Genome 540 Discussion

Conor Camplisson

March 9th, 2023

Genome Sciences
UNIVERSITY OF WASHINGTON

Outline

- Related topics:
  - Snakemake overview
  - Example image processing pipeline

- Homework 9 Questions

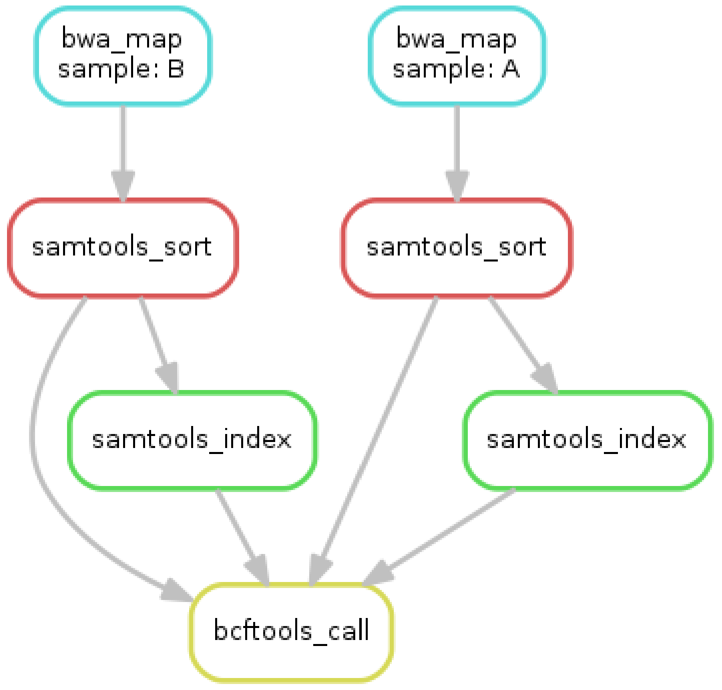
Outline

- Related topics:
  - Snakemake overview
  - Example image processing pipeline

- Homework 9 Questions

Intro to Snakemake

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"


rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"


rule samtools_index:
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"
```



https://slides.com/johanneskoester/sustainable-data-analysis-with-snakemake-non-bio

Intro to Snakemake

Why use virtual Snakemake?

Bash pipelines work. Snakemake just automates several tedious aspects:

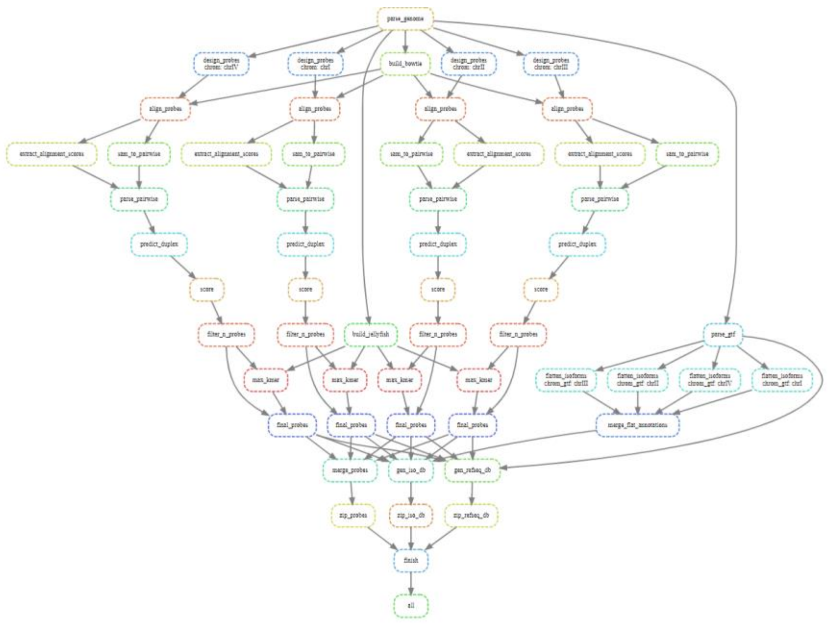
- Automatic creation of target output directories, useful flags for temp files with (optional) auto-delete

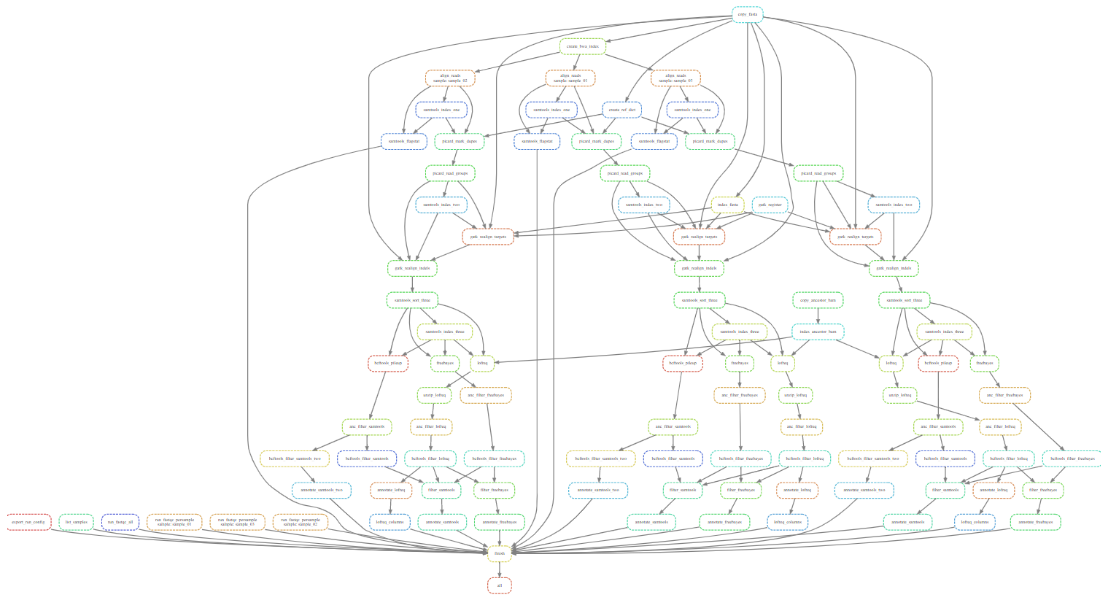
- Many utils for logging, benchmarking resource usage, reporting, etc.

- `--cluster` switch for local (1 CPU) vs. parallel (cluster nodes) execution
  - edit pipeline locally, push to cluster/cloud, run same code at scale!

- Move on to next step, per file, as soon as it's available
  - Job dependency graph more efficient than iteration (resource utilization)
  - Listen for target output file creation asynchronously, start next job

Simple pipeline in Snakemake





PaintSHOP Pipeline

Snakemake pipeline for genome-scale mining of optimal homology sequences for PaintSHOP

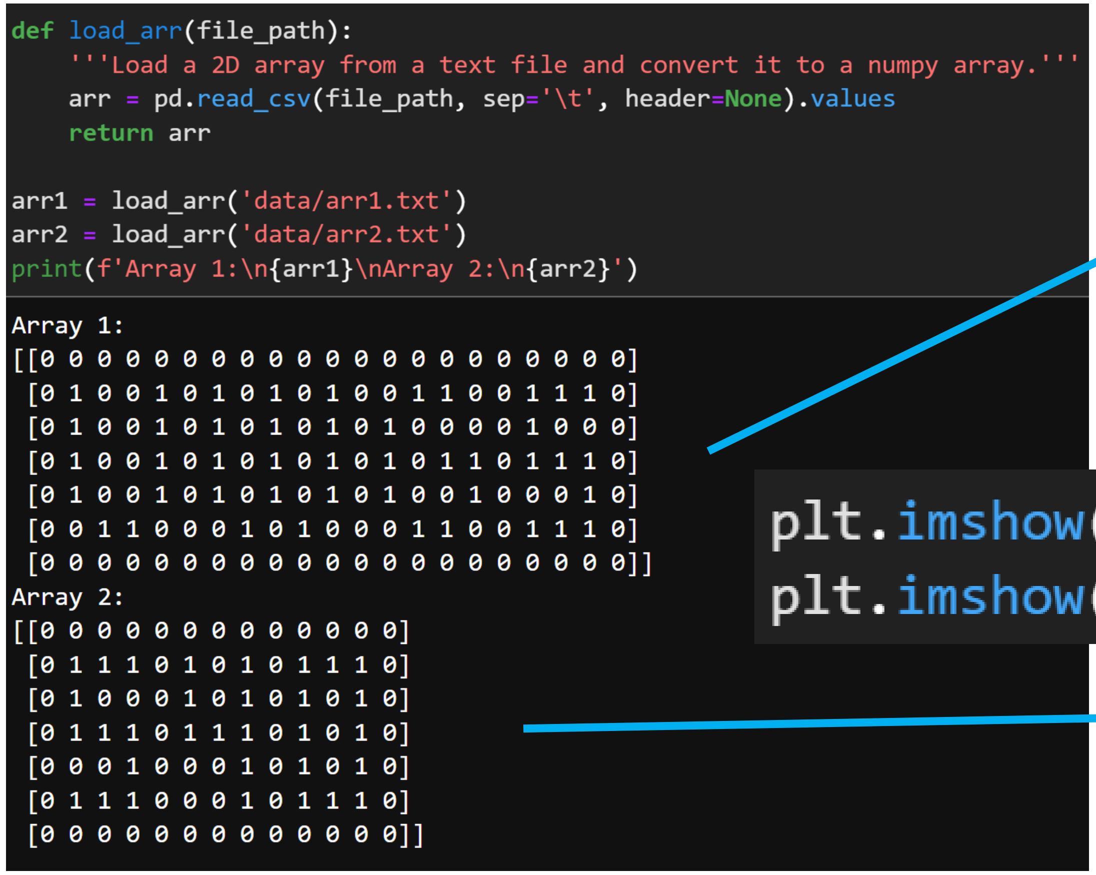
yEvo Pipeline

Variant calling Snakemake pipeline for yEvo sequencing data

GitHub CONDA snakemake

Images & Python

Load .txt files and convert to numpy

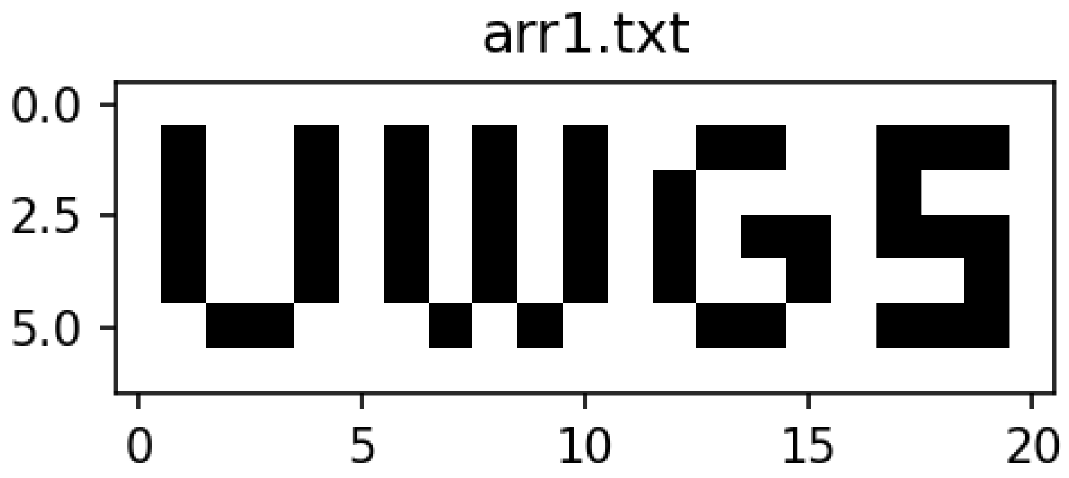


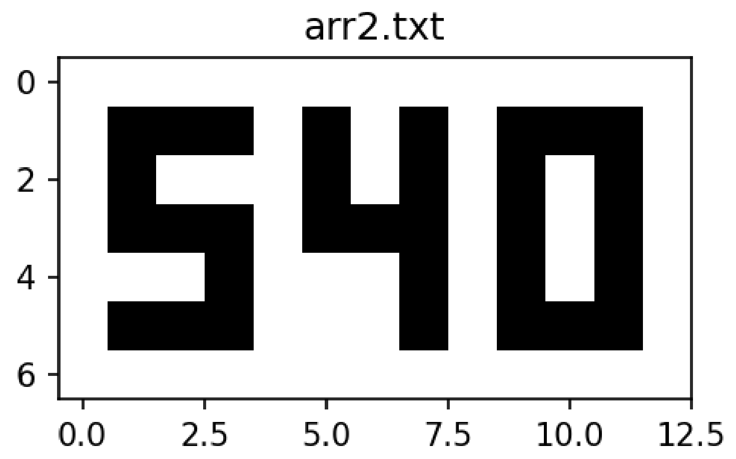
```python
def load_arr(file_path):
    '''Load a 2D array from a text file and convert it to a numpy array.'''
    arr = pd.read_csv(file_path, sep='\t', header=None).values
    return arr

arr1 = load_arr('data/arr1.txt')
arr2 = load_arr('data/arr2.txt')
print(f'Array 1:\n{arr1}\nArray 2:\n{arr2}')
```

```
Array 1:
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 1 1 0]
 [0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0]
 [0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 1 1 0]
 [0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0]
 [0 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 0 1 1 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
Array 2:
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 1 1 0 1 0 1 0 1 1 1 0]
 [0 1 0 0 0 1 0 1 0 1 0 1 0]
 [0 1 1 1 0 1 1 1 0 1 0 1 0]
 [0 0 0 1 0 0 0 1 0 1 0 1 0]
 [0 1 1 1 0 0 0 1 0 1 1 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

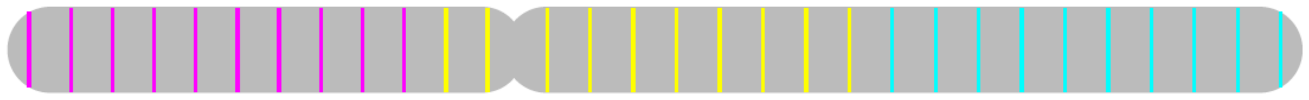
Visualize with matplotlib plt.imshow()

```python
plt.imshow(arr1)
plt.imshow(arr2)
```





Snakemake Demo Plan: Image Processing



Pattern 1: 3-color side-by-side

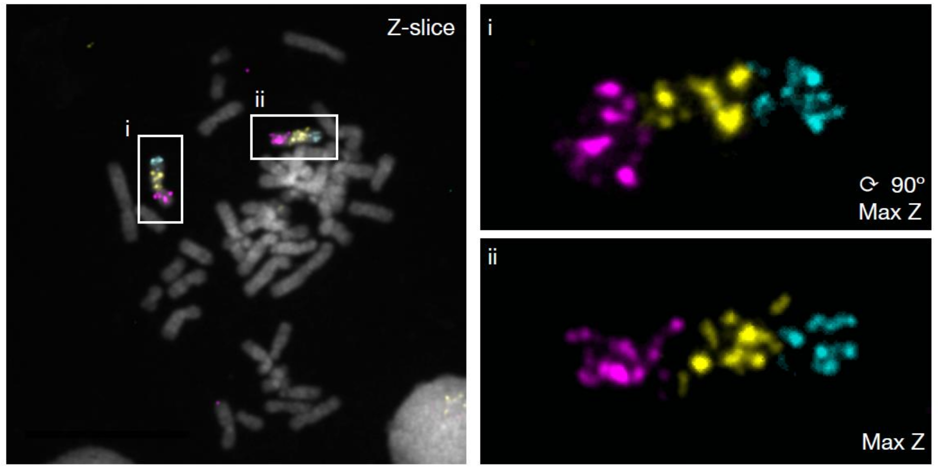


Image processing with python and Snakemake

- Multidimensional array computing with numpy
  - An image == a numpy array
  - Pre-processing, matrix operations, masking, etc.

