High-throughput Image Phenotyping Scripts Guide

Konstantin Divilov
kd379@cornell.edu

March 21, 2017

1 Prerequisites

Before continuing, please install Python 3 (the scripts will not work with Python 2), e.g., Python 3.6.0, (https://www.python.org/downloads/) with the following packages: **cv2**, **numpy**. All other packages used in the scripts are included in the Python Standard Library. OpenCV (**cv2**) is the main workhorse of the scripts, and it is a library of functions that can be called within a script to perform simple to complex image processing algorithms. This keeps the code simple to read and easy to modify for purposes outside of those intended. However, this does make the underlying code functionality abstract and non-transparent, and thus we suggest you look up any functions of interest in the OpenCV documentation (http://docs.opencv.org/). Python package installation is OS-dependent, i.e., Windows and Linux and OS X users have different ways of installing packages. If you are using Windows, it is probably simplest to install PyCharm (https://www.jetbrains.com/pycharm/download/), a Python integrated development environment (IDE), and install the packages **opencv-python** and **numpy** under Settings → Project → Project Interpreter. With Linux/OS X, one should first install pip (https://pip.pypa.io/en/stable/installing/), and then install the packages using **pip install opencv-python** and **pip install numpy**.

2 Introduction

In order to analyze leaf discs traits using the scripts, one first has to obtain pictures of leaf discs. The methodology used can be found in the paper accompanying this guide[1]. We were able to fit 12 rows and 15 columns of 1 cm in diameter leaf discs on a 30 × 20 cm Pyrex dish, for a total of 180 leaf discs. We did not use every space allotted, and images with empty spaces can be dealt with as if all leaf discs are present with one exception, which is that the size of the image has to be similar to ones that have all or most of the leaf discs present or else the code has to be adjusted. So, if one is taking pictures of leaf discs with 4 rows and 5 columns, and a column is missing, one should not take a picture with only 4 columns but rather should take a picture with the empty column. If one does this, the diameter of the leaf discs in pixels will be roughly constant. Pictures of the leaf discs were taken with leaf discs in groups of 20 (4 rows, 5 columns) for a total of 9 images per dish. We used an iPhone 5s to take pictures, although any similar smartphone/digital camera can be used. The images were downloaded as JPEG files, but the scripts work with any file format. The names of the downloaded images were given in order of when the image was taken, e.g., IMG_123 (first image taken), IMG_124, IMG_125 (last image taken). We changed the image names in bulk from what they were to 1 to 9 (or in our case 1 to 72, as we analyze 8 dishes at once, i.e., an entire experiment at one time point post-inoculation) using the Bulk Rename Utility (Windows only) (http://www.bulkrenameutility.co.uk/). It should be noted that if the grapevine genotypes you have selected to phenotype have leaf trichomes on the abaxial side, the output of the scripts will correspond to a different phenotype than if they were glabrous (Table 1).

[1] A link to the paper after acceptance will go here.

Table 1: The final output phenotype if...

		Sporulation present?	
		No	Yes
Leaf trichomes present?	No	Random noise (ε)	Sporulation quantity + ε
	Yes	Leaf trichome quantity + ε	Sporulation and leaf trichome quantity + ε

3 Quickstart Guide

There are four scripts in total, **crop.py**, **circles.py**, **lines.py**, **values.py**. We will now explain how to go from nine raw JPEG images named 1.jpg, 2.jpg, ..., 9.jpg to a file containing the phenotype of each leaf disc in the images. We assume that anytime one of the scripts is executed, that you have the particular **.py** file in the current directory.

- Open your command language interpreter, e.g., Windows Command Prompt or Bash.
- **cd [folder containing raw images]**
    1. This command moves you from your current directory to the one containing your images. Use **cd ..** to move out of a directory if needed.

