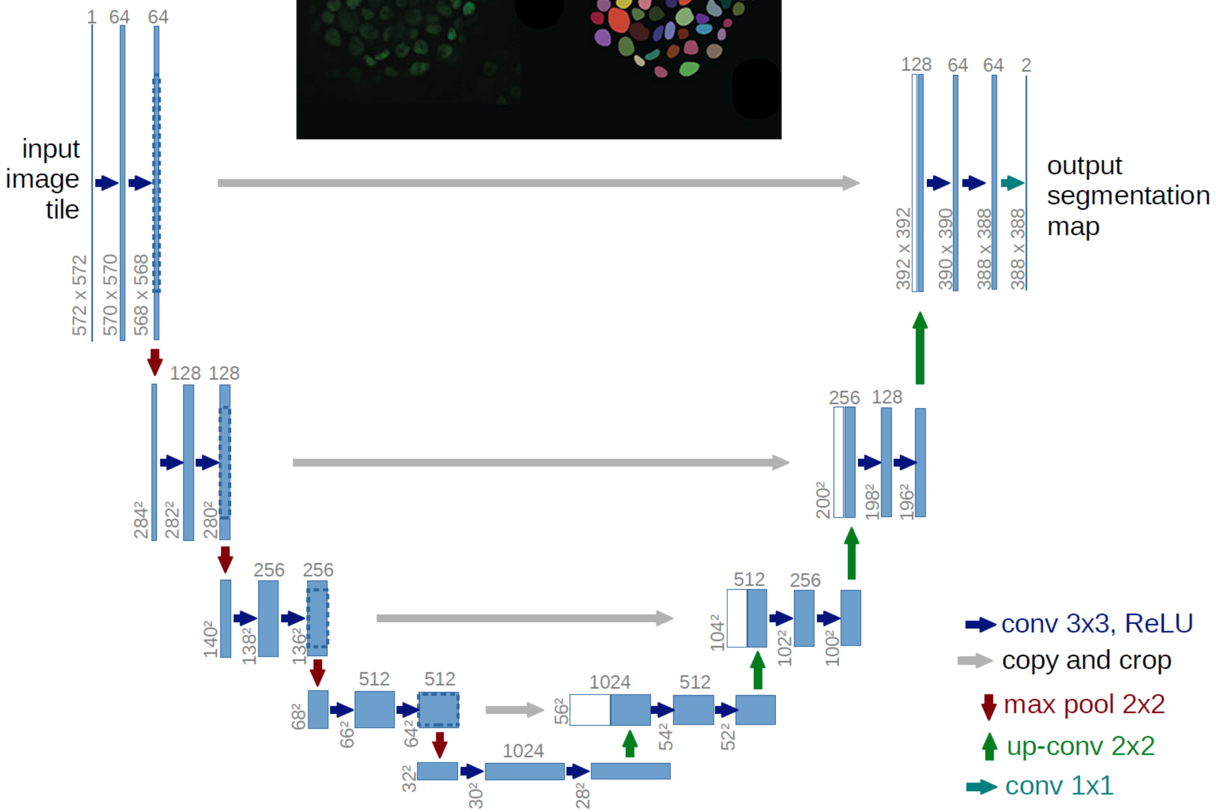


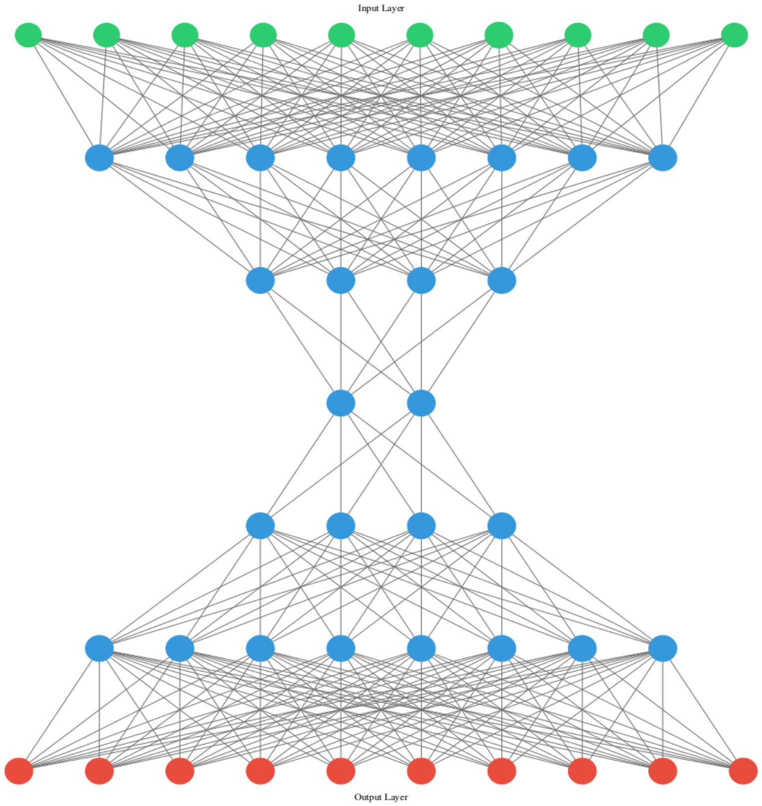


Faster-RCNN had lower accuracy of cell detection and was much slower than Mask-RCNN

Instance segmentation with UNET







StarDist and CellPose segmentation algorithms

Cell Detection with Star-convex Polygons

Uwe Schmidt[1,*], Martin Weigert[1,*], Coleman Broaddus[1], and Gene Myers[1,2]

[1] Max Planck Institute of Molecular Cell Biology and Genetics, Dresden, Germany
Center for Systems Biology Dresden, Germany
[2] Faculty of Computer Science, Technical University Dresden, Germany

Abstract. Automatic detection and segmentation of cells and nuclei in microscopy images is important for many biological applications. Recent successful learning-based approaches include per-pixel cell segmentation with subsequent pixel grouping, or localization of bounding boxes with subsequent shape refinement. In situations of crowded cells, these can be prone to segmentation errors, such as falsely merging bordering cells or suppressing valid cell instances due to the poor approximation with bounding boxes. To overcome these issues, we propose to localize cell nuclei via *star-convex polygons*, which are a much better shape representation as compared to bounding boxes and thus do not need shape refinement. To that end, we train a convolutional neural network that predicts for every pixel a polygon for the cell instance at that position. We demonstrate the merits of our approach on two synthetic datasets and one challenging dataset of diverse fluorescence microscopy images.