- Ideal for parallelization
  - Many images per experiment
    - Multiple channels per image, parallelize

- Ideal use case for cluster deployment (large data)
  - Snakemake greatly facilitates

Pipeline Specification

Input: .nd2 files (3D hyperstacks)

Steps: split channels, z-project, detect fluorescent objects (puncta), compute & plot stats

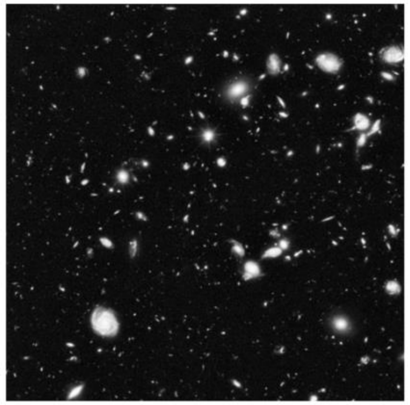
Output:

- plots of pixel intensity, spot size
- .csv file with stats per sample

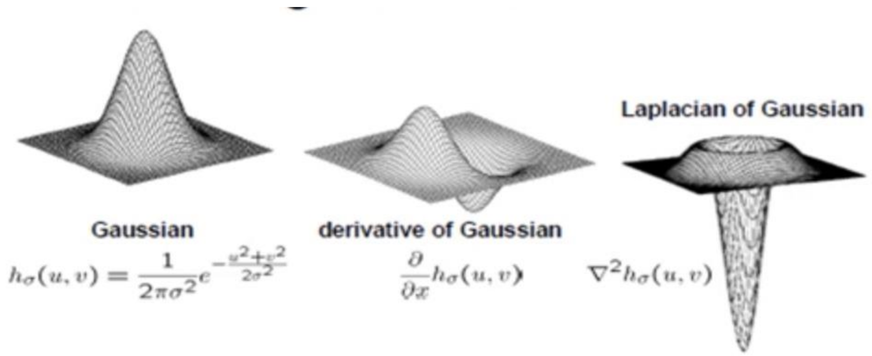
Blob detection with Gaussians

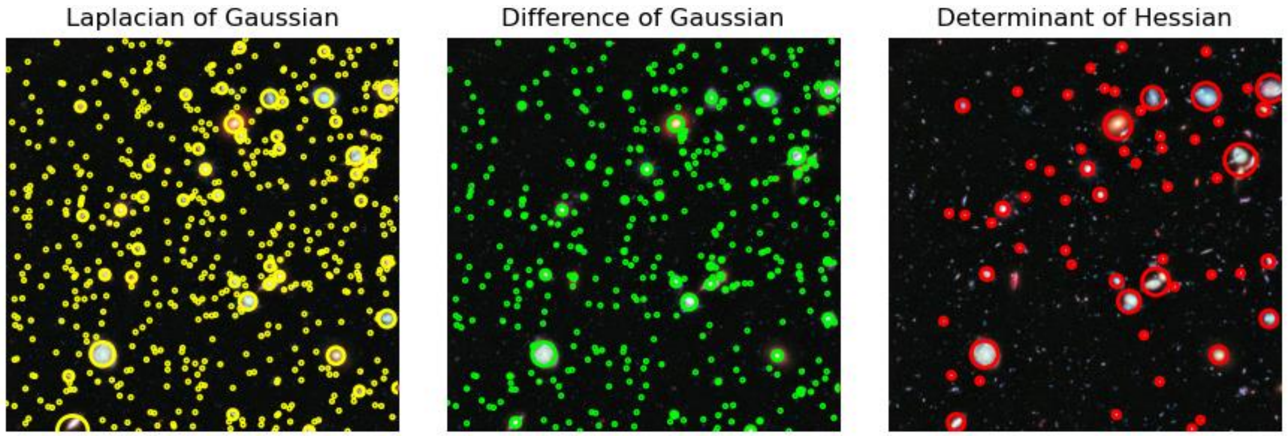
Hubble eXtreme Deep Field

To greyscale



Choosing a Gaussian Model


Gaussian
$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$
derivative of Gaussian
$\frac{\partial}{\partial x} h_\sigma(u,v)$
Laplacian of Gaussian
$\nabla^2 h_\sigma(u,v)$



Each bright dot in the image is a star or a galaxy.

Three different blob finding algorithms (all using Gaussian models) are used:

https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_blob.html

Image Segmentation

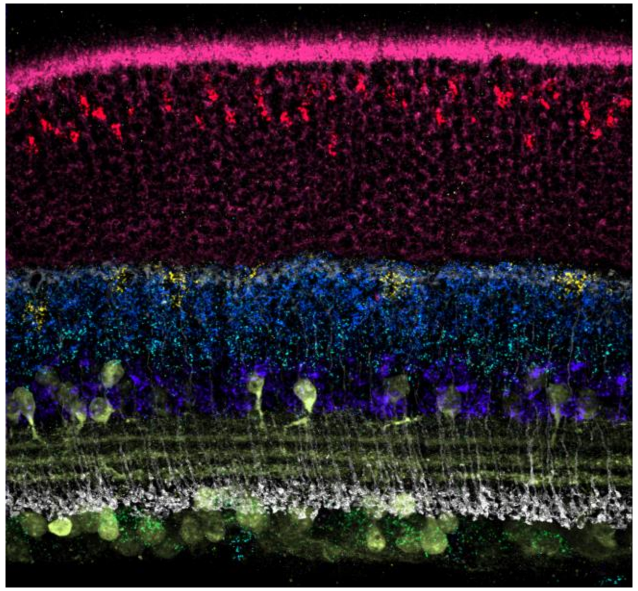
Segmentation Problems

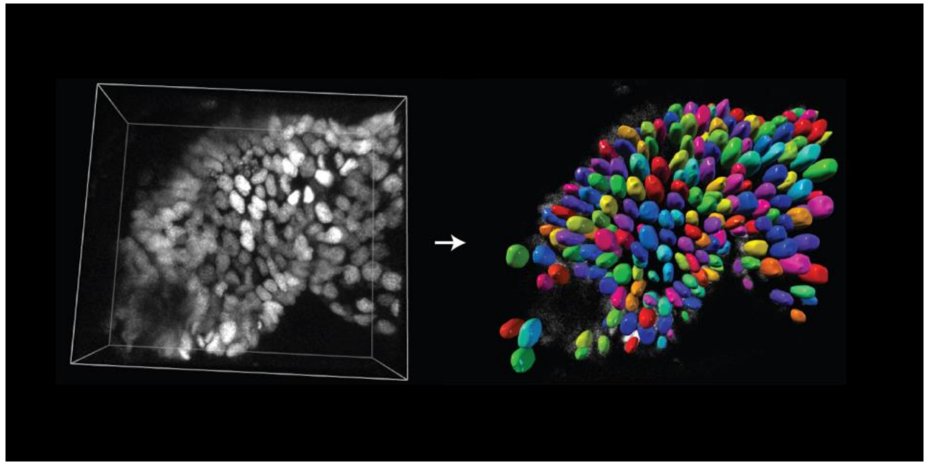
GS 540:

Segment:
- A Chromosome into elevated/non-elevated CN (HW6, HW7)
- A genome into GC-rich/AT-rich states (HW8)
- An alignment into conserved/neutral states (HW9)

Microscopy:

Answer for all pixels:
- [cell segmentation] Is this pixel in a cell?
  - Which pixels does this cell occupy?
- [nuclear segmentation] Is this pixel in the nucleus?
  - Which pixels does the nucleus occupy?





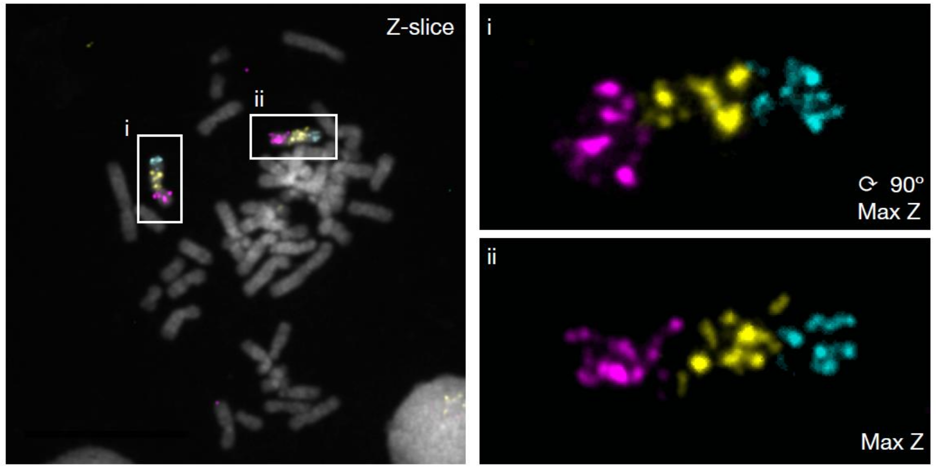
Active area of research:
- necessary to cash in on spatial bio wet lab technologies
- hard problems, diverse cell shapes, crowding, 3D
- Many recent machine learning approaches

i) Kishi, J.Y., Lapan, S.W., Beliveau, B.J. et al. *Nat Methods* **16**, 533–544 (2019)
ii) https://github.com/stardist/stardist

Some conceptual overlap

	Segmentation Problems	“Object Finding” Problems
GS 540:	• elevated/non-elevated CN (HW6, HW7) • GC-rich/AT-rich states (HW8) • conserved/neutral states (HW9)	Where are the “sites”? • Build a data structure (HW1) or train a site model (HW3) • Scan through every position in the 1D sequence and assess that position using model
Microscopy:	• Cell segmentation • Nuclear segmentation • Other applications (astronomy, computer vision, etc.)	Where are the fluorescent spots? • Use a Gaussian model • Scan through every position in the 2D image and assess that position using model

Snakemake Demo: Image Processing



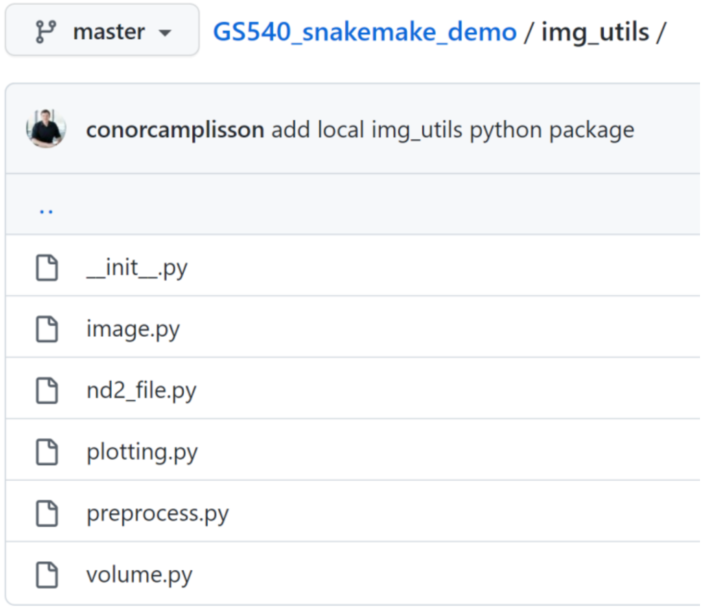
Pipeline Specification

Input: .nd2 files (3D hyperstacks)

Steps: split channels, z-project, detect fluorescent objects (puncta), compute & plot stats

Output:
- plots of pixel intensity, spot size
- .csv file with stats per sample

Added img_utils python package



master GS540_snakemake_demo / img_utils /

conorcamplisson add local img_utils python package

..
- __init__.py
- image.py
- nd2_file.py
- plotting.py
- preprocess.py
- volume.py

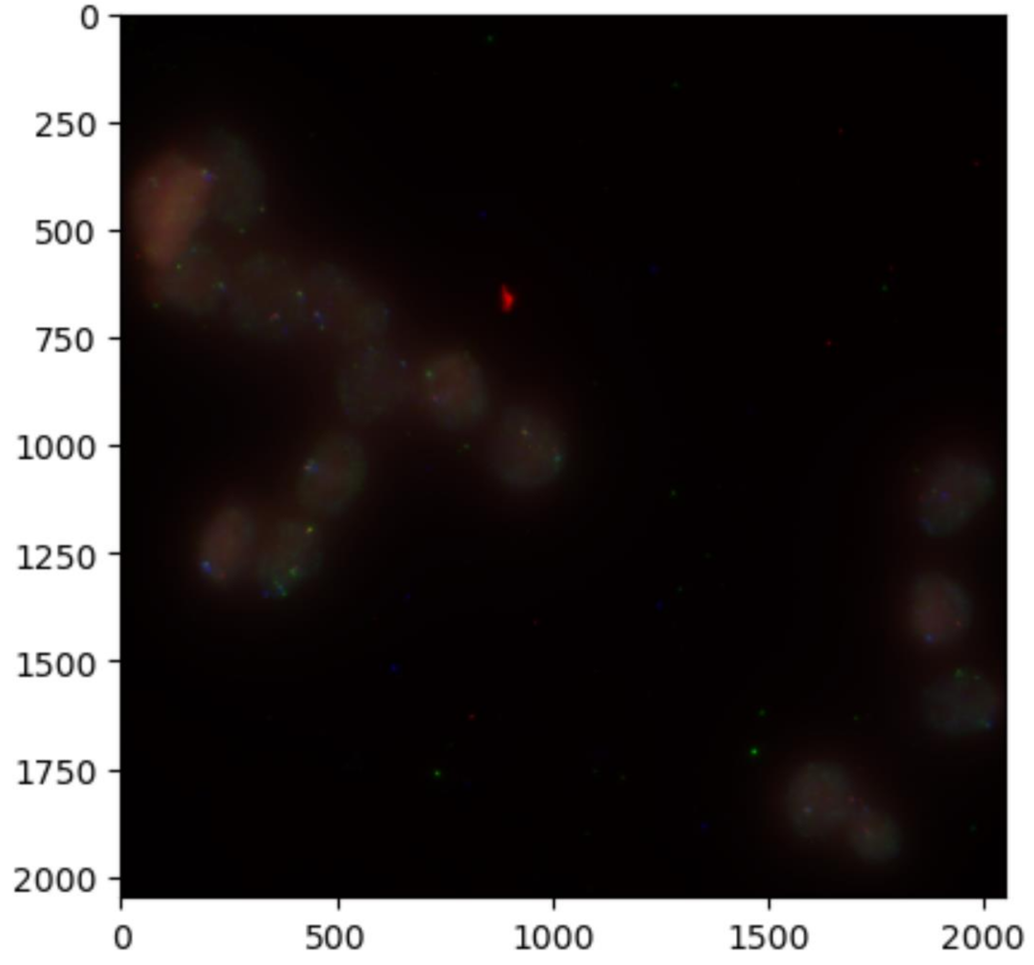
GitHub

Access the demo pipeline repo:

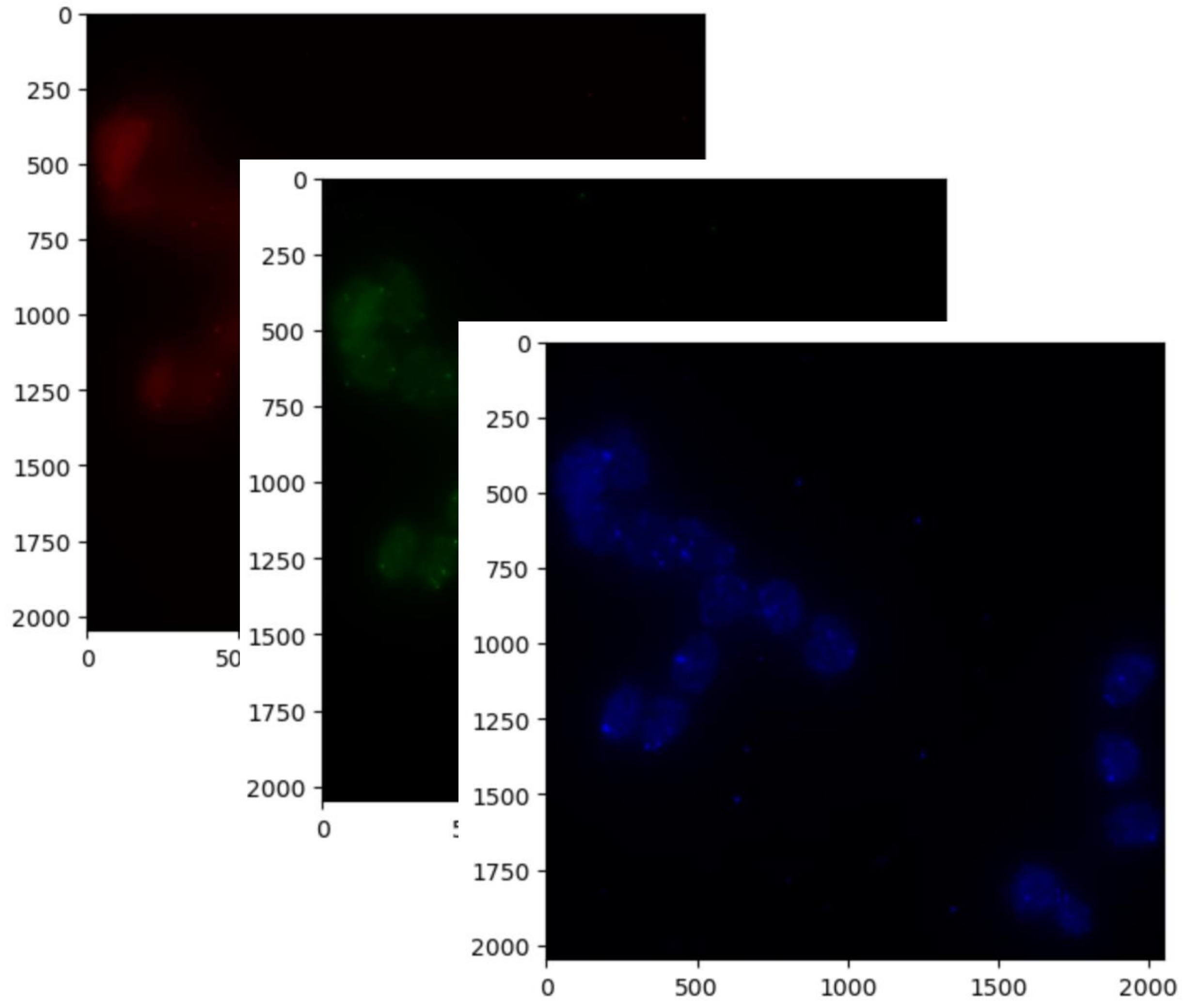
https://github.com/conorcamplisson/GS540_snakemake_demo

Step 1: load .nd2, split fluor channels

3D multi-channel hyperstack



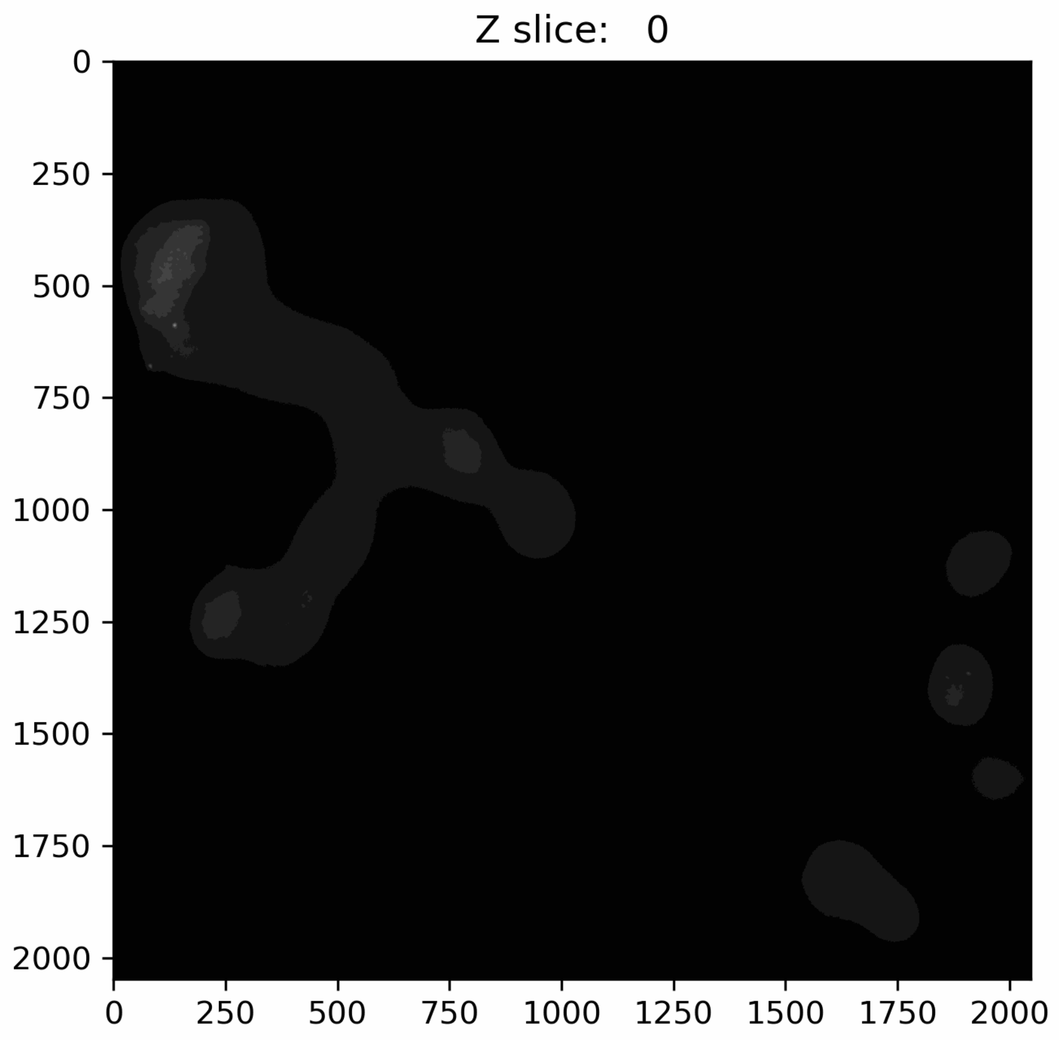
Individual channel 3D z-stacks



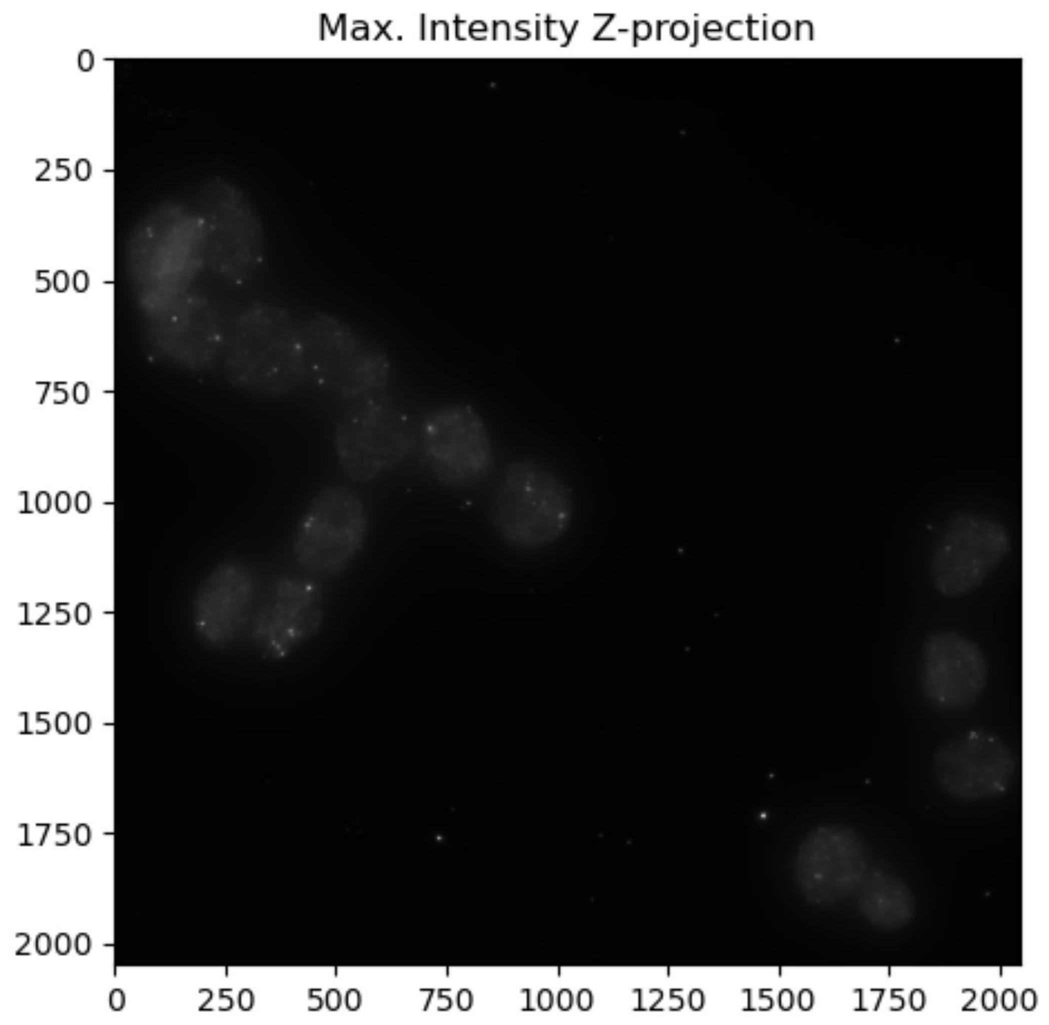
High level custom 'img_utils' api uses nd2reader under the hood