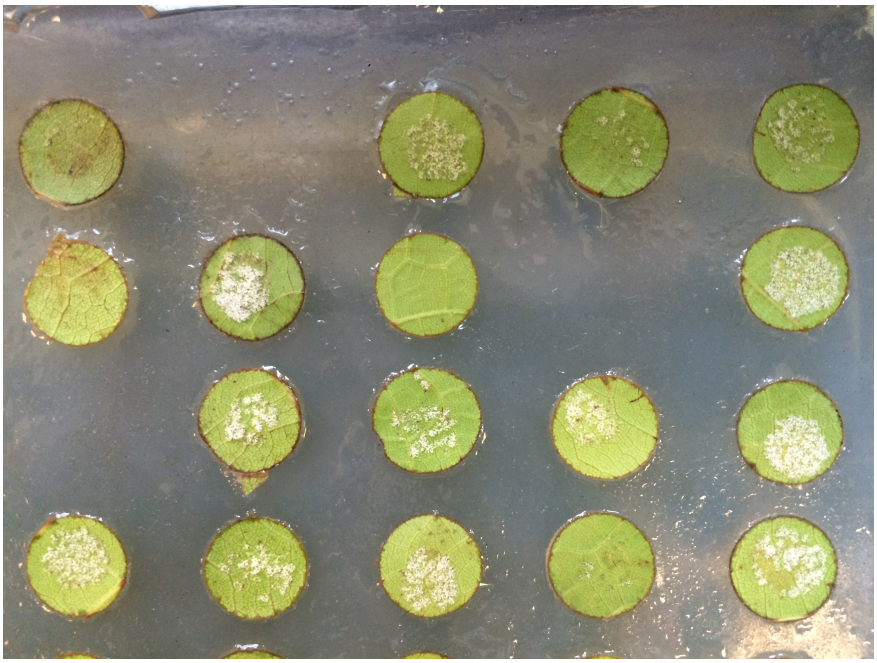

Figure 1: An uncropped image taken by an iPhone.

- **python crop.py -s 1 -e 9**
    1. This script will crop images 1.jpg through 9.jpg using the parameters in the script header and place the cropped images in a folder called **crop**. Change the file extension option if you have files other than .jpg by adding **-ext png** to the command if you have files in png format, for example.
    2. Manually inspect the cropped images in the folder **crop** to ensure that all leaf discs you want analyzed are still there.
    3. Manually crop images if needed.

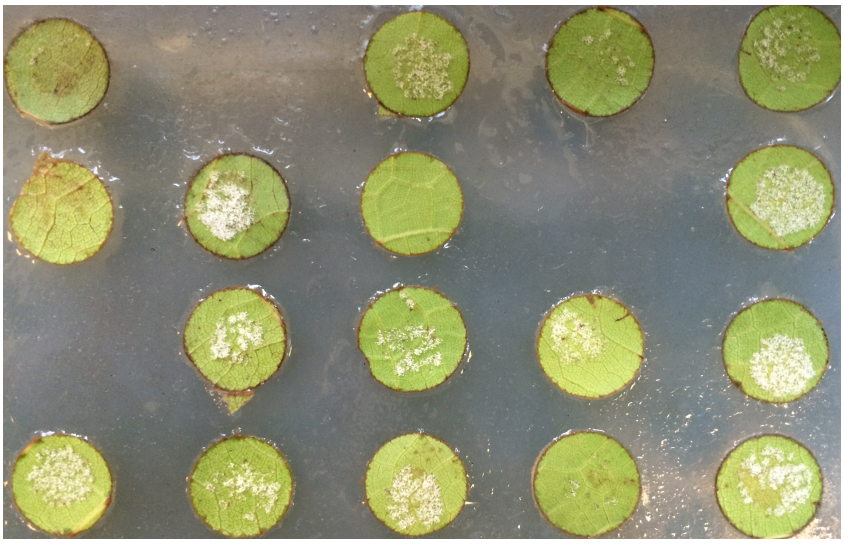

Figure 2: The image after cropping.

