

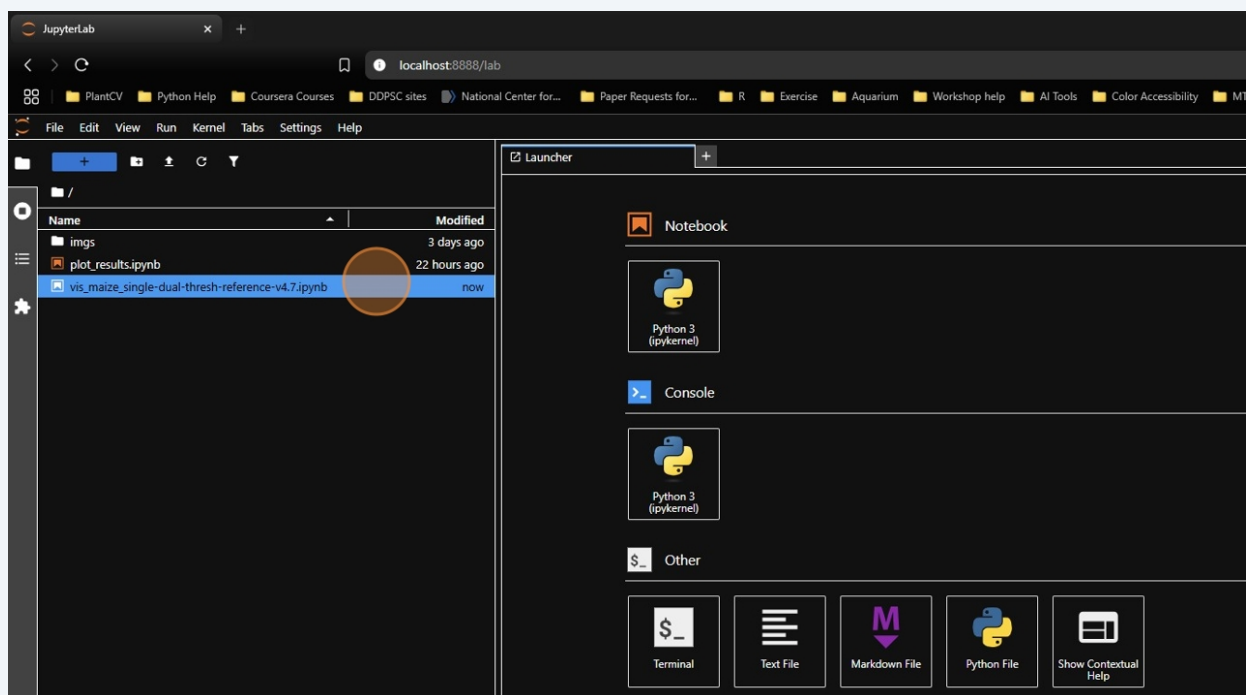
How to Parallelize Your Workflows Using PlantCV - Single Images

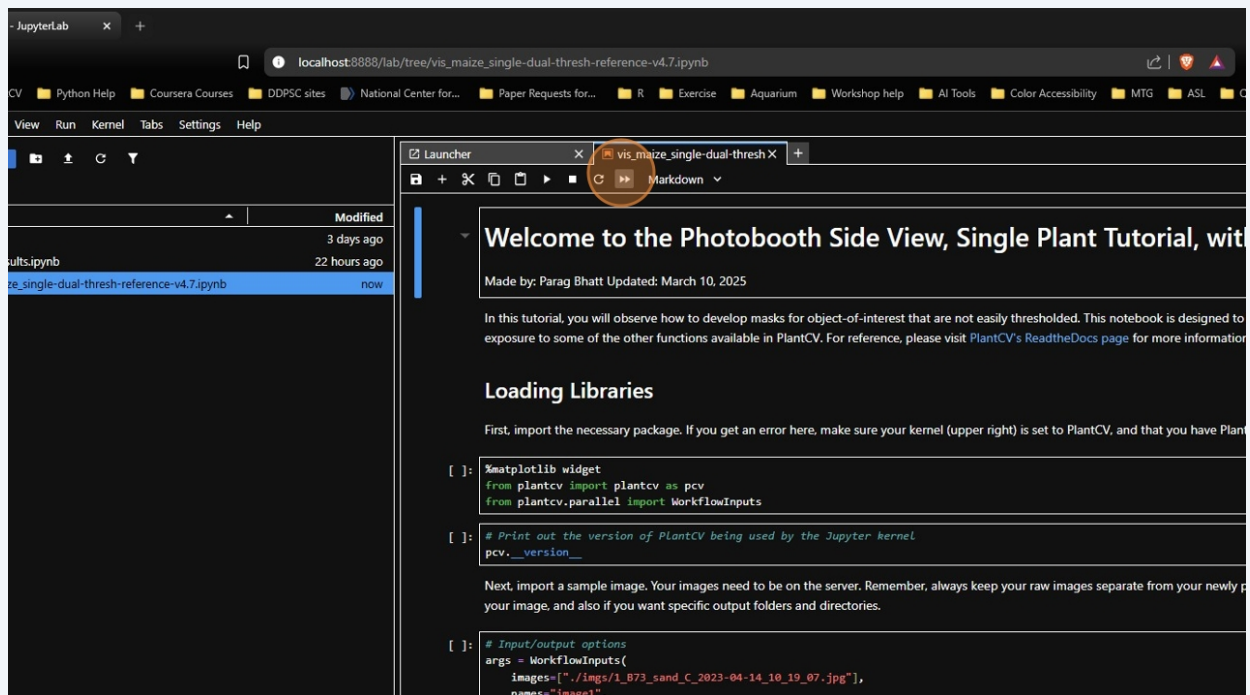
This guide provides step-by-step instructions on how to parallelize single plant phenotyping workflows using PlantCV. By following these steps, users can optimize their workflow efficiency and increase productivity.