Cellpose: a generalist algorithm for cellular segmentation

Carsen Stringer[1*], Michalis Michaelos[1], Marius Pachitariu[1*]
[1] HHMI Janelia Research Campus, Ashburn, VA, USA
* correspondence to (stringerc, pachitarium) @ janelia.hhmi.org

Many biological applications require the segmentation of cell bodies, membranes and nuclei from microscopy images. Deep learning has enabled great progress on this problem, but current methods are specialized for images that have large training datasets. Here we introduce a generalist, deep learning-based segmentation algorithm called Cellpose, which can very precisely segment a wide range of image types out-of-the-box and does not require model retraining or parameter adjustments. We trained Cellpose on a new dataset of highly-varied images of cells, containing over 70,000 segmented objects. To support community contributions to the training data, we developed software for manual labelling and for curation of the automated results, with optional direct upload to our data repository. Periodically retraining the model on the community-contributed data will ensure that Cellpose improves constantly.

Introduction

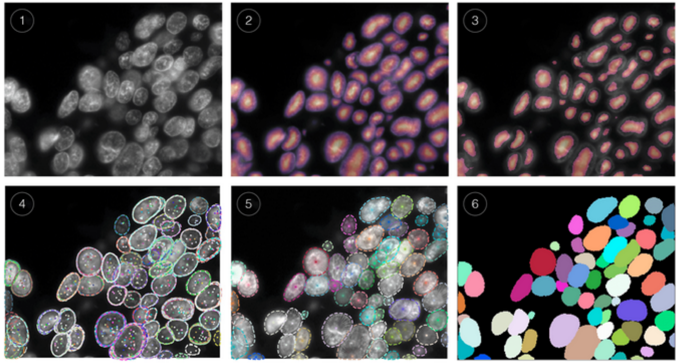
Quantitative cell biology requires simultaneous measurements of different cellular properties, such as shape, position, RNA expression and protein expression [1]. A first step towards assigning these properties to single cells is the segmentation of an imaged volume into cell bodies, usually based on a cytoplasmic or membrane marker [2–8]. This step can be straightfor-