Step 2: Max. z-project each channel

Individual channel 3D z-stacks

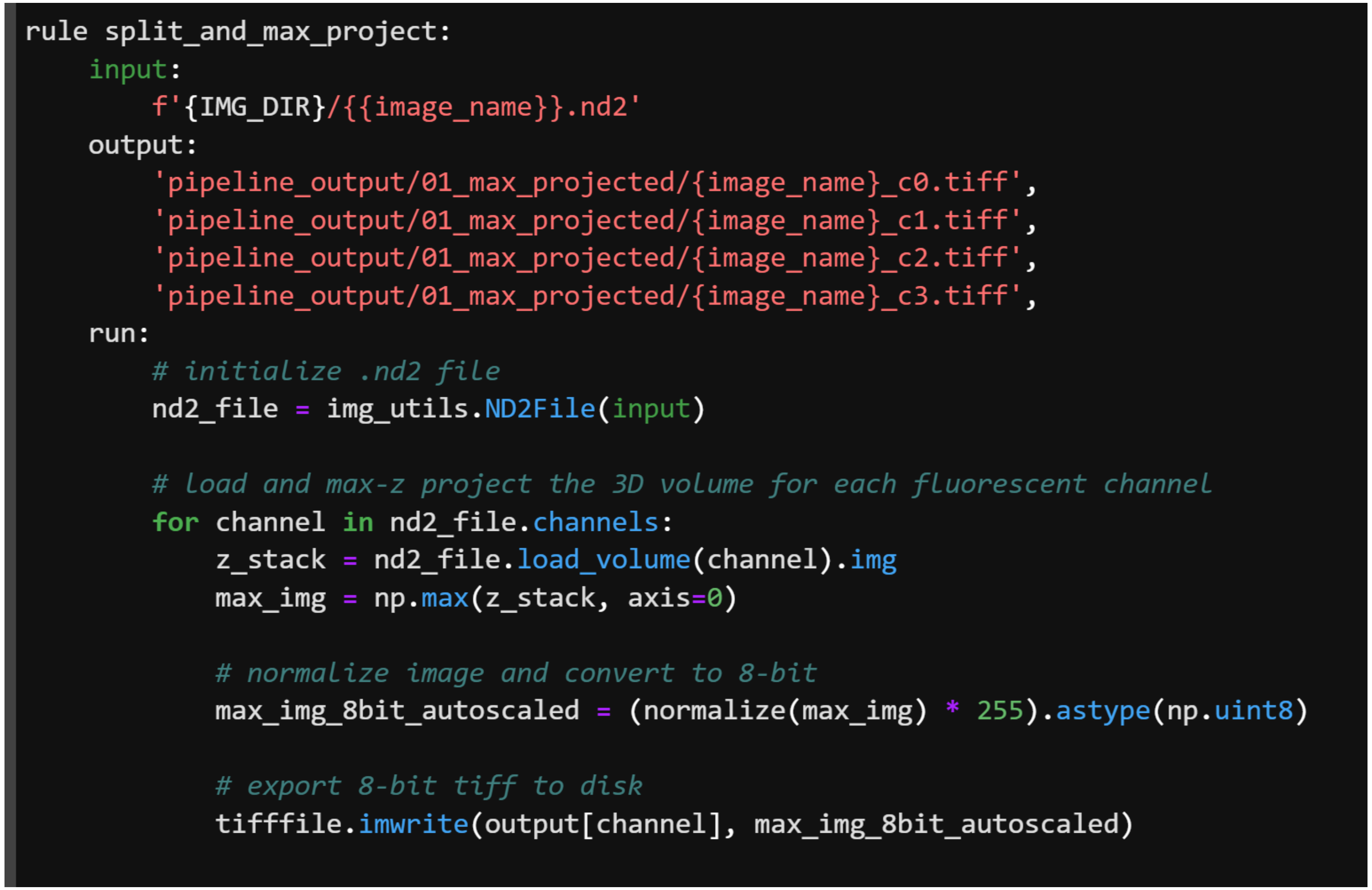


2D max z-projections



Load .nd2, split channels, max z-project each channel

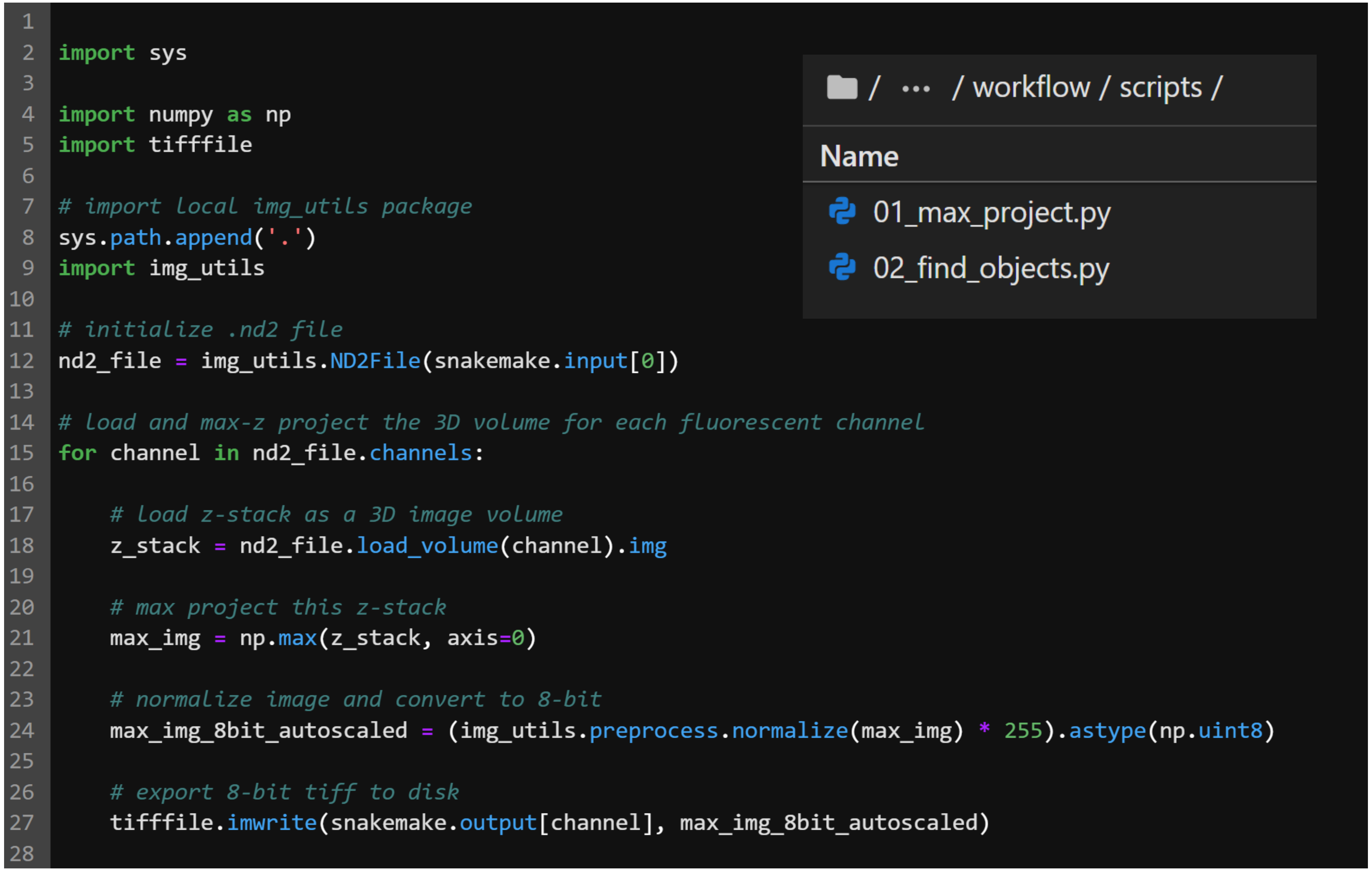
Snakemake rule:



```
rule split_and_max_project:
    input:
        f'{IMG_DIR}/{{image_name}}.nd2'
    output:
        'pipeline_output/01_max_projected/{image_name}_c0.tiff',
        'pipeline_output/01_max_projected/{image_name}_c1.tiff',
        'pipeline_output/01_max_projected/{image_name}_c2.tiff',
        'pipeline_output/01_max_projected/{image_name}_c3.tiff',
    run:
        # initialize .nd2 file
        nd2_file = img_utils.ND2File(input)

        # load and max-z project the 3D volume for each fluorescent channel
        for channel in nd2_file.channels:
            z_stack = nd2_file.load_volume(channel).img
            max_img = np.max(z_stack, axis=0)

            # normalize image and convert to 8-bit
            max_img_8bit_autoscaled = (normalize(max_img) * 255).astype(np.uint8)

            # export 8-bit tiff to disk
            tifffile.imwrite(output[channel], max_img_8bit_autoscaled)
```

Load .nd2, split channels, max z-project each channel

Python script:



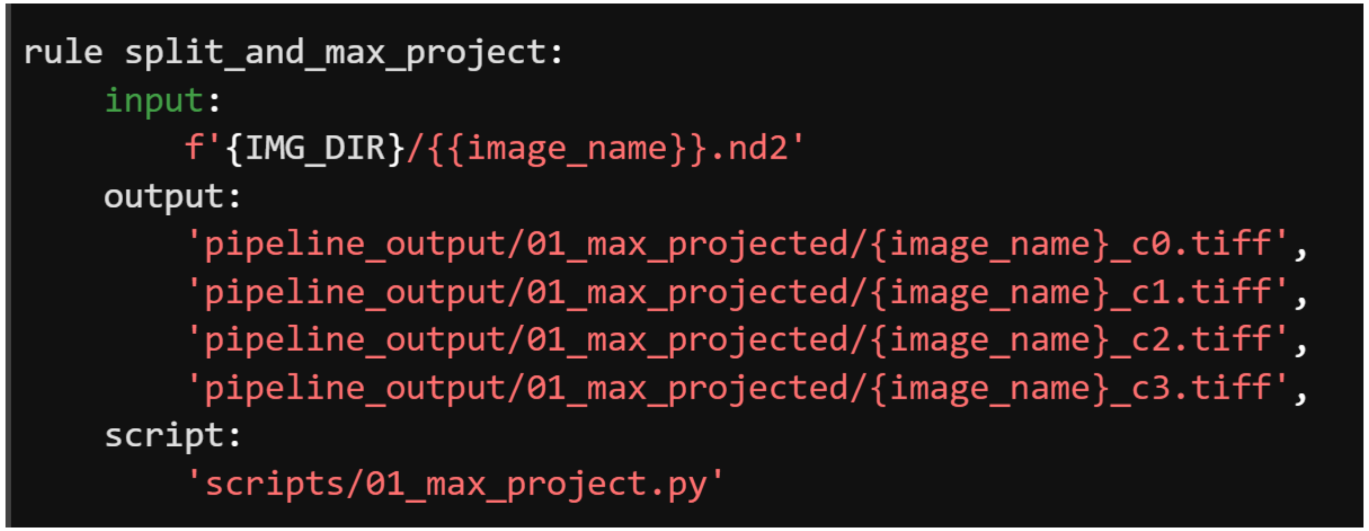
```python

import sys

import numpy as np
import tifffile

# import local img_utils package
sys.path.append('.')
import img_utils

# initialize .nd2 file
nd2_file = img_utils.ND2File(snakemake.input[0])

# load and max-z project the 3D volume for each fluorescent channel
for channel in nd2_file.channels:

    # load z-stack as a 3D image volume
    z_stack = nd2_file.load_volume(channel).img

    # max project this z-stack
    max_img = np.max(z_stack, axis=0)

    # normalize image and convert to 8-bit
    max_img_8bit_autoscaled = (img_utils.preprocess.normalize(max_img) * 255).astype(np.uint8)

    # export 8-bit tiff to disk
    tifffile.imwrite(snakemake.output[channel], max_img_8bit_autoscaled)

```

/ ··· / workflow / scripts /

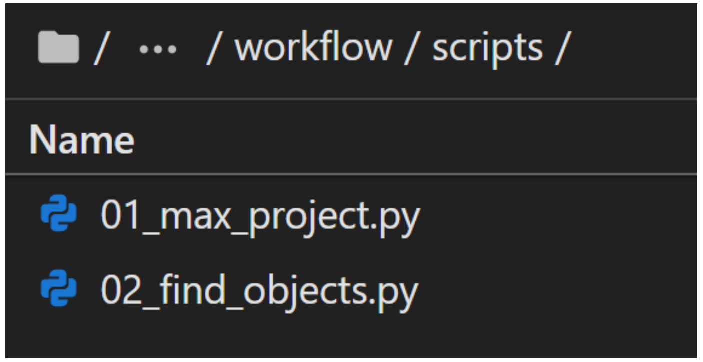
Name
01_max_project.py
02_find_objects.py

Load .nd2, split channels, max z-project each channel

Snakemake rule:



```
rule split_and_max_project:
    input:
        f'{IMG_DIR}/{{image_name}}.nd2'
    output:
        'pipeline_output/01_max_projected/{image_name}_c0.tiff',
        'pipeline_output/01_max_projected/{image_name}_c1.tiff',
        'pipeline_output/01_max_projected/{image_name}_c2.tiff',
        'pipeline_output/01_max_projected/{image_name}_c3.tiff',
    script:
        'scripts/01_max_project.py'
```