1

Often you will start with a workflow from another tutorial that you will have then edited to work well for your experiment. Here we assume that you start with an .ipynb workflow that runs without errors on several representative test images. Ideally you have trained your workflow using a step-up image training protocol (i.e. training your workflow on 1 image > 2 images > 4 images > 10 images > 20 images)

NOTE: Make sure you have ran your workflow and have produced a JSON output file.

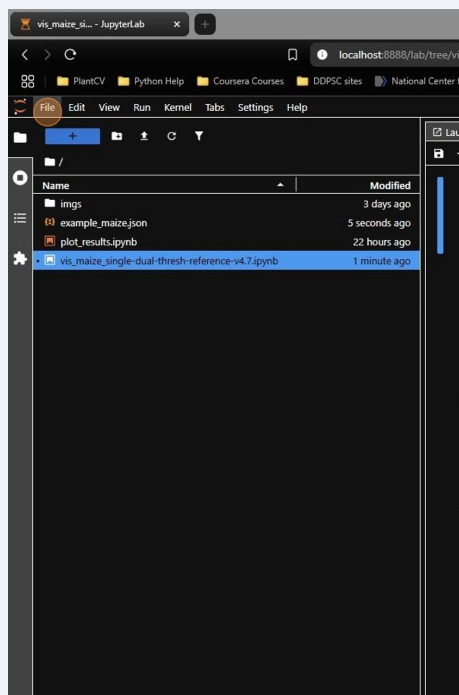


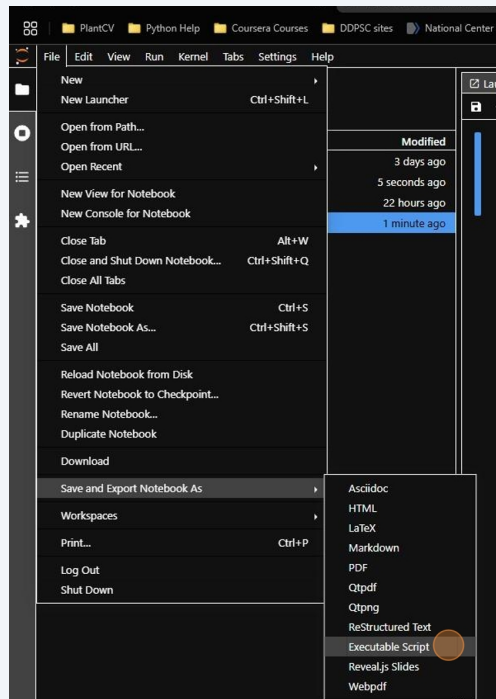


2

Convert your Jupyter Notebook (.ipynb) to an executable script (a Python script .py) selecting **File > Save and Export Notebook As... > Executable Script**

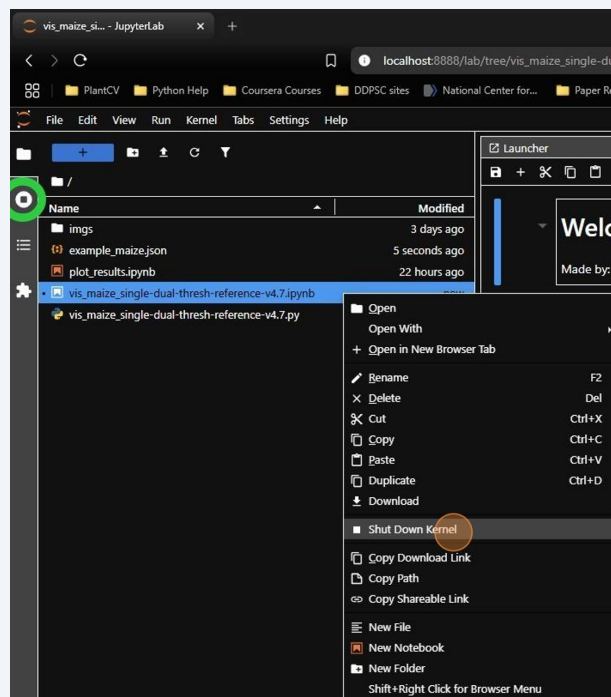
Save your Python script in the same directory as your Jupyter notebook for easy access. You may need to adjust the title of your script so it is concise (in the example, I am using a reference document and do not want "reference" in my final script.