- **cd crop**
- **python circles.py -s 1 -e 9**
    1. This script will detect leaf discs as circles in images 1.jpg through 9.jpg using the options set in the script header and will place the images with the detected circles in a folder called **circles**.
    2. Manually inspect the images in the folder **circles** to ensure that all leaf discs have purple circles around them and that the y coordinate of the purple centers of all the leaf discs within a row are within the y coordinate range of the blue vertical line specific for that row. The circle detection algorithm does not perfectly detect circles.

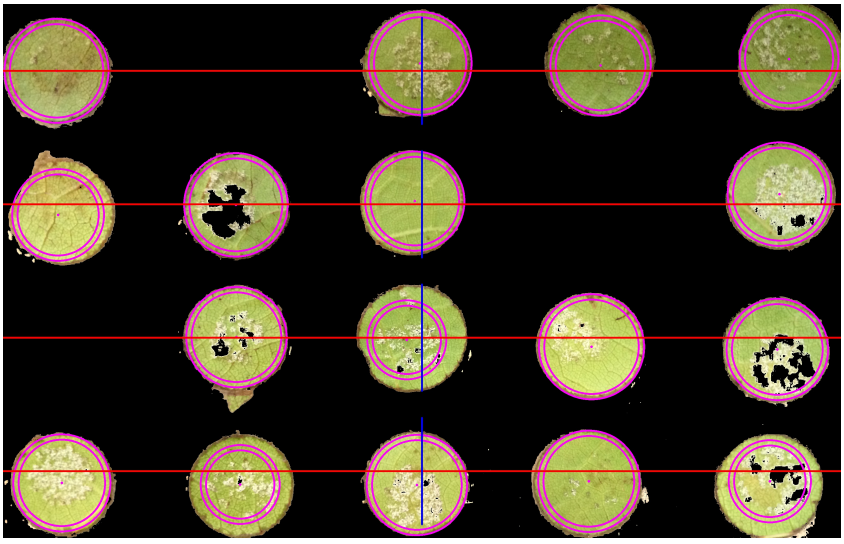

Figure 3: The cropped image after background filtering and circle detection. Detected circles are in purple. Before continuing, all leaf discs must have a purple circle around them and a purple circle center that must be within the blue vertical line in the row a circle is in. The smaller purple circle of the two is used to prevent circle edge pixels from being taken into account in downstream analysis. To use only the outer circle, specify **-sub 0** in the command.

- If any of the parameters in the script header were adjusted for **circles.py**, copy over the parameters in **circles.py** to **lines.py**.
- **python lines.py -s 1 -e 9**
    1. This script will detect leaf veins as lines in images 1.jpg through 9.jpg using the parameters in the script header and place the images with the detected lines in a folder called **lines**. The folder will also have images before detection of the lines and after removal of the lines.
    2. Manually inspect the images in the folder **lines** to ensure that most leaf veins have green lines around them. Due to the sensitivity of the line detection, it is not expected that all leaf veins will be identified. The white pixels within the purple circles are what the **values.py** script will use to calculate the phenotype.