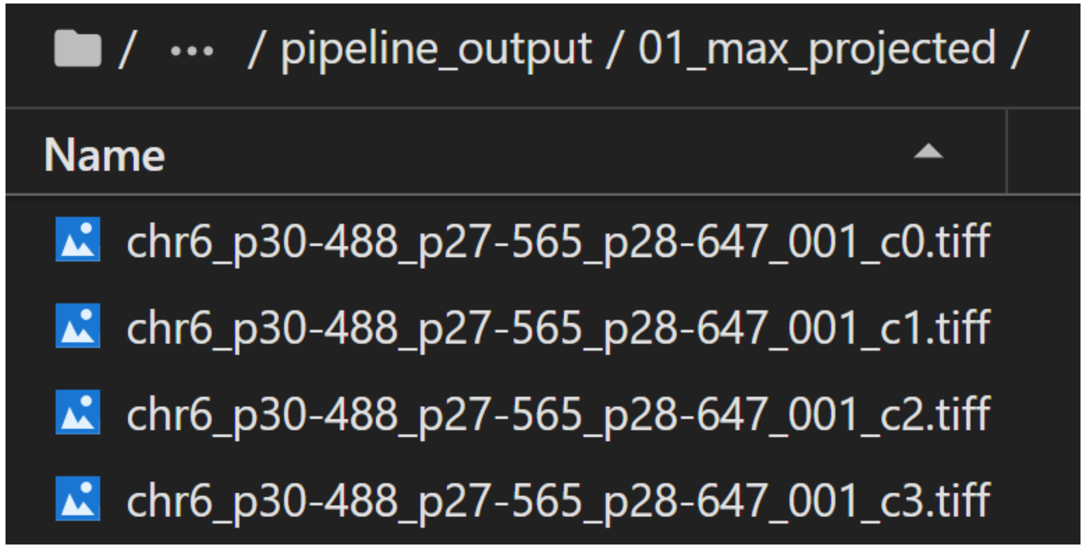


/ ··· / workflow / scripts /

Name
- 01_max_project.py
- 02_find_objects.py

→

2D max z-projections

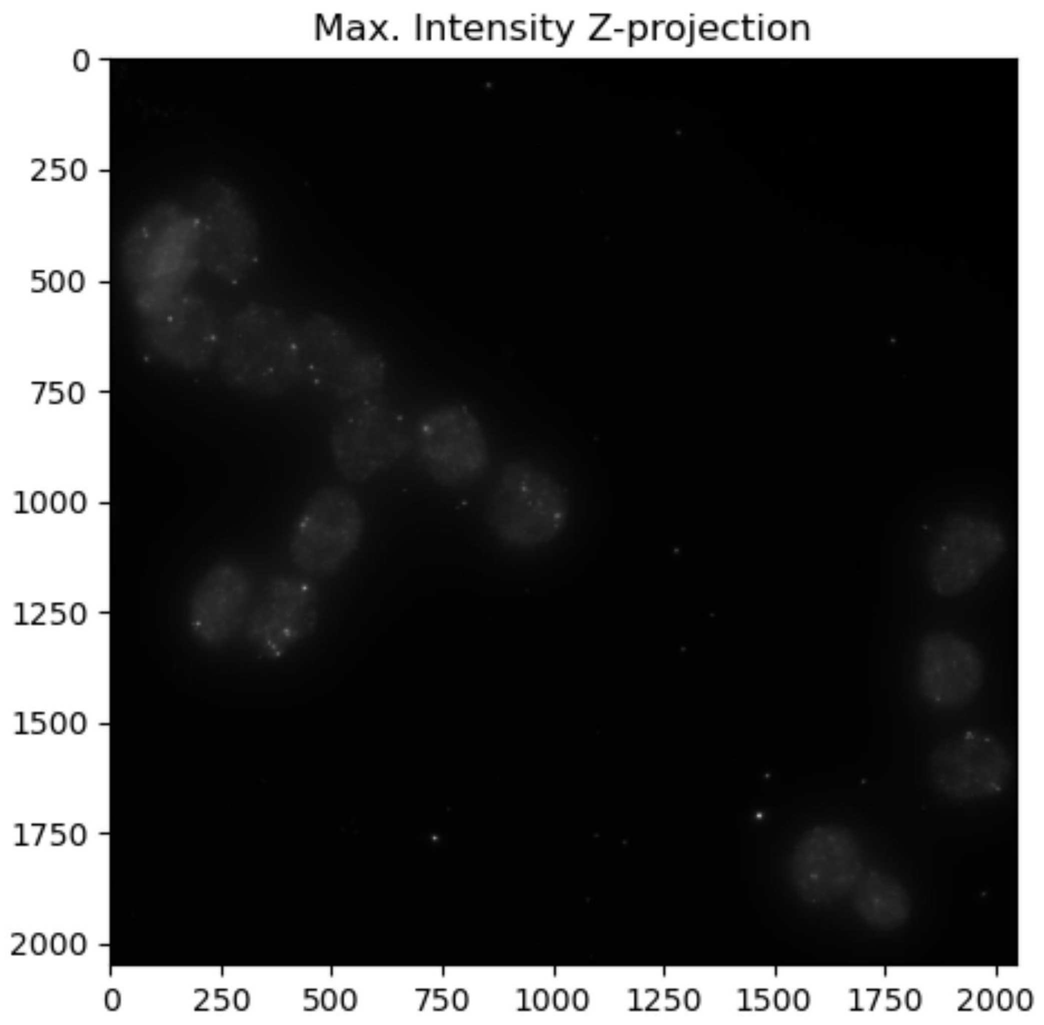


/ ··· / pipeline_output / 01_max_projected /

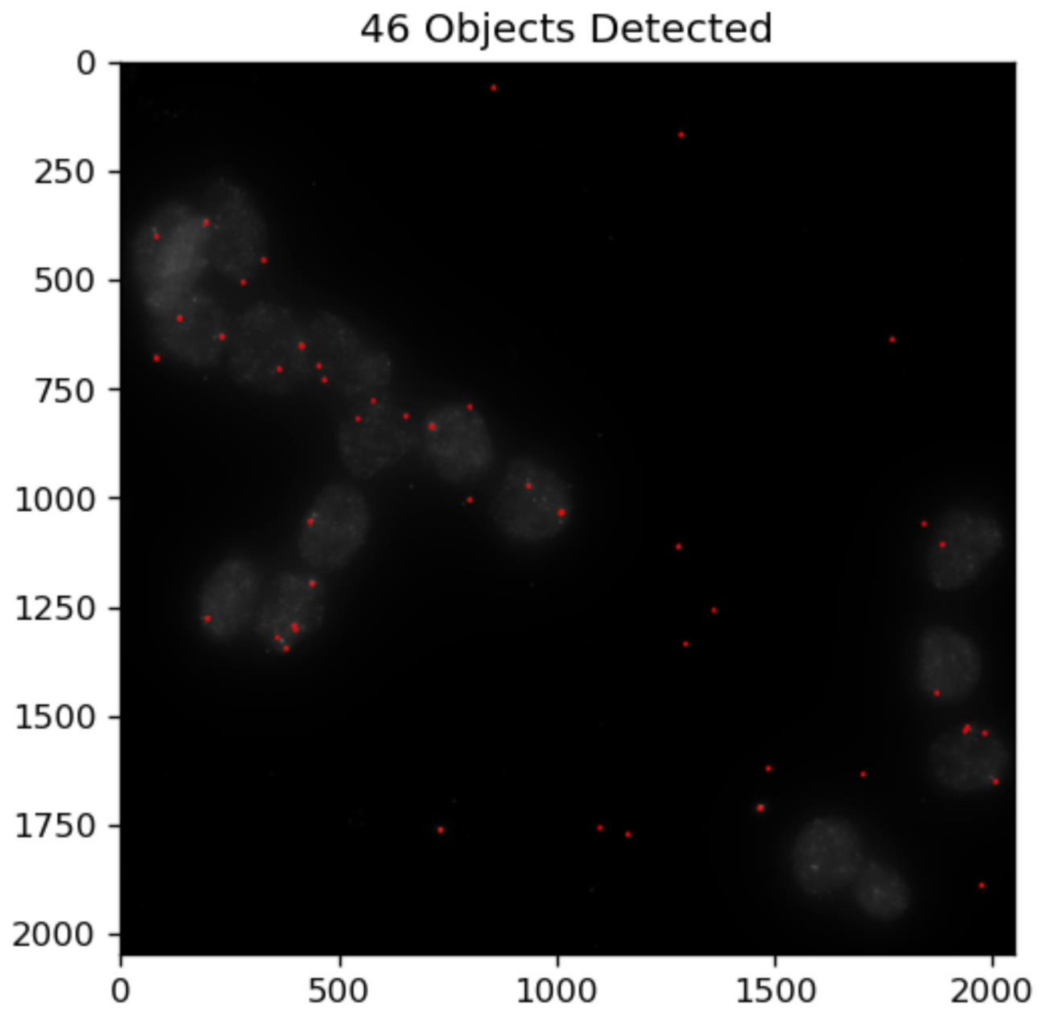
Name
- chr6_p30-488_p27-565_p28-647_001_c0.tiff
- chr6_p30-488_p27-565_p28-647_001_c1.tiff
- chr6_p30-488_p27-565_p28-647_001_c2.tiff
- chr6_p30-488_p27-565_p28-647_001_c3.tiff

Step 3: Find fluorescent objects

2D max z-projections

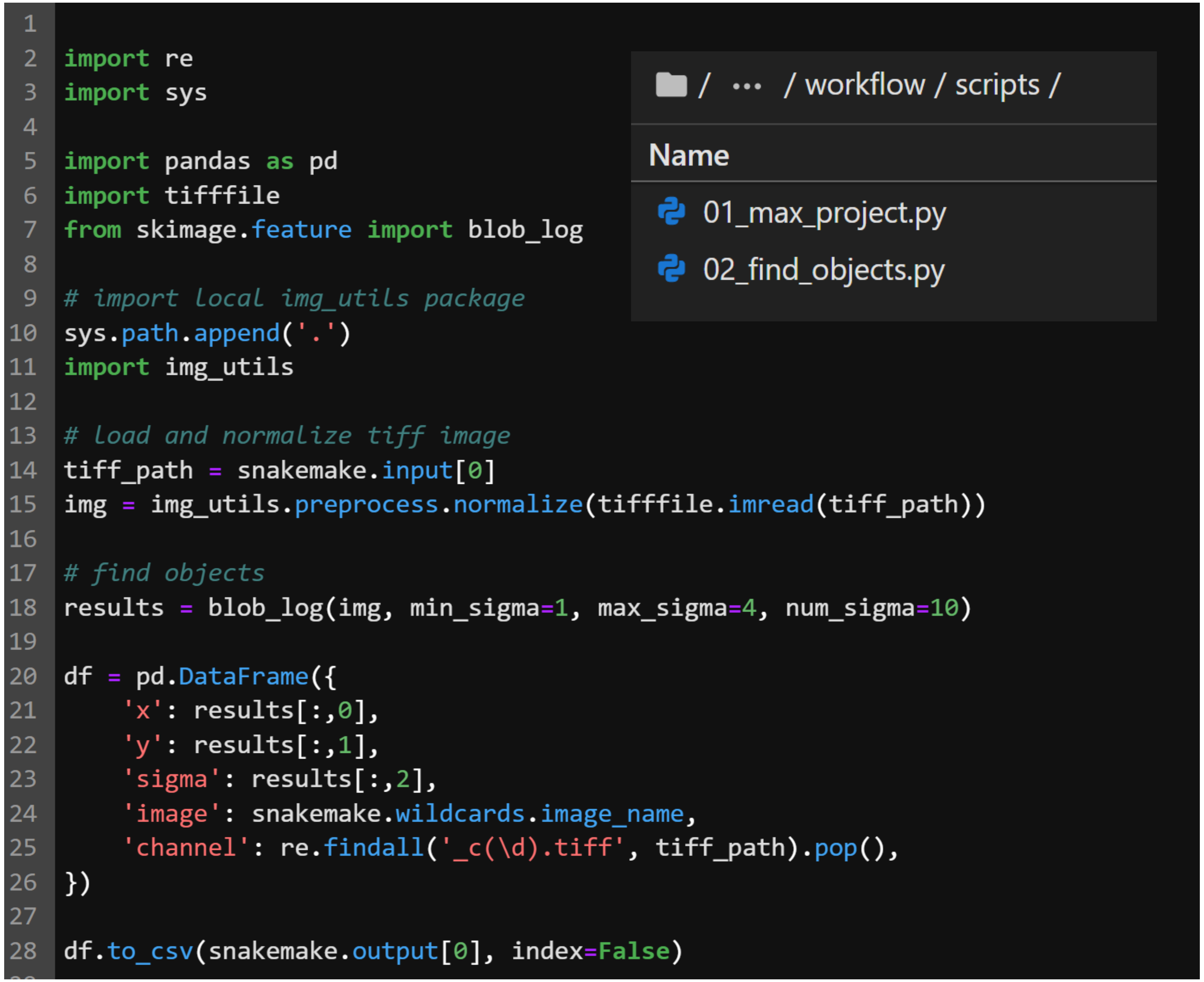


2D max z-projection



Find fluorescent objects

Python script:



```python

import re
import sys

import pandas as pd
import tifffile
from skimage.feature import blob_log

# import local img_utils package
sys.path.append('.')
import img_utils

# load and normalize tiff image
tiff_path = snakemake.input[0]
img = img_utils.preprocess.normalize(tifffile.imread(tiff_path))

# find objects
results = blob_log(img, min_sigma=1, max_sigma=4, num_sigma=10)

df = pd.DataFrame({
    'x': results[:,0],
    'y': results[:,1],
    'sigma': results[:,2],
    'image': snakemake.wildcards.image_name,
    'channel': re.findall('_c(\d).tiff', tiff_path).pop(),
})

df.to_csv(snakemake.output[0], index=False)
```

/ ••• / workflow / scripts /

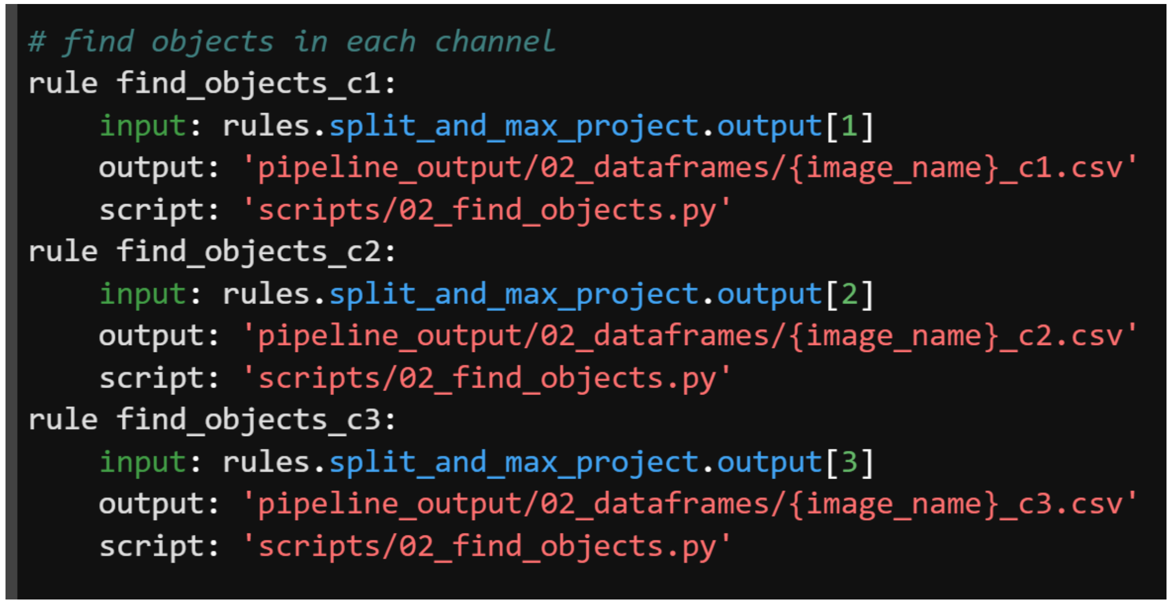
Name

01_max_project.py

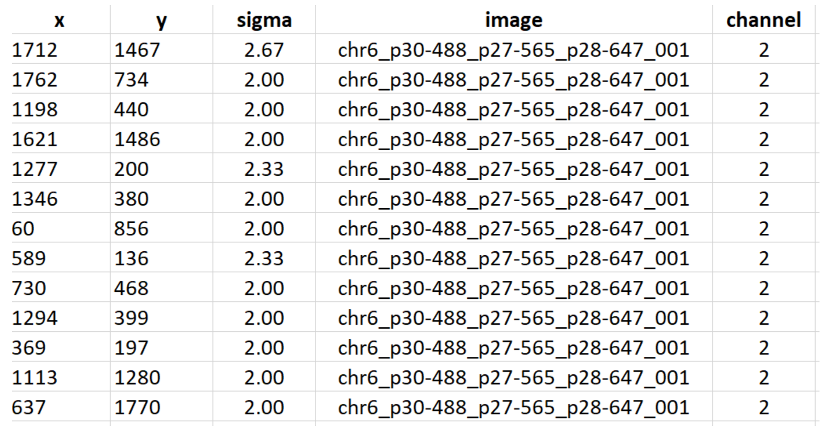
02_find_objects.py

Find fluorescent objects

Snakemake rules:

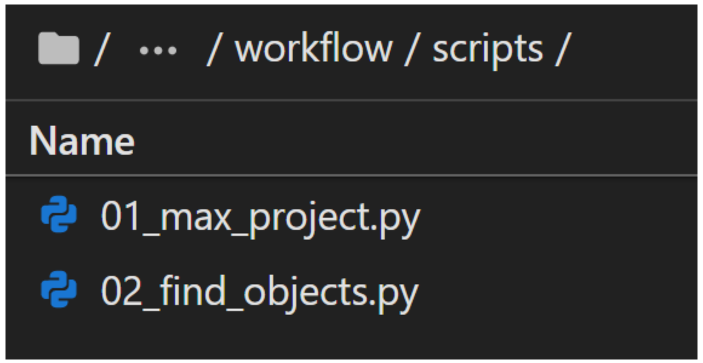


```
# find objects in each channel
rule find_objects_c1:
    input: rules.split_and_max_project.output[1]
    output: 'pipeline_output/02_dataframes/{image_name}_c1.csv'
    script: 'scripts/02_find_objects.py'
rule find_objects_c2:
    input: rules.split_and_max_project.output[2]
    output: 'pipeline_output/02_dataframes/{image_name}_c2.csv'
    script: 'scripts/02_find_objects.py'
rule find_objects_c3:
    input: rules.split_and_max_project.output[3]
    output: 'pipeline_output/02_dataframes/{image_name}_c3.csv'
    script: 'scripts/02_find_objects.py'
```



x	y	sigma	image	channel
1712	1467	2.67	chr6_p30-488_p27-565_p28-647_001	2
1762	734	2.00	chr6_p30-488_p27-565_p28-647_001	2
1198	440	2.00	chr6_p30-488_p27-565_p28-647_001	2
1621	1486	2.00	chr6_p30-488_p27-565_p28-647_001	2
1277	200	2.33	chr6_p30-488_p27-565_p28-647_001	2
1346	380	2.00	chr6_p30-488_p27-565_p28-647_001	2
60	856	2.00	chr6_p30-488_p27-565_p28-647_001	2
589	136	2.33	chr6_p30-488_p27-565_p28-647_001	2
730	468	2.00	chr6_p30-488_p27-565_p28-647_001	2
1294	399	2.00	chr6_p30-488_p27-565_p28-647_001	2
369	197	2.00	chr6_p30-488_p27-565_p28-647_001	2
1113	1280	2.00	chr6_p30-488_p27-565_p28-647_001	2
637	1770	2.00	chr6_p30-488_p27-565_p28-647_001	2