3

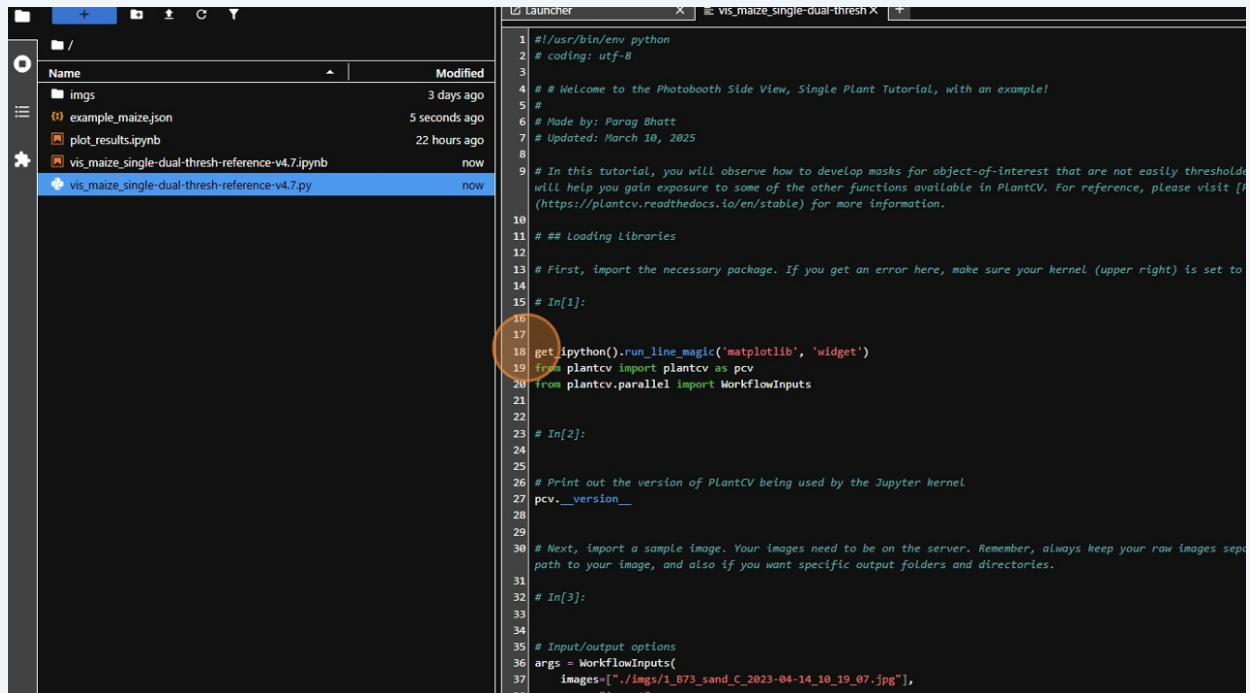
Right-click on the notebook version of your workflow and select **Shut Down Kernel**. This will help conserve resources while your script is being executed in parallel. If you have additional notebooks that are active, shut down those kernels as well by visiting the **Running Terminals and Kernels** button on the left side of the screen (Green Circle).



- 4 Open your new Python script. We will have to make some edits to make it a functioning workflow script.

The first thing we will want to do is "comment out" things we do not want to run in parallel. In the example image this would mean adding an "#" ahead of line 7: `get_ipython().run_line_magic('matplotlib', 'widget')`

You can use the hotkey **Ctrl + /** or **Cmd + /** (depending on your operating system) to comment the whole line out.