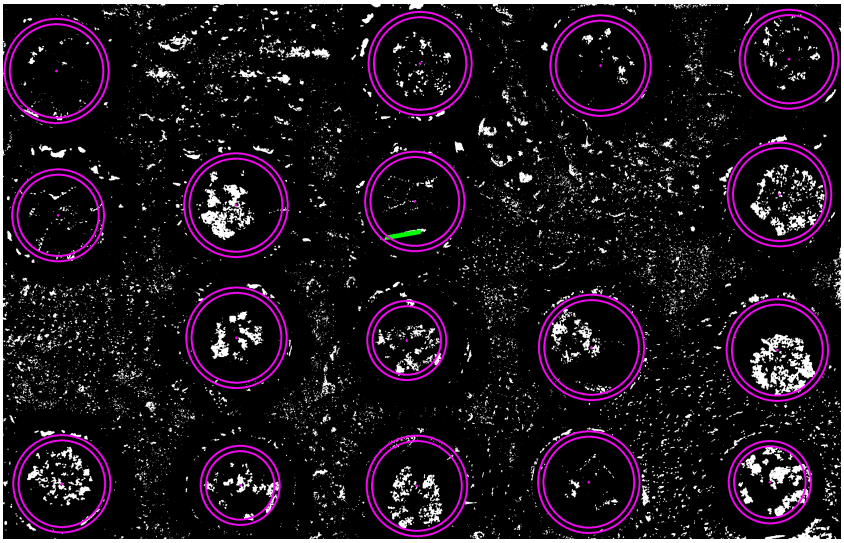

Figure 4: The cropped image after background filtering, Wallis filtering, and circle and line detection. Detected lines are in green.

- If any of the parameters in the script header were adjusted for either the **circles.py** or **lines.py** scripts, copy over the parameters from one or both of the scripts over to **values.py**.
- **python values.py -s 1 -e 9**
    1. This script will obtain the phenotype from each leaf disc in the nine images from left to right and top to bottom. The phenotype for a leaf disc is the number of white pixels within the inner purple circle corresponding to the leaf disc divided by the area of that circle.
    2. Make sure that the number of phenotypes in **values.csv** equals the number of leaf discs in the images. If it does not, an error was made along the way. Missing leaf discs will be skipped automatically.

Table 2: Contents of **values.csv** resulting from running **values.py** on the cropped image.

image_name	row	leaf_disc_number	value
1.jpg	1	1	0.021629482
1.jpg	1	2	0.052762717
1.jpg	1	3	0.031871589
1.jpg	1	4	0.053501783
1.jpg	2	1	0.032555673
1.jpg	2	2	0.166051868
1.jpg	2	3	0.021797458
1.jpg	2	4	0.157020681
1.jpg	3	1	0.10400559
1.jpg	3	2	0.143049053
1.jpg	3	3	0.091177916
1.jpg	3	4	0.226341459
1.jpg	4	1	0.131889259
1.jpg	4	2	0.120803886
1.jpg	4	3	0.131038923
1.jpg	4	4	0.022204593
1.jpg	4	5	0.276065931

4 General Overview of the Scripts

Here we will give a top-level view of what the scripts do and why you should use one at a particular time.

When taking a picture of 20 leaf discs, some leaf discs other than the desired 20 may appear in an image, which is undesirable as only the targeted 20 leaf discs are to be analyzed at this point. It is thus necessary to crop the images such that only 20 leaf discs are visible, though very small portions of undesirable leaf discs can be kept. When using **crop** to crop the images, the image background, agar in our case, is removed, and the leaf discs are detected as circles. The image background is removed by converting the image from BGR[2] color space to Lab color space and thresholding the L, a, and b layers of the image. Any detected circles that are not entirely in the image, e.g., half of a circle is in the image and half is out of the image, are removed, and the image is resized such that the new image border touches at least one circle's edge in each cardinal direction. A user-specified padding then allows for the new image border to be some distance away from the circles' edges (20 pixels in our case). This is done as the circle detection does not work flawlessly, and so this avoids the cropping of the desired leaf discs.

After obtaining the cropped images, the next step is to obtain sporulation values from each leaf disc. While in **crop** it was not necessary to detect each leaf disc for successful cropping, e.g., not detecting a leaf disc in the center of an image would not change the result of the cropping, it is necessary to do so now in order to obtain the phenotypes using **values**. A script called **circles** was made for the purpose of making sure all the leaf discs were detected before the quantification of sporulation. The output of the script is an image with three components: (i) detected leaf discs (drawn as circles) and their centers highlighted in purple[3], (ii) vertical blue bars of user-specified size for each row, (iii) horizontal red lines for each row going across the center of the rows. The goal of using **circles** is to make sure all the leaf discs have purple circles on top of them, and that all circle centers are within one of the vertical blue bars before running **values**, and this is done by adjusting the script's image processing parameters/options. The horizontal red lines only serve to show where the centers of rows are. The circles are detected by an algorithm called Hough circle transform that detects circles based on how likely a circle is present at a particular place, which is decided by white pixel density. The Hough circle transform in OpenCV performs Canny edge detection beforehand,

[2]OpenCV loads color images in BGR rather than RGB color space.