puter vision algorithms like Mask R-CNN [15, 16] and adapted the algorithms for the biological problem. Following the competition, this dataset generated further progress, with other methods like Stardist and nucleAlzer being developed specifically for this dataset [17, 18].

In this work, we followed the approach of the Data Science Bowl team to collect and segment a large dataset of cell images from a variety of microscopy

Usage

We provide example workflows for 2D and 3D via Jupyter notebooks that illustrate how this package can be used.



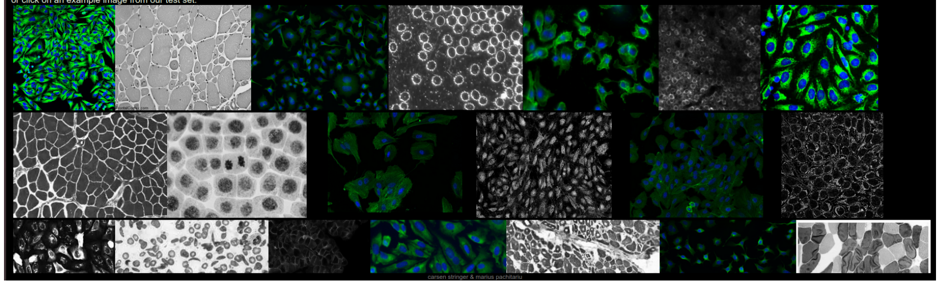
Pretrained Models for 2D

cellpose
a generalist algorithm for cellular segmentation

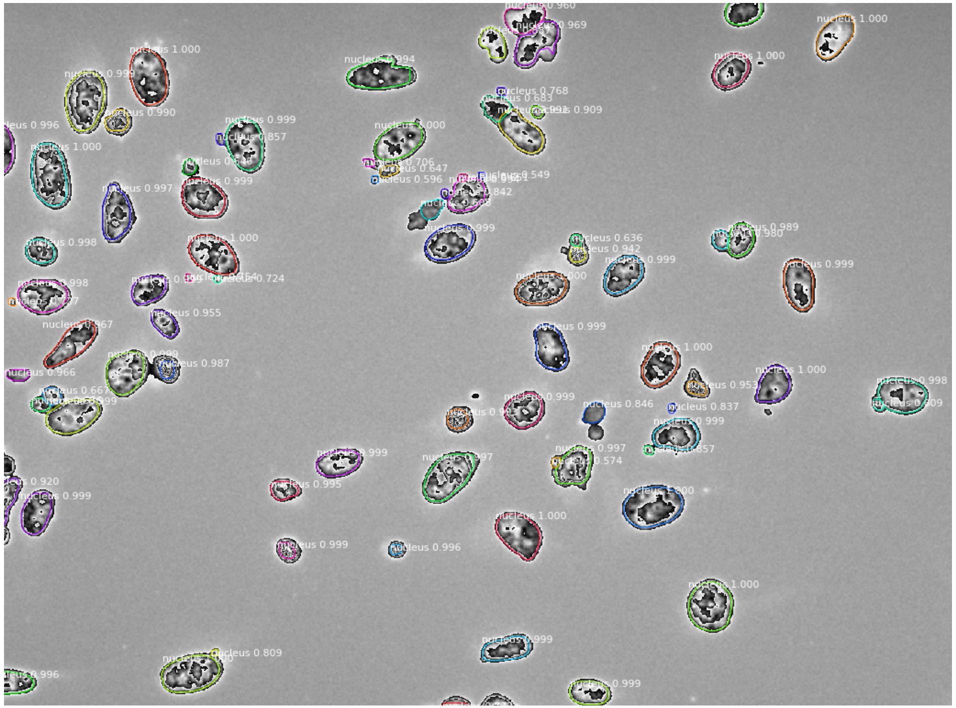
Check out full documentation here.
Try cellpose by uploading one PNG or JPG <10 MB. Images are resized to a max size of **512x512** pixels.