.csv dataframes

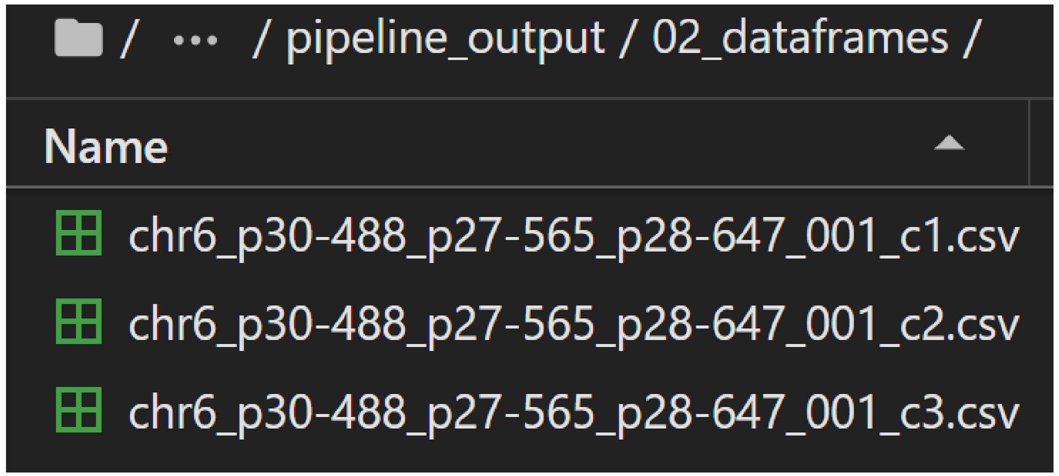


/ ••• / workflow / scripts /

Name

01_max_project.py

02_find_objects.py



/ ••• / pipeline_output / 02_dataframes /

Name

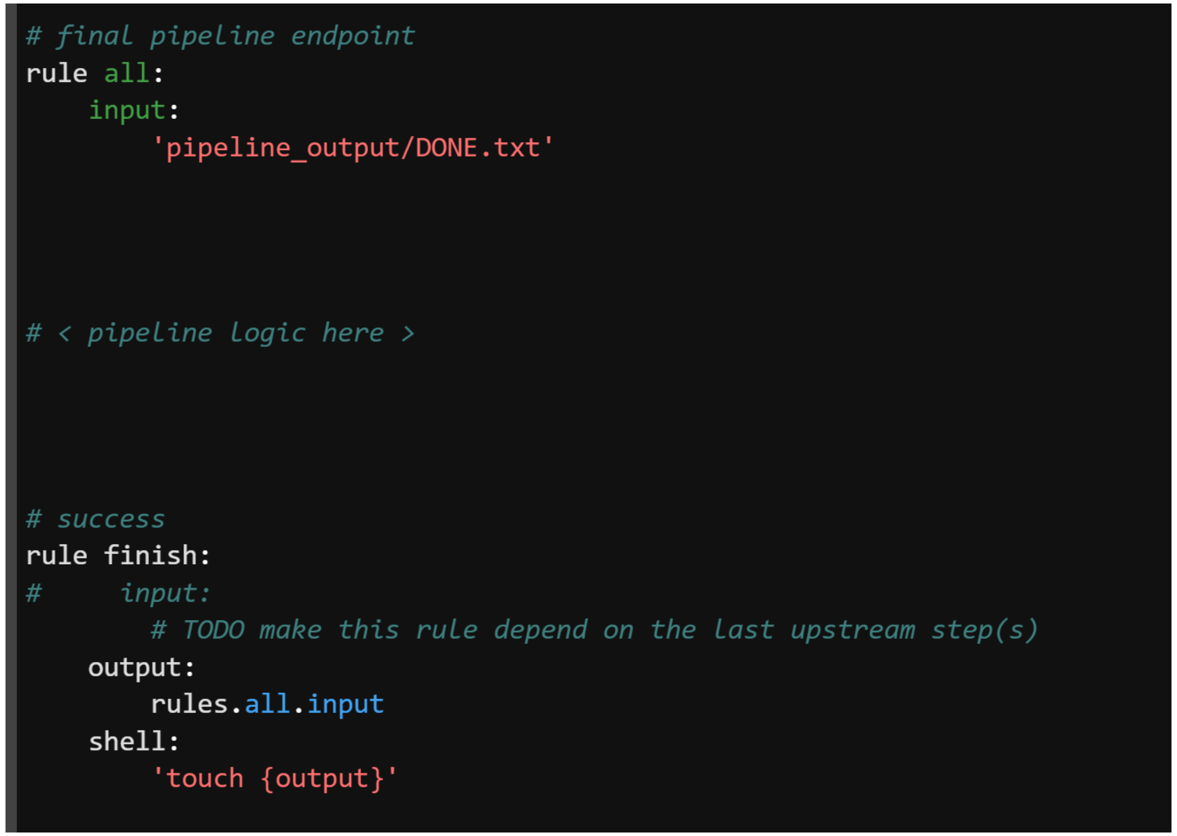
chr6_p30-488_p27-565_p28-647_001_c1.csv

chr6_p30-488_p27-565_p28-647_001_c2.csv

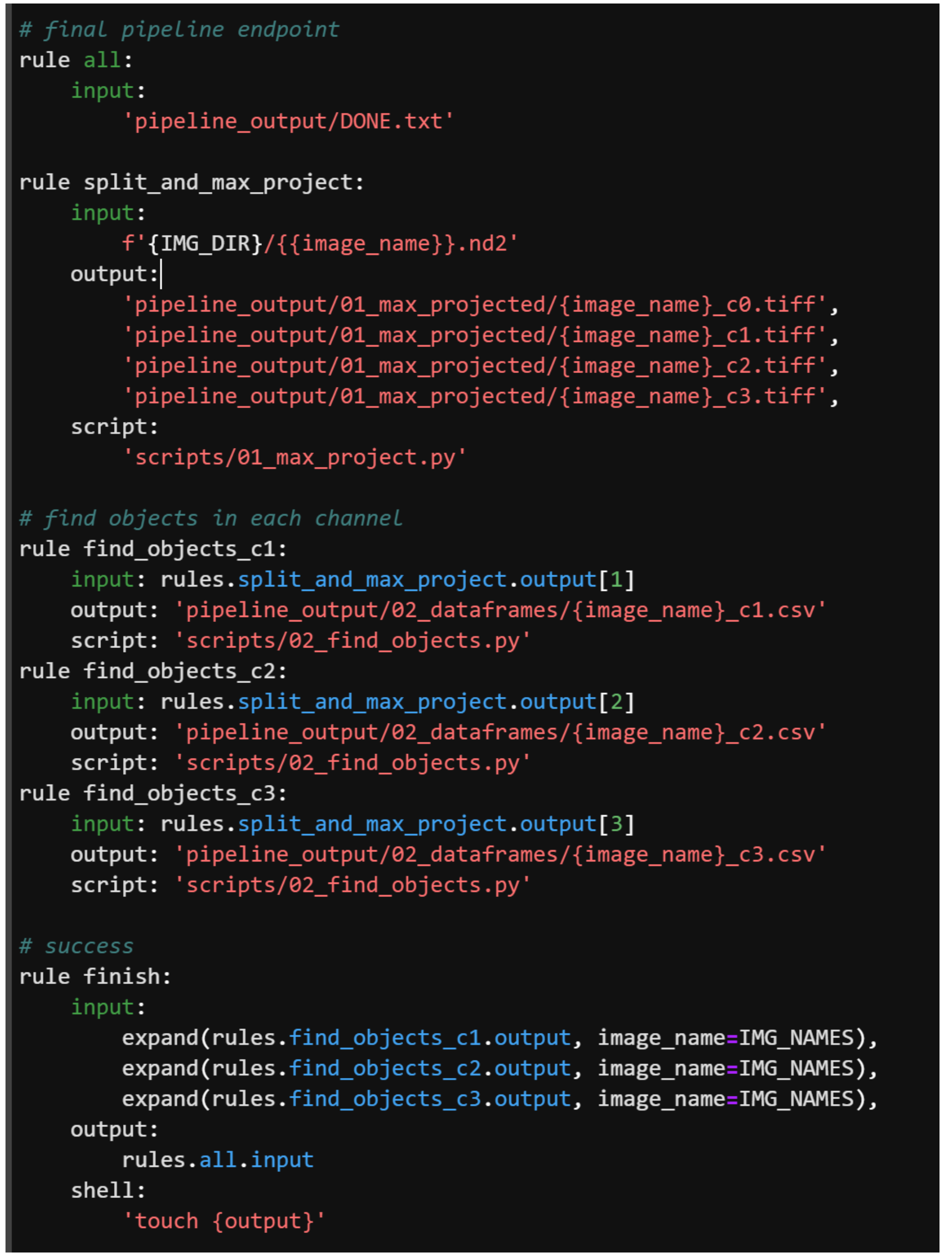
chr6_p30-488_p27-565_p28-647_001_c3.csv

Starting a Snakemake pipeline

One useful pattern

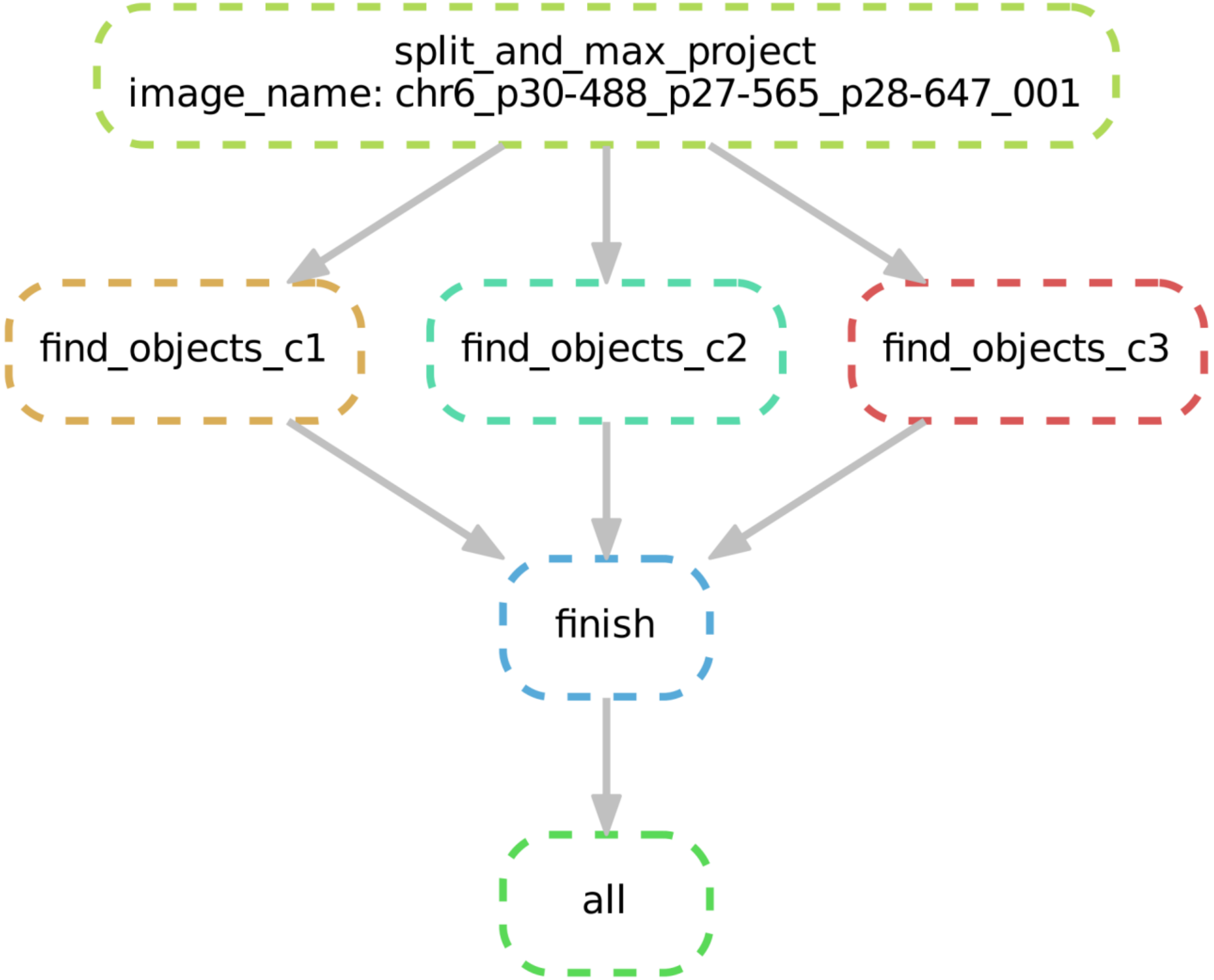


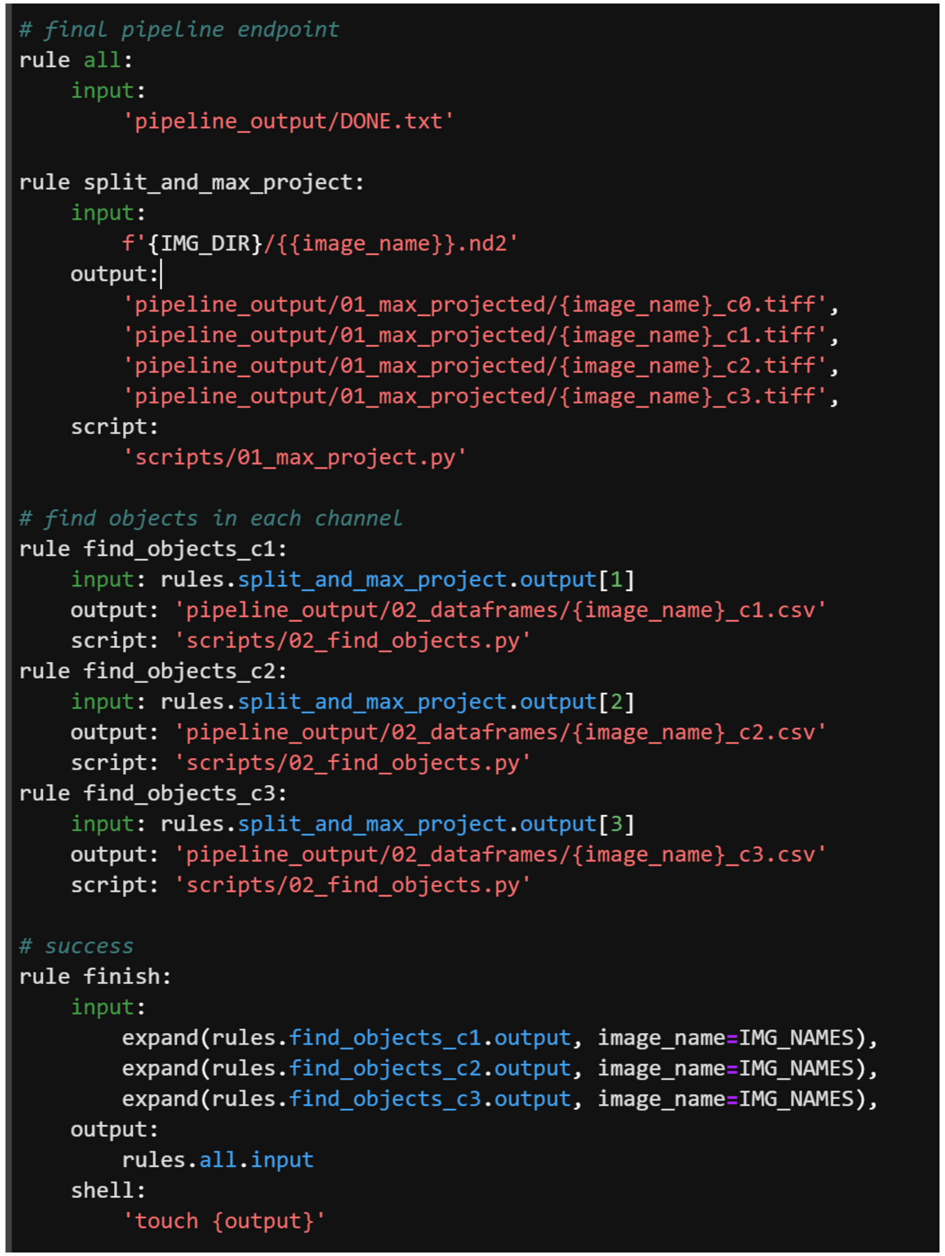
```
# final pipeline endpoint
rule all:
    input:
        'pipeline_output/DONE.txt'



# < pipeline logic here >



# success
rule finish:
#     input:
        # TODO make this rule depend on the last upstream step(s)
    output:
        rules.all.input
    shell:
        'touch {output}'
```



```
# final pipeline endpoint
rule all:
    input:
        'pipeline_output/DONE.txt'

rule split_and_max_project:
    input:
        f'{IMG_DIR}/{{image_name}}.nd2'
    output:
        'pipeline_output/01_max_projected/{image_name}_c0.tiff',
        'pipeline_output/01_max_projected/{image_name}_c1.tiff',
        'pipeline_output/01_max_projected/{image_name}_c2.tiff',
        'pipeline_output/01_max_projected/{image_name}_c3.tiff',
    script:
        'scripts/01_max_project.py'

# find objects in each channel
rule find_objects_c1:
    input: rules.split_and_max_project.output[1]
    output: 'pipeline_output/02_dataframes/{image_name}_c1.csv'
    script: 'scripts/02_find_objects.py'
rule find_objects_c2:
    input: rules.split_and_max_project.output[2]
    output: 'pipeline_output/02_dataframes/{image_name}_c2.csv'
    script: 'scripts/02_find_objects.py'
rule find_objects_c3:
    input: rules.split_and_max_project.output[3]
    output: 'pipeline_output/02_dataframes/{image_name}_c3.csv'
    script: 'scripts/02_find_objects.py'

# success
rule finish:
    input:
        expand(rules.find_objects_c1.output, image_name=IMG_NAMES),
        expand(rules.find_objects_c2.output, image_name=IMG_NAMES),
        expand(rules.find_objects_c3.output, image_name=IMG_NAMES),
    output:
        rules.all.input
    shell:
        'touch {output}'
```