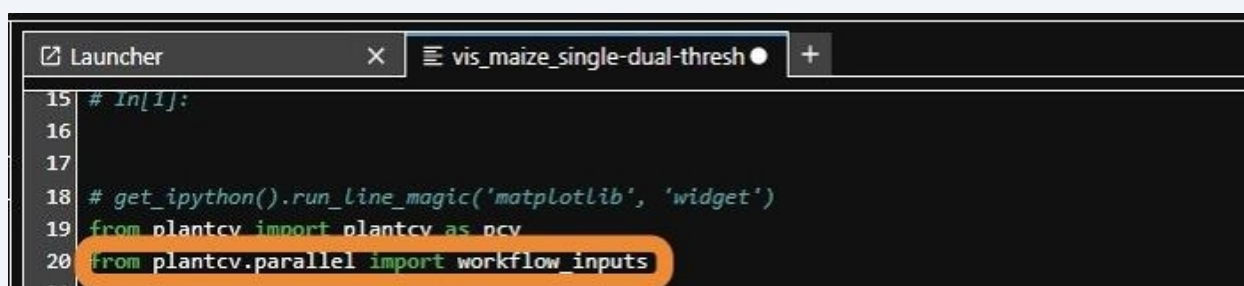


```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # Welcome to the Photobooth Side View, Single Plant Tutorial, with an example!
5
6 # Made by: Parag Bhatt
7 # Updated: March 10, 2025
8
9 # In this tutorial, you will observe how to develop masks for object-of-interest that are not easily thresholded.
10 # This will help you gain exposure to some of the other functions available in PlantCV. For reference, please visit [1]
11 # (https://plantcv.readthedocs.io/en/stable) for more information.
12
13 ## Loading Libraries
14
15 # First, import the necessary package. If you get an error here, make sure your kernel (upper right) is set to
16 # In[1]:
17
18 get_ipython().run_line_magic('matplotlib', 'widget')
19 from plantcv import plantcv as pcv
20 from plantcv.parallel import WorkflowInputs
21
22 # In[2]:
23
24
25 # Print out the version of PlantCV being used by the Jupyter kernel
26 pcv.__version__
27
28
29 # Next, import a sample image. Your images need to be on the server. Remember, always keep your raw images separate
30 # path to your image, and also if you want specific output folders and directories.
31
32 # In[3]:
33
34
35 # Input/output options
36 args = WorkflowInputs(
37     images=["./imgs/1_073_sand_C_2023-04-14_10_19_07.jpg"],
38     names="image1"
```

- 5 We need to make a change to our import statements to say:

`from plantcv.parallel import workflow_inputs` in place of `from plantcv.parallel import WorkflowInputs`

This will tell the parallel process to use `workflow_inputs()` to set parameters based on the configuration JSON file instead of using the hard coded arguments from the notebook.



```
15 # In[1]:
16
17
18 # get_ipython().run_line_magic('matplotlib', 'widget')
19 from plantcv import plantcv as pcv
20 from plantcv.parallel import workflow_inputs
21
```

6

Next, we will change our argument definitions for this workflow by "commenting out" the our **WorkflowInputs** parameters. Remember that you can comment out an entire section by using **CTRL or Cmd + /** in Jupyter.

See the image below:

```

15 # In[1]:
16
17
18 # get_ipython().run_line_magic('matplotlib', 'widget')
19 from plantcv import plantcv as pcv
20 from plantcv.parallel import workflow_inputs
21
22
23 # In[2]:
24
25
26 # Print out the version of PlantCV being used by the Jupyter kernel
27 pcv.__version__
28
29
30 # Next, import a sample image. Your images need to be on the server.
31 # path to your image, and also if you want specific output folders and
32
33 # In[3]:
34
35 # input/output options
36 args = WorkflowInputs(
37     images=["./imgs/1_B73_sand_C_2023-04-14_10_19_07.jpg"],
38     names="image1",
39     result="example_maize.json",
40     outdir=".",
41     writeimg=False,
42     debug="plot"
43 )
  
```

7

For your workflow you may also want to comment out commands that only print images or that look for Google Colab environments. You can use the hotkey **Ctrl + /** or **Cmd + /** (depending on your operating system) to comment the whole line out.

[Click here](#)

```
95
96
97 ## Detecting and Calibrating Colors with Color Cards/Color Checkers
98
99 # Color Cards (or color checkers) are useful reference tools used by imaging exper-
reproduction in their images. The device usually consists of 24 color swatches (o
skin color, water, leaves, flowers, etc.) in addition to white, gray, and black t
also be used to calibrate the sizes of other objects in your image so it is imper
how to use a color card, visit the following [article](https://petapixel.com/how-
100
101 # In this first step, we will [detect](https://plantcv.readthedocs.io/en/stable/t
the Transform subpackage, all we need to do is load our image into the function a
as a scaling factor to determine the measurements of your object-of-interest.
102
103 # In[8]:
104
105
106 # Load your rotated or cropped image into the rgb_img argument to detect the colo
107 cc_mask = pcv.transform.detect_color_card(rgb_img=rotated_img)
108
109 # We will also print the average chip size and store the values in outputs.observe
110 print(pcv.outputs.metadata['median_color_chip_size']['value'])
111
112
113 # In[9]:
114
```