[3]There are two circles per detected leaf disc, and this is because it often happens that in the downstream image processing, edges of the leaf disc are included within the detected leaf disc, or purple circle, which might lead to increased error. We therefore decrease the detected circle radius by a user-specified parameter (20 pixels in our case). This new modified circle is drawn as a purple circle as well, leading to two circle per detected leaf disc.

so that the image from which circles are obtained is a binary image, i.e., the pixels are only black or white. The more white pixels, the greater the likelihood of finding a circle given a detection threshold; however, the likelihood of detecting false positives is also greater.

After all the leaf discs have been detected using **circles**, parameters used in **circles** are transferred to **values** and **lines**. **values** and **lines** detect the leaf discs the same way as **circles** does and additionally process the blue slice of the original unthresholded image (the B in BGR) through a Wallis filter. This filter divides the image into squares of user-defined size and forces the pixels in the squares, which range in value from 0 to 255, to have a user-defined mean and standard deviation, but in our case we will only change the standard deviation. This alleviates the problem of having a portion of the image brighter than another due to a light source. This is especially problematic in our case as we seek to extract the brightest pixels, as the sporulation is white, and so we want to detect the sporulation in both the bright and dark part of the image with as little bias as possible. After using the Wallis filter and extracting the brightest pixels, i.e., those with a value of 255, it sometimes happens that leaf veins will be included in the filtered image as they are of a brighter hue than the surrounding leaf disc. As leaf veins form straight lines, they can be detected with some precision using a Hough line transform, which is based on the same principles as the Hough circle transform. **lines** is a script that should be run before **values** to insure that leaf veins, if present, are being detected and removed[4]. **lines** outputs three images. The first is the Wallis-filtered image, the second is the same image but with detected circles in purple and detected lines in green, and the third is the same as the first, but with the detected lines rendered as black pixels. After choosing the best parameters for **lines**, the parameters are transferred to **values**, which performs circle detection and line removal and additionally isolates the area within each individual leaf disc (detected circle) to determine how many white pixels are inside and divides that value by the area of the detected circle to obtain the phenotype. In order to assign the phenotype of one leaf disc to the proper position of the output vector, we first need to know in what position a leaf disc is in. In order to arrange the detected circles, we say that a detected circle is in row r of the n user-defined rows if a detected circle center is within p pixels of $\frac{2r+1}{2n} * height$ in the y-axis. If one has an image and cuts it into 4 equal parts, the centers of each part are equal to $\frac{2r+1}{2n} * height$. As an example of leaf disc row assignment, if a 100 pixel (height) x 150 pixel (width) image has 4 rows of leaf discs, a detected circle center at (40,72) would be in the third row[5] if we give p a value of 15 as $\frac{2*2+1}{2*4} * 100 = 62.5$ and 72 is in the interval 62.5 ± 15. After the detected circles are ordered by row, the x-axis value of the detected circle centers is used to order them by column. Now we iterate through all the detected circles by row, i.e., we go from [1,1] $\rightarrow$ [1,C] to [2,1] $\rightarrow$ [2,C] and eventually to [R,1] $\rightarrow$ [R,C] where R and C are the number of leaf disc rows and the number of leaf discs within a row, respectively, and mask the entire image save the desired circle. The number of white pixels in the image, which corresponds to the sporulation quantity for one leaf disc, is then divided by the area of the detected circle corresponding to that leaf disc and placed in a vector. **values** outputs that vector, along with the image file name, row number, and leaf disc number, as a csv file. If more than one image is being analyzed, the vectors are appended to each other so that the n^{th} image's values begin after the $n-1^{th}$ image's values. The number of rows in the file will equal the number of detected circles across all the user-specified images desired to be analyzed. All further downstream data manipulation and analysis is left to the user.