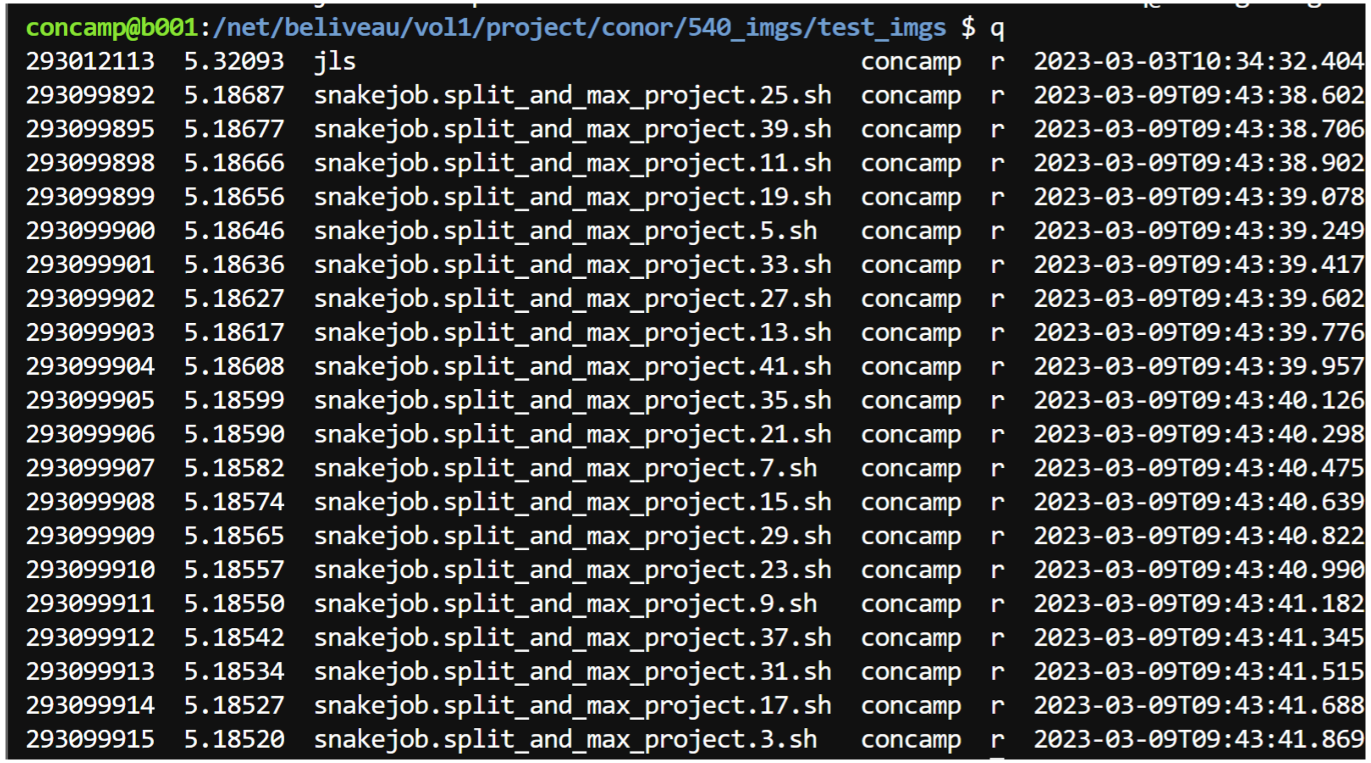
Starting a Snakemake pipeline



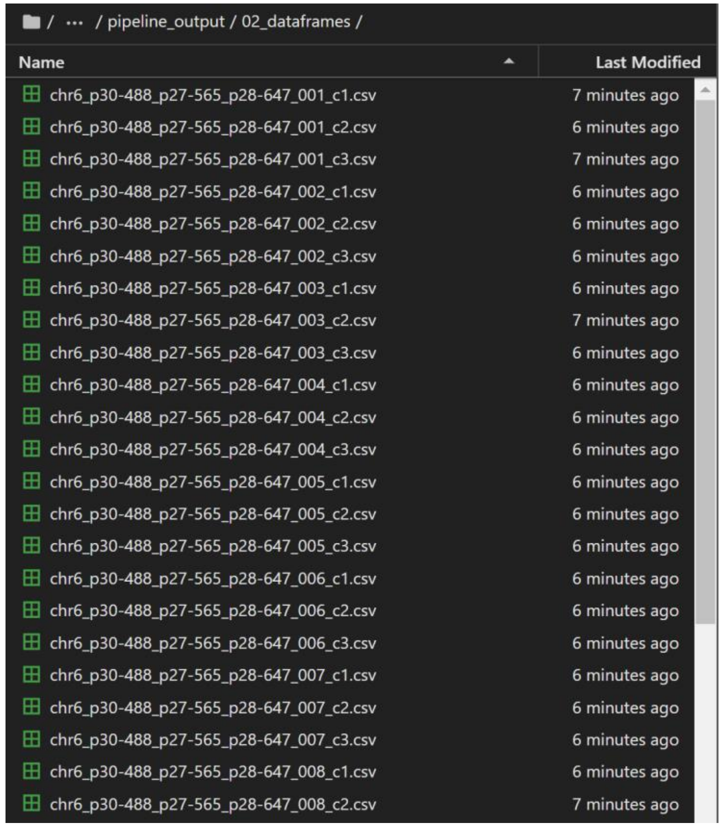


```
# final pipeline endpoint
rule all:
    input:
        'pipeline_output/DONE.txt'

rule split_and_max_project:
    input:
        f'{IMG_DIR}/{{image_name}}.nd2'
    output:
        'pipeline_output/01_max_projected/{image_name}_c0.tiff',
        'pipeline_output/01_max_projected/{image_name}_c1.tiff',
        'pipeline_output/01_max_projected/{image_name}_c2.tiff',
        'pipeline_output/01_max_projected/{image_name}_c3.tiff',
    script:
        'scripts/01_max_project.py'

# find objects in each channel
rule find_objects_c1:
    input: rules.split_and_max_project.output[1]
    output: 'pipeline_output/02_dataframes/{image_name}_c1.csv'
    script: 'scripts/02_find_objects.py'
rule find_objects_c2:
    input: rules.split_and_max_project.output[2]
    output: 'pipeline_output/02_dataframes/{image_name}_c2.csv'
    script: 'scripts/02_find_objects.py'
rule find_objects_c3:
    input: rules.split_and_max_project.output[3]
    output: 'pipeline_output/02_dataframes/{image_name}_c3.csv'
    script: 'scripts/02_find_objects.py'

# success
rule finish:
    input:
        expand(rules.find_objects_c1.output, image_name=IMG_NAMES),
        expand(rules.find_objects_c2.output, image_name=IMG_NAMES),
        expand(rules.find_objects_c3.output, image_name=IMG_NAMES),
    output:
        rules.all.input
    shell:
        'touch {output}'
```

Running at scale (parallelized)

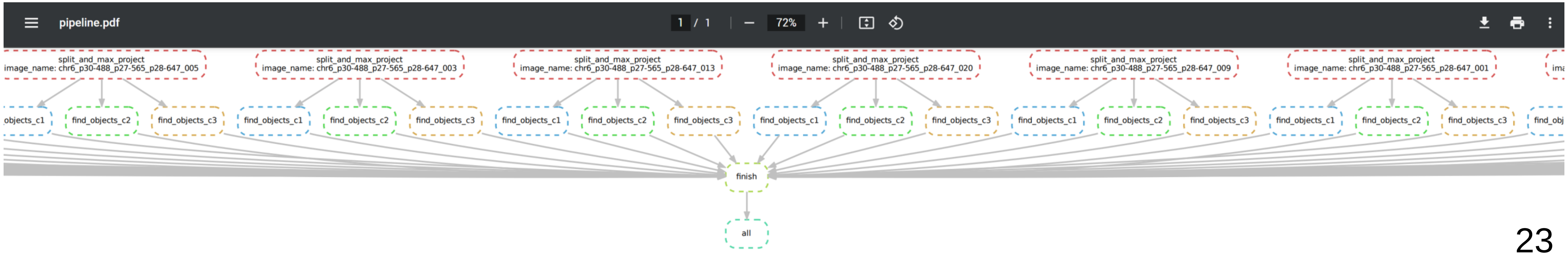


```
concamp@b001:/net/beliveau/vol1/project/conor/540_imgs/test_imgs $ q
293012113  5.32093  jls                                concamp  r  2023-03-03T10:34:32.404
293099892  5.18687  snakejob.split_and_max_project.25.sh  concamp  r  2023-03-09T09:43:38.602
293099895  5.18677  snakejob.split_and_max_project.39.sh  concamp  r  2023-03-09T09:43:38.706
293099898  5.18666  snakejob.split_and_max_project.11.sh  concamp  r  2023-03-09T09:43:38.902
293099899  5.18656  snakejob.split_and_max_project.19.sh  concamp  r  2023-03-09T09:43:39.078
293099900  5.18646  snakejob.split_and_max_project.5.sh   concamp  r  2023-03-09T09:43:39.249
293099901  5.18636  snakejob.split_and_max_project.33.sh  concamp  r  2023-03-09T09:43:39.417
293099902  5.18627  snakejob.split_and_max_project.27.sh  concamp  r  2023-03-09T09:43:39.602
293099903  5.18617  snakejob.split_and_max_project.13.sh  concamp  r  2023-03-09T09:43:39.776
293099904  5.18608  snakejob.split_and_max_project.41.sh  concamp  r  2023-03-09T09:43:39.957
293099905  5.18599  snakejob.split_and_max_project.35.sh  concamp  r  2023-03-09T09:43:40.126
293099906  5.18590  snakejob.split_and_max_project.21.sh  concamp  r  2023-03-09T09:43:40.298
293099907  5.18582  snakejob.split_and_max_project.7.sh   concamp  r  2023-03-09T09:43:40.475
293099908  5.18574  snakejob.split_and_max_project.15.sh  concamp  r  2023-03-09T09:43:40.639
293099909  5.18565  snakejob.split_and_max_project.29.sh  concamp  r  2023-03-09T09:43:40.822
293099910  5.18557  snakejob.split_and_max_project.23.sh  concamp  r  2023-03-09T09:43:40.990
293099911  5.18550  snakejob.split_and_max_project.9.sh   concamp  r  2023-03-09T09:43:41.182
293099912  5.18542  snakejob.split_and_max_project.37.sh  concamp  r  2023-03-09T09:43:41.345
293099913  5.18534  snakejob.split_and_max_project.31.sh  concamp  r  2023-03-09T09:43:41.515
293099914  5.18527  snakejob.split_and_max_project.17.sh  concamp  r  2023-03-09T09:43:41.688
293099915  5.18520  snakejob.split_and_max_project.3.sh   concamp  r  2023-03-09T09:43:41.869
```



/ ··· / pipeline_output / 02_dataframes /

Name	Last Modified
chr6_p30-488_p27-565_p28-647_001_c1.csv	7 minutes ago
chr6_p30-488_p27-565_p28-647_001_c2.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_001_c3.csv	7 minutes ago
chr6_p30-488_p27-565_p28-647_002_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_002_c2.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_002_c3.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_003_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_003_c2.csv	7 minutes ago
chr6_p30-488_p27-565_p28-647_003_c3.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_004_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_004_c2.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_004_c3.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_005_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_005_c2.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_005_c3.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_006_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_006_c2.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_006_c3.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_007_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_007_c2.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_007_c3.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_008_c1.csv	6 minutes ago
chr6_p30-488_p27-565_p28-647_008_c2.csv	7 minutes ago


pipeline.pdf
1 / 1
72%
split_and_max_project image_name: chr6_p30-488_p27-565_p28-647_005
split_and_max_project image_name: chr6_p30-488_p27-565_p28-647_003
split_and_max_project image_name: chr6_p30-488_p27-565_p28-647_013
split_and_max_project image_name: chr6_p30-488_p27-565_p28-647_020
split_and_max_project image_name: chr6_p30-488_p27-565_p28-647_009
split_and_max_project image_name: chr6_p30-488_p27-565_p28-647_001
find_objects_c1
find_objects_c2
find_objects_c3
finish
all