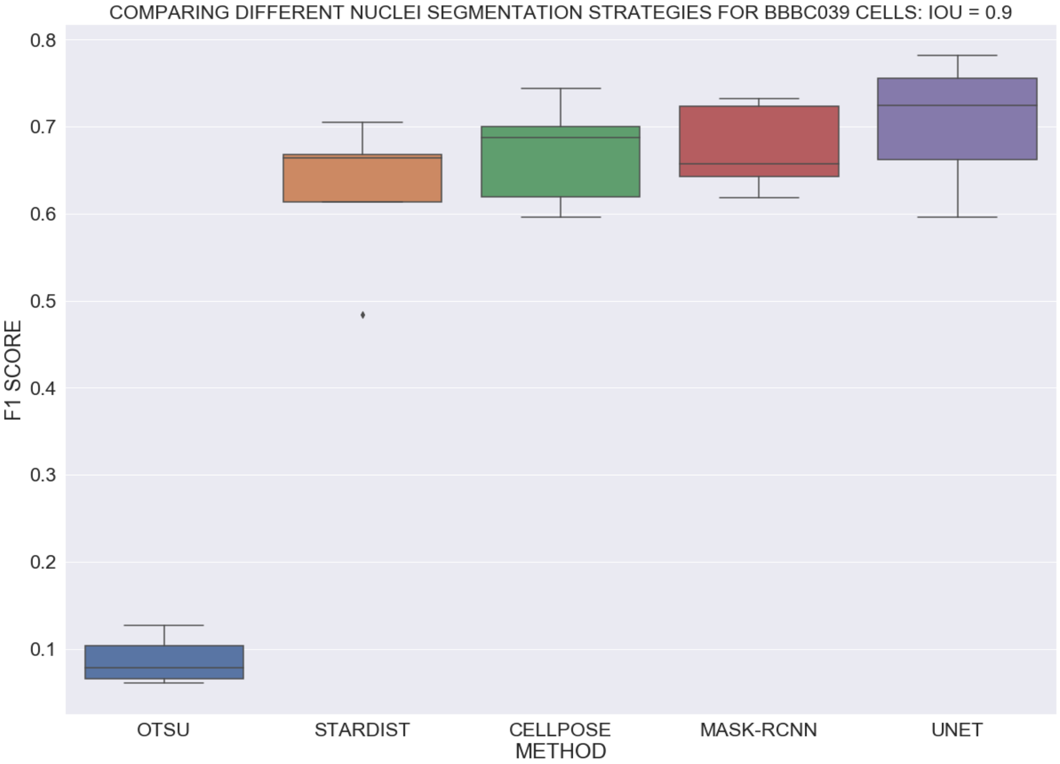
Drop files here or click to upload.

or click on an example image from our test set:



Mask-RCNN for nuclei segmentation

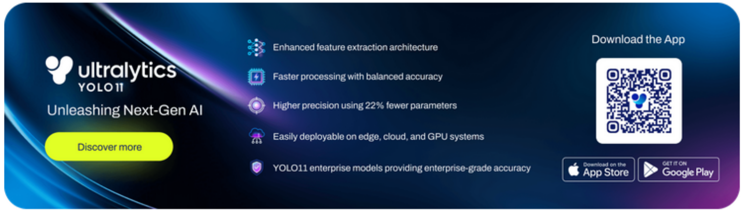




Strategy: use UNET / Mask-RCNN for segmenting and cropping cells, and run clustering

General purpose YOLO and SAM models

README | Code of conduct | AGPL-3.0 license | Security



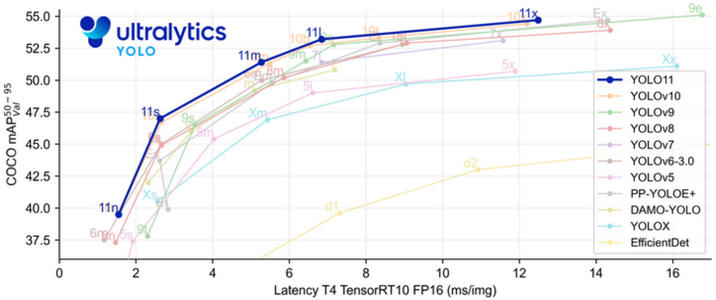
中文 | 한국어 | 日本語 | Русский | Deutsch | Français | Español | Português | Türkçe | Tiếng Việt | العربية