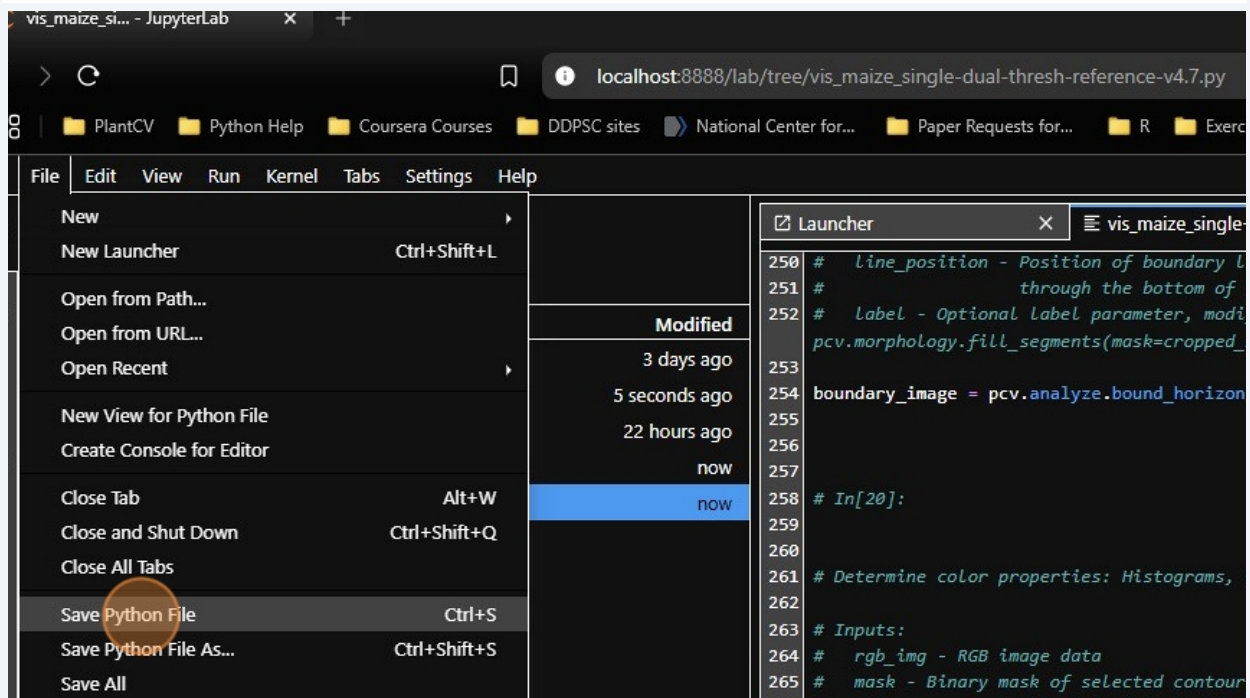
```
149 ## Creating a Mask of your Object-of-Interest
150
151 # In[12]:
152
153
154 # Update params related to plotting so we can see better
155 pcv.params.text_size=50
156 pcv.params.text_thickness=15
157
158
159 # Look at the colorspace - which of these looks the best for
160 colorspace_img = pcv.visualize.colors_spaces(rgb_img=img_cc,
161
162
163 # In[13]:
164
165
166 # Look at the colorspace above and determine which spaces
167 # We will use **pcv.visualize.pixel_scatter_plot** to plot
168 # The horizontal and vertical coordinates will represent th
169 # where each dot is given by the original RGB color of the
```



```
149 # ## Creating a Mask of your Object-of-Interest
150
151 # In[12]:
152
153
154 # Update params related to plotting so we can see better
155 pcv.params.text_size=50
156 pcv.params.text_thickness=15
157
158
159 #Look at the colorspace - which of these Looks the best for masking? Which channe
160 # |colorspace_img = pcv.visualize.colorsapces(rgb_img=img_cc, original_img=False)
161
162
163 # In[13]:
164
165
166 # Look at the colorsapces above and determine which spaces provide you with the g
167 # We will use **pcv.visualize.pixel_scatter_plot** to plot the pixel intensities
168 # The horizontal and vertical coordinates will represent the pixel intensities fr
169 # where each dot is given by the original RGB color of the pixel.
170
171 # For this image,
172 color_scatter = pcv.visualize.pixel_scatter_plot(paths_to_imgs = ["/imgs/1_B73_s
173
174
```

8 Save the changes you made to your Python script.

```
250 # Line_position - Position of boundar
251 # through the bottom
252 # Label - Optional label parameter, r
253 pcv.morphology.fill_segments(mask=cropp
254
255 boundary_image = pcv.analyze.bound_hor
256
257
258 # In[20]:
259
260
261 # Determine color properties: Histogram
262
263 # Inputs:
264 # rgb_img - RGB image data
265 # mask - Binary mask of selected area
```



In the next step we will have to change the working directory in terminal to the location where our scripts are. There are a couple of ways to obtain the file path for your working directory:

1. In Jupyter if you hover over the folder icon in the file browser it will show the working directory.
2. In Jupyter you can open a terminal and run `pwd` to print the working directory.

NOTE: You may want to copy the file path to a document so you can see how much of the file path is copied (the full file path will not be copied).

9

In this guide, the current working directory is local to the root directory, so we did not need to change directories (See image below, outlined in green). Remember that to change directories, you need to use the operator `cd`. If you haven't done so already, activate your PlantCV environment by typing: `conda activate plantcv`

Once you have changed your working directory in conda, type `dir` or `ls` to view the contents of the directory. Make sure that your Python script is in the correct location. (See image below, outlined in orange).

```
Miniforge Prompt - conda deactivate plantcv - conda deactivate - conda deactivate
(base) C:\Users\pbhatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>conda activate plantcv
(plantcv) C:\Users\pbhatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>dir
Volume in drive C has no label.
Volume Serial Number is 769C-CE5C

Directory of C:\Users\pbhatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking

03/24/2025  09:12 PM    <DIR>          .
03/24/2025  09:12 PM    <DIR>          ..
03/24/2025  09:12 PM    <DIR>          .ipynb_checkpoints
03/21/2025  10:40 AM             1,935,082  example_maize.json
03/24/2025  09:12 PM    <DIR>          imgs
03/21/2025  11:08 AM             1,841  plot_results.ipynb
04/29/2024  08:00 AM             18,684  START-HERE_vis_maize_single-dual-thresh.ipynb
03/10/2025  02:00 PM             4,807,402  vis_maize_single-dual-thresh-reference-v4.2.2.ipynb
03/21/2025  09:45 AM             4,390,528  vis_maize_single-dual-thresh-reference-v4.7.ipynb
03/21/2025  09:47 AM             11,781  vis_maize_single-dual-thresh-reference-v4.7.py
05/03/2024  10:04 AM             4,050,253  vis_maize_single-dual-thresh-reference-v4.7.ipynb
              7 File(s)          15,215,571 bytes
              4 Dir(s)          80,992,677,888 bytes free

(plantcv) C:\Users\pbhatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>
```

10

Now we need to create our parallel configuration file by typing:

```
plantcv-run-workflow --template config.json
```

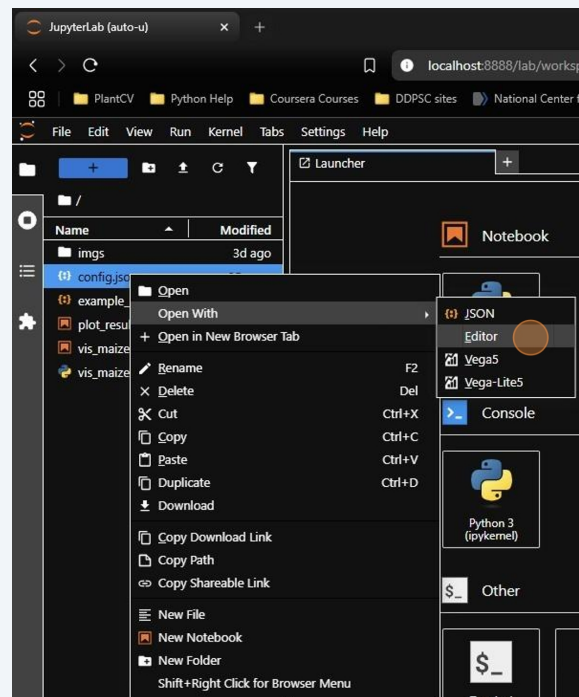
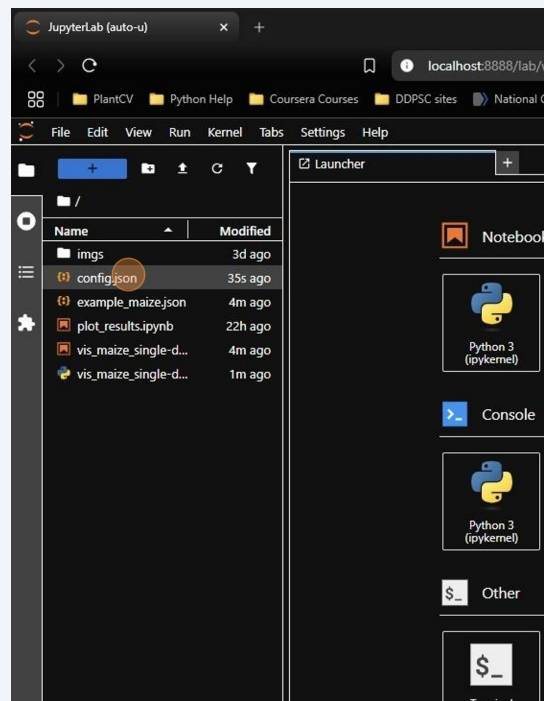
Since we have changed our working directory to our current folder, you should see **config.json** appear in the file explorer like shown below.

```
(plantcv) C:\Users\pbhatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>plantcv-run-workflow --template config.json
```

11

Since we have changed our working directory to our current folder, you should see **config.json** appear in the file explorer like shown below.

We will need to edit **config.json** so that we can configure the parallel analysis. Right-click **config.json** and hover over **Open With** and select **Editor** to make changes to the file.



In Editor, make changes to **config.json**, common things that should be changed are listed below but the full set of options is in the [documentation](#) and is worth reviewing.

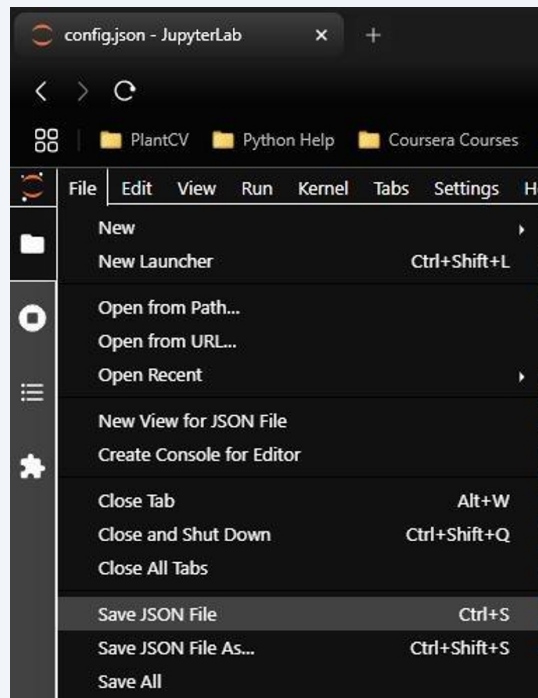
- "input_dir": "./imgs" *[Put file path/name of input directory for images you want analyzed]*
- "json": "example_maize.json" *[Put the path/name of the data output file (located in the args container under results within your workflow)]*
- "filename_metadata": ["id", "genotype", "soiltype", "treatment", "timestamp"] *[list of metadata terms to collect. Supported metadata terms include: camera, imgtype, zoom, exposure, gain, frame, lifter, timestamp, id, plantbarcode, treatment, cartag, measurementlabel, and other]. You can create custom metadata terms, see below.*
- "workflow": "multi-plant-workflow.py" *[path/name of user-defined (your) PlantCV workflow Python script]*
- "img_outdir": "./output_images" *[path/name of output directory where measured images will be stored. Default is "./output_images"]*
- "imgformat": "jpg" *[image file format/extension. Default is "png"]*
- "delimiter": "(.+)(.+)(.+)(.{1})(\d{4}-\d{2}-\d{2}\d{2}\d{2}\d{2})" *[filename separator syntax for collecting metadata terms above]*
- "timestampformat": "%Y-%m-%d_%H_%M_%S" *[date format as observed in your naming scheme. For explanation what each of the symbols mean, see the python [time format documentation](#)]*
- "append": false *[(bool, default = False): if **True** will append results to an existing json file. If **False**, will delete previous results stored in the specified JSON file.]*
- "cluster": "LocalCluster" *[There are several cluster types, the default option is "LocalCluster" which will run in parallel on the machine you run the run workflow command from. The complete list of options is: "LocalCluster", "HTCondorCluster", "LSFCluster", "MoabCluster", "OARCluster", "PBSCluster", "SGECluster", and "SLURMCluster" which can be read about in the [dask docs](#).]*
- cluster_config:
 - n_workers: In the example below this is still 1, but you will increase this based on how many cores you have available/want to use. This controls the number of workers to run in parallel. The "cores" argument is how many cores each worker needs, which will almost always stay as 1 which is considered sequential processing. Adding more workers will push it towards multiprocessing (parallel execution).

13

The metadata terms we defined in `filename_metadata` are not typically coded in the configuration file, so we will need to customize the list for our needs. Scroll to the bottom of the file and replace the metadata terms with those listed below so they reflect the ones defined under `filename_metadata`.

File Name	Last Modified	Line	Code
example_maize.json	3d ago	23	<code>"cluster": "LocalCluster",</code>
plot_results.ipynb	3d ago	24	<code>"cluster_config": {</code>
START-HERE_vis_...	10mo ago	25	<code> "n_workers": 1,</code>
vis_maize_single-d...	14d ago	26	<code> "cores": 1,</code>
vis_maize_single-d...	3d ago	27	<code> "memory": "1GB",</code>
vis_maize_single-d...	3d ago	28	<code> "disk": "1GB",</code>
vis_maize_single-d...	3d ago	29	<code> "log_directory": null,</code>
vis_maize_single-d...	10mo ago	30	<code> "local_directory": null,</code>
		31	<code> "job_extra_directives": null</code>
		32	<code>},</code>
		33	<code>"metadata_terms": {</code>
		34	<code> "id": {</code>
		35	<code> "label": "image identifier",</code>
		36	<code> "datatype": "<class 'str'>",</code>
		37	<code> "value": "none"</code>
		38	<code> },</code>
		39	<code> "genotype": {</code>
		40	<code> "label": "plant genotype",</code>
		41	<code> "datatype": "<class 'str'>",</code>
		42	<code> "value": "none"</code>
		43	<code> },</code>
		44	<code> "soiltype": {</code>
		45	<code> "label": "soil type",</code>
		46	<code> "datatype": "<class 'str'>",</code>
		47	<code> "value": "none"</code>
		48	<code> },</code>
		49	<code> "treatment": {</code>
		50	<code> "label": "treatment identifier",</code>
		51	<code> "datatype": "<class 'str'>",</code>
		52	<code> "value": "none"</code>
		53	<code> },</code>
		54	<code> "timestamp": {</code>
		55	<code> "label": "datetime of image",</code>
		56	<code> "datatype": "<class 'datetime.datetime'>",</code>
		57	<code> "value": null</code>
		58	<code> }</code>
		59	<code>}</code>
		60	<code>}</code>

14 Save the changes you have made to the **config.json**



15 Now that we have made the necessary changes to our parallel configuration file, it is time for us to run our workflow. To execute your parallel analysis, return to your terminal and type `plantcv-run-workflow --config config.json` into the prompt.

```
Miniforge Prompt - conda deactivate plantcv - conda deactivate - conda deactivate - plantcv-run-workflow --config config.json
(plantcv) C:\Users\pbhatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>plantcv-run-workflow --config config.json
```



If you successfully set up your **config.json** then you should see a number of files found and a progress bar on your screen with how long it will take to analyze your dataset. You will also see that your job list will include X workflows.

If you did not set up your **config.json** then you will receive error messages that detail where PlantCV is having issues finding an image directory, your workflow, incorrect date formats, etc.

When the job has completed, you will see that PlantCV automatically converts the JSON file into a CSV so you can carry the data over to whichever statistical analysis software.

```
Miniforge Prompt - conda deactivate plantcv - conda deactivate - conda deactivate
(plantcv) C:\Users\pghatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>plantcv-run-workflow --config config.json
Starting run 2025-03-24_21-52-25

Reading image metadata...
Reading image metadata took 4.643219709396362 seconds.
Building job list...
Task list includes 10 workflows
Building job list took 0.12076846557617188 seconds.
Processing images...
[ ] | 0% Completed | 28.4sDeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev
58g432fcd2f.d20250311
[ ] | 0% Completed | 29.7sDeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev
58g432fcd2f.d20250311
[ ] | 0% Completed | 30.6sDeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev
58g432fcd2f.d20250311
[ ] | 0% Completed | 31.8sDeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev
58g432fcd2f.d20250311
[ ] | 0% Completed | 32.1sDeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev
58g432fcd2f.d20250311
DeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev60g432fcd2f.d20250311
DeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev60g432fcd2f.d20250311
DeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev60g432fcd2f.d20250311
#####
[ ] | 80% Completed | 1min 13.5sDeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev60g432fcd2f.d20250311
DeprecationWarning: The 'label' parameter is no longer utilized, since color chip size is now metadata. It will be removed in PlantCV v5.0. Current PlantCV version: 4.7.dev60g432fcd2f.d20250311
Processing images took 87.97899580001831 seconds. Completed | 1min 21.0s[2K]
Processing results...
Processing results took 0.201975584018015137 seconds.
Converting json to csv...
Processing results took 0.1443397998808145 seconds.

(plantcv) C:\Users\pghatt\Documents\Workshop\Single-Plant-Analysis-Tutorial\02_Advanced-Maize-Masking>
```

16 You should see two CSV files appear in your directory:

- **example-maize.json-single-value-traits.csv**
- **example-maize.json-multi-value-traits.csv**

The **single-value-traits.csv** file will be in wide format, with a column per trait, whereas the **multi-value-traits.csv** file will be in long format, with one row per value/label. The hierarchical organization of these files enable more efficient data processing downstream.