Outline

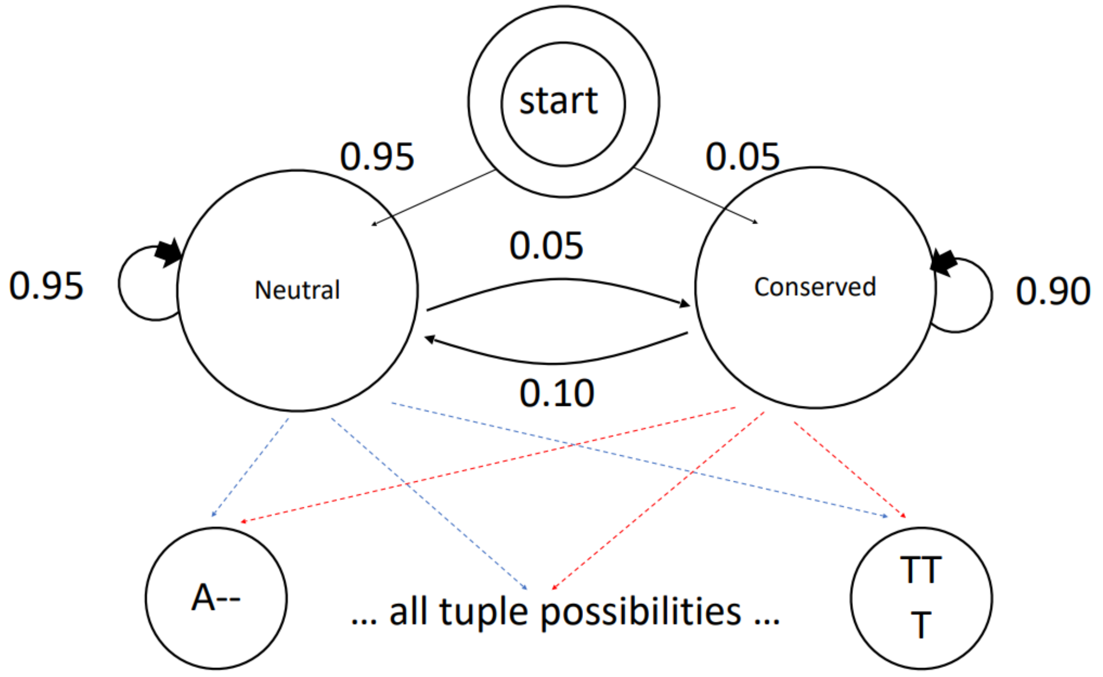
- Related topics:
  - Snakemake overview
  - Example image processing pipeline
- Homework 9 Questions

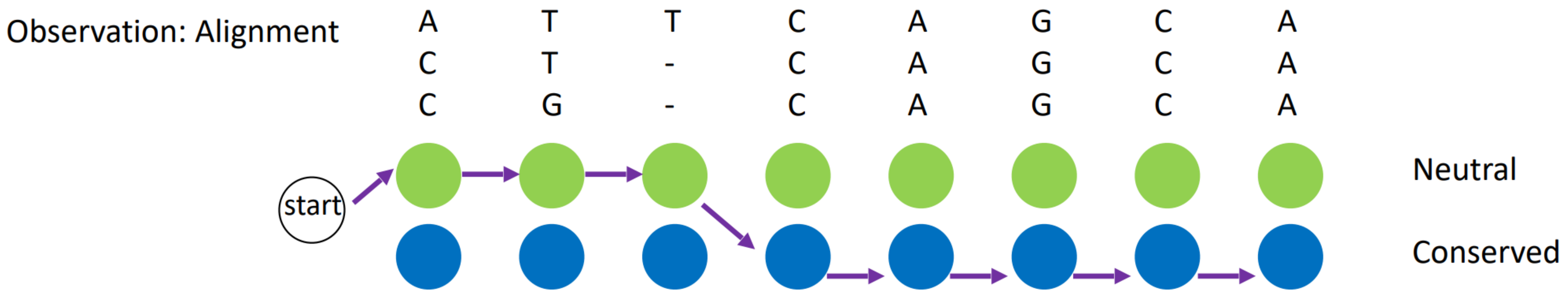
Homework 9 Overview

- ENCODE region 010 (chromosome 7)
- Multiple alignment of human, dog, and mouse
- 2 states:
  - neutral (fast-evolving)
  - conserved (slow-evolving)
- Emitted symbols are multiple alignment columns (e.g. ‘AAT’)
- Viterbi parse (no iteration)

CX HW9 slides:
http://bozeman.genome.washington.edu/compbio/mbt599_2022/TA_discussion/class20.pdf

HW9 – Model Structure





HW9 – Model Parameters

Alignment Column Counts Provided

Ancient Repeat Sequences

AAA	10222095
AAC	481243
AAT	420185
AAG	1415675
AA-	273456
ACA	852624
ACC	179459
ACT	99493
ACG	167810
AC-	29636
ATA	874547
ATC	113150
ATT	220714
ATG	185789
AT-	32253
AGA	2116012
AGC	139953
AGT	131553
AGG	881616
AG-	73372
A-A	760405
A-C	57350
A-T	56348
A-G	155911
A--	39186

1st base: human
2nd base: dog
3rd base: mouse

Putative Functional Sites

AAA	2375583
AAC	21337
AAT	10886
AAG	56328
AA-	3205
ACA	33210
ACC	12122
ACT	2270
ACG	5187
AC-	374
ATA	21805
ATC	2871
ATT	7426
ATG	4369
AT-	294
AGA	81919
AGC	4455
AGT	2735
AGG	50413
AG-	796
A-A	6234
A-C	557
A-T	350
A-G	1349
A--	1282

Calculate Emission Probabilities

- For ‘neutral’ state emission probabilities, use observed frequencies in neutral data set (ancient repeat sequences)
- For ‘conserved’ state emission probabilities, use observed frequencies in functional data set

Initiation, Transition Probabilities

- Given in problem set description

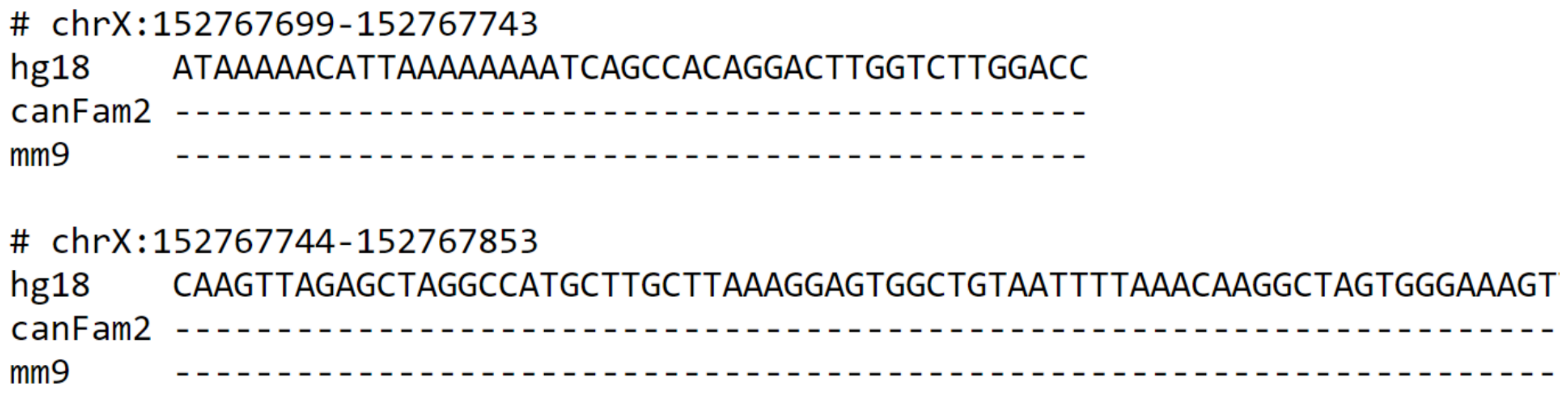
HW9 – Input Data

Original maf format:

- Sequences broken into alignment blocks based on the species included
- Official file format specs

Homework file format:

- Only 3 species
- Gaps in human sequence were removed and ambiguous bases replaced with ‘A’ for simplicity



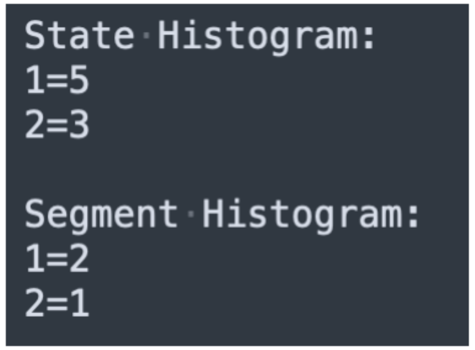
```
# chrX:152767699-152767743
hg18    ATAAAAACATTAAAAAAAATCAGCCACAGGACTTGGTCTTGGACC
canFam2 ---------------------------------------------
mm9     ---------------------------------------------

# chrX:152767744-152767853
hg18    CAAGTTAGAGCTAGGCCATGCTTGCTTAAAGGAGTGGCTGTAATTTTAAACAAGGCTAGTGGGAAAGT
canFam2 --------------------------------------------------------------------
mm9     --------------------------------------------------------------------
```

HW9 – Output

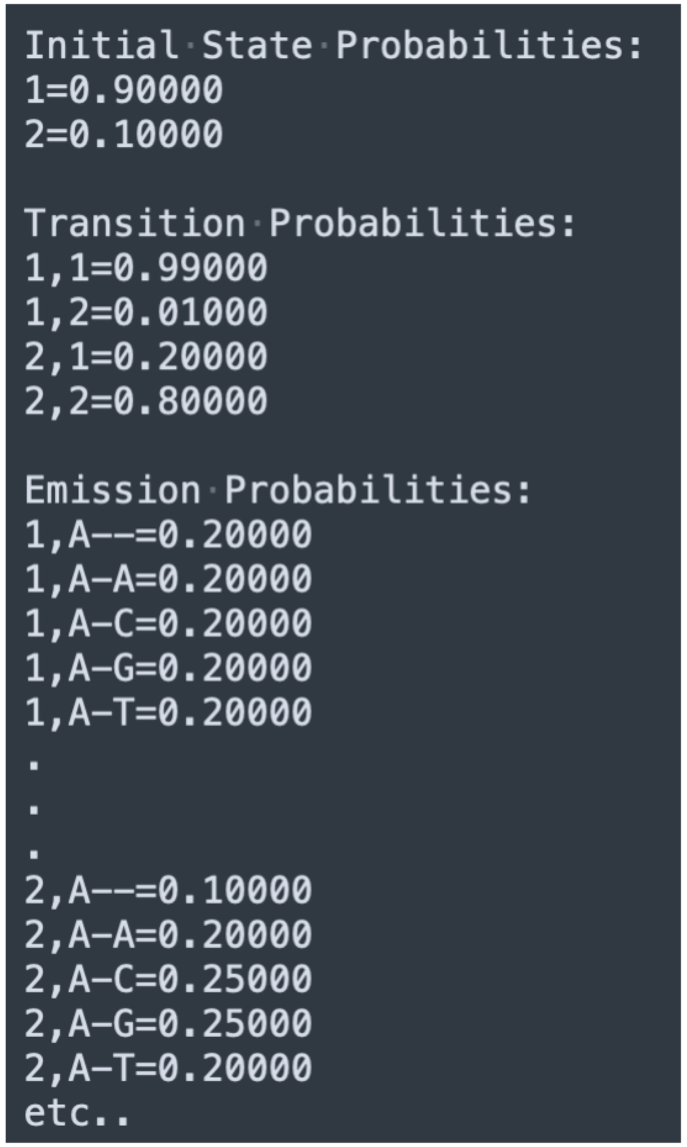
Output

- State and segment histograms
- Parameter values
  - Initiation/transition probabilities you were given in the assignment
  - Emission probabilities you calculated from neutral and conserved data sets
- Coordinates of 10 longest conserved segments (report positions relative to the start of the chromosome)
- Brief annotations for the 5 longest conserved segments (look at UCSC genome browser, and make sure using the correct genome version, e.g. hg18)

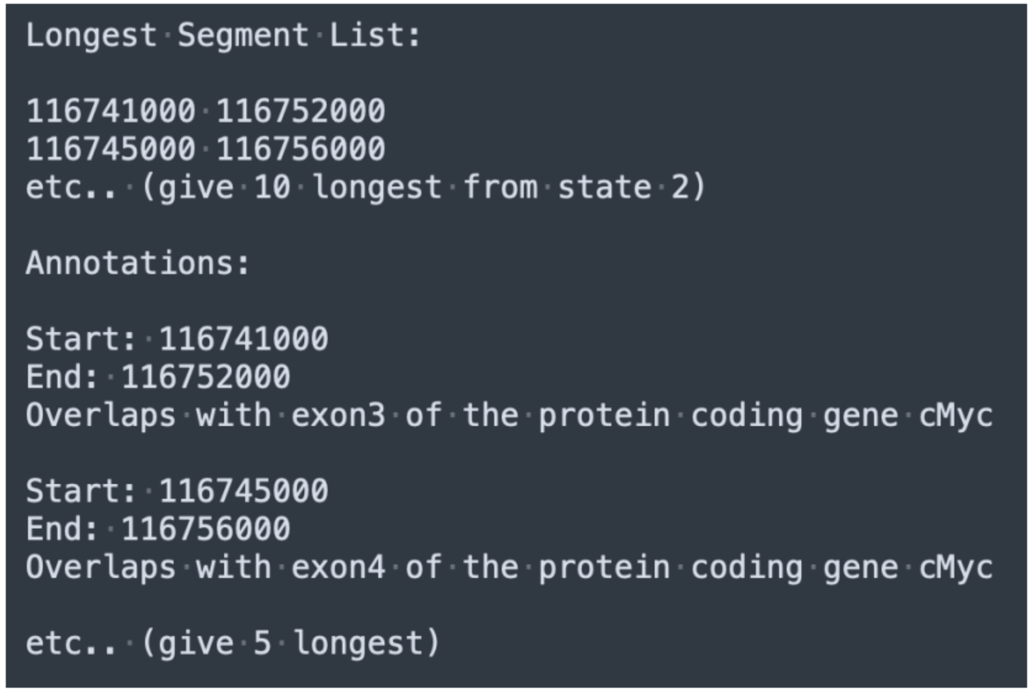
HW9 – Output



```
State Histogram:
1=5
2=3

Segment Histogram:
1=2
2=1
```



```
Initial State Probabilities:
1=0.90000
2=0.10000

Transition Probabilities:
1,1=0.99000
1,2=0.01000
2,1=0.20000
2,2=0.80000

Emission Probabilities:
1,A--=0.20000
1,A-A=0.20000
1,A-C=0.20000
1,A-G=0.20000
1,A-T=0.20000
.
.
.
2,A--=0.10000
2,A-A=0.20000
2,A-C=0.25000
2,A-G=0.25000
2,A-T=0.20000
etc..
```



```
Longest Segment List:

116741000 116752000
116745000 116756000
etc.. (give 10 longest from state 2)

Annotations:

Start: 116741000
End: 116752000
Overlaps with exon3 of the protein coding gene cMyc

Start: 116745000
End: 116756000
Overlaps with exon4 of the protein coding gene cMyc

etc.. (give 5 longest)
```