Ultralytics CI passing | downloads 97M | DOI 10.5281/zenodo.7347926 | Discord 819 online | Forums 419 users | Reddit 568
Run on Gradient | Open in Colab | Open in Kaggle | launch binder

Ultralytics creates cutting-edge, state-of-the-art (SOTA) YOLO models built on years of foundational research in computer vision and AI. Constantly updated for performance and flexibility, our models are **fast**, **accurate**, and **easy to use**. They excel at object detection, tracking, instance segmentation, image classification, and pose estimation tasks.

Find detailed documentation in the Ultralytics Docs. Get support via GitHub Issues. Join discussions on Discord, Reddit, and the Ultralytics Community Forums!

Request an Enterprise License for commercial use at Ultralytics Licensing.



Documentation

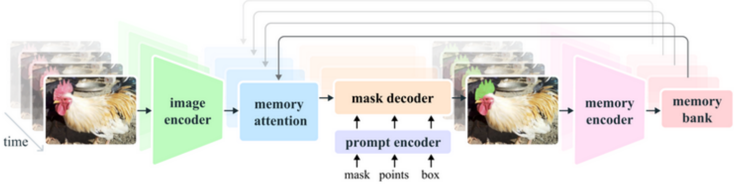
See below for quickstart installation and usage examples. For comprehensive guidance on training, validation, prediction, and deployment, refer to our full Ultralytics Docs.

README | Code of conduct | Apache-2.0 license | Security

Latest updates -- SAM 2: Segment Anything in Images and Videos

Please check out our new release on **Segment Anything Model 2 (SAM 2)**.

- SAM 2 code: https://github.com/facebookresearch/segment-anything-2
- SAM 2 demo: https://sam2.metademolab.com/
- SAM 2 paper: https://arxiv.org/abs/2408.00714



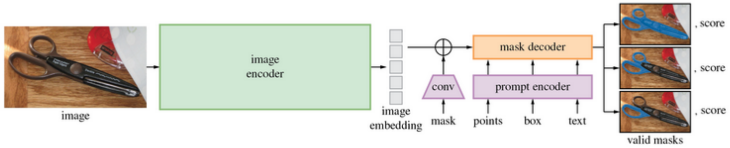
Segment Anything Model 2 (SAM 2) is a foundation model towards solving promptable visual segmentation in images and videos. We extend SAM to video by considering images as a video with a single frame. The model design is a simple transformer architecture with streaming memory for real-time video processing. We build a model-in-the-loop data engine, which improves model and data via user interaction, to collect **our SA-V dataset**, the largest video segmentation dataset to date. SAM 2 trained on our data provides strong performance across a wide range of tasks and visual domains.

Segment Anything

Meta AI Research, FAIR

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alex Berg, Wan-Yen Lo, Piotr Dollar, Ross Girshick

[`Paper`] [`Project`] [`Demo`] [`Dataset`] [`Blog`] [`BibTeX`]



The **Segment Anything Model (SAM)** produces high quality object masks from input prompts such as points or boxes, and it can be used to generate masks for all objects in an image. It has been trained on a dataset of 11 million images and 1.1 billion masks, and has strong zero-shot performance on a variety of segmentation tasks.

Session summary

Take home messages of the session:

1) CNNs are superior for cell detection and segmentation

2) Grad-CAM facilitates interpreting Deep Learning models

3) Lots of image analysis can be easily done via OpenCV

4) Mask-RCNN recommended for cell instance segmentation

5) YOLO, SAM and CellPose algorithms are becoming standard

National Bioinformatics Infrastructure Sweden (NBIS)

Knut och Alice Wallenbergs Stiftelse

Vetenskapsrådet

LUNDS UNIVERSITET