5 Scripts Settings

All of the scripts are run in a command language interpreter (CLI) as **python [script name].py -[option] [option's value]**. E.g., if one wanted to run the **crop.py** script in a folder containing the script and images named 5 through 72 using default image processing parameters, one would move into the directory in the CLI and type **python crop.py -s 5 -e 72**. Most options available to scripts are shared among them, while some are unique to certain scripts. If options are not specified, the default values of the options will be used. One can find the default values of a script's options, along with the entire source code of a script by opening the script in programs such as Notepad, Notepad++, or PyCharm, among others. However, the

[4]We do not find that it is necessary to run **lines** on all the images as one setting works well for most images encountered. This is because the threshold of line detection cannot be lowered beyond a point since dense sporulation, which is a large mass of white pixels in the Wallis-filtered image, will then be detected to have lines in them.

[5]In defining r, the third row would have a value of 2 as we use zero-based numbering, which is used in Python.

option information can also be accessed by typing **python [script name].py -help** in the CLI when in the same directory as the script. Below is the output when this is done for crop.py:

```
usage: crop.py [-h] [-ext EXT] [-s S] [-e E] [-buff BUFF] [-l L] [-a A] [-b B]
               [-can1 CAN1] [-can2 CAN2] [-minrad MINRAD] [-maxrad MAXRAD]
               [-cdist CDIST] [-circlethresh CIRCLETHRESH]

High-Throughput Downy Mildew Phenotyping - Cropping images

optional arguments:
  -h, --help            show this help message and exit
  -ext EXT              Image file extension. Default: jpg
  -s S                  First image name. Default: 1
  -e E                  Last image name. If only one image is to be analyzed,
                        set -e to what -s is. Default: 1
  -buff BUFF            Value in pixels that will be added to the sides of the
                        cropped image. This is added as the Hough circle
                        transform is does not perfectly detect leaf discs.
                        Default: 20
  -l L                  Threshold value for L in Lab color space. Default: 255
  -a A                  Threshold value for a in Lab color space. This
                        threshold is inversed. Default: 130
  -b B                  Threshold value for b in Lab color space. Default: 150
  -can1 CAN1            Lower hysteresis parameter in pixels for the Canny
                        filter. Values below this threshold are not kept.
                        Default: 5
  -can2 CAN2            Upper hysteresis parameter in pixels for the Canny
                        filter. Values above this threshold are kept. Values
                        in between this parameter and the lower parameter only
                        are kept if they are connected by a pixel above this
                        parameter. Default: 70
  -minrad MINRAD        Minimum circle radius parameter in pixels for the
                        Hough circle transform. Default: 150
  -maxrad MAXRAD        Maximum circle radius parameter in pixels for the
                        Hough circle transform. Default: 210
  -cdist CDIST          Minimum distance between the centers of the detected
                        circles. Default: 400
  -circlethresh CIRCLETHRESH
                        The accumulator threshold for the circle centers for
                        the Hough circle transform. Smaller values will mean
                        more circles will be detected. Default: 30
```

Options are always on the top of the source code for a script after the package importation commands and are generally straightforward. Below are the options for the script **crop.py**:

```
parser = argparse.ArgumentParser(description='High-Throughput Downy Mildew Phenotyping - Cropping
    images')
parser.add_argument('-ext', type=str, default='jpg',
                    help='Image file extension. Default: jpg')
parser.add_argument('-s', type=int, default=1,
                    help='First image name. Default: 1')
parser.add_argument('-e', type=int, default=1,
                    help='Last image name. If only one image is to be analyzed, set -e to what -s
                        is. Default: 1')
parser.add_argument('-buff', type=int, default=20,
```

```
                    help='Value in pixels that will be added to the sides of the cropped image. This
                        is added as the Hough circle transform is does not perfectly detect leaf
                        discs. Default: 20')
parser.add_argument('-l', type=int, default=255,
                    help='Threshold value for L in Lab color space. Default: 255')
parser.add_argument('-a', type=int, default=130,
                    help='Threshold value for a in Lab color space. This threshold is inversed.
                        Default: 130')
parser.add_argument('-b', type=int, default=150,
                    help='Threshold value for b in Lab color space. Default: 150')
parser.add_argument('-can1', type=int, default=5,
                    help='Lower hysteresis parameter in pixels for the Canny filter. Values below
                        this threshold are not kept. Default: 5')
parser.add_argument('-can2', type=int, default=70,
                    help='Upper hysteresis parameter in pixels for the Canny filter. Values above
                        this threshold are kept. Values in between this parameter and the lower
                        parameter only are kept if they are connected by a pixel above this
                        parameter. Default: 70')
parser.add_argument('-minrad', type=int, default=150,
                    help='Minimum circle radius parameter in pixels for the Hough circle transform.
                        Default: 150')
parser.add_argument('-maxrad', type=int, default=210,
                    help='Maximum circle radius parameter in pixels for the Hough circle transform.
                        Default: 210')
parser.add_argument('-cdist', type=int, default=400,
                    help='Minimum distance between the centers of the detected circles. Default:
                        400')
parser.add_argument('-circlethresh', type=int, default=30,
                    help='The accumulator threshold for the circle centers for the Hough circle
                        transform. Smaller values will mean more circles will be detected. Default:
                        30')
args = parser.parse_args()
```

Each option is referred to as an argument. The first value is the option's CLI call name (e.g., **-dist**), the second is the value type of the option, in this case they are all integers with the exception of the image file extension, which is a string, the third is the default value of the option, and the fourth is a description of the option. If additional help is required, the argument can be looked up in the source code of the script by searching for **args.option**, where **option** is the argument without the minus sign, e.g., the argument for **-dist** would be **args.dist**. The source code is well documented with comments above most functions. If additional help is needed, see the OpenCV documentation (http://docs.opencv.org/).

We have found it more useful to edit the scripts' options directly by changing the default values rather than use the **-[option]** in the CLI. Editing directly allows one to always have the latest settings saved on file for later use. It also allows for efficient transfer of settings by copying and pasting. For example, suppose you want to copy the settings from **circles** to **values**; just copy and paste the options from the former to the latter rather than writing the settings down and changing the defaults using the CLI.

6 Miscellaneous

In order to better organize your current directory when new images are created, all the scripts with the exception of **values** make new folders within which the processed images reside. This is especially important when using all the scripts after **crop** because if they are not in the folder **crop**, but rather in the original directory, you will be running scripts on the original images rather than the cropped images.

When starting to crop images without prior knowledge of what settings might be useful over the defaults, we suggest using the **circles** script on the images to gain experience with how circles are being detected. This will greatly reduce the need to manually crop images, which might be needed sometimes.

When working with leaf discs with trichomes, we found that it is not a good use of time to find the

one set of parameters that will work best for all the images (72 in our case). Often, two or three images have issues with the parameter settings, and so we use **values** with one set of parameters to analyze one set of images, and another set of parameters to analyze another set of images. This problem stems from the inability to threshold out the white color too strongly lest whole leaf discs that have a lot of trichomes get removed. For glabrous leaf discs, one parameter setting that works for all the images can be found relatively easily.

To speed up the scripts, all of them automatically use all the cores on a CPU asynchronously. This asynchronization is what causes the script progress printout in the **circles** and **values** scripts to not be in order. In the code, the parallelization is defined by

```
pool = ThreadPool(multiprocessing.cpu_count())
```

If one wants to use only one core, one can change the line to

```
pool = ThreadPool(1)
```

in every